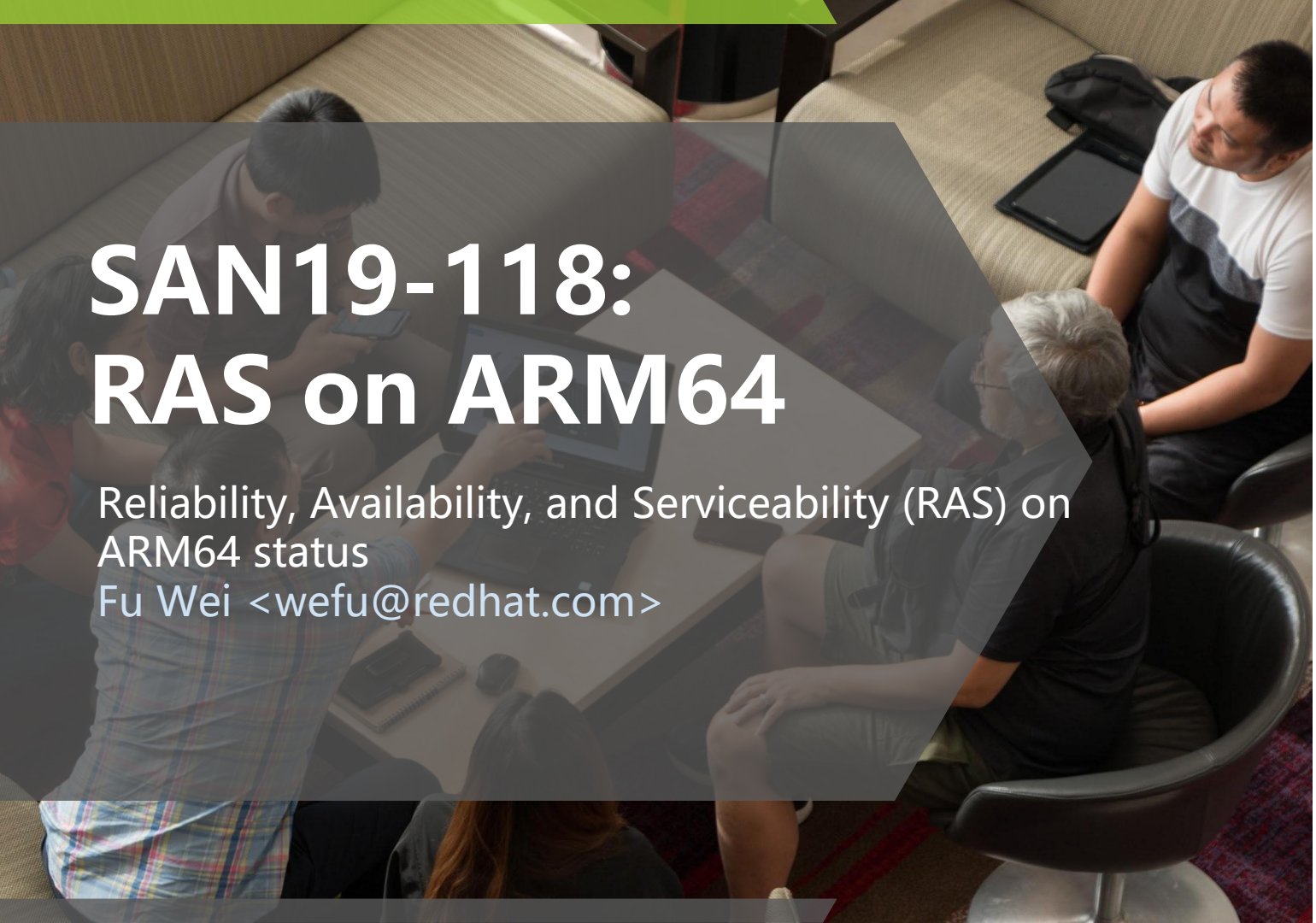


SAN19-118: RAS on ARM64

Reliability, Availability, and Serviceability (RAS) on ARM64 status

Fu Wei <wefu@redhat.com>

Linaro connect
San Diego 2019

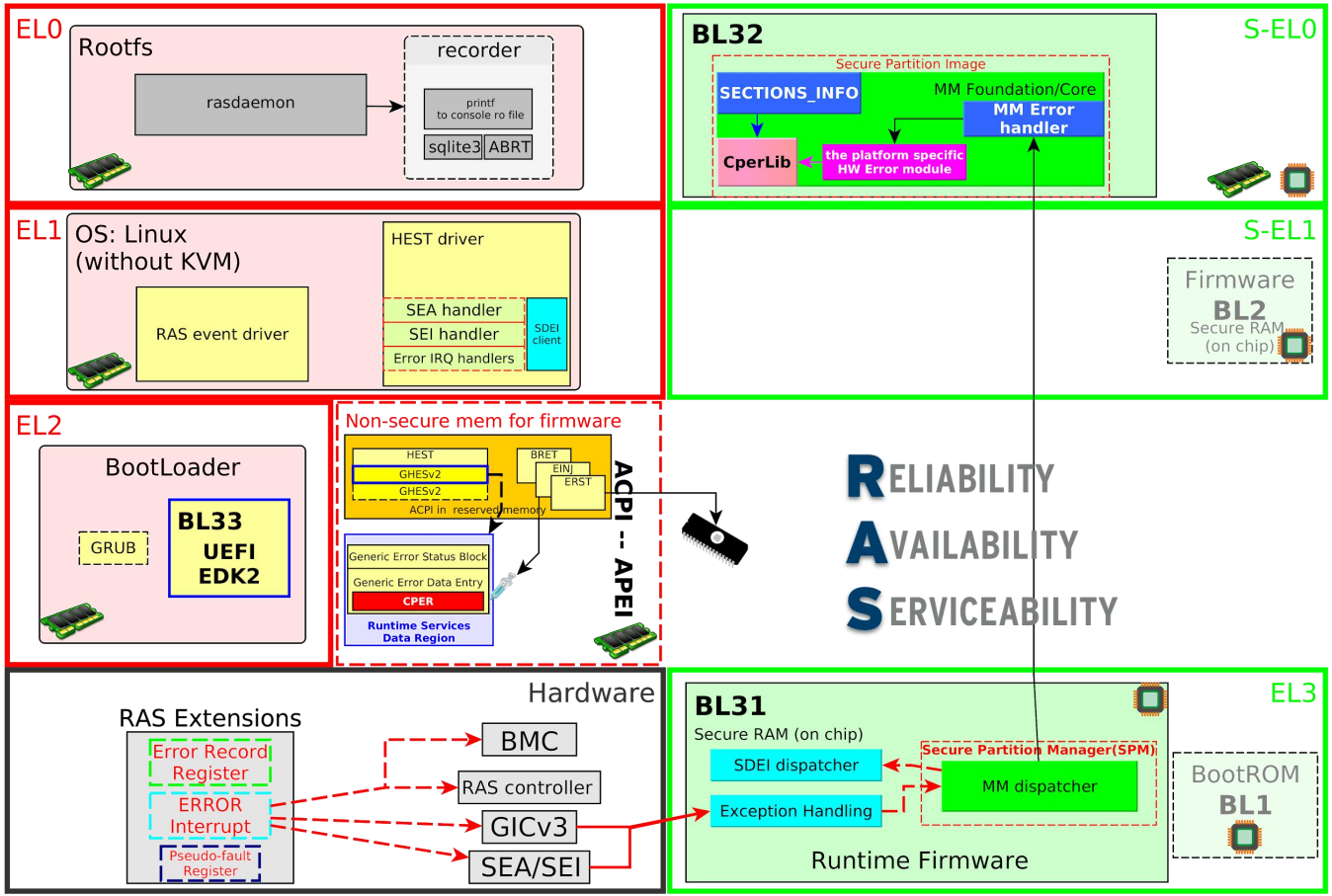
AGENDA

1. Brief introduction
   - Architecture, RAS Extension, APEI, SDEI
2. Prototype overview
   - Firmware First Error Handling
     - Overview
     - APEI protocol (boot time)
     - CperLib (run time)
3. Status & Plans
   - SBSA QEMU & SPM
   - EINJ

Brief introduction

Architecture, RAS Extension, APEI, SDEI

RAS Architecture Overview



Hardware support for RAS

RAS Extension

- **ESB (Error Synchronization Barrier) instructions**
- **RAS Extension registers**
- **Corrupted data poisoning**

CPU

- ARMv8-A architecture (a mandatory extension to ARMv8.2)
- EL2, EL3, or both
- Virtualization extension or Security extensions or both

GICv3

- Interrupt routing modes
- Private and shared interrupts (PPI/SPI)
- Ability to set an interrupt pending event signaling and delegation Interrupt groups/priority

RAS Extension: Gather HW error info for FW

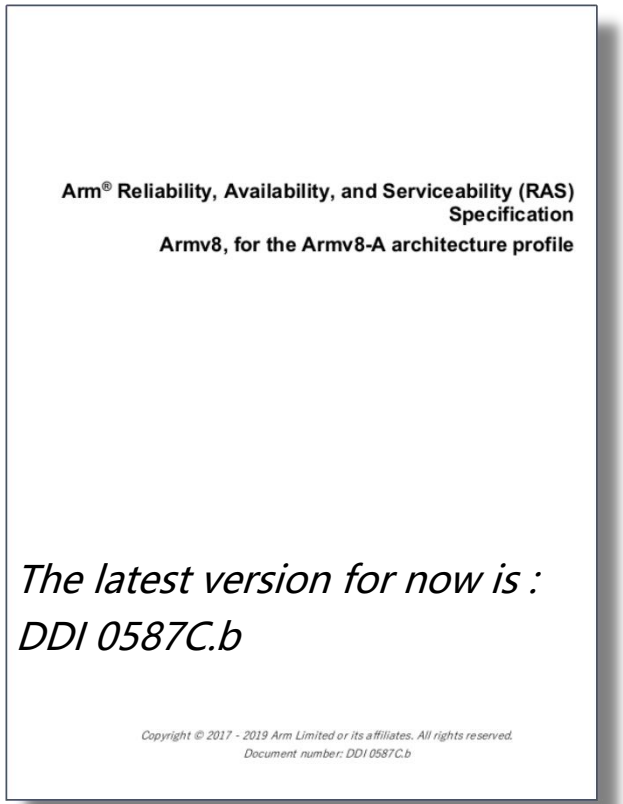
ESB instruction
Help to **locate** Error

