Chrome OS
Open Source Firmware

Duncan Laurie

Linux.conf.au 2013

chrome

bit.ly/chromefw

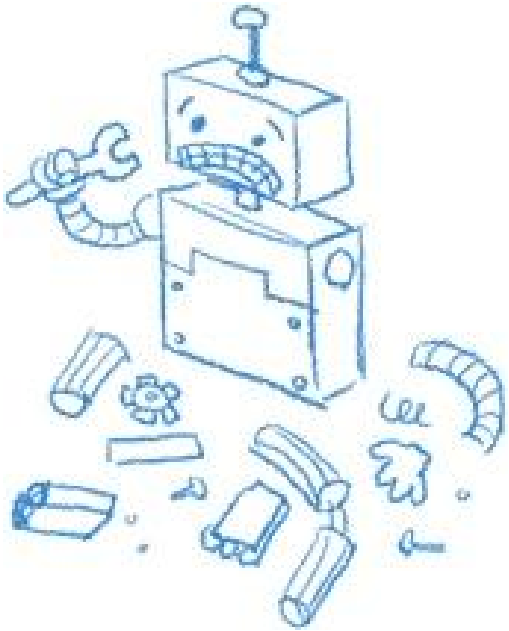
What is Chrome OS?

- Open source project based on Gentoo Linux
- Targeted to specific devices
- Security is paramount
  - Verified Boot
  - Hardened Linux
  - Pwnium 3: March 7, 2013

Why Invest In Firmware?

Knowledge of the Platform

- Firmware is hard
- Bugs will be found
- Time is money



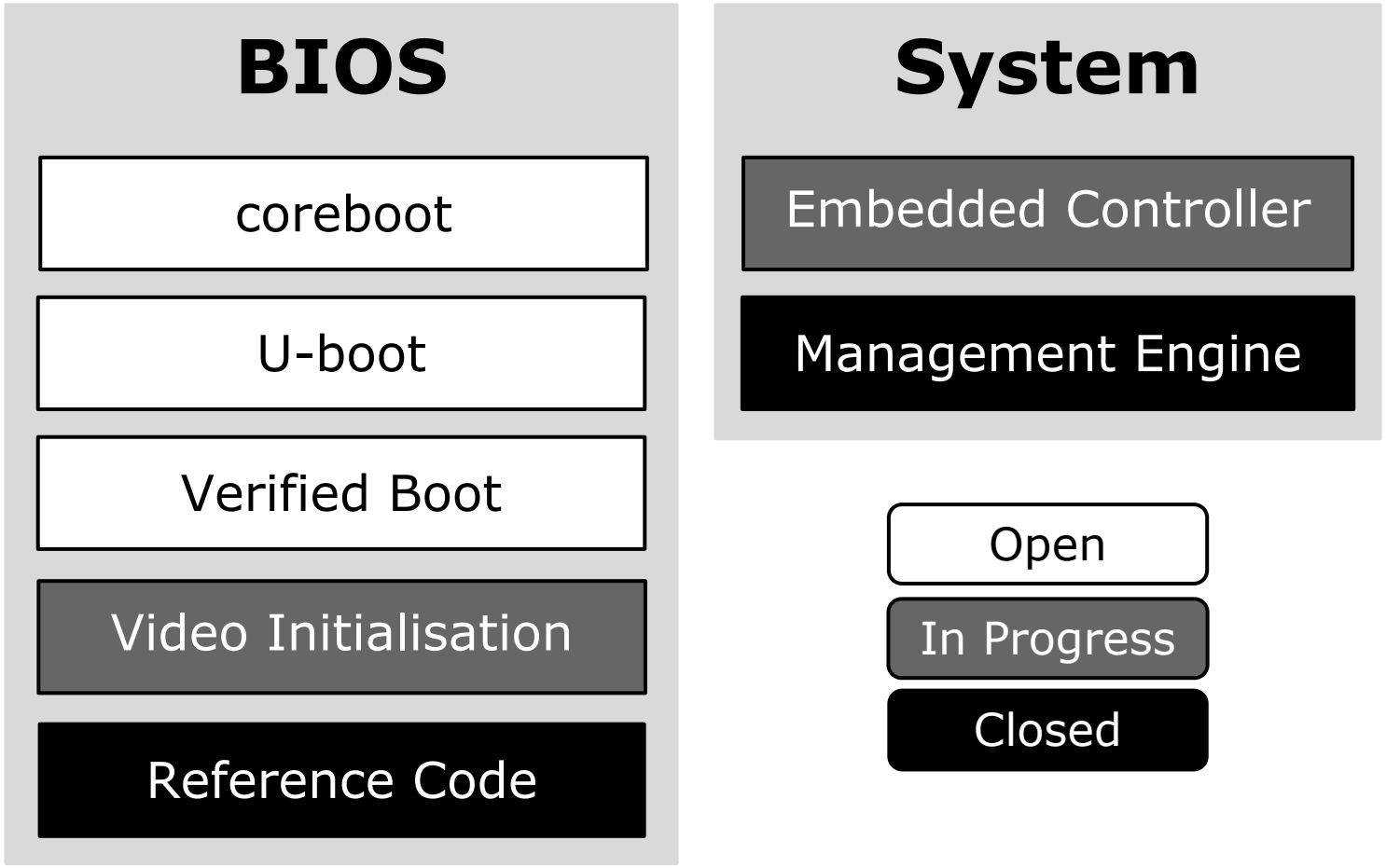
Control of the Platform

- Consistent behavior across architectures
- Maximize power and performance
- Flexible Firmware/OS interfaces

