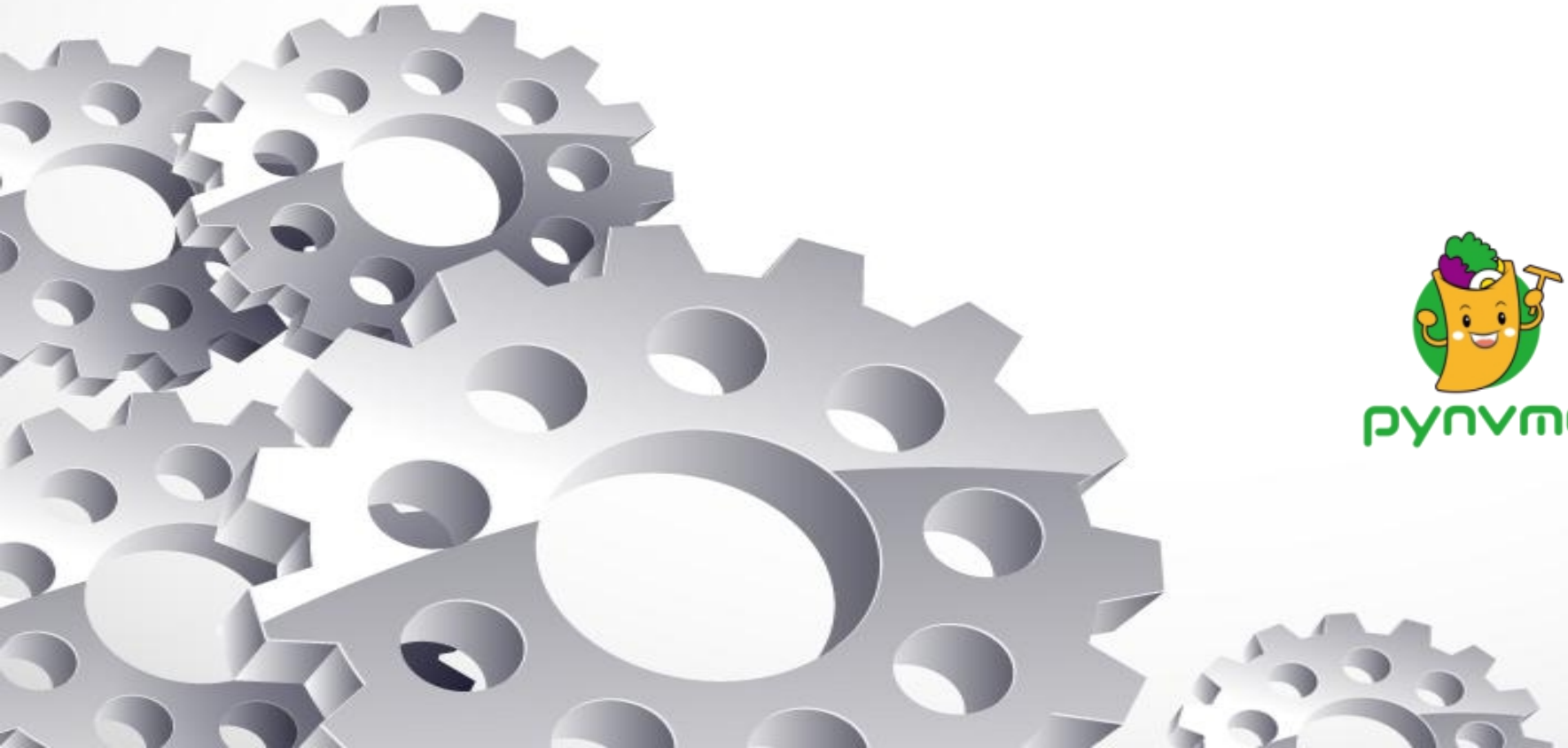

TEST
CMDLOG 0000:3d:00.0 Q05
CMDLOG 0000:3d:00.0 Q13
CMDLOG 0000:3d:00.0 Q00
PYNVME QPAIRS
0000:3d:00.0 Q00: >>
0000:3d:00.0 Q01: >>
0000:3d:00.0 Q02: >>
0000:3d:00.0 Q03: >>
0000:3d:00.0 Q04: >>
0000:3d:00.0 Q05: >>
0000:3d:00.0 Q06: >>
0000:3d:00.0 Q07: >>
0000:3d:00.0 Q08: >>
0000:3d:00.0 Q09: >>
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: >>
2020-04-15 13:33:21.463617 [cmd001: Read]
0x00050002, 0x00000001, 0x00000000, 0x0000000
0x00000000, 0x00000000, 0x0c7d3000, 0x0000000
0x00000000, 0x00000000, 0x213d9b90, 0x0000000
0x00000007, 0x00000000, 0x00000000, 0x0000000
not completed
...
2020-04-15 13:33:21.463613 [cmd002: Read]
0x00010002, 0x00000001, 0x00000000, 0x0000000
0x00000000, 0x00000000, 0x0c7d7000, 0x0000000
0x00000000, 0x00000000, 0x16b26aa0, 0x0000000
0x00000007, 0x00000000, 0x00000000, 0x0000000
not completed
...
2020-04-15 13:33:21.463607 [cmd003: Read]
0x00030002, 0x00000001, 0x00000000, 0x0000000
0x00000000, 0x00000000, 0x0c7db000, 0x0000000
0x00000000, 0x00000000, 0x151c50c0, 0x0000000
0x00000007, 0x00000000, 0x00000000, 0x0000000
not completed
Performance Gauge
pynvme
MB/s
1353.80
IOPS
K/s
330.52
2020-04-15 13:32:39.104405 [cmd001: Create I/O Submission Queue]
0x005a0001, 0x00000000, 0x00000000, 0x00000000
0x00000000, 0x00000000, 0x0ea00000, 0x00000001
0x00000000, 0x00000000, 0x00070010, 0x00100001
0x00000000, 0x00000000, 0x00000000, 0x00000000
2020-04-15 13:32:39.104519: [cpl: SUCCESS]
0x00000000, 0x00000000, 0x00000056, 0x0000005a
2020-04-15 13:32:39.103458 [cmd002: Create I/O Completion Queue]
0x005a0005, 0x00000000, 0x00000000, 0x00000000
0x00000000, 0x00000000, 0x0e800000, 0x00000001
0x00000000, 0x00000000, 0x00070010, 0x00100003
0x00000000, 0x00000000, 0x00000000, 0x00000000
2020-04-15 13:32:39.104403: [cpl: SUCCESS]
0x00000000, 0x00000000, 0x00000055, 0x0000005a
2020-04-15 13:32:38.533985 [cmd003: Create I/O Submission Queue]
0x005a0001, 0x00000000, 0x00000000, 0x00000000
0x00000000, 0x00000000, 0x0e600000, 0x00000001
0x00000000, 0x00000000, 0x0007000f, 0x000f0001
0x00000000, 0x00000000, 0x00000000, 0x00000000
2020-04-15 13:32:38.534648: [cpl: SUCCESS]
0x00000000, 0x00000000, 0x00000054, 0x0000005a
2020-04-15 13:32:38.533838 [cmd004: Create I/O Completion Queue]
0x005a0005, 0x00000000, 0x00000000, 0x00000000
0x00000000, 0x00000000, 0x0e400000, 0x00000001
0x00000000, 0x00000000, 0x0007000f, 0x000f0003
PYTHON
scripts
conformance
ftl
performance
snia
stress
tcg
trace
test_examples.py
test_utilities.py
pynvme builds your own tests.
master*
Python 3.7.6 64-bit
Run Tests
Ln 1, Col 1
Spaces: 4
Plain Text

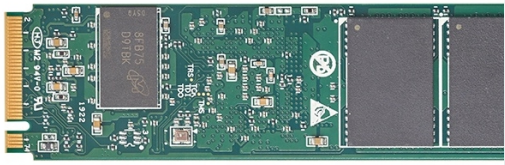
Requirement



pynvme builds your own tests.

