

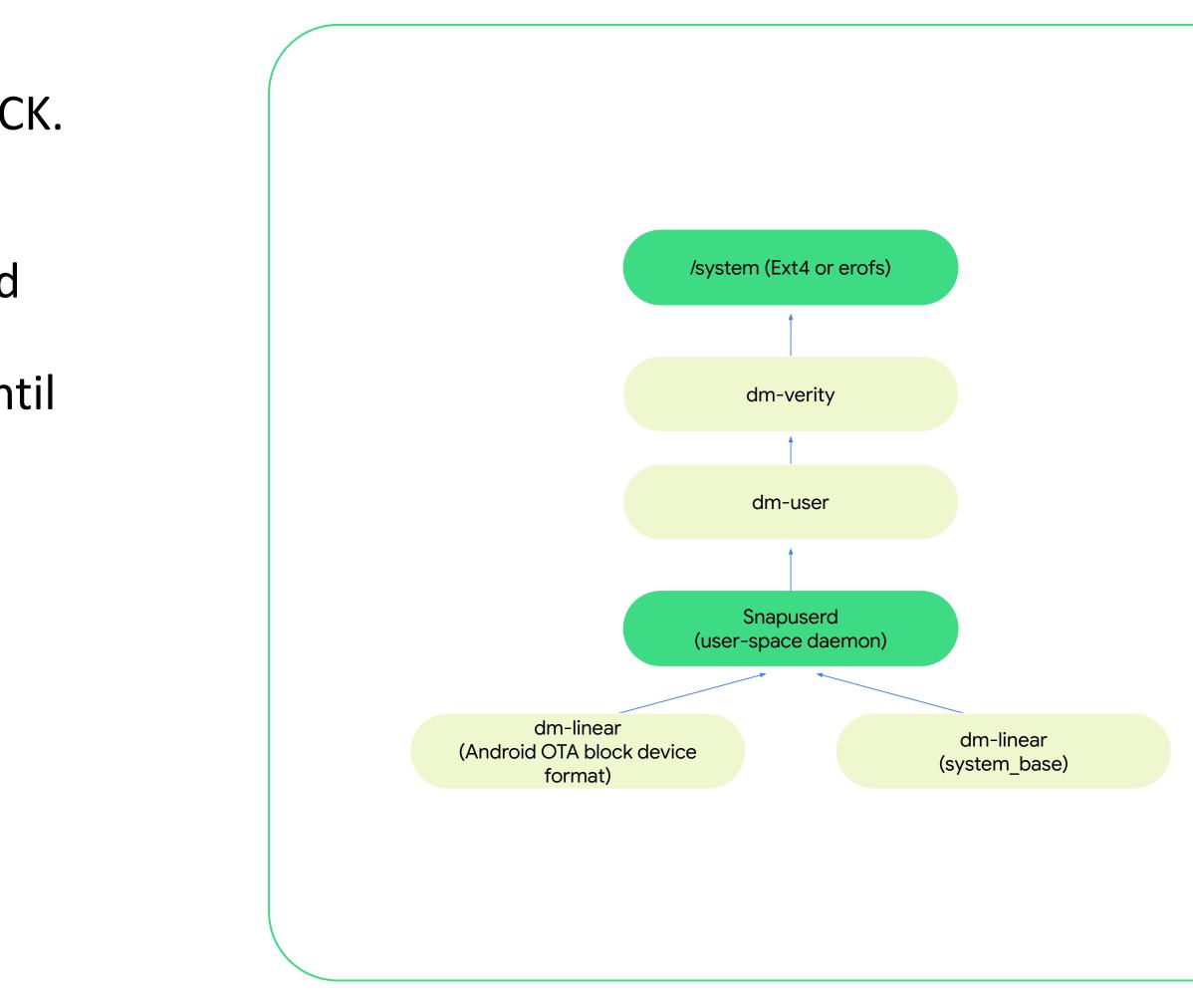
Linux Plumbers Conference

Vienna, Austria | September 18-20, 2024

ublk based zero copy I/O - use case in Android Akilesh Kailash (akailash@google.com) Vienna, Austria LINUX PLUMBERS CONFERENCE Sept. 18-20, 2024

