TrueNAS

FreeBSD, ZFS and iSCSI

or one year of TrueNAS development

Alexander Motin
mav@ixsystems.com

iXsystems

About myself

Alexander Motin

OS/Services team leader at iXsystems

- FreeBSD src committer since 2007
  <mav@FreeBSD.org>

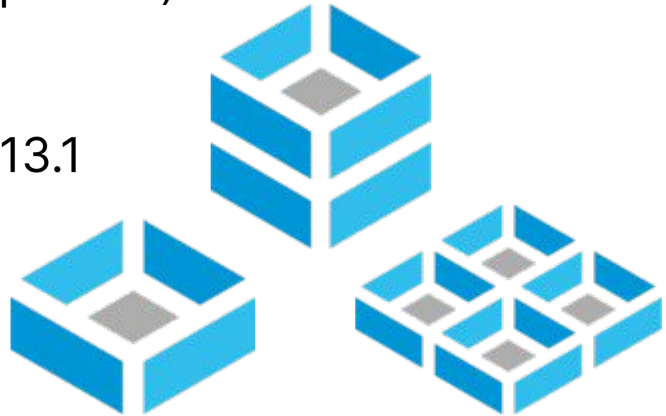
- With iXsystems since 2009
  <mav@iXsystems.com>

About TrueNAS

Software

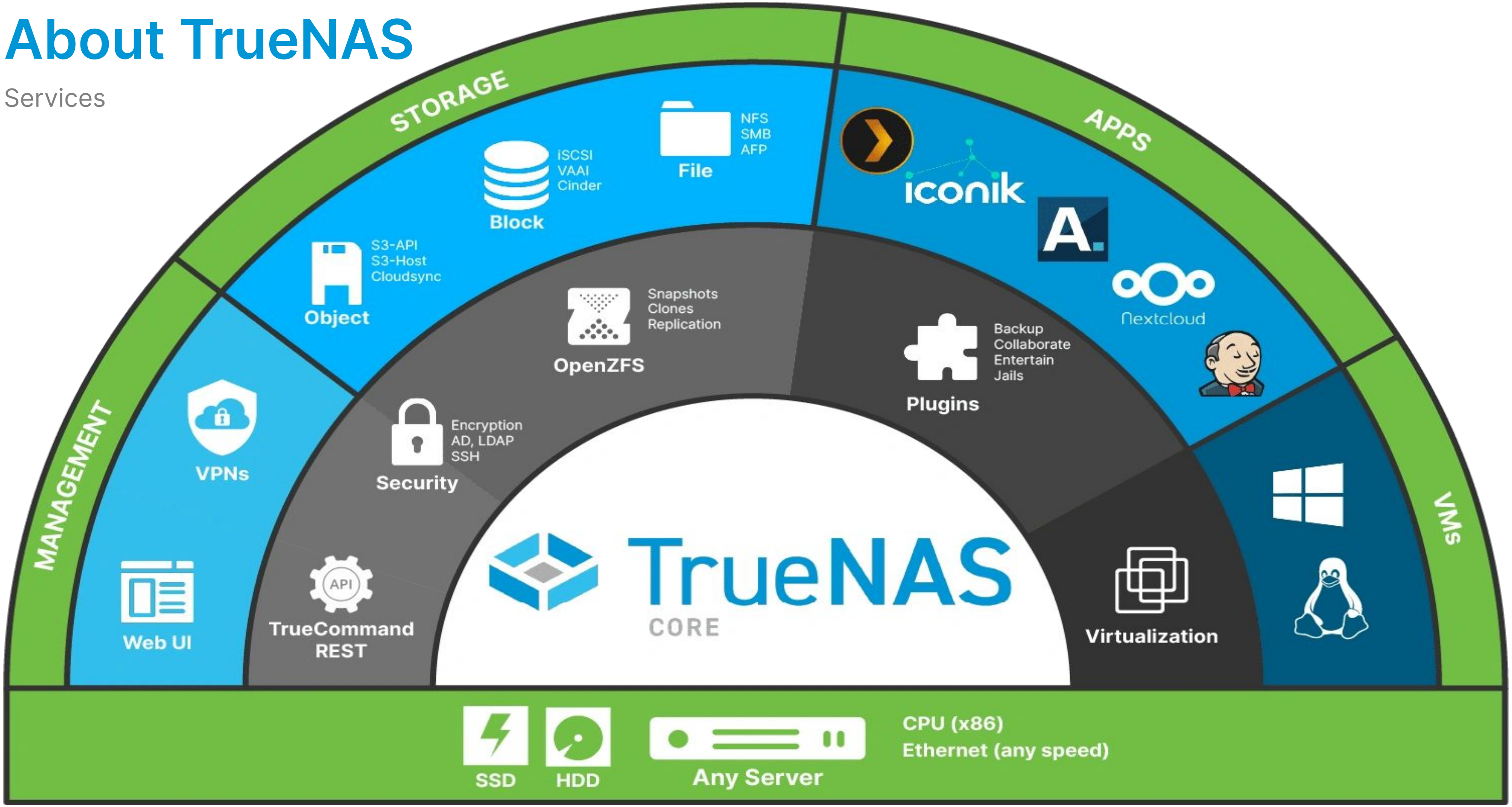
- 2005 Started as FreeNAS
- 2009 Taken over by iXsystems
- 2013 TrueNAS – appliance edition of FreeNAS
- 2020 FreeNAS → TrueNAS Core (community)
  TrueNAS → TrueNAS Enterprise (appliance)
- 2021 Started TrueNAS SCALE (scale-out)

TrueNAS Core/Enterprise – FreeBSD 12.2 → 13.1
TrueNAS SCALE – Debian 11.



