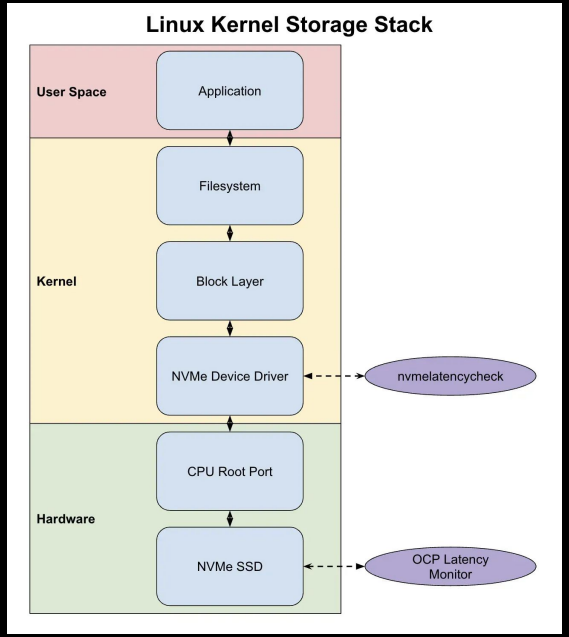
Flash Fleet – Update

Vineet Parekh, Meta

Latency Monitoring

- **Previous Limitations**: Host-level checks missed extreme tail latencies (P99.999999+) due to their transient nature.
- **Challenge**: Intermittent SSD failures made debugging and data collection difficult for vendors.
- **Solution – Latency Monitoring**: Enables precise detection of tail latencies and stalls, improving visibility and SSD-level diagnostics.



Thousands Of

Drives Enabled

SSDs with Latency Monitoring active

How Latency Monitoring Works

Latency Buckets: I/O completion times are categorized into configurable latency ranges.

Operation Tracking: Monitors read, write, and TRIM operations across these latency thresholds.

Dual Counters: Uses active (short-term) and static (long-term) counters for real-time and historical insights.

Classification

Each I/O operation is sorted into one of four latency buckets based on completion time

Thresholds

Host-defined boundaries (e.g., 100ms, 250ms, 500ms, 1000ms) determine bucket assignment

Persistence

Data persists across resets and power cycles for long-term analysis

Latency Bucket Configuration

Bucket	Latency Range	Example Threshold
Bucket 0	≤ Threshold A	100ms
Bucket 1	> A and ≤ B	250ms
Bucket 2	> B and ≤ C	500ms
Bucket 3	> C and ≤ D	1000ms

Thresholds A through D are configured by the host system and define the latency boundaries for different operation types (read, write, and TRIM commands). This bucketing approach allows for granular analysis of performance degradation patterns.

Core Components: Active Counters

Active Bucket Counter

Track I/O latency in the current window; reset periodically (e.g., every 7 days). Ideal for short-term diagnosis and recent latency spikes.

Active Measured Latency

Captures mean latency (ms) of I/Os within each bucket, indicating severity. Returns zero if unsupported or no matching I/Os.

Active Latency Time Stamp

Marks when each active bucket was last updated. Aids in correlating latency events with host or app logs. Shows N/A if inactive.

Core Components: Static Counters

Static Bucket Counter

Store total I/O counts from the last window; persist across resets/power cycles. Useful for spotting chronic issues or long-term degradation.

Static Measured Latency

Tracks average latency per bucket since reset. Complements static counters for trend analysis and historical comparisons.

