presented by

Next Frontiers in Firmware Standardization: OCP Open Platform Firmware Project Efforts and Their Effect on UEFI

UEFI 2025 Developers Conference & Plugfest

October 9, 2025

Felix Polyudov

Meet the Presenters

Felix Polyudov

AMI Fellow, Head of Firmware Core Architecture

Felix Polyudov is a Fellow at AMI and serves as the Head of Firmware Core Architecture, bringing over 25 years of expertise in firmware architecture and development. As an experienced thought leader, Felix has driven innovation across platform firmware, shaping foundational technologies that power today's computing systems. His leadership continues to influence the evolution of UEFI and next-generation firmware standards.

Agenda

- Open Platform Firmware (OPF) Project Overview
- Open Silicon Firmware Interface
- Unified Platform Configuration Interface

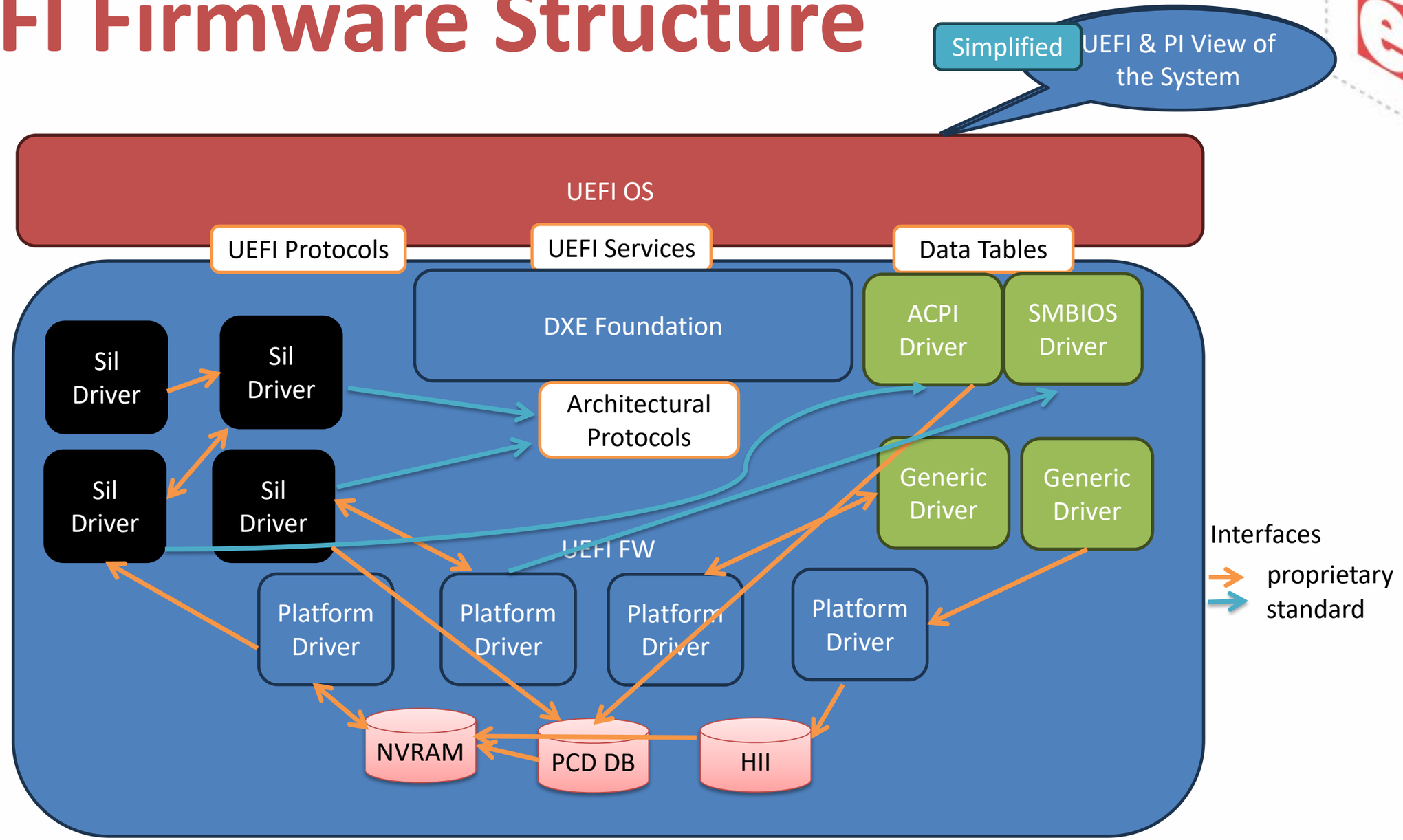
Why OPF?

