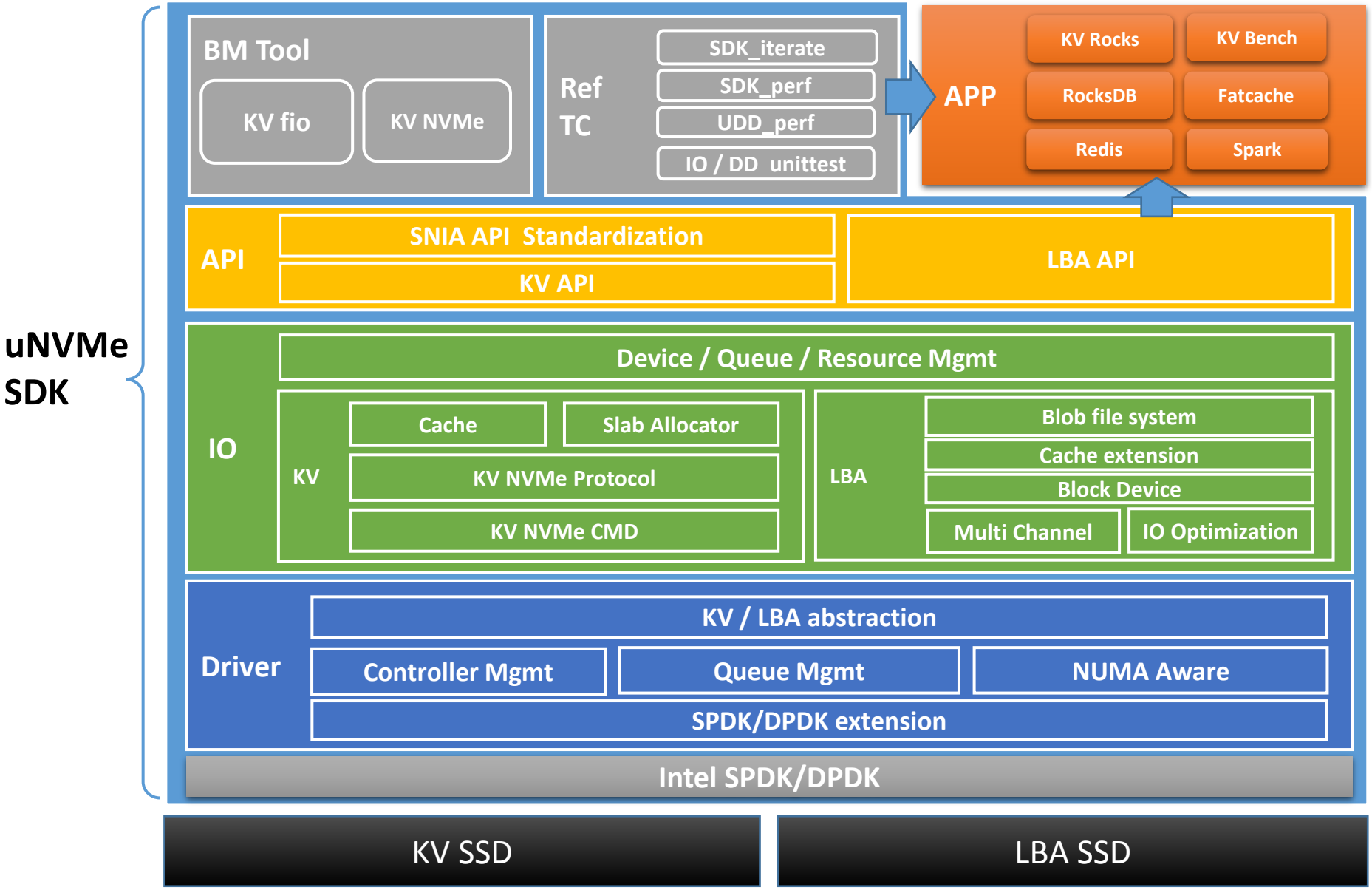
uNVMe-Blobfs

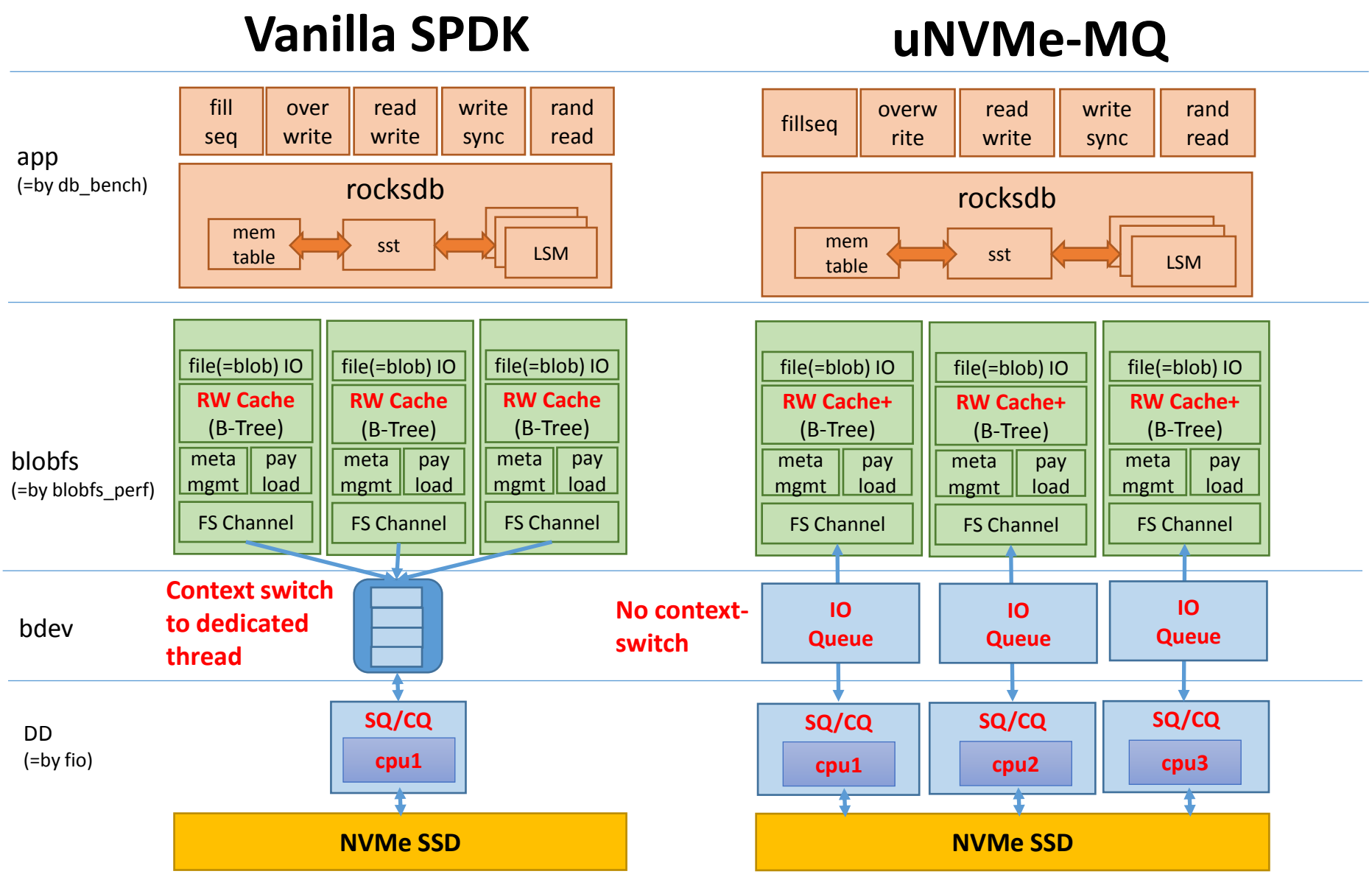
(OpenMPDK-uNVMe)

Kyungsan Kim / SW Dev Team / Memory Business Unit

uNVMe SDK – Dec, 2018

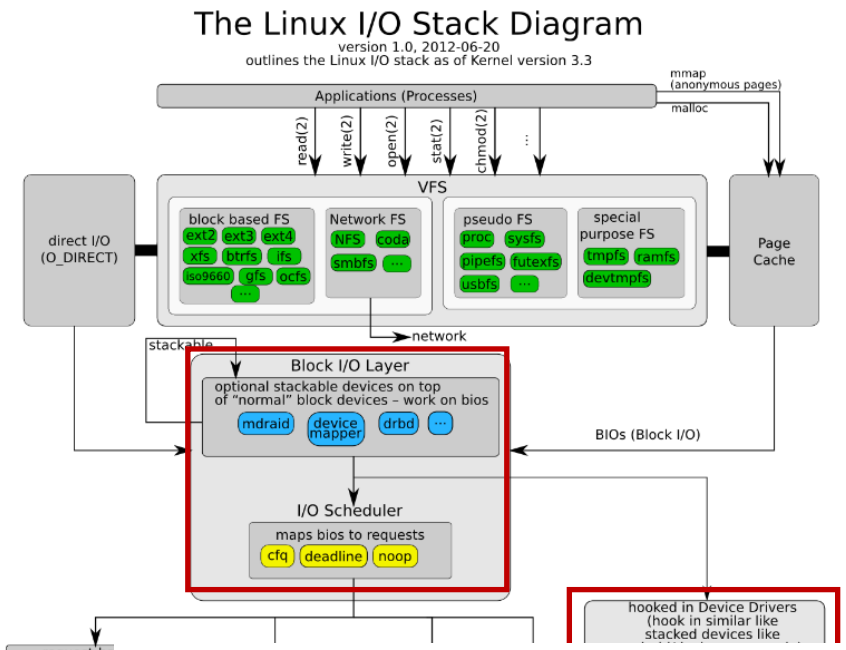


