Introduction to Autotools

July 8, 2007

Jim Huang (jserv)
<jserv@openmoko.org>

Overview

- Autoconf, Automake, and libtool working together
- Address portability, configuration needs
- Support GNU Coding Standards
- Provide consistent user experience
- Very popular in Free/Open Source World

The Bad Old Days

- 1990
- Edit Makefile
- Edit source
- Build
- No “make install”

Handy Quote

It it easier to write a portable shell than to write a portable shell script. -- Larry Wall

Useful tools

- colormake
  - make with colorful decoration
- remake
  - modified version of GNU make with debugging and tracing support

The real Problem

- How do we handle platform specific issues?
  - Providing a different Makefile for each architecture
  - Using Autoconf, Automake and Libtool
- The installer needs only
  - Bourne shell
  - C compilers
  - Make program

Some advantages when using GNU autotools

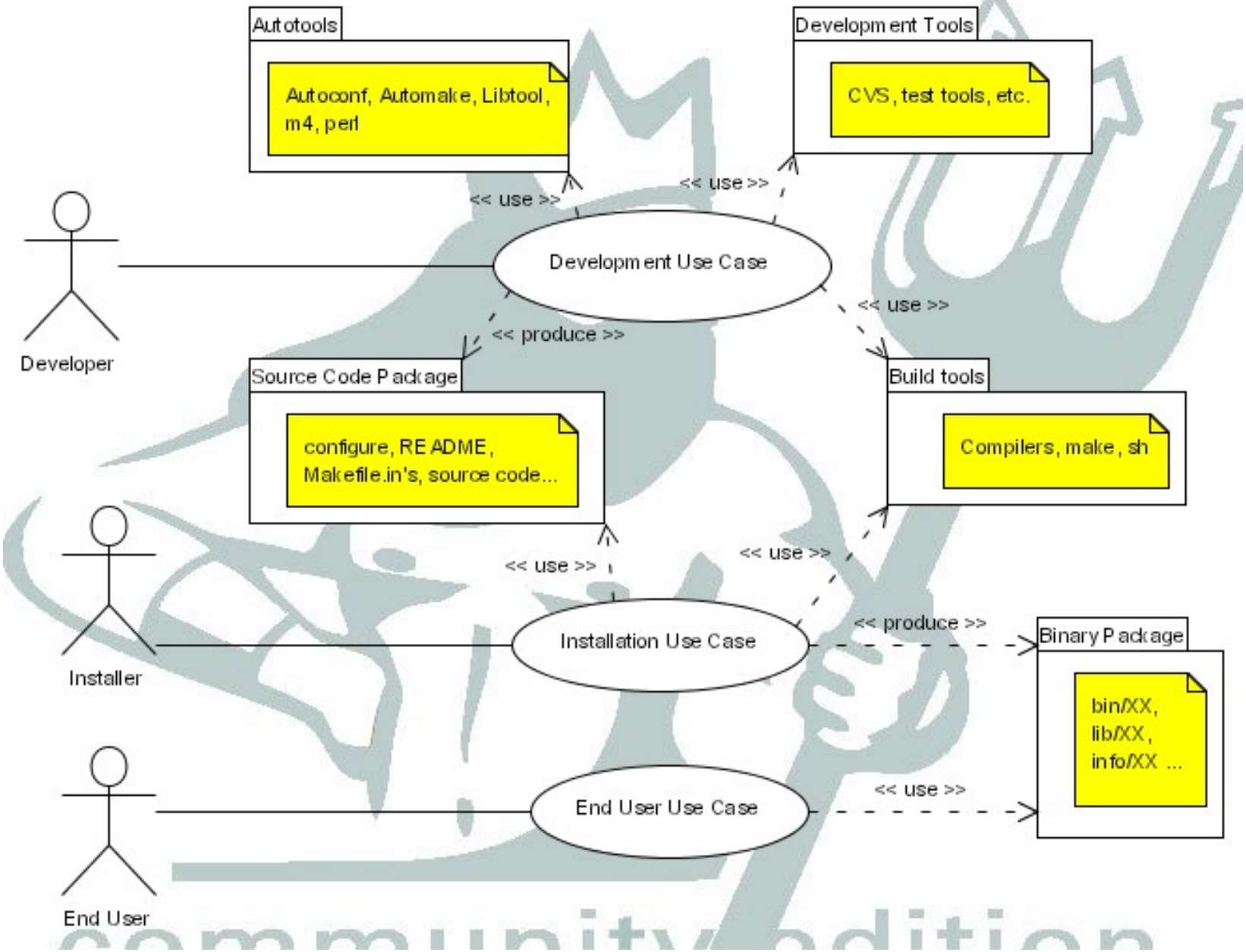
- The installation of a program is straightforward:

```
./configure; make; make install
```

