从零开始为 RISC-V 构建一个 Linux 系统

第 2 章 构建系统（Build System）

汪辰

目录

01 Build System 概述

02 GNU Make

03 GNU Autotools

04 Kconfig & Kbuild

01 Build System 概述

构建系统的定义

- “**构建系统（Build System）**”或“构建自动化工具”是一种软件工具或软件工具套件。
- Build System 用于对以下操作过程实现自动化，它们包括（不限于）编译、链接和打包源代码，生成可执行程序、库或其他可部署的组件。
- 构建系统的主要目标是管理源文件之间复杂的依赖关系，确保高效的构建（譬如通过仅重新构建必要的组件），使构建过程标准化，并保证不同环境下的一致性。

构建系统的主要功能

- **依赖管理**：跟踪源文件、库和其他资源之间的依赖关系，以确定正确的编译和链接顺序，以及避免不必要的重新编译。
- **自动化管理**：自动化构建过程中的重复性任务，例如（通过调用编译器和链接器）编译和链接并生成二进制代码、资源编译和打包。在可能的情况下利用并行处理来优化构建时间，提高效率。
- **配置管理**：提供配置选项机制，可以对目标平台和其他项目特定特性进行设置。
- **跨平台支持**：在不同平台环境下确保产生完全相同的输出。

常见的构建系统

- **GNU Make**：一款经典且广泛使用的构建自动化工具，常用于 C/C++ 项目。
- **GNU Autotools**：一套用于构建源代码和打包二进制文件的工具（Autoconf、Automake、Libtool）。核心任务正是为了自动化地生成可移植的、标准的 Makefile，从而让编译安装过程变得简单。
- **CMake**：一款跨平台构建系统生成器，专注于定义“构建什么”和“如何配置”，为其他构建工具（如 Make）生成构建文件（如 Makefile）。
- **Apache Maven**：一款基于 Java 的构建系统，强调约定优于配置，并提供依赖管理功能。
- **Gradle**：一款现代的构建自动化系统，结合了 Ant 的灵活性和 Maven 的依赖管理功能，支持多种编程语言。

02

GNU Make

Make - 简介

https://www.gnu.org/software/make/

- Make 是一种自动化工程管理工具。
- Makefile 是配合 make，用于描述构建工程过程中所管理的对象以及如何构造工程的过程。
- Make 如何找到 Makefile
  - ✓ 隐式查找：当前目录下按顺序找寻文件名为“GNUmakefile”、“makefile”、“Makefile”的文件
  - ✓ 显式查找：-f

Make - Makefile

- Makefile 由一条或者多条规则（rule）组成
- 每条规则由三要素构成
  - ✓ target：目标，实际要制作的文件
  - ✓ prerequisites: 生成 target 所需要的依赖
  - ✓ command：为了生成 target 需要执行的命令，可以有多条

缩进必须是一个 TAB

```
target...:prerequisites...
    command...
    ...
```

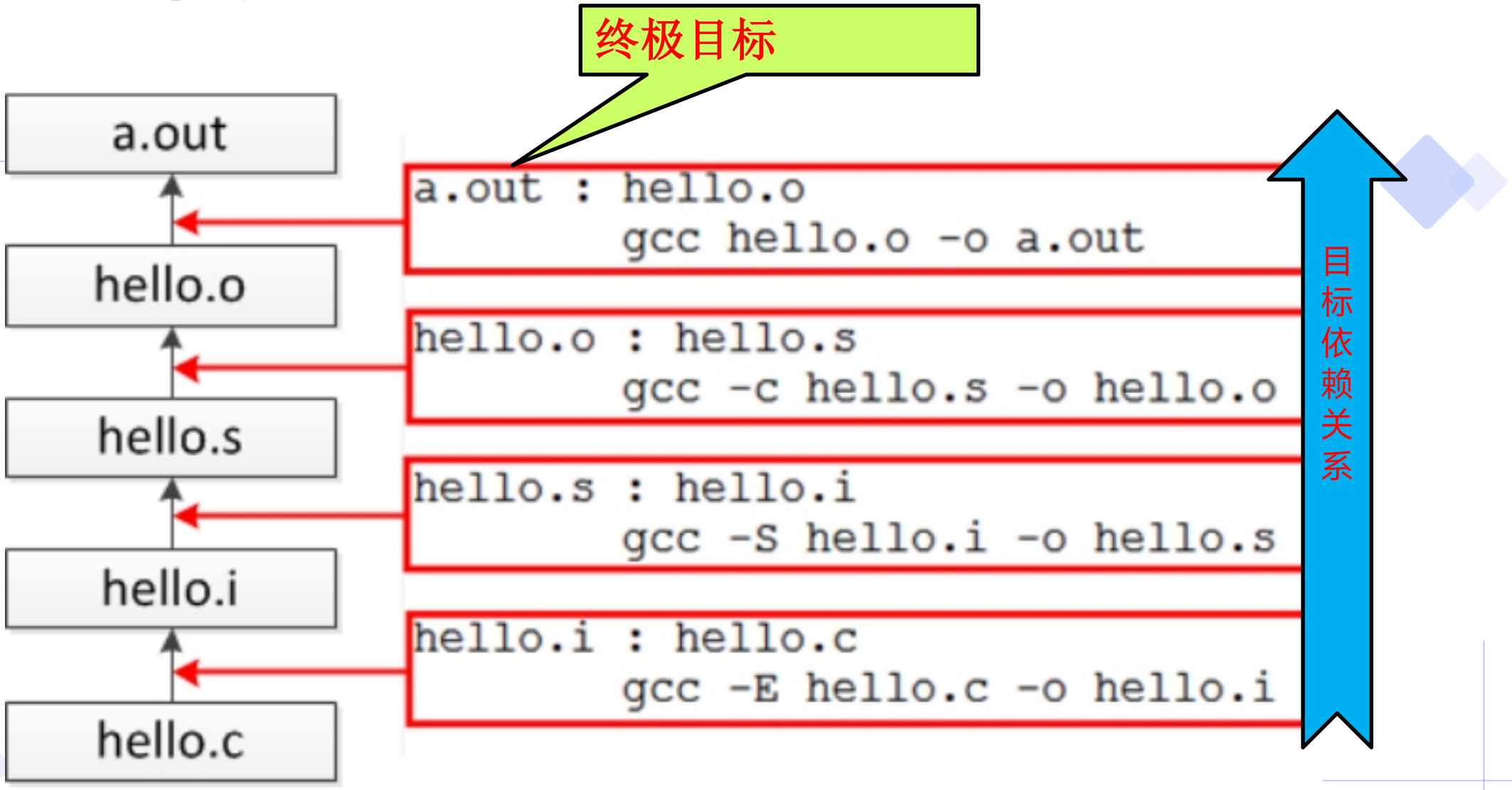
- 一个简单的 Makefile 规则例子如下:

```
hello: hello.c
    gcc hello.c -o hello
```

Make - 简介

Makefile 中规则的执行顺序

Make - 简介

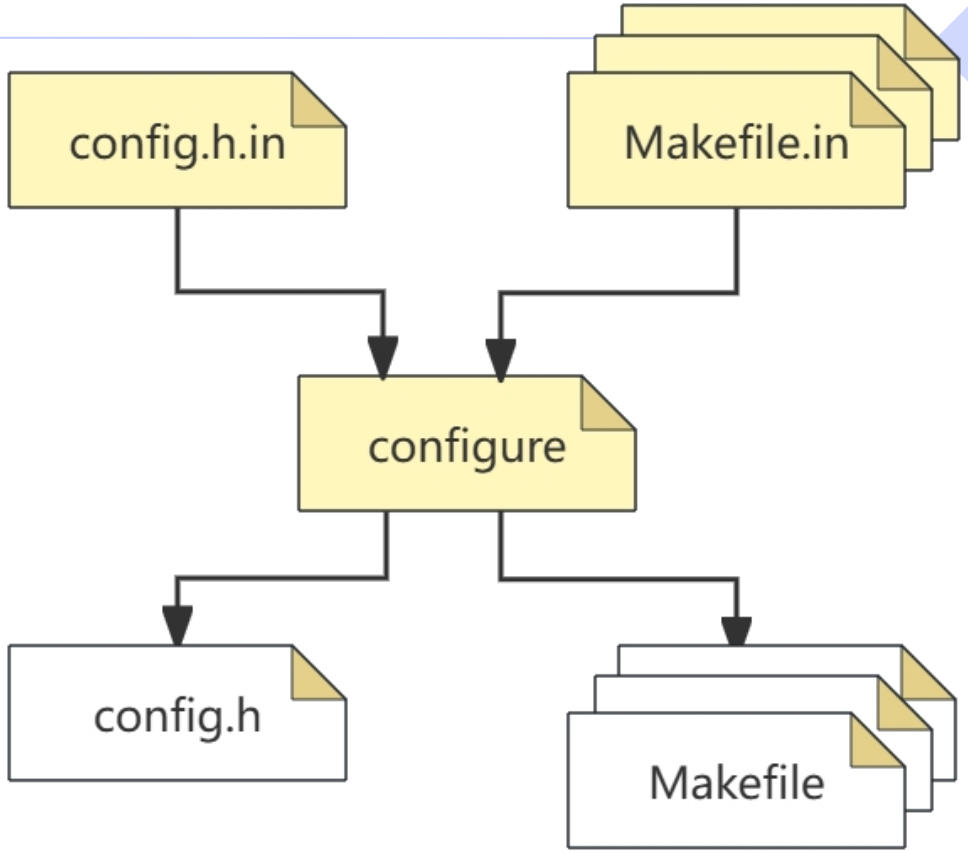

终极目标
a.out
hello.o
hello.s
hello.i
hello.c
a.out : hello.o
gcc hello.o -o a.out
hello.o : hello.s
gcc -c hello.s -o hello.o
hello.s : hello.i
gcc -S hello.i -o hello.s
hello.i : hello.c
gcc -E hello.c -o hello.i
目标依赖关系

03

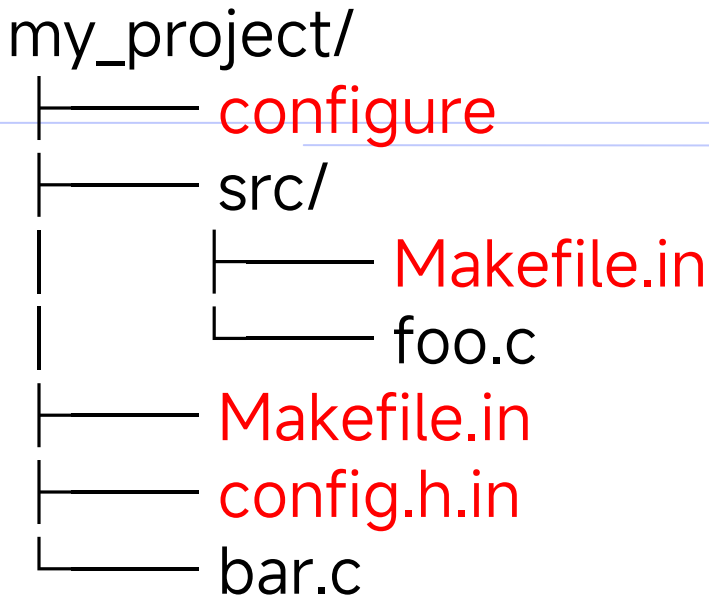
GNU Autotools

Autotools 是什么

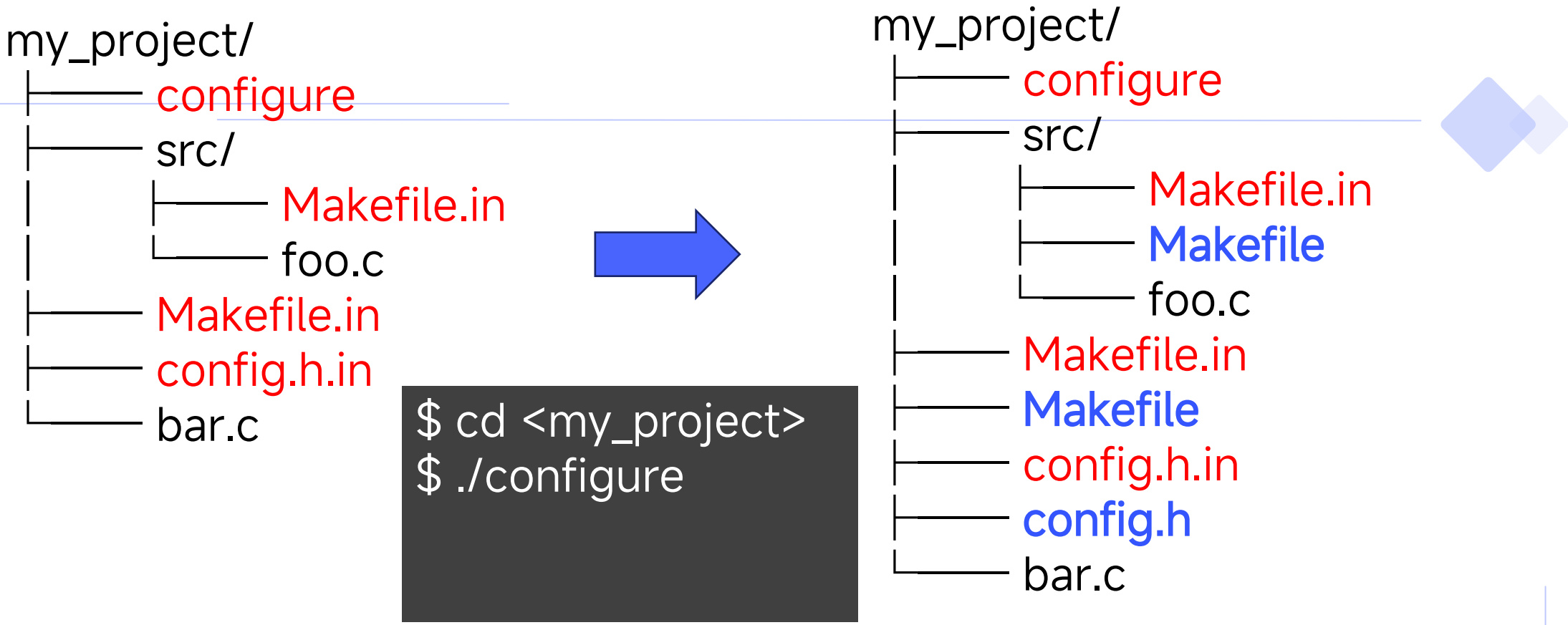
- Makefile 语法结构复杂，当项目非常大（特别是跨平台）的时候，维护 Makefile 比较困难。
- Autotools 工具就是专门用来帮助我们生成 Makefile 的，在很大程度上降低了开发的难度。



