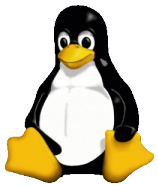
ARM



UEFI+Linux on ARM

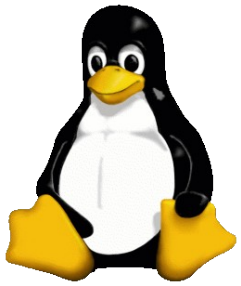
Making it Just Work

Korea Linux Forum
12 November 2013

Presentation by Grant Likely

Linaro

Agenda



ARM

- Introduction
- Rationale
  - Enterprise
  - Mobile
  - Embedded
- Boot Sequence
- Runtime Services
- Device Tree, ACPI and SMBIOS
- Current Status
- Future Work?
- Questions

Linaro

Rationale: Enterprise

- New ARM enterprise products
  - Competing with x86 platforms
- Any gratuitous difference from x86 is a cost
  - For vendors – Engineering and manufacturing tools won't carry over
  - For customers – Integrating into data centre requires new knowledge
- UEFI and U-Boot behave very differently
  - U-Boot
    - Boot variables specify kernel, initrd and command line
    - Currently no default behaviour for booting automatically
  - UEFI
    - Specification for how to choose boot device.
    - Specification for ABI and execution environment.

Rationale: Enterprise

- ARM servers should behave the same
  - Use same firmware ABI – UEFI
  - Use same hardware description ABI – ACPI
  - Use same interfaces
    - Network boot – DHCP, and TFTP of UEFI executable
    - Block device – GPT Partition table, FAT system partition
    - Secure Boot – Ship in Setup Mode, as is appropriate for server machines
    - Firmware device drivers – same ABIs
  - Same software stack
    - Second stage boot selection GRUB, Gummiboot or rEFInd
    - UEFI stub embedded in kernel
- How can Linux developers influence UEFI development?

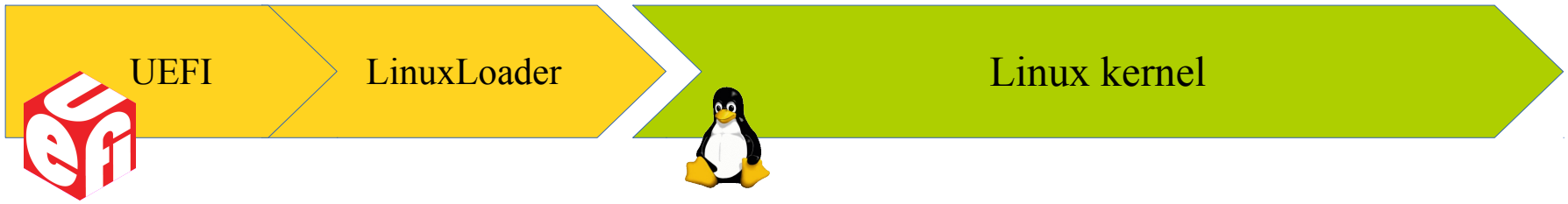
Rationale: Mobile

- Smart phones have become general-purpose computers
  - Abstracted at the userspace level.
- Much of the hardware looks the same anyway
  - Big processor, large touch screen, a bunch of sensors and wireless connectivity
  - Many SoC vendors provide U-Boot support,
  - but handset vendors ship Little Kernel.
    - Mostly a licensing issue
- Can the benefit of UEFI on server “trickle down” to the mobile platforms?
  - Familiar development environment
- UEFI adoption in mobile must be organic
  - No economic pressure to switch; Cannot be forced
  - UEFI must prove itself to be better than alternatives

