U-Boot Standard Boot and What's Next

Open Source Summit 2025
Denver, Colorado

Simon Glass
Fellow Software Engineer

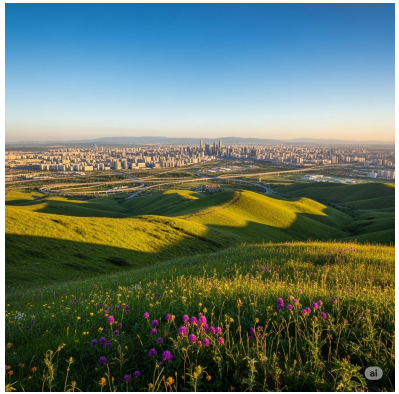
Agenda

- The state of play
- Standard Boot
- Relevant recent work
- Demo
- What's next?

The good news

- Mainline U-Boot boots 1300 boards
  - “Universal Bootloader”
  - Supports QEMU for cloud
  - Can run as an EFI app on x86 and ARM
  - Has a GPL-licensed EFI-loader stack
  - Uses Linux devicetree; supports complex SoCs
- Arm SystemReady can boot many distros
  - Uses EFI loader
  - Adoption is growing
- Code quality
  - Has 1000 tests and Gitlab CI
  - Coverity tracking is in place
  - Overall code has improved significantly from 10 years ago

Everything looks good!



But the environment is ugly

```
mmc_boot=if mmc dev ${devnum}; then devtype=mmc; run scan_dev_for_boot_part; fi
boot_prefixes=/ /boot/
boot_scripts=boot.scr.uimg boot.scr
boot_script_dhcp=boot.scr.uimg
boot_targets=mmc1 mmc0 dhcp
boot_syslinux_conf=extlinux/extlinux.conf
boot_extlinux=sysboot ${devtype} ${devnum}:${distro_bootpart} any ${scriptaddr}
${prefix}${boot_syslinux_conf}
scan_dev_for_extlinux=if test -e ${devtype} ${devnum}:${distro_bootpart} ${prefix}${boot_syslinux_conf};
then echo Found ${prefix}${boot_syslinux_conf}; run boot_extlinux; echo EXTLINUX FAILED: continuing...;
fi
boot_a_script=load ${devtype} ${devnum}:${distro_bootpart} ${scriptaddr} ${prefix}${script}; source
${scriptaddr}
scan_dev_for_scripts=for script in ${boot_scripts}; do if test -e ${devtype} ${devnum}:${distro_bootpart}
${prefix}${script}; then echo Found U-Boot script ${prefix}${script}; run boot_a_script; echo SCRIPT
FAILED: continuing...; fi; done
scan_dev_for_boot=echo Scanning ${devtype} ${devnum}:${distro_bootpart}...; for prefix in
${boot_prefixes}; do run scan_dev_for_extlinux; run scan_dev_for_scripts; done;
scan_dev_for_boot_part=part list ${devtype} ${devnum} -bootable devplist; env exists devplist || setenv
devplist 1; for distro_bootpart in ${devplist}; do if fstype ${devtype} ${devnum}:${distro_bootpart}
bootfstype; then part uuid ${devtype} ${devnum}:${distro_bootpart} distro_bootpart_uuid ; run
scan_dev_for_boot; fi; done; setenv devplist
```

(1600 bytes of boilerplate distro-boot scripts)

EFI is really complicated

- Large amount of software
  - Over 120K of code to interface with GRUB
  - 2400-page spec
  - Expanding gap between U-Boot boot_jump_linux() and Linux _start
- Bootmgr doesn’t do lazy initialization
  - Boot can be quite slow
  - Hard to control within U-Boot
- Devicetree is challenging
  - Putting it in firmware seems more ‘scalable’
  - But in practice new kernels can break
  - Devicetree fixups in GRUB are challenging
  - Secure boot needs devicetree stopgaps

Some other problems

- It is easy to break boards
  - Sometimes changes in generic code cause breakage
  - Bug reports often come long after the patch lands
  - Tough to debug problems without the right hardware
- Firmware images are complex
  - Lots of components to carefully put together
  - Vendor tools (no source code, no Debian package)
  - Vendor blobs (no source code, manual downloads)
- U-Boot can be quite large
  - More and more features all the time
  - Command line is a good part of that

Everything looks bad!

What is my role?

- Vision for better boot firmware
- Open Source
- Reliable
- Deterministic
- Testable

- Let’s look at standard boot…
  - …and a few other things that might help

Standard Boot

- Boot automatically
  - Distros, ChromeOS, Android, etc.
  - Systematic search for Operating Systems
  - Streamlines the boot
- No scripts
  - New 'bootflow' command
  - Can run without command-line interface

Concepts

- boot**dev** (Boot Device)
  - Thin software layer on top of media devices
- boot**meth** (Boot Method)
  - Knows how to locate extlinux, ChomeOS, EFI, Android…
- boot**flow** (Boot Flow)
  - Contains all the details needed to boot the OS

Bootdev

- **Where** to find bootflows
  - Abstraction layer over a physical media device
  - Typically this is a block device
  - Networks and SPI flash are supported too
  - One API call: get_bootflow(), called for every bootmeth
- Bootdev 'hunters' complete the picture
  - E.g. a USB hunter can scan a USB bus
  - Produce new bootdevs dynamically
  - Hunters only run when needed (lazy init for speed!)

```
mmc            2  [ + ]   mmc_sandbox          |-- mmc2
blk            0  [   ]   mmc_blk              |   |-- mmc2.blk
bootdev        9  [   ]   mmc_bootdev          |   `-- mmc2.bootdev
mmc            1  [ + ]   mmc_sandbox          |-- mmc1
blk            1  [   ]   mmc_blk              |   |-- mmc1.blk
bootdev       10  [   ]   mmc_bootdev          |   `-- mmc1.bootdev
mmc            0  [ + ]   mmc_sandbox          |-- mmc0
```

Bootmeth

- **How** to find bootflows
  - Scan a bootdev to find bootable instructions
  - e.g. distro bootmeth searches for extlinux.conf
  - ChromeOS bootmeth searches for kernel/root partitions GUIDs
- Simple API:
  - read_bootflow() - read a bootflow from bootdev if possible
  - read_file() - read an associated file (e.g. devicetree, initrd)
  - read_all() - read all files into memory ready for boot
  - boot() - boot it

Bootflow

- Describes the OS which can be booted
  - Includes references to the bootmeth and bootdev
  - Provides metadata such as OS name, logo
- List of images to be loaded
- Links to the bootdev and bootmeth
- Also supports kernel arguments

```
=> bootflow list
Showing all bootflows
Seq  Method       State   Uclass    Part  Name                        Filename
---  -----------  ------  --------  ----  ------------------------  ----------------
  0  extlinux     ready   mmc          1  mmc1.bootdev.part_1       /extlinux/extlinux.conf
  1  efi          ready   usb_mass_    1  hub1.p2.usb_mass_storage. /EFI/BOOT/BOOTSBOX.EFI
  2  extlinux     ready   usb_mass_    1  hub1.p4.usb_mass_storage. /extlinux/extlinux.conf
---  -----------  ------  --------  ----  ------------------------  ----------------
=>
```

