TEST

test_examples.py ●

write_uncorrectable_test.py ×

OUTPUT   Python Test I ∨

PYNVME QPAIRS

0000:3d:00.0 Q00: >>

CAZ-82512-Q11 NVMe LITEON 512GB

0000:3d:00.0 Q02: >>

0000:3d:00.0 Q03: >>

0000:3d:00.0 Q04: >>

0000:3d:00.0 Q05: >>

0000:3d:00.0 Q06: >>

PYTHON

scripts > conformance > 02_nvm_cmd > write_uncorrectable_test.py

```python
1    import time
2    import pytest
3    import logging
4
5    from nvme import Controller, Namespace, Buffer, Qpair, Pcie, Subsyst
6    from scripts.psd import IOCQ, IOSQ, PRP, PRPList, SQE, CQE
7
8
9    # TODO: lba_start=1, lba_step=3, lba_count=3
10
11
     Run Test | Debug Test
12   def test_write_uncorrectable_large_lba(nvme0, nvme0n1, buf):
13       ncap = nvme0n1.id_data(15, 8)
14
15       qpair = Qpair(nvme0, 16)
16       nvme0n1.write_uncorrectable(qpair, ncap-1).waitdone()
17       with pytest.warns(UserWarning, match="ERROR status: 00/80"):
18           nvme0n1.write_uncorrectable(qpair, ncap).waitdone()
19       with pytest.warns(UserWarning, match="ERROR status: 00/80"):
20           nvme0n1.write_uncorrectable(qpair, ncap+1).waitdone()
21       with pytest.warns(UserWarning, match="ERROR status: 00/80"):
```

scripts

conformance

01_admin_cmd

02_nvm_cmd

compare_test.py

deallocate_test.py

flush_test.py

read_test.py

verify_test.py

write_test.py

write_uncorrectable_test.py

write_zeroes_test.py

03_features

04_registers

ftl

performance

snia

stress

tcg

trace

test_examples.py

test_utilities.py

python /home/cranechu/.vscode/extensions/
ms-python.python-2020.3.71659/
pythonFiles/testing_tools/run_adapter.py
discover pytest -- --rootdir /home/
cranechu/pynvme -s --cache-clear
--pciaddr=0000:3a:00.0
python /home/cranechu/.vscode/extensions/
ms-python.python-2020.3.71659/
pythonFiles/testing_tools/run_adapter.py
discover pytest -- --rootdir /home/
cranechu/pynvme -s --cache-clear
--pciaddr=0000:3a:00.0

Performance Gauge ×

1800
1200   2400
pynvme
600   3000
0   3600
MB/s
1359.08

300
200   400
IOPS
100   500
0   600
K/s
331.81

pynvme builds your own tests.

dc135853*   Python 3.7.6 64-bit   ⊗ 0 ⚠ 0   Run Tests   Ln 1, Col 1   Spaces: 4   UTF-8   LF   Python
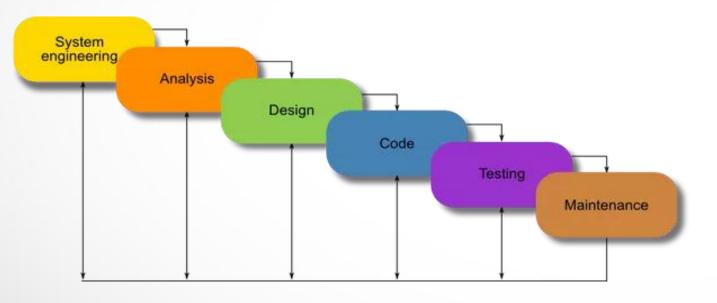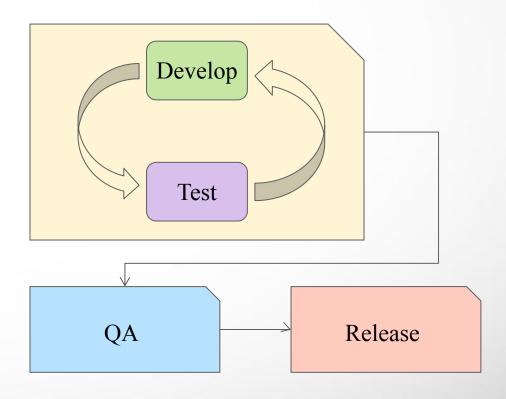
Test v.s. Quality Assurance