RAS Extension registers
- Provide the error **info** to FW
- Control the **Interrupt** by FW

ARMv8-A RAS extensions standardize **the interface** between HW and FW

- a mandatory extension to ARMv8.2， an optional extension to the Armv8.0 and Armv8.1

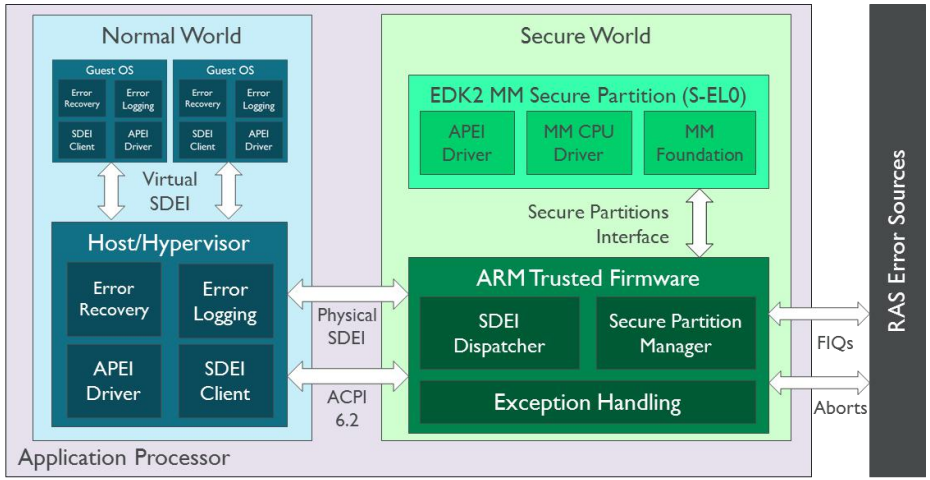
Arm ® Reliability, Availability, and Serviceability (RAS) Specification Documentations



The latest version for now is :
DDI 0587C.b

The latest version for now is :
DDI 0487E.a

Firmware support for RAS: SPM & MM Secure Partition



Firmware First Handling of AP RAS Errors

- **Secure Partition Manager** in **BL31** exports standard ABI to
  - Initialize the partition
  - Delegate SMC requests to the partition
  - **For RAS**, delegate **RAS error handling to a MM Secure Partition of RAS**
- **MM Secure Partition** implements management functions , runs in **S-EL0**
  - Leverages existing firmware code based on EDK2: **StandaloneMm**
  - Minimise code in EL3

Arm SPA Documentations

arm

Arm® Platform Security Architecture
Firmware Framework 1.0
Architecture & Technology Group

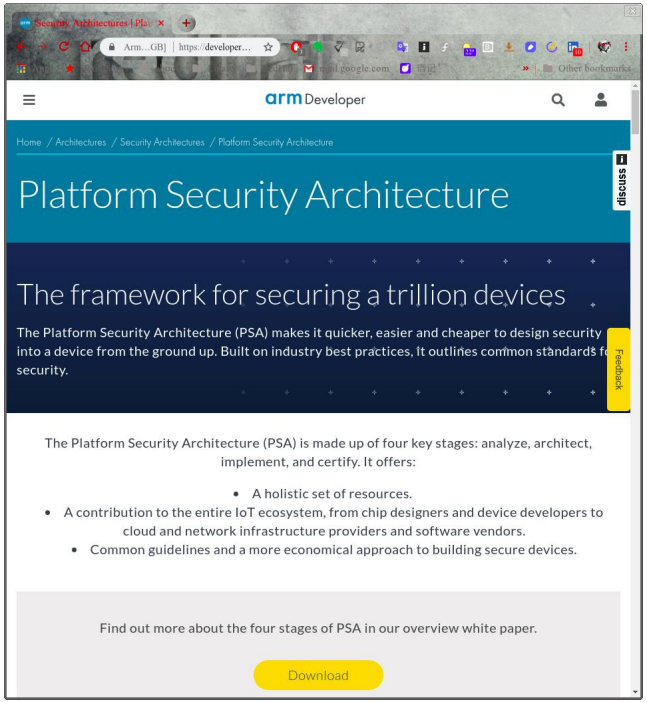
Document number:	DEN 0063
Release Quality:	Release
Issue Number:	0
Confidentiality	Non-Confidential
Date of Issue:	19/06/2019

Abstract

This manual is part of the Arm Platform Security Architecture family of specifications. It defines a standard programming environment and firmware interfaces for implementing and accessing security services within a device's Root of Trust.

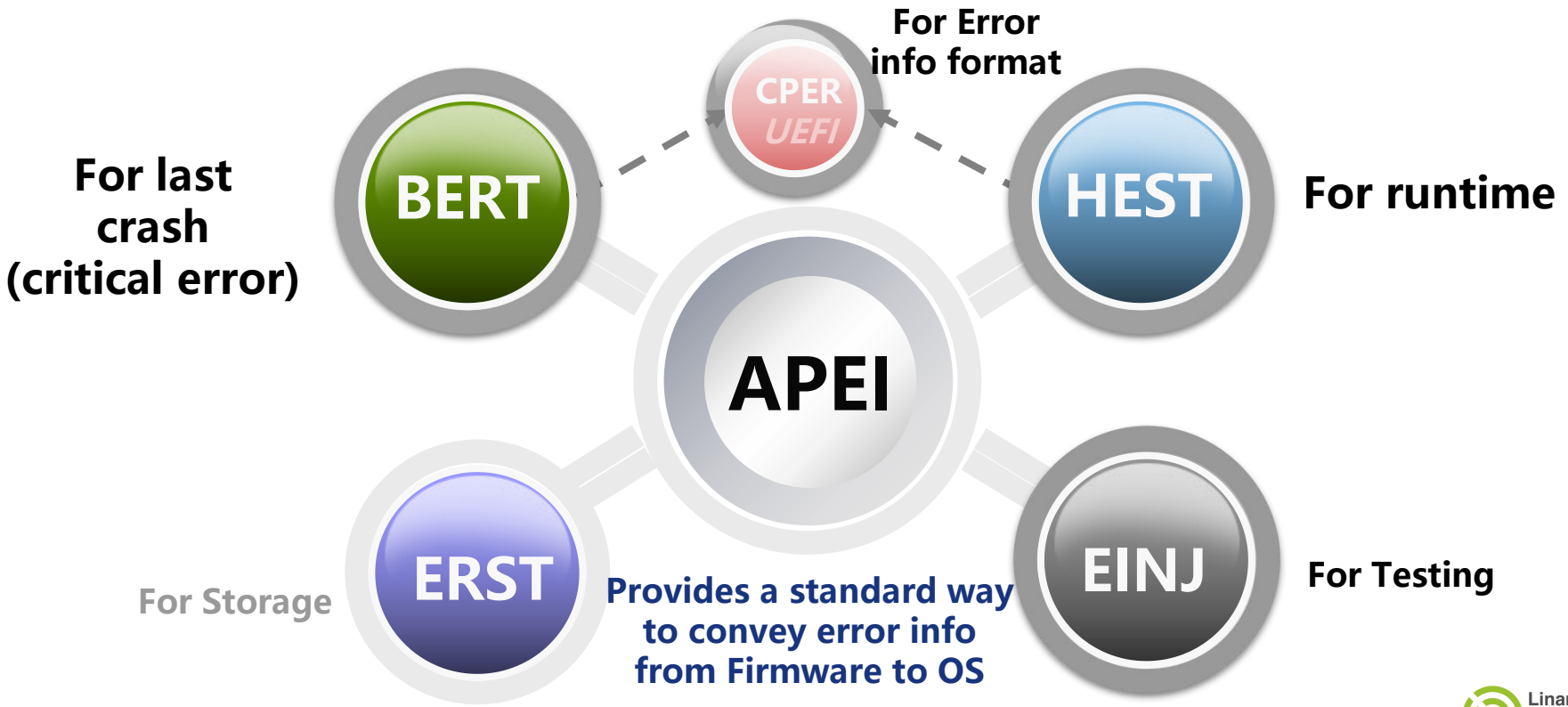
The latest version for now is :

DEN 0063 1.0.0-2

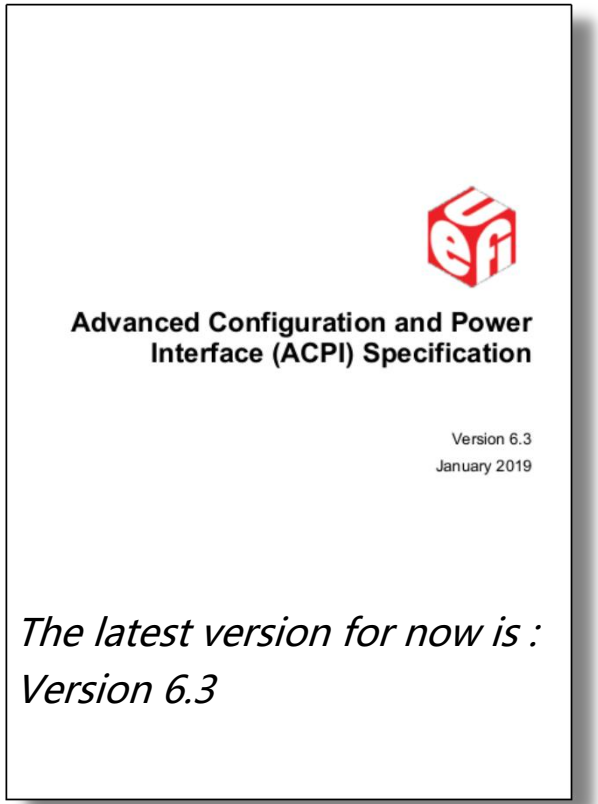


https://developer.arm.com/architectures/security-architectures/platform-security-architecture

