OPEN. FOR BUSINESS.

OCP SUMMIT

OSF Track

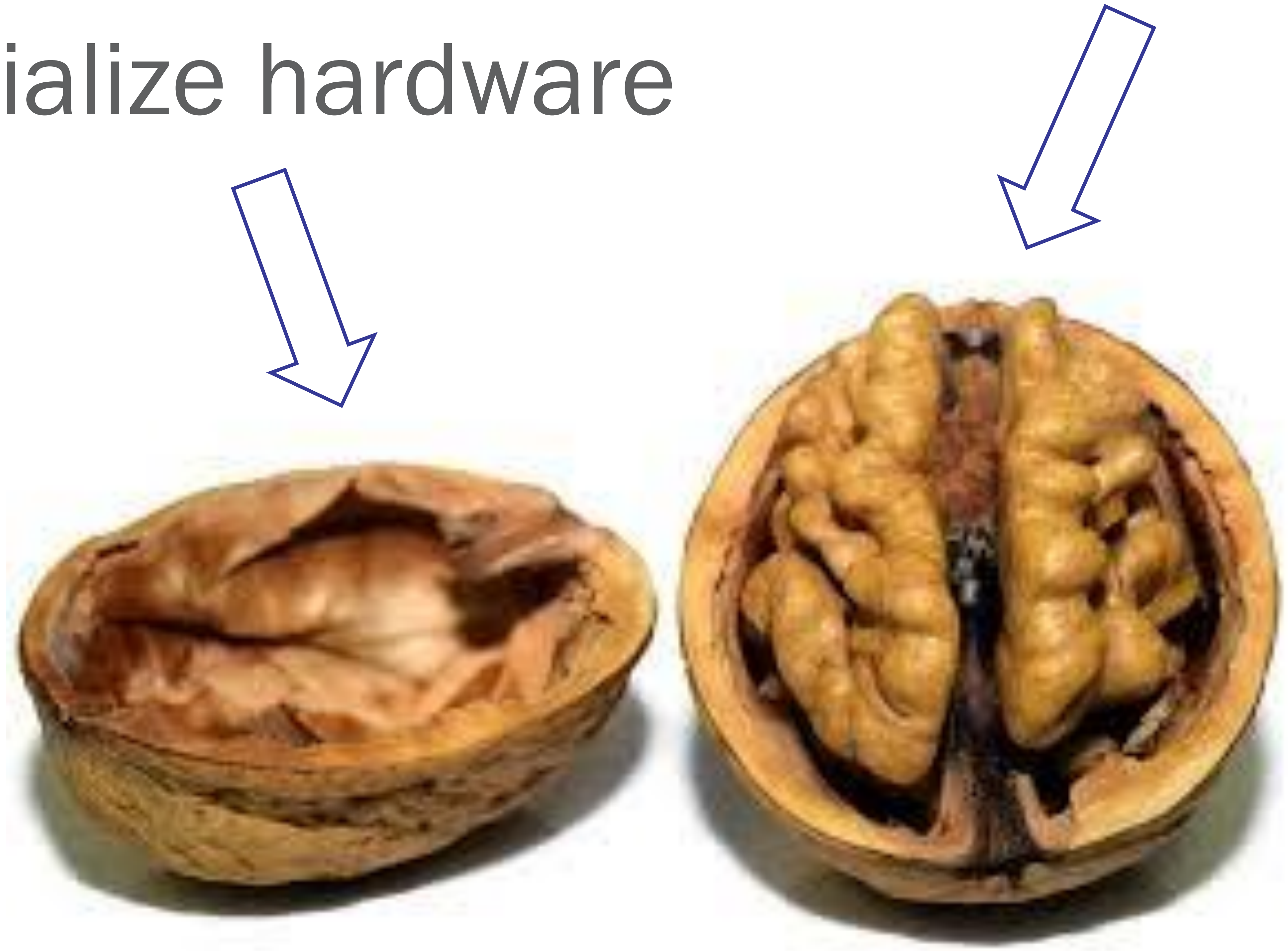
Turning Linux engineers into firmware engineers

David Hendricks Firmware Engineer/Facebook

Andrea Barberio Production Engineer/Facebook

System firmware in a nutshell

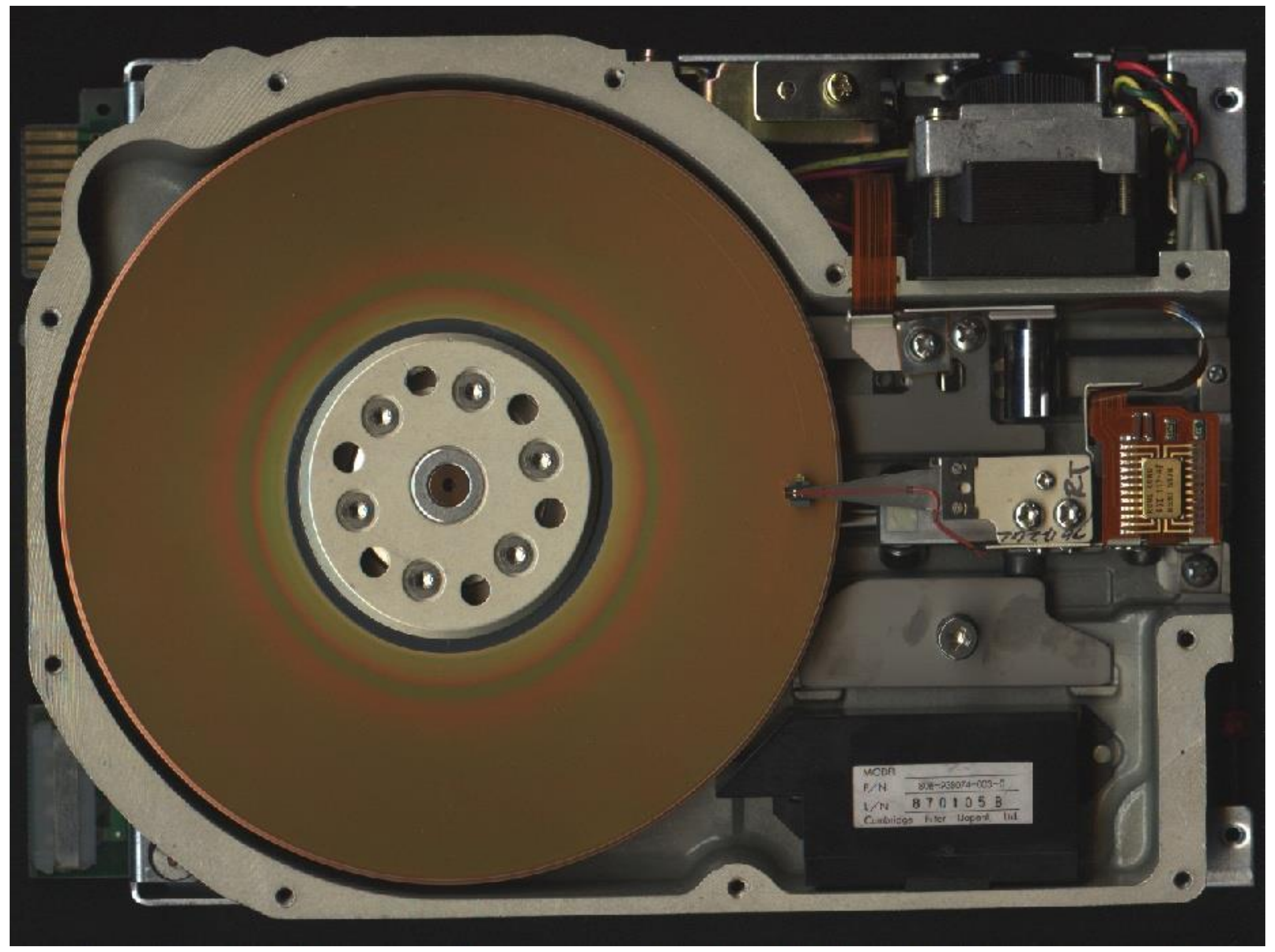
- First bit of code that runs when CPU is turned on
- Sometimes still referred to as "BIOS"

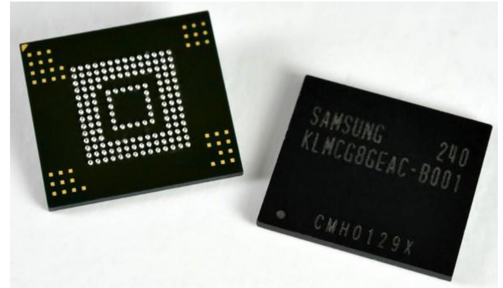


Load OS

Initialize hardware

Problem: Local booting is more complex



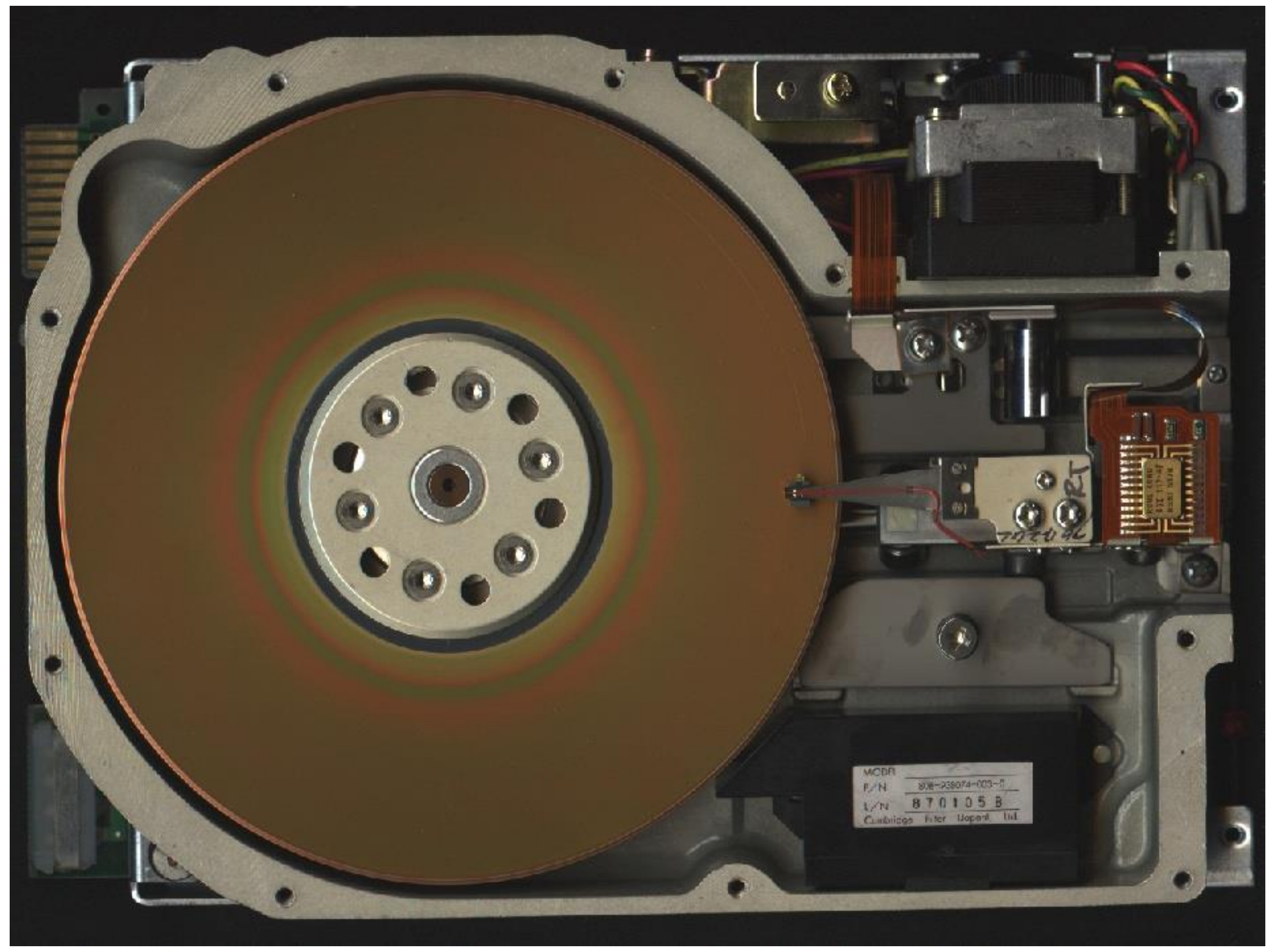


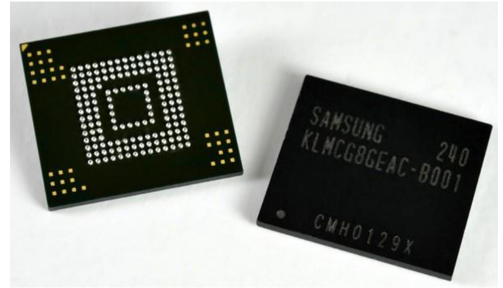
By Toniperis [CC BY-SA 4.0], from Wikimedia Commons