Firmware Components

Chrome OS Verified Boot

Firmware Components (x86)



coreboot

- GPLv2 BIOS replacement
  - Started as LinuxBIOS in 1999 by Ron Minnich
  - Renamed to coreboot in 2008 by Stefan Reinauer
- Mostly C, some Assembly and ACPI
- High-level organization similar to EFI
- Payload instead of fixed bootloader

coreboot Stages

- Bootblock
  - Prepare Cache-as-RAM and Flash access
- ROM Stage
  - Memory and early chipset init
- RAM Stage
  - Device enumeration and resource assignment
  - ACPI Table creation
  - SMM Handler
- Payload

U-boot

- GPLv2 ARM firmware base
- Chrome OS Verified Boot integration
- x86 support as a Coreboot payload
  - u-boot.git/board/chromebook-x86

Reference Code Binary

Closed

- EFI PEI wrapper produces standalone binary
- Executed by coreboot to initialize memory
- Distributed as binary via coreboot.org
- Intel *Firmware Support Package*
  - bit.ly/intelfsp

Management Engine

Closed

- Microcontroller integrated in Intel chipset
- Required in every Intel system
- Reduced control over the platform
- Increased complexity in host firmware

Video Initialisation

- Not needed for normal Chrome OS boot
- Firmware graphics needed for Recovery
- Extract setup from existing kernel driver
  - Coccinelle Semantic Patch Language (SmPL)
  - i915tool.googlecode.com

Embedded Controller

- Microcontroller found in notebook devices
- Responsible for key platform tasks
  - Power sequencing
  - PS/2 Keyboard & Touchpad
  - Fan Control
  - Battery charging
  - Lid and power button control
  - Device power control
  - System LED behavior

x86 Embedded Controller

- LPC bus communication
- Firmware Interface
  - Custom protocol
  - System Management Interrupt (SMI)
- ACPI Interface
  - I/O ports 0x62 and 0x66
  - System Control Interrupt (SCI)
  - EC RAM provides 255 bytes of state
  - EC Query Codes provide 255 events

ARM Embedded Controller

- I2C or SPI bus communication
- No common interface standards
- Firmware and Kernel can share interface

Chrome EC

- Embedded Controllers are vital but closed
- Chrome EC is open!
  - chromiumos/platform/ec.git
  - Verified update is part of host firmware
- Samsung ARM Chromebook
  - ST Micro STM32 Cortex-M4
- Chromebook Pixel
  - Texas Instruments Stellaris Cortex-M4

Firmware Features

Verified Boot - Firmware

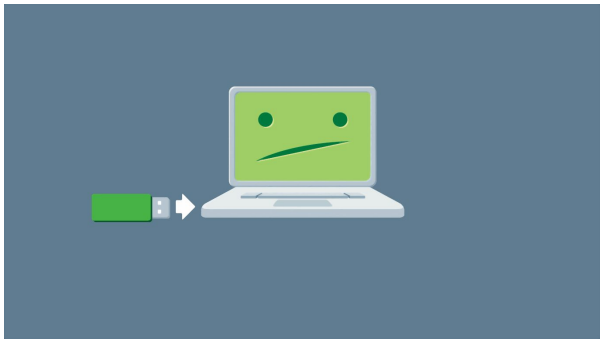
- Design is similar to UEFI Secure Boot
- Root Of Trust is Read-Only firmware
- RO firmware verifies signed RW firmware
- Firmware verifies signed kernel from disk
- Reference implementation available
  - chromiumos/platform/vboot_reference.git

Verified Boot - Kernel

- Root filesystem is RO image
  - Hash each block in the image
  - Block hashes are bundled and structured in a tree
  - Hash of first block stored with kernel and signed
- Bootloader passes first block hash to kernel
- Blocks are hashed and checked against tree
- Transparent block device
  - linux.git/drivers/md/dm-verity.c

Chrome OS Recovery Mode

- Read-only firmware used to boot signed USB
- Can be initiated by the system or user
- User must assert physical presence

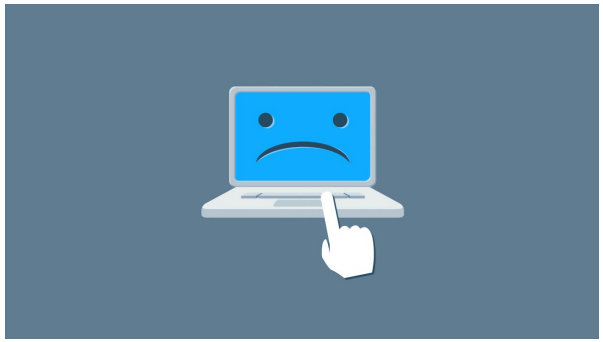


chrome

bit.ly/chromefw

Chrome OS Developer Mode

- Jailbreak mode built into every device
- Hardware switch or transition via recovery
- Transition will erase local state
- Enables root shell in Chrome OS
- Boot self-signed images



chrome

bit.ly/chromefw

Event Logging

- Persistent log of system events
- Based on SMBIOS System Event Log
  - coreboot.git/drivers/elog/elog.c
- Kernel/sysfs interface to write events
  - linux.git/drivers/firmware/google/gsmi.c
- Userland can find and parse the log
  - chromiumos/platform/mosys.git/lib/smbios/eventlog.c

