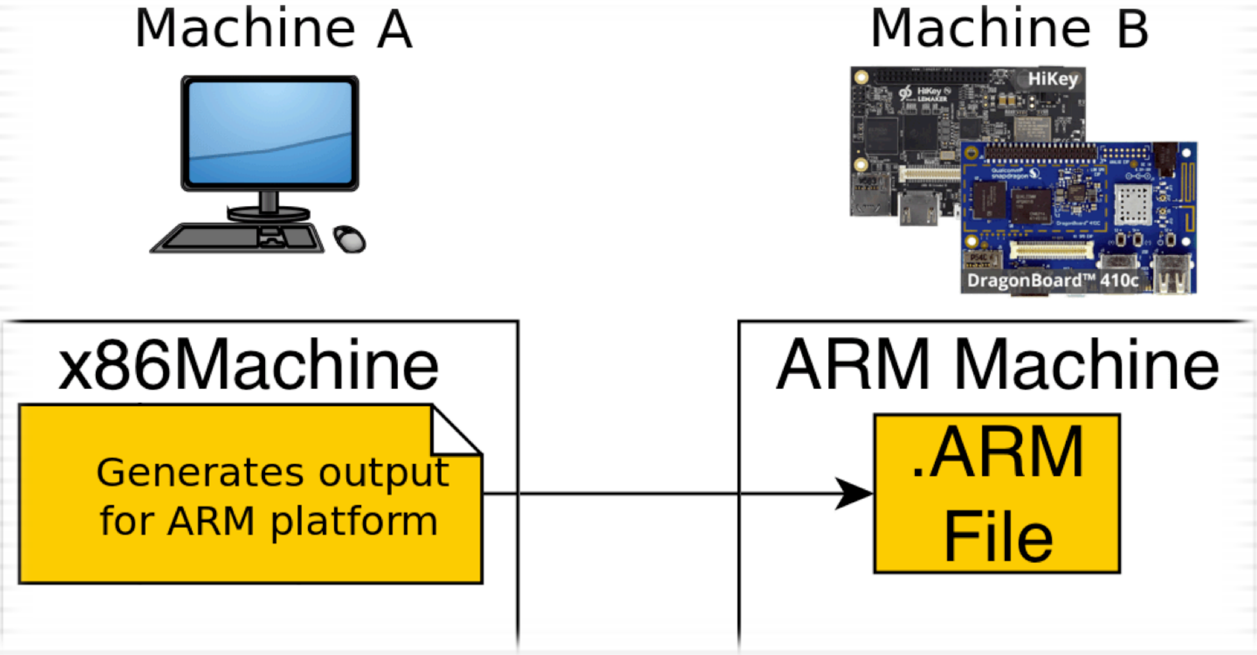
