Next generation storage

THE LINUX FOUNDATION

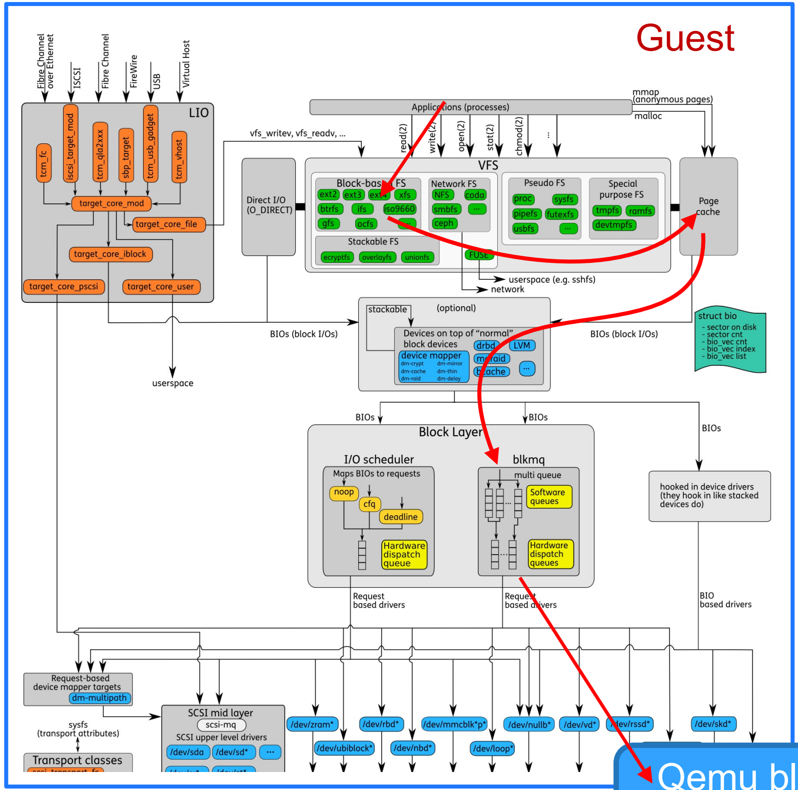
LF EDGE

Storage features required by a cloud provider

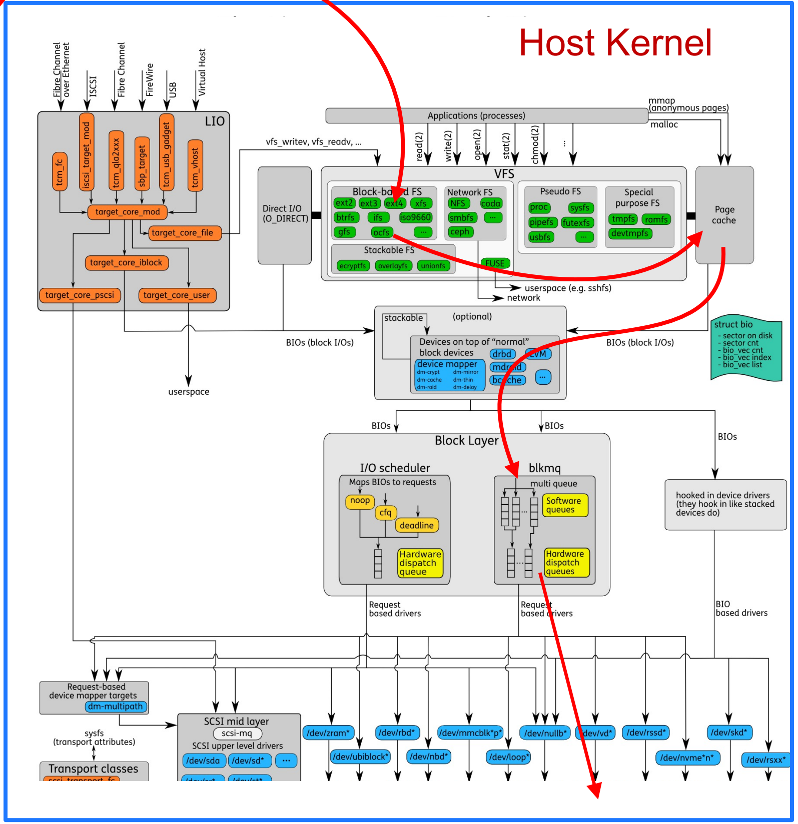
- Full disk encryption
- Thin provisioning
- Snapshoting
- Compression

