Linux Plumbers Conference | Richmond, VA | Nov. 13-15, 2023

Large Block Sizes in Linux

SAMSUNG
GOST
Global Open Source Team

Luis Chamberlain
Pankaj Raghav
Daniel Gomez

Agenda:

- Introduction to Large Block Sizes (LBS)
- Use-cases for LBS in Linux
  - Enables:
    - existing storage device support
    - enhancing new storage device support
- Plumbing & Implementation
- Testing

Introduction

LBS in a nutshell

bs > ps

block size > page size

Reviving a 16 year old effort:

- 2007: Christoph Lamenter posted Large Block Size support
  - Only page cache changes
  - Added more complexity to the core VM subsystem.
  - Missed an equivalent buffer-head solution
- 2007 & 2009: Nick Piggin posted fsblock & fsblock v2
  - Alternative to buffer heads. Did not get much traction.
    - `rm fs/buffer.c`
    - Not the way we do development

Reviving a 16 year old effort:

- 2007: Christoph Lamenter posted Large Block Size support
  - Only page cache changes
  - Added more complexity to the core VM subsystem.
  - Missed an equivalent buffer-head solution
- 2007 & 2009: Nick Piggin posted fsblock & fsblock v2
  - Alternative to buffer heads. Did not get much traction.
    - `rm fs/buffer.c`
    - Not the way we do development
- 2013: NFS block layout exports → block ranges for multipage writes → multipage buffered writes → replacement for buffer-heads: iomap
- 2017 - 2021: Matthew Wilcox with Folios → merged v4.20
  - xarray and multi-index support !
- 2018: Dave Chinner xfs: Block size > PAGE_SIZE support 5 years ago
  - Halted due to the ongoing folio work

Common LBS restrictions

- Main **common** limitation was the tight coupling of **system page size** in the **Page Cache**

LBS in Block device context:

- Block layer can handle **larger IOs**. **Minimum** guaranteed **IO** size should be **logical block size**.
- LBS: addressing support for
  - logical block size > ps
  - physical block size > ps

LBS in NVMe logical block size example:

Example of existing max LBA format size limitation on NVMe block driver

LBA format in NVMe sets logical block size

LBS support enables future LBA formats > ps

If LBA format is 16k → logical block size → 16k

Will set the capacity to 0 today effectively disabling these devices. If you lift this it crashes.

```
#drivers/nvme/host/core.c
    /*
     * The block layer can't support LBA sizes
     * larger than the page size yet, so catch
     * this early and don't allow block I/O.
     */
    if (ns->lba_shift > PAGE_SHIFT) {
        capacity = 0;
        bs = (1 << 9);
    }
```

LBS in NVMe logical block size example:

Example of existing max LBA format size limitation on NVMe block driver

LBA format in NVMe sets logical block size

LBS support enables future LBA formats > ps

If LBA format is 16k → logical block size → 16k

Will set the capacity to 0 today effectively disabling these devices. If you lift this it crashes.

```
#drivers/nvme/host/core.c
    /*
     * The block layer can't support LBA sizes
     * larger than the page size yet, so catch
     * this early and don't allow block I/O.
     */
    if (ns->lba_shift > PAGE_SHIFT) {
        capacity = 0;
        bs = (1 << 9);
    }
```

But bumping LBA format is radical
Maybe we don't want that ... more on this later

LBS in File System context:

- Block size: minimum data block allocation unit in a filesystem.
- All filesystems in Linux only support **bs <= ps**
- xfs example case for LBS
- other filesystems
  - TBD

Without LBS

- You can **create** filesystems with bs > ps
- Cannot mount bs > ps

```
$ mkfs.xfs -f -b size=64k -s size=4k /dev/nvme0n1
$ mount -t xfs /dev/nvme0n1 /mnt #Error!

#dmesg
XFS (nvme0n1): File system with blocksize 16384
bytes. Only pagesize (4096) or less will currently
work.
```

LBS use case types

- Works on all existing storage devices and block drives
  - HDDs, SATA SSDs, scsi,etc
    - LBA formats:
      - 512 byte
      - 4k
- Enhance new technology and new storage device experience

- LBS proof of concepts:
  - qemu with LBA format > 4k
  - qemu with NVMe hacks

Existing device use case: testing, forensics, recovery:

- Systems with PAGE_SIZE > 4k are not easily available to many developers
  - Test filesystem bugs with larger block sizes on x86
  - Extract files with larger block sizes on x86
    - Example: a poor sole waiting 6 years for a resolution (post on serverfault)

May 20, 2011
Patriot PCNASJV35S4 Diskless System
Javelin S4 4-Bay Media Server

- 800MHz AMCC PowerPC processor
  - PPC 440 supported different page sizes:
    - 1KB, 4KB, 16KB, 64KB, 256KB, 1MB, 16MB and 256MB
- 4 SATA HDDs
- 256MB RAM
- xfs with 64k block size

Existing device use case: writes are typically large

- Under some workloads you may only want to deal with files >= 16k
- Large folios are used today with or without LBS:
  - readahead
  - iomap write path
    - However LBS will ensure no small writes for inodes for data are ever issued

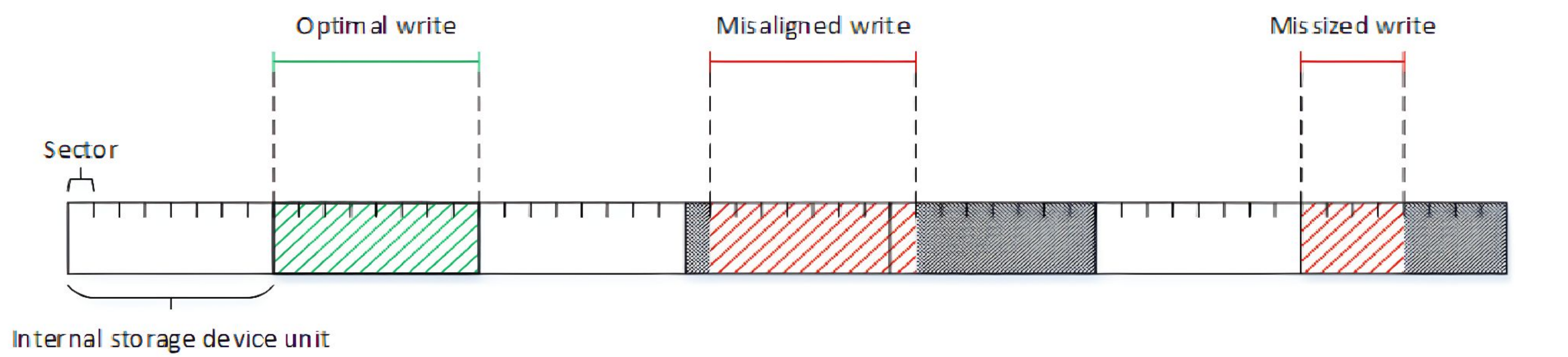
Lessons from databases:

- Databases already work on **bigger internal page sizes**
  - **MySQL default page size has been 16KB for InnoDB for a long time**
- Databases would prefer all or nothing transaction (**no torn writes**)
- Most databases uses **Direct IO** to **circumvent** the **torn writes issue**.
  - Hyperscalers have innovated with large atomics for this reason
- Some databases only have **Buffered IO** support - PostgreSQL
  - Jonathan Katz: "*Direct I/O is a long-term feature in the works. It will take years to implement. It's a complex problem.*"

    November 9, 2023 - Open Source Summit Spain
- **LBS support** enables databases to use large filesystem block sizes with **buffered IO**
  - No new device support is required for buffered-IO LBS support
  - However new devices with larger atomics would be nice

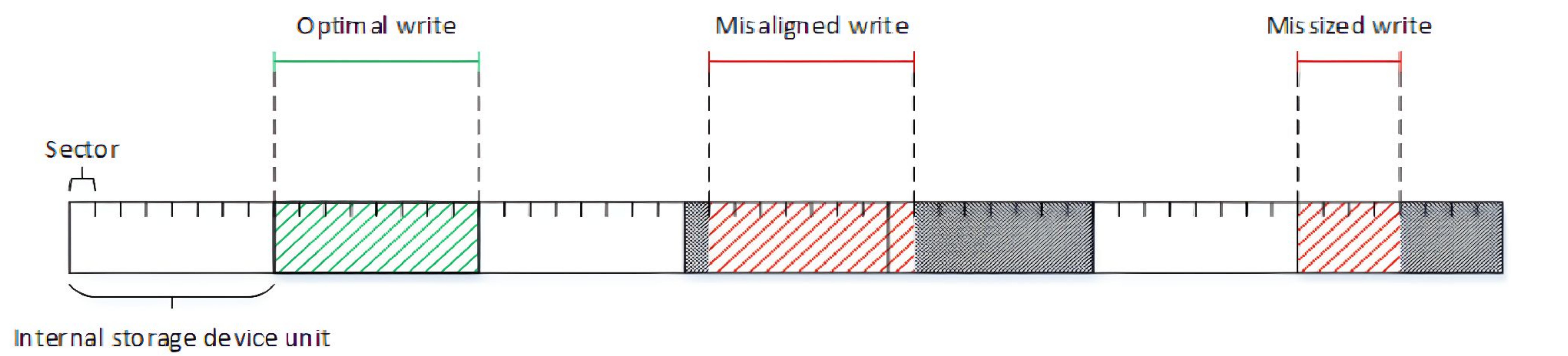
New device use case: High Capacity SSDs:

- **Indirection Unit** provides internal **logical to physical mapping of LBAs** in an **SSD**.
- Most SSDs available in the market have **4k IUs**.
- **High capacity SSDs are** using larger **IU** to increase capacity and reduce DRAM costs
  - Writes aligned to IU will provide **best performance**



New device use case: High Capacity NVMe SSDs:

- Example device:
  - 4k LBA format
  - nawupf >= npwg > ps → where vendors can enable large atomics
  - LBS device with
    - 4k logical block size but a larger preferred write granularity and atomic support
    - Backwards compatible



Where LBS is not great:

- **LBS** is **not suitable** for **all workloads**
  - **Smaller IOs** with **LBS** can cause write amplification (WAF) due to **read modify writes**
  - But if you do a large write on a 4k bs filesystem writes are not restricted to only 4k, typically larger IOs are used
  - Do your WAF homework, IO volume count is what matters
  - LBS is suitable to store **large data** that can be processed in **larger IO chunks**.

Plumbing

LBS plumbing:

- Historically, **page cache** was closely **tied** to a **PAGE**.
- No support to track the "**blocks**" (filesystem or block device) **> page size** as a **single unit** in the **page cache** to **avoid eviction** of **partial blocks**.

