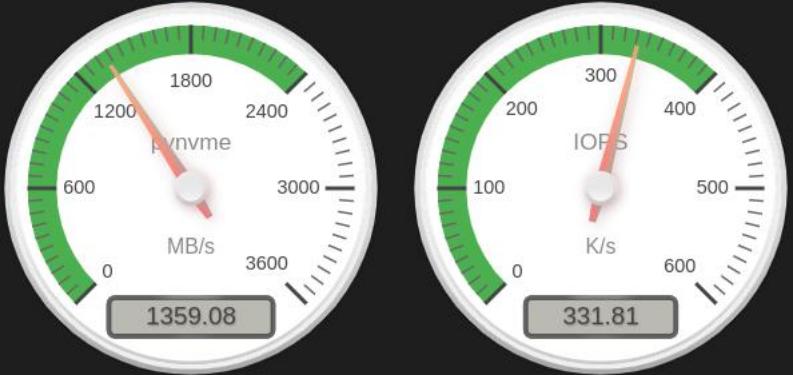


TEST 00:3d:00.0 Q02 test\_examples.py ● write\_uncorrectable\_test.py × ⏪ ⏴ ⏵ ...

PYNVME QPAIRS  
0000:3d:00.0 Q00: >> 1 import time  
00 CAZ-82512-Q11 NVMe LITEON 512GB 2 import pytest  
0000:3d:00.0 Q02: >> 3 import logging  
0000:3d:00.0 Q03: >> 4  
0000:3d:00.0 Q04: >> 5 from nvme import Controller, Namespace, Buffer, Qpair, Pcie, Subsyst  
0000:3d:00.0 Q05: >> 6 from scripts.psd import IOCQ, IOSQ, PRP, PRPList, SQE, CQE  
0000:3d:00.0 Q06: >> 7  
0000:3d:00.0 Q07: >> 8  
0000:3d:00.0 Q08: >> 9 # TODO: lba\_start=1, lba\_step=3, lba\_count=3  
0000:3d:00.0 Q09: >> 10  
0000:3d:00.0 Q10: >> 11 Run Test | Debug Test  
0000:3d:00.0 Q11: >> 12 def test\_write\_uncorrectable\_large\_lba(nvme0, nvme0n1, buf):  
0000:3d:00.0 Q12: >> 13 ncap = nvme0n1.id\_data(15, 8)  
0000:3d:00.0 Q13: >> 14  
0000:3d:00.0 Q14: >> 15 qpair = Qpair(nvme0, 16)  
0000:3d:00.0 Q15: >> 16 nvme0n1.write\_uncorrectable(qpair, ncap-1).waitdone()  
0000:3d:00.0 Q16: >> 17 with pytest.warns(UserWarning, match="ERROR status: 00/80"):  
0000:3d:00.0 Q17: >> 18 nvme0n1.write\_uncorrectable(qpair, ncap).waitdone()  
0000:3d:00.0 Q18: >> 19 with pytest.warns(UserWarning, match="ERROR status: 00/80"):  
0000:3d:00.0 Q19: >> 20 nvme0n1.write\_uncorrectable(qpair, ncap+1).waitdone()  
0000:3d:00.0 Q20: >> 21 with pytest.warns(UserWarning, match="ERROR status: 00/80"):  
  
Performance Gauge × ...  


