presented by

Driver Development with EDKII

UEFI 2020 Virtual Plugfest

September 15, 2020

Presented by Tomas Pilar, NUVIA Inc.

Meet the Presenter

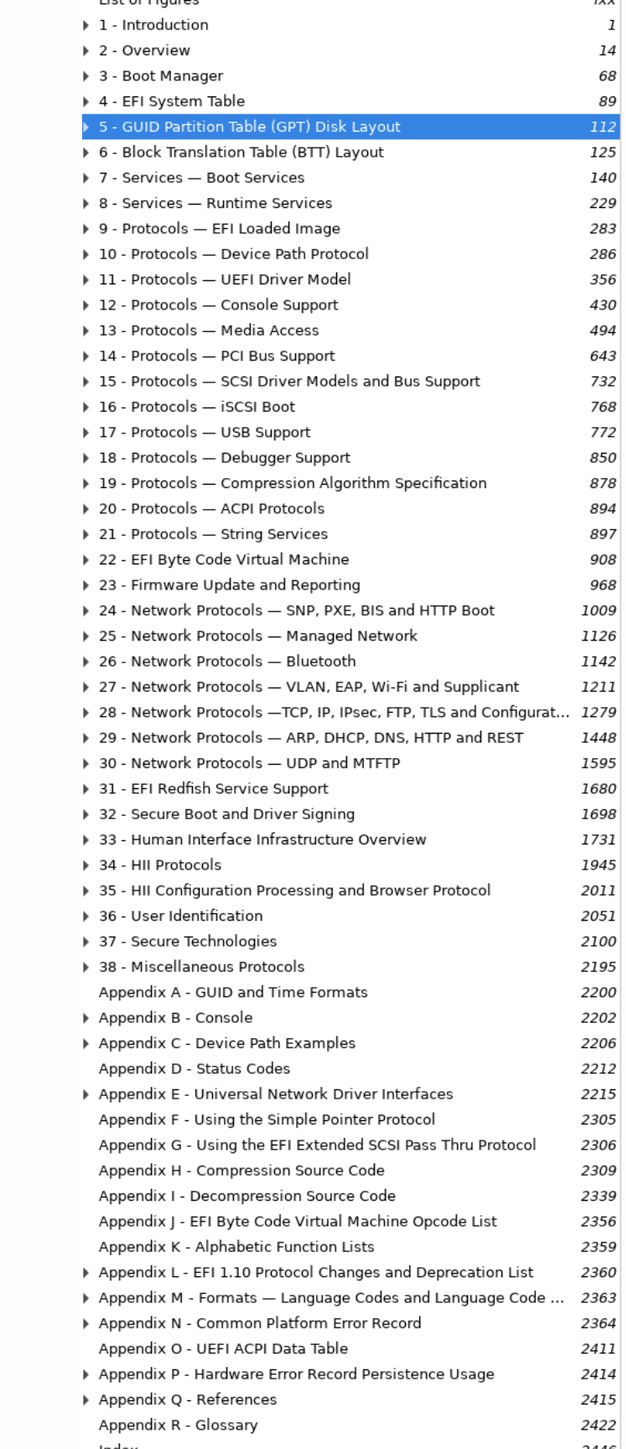
Tomas Pilar
Firmware Engineer
Member Company: NUVIA Inc.

Agenda

- Starting Resources
- UEFI Fundamentals
- Common Issues
- Force Multipliers
- Interesting Bugs

So you want to write a driver ...

Starting Resources



List of Figures lxx
1 - Introduction 1
2 - Overview 14
3 - Boot Manager 68
4 - EFI System Table 89
5 - GUID Partition Table (GPT) Disk Layout 112
6 - Block Translation Table (BTT) Layout 125
7 - Services — Boot Services 140
8 - Services — Runtime Services 229
9 - Protocols — EFI Loaded Image 283
10 - Protocols — Device Path Protocol 286
11 - Protocols — UEFI Driver Model 356
12 - Protocols — Console Support 430
13 - Protocols — Media Access 494
14 - Protocols — PCI Bus Support 643
15 - Protocols — SCSI Driver Models and Bus Support 732
16 - Protocols — iSCSI Boot 768
17 - Protocols — USB Support 772
18 - Protocols — Debugger Support 850
19 - Protocols — Compression Algorithm Specification 878
20 - Protocols — ACPI Protocols 894
21 - Protocols — String Services 897
22 - EFI Byte Code Virtual Machine 908
23 - Firmware Update and Reporting 968
24 - Network Protocols — SNP, PXE, BIS and HTTP Boot 1009
25 - Network Protocols — Managed Network 1126
26 - Network Protocols — Bluetooth 1142
27 - Network Protocols — VLAN, EAP, Wi-Fi and Supplicant 1211
28 - Network Protocols —TCP, IP, IPsec, FTP, TLS and Configurat... 1279
29 - Network Protocols — ARP, DHCP, DNS, HTTP and REST 1448
30 - Network Protocols — UDP and MTFTP 1595
31 - EFI Redfish Service Support 1680
32 - Secure Boot and Driver Signing 1698
33 - Human Interface Infrastructure Overview 1731
34 - HII Protocols 1945
35 - HII Configuration Processing and Browser Protocol 2011
36 - User Identification 2051
37 - Secure Technologies 2100
38 - Miscellaneous Protocols 2195
Appendix A - GUID and Time Formats 2200
Appendix B - Console 2202
Appendix C - Device Path Examples 2206
Appendix D - Status Codes 2212
Appendix E - Universal Network Driver Interfaces 2215
Appendix F - Using the Simple Pointer Protocol 2305
Appendix G - Using the EFI Extended SCSI Pass Thru Protocol 2306
Appendix H - Compression Source Code 2309
Appendix I - Decompression Source Code 2339
Appendix J - EFI Byte Code Virtual Machine Opcode List 2356
Appendix K - Alphabetic Function Lists 2359
Appendix L - EFI 1.10 Protocol Changes and Deprecation List 2360
Appendix M - Formats — Language Codes and Language Code ... 2363
Appendix N - Common Platform Error Record 2364
Appendix O - UEFI ACPI Data Table 2411
Appendix P - Hardware Error Record Persistence Usage 2414
Appendix Q - References 2415
Appendix R - Glossary 2422

Starting Resources I

The Spec

https://uefi.org/uefi

https://uefi.org/specifications

- “Complete” - 38 chapters, 12 appendices, 2500 pages
- Not actually complete - not all APIs, no libraries
- Read chapters 1-11 (except 5 & 6)
- “Code First” approach adopted only very recently
- Excellent reference (corner cases exist however)