By Dmitry Nosachev [CC BY-SA 4.0], from Wikimedia Commons

Then	Now
Few interfaces	Many interfaces and protocols

Problem: Local booting is more complex



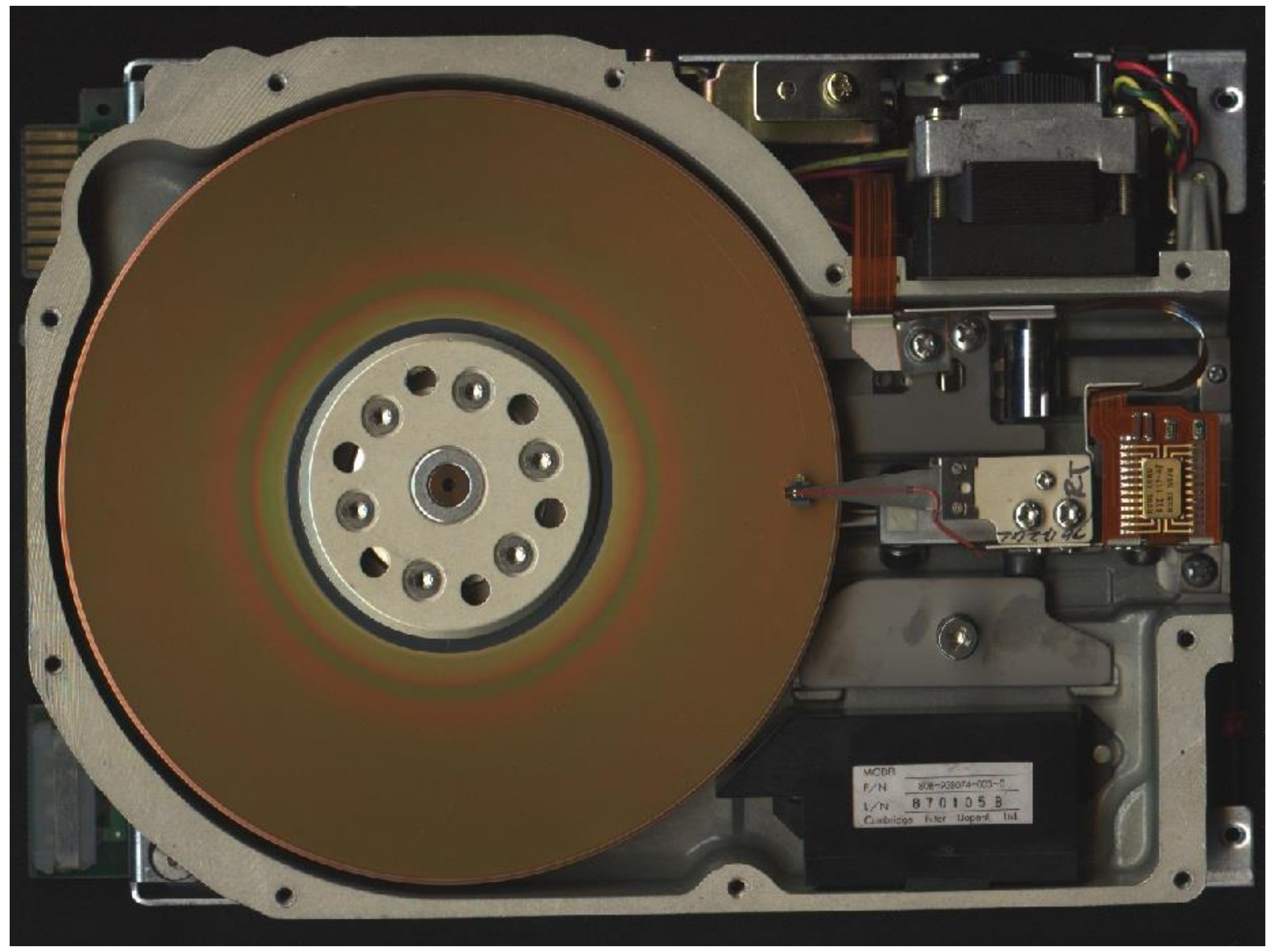


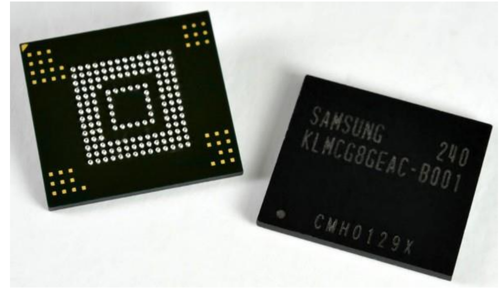
By Toniperis [CC BY-SA 4.0], from Wikimedia Commons

By Dmitry Nosachev [CC BY-SA 4.0], from Wikimedia Commons

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

Problem: Local booting is more complex



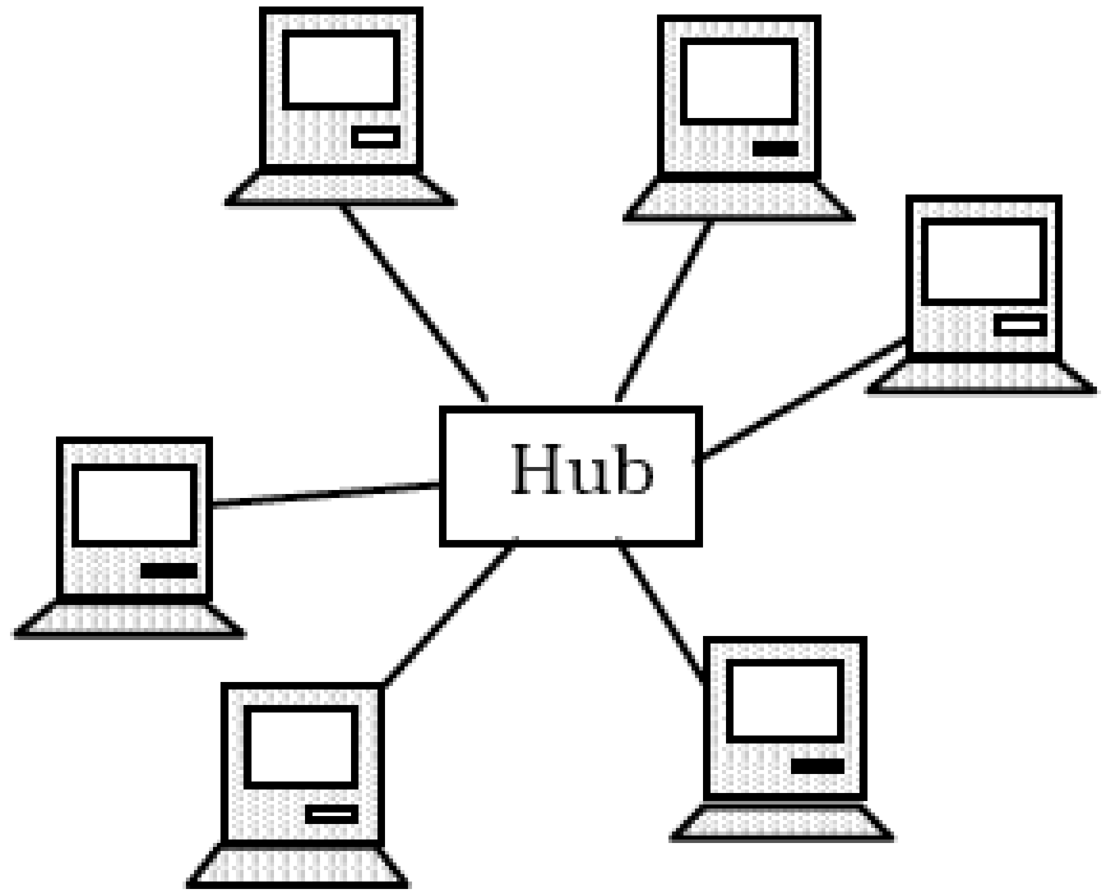


By Toniperis [CC BY-SA 4.0], from Wikimedia Commons

By Dmitry Nosachev [CC BY-SA 4.0], from Wikimedia Commons

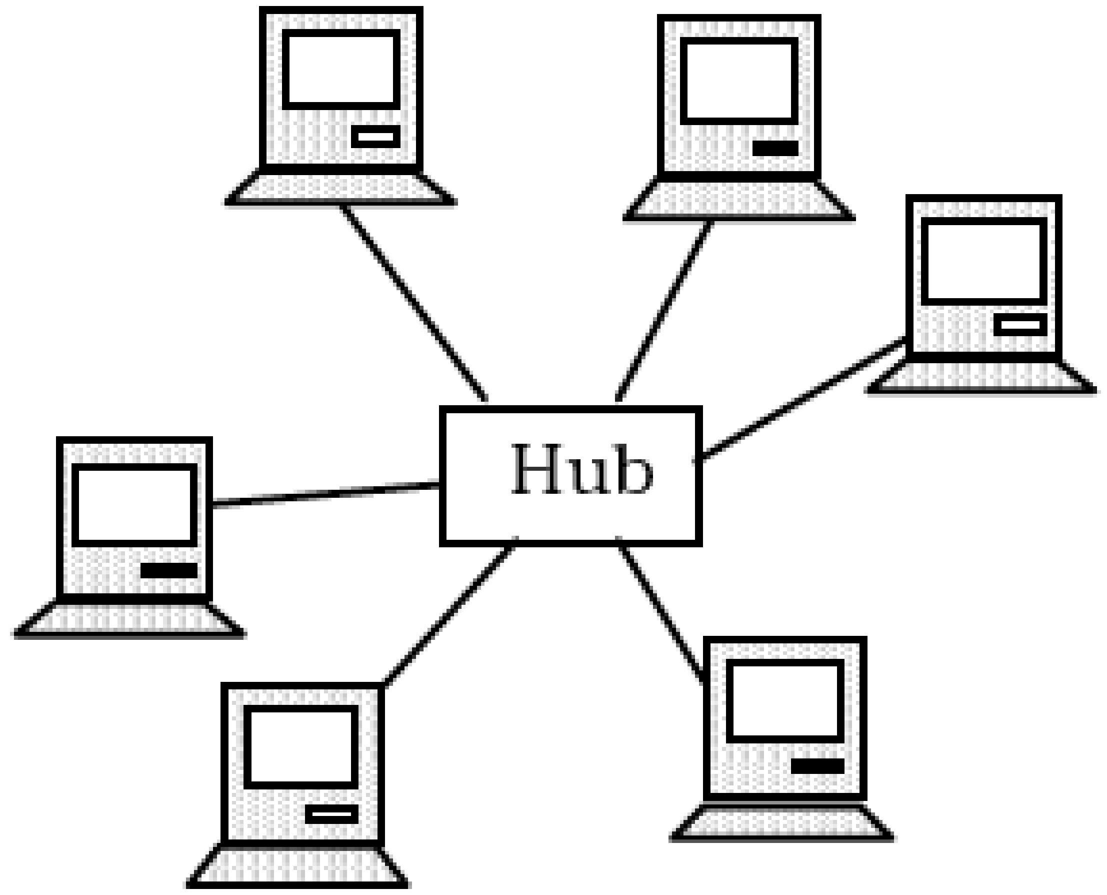
Then	Now
Few interfaces	Many interfaces and protocols
Simple, low-speed links	High-speed links
Blindly execute MBR (CHS 0/0/1)	Decrypt, verify, mount filesystem

Problem: Network booting is more complex



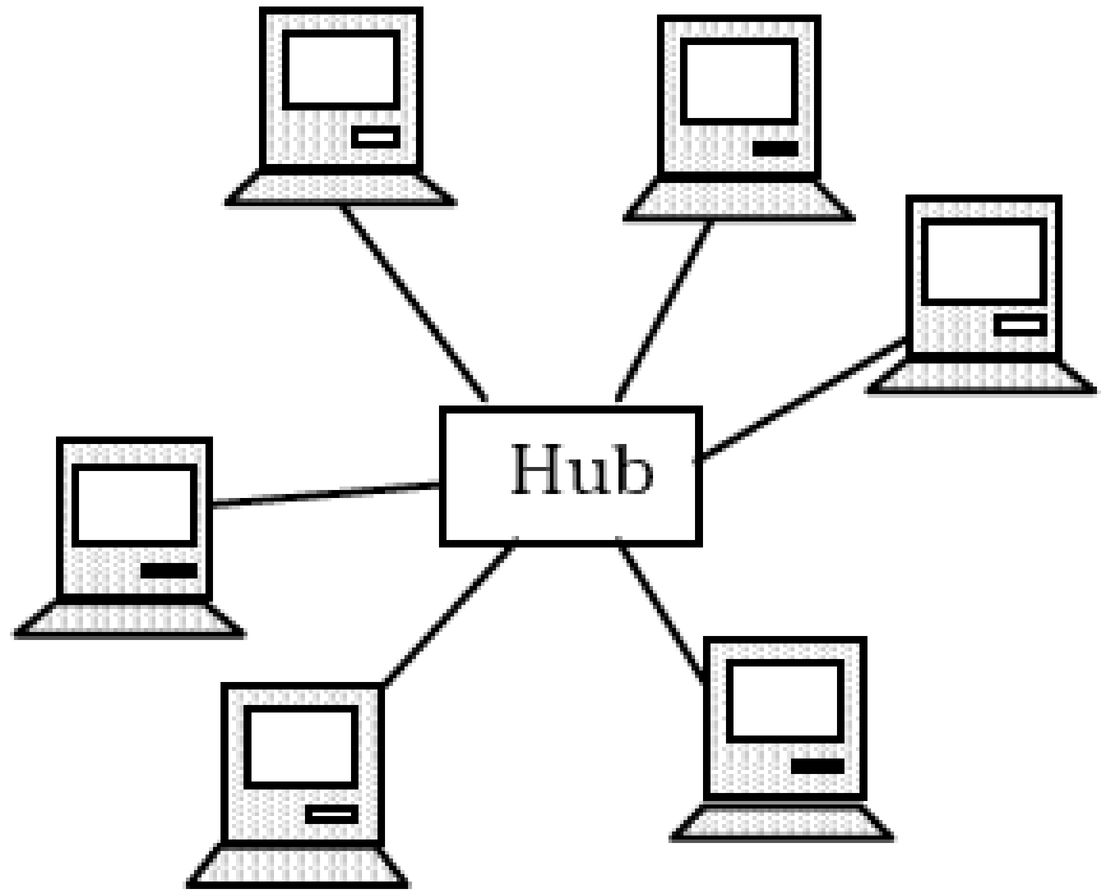
Then	Now
Small, trusted networks	Global, untrusted networks