- Develop => QA
- Waterfall
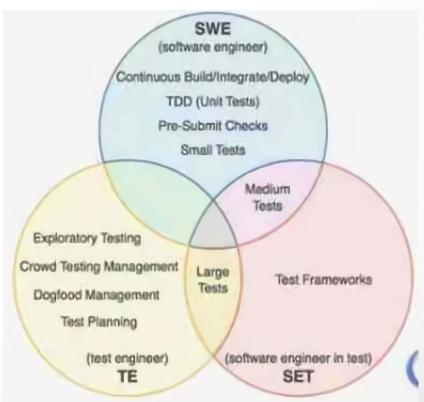
- (Develop+Test) => QA
- Agile

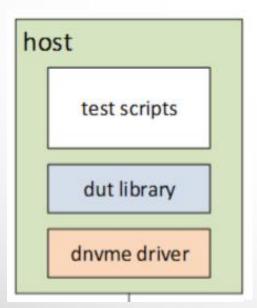# Experience of SSD Firmware Development

- Firmware Engineer & Test Engineer
  - FE develops firmware code
  - TE develops scripts to test firmware code

- test tool (DriveMaster):
  - difficult to develop and maintain
  - form a gap between FE and TE
  - source code is not open

- How Google Tests Software
  - SWE(FE), SET(TE), TE(QA)

SWE (software engineer)
Continuous Build/Integrate/Deploy
TDD (Unit Tests)
Pre-Submit Checks
Small Tests

Medium Tests

Exploratory Testing
Crowd Testing Management
Dogfood Management
Test Planning

Large Tests

Test Frameworks

(test engineer) TE

(software engineer in test) SET

Experience in a Startup: dnvme

- dnvme: @2015
  - wrap with python in userspace
  - functional tests
  - integrited with Jenkins
  - firmware engineers develop scripts
  - PASS IOL test on the first try

- challenges:
  - low performance:
    - IOPS, latency, consistency
    - test efficiency
    - stress tests
  - maintainness: kernel module
  - function coverage: PRP, ...
  - GPL License
- dnvme is good at NVMe test, but in-efficient on SSD test.

Born for NVMe: SPDK
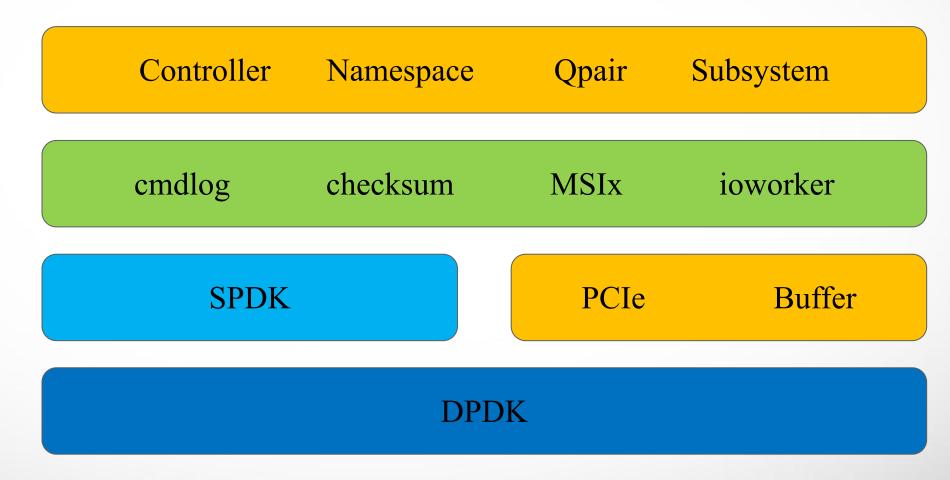
- SPDK is open from 2016
  - super high performance: 10M IOPS
  - user space application
  - based on DPDK environment, which abstracts low-level resources
    - PCIe
    - memory
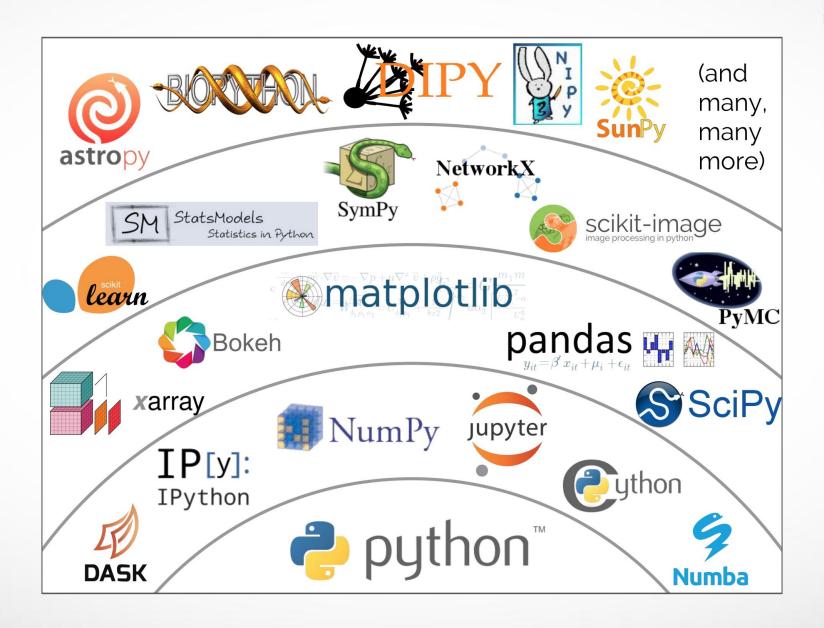
- meet challenges:
  - low performance:
    - IOPS, latency, consistency
    - test efficiency
    - stress tests
  - maintainness: kernel module
  - function coverage: PRP, ...
  - BSD License