Iteration

- Simple but powerful API:
  - bootflow_scan_first() - returns first OS
  - bootflow_scan_next() - returns the next
- By default, fast devices are scanned first
- It can boot the first OS it finds
- bootmeths and bootdevs support custom ordering

Put it all together

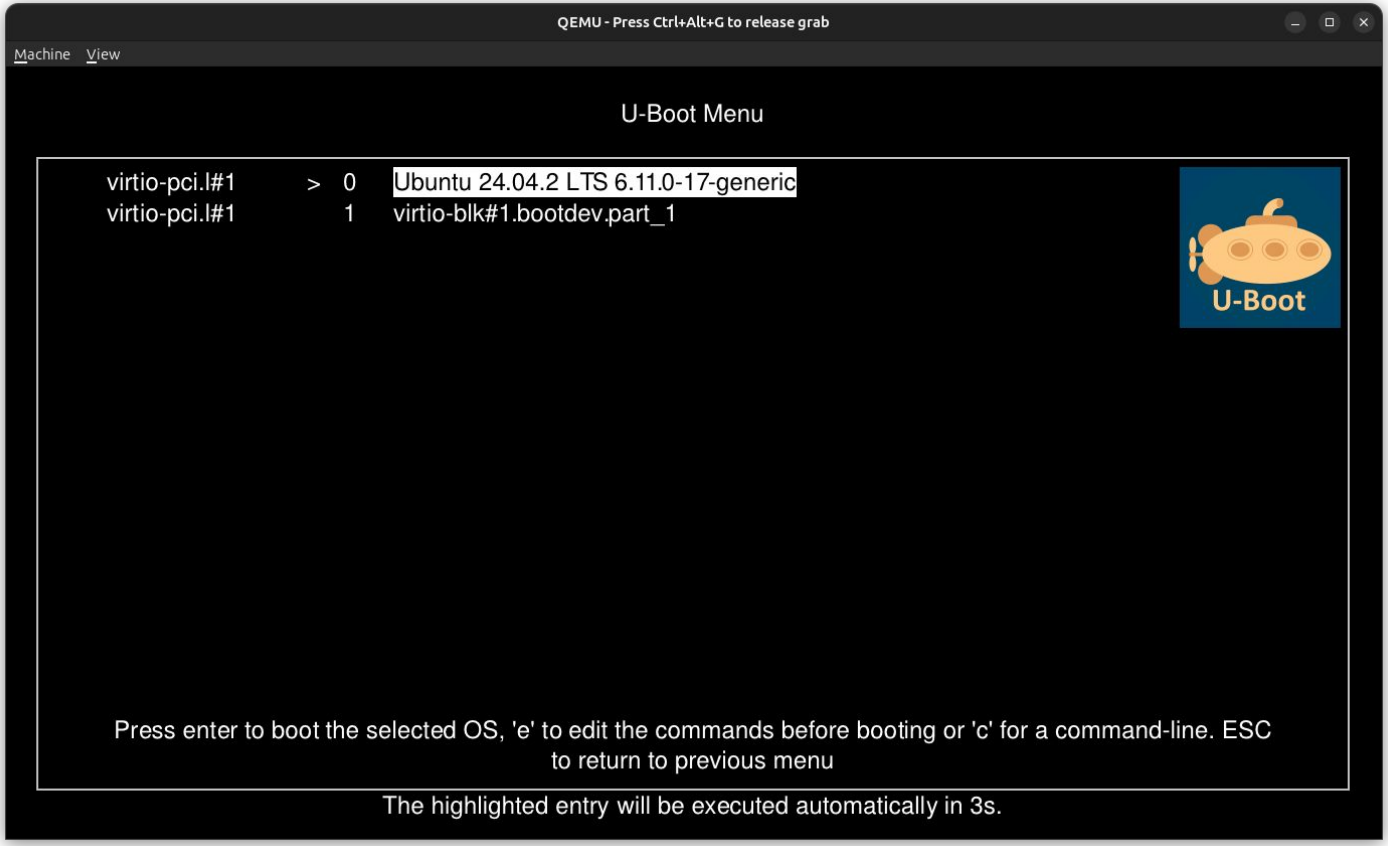
- 'bootflow' command
  - bootflow scan - scan for bootflows
  - bootflow list - list bootflows
  - bootflow info - show information

```
=> bootfl select 0
=> bootflow info
Name:       mmc1.bootdev.part_1
Device:     mmc1.bootdev
Block dev:  mmc1.blk
Method:     extlinux
State:      ready
Partition:  1
Subdir:     (none)
Filename:   /extlinux/extlinux.conf
Buffer:     8e6b8e0
Size:       253 (595 bytes)
OS:         Fedora-Workstation-armhfp-31-1.9
(5.3.7-301.fc31.armv7hl)
Cmdline:    ro
root=UUID=9732b35b-4cd5-458b-9b91-80f7047e0b8a
rhgb quiet LANG=en_US.UTF-8 cma=192MB
cma=256MB
Logo:       (none)
FDT:        <NULL>
Error:      0
=>
```

Read and list images

```
=> bootflow read
1:    Fedora-Workstation-armhfp-31-1.9 (5.3.7-301.fc31.armv7hl)
Retrieving file: /vmlinuz-5.3.7-301.fc31.armv7hl
Retrieving file: /initramfs-5.3.7-301.fc31.armv7hl.img
append: ro root=UUID=9732b35b-4cd5-458b-9b91-80f7047e0b8a rhgb quiet LANG=en_US.UTF-8 cma=192MB
cma=256MB
Retrieving file: /dtb-5.3.7-301.fc31.armv7hl/sandbox.dtb
=> bootstd images
Seq  Bootflow              Type                 At      Size  Filename
---  -------------------  --------------  --------  --------  ----------------
  0  mmc1.bootdev.part_1  extlinux_cfg     8e6b8e0       253  /extlinux/extlinux.conf
  0  mmc1.bootdev.part_1  kernel           1000000       67c  /extlinux/extlinux.conf
  0  mmc1.bootdev.part_1  ramdisk          2000000         7  /extlinux/extlinux.conf
  0  mmc1.bootdev.part_1  flat_dt            c00000        48  /extlinux/extlinux.conf
---  -------------------  --------------  --------  --------  ----------------
(4 images)
=> bootflow cmdline set console ttyO0,115200
=> bootflow boot
** Booting bootflow 'mmc1.bootdev.part_1' with extlinux
## Loading kernel kernel-1 from FIT Image at 01000000 ...
   Using 'conf-1' configuration
   Verifying Hash Integrity ... OK
…
```

Boot menu


QEMU - Press Ctrl+Alt+G to release grab
Machine View
U-Boot Menu
virtio-pci.l#1 > 0 Ubuntu 24.04.2 LTS 6.11.0-17-generic
virtio-pci.l#1 1 virtio-blk#1.bootdev.part_1
U-Boot
Press enter to boot the selected OS, 'e' to edit the commands before booting or 'c' for a command-line. ESC to return to previous menu
The highlighted entry will be executed automatically in 3s.

