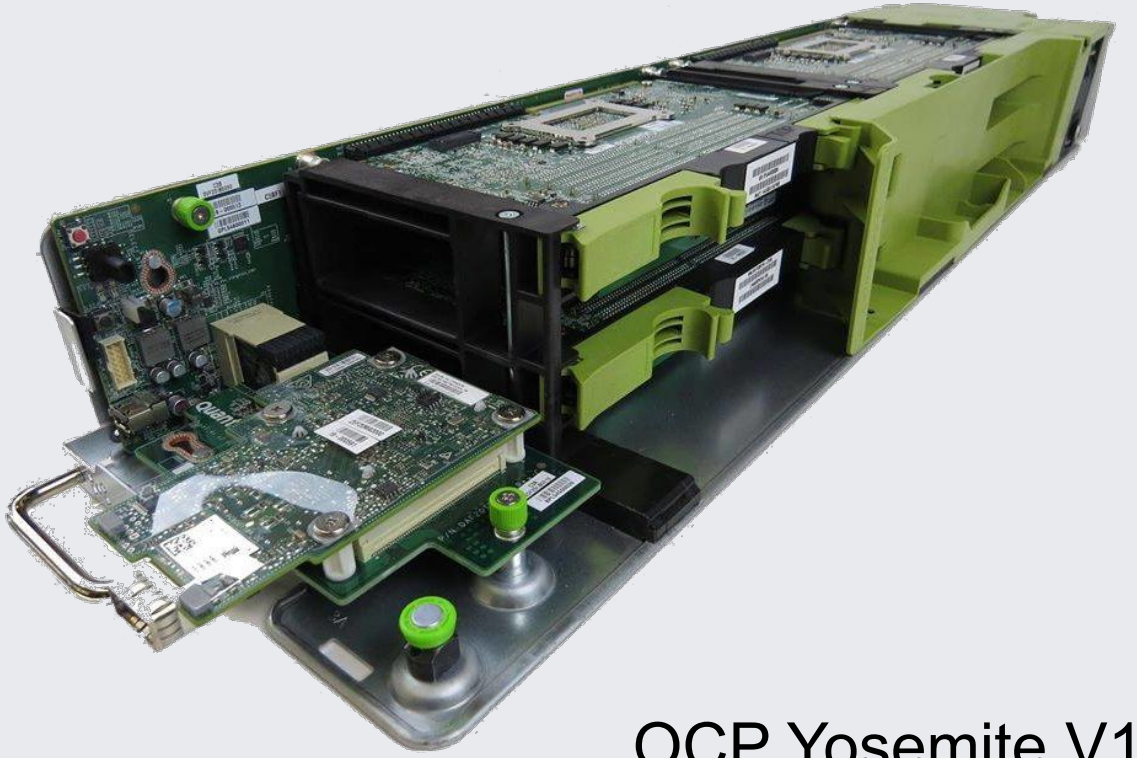
Open Source Firmware @ Facebook

Design, deployment, and demo

David Hendricks: Firmware Engineer
Andrea Barberio: Production Engineer

It started with a quick hack back in ‘17...



OCP Yosemite V1/Mono Lake

```
[1D][19][A1][A3][A3][A7][A9][AA][AA][AB][AF][B0][B1][AF][1D][19][A1][A3][A3][A7]
[A9][A7][A7][A7][A8][A9][A9][A9][AA][AA][AA][AE][AF][B0][B1][B4][B2][B3][B6][B7]
[B7][B7][B7][B7][B7][B7][B7][B7][B7][B7][B7][B7][B7][B7][B7][B7][B7][B7][B7][B7]
[B7][B7][B7][B7][B7][B7][B7][B7][B7][B7][B7][B7][B7][B8][B9][B9][BA][B7][BB][BC]
[BF][33][32][4F][9A]
```

Vendor firmware

```
system 00:00: [mem 0xfec01000-0xfed00fff] could not be reserved
system 00:00: [mem 0xfed00000-0xfedfffff] has been reserved
system 00:00: [mem 0xfee00000-0xfeeffffff] has been reserved
system 00:00: [mem 0xff000000-0xffffffff] has been reserved
system 00:01: [mem 0xfed00000-0xfed003ff] has been reserved
pnp: PnP ACPI: found 4 devices
pci 0000:00:1f.6: BAR 0: assigned [mem 0x380000000000-0x380000000fff 64bit]
pci 0000:00:01.0: PCI bridge to [bus 01]
pci 0000:00:02.0: PCI bridge to [bus 02]
pci 0000:00:02.0:   bridge window [mem 0xfa300000-0xfa3fffff]
pci 0000:00:02.2: PCI bridge to [bus 03]
pci 0000:00:02.2:   bridge window [mem 0xfa400000-0xfa4fffff]
pci 0000:00:02.2:   bridge window [mem 0xfa000000-0xfa2fffff 64bit pref]
pci 0000:00:03.0: PCI bridge to [bus 04]
pci 0000:00:03.0:   bridge window [mem 0xfa500000-0xfa5fffff]
pci 0000:00:03.0:   bridge window [mem 0xf8000000-0xf9ffffff 64bit pref]
NET: Registered protocol family 2
tcp_listen_portaddr_hash hash table entries: 16384 (order: 6, 262144 bytes)
TCP established hash table entries: 262144 (order: 9, 2097152 bytes)
TCP bind hash table entries: 65536 (order: 8, 1048576 bytes)
TCP: Hash tables configured (established 262144 bind 65536)
UDP hash table entries: 16384 (order: 7, 524288 bytes)
UDP-Lite hash table entries: 16384 (order: 7, 524288 bytes)
```

coreboot + LinuxBoot

System firmware in a nutshell

- First bit of code that runs when CPU is turned on
- Sometimes referred to as "BIOS"

Initialize Hardware

Load target OS kernel

System firmware in a nutshell

- First bit of code that runs when CPU is turned on
- Sometimes referred to as "BIOS"

Initialize Hardware

coreboot

Load target OS kernel

System firmware in a nutshell

- First bit of code that runs when CPU is turned on
- Sometimes referred to as "BIOS"

- Bootblock

Initialize Hardware

coreboot

Load target OS kernel

System firmware in a nutshell

- First bit of code that runs when CPU is turned on
- Sometimes referred to as "BIOS"

- Bootblock
- Romstage

Initialize Hardware

coreboot

Load target OS kernel

System firmware in a nutshell

- First bit of code that runs when CPU is turned on
- Sometimes referred to as "BIOS"