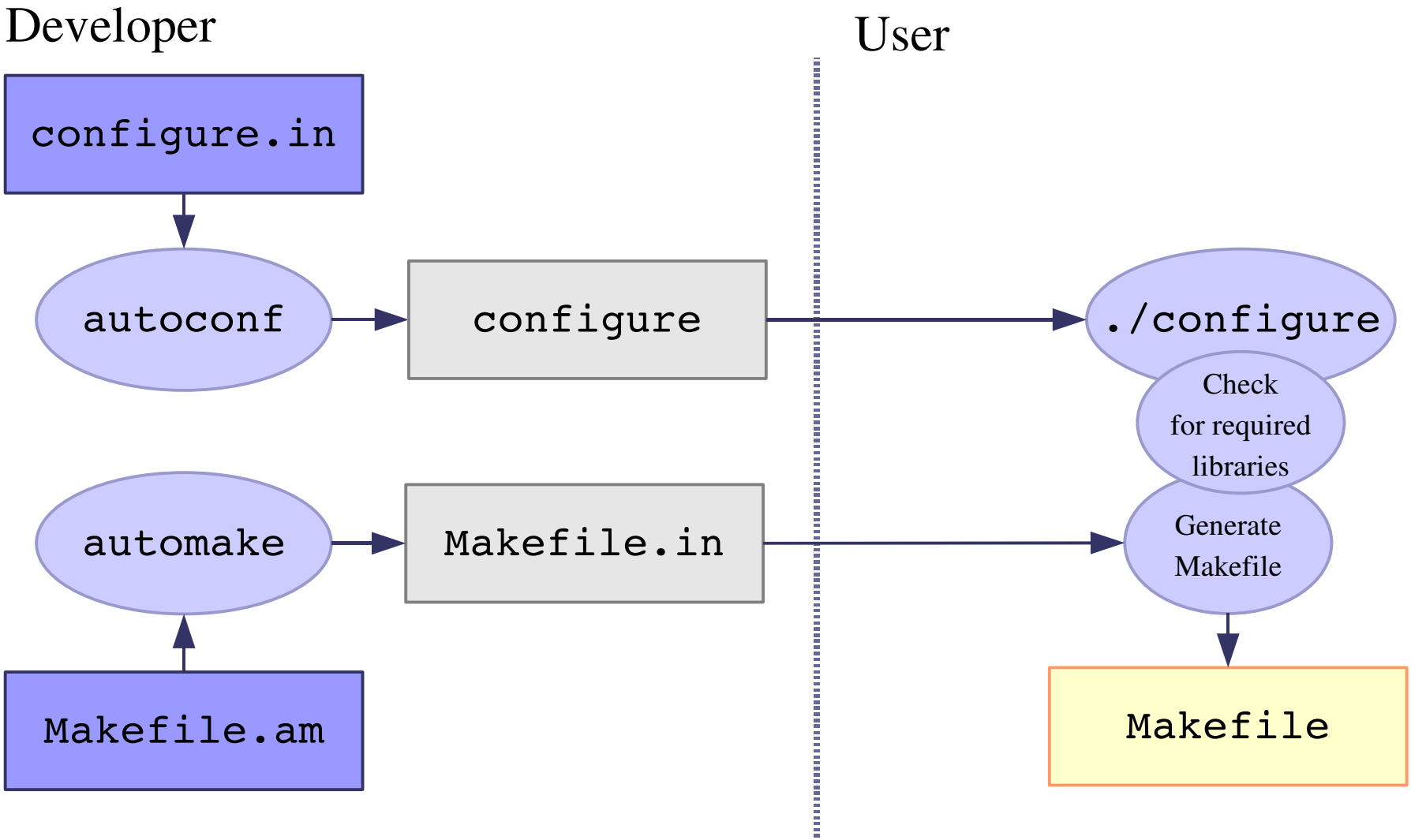
- This procedure checks for system parameters, libraries, location of programs, availability of functions and writes a **Makefile**