pynvme: a software-defined SSD test framework

- pynvme abstracts test resources from low to high level, and provides Python API to access all these resources.

Controller	Namespace	Qpair	Subsystem

cmdlog	checksum	MSIx	ioworker

SPDK		PCIe	Buffer

DPDK
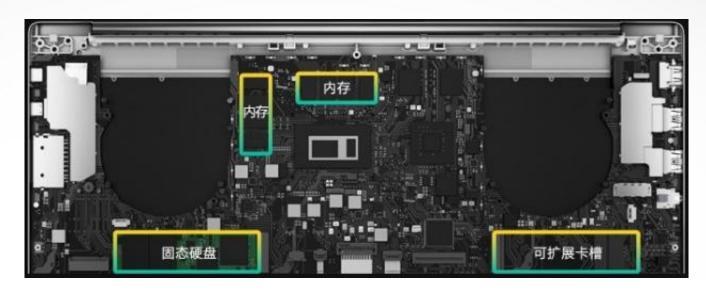
Open to the Ecosystem of Python

Flexible Hardware Configuration

- Single Node
  - laptop
  - workstation
- Mass Deploy
  - server

Test Scripts

```python
1   import time
2   import pytest
3   import logging
4
5   import nvme as d
6
7
8   # intuitive, spec, qpair, vscode, debug, cmdlog, assert
9   def test_hello_world(nvme0, nvme0n1, qpair):
10      # prepare data buffer and IO queue
11      read_buf = d.Buffer(512)
12      write_buf = d.Buffer(512)
13      write_buf[10:21] = b'hello world'
14
15      # send write and read command
16      def write_cb(cdw0, status1):  # command callback function
17          nvme0n1.read(qpair, read_buf, 0, 1)
18      nvme0n1.write(qpair, write_buf, 0, 1, cb=write_cb)
19
20      # wait commands complete and verify data
21      assert read_buf[10:21] != b'hello world'
22      qpair.waitdone(2)
23      assert read_buf[10:21] == b'hello world'
```

```python
8   def test_quarch_dirty_power_cycle_single(nvme0, nvme0n1, subsystem, buf, verify):
9       # get the unsafe shutdown count before test
10      nvme0.getlogpage(2, buf, 512).waitdone()
11      orig_unsafe_count = buf.data(159, 144)
12      logging.info("unsafe shutdowns: %d" % orig_unsafe_count)
13      assert verify == True
14
15      # 128K random write
16      cmdlog_list = [None]*1000
17      with nvme0n1.ioworker(io_size=256,
18                            lba_random=True,
19                            read_percentage=30,
20                            region_end=256*1000*1000,
21                            time=30,
22                            qdepth=1024,
23                            output_cmdlog_list=cmdlog_list):
24          # sudden power loss before the ioworker end
25          time.sleep(10)
26          subsystem.poweroff()
27
28      # power on and reset controller
29      time.sleep(5)
30      subsystem.poweron()
31      time.sleep(0)
32      nvme0.reset()
```

Test Scripts: dirty power cycle

- [PCIe Card Module](#)
- [Torridon Interface Kit](#)

- poweroff process
  - poweroff when ioworker is alive
  - remove device from system

- poweron process
  - poweron
  - remove kernel driver
  - rescan device
  - nvme initialization

