Space maps in Ext4

Saurabh Kadekodi (saurabhkadekodi@gmail.com)
Shweta Jain (atewhs.jain@gmail.com)

One liner...

To design and implement an extent based free space management technique for Ext4 filesystem - ***Space Maps***; and an allocator, which uses space maps to manage the allocation and deallocation of disk blocks.

Scenario

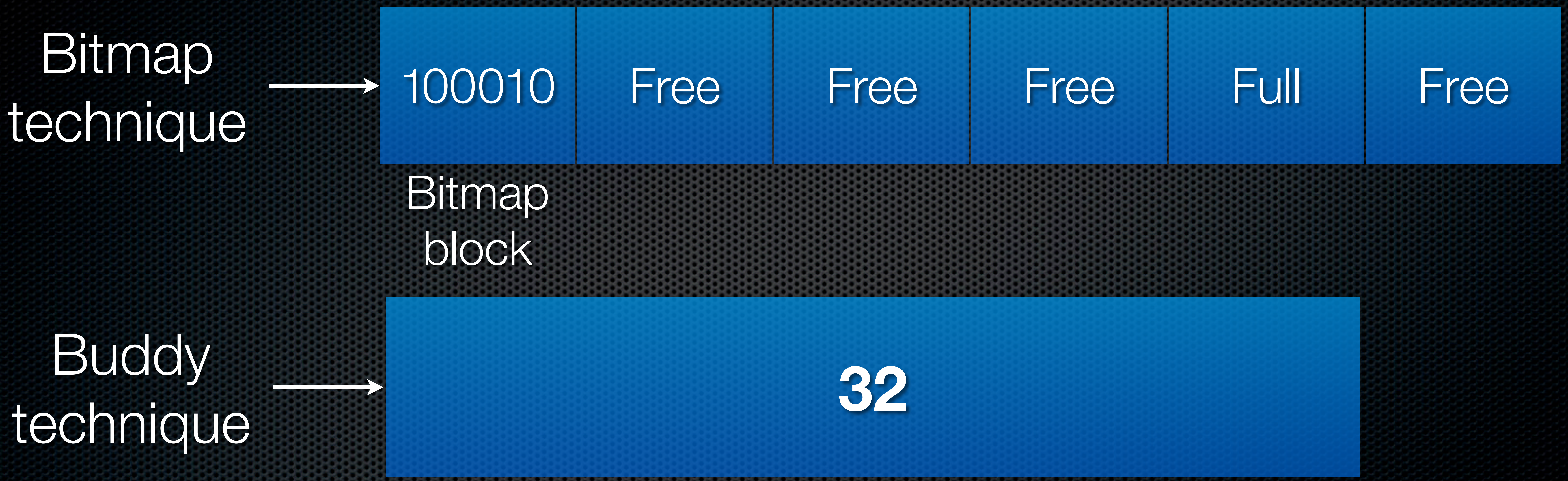

Process issues write call
VFS directs call to Ext4
Ext4 write call is invoked
Bitmaps used to find free space
Low level I/O operations

Scenario

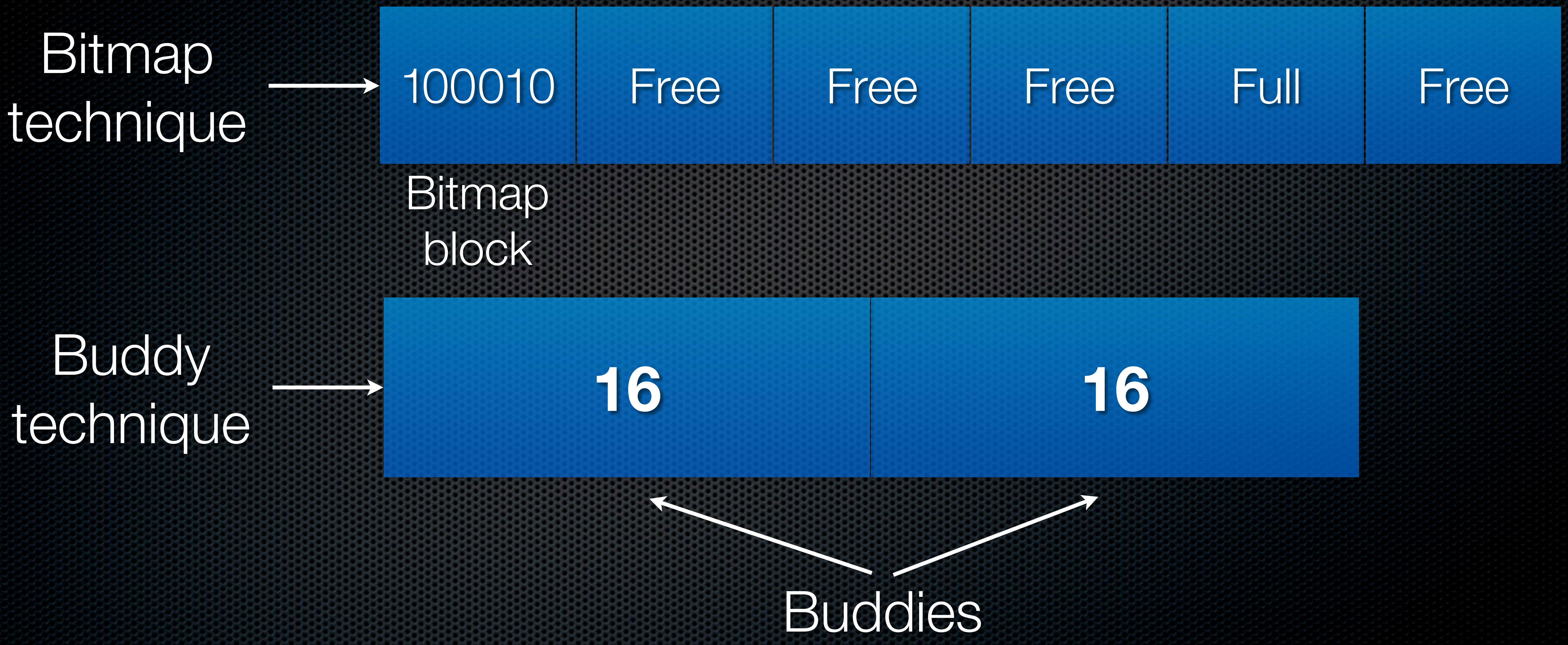
Process issues write call → VFS directs call to Ext4 → Ext4 write call is invoked → Space maps used to find free space → Low level I/O operations

