[Software Development]

Autotools

Davide Balzarotti

Eurecom – Sophia Antipolis, France

IMPORTANT. The deadline for the homework is

January, Monday 18 at 9:00 AM

Homework Status

- 114 registered students
  - 92% completed at least one challenge
  - 73 command line ninjas
  - 55 python masters
  - 5 dev fu
- 2463 Submissions
  - 23% (--) of which were correct

Software Development Tools

1. Configuring and Building the program
   - ~~GCC~~
   - ~~Makefiles~~
   - Autotools
2. Packaging and Distributing the application
3. Debugging and Profiling

So far ...

- We learned how to use GCC to manually compile programs and libraries
- We learned how to use makefiles to define all the steps and the dependencies required to automatically build more complex projects

… But Unfortunately

- Not all systems are exactly the same
  - C libraries can be slightly different in different OSs
  - Tools (compilers, sed, tar, ..) can have different names or parameters
  - Files and libraries can be located in different places

- This would require to..
  - … change the source code to deal with different C functions
  - … change the makefiles to deal with different tools and their options

Example

- C functions...
  - may not exist everywhere (e.g., `strtod()`)
  - may have different names (e.g., `strchr()` vs. `index()`)
  - may have varying prototypes
    (e.g., `int setpgrp(void)` vs. `int setpgrp(int, int)`)
  - may have a different behavior (e.g., `malloc(0)`)
- And also when the function is the same...
  - it may be located in different libraries
    (is `pow()` in `libm.so` or in `libc.so`?)
  - it may be defined in a different header file
    (`string.h` vs. `strings.h` vs. `memory.h`)

Dealing with Portability

- Distributing a software that has to run on a wide variety of Unix variants requires the developer to be familiar with the detailed differences between the variants
  - It is possible to use a number of `#define` and `#ifdef`, but maintaining them by hand for each system is cumbersome
  - It is possible to ask the user to edit the necessary `-L`, `-I`, and `-l` compilation options in the Makefile, but it is burdensome

Dealing with Portability

- Distributing a software that has to run on a wide variety of Unix variants requires the developer to be familiar with the detailed differences between the variants
  - It is possible to use a number of `#define` and `#ifdef`, but maintaining them by hand for each system is cumbersome
  - It is possible to ask the user to edit the necessary `-L`, `-I`, and `-l` compilation options in the Makefile, but it is burdensome

- In 1991, David J. MacKenzie got tired of customizing his project Makefile for the 20 platforms he had to deal with
  - To solve the problem, he decided to write a little shell script called `configure` to automatically adjust the Makefile
  - Today this process has been standardized by the GNU project

Installing from Sources: the GNU Build System

- Step 1: Unpacking
  - Software is usually distributed in tarball format
    - Not compressed (`.tar`)
    - Compressed with gzip (`.tgz` or `.tar.gz`)
    - Compressed with bzip2 (`.tbz`, `.tb2`, or `.tar.bz2`)
  - First, the content of the package must be extracted

    ```
    > tar xvzf package.tgz
    ```

    (or `tar xvjf package.bz2`)
  - What's inside the package?
    - Source code
    - Documentation
    - GNU-style specific files (NEWS, README, AUTHORS, ChangeLog)
    - GNU-style generic files (INSTALL, COPYING)
    - Configuration script: configure

Installing from Sources: the GNU Build System

- Step 2: Configure the package
  - Each GNU software package contains a script to configure the building system
  - The configure script tests the system features, check the required dependencies and create the makefiles

```
> ./configure
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /bin/mkdir -p
checking for gawk... no
checking for mawk... mawk
checking whether make sets $(MAKE)... yes
checking for gcc... gcc
checking for C compiler default output file name... a.out
checking whether the C compiler works... yes
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
configure: creating ./config.status
config.status: creating Makefile
config.status: creating src/Makefile
config.status: creating config.h
config.status: config.h is unchanged
```

Installing from Sources: the GNU Build System

- Step 3: Building
  - Compile the program by executing the instruction in the makefiles

    ```
    > make
    ```

  - This creates the binaries but leave them in the current directory (or, more likely, in a sub-directory)

- Step 4: Installing
  - Copy the binary in a system directory (usually `/usr/local/bin/`)
  - This step requires to be root

    ```
    > sudo make install
    ```

- Unistalling:
  - `> sudo make uninstall`

Part I

Autotools Overview

GNU Autotools

- Autotools is a *suite* of programming tools produced by the GNU project, designed to assist the developer in making source code packages portable to many Unix-like systems
- It helps developers to prepare a software distribution that can be easily compiled and installed by the user with the *configure* → *make* → *make install* sequence

- Consider using Autotools if:
  - You are developing a C/C++ project
  - You are distributing the source code
  - You cannot predict the environment (operating system and/or hardware platform) that your target audience will be using

The tools set

aclocal - Generates local macros and gather them into `aclocal.m4`

autoheader - Creates a template of `#define` statements in `config.h.in` to be used by *configure* that will define platform constants and other similar things

automake - Generates `Makefile.in` from `aclocal.m4`, `configure.ac` and `Makefile.am`.

autoconf - Generates `configure` from `aclocal.m4` and `configure.ac`.

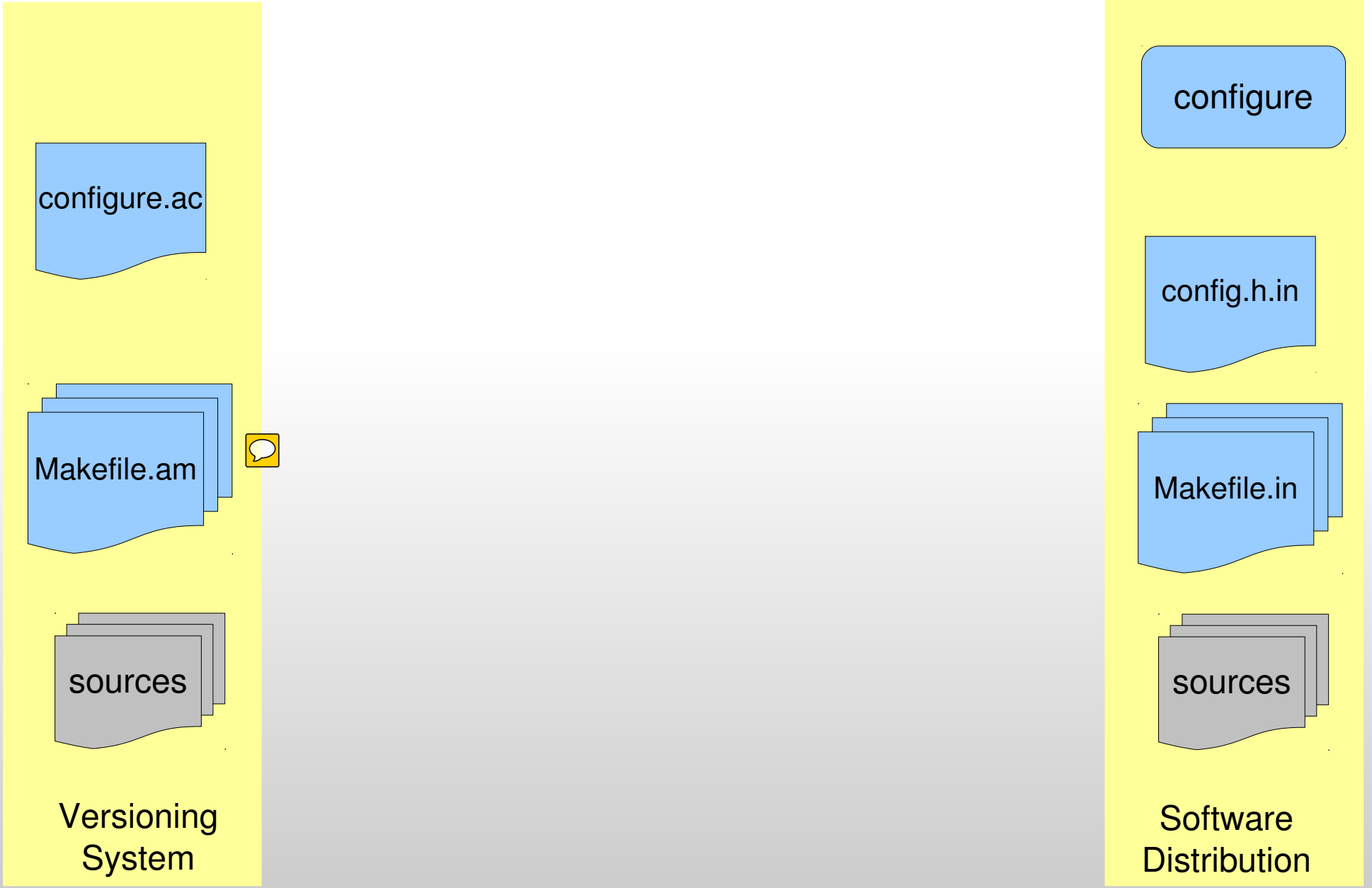
configure - script to configure the build for the local machine. Generates `Makefile` from `Makefile.in` and `config.h.in`.

libtool - Simplifies the inclusion of dynamic libraries depending upon the platform.

