Enterprise SW Update

Systems & Software Group (SSG)

ARM Software Charter

Lead

- Lead the Open Source with power efficient solutions
- Investing in community stewardship
- Evolving IP products through Software innovation

Standardise

- Timely implementations of ARM-based standards
- Drive open standards which reduce fragmentation
- Producing reference Software

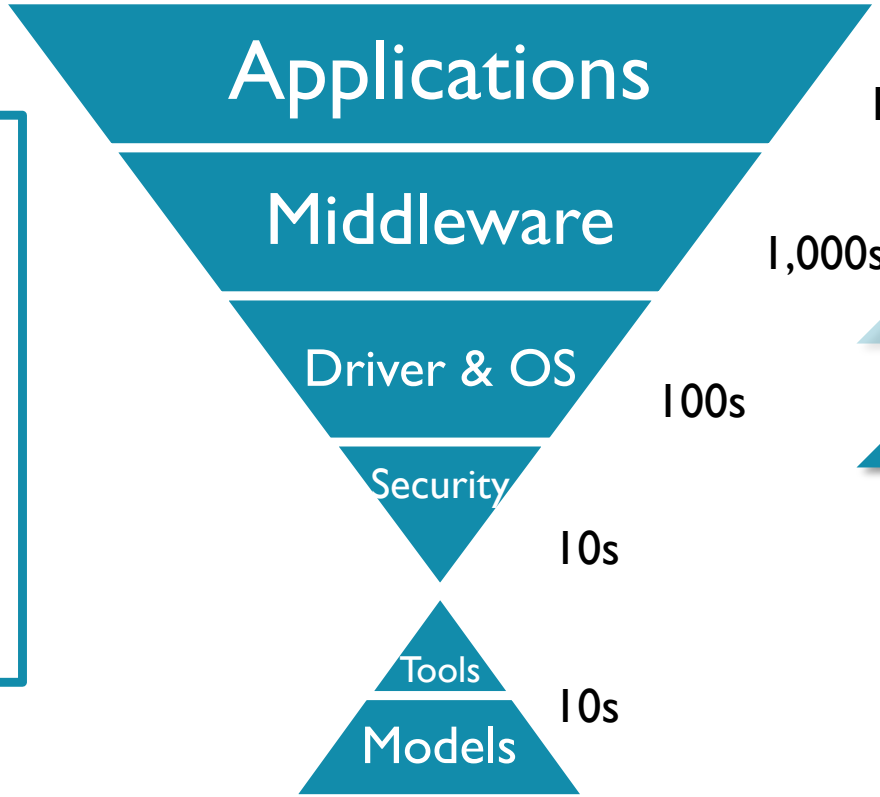
Enable

- Enable ARM IP
- Add ARM IP support to Virtualisation Software
- Performance & Efficiency optimisation

ARM and Partners Enable the Ecosystem

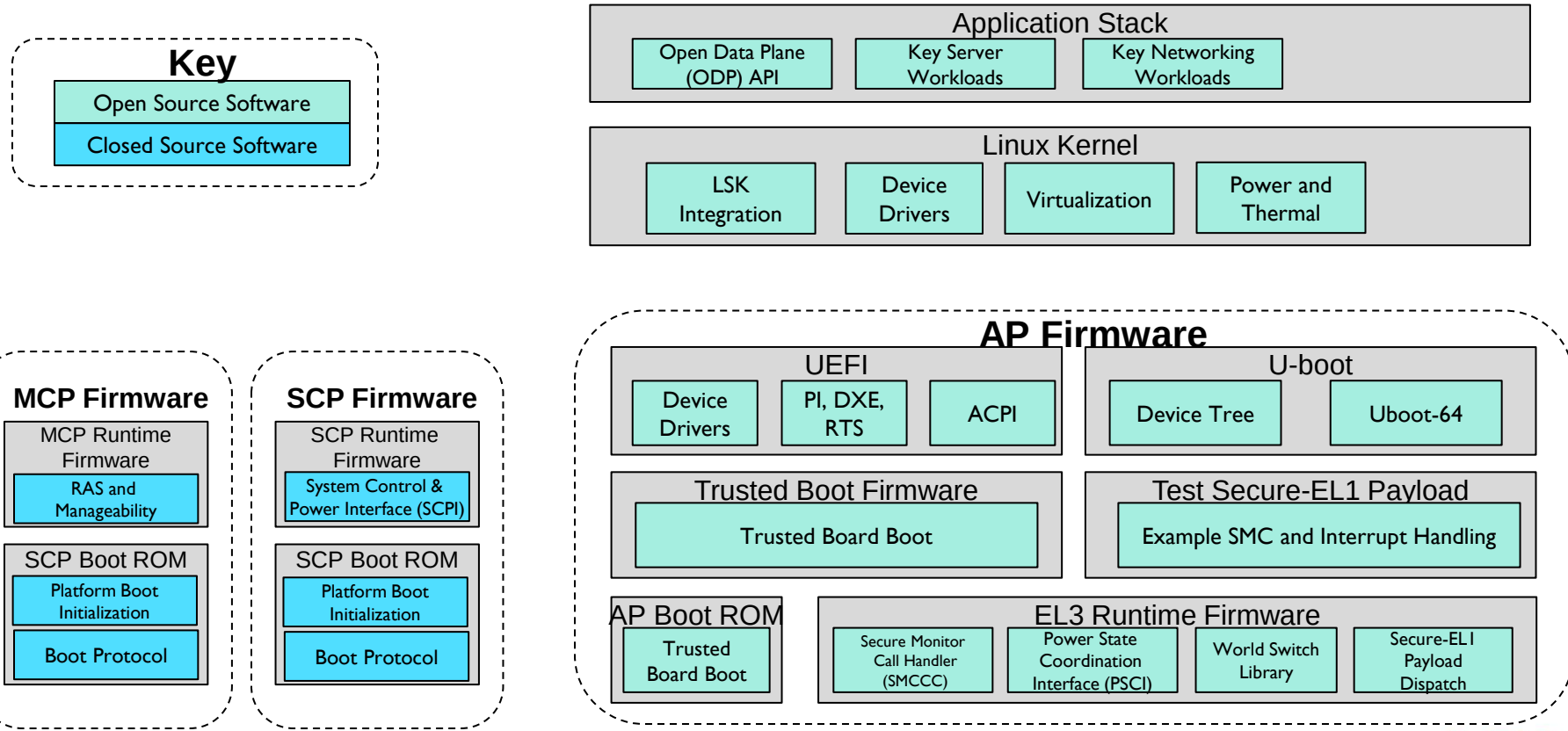
ARM Ecosystem and SW Strategy

- A collective effort from ARM and partners
- ARM focuses on enabling key building blocks
- The partnership ensures the entire ecosystem is enabled
- Linaro is all about this



Subsystems

Ashbrook Software Overview



Software deliverables

- System Control Processor (SCP) source, and guide to integration/extension
- ARM Trusted Firmware source ported to subsystem
- u-boot or UEFI source ported to subsystem
- Linaro Stable Kernel (LSK) ported to subsystem
  - Version appropriate to Enterprise – so currently v3.18 LSK based
- ODP API v1.0 ported to subsystem
- Release documentation, build instructions

- All with the latest ARM features pre-integrated
  - Security, Power, Performance