Rationale: Embedded

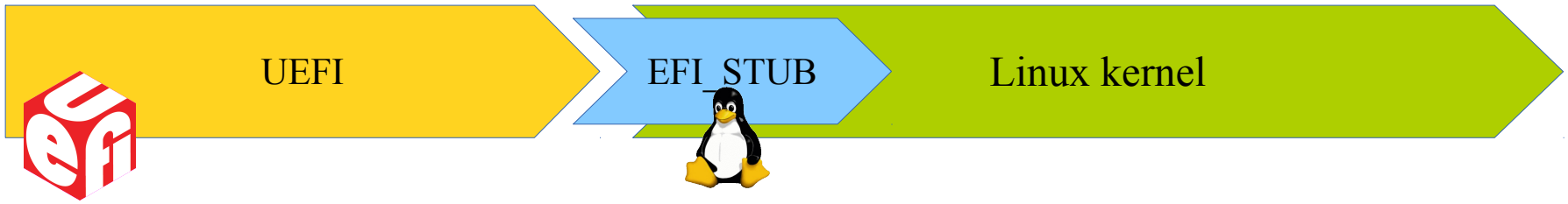
- Heavy vertical integration
  - Vendor controls entire stack
  - Smaller incentive to standardize
  - Device-specific customization and builds are common
- For UEFI to be relevant for embedded:
  - Must prove itself a better solution
  - Must make *development* easier
    - Standardization helps, but is not enough
  - Tianocore needs to match U-Boot's healthy community

Boot Sequence: Old Method – LinuxLoader



- ARM Prototype for booting Linux on UEFI
  - Built into Tianocore
  - Loads initrd and device tree images
- Problems
  - Build into UEFI, cannot be updated independently
  - No guarantee firmware will provide LinuxLoader
    - Not part of UEFI spec
  - No runtime services – Linux unaware of UEFI

Boot Sequence: UEFI Stub

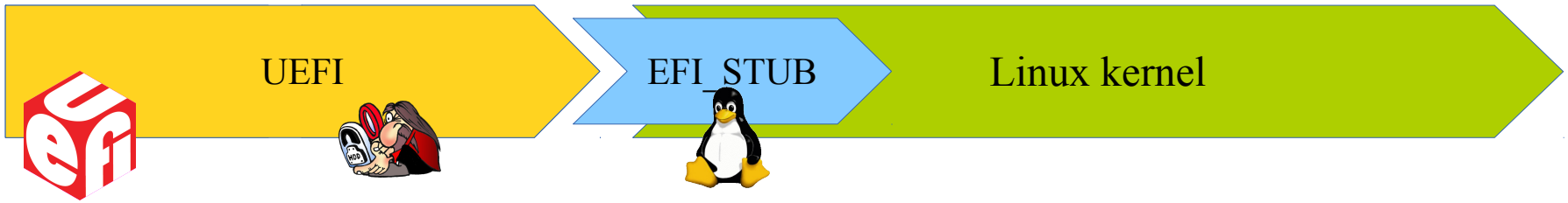


- A UEFI OS Loader embedded into the Linux kernel
  - Linux becomes native UEFI PE/COFF binary
  - Generalized from x86 Linux EFI_STUB
  - Easy to modify
  - 100% compatible with non-UEFI firmware
    - Single kernel image works with both UEFI and U-Boot

Boot Sequence: Embedding the UEFI Stub


```
 start:
                .type   start,#function
-               .rept   7
+#ifdef CONFIG_EFI_STUB
+               .word   0x62805a4d              @ Magic MSDOS signature
+                                               @ for PE/COFF + ADD opcode
+#else
+               mov     r0, r0
+#endif
+               .rept   5
                mov     r0, r0
                .endr
    ARM(                mov     r0, r0          )
    ARM(                b       1f              )
  THUMB(                adr     r12, BSYM(1f)   )
  THUMB(                bx      r12             )
+ THUMB(                .thumb                  )
+1:
+               b       zimage_continue

                .word   0x016f2818              @ Magic numbers to help the loader
                .word   start                   @ absolute load/run zImage address
                .word   _edata                  @ zImage end address
+
+       .org    0x3c
+       .long   pe_header                       @ Offset to PE-COFF header
```