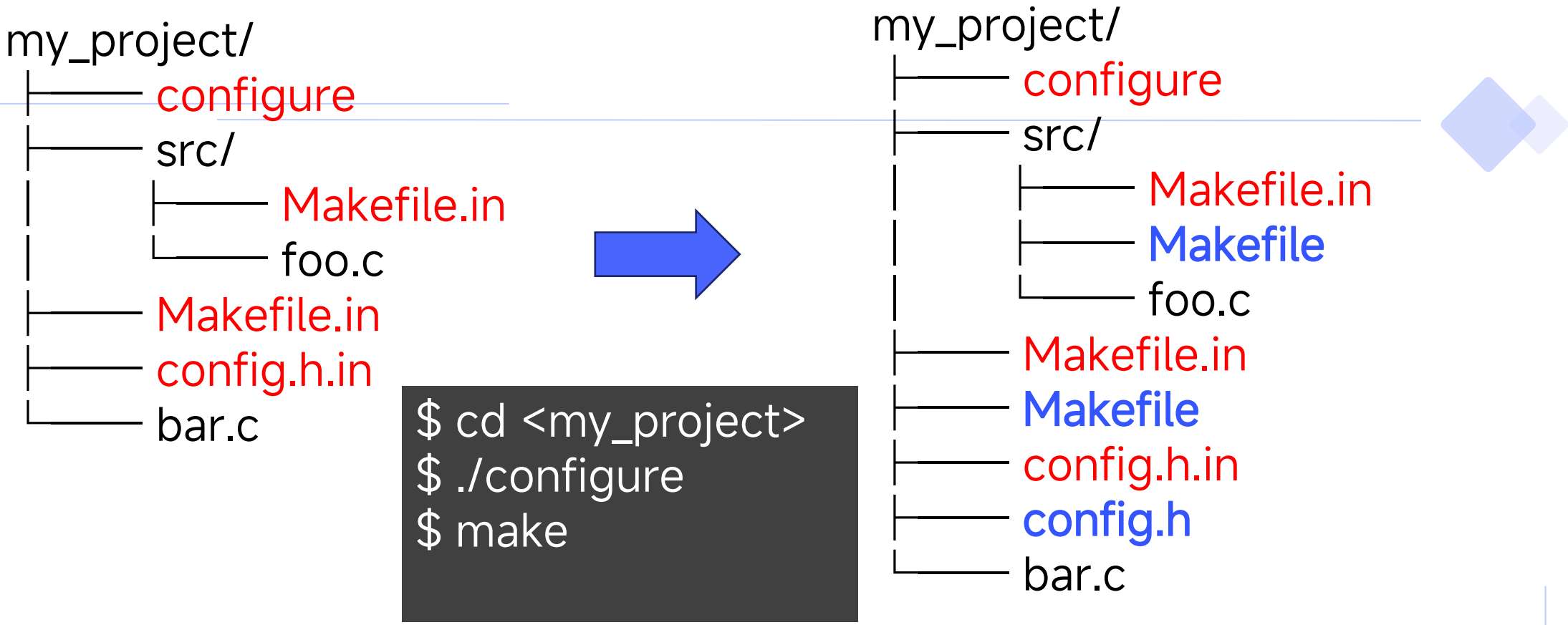
基于 Autotools 的新的构建步骤



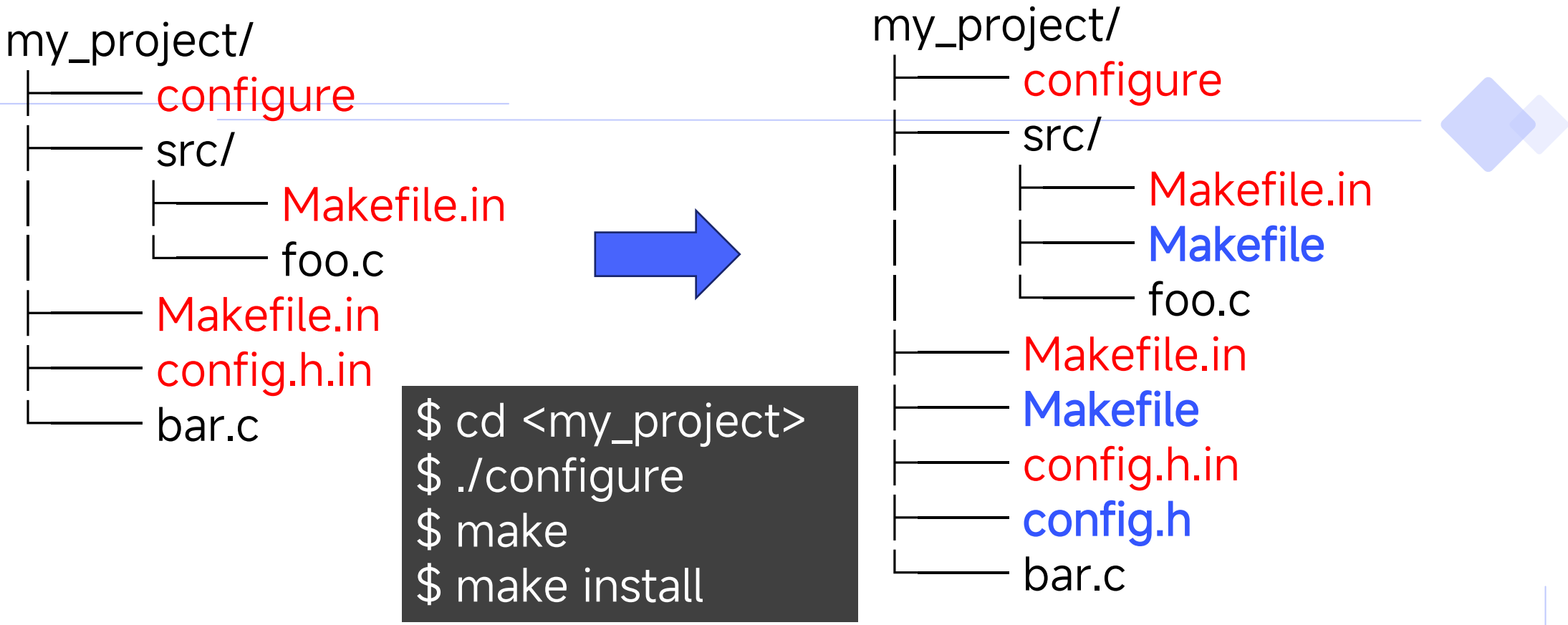
基于 Autotools 的新的构建步骤



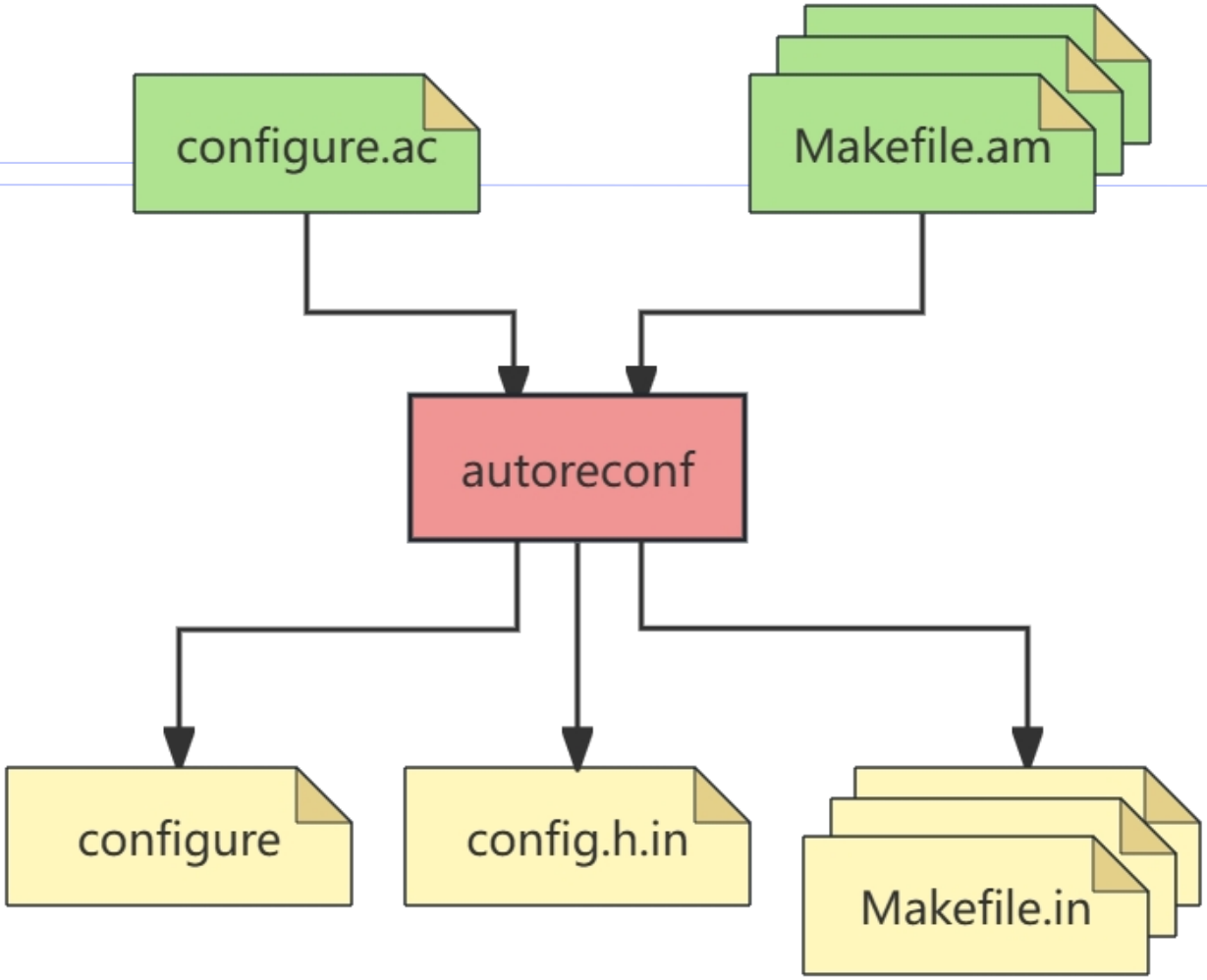
基于 Autotools 的新的构建步骤



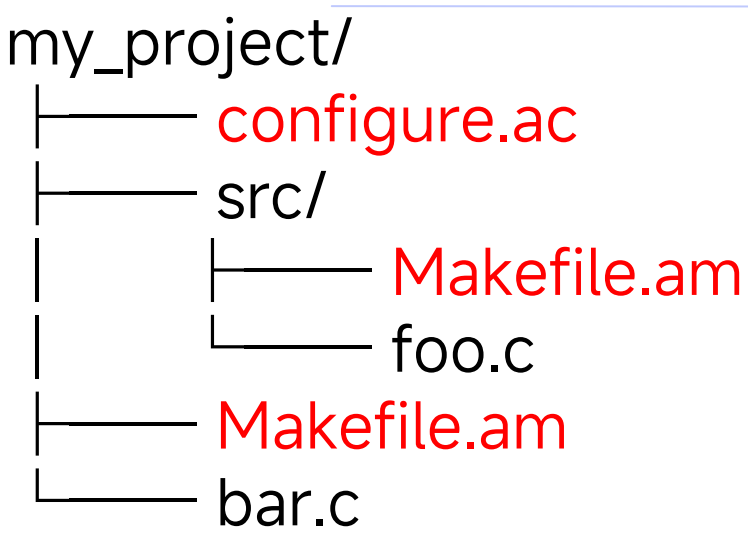
基于 Autotools 的新的构建步骤



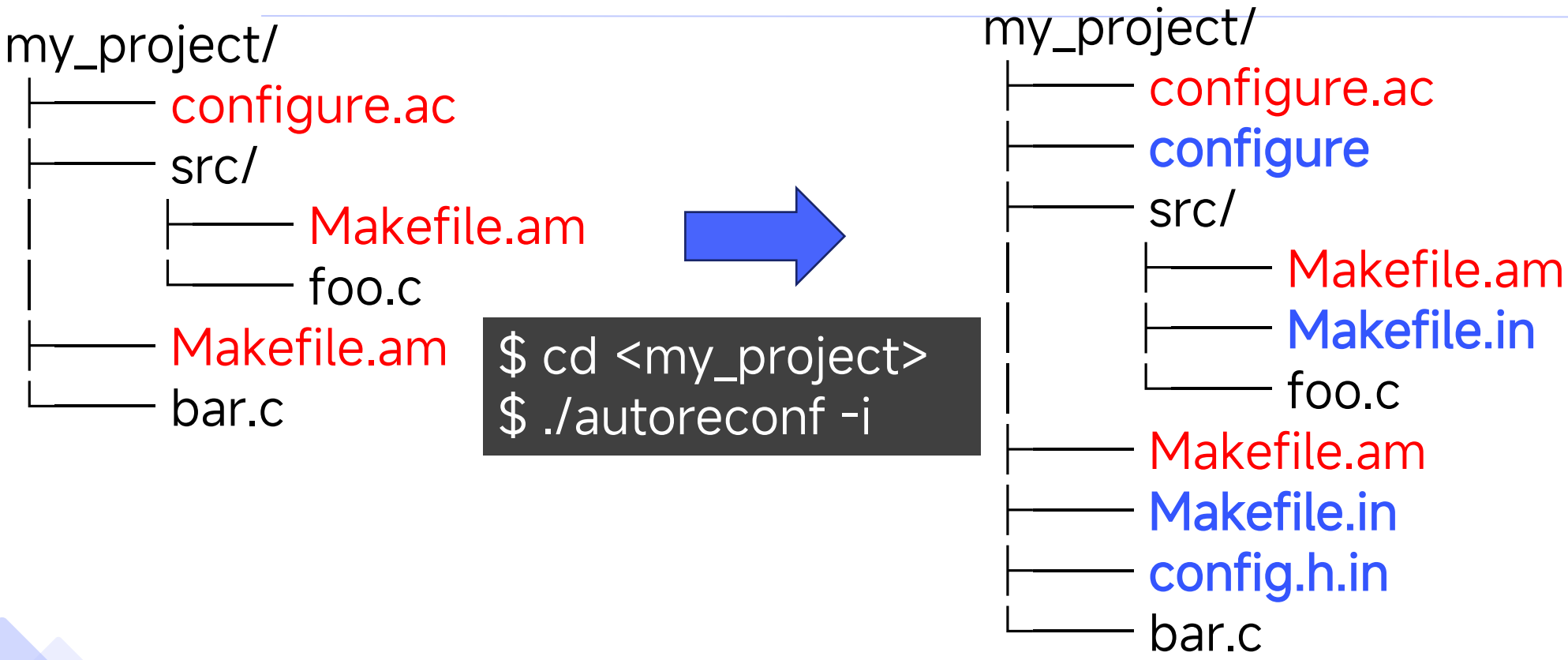
基于 Autotools 开发



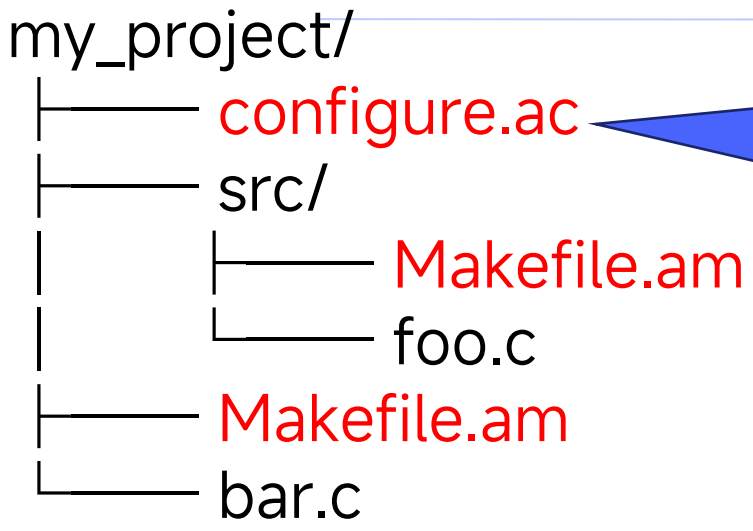
基于 Autotools 开发



基于 Autotools 开发



基于 Autotools 开发 - configure.ac



configure.ac 用于描述项目的特性、需要检查的编译器、库、头文件等系统依赖。

基于 Autotools 开发 - configure.ac

configure.ac 用于描述项目的特性、需要检查的编译器、库、头文件等系统依赖。

```
AC_PREREQ([2.71])
AC_INIT([pkgconf],[2.3.0],[https://github.com/pkgconf/pkgconf/issues/new])
AC_CONFIG_SRCDIR([cli/main.c])
AC_CONFIG_MACRO_DIR([m4])
AX_CHECK_COMPILE_FLAG([-Wall], [CFLAGS="$CFLAGS -Wall"])
AX_CHECK_COMPILE_FLAG([-Wextra], [CFLAGS="$CFLAGS -Wextra"])
AX_CHECK_COMPILE_FLAG([-Wformat=2], [CFLAGS="$CFLAGS -Wformat=2"])
AX_CHECK_COMPILE_FLAG([-std=gnu99], [CFLAGS="$CFLAGS -std=gnu99"], [
        AX_CHECK_COMPILE_FLAG([-std=c99], [CFLAGS="$CFLAGS -std=c99"])
])
AC_CONFIG_HEADERS([libpkgconf/config.h])
AC_CHECK_DECLS([strlcpy, strlcat, strndup], [], [], [[#include <string.h>]])
AC_CHECK_DECLS([reallocarray])
AC_CHECK_HEADERS([sys/stat.h])
……
```

