# Next generation storage

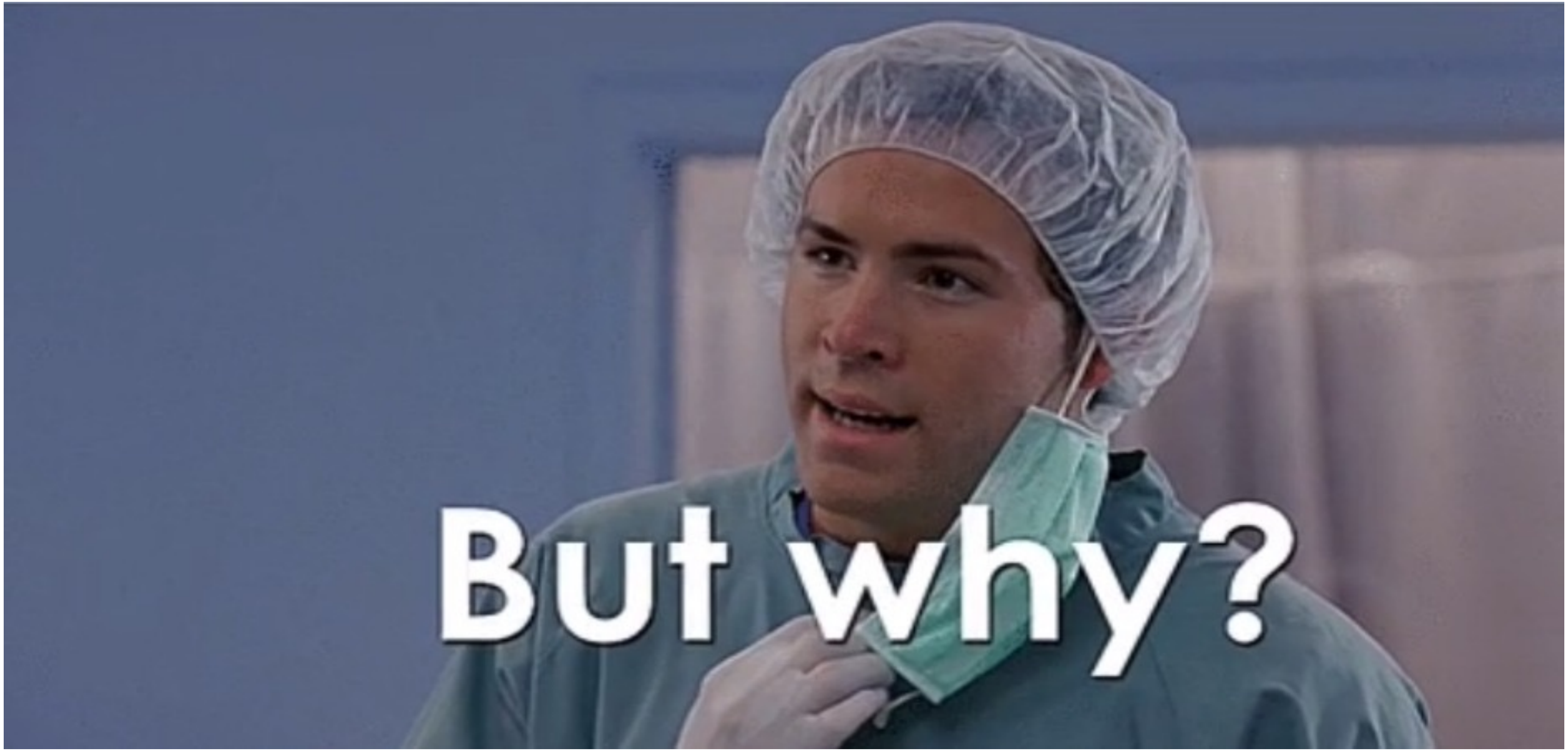
THE LINUX FOUNDATION

LF EDGE

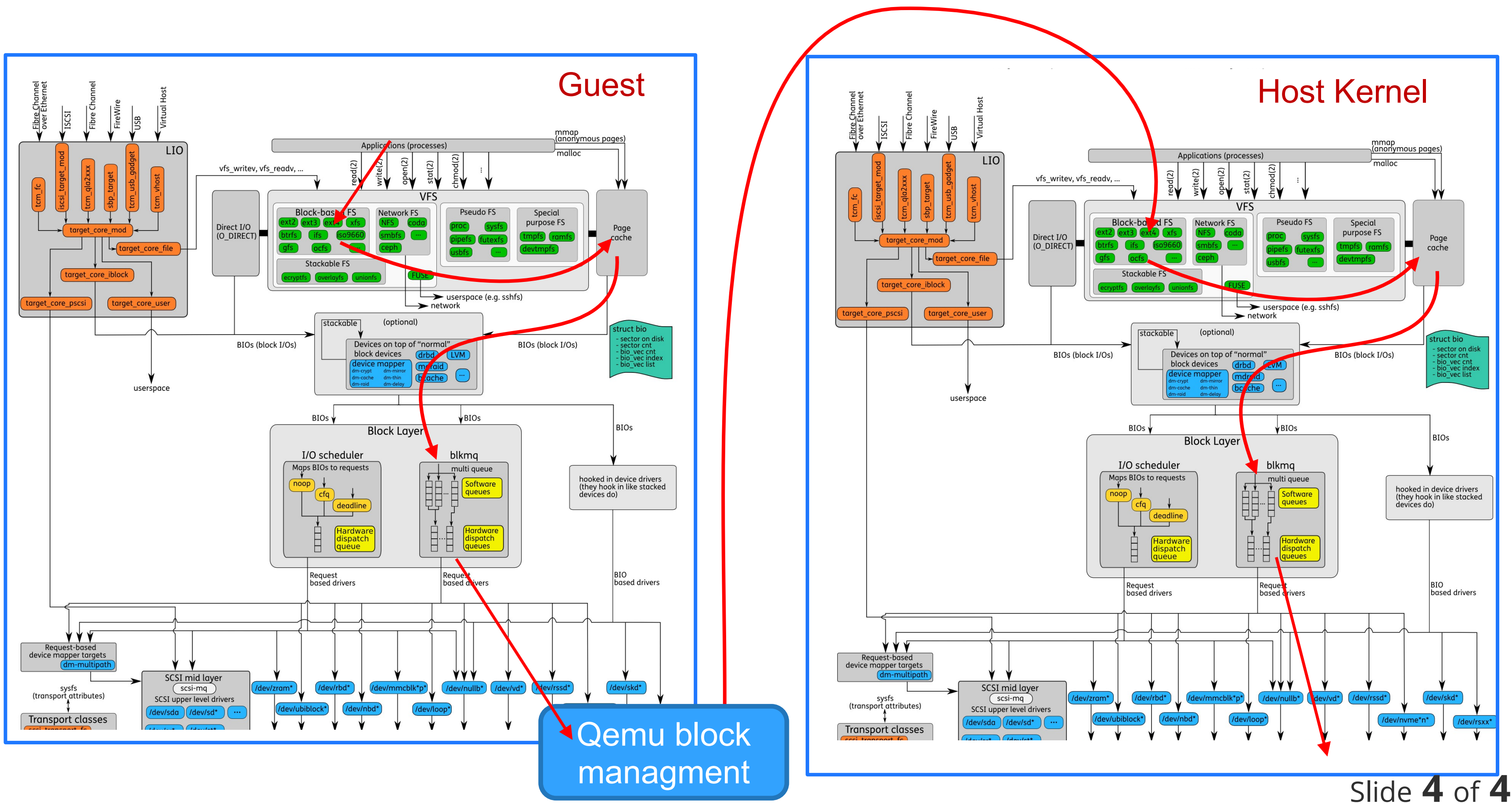
# Storage features required by a cloud provider

- Full disk encryption
- Thin provisioning
- Snapshoting
- Compression

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


But why?

