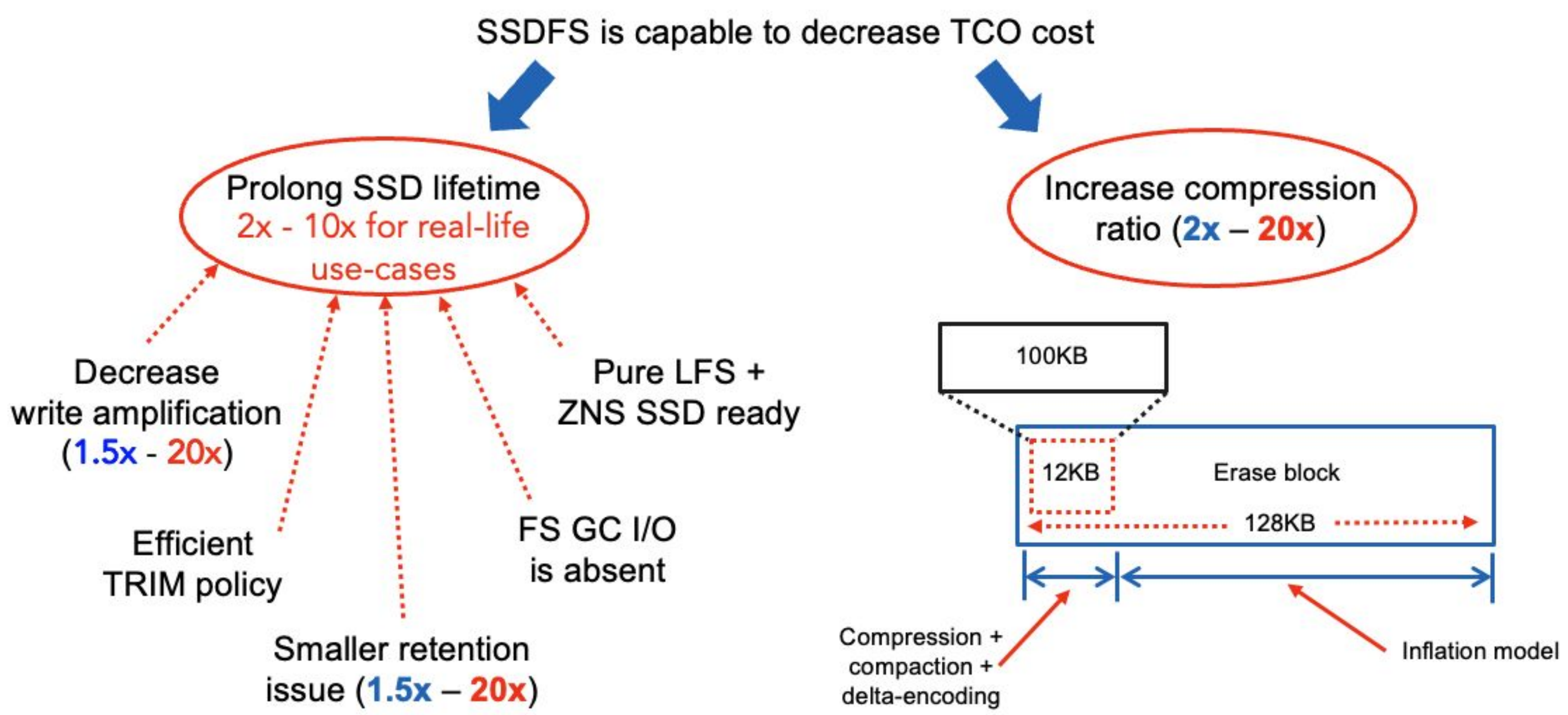
SSDFS: ZNS/FDP ready LFS file system saving your space and decreasing TCO cost

Viacheslav Dubeyko

Content

1. What is the point of SSDFS?
2. Recently implemented and stabilized features
3. Current status and upstreaming plans
4. Planned new features

What is the point of SSDFS?



