



OCP FMFM (FLEET MEMORY FAULT MANAGEMENT) FRAMEWORK REQUIREMENTS

Version 1.0

Revision 1

Acknowledgement

This document is a result of collective effort of several OCP participants, who came together in the weekly meetings of the Fleet Memory Fault Management (FMFM) sub-project over a period of two plus years, to share their expertise in the field of data-center reliability, availability and serviceability (RAS), to distill their decades of collective RAS experience and pain-points into a common set of fleet memory fault management solution requirements.

Contributors: (Sorted alphabetically by company name then last name)

Company Name	Individual (s) Representing Company
AMD	Hank Bruning (Principal Member of Technical Staff)
AMD	Majed Beigi (Senior Member of Technical Staff)
AMD	Vilas Sridharan (Sr Fellow)
Cisco	Charlie Slayman (Technical Leader)
Dell	Kevin Cross
Formerly Intel	Shen Zhou
Formerly Intel	Yogesh Varma (RAS Technologist) OCP FMFM Co-Lead
Formerly Meta	Panos Christeas (Engineer)
Formerly SK Hynix	Paul Fahey
Google	Sean Malley (Staff Software Engineer)
Meta	Anil Agrawal (Engineer)
Micron	Ana Maria Camacho Negrete (DRAM Systems Engineer)
Micron	Dale Hiscock (DRAM Systems Eng Principal Engineer)
Micron	Evan Pearson (DRAM Systems Eng Manager)

Micron	Owen Straub (DRAM Systems Engineer)
Micron	Sakul Gupta (MTS, Firmware Security Engineering)
Micron	Tony Geidl (DRAM Systems Eng Director)
Microsoft (formerly Google)	Drew Walton (Fault Management Lead) OCP FMFM Co-Lead
Microsoft (formerly AMD)	Carlos Vallin (Principal Member of Technical Staff)
Microsoft	Rama B
Pure Storage	Meeta Saggi (Member of Technical Staff)
Samsung	Jongmin Lee (Memory QA & Fault Management Principal Engineer)
Samsung	Aaron Choi (Memory Product Planning Principal Engineer)
Samsung	Stuti Patel (Fault Management Engineer)
Samsung	Jude Ko (Fault Management Engineer)
SK Hynix	Roy Chung (RASM Lab)

Executive Summary

This document outlines the specifications for a fault resilient system architecture designed to address DRAM faults. It falls under the umbrella of a broader corpus of OCP contributions in Fault Management (see [OCP Fault Management Infrastructure Requirements Ver 1.0](#)). The framework described in this specification defines the content and format of memory error logs, such that they can be ingested across a standardized API into vendor proprietary analysis engines. It also outlines the framework for integrating the actions recommended by the analysis engines back into the server fleet.

The initial version of this specification focuses on DDR5 memory. The terms memory fault and DRAM fault will be used interchangeably in this document, and they refer specifically to DDR5 memory. Future versions will extend that baseline to include other types of ECC-based memory technologies.

License

OCP encourages participants to share their proposals, specifications and designs with the community. This is to promote openness and encourage continuous and open feedback. It is important to remember that by providing feedback for any such documents, whether in written or verbal form, that the contributor or the contributor's organization grants OCP and its members irrevocable right to use this feedback for any purpose without any further obligation.

It is acknowledged that any such documentation and any ancillary materials that are provided to OCP in connection with this document, including without limitation any white papers, articles, photographs, studies, diagrams, contact information (together, "Materials") are made available under the Creative Commons Attribution-ShareAlike 4.0 International License found here:

<https://creativecommons.org/licenses/by-sa/4.0/>, or any later version, and without limiting the foregoing, OCP may make the Materials available under such terms.

As a contributor to this document, all members represent that they have the authority to grant the rights and licenses herein. They further represent and warrant that the Materials do not and will not violate the copyrights or misappropriate the trade secret rights of any third party, including without limitation rights in intellectual property. The contributor(s) also represent that, to the extent the Materials include materials protected by copyright or trade secret rights that are owned or created by any third-party, they have obtained permission for its use consistent with the foregoing. They will provide OCP evidence of such permission upon OCP's request. This document and any "Materials" are published on the respective project's wiki page and are open to the public in accordance with OCP's Bylaws and IP Policy. This can be found at <http://www.opencompute.org/participate/legal-documents/>. If you have any questions please contact OCP.



About Open Compute Foundation

The Open Compute Project Foundation is a 501(c)(6) organization which was founded in 2011 by Facebook, Intel, and Rackspace. Our mission is to apply the benefits of open source to hardware and rapidly increase the pace of innovation in, near and around the data center and beyond. The Open Compute Project (OCP) is a collaborative community focused on redesigning hardware technology to efficiently support the growing demands on compute infrastructure. For more information about OCP, please visit us at <http://www.opencompute.org>



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

Table of Contents

Acknowledgement	2
Executive Summary	4
License	5
About Open Compute Foundation	6
Compliance with OCP Tenets	8
Openness	8
Efficiency	8
Impact	8
Scale	8
Sustainability	8
1. Problem Statement and Rationale	9
2. Goals	9
3. Overview of the Fleet Memory Fault Management (FMFM) Framework	10
Vendor Specific Fault Analyzer Usage	12
VSFA Integration Details	14
4. Required DDR DIMM Data Logging Capabilities	16
Boot Time Data Logging	16
Run Time Data Logging (Runtime error logging)	24
Crash Time Data Logging (Collect data before system shutdown/reboot)	27
5. Output of DRAM Fault Analyzer (DFA)	28
DFA Log Output Visibility and Consumer:	28
DFA Log Output Format:	29
Recommended DFA Log Output Data Fields:	30
6. Output of System Fault Analyzer	31
SFA Log Output Visibility and Consumer:	31
SFA Log Output Format:	31
Recommended SFA Log Output Data Format:	31
7. FMFM Error Handling Flow Examples	32
8. FMFM Relationship with Complementary Industry Standards	32
9. Conclusion	33
10. References	33
11. Appendices	34
Appendix A - Glossary of Terms	34
Appendix B - Implementation Considerations	34
Appendix C - Memory Analyzer Logs JSON Schema	34

Compliance with OCP Tenets

Openness

This document is designed to facilitate collaboration between DRAM suppliers, SoC suppliers, server suppliers and server fleet operators. It describes error logging details for memory errors and suggests a framework that third parties can use to collect and analyze this data.

Efficiency

The solution described in this document will improve how memory failures are diagnosed and repaired, improving the efficiency of data center operations and reducing the number of DRAMs that are replaced.

Impact

By standardizing the information collected for memory errors, this document enables memory suppliers and SoC suppliers to gather billions of detailed device hours of failure data. Making collection of this data commonplace will be transformative, enabling silicon suppliers to discover new failure modes, improve their products and build tools that help their customers understand hardware failures.

Scale

The data collection and analysis tools described in this document promote scaling by enabling automated error handling and diagnosis. By accurately determining what hardware errors mean, this solution reduces unnecessary manual interventions. This solution is designed with flexibility in mind so that it can be adopted to any server fleet operator.