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

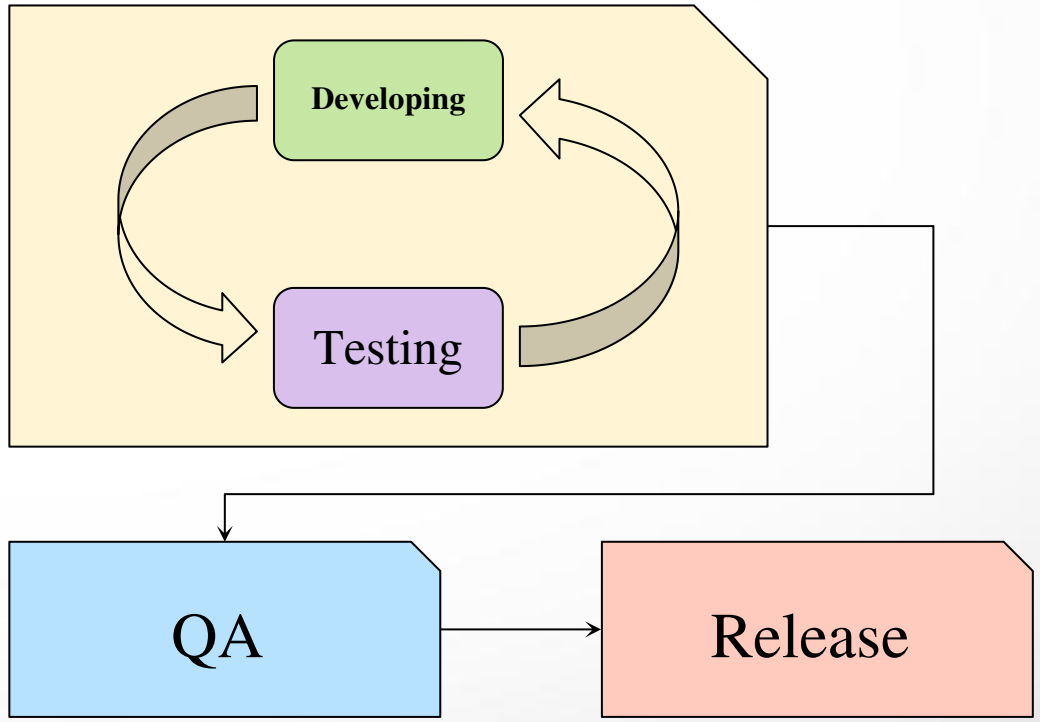


- Agile: good for the constant change and uncertainty

Agile Testing

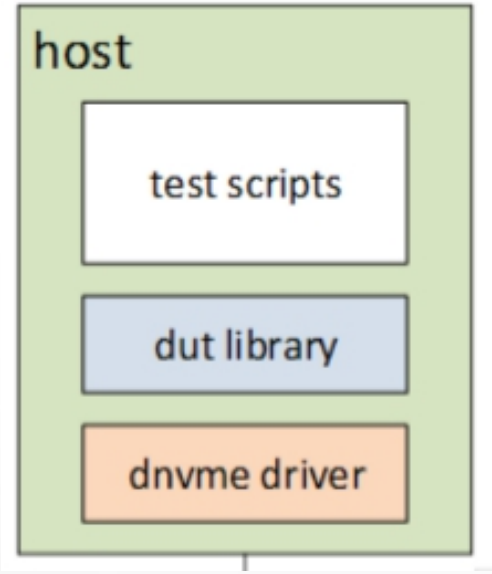
- Developing and Testing are done interactively and iteratively.
- QA verifies the product of Dev/Test for customers.
- Testing and QA are different. Testing tools and QA tools are also different.
- Most available tools in the market are QA tools.

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



Experience 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



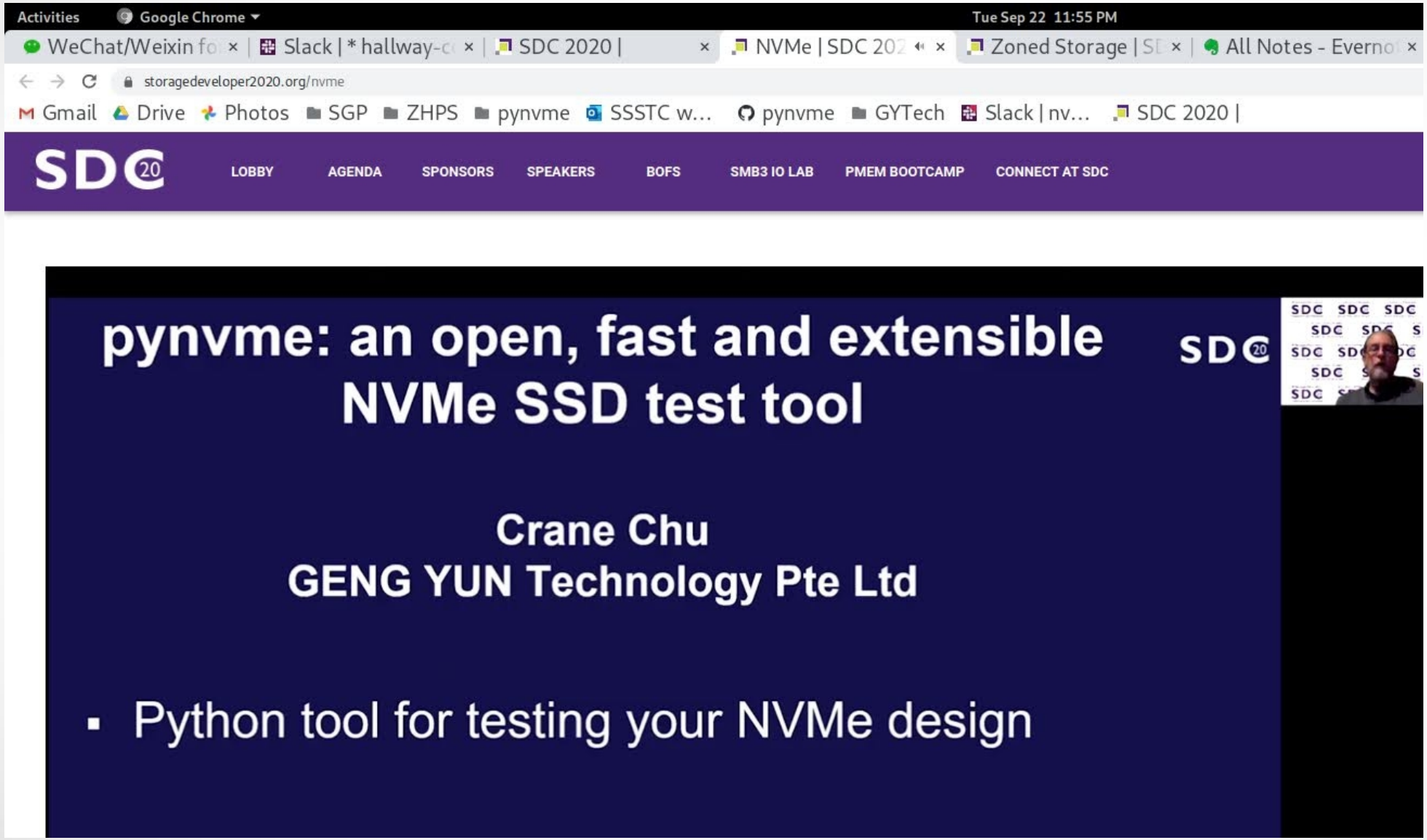
- challenges:
  - low performance:
    - IOPS, latency, consistency
    - test efficiency
    - stress tests
  - maintainness: kernel module
  - function coverage: PRP, ...
  - GPL License
- dnvme is good at NVMe test, but in-efficient on SSD test.

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

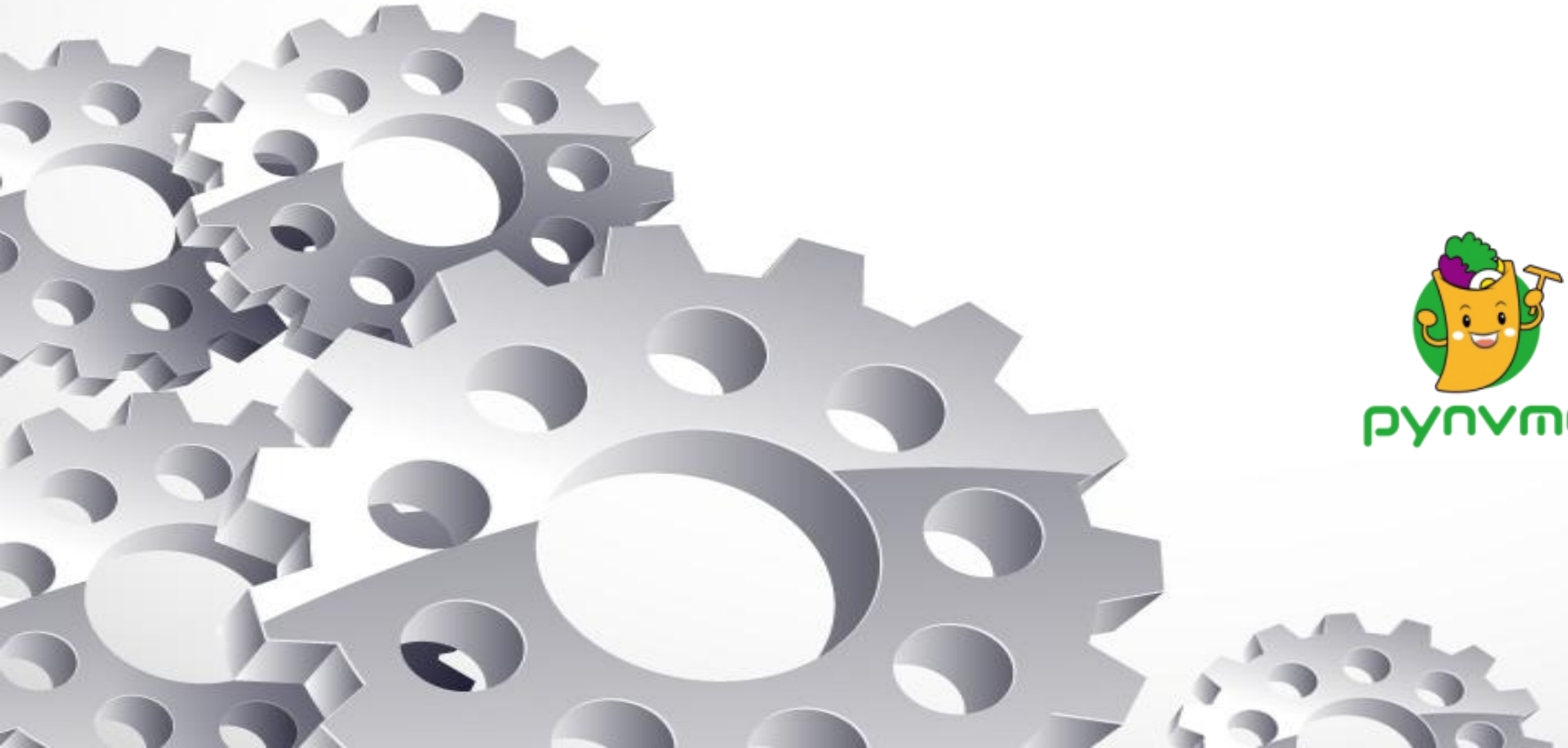
- meet challenges:
  - low performance:
    - IOPS, latency, consistency
    - test efficiency
    - stress tests
  - maintainness: kernel module
  - function coverage: PRP, ...
  - BSD License