- `./configure` supports many options to overwrite defaults settings

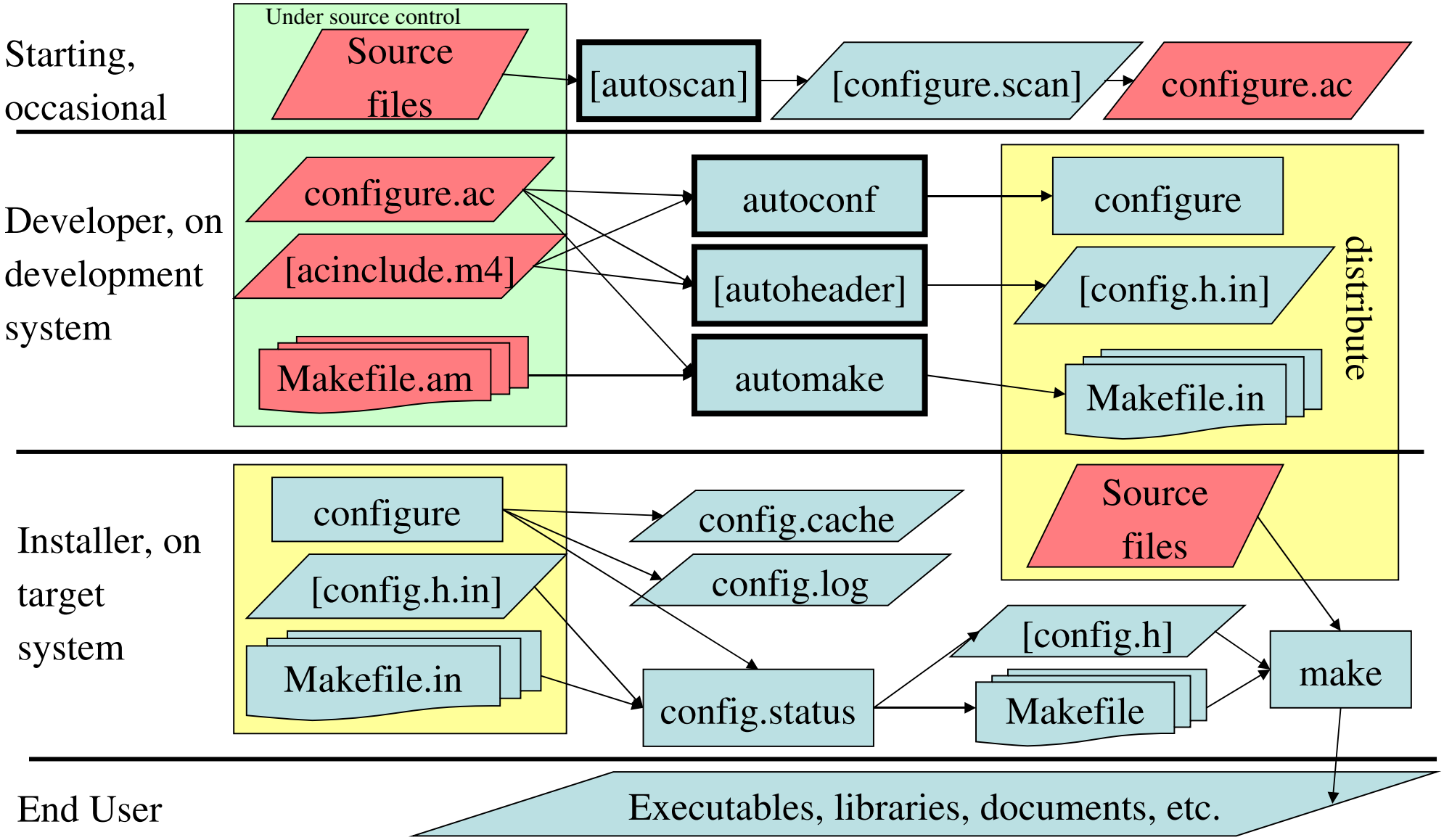