SSDFS is capable to decrease TCO cost

Prolong SSD lifetime 2x - 10x for real-life use-cases

Increase compression ratio (2x – 20x)

Decrease write amplification (1.5x - 20x)

Pure LFS + ZNS SSD ready

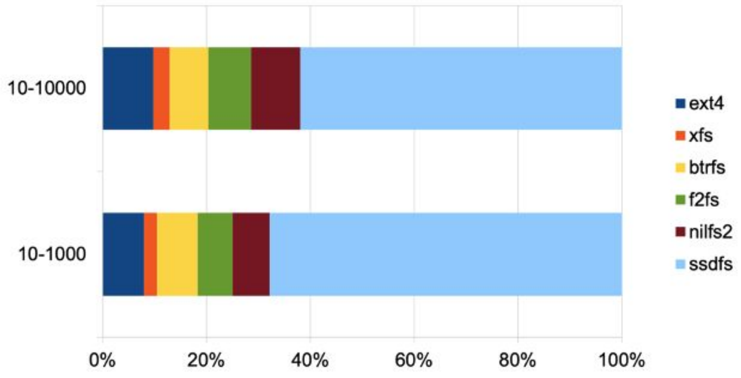
Efficient TRIM policy

FS GC I/O is absent

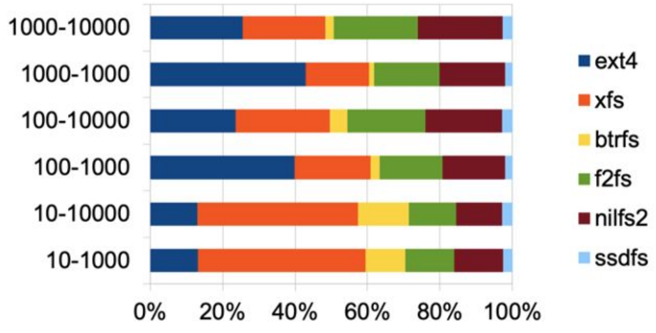
Smaller retention issue (1.5x – 20x)

