Whamcloud

LDISKFS Block Allocator Problems and Solutions

Lustre Administrators & Developers Workshop 2022

ablagodarenko@whamcloud.com

ddn

LDISKFS Multi-Block Allocator (mballoc)

Whamcloud

Ext4 has quite sophisticated block allocation optimization subsystem
Multi-block buddy allocator efficiently allocates large chunks of space

Buddy allocator is used if:

- File size is bigger than `s_mb_stream_request` (64KB)
- Group preallocation can't find required blocks
- Required blocks are not found in inode preallocation list
- Required blocks are not found in locality group realloc space

Allocator tries to preallocate as many blocks possible within the preallocation window

There is No Limit to Perfection

Whamcloud

LDISKFS allocator

- Some performance drop when a target is about to be filled
- Too much metadata: slow mount
- Uses free space at end of disk (slow) when space at start is available

New upstream Ext4 allocator

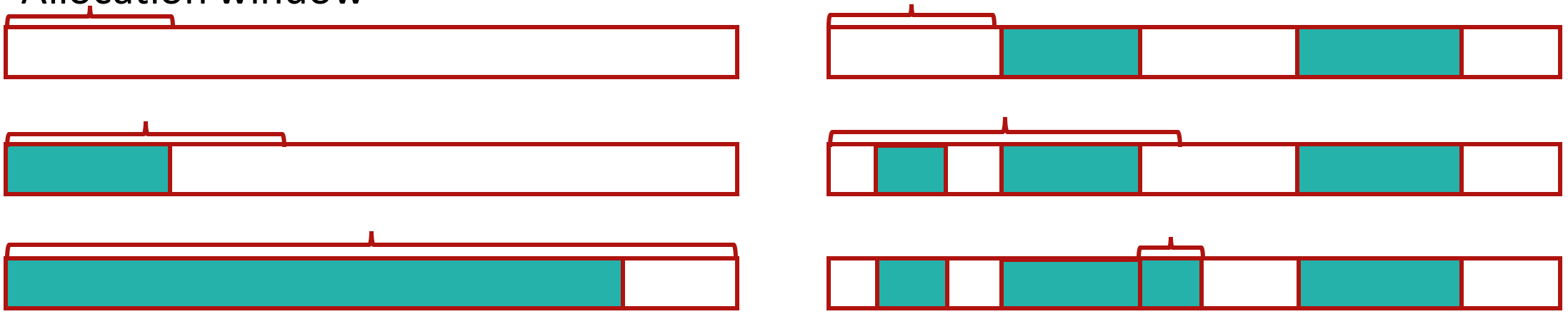
- Files are spread across the target after optimization
- Last groups have more priority than first one

- not a problem for flash drives though

Allocation Window

Whamcloud

Unfragmented vs. fragmented



Allocator scans whole disk trying to find large-enough contiguous range of blocks.
As disks become larger - the problem becomes increasingly visible.

Allocation Window Tuning

```
[ 2^0                          2^13]
[ 0 1 1 0 0 1 0 1 1 1 1 0 0 0 ]
[ 1 0 1 1 2 2 1 1 3 1 1 1 0 0 ]
[ 0 0 0 0 0 1 0 0 1 0 0 1 0 0 ]
```