…and a cleaner environment

no distro scripts!

```
bootcmd=bootflow scan
bootdelay=1
baudrate=115200
ipaddr=192.168.10.2
serverip=192.168.10.1
preboot=test -n ${fdtfile} || setenv fdtfile ${soc}-colibri-${fdt_board}.dtb
netmask=255.255.255.0
loadaddr=0x95c00000
arch=arm
cpu=armv8
board=colibri-imx8x
board_name=colibri-imx8x
vendor=toradex
soc=imx8
m4_0_image=m4_0.bin
loadm4image_0=load mmc ${mmcdev}:${mmcpart} ${loadaddr} ${m4_0_image}
m4boot_0=run loadm4image_0; dcache flush; bootaux ${loadaddr} 0
fdt_addr_r=0x9d400000
kernel_addr_r=0x95c00000
kernel_comp_addr_r=0xb0000000
kernel_comp_size=0x08000000
ramdisk_addr_r=0x9d500000
scriptaddr=0x9d480000
boot_script_dhcp=boot.scr
```

```
Patch:

 configs/colibri-imx8x_defconfig | 2 +-
 include/configs/colibri-imx8x.h | 8 --------
 2 files changed, 1 insertion(+), 9 deletions(-)
```

Also, it has 5000 lines of tests

```
static int bootflow_efi(struct unit_test_state *uts)
{
…
	ut_assertok(run_command("bootflow scan", 0));
…
	ut_assert_nextlinen(
		"  1  efi           ready   usb_mass_    1  hub1.p2.usb_mass_storage. /EFI/BOOT/BOOTSBOX.E
…
	ut_assertok(run_command("bootflow select 1", 0));
..
	ut_asserteq(1, run_command("bootflow boot", 0));
	ut_assert_nextline(
		"** Booting bootflow 'hub1.p2.usb_mass_storage.lun0.bootdev.part_1' with efi");
…
	ut_assert_nextline("Exiting boot services");
…
	/* make sure the bootflow is still present */
	ut_assertnonnull(std->cur_bootflow);
…
	/* the USB block-device should have been removed */
	ut_assertok(uclass_find_device_by_seq(UCLASS_USB, 1, &usb));
	ut_assert(!device_active(usb));

	/* check memory allocations are as expected */
	if (!hdr)
		return 0;
```

Standard boot status

- Documentation at https://docs.u-boot.org/en/latest/develop/bootstd/overview.html
- Migration in progress
  - 277 boards now use CONFIG_BOOTSTD_DEFAULTS
  - (another 125 have been pending for a while)

New and upcoming Features

Drop the command line

- CONFIG_CMDLINE is normally enabled
  - Provides a rich set of powerful commands
  - But it is a large part the size of U-Boot
  - What if we could boot without this?
- Standard boot supports this!
  - qemu-x86 reduces from 920K to 531K
  - (but EFI needs cmdline for now)

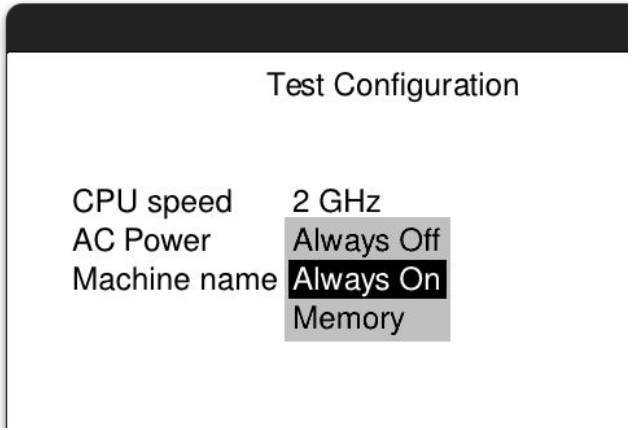
Boot to Linux in 100ms

- Various CONFIG changes:
  - Remove bootdelay
  - Disable U-Boot command line (fully programmatic boot)
  - Disable networking
  - Disable video
  - Disable EFI
- Results
  - QEMU with kvm
  - 82ms from start of U-Boot to Linux handoff*
  - 174K u-boot.bin
  - See https://u-boot.org/blog/booting-into-linux-in-100ms/

* Includes time for SPL, which can be removed with further work

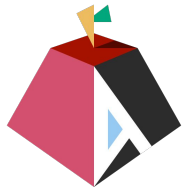
Expo - Flexible and powerful GUI menu

- Introduced in early 2023
  - Supports text, images, Truetype fonts
  - Alignment, multi-line text, etc.
  - Configuration editor with schema
- Coming for Christmas:
  - Simple text editor



EFI App

- Recent work ongoing on x86 and ARM
  - Boot an OS in bzimage format
  - or EFI
- Qualcomm devices use an EFI 'stub'
  - Starts as an API, then takes over
  - Provides its own EFI interface to Linux
  - Resolves issues with low-quality EFI firmware
- Note: Apple devices don't use the EFI app
  - U-Boot starts from first-stage 'Asahi' firmware

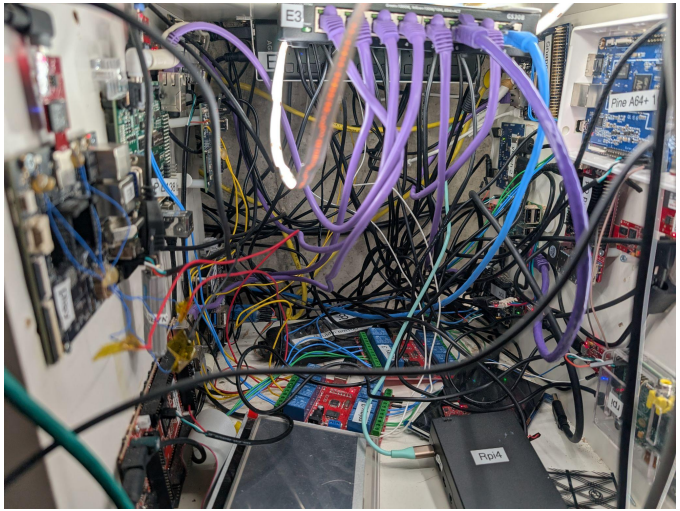


Asahi Linux

My lab

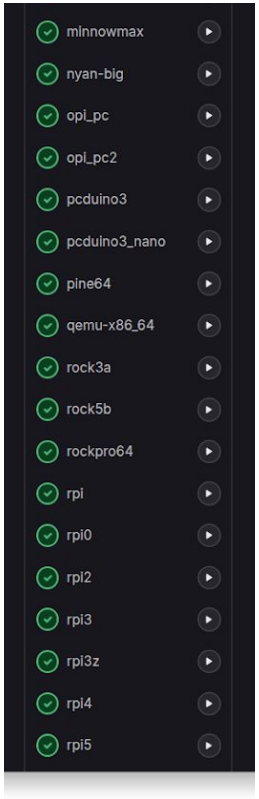
- 40 boards in my daughter's old 'cube' bookcase
- Lots of SD-wires
- 7 hubs, DLI power
- Labgrid†