Current Technique

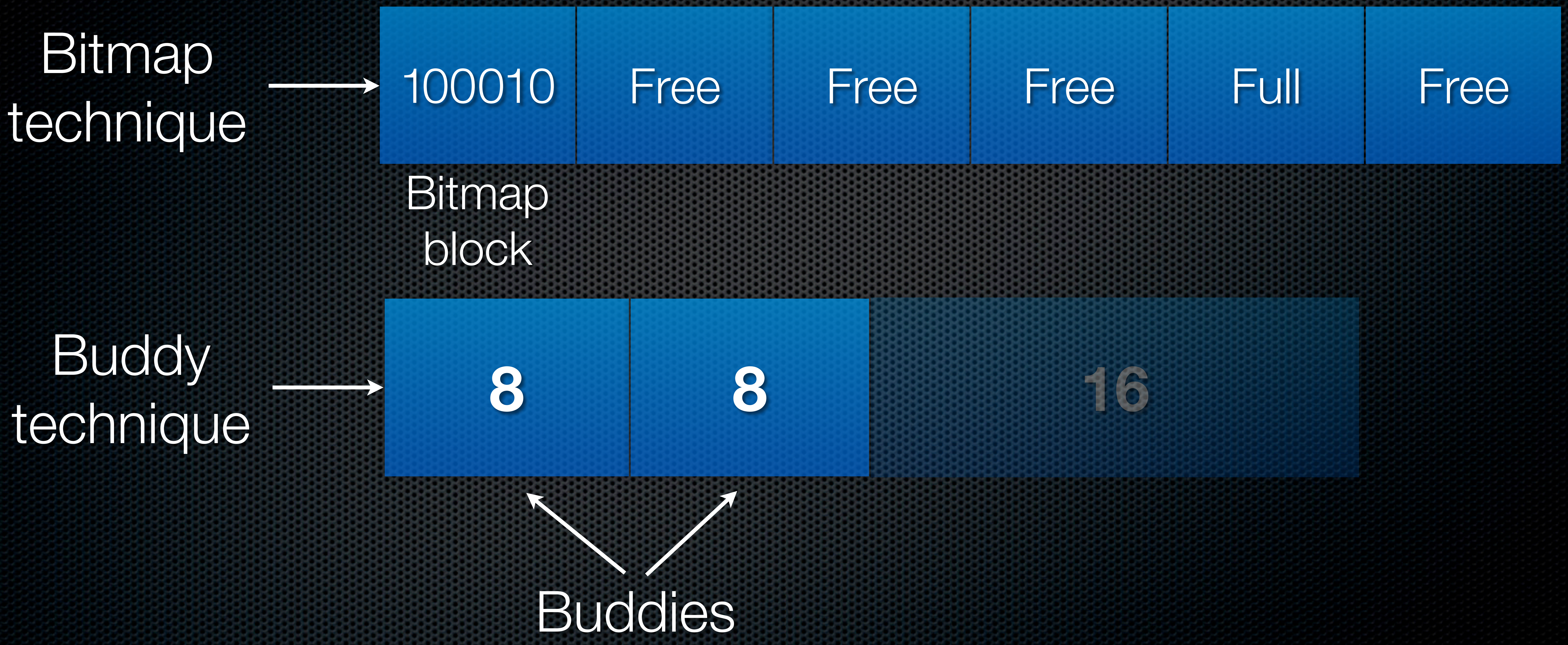
Bitmap + Buddy



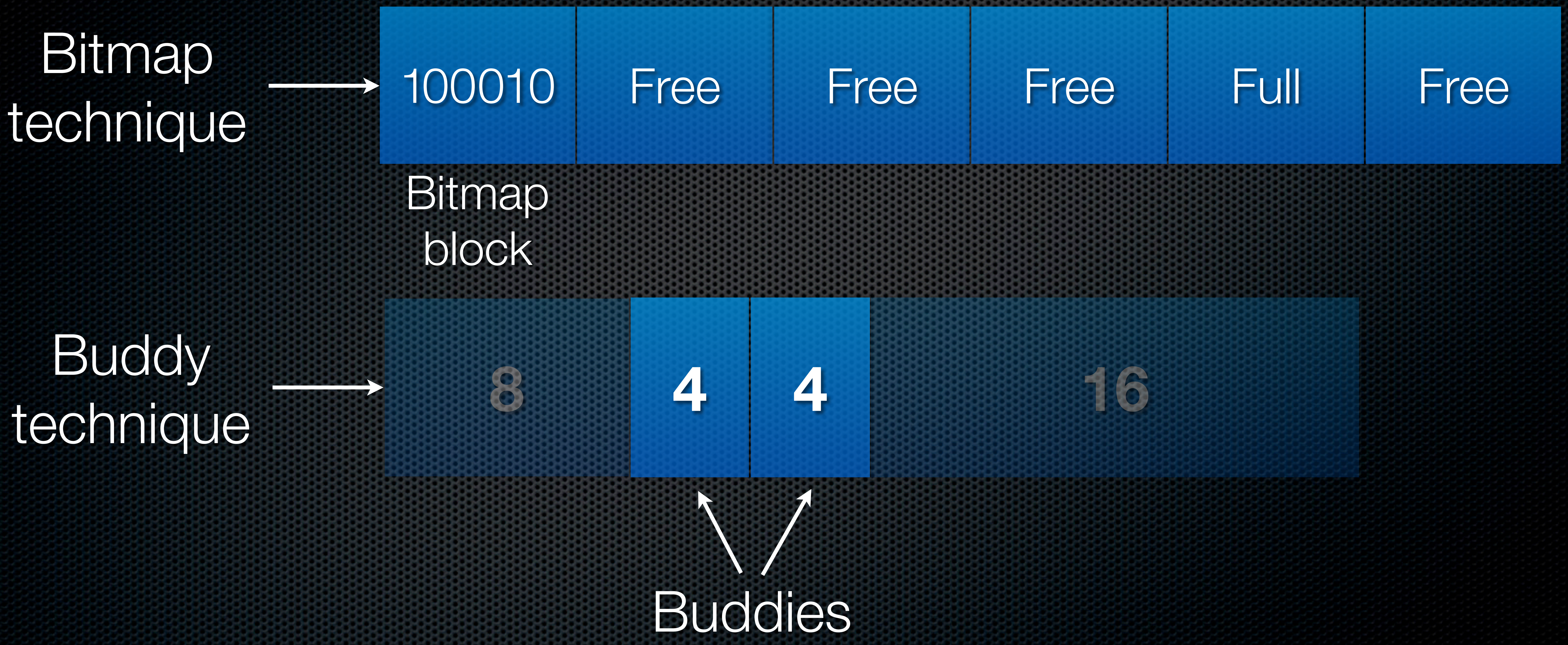
Bitmap + Buddy



Bitmap + Buddy



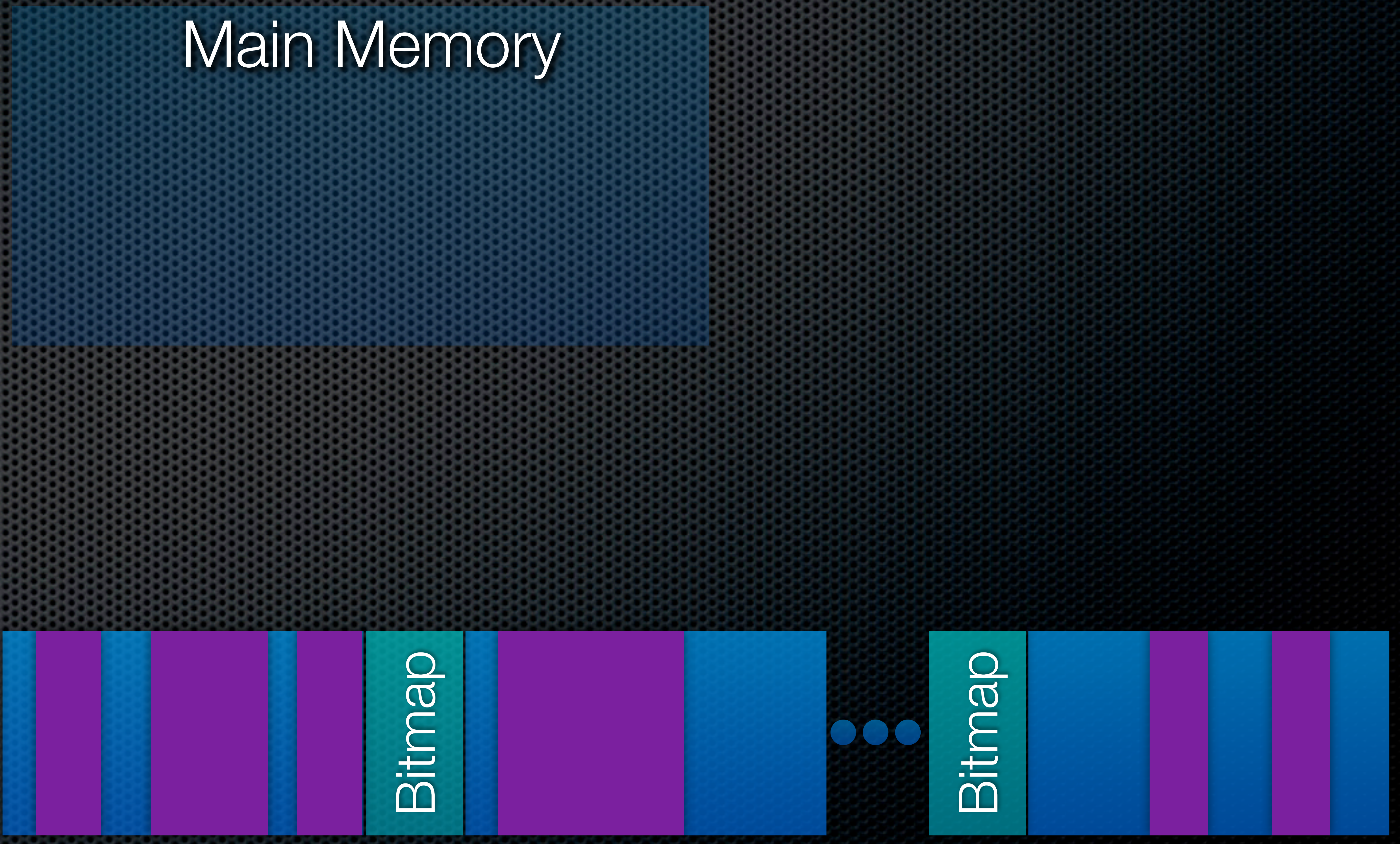
Bitmap + Buddy



Bitmap + Buddy

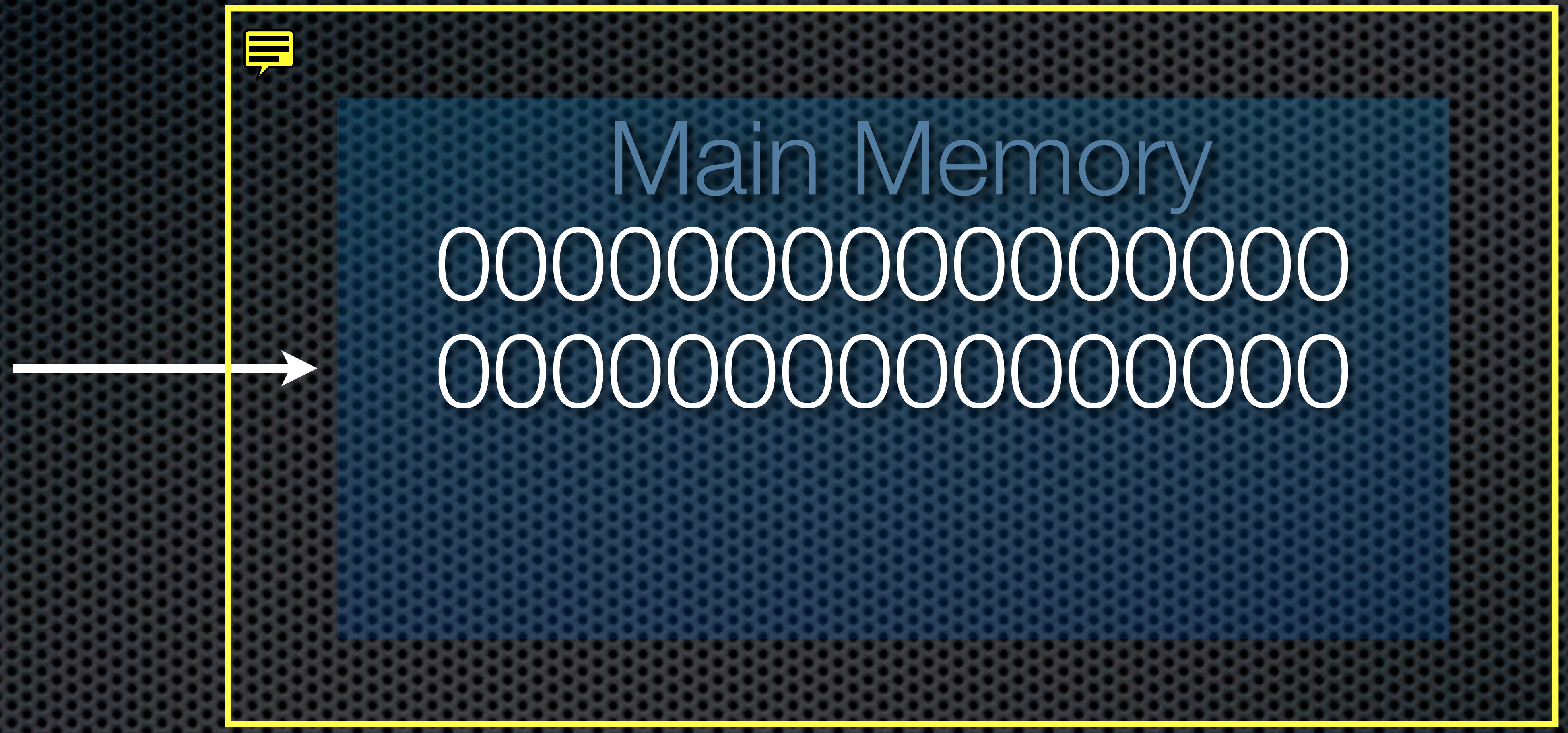
Disadvantages of current mechanism

Process issues call to delete purple directory



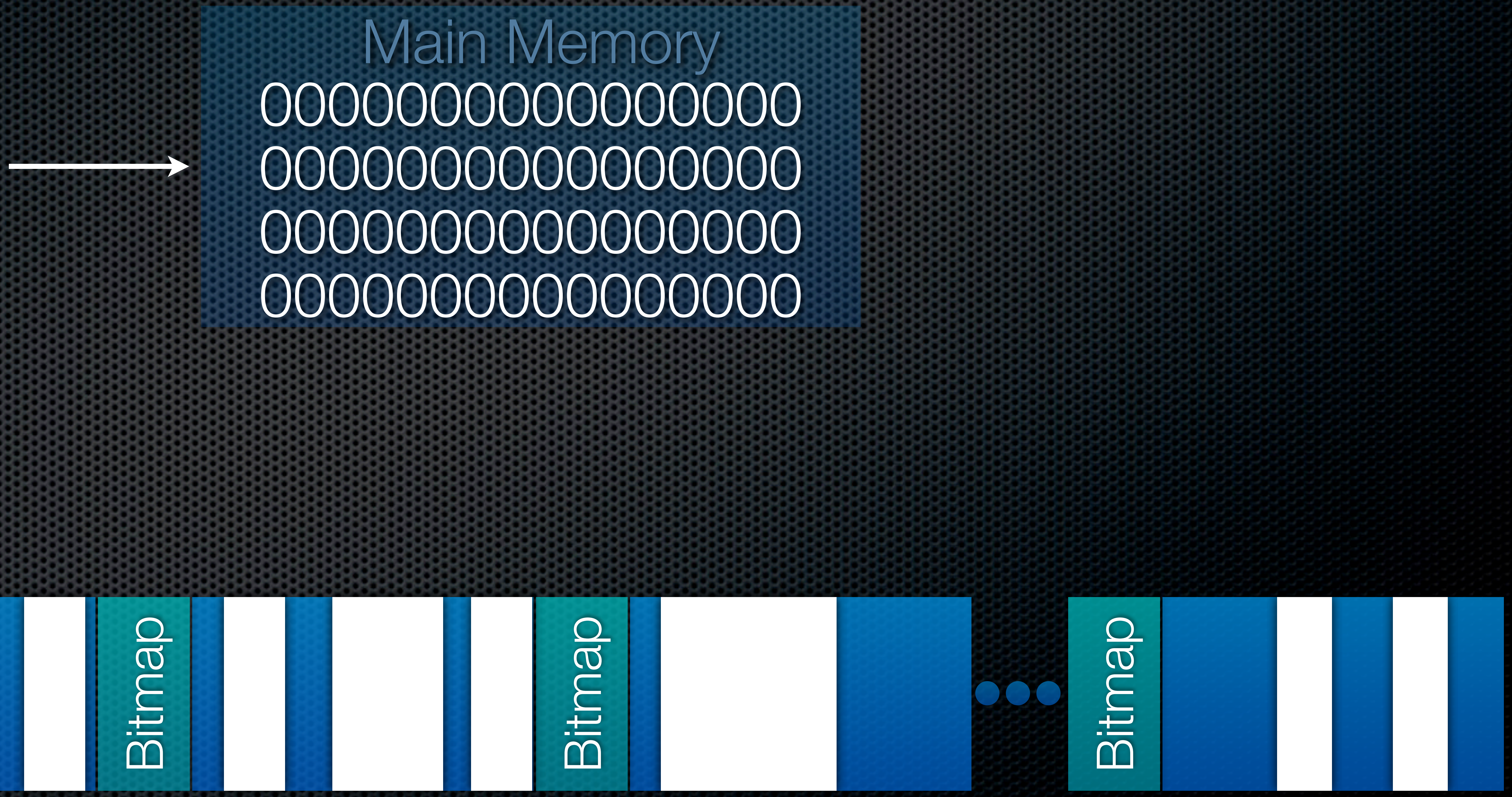
Disadvantages of current mechanism

Process issues call to delete purple directory



Disadvantages of current mechanism