- Bootblock
- Romstage
- Ramstage

Initialize Hardware

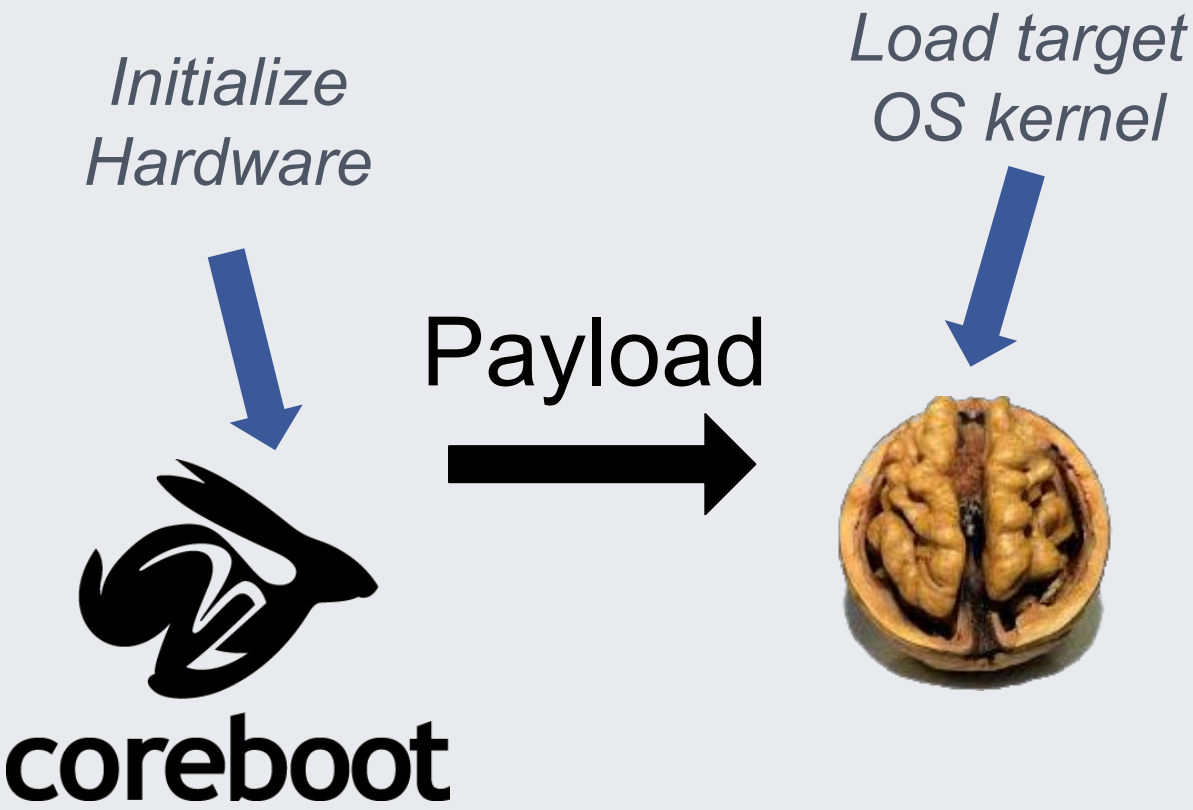
coreboot

Load target OS kernel

System firmware in a nutshell

- First bit of code that runs when CPU is turned on
- Sometimes referred to as "BIOS"

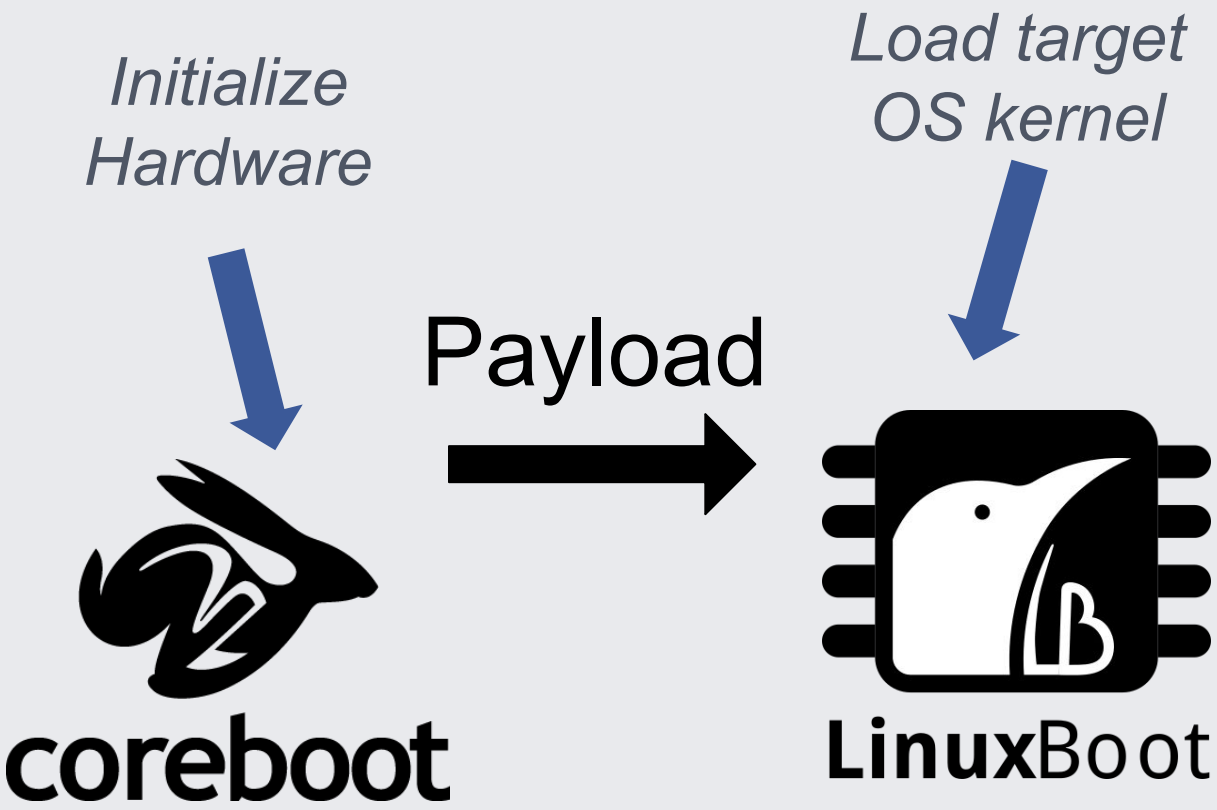
- Bootblock
- Romstage
- Ramstage



System firmware in a nutshell

- First bit of code that runs when CPU is turned on
- Sometimes referred to as "BIOS"

