crossdev-stages

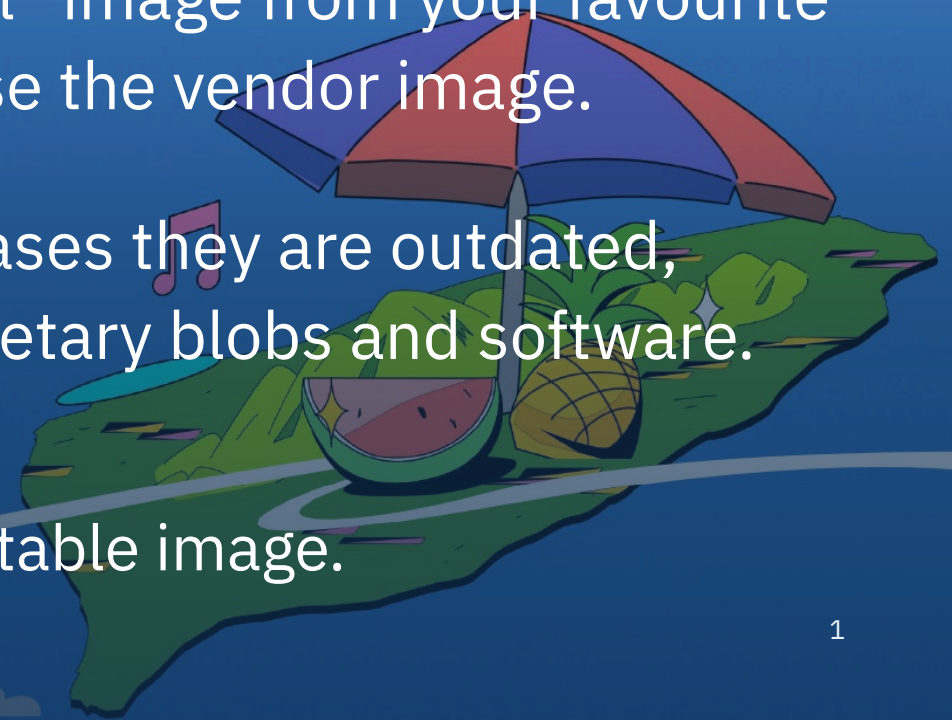
Build Linux images for your RISC-V / ARM board with only a config file

Sungjoon Moon | 文誠浚 | 문성준

COSCUP 2026 | August 8, 2026

You got a new board, now what?

- You need a bootable image.
- You cannot just install the “universal” image from your favourite GNU/Linux distro website. So you use the vendor image.
- Sometimes they are fine. In many cases they are outdated, never upstreamed, and full of proprietary blobs and software.



We need a better bootable image.

Other option is building it yourself.

Build the image from sources you pick yourself.

Then you can try upstream for yourself:
a mainline kernel, upstream U-Boot, a current userland.

We tried to solve this with Gentoo.

But seriously, Gentoo?

What is Gentoo?

Gentoo builds every package from source.

A USE flag turns one feature on or off. Global, or for one package.

```
USE="wayland -X"      # build wayland support, drop X support
```

You can even set the compiler flags: `CFLAGS`, `CXXFLAGS`, `LDFLAGS`.

All of this goes through portage, the package manager.

How Gentoo configures the system

`/etc/portage/make.conf` is the main Portage configuration file.

What you set there applies to every package you emerge.

```
CFLAGS="-O3 -march=rv64gcv_zvl128b -pipe"     # k230
CXXFLAGS="${CFLAGS}"
USE="-kde -qt6 wayland"
```

`/etc/portage/package.*` then refines it per package:

`package.use` `package.env` `package.accept_keywords`

`package.mask` `package.license`

Gentoo also ships crossdev, which installs cross-compilers.

What is crossdev?

crossdev generates a cross-compiler environment for Gentoo.
It is a small shell wrapper around `emerge`.