Event Log Example

```
12 | 2013-01-15 10:47:43 | ACPI Wake     | S5
13 | 2013-01-15 10:47:43 | EC Event      | Lid Open
14 | 2013-01-15 10:47:43 | System boot   | 142
15 | 2013-01-15 11:51:42 | ACPI Enter    | S3
16 | 2013-01-15 21:05:37 | ACPI Wake     | S3
17 | 2013-01-15 21:05:37 | Wake Source   | GPIO | 11
18 | 2013-01-15 21:05:38 | Kernel Event | Oops
19 | 2013-01-15 21:05:38 | Kernel Event | Panic
20 | 2013-01-15 21:05:39 | System boot   | 143
```

10:47 - Power on because lid was opened
11:51 - System is suspended
21:05 - Wake from suspend due to GPIO 11 (Touchpad)
21:05 - Kernel oops+panic on resume

Memory Console

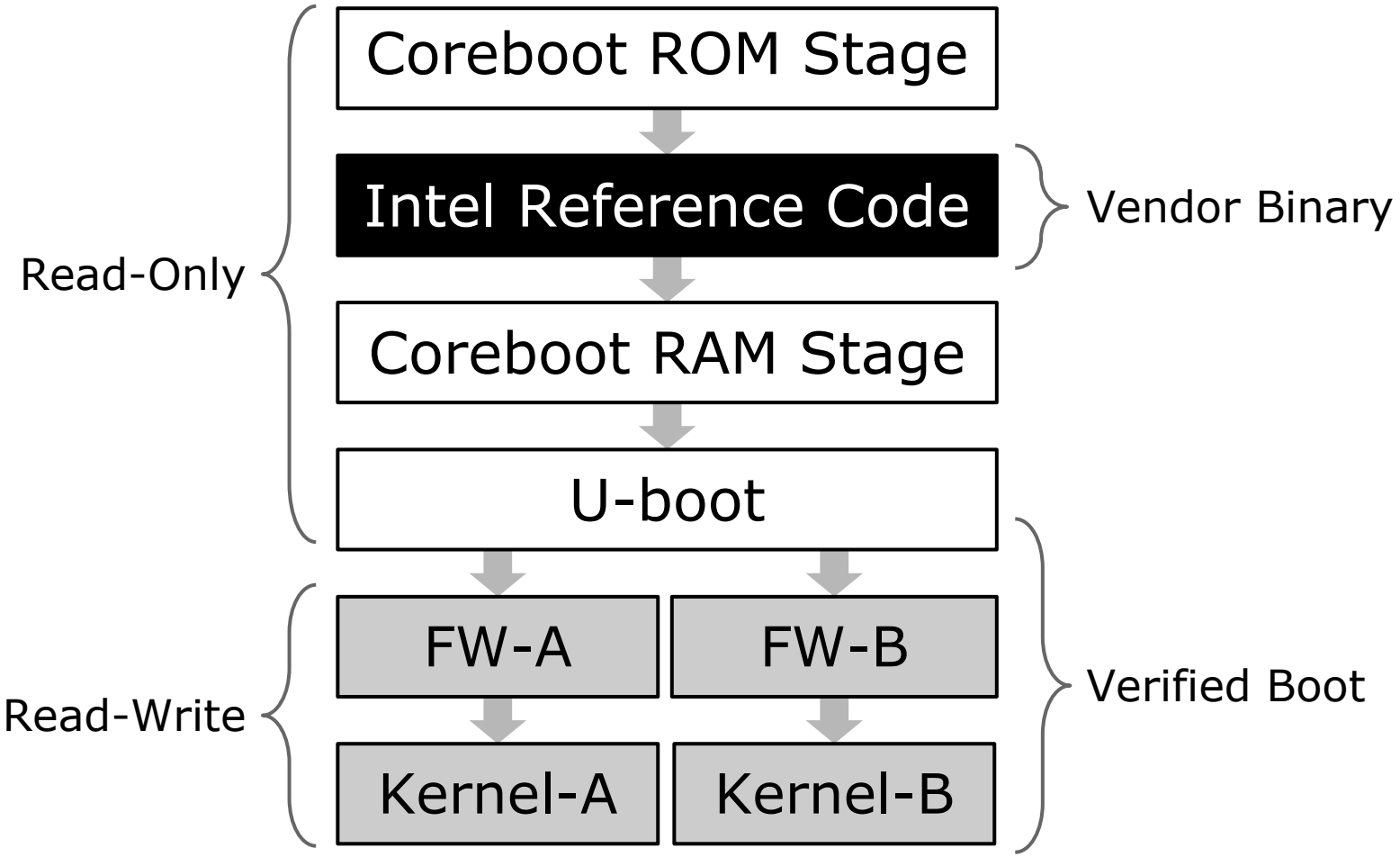
- Serial is essential for development & debug
- Console output saved to memory buffer
  - coreboot.git/src/lib/cbmem_console.c
- Log exported at /sys/firmware/log
  - linux.git/drivers/firmware/google/memconsole.c

