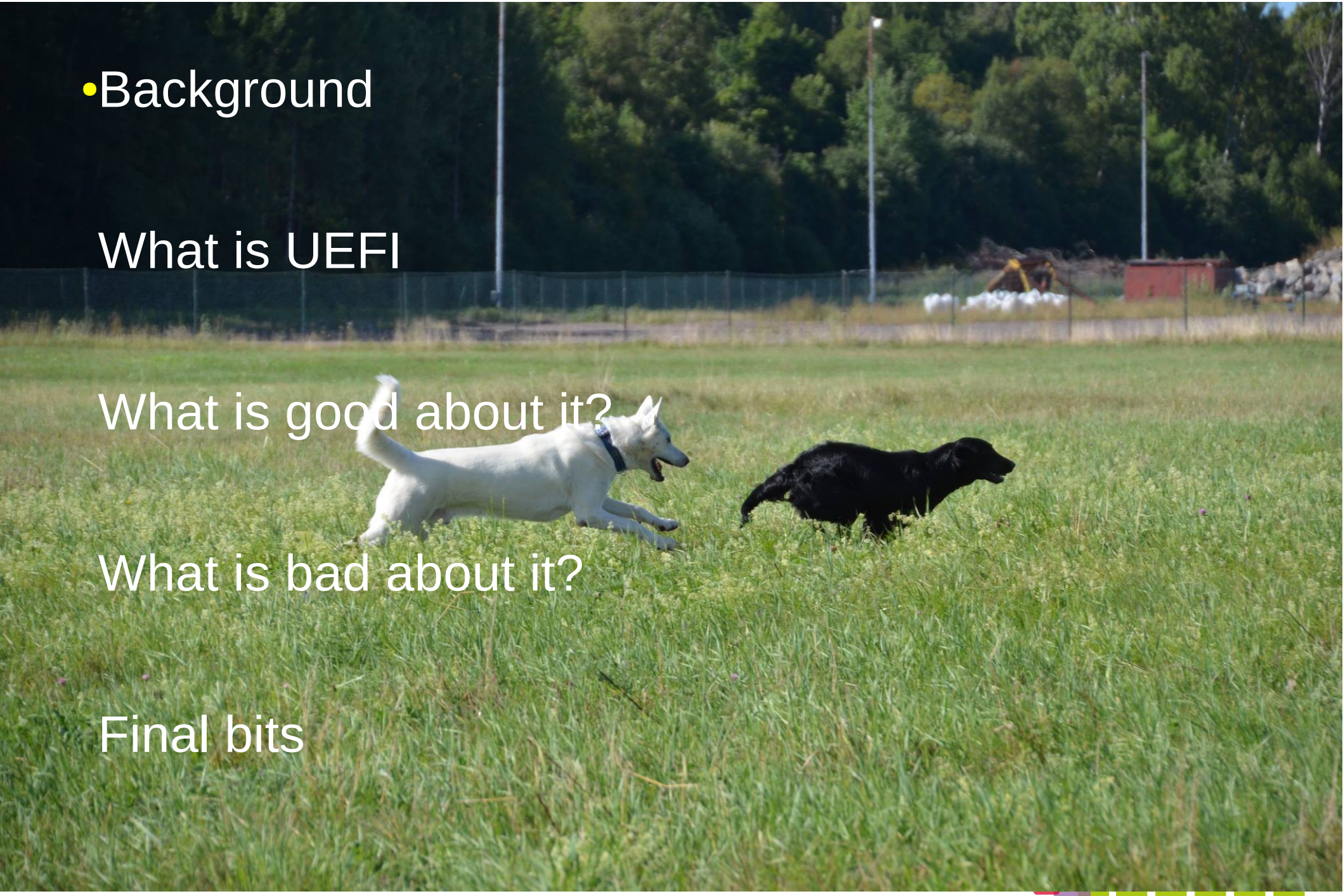
UEFI is not your enemy

Leif Lindholm

LinuxTag 2014

Linaro®



- Background

What is UEFI

What is good about it?