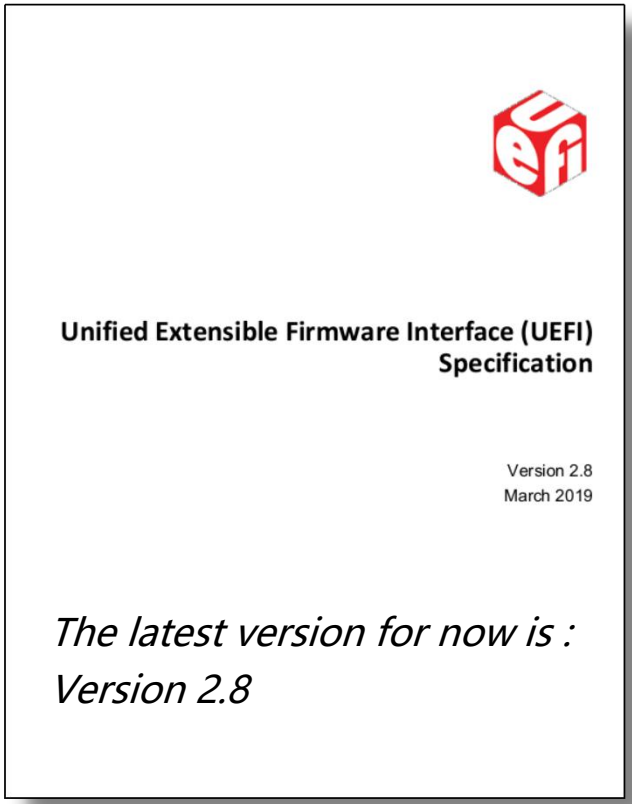
Firmware support for RAS: APEI (ACPI Platform Error Interfaces)



ACPI and UEFI Documentations



The latest version for now is :
Version 6.3



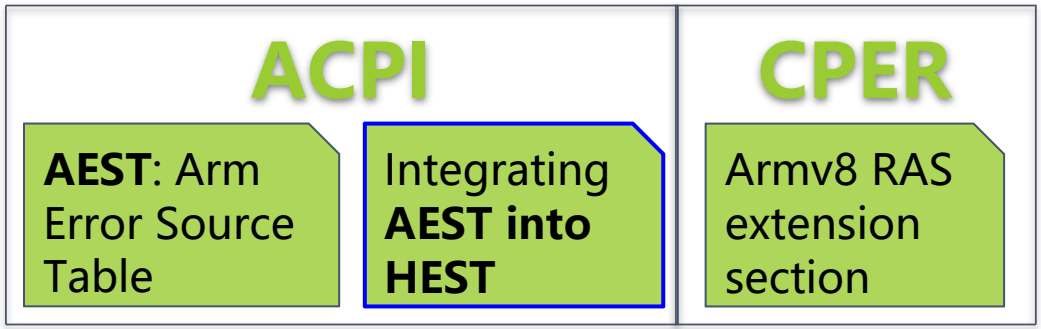
The latest version for now is :
Version 2.8

ACPI for the Armv8 RAS Extensions 1.0(Provisional)

This document describes ACPI extensions that enable **kernel first** RAS handling for systems that employ the Arm RAS extensions.

For PEs, this specification covers:

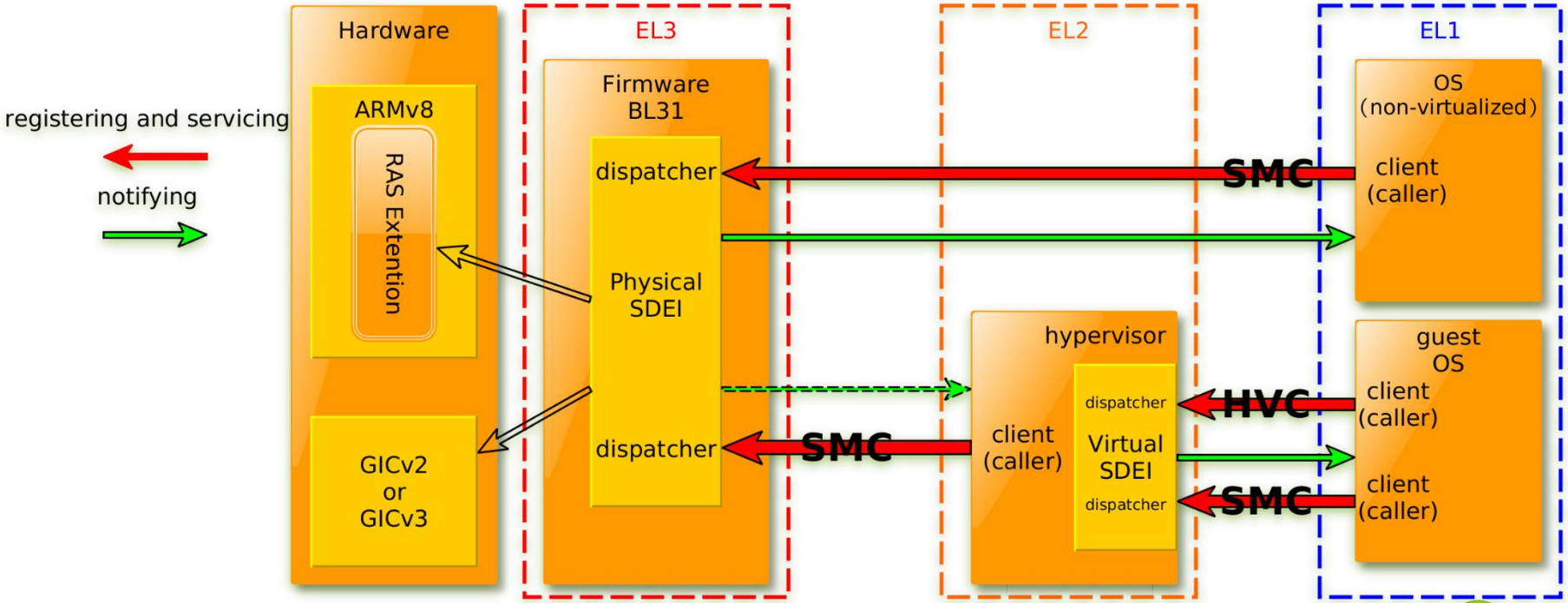
1. **Armv8 & 8.4 RAS extension**
2. **RAS system architecture v1.0 & v1.1**.



Code First

From firmware to OS: SDEI usage in RAS

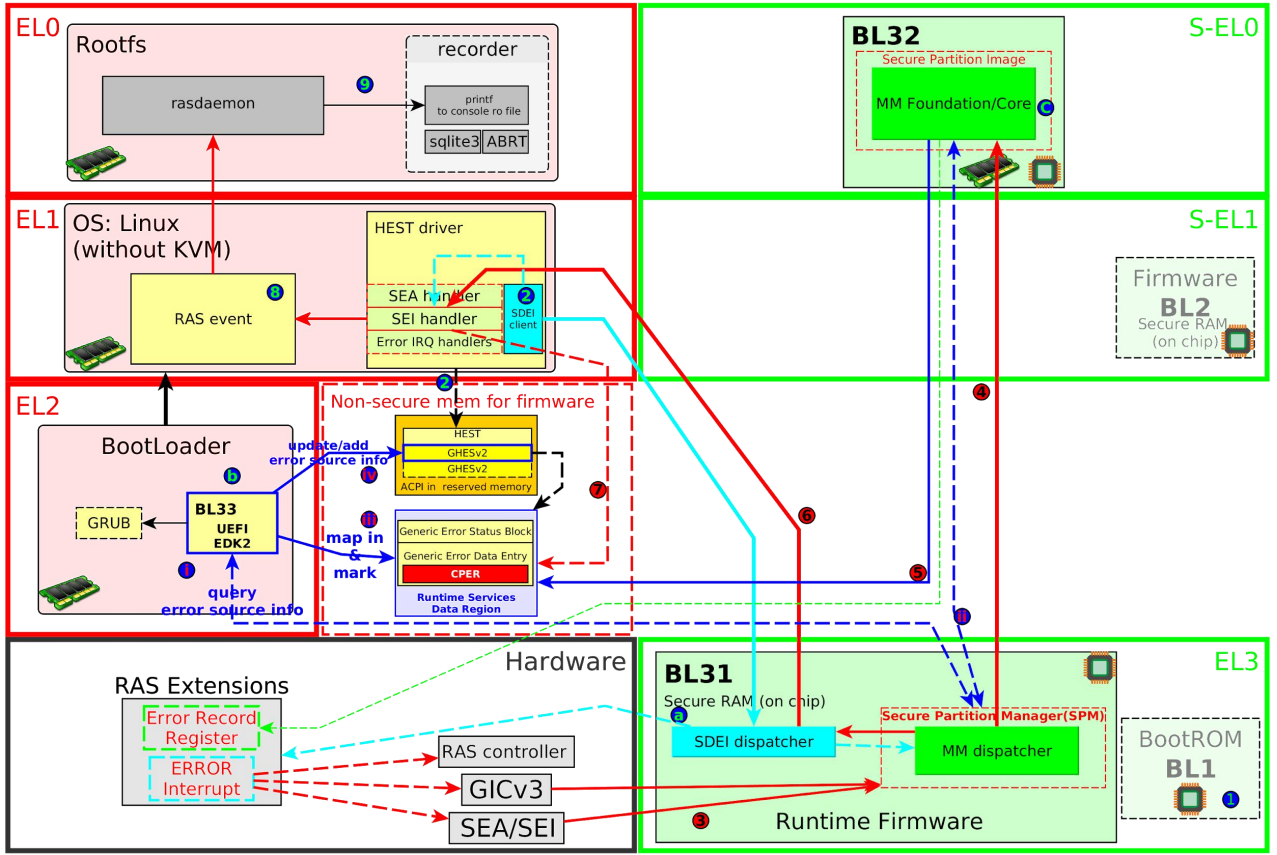
Software Delegated Exception Interface: An interface between FW & OS, for registering, notifying and servicing system events using SMC/HVC. SDEI Specification (ARM DEN0054A)