- Bootblock
- Romstage
- Ramstage

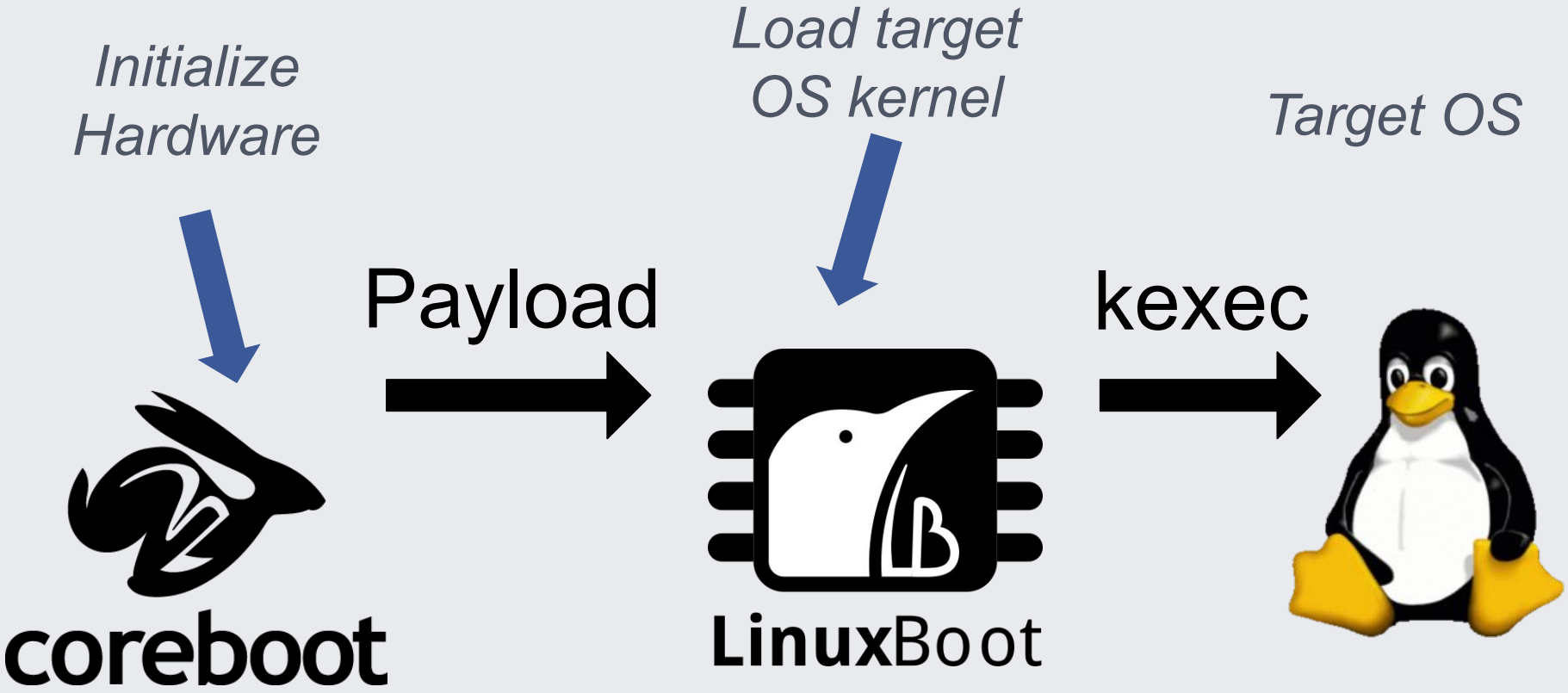


LinuxBoot logo by Susanne Nähler

System firmware in a nutshell

- First bit of code that runs when CPU is turned on
- Sometimes referred to as "BIOS"

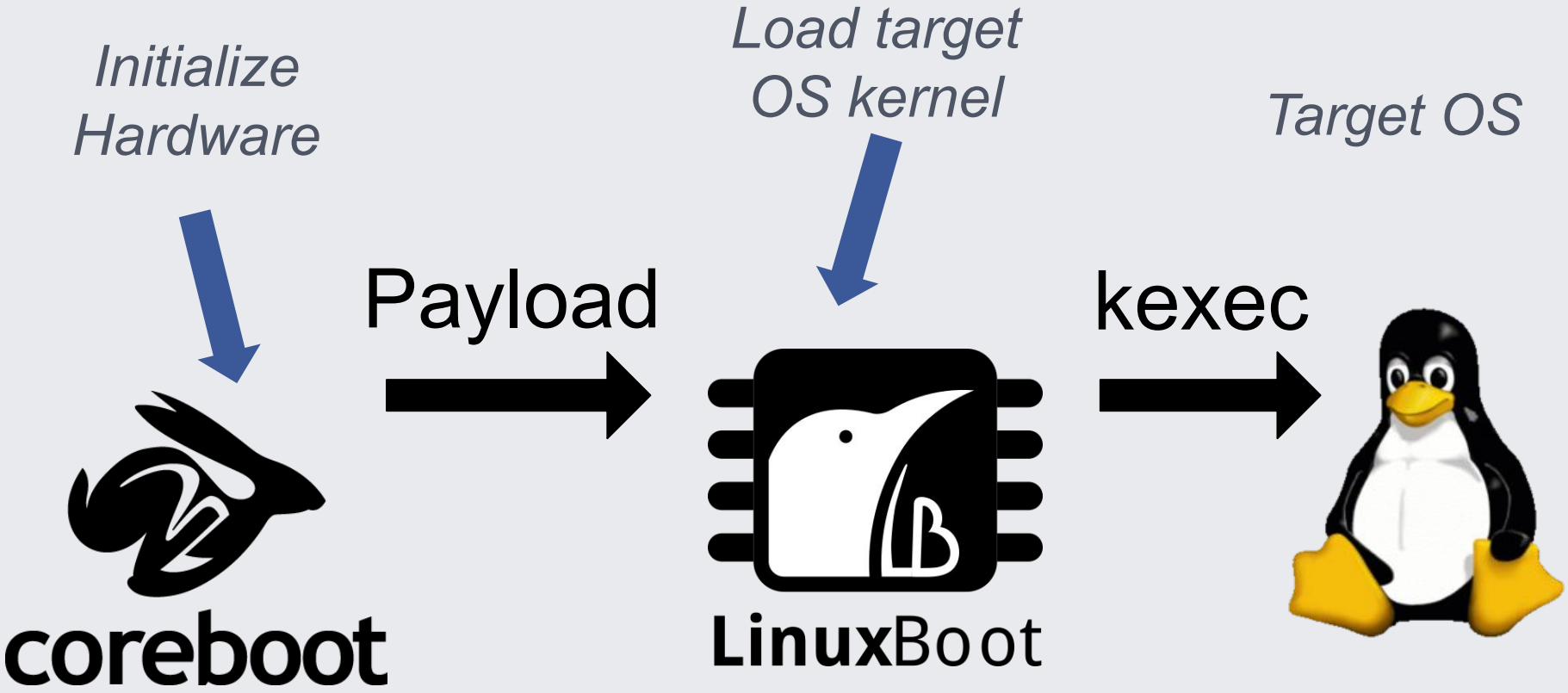
- Bootblock
- Romstage
- Ramstage



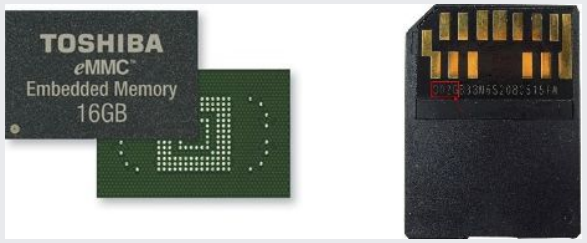
System firmware in a nutshell

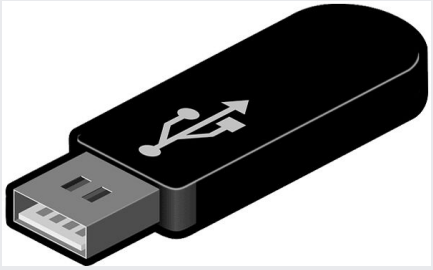
- First bit of code that runs when CPU is turned on
- Sometimes referred to as "BIOS"

- Bootblock
- Romstage
- Ramstage



Problem: Local booting is more complex

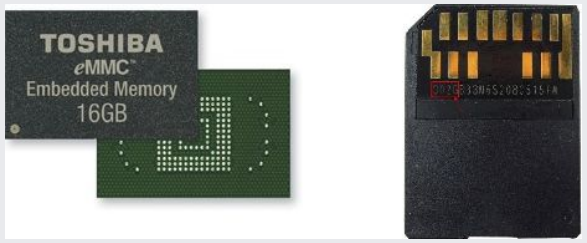


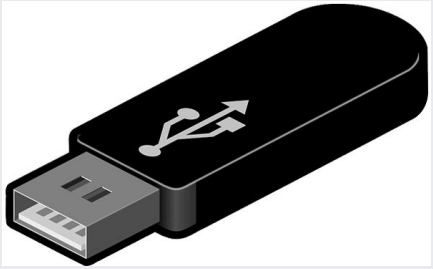


SSD: By Dmitry Nosachev [CC BY-SA 4.0], from Wikimedia Commons
eMMC: By Toniperis [CC BY-SA 4.0], from Wikimedia Commons

Then	Now
Few, simple interfaces	Many interfaces and protocols

Problem: Local booting is more complex

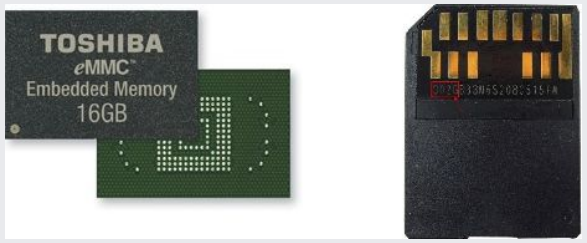


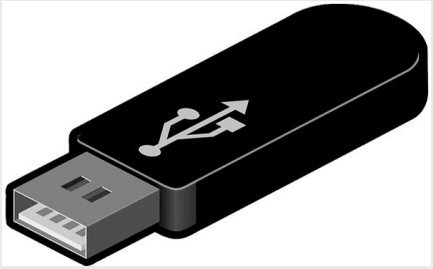


SSD: By Dmitry Nosachev [CC BY-SA 4.0], from Wikimedia Commons
eMMC: By Toniperis [CC BY-SA 4.0], from Wikimedia Commons