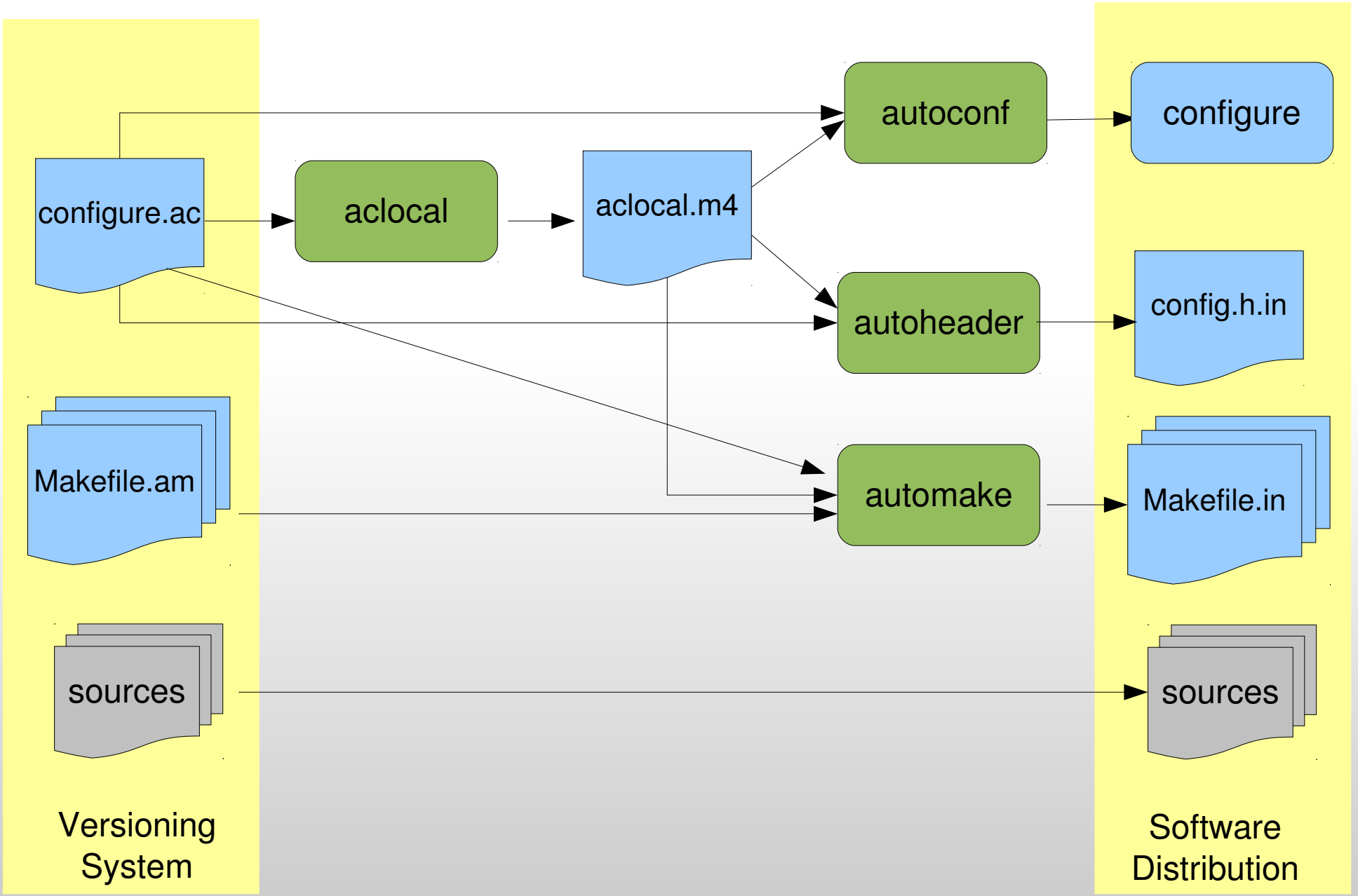
autoreconf - Run all tools in the right order

autoscan Scan sources for common portability problems, and related macros missing from `configure.ac`.