Starting Resources II

Written by Intel (2012)

https://edk2-docs.gitbook.io/edk-ii-uefi-driver-writer-s-guide/

Revision History

Revision	Description	Date
0.31	Initial draft.	4/3/03
0.70	Initial draft. Edited for formatting and grammar.	6/3/03
0.90	Incorporated industry review comments.	7/20/04
	Updated the coding conventions.	
	Updated for the 1.10.14.62 release of the EFI Sample Implementation.	
	Updated the supported versions of Microsoft Visual Studio *and Windows*.	
	Removed TBD sections that appeared in the 0.7 version. Edited for grammar and formatting.	
0.91	Updated for UEFI 2.0	10/31/06
0.92	New formatting	11/27/06
0.93	Review feedback incorporated	1/14/2007
0.94	Additional formatting	2/27/2007
0.95	Additional formatting	3/23/2007
0.96	Additional formatting	4/25/2008
0.97	Clarify role of EDK as being implementation-specific and added definitions myriad of library references so the meaning of the implementation specific code examples could be clarified without having to reference documents aside from the UEFI Specification.	6/25/2008
0.98	Updated for UEFI 2.3.1 and EDK II	2/12/12
1.00	Review feedback incorporated, additional formatting	2/27/12
1.01	Review feedback incorporated	3/8/12
1.1	Conversion to GitBook markdown format	4/18/18

Driver Writer's Guide

- Good introductory material
- Despite age, content still relevant
- BOLO:
  - Obsolete protocols
  - Industry not following described practices
  - EFI ByteCode (EBC)

Starting Resources III

devel@edk2.groups.io

- Developments (patches)
- General help

discuss@edk2.groups.io

- More abstract questions

bugs@edk2.groups.io

- Bugzilla feed

Group Information

http://tianocore.org
556 Members
28,304 Topics, Last Post: 10:38am
Started on 2016-07-29
Feed

Mailing List

- Where development and maintenance happens
- Plenty of people happy to answer questions

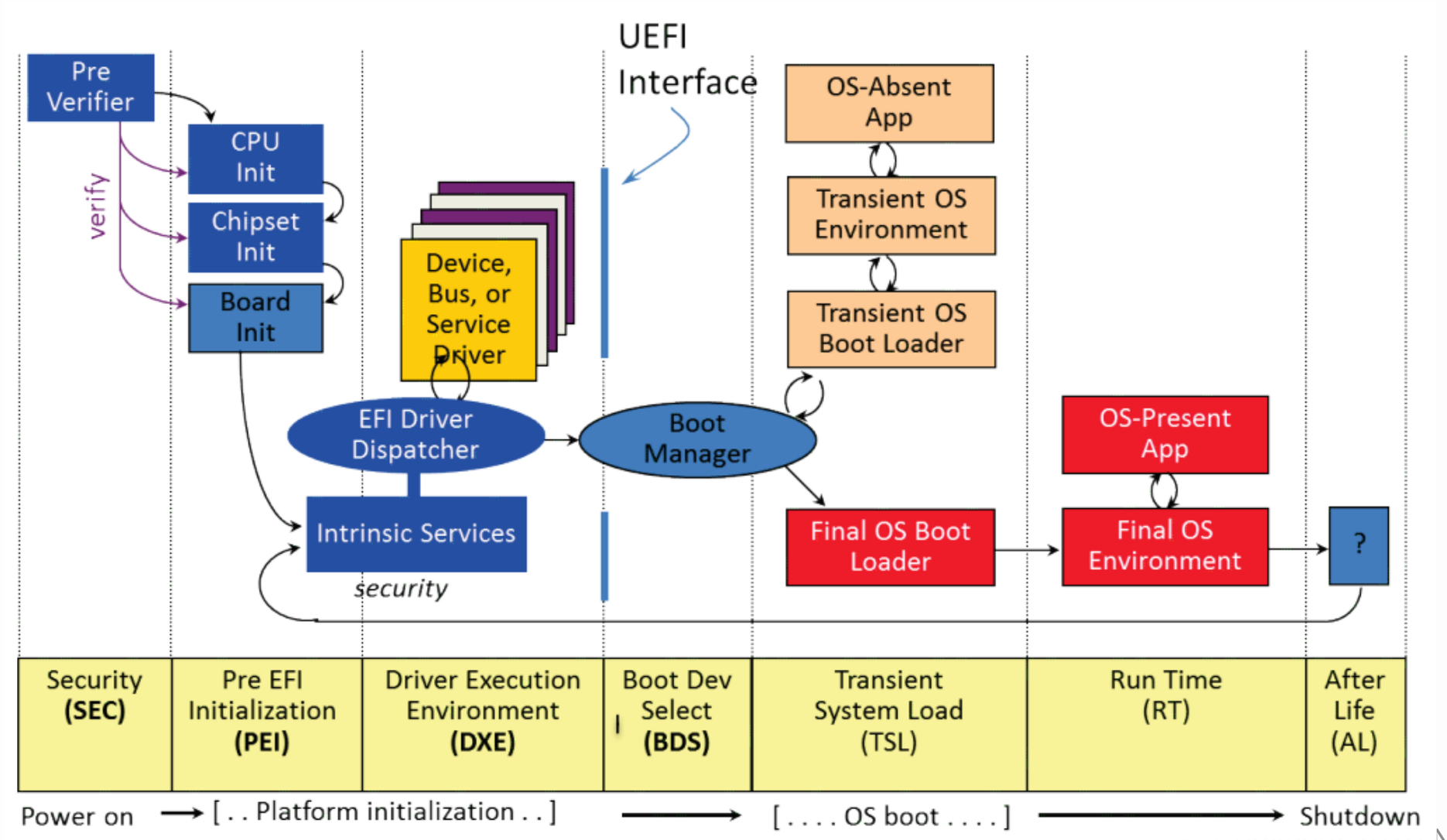
…. provided at least a rudimentary bit of research was done first

	Jan	Feb	Mar	Apr
2020	988	1,499	1,674	1,688
2019	1,616	1,212	1,010	1,623
2018	1,091	1,285	1,309	1,023
2017	870	1,035	1,361	942

