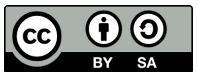
Introduction to CMake

Houmin Wei

Electronics Engineering & Computer Science
Peking University

January 6, 2018



Outline

Part I: CMake

CMake tool sets

CMake

CMake is a *cross-platform build systems generator* which makes it easier to build software in a unified manner on a broad set of platforms:

CMake has friends softwares that may be used on their own or together:

- CMake: build system generator
- CPack: package generator
- CTest: systematic test driver
- CDash: a dashboard collector

Outline of Part I: CMake

Outline of Part II: CPack

Part I: CMake Part II: CPack

Outline of Part III: CTest and CDash

Part I: CMake Part II: CPack Part III: CTest and CDash

Build what?

Software build system

A software build system is the usage of a [set of] tool[s] for building software applications.

Why do we need that?

Build what?

Software build system

A software build system is the usage of a [set of] tool[s] for building software applications.

Why do we need that?

- because most softwares consist of several parts that need some *building* to put them together,

Build what?

Software build system

A software build system is the usage of a [set of] tool[s] for building software applications.

Why do we need that?

- because most softwares consist of several parts that need some *building* to put them together,
- because softwares are written in *various languages* that may share the same building process,

Build what?

Software build system

A software build system is the usage of a [set of] tool[s] for building software applications.

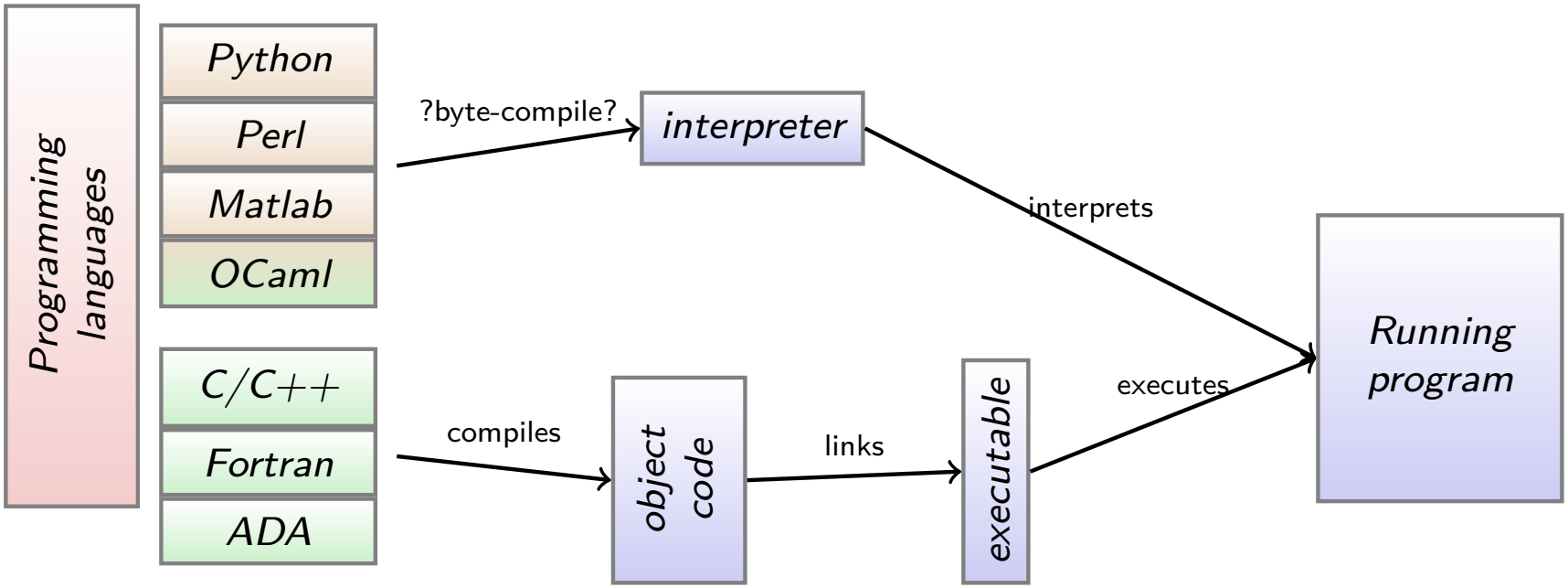
Why do we need that?

- because most softwares consist of several parts that need some *building* to put them together,
- because softwares are written in *various languages* that may share the same building process,
- because we want to build *the same software for various computers* (PC, Macintosh, Workstation, mobile phones and other PDA, embedded computers) and systems (Windows, Linux, *BSD, other Unices (many), Android, etc...)

Programming languages

Compiled vs interpreted or what?

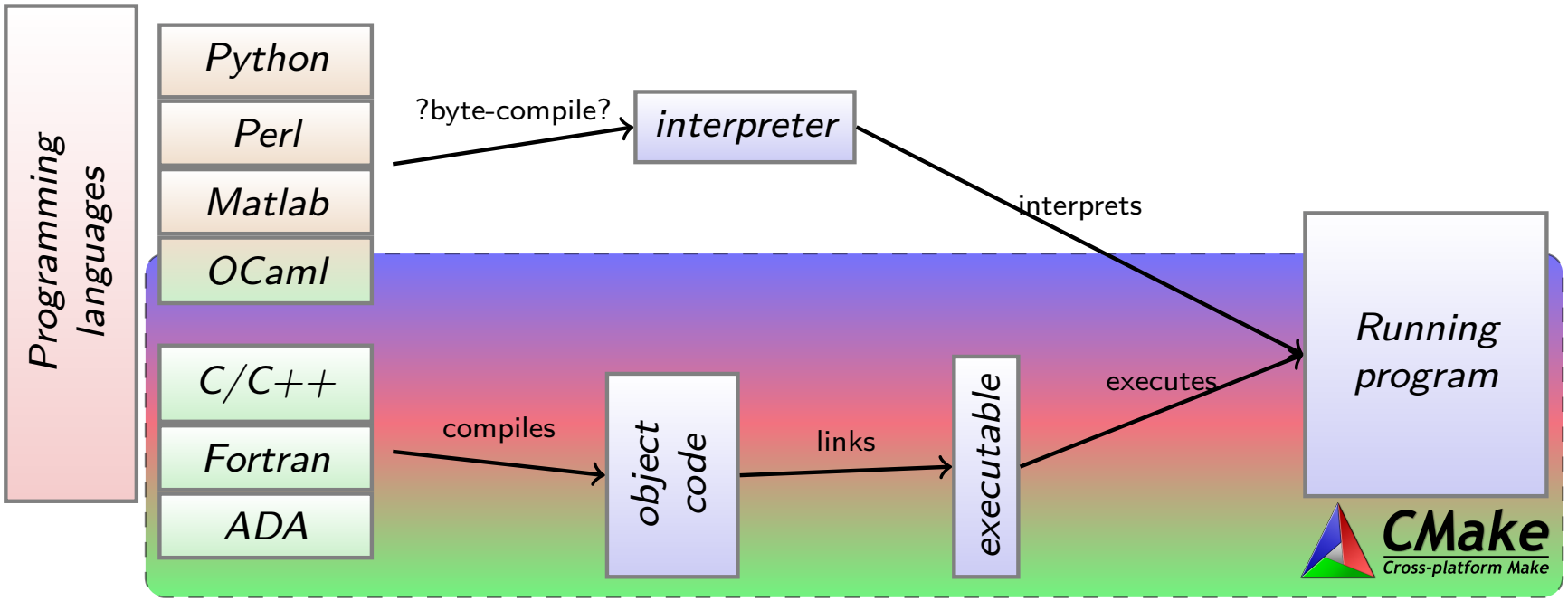
Building an application requires the use of some programming *language*: Python, Java, C++, Fortran, C, Go, Tcl/Tk, Ruby, Perl, OCaml,...



Programming languages

Compiled vs interpreted or what?

Building an application requires the use of some programming *language*: Python, Java, C++, Fortran, C, Go, Tcl/Tk, Ruby, Perl, OCaml,...



Build systems: several choices

Alternatives

CMake is not the only build system [generator]:

- (portable) hand-written Makefiles, depends on `make` tool (may be GNU Make).
- Apache ant (or Maven or Gradle), dedicated to Java (almost).
- Portable IDE: Eclipse, Code::Blocks, Geany, NetBeans, ...
- GNU Autotools: Autoconf, Automake, Libtool. Produce makefiles. Bourne shell needed (and M4 macro processor).
- SCons only depends on Python.
- ...

Build systems or build systems generator

Build systems

A tool which builds, a.k.a. compiles, a set of source files in order to produce binary executables and libraries. Those kind of tools usually takes as input a file (e.g. a Makefile) and while reading it issues compile commands. The main goal of a build tool is to (re)build the minimal subset of files when something changes. A non exhaustive list: [GNU] make, ninja, MSBuild, SCons, ant, ...

A **Build systems generator** is a tool which generates files for a particular build system. e.g. CMake or Autotools.

What build systems do?

Targets and sources

The main feature of a build system is to offer a way to describe how a target (executable, PDF, shared library...) is built from its sources (set of object files and/or libraries, a latex or rst file, set of C/C++/Fortran files...). Basically a *target* **depends** on one or several sources and one can run a set of **commands** in order to built the concerned *target* from its *sources*.

The main goals/features may be summarized as:

- describe dependency graph between sources and targets
- associate one or several commands to *rebuilt* target from source(s)
- issue the *minimal* set of commands in order to rebuild a target

A sample `Makefile` for `make`

```
1  CC=gcc
2  CFLAGS=-Wall -Werror -pedantic -std=c99
3  LDFLAGS=
4  EXECUTABLES=Acrodictlibre Acrolibre
5
6  # default rule (the first one)
7  all : $(EXECUTABLES)
8  # explicit link target
9  Acrolibre: acrolibre.o
10   $(CC)  $(CFLAGS) -o $@ $^
11 # explicit link and compile target
12 Acrodictlibre: acrolibre.c acrodict.o
13   $(CC)  $(CFLAGS) -DUSE_ACRODICT -o $@ $^
14
15 # Implicit rule using file extension
16 # Every .o file depends on corresponding .c (and may be .h) file
17 %.o : %.c %.h
18   $(CC) $(CFLAGS) -c $<
19 %.o : %.c
20   $(CC) $(CFLAGS) -c $<
21 clean:
22   @\rm -f *.o $(EXECUTABLES)
```

Comparisons and [success] stories

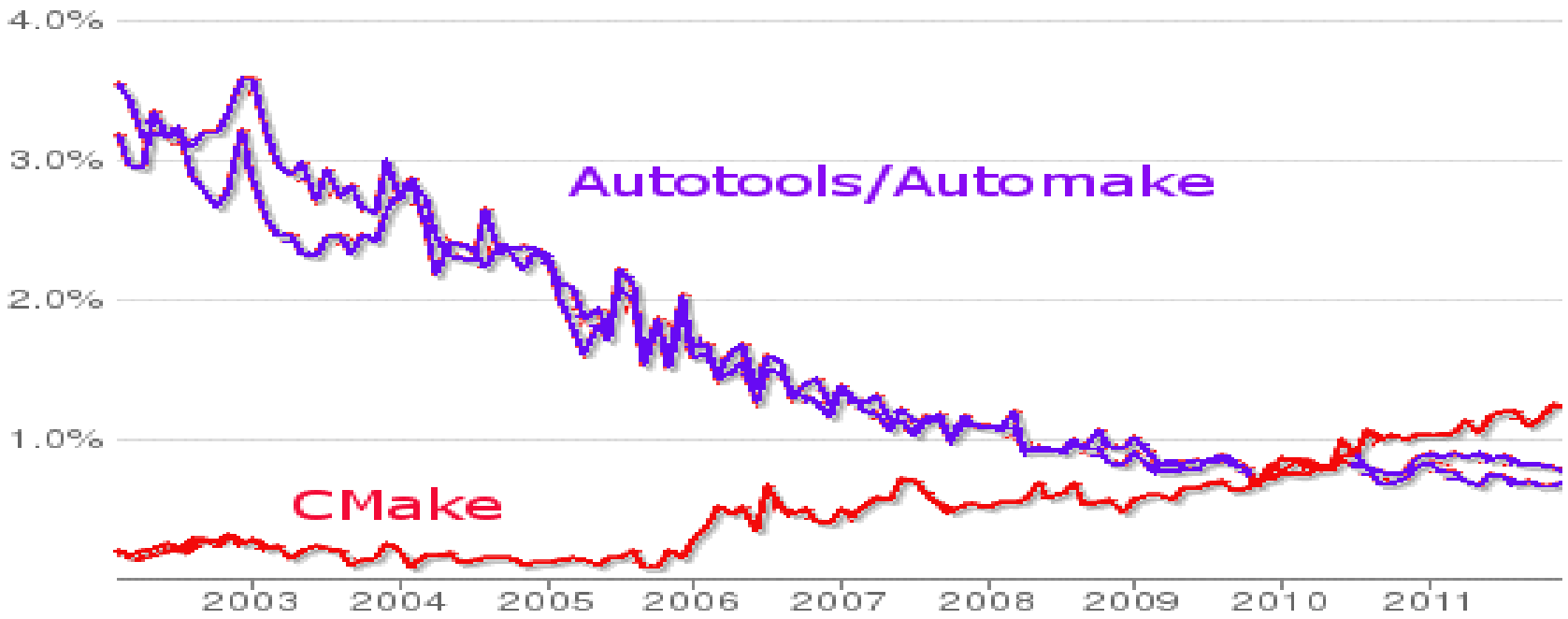
Disclaimer

This presentation is biased. *I mean totally*.
I am a big CMake fan, I did contribute to CMake, thus I'm not impartial *at all*. But I will be ready to discuss why CMake is the greatest build system out there :-)

Go and forge your own opinion:

- Bare list: http://en.wikipedia.org/wiki/List_of_build_automation_software
- A comparison: http://www.scons.org/wiki/SconsVsOtherBuildTools
- KDE success story (2006): "*Why the KDE project switched to CMake – and how*" http://lwn.net/Articles/188693/

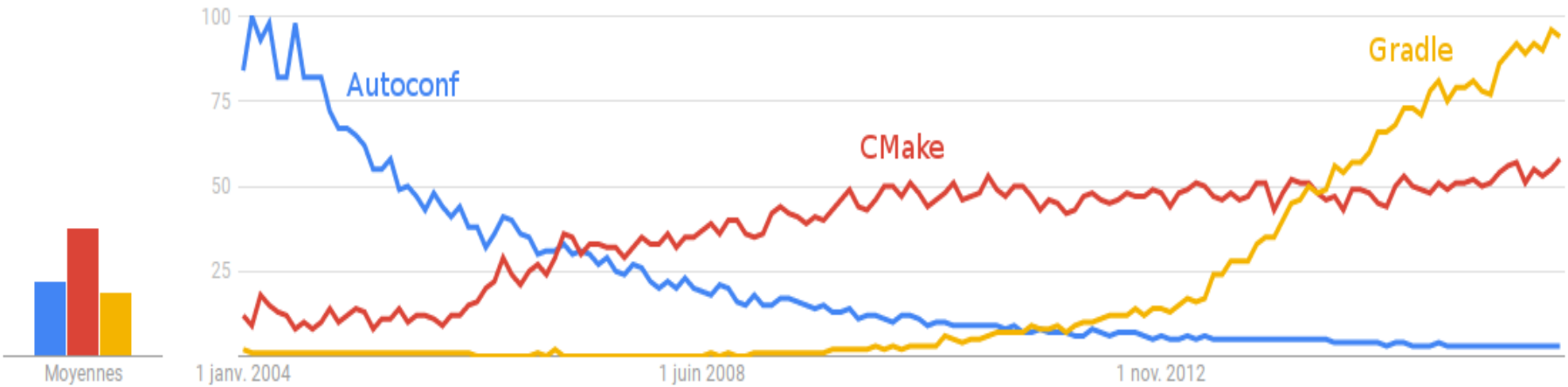
CMake/Auto[conf|make] on OpenHub



https://www.openhub.net/languages/compare

Language comparison of CMake to automake and autoconf showing the percentage of developers commits that modify a source file of the respective language (data from 2012).

CMake/Autoconf/Gradle on Google Trend



https://www.google.com/trends

Scale is based on the average worldwide request traffic searching for CMake, Autoconf and Gradle in all years (2004–now).

Outline

Part I: CMake

A build system generator

- CMake is a *generator*: it generates *native* build systems files (Makefile, Ninja, IDE project files [XCode, CodeBlocks, Eclipse CDT, Codelite, Visual Studio, Sublime Text...], ...),
- CMake scripting language (declarative) is used to describe the build,
- The developer edits `CMakeLists.txt`, invokes CMake but should *never* edit the generated files,
- CMake may be (automatically) re-invoked by the build system,
- CMake has friends who may be very handy (CPack, CTest, CDash)

The CMake workflow

When do things take place?

CMake is a *generator* which means it does not compile (i.e. build) the sources, the underlying build tool (make, Ninja, XCode, Visual Studio...) does.

The CMake workflow

1. *CMake time*: CMake is running & processing `CMakeLists.txt`

When do things take place?

CMake is a *generator* which means it does not compile (i.e. build) the sources, the underlying build tool (make, Ninja, XCode, Visual Studio...) does.

The CMake workflow

1. *CMake time*: CMake is running & processing `CMakeLists.txt`
2. *Build time*: the build tool runs and invokes (at least) the compiler

When do things take place?

CMake is a *generator* which means it does not compile (i.e. build) the sources, the underlying build tool (make, Ninja, XCode, Visual Studio...) does.

The CMake workflow

1. *CMake time*: CMake is running & processing `CMakeLists.txt`
2. *Build time*: the build tool runs and invokes (at least) the compiler
3. *Install time*: the compiled binaries are installed
   i.e. from build area to an install location.

When do things take place?

CMake is a *generator* which means it does not compile (i.e. build) the sources, the underlying build tool (make, Ninja, XCode, Visual Studio...) does.

The CMake workflow

1. *CMake time*: CMake is running & processing `CMakeLists.txt`
2. *Build time*: the build tool runs and invokes (at least) the compiler
3. *Install time*: the compiled binaries are installed
   i.e. from build area to an install location.
4. *CPack time*: CPack is running for building package

When do things take place?

CMake is a *generator* which means it does not compile (i.e. build) the sources, the underlying build tool (make, Ninja, XCode, Visual Studio...) does.

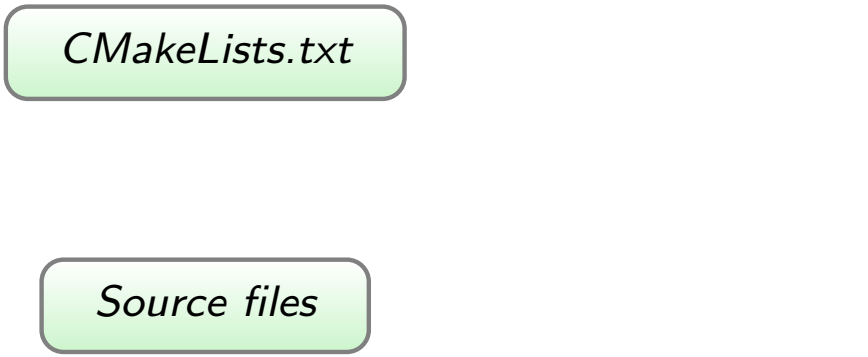
The CMake workflow

1. *CMake time*: CMake is running & processing `CMakeLists.txt`
2. *Build time*: the build tool runs and invokes (at least) the compiler
3. *Install time*: the compiled binaries are installed
   i.e. from build area to an install location.
4. *CPack time*: CPack is running for building package
5. *Package Install time*: the package (from previous step) is installed

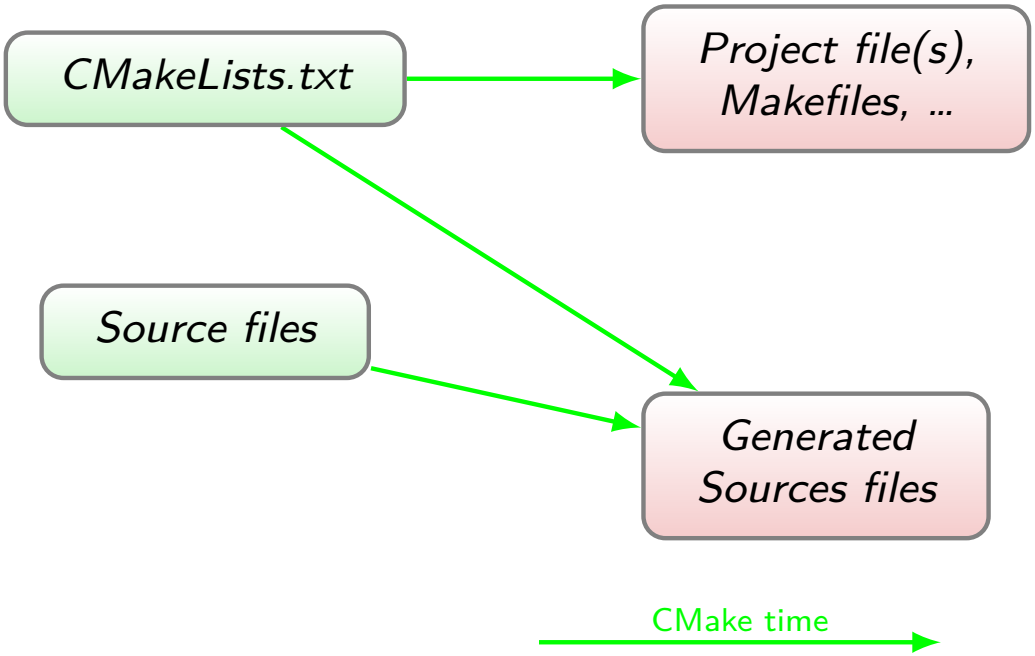
When do things take place?

