Linux: the first second

Alison Chaiken
alison@she-devel.com
January 25, 2018

All code for demos

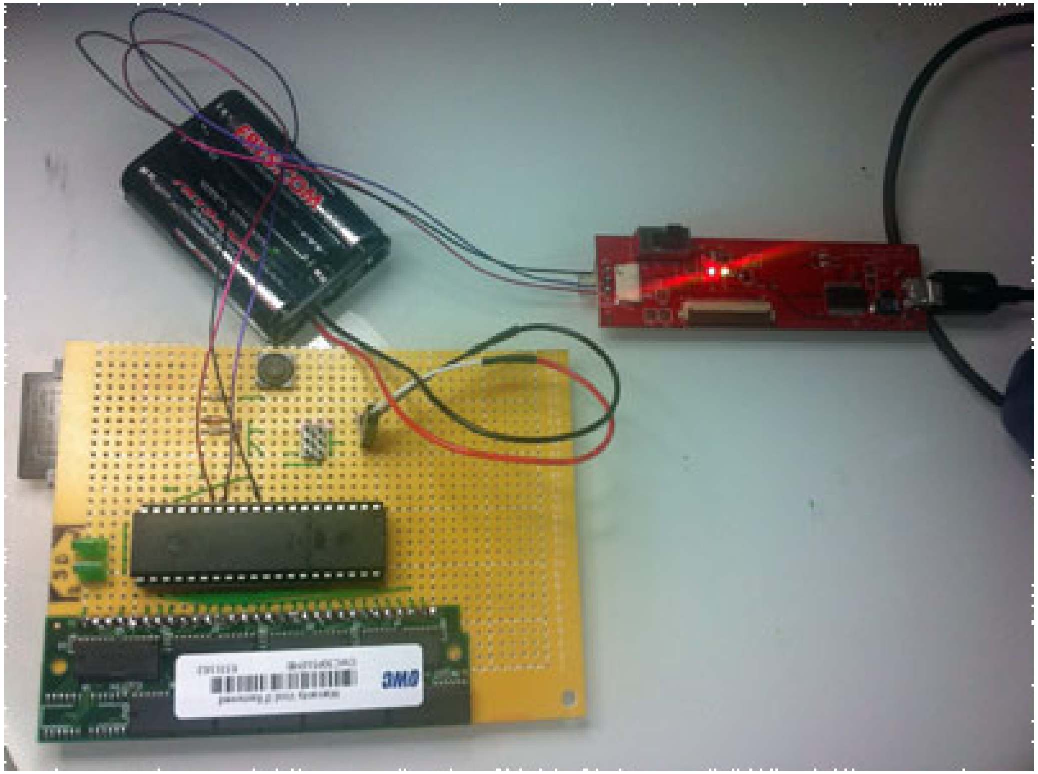
Related blog post at opensource.com

Fast boot: US gov't requires reverse-video in 2s

~~Panic~~ Concern ensues among automakers shipping Linux.

Slow boot: Linux boot on 8-bit AVR

"uARM is certainly no speed demon. It takes about 2 hours to boot to bash prompt".



System:

8-bit micro,

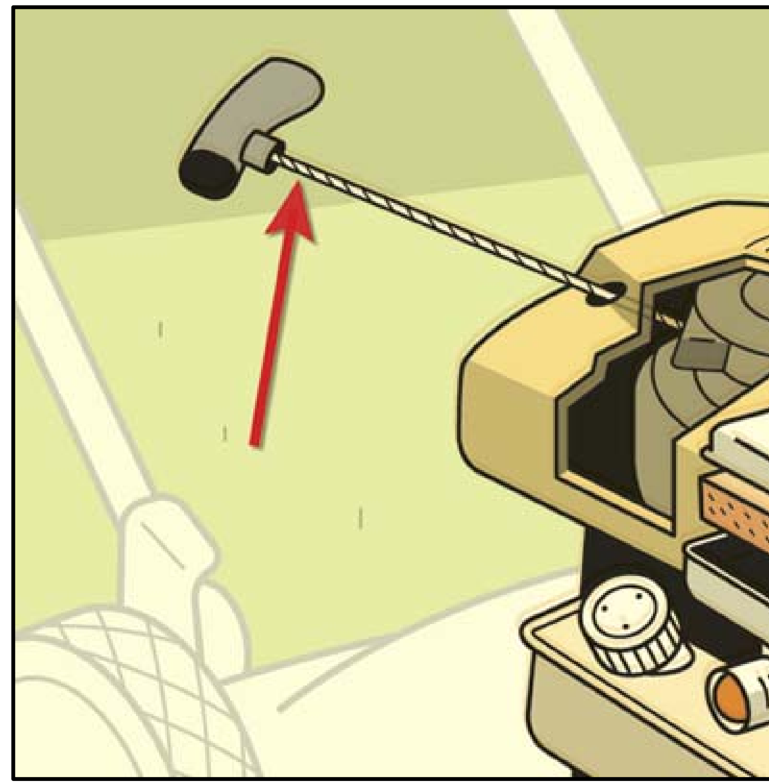
external storage,

external RAM,

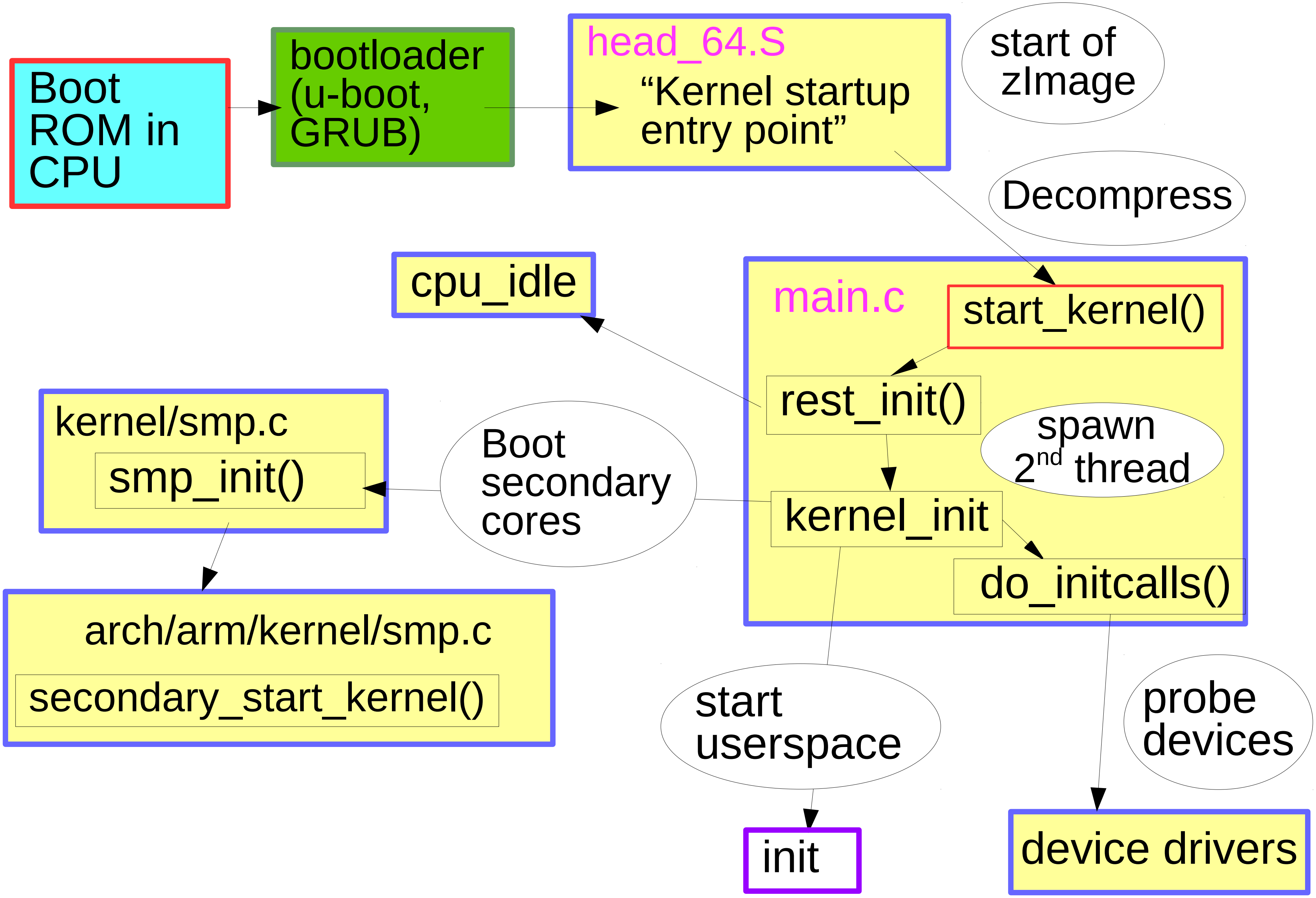
32-bit ARMv5 emulation.

How Linux starts

- What precisely does “off” mean?
- Fun with bootloaders
- ACPI vs DTB
- The kernel as PID 0
- How does PID 1 start?
- What *is* an initrd?



http://spartakirada.com/lawnmower-pullcord/


Boot ROM in CPU
bootloader (u-boot, GRUB)
head_64.S
"Kernel startup entry point"
start of zImage
Decompress
main.c
start_kernel()
cpu_idle
rest_init()
spawn 2nd thread
kernel/smp.c
smp_init()
Boot secondary cores
kernel_init
do_initcalls()
arch/arm/kernel/smp.c
secondary_start_kernel()
start userspace
probe devices
init
device drivers