pkgconf-2.3.0

基于 Autotools 开发 - configure.ac

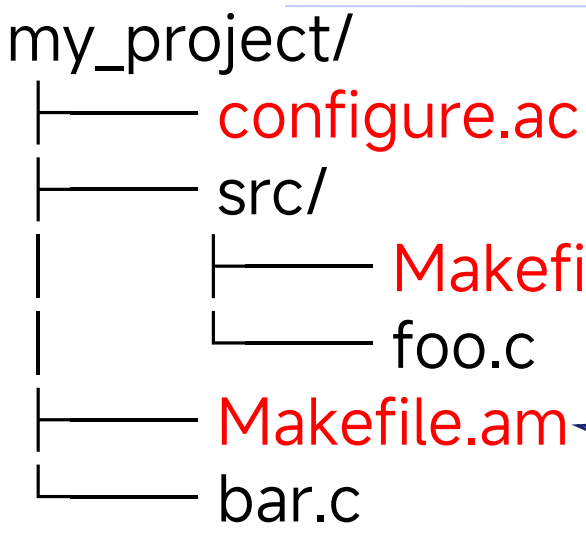
configure.ac 用于描述项目的特性、需要检查的编译器、库、头文件等系统依赖。

```
AC_PREREQ([2.71])
AC_INIT([pkgconf],[2.3.0],[https://github.com/pkgconf/pkgconf/issues/new])
AC_CONFIG_SRCDIR([cli/main.c])
AC_CONFIG_MACRO_DIR([m4])
AX_CHECK_COMPILE_FLAG([-Wall], [CFLAGS="$CFLAGS -Wall"])
AX_CHECK_COMPILE_FLAG([-Wextra], [CFLAGS="$CFLAGS -Wextra"])
AX_CHECK_COMPILE_FLAG([-Wformat=2], [CFLAGS="$CFLAGS -Wformat=2"])
AX_CHECK_COMPILE_FLAG([-std=gnu99], [CFLAGS="$CFLAGS -std=gnu99"], [
        AX_CHECK_COMPILE_FLAG([-std=c99], [CFLAGS="$CFLAGS -std=c99"])
])
AC_CONFIG_HEADERS([libpkgconf/config.h])
AC_CHECK_DECLS([strlcpy, strlcat, strndup], [], [], [[#include <string.h>]])
AC_CHECK_DECLS([reallocarray])
AC_CHECK_HEADERS([sys/stat.h])
……
```

pkgconf-2.3.0

```
checking for gcc... /usr/bin/gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether the compiler supports GNU C... yes
checking whether /usr/bin/gcc accepts -g... yes
checking for /usr/bin/gcc option to enable C11 features... none
needed
checking whether /usr/bin/gcc understands -c and -o together...
yes
checking whether C compiler accepts -Wall... yes
checking whether C compiler accepts -Wextra... yes
checking whether C compiler accepts -Wformat=2... yes
checking whether C compiler accepts -std=gnu99... yes
checking for /usr/bin/gcc options needed to detect all
undeclared functions... none needed
checking whether strlcpy is declared... no
checking whether strlcat is declared... no
checking whether strndup is declared... yes
checking for stdio.h... yes
checking for stdlib.h... yes
checking for string.h... yes
checking for inttypes.h... yes
checking for stdint.h... yes
checking for strings.h... yes
checking for sys/stat.h... yes
checking for sys/types.h... yes
checking for unistd.h... yes
checking whether reallocarray is declared... yes
```

基于 Autotools 开发 - Makefile.am



Makefile.am 用相对更简单的语法描述编译目标、依赖关系等。然后由 Automake 转换成标准且复杂的 Makefile.in，再由 configure 转换成最终的 Makefile。

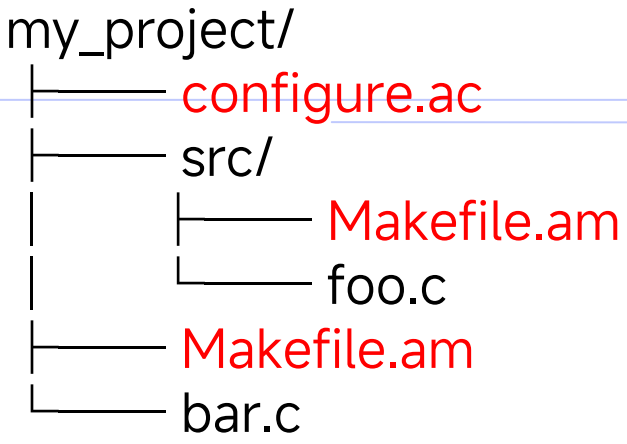
基于 Autotools 开发 - Makefile.am

Makefile.am 用相对更简单的语法描述编译目标（可执行文件、库）、依赖关系（需要哪些源文件、头文件）等。

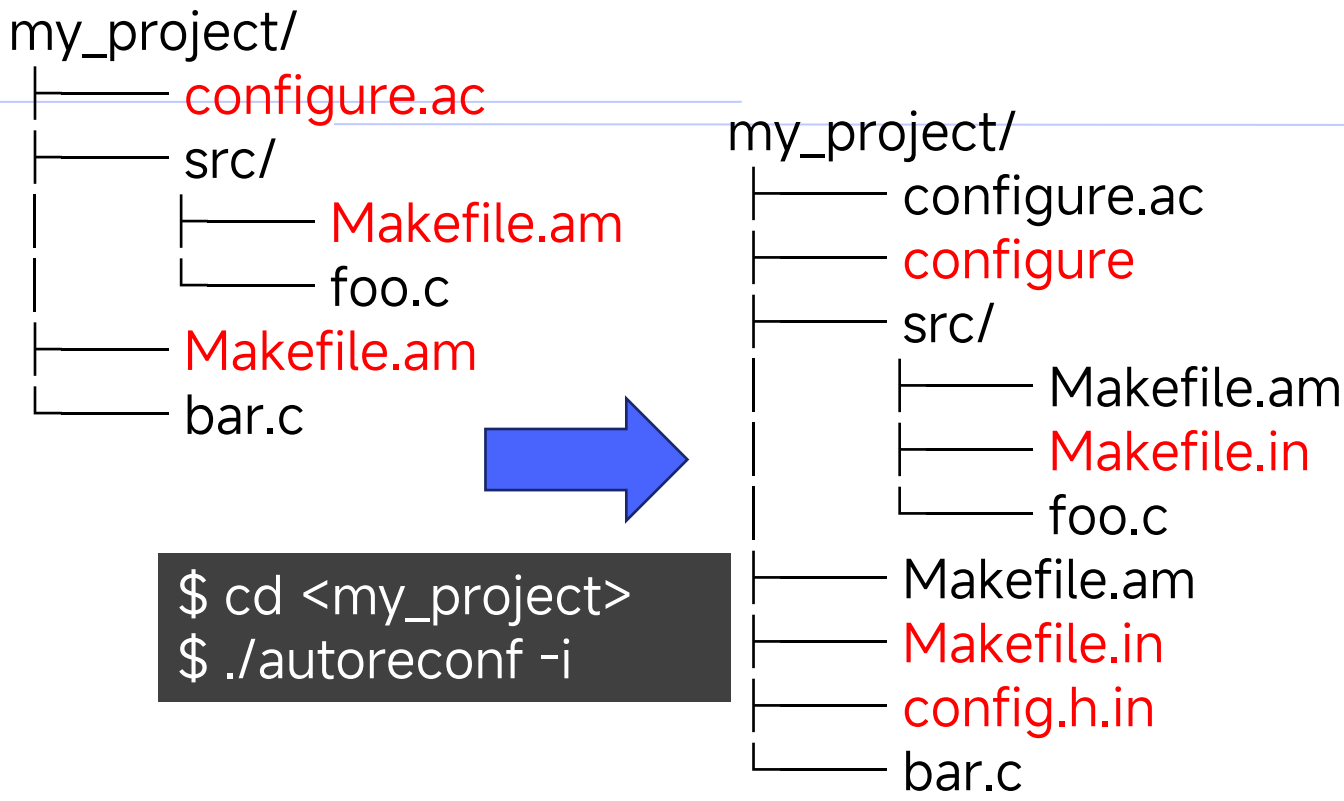
```
……
bin_PROGRAMS = pkgconf bomtool
lib_LTLIBRARIES = libpkgconf.la
……
libpkgconf_la_SOURCES  =              \
                libpkgconf/audit.c            \
                libpkgconf/cache.c            \
                libpkgconf/client.c           \
                libpkgconf/pkg.c              \
                ……
libpkgconf_la_LDFLAGS = -no-undefined -version-info 5:0:0 -export-symbols-regex
'^pkgconf_’
……
pkgconf_LDADD    = libpkgconf.la
pkgconf_SOURCES  =                  \
        cli/main.c                  \
        cli/getopt_long.c           \
        cli/renderer-msvc.c
pkgconf_CPPFLAGS = -I$(top_srcdir)/libpkgconf -I$(top_srcdir)/cli
……
```