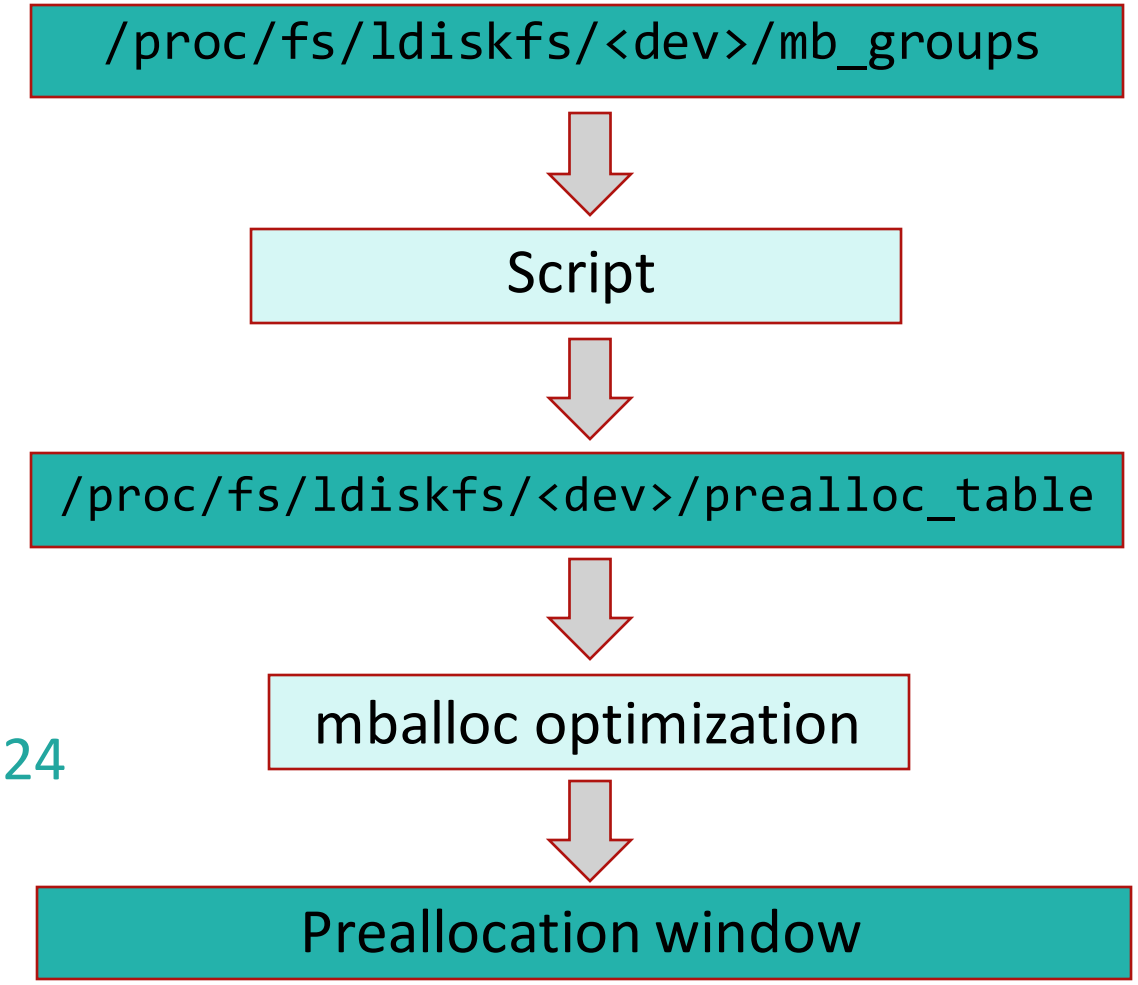
1 2 4 8 16 32 64 128 256 512 1024 2048 4096