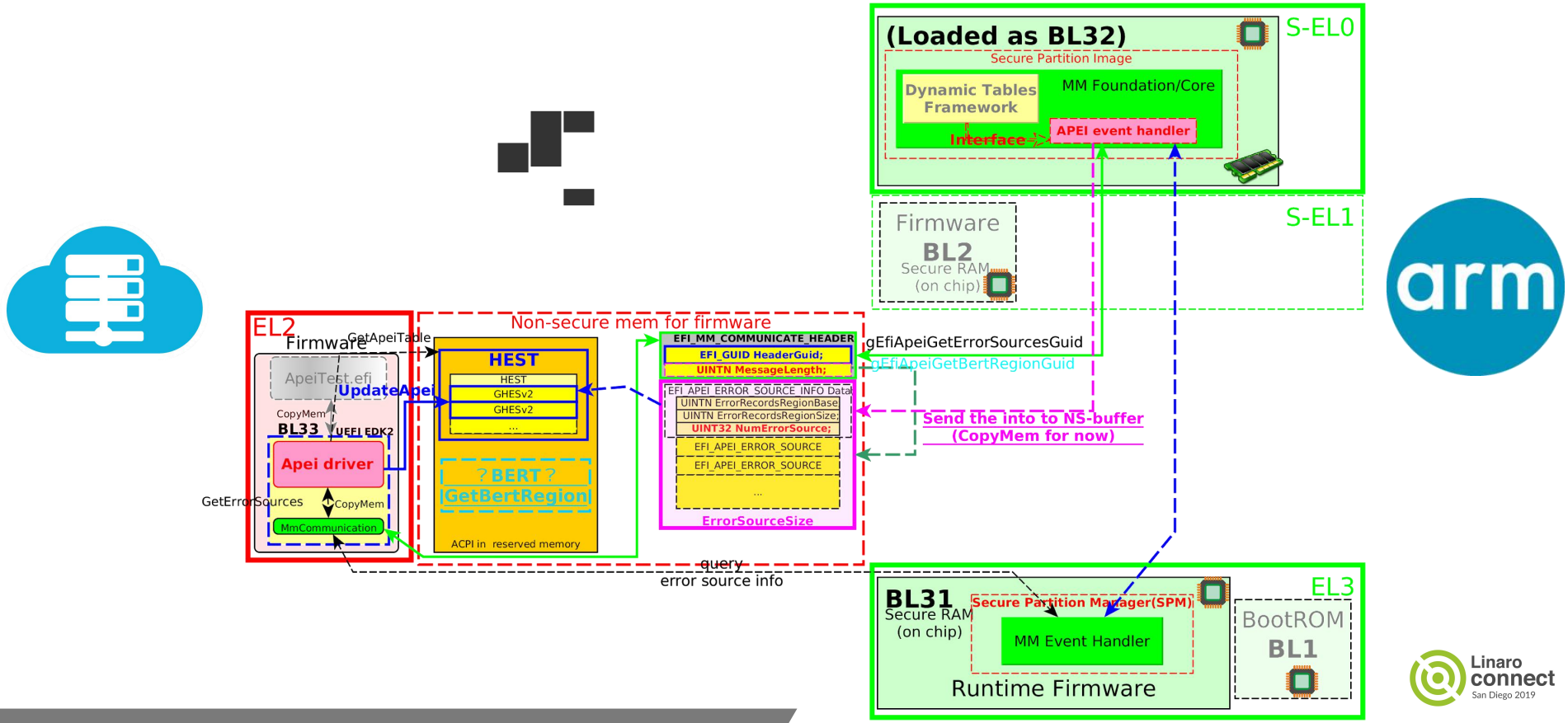
Prototype overview

- Firmware First Error Handling

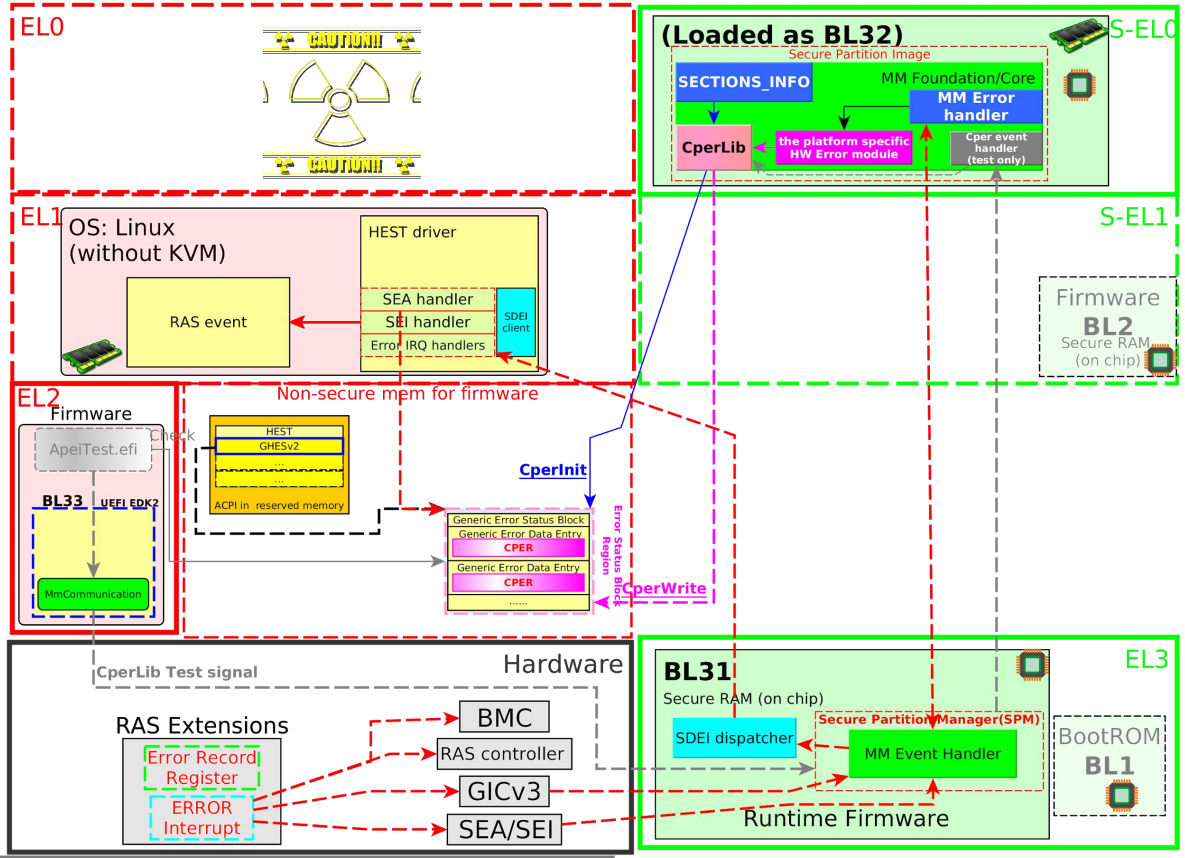
Firmware First Error Handling Overview



APEI protocol: Updating APEI table in boot time

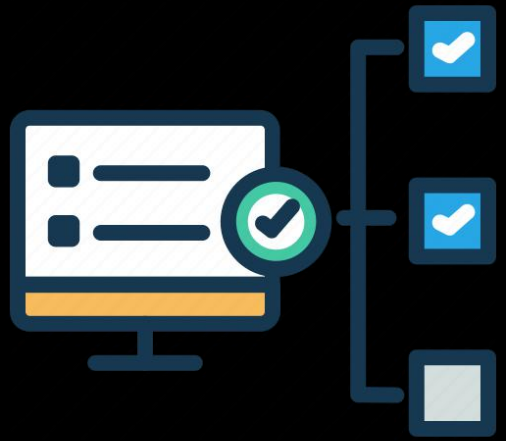


CperLib : Generating the CPER blob in runtime



Status & Plans

SBSA QEMU & Secure Partition
TODO

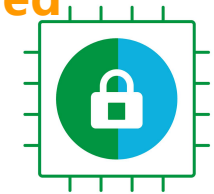


SBSA QEMU & Secure Partition

- SBSA QEMU
  - SBSA QEMU patchset has been **upstreamed**

Great thanks to **Hongbo** and **Radosław**

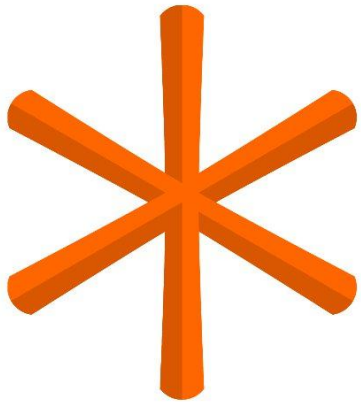
- ARM-TF
  - SBSA QEMU support has been **partly upstreamed**, and the rest of patches are posted.
  - StandaloneMm support(on SBSA QEMU) patches are **debugged**



TrustedFirmware
.org

SBSA QEMU & Secure Partition

- EDK2 and edk2-platform
  - StandaloneMm support patchset has been **upstreamed**
  - StandaloneMm support for SBSA QEMU patches are under **debugging**.
  - **APEI protocol** and **CperLib** patches are under **debugging**.

