Software building II

http://d3s.mff.cuni.cz

Department of Distributed and Dependable Systems

D3S

Tomáš Kalibera,
Tomáš Pop,
Peter Libič

CHARLES UNIVERSITY IN PRAGUE

faculty of mathematics and physics

Plan

- SW building
  - Requirements
  - Typical scenarios
  - Dependency problems
  - ....
- Tools
  - Make
  - Ant

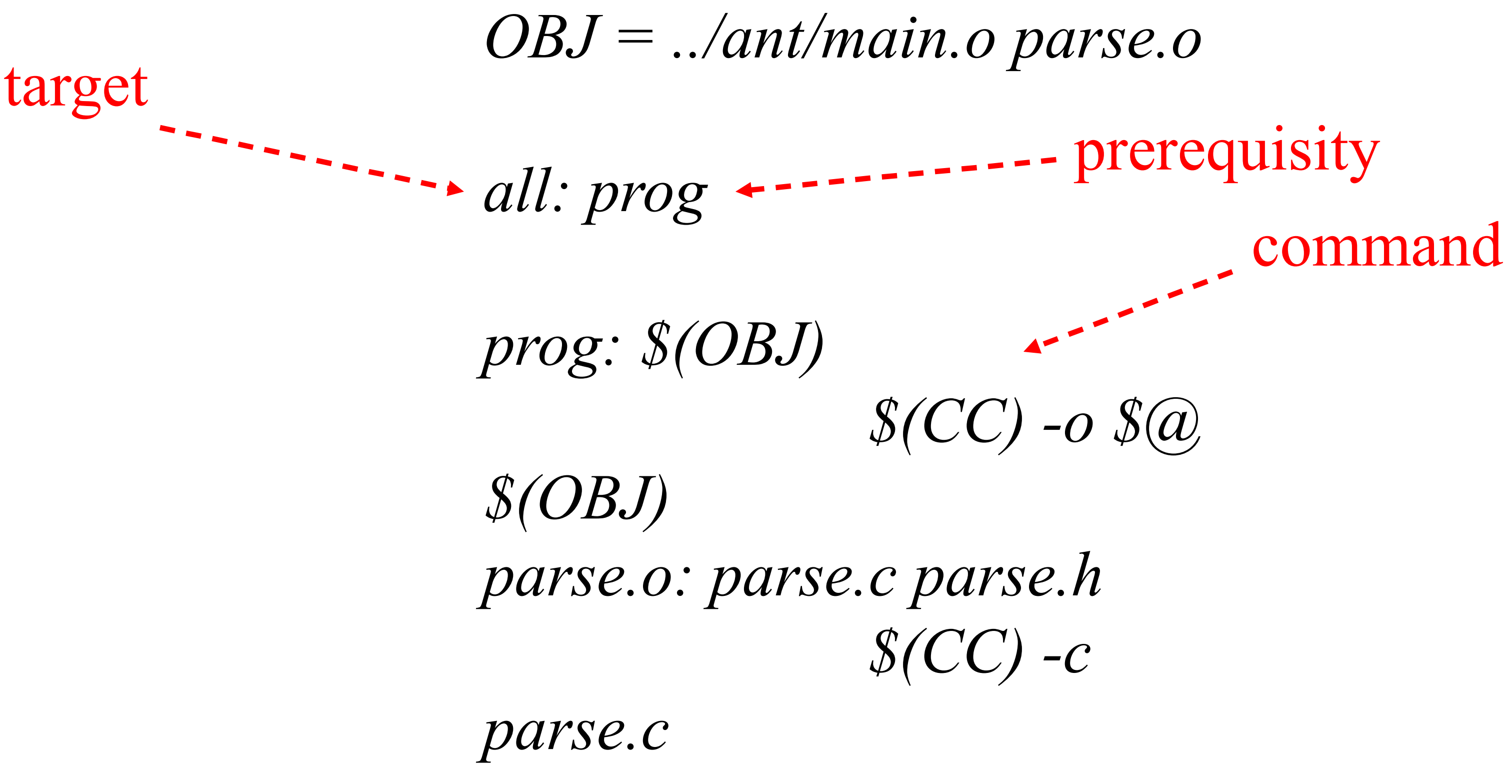
Previous lecture

  - Make – advanced options
  - Autotools {conf, make, scan,..}
  - Ivy, Maven, SCons...

Today's lecture

Department of Distributed and Dependable Systems

Makefile: basic example



```
OBJ = ../ant/main.o parse.o

all: prog

prog: $(OBJ)
                    $(CC) -o $@
$(OBJ)
parse.o: parse.c parse.h
                    $(CC) -c
parse.c
```

