Linux: the first second

Alison Chaiken
alison@she-devel.com
March 3, 2017

The Fifteenth Annual
Southern California Linux Expo

How Linux starts

- What precisely does “off” mean?
- Why does my bootloader have two binaries?
- ACPI vs DTB
- When does the kernel first spawn a second thread?
- What *is* an initrd?
- How does PID 1 start?
- Out-of-bounds: systemd; SecureBoot; fastboot

https://flic.kr/p/6h96xg

THE FOLLOWING CONTENT MAY BE TOO DISTURBING FOR SOME VIEWERS

-- you will witness a kernel panic and a boot failure;

-- a NULL pointer will be dereferenced … successfully!

Applying power

Warm vs. power-on reset

	Clears memory? Restarts clocks?	Pros	Cons	Examples
Power-on Reset	Yes, then reads boot-mode pins.	Won't fail.	Slightly slower.	Plug-in device
Warm Reset	DDR set to 'self-refresh', then reset clocks and jump to stored address.	Faster; retains 'reset reason' and RAM data.	Can fail.	'reboot'; watchdog; JTAG

x86_64: Never genuinely off

Source: Intel

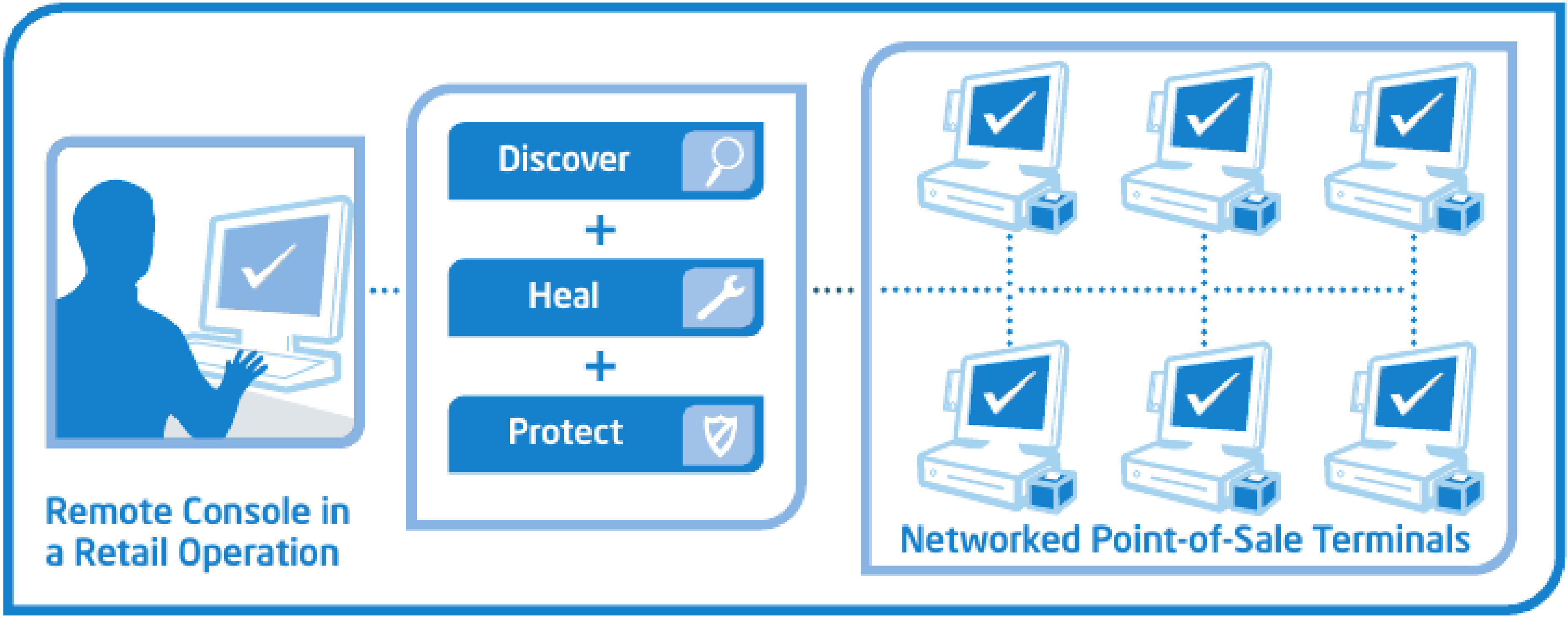
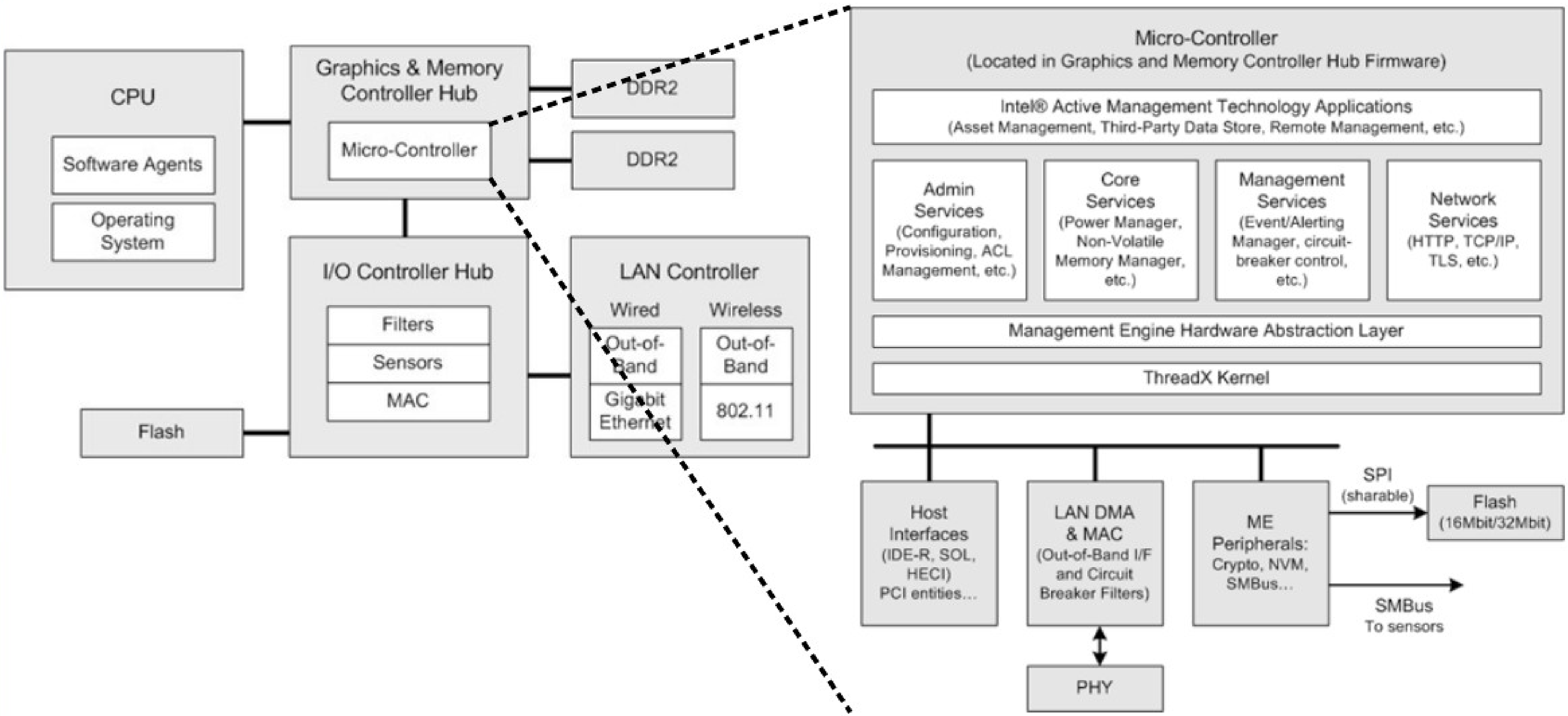


Figure 1. Example using Intel® Active Management Technology in a retail operation to monitor a network of embedded systems even while the enabled systems are powered off.