Development Use Cases



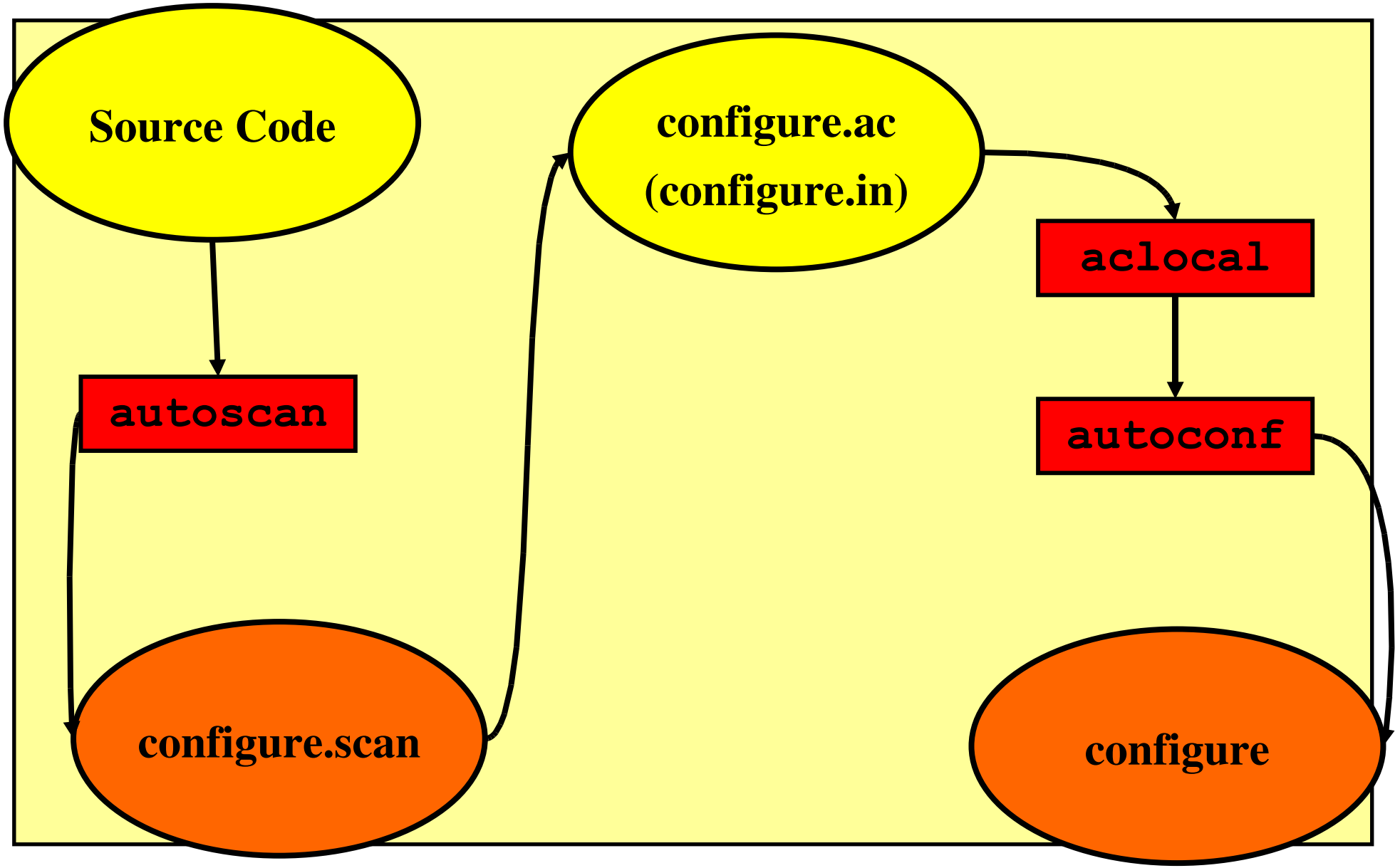
GNU toolchain flow (simplified)