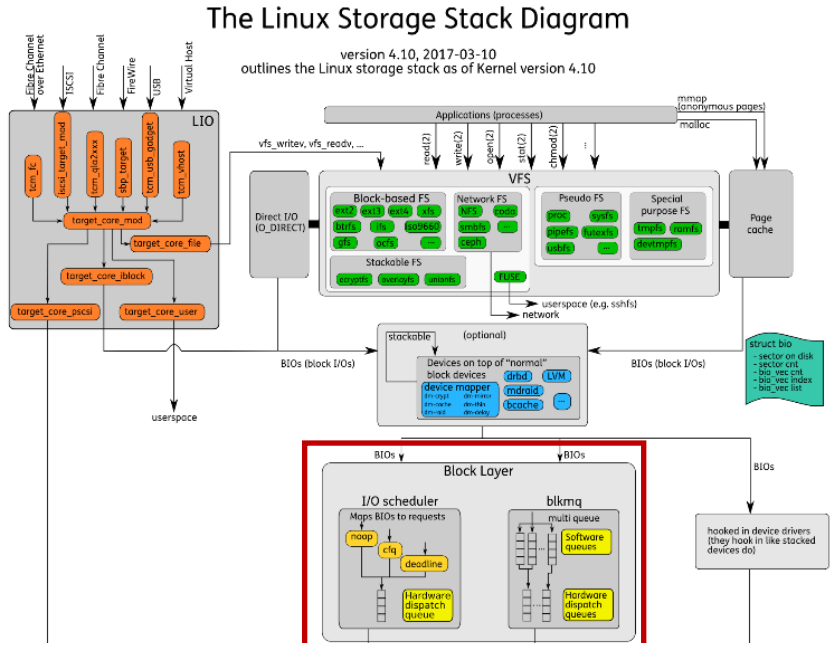
uNVMe-Phase2 Goal



History of Linux Kernel Block layer

- Around 2013 Oct, Linux kernel introduced a big change in storage IO stack, ***block-mq,*** that reflects prevalent of multi-core system with high speed storage

