

Software Development Autotools

Davide Balzarotti

Eurecom – Sophia Antipolis, France

Software Development Tools

1. Writing and managing code
2. Configuring and Building the program
 - ✓ GCC
 - ✓ Makefiles
 - ✓ Autotools
3. Packaging and Distributing the application
4. 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

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
- Unfortunately, not all systems are exactly the same
 - C libraries can be slightly different in different OSs
 - Tools (compilers, sed, tar, ..) can be different or have different names
 - Files and libraries can be located in different places
- This would require
 - changes to the source code to take care of different C functions
 - changes to makefiles to take care of 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...
 - may be located in different libraries
(is `pow()` in `libm.so` or in `libc.so`?)
 - 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 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 Makefiles 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 in 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
```

## ■ Uninstalling:

- ```
> sudo make uninstall
```

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 distribute 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`.

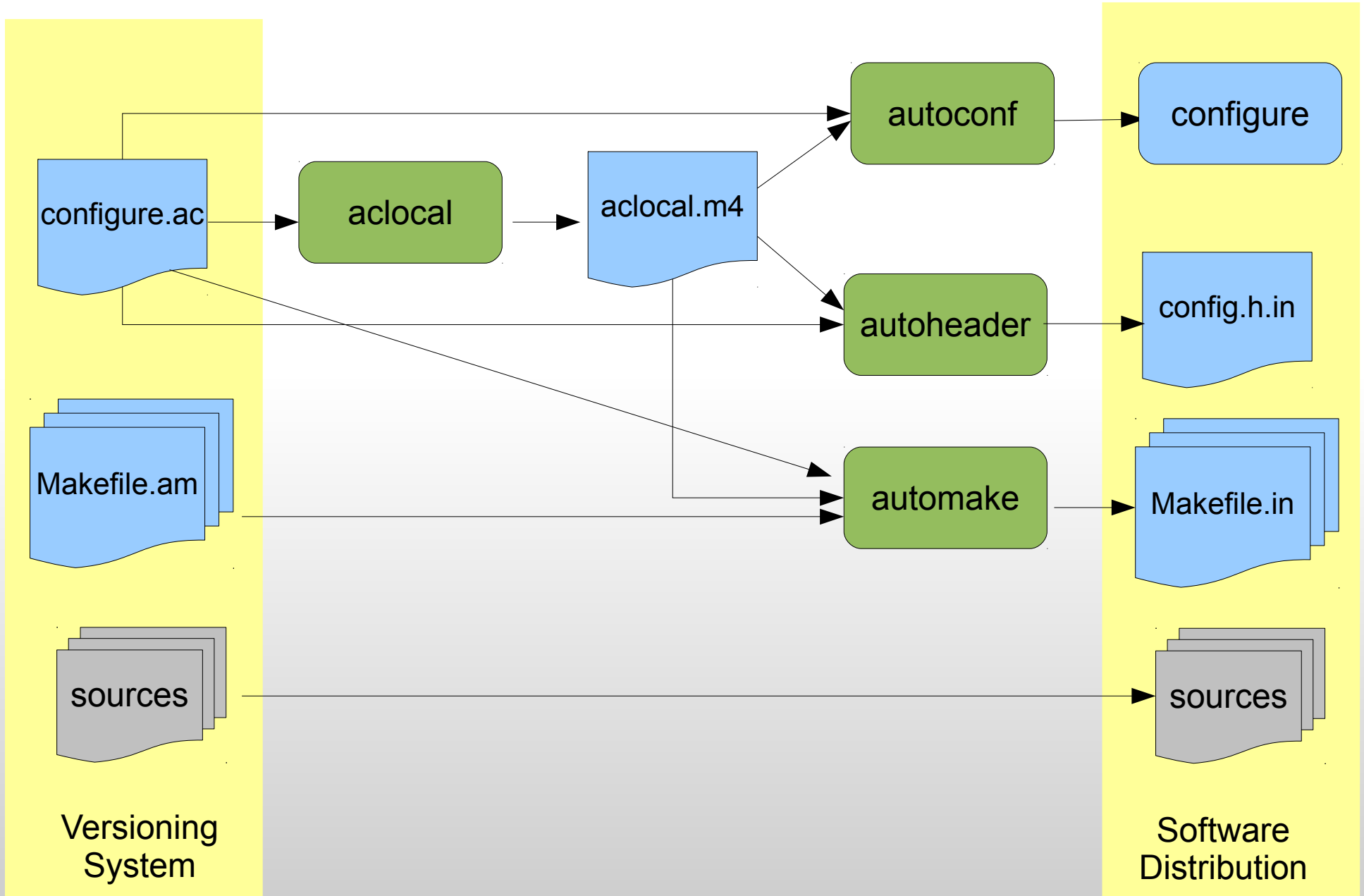
configure - script to configure the build for the local machine. Generates `Makefile` from `Makefile.in` and `config.h.in`.