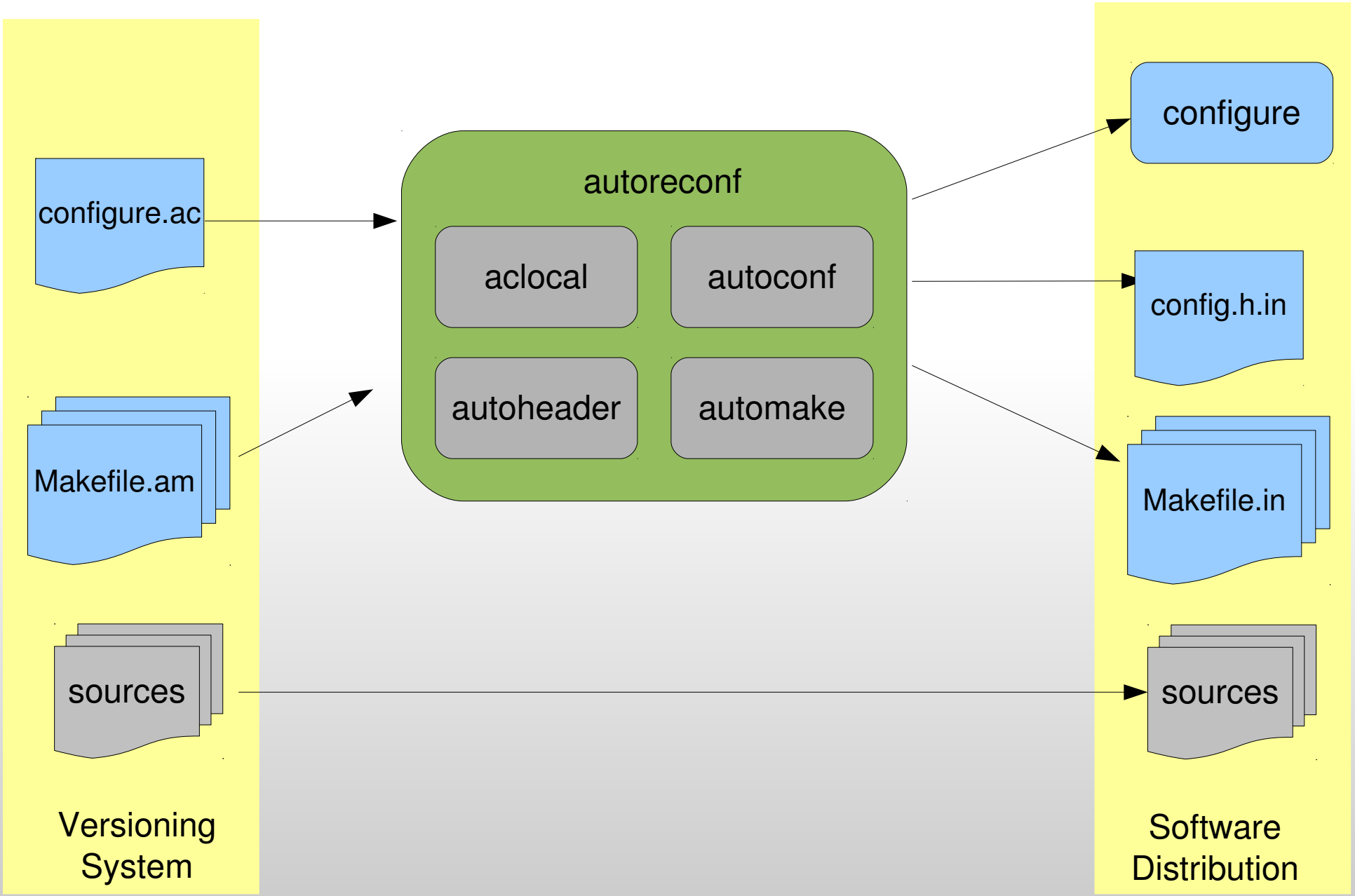
Developer's View



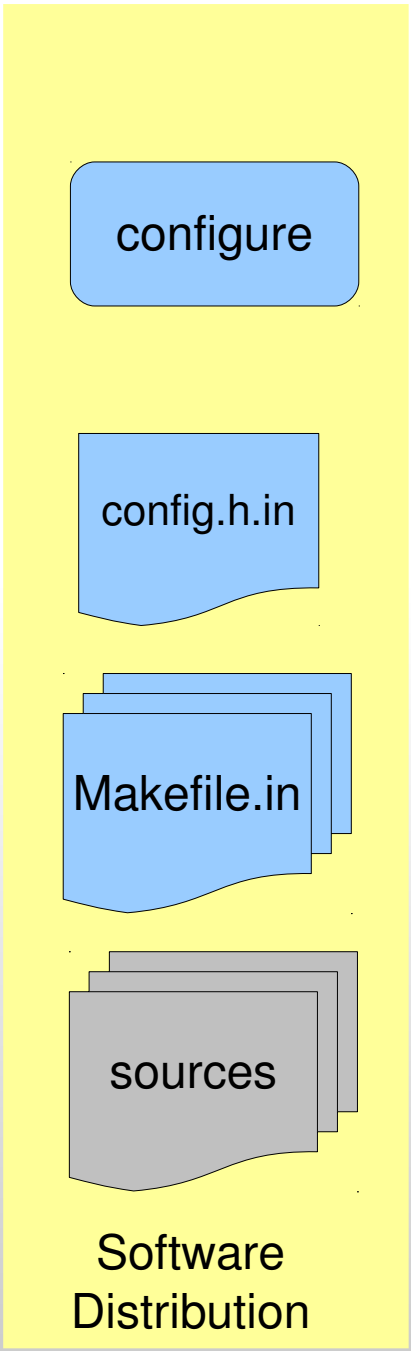
Developer's View



Developer's View

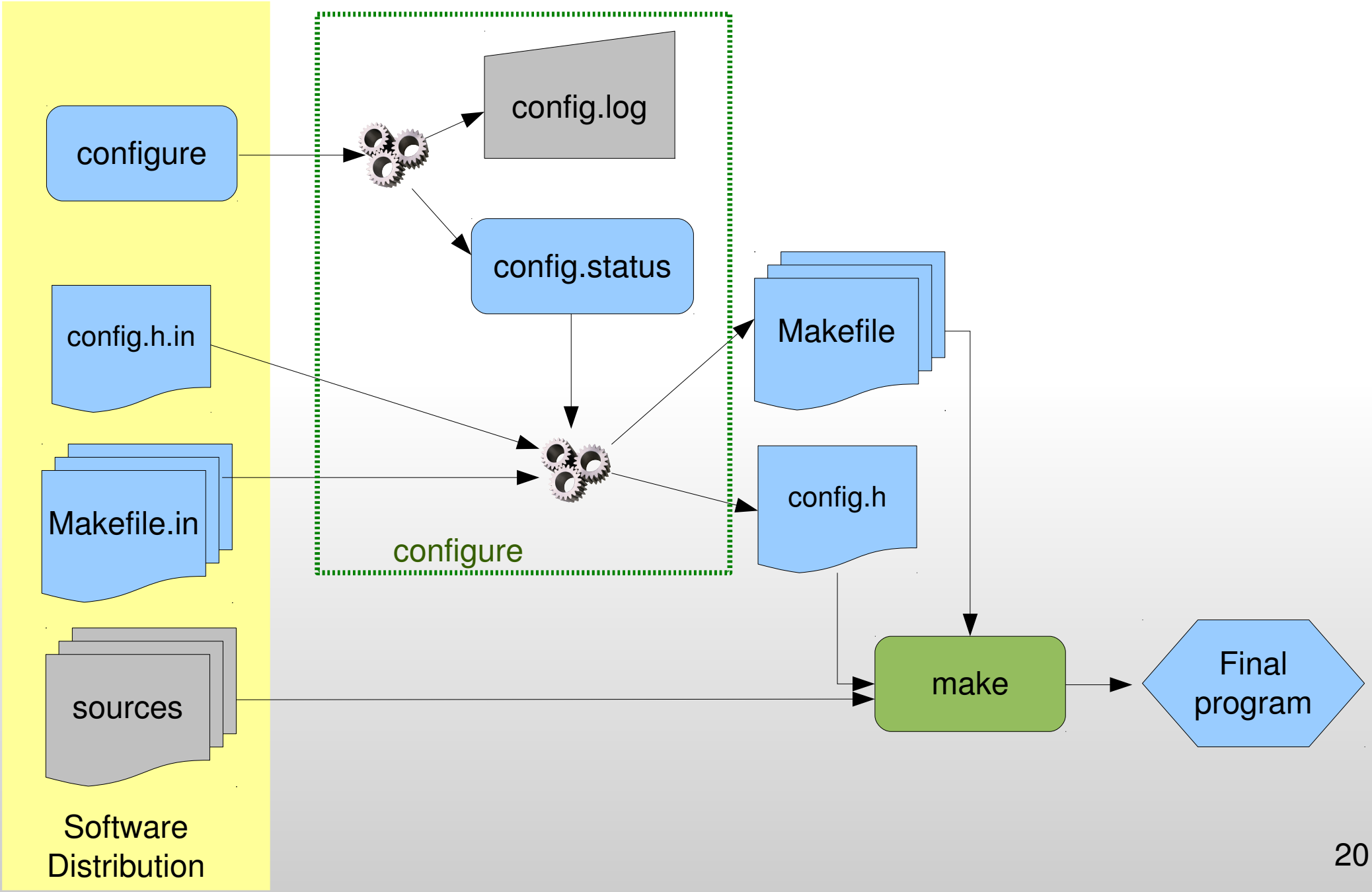


User's View

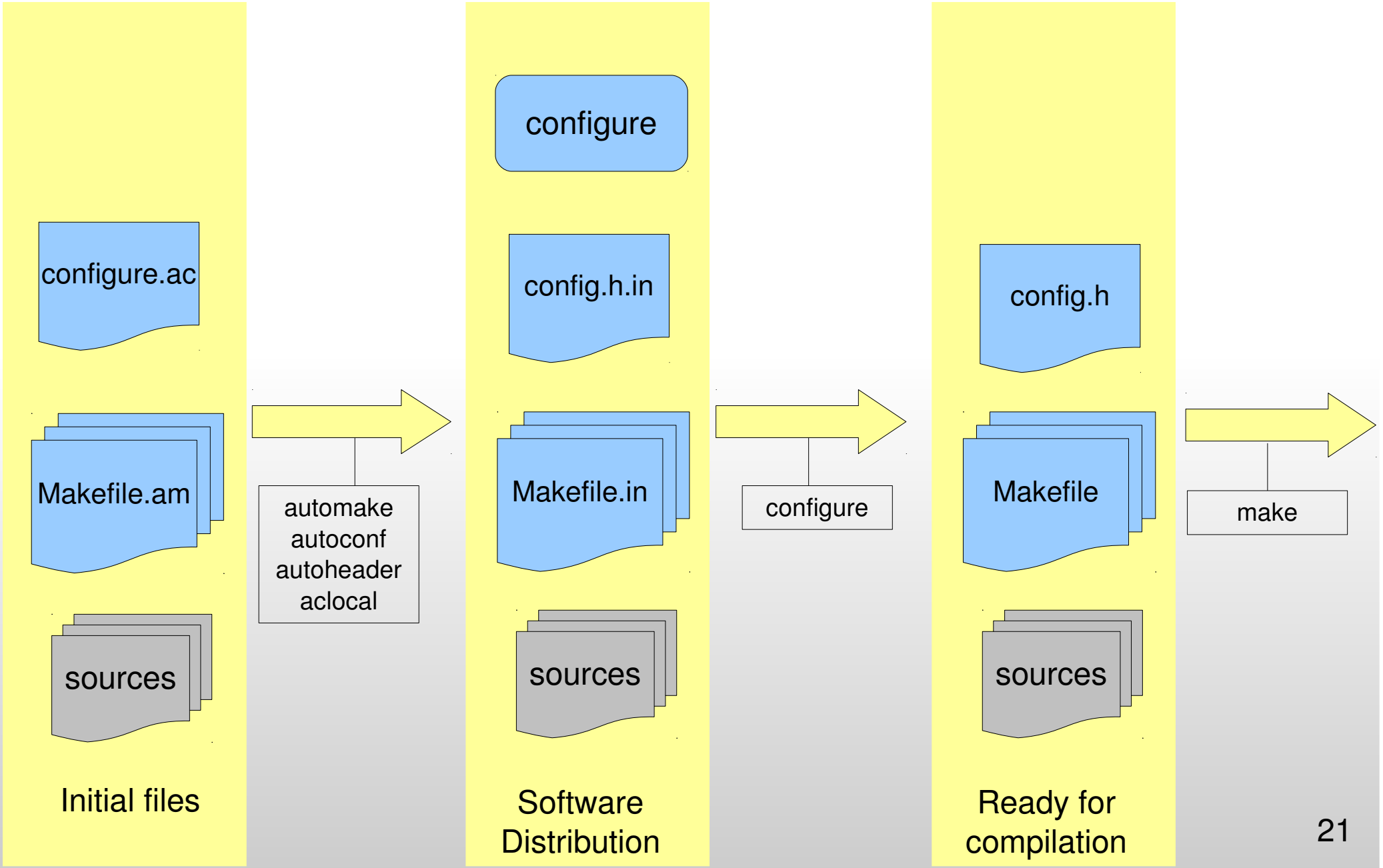

configure
config.h.in
Makefile.in
sources
Software
Distribution


Final
program

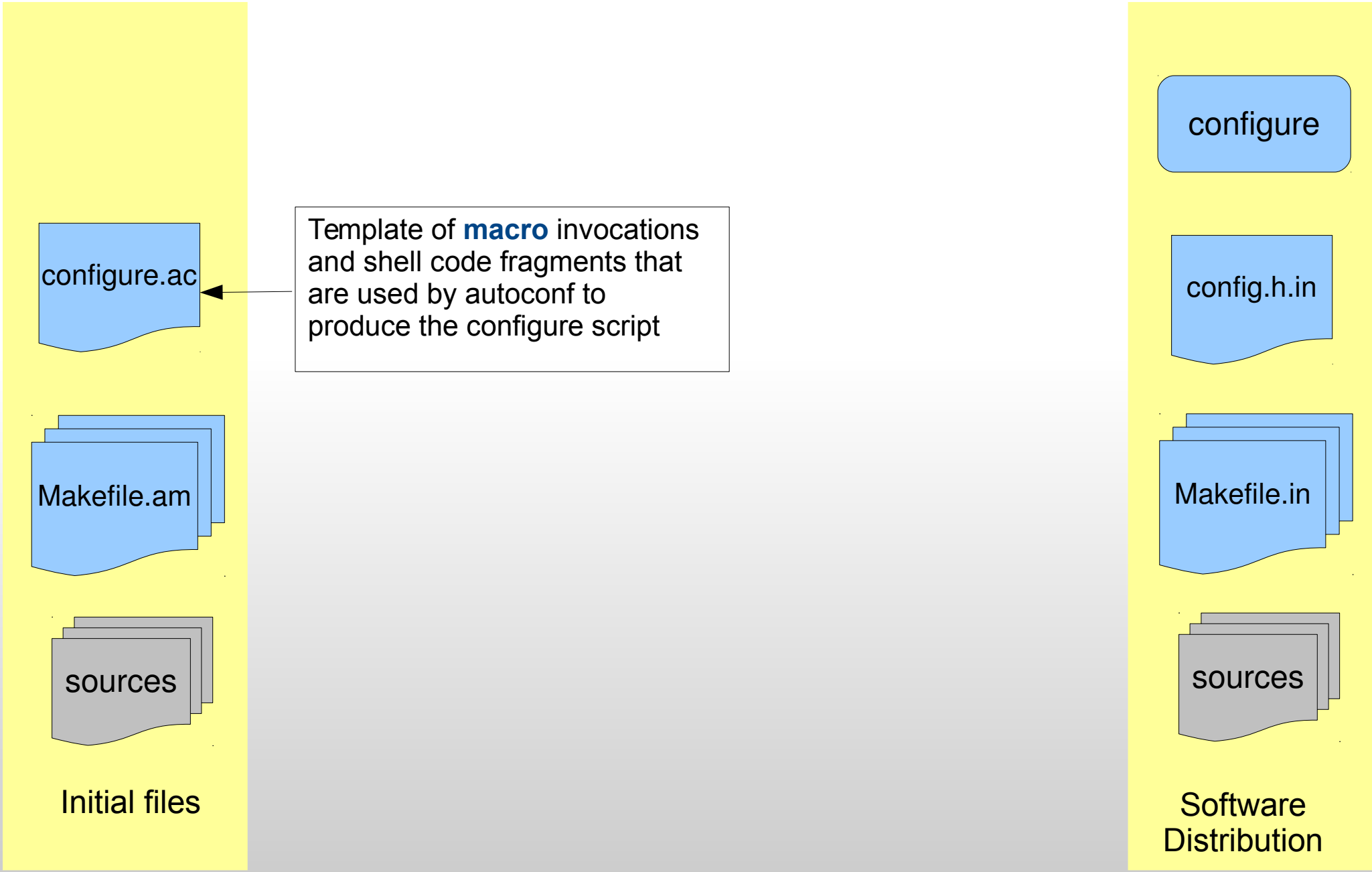
User's View


configure
config.log
config.status
config.h.in
Makefile
Makefile.in
config.h
configure
sources
make
Final program
Software Distribution