target

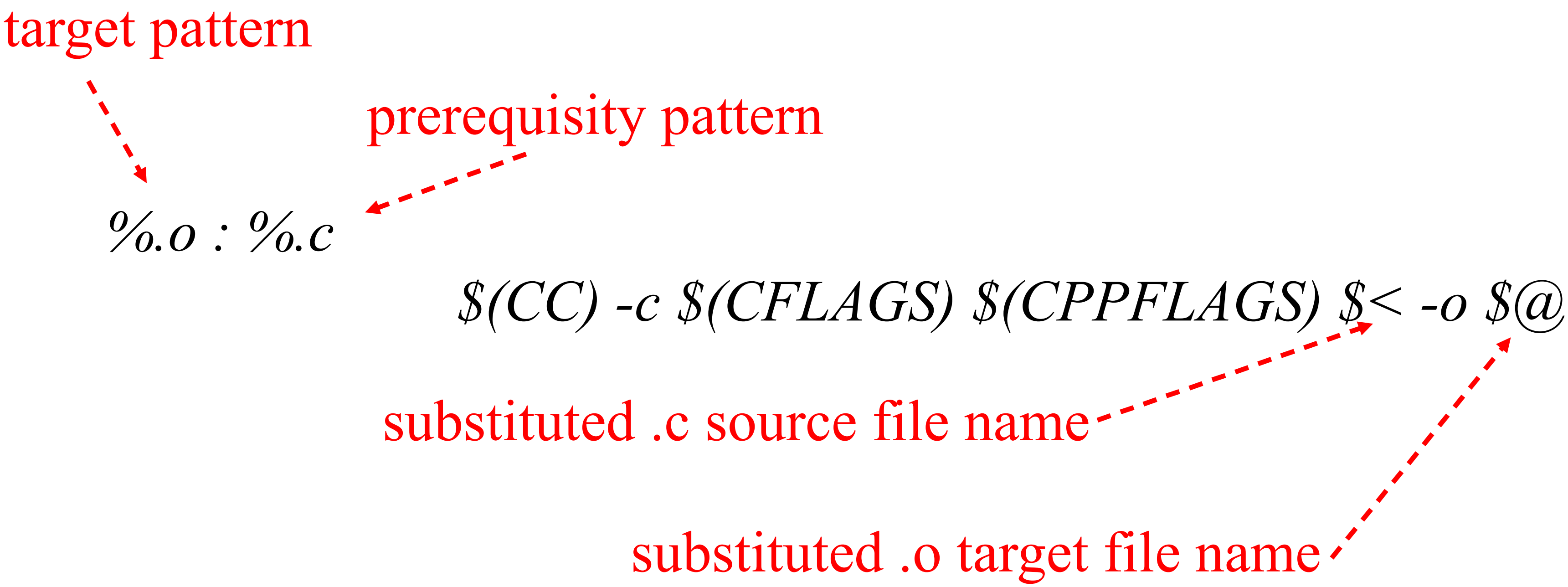
prerequisity

command

Example from Peter Miller’s article “Recursive Make Considered Harmful”

Make: advanced topics and tricks

(Implicit) pattern rules



target pattern

prerequisity pattern

```
%.o : %.c
        $(CC) -c $(CFLAGS) $(CPPFLAGS) $< -o $@
```

substituted .c source file name

substituted .o target file name

- Applies if
  - There is no other rule with a command for this target
  - If the prerequisity exists

Static pattern rules

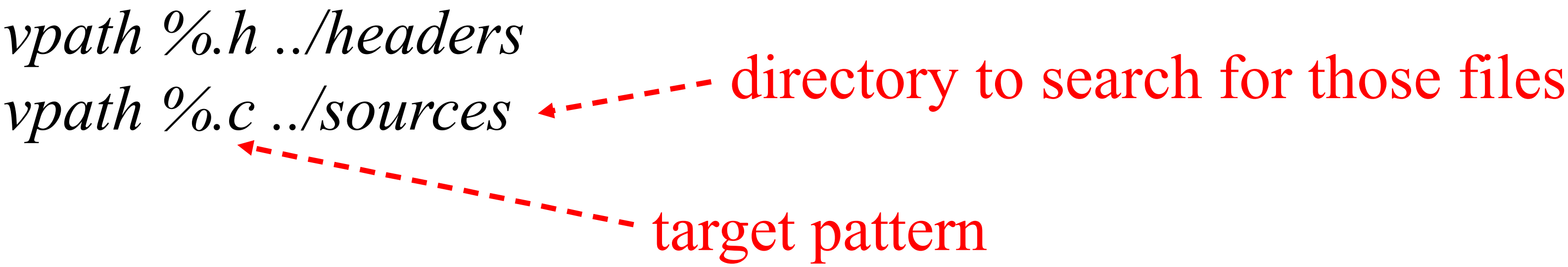
```
objects = foo.o bar.o

$(objects): %.o: %.c
                $(CC) -c $(CFLAGS) $< -o $@
```

list of targets

- Used to construct multiple targets in the same way
- Have precedence over implicit rules, including the built-in ones
- Prerequisities do not have to exist - they can be potentially re-made

Search path

```
VPATH = src:../include
```



```
vpath %.h ../headers
vpath %.c ../sources
```

directory to search for those files

target pattern

- Files are always generated in the directory specified in the Makefile
- Files are, if not present on the specified locations, looked up in the locations from vpath or VPATH
  - Automatic variables $< and $^ are set to true file locations

Generating parts of make files (i.e. deps)

```
sources = foo.c bar.c
...
include $(sources:.c=.d)

%.d: %.c
  sed magic to generate .d files
  using cc –M (or gcc –MM)
```

update files foo.d and bar.d, include them into this makefile and re-read this makefile

command to generate the .d files from C sources

Suggested form of dependencies (file **main.d**)

```
main.o main.d: main.h main.c
```

(GNU) Makefile – automatic variables

Variable	Meaning
$@	target
$<	First prereqisity
$^	All prerequsities
…	
$(@D)	Directory part of the target
…	

```
prog: $(OBJ)
        $(CC) -o $@ $^
parse.o: parse.c parse.h
        $(CC) -c $<
```

- There are many other
    - http://www.gnu.org/software/make/manual/make.html#Automatic-Variables

Portability

Portability issues in C programs

- Why ?
  - The standard in not always obeyed
  - The standard does not specify everything (something is “implementation defined”)
  - The standard only covers a subset of functionality required in many applications
  - Some systems have bugs
  - The standard definitely does not cover compilation and building (build system, compilation options, etc)

Portability issues in C programs

- There are surprisingly many compatibility issues in C on different platforms, such as
  - exit() - returns void, int
  - free(NULL) – does nothing, does not work
  - isinf, isnan – sometimes macros, sometimes functions, in different headers and libraries
  - malloc(0) – returns NULL or valid pointer
  - setenv – prefered over putenv, but sometimes only putenv is available, and it sometimes does not work
  - Many more…