Process issues call to delete purple directory



- Seeking is the bottleneck; not the in-memory processing

Disadvantages of current mechanism

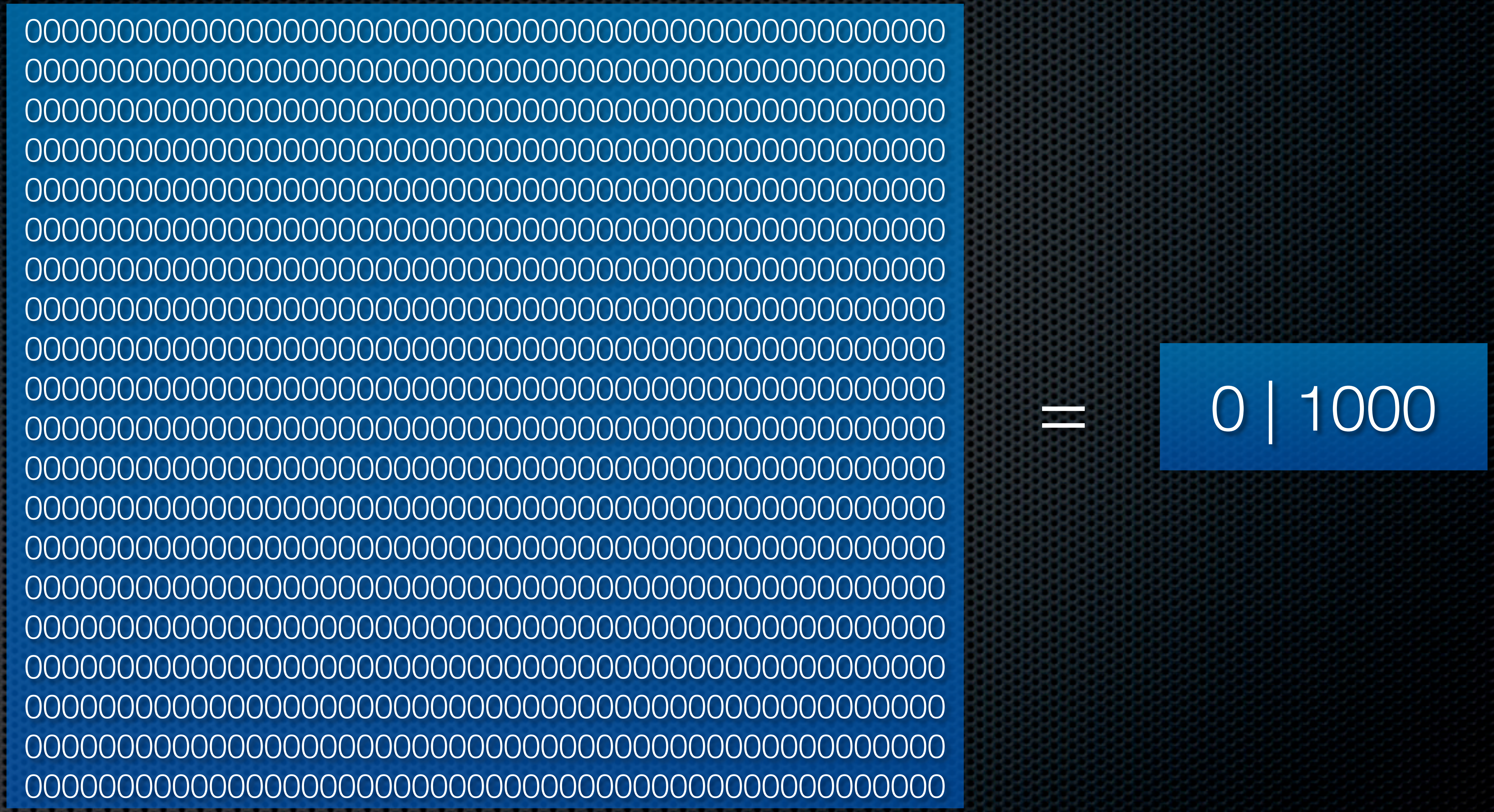
- Bitmaps scale linearly with the filesystem size

  1 PB = 32 GB bitmaps

- Bitmaps themselves consume a lot of memory; now buddy is also involved
- Cost to construct buddy in memory
- Alignment to powers of 2 leads to internal fragmentation
- In case of an aged filesystem, as contiguous free space is scarce, we have to rely on inefficient bitmap lookup