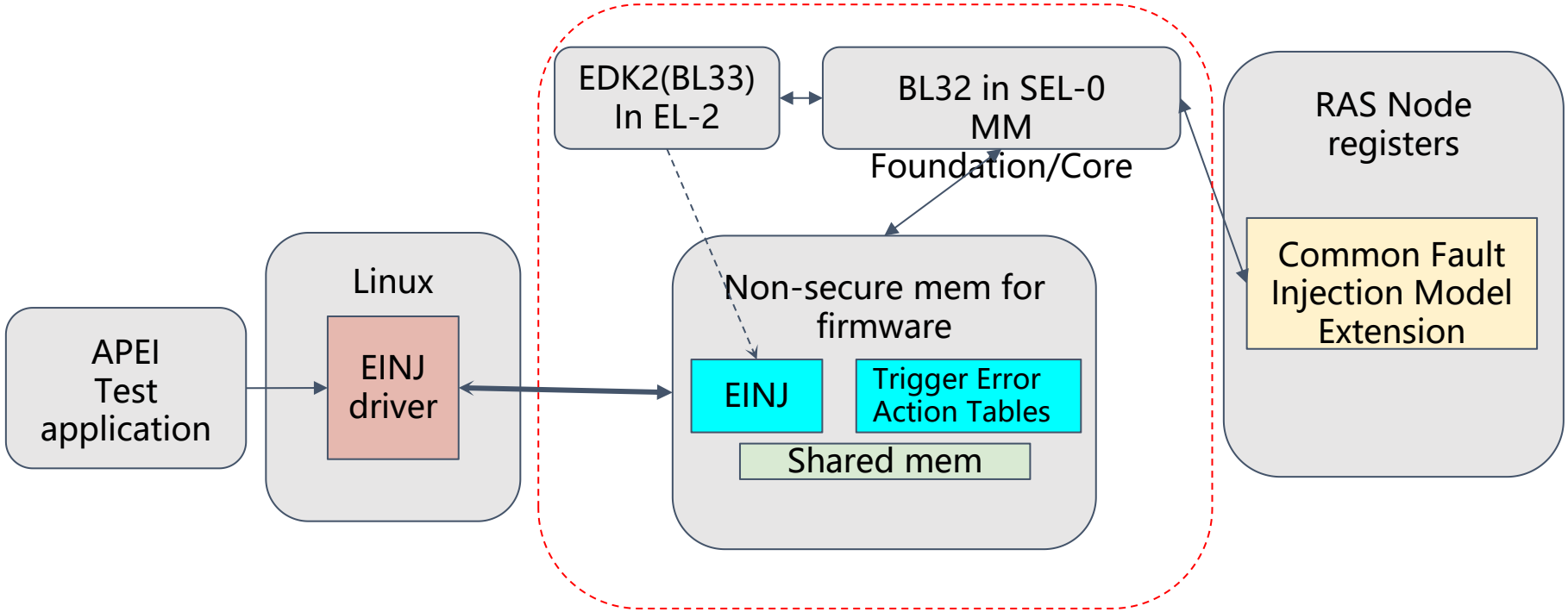


TODO

- Go on upstreaming ARM-TF patchset (target: v2.2)
- Hardware
  - Test on a real hardware (ARMv8.2+)

- Firmware
  - EDK2:
    - Improve and upstream **APEI protocol** and **CperLib** code (target: 201912)
    - EINJ prototype(target: before next Connect)
    - prototype solution of BERT

EINJ: Error Injection with Common Fault Injection Model Extension



Acknowledgments

Achin Gupta
Arvind Chauhan
Charles Garcia-Tobin
Daniil Egranov
Dong Wei
Supreeth Venkatesh
Thomas Abraham

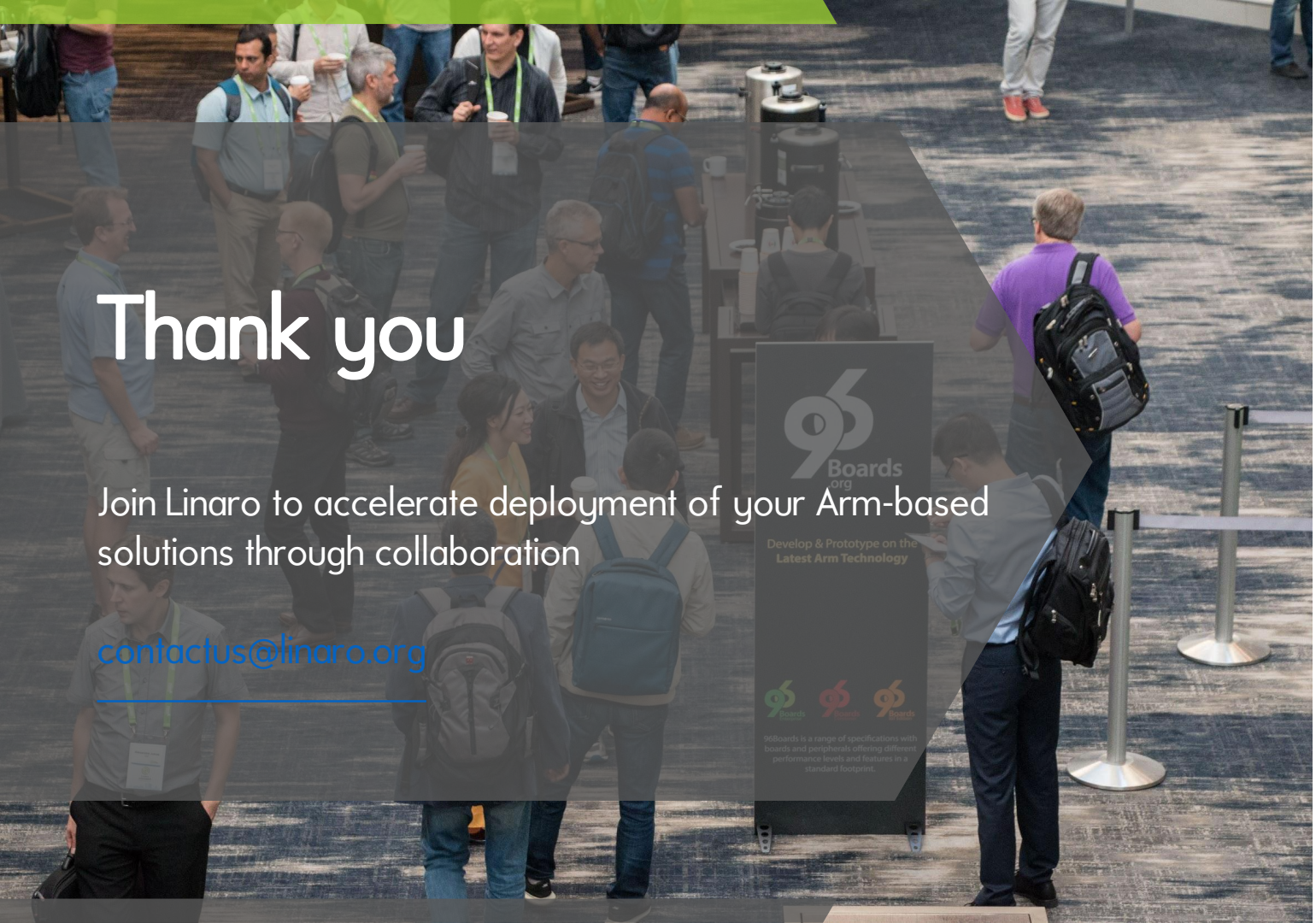
Leif Lindholm
Radosław Biernacki
Tanmay Jagdale
Zhang HongBo

Al Stone
John Feeney
Jon Masters

Thank you

Join Linaro to accelerate deployment of your Arm-based solutions through collaboration

contactus@linaro.org



Linaro connect
San Diego 2019

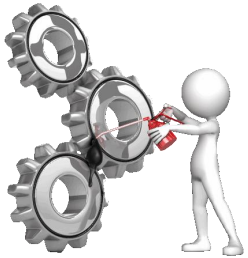
FYI

More detail for RAS on AArch64

What is RAS? -- Definition

Reliability
Continuity, Computation needs be **correct** and **reliable**.

Availability
Readiness, System needs to remain available as long as possible.



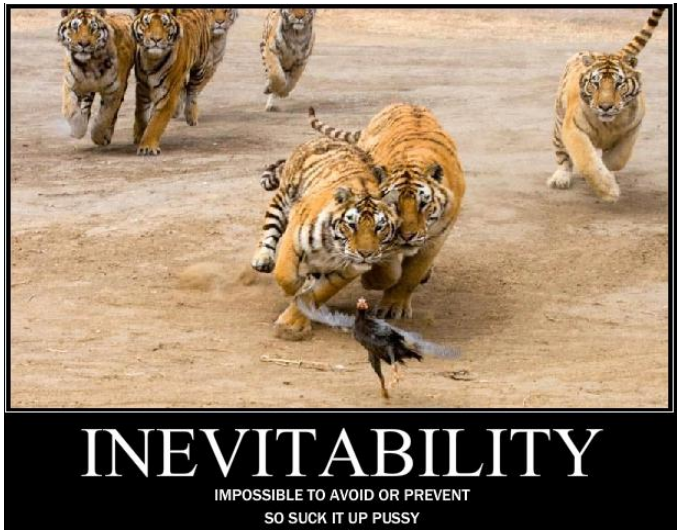
Serviceability
Ability to undergo modifications and repairs, system should provide information to administrator to aid in system servicing.

The RAS architecture primarily cares about
the ERRORs produced from HARDWARE.

Why do we need RAS? -- Reality

Inevitability

Although faults are rare, enterprise systems can be very large. So failures are inevitable.