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

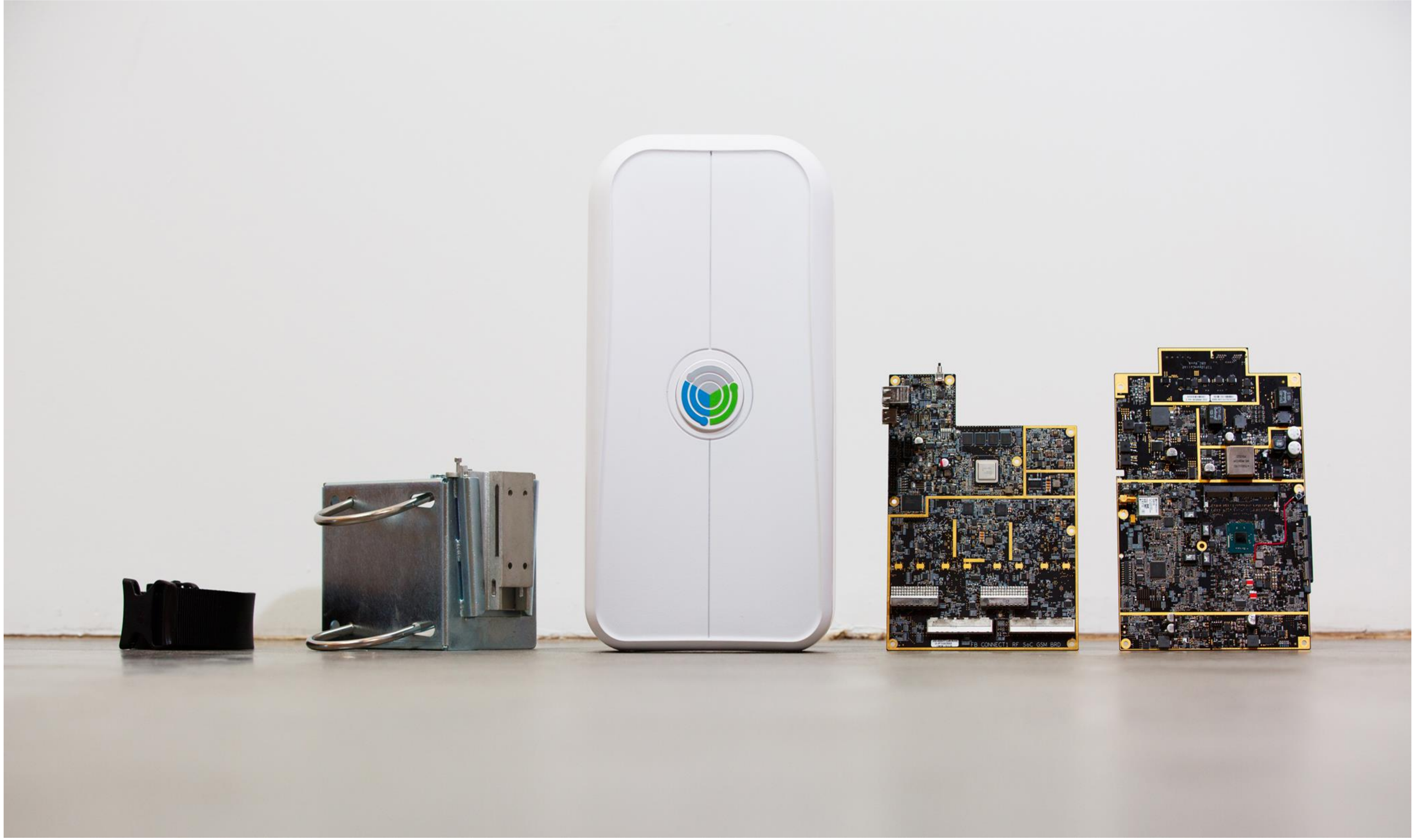


Then	Now
Small, trusted networks	Global, untrusted networks
Few, simple interfaces and protocols	Many interfaces and protocols
TFTP/PXE, security an afterthought	TLS/HTTPS, designed for security

Why is Facebook interested in this

That's a lot of servers & networking gear

Why is Facebook interested in this



TELECOM INFRA PROJECT

Building out infrastructure for the world

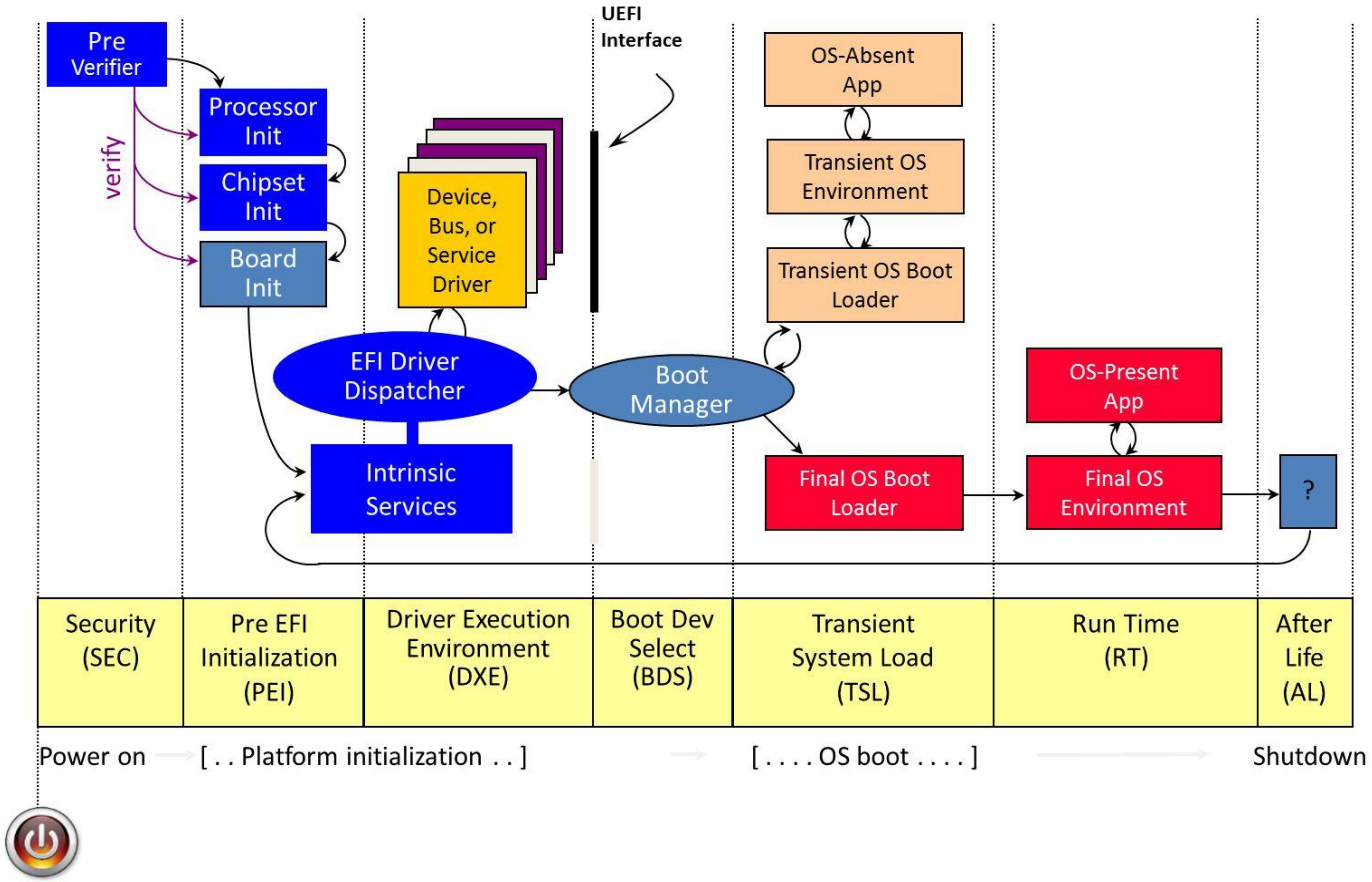
Why is Facebook interested in this

- Many platforms, many firmwares, many bugs, many headaches
  - And it's getting more complicated as we're growing
- Firmware has not received as much attention as HW and SW
  - As our hardware portfolio grows, this must change
- Firmware is an important part of every product
  - Need in-house expertise, reusability, uniformity, maintainability
- Be open
  - Make it easy to work with partners and external parties
  - Re-use code across wide-range of products over many generations

Many parts to this puzzle…

UEFI Boot Flow

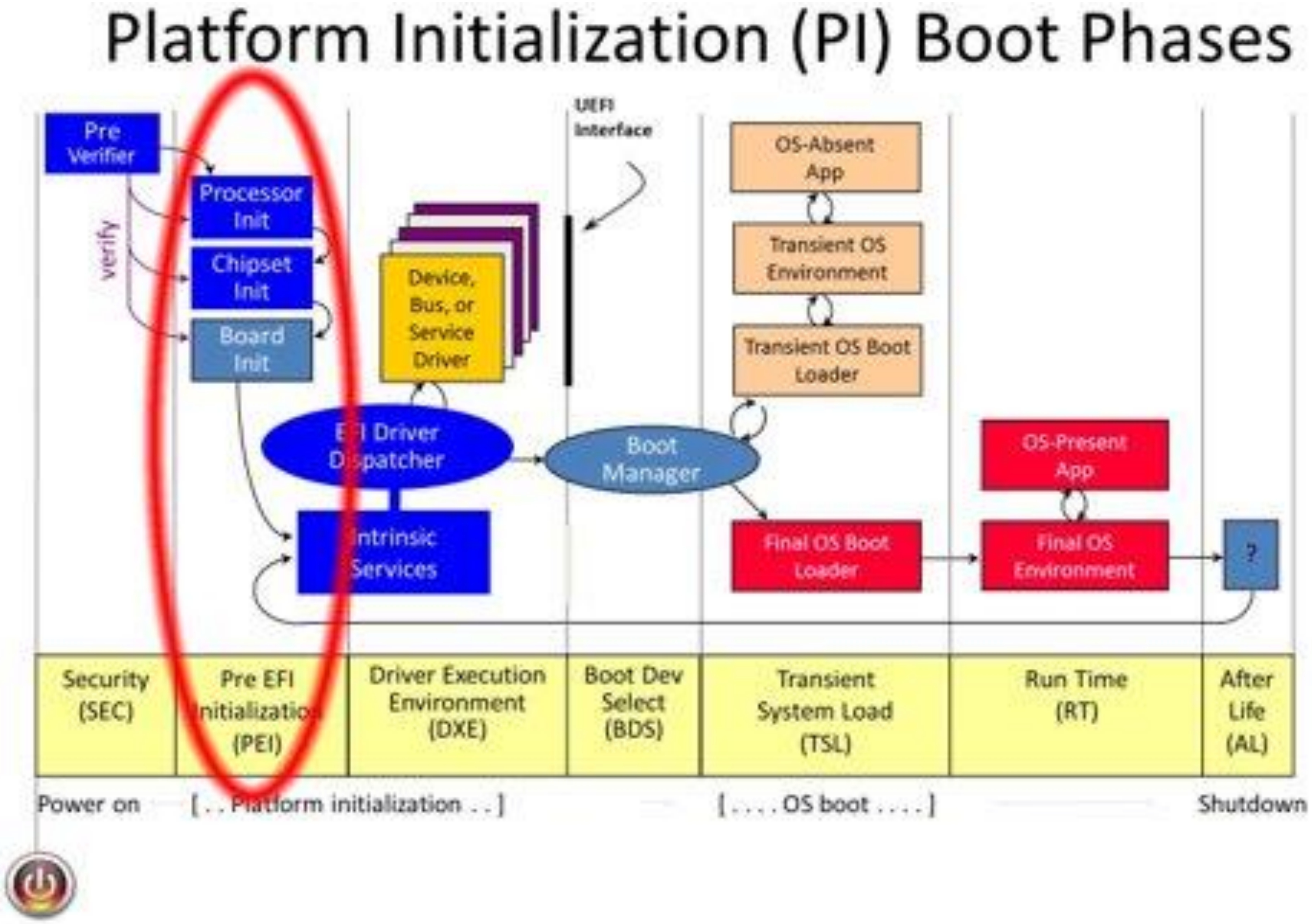
Platform Initialization (PI) Boot Phases