# Current Storage overview

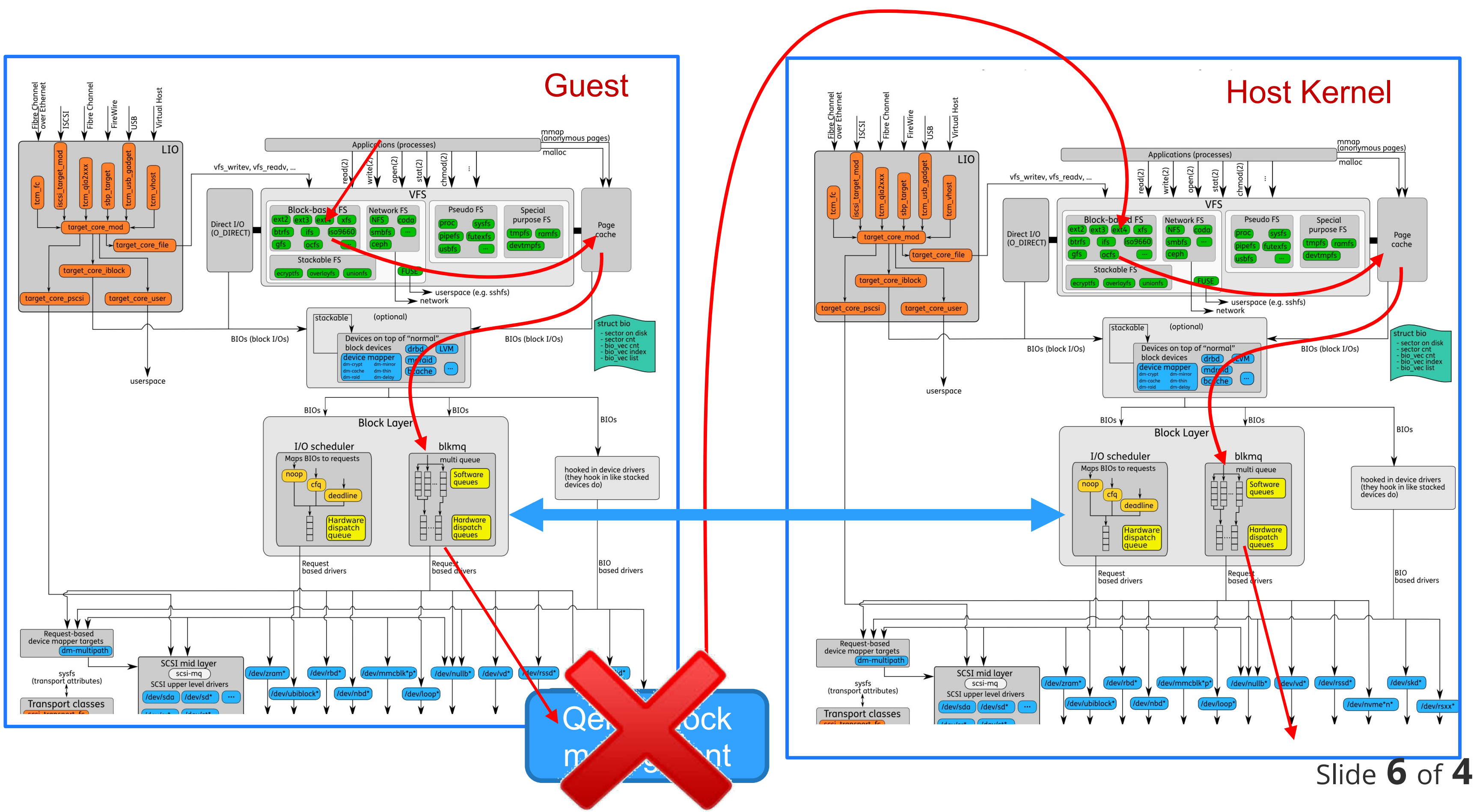


Qemu block managment

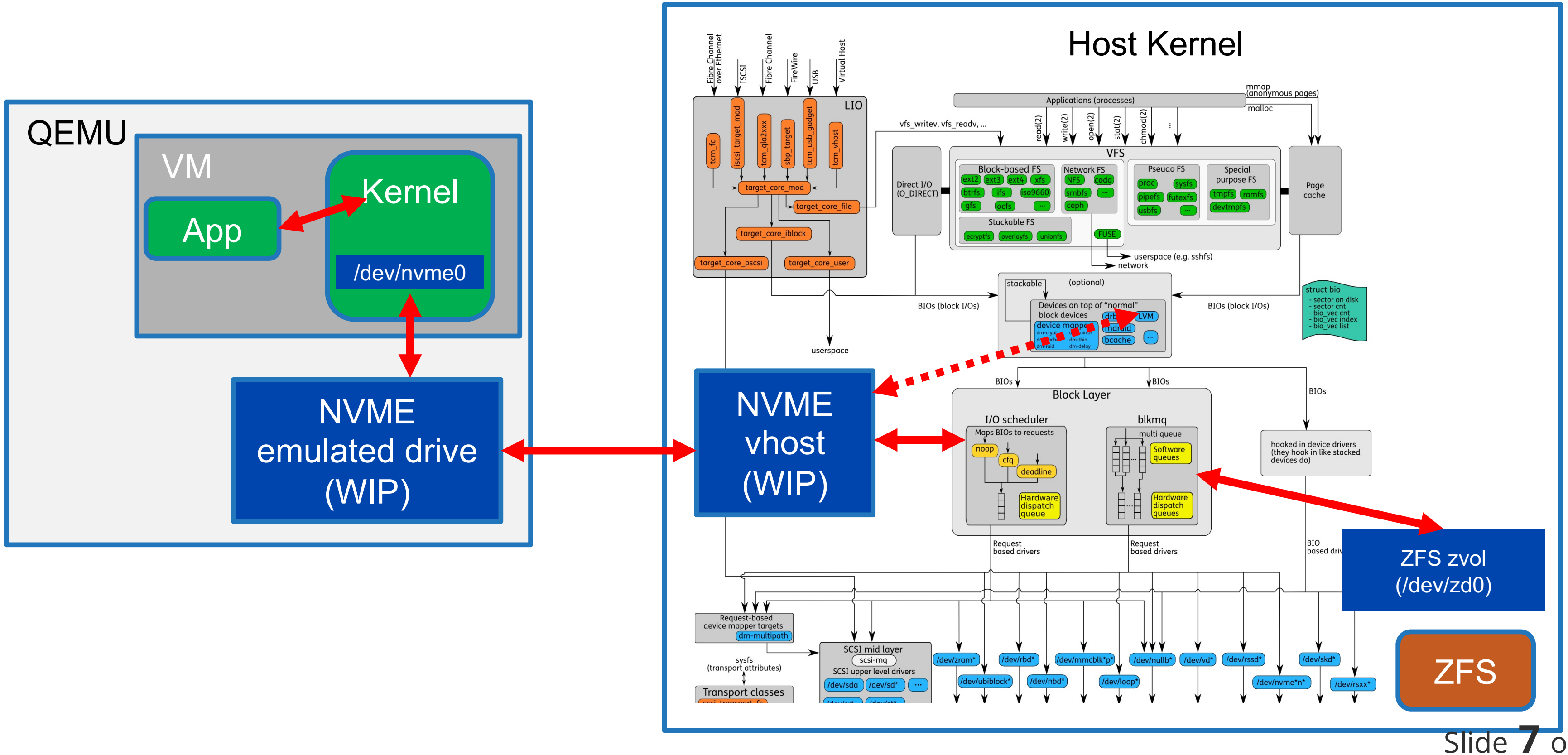
# Problems with current implementation

- Limited support of transparent compression and thin provisioning
- Lack of multiple physical drive support
- Lack of quotas support
- Performance degradation with multiple VMs running on the system
- CPU and RAM consumption with multiple VMs running on the system
- Has no awareness of other qemu processes and their block caching layers
- Proved to be notoriously difficult to optimize even for a single VM

# New gen storage overview



# Next gen storage overview



# 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

# 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)

# 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 is done
- Autobench utility for unattended benchmarking
- Scripted (but still pretty involving) benchmarking from the Eve debug container

# Priorities

- NVMe/VHOST technologie gives us significanly more room for vertical optimisation
- Efforts on ZFS and NVMe are not dependent on each other and can be executed in parallel
- However ZFS answers current existing custorme pain
- Thererfore  ZFS efforts have a higher priority
- However upgrade SCSI -> NVMe might cause problems if guest OS is not using UUID (disk name will change e.g. sda1 -> nvme0n1

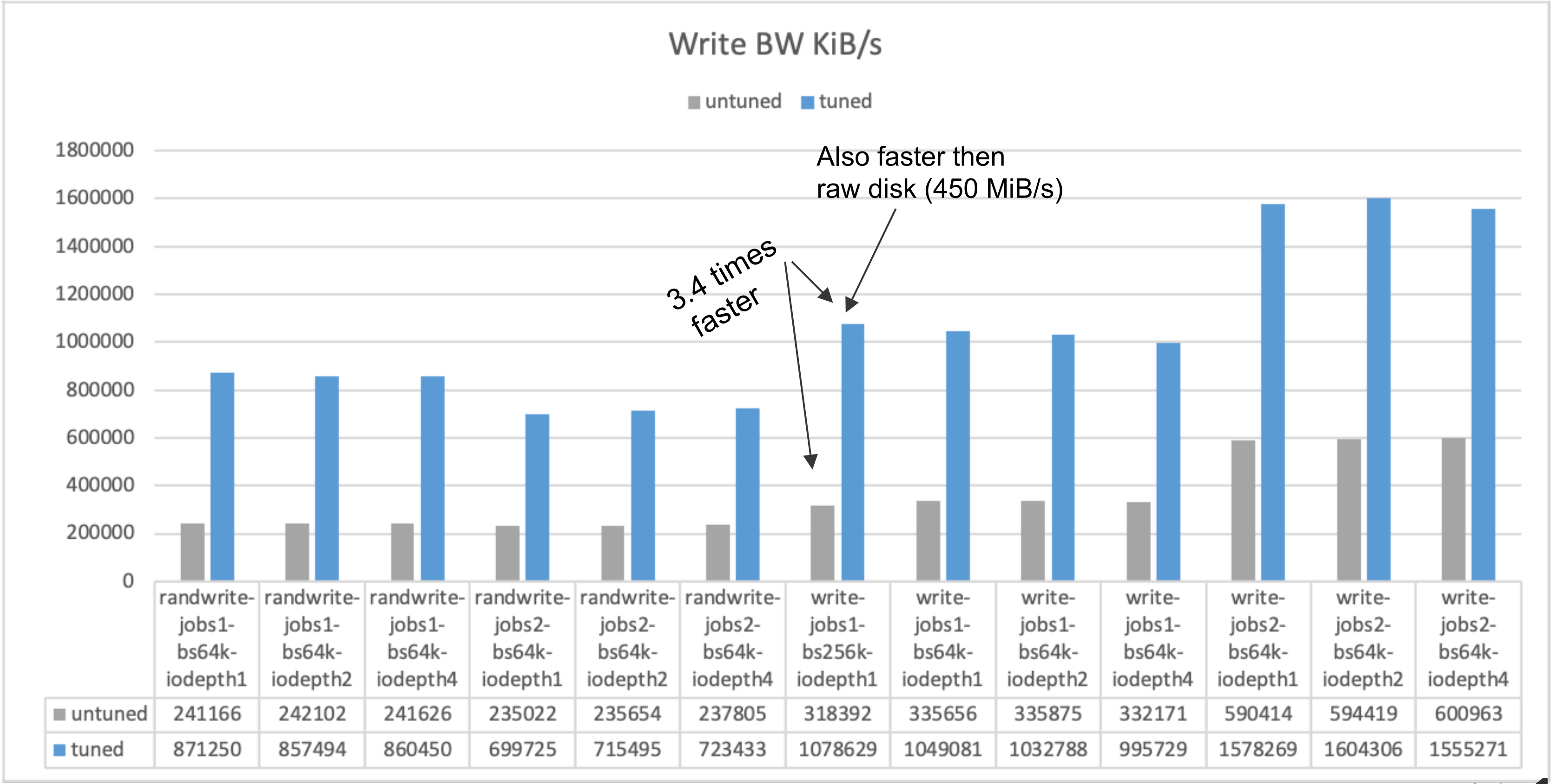
# Technologies overview

| Technology | Multiple Disks | Online expansion | Performance | Resiliency |
|---|---|---|---|---|
| scsi-vhost + zfs | 👍 | 👍 | **OK** unless multiple random access in parallel | 👍 |
| NVMe-vhost + zfs | 👍 | 👍 | **Better** (theoretically) | 👍 |
| lvm + vhost + md | 👍 | +/- | **Native** | 👍 |
| qcow2-virtio | ❌ | ❌ | Very good but at the price of resiliency | Good unless sudden power off is a frequent case |

# 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 Klara efforts

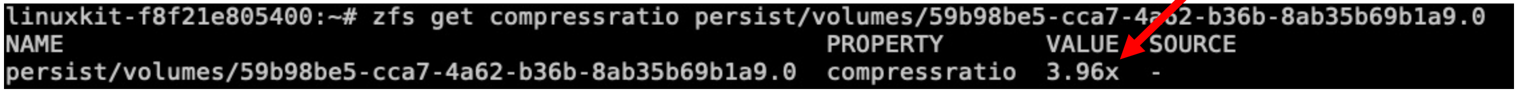


Also faster then raw disk (450 MiB/s)