About TrueNAS

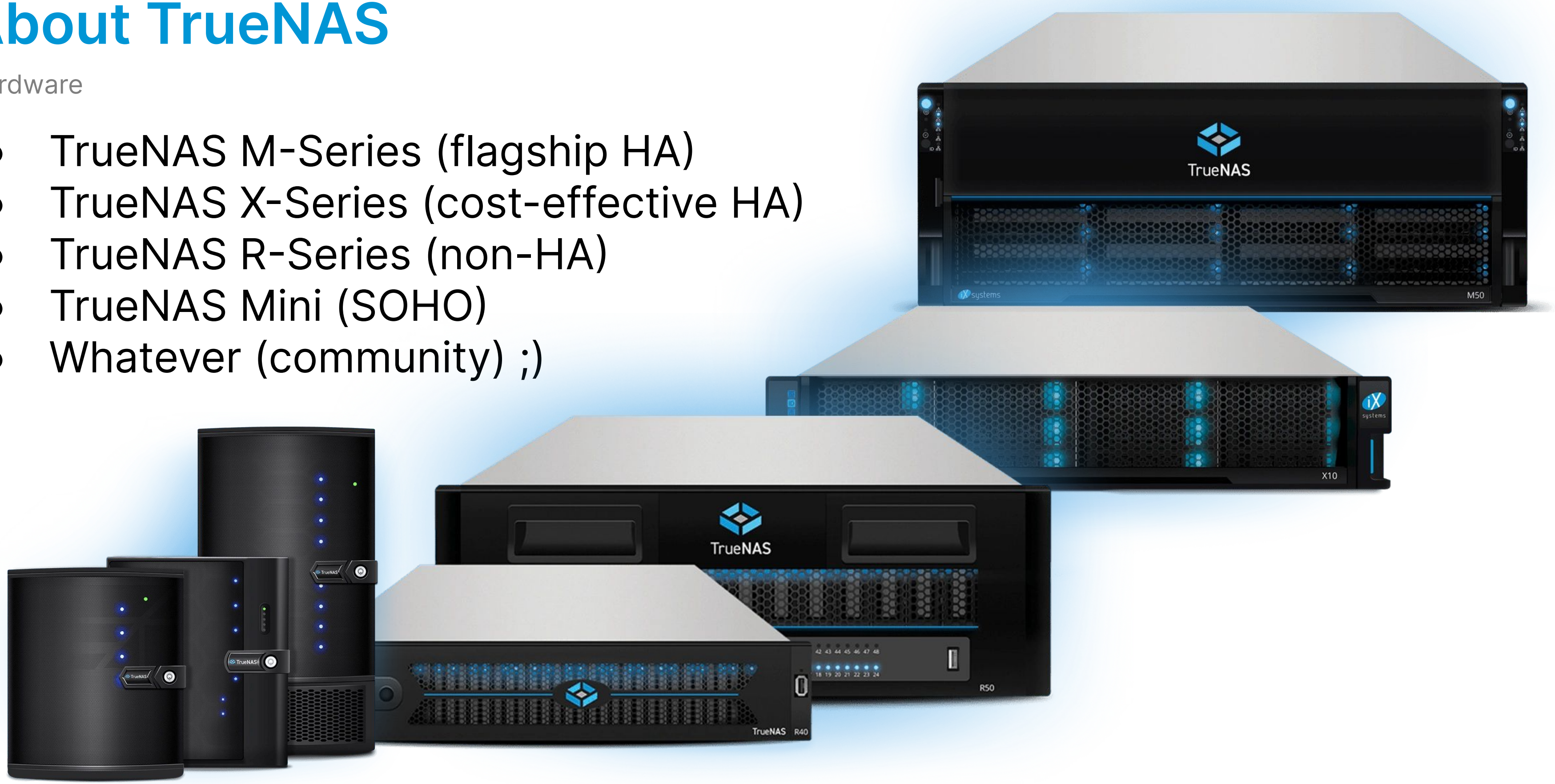
Services



About TrueNAS

Hardware

- TrueNAS M-Series (flagship HA)
- TrueNAS X-Series (cost-effective HA)
- TrueNAS R-Series (non-HA)
- TrueNAS Mini (SOHO)
- Whatever (community) ;)



From UFS to OpenZFS

History of ZFS in TrueNAS

- 2005 FreeNAS started with UFS
- 2010 FreeNAS switched to FreeBSD ZFS port
- 2020 TrueNAS 12 switched to OpenZFS 2.0 –
  re-integration of FreeBSD ZFS and ZFS-on-Linux
- 2021 Core/Enterprise/SCALE unified by OpenZFS 2.1

FreeBSD main – OpenZFS master
FreeBSD stable/13 – OpenZFS 2.1

Thanks to Matt Macy, Ryan Moeller and others for the OpenZFS re-integration!

TrueNAS

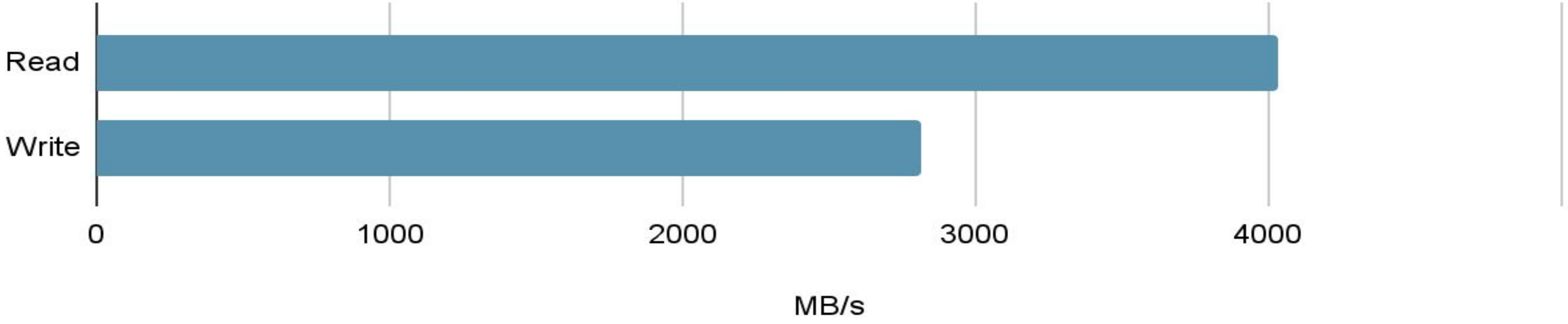
Enough marketing,
lets go engineering!

iXsystems

ZFS-backed iSCSI target benchmarking

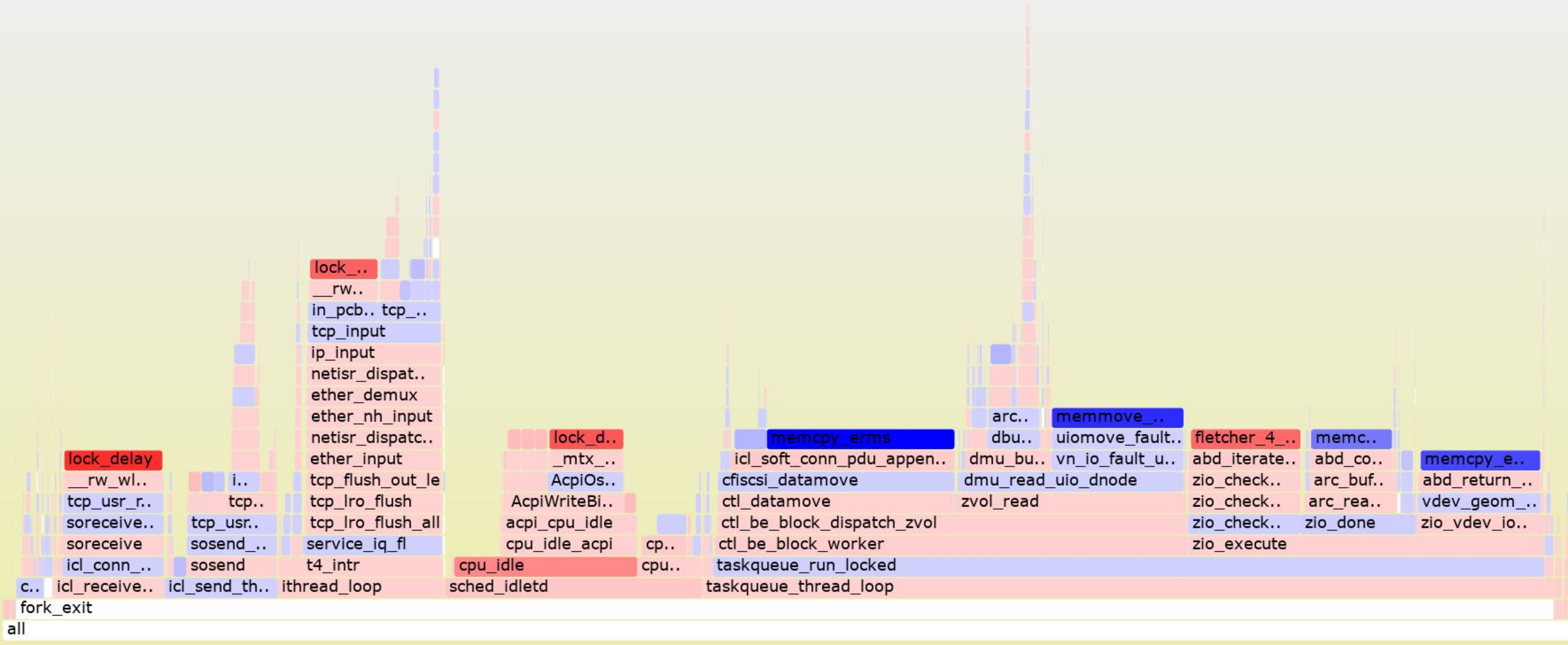
Baseline performance test

- Hardware: 2x Xeon Gold 6242R, 768GB RAM, 9 NVMe SSDs, Chelsio T62100 @ 100Gbps. Initiator: Core i7, Chelsio T62100.
- Software: FreeBSD main from June 2020, native FreeBSD ZFS, CTL.
- Target configuration: striped ZFS pool of 9 NVMe SSDs, ARC limited to 12GB, single ZVOL with 64KB block size, single iSCSI target LUN.
- Initiator configuration: Windows 10, software initiator tuned for large I/O.
- Test: CrystalDiskMark, sequential 256KB read/write over 64GB, Q32T2



