From Makefiles to Autotools

Yann Pouillon[1,2], Florian Lorenzen[3]

1. Facultad de Químicas, Universidad del País Vasco UPV/EHU, Donostia-San Sebastián, Spain.
2. European Theoretical Spectroscopy Facility (ETSF), Spain.
3. Institut für Theoretische Physik, Freie Universität Berlin

CECAM Tutorial — Zaragoza, Spain
2010/06/23

Building software with the GNU AUTOTOOLS

Building software in an easy way

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

What do the AUTOTOOLS offer?

- Detect platform specificities, compilers, flags, locations, ...
- Provide comprehensive information about errors
- Build, install, and package, programs and libraries
- *De-facto* standard in the free software community

Prerequisites to use an AUTOTOOLS build system

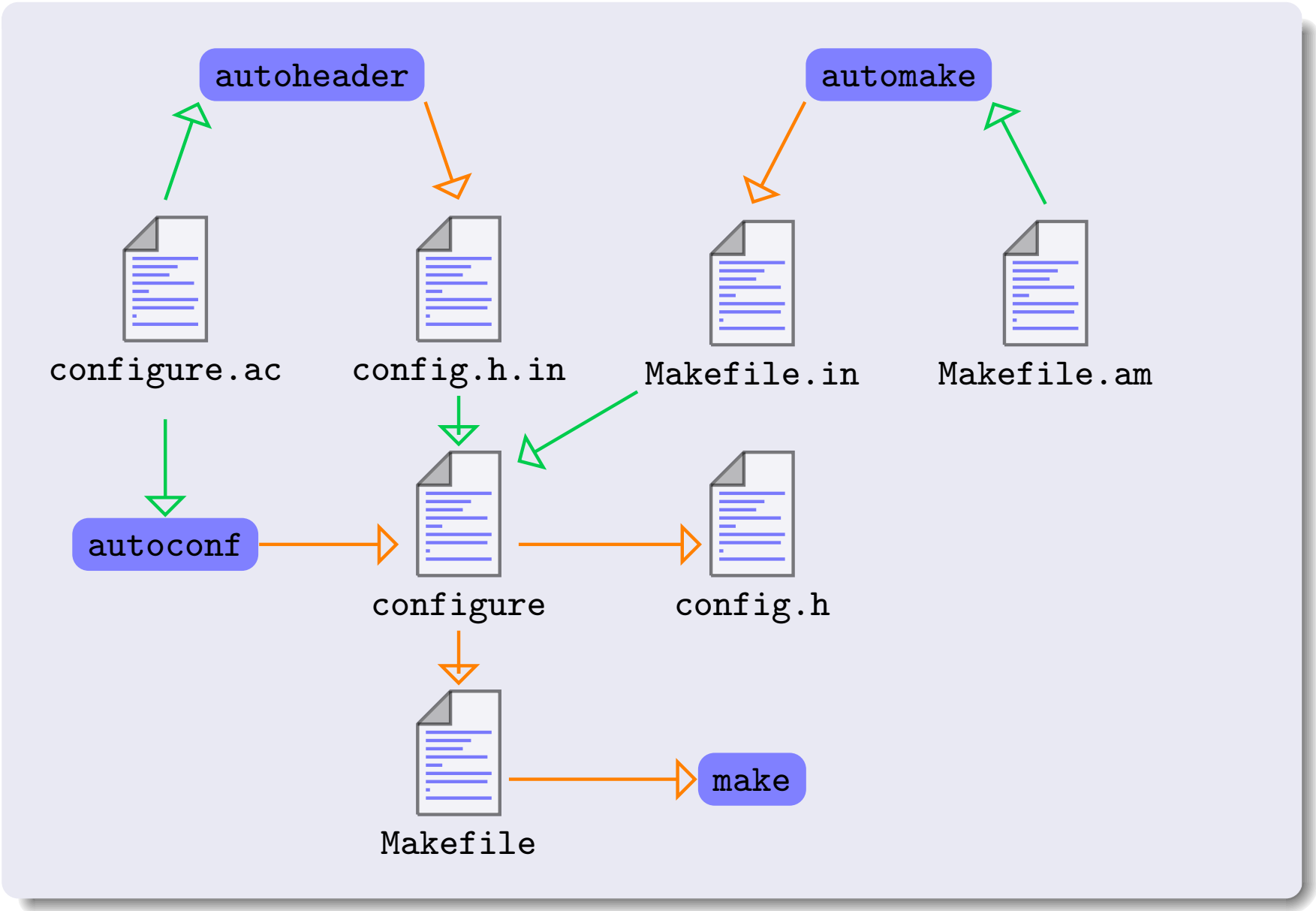
For users

- Standard tools: `sh`, `make`
- Compilers & libraries (e.g. FFTW, LAPACK)