NVMe: MB/s 1359.08  
IOPS: K/s 331.81

OUTPUT ... Python Test L ⌂ ⌂ ⌂ ⌂ ⌂

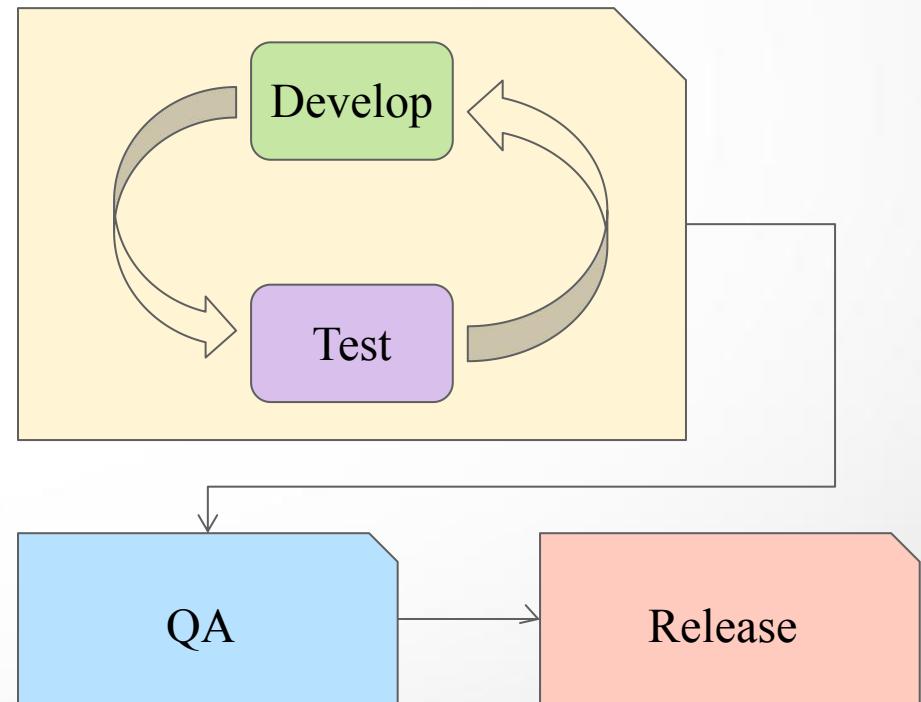
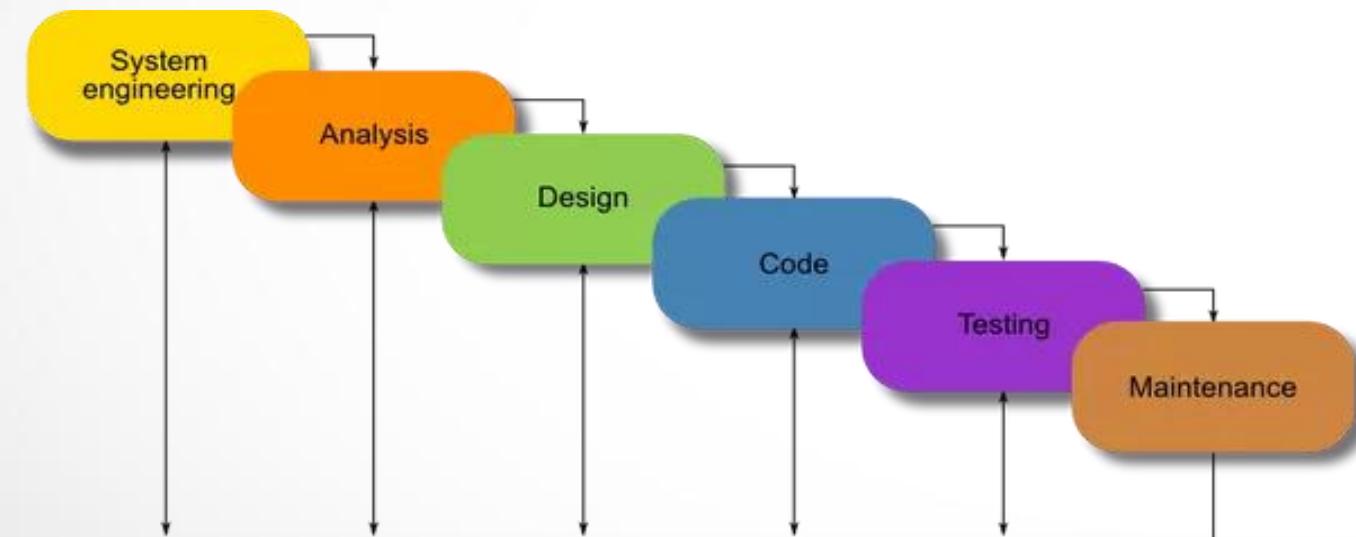
```
python /home/cranechu/.vscode/extensions/ms-python.python-2020.3.71659/pythonFiles/testing_tools/run_adapter.py discover pytest -- --rootdir /home/cranechu/pynvme -s --cache-clear --pciaddr=0000:3a:00.0  
python /home/cranechu/.vscode/extensions/ms-python.python-2020.3.71659/pythonFiles/testing_tools/run_adapter.py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

 pynvme builds your own tests.

# Test v.s. Quality Assurance



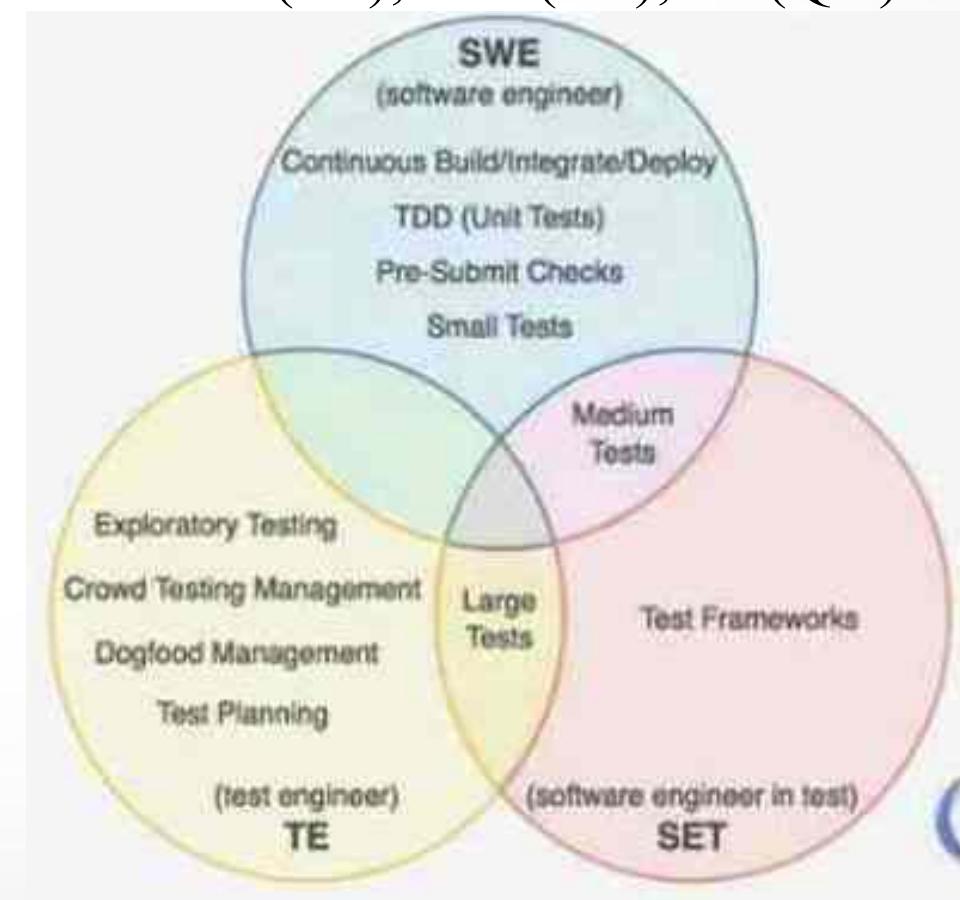
- Develop => QA
- Waterfall
- (Develop+Test) => QA
- Agile



# 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)



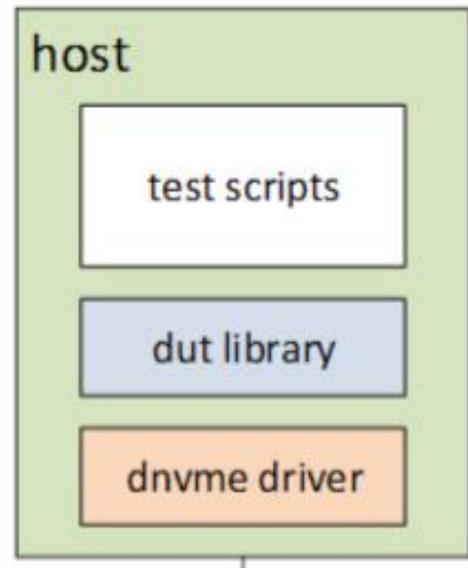
# Pair programming in CZ-5



# Experience in a Startup



- dnvme: @2015
  - wrap with python in userspace
  - functional tests
  - integrated 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, ...
- dnvme is good at NVMe test, but in-efficient on SSD test.

# Storage Performance Development Kit



- 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, ...



# 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

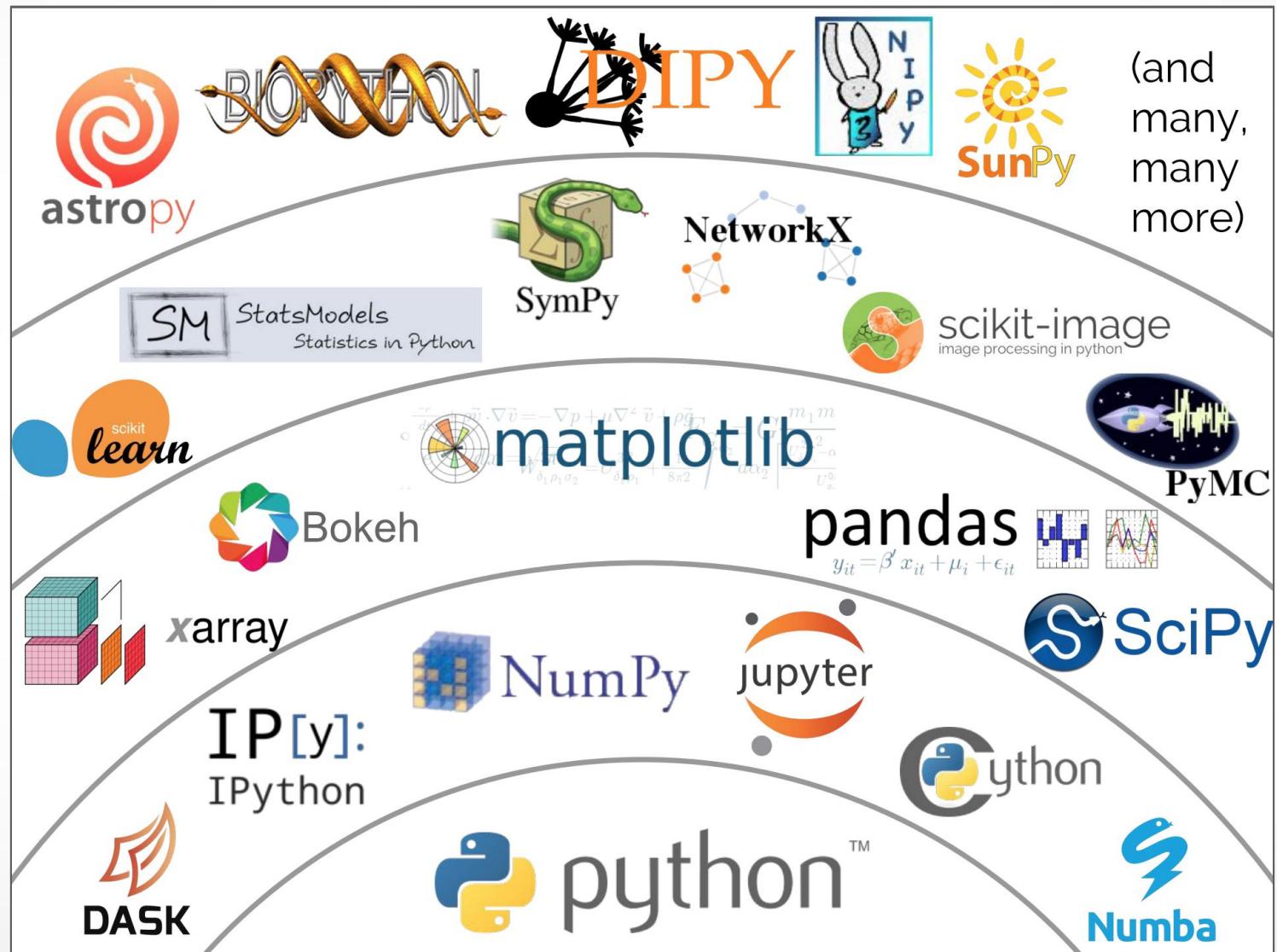
PCIe

Buffer

DPDK

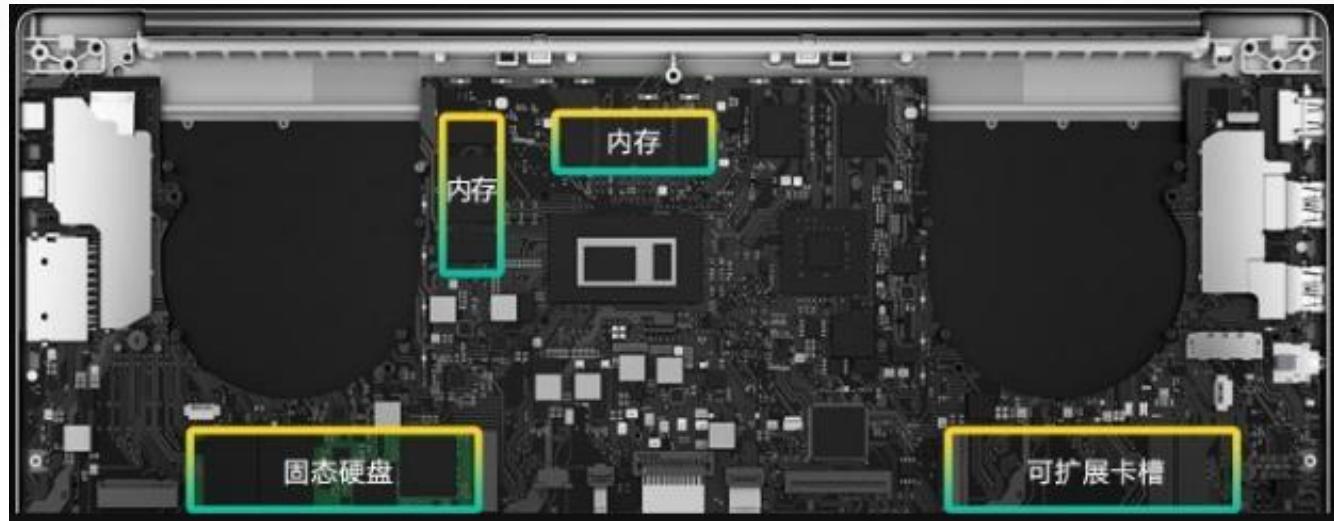


# Open to the Ecosystem of Python



# Flexible Hardware Configuration

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



# Test Scripts