Static Latency Time Stamp

Logs when each static bucket was last updated. Helps identify timing of stalls or degradation for effective debugging.

```
-Latency Monitor/C3 Log Page Data-
  Controller   :  nvme0
  Feature Status                        0x7
  Active Bucket Timer                   4250 min
  Active Bucket Timer Threshold         10080 min
  Active Threshold A                    30 ms
  Active Threshold B                    100 ms
  Active Threshold C                    155 ms
  Active Threshold D                    235 ms
  Active Latency Minimum Window         1000 ms
  Active Latency Stamp Units            4
  Static Latency Stamp Units            1028
  Debug Log Trigger Enable              4032
                                                  Read                          Write                Deallocate/Trim
  Active Latency Mode: Bucket 0                      1                              1                              1
  Active Latency Mode: Bucket 1                      1                              1                              1
  Active Latency Mode: Bucket 2                      1                              1                              1
  Active Latency Mode: Bucket 3                      1                              1                              1

  Active Bucket Counter: Bucket 0                    0                             93                              0
  Active Bucket Counter: Bucket 1                    0                              0                              0
  Active Bucket Counter: Bucket 2                    0                              0                              0
  Active Bucket Counter: Bucket 3                    0                              0                              0
  Active Measured Latency: Bucket 0                  0 ms                           0 ms                           0 ms
  Active Measured Latency: Bucket 1                  0 ms                           0 ms                           0 ms
  Active Measured Latency: Bucket 2                  0 ms                           0 ms                           0 ms
  Active Measured Latency: Bucket 3                 75 ms                           0 ms                           0 ms
  Active Latency Time Stamp: Bucket 0              N/A                            N/A                            N/A
  Active Latency Time Stamp: Bucket 1              N/A                            N/A                            N/A
  Active Latency Time Stamp: Bucket 2              N/A                            N/A                            N/A
  Active Latency Time Stamp: Bucket 3    2025-06-29 11:56:10.534 GMT              N/A                            N/A
  Static Bucket Counter: Bucket 0                    0                            319                              0
  Static Bucket Counter: Bucket 1                    0                              0                              0
  Static Bucket Counter: Bucket 2                    0                              0                              0
  Static Bucket Counter: Bucket 3                    0                              0                              2
  Static Measured Latency: Bucket 0                  0 ms                        3521 ms                           0 ms
  Static Measured Latency: Bucket 1                  0 ms                           0 ms                           0 ms
  Static Measured Latency: Bucket 2                  0 ms                           0 ms                           0 ms
  Static Measured Latency: Bucket 3                 77 ms                           0 ms                           0 ms
  Static Latency Time Stamp: Bucket 0              N/A         2025-06-20 00:51:32.449 GMT                    N/A
  Static Latency Time Stamp: Bucket 1              N/A                            N/A                            N/A
  Static Latency Time Stamp: Bucket 2              N/A                            N/A                            N/A
  Static Latency Time Stamp: Bucket 3    2025-06-24 05:09:30.273 GMT              N/A                            N/A
[[root@twshared11580.08.gtp3 ~]# ls -l /var/log/iostat*
```

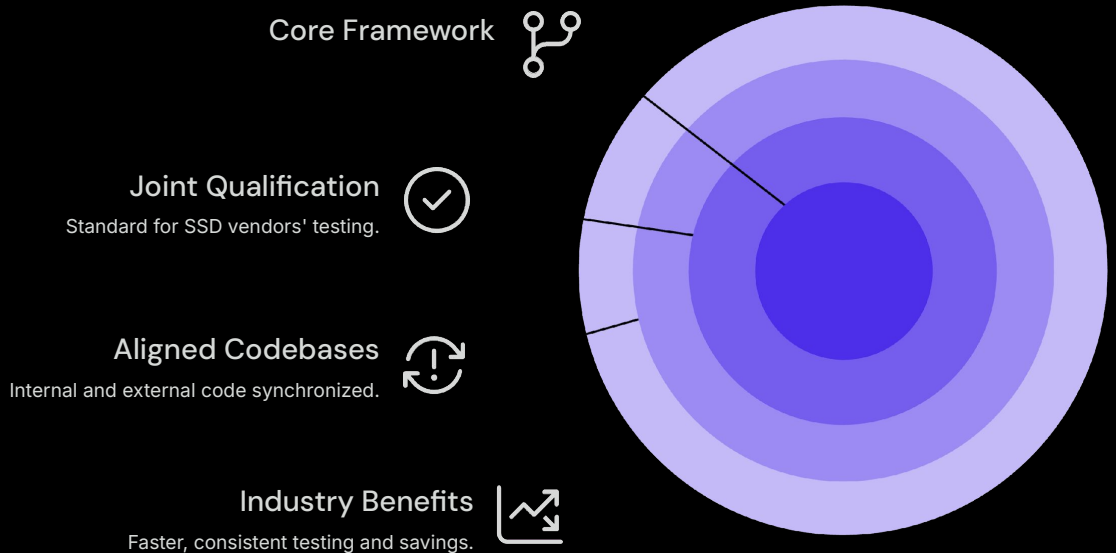
Fleet-Wide Detection Results

- Active bucket data rolls into static buckets after 7 days, preserving latency history for analysis.
- A **stall** is defined as a read or write operation taking **over 1 second** to complete.
- Latency events are time stamped for precise correlation within the monitoring window.
- In a sample 8-day period, **6.7K drives** showed stall events based on Latency Monitoring logs.

