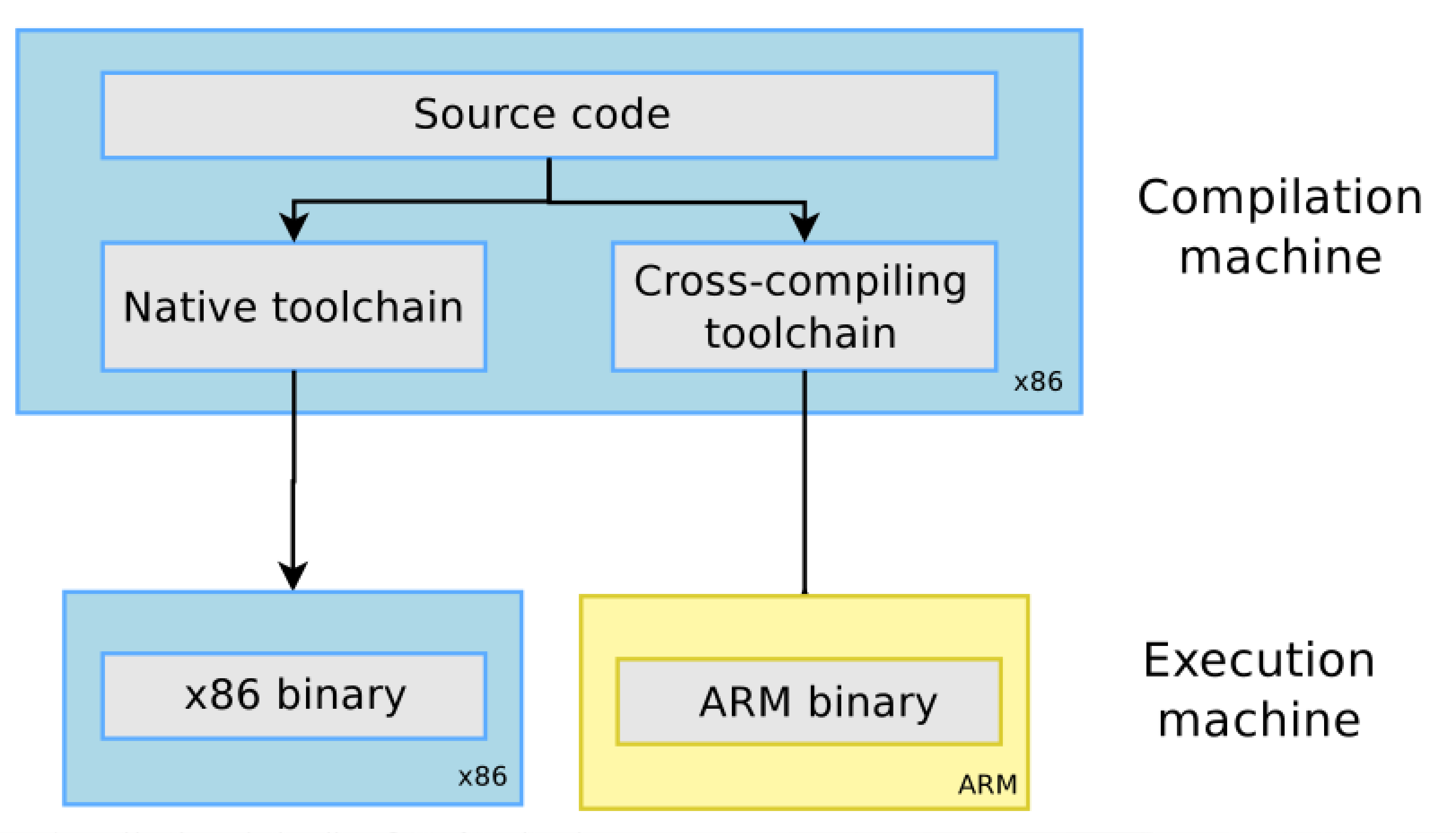
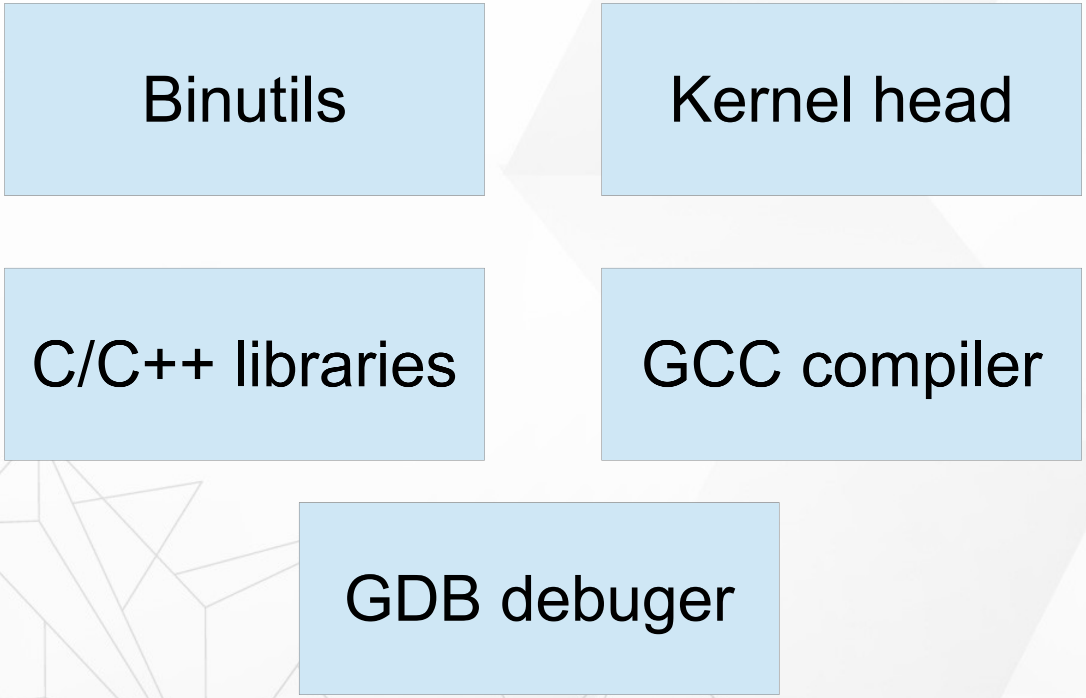
CH4 Cross Compilation Toolchain