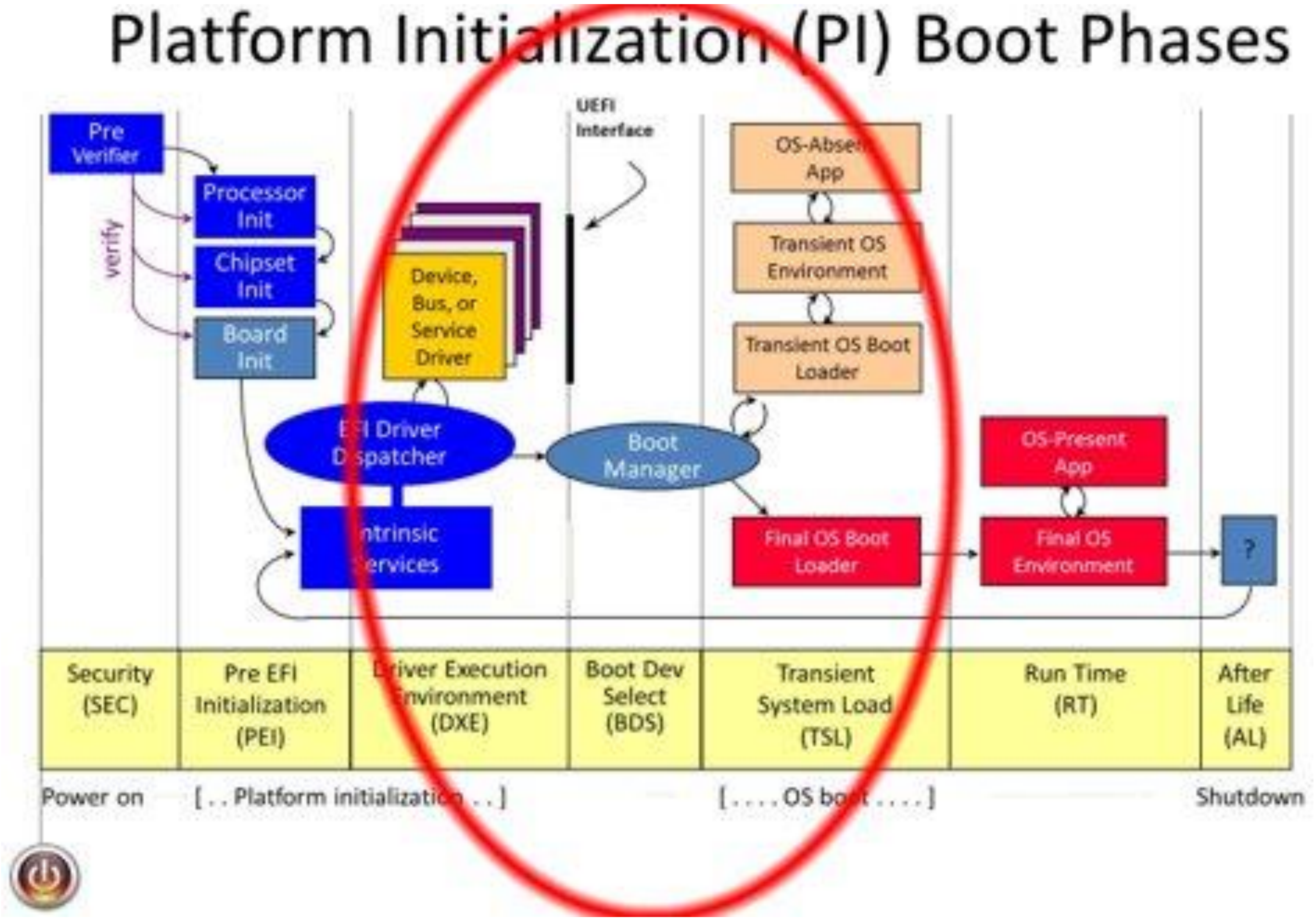
Security (SEC)	Pre EFI Initialization (PEI)	Driver Execution Environment (DXE)	Boot Dev Select (BDS)	Transient System Load (TSL)	Run Time (RT)	After Life (AL)

Power on → [. . Platform initialization . .] → [. . . . OS boot . . . .] → Shutdown

UEFI Boot Flow

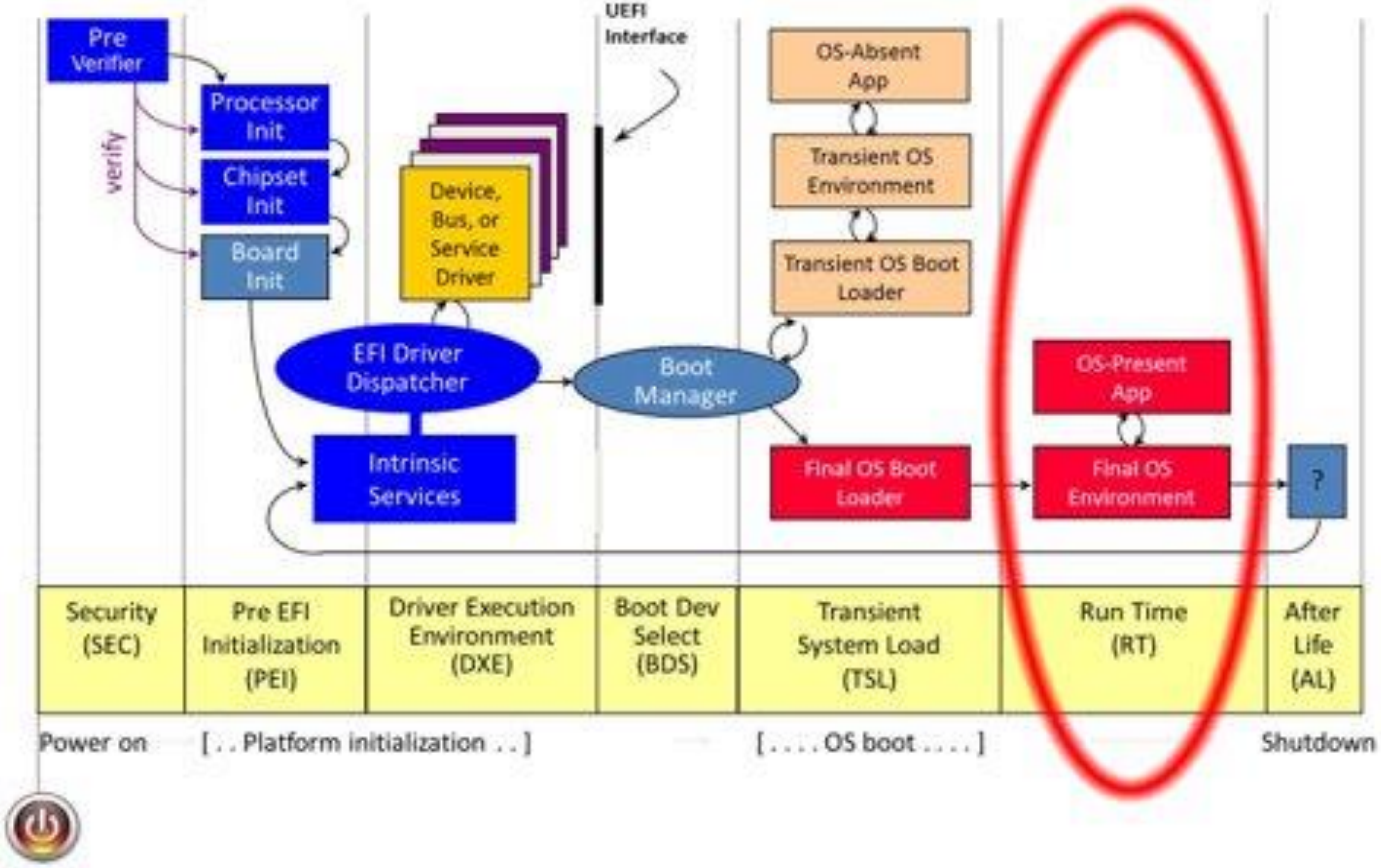


UEFI Boot Flow

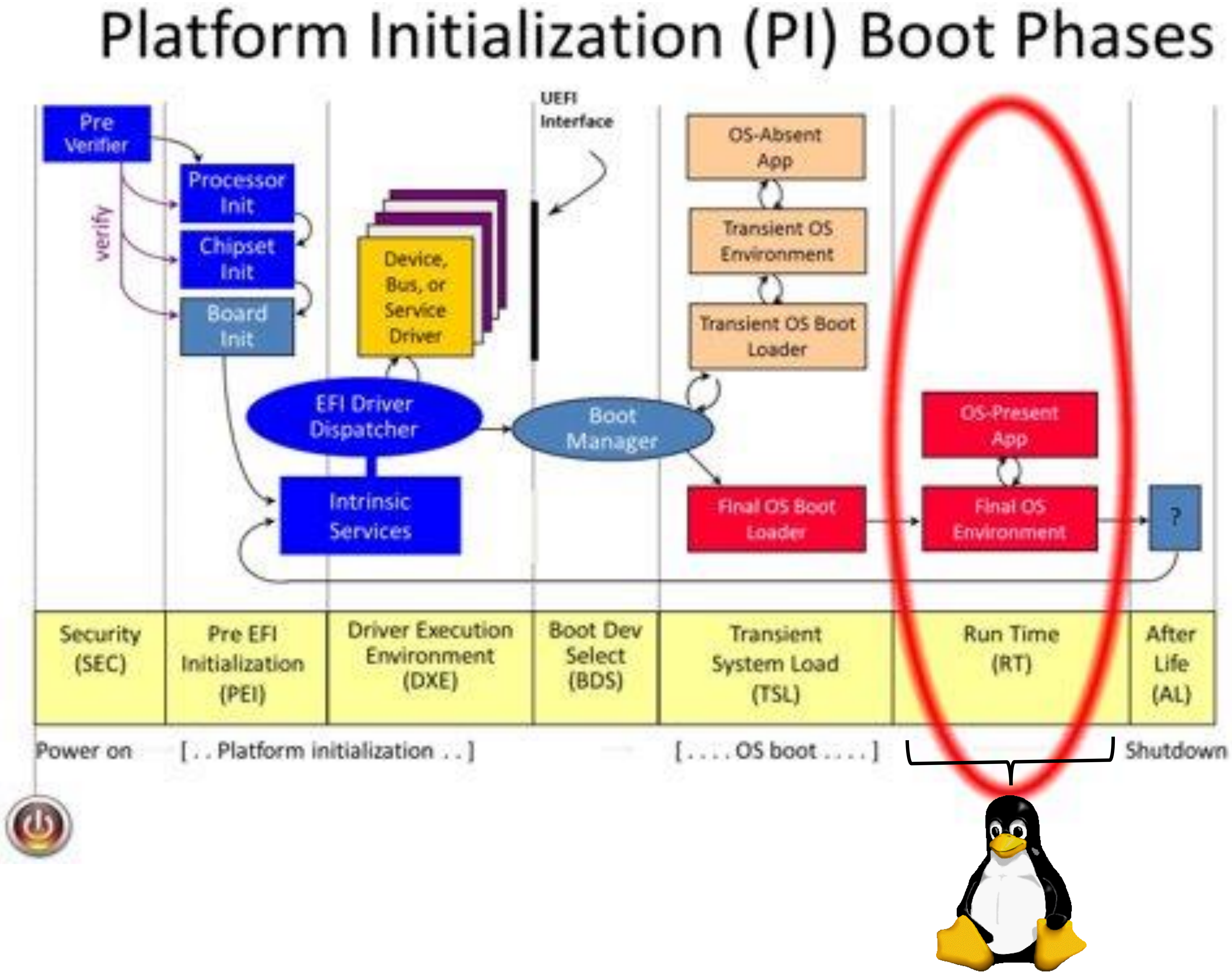


UEFI Boot Flow

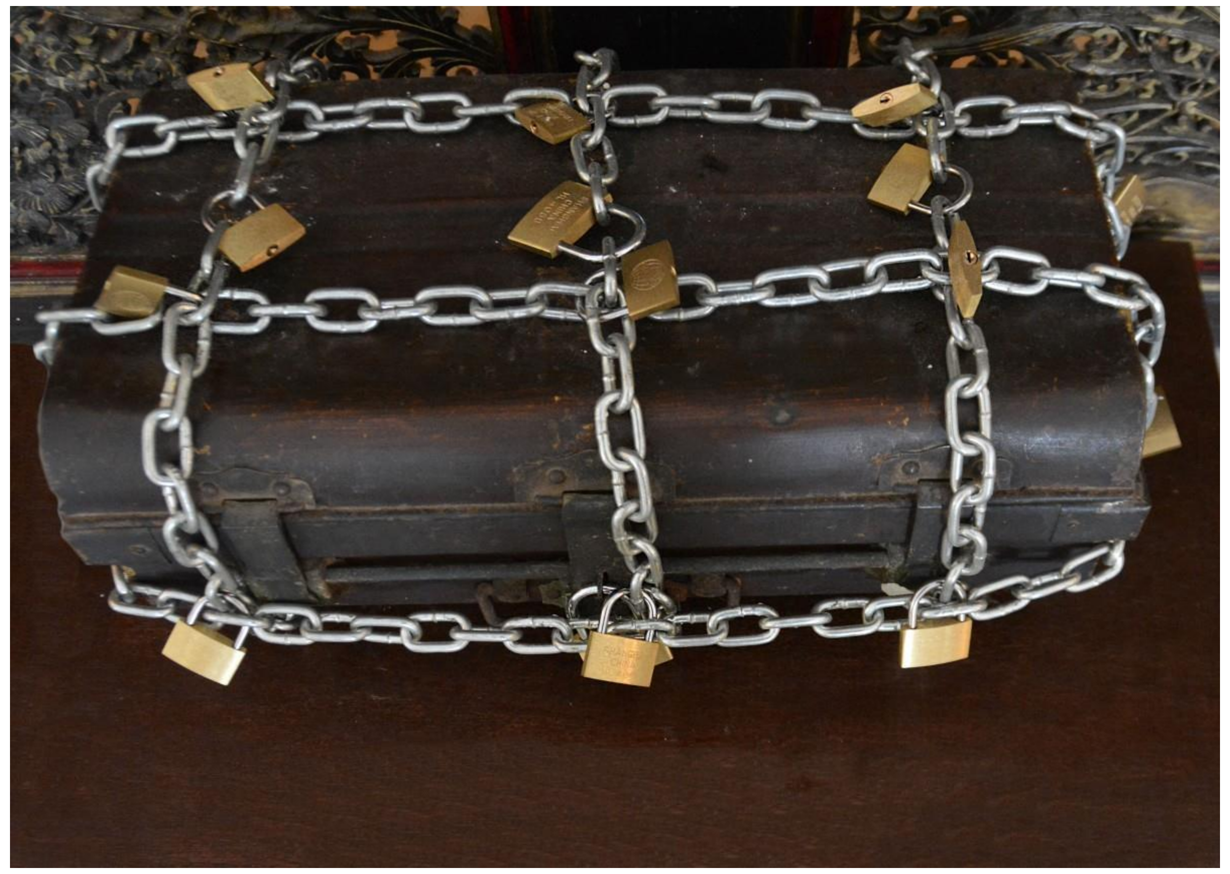
Platform Initialization (PI) Boot Phases