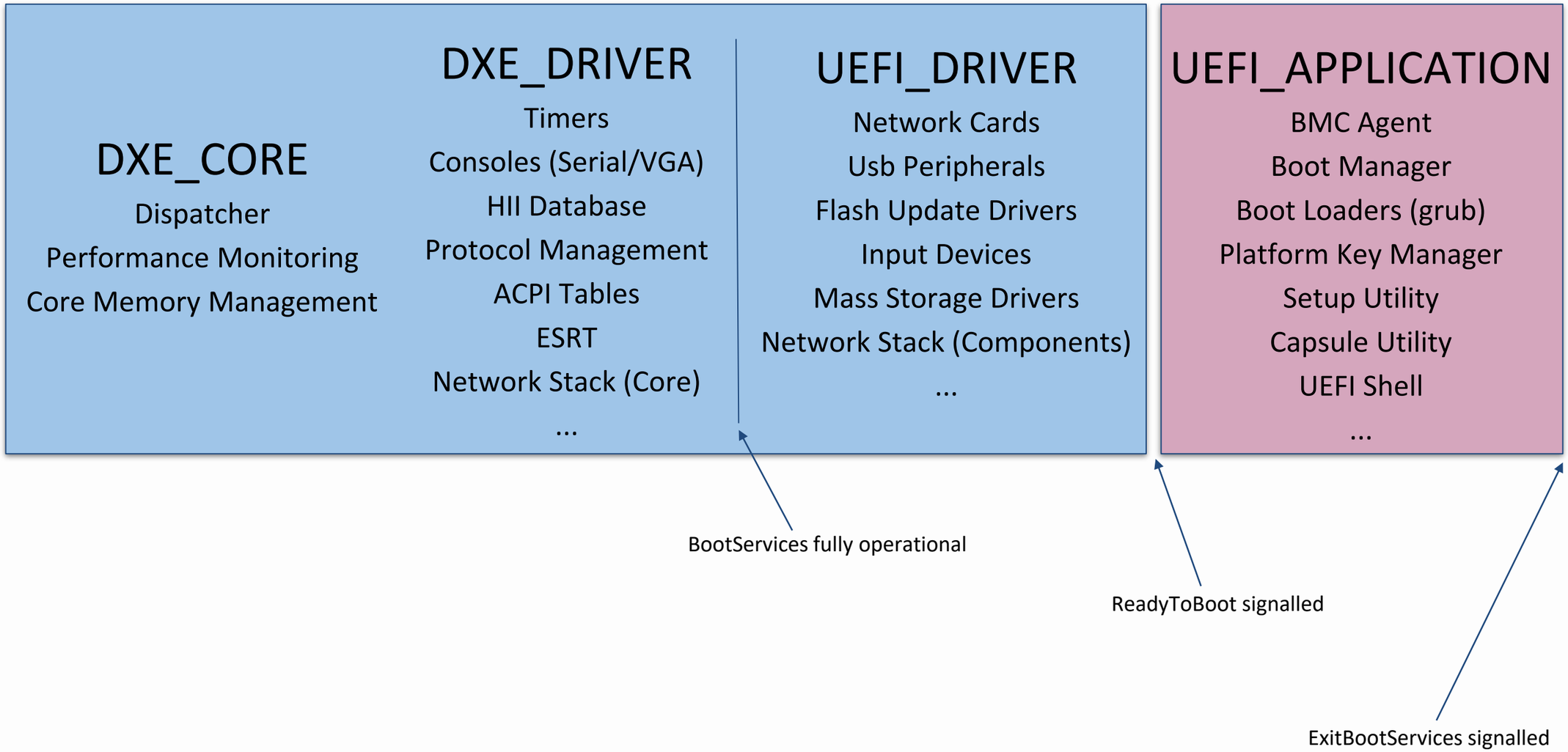
We are not in Kansas anymore ...

UEFI Fundamentals

Architecture Boot Flow



Driver Execution Context



Environment I

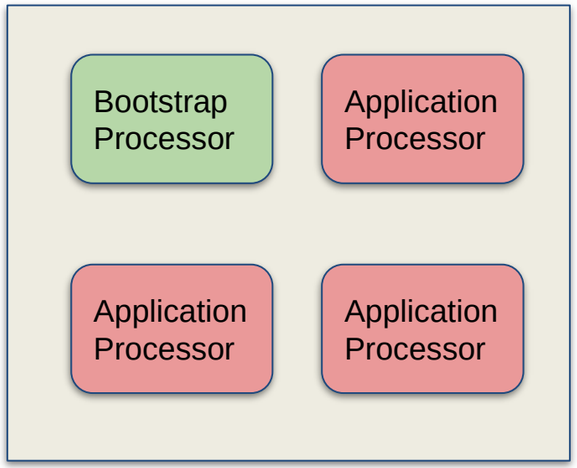
No memory protection

- Zero page might be mapped - NULL pointers, Interrupt Vector Table
- Can scribble anywhere in memory - Service Table, Rootkits etc.
- (Optional) Stack and heap protection

No privilege differentiation

- Files have no privileges, write and execute anything
- Secure Boot is not automatic!

Environment II



Single Threaded Execution

- Single queue data transfers
- No need for parallelization techniques

No Interrupts

- Actually, no Interrupts for you specifically (why?)
- Drivers are pull-based
- Data transfer driven by regular poll driven by platform stack

Event System Pre-emption

- Events can pre-empt execution at any point to execute callbacks
- Shared resources can get trampled over

Code Fundamentals I

```
//
// Build me an army worthy of Mordor
//
EFI_STATUS Status = gBS->LocateProtocol (
  &gIsengardProtocolGuid,
  NULL,
  (VOID **)&Isengard);

if (EFI_ERROR (Status)) {
  return Status;
}

Status = Isengard->BuildArmy (
  Isengard,
  EFI_WORTHY_OF_MORDOR,
  &Army);

if (EFI_ERROR (Status)) {
  return Status;
}
```

```
#define ISENGARD_PROTOCOL GUID \
        { 0x…

EFI_GUID gIsengardProtocolGuid;

Typedef struct _ISENGARD_PROTOCOL {
                EFI_STATUS (*BuildArmy)(...);
                // OtherStuff
} ISENGARD_PROTOCOL;
```