LBS plumbing:

- Historically, **page cache** was closely **tied** to a **PAGE**.
- No support to track the "**blocks**" (filesystem or block device) **> page size** as a **single unit** in the **page cache** to **avoid eviction** of **partial blocks**.

Willy on [**LBS support**]:

> The important reason to need **large folios** to support **large drive block sizes** is that the **block size is the minimum I/O size**. That means that if we're going to **write from the page cache**, we need the **entire block** to be **present**. We can't **evict one page** and then try to **write back the other pages** -- we'd have to read the page we evicted back in. So we want to **track dirtiness and presence on a per-folio basis**; and we must restrict **folio size to be no smaller than block size**.

Page cache before:

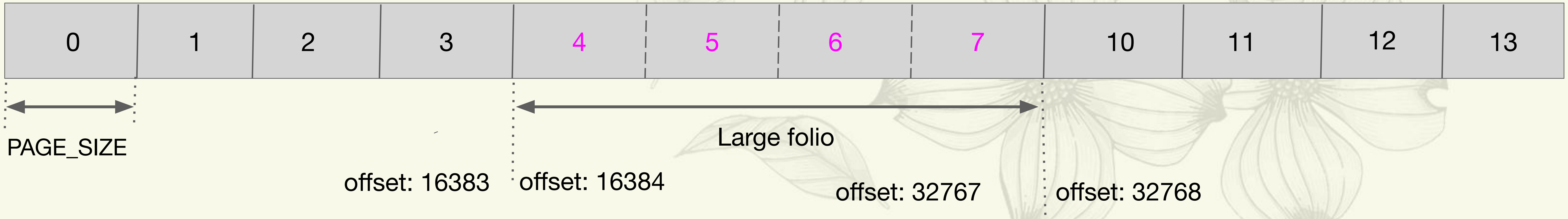
- Historically, **page cache** was closely **tied** to a **PAGE**.

Page cache at the moment:

- Historically, **page cache** was closely **tied** to a **PAGE**.
- **Large folio** support has been added to the **page cache.**
- **Readahead** can use **large folios** if the **filesystem supports** it.
  - **XFS**, **shmem**, **AFS** and **EROFS**
- Since 6.6, **XFS buffered writes** can also use a **large folios.**

Page cache at the moment: 4k PAGE_SIZE

Address space mapping page index example: 48k file



- folio index: offset >> PAGE_SHIFT
- 12 >> 12 → 0
- 4095 >> 12 → 0
- 4096 >> 12 → 1
- 16384 >> 12 → 4
- 32677 >> 12 → 7

- index 4- 7 will return the same folio
- This is one feature which xarray multi-index support allows
  - One folio on multiple indexes
  - **index** must be **aligned** to the **folio order**

Missing piece in the puzzle for LBS XFS:

Large folio support in IOMAP

Large folio support in the page cache

Missing piece in the puzzle for LBS XFS:

- Dave chinner on LBS support for XFS:

> the main blocker why **bs > ps** could not work on XFS was due to the **limitation in page cache**: `**filemap_get_folio**(FGP_CREAT) always **allocate** at **least filesystem block size**`

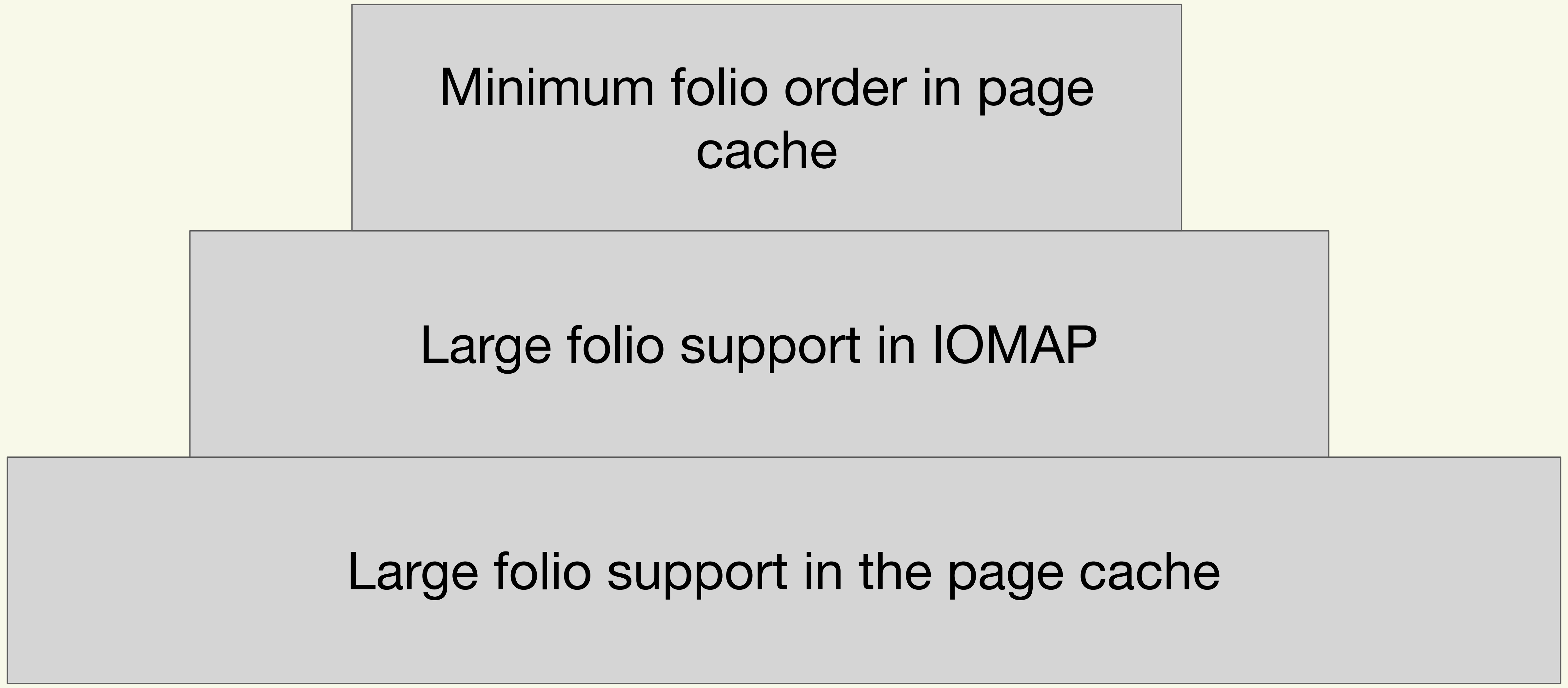
Large folio support in IOMAP

Large folio support in the page cache

Missing piece in the puzzle for LBS XFS:

- Dave chinner on LBS support for XFS:

> the main blocker why **bs > ps** could not work on XFS was due to the **limitation in page cache**: `**filemap_get_folio**(FGP_CREAT) always **allocate** at **least filesystem block size**`



Linux Plumbers Conference | Richmond, VA | Nov. 13-15, 2023

Page cache with min_order folio support:

- **Folios** added to the **page cache** will be **at least** with a **minimum order**.

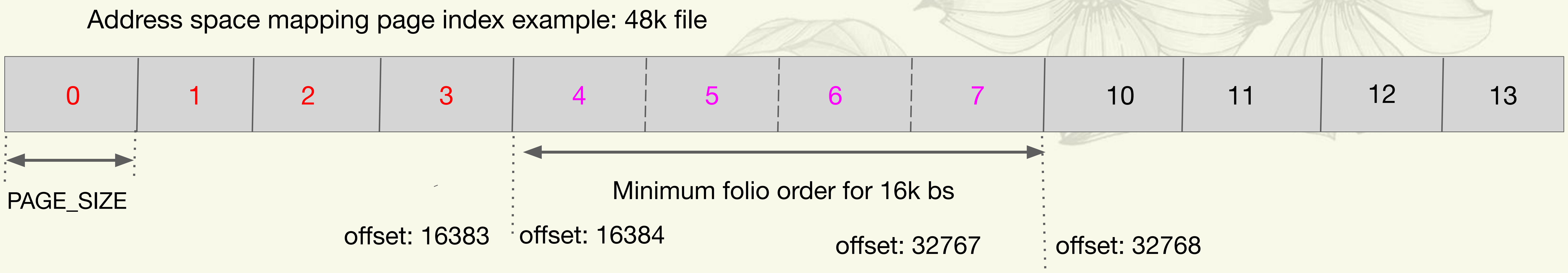
Page cache with min_order folio support:

- **Folios** added to the **page cache** will be **at least** with a **minimum order**.
- Filesystems can **set the min_order** of the **page cache** while setting up an **inode.**

Page cache with min_order folio support:

- **Folios** added to the **page cache** will be **at least** with a **minimum order**.
- Filesystems can **set the min_order** of the **page cache** while setting up an **inode.**
- **min_order** typically corresponds to the **FSB** for **filesystems** or **logical block size** if it is **block cache.**

Page cache with 4k PAGE_SIZE and min_order 2:



- A **folio index** must always aligned to the **minimum order**

Scope of this work:

- folios and xarray multi-index page cache surgery by Matthew Wilcox already removed the assumption of page size
  - **We build on this:**
    - **Add LBS support by re-using using xarray multi-index support for a minimum address space mapping order requirement**.
      - Used for inode allocation
    - Adds API to control **minimum folio** order in the **page cache**

Scope of this work:

- folios and xarray multi-index page cache surgery by Matthew Wilcox already removed the assumption of page size
  - **We build on this:**
    - **Add LBS support by re-using using xarray multi-index support for a minimum address space mapping order requirement**.
      - Used for inode allocation
    - Adds API to control **minimum folio** order in the **page cache**
- Enable **LBS** support in **XFS**.
  - Most **heavy lifting** already done by the **community** by it using **iomap** and **supporting multiple block sizes**
  - Minor filesystem changes on our side
  - **fstests** gives a good test bed to **stress test** the page cache and **shake out** all the **bugs**.