IPMI: run from Baseboard Management Controller
AMT: run from Graphics and Memory Controller Hub

ME: High-level overview

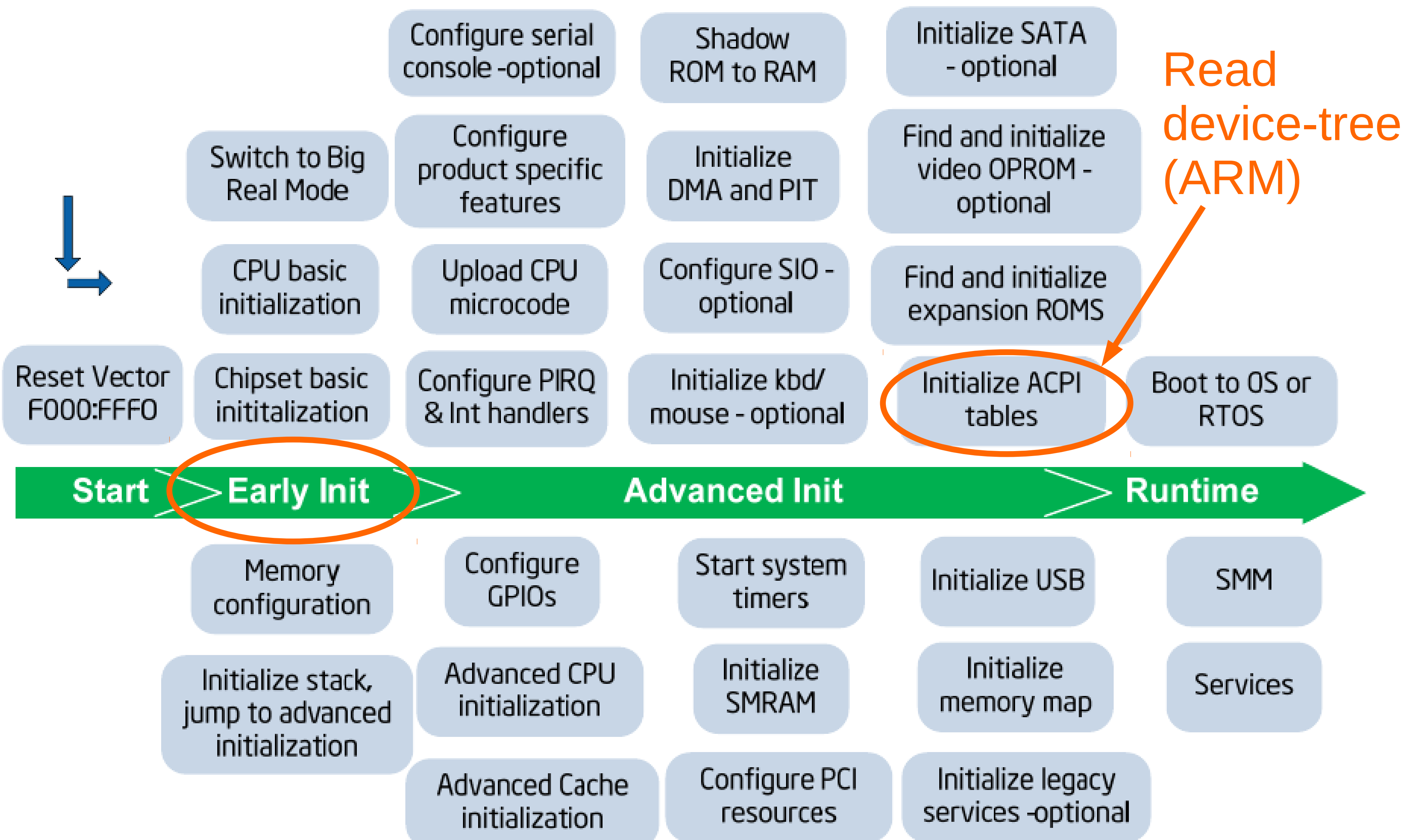


Credit: Intel 2009

Source: https://recon.cx/2014/slides/Recon%202014%20Skochinsky.pdf

Bootloaders:
x86 and u-boot

Bootloaders according to Intel

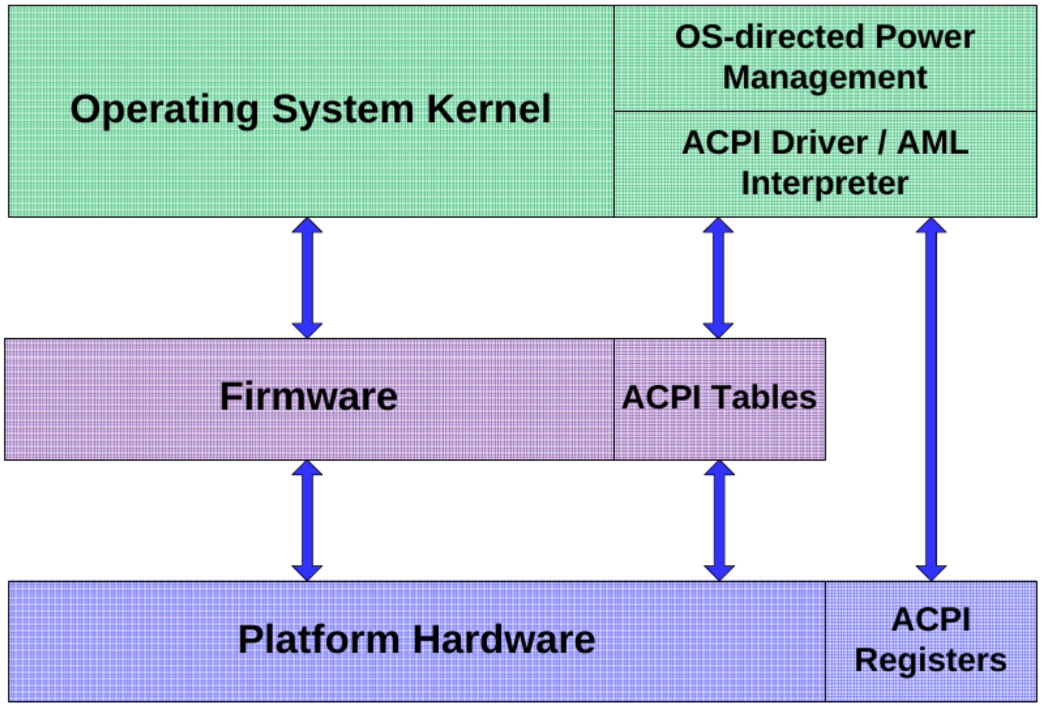


'Shim' bootloader ≈ 'Early Init'

- ARM: “SPL”, “XLoader” or “MLO” in addition to u-boot.img.
- “Software program loader” separates CPU-specific code.
- **Problem**: DRAM controller must be initialized.
- **Solution**: load into SRAM ('OCRAM' in i.MX6, 'l2ram' for TI).
  - *Why this works*: SRAM (and pNOR) are mapped memory.
- **Problem**: SRAM is little! (256K on i.MX6, 2 MB on DRA7x).
- **Solution**: start with a tiny SPL.