Test Scripts: 3 ways of sending IO

```
8    # intuitive, spec, qpair, vscode, debug, cmdlog, assert
9    def test_hello_world(nvme0, nvme0n1, qpair):
10       # prepare data buffer and IO queue
11       read_buf = d.Buffer(512)
12       write_buf = d.Buffer(512)
13       write_buf[10:21] = b'hello world'
14
15       # send write and read command
16       def write_cb(cdw0, status1):  # command callback function
17           nvme0n1.read(qpair, read_buf, 0, 1)
18       nvme0n1.write(qpair, write_buf, 0, 1, cb=write_cb)
```

```
def test_ioworker_simplified(nvme0n1):
    nvme0n1.ioworker(io_size=2, time=2).start().close()
```

```
def test_send_single_cmd(nvme0):
    cq = IOCQ(nvme0, 1, 10, PRP())
    sq = IOSQ(nvme0, 1, 10, PRP(), cqid=1)

    # first cmd, invalid namespace
    sq[0] = [8] + [0]*15
    sq.tail = 1
    time.sleep(0.1)
    status = (cq[0][3]>>17)&0x7ff
    assert status == 0x000b

    sq.delete()
    cq.delete()
```

Test Scripts: ioworker

Input Parameters:

- lba_start, lba_step, lba_align
- lba_random
- region_start, region_end
- distribution

- io_size (int, range, list, dict)
- read_percentage, io_percentage
- time, io_count
- qdepth, qprio
- pvalue, ptype
- iops
- ...

Output Parameters:

- io_count_read
- io_count_write
- mseconds
- latency_max_us
- latency_average_us
- error
- cpu_usage

- output_io_per_second (optional)
- output_percentile_latency (optional)
- output_cmdlog_list (optional)
- ...

API document: https://pynvme.readthedocs.io/api.html

Test Scripts: multiprocessing

```python
72  def test_ioworker_with_temperature_and_trim(nvme0, nvme0n1):
73      # start trim process
74      import multiprocessing
75      mp = multiprocessing.get_context("spawn")
76      p = mp.Process(target = subprocess_trim,
77                     args = (nvme0.addr.encode('utf-8'),
78                            300000))
79      p.start()
80
81      # start read/write ioworker and admin commands
82      smart_log = d.Buffer(512, "smart log")
83      with nvme0n1.ioworker(io_size=8, lba_align=16,
84                            lba_random=True, qdepth=16,
85                            read_percentage=67, iops=10000, time=10):
86          for i in range(15):
87              nvme0.getlogpage(0x02, smart_log, 512).waitdone()
88              ktemp = smart_log.data(2, 1)
89
90              from pytemperature import k2c
91              logging.info("temperature: %0.2f degreeC" % k2c(ktemp))
92              time.sleep(1)
93
94      # wait trim process complete
95      p.join()
96
```
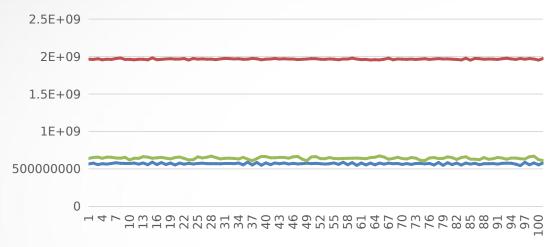
```python
60  # ioworker with admin commands, multiprocessing, log, cmdlog, pythonic
61  def subprocess_trim(pciaddr, loops):
62      nvme0 = d.Controller(pciaddr)
63      nvme0n1 = d.Namespace(nvme0)
64      q = d.Qpair(nvme0, 8)
65      buf = d.Buffer(4096)
66      buf.set_dsm_range(0, 8, 8)
67
68      # send trim commands
69      for i in range(loops):
70          nvme0n1.dsm(q, buf, 1).waitdone()
71
```

Test Scripts: gallery
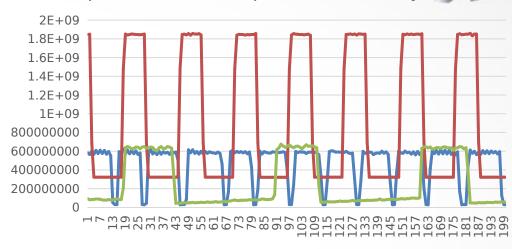
sequential write 1st pass (x: second, y: B/s)

sequential write 3rd pass (x: second, y: B/s)