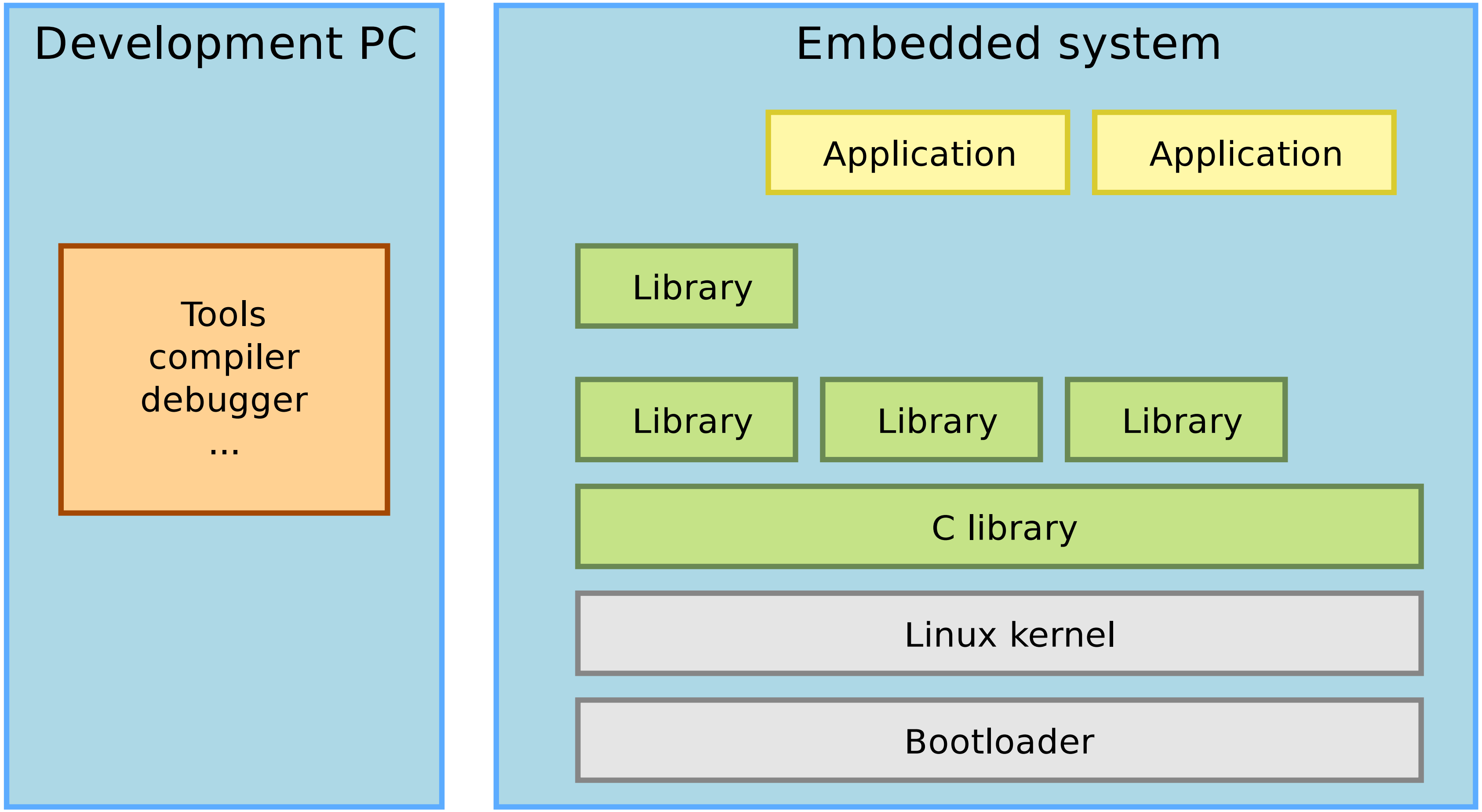
11. Cross-compiling toolchains