Advanced Configuration and Power Interface

Source: Intel



Entertainment: 'sudo acpidump | grep Windows'

Getting more detailed u-boot messages

U-boot config:
# Boot timing
CONFIG_BOOTSTAGE=y
CONFIG_BOOTSTAGE_REPORT=y
CONFIG_BOOTSTAGE_FDT=y

```
Starting kernel ...

Timer summary in microseconds:
       Mark    Elapsed  Stage
          0          0  reset
     71,000     71,000  id=64
     94,000     23,000  id=65
     96,000      2,000  main_loop
  6,137,000  6,041,000  usb_start
 22,826,000 16,689,000  id=80
 22,826,000          0  eth_start
 22,826,000          0  tftp_start
 23,076,000    250,000  id=82
 23,076,000          0  tftp_done
 23,076,000          0  id=81
 23,076,000          0  id=84
 29,384,000  6,308,000  id=1
 29,384,000          0  bootm_start ***
 29,402,000     18,000  id=3
 29,402,000          0  id=2
 29,588,000    186,000  id=5
 29,588,000          0  id=6
 29,588,000          0  id=4
 29,745,000    157,000  id=7
 29,775,000     30,000  id=15
 29,803,000     28,000  start_kernel
```

/* Exiting U-Boot, entering OS */

Interpretation of bootstages

```
enum bootstage_id {
BOOTSTAGE_ID_START = 0,
BOOTSTAGE_ID_CHECK_MAGIC, /* Checking image magic */
BOOTSTAGE_ID_CHECK_HEADER,     /* Checking image header */
BOOTSTAGE_ID_CHECK_CHECKSUM,      /* Checking image checksum */
BOOTSTAGE_ID_CHECK_ARCH,  /* Checking architecture */

BOOTSTAGE_ID_CHECK_IMAGETYPE = 5,/* Checking image type */
BOOTSTAGE_ID_DECOMP_IMAGE,     /* Decompressing image */
BOOTSTAGE_ID_KERNEL_LOADED,   /* Kernel has been loaded */
BOOTSTAGE_ID_DECOMP_UNIMPL = 7,    /* Odd decompression algorithm */
BOOTSTAGE_ID_RUN_OS      = 15,      /* Exiting U-Boot, entering OS */

/* Boot stages related to loading a kernel from an network device */
BOOTSTAGE_ID_NET_CHECKSUM = 60,
BOOTSTAGE_ID_NET_ETH_START = 64,
BOOTSTAGE_ID_NET_ETH_INIT,

BOOTSTAGE_ID_NET_START = 80,
BOOTSTAGE_ID_NET_NETLOOP_OK,
};
```

Passing info from Kernel to Bootloader

- U-boot can pass info in registers or in kernel cmdline.
- Most arches pass “Reset cause” to bootloader.
- ***mtdoops*** and ***ramoops*** save backtraces in persistent stores specified by device-tree.

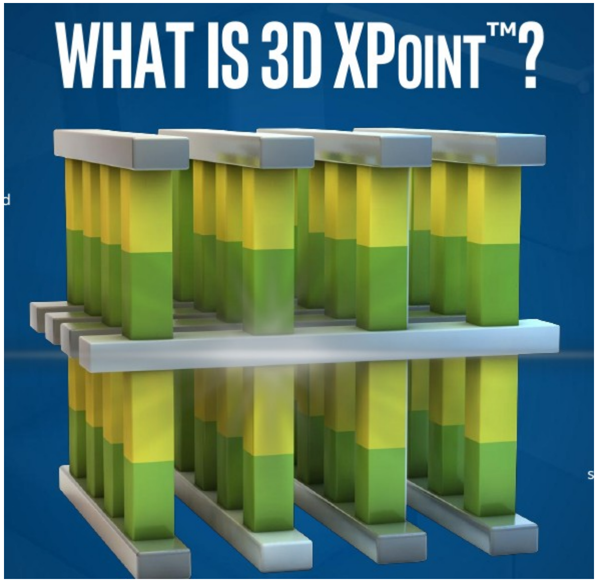
mtdoops in ARMv7 Device-tree

```
flash: m25p80@0 {
        compatible = "sst,sst25vf016b", "jedec,spi-nor";
        spi-max-frequency = <0x1312d00>;
        reg = <0x0>;
        #address-cells = <0x1>;
        #size-cells = <0x1>;

        mtd@00000000 {
                label = "u-boot.img";
                reg = <0x0 0xc0000>;
        };

        mtd@000C0000 {
                label = "u-boot.env";
                reg = <0xc0000 0x2000>;
        };

        mtd@000C2000 {
                label = "splash";
                reg = <0xc2000 0x4000>;
        };

        mtd@000C6000 {
                label = "mtdoops";
                reg = <0xc6000 0x13a000>;
        };
```

The coming revolution in non-volatile storage

Source: Micron

Specs:
ArsTechnica



AKA,
'Optane'
by Intel

- 1 Gb non-volatile memory → suspend even for brief inactivity.
- POR will become a rare event.
- Profound security implications.
- Linux drivers: Matthew Wilcox, XIP →DAX

from notebook interpreted by companion smartphone app

- Intel Optane Performance: *Non-volatile storage medium in the PCIe M.2 format for significant improvements in endurance, performance, and power consumption*

Source:
PCPER

ThinkPad X270 (Image credit: Lenovo)

Starting up the kernel

The kernel is an ELF binary

- vmlinux is a regular ELF binary.
  - readelf -e vmlinux
- Extract vmlinux from vmlinuz:
  - <path-to-kernel-source>/scripts/extract-vmlinux \
    /boot/vmlinuz-$(uname -r) > vmlinux

https://flic.kr/p/6xuhiK

Quiz:

How do ELF binaries start?

Examining ELF binary start with GDB

(results depend on toolchain)

- Compile your C program with '-ggdb'.
- gdb <some-binary-executable>
- Type 'info files'
- Look for 'Entry point'.