Then	Now
Few, simple interfaces	Many interfaces and protocols
Simple, low-speed links	High-speed links

Problem: Local booting is more complex





SSD: By Dmitry Nosachev [CC BY-SA 4.0], from Wikimedia Commons
eMMC: By Toniperis [CC BY-SA 4.0], from Wikimedia Commons

Then	Now
Few, simple interfaces	Many interfaces and protocols
Simple, low-speed links	High-speed links
Blindly execute MBR (CHS 0/0/1)	Scan, mount+decrypt, verify

Problem: Network booting is more complex

Then	Now
Small, trusted networks	Global, untrusted networks

Problem: Network booting is more complex

Then	Now
Small, trusted networks	Global, untrusted networks
Few, simple interfaces and protocols	Many interfaces and protocols

Problem: Network booting is more complex

Then	Now
Small, trusted networks	Global, untrusted networks
Few, simple interfaces and protocols	Many interfaces and protocols
TFTP/PXE, security an afterthought	TLS/HTTPS, designed for security BitTorrent for scalability

Another way of looking at it...

5-year growth of certain subsystems in Linux	3.15 (June 2014)	5.1 (May 2019)	Δ
sloccount net/core net/ipv{4,6}	131,550	174,195	+32%
git log --oneline net/core net/ipv{4,6} | wc -l	10,645	18,004	+69%
sloccount drivers/net/ethernet	729,678	1,295,169	+77%
git log --oneline drivers/net/ethernet | wc -l	7,320	27,244	+272%
sloccount drivers/block	90,740	74,149	-18%
git log --oneline drivers/block | wc -l	4,238	6,299	+49%

TL;DR: Booting is complicated

Then/Now	Now/Future
SysFW/BIOS is an OS	Let Linux Do It

The Solution: Let Linux Do It

LinuxBoot

Then/Now	Now/Future
SysFW/BIOS is an OS	Let Linux Do It
Opaque	Open, well-understood at FB

The Solution: Let Linux Do It

LinuxBoot

Then/Now	Now/Future
SysFW/BIOS is an OS	Let Linux Do It
Opaque	Open, well-understood at FB
Proprietary ecosystem	Auditable, debuggable

The Solution: Let Linux Do It

LinuxBoot

Then/Now	Now/Future
SysFW/BIOS is an OS	Let Linux Do It
Opaque	Open, well-understood at FB
Proprietary ecosystem	Auditable, debuggable
Vendor-specific tooling	Open-source tools

The Solution: Let Linux Do It

LinuxBoot

Then/Now	Now/Future
SysFW/BIOS is an OS	Let Linux Do It
Opaque	Open, well-understood at FB
Proprietary ecosystem	Auditable, debuggable
Vendor-specific tooling	Open-source tools
Product-specific	Portable, re-usable

FW-based runtime services (thar be dragons...)

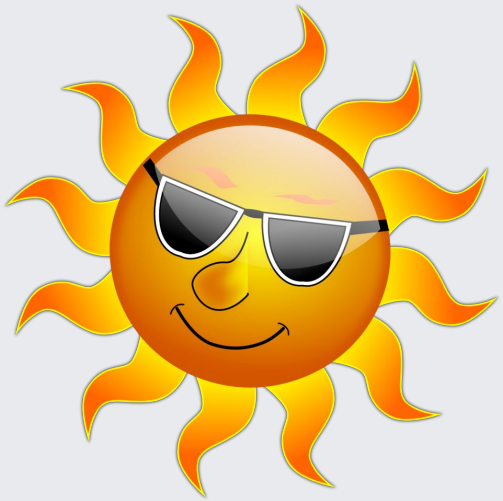
- Run at highest privilege level
- Can access some (or all) resources

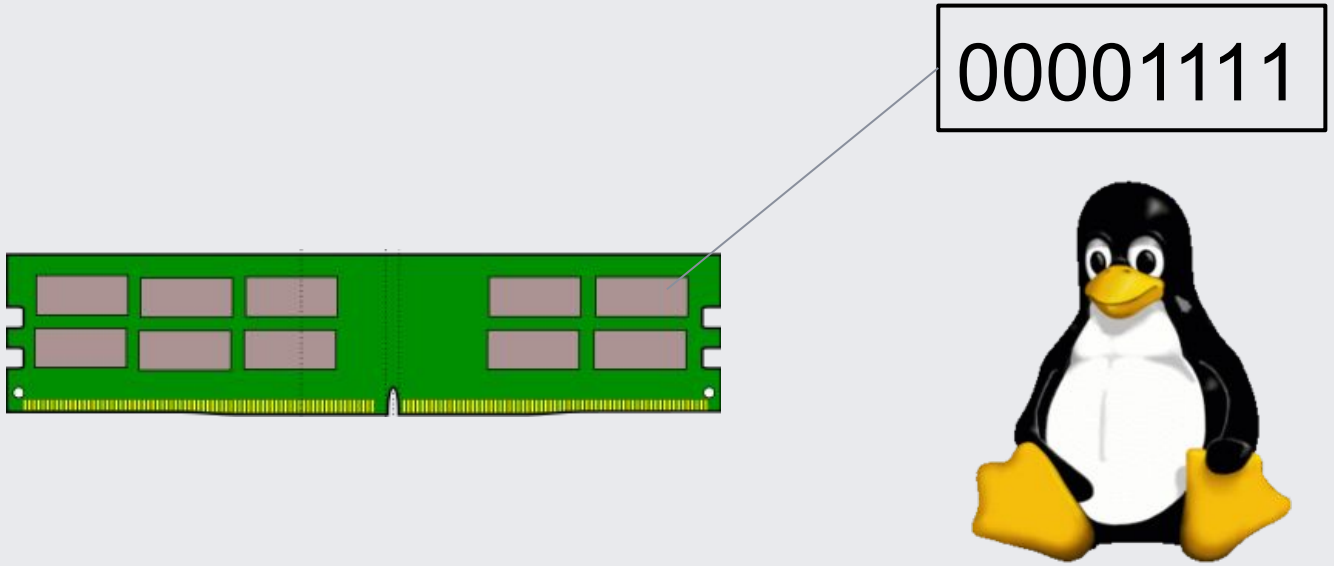
FW-based runtime services (thar be dragons...)

- Run at highest privilege level
- Can access some (or all) resources
- Run outside your OS/VM/container security model
  - x86: System Management Mode (SMM), UEFI Runtime
  - Non-x86: ARM Trusted Firmware, RISC-V M-Mode
  - Example: “Firmware First” HW error handlers

Example: HW Error Handling (“Firmware First”)