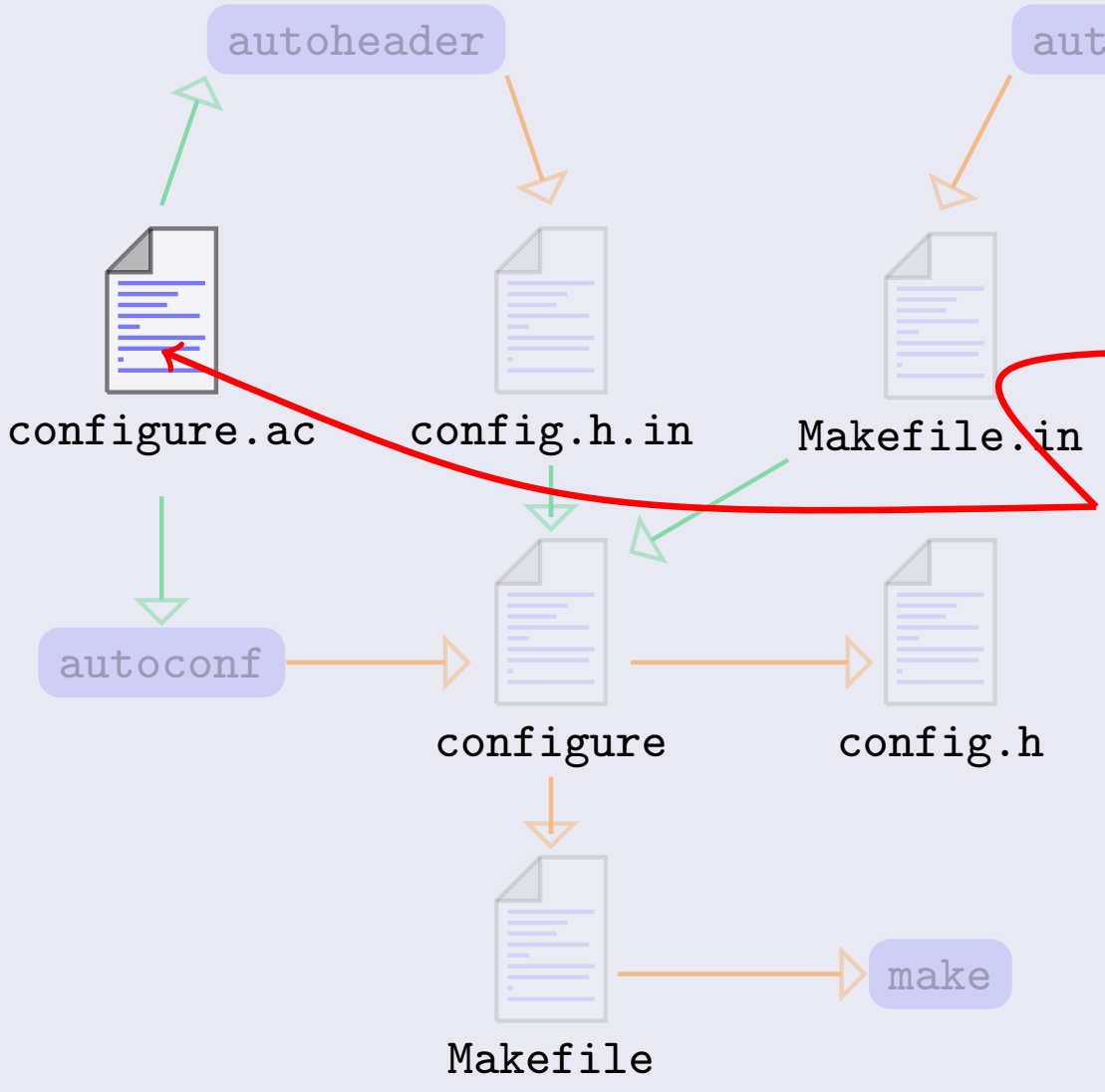
For developers (i. e. for us)

- AUTOCONF, AUTOMAKE, LIBTOOL, M4, PERL, PYTHON
- Only we as developers need the AUTOTOOLS installed!