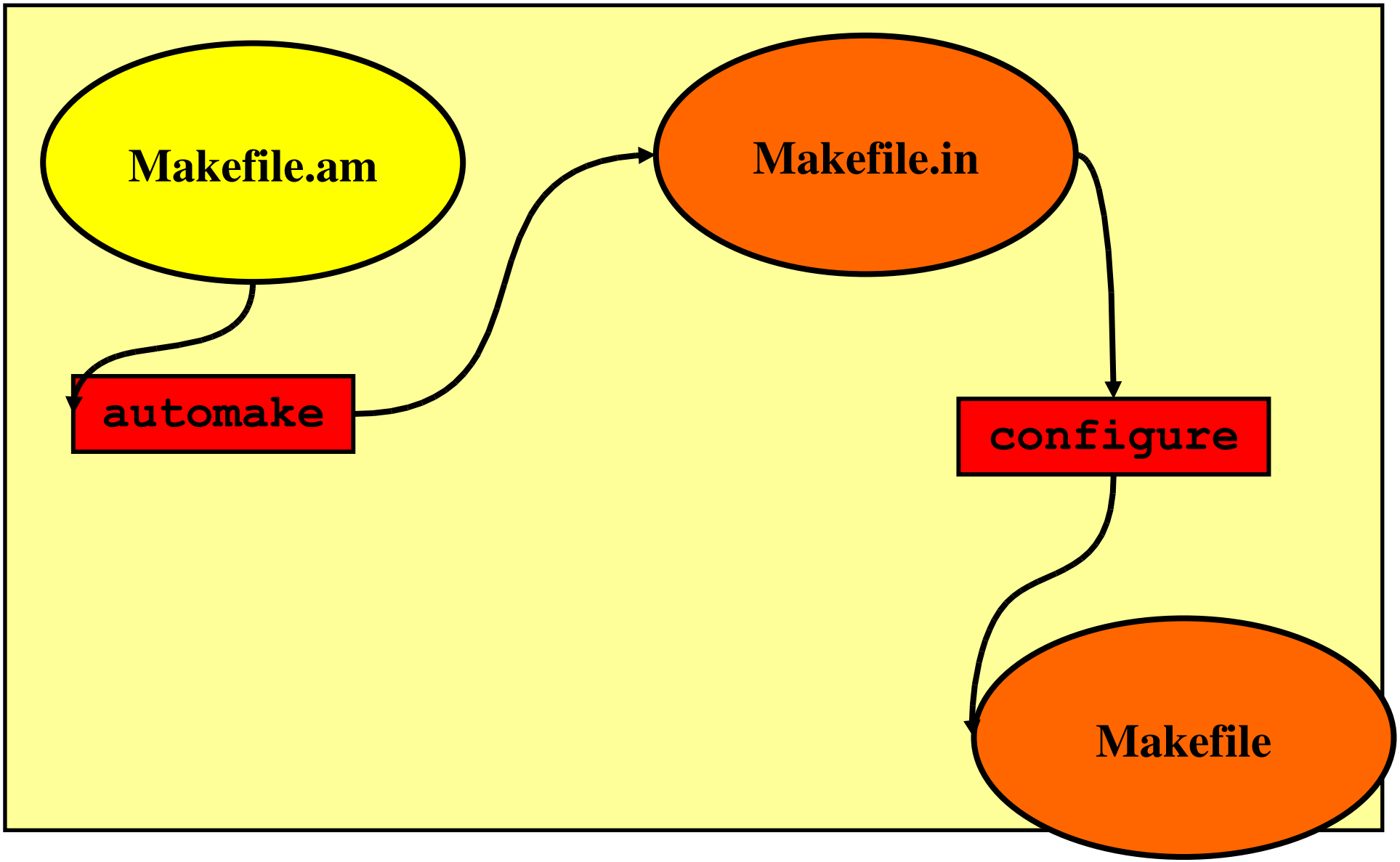
Autotools Overview (no libtool)



GNU autoconf



GNU automake



Autoconf(1)

- Preprocessor based on m4
- Turns configure.ac into configure
- World's most insane shell scripts
  - But portable ones!