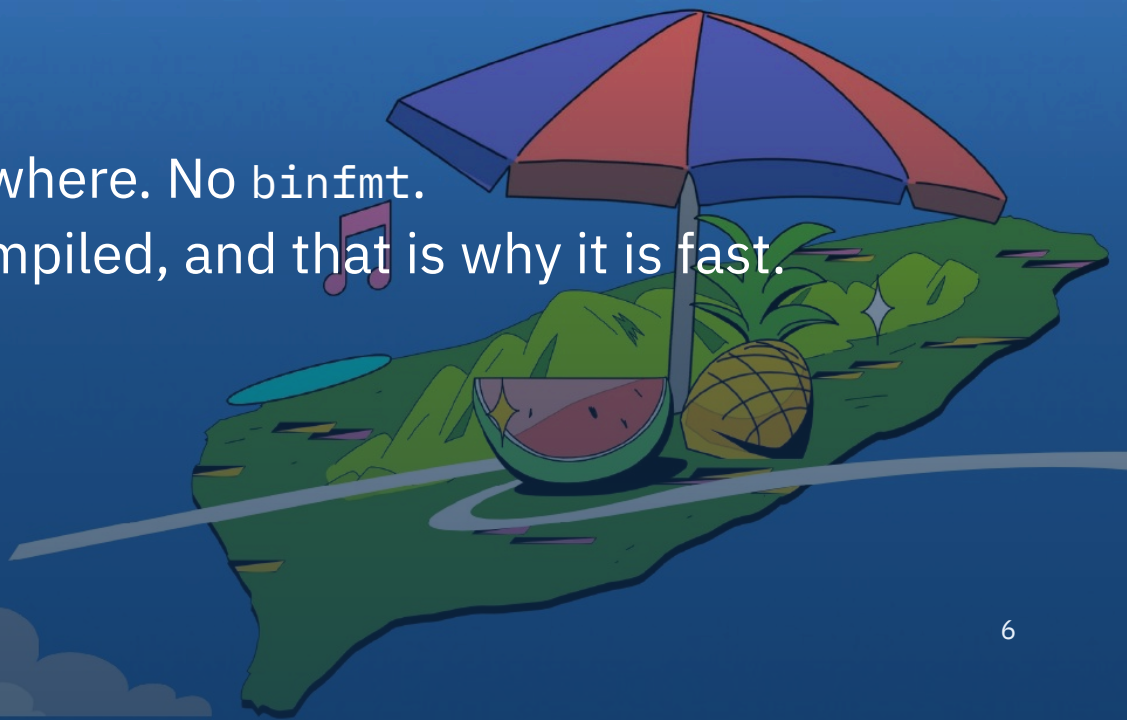
crossdev breaks the circle by building them in stages:

`binutils` → bare `gcc`, no libc → `linux-headers` → `glibc` → full `gcc`

We do not need QEMU

A real toolchain, with `CBUILD`, `CHOST` and `CTARGET`
set to whatever you want.

No QEMU anywhere. No `binfmt`.
Everything is really cross-compiled, and that is why it is fast.



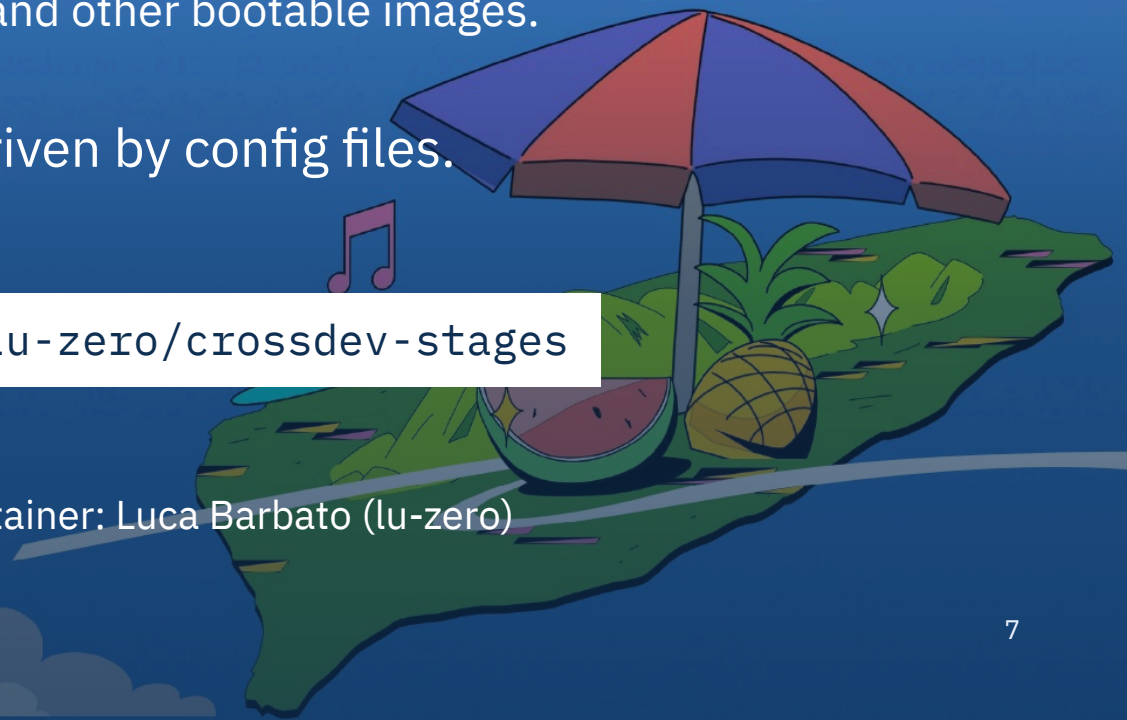
What is crossdev-stages?

crossdev + portage + hakoniwa, packed with Rust

A containerized Gentoo host, with a cross-compiler toolchain,
that produces Gentoo and other bootable images.