Android OTA - Storage Stack - Overview

- Userspace snapshots
 - dm-user: Out of tree kernel driver in ACK. Routes I/O request from verity to userspace daemon
 - Snapuserd daemon: Snapshot logic and snapshot merge
 - Root partition mounted off dm-user until snapshot merge is completed



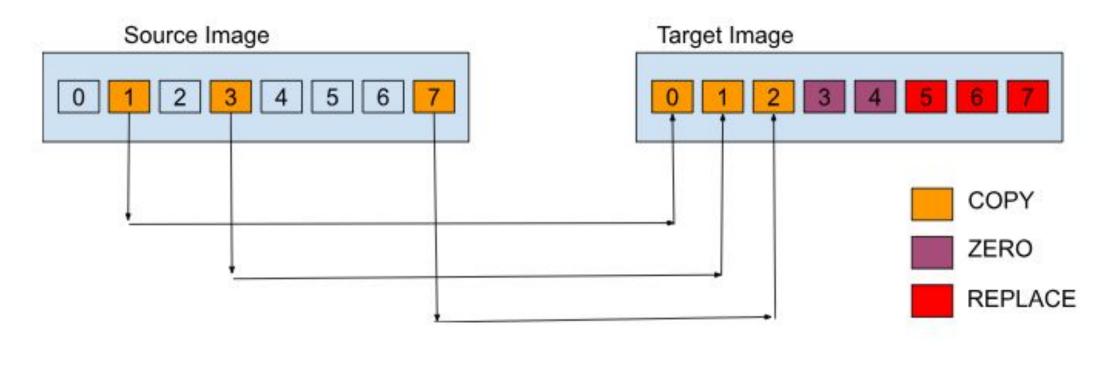
LINUX PLUMBERS CONFERENCE



Android OTA Format

- Encodes three block-level operations:
 - ZERO: The destination block is zeroed.
 - COPY: The destination block is copied from a pre-existing block.
 - **REPLACE**: The destination block is replaced with new data. gz / lz4 compressed or could be uncompressed
 - XOR: XOR the destination block with source block.
- The metadata operations of the OTA format is stored in a block device.
 - Will be used by userspace daemon when snapshots are constructed

LINUX PLUMBERS CONFERENCE





I/O Path for COPY operation - 4k block size

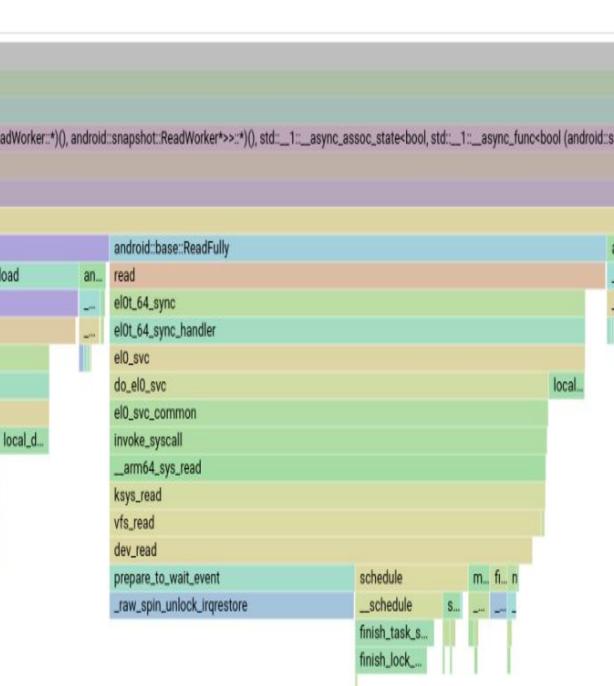
γ (metric: cpu-cycles +) Add pivot or filter...