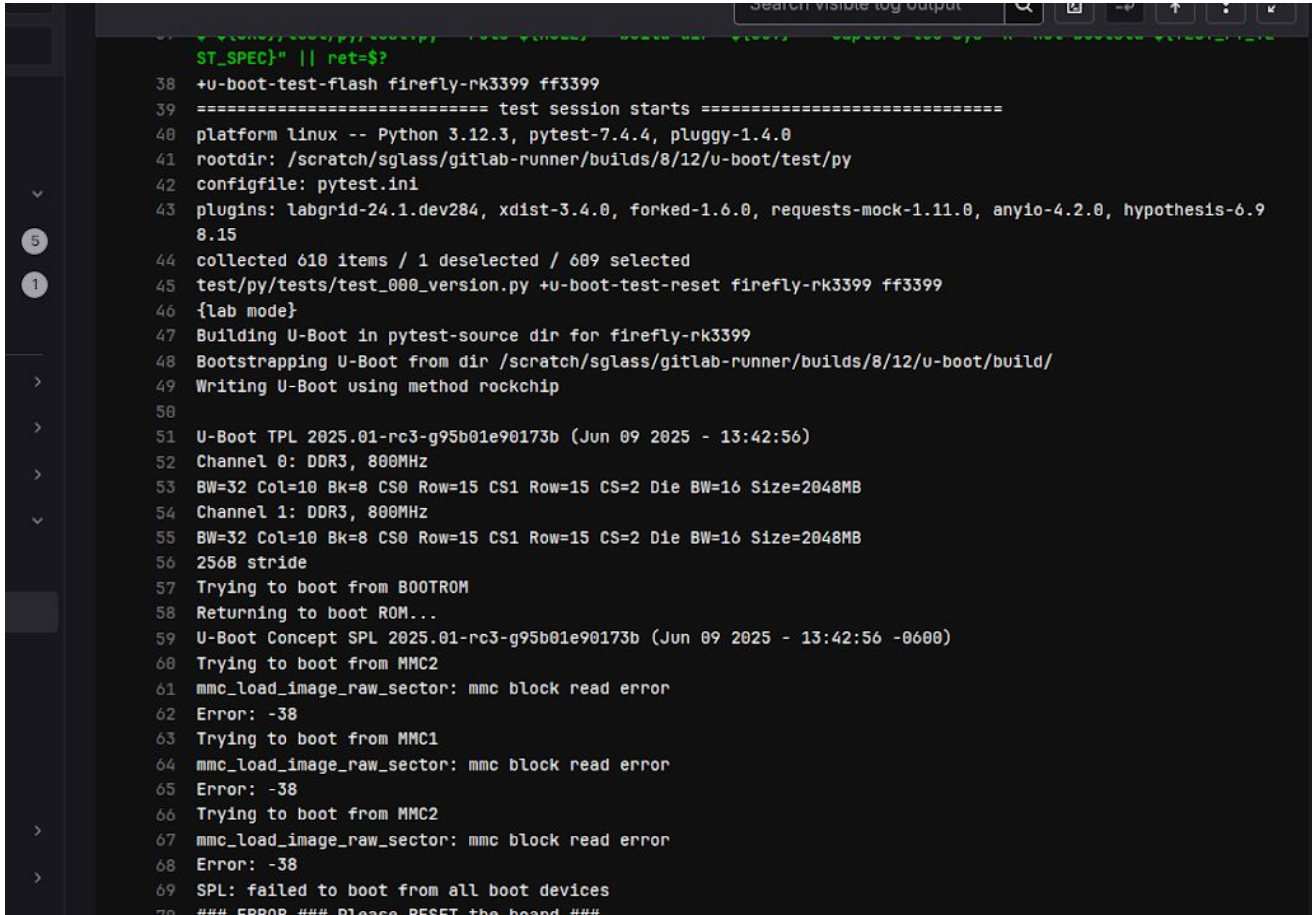
```
$ ub-int rpi3
Building U-Boot in sourcedir for rpi_3_32b
Bootstrapping U-Boot from dir /tmp/b/rpi_3_32b
Writing U-Boot using method rpi3


U-Boot 2025.07-rc4-00451-g62d24441e0a2

DRAM:  992 MiB
RPI 3 Model B (0xa22082)
```



† With 72 patches https://github.com/labgrid-project/labgrid/pull/1411

Lab in CI


minnowmax
nyan-big
opi_pc
opi_pc2
pcduino3
pcduino3_nano
pine64
qemu-x86_64
rock3a
rock5b
rockpro64
rpi
rpi0
rpi2
rpi3
rpi3z
rpi4
rpi5


ST_SPEC}" || ret=$?
+u-boot-test-flash firefly-rk3399 ff3399
============================= test session starts ==============================
platform linux -- Python 3.12.3, pytest-7.4.4, pluggy-1.4.0
rootdir: /scratch/sglass/gitlab-runner/builds/8/12/u-boot/test/py
configfile: pytest.ini
plugins: labgrid-24.1.dev284, xdist-3.4.0, forked-1.6.0, requests-mock-1.11.0, anyio-4.2.0, hypothesis-6.98.15
collected 610 items / 1 deselected / 609 selected
test/py/tests/test_000_version.py +u-boot-test-reset firefly-rk3399 ff3399
{lab mode}
Building U-Boot in pytest-source dir for firefly-rk3399
Bootstrapping U-Boot from dir /scratch/sglass/gitlab-runner/builds/8/12/u-boot/build/
Writing U-Boot using method rockchip
U-Boot TPL 2025.01-rc3-g95b01e90173b (Jun 09 2025 - 13:42:56)
Channel 0: DDR3, 800MHz
BW=32 Col=10 Bk=8 CS0 Row=15 CS1 Row=15 CS=2 Die BW=16 Size=2048MB
Channel 1: DDR3, 800MHz
BW=32 Col=10 Bk=8 CS0 Row=15 CS1 Row=15 CS=2 Die BW=16 Size=2048MB
256B stride
Trying to boot from BOOTROM
Returning to boot ROM...
U-Boot Concept SPL 2025.01-rc3-g95b01e90173b (Jun 09 2025 - 13:42:56 -0600)
Trying to boot from MMC2
mmc_load_image_raw_sector: mmc block read error
Error: -38
Trying to boot from MMC1
mmc_load_image_raw_sector: mmc block read error
Error: -38
Trying to boot from MMC2
mmc_load_image_raw_sector: mmc block read error
Error: -38
SPL: failed to boot from all boot devices
Duration: 2 minutes 26
Finished: 1 week ago
Queued: 6 seconds
Timeout: 10m (from run
Runner: #8 (VKe9dm-A
Tags: lab
Job artifacts
The artifacts were remo
Commit 95b01e90
emulation: Allow bootin
INE
Pipeline #940
sjg-lab
Related jobs
ff3399
samus
samus_tpl
zybo
bbb