3.4 times faster

| | 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 |

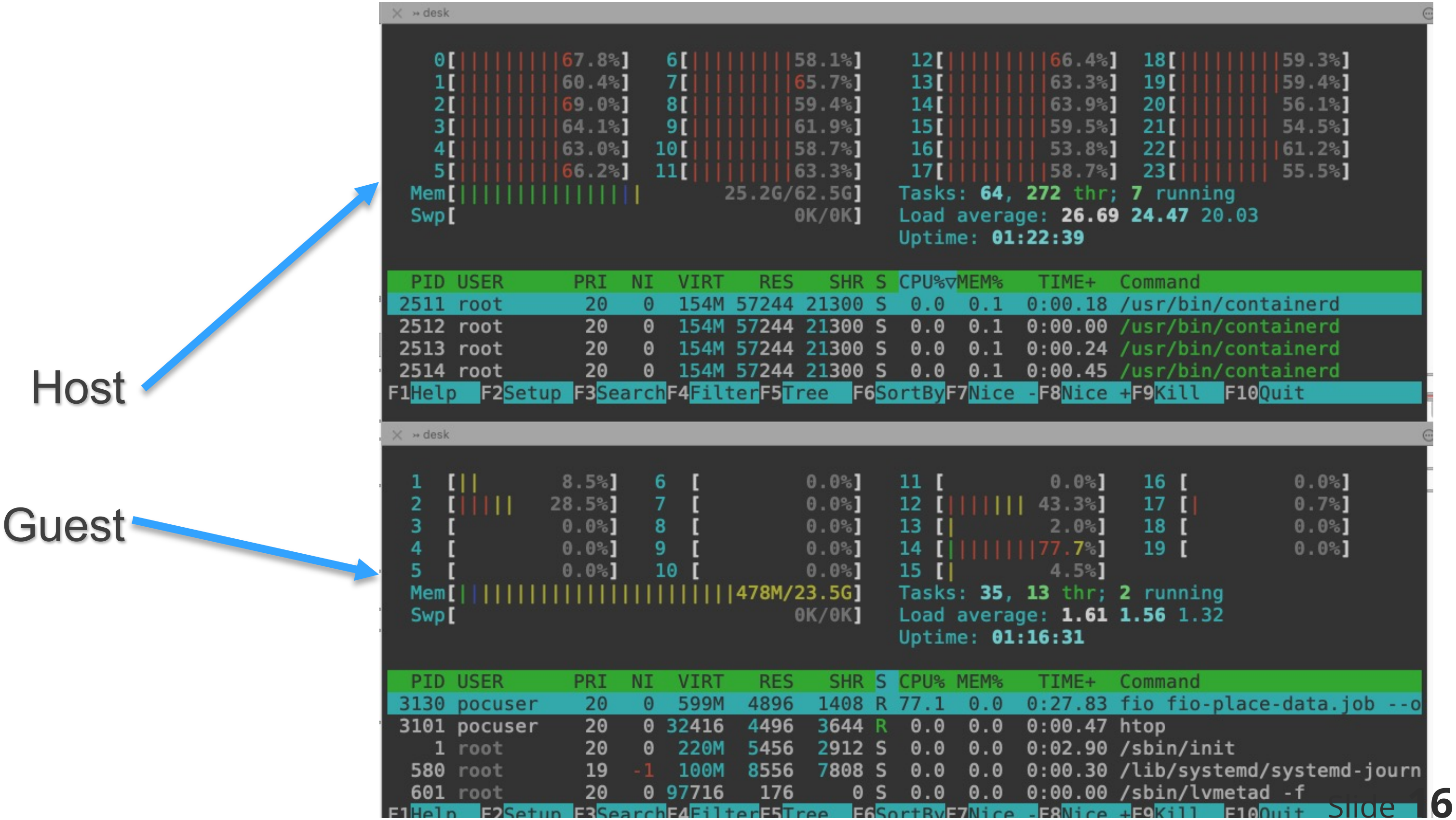
# ZFS compression ratio



```
linuxkit-f8f21e805400:~# zfs get compressratio persist/volumes/59b98be5-cca7-4a62-b36b-8ab35b69b1a9.0
NAME                                          PROPERTY       VALUE  SOURCE
persist/volumes/59b98be5-cca7-4a62-b36b-8ab35b69b1a9.0  compressratio  3.96x  -
```

Tuned

# Compression – CPU penalty

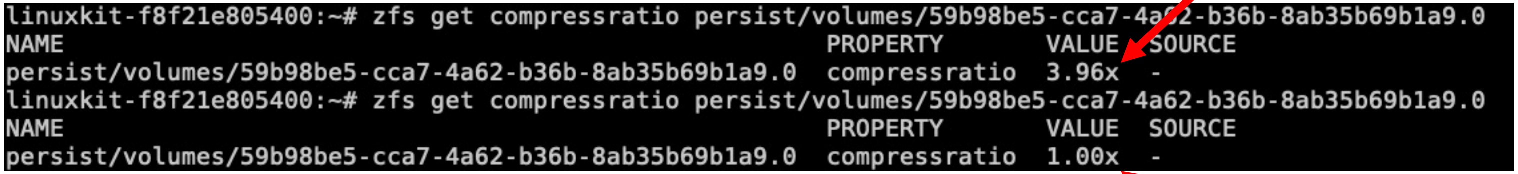


```
   0[||||||||67.8%]   6[||||||||58.1%]     12[||||||||66.4%]  18[||||||||59.3%]
   1[||||||||60.4%]   7[||||||||65.7%]     13[||||||||63.3%]  19[||||||||59.4%]
   2[||||||||69.0%]   8[||||||||59.4%]     14[||||||||63.9%]  20[|||||||  56.1%]
   3[||||||||64.1%]   9[||||||||61.9%]     15[||||||||59.5%]  21[|||||||  54.5%]
   4[||||||||63.0%]  10[||||||||58.7%]     16[|||||||  53.8%]  22[||||||||61.2%]
   5[||||||||66.2%]  11[||||||||63.3%]     17[||||||||58.7%]  23[|||||||  55.5%]
  Mem[|||||||||||||||||      25.2G/62.5G]  Tasks: 64, 272 thr; 7 running
  Swp[                             0K/0K]  Load average: 26.69 24.47 20.03
                                           Uptime: 01:22:39

  PID USER      PRI  NI  VIRT   RES   SHR S CPU%▽MEM%   TIME+  Command
 2511 root       20   0  154M 57244 21300 S  0.0  0.1  0:00.18 /usr/bin/containerd
 2512 root       20   0  154M 57244 21300 S  0.0  0.1  0:00.00 /usr/bin/containerd
 2513 root       20   0  154M 57244 21300 S  0.0  0.1  0:00.24 /usr/bin/containerd
 2514 root       20   0  154M 57244 21300 S  0.0  0.1  0:00.45 /usr/bin/containerd
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

```
  1  [||      8.5%]   6  [          0.0%]   11 [          0.0%]   16 [          0.0%]
  2  [|||||  28.5%]   7  [          0.0%]   12 [||||||| 43.3%]   17 [|         0.7%]
  3  [        0.0%]   8  [          0.0%]   13 [|         2.0%]   18 [          0.0%]
  4  [        0.0%]   9  [          0.0%]   14 [||||||||77.7%]   19 [          0.0%]
  5  [        0.0%]   10 [          0.0%]   15 [|         4.5%]
  Mem[||||||||||||||||||||||||||478M/23.5G]  Tasks: 35, 13 thr; 2 running
  Swp[                              0K/0K]  Load average: 1.61 1.56 1.32
                                            Uptime: 01:16:31

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 3130 pocuser    20   0  599M  4896  1408 R 77.1  0.0  0:27.83 fio fio-place-data.job --o
 3101 pocuser    20   0 32416  4496  3644 R  0.0  0.0  0:00.47 htop
    1 root       20   0  220M  5456  2912 S  0.0  0.0  0:02.90 /sbin/init
  580 root       19  -1  100M  8556  7808 S  0.0  0.0  0:00.30 /lib/systemd/systemd-journ
  601 root       20   0 97716   176     0 S  0.0  0.0  0:00.00 /sbin/lvmetad -f
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

# ZFS compression ratios – FIO –refill_buffers

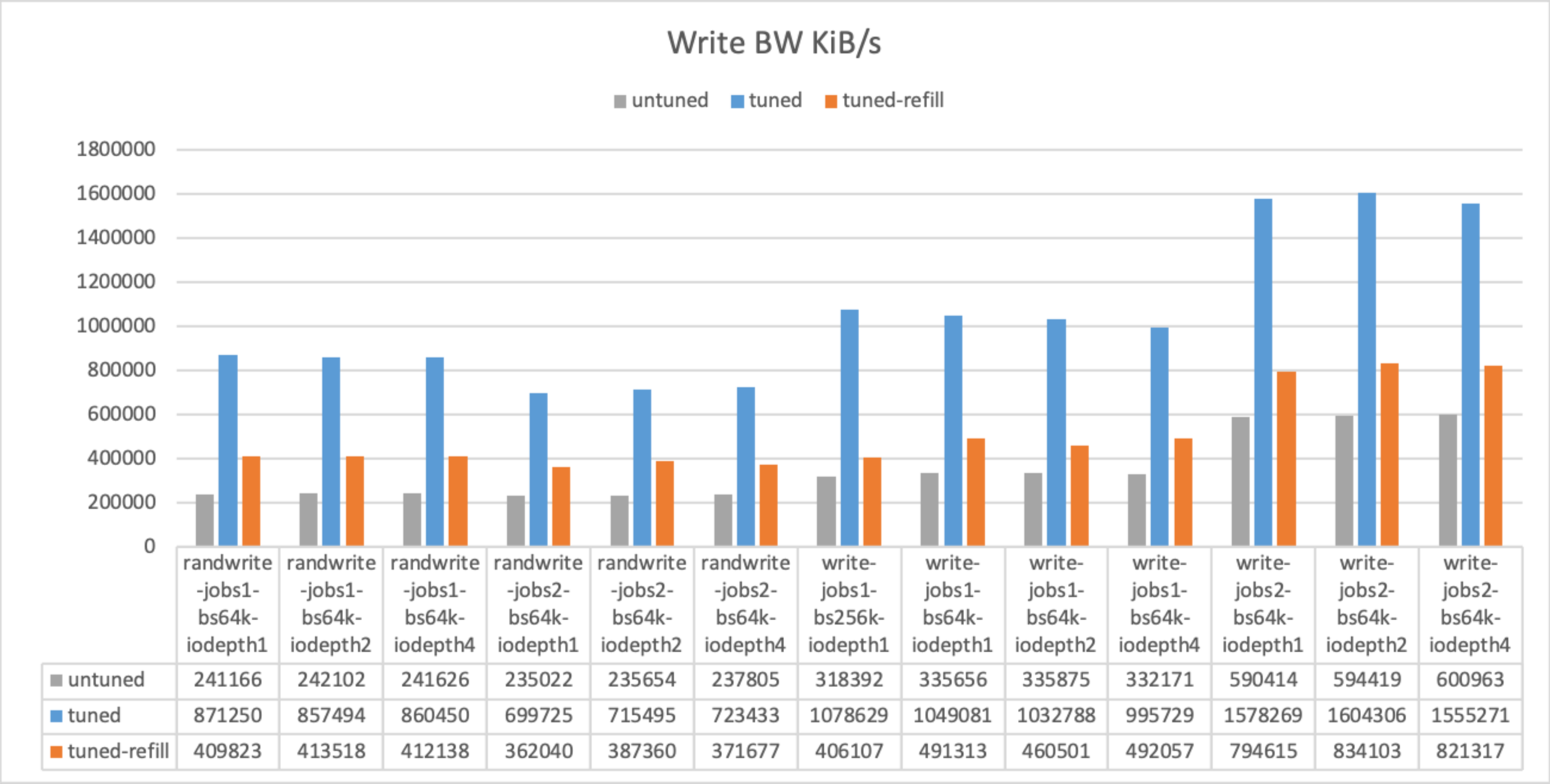
Tuned



```
linuxkit-f8f21e805400:~# zfs get compressratio persist/volumes/59b98be5-cca7-4a62-b36b-8ab35b69b1a9.0
NAME                                          PROPERTY       VALUE  SOURCE
persist/volumes/59b98be5-cca7-4a62-b36b-8ab35b69b1a9.0  compressratio  3.96x  -
linuxkit-f8f21e805400:~# zfs get compressratio persist/volumes/59b98be5-cca7-4a62-b36b-8ab35b69b1a9.0
NAME                                          PROPERTY       VALUE  SOURCE
persist/volumes/59b98be5-cca7-4a62-b36b-8ab35b69b1a9.0  compressratio  1.00x  -
```

