SDC 20

BY Developers FOR Developers

Storage Developer Conference
September 22-23, 2020

pynvme: an open, fast and extensible NVMe SSD test tool

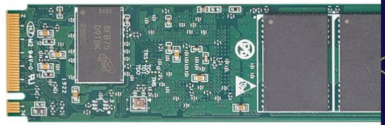
Crane Chu, Engineer, Founder
GENG YUN Technology Pte Ltd

Requirement

Changes in SSD

- SSD has been changing for the decade:
  - media:
    - SLC, MLC, TLC, QLC
    - 2D => 3D
    - PCM, 3D-Xpoint...
  - host:
    - PATA, SATA, PCIe/NVMe
    - open-channel
    - up coming: ZNS, KV, ... ?
  - DRAM
  - form factor

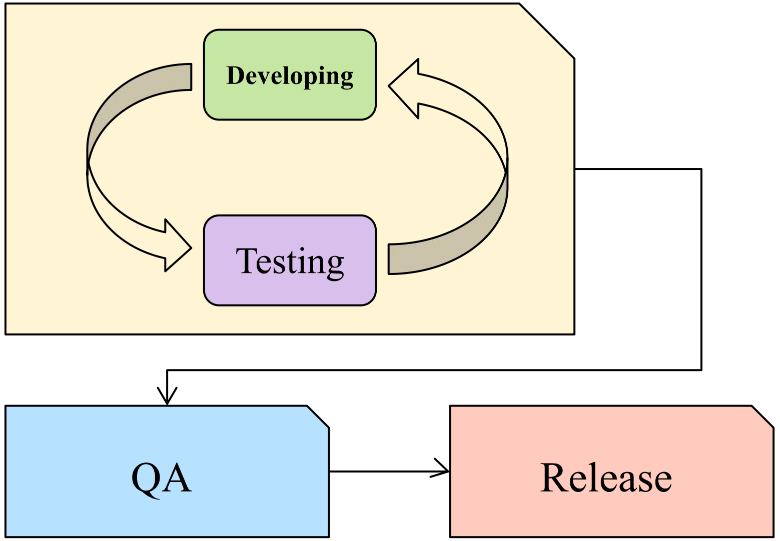
- Agile: good for the constant change and uncertainty



Agile Testing

- Developing and Testing are done interactively and iteratively.
- QA verifies the product of Dev/Test for customers.
- Testing and QA are different. Testing tools and QA tools are also different.
- Most available tools in the market are QA tools.

test	QA
for developer	for customer
before checkin	before release
automatic	manual
white-box	black-box
short	long
changing	stable

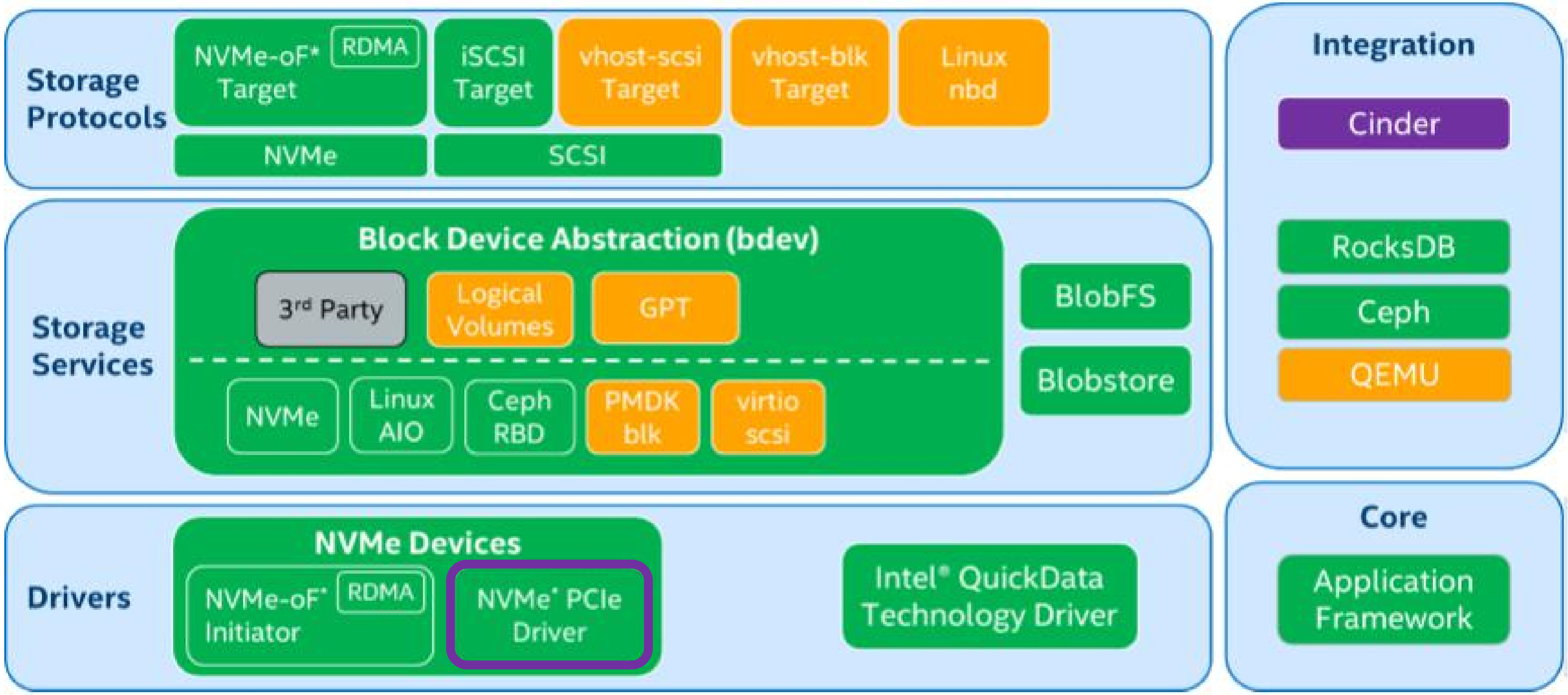


Experience with dnvme

- extended dnvme with a python wrapper in user space, thus developers can write test scripts in Python.
- Some essential problems:
  - performance:
    - IOPS, latency, consistency
    - test efficiency
    - stress tests
  - maintainness: kernel module
  - memory: user can only allocate virtual memory
- Then, SPDK comes ...
  - high performance
  - user space driver
  - DMA memory
- python + nvme: pynvme https://github.com/pynvme/pynvme