Portability issues in shell and utilities

- Shell and utilities are used at least for compilation of C programs
  - Awk – many different implementations, not all are compatible
  - Grep – many implementations with different features (-e, -E, -F are not always present)
  - Lex, yacc – multiple implementations
  - Sed – POSIX conformance issues, length limits

Solutions

- Virtualized environment
  - C# (resembles C), Java
  - C# runtimes are not feature-complete everywhere
    - only on Windows have all libraries typical for today’s programming environments
- Explicit support for different platforms
  - Knowing all portability issues
  - Resolving them explicitly
    - C preprocessor macros
    - Lowest common denominator shell scripts and utilities

Solutions

- Virtualized environment
  - C# (resembles C), Java
  - C# runtimes are not feature-complete everywhere
    - only on Windows have all libraries typical for today’s programming environments
- Explicit support for different platforms
  - Knowing all portability issues
  - Resolving them explicitly
    - C preprocessor macros
    - Lowest common denominator shell s [illegible] d utilities

GNU Build System

Autoconfig & related tools overview

End user’s perspective

- Download
- “./configure”
  - Automatically test the target system
    - If some important library is missing, it has to be installed manually, however
  - Automatically generate Makefile
- “make”
- “make install”

End user’s perspective

- Download
- “./configure”
  - Automatically test the target system
    - If some important library is missing, it has to be installed manually, however
  - Automatically generate Makefile
- “make”
- “make install”

Very easy to use

More end user’s features

- Installation root directory
  - “configure --prefix”, “configure --exec-prefix”
- Cross-compilation
  - “configure --host”
- Optional features of the software
  - “configure --enable-FEATURE”,

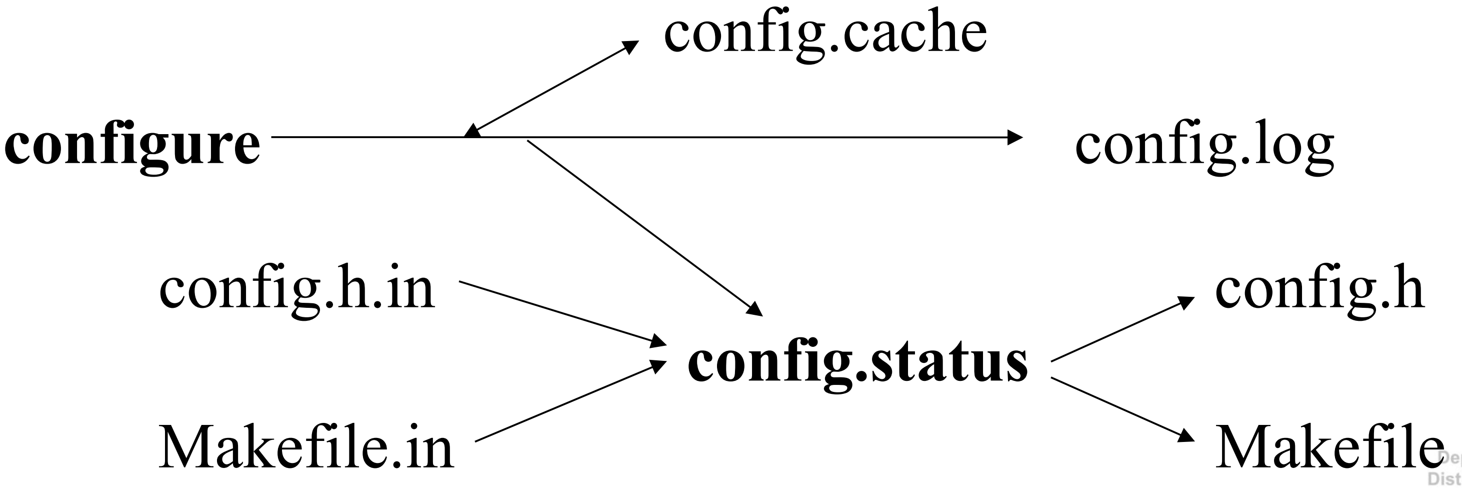
    i.e. “--enable-debug
- Optional packages (libraries) to build with
  - “configure --with-PACKAGE”

More end user's features

- Help (application specific)
  - "configure --help"
- Separate build trees
  - " cd $BUILD ; $SRC/configure ; make "