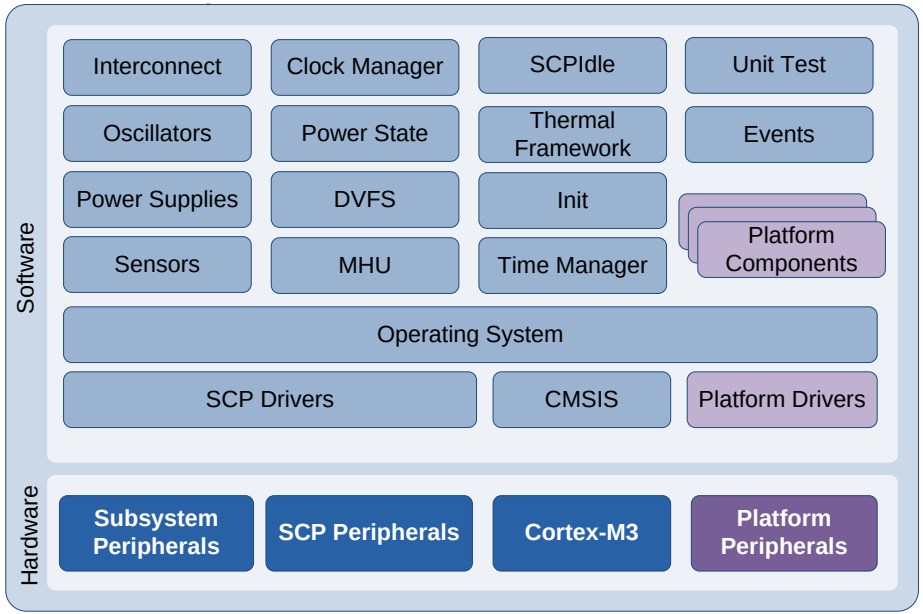
Software Release Details

- Subsystem software releases are "development" quality designed to be a starting point for partner software work
  - Integrated all the software common to all SoCs using that subsystem (open-source + proprietary)
  - Validated within ARMs test framework
  - Take the ARM software releases; modify, extend and develop software stack
- For a subsystem we cannot validate all features of a derived SoC
  - Hence all releases are 'development' quality
  - Tested on FVP, emulation and ARM Development Platforms (where possible)
  - Test core functionality and ARM provided interfaces
- Maximise use of standard public open-source software
  - Validated on many other systems
  - Our releases contain just the small deltas from the open-source software
  - SCP is not an open-source component, licensed as part of the subsystem

System Control Processor

System Control Processor

- SCP Firmware contains
  - Frameworks and components related to the Armstrong design
  - Customer extension components related to the platform built around the Armstrong



Customer Extensions to Subsytem

Subsystem Frameworks and Components

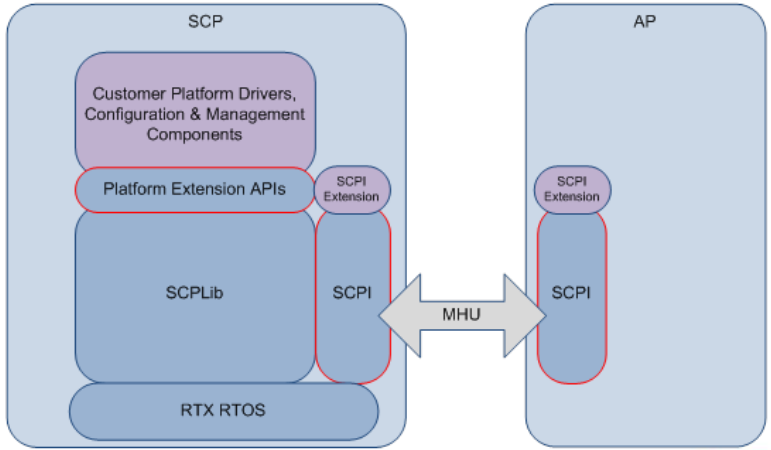
ARM

System Control Processor Overview

- Cortex M3 based subsystem controlled by the AP using the System Control & Power Interface (SCPI) API
- Main responsibilities:
  - Initiating power on sequence and system start up
  - Initial platform configuration
  - Clock and regulator management
  - Servicing power state requests from OSPM
  - Managing a consistent set of hardware states for hardware
  - Handling hardware wake-up requests (timers, interrupts)
  - Managing transitions between operating points to support DVFS
  - Save\restore of state in interconnect and customer integrated DMC during power state transitions
  - Reconciling any capability, performance and wakeup latency requirements with other resource management (PMIC, PLL, sleep on/off/response times)
  - Thermal "loop" (temperature sensor) control

SCP Interfaces

- Two main interface points to core Library
- AP control of the SCP through the use of the SCPI API and Message Handling Unit (MHU) driver, with the ability to add customer's platform-specific commands
- Platform specific extensions via the Platform Extension APIs to allow integration of platform-specific components
- Platform Drivers for customer-specific hardware under SCP control
  - e.g. Power Supplies, Oscillators, Sensors
- Platform Management Components
  - Abstract platform drivers from SCP components
  - Enables SCPI extension handling
- Configuration
  - Start-up options
  - Memory layout



Message Handling Unit (MHU)

- Allows inter-processor communication between SCP and AP
- 31-bit register area for messages + interrupt mechanism
  - For messages that require extra payload, (Non-Secure) Scratch RAM and Secure RAM are used
- 2 priority levels (High and Low) plus a secure channel in both directions
  - Up to 6 simultaneous messages
  - Secure messages processed first, then high priority, then low priority
  - Secure channel only accessible to secure world software
  - Independent to channel priority (in the software)
- Messages are decoded into SCPI commands

Manageability Control Processor (MCP)

What is an MCP?

- MCP stands for Management Control Processor
- The MCP is separate IP block that provides the following functionality:
  - On chip management
  - Agentless management
  - RAS on chip
  - Separation of trust domains (MCP = non-trusted; SCP = trusted)
  - Provides single interface to the platform BMC
- MCP current implementation is a Cortex M7

Why incorporate MCP into SOC design?

- Create Enterprise Standards
  - BMC to SOC communications are not standard today. The MCP provides a means to standardize this interface by reducing the need for BMC and platform vendors to support different interfaces for every SOC vendor.
- ARM Provided Reference Code
  - ARM will provide reference code with the MCP to guide partners on how to implement on-chip management and RAS
  - SOC vendors can choose to use reference code as a base for their own implementation
- Prepare for the Future
  - BMC operations will eventually be consolidated into the SOC as having a separate BMC on the platform increases the platform cost. The MCP provides a place for BMC vendors to place their code as this transition takes place.
  - The Cortex-M7 has an option for a GbE interface that can leverage a network accelerators NCSI interface for out-of-band communications. For low cost servers, this could remove the need for a platform BMC.

ARM

RAS/Manageability

RAS in ARM Architecture

- ARM architecture for A processors has had very little to say about RAS functionality prior to v8.2
  - Errors reported to software via fault status register and aborts and system error interrupts (SEI)
  - Additional important information on the faults are in implementation-specific registers in the core and in devices (such as memory controllers)
- But, many implementations had RAS features
  - Primarily, ECC on caches and main memory
- Additional RAS features could have been implemented as well, such as
  - Memory scrubbing
  - Bus error protection
  - Fault isolation
- One useful RAS mechanism hasn't usually been implemented, an error fence to aid in containing errors to the software that caused them

v8.2 RAS Extensions Summary

A language for describing RAS

- Detection
- Consumption
- Containment
- Isolation
- Recovery
- Poisoning

Error Reporting

- Synchronous and asynchronous external aborts (SError)
- Error recovery interrupts
- Firmware-first error handling
- Standard syndrome registers for error handler

- Provides a framework for building RAS features
- Few actual additions to architecture
- Does not specify other RAS features in implementations

Error Memory Barrier

- For isolating uncontainable errors to the software environment that caused them.
- EMB – new error barrier instruction
- Issued before switching VM or process

Fault Handling

- Fault handling interrupts
- Error records

- *Today's systems require implementation-specific software*

- Required for v8.2
- Status: v8.2 is beta

ARM

RAS Architecture Extensions

- Give tools to build RAS enabled systems, but implementations need to fill in quite a bit of detail:
  - Which portions of the design should be protected and against which errors?
  - How to join IP-level RAS features with system-level behaviors?
  - Where to send fault handling and error recovery interrupts?
  - Is it possible to recover from a particular error and how to do so?
  - Etc.
- So, quite a bit is needed to bridge the low-level tools of the RAS extensions to the needs of an end system

Hardware for RAS

- Error detection
  - Detectors as appropriate for system design and for FIT goal
  - Cover most likely fault sources (e.g. DRAM data corruption or watchdog timeouts)
- Error correction
  - Some detected errors can be corrected; and correction lowers FIT rate
  - Correction can be performed by hardware, firmware or software (most usually hardware)
- Signaling software – Faults and errors are signaled to software via:
  - Abort
  - System error interrupt
  - Fault handling interrupt †,
  - Error recovery interrupt †

† New in the v8.2 RAS extensions

Hardware for RAS (2)

- Recording information about the error
  - At the point of detection, usually store information about the error into registers
  - RAS architecture extensions give standard ways to access this information and slightly standardize the information recorded
- Signaling deferred errors
  - A system that supports deferred errors will need a way to signal poison between IP
- Logging
  - Errors need to be logged in non-volatile storage, so such storage must exist (somewhere)

Legacy vs v8.2

- In v8.2, when RAS extensions are in the architecture, RAS becomes more standard
  - Defined interfaces for signaling errors to software and for getting error reports
  - New instruction ESB to fence errors before changing software environment (e.g.VM switch)
- Everything RAS could be done by implementation-specific means today