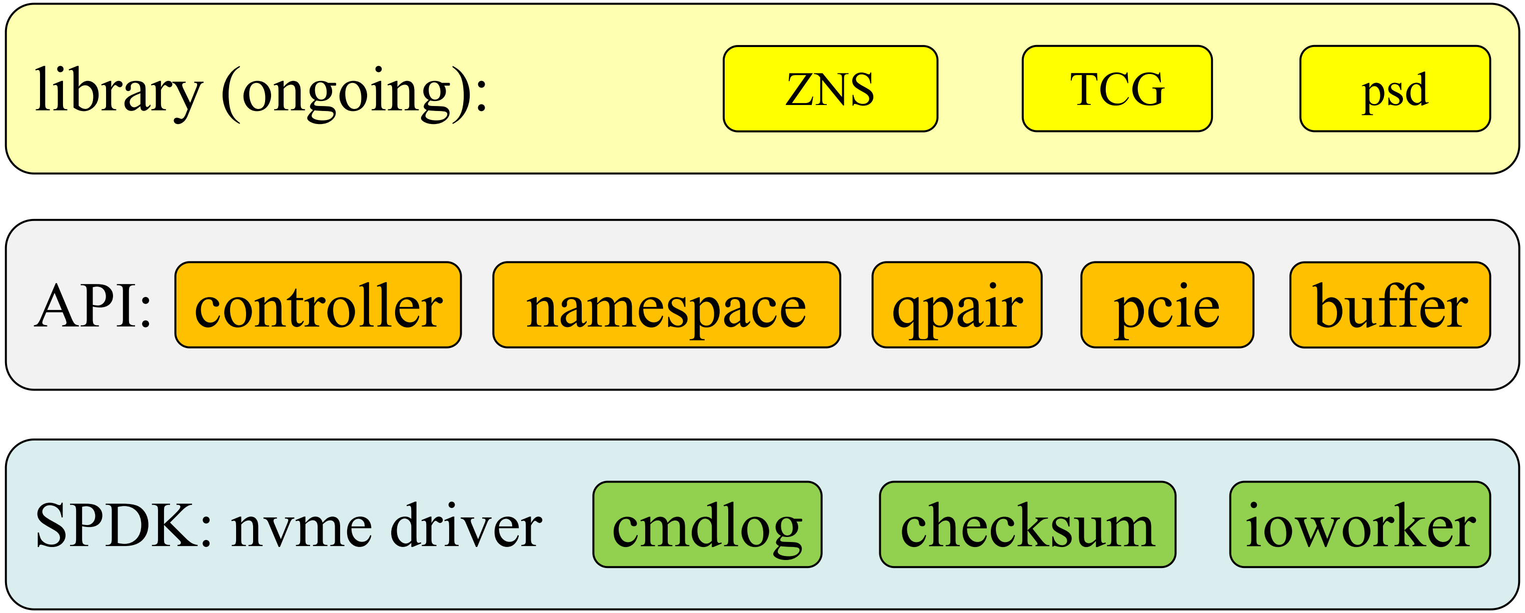
Design

SPDK



Storage Protocols
NVMe-oF* Target
RDMA
iSCSI Target
vhost-scsi Target
vhost-blk Target
Linux nbd
NVMe
SCSI
Storage Services
Block Device Abstraction (bdev)
3rd Party
Logical Volumes
GPT
NVMe
Linux AIO
Ceph RBD
PMDK blk
virtio scsi
BlobFS
Blobstore
Drivers
NVMe Devices
NVMe-oF* Initiator
RDMA
NVMe* PCIe Driver
Intel® QuickData Technology Driver
Integration
Cinder
RocksDB
Ceph
QEMU
Core
Application Framework

Architecture



Open Ecosystem

Python

pytest

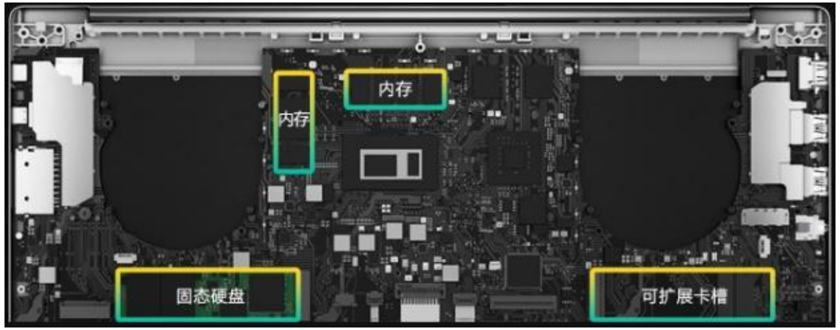
Visual Studio Code

Jenkins

matplotlib

GitHub

Hardware



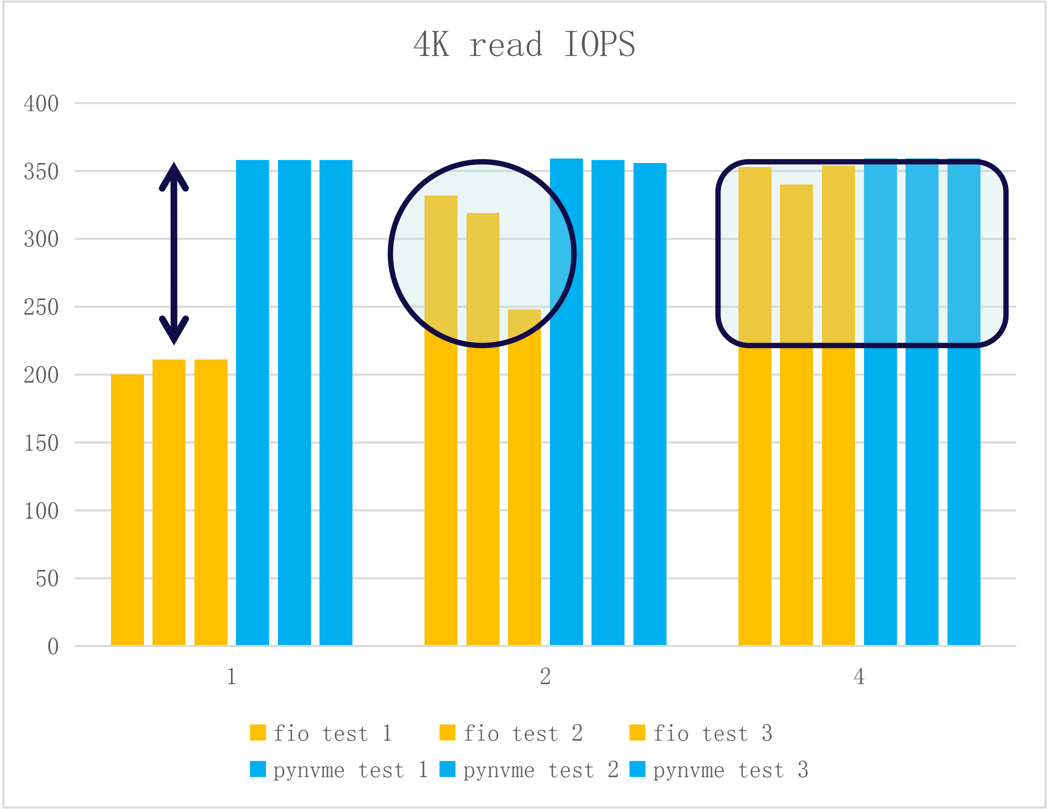
High Performance

fio (unit: K IOPS)