16,699,772,992 cpu-cycles (100.00%) __start_thread _pthread_start void* std::_1::_thread_proxy[abi:nn190000]<std::_1::_async_assoc_state<bool, std::_1::_async_func<bool (android::snapshot::ReadWorker:*)(), android::snapshot::ReadWorker:*)(), std::_1::_async_assoc_state<bool, std::_1::_async_func<bool (android::snapshot::ReadWorker:*)(), android::snapshot::ReadWorker:*)(), std::_1::_async_assoc_state<bool, std::_1::_async_func<bool (android::snapshot::ReadWorker:*)(), android::snapshot::ReadWorker:*)(), std::_1::_async_assoc_state<bool, std::_1::_async_func<bool (android::snapshot::ReadWorker:*)(), android::snapshot::ReadWorker:*)(), std::_1::_async_assoc_state<bool, std::_1::_async_func<bool (android::snapshot::ReadWorker:*)(), android::snapshot::ReadWorker:*)(), android::snapshot::ReadWorker:*)(), std::_1::_async_assoc_state<bool, std::_1::_async_func<bool (android::snapshot::ReadWorker:*)(), android::snapshot::ReadWorker:*)(), android::snapshot std::_1::_async_assoc_state<bool, std::_1::_async_func<bool (android::snapshot::ReadAhead::*)(), android::snapshot::ReadAhead*>>::_execute ReadWorker::Run DmUserBlockServer::ProcessRequests 16,683,467,385 cpu-cycles (99.90%) ReadWorker::ReadAlignedSector ReadWorker::ReadDataFromBaseDevice DmUserBlockServer::WriteDmUserPayload android::base::ReadFullyAtOffset _memcpy_... android::base::WriteFully pread el0t_64_sync el0t_64_sync el0t_64_sync_handler el0t_64_sync_handler el0_svc el0_svc do_el0_svc do_el0_svc el0_svc_common el0_svc_common invoke_syscall invoke_syscall _arm64_sys_pread64 _arm64_sys_write vfs_read ksys_write blkdev_read_iter vfs_write blkdev_direct_IO dev_write bio_endio submit_bio_wait wait_for_co... submit_bio IOV clone_endio wait_for_co... __dm_io_complete submit_bio_noacct submit_bio_noacct_nocheck _raw_... io_s... bio_endio submit_bio_wait_. _submit_bio __blk_flush_plug blk_mg_submit_bio complete dm_sub blk_mq_flush_plug_list _blk_mg_alloc... blk_mg_... _raw_spin_unlock_ _m... bfq_insert_requests bfq_li... k... b... bfq_bio. blk_mq_run_hw_queue bfqg_stat... _raw_s... blk_mq_sched_dispatch_requests _ra... _r... b... _blk_mq_sched_dispatch_requests percpu_c... blk_mg_dispatch_rg_list scsi_queue_rq _ra... ufshcd_queuecommand ufshcd_send_command ufshcd.. __traceiter_android_vh_... _raw_sp... ktim... _raw_...

pixel_ufs_send_comma

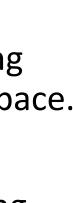
__store_cmd_log read

LINUX PLUMBERS CONFERENCE



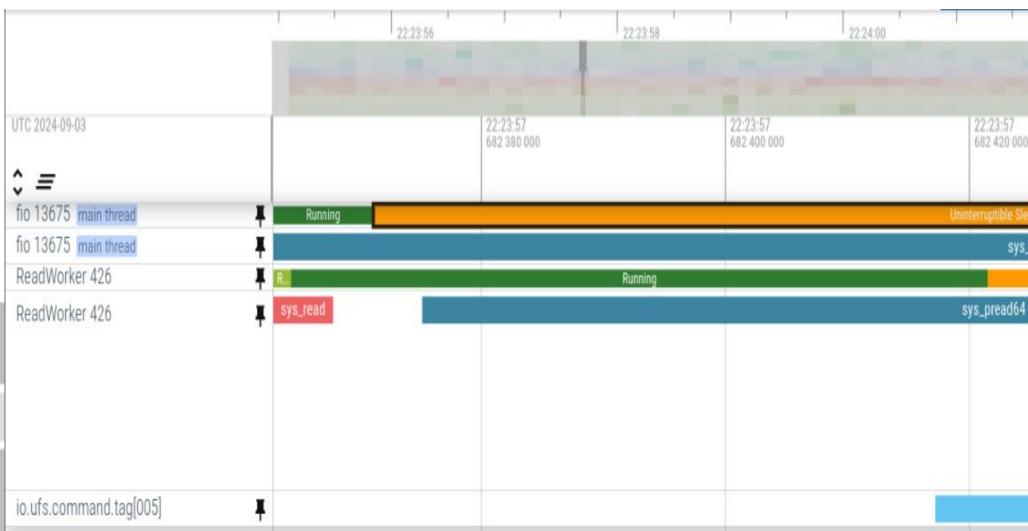
• COPY operation overhead in the I/O path