SDC2020



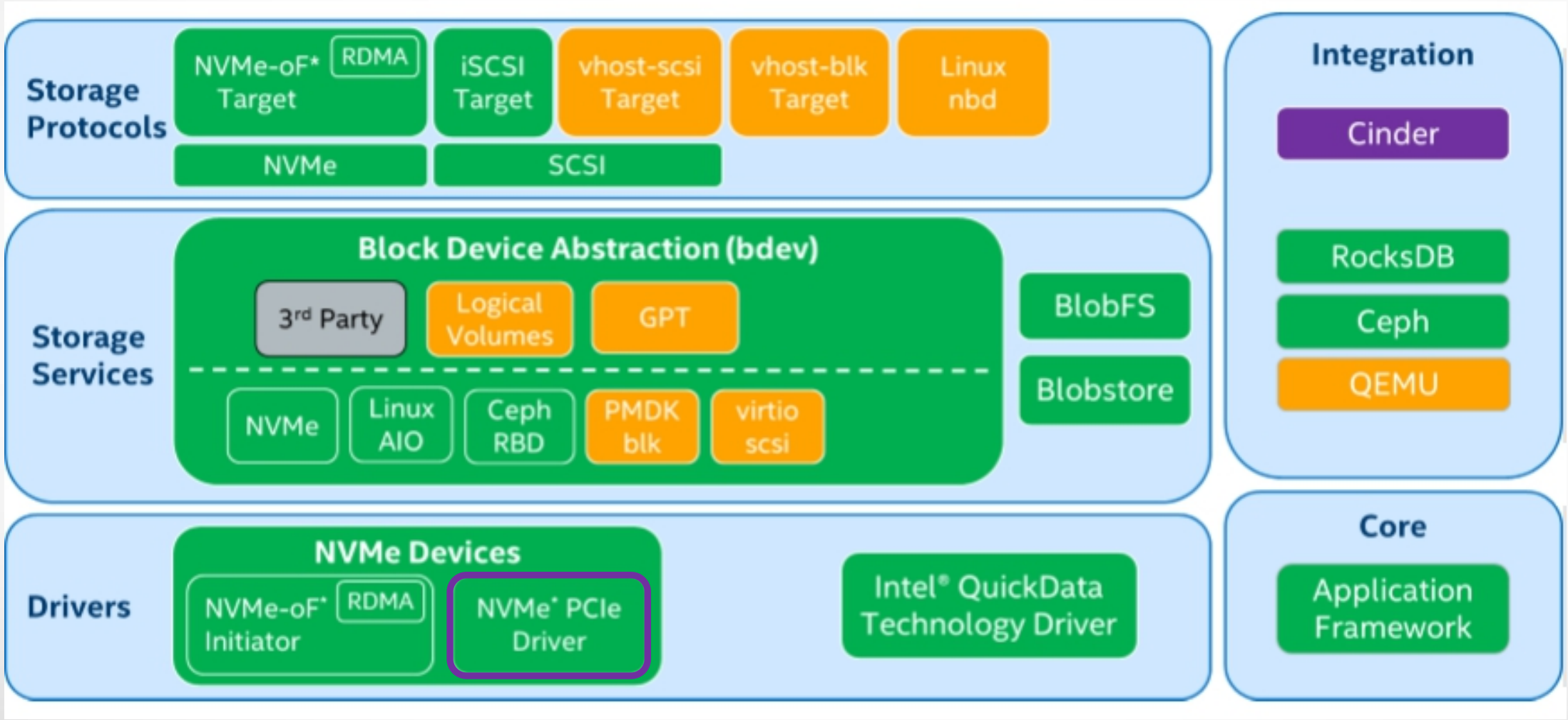
https://www.youtube.com/watch?v=Yoru7vzVyL8

Design



pynvme builds your own tests.

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

SPDK/DPDK

- Moving all of the necessary drivers into userspace, which avoids syscalls and enables zero-copy access from the application.

- Polling hardware for completions instead of relying on interrupts, which lowers both total latency and latency variance.

- Avoiding all locks in the I/O path, instead relying on message passing.