pkgconf-2.3.0

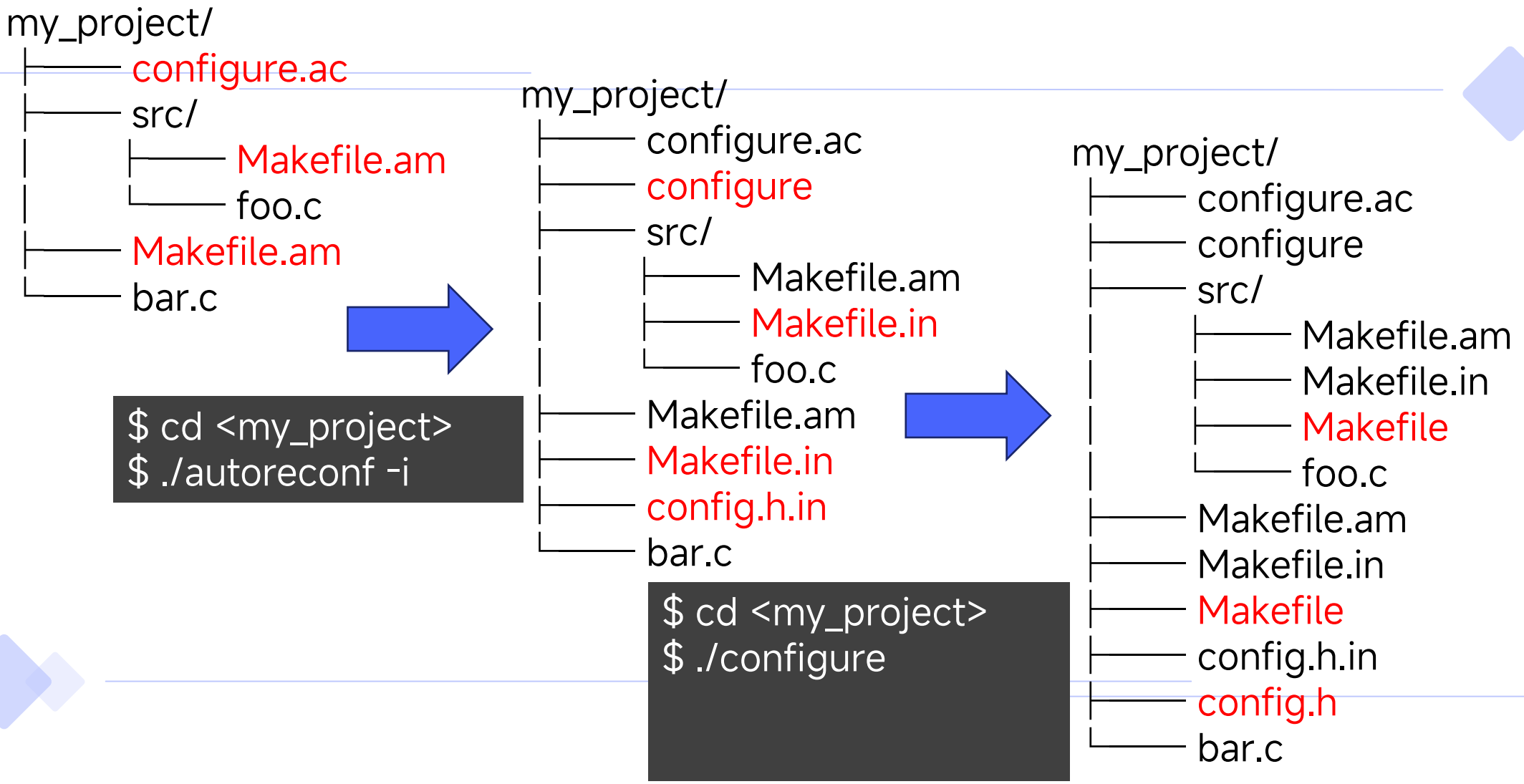
基于 Autotools 完整的开发和构建步骤



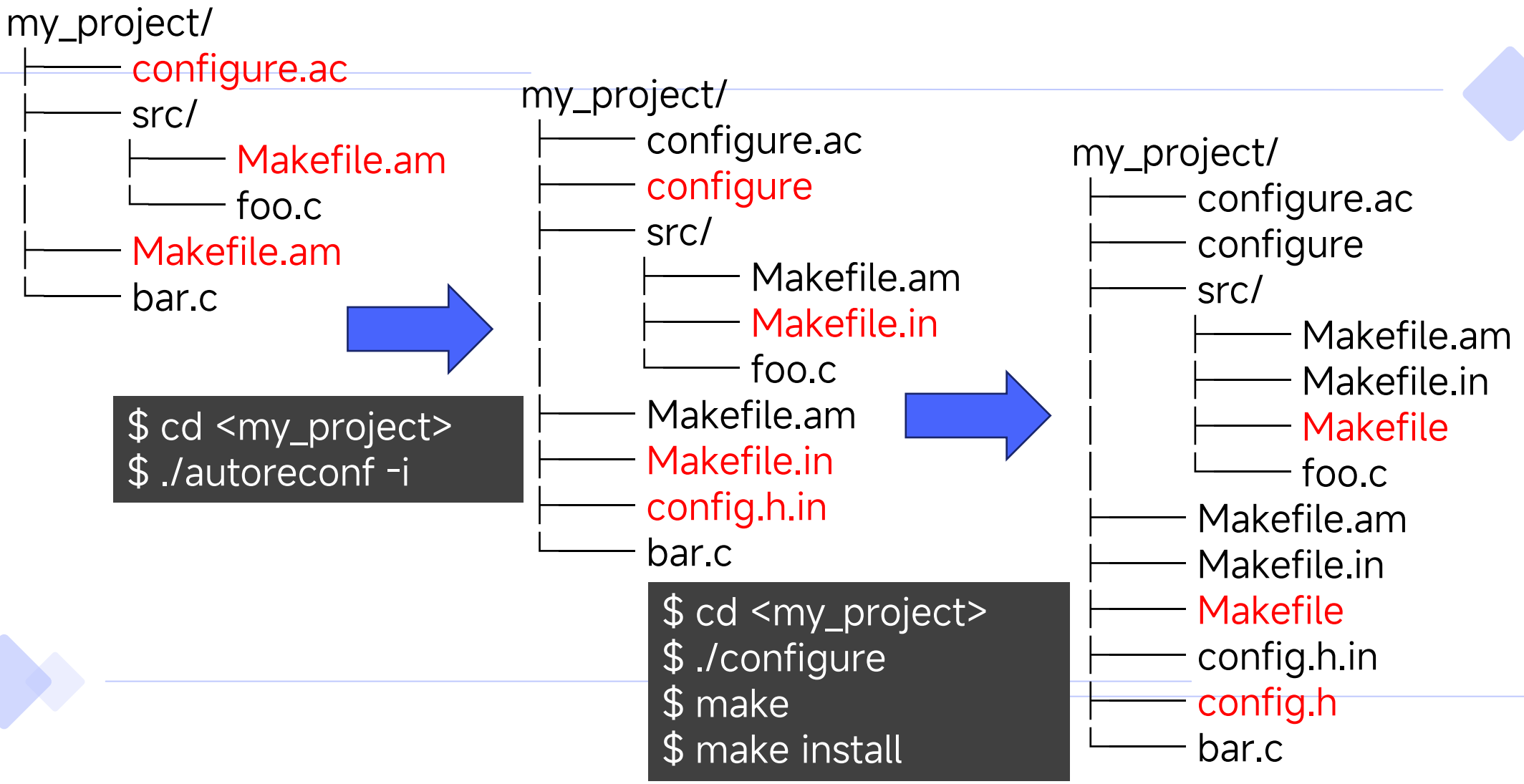
基于 Autotools 完整的开发和构建步骤



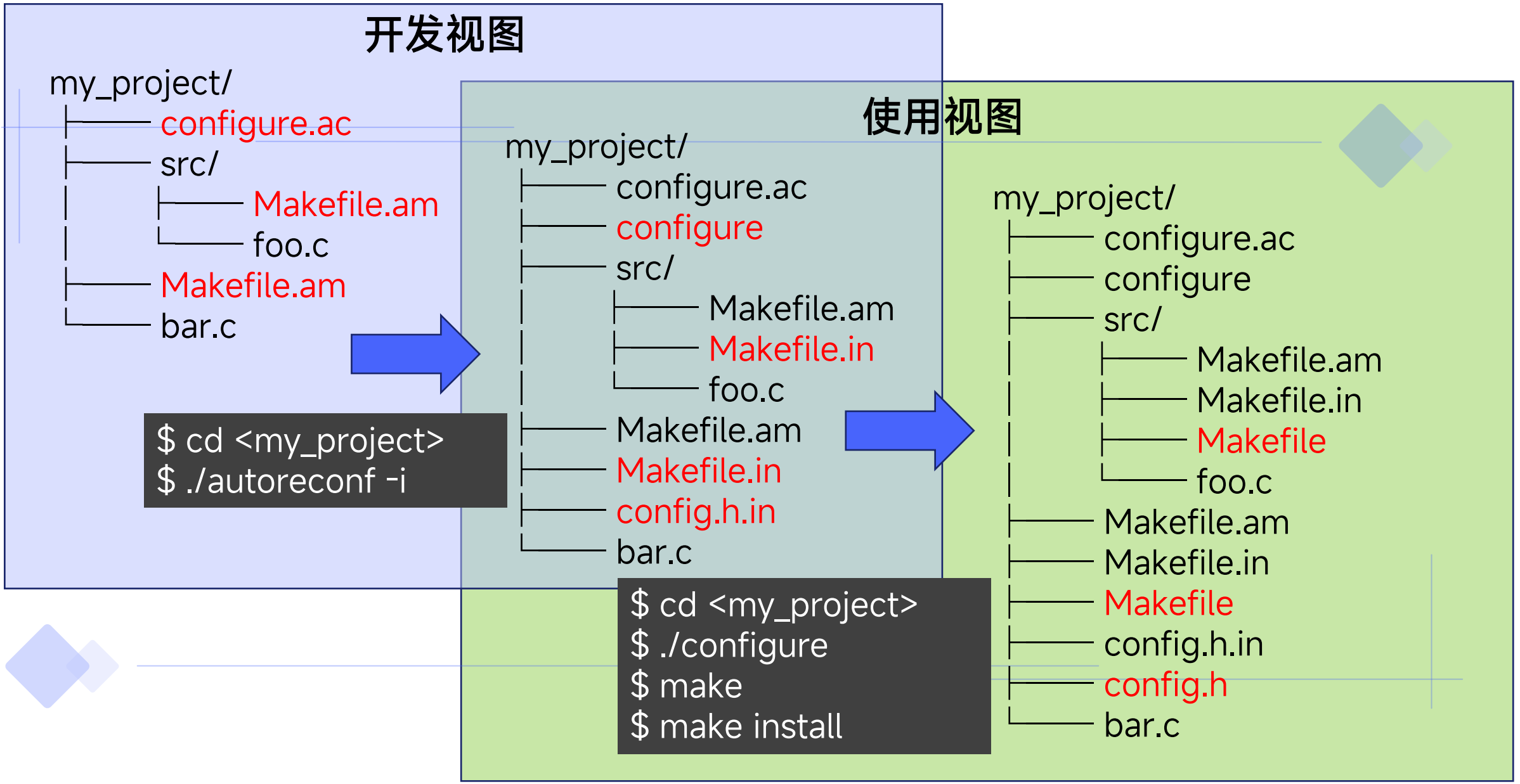
基于 Autotools 完整的开发和构建步骤



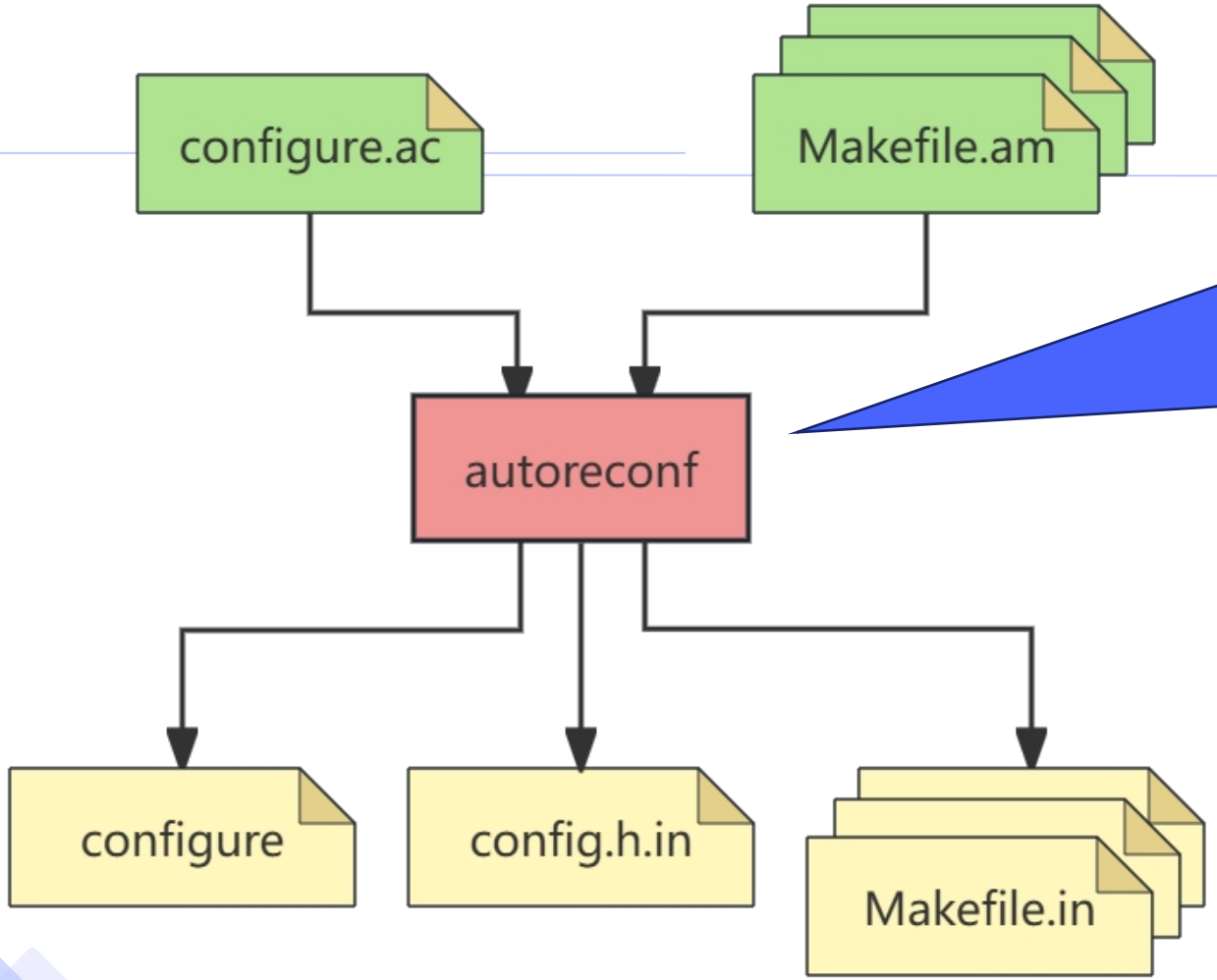
基于 Autotools 完整的开发和构建步骤



基于 Autotools 完整的开发和构建步骤



基于 Autotools 开发 - autoreconf 背后的秘密



autoreconf 本质上是一个封装脚本，它封装并自动调用了 autotools 软件包中的一系列底层工具。

基于 Autotools 开发 - autoreconf 背后的秘密

autotools 软件包的组成：

- autoscan：扫描源码，生成一个初始的 configure.scan 文件。将其重命名为 configure.ac
- aclocal：收集所有必需的 M4 宏，生成 aclocal.m4 文件。
- autoheader: 扫描 configure.ac 文件, 生成 config.h.in 模板头文件
- automake: 根据 configure.ac 和 Makefile.am 生成 Makefile.in 模板
- autoconf：根据 configure.ac 和 aclocal.m4，生成 configure 脚本。

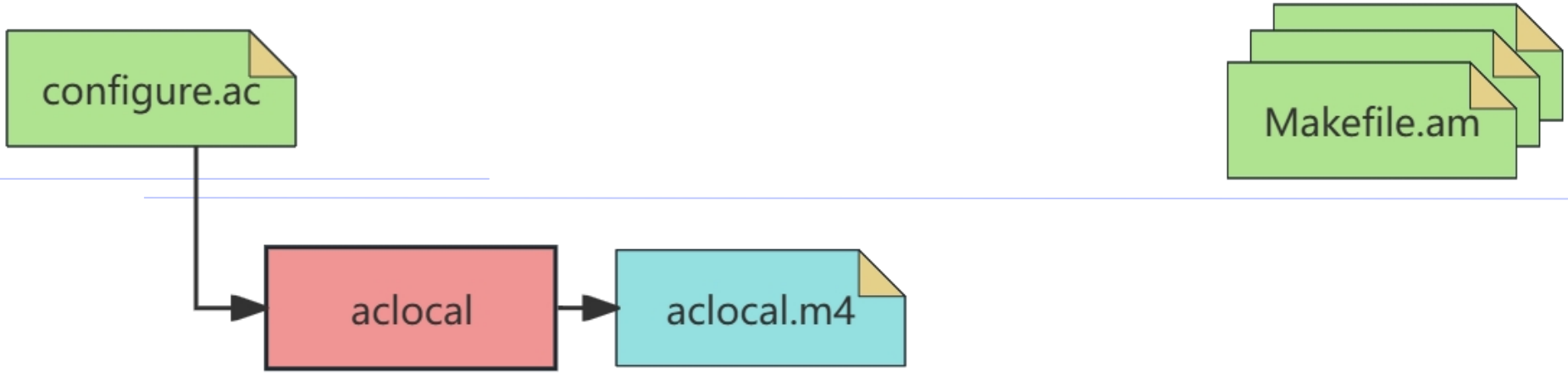
基于 Autotools 开发 - autoreconf 背后的秘密

使用 autoreconf 相当于自动执行了以下流程：

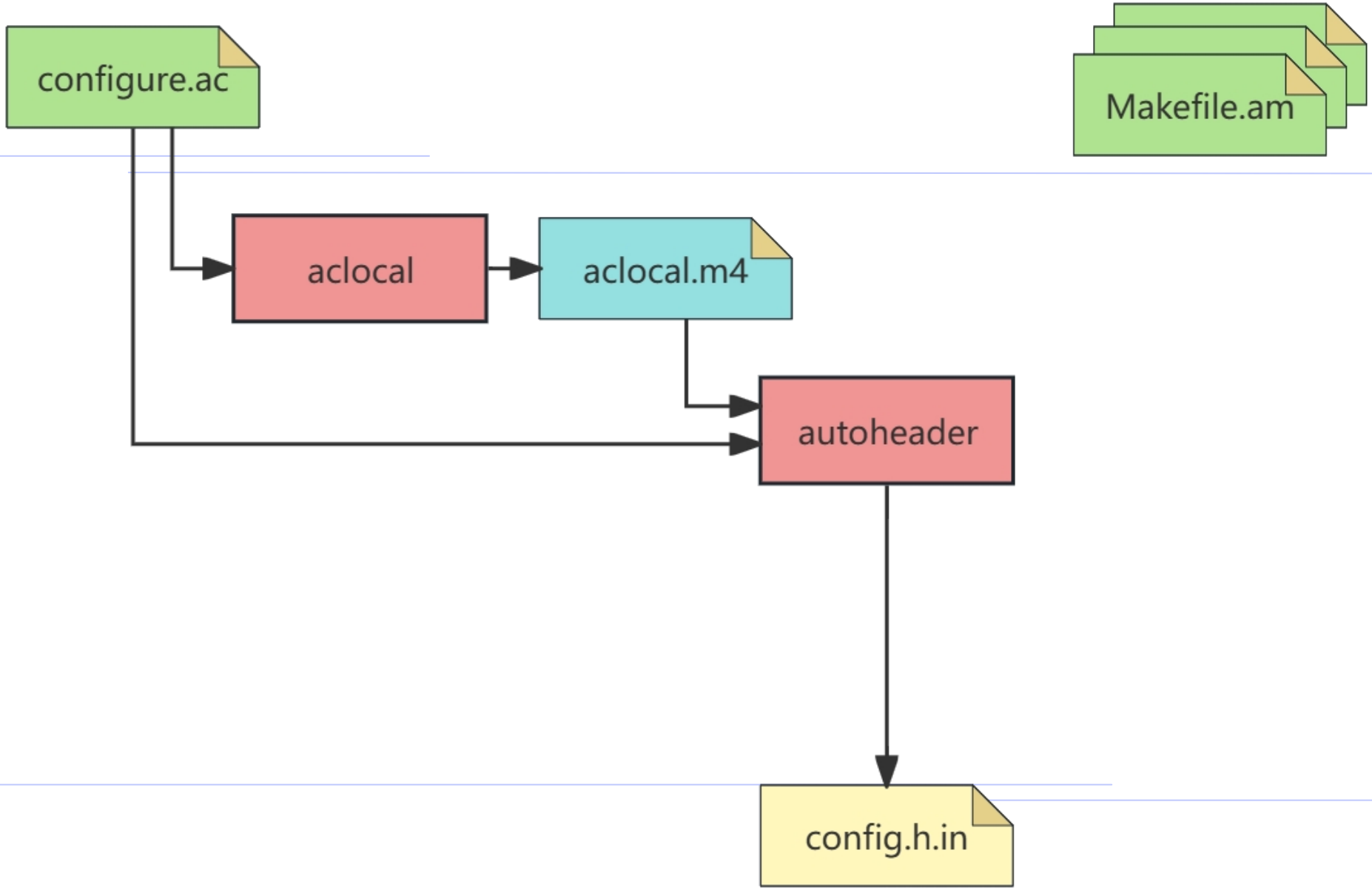
- aclocal --force -I m4
- autoheader --force
- automake --add-missing --copy --force-missing
- autoconf --force