Extents

```
struct extent
{
    int start_block;
    int no_of_blocks;
}
```



Extents in Ext4

- Indirect block mapping in inode
- Preallocation

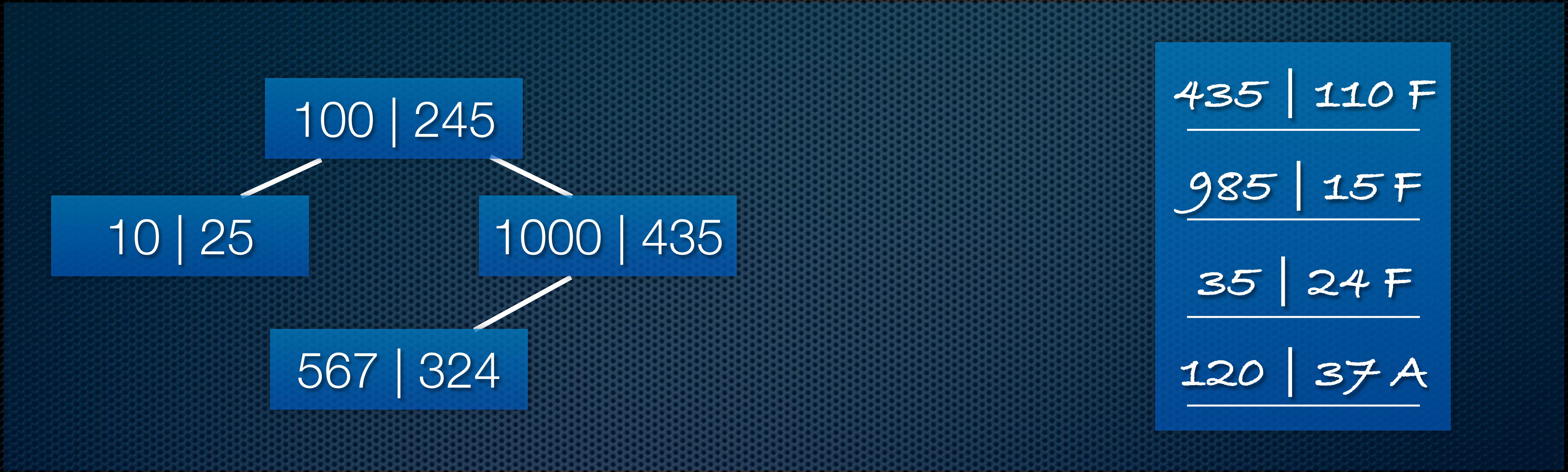
Space maps overview

Space Map

Space maps overview

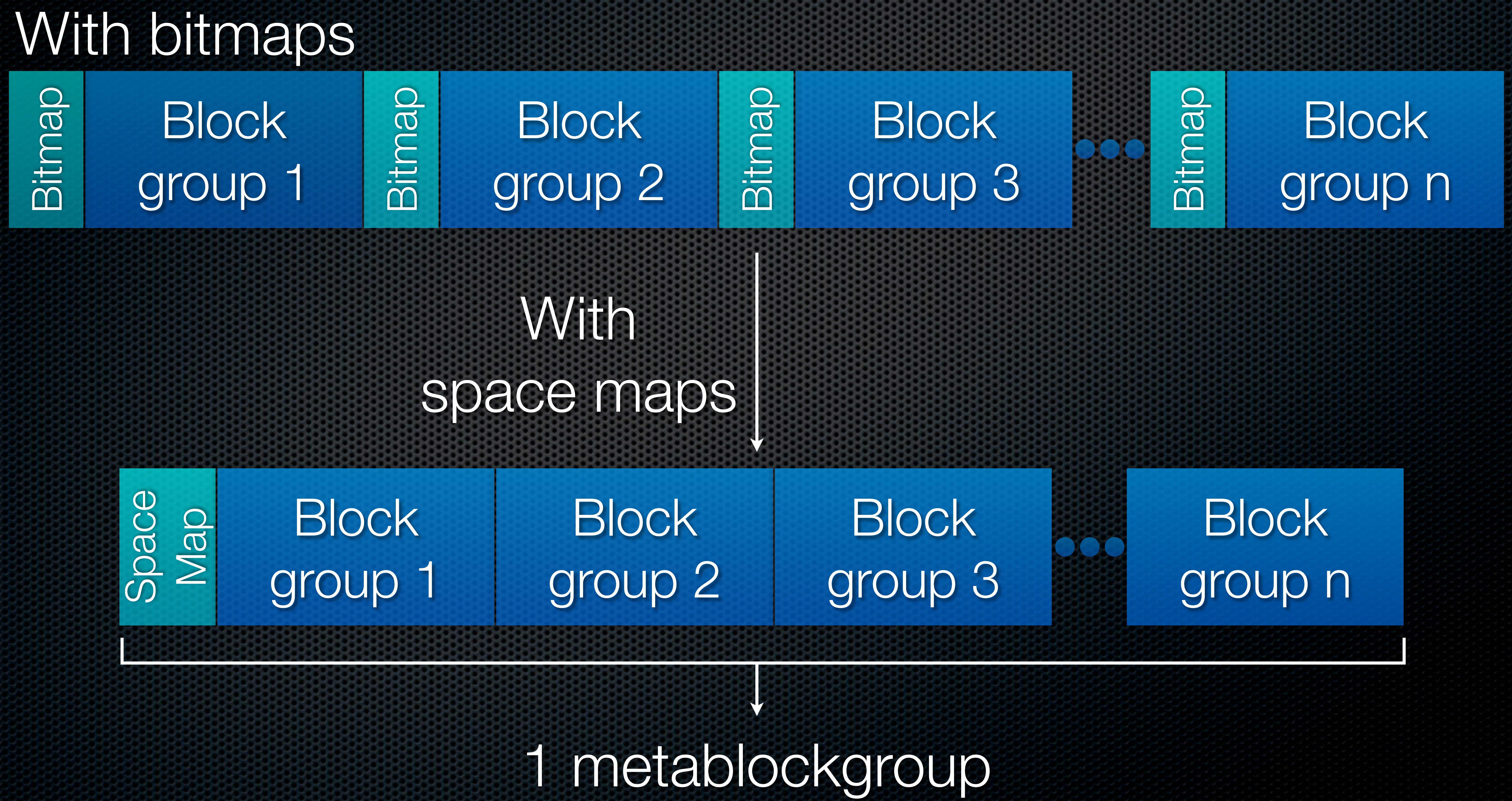
RB Tree + Log

Space maps overview

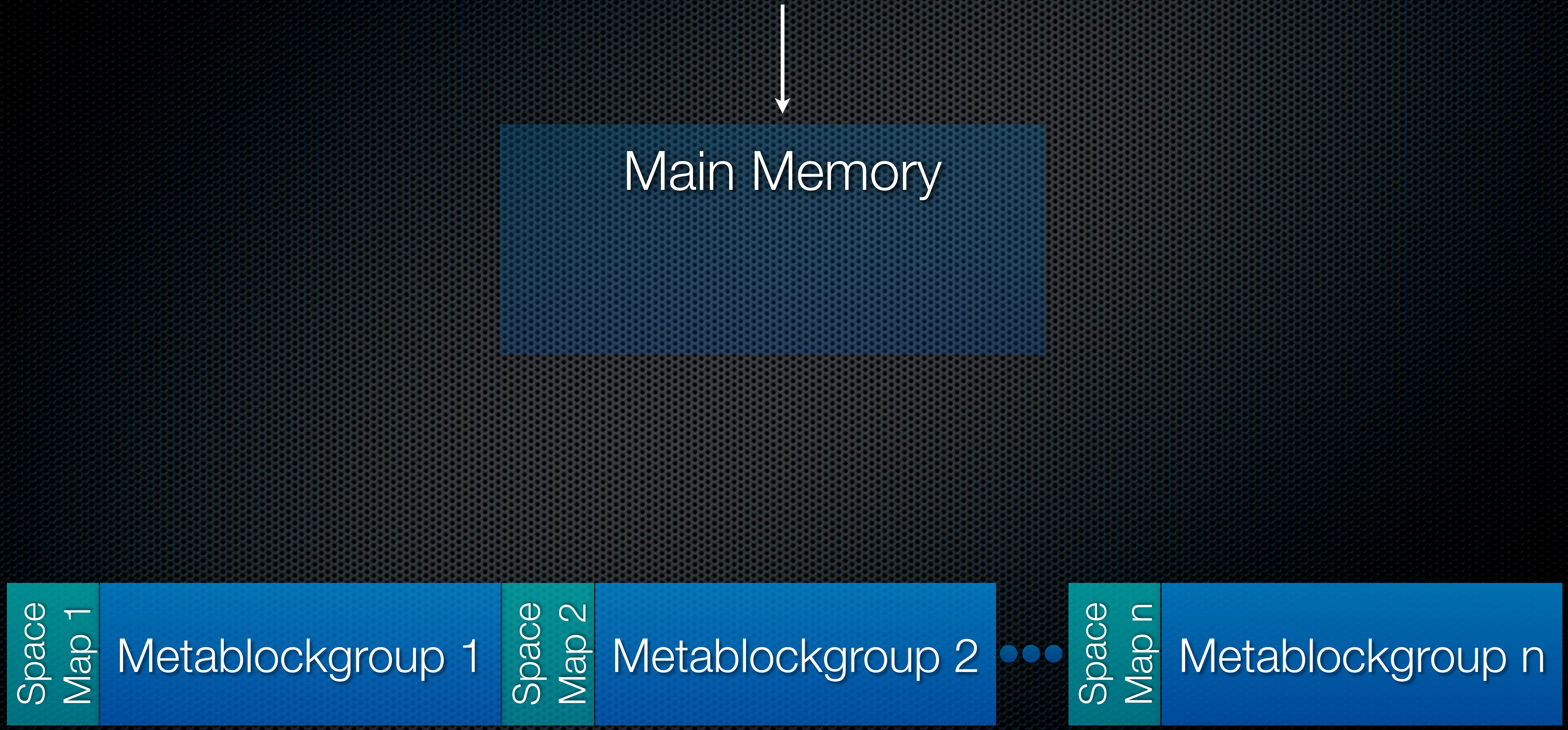


- RB Tree - A red black tree having extents of free space, sorted by offset.
- Log - A scratchpad maintaining the recent allocations and frees.

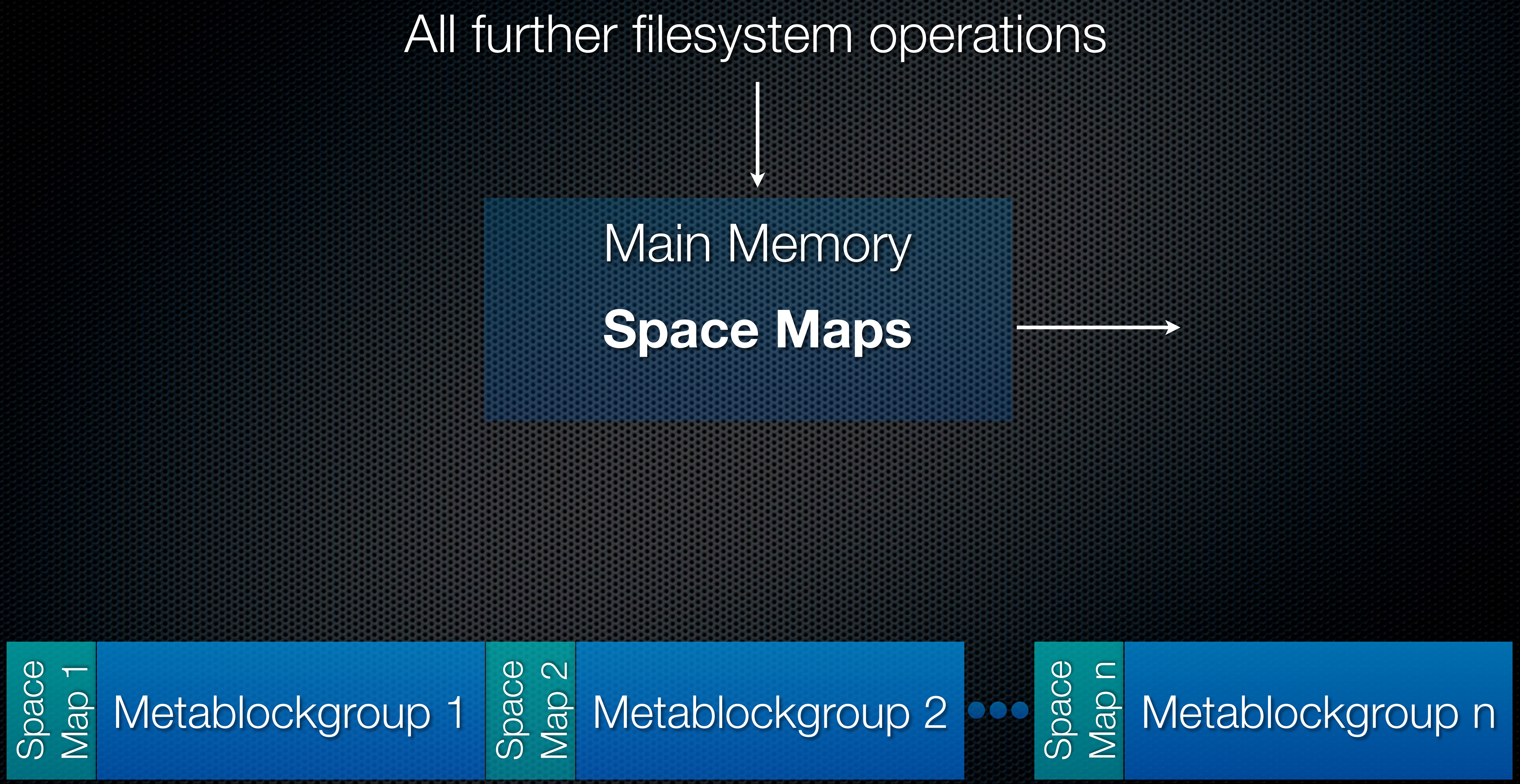
Metablockgroup



Mounting the filesystem




Main Memory
Space Maps
Space Map 1
Metablockgroup 1
Space Map 2
Metablockgroup 2
Space Map n
Metablockgroup n


All further filesystem operations
Main Memory
Space Maps
Space Map 1
Metablockgroup 1
Space Map 2
Metablockgroup 2
Space Map n
Metablockgroup n