1 2 4 8 16 32 64 128 256 512 1024

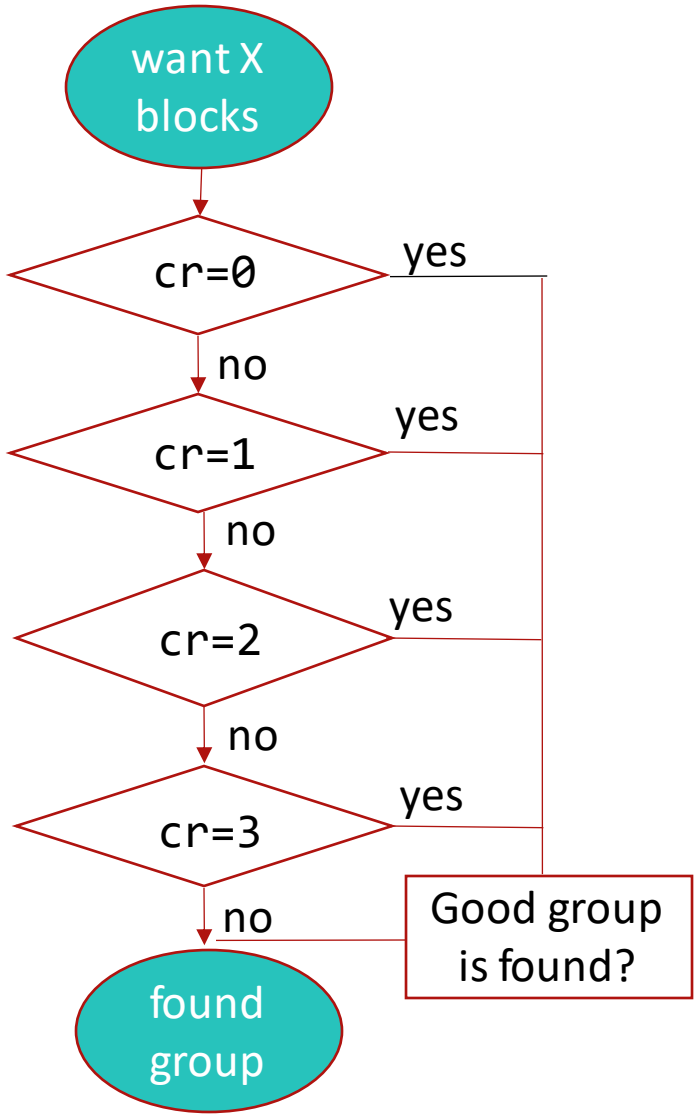
offset + requested size = 3000

Normally requested 4096, but limited to 1024

`prealloc_table` should be adjusted periodically to adjust current state



4-pass Group Selection Loops



4 loops across all block allocation groups

- `cr = 0`, want only aligned 2^n-block chunks
- `cr = 1`, average free chunk has enough blocks
- `cr = 2`, group has enough free blocks
- `cr = 3`, use any free blocks (fragment)

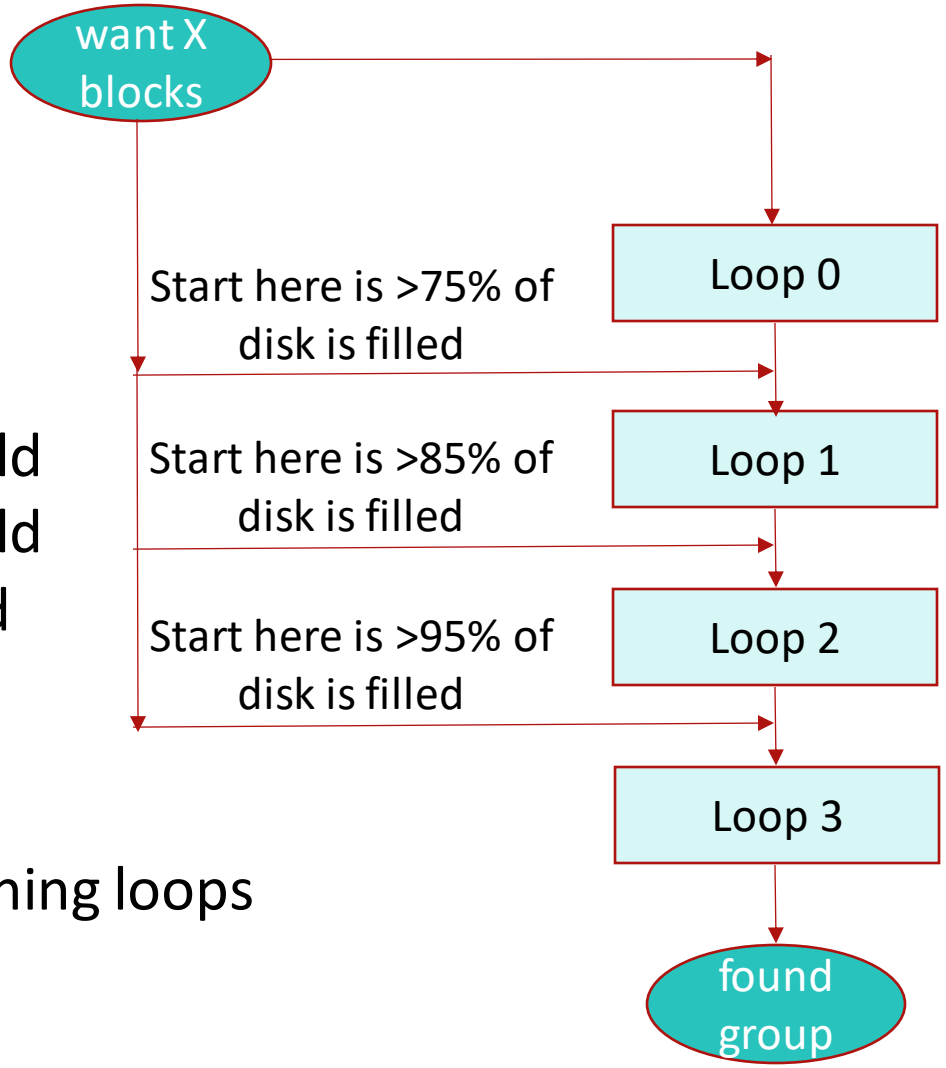
Based on the file size requested (offset + requested size) block count is rounded to the nearest large block range e.g.: (16K, 32K, 64K, 128K, 256K, 512K, 1M, 2M, 4M, 8M etc. - `prealloc_table`)