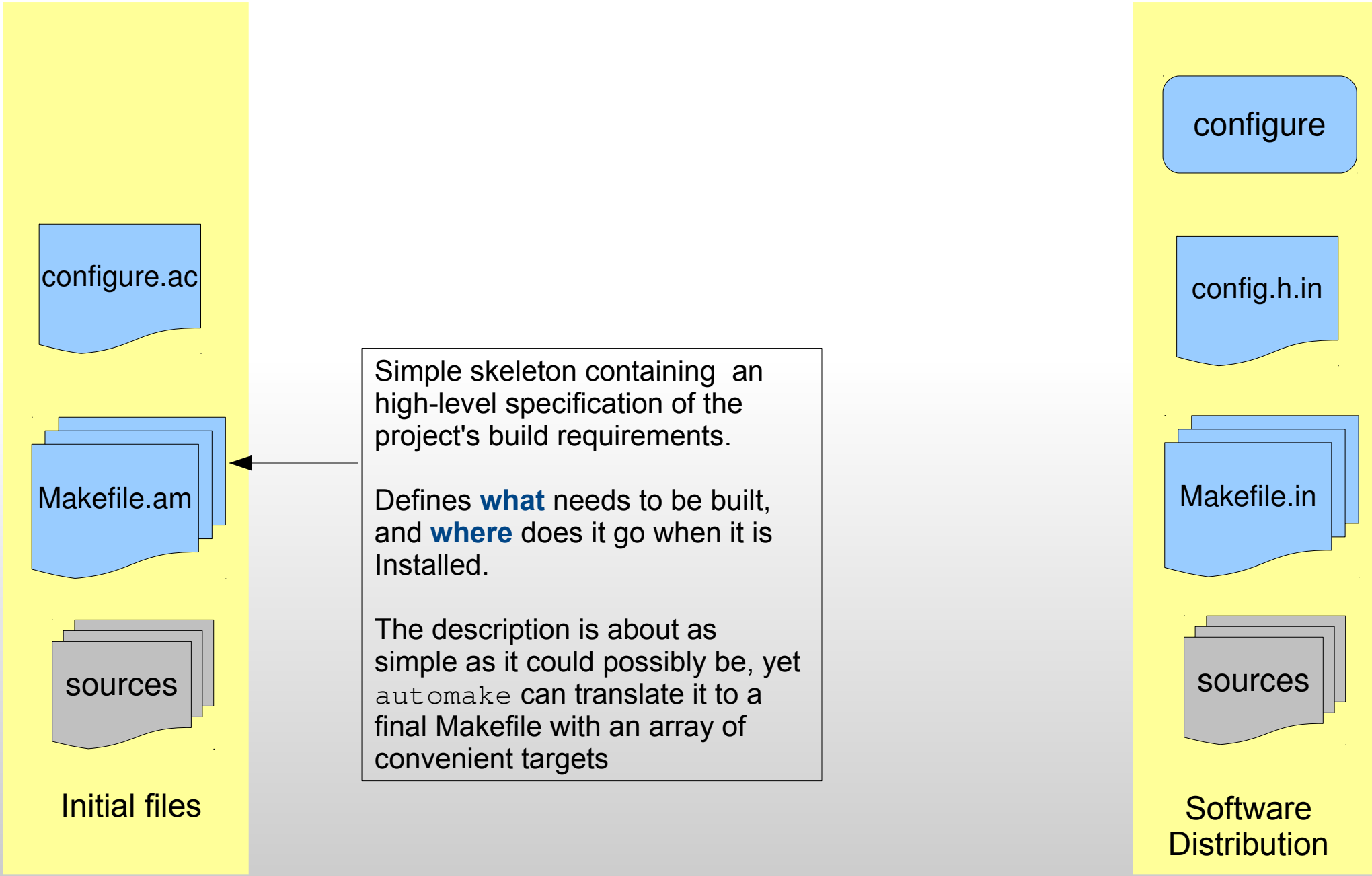
The Global Picture



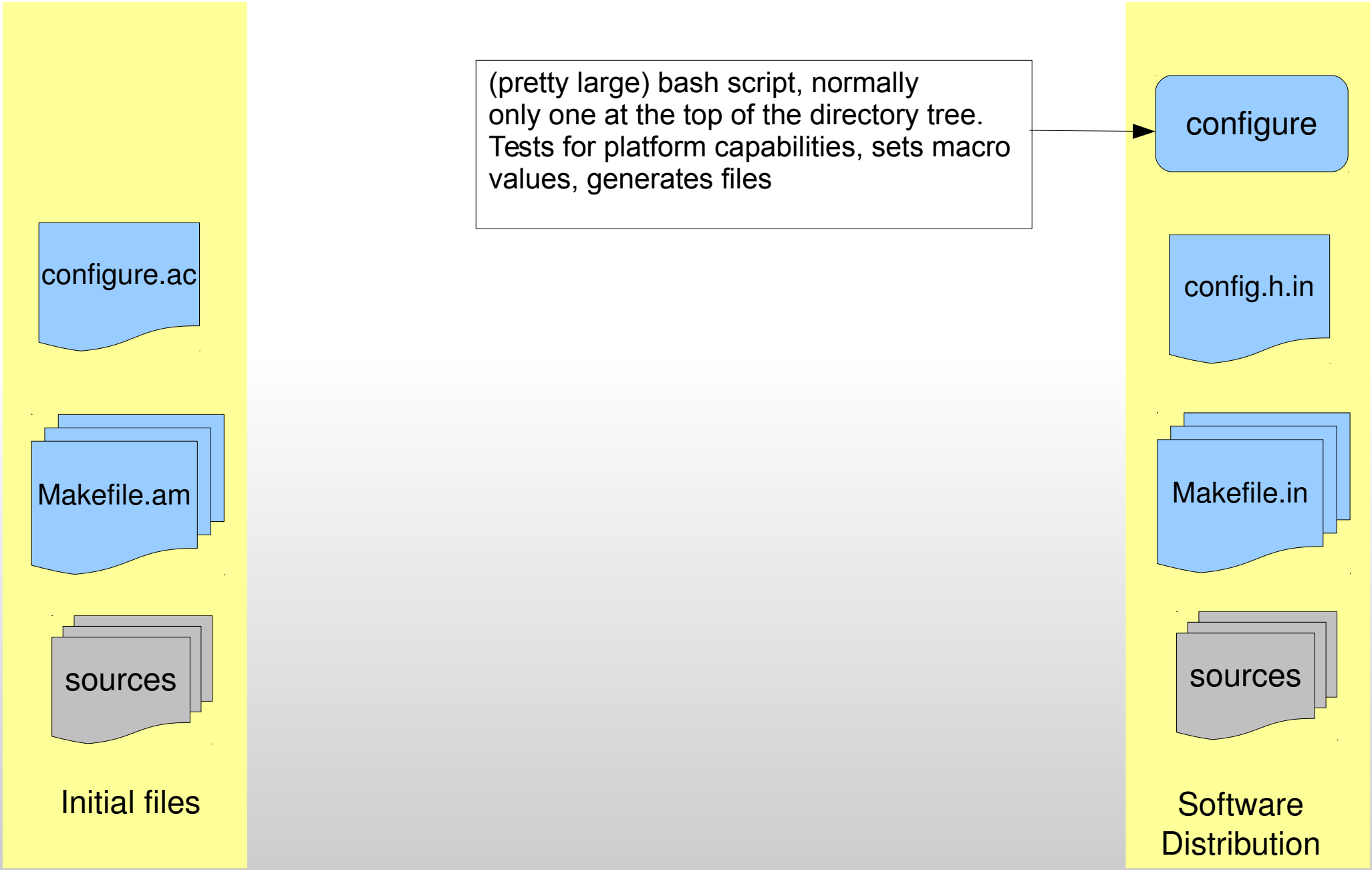
A Quick Look Inside the Boxes



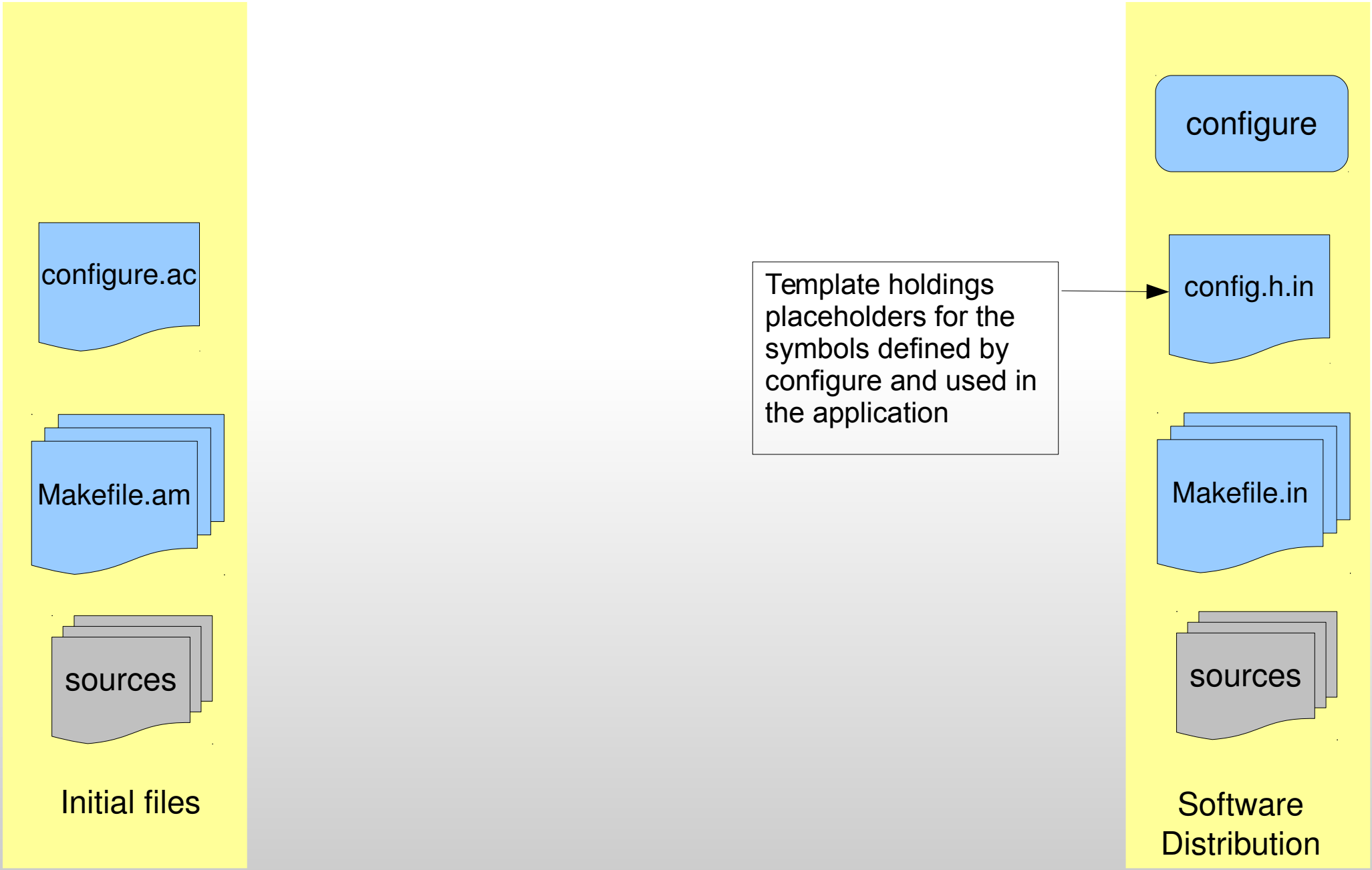
A Quick Look Inside the Boxes



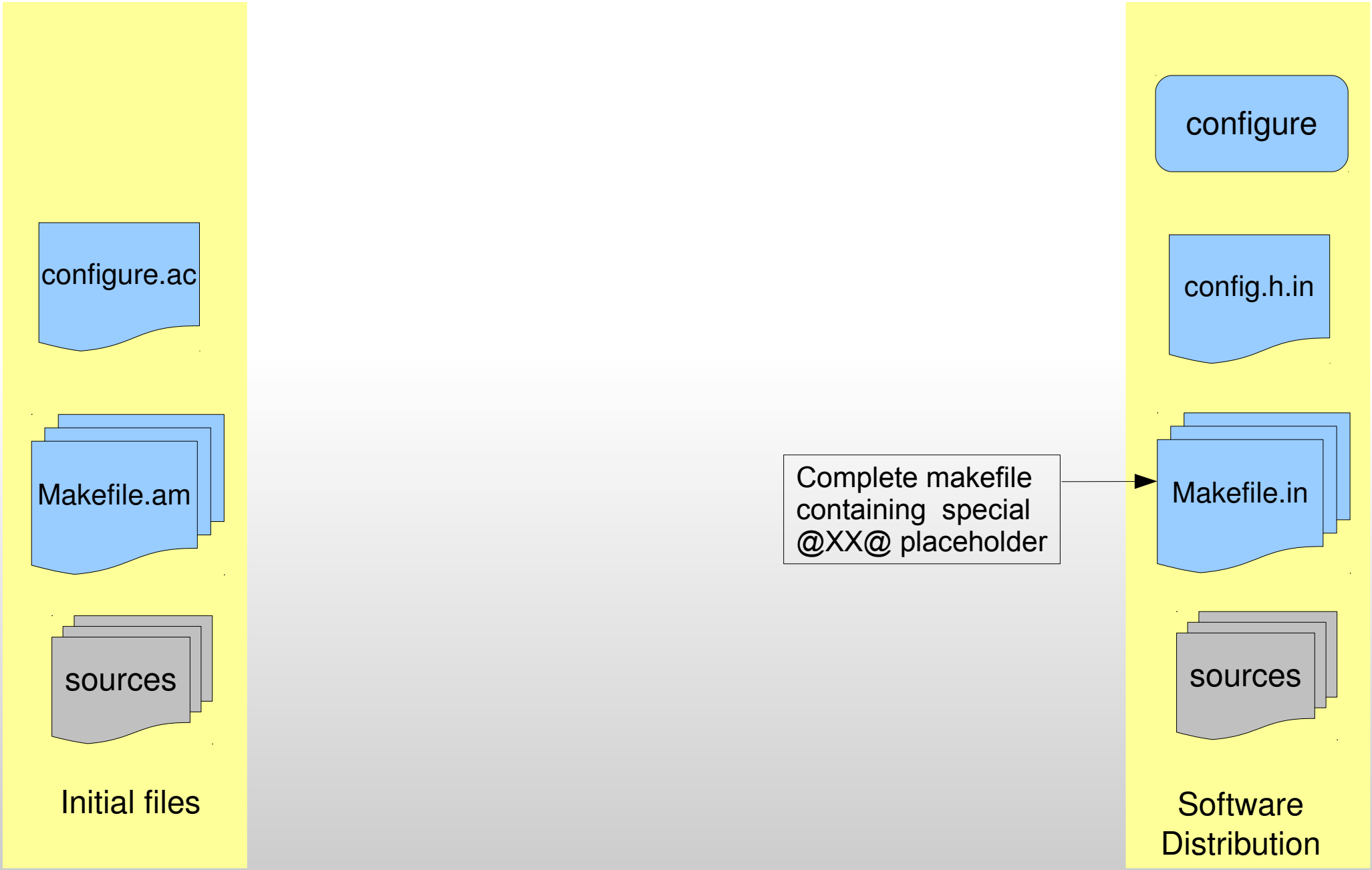
A Quick Look Inside the Boxes



A Quick Look Inside the Boxes



A Quick Look Inside the Boxes



Very Simple Example

src/func.h

```
void f(int x, int y, int z);
```

src/func.c

```
#include <stdio.h>
#include <func.h>

void f(int x, int y, int z){
    printf("%d\n",x+y+z);
}
```

src/main.c

```
#include <func.h>

int main(){
    f(1,2,3);
    return 0;
}
```

Very Simple Example

src/func.h

```
void f(int x, int y, int z);
```

src/func.c

```
#include <stdio.h>
#include <func.h>

void f(int x, int y, int z){
    printf("%d\n",x+y+z);
}
```

src/main.c

```
#include <func.h>

int main(){
    f(1,2,3);
    return 0;
}
```

Makefile.am

```
SUBDIRS = src
```

src/Makefile.am

```
bin_PROGRAMS = very_simple
very_simple_SOURCES = main.c func.c func.h
```

configure.ac

```
AC_PREREQ(2.59)

AC_INIT([very_simple], [1.0], [davide@foo.bar])
AM_INIT_AUTOMAKE([1.9 foreign])

AC_PROG_CC

AC_CONFIG_FILES([Makefile src/Makefile])
AC_OUTPUT
```

```
>
```

```
configure.ac
Makefile.am
src/
 Makefile.am
 func.c
 func.h
 main.c
```

```
> aclocal
>
```

```
configure.ac
Makefile.am
src/
 Makefile.am
 func.c
 func.h
 main.c
autom4te.cache/
aclocal.m4
```

```
> aclocal
> autoconf
```

```
configure.ac
configure
Makefile.am
src/
 Makefile.am
 func.c
 func.h
 main.c
autom4te.cache/
aclocal.m4
```

```
> aclocal
> autoconf
> automake -add-missing

configure.ac:5: installing
`./install-sh'
configure.ac:5: installing
`./missing'
src/Makefile.am: installing
`./depcomp'

>
```

```
configure.ac
configure
Makefile.am
Makefile.in
src/
 Makefile.am
 Makefile.in
 func.c
 func.h
 main.c
autom4te.cache/
aclocal.m4
install-sh
depcomp
missing
```

```
> aclocal
> autoconf
> automake -add-missing

configure.ac:5: installing
`./install-sh'
configure.ac:5: installing
`./missing'
src/Makefile.am: installing
`./depcomp'

> ./configure
checking for a …
checking wheather …
…
config.status: creating Makefile
…

>
```

```
configure.ac
configure
Makefile.am
Makefile.in
Makefile
src/
 Makefile.am
 Makefile.in
 Makefile
 func.c
 func.h
 main.c