- All the above is (or can be) covered by qcow2 in current implementation

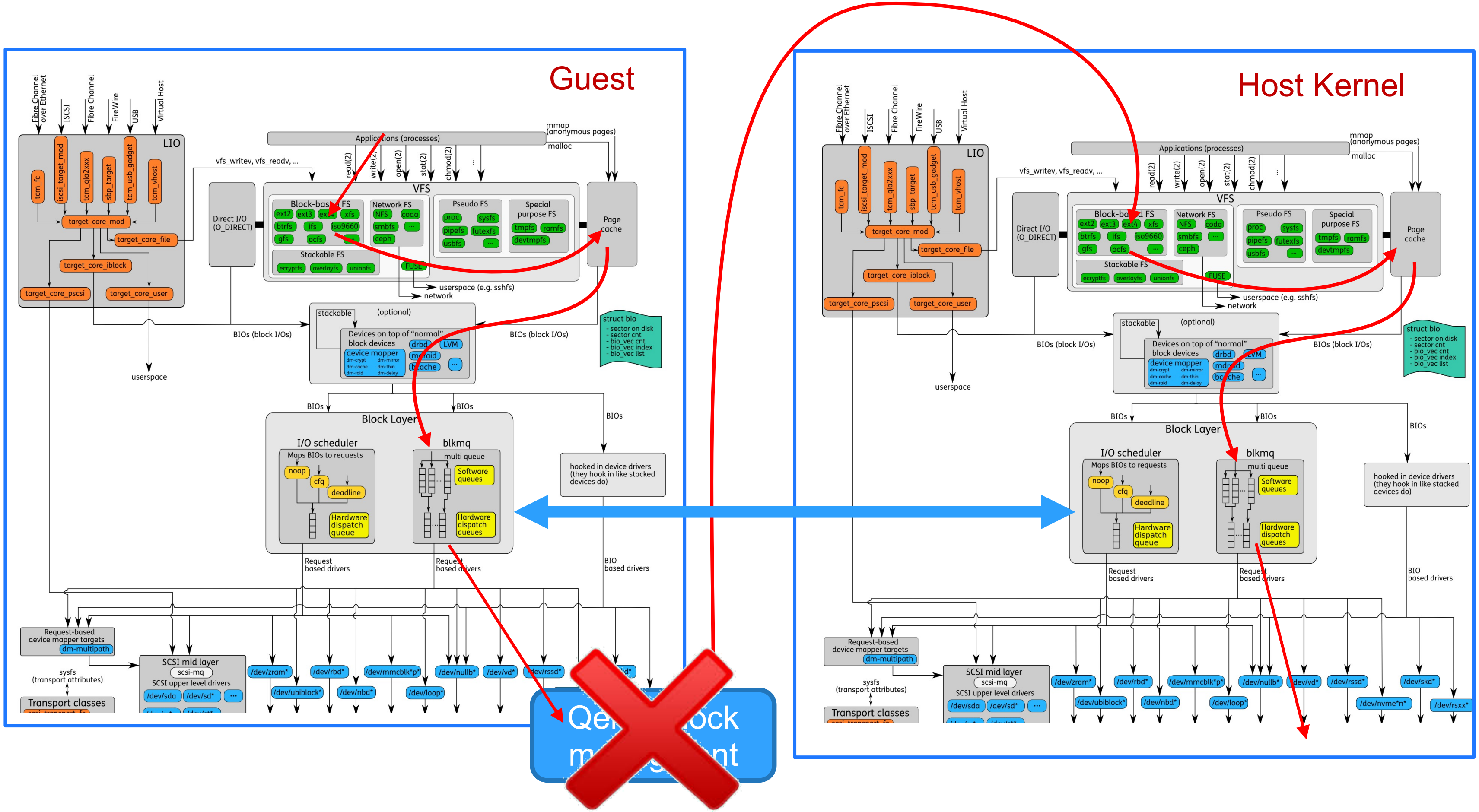
Current Storage overview



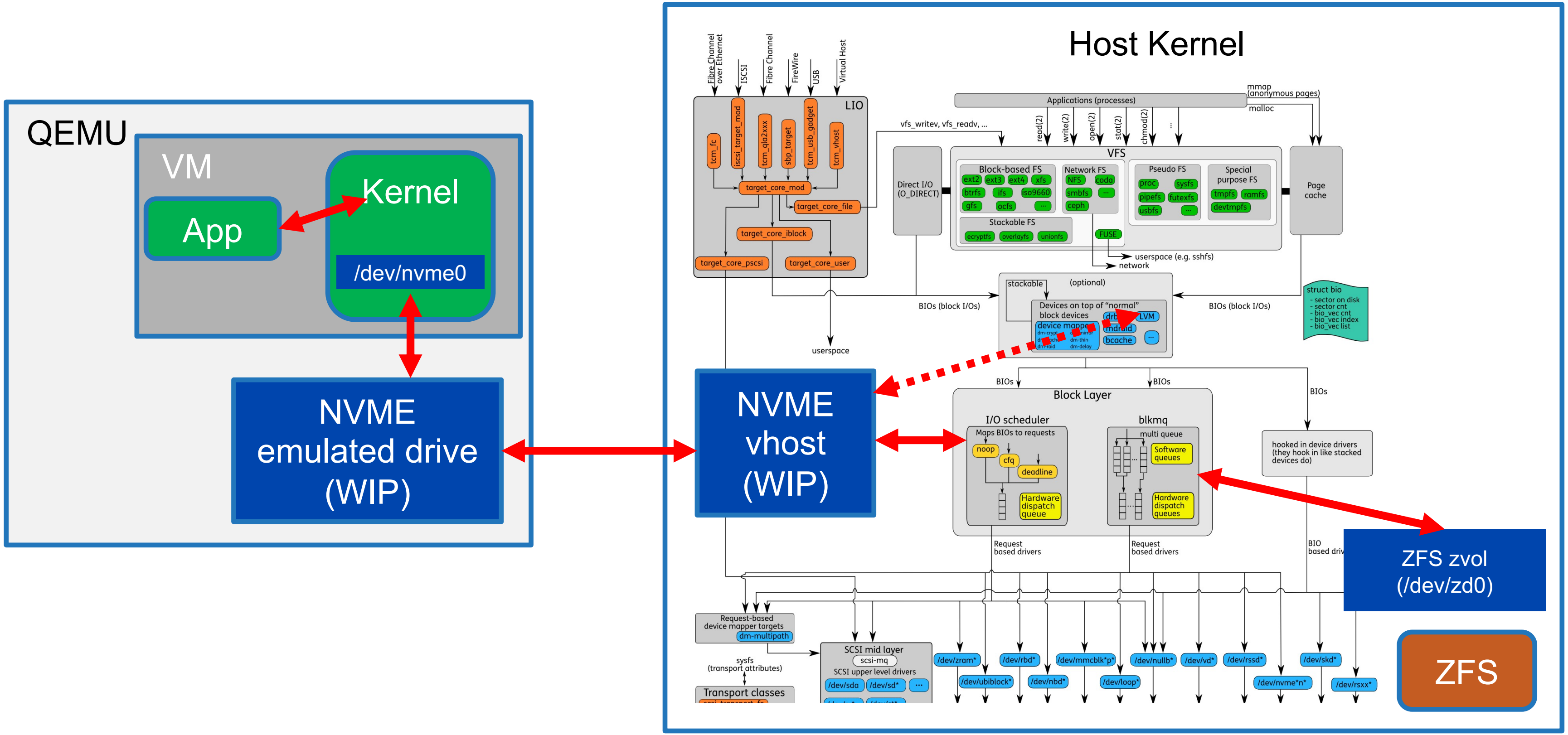
Qemu block managment



New gen storage overview



Next gen storage overview



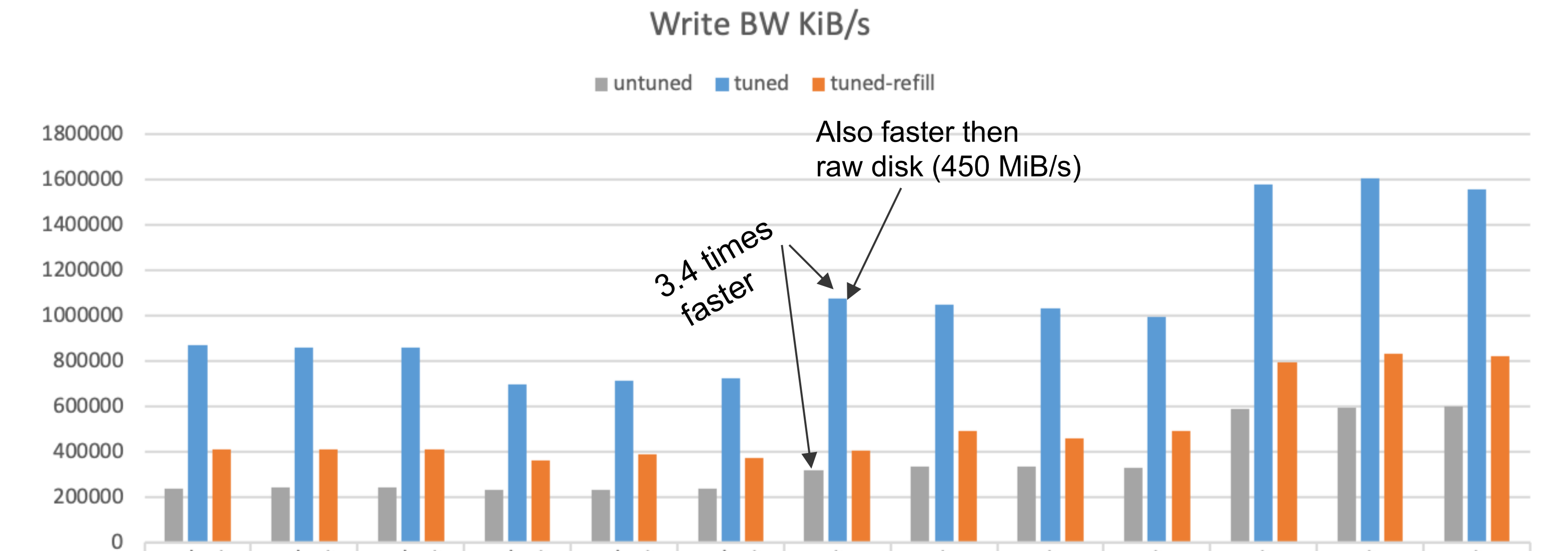
ZFS current status

- Initial support in Eve:
  - Software raid
  - Image deploymend to edge nodes
  - Zvols attached to VMs via scsi/vhost
- Lots and lots of benchmarking was done