UEFI Boot Flow

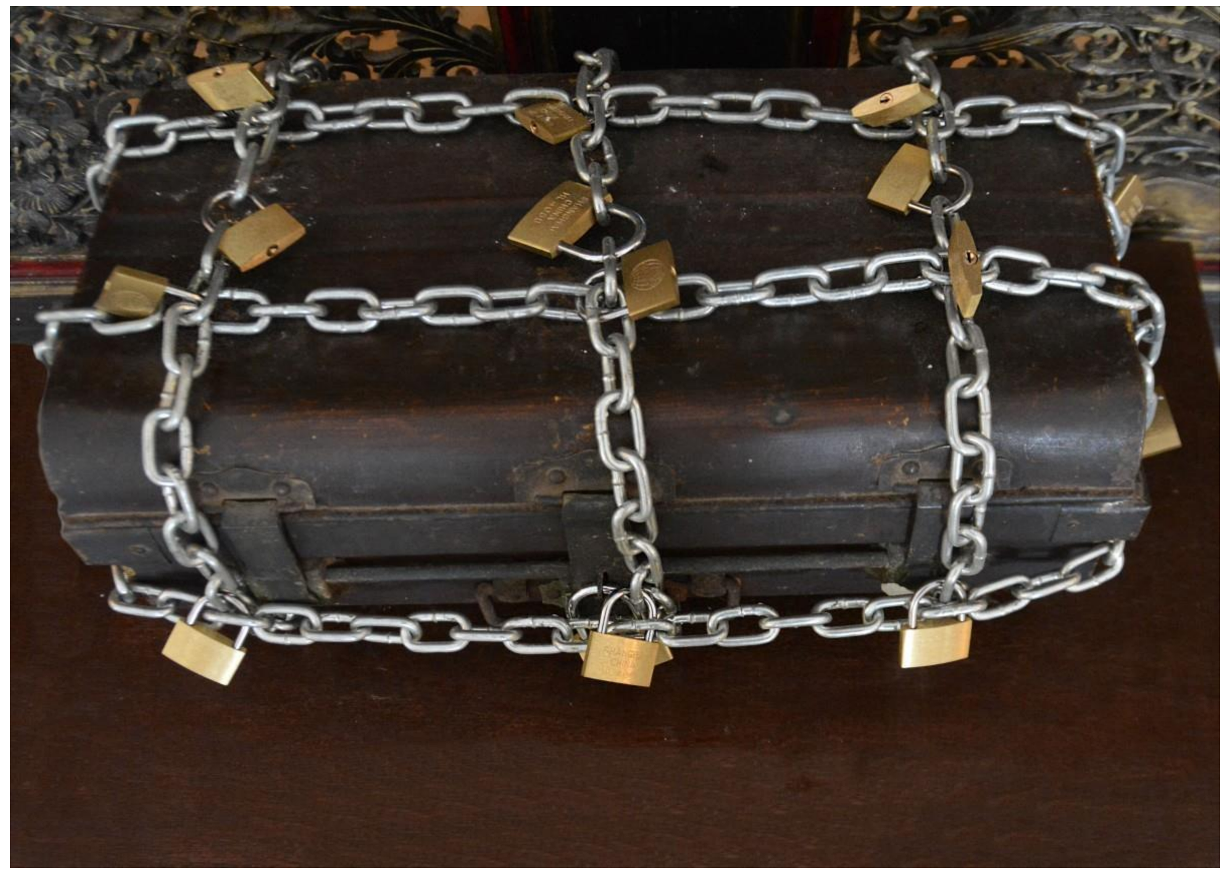


The Result: SysFW Is Complex



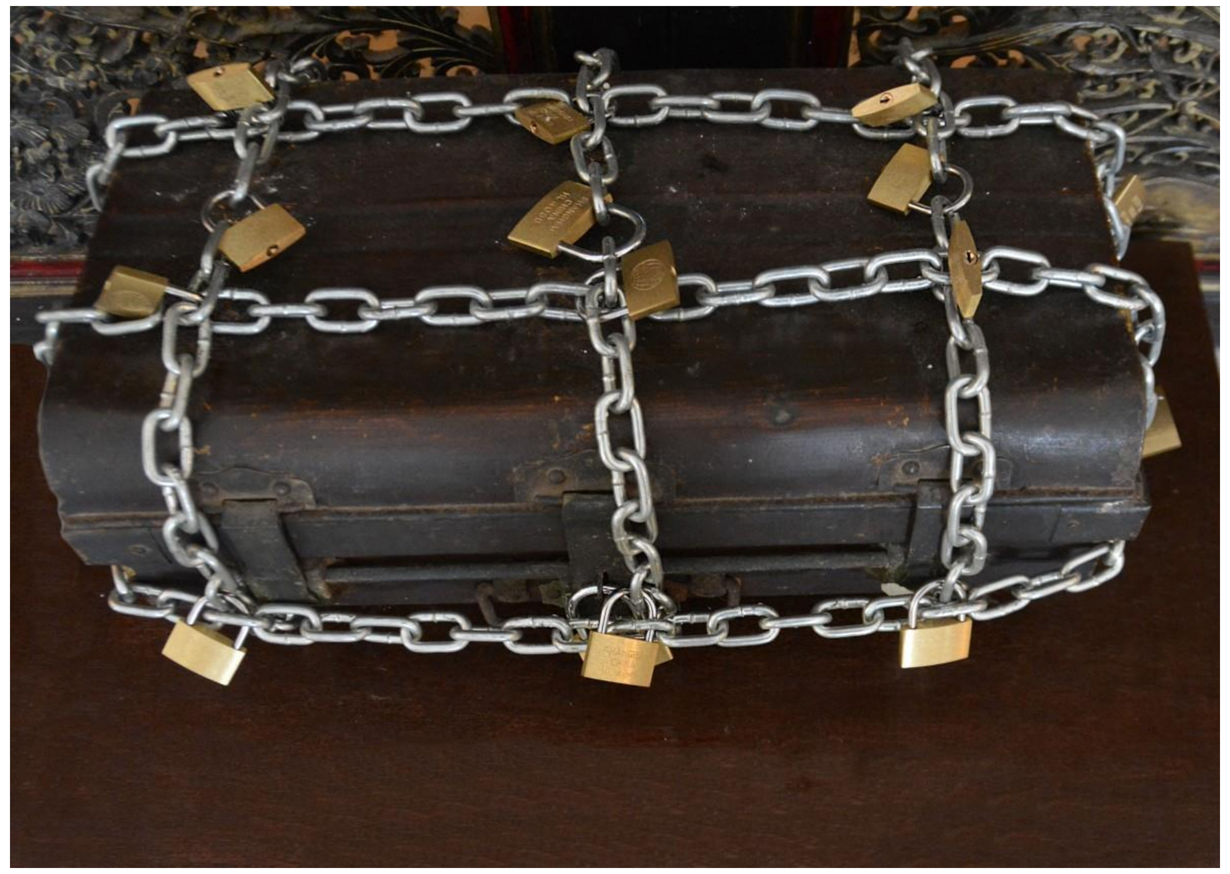
Then/Now	Now/Future
SysFW contains an OS	

The Result: SysFW Is Complex



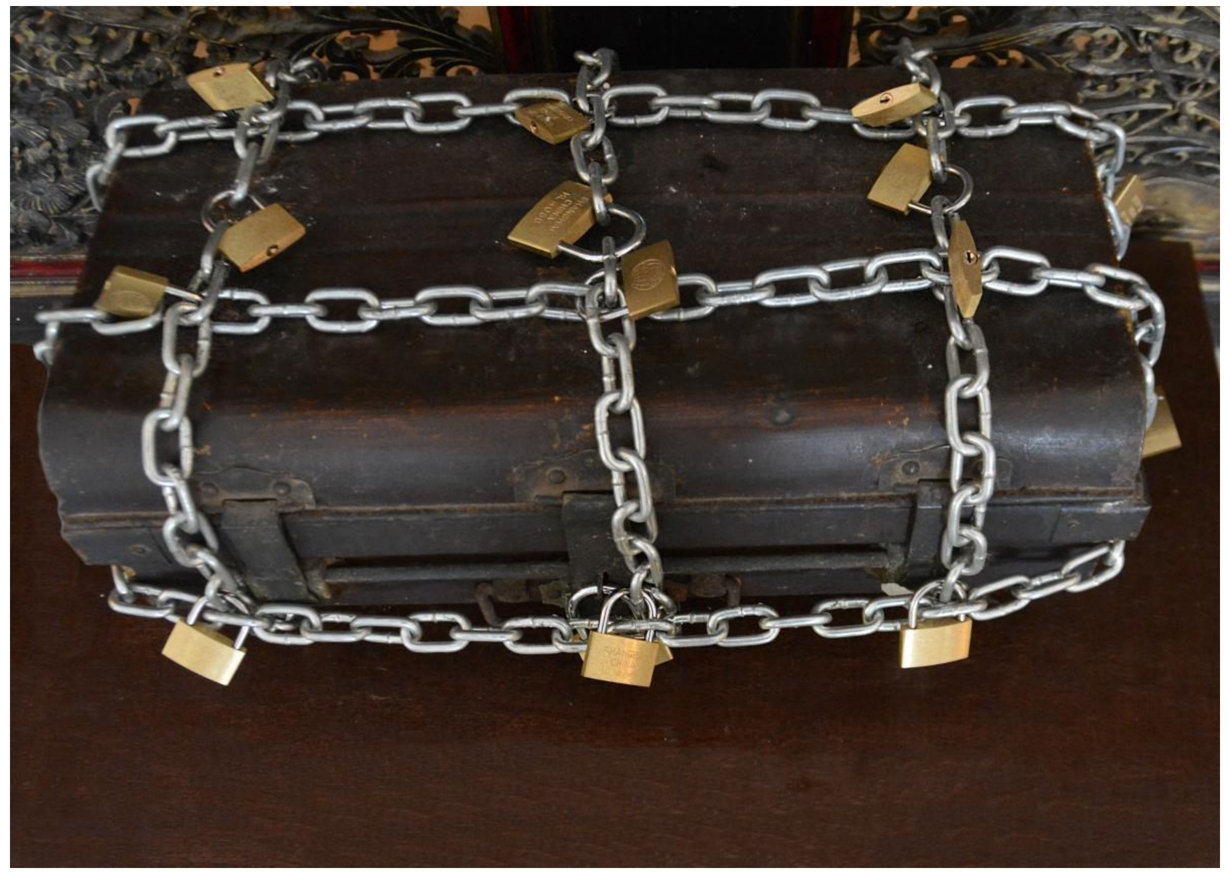
Then/Now	Now/Future
SysFW contains an OS	
Opaque	
Proprietary ecosystem	

The Result: SysFW Is Complex



Then/Now	Now/Future
SysFW contains an OS	
Opaque	
Proprietary ecosystem	
Vendor-specific tooling	
Product-specific	