Upcoming RAS hardware features

- ARM IP will implement the following features for LAC/EAC products in 4Q 2015 and 1H 2016:
  - Interconnect that implements data error protection throughout – byte parity on transport and ECC in the system cache
  - Data poisoning on 64-bit data – interconnect, memory controller, caches, CPUs
  - Memory controller Reed-Solomon advanced error detection and correction per cache line that can correct 4-bit memory chip failures
  - ARM v8.2 RAS error reporting – CPUs, interconnect, memory controller, GIC

PCIe RAS

- PCIe is important to the system reliability and serviceability
- ARM does not offer PCIe root port IP
- The PCIe root port should be selected for the desired RAS features

We believe that the following PCIe RAS features are desirable for a server product to be competitive:

- Advanced error reporting (AER)
- End-to-end CRC (ECRC)
- Downstream port containment and enhanced DPC (DPC and eDPC)
- Hot plug

RAS in Subsystems

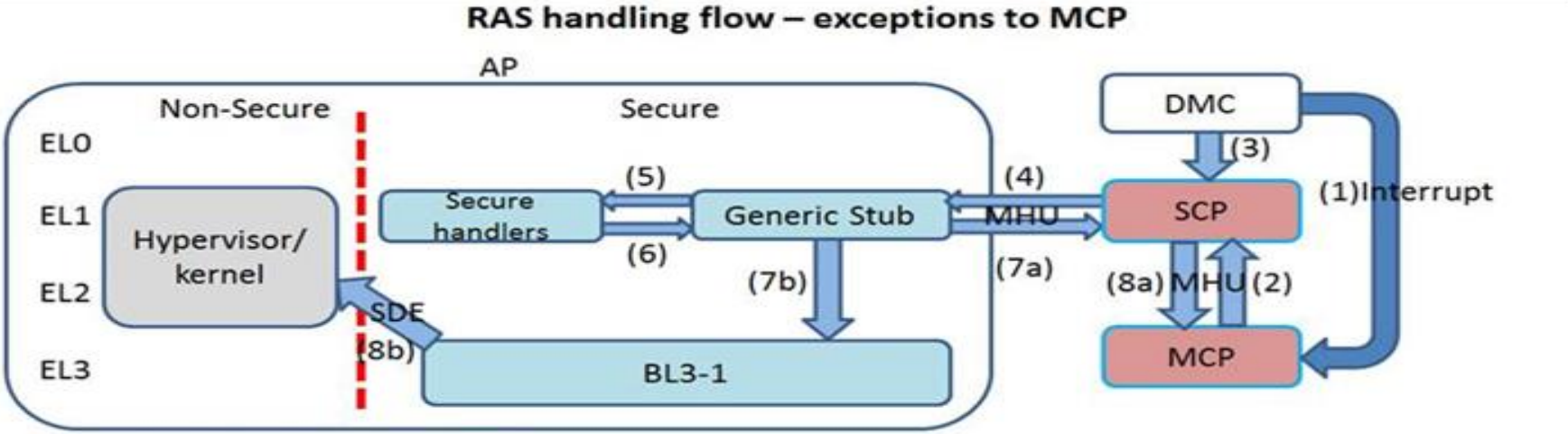
System-Level Standards

- IPMI 2.0/DCMI 1.5 reference code
  - Power on/off/status
  - FRU Data
  - System Event Log (SEL)
  - SOC Power/Thermal Data to BMC
  - In-band communications messages via SCP Mailbox (MHU)
  - Future out-of-band interface standard for BMC<->MCP communications in development
- Alert events according to ASF v2.0
- Firmware update according to PIC HPM.1
- SBSA Level 3
- ARM RAS HW Standards for v8.0 and later

ARM Subsystems RAS support

- ARM subsystems will use ARM IP components supporting RAS features from previous slides
  - Ashbrook is first subsystem with many of the RAS features
- In addition, will have firmware support for RAS that coordinates error logging and handling between main processors, embedded system controllers (SCP and MCP) and an external BMC.
  - Primary function of SCP is power control
  - Primary function of MCP is management interface
  - Primary function of BMC is platform management
  - RAS firmware on the main processors (application processors) handles RAS interrupts and controls software recovery actions
- See the S/W slides next.

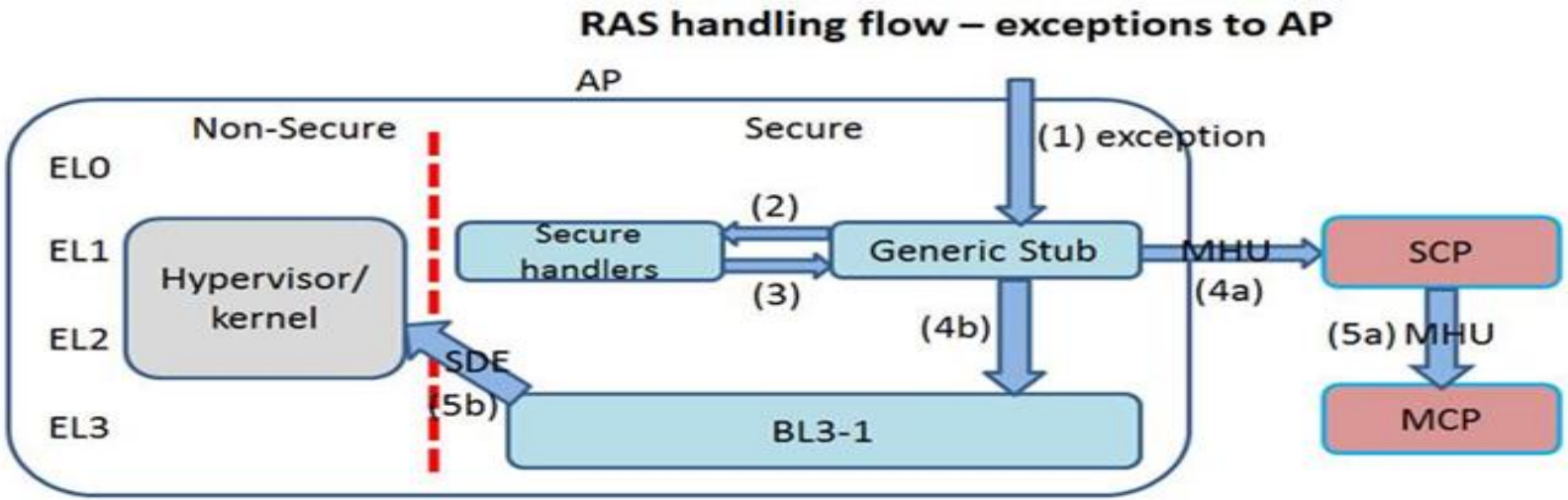
MCP-Directed RAS Flow



(1) DMC interrupt comes to MCP
(2) MCP requests SCP to handle DMC event
(3) SCP reads relative DMC registers
(4) SCP reports DMC event to Generic Stub with DMC registers
(5) Generic Stub calls secure handlers (chain)

(6) Secure handlers create the CPER for this error and return back to Generic Stub
According to policy from handler, Generic stub may route the CPER to :
(7a)SCP, then SCP sends the CPER to MCP (8a)
(7b)Hypervisor/kernel using SDE interface(8b)