- Autobench utility for unattended benchmarking
- Scripted (but still pretty involving) benchmarking from the Eve debug container

ZFS current status

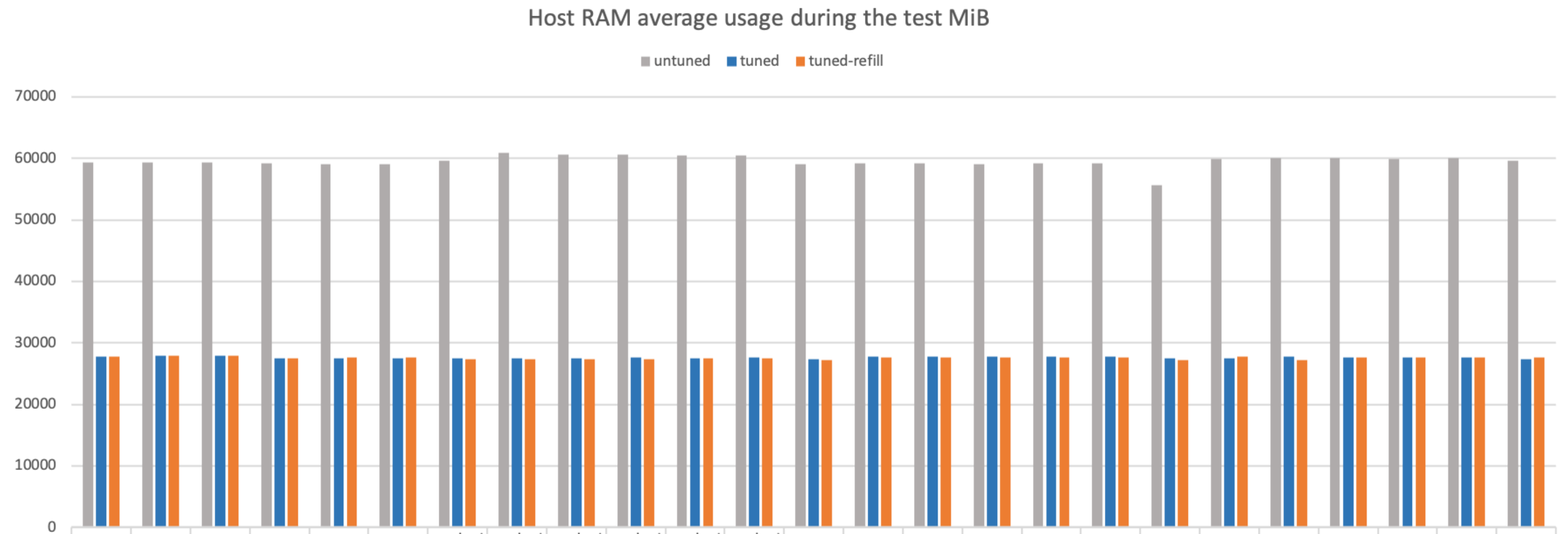
- Performs very well on liniar workloads enen on tiny machines
- Highly parallel workloads is a problem on smaller machines (e.g Atom with 8GiB RAM)
  - Latency on highly parallel workloads (4 jobs each submitting 16 requests at once) reaches tens of _seconds_