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

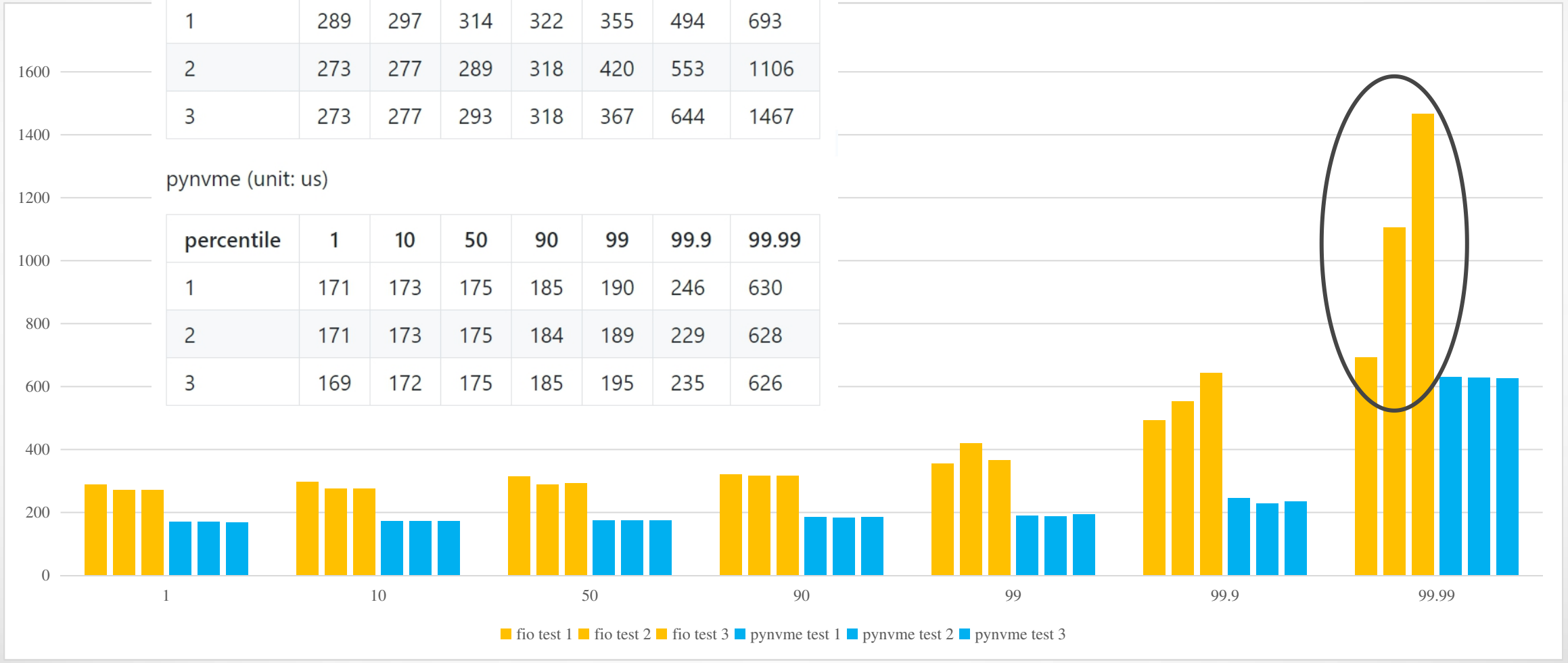
Performance: latency

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

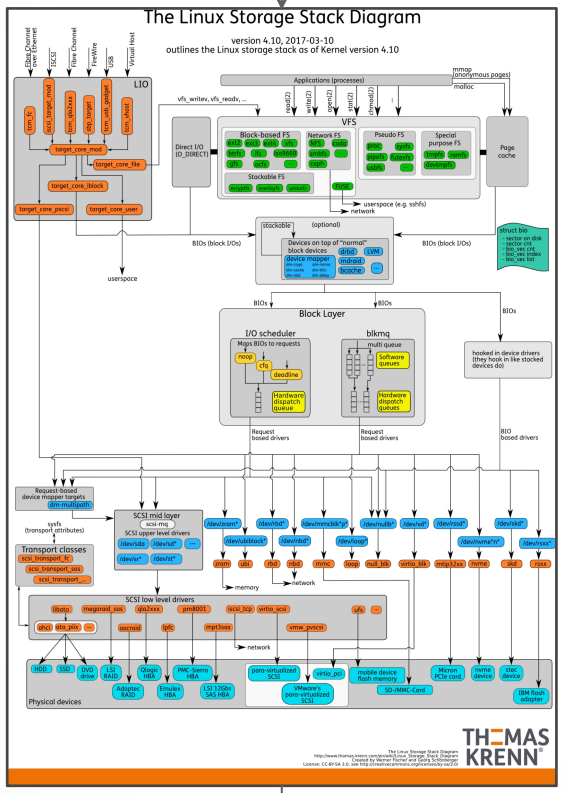
pynvme (unit: us)