autom4te.cache/
aclocal.m4
install-sh
depcomp
missing
config.log
config.status
```

```
> aclocal
> autoconf
> automake -add-missing

configure.ac:5: installing
`./install-sh'
configure.ac:5: installing
`./missing'
src/Makefile.am: installing
`./depcomp'

> ./configure
checking for a …
checking wheather …
…
config.status: creating Makefile
…

> make dist
…

>
```

```
configure.ac
configure
Makefile.am
Makefile.in
Makefile
src/
 Makefile.am
 Makefile.in
 Makefile
 func.c
 func.h
 main.c
autom4te.cache/
aclocal.m4
install-sh
depcomp
missing
config.log
config.status
very_simple-1.0.tar.gz
```

The General Idea

- Use `configure.ac` to tell `autoconf`:
  1. Which are the requirements of the application
  2. Which tests must be run to find them out
  3. Which variables to set according to the tests results
- Use a set of `makefile.am` (usually one per directory) to tell `automake`:

  1. What needs to be compiled (and from which sources)
  2. What needs to be distributed
  3. Where to install the files
- Include a `config.h` file in the sources and change the program accordingly with the variable that `configure` is going to put there

Part II

Automake

Writing Makefile.am

- The role of `automake` is to turn files called `Makefile.am` into `Makefile.in` for use with the configure script
- `automake` helps creating portable (but quite complex) makefiles with lots of predefined targets
- Each `Makefile.am` must be written (by hand) according to the make syntax
  - `Automake` also recognizes special macro and target names and generates code based on these
  - All macros and targets, including those which `Automake` does not recognize, are passed through to the generated `Makefile.in`
- Automake also scans `configure.ac` and uses the information it discovers to generate extra code, and sometimes to provide extra error checking

Setting up Automake in Configure.ac

AM_INIT_AUTOMAKE([OPTIONS])

- OPTIONS is a space separated list of options
- Useful options:
  - `-Wall` Turn all warnings on
  - `-Werror` Report warnings as errors
  - `1.10.1` Specify that a minimum version of automake is required
  - `foreign` Tell automake it should not be too strict when checking conformance to GNU standards (for instance, not complaining about missing files like NEWS, AUTHORS, ChangeLog..)