```
1 import pytest
2 import nvme as d
3
4 import time
5 import logging
6
7 # intuitive, spec, qpair, vscode, debug, cmdlog, assert
8 def test_hello_world(nvme0, nvme0n1):
9     # prepare data buffer and IO queue
10    read_buf = d.Buffer(512)
11    write_buf = d.Buffer(512)
12    write_buf[10:21] = b'hello world'
13    qpair = d.Qpair(nvme0, 16) # create IO SQ/CQ pair, with 16 queue-depth
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'
```

```
1 import time
2 import pytest
3 import logging
4
5 import nvme as d
6
7 @pytest.mark.parametrize("repeat", range(10))
8 def test_quarch_dirty_power_cycle(nvme0, nvme0n1, subsystem, buf, verify, repeat):
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
14    # 128K sequential write
15    cmdlog_list = [None]*1000
16    with nvme0n1.ioworker(io_size=256, lba_random=False,
17                           read_percentage=0, lba_start=0,
18                           qdepth=1023, time=15,
19                           output_cmdlog_list=cmdlog_list):
20        # sudden power loss before the ioworker end
21        time.sleep(5)
22        subsystem.poweroff()
23
24    # power on
25    subsystem.poweron()
26    subsystem.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



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

```
def test_hello_world(nvme0, nvme0n1):
    # prepare data buffer and IO queue
    read_buf = d.Buffer(512)
    write_buf = d.Buffer(512)
    write_buf[10:21] = b'hello world'
    qpair = d.Qpair(nvme0, 16) # create IO SQ/CQ pair,
                                # 16 entries

    # send write and read command
    def write_cb(cdw0, status1): # command callback function
        nvme0n1.read(qpair, read_buf, 0, 1)
        nvme0n1.write(qpair, write_buf, 0, 1, cb=write_cb)

    # wait commands complete and verify data
    assert read_buf[10:21] != b'hello world'
    qpair.waitdone(2)
    assert read_buf[10:21] == b'hello world'
```

```
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



## Inputs:

- 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**
- ...

## Outputs:

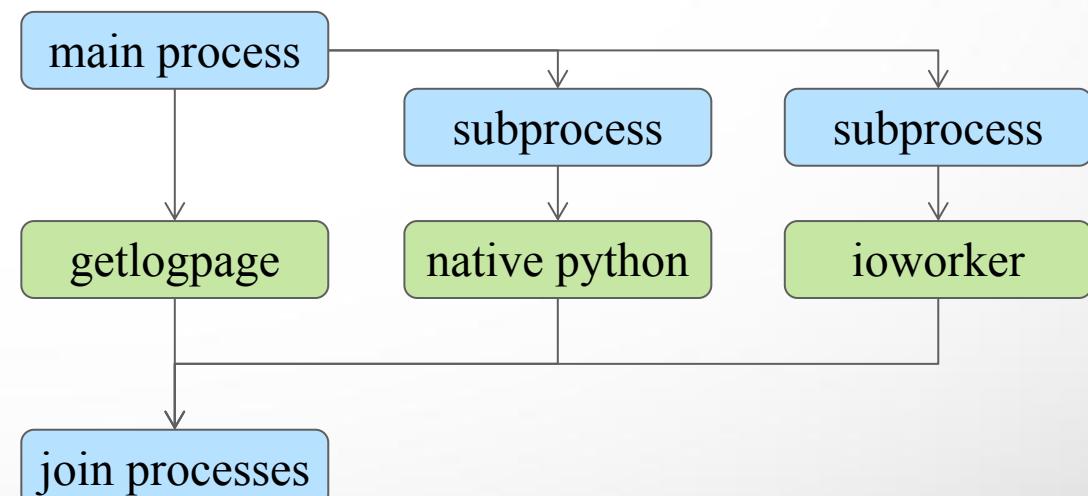
- io\_count\_read
- io\_count\_write
- mseconds
- latency\_max\_us
- latency\_average\_us
- error
- cpu\_usage
- output\_io\_per\_second (optional)
- output\_percentile\_latency (optional)
- output\_cmdlog\_list (optional)
- ...

# Test Scripts: multiprocessing



```
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
```

```
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: API document



A screenshot of a code editor showing a Python script named `test_examples.py`. The script is located in a directory structure under `PYNVME QPAIRS` and `PYTHON`. The code is annotated with comments explaining the functionality of various NVMe commands. A red box highlights a section of the code related to the `sanitize` command.

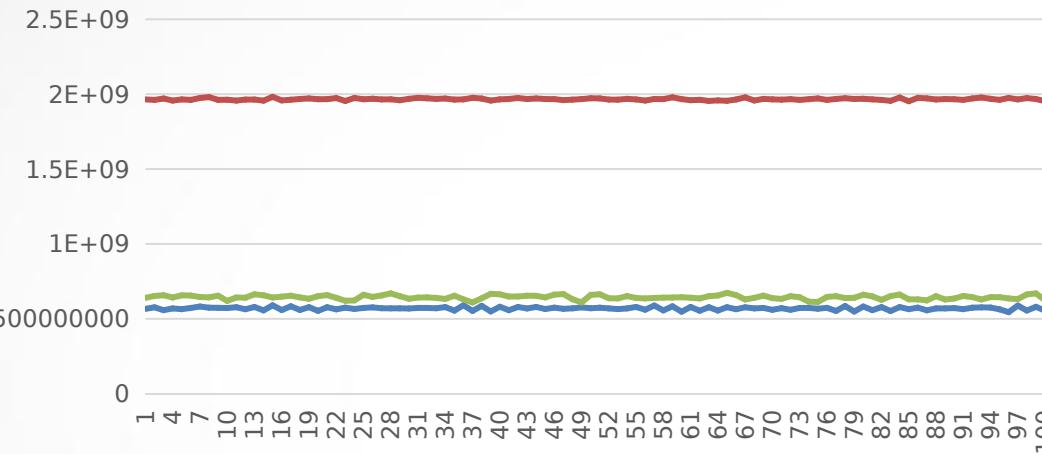
```
38 def test_sanitize(nvme0: d.Controller, buf):
39     if nvme0.id_data(331, 328) == 0:
40         pytest.skip("sanitize operation is not supported")
41
42     logging.info("supported sanitize operation: %d" % nvme0.id_data(331, 328))
43     sg.OneLineProgressMeter('sanitize progress', 0, 100, 'progress', orientation='vertical')
44     nvme0n1 = d.Namespace(nvme0)
45     nvme0.sanitize().waitdone() # sanitize clears namespace
46
47     # check
48     nvme0.Controller.sanitize(option=2, pattern=0, cb=None) # sanitize admin command
49     while not nvme0.Controller.sanitize(option=2, pattern=0, cb=None).is_completed():
50         time.sleep(0.1)
51
52     # Parameters
53     nvme0.sanitize(option=2, pattern=0, cb=None) # sanitize option field in the command
54     nvme0.sanitize(pattern=0x5aa5a55a, cb=None) # or overwrite method. Default: 0x5aa5a55a
55     nvme0.sanitize(cb=None) # callback function called at completion. Default: None
56
57 # simple i
58 def test_i
59     # Returns
60     nvme0n1.ioworker(time=1).start().close()
61     nvme0n1.ioworker(io_size=[1, 2, 3, 7, 8, 16], time=1).start().close()
62     nvme0n1.ioworker(op_percentage={2:10, 1:20, 0:30, 9:40}, time=1).start().close()
63     test_hello_world(nvme0, nvme0n1)
64
65 # ioworker with admin commands, multiprocessing, log, cmdlog, pythonic
66 def subprocess_trim(pciaddr, seconds):
67     nvme0 = d.Controller(pciaddr)
68     nvme0n1 = d.Namespace(nvme0)
69     q = d.Qpair(nvme0, 8)
70     buf = d.Buffer(4096)
71     buf.set_dsm_range(0, 8, 8)
72
```

The code editor interface includes a sidebar with project navigation, a terminal window showing a bash session, and a status bar at the bottom.