Applying power

https://www.flickr.com/photos/nicknormal/11939558655/in/album-72157644792628117/

x86_64: Never genuinely off

Source: Intel

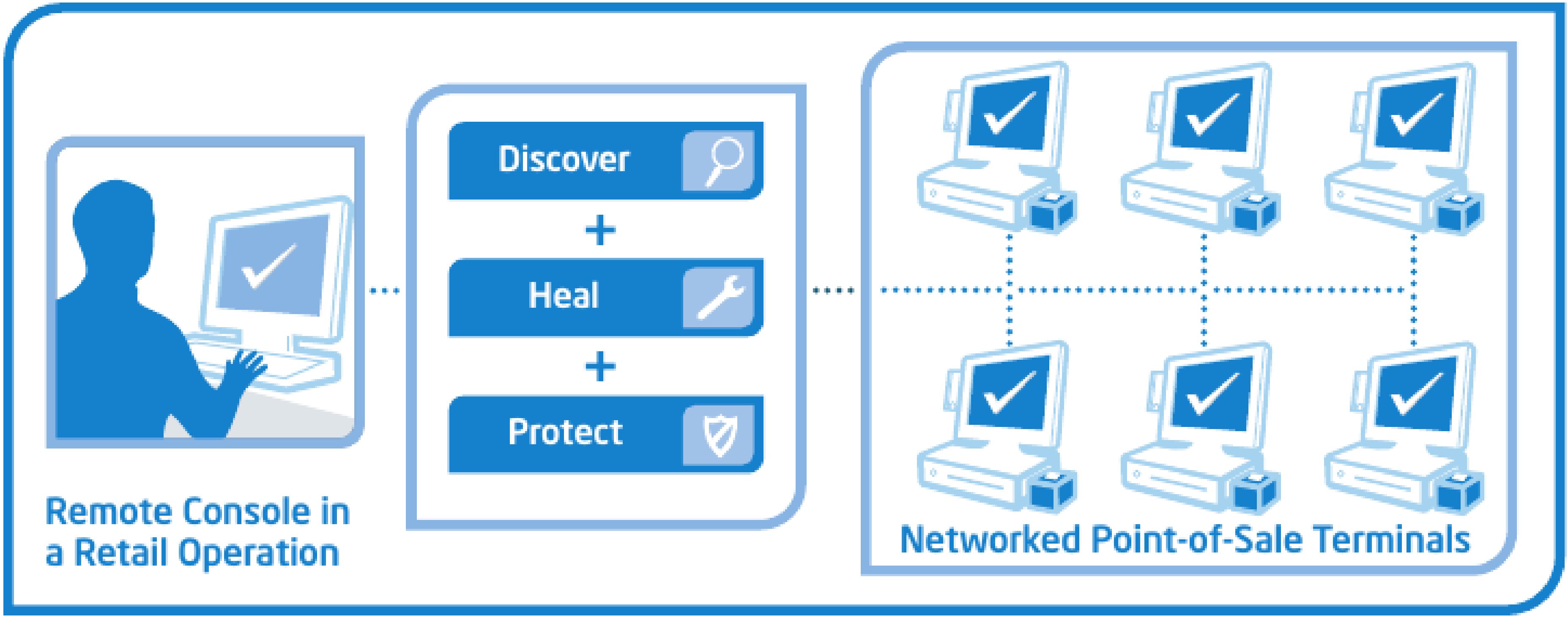
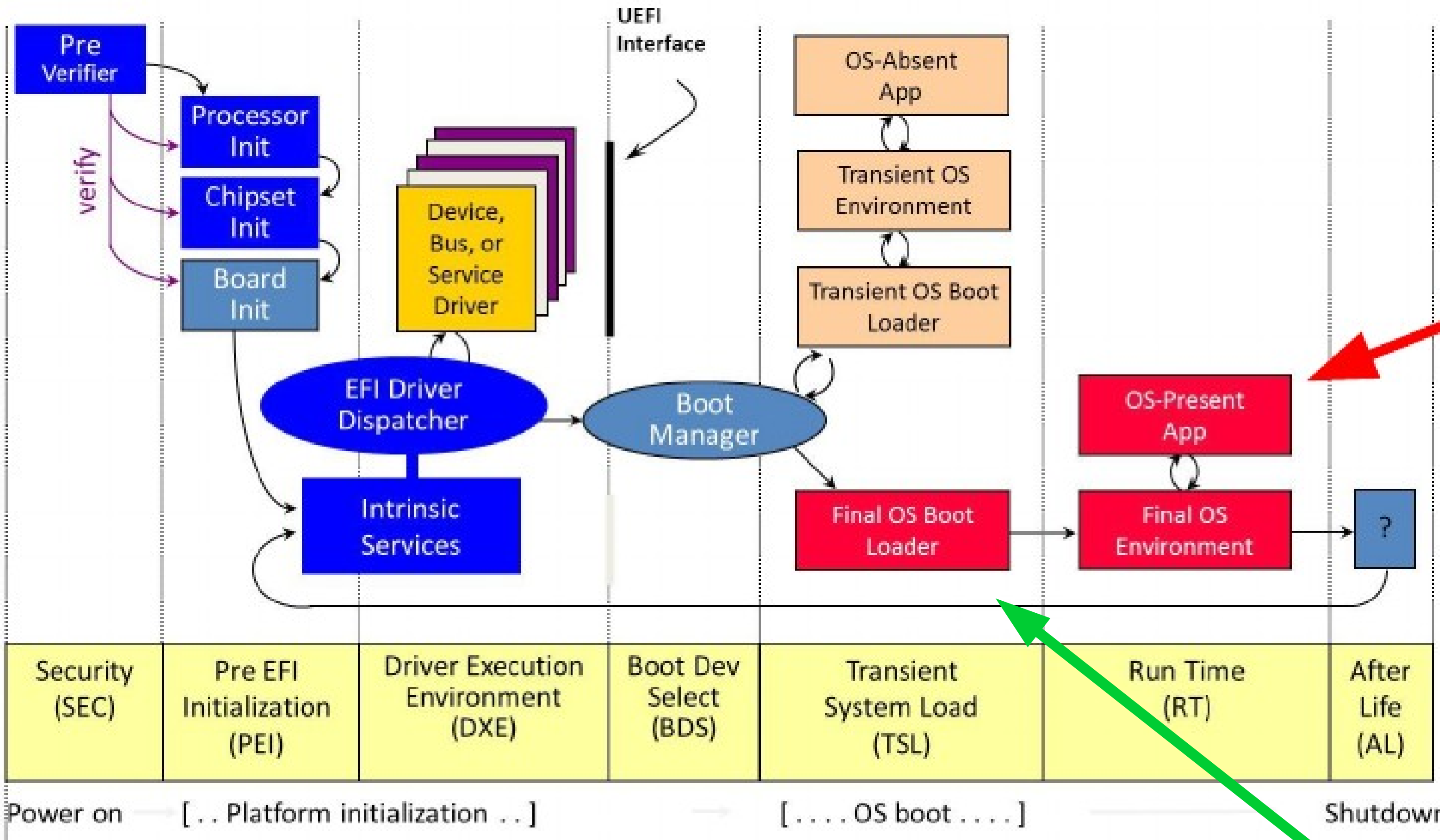


Figure 1. Example using Intel® Active Management Technology in a retail operation to monitor a network of embedded systems even while the enabled systems are powered off.

IPMI: run from Baseboard Management Controller
AMT: run from Platform Controller Hub

Platform Initialization (PI) Boot Phases

OS-present App, a.k.a. exploit home



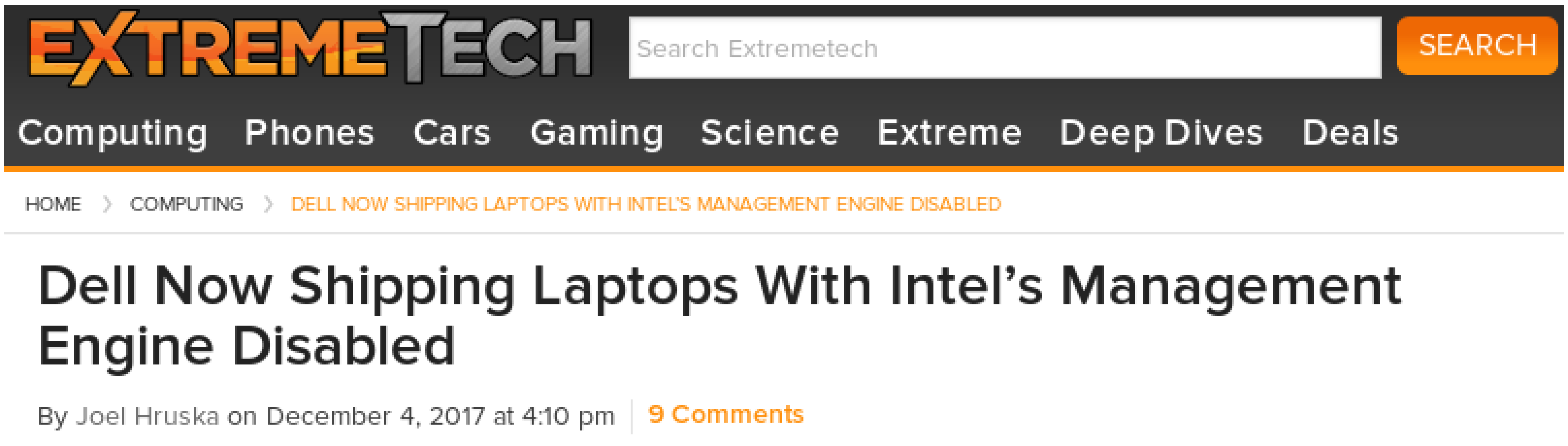
on PCH

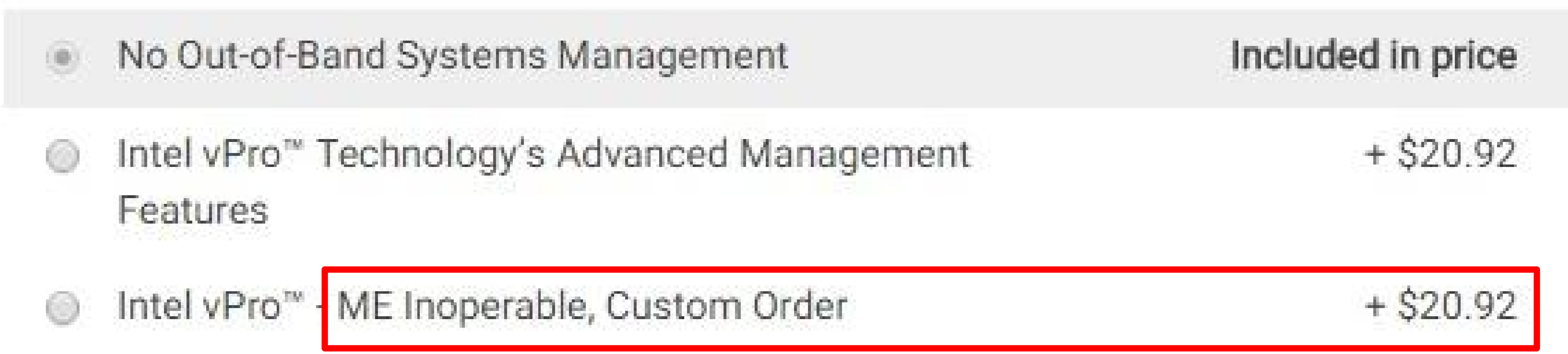
on CPU

GRUB (x86)
or
u-boot (ARM)

Source: Minnich et al., ELCE2017

Purism, System76, Dell turn AMT off





Source: ExtremeTech, December 2017

ARM Bootloader: u-boot

Fun with u-boot's sandbox

(demo placeholder)

- How-to:

```
make ARCH=sandbox defconfig
make
./u-boot
```

- Even more fun:

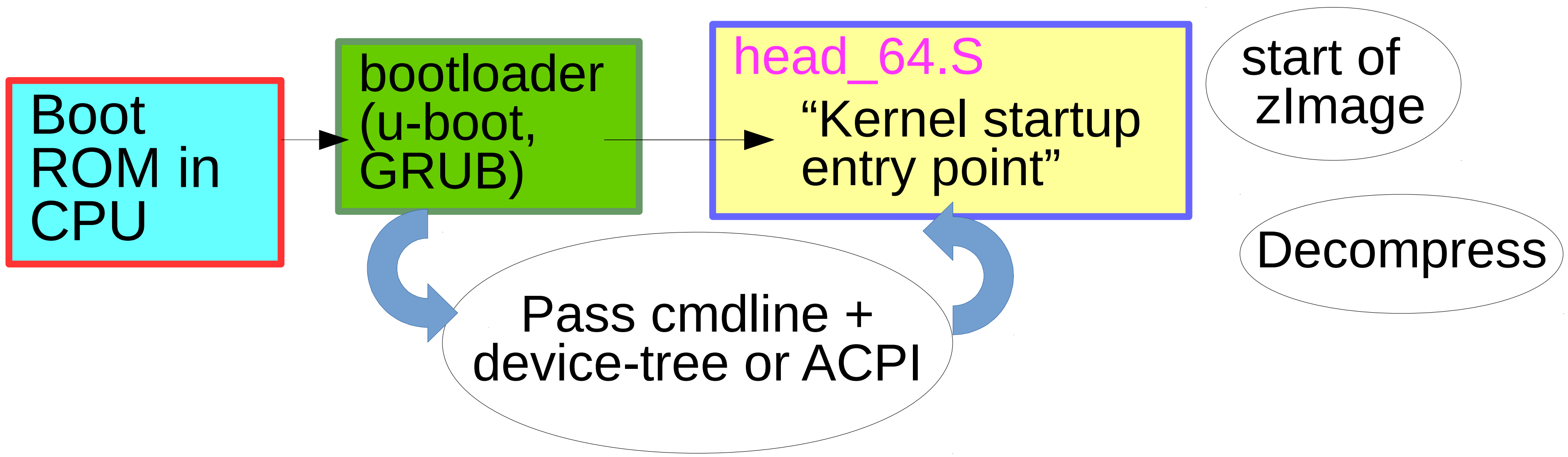
```
make_test_disk.sh
file test.raw; gdisk -l test.raw
./u-boot
host bind 0 test.raw
printenv
gpt read host 0
fatls host 0:1
fdt addr $fdt_addr_
fdt header
```

u-boot : u-boot — Konsole

File Edit View Bookmarks Settings Help

```
$$\# uname -m
x86_64
$$\# pwd
/home/alison/gitsrc/u-boot
$$\# ./u-boot


U-Boot 2017.11-00060-g6b18e4693c (Nov 19 2017 - 13:06:48 -0800)

DRAM:  128 MiB
MMC:
Using default environment

In:    serial
Out:   serial
Err:   serial
SCSI:  Net:   No ethernet found.
IDE:   Bus 0: not available
Hit any key to stop autoboot:  0
reading bzImage
FAT: Misaligned buffer address (00007ff5aff71008)
7972624 bytes read in 16 ms (475.2 MiB/s)
setting up X86 zImage [ 0 - 7972624 ]
## Transferring control to Linux (at address 00000000)...
sandbox: continuing, as we cannot run Linux
=>
```

demo placeholder



How the system reaches the kernel initialization stage

Kernel's “address book”: ACPI or Device-tree

- ACPI tables in SPI-NOR flash.
- *At boot:*

  'dmesg | grep DT'
- *Examine:*

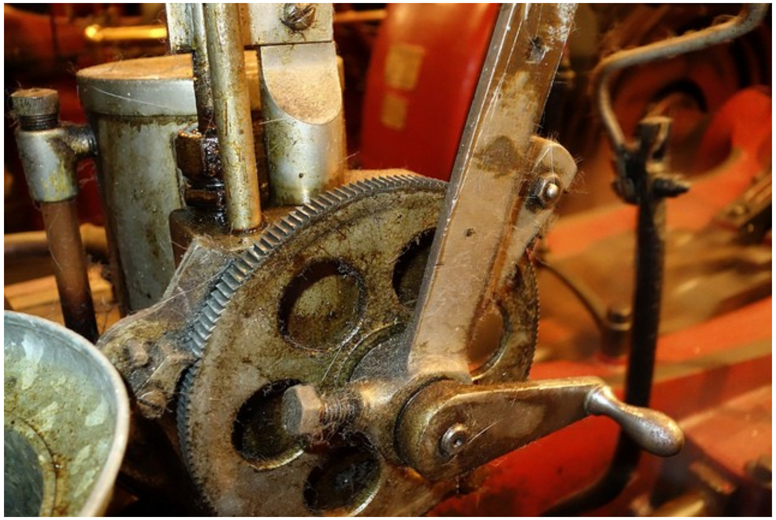
  'acpidump | grep Windows'
- *Get source:* run iasl to extract
- *Modify:* boot-time 'BIOS' menu.

- device-tree in /boot.
- *At boot:*

  each driver reads the DTB.
- *Examine:*

  'strings /boot/foo.dtb'
- *Get source:* from kernel
- *Modify*: edit source, run dtc, copy to /boot.

Starting up the kernel



https://www.flickr.com/photos/nicknormal/11939856433/in/album-72157644792628117/

The kernel is an ELF binary

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

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

Quiz:

How do ELF binaries start?

Quiz:

Where do argc and argv come from?

Inspecting the start of ls with GDB

demo placeholder

```
[alison@hildesheim coreutils-8.28]$ gdb src/ls
Reading symbols from src/ls...done.
(gdb) b _init
Breakpoint 1 at 0x3338
(gdb) run
Starting program: /home/alison/embedded/LCA/demos/coreutils-8.28/src/ls

Breakpoint 1, _init (argc=0x1, argv=0x7fffffffe2e8, envp=0x7fffffffe2f8)
    at ../csu/init-first.c:52
52      {
(gdb) bt
#0  _init (argc=0x1, argv=0x7fffffffe2e8, envp=0x7fffffffe2f8) at ../csu/init-first.c:52
#1  0x00007ffff7de742a in call_init (l=0x7ffff7fd5000, argc=argc@entry=0x1,
    argv=argv@entry=0x7fffffffe2e8, env=env@entry=0x7fffffffe2f8) at dl-init.c:58
#2  0x00007ffff7de7576 in call_init (env=0x7fffffffe2f8, argv=0x7fffffffe2e8, argc=0x1,
    l=<optimized out>) at dl-init.c:119
#3  _dl_init (main_map=0x7ffff7ffe150, argc=0x1, argv=0x7fffffffe2e8,
    env=0x7fffffffe2f8) at dl-init.c:120
#4  0x00007ffff7dd8eda in _dl_start_user () from /lib64/ld-linux-x86-64.so.2
#5  0x0000000000000001 in ?? ()
```

Examining ELF binary start with GDB

(results depend on toolchain and libc)

- Compile your C program with '-ggdb'.
- gdb <some-binary-executable>

intel

- set backtrace past-main on
- set backtrace past-entry on
- Type 'run'
- frame 1; list

arm

- Type 'info files'
- Find 'Entry point'.
- Type '*l* *(hex address)'
- Type '*l* 1,80'
- Type 'info functions' or 'info sources'

demo placeholder

The kernel as PID 0

- *Userspace* processes need to start need:
  - stack,
  - heap,
  - STD* file descriptors
  - environment
- glibc and libgcc allocate these resources.
  - Source is in start.S (ARM) and libc-start.c.
- Corresponding *kernel* resources provided via inline ASM.
  - Reads cmdline, device-tree or ACPI.

Examining ARM32 kernel start with GDB

(demo placeholder)

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

2 Type:

   arm-linux-gnueabihf-gdb vmlinux

3 Type:

   info files

4 Find 'Entry point'.

5 Type:

   l *(hex address)

6 Type

   l 1,80

What's in ARM's 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
```

demo placeholder

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

Examining x86_64 kernel with GDB

(demo placeholder)

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

2 Type:

gdb vmlinux

3 Type:

info files

4 Find '.init.text'.

5 Type:

l *(hex address)

6 Type

l 200,290

What's in x86_64 head_64.S?

```
(gdb) info files
Symbols from "/home/alison/gitsrc/linux-trees/linux/vmlinux".
Local exec file:
        `/home/alison/gitsrc/linux-trees/linux/vmlinux', file type elf64-x86-64.
warning: Cannot find section for the entry point of /home/alison/gitsrc/linux-trees/linux/vmlinux.
        Entry point: 0x1000000
        0xffffffff81000000 - 0xffffffff820916eb is .text
        0xffffffff820916ec - 0xffffffff820918c0 is .notes
        0xffffffff820918c0 - 0xffffffff82093870 is __ex_table
        0xffffffff82200000 - 0xffffffff823e5562 is .rodata
        0xffffffff823e5568 - 0xffffffff823e9240 is .pci_fixup
        0xffffffff823e9240 - 0xffffffff823fa450 is __ksymtab
        0xffffffff823fa450 - 0xffffffff82408210 is __ksymtab_gpl
        0xffffffff82408210 - 0xffffffff8240c694 is __kcrctab
        0xffffffff8240c694 - 0xffffffff8240fe04 is __kcrctab_gpl
        0xffffffff8240fe04 - 0xffffffff82435a23 is __ksymtab_strings
        0xffffffff82435a40 - 0xffffffff82435af0 is __init_rodata
        0xffffffff82435af0 - 0xffffffff82437738 is __param
        0xffffffff82437738 - 0xffffffff82438000 is __modver
        0xffffffff82600000 - 0xffffffff837c3340 is .data
        0xffffffff837c3340 - 0xffffffff837d18e4 is __bug_table
        0xffffffff837d2000 - 0xffffffff837d3000 is .vvar
        0x0000000000000000 - 0x000000000001c0d8 is .data..percpu
        0xffffffff837f0000 - 0xffffffff8387c373 is .init.text
```

demo placeholder

```
(gdb) l *(0xffffffff837f0000)
0xffffffff837f0000 is at arch/x86/kernel/head_64.S:287.
282             .endif
283             pushq $i                # 72(%rsp) Vector number
284             jmp early_idt_handler_common
285             i = i + 1
286             .fill early_idt_handler_array + i*EARLY_IDT_HANDLER_SIZE - ., 1, 0xcc
287             .endr
288     ENDPROC(early_idt_handler_array)
289
290     early_idt_handler_common:
291             /*
(gdb) l 200,290
```

The kernel's main() function