Cross Compilation toolchain



GCC Components

- The GNU C Compiler
- The GNU Compiler Collection



Binutils

Kernel head

C/C++ libraries

GCC compiler

GDB debuger

Binutils

- Binutils
  - **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
  - **strip** : to strip useless parts of binaries in order to reduce their size

Kernel head

- The C library and compiled programs needs to interact with the kernel
- Compiling the C library requires kernel headers, and many applications also require them
- The kernel to user space ABI is backward compatible

GCC

- GCC originally stood for the "GNU C Compiler."
- GNU Compiler Collection
  - C, C++, Ada, Objective-C, Fortran, JAVA ...
- http://gcc.gnu.org/

GCC flag

- arm-linux-gnueabihf-gcc –help
- -c : Compile and assemble, but do not link
- -o <file> : Place the output into <file>
- -shared : Create a shared library
- -g : add debug information
- -O : sets the compiler's optimization level
- -Wall : enables all compiler's warning messages
- -D : defines a macro to be used by the preprocessor
- -I : adds include directory of header files
- -L,-l :
  - -L looks in directory for library files
  - -l links with a library file

C library

- The C library is an essential component of a Linux system
- Several C libraries are available:
  - **glibc, uClibc, eglibc, dietlibc, newlib**
- The choice of the C library must be made at the time of the cross-compiling toolchain generation, as the GCC compiler is compiled against a specific C library.