ZFS-backed iSCSI target profiling

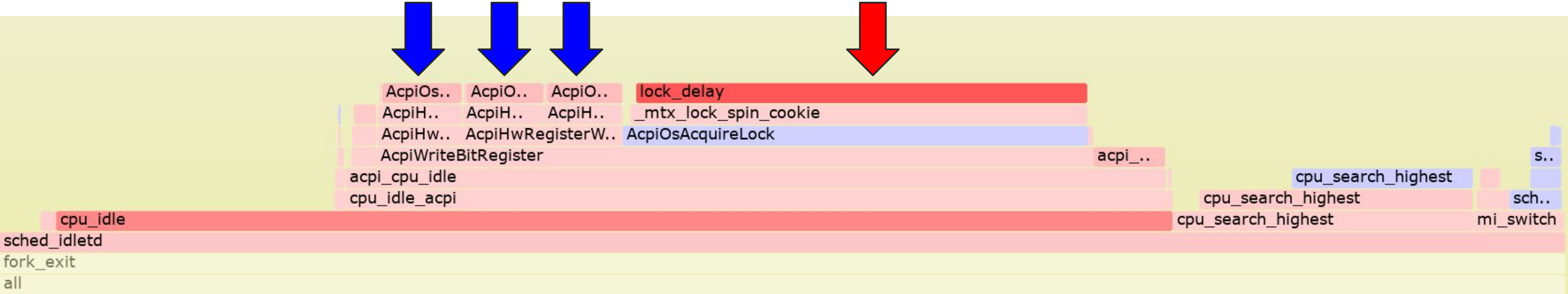
Baseline CPU profile (Read)



ZFS-backed iSCSI target optimization

Unexpected idle loop disaster

- Specific request rate combined with 80 logical CPU cores created 8% overhead and lock contention in ACPI CPU idle handler.
- Caused by appeared to be unneeded hardware registers accesses.
- Fixed on March 8, 2021 with
  455219675db “Change mwait_bm_avoidance use”,
  075e4807df3 “Do not read timer extra time when MWAIT is used”

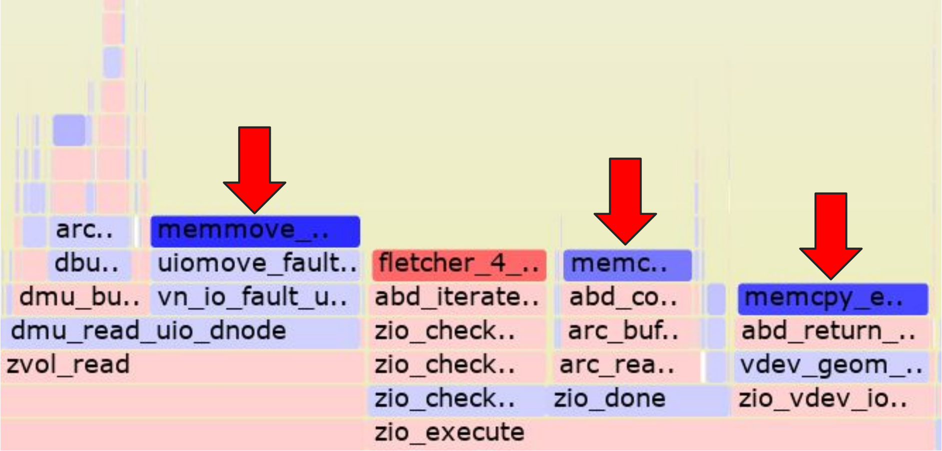


ZFS-backed iSCSI target optimization

Too many memory copies in ZFS

- ZFS copied data 3 times: I/O aggregation scatter/gather, ARC → DMU (may be a decompression instead), DMU → CTL.
- Copy for I/O aggregation scatter/gather removed on July 7, 2021 with eb5983e1b7b4 “Use unmapped I/O for scattered/gang ABD buffers”
- The trick only works for page-aligned I/O. Need proper scatter/gather in GEOM, CAM, drivers and hardware for general case.

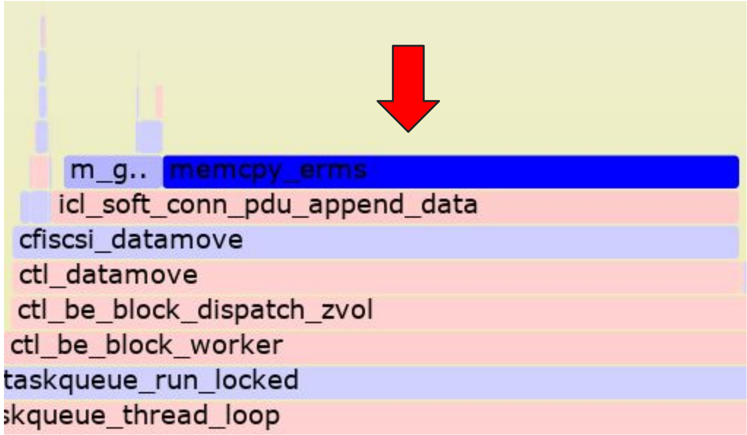
Thanks to Brian Atkinson for platform-independent part of gand ABD implementation.



ZFS-backed iSCSI target optimization

Memory copy in iSCSI target transmission path

- iSCSI target copied data from CTL buffer into mbuf chain for TCP send.
- Fixed on June 8, 2020 with
  9a4510ac322 “Implement zero-copy iSCSI target transmission/read.”
- Some old pre-busdma NIC drivers expect physically contiguous mbuf’s.
- Fixed data corruption by cxgb(4) driver in 9dc7c250b8.
- Few remaining broken 100Mbps NIC drivers are irrelevant for iSCSI and I hope they removed soon anyway.

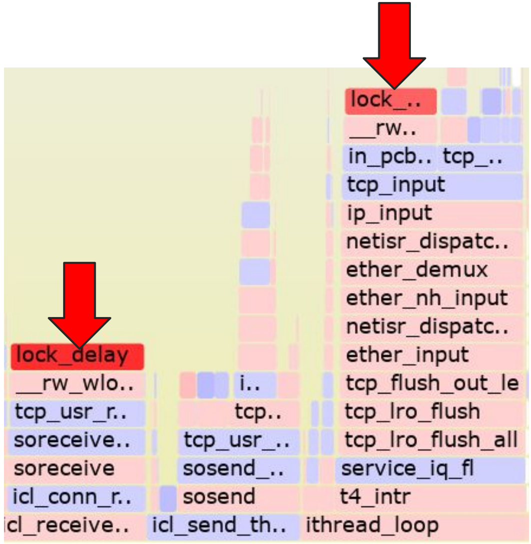


ZFS-backed iSCSI target optimization

TCP lock contention in iSCSI target

- iSCSI suffered from contention on TCP connection lock.
- Fixed in January-March 2021 with
  - b75168ed “Make software iSCSI more configurable”
  - 6895f89fe “Coalesce socket reads in software iSCSI”
  - b85a67f54 “Optimize TX coalescing by keeping pointer to last mbuf”

  , plus by large mbufs of “zero-copy iSCSI target transmission/read” change above.



