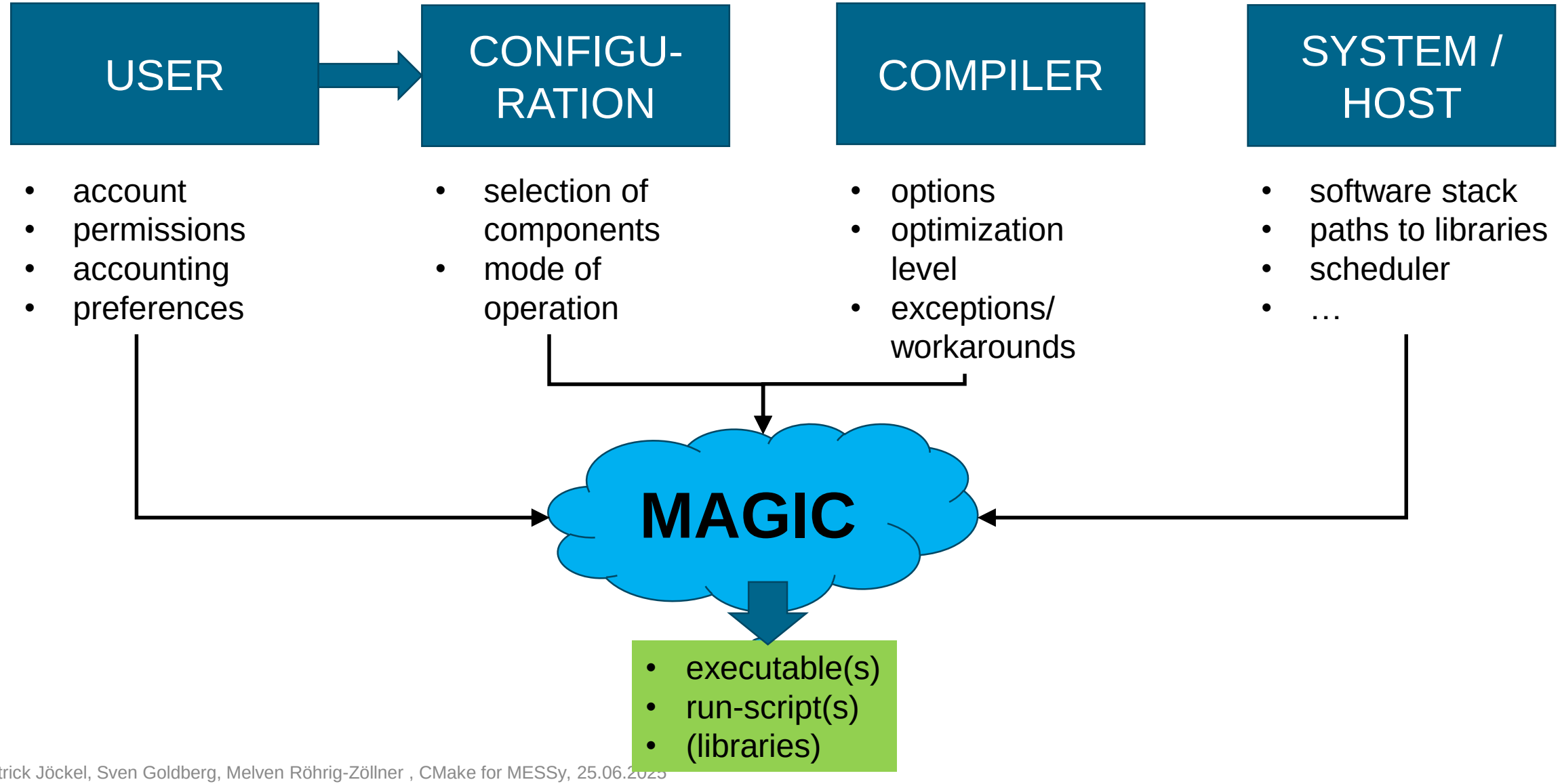
CMake:
The new build-system for MESSy

A tutorial during the 13th MESSy symposium, 25.06.2025