Implementation

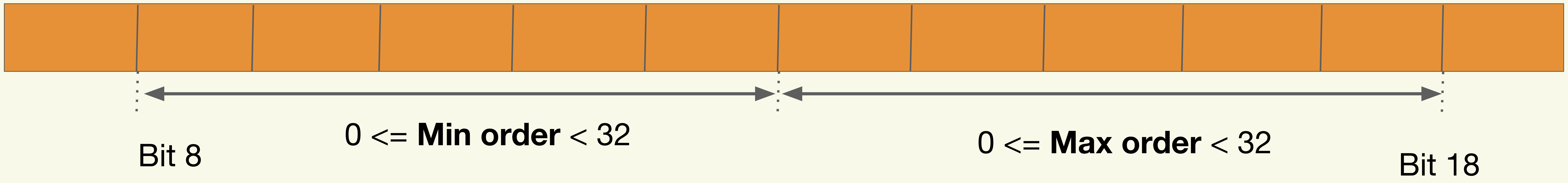
API to set minimum folio order:

```
void mapping_set_folio_orders(struct address_space *mapping,
                              unsigned int min, unsigned int max)
```

API to set minimum folio order:

```
void mapping_set_folio_orders(struct address_space *mapping,
                              unsigned int min, unsigned int max)
```

```
struct address_space {
    struct inode        *host;
    struct xarray       i_pages;
...
    unsigned long       flags;
...
};
```



Usage:

- **Set** the **preferred minimum order** while **allocating** a **folio** in the **page cache** during the **initialization** of **inodes**.

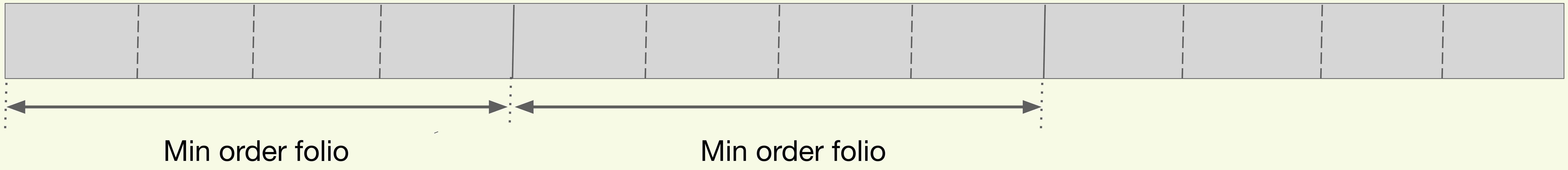
```
/*
 * Allocate and initialise an xfs_inode.
 */
struct xfs_inode *
xfs_inode_alloc(
	struct xfs_mount	*mp,
	xfs_ino_t		ino)
{
	//...
	mapping_set_folio_orders(VFS_I(ip)->i_mapping, min_order, MAX_PAGECACHE_ORDER);
	//...
}
```

Changes to allocation and placement:

- **filemap_alloc_folio** always with **at least min order** and **filemap_add_folio** at **index aligned** to the **min order.**

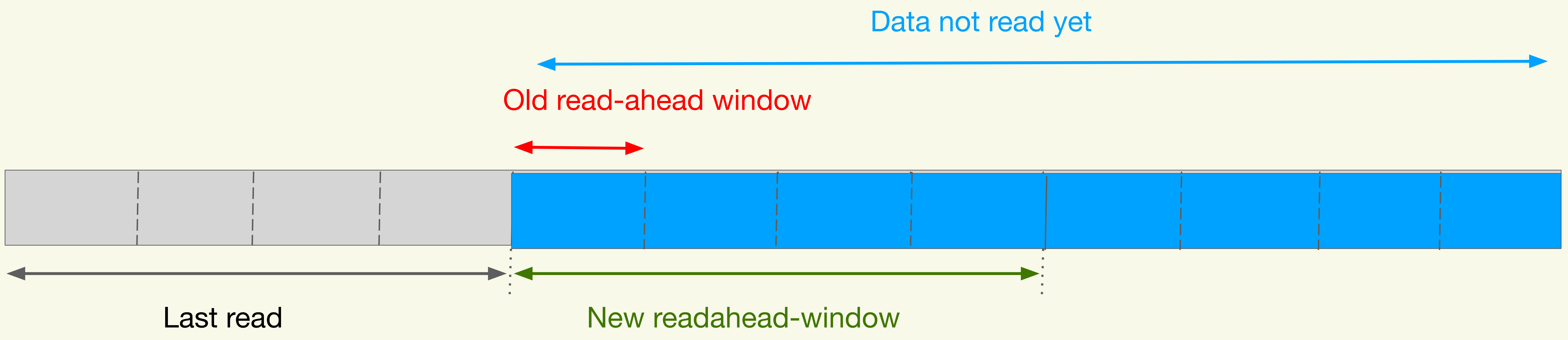
```
    int min_order = mapping_min_folio_order(mapping);
    int nr_of_pages = (1U << min_order);

    index = round_down(index, nr_of_pages);
...
    folio = filemap_alloc_folio(gfp_mask, min_order);
...
    filemap_add_folio(mapping, folio, index, gfp_mask);
```



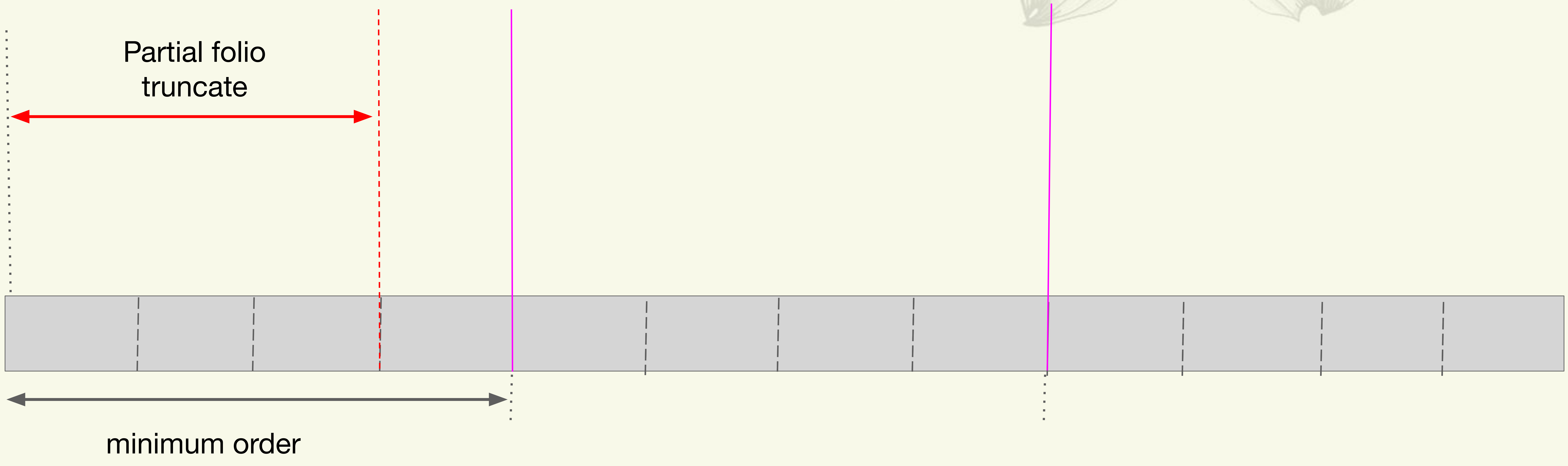
Changes in readahead:

- Readahead uses a heuristic to read things ahead of needing them
- It's algorithm is archaic, and could be improved, but we just need it to work
- Readahead allocates folios and moves the index accordingly
  - **These moves and shifts must account for the minimum order**



Changes in truncate:

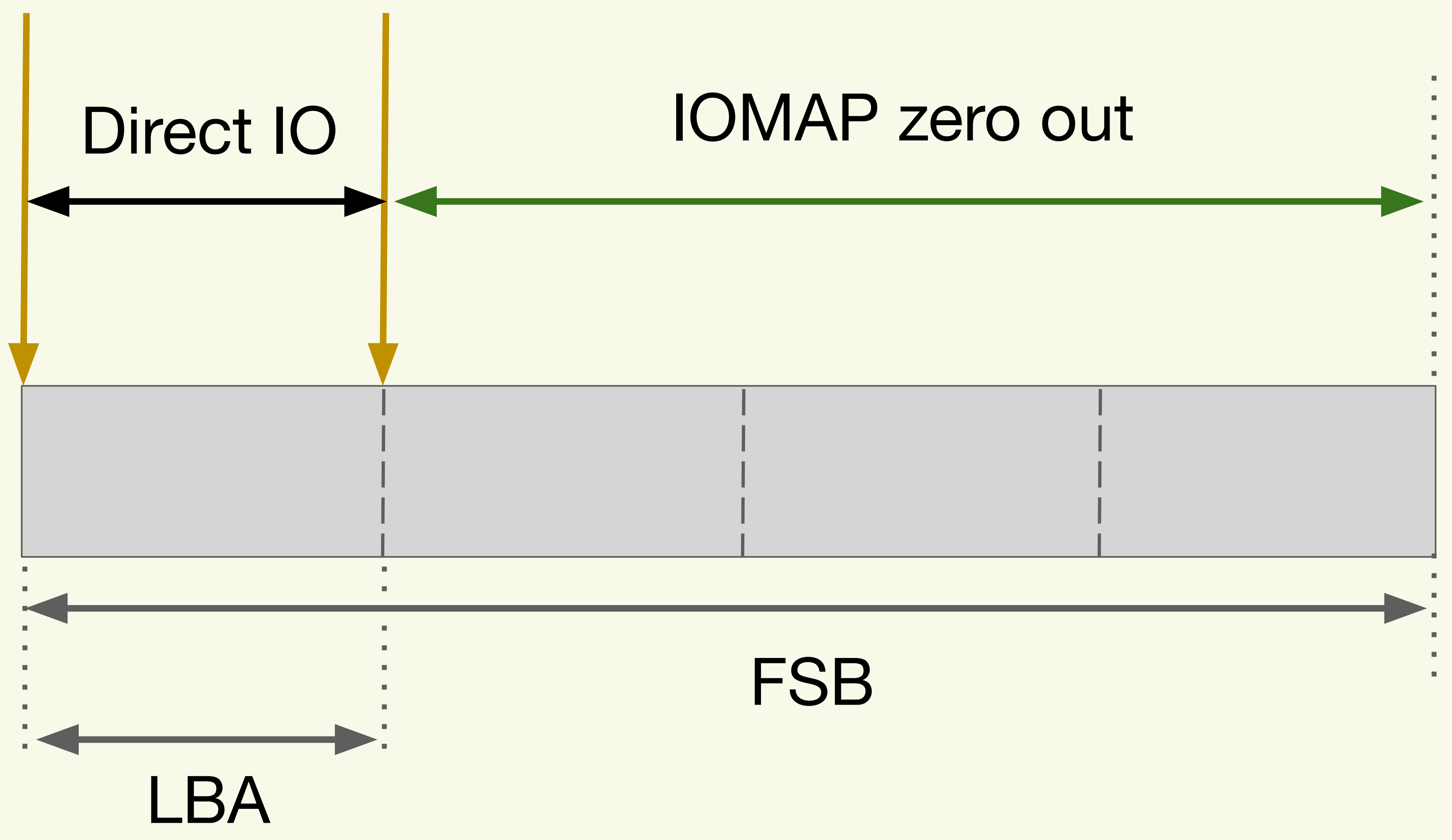
- **Partial truncate** on a **large folio** can result in **splitting**. (**truncate_inode_partial_folio()**)
- **Do not split a folio** which has a **minimum order** that needs to be maintained.
- **Truncate** it **completely** or do **not truncate**.