- Open Platform Firmware project (OPF) is one of the projects under the Open Compute Project (OCP) umbrella
- Both UEFI and OPF are industry bodies that are working on system firmware
  - UEFI and OPF may at times look at things from different angles or have different approaches, but both are trying to improve the firmware ecosystem
- The purpose of this presentation is to:
  - Spread the word
    - Both organizations may benefit from cross-pollination
    - Issues that OPF is trying to tackle should be is of interest to the UEFI community
  - Review implications of the OPF work for the UEFI specification

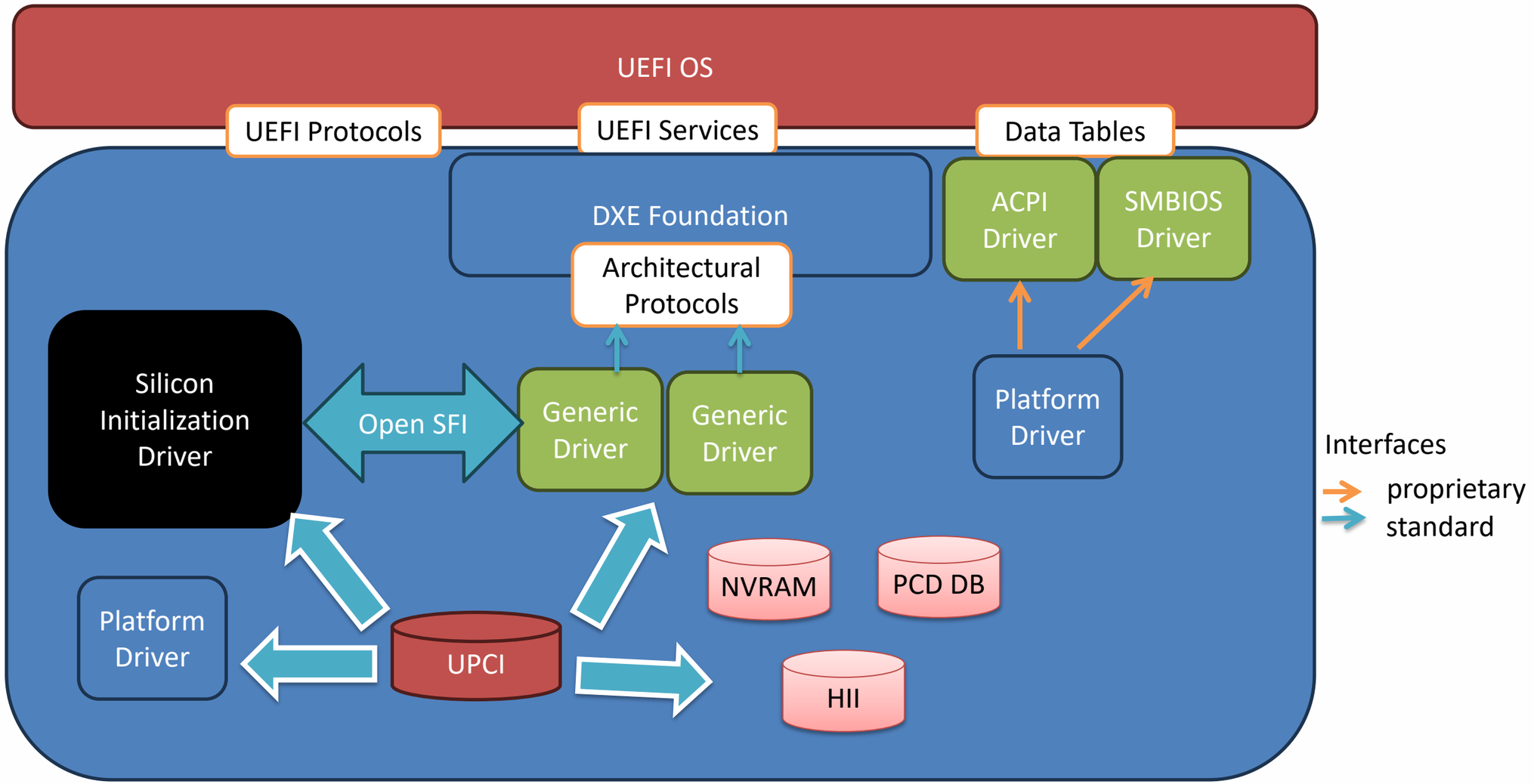
OPF Workstreams

- Two new OPF workstreams have been started in 2025
  - Open Silicon Firmware Interface (openSFI)
    - Attempts to define a unified interface between host firmware and silicon initialization components
  - Unified Platform Configuration Inteface (UPCI)
    - Attemps to define a unified platform configuration ecosystem