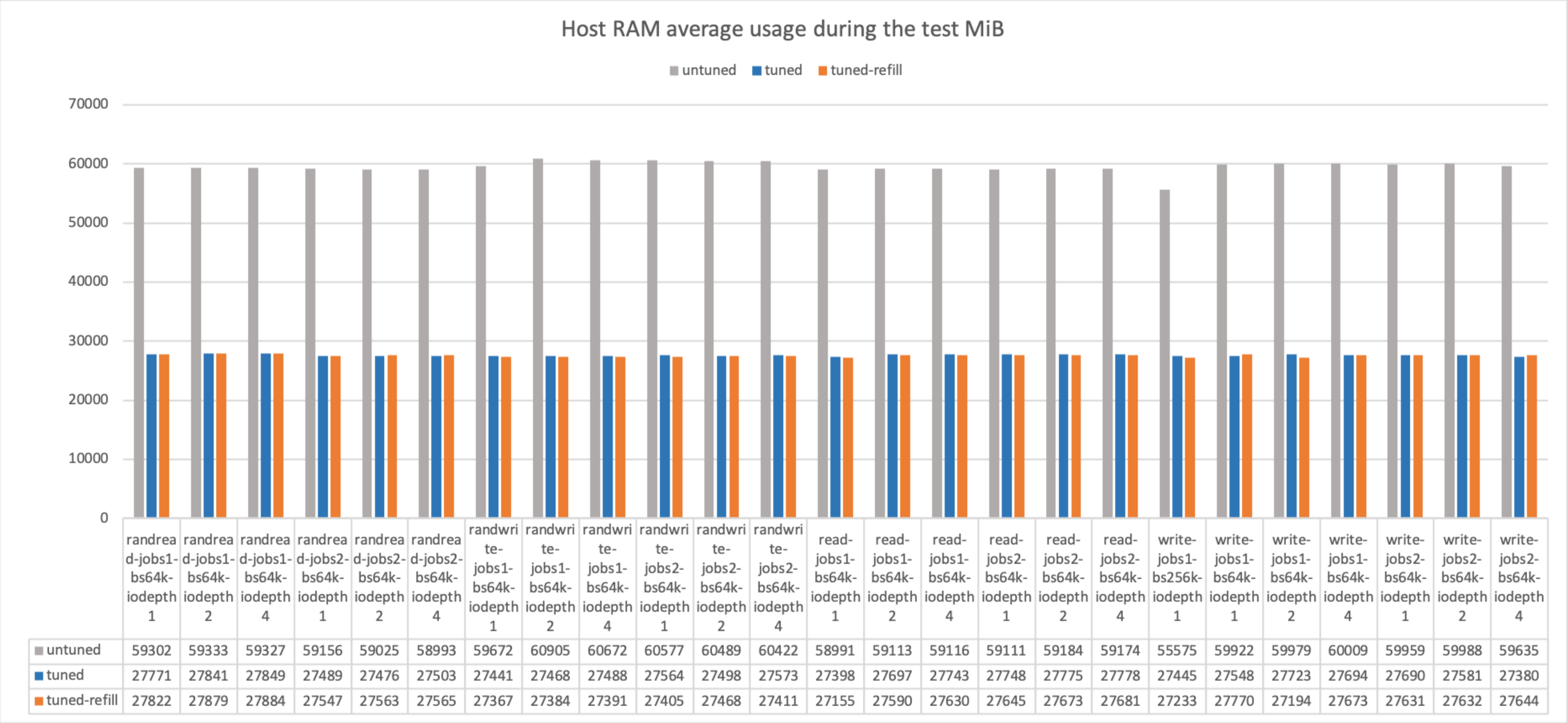
Tuned-refill

# ZFS Klara efforts



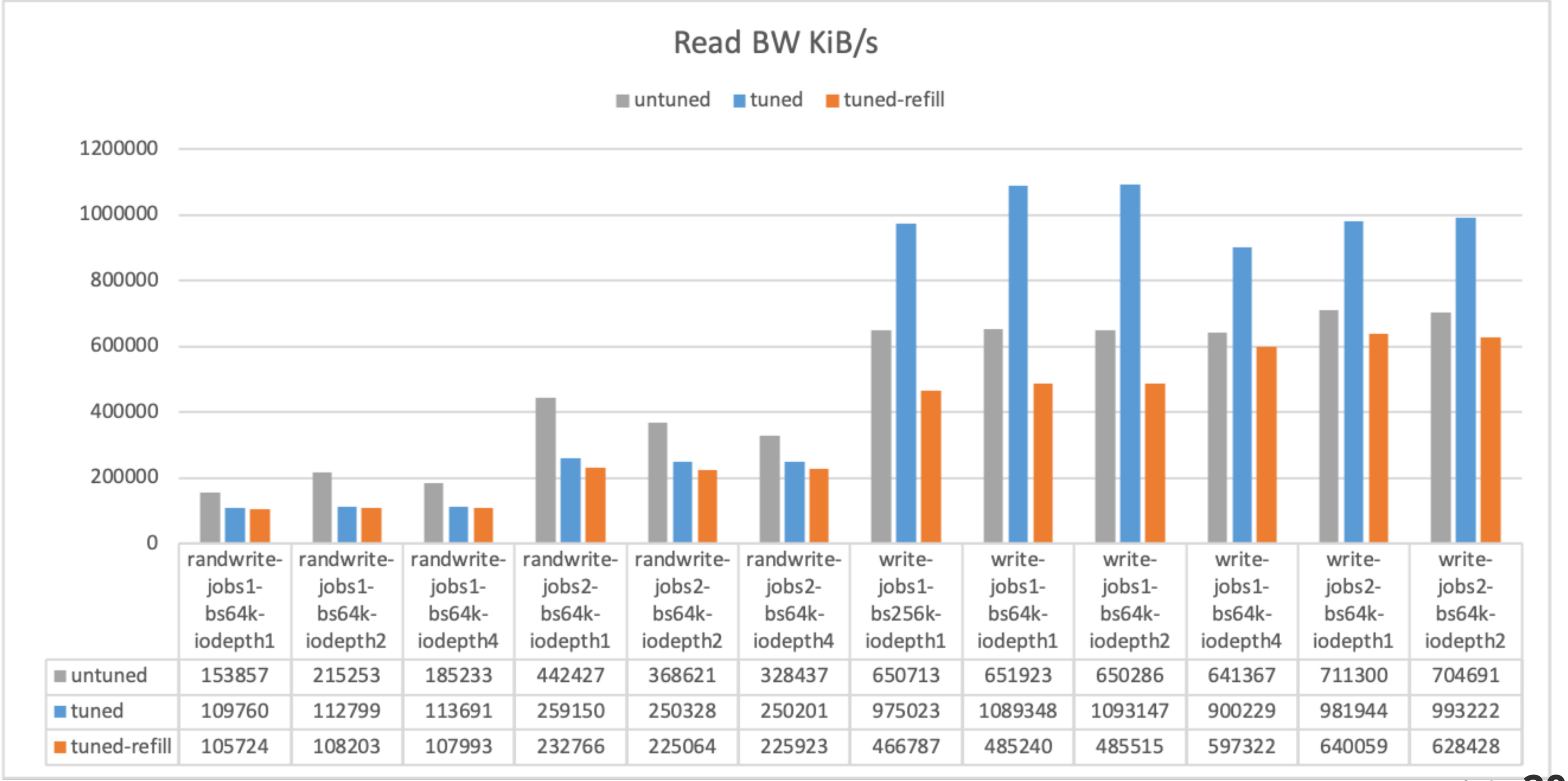
| | randwrite-jobs1-bs64k-iodepth1 | randwrite-jobs1-bs64k-iodepth2 | randwrite-jobs1-bs64k-iodepth4 | randwrite-jobs2-bs64k-iodepth1 | randwrite-jobs2-bs64k-iodepth2 | randwrite-jobs2-bs64k-iodepth4 | write-jobs1-bs256k-iodepth1 | write-jobs1-bs64k-iodepth1 | write-jobs1-bs64k-iodepth2 | write-jobs1-bs64k-iodepth4 | 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 Klara efforts