Sustainability

This contribution provides a standard set of memory error data and suggests an approach to analyzing that data that enables the designers of the silicon to help users understand what is happening with the memory in their fleets. Accurately diagnosing memory failures will improve swap rates, memory reliability and reduce downtime, all of which promote efficient use of materials and reduce environmental impacts.



1. Problem Statement and Rationale

Current memory fault management techniques employed by server fleet operators often fall short in extracting deeper insights from error data, and thus lack the ability to implement precise Reliability, Availability, Serviceability (RAS) actions. This is because microarchitectural details are often proprietary or constantly changing across different generations of products, and the domain knowledge expertise lies with the memory and CPU/SOC vendors, who own the IP. For example, many server fleet operators use operating systems which support page offlining to handle memory correctable errors. The mechanism in the OS that triggers page offlining relies on counting errors before disallowing the use of a specific memory page. While it is a reasonable frontline defense mechanism, it fails to provide deeper insights that would allow more effective memory fault management—insights such as the nature and severity of the memory fault, the likelihood of a subsequent uncorrectable error, the precise identification of faulty elements (e.g. DRAM rows/banks, entire DIMM etc) and, most importantly, the sorts of actions that could mitigate against these underlying faults.

A deep understanding of device microarchitectures can significantly enhance the quality of memory fault management. However, this is often challenging for server fleet operators due to the complexity, proprietary nature and variability of this information across different silicon vendors and even among different generations of the same silicon vendor's products. On the other side of the equation, memory and CPU vendors don't have ready access to large volumes of real world error data—data that would be immensely valuable to drive further innovation in the field. The rich and diverse set of error data from real-world datacenter usage patterns also lends itself well to building AI training sets, which could drive even deeper inferences into memory fault management.

The gap between server fleet operators, who are on the frontline of encountering and detecting interesting memory error pathologies, and the CPU/memory domain experts, is what the proposed framework aims to bridge.

2. Goals

The goal of this specification is to define a framework that comprises:

- **Standardized content for platform error logs:** Current memory fault handling solutions are not standardized in the error information being captured and utilized for driving RAS actions. Logs may be generated by the OS, the BIOS or the BMC, among others, and these logs all have different types of content, each offering a partial view of the fault encountered. In order to drive precise RAS actions, this



framework defines a complete set of error information that should be captured in each memory error log. In order to effectively ingest these logs into analysis pipelines, we propose a standard format based on CPER records.

- **Vendor Specific Fault Analyzers (VSFA):** These analysis engines are provided by the component vendors (CPU or memory) and live on the server fleet operators' infrastructure. They are accessible via well defined API and receive streams of error logs as inputs and provide RAS action or repair action (FRU replacement, reseal, etc.) recommendations as outputs. Examples of RAS actions may be Post Package Repair, migrating workloads from the machine, page offline or doing nothing at all. Internal details of the VSFAs are proprietary and will differ from silicon vendor to silicon vendor or even across product generations. In this way, proprietary implementation details are not exposed—allowing the memory and CPU vendor to continue innovating—and the server fleet operators are able to leverage the valuable inferences from these analyzers in order to enhance outcomes in memory fault management.

Although the components of this framework are inherently location-agnostic, meaning they can live in the OS/BIOS, the BMC or the server fleet operators' common infrastructure, this initial version of the specification will impose some constraints on this flexibility.

3. Overview of the Fleet Memory Fault Management (FMFM) Framework

The OCP HW Fault Management framework (see [OCP Fault Management Infrastructure Requirements Ver 1.0](#)) describes how vendor analysis tools are integrated in the Cloud Provider Infrastructure to manage errors. Fault management automation feeds error logs to the corresponding vendor-provided tools, which decode and analyze them. Based on analysis, the analyzers may suggest service actions to be taken. The OCP FMFM framework (this specification) is focused on server fleet memory fault management by establishing a standard for server memory error classification, terminology, logging requirements, interfaces as well as decoders and analyzers that can help server fleet operators to take appropriate memory RAS handling action.

The following analyzer modules are relevant in the context of Fleet Memory Fault Management:

1. **Error Record Decoder (ERD):** Provides first-level decoding of error logs produced by the processor managing the memory. Examples of processors include host processor (CPU) and accelerator processor (GPU). This module decodes processor-specific error information and provides standard inputs to the DRAM Fault Analyzer module. The processor vendor provides this module.

2. **DRAM Fault Analyzer (DFA):** Provides next-level analysis based on DRAM vendor specifics, which may include DRAM fault model analysis, bank/device granularity, and other parameters specific to the memory architecture. The memory vendor provides this analyzer.
3. **System Fault Analyzer (SFA):** Provides last-level analysis based on the outputs from the DFA module, which may include recommended memory-related service actions, such as performing post-package repair, page offlining, reboot, or RMA. The CPU/memory controller vendor will provide the SFA with the DFA integrated into it.

The following diagram illustrates how these modules are integrated into a single Vendor-Specific Fault Analyzer (VSFA).