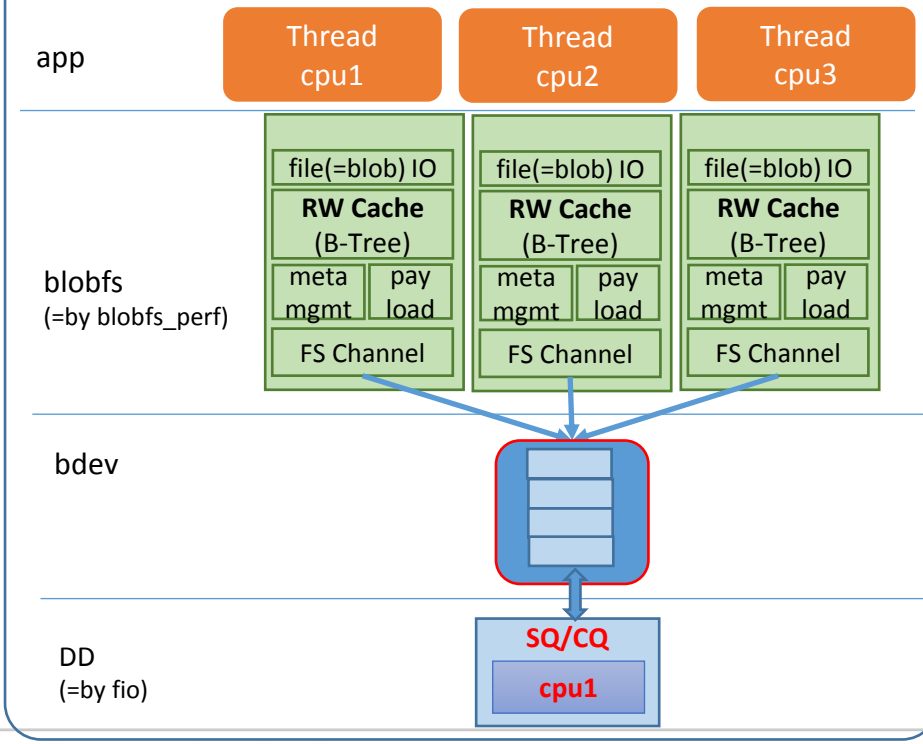




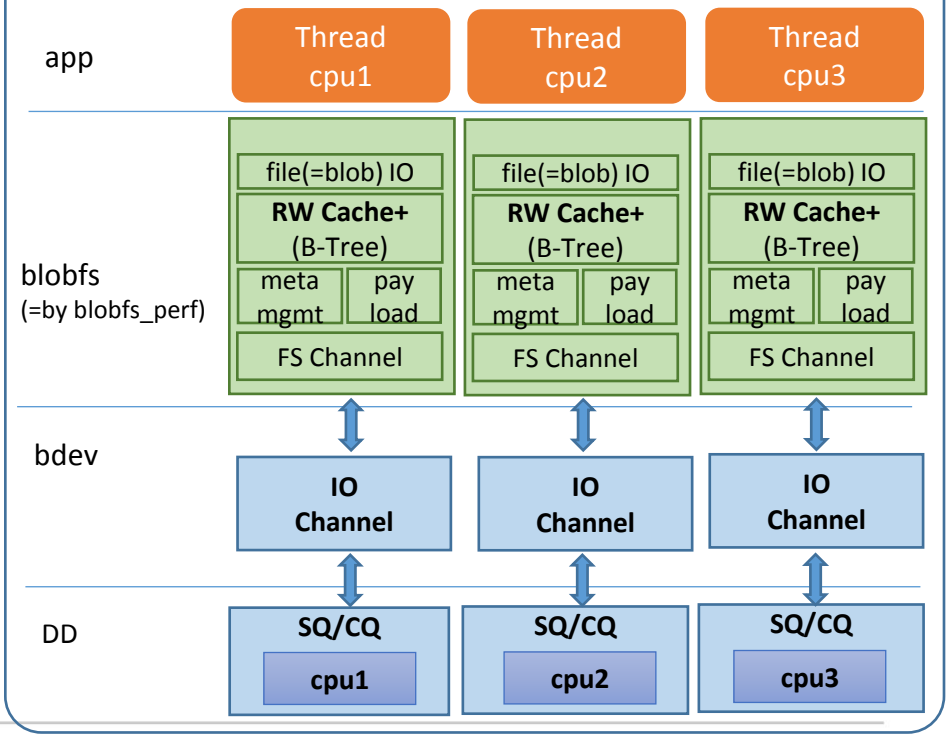
Driver	Device Name	Supported Devices	blk-mq Since Kernel Version
null_blk	/dev/nullb*[6]	none (test drivers)	3.13 (git commit)
virtio-blk	/dev/vd*	Virtual guest drivers (e.g. under KVM[7][8])	3.13 (git commit)
mtip32xx	/dev/rssd*	Micron RealSSD PCIe	3.16 (git commit)
scsi (scsi_mq)	/dev/sd*	e.g. SAS and SATA SSDs/HDDs	3.17 (git commit)
NVMe	/dev/nvme*	e.g. Intel SSD DC P3600 DC P3700 Series[9]	3.19 (git commit)
rbd	/dev/rdb*	RADOS Block Device (Ceph)	4.0 (git commit)
ubi/block	/dev/ubiblock*		4.0 (git commit)
loop	/dev/loop*	Loopback-Device	4.0 (git commit)
dm / dm-mpath		request-based device mapper targets (derzeit ist dies ausschließlich dm-multipath)	planned for 4.1[10]