```
start_kernel() {:
  boot_cpu_init();
  setup_arch(&command_line);
  page_alloc_init();
  pr_notice("Kernel command line: );
  mm_init();
  sched_init();
  init_IRQ();
  init_timers(); timekeeping_init();
  console_init();
  rest_init();
}
```

“Activate the first processor.”

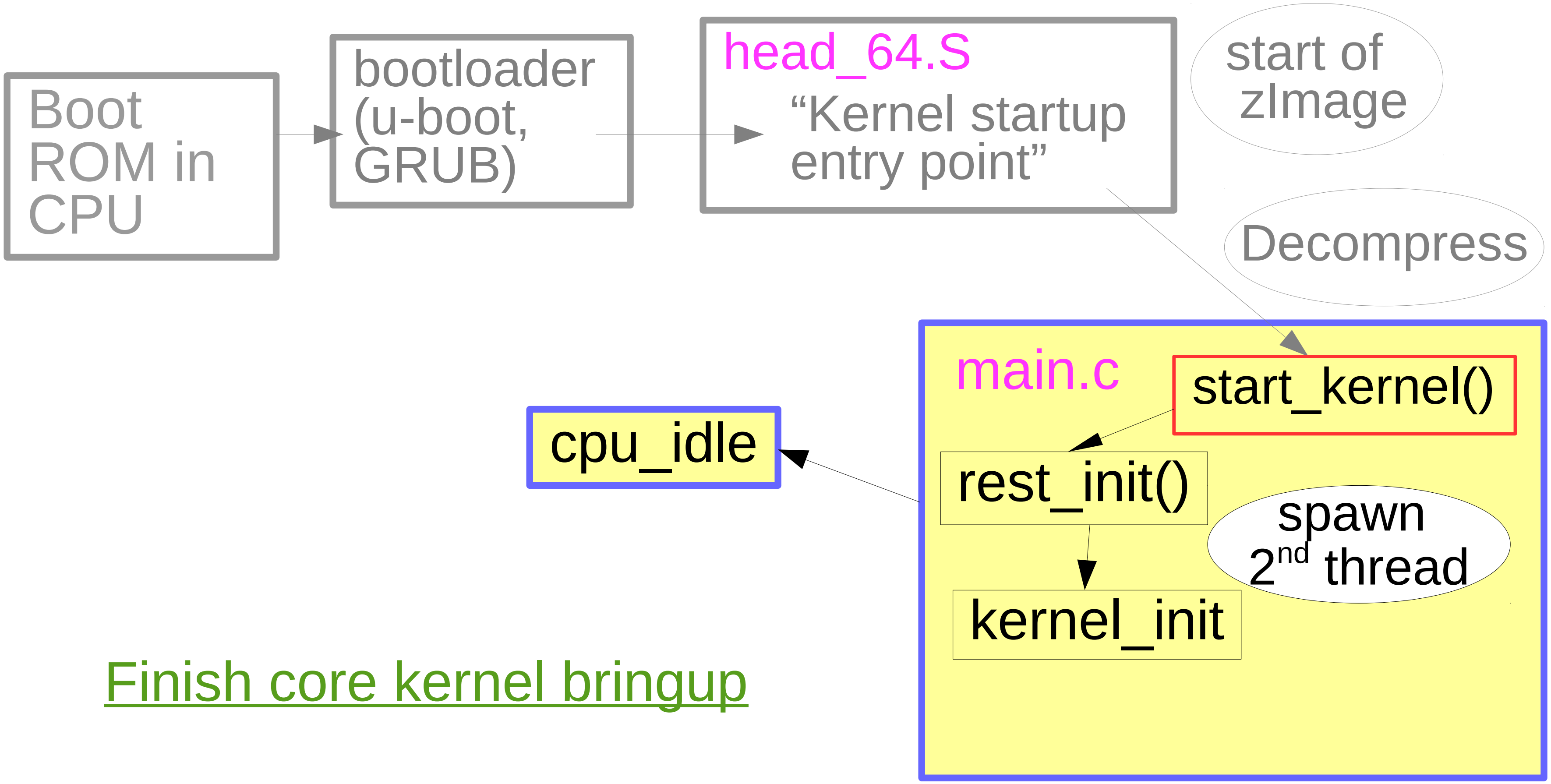
process the device-tree

setup page tables
and start virtual memory

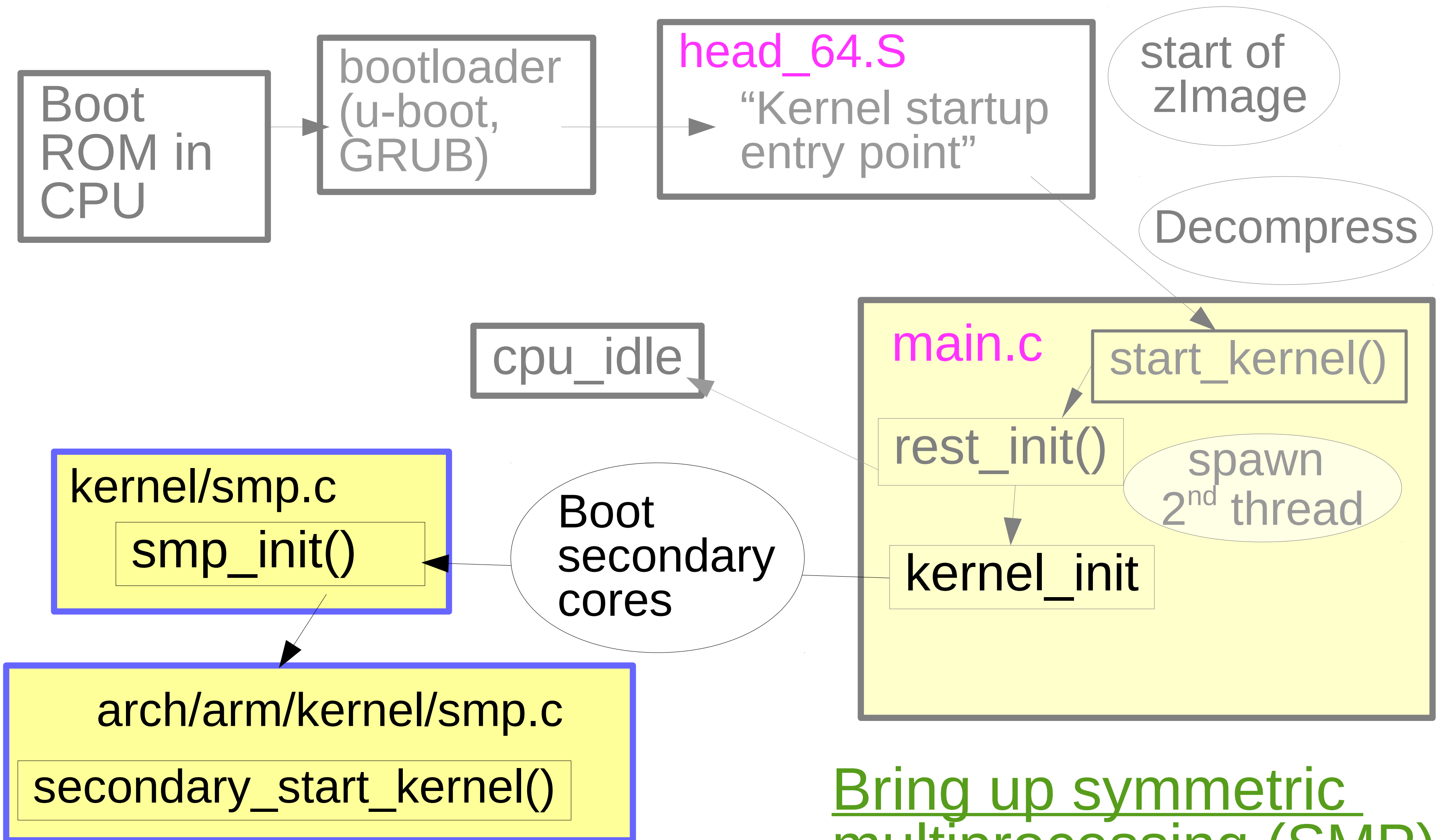
All timestamps before
are [0.000000]

start
userspace

All on
one core!



Finish core kernel bringup



Bring up symmetric multiprocessing (SMP)

Kernel boot via BCC

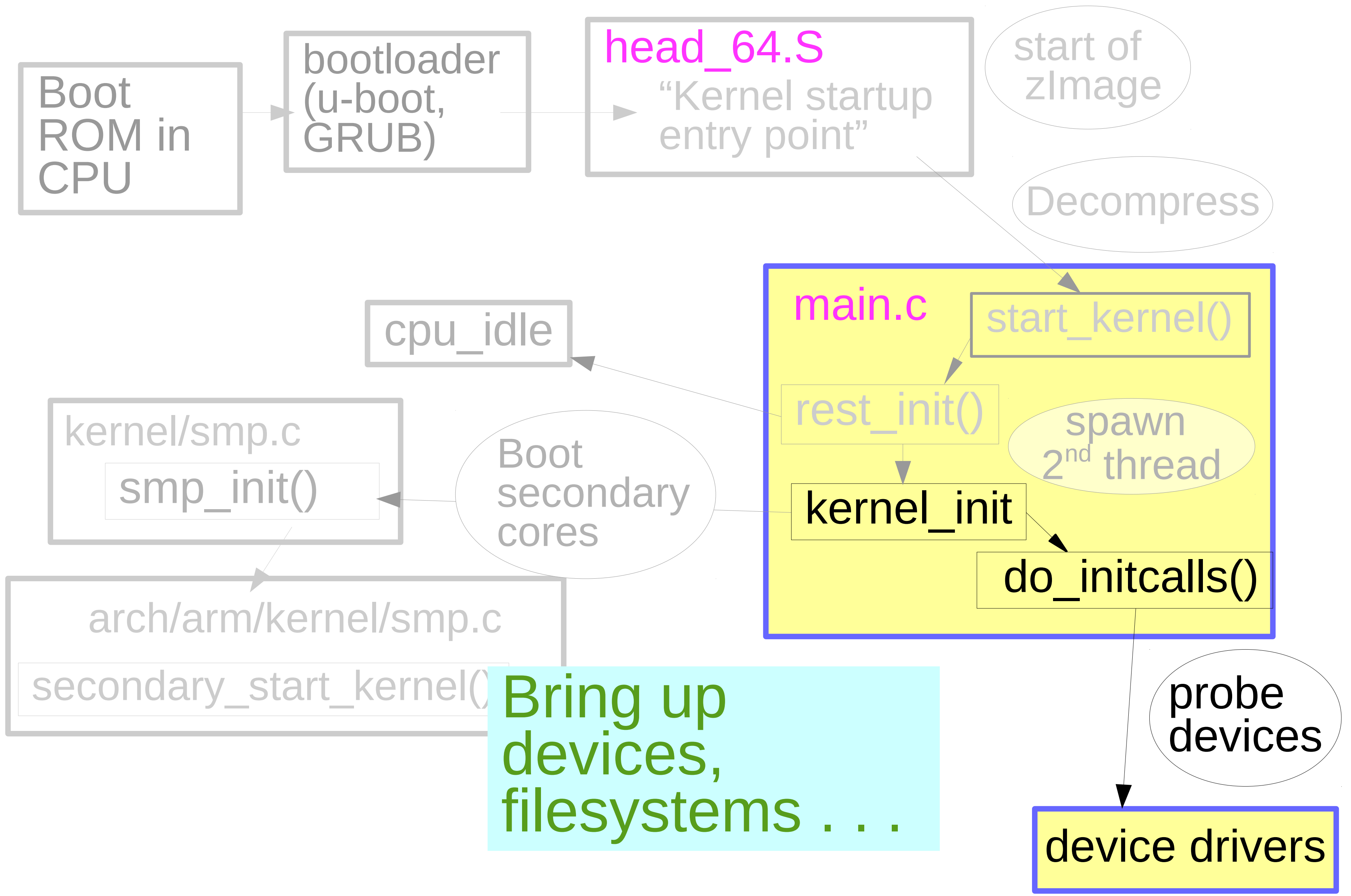
```
[alison@hildesheim tools (master)]$ sudo LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH ./offcputime.p
y -K
Tracing off-CPU time (us) of all threads by kernel stack... Hit Ctrl-C to end.
^C
    finish_task_switch
    schedule_idle
    do_idle
    cpu_startup_entry
    start_secondary
    verify_cpu
    -                 swapper/3 (0)
        199