Uniform Naming Scheme

- Automake variables follow a scheme that makes it easy to specify how programs (and other derived objects) are built, and how they are installed

where_PRIMARY = targets

Uniform Naming Scheme

- Automake variables follow a scheme that makes it easy to specify how programs (and other derived objects) are built, and how they are installed

where_PRIMARY = targets

- PRIMARY defines what is the target
  (and therefore how it must be built)
  - PROGRAMS
  - LIBRARIES
  - LTLIBRARIES (Libtool libraries)
  - HEADERS
  - SCRIPTS
  - DATA

Uniform Naming Scheme

- Automake variables follow a scheme that makes it easy to specify how programs (and other derived objects) are built, and how they are installed

where_PRIMARY = targets

- where defines where the targets must be installed
  - bin_ installed in `$(bindir)`
  - lib_ installed in `$(libdir)`
  - noinst_ not installed

More on Installation Directories

- A number of standard directory are defined by default

Directory Variable	Default Value
`prefix`	`/usr/local`
`bindir`	`prefix/bin`
`libdir`	`prefix/lib`
`includedir`	`prefix/include`
`datadir`	`prefix/share`
`mandir`	`prefix/share/man`
`infodir`	`prefix/share/info`

- Automake allows to extend the list of possible installation directories
- A given prefix (e.g., `mypath`) is valid if a variable with the same name with 'dir' appended is defined (e.g., `mypathdir`)

```
xmldir = $(datadir)/xml
xml_DATA = file.xml
```

Other Variables

bin_PROGRAMS = hello 

Defines a target “hello”, which is a program installed in the bin directory

Now, the target name can be used with a number of assisting variables:

- hello_SOURCES = hello.c version.c system.h
  - Header files are not compiled. We list them only so they get distributed (automake does not distribute files it does not know about)
  - The list of source files cannot contain variable @var@ defined via AC_SUBST
- hello_LDADD = ../lib/mylib.a
  - Tell the linker a list of extra objects and libraries to link
  - Use plain file names to refer to libraries inside your package
- hello_LDFLAGS = … or hello_CFLAGS = ...
  - This variable is used to pass extra flags to the link (or compiler) step

Special Prefixes

- nobase_
  - Normally files are installed by copying them into the appropriate directory. The base name of the file is used when installing
  - Prepending `nobase_` will force the installer to keep the same directory structure
  - Example:

```
include_HEADERS = sys/types.h              install  $(includedir)/types.h
nobase_include_HEADERS = sys/types.h       install  $(includedir)/sys/types.h
```

- dist_ and nodist_
  - Force the targets to be included (or not included) in the distribution
  - Example:

```
dist_datadir_DATA = clean-kr.am clean.am
dist_mandir_MANS = cpio.1 mt.1
```

Recursive Subdirectories

- In packages with subdirectories, the top level `Makefile.am` must tell `automake` which subdirectories has to be built

  SUBDIRS = dir1 dir2 … dirN

- All subdirectories must contain (at build time) a Makefile

What is Distributed

- `make dist` and `make distcheck` create a tarball containing:
  - All sources declared using ..._SOURCES
  - All headers declared using ..._HEADERS
  - All scripts declared with dist_..._SCRIPTS
  - All data files declared with dist_..._DATA
  - ...
  - Common files such as ChangeLog , NEWS, etc.
    See `automake --help` for a complete list of files
  - Extra files or directories listed into EXTRA_DIST

    EXTRA_DIST = UTILS

    Add UTILS to the distribution

Conditional Makefiles

- Makefiles.am can contains conditional parts delimited by `if/endif` blocks
  - Can be used to build some programs only when a certain variable (set by configure) is set
  - However, it cannot change what is distributed !!

```
bin_PROGRAMS = foo
foo_SOURCES = foo.c
if WANT_BAR
  foo_SOURCES += bar.c
endif
```

- `bar.o` is compiled and linked to the foo program only if WANT_BAR is set
- Neverthless, both `bar.c` and `foo.c` are always included in the distribution

A Real Example (part of it)

```
SUBDIRS = resources .

bin_PROGRAMS = filezilla
filezilla_SOURCES = aboutdialog.cpp \
        asyncrequestqueue.cpp \
        aui_notebook_ex.cpp \
        …

if USE_BINRELOC
filezilla_SOURCES += prefix.cpp
endif
noinst_HEADERS = aboutdialog.h \
      asyncrequestqueue.h \
…

filezilla_CPPFLAGS = $(WX_CPPFLAGS)
filezilla_CFLAGS = $(WX_CFLAGS_ONLY)

dist_noinst_DATA = interface.vcproj
```

Part III

Autoconf

Writing configure.ac

- This is where things get quite messy :(
- `configure.ac` is a shell script that is processed by autoconf
  - Since the purpose of using autotools is portability, the shell code itself should be portable (plain sh, avoiding shell-specific syntax)
- `configure.ac` can contains macro invocations
  - Autoconf process them using an existing general-purpose macro language, called M4
  - A large set of macros already exist to check for many features
  - New macro can be written to produce custom checks
  - It is quite common to have `configure.ac` without shell code, containing only macro invocation

Getting Started: Autoscan

- The autoscan tool can help creating and maintain a `configure.ac` file for a software package
- autoscan examines the source files for common portability problems. Based on its finding:
  - It creates a file `configure.scan` which can be used as a preliminary `configure.ac` for the package
  - It checks a possibly existing `configure.ac` for completeness and suggests the necessary changes
- The `autoscan` output can contain mistakes (like macros in the wrong order) and things that must be filled up by the developer
  - It is a good start but it almost always requires some manual adjustments

Standard Layout

```
# Prelude
AC_INIT(package, version, bug-report-address)
AM_INIT_AUTOMAKE([options])
AC_CONFIG_SRCDIR([file])
# checks for programs

# checks for libraries

# checks for header files

# checks for types and structures

# checks for compiler characteristics

# checks for library functions

# checks for system services

# output files
AC_CONFIG_FILES([file...])
AC_OUTPUT
```

Macros

- Macro arguments need to be quoted
  - In M4 the quote characters are [and] (and not ' or ")
- By convention, the first characters specify the type of macro
  - Autoconf provides a set of macros (m4_*, AS_*, AH_*, AC_*, AT_*)
  - Other macros can be provided by third-party tools (e.g., Automake AM_* macros). These macros must be defined in the aclocal.m4 file
- The aclocal tool automates the construction of `aclocal.m4` from various sources

  - A system-wide directory (usually `/usr/share/aclocal/`) where third-party packages may install their macros
  - Automake's own private macro directory
  - A directory specified on the command line containing the user's macros

Few Examples of Macros

AC_DEFINE(VARIABLE, VALUE, DESCRIPTION)

- Define a C pre-processor symbol (usually to store the result of a feature test). If AC_CONFIG_HEADERS has been called, AC_OUTPUT creates a header file by substituting the correct variable values into #define statements in a template file
- Example:

```
AC_DEFINE([ANSWER], [42], [The famous answer])
```

add to the config.h the following piece of code:

```
/* The famous answer */
#define ANSWER 42
```

AC_CHECK_PROGS(VAR, PROGS, [VAL-IF-NOT-FOUND])

- Define VAR to the first PROGS found, or to VAL-IF-NOT-FOUND otherwise
- Example:

```
AC_CHECK_PROGS([TAR], [tar gtar], [None])
if test "$TAR" = None; then
  AC_MSG_ERROR([This package needs tar.])
fi
```

Few Examples of Macros

AC_SEARCH_LIBS(F, LIBS, [ACT-IF-FOUND], [ACT-IF-NOT])

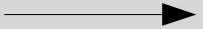
- Search for a library in LIBS defining the function F (if it's not already available). Add -llibrary to LIBS for the first library found to contain function, and then run ACT-IF-FOUND

AC_SUBST(VAR, [VALUE])

- Make AC_OUTPUT substitute the variable VAR into output files
- This means that AC_OUTPUT replaces instances of '@variable@' in input files with the value that the shell variable VAR has when AC_OUTPUT is called
- If VALUE is given, in addition assign it to VAR

- Example:

```
HTML_DIR=/var/html/
AC_SUBST(HTML_DIR)
```



Replace all occurrences of @HTML_DIR@ in any output file (usually all Makefile.am) with the value `/var/html/`

Few Examples of Macros

AC_CONFIG_HEADERS(HEADERS)

- For each file X.in in HEADERS, create the corresponding header file X, by substituting the #undef placeholder with the variables defined by AC_DEFINE
- It's better to use only one such header (by convention config.h) that can be automatically created by invoking the autoheader tool

AC_CONFIG_FILES(FILES)

- Make AC_OUTPUT create each file X in FILES by copying an input file (by default X.in), substituting each @*variable*@ entry with the output variable values defined by AC_SUBST (plus a number of pre-defined ones like CFLAGS, LIBS, ...)
- Example:

```
AC_CONFIG_FILES([Makefile sub/Makefile script.sh:script.in])
  Creates Makefile from Makefile.in
          sub/Makefile from sub/Makefile.in
          script.sh from script.in
```

Writing Test Programs

The default set of macros provides a large set of tests but if you don't find what you need, you have to write a new one

AC_LANG_PROGRAM(prologue, body)

- Expands into a source file which consists of the prologue, and then body as body of the main function

AC_COMPILE_IFELSE (program, [action-if-true], [action-if-false])

AC_LINK_IFELSE (program, [action-if-true], [action-if-false])

AC_RUN_IFELSE (program, [action-if-true], [action-if-false])

- Run the compiler on the program, and execute action-if-true if compilation succeed, action-if-false otherwise
- Run the compiler and the linker...
- Compile and link the program and verify that returns an exit status of 0 if executed

Example

```
AC_COMPILE_IFELSE(
   [ AC_LANG_PROGRAM(
         [[#include <pthread.h>]],
         [[pthread_mutexattr_setprotocol(NULL,  2);]] )
    ],
    [ AC_MSG_RESULT([yes]) ],
    [ AC_MSG_FAILURE([no]) ]
)
```

Summary (so far..)