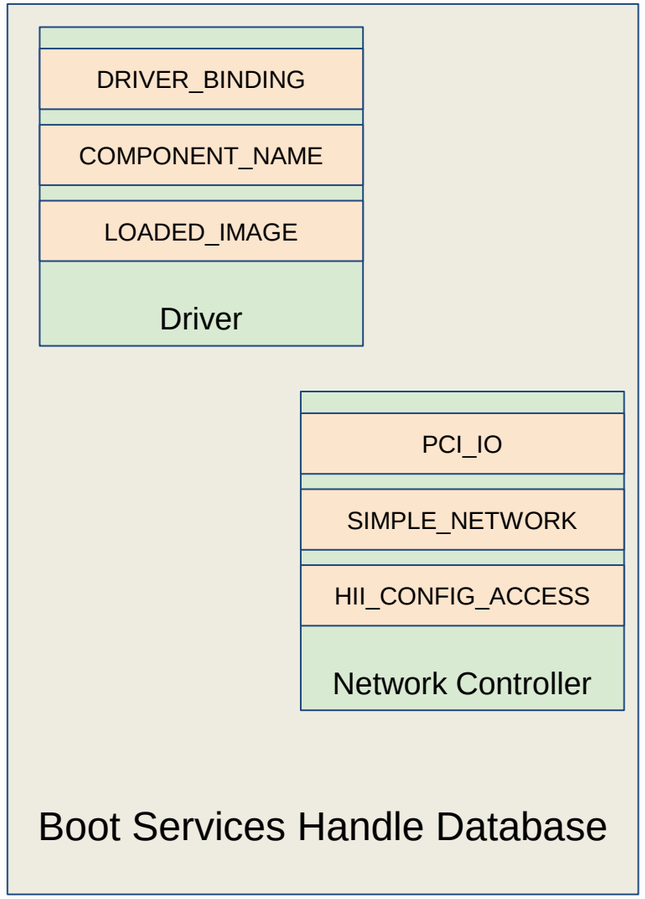
FooProtocol.h

Protocols

"How do I do X? There is a protocol for that"

- All interfaces are implemented in form of protocols
- Protocol structs can contain data, methods, pointers …
- Extensible - trivial to make new protocols
- Discoverable - Boot Services include protocol services (un/install, search)

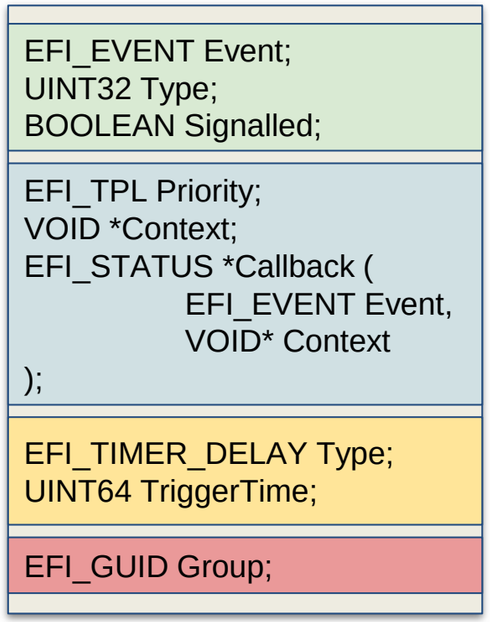
Code Fundamentals II



Handles

- There are no naked protocols
  - Identify protocol instances
  - Semantically group protocols
- Automatic and transparent
- EFI_HII_HANDLE != EFI_HANDLE
- SHELL_HANDLE != EFI_HANDLE

Code Fundamentals III



Events

- Asynchronous support
- Signal/Wait
  - More dynamic than protocols
  - More anonymous than protocols
- Timers
  - Once (Timeouts)
  - Recurring (Polling)
- Can be grouped using a GUID and signalled together
  - System-wide “announcements”

Code Fundamentals IV

TPL_HIGH_LEVEL

TPL_NOTIFY

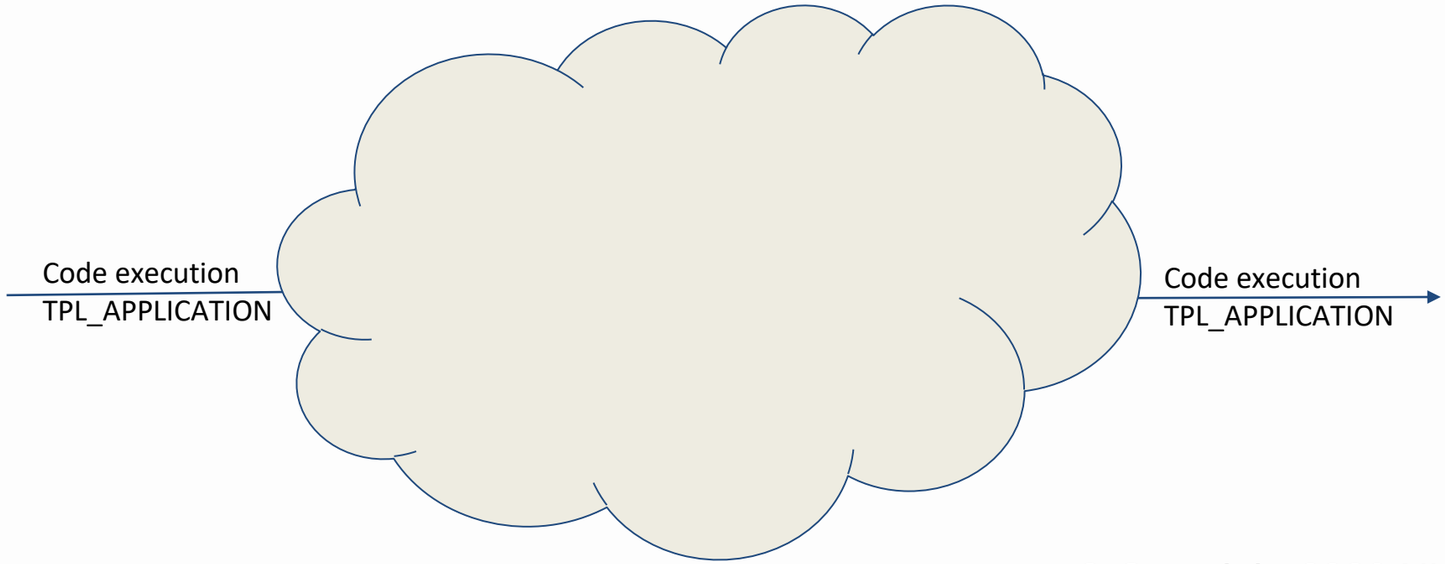
TPL_CALLBACK

TPL_DRIVER (deprecated)

TPL_APPLICATION

Task Priority Levels

- Prioritisation of callbacks and events
- TPL_CALLBACK and TPL_NOTIFY have event queues