基于 Autotools 开发 - autoreconf 背后的秘密

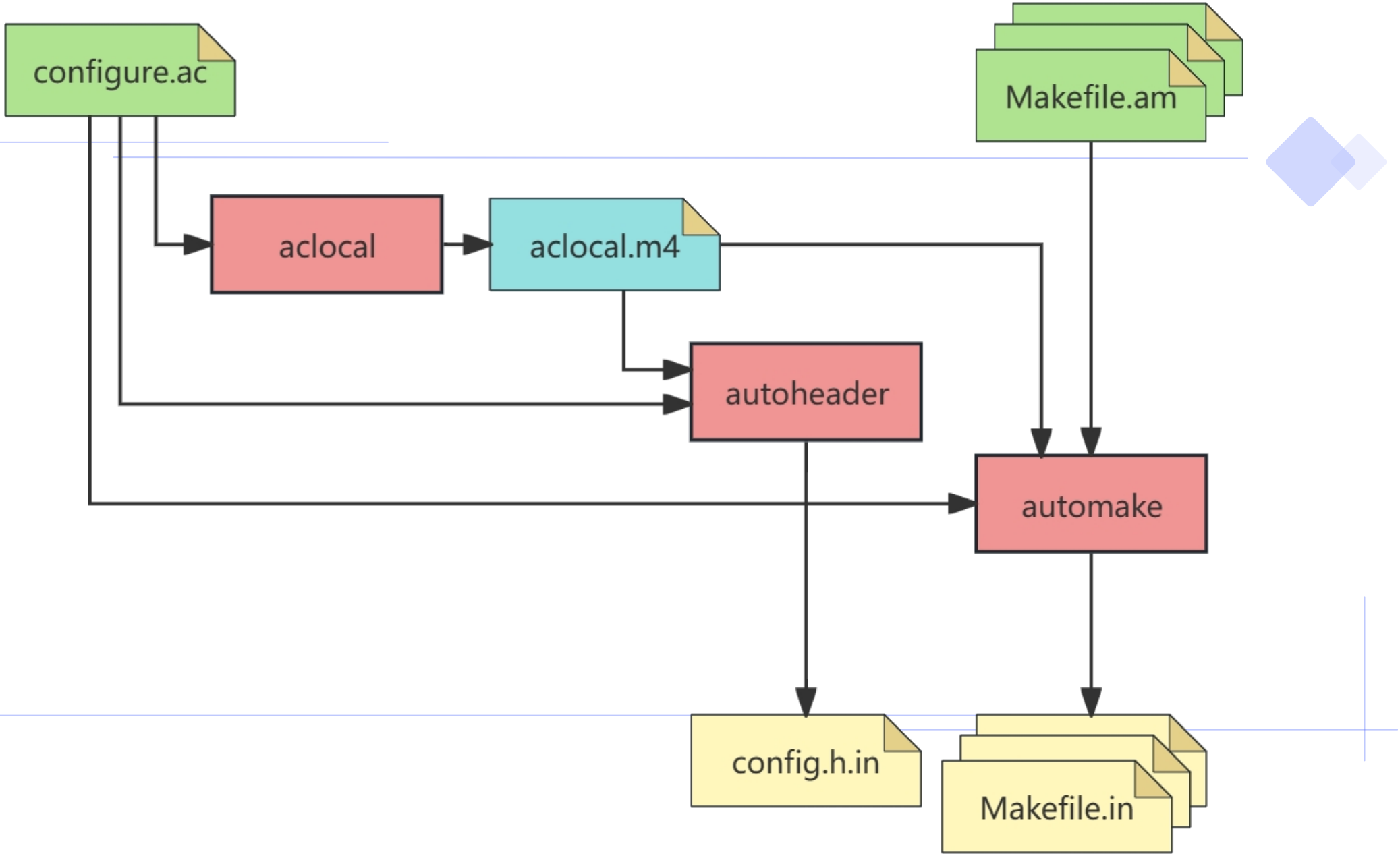
基于 Autotools 开发 - autoreconf 背后的秘密



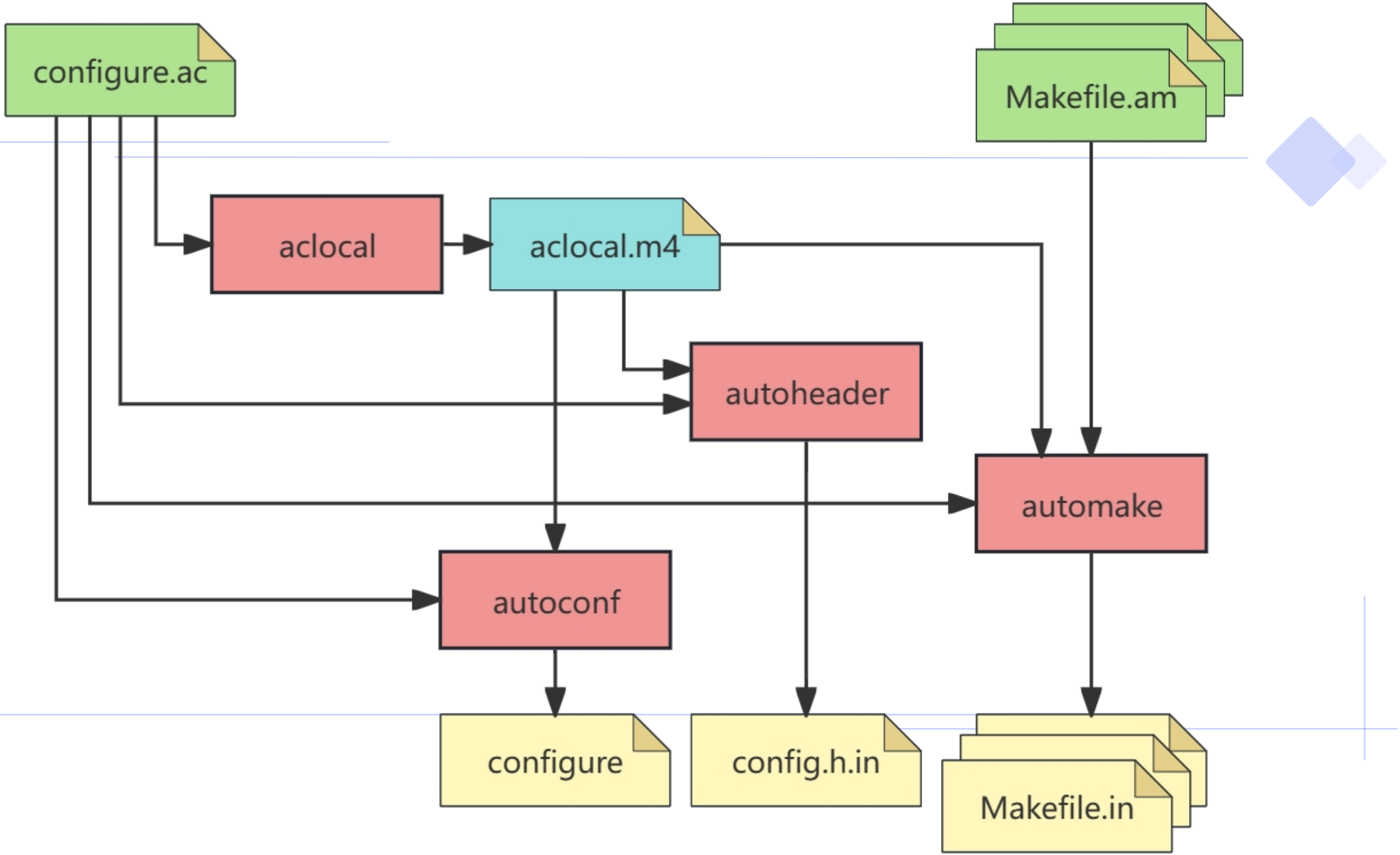
基于 Autotools 开发 - autoreconf 背后的秘密



基于 Autotools 开发 - autoreconf 背后的秘密



基于 Autotools 开发 - autoreconf 背后的秘密

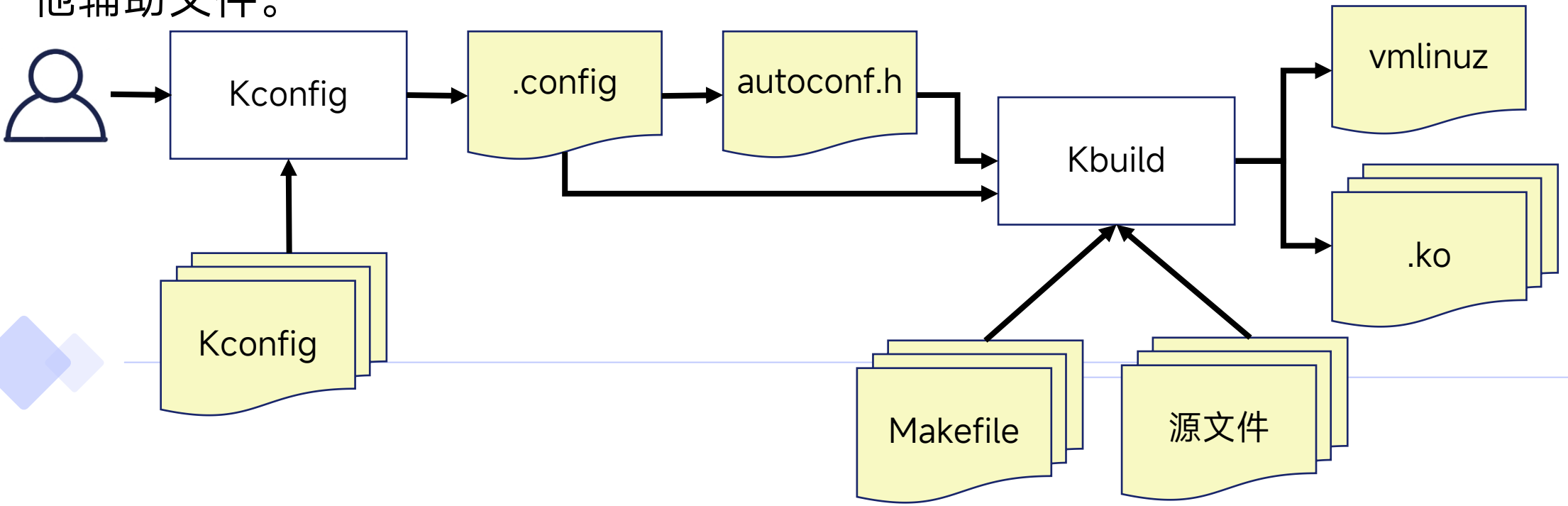


04

Kconfig & Kbuild

Kconfig & Kbuild 是什么

- **Kconfig** 是由 Linux 内核社区开发的、用于管理配置选项的系统。它的核心作用是：让用户能够方便地、有选择性地配置复杂的 Linux 内核项目，决定哪些部分需要被编译进最终的二进制内核文件中。
- **Kbuild** 是 Linux 内核的构建系统，它的名字来源于 Kernel Build System。它负责根据用户的配置（Kconfig 的输出）将成千上万个分散的内核源文件，头文件等最终编译和链接成内核镜像（vmlinuz）、内核模块（.ko）以及其他辅助文件。



Kconfig & Kbuild 是什么

Linux 内核使用专属的构建系统，而没有使用 Autotools 等是因为其项目的以下特点：

- **规模与复杂性**：内核需要为成千上万种不同的硬件配置和处理器架构进行编译。虽然 Autotools 也处理可移植性，但内核的规模和特定需求远超 Autotools 的设计常规。
- **编译模型特殊**：内核大部分代码是“要么全部编译，要么全不编译”或“编译为可加载模块”。这种精细的、基于配置的模块化编译不是 Autotools 的标准功能。
- **定制化需求极强**：内核构建过程需要处理许多独特的任务，例如生成系统调用表、管理符号版本、处理架构特定的汇编代码等。这些都需要一个完全定制化的构建系统来掌控。

它最初为 Linux 内核而设计，但现在已被许多其他开源项目（如 U-Boot、OpenSBI，Buildroot、BusyBox 等）所借鉴和采用。

Kconfig 的核心组成

- **Kconfig 文件**：配置文件，定义了所有可用的配置选项，定义了每个选项的类型（布尔值、字符串、整数等）、依赖关系、帮助信息以及它们之间的层次结构。它通常位于源代码树的各个子目录中，形成一个层次化的结构。
- **配置界面**：提供了一个用户操作界面，根据 Kconfig 文件定义的信息以图形化方式向用户展示选项并进行配置。
- **.config/autoconfig.h 文件**: 配置的输出文件，保存了用户通过配置界面做出的所有选择，供下一步 Kbuild 使用。

Kconfig - Kconfig 文件

arch/riscv/Kconfig

```
config MODULE_SECTIONS
        bool
        select HAVE_MOD_ARCH_SPECIFIC

config SMP
        bool "Symmetric Multi-Processing"
        help
          This enables support for systems with more than one CPU.  If
          you say N here, the kernel will run on single and
          multiprocessor machines, but will use only one CPU of a
          multiprocessor machine. If you say Y here, the kernel will run
          on many, but not all, single processor machines. On a single
          processor machine, the kernel will run faster if you say N
          here.

          If you don't know what to do here, say N.

config NR_CPUS
        int "Maximum number of CPUs (2-512)"
        depends on SMP
        range 2 512 if !RISCV_SBI_V01
        range 2 32 if RISCV_SBI_V01 && 32BIT
```

Kconfig - Kconfig 文件

arch/riscv/Kconfig

```
config MODULE_SECTIONS
        bool
        select HAVE_MOD_ARCH_SPECIFIC

config SMP
        bool "Symmetric Multi-Processing"
        help
          This enables support for systems with
          you say N here, the kernel will run o
          multiprocessor machines, but will use
          multiprocessor machine. If you say Y here, the kernel will run
          on many, but not all, single processor machines. On a single
          processor machine, the kernel will run faster if you say N
          here.

          If you don't know what to do here, say N.

config NR_CPUS
        int "Maximum number of CPUs (2-512)"
        depends on SMP
        range 2 512 if !RISCV_SBI_V01
        range 2 32 if RISCV_SBI_V01 && 32BIT
```

配置符号，每个配置项对应一个，最终在 .config 和 autoconf.h 中体现的符号会加上“CONFIG_”前缀

Kconfig - Kconfig 文件

arch/riscv/Kconfig

```
config MODULE_SECTIONS
        bool
        select HAVE_MOD_ARCH_SPECIFIC

config SMP
        bool "Symmetri
        help
          This enables support for syste
          you say N here, the kernel will run o
          multiprocessor machines, but will use
          multiprocessor machine. If you say Y here, the kernel will run
          on many, but not all, single processor machines. On a single
          processor machine, the kernel will run faster if you say N
          here.

          If you don't know what to do here, say N.

config NR_CPUS
        int "Maximum number of CPUs (2-512)"
        depends on SMP
        range 2 512 if !RISCV_SBI_V01
        range 2 32 if RISCV_SBI_V01 && 32BIT
```

配置符号，每个配置项对应一个，最终在 .config 和 autoconf.h 中体现的符号会加上“CONFIG_”前缀

Kconfig - Kconfig 文件

arch/riscv/Kconfig

```
config MODULE_SECTIONS
        bool
        select HAVE_MOD_ARCH_SPECIFIC

config SMP
        bool "Symmetric Multi-Processing"
        help
          This enables support for syste
          you say N here, the kernel
          multiprocessor machines,
          multiprocessor machine
          on many, but not all
          processor machine
          here.

          If you do
config NR_CPUS
        int "Maximum number of CPUs (2-512)"
        depends on SMP
        range 2 512 if !RISCV_SBI_V01
        range 2 32 if RISCV_SBI_V01 && 32BIT
```

配置符号，每个配置项对应一个，最终在 .config 和 autoconf.h 中体现的符号会加上“CONFIG_”前缀

Kconfig - Kconfig 文件

arch/riscv/Kconfig

```
config MODULE_SECTIONS
        bool
        select HAVE_MOD_ARCH_SPECIFIC

config SMP
        bool "Symmetric Multi-Processing"
        help
          This enables support for systems with more th      CPU.  If
          you say N here, the kernel will run on single and
          multiprocessor machines, but will use only one CPU of a
          multiprocessor machine. If you say Y here, the kernel will run
          on many, but not all, single processor machines. On a single
          processor machine, the kernel will run faster if you say N
          here.

          If you don't know what to do here, say N.

config NR_CPUS
        int "Maximum number of CPUs (2-512)"
        depends on SMP
        range 2 512 if !RISCV_SBI_V01
        range 2 32 if RISCV_SBI_V01 && 32BIT
```

标题，体现在配置界面中

帮助信息，体现在配置界面中

Kconfig - Kconfig 文件

arch/riscv/Kconfig

```
config MODULE_SECTIONS
        bool
        select HAVE_MOD_A

config SMP
        bool "Symmetric Multi-Processing"
        help
          This enables support for systems with
          you say N here, the kernel will run o
          multiprocessor machines, but will use
          multiprocessor machine. If you say Y
          on many, but not all, single processo
          processor machine, the kernel will ru
          here.

          If you don't know what to do here, say N.

config NR_CPUS
        int "Maximum number of CPUs (2-512)"
        depends on SMP
        range 2 512 if !RISCV_SBI_V01
        range 2 32 if RISCV_SBI_V01 && 32BIT
```

类型。最常见的是 bool，y 表示编译进内核 vmlinuz，或者 n 表示不参与编译。

还有其他类型：
tristate: y/n/m
string: 字符串
Int：整数

Kconfig - Kconfig 文件

arch/riscv/Kconfig

```
config MODULE_SECTIONS
        bool
        select HAVE_MOD_ARCH_SPE

config SMP
        bool "Symmetric Multi-Processing"
        help
          This enables support for systems with
          you say N here, the kernel will run o
          multiprocessor machines, but will use
          multiprocessor machine. If you say Y
          on many, but not all, single processo
          processor machine, the kernel will ru
          here.

          If you don't know what to do here, say N.

config NR_CPUS
        int "Maximum number of CPUs (2-512)"
        depends on SMP
        range 2 512 if !RISCV_SBI_V01
        range 2 32 if RISCV_SBI_V01 && 32BIT
```

类型。最常见的是 bool，y 表示编译进内核 vmlinuz，或者 n 表示不参与编译。

还有其他类型：
tristate: y/n/m
string: 字符串
int：整数

Kconfig - Kconfig 文件

arch/riscv/Kconfig

```
config MODULE_SECTIONS
        bool
        select HAVE_MOD_ARCH_SPECIFIC

config SMP
        bool "Symmetric Multi-Processing"
        help
          This enables support for systems with
          you say N here, the kernel will ru
          multiprocessor machines, but wi
          multiprocessor machine. If
          on many, but not all, si
          processor machine, th
          here.

          If you don'          what to do here, say N.

config NR_CPUS
        int "Maximum number of CPUs (2-512)"
        depends on SMP
        range 2 512 if !RISCV_SBI_V01
        range 2 32 if RISCV_SBI_V01 && 32BIT
```