Compression + compaction + delta-encoding

Inflation model

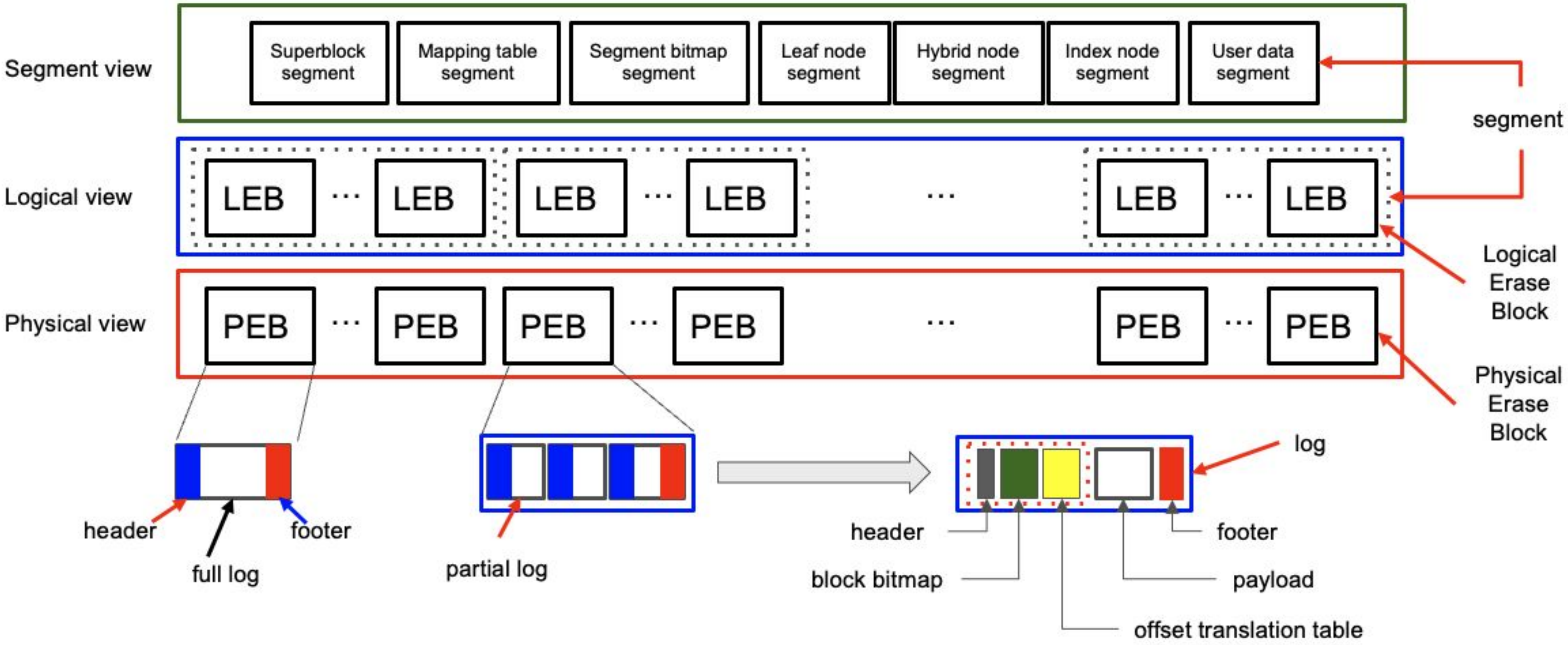


SSDFS can prolong SSD lifetime 2x - 10x for real-life use-cases

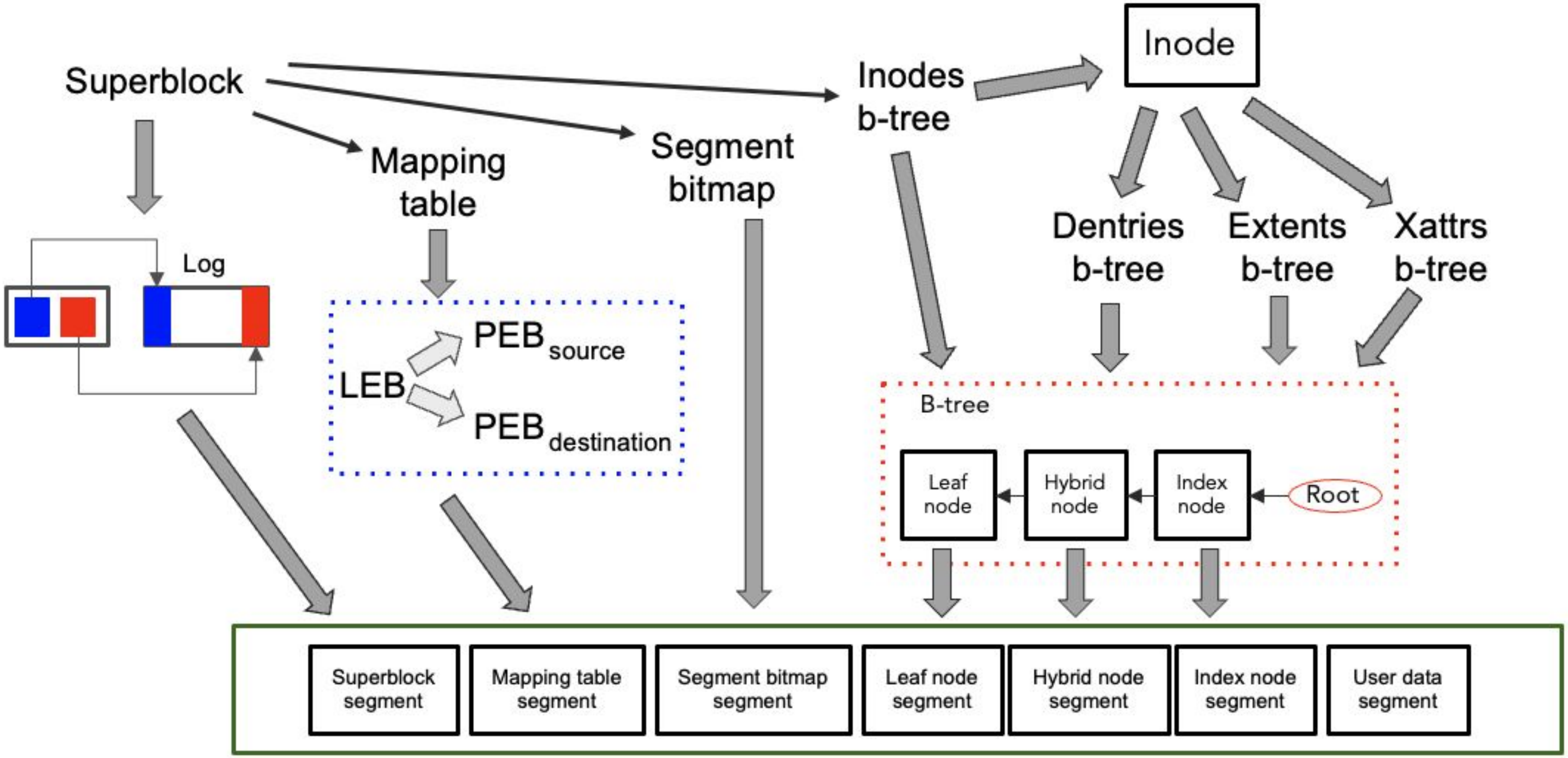


SSDFS is capable to generate **smaller amount** (1.5x - 20x) **of write I/O** requests comparing with other file systems.

SSDFS architecture (logical vs. physical view)



SSDFS architecture (metadata)



Recently implemented and stabilized features