What is bad about it?

Final bits

Overview

UEFI has managed to acquire a bit of a bad reputation in the open source/free software community. This presentation aims to:

- set the record straight about what problems exist out there in the UEFI ecosystem
- explain how they relate to the basic UEFI standard and to the codebase
- (and hopefully dispell some misunderstandings)

I have started hearing things which would in the past just have been shrugged off as BIOS bugs now referred to as "UEFI secure boot bollox on a slippery slope to ensure you will not be able to run Linux on your hardware".

In order for the rest of the presentation to be of any value, I need to start by attempting to explain how UEFI is not the evil plot by the industry to enforce device lockdown.

And even if you have already made your mind up that the very concept of binary signing is evil because it is possible to use it for device lockdown, this presentation will hopefully help clarify which bits you should focus on opposing.

First, in order of decreasing level of nefariousness, I will go through four levels of distinct concepts which all to some extent form part of what people tend to bundle together under the banner "UEFI Secure Boot" (or sloppily, just "secure boot", or even just “UEFI”):

- Microsoft logo requirements for Windows 8
- UEFI Secure Boot
- UEFI
- Shim

UEFI != ACPI

Another common theme in certain discussions (certainly in the ARM camp) is bundling UEFI and ACPI together.

UEFI provides mechanisms (defined by the standard) for the firmware to present ACPI data to the operating system.

The same methods can be used to present device-tree data.

Microsoft logo requirements

As part of the launch of Windows 8, Microsoft wrote down some rules about how the firmware on _any_ devices that ship with Windows preinstalled must operate. Part of this was that it must be able to cryptographically verify the signature of any images it loads, using the UEFI Secure Boot protocol.

“For logo-certified Windows RT 8.1 and Windows RT PCs, Secure Boot is required to be configured so that it cannot be disabled.”

For x86 devices, this document explicitly states that this signature checking mechanism must be possible to disable. For Windows RT (currently all ARM) devices, this document explicitly states that this signature checking mechanism must NOT be possible to disable.

This annoys me to no end, but from Microsoft's point of view, it's Windows vs. Windows RT - and RT devices will only ever run the OS shipped with it (yeah, right!).

Logo requirements #2

The entirely obvious problems with this are exacerbated by the facts that:

- the implementation details (and shortcomings of the UEFI specification prior to 2.4) mandate only one key can be registered, meaning all OS installers must use the same key.

- Microsoft, for all intents and purposes, are their own CA for Windows installers. Hence anyone who wants to install software on anything shipped with Secure Boot enabled must have their installer signed by Microsoft.

UEFI Secure Boot

UEFI Secure Boot is a standard for supporting, and enforcing, the cryptographic verification of loaded images before they can be executed. That is all it is.

It depends entirely on hardware, software that comes before it, and software that comes after it to actually achieve something that with a straight face could be called secure boot. So just as if you have a color blind (or obnoxious) friend with a cute red pet lizard called Green, just remember that in the context of the UEFI environment, Secure Boot is a name, not a description.

I am not actually going to dwell much on the use of UEFI Secure Boot in the rest of the presentation.

UEFI

UEFI does not mandate the use of Secure Boot. Nor does it mandate that when Secure Boot exists, there should be anything restricting the device owner from disabling it.

Hating UEFI because it is possible to mandate that it must only run signed images makes about as much sense as hating Linux because you can run software that uses DRM (Spotify, flash) on it.