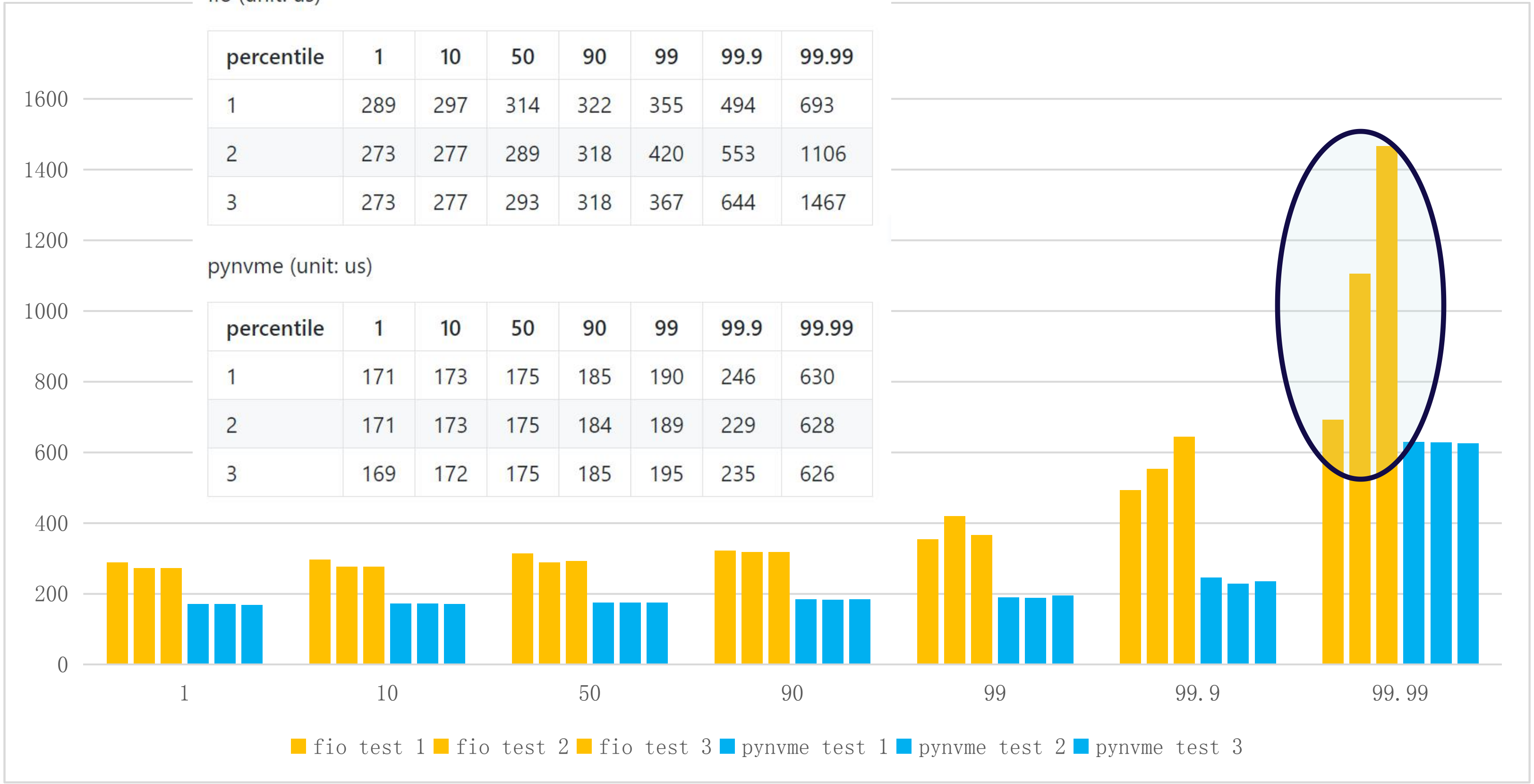
Q count	1	2	4
test 1	200	332	353
test 2	211	319	340
test 3	211	248	354

pynvme (unit: K IOPS)

Q count	1	2	4
test 1	358	359	359
test 2	358	358	359
test 3	358	356	359



Low Latency



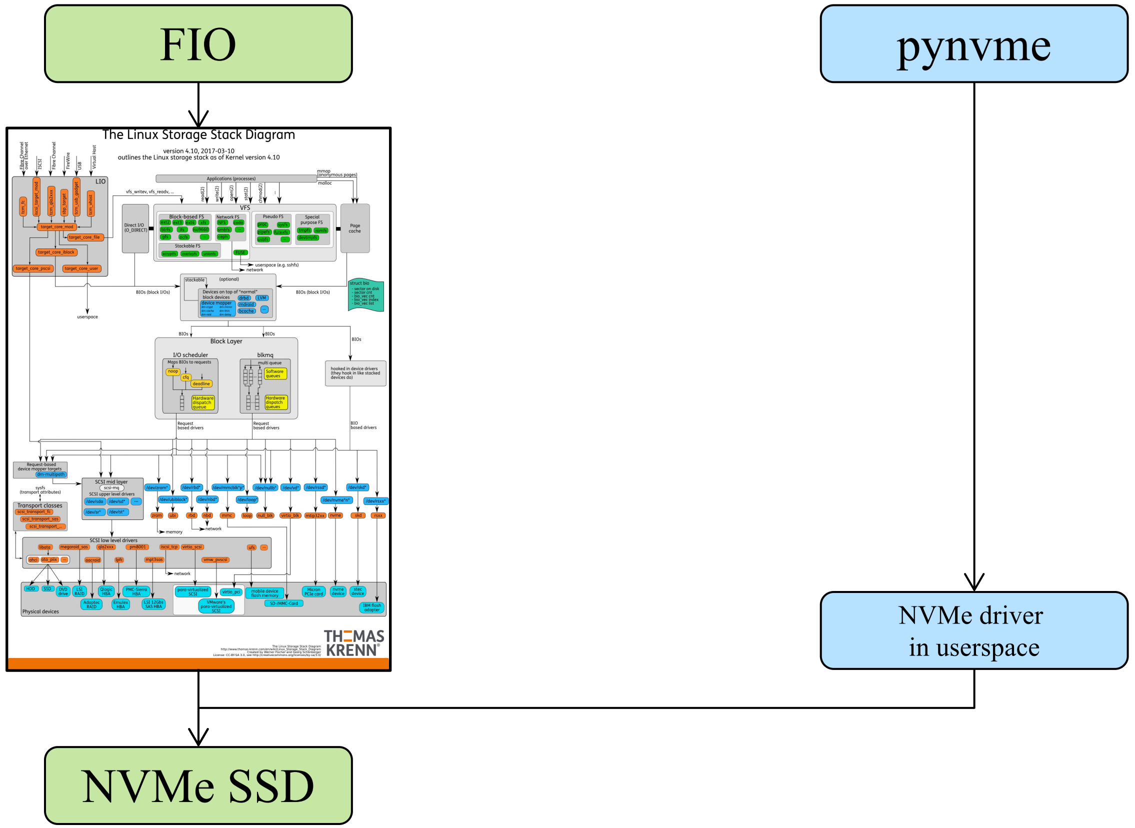
fio (unit: us)