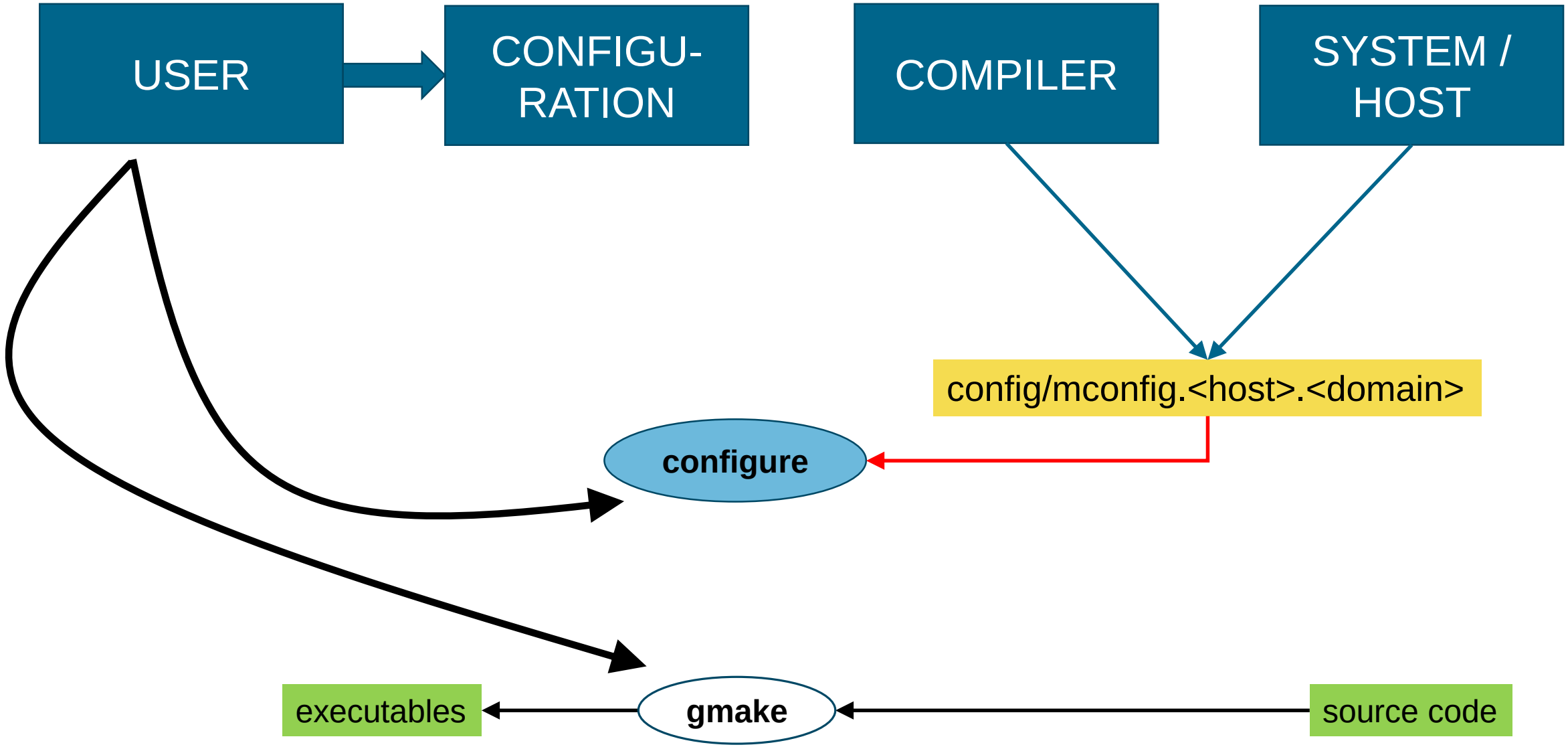
Patrick Jöckel (DLR-PA), Sven Goldberg, Melven Röhrig-Zöllner (DLR-SC)