CMake is a *generator* which means it does not compile (i.e. build) the sources, the underlying build tool (make, Ninja, XCode, Visual Studio...) does.

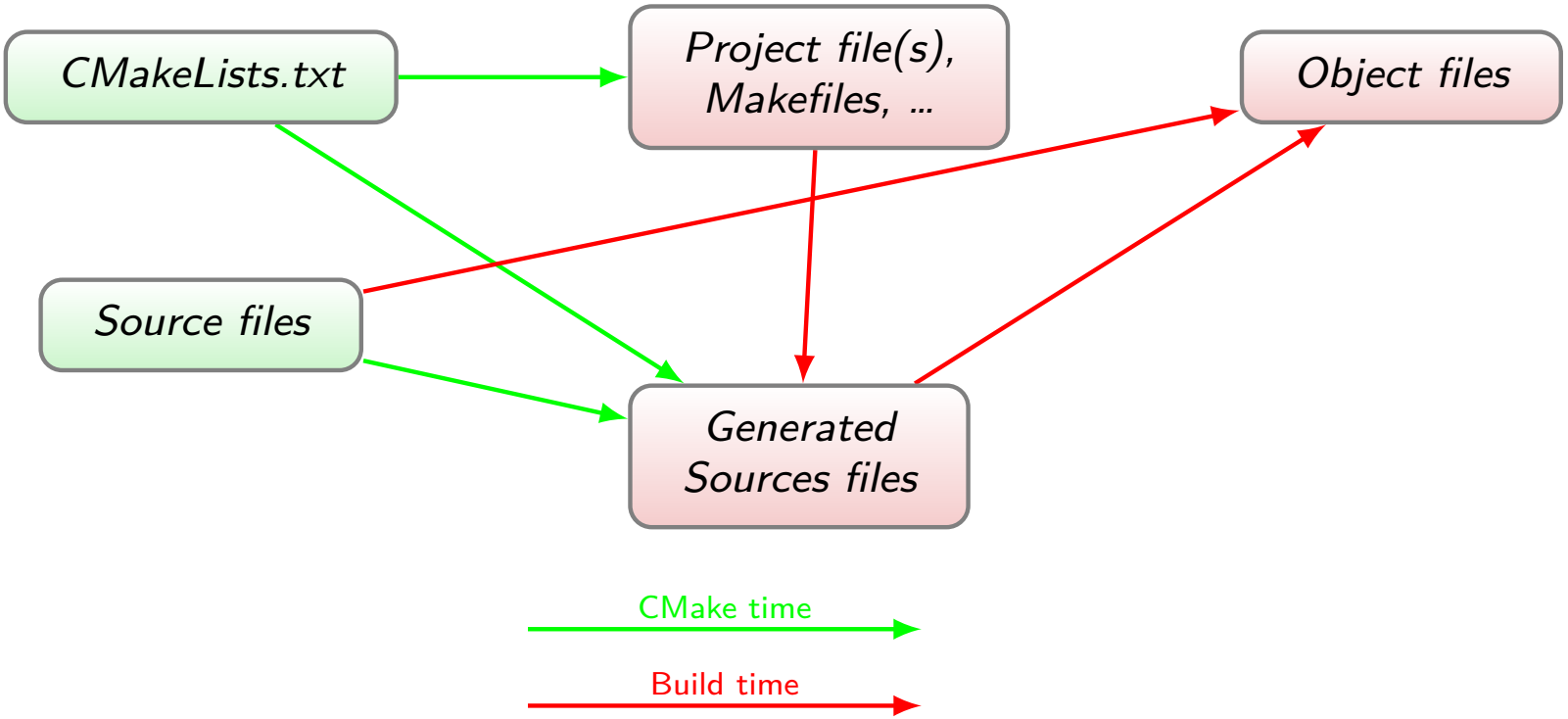
The CMake workflow (pictured)



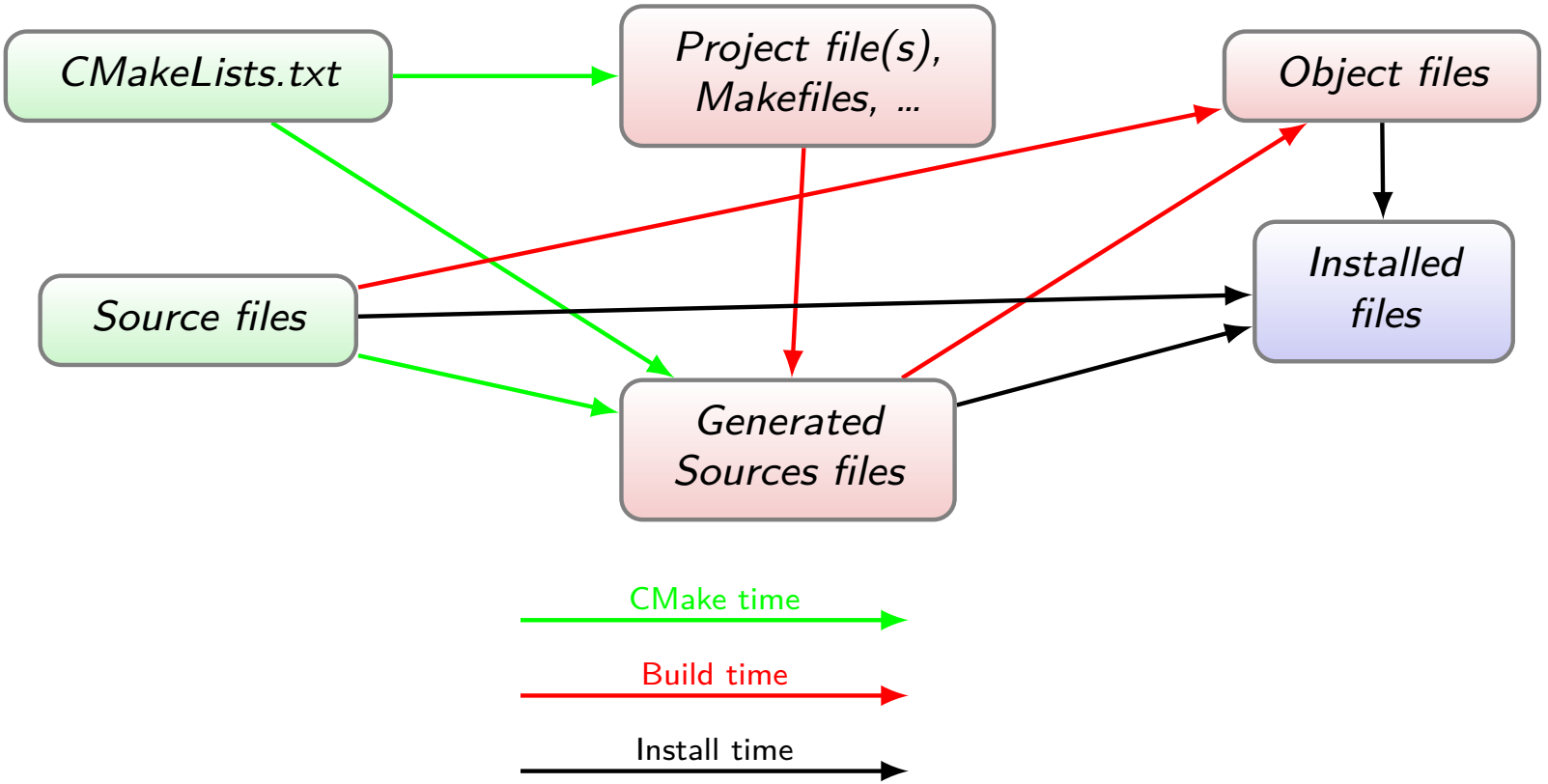
The CMake workflow (pictured)



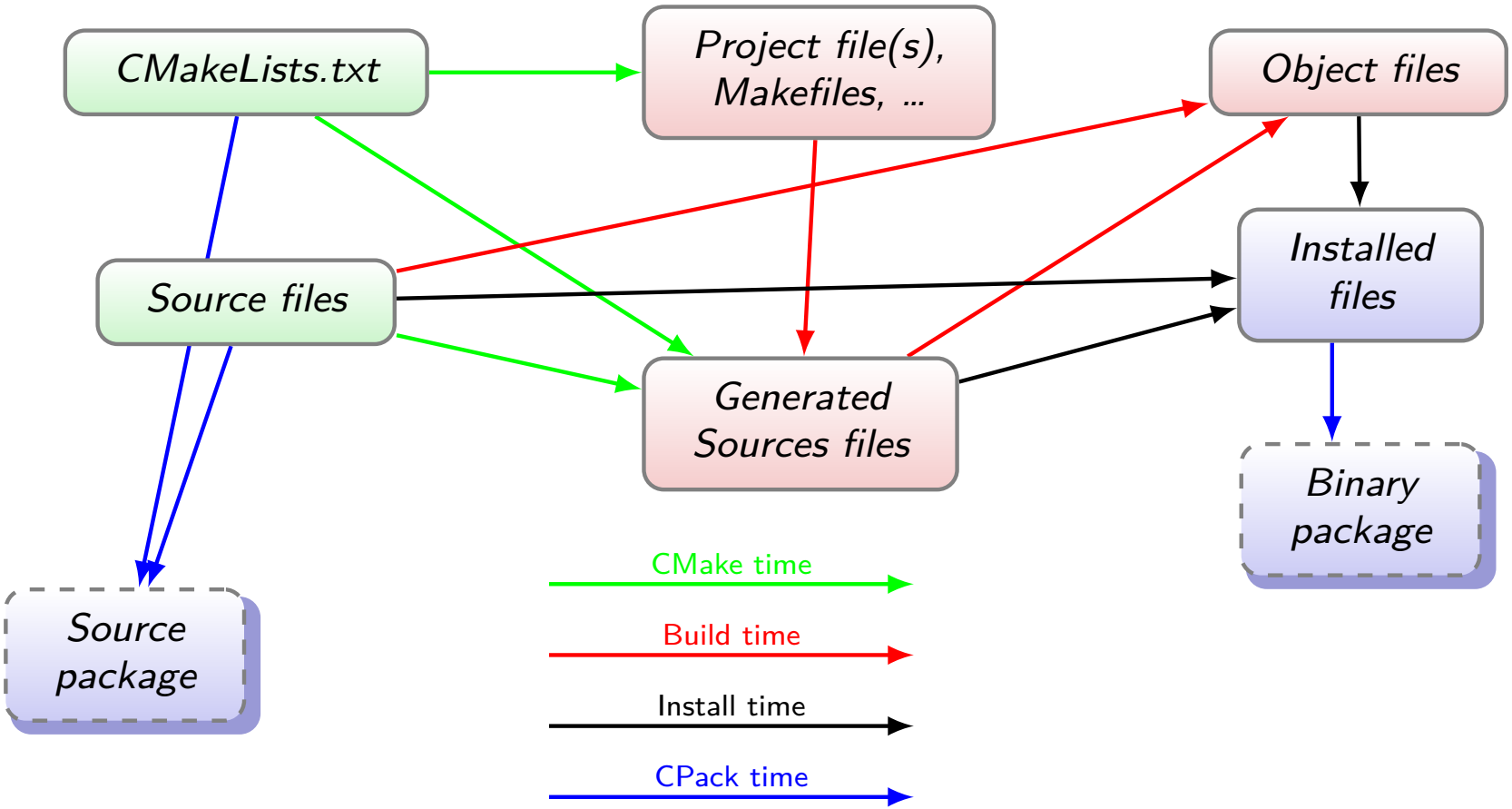
The CMake workflow (pictured)



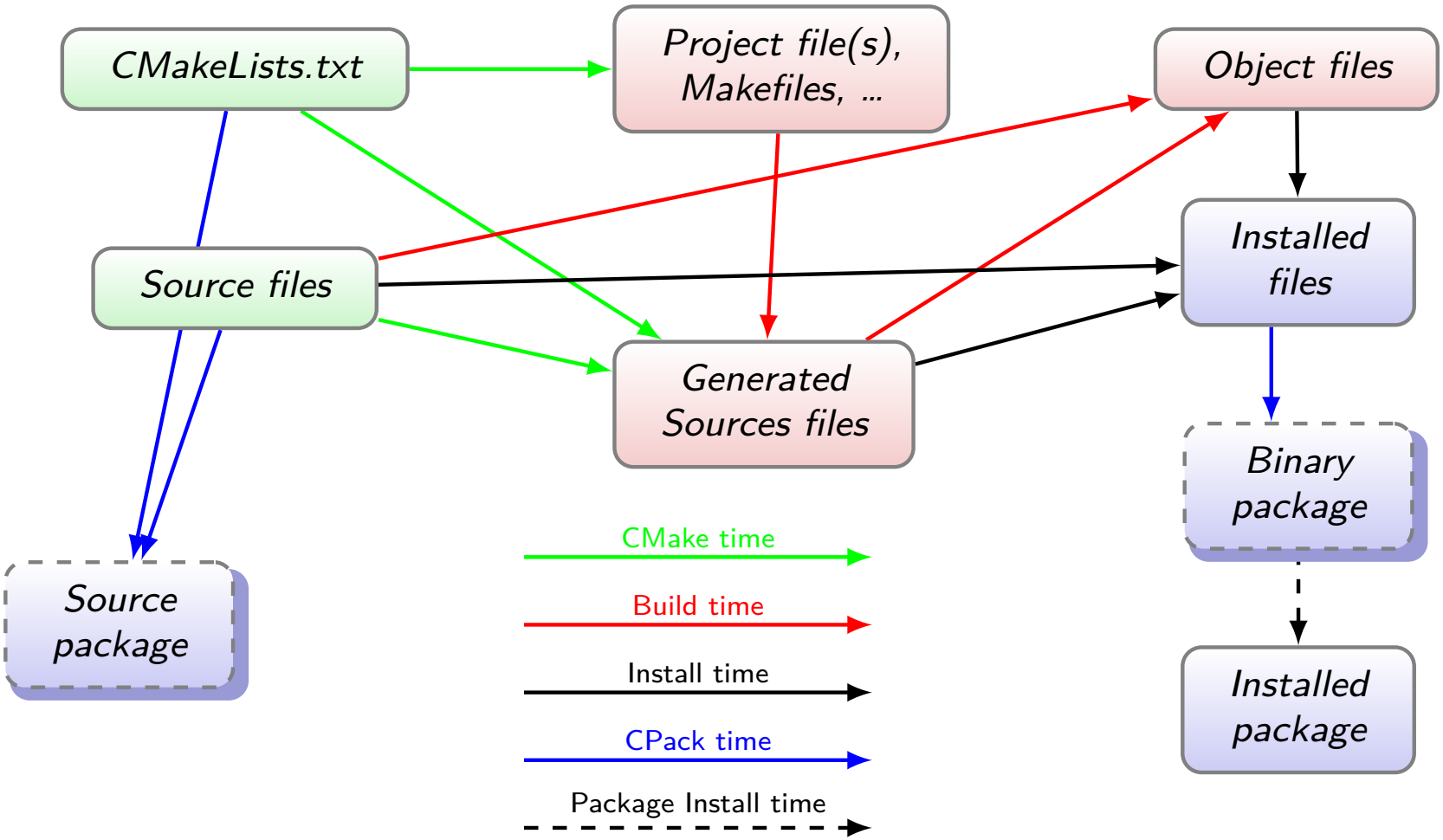
The CMake workflow (pictured)



The CMake workflow (pictured)



The CMake workflow (pictured)



Building an executable

```
1 cmake_minimum_required (VERSION 3.0)
2 # This project use C source code
3 project (TotallyFree C)
4 set(CMAKE_C_STANDARD 99)
5 set(CMAKE_C_EXTENSIONS False)
6 # build executable using specified list of source files
7 add_executable(Acrolibre acrolibre.c)
```

Listing 1: Building a simple program

CMake scripting language is [mostly] declarative. It has *commands* which are documented from within CMake:

```
 $ cmake --help-command-list | wc -l
 117
 $ cmake --help-command add_executable
...
  add_executable
       Add an executable to the project using the specified source files.
```

Builtin documentation

CMake builtin doc for 'project' command

```
  $ cmake --help-command project
project
-------

Set a name, version, and enable languages for the entire project.

 project(<PROJECT-NAME> [LANGUAGES] [<language-name>...])
 project(<PROJECT-NAME>
         [VERSION <major>[.<minor>[.<patch>[.<tweak>]]]]
         [LANGUAGES <language-name>...])

Sets the name of the project and stores the name in the
``PROJECT_NAME`` variable.
[...]
       Optionally you can specify which languages your project supports.
       Example languages are CXX (i.e.  C++), C, Fortran, etc.  By default C and CXX are enabled.
       E.g.  if you do  not have a C++ compiler, you can disable the check  for it by  explicitly
       listing the languages  you want  to support, e.g.  C. By using the special language "NONE"
       all checks for any language can be disabled.
```

Online doc : https://cmake.org/documentation/

Unix Manual: `cmake-variables(7)`, `cmake-commands(7)`, `cmake-xxx(7)`, ...

All doc generated using Sphinx

Generating & building

Building with CMake and make is easy:

Building with make

```
$ ls totally-free
acrolibre.c  CMakeLists.txt
$ mkdir build
$ cd build
$ cmake ../totally-free
-- The C compiler identification is GNU 4.6.2
-- Check for working C compiler: /usr/bin/gcc
-- Check for working C compiler: /usr/bin/gcc -- works
...
$ make
...
 [100%] Built target Acrolibre
$ ./Acrolibre toulibre
```

Source tree vs Build tree

Even the most simple project should never mix-up sources with generated files. CMake supports *out-of-source* build.

Generating & building

Building with CMake and ninja is easy:

Building with ninja

```
$ ls totally-free
acrolibre.c  CMakeLists.txt
$ mkdir build-ninja
$ cd build-ninja
$ cmake -GNinja ../totally-free
-- The C compiler identification is GNU 4.6.2
-- Check for working C compiler: /usr/bin/gcc
-- Check for working C compiler: /usr/bin/gcc -- works
...
$ ninja
...
[6/6] Linking C executable Acrodictlibre
$ ./Acrolibre toulibre
```

Source tree vs Build tree

Even the most simple project should never mix-up sources with generated files. CMake supports *out-of-source* build.