ARM TF-Directed RAS Flow

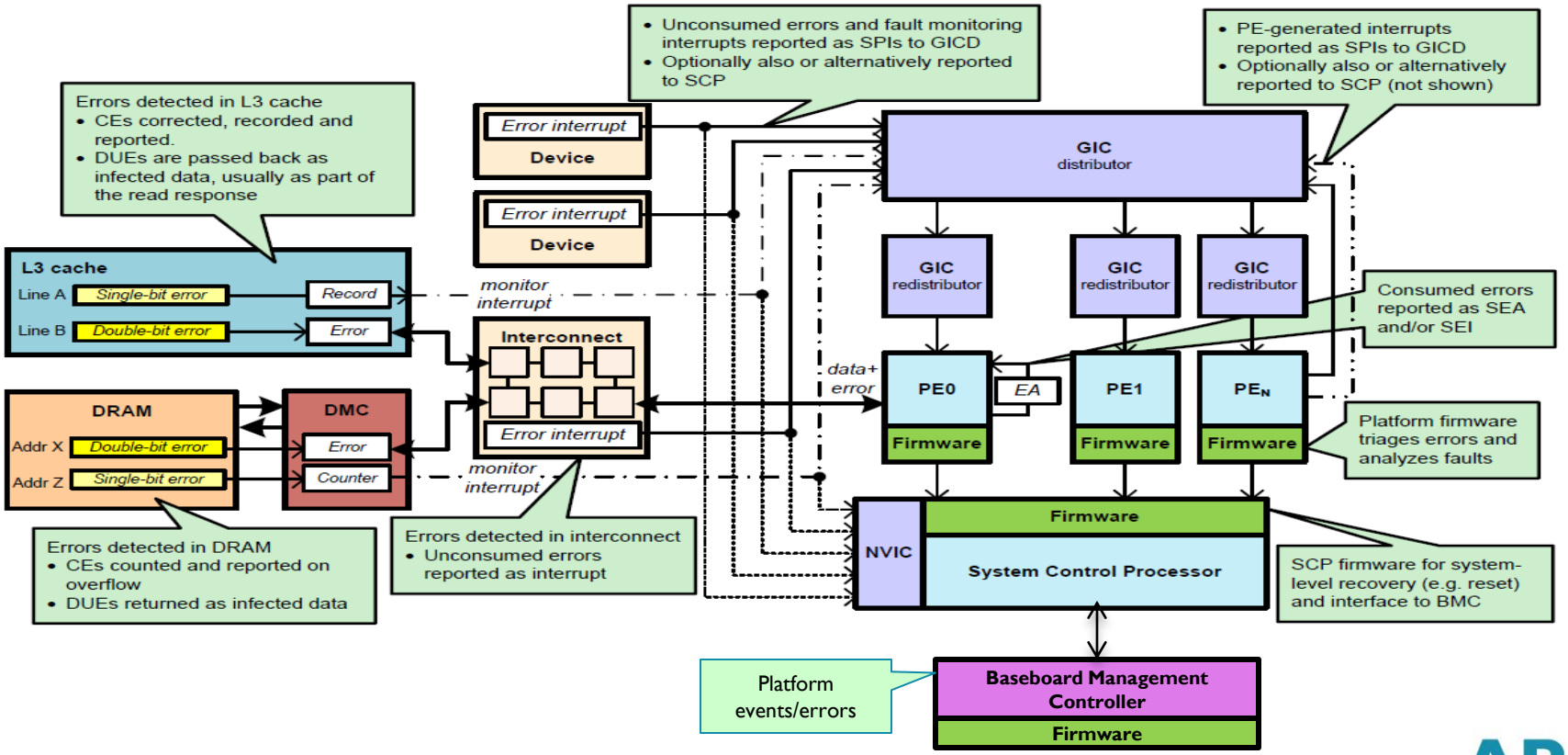


(1) Exception comes to AP (Generic Stub)
(2) Generic Stub calls secure handlers (chain)
(3) Secure handlers create the CPER for this error and return back to Generic Stub

According to policy from handler, Generic stub may route the CPER to :
(4a)SCP, then SCP sends the CPER to MCP (5a)
(4b)Hypervisor/kernel using SDE interface(5b)

Software RAS

System View of RAS



ARM

SW RAS – What ARM is working on

- Working with partners to build a Minimum ARM Enterprise SW RAS Standard that will complement the ARM HW RAS spec that is in review
- Delegated exceptions for communicating asynchronous error events to OS and hypervisor
- Reviewing proposals for change to standards and SW (ACPI, UEFI, ARM Trusted FW, SBBR, hypervisors, perf) and providing feedback
- Working with Linaro to review and update LEG cards regarding RAS

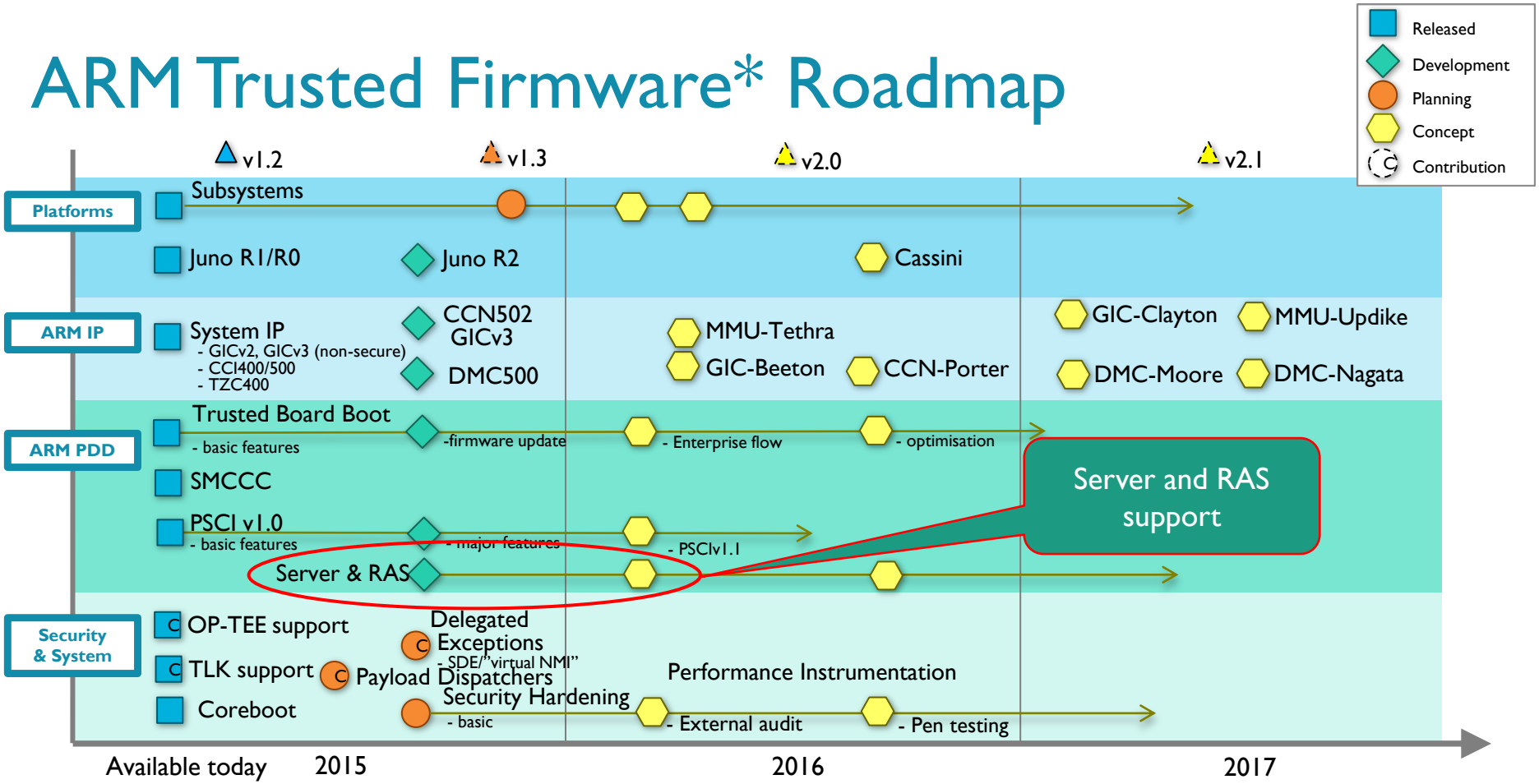
NOTE

ARM also plans to work with one or more lead partners to ensure the 'Minimum SW RAS Standard' has an open source implementation available tied to physical hardware

ARM Trusted Firmware Support for RAS

- ARM Trusted Firmware (TF) support for RAS:
  - Receive fault reporting and error recovery interrupts
  - Collect information into common platform error record (CPER)
  - Route to handlers in TF, silicon vendor and OEM code
  - Send logging information to BMC
- First TF implementations in 2016