Code Fundamentals IV

TPL_HIGH_LEVEL

TPL_NOTIFY

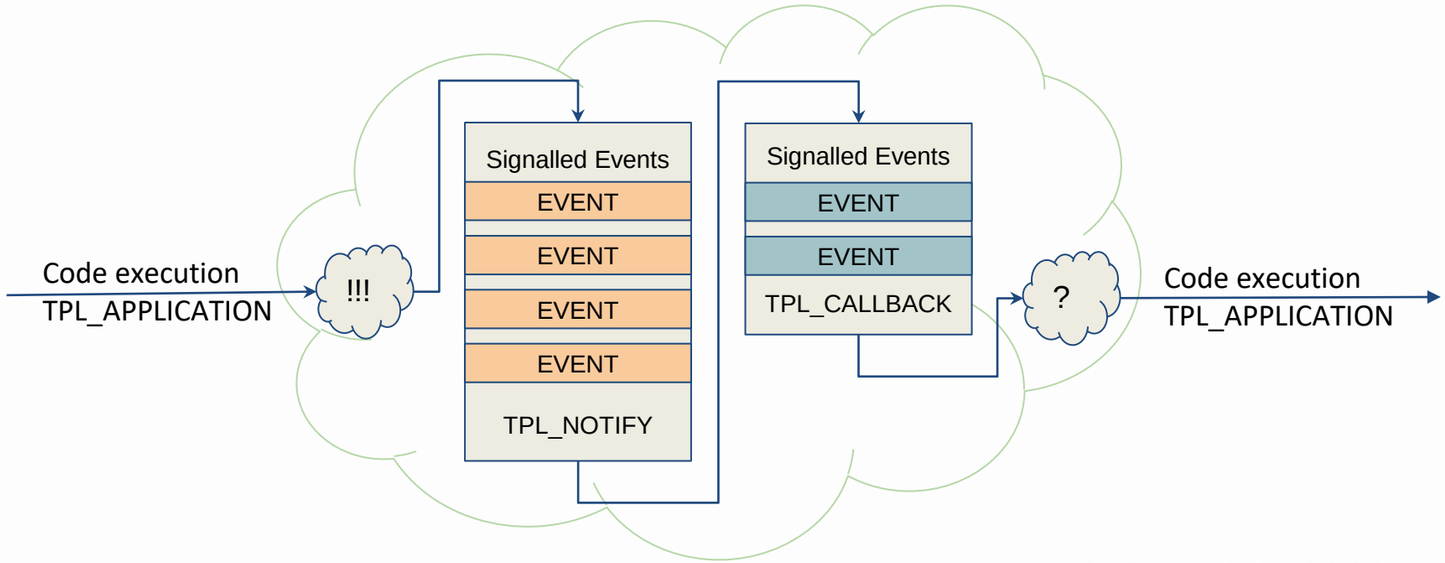
TPL_CALLBACK

TPL_DRIVER (deprecated)

TPL_APPLICATION

Task Priority Levels

- Prioritisation of callbacks and events
- TPL_CALLBACK and TPL_NOTIFY have event queues



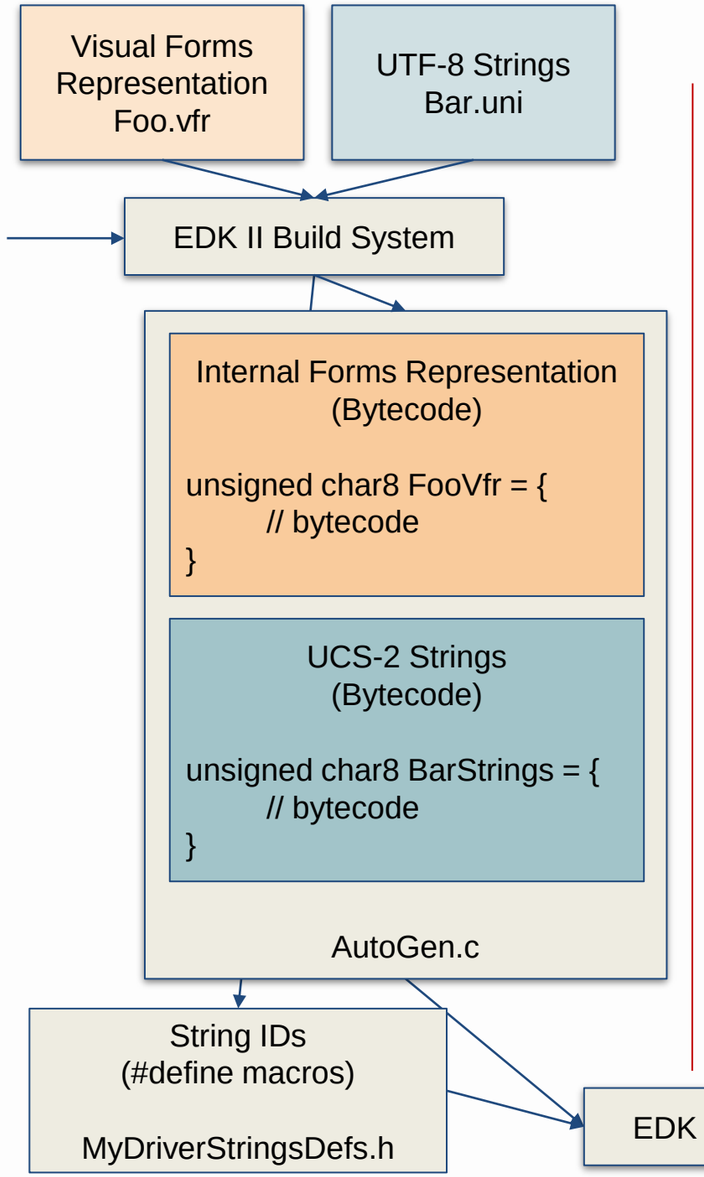
Now we transpile the .i file to .iii and carefully add the eye of newt ...

Human Interaction Interfaces

~~Concept~~ ~~Architecture~~

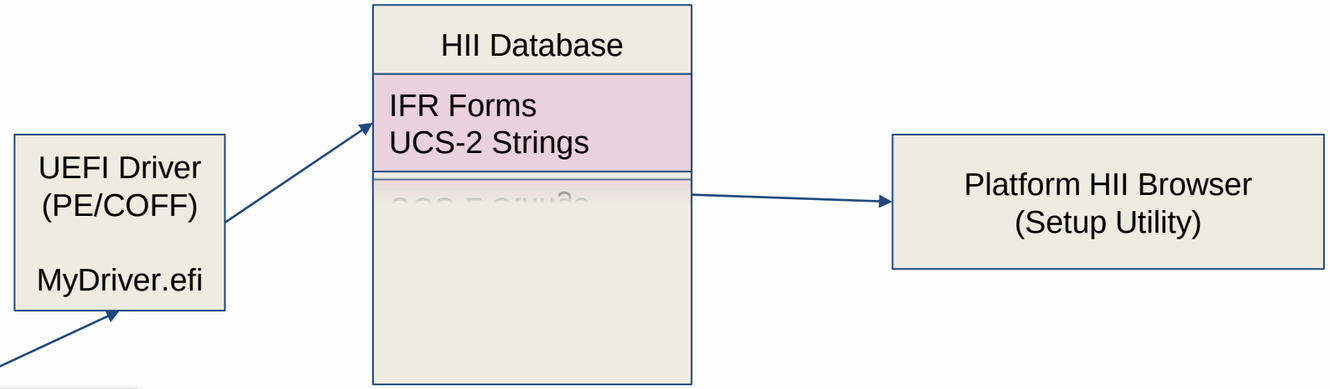
HII Rube-Goldberg Machine

~~Design~~



Modularise the forms

- Look and feel (Platform)
- Strings (Static/Driver)
- Form logic (Static/Driver)
- Configuration (Driver or Platform)



Things that will eat your face in the dark ...