And all of it is driven by config files.

`https://github.com/lu-zero/crossdev-stages`



Original author and maintainer: Luca Barbato (lu-zero)

Two layers

Layer 1 A Gentoo host, in a sandbox.

It gives you the `crossdev` toolchain.

Layer 2 The rootfs you want.

For now that is Gentoo, but nothing stops Debian, Fedora, whatever.

Your CFLAGS, your USE flags, vendor or upstream sources.

Then we do the part Gentoo does not do:

kernel, U-Boot, firmware, partition layout,

and we pack all of it into a bootable image.

hakoniwa

hakoniwa (箱庭) is hako (箱, box) plus niwa (庭, garden).

A miniature garden, kept in a box.

It gives you unprivileged Linux user namespaces, as a Rust library.

- No daemon, no image, no registry. It is a syscall.
- So the whole build runs without `sudo`.

Why not bubblewrap? It is a CLI only. We wanted an API.

Do you need Gentoo for running crossdev-stages?

Do you need Gentoo for running crossdev-stages?

No.

Do you need Gentoo for running crossdev-stages?

No.

Any Linux that can run hakoniwa works.
It only has to support unprivileged user namespaces.

A board is a directory

`boards/<board_name>/`

boards/k230/board.conf

```
BOARD_NAME="k230"
BOARD_ARCH="riscv64"
BOARD_CFLAGS="-O3 -march=rv64gcv_zvl128b -pipe"     # <- the whole point
CROSS_COMPILE="riscv64-unknown-linux-gnu-"

OPENSBI_REPO="https://github.com/k230-linux/opensbi.git"
OPENSBI_FW_TYPE="payload"
OPENSBI_MAKE_FLAGS="FW_PAYLOAD_PATH=/build/linux/arch/riscv/boot/Image"
U_BOOT_REPO="https://github.com/k230-linux/k230-vendor-uboot.git"
KERNEL_REPO="https://github.com/k230-linux/k230-linux-kernel.git"
KERNEL_TAG="hdmi"                                   # <- a vendor fork, not upstream

BOOT_PIPELINE=("opensbi" "uboot")
BUILD_STEPS=(deps checkout kernel bootloader assemble pack)

# vector instructions broken in libgcrypt on this SoC
WORKAROUND_PKGS=("dev-libs/libgcrypt")
WORKAROUND_CFLAGS=("-O3 -march=rv64gc -pipe")       # <- one package, safe flags
```

ARM or RISC-V, same file

odroid-m2 | aarch64

```
BOARD_ARCH="aarch64"
KERNEL_ARCH="arm64"

KERNEL_REPO="https://git.kernel.org/pub/scm/\
linux/kernel/git/torvalds/linux.git"
KERNEL_TAG="v7.0"

TFA_REPO="https://github.com/ARM-software/\
arm-trusted-firmware.git"
TFA_PLAT="rk3588"
RKBIN_REPO="https://github.com/rockchip-linux/\
rkbin.git"

BOOT_PIPELINE=("rkbin" "tfa" "uboot")
```