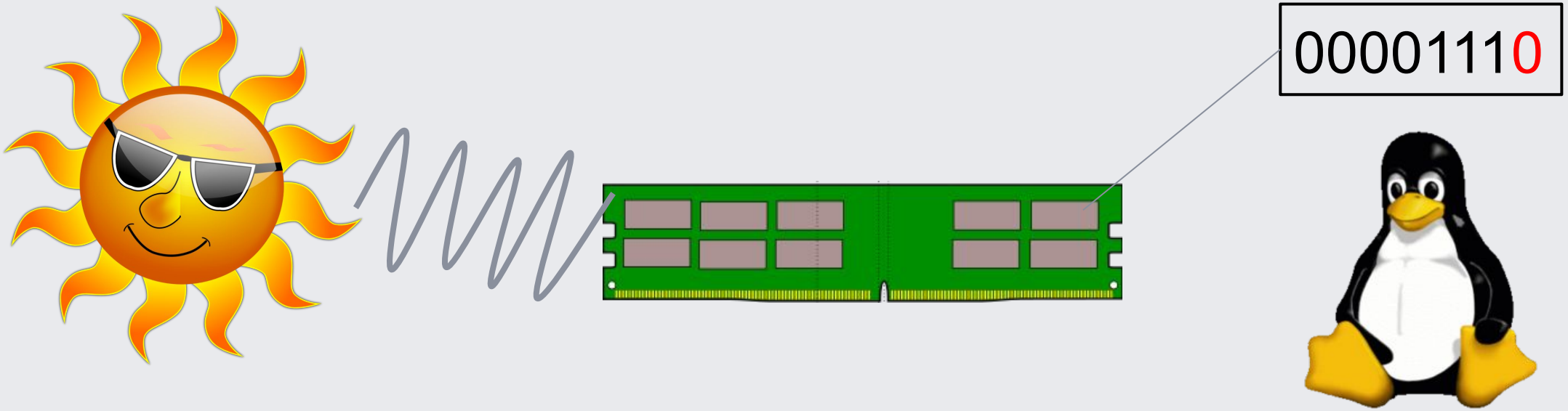
- Code running merrily within the confines of OS/VM





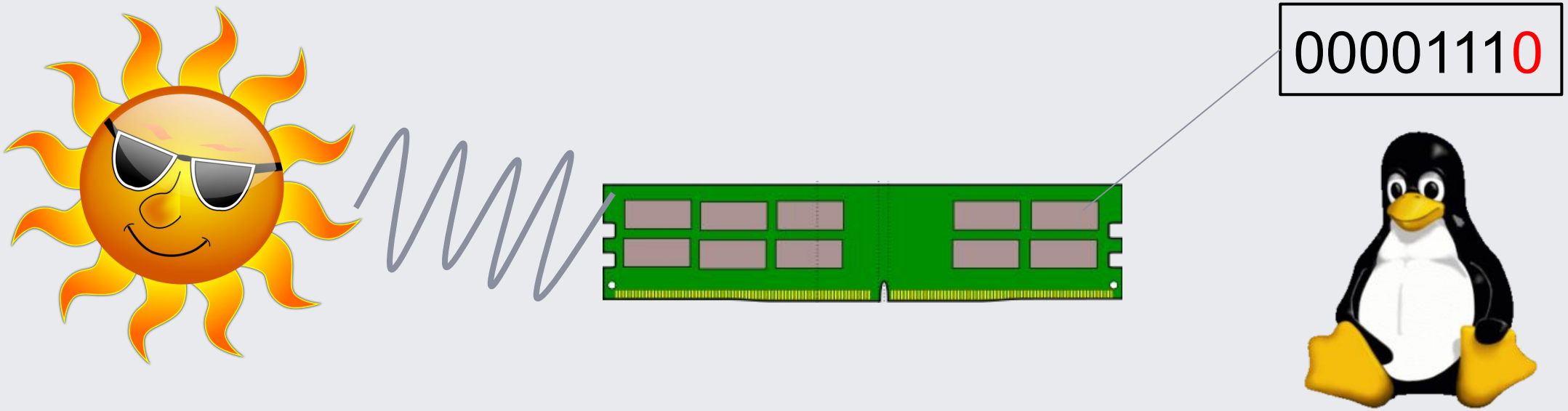
Example: HW Error Handling (“Firmware First”)

- Uh oh, something just made a bit flip!



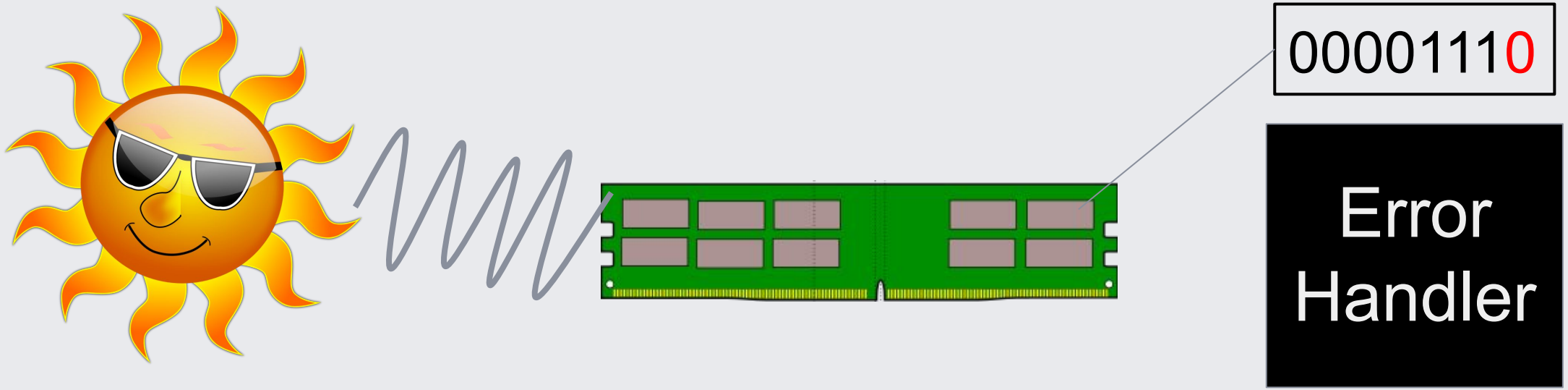
Example: HW Error Handling (“Firmware First”)

- Uh oh, something just made a bit flip!
- Error detected by DRAM controller, CPU interrupted



Example: HW Error Handling (“Firmware First”)

- Uh oh, something just made a bit flip!
- Error detected by DRAM controller, CPU interrupted
- CPU jumps to FW error handler



FW-based runtime services (thar be dragons...)

- You will (and may already) have a webserver in your FW
  - RESTful services used for remote configuration
  - Standard in UEFI spec as of June 2019

"New UEFI Specification, Version 2.8, captures evolution of platform firmware technology"

...

"The new specification adds support for REST. REST is a software architectural style that defines a set of constraints to be used for creating Web services. RESTful Web services provide interoperability between computer systems on the internet."

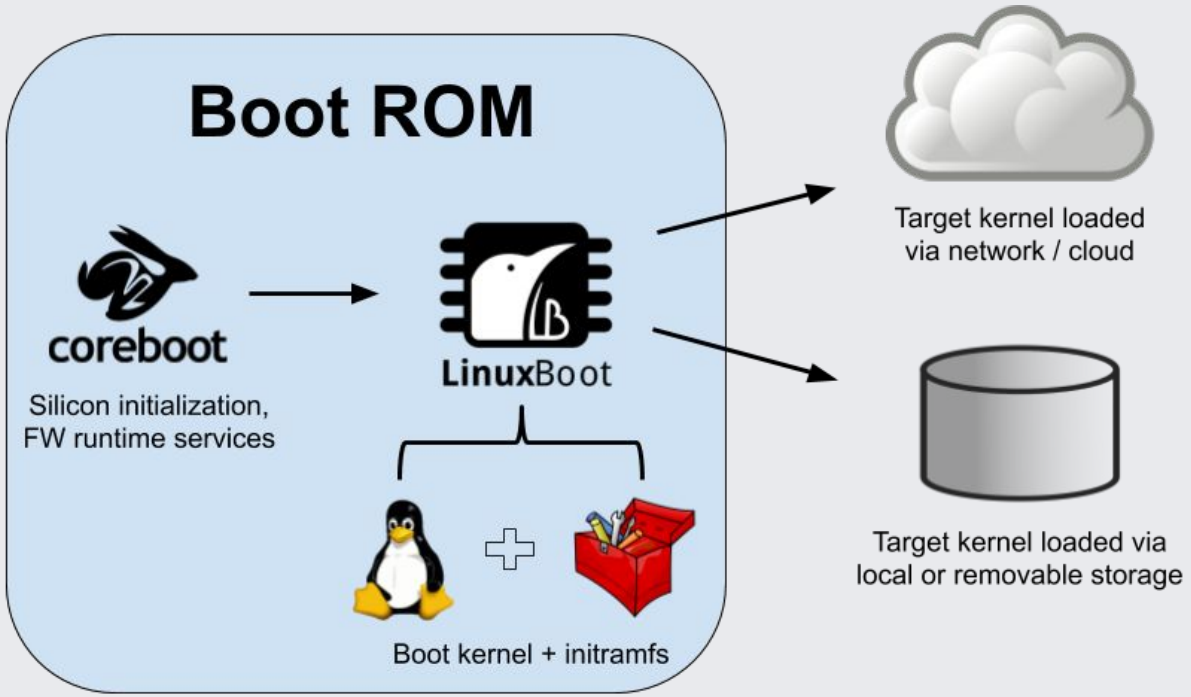
https://www.businesswire.com/news/home/20190604005069/en/Industry-Standard-Firmware-Takes-Step-New-UEFI-Forum