The build process

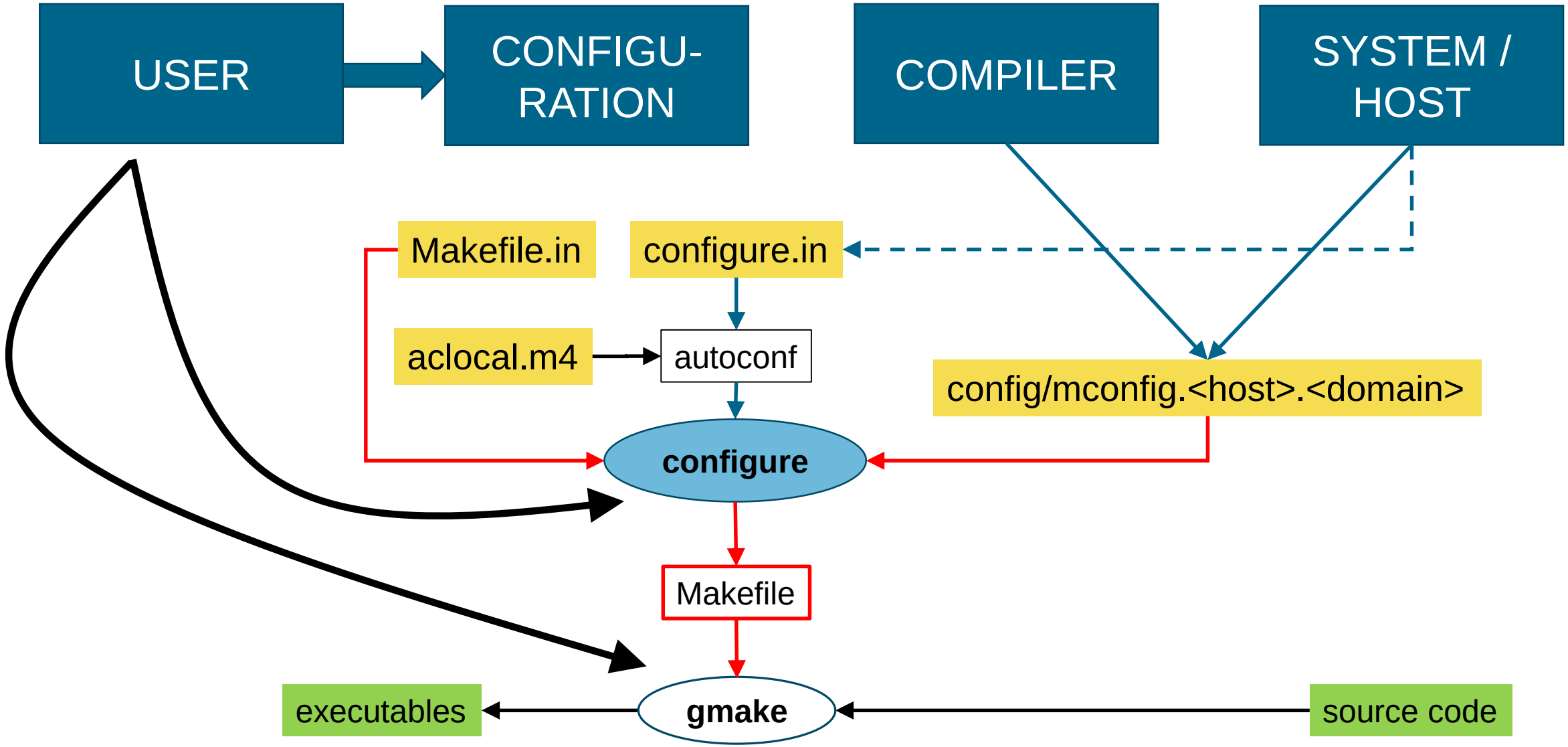
Required information to build our software