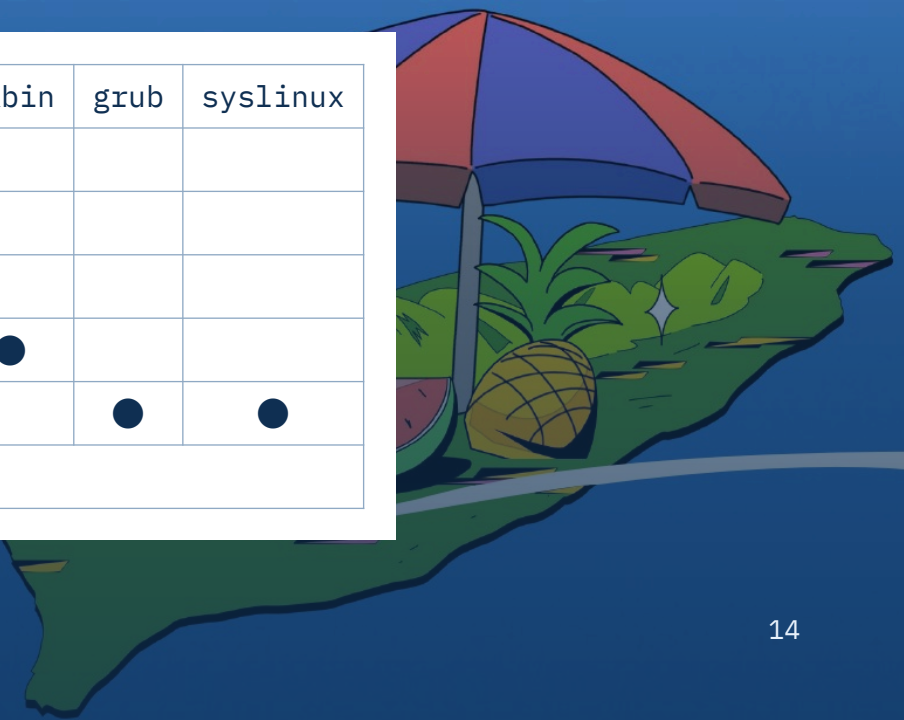
k230 | riscv64

```
BOARD_ARCH="riscv64"
KERNEL_ARCH="riscv"

KERNEL_REPO="https://github.com/k230-linux/\
k230-linux-kernel.git"
KERNEL_TAG="hdmi"

OPENSBI_REPO="https://github.com/k230-linux/\
opensbi.git"
OPENSBI_FW_TYPE="payload"


BOOT_PIPELINE=("opensbi" "uboot")
```

BOOT_PIPELINE

```
BOOT_PIPELINE=("rkbin" "tfa" "uboot")
```

board	opensbi	uboot	tfa	rkbin	grub	syslinux
k1, k3, ky-x1	●	●				
k230	●	●				
blackhole	●					
odroid-m2		●	●	●		
pentium-mmx					●	●
odroid-m1, m1s	override-bootloader.sh					



How an image gets built

step	what it does
`deps`	cross-emerge the board's package list
`checkout`	clone kernel, OpenSBI, U-Boot, firmware
`kernel`	build Linux and modules
`bootloader`	run BOOT_PIPELINE, stage by stage
`assemble`	initramfs, lay out the root tree
`pack`	genimage writes the disk image

```
$ crossdev-stages --dry-run image build --board k230
Board:   k230      Arch: riscv64      CFLAGS: -O3 -march=rv64gcv_zvl128b -pipe
Steps:   deps checkout kernel bootloader assemble pack
```

When config is not enough

Override any step, or hook a shell script before or after it.

```
override-{step}.sh    pre-{step}.sh    post-{step}.sh

boards/k3/override-kernel.sh
boards/odroid-m1/override-bootloader.sh
boards/k230/post-assemble.sh
```

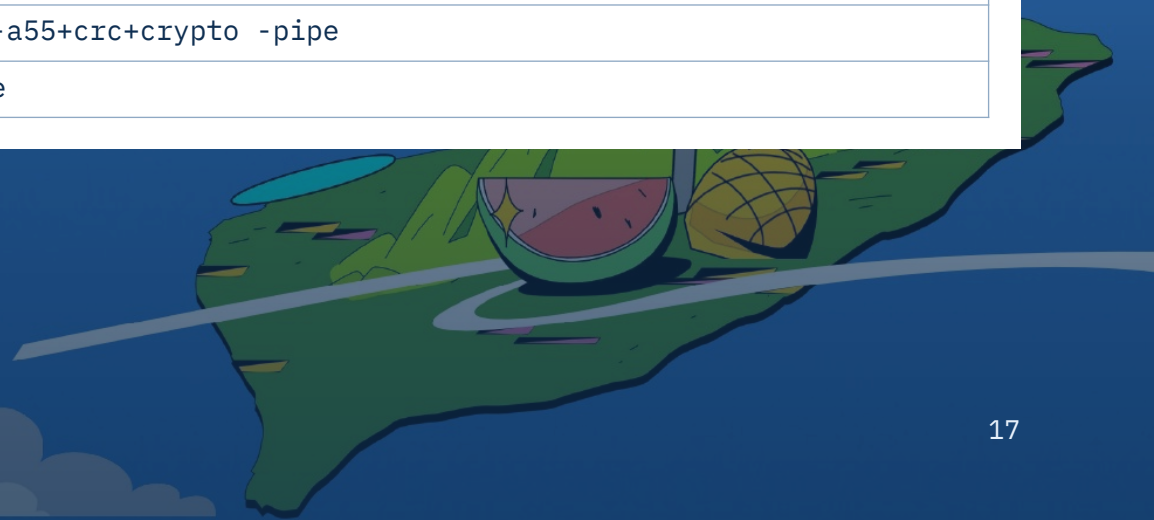
The rootfs is nearly the same on every board, firmware aside.