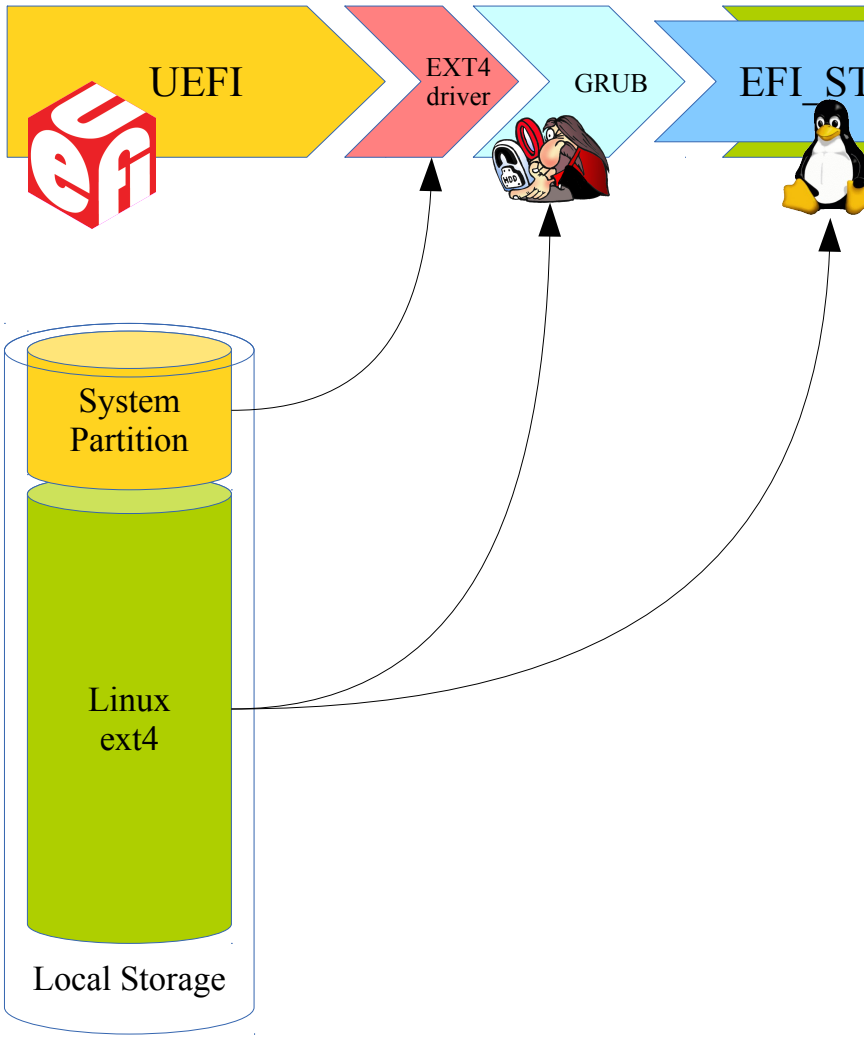
Boot Sequence: GRUB



- Combines boot menu, OS loader and filesystem drivers
- Linux distributions already use GRUB
  - Familiar to users
- Also boots non-Linux
  - Xen
  - *BSD
- Complex boot scenarios
  - ie. kernel and initrd from different sources
- Substitute Gummiboot or rEFInd here

