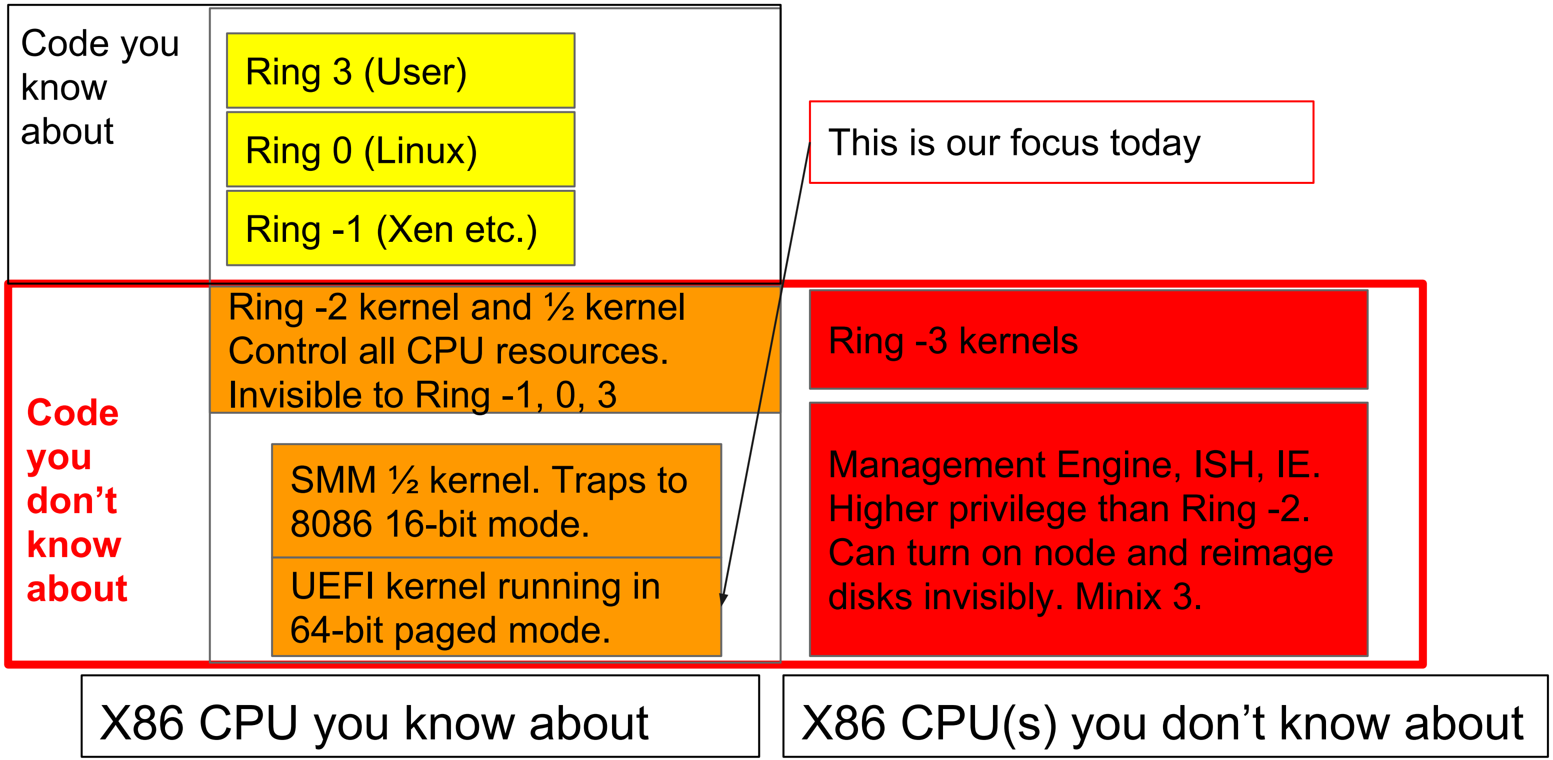
LinuxBoot: Linux as Firmware

Chris Koch, Gan Shun Lim	Google
with Ron Minnich, Ryan O'Leary, Xuan Chen	Google
with Trammell Hudson	Two Sigma
with Jean-Marie Verdun, Guillaume Giamarchi	Horizon Computing
with David Hendricks, Andrea Barberio	Facebook
with Philipp Deppenwiese	9elements
with Andrey Mirtchovski	Cisco