The bootloader is different on every single board.

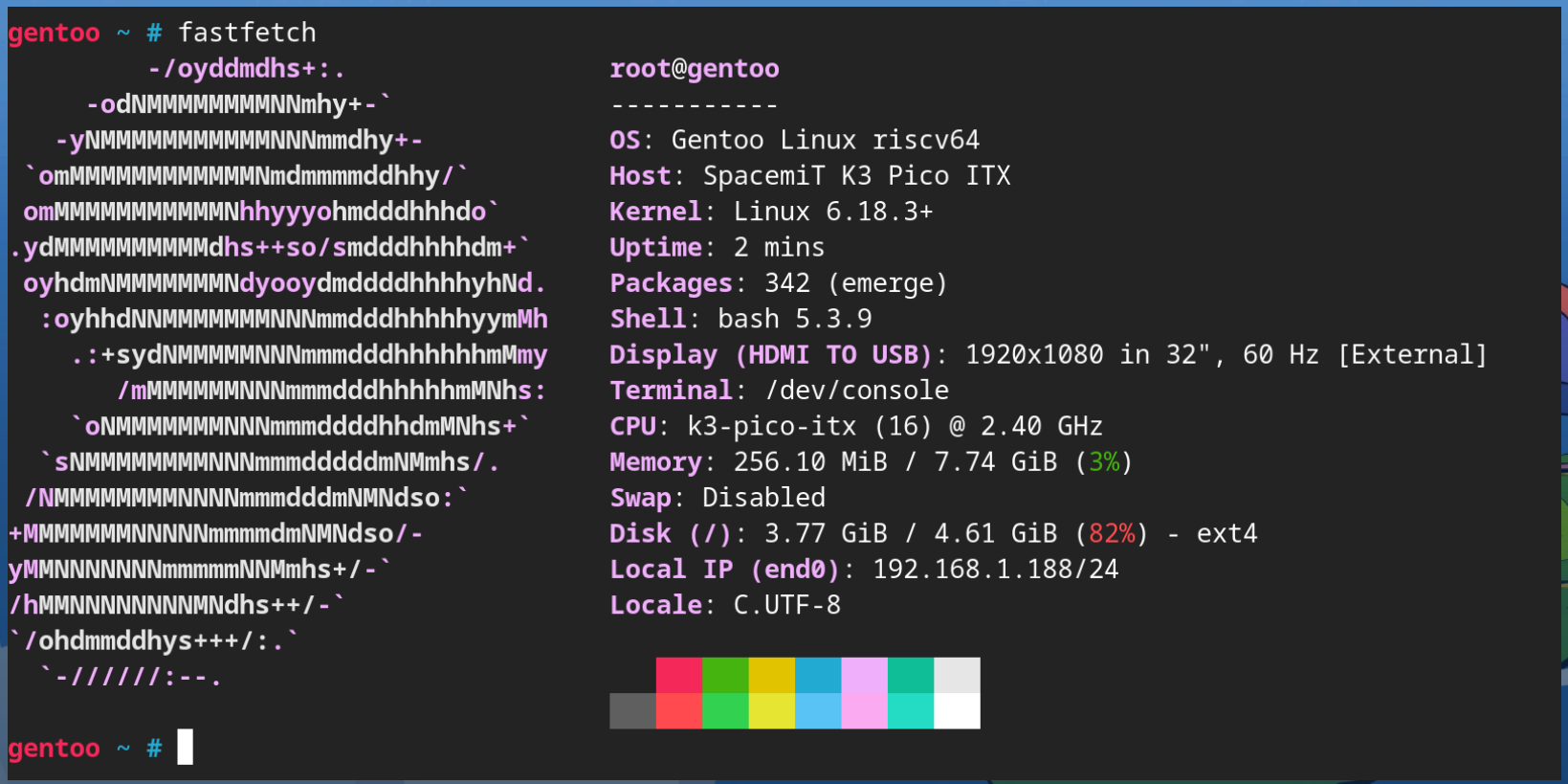
The design is borrowed from ebuilds:
`src_configure`, `src_compile`, `src_install`, each one overridable.