4K random write (x: second, y: IOPS)

latency against IOPS, 2R1W (x: IOPS, y: us)
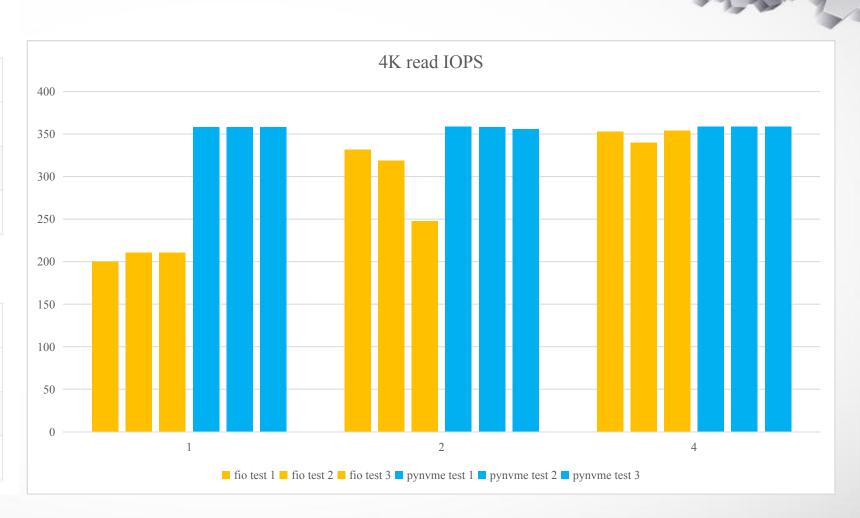
Performance: IOPS compared with FIO

fio (unit: K IOPS)
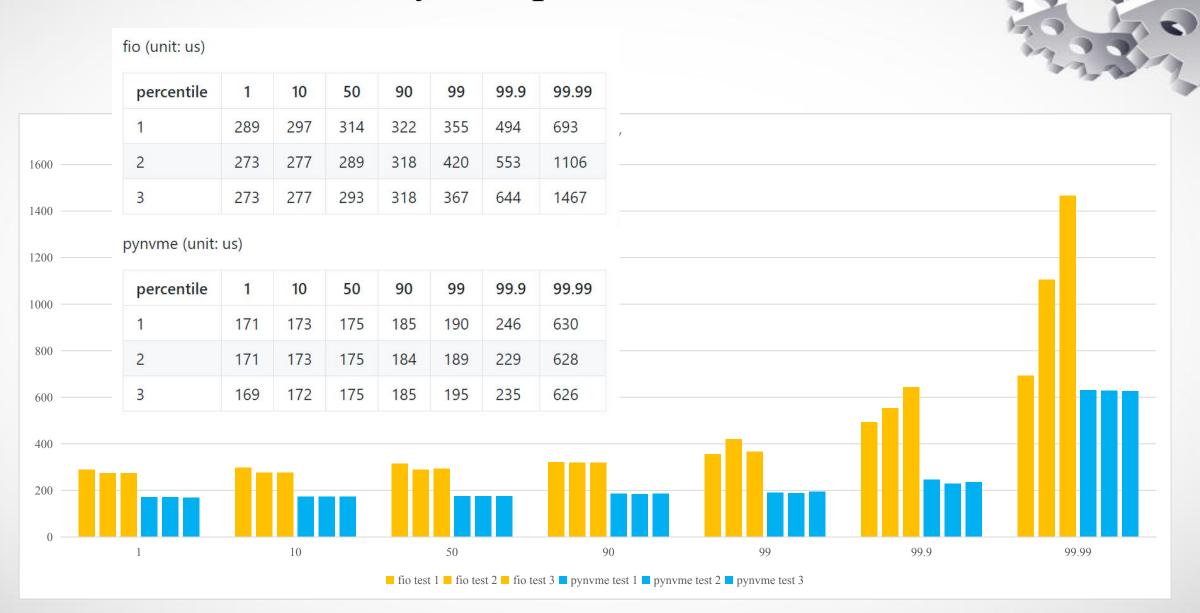
Q count	1	2	4
test 1	200	332	353
test 2	211	319	340
test 3	211	248	354