Hidden surprises in IOMAP direct IO path:

- iomap_dio_zero() will pad a FSB with zeroes if the direct IO size < FSB.
- Uses boot time allocated ZERO_PAGE to zero out.
- Hidden assumption that block size <= PAGE_SIZE.

```
static void iomap_dio_zero(..., loff_t pos, unsigned len)
{
    struct page *page = ZERO_PAGE(0);
 ...
    bio->bi_iter.bi_sector = iomap_sector(&iter->iomap, pos);
 ...
    __bio_add_page(bio, page, len, 0);
 ...
}
```



Testing

IO distribution analysis with FIO:

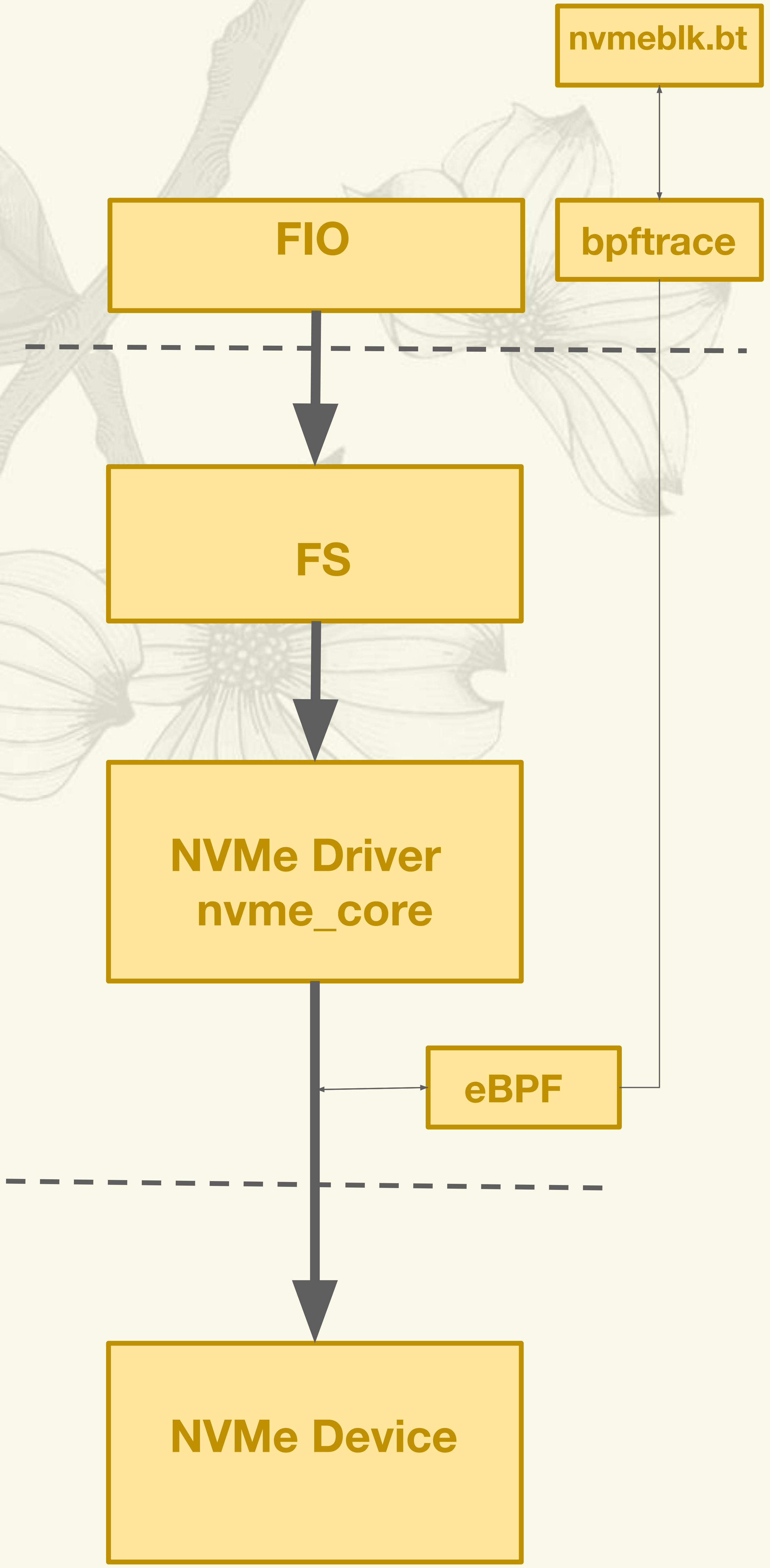
- Preliminary analysis to verify **IO size** with **LBS support** in **XFS**.
- Baseline is **ext4** with **bigalloc** and **XFS** with **default block size(4k)**.
- FIO job with **64k IO** block size:

```
$ fio --directory=/mnt/ --bs=64k --ioengine=io_uring
--rw=randwrite --size=50G --create_on_open=1 --nrfiles=10
--fsync_on_close=1 --name=yolo
```

- This is an **ideal workload**. More of a **litmus test.**
- More real world benchmarks to needs to be performed.

IO distribution analysis with FIO:

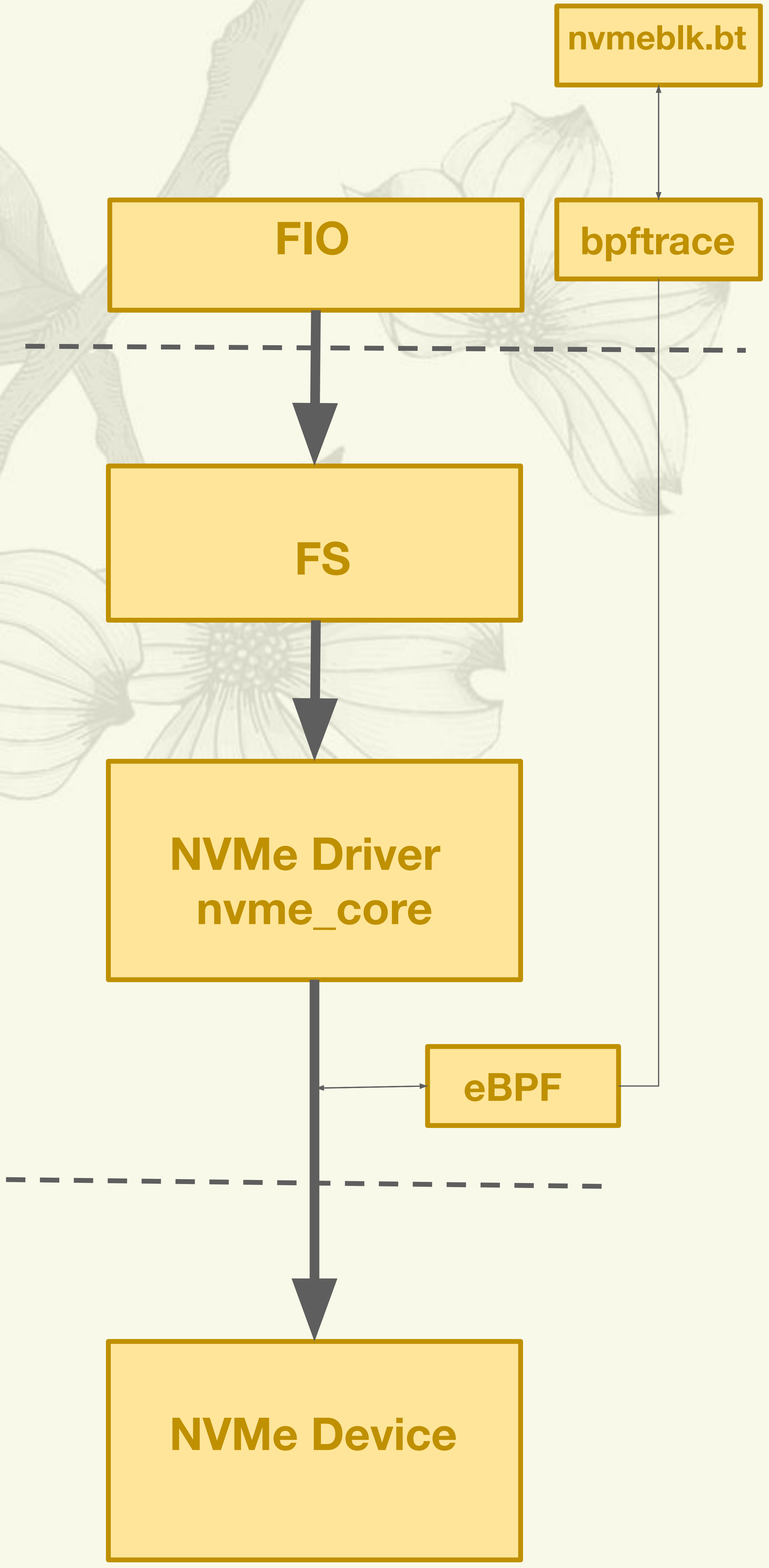
FS	Filesystem Block Size
EXT4	4k with Cluster size 64k(bigalloc)
XFS	4k
XFS with LBS support	64k

