GNU Make简介及应用

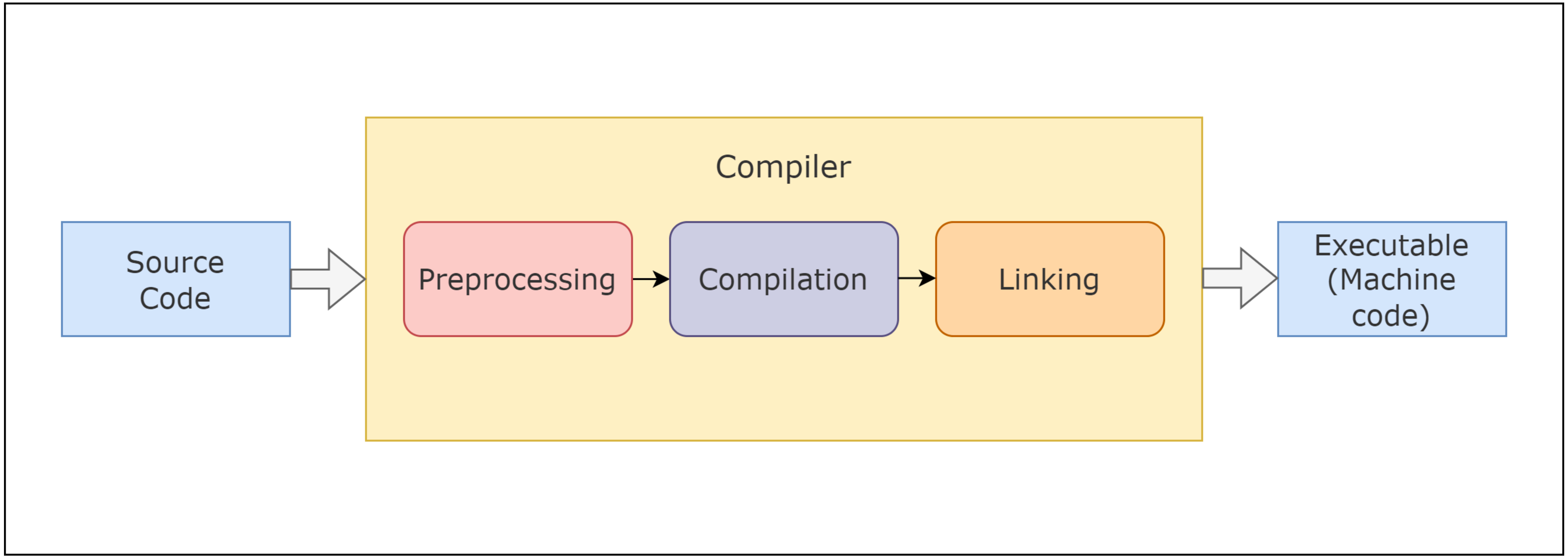
索兵兵

西北大学现代物理研究所

目录

1.GNU Make系统简介

编译软件流程



1.GNU Make系统简介

假设一个项目包含
main.c, matmulslow.c, matmulfast.c, matmul.h

其中main.c需要调用matmulslow.c和matmulfast.c的中函数, matmulslow.c, matmulfast.c依赖于头文件matmul.h

编译、链接要执行的命令为

```
$ gcc matmulslow.c -c

$ gcc matmulfast.c -c

$ gcc main.c -c

$ gcc main.o matmulslow.o matmulfast.o -o myexec.x
```

1.GNU Make系统简介

- Why we need makefile?
  - 项目有多个源文件
  - 源文件的依赖关系复杂
  - 多个开发者协同开发
  - 对部分文件做了更改，需要快速重新编译

目录

2.GNU Make的使用

一个简单的Makefile

```
# This is a simple makefile

release: main.o matmulslow.o matmulfast.o
	gcc matmulslow.o matmulfast.o main.o -o myexec.x
clean:
	rm matmulslow.o matmulfast.o main.o


main.o: main.c
	cc main.c -c
matmulslow.o: matmulslow.c matmul.h
	cc matmulslow.c -c
matmulfast.o: matmulfast.c matmul.h
	cc matmulfast.c -c
```