Generating & building

Cross-Building with CMake and make is easy:

Building with cross-compiler

```
$ ls totally-free
acrolibre.c  CMakeLists.txt
$ mkdir build-win32
$ cd build-win32
$ cmake -DCMAKE_TOOLCHAIN_FILE=../totally-free/Toolchain-cross-linux.cmake ../totally-free
-- The C compiler identification is GNU 6.1.1
-- Check for working C compiler: /usr/bin/i686-w64-mingw32-gcc
...
$ make
...
[100%] Linking C executable Acrolibre.exe
[100%] Built target Acrolibre
$ ./Acrolibre toulibre
```

Source tree vs Build tree

Even the most simple project should never mix-up sources with generated files. CMake supports *out-of-source* build.

Always build out-of-source

Out-of-source is better

People are lazy (me too) and they think that because building in source is possible and authorizes less typing they can get away with it. In-source build is a *BAD* choice.

Out-of-source build is *always* better because:

Always build out-of-source

Out-of-source is better

People are lazy (me too) and they think that because building in source is possible and authorizes less typing they can get away with it. In-source build is a *BAD* choice.

Out-of-source build is *always* better because:

1. Generated files are separated from manually edited ones (thus you don't have to clutter your favorite VCS ignore files).

Always build out-of-source

Out-of-source is better

People are lazy (me too) and they think that because building in source is possible and authorizes less typing they can get away with it. In-source build is a *BAD* choice.

Out-of-source build is *always* better because:

1. Generated files are separated from manually edited ones (thus you don't have to clutter your favorite VCS ignore files).
2. You can have several build trees for the same source tree

Always build out-of-source

Out-of-source is better

People are lazy (me too) and they think that because building in source is possible and authorizes less typing they can get away with it. In-source build is a *BAD* choice.

Out-of-source build is *always* better because:

1. Generated files are separated from manually edited ones (thus you don't have to clutter your favorite VCS ignore files).
2. You can have several build trees for the same source tree
3. This way it's always safe to completely delete the build tree in order to do a clean build

Building program + autonomous library

We now have the following set of files in our source tree:

- `acrolibre.c`, the main C program
- `acrodict.h`, the Acrodict library header
- `acrodict.c`, the Acrodict library source
- `CMakeLists.txt`, the soon to be updated CMake input file

Building program + autonomous library

Conditional build

We want to keep a version of our program that can be compiled and run without the new Acrodict library *and* the new version which uses the library.

We now have the following set of files in our source tree:

- `acrolibre.c`, the main C program
- `acrodict.h`, the Acrodict library header
- `acrodict.c`, the Acrodict library source
- `CMakeLists.txt`, the soon to be updated CMake input file

The main program source

```c
#include <stdlib.h>
#include <stdio.h>
#include <strings.h>
#ifdef USE_ACRODICT
#include "acrodict.h"
#endif
int main(int argc, char* argv[]) {

  const char * name;
#ifdef USE_ACRODICT
  const acroItem_t*  item;
#endif

  if (argc < 2) {
    fprintf(stderr,"%s: you need one argument\n",argv[0]);
    fprintf(stderr,"%s <name>\n",argv[0]);
    exit(EXIT_FAILURE);
  }
  name = argv[1];

#ifndef USE_ACRODICT
  if (strcasecmp(name,"toulibre")==0) {
    printf("Toulibre is a french organization promoting FLOSS.\n");
  }
#else
  item = acrodict_get(name);
  if (NULL!=item) {
    printf("%s: %s\n",item->name,item->description);
  } else if (item=acrodict_get_approx(name)) {
    printf("<%s> is unknown may be you mean :\n",name);
    printf("%s: %s\n",item->name,item->description);
  }
#endif
  else {
    printf("Sorry, I don't know: <%s>\n", name);
    return EXIT_FAILURE;
  }
  return EXIT_SUCCESS;
}
```

The library source

```
1 #ifndef ACRODICT_H
2 #define ACRODICT_H
3 typedef struct acroItem {
4   char* name;
5   char* description;
6 } acroItem_t;
7
8 const acroItem_t*
9 acrodict_get(const char* name);
10 #endif
```

```
1 #include <stdlib.h>
2 #include <string.h>
3 #include "acrodict.h"
4 static const acroItem_t acrodict[] = {
5   {"Toulibre", "Toulibre is a french
      organization promoting FLOSS"},
6   {"GNU", "GNU is Not Unix"},
7   {"GPL", "GNU general Public License"},
8   {"BSD", "Berkeley Software Distribution"},
9   {"CULTe","Club des Utilisateurs de
      Logiciels libres et de gnu/linux de
      Toulouse et des environs"},
10  {"Lea", "Lea-Linux: Linux entre ami(e)s"},
11  {"RMLL","Rencontres Mondiales du Logiciel
      Libre"},
12  {"FLOSS","Free Libre Open Source Software"
      },
13  {"",""}};
14 const acroItem_t*
15 acrodict_get(const char* name) {
16   int current =0;
17   int found   =0;
18   while ((strlen(acrodict[current].name)>0)
      && !found) {
19     if (strcasecmp(name,acrodict[current].
      name)==0) {
20       found=1;
21     } else {
22       current++;
23     }
24   }
25   if (found) {
26     return &(acrodict[current]);
27   } else {
28     return NULL;
29   }
30 }
```

A sample `Makefile` for `make`

```
1  CC=gcc
2  CFLAGS=-Wall -Werror -pedantic -std=c99
3  LDFLAGS=
4  EXECUTABLES=Acrodictlibre Acrolibre
5
6  # default rule (the first one)
7  all : $(EXECUTABLES)
8  # explicit link target
9  Acrolibre: acrolibre.o
10   $(CC)  $(CFLAGS) -o $@ $^
11 # explicit link and compile target
12 Acrodictlibre: acrolibre.c acrodict.o
13   $(CC)  $(CFLAGS) -DUSE_ACRODICT -o $@ $^
14
15 # Implicit rule using file extension
16 # Every .o file depends on corresponding .c (and may be .h) file
17 %.o : %.c %.h
18   $(CC) $(CFLAGS) -c $<
19 %.o : %.c
20   $(CC) $(CFLAGS) -c $<
21 clean:
22   @\rm -f *.o $(EXECUTABLES)
```

Building a library

```
1 cmake_minimum_required (VERSION 3.0)
2 project (TotallyFree C)
3 set(CMAKE_C_STANDARD 99)
4 set(CMAKE_C_EXTENSIONS False)
5 add_executable(Acrolibre acrolibre.c)
6 set(LIBSRC acrodict.c acrodict.h)
7 add_library(acrodict ${LIBSRC})
8 add_executable(Acrodictlibre acrolibre.c)
9 target_link_libraries(Acrodictlibre acrodict)
10 set_target_properties(Acrodictlibre PROPERTIES COMPILE_FLAGS "-DUSE_ACRODICT")
```

Listing 2: Building a simple program + shared library

- we precise that we want to compile with C99 flags

Building a library

```
1 cmake_minimum_required (VERSION 3.0)
2 project (TotallyFree C)
3 set(CMAKE_C_STANDARD 99)
4 set(CMAKE_C_EXTENSIONS False)
5 add_executable(Acrolibre acrolibre.c)
6 set(LIBSRC acrodict.c acrodict.h)
7 add_library(acrodict ${LIBSRC})
8 add_executable(Acrodictlibre acrolibre.c)
9 target_link_libraries(Acrodictlibre acrodict)
10 set_target_properties(Acrodictlibre PROPERTIES COMPILE_FLAGS "-DUSE_ACRODICT")
```

Listing 3: Building a simple program + shared library

- we precise that we want to compile with C99 flags
- we define a variable and ask to build a library

Building a library

```
1 cmake_minimum_required (VERSION 3.0)
2 project (TotallyFree C)
3 set(CMAKE_C_STANDARD 99)
4 set(CMAKE_C_EXTENSIONS False)
5 add_executable(Acrolibre acrolibre.c)
6 set(LIBSRC acrodict.c acrodict.h)
7 add_library(acrodict ${LIBSRC})
8 add_executable(Acrodictlibre acrolibre.c)
9 target_link_libraries(Acrodictlibre acrodict)
10 set_target_properties(Acrodictlibre PROPERTIES COMPILE_FLAGS "-DUSE_ACRODICT")
```

Listing 4: Building a simple program + shared library

- we precise that we want to compile with C99 flags
- we define a variable and ask to build a library
- we link an executable to our library

Building a library

```
1 cmake_minimum_required (VERSION 3.0)
2 project (TotallyFree C)
3 set(CMAKE_C_STANDARD 99)
4 set(CMAKE_C_EXTENSIONS False)
5 add_executable(Acrolibre acrolibre.c)
6 set(LIBSRC acrodict.c acrodict.h)
7 add_library(acrodict ${LIBSRC})
8 add_executable(Acrodictlibre acrolibre.c)
9 target_link_libraries(Acrodictlibre acrodict)
10 set_target_properties(Acrodictlibre PROPERTIES COMPILE_FLAGS "-DUSE_ACRODICT")
```

Listing 5: Building a simple program + shared library

- we precise that we want to compile with C99 flags
- we define a variable and ask to build a library
- we link an executable to our library
- we compile the source files of a particular target with specific compiler options

Building a library - continued I

And it builds...

All in all CMake generates appropriate Unix makefiles which build all this smoothly.

CMake + Unix Makefile

```
$ make
[ 33%] Building C object CMakeFiles/acrodict.dir/acrodict.c.o
Linking C shared library libacrodict.so
[ 33%] Built target acrodict
[ 66%] Building C object CMakeFiles/Acrodictlibre.dir/acrolibre.c.o
Linking C executable Acrodictlibre
[ 66%] Built target Acrodictlibre
[100%] Building C object CMakeFiles/Acrolibre.dir/acrolibre.c.o
Linking C executable Acrolibre
[100%] Built target Acrolibre
$ ls -F
Acrodictlibre*  CMakeCache.txt  cmake_install.cmake  Makefile
Acrolibre*      CMakeFiles/     libacrodict.so*
```

Building a library - continued II

And it works...

We get the two different variants of our program, with varying capabilities.

```
$ ./Acrolibre toulibre
Toulibre is a french organization promoting FLOSS.
$ ./Acrolibre FLOSS
Sorry, I don't know: <FLOSS>
$ ./Acrodictlibre FLOSS
FLOSS: Free Libre Open Source Software
```

```
$ make help
The following are some of the valid targets
    for this Makefile:
... all (the default if no target is provided)
... clean
... depend
... Acrodictlibre
... Acrolibre
... acrodict
...
```

Generated `Makefiles` has several builtin targets besides the expected ones:

- one per target (library or executable)
- clean, all
- more to come ...

Building a library - continued III

And it is homogeneously done whatever the generator...

The obtained build system contains the same set of targets whatever the combination of generator and [cross-]compiler used: Makefile+gcc, Ninja+clang, XCode, Visual Studio, etc...

User controlled build option

User controlled option

Maybe our users don't want the acronym dictionary support. We can use CMake `OPTION` command.

```
1  cmake_minimum_required (VERSION 3.0)
2  # This project use C source code
3  project (TotallyFree C)
4  # Build option with default value to ON
5  option(WITH_ACRODICT "Include acronym dictionary support" ON)
6  set(BUILD_SHARED_LIBS true)
7  # build executable using specified list of source files
8  add_executable(Acrolibre acrolibre.c)
9  if (WITH_ACRODICT)
10    set(LIBSRC acrodict.h acrodict.c)
11    add_library(acrodict ${LIBSRC})
12    add_executable(Acrodictlibre acrolibre.c)
13    target_link_libraries(Acrodictlibre acrodict)
14    set_target_properties(Acrodictlibre PROPERTIES COMPILE_FLAGS "-DUSE_ACRODICT")
15 endif(WITH_ACRODICT)
```

Listing 6: User controlled build option

Too much keyboard, time to click? I

CMake comes with severals tools

A matter of choice / taste:

- a command line: `cmake`
- a curses-based TUI: `ccmake`
- a Qt-based GUI: `cmake-gui`

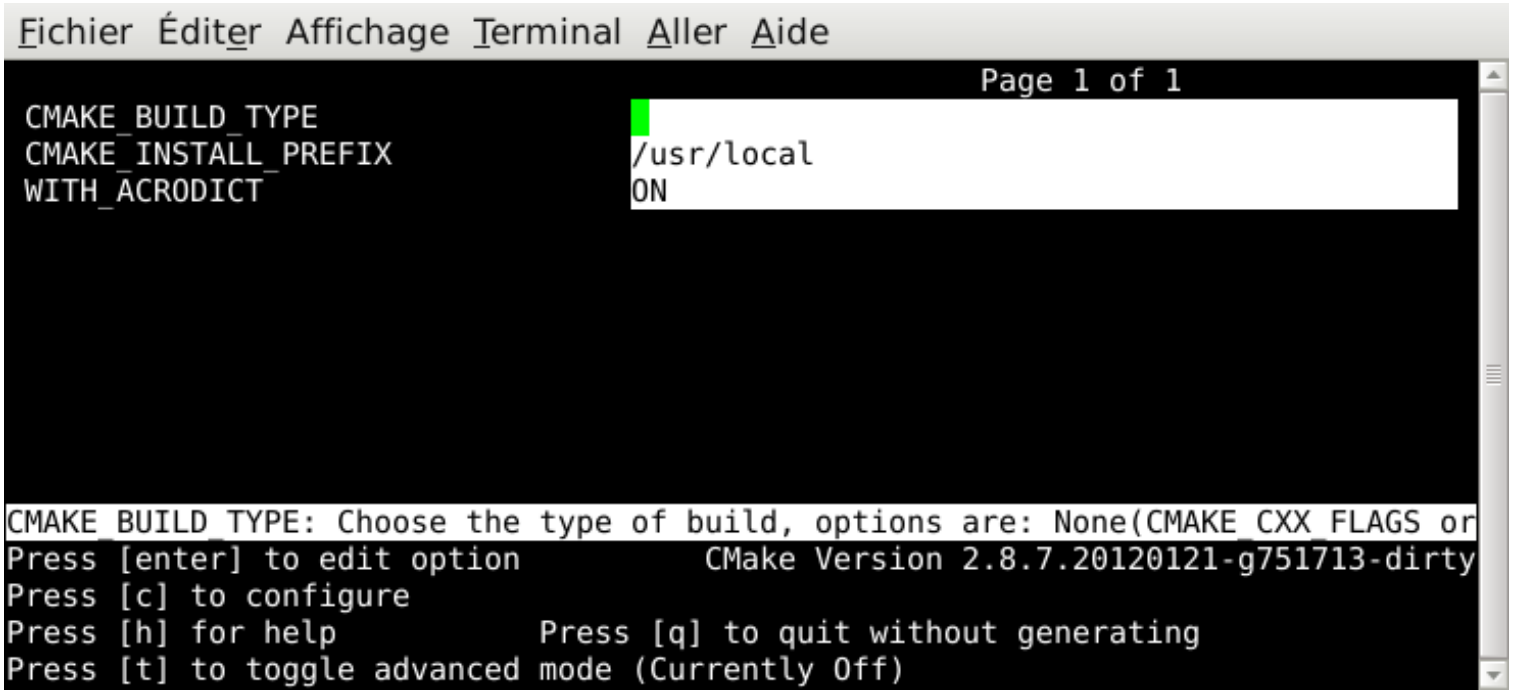
Calling convention

All tools expect to be called with a single argument which may be interpreted in 2 different ways.

- path to the source tree, e.g.: `cmake /path/to/source`
- path to an existing build tree, e.g.: `cmake-gui .`

Too much keyboard, time to click? II

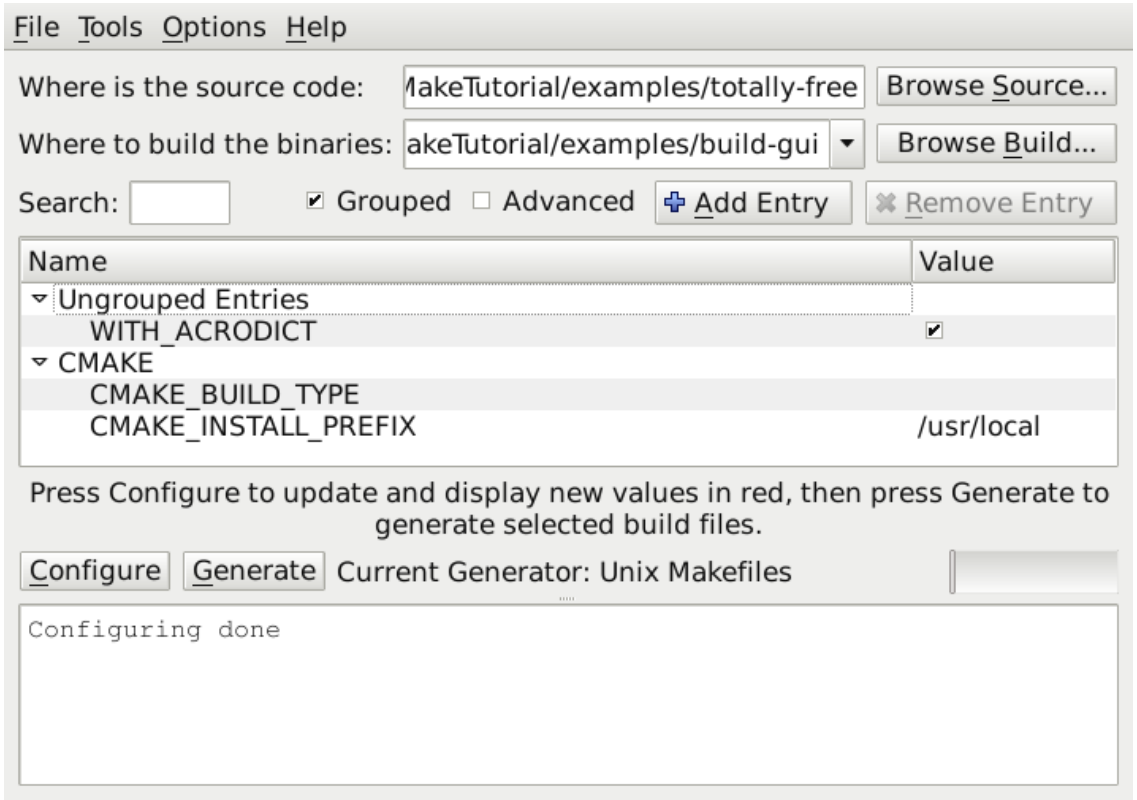
`ccmake` : the curses-based TUI (demo)



Here we can choose to toggle the `WITH_ACRODICT OPTION`.

Too much keyboard, time to click? III

`cmake-gui` : the Qt-based GUI (demo)



Again, we can choose to toggle the `WITH_ACRODICT OPTION`.

Remember CMake is a build generator?

The number of active generators depends on the platform we are running on Unix, Apple, Windows:

```
Borland Makefiles
MSYS Makefiles
MinGW Makefiles
NMake Makefiles
NMake Makefiles JOM
Unix Makefiles
Visual Studio 10
Visual Studio 10 IA64
Visual Studio 10 Win64
Visual Studio 11
Visual Studio 11 Win64
Visual Studio 6
Visual Studio 7
Visual Studio 7 .NET 2003
Visual Studio 8 2005
Visual Studio 8 2005 Win64
Visual Studio 9 2008
Visual Studio 9 2008 IA64
Visual Studio 9 2008 Win64
Watcom WMake
CodeBlocks - MinGW Makefiles
CodeBlocks - NMake Makefiles
CodeBlocks - Unix Makefiles
Eclipse CDT4 - MinGW Makefiles
Eclipse CDT4 - NMake Makefiles
Eclipse CDT4 - Unix Makefiles
KDevelop3
KDevelop3 - Unix Makefiles
XCode
Ninja
```

Equally simple on other platforms

It is as easy for a Windows build, however names for executables and libraries are computed in a platform specific way.

CMake + MinGW Makefile

```
$ ls totally-free
acrodict.h acrodict.c acrolibre.c  CMakeLists.txt
$ mkdir build-win32
$ cd build-win32
$ cmake -DCMAKE_TOOLCHAIN_FILE=../totally-free/Toolchain-cross-linux.cmake ../totally-free
...
$ make
Scanning dependencies of target acrodict
[ 33%] Building C object CMakeFiles/acrodict.dir/acrodict.c.obj
Linking C shared library libacrodict.dll
Creating library file: libacrodict.dll.a
[ 33%] Built target acrodict
Scanning dependencies of target Acrodictlibre
[ 66%] Building C object CMakeFiles/Acrodictlibre.dir/acrolibre.c.obj
Linking C executable Acrodictlibre.exe
[ 66%] Built target Acrodictlibre
Scanning dependencies of target Acrolibre
[100%] Building C object CMakeFiles/Acrolibre.dir/acrolibre.c.obj
Linking C executable Acrolibre.exe
[100%] Built target Acrolibre
```

Installing things

Install

Several parts or the software may need to be installed: this is controlled by the CMake `install` command.