ARM Trusted Firmware* Roadmap



* http://github.com/ARM-software/arm-trusted-firmware

What Work is Needed

- Standard Error Reporting Mechanisms
  - UEFI Common Platform Error Record (CPER) needs to be updated
  - UEFI Run-time services support defined
  - ACPI Platform Error Interface (APEI) needs to be updated
  - Synchronous Error Aborts (SEA) for contained consumed errors
  - Asynchronous external aborts (AEA) for (uncontained consumed errors)
    - Needs to report the level of contamination
  - Interrupts for all other errors need to be defined
  - Common Record format for RC/EL3 to BMC fault reporting
- Reference implementation of SDE (Software Delegated Exception) in EL3 FW
- Routing of RAS Interrupts
  - Initial Routing of interrupts to EL3 where a small amount of monitor code will redirect the interrupt to either Secure EL1 or EL2/EL1 for processing
- Controls for Routing Synchronous External Aborts to EL2
  - Today's systems require EL1 code to pass a synchronous external abort up to FW, which isn't truly 'firmware first'
- Secure Virtual SEI for EL3 to send an SEI to lower EL's
  - High Priority interrupt needed
  - Registers to define ESR on taking Secure or Non-secure virtual SEIs
  - Secure EL1 Partitions?
- Changes to Perf for error/fault management within OS
- Hypervisor fault handling defined (Type 1 and Type 2 hypervisors have different requirements)

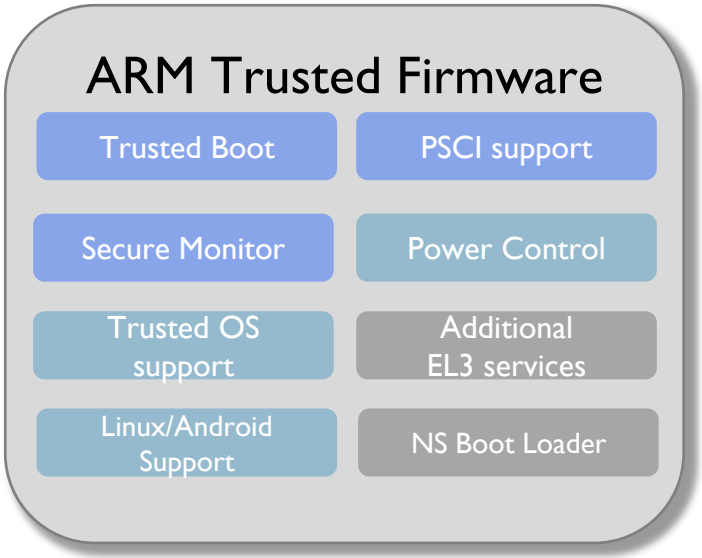
Backup

ARM Trusted Firmware

ARM Trusted Firmware

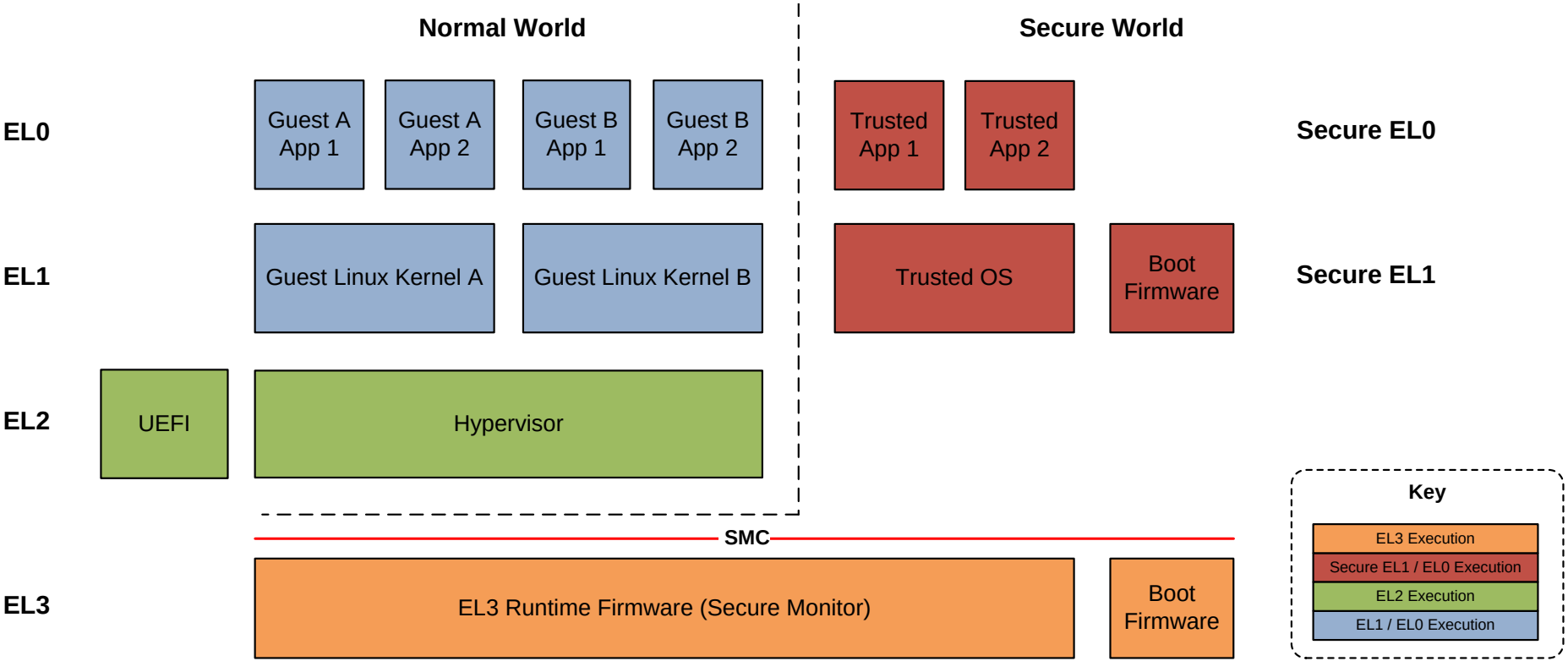
- 64bit Boot and Runtime Firmware for ARMv8-A
  - Mostly executes within the EL3 exception level

- Trusted boot sequence
- Runtime CPU power interfaces (PSCI), used by Linux
- Flexible Secure OS integration
- Extensible framework for SiP and OEM



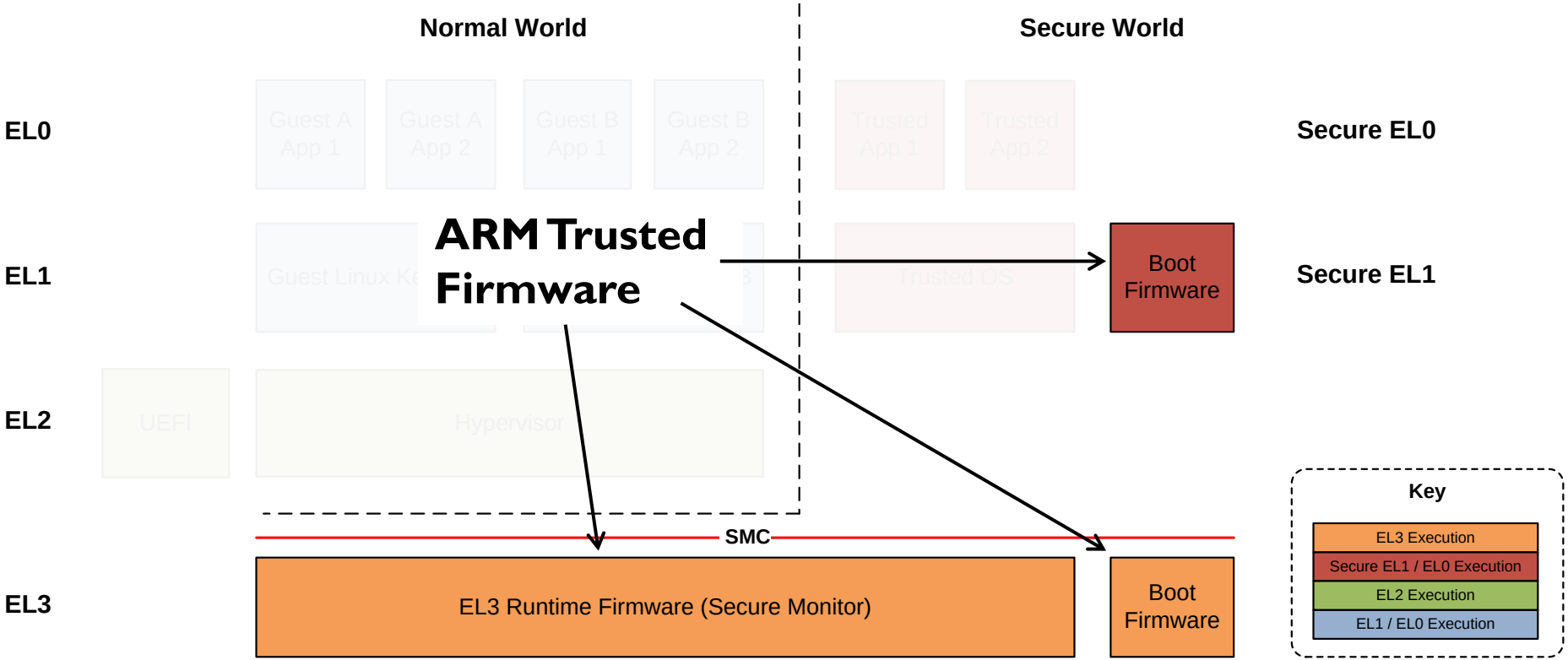
- Open Source Project with BSD license and standard CLA
  - Managed by ARM, hosted at GitHub
  - https://github.com/ARM-software/arm-trusted-firmware
  - http://www.arm.com/community/open-source-contributing.php

What is ARM Trusted Firmware?