注释行

规则(rule target)

目标(target)

依赖文件(depend)

注意：这里是个tab，不是空格

编译工程文件

$ make release

或

$ make

删除obj文件

$ make clean

2.GNU Make的使用

```
…

main.o: main.c

    cc main.c -c

matmulslow.o: matmulslow.c matmul.h

    cc matmulSlow.c -c

matmulfast.o: matmulfast.c matmul.h

    cc matmulfast.c -c
```

*.o文件默认依赖于同名的.c文件

```
…

main.o:

    cc main.c -c

matmulslow.o: matmul.h

    cc matmulslow.c -c

matmulfast.o: matmul.h

    cc matmulfast.c -c
```

2.GNU Make的使用

在Makefile中使用变量

```
# This is a simple makefile
objects = main.o matmulslow.o \
          matmulfast.o
release: $(objects)
    gcc matmulslow.o matmulfast.o main.o -o myexec.x
clean:
    rm $(objects)
main.o:
    cc main.c -c
matmulslow.o: matmul.h
    cc matmulslow.c -c
matmulfast.o: matmul.h
    cc matmulfast.c -c
```

续行符

引用变量objects

定义变量objects

2.GNU Make的使用

Makefile中的内容

1. Explicit rule: 又叫rule target

```
rulename: prerequisites
    Recipe1
    Recipe2
```

2. Implicit rule: 系统根据默认预设规则来处理的rule target，例如:

```
foo: foo.o bar.o
        cc -o foo foo.o bar.o $(CFLAGS) $(LDFLAGS)
```

3. Variable : 即变量

2.GNU Make的使用

4. Directives : make处理时执行一些特殊指令的行

   包含其他Makefile文件，如 include filename

   条件处理，如 if … else..

   定义变量，如用define定义多行变量

5. #开始的为注释行

2.GNU Make的使用

编写Makefile中rule target

简单的rule target

```
foo.o: foo.c defs.h

    cc foo.o -c -g
```

foo.c依赖于defs.h, 如果defs.h被更新，执行make命令将编译foo.o,所有依赖于foo.o的其他rule target将被更新

Phony target

```
.PHONY: clean

clean

    Rm *.o
```

Phony target用来定义用make命令执行时的rule target，主要目的是为了避免和文件名重复和提高make的性能

2.GNU Make的使用

Suffix rules

```
.SUFFIXES: .F90 .f90 .F .src
%.o: %.F90 $(FMODS)
	$(FC) $(F90FLAGS) $(IFLAGS) $(FDFLAGS) $(MFLAGS)  -c $<

%.o: %.f90 $(FMODS)
	$(FC) $(F90FLAGS) $(IFLAGS) $(FDFLAGS) $(MFLAGS)  -c $<

%.o: %.F $(FMODS)
	$(FC) $(F77FLAGS) $(IFLAGS) $(FDFLAGS) $(MFLAGS)  -c $<

%.o: %.f $(FMODS)
	$(FC) $(F77FLAGS) $(IFLAGS) $(FDFLAGS) $(MFLAGS)  -c $<

%.F: %.src
	@$(SED) 's/CI$(intlen)/    /g' $< > $@
```

2.GNU Make的使用

Using conditional rules

常用的条件语句有：ifeq, ifneq, ifdef, ifndef

```
libs_for_gcc = -lgnu
normal_libs =


foo: $(objects)
ifeq ($(CC),gcc)
      $(CC) -o foo $(objects) $(libs_for_gcc)
else
      $(CC) -o foo $(objects) $(normal_libs)
endif
```

```
conditional-directive
text-if-true
endif
```

```
conditional-directive
text-if-true
else conditional-directive
text-if-true
else  conditional-directive
text-if-true
endif
```

2.GNU Make的使用

Functions in Makefile

Makefile中的函数用来处理文本，使用格式为

$(function arguments)

字符串分析与处理函数

$(subst from,to,text)

$(patsubst pattern,replacement,text)

$(strip string)

$(findstring find,in)

$(filter pattern…,text)

$(filter-out pattern…,text)

$(sort list)

$(words text)