“The threat is not the UEFI specification itself, but in how computer manufacturers choose to implement the boot restrictions.“ - FSF

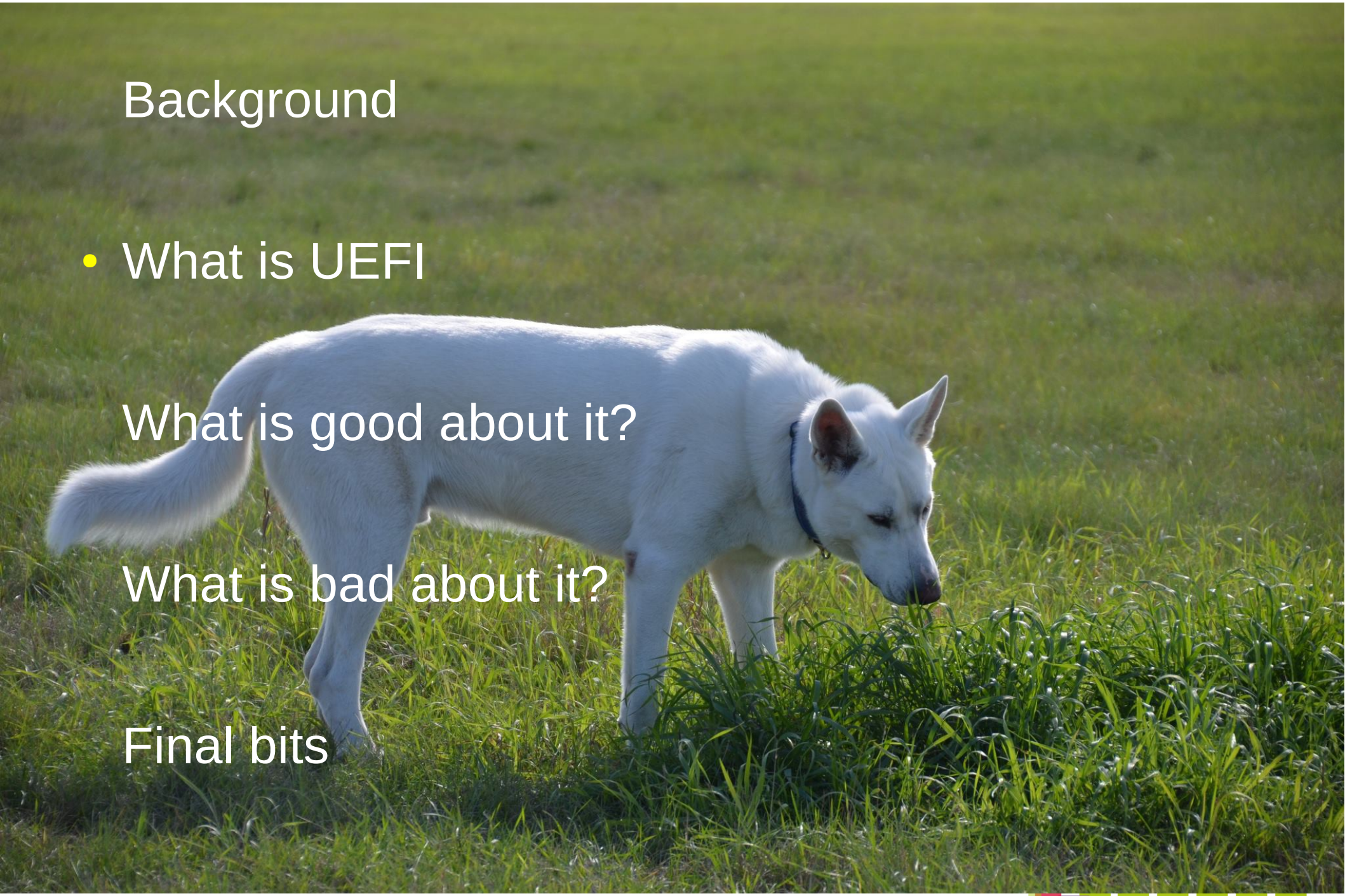
Shim

Shim is a UEFI application that also installs a UEFI protocol for use by other applications. It was written by Matthew Garret, in order to make it possible to load binaries _not_ signed by the primary firmware key.