rEFInd
Boot Management

Boot Sequence: Filesystem Drivers

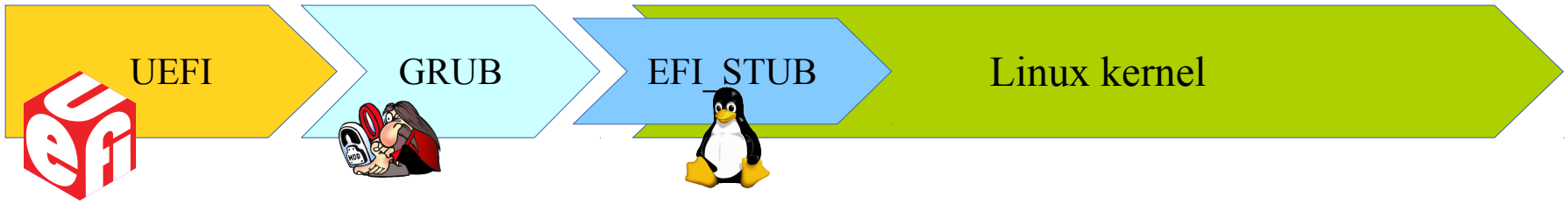


- Native filesystem drivers loaded at boot time
- rEFInd maintains drivers
  - EXT4
  - BTRFS
  - ISO-9660
- Works on ARM
  - nobody currently providing binaries

Boot Sequence: Kernel Entry

- UEFI Stub will:
  - Attempt to find an FDT pointer in the system table
  - Generate an empty FDT if necessary
  - Store kernel boot arguments in FDT
    - System table pointer
    - UEFI memory map
    - Initrd pointer
    - Kernel command line
  - Call ExitBootServices()
  - Jump to kernel entry point
- In turn, stub calls normal kernel entry point
  - 32-bit: r2 = Flattened Device Tree (FDT) pointer
  - 64-bit: x0 = Flattened Device Tree (FDT) pointer
- What about ACPI?
  - FDT is still passed to the kernel, but limited to kernel arguments
  - ACPI pointer stored in UEFI configuration table.

Runtime Services



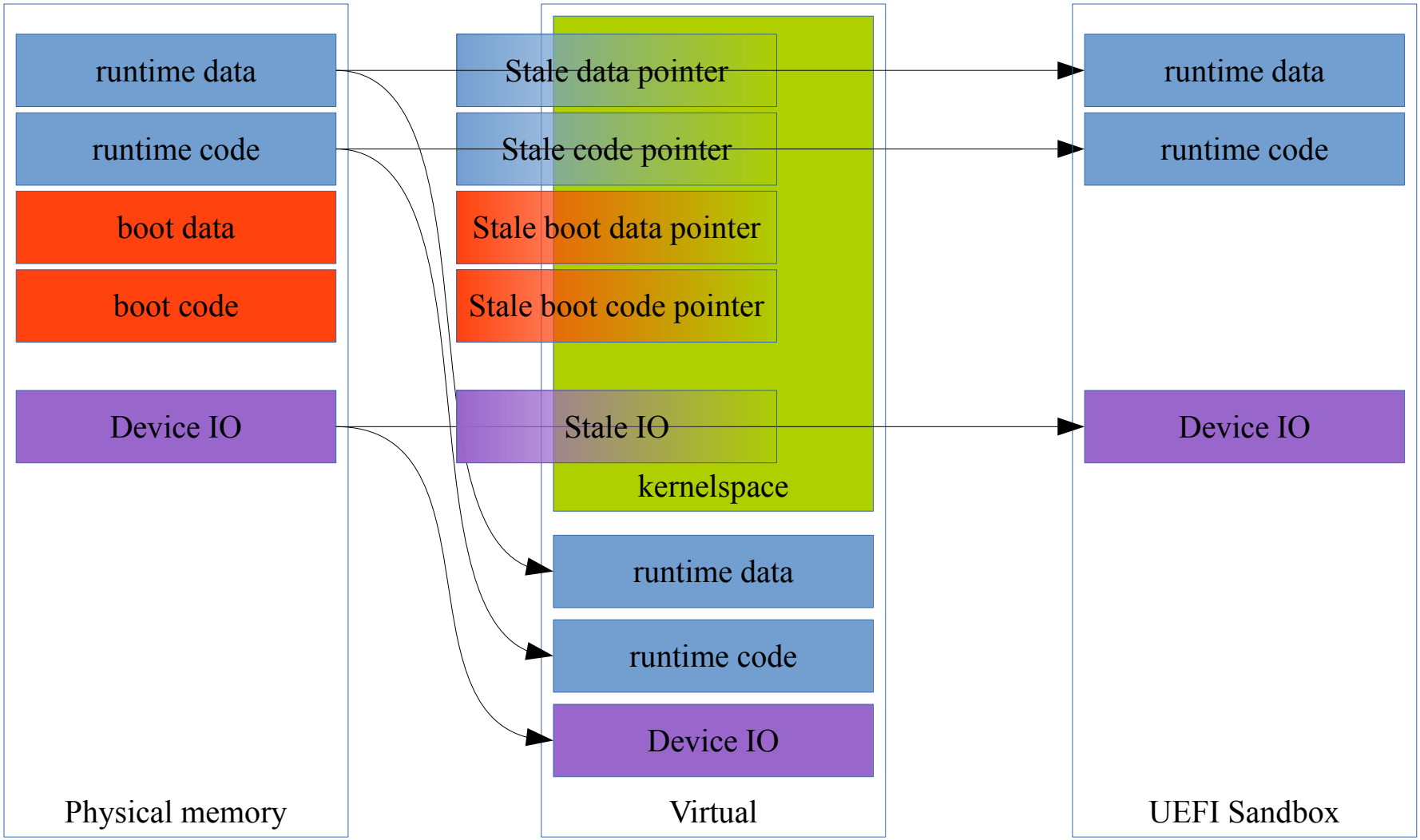
- Limited OS interaction with UEFI
  - Query/Set time of day & Wakeup time
  - Query/Set UEFI variables
  - Capsule service
- Required to configure boot
  - grub_install & efibootmgr
- Linux must:
  - Obtain UEFI system table
  - Retrieve UEFI memory map
  - Use UEFI runtime calling convention
  - Call SetVirtualAddressMap() to provide UEFI with memory map

```
typedef struct {
  EFI_TABLE_HEADER Hdr;
  CHAR16 *FirmwareVendor;
  UINT32 FirmwareRevision;

  EFI_HANDLE ConsoleInHandle;
  EFI_SIMPLE_TEXT_INPUT_PROTOCOL *ConIn;
  EFI_HANDLE ConsoleOutHandle;
  EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL *ConOut;
  EFI_HANDLE StandardErrorHandle;
  EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL *StdErr;

  EFI_RUNTIME_SERVICES *RuntimeServices;
  EFI_BOOT_SERVICES *BootServices;

  UINTN NumberOfTableEntries;
  EFI_CONFIGURATION_TABLE *ConfigurationTable;
} EFI_SYSTEM_TABLE;
```