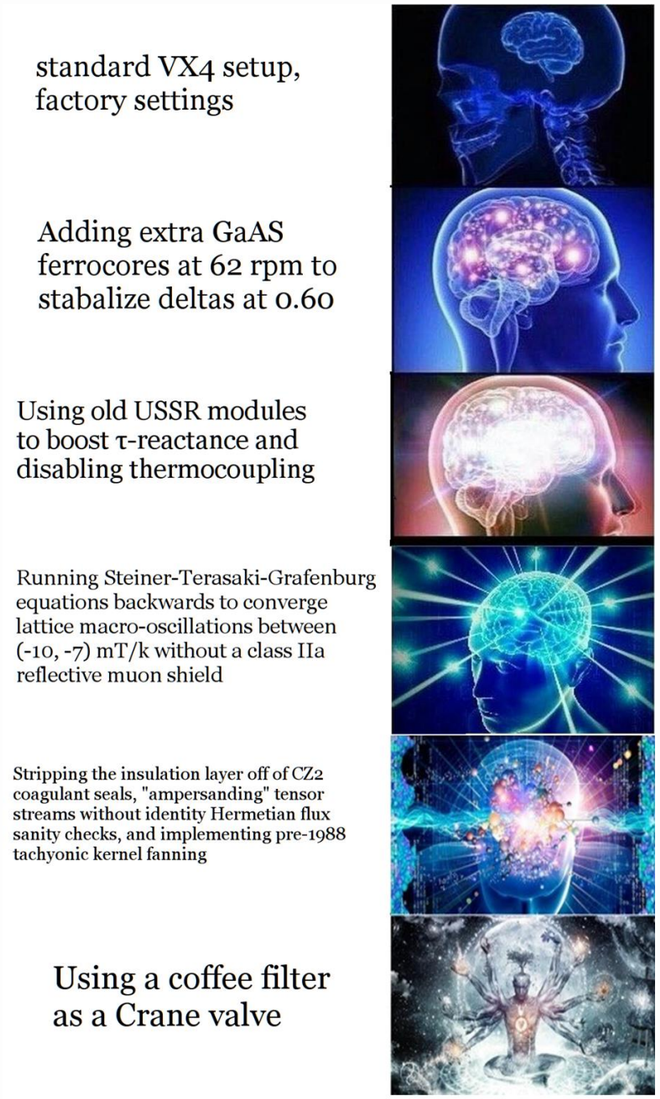
Common Issues

"Be conservative in what you send, be liberal in what you accept"

- No Platform Ever

~~General Abstract Nonsense~~

Inconsistent Nomenclature



BIOS

- (Not!Uefi) Legacy 16 bit, real mode boot execution environment
- (Not!OptionROM) Monolithic platform firmware loaded from baseboard
- (Not!CPUFirmware) Later stage generic firmware loaded by earlier stage platform firmware
- (Not!OS) Pre-boot code

UEFI

- (Not!BIOS) Extensible Firmware Interface Specification
- (Not!OS) Pre-boot code
- (Not!Uboot) EDK II based platform firmware

Option ROM

- (Not!UEFI) Legacy 16-bit executable blob rather than UEFI Driver
- (Not!BIOS) Vendor code loaded from device flash
- (Not!PlatformFlash) Expansion ROM BAR or flash on peripheral devices

HII Madness

Normal Mode

- Packed configuration blobs and string alignment
- UTF-16LE != UCS-2

Hard Mode

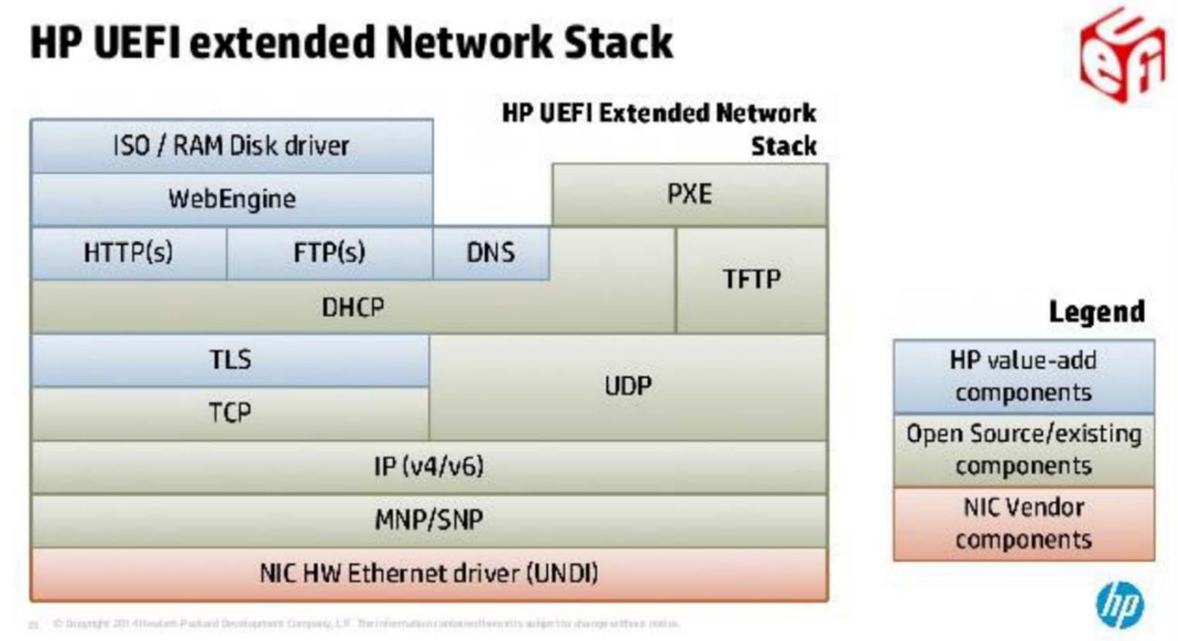
- Build errors of “Wagnerian Fierceness”
- Callbacks are unreliable
- It matters where you publish the package and the access protocol

“2020” Mode

- Values shared across formsets
- Patching IFRs dynamically
- RESET_REQUIRED does not ensure PCI reset

Enhanced BIOS Code

- Closed source
- Just close enough to EDK2 main tree to be maddening



Joys of OEM Deals

- VFR Content
- Strings Translations
- Proprietary Management Features
- Configuration Parity
- Expose the same information through a bazillion different interfaces
- “Thou shalt not write a runtime driver!”