Contribution over SPDK(1) – MQ (Multi-queue)

- **SPDK Limitation**
- blobfs and bdev layer has 3 limitations in that it is still designed to utilize **single block layer queue on a dedicated CPU.** Therefore,
- ✓ 1) Request queue contention : on every single IO request for queue manipulation.
- ✓ 2) Scalability : doesn't make use of multi core and multi-queue SSD capability, unlikely current kernel IO stack.
- ✓ 3) Remote memory access : single cpu on bdev layer forces remote memory access across CPU sockets. Hence, Blobfs is not NUMA-aware.



- **uNVMe MQ**
- ✓ Support **per CPU queue** to avoid the architectural inefficiencies
- ✓ On this release, we have applied MQ only on **READ** requests

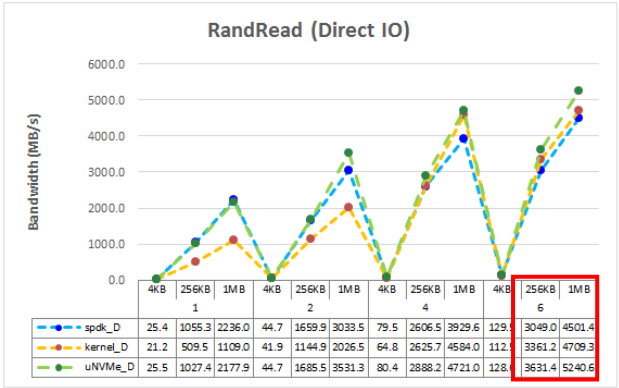


Contribution over SPDK(1) – MQ (Multi-queue)

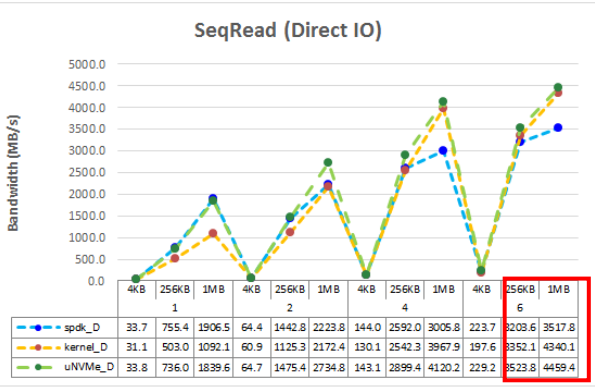
uNVMe-MQ

- developing multi queue block layer over SPDK blobfs
- Around 30**% gain on Seq/Rand Read workload** at Blob filesystem layer (*RocksDB db_bench*)
- Applying uNVMe-MQ on rocksdb workload on many testbed

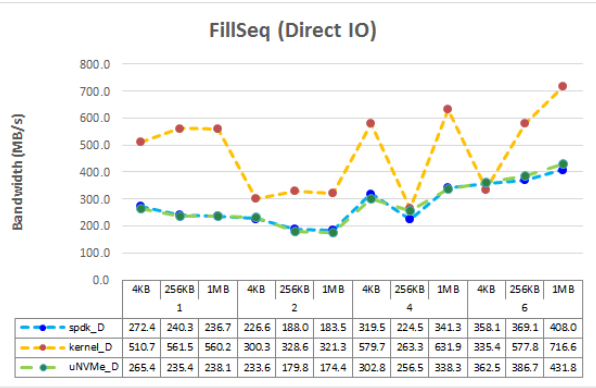
<Rocksdb Performance comparison - SPDK, kernel and uNVMe MQ Direct IO>



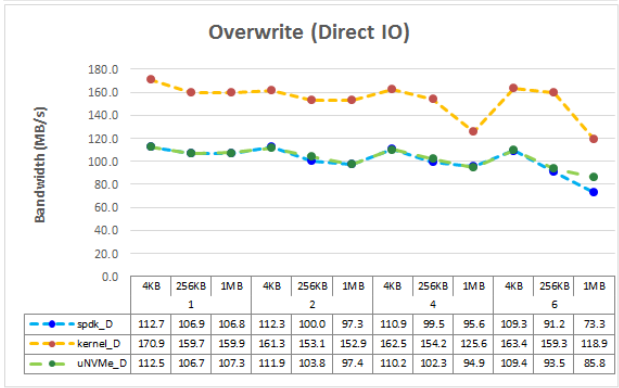
	4KB	256KB	1MB	4KB	256KB	1MB	4KB	256KB	1MB	4KB	256KB	1MB
	1			2			4			6		
spdk_D	25.4	1055.3	2236.0	44.7	1659.9	3033.5	79.5	2606.5	3929.6	129.1	3049.0	4501.4
kernel_D	21.2	509.5	1109.0	41.9	1144.9	2026.5	64.8	2625.7	4584.0	112.9	3361.2	4709.3
uNVMe_D	25.5	1027.4	2177.9	44.7	1685.5	3531.3	80.4	2888.2	4721.0	128.0	3631.4	5240.6