Boards it builds today

board	arch	BOARD_CFLAGS
k1, k1-upstream	riscv64	`-O3 -march=rv64gcv_zvl256b -pipe`
ky-x1	riscv64	`-O3 -march=rv64gcv_zvl256b -pipe`
k230	riscv64	`-O3 -march=rv64gcv_zvl128b -pipe`
k3	riscv64	`-O3 -march=rva23u64 -pipe`
blackhole	riscv64	`-O3 -march=rv64gcv_zvl512b -pipe`
odroid-m1, m1s	aarch64	`-O3 -mcpu=cortex-a55+crc+crypto -pipe`
odroid-m2	aarch64	`-O3 -mcpu=cortex-a76.cortex-a55+crc+crypto -pipe`
pentium-mmx	i586	`-march=pentium-mmx -O2 -pipe`



And it boots


gentoo ~ # fastfetch
root@gentoo
OS: Gentoo Linux riscv64
Host: SpacemiT K3 Pico ITX
Kernel: Linux 6.18.3+
Uptime: 2 mins
Packages: 342 (emerge)
Shell: bash 5.3.9
Display (HDMI TO USB): 1920x1080 in 32", 60 Hz [External]
Terminal: /dev/console
CPU: k3-pico-itx (16) @ 2.40 GHz
Memory: 256.10 MiB / 7.74 GiB (3%)
Swap: Disabled
Disk (/): 3.77 GiB / 4.61 GiB (82%) - ext4
Local IP (end0): 192.168.1.188/24
Locale: C.UTF-8
gentoo ~ #

Building everything, every time?

No. Portage makes binary packages, and reuses them.

Once you support many boards, you end up with CFLAGS that cannot be shared.

But the same build gets written in different ways.
One extra space. The same flags in another order.

We need to know when two sets of flags mean the same thing.

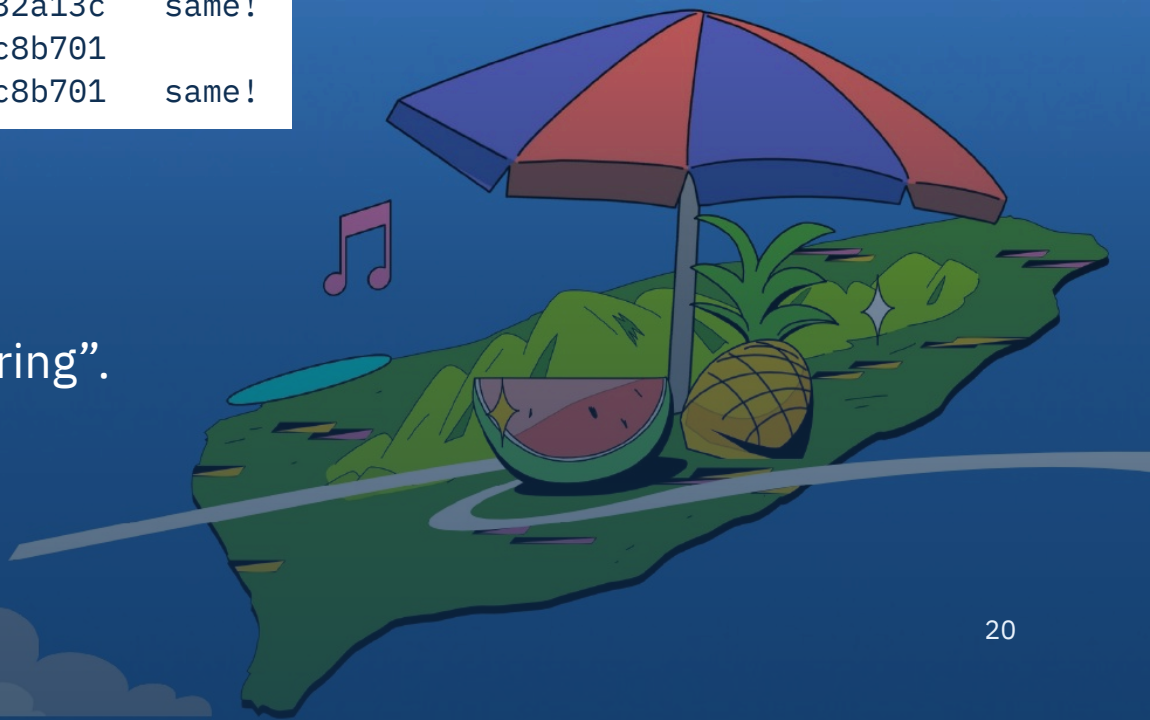
sokgi

So we made sokgi (솎기, “thinning”).

```
-O3 -march=rv64gcv_zvl256b -pipe      5bff2670cc32a13c
-pipe -march=rv64gcv_zvl256b -O3      5bff2670cc32a13c   same!
-O3 -O2 -march=cortex-a55             990248d518c8b701
-O2 -march=cortex-a55                 990248d518c8b701   same!
```

It canonicalizes the flags, then hashes them.