pynvme (unit: K IOPS)
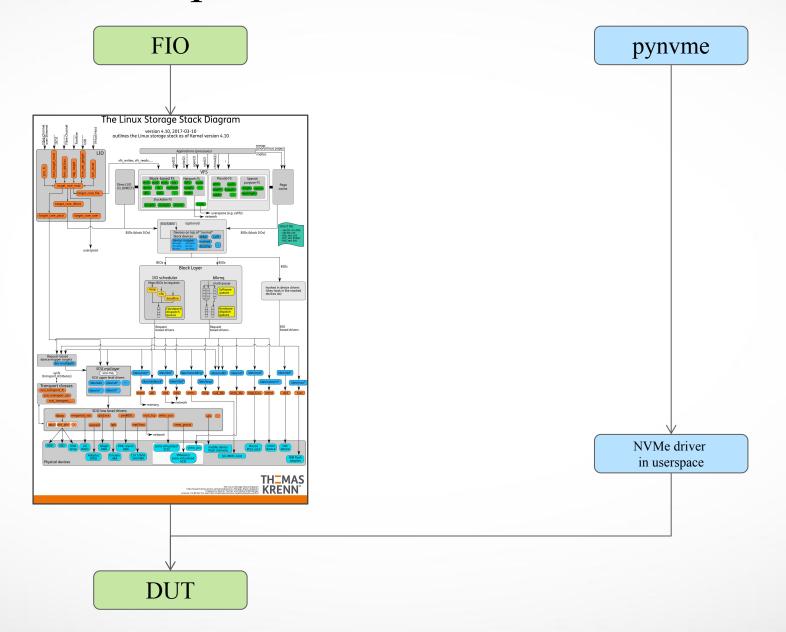
Q count	1	2	4
test 1	358	359	359
test 2	358	358	359
test 3	358	356	359

4K read IOPS

Performance: latency compared with FIO

fio (unit: us)

percentile	1	10	50	90	99	99.9	99.99
1	289	297	314	322	355	494	693
2	273	277	289	318	420	553	1106
3	273	277	293	318	367	644	1467

pynvme (unit: us)

percentile	1	10	50	90	99	99.9	99.99
1	171	173	175	185	190	246	630
2	171	173	175	184	189	229	628
3	169	172	175	185	195	235	626
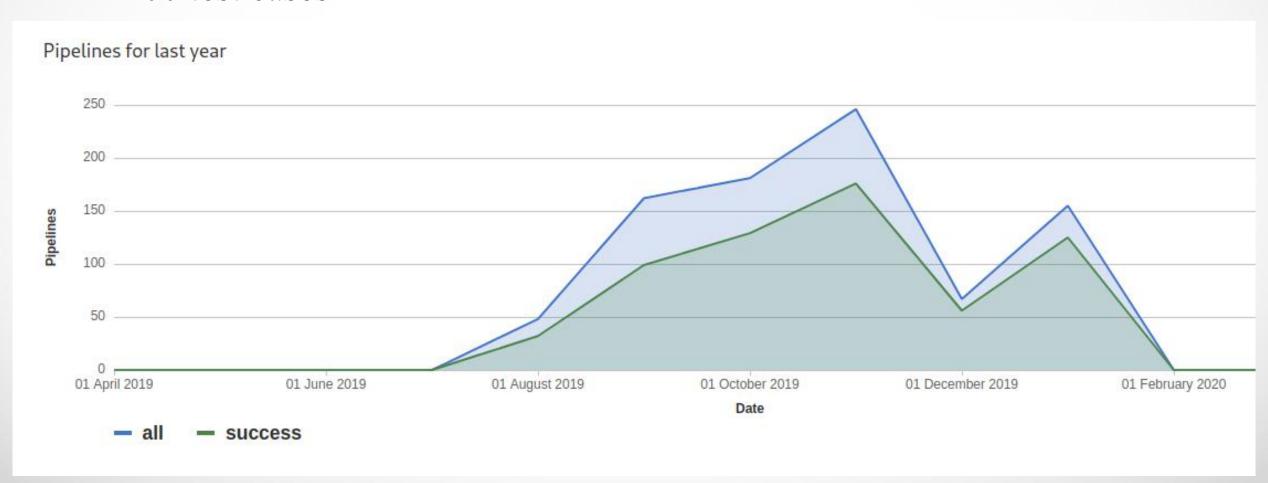
Performance: userspace driver
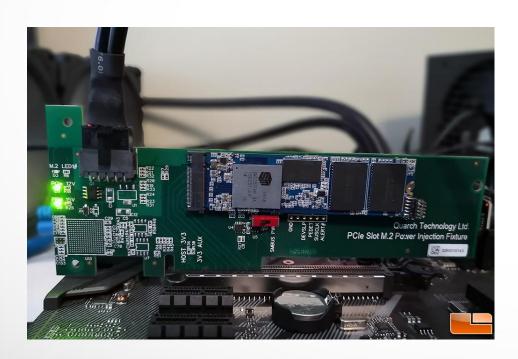
Quality: test of the test infrastructure