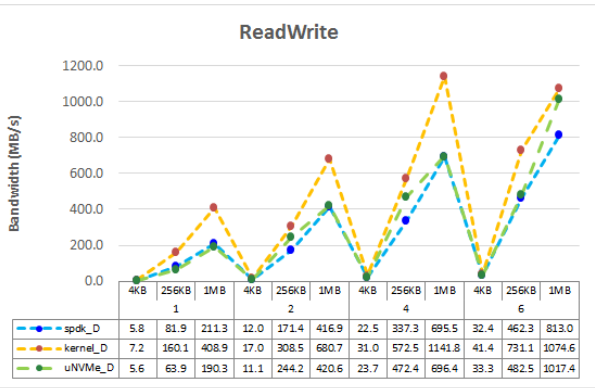
	4KB	256KB	1MB	4KB	256KB	1MB	4KB	256KB	1MB	4KB	256KB	1MB
	1			2			4			6		
spdk_D	33.7	755.4	1906.5	64.4	1442.8	2223.8	144.0	2592.0	3005.8	223.7	3203.6	3517.8
kernel_D	31.1	503.0	1092.1	60.9	1125.3	2172.4	130.1	2542.3	3967.9	197.6	3352.1	4340.1
uNVMe_D	33.8	736.0	1839.6	64.7	1475.4	2734.8	143.1	2899.4	4120.2	229.2	3523.8	4459.4



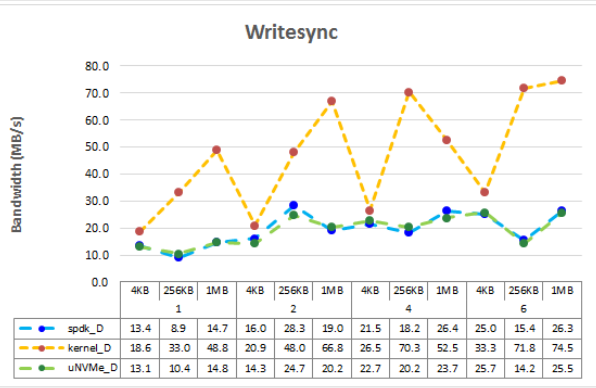
	4KB	256KB	1MB	4KB	256KB	1MB	4KB	256KB	1MB	4KB	256KB	1MB
	1			2			4			6		
spdk_D	272.4	240.3	236.7	226.6	188.0	183.5	319.5	224.5	341.3	358.1	369.1	408.0
kernel_D	510.7	561.5	560.2	300.3	328.6	321.3	579.7	263.3	631.9	335.4	577.8	716.6
uNVMe_D	265.4	235.4	238.1	233.6	179.8	174.4	302.8	256.5	338.3	362.5	386.7	431.8



	4KB	256KB	1MB	4KB	256KB	1MB	4KB	256KB	1MB	4KB	256KB	1MB
	1			2			4			6		
spdk_D	112.7	106.9	106.8	112.3	100.0	97.3	110.9	99.5	95.6	109.3	91.2	73.3
kernel_D	170.9	159.7	159.9	161.3	153.1	152.9	162.5	154.2	125.6	163.4	159.3	118.9
uNVMe_D	112.5	106.7	107.3	111.9	103.8	97.4	110.2	102.3	94.9	109.4	93.5	85.8



	4KB	256KB	1MB	4KB	256KB	1MB	4KB	256KB	1MB	4KB	256KB	1MB
	1			2			4			6		
spdk_D	5.8	81.9	211.3	12.0	171.4	416.9	22.5	337.3	695.5	32.4	462.3	813.0
kernel_D	7.2	160.1	408.9	17.0	308.5	680.7	31.0	572.5	1141.8	41.4	731.1	1074.6
uNVMe_D	5.6	63.9	190.3	11.1	244.2	420.6	23.7	472.4	696.4	33.3	482.5	1017.4



	4KB	256KB	1MB	4KB	256KB	1MB	4KB	256KB	1MB	4KB	256KB	1MB
	1			2			4			6		
spdk_D	13.4	8.9	14.7	16.0	28.3	19.0	21.5	18.2	26.4	25.0	15.4	26.3
kernel_D	18.6	33.0	48.8	20.9	48.0	66.8	26.5	70.3	52.5	33.3	71.8	74.5
uNVMe_D	13.1	10.4	14.8	14.3	24.7	20.2	22.7	20.2	23.7	25.7	14.2	25.5

System: i7-7700k 4core 8threads @4.50GHz , 16GB DRAM, Ubuntu 16.04.3(kernel 4.9.82), PM1725a 1.6TB
RocksDB: v5.6.1, 16B key, 3K/255KB/1023KB value, disable block cache, repeat 5 time on readseq, fillseq from N threads
SPDK: v18.04.1
- Please refer Evaluation Guide for Test configuration (thread = 1/2/4/6 and so on)

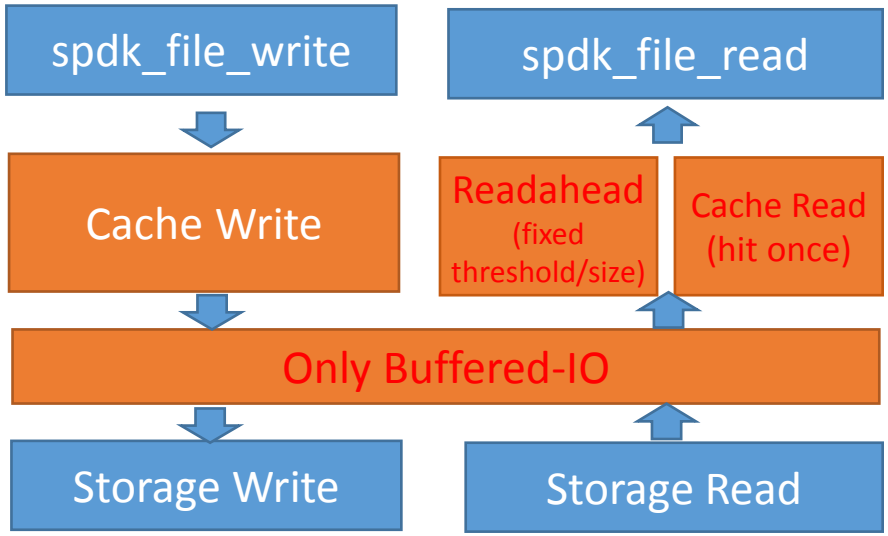
Contribution over SPDK(2) - Cache

- **SPDK Limitation**

✓ **Readahead**: only when the amount of read requests beyond **consecutive 128KB**, it is judged as sequential read and **prefetch 512KB** in size constantly, otherwise prefetch is never triggered