Newly implemented features:

- folio support
- Offset translation table compression
- Storing offset translation table in every log
- Erase block inflation model
- Erase block based deduplication
- Fixed set of superblock segments
- recoverfs tool
- Snapshot rules

Stabilized features:

- Support 8K, 16K, 32K logical block sizes
- Support multiple erase blocks in segment (not fully stabilized)

Folio support

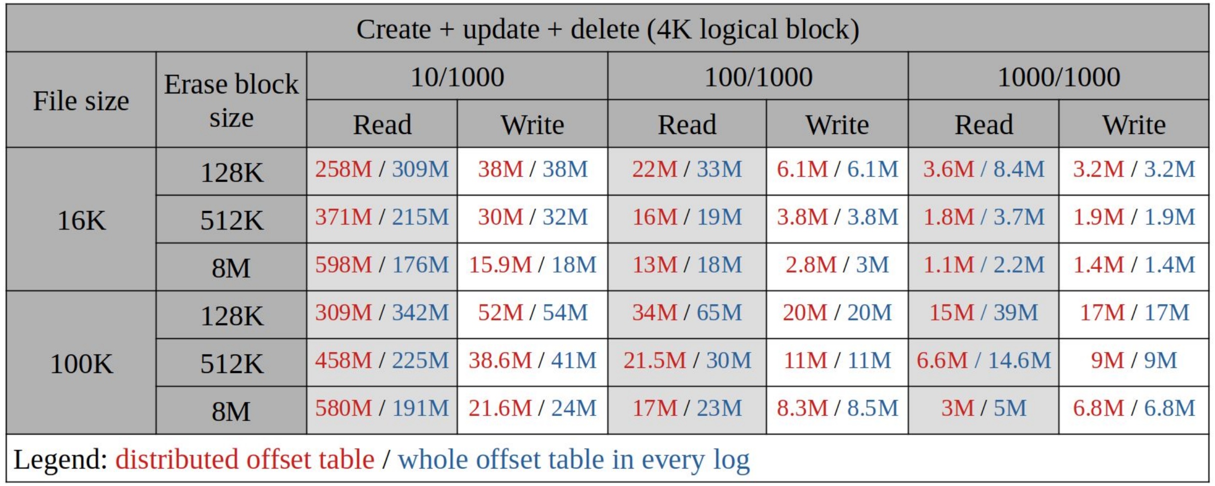
Current status:

- Folio support is implemented and tested
- It looks mostly stable (however, some particular issues could be found)
- Finally, 8K, 16K, 32K logical block sizes support works in predictable way now

Some worries:

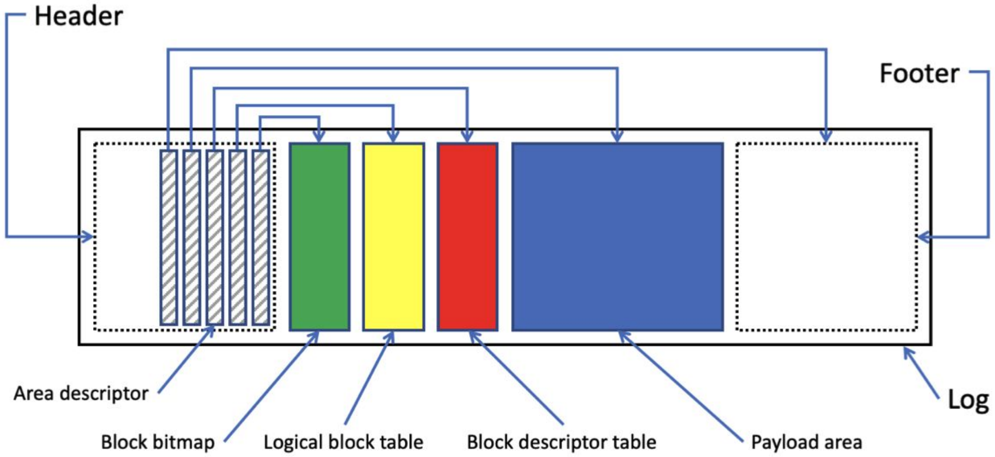
- Technically speaking, there is no guarantee that folio of 16K or 32K will be allocated (in the case of memory fragmentation). So, file system needs to be ready to process logical block that contains several smaller folios (for example, potentially, 32K can be represented by 16K, 8K, 4K, 4K folios set)
- Readahead logic doesn’t take into account the logical block size for a particular file system volume. As a result, real-life case is of having 32K folios in the page cache even if file system works with (and expects) 16K logical block sizes. Why readahead logic doesn’t take into account the logical block size?

Storing offset table in every log vs. distributed model



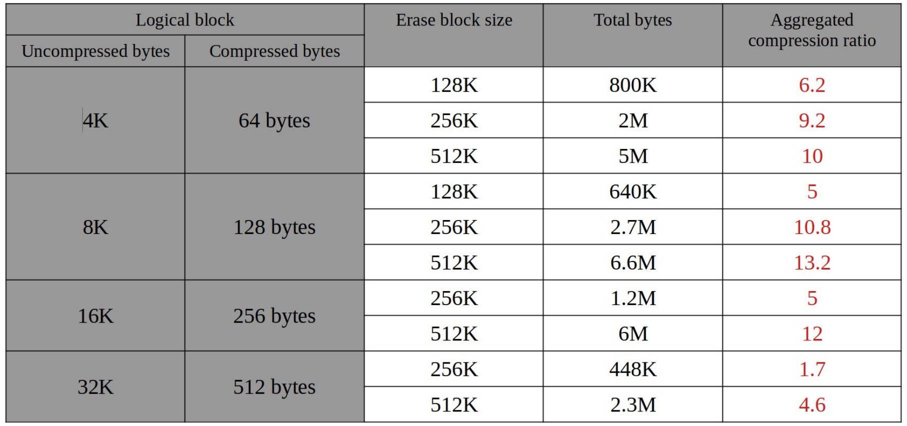
Create + update + delete (4K logical block)							
File size	Erase block size	10/1000		100/1000		1000/1000	
		Read	Write	Read	Write	Read	Write
16K	128K	258M / 309M	38M / 38M	22M / 33M	6.1M / 6.1M	3.6M / 8.4M	3.2M / 3.2M
	512K	371M / 215M	30M / 32M	16M / 19M	3.8M / 3.8M	1.8M / 3.7M	1.9M / 1.9M
	8M	598M / 176M	15.9M / 18M	13M / 18M	2.8M / 3M	1.1M / 2.2M	1.4M / 1.4M
100K	128K	309M / 342M	52M / 54M	34M / 65M	20M / 20M	15M / 39M	17M / 17M
	512K	458M / 225M	38.6M / 41M	21.5M / 30M	11M / 11M	6.6M / 14.6M	9M / 9M
	8M	580M / 191M	21.6M / 24M	17M / 23M	8.3M / 8.5M	3M / 5M	6.8M / 6.8M