ZFS efforts



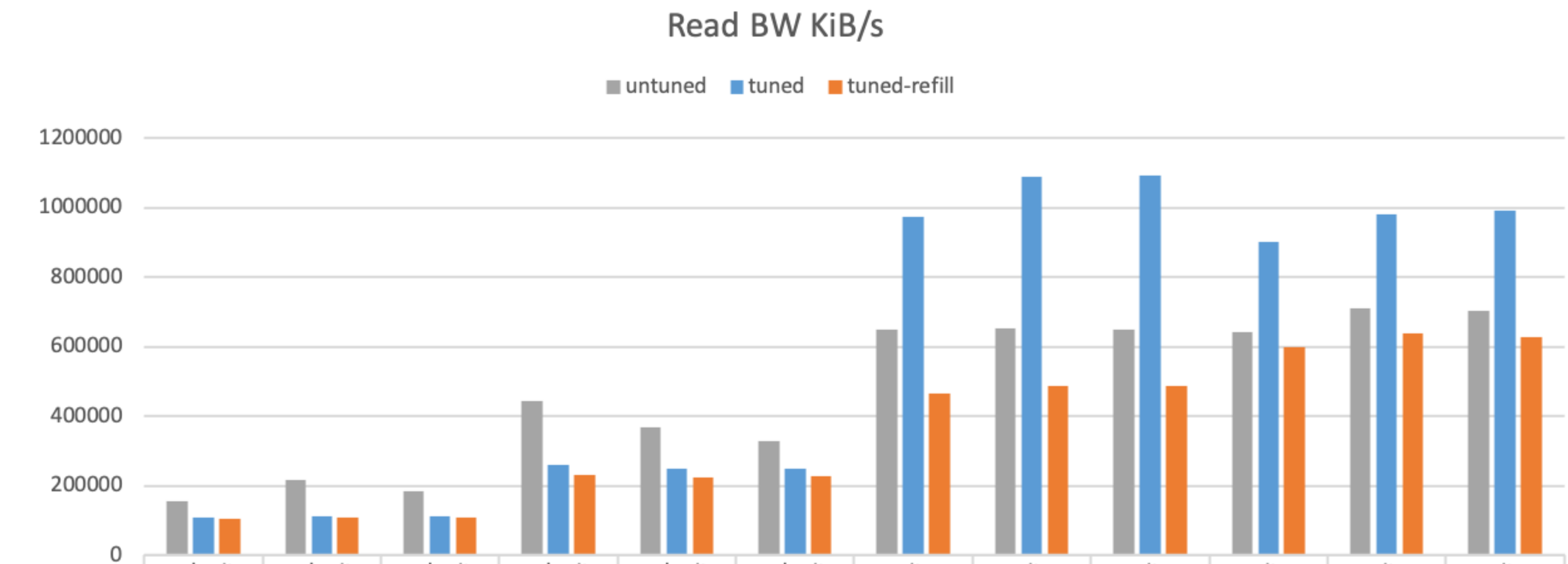
	randwrite-jobs1-bs64k-iodepth1	randwrite-jobs1-bs64k-iodepth2	randwrite-jobs1-bs64k-iodepth4	randwrite-jobs2-bs64k-iodepth1	randwrite-jobs2-bs64k-iodepth2	randwrite-jobs2-bs64k-iodepth4	write-jobs1-bs256k-iodepth1	write-jobs1-bs64k-iodepth1	write-jobs1-bs64k-iodepth2	write-jobs1-bs64k-iodepth4	write-jobs2-bs64k-iodepth1	write-jobs2-bs64k-iodepth2	write-jobs2-bs64k-iodepth4
untuned	241166	242102	241626	235022	235654	237805	318392	335656	335875	332171	590414	594419	600963
tuned	871250	857494	860450	699725	715495	723433	1078629	1049081	1032788	995729	1578269	1604306	1555271
tuned-refill	409823	413518	412138	362040	387360	371677	406107	491313	460501	492057	794615	834103	821317

ZFS efforts



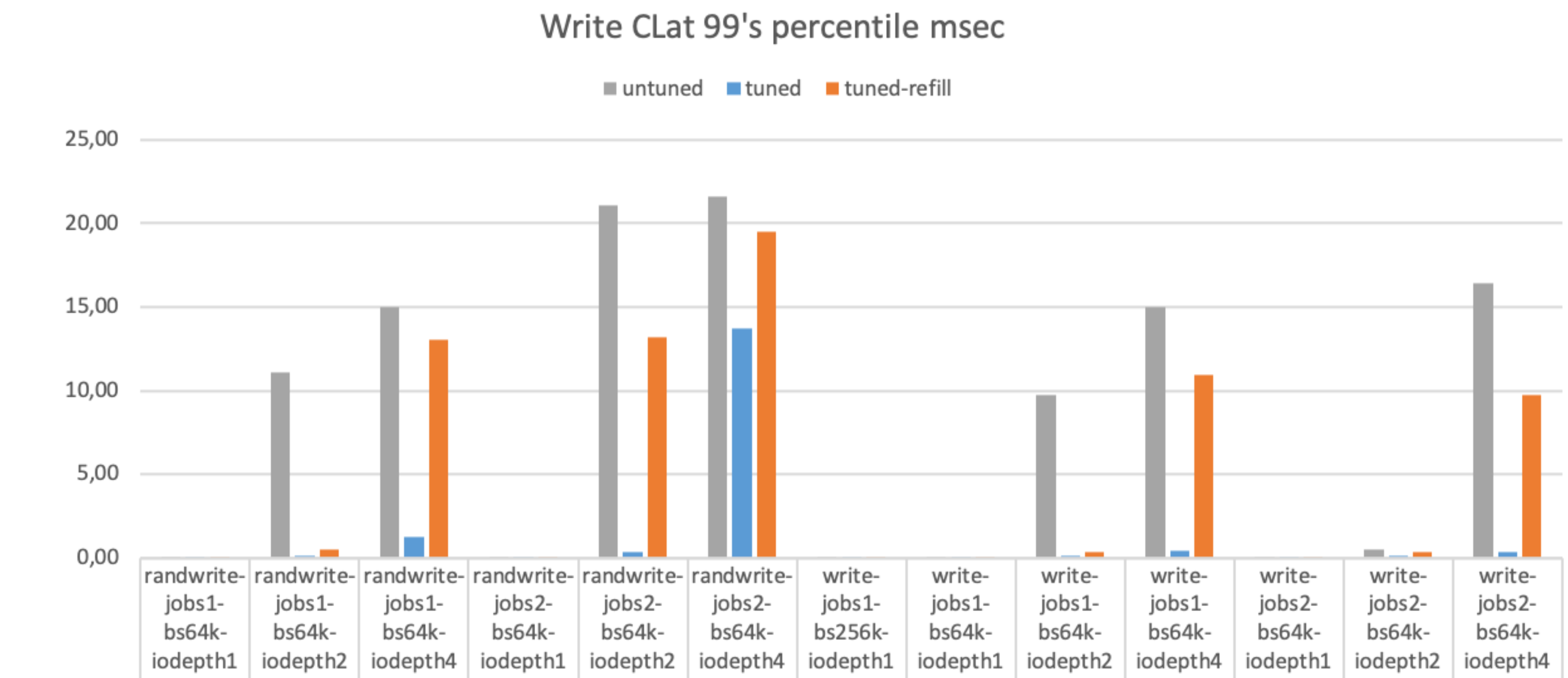
	randread-jobs1-bs64k-iodepth 1	randread-jobs1-bs64k-iodepth 2	randread-jobs1-bs64k-iodepth 4	randread-jobs2-bs64k-iodepth 1	randread-jobs2-bs64k-iodepth 2	randread-jobs2-bs64k-iodepth 4	randwrite-jobs1-bs64k-iodepth 1	randwrite-jobs1-bs64k-iodepth 2	randwrite-jobs1-bs64k-iodepth 4	randwrite-jobs2-bs64k-iodepth 1	randwrite-jobs2-bs64k-iodepth 2	randwrite-jobs2-bs64k-iodepth 4	read-jobs1-bs64k-iodepth 1	read-jobs1-bs64k-iodepth 2	read-jobs1-bs64k-iodepth 4	read-jobs2-bs64k-iodepth 1	read-jobs2-bs64k-iodepth 2	read-jobs2-bs64k-iodepth 4	write-jobs1-bs256k-iodepth 1	write-jobs1-bs64k-iodepth 1	write-jobs1-bs64k-iodepth 2	write-jobs1-bs64k-iodepth 4	write-jobs2-bs64k-iodepth 1	write-jobs2-bs64k-iodepth 2	write-jobs2-bs64k-iodepth 4
untuned	59302	59333	59327	59156	59025	58993	59672	60905	60672	60577	60489	60422	58991	59113	59116	59111	59184	59174	55575	59922	59979	60009	59959	59988	59635
tuned	27771	27841	27849	27489	27476	27503	27441	27468	27488	27564	27498	27573	27398	27697	27743	27748	27775	27778	27445	27548	27723	27694	27690	27581	27380
tuned-refill	27822	27879	27884	27547	27563	27565	27367	27384	27391	27405	27468	27411	27155	27590	27630	27645	27673	27681	27233	27770	27194	27673	27631	27632	27644