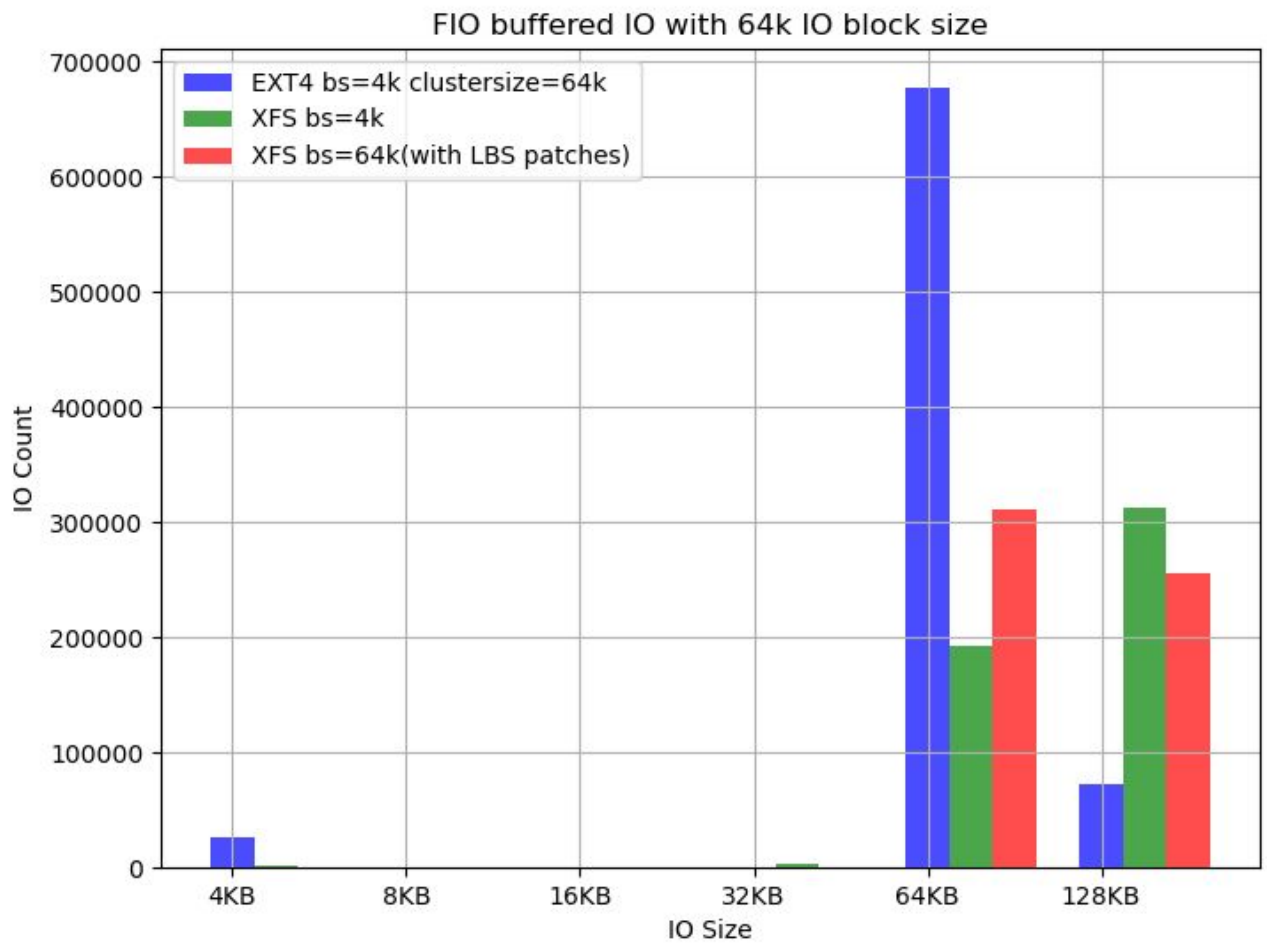


IO distribution analysis with FIO:

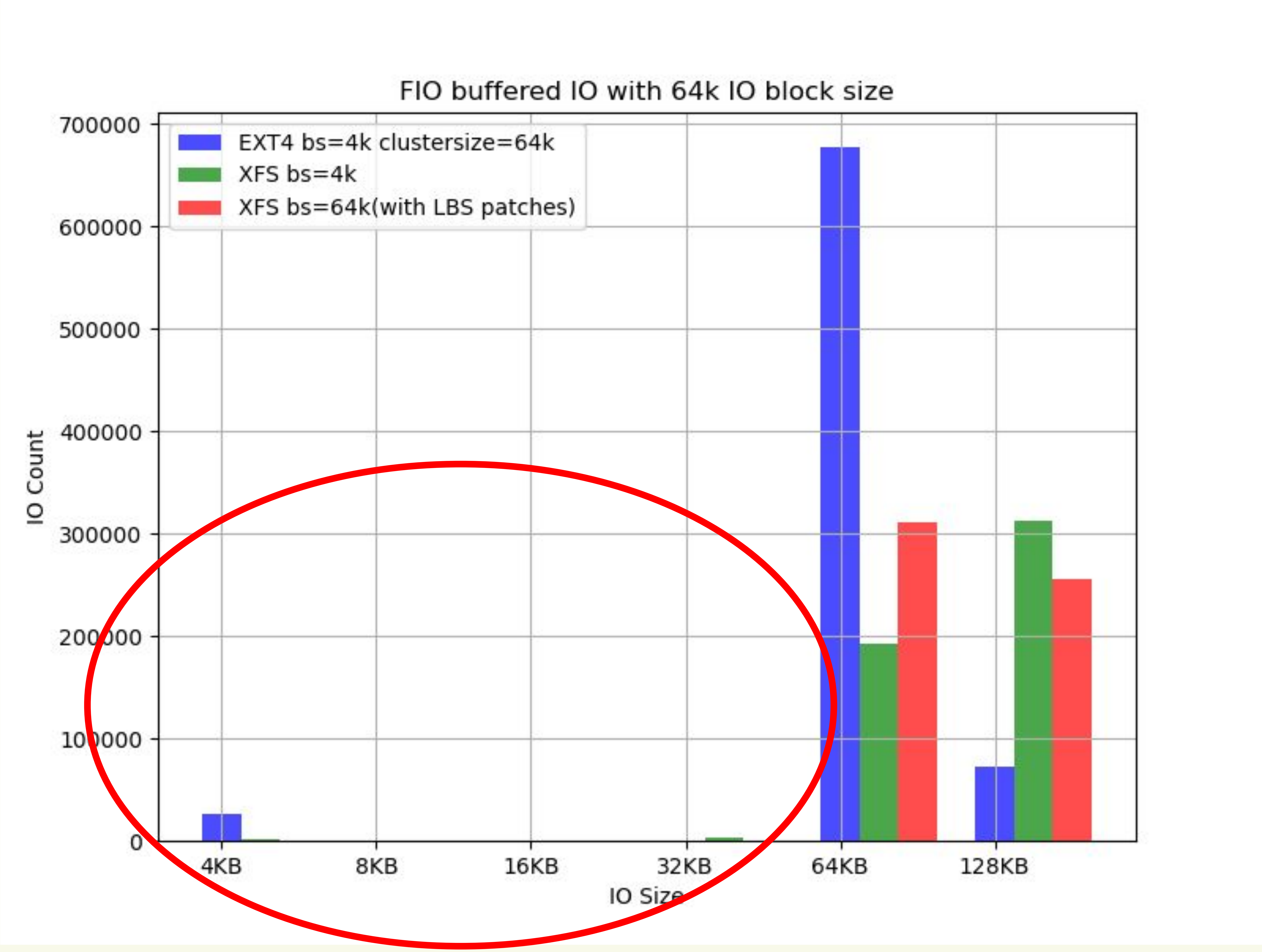
FS	**Filesystem Block Size**
EXT4	4k with Cluster size 64k(bigalloc)
XFS	4k
XFS with LBS support	64k

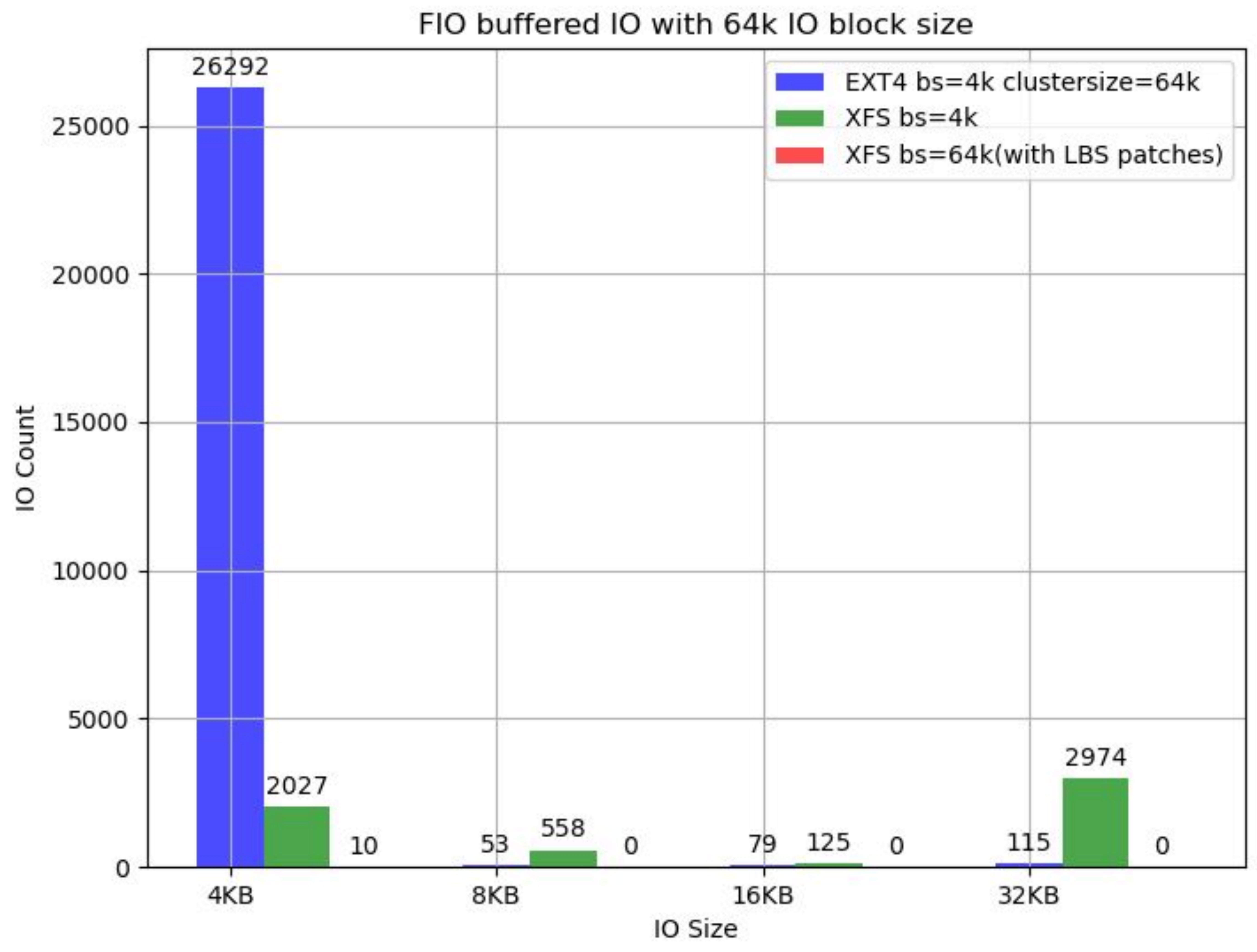
x86_64 VM(qemu)	RAM	**16GB**
	Storage (NVMe)	capacity: 3.76TB
		LBA size: 4k
		MDTS: 256k
Kernel version	v6.6-rc5	