What is behind the scenes

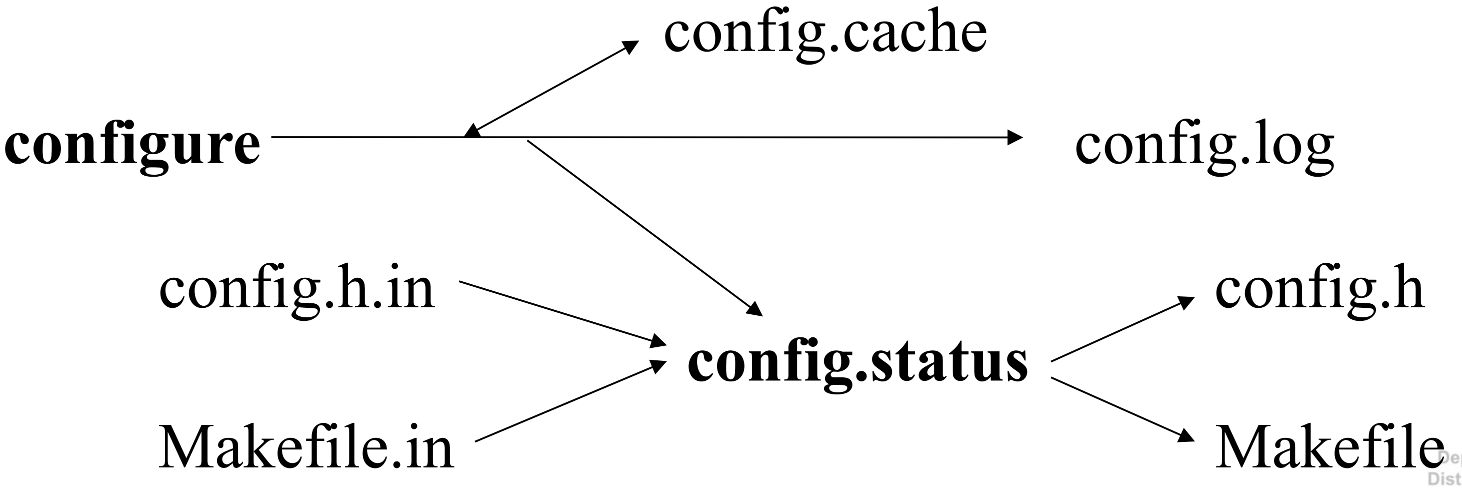
- “./configure”
  - Automatically tests the target system; if some test fails (i.e. important library is missing), it has to be resolved manually
  - Automatically generates Makefiles



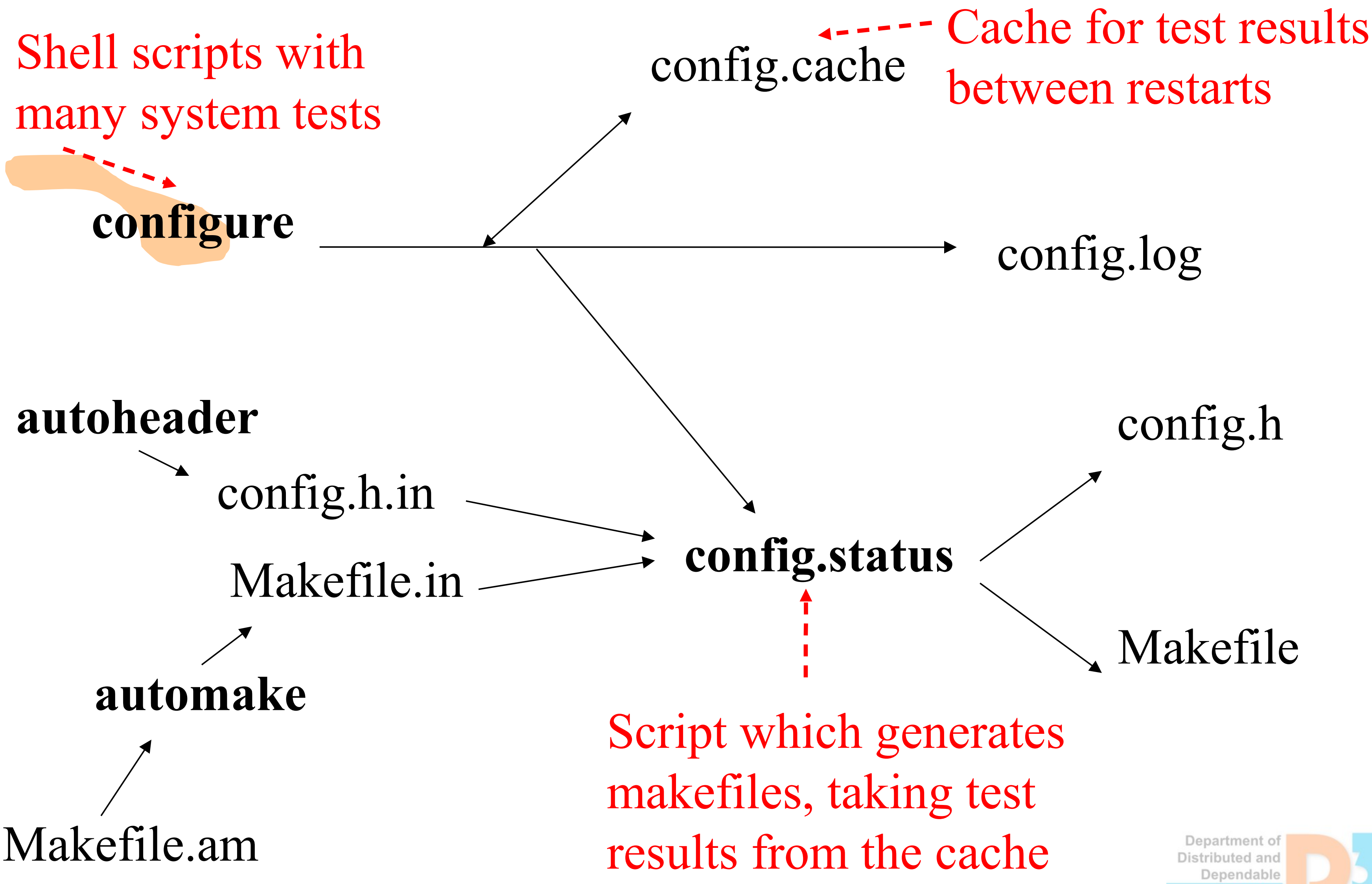
What is behind the scen

Rather complex to comprehend

- "./configure"
  - Automatically tests the target system; if some test fails (i.e. important library is missing), it has to be resolved manually
  - Automatically generates Makefiles