Floating point support

- For processors having a **floating point unit**, the toolchain should generate hard float code, in order to use the floating point instructions directly
- For processors without a floating point unit
  - Generate hard float code and rely on the kernel to emulate the floating point instructions
  - Generate soft float code, so that instead of generating floating point instructions, calls to a user space library are generated

Obtain a Toolchain

- Building a cross-compiling toolchain by ourself
  - Crosstool-NG
  - http://crosstool-ng.org/#introduction

- Pre-build toolchain
  - Linaro - https://wiki.linaro.org/WorkingGroups/ToolChain
  - By Linux distribution -
    - sudo apt-get install gcc-arm-linux-gnueabi
  - CodeSourcery
  - BSP

Installing and using a pre-compiled toolchain

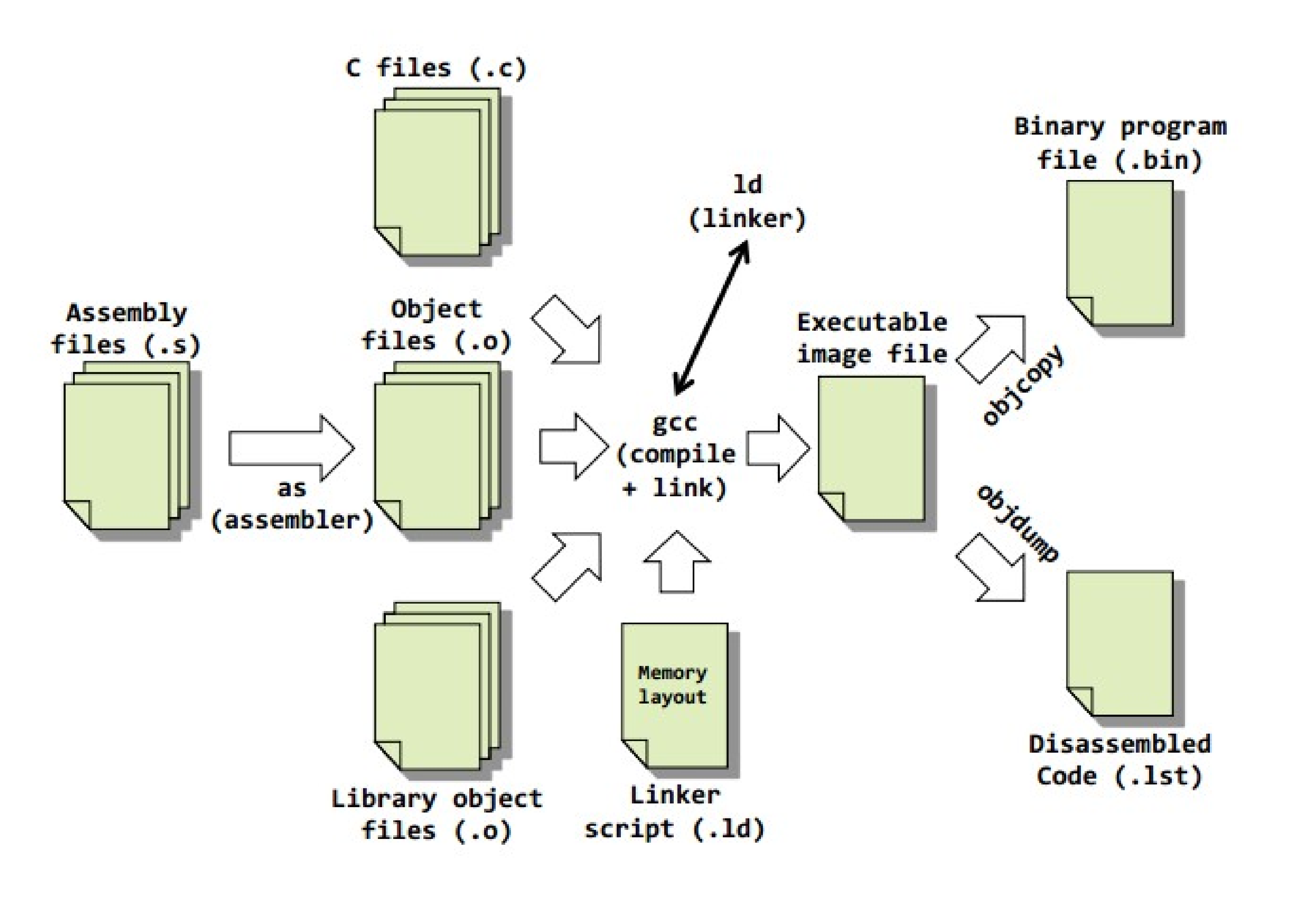
- Add the path to toolchain binaries in your PATH: export
  - PATH=/path/to/toolchain/bin/:$PATH
- Compile your applications
  - PREFIX-gcc -o testme testme.c
- PREFIX
  - depends on the toolchain configuration