类型。最常见的是 bool，y 表示编译进内核 vmlinuz，或者 n 表示不参与编译。

还有其他类型：
tristate: y/n/m
string: 字符串
int：整数

Kconfig - Kconfig 文件

arch/riscv/Kconfig

```
config MODULE_SECTIONS
        bool
        select HAVE_MOD_ARCH_SPECIFIC

config SMP
        bool "Symmetric Multi-Processing"
        help
          This enables support for systems with
          you say N here, the kernel will run o
          multiprocessor machines, but will use
          multiprocessor machine. If you say Y h
          on many, but not all, single processor machine
          processor machine, the kernel will run fast      ay N
          here.

          If you don't know what to d      , say N.

config NR_CPUS
        int "Maximum number of CPUs (2-512)"
        depends on SMP
        range 2 512 if !RISCV_SBI_V01
        range 2 32 if RISCV_SBI_V01 && 32BIT
```

依赖关系：
使用 depends on 关键字，表示当前选项只有在它所依赖的选项被使能时，才对用户可见并可配置。

Kconfig - Kconfig 文件

arch/riscv/Kconfig

```
config MODULE_SECTIONS
        bool
        select HAVE_MOD_ARCH_SPECIFIC

config SMP
        bool "Symmetric Multi-Processing
        help
          This enables support for systems with
          you say N here, the kernel will run on
          multiprocessor machines, but will use
          multiprocessor machine. If you say Y h
          on many, but not all, single processor
          processor machine, the kernel will run
          here.

          If you don't know what to do here, say N.

config NR_CPUS
        int "Maximum number of CPUs (2-512)"
        depends on SMP
        range 2 512 if !RISCV_SBI_V01
        range 2 32 if RISCV_SBI_V01 && 32BIT
```

反向依赖关系：
使用 select 关键字，表示如果当前选项被使能，则将强制使能另一个选项。

Kconfig - 配置界面

最常见的几种界面是：

- make menuconfig：基于文本的彩色菜单界面（最常用）。
- make nconfig：menuconfig 的增强版，界面更现代。
- make xconfig：基于 Qt 的图形化界面。
- make gconfig：基于 GTK 的图形化界面。
- make config：最简单的命令行逐项问答式配置。

Kconfig - 配置界面

最常见的几种界面是：

```
config MODULE_SECTIONS
        bool
        select HAVE_MOD_ARCH_SPECIFIC

config SMP
        bool "Symmetric Multi-Processing"
        help
          This enables support for systems with more than one CPU.  If
          you say N here, the kernel will run on single and
          multiprocessor machines, but will use only one CPU of a
          multiprocessor machine. If you say Y here, the kernel will run
          on many, but not all, single processor machines. On a single
          processor machine, the kernel will run faster if you say N
          here.

          If you don't know what to do here, say N.

config NR_CPUS
        int "Maximum number of CPUs (2-512)"
        depends on SMP
        range 2 512 if !RISCV_SBI_V01
        range 2 32 if RISCV_SBI_V01 && 32BIT
```

彩色菜单界面（最常用）。

增强版，界面更现代。

化界面。

形化界面。

逐项问答式配置。

Kconfig - 配置界面

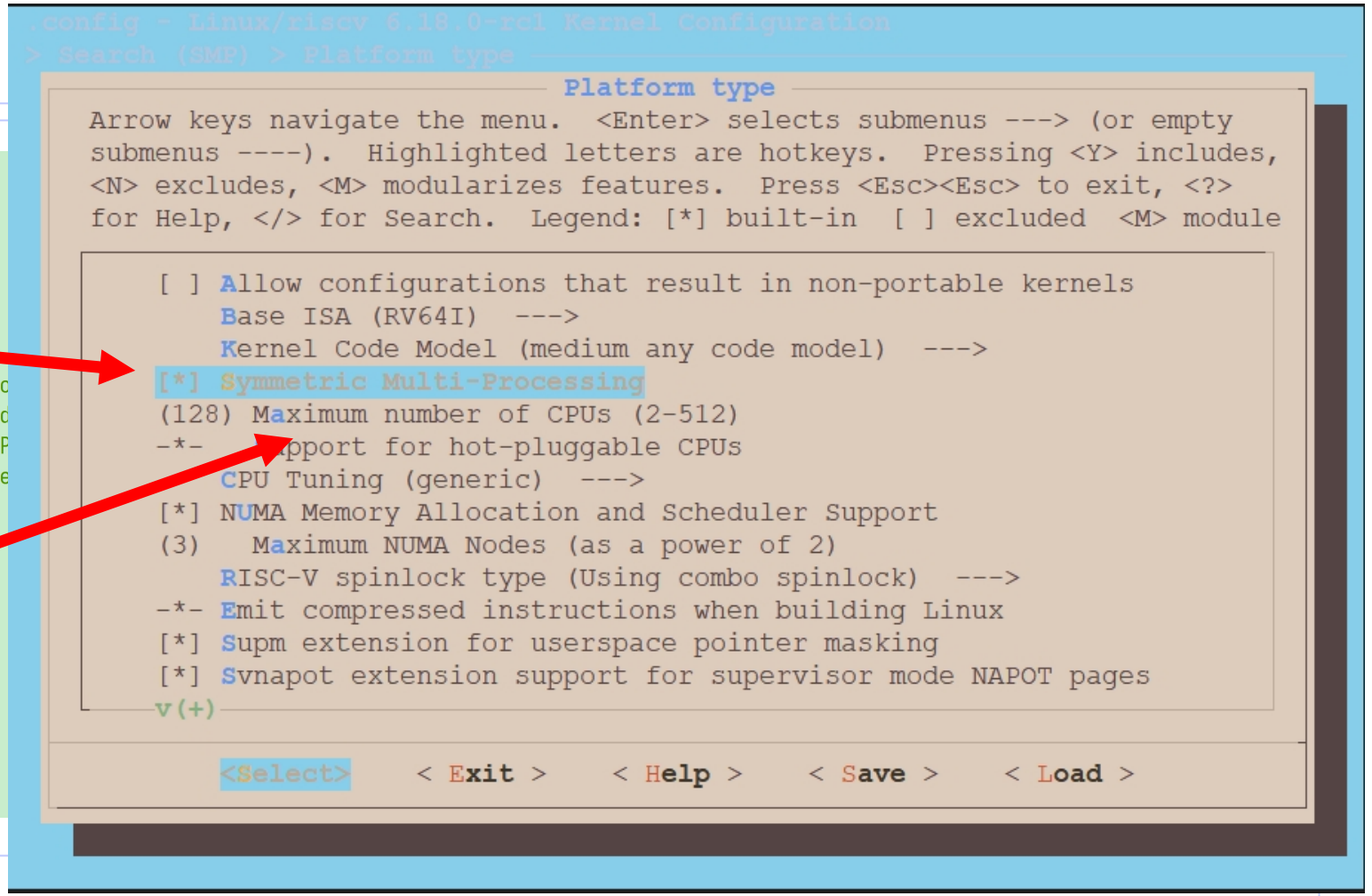
最常见的几种界面是：

```
config MODULE_SECTIONS
        bool
        select HAVE_MOD_ARCH_SPECIFIC

config SMP
        bool "Symmetric Multi-Processing"
        help
          This enables support for systems with more than o
          you say N here, the kernel will run on single and
          multiprocessor machines, but will use only one CP
          multiprocessor machine. If you say Y here, the ke
          on many, but not all, single processor machines.
          processor machine, the kernel will run faster if
          here.

          If you don't know what to do here, say N.

config NR_CPUS
        int "Maximum number of CPUs (2-512)"
        depends on SMP
        range 2 512 if !RISCV_SBI_V01
        range 2 32 if RISCV_SBI_V01 && 32BIT
```



Kconfig - .config 和 autoconf.h

.config

```
# CONFIG_NONPORTABLE is not set
CONFIG_ARCH_RV64I=y
# CONFIG_CMODEL_MEDLOW is not set
CONFIG_CMODEL_MEDANY=y
CONFIG_SMP=y
CONFIG_HOTPLUG_CPU=y
CONFIG_TUNE_GENERIC=y
# CONFIG_RISCV_TICKET_SPINLOCKS is not set
CONFIG_RISCV_COMBO_SPINLOCKS=y
CONFIG_RISCV_ALTERNATIVE=y
CONFIG_RISCV_ALTERNATIVE_EARLY=y
CONFIG_RISCV_ISA_C=y
CONFIG_RISCV_ISA_SUPM=y
```

include/generated/autoconf.h

```
#define CONFIG_NET_VENDOR_REALTEK 1
#define CONFIG_OVERLAY_FS_MODULE 1
#define CONFIG_VT 1
#define CONFIG_LIBNVDIMM 1
#define CONFIG_SMP 1
#define CONFIG_ARCH_MMAP_RND_COMPAT_BITS 8
#define CONFIG_SUN55I_PCK600 1
#define CONFIG_ARCH_SUPPORTS_MSEAL_SYSTEM_MAPPINGS 1
#define CONFIG_HID_REDRAGON 1
#define CONFIG_SECURITYFS 1
#define CONFIG_QUEUED_RWLOCKS 1
#define CONFIG_LSM_MMAP_MIN_ADDR 65536
#define CONFIG_ARCH_HAS_VDSO_ARCH_DATA 1
#define CONFIG_HAVE_ARCH_AUDITSYSCALL 1
```

Kbuild 的核心组成

- **顶层 Makefile**：位于内核源码的根目录，是构建过程的起点，定义了全局的变量、规则和目标（如 make vmlinux, make modules）；负责组织整个源码树，并 ***递归*** 地进入内核源码的各个子目录（如 kernel/, drivers/, fs/），调用这些子目录中的 Makefile（通常名为 Makefile 或 Kbuild）来构建该目录下的目标。
- **子目录 Makefiles / Kbuild 文件**：位于每个内核子目录中，定义本目录下需要编译的源文件列表。
- **脚本与辅助工具**：Kbuild 本身包含很多脚本（如 scripts/Makefile.*），这些脚本定义了构建系统的核心逻辑，例如如何构建 vmlinux，如何处理模块，如何生成依赖关系等。以及其他工具：如用于生成依赖信息的 fixdep，用于解决符号版本的 modpost 等。

Kbuild 的关键语法

arch/riscv/kernel/Makefile

```
obj-y   += riscv_ksyms.o
obj-y   += stacktrace.o
obj-y   += cacheinfo.o
obj-y   += patch.o
obj-y   += vendor_extensions.o
obj-y   += vendor_extensions/
obj-y   += probes/
obj-y   += tests/
obj-$(CONFIG_MMU) += vdso.o vdso/

obj-$(CONFIG_RISCV_MISALIGNED)  += traps_misaligned.o
obj-$(CONFIG_RISCV_MISALIGNED)  += unaligned_access_speed.o
obj-$(CONFIG_RISCV_PROBE_UNALIGNED_ACCESS)              += copy-unaligned.o
obj-$(CONFIG_RISCV_PROBE_VECTOR_UNALIGNED_ACCESS)       += vec-copy-unaligned.o
```

Kbuild 的关键语法

arch/riscv/kernel/Makefile

```
obj-y    += riscv_ksyms.o
obj-y    += stacktrace.o
obj-y    += cacheinfo.o
obj-y    += patch.o
obj-y    += vendor_extensions.o
obj-y    += vendor_extensions/
obj-y    += probes/
obj-y    += tests/
obj-$(CONFIG_MMU) += vdso.o vdso/

obj-$(CONFIG_RISCV_MISALIGNED)  += traps_misaligned.o
obj-$(CONFIG_RISCV_MISALIGNED)  += unaligned_access_speed.o
obj-$(CONFIG_RISCV_PROBE_UNALIGNED_ACCESS)              += copy-unaligned.o
obj-$(CONFIG_RISCV_PROBE_VECTOR_UNALIGNED_ACCESS)       += vec-copy-unaligned.o
```

obj-y: 列出要编译并链接进内核 vmlinuz 的目标文件

obj-m: 要编译成可加载内核模块（.ko）的目标文件

Kbuild 的关键语法

arch/riscv/kernel/Makefile

```
obj-y    += riscv_ksyms.o
obj-y    += stacktrace.o
obj-y    += cacheinfo.o
obj-y    += patch.o
obj-y    += vendor_extensions.o
obj-y    += vendor_extensions/
obj-y    += probes/
obj-y    += tests/
obj-$(CONFIG_MMU) += vdso.o vdso/

obj-$(CONFIG_RISCV_MISALIGNED)  += traps_misaligned.o
obj-$(CONFIG_RISCV_MISALIGNED)  += unaligned_access_speed.o
obj-$(CONFIG_RISCV_PROBE_UNALIGNED_ACCESS)              += copy-unaligned.o
obj-$(CONFIG_RISCV_PROBE_VECTOR_UNALIGNED_ACCESS)       += vec-copy-unaligned.o
```

根据 .config 中相应配置项的取指来决定编译行为

MMU 取值 y、m 或者 n

Kbuild 的关键语法

arch/riscv/kernel/Makefile

```
obj-y    += riscv_ksyms.o
obj-y    += stacktrace.o
obj-y    += cacheinfo.o
obj-y    += patch.o
obj-y    += vendor_extensions.o
obj-y    += vendor_extensions/
obj-y    += probes/
obj-y    += tests/
obj-$(CONFIG_MMU) += vdso.o vdso/

obj-$(CONFIG_RISCV_MISALIGNED)  += traps_misaligned.o
obj-$(CONFIG_RISCV_MISALIGNED)  += unaligned_access_speed.o
obj-$(CONFIG_RISCV_PROBE_UNALIGNED_ACCESS)              += copy-unaligned.o
obj-$(CONFIG_RISCV_PROBE_VECTOR_UNALIGNED_ACCESS)       += vec-copy-unaligned.o
```

根据 .config 中相应配置项的取指来决定编译行为

RISCV_MISALIGNED 取值 y、m 或者 n

Kbuild 的关键语法

drivers/clk/sophgo/Makefile

```
# SPDX-License-Identifier: GPL-2.0
obj-$(CONFIG_CLK_SOPHGO_CV1800) += clk-sophgo-cv1800.o

clk-sophgo-cv1800-y             += clk-cv1800.o
clk-sophgo-cv1800-y             += clk-cv18xx-common.o
clk-sophgo-cv1800-y             += clk-cv18xx-ip.o
clk-sophgo-cv1800-y             += clk-cv18xx-pll.o

obj-$(CONFIG_CLK_SOPHGO_SG2
obj-$(CONFIG_CLK_SOPHGO_SG2042_
obj-$(CONFIG_CLK_SOPHGO_SG2042_RPU
obj-$(CONFIG_CLK_SOPHGO_SG2044)
obj-$(CONFIG_CLK_SOPHGO_SG2044_PLL
```