What is behind the scenes



Shell scripts with many system tests

configure

config.cache

Cache for test results between restarts

config.log

autoheader

config.h.in

Makefile.in

automake

Makefile.am

config.status

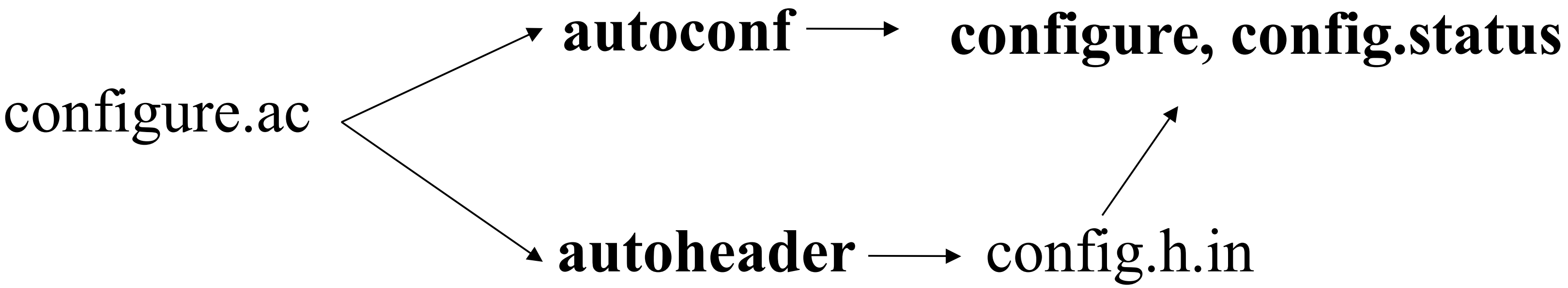
config.h

Makefile

Script which generates makefiles, taking test results from the cache

“configure” script

“configure” script



configure.ac

autoconf

configure, config.status

autoheader

config.h.in

- “configure” is a very portable shell script
  - Uses lowest-common-denominator of known shells (no functions, …)
  - Generated from a template, based on a library of known tests of known issues

“configure” script template structure

AC_INIT(package, version, bug-report-address)

information on the package
checks for programs
checks for libraries
checks for header files
checks for types
checks for structures
checks for compiler characteristics
checks for library functions
checks for system services

AC_CONFIG_FILES([file...])
AC_OUTPUT

Configure.ac example snippet

```
AC_INIT([GNU cflow], [1.2], [bug-cflow@gnu.org])
AC_CONFIG_HEADER([config.h])

# Checks for programs.
AC_PROG_CC
AC_PROG_LEX

# Checks for header files.
AC_HEADER_STDC
AC_CHECK_HEADERS([stdlib.h string.h unistd.h locale.h])

AC_OUTPUT
```

“configure” script language: M4

- History
  - Macro language designed by Kernighan and Ritchie, 1977
  - Re-implemented as GNU project
- Processing scheme
  1. Macro arguments are processed
  2. Macro is expanded
  3. The expansion output is processed

“configure” script language: M4

- Basic implications for configure.ac
  - There are not types, just text and macros
  - Arguments of macro calls must be quoted (‘[‘,’]’), even recursively
  - Arguments that should be taken as literal strings must be (at least sometimes) double-quoted (‘[[‘,’]]’)

Configure.ac example snippet

```
AC_INIT([GNU cflow], [1.2], [bug-cflow@gnu.org])
AC_CONFIG_HEADER([config.h])

# Checks for programs.
AC_PROG_CC
AC_PROG_LEX

# Checks for header files.
AC_HEADER_STDC
AC_CHECK_HEADERS([stdlib.h string.h unistd.h locale.h])

AC_OUTPUT
```