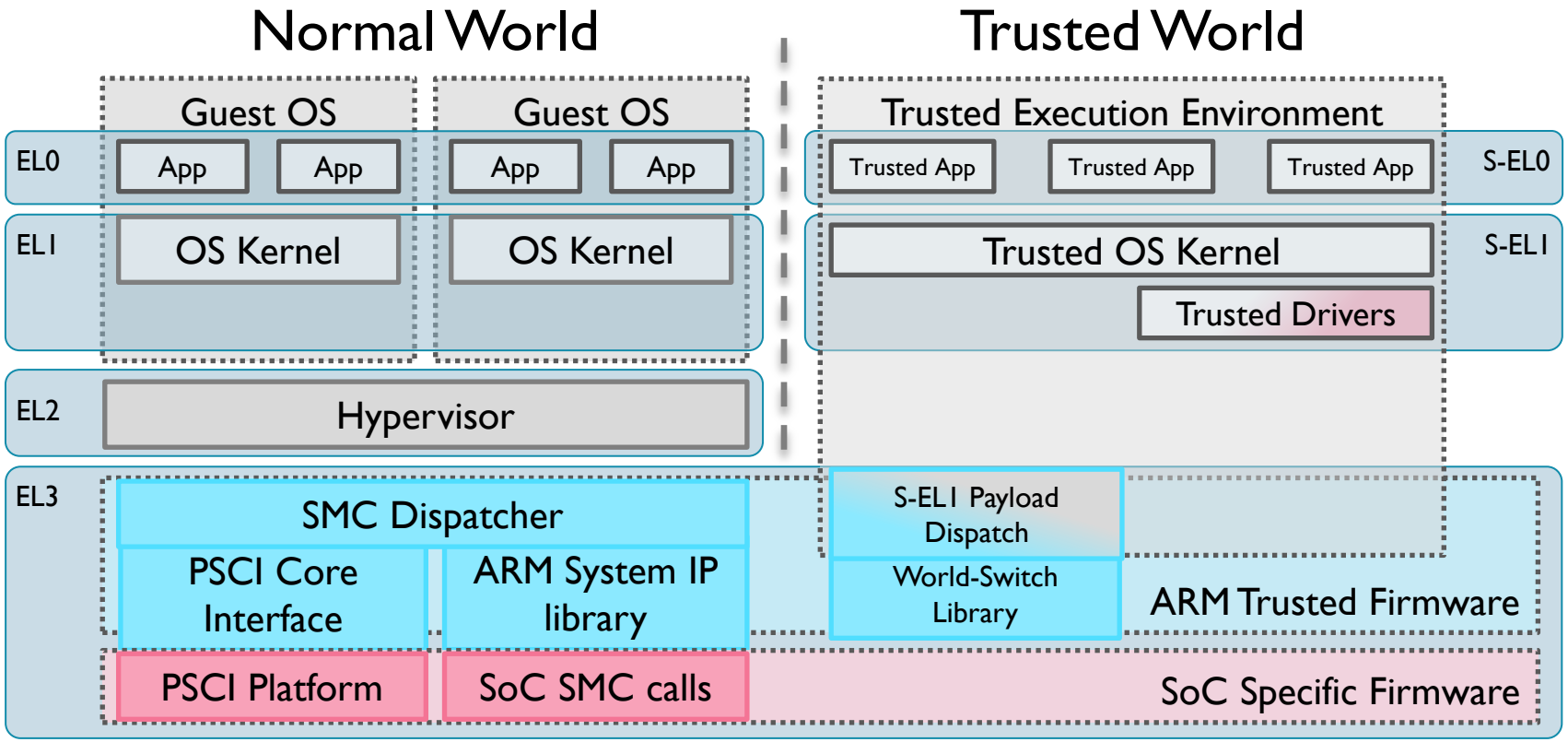


ARM

What is ARM Trusted Firmware?



ARM Trusted Firmware Architectural Overview

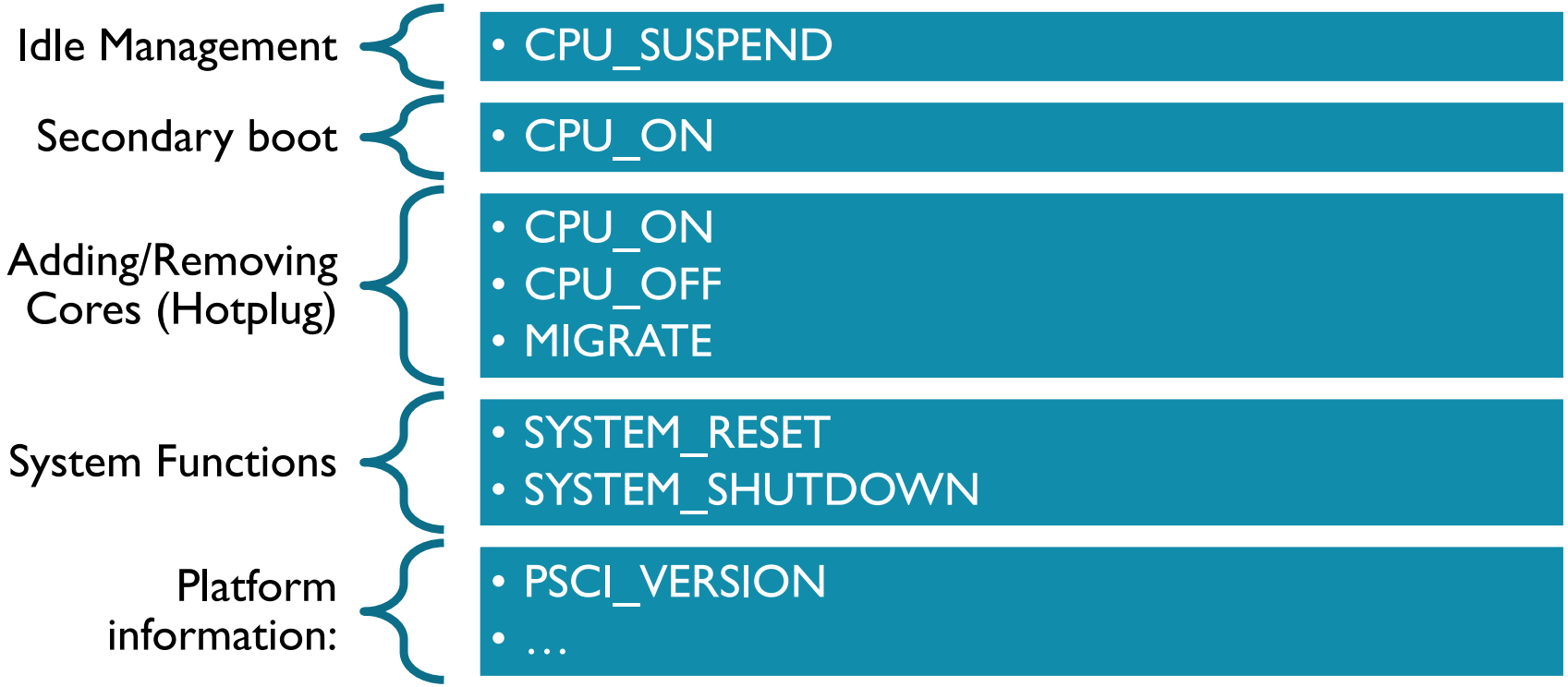


ARM

Power State Coordination Interface (PSCI)