- Data moves back and forth between kernel and userspace
- ~50% of CPU cycles spent in reading data from source image into userspace.
- ~15% of CPU cycles spent in sending the data back to kernel from the userspace daemon
- Impacts OTA boot time and application I/O performance until snapshot merge is complete





Perfetto - I/O Path for COPY operation - a 4K I/O request



- ~100us time spent by application (fio) waiting for one 4k I/O to complete
- Userspace daemon (ReadWorker) is again blocked reading from underlying block device
- The actual I/O on underlying UFS is ~23us

r

LINUX PLUMBERS CONFERENCE

22:24.02	22:24:04	22:24:06		22:24:08	1 1
	22:23:57 682 440 000	22:23:57 682 460 000		22:23:57 682 480 000	
					542-55-
ep (IO)			Runn	able	Running
_pread64					sys_pread64
Uninterruptible Sleep (10)		Runnable	Running		Ru
			sys_write	sys_read	
READ (10)					



ublk - userspace block device - zero copy

- V5 version of the patch evaluated on Pixel 6 running Android Mainline
 - Work in progress to move storage stack towards ublk replacing dm-user.
 - Zero copy effort helps in cutting down unnecessary data movement between userspace and kernel.
 - Primarily improving Android boot time and application performance until snapshot merge is complete.

LINUX PLUMBERS CONFERENCE

• Evaluate ongoing upstream patch - ublk zero copy based on io_uring providing sqe group buffer from Ming Lei https://lore.kernel.org/io-uring/80d4150e-a4fe-4c05-be23-4ceebd40d7fd@gmail.com/T/#md440a9cebf9a1ed8e5cc204e6dcfdaa2f898e7a4

• ublk zero copy partially addresses some of the I/O path overhead - primarily for COPY operations in the OTA format

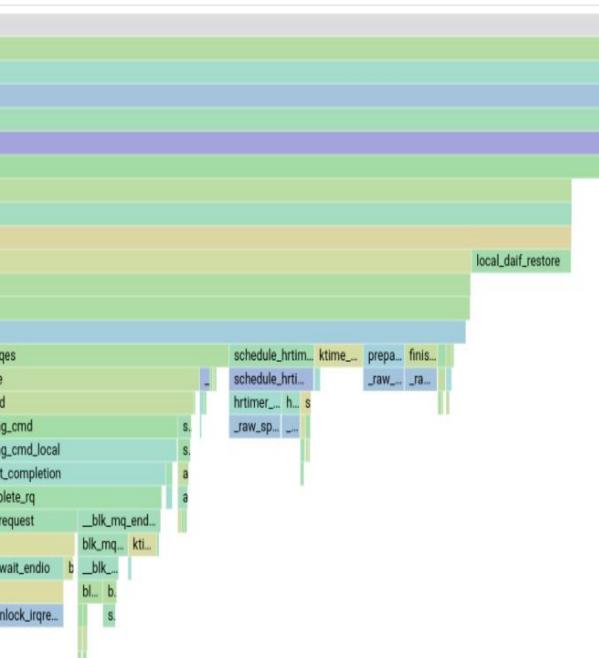
• Goal is to cut down the existing I/O path overhead for both COPY and REPLACE operations which accounts to 75% of operations



I/O Path for COPY operation - 4k block size

Y metric: cpu-cycles - Add pivot or filter...