- x86_64:
  - Type 'b *(hex address)'
  - Type 'run'
  - Type 'info functions'
- ARM:
  - Type '*l* *(hex address)'
  - Type '*l* 1,80'
  - Type 'info functions' or 'info sources'

```
Reading symbols from drm_info_arm...done.
(gdb) info files
Symbols from "/home/alison/gitsrc/drm-tests-0.1/drm_info_arm".
Local exec file:
        `/home/alison/gitsrc/drm-tests-0.1/drm_info_arm', file type elf32-littl
        Entry point: 0x893c
        0x00008134 - 0x0000814d is .interp
        0x00008150 - 0x00008170 is .note.ABI-tag
        0x00008170 - 0x00008194 is .note.gnu.build-id
        0x00008194 - 0x00008284 is .gnu.hash
        0x00008284 - 0x000084c4 is .dynsym
        0x000084c4 - 0x000086ea is .dynstr
        0x000086ea - 0x00008732 is .gnu.version
        0x00008734 - 0x00008754 is .gnu.version_r
        0x00008754 - 0x00008764 is .rel.dyn
        0x00008764 - 0x00008814 is .rel.plt
        0x00008814 - 0x00008820 is .init
        0x00008820 - 0x0000893c is .plt
        0x0000893c - 0x00009660 is .text
        0x00009660 - 0x00009668 is .fini
        0x00009668 - 0x00009b48 is .rodata
        0x00009b48 - 0x00009b50 is .ARM.exidx
        0x00009b50 - 0x00009b54 is .eh_frame
        0x00011b54 - 0x00011b58 is .init_array
        0x00011b58 - 0x00011b5c is .fini_array
        0x00011b5c - 0x00011b60 is .jcr
        0x00011b60 - 0x00011c60 is .dynamic
        0x00011c60 - 0x00011cc8 is .got
        0x00011cc8 - 0x00011f58 is .data
        0x00011f58 - 0x00011f60 is .bss
(gdb) l *(0x893c)
0x893c is at ../ports/sysdeps/arm/start.S:79.
74                      .type _start,#function
75              _start:
76                      /* Protect against unhandled exceptions.  */
77                      .fnstart
78                      /* Clear the frame pointer and link register since this is the
 frame. */
```

The kernel as PID 0

- Userspace processes need a stack, a heap, STD* file descriptors and an environment to start.
- An ASM constructor “ctor” from crti.o, crtn.o and crt1.0 provided by libgcc allocates these resources.
- Source is start.S.
- Corresponding kernel resources are provided via inline ASM.

Examining kernel start with GDB

1 Type 'file vmlinux'. (If zImage, extract with linux/scripts/extract-vmlinux).

2 Type:

    gdb vmlinux

3 Type:

    info files

4 Find 'Entry point'.

5 Type:

    l *(hex address)

6 Type

    l 1,80

```
Reading symbols from vmlinux...done.
(gdb) info files
Symbols from "/home/alison/gitsrc/linux-trees/linux-4.4.4-nitrogen/vmlinux".
Local exec file:
        `/home/alison/gitsrc/linux-trees/linux-4.4.4-nitrogen/vmlinux',
        file type elf32-littlearm.
        Entry point: 0x80008000
        0x80008000 - 0x800081c0 is .head.text
        0x800081c0 - 0x808ccc00 is .text
        0x808ccc00 - 0x808ccc78 is .text.head
        0x808cd000 - 0x80ba5784 is .rodata
        0x80ba5788 - 0x80ba6c08 is .pci_fixup
        0x80ba6c08 - 0x80ba6c14 is .builtin_fw
        0x80ba6c14 - 0x80bafc3c is __ksymtab
```

```
        0x80dd5e40 - 0x80e8c8f4 is .bss
(gdb) l *(0x80008000)
0x80008000 is at arch/arm/kernel/head.S:87.
warning: Source file is more recent than executable.
82
83       THUMB( badr    r9, 1f          )       @ Kernel is always entered in ARM.
84       THUMB( bx      r9              )       @ If this is a Thumb-2 kernel,
85       THUMB( .thumb                  )       @ switch to Thumb now.
86       THUMB(1:                       )
87
88      #ifdef CONFIG_ARM_VIRT_EXT
89              bl      __hyp_stub_install
```

Kernel starts in head.S, not start.S.

What's in head.S?

- Type 'file vmlinux.o'
- Try 'arm-linux-gnueabihf-gdb vmlinux.o'
- Type 'info files'
- Type 'l *(0x0)'   <---- **actually works!**

```
(gdb) l *(0x0),*(0x60)
0x0 is at arch/arm/kernel/head.S:367.
367             bl      __hyp_stub_install_secondary
368     #endif
369             safe_svcmode_maskall r9
370
371             mrc     p15, 0, r9, c0, c0              @ get processor id
372             bl      __lookup_processor_type
373             movs    r10, r5                         @ invalid processor?
374             moveq   r0, #'p'                        @ yes, error 'p'
375      THUMB( it      eq )            @ force fixup-able long branch encoding
376             beq     __error_p
377
378             /*
379              * Use the page tables supplied from  __cpu_up.
380              */
381             adr     r4, __secondary_data
382             ldmia   r4, {r5, r7, r12}               @ address to jump to after
383             sub     lr, r4, r5                      @ mmu has been enabled
```

The kernel's main() function: highlights of start_kernel()

start_kernel() {:

- boot_cpu_init();
- page_address_init();
- setup_arch(&command_line);
- page_alloc_init();
- pr_notice("Kernel command line:);
- parse_args("Booting kernel", command_line);
- mm_init();
- sched_init();
- init_IRQ();
- init_timers(); timekeeping_init();
- rest_init();

}

"Activate the first processor."

process the device-tree

setup page tables
and start virtual memory

All timestamps before
are [0.000000]

start
userspace

About Initrds

What *is* an initrd anyway?

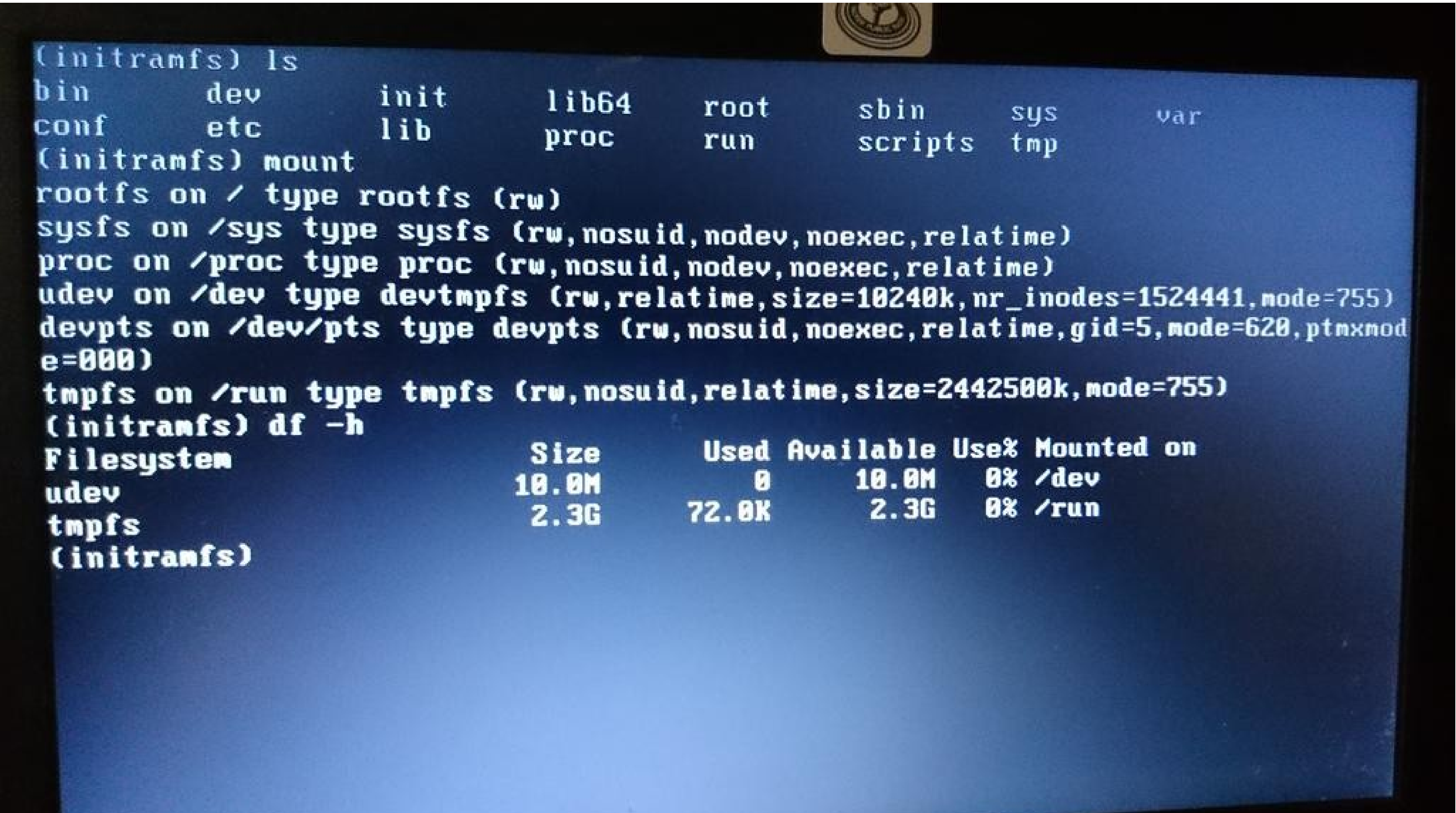
- 'init ramdisk' = filesystem that is loaded into memory by the kernel before the rootfs mounts.
- Why?
  - To provide a **'rescue shell'** in case rootfs doesn't mount.
  - To provide modules that don't fit in zImage.
  - To provide a safe environment to run agressive tests.
  - To facilitate software updates on devices with limited storage.