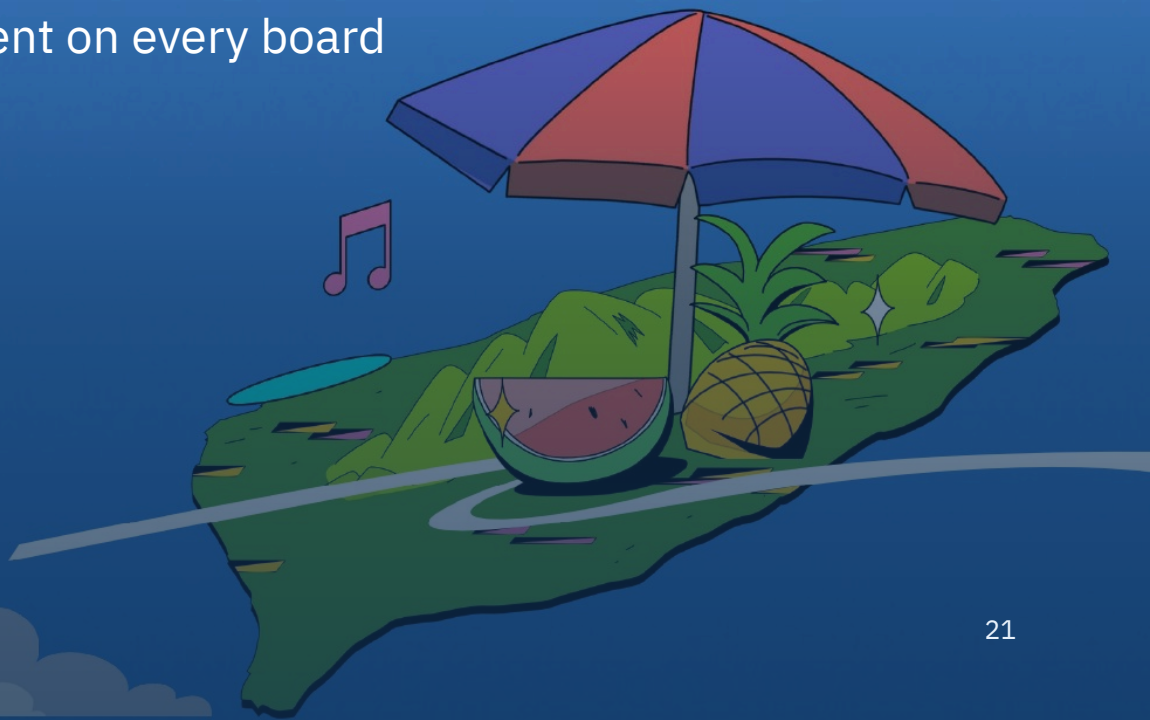
A key that means “same build”, not “same string”.



So why crossdev-stages?

Because every new board means four things, by hand:

- a cross toolchain for that CPU
- a userland actually built for that CPU
- a kernel, a bootloader and firmware, different on every board
- a partition layout, and a disk image



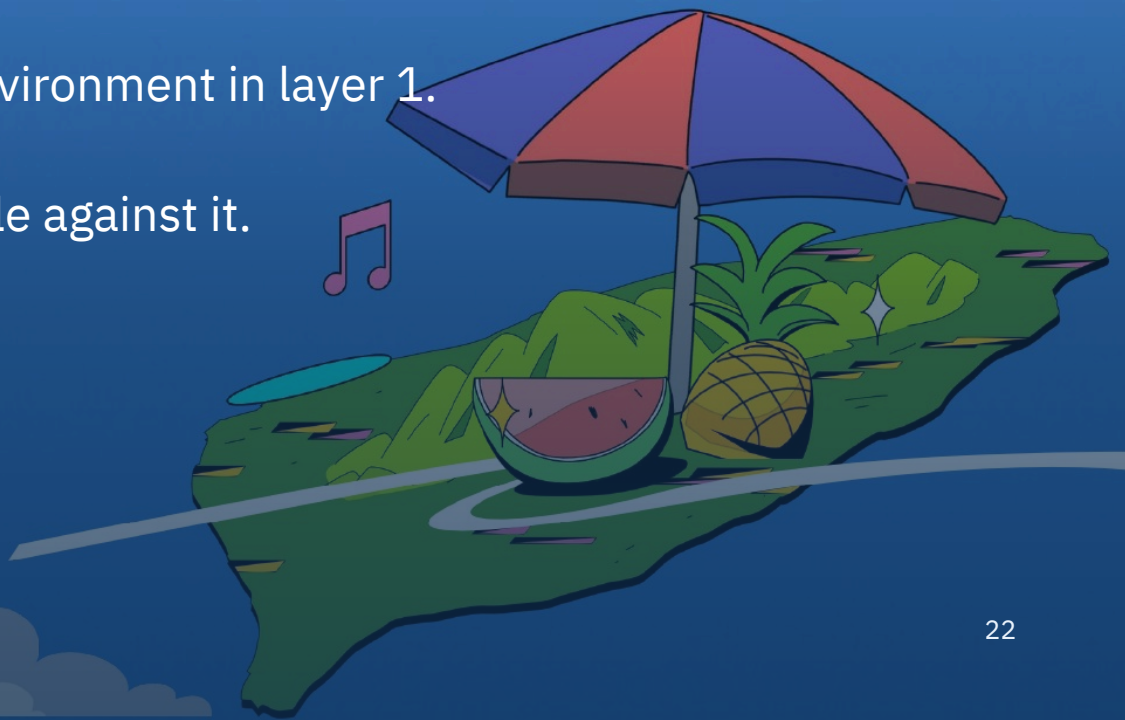
Future plan: crossdev-stages as a library