www.linuxboot.org

State of Intel x86 platforms today



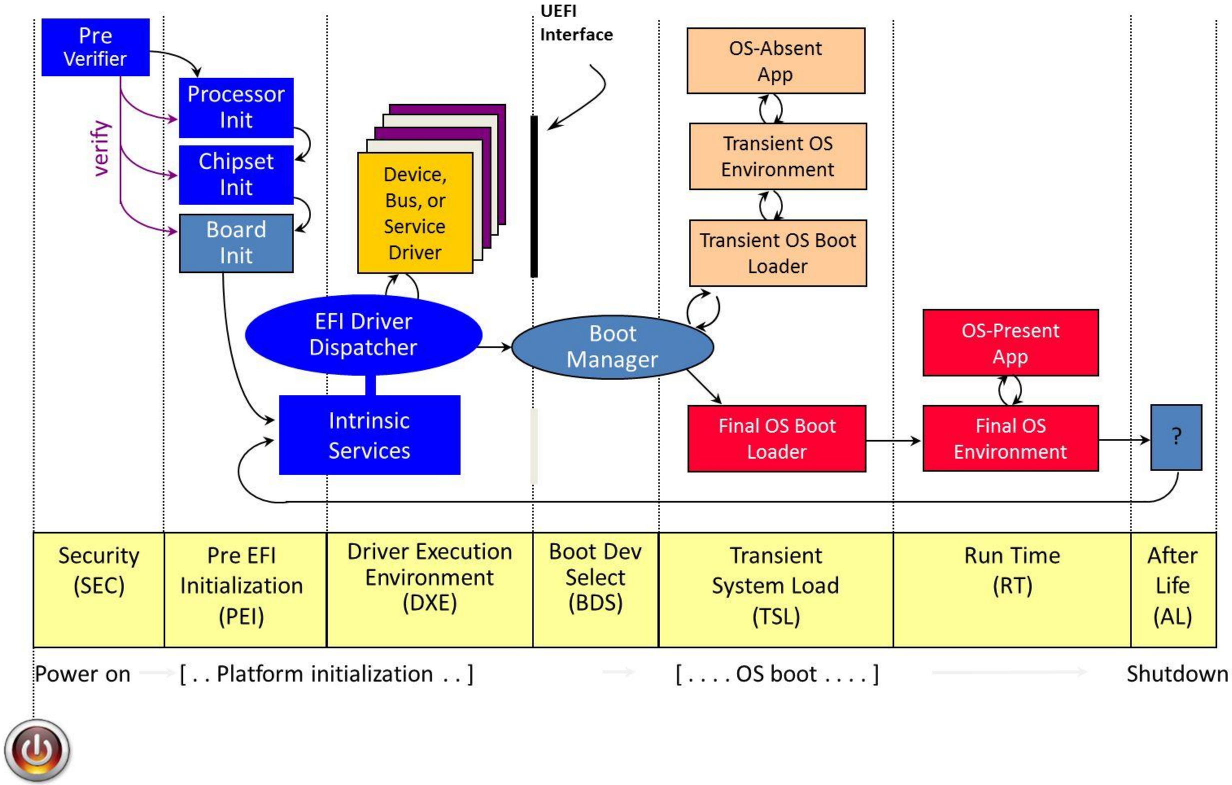
What’s in x86 firmware?

- Mostly closed source UEFI
- Completely proprietary and potentially exploit friendly
- Controlled by vendor; hard to update without vendor support
- Varies from board to board, even on two ostentatiously identical machines

UEFI Boot

OCP Winterfell node has over 120 files in the DXE Firmware Volume

Platform Initialization (PI) Boot Phases



What's in the DXE firmware volume? (and more)