ACPI

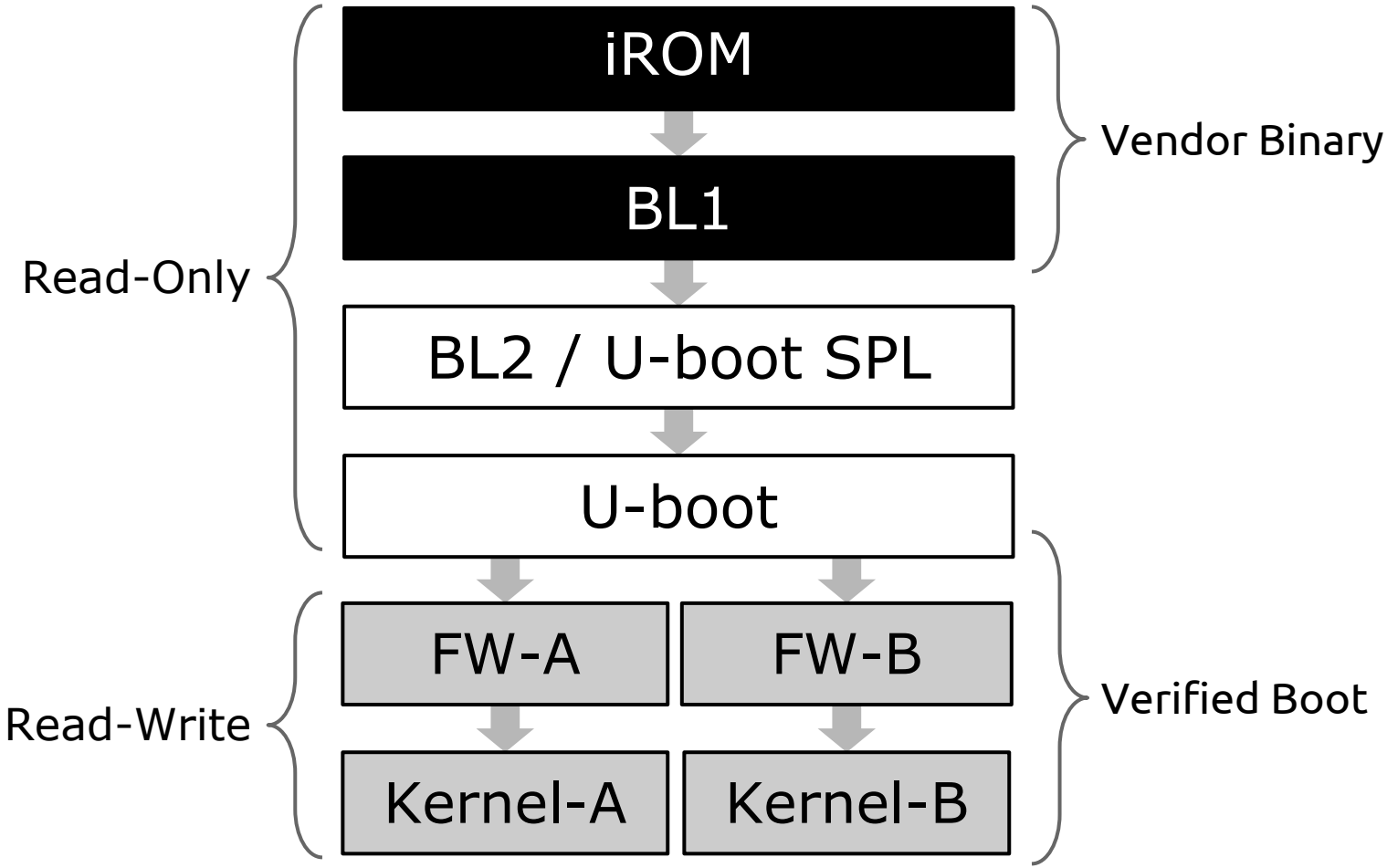
- Advanced Configuration and Power Interface
- Firmware code executed in OS context
- ACPI Source Language (ASL)
  - Every Variable/Method is exactly 4 characters
  - Compiles into AML bytecode
- System Control Interrupt (SCI)

Chrome OS Boot Process

Chrome OS Boot (Intel)



Chrome OS Boot (Exynos)



Exynos 5250 Firmware

- iROM
  - On-chip ROM
  - Verify and load BL1
- BL1: pre-boot firmware
  - 8KB signed binary provided by Samsung
  - Chip-specific initialization
  - Verify and load BL2