FW-based runtime services (thar be dragons...)

- Some FW runs out-of-band on a separate microcontroller
  - Intel Management Engine (ME)
  - AMD Platform Security Processor (PSP)
  - Qualcomm Integrated Management Controller (IMC)
  - … various other management controllers
- They often have direct access to HW resources
  - Including DRAM controller, PCIe bus, etc.

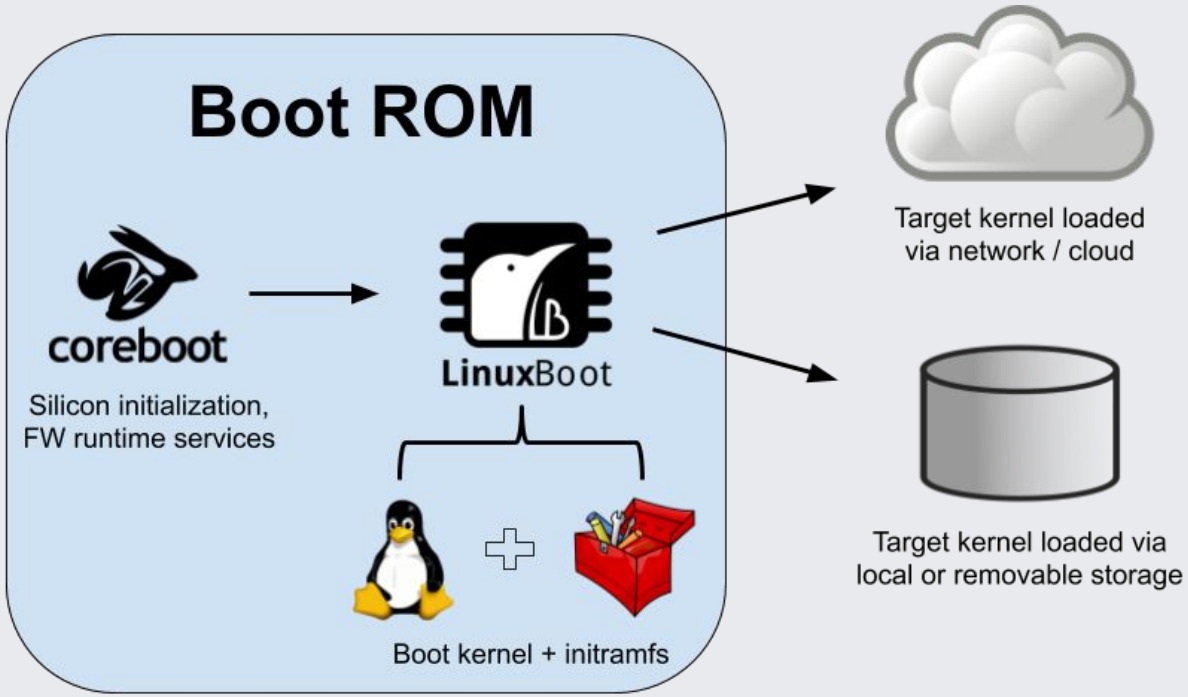
To recap

- Put a kernel+initramfs in boot ROM
- Do minimal silicon init and jump to Linux as soon as possible
- Use Linux to boot Linux



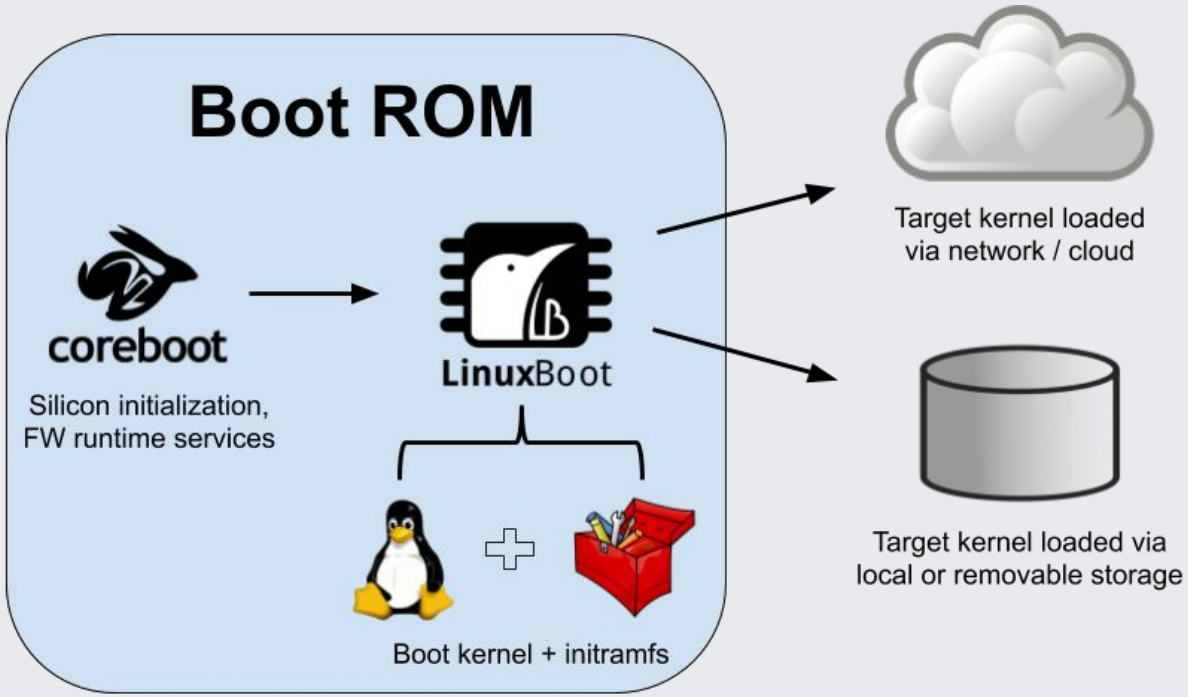
To recap

- Put a kernel+initramfs in boot ROM
- Do minimal silicon init and jump to Linux as soon as possible
- Use Linux to boot Linux
- **Production-quality drivers, networking**