Some performance drop when a target is close to full

Loops skipping solution

Based on filesystem
fullness condition

Echo "25"> /sys/fs/ldiskfs/<dev>/mb_c1_threshold
Echo "15"> /sys/fs/ldiskfs/<dev>/mb_c2_threshold
Echo "5"> /sys/fs/ldiskfs/<dev>/mb_c3_threshold



Force mballoc to skip (likely) useless scanning loops

Set mb_cX_threshold Permanently (LU-14305)


```
# lustre/tests/llmount.sh
# cat /sys/fs/ldiskfs/loop1/mb_c1_threshold
25
# tunefs.lustre /dev/lustre-ost1
...
Parameters: mgsnode=192.168.56.102@tcp sys.timeout=20
# umount /dev/lustre-ost1
# tunefs.lustre --mountfsoptions="errors=remount-ro,mb_c1_threshold=45" /dev/lustre-ost1
...
Persistent mount opts: errors=remount-ro,mb_c1_threshold=45
Parameters: mgsnode=192.168.56.102@tcp sys.timeout=20
# mount -t lustre /dev/lustre-ost1 /mnt/lustre-ost1
# cat /sys/fs/ldiskfs/loop1/mb_c1_threshold
45
```

`lctl set_param -P` `mb_cX_threshold` support is planned LU-16040 as particular case of `/sys/fs/ldiskfs` and `/proc/fs/ldiskfs` support

How to diagnose allocator problems

Whamcloud

```
cat /proc/fs/ldiskfs/<dev>/mb_stats |
    grep useless

useless_c1_loops: 0
useless_c2_loops: 0
useless_c3_loops: 0

/sys/fs/ldiskfs/<dev>/mb_c1_threshold:15
/sys/fs/ldiskfs/<dev>/mb_c2_threshold:10
/sys/fs/ldiskfs/<dev>/mb_c3_threshold:5
```

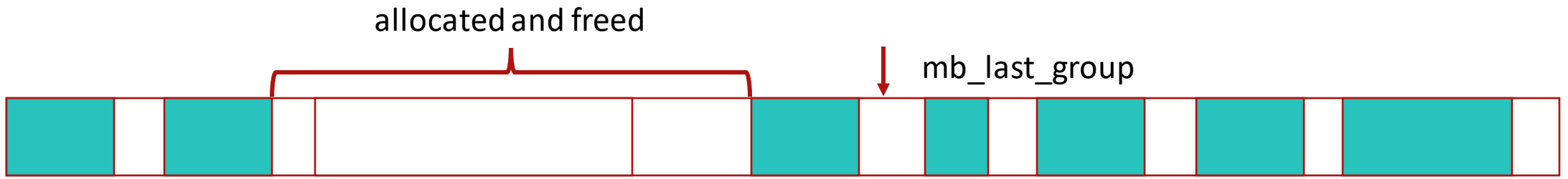
```
cat /proc/fs/ldiskfs/<dev>/mb_stats |
    grep useless

useless_c1_loops: 1343197
useless_c2_loops: 2520822
useless_c3_loops: 0

/sys/fs/ldiskfs/<dev>/mb_c1_threshold:4
/sys/fs/ldiskfs/<dev>/mb_c2_threshold:3
/sys/fs/ldiskfs/<dev>/mb_c2_threshold:1
```

- Too low free space threshold avoids heuristics
- Causes excessive scanning of full groups