Binman - firmware images

- Describe your firmware in the device tree
  - Including blobs!
  - Hierarchical image description
  - Supports FIT, CBFS, FIP, etc.
  - **Embeds the dataset** in the firmware
- Structure is visible to OS
- Command-line interface
  - binman ls
  - binman replace
  - binman extract
- pip install binary-manager

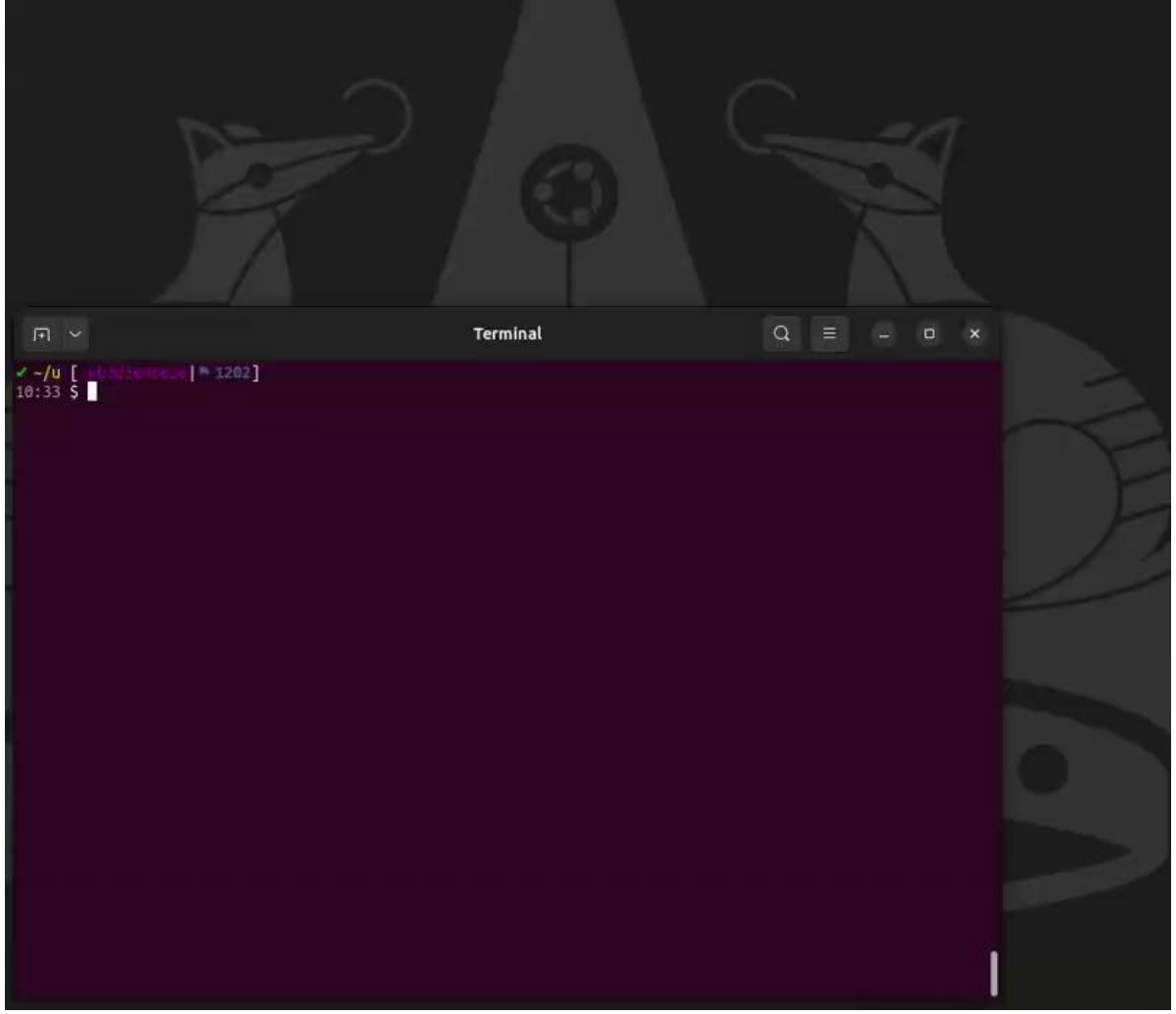
```
binman {
        pad-byte = <0x2e>;

        fill {
                size = <0x40>;
                fill-byte = [41];
        };

        atf-bl31 {
                filename = "bl31.bin";
        };

        u-boot-spl {
                offset = <CONFIG_SPL_PAD_TO>;
                align = <4>;
        };

        u-boot {
        };
```