MAXPHYS of 1MB

More efficient and predictable large I/O.

- ZFS defaults to 128KB blocks, but can be set up to 1MB (or more).
- ZFS aggregates consecutive I/O requests up to 1MB for HDDs.
- ZFS includes own I/O scheduler, controlling device queue depth.

⇩

- OS should not fragment I/O requests at least up to 1MB.

- Initial heavy lifting was done by Konstantin Belousov on November 28, 2020 with cd853791040.
- Later I improved and/or fixed large I/O handling in CAM, CTL, mpt(4), mrsas(4), nvme(4) and pms(4).

I/O QoS

The problem

- ZFS differentiates 9 types of I/O (priorities), that can be divided into 3 groups: synchronous (TBD ASAP), asynchronous interactive (within seconds) and asynchronous non-interactive (within minutes).
- ZFS I/O scheduler does not know disk specifics, so it had to be conservative, balancing between lower performance of short queues and high latency or even starvation of long ones.
- Disk's internal scheduler does not know about ZFS priorities and has to balance between good throughput and acceptable latency on average.
- I have found that some HDDs may delay random reads up to 4 seconds when they detect concurrent sequential read stream! It is not acceptable for many applications.

I/O QoS

Hardware way

Best solution would be to pass QoS information to the hardware:

- SATA priority – very simplistic, sometimes implemented, results vary.
- SCSI priority – poorly specified, not implemented except some SATL.
- NVMe priority – better specified, harder to use, less needed. ;)

I've really tried it on October-November 2020:
8836496815 "Introduce support of SCSI Command Priority"
06c888ecb9 "Add icc (Isochronous Command Completion) ccb_ataio field."

Unfortunately not very successfully due to limited hardware support.

Thanks to Muhammad Ahmad from Seagate. Hope to see better hardware. ;)

I/O QoS

Software way

- Ended up with some workarounds in kernel:
  0177b8871 “Enable bioq 'car limit' added at r335066 at 128 bios”

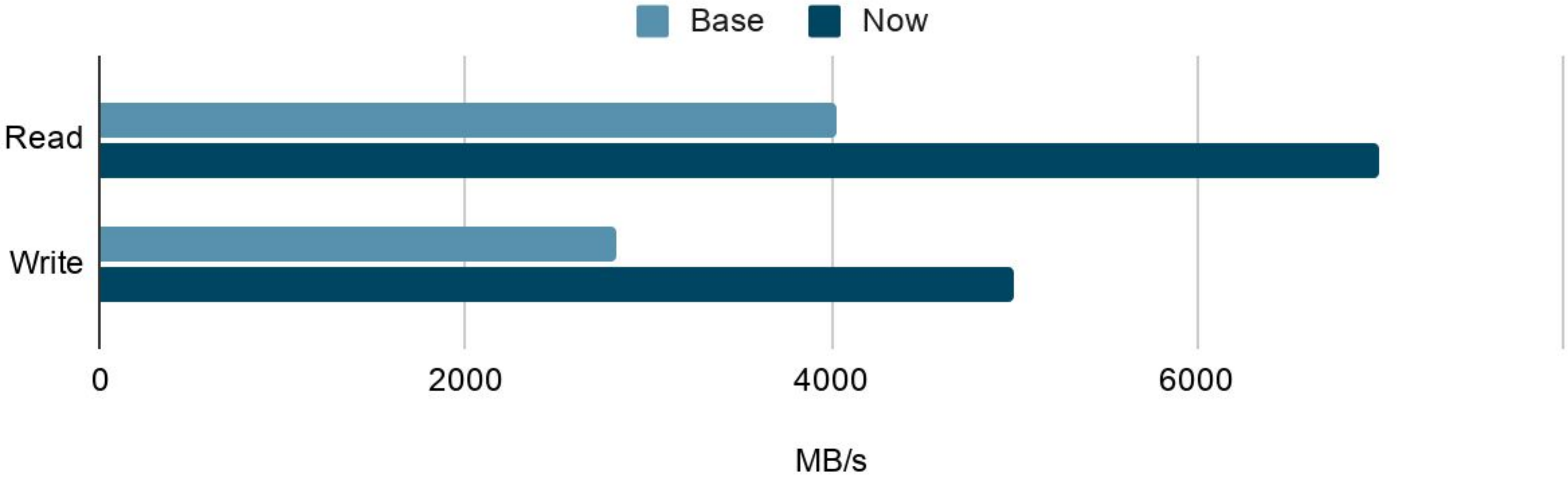
- and in ZFS:
  6f5aac3ca “Reduce latency effects of non-interactive I/O”
  891568c99 “Split dmu_zfetch() speculation and execution parts”
  7457b024b ”Scale worker threads and taskqs with number of CPUs”
  41d6eecd5 “Improve scrub maxinflight_bytes math”

- Now ZFS should be able to detect interactive workload starvation and throttle non-interactive one, dramatically reducing maximum latency.

ZFS-backed iSCSI target benchmarking

Repeated performance test

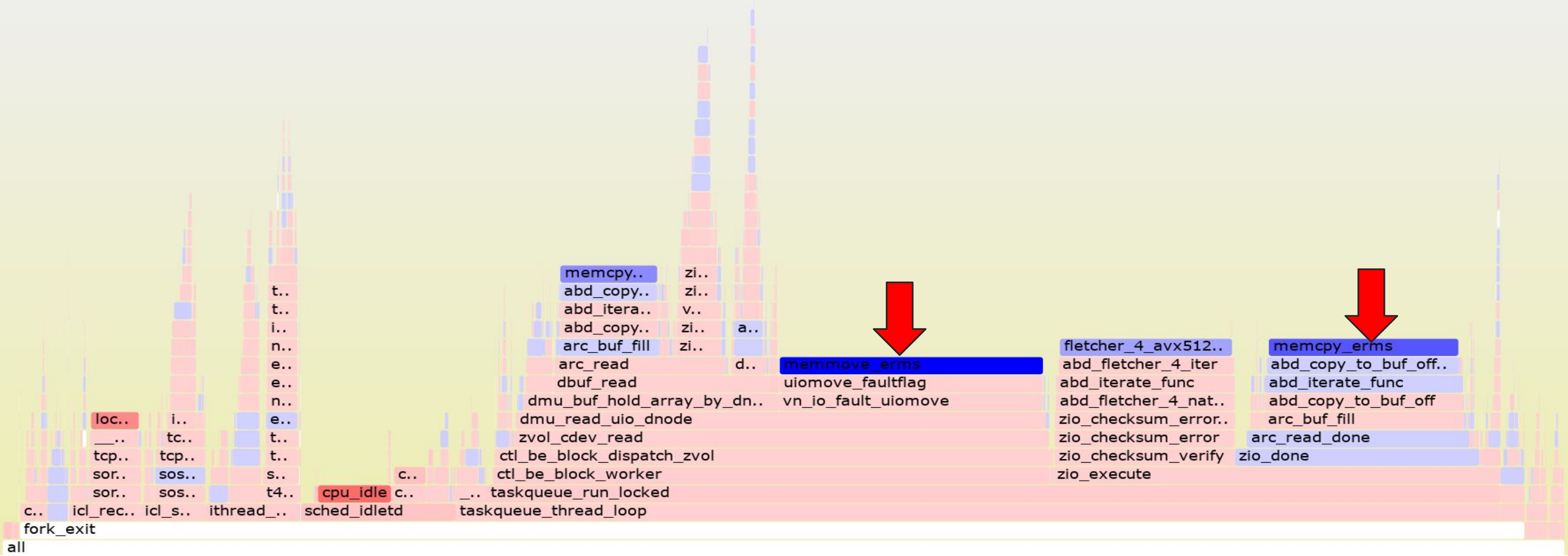
- Software: FreeBSD main from September 2021, OpenZFS 2.1, CTL.
- Now bottlenecked by single CPU on Windows initiator side in most tests.