A utility that just sidestepped the chain-of-trust checking would be unlikely to be signed by any serious CA, so what Shim does is simply providing a second level of authentication; a key database that can be kept in addition to the primary firmware key, and let the operator securely add/remove keys (given proper hardware implementation).

Used by commercial distro vendors in conjunction with GRUB and an out-of-tree patch providing support for using the shim protocol for loading kernel/initrd on x86.

The FSF recognised Matthew's efforts in this area by giving him their Award for Advancement of Free Software in 2014.



Background

- What is UEFI

What is good about it?

What is bad about it?

Final bits

What is UEFI?

UEFI is something almost unique in the history of mankind; it is a specification for a firmware architecture, which has gained critical mass in the (commercial) community and is already the defacto standard for x86 machines.

Moreover, UEFI is only the specification - not the implementation.

The origin is EFI, the Extensible Firmware Interface developed by Intel/HP for the IA64 architecture. Version 1.10 was handed over to the UEFI Forum as the starting point of the UEFI specification.

Why was it needed?

It replaces BIOS: a horrible, outdated piece of crud, tied to an architecture that has not really existed for decades. A "secret sauce" piece of software reverse-engineered out of the original IBM PC and then bolted onto for as long as was possible, before it simply could no longer be extended to support more RAM, larger hard drives or fundamental changes to system boot architecture. An entirely closed world run by a very small group of companies all busy duplicating each other's efforts.

UEFI is all nice and shiny!

Well, no.

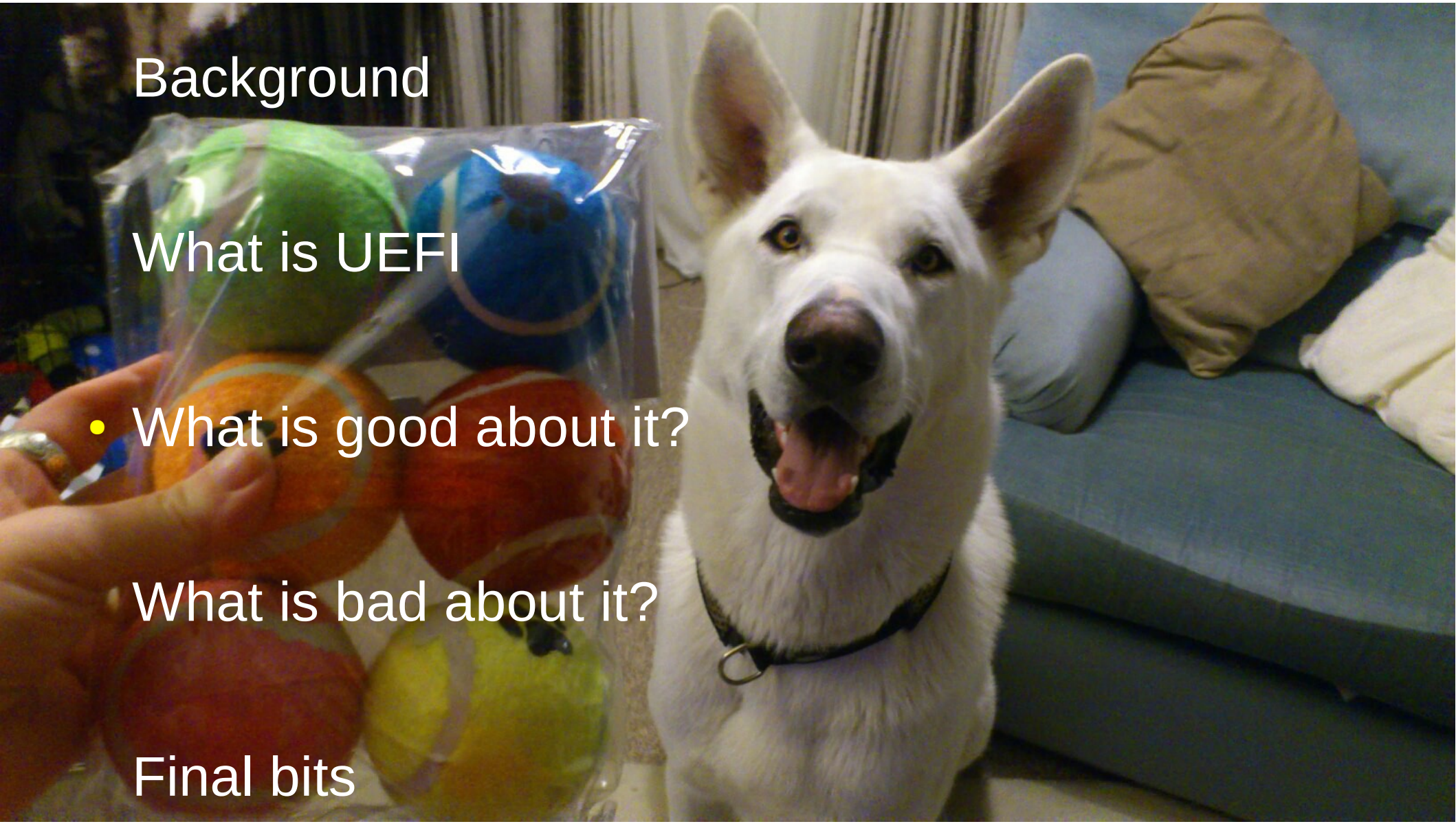
But at least its overall architecture is 20+ years more modern than BIOS. It was developed clean for IA64, and it is actually fairly well designed.