It is better to allocate at the beginning of the disk

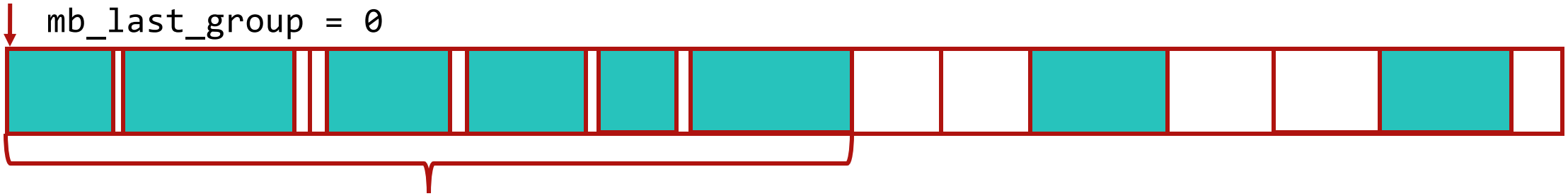
Whamcloud

- `pdsh -g oss 'cat /proc/fs/ldiskfs/*/mb_last_group' | sort`
- `obdfilter` shows 30% performance drop for OSTs with high `mb_last_group`
- Spinning hard drive is faster at the start and up to 40% slower at the end



- `echo 0 > /proc/fs/ldiskfs/*/mb_last_group`

Tuning `mb_last_group` solution



`/proc/fs/ldiskfs/*/mb_groups`

↓

Heuristic algorithm

↓

`/proc/fs/ldiskfs/*/mb_last_group`

Solution based on the new block allocator from LU-14438

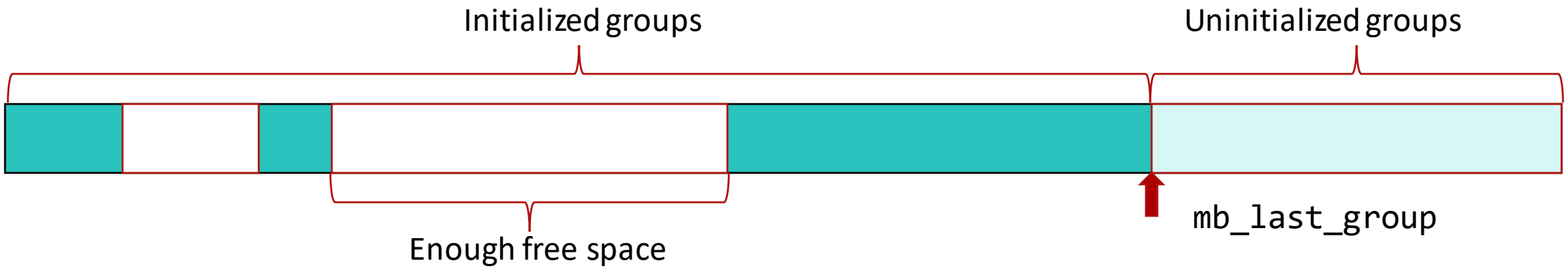
Too much metadata: slow mount (LU-12988)

Whamcloud

- mballoc maintains internal in-memory structures (**buddy cache**) to speed up searching
- cache is built from regular **on-disk bitmaps**
- if **cache is cold**, reading it may take a lot of time
- during **Lustre server** startup few small files need to be updated (e.g. config backup)
- at this point buddy/bitmap **cache is empty** but mballoc wants to find a big chunk of free space for group preallocation and reads bitmaps one by one
- sometimes this can take a **very long time**

Slow mount optimization fix (LU-15319)

LU-13291 "ldiskfs: mballoc don't skip uninit-on-disk groups"
suggests to skip initialized groups at cr < 2 rather than read them into memory