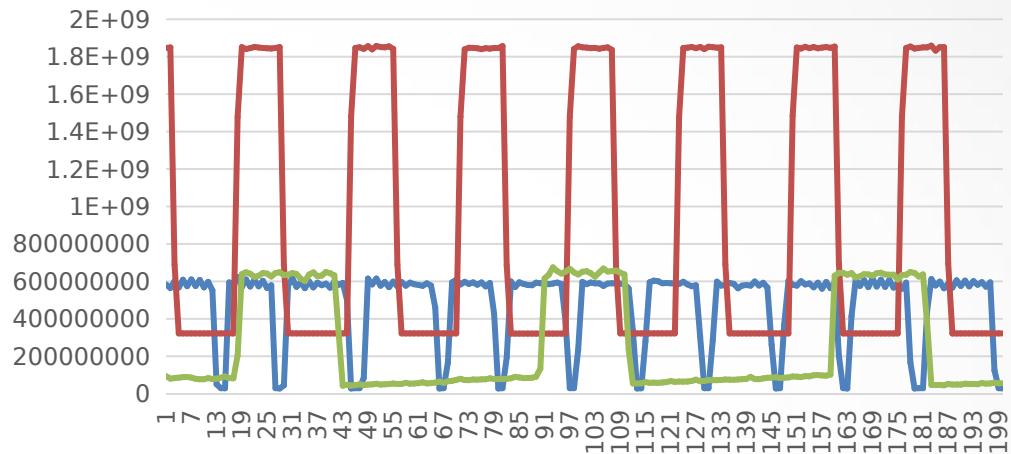
# Test Scripts: gallery



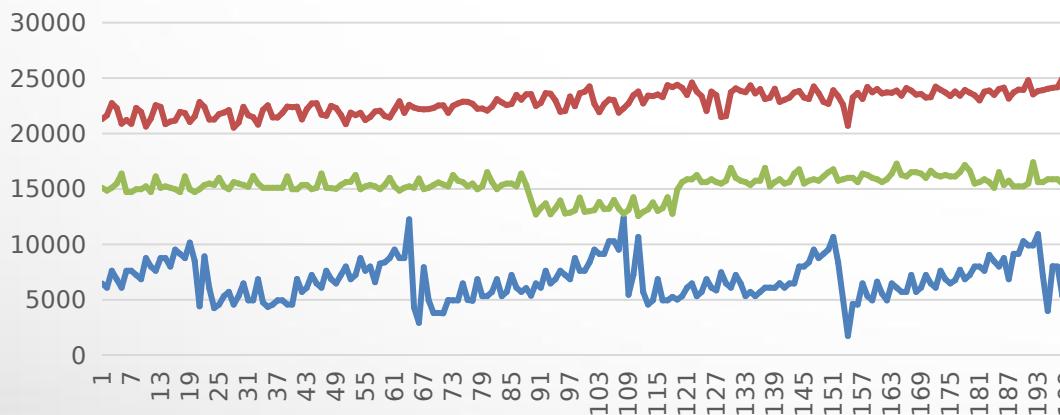
sequential write 1st pass (x: second, y: B/s)



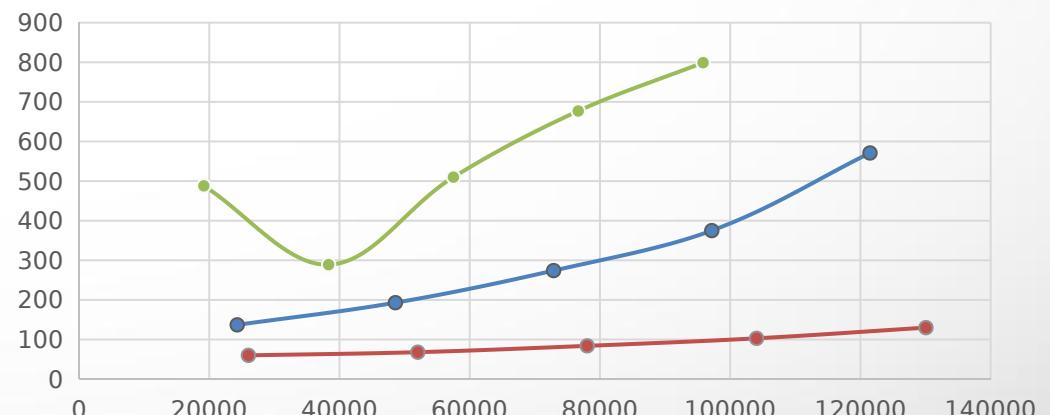
sequential write 3rd pass (x: second, y: B/s)



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



latency against IOPS, 2R1W (x: IOPS, y: us)



# Performance: IOPS compared with FIO

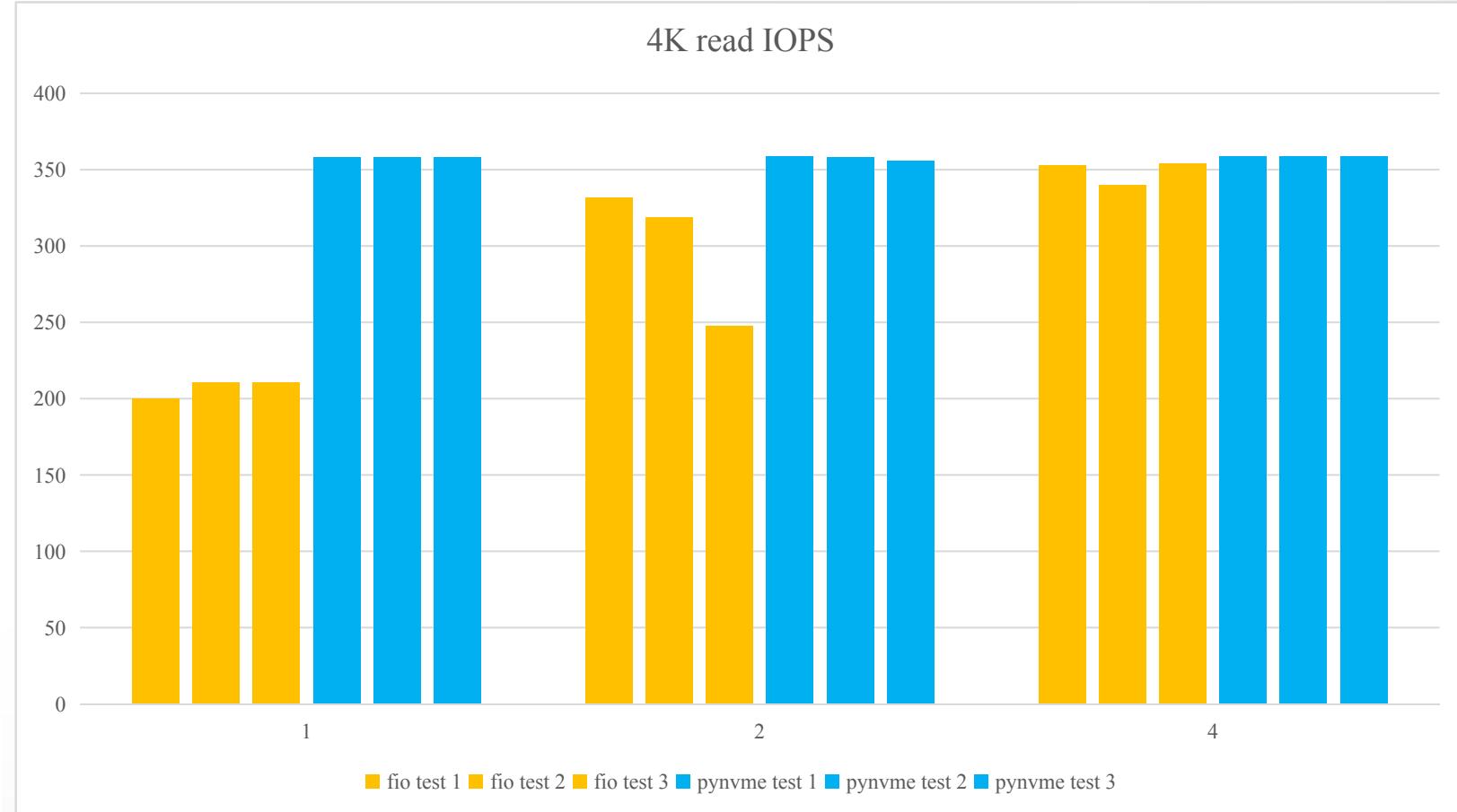


fio (unit: K IOPS)