    finish_task_switch
    schedule_idle
    do_idle
    cpu_startup_entry
    rest_init
    start_kernel
    x86_64_start_reservations
    x86_64_start_kernel
    verify_cpu
    -                 swapper/0 (0)
        263
```

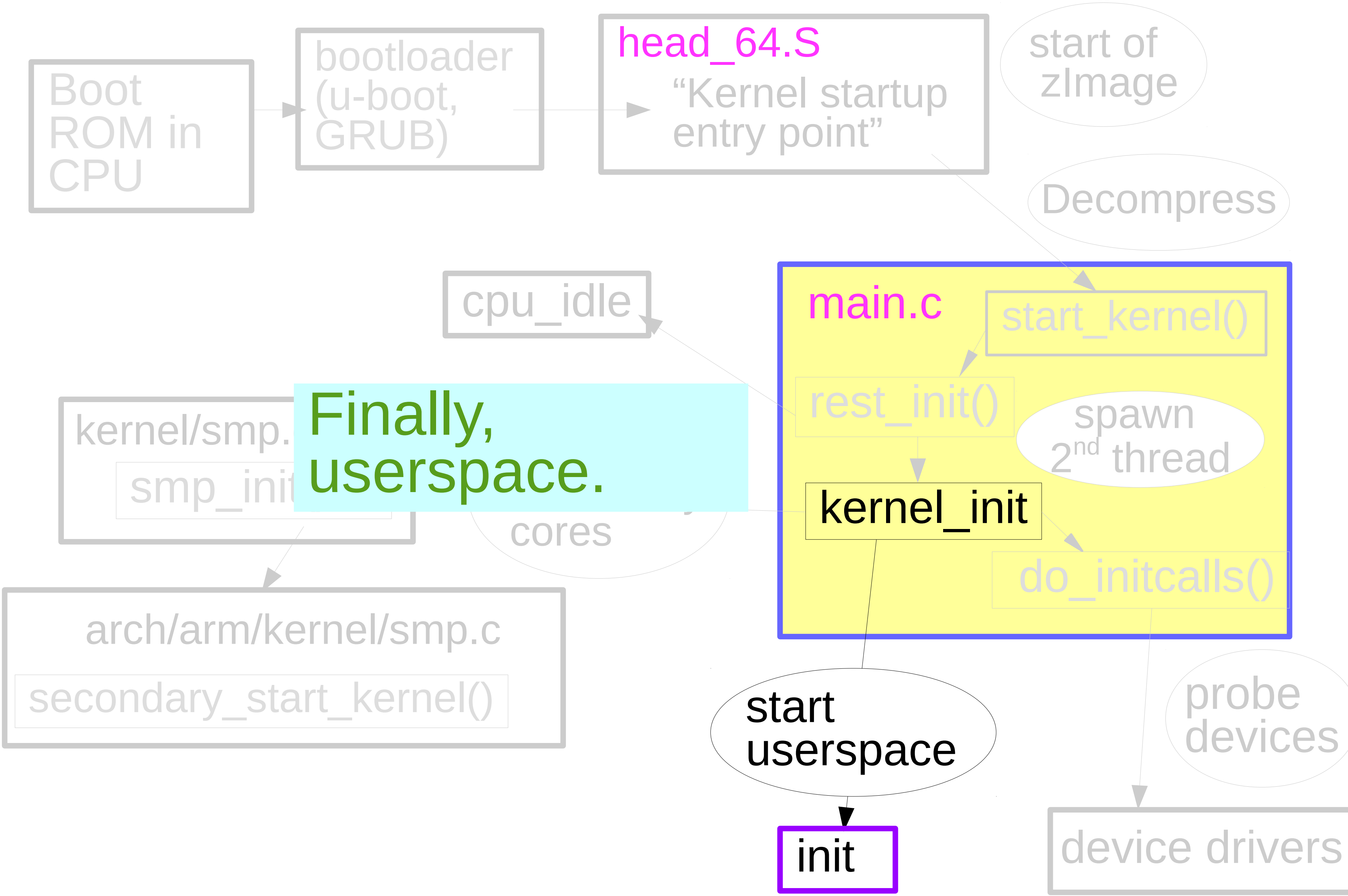
Stack for 2nd core

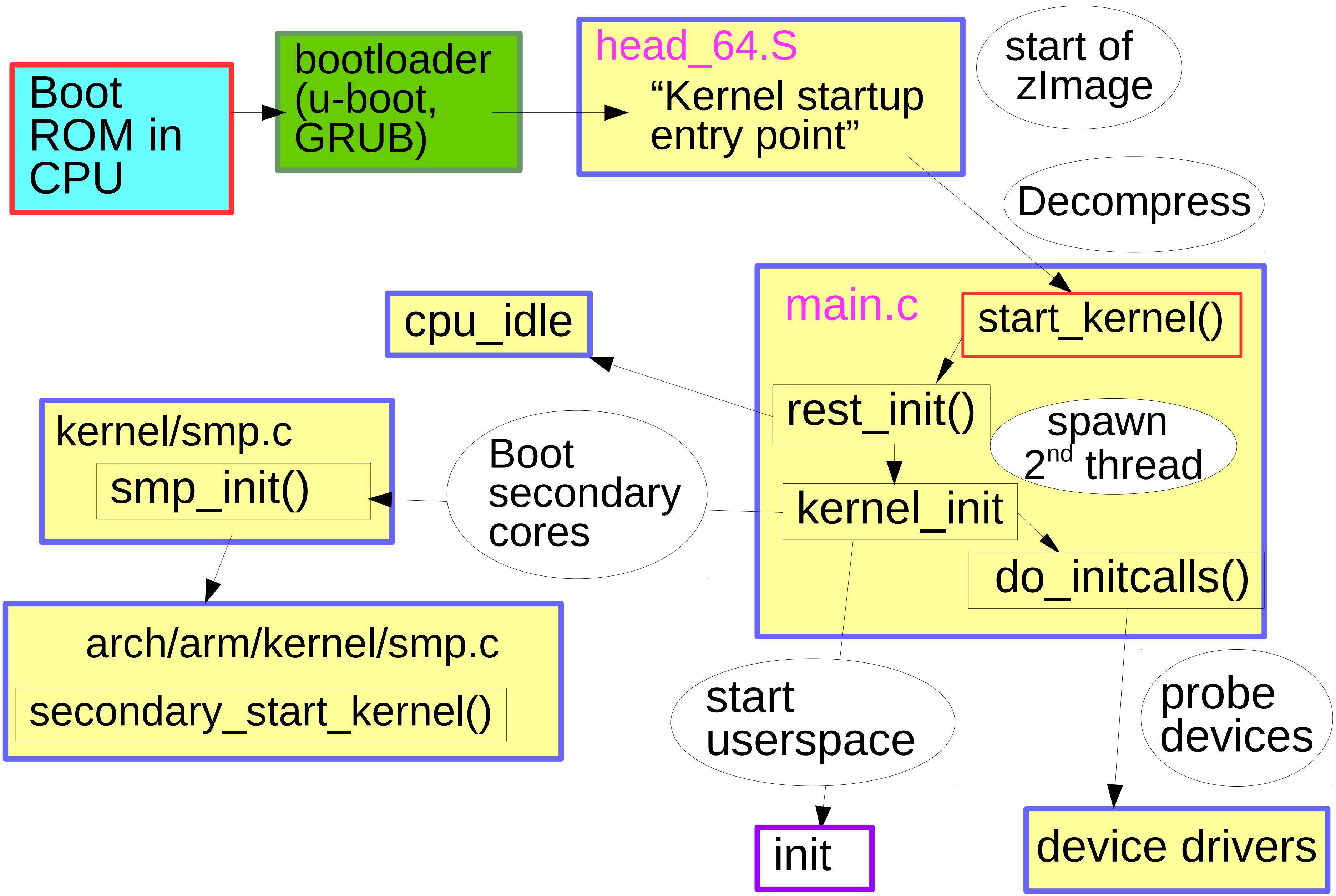
demo placeholder

Stack for CPU0

x86_64_start_kernel: head_64.S


Boot ROM in CPU
bootloader (u-boot, GRUB)
head_64.S
"Kernel startup entry point"
start of zImage
Decompress
main.c
start_kernel()
rest_init()
spawn 2nd thread
kernel_init
do_initcalls()
cpu_idle
kernel/smp.c
smp_init()
Boot secondary cores
arch/arm/kernel/smp.c
secondary_start_kernel()
Bring up devices, filesystems . . .
probe devices
device drivers


Boot ROM in CPU
bootloader (u-boot, GRUB)
head_64.S
"Kernel startup entry point"
start of zImage
Decompress
cpu_idle
main.c
start_kernel()
rest_init()
spawn 2nd thread
kernel_init
do_initcalls()
kernel/smp.
smp_init
Finally, userspace.
cores
arch/arm/kernel/smp.c
secondary_start_kernel()
start userspace
init
probe devices
device drivers


Boot ROM in CPU
bootloader (u-boot, GRUB)
head_64.S
"Kernel startup entry point"
start of zImage
Decompress
main.c
start_kernel()
rest_init()
spawn 2nd thread
kernel_init
do_initcalls()
cpu_idle
kernel/smp.c
smp_init()
Boot secondary cores
arch/arm/kernel/smp.c
secondary_start_kernel()
start userspace
init
probe devices
device drivers

Summary

- Practicing with u-boot sandbox is comparatively relaxing.
- Viewing the kernel as ELF helps to understand early boot.
- Several processors and SW components participate in boot.
- Until the scheduler and SMP start, the boot process is relatively simple.

Acknowledgements

- Big thanks to Joel Fernandes and Akkana Peck for suggestions.
- Shout-out to Linaro for making ARM so much easier than x86.

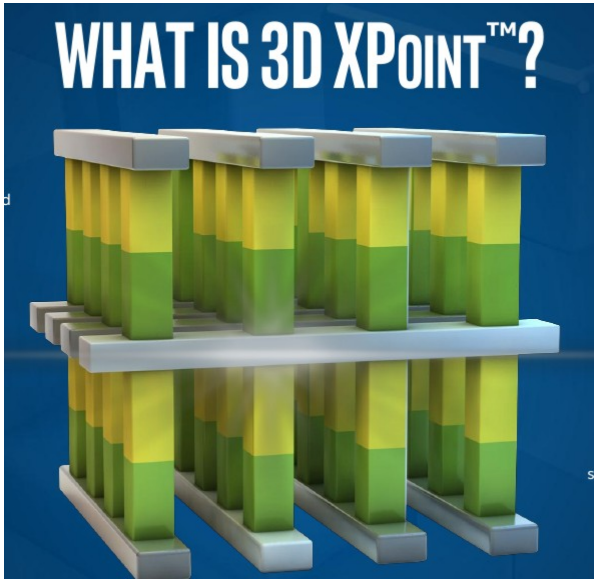
Major References

- *Embedded Linux Primer* by Chris Hallinan and Essential Linux Device Drivers by Sreekrishnan Venkateswaran (books)
- Booting ARM Linux by Russell King and THE LINUX/x86 BOOT PROTOCOL (Documentation/)
- Program startup process in userspace at linux-insides blog, Michael Kerrisk's TLPI (book)
- Matthew Garrett's comprehensive series on UEFI
- Status of Intel Management Engine on various laptops (Coreboot) and servers (FSF)
- Nov, 2017 Intel Management Engine exploits and vulnerability detection tool
- All about ACPI talk by Darren Hart, ELCE 2013, Arch Wiki on hacking ACPI tables
- 'apt-get install debian-kernel-handbook'; GDB docs chapter 8

Cold-boot may become rare

Source: Micron

Specs:
ArsTechnica

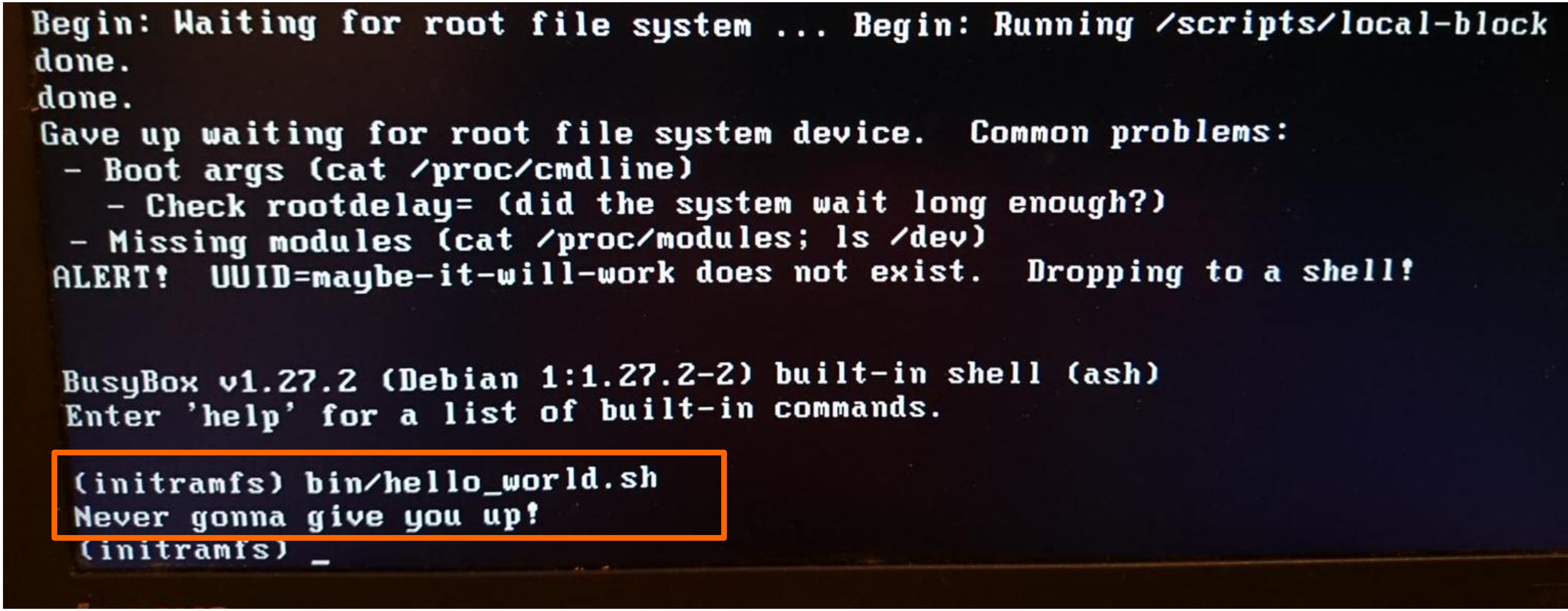


AKA,
'Optane'
by Intel

- Non-volatile *RAM* → suspend even for brief inactivity.
- Minimal diff between 'suspend' and 'hibernate'?
- Linux drivers: Matthew Wilcox, XIP →DAX

About Initrds

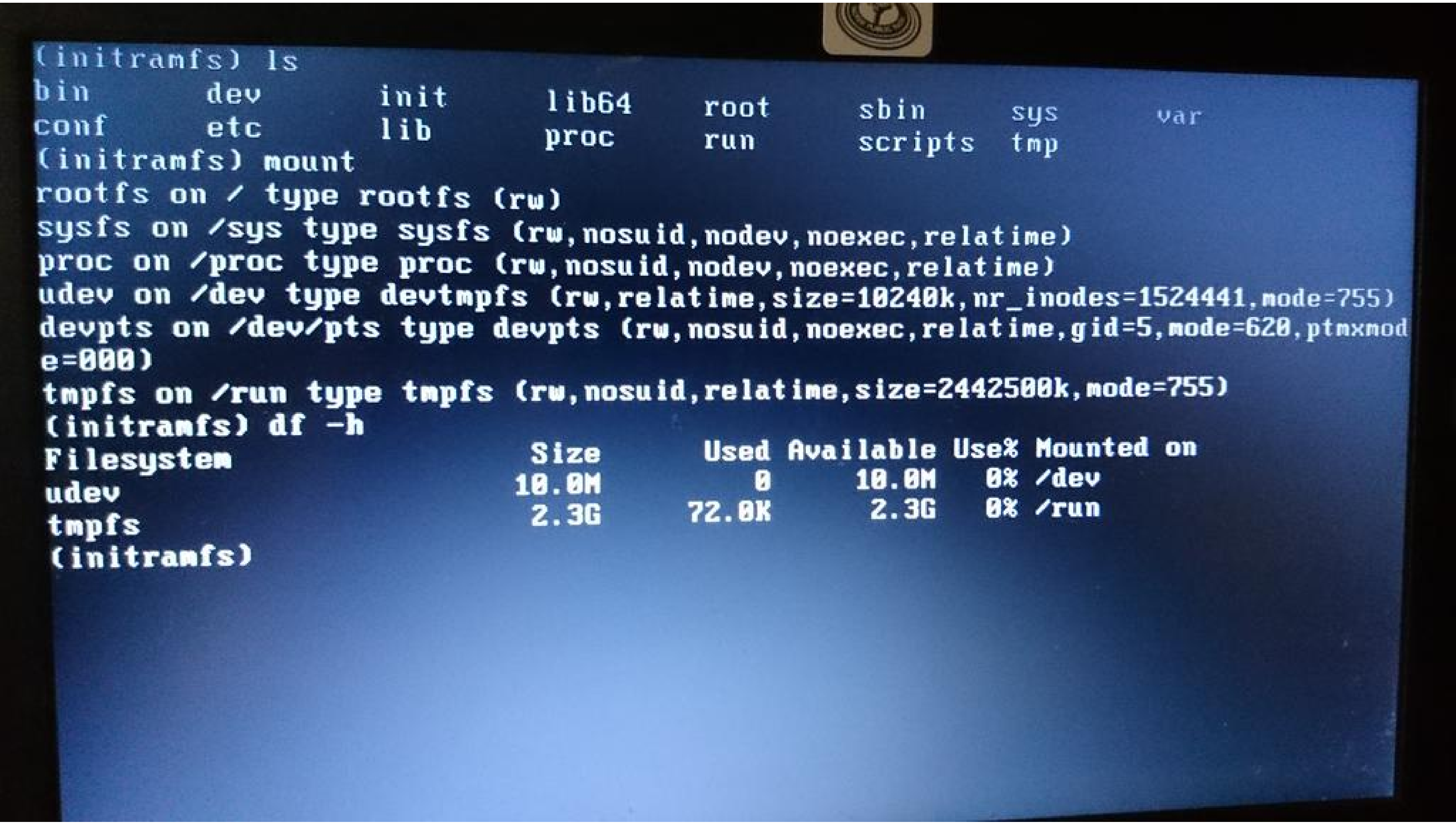
Booting into Rescue Shell



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.

Exploring initramfs



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
```