ZFS efforts



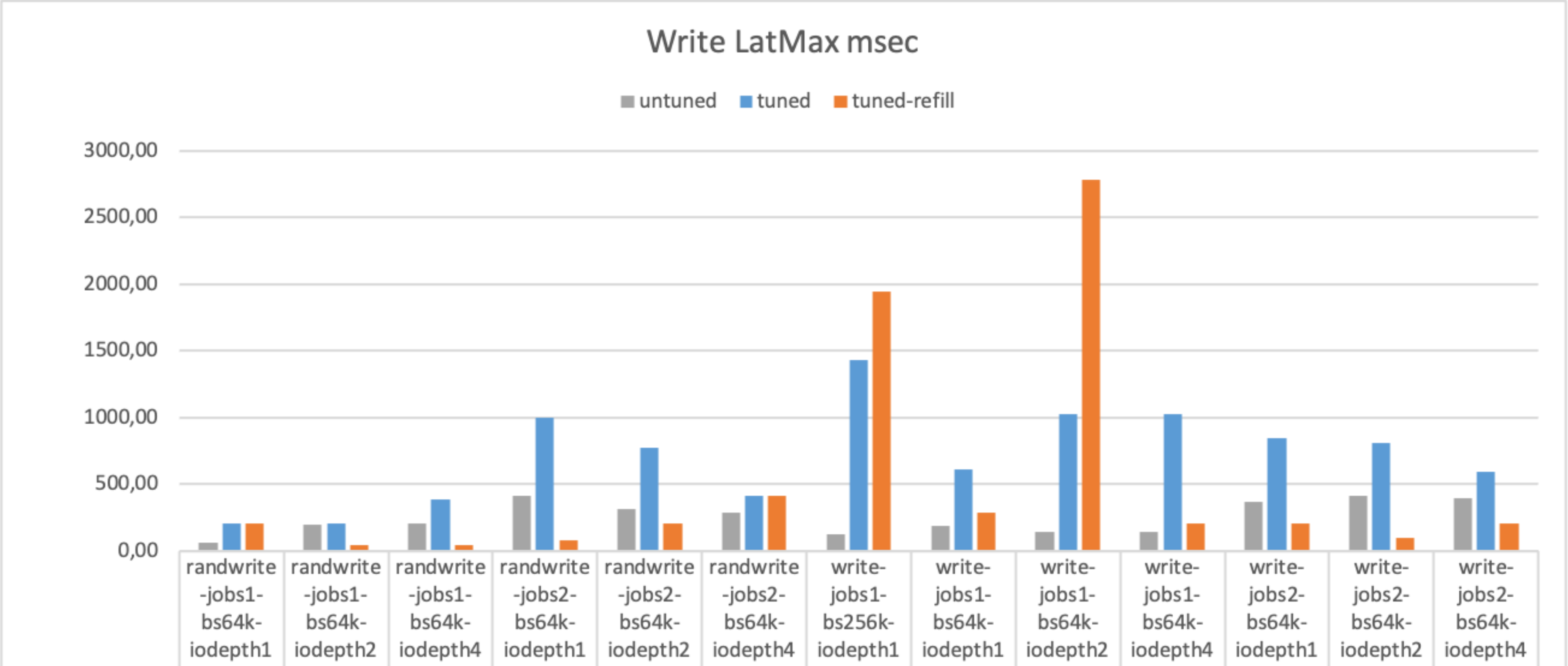
	randwrite-jobs1-bs64k-iodepth1	randwrite-jobs1-bs64k-iodepth2	randwrite-jobs1-bs64k-iodepth4	randwrite-jobs2-bs64k-iodepth1	randwrite-jobs2-bs64k-iodepth2	randwrite-jobs2-bs64k-iodepth4	write-jobs1-bs256k-iodepth1	write-jobs1-bs64k-iodepth1	write-jobs1-bs64k-iodepth2	write-jobs1-bs64k-iodepth4	write-jobs2-bs64k-iodepth1	write-jobs2-bs64k-iodepth2
untuned	153857	215253	185233	442427	368621	328437	650713	651923	650286	641367	711300	704691
tuned	109760	112799	113691	259150	250328	250201	975023	1089348	1093147	900229	981944	993222
tuned-refill	105724	108203	107993	232766	225064	225923	466787	485240	485515	597322	640059	628428