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

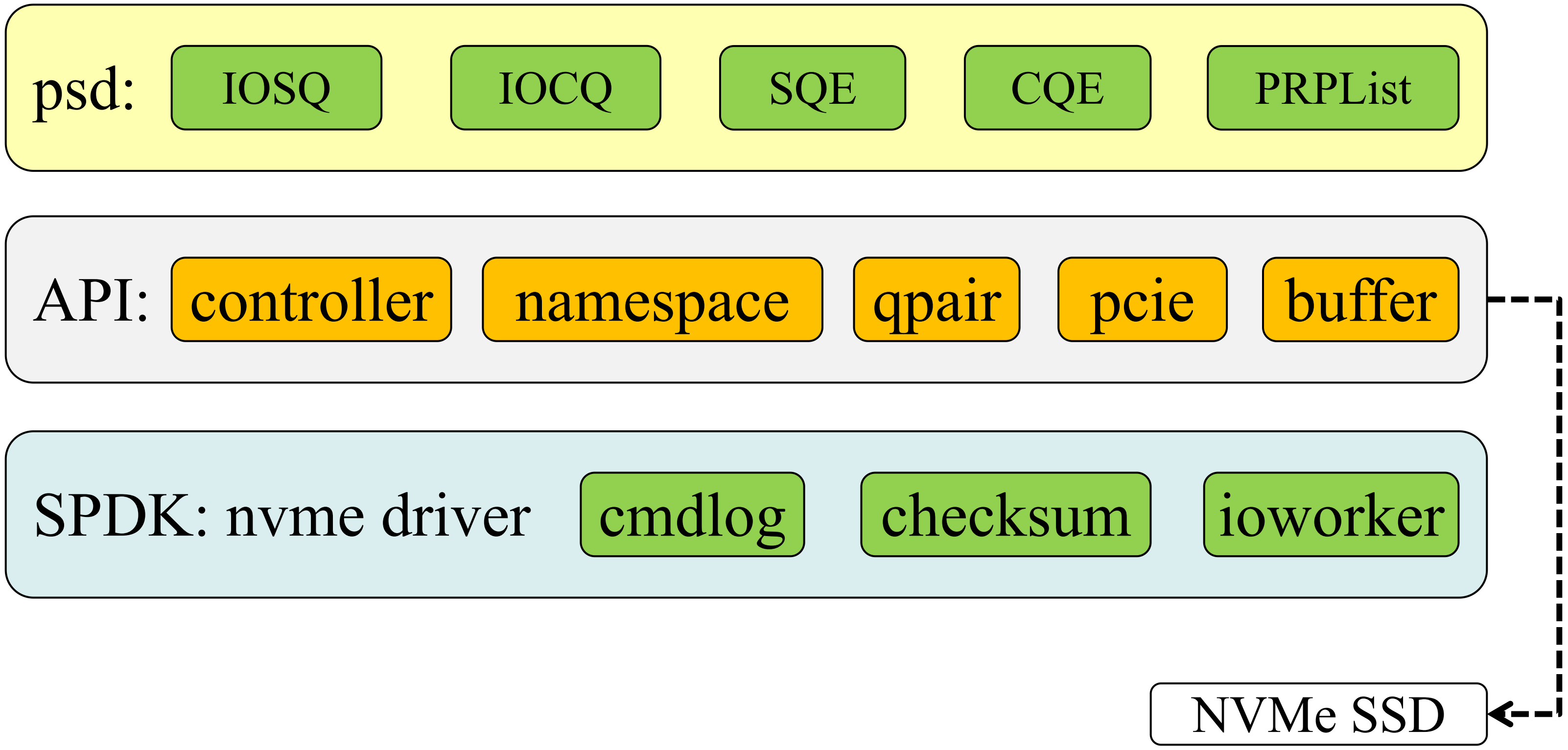
pynvme (unit: us)

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

Design


FIO
pynvme
The Linux Storage Stack Diagram
version 4.10, 2017-03-10
outlines the Linux storage stack as of Kernel version 4.10
NVMe driver in userspace
NVMe SSD

psd: Python Space Driver



Features

- access PCI configuration space
- access NVMe registers in BAR space
- send any NVMe admin/IO commands
- support callback functions for NVMe commands
- support MSI/MSIx interrupts
- transparent checksum verification on every LBA
- generates IO workload of high performance and low latency
- support multiple namespaces
- support multiple tests on different controllers
- integrate with the test framework pytest
- integrate with VSCode to display cmdlog in GUI
- support NVMe over TCP targets

- doc: https://pynvme.readthedocs.io/

Examples

Example: hello world

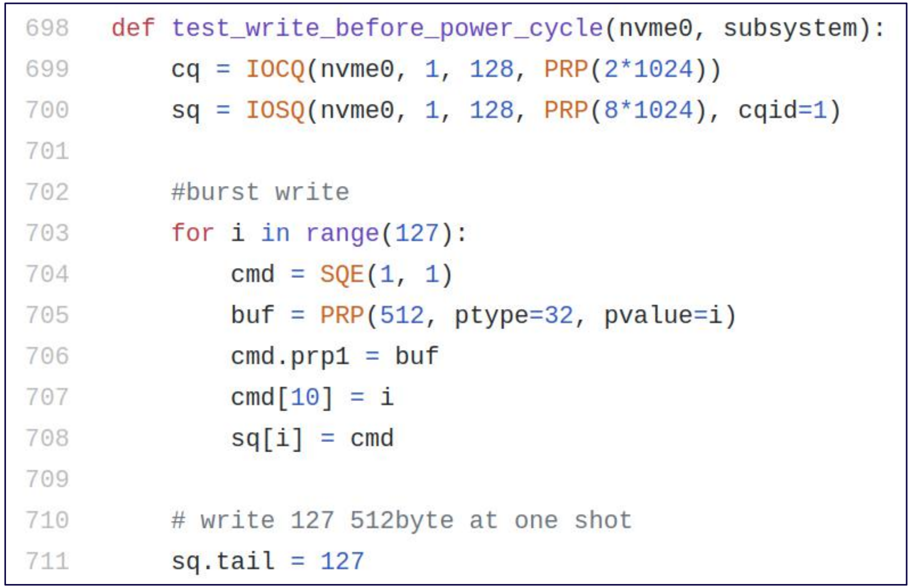
```python
import time
import pytest
import logging

import nvme as d


# intuitive, spec, qpair, vscode, debug, cmdlog, assert
def test_hello_world(nvme0, nvme0n1, qpair):
    # prepare data buffer and IO queue
    read_buf = d.Buffer(512)
    write_buf = d.Buffer(512)
    write_buf[10:21] = b'hello world'

    # 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'
```

- pytest fixtures
- callback functions for commands
- sync point: waitdone()

- run test:

```
> make setup
> make test TESTS=scripts/test_examples.py::test_hello_world
> make test TESTS=scripts/test_examples.py::test_hello_world pciaddr=0000:3d:00.0
> make test TESTS=scripts/test_examples.py::test_hello_world pciaddr=172.168.5.44
```

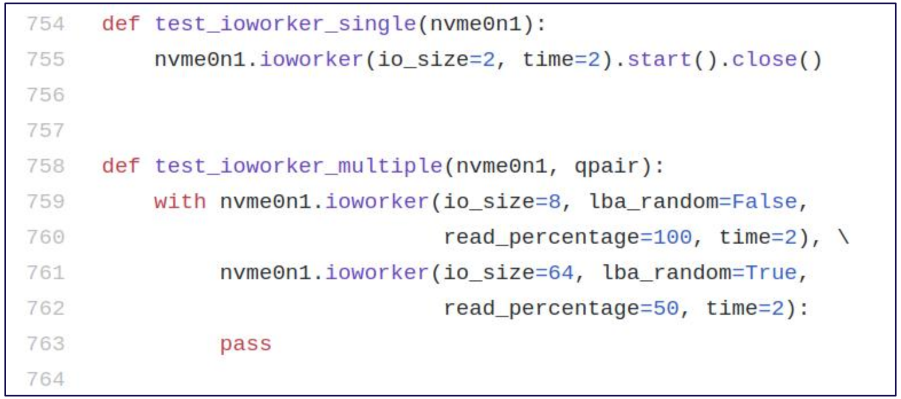
Example: psd



```python
def test_write_before_power_cycle(nvme0, subsystem):
    cq = IOCQ(nvme0, 1, 128, PRP(2*1024))
    sq = IOSQ(nvme0, 1, 128, PRP(8*1024), cqid=1)

    #burst write
    for i in range(127):
        cmd = SQE(1, 1)
        buf = PRP(512, ptype=32, pvalue=i)
        cmd.prp1 = buf
        cmd[10] = i
        sq[i] = cmd

    # write 127 512byte at one shot
    sq.tail = 127
```

- create IOCQ/IOSQ respectively with specified sqid and cqid
- fill commands in SQ entries
- manually trigger doorbell