✓ **Reclaim Policy**: **when a cached data hits** on a read request, **it is reclaimed instantly**, thus when the read data is revisited, it necessarily leads to disk read

✓ **Buffered IO only**: blobfs **always occupies GB size cache** memory

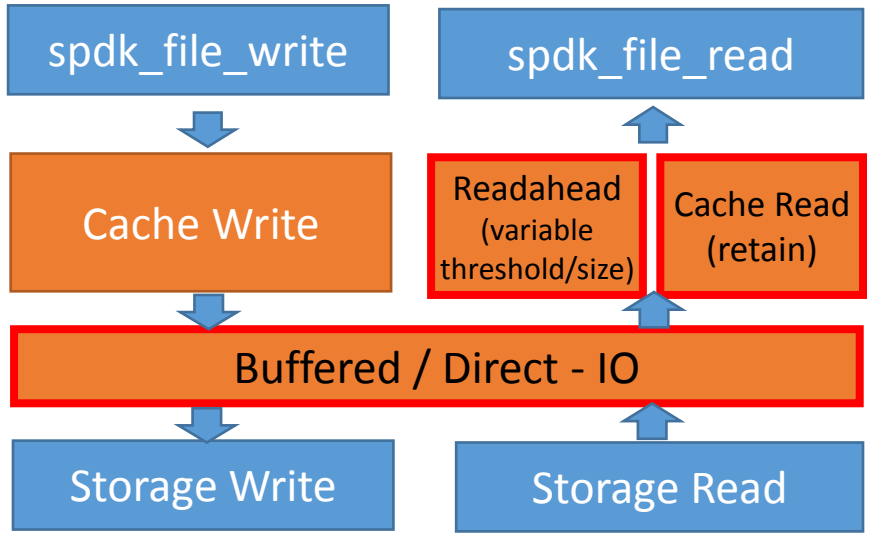


- **uNVMe-cache**

✓ **Readahead** : altered blobfs and provide an API to **configure best-fit prefetch threshold and size** depending on IO size out of an application

✓ **Reclaim Policy** : altered blobfs and provide an API to **determine reclaim policy**

✓ **Support Direct IO** : Altered blobfs and provide API to **support DIRECT IO** as well as Buffered IO as a selective way for cache supporting application

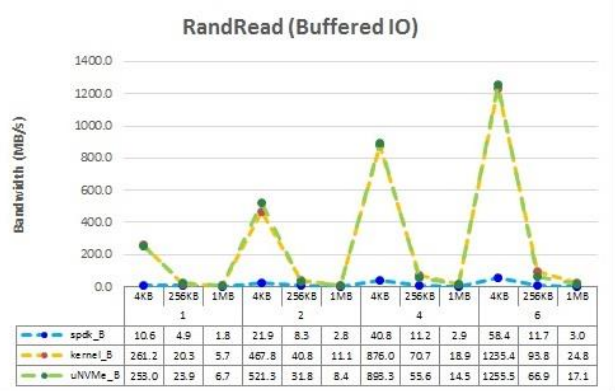


Contribution over SPDK(2) - Cache

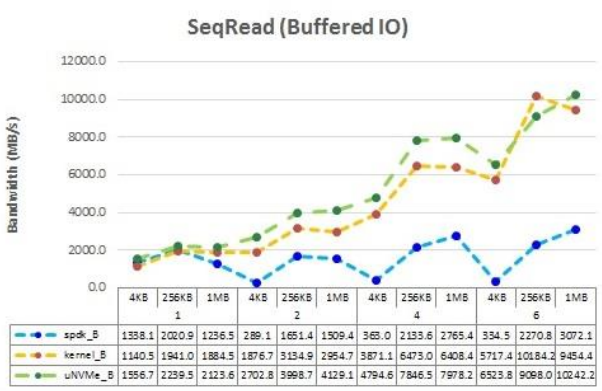
uNVMe read cache

- uNVMe-read cache shows performance gain against SPDK on db_bench workloads

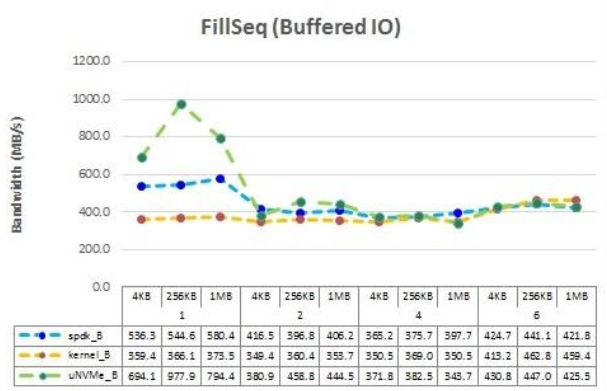
<Rocksdb Performance comparison - SPDK, kernel and uNVMe MQ Buffered IO>