ZFS efforts



	randwrite-jobs1-bs64k-iodepth1	randwrite-jobs1-bs64k-iodepth2	randwrite-jobs1-bs64k-iodepth4	randwrite-jobs2-bs64k-iodepth1	randwrite-jobs2-bs64k-iodepth2	randwrite-jobs2-bs64k-iodepth4	write-jobs1-bs256k-iodepth1	write-jobs1-bs64k-iodepth1	write-jobs1-bs64k-iodepth2	write-jobs1-bs64k-iodepth4	write-jobs2-bs64k-iodepth1	write-jobs2-bs64k-iodepth2	write-jobs2-bs64k-iodepth4
untuned	0,00	11,08	15,01	0,00	21,10	21,63	0,00	0,00	9,76	15,01	0,00	0,51	16,45
tuned	0,00	0,17	1,24	0,00	0,33	13,70	0,00	0,00	0,12	0,47	0,00	0,13	0,40
tuned-refill	0,00	0,50	13,04	0,00	13,17	19,53	0,00	0,00	0,36	10,94	0,00	0,34	9,76

ZFS efforts



	randwrite-jobs1-bs64k-iodepth1	randwrite-jobs1-bs64k-iodepth2	randwrite-jobs1-bs64k-iodepth4	randwrite-jobs2-bs64k-iodepth1	randwrite-jobs2-bs64k-iodepth2	randwrite-jobs2-bs64k-iodepth4	write-jobs1-bs256k-iodepth1	write-jobs1-bs64k-iodepth1	write-jobs1-bs64k-iodepth2	write-jobs1-bs64k-iodepth4	write-jobs2-bs64k-iodepth1	write-jobs2-bs64k-iodepth2	write-jobs2-bs64k-iodepth4
untuned	60,83	191,35	204,96	407,83	308,86	285,86	123,61	187,11	142,72	145,53	362,99	412,40	393,34
tuned	204,00	203,91	387,95	995,75	774,16	413,68	1428,00	611,99	1020,01	1020,07	841,49	812,18	590,33
tuned-refill	202,09	43,06	41,55	80,82	203,52	407,45	1942,39	283,43	2782,79	205,88	203,74	97,99	205,17

ZFS next steps: storage health reports

- Revise current storage health reporting in Eve OS
- Add
  - Reporting multiple disks
  - S.M.A.R.T reporting
  - Zpool status errors
- Collaborate with cloud team to establish protocol

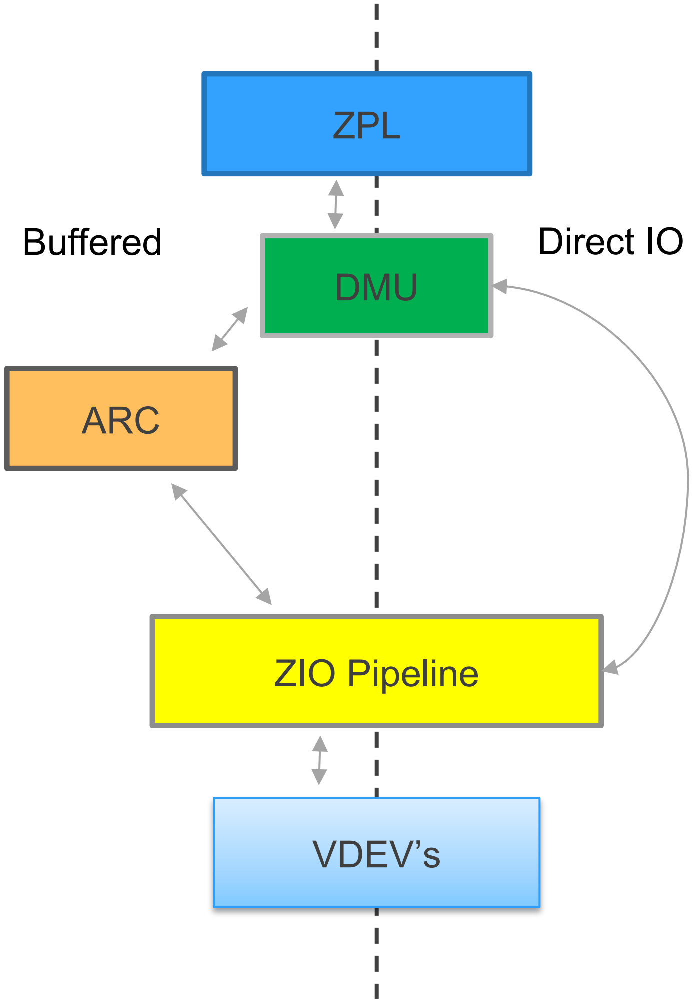
ZFS WIP: 20% disk space reservation

From [Oracle: Recommended Storage Pool Practices]