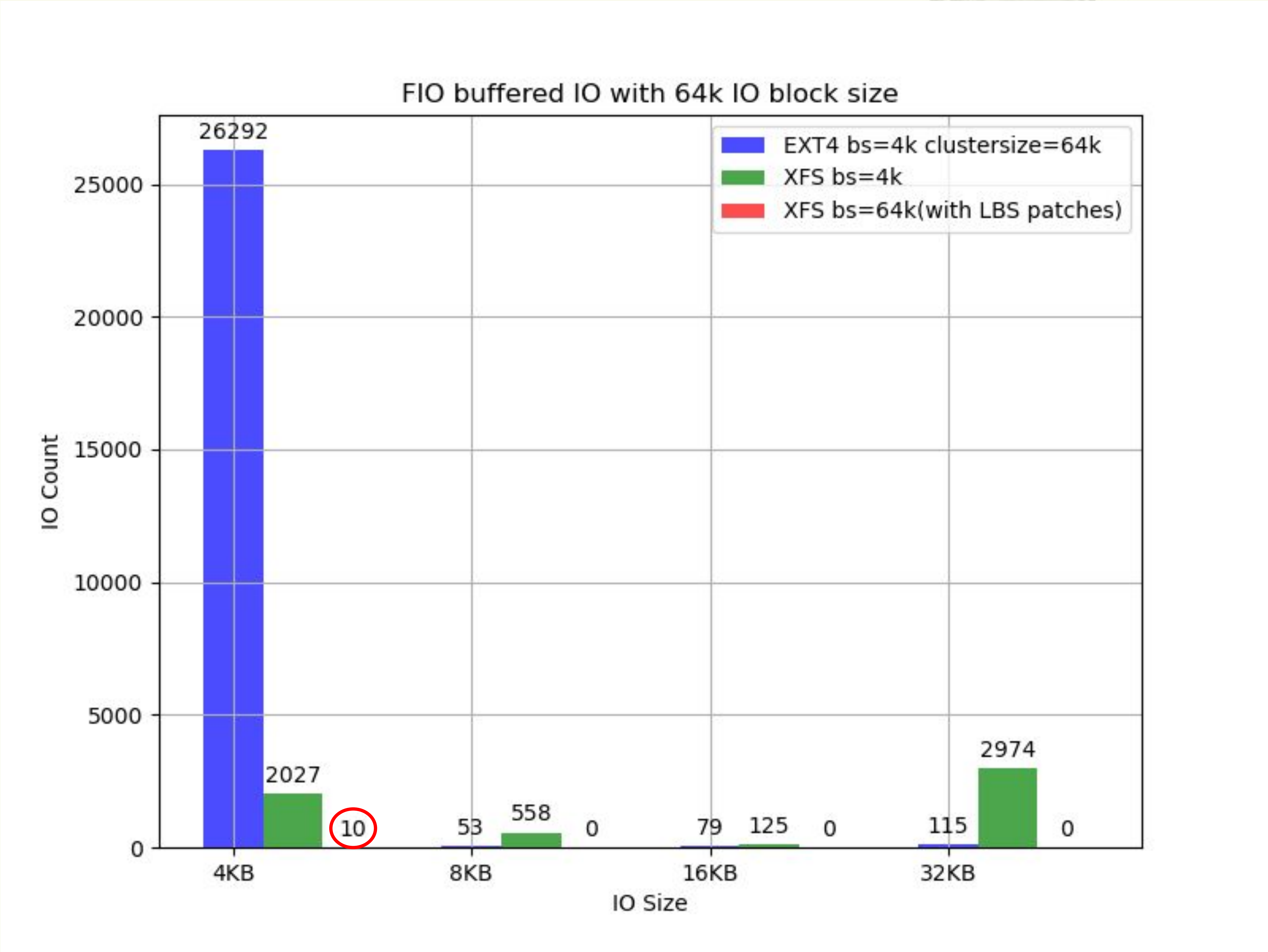



FIO buffered IO with 64k IO block size
EXT4 bs=4k clustersize=64k
XFS bs=4k
XFS bs=64k(with LBS patches)
IO Count
700000
600000
500000
400000
300000
200000
100000
0
4KB
8KB
16KB
32KB
64KB
128KB
IO Size

Linux Plumbers Conference | Richmond, VA | Nov. 13-15, 2023




FIO buffered IO with 64k IO block size
EXT4 bs=4k clustersize=64k
XFS bs=4k
XFS bs=64k(with LBS patches)
IO Count
IO Size
0
5000
10000
15000
20000
25000
4KB
8KB
16KB
32KB
26292
2027
10
53
558
0
79
125
0
115
2974
0

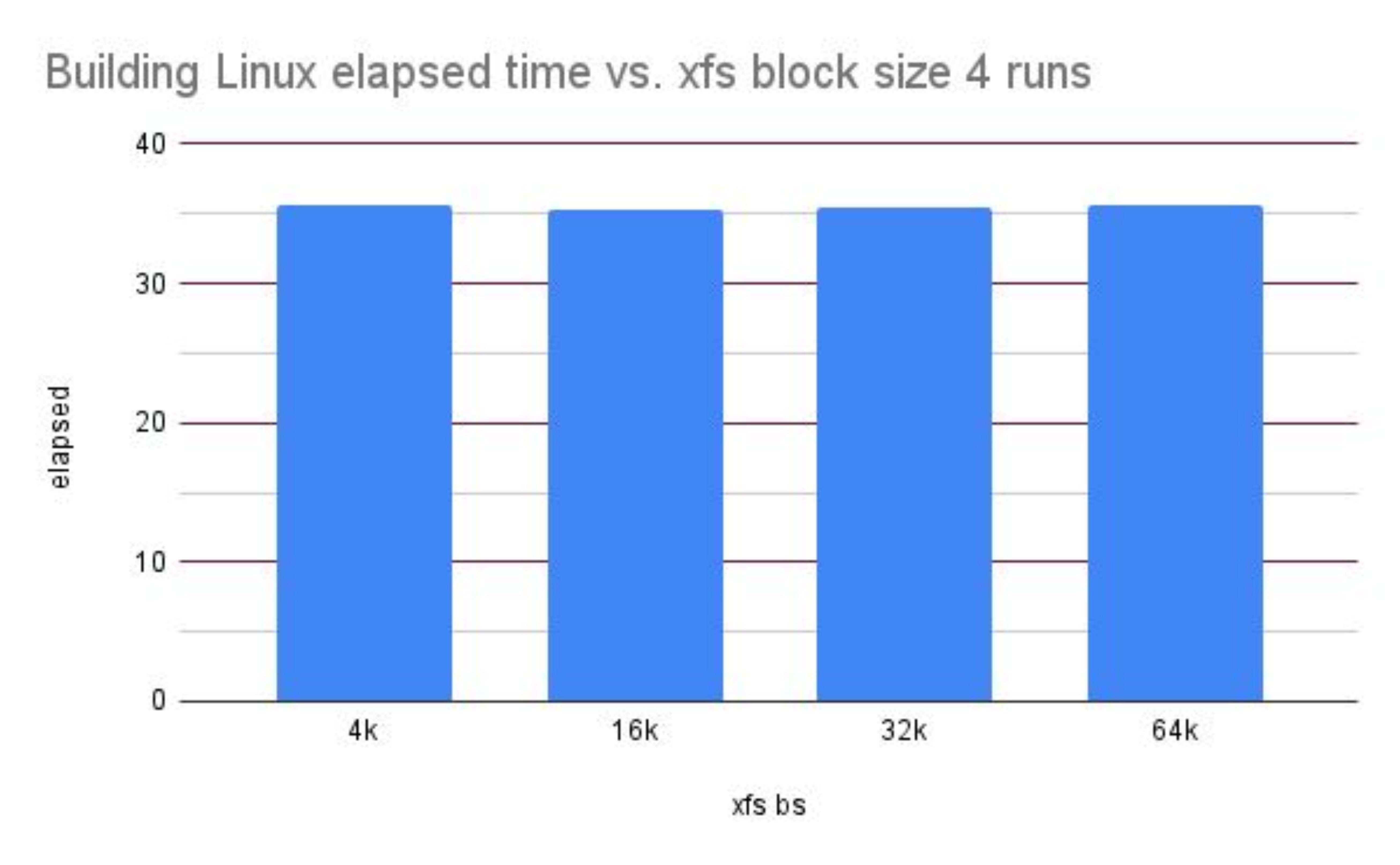


The **4k IOs** in **XFS** with **64k** block size is coming from **metadata writes(xfsaild)**