- CI in gitlab.com
- ~200 test cases

Pipelines for last year

vPower to ON and OFF
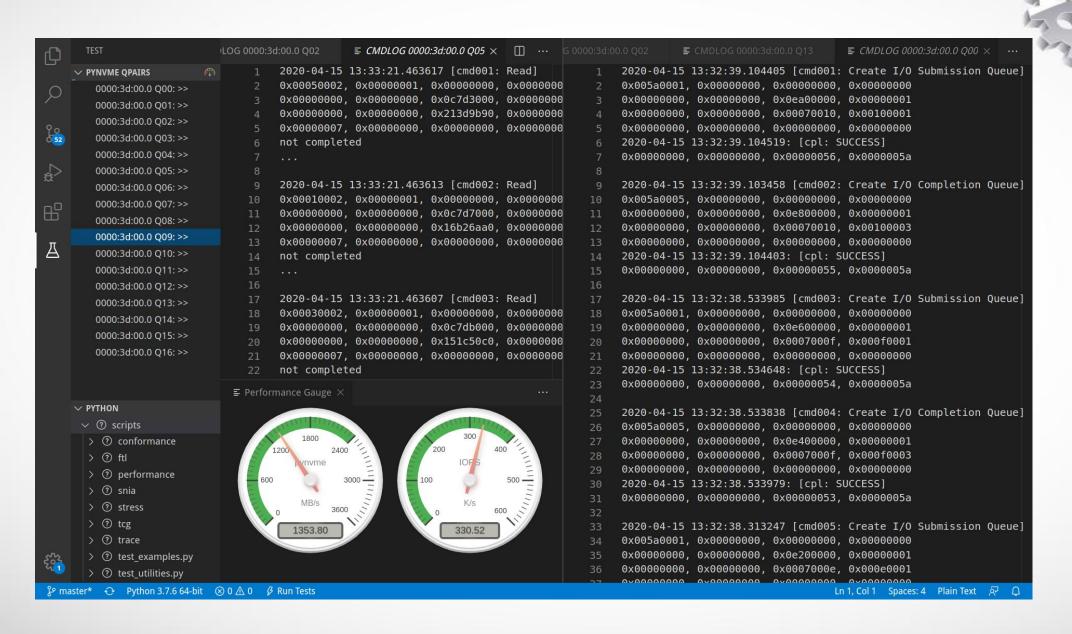
- Usually we need a special hardware box to control the power of SSD
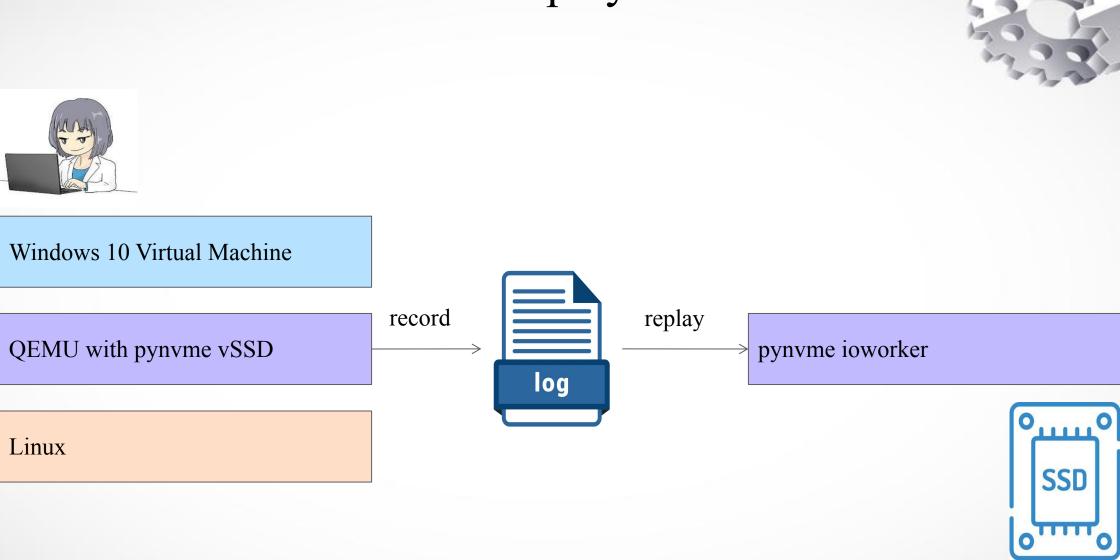
- vPower contorls the power of SSD:
  – power off: enter S3/sleep mode
  – power on: wake up by RTC