$(lastword names…)

文件名相关函数

$(dir names)

$(notdir names)

$(suffix names)

$(basename names)

$(addsuffix suffix,names)

$(addprefix prefix,names)

$(join lists,list2)

$(wildcard partten)

$(realpath names)

$(abspath names)

条件判断函数

$(if condition,then part, else part)

$(or condition1,condition2,…)

$(and condition1,condition2,…)

$(intcmp lhs,rhs[,lt-part[,eq-part[,gt-part]]])

其它函数可参阅Makefile手册

```
VERSION  =1.00
CC  =gcc
DEBUG  =-DUSE_DEBUG
CFLAGS =-Wall
SOURCES  =$(wildcard ./source/*.c)
INCLUDES  =-I./include
LIB_NAMES  =-lfun_a -lfun_so
LIB_PATH  =-L./lib
OBJ  =$(patsubst %.c, %.o, $(SOURCES))
TARGET  =app

#links
$(TARGET):$(OBJ)
 @mkdir -p output
 $(CC) $(OBJ) $(LIB_PATH) $(LIB_NAMES) -o output/$(TARGET)$(VERSION)
 @rm -rf $(OBJ)

#compile
%.o: %.c
 $(CC) $(INCLUDES) $(DEBUG) -c $(CFLAGS) $< -o $@

.PHONY:clean
clean:
 @echo "Remove linked and compiled files......"
 rm -rf $(OBJ) $(TARGET) output
```

```
VERSION     =
CC          =gcc
DEBUG    =
CFLAGS   =-Wall
AR    =ar
ARFLAGS     =rv
SOURCES   =$(wildcard *.c)
INCLUDES    =-I.
LIB_NAMES   =
LIB_PATH  =
OBJ         =$(patsubst %.c, %.o, $(SOURCES))
TARGET      =libfun_a

#link
$(TARGET):$(OBJ)
 @mkdir -p output
 $(AR) $(ARFLAGS) output/$(TARGET)$(VERSION).c
 @rm -rf $(OBJ)

#compile
%.o: %.c
 $(CC) $(INCLUDES) $(DEBUG) -c $(CFLAGS) $< -c

.PHONY:clean
clean:
 @echo "Remove linked and compiled files......
 rm -rf $(OBJ) $(TARGET) output
```

```
VERSION    =
CC         =gcc
DEBUG      =
CFLAGS     =-fPIC -shared
LFLAGS    =-fPIC -shared
SOURCES    =$(wildcard *.c)
INCLUDES   =-I.
LIB_NAMES  =
LIB_PATH   =
OBJ        =$(patsubst %.c, %.o, $(SOURCES))
TARGET     =libfun_so

#link
$(TARGET):$(OBJ)
 @mkdir -p output
  $(CC)  $(OBJ)  $(LIB_PATH)  $(LIB_NAMES)  $(LFLAGS)  -
o output/$(TARGET)$(VERSION).so
 @rm -rf $(OBJ)

#compile
%.o: %.c
$(CC) $(INCLUDES) $(DEBUG) -c $(CFLAGS) $< -o $@

.PHONY:clean
clean:
 @echo "Remove linked and compiled files......"
 rm -rf $(OBJ) $(TARGET) output
```

```
VERSION   =
CC        =gcc
DEBUG     =
CFLAGS    =-fPIC -shared
LFLAGS   =-fPIC -shared
SOURCES   =$(wildcard *.c)
INCLUDES  =-I.
LIB_NAMES =
LIB_PATH  =
OBJ       =$(patsubst %.c, %.o, $(SOURCES))
TARGET    =libfun_so

#link
$(TARGET):$(OBJ)
 @mkdir -p output
  $(CC)  $(OBJ)  $(LIB_PATH)  $(LIB_NAMES)  $(LFLAGS)  -o  output/$(TARGET)$
(VERSION).so
 @rm -rf $(OBJ)

#compile
%.o: %.c
 $(CC) $(INCLUDES) $(DEBUG) -c $(CFLAGS) $< -o $@

.PHONY:clean
clean:
 @echo "Remove linked and compiled files......"
 rm -rf $(OBJ) $(TARGET) output
```