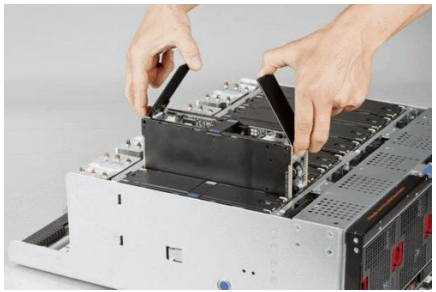


Impacts

Enterprise systems often provide mission-critical services. Any system failure or data corruption impacts the customer’ s business and reputation

Why do we need RAS? -- Importance

So we **inevitably have to maintain** system very well.
For reducing the expense for maintenance, we should :



Replace only **failed parts**

Scheduled maintenance
cheaper than
unscheduled service outages

RELIABILITY
AVAILABILITY
SERVICEABILITY

History

ECC in memory controllers and I/O RAMs
Machine Check Architecture (MCA)

- A mechanism in which the CPU reports hardware errors to the OS
  - model-specific registers (**MSRs**)
    - set up machine checking
    - record detected errors
    - the info they contain is CPU specific
  - Machine Check Exception (**MCE**)
    - signals the detection of an uncorrected machine-check error
    - handler collect information about error from MSRs
  - Utility: mcelog

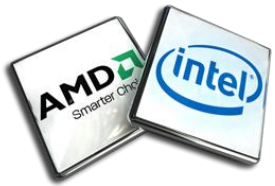
PCI-E: Advanced Error Reporting (AER)

Linux kernel

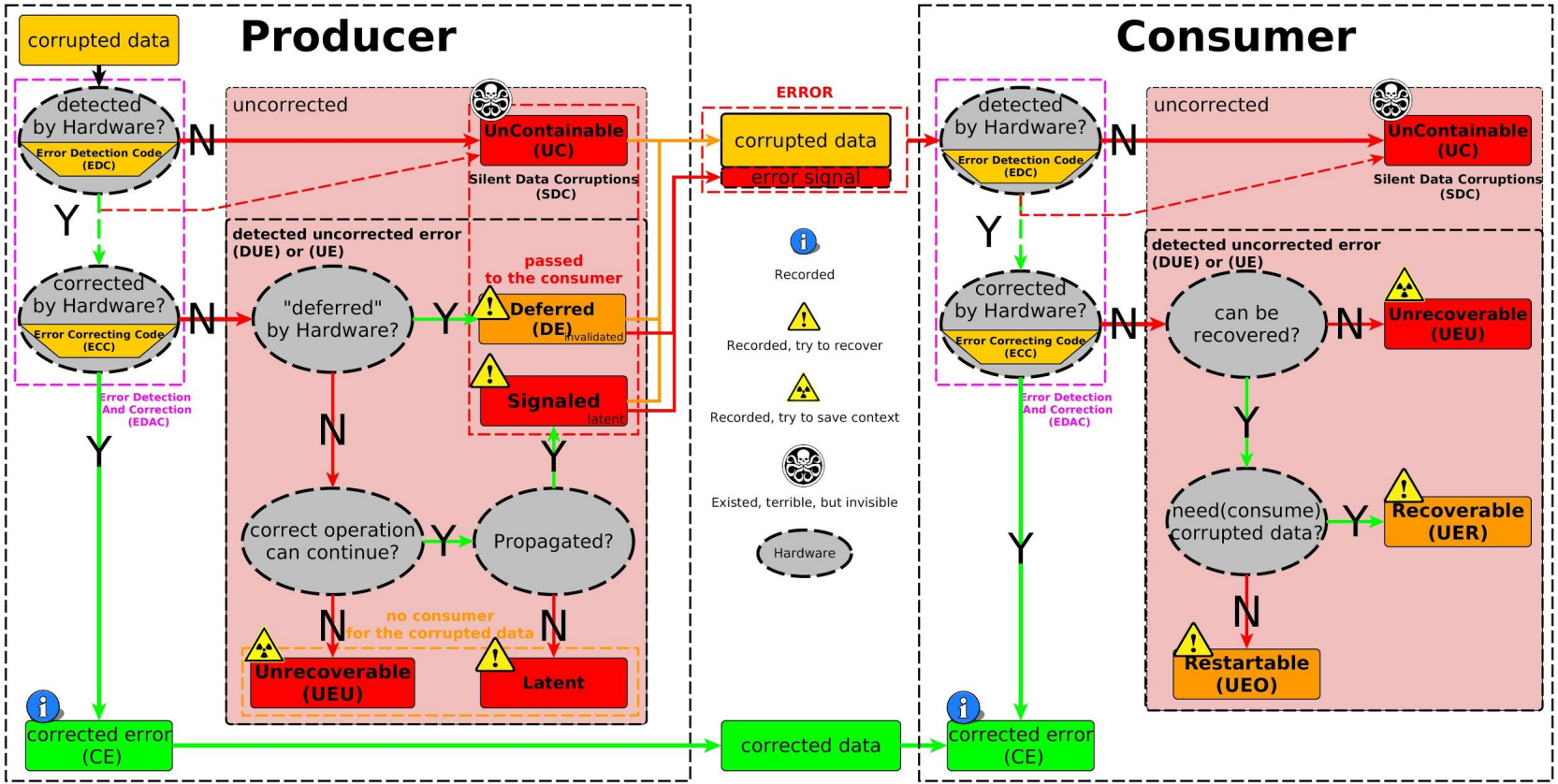
- **EDAC** (Error Detection and Correction)
  - designed to report and possibly act on hardware errors
  - inspect the hardware directly (system-specific handling and decoding.)
  - only support memory controller and PCI/AGP errors

Firmware (first) FF

- APEI
- UEFI

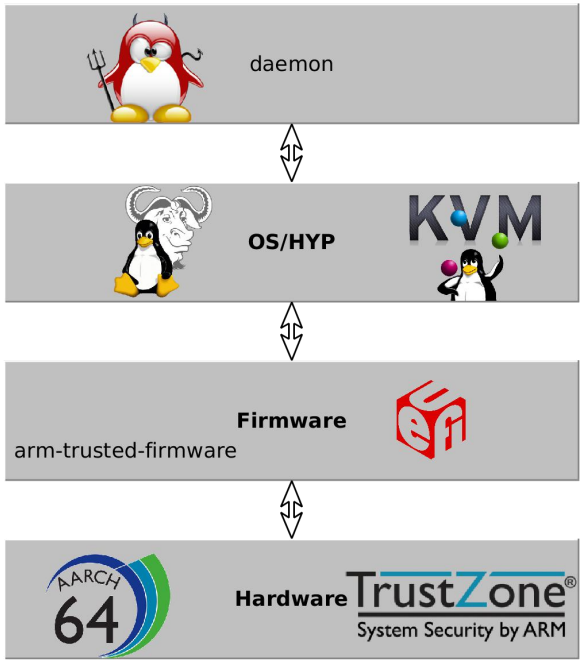


Taxonomy of Error in Diagram



RAS Architecture

Firmware First error handling requires standard interfaces between multiple SW components.



RAS Extension: ESB instruction

- ESB (Error Synchronization Barrier) can be used to **synchronize** **Unrecoverable errors**(containable errors)**.**
- Software can determine that:
  - The error was reported as **Unrecoverable**.
  - The preferred return **address** of the SEI is an ESB instruction.
  - The software between **that ESB and the previous ESB** can be **isolated.**
  - ESB might update :
    - DISR_EL1 / DISR (Deferred Interrupt Status Register)
    - VDISR_EL2 / VDISR (Virtual Deferred Interrupt Status Register)

RAS Extension: Registers

System register views

(in Arm ® Architecture Reference Manual)

- Processor Feature Register
- Memory Model Feature Register
- Error Record Register
  - Error Record ID Register
  - Error Record **Select** Register
  - [**Selected** Error Record] Feature Register
  - [...] Control Register
  - [...] Primary Status Register
  - [...] Address Register
  - [...] Miscellaneous Register **[0-3]**
- Deferred Interrupt Status Register
- Hypervisor Configuration Register
- Virtual SError Exception Syndrome Register
- Virtual Deferred Interrupt Status Register
- Secure Configuration Register

Memory-mapped view

- Error Record Feature Register
- Error Record Control Register
- Error Record Primary Status Register
- Error Record Address Register
- Error Record Miscellaneous Register**[0-3]**
- **Pseudo-fault Generation Feature register**
- **Pseudo-fault Generation Control register**
- **Pseudo-fault Generation Countdown register**
- Error Group Status Register
- Implementation Identification Register
- Error Interrupt Configuration Register[FHI, ERI, CRI][0-2]
- Error Interrupt Status Register
- Device Affinity Register
- Device Architecture Register
- Device Configuration Register
- Peripheral Identification Register[0-4]
- Component Identification Register[0-3]