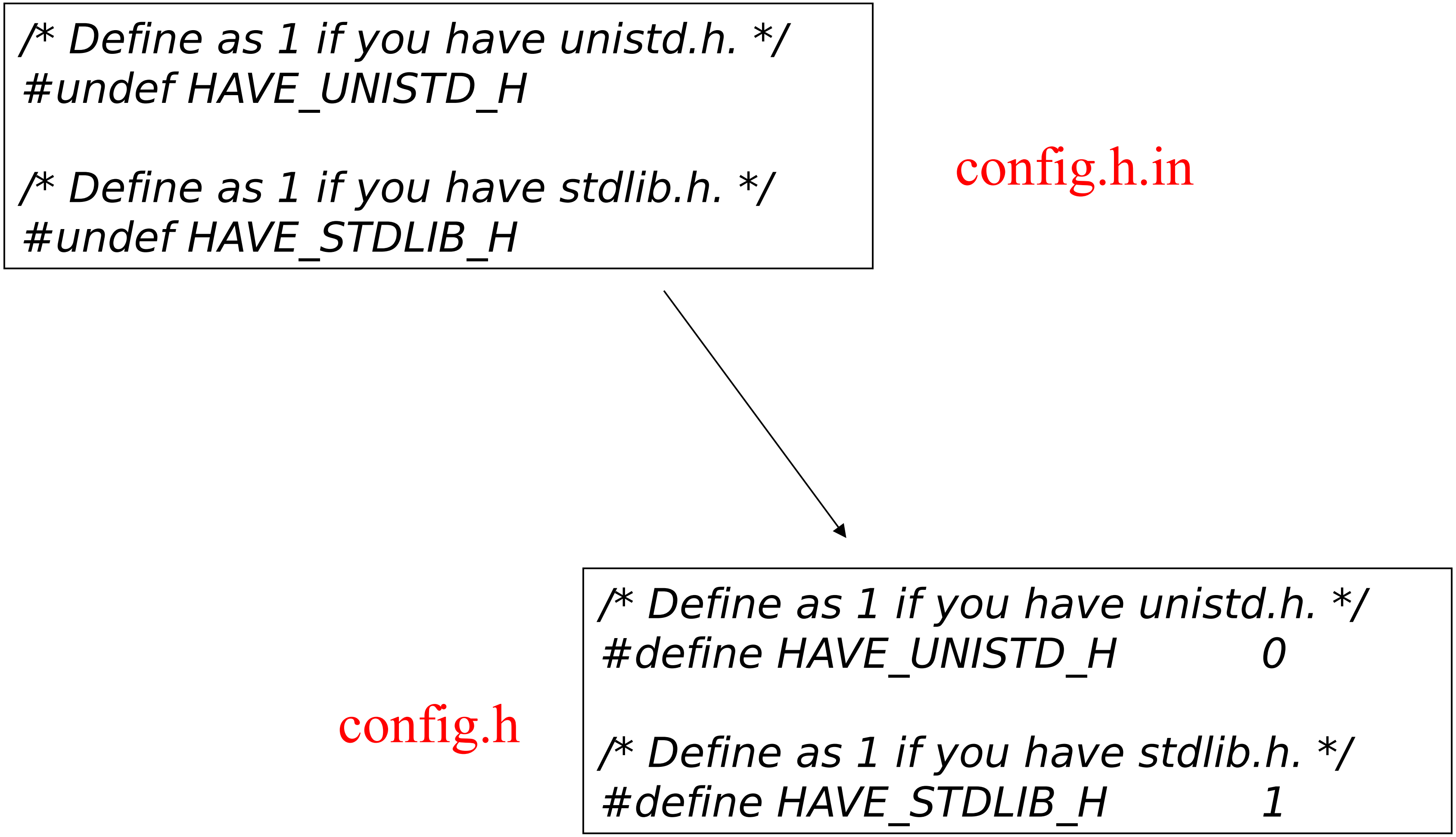
Multiple arguments to a macro

Macros

A single argument to a macro

Achieving “multi-platformness” in C

Platform info in generated config.h



```
/* Define as 1 if you have unistd.h. */
#undef HAVE_UNISTD_H

/* Define as 1 if you have stdlib.h. */
#undef HAVE_STDLIB_H
```

config.h.in

```
/* Define as 1 if you have unistd.h. */
#define HAVE_UNISTD_H           0

/* Define as 1 if you have stdlib.h. */
#define HAVE_STDLIB_H           1
```

config.h

Using config.h in the C sources

Created by configure
and config.status

program source file

```
#include <config.h>

#if HAVE_LOCALE_H
# include <locale.h>
#endif
#if !HAVE_SETLOCALE
# define setlocale(category, locale) /* empty */
#endif
...
```

defined in config.h

Creating configure.ac

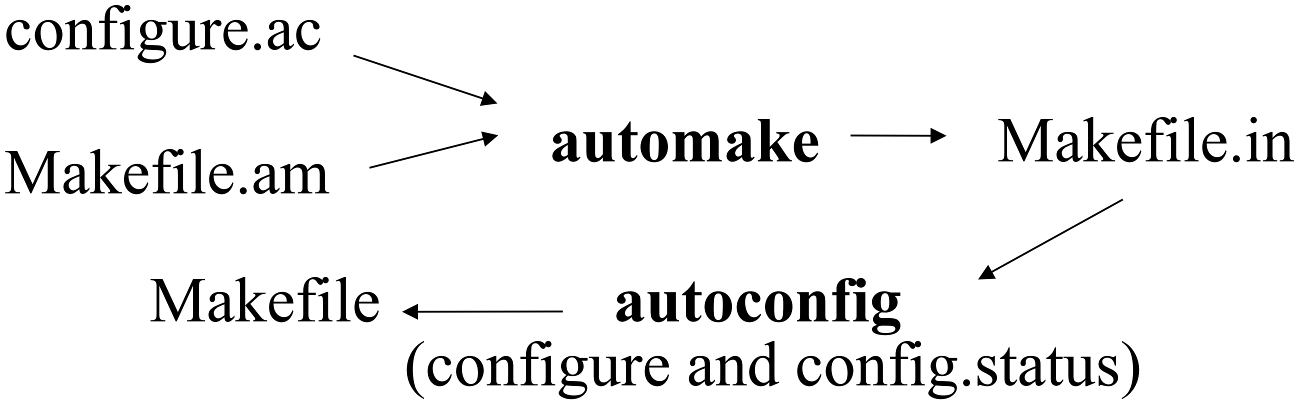
- Autoscan
  - Analyzes C sources for common portability problems
  - Generates a skeleton for corresponding configure.ac file
  - The analysis is very simplistic – just looking for pre-defined symbols
- Ifnames
  - Reports variables used in C preprocessor conditionals (which are often means to solve platform dependency issues)

Creating portable Makefiles easily

Variable substitutions (“configure”)

- Syntax
  - @variable@ substituted for variable_value
- Typically used variables
  - Values come from “configure” command line arguments, the tests and GNU conventions
  - Compiler, flags, etc – CC, CFLAGS, CPPFLAGS
  - Directories – srcdir, top_srcdir, includedir, …
  - Available tools – awk, sed, echo
  - Availability of libraries and header files
  - Sizes of certain types