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



Performance: design

FIO

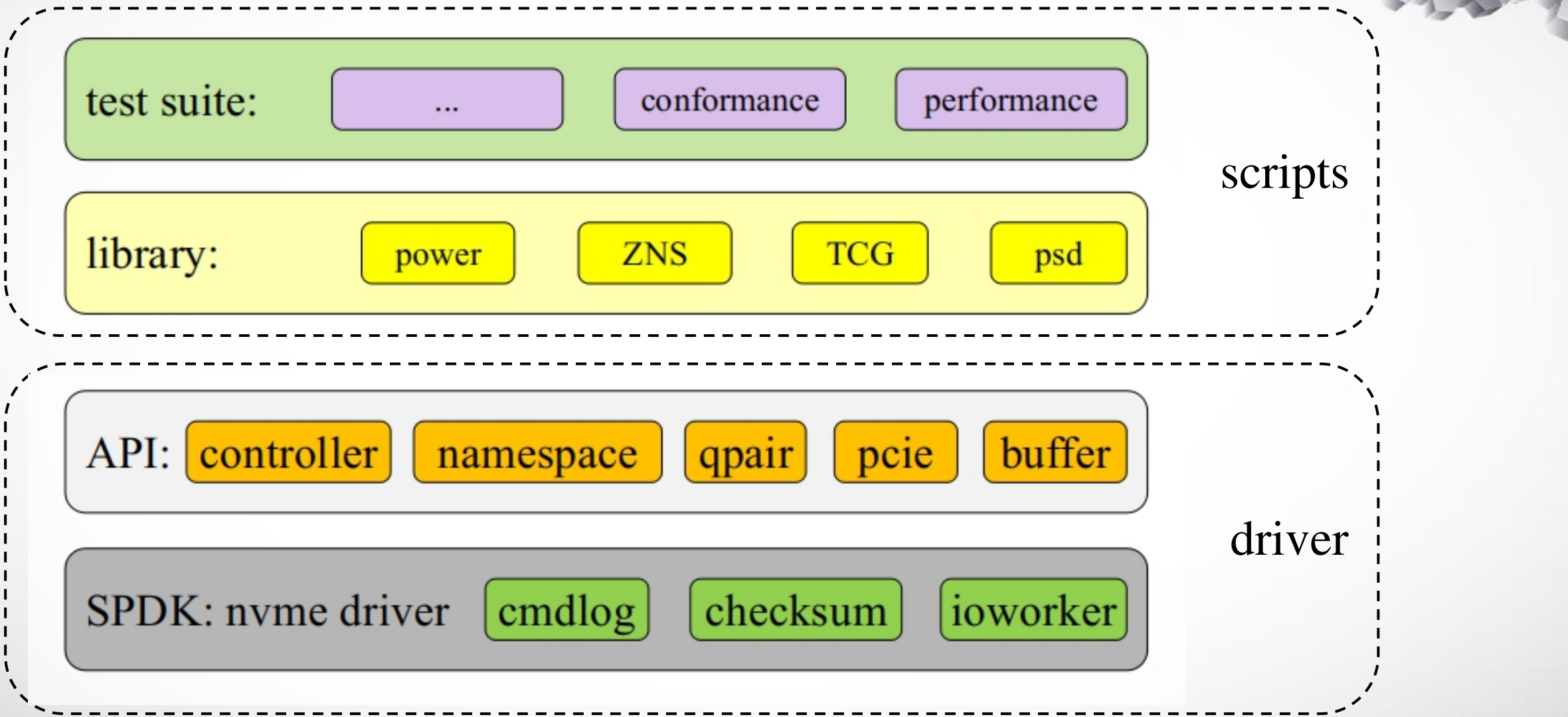


DUT

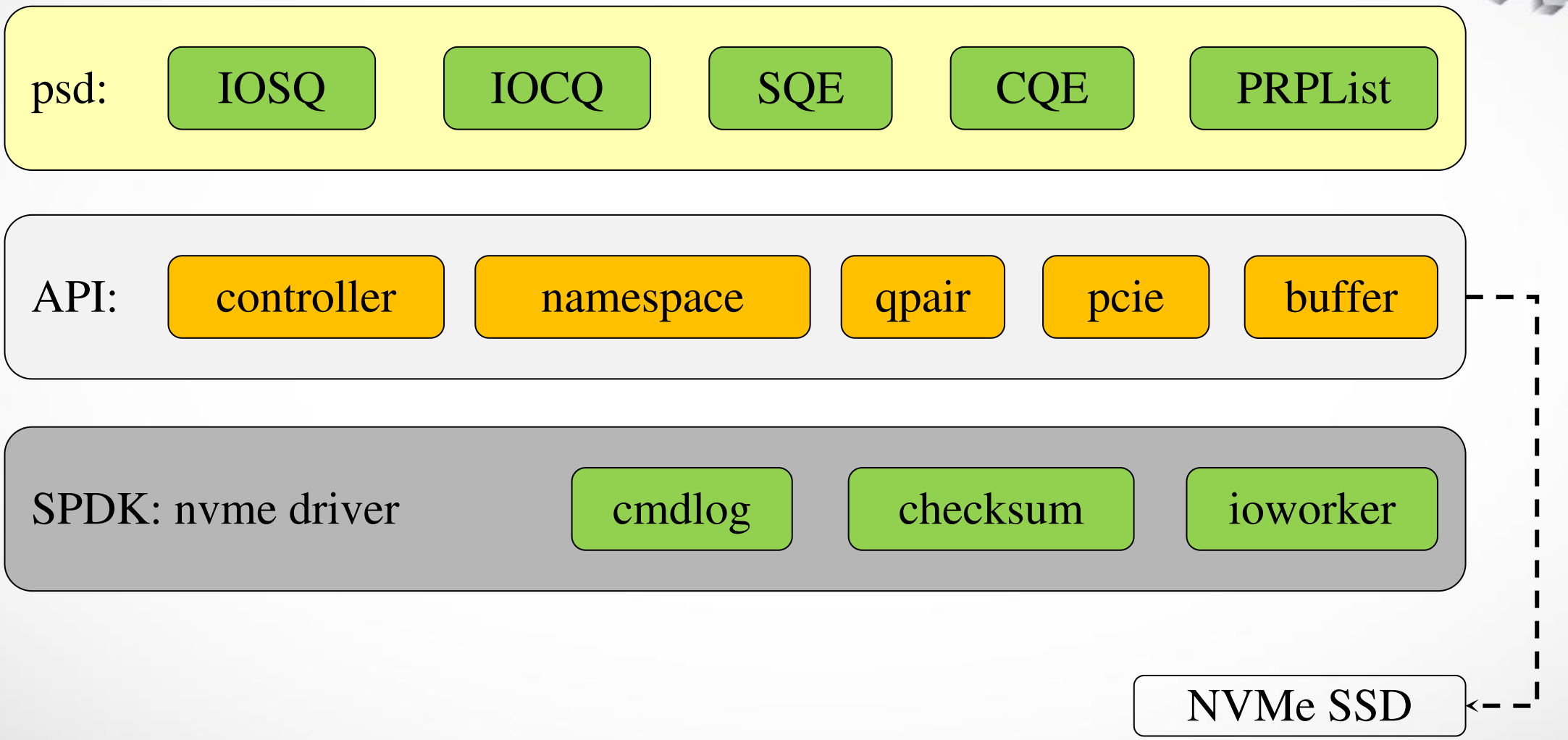
pynvme

NVMe driver
in userspace

Architecture



psd: Python Space Driver



Open Ecosystem

Cython

pytest

Visual Studio Code

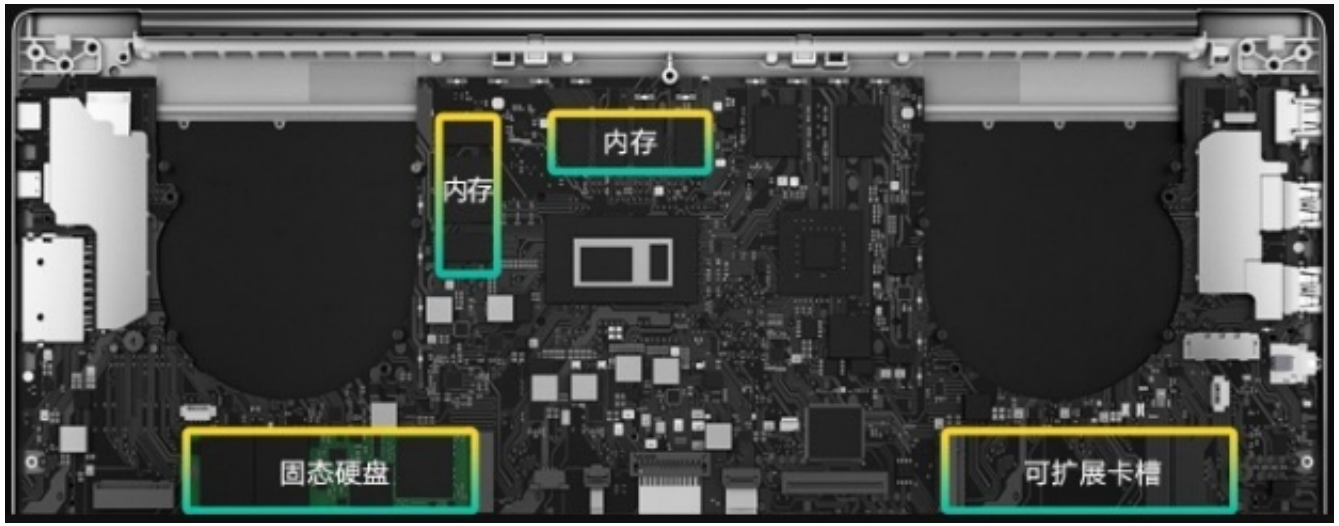
Jenkins

matplotlib

GitHub

Flexible Hardware Configuration

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



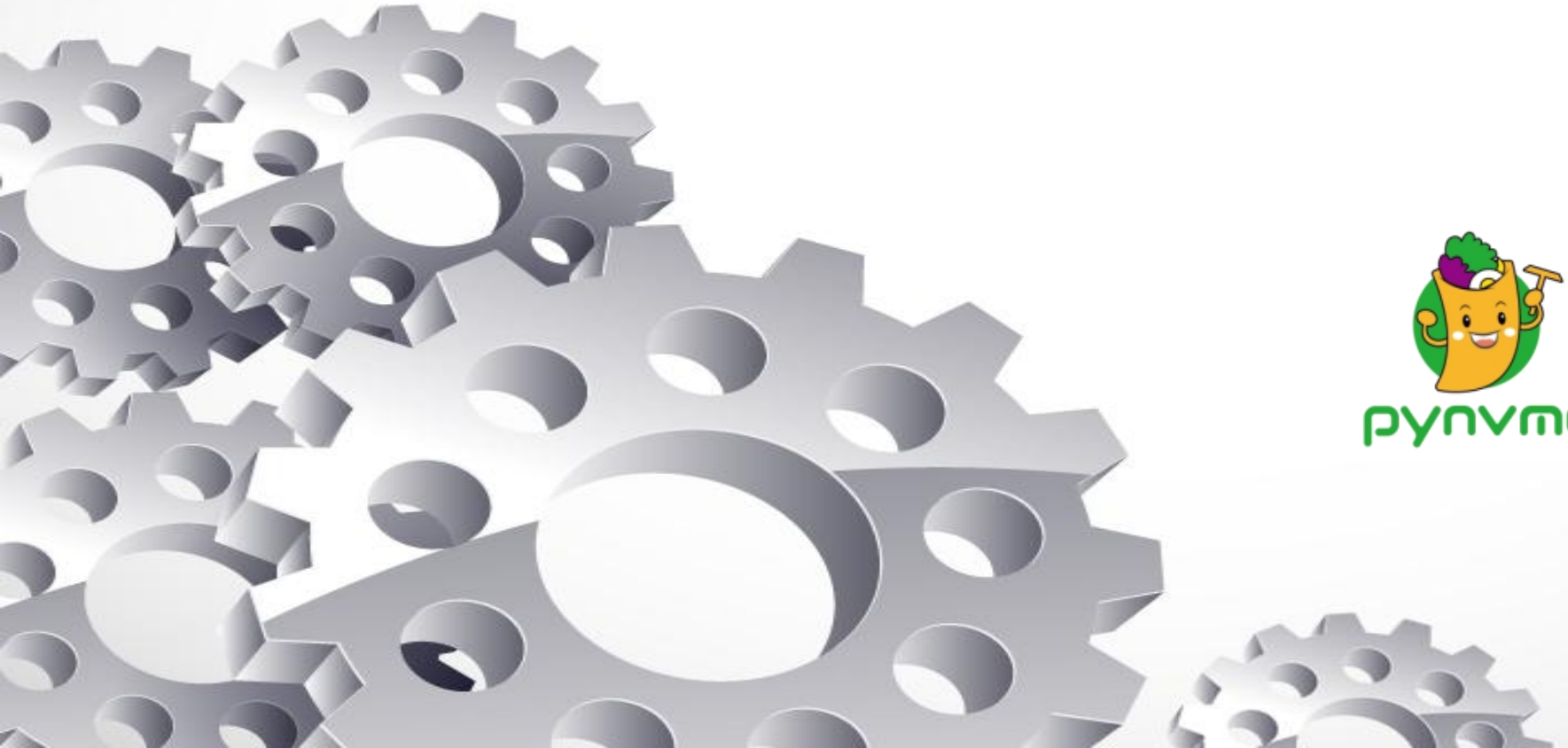
Install and Test

> git clone https://github.com/pynvme/pynvme
> cd pynvme
> ./install.sh

> make setup
> make test TESTS="driver_test.py::test_ioworker_iops_multiple_queue[1]"

> make test TESTS=scripts/stress/endurance_test.py::test_replay_jedec_client_trace pciaddr=02:00.0
> make test TESTS=scripts/stress/endurance_test.py::test_replay_jedec_client_trace pciaddr=172.168.5.44

Scripts



pynvme builds your own tests.

Test Scripts

```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'
```

```python
def test_quarch_dirty_power_cycle_single(nvme0, nvme0n1, subsystem, buf, verify):
    # get the unsafe shutdown count before test
    nvme0.getlogpage(2, buf, 512).waitdone()
    orig_unsafe_count = buf.data(159, 144)
    logging.info("unsafe shutdowns: %d" % orig_unsafe_count)
    assert verify == True

    # 128K random write
    cmdlog_list = [None]*1000
    with nvme0n1.ioworker(io_size=256,
                          lba_random=True,
                          read_percentage=30,
                          region_end=256*1000*1000,
                          time=30,
                          qdepth=1024,
                          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()
    time.sleep(0)
    nvme0.reset()
```

Test Scripts: dirty power cycle

- Quarch
  - PCIe Card Module
  - Torridon Interface Kit
  - PAM
- 3-rd party fixtures supported

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

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

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/

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)
```

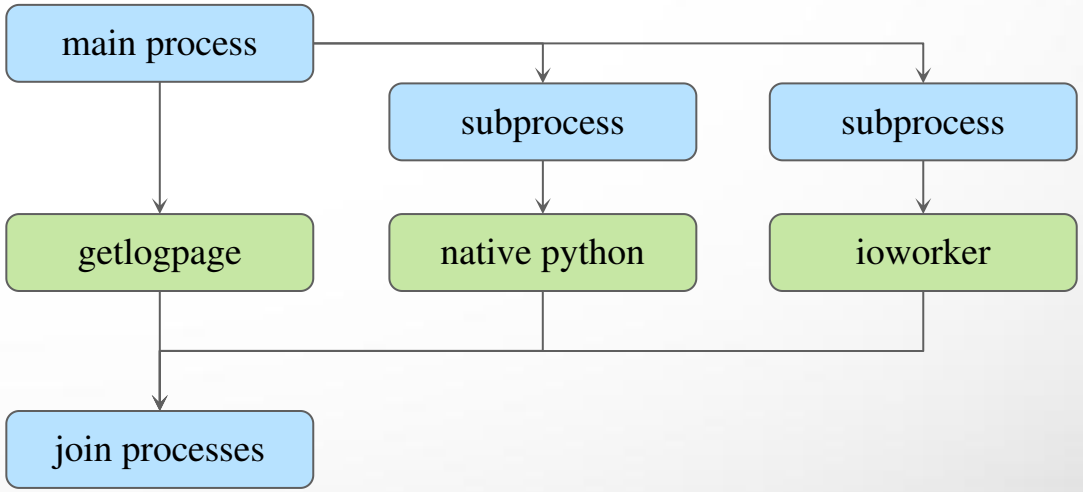
```
def test_ioworker_simplified(nvme0n1):
    nvme0n1.ioworker(io_size=2, time=2).start().close()
```