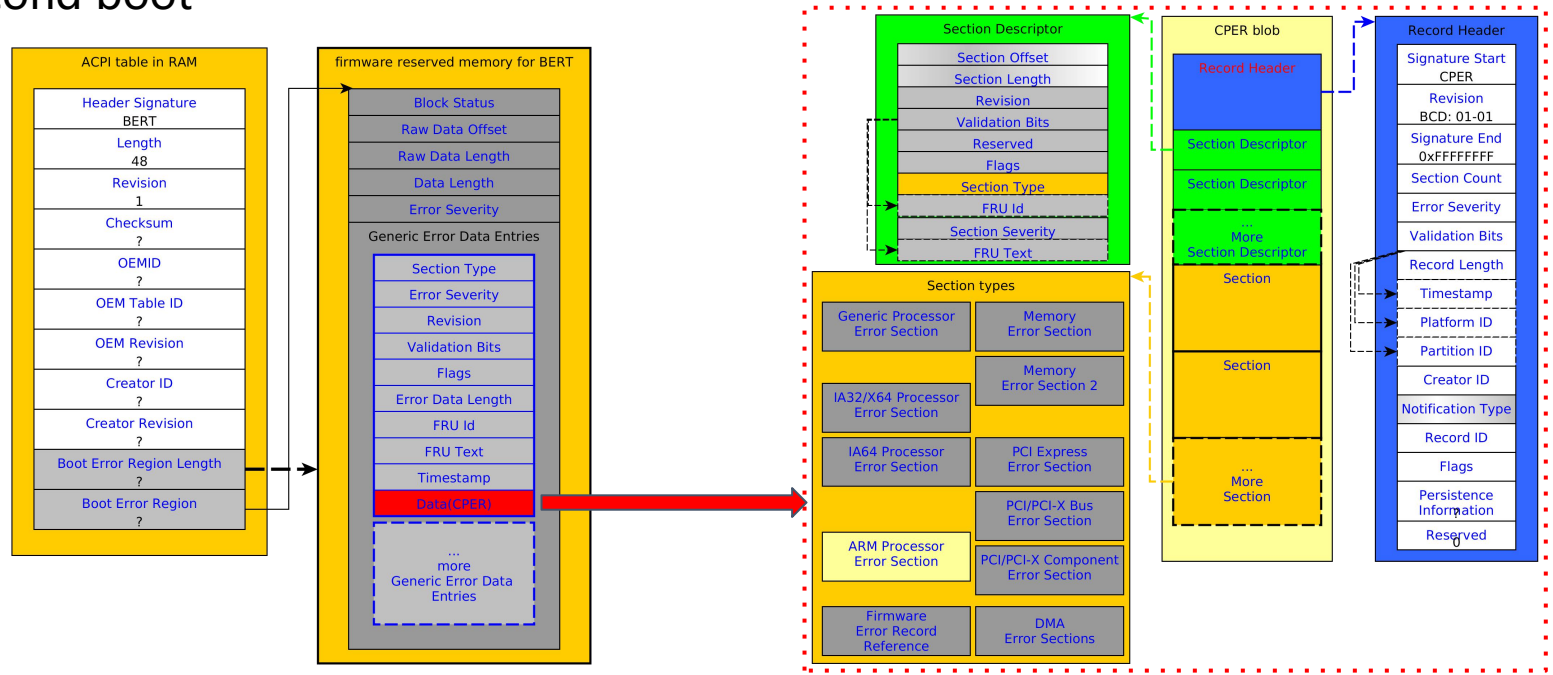
APEI tables

BERT: Boot Error Record Table

Record fatal errors, then report it in the second boot

CPER(in the Appendix of UEFI spec)

Common Platform Error Record

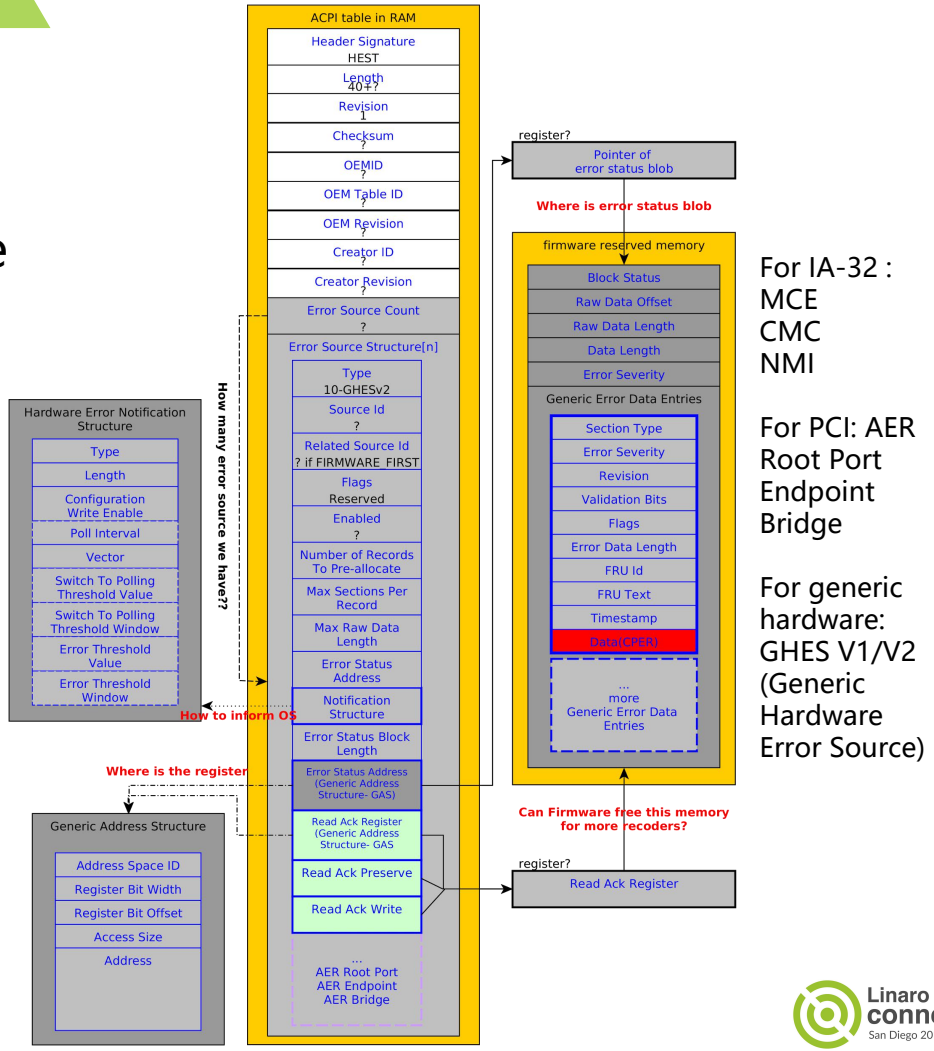


APEI tables

HEST: Hardware Error Source Table

- Key info:
  HOW to get trigger
  WHERE are the error records
  HOW to release records' mem

- For ARM64 : **GHES v2**
  HOW to get trigger: Notification Structure
  WHERE are the error records:
  Error Status Address
  (GAS : Generic Address Structure)
  HOW to release records' mem:
  Read Ack Regi



For IA-32 :
MCE
CMC
NMI

For PCI: AER
Root Port
Endpoint
Bridge

For generic hardware:
GHES V1/V2
(Generic Hardware Error Source)

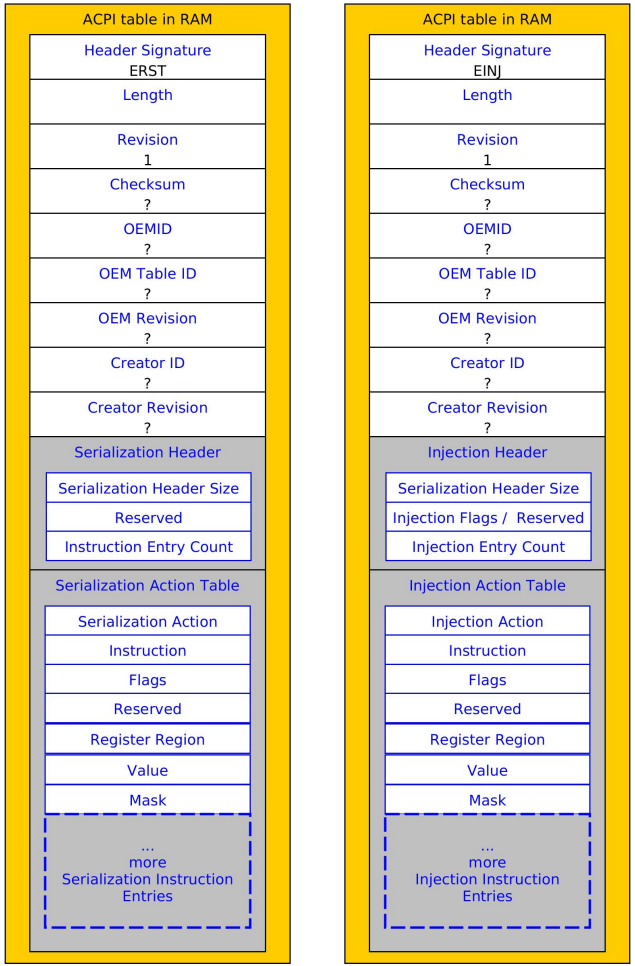
APEI tables

ERST: Error Record Serialization Table

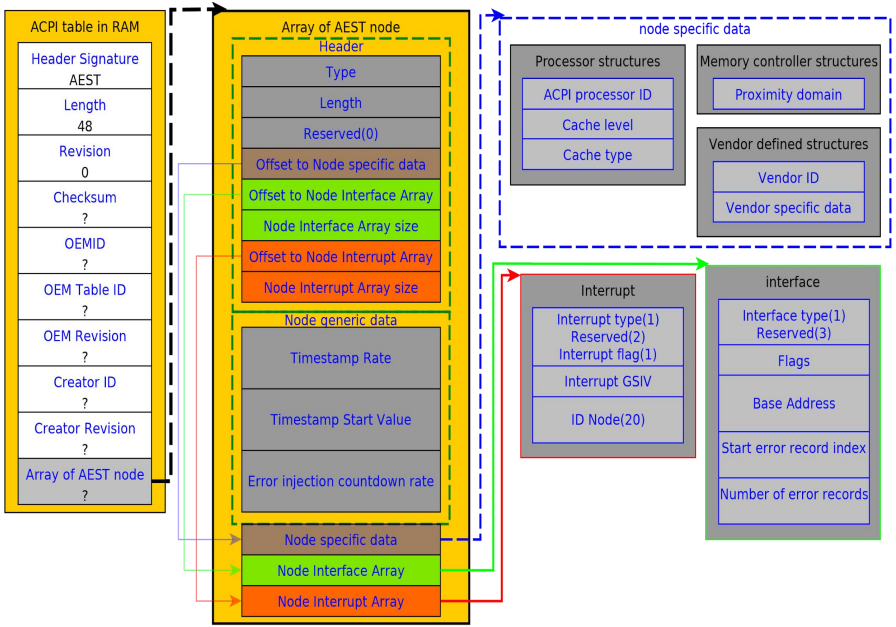
Operation abstract, provides details necessary to communicate with on-board persistent storage for error recording.

EINJ: Error Injection Table

Operation abstract, provides a generic interface which OSPM can **inject hardware errors to the platform** without requiring platform specific software.