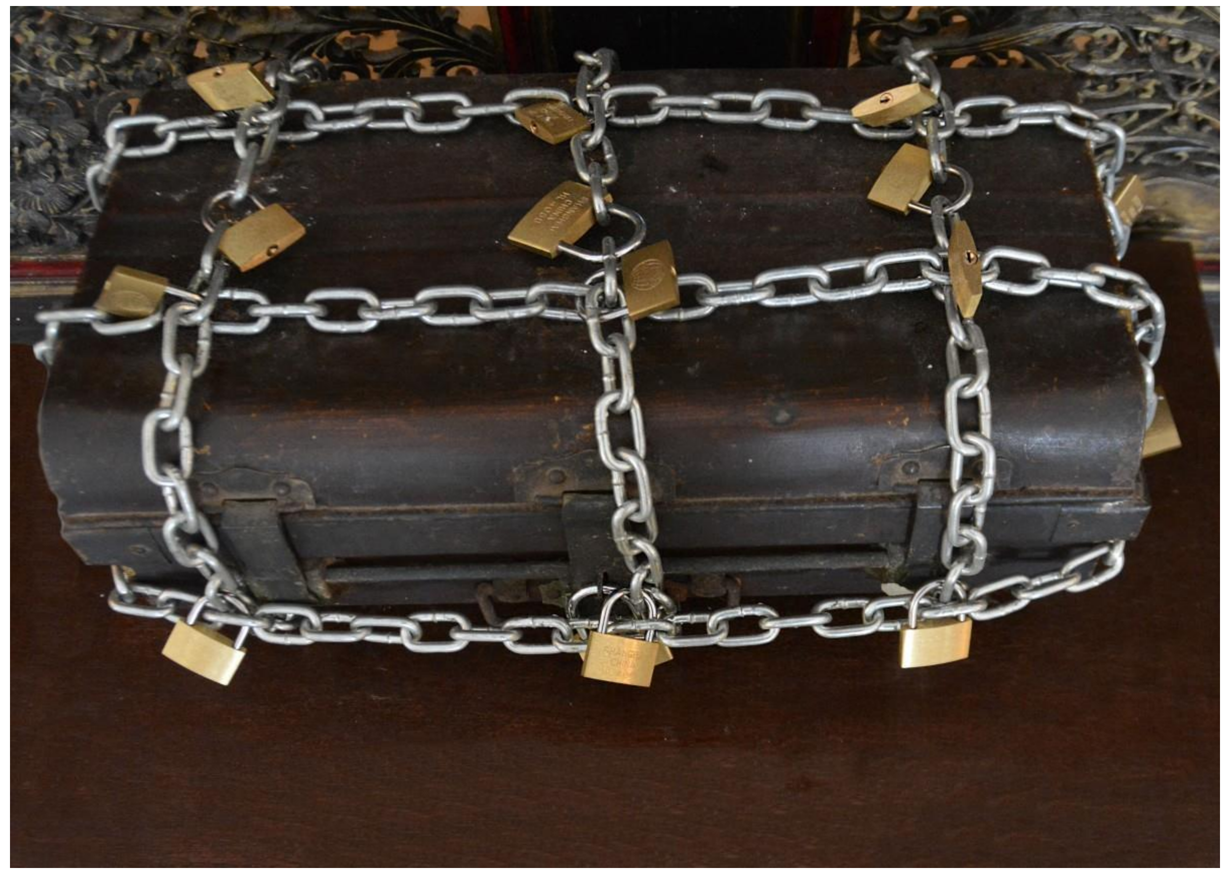
Solution: Open System Firmware



LinuxBoot

Then/Now	Now/Future
SysFW contains an OS	**Open-source, e.g. LinuxBoot**
Opaque	
Proprietary ecosystem	
Vendor-specific tooling	
Product-specific	

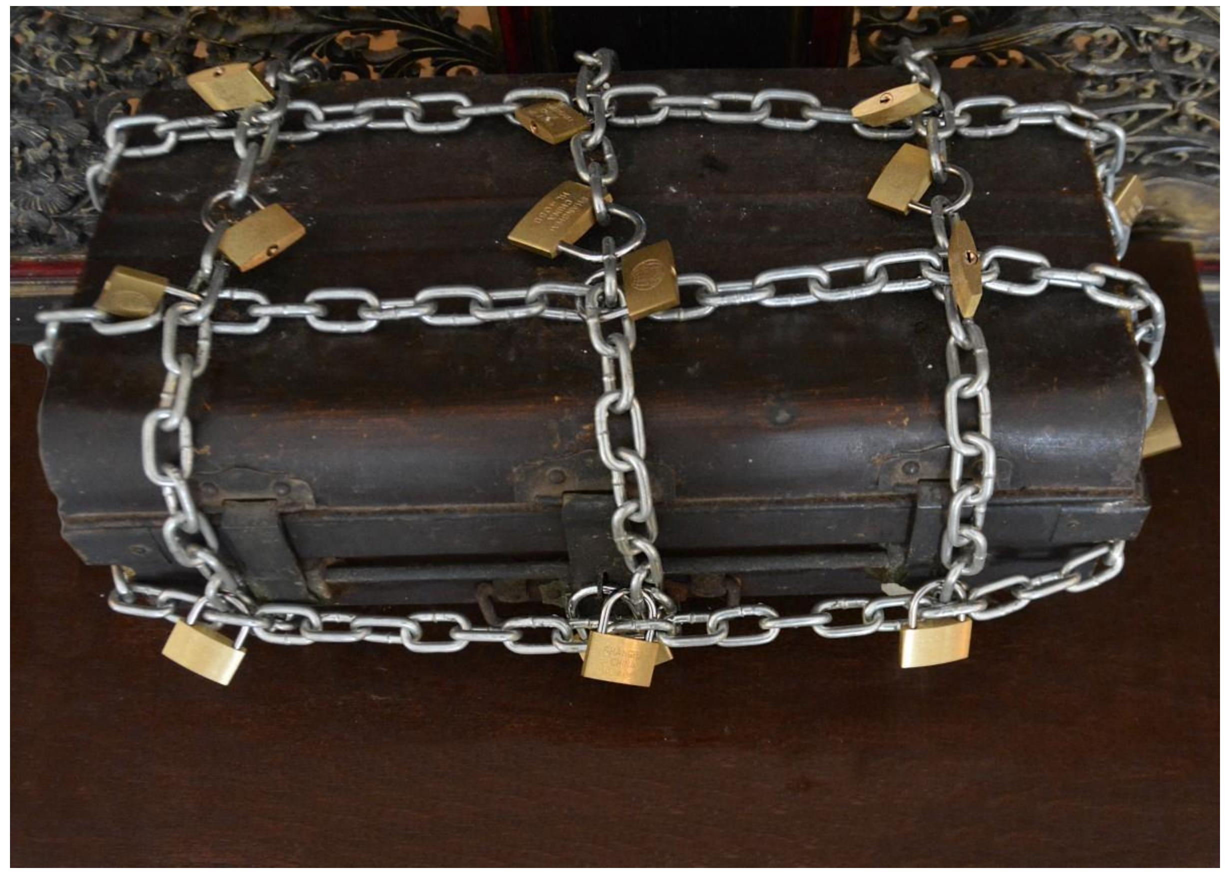
Solution: Open System Firmware



LinuxBoot

Then/Now	Now/Future
SysFW contains an OS	Open-source, e.g. LinuxBoot
Opaque	**Well-understood at FB**
Proprietary ecosystem	**Auditable, debuggable**
Vendor-specific tooling	
Product-specific	

Solution: Open System Firmware



LinuxBoot

Then/Now	Now/Future
SysFW contains an OS	Open-source, e.g. LinuxBoot
Opaque	Well-understood at FB
Proprietary ecosystem	Auditable, debuggable
Vendor-specific tooling	**Open-source tools**
Product-specific	**Portable, re-usable**

OCP Open System Firmware

Two current workstreams:

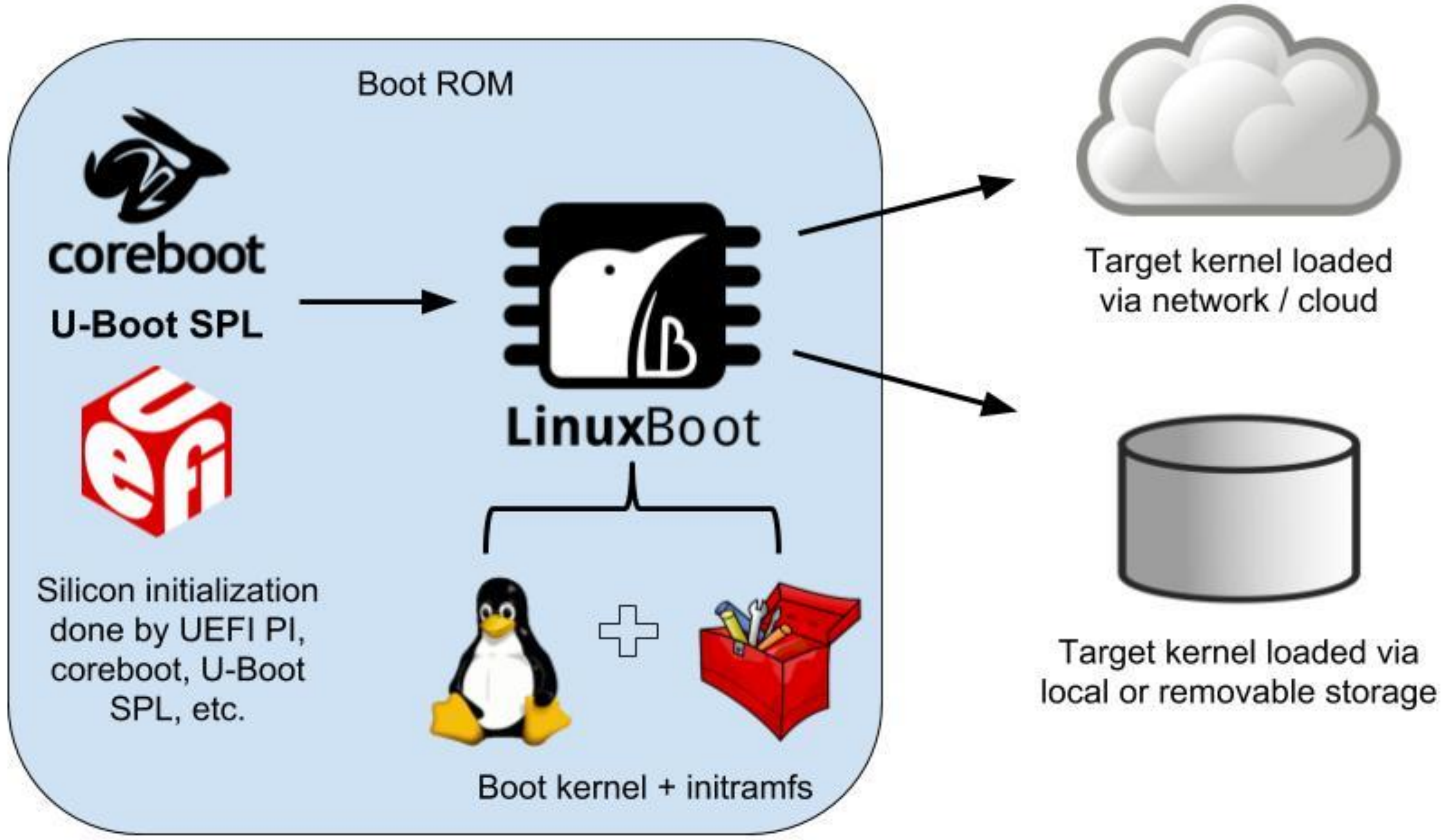
- OpenEDK – Buildable UEFI implementation
- LinuxBoot – Buildable hybrid of coreboot (or UEFI) and Linux
- Same/similar HW init, difference is the OS loading part

Which to use?

- Depends on your use case
- Both aim to offer fully buildable, customizable boot solutions
- Opportunities to share code between the two
- Especially for the early boot phases, runtime services (ACPI, RAS, etc).

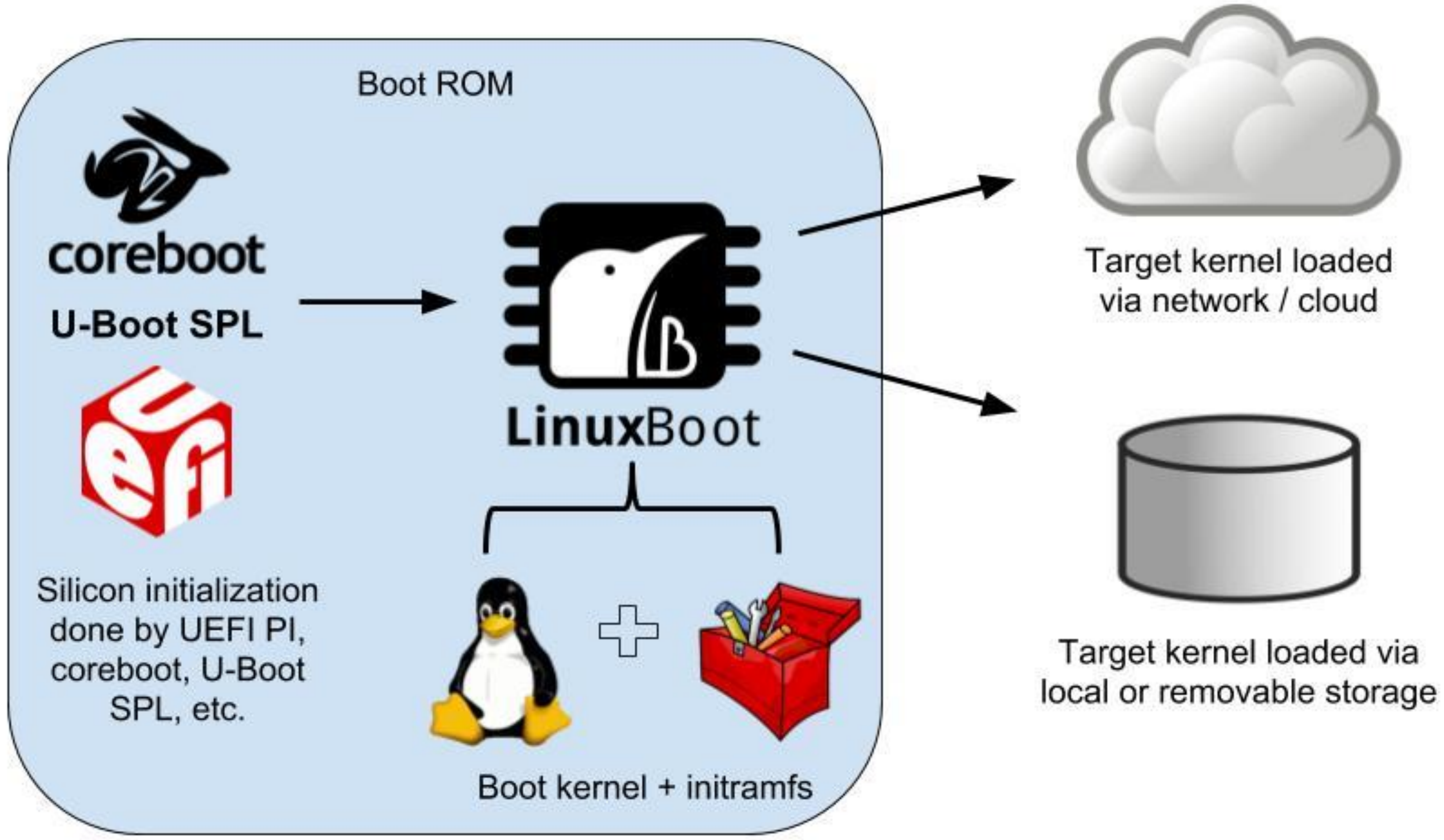
LinuxBoot Approach: Let Linux Do It

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


Boot ROM
coreboot
U-Boot SPL
Silicon initialization done by UEFI PI, coreboot, U-Boot SPL, etc.
LinuxBoot
Boot kernel + initramfs
Target kernel loaded via network / cloud
Target kernel loaded via local or removable storage