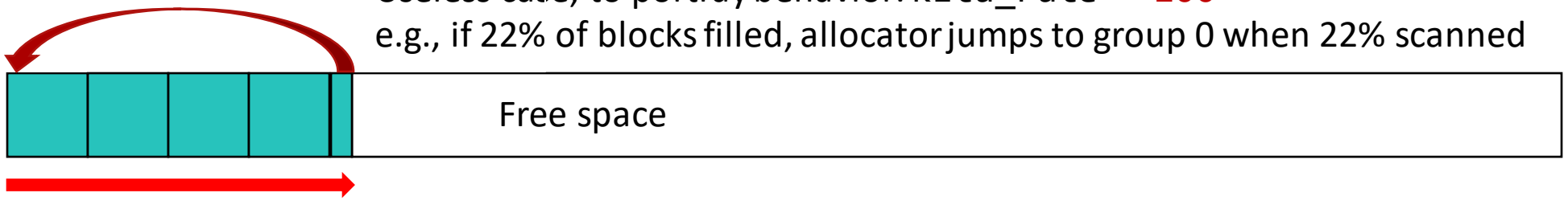
Separate thread uploads buddy data at the same time, but do it behind cr0 loop at the mount

```
if (cr < 2 && !ext4_mb_uninit_on_disk(ac->ac_sb, group))
                return 0;
        ret = ext4_mb_init_group(ac->ac_sb, group, GFP_NOFS);
```

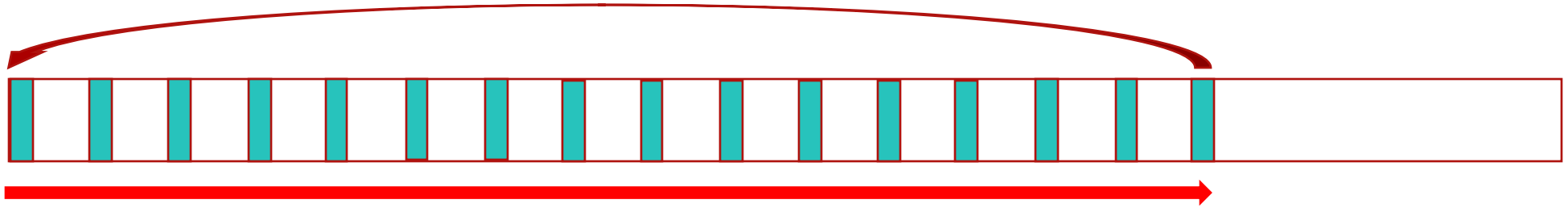
After the first allocation `mb_last_group` continues from this place that is too far from the beginning.

mb_last_group heuristics (LU-16162)

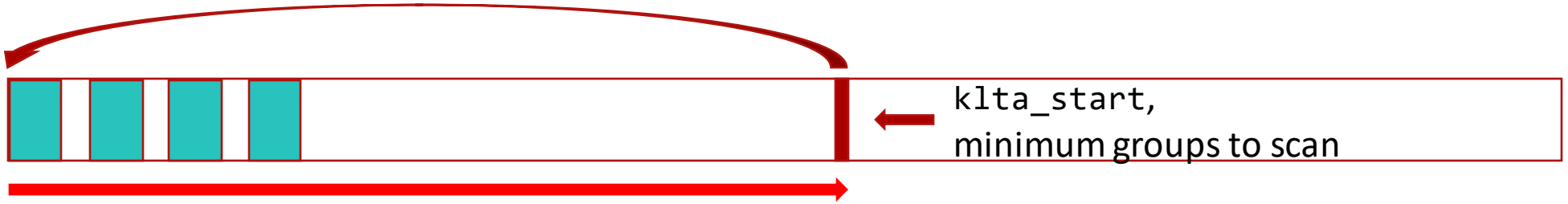
Useless case, to portray behavior: `klta_rate = 100`
e.g., if 22% of blocks filled, allocator jumps to group 0 when 22% scanned



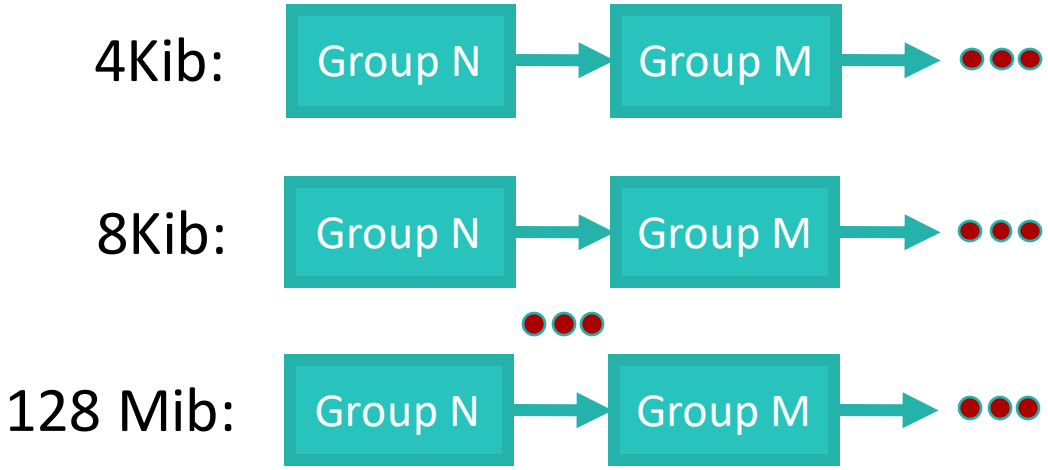
Default configuration: `klta_rate = 25`
If 22% of blocks filled, allocator jumps to group 0 when 88% of groups scanned