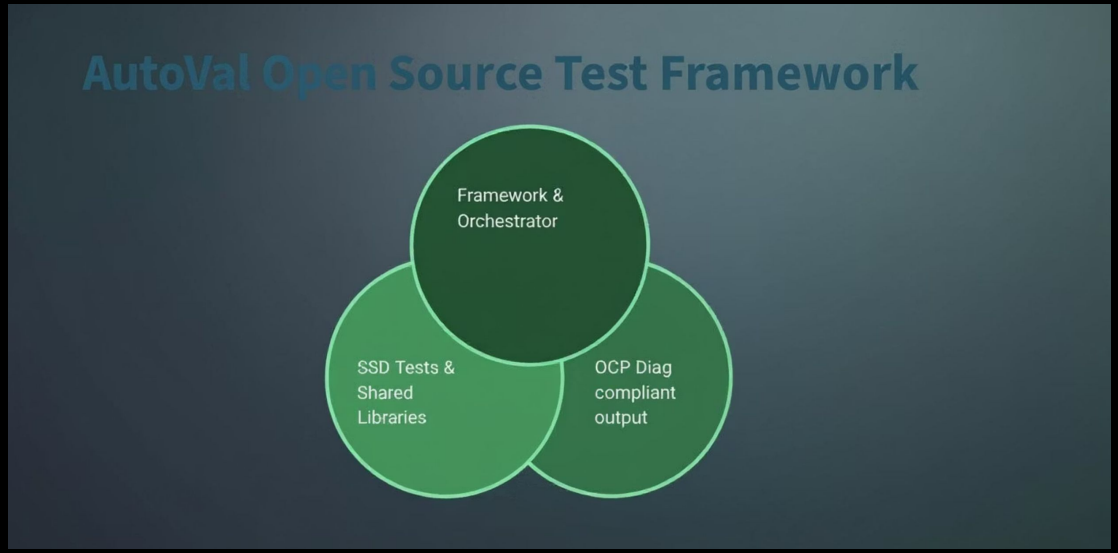
Vendor A	3
Vendor B	14
Vendor C	6716
Vendor D	7

Open Source SSD Test Cases: Revolutionizing Storage Qualification

- **Open-Sourced Testing**: Meta has open-sourced its SSD qualification tests via OCP Diag Autoval on GitHub, aligning internal and vendor workflows.
- **Industry Standardization**: Establishes a unified framework for joint SSD and firmware qualification, accelerating testing and reducing industry-wide effort.



Key Benefits and Savings



25%

Time Reduction

Qualification timeline reduced from eight weeks to six weeks in one program.

100%

Lab Consistency

Results are now fully reproducible and comparable between Meta and vendor labs.

↓

Resource Optimization

Dramatic reduction in resource commitment for a single drive qualification process.

Implementation Timeline

1. **November 2023**
   Meta discontinues Hi5 support
2. **Q1 2024**
   SSD test cases shared under NDA with SSD Vendors
3. **May 2024**
   OCP Diag Autoval Framework + several SSD test cases made public on GitHub
4. **November 2024**
   SSD Vendors agreed to use and support GitHub codebase
5. **H1 2025**
   Several Joint Quals with Meta and SSD vendors kicked off using Open source SSD tests
6. **June 2025**
   Remaining SSD test cases uploaded to GitHub

AutoVal Framework Core Capabilities

Setup

During this stage, `TestBase` performs the following steps.

- Creates log directories
- Initializes the test inputs such as test control arguments, hosts config
- Verifies if the DUT is reachable
- Initiates background monitors according to the test control file settings

Execute

During this stage, `TestBase` executes the test case. The test-specific code execution starts by `execute()` method. This method must be implemented by all test cases that inherit from `TestBase`.

Cleanup

During this stage, `TestBase` releases resources and performs test cleanup activities such as deleting temporary log directories, compressing additional test logs, and storing data in the results directory. If the test case fails, cleanup will call a method `on_fail()`

Multi-Step Validation

Tests consist of multiple steps with individual pass/fail verdicts and diagnostic feedback, improving failure traceability and debugging.

Reusable Libraries

Shared functionality is encapsulated in common libraries, reducing duplication and enforcing consistency across test cases.