当 CLK_SOPHGO_CV1800 取值为 m 时，clk-sophgo-cv1800.ko 由多个 .o 链接而成。

Kbuild 的关键语法

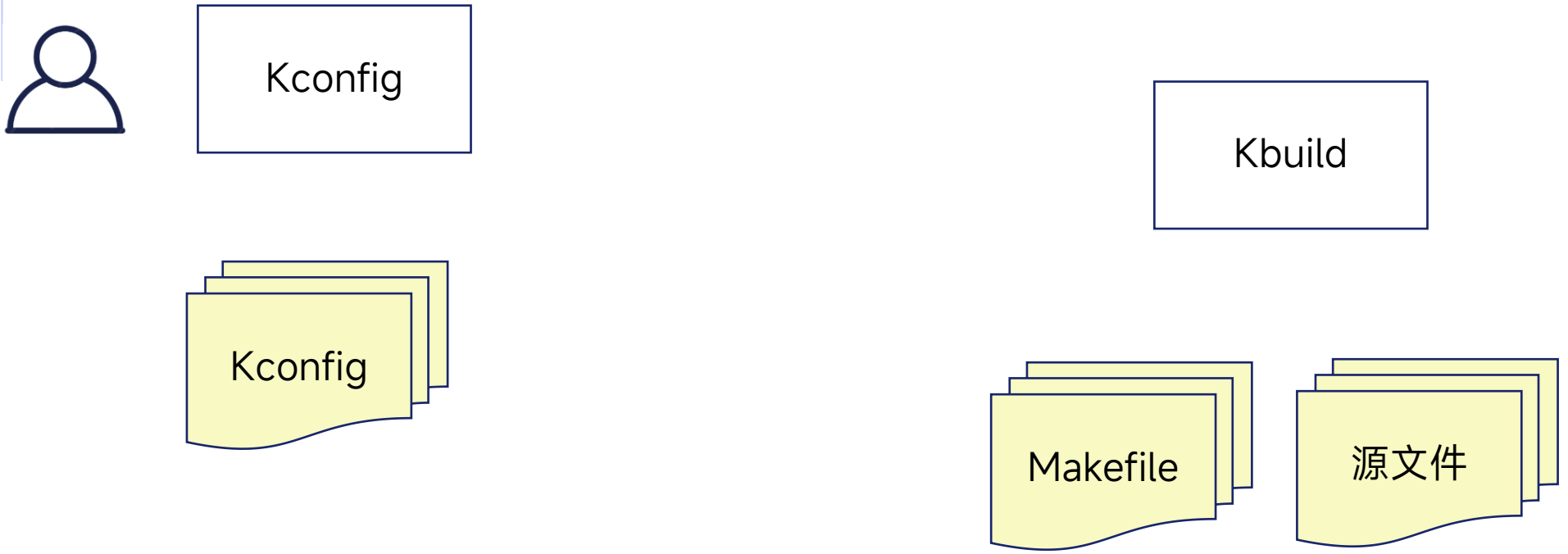
arch/riscv/kernel/Makefile

```
obj-y    += riscv_ksyms.o
obj-y    += stacktrace.o
obj-y    += cacheinfo.o
obj-y    += patch.o
obj-y    += vendor_extensions.o
obj-y    += vendor_extensions/
obj-y    += probes/
obj-y    += tests/
obj-$(CONFIG_MMU) += vdso.o vdso/

obj-$(CONFIG_RISCV_MISALIGNED)  += traps_misaligned.o
obj-$(CONFIG_RISCV_MISALIGNED)  += unaligned_access_speed.o
obj-$(CONFIG_RISCV_PROBE_UNALIGNED_ACCESS)              += copy-unaligned.o
obj-$(CONFIG_RISCV_PROBE_VECTOR_UNALIGNED_ACCESS)       += vec-copy-unaligned.o
```

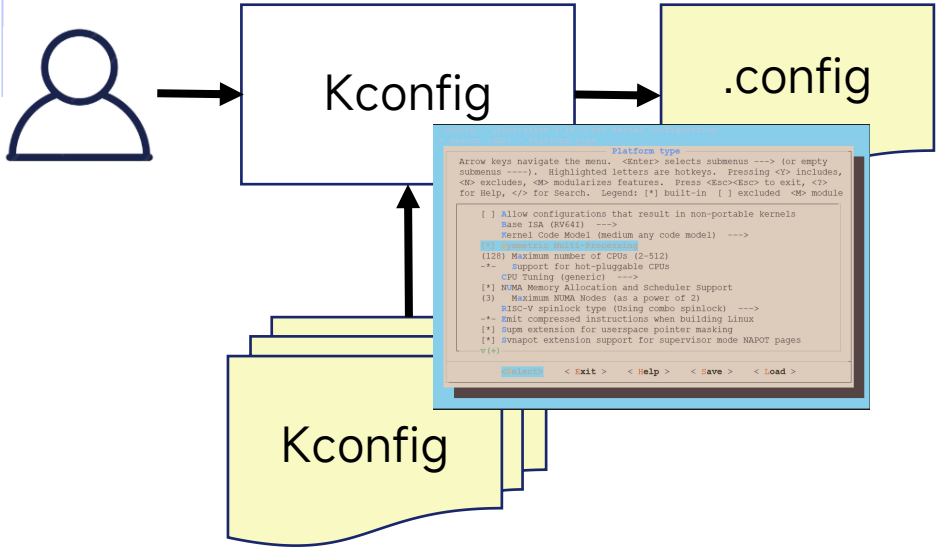
指定是否进入该子目录继续构建。

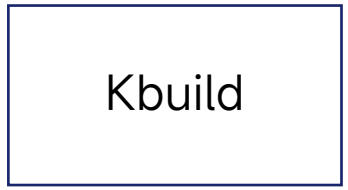
完整的内核构建流程

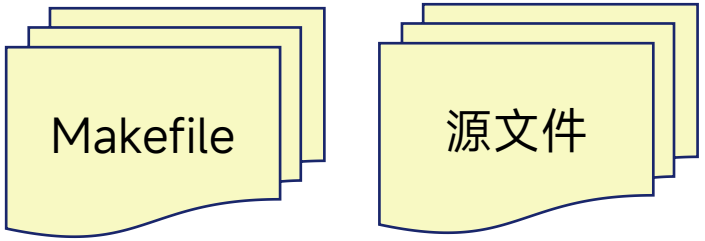


完整的内核构建流程

```
$ make menuconfig
```

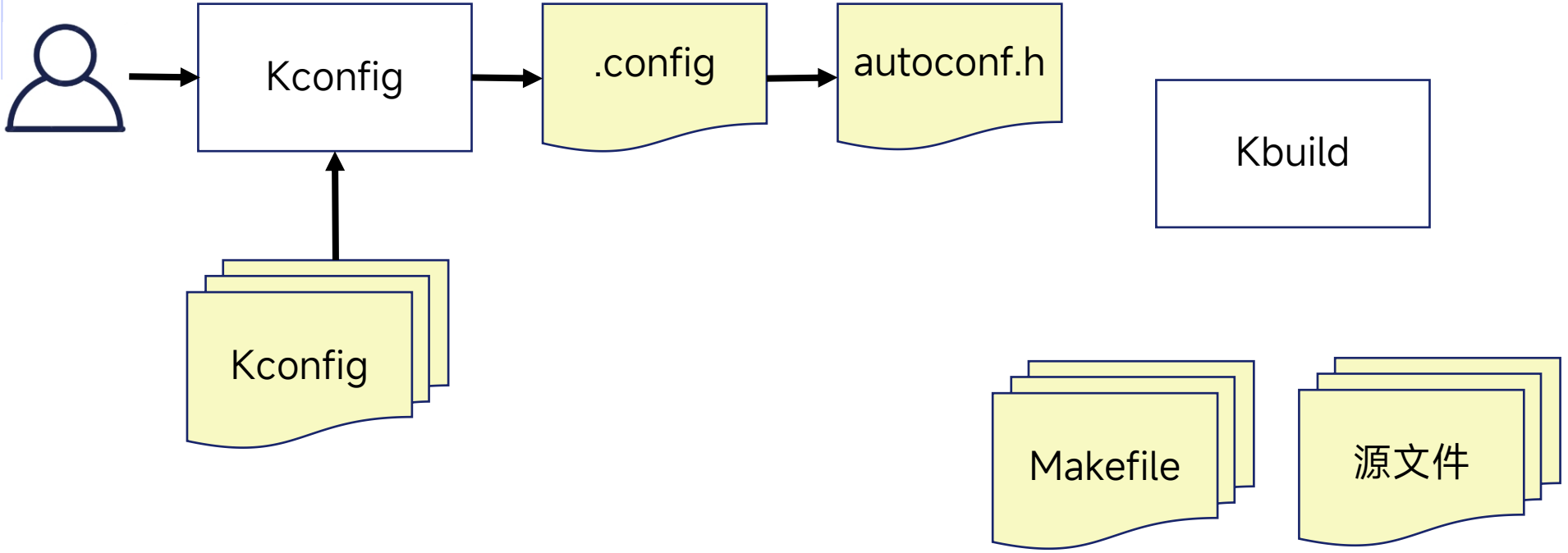






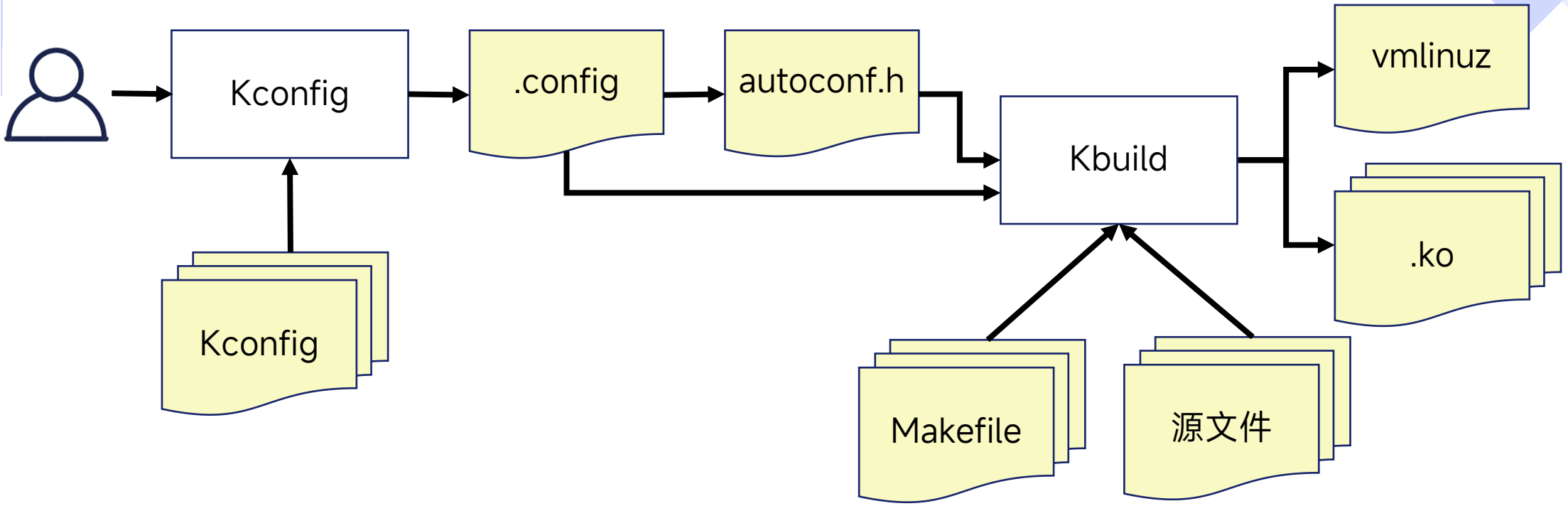
完整的内核构建流程

```
$ make menuconfig
$ make
```