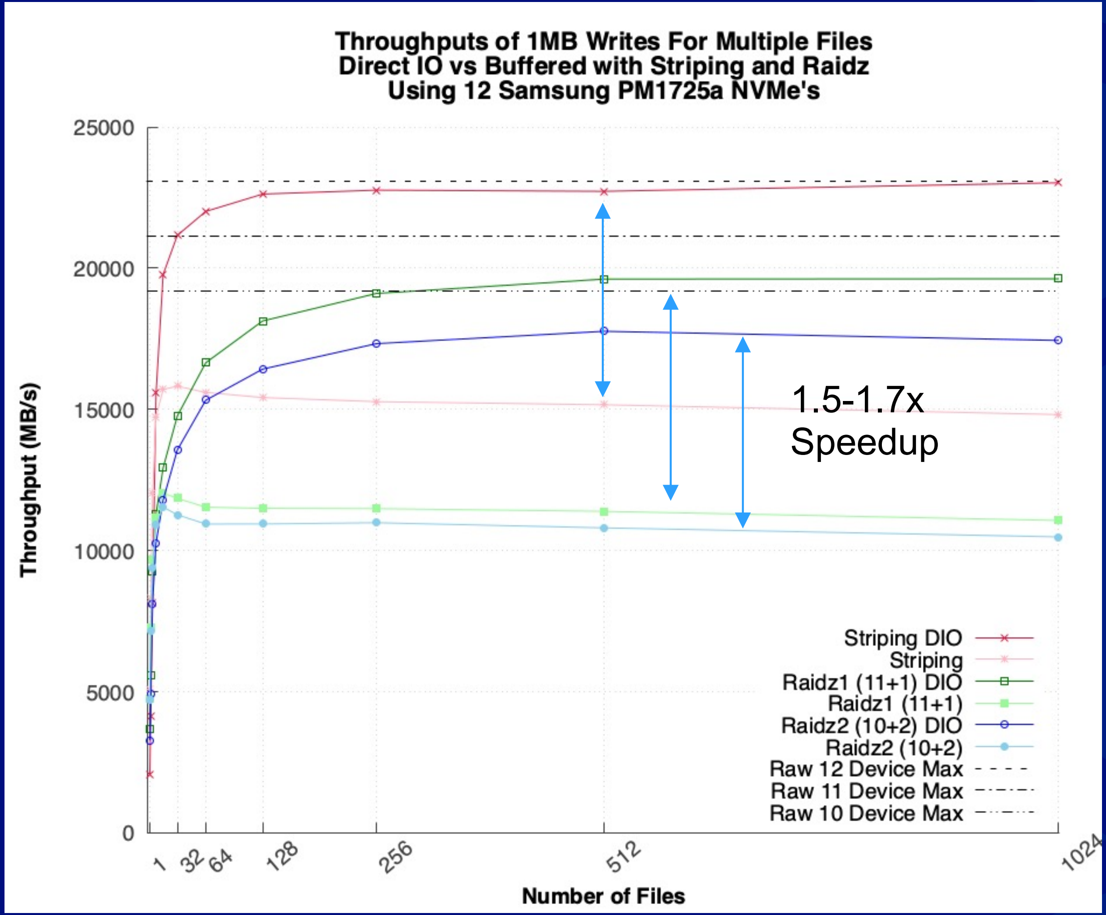
"Pool performance can degrade when a pool is very full and file systems are updated frequently, such as on a busy mail server. Full pools might cause a performance penalty, but no other issues. If the primary workload is immutable files, then keep pool in the 95-96% utilization range. Even with mostly static content in the 95-96% range, write, read, and resilvering performance might suffer."

Implementation: Direct IO Read in ZFS (Big Picture)

- Buffered
  - Cached? → Copy from ARC
  - Issue to ZFS pipeline
    - Copy to ARC
    - Copy to user buffer
- Direct IO
  - Bypass ARC
  - User pages are directly mapped into an ABD



Seq. Write Performance Results: ZFS NVMe Zpools



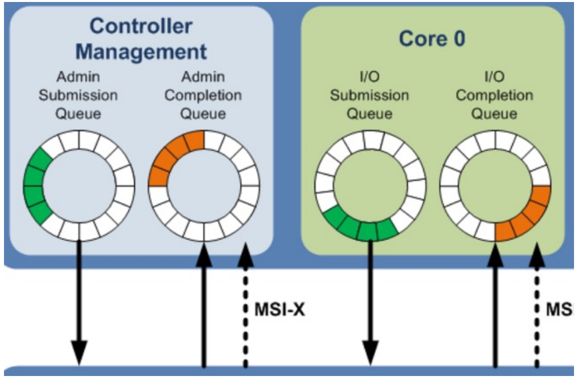
ZFS Future Ideas

- TaskQ and Thread Scaling
  - Provides a knob to adjust how many TaskQ/Threads will be running
- Thread/CPU Pinning
- Duty Cycle Limiting
  - Lowers thread priority if it takes to much of cpu time
- Async DMU / Async CoW
  - Deferring the reads so writes are not blocked
- ZFS Block Reference Table
  - Explicit files cloning (**`cp --reflink`**)

Shadow doorbell – paravirtualizable NVMe

- Updating **Tail/Head** registers are an MMIO operation
- Therefore each write generates vmexit
- NVMe 1.3 introduced ”shadow doorbell” concept
- If requested, **Tail/Head** registers are mirrored to a memory page
- Now Host OS can poll doorbells and process queus when it is convinient, avoiding expencive vmexits
- This effectively makes NVMe a pravirtualized protocol out of the box

NVMe/VHOST current status



- Intial hoocking into NVMe fabric machinery
- Functioning communication over hardcoded Admin queue
- Working Guest Phisical -> Host Virtual translations in the vhost driver
- Guest recognizes the NVMe device, successfully issues commands to create Submissions/Compleation queues, but operation fails (not implemented)

NVME/VHOST next steps – Prototype

- Implement creation of data queus
- Rework Admin queue creation – move away from hardcoded implementation
- Implement the minimum set of commands required to operate under linux
- Implement Shadow Queue
- Make sure works with Windows

NVME/VHOST next steps – towards first product

- Submit RFC patches to the mailing list once Prototype phase is ready
- Address comments, work on cleaning up hacks
- Run correctness tests, implement any missing bits an pieses

Some wild ideas: Vertical optimization

- ZFS worker threads per NVMe queue to improve cache locality
- ZFS objects exposed directly to virtual machine – paravirtualized file system
- Split available memory in 2 parts - base system and virtual machines. Allows to win back 400MiB on 25 GiB of ram dedicated to VM.
- Image online deployment
  - With zfs image deployment has to happen in 2 steps – download qcow2 and roll it out to zvol
  - There are multiple ways to do that online

Thank you!