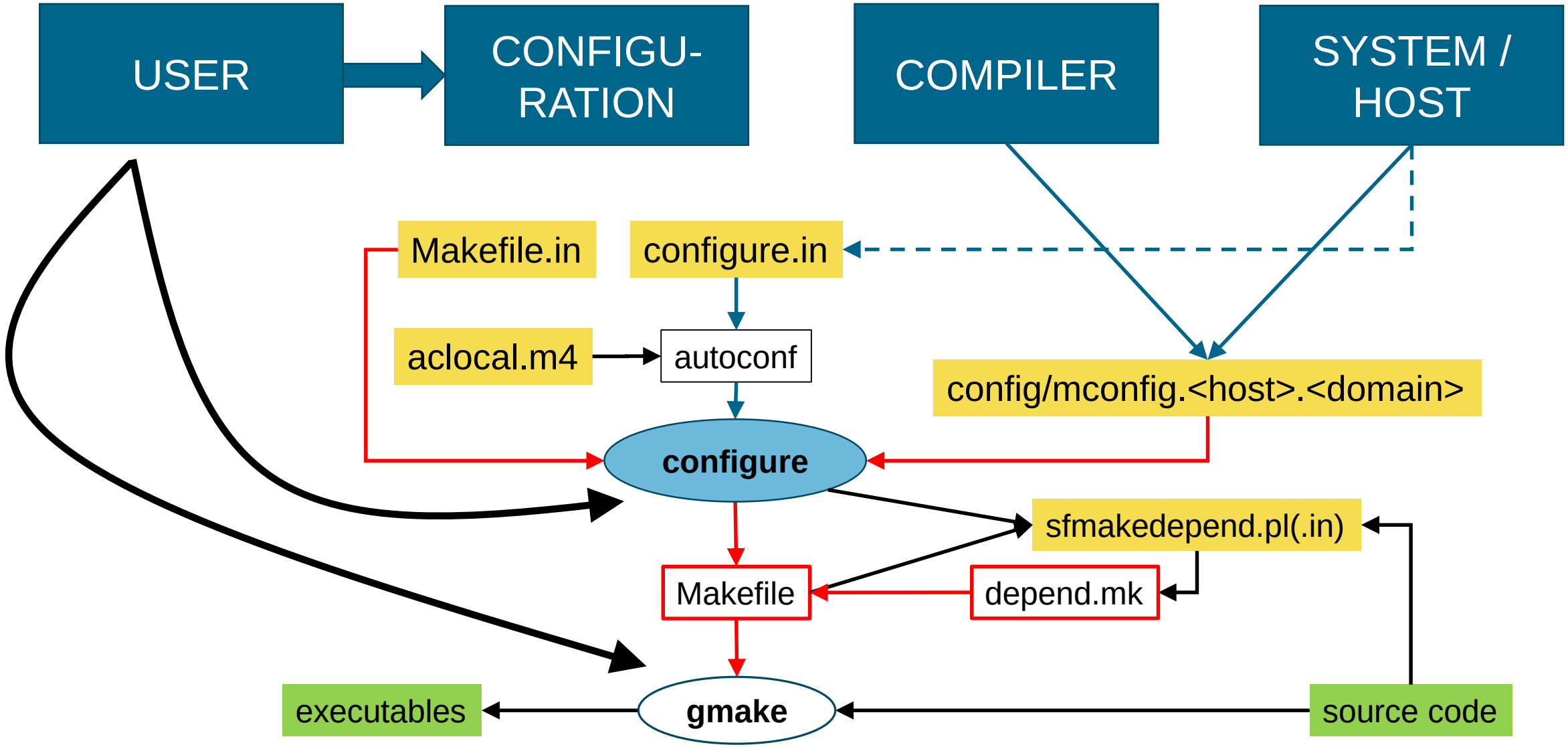
Building with “autotools” / “autoconf” / “configure”