What's in an initrd and why?

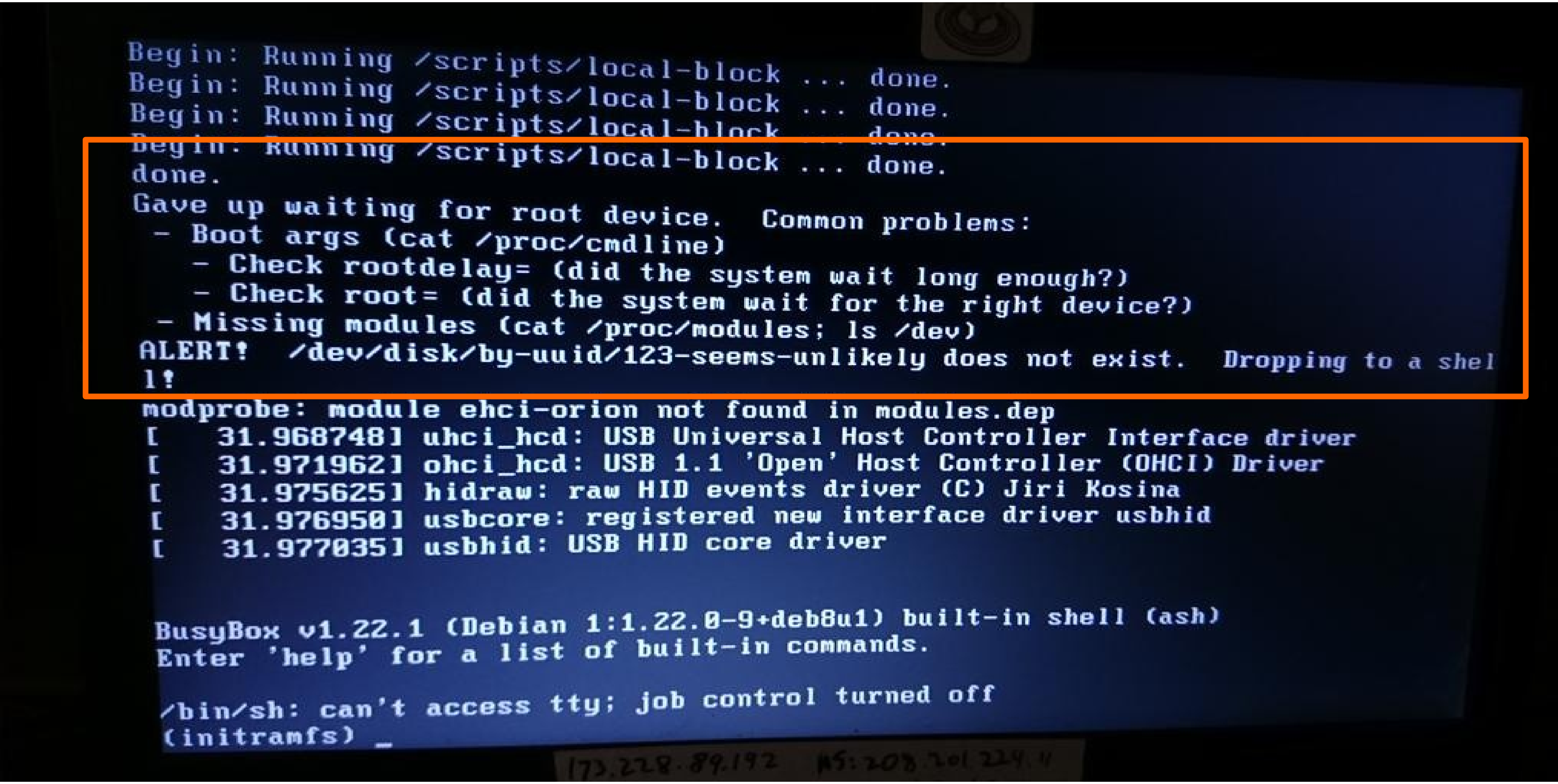
- Boot into the rescue shell by providing a broken cmdline in /boot/grub/grub.cfg
  - Type 'ls'
- Or try 'lsinitramfs /boot/$(uname -r)'
- initrd is a gzipped cpio archive:

```
cp /boot/initrd-$(uname -r)   /tmp/initrd.gz
gunzip /tmp/initrd.gz
cpio -t < /tmp/initrd
```

Exploring initramfs



Booting into Rescue Shell



/scripts/local-block is the function local_block() inside initramfs /scripts/local

OMG! My life is over! (rescue shell tips)

Inhale on a 4-count, then exhale on a 10-count.

- Oh no! 'help' scrolls pages of unreadable crap!

Relax your jaw. Make circles with your neck.

- Read 'man busybox'.
- 'help | grep' works in busybox.
- Look in /bin and /sbin. There's fsck!!
- You have sed and vi (but not emacs ;-( )
- Type 'reboot -f' when you are bored.

Hibernation and Suspension

{Hibernate, Suspend} == Freeze

- *Suspend*: save state to memory.
- *Hibernate*: save state to disk (swap).
- Main code is in kernel/freezer.c.
- Freezing of processes preserves state, protects filesystems.
- Each core must save its state separately.
- To watch, append *no_console_suspend* to bootargs.