ZFS-backed iSCSI target profiling

New CPU profile (Read)

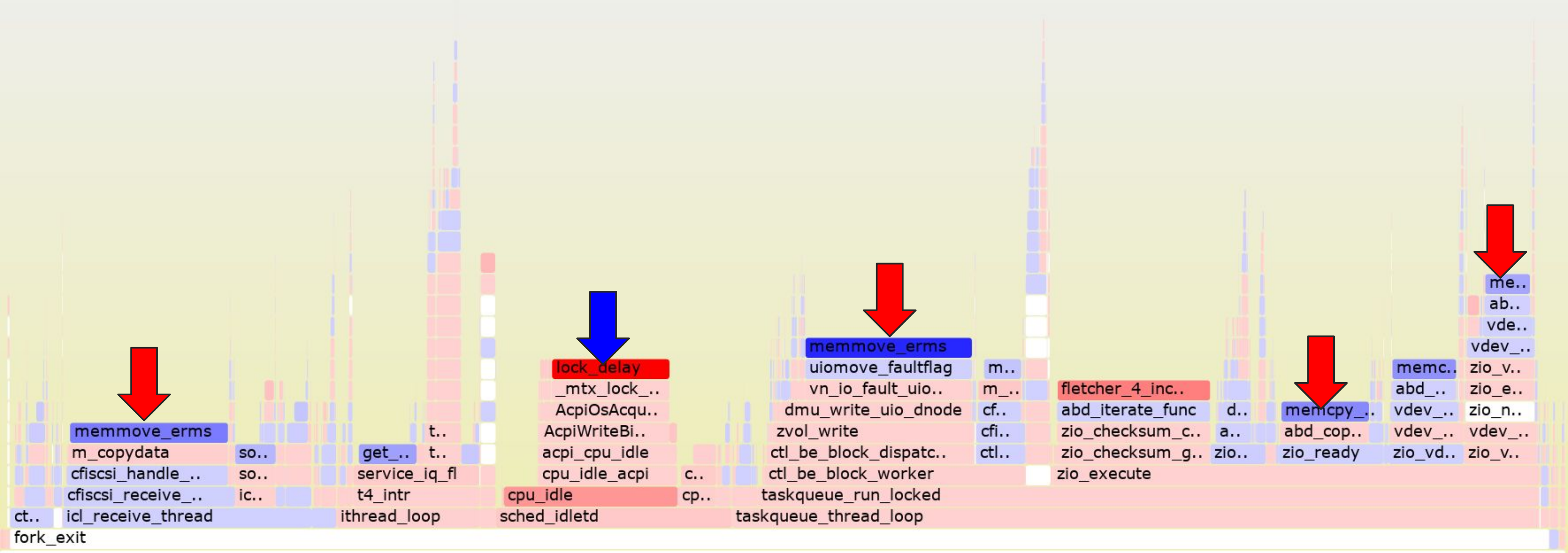
- Now only 2 data copies on read instead of 4! Can be 1 if we disable ABD ARC and compression, but that means KVA mapping and fragmentation.



ZFS-backed iSCSI target profiling

Baseline CPU profile (Write)

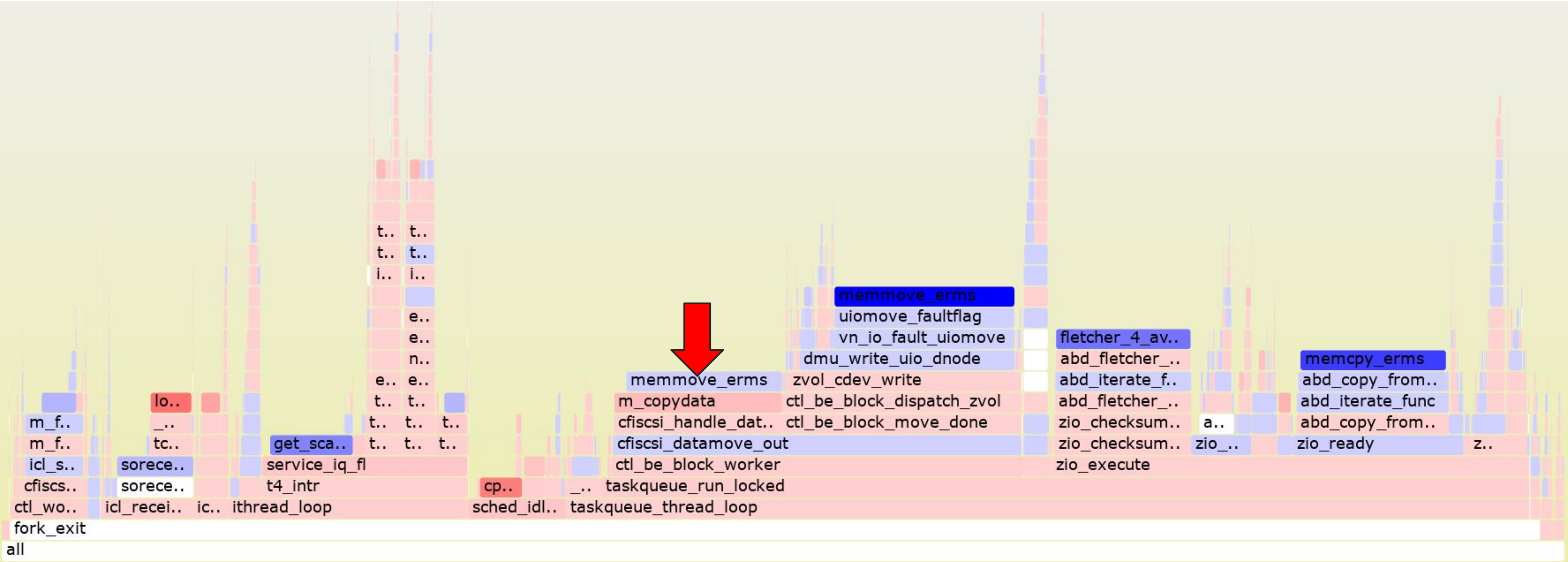
- Write was pretty much alike to read.