Runtime Services: Virtual Address Map Pain

- SetVirtualAddressMap() requires **all** UEFI code to correctly update internal pointer references
  - Easy to make mistakes
  - Stray pointer references can corrupt Linux data structures if not protected
  - Sleepy vendors won't patch firmware bugs
- Doesn't play well with kexec
  - Can only be called once
  - New kernel forced to use same map as old one
- What do we do?
  - Use SetVirtualAddressMap() anyway and live with bugs?
  - Use separate page tables when executing runtime services?
  - Don't support runtime services in Linux?

Runtime Services: Virtual Address Map Pain



Platform Configuration

- Loaded by UEFI
- Provided via System Table
  - SMBIOS
  - ACPI
  - Flattened Device Tree

```
typedef struct {
  EFI_TABLE_HEADER Hdr;
  CHAR16 *FirmwareVendor;
  UINT32 FirmwareRevision;

  EFI_HANDLE ConsoleInHandle;
  EFI_SIMPLE_TEXT_INPUT_PROTOCOL *ConIn;
  EFI_HANDLE ConsoleOutHandle;
  EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL *ConOut;
  EFI_HANDLE StandardErrorHandle;
  EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL *StdErr;

  EFI_RUNTIME_SERVICES *RuntimeServices;
  EFI_BOOT_SERVICES *BootServices;

  UINTN NumberOfTableEntries;
  EFI_CONFIGURATION_TABLE *ConfigurationTable;
} EFI_SYSTEM_TABLE;


typedef struct{
  EFI_GUID VendorGuid;
  VOID     *VendorTable;
} EFI_CONFIGURATION_TABLE;
```