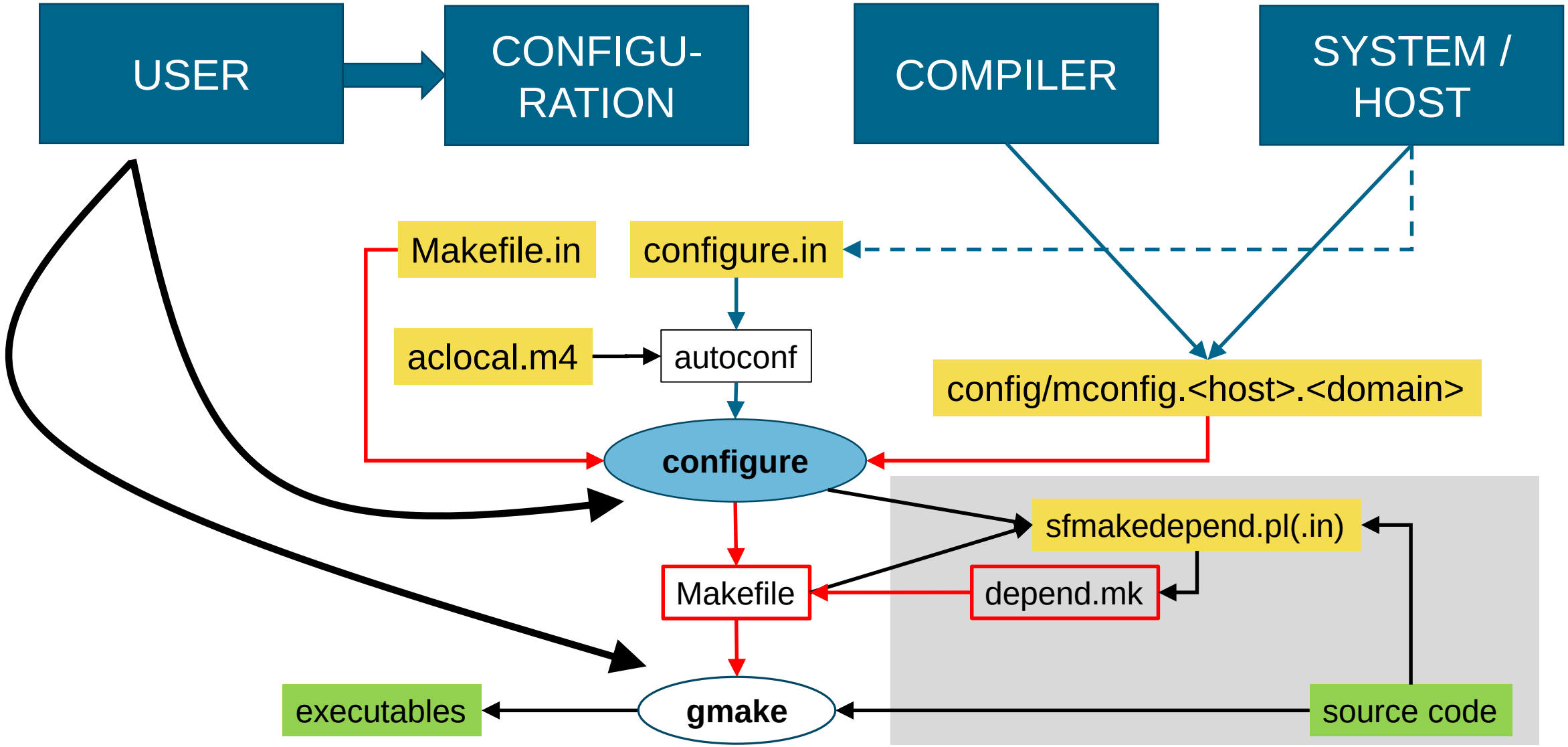
Building with “autotools” / “autoconf” / “configure”