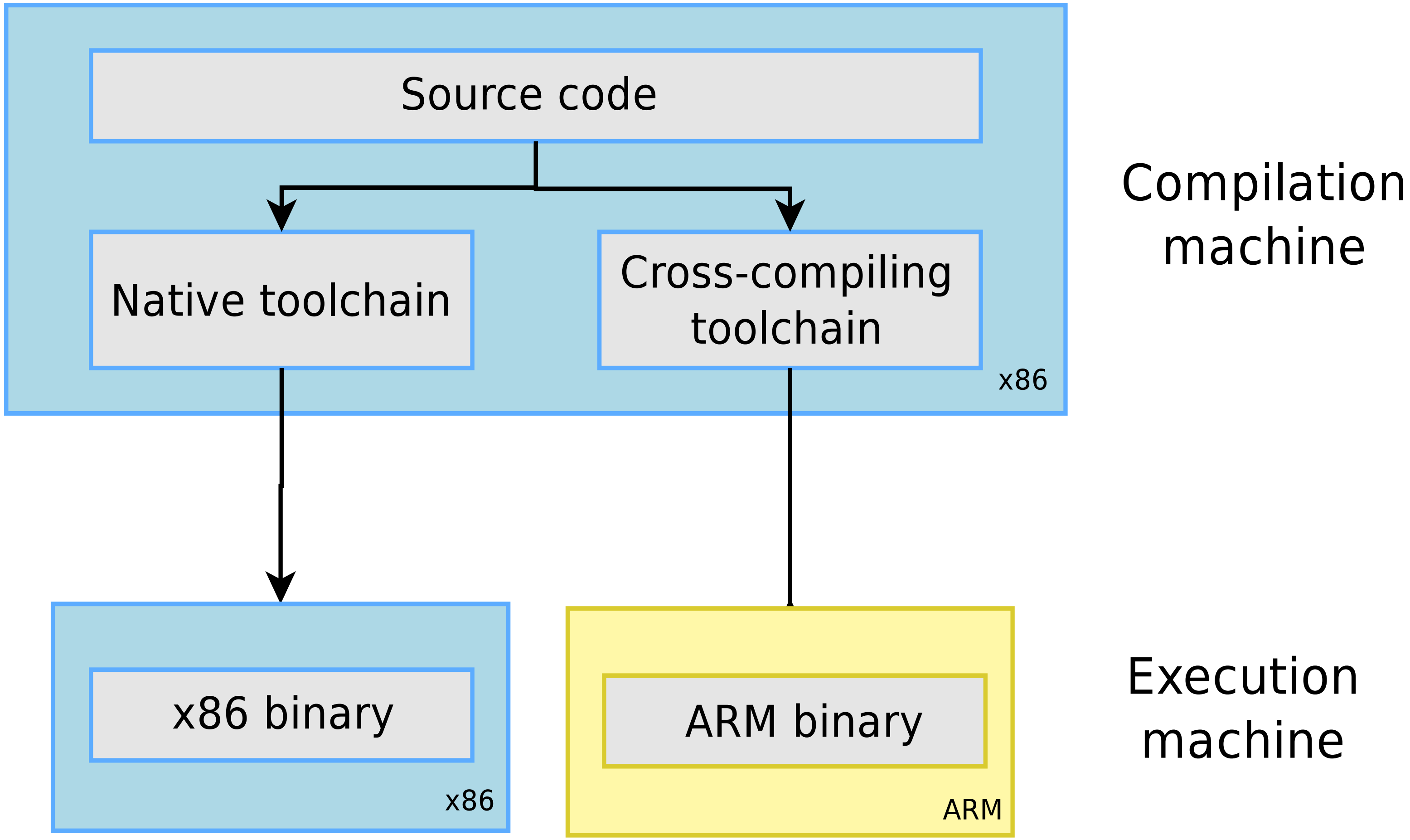
Global architecture



Definition and Components

- The usual development tools available on a GNU/Linux workstation is a native toolchain
- This toolchain runs on your workstation and generates code for your workstation, usually x86
- For embedded system development, it is usually impossible or not interesting to use a native toolchain
- The target is too restricted in terms of storage and/or memory I The target is very slow compared to your workstation.
- You may not want to install all development tools on your target.
- Therefore, cross-compiling toolchains are generally used. They run on your workstation but generate code for your target.

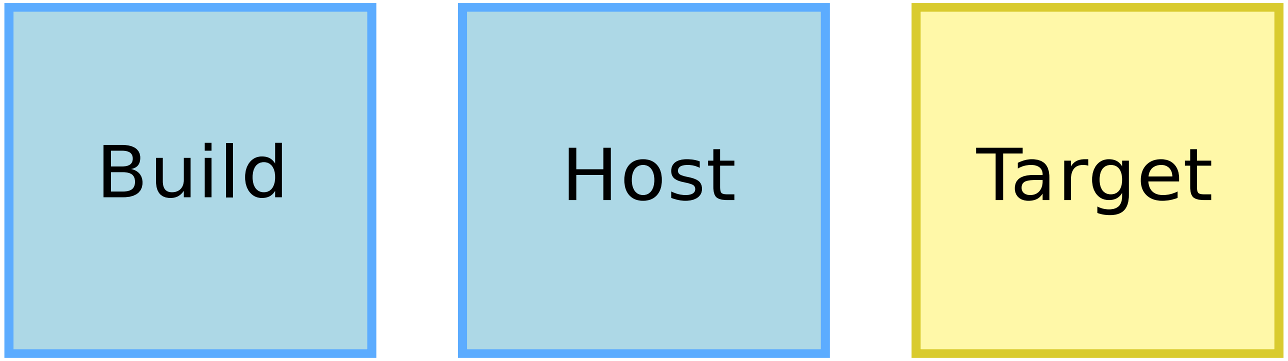
Definition (2)



Machines in build procedures

- Three machines must be distinguished when discussing toolchain creation
  - The build machine, where the toolchain is built.
  - The host machine, where the toolchain will be executed.
  - The target machine, where the binaries created by the toolchain are executed.