How to suspend from CLI

```
root@nitrogen:~# cat /proc/cmdline
console=ttymxc1,115200 ip=172.17.0.1 rootwait rw root=/dev/nfs nfsroot=172.17.0.5:/tftpboot,v3,tcp no_console_suspend
root@nitrogen:~# echo +30 > /sys/class/rtc/rtc0/wakealarm
root@nitrogen:~# echo mem > /sys/power/state
PM: Syncing filesystems ... done.
Freezing user space processes ... (elapsed 0.001 seconds) done.
Freezing remaining freezable tasks ... (elapsed 0.001 seconds) done.
PM: suspend of devices complete after 78.751 msecs
PM: suspend devices took 0.080 seconds
PM: late suspend of devices complete after 1.506 msecs
PM: noirq suspend of devices complete after 1.500 msecs
Disabling non-boot CPUs ...
CPU1: shutdown
CPU2: shutdown
CPU3: shutdown
Enabling non-boot CPUs ...
CPU1 is up
CPU2 is up
CPU3 is up
PM: noirq resume of devices complete after 1.269 msecs
PM: early resume of devices complete after 1.128 msecs
PM: resume of devices complete after 67.159 msecs
PM: resume devices took 0.070 seconds
Restarting tasks ... done.
ata1: SATA link down (SStatus 0 SControl 300)
fec 2188000.ethernet eth0: Link is Up - 1Gbps/Full - flow control rx/tx
```

no_console_suspend in bootargs gives debug output.

How to Hibernate from CLI

```
root@stretch-qemu:~# ls /sys/class/rtc/rtc0
date  device   max_user_freq  power        subsystem  uevent
dev   hctosys  name           since_epoch  time       wakealarm
root@stretch-qemu:~# echo +30 > /sys/class/rtc/rtc0/wakealarm
root@stretch-qemu:~# echo disk > /sys/power/state
```

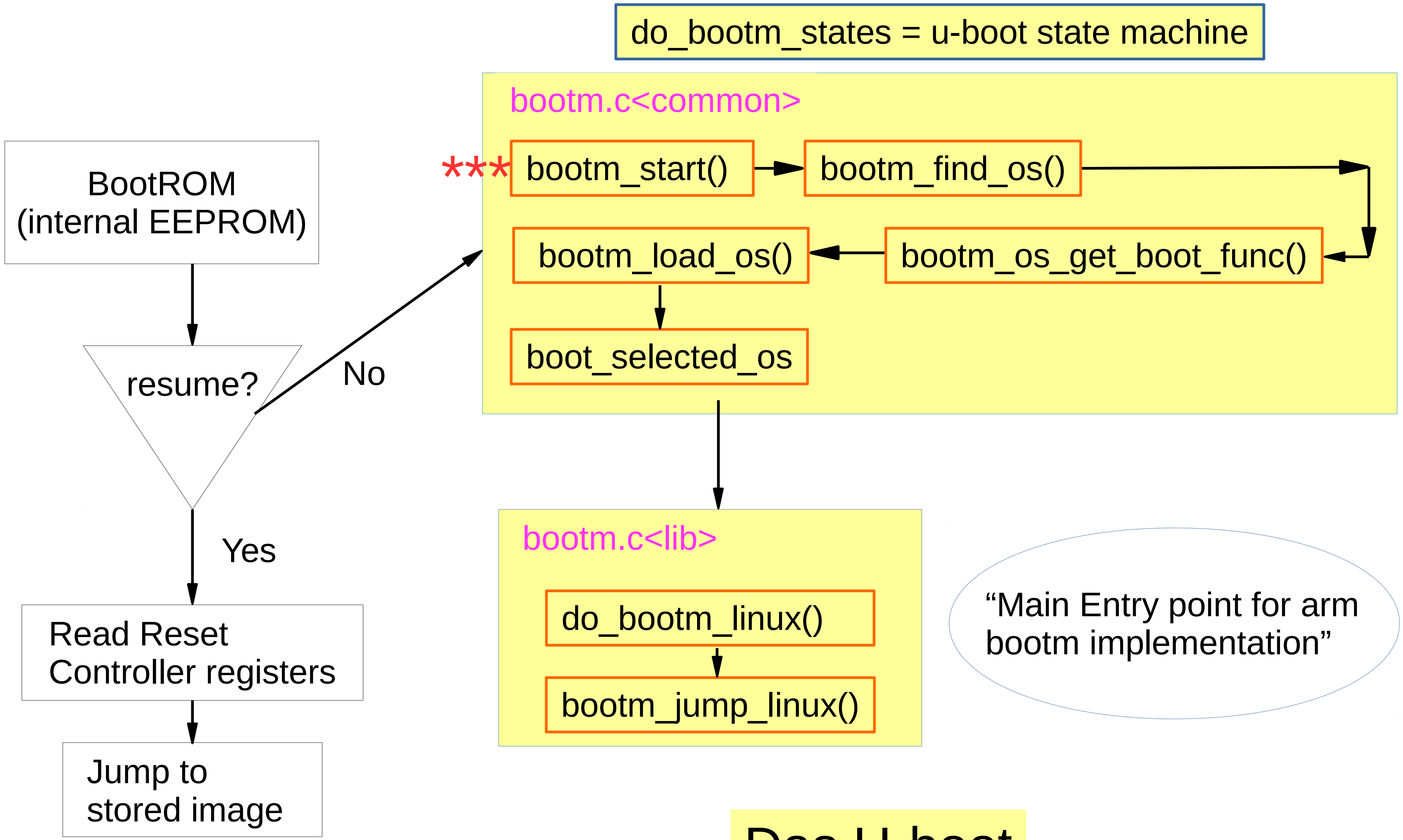
Only works for devices with swap space.

Summary

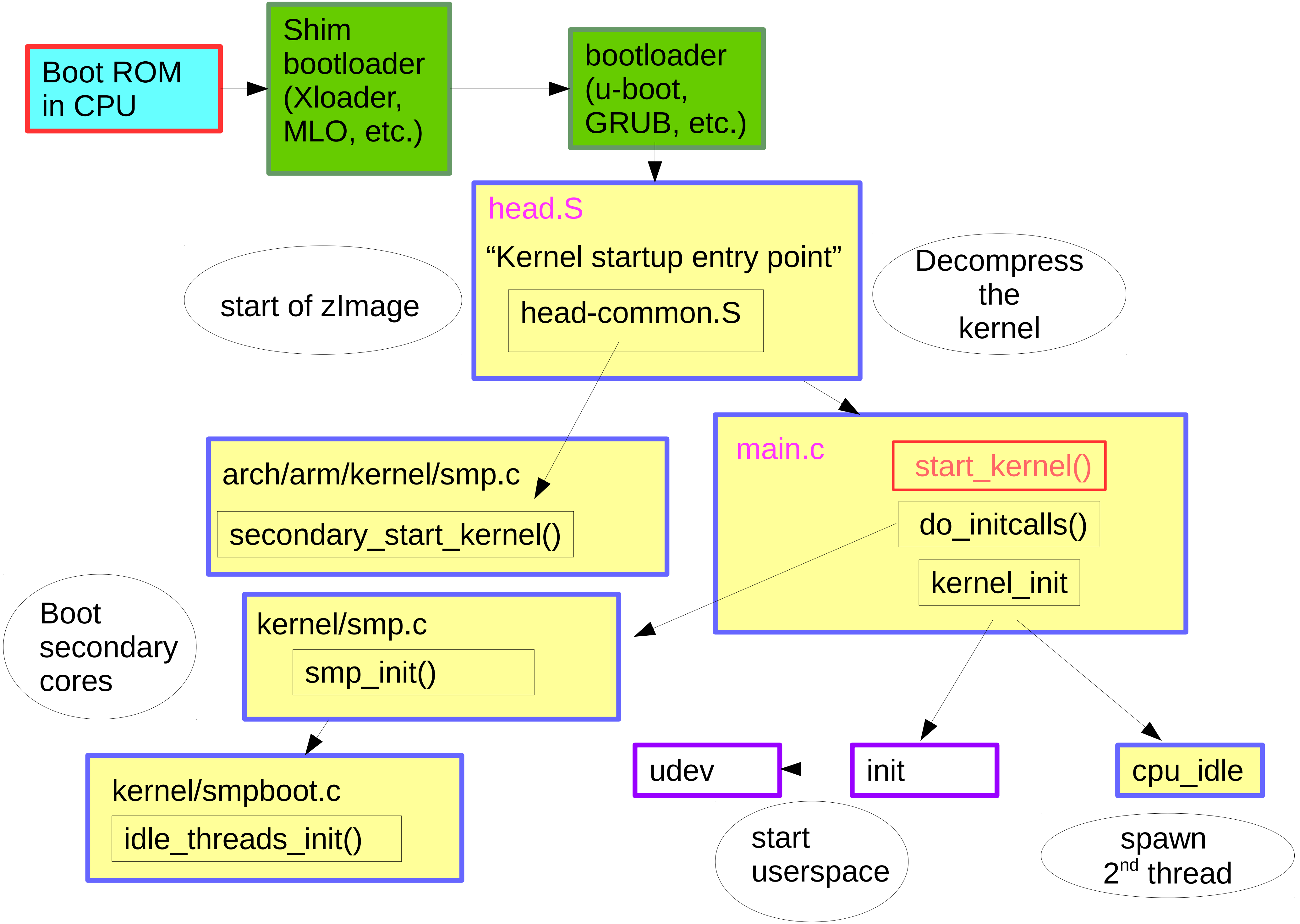
- u-boot's CONFIG_BOOTSTAGES, acpidump and systemd-bootchart provide details about boot.
- Practicing with QEMU is entertaining and low-risk.
- Examine your kernel with GDB and binutils.
- ARM and x86 boot processes are quite distinct.
- High-speed non-volatile memory is about to have a massive impact on Linux.