ZFS-backed iSCSI target profiling

New CPU profile (Write)

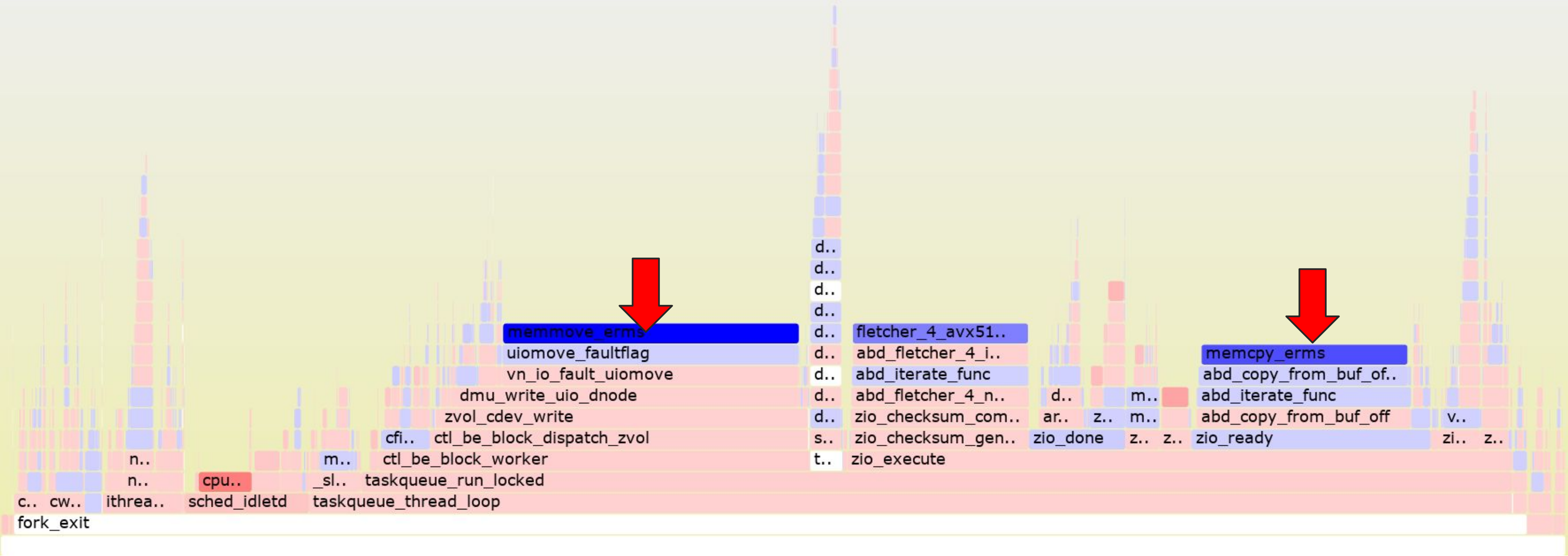
- After all the changes iSCSI receive still copies from TCP to CTL buffer.



ZFS-backed iSCSI target profiling

New CPU profile (Offloaded write)

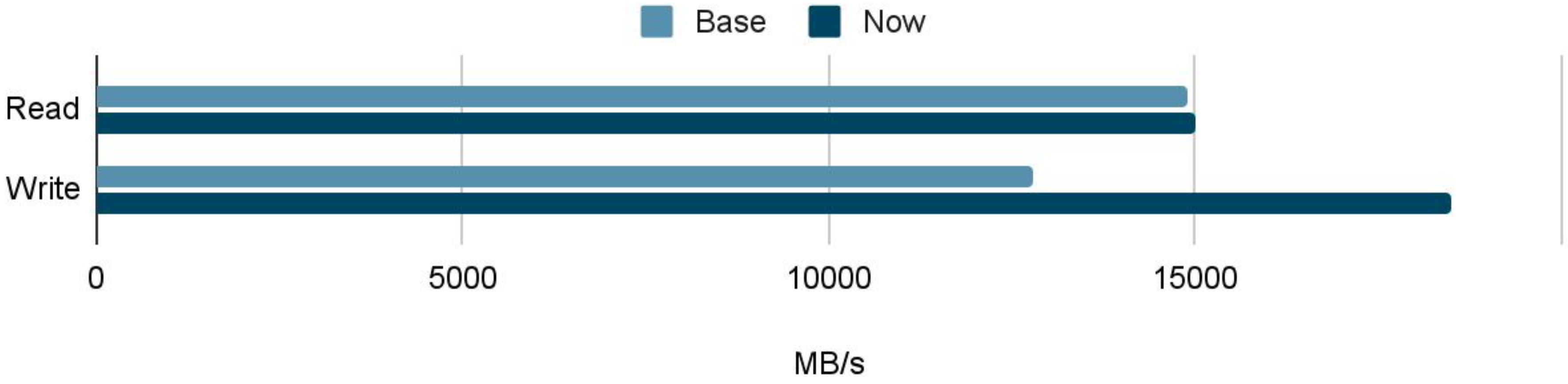
- Can be fixed by Chelsio iSCSI offload (cxgbei).
- Thanks to John Baldwin, now can be only 2 (or 1) copies on write too!



Raw ZFS throughout benchmarking

Repeated performance test

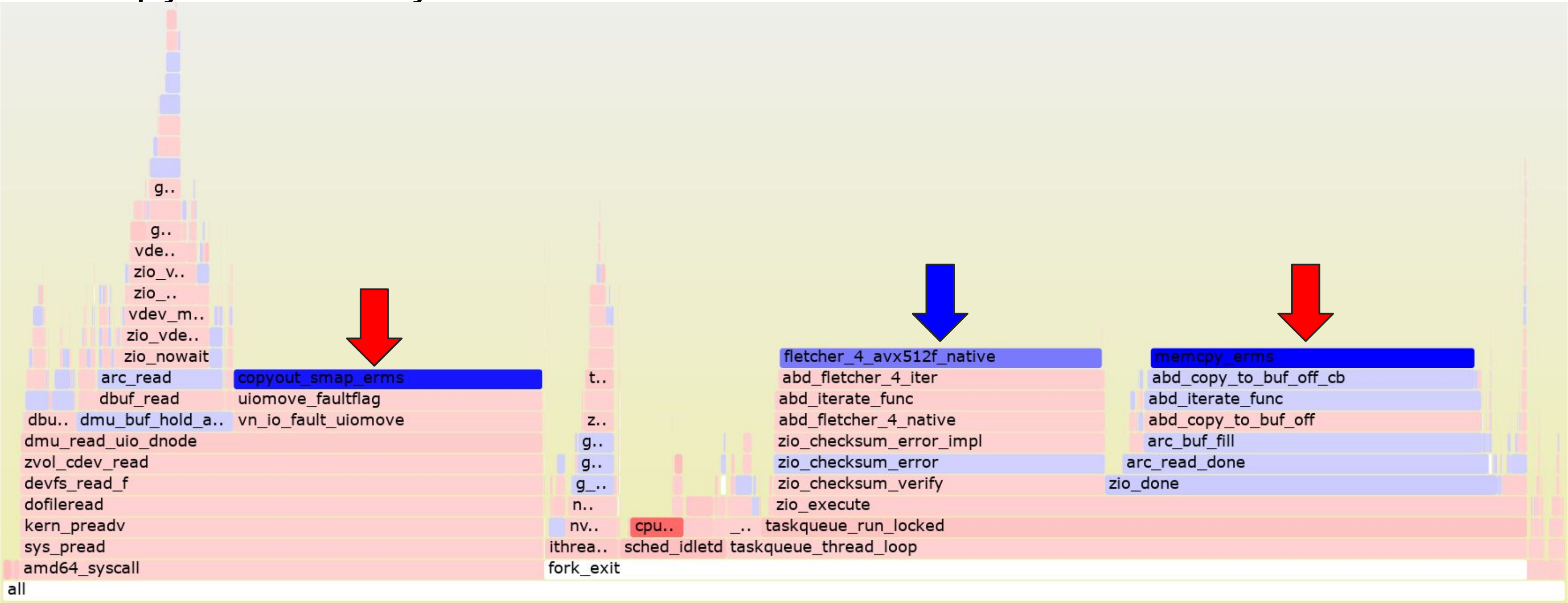
- Hardware: 2x Xeon Gold 6242R, 768GB RAM, 10 NVMe SSDs.
- Software: FreeBSD main from November 2021, OpenZFS master.
- Test: fio, sequential 1MB read/write over 12 128GB ZVOLs, Q1T12, ARC limited to metadata.
- Both read and write are now bottlenecked by the SSDs.