The common case



Build

Host

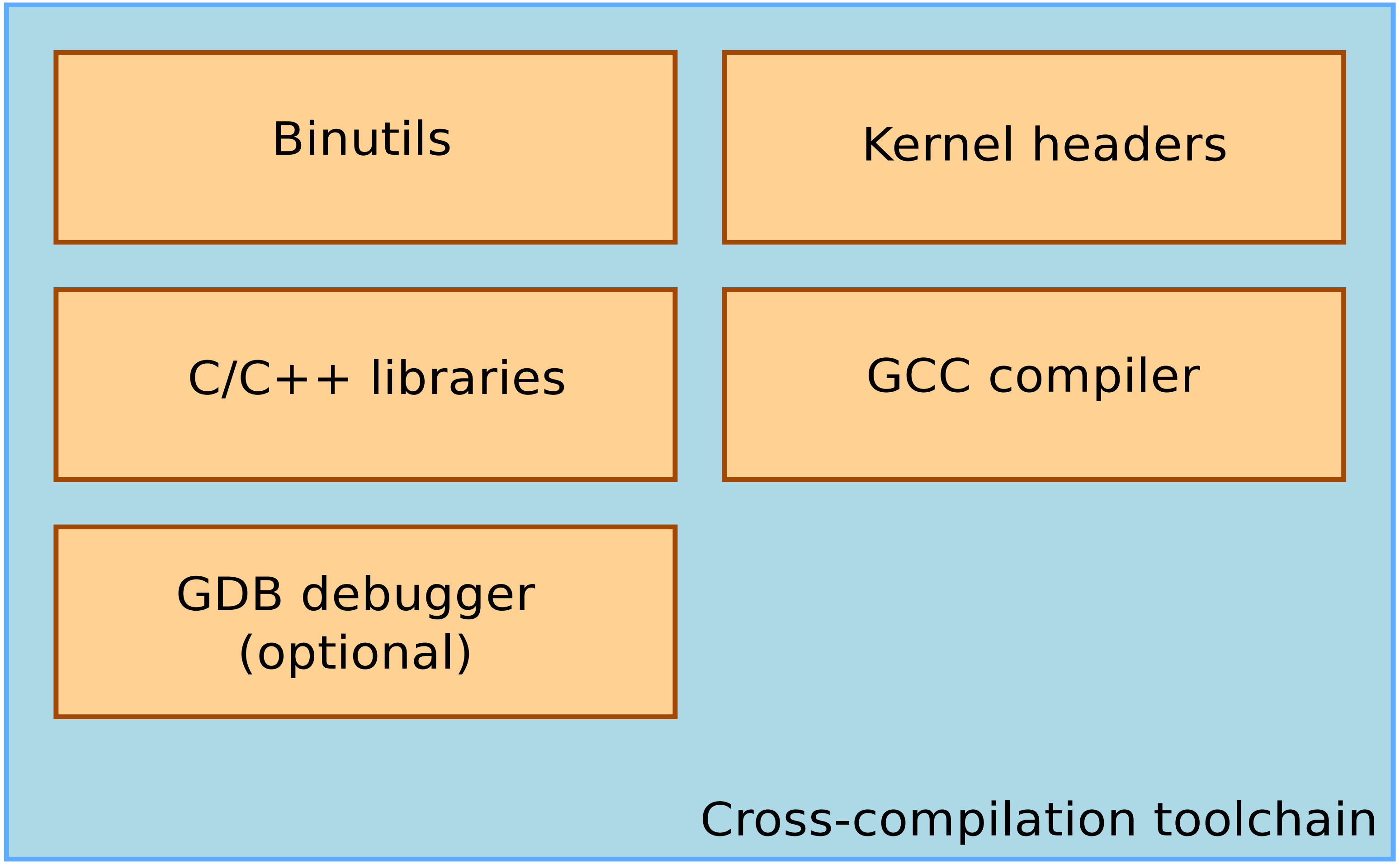
Target

Cross build

used to build a toolchain that runs
on your workstation but generates
binaries for the target

The most common case in embedded development

Components

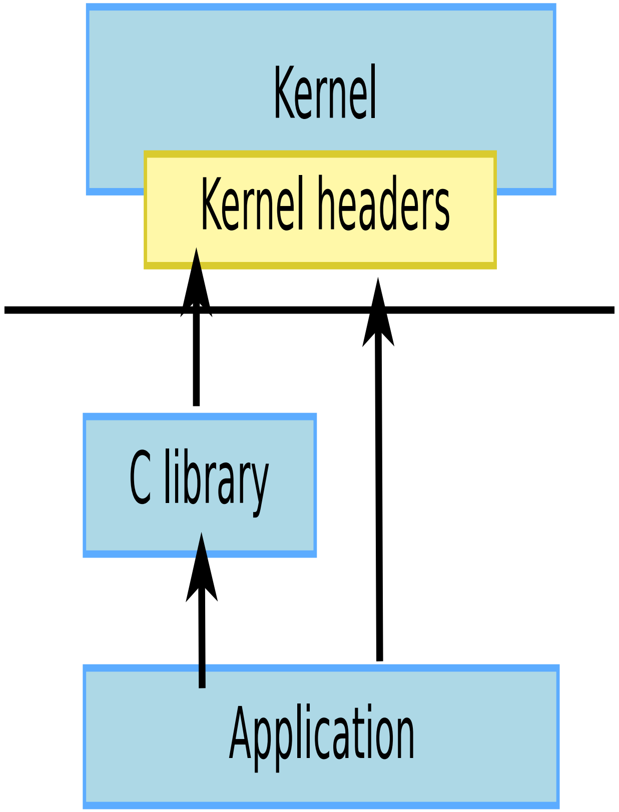

Binutils
Kernel headers
C/C++ libraries
GCC compiler
GDB debugger (optional)
Cross-compilation toolchain