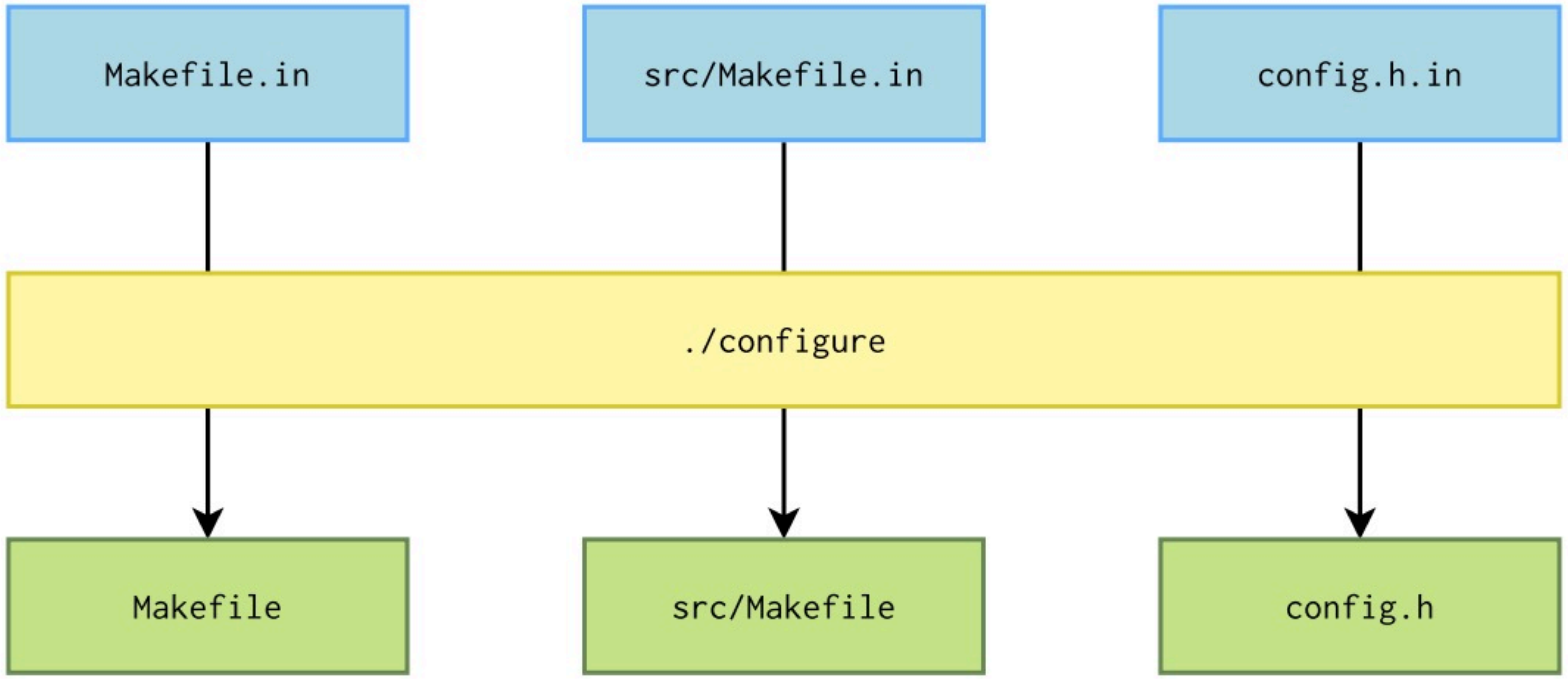
目录

3.GNU autotools和CMake

GNU autotools

Linux用户典型的编译软件命令