Raw ZFS throughput profiling

CPU profile (Read)

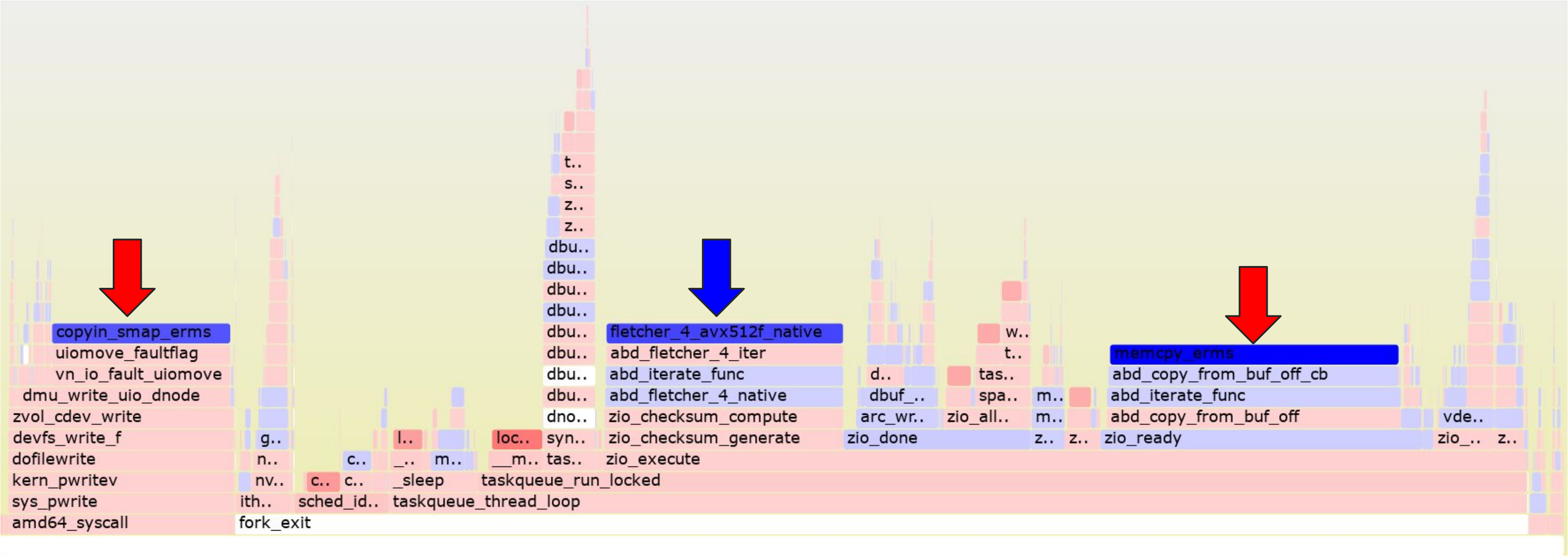
- Reading 15GB/s with only 13% CPU usage, out of which 40% by memory copy and 20% by checksums.



Raw ZFS throughput profiling

CPU profile (Write)

- Re-writing 18.5GB/s with only 35% CPU usage, out of which 30% by memory copy, 15% by checksums and 10% by lock contention.



ZFS CPU/IOPS optimizations

Not only throughput matters

- Aside of throughput ZFS was optimized for IOPS too:

86706441a86	Introduce write-mostly sums
c4c162c1e8f	Use wmsum for arc, abd, dbuf and zfetch statistics.
f8020c93635	Make metaslab class rotor and aliquot per-allocator.
29274c9f6d7	Optimize small random numbers generation
42afb12da70	Remove refcount from spa_config_*()
97752ba22a4	Move gethrtime() calls out of vdev queue lock
f7de776da2e	Fix ARC ghost states eviction accounting
c1b5869bab9	Introduce dsl_dir_diduse_transfer_space()
1b50749ce97	Optimize allocation throttling
7eebcd2be6a	Avoid small buffer copying on write
7f9d9e6f39f	Avoid vq_lock drop in vdev_queue_aggregate()
6b88b4b501a	Remove b_pabd/b_rabd allocation from arc_hdr_alloc()

FreeBSD CPU scheduler optimizations

IOPS also depend on CPU scheduler

- ZFS uses several context switches per I/O and even per block. So it got attention:

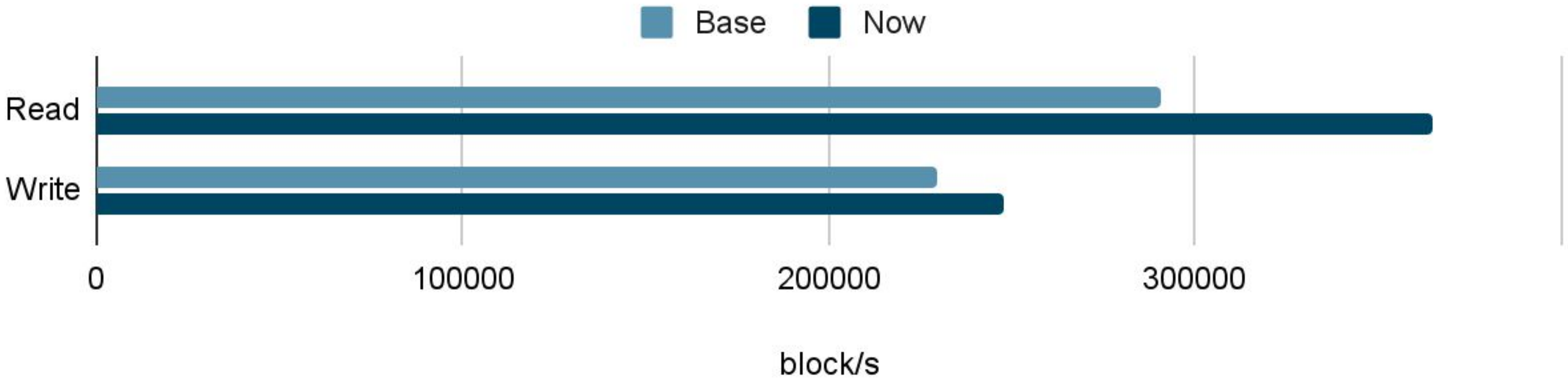
f91aa773b	Add wakeup_any(), cheaper wakeup_one() for taskqueue
c9205e35	Fix/improve interrupt threads scheduling
6df35af4d	Allow sleepq_signal() to drop the lock
aefe0a8c3	Refactor/optimize cpu_search_*()
2668bb2a	sched_ule(4): Reduce duplicate search for load
8bb173fb5	sched_ule(4): Use trylock when stealing load
e745d729b	sched_ule(4): Improve long-term load balancer
ef50d5fbc	x86: Add NUMA nodes into CPU topology

...