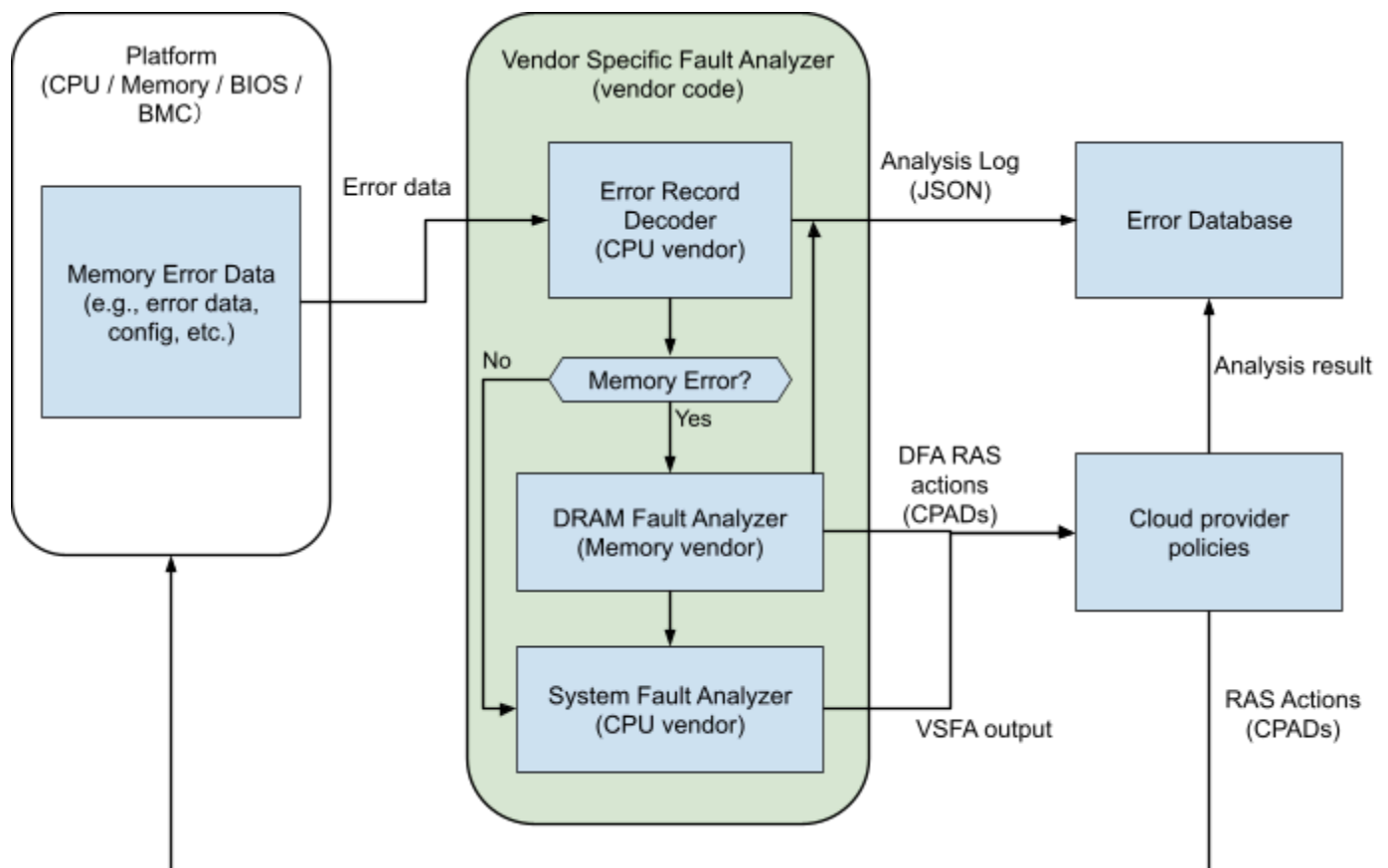


Figure-1: FMFM Framework Overview

All analyzer modules must meet certain standards for interoperability in the Cloud Provider Infrastructure:

1. Analyzers shall provide a unique identifier (GUID) that the Cloud Provider Fault Management automation can use to distribute error logs to the corresponding analyzer. The API for retrieving the GUIDs will follow the design described in the [OCP Hardware Fault Management Requirements Document](#); the RAS API specification will allow fleet operators to associate GUIDs in the CPERs with GUIDs provided by the analyzers.
2. Analyzers shall provide standard APIs to configure analyzer behavior. This is particularly important for the Error Record Decoder, since it may need to configure its outputs depending on the type of DRAM Fault Analyzer module.
3. Input/output data formats must be standardized to ensure interoperability in a hybrid fleet with multiple vendors. Appendix C documents standardized schemas for exchanging information between analyzer modules.
4. Analyzers shall emit Common Platform Action Descriptors(CPADs) for automatable RAS actions and part replacements.
5. Analyzers shall provide analysis logs that contain additional information in JSON to support dashboarding, human analysis, etc.

Vendor Specific Fault Analyzer Usage

This section of the document is not intended to be prescriptive and is intended to provide context for how analysis tools might be used in server operator fleets.

Cloud Providers will run the Vendor Specific Fault Analyzer whenever there is a new error log to process. The analyzer can be run anywhere the cloud provider chooses, including the BMC and in their infrastructure. Many Cloud Providers have resource-constrained BMCs and are also reluctant to add something to the BMC that will drive additional firmware updates. Such Cloud Providers will run the analyzer in their resource management infrastructure where resources are not limited and updates are easy.

When a memory error occurs, the cloud provider will send the new error log plus a specified number of prior error logs to the analyzer as shown below:



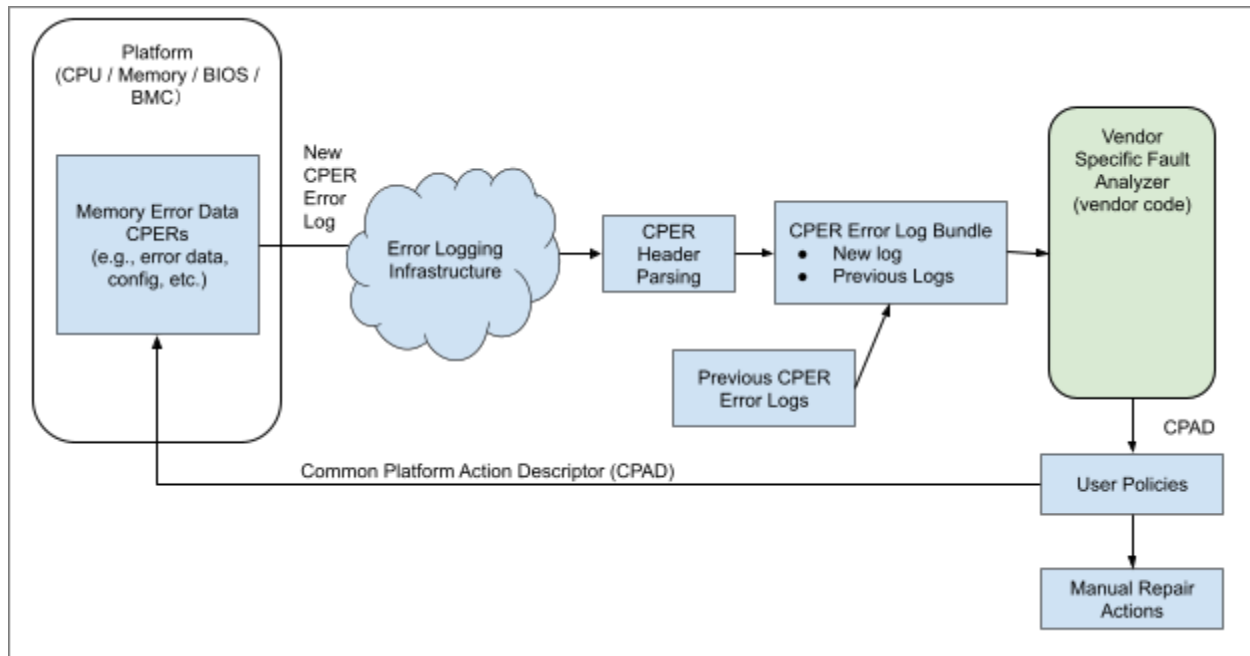


Figure-2: VSFA usage example

The first step in processing a CPER error log is parsing the header to use the CPER's Creator ID field to determine which Vendor Specific Analyzer should be used to process the CPER. The details for associating the CPER with the correct analyzer are described in the RAS API. Once the header is parsed and the target analyzer identified, the infrastructure will be able to tell how much prior data needs to be sent to the Vendor Specific Analyzer based on a call to the analyzer. The Vendor Specific Analyzer is expected to have an interface to tell users how much data to send to it, allowing the VSA to automatically get the desired amount of data any time its needs (and code) change.

The previous error logs are needed to identify failure patterns. The Vendor Specific Analyzer will process the newest log and can look at the prior logs if needed. The number of previous error logs should be specified by the Vendor Specific Analyzer via an API call that is made when the analyzer is started.