Debugging On A Bad Day

Type: 2
Sev: 80
Class: 3
SubClass: D
Op: 6

Bad news - your platform does not have a crash handler.

Enjoy your invalid X64 instruction.

Debugging On A Good Day

Good news - Your platform has a crash handler, you get registers, stack trace and current task priority level.
Bad news - The platform crashed in closed-source “OEM Enhancement” or the BMC Agent after your driver fed it some garbage data.

Go back to square one. Do not pass go. Do not collect $200.

```
X64 Exception Type 02 - NMI Detected. Please check the IML for more details.

RCX=000000000000040B DX=000000000000040B RB=00000000000F4240 R9=0000000000000001
RSP=000000006B5E2CB0 BP=000000006B5E2D90 AX=0000000000ABB6A6 BX=000000000000040B
R10=0000000000000200 11=000000006B5E2CA0 12=0000000000000000 13=000000007124B018
R14=B000000000000007 15=0000000000000040 SI=0000000000CC7BA6 DI=0000000000400000
CR2=0000000000000000 CR3=000000005B542000 CR0=B0000013 CR4=00000668 CR8=00000000
CS=003B DS=0030 SS=0030 ES=0030 RFLAGS=00000246 MSR1D9=4B01 345=32C4 1C9=000A

LBRs From      To       From      To       From      To       From      To
01h  77E84B3F->740D1052 77E84B18->77E84B3B 77E82A77->77E84B0C 77E84CD2->77E82A6A
05h  77E82A65->77E84CD0 77E82AB4->77E82A65 77E84B44->77E82A7C 77E84B18->77E84B3B
09h  77E82A77->77E84B0C 77E84CD2->77E82A6A 77E82A65->77E84CD0 77E82AB4->77E82A65
0Dh  77E84B44->77E82A7C 77E84B18->77E84B3B 77E82A77->77E84B0C 740D105D->784673B0

CALL ImageBase        ImageName+Offset
00h  0000000077E78000 SystemHealthDxe+00CB3Fh
01h  0000000077E78000 SystemHealthDxe+00AA7Ch
02h  0000000077E78000 SystemHealthDxe+00AAC8h
03h  0000000077E78000 SystemHealthDxe+005FC4h
04h  0000000077DC2000 BdsDxe+048947h
05h  0000000077DC2000 BdsDxe+04BA5Eh
06h  0000000077DC2000 BdsDxe+027B4Eh
07h  0000000010000000 No Image Information
```

```
A system restart is required.                                    BIOS rev 1.7.11
The system detected an exception during the UEFI pre-boot environment.
Press ENTER on the serial console (@115200 8-N-1) for detailed exception info.
--------------------------------------------------------------------------------
Type: Page fault (14) Source: Software (UEFI0012)
AX=0000000000000000 BX=0000000000000000 SI=000000005A40990C DI=0000000076DDE6B0
CX=00000000720EA080 DX=0000000072631218 RB=0000000072630618 R9=0000000072630618
BP=000000005A40990C SP=0000000076DDE680 IP=00000000720B4B2B Flags=00010246
(CurrentTPL = 10, TSC ticks since last event 00000006FD64)

LBRfr1 73FDEA0A CpuArchDxe.efi +00BA0A
LBRfr0 73FDEAFF CpuArchDxe.efi +00BAFF
-->rip 720B4B2B x64sas2.efi +00CB2B <-- Crash occurred here
   s00 76DFA581 DxeCore.efi +01B581
   s01 76DF9707 DxeCore.efi +01A707
   s02 76DF9AB7 DxeCore.efi +01AAB7
   s03 729617B3 DellBdsDxe.efi +0157B3
   s04 729617B9 DellBdsDxe.efi +0157B9
   s05 7296183D DellBdsDxe.efi +01583D
   s06 7294E993 DellBdsDxe.efi +002993
   s07 7294FCBA DellBdsDxe.efi +003CBA
   s08 76DE0C62 DxeCore.efi +001C62
   s09 76DDF873 DxeCore.efi +000873
       Stack Unwind Complete after 12 Frames
```


Driver Model

- Bunch of old protocols
  - Might crash platform if implemented
- Quite a few protocols are not useful
  - Might crash platform if not implemented
- Multiple Devices
  - Might load one driver, might replicate
  - Devices might load different versions of driver
  - Driver Override Protocols not always honored
  - Shared resources?
  - Inter-device communication?

Network Stack

> You want to write a network driver
> Spec/DWG says you should prefer SNP over UNDI
> NII is optional, used to expose PXE blob for UNDI
> BIOS not consider you a network card without NII
> If you install NII, BIOS will try and build its SNP on it
> PXE blob location in NII is a raw pointer
> You don’t have PXE blob as SNP device
> BIOS *might* ignore NII already open BY_DRIVER
> PXE Client will use NII version fields for PXE/BOOTP

> You are the only person writing an SNP driver

Self-Certification Tests

Mostly designed for IBVs or OEMs

- Test API, not functionality
- Small set runnable by IHVs

Now actually open-source (Good!)

ExitBootServices Delights

- No Timers
  - No timeouts
  - No polling
- No Memory Allocation
  - Hope you have no dynamic message boxes
- What if there is no ExitBootServices?
  - Never do device-driven periodic DMA
  - Limit extant descriptors

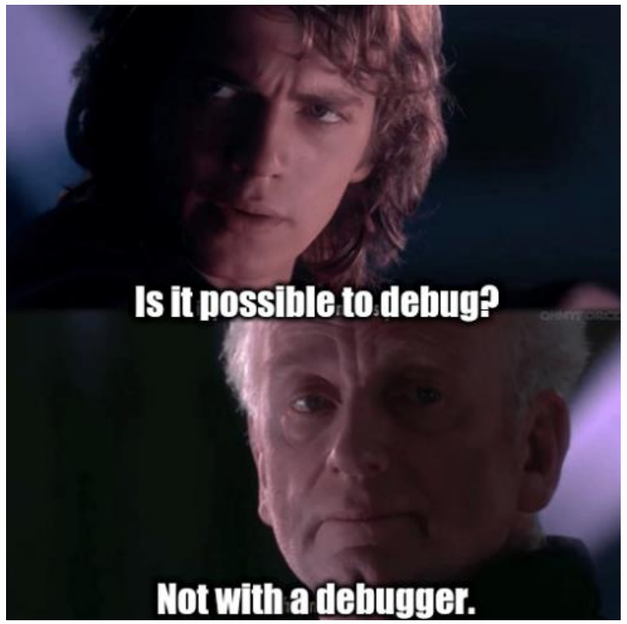
Life Pro Tips ...