Building Linux

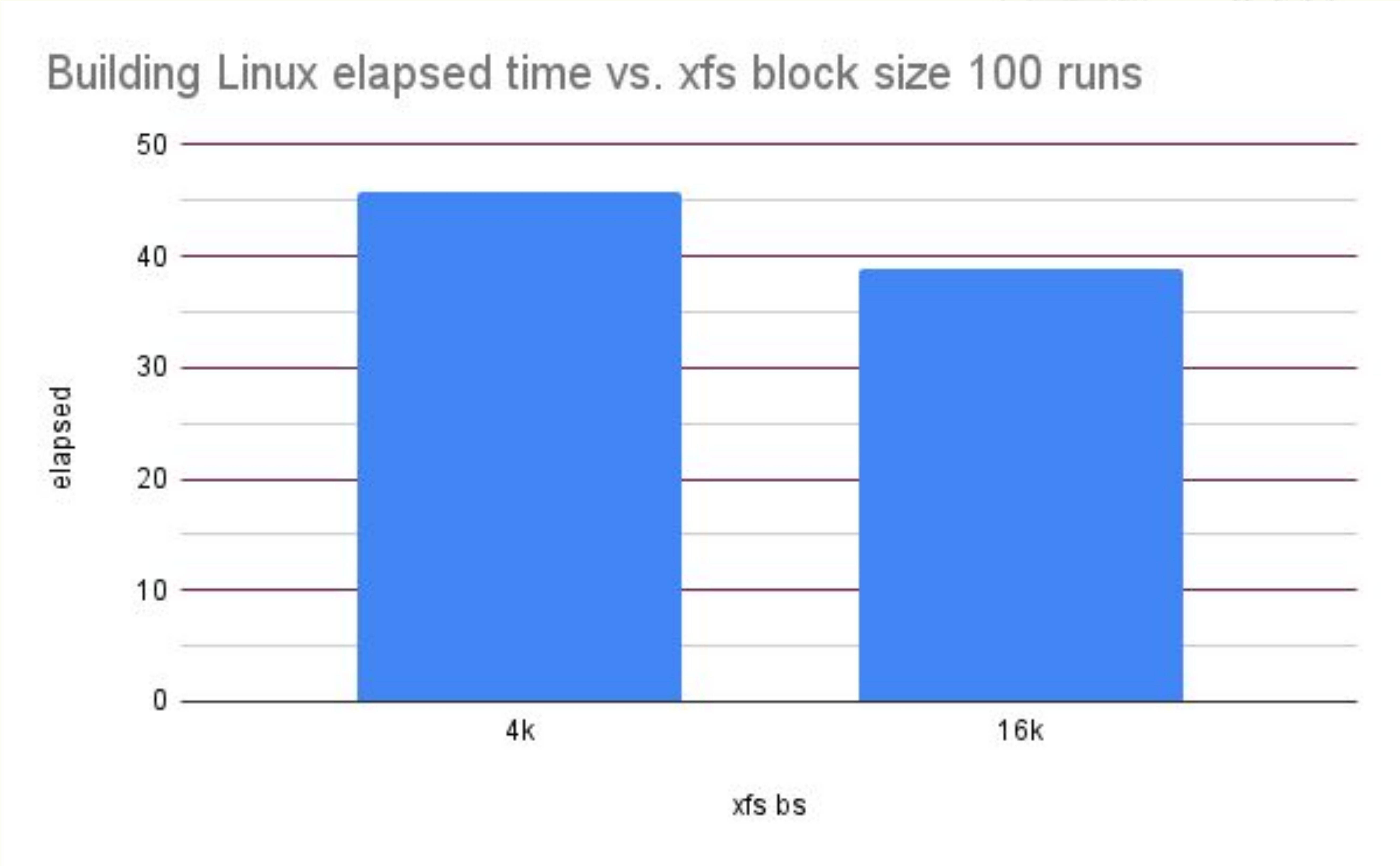
```
x=100
perf stat --repeat $x \
  --pre 'make -s mrproper defconfig' \
  -- make -s -j$(nproc) bzImage
```

Building Linux



- Intel Xeon Gold 6438Y+
- nproc: 128
- 1 TiB Memory

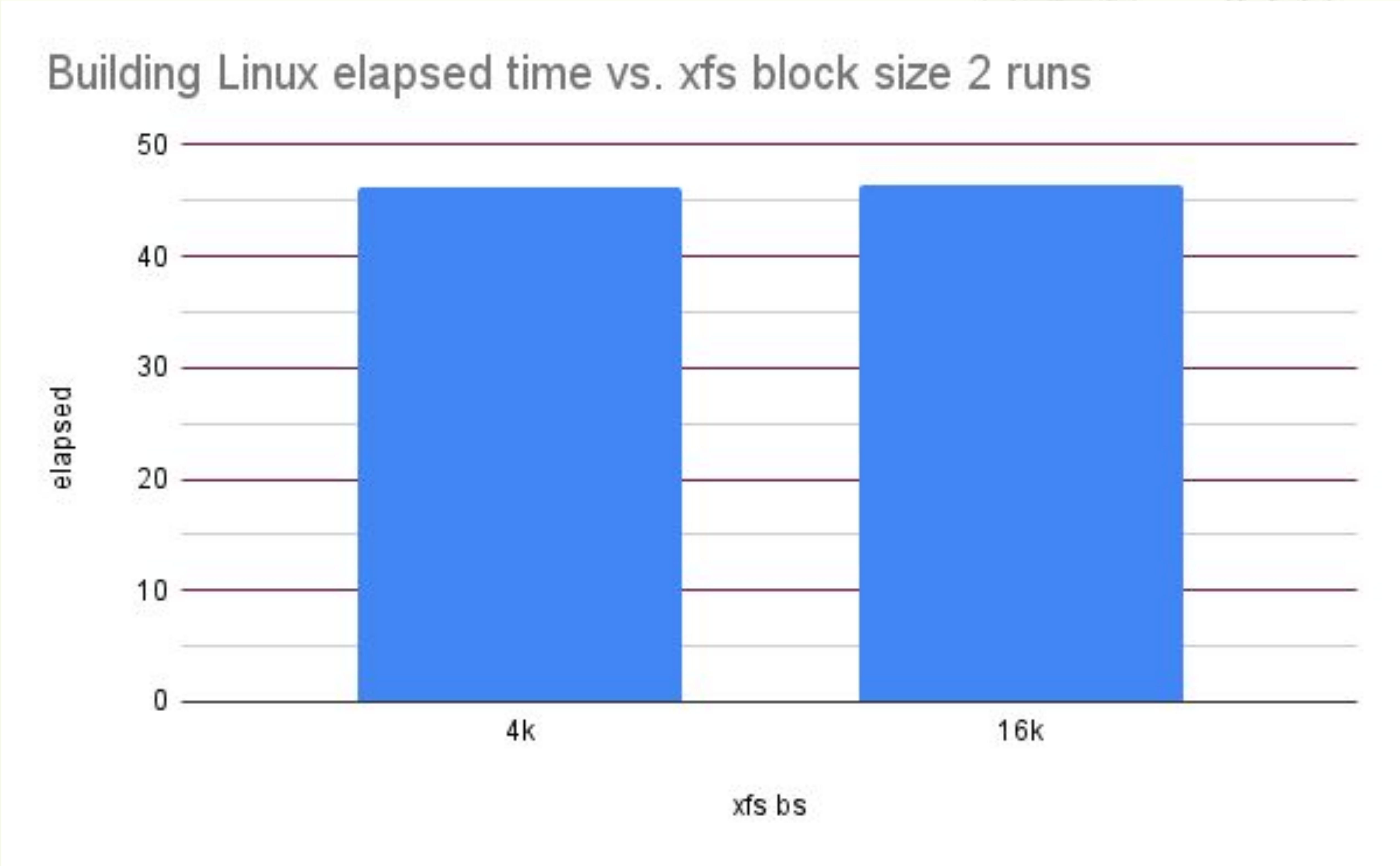
Building Linux



- Intel Xeon Gold 6438Y+
- nproc: 128
- 1 TiB Memory

Linux Plumbers Conference | Richmond, VA | Nov. 13-15, 2023

Building Linux



- AWS c7a.metal-48xl
- AMD EPYC 9R14
- 192 VCPUS but bare metal??
- 384 GiB RAM

AMD EPYC 9R14 - No PTE Coalescing fail : (

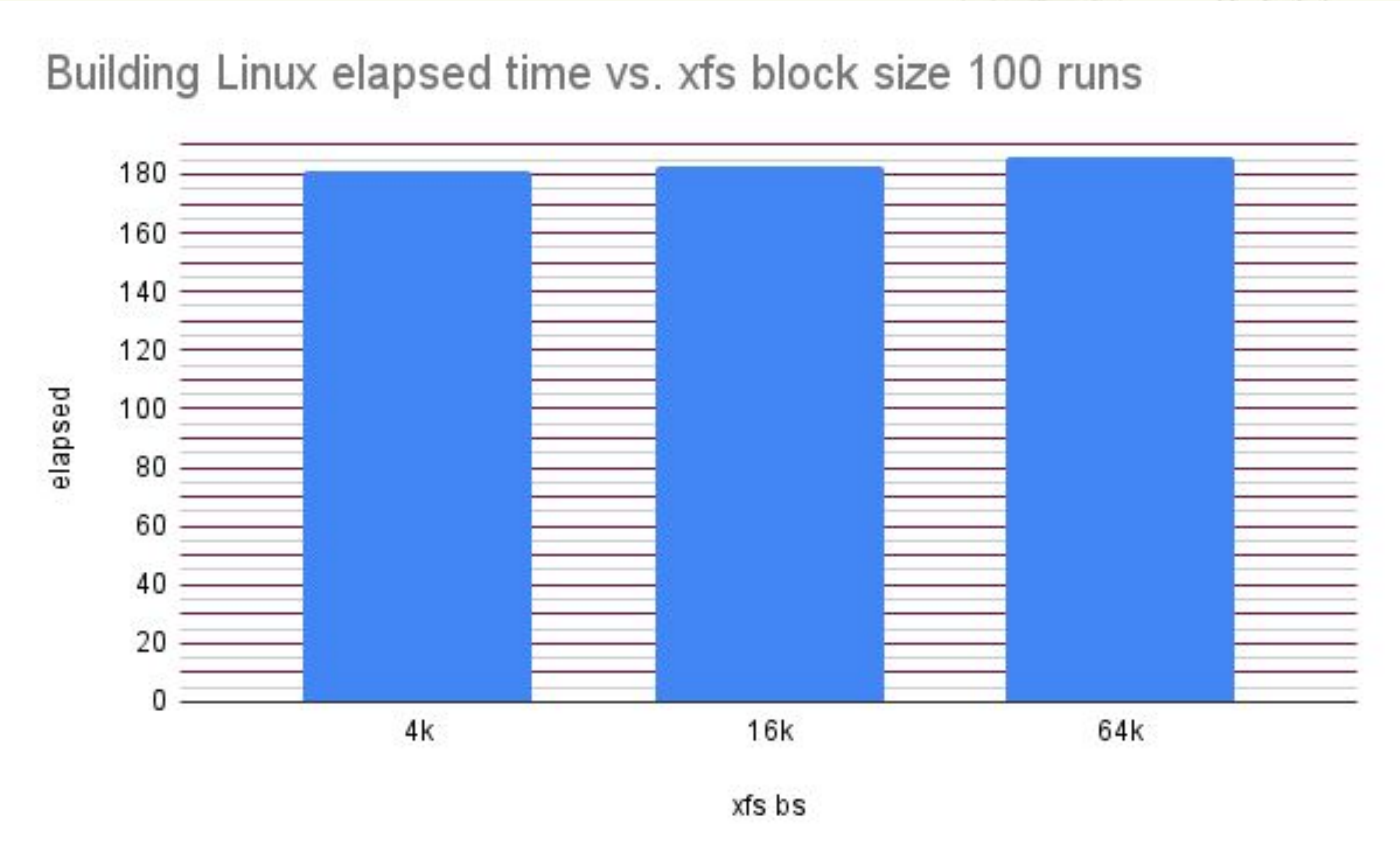
```
mcgrof@amd /mnt-xfs-16k/linux (git::master)$ /home/mcgrof/build-pg-v1.sh

 Performance counter stats for 'make -s -j96 bzImage' (2 runs):

       46513784329 ns   duration_time                                        ( +-  0.74% )
     1051852925250 ns   user_time                                            ( +-  0.01% )
      935936429750 ns   system_time                                          ( +-  0.10% )
          52656536      page-faults                                          ( +-  0.02% )
               264      major-faults                                         ( +-  0.19% )
          52656272      minor-faults                                         ( +-  0.02% )
...
                 0      bp_l1_tlb_miss_l2_tlb_miss.coalesced_4k                     (19.94%)
...
                 0      ls_l1_d_tlb_miss.tlb_reload_coalesced_page_hit                    (20.33%)
                 0      ls_l1_d_tlb_miss.tlb_reload_coalesced_page_miss                   (20.34%)
```

AMD TLB Coalescing on
AWS c7a.metal-48xl
“Bare metal” ?