Example: ioworker



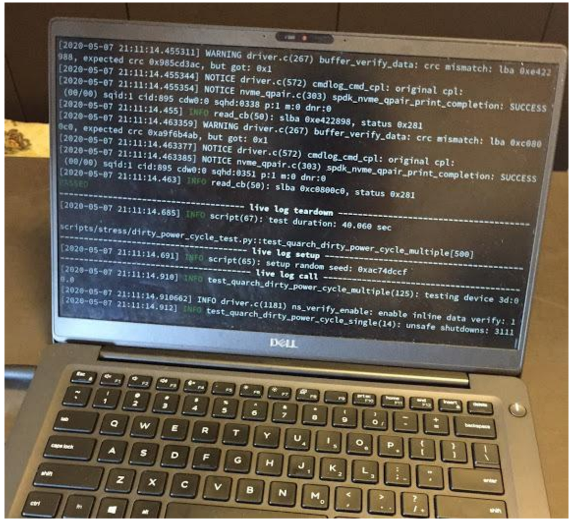
```python
def test_ioworker_single(nvme0n1):
    nvme0n1.ioworker(io_size=2, time=2).start().close()


def test_ioworker_multiple(nvme0n1, qpair):
    with nvme0n1.ioworker(io_size=8, lba_random=False,
                          read_percentage=100, time=2), \
         nvme0n1.ioworker(io_size=64, lba_random=True,
                          read_percentage=50, time=2):
        pass

```

- fio-like IO generator
  - Python API
  - better performance
- sending IO in separated processes
- define IO pattern in parameters
- support multiple ioworkers
- support workload from simple to complex

Example: dirty power cycle

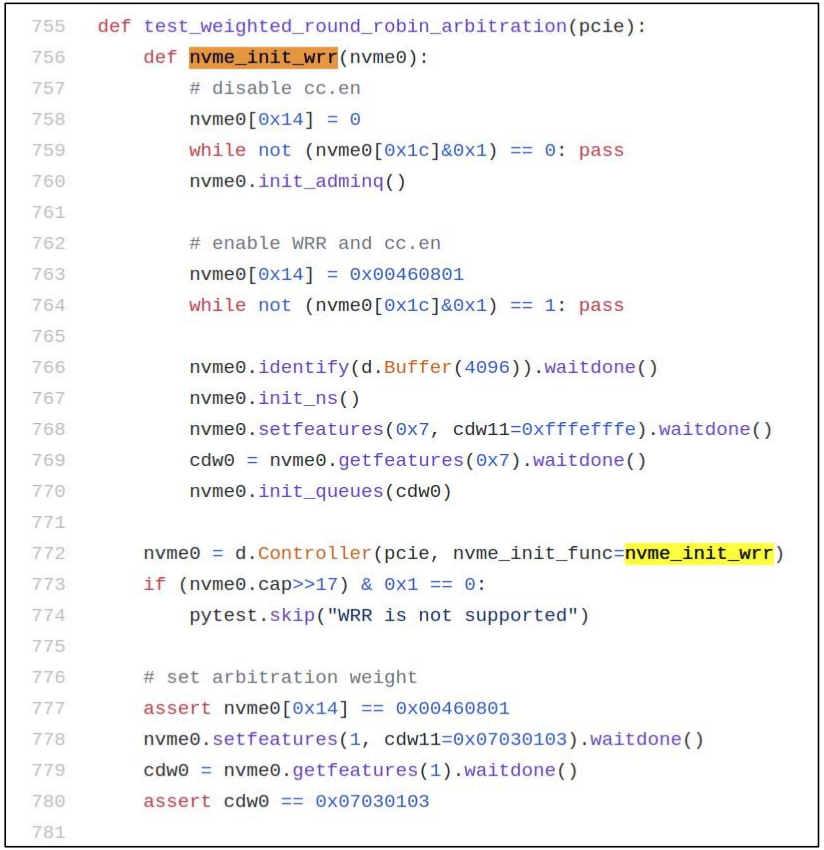
```python
    # 128K random write
    cmdlog_list = [None]*1000
    with nvme0n1.ioworker(io_size=256,
                          lba_random=True,
                          read_percentage=30,
                          region_end=region_end,
                          time=30,
                          qdepth=qdepth,
                          output_cmdlog_list=cmdlog_list):
        # sudden power loss before the ioworker end
        time.sleep(10)
        subsystem.poweroff()

    # power on and reset controller
    time.sleep(5)
    subsystem.poweron()
    nvme0.reset()
```

- ioworker sending IO in a separated process
- cut power in main process when ioworker is working



```
[2020-05-07 21:11:14.455311] WARNING driver.c(267) buffer_verify_data: crc mismatch: lba 0xe422
988, expected crc 0x985cd3ac, but got: 0x1
[2020-05-07 21:11:14.455344] NOTICE driver.c(572) cmdlog_cmd_cpl: original cpl:
[2020-05-07 21:11:14.455354] NOTICE nvme_qpair.c(303) spdk_nvme_qpair_print_completion: SUCCESS
 (00/00) sqid:1 cid:895 cdw0:0 sqhd:0338 p:1 m:0 dnr:0
[2020-05-07 21:11:14.455] INFO read_cb(50): slba 0xe422898, status 0x281
[2020-05-07 21:11:14.463359] WARNING driver.c(267) buffer_verify_data: crc mismatch: lba 0xc080
0c0, expected crc 0xa9f6b4ab, but got: 0x1
[2020-05-07 21:11:14.463377] NOTICE driver.c(572) cmdlog_cmd_cpl: original cpl:
[2020-05-07 21:11:14.463385] NOTICE nvme_qpair.c(303) spdk_nvme_qpair_print_completion: SUCCESS
 (00/00) sqid:1 cid:895 cdw0:0 sqhd:0351 p:1 m:0 dnr:0
[2020-05-07 21:11:14.463] INFO read_cb(50): slba 0xc0800c0, status 0x281
PASSED
------------------------------ live log teardown ------------------------------
[2020-05-07 21:11:14.685] INFO script(67): test duration: 40.060 sec

scripts/stress/dirty_power_cycle_test.py::test_quarch_dirty_power_cycle_multiple[500]
------------------------------ live log setup ------------------------------
[2020-05-07 21:11:14.691] INFO script(65): setup random seed: 0xac74dccf
------------------------------ live log call ------------------------------
[2020-05-07 21:11:14.910] INFO test_quarch_dirty_power_cycle_multiple(125): testing device 3d:0
0.0
[2020-05-07 21:11:14.910662] INFO driver.c(1181) ns_verify_enable: enable inline data verify: 1
[2020-05-07 21:11:14.912] INFO test_quarch_dirty_power_cycle_single(14): unsafe shutdowns: 3111
```