Binman can fetch / build tools

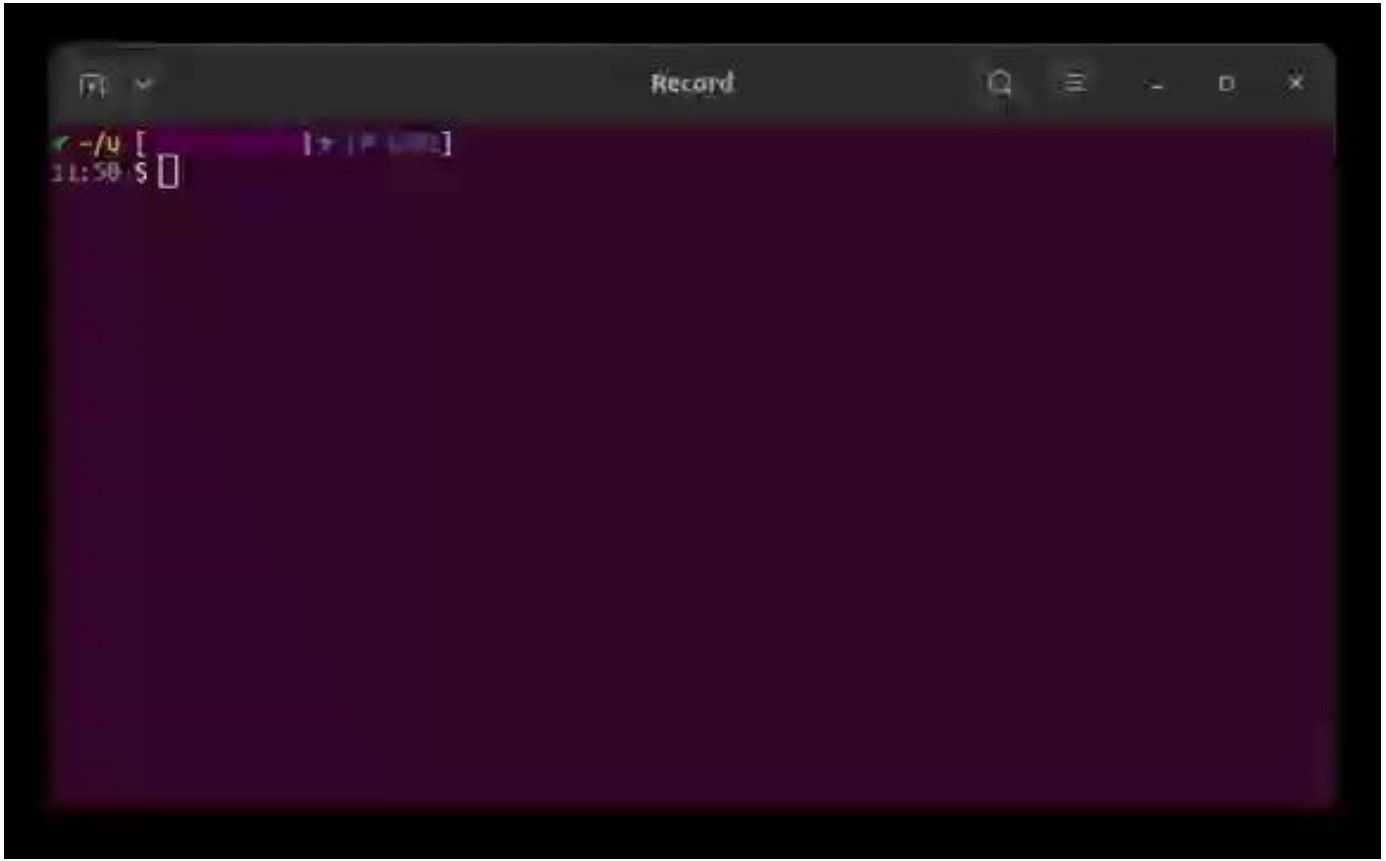
```
08:02 $ binman tool -l
Name              Version      Description                Path
---------------   ----------   -------------------------  ------------------------------
bootgen           -            Xilinx Bootgen             (not found)
bzip2             1.0.8        bzip2 compression          /usr/bin/bzip2
…
08:02 $ binman tool -f bootgen
Fetch: bootgen
- trying method: binary download
- trying method: build from source
- clone git repo 'https://github.com/Xilinx/bootgen' to '/tmp/binmanf.74hyuh7j'
- build target 'all'
- writing to '/home/sglass/.binman-tools/bootgen'
08:02 $ binman tool -l
Name              Version      Description                Path
---------------   ----------   -------------------------  ------------------------------
bootgen           ****** Bootg Xilinx Bootgen             /home/sglass/.binman-tools/bootgen
bzip2             1.0.8        bzip2 compression          /usr/bin/bzip2
…
```

Demo time

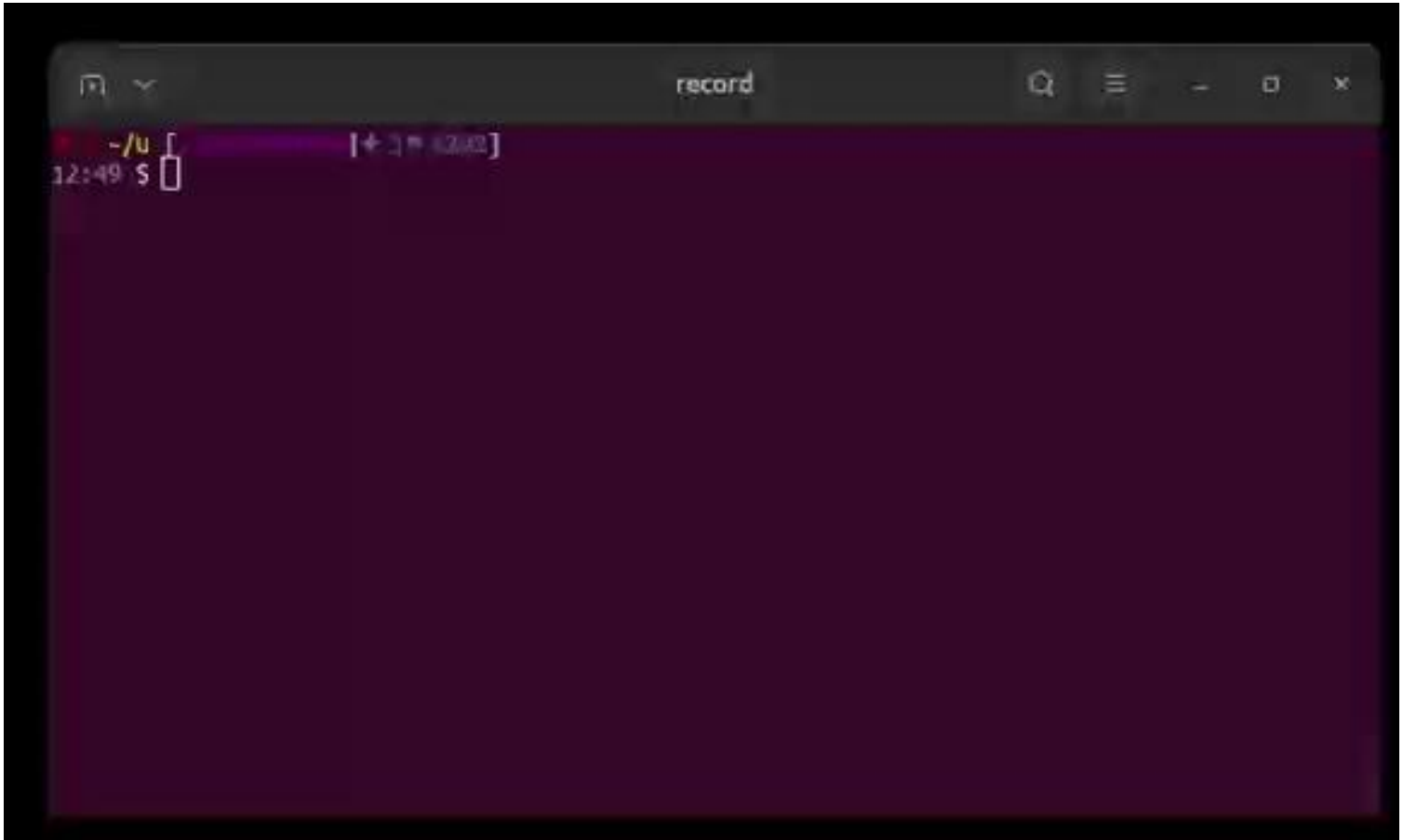
Standard boot with QEMU


Terminal
~/u [
1202]
10:33 $

Boot into Linux in 100ms



Binman on BeaglePlay



What’s next?

My wish list

- Open source from reset
- Open standards and APIs
- Trace device firmware to source-code
- Declare the boot flow in advance

Is EFI the solution?