Building Linux



- AWS c7a.8xlarge
- AMD EPYC 9R14
- 32 VCPUs
- 64 GiB RAM

Building Linux

- Needs more evaluation

Device test matrix

- Plenty to test!
- First focus on avoiding regressions
- Testing 512 LBA and 4k LBA
- Even though larger LBAs are functional as tested with qemu

LBA Format	block size	sector size
512 bytes	512 bytes	512 bytes
	4k	512 bytes
	16k	512 bytes
	32k	512 bytes
	64k	512 bytes
4k	4k	4k
	16k	4k
		16k
	32k	4k
		16k
		32k
	64k	4k
		16k
		32k
		64k
16k	16k	16k
	32k	16k
		32k
	64k	16k
		32k
		64k
32k	32k	32k
	64k	32k
64k	64k	64k

Legend
No new drive requireed
IU matching npwg + awupf
New LBA formats
XFS format change

XFS test profile matrix

Baseline profiles

1. xfs_crc
2. xfs_crc_logdev
3. xfs_crc_rtdev
4. xfs_crc_rtdev_extsize_28k
5. xfs_crc_rtdev_extsize_64k
6. xfs_crc_logdev_rtdev
7. xfs_nocrc
8. xfs_nocrc_512
9. xfs_nocrc_1k
10. xfs_nocrc_2k
11. xfs_nocrc_4k
12. xfs_reflink
13. xfs_reflink_1024
14. xfs_reflink_normapbt
15. xfs_reflink_stripe_len
16. xfs_reflink_nrext64
17. xfs_reflink_logdev
18. xfs_reflink_2k
19. xfs_reflink_4k
20. xfs_reflink_dir_bsize_8k

New LBS profiles

1. xfs_nocrc_16k
2. xfs_nocrc_16k_4ks
3. xfs_nocrc_32k
4. xfs_nocrc_32k_4ks
5. xfs_nocrc_64k
6. xfs_nocrc_64k_4ks
7. xfs_reflink_16k
8. xfs_reflink_16k_4ks
9. xfs_reflink_32k
10. xfs_reflink_32k_4ks
11. xfs_reflink_64k
12. xfs_reflink_64k_4ks

Plenty to test!

Kdevops fstests testing

- kdevops allows you to get a test rig for all this up in about 20-30 minutes
- You'll need about **4 TB drive for all test baseline profiles**
- 4 GiB per guest x 20 >= **80 GiB RAM**
  - Have not needed yet more for LBS – surprising result
- Our initial priority: detect regressions fast
- Pick a baseline kernel target: v6.6-rc5
- **Get baseline**
- Build huge confidence in baseline
  - Objecive: 100 loops of fstests without no new failures
- Means we will report bugs
- SOAK_DURATION=9900

Linux Plumbers Conference | Richmond, VA | Nov. 13-15, 2023

kdevops v6.6-rc5 baseline xfs bug hunting screenshot

```
Every 60.0s: ./scripts/workflows/fstests/fstests_watchdog.py hosts baseline
deb-server-666-number-of-the-beast: Wed Nov  8 20:11:57 2023
```

Hostname	Test-name	Completion %	runtime(s)	last-runtime(s)	Stall-status	Kernel
base-xfs-crc	generic/642	1% (soak)	60	11028	OK	6.6.0-rc5
base-xfs-crc-logdev	generic/601	100%	6	6	OK	6.6.0-rc5
base-xfs-crc-rtdev	generic/591	175%	7	4	OK	6.6.0-rc5
base-xfs-crc-rtdev-extsize-28k	generic/642	103% (soak)	10788	10505	OK	6.6.0-rc5
base-xfs-crc-rtdev-extsize-64k	generic/531	31395%	11616	37	Hung-Stalled	6.6.0-rc5
base-xfs-crc-logdev-rtdev	None	0%	0	0	OK	6.6.0-rc5
base-xfs-reflink	generic/476	79% (soak)	9132	11556	OK	6.6.0-rc5
base-xfs-reflink-normapbt	generic/476	91% (soak)	9138	10069	OK	6.6.0-rc5
base-xfs-reflink-stripe-len	generic/476	79% (soak)	8347	10500	OK	6.6.0-rc5
base-xfs-reflink-nrext64	generic/476	72% (soak)	8491	11799	OK	6.6.0-rc5
base-xfs-reflink-logdev	generic/476	74% (soak)	8527	11546	OK	6.6.0-rc5
base-xfs-reflink-1024	generic/476	59% (soak)	6831	11608	OK	6.6.0-rc5
base-xfs-reflink-2k	generic/476	69% (soak)	7956	11508	OK	6.6.0-rc5
base-xfs-reflink-4k	generic/476	72% (soak)	8610	11901	OK	6.6.0-rc5
base-xfs-reflink-dir-bsize-8k	generic/476	77% (soak)	8956	11603	OK	6.6.0-rc5
base-xfs-nocrc	None	0%	0	0	OK	6.6.0-rc5
base-xfs-nocrc-512	None	0%	0	0	OK	6.6.0-rc5
base-xfs-nocrc-1k	None	0%	0	0	OK	6.6.0-rc5
base-xfs-nocrc-2k	None	0%	0	0	OK	6.6.0-rc5
base-xfs-nocrc-4k	None	0%	0	0	OK	6.6.0-rc5

v6.6-rc5 upstream baseline xfs results so far...

- Baseline confidence: ~ 25 full loops of running fstests
- All fstests results kept in tarballs on kdevops git tree
- All failures itemized as expunges, crashes/hangs in github gists
- Approximate failure rate notation: F:1/20 fails about ~ 1/20 times
- 443 known failed tests, 47 crashes

v6.6-rc5 upstream baseline xfs results so far...

- Baseline confidence: ~ 25 full loops of running fstests
- All fstests results kept in tarballs on kdevops git tree
- All failures itemized as expunges, crashes/hangs in github gists
- Approximate failure rate notation: F:1/20 fails about ~ 1/20 times
- 443 known failed tests, 47 crashes
- `Assertion failed: ip->i_nblocks == 0, file: fs/xfs/xfs_inode.c`
- `Assertion failed: (irec->br_blockcount & ~XFS_IEXT_LENGTH_MASK) == 0`
  - `xfs_inodegc_worker() → xfs_ifree()|xfs_ixset()`
- `hung tasks:`
  - `xfs_log_unmount()`
  - `iomap_writepages() → xfs_buf_read_map()`
  - ...

v6.6-rc5 upstream baseline xfs results so far...

- Rare flaky crashes as well such as:
- `invalidate_inode_pages2_range()` crash
  - `buffered IO + async DIO` – *this is stupid to do anyway but we support it (TM)*
  - `VM_BUG_ON_FOLIO(!folio_contains(folio, indices[i]), folio)`
  - F:1/1604
- fsstress + compaction crashes → means we should add a new test
  - readahead triggered alloc + compaction
  - F:1/20

v6.6-rc5 upstream LBS results so far

v6.6-rc5 upstream LBS results so far

- Few actual LBS related issues, few tests bugs in light of LBS
  - xfs/599: # of xattrs based on block size - reported - no feedback yet
    - n=16k takes 5 seconds
    - n=32k takes 30 seconds
    - n=64k takes 6-7 minutes
    - n=1048576 takes 30 hours

v6.6-rc5 upstream LBS results so far

- Few actual LBS related issues, few tests bugs in light of LBS
  - xfs/599: # of xattrs based on block size - reported - no feedback yet
    - n=16k takes 5 seconds
    - n=32k takes 30 seconds
    - n=64k takes 6-7 minutes
    - n=1048576 takes 30 hours

Some tests need to be fixed for larger block sizes (without LBS)

v6.6-rc5 upstream LBS results so far

Few actual LBS related issues, few tests bugs in light of LBS

- xfs/599: # of xattrs based on block size - reported - no feedback yet
  - n=16k takes 5 seconds
  - n=32k takes 30 seconds
  - n=64k takes 6-7 minutes
  - n=1048576 takes 30 hours

Some tests need to be fixed for larger block sizes (without LBS)

Not yet done with testing but... zero regressions detected so far

LBS next steps

- Finish testing minorder patches
- Post patches
- Block device cache:
  - dynamic aops not ideal - patches posted
  - iomap buffer-head compatibility suggested instead → requires work
  - buffer-head large folio support from Hannes
    - not ideal unless we have a real filesystem to help test this
    - Other filesystem, filesystem developers decide:
      - gfs2 seems like a good next target due to interest by Andreas
- Lift NVMe restrictions - already implemented

Concerns

Fragmentation concerns

- Thesis:
  - reclaim should address concerns:
    - As you allocate large folios these same large folios will be available after reclaim for use
    - Should not starve 4k
- Testing thesis should be possible now

Fragmentation concerns

- I asked for simple memory fragmentation measurement
- Proposal suggested by John Hubbard:
- a) Let BLOCKS be the number of 4KB pages (or more generally, then number of smallest sized objects allowed) in the area.
- b) Let FRAGS be the number of free **or** allocated chunks (no need to consider the size of each, as that is automatically taken into consideration).
- Then:

  fragmentation percentage = (FRAGS / BLOCKS) * 100%
- Memory compaction for high order folios RFC from Zi Yan patches posted.

Interested?

Call for action:

- Let's chat, come talk
- Review of our patches
- Help test
- Join our monthly LBS virtual zoom cabal to coordinate
  - Next one: December 5pm PST / 10am Japan
  - Just email us if interested
- Discord kdevops server #large-block

Questions?

Linux Plumbers Conference

Richmond, Virginia | November 13-15, 2023