The big picture



The big picture



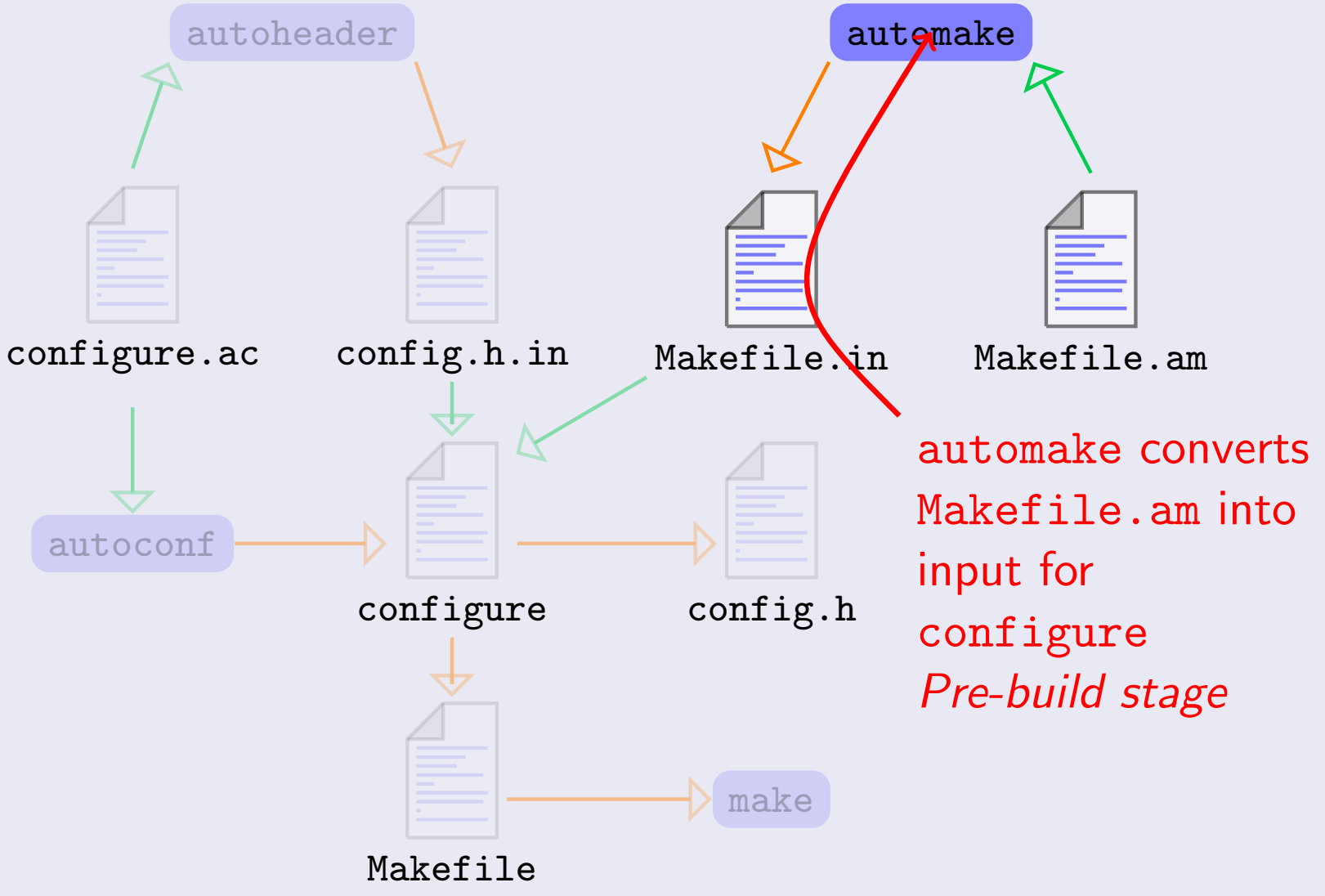
Files we have to write:

