Make

- Dependency tracking
- Based on file time-stamps
- Different (also non-standard) implementations (g)make (GNU Make), nmake (Microsoft), pmake (BSD), …
- Reads file Makefile by default
- `$ make target`

Make language (excerpt)

- Macros

  ```
  CFLAGS = -Wall
  ```
- Suffix rules

  ```
  %.o: %.d
      $(DC) $(DFLAGS) -c $< -o $@
  ```
- Rules

  ```
  targets: prerequisites
      commands
  ```

At least one target (comma separated), zero or more prerequisites (comma separated) and zero or more commands (newline separated).

Example Makefile

```
DC = dmd
DFLAGS = -property -w -wi -gc -Isrc
LD = dmd

SOURCES = $(wildcard src/*.d)
OBJECTS = $(patsubst %.d, %.o, $(SOURCES))
BINARY = wordcloud

%.o: %.d
	$(DC) $(DFLAGS) -c $< -of$@
```

Example Makefile (cont.)

```
.PHONY: all
all: build
.PHONY: build
build: $(BINARY)
$(BINARY): $(OBJECTS)
	$(LD) $(LDFLAGS) $^ -of$@


.PHONY: clean
clean:
	rm -f $(OBJECTS) $(BINARY)
```

AutoTools

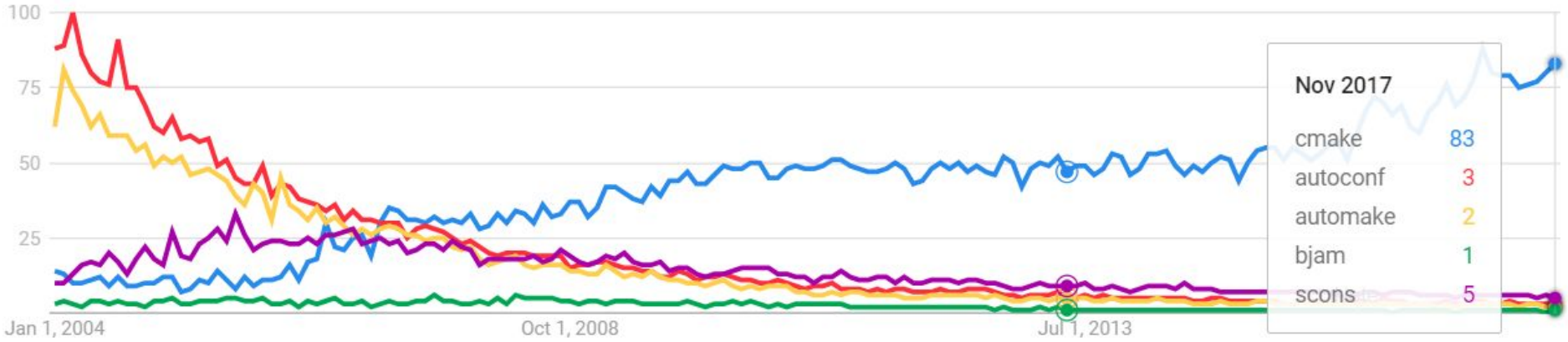
Using Autotools

```
./configure <options>

make

make install
```

- Unix only
- Old-school :P

CMake

Interest over time



CMake

- cross-platform, open-source build system
- used to build, test and package software
- uses CMakeLists.txt config files
- platform, and compiler independent
- generates Makefiles/Workspaces
  - standalone: Make, NMake, Bjam, …
  - integrated: Visual Studio, Eclipse, XCode
  - generators: Autotools, **CMake**, GYP

Using CMake

- user writes single CMakeLists.txt that CMake understands
- CMake generates Makefiles/project-solutions
- CMake takes care of working with compiler/platform/architecture
- just don’t make any assumptions during configuration time!
- CMake is fast: generates a CMakeCache.txt
  - modify a little bit of state without rerunning whole configuration pipeline
- there is also a GUI tool: cmake-gui
- today: many IDEs provide native CMake support

Example

```
$> cd src
$> mkdir build
$> cd build
$> cmake ..
$> cmake --build .
```

Note: `cd` and `mkdir` are platform specific, thus it is better to use the commands provided by `cmake -E help`. In this example:

`cmake -E chdir` and `cmake -E make_directory`

cmake --build

```
cmake  --build <dir> \
       --target <target_name> \
       --config <cfg_name> \
       --clean-first
```

cmake_minimum_required

cmake_minimum_required(VERSION 3.1.3 FATAL_ERROR)

- find the version that supports everything you need as the minimum, in order to increase portability
- don’t set 2.8 as the minimum
- if you use 2.6 or 2.4, God kills a kitten

project()

project(<name> VERSION <version> LANGUAGES CXX)

- CMake will place a call to project if you don’t do it yourself
- languages restricts the compilers that are looked for on your system

add_subdirectory

add_subdirectory(<src_dir> [<bin_dir>])

- creates a Makefile/solution for every so created subdirectory
- modular design!
- good design:
  - try to assume that your project could also be a sub-project
  - don’t modify global variables/flags
  - don’t assume your project is the root of the build

find_package()

find_package(Boost 1.56.0 REQUIRED COMPONENTS program_options)

- defines variables/Targets to use in your project