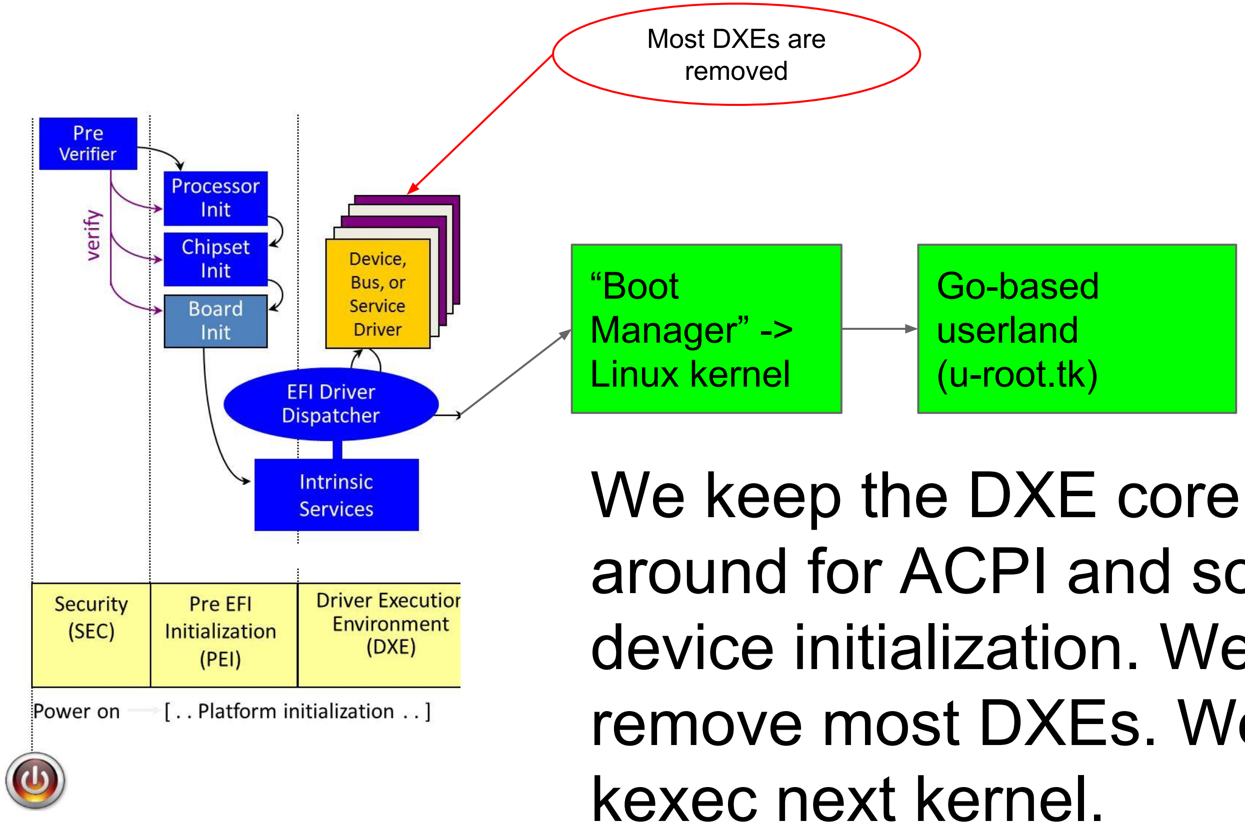
CsmVideo
Terminal
SBAHCI
AHCI
AhciSmm
BIOSBLKIO
IdeSecurity
IDESMM
CSMCORE
HeciSMM
AINT13
HECIDXE
AMITSE
DpcDxe

ArpDxe
SnpDxe
MnpDxe
UefiPxeBcDxe
NetworkStackSetupScreen
TcpDxe
Dhcp4Dxe
Ip4ConfigDxe
Ip4Dxe
Mtftp4Dxe
Udp4Dxe
Dhcp6Dxe
Ip6Dxe
Mtftp6Dxe

Udp6Dxe
IpSecDxe
UNDI
IsaBusDxe
IsaIoDxe
IsaSerialDxe
DiskIoDxe
ScsiBus
Scsidisk
GraphicsConsoleDxe
CgaClassDxe
SetupBrowser
EhciDxe
UhciDxe

UsbMassStorageDxe
UsbKbDxe
UsbMouseDxe
UsbBusDxe
XhciDxe
USB/XHCI/etc
Legacy8259
DigitalTermometerSensor (sic)

LinuxBoot/NERF



We keep the DXE core around for ACPI and some device initialization. We remove most DXEs. We kexec next kernel.