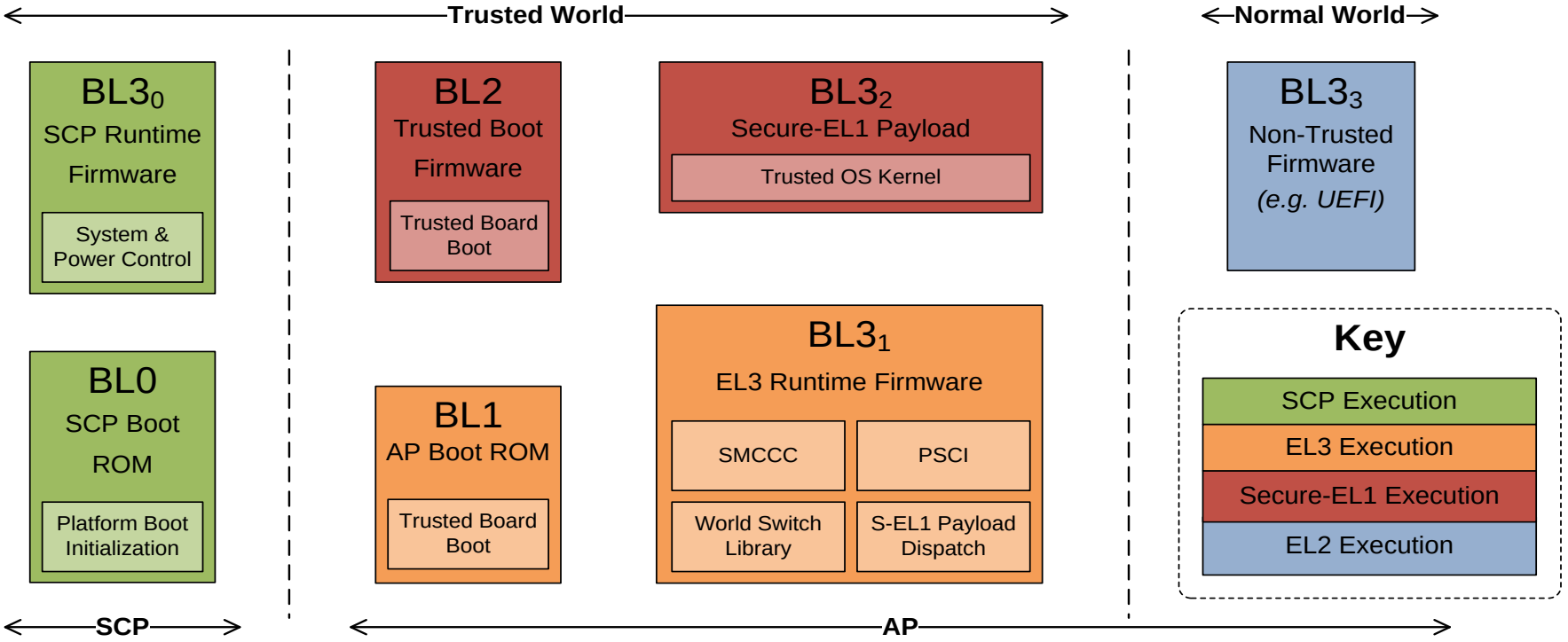
- ARM standardised interface for OS power management
- Supports virtualisation and power management of Trusted OS
- Main method for power control in Linux AArch64 kernel
- Draft specification (v0.2) available today in the ARM Information Center: http://infocenter.arm.com/help/topic/com.arm.doc.den0022b/index.html
- v1.0 planned for Q4

PCSI Functions



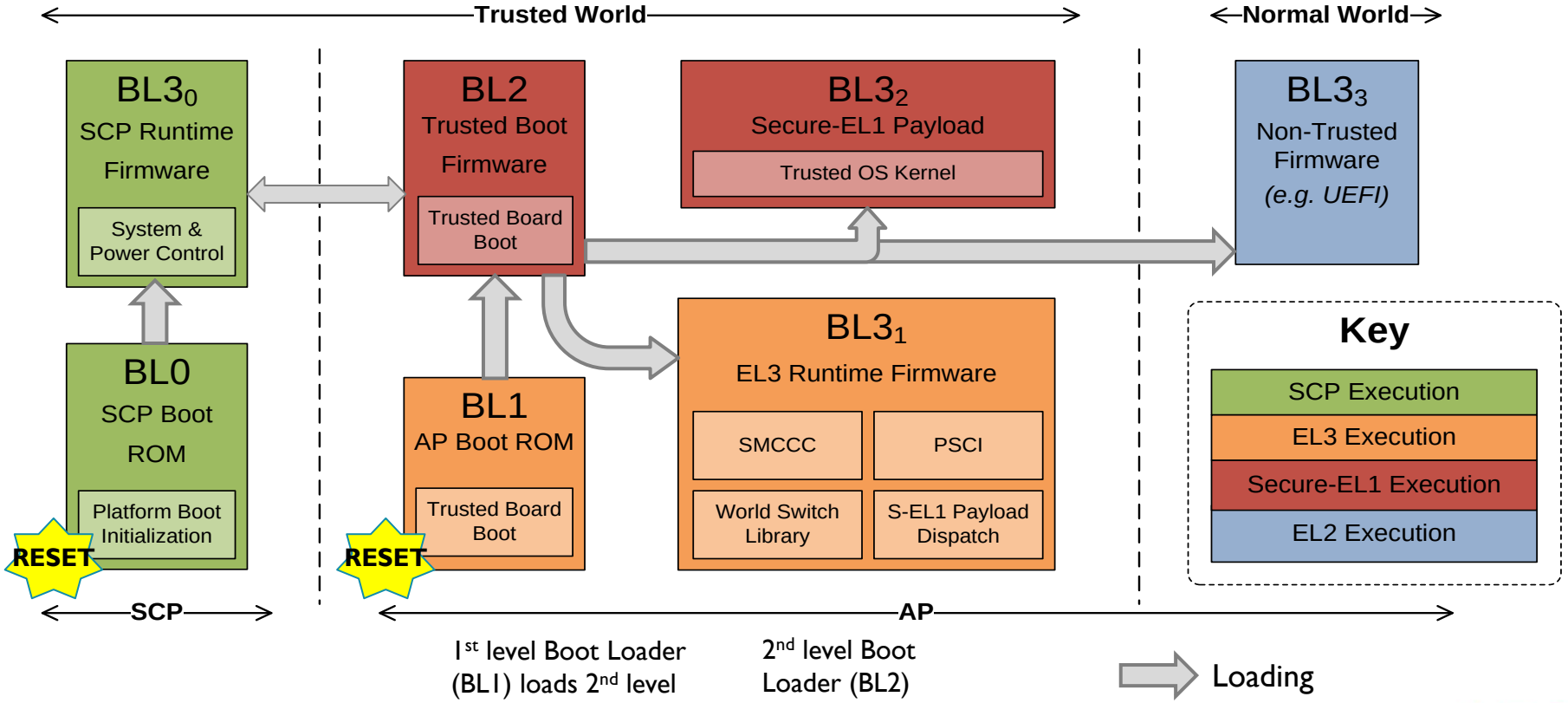
43

Trusted Boot Image Terminology



ARM

Trusted Boot Image Loading Flow



Firmware Image Packages (FIPs)

- All non-ROM firmware images stored as a FIP in Non-Volatile Memory (NVM)
  - For now: BL2, $BL3_0$, $BL3_1$, $BL3_2$ and $BL3_3$
  - Later: Recovery firmware images and certificates (for TBB)

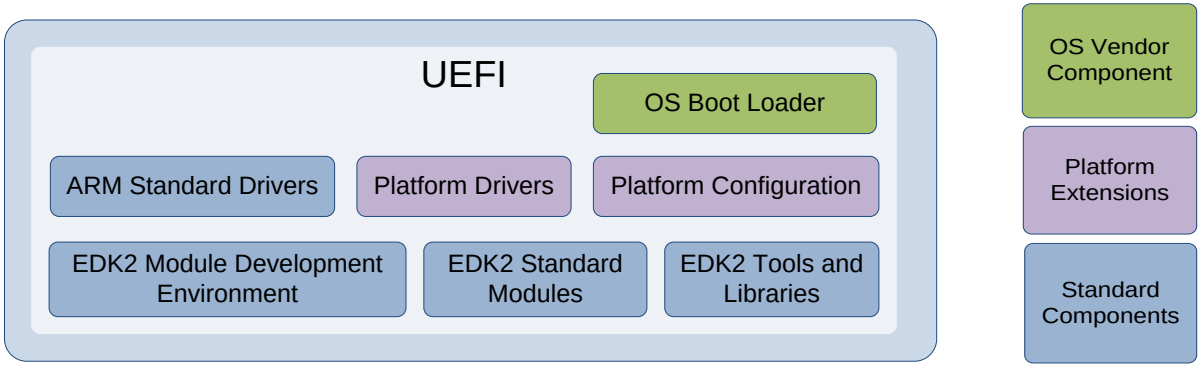
- FIP begins with a Table of Contents (ToC)
  - A header plus an entry per image/certificate