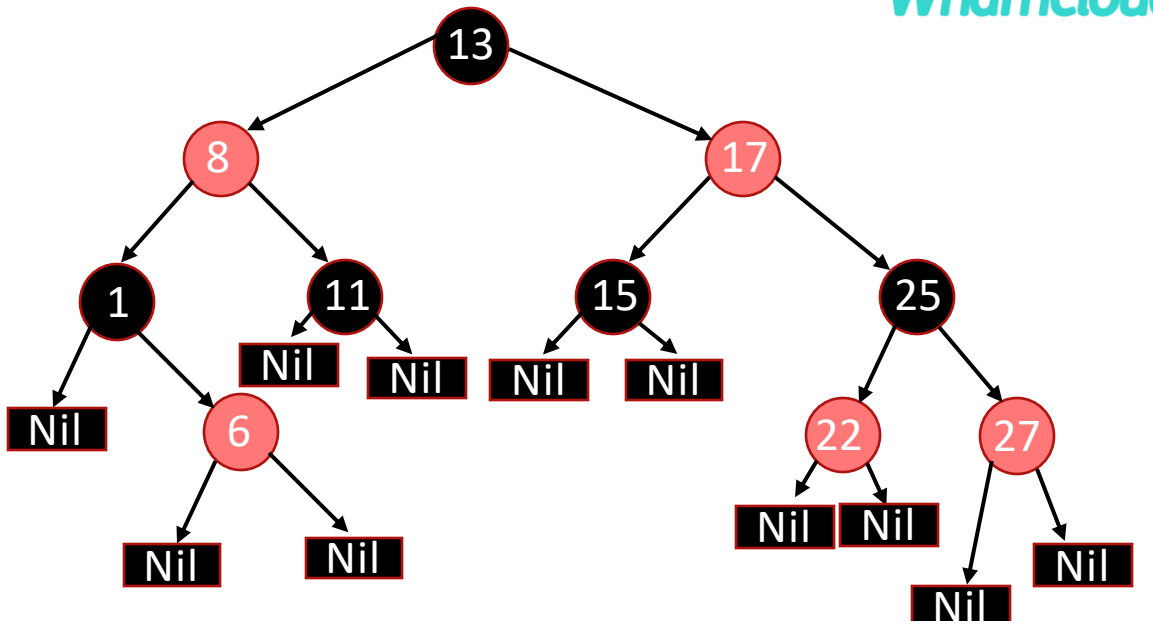


Example with `klta_start`: `klta_rate = 25`, 10% of blocks filled
If chosen group < `klta_start`, then do not jump to group 0 until klta_start hit