Toolchain building utilities

- **Buildroot**
  - Makefile-based
  - http://www.buildroot.net
- PTXdist
  - Makefile-based
  - http://pengutronix.de/software/ptxdist/
- OpenEmbedded / Yocto
  - A featureful, but more complicated build system
  - http://www.openembedded.org/
  - https://www.yoctoproject.org/

Compile, Assembler, Linker

Software Development Tools Overview



Tools Descriptions

- C/C++ compiler
  - produces ARM machine code object modules
- Assembler
  - Translates Assembly Language Source Files Into Machine Language Object modules
- Linker
  - Combines object files into a single executable object module

Build Linux Library

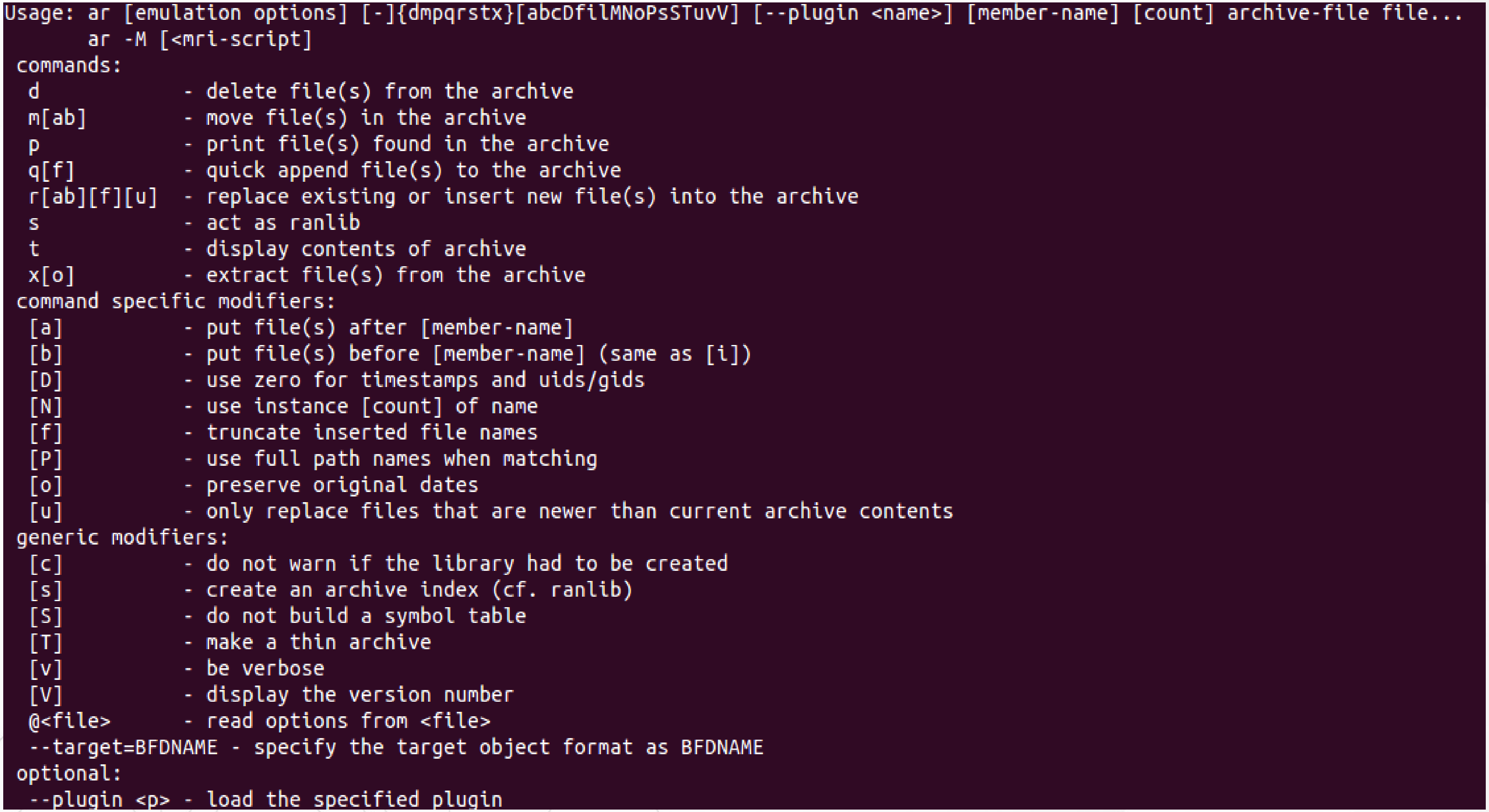
Linux Library

- Static Libraries
  - statically aware
- Dynamically Linked "Shared Object" Libraries
  - Dynamically linked at run time

Static Libraries

- static_lib_name.a
- Create static library with **ar**
  - **ar --help**
  - **ar** -cvq libctest.a test1.o test2.o
- Compile
  - gcc -o test main.c libctest.a
  - gcc -o test main.c -L/path/to/library-directory -lctest

ar



```
Usage: ar [emulation options] [-]{dmpqrstx}[abcDfilMNoPsSTuvV] [--plugin <name>] [member-name] [count] archive-file file...
       ar -M [<mri-script]
 commands:
  d            - delete file(s) from the archive
  m[ab]        - move file(s) in the archive
  p            - print file(s) found in the archive
  q[f]         - quick append file(s) to the archive
  r[ab][f][u]  - replace existing or insert new file(s) into the archive
  s            - act as ranlib
  t            - display contents of archive