`Makefile.am`

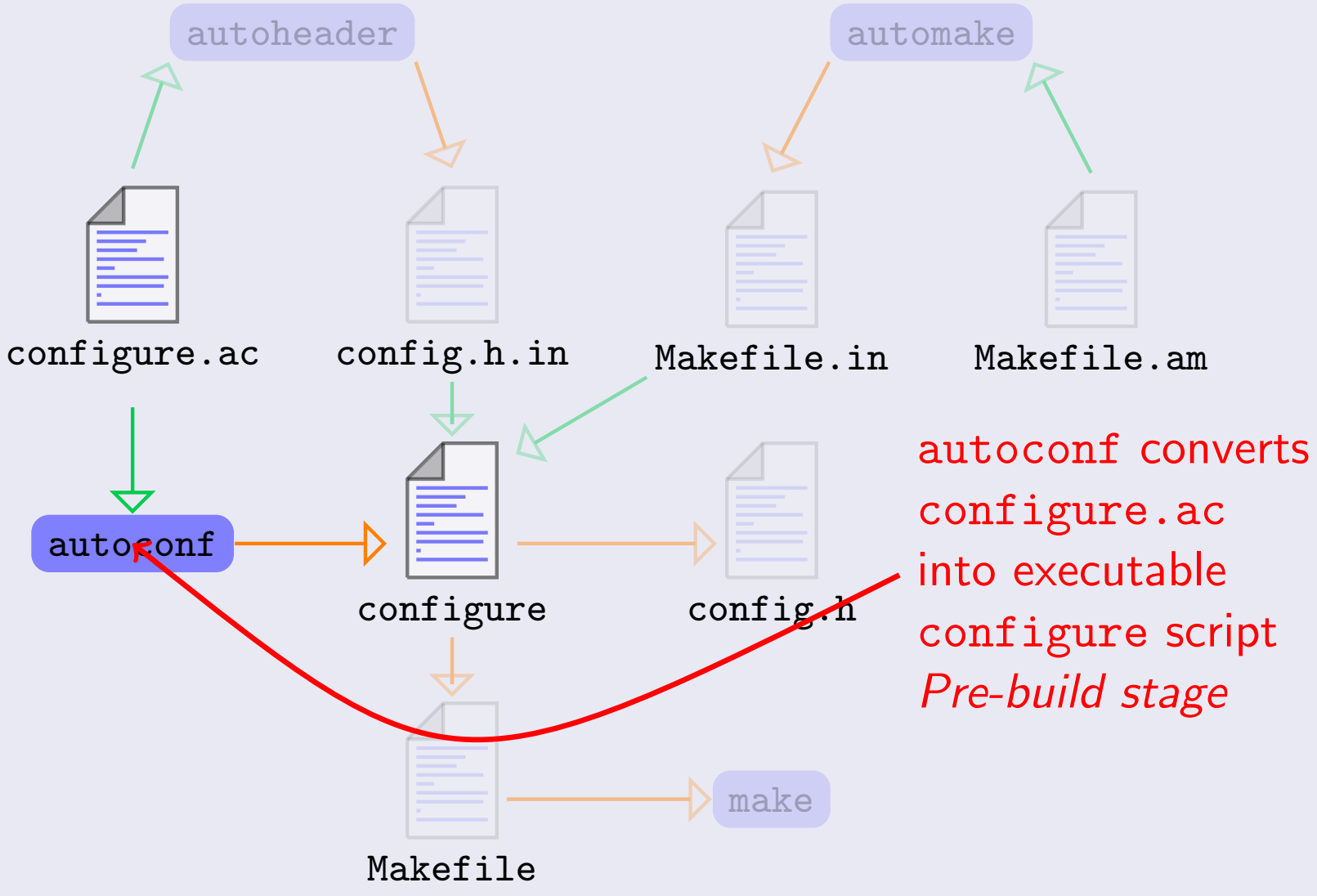
→ how to build `configure.ac`

→ how to configure

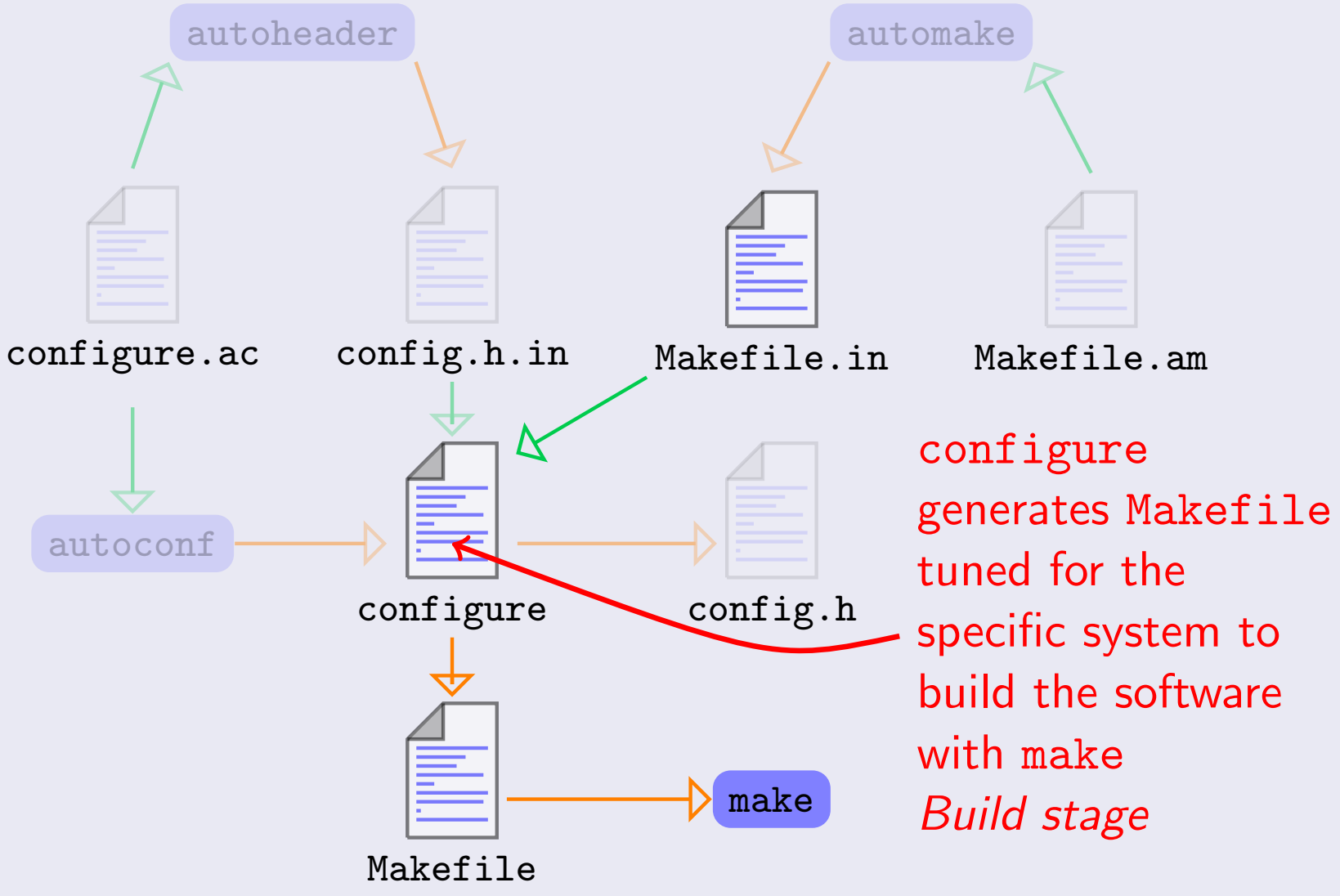
The big picture



The big picture



The big picture



Makefile.am and AUTOMAKE (1)

- `Makefile.am`: high-level template for the actual `Makefile`
- *One* `Makefile.am` per subdirectory
- `automake`: `Makefile.am` → `Makefile.in`
  Still to be processed by `configure`
- Standard targets of `automake` generated `Makefiles`:

`all`	Build the package
`check`	Runs the test suite of the package
`install`	Install the package in the system
`dist`	Prepare a source tarball for distribution
`clean`	Delete built objects

  plus other targets . . .

Makefile.am and AUTOMAKE (2)

Syntax of a Makefile.am

- Makefile.am consists in:
  - variable definitions
  - Makefile snippets (optionally)
- Some variables are transformed into actions

An example for package ebox_c

```
#  Name of the executable(s), installed under $prefix/bin
bin_PROGRAMS = ebox_c

#  Sources to build ebox_c (AUTOMAKE knows how to compile C)
ebox_c_SOURCES = ebox_c.c cg.c cg.h blas.c blas.h ...

#  Libraries to link with, the variables are set by configure
LDADD = $(BLAS_LIBS) $(FLIBS) $(FFTW3_LIBS)
```

Makefile.am and AUTOMAKE (3)

Some important AUTOMAKE variables

PROGRAMS	Program executables to build
LIBRARIES	(Static) libraries to build
SOURCES	Source files
SUBDIRS	Subdirectories with `Makefile.ams`
LDADD	Libraries to link with
DATA	Data files like documentation, datasets, ...