13,618,592,702 cpu-cycles (100.00%)		
start_thread		
pthread_start		
ublksrv_io_handler_fn		
ublksrv_process_io		
io_uring_submit_and_wait_timeout		
_io_uring_get_cqe		
el0t_64_sync		
el0t_64_sync_handler		
el0_svc		
do_el0_svc		
el0_svc_common		
invoke_syscall		
_arm64_sys_io_uring_enter		
io_run_task_work	io_su	ubmit_sqes
task_work_run		sue_sqe
tctx_task_work		ring_cmd
tctx_task_work_run		_ch_uring_c
io_handle_tw_list		_ch_uring_c
io_req_task_submit		_commit_co
lo_issue_sqe		lk_complet
io_read		update_requ
io_read		endio
blkdev_read_iter	subr	nit_bio_wai
blkdev_direct_IO	to	plete
submit_bio	b ati	_spin_unlo
submit_bio_noacct	b kt	
submit_bio_noacct_nocheck		
submit_bio	kti	
_blk_flush_plug	blk_mq_submit_blo dm_sub	
blk_mq_flush_plug_list	blk_mq_alloc_req blk_mq_s a	
blk_mq_run_hw_queue	bfq_insert_requests bfq_lim kti bl bfq_bio d l	
blk_mq_sched_dispatch_requests	bfqg_statsraw_sprabfrara t	
blk_mq_sched_dispatch_requests t	t percpu_cou io s	
blk_mq_dispatch_rq_list bfq	_r k	
scsi_queue_rqra		
ufshcd_queuecommand		
ufshcd_send_command ufshcd_h		
traceiter_android_vh_ufs_sendraw_spin ktime_get _raw_sp		
pixel_ufs_send_command		
store_cmd_log		
readl kt		
		-
	LINUX PLUMBERS C	1

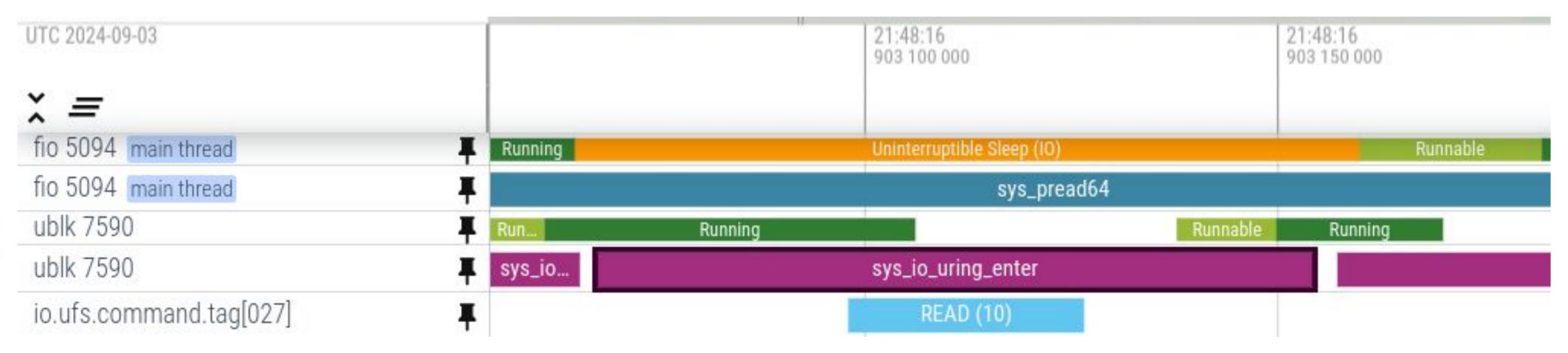


- Data path is simple
- Two SQE entries Lead SQE provides kbuf through io_import_group_kbuf
- One syscall completing entire I/O (io_uring_submit_and_wait)

ONFERENCE Vienna, Austria Sept. 18-20, 2024



Perfetto - I/O Path for COPY operation - a 4K I/O request



- No more additional I/O path overhead in the daemon
- Reduce CPU contention by more than 50%
- Android boot time