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



# 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

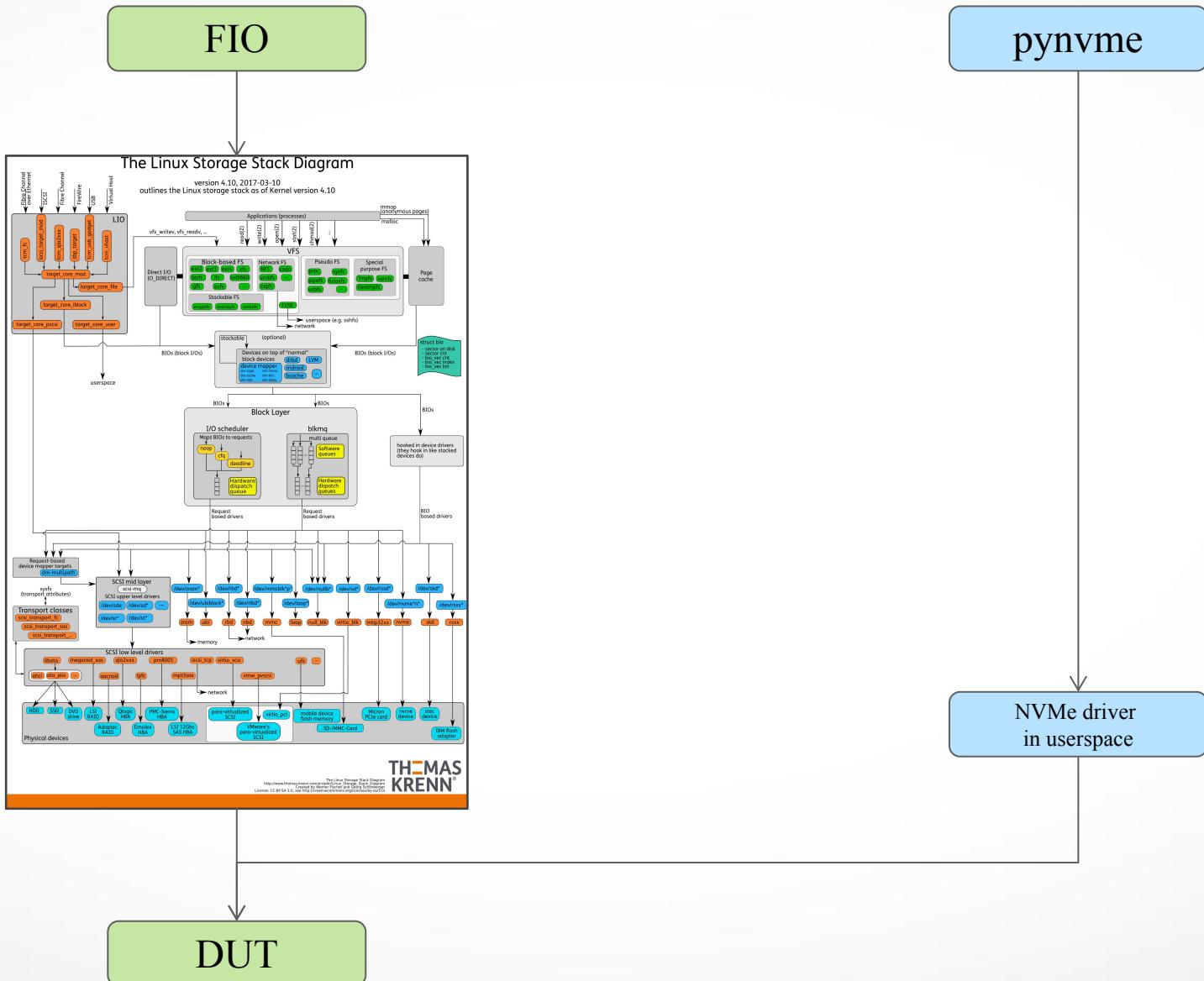
pynvme (unit: us)

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

■ fio test 1 ■ fio test 2 ■ fio test 3 ■ pynvme test 1 ■ pynvme test 2 ■ pynvme test 3

1 10 50 90 99 99.9 99.99

# Performance: userspace driver

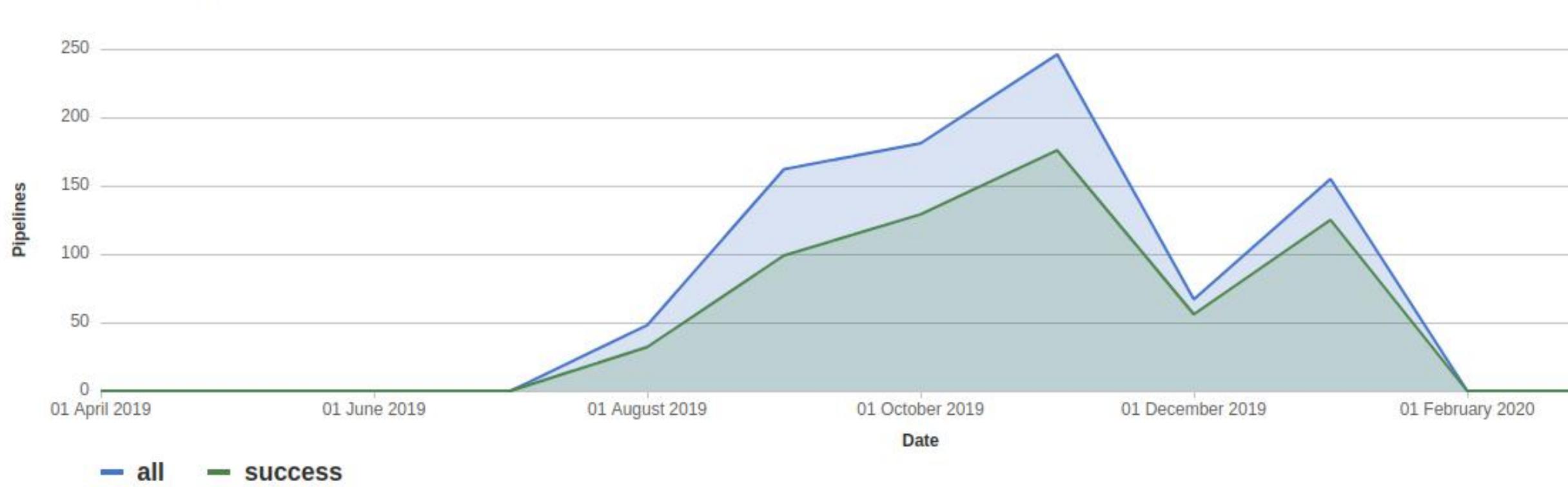


# Quality: test of the test infrastructure



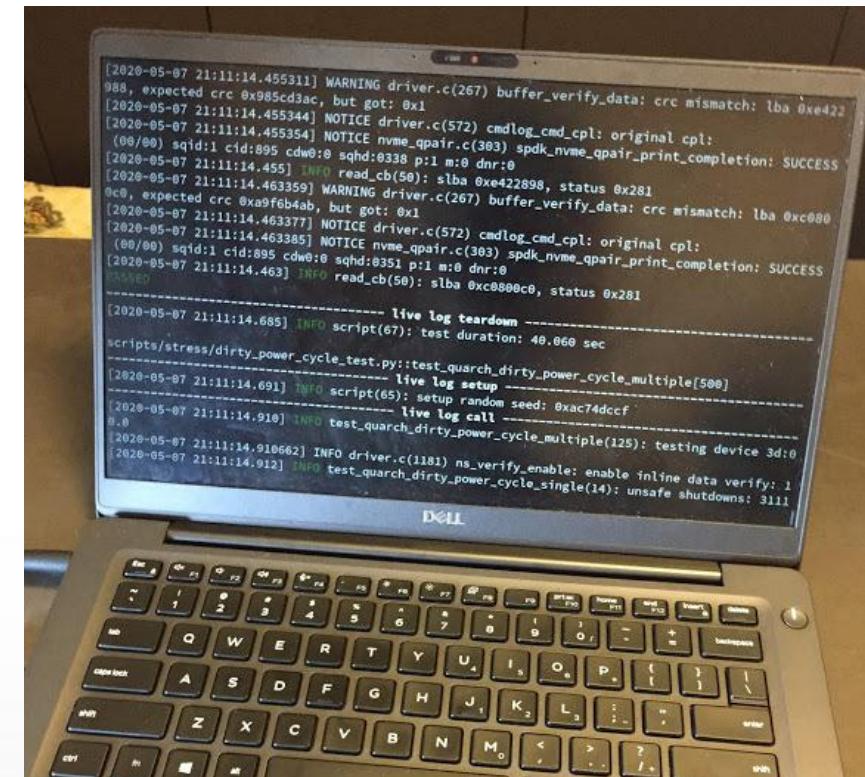
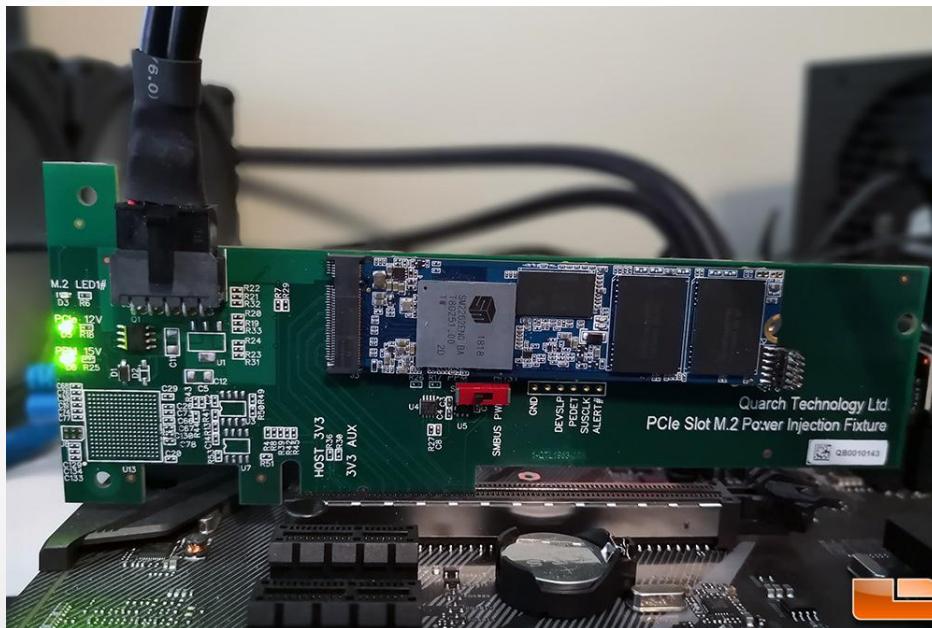
- CI in gitlab.com
- ~200 test cases