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.

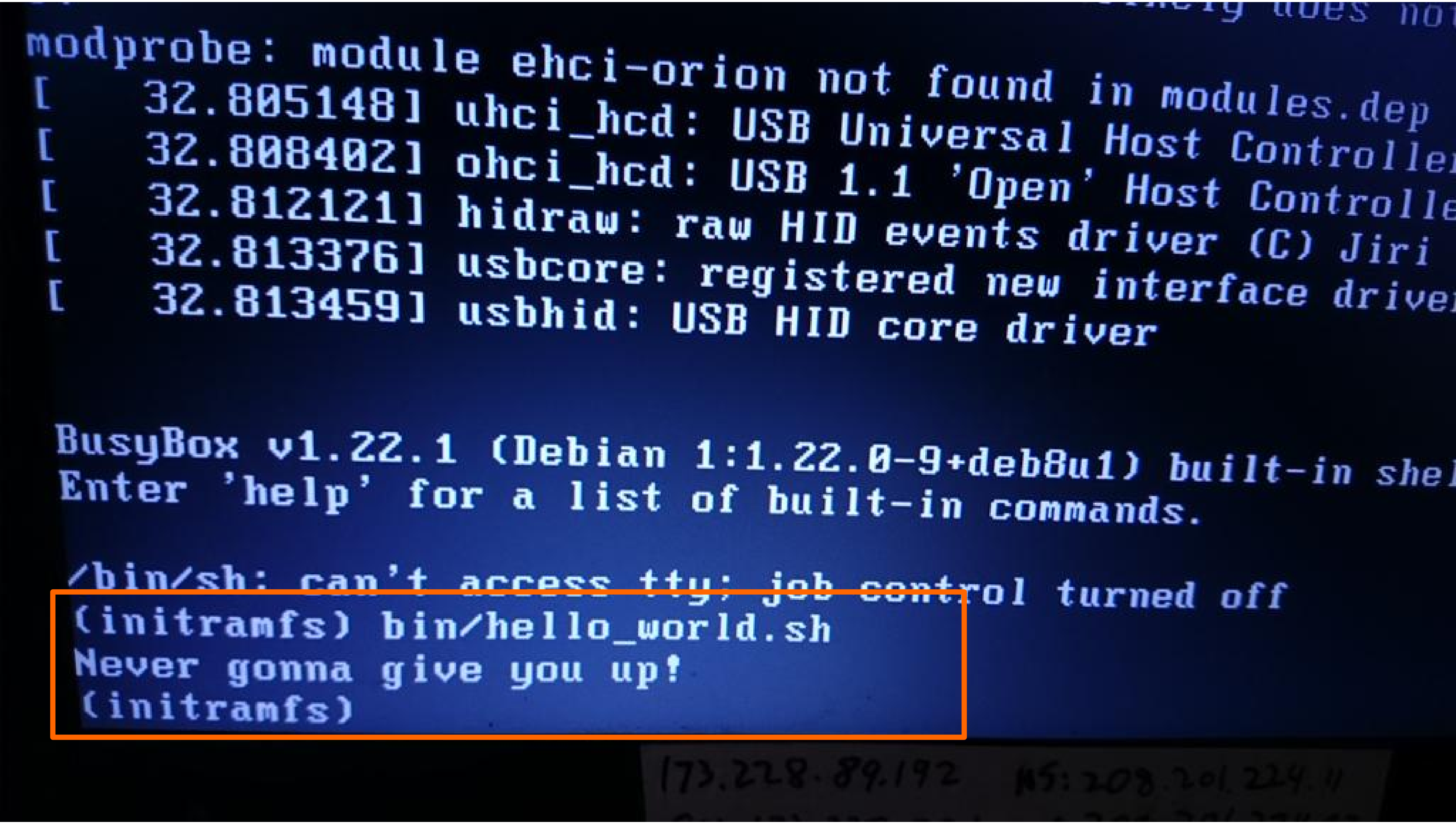
- Read 'man busybox'.
- 'help | grep' works in busybox.
- Look in /bin and /sbin. There's fsck!!
- You have sed and vi (but not emacs ;-( )
- Type 'reboot -f' or 'exit' when you are bored.