Legend: distributed offset table / whole offset table in every log

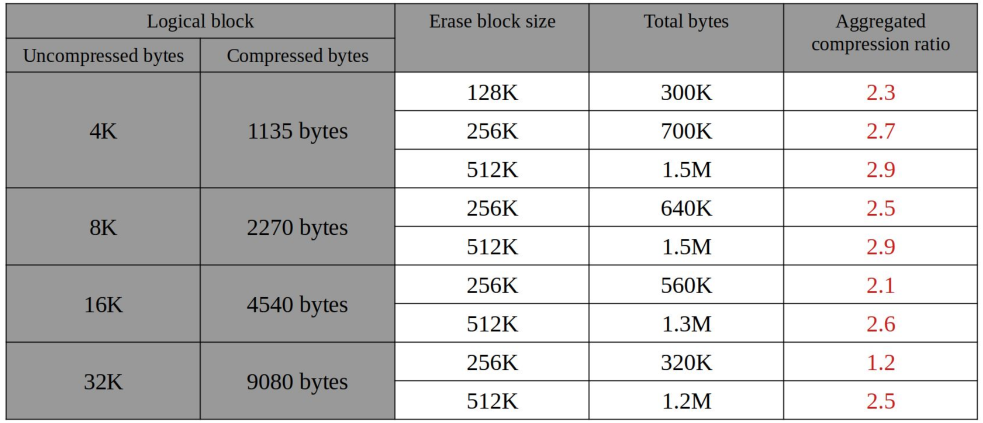


- Storing offset translation table in every log mostly doesn't initiate more write I/O requests compared with distributed model of offset translation table
- 128KB, 256KB erase blocks don't benefit from storing offset translation table in every log for the case of read I/O requests
- Storing offset translation table in every log makes sense for 512K, 2MB, 8MB and bigger erase block sizes because it could reduce amount of read I/O requests 3x times
- However, if number of logs per erase block is lesser than 20, then storing offset translation table in every log approach can generate more read I/O requests
- "Cold" data could benefit from distributed model of offset translation table
- Storing offset translation table in every log is beneficial for the case of frequently updated data.

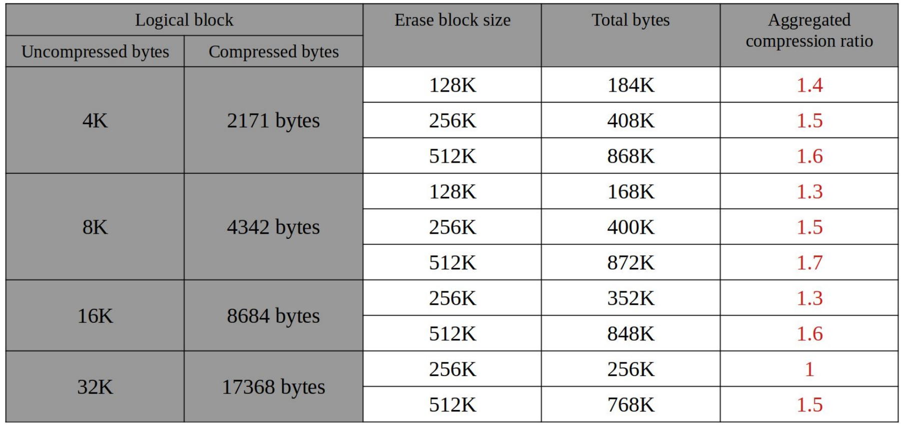
Erase block inflation model + moving scheme