Binutils

- Binutils is a set of tools to generate and manipulate binaries for a given CPU architecture
  - as, the assembler, that generates binary code from assembler source code
  - ld, the linker
  - ar, ranlib, to generate .a archives, used for libraries
  - objdump, readelf, size, nm, strings, to inspect binaries.
    - Very useful analysis tools !
  - strip, to strip useless parts of binaries in order to reduce their size.
- http://www.gnu.org/software/binutils/
- GPL license

Kernel headers

- The C library and compiled programs needs to interact with the kernel
  - Available system calls and their numbers
  - Constant definitions
  - Data structures, etc.
- Therefore, compiling the C library requires kernel headers, and many applications also require them.
- Available in<linux/...> and <asm/...> and a few other directories corresponding to the ones visible in include/ in the kernel sources



Kernel headers

- The kernel-to-userspace ABI is backward compatible :
  - Binaries generated with a toolchain using kernel headers older than the running kernel will work without problem, but won't be able to use the new system calls, data structures, etc.
  - Binaries generated with a toolchain using kernel headers newer than the running kernel might work on if they don't use the recent features, otherwise they will break

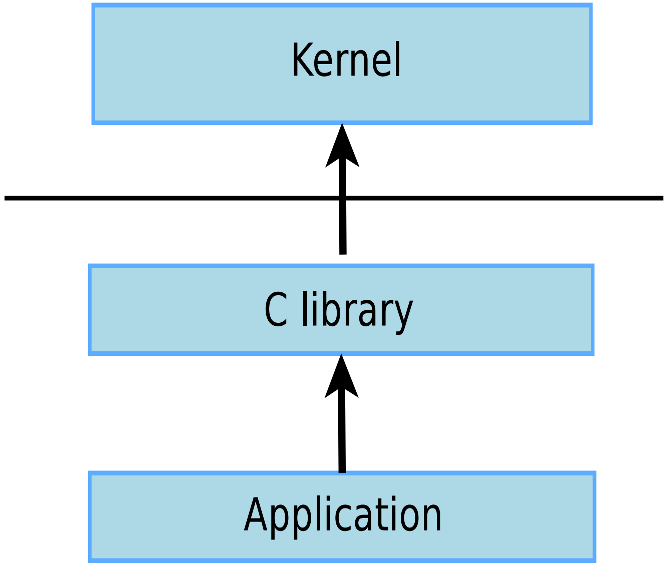
- Using the latest kernel headers is not necessary, unless access to the new kernel features is needed

GCC

- GNU Compiler Collection, the famous free software compiler
- Can compile C, C++, Ada, Fortran, Java, Objective-C, Objective-C++, and generate code for a large number of CPU architectures, including ARM, AVR, Blackfin, CRIS, FRV, M32, MIPS, MN10300, PowerPC, SH, v850, i386, x86 64, IA64, Xtensa, etc.
- http://gcc.gnu.org/
  - Available under the GPL license, libraries under the LGPL.

C library

- The C library is an essential component of a Linux system
  - Interface between the applications and the kernel
  - Provides the well-known standard C API to ease application development
- Several C libraries are available :
  - glibc, uClibc, eglibc, dietlibc, newlib, etc.



Obtaining a Toolchain

Building a toolchain manually

- Building a cross-compiling toolchain by yourself is a difficult and painful task ! Can take days or weeks !
  - Lots of details to learn: many components to build, complicated configuration
  - Lots of decisions to make (such as C library version, ABI, floating point mechanisms, component versions)

Get a pre-compiled toolchain

- Solution that many people choose
  - Advantage: it is the simplest and most convenient solution I Drawback: you can’t fine tune the toolchain to your needs
  - Determine what toolchain you need: CPU, endianism, C library, component versions, ABI, soft float or hard float, etc.
- Check whether the available toolchains match your requirements.
- Possible choices
  - Sourcery CodeBench toolchains
  - Linaro toolchains
- More references at http://elinux.org/Toolchains.

Installing and using a pre-compiled toolchain

- Follow the installation procedure proposed by the vendor

- Usually, it is simply a matter of extracting a tarball wherever you want.

- Then, add the path to toolchain binaries in your PATH:
  - export PATH=/path/to/toolchain/bin/:$PATH

- Finally, compile your applications PREFIX-gcc -o foobar foobar.c

- PREFIX depends on the toolchain configuration, and allows to distinguish cross-compilation tools from native compilation utilities

Lab 11

□ Installer et configurer une chaine de cross compilation ARM.

□ Cross compiler votre premier programme et vérifier la bonne génération de votre binaire