Remember `cmake --help-command install`!!

```
1 ...
2 add_executable(Acrolibre acrolibre.c)
3 install(TARGETS Acrolibre DESTINATION bin)
4 if (WITH_ACRODICT)
5   ...
6   install(TARGETS Acrodictlibre acrodict
7             RUNTIME DESTINATION bin
8             LIBRARY DESTINATION lib
9             ARCHIVE DESTINATION lib/static)
10  install(FILES acrodict.h DESTINATION include)
11 endif(WITH_ACRODICT)
```

Listing 7: install command examples

Controlling installation destination

Use relative `DESTINATION`

One should always use relative installation `DESTINATION` unless you really want to use absolute path like `/etc`.

Then depending on when you install:

Controlling installation destination

Use relative DESTINATION

One should always use relative installation DESTINATION unless you really want to use absolute path like /etc.

Then depending on when you install:

- At CMake-time set CMAKE_INSTALL_PREFIX value

```
$ cmake --help-variable CMAKE_INSTALL_PREFIX
```

Controlling installation destination

Use relative DESTINATION

One should always use relative installation DESTINATION unless you really want to use absolute path like /etc.

Then depending on when you install:

- At CMake-time set CMAKE_INSTALL_PREFIX value

```
$ cmake --help-variable CMAKE_INSTALL_PREFIX
```

- At Install-time use DESTDIR mechanism (Unix Makefiles)

```
$ make DESTDIR=/tmp/testinstall install
```

Controlling installation destination

Use relative DESTINATION

One should always use relative installation DESTINATION unless you really want to use absolute path like /etc.

Then depending on when you install:

- At CMake-time set CMAKE_INSTALL_PREFIX value

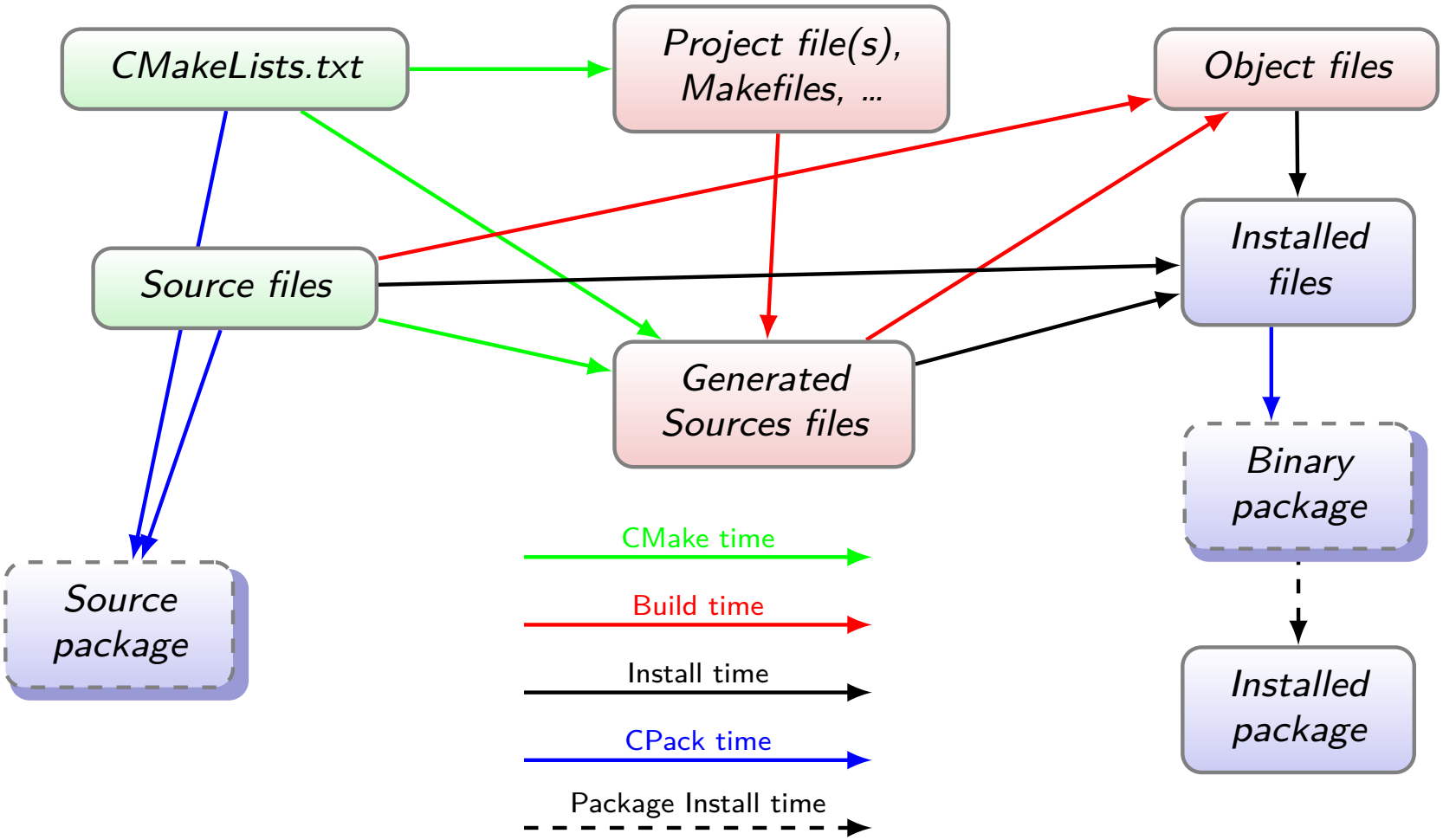
```
$ cmake --help-variable CMAKE_INSTALL_PREFIX
```

- At Install-time use DESTDIR mechanism (Unix Makefiles)

```
$ make DESTDIR=/tmp/testinstall install
```

- At CPack-time, CPack what? ...be patient.
- At Package-install-time, we will see that later

The CMake workflow (pictured)



Using CMake variables

CMake variables

They are used by the user to simplify its `CMakeLists.txt`, but CMake uses many (~400+) of them to control/change its [default] behavior. Try: `cmake --help-variable-list`.

Inside a CMake script

```
set(CMAKE_INSTALL_PREFIX /home/eric/testinstall)
```

```
$ cmake --help-command set
```

On the command line/TUI/GUI

Remember that (besides options) each CMake tool takes a single argument (source tree or existing build tree)

```
$ cmake -DCMAKE_INSTALL_PREFIX=/home/eric/testinstall .
```

The install target

Install target

The `install` target of the underlying build tool (in our case `make`) appears in the generated build system as soon as some `install` commands are used in the `CMakeLists.txt`.

```
$ make DESTDIR=/tmp/testinstall install
[ 33%] Built target acrodict
[ 66%] Built target Acrodictlibre
[100%] Built target Acrolibre
Install the project...
-- Install configuration: ""
-- Installing: /tmp/testinstall/bin/Acrolibre
-- Installing: /tmp/testinstall/bin/Acrodictlibre
-- Removed runtime path from "/tmp/testinstall/bin/Acrodictlibre"
-- Installing: /tmp/testinstall/lib/libacrodict.so
-- Installing: /tmp/testinstall/include/acrodict.h
$
```

Package the whole thing

CPack

CPack is a CMake friend application (detailed later) which may be used to easily package your software.

```
1  ...
2  endif(WITH_ACRODICT)
3  ...
4  # Near the end of the CMakeLists.txt
5  # Chose your CPack generator
6  set(CPACK_GENERATOR "TGZ")
7  # Setup package version
8  set(CPACK_PACKAGE_VERSION_MAJOR 0)
9  set(CPACK_PACKAGE_VERSION_MINOR 1)
10 set(CPACK_PACKAGE_VERSION_PATCH 0)
11 # 'call' CPack
12 include(CPack)
```

Listing 8: add CPack support

```
$ make package
[ 33%] Built target acrodict
[ 66%] Built target Acrodictlibre
[100%] Built target Acrolibre
Run CPack packaging tool...
CPack: Create package using TGZ
CPack: Install projects
CPack: - Run preinstall target for: TotallyFree
CPack: - Install project: TotallyFree
CPack: Create package
CPack: - package: <build-tree>/...
      TotallyFree-0.1.0-Linux.tar.gz generated.
$ tar ztvf TotallyFree-0.1.0-Linux.tar.gz
... TotallyFree-0.1.0-Linux/include/acrodict.h
... TotallyFree-0.1.0-Linux/bin/Acrolibre
... TotallyFree-0.1.0-Linux/bin/Acrodictlibre
... TotallyFree-0.1.0-Linux/lib/libacrodict.so
```

CPack the packaging friend

CPack is a standalone generator

As we will see later on, CPack is standalone application, which like CMake is a *generator*.

```
$ cpack -G ZIP
CPack: Create package using ZIP
CPack: Install projects
CPack: - Run preinstall target for: TotallyFree
CPack: - Install project: TotallyFree
CPack: Create package
CPack: - package: <build-tree>/...
      TotallyFree-0.1.0-Linux.zip generated.
$ unzip -t TotallyFree-0.1.0-Linux.zip
Archive:  TotallyFree-0.1.0-Linux.zip
    testing: To.../include/acrodict.h   OK
    testing: To.../bin/Acrolibre   OK
    testing: To.../bin/Acrodictlibre   OK
    testing: To.../lib/libacrodict.so   OK
No errors detected in compressed
   data of TotallyFree-0.1.0-Linux.zip.
```

```
$ cpack -G RPM
CPack: Create package using RPM
CPack: Install projects
CPack: - Run preinstall target for: TotallyFree
CPack: - Install project: TotallyFree
CPack: Create package
CPackRPM: Will use GENERATED spec file: <build-tree>/...
      _CPack_Packages/Linux/RPM/SPECS/totallyfree.spec
CPack: - package: <build-tree>/...
      TotallyFree-0.1.0-Linux.rpm generated.
$ rpm -qpl TotallyFree-0.1.0-Linux.rpm
/usr
/usr/bin
/usr/bin/Acrodictlibre
/usr/bin/Acrolibre
/usr/include
/usr/include/acrodict.h
/usr/lib
/usr/lib/libacrodict.so
```

Didn't you mentioned testing? I

CTest

CTest is a CMake friend application (detailed later) which may be used to easily test your software (if not cross-compiled though).

```
1  ...
2  endif(WITH_ACRODICT)
3  ...
4  enable_testing()
5  add_test(toulibre-builtin
6           Acrolibre "toulibre")
7  add_test(toulibre-dict
8           Acrodictlibre "toulibre")
9  add_test(FLOSS-dict
10          Acrodictlibre "FLOSS")
11 add_test(FLOSS-fail
12          Acrolibre "FLOSS")
```

Listing 9: add CTest support

```
$ make test
Running tests...
Test project <buildtree-prefix>/build
    Start 1: toulibre-builtin
1/4 Test #1: toulibre-builtin ....   Passed  0.00 sec
    Start 2: toulibre-dict
2/4 Test #2: toulibre-dict........   Passed  0.00 sec
    Start 3: FLOSS-dict
3/4 Test #3: FLOSS-dict ..........   Passed  0.00 sec
    Start 4: FLOSS-fail
4/4 Test #4: FLOSS-fail ..........***Failed  0.00 sec
75% tests passed, 1 tests failed out of 4
Total Test time (real) =   0.01 sec
The following tests FAILED:
    4 - FLOSS-fail (Failed)
```

Didn't you mentioned testing? II

Tailor success rule

CTest uses the return code in order to get success/failure status, but one can tailor the success/fail rule.

Didn't you mentioned testing? III

```
1  ...
2  endif(WITH_ACRODICT)
3  ...
4  enable_testing()
5  add_test(toulibre-builtin
6           Acrolibre "toulibre")
7  add_test(toulibre-dict
8           Acrodictlibre "toulibre")
9  add_test(FLOSS-dict
10          Acrodictlibre "FLOSS")
11 add_test(FLOSS-fail
12          Acrolibre "FLOSS")
13 set_tests_properties (FLOSS-fail
14          PROPERTIES
15          PASS_REGULAR_EXPRESSION
16          "Sorry, I don't know:.*FLOSS")
```

Listing 10: add CTest support

```
$ make test
Running tests...
Test project <buildtree-prefix>/build
    Start 1: toulibre-builtin
1/4 Test #1: toulibre-builtin ....   Passed  0.00 sec
    Start 2: toulibre-dict
2/4 Test #2: toulibre-dict........   Passed  0.00 sec
    Start 3: FLOSS-dict
3/4 Test #3: FLOSS-dict ..........   Passed  0.00 sec
    Start 4: FLOSS-fail
4/4 Test #4: FLOSS-fail ..........   Passed  0.00 sec

100% tests passed, 0 tests failed out of 4

Total Test time (real) =   0.01 sec
```

CTest the testing friend

CTest is a standalone generic test driver

CTest is standalone application, which can run a set of *test* programs.

```
$ ctest -R toulibre-
Test project <build-tree>/build
    Start 1: toulibre-builtin
1/2 Test #1: toulibre-builtin .. Passed 0.00 sec
    Start 2: toulibre-dict
2/2 Test #2: toulibre-dict ..... Passed 0.00 sec

100% tests passed, 0 tests failed out of 2

Total Test time (real) =   0.01 sec
```

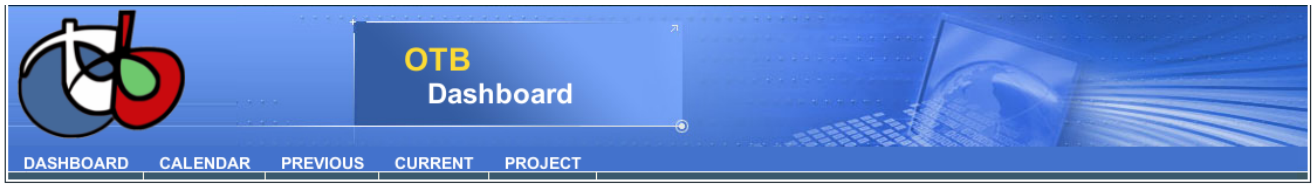
```
$ ctest -R FLOSS-fail -V
Test project <build-tree>
Constructing a list of tests
Done constructing a list of tests
Checking test dependency graph...
Checking test dependency graph end
test 4
    Start 4: FLOSS-fail
4: Test command: <build-tree>/Acrolibre "FLOSS"
4: Test timeout computed to be: 9.99988e+06
4: Sorry, I don't know: <FLOSS>
1/1 Test #4: FLOSS-fail ...........***Failed 0.00 sec

0% tests passed, 1 tests failed out of 1
Total Test time (real) =   0.00 sec

The following tests FAILED:
^^I  4 - FLOSS-fail (Failed)
Errors while running CTest
```

CDash the test results publishing

Dashboard

CTest may help publishing the results of the tests on a CDash dashboard (http://www.cdash.org/) for easing collective regression testing. More on this later...

http://www.orfeo-toolbox.org/–http://dash.orfeo-toolbox.org/



No update data as of **Friday, February 03 2012 19:00:00 CET**

Help

[Show Filters]

Nightly | Continuous | Experimental | Nightly Applications | Continuous Applications | Experimental Applications | Coverage | Dynamic Analysis

Site	Build Name	Update		Configure			Build			Test				Build Time	Labels
		Files	Min	Error	Warn	Min	Error	Warn	Min	NotRun	Fail	Pass	Min		
pc-christophe.cst.cnes.fr	ArchLinux-64bits-Release	6	0.1	0	1	1.1	0	6	126	0	60 +15	2410 -15	130.6	2012-02-03T20:11:37 CET	(none)
pc-grizonnetm.cst.cnes.fr	Basic-Ubuntu10.04-64bits-Release	0	0.1	0	2	0.7	0	0	23	0	67 +2 -27	2346 +27 -2	25.2	2012-02-03T23:37:34 CET	(none)
titan	Deb50-64bits-Release	0	0	1	0	0	1	1	0	0	0	0	0	2012-02-03T20:15:19 CET	(none)
leod.c-s.fr	MacOSX10.5-Release-macport	6	0.1	0	0	1.9	0	0	72.9	0	87 +1 -1	2396 +1 -1	46.1	2012-02-03T20:42:25 CET	(none)

Summary

CMake basics

Using CMake basics we can already do a lot of things with minimal writing.

- Write simple build specification file: `CMakeLists.txt`
- Discover compilers (C, C++, Fortran)
- Build executable and library (shared or static) in a cross-platform manner
- Package the resulting binaries with CPack
- Run systematic tests with CTest and publish them with CDash

Seeking more information or help

There are several places you can go by yourself:

1. (re-)Read the documentation: https://cmake.org/documentation
2. Read the FAQ: https://cmake.org/Wiki/CMake_FAQ
3. Read the Wiki: https://cmake.org/Wiki/CMake
4. Ask on the Mailing List: https://cmake.org/mailing-lists
5. Browse the built-in help:

```
man cmake-xxxx
cmake --help-xxxxx
assistant -collectionFile examples/CMake.qhc
```

Outline

Part I: CMake

Outline

Part I: CMake

How to discover system

System/compiler specific variables

Right after the `project` command CMake has set up variables which can be used to tailor the build in a platform specific way.

- system specific
  - WIN32 True on Windows systems, including Win64.
  - UNIX True for UNIX and UNIX like operating systems.
  - APPLE True if running on Mac OS X.
  - CYGWIN True for Cygwin.
  - Have a look at `cmake --system-information` output
- compiler specific
  - CMAKE_<LANG>_COMPILER_ID A short string unique to the compiler vendor : Clang, GNU, MSVC, Cray, Absoft TI, XL...

Handle system specific code

Some functions like `strcasestr` (lines 6 and 7) may not be available on all platforms.

```
1  const acroItem_t* acrodict_get_approx(const char* name) {
2    int current =0;
3    int found   =0;
4  #ifdef GUESS_NAME
5    while ((strlen(acrodict[current].name)>0) && !found) {
6      if ((strcasestr(name,acrodict[current].name)!=0) ||
7          (strcasestr(acrodict[current].name,name)!=0)) {
8        found=1;
9      } else {
10       current++;
11     }
12   }
13   if (found) {
14     return &(acrodict[current]);
15   } else
16 #endif
17   {
18     return NULL;
19   }
20 }
```

Listing 11: excerpt from acrodict.c

Use system specific option

```
1 # Build option with default value to ON
2 option(WITH_ACRODICT "Include␣acronym␣dictionary␣support" ON)
3 if(NOT WIN32)
4   option(WITH_GUESS_NAME "Guess␣acronym␣name" ON)
5 endif(NOT WIN32)
6 ...
7 if (WITH_ACRODICT)
8    # list of sources in our library
9    set(LIBSRC acrodict.h acrodict.c)
10   if (WITH_GUESS_NAME)
11     set_source_files_properties(acrodict.c PROPERTIES COMPILE_FLAGS "-DGUESS_NAME")
12   endif(WITH_GUESS_NAME)
13   add_library(acrodict ${LIBSRC})
14 ...
```

Line 4 defines a CMake option, but not on WIN32 system. Then on line 11, if the option is set then we pass a source specific compile flags.

`cmake --help-command set_source_files_properties`

System specific in real life

Real [numeric] life project

Real projects (i.e. not the toy of this tutorial) have many parts of their `CMakeLists.txt` which deal with system/compiler specific option/feature.

- MuseScore
- CERTI, `http://git.savannah.gnu.org/cgit/certi.git/tree`
- SchedMCore, `https://svn.onera.fr/schedmcore/trunk/`
- CMake (of course)
- LLVM, `http://llvm.org/docs/CMake.html`
- many more ...