But it is still a firmware infrastructure, and hence horrible.

TianoCore + edk2

On releasing the specification, Intel also released the overall framework (but not the platform support code) into an open source project called TianoCore. TianoCore does its overall development in the edk2 (EFI Development Kit v2) tree.

It is an active project, contributed to by both hardware, BIOS and software vendors. But the “UEFI BIOS” in modern PCs is augmented with additional bits and bobs provided by the BIOS vendors.



- Background
- What is UEFI
- What is good about it?
- What is bad about it?
- Final bits

Linaro®

In general

It provides a standardised execution environment, into which any boot loader, device driver or boot time configuration utility can be installed.

This execution environment provides things like direct block access support, direct Ethernet support, console support - all portable across *any* implementation (in theory – and usually in practise).

Filesystem support

- It has explicit support for GUID Partition Table (death to MBR!).
- While it mandates VFAT, Microsoft have released their VFAT driver with explicit patent grants and stuff for uses in UEFI firmware.
- Support for additional filesystems can be added by loading drivers. The rEFInd boot program, for example, comes with a GPLd ext2 driver.

Extensible (the 'E')

Supports running applications and loading drivers and protocols (think “shared libraries”).

Expansion cards can have drivers installed into the EFI system partition, or in an option ROM, and loaded automatically on boot.

Versioned APIs.

Even supports architecture independent applications/drivers via EBC (EFI Byte Code).

Runtime Services

While somewhat horriffic from a system design point of view (my description is that it is somewhat like bits of UEFI hanging around post boot to act as a shared library for the kernel), it provides an unparallelled level of integration between operating system and firmware.

Lets the operating system set environment variables, including boot images and priority order of those – in Linux using *efibootmgr*, which simply operates on /sys files.

“Capsules” provides a standardised interface from within the operating system to do things like scheduling a firmware update on next reboot.

Standardised interface for system reboot/poweroff and RTC access.

Secure Boot

No, seriously.

Where the device owner is in control of this mechanism (and the adjacent hardware and software are doing the right thing), this can be a quite useful security feature.

Just ship the system in “setup mode” and nobody gets hurt.

It has a written specification

This may not seem to be very much, but it is huge.