Design Details

Structural details

In memory

RB tree node

Start block number	Length	rb_node
4 bytes	4 bytes	12 bytes

Log entry

Start block number	Length	Flags
4 bytes	4 bytes	1 byte

On disk

Tree node

Start block number	Length
4 bytes	4 bytes

Log node

Start block number	Length	Flags
4 bytes	4 bytes	1 byte

Working

When the filesystem is newly made

In memory log

Empty

In memory tree

0 | 5000

On disk log

Empty

On disk tree

0 | 5000

Allocation request of 200 blocks from offset 600

In memory log

600|200 A

In memory tree

0 | 5000

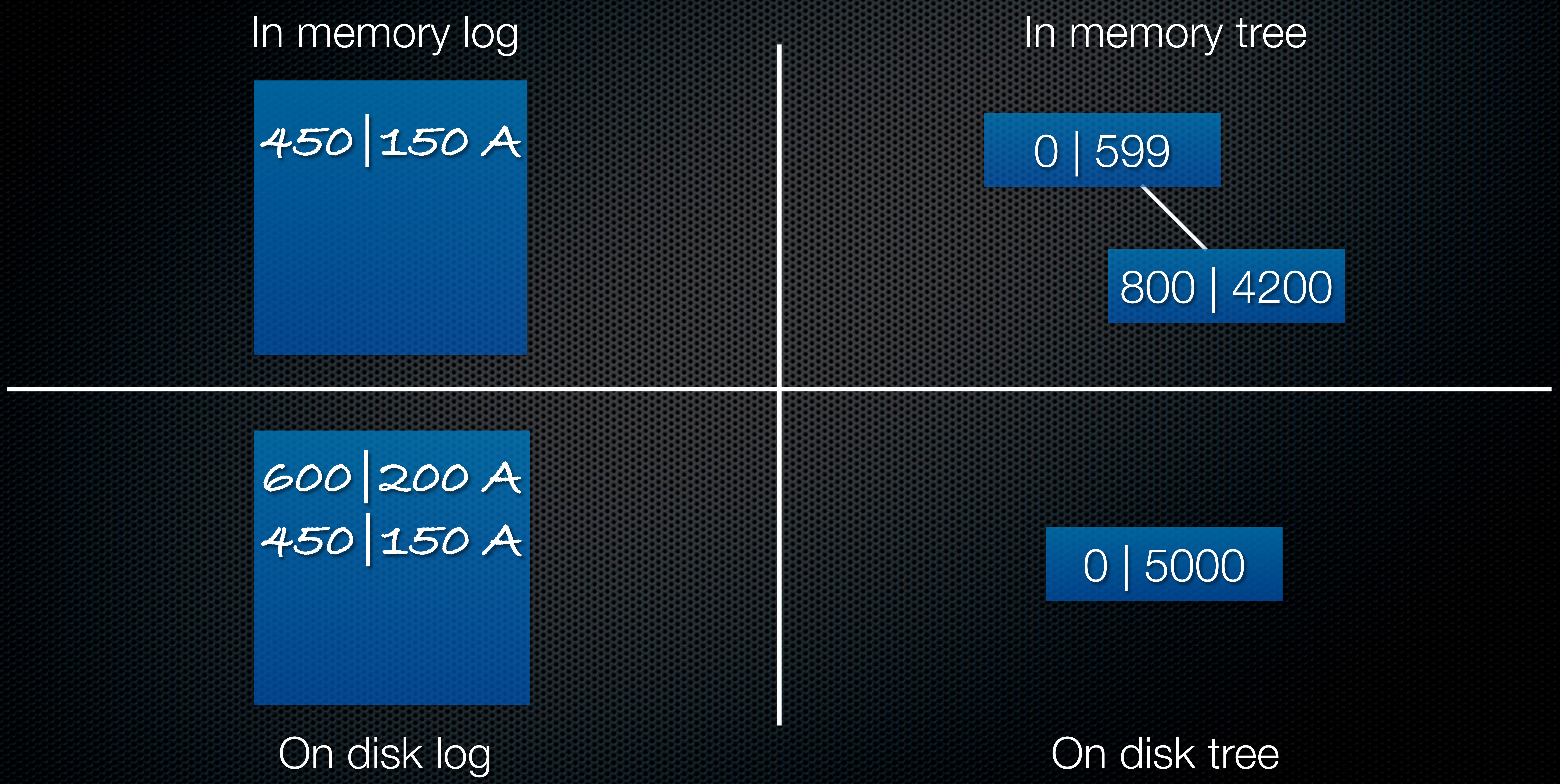
600|200 A

On disk log

0 | 5000

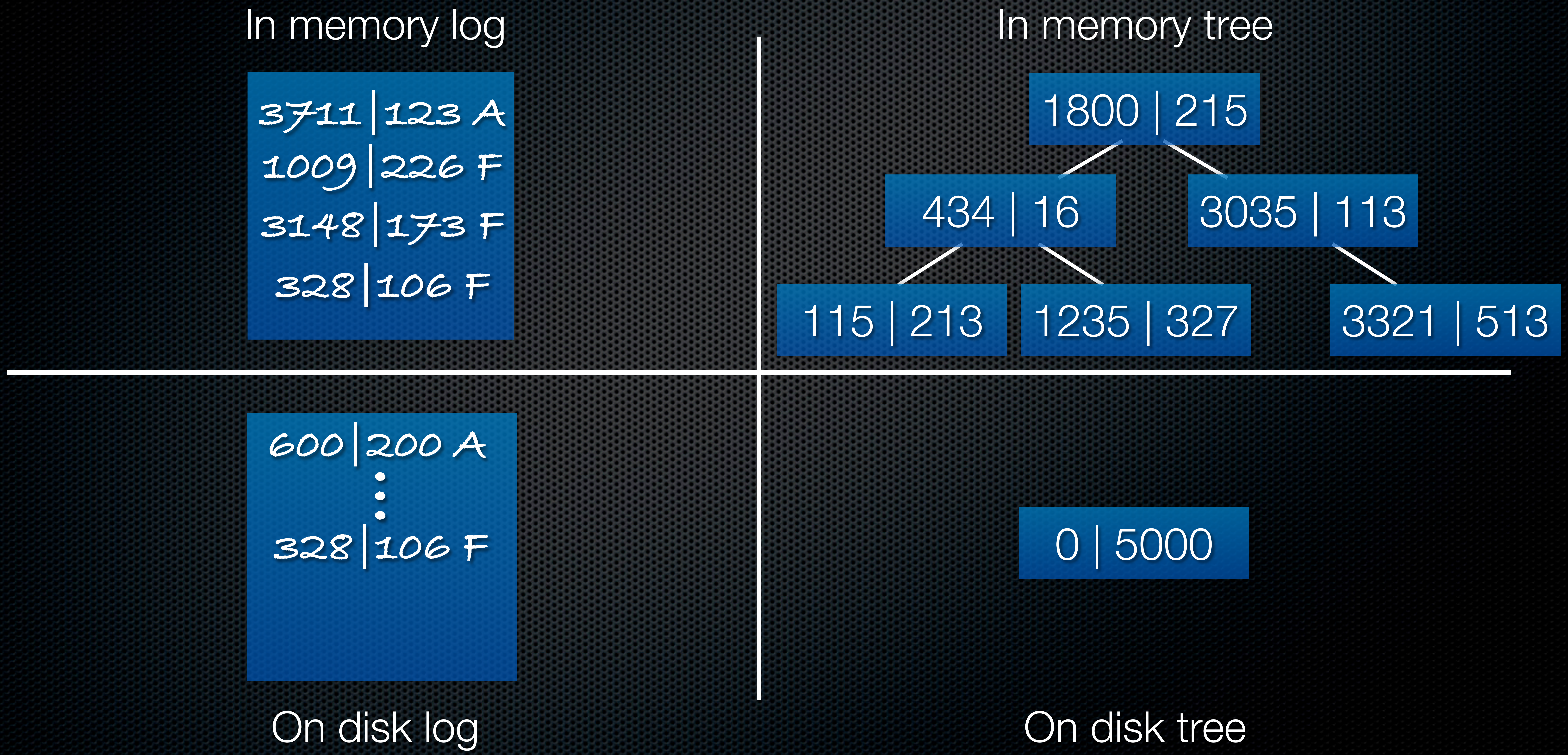