Variables can be prefixed

*name*_	Name of library or executable, e. g. `ebox_c_SRC`
*dir*_	Name of directory, e. g. `bin`, `include`
dist_	Include files in a distribution, e. g. `dist_DATA`

configure.ac and AUTOCONF (1)

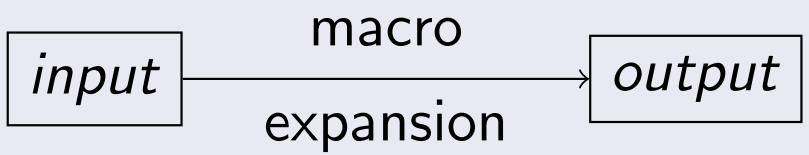
- configure: Bourne shell (sh) script to figure out how to build and install on the current system
- configure.ac
  - Template for configure script
  - Mixture of sh code and M4 macros
- M4 macros provided by AUTOCONF distribution with features like finding a compiler, a library, a header file, handling of command line options, ...
- autoconf converts configure.ac → configure (by M4 macro expansion)
- configure's command line options: set installation directories, compiler flags, enable/disable support for libraries, ...

The M4 macro processor (1)

Why bother about M4?

- Detailed knowledge of M4 not required (usually)
- Basic understanding helps to find errors in `configure.ac` and is required to write custom macros

What is a macro processor doing?

- 

- Macro expansion: replace name of macro by body and substitute arguments
- Rule of thumb: macro $\approx$ function
- Quoting prevents macro expansion

The M4 macro processor (2)

M4 example

- example.m4

```
m4_define([tabecho], [echo -e "$1\t$2"])
m4_define([capitalize], [`echo $1 | tr a-z A-Z`])
tabecho([capitalize([x])], [y])
```

- Quotation marks: [,]
- Arguments to macros: $1, $2, ...
- Macros call: *name*(...)
- Processing with m4 (macro expansion):

```
$ m4 < example.m4
echo -ne "`echo x | tr a-z A-Z`\ty"
```

- Executing with sh:

```
X       y
```

Writing portable `sh` code

Why portable `sh` code matters

- Design goal of AUTOTOOLS: portability
- `configure.ac` → `sh` code
- `sh` code must be portable

Some rules of thumb

- Do not use `[`, use `test` instead
- For tests like `$foo = "bar"`, write `x"$foo" = x"bar"`, `$foo` might be empty
- Do not use shell functions and aliases
- Do not use extension of `bash`, `ksh`, or `tcsh`
- Read chapter *Portable Shell* of the AUTOCONF manual

Guidelines to write a `configure.ac`

- Check for each external package (using `--with-`*package*)
- Check for commands except basic ones like `cp`, `rm`, ...
- Abort `configure` with *understandable* error messages
- Look for specific checks:
  - AUTOCONF macros, listed in chapter *Existing Tests* of the AUTOCONF manual
  - AUTOCONF macro archives at
    `http://ac-archive.sourceforge.net/`
    `http://autoconf-archive.cryp.to/`
- Last resort: write your own macro
  - Take a similar macro as starting point
  - Macro should be self-contained (for future reuse)

Tips and tricks

When `configure` does not work

- `config.log`: list of all `configure` actions
- Look at the generated `configure`

Tools to make life easier

`autoheader` Generate `config.h.in` automatically

`autoreconf` Runs `autoconf`, `autoheader`, and `automake` in correct order if some of the source files changed (the *preferred* way to run these tools)

`autoscan` Produces a preliminary `configure.ac` by examining the project's source

References

Software manuals

- GNU AUTOCONF manual: http://www.gnu.org/software/autoconf/manual/
- GNU AUTOMAKE manual: http://www.gnu.org/software/automake/manual/
- GNU LIBTOOL manual: http://www.gnu.org/software/libtool/manual/
- GNU M4 manual: http://www.gnu.org/software/m4/manual/

Books/Tutorials

- G. V. Vaughan, B. Elliston, T. Tromey, and I. L. Taylor. GNU AUTOCONF, AUTOMAKE and LIBTOOL. New Riders, 2000. http://http://sourceware.org/autobook/
- Felipe Bergo. autotut: Using GNU auto{conf,make,header}. http://seul.org/docs/autotut/