ACPI or Flattened Device Tree

- Doesn't UEFI mean ACPI?
  - No!
  - FDT works today with UEFI
- Kernel supports both ACPI and FDT
  - FDT is not going away any time soon
- Platform can supply either ACPI or FDT
  - Not both!
  - If platform provides both then the FDT shall be ignored

A note on development philosophy

- Some things you may have noticed:
  - UEFI Stub doesn't break U-Boot
  - UEFI works with FDT and ACPI
  - ACPI doesn't disable FDT
  - X86 and ARM share UEFI Stub implementation
  - As much as possible, ARM UEFI behaves the same as x86 UEFI
- Breaking users is unfriendly
- Clear migration paths are important
  - By design, all the new development has been done in a way that doesn't required old features to be turned off
  - Want to make it as easy as possible for developers to get started with UEFI

Other Topics: Virtualization

- UEFI must call OS Loader (stub) in HYP/EL3 mode
  - 64-bit (v8): EL2
    - Implemented and working
  - 32-bit (v7a): Hypervisor mode
    - Proposal drafted and debated
    - Requires new protocol to elevate permission
    - Not implemented due to lack of interest
- KVM works on both

Other Topics: Security

- Secure Boot
  - All functionality exists in Tianocore
  - Strongly recommend shipping hardware in Setup Mode
  - Hardware requires secure storage
- TrustZone
  - ARM recommends Generic Secure Firmware
  - Power State Coordination Interface (PSCI)

Current State

	arm	arm64
GRUB-EFI	In mainline	In development
EFI_STUB	Working, Merged soon	Working, under review
Runtime services	Working, Merged soon	Working, under review
Filesystem drivers	Tested	
ACPI	Prototype, patches avaliable	Prototype, patches avaliable
Secure Boot	Investigating	Investigating
Hypervisor support	Prototype patches available no further work planned	In mainline

Future Work

- Upstreaming completed work
- Board ports
- Features desired/expected by embedded
- Kexec support
- Working with upstream Tianocore

Resources

- https://wiki.linaro.org/LEG/Engineering/Kernel/UEFI
- http://tianocore.sourceforge.net
- http://www.uefi.org
- https://wiki.linaro.org/LEG/Engineering/Kernel/ACPI
- http://sourceforge.net/apps/mediawiki/tianocore/index.php?title=ArmPlatformPkg

Acknowledgements

- Thank you to the following organizations and people
  - Linaro Enterprise Group and Linaro member companies sponsoring this work
    - UEFI Team: Leif Lindholm, Roy Franz, Mark Salter, Yi Li, Reece Pollack, Rony Nandy, Steven Kinney
    - ACPI Team: Al Stone, Graeme Gregory, Hanjun Guo, Naresh Bhat, Tomasz Nowicki
    - Ryan Harkin, Olivier Martin
  - UEFI Forum which has been a great organization to work with
  - Linux Foundation for providing me the opportunity to speak
  - Many Others

Questions?

Additional Material

Linaro Overview

- Linaro is a not for profit software engineering company
- Members are ARM SoC vendors and other companies interested in the ARM ecosystem
- Rather than each company duplicating open source effort for common software, the cost is shared between competitors, and the software is built once
- The work is carried out in the open, and the results are extensively tested and then upstreamed into the relevant open source projects – e.g. kernel.org

Linaro

Linaro Enterprise Group

- Formed in November 2012
- Working on core open-source software for ARM servers
  - Boot architecture – UEFI/ACPI
  - Virtualization – KVM/Xen
  - ARMv8 bringup & optimization
  - LAMP, OpenJDK, Hadoop, OpenStack
- Reduces costs, eliminates fragmentation, accelerates time to market
- Members can focus on innovation and differentiated value-add

Group Members

http://www.linaro.org/engineering/leg