On disk tree

Allocation request of 150 blocks from offset 450

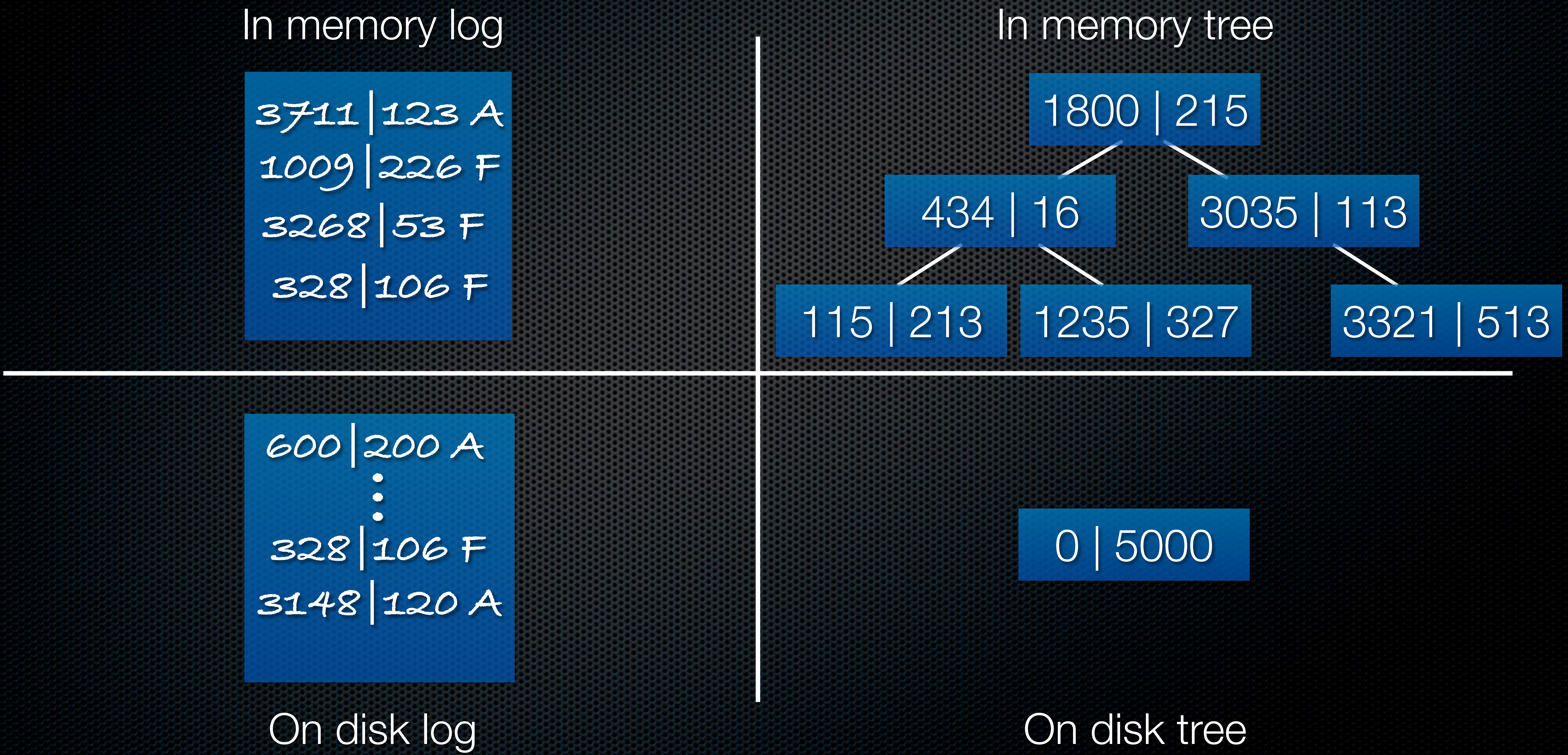


Free request of 150 blocks from offset 650

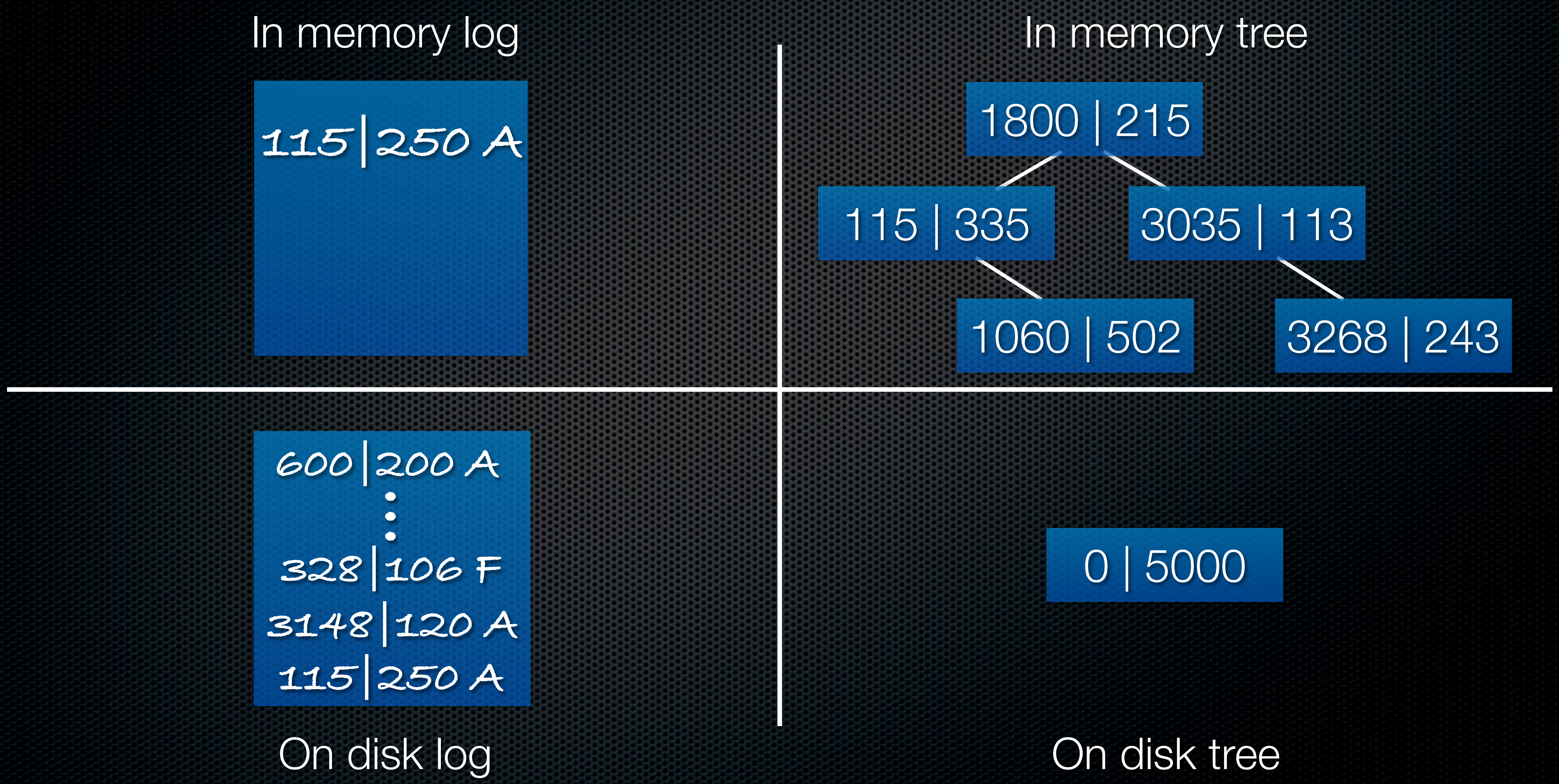
After a few filesystem operations



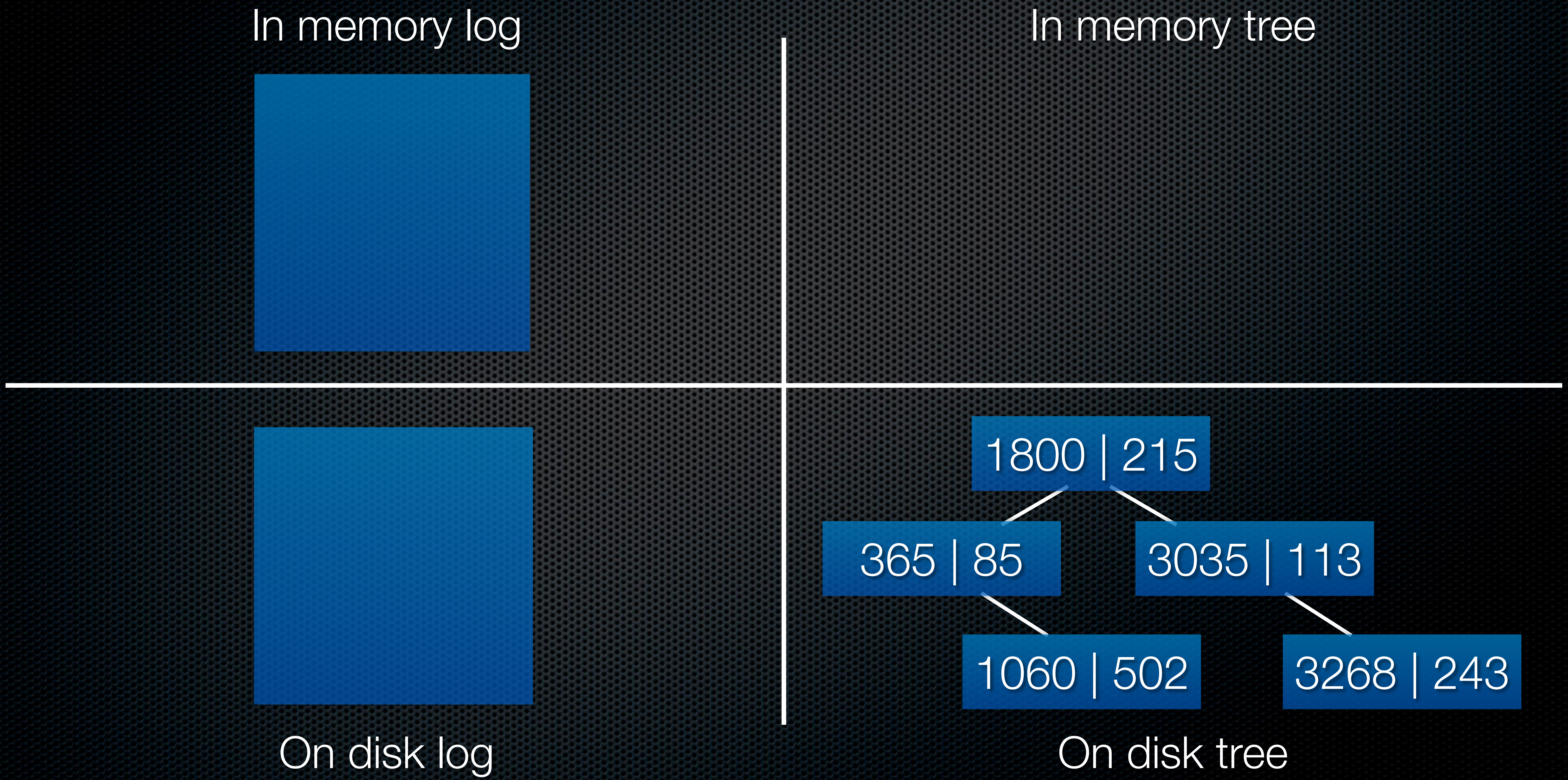
Allocation request of 120 blocks



Allocation request of 250 blocks from offset 115



While unmounting



Evaluation

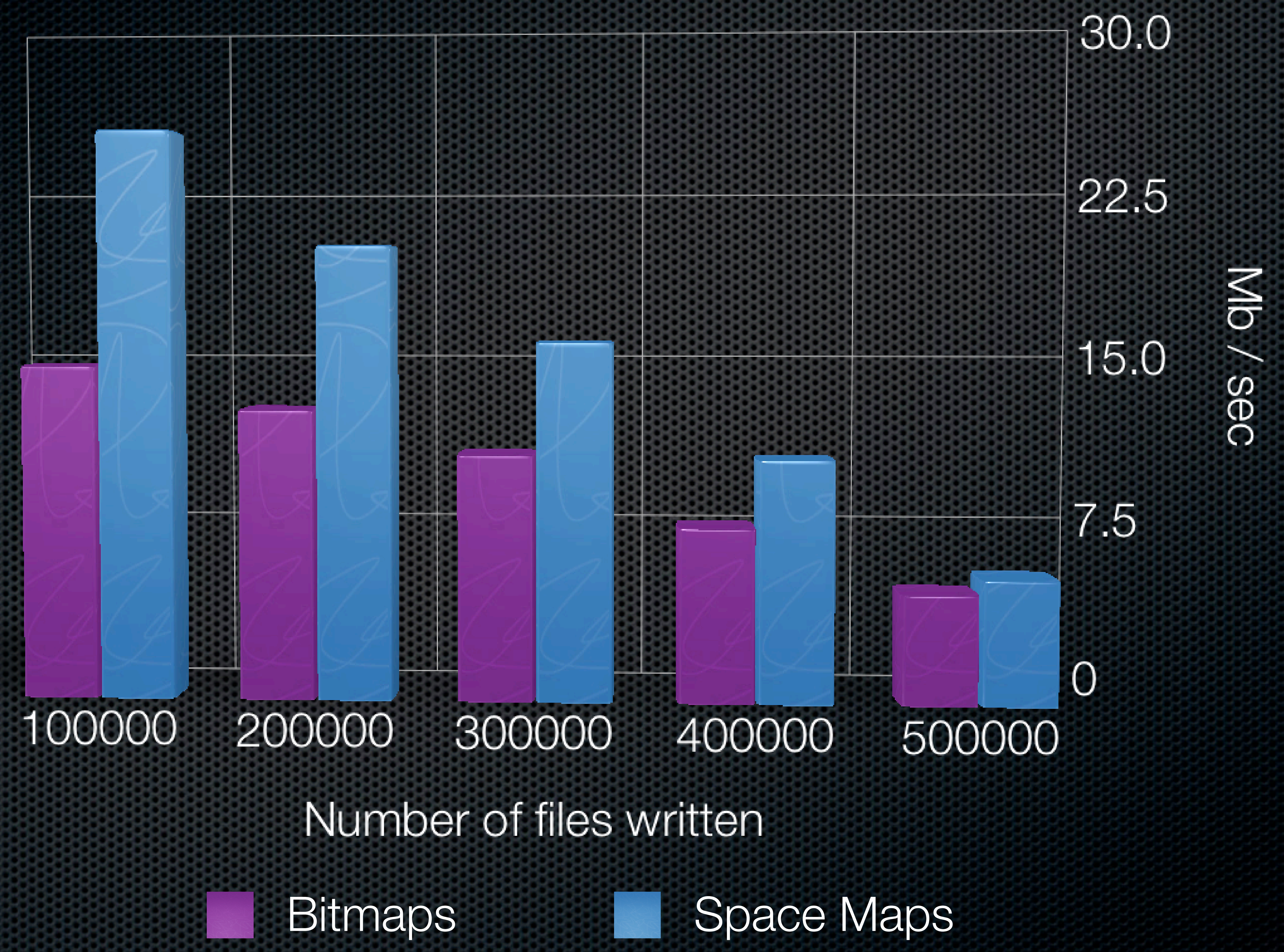
Testing Environment

- Kernel version 2.6.33.2
- Partition size 50 GB
- 4 GB RAM reduced to 384 MB to prevent excessive caching
- 1K block size to increase number of bitmaps
- Intel Core2Duo processor (2.9 GHz)