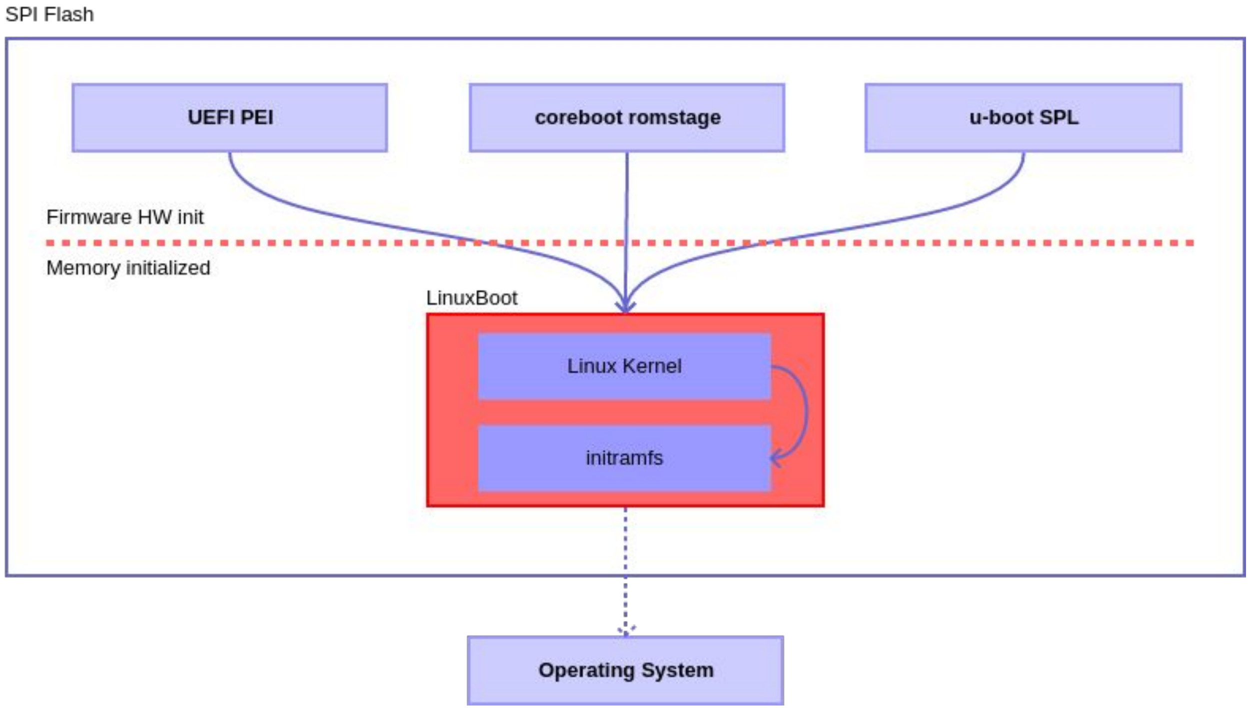
LinuxBoot DXE FV comparison

- Only 31 files
- Most of them are SMM/SMI related DXEs and ACPI
- SMM can potentially be removed one day or at least controlled by the kernel

What's the point?

- Control and update your firmware
- Reduce number of distinct drivers on the system
- Use Linux Kernel Engineers instead of having another UEFI team
- Remove unneeded legacy support
- Some apps/DXEs can be written as a user program in Linux

Forms of LinuxBoot



Common Questions

- Are we simply replacing GRUB?
  - No, we replace what is used to run GRUB
- Why have linux boot another linux?
  - Firmware flash size is small, you probably want a more capable runtime kernel
- Why have Go? What’s wrong with PXE?

Linux + what's in the initramfs?

- **Whatever you want.**
  - **We provide mechanisms, not policy.**
- Stages of firmware we are replacing...
  - Drivers
  - Bootloaders
  - Debugging shells
  - ...
- Busybox?
- systemd-boot?

u-root: userspace in Go

- We have the full toolset of Linux applications at our fingertips in firmware now.
  - Let’s use them!
  - Let’s use a memory-safe language.
  - Let’s use a language that makes concurrency easy.

- u-root: 3M (compressed) initramfs in **Go**
  - busybox-like tools (dd, ls, cpio, ...)
  - kexec-based bootloaders (PXE- and GRUB-compatible boot tools, ...)

- **LinuxBoot + u-root: NERF**
- **There are other runtimes: e.g. Heads.**

u-root: 30 Go commands in 3M? How?!

- Source Mode: 6M compressed.
  - Go toolchain (compiler, linker, assembler, etc).
  - All commands in source.
  - Compiled and cached in tmpfs on the fly.
    - ~200ms to compile basic command.
  - Architecture-independent.

- BB Mode: 3M compressed.
  - Take all source, rewrite using AST to compile all into one binary.
  - Busybox-style: argv[0] decides what to execute.
  - Initramfs contains **one** binary.

Implications

- Standard Linux shell
  - Your firmware runs a shell you are used to!
  - No custom UEFI shells with strange commands.
  - Just use the tools you already know
- **sshd**: ssh into your firmware to debug!
  - No more bricked machines: just ssh in when it fails to boot past firmware.

- **(u-root only) init**: custom-built init in Go is faster.
  - No need for systemd, upstart, scripts.
  - Go code easier to understand than a sea of scripts

Implications (2)

- **(u-root only) Source mode**: debugging commands on the fly
  - Rewrite the source, remove the cached version, run to recompile.
  - Versatility of scripts with features and type system of Go.
- **PXE boot**
  - No more 16-bit code.
  - Trivial to use modern features.
    - HTTP(S), IPv6, ...
    - Just use a kernel & language with well-tested, audited support for them!
  - Trivial to parallelize.
    - Stop waiting for NICs to time out trying PXE boot in serial...
    - Just spawn a thread to try on each NIC.

Implications (3)

- Develop firmware applications using modern toolsets
  - Use Go static analysis tools
  - Race detector, memory sanitizer, etc...
  - Continuous Integration testing
  - Open documentation

- (Bootloader) Apps run in Ring 3 - UEFI runs them in ring 0
  - Application crashes - kernel is still up
  - ssh in and debug!

Implications (4)

- Want to write your own bootloader?
  - Hire a firmware engineer…
  - Wait, no. Just hire a normal Linux application engineer.
  - Leverage Linux knowledge already out there.

- You're starting to get the gist...

Links

- LinuxBoot website: www.linuxboot.org
- LinuxBoot GitHub: github.com/linuxboot/linuxboot
- u-root GitHub: github.com/u-root/u-root
- Heads: www.osresearch.net