For Cloud Providers, the Vendor Specific Analyzer must be stateless, meaning that it does not have an internal database and simply processes the data sent to it and provides a response. There are several reasons for requiring that the analyzer be stateless: One is that there are security concerns with embedding a database into the analyzer. Another reason for a stateless analyzer lies in how cloud providers write fault-tolerant software. They assume that any machine can fail at any time. If the analyzer stored state, we would lose that state when the machine running the analyzer failed. If the analyzer is stateless, the cloud provider can build reliable storage

for error logs and call any machine running an analyzer with all the error logs it needs when there is a new error; it does not matter if a machine running an analyzer fails because they don't have any state and are interchangeable.

VSFA Integration Details

Each SoC/Accelerator vendor will have their own Vendor Specific Fault Analyzer. This section shows how a server operator might integrate multiple analyzers into their framework.

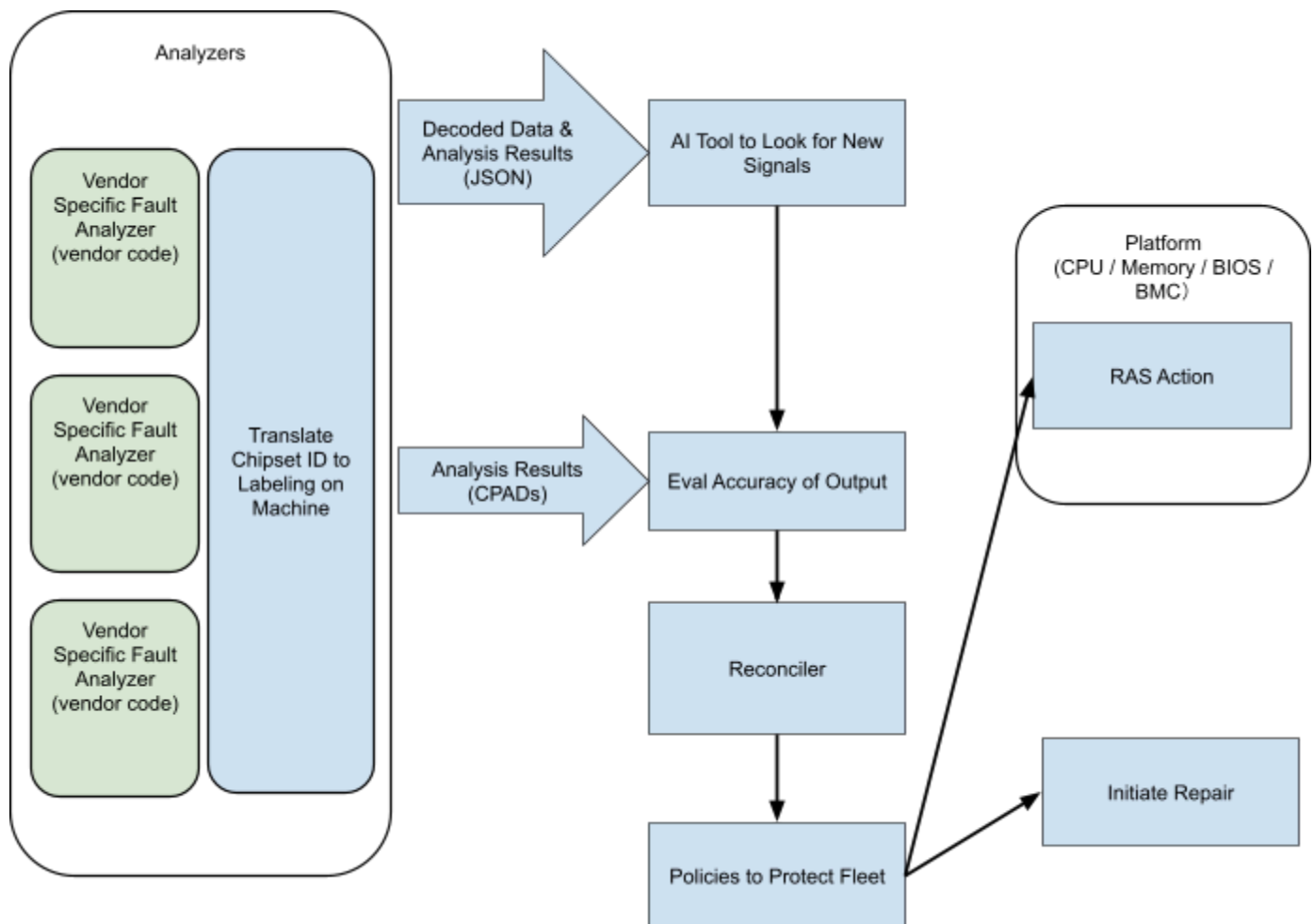


Figure-3: VSFA system integration

The first step is to identify FRUs that were involved in the error. Since the analyzers process CPER error logs from a chip, these error logs reference components in the machine based on the chip's numbering. For example, a

memory controller might log which rank number had an error. The DIMM, however, is installed in a slot in the machine that is not labelled according to the rank number. So we will need to translate the rank number to the label on the DIMM slot.

There are two outputs from analyzers in the OCP Fault Management framework. The analyzer will emit Common Platform Action Descriptors (CPADs) to drive Repair and RAS actions such as part replacements or self-healing actions like Post Package Repair (PPR). The analyzers also output decoded data in the form of a JSON file. This decoded data might be studied in order to find new failure modes or discover new ways to analyze the data. AI could be used to look for new signals in the decoded data.

In an ideal world, the analyzers' conclusions would be perfectly accurate and always correctly identify the best way to mitigate failures. In reality, analyzers will be more accurate for some errors and less accurate for others. As the server operator learns which actions are most effective, they will want to use effectiveness to guide their response to analysis results. If multiple analyzers provide conclusions for a machine, e.g. when a CPU/SOC from one vendor and a PCIe device from another vendor both log errors, the server operator will need to reconcile these conclusions. This is beyond the scope of this document and will not be described here other than to say that the server operator might prioritize actions based on the certainty that the action will solve the problem among other strategies.

The analysis results might also feed AI tools that might:

- Predict crashes
- Look for new failure modes
- Improve the accuracy of the analysis results
- Look for fleet-scale issues
- and more

Server operators will also need to enforce policies that protect their fleets from mistakes in the analyzer. If an analyzer starts suggesting unexpected repairs or large numbers of RAS actions, we might rate limit them or alert fleet operators of a potential problem. The policies to protect the fleet will be specific to the server operator and outside the scope of this specification.

4. Required DDR DIMM Data Logging Capabilities

This section lists data that is logged for analyzing memory errors. It will be updated as memory technology evolves and new specifications are published.

DDR DIMM data logging consists of three phases:

- Boot time data logging: collect server and DRAM configuration, boot time failure, PMIC error history data, etc during system boot
- Run time data logging: collect runtime error data when any error is detected.
- Crash time data logging: collect crash time error data when system crash is detected

Usages of the data logging (How to use):

- Runtime Analysis Data is used as input to the Vendor Specific Fault Analyzer.
 - An API call to the CSP infrastructure will be used for the DFA to request periods of error logging for analysis. This will allow the analysis timeframe to be set by each individual server operator
 - For DFA runtime analysis, a minimum of 2 hours of error logs is recommended
 - DRAM vendor analysis suggests that >93% of DRAM errors are fully defined within a 2 hour runtime window. However, this will vary with workload, etc
- Offline Analysis: Data is utilized for the silicon vendor, memory vendor, and data center operator in post failure offline analysis.