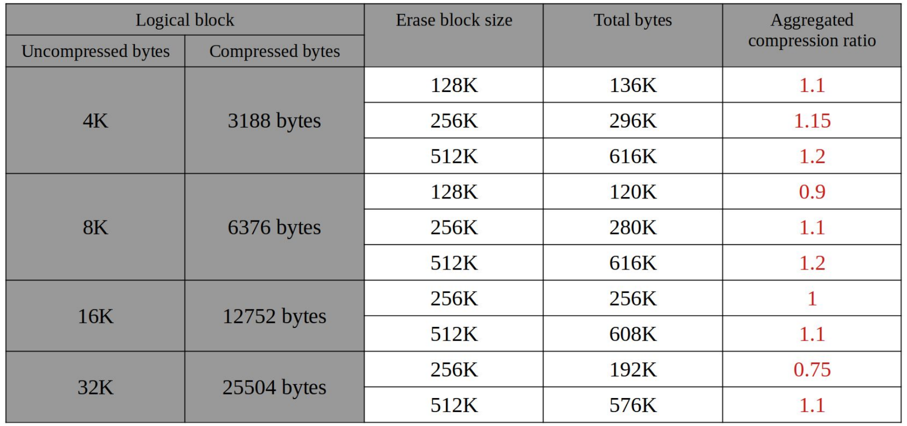
Logical block		Erase block size	Total bytes	Aggregated compression ratio
Uncompressed bytes	Compressed bytes			
4K	64 bytes	128K	800K	6.2
		256K	2M	9.2
		512K	5M	10
8K	128 bytes	128K	640K	5
		256K	2.7M	10.8
		512K	6.6M	13.2
16K	256 bytes	256K	1.2M	5
		512K	6M	12
32K	512 bytes	256K	448K	1.7
		512K	2.3M	4.6



Logical block		Erase block size	Total bytes	Aggregated compression ratio
Uncompressed bytes	Compressed bytes			
4K	1135 bytes	128K	300K	2.3
		256K	700K	2.7
		512K	1.5M	2.9
8K	2270 bytes	256K	640K	2.5
		512K	1.5M	2.9
16K	4540 bytes	256K	560K	2.1
		512K	1.3M	2.6
32K	9080 bytes	256K	320K	1.2
		512K	1.2M	2.5

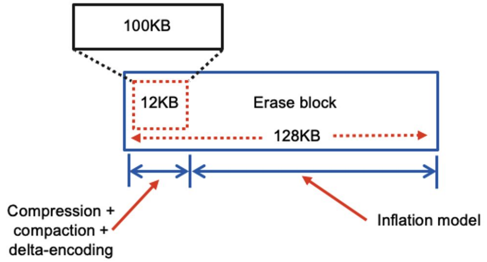


Logical block		Erase block size	Total bytes	Aggregated compression ratio
Uncompressed bytes	Compressed bytes			
4K	2171 bytes	128K	184K	1.4
		256K	408K	1.5
		512K	868K	1.6
8K	4342 bytes	128K	168K	1.3
		256K	400K	1.5
		512K	872K	1.7
16K	8684 bytes	256K	352K	1.3
		512K	848K	1.6
32K	17368 bytes	256K	256K	1
		512K	768K	1.5