UEFI Firmware Structure



UEFI FW Structure. OPF Vision



Open Silicon Firmware Interface

- OpenSFI Specification is a work in progress. v0.3 is about to be released. Interfaces and parameters may change before the final publication.
- Key goal: streamline firmware implementation by defining interface between host firmware and silicon initialization code
- The specification defines high level silicon API
  - InitializeSilicon
  - ExecuteSiliconRuntimeOp

InitializeSilicon

```
int InitializeSilicon(
    void *Input,
    void *Output,
    uint32_t Stage,
    uint32_t SoCID
);
typedef enum {
 SFI_INIT_STAGE_LIB_INIT,
 SFI_INIT_STAGE_PRE_MEM,
 SFI_INIT_STAGE_MEM_INIT,
 SFI_INIT_STAGE_IO_INIT,
 SFI_INIT_STAGE_CPU_START,
 SFI_INIT_STAGE_SEC_INIT,
 SFI_INIT_STAGE_FINALIZE
} INIT_STAGES;
```

- An interface to call sequential initialization steps
- C-based
- `Stage` - initialization step to perform
- `SoCID` - Vendor-specific identifier for target System-on-chip configuration
- Format of the `Input/Output` buffers is to be defined by the future versions of the specification
- Returns error code or zero on success

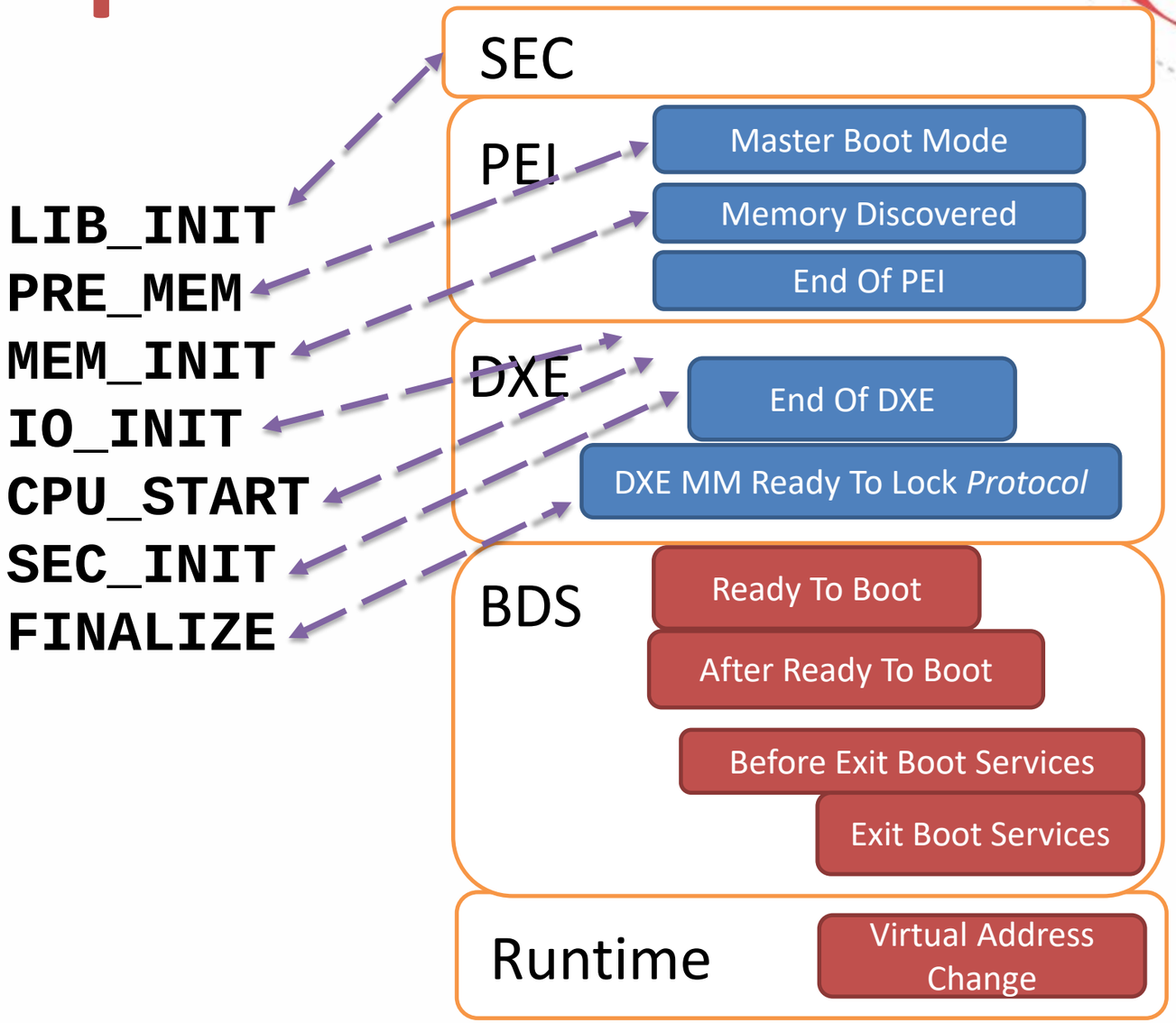
ExecuteSiliconRuntimeOp

```
int ExecuteSiliconRuntimeOp(
    void *Input,
    void *Output,
    uint32_t RuntimeStageID,
    uint32_t SoCID
);
```