The following table is a list of things that are captured at boot time. We will also capture the identity of the error source as defined in the OCP RAS API so that we can associate silicon that logs errors, CPER logs from that silicon, analyzers for those CPERs and Common Platform Action Descriptors(CPAD) that specific RAS actions to be taken on that silicon.

Boot Time Data Logging

Data to Log	Log Details	Logging Frequency (this could vary and be tunable)	Log Source / Detection Point	How to Use <u>VSFA</u> analyzer input -	Schema Id (For decoded JSON)

				Analyzed at Runtime; Offline/ post-failure analysis	
Platform Firmware Version Number	Platform Firmware version that Collected CPER (whatever collected the CPER, possibly the BIOS or a microcontroller FW)	Every boot	System firmware (BIOS / BMC)	R O	InstalledFirmware
System Memory Configuration	- Number of CPUs - Number of loaded DIMMs per channel - Number of DIMM loading channel	Every boot	System firmware (BIOS / BMC)	R O	SystemMemoryConfig
DIMM Inventory Information	- DIMM supplier - DIMM part number - DIMM serial number	Every boot	System firmware (BIOS / BMC)	R O	InstalledMemory
Primordial boot timestamp	- Ideal data: the timestamp of the first boot of the memory DIMM during its lifetime. - Realistic data: the timestamp of the first boot of the memory DIMM in the current system NOTE: Provide “time to failure (TTF)” to the memory supplier for acceleration test to duplicate failure and finally increase test coverage	Once	System firmware (BIOS / BMC)	O	PrimordialBootDT
Current Boot timestamp	The timestamp of the current boot of the system.	Every boot	System firmware	R O	CurrentBootDT

			(BIOS / BMC)		
PMIC Error Log History Data	Global History of Error Logs R04 [7]: GLOBAL_ERROR_COUNT R04 [6]: GLOBAL_ERROR_LOG_BUCK_OV_OR_UV R04 [5]: GLOBAL_ERROR_LOG_VIN_BULK_OVER_VOLTAGE R04 [4]: GLOBAL_ERROR_LOG_CRITICAL_TEMPERATURE R04[3:0]: Reserved Error Log Registers: 0x05 R05[7]: Reserved R05 [6]: SWA_POWER_GOOD R05 [5]: SWB_POWER_GOOD R05 [4]: SWC_POWER_GOOD R05 [3]: SWD_POWER_GOOD R05 [2:0]: PMIC_ERROR_LOG Error Log Registers: 0x06 R06[7]: Reserved R06 [6]: SWA_UNDER_VOLTAGE_LOCKOUT R05 [5]: SWB_UNDER_VOLTAGE_LOCKOUT R05 [4]: SWC_UNDER_VOLTAGE_LOCKOUT R06 [3]: SWA_OVER_VOLTAGE R06 [2]: SWB_OVER_VOLTAGE R06 [1]: SWC_OVER_VOLTAGE R06 [0]: SWD_OVER_VOLTAGE NOTE: - During boot the system is able to check the PMIC fault that occurred during the previous boot. - These registers record the PMIC state at each abnormal power down cycle.	Every boot	PMIC data	R O Distinguish the error from DRAM device or PMIC device.	PmicBootErrorLog

	- These registers report the cumulative error of each abnormal power down sequence. These registers are persistent until an erase command is issued. - The PMIC writes this register on its own when it internally generates the VR Disable command on its own due to failure. However, it may not be guaranteed depending on how fast the host may shut off the input power to the PMIC				
DCA/DFE/VR EFDQ Offset	DCA/DFE/VREFDQ offset - DCA/DFE/VREFDQoffset (MR 42, 43, 44, 103-254) - Can omit 117, 119, 125, 127, 135, 143, 155, 159, 167, 175, 183, 191, 199, 207, 215, 223, 231, 239, 247 Need Duty Cycle Adjuster (DCA) Bits for DQS/DQ signal integrity. Additionally other Signal Calibration Fields (DFE & VREFDQ) for fine-tuning internal parameters. Other bits in MR42 can be ignored.	- Fast boot disabled condition: Every boot (Cold/warm boot) - Fast boot enabled condition: first boot	DRAM mode register	O	DramOffsetRegisters
RCD Control Words	Unpaged Control Words: - Drive Characteristics: RW0A: QCK Signals Driver Characteristics RW0C: QCA and QxCS_n Signals Characteristics RW0E: QCK, QCA and QCS Output Slew Rate RW0D: Data Buffer Interface Driver Characteristics RW0F: BCK, BCOM and BCS Output Slew Rate - Delay Offset: RW12: QACK Output Delay RW13: QBCK Output Delay	- Fast boot disabled condition: Every boot (Cold/warm boot) - Fast boot enabled condition: first boot	RCD data	O	RcdControlWords

	RW14: QCKK Ouput Delay Control RW15: QDCK Output Delay Control RW17: QACS0 Output Delay RW18: QACS1 Output Delay RW19: QBCS0 Output Delay RW1A: QBCS1 Output Delay RW1B: QACA Output Delay RW1C: QBCA Output Delay RW1D: BCS and BCOM Output Delay RW1E: BCK Output Delay Paged Control Words (Page 0) RW60: DCA0 Rx DFE Gain Coefficient RW61: DCA0 Rx DFE Tap 1 Coefficient RW62: DCA0 Rx DFE Tap 2 Coefficient RW63: DCA0 Rx DFE Tap 3 Coefficient RW64: DCA0 Rx DFE Tap 4 Coefficient RW68: DCA1 Rx DFE Gain Coefficient RW69: DCA1 Rx DFE Tap 1 Coefficient RW6A: DCA1 Rx DFE Tap 2 Coefficient RW6B: DCA1 Rx DFE Tap 3 Coefficient RW6C: DCA1 Rx DFE Tap 4 Coefficient RW70: DCA2 Rx DFE Gain Coefficient RW71: DCA2 Rx DFE Tap 1 Coefficient RW72: DCA2 Rx DFE Tap 2 Coefficient RW73: DCA2 Rx DFE Tap 3 Coefficient RW74: DCA2 Rx DFE Tap 4 Coefficient RW78: DCA3 Rx DFE Gain Coefficient RW79: DCA3 Rx DFE Tap 1 Coefficient RW7A: DCA3 Rx DFE Tap 2 Coefficient				
--	--	--	--	--	--