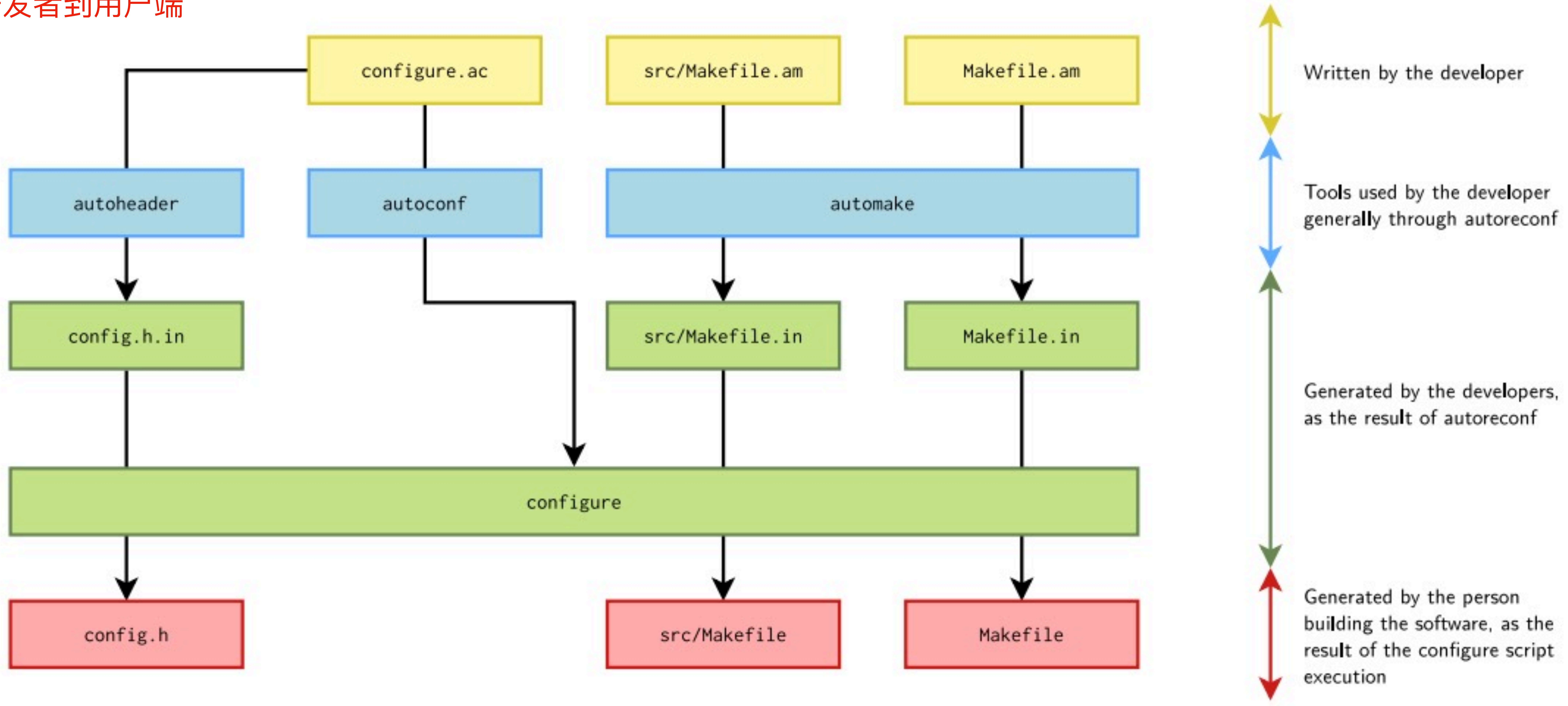
```
$ ./configure
$ make
$ make install
```