Pipelines for last year



# vPower to ON and OFF

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



# vAnalyzer

TEST

PYNVME QPAIRS

0000:3d:00.0 Q00: >>

0000:3d:00.0 Q01: >>

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: >>

0000:3d:00.0 Q07: >>

0000:3d:00.0 Q08: >>

0000:3d:00.0 Q09: >> **selected**

0000:3d:00.0 Q10: >>

0000:3d:00.0 Q11: >>

0000:3d:00.0 Q12: >>

0000:3d:00.0 Q13: >>

0000:3d:00.0 Q14: >>

0000:3d:00.0 Q15: >>

0000:3d:00.0 Q16: >>

PYTHON

scripts

conformance

ftl

performance

snia

stress

tcg

trace

test\_examples.py

test\_utilities.py

LOG 0000:3d:00.0 Q02

CMDLOG 0000:3d:00.0 Q05

G 0000:3d:00.0 Q02

CMDLOG 0000:3d:00.0 Q13

CMDLOG 0000:3d:00.0 Q00

Performance Gauge

1353.80 MB/s

330.52 K/s

1 2020-04-15 13:33:21.463617 [cmd001: Read]

2 0x00050002, 0x00000001, 0x00000000, 0x00000000

3 0x00000000, 0x00000000, 0x0c7d3000, 0x00000000

4 0x00000000, 0x00000000, 0x213d9b90, 0x00000000

5 0x00000007, 0x00000000, 0x00000000, 0x00000000

6 not completed

7 ...

9 2020-04-15 13:33:21.463613 [cmd002: Read]

10 0x00010002, 0x00000001, 0x00000000, 0x00000000

11 0x00000000, 0x00000000, 0x0c7d7000, 0x00000000

12 0x00000000, 0x00000000, 0x16b26aa0, 0x00000000

13 0x00000007, 0x00000000, 0x00000000, 0x00000000

14 not completed

15 ...

17 2020-04-15 13:33:21.463607 [cmd003: Read]

18 0x00030002, 0x00000001, 0x00000000, 0x00000000

19 0x00000000, 0x00000000, 0x0c7db000, 0x00000000

20 0x00000000, 0x00000000, 0x151c50c0, 0x00000000

21 0x00000007, 0x00000000, 0x00000000, 0x00000000

22 not completed

1 2020-04-15 13:32:39.104405 [cmd001: Create I/O Submission Queue]

2 0x005a0001, 0x00000000, 0x00000000, 0x00000000

3 0x00000000, 0x00000000, 0x0ea0000, 0x00000001

4 0x00000000, 0x00000000, 0x00070010, 0x00100001

5 0x00000000, 0x00000000, 0x00000000, 0x00000000

6 2020-04-15 13:32:39.104519: [cpl: SUCCESS]

7 0x00000000, 0x00000000, 0x00000056, 0x0000005a

8

9 2020-04-15 13:32:39.103458 [cmd002: Create I/O Completion Queue]

10 0x005a0005, 0x00000000, 0x00000000, 0x00000000

11 0x00000000, 0x00000000, 0x0e800000, 0x00000001

12 0x00000000, 0x00000000, 0x00070010, 0x00100003

13 0x00000000, 0x00000000, 0x00000000, 0x00000000

14 2020-04-15 13:32:39.104403: [cpl: SUCCESS]

15 0x00000000, 0x00000000, 0x00000055, 0x0000005a

16

17 2020-04-15 13:32:38.533985 [cmd003: Create I/O Submission Queue]

18 0x005a0001, 0x00000000, 0x00000000, 0x00000000

19 0x00000000, 0x00000000, 0x0e600000, 0x00000001

20 0x00000000, 0x00000000, 0x0007000f, 0x000f0001

21 0x00000000, 0x00000000, 0x00000000, 0x00000000

22 2020-04-15 13:32:38.534648: [cpl: SUCCESS]

23 0x00000000, 0x00000000, 0x00000054, 0x0000005a

24

25 2020-04-15 13:32:38.533838 [cmd004: Create I/O Completion Queue]

26 0x005a0005, 0x00000000, 0x00000000, 0x00000000

27 0x00000000, 0x00000000, 0x0e400000, 0x00000001

28 0x00000000, 0x00000000, 0x0007000f, 0x000f0003

29 0x00000000, 0x00000000, 0x00000000, 0x00000000

30 2020-04-15 13:32:38.533979: [cpl: SUCCESS]

31 0x00000000, 0x00000000, 0x00000053, 0x0000005a

32

33 2020-04-15 13:32:38.313247 [cmd005: Create I/O Submission Queue]

34 0x005a0001, 0x00000000, 0x00000000, 0x00000000

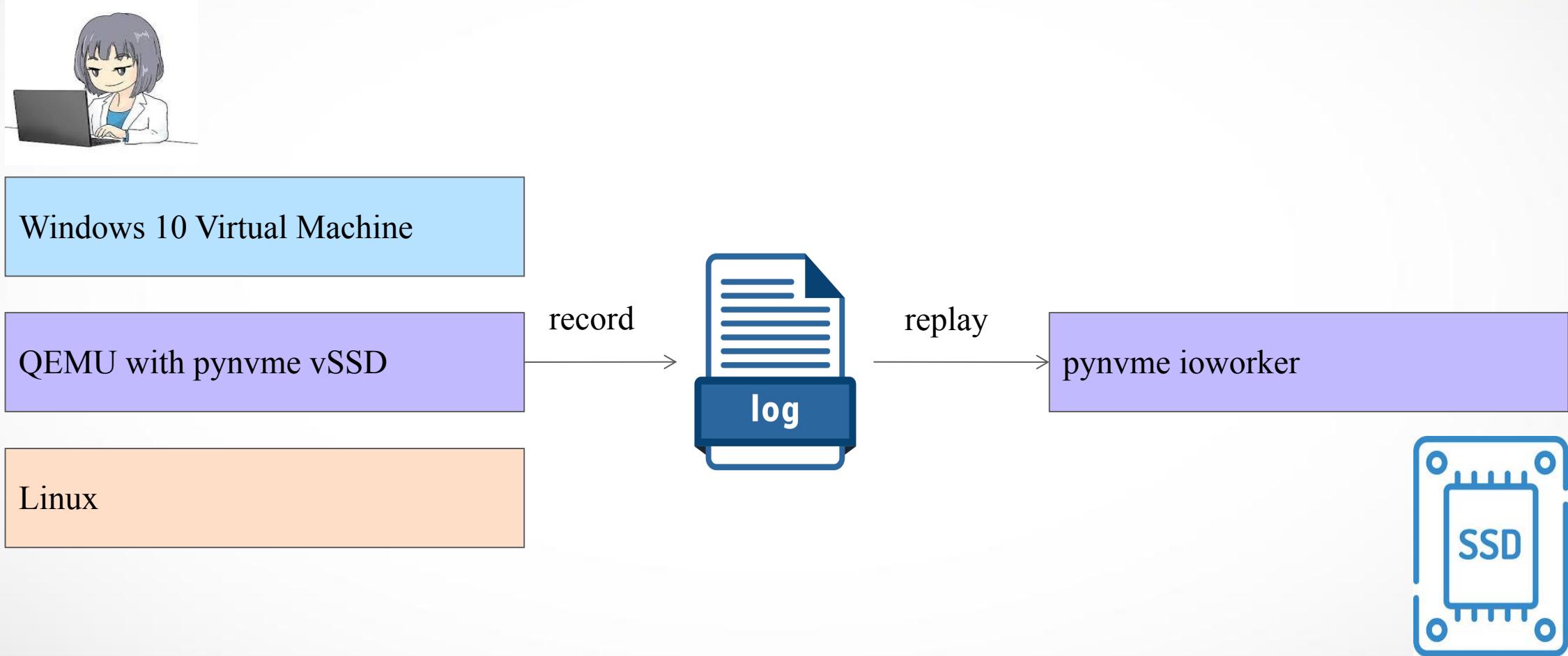
35 0x00000000, 0x00000000, 0x0e200000, 0x00000001