Major References

- *Embedded Linux Primer* by Chris Hallinan (book)
- Booting ARM Linux by Russell King and THE LINUX/x86 BOOT PROTOCOL (Documentation/)
- Program startup process in userspace at linux-insides blog
- Matthew Garrett's comprehensive series on UEFI
- Status of Intel Management Engine on various laptops (Coreboot) and servers (FSF)
- All about ACPI talk by Darren Hart, ELCE 2013
- Arch Wiki on hacking ACPI tables


do_bootm_states = u-boot state machine
bootm.c<common>
***
bootm_start()
bootm_find_os()
bootm_load_os()
bootm_os_get_boot_func()
boot_selected_os
BootROM
(internal EEPROM)
resume?
No
Yes
Read Reset
Controller registers
Jump to
stored image
bootm.c<lib>
do_bootm_linux()
bootm_jump_linux()
“Main Entry point for arm
bootm implementation”

Das U-boot


Boot ROM in CPU
Shim bootloader (Xloader, MLO, etc.)
bootloader (u-boot, GRUB, etc.)
head.S
“Kernel startup entry point”
head-common.S
start of zImage
Decompress the kernel
arch/arm/kernel/smp.c
secondary_start_kernel()
main.c
start_kernel()
do_initcalls()
kernel_init
Boot secondary cores
kernel/smp.c
smp_init()
kernel/smpboot.c
idle_threads_init()
udev
init
cpu_idle
start userspace
spawn 2nd thread

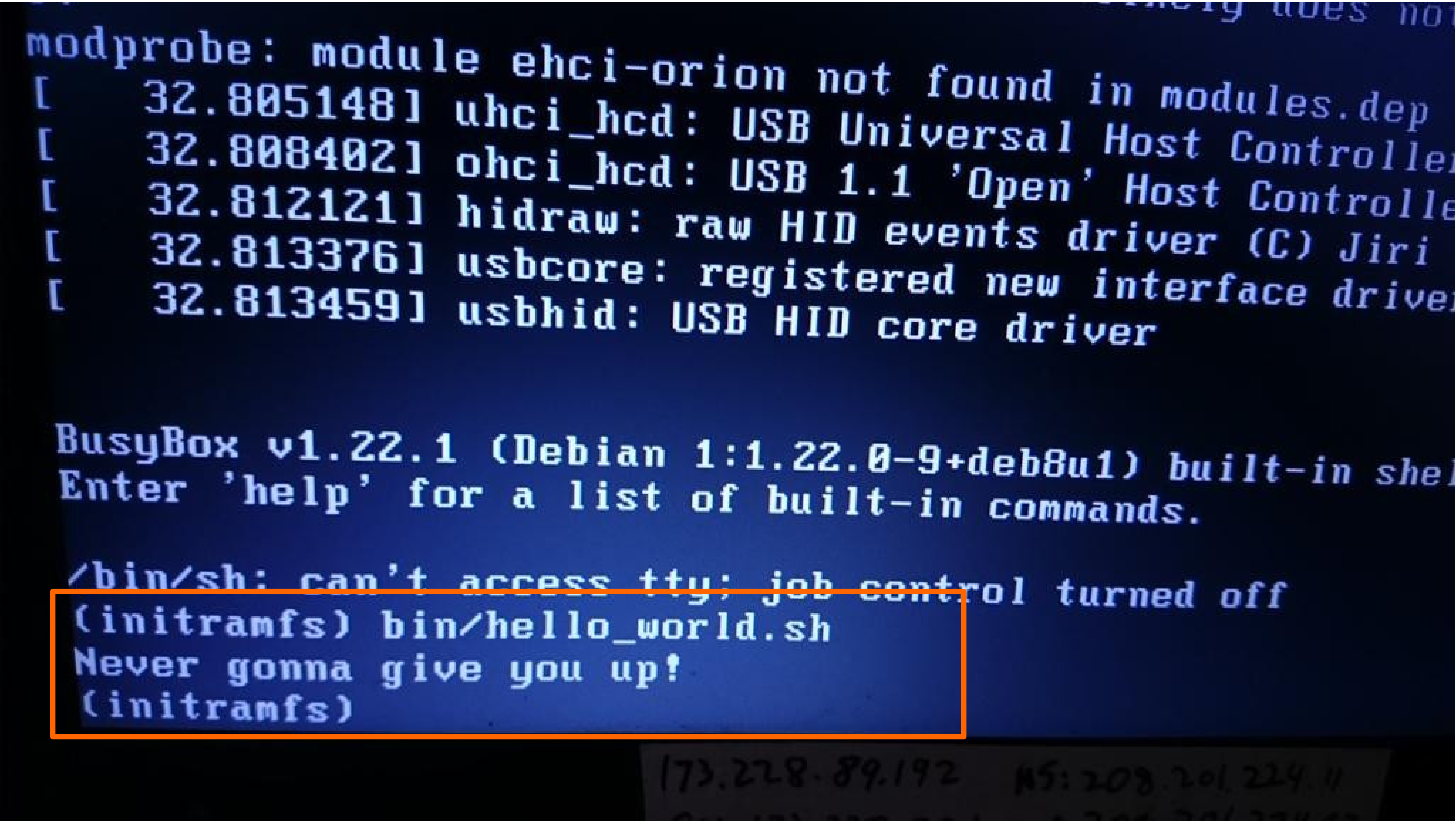
How to create your own initrd

- Unpack one that already works with gunzip and 'cpio -i'
- Copy in your binary.
- Use gen_initramfs.h from kernel source tree:
  - scripts/gen_initramfs_list.sh -o <archive> <path to source>
- Run 'lsinitramfs <archive>' to check the result.
- cp <archive> /boot; edit /boot/grub/grub.cfg

  CAUTION: your system boots fine, right? You're crazy to mess with the bootloader, you moron.