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

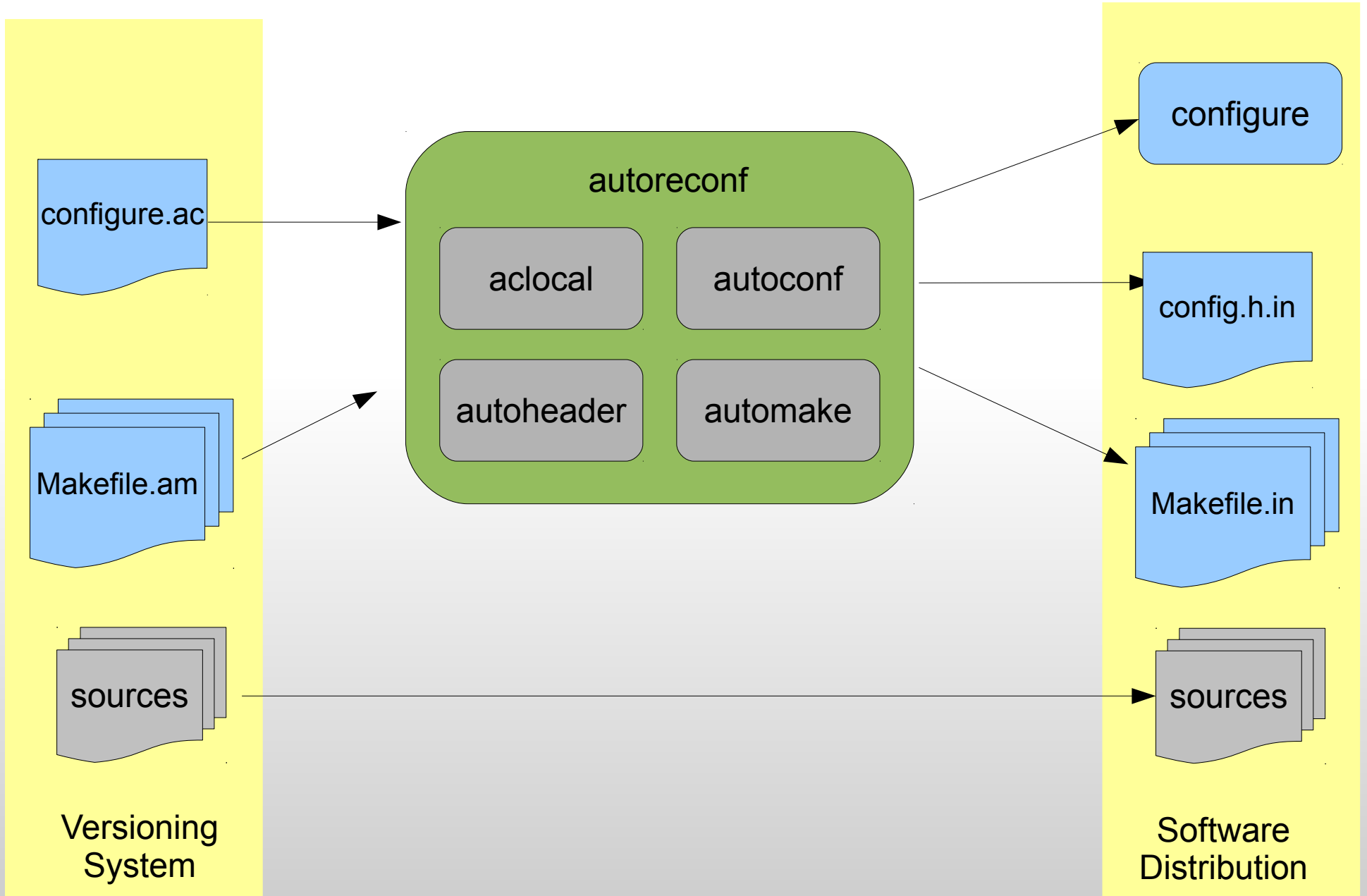
autoreconf - Run all tools in the right order

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

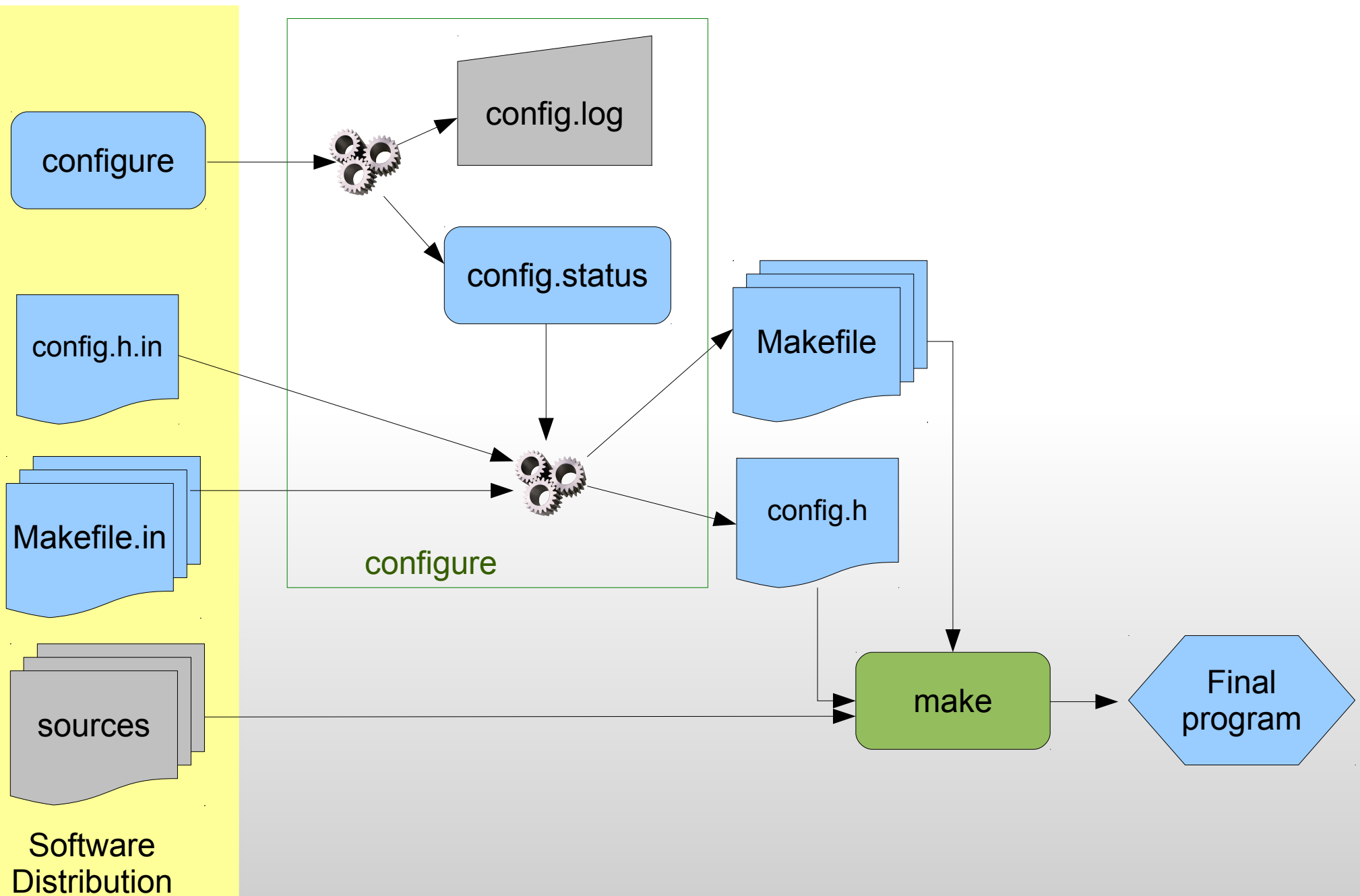
Developer's View



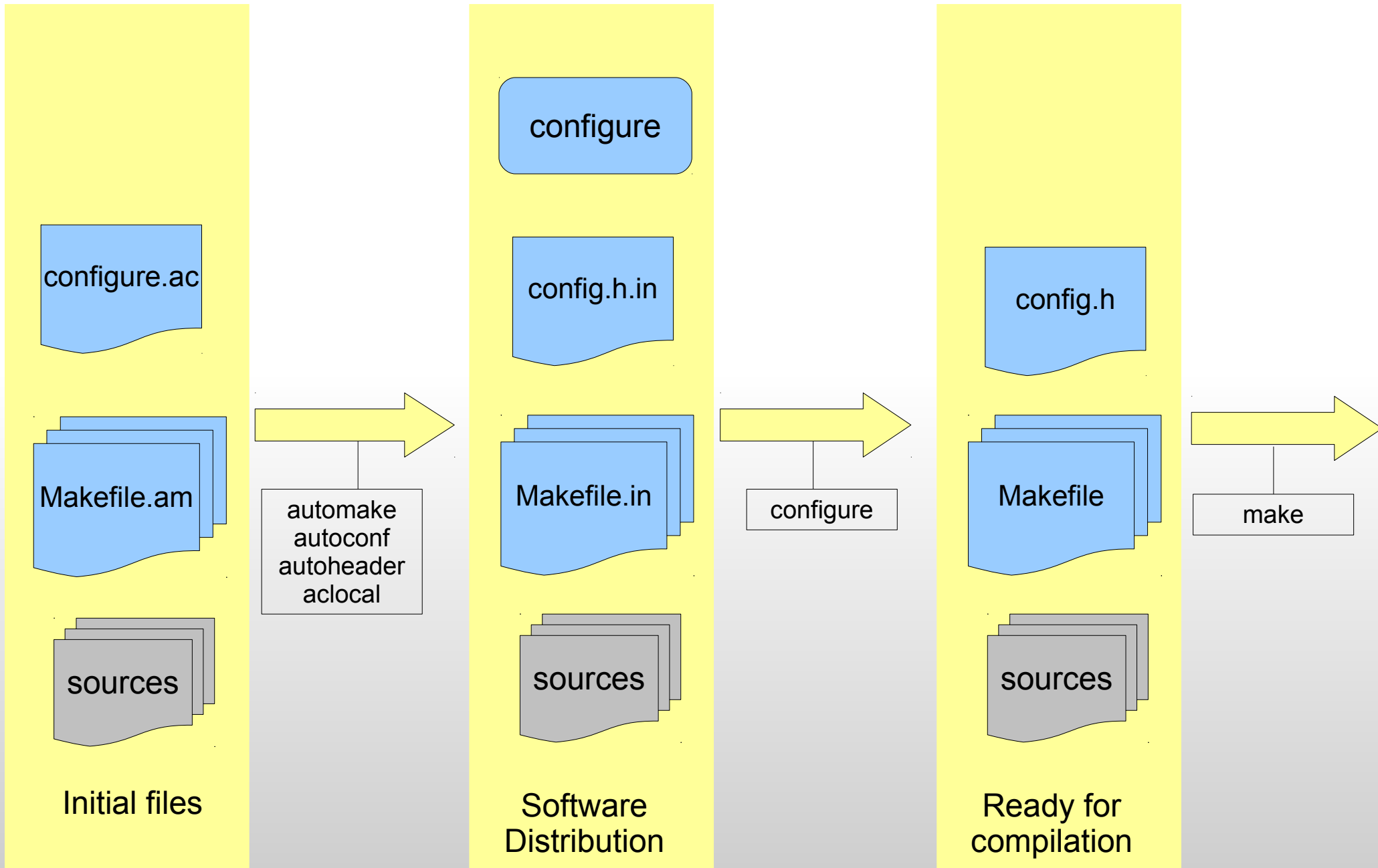
Developer's View



User's View



The Global Picture



A Quick Look Inside the Boxes

configure.ac

Template of macro invocations and shell code fragments that are used by `autoconf` to produce the `configure` script

Makefile.am

Simple skeleton containing an high-level specification of the project's build requirements.

Defines what needs to be built, and where does it go when it is Installed.