ACPI for the Armv8 RAS Extensions 1.0(proposal-1)

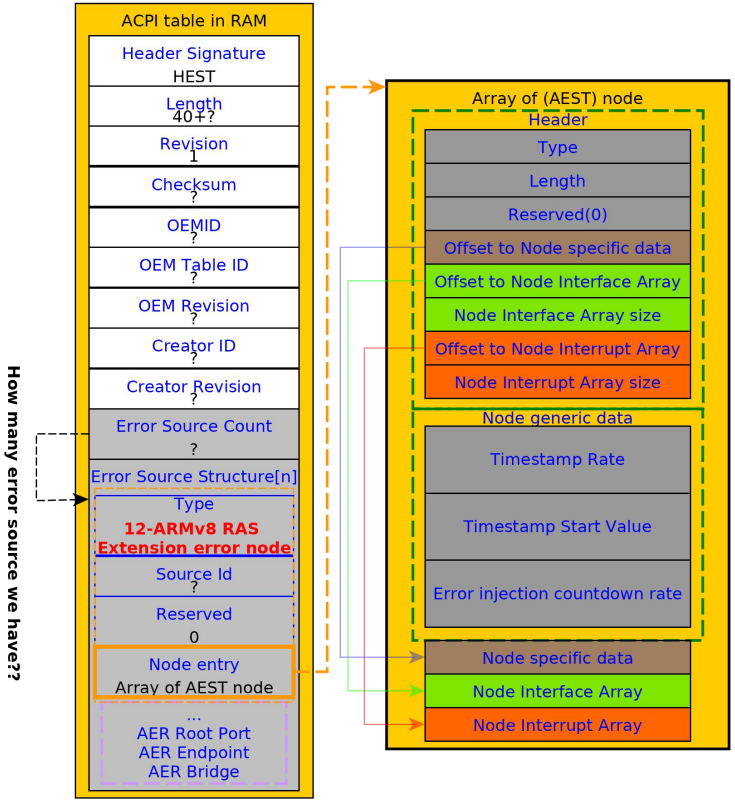


AEST: Arm Error Source Table

AEST node are composed of the following parts:

- A **header**
- A set of **common** fields for all error nodes(Node generic data)
- A component **specific** section
- A section that describes the **interfaces** associated with an error node.
- A section describing associated **interrupts** with the error node.

ACPI for the Armv8 RAS Extensions 1.0(proposal-2)

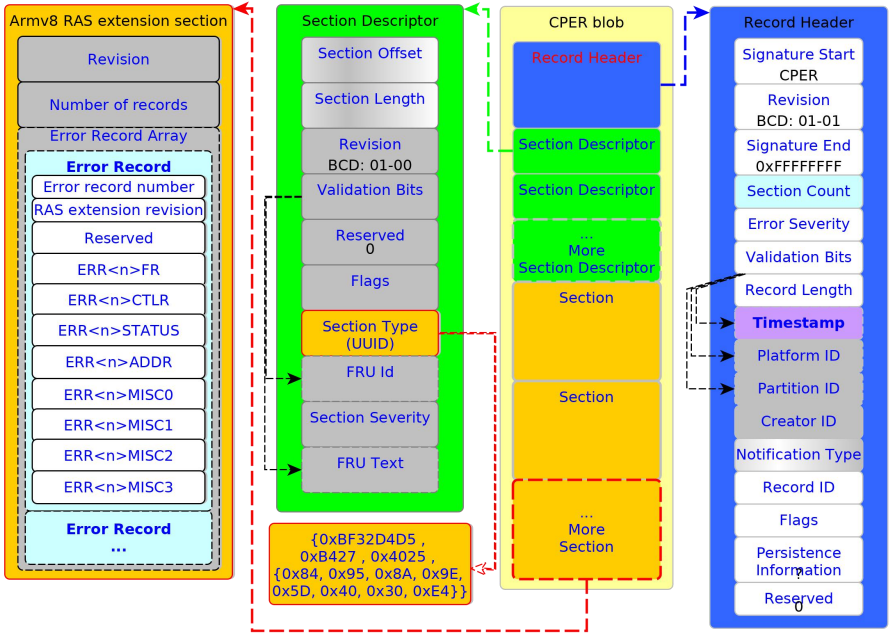


Integrating AEST into HEST

Q: Do we have to add a brand new table for the extension?

An alternative way is to integrate the "AEST" nodes into the HEST directly by creating new error source structure, Type: **12-ARMv8 RAS Extension error node**

ACPI for the Armv8 RAS Extensions 1.0(proposal-3)



CPER Armv8 RAS extension section

The section gathers the content of a RAS extension node’ s registers without any specific information that can associate it with components in the SoC, because it is used alongside other error sections already defined for CPER, such as processor sections, generic and Arm, or Memory sections.

Uncorrected Error -- HEST & MM

Boot Time

1. System boot: BootROM-->BL2-->BL3x
   a. **BL31** initializes SPM, SDEI dispatcher and BL32(MM dispatcher)
   b. **UEFI** (BL33), DXE, UEFI Platform Driver:
      i. query MM partitions by **APEI protocol** for error source info
      ii. MM partitions return error source info back to **UEFI**
      iii. **UEFI** map in and mark error record region as Runtime Services Data Region
      iv. Update/add error source info in **HEST**
2. OS starts running: **HEST driver** scan HEST table and register error handlers by **SDEI**

RunTime

3. UE occurred, the event will be routed to EL3 (SPM)
4. SPM routes the event to RAS error handler in S-EL0 (MM partitions)
5. MM Foundation **(CperLib)** creates the CPER blobs by the info from RAS Extension
6. SPM notifies **SDEI** to call the corresponding OS registered handler
7. OS gets the CPER blobs by Error Status Address block, process the error, try to recovery.
8. report the error event by **RAS event**
9. **rasdaemon** log error info from RAS event to recorder

APEI protocol: API

Main

- Apei.c
  - **UpdateApei** :
    updates error source information to the specified APEI(HEST/BERT) Table

```
typedef
EFI_STATUS
        (EFIAPI *EFI_APEI_UPDATE) (
        IN CONST EFI_APEI_PROTOCOL     *This,
        IN UINT32
        Signature
        );
```

- ApeiCommon.c
  - GetApeiTable
  - AcpiTableChecksum

For HEST table:

- ApeiHest.c
  - UpdateHest
    - GetErrorSources
    - UpdateGhes

For BERT table(TBD):

- ApeiBert.c:
  - UpdateBert
    - GetErrorInfo
    - FillBert

CperLib: API

- **CperInit:**
  parse the error source info, and return the error record address info

```
EFI_STATUS
EFIAPI
CperInit (
  IN EFI_APEI_ERROR_SOURCE  *ErrorSource,
  IN OUT UINTN                        *ErrorRecordAddress
);
```

The Dynamic Tables module passes the ErrorSource to help CperLib get the memory region info for other APIs, like CperWrite

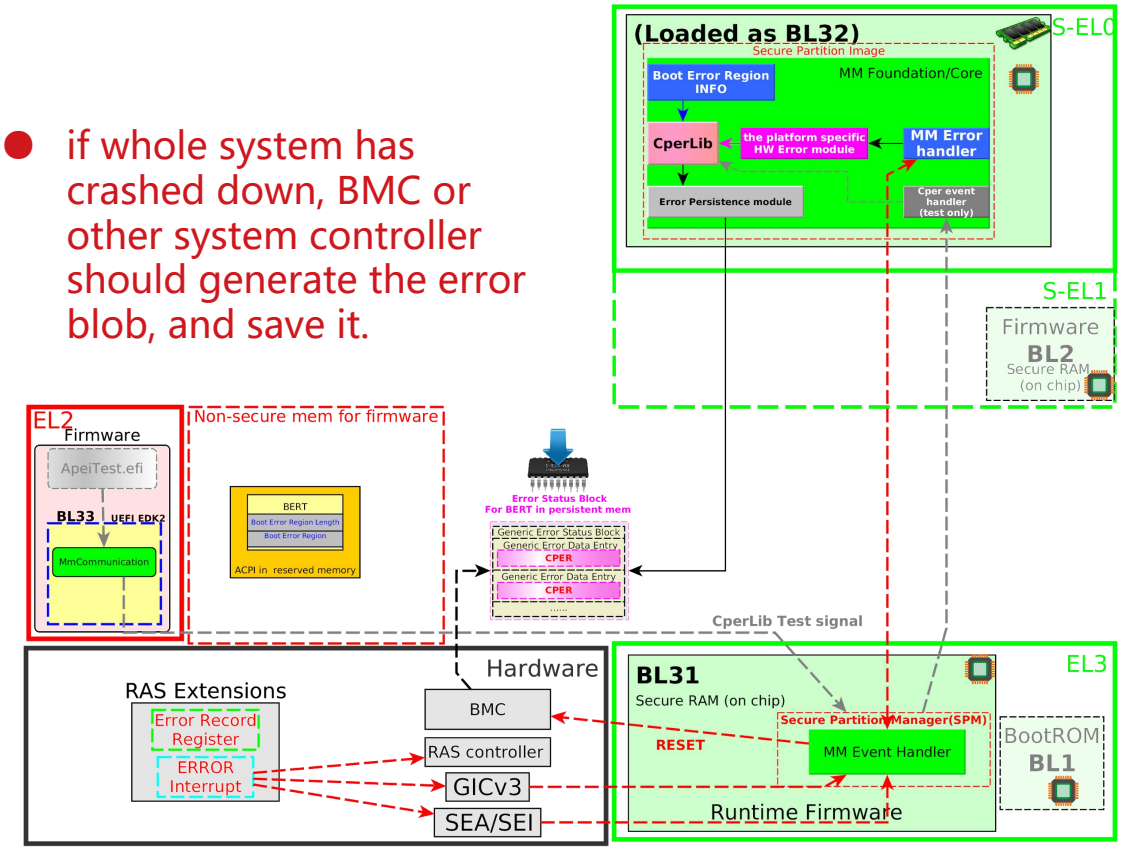
- **CperWrite:**
  creates a CPER blob at the defined memory region from Section info data

```
EFI_STATUS
EFIAPI
CperWrite (
  IN SECTIONS_INFO    *SectionInfo,
  IN UINTN                                ErrorRecordAddress
);
```

Wrap the given section data into CPER blob and put it into the specific memory region

BERT : When catastrophic errors occur(TODO)

- if whole system has crashed down, BMC or other system controller should generate the error blob, and save it.



BERT: After emergency reboot(to be improved)