Test Life Cycle Management

Each test follows a clear initialization, execution, and teardown cycle with built-in error recovery and result reporting.

Environment Flexibility

Designed to operate in secure lab environments using SSH-based access to control DUTs and associated BMCs.

SSD Test Case Coverage and Interfaces

```python
def execute(self) -> None:
    power_trigger = self.test_control.get("power_trigger", False)
    workloads = self.test_control["workloads"]
    run_definition = workloads["nvme_flush_write"]

    for current_cycle in range(1, self.iteration_count + 1):
        self.validate_no_exception(
            self.run_fio,
            [run_definition, power_trigger],
            "Cycle %d: Fio write job completed." % current_cycle,
        )
```

Test Code

Execute a test:

```
autoval_test_runner
autoval_ssd.tests.nvme_cli.nvme_cli --config
hosts.json --test_control
autoval_ssd/tests/nvme_cli/control.json
```

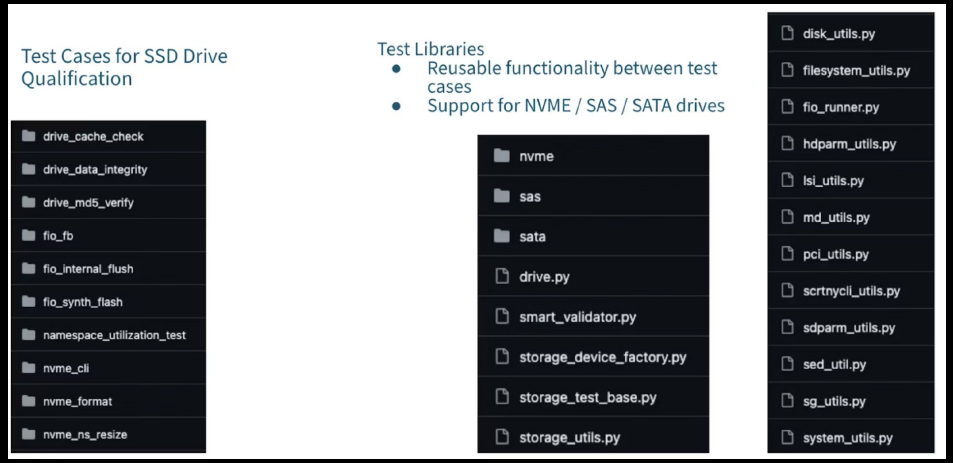
```json
"drive_interface": "nvme",
"power_trigger": true,
"iteration_count": 3,
"cycle_type": "warm",
"workloads": {
    "nvme_flush_write": {
        "args": {
            "RW": "write",
            "VERIFY": "md5",
            "BLKSIZE": "4k",
            "RUNTIME": "10m",
            "SIZE": "100G",
            "DEPTH": 128
        }
    },
```

Test Control

- NVMe CLI interactions
- Data integrity verification
- Cache and flush behavior testing
- Namespace resizing
- Filesystem operations

The AutoVal framework supports SSD testing across multiple storage interfaces including NVMe, SAS, and SATA, with a comprehensive library of test modules.

Test Execution Flow



Command Invocation

Test initiated via CLI command specifying test module, configuration file, and control parameters.

Configuration Loading

Host details loaded from JSON config file, while test parameters pulled from control file.

Test Execution

Framework orchestrates test logic execution, leveraging shared libraries for consistency.

Result Generation

Output generated in OCP Diag-compliant JSON format with pass/fail status, metadata, and timestamps.

Diagnostic Output and Result Analysis

The AutoVal framework generates standardized, parsable output that can be analyzed programmatically or reviewed manually by engineering teams.

Standardized Format

OCP Diag-compliant JSON format ensures consistent reporting across all test runs regardless of environment.

Comprehensive Metadata

Output includes configuration details, test parameters, timestamps, and complete execution history.

Collaborative Debugging

Common output format enables Meta and vendor teams to efficiently compare results and resolve issues together.

Resources and Next Steps

Official GitHub Repositories

- OCP Diag Autoval Framework
- OCP Diag Autoval SSD Test Cases
- OCP Diag Autoval SSD Test Suites

Documentation and References

- OCP Official Blog Post

How to Get Started

For other hyperscalers and SSD vendors looking to implement the AutoVal framework, start by cloning the GitHub repositories and following the documentation. For questions or support, use GitHub issue tracking to engage with the community.

Thank You!