THE LINUX FOUNDATION
LF EDGE

Backup

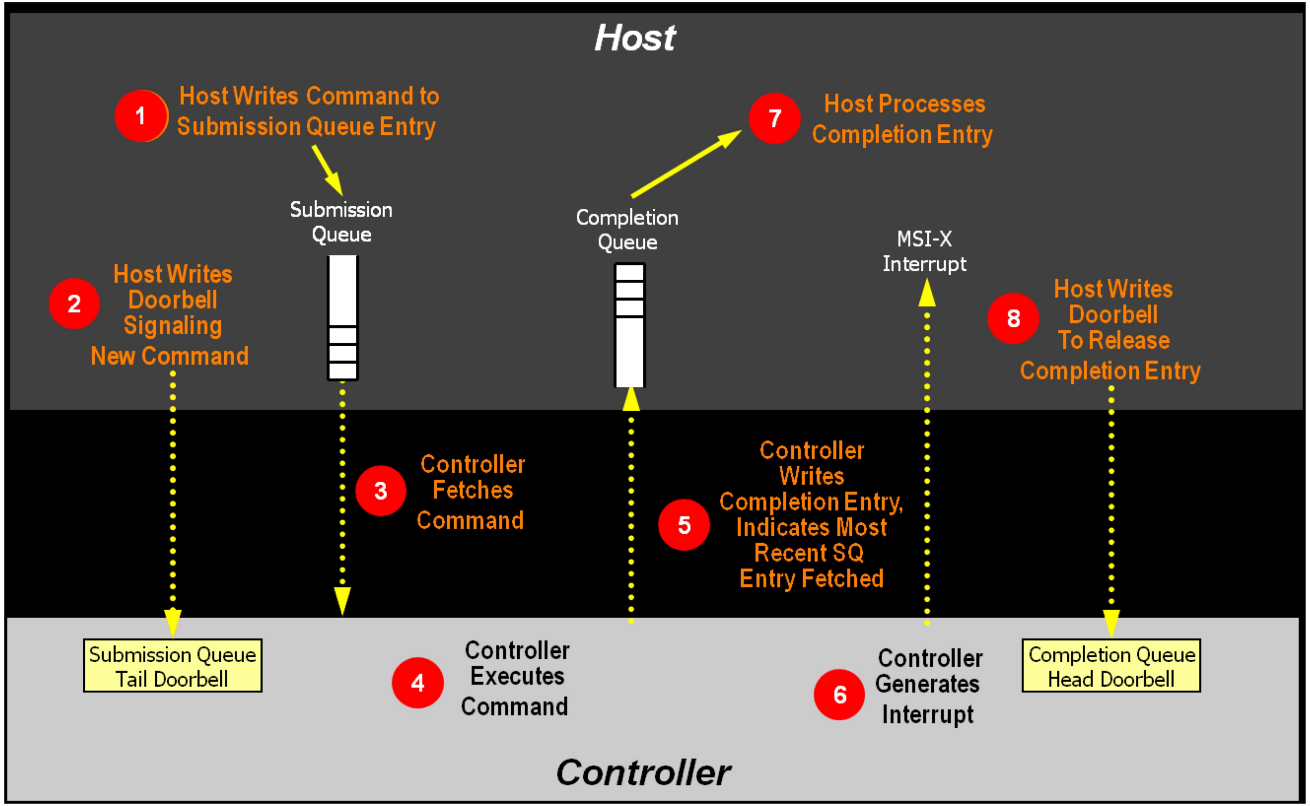
ZFS benefits

- Boot Environments (Failsafe OS upgrades)
- ZFS Encryption (Take data offline and put it at rest)
- Online Expansion (Add more space without interruption)
- Quotas and Reservations
- ZFS Project IDs
- Resilience and Redundancy (Bitrot detection, Disk failure)

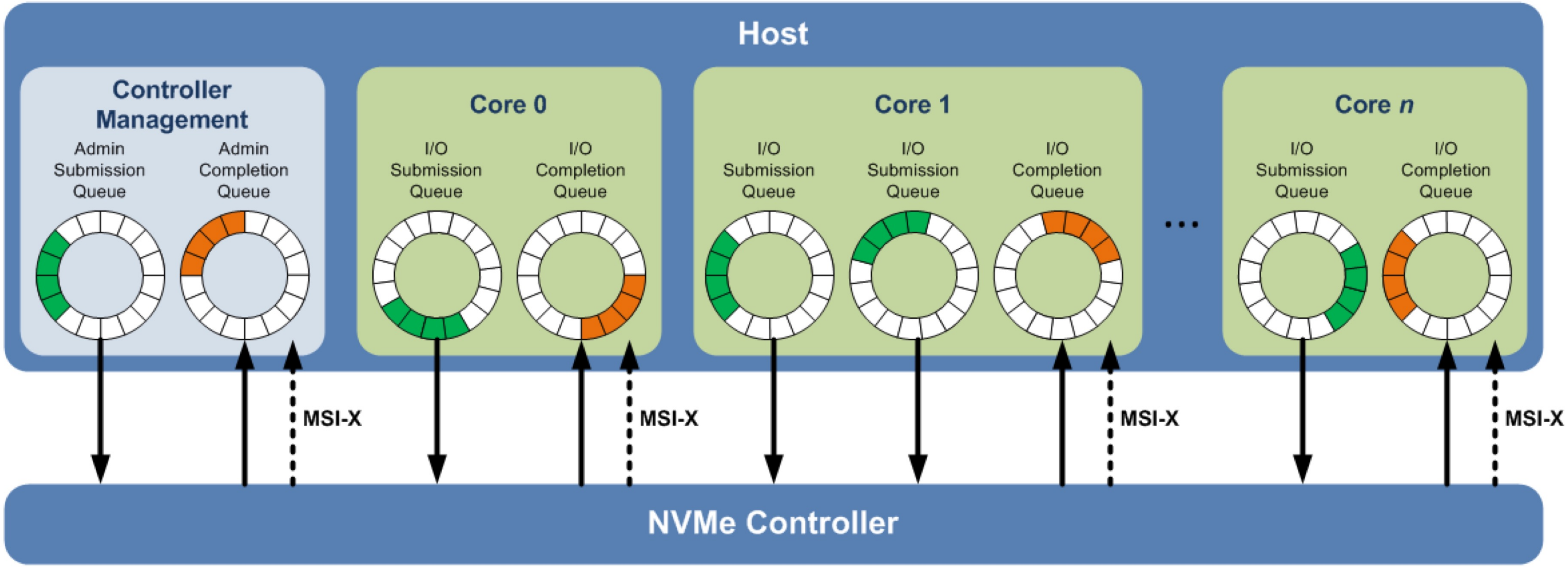
Why not LVM?

- LVM does support compression and thin provisioning, but the performance penalty is very high, which kills the major benefit of LVM-based solution
- Growing of the disk space takes a lot more steps in LVM (add disk, grow volume group, grow logical volume, grow file system sitting on the virtual media), which in generally can not be done online (or the process is quite finicky and dangerous)
- LVM lacks quota support. Once a Logical Volume was allocated to a container, you can't easily change the size of that volume. While in filesystem base approach you would need only change the quota of a dataset

NVMe background



NVMe background



Related work

- [RFC,v1] block/NVMe: introduce a new vhost NVMe host device to QEMU
- Linux NVME-vhost driver by Ming Lin <ming.l@ssi.samsung.com>