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.

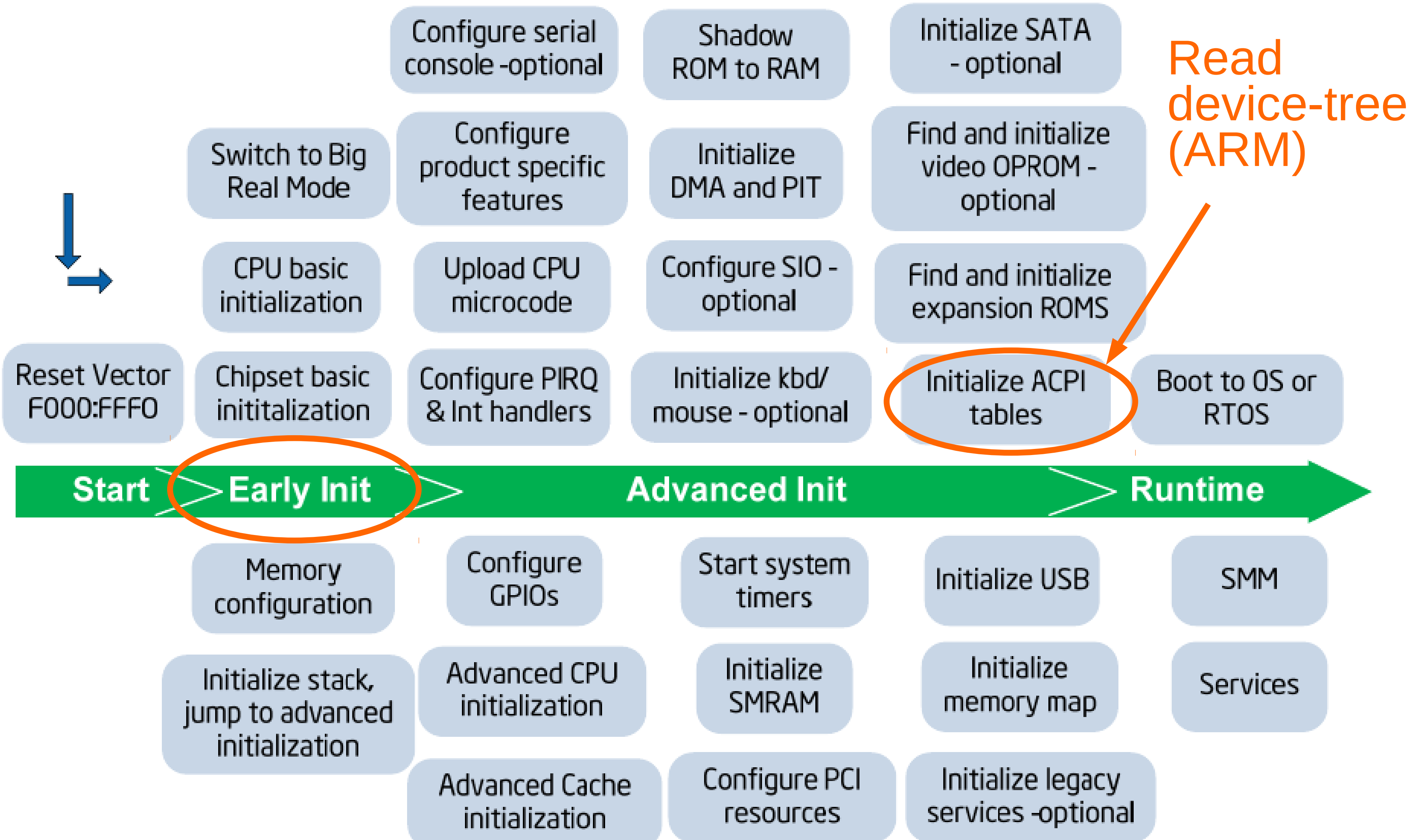
The magnificent result!



```
[alison@hildesheim LCA]$ sudo Intel_IME_vulnerability_detection/intel_sa00086.py
INTEL-SA-00086 Detection Tool
Copyright(C) 2017, Intel Corporation, All rights reserved

Application Version: 1.0.0.146
Scan date: 2017-12-17 02:48:44 GMT

*** Host Computer Information ***
Name: hildesheim
Manufacturer: LENOVO
Model: 20AL009CUS
Processor Name: Intel(R) Core(TM) i7-4600U CPU @ 2.10GHz
OS Version: debian buster/sid  (4.13.13)

*** Intel(R) ME Information ***
Engine: Intel(R) Management Engine
Version: 9.5.22.1760
SVN: 0

*** Risk Assessment ***
Based on the analysis performed by this tool: This system is vulnerable.
```

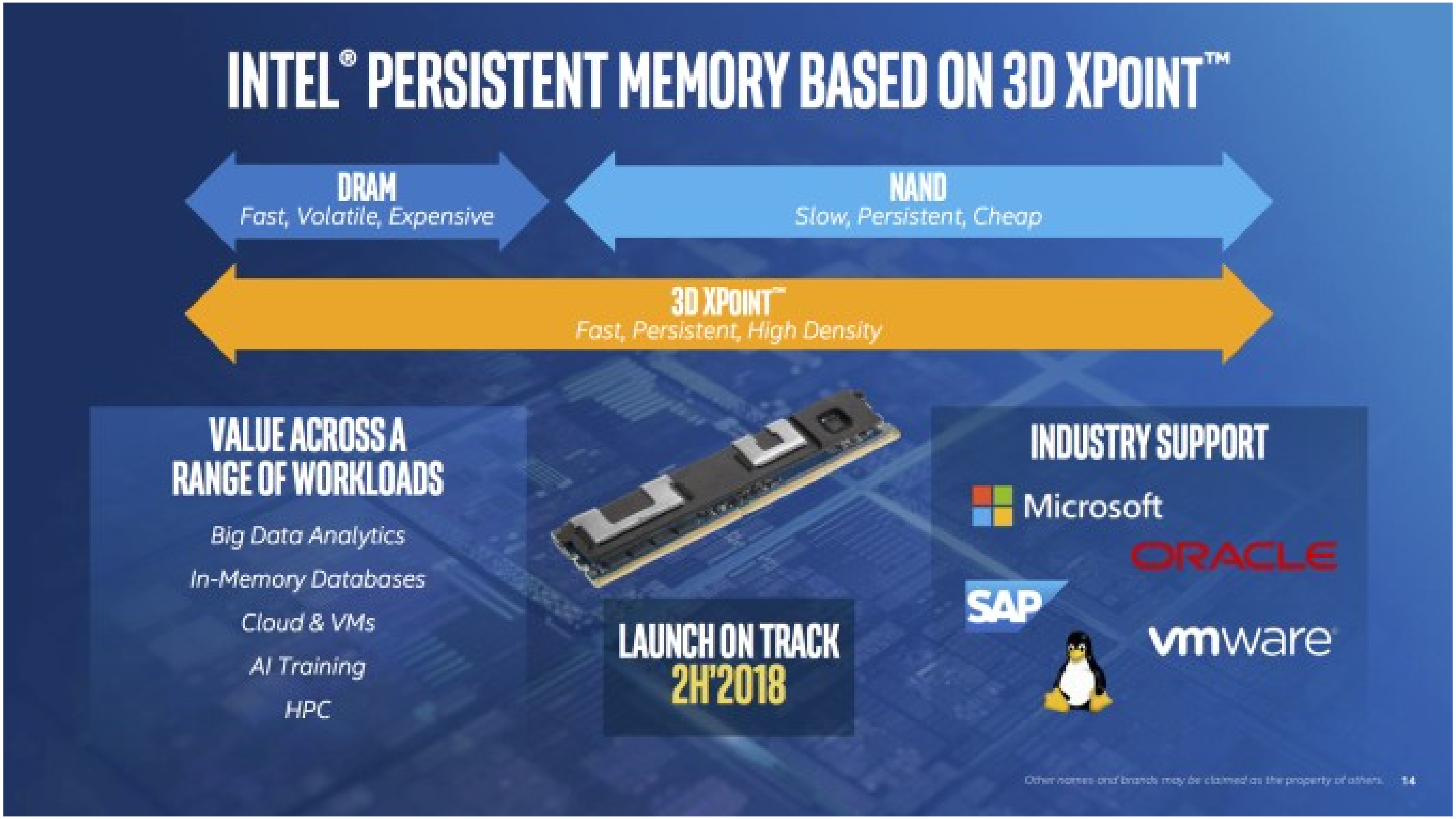
The Lenovo laptop on which the slides were created has known IME vulnerabilities described by unpatched CVEs. This has nothing to do with Meltdown and Spectre.