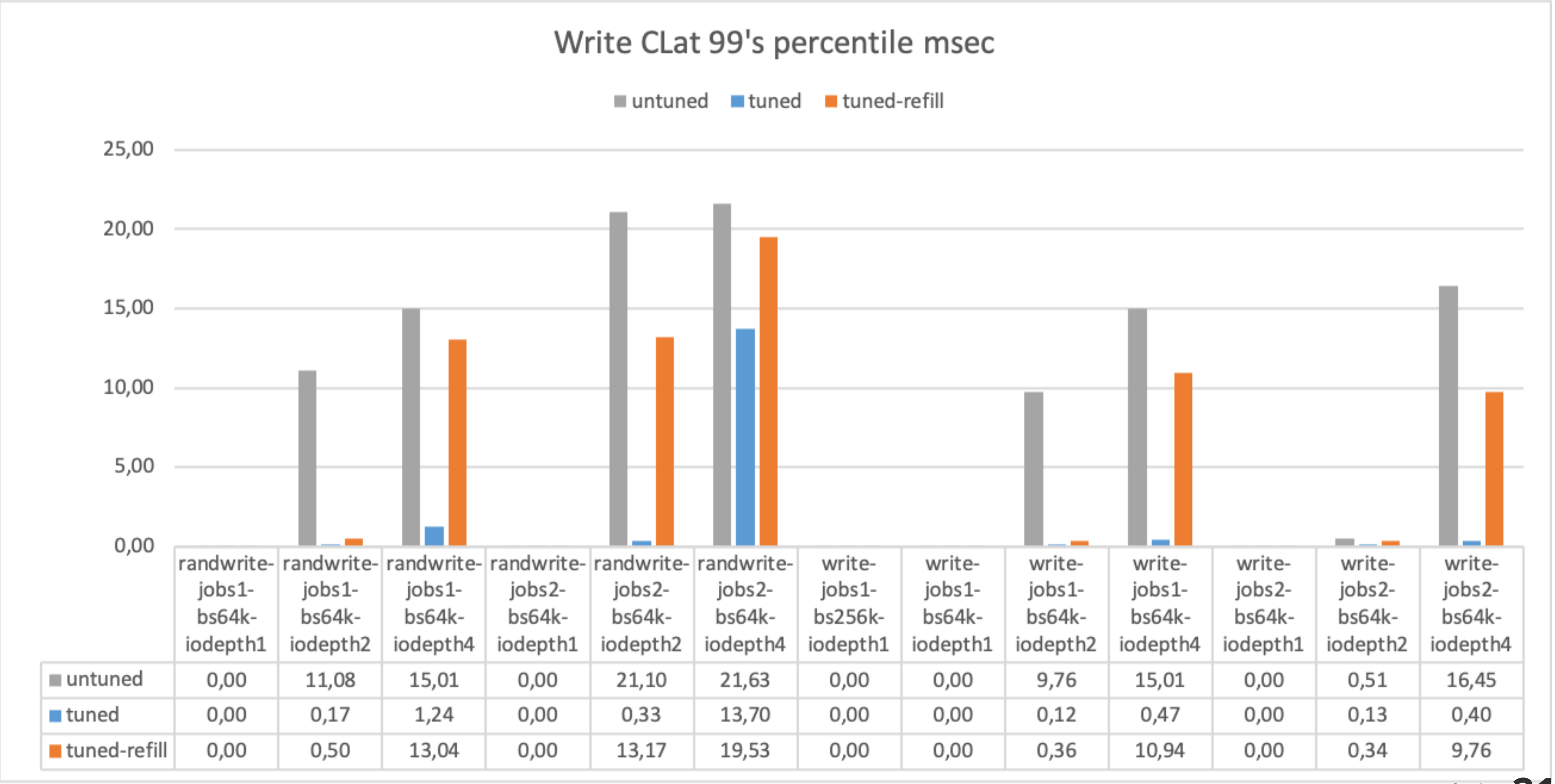
| | randread-jobs1-bs64k-iodepth 1 | randread-jobs1-bs64k-iodepth 2 | randread-jobs1-bs64k-iodepth 4 | randread-jobs2-bs64k-iodepth 1 | randread-jobs2-bs64k-iodepth 2 | randread-jobs2-bs64k-iodepth 4 | randwrite-jobs1-bs64k-iodepth 1 | randwrite-jobs1-bs64k-iodepth 2 | randwrite-jobs1-bs64k-iodepth 4 | 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 Klara efforts



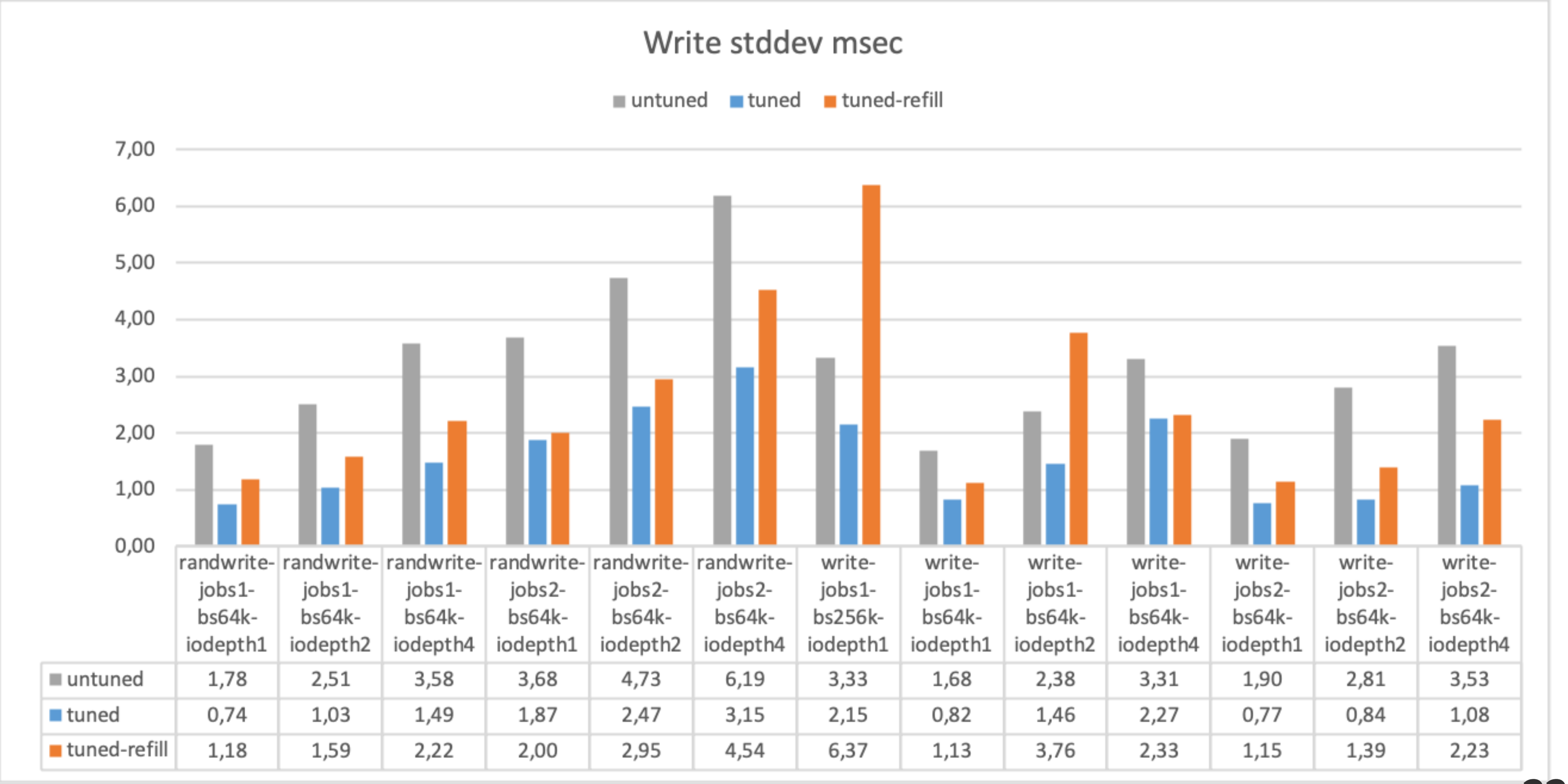
| | randwrite-jobs1-bs64k-iodepth1 | randwrite-jobs1-bs64k-iodepth2 | randwrite-jobs1-bs64k-iodepth4 | randwrite-jobs2-bs64k-iodepth1 | randwrite-jobs2-bs64k-iodepth2 | randwrite-jobs2-bs64k-iodepth4 | write-jobs1-bs256k-iodepth1 | write-jobs1-bs64k-iodepth1 | write-jobs1-bs64k-iodepth2 | write-jobs1-bs64k-iodepth4 | 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 Klara efforts