Automake makefile generator

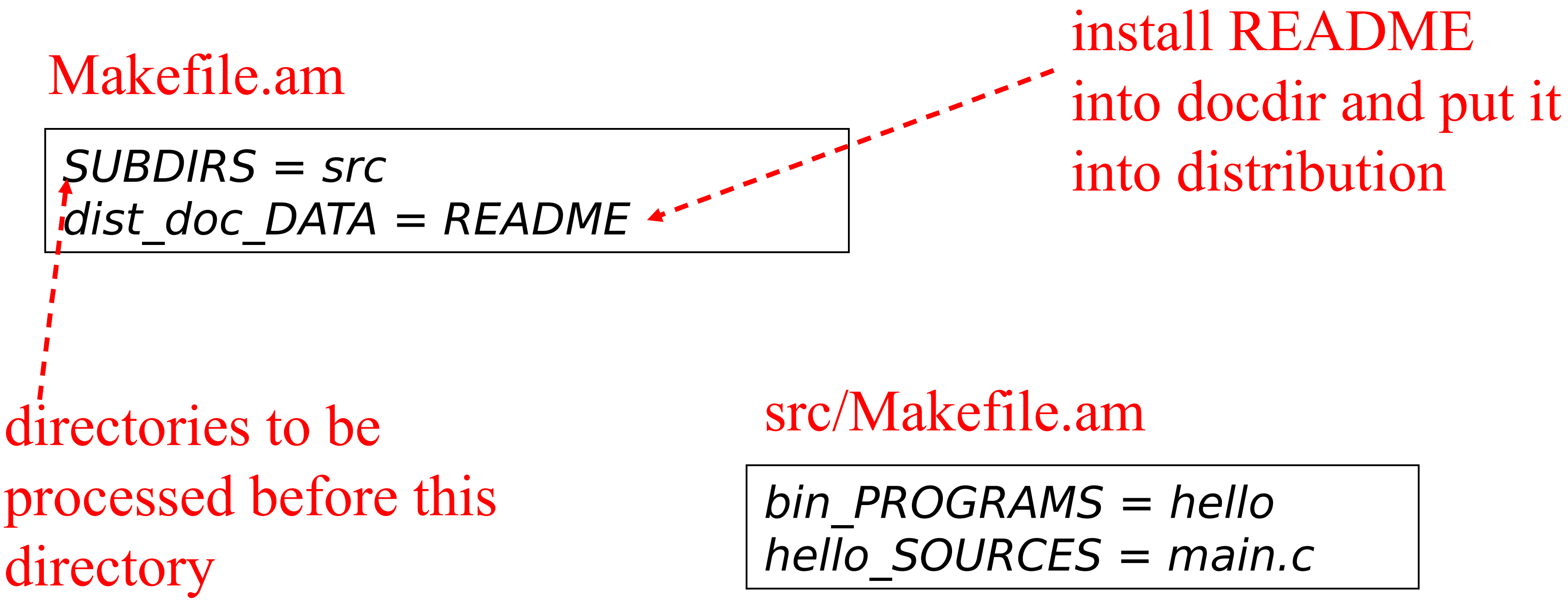


- Backward compatibility
  - Uses only features available in most make implementations, not just GNU Make
  - Warns about potential incompatibilities of constructs that would otherwise get directly into generated makefiles

Automake Makefile generator

- Support for a wide range of targets
  - install, install-exec, install-data
  - install-strip (no debug symbols)
  - uninstall
  - clean
  - distclean (clean to what is distributed - remove also files generated by configure)
  - check (run test of compiled binaries)
  - installcheck (run test of installed program)
  - dist – create source code distribution package (tarball)

Automake MF template (Makefile.am)



Makefile.am

```
SUBDIRS = src
dist_doc_DATA = README
```

install README into docdir and put it into distribution

directories to be processed before this directory

src/Makefile.am

```
bin_PROGRAMS = hello
hello_SOURCES = main.c
```

- Syntax
  - Similar to make, but pre-defined meaning of some variables
- Parts with no special meaning are copied to Makefile

Automake MF template (Makefile.am)

Makefile.am

```
SUBDIRS = src
dist_doc_DATA = README
```

“hello” is a program and should be installed into bindir

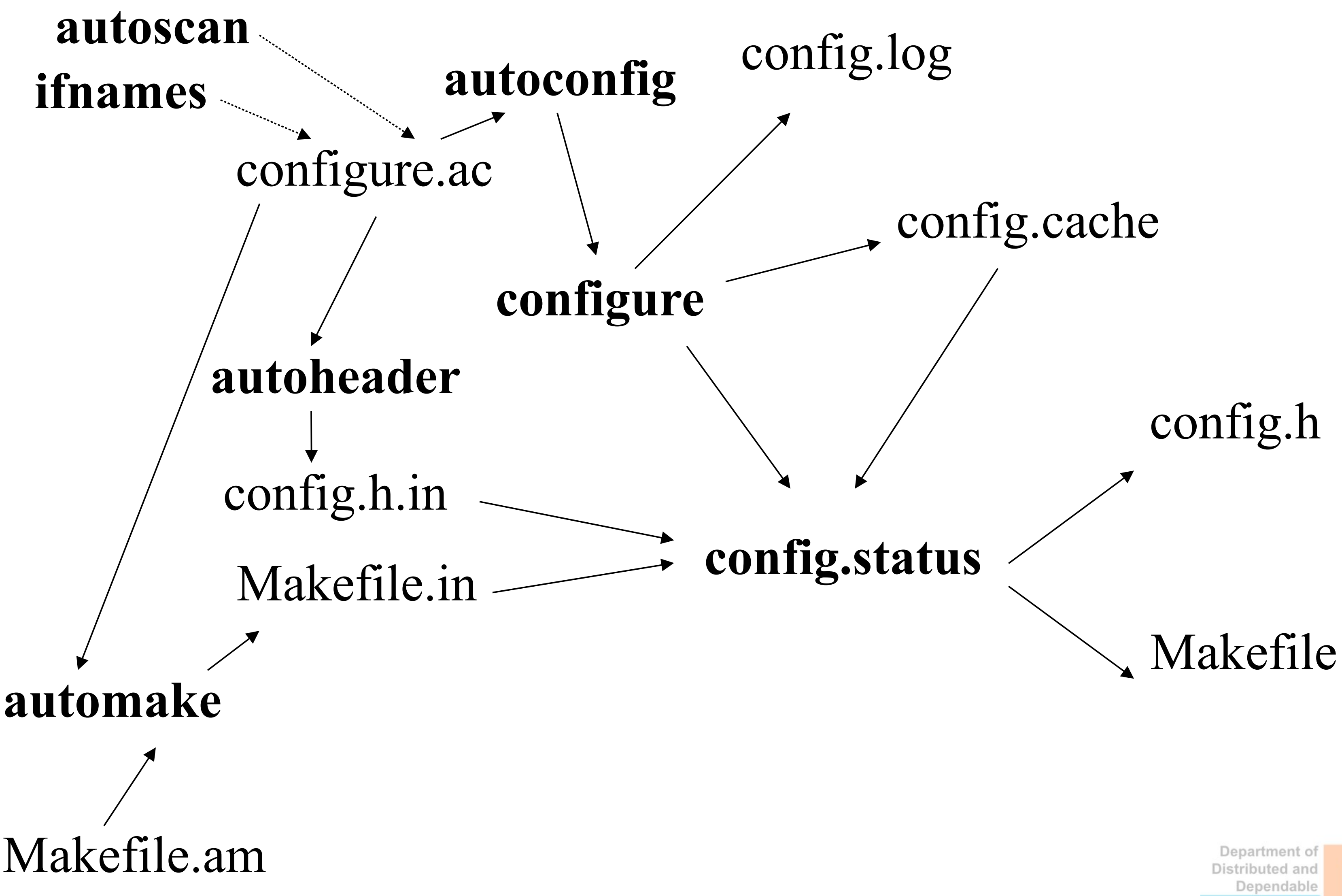
src/Makefile.am

```
bin_PROGRAMS = hello
hello_SOURCES = main.c
```