vAnalyzer

vTracer: IO Recorder and Replayer

Windows 10 Virtual Machine

QEMU with pynvme vSSD

Linux

record

log

replay

pynvme ioworker

SSD
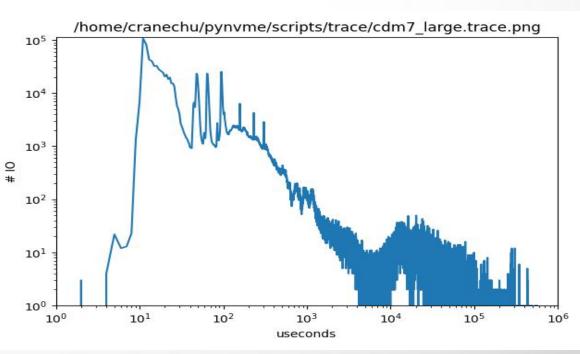
vTracer: Demo
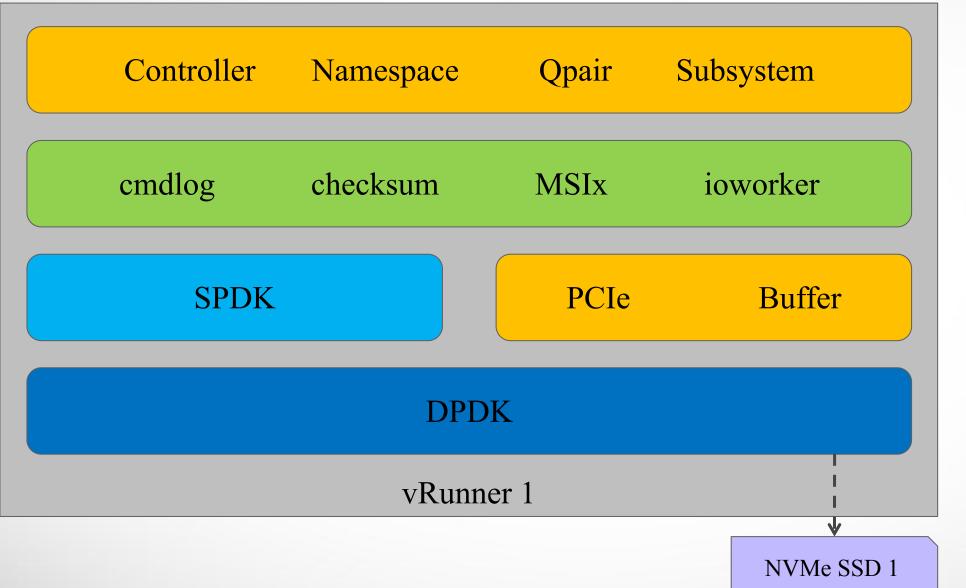
vRunner: Mass Deploy

Services
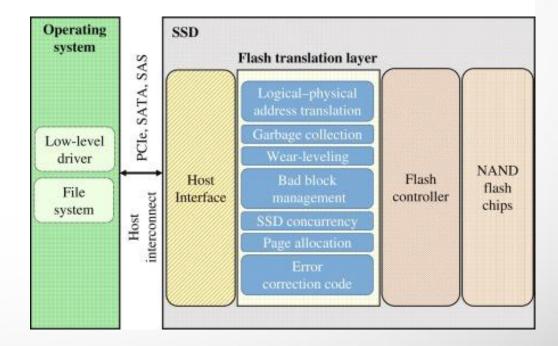
```
5   def test_ioworker_jedec_workload(nvme0n1):
6       distribution = [1000]*5 + [200]*15 + [25]*80
7       iosz_distribution = {1: 4,
8                            2: 1,
9                            3: 1,
10                           4: 1,
11                           5: 1,
12                           6: 1,
13                           7: 1,
14                           8: 67,
15                           16: 10,
16                           32: 7,
17                           64: 3,
18                           128: 3}
19
20       nvme0n1.ioworker(io_size=iosz_distribution,
21                        lba_random=True,
22                        qdepth=128,
23                        distribution = distribution,
24                        read_percentage=0,
25                        ptype=0xbeef, pvalue=100,
26                        time=12*3600).start().close()
```
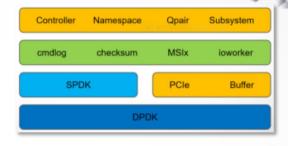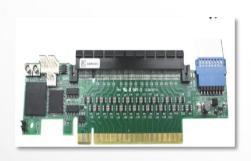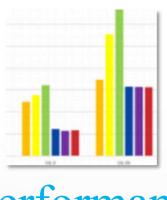
Values

Ecosystem

Scripts

Expendability

Hardware

Performance

Service

pynvme builds your own tests.

https://github.com/pynvme/pynvme

Thanks!