Bootloaders according to Intel



Source:
Anandtech

Coming soon to a system near you



Investigating your laptop's PCH

- Try:

  lsmod | grep pch

- Try:

  find /lib/modules/$(uname -r)/ -name “*pch*”

- Then (for example):

```
[alison@hildesheim LCA]$ modinfo pch_udc
filename:       /lib/modules/4.13.0-1-amd64/kernel/drivers/usb/gadget/udc/pch_udc.ko
license:        GPL
author:         LAPIS Semiconductor, <tomoya-linux@dsn.lapis-semi.com>
description:    Intel EG20T USB Device Controller
```

EG20T = Intel Topcliff PCH

Why bootloaders have two parts

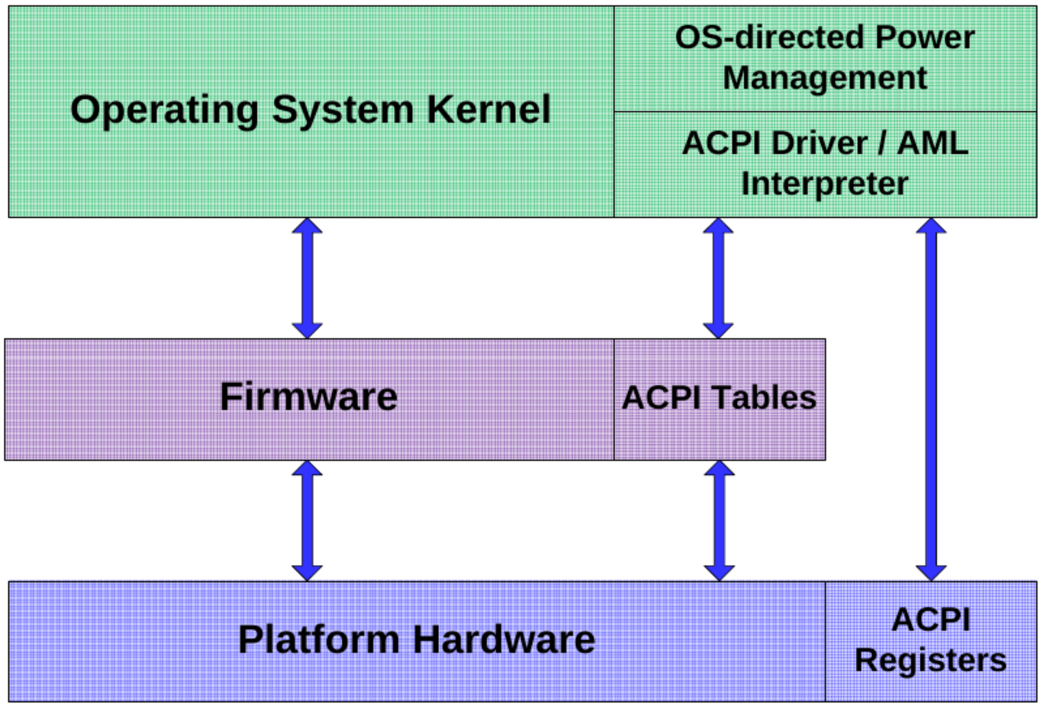
- ARM: “SPL”, “XLoader” or “MLO” in addition to u-boot.img.
- **Problem**: DRAM controller must be initialized.
- **Solution**: load into SRAM ('OCRAM' in i.MX6, 'l2ram' for TI).
  - *Why this works*: SRAM (and pNOR) are mapped memory.
- **Problem**: SRAM is little! (256K on i.MX6, 2 MB on DRA7x).
-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

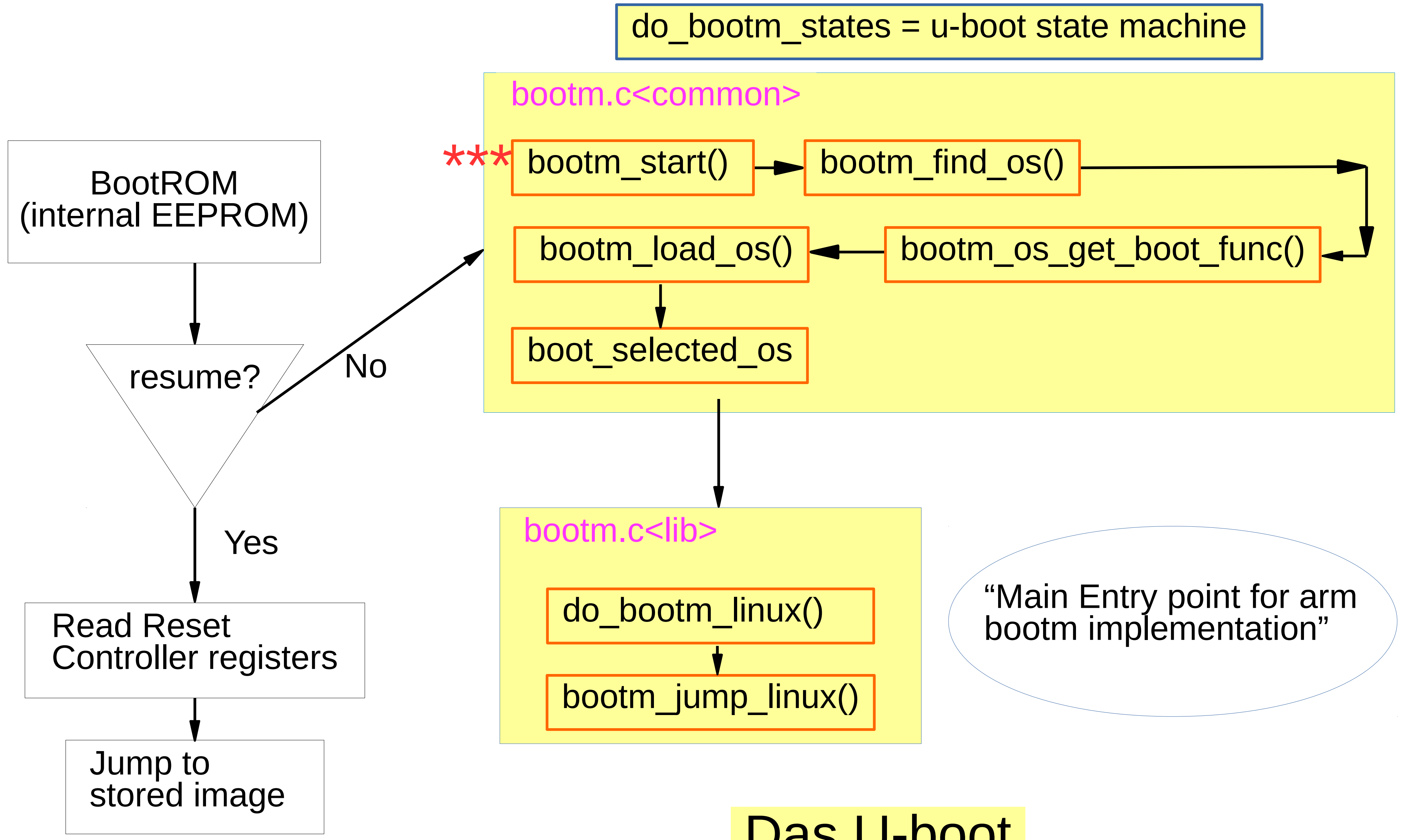
Warm vs. power-on reset

	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

Advanced Configuration and Power Interface

Source: Intel




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

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

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:

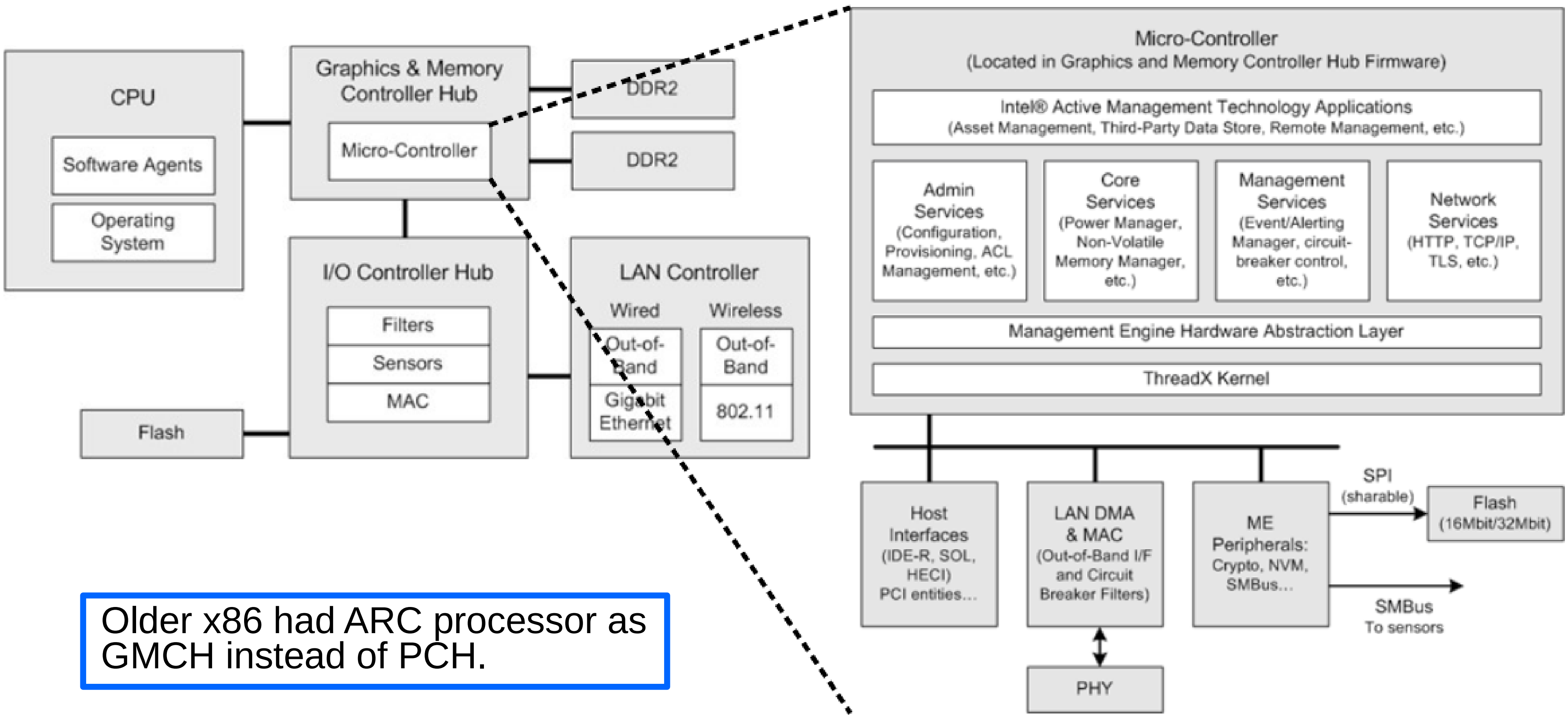
1. Setup and initialise the RAM.
2. Initialise one serial port.
3. Detect the machine type.
4. Setup the kernel ~~tagged list~~. device-tree
5. Load initramfs.
6. Call the kernel image.

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

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.

ME: High-level overview



Older x86 had ARC processor as GMCH instead of PCH.

Credit: Intel 2009

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