36 0x00000000, 0x00000000, 0x0007000e, 0x000e0001

37 0x00000000, 0x00000000, 0x00000000, 0x00000000

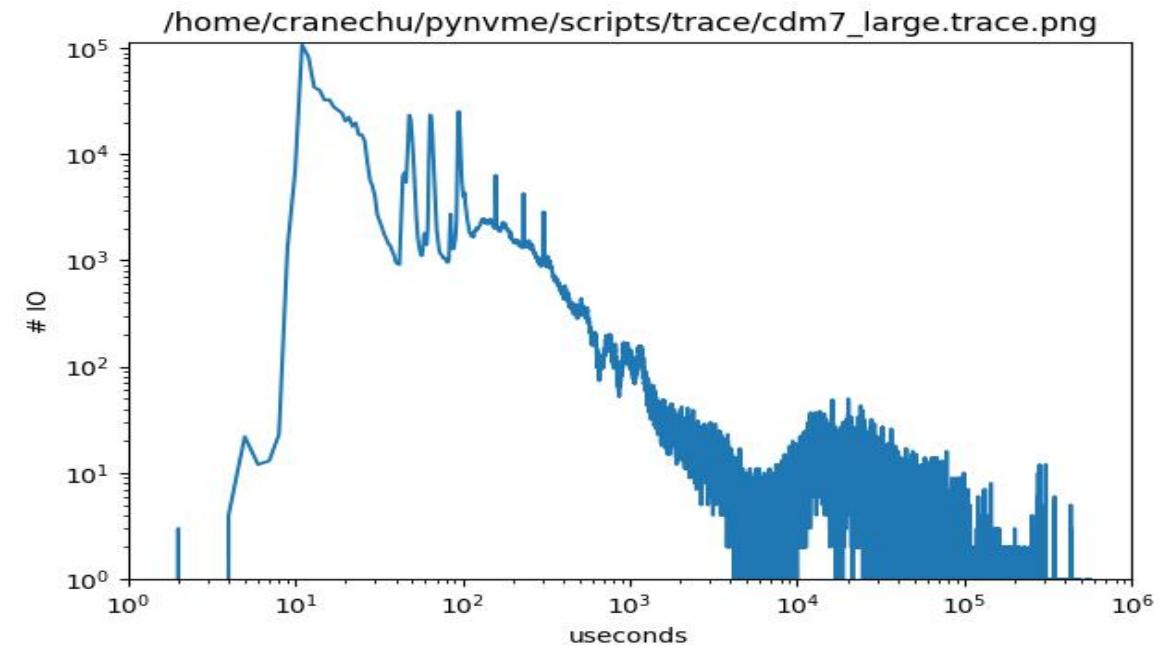
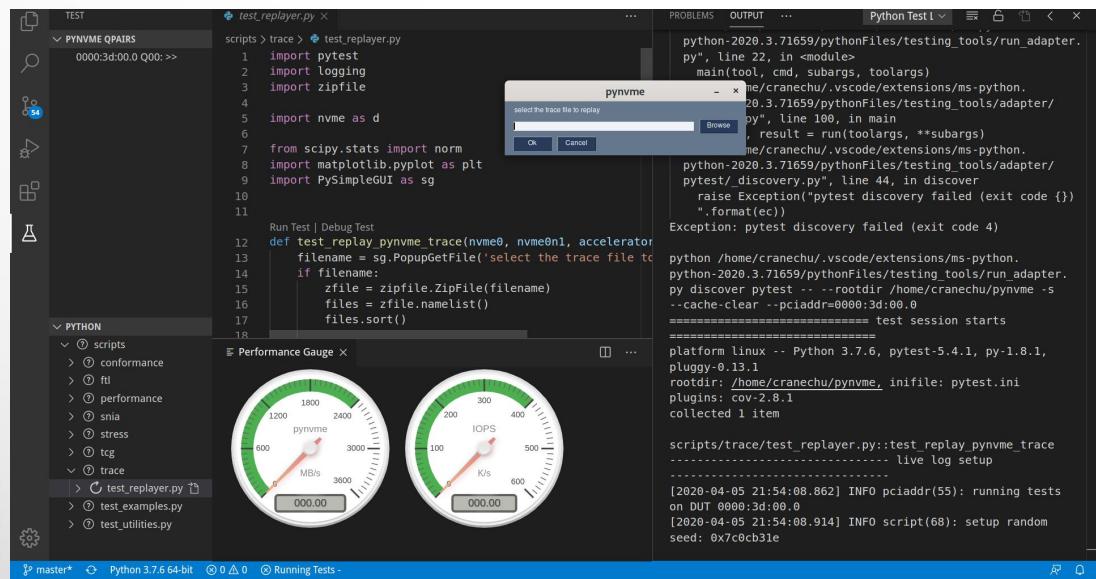
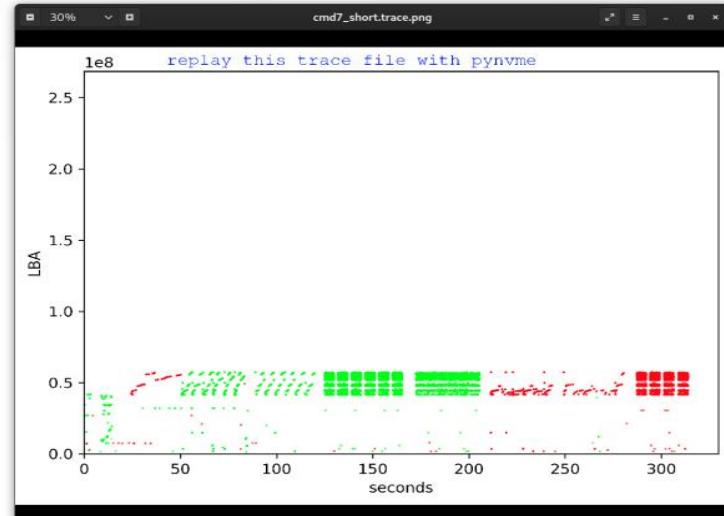
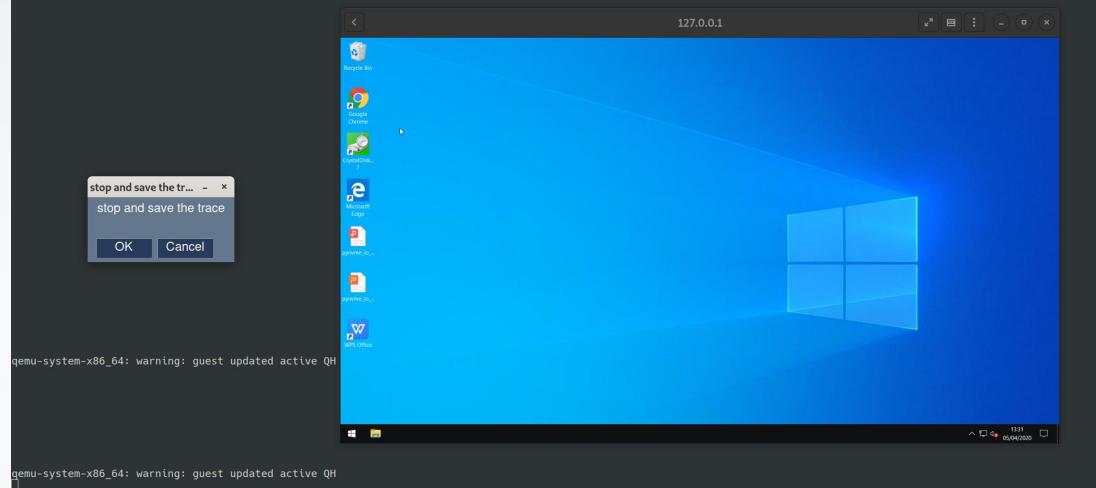
Ln 1, Col 1 Spaces: 4 Plain Text