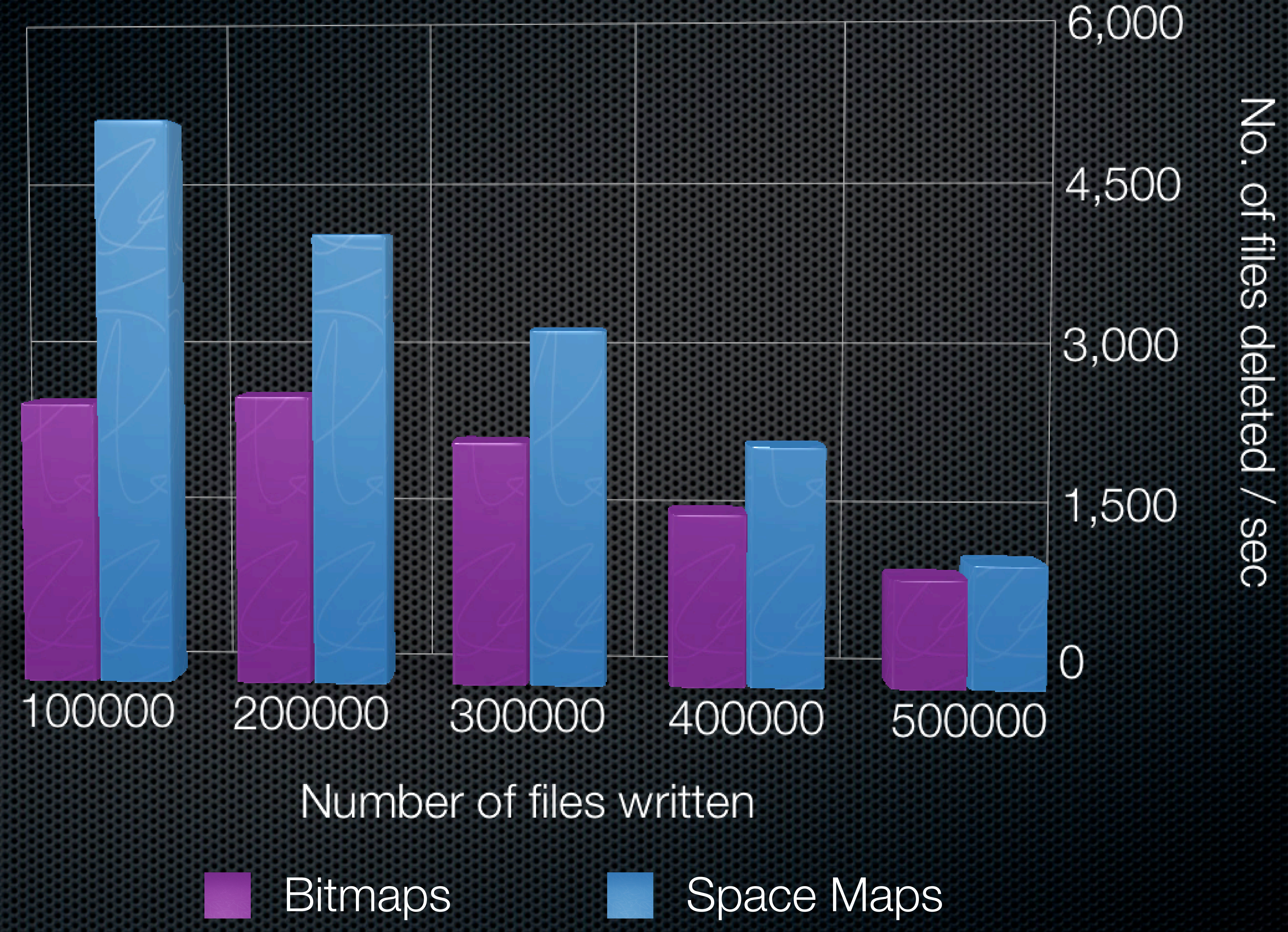
1. Small file writes

Small file operations;
A mail-server like workload using *postmark*

1. Small file writes



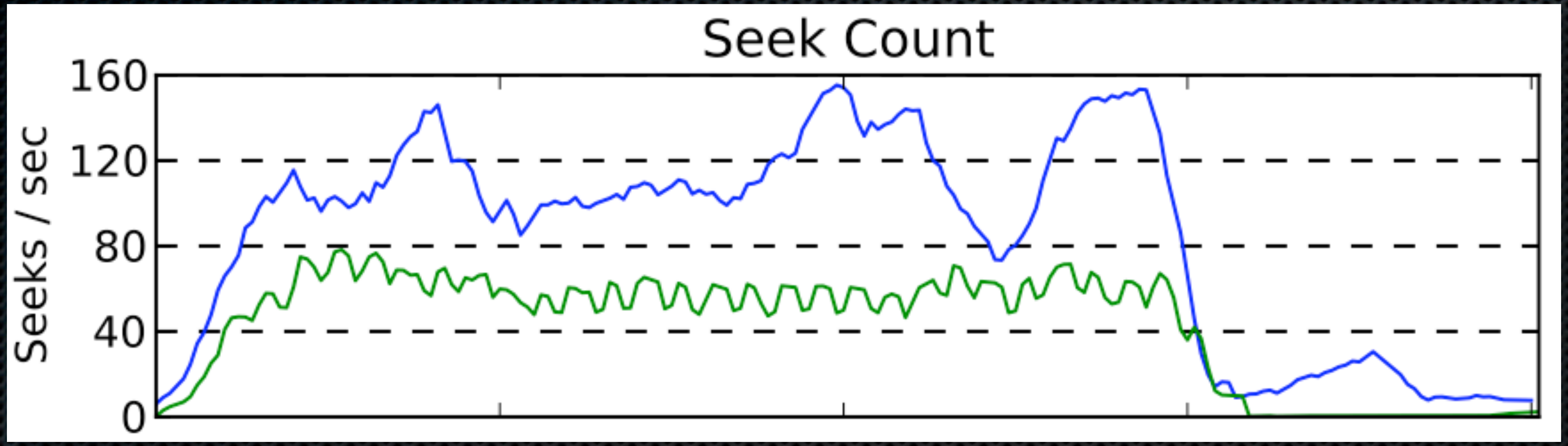
2. Small file deletes

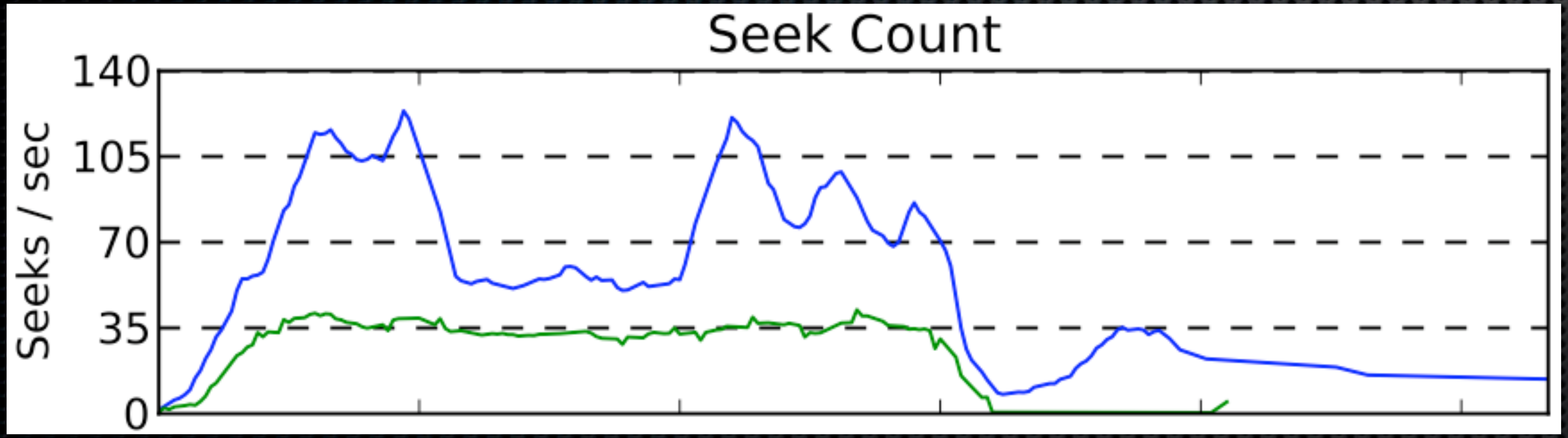


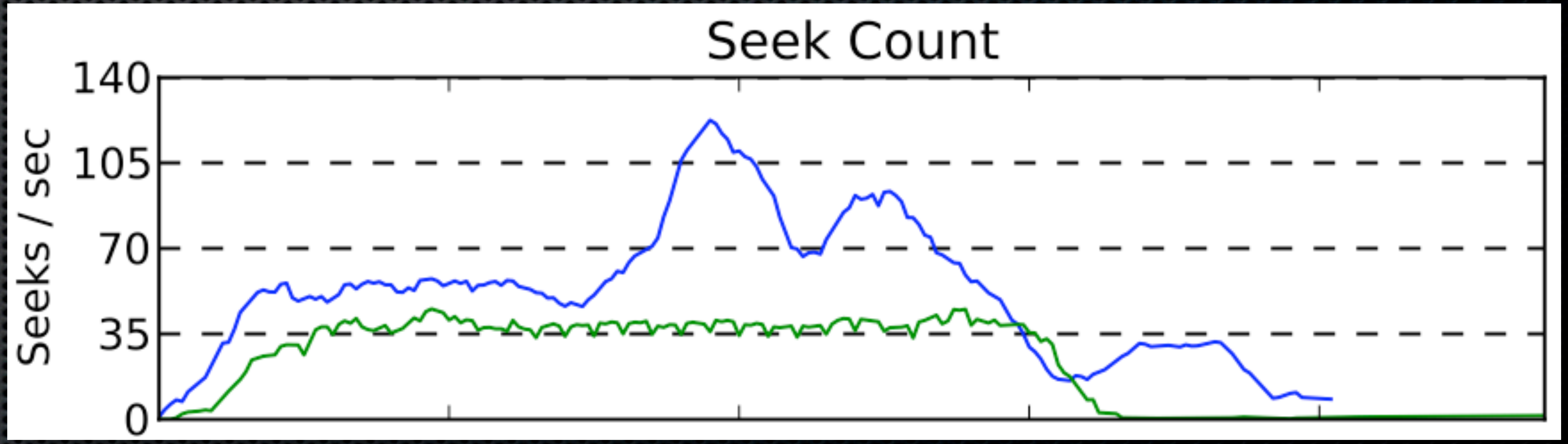
Seekwatcher graphs

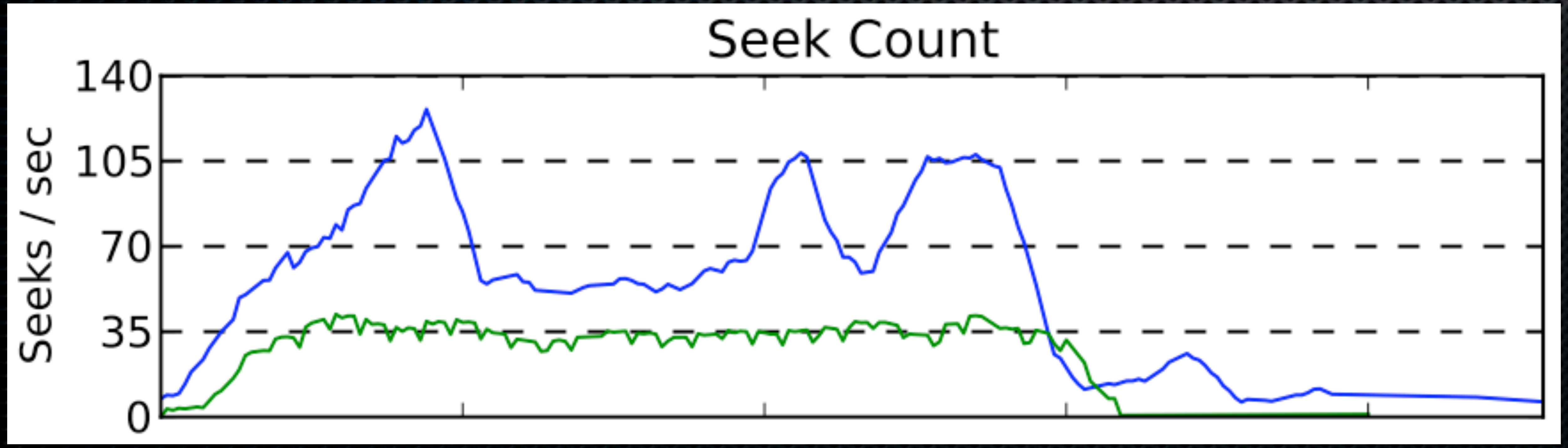
Simultaneous large file-
small file creation