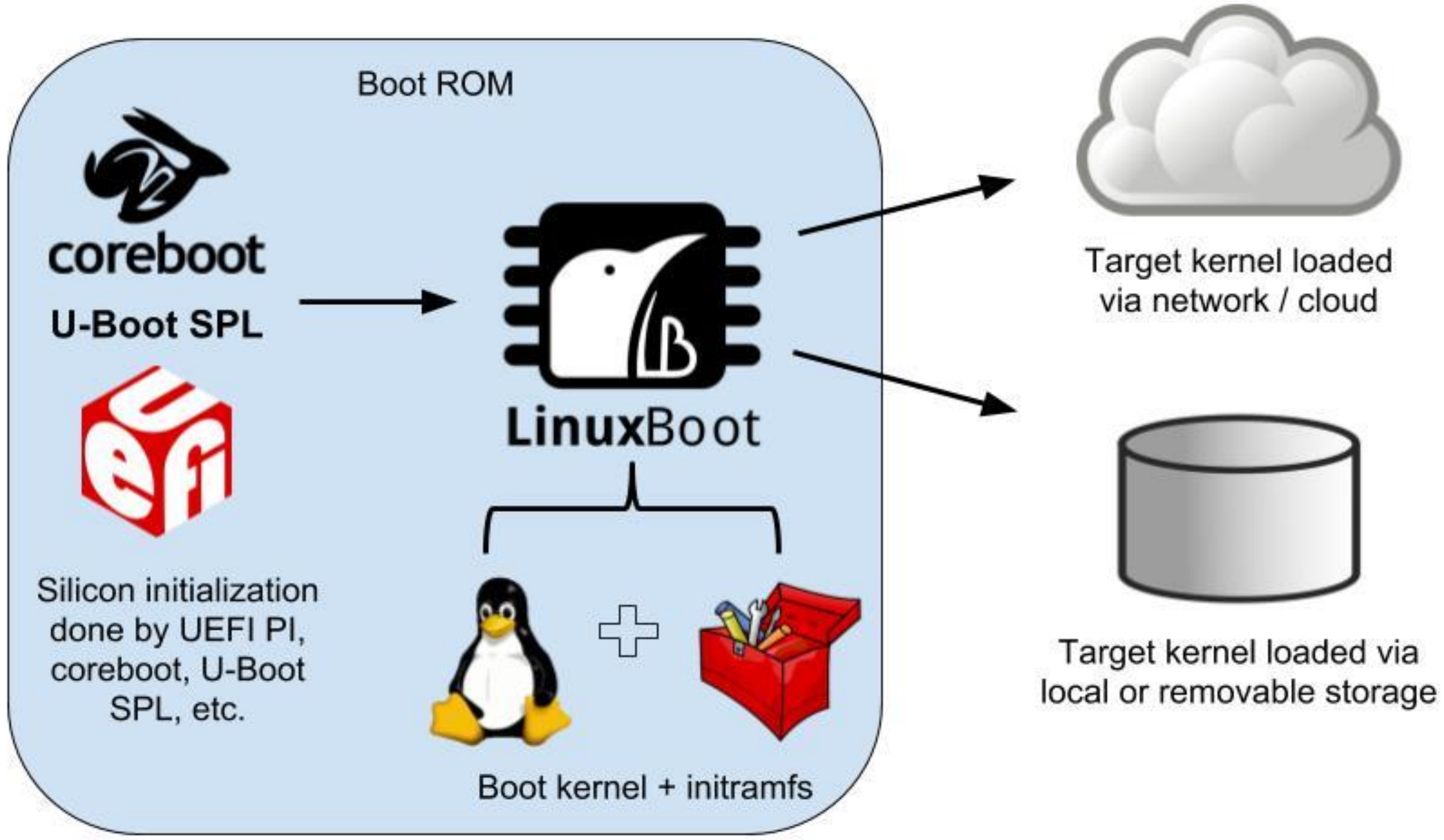
LinuxBoot Approach: Let Linux Do It

- 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**

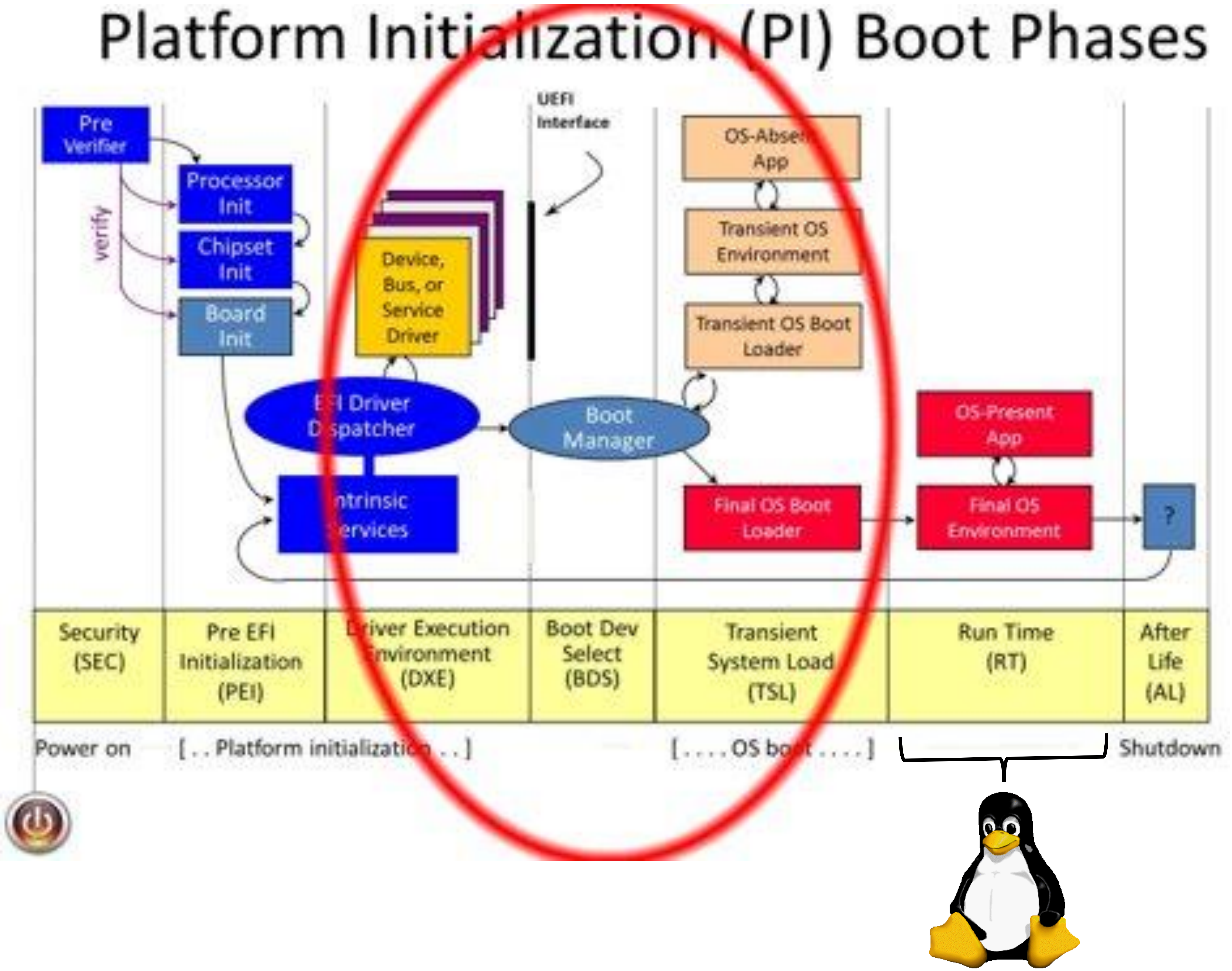

Boot ROM
coreboot
U-Boot SPL
Silicon initialization done by UEFI PI, coreboot, U-Boot SPL, etc.
LinuxBoot
Boot kernel + initramfs
Target kernel loaded via network / cloud
Target kernel loaded via local or removable storage

LinuxBoot Approach: Let Linux Do It

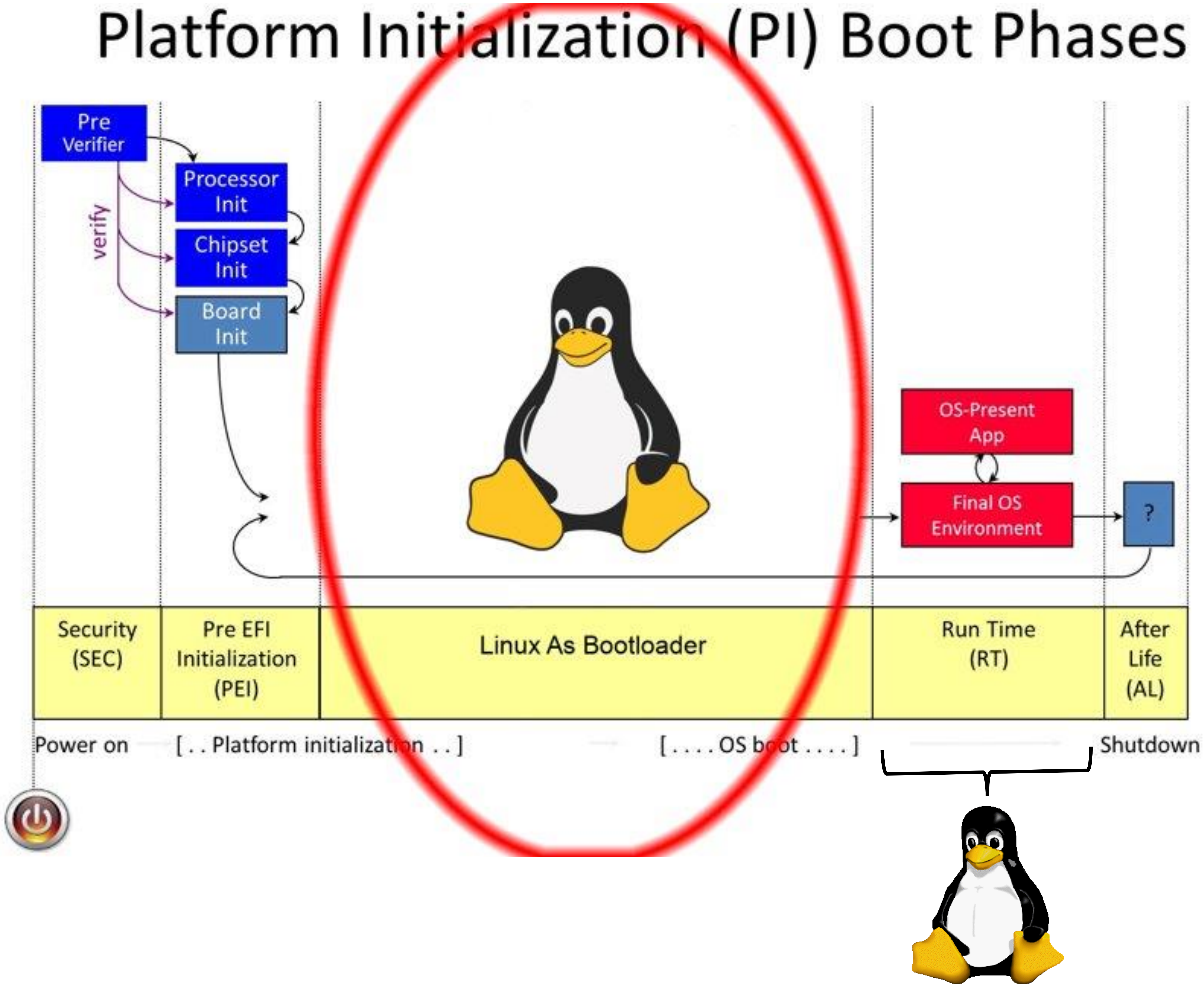
- Put a kernel+initramfs in boot ROM
- Do minimal silicon init and jump to Linux as soon as possible
- Use Linux to boot Linux
- Production-quality drivers, networking
- Add features + tools as needed
- Debug, build, deploy on our schedule
- **Flexible security architecture**
- **Boot in seconds, not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


Boot ROM
coreboot
U-Boot SPL
Silicon initialization done by UEFI PI, coreboot, U-Boot SPL, etc.
LinuxBoot
Boot kernel + initramfs
Target kernel loaded via network / cloud
Target kernel loaded via local or removable storage

So we want to turn this…



...into this



Platform Initialization (PI) Boot Phases

Pre Verifier

verify

Processor Init

Chipset Init

Board Init

OS-Present App

Final OS Environment

?