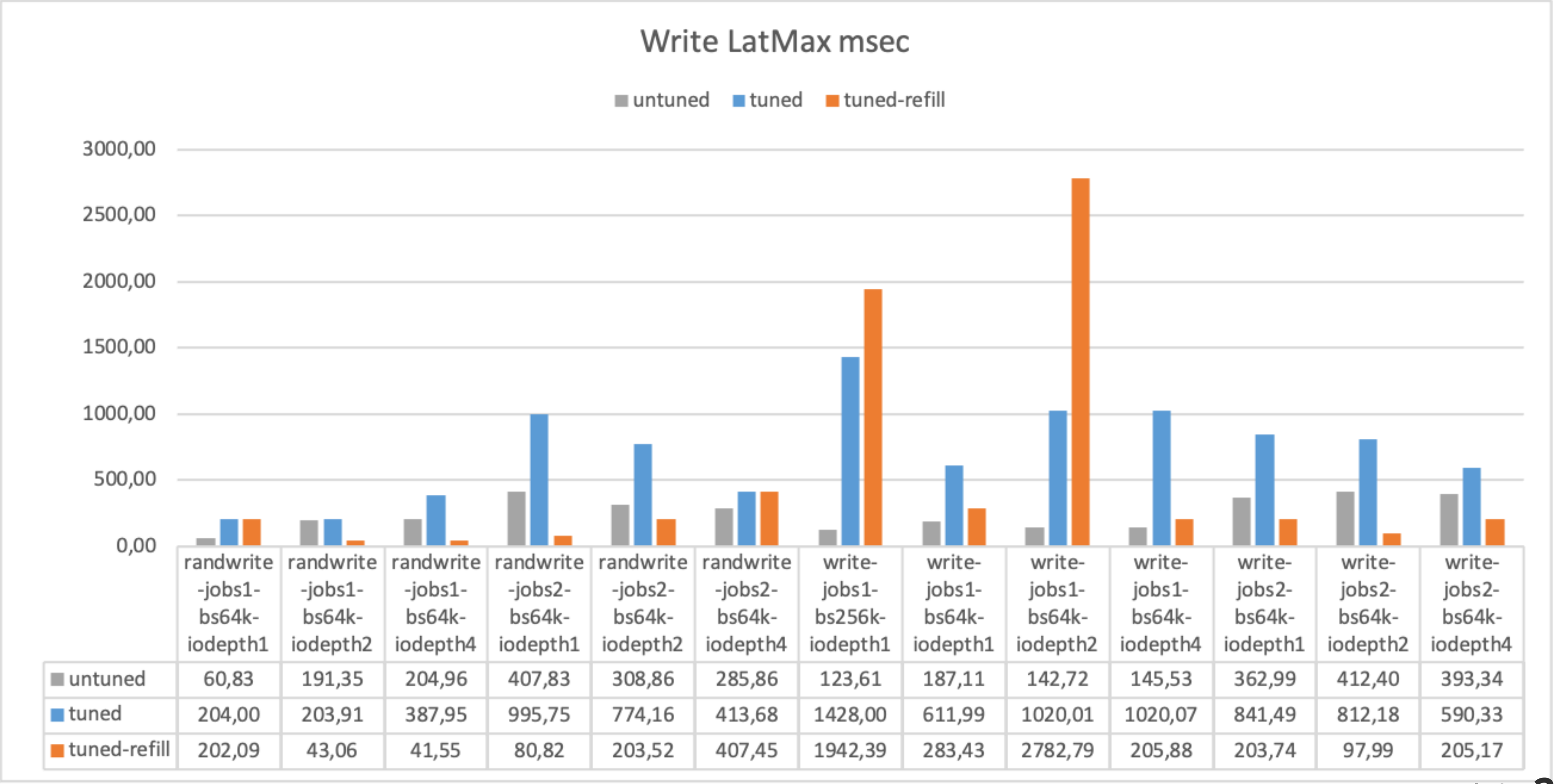
| | randwrite-jobs1-bs64k-iodepth1 | randwrite-jobs1-bs64k-iodepth2 | randwrite-jobs1-bs64k-iodepth4 | randwrite-jobs2-bs64k-iodepth1 | randwrite-jobs2-bs64k-iodepth2 | randwrite-jobs2-bs64k-iodepth4 | write-jobs1-bs256k-iodepth1 | write-jobs1-bs64k-iodepth1 | write-jobs1-bs64k-iodepth2 | write-jobs1-bs64k-iodepth4 | 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 Klara 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 | 1,78 | 2,51 | 3,58 | 3,68 | 4,73 | 6,19 | 3,33 | 1,68 | 2,38 | 3,31 | 1,90 | 2,81 | 3,53 |
| tuned | 0,74 | 1,03 | 1,49 | 1,87 | 2,47 | 3,15 | 2,15 | 0,82 | 1,46 | 2,27 | 0,77 | 0,84 | 1,08 |
| tuned-refill | 1,18 | 1,59 | 2,22 | 2,00 | 2,95 | 4,54 | 6,37 | 1,13 | 3,76 | 2,33 | 1,15 | 1,39 | 2,23 |

# ZFS Klara 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 Klara efforts - Summary

- Given the machine is powerfull enough, performance improvement is dramatic (thanks to write amplification patch)
- ARC size manipulations allowed to reduce memory consumption significanlty
  - hard limit is 1.6 GiB to server 460 GiB storage
- Compression ratio 4 is not uncommon in real life, but even with uncompressable data performance has noticably improoved
- Latency can still go really high, but sttdev and p99 are very low – must be just a few outliers in the sample data

# ZFS support – important missing pieces

- Disk and pool health reports
- Prevent pool from filling up more then 80 %
- Final bits of Klara’s effort
  - Upstreaming patches (WIP)
  - Tunable integration (WIP)

# 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.”

# ZFS WIP: 20% disk space reservation

- Add a dummy data-set, whose only job is taking 20% of space from the free space pool
- Will be keept empty all the time
- If pool does not have any free space, and user needs to overwrite existing data, some blocks will be borrowed from the dummy data-set to write new copies of modified blocks
- Eventually the original blocks will be freed and will be used to pay the debt to dummy dataset

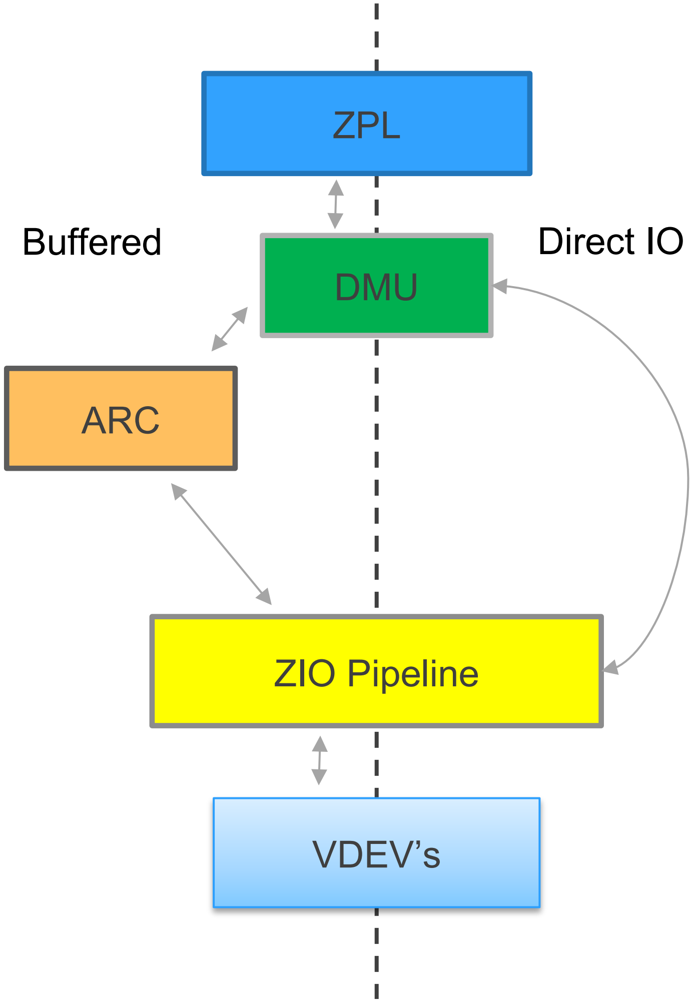
```
zfs create -o refreservation=(20% of space) poolname/reserved
```

# ZFS deployment

- No transition for existing deployments, only newly created edge nodes
- Therefore will have to support 2 architectures for a while
- Minimal supported configuration (e.g. no zfs on RaspberiPi)
- Pilot with customers who require software RAID

# Implementation: Direct IO Read in ZFS (Big Picture)

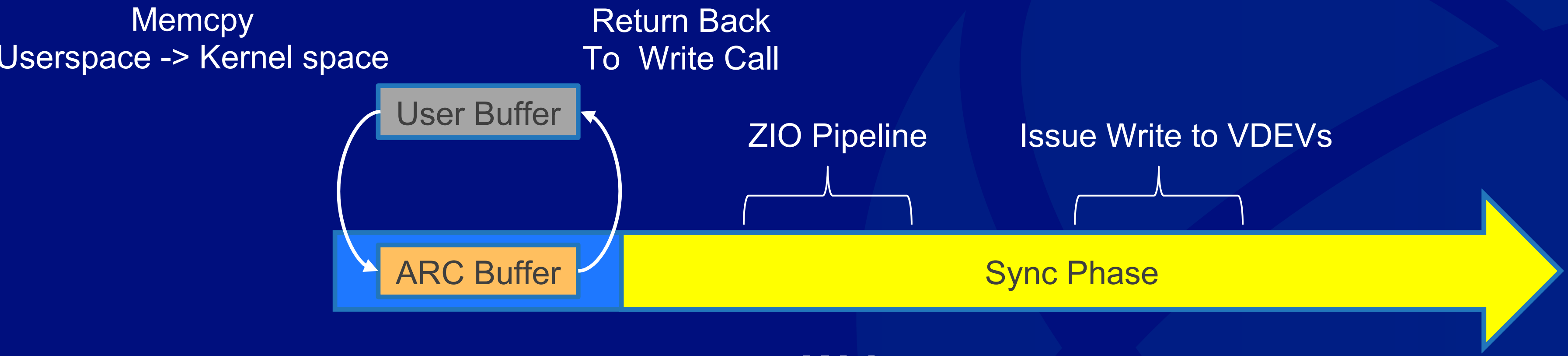
- Buffered
  - Cached? → Copy from ARC
  - Issue to ZFS pipeline
    - Copy to ARC
    - Copy to user buffer
- Direct IO
  - Bypass ARC
  - User pages are directly mapped into an ABD