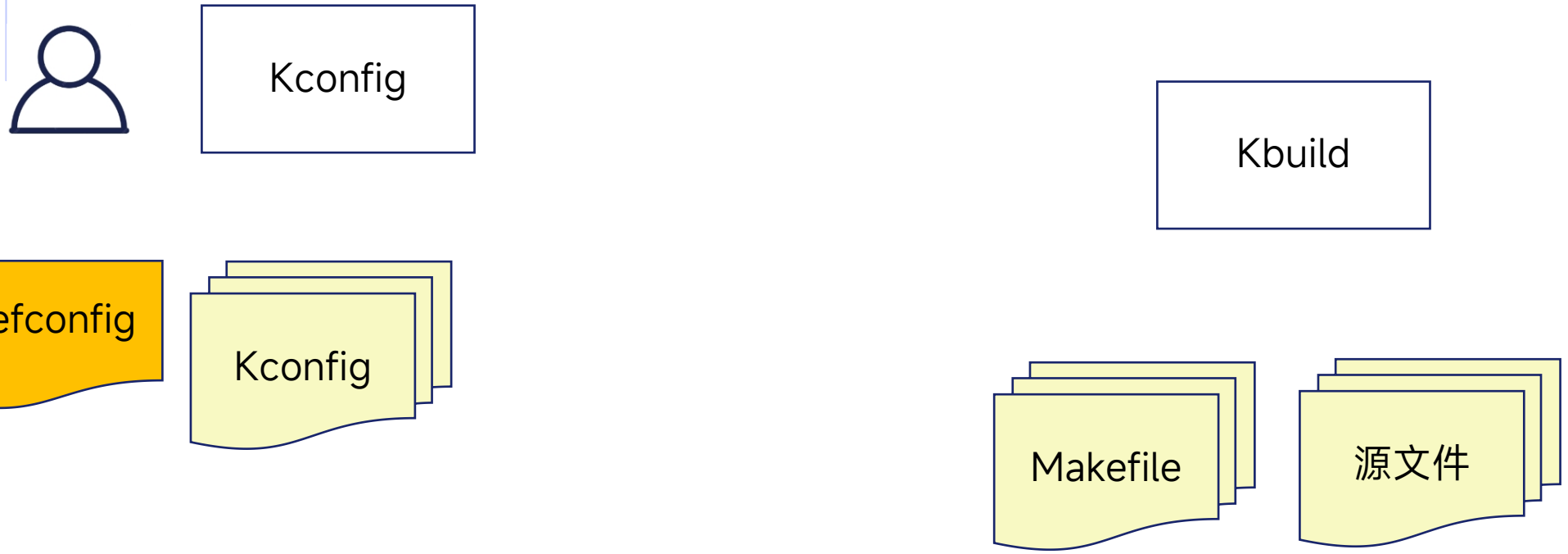


完整的内核构建流程

```
$ make menuconfig
$ make
```

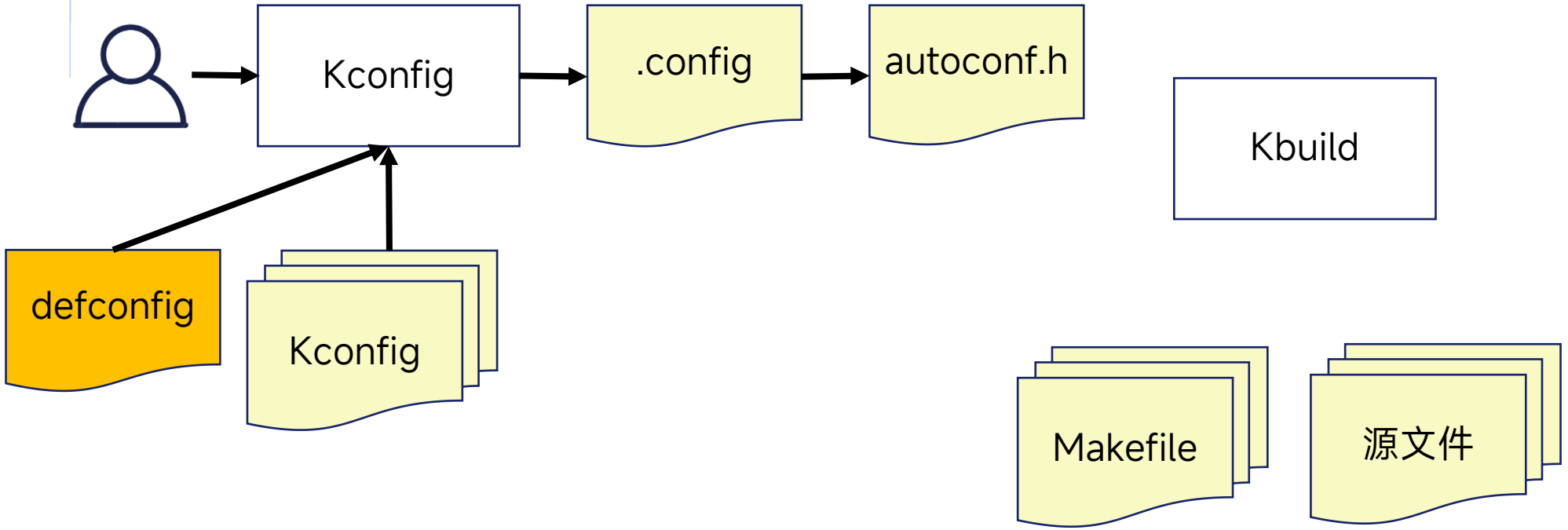


完整的内核构建流程



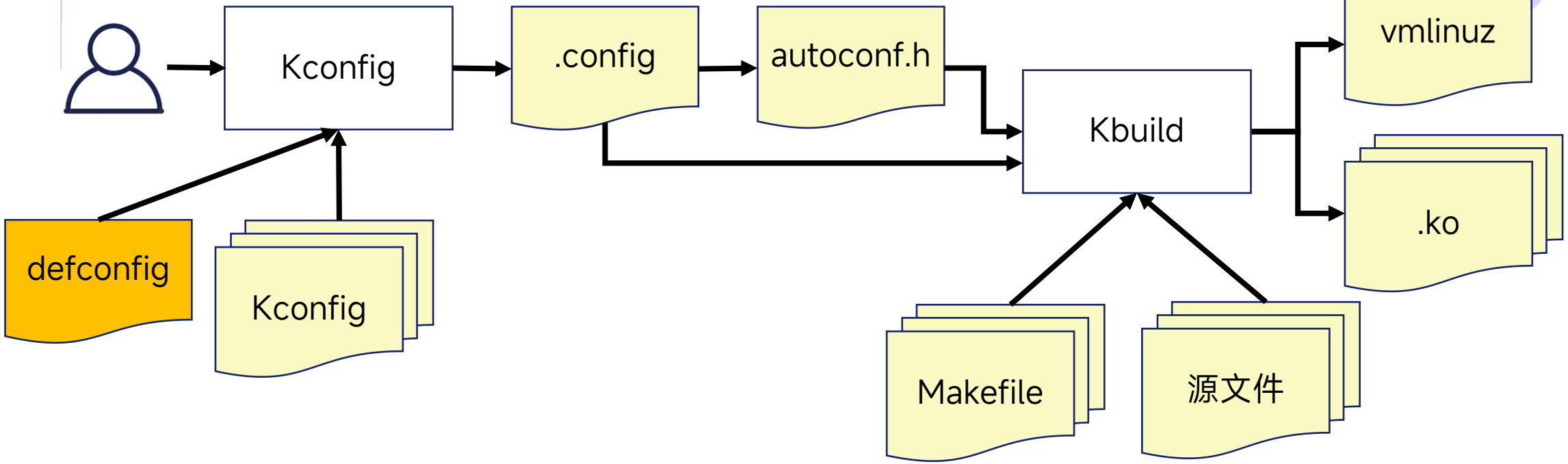
完整的内核构建流程

```
$ make defconfig
```

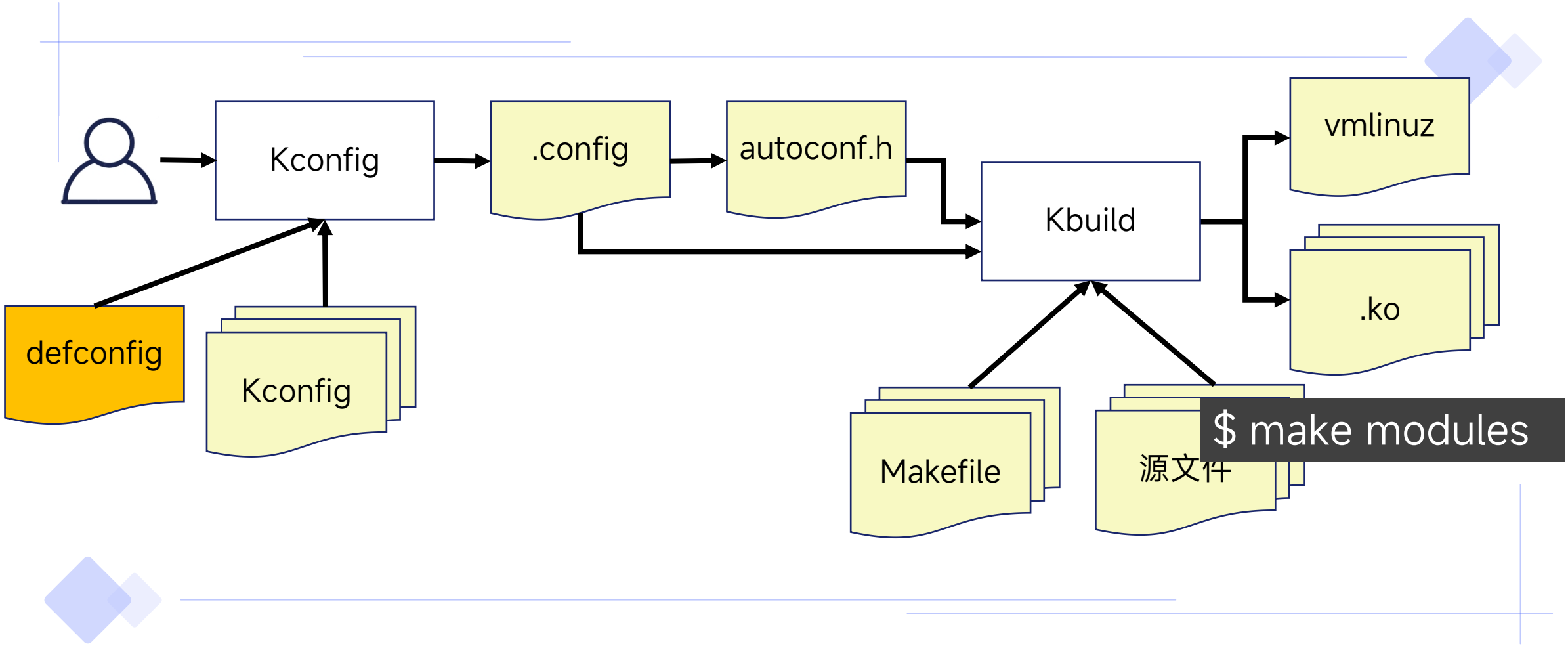


完整的内核构建流程

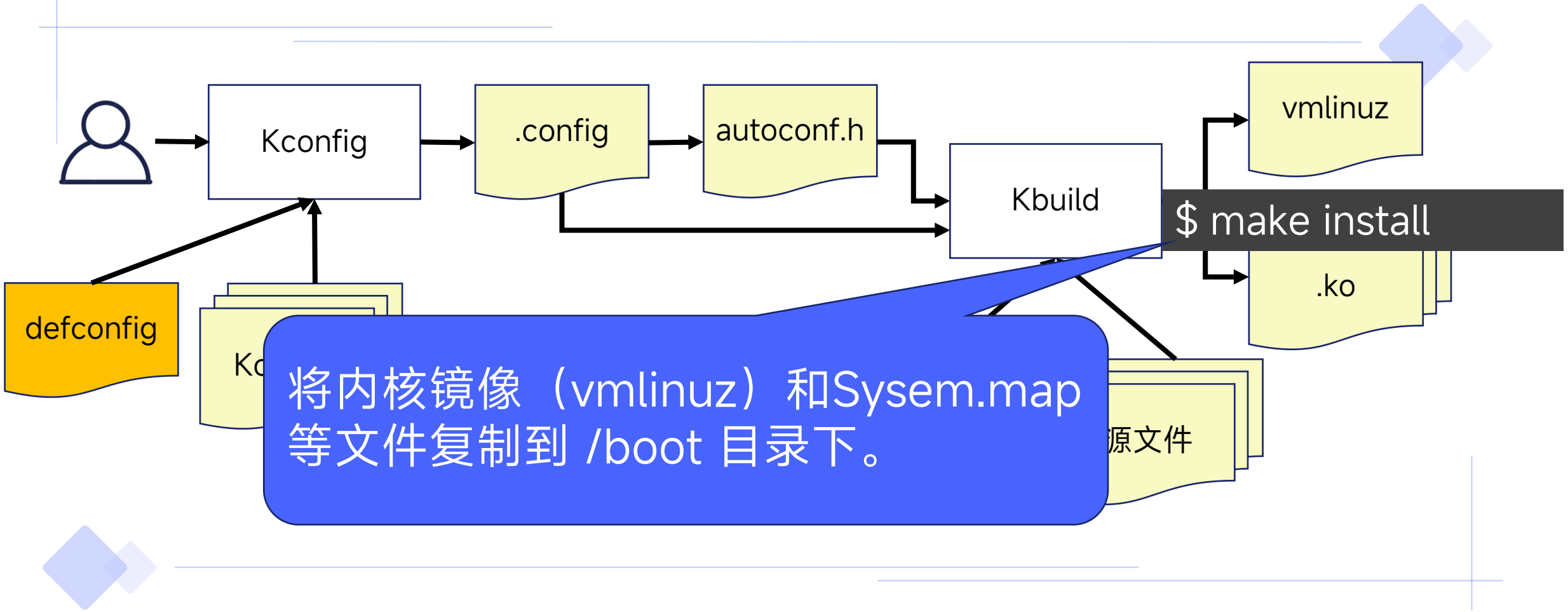
```
$ make defconfig
$ make
```



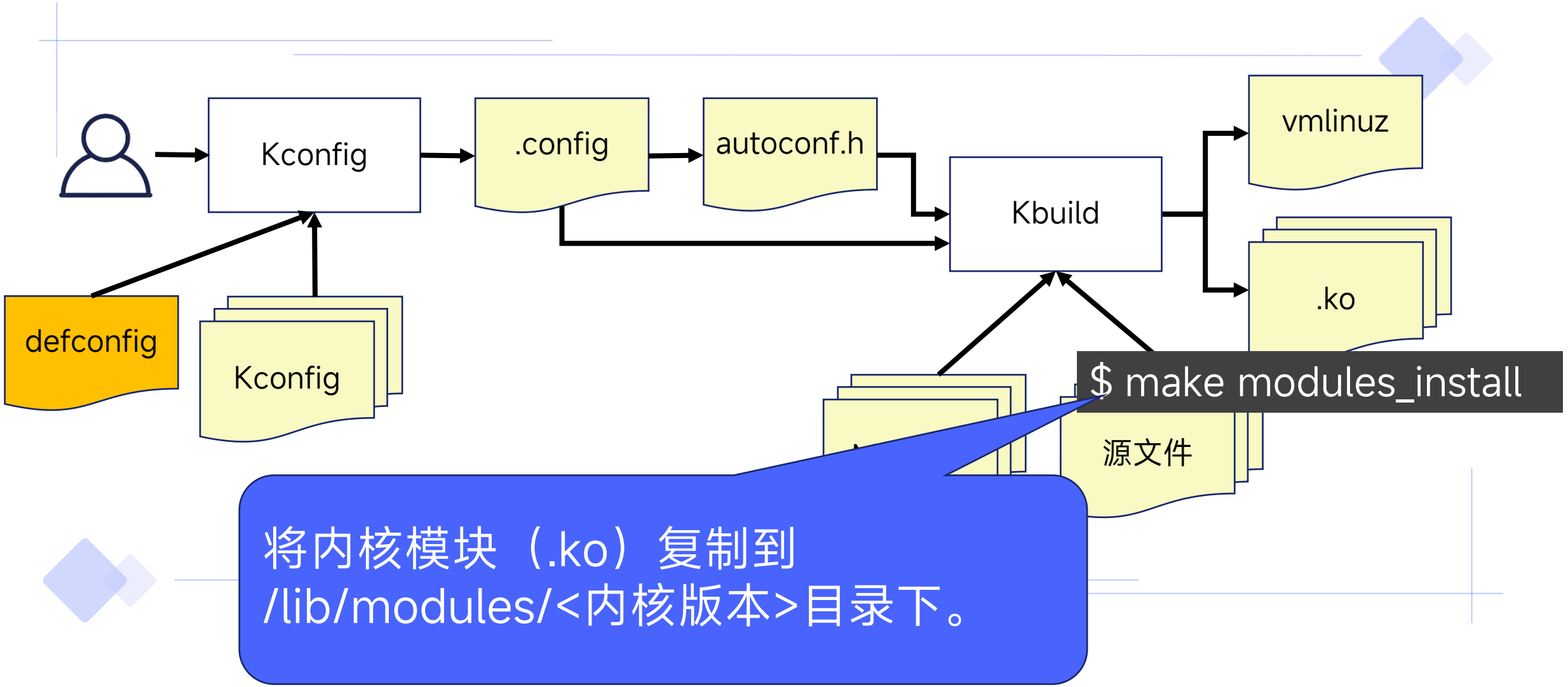
完整的内核构建流程


Kconfig
.config
autoconf.h
Kbuild
vmlinuz
.ko
defconfig
Kconfig
Makefile
源文件
$ make modules

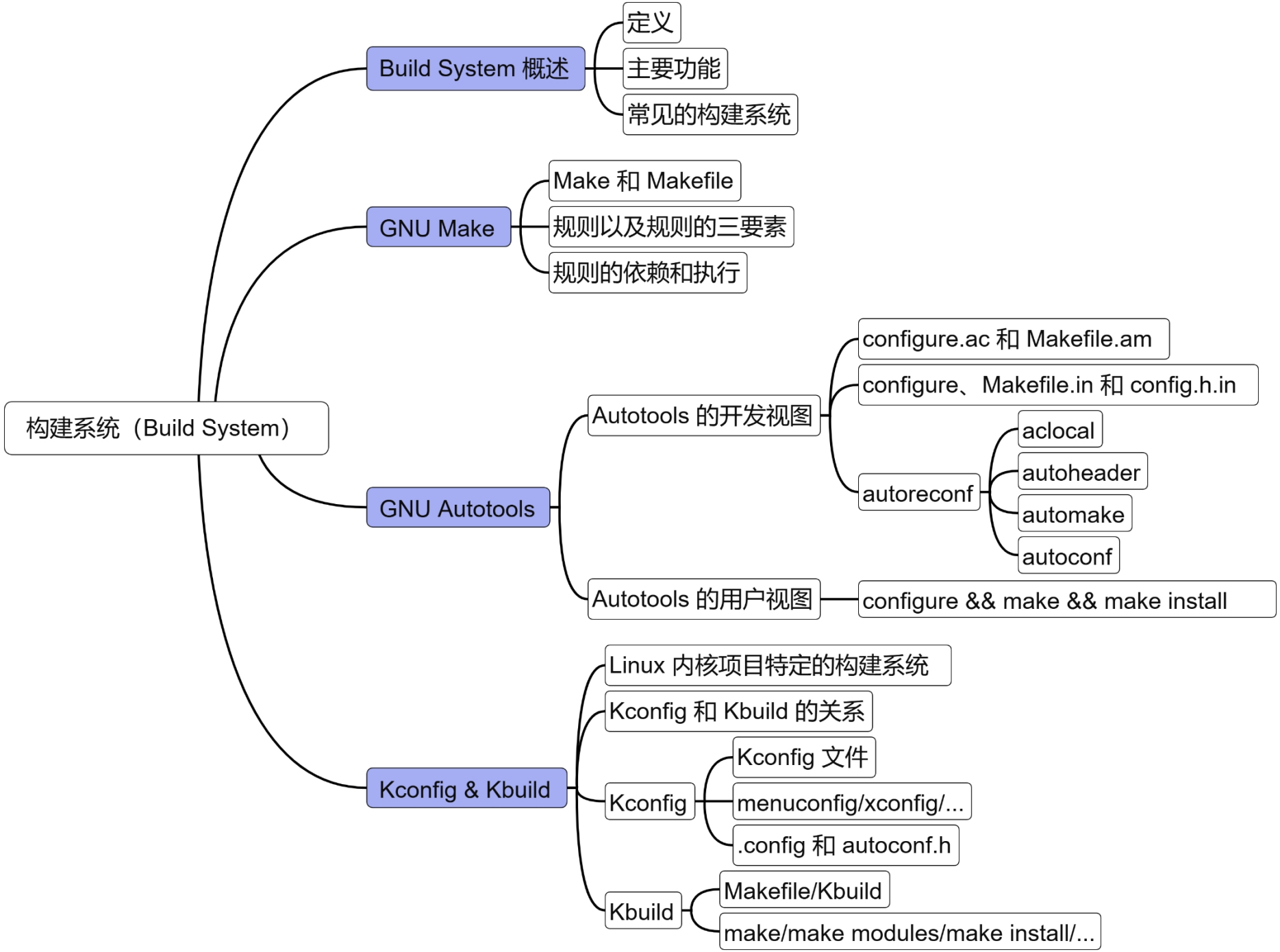
完整的内核构建流程


Kconfig
.config
autoconf.h
Kbuild
vmlinuz
$ make install
.ko
defconfig
源文件
将内核镜像（vmlinuz）和Sysem.map
等文件复制到 /boot 目录下。

完整的内核构建流程


Kconfig
.config
autoconf.h
Kbuild
vmlinuz
.ko
defconfig
Kconfig
源文件
$ make modules_install
将内核模块（.ko）复制到
/lib/modules/<内核版本>目录下。

本章总结


构建系统（Build System）
Build System 概述
定义
主要功能
常见的构建系统
GNU Make
Make 和 Makefile
规则以及规则的三要素
规则的依赖和执行
GNU Autotools
Autotools 的开发视图
configure.ac 和 Makefile.am
configure、Makefile.in 和 config.h.in
autoreconf
aclocal
autoheader
automake
autoconf
Autotools 的用户视图
configure && make && make install
Kconfig & Kbuild
Linux 内核项目特定的构建系统
Kconfig 和 Kbuild 的关系
Kconfig
Kconfig 文件
menuconfig/xconfig/...
.config 和 autoconf.h
Kbuild
Makefile/Kbuild
make/make modules/make install/...

谢谢