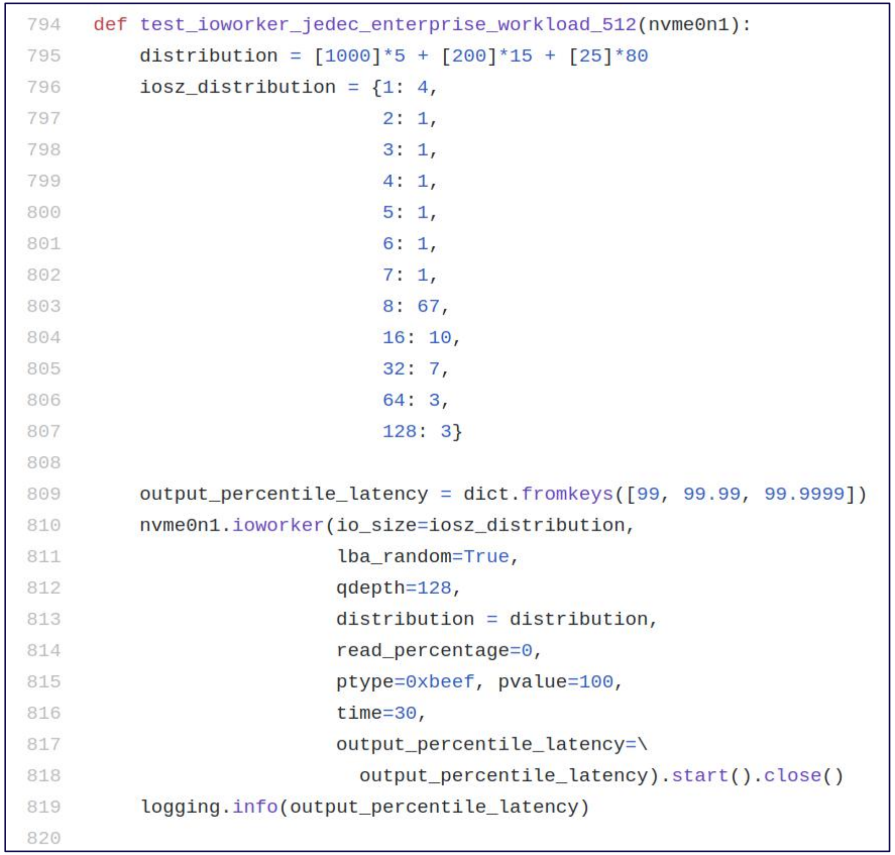
Example: customized nvme init



```
def test_weighted_round_robin_arbitration(pcie):
    def nvme_init_wrr(nvme0):
        # disable cc.en
        nvme0[0x14] = 0
        while not (nvme0[0x1c]&0x1) == 0: pass
        nvme0.init_adminq()

        # enable WRR and cc.en
        nvme0[0x14] = 0x00460801
        while not (nvme0[0x1c]&0x1) == 1: pass

        nvme0.identify(d.Buffer(4096)).waitdone()
        nvme0.init_ns()
        nvme0.setfeatures(0x7, cdw11=0xfffefffe).waitdone()
        cdw0 = nvme0.getfeatures(0x7).waitdone()
        nvme0.init_queues(cdw0)

    nvme0 = d.Controller(pcie, nvme_init_func=nvme_init_wrr)
    if (nvme0.cap>>17) & 0x1 == 0:
        pytest.skip("WRR is not supported")

    # set arbitration weight
    assert nvme0[0x14] == 0x00460801
    nvme0.setfeatures(1, cdw11=0x07030103).waitdone()
    cdw0 = nvme0.getfeatures(1).waitdone()
    assert cdw0 == 0x07030103

```

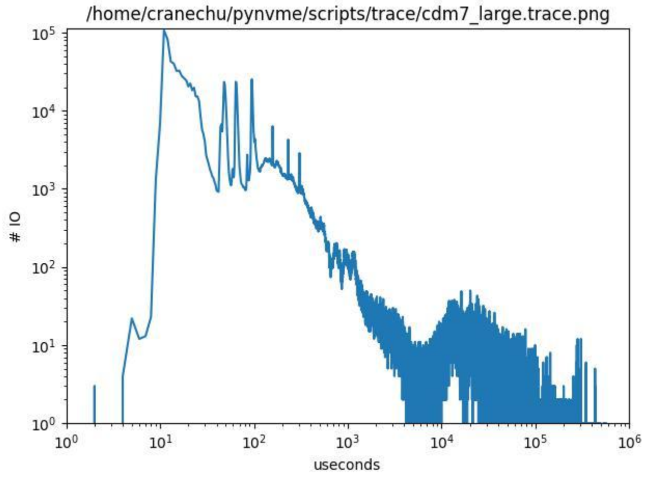
- pynvme has a default nvme initializaiton process
- users can define different nvme initialization process in test scripts

Example: latency of JEDEC workload

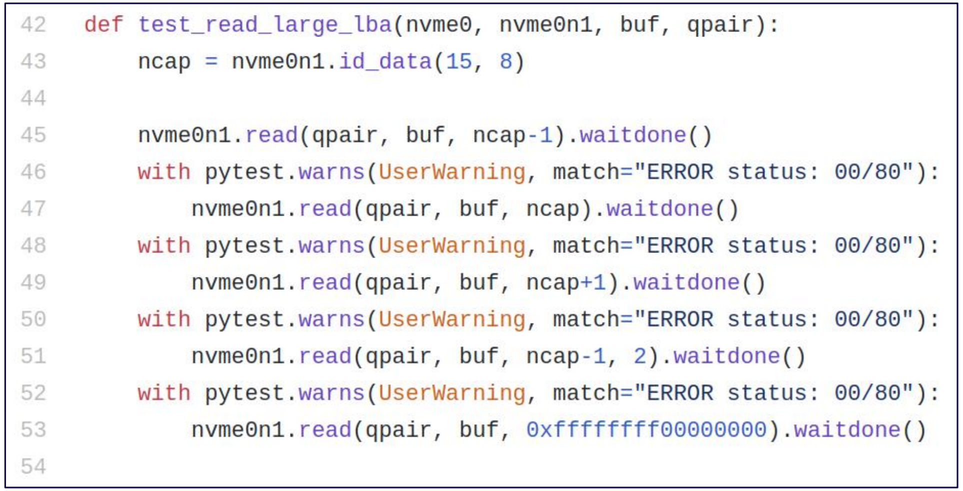


```python
def test_ioworker_jedec_enterprise_workload_512(nvme0n1):
    distribution = [1000]*5 + [200]*15 + [25]*80
    iosz_distribution = {1: 4,
                         2: 1,
                         3: 1,
                         4: 1,
                         5: 1,
                         6: 1,
                         7: 1,
                         8: 67,
                         16: 10,
                         32: 7,
                         64: 3,
                         128: 3}

    output_percentile_latency = dict.fromkeys([99, 99.99, 99.9999])
    nvme0n1.ioworker(io_size=iosz_distribution,
                     lba_random=True,
                     qdepth=128,
                     distribution = distribution,
                     read_percentage=0,
                     ptype=0xbeef, pvalue=100,
                     time=30,
                     output_percentile_latency=\
                       output_percentile_latency).start().close()
    logging.info(output_percentile_latency)

```

- ioworker generates IO according to JEDEC enterprise workload
- scripts get latency at different percentile (99%, 99.99%, 99.9999%)