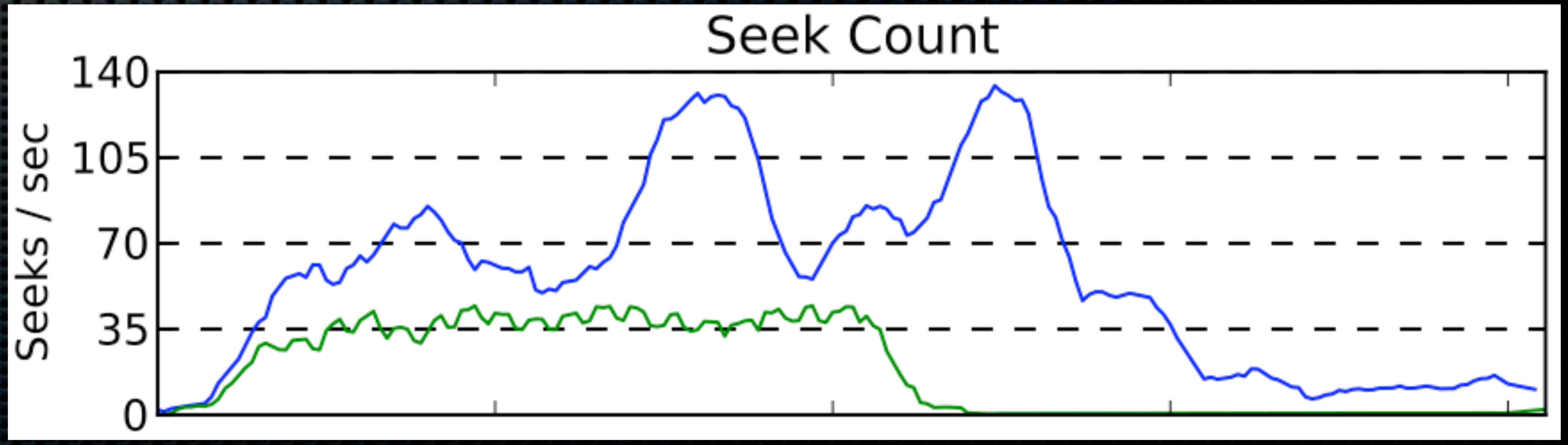
Seekwatcher graphs



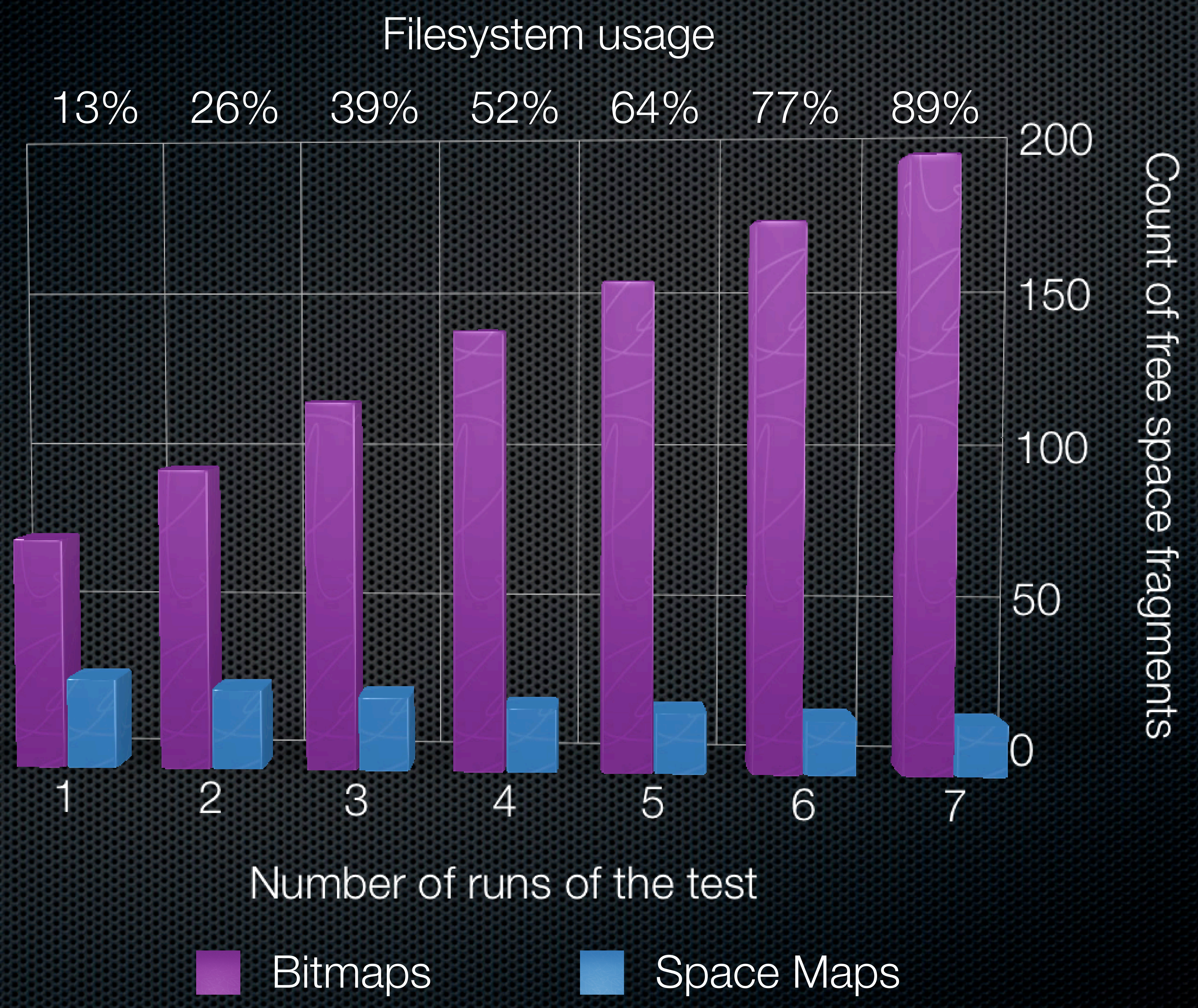




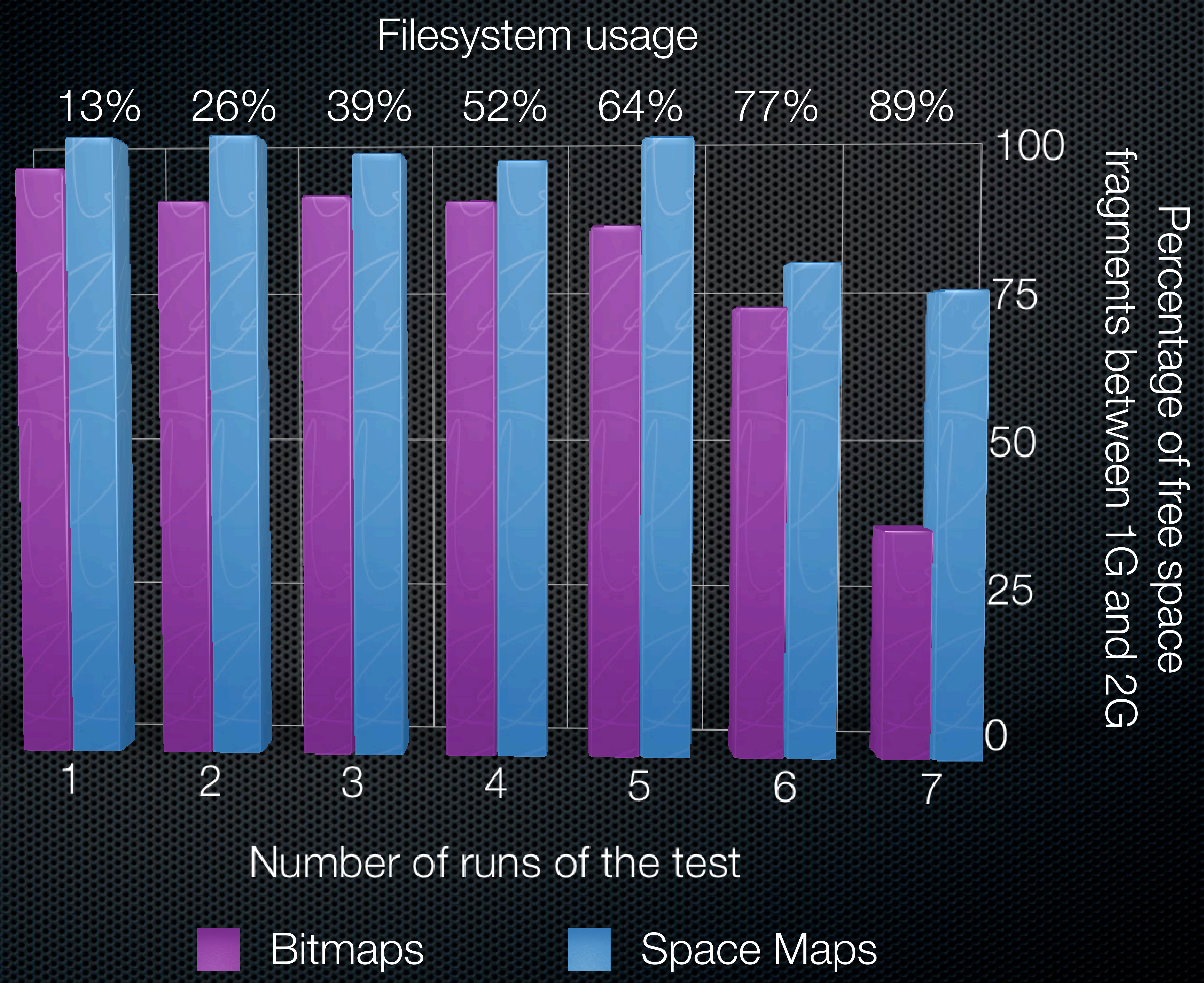




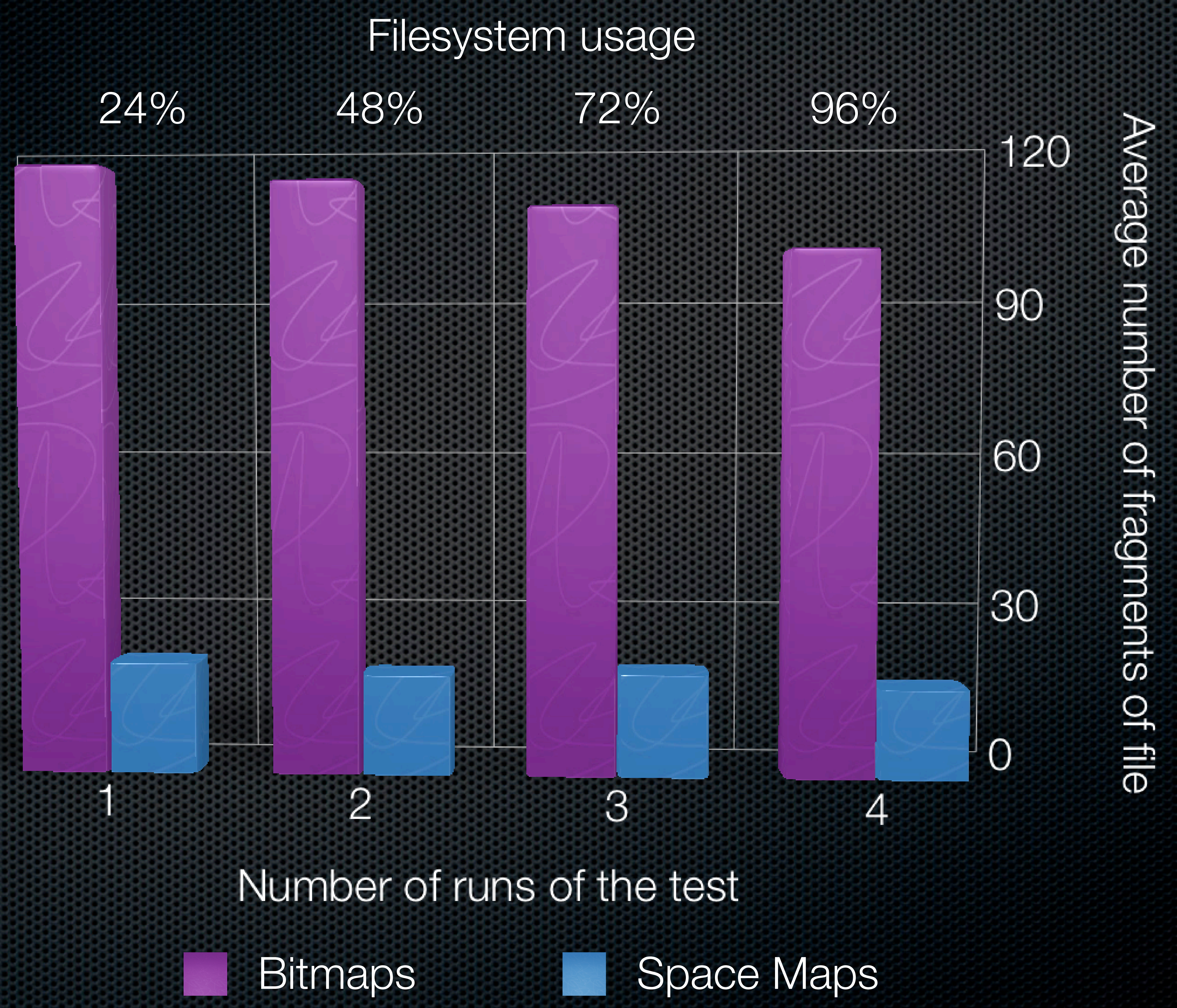
1. Free space fragments using *e2freefrag*



2. Quality of the free space fragments



3. File fragmentation measure using *filefrag*



Resulting benefits

- Reduction in seeks leading to faster allocation and deallocation.
- Reduced free space fragmentation and file fragmentation.

- Log based design; perfect locality. Only last block of on-disk log required in memory.
- No linear relationship with filesystem; thus scalable.
- Contiguous updates on disk, thus benefiting reads.

Limitations

- Mount / Unmount time delay for reading / writing space maps respectively.
- Every alternate block if full (theoretically worst case scenario).

The road ahead…

Future enhancements

- Separation of space maps based on file sizes.
- Efficient data structure for log.

Conclusion

With space maps, free space information is available completely in-memory resulting in faster filesystem operations.

Further optimizations in space maps will make it more robust and definitely lift the performance of Ext4 even higher.

References

- Mathur A., Cao M., Bhattacharya S., Dilger A., Tomas A and Viver L.; **The New ext4 filesystem: current status and future plans**
- Avantika Mathur, Mingming Cao and Andreas Dilger; **ext4: the next generation of the ext3 filesystem**
- Aneesh Kumar K.V, Mingming Cao, Jose R Santos and Andreas Dilger; **Ext4 block and inode allocator improvements**
- **Ext4** (http://kernelnewbies.org/Ext4)
- Mingming Cao, et.al.; **State of the Art: Where we are with the Ext3 filesystem**
- Jeff Bonwick; **Space Maps (zFS)**
- Rob Landley; **Red-black tree**
- Chris Mason; **Seekwatcher**
- Rupesh Thakare, Andreas Dilger, Kalpak Shah; **e2freefrag**
- Jeffrey Katcher; **PostMark: A New File System Benchmark**
- Jens Axboe, Alan D. Brunelle and Nathan Scott; **blktrace User Guide**
- Theodore Tso; **filefrag**

Thank You