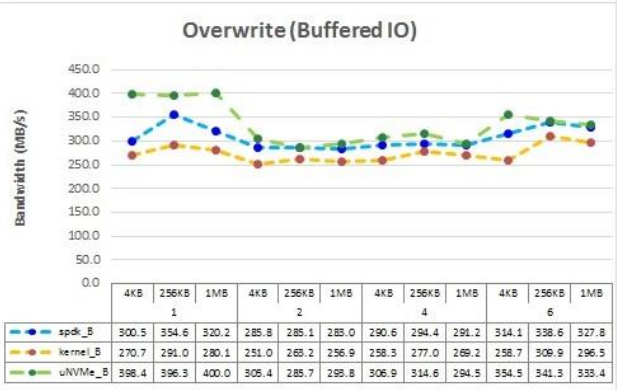
	4KB	256KB	1MB	4KB	256KB	1MB	4KB	256KB	1MB	4KB	256KB	1MB
	1			2			4			6		
spdk_B	10.6	4.9	1.8	21.9	8.3	2.8	40.8	11.2	2.9	58.4	11.7	3.0
kernel_B	261.2	20.3	5.7	467.8	40.8	11.1	876.0	70.7	18.9	1235.4	93.8	24.8
uNVMe_B	253.0	23.9	6.7	521.3	31.8	8.4	895.3	55.6	14.5	1255.5	66.9	17.1



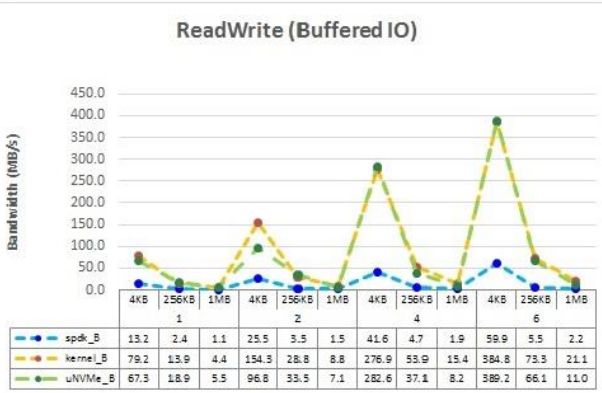
	4KB	256KB	1MB	4KB	256KB	1MB	4KB	256KB	1MB	4KB	256KB	1MB
	1			2			4			6		
spdk_B	1338.1	2020.9	1236.5	289.1	1651.4	1509.4	365.0	2133.6	2765.4	334.5	2270.8	3072.1
kernel_B	1140.5	1941.0	1884.5	1876.7	3134.9	2954.7	3871.1	6473.0	6408.4	5717.4	10184.2	9454.4
uNVMe_B	1556.7	2239.5	2123.6	2702.8	3998.7	4129.1	4794.6	7846.5	7978.2	6523.8	9098.0	10242.2



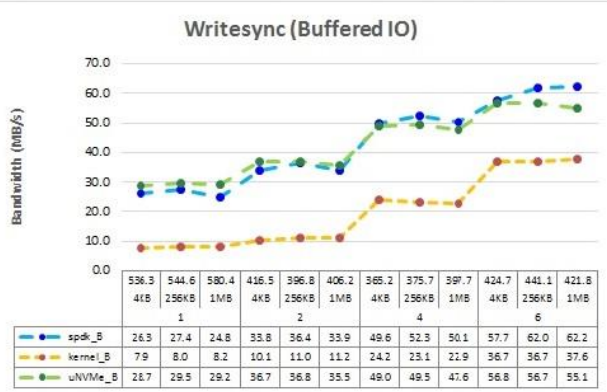
	4KB	256KB	1MB	4KB	256KB	1MB	4KB	256KB	1MB	4KB	256KB	1MB
	1			2			4			6		
spdk_B	536.3	544.6	580.4	416.5	396.8	406.2	365.2	375.7	397.7	424.7	441.1	421.8
kernel_B	359.4	366.1	375.5	349.4	360.4	353.7	350.5	369.0	350.5	413.2	462.8	459.4
uNVMe_B	694.1	977.9	794.4	380.9	458.8	444.5	371.8	382.5	343.7	430.8	447.0	425.5



	4KB	256KB	1MB	4KB	256KB	1MB	4KB	256KB	1MB	4KB	256KB	1MB
	1			2			4			6		
spdk_B	300.5	354.6	320.2	285.8	285.1	283.0	290.6	294.4	291.2	314.1	338.6	327.8
kernel_B	270.7	291.0	280.1	251.0	263.2	256.9	258.3	277.0	269.2	258.7	309.9	296.5
uNVMe_B	398.4	396.3	400.0	305.4	285.7	298.8	306.9	314.6	294.5	354.5	341.3	333.4



	4KB	256KB	1MB	4KB	256KB	1MB	4KB	256KB	1MB	4KB	256KB	1MB
	1			2			4			6		
spdk_B	13.2	2.4	1.1	25.5	3.5	1.5	41.6	4.7	1.9	59.9	5.5	2.2
kernel_B	79.2	13.9	4.4	154.3	28.8	8.8	276.9	53.9	15.4	384.8	73.3	21.1
uNVMe_B	67.3	18.9	5.5	96.8	33.5	7.1	282.6	37.1	8.2	389.2	66.1	11.0



	4KB	256KB	1MB	4KB	256KB	1MB	4KB	256KB	1MB	4KB	256KB	1MB
	1			2			4			6		
spdk_B	26.3	27.4	24.8	33.8	36.4	33.9	49.6	52.3	50.1	57.7	62.0	62.2
kernel_B	7.9	8.0	8.2	10.1	11.0	11.2	24.2	23.1	22.9	36.7	36.7	37.6
uNVMe_B	28.7	29.5	29.2	36.7	36.8	35.5	49.0	49.5	47.6	56.8	56.7	55.1