```
698 def test_write_before_power_cycle(nvme0, subsystem):
699     cq = IOCQ(nvme0, 1, 128, PRP(2*1024))
700     sq = IOSQ(nvme0, 1, 128, PRP(8*1024), cqid=1)
701
702     #burst write
703     for i in range(127):
704         cmd = SQE(1, 1)
705         buf = PRP(512, ptype=32, pvalue=i)
706         cmd.prp1 = buf
707         cmd[10] = i
708         sq[i] = cmd
709
710     # write 127 512byte at one shot
711     sq.tail = 127
```

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: power/reset events

```
188  # PCIe, different of power states and resets
189  def test_power_and_reset(pcie, nvme0, subsystem):
190      pcie.aspm = 2              # ASPM L1
191      pcie.power_state = 3       # PCI PM D3hot
192      pcie.aspm = 0
193      pcie.power_state = 0
194
195      nvme0.reset()              # controller reset: CC.EN
196      nvme0.getfeatures(7).waitdone()
197
198      pcie.reset()               # PCIe reset: hot reset, TS1, TS2
199      nvme0.reset()              # reset controller after pcie reset
200      nvme0.getfeatures(7).waitdone()
201
202      pcie.flr()                 # PCIe function level reset
203      nvme0.reset()              # reset controller after pcie reset
204      nvme0.getfeatures(7).waitdone()
205
206      subsystem.reset()          # NVMe subsystem reset: NSSR
207      nvme0.reset()              # controller reset: CC.EN
208      nvme0.getfeatures(7).waitdone()
209
210      subsystem.power_cycle(10)  # power cycle NVMe device: cold reset
211      nvme0.reset()              # controller reset: CC.EN
212      nvme0.getfeatures(7).waitdone()
```

```
42  @pytest.mark.parametrize("ps", [4, 3, 2, 1, 0])
43  def test_format_at_power_state(nvme0, nvme0n1, ps):
44      nvme0.setfeatures(0x2, cdw11=ps).waitdone()
45      assert nvme0n1.format(ses=0) == 0
46      assert nvme0n1.format(ses=1) == 0
47      p = nvme0.getfeatures(0x2).waitdone()
48      assert p == ps
```

```
191  @pytest.mark.parametrize("aspm", [0, 2])
192  @pytest.mark.parametrize("gen", [1, 2, 3, 2, 1, 1, 2, 3])
193  def test_pcie_link_speed(pcie, nvme0, nvme0n1, gen, aspm):
194      linkctr2_addr = pcie.cap_offset(0x10)+0x30
195      linkctr2 = pcie.register(linkctr2_addr, 4)
196      logging.info(linkctr2)
197
198      pcie[linkctr2_addr] = (linkctr2 & 0xf0) | gen
199      logging.info(pcie.register(linkctr2_addr, 4))
200      pcie.reset()
201      nvme0.reset()
202      test_pcie_read_bandwidth(nvme0n1)
203      test_pcie_link_control_aspm(nvme0, pcie, aspm)
```

Test Scripts: NVMe initialization for WRR

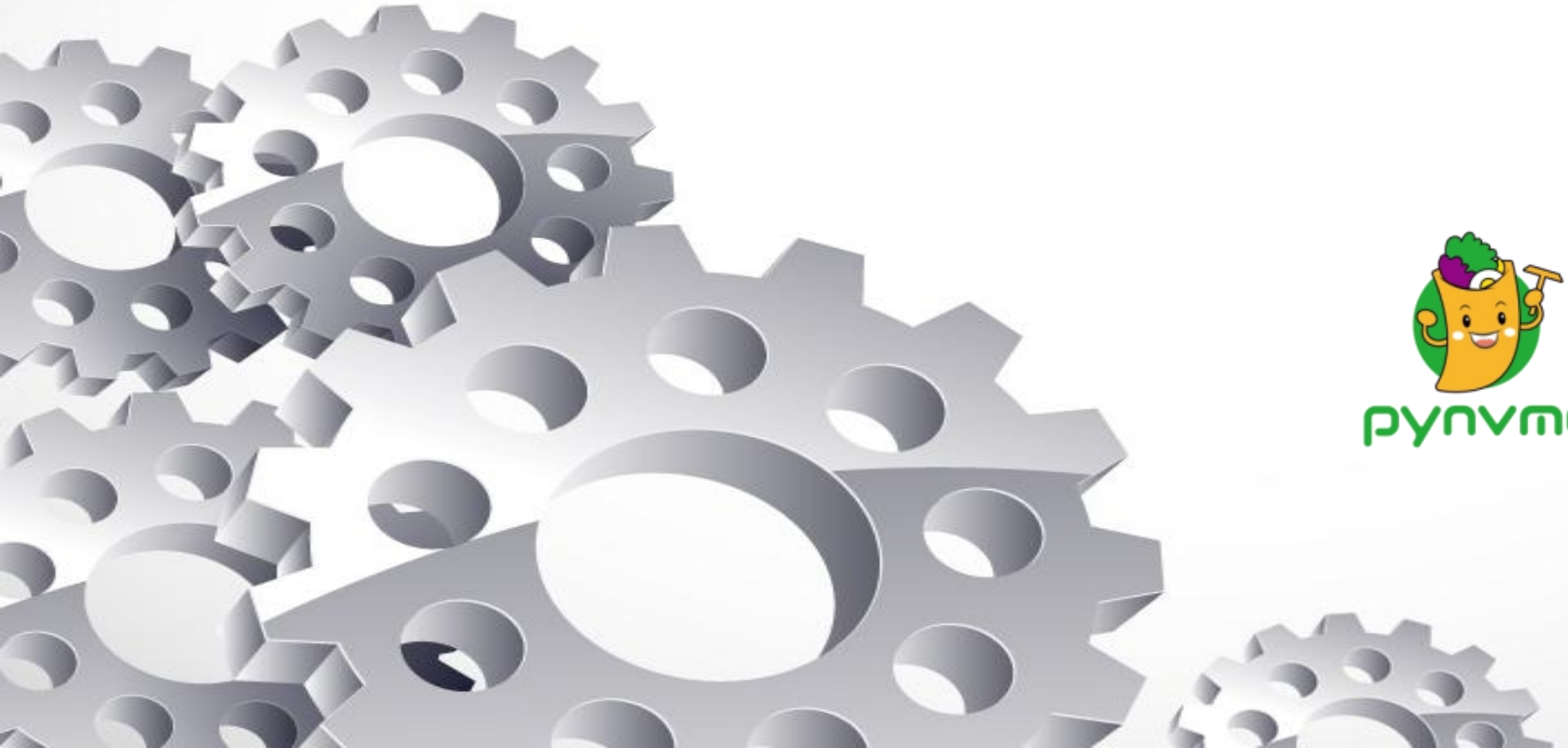
```
9   def nvme_init_wrr(nvme0):
10      logging.info("user defined nvme init")
11
12      nvme0[0x14] = 0
13      while not (nvme0[0x1c]&0x1) == 0: pass
14
15      # 3. set admin queue registers
16      nvme0.init_adminq()
17
18      # 4. set register cc
19      if (nvme0.cap>>17) & 0x1:
20          logging.info("set WRR arbitration")
21          nvme0[0x14] = 0x00460800
22      else:
23          nvme0[0x14] = 0x00460000
24
25      # 5. enable cc.en
26      nvme0[0x14] = nvme0[0x14] | 1
27
28      # 6. wait csts.rdy to 1
29      while not (nvme0[0x1c]&0x1) == 1: pass
30
31      # 7. identify controller
32      nvme0.identify(Buffer(4096)).waitdone()
33
34      # 8. create and identify all namespace
35      nvme0.init_ns()
36
37      # 9. set/get num of queues
38      nvme0.setfeatures(0x7, cdw11=0x00ff00ff).waitdone()
39      nvme0.getfeatures(0x7).waitdone()
```