Autoconf(2)

- Some philosophy
  - Feature tests, not platform tests
  - The programmer makes the code portable
  - The programmer decides what matters

Autoconf(3)

- Test what is on the actual machine
- Write conclusions into text files
  - Makefile, config.h, etc
- Uniform command line options
  - Standard directories like bindir, includedir
- Large, extensible library of tests

Autoconf Example

```
AC_INIT([amhello], [1.0], [bug-report@address])
AM_INIT_AUTOMAKE([-Wall -Werror])
AC_PROG_CC
AC_CHECK_LIB(z, deflate)

AC_CHECK_HEADERS([unistd.h bstring.h])

AC_CONFIG_HEADERS([config.h])
AC_CONFIG_FILES([Makefile])
AC_OUTPUT
```

Boilerplate

```
AC_INIT([amhello], [1.0], [bug-report@address])
AM_INIT_AUTOMAKE([-Wall -Werror])
AC_PROG_CC
AC_CHECK_LIB(z, deflate)

AC_CHECK_HEADERS([unistd.h bstring.h])

AC_CONFIG_HEADERS([config.h])
AC_CONFIG_FILES([Makefile])
AC_OUTPUT
```

Program Checks

```
AC_INIT([amhello], [1.0], [bug-report@address])
AM_INIT_AUTOMAKE([-Wall -Werror])
AC_PROG_CC
AC_CHECK_LIB(z, deflate)

AC_CHECK_HEADERS([unistd.h bstring.h])

AC_CONFIG_HEADERS([config.h])
AC_CONFIG_FILES([Makefile])
AC_OUTPUT
```

Library Checks

```
AC_INIT([amhello], [1.0], [bug-report@address])
AM_INIT_AUTOMAKE([-Wall -Werror])
AC_PROG_CC
AC_CHECK_LIB(z, deflate)
```

Adds -lz to LIBS; defines HAVE_LIBZ

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

AC_CONFIG_HEADERS([config.h])
AC_CONFIG_FILES([Makefile])
AC_OUTPUT
```

Header Checks

```
AC_INIT([amhello], [1.0], [bug-report@address])
AM_INIT_AUTOMAKE([-Wall -Werror])
AC_PROG_CC
AC_CHECK_LIB(z, deflate)

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

defines HAVE_UNISTD_H, HAVE_BSTRING_H

```
AC_CONFIG_HEADERS([config.h])
AC_CONFIG_FILES([Makefile])
AC_OUTPUT
```

Automake

- Support GNU Coding Standards
- Make common things simple
- Automate some difficult things
- Turns Makefile.am into Makefile.in
- Unrecognized input passed through

Automake Features(1)

- Low-cost, precise automatic dependency tracking
- Non-srcdir builds
- dist/distcheck
- Various clean targets
- install/uninstall
- Parallel builds

Automake Features(2)

- DESTDIR
- Hooks
- Platform integration

Automake Example

```
bin_PROGRAMS = hello

hello_SOURCES = hello.c
```

Install Directory(1)

```
bin_PROGRAMS = hello

hello_SOURCES = hello.c
```

Install Directory(2)

- You can make your own directories
  - mylibdir = $(libdir)/whatever
  - mylib_PROGRAMS = echo

Primary(1)

```
bin_PROGRAMS = hello

hello_SOURCES = hello.c
```

Primary(2)

- There are many primaries
  - PROGRAMS, LIBRARIES, LTLIBRARIES, LISP, PYTHON, JAVA, SCRIPTS, DATA, HEADERS, MANS, TEXINFOS
- You won't use most of them
- Other prefixes control semantics: nobase, nodist
  - nobase_nodist_include_HEADERS = foo/foo.h

Object Naming(1)

```
bin_PROGRAMS = hello

hello_SOURCES = hello.c
```

Object Naming(2)

- Names are made “make-friendly” by automake
  - Weird characters are turned to “_”

Object Suffix(1)

```
bin_PROGRAMS = hello

hello_SOURCES = hello.c
```

Object Suffix(2)

- Object suffixes depend on the type of object
- Typical ones: SOURCES, CFLAGS, LDFLAGS
- SOURCES handles many languages
  - C, C++, Yacc, Lex, assembly, Fortran, Java (with gcj)
  - Also header files

Libtool

- Portable creation of shared libraries
- Integration with autoconf and automake
- Side features
  - libltdl
  - Convenience libraries

Libtool Example(1)

- configure.ac:
  - AC_PROG_LIBTOOL
- That's it!

Libtool Example(2)

```
lib_LTLIBRARIES = libexample.la
libexample_LA_SOURCES = src/example1.c x2.c
```

URLs

- Google Alexandre Duret-Lutz
- http://sources.redhat.com/autobook/
- http://www.gnu.org/software/autoconf/
- http://autoconf-archive.cryp.to/
- http://www.gnu.org/prep/standards/