- Run grub-script-check.

The magnificent result!



Getting more detailed kernel messages at boot

- Remove 'quiet' from the kernel command line.
- How to keep 'quiet' from coming back:
  - edit /etc/grub.d/10_linux and add:

    export GRUB_DISABLE_SUBMENU=y

    export GRUB_CMDLINE_LINUX_DEFAULT=""

CAUTION: your system boots fine, right? You're crazy to mess with the bootloader, you moron.

- Always run 'grub-script-check /boot/grub/grub.cfg' afterwards.

Learning more with systemd-bootchart

- Make sure kernel is compiled with CONFIG_SCHEDSTATS=y.
- 'apt-get install systemd-bootchart'
- Interrupt grub by typing 'e'
- Append 'init=/lib/systemd/systemd-bootchart' to the line that starts with 'linux'
- After boot, open the SVG image in /run/log/ with a browser.

A change in compiling your own kernel

```
  LD      kernel/built-in.o
  CC      certs/system_keyring.o
make[1]: *** No rule to make target 'debian/certs/benh@debian.org.cert.pem', needed by 'cert
s/x509_certificate_list'.  Stop.
Makefile:970: recipe for target 'certs' failed
make: *** [certs] Error 2
```

- *To*: 823107-done@bugs.debian.org
- *Subject*: Re: Bug#823107: linux: make deb-pkg fails: No rule to make target 'debian/certs/benh@debian.org.cert.pem'
- *From*: Ben Hutchings <ben@decadent.org.uk>
- *Date*: Sat, 30 Apr 2016 22:50:04 +0200

```
Closing, this is not a bug.

You wrote:
[...]
> Should I remove CONFIG_SYSTEM_TRUSTED_KEYS from .config before building
> the kernel? I hope not.
[...]

Yes, you must do that.  Your custom kernel configuration should be
based on the appropriate file provided in linux-source-4.5.  These have
the CONFIG_MODULE_SIG_ALL, CONFIG_MODULE_SIG_KEY and
CONFIG_SYSTEM_TRUSTED_KEYS settings removed so that custom kernels will
get modules signed by a one-time key.

Ben.
```

Appendix: running QEMU

```
#!/bin/bash
ROOTDIR=/home/alison/ISOs
HDNAME=debian-testing
VERSION=4.9.5

# Load kernel via GRUB; console shows in QEMU window.
#qemu-system-x86_64 -machine accel=kvm -name ${HDNAME} -boot c -drive file=$
{ROOTDIR}/${HDNAME}.raw,format=raw -m 4096 -smp cpus=1 -net nic,model=e1000
-net user,hostfwd=tcp:127.0.0.1:6666-:22 -localtime -serial stdio

# Load kernel from external file; console shows in xterm; GRUB doesn't run.
qemu-system-x86_64 -machine accel=kvm -name ${HDNAME} -initrd
/home/alison/embedded/SCALE2017/kernel/initrd.img-${VERSION} -kernel
/home/alison/embedded/SCALE2017/kernel/vmlinuz-${VERSION} -boot c -drive file=$
{ROOTDIR}/${HDNAME}.raw,format=raw -m 4096 -smp cpus=1 -net nic,model=e1000
-net user,hostfwd=tcp:127.0.0.1:6666-:22 -localtime -serial stdio -append
"console=ttyAMA0  console=ttyS0 root=UUID=8e6a1c7e-b3c4-4a37-8e21-56a137c9dded
ro"
```

Finding u-boot start with GDB

```
[alison@hildesheim u-boot-imx6 (boundary-v2016.03)]$ file u-boot
u-boot: ELF 32-bit LSB shared object, ARM, EABI5 version 1 (SYSV), dynamically
 linked, interpreter /usr/lib/ld.so.1, not stripped
[alison@hildesheim u-boot-imx6 (boundary-v2016.03)]$ arm-linux-gnueabihf-gdb u
-boot
```

```
(gdb) info files
Symbols from "/home/alison/gitsrc/u-boot-imx6/u-boot".
Local exec file:
        `/home/alison/gitsrc/u-boot-imx6/u-boot', file type elf32-littlearm.
        Entry point: 0x17800000
        0x17800000 - 0x17852864 is .text
        0x17852868 - 0x1786646e is .rodata
        0x17866470 - 0x1786649c is .hash
        0x178664a0 - 0x1786b25c is .data
        0x1786b25c - 0x1786b268 is .got.plt
        0x1786b268 - 0x1786bdd0 is .u_boot_list
        0x17877a30 - 0x17877a90 is .dynsym
        0x1786bdd0 - 0x17877a30 is .rel.dyn
        0x1786bdd0 - 0x178b7fd8 is .bss
        0x17877a90 - 0x17877aba is .dynstr
        0x17877abc - 0x17877b3c is .dynamic
        0x17877b3c - 0x17877b4d is .interp
(gdb) l *(0x17800000)
0x17800000 is at arch/arm/lib/vectors.S:54.
49
50      #ifdef CONFIG_SYS_DV_NOR_BOOT_CFG
51              .word   CONFIG_SYS_DV_NOR_BOOT_CFG
52      #endif
53
54              b       reset
55              ldr     pc, _undefined_instruction
56              ldr     pc, _software_interrupt
57              ldr     pc, _prefetch_abort
58              ldr     pc, _data_abort
```

The ARM bootloader

- Read fundamental configuration from fuses, switches and GPIOs.
- Then, for ARM:

  1. Setup and initialise the RAM.
  2. Initialise one serial port.
  3. Detect the machine type.
  4. Setup the kernel ~~tagged list~~. device-tree
  5. Load initramfs.
  6. Call the kernel image.

  Code in the SPL: board_init_f() and jump_to_image_linux()

Where do messages originate?

u-boot

misc.c in zImage header

from kernel proper

[54.590327] Starting kernel ...

[54.593459]

Uncompressing Linux... done, booting the kernel.

Linux version 3.0.35-2508-g54750ff (gcc version 4.6.3 #1 SMP PREEMPT

CPU: ARMv7 Processor [412fc09a] revision 10 (ARMv7), cr=10c53c7d
CPU: VIPT nonaliasing data cache, VIPT aliasing instruction cache

from CPU

Machine: Freescale i.MX 6Quad/DualLite/Solo Sabre-SD Board
Memory policy: ECC disabled, Data cache writealloc
CPU identified as i.MX6Q, silicon rev 1.1
PERCPU: Embedded 7 pages/cpu @8c008000 s5440 r8192 d15040 u32768
Built 1 zonelists in Zone order, mobility grouping on. Total pages: 227328

from kernel

Kernel command line: console=ttymxc1,115200 ip=dhcp rootwait root=/dev/nfs
nfsroot=172.17.0.1:/tftpboot/alison/mx6q/fsl-mx6,v3,tcp

passed from u-boot

Image, zImage, uImage, vmlinux, vmlinuz?

- *Image* is the raw executable.
- *zImage* is compressed version of Image with prepended uncompression instructions in ASM.
- *uImage* is a *zImage* with a u-boot header.
- *vmlinux* is ELF executable containing *Image* in .text section.
- *vmlinuz* is a stripped version of vmlinux.