What about `projectConfig.h` file? I

Project config files

Sometimes it's easier to test for features and then write a configuration file (`config.h`, `project_config.h`, ...). The CMake way to do that is to:

1. lookup system information using CMake variable, functions, macros (built-in or imported) then set various variables,
2. use the defined variable in order to write a template configuration header file
3. then use `configure_file` in order to produce the actual config file from the template.

What about projectConfig.h file? II

```
1  # Load Checker macros
2  INCLUDE(CheckFunctionExists)
3
4  FIND_FILE(HAVE_STDINT_H NAMES stdint.h)
5  FIND_FILE(HAVE_SYS_SELECT_H NAMES select.h
6    PATH_SUFFIXES sys)
7  INCLUDE(CheckIncludeFile)
8  CHECK_INCLUDE_FILE(time.h HAVE_TIME_H)
9  FIND_LIBRARY(RT_LIBRARY rt)
10 if(RT_LIBRARY)
11   SET(CMAKE_REQUIRED_LIBRARIES ${CMAKE_REQUIRED_LIBRARIES} ${RT_LIBRARY})
12 endif(RT_LIBRARY)
13
14 CHECK_FUNCTION_EXISTS(clock_gettime HAVE_CLOCK_GETTIME)
15 CHECK_FUNCTION_EXISTS(clock_settime HAVE_CLOCK_SETTIME)
16 CHECK_FUNCTION_EXISTS(clock_getres HAVE_CLOCK_GETRES)
17 CHECK_FUNCTION_EXISTS(clock_nanosleep HAVE_CLOCK_NANOSLEEP)
18 IF (HAVE_CLOCK_GETTIME AND HAVE_CLOCK_SETTIME AND HAVE_CLOCK_GETRES)
19     SET(HAVE_POSIX_CLOCK 1)
20 ENDIF (HAVE_CLOCK_GETTIME AND HAVE_CLOCK_SETTIME AND HAVE_CLOCK_GETRES)
21 ...
22 CONFIGURE_FILE(${CMAKE_CURRENT_SOURCE_DIR}/config.h.cmake
23                 ${CMAKE_CURRENT_BINARY_DIR}/config.h)
```

Listing 12: Excerpt from CERTI project's main CMakeLists.txt

What about projectConfig.h file? III

Excerpt from CERTI config.h.cmake

```
/* define if the compiler has numeric_limits<T> */
#cmakedefine HAVE_NUMERIC_LIMITS

/* Define to 1 if you have the <stdint.h> header file. */
#cmakedefine HAVE_STDINT_H 1

/* Define to 1 if you have the <stdlib.h> header file. */
#cmakedefine HAVE_STDLIB_H 1

/* Define to 1 if you have the <strings.h> header file. */
#cmakedefine HAVE_STRINGS_H 1
...
/* Name of package */
#cmakedefine PACKAGE "@PACKAGE_NAME@"

/* Define to the address where bug reports for this package should be sent. */
#cmakedefine PACKAGE_BUGREPORT "@PACKAGE_BUGREPORT@"

/* Define to the full name of this package. */
#cmakedefine PACKAGE_NAME "@PACKAGE_NAME@"

/* Define to the full name and version of this package. */
#cmakedefine PACKAGE_STRING "@PACKAGE_NAME@-@PACKAGE_VERSION@"
```

And you get something like:

What about projectConfig.h file? IV

Excerpt from generated CERTI config.h

```
/* define if the compiler has numeric_limits<T> */
#define HAVE_NUMERIC_LIMITS

/* Define to 1 if you have the <stdint.h> header file. */
#define HAVE_STDINT_H 1

/* Define to 1 if you have the <stdlib.h> header file. */
#define HAVE_STDLIB_H 1

/* Define to 1 if you have the <strings.h> header file. */
#define HAVE_STRINGS_H 1
...
/* Name of package */
/* #undef PACKAGE */

/* Define to the address where bug reports for this package should be sent. */
#define PACKAGE_BUGREPORT "certi-devel@nongnu.org"

/* Define to the full name of this package. */
#define PACKAGE_NAME "CERTI"

/* Define to the full name and version of this package. */
/* #undef PACKAGE_STRING */
```

Outline

Part I: CMake

The `find_package` command I

Finding external package

Project may be using external libraries, programs, files etc...Those can be found using the `find_package` command.

```
1 find_package(LibXml2)
2 if (LIBXML2_FOUND)
3     add_definitions(-DHAVE_XML ${LIBXML2_DEFINITIONS})
4     include_directories(${LIBXML2_INCLUDE_DIR})
5 else (LIBXML2_FOUND)
6     set(LIBXML2_LIBRARIES "")
7 endif (LIBXML2_FOUND)
8 ...
9 target_link_libraries(MyTarget ${LIBXML2_LIBRARIES})
```

Listing 13: using libxml2

- Find modules usually define standard variables (for module XXX)

The `find_package` command II

1. `XXX_FOUND`: Set to false, or undefined, if we haven't found, or don't want to use `XXX`.
2. `XXX_INCLUDE_DIRS`: The final set of include directories listed in one variable for use by client code.
3. `XXX_LIBRARIES`: The libraries to link against to use `XXX`. These should include full paths.
4. `XXX_DEFINITIONS`: Definitions to use when compiling code that uses `XXX`.
5. `XXX_EXECUTABLE`: File location of the `XXX` tool's binary.
6. `XXX_LIBRARY_DIRS`: Optionally, the final set of library directories listed in one variable for use by client code.

- See doc `cmake --help-module FindLibXml2`
- Many modules are provided by CMake (230 as of CMake 3.6.2)

The `find_package` command III

- Projects which are built with CMake usually provide a *Project Config* file see doc: https://cmake.org/cmake/help/git-master/manual/cmake-packages.7.html
- You may write your own: https://cmake.org/Wiki/CMake:Module_Maintainers
- A module may provide not only CMake variables but new CMake macros (we will see that later with the `MACRO`, `FUNCTION` CMake language commands)

The other `find_xxxx` commands

The `find_xxx` command family

`find_package` is a *high level* module finding mechanism but there are lower-level CMake commands which may be used to write find modules or anything else inside `CMakeLists.txt`

- to find an executable program: `find_program`
- to find a library: `find_library`
- to find any kind of file: `find_file`
- to find a path where a file resides: `find_path`

FindPrelude.cmake example I

The FindPrelude.cmake is part of the *Prelude* synchronous language compiler made by ONERA: https://forge.onera.fr/projects/Prelude. This a source-to-source compiler which takes as input prelude file (.plul) and generates a bunch of C files which may be compiled in a dynamic library. The FindPrelude.cmake helps to automatize this task.

```
1  # - Find Prelude compiler
2  # Find the Prelude synchronous language compiler with associated includes path.
3  # See http://www.lifl.fr/~forget/prelude.html
4  # and https://forge.onera.fr/projects/prelude
5  # This module defines
6  #  PRELUDE_COMPILER, the prelude compiler
7  #  PRELUDE_COMPILER_VERSION, the version of the prelude compiler
8  #  PRELUDE_INCLUDE_DIR, where to find dword.h, etc.
9  #  PRELUDE_FOUND, If false, Prelude was not found.
10 # On can set PRELUDE_PATH_HINT before using find_package(Prelude) and the
11 # module with use the PATH as a hint to find preludec.
12 #
13 # The hint can be given on the command line too:
14 #   cmake -DPRELUDE_PATH_HINT=/DATA/ERIC/Prelude/prelude-x.y /path/to/source
15 #
```

FindPrelude.cmake example II

```
16 # The module defines some functions:
17 #   Prelude_Compile(NODE <Prelude Main Node>
18 #                    PLU_FILES <Prelude files>
19 #                   [USER_C_FILES <C files>]
20 #                   [NOENCODING]
21 #                   [REAL_IS_DOUBLE]
22 #                   [BOOL_IS_STDBOOL]
23 #                   [TRACING fmt])
24 #
25
26 if(PRELUDE_PATH_HINT)
27   message(STATUS "FindPrelude: using PATH HINT: ${PRELUDE_PATH_HINT}")
28 else()
29   set(PRELUDE_PATH_HINT)
30 endif()
31
32 #One can add his/her own builtin PATH.
33 #FILE(TO_CMAKE_PATH "/DATA/ERIC/Prelude/prelude-x.y" MYPATH)
34 #list(APPEND PRELUDE_PATH_HINT ${MYPATH})
35
36 # FIND_PROGRAM twice using NO_DEFAULT_PATH on first shot
37 find_program(PRELUDE_COMPILER
38   NAMES preludec
39   PATHS ${PRELUDE_PATH_HINT}
40   PATH_SUFFIXES bin
41   NO_DEFAULT_PATH
42   DOC "Path to the Prelude compiler command 'preludec'")
```

FindPrelude.cmake example III

```
43
44 find_program(PRELUDE_COMPILER
45   NAMES preludec
46   PATHS ${PRELUDE_PATH_HINT}
47   PATH_SUFFIXES bin
48   DOC "Path to the Prelude compiler command 'preludec'")
49
50 if(PRELUDE_COMPILER)
51     # get the path where the prelude compiler was found
52     get_filename_component(PRELUDE_PATH ${PRELUDE_COMPILER} PATH)
53     # remove bin
54     get_filename_component(PRELUDE_PATH ${PRELUDE_PATH} PATH)
55     # add path to PRELUDE_PATH_HINT
56     list(APPEND PRELUDE_PATH_HINT ${PRELUDE_PATH})
57     execute_process(COMMAND ${PRELUDE_COMPILER} -version
58         OUTPUT_VARIABLE PRELUDE_COMPILER_VERSION
59         OUTPUT_STRIP_TRAILING_WHITESPACE)
60     message(STATUS "Prelude compiler version is : ${PRELUDE_COMPILER_VERSION}")
61     execute_process(COMMAND ${PRELUDE_COMPILER} -help
62         OUTPUT_VARIABLE PRELUDE_OPTIONS_LIST
63         OUTPUT_STRIP_TRAILING_WHITESPACE)
64     set(PRELUDE_TRACING_OPTION)
65     string(REGEX MATCH "-tracing output" PRELUDE_TRACING_OPTION "${PRELUDE_OPTIONS_LIST}")
66     if (PRELUDE_TRACING_OPTION)
67       message(STATUS "Prelude compiler support -tracing.")
68       set(PRELUDE_SUPPORT_TRACING "YES")
69     else(PRELUDE_TRACING_OPTION)
```

FindPrelude.cmake example IV

```
70        message(STATUS "Prelude compiler DOES NOT support -tracing.")
71        set(PRELUDE_SUPPORT_TRACING "NO")
72      endif(PRELUDE_TRACING_OPTION)
73  endif(PRELUDE_COMPILER)
74
75  find_path(PRELUDE_INCLUDE_DIR
76            NAMES dword.h
77            PATHS ${PRELUDE_PATH_HINT}
78            PATH_SUFFIXES lib/prelude
79            DOC "The Prelude include headers")
80
81  # Check if LTTng is to be supported
82  if (NOT LTTNG_FOUND)
83    option(ENABLE_LTTNG_SUPPORT "Enable LTTng support" OFF)
84    if(ENABLE_LTTNG_SUPPORT)
85      find_package(LTTng)
86      if (LTTNG_FOUND)
87        message(STATUS "Will build LTTng support into library...")
88        include_directories(${LTTNG_INCLUDE_DIR})
89      endif(LTTNG_FOUND)
90    endif(ENABLE_LTTNG_SUPPORT)
91  endif()
92
93  # Macros used to compile a prelude library
94  include(CMakeParseArguments)
95  function(Prelude_Compile)
96    set(options NOENCODING REAL_IS_DOUBLE BOOL_IS_STDBOOL)
```

FindPrelude.cmake example V

```
97  set(oneValueArgs NODE TRACING)
98  set(multiValueArgs PLU_FILES USER_C_FILES)
99  cmake_parse_arguments(PLU "${options}" "${oneValueArgs}" "${multiValueArgs}" ${ARGN})
100
101 if(PLU_NOENCODING)
102   set(PRELUDE_ENCODING "-no_encoding")
103   set(PRELUDE_OUTPUT_DIR "${CMAKE_CURRENT_BINARY_DIR}/${PLU_NODE}/noencoding")
104   set(PRELUDE_ENCODING_SUFFIX "-noencoding")
105 else()
106   set(PRELUDE_ENCODING)
107   set(PRELUDE_OUTPUT_DIR "${CMAKE_CURRENT_BINARY_DIR}/${PLU_NODE}/encoded")
108   set(PRELUDE_ENCODING_SUFFIX "-encoded")
109 endif()
110
111 if(PLU_REAL_IS_DOUBLE)
112   set(PRELUDE_REAL_OPT "-real_is_double")
113 else()
114   set(PRELUDE_REAL_OPT "")
115 endif()
116
117 if(PLU_BOOL_IS_STDBOOL)
118   set(PRELUDE_BOOL_OPT "-bool_is_stdbool")
119 else()
120   set(PRELUDE_BOOL_OPT "")
121 endif()
122
123 if (PRELUDE_SUPPORT_TRACING)
```

FindPrelude.cmake example VI

```
124     if(PLU_TRACING)
125       set(PRELUDE_TRACING_OPT "-tracing")
126       set(PRELUDE_TRACING_OPT_VALUE "${PLU_TRACING}")
127     else()
128       set(PRELUDE_TRACING_OPT "-tracing")
129       set(PRELUDE_TRACING_OPT_VALUE "no")
130     endif()
131   else(PRELUDE_SUPPORT_TRACING)
132     set(PRELUDE_TRACING_OPT "")
133     set(PRELUDE_TRACING_OPT_VALUE "")
134   endif(PRELUDE_SUPPORT_TRACING)
135
136   file(MAKE_DIRECTORY ${PRELUDE_OUTPUT_DIR})
137   set(PRELUDE_GENERATED_FILES
138       ${PRELUDE_OUTPUT_DIR}/${PLU_NODE}.c
139       ${PRELUDE_OUTPUT_DIR}/${PLU_NODE}.h)
140
141   add_custom_command(
142       OUTPUT ${PRELUDE_GENERATED_FILES}
143       COMMAND ${PRELUDE_COMPILER} ${PRELUDE_ENCODING} ${PRELUDE_BOOL_OPT} ${PRELUDE_REAL_OPT} $
      {PRELUDE_TRACING_OPT} ${PRELUDE_TRACING_OPT_VALUE} -d ${PRELUDE_OUTPUT_DIR} -node ${
      PLU_NODE} ${PLU_PLU_FILES}
144       DEPENDS ${PLU_PLU_FILES}
145       WORKING_DIRECTORY ${CMAKE_CURRENT_SOURCE_DIR}
146       COMMENT "Compile␣prelude␣source(s):␣${PLU_PLU_FILES})"
147   )
148   set_source_files_properties(${PRELUDE_GENERATED_FILES}
```

FindPrelude.cmake example VII

```
149                                 PROPERTIES GENERATED TRUE)
150   include_directories(${PRELUDE_INCLUDE_DIR} ${CMAKE_CURRENT_SOURCE_DIR} ${PRELUDE_OUTPUT_DIR})
151   add_library(${PLU_NODE}${PRELUDE_ENCODING_SUFFIX} SHARED
152               ${PRELUDE_GENERATED_FILES} ${PLU_USER_C_FILES}
153             )
154   if(LTTNG_FOUND)
155     target_link_libraries(${PLU_NODE}${PRELUDE_ENCODING_SUFFIX} ${LTTNG_LIBRARIES})
156   endif()
157   message(STATUS "Prelude: Added rule for building prelude library: ${PLU_NODE}")
158 endfunction(Prelude_Compile)
159
160 # handle the QUIETLY and REQUIRED arguments and set PRELUDE_FOUND to TRUE if
161 # all listed variables are TRUE
162 include(FindPackageHandleStandardArgs)
163 FIND_PACKAGE_HANDLE_STANDARD_ARGS(PRELUDE
164                                   REQUIRED_VARS PRELUDE_COMPILER PRELUDE_INCLUDE_DIR)
165 # VERSION FPHSA options not handled by CMake version < 2.8.2)
166 #                                    VERSION_VAR PRELUDE_COMPILER_VERSION)
167 mark_as_advanced(PRELUDE_INCLUDE_DIR)
```

Advanced use of external package I

Installed External package

The previous examples suppose that you have the package you are looking for on your host.

- you did install the runtime libraries
- you did install eventual developer libraries, headers and tools

What if the external packages:

- are only available as source (tarball, VCS repositories, ...)
- use a build system (autotools or CMake or ...)

Advanced use of external package II

ExternalProject_Add

The `ExternalProject.cmake` CMake module defines a high-level macro which does just that:

1. download/checkout source
2. update/patch
3. configure
4. build
5. install (and test)

...an external project

Advanced use of external package III

```
$ cmake --help-module ExternalProject
```

Outline

The different CMake "modes"

- Normal mode: the mode used when processing `CMakeLists.txt`
- Command mode: `cmake -E <command>`, command line mode which offers basic commands in a portable way:

- Process scripting mode: `cmake -P <script>`, used to execute a CMake script which is not a CMakeLists.txt filename.

- Wizard mode: `cmake -i`, interactive equivalent of the Normal mode.

The different CMake "modes"

- Normal mode: the mode used when processing `CMakeLists.txt`
- Command mode: `cmake -E <command>`, command line mode which offers basic commands in a portable way:
  works on all supported CMake platforms. I.e. you don't want to rely on shell or native command interpreter capabilities.
- Process scripting mode: `cmake -P <script>`, used to execute a CMake script which is not a CMakeLists.txt filename.

- Wizard mode: `cmake -i`, interactive equivalent of the Normal mode.

The different CMake "modes"

- Normal mode: the mode used when processing `CMakeLists.txt`
- Command mode: `cmake -E <command>`, command line mode which offers basic commands in a portable way:
  works on all supported CMake platforms. I.e. you don't want to rely on shell or native command interpreter capabilities.
- Process scripting mode: `cmake -P <script>`, used to execute a CMake script which is not a CMakeLists.txt filename.
  Not all CMake commands are scriptable!!
- Wizard mode: `cmake -i`, interactive equivalent of the Normal mode.

Command mode

list of command mode commands

```
  $ cmake -E
CMake Error: cmake version 3.6.2
Usage: cmake -E <command> [arguments...]
Available commands:
  chdir dir cmd [args...]   - run command in a given directory
  compare_files file1 file2 - check if file1 is same as file2
  copy <file>... destination  - copy files to destination (either file or directory)
  copy_directory <dir>... destination   - copy content of <dir>... directories to 'destination' directory
  copy_if_different <file>... destination  - copy files if it has changed
  echo [<string>...]        - displays arguments as text
  echo_append [<string>...] - displays arguments as text but no new line
  env [--unset=NAME]... [NAME=VALUE]... COMMAND [ARG]...
                            - run command in a modified environment
  environment               - display the current environment
  make_directory <dir>...   - create parent and <dir> directories
  md5sum <file>...          - create MD5 checksum of files
  remove [-f] <file>...     - remove the file(s), use -f to force it
  remove_directory dir      - remove a directory and its contents
  rename oldname newname    - rename a file or directory (on one volume)
  tar [cxt][vf][zjJ] file.tar [file/dir1 file/dir2 ...]
                            - create or extract a tar or zip archive
  sleep <number>...         - sleep for given number of seconds
  time command [args...]    - run command and return elapsed time
  touch file                - touch a file.
  touch_nocreate file       - touch a file but do not create it.
Available on UNIX only:
  create_symlink old new    - create a symbolic link new -> old
```

CMake scripting

Overview of CMake language

CMake is a declarative language which contains 90+ *commands*. It contains general purpose constructs: `set`,`unset`, `if`,`elseif`,`else`,`endif`, `foreach`, `while`, `break` see cmake-language(7

Remember:

```
$ cmake --help-command-list
$ cmake --help-command <command-name>
$ cmake --help-command message
cmake version 2.8.7
  message
       Display a message to the user.
         message([STATUS|WARNING|AUTHOR_WARNING|FATAL_ERROR|SEND_ERROR]
                 "message to display" ...)
       The optional keyword determines the type of message:
         (none)         = Important information
         STATUS         = Incidental information
         WARNING        = CMake Warning, continue processing
         AUTHOR_WARNING = CMake Warning (dev), continue processing
         SEND_ERROR     = CMake Error, continue but skip generation
         FATAL_ERROR    = CMake Error, stop all processing
```

Higher level commands as well

- file manipulation with file : READ, WRITE, APPEND, RENAME, REMOVE, MAKE_DIRECTORY
- advanced files operations: GLOB, GLOB_RECURSE file name in a path, DOWNLOAD, UPLOAD
- working with path: file (TO_CMAKE_PATH /TO_NATIVE_PATH ...), get_filename_component
- execute an external process (with stdout, stderr and return code retrieval): execute_process
- builtin list manipulation command: list with sub-commands LENGTH, GET, APPEND, FIND, APPEND, INSERT, REMOVE_ITEM, REMOVE_AT, REMOVE_DUPLICATES REVERSE, SORT
- string manipulation: string , upper/lower case conversion, length, comparison, substring, regular expression match, ...