LINUX PLUMBERS CONFERENCE

Caveats / Challenges

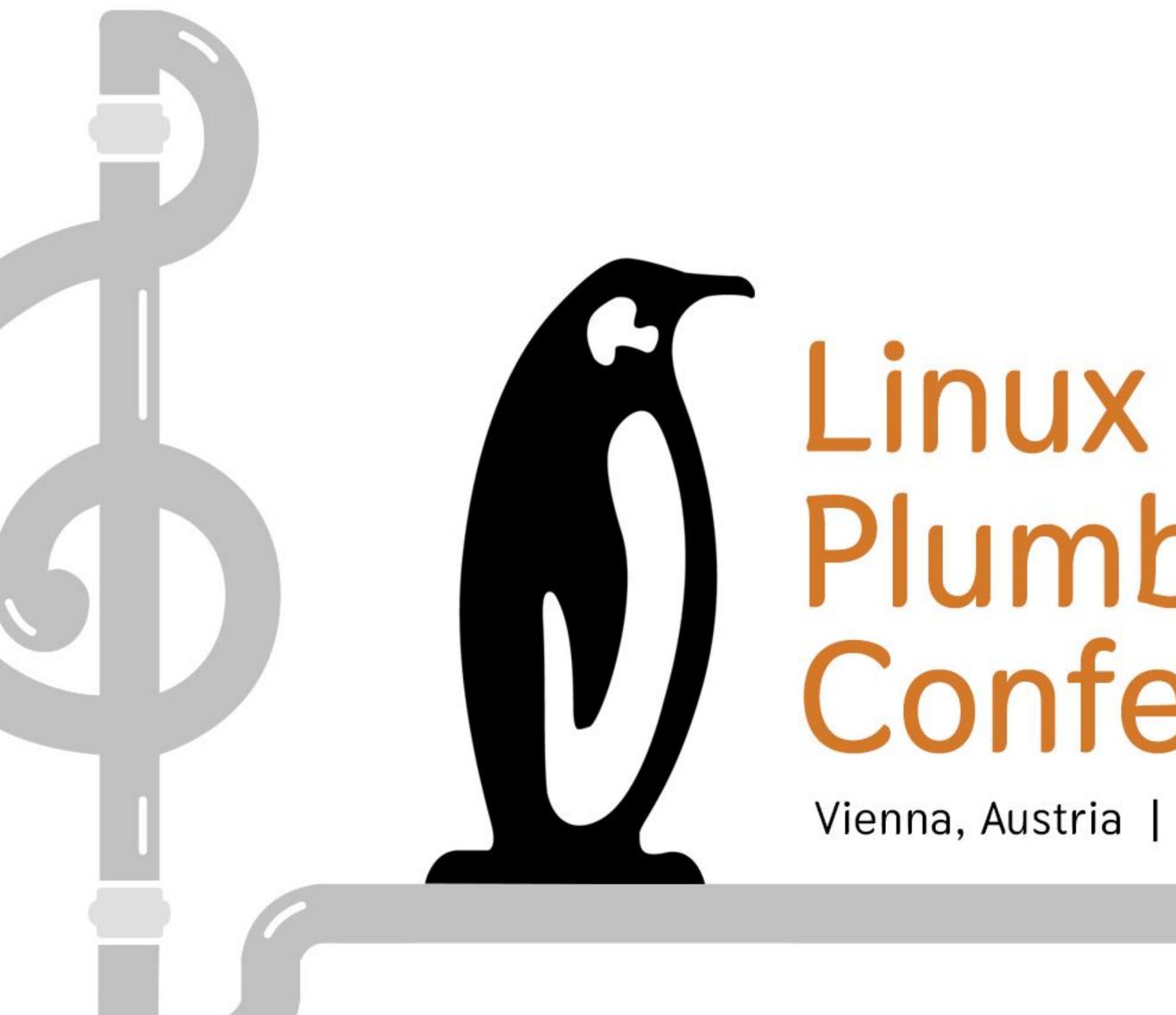


- How to handle REPLACE and ZERO operations through zero copy ?
- REPLACE operations are blocks which are compressed (Iz4 or zstd) and stored in block device.
- The I/O path is similar to COPY operation but the data is retrieved from compressed block device.
- Similar I/O path overhead as observed on COPY operations.
- - This will significantly help Full OTA which primarily consists of all REPLACE blocks.
 - Will further reduce the CPU contention post boot.
- Thoughts / Questions ?

• Currently, data is transferred to userspace, de-compressed in userspace and then data is transferred back to kernel.

• ublk zero copy would help but the data has to be decompressed in the kernel after importing kbuf from io_import_group_kbuf.





Linux Plumbers Conference

Vienna, Austria | September 18-20, 2024