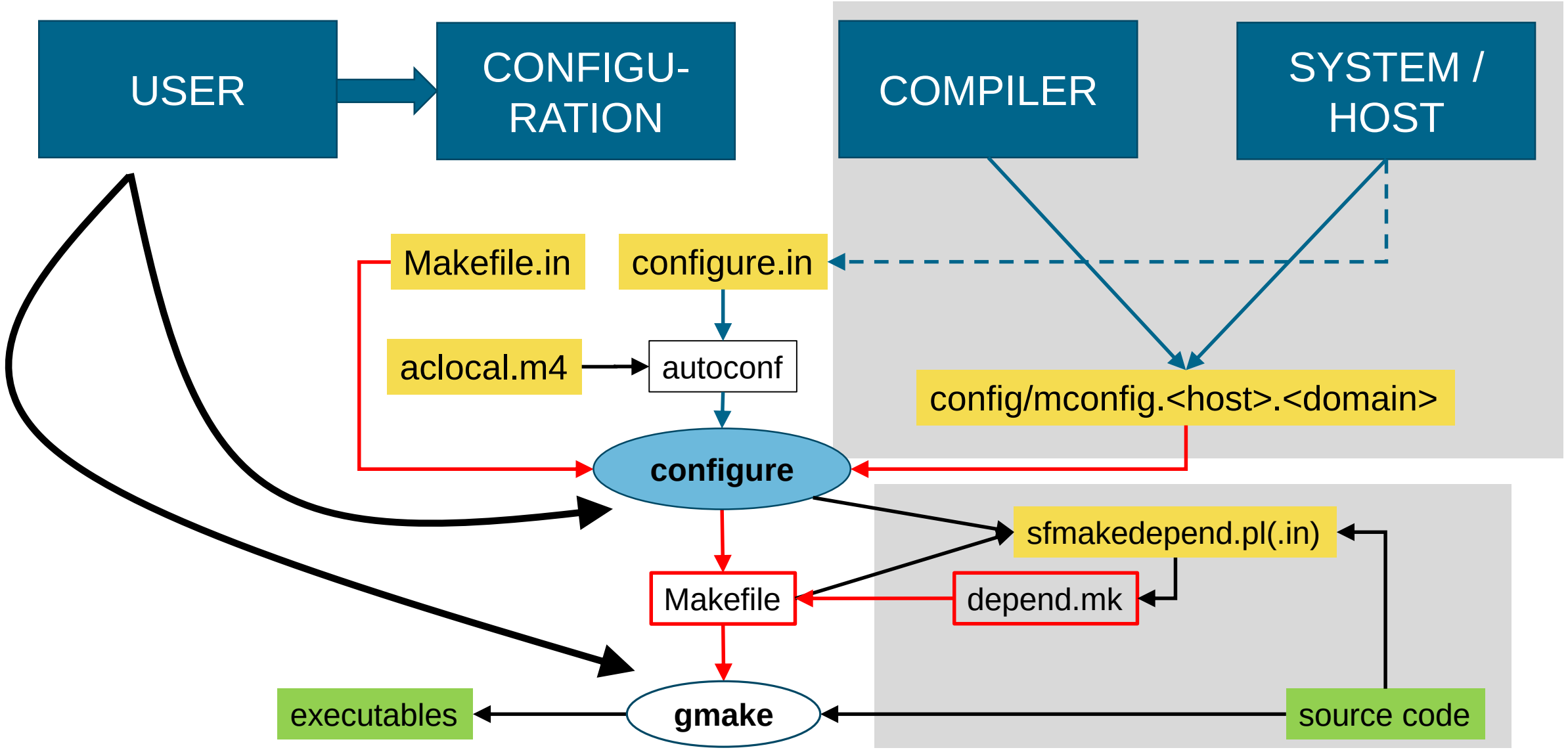
Building with “autotools” / “autoconf” / “configure”



Building with “autotools” / “autoconf” / “configure”



Building with “autotools” / “autoconf” / “configure”



Constant sources of pain

- redundant compiler-related information (options) for several hosts/domains
- source code dependencies based on “hacked” perl-script
- source code “internal“ configuration dependencies need to be manually implemented in **`configure.in`**, e.g., inter-library-dependencies such as
  - pnetcdf requires MPI
  - option X works for base-model Y, but not for Z
  - …
- several times reworked, still imperfect!