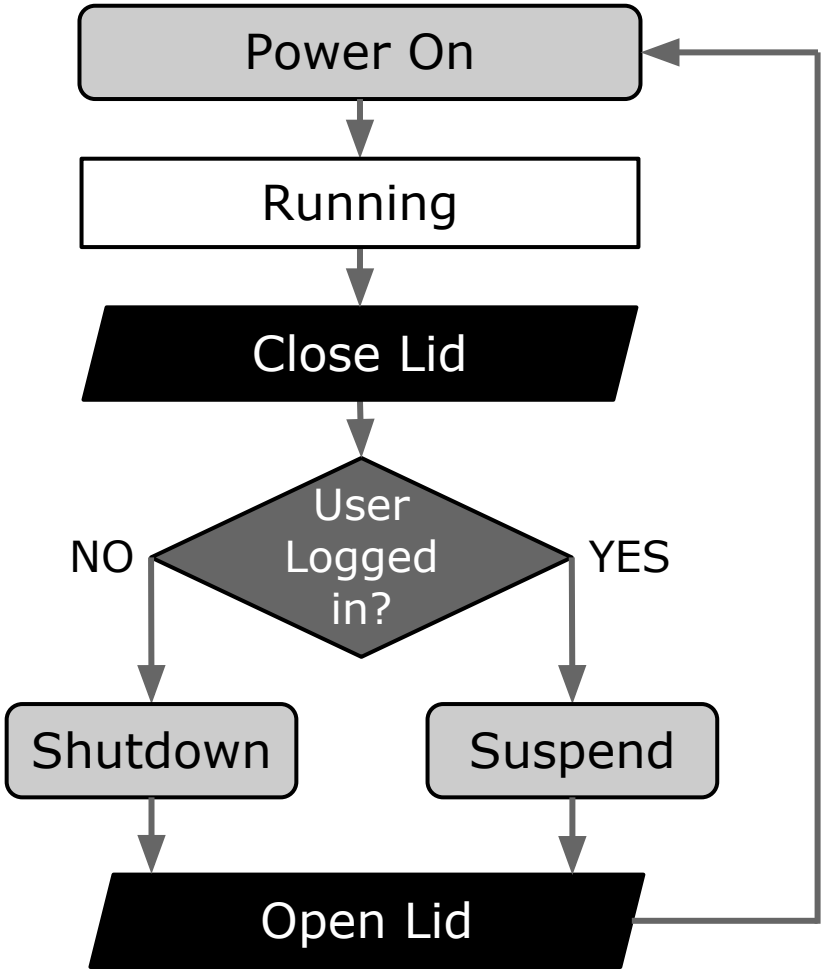
Exynos 5250 Firmware

- BL2: U-boot Secondary Program Loader
  - ~16KB for platform-specific initialization
  - Setup memory and load full U-boot
- U-boot
  - Integrated with Chrome OS Verified Boot
  - (optional) Verify and execute read-write U-boot
  - Verify and and boot kernel

Example

Tying it all together...

Chrome OS Laptop Lid Policy



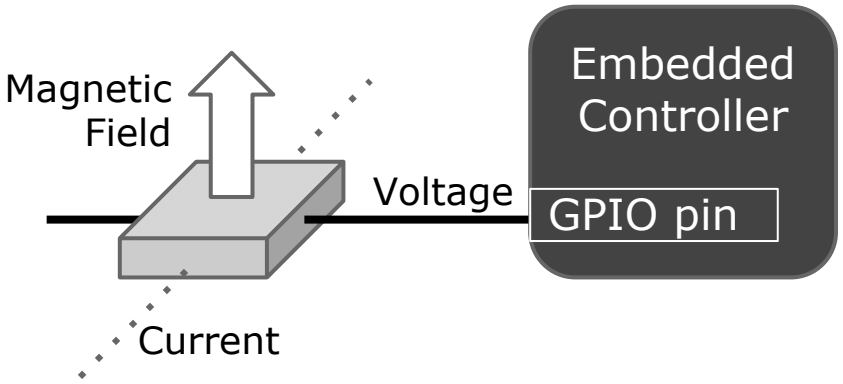
chrome

bit.ly/chromefw

Lid Policy Implementation

- Must be consistent across Firmware and OS
- Where does Lid State come from?
- How to get notified of state change?

Laptop Lid Switch




```
Device (\_SB.LID)
{
     // ID for Lid Device
     Name (_HID, EISAID ("PNP0C0D"))
     Name (_UID, 1)

     // Method for Lid Status
     Method (_LID) {
          Return (\_SB.PCI.LPC.EC.LIDS)
     }
}
```

Hall Effect Sensor

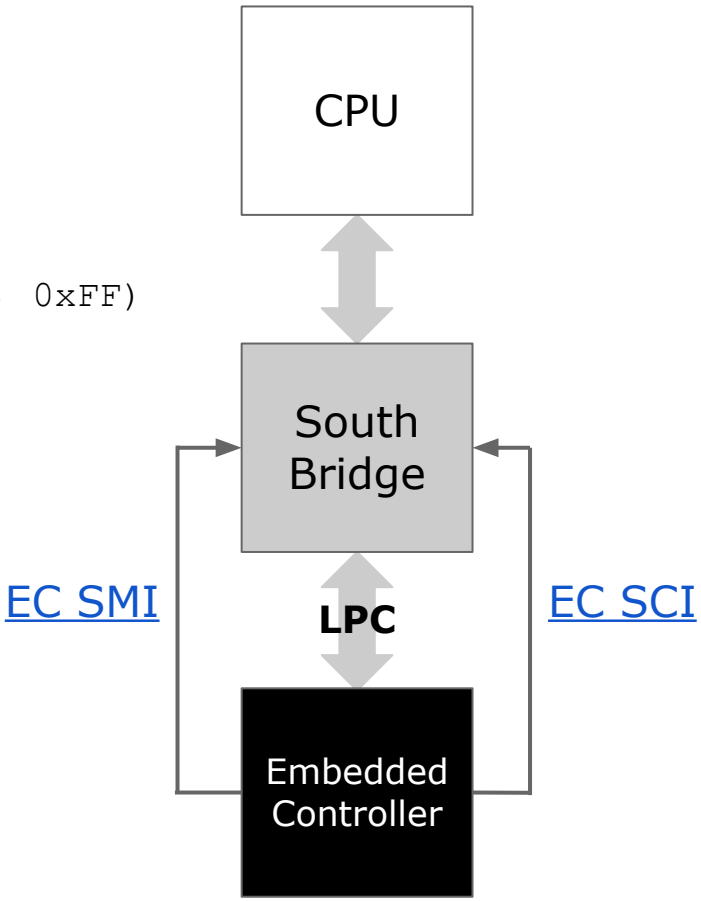
If Current is constant, then Voltage varies directly with the Magnetic Field