Firmware / kernel intermediation 1995

Task:
- Load files†
- Setup tables
- Start Linux

vmlinux
initrd
cmdline
devicetree
dt overlays
acpi
smbios

Firmware 1 BIOS

Interfacing 2 lilo

Kernel 3 stext / startup_64

† OS is already chosen / known

Firmware / kernel intermediation 2005

Task:
- Load files†
- Setup tables
- Start Linux

vmlinux
initrd
cmdline
devicetree
dt overlays
acpi
smbios

Firmware 1 — U-Boot, Tianocore

Interfacing 2 — grub, serial, decompress, aslr

Kernel 3 — stext / startup_64

† OS is already chosen / known

Firmware / kernel intermediation 2015

Task:
- Load files†
- Setup tables
- Start Linux

vmlinux
initrd
cmdline
devicetree
dt overlays
acpi
smbios

Firmware 1 — U-Boot, Tianocore

Interfacing 2 — grub, serial, decompress, kaslr, cache

Kernel 3 — stext / startup_64

† OS is already chosen / known

Firmware / kernel intermediation 2025

Task:
- Load files
- Setup tables
- Start Linux

vmlinux
initrd
cmdline
devicetree
dt overlays
acpi
smbios

Firmware 1
U-Boot
Tianocore

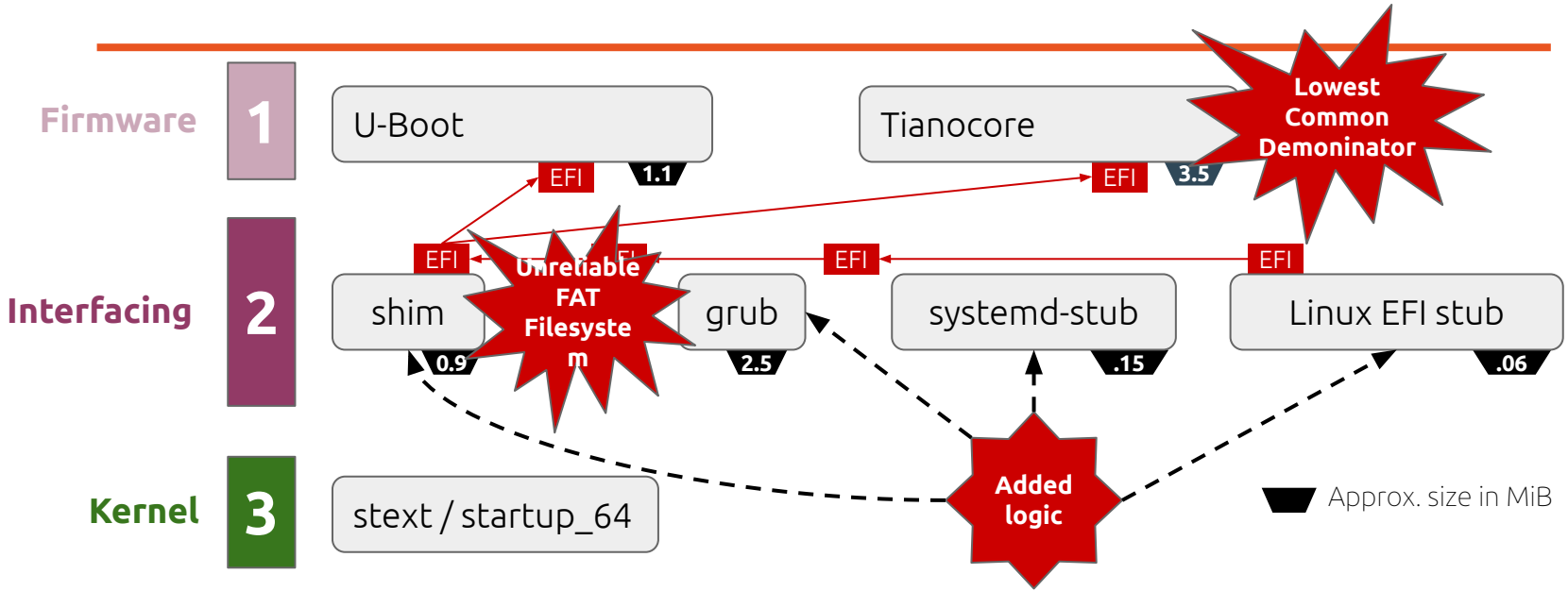
Interfacing 2
shim
grub
systemd-stub
Linux EFI stub

Kernel 3
stext / startup_64

Firmware / kernel intermediation 2025

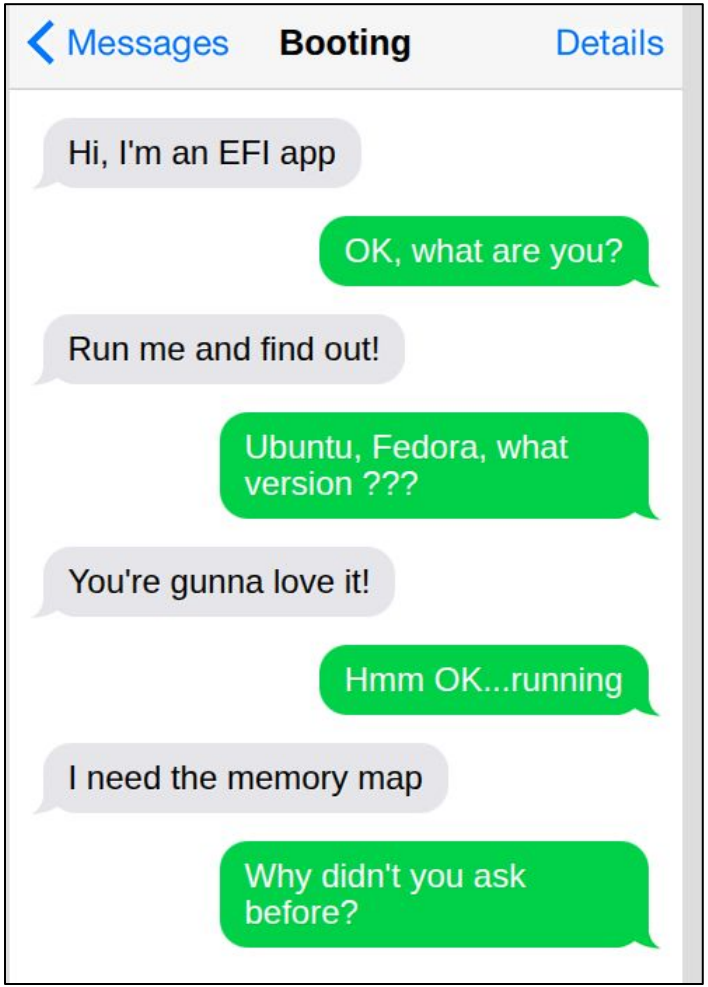
Task:
- Load files
- Setup tables
- Start Linux

vmlinux
initrd
cmdline
devicetree
dt overlays
acpi
smbios



Why the intermediation?