Portable script for building CMake I

As an example of what can be done with pure CMake script (script mode) here is a script for building the CMake package using a previously installed CMake.

```
1  # Simple cmake script which may be used to build
2  # cmake from automatically downloaded source
3  #
4  #   cd tmp/
5  #   cmake -P CMake-autobuild-v2.cmake
6  # you should end up with a
7  #   tmp/cmake-x.y.z source tree
8  #   tmp/cmake-x.y.z-build build tree
9  # configure and compiled tree, using the tarball found on Kitware.
10 #
11 # if you access the internet through a proxy then you should
12 # set the "http_proxy" and "https_proxy" environment variable
13 # to apppropriate value before running the CMake script.
14 #  e.g.:
15 #     export http_proxy=http://myproxy.mydomain.fr:80
16 #     export https_proxy=https://myproxy.mydomain.fr:80
17
18
19 cmake_minimum_required(VERSION 3.0)
20 set(CMAKE_VERSION "3.6.2")
```

Portable script for building CMake II

```
21 set(CMAKE_FILE_PREFIX "cmake-${CMAKE_VERSION}")
22 string(REGEX MATCH "[0-9]\\.[0-9]" CMAKE_MAJOR "${CMAKE_VERSION}")
23 set(CMAKE_REMOTE_PREFIX "http://www.cmake.org/files/v${CMAKE_MAJOR}/")
24 set(CMAKE_FILE_SUFFIX ".tar.gz")
25 set(CMAKE_BUILD_TYPE "Debug")
26 set(CMAKE_BUILD_QTDIALOG "ON")
27 set(CMAKE_BUILD_GENERATOR "")
28 #try Ninja (https://ninja-build.org) if you have it installed
29 #set(CMAKE_BUILD_GENERATOR "-GNinja")
30 set(CPACK_GEN "TGZ")
31 #try another CPack generator
32 set(CPACK_GEN "RPM")
33
34 set(LOCAL_FILE "./${CMAKE_FILE_PREFIX}${CMAKE_FILE_SUFFIX}")
35 set(REMOTE_FILE "${CMAKE_REMOTE_PREFIX}${CMAKE_FILE_PREFIX}${CMAKE_FILE_SUFFIX}")
36
37 message(STATUS "Trying␣to␣autoinstall␣CMake␣version␣${CMAKE_VERSION}␣using␣${REMOTE_FILE}␣file
       ...")
38
39 message(STATUS "Downloading...")
40 if (EXISTS ${LOCAL_FILE})
41    message(STATUS "Already␣there:␣nothing␣to␣do")
42 else (EXISTS ${LOCAL_FILE})
43    message(STATUS "Not␣there,␣trying␣to␣download...")
44    file(DOWNLOAD ${REMOTE_FILE} ${LOCAL_FILE}
45         TIMEOUT 600
46         STATUS DL_STATUS
```

Portable script for building CMake III

```
47          LOG DL_LOG
48          SHOW_PROGRESS)
49     list(GET DL_STATUS 0 DL_NOK)
50     if ("${DL_LOG}" MATCHES "404␣Not␣Found")
51        set(DL_NOK 1)
52     endif ("${DL_LOG}" MATCHES "404␣Not␣Found")
53     if (DL_NOK)
54        # we shall remove the file because it is created
55        # with an inappropriate content
56        file(REMOVE ${LOCAL_FILE})
57        message(SEND_ERROR "Download␣failed:␣${DL_LOG}")
58     else (DL_NOK)
59        message(STATUS "Download␣successful.")
60     endif (DL_NOK)
61 endif (EXISTS ${LOCAL_FILE})
62
63 message(STATUS "Unarchiving␣the␣file")
64 execute_process(COMMAND ${CMAKE_COMMAND} -E tar zxvf ${LOCAL_FILE}
65                 RESULT_VARIABLE UNTAR_RES
66                 OUTPUT_VARIABLE UNTAR_OUT
67                 ERROR_VARIABLE UNTAR_ERR
68                 )
69 message(STATUS "CMake␣version␣${CMAKE_VERSION}␣has␣been␣unarchived␣in␣${
       CMAKE_CURRENT_SOURCE_DIR}/${CMAKE_FILE_PREFIX}.")
70
71 message(STATUS "Configuring␣with␣CMake␣(build␣type=${CMAKE_BUILD_TYPE},␣QtDialog=${
       CMAKE_BUILD_QTDIALOG},␣build␣generator=${CMAKE_BUILD_GENERATOR})...")
```

Portable script for building CMake IV

```
72 file(MAKE_DIRECTORY ${CMAKE_FILE_PREFIX}-build)
73 execute_process(COMMAND ${CMAKE_COMMAND} ${CMAKE_BUILD_GENERATOR} -DCMAKE_BUILD_TYPE=${
       CMAKE_BUILD_TYPE} -DBUILD_QtDialog:BOOL=${CMAKE_BUILD_QTDIALOG} ../${CMAKE_FILE_PREFIX}
74                 WORKING_DIRECTORY ${CMAKE_FILE_PREFIX}-build
75                 RESULT_VARIABLE CONFIG_RES
76                 OUTPUT_VARIABLE CONFIG_OUT
77                 ERROR_VARIABLE CONFIG_ERR
78                 TIMEOUT 200
79                 )
80 if (CONFIG_RES)
81    message(ERROR "Configuration␣failed:␣${CONFIG_OUT}␣/␣${CONFIG_ERR}")
82 endif()
83
84 message(STATUS "Building␣with␣cmake␣--build␣...")
85 execute_process(COMMAND ${CMAKE_COMMAND} --build .
86                 WORKING_DIRECTORY ${CMAKE_FILE_PREFIX}-build
87                 RESULT_VARIABLE CONFIG_RES
88                 OUTPUT_VARIABLE CONFIG_OUT
89                 ERROR_VARIABLE CONFIG_ERR
90                 )
91
92 message(STATUS "Create␣package␣${CPACK_GEN}␣with␣CPack...")
93 execute_process(COMMAND ${CMAKE_CPACK_COMMAND} -G ${CPACK_GEN}
94                 WORKING_DIRECTORY ${CMAKE_FILE_PREFIX}-build
95                 RESULT_VARIABLE CONFIG_RES
96                 OUTPUT_VARIABLE CONFIG_OUT
97                 ERROR_VARIABLE CONFIG_ERR
```

Portable script for building CMake V

```
98                 )
99 message(STATUS "CMake version ${CMAKE_VERSION} has been built in ${CMAKE_CURRENT_SOURCE_DIR}/${
       CMAKE_FILE_PREFIX}.")
100 string(REGEX MATCH "CPack: - package: (.*) generated" PACKAGES "${CONFIG_OUT}")
101 message(STATUS "CMake package(s) are: ${CMAKE_MATCH_1}")
```

Portable script for building ROSACE Case Study

Another example taken from the "*ROSACE Open Source Case Study*" may be found here see ROSACE-CaseStudy-auto.cmake
The script:

- Download or checkout Lustre compiler and build it
- Download or checkout Prelude compiler and build it
- Download or checkout SchedMCore toolsuit and build it
- Checkout the ROSACE Case Study and build it using the previously built tools.

Build specific commands

- create executable or library: `add_executable`, `add_library`
- add compiler/linker definitions/options: `add_definitions`, `include_directories`, `target_link_libraries`
- powerful installation specification: `install`
- probing command: `try_compile`, `try_run`
- fine control of various properties: `set_target_properties`, `set_source_files_properties`, `set_directory_properties`, `set_tests_properties`, `set_property`: 300+ different properties may be used.

```
$ cmake --help-property-list
$ cmake --help-property COMPILE_FLAGS
```

Outline

What are CMake targets?

CMake target

Many times in the documentation you may read about CMake *target*. A target is something that CMake should build (i.e. generate something enabling the building of the *target*).
A CMake target has dependencies and properties.

1. Executables are targets: `add_executable`
2. Libraries are targets: `add_library`
3. There exist some builtin targets: `install`, `clean`, `package`, ...
4. You may create custom targets: `add_custom_target`

Target dependencies and properties I

A CMake target has dependencies and properties.

Dependencies

Most of the time, source dependencies are computed from target specifications using CMake builtin dependency scanner (C, C++, Fortran) whereas library dependencies are inferred via `target_link_libraries` specification.

If this is not enough then one can use `add_dependencies`, or some properties.

Properties

Properties may be attached to either *target* or *source file* (or even *test*). They may be used to tailor the prefix or suffix to be used for libraries, compile flags, link flags, linker language, shared libraries version, ...

Target dependencies and properties II

see : `set_target_properties` or `set_source_files_properties`

Sources vs Targets

Properties set to a target like `COMPILE_FLAGS` are used for all sources of the concerned target. Properties set to a source are used for the source file itself (which may be involved in several targets).

Custom targets and commands

Custom

Custom targets and custom commands are a way to create a *target* which may be used to execute arbitrary commands at Build-time.

- for target : `add_custom_target`
- for command : `add_custom_command`, in order to add some custom build step to another (existing) target.

This is usually for: generating source files (Flex, Bison) or other files derived from source like embedded documentation (Doxygen), ...

Outline

Generated files

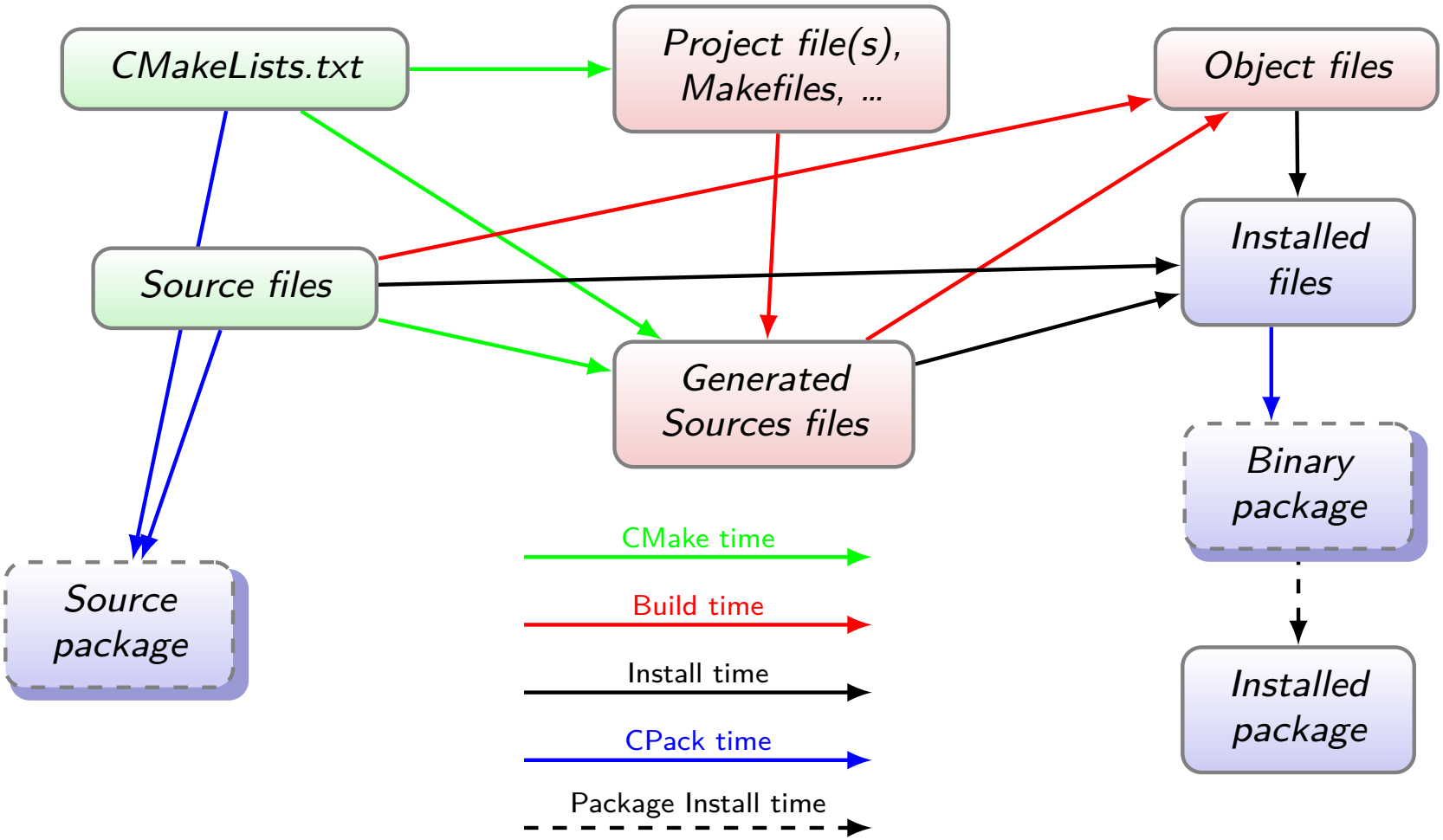
List all the sources

CMake advocates to specify all the source files explicitly (i.e. do not use `file(GLOB ...)`) This is the only way to keep robust dependencies. Moreover you usually already need to do that when using a VCS (CVS, Subversion, Git, hg,...).

However some files may be generated during the build (using `add_custom_xxx`), in which case you must tell CMake that they are `GENERATED` files using:

```
1 set_source_files_properties(${SOME_GENERATED_FILES}
2                             PROPERTIES GENERATED TRUE)
```

The CMake workflow (pictured)



Example I

```
1  ### Handle Source generation for task file parser
2  include_directories(${CMAKE_CURRENT_SOURCE_DIR})
3  find_package(LexYacc)
4  set(YACC_SRC                ${CMAKE_CURRENT_SOURCE_DIR}/lsmc_taskfile_syntax.yy)
5  set(YACC_OUT_PREFIX         ${CMAKE_CURRENT_BINARY_DIR}/y.tab)
6  set(YACC_WANTED_OUT_PREFIX ${CMAKE_CURRENT_BINARY_DIR}/lsmc_taskfile_syntax)
7  set(LEX_SRC                ${CMAKE_CURRENT_SOURCE_DIR}/lsmc_taskfile_tokens.ll)
8  set(LEX_OUT_PREFIX         ${CMAKE_CURRENT_BINARY_DIR}/lsmc_taskfile_tokens_yy)
9  set(LEX_WANTED_OUT_PREFIX ${CMAKE_CURRENT_BINARY_DIR}/lsmc_taskfile_tokens)
10
11 #Exec Lex
12 add_custom_command(
13    OUTPUT  ${LEX_WANTED_OUT_PREFIX}.c
14    COMMAND ${LEX_PROGRAM} ARGS -l -o${LEX_WANTED_OUT_PREFIX}.c ${LEX_SRC}
15    DEPENDS ${LEX_SRC}
16    )
17 set(GENERATED_SRCS ${GENERATED_SRCS} ${LEX_WANTED_OUT_PREFIX}.c)
18 #Exec Yacc
19 add_custom_command(
20    OUTPUT  ${YACC_WANTED_OUT_PREFIX}.c ${YACC_WANTED_OUT_PREFIX}.h
21    COMMAND ${YACC_PROGRAM} ARGS ${YACC_COMPAT_ARG} -d ${YACC_SRC}
22    COMMAND ${CMAKE_COMMAND} -E copy ${YACC_OUT_PREFIX}.h  ${YACC_WANTED_OUT_PREFIX}.h
23    COMMAND ${CMAKE_COMMAND} -E copy ${YACC_OUT_PREFIX}.c  ${YACC_WANTED_OUT_PREFIX}.c
24    DEPENDS ${YACC_SRC}
25    )
26 set(GENERATED_SRCS ${GENERATED_SRCS}
```

Example II

```
27      ${YACC_WANTED_OUT_PREFIX}.c ${YACC_WANTED_OUT_PREFIX}.h)
28 # Tell CMake that some file are generated
29 set_source_files_properties(${GENERATED_SRCS} PROPERTIES GENERATED TRUE)
30
31 # Inhibit compiler warning for LEX/YACC generated files
32 # Note that the inhibition is COMPILER dependent ...
33 # GNU CC specific warning stop
34 if (CMAKE_COMPILER_IS_GNUCC)
35    message(STATUS "INHIBIT␣Compiler␣warning␣for␣LEX/YACC␣generated␣files")
36    SET_SOURCE_FILES_PROPERTIES(${YACC_WANTED_OUT_PREFIX}.c ${YACC_WANTED_OUT_PREFIX}.h
37                                PROPERTIES COMPILE_FLAGS "-w")
38
39    SET_SOURCE_FILES_PROPERTIES(${LEX_WANTED_OUT_PREFIX}.c
40                                PROPERTIES COMPILE_FLAGS "-w")
41 endif(CMAKE_COMPILER_IS_GNUCC)
42 ...
43 set(LSCHED_SRC
44     lsmc_dependency.c  lsmc_core.c  lsmc_utils.c
45     lsmc_time.c lsmc_taskfile_parser.c
46     ${GENERATED_SRCS})
47 add_library(lsmc ${LSCHED_SRC})
```

Outline

Part I: CMake

Outline

Part I: CMake

Cross-compiling

Definition: Cross-compiling

Cross-compiling is when the *host* system, *the one the compiler is running on*, is not the same as the *target* system, *the one the compiled program will be running on*.

CMake can handle cross-compiling using a *Toolchain* description file, see https://cmake.org/Wiki/CMake_Cross_Compiling.

```
mkdir build-win32
cd build-win32
cmake -DCMAKE_TOOLCHAIN_FILE=../totally-free/Toolchain-cross-mingw32-linux.cmake ../totally-free/
```

Demo

Linux to Windows Toolchain example

```
# the name of the target operating system
SET(CMAKE_SYSTEM_NAME Windows)

# Choose an appropriate compiler prefix
# for classical mingw32
# see http://www.mingw.org/
#set(COMPILER_PREFIX "i586-mingw32msvc")
# for 32 or 64 bits mingw-w64
# see http://mingw-w64.sourceforge.net/
set(COMPILER_PREFIX "i686-w64-mingw32")
#set(COMPILER_PREFIX "x86_64-w64-mingw32"

# which compilers to use for C and C++
find_program(CMAKE_RC_COMPILER NAMES ${COMPILER_PREFIX}-windres)
#SET(CMAKE_RC_COMPILER ${COMPILER_PREFIX}-windres)
find_program(CMAKE_C_COMPILER NAMES ${COMPILER_PREFIX}-gcc)
#SET(CMAKE_C_COMPILER ${COMPILER_PREFIX}-gcc)
find_program(CMAKE_CXX_COMPILER NAMES ${COMPILER_PREFIX}-g++)
#SET(CMAKE_CXX_COMPILER ${COMPILER_PREFIX}-g++)

# here is the target environment located
SET(USER_ROOT_PATH /home/erk/erk-win32-dev)
SET(CMAKE_FIND_ROOT_PATH  /usr/${COMPILER_PREFIX} ${USER_ROOT_PATH})
# adjust the default behaviour of the FIND_XXX() commands:
# search headers and libraries in the target environment, search
# programs in the host environment
set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
```

Outline

Selecting language standard

Sometimes one needs to select a required programming language standard level like C99 or C++11. The command line option used to select the appropriate standard vary from one compiler to another.
There exist CMake variables controlling this are:

- for C++: `CMAKE_CXX_STANDARD`, `CMAKE_CXX_STANDARD_REQUIRED`
- for C: `CMAKE_C_STANDARD`, `CMAKE_C_STANDARD_REQUIRED`

See a nice tutorial
https://crascit.com/2015/03/28/enabling-cxx11-in-cmake/ giving more detailed usage example for C++11.