Force Multipliers

The Plugfest

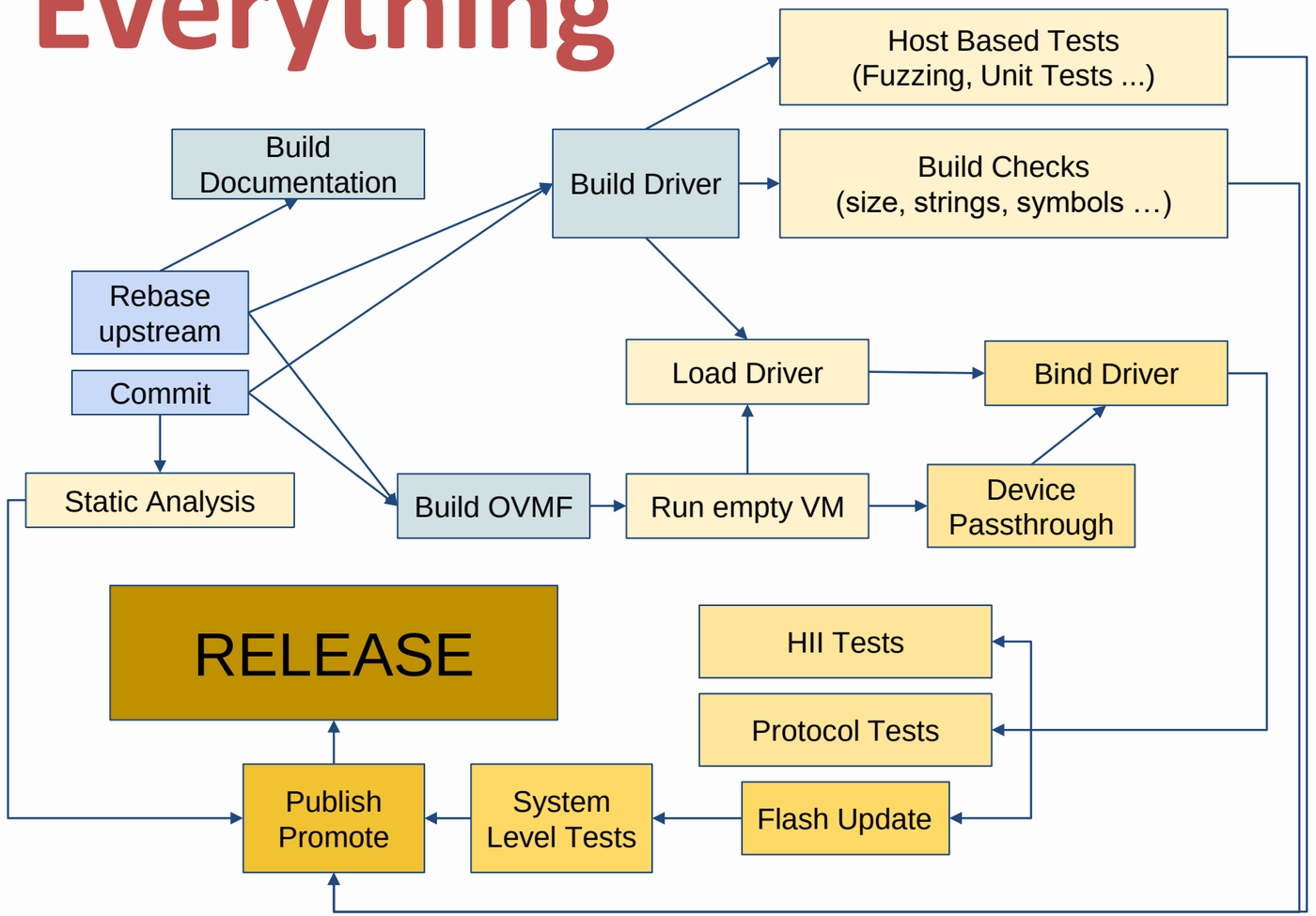
- People
  - You get to know the engineers on the other side
  - Those email addresses are people too
- Hardware
  - Much easier to patch pre-production BIOS
  - Ask Ard about Chucky
- It’s not just you
  - War stories are gold

Embrace the Print



- Print everywhere
  - Info at start of function
  - Print return codes
  - Print strings, device paths, stages of functions
- Filter by verbosity, make a library out of it
- Use TPL_HIGH_LEVEL
- Output to all kinds of places
  - Shunt down the serial port (bypass SERIAL_IO)
  - Write to consoles
  - Write to memory and dump once you can

Automate Everything



Code Intelligence

- Doxygen (Driver & Upstream)
- Static Analysis
- IDE with semantic understanding
- Modularise and encapsulate in Libraries (modules are cheap)
  - Debug information
  - Mailbox communication
  - Resources handling
- Store .uni files a UTF-8
- Build only your module (build -m foo.inf) in automation

Crash Handlers

- -b DEBUG
  - (.debug = .efi + symbols)
  - Load in GDB (disas /s)
- Learn (a bit of) architecture and calling conventions
  - Which registers correspond to parameters, return values ...
  - AFAFAFAF is poison
- Ask your new Plugfest friends to decode a crash for you

Create Protocols & Handles

- Protocols are your friends
  - Private structs
  - Mailbox communication
  - Version, Checksum and Magic!
- Handles are free
  - Separate logical parts of your device (PCI, NIC, HII, Flash)

Do not try this at home ….

Interesting Examples

Interesting Examples

SNP.Mode.MaxPacketSize = 1514	No iSCSI disk in Windows
Improve SNP efficiency	DoS platform with ARP
SNP & NII on different handles	iPXE scrubs the iBFT table
Periodically refresh HII_CONFIG_ACCESS	Setup Utility now blinks once a second
Forget EFIAPI function attribute	Crash in NBP which is still using your SNP
Use grub2.02 beta2	MNP steals data from grub
BIOS changes MNP poll timeout to 1s	Standard TFTP server retransmit is 1s
Publish FMPv3 descriptors	Platform crashes when gathering inventory

Questions?

Thanks for attending the UEFI 2020 Virtual Plugfest

For more information on UEFI Forum and UEFI Specifications, visit http://www.uefi.org

presented by

NUVIA