- Lowest-common-denominator
  - Primitive firmware
  - Old forks of Tianocore
  - Old filesystem (FAT)
- Business model
  - OEMs do firmware
  - Microsoft does the OS
- This is not just about x86
  - Arm is copying the same boot approach



Flat Image Tree (FIT)

- Single binary holding the OS and related files
  - Circa 2008 with a few major extensions since
  - initrd, devicetrees and overlays
  - Flexible signatures and hashes
  - **Automatic devicetree selection**
  - Self-describing schema, metadata
- Spec moved out of U-Boot in 2023
  - Recent activity to add rollback support, etc.
  - https://github.com/open-source-firmware/flat-image-tree
- Does not mandate EFI
- Try Linux '**make image.fit**' on arm64

What can we do instead of callbacks?

- Publish the information as metadata

- Linux can put its requirements into the FIT
  - Built at the same time as the kernel
- Firmware can read this and take action
  - Provide the requested data in the devicetree

Our boot stack needs to evolve

- EFI was designed for closed source
  - As an remote-procedure-call mechanism it has value
  - But it gets in the way of innovation
  - Creating a proper EFI implementation is **hard**
  - Apart from Tianocore, only U-Boot has managed it
- Lowest common denominator
  - Assumes very basic firmware features
  - No ext4 / journaling
- But we live in an open source world
  - Let’s invest in open source booting
  - Innovate in U-Boot, Linux
  - Make it great!

What could we have in 10 years? 2035

Task:
- Load files
- Setup tables
- Start Linux

FIT: kernel
initrd
devicetree
dt overlays

acpi

smbios

Firmware 1 — ABrec ⇨ U-Boot ⇨ UPL Specification ⇨ New GPL firmware

"Here are the boot features I support"

Interfacing 2

Kernel 3 — stext / startup_64

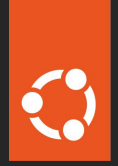
"Here's what I need"

Thanks for listening!

- Bootstd at https://docs.u-boot.org/en/latest/develop/bootstd/overview.html
- Binman at https://docs.u-boot.org/en/latest/develop/package/binman.html

- If you have comments or feedback, or just want to discuss this topic:

Simon Glass
Fellow Software Engineer
simon.glass@canonical.com



“ You didn’t have enough slides!

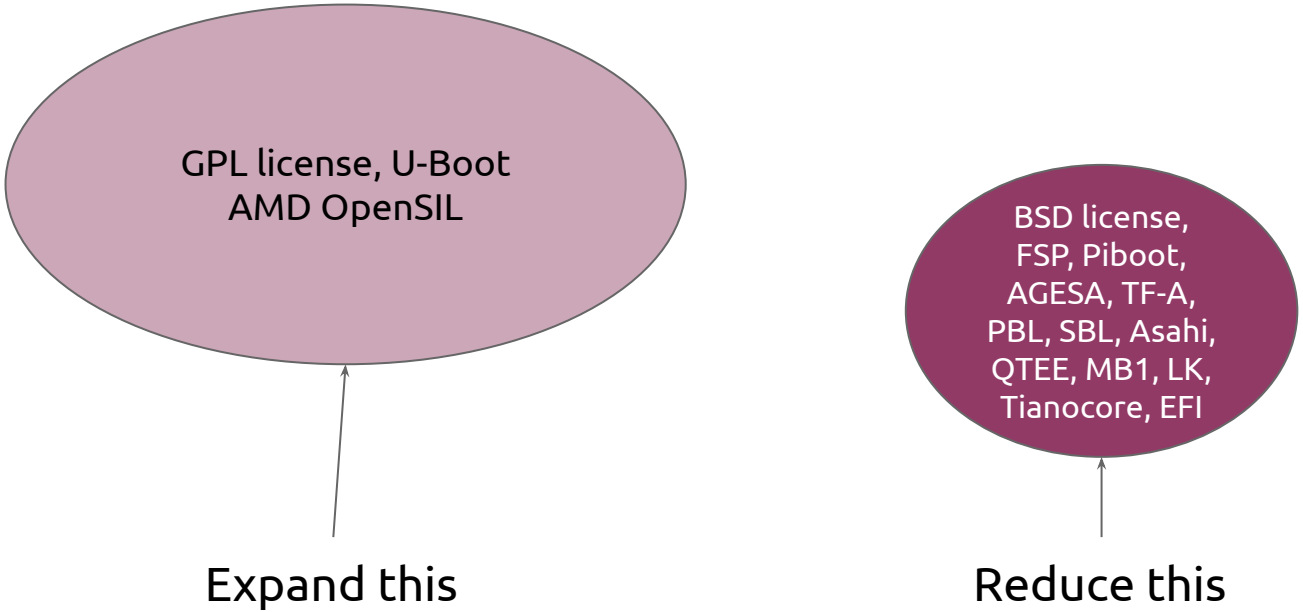
VBE - Verified Boot for Embedded

- Google-era rethink of booting
- OS publishes 'requests' in the FIT
  - Instead of calling back into firmware
- Firmware updateable from pre-RAM phase
- fwupd driver updates firmware
- A/B/recovery flows
- Implemented on RK3399 with 192KB SRAM

```
U-Boot TPL 2025.01-rc3-01787-gf737237d29a5 (Jun 22 2025 - 08:07:53)
Trying to boot from vbe_abrec
load: Firefly-RK3399 Board
## Loading firmware (any) from FIT Image at ff8ecae8 ...
   Using 'config-7' configuration
   Trying 'image-vpl' firmware subimage
## Loading fdt (any) from FIT Image at ff8ecae8 ...
   Using 'config-7' configuration
   Trying 'fdt-7' fdt subimage

U-Boot VPL 2025.01-rc3-01787-gf737237d29a5 (Jun 22 2025 - 08:07:53)
Trying to boot from vbe_abrec
load: Firefly-RK3399 Board
Starting with empty state
VBE: Firmware pick A at 800000
## Loading firmware (SPL Phase) from FIT Image at ff8ec808 ...
   Using 'config-7' configuration
   Trying 'spl' firmware subimage
## Loading fdt (SPL Phase) from FIT Image at ff8ec808 ...
   Using 'config-7' configuration
   Trying 'fdt-7' fdt subimage
Channel 0: DDR3, 800MHz
BW=32 Col=10 Bk=8 CS0 Row=15 CS1 Row=15 CS=2 Die BW=16 Size=2048MB
Channel 1: DDR3, 800MHz
BW=32 Col=10 Bk=8 CS0 Row=15 CS1 Row=15 CS=2 Die BW=16 Size=2048MB
256B stride

U-Boot Concept SPL 2025.01-rc3-01787-gf737237d29a5 (Jun 22 2025 - 08:07:53 -0600)
Trying to boot from vbe_abrec
load: Firefly-RK3399 Board
VBE: Firmware pick A at 900000
## Checking hash(es) for config config-7 ... OK
```

SDRAM init is after A/B/ recovery selection

Open source is a better bet



Fitter FIT

- Tianocore supports FIT
  - But more for the firmware domain (Universal Payload Specification)
  - Anyway, vendor forks probably don’t
- Could we create an EFI app which loads a FIT?