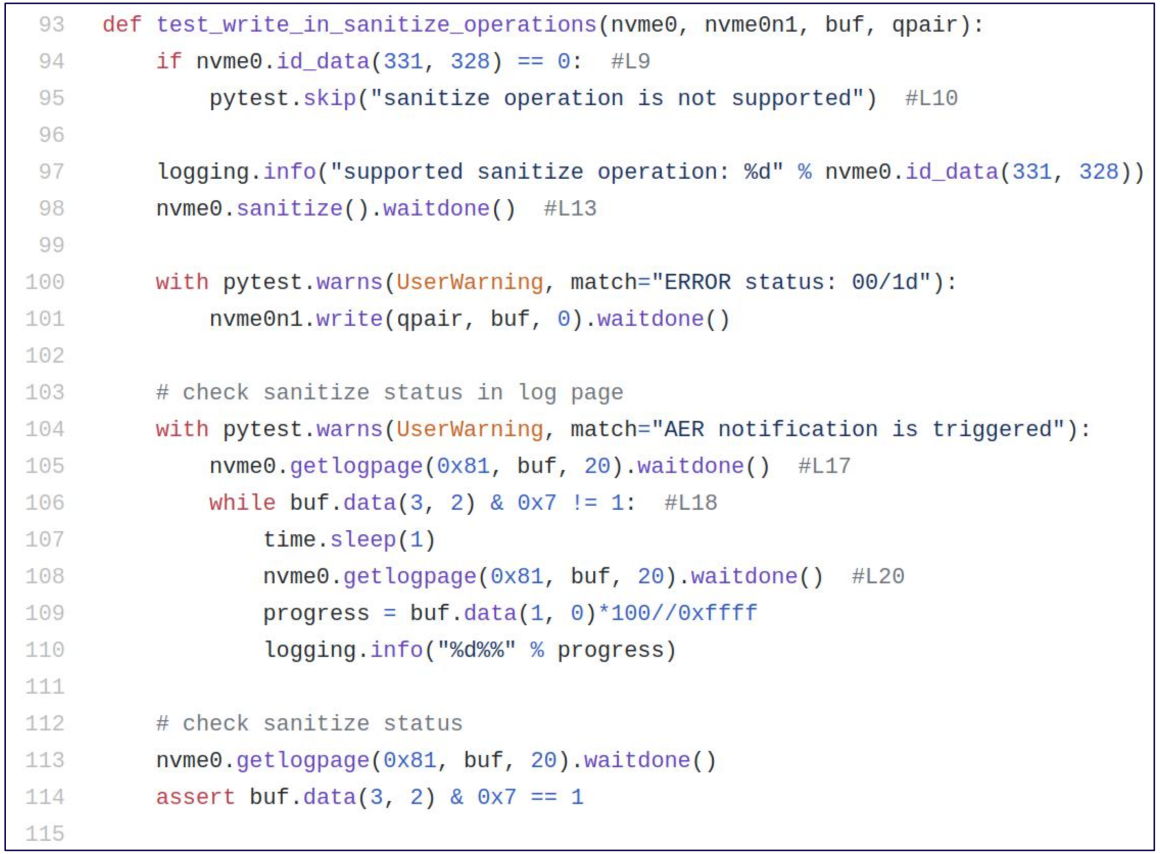
Example: check on error code



```python
def test_read_large_lba(nvme0, nvme0n1, buf, qpair):
    ncap = nvme0n1.id_data(15, 8)

    nvme0n1.read(qpair, buf, ncap-1).waitdone()
    with pytest.warns(UserWarning, match="ERROR status: 00/80"):
        nvme0n1.read(qpair, buf, ncap).waitdone()
    with pytest.warns(UserWarning, match="ERROR status: 00/80"):
        nvme0n1.read(qpair, buf, ncap+1).waitdone()
    with pytest.warns(UserWarning, match="ERROR status: 00/80"):
        nvme0n1.read(qpair, buf, ncap-1, 2).waitdone()
    with pytest.warns(UserWarning, match="ERROR status: 00/80"):
        nvme0n1.read(qpair, buf, 0xffffffff00000000).waitdone()

```

- pynvme through a warning when command completes with an error
- scripts can use pytest to capture and check the warning

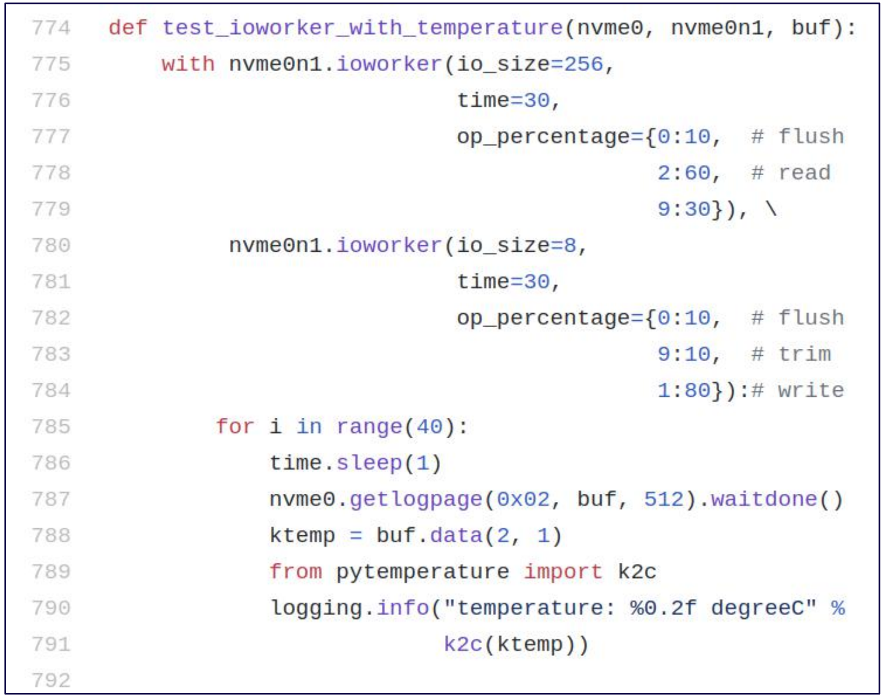
Example: sanitize an aer



```python
def test_write_in_sanitize_operations(nvme0, nvme0n1, buf, qpair):
    if nvme0.id_data(331, 328) == 0:  #L9
        pytest.skip("sanitize operation is not supported")  #L10

    logging.info("supported sanitize operation: %d" % nvme0.id_data(331, 328))
    nvme0.sanitize().waitdone()  #L13

    with pytest.warns(UserWarning, match="ERROR status: 00/1d"):
        nvme0n1.write(qpair, buf, 0).waitdone()

    # check sanitize status in log page
    with pytest.warns(UserWarning, match="AER notification is triggered"):
        nvme0.getlogpage(0x81, buf, 20).waitdone()  #L17
        while buf.data(3, 2) & 0x7 != 1:  #L18
            time.sleep(1)
            nvme0.getlogpage(0x81, buf, 20).waitdone()  #L20
            progress = buf.data(1, 0)*100//0xffff
            logging.info("%d%%" % progress)

    # check sanitize status
    nvme0.getlogpage(0x81, buf, 20).waitdone()
    assert buf.data(3, 2) & 0x7 == 1
```

- check if sanitize is supported
- start sanitize operation
- send a write command, and check if it is aborted due to the sanitize operation in progress
- monitor the sanitize progress till it is completed
- check the final sanitize status in logpage