3.GNU autotools和CMake

GNU autotools

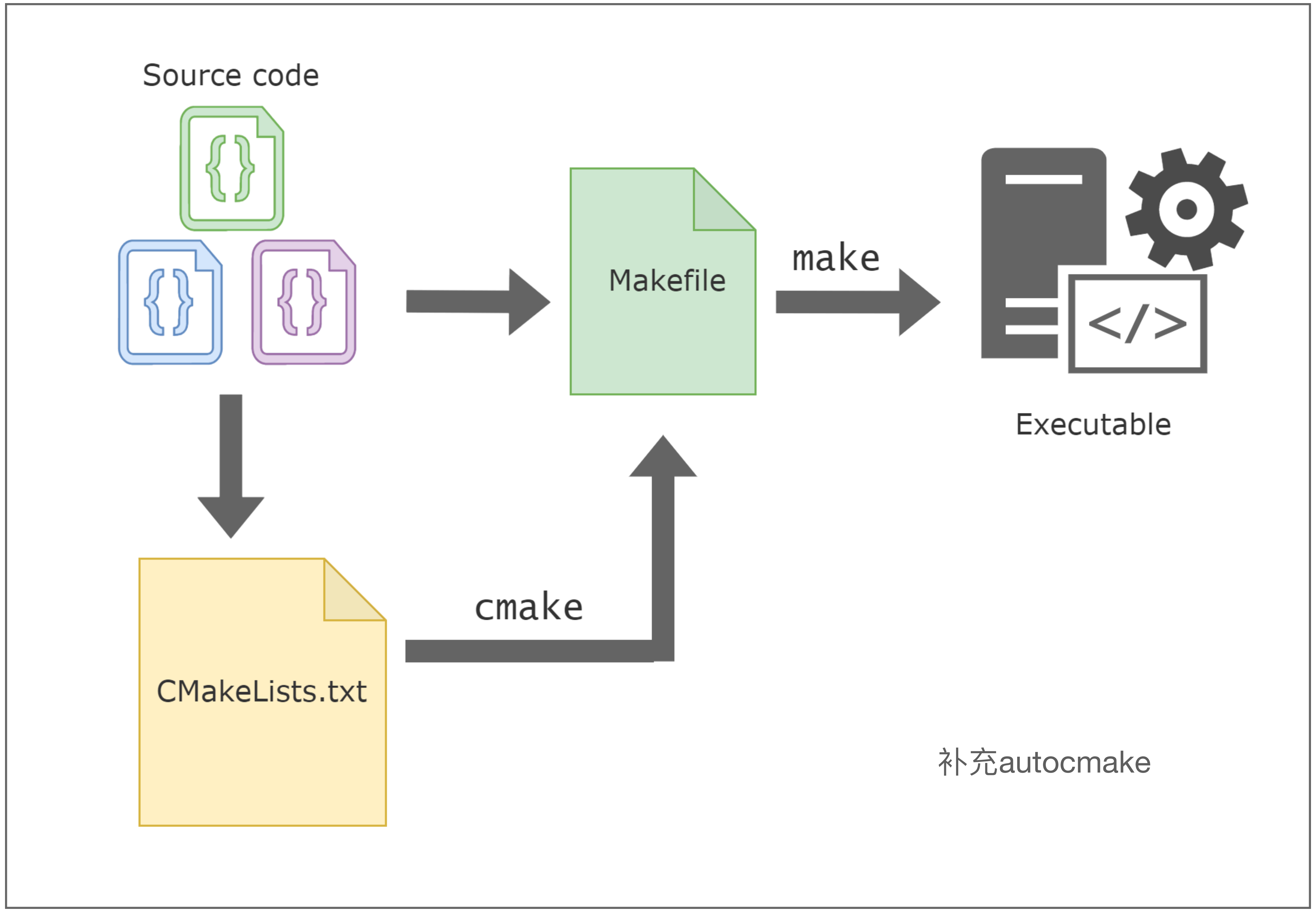
从开发者到用户端



configure.ac
src/Makefile.am
Makefile.am
autoheader
autoconf
automake
config.h.in
src/Makefile.in
Makefile.in
configure
config.h
src/Makefile
Makefile
Written by the developer
Tools used by the developer generally through autoreconf
Generated by the developers, as the result of autoreconf
Generated by the person building the software, as the result of the configure script execution

3.GNU autotools和CMake

Cmake V.S. Make



1. Cmake的核心文件是CMakeLists.txt
2. Cmake可以跨平台使用
3. Cmake并不独立编译工程，而是依赖于Make工作

目录

4.学习资料及建议

GNU Make手册

https://www.gnu.org/software/make/manual/make.html

中文手册

https://file.elecfans.com/web1/M00/7D/E7/o4YBAFwQthSADYCWAAT9Q1w_4U0711.pdf

CMake手册
https://cmake.org/documentation/

学习建议：

1 学会写简单的Makefile

2 如果做大项目开发，应学习CMake