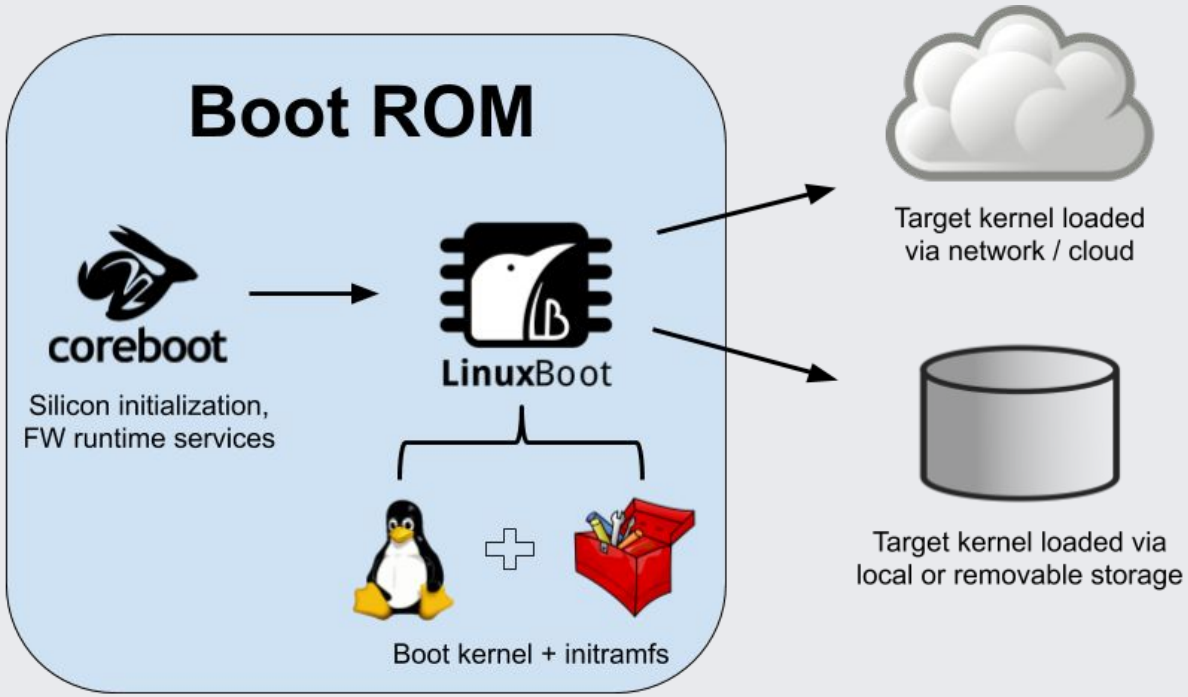
To recap

- Put a kernel+initramfs in boot ROM
- Do minimal silicon init and jump to Linux as soon as possible
- Use Linux to boot Linux
- Production-quality drivers, networking
- **Add features + tools as needed**
- **Debug, build, deploy on our schedule**



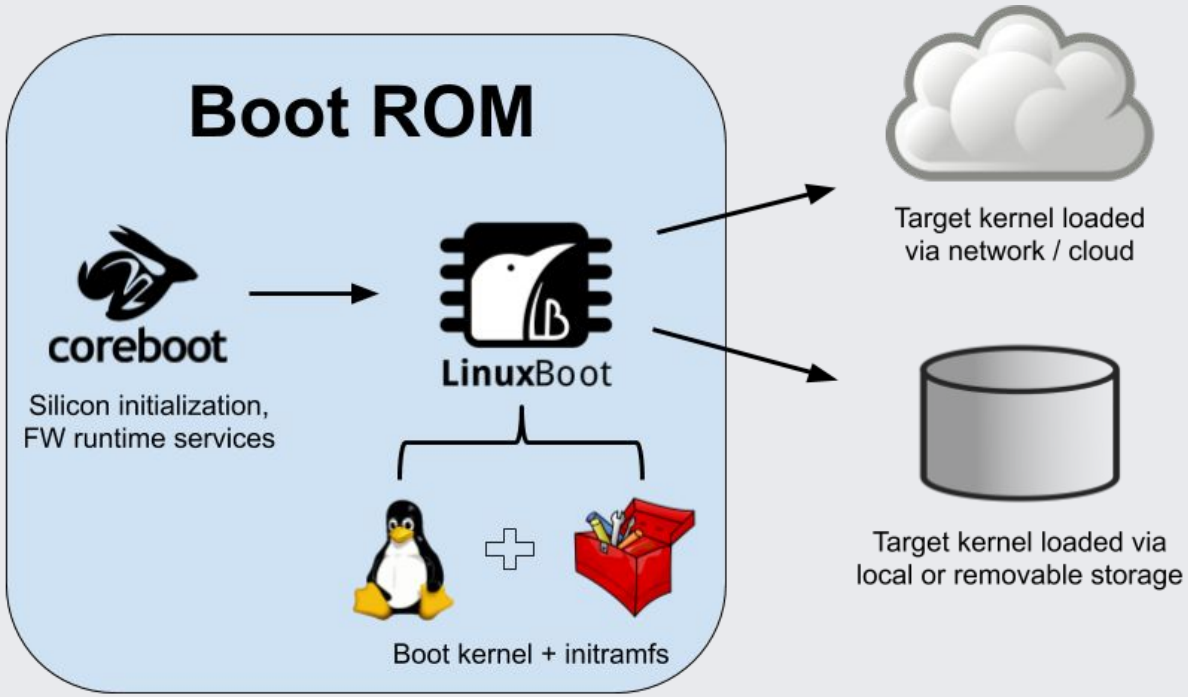
To recap

- Put a kernel+initramfs in boot ROM
- Do minimal silicon init and jump to Linux as soon as possible
- Use Linux to boot Linux
- Production-quality drivers, networking
- Add features + tools as needed
- Debug, build, deploy on our schedule
- **Flexible security architecture**



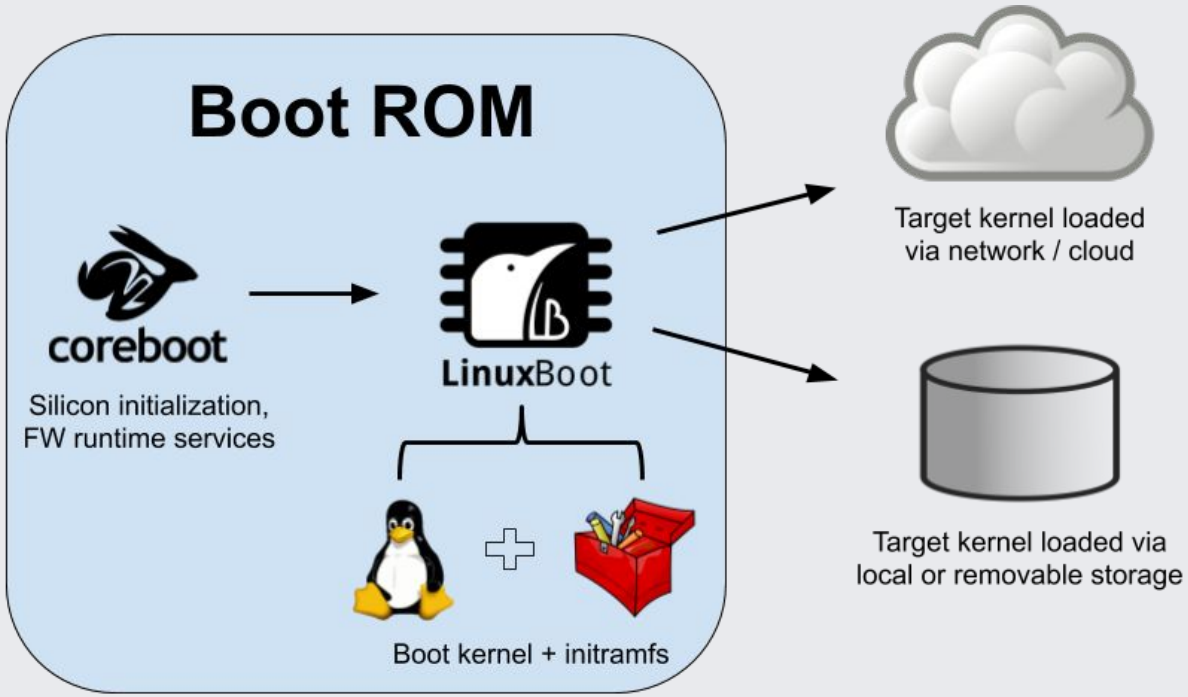
To recap

- Put a kernel+initramfs in boot ROM
- Do minimal silicon init and jump to Linux as soon as possible
- Use Linux to boot Linux
- Production-quality drivers, networking
- Add features + tools as needed
- Debug, build, deploy on our schedule
- Flexible security architecture
- **Boot in seconds, not minutes**