File	Written by..	Required by..
`configure.ac`	*HAND* + autoscan	aclocal, autoconf, automake, autoheader
`Makefile.am`	*HAND*	automake
`aclocal.m4`	aclocal	autoconf, automake
`configure`	autoconf	
`config.h.in`	autoheader	configure
`Makefile.in`	automake	configure
`Makefile`	configure	
`config.h`	configure	

Part IV

Libtool

The Problem of Shared Libraries

- The format of shared libraries differs between systems
  - `libhello.so`
  - `libhello.dll`
  - `libhello.sl`
  - `libhello.dylib`
- Also in unix-like systems shared libraries are built, named and managed in different ways
  - Some platforms don't even provide native shared libraries (Ultrix 4.2)
  - Some platforms name their libraries libXX.so, while others use just XX.so
  - Different compilers require different flags to build the library
- How can we keep track of all this?

Libtool

- GNU `libtool` simplifies the developer's job by
  - Encapsulating the platform-specific dependencies
  - Hiding the complexity of using shared libraries behind a consistent, portable interface
  - Providing to the user a simple, portable way to build libraries
- The libtool solution consists of
  - A new library format that abstracts all the others:
    `libname.la` (libtool archive)
  - A wrapper script for the compiler and linker that translates operations involving `libname.la` into the correct operation for the current system using the real library
- Libtool is usable by itself, but we will see how to integrate it with autoconf and automake

First Step: libtoolize

- The `libtoolize` shell script provides a standard way to add libtool support to an autotool package
- `libtoolize` prepare `configure` to generate a custom version of the libtool script in the project directory. This script is then executed at the appropriate time by the automake-generated makefiles
- Libtoolize adds to the project the following files:
  - `config.guess`: script that try to guess the canonical system name
  - `config.sub`: contains a list of the machine supported by (some of) the GNU software and tries to match them to a canonical name
  - `ltmain.sh`: script that provides generalized library-building support services

Adding libtool to an Autotool Project

- Add AC_PROG_LIBTOOL to `configure.ac` to initialize libtool
- Use the LTLIBRARIES primary to declare libtool archives in `Makefile.am`

```
lib_LTLIBRARIES = mylib.la
```

- Use the LDADD variable on the program target to link against local libtool archives

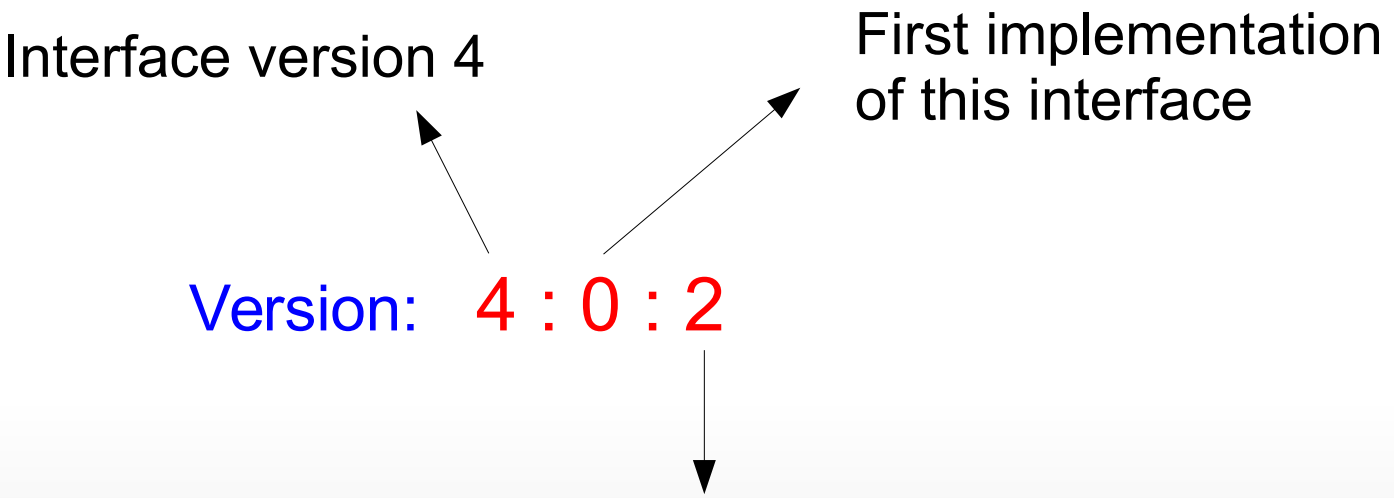
```
bin_PROGRAMS = myprog
myprog_LDADD = mylib.la
```

- By default, both static and shared libraries are built

Library Names and Versions

- Libtool has its own formal versioning scheme that tracks changes to the library interface
  - The interface is the set of exported entry points into the library
- The library version is composed by three parts:
  - CURRENT – The latest interface implemented by the library
  - REVISION – The implementation number of the CURRENT interface (e.g., build number or number of bugs fix)
  - AGE – The difference between the newest and oldest interfaces that this library implements. In other words, the library implements all the interface numbers in the range from number current - age to current

Library Names and Versions



The library is backward compatible with the two previous versions.

It can be linked into executables which were built with a release of this library that exported the current interface number, or any of the previous two interfaces.

Naming Conventions

- Start with version information `0:0:0` for each libtool library
  - These numbers should be specified using `-version-info` (if not, the default is 0:0:0)

```
lib_LTLIBRARIES = libhello.la
libhello_la_SOURCES = say.c say.h
libhello_la_LDFLAGS = -version-info 0:0:0
```

- The version number of a project is completely different from the version number of any library shipped with the project (!)
  - It is a very common mistake to try to force the libraries to have the same version number as the current release version of the package

Update the Library Version

- Update the version information only immediately before a public release of your software

- If the library source code has changed since the last update (e.g. for a bugfix) but the interface has not changed, then increment *revision*

$$c : r : a \rightarrow c : r+1 : a$$

Update the Library Version

- Update the version information only immediately before a public release of your software

- If the library source code has changed since the last update (e.g. for a bugfix) but the interface has not changed, then increment *revision*

$$c : r : a \rightarrow c : r+1 : a$$

- If any interface has been added, removed, or changed since the last update, increment *current*, and set *revision* to 0
  - If the new interface is a superset of the previous interface (that is, if the previous interface has not been broken by the changes in this new release), increment the a*ge*

$$c : r : a \rightarrow c+1 : 0 : a+1$$

Update the Library Version

- Update the version information only immediately before a public release of your software

- If the library source code has changed since the last update (e.g. for a bugfix) but the interface has not changed, then increment *revision*

$$c : r : a \rightarrow c : r+1 : a$$

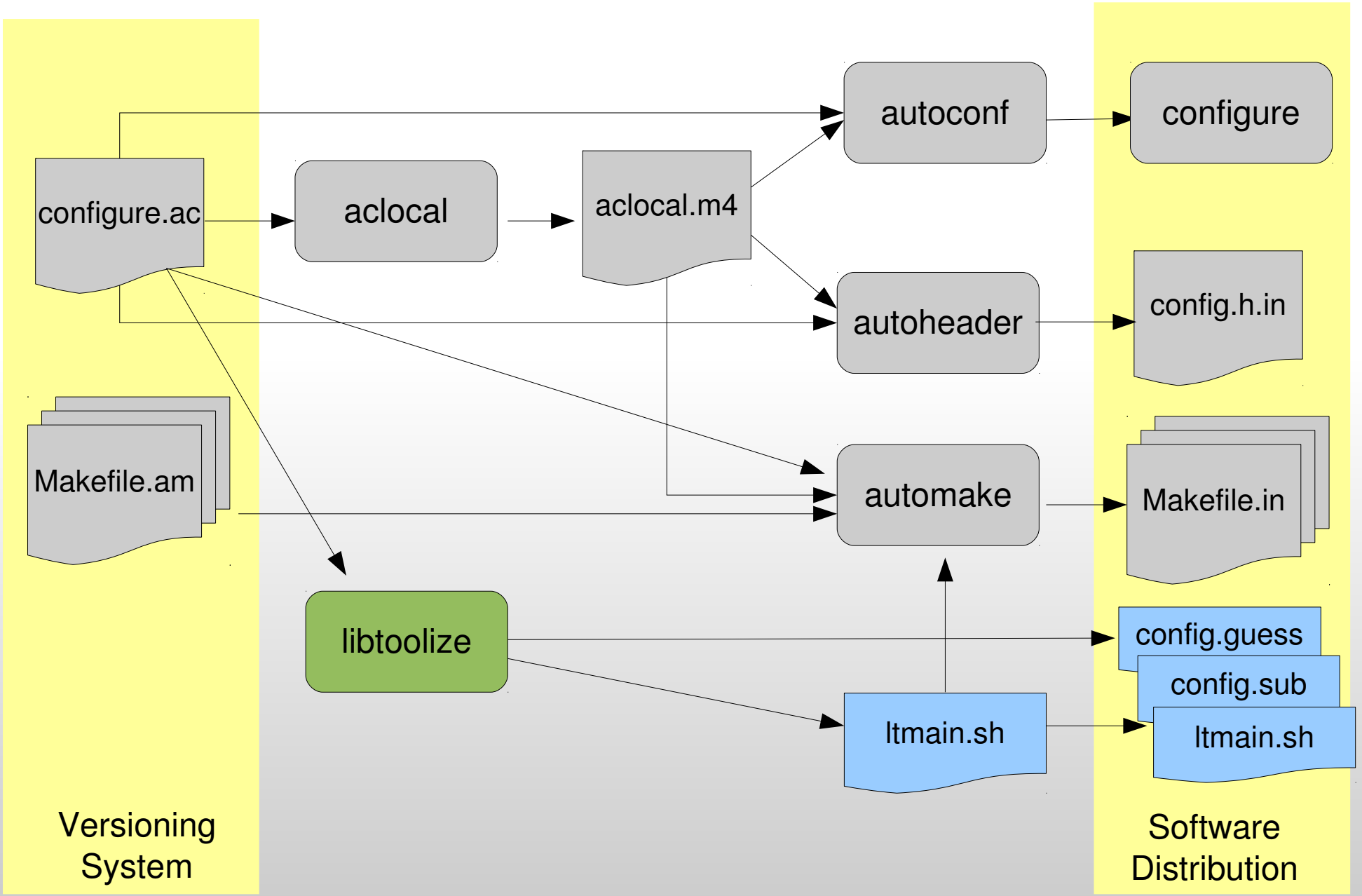
- If any interface has been added, removed, or changed since the last update, increment *current*, and set *revision* to 0
  - If the new interface is a superset of the previous interface (that is, if the previous interface has not been broken by the changes in this new release), increment the a*ge*