Los Alamos NATIONAL LABORATORY

# Implementation: Direct IO Write in ZFS (Big Picture)

## Buffered IO Write Path



Memcpy
Userspace -> Kernel space

Return Back
To Write Call

User Buffer

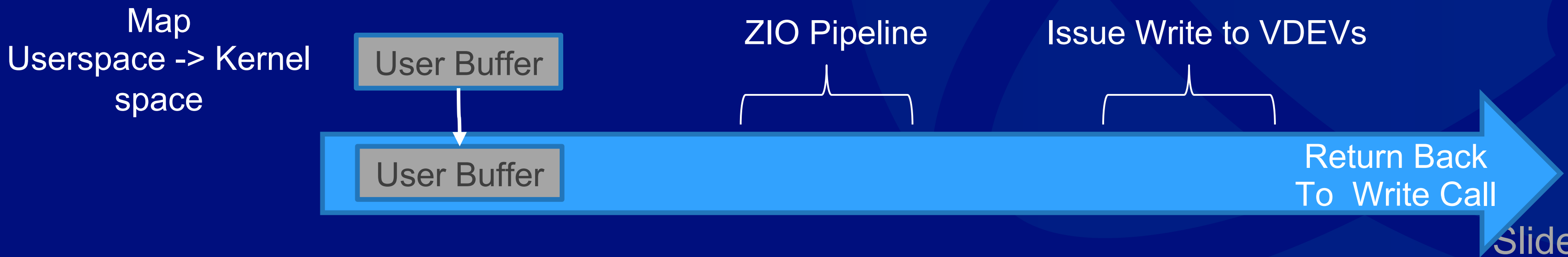
ARC Buffer

ZIO Pipeline

Issue Write to VDEVs

Sync Phase

## Direct IO Write Path



Map
Userspace -> Kernel
space

User Buffer

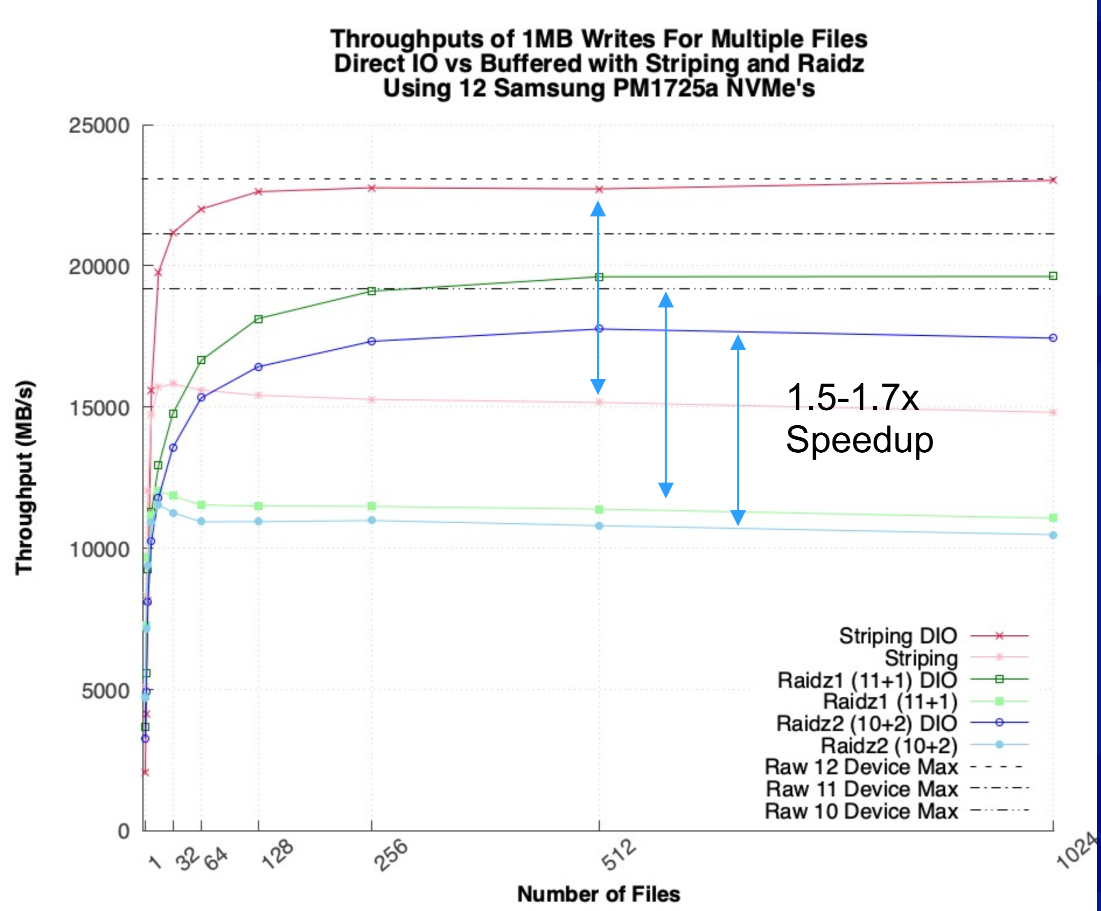
User Buffer

ZIO Pipeline

Issue Write to VDEVs

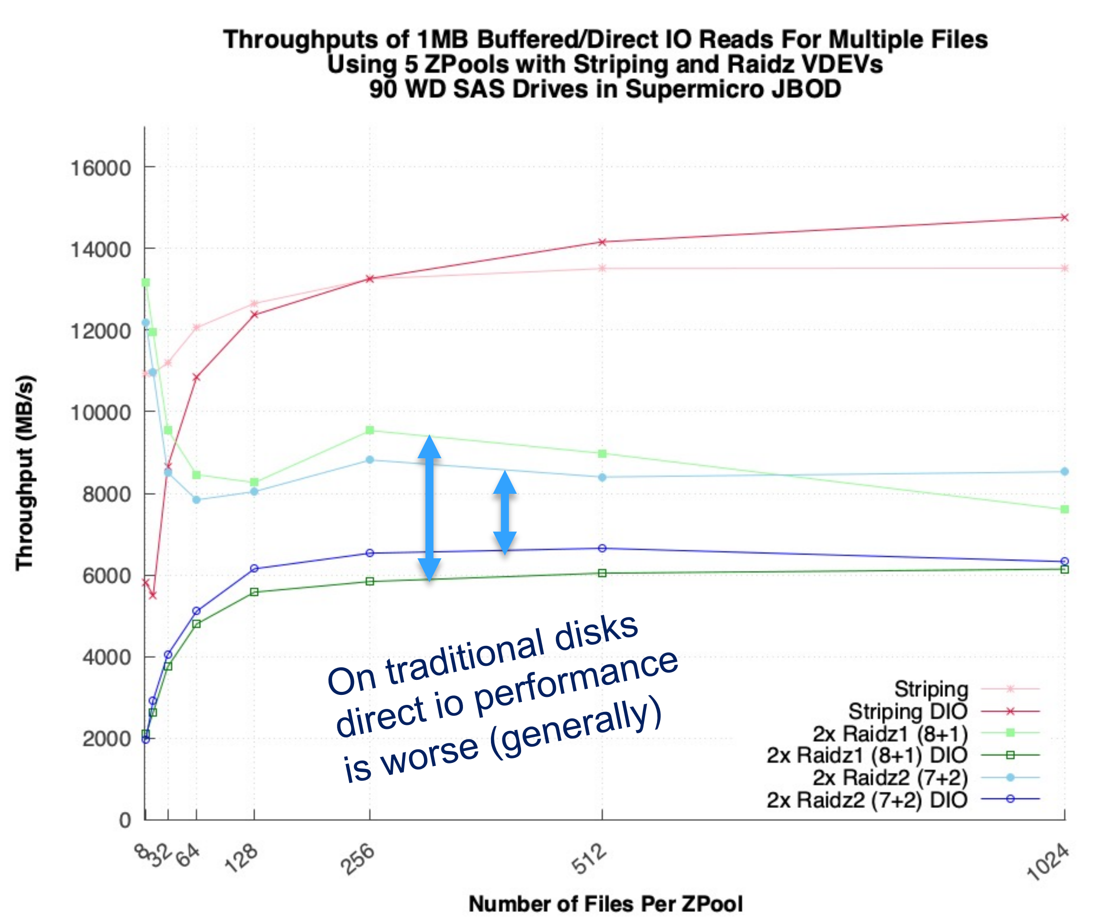
Return Back
To Write Call

# Seq. Write Performance Results: ZFS NVMe Zpools



1.5-1.7x Speedup

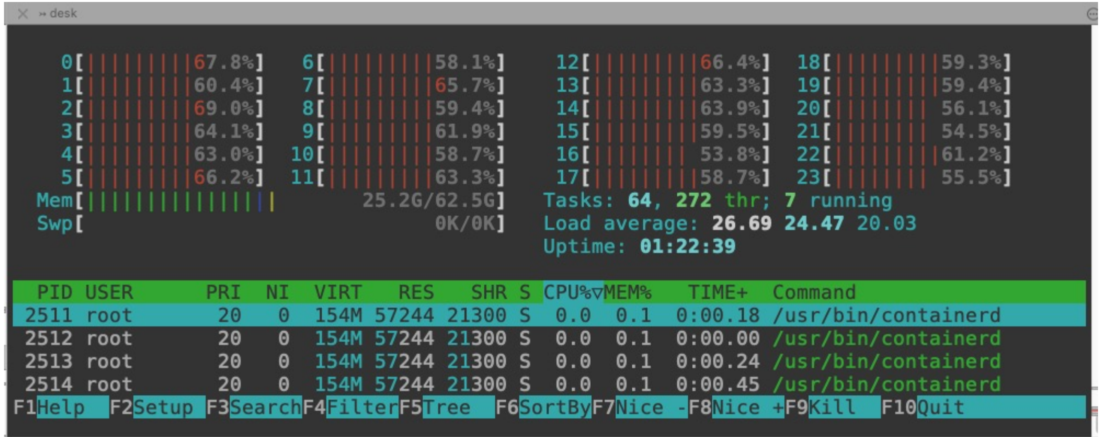
# Seq. Read Performance Results: JBOD with ZFS



On traditional disks direct io performance is worse (generally)

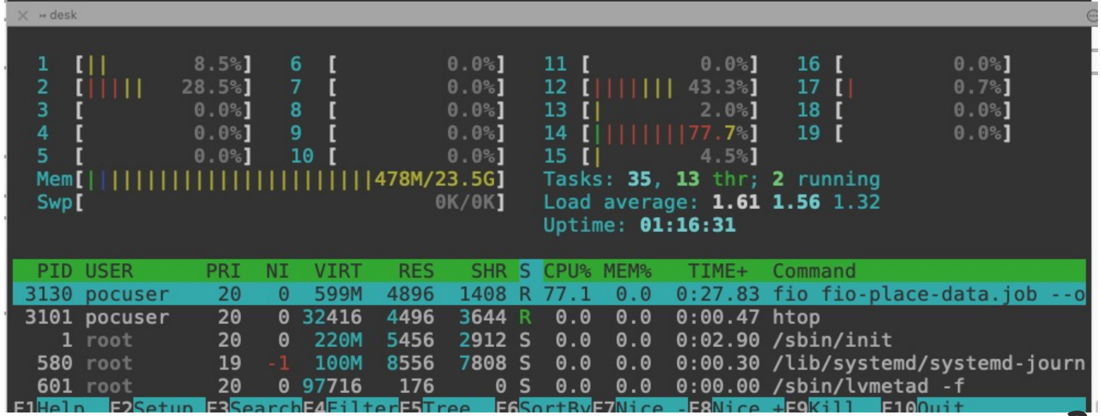
# ZFS future: Adaptive Compression depending on system load

Host



```
» desk

    0[|||||||||67.8%]   6[|||||||||58.1%]    12[|||||||||66.4%]  18[|||||||||59.3%]
    1[|||||||||60.4%]   7[|||||||||65.7%]    13[|||||||||63.3%]  19[|||||||||59.4%]
    2[|||||||||69.0%]   8[|||||||||59.4%]    14[|||||||||63.9%]  20[||||||||  56.1%]
    3[|||||||||64.1%]   9[|||||||||61.9%]    15[|||||||||59.5%]  21[||||||||  54.5%]
    4[|||||||||63.0%]  10[|||||||||58.7%]    16[||||||||  53.8%]  22[|||||||||61.2%]
    5[|||||||||66.2%]  11[|||||||||63.3%]    17[|||||||||58.7%]  23[||||||||  55.5%]
  Mem[||||||||||||||||        25.2G/62.5G]  Tasks: 64, 272 thr; 7 running
  Swp[                              0K/0K]  Load average: 26.69 24.47 20.03
                                            Uptime: 01:22:39

  PID USER      PRI  NI  VIRT   RES   SHR S CPU%▽MEM%   TIME+  Command
 2511 root       20   0  154M 57244 21300 S  0.0  0.1  0:00.18 /usr/bin/containerd
 2512 root       20   0  154M 57244 21300 S  0.0  0.1  0:00.00 /usr/bin/containerd
 2513 root       20   0  154M 57244 21300 S  0.0  0.1  0:00.24 /usr/bin/containerd
 2514 root       20   0  154M 57244 21300 S  0.0  0.1  0:00.45 /usr/bin/containerd
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

Guest