# vTracer: IO Recorder and Replayer

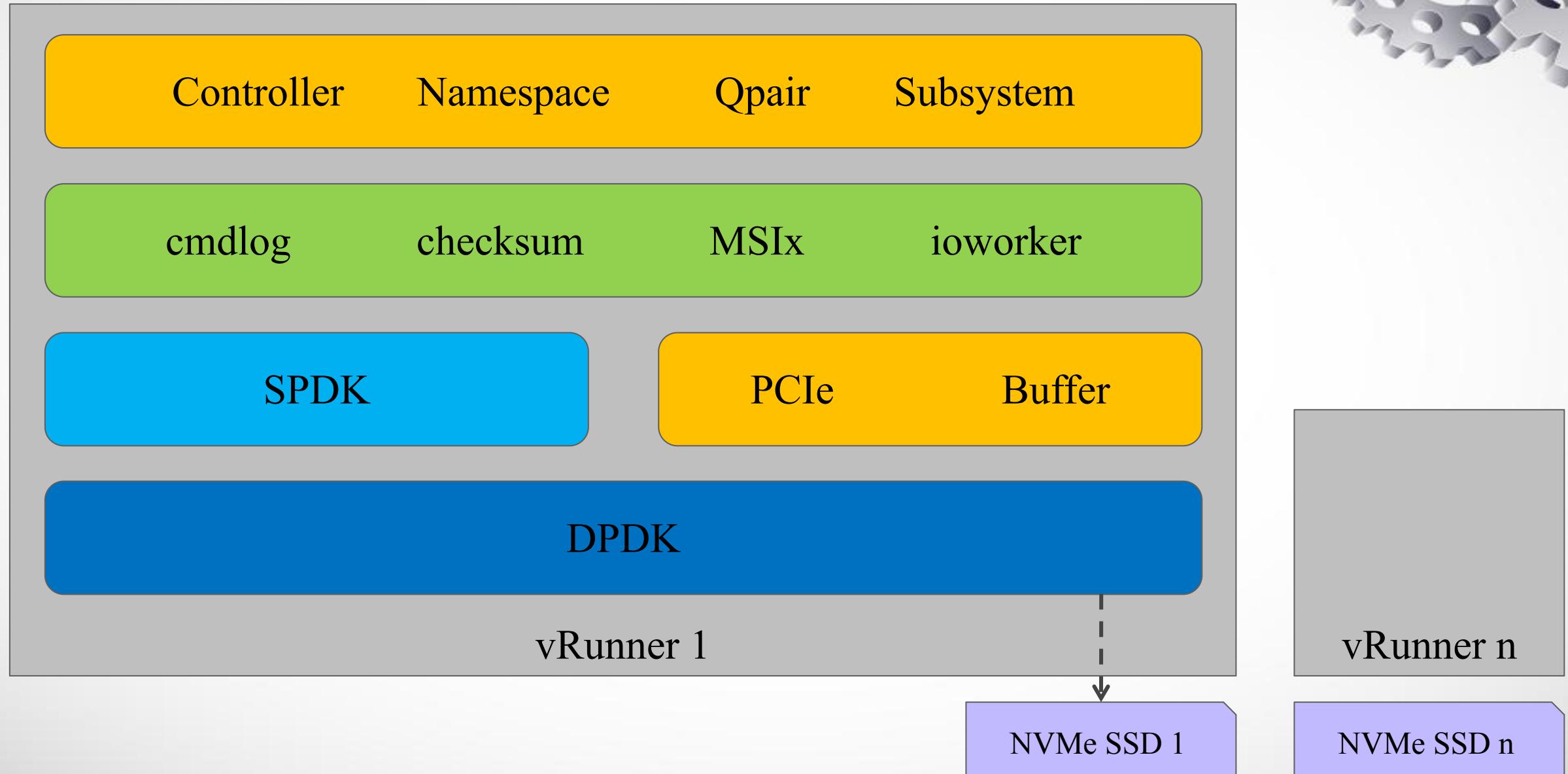


# vTracer: Demo

```
laptop:~/pyname$ ./pts/trace ~/home/cranechu/qemu/build/x86_64-safemu/qemu-system-x86_64 -m 4096 -drive file=/home/cranechu/vm/wln10.100.qcow2,lf=none,id=drv0,discard=on -device nvme,drive=drv0,serial=foo -device usb-ehci,id=ehci -device usb-tablet,bus=ehci.0 -enable-kvm -cpu host -smp 4,sockets=1,cores=4,threads=1 | ./recorder.py
VNC server running on ::1:5900
trace time base 0
create minute trace file /tmp/pyname_trace/0/1
create minute trace file /tmp/pyname_trace/0/2
create minute trace file /tmp/pyname_trace/0/3
create minute trace file /tmp/pyname_trace/0/4
```

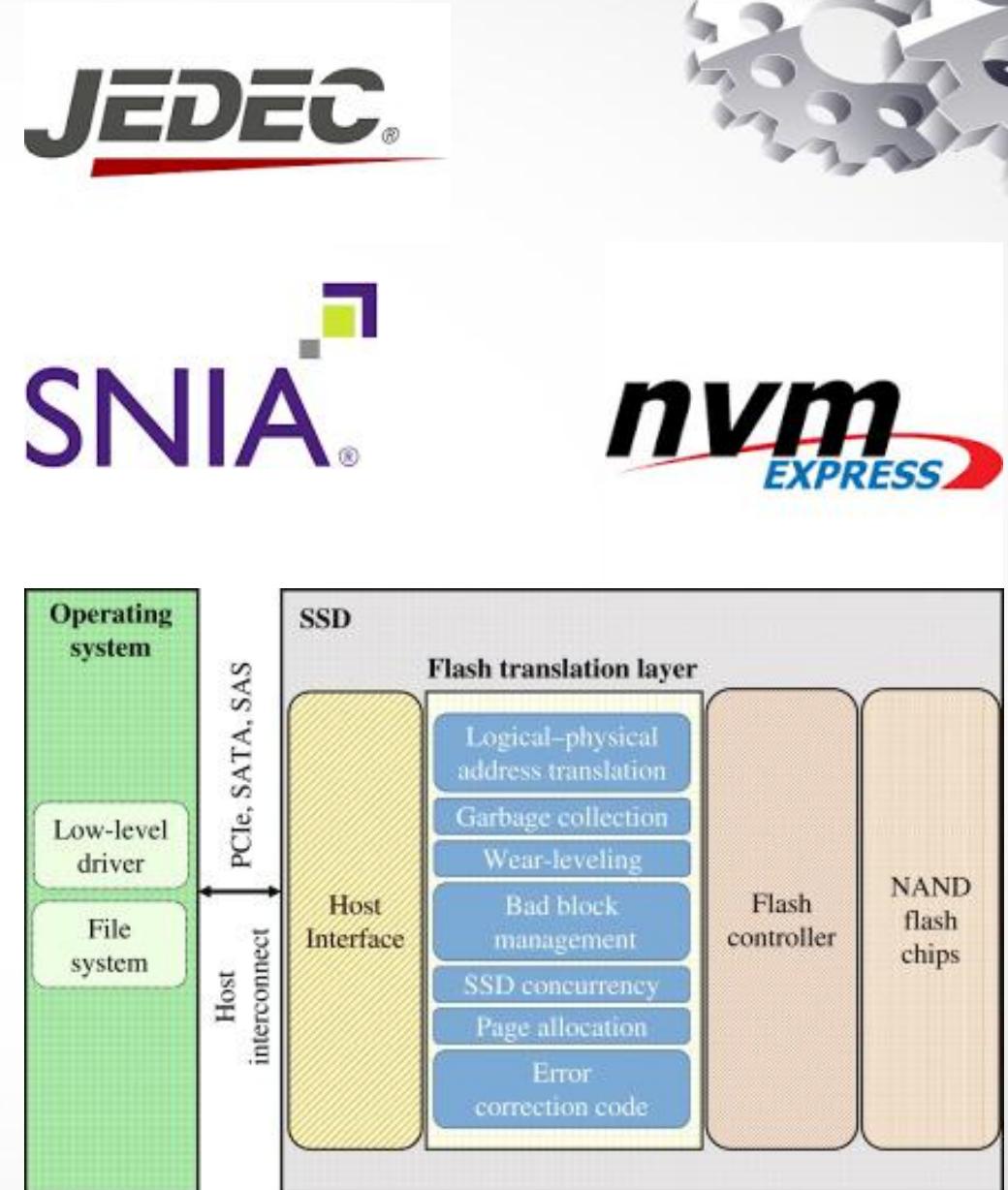


# vRunner: Mass Deploy



# 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()
```



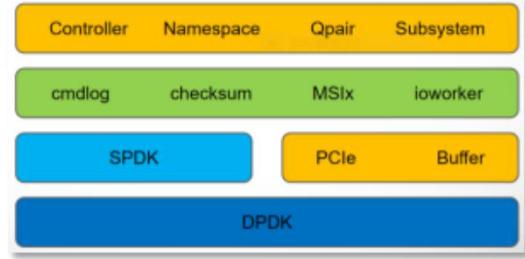
# Values



Ecosystem



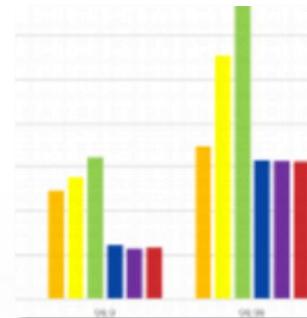
Scripts



Expendability



Hardware



Performance



Service



**pynvme** builds your own tests.



**Thanks!**