Security (SEC)	Pre EFI Initialization (PEI)	Linux As Bootloader	Run Time (RT)	After Life (AL)

Power on [. . Platform initialization . .] [. . . . OS boot . . . .] Shutdown

LinuxBoot @ Facebook

OS Provisioning

LinuxBoot

@ Facebook - OS Provisioning

Production Engineering: scalability, reliability, security, speed

OS Provisioning team:

- Automatically install the OS on our machines
- Simple on a single machine
- Can be very complex at large scale
  - Lots of moving parts
  - Network introduces noise

LinuxBoot @ Facebook – Provisioning workflow

How does it look from the host firmware perspective?

- Power on
- DHCPv6 client: acquire network configuration
- TFTP client: download Network Boot Program
- Execute the installer

@ Facebook – Provisioning issues

- Some DHCP and TFTP implementations are buggy
  - and TFTP is slow and unreliable
- Different machine types have different firmwares
  - each firmware has its own set of bugs
- Most of the times PXE booting works
  - but at scale a small fraction of errors can translate into a lot of operations

- What we need:
  - Reliable clients
  - Better protocols
  - Control the implementation: know what we run, fix it, improve it

LinuxBoot @ Facebook – Improve provisioning

LinuxBoot can simplify provisioning

- Tested DHCP and TFTP implementations
- Better protocols: HTTP and HTTPS
- Consistent firmware versions across the fleet
- We know and control the firmware

We expect to largely reduce netboot failures in provisioning with LinuxBoot

LinuxBoot

@ Facebook - Firmware upgrades

Now:

- Upgrading firmware now depends on vendors
- Different vendors have different standards and response time
- Debugging closed source firmware can be hard
- Vendors may be unable to reproduce the issue in their infra
- Once the updated firmware is ready, we need to run our validation
  - The time between bug identification and roll-out to prod can be very long

We want to speed up the upgrade process and enable in-house debugging

LinuxBoot @ Facebook - Firmware testing

Now:

- First phase of firmware testing is done by vendors
  - Vendor tests are black-box to us
- Then we run our own tests
  - Once the tests pass, a firmware is released for production use
- Then eventually deploy to production
- If we find a bug in production we have to go back to the vendor, and repeat the above process

With LinuxBoot we can speed up and minimize the number of steps.

We can also enable instrumentation for firmware testing

LinuxBoot @ Facebook – Openness

Open Source means:

- Auditability
- Debuggability
- Transparency in the security model
- Portability
- Modern development practices
- Collaboration with the community

LinuxBoot

@ Facebook – Code reuse

LinuxBoot is modular and multiplatform

Can run on datacenter servers, but the same code can run on completely different platforms

Example:

- At Facebook: Datacenter servers, OpenCellular
- For you to try at home: coreboot-supported platforms such as Chromebooks

LinuxBoot

@ Facebook – not just servers

Use cases: booting a datacenter switch, a server, or a cellular base station

- Shared code
- Per-platform continuous firmware build and testing
- Run-time configuration determines the machine's behaviour
  - Boot config, network boot program, etc
- Common, open system firmware and tools
  - Operators and communities must be able to build and maintain sources
  - Re-use code for datacenter and telecom infrastructure
- Adaptable to different threat models

LinuxBoot

@ Facebook – not just firmware

- LinuxBoot is not just for firmware
- Multiple reusable components
- Linux, u-root and systemboot can be used also as
  - Network installer: YARD
  - Pre-provisioned bootloader/installer: ProvLauncher

Systemboot

@ Facebook - Systemboot

- Systemboot is a "distro" that implements a bootloader
- Based on u-root, that we are contributors of
- Written in Go
- Provides different tools for different boot scenarios

- The goal is to create components that we can iterate fast on
  - Generic and stable ones will be contributed back to u-root

@ Facebook – Inside Systemboot

- **LinuxBoot VPD:** non-volatile key-value store
- **netboot:** boot a kernel over the network using DHCPv6 or SLAAC, HTTP(s) and kexec
- **localboot:** boot a kernel from disk using Grub/Grub2 or custom location on disk
- **Booters interface:** define your custom boot method or policy
- **TPMTool:** high-level TPM library and command-line utility
- **uinit:** wrap all the above in in an executable used as entry point

@ Facebook – VPD library

- Vital Product Data
- Key-value store on the flash chip
- Based on ChromeOS's VPD
- Used for non-volatile storage, similar to UEFI variables
  - We use it to store boot configuration (netboot and localboot config)
  - Can be extended to other uses
  - If you don't like VPD, can be easily swapped out

@ Facebook – VPD for boot order

- Boot order is stored in VPD variables
- Value in JSON format
- Examples:

```
Boot0000={
    "type":"netboot",
    "method":"dhcpv6",
    "mac":"00:fa:ce:b0:0c:00"
}
```

```
Boot0001={
"type": "localboot",
"method": "path",
"kernel": "/path/to/kernel",
"device_guid": "..."
}
```

@ Facebook – netboot

- Used to boot a kernel downloaded over the network
- Three phases
  - Acquire network configuration (DHCPv6, SLAAC or DHCPv4)
  - Download kernel via HTTP or HTTPS
  - Kexec the downloaded kernel

@ Facebook – localboot

- Similar to netboot, but boot from local storage
- Two way of operating, Grub mode and Path mode
- Grub mode
  - Scan local disks
  - Find Grub/Grub2 configuration
  - Identify kernel, initramfs and kernel command line
  - kexec
- Path mode (using VPD variables)
  - Look for specific disk and partition
  - Find kernel at specific path
  - Optionally specify initramfs and kernel command line
  - kexec

@ Facebook – Booters interface

- A generic interface to create new booters
  - netboot and localboot are based on it
  - New booters can implement it
  - You can implement higher level policies, e.g. recovery from failed boot
- Very simple
  - define TypeName() and Boot() methods
  - Define JSON format by extending the generic booter JSON
  - Register the booter, and systemboot will pick it up

@ Facebook – Example: netbooter

- https://github.com/systemboot/systemboot/blob/master/pkg/booter/netbooter.go
- Define NetBooter structure with JSON annotations, define NewNetbooter to parse JSON into NetBooter
- Then implement Boot() and TypeName():

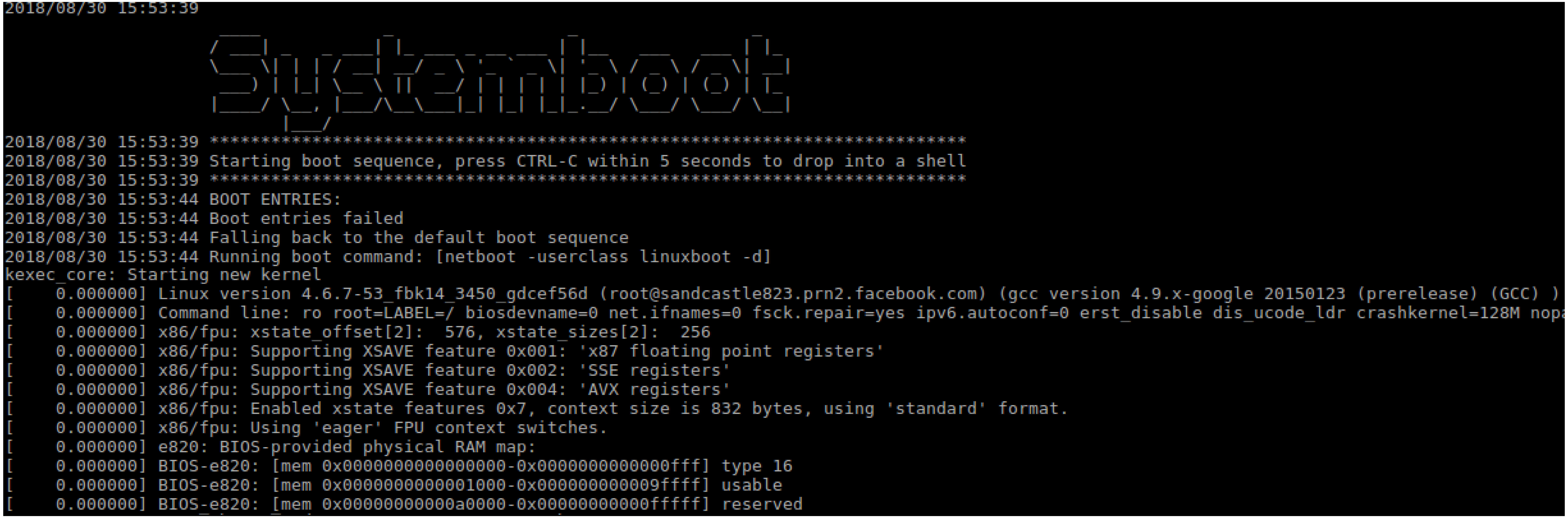
```
// Boot will run the boot procedure. In the case of NetBooter, it will call the
// `netboot` command
func (nb *NetBooter) Boot() error {
    bootcmd := []string{"netboot", "-d", "-userclass", "linuxboot"}
    log.Printf("Executing command: %v", bootcmd)
    cmd := exec.Command(bootcmd[0], bootcmd[1:]...)
    cmd.Stdin, cmd.Stdout, cmd.Stderr = os.Stdin, os.Stdout, os.Stderr
    if err := cmd.Run(); err != nil {
        Return  fmt.Errorf("Error executing %v: %v", cmd, err)
    }
    return nil
}

// TypeName returns the name of the booter type
func (nb *NetBooter) TypeName() string {
    return nb.Type
}
```

LinuxBoot

@ Facebook – TPMTool

- High-level TPM library
  - Goal: simplify the use of the TPM
  - Based on Google's go-tpm
  - Parts of it have been merged in go-tpm
  - Can show info, take and clear TPM ownership, seal/unseal, dump PCRs, pre-calculate hashes, dump TPM event log, and more
- TPMTool
  - High-level userspace utility for TPM
  - Written by Philipp Deppenwiese / 9elements CyberSecurity
  - See tpmtool.org

@ Facebook – Systemboot demo



```
2018/08/30 15:53:39
  _____           _                 _                 _
 / ____|         | |               | |               | |
| (___  _   _ ___| |_ ___ _ __ ___ | |__   ___   ___ | |_
 \___ \| | | / __| __/ _ \ '_ ` _ \| '_ \ / _ \ / _ \| __|
 ____) | |_| \__ \ ||  __/ | | | | | |_) | (_) | (_) | |_
|_____/ \__, |___/\__\___|_| |_| |_|_.__/ \___/ \___/ \__|
         __/ |
        |___/
2018/08/30 15:53:39 ***************************************************************************
2018/08/30 15:53:39 Starting boot sequence, press CTRL-C within 5 seconds to drop into a shell
2018/08/30 15:53:39 ***************************************************************************
2018/08/30 15:53:44 BOOT ENTRIES:
2018/08/30 15:53:44 Boot entries failed
2018/08/30 15:53:44 Falling back to the default boot sequence
2018/08/30 15:53:44 Running boot command: [netboot -userclass linuxboot -d]
kexec_core: Starting new kernel
[    0.000000] Linux version 4.6.7-53_fbk14_3450_gdcef56d (root@sandcastle823.prn2.facebook.com) (gcc version 4.9.x-google 20150123 (prerelease) (GCC) )
[    0.000000] Command line: ro root=LABEL=/ biosdevname=0 net.ifnames=0 fsck.repair=yes ipv6.autoconf=0 erst_disable dis_ucode_ldr crashkernel=128M nopa
[    0.000000] x86/fpu: xstate_offset[2]:  576, xstate_sizes[2]:  256
[    0.000000] x86/fpu: Supporting XSAVE feature 0x001: 'x87 floating point registers'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x002: 'SSE registers'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x004: 'AVX registers'
[    0.000000] x86/fpu: Enabled xstate features 0x7, context size is 832 bytes, using 'standard' format.
[    0.000000] x86/fpu: Using 'eager' FPU context switches.
[    0.000000] e820: BIOS-provided physical RAM map:
[    0.000000] BIOS-e820: [mem 0x0000000000000000-0x0000000000000fff] type 16
[    0.000000] BIOS-e820: [mem 0x0000000000001000-0x000000000009ffff] usable
[    0.000000] BIOS-e820: [mem 0x00000000000a0000-0x00000000000fffff] reserved
```

Conclusion

- With LinuxBoot **you are in control of the firmware**
- Simpler stack and more capabilities
- Support many platforms and use cases
- We are doing for firmware what Linux has done for the OS
- We are amplifying our firmware development capabilities by **turning Linux engineers into firmware engineers**

OCP
SUMMIT

OPEN.

FOR
BUSINESS.