```
» desk

  1  [||       8.5%]   6  [           0.0%]   11 [          0.0%]   16 [           0.0%]
  2  [|||||   28.5%]   7  [           0.0%]   12 [|||||||  43.3%]   17 [|          0.7%]
  3  [         0.0%]   8  [           0.0%]   13 [|         2.0%]   18 [           0.0%]
  4  [         0.0%]   9  [           0.0%]   14 [||||||||77.7%]   19 [           0.0%]
  5  [         0.0%]   10 [           0.0%]   15 [|         4.5%]
  Mem[||||||||||||||||||||||||478M/23.5G]   Tasks: 35, 13 thr; 2 running
  Swp[                            0K/0K]   Load average: 1.61 1.56 1.32
                                            Uptime: 01:16:31

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 3130 pocuser    20   0  599M  4896  1408 R 77.1  0.0  0:27.83 fio fio-place-data.job --o
 3101 pocuser    20   0 32416  4496  3644 R  0.0  0.0  0:00.47 htop
    1 root       20   0  220M  5456  2912 S  0.0  0.0  0:02.90 /sbin/init
  580 root       19  -1  100M  8556  7808 S  0.0  0.0  0:00.30 /lib/systemd/systemd-journ
  601 root       20   0 97716   176     0 S  0.0  0.0  0:00.00 /sbin/lvmetad -f
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

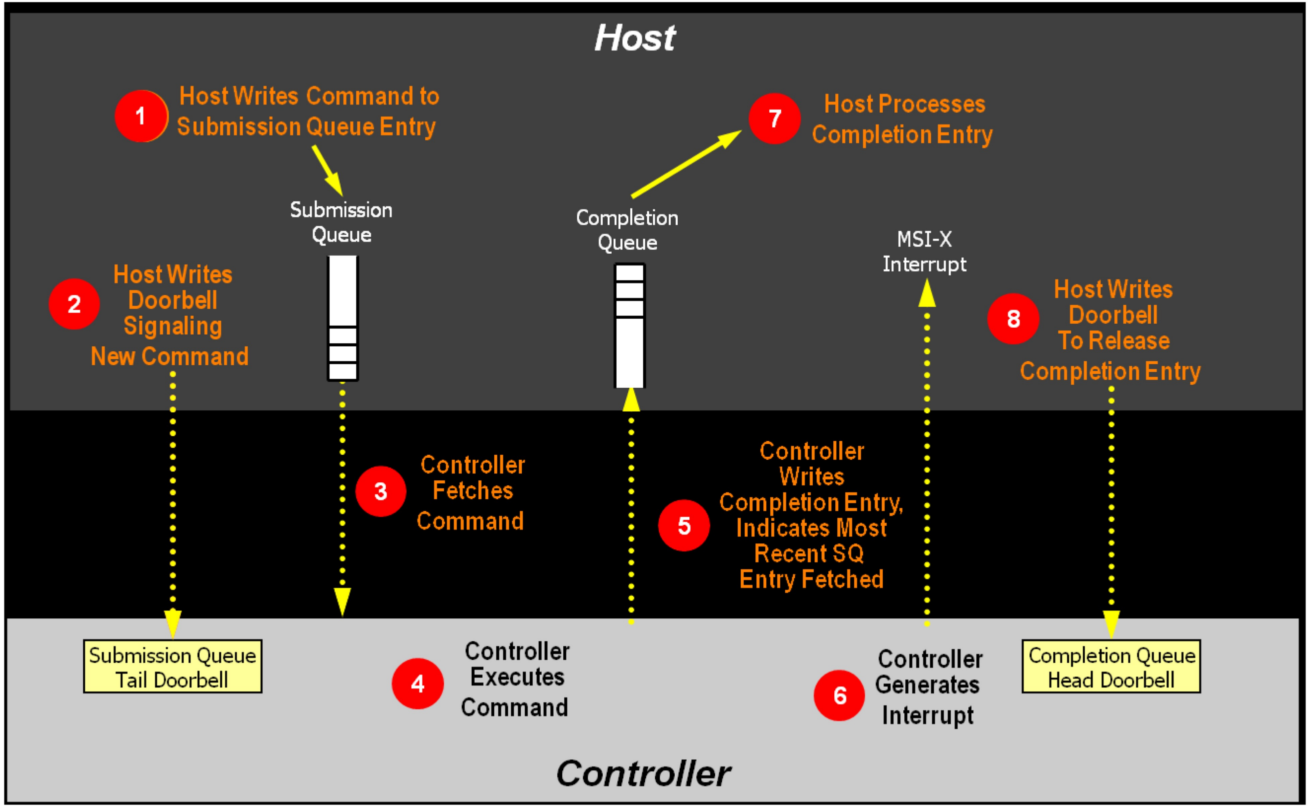
# ZFS Future Ideas

- 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`**)

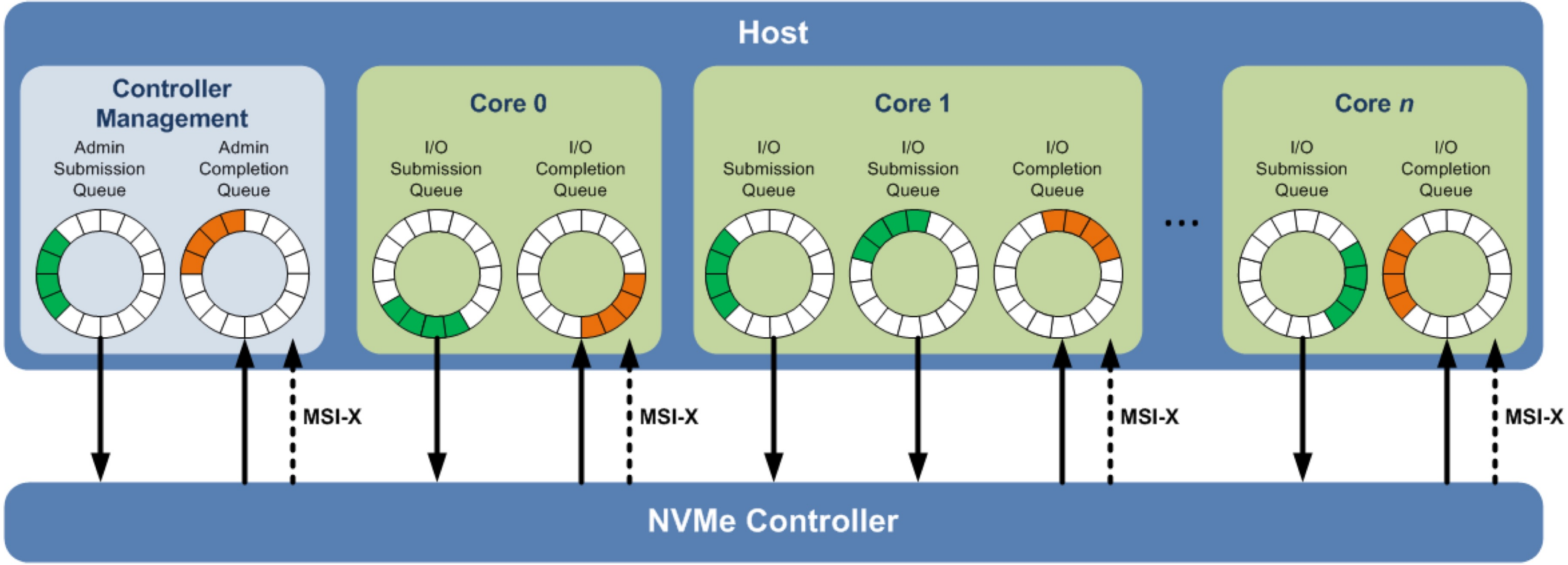
# ZFS Future Ideas: namespaces support

- Is it possible to make zfs threads aware from which cgroup the data is coming?
- To reduce ”noisy neighbour” effects

# NVMe background


Host
1 Host Writes Command to Submission Queue Entry
Submission Queue
Completion Queue
7 Host Processes Completion Entry
2 Host Writes Doorbell Signaling New Command
MSI-X Interrupt
8 Host Writes Doorbell To Release Completion Entry
3 Controller Fetches Command
5 Controller Writes Completion Entry, Indicates Most Recent SQ Entry Fetched
Submission Queue Tail Doorbell
4 Controller Executes Command
6 Controller Generates Interrupt
Completion Queue Head Doorbell
Controller

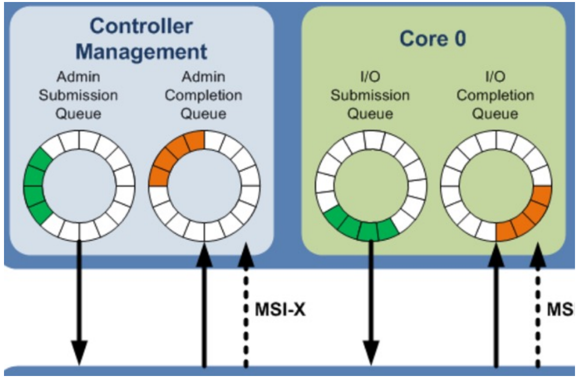
# NVMe background



# 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

# Tooling: Eve development builds

- Enabling some of the dangerous stuff
- E.g. configuration overrides (/config/storage.cfg)
- If onboarded should zcontrol should be showing lots of red banners all the time showing this device is not supported

# Improve testing coverage

- Block device correctness
- Sudden power off (semi-automated implementation exists, integration in the proper test suit is required)
- FIO with data verification

# Enable LAB for remote development

- Latencies US-Europe are insanely painful
  - Would be cool to start buidling a LAB in Europe
  - .. or fix the damn latency
- TrueNAS running in LAB network
  - iSCSI/NFS/ftp boot
  - Quick switch back and forth between different OS (e.g. Debian->EveOS->RHEL)
  - Storing app images to not wait for ages while qcow2 is beeing downloaded
- Console access (to see boot messages, enter BIOS menu)
- Power controll
- One shared jump server to run tmux on

# Thank you!

THE LINUX FOUNDATION

LF EDGE

# Backup

# 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>