CMAKE

https://cmake.org

Advantages of CMake for developers

- no more redundant compiler information, e.g. INTEL is treated the same way on all HOSTs
- cmake intrinsically “knows” about a wide variety of compilers and setting
- cmake has very powerful “package” finders

✓ porting to new HOSTs is very much simplified

✓ (cross platform building)

- auto-generated source code dependencies (USE, include, #include) work like a charm
- very powerful, *target*-based inter/intra-dependencies: you simply specify what needs to be “known” up-stream!, no more unnecessary include- and/or library paths: simplifies encapsulation

Advantages of CMake for users

- out-of-source builds
  - you can have several builds (with different configurations) in parallel, e.g.
    - one “optimized” and one with compiler-run-time checks for debugging
    - build with different compiler suites
    - …
- faster builds (in particular with ninja instead of gmake)
- user friendly ASCII-menu user interface (even a GUI is available!)

Building with CMake

- scripts to setup the environment for a specific HOST/DOMAIN
  (only these scripts are rquired for “porting”!)
  - **config/setup_env_<host>.<domain>_<COMPILER-suite>_<VERSION>.[bash,tcsh]**
    Examples:
    config/setup_env_levante.dkrz.de_GNU_11.2.0.bash
    config/setup_env_levante.dkrz.de_GNU_11.2.0.tcsh

    - resets “module environment”
    - loads required modules from software stack
    - sets required environment variables (if not done by modules)

- all the rest is done by cmake …

CMake: recommended workflow – part I: configuration & compilation

- read **readme/instructions/how-to-use-cmake.md**
- **source config/setup_env_...[.tcsh,.bash]**
  (depending on the shell you use)
- **cd build**
- **cmake .. [-G Ninja] [configuration options]**
  OR
- **cmake .. [-G Ninja]**
  ccmake .. (select your configuration interactively)
- **gmake [options] [target]**
  OR
- **ninja [target]**

CMake: recommended workflow – part II: run-script

- Create all run-scripts from their **`messy/nml/*/xmessy_mmd.header`**:
  `gmake runscripts`
  `ninja runsripts`
- Create one specific run-script:
  `cd ..; ./messy/util/xmkr EMAC/AGCM; cd build`

→ **`./messy/util/xmessy_mmd.EMAC-AGCM`** etc.

Search for executables:

1. **`$MESSY_BASEDIR/bin`** **[old “configure”-based]**
2. **`$MESSY_BASEDIR/build/bin`** **[cmake default]**
3. **`BUILDDIR/bin`** **[cmake customized build]**
   `BUILDDIR=…` **to be set in run-script**

Workshop tasks

1. [levante]:
   /home/b/b302019/MESSY_SYMPOSIUM/messy_d2.55.2_devel_66b9a2c3_src.zip
2. Build EMAC
   a) with “old” configure and “gmake” [How long does it take?]
   b) `gmake distclean`
   c) with cmake (same configuration as in a)!) and gmake [How long does it take?]
3. Create a parallel build (same configuration) with ninja [How long does it take?]
4. Create a parallel build with the same configuration, but different **`BUILD_TYPE`**
5. Run **`EMAC/AGCM`** (1 day) with exe of 2.c) and 4) [Which one is how much faster?]
6. Build the other targets as well (tools, mbm, docu).

CMake: Creation of chemical mechanism(s)

- `cd build`
- `cmake .. [options]`
- `[ccmake ..]`
- `gmake / ninja tools`
- `../messy/util/xconfig`
- `gmake / ninja`

Impressum

Thema:	**CMAKE: The new build-system MESSy**
Datum:	25.06.2025
Autor:	Patrick Jöckel, Sven Goldberg, Melven Röhrig-Zöllner
Institut:	MESSy Consortium
Bildcredits:	Alle Bilder „DLR (CC BY-NC-ND 3.0)“, if not labelled otherwise