	RW7B: DCA3 Rx DFE Tap 3 Coefficient RW7C: DCA3 Rx DFE Tap 4 Coefficient Paged Control Words (Page 1) RW60: DCA4 Rx DFE Gain Coefficient RW61: DCA4 Rx DFE Tap 1 Coefficient RW62: DCA4 Rx DFE Tap 2 Coefficient RW63: DCA4 Rx DFE Tap 3 Coefficient RW64: DCA4 Rx DFE Tap 4 Coefficient RW68: DCA5 Rx DFE Gain Coefficient RW69: DCA5 Rx DFE Tap 1 Coefficient RW6A: DCA5 Rx DFE Tap 2 Coefficient RW6B: DCA5 Rx DFE Tap 3 Coefficient RW6C: DCA5 Rx DFE Tap 4 Coefficient RW70: DCA6 Rx DFE Gain Coefficient RW71: DCA6 Rx DFE Tap 1 Coefficient RW72: DCA6 Rx DFE Tap 2 Coefficient RW73: DCA6 Rx DFE Tap 3 Coefficient RW74: DCA6 Rx DFE Tap 4 Coefficient RW78: DPAR Rx DFE Gain Coefficient RW79: DPAR Rx DFE Tap 1 Coefficient RW7A: DPAR Rx DFE Tap 2 Coefficient RW7B: DPAR Rx DFE Tap 3 Coefficient RW7C: DPAR Rx DFE Tap 4 Coefficient				
--	--	--	--	--	--

Data Buffer Control Words	Unpaged Control Words - Drive/termination: RW86: DQS PARK RW87: Host Interface Park Termination RW88: Reserved RW89: Reserved RW8A: Host Interface DQ Driver RW8B: DRAM Interface MDQ Driver and Termination RW8C: MDQS and MDQ Park Paged Control Words: (Page 4 - 0x04) RWE0: DQ0 Rx DFE Gain Coefficients RWE1: DQ0 Rx DFE Tap 1 Coefficients RWE2: DQ0 Rx DFE Tap 2 Coefficients RWE3: DQ0 Rx DFE Tap 3 Coefficients RWE4: DQ0 Rx DFE Tap 4 Coefficients RWE8: DQ1 Rx DFE Gain Coefficients RWE9: DQ1 Rx DFE Tap 1 Coefficients RWEA: DQ1 Rx DFE Tap 2 Coefficients RWEB: DQ1 Rx DFE Tap 3 Coefficients RWEC: DQ1 Rx DFE Tap 4 Coefficients RWF0: DQ2 Rx DFE Gain Coefficients RWF1: DQ2 Rx DFE Tap 1 Coefficients RWF2: DQ2 Rx DFE Tap 2 Coefficients RWF3: DQ2 Rx DFE Tap 3 Coefficients RWF4: DQ2 Rx DFE Tap 4 Coefficients	Every boot(Cold/w arm boot) if fast cold boot is disabled	Data buffer	O	DataBufferControlWords

	RWF8: DQ3 Rx DFE Gain Coefficients RWF9: DQ3 Rx DFE Tap 1 Coefficients RWFA: DQ3 Rx DFE Tap 2 Coefficients RWFB: DQ3 Rx DFE Tap 3 Coefficients RWFC: DQ3 Rx DFE Tap 4 Coefficients Paged Control Words: (Page 5 - 0x05) RWE0: DQ4 Rx DFE Gain Coefficients RWE1: DQ4 Rx DFE Tap 1 Coefficients RWE2: DQ4 Rx DFE Tap 2 Coefficients RWE3: DQ4 Rx DFE Tap 3 Coefficients RWE4: DQ4 Rx DFE Tap 4 Coefficients RWE8: DQ5 Rx DFE Gain Coefficients RWE9: DQ5 Rx DFE Tap 1 Coefficients RWEA: DQ5 Rx DFE Tap 2 Coefficients RWEB: DQ5 Rx DFE Tap 3 Coefficients RWEC: DQ5 Rx DFE Tap 4 Coefficients RWF0: DQ6 Rx DFE Gain Coefficients RWF1: DQ6 Rx DFE Tap 1 Coefficients RWF2: DQ6 Rx DFE Tap 2 Coefficients RWF3: DQ6 Rx DFE Tap 3 Coefficients RWF4: DQ6 Rx DFE Tap 4 Coefficients RWF8: DQ7 Rx DFE Gain Coefficients RWF9: DQ7 Rx DFE Tap 1 Coefficients RWFA: DQ7 Rx DFE Tap 2 Coefficients				
--	--	--	--	--	--

	RWFB: DQ7 Rx DFE Tap 3 Coefficients RWFC: DQ7 Rx DFE Tap 4 Coefficients				
Training Event Logs	Failure training step and DIMM location: • Failure training step • Failed DIMM location ○ Host ID ○ Socket ○ Channel ○ Slot ○ Sub-Channel ○ Device ○ DQ NOTE: Boot failure (specially training failure) is usually intermittent and needs to be logged	Every boot if it has. (Fast cold boot will be disabled automatically whenever configuration is changed. If there is no change, it won't be trained and no data will be saved)	BIOS / MRC	O	TrainingEventLogs
Memory Controller metadata	Memory Controller specific configuration meta data such as ECC mode, etc.	Every boot	Memory Controller	R	MemoryControllerMetadata

Run Time Data Logging (Runtime error logging)

Data to Log	Log Details	Logging Frequency	Log Source / Detection Point	How to Use <u>VSFA</u> analyzer input - Analyzed	Schema Id (For decoded JSON)

				at Runtime; <u>Offline/</u> post-failure analysis	
Error Timestamp	Timestamp when error is detected: - Recommended format [Year:Month:Date:Hour:Min:Sec:MSec] NOTE: - Sometimes there is time difference between "System" and "OS" - Recommended time is OS time due to the manufacturing site and real deployed field time difference	Every error	Memory Controller	R O	ErrorDT
Error Type	Error Type: - Severity: Informational / CE / Recoverable / UE / Fatal - Defined in OCP HW FM Requirements Doc - Informational may be used to indicate important events such as a CPAD was received and acted upon - Demand read error / Patrol scrub error / Other	Every error Error storm condition: Draw back to second level or sub-second level sampling during error storms period.	Memory Controller	R O	ErrorScrubType, ErrorSeverity
Memory Error Location	Memory Error Location (address sent from the memory controller to the DRAMs): - Host ID - Socket	Every error Error storm condition: Draw back to second level or sub-second level sampling during error storms period.	Memory Controller	R O	HostID, SocketId, ChannelId, ChannelSlotId, SubChannel, Rank, DeviceId,

	- Channel - Slot - Sub-channel - Rank (+CID for TSV) - Device - Bank Group - Bank - Row - Column				BankGroup, Bank, Row, Column ChannelSlotId is the memory controller's view of the DIMM on the channel.
Error Bit(s) Location	ECC syndrome for error bit(s) NOTE: - This is the combination of DQ in the device and the burst length information - By accumulating the error bit location(DQ & Burst length), Memory suppliers are able to identify the DRAM failure mode	Every error Error storm condition: Draw back to second level or sub-second level sampling during error storms period.	Memory Controller	R O	ErrorVector
OS Physical Address	System OS physical address where the error is detected. This is the address used by the operating system to access a memory location. It is not a X-Y coordinate in the DRAM array.	Every error Error storm condition: Draw back to second level or sub-second level sampling during error storms period.	Memory Controller	R O	OsPhysicalAddress
CPU Vendor Metadata	CPU vendor specific data such as retry information, permanent fault bit(s), etc.	Every error Error storm condition: Draw back to second level or sub-second level sampling during error storms period.	Memory Controller	R	ProcessorMetadata
DRAM ECS Mode Registers	ECS MRs (MR14 - 20) NOTE:	Every 24 hours	DDR5 ECS	R	EcsModeRegisters