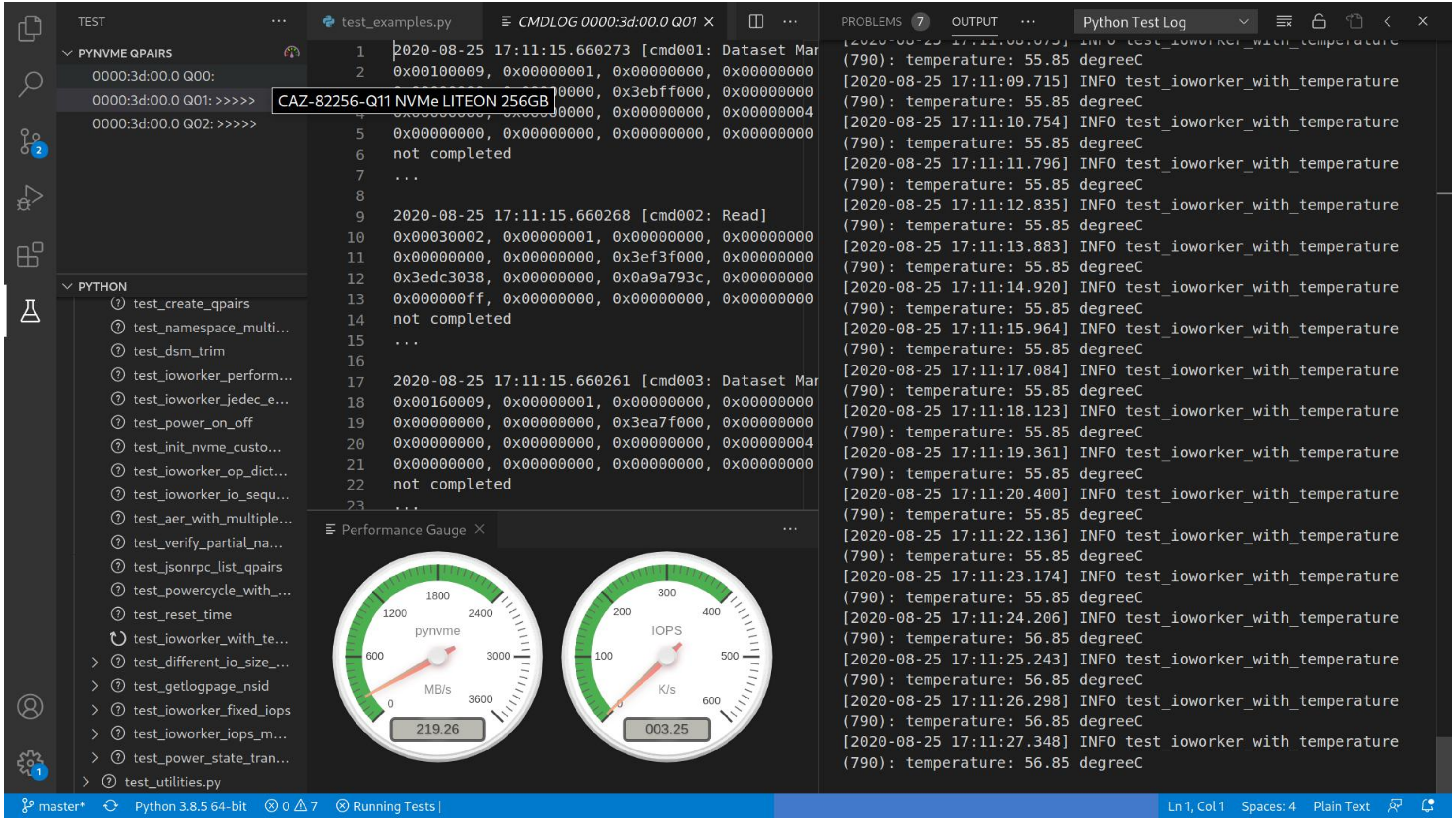
Example: multiple processes



```python
def test_ioworker_with_temperature(nvme0, nvme0n1, buf):
    with nvme0n1.ioworker(io_size=256,
                          time=30,
                          op_percentage={0:10,  # flush
                                         2:60,  # read
                                         9:30}), \
         nvme0n1.ioworker(io_size=8,
                          time=30,
                          op_percentage={0:10,  # flush
                                         9:10,  # trim
                                         1:80}):# write
        for i in range(40):
            time.sleep(1)
            nvme0.getlogpage(0x02, buf, 512).waitdone()
            ktemp = buf.data(2, 1)
            from pytemperature import k2c
            logging.info("temperature: %0.2f degreeC" %
                         k2c(ktemp))

```

- process 1: ioworker with flush, read, trim commands
- process 2: ioworker with flush, write, trim commands
- main process: print temperature value in SMART data for every second

Live Demo in VSCode



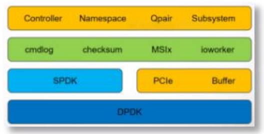
```
2020-08-25 17:11:15.660273 [cmd001: Dataset Mar
0x00100009, 0x00000001, 0x00000000, 0x00000000
0x00000000, 0x00000000, 0x3ebff000, 0x00000000
0x00000000, 0x00000000, 0x00000000, 0x00000004
0x00000000, 0x00000000, 0x00000000, 0x00000000
not completed
...

2020-08-25 17:11:15.660268 [cmd002: Read]
0x00030002, 0x00000001, 0x00000000, 0x00000000
0x00000000, 0x00000000, 0x3ef3f000, 0x00000000
0x3edc3038, 0x00000000, 0x0a9a793c, 0x00000000
0x000000ff, 0x00000000, 0x00000000, 0x00000000
not completed
...

2020-08-25 17:11:15.660261 [cmd003: Dataset Mar
0x00160009, 0x00000001, 0x00000000, 0x00000000
0x00000000, 0x00000000, 0x3ea7f000, 0x00000000
0x00000000, 0x00000000, 0x00000000, 0x00000004
0x00000000, 0x00000000, 0x00000000, 0x00000000
not completed
...
```

```
[2020-08-25 17:11:08.675] INFO test_ioworker_with_temperature
(790): temperature: 55.85 degreeC
[2020-08-25 17:11:09.715] INFO test_ioworker_with_temperature
(790): temperature: 55.85 degreeC
[2020-08-25 17:11:10.754] INFO test_ioworker_with_temperature
(790): temperature: 55.85 degreeC
[2020-08-25 17:11:11.796] INFO test_ioworker_with_temperature
(790): temperature: 55.85 degreeC
[2020-08-25 17:11:12.835] INFO test_ioworker_with_temperature
(790): temperature: 55.85 degreeC
[2020-08-25 17:11:13.883] INFO test_ioworker_with_temperature
(790): temperature: 55.85 degreeC
[2020-08-25 17:11:14.920] INFO test_ioworker_with_temperature
(790): temperature: 55.85 degreeC
[2020-08-25 17:11:15.964] INFO test_ioworker_with_temperature
(790): temperature: 55.85 degreeC
[2020-08-25 17:11:17.084] INFO test_ioworker_with_temperature
(790): temperature: 55.85 degreeC
[2020-08-25 17:11:18.123] INFO test_ioworker_with_temperature
(790): temperature: 55.85 degreeC
[2020-08-25 17:11:19.361] INFO test_ioworker_with_temperature
(790): temperature: 55.85 degreeC
[2020-08-25 17:11:20.400] INFO test_ioworker_with_temperature
(790): temperature: 55.85 degreeC
[2020-08-25 17:11:22.136] INFO test_ioworker_with_temperature
(790): temperature: 55.85 degreeC
[2020-08-25 17:11:23.174] INFO test_ioworker_with_temperature
(790): temperature: 55.85 degreeC
[2020-08-25 17:11:24.206] INFO test_ioworker_with_temperature
(790): temperature: 56.85 degreeC
[2020-08-25 17:11:25.243] INFO test_ioworker_with_temperature
(790): temperature: 56.85 degreeC
[2020-08-25 17:11:26.298] INFO test_ioworker_with_temperature
(790): temperature: 56.85 degreeC
[2020-08-25 17:11:27.348] INFO test_ioworker_with_temperature
(790): temperature: 56.85 degreeC
```

pynvme builds your own tests.

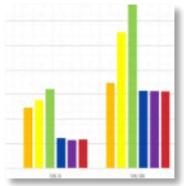
Ecosystem

Scripts



Extendability

Hardware



Performance

Service

Please take a moment
to rate this session.

Your feedback matters to us.

Thank you!!!