The description is about as simple as it could possibly be, yet `automake` can translate it to a final `Makefile` with an array of convenient make targets

sources

Initial files

configure

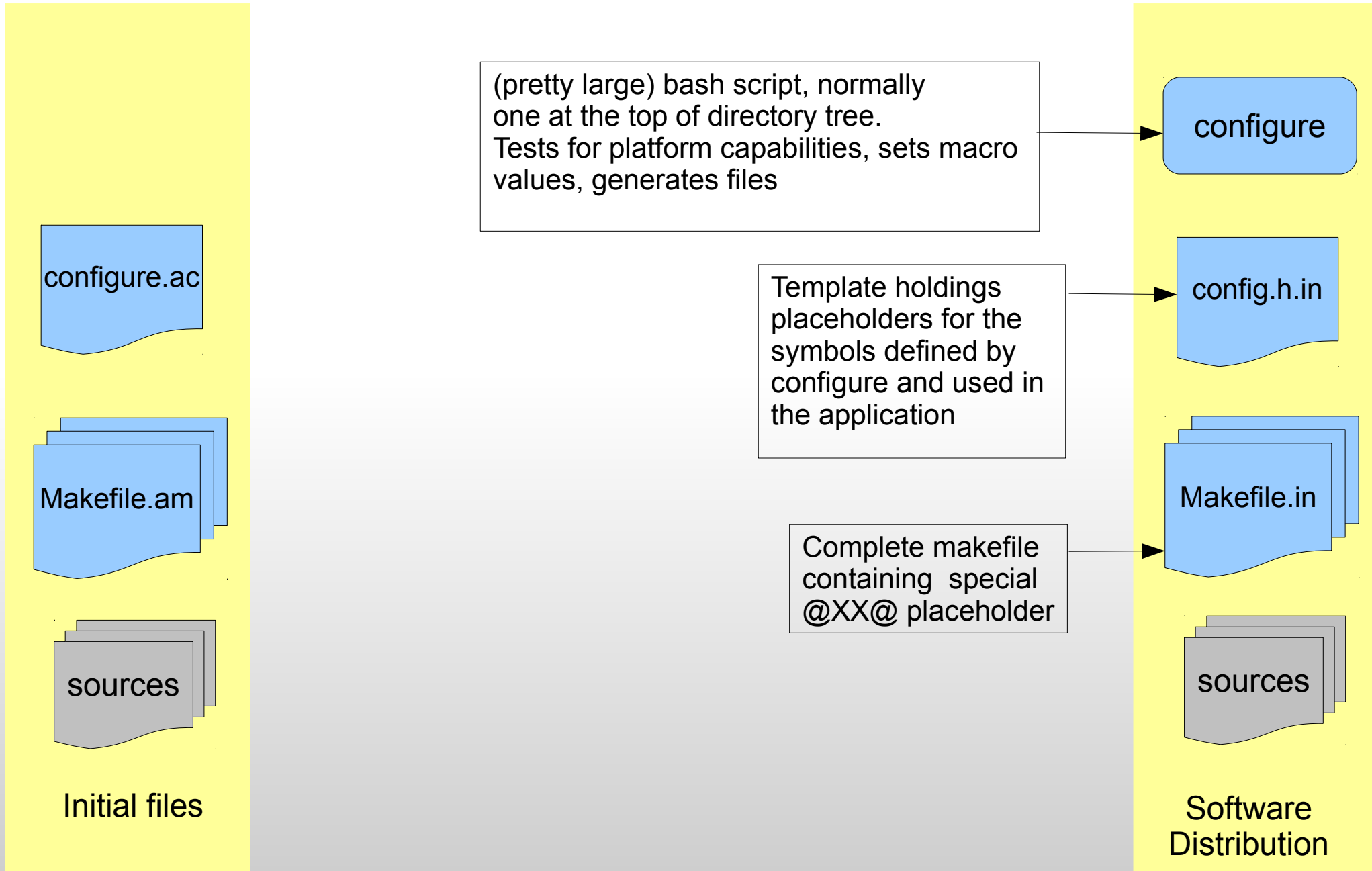
config.h.in

Makefile.in

sources

Software Distribution

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`:
 - Which are the requirements of the application
 - Which tests must be run to find them out
 - Which variables to set according to the tests results
- Use a set of `makefile.am` (usually one per directory) to tell `automake`:
 - What has to be compiled (and from which sources)
 - What has to be distributed
 - 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

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 generally follow a scheme that makes it easy to decide how programs (and other derived objects) are built, and how they are installed

`where_PRIMARY` = targets

Uniform Naming Scheme

- Automake variables generally follow a scheme that makes it easy to decide 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 generally follow a scheme that makes it easy to decide 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
<code>prefix</code>	<code>/usr/local</code>
<code>bindir</code>	<code>prefix/bin</code>
<code>libdir</code>	<code>prefix/lib</code>
<code>includedir</code>	<code>prefix/include</code>
<code>datadir</code>	<code>prefix/share</code>
<code>mandir</code>	<code>prefix/share/man</code>
<code>infodir</code>	<code>prefix/share/info</code>

- 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

- Associated with each PRIMARY are several assisting variables that are named after the target
- For example, if `makefile.am` contains:
`bin_PROGRAMS = hello`
- The following variables associated with `hello` are available:
 - `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

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
- Nevertheless, both `bar.c` and `foo.c` are always included in the distribution

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

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
```