Enabling/selecting language feature

Sometimes requiring a whole standard support is too constraining since one only requires a specific langage feature which may be supported by the compiler long before it supports the whole standard requirements.
Since CMake 3.1 one can inspect and enable language features:
https://cmake.org/cmake/help/latest/manual/cmake-compile-features.7.html

Outline

Part I: CMake

Exporting/Import your project

Export/Import to/from others

CMake can help a project using CMake as a build system to export/import targets to/from another project using CMake as a build system.

No more time for that today sorry, see:
https://cmake.org/cmake/help/latest/manual/cmake-packages.7.html#creating-packages

Outline

Part I: CMake Part II: CPack

Introduction

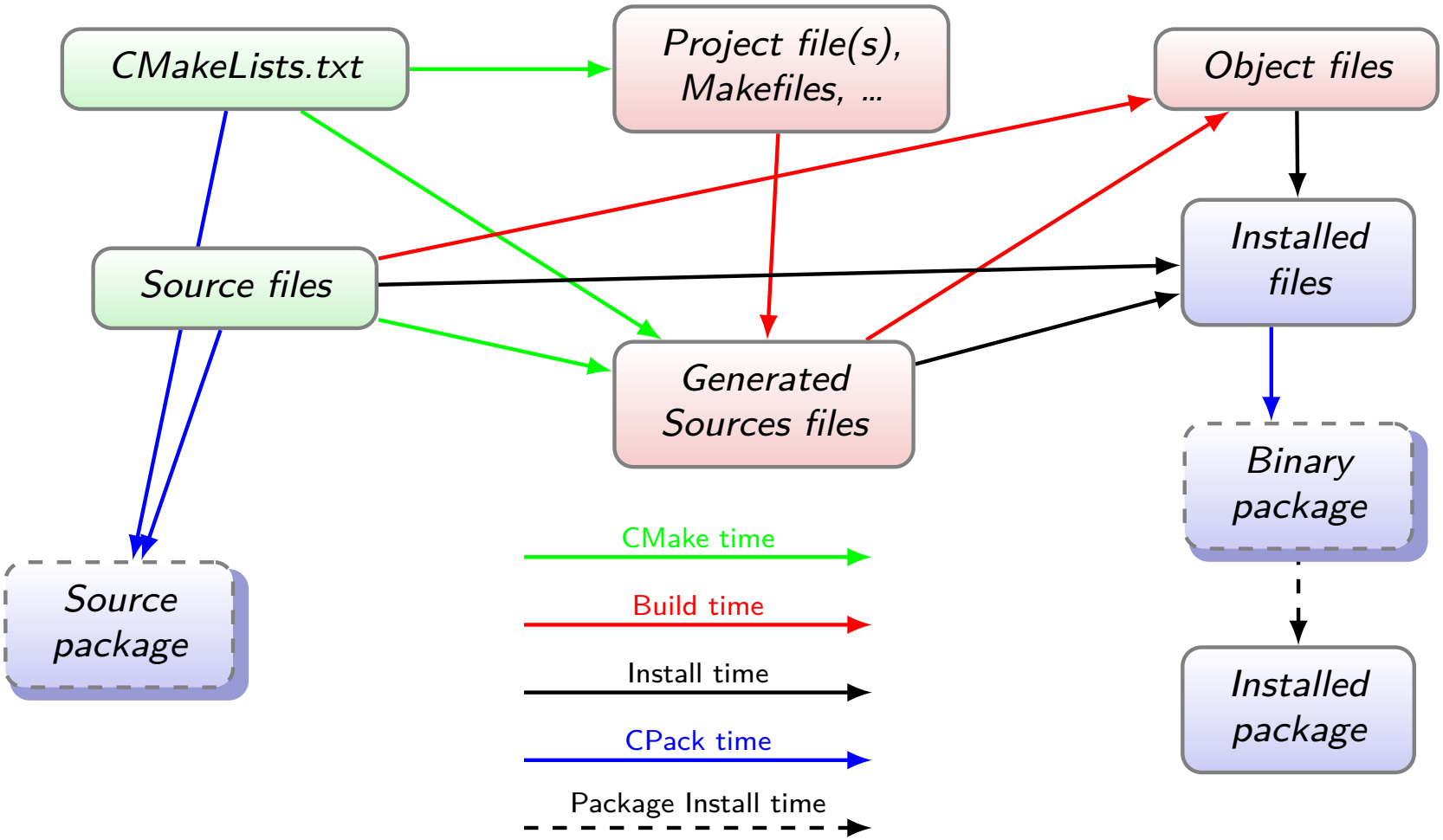
A Package **generator**

In the same way that CMake *generates* build files, CPack *generates* package files.

- Archive generators [ZIP,TGZ,...] (All platforms)
- DEB, RPM (Linux)
- Cygwin Source or Binary (Windows/Cygwin)
- NSIS (Windows, Linux)
- DragNDrop, Bundle, OSXX11 (Mac OS)

Outline

The CMake workflow (pictured)



The CPack application

CPack standalone

CPack is a standalone application whose behavior is driven by a configuration file e.g. `CPackConfig.cmake`. This file is a CMake language script which defines `CPACK_XXXX` variables:
the config parameters of the CPack run.

CPack with CMake

When CPack is used to package a project built with CPack, then the CPack configuration is usually generated by CMake by including `CPack.cmake` in the main `CMakeLists.txt`:

```
include(CPack)
```

CPack variables in `CMakeLists.txt`

When used with CMake, one writes something like this in `CMakeLists.txt`:

```
1  set(CPACK_GENERATOR "TGZ")
2  if (WIN32)
3     list(APPEND CPACK_GENERATOR "NSIS")
4  elseif (APPLE)
5     list(APPEND CPACK_GENERATOR "Bundle")
6  endif(WIN32)
7  set(CPACK_SOURCE_GENERATOR "ZIP;TGZ")
8  set(CPACK_PACKAGE_VERSION_MAJOR 0)
9  set(CPACK_PACKAGE_VERSION_MINOR 1)
10 set(CPACK_PACKAGE_VERSION_PATCH 0)
11 include(CPack)
```

This will create `CPackSourceConfig.cmake` and `CPackConfig.cmake` in the build tree and will bring you the *package* and *package_source* built-in targets.

A CPack config file I

A CPack config file looks like this one:

```
# This file will be configured to contain variables for CPack.
# These variables should be set in the CMake list file of the
# project before CPack module is included.
...
SET(CPACK_BINARY_BUNDLE "")
SET(CPACK_BINARY_CYGWIN "")
SET(CPACK_BINARY_DEB "")
...
SET(CPACK_BINARY_ZIP "")
SET(CPACK_CMAKE_GENERATOR "Unix Makefiles")
SET(CPACK_GENERATOR "TGZ")
SET(CPACK_INSTALL_CMAKE_PROJECTS "/home/erk/erkit/CMakeTutorial/examples/build;TotallyFree;ALL
    ;/")
SET(CPACK_INSTALL_PREFIX "/usr/local")
SET(CPACK_MODULE_PATH "")
SET(CPACK_NSIS_DISPLAY_NAME "TotallyFree 0.1.0")
SET(CPACK_NSIS_INSTALLER_ICON_CODE "")
SET(CPACK_NSIS_INSTALL_ROOT "$PROGRAMFILES")
SET(CPACK_NSIS_PACKAGE_NAME "TotallyFree 0.1.0")
SET(CPACK_OUTPUT_CONFIG_FILE "/home/erk/erkit/CMakeTutorial/examples/build/CPackConfig.cmake")
SET(CPACK_PACKAGE_DEFAULT_LOCATION "/")
SET(CPACK_PACKAGE_DESCRIPTION_FILE "/home/erk/CMake/cmake-Verk-HEAD/share/cmake-2.8/Templates/
    CPack.GenericDescription.txt")
SET(CPACK_PACKAGE_DESCRIPTION_SUMMARY "TotallyFree built using CMake")
```

A CPack config file II

```
23 SET(CPACK_PACKAGE_FILE_NAME "TotallyFree-0.1.0-Linux")
24 SET(CPACK_PACKAGE_INSTALL_DIRECTORY "TotallyFree␣0.1.0")
25 SET(CPACK_PACKAGE_INSTALL_REGISTRY_KEY "TotallyFree␣0.1.0")
26 SET(CPACK_PACKAGE_NAME "TotallyFree")
27 SET(CPACK_PACKAGE_RELOCATABLE "true")
28 SET(CPACK_PACKAGE_VENDOR "Humanity")
29 SET(CPACK_PACKAGE_VERSION "0.1.0")
30 SET(CPACK_RESOURCE_FILE_LICENSE "/home/erk/CMake/cmake-Verk-HEAD/share/cmake-2.8/Templates/
       CPack.GenericLicense.txt")
31 SET(CPACK_RESOURCE_FILE_README "/home/erk/CMake/cmake-Verk-HEAD/share/cmake-2.8/Templates/CPack
       .GenericDescription.txt")
32 SET(CPACK_RESOURCE_FILE_WELCOME "/home/erk/CMake/cmake-Verk-HEAD/share/cmake-2.8/Templates/
       CPack.GenericWelcome.txt")
33 SET(CPACK_SET_DESTDIR "OFF")
34 SET(CPACK_SOURCE_CYGWIN "")
35 SET(CPACK_SOURCE_GENERATOR "TGZ;TBZ2;TZ")
36 SET(CPACK_SOURCE_OUTPUT_CONFIG_FILE "/home/erk/erkit/CMakeTutorial/examples/build/
       CPackSourceConfig.cmake")
37 SET(CPACK_SOURCE_TBZ2 "ON")
38 SET(CPACK_SOURCE_TGZ "ON")
39 SET(CPACK_SOURCE_TZ "ON")
40 SET(CPACK_SOURCE_ZIP "OFF")
41 SET(CPACK_SYSTEM_NAME "Linux")
42 SET(CPACK_TOPLEVEL_TAG "Linux")
```

CPack running steps I

For a CMake enabled project one can run CPack in two ways:

1. use the build tool to run targets: `package` or `package_source`
2. invoke CPack manually from within the *build tree* e.g.:
   `$ cpack -G RPM`

The CPack documentation is currently found on the Wiki or on the CPack specific modules:

- https://cmake.org/Wiki/CMake:CPackPackageGenerators
- https://cmake.org/Wiki/CMake:Component_Install_With_CPack
- `cpack --help-module CPackXXX` with CPack, CPackComponent, CPackRPM, CPackDEB, CPackIFW, CPackWIX, ...

Whichever way you call it, the CPack steps are:

CPack running steps II

1. cpack command starts and parses arguments etc...
2. it reads `CPackConfig.cmake` (usually found in the build tree) or the file given as an argument to `--config` command line option.
3. it iterates over the generators list found in `CPACK_GENERATOR` (or from `-G` command line option). For each generator:
   1. (re)sets `CPACK_GENERATOR` to the one currently being iterated over
   2. includes the `CPACK_PROJECT_CONFIG_FILE`
   3. installs the project into a CPack private location (using `DESTDIR`)
   4. calls the generator and produces the package(s) for that generator

CPack running steps III

cpack command line example

```
$ cpack -G "TGZ;RPM"
CPack: Create package using TGZ
CPack: Install projects
CPack: - Run preinstall target for: TotallyFree
CPack: - Install project: TotallyFree
CPack: Create package
CPack: - package: <...>/build/TotallyFree-0.1.0-Linux.tar.gz generated.
CPack: Create package using RPM
CPack: Install projects
CPack: - Run preinstall target for: TotallyFree
CPack: - Install project: TotallyFree
CPack: Create package
 CPackRPM: Will use GENERATED spec file: <...>/build/_CPack_Packages/Linux/RPM/SPECS/totallyfree.spec
CPack: - package: <...>/build/TotallyFree-0.1.0-Linux.rpm generated.
$
```

CPack running steps IV

make package example

```
$ make package
[ 33%] Built target acrodict
[ 66%] Built target Acrodictlibre
[100%] Built target Acrolibre
Run CPack packaging tool...
CPack: Create package using TGZ
CPack: Install projects
CPack: - Run preinstall target for: TotallyFree
CPack: - Install project: TotallyFree
CPack: Create package
CPack: - package: <...>/build/TotallyFree-0.1.0-Linux.tar.gz generated.
```

Rebuild project

In the `make package` case CMake is checking that the project does not need a rebuild.

CPack running steps V

make package_source example

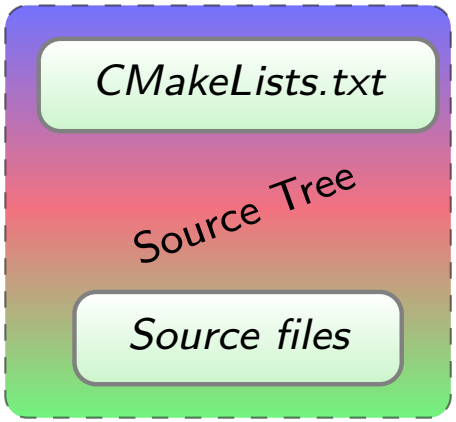
```
$ make package_source
make package_source
Run CPack packaging tool for source...
CPack: Create package using TGZ
CPack: Install projects
CPack: - Install directory: <...>/totally-free
CPack: Create package
CPack: - package: <...>/build/TotallyFree-0.1.0-Source.tar.gz generated.
CPack: Create package using TBZ2
CPack: Install projects
CPack: - Install directory: <...>/totally-free
CPack: Create package
CPack: - package: <...>/build/TotallyFree-0.1.0-Source.tar.bz2 generated.
CPack: Create package using TZ
CPack: Install projects
CPack: - Install directory: <...>/totally-free
CPack: Create package
CPack: - package: <...>/build/TotallyFree-0.1.0-Source.tar.Z generated.
```

The CPack workflow (pictured)

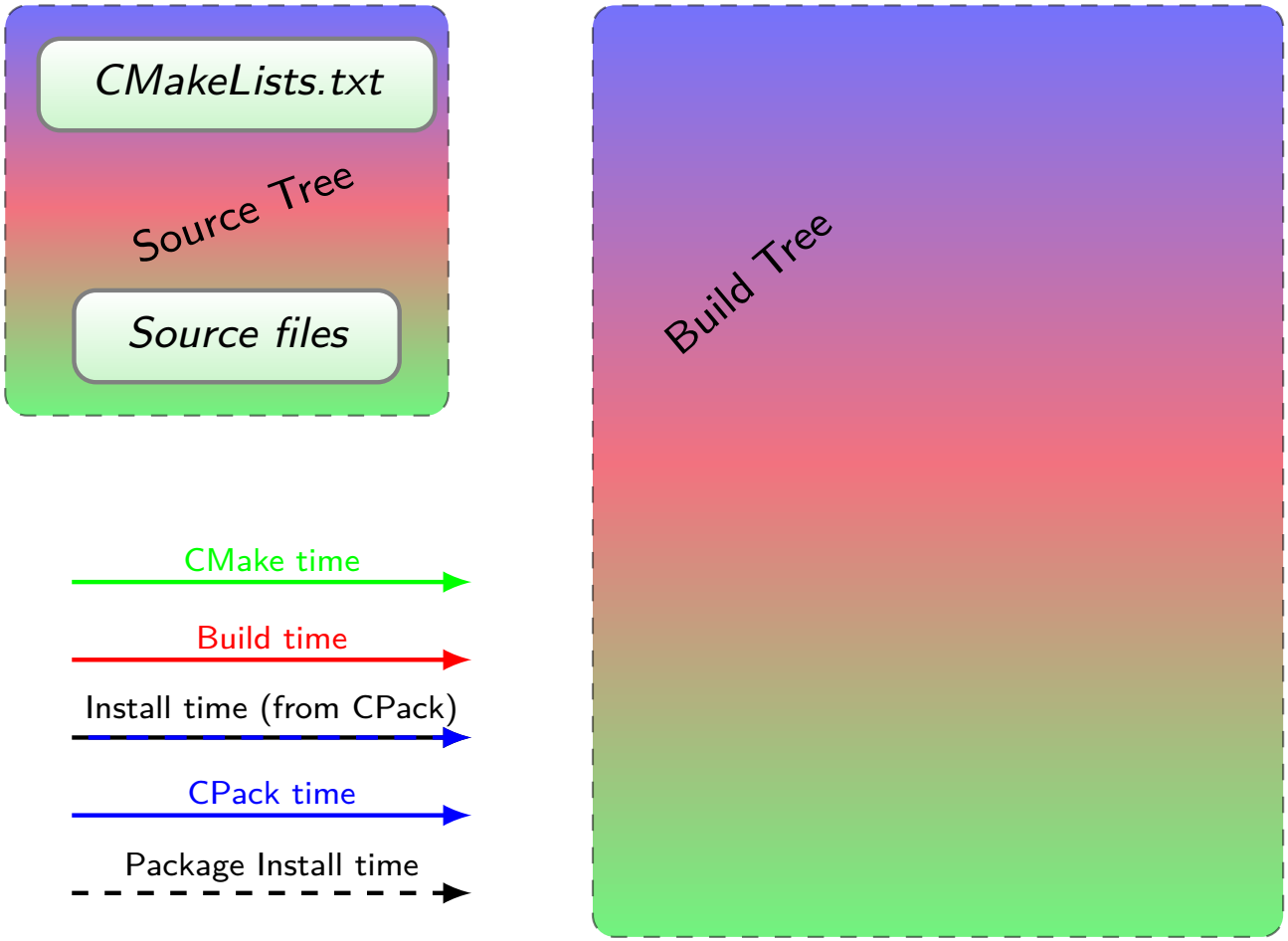
CMakeLists.txt

Source files

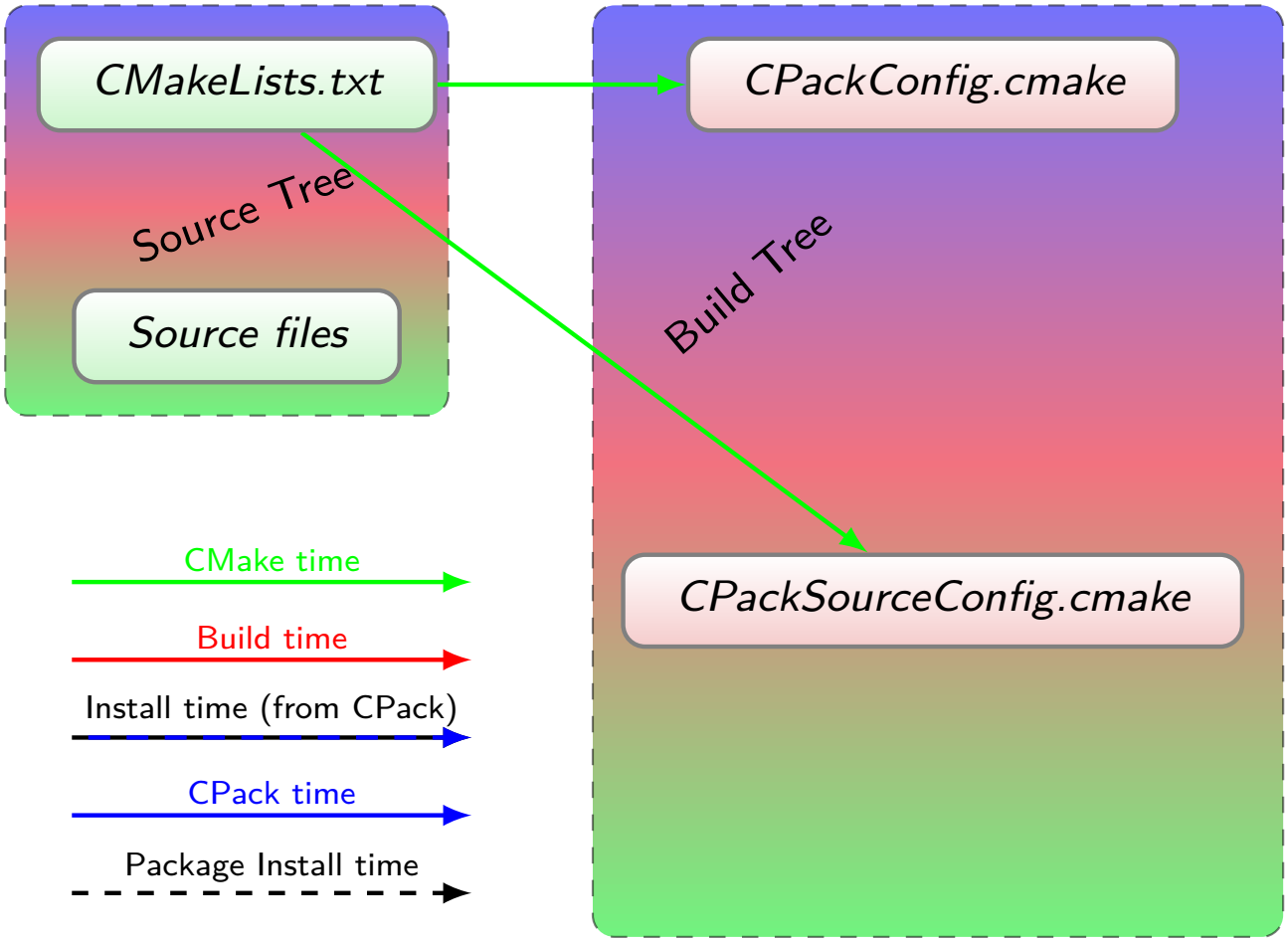
The CPack workflow (pictured)