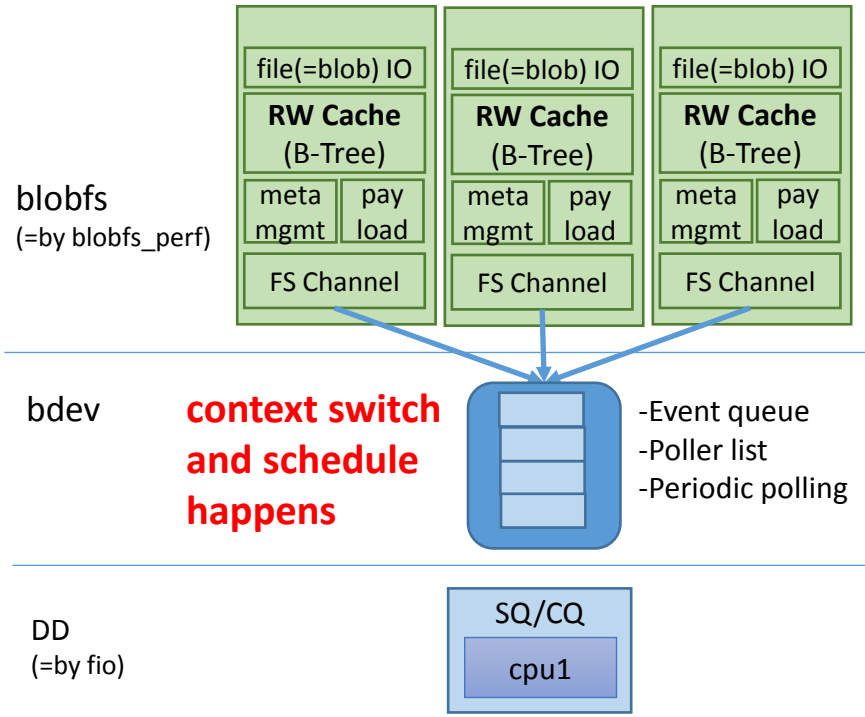
System : xeon E5-2667 16cores 32threads @3.60GHz , 256GB DRAM, Debian 9.1(kernel 4.9), PM983 4TB
RocksDB: v5.6.1, 16B key, 1000B value, 1M keys for each thread, disable block cache, repeat 5 time on readseq, fillseq from N threads
SPDK: v18.04.1, blobfs cache 4GB **uNVME**: blobfs cache 4GB , RETAIN_CACHE=1, PREFETCH_CTL=1, 4KB prefetch_threshold for 4KB block_size, 128KB prefetch_threshold for 256KB/1MB block_size
- Please refer Evaluation Guide for Test configuration (thread = 1/2/4/6 and so on)

Contribution over SPDK(3) – IO Path (TBD)

- **SPDK Limitation**

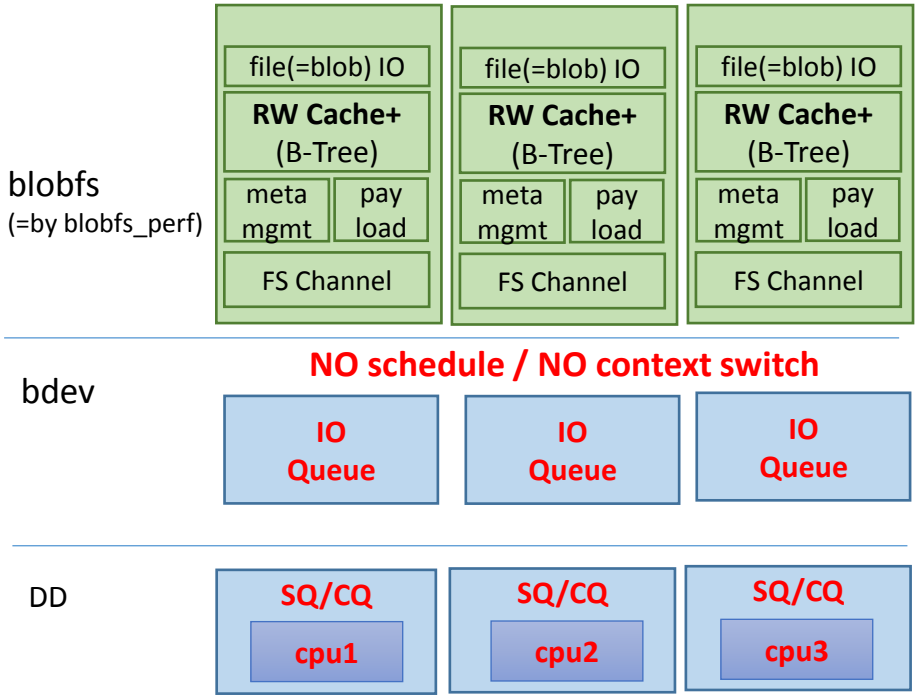
✓ Context-switch happens at blobfs to block device layer on every single blob IO request.

✓ Then, Block IO thread performs sort of scheduling : event queue, polling, interval polling . And big-loop of the IO thread consumes a certain amount of CPU cycle.



- **uNVMe path**

✓ Plan to provide IO path from top application top to bottom driver without IO schedule and context-switch



QA Process

Reuse of uNVMe-Rocksdb Phase1

- db_bench
- Workload: fillseq / randread / seqread / overwrite / writesync / readwrite

※ *For testing direct / buffered IO, please refer test configuration of Evaluation Guide*

Newly introduce blobfs layer BM Tool: Blobfs_perf

- Performs file IO on blobfs perf
- Seqwrite / seqread / randread/
- Buffered / Direct-IO
- Control read-cache reclaim
- Control prefetch size and prefetch threshold
- Multi-threads
- IO size and block size
- Working on vanilla SPDK / uNVMe

THE NEXT CREATION STARTS HERE

Placing **memory** at the forefront of future innovation and creative IT life