Raw ZFS IOPS benchmarking

Repeated performance test

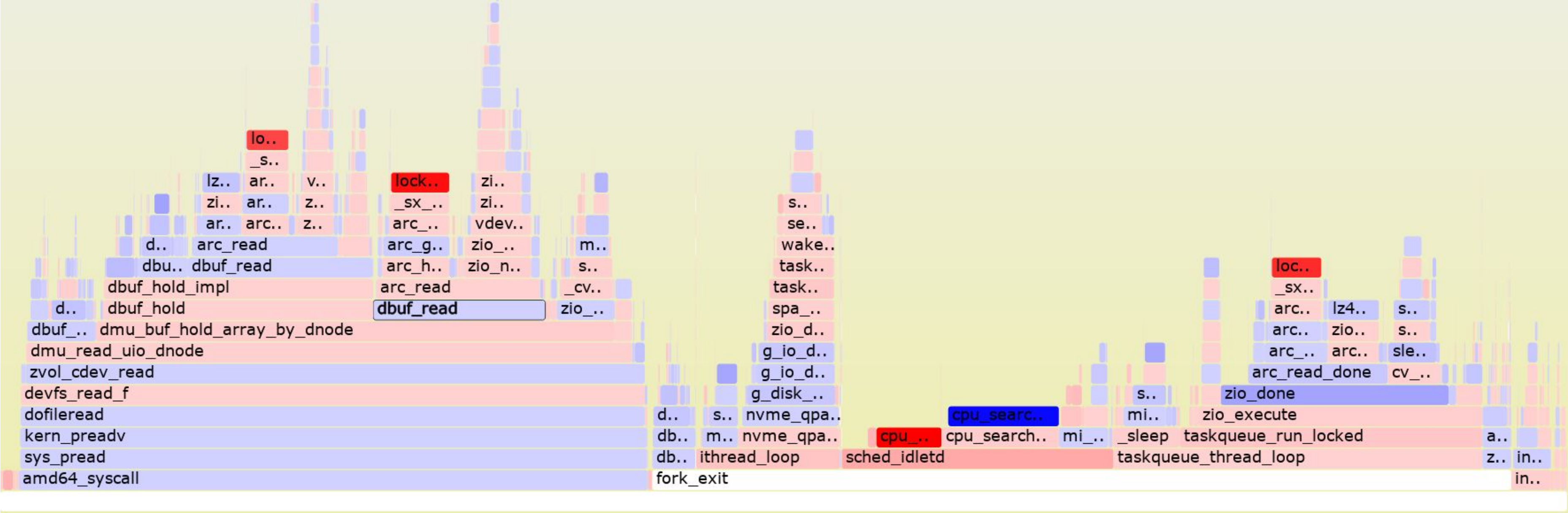
- Hardware: 2x Xeon Gold 6242R, 768GB RAM, 10 NVMe SSDs.
- Software: FreeBSD main from November 2021, OpenZFS master.
- Test: fio, sequential 4KB read/write over 12 128GB ZVOLs, Q4T12, ARC limited to metadata.



Raw ZFS IOPS profiling

CPU profile (Read)

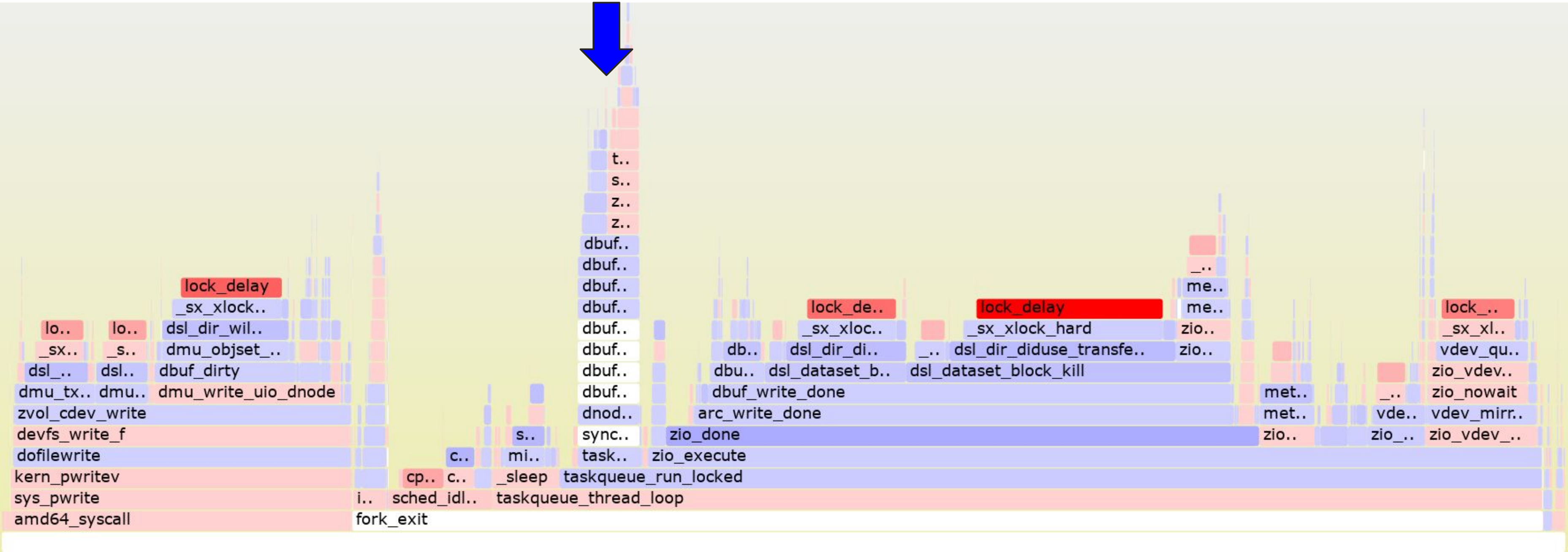
- Reading 365K blocks/s with 35% CPU usage, out of which 12% by lock contention (primarily ARC eviction) and 7% by CPU scheduler.



Raw ZFS IOPS profiling

CPU profile (Write)

- Re-writing 248K blocks/s with 30% CPU usage, out of which 43% is lock contention (mostly ZFS dsl_dir) and 4% bottleneck in ZFS sync thread.



PCIe hot-plug

Not all work is about performance

- Need PCIe hot-plug to use NVMe in enterprise environment.
- Should not crash on PCIe error:
  855e49f3b “Add initial driver for ACPI Platform Error Interfaces.”
- Should receive hot-plug events and configure devices:
  4cee4598e “Add mostly dummy hw.pci.enable_aspm tunable.”
  5a898b2b7 “Set PCIe device's Max_Payload_Size to match PCIe...”
  15cb3b540 “pcib(4): Write window registers after resource adjust...”
- Still have big problems if BIOS-reserved resources are insufficient. Need resource relocation.
- Fixed vmd(4) assumes resource reservation by BIOS:
  7af4475a6 “vmd(4): Major driver refactoring”
  , but causes interrupt sharing and some other problems.

Random driver fixes

TrueNAS communify is full of surprises

- Fixed number of issues in SAS disk detection and hot-plug:
  b99419aee "mpr/mps(4): Make device mapping some more robust."
  e3c5965c2 "mpr(4): Handle mprsas_alloc_tm() errors on device re"
  9781c28c6 "mpr(4): Fix unmatched devq release."
  84d5b6bd6 "cam(4): Fix quick unplug/replug for SCSI."
  02d819401 "mps/mpr(4): Move xpt_register_async() out of lock."
- Fixed other random driver issues:
  e8144a13e "ciss(4): Properly handle data underrun."
  6c2d4404 "ipmi(4): Limit maximum watchdog pre-timeout interval."
  8434a65c "pms(4): Do not return CAM_REQ_CMP on errors."
- Major isp(4) driver cleanup.

... and more ...

TrueNAS

iXsystems™

THANK YOU

We are hiring!

🌐 www.iXsystems.com | 🌐 www.TrueNAS.com | ✉ mav@ixsystems.com