To recap

- Put a kernel+initramfs in boot ROM
- Do minimal silicon init and jump to Linux as soon as possible
- Use Linux to boot Linux
- Production-quality drivers, networking
- Add features + tools as needed
- Debug, build, deploy on our schedule
- Flexible security architecture
- Boot in seconds, not minutes
- **Bring modern, open-source development to the firmware**



Present

- Coreboot Project (coreboot.org)
  - Celebrating its 20th anniversary in 2019
  - Member of Software Freedom Conservancy
- LinuxBoot Project (linuxboot.org)
  - Linux Foundation member project as of Jan. 2018
- Open Compute Project (opencompute.org)
  - Industry effort to develop open HW designs for hyperscale deployment
  - Founded in 2011
    - 100+ members including FB, Google, Tencent, Alibaba, MS, Intel, etc.
  - “Open System Firmware” (OSF) initiative started in March 2018

OPEN Compute Project®

Present

- Deployment
  - In progress at Facebook
    - “Reinventing Facebook’s data center network with F16, Minipack”
  - Google also working on deployment
    - Google around for their LinuxBoot talks
  - Sesame Open Hardware
    - Selling refurbished OCP servers preloaded with LinuxBoot
    - Great for SMBs, universities, etc.
  - Probably more that we’re not aware of...

OPEN Compute Project®

Future

- Continue development on existing and upcoming OCP platforms
- **March 2021**
  - Tentative cutover date for “OCP Accepted” criteria

OCP Open System Firmware

Because if you don't own your firmware
your firmware owns you*!*

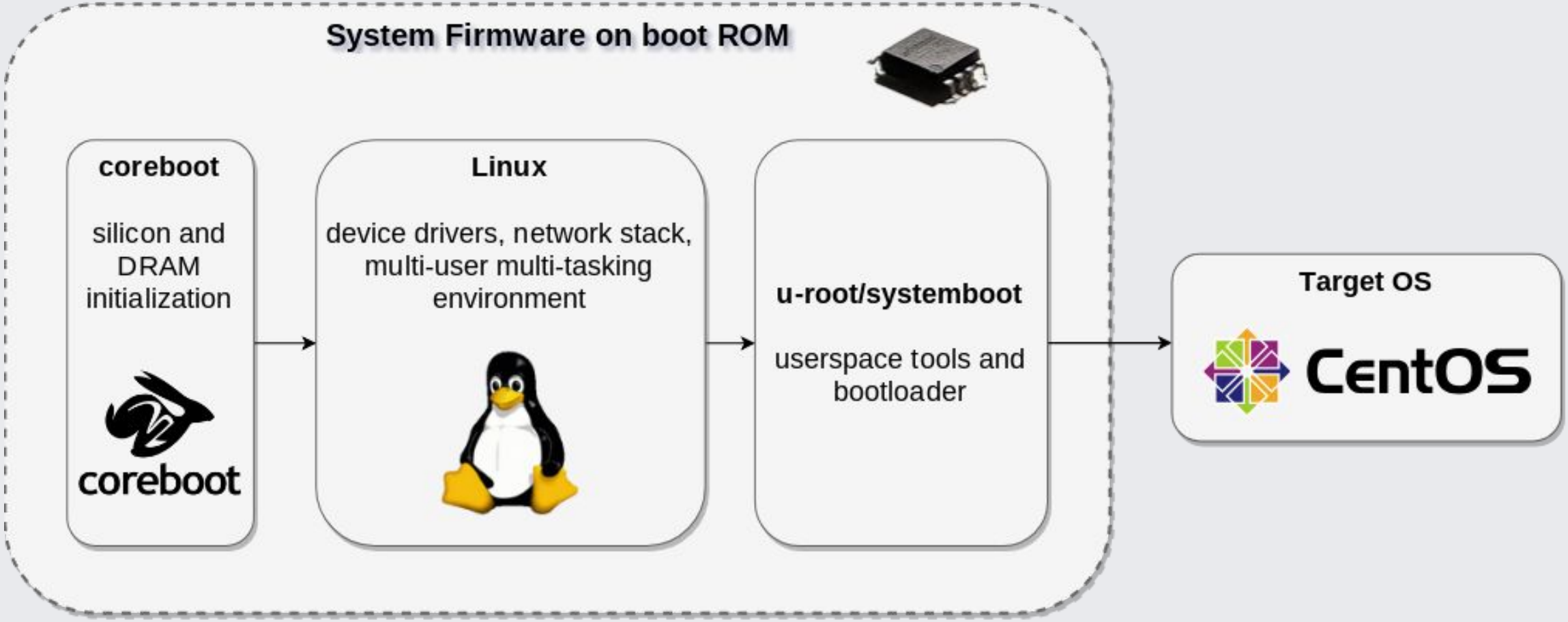
Hands on with LinuxBoot

Andrea Barberio

Host Provisioning / LinuxBoot team @ Facebook infrastructure

- Host provisioning infrastructure
- LinuxBoot development and production

LinuxBoot architecture @ FB



source : systemboot.org

Boot entries

- Boot entries and their order are stored in VPD variables
- Value in JSON format. Example:

- **Boot0000**={
  "type":"netboot",
  "method":"dhcpv6",
  "mac":"00:fa:ce:b0:0c:00"
  “debug_on_failure”: true,
  }

- **Boot0002**={
  "type": "localboot",
  “method”: “path”,
  "device_guid": "abcdef...",
  "kernel": "/path/to/kernel",
  “ramfs”: “/boot/ramfs.cpio.xz”,
  }

In production: testing (1)

- We run LinuxBoot in production[1]. How to test it before hitting prod?
- **Unit tests:** coreboot, u-root, systemboot have unit and integration tests running in public CIs
- but this is not enough

[1] https://code.fb.com/data-center-engineering/f16-minipack/

In production: testing (2)

- **Continuous end-to-end testing:** reflash and run continuous tests (e.g. stress testing, dependencies)
- **On-demand end-to-end testing**: every diff triggers a test run, on real hardware and VMs. Results are reported back on the code review tool

- We created an open source system testing framework called **ConTest** (for continuous testing), soon on GitHub

In production: troubleshooting

- *Manual troubleshooting* using Linux programs
- *Automated troubleshooting* and remediations with the **checker framework**
  - runs through a list of checks and remediations *at the time of the failure*
- Example: **fixmynetboot**
  - it runs when netboot fails. It *checks* interface status, addresses, link speed, DHCP, HTTP, and so on.
  - it can try to *remediate*
  - if everything fails, it can drop into an **emergency shell** for manual troubleshooting
- Imagine doing the same for disks, etc

Live demo

- Using a virtual machine, for simplicity
    - built using github.com/insomniacslk/osf-build
- Showing boot flow (coreboot, Linux, u-root, systemboot)
- Boot sequence and boot variables
- Network booting via DHCPv6 and HTTPS
- Troubleshooting stories and automated troubleshooting
- Disk booting with either Grub or local boot parameters

Thanks!

Resources

- coreboot.org
- linuxboot.org
- u-root.tk
- systemboot.org
- opencompute.org

Contact

- Slack: http://slack.u-root.com
- IRC: #coreboot on freenode

Participate

OPEN Compute Summit

OPEN SOURCE FIRMWARE CONFERENCE 2019 SEPTEMBER 03–06