cross-rs is a cargo wrapper, plus one vendor image per target.

Needs Docker or Podman. Fixed Ubuntu baseline, fixed glibc.

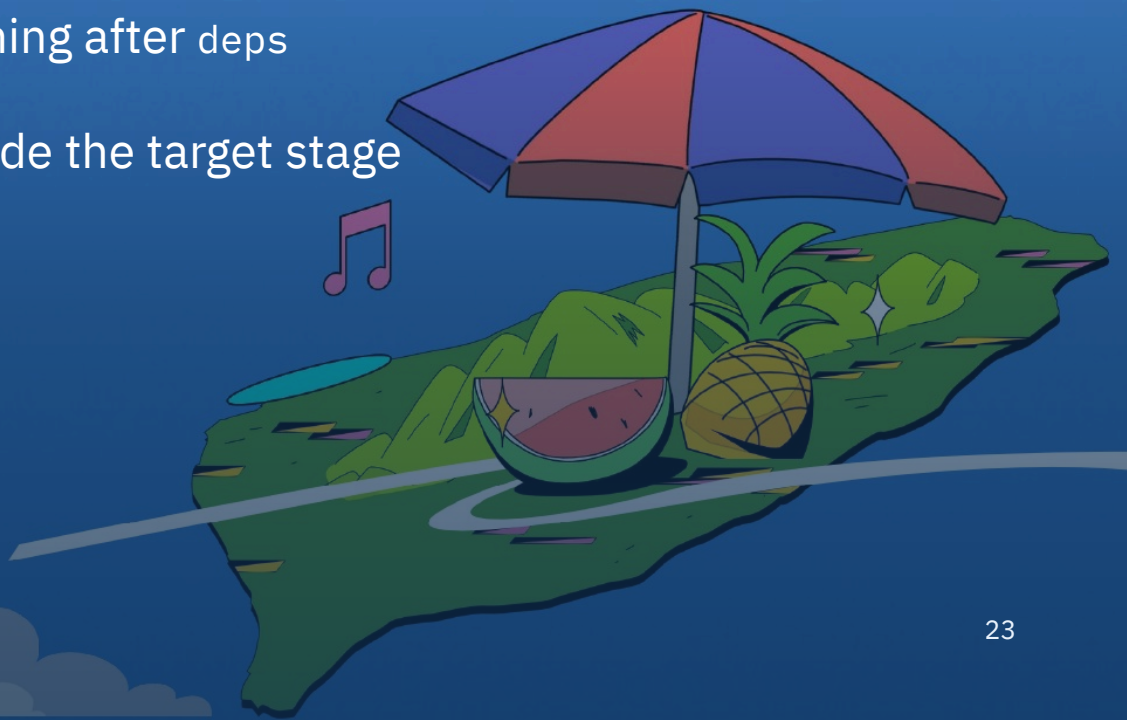
We already give you an excellent `crossdev` environment in layer 1.

As a library, Rust projects could cross-compile against it.



Also planned

- `clang` as a second toolchain
- A per-hash binpkg store, so boards with the same sokgi hash share builds
- `image build --parallel`: parallelize everything after `deps`
- `crossdev-stages enter <board>`, a shell inside the target stage
- Weekly CI images for every board



Thank You!

Questions?

https://github.com/lu-zero/crossdev-stages

Sungjoon Moon | 文誠浚 | 문성준

github.com/OctopusET | sumoon@seoulsaram.org

souk4711/hakoniwa | OctopusET/sokgi