Writing `configure.ac`

- That's where things get 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 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
 - 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). This macro 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

Useful Macros

`AC_MSG_ERROR(ERROR-DESCRIPTION, [EXIT-STATUS])`

- Print ERROR-DESCRIPTION (also to config.log) and abort configure

`AC_MSG_WARN(WARNING-DESCRIPTION)`

- Likewise, but don't abort

`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
```

Useful Macros

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/

Prelude Macros

AC_INIT(PACKAGE, VERSION, BUG-REPORT-ADDRESS)

- Mandatory Autoconf initialization
- Example: AC_INIT([MyProgram], [1.1], [me@myplace.com])

AC_PREREQ(VERSION)

- Specify that a minimum Autoconf version is required
- Example: AC_PREREQ([2.61])

AC_CONFIG_SRCDIR(FILE)

- A safety check. FILE should be a distributed source file, and this makes sure that 'configure' is not run from the wrong directory
- Example: AC_CONFIG_SRCDIR([src/main.c])

AC_CONFIG_AUX_DIR(DIRECTORY)

- Specify where to store auxiliary scripts such as install-sh and depcomp
- Example: AC_CONFIG_AUX_DIR([build-aux])

Programs Checks Macros

`AC_PROG_CC`, `AC_PROG_CXX`, `AC_PROG_F77`, ...

- Check for existing compilers and set it to corresponding variables (e.g., `CC` will contain the name of the C compiler)

`AC_PROG_SED`, `AC_PROG_LEX`, `AC_PROG_GREP`, ...

- Find good implementations and set `$SED`, `$YACC`, `$LEX` variables

`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
```

Programs Checks Macros

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

- Like `AC_CHECK_PROGS`, but if any of `PROGS` are found, set `VAR` to the absolute name of the program found

`AC_PATH_PROGS_FEATURE_CHECK (VAR, PROGS, TEST, [ACTION-IF-NOT-FOUND])`

- Run `TEST` for each programs in `PROGS` with `ac_path_VAR` set to the absolute name of the candidate program. If no invocation of `TEST` sets the shell variable `ac_cv_path_VAR`, then `ACTION-IF-NOT-FOUND` is executed
- Example:

```
AC_PATH_PROGS_FEATURE_CHECK( [M4], [m4 gm4],  
[[ m4out=`echo 'changequote([,])indir([divnum])' | $ac_path_M4`  
test $m4out = 0 && ac_cv_path_M4=$ac_path_M4]],  
[AC_MSG_ERROR([could not find m4 that supports indir])])
```

Compiler Features Macros

AC_C_INLINE

- If the C compiler supports the keyword *inline*, do nothing. Otherwise define *inline* to `__inline__` or `__inline` if it accepts one of those

AC_C_VOLATILE

- If the C compiler does not understand the keyword *volatile*, define *volatile* to be empty

AC_CHECK_SIZEOF (TYPE-OR-EXPR)

- Define `SIZEOF_TYPE-OR-EXPR` to be the size in bytes of TYPE-OR-EXPR
- Example:
AC_CHECK_SIZEOF([int *]) defines `SIZEOF_INT_P` to be 8 on DEC Alpha AXP

AC_C_VARARRAYS

- If the C compiler supports variable-length arrays, define `HAVE_C_VARARRAYS`

Library Checks Macros

AC_CHECK_LIB(LIBRARY, F, [ACT-IF-FOUND], [ACT-IF-NOT])

- Check whether LIBRARY exists and contains the function F. Execute ACT-IF-FOUND if it does, ACT-IF-NOT otherwise.
- If ACT-IF-FOUND is not set, the default action is to do `LIBS="-lLIBRARY $LIBS"` and `#define HAVE_LIBLIBRARY`. (automake uses `$LIBS` for linking everything)
- Example:
`AC_CHECK_LIB([nsl], [gethostbyname])`

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

Headers Checks Macros

`AC_CHECK_HEADERS(LIST, [ACT-IF-FOUND], [ACT-IF-NOT])`

- For each system header file X in LIST that exists, define HAVE_X.
- ACT-IF-FOUND can contain additional shell code to execute when one of the header files is found. You can give it a value of 'break' to break out of the loop on the first match

- Example:

```
AC_CHECK_HEADERS([foo.h unistd.h])
```

defines (if found) the variables

HAVE_FOO_H and HAVE_UNISTD_H

Output 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(FILE)

- 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
```

Output Macros

- **AC_OUTPUT**

- **Generate** `config.status` and launch it
- `config.status` performs all the configuration actions and creates all the output files (as defined by the macro `AC_CONFIG_FILES`) and the header files (as defined by the macro `AC_CONFIG_HEADERS`)
- Call this macro once, at the end of `configure.ac`
 - Any line that follows the `AC_OUTPUT` call is executed by `configure` after `config.status` was run

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

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	

Acknowledgements

These slides were 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`