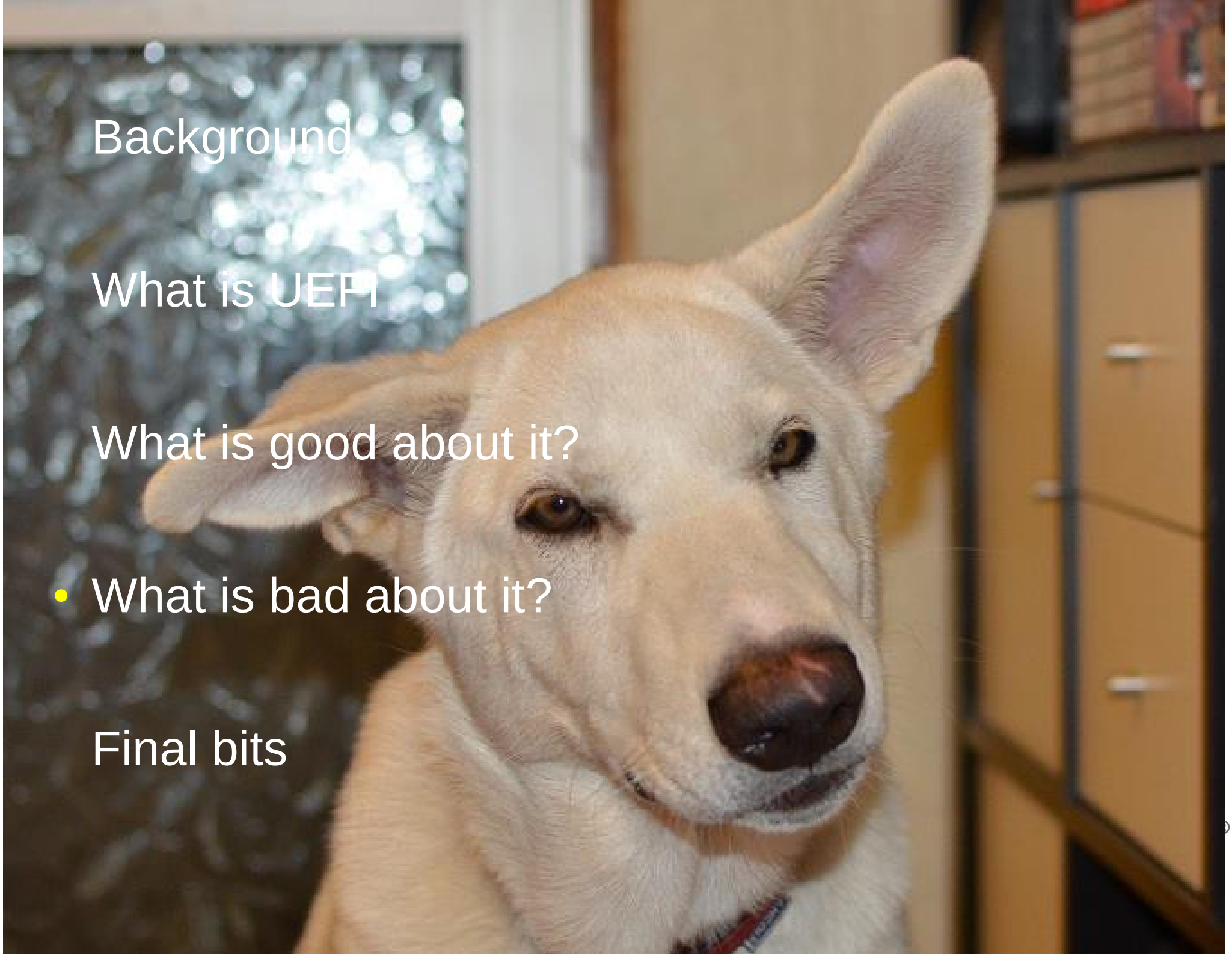
It has a conformance test suite

This may not seem to be very much, but it is huge.

Critical mass

Known to work over at least 4 different architectures.