$$c : r : a \rightarrow c+1 : 0 : a+1$$

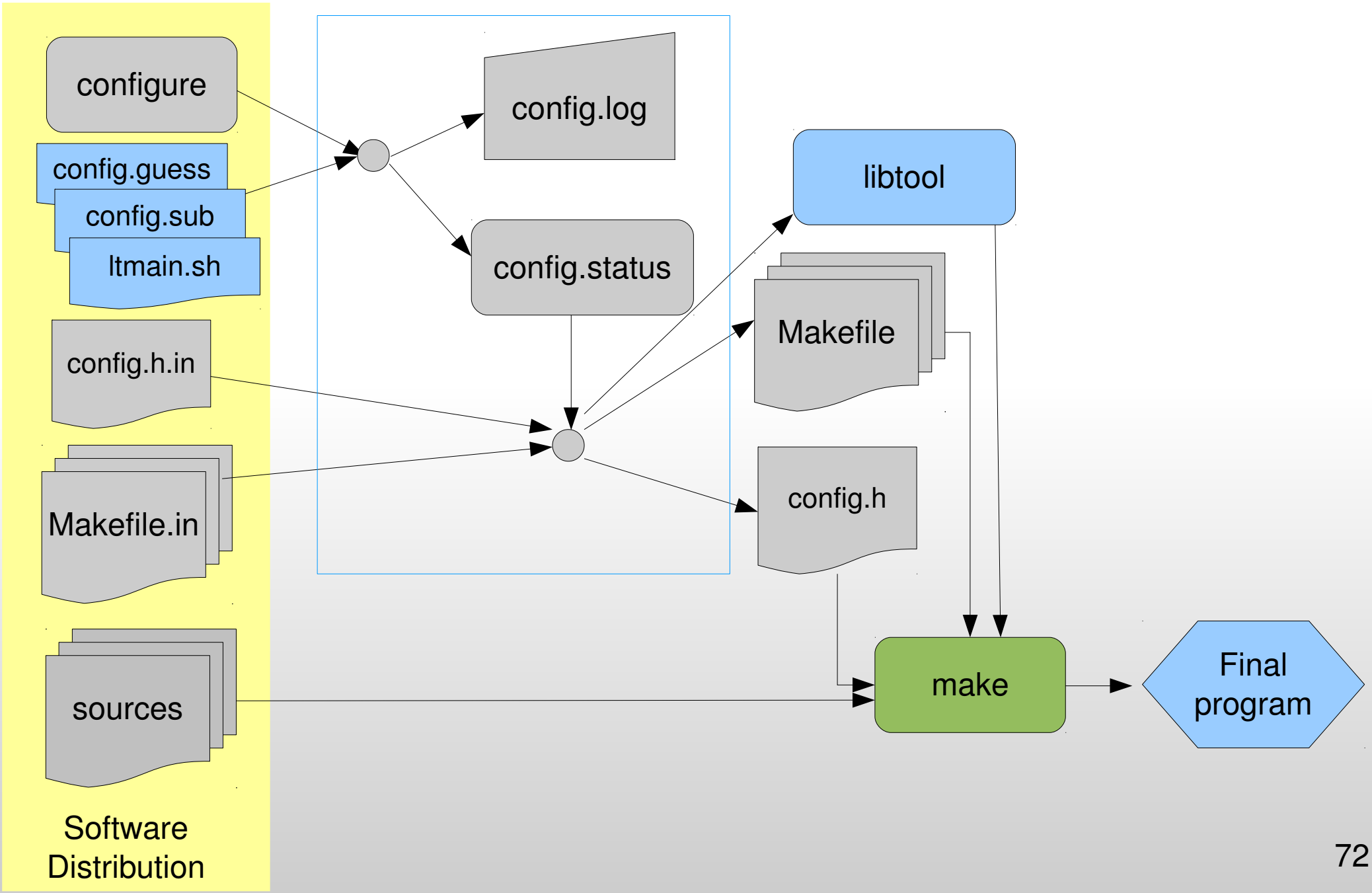
  - f the new interface has removed elements with respect to the previous interface, then you have broken backward compatibility, then set a*ge* to 0

$$c : r : a \rightarrow c+1 : 0 : 0$$

Developer's View (with libtool)



Installer's View (with libtool)



Very Simple Example

src/func.h

```
void f(int x, int y, int z);
```

src/func.c

```
#include <stdio.h>
#include <func.h>

void f(int x, int y, int z){
    printf("%d\n",x+y+z);
}
```

src/main.c

```
#include <func.h>

int main(){
    f(1,2,3);
    return 0;
}
```

Makefile.am

```
SUBDIRS = src
```

src/Makefile.am

```
lib_LTLIBRARIES = libfunc.la
libfunc_la_SOURCES = func.c func.h
libfunc_la_LDFLAGS = -version-info 1:0:0

bin_PROGRAMS = very_simple
very_simple_SOURCES = main.c
very_simple_LDADD = libfunc.la
```

configure.ac

```
AC_INIT([very_simple], [1.0], [davide@foo.bar])
AC_PROG_LIBTOOL
AM_INIT_AUTOMAKE([1.9 foreign])
AC_PROG_CC
AC_CONFIG_FILES([Makefile src/Makefile])
AC_OUTPUT
```

```
> aclocal
> libtoolize
> autoconf
> automake -a
```

```
configure.ac
configure
Makefile.am
Makefile.in
config.guess
ltmain.sh
config.sub
src/
 Makefile.am
 Makefile.in
 func.c
 func.h
 main.c
autom4te.cache/
aclocal.m4
install-sh
depcomp
missing
config.log
config.status
```

```
> aclocal
> libtoolize
> autoconf
> automake -a
> configure
....
....
>
```

```
configure.ac
configure
Makefile.am
Makefile.in
Makefile
config.guess
ltmain.sh
config.sub
libtool
src/
 Makefile.am
 Makefile.in
 Makefile
 func.c
 func.h
 main.c
autom4te.cache/
aclocal.m4
install-sh
depcomp
missing
config.log
config.status
```

```
> aclocal
> libtoolize
> autoconf
> automake -a
> configure
....
....
> make
....
....
>
```

```
src/
 Makefile.am
 Makefile.in
 Makefile
 func.c
 func.h
 main.c
 very_simple
 libfunc.la
 func.o
 func.lo
 main.o
 .libs/
   very_simple
   libfunc.so.1
   libfunc.la
   func.o
   libfunc.so.1.0.0
   libfunc.a
   libfunc.lai
   libfunc.so
```

```
> make
....
....
> file src/*
 func.c:       ASCII C program text
 func.lo:      ASCII English text
 func.o:       ELF 32-bit LSB ...
 libfunc.la:   libtool library file
 main.o:       ELF 32-bit LSB
 very_simple: Bourne-Again shell script text executable
 ....
```

```
> make
....
....
> file src/*
 func.c:       ASCII C program text
 func.lo:      ASCII English text
 func.o:       ELF 32-bit LSB ...
 libfunc.la:   libtool library file
 main.o:       ELF 32-bit LSB
 very_simple: Bourne-Again shell script text executable
  ....

> file src/.libs/*
 func.o:             ELF 32-bit LSB relocatable...
 libfunc.a:          current ar archive
 libfunc.la:         symbolic link to `../libfunc.la'
 libfunc.lai:        libtool library file
 libfunc.so:         symlink to `libfunc.so.1.0.0'
 libfunc.so.1:       symlink to `libfunc.so.1.0.0'
 libfunc.so.1.0.0: ELF 32-bit LSB shared object...
 very_simple:        ELF 32-bit LSB executable,
     dynamically linked (uses shared libs), not stripped
```

Uninstalled Binaries

(after you run make but before you run make install)

- The program (`very_simple`) is not actually a binary, but a shell script which sets up the environment so that when the real binary is called it finds its shared libraries in the correct locations
  - The real binary is built too, but it is stored in an hidden subdirectory
- If you need to look at the binary with another program (to debug it, for example) you have to use the `libtool` script to run the program

```
> libtool gdb very_simple
```

Installed Binaries

```
> ./configure --prefix /tmp/test
....
....
> make & make install
....
> cd /tmp/test
> ls
  bin/ lib/

> find . -exec file {} \;
./lib/libfunc.so: symbolic link to `libfunc.so.1.0.0'
./lib/libfunc.so.1: symbolic link to `libfunc.so.1.0.0'
./lib/libfunc.la: libtool library file
./lib/libfunc.a: current ar archive
./lib/libfunc.so.1.0.0: ELF 32-bit LSB shared object..

./bin/very_simple: ELF 32-bit LSB executable, dynamically
                   linked (uses shared libs)
```

Acknowledgements

These slides where inspired (and partially copied) from the great tutorial from Alexandre Duret-Lutz

You can find it here:

`http://www.lrde.epita.fr/~adl/autotools.html`