ACPI Lid Device

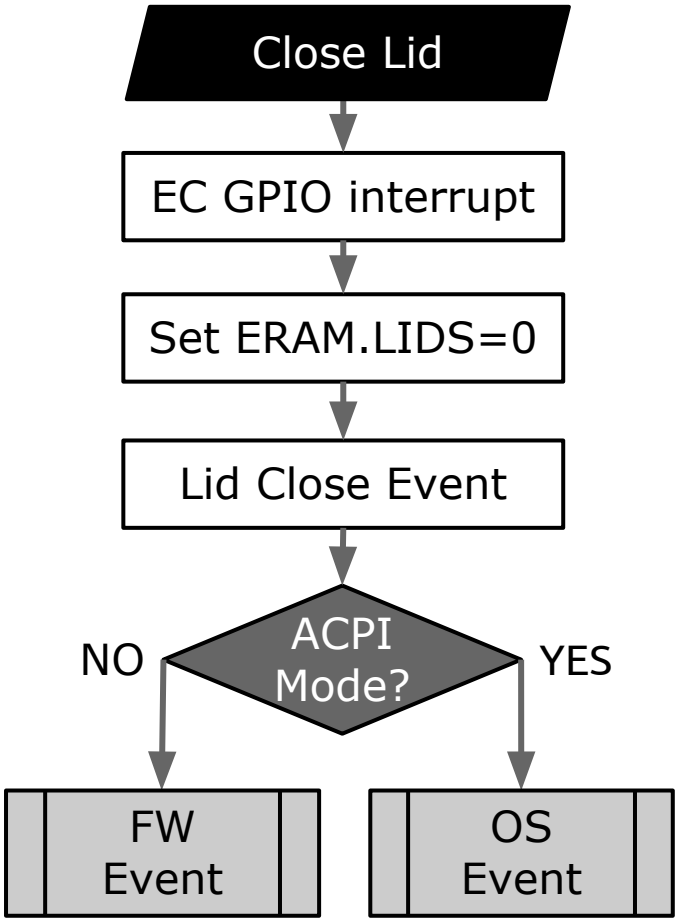
Lid status is read from Embedded Controller when `\_SB.LID._LID()` is called

ACPI Embedded Controller

```
Device (\_SB.PCI.LPC.EC)
{
  Name (_HID, EISAID ("PNP0C09"))
  Name (_UID, 1)
  Name (_GPE, Add(EC_SCI_GPIO, 16))

  // Memory Map exported by EC
  OperationRegion (ERAM, EmbeddedControl, 0x00, 0xFF)
  Field (ERAM, ByteAcc, Lock, Preserve) {
    Offset (0x83),
    LIDS, 1, // Lid Switch State = 0x83[0]
  }

  // LPC I/O port interface
  Name (_CRS, ResourceTemplate() {
    IO (Decode16, 0x62, 0x62, 0, 1)
    IO (Decode16, 0x66, 0x66, 0, 1)
  })

  // Lid Close Event = 0x5E
  Method (_Q5E) {
    Notify (\_SB.LID, 0x80)
  }
}
```