program “hello” can be built from source “main.c”

Since “main.c” is a source, it will also be put into distribution (make dist)

GNU Build Tools



Generating all the files

- Files typically distributed with an application
  - Makefile.in (user then does not need automake)
  - config.h.in
  - configure
  - The sources as well, since this is open-source world (Makefile.am, configure.ac)
- Autoreconf
  - Runs autoconf, autoheader, automake (and other tools not covered here)
  - Can do this recursively (i.e. automake generating makefiles “Makefile.in" in all subdirectories)

Links

- Make
  - http://www.gnu.org/software/make/manual/make.pdf
- Autoconfig, automake
  - http://www.gnu.org/software/autoconf/manual/autoconf.pdf
  - http://www.gnu.org/software/automake/manual/automake.pdf
  - http://www-src.lip6.fr/homepages/Alexandre.Duret-Lutz/dl/autotools.pdf

Apache Maven

- „Forces“ the „best practices“
- Complex, whole application deployment lifecycle
  - Compile, test, packaging, integration with other tools, resolving external dependencies ...
- Based on *POM* (stored in pom.xml file)
  - *Project Object Model*
  - Contains all necessary information about the project
  - If you follow best practices POM is short and can be generated (i.e. in IDE)
  - Super POM – POMs can have hierarchical structure with inheritance

Apache Maven

- Uses so called Archetypes
  - Template for projects
  - Java
  - Maven plugin
  - ...
  - Implicitly expects following „best practices“
- Lots of plugins
  - Can be integrated easily with e.g. Eclipse
- Explicit support for external dependencies
  - Public repositories
  - You can create your own {local, company} repository

Apache Maven - Example

- Creating a simple skeleton for maven controlled project

```
mvn archetype:create \
  -DarchetypeGroupId=org.apache.maven.archetypes \
  -DgroupId=com.mycompany.app \
  -DartifactId=my-app
```

- Right now you can run
  - mvn compile
  - mvn clean
  - mvn test, test-compile *#(j unit test)*
  - mvn package
  - ....

Apache Maven - Example

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>Maven Quick Start Archetype</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Will be packged as jar

External dependency

pom.xml

```
my-app
|-- pom.xml
`-- src
    |-- main
    |   `-- java
    |       `-- com
    |           `-- mycompany
    |               `-- app
    |                   `-- App.java
    `-- test
        `-- java
            `-- com
                `-- mycompany
                    `-- app
                        `--AppTest.java
```

Apache Ivy (deps for Apache Ant)

- Designed to resolve external dependencies
  - Using Maven repositories

Ivy.xml

```
<ivy-module version="2.0">
    <info organisation="org.apache" module="hello-ivy"/>
    <dependencies>
        <dependency org="commons-lang" name="commons-lang" rev="2.0"/>
        <dependency org="commons-cli" name="commons-cli" rev="1.0"/>
    </dependencies>
</ivy-module>
```

build.xml

```
<project xmlns:ivy="antlib:org.apache.ivy.ant" name="hello-ivy"
default="run">
   ....
   <target name="resolve" description="--> retrieve dependencies with ivy">
        <ivy:retrieve />
    </target>
</project>
```

SCons – „better (newer) make“

- *SConstruct* file in python
- File modification detection via MD5/timestamp/...
- Detect implicit dependencies automatically
  - SCons scanner
  - headers, source files...
- Explicit dependencies could be added

```
Program('hello.c')
Object('hello.c')
Java('classes', 'src')
Program('hello.c', CPPPATH = '.')

goodbye = Program('goodbye.c')
Depends(hello, goodbye)
```

Sconstruct examples (fragments)

Where to look for header dependencies

Links

- Maven
  - http://maven.apache.org/guides/getting-started/index.html
  - Contains links to guide to POM, Archetypes, Maven configuration…
- Ivy
  - http://ant.apache.org/ivy/history/latest-milestone/tutorial/start.html
- Scons
  - http://www.scons.org/doc/production/HTML/scons-user/index.html