The New One Ext4 block allocator: `mb_optimize_scan`





for `cr0` there is a list for each order of free chunks in group for O(1) lookup

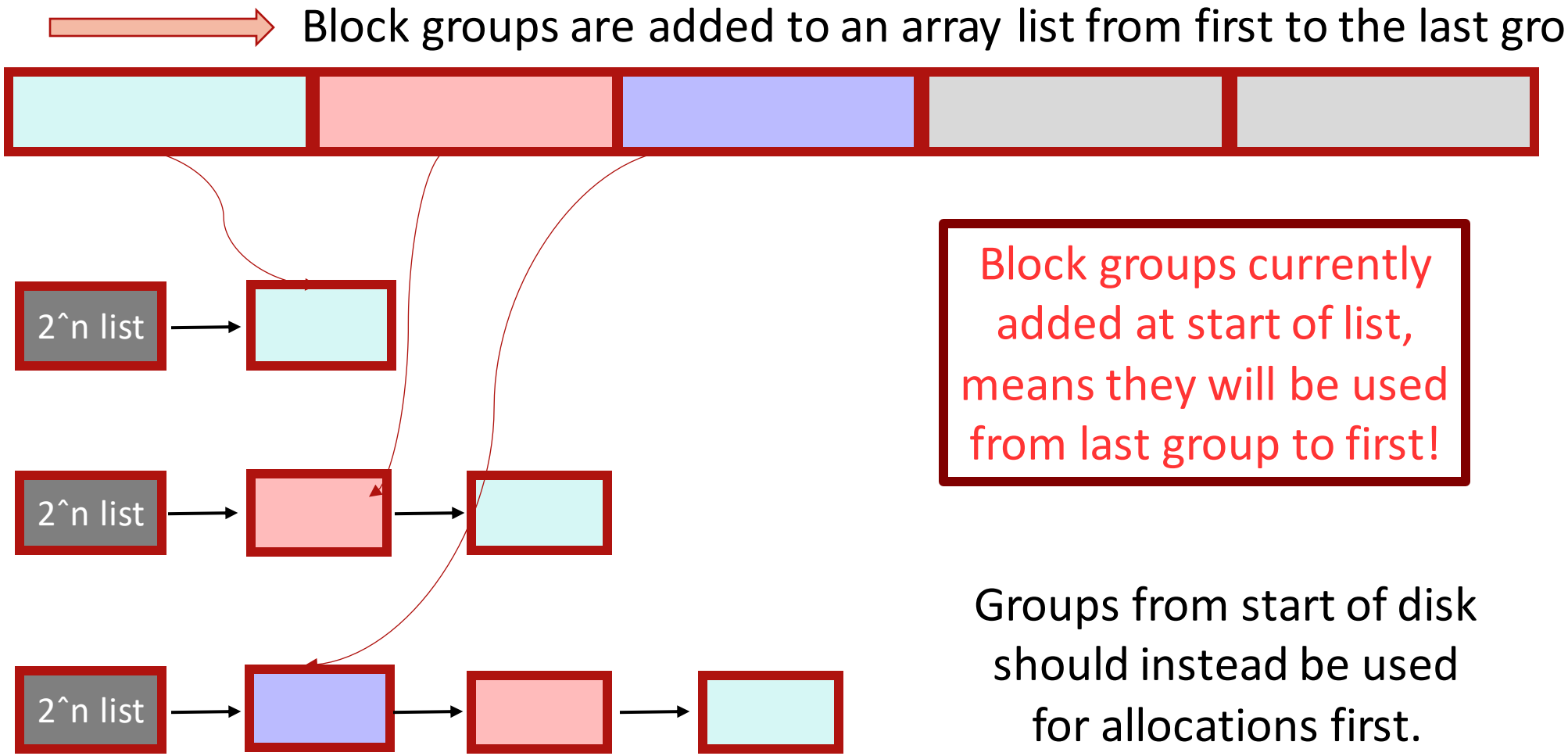
for `cr1` currently is a red-black tree of groups sorted by average free blocks size for O(log2) lookup

LU-14438

https://lwn.net/Articles/849511/

Free space used at end of disk and this space is too slow

Whamcloud



Block groups currently added at start of list, means they will be used from last group to first!

Groups from start of disk should instead be used for allocations first.

Files too spread across disk with `mb_optimized_scan`

"The new allocator strategy spreads allocations over more block groups, we end up with more open erase blocks on the SD card which forces the firmware to do more garbage collection and RMW of erase blocks and write performance tanks..."
- `linux-ext4` posting

Upstream ext4 patches underway to fix `mb_optimize_scan`:

- Make mballoc try inode local group first even with `mb_optimize_scan`
- Avoid unnecessary spreading of allocations among groups
- Array of groups sorted by average fragment size for `cr2` instead of rbtree

Proposals to further improve performance:
ext4: Multiple arrays or skip lists for groups to bias allocations to start of disk

Work in progress

The new Ext4 block allocation algorithm after bug fixing looks good and will be ported to Lustre FS LDiskFS backend soon.

Whamcloud

Thank you!

ablagodarenko@whamcloud.com

ddn