Lid Close: EC Actions



Lid Close Event

Firmware Event Handling

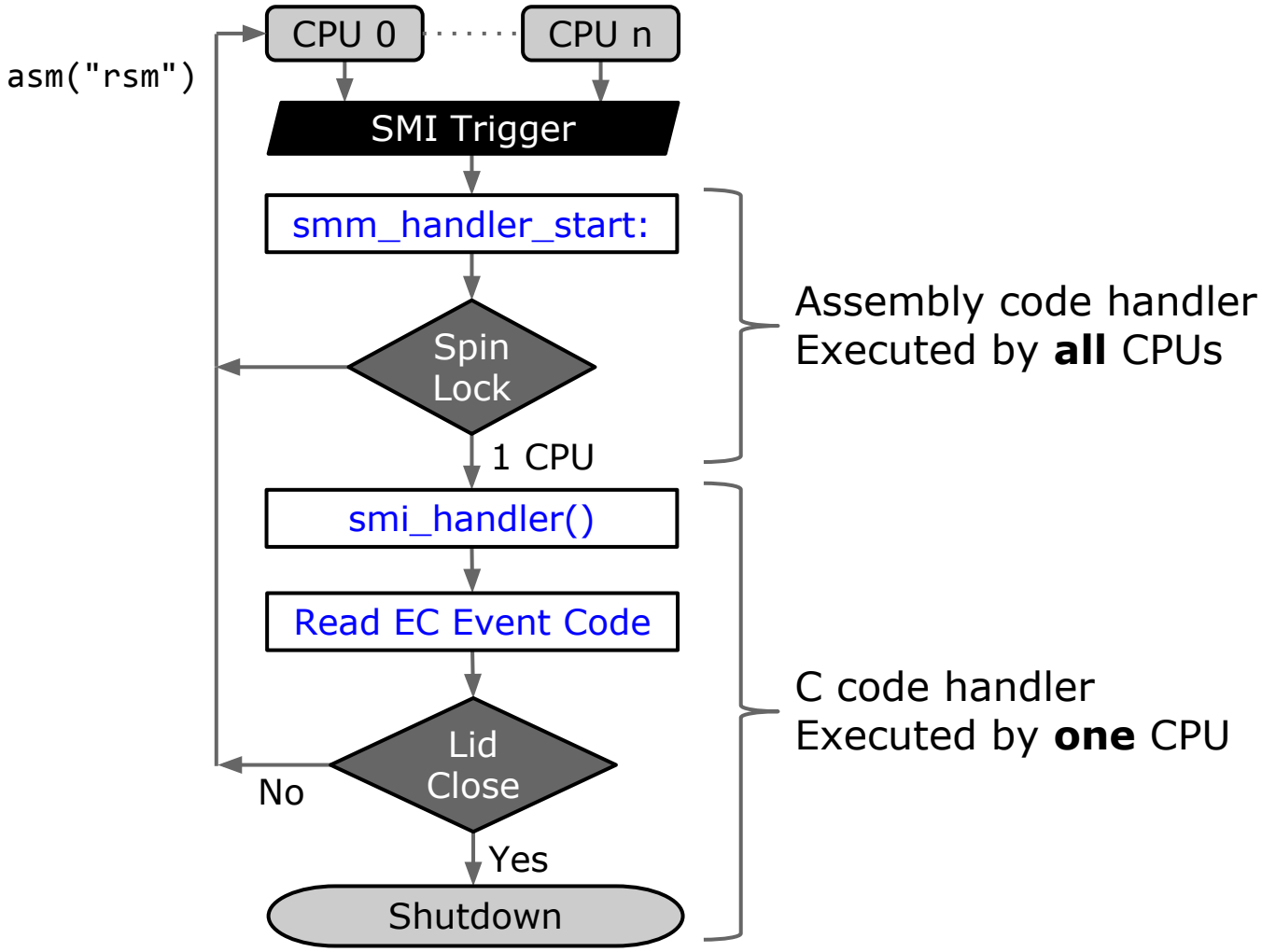
System Management Interrupt

- Special Non-Maskable Interrupt
- Used to enter System Management Mode
- Triggered for various reasons
  - I/O or Memory transaction
  - Periodic Timer
  - Watchdog Timeout
  - Attempt to Write Flash
  - External GPIO Pin

System Management Interrupt

- CPU state stored in SMRAM State Save
- RIP changed to the start of SMM handler
- All CPUs will enter SMM
- RSM instruction returns to execution