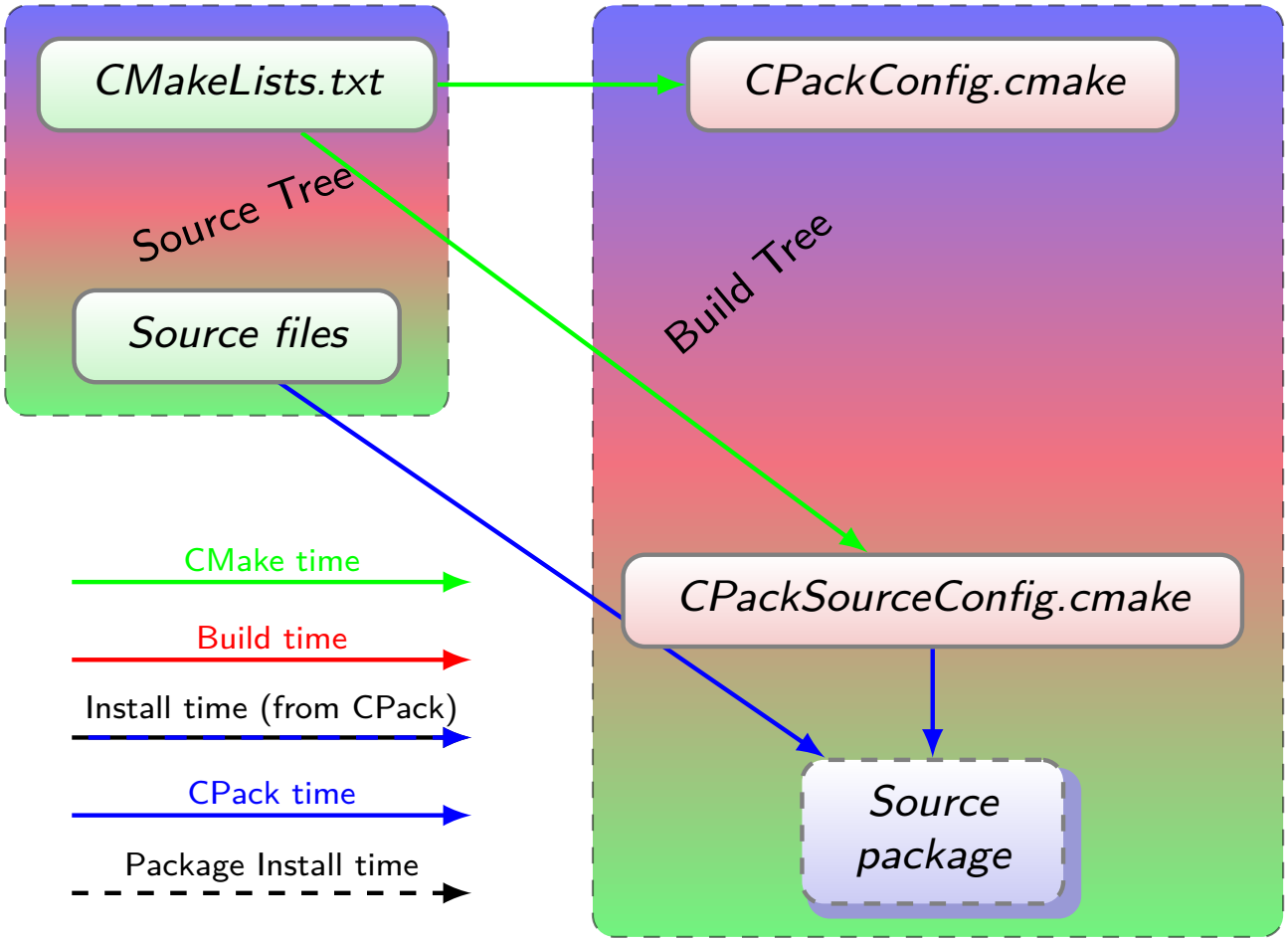
The CPack workflow (pictured)



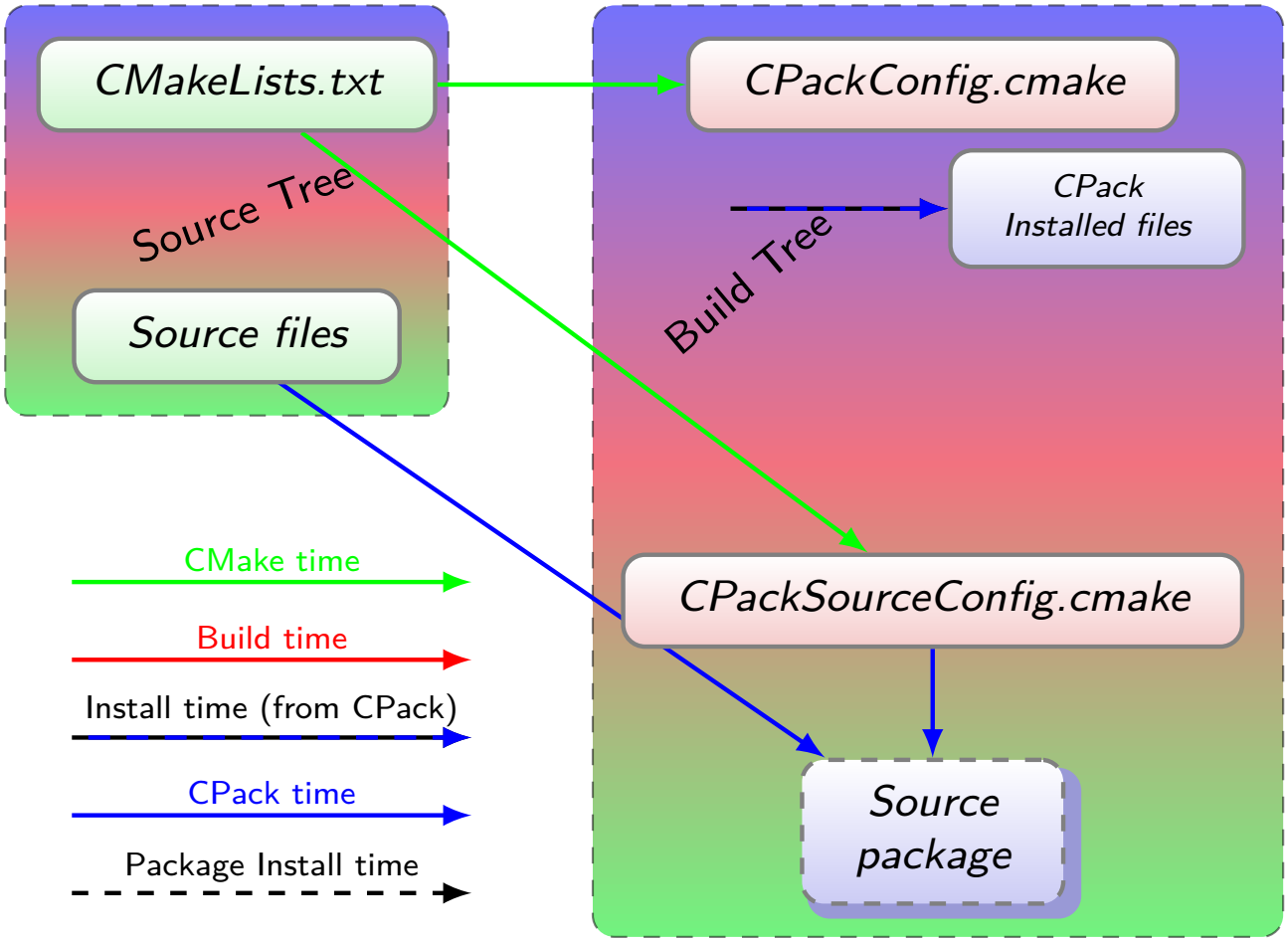
The CPack workflow (pictured)



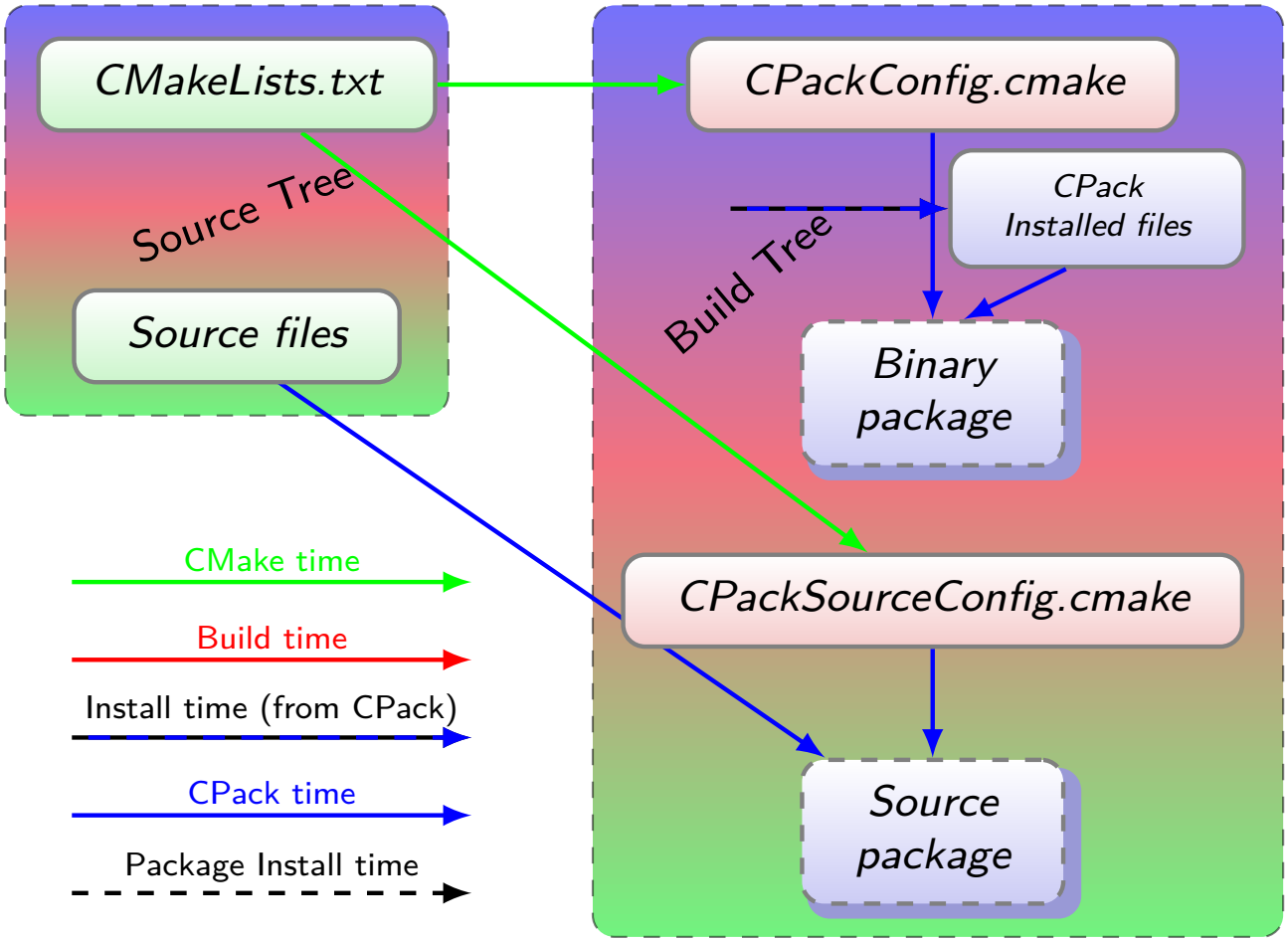
The CPack workflow (pictured)



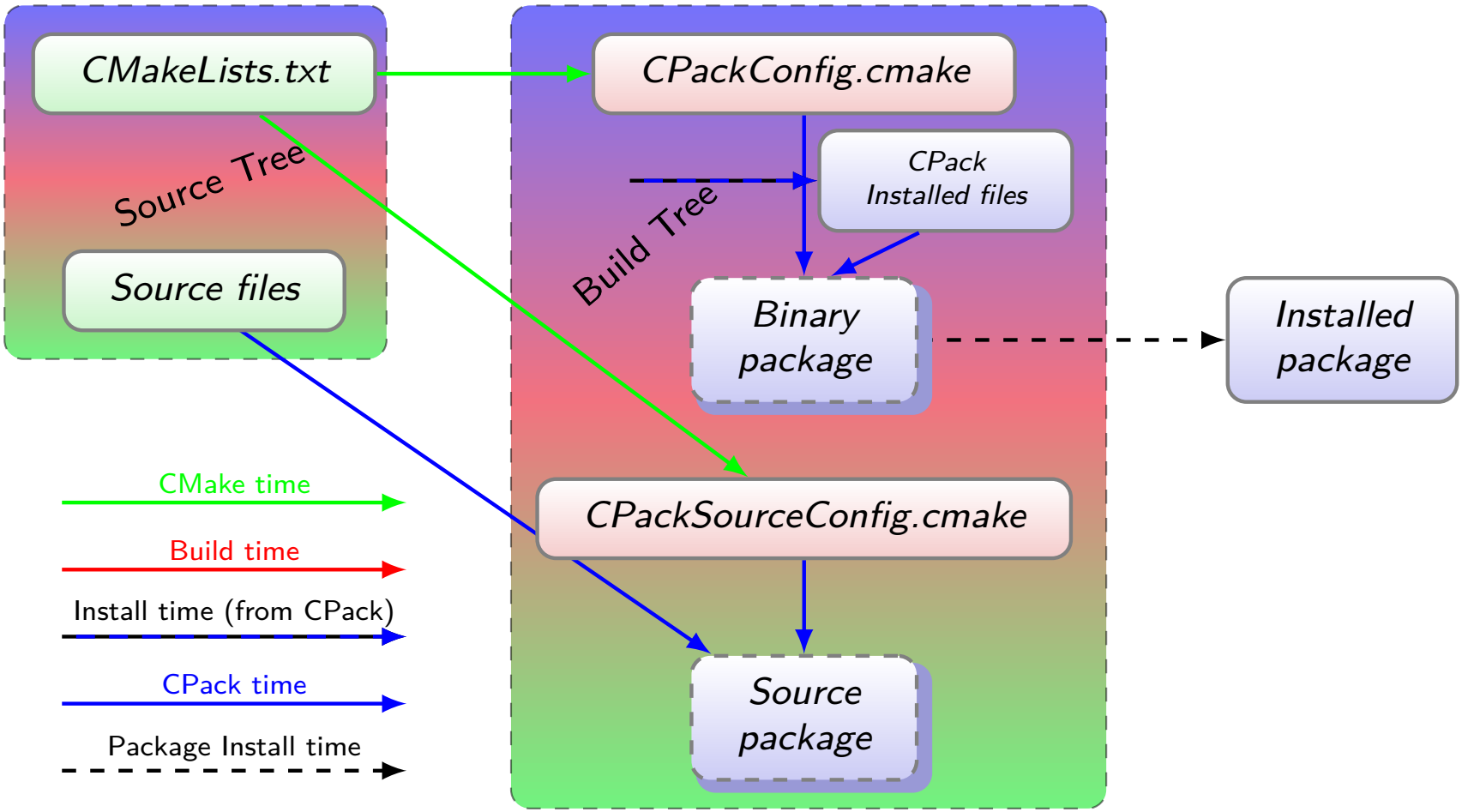
The CPack workflow (pictured)



The CPack workflow (pictured)



The CPack workflow (pictured)



Source vs Binary Generators

CPack does not really distinguish “source” from “binaries”!!

CPack source package

The CPack configuration file is: `CPackSourceConfig.cmake`. The CPack source generator is essentially packaging directories with install, exclude and include rules.

CPack binary package

The CPack configuration file is: `CPackConfig.cmake`. Moreover CPack knows that a project is built with CMake and inherits many properties from the install rules found in the project.

Outline

Part I: CMake Part II: CPack

Archive Generators

A family of generators

The archive generators is a family of generators which is supported on all CMake supported platforms through `libarchive`: http://code.google.com/p/libarchive/.

STGZ Self extracting Tar GZip compression

TBZ2 Tar BZip2 compression

TGZ Tar GZip compression

TZ Tar Compress compression

TXZ Tar XZ compression

7Z 7-zip archive

ZIP Zip archive

Linux-friendly generators I

- Tar-kind archive generators
- Binary RPM: only needs `rpmbuild` to work.
- Binary DEB: works on any Linux distros.
- IFW: Qt Installer framework

CPack vs native tools

One could argue "why use CPack for building .deb or .rpm". The primary target of CPack RPM and DEB generators are people who are NOT professional packagers. Those people can get a clean package without too much effort and get a better package than a bare TAR archive.

No official packaging replacement

Those generators are no replacement for official packaging tools.

Windows-friendly generators

- Zip archive generator
- NullSoft System Installer generator:
  http://nsis.sourceforge.net/
  Supports component installation, produces nice GUI installer.
- WiX installer: http://wixtoolset.org/
  Windows Installer XML which produces MSI.
- IFW: Qt Installer framework
- Cygwin: Binary and Source generators.

Mac OS-friendly generators

- Tar-kind archive generators
- DragNDrop
- PackageMaker
- Bundle
- OSXX11
- may be Qt IFW as well...

Don't ask me

I'm not a Mac OS user and I don't know them. Go and read the CPack doc or ask on the ML. https://cmake.org/mailing-lists/

Packaging Components I

CMake+CPack installation components?

Sometimes you want to split the installer into *components*.

1. Use `COMPONENT` argument in your install rules (in the CMakeLists.txt),
2. Add some more [CPack] information about how to group components,
3. Choose a component-aware CPack generator
4. Choose the behavior (1 package file per component, 1 package file per group, etc...)
5. Possibly specify generator specific behavior in `CPACK_PROJECT_CONFIG_FILE`
6. Run CPack.

Packaging Components II

More detailed documentation here:

https://cmake.org/Wiki/CMake:Component_Install_With_CPack

Component aware generator

- Not all generators do support components (i.e. they are `MONOLITHIC`)
- Some produce a single package file containing all components. (e.g. NSIS, WiX, Qt IFW)
- Others produce several package files containing one or several components. (e.g. ArchiveGenerator, RPM, DEB)

Outline

Part I: CMake Part II: CPack Part III: CTest and CDash

Outline

Part I: CMake Part II: CPack Part III: CTest and CDash

More to come on CTest/CDash

Sorry...out of time!!

CMake and its friends are so much fun and powerful that I ran out of time to reach a detailed presentation of CTest/CDash, stay tuned for next time...

In the meantime:

- Go there: `http://www.cdash.org`
- Open your own (free) Dashboard: `http://my.cdash.org/`

Outline

Part I: CMake Part II: CPack Part III: CTest and CDash

References I

[?, ?, ?, ?, ?, ?, ?]