```
42  def test_ioworker_with_wrr(pcie):
43      nvme0 = Controller(pcie, nvme_init_func=nvme_init_wrr)
44
45      if (nvme0.cap>>17) & 0x1 == 0:
46          pytest.skip("WRR is not supported")
47
48      # 8:4:2
49      assert nvme0[0x14] == 0x00460801
50      nvme0.setfeatures(1, cdw11=0x07030103).waitdone()
51      cdw0 = nvme0.getfeatures(1, cdw11=0x7313).waitdone()
52      assert cdw0 == 0x07030103
53
54      nvme0n1 = Namespace(nvme0, 1, 0x10000)
55      nvme0n1.format(512)
56
57      l = []
58      for i in range(3):
59          a = nvme0n1.ioworker(io_size=8,
60                               read_percentage=100,
61                               region_end=0x10000,
62                               qprio=i+1,
63                               time=10)
64          l.append(a)
```

ioworker: Python API

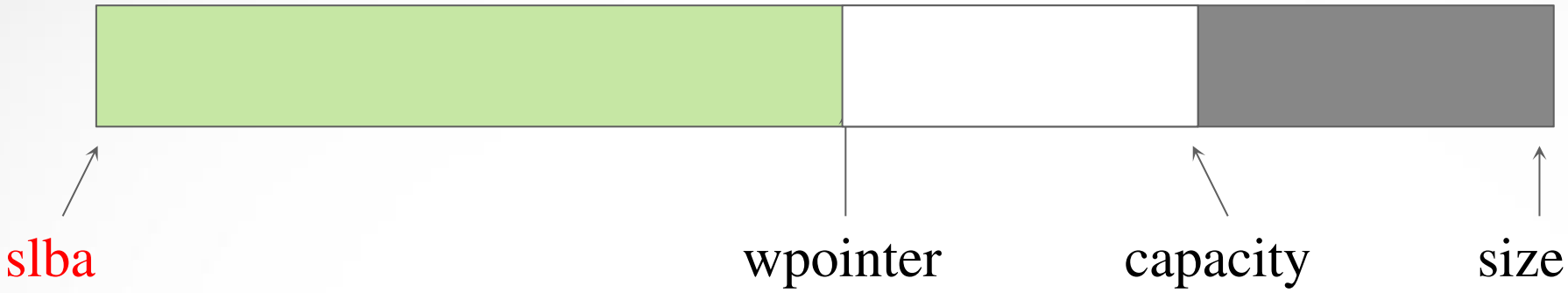
```
774  def test_ioworker_with_temperature(nvme0, nvme0n1, buf):
775      with nvme0n1.ioworker(io_size=256,
776                            time=30,
777                            op_percentage={0:10,  # flush
778                                           2:60,  # read
779                                           9:30}), \
780           nvme0n1.ioworker(io_size=8,
781                            time=30,
782                            op_percentage={0:10,  # flush
783                                           9:10,  # trim
784                                           1:80}):# write
785          for i in range(40):
786              time.sleep(1)
787              nvme0.getlogpage(0x02, buf, 512).waitdone()
788              ktemp = buf.data(2, 1)
789              from pytemperature import k2c
790              logging.info("temperature: %0.2f degreeC" %
791                           k2c(ktemp))
792
```

ZNS



pynvme builds your own tests.

Zone Class



Zone.state

- Empty
- Implicitly Opened
- Explicitly Opened
- Closed
- Read Only
- Full
- Offline

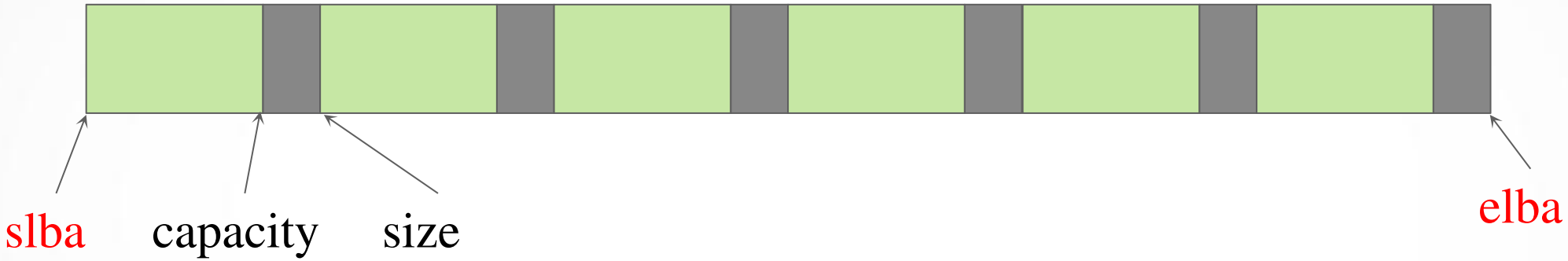
Zone actions: applied to all physical zones in a super zone

- close()
- finish()
- open()
- reset()
- offline()
- set_descriptor_extension()

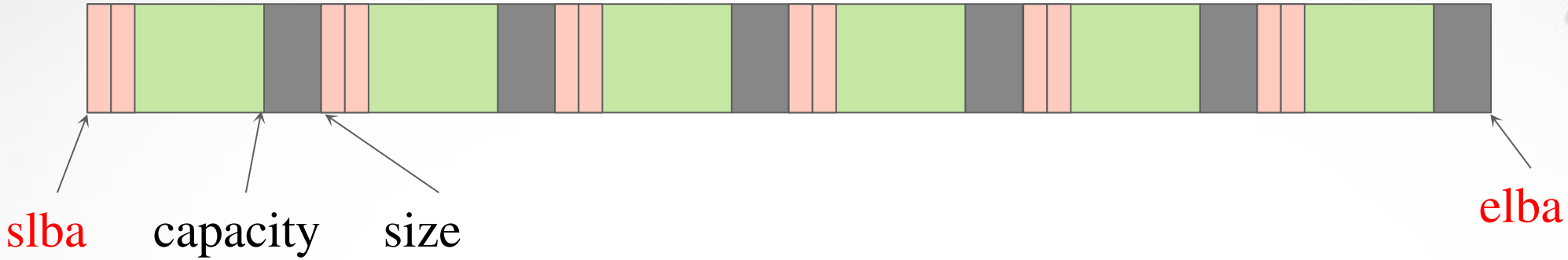
IO:

- read(offset)
- write(offset)
- ioworker()

Super Zone



ioworker: fill super zone

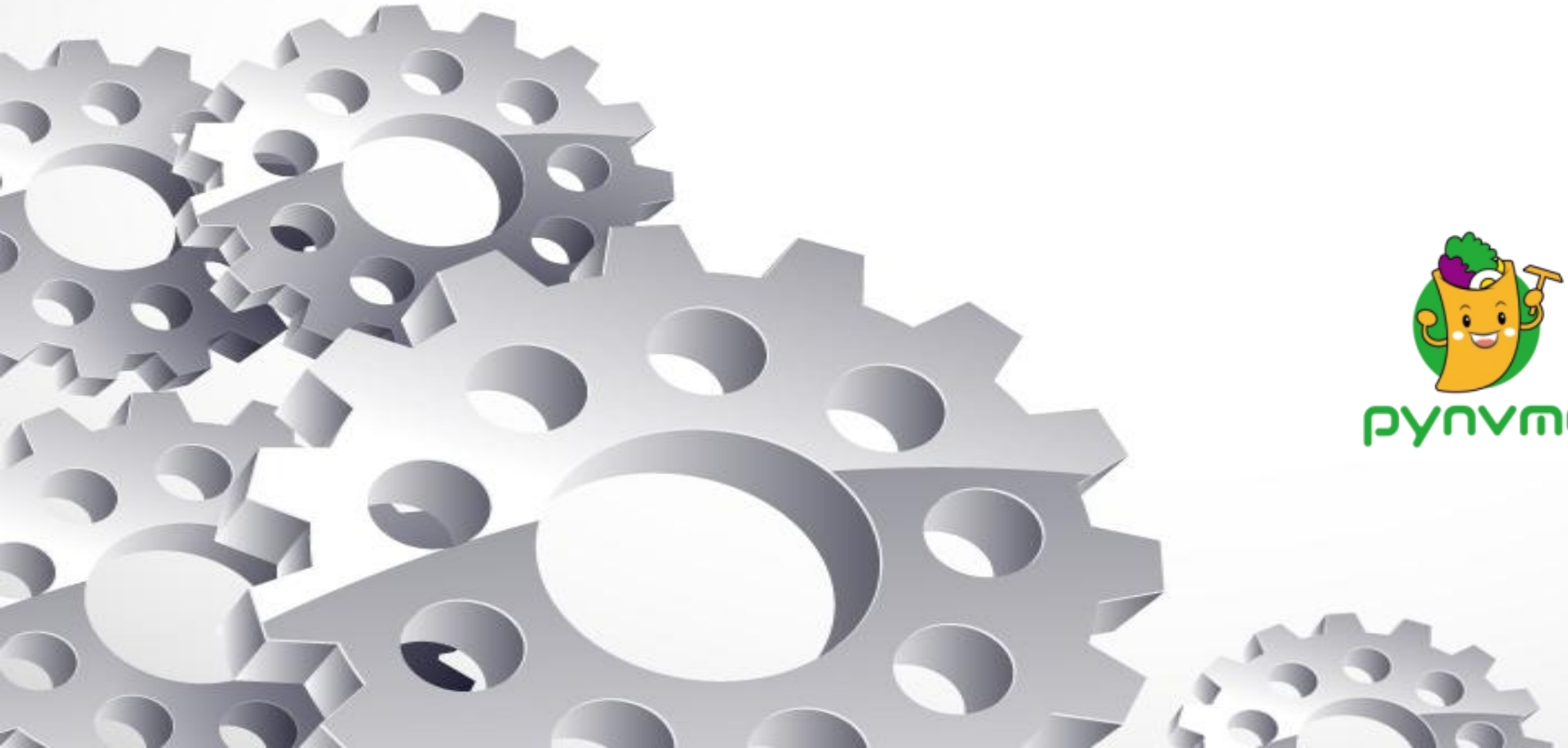


- √ maximize opened zone count
- √ increase write bandwidth
- √ avoid the LBA holes between capacity and size
- √ leverage the step parameter of ioworker

notes:
1. enough qdepth: 8*(physical zone count)
2. 100% write in one ioworker. Mixed read in another ioworker.