- BL1 and BL2 read FIP contents to load images/certificates into Trusted RAM

- A tool (and makefile support) is provided for FIP creation/enquiry

UEFI

- UEFI specs owned by UEFI forum
  - AArch32 supported since v2.3. AArch64 supported since v2.4.
- EDK2 is open source implementation of UEFI forum specs
  - Large developer community https://github.com/tianocore/edk2.git
  - ARM maintains the ARM architecture and contributes to the overall infrastructure. Linaro maintains a platforms tree and makes associated monthly releases [to the tree]
- Modular nature of EDK2 makes it easy to re-use and customize for each platform
  - Minimizes platform specific porting

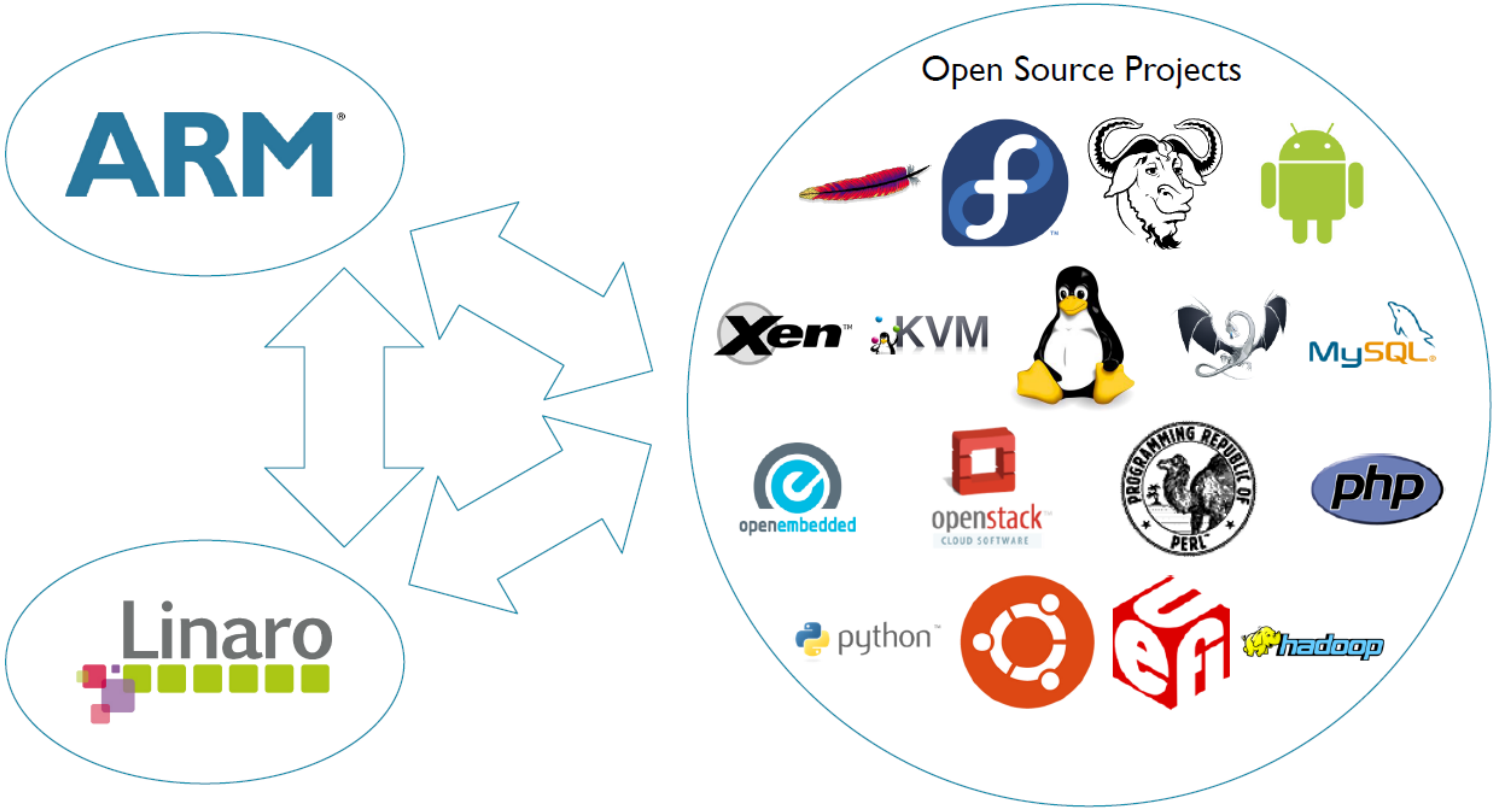


SW Workloads

ARMv8-A Infrastructure Ecosystem Building Momentum

ARM

Engaging the Open Source community around ARMv8-A



Segments

- Mobile
- Home
- Server
- Network
- …

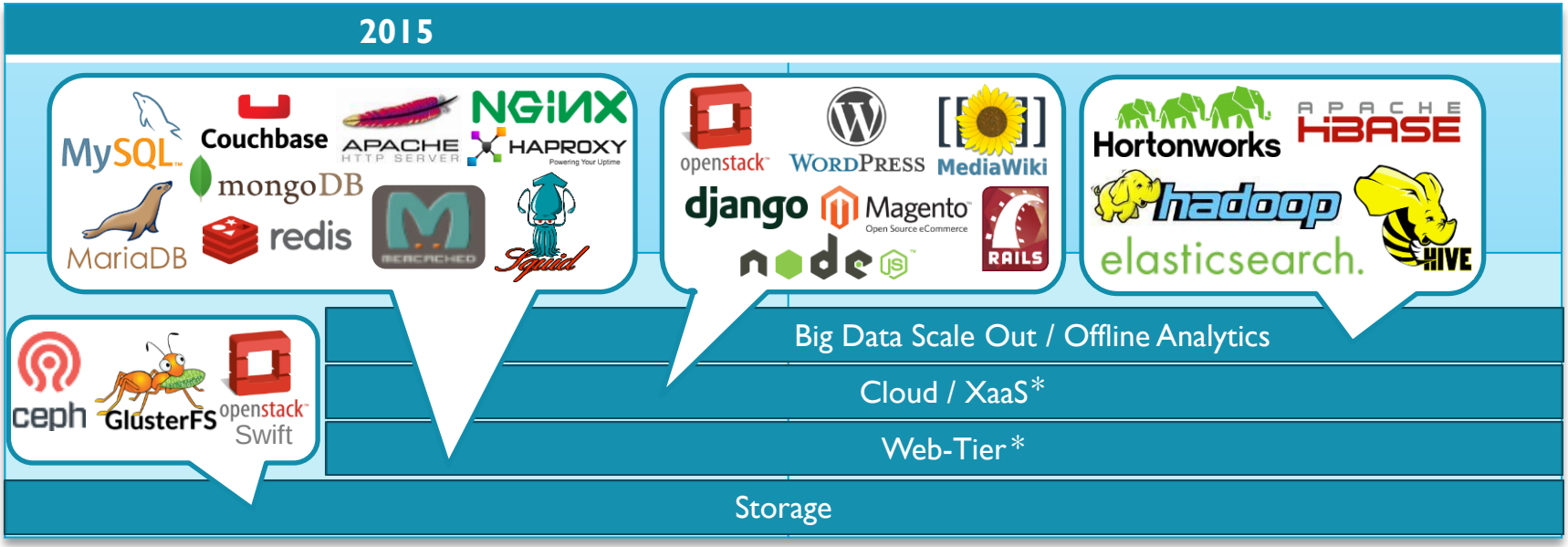
ARM

Web-scale Workloads – Largely Built On Open Source

ARM

Immediate Target Workloads

Which ones matter to you?



* Languages: PHP, Ruby, Java, Python, Ruby, Perl, V8 Javascript, Erlang, Go, Scala etc…

ARM