  x[o]         - extract file(s) from the archive
 command specific modifiers:
  [a]          - put file(s) after [member-name]
  [b]          - put file(s) before [member-name] (same as [i])
  [D]          - use zero for timestamps and uids/gids
  [N]          - use instance [count] of name
  [f]          - truncate inserted file names
  [P]          - use full path names when matching
  [o]          - preserve original dates
  [u]          - only replace files that are newer than current archive contents
 generic modifiers:
  [c]          - do not warn if the library had to be created
  [s]          - create an archive index (cf. ranlib)
  [S]          - do not build a symbol table
  [T]          - make a thin archive
  [v]          - be verbose
  [V]          - display the version number
  @<file>      - read options from <file>
  --target=BFDNAME - specify the target object format as BFDNAME
 optional:
  --plugin <p> - load the specified plugin
```

Dynamically Linked "Shared Object" Libraries

- Dynamic_lib_name.so
- Create share library
  - gcc -**shared** -Wl,-soname,libctest.so.1 -o libctest.so.1.0 test1.o test2.o
  - ln -s libctest.so.1.0 libctest.so.1
  - ln -s libctest.so.1 libctest.so
- gcc -o test main.c -L/library_PATH/ -lctest
- export LD_LIBRARY_PATH=LIB_PATH: $LD_LIBRARY_PATH
- ./test

Dynamically Linked "Shared Object" Libraries

- ldconfig
- configure dynamic linker run-time bindings
- /etc/ld.so.conf
  - 1. #vim /etc/ld.so.conf and add LIB path
    - /usr/local
  - 2. #ldconfig /usr/local/
    - /etc/ld.so.cache