```
def test_zns_write_super_zone_basic(nvme0n1, buf, qpair, nzones=1):
    Zone(qpair, nvme0n1, 0, 0x4000*nzones).reset().\
        ioworker(io_size=32, io_count=384*nzones, qdepth=8*nzones).start().close()
```

Services



pynvme builds your own tests.

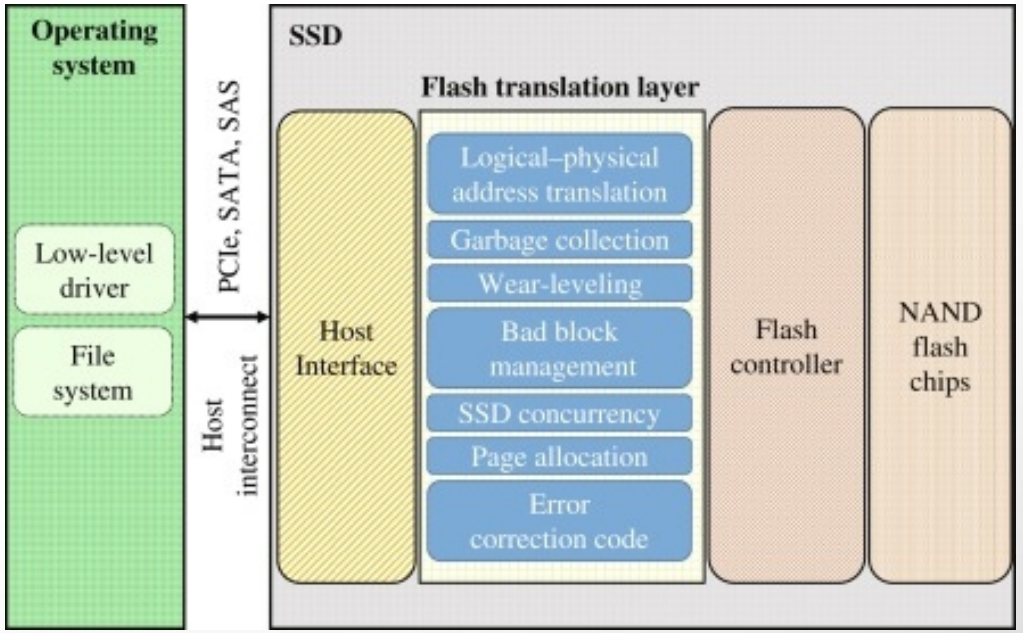
Services

```
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()
```

JEDEC

SNIA

nvm EXPRESS



HMB example

Test 3.4 – Host Memory Buffer (M) .....
Case 1: Proper Structure (M).....
Case 2: Configuration (FYI).....
Case 3: Reset Persistent (FYI).....
Case 4: Enable HMB when Already Enabled (FYI) .....
Case 5: Disable HMB when Already Disabled (FYI) .....

5

v.s.

32

E43 | fx | HMB_047

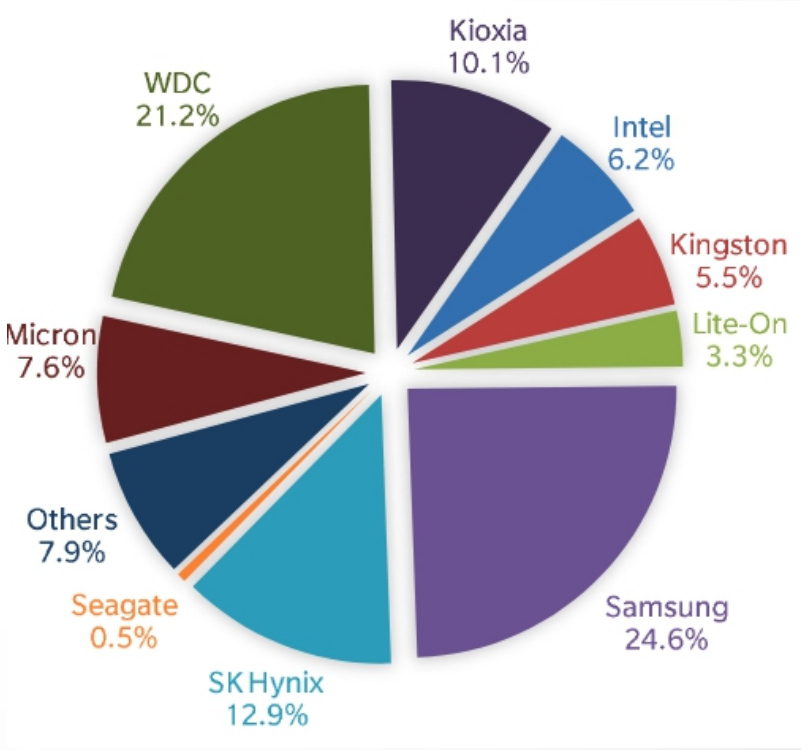
	A	B	C	D	
41	Test_HMB_028	test_HMB_sanitize_format	verify sanitize and user data erase command can be passed when enabling hmb	1. Host send “set feature command”to enable hmb,check cq 2. start ioworker without error 3. host send "sanitize command" 4. run ioworker without error 5. host send " user data erase command " 6. start ioworker, there is no error 7. host send "TCG operation " ,there is no error	HMB_042 HMB_043 HMB_044
42	Test_HMB_029	test_HMB_stress	enable and disable hmb stress	1. Host send “set feature command”to enable hmb,check cq 2. start ioworker 3. Host send “set feature command”to disable hmb,check cq 4. run ioworker 5. repeat step1-4 many times 6. open the thread to do ioworker 7. Host send “set feature command”to enable hmb,check cq 8. Host send “set feature command”to disable hmb,check cq 9. repeat step7-8 many times	HMB_045 HMB_046
43	Test_HMB_030	test_HMB_out_of_bounds	verify memory access out of bounds	1. get three Buffers 2. The middle of buffer is used to hmb 3. Host send “set feature command”to enable hmb,check cq 4. run ioworker 5. check the pattern in the first and end buffer has not changed	HMB_047
44	Test_HMB_031	test_HMB_data_consistency	verify data consistency	1. Host send “set feature command”to enable hmb,check cq 2. write data 3. Host send “set feature command”to disable hmb,check cq 4. verify data at step 2 5. write data 6. Host send “set feature command”to enable hmb,check cq 7. verify data at step 5	HMB_048 HMB_049
45	Test_HMB_032	test_HMB_with_command	verify admin command and nvm command without error when	1. Host send “set feature command”to enable hmb,check cq 2. Host send admin command(like: AER/set feature..) and nvm comman	HMB_050
46					

GYTech

- PyNVMe3 is derived from pynvme with more functions and better commercial services. https://pynv.me/3/

- the gap between SSD development and test
  - many SSD controller and product vendors
  - SSD fast growing

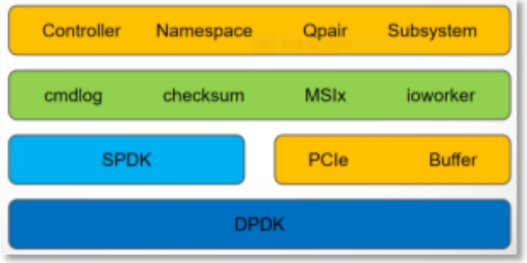
- the gap between SSD vendors and customers
  - 3rd party test suites and technology supplier



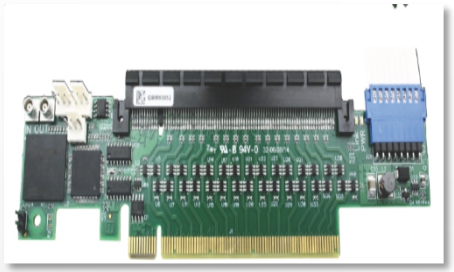
Values

Ecosystem

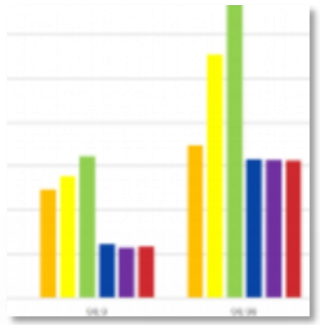
Scripts



Expendability



Hardware



Performance

Service

pynvme builds your own tests.

https://pynv.me/3/

Thanks!