	The timing is correlated to the patrol scrub period and the default setting is 24hr		mode register		
Memory Temperature Data	Memory temperature: DRAM TSOD data	Every error Error storm condition: Draw back to second level or sub-second level sampling during error storms period.	Temperature sensor on DIMM	R O	DimmTemp
PMIC Runtime Status	PMIC runtime status: PMIC real time status registers 0x8, 0x9, 0xA, 0xB. NOTE: These registers are set when value goes below/above threshold.	Every error Error storm condition: Draw back to second level or sub-second level sampling during error storms period.	PMIC registers	R	PmicRuntimeRegisters

Crash Time Data Logging (Collect data before system shutdown/reboot)

Data to Log	Log Details	Logging Frequency	Log Source / Detection Point	How to Use <u>V</u> SFA analyzer input - Analyzed at <u>R</u> untime <u>O</u> ffline/post-failure analysis	Schema Id
Crash Timestamp	Timestamp of system crash detected:	Every crash	System firmware (BIOS/BMC)	O	SystemCrashDT, OsCrashDT, Uptime

	- Recommended format [Year:Month:Date:Hour:Min:Sec:MS ec] NOTE: - Sometimes there is time difference between “System” and “OS” - Recommended time is OS time due to the manufacturing site and real deployed field time difference				
PMIC Error Log	PMIC error log registers: Register 0x4, 0x5, 0x6	When system crash caused by PMIC errors	PMIC register	O	PmicCrashRegisters
Fatal Error Flag	Fatal error detected e.g., Command Address Parity(CAP) fatal error	When fatal error is detected	Fatal error	O	FatalError

5. Output of DRAM Fault Analyzer (DFA)

The interface between the memory vendor’s DRAM Fault Analyser (DFA) and the CPU vendor’s System Fault Analyzer (SFA) does not need to be exposed externally. We may want some level of standardization of the interface, but there may be microarchitectural details that the memory vendors want to pass to the CPU analyzer that are proprietary to the memory vendor. We do not expect the memory vendors to reveal this proprietary information on public interfaces in the analyzers.

DFA Log Output Visibility and Consumer:

From a visibility perspective, the output of the DFA can be classified as two levels :

- Level 1:** Public information - Information that is safe to share on external interfaces.



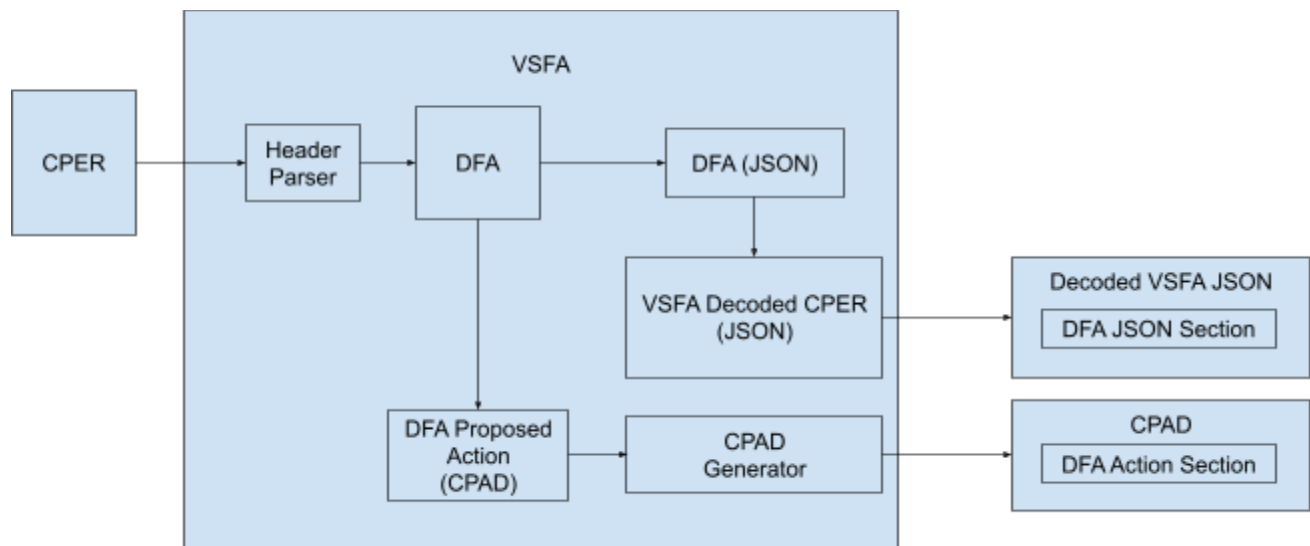
- Level 2:** Proprietary information - Information that is shared only on internal interfaces (not output to the analyzer's end user). The decoding of the proprietary information is between CPU vendors and memory vendors under NDA.
 - This part of the interface will not be defined in the public OCP forums
 - We acknowledge it is needed and that the memory and CPU vendors are free to implement it

The silicon vendors' System Fault Analyzer (SFA) is the primary consumer of the DFA logs. It could be used for making RMA decisions and/or in-field repair by the SFA.

DFA Log Output Format:

The DFA output is intended to give direct feedback from DRAM vendors to system operators, without the need for strict integration into the System Fault Analyzer (SFA). DFA output is intended to be referenced in case of a DRAM error, to view the DRAM vendor's own output for what the cause of the error is and how it should be handled. The SFA may provide additional context for the same error, or in the case of the DFA suggesting a system error, the SFA can provide additional context and final decision.

As mentioned above, the Vendor Specific Fault Analyzers are expected to emit CPADs to drive actions such as part replacements and self-repairs. Likewise, the DFA will emit a CPAD section to request an action and JSON decoded data that describes what the DFA has concluded about the DRAM from the error(s). The decoded data must be exposed to the server operator with the VSFA's decoded output as shown below:



Recommended DFA Log Output Data Fields:

- Analysis result - DIMM and Failure Info:
 - DIMM serial number
 - Faulty component location:
 - Host ID, Socket, Channel, Slot, Sub-channel, Rank, Chip Select ID (CID) for Through Silicon Via(TSV) for high-density DIMMs, Device
 - History of multiple devices failure
 - Classified DRAM fault mode (level1)
 - Intra-bank fault: Single Bit(SB), COL, ROW, BANK
 - Inter-bank fault: IO, DEV, and others
 - Classified DRAM fault mode (level2)
 - Proprietary information defined between CPU vendor and memory vendor under NDA
 - E.g. Impacted DQ-Bit Line(BL) map, Impacted DRAM rows, mis-correction of in-DRAM ECC, etc.
- RAS Action Suggestions:
 - sPPR/ hPPR: with faulty row address (for cell fault, row fault)
 - Page offline: with a range of faulty DRAM addresses to bound fault (all DRAM fault types)
 - CPU analyzer provides the address translation from DRAM address to OS physical address mapping
 - The DFA should be able to infer mapped out pages until we design handshake with OS
 - If the error is only detected by patrol scrub, the OS might have mapped out the page
 - In this version of this specification, the DFA might suggest mapping page out again if it sees repeated patrol scrub errors
 - Future goal is to have the OS report that it mapped the page out to the RAS API
 - System issue Y/N
 - DFA makes suggestion for system issue, but leaves CPU analyzer as the expert to make system issue RAS decision
 - DRAM retraining
 - System reboot
 - DIMM reseal
 - System overtemp
 - Bad training parameters
 - Live-migration
 - High/ Low severity classifier (UE predictive analysis)
 - High severity faults suggested for DIMM removal and RMA
 - Confidence level
 - Number from 0 to 100
 - Vendors can use numbers as they see fit
 - Will be used to guide policy decisions by SFA users

6. Output of System Fault Analyzer

The output of CPU vendors' System Fault Analyzer (SFA) provides the last-level analysis based on the error data from DFA in conjunction with the other CPU/SOC level error handling capabilities and configurations.

SFA Log Output Visibility and Consumer:

The Fleet operator's fault handling agent is the primary consumer of the SFA output. Depending on the implementation type, this agent could be a CSP infrastructure Control plane or a simple node level-BMC. This would be exposed on a RAS API aware interface. (E.g. - using Redfish RAS API extension.)

SFA Log Output Format:

The SFA output should be:

- Common Platform Action Descriptor(CPAD) for actions
 - Repairs
 - Invoking RAS features such as PPR
- In a Machine-readable format such as JSON.
 - For decoded error logs

Recommended SFA Log Output Data Format:

The following items should be included in the SFA output. This list will get updated as the OCP RAS API team defines the CPAD format.

- Output Timestamp:
 - Timestamp of the analyzer output:
 - Recommended format [Year:Month:Date:Hour:Min:Sec:MSec]
- Identified Fault Details:
 - Faulty DIMM serial number
 - Faulty DIMM location
 - Host ID
 - Socket
 - Channel
 - Slot
 - Sub-channel
 - Inferred fault mode (level 1 - fault mode classification):
 - Single bit fault
 - Row fault
 - Multi-Row fault
 - Column fault
 - Bank fault
 - Device fault



- DIMM fault
 - System issue
 - Potential Row Hammering
- Risk classifier
 - Risk to SDC - fault pattern exposes risk to SDC according to ECC
 - Risk to UE - fault pattern exposes risk to UE according to ECC
 - Risk to repeated CE or CE storm - fault pattern could be correctable but exposes the risk of occurring errors repeatedly (intermittent or permanent)
 - Low Risk - e.g. soft errors
- Severity classifier
 - [high severity] field irrecoverable high severity fault: fault cannot be recovered with in-field RAS action or fault still exists after suggested RAS action attempt, suggesting for DIMM removal and RMA
 - [medium severity] field recoverable fault: fault can be recovered with field RAS action, suggesting for RAS action
 - [low severity] no immediate high risk, keep monitoring
- RAS Action Suggestion:
 - RAS action suggestion, format [suggested RAS action, RAS action target]
 - [sPPR, faulty row address]
 - [hPPR, faulty row address]
 - [Page Offline, faulty page address(es)]
 - [Bank sparing, faulty bank address]
 - [Fast boot]
 - [Cold boot]
 - [DIMM reseal, faulty DIMM serial number / location]
 - [Faulty DIMM Removal, faulty DIMM serial number / location]

7. FMFM Error Handling Flow Examples

The OCP Fault Management Teams have created separate fault management scenario documents.

8. FMFM Relationship with Complementary Industry Standards

The following table lists some of the existing industry standards that are leveraged for creating FMFM framework.

#	FMFM Framework Requirement	Complementary Industry Standard	Relationship & Reference
1	Standard Input Log Format	UEFI CPER	Adopted by FMFM UEFI Spec, Appendix N



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](#).

2	Standard Output Log Format	RAS API -> UEFI CPAD	Adopted by FMFM OCP RAS API Specification
3	DDR5 Hardware RAS Actions	JEDEC	Adopted by FMFM - JEDEC DDR5
4	Software RAS Actions	OS Dependent	Respective OS's Documentation - Linux , Windows

9. Conclusion

OCP Fleet Memory Fault Management Requirements create a baseline framework for vendor agnostic DDR5 memory fault logging and analysis. The framework is designed as an industry standards based memory error handling interface that is expected to reduce vendor specific fault management software overhead. The FMFM framework is compliant with the logging formats prescribed in the OCP RAS API specification. FMFM framework enables vendors to protect their intellectual property (IP), promoting innovation. Future work will expand the FMFM framework to additional DRAM technologies including LPDDR and HBM.

10. References

1. [OCP Fault Management Infrastructure Requirements](#)
2. ACPI Specification: https://uefi.org/sites/default/files/resources/ACPI_Spec_6_5_Aug29.pdf
3. UEFI Specification: https://uefi.org/sites/default/files/resources/UEFI_Spec_2_10_Aug29.pdf
4. JEDEC DDR4 SDRAM Specification: <https://www.jedec.org/standards-documents/docs/jesd79-4a>
5. JEDEC DDR5 SDRAM Specification: <https://www.jedec.org/standards-documents/docs/jesd79-5b>
6. [JEDEC PMIC5000/PMIC5010 Power Management IC Standard JESD301-1A.03](#)
7. OCP RAS API Specifiation - <https://www.opencompute.org/documents/2022-open-compute-specification-ras-api-v0-8-docx-pdf>

11. Appendices

Appendix A - Glossary of Terms

 OCP Fault Management Glossary of Terms

Appendix B - Implementation Considerations

This section dives into some implementation considerations any BMC-based error logging solution would need to support.

Appendix C - Memory Analyzer Logs JSON Schema

[JSON Schema](#) is a standardized method for defining and validating the structure of JSON documents. Within the context of Fleet Memory Fault Management, it provides a formal mechanism to enforce data consistency and improve accessibility across diverse systems and vendors. By implementing a JSON Schema, log entries can be automatically validated to ensure that required fields are present and that data types are strictly enforced. This consistency reduces the need for manual preprocessing and allows engineers to focus more on failure analysis rather than data cleanup. This section addresses the need for standardized error data formatting from the spec.

The schema design supports both one-to-one and one-to-many relationships, enabling straightforward ingestion into structured storage systems such as relational or document-oriented databases. This structure is particularly well-suited for integration with automated fault analysis tools.

Fleet operators are expected to provide three distinct log types: boot, runtime, and crash. Due to their size, the full schema definitions can be accessed via the OCP FMFM [shared drive](#). While the schema enforces strict requirements for field names, formats, and data types, it also allows for extensibility. Vendors may include additional, proprietary fields as needed. These optional fields are not subject to validation constraints and may follow any format, providing flexibility without compromising core data integrity.