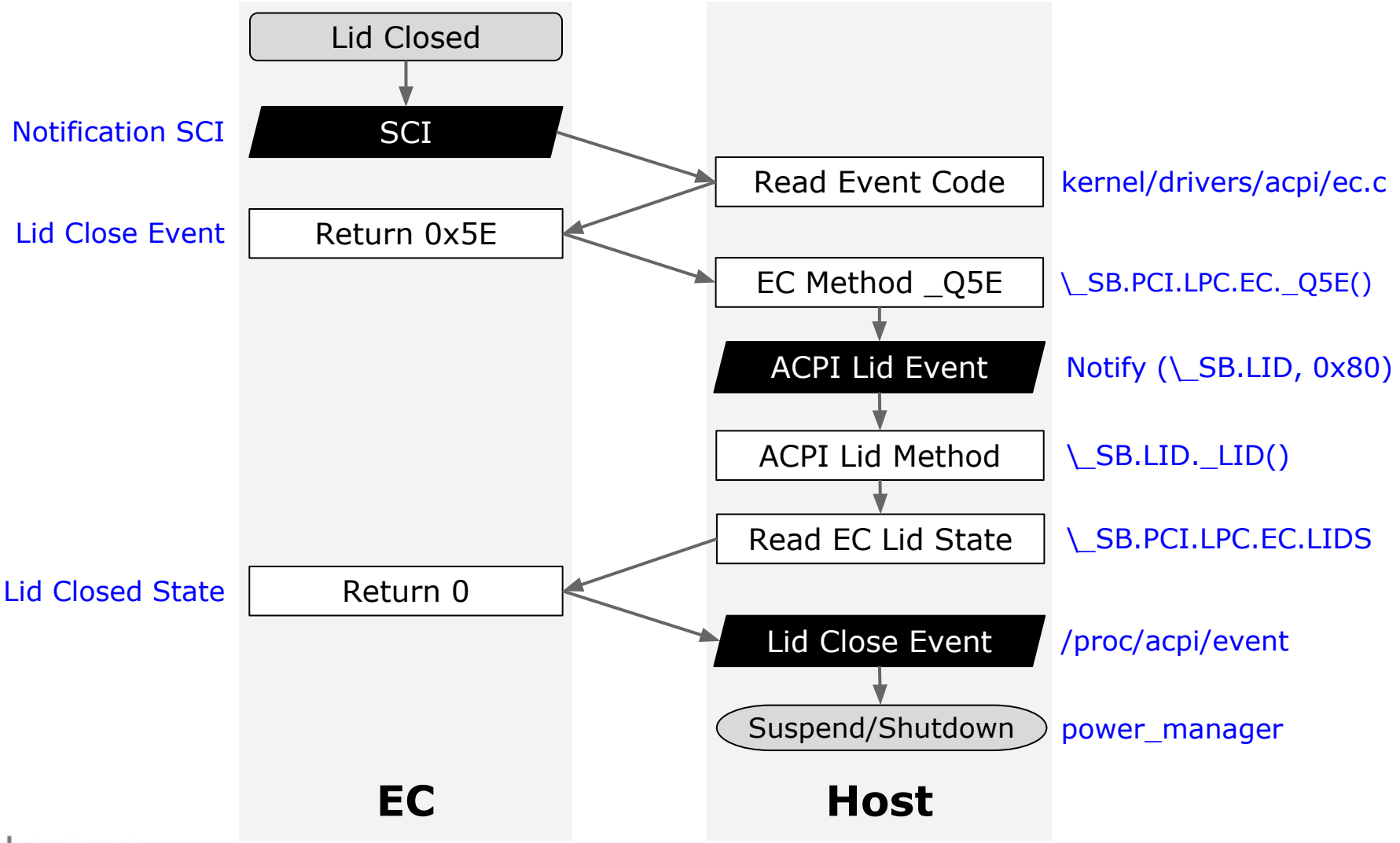
Lid Close: SMI Event



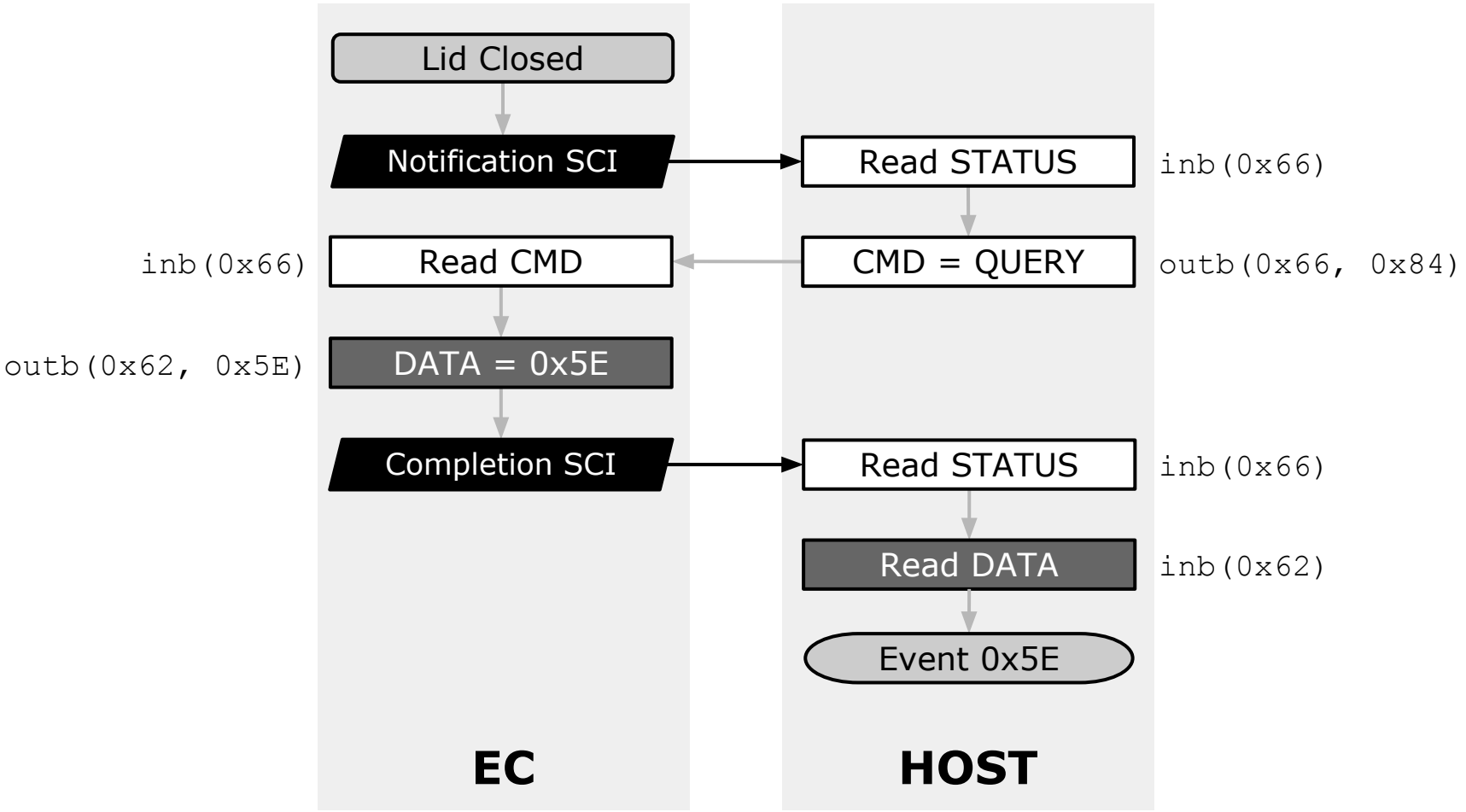
Lid Close Example

OS Event Handling

Lid Close: ACPI Event



ACPI Event Query



Questions?