Handy if you want to slot seamlessly into the existing <*whatever is cool this week*>scale server exosystem.



- Background
- What is UEFI
- What is good about it?
- What is bad about it?
- Final bits

Specification

Behind a registration wall.

That registration requires acknowledging that implementation is restricted to UEFI forum members.

- ...But membership at “adopter” level is costless, and available either as organisation or individual.

Source code

Well, UEFI does not exactly come entirely without legacy:

- drivers and executables are PE/COFF format (!)
- APISpecificationSyntaxIsUngodlyNeverendingCamelCase().
  - And coding style is Windows^M
- Repository is svn, but there are official git mirrors.
  - Of course, mixing git and svn has its own problems.
- UCS-2, not UTF.
- Test suite has historically only been available to licensees (and they have only received it as a .zip file drop). But it has now moved to a (restricted-access) github repository.

```
//
// Define the maximum extended data size that is supported when a status code is reported.
//
#define MAX_EXTENDED_DATA_SIZE  0x200

EFI_STATUS_CODE_PROTOCOL  *mReportStatusCodeLibStatusCodeProtocol = NULL;
EFI_EVENT                 mReportStatusCodeLibVirtualAddressChangeEvent;
EFI_EVENT                 mReportStatusCodeLibExitBootServicesEvent;
BOOLEAN                   mHaveExitedBootServices = FALSE;

/**
  Locate the report status code service.

  Retrieve ReportStatusCode() API of Report Status Code Protocol.

**/
VOID
InternalGetReportStatusCode (
  VOID
  )
{
  EFI_STATUS  Status;

  if (mReportStatusCodeLibStatusCodeProtocol != NULL) {
    return;
  }

  if (mHaveExitedBootServices) {
    return;
  //
  // Check gBS just in case ReportStatusCode is called before gBS is initialized.
  //
  if (gBS != NULL && gBS->LocateProtocol != NULL) {
    Status = gBS->LocateProtocol (&gEfiStatusCodeRuntimeProtocolGuid, NULL, (VOID**) &mReportStatusCodeLibStatusCodeProtocol);
    if (EFI_ERROR (Status)) {
      mReportStatusCodeLibStatusCodeProtocol = NULL;
    }
  }
}
```

Tianocore edk2

- Contains no* platform support
  - BSD licensed, and partly due to this, partly due to historic ecosystem, very low availability of device drivers.

CSM

Compatibility Support Module. “BIOS emulation” - backwards compatibility mode available in early PC ports.

Apart from carrying old crud over, having this support meant manufacturers didn't bother testing actually booting with UEFI properly, and there were many horrible bugs.

It is finally dying.

Background

What is UEFI

What is good about it?

What is bad about it?

- Final bits

Linaro®

Linaro

Linaro maintains a tree of edk2 with added support for a few platforms.

- https://git.linaro.org/uefi/linaro-edk2.git

Member landing teams keep “their” platforms from bitrotting.

We also do various bits of peripheral (ARM) development – GRUB, Linux UEFI runtime services support, kernel UEFI stub support, ACPI support.

ACPI

As of November 2013, the UEFI Forum is now the owner of the ACPI specification.

The previous world was that for each Windows release, the ACPI group would get together and discuss, release a new spec, and then go into hiatus until the next time.

And, no, UEFI does not mandate the use of ACPI.

Resources

- Build/run UEFI for AArch64
  - http://sourceforge.net/apps/mediawiki/tianocore/index.php?title=ArmPlatformPkg/AArch64
  - (search for 'sourceforge aarch64 uefi')
- FSF
  - http://www.fsf.org/campaigns/campaigns/secure-boot-vs-restricted-boot
- UEFI Forum - http://www.uefi.org/
  - Whitepaper: UEFI Secure Boot in Modern Computer Security Solutions
- Matthew Garrett's blog
  - http://mjg59.dreamwidth.org/

Linaro®