- Both functions have the same interface
- InitializeSilicon is intended for a one-time initialization operations
- ExecuteSiliconRuntimeOp is intended for lifecycle post-initialization operations such as telemetry and reconfiguration

Impact on UEFI Specification

- Protocol/PPI Wrappers
  - OpenSFI is a C API
  - UEFI is an ABI with bindings for multiple CPU architectures
  - OpenSFI UEFI Protocol/PPI can be defined to maintain the same level of abstraction
- Aligning OpenSFI initialization phases with UEFI boot flow events
  - To ensure interoperability between the two specifications, OpenSFI initialization phases should fit into the UEFI boot flow



Unified Platform Configuration Inteface

- Key Goals
  - Simplify platform customization process
  - Provide cleaner abstractions between generic, silicon-specific, and platform-specific code (IFW/IBV, SiV/IHV, and OEM/OEM/CSP code)

- Scope (How UPCI defines *Platform Configuration*)
  - SoC Configuration (customization of silicon components by supplying data to silicon initialization interfaces)
  - Information about buses/interfaces that cannot be auto-enumerated/configured (I2C, I3C, SPI, etc).
  - Information about peripherals beyond SoC (TPM, EC, Super I/O)
  - Board layout (PCIe® Slots, DIMM Slots, I/O ports)
  - Board information (Manufacturer, UUID, etc.)
  - Vendor Policies (device/port policies, FW policies)

UPCI Approach

- UPCI is a data interface
  - No dependencies on a specific programing language, execution environment, or firmware architecture
- UPCI aims to define a human-readable and machine-readable configuration data description formats
  - Human-readable representation (HRR)
    - Facilitates multi-vendor collaboration
    - Simplifies platform customization
    - Enables reuse of board configuration components
    - Provides foundation for rich tooling
  - Machine-readable representation (MRR)
    - Single source of platform configuration
    - Instrumental in addressing platform construction and life-cycle issues such as size efficient multi-SKU configuration and configuration migration during firmware updates

UPCI Approach

The UPCI effort involves several activities

- UPCI Framework
  - Define UPCI Human-Readable and Machine-Readable formats
- Data Structures
  - Use the framework to standardize pieces of platform configuration
    - CPU, memory, PCIe®, TPM, etc.
- Interoperability with the industry standards
  - Ensure that UPCI structures are useable with the other standards either directly or after a universal export procedure
- Based on ideas floating in the UEFI Community
  - YAML-based configuration
    - Presented by Vincent Zimmer during the UEFI Fall 2023 Developers Conference
    - Part of the Universal Scalable Firmware specification
    - Used by the Slim Bootloader Project

Platform Configuration in UEFI FW

- UEFI firmware components use several sources of configuration data
  - UEFI Variables (a.k.a. NVRAM)
    - Runtime interface: UEFI Variable Services
    - Developer's tooling: No standard tooling
  - PCD Tokens
    - Runtime interface: PCD Protocol/PPI
    - Developer's tooling: DSC file
  - HII Config Routing Protocol
    - Developers tooling: VFR/UNI, C code
  - Policy Protocols
    - Developers tooling: C code
- Usage of multiple configuration methods…
  - Complicates development of the provisioning, manufacturing, and configuration tooling
  - Increases platform customization effort
- No simple mechanism to export config data for usage with other industry standards such as ACPI or RedFish
  - Manual export (C code with field-by-field initialization)
    - Low efficiency, error-prone
  - HII-based export
    - High level of complexity
    - Depends on setup controls

UPCI Impact on UEFI

- UEFI Specification
  - A protocol and PPI to get/set UPCI data can be considered to standardize access to UPCI data structures
  - New IFR *varstore* type can be introduced to access UPCI data structures from HII
- EDK2
  - Implementation of DynamicTablesPkg and RedfishClientPkg can be streamlined by leveraging configuration data in UPCI format
  - Long term: PEIM/Drivers should use UPCI as a preferred configuration interface
  - Short term: To ensure backward compatibility platform firmware can publish instances of Policy and HII Config Access Protocols built on top of UPCI

Questions?

References

- OPF Wiki
- OPF Mailing list
- Firmware Configuration – Past, Present, and Future (video)
- USF YAML Format Boot Configuration