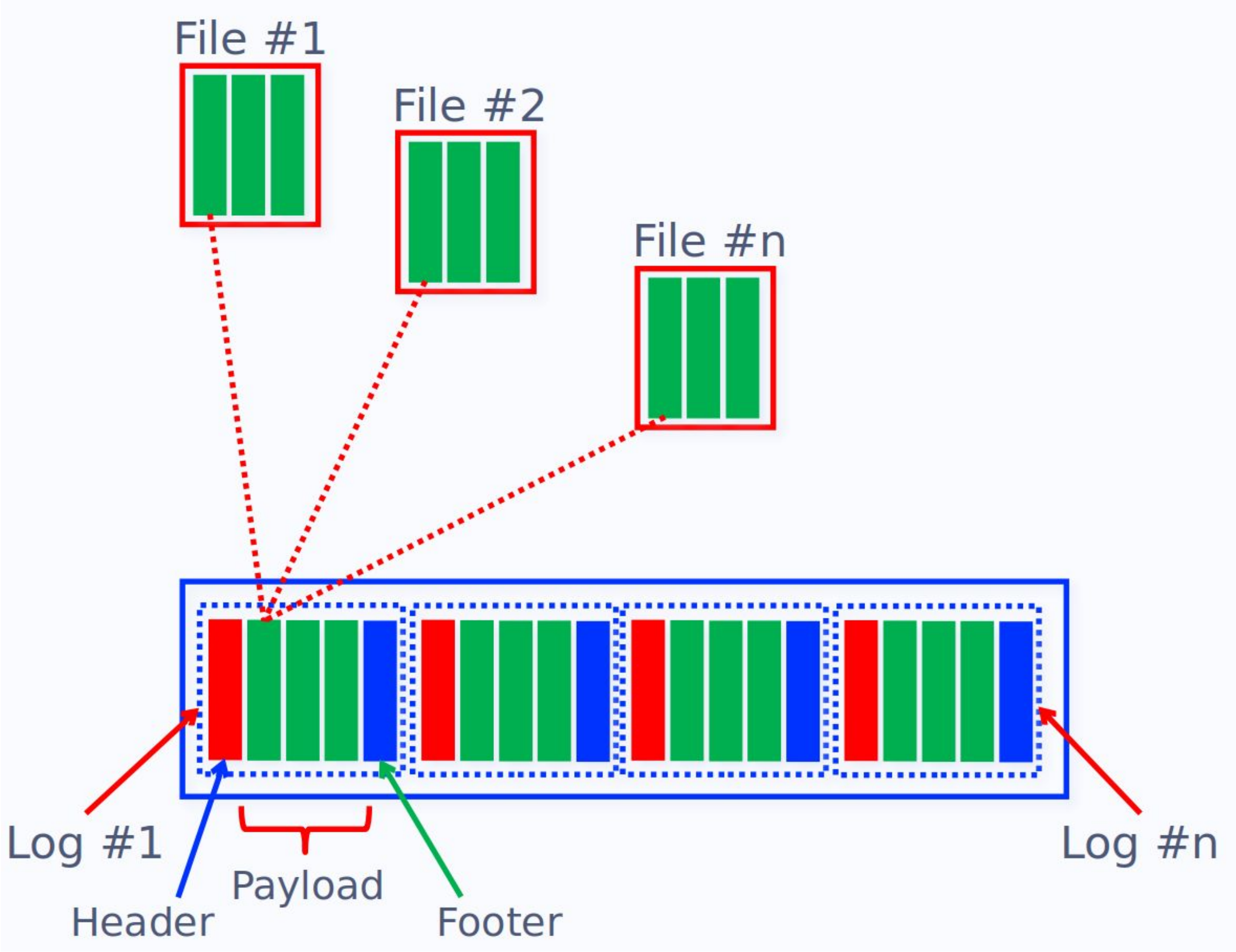


Logical block		Erase block size	Total bytes	Aggregated compression ratio
Uncompressed bytes	Compressed bytes			
4K	3188 bytes	128K	136K	1.1
		256K	296K	1.15
		512K	616K	1.2
8K	6376 bytes	128K	120K	0.9
		256K	280K	1.1
		512K	616K	1.2
16K	12752 bytes	256K	256K	1
		512K	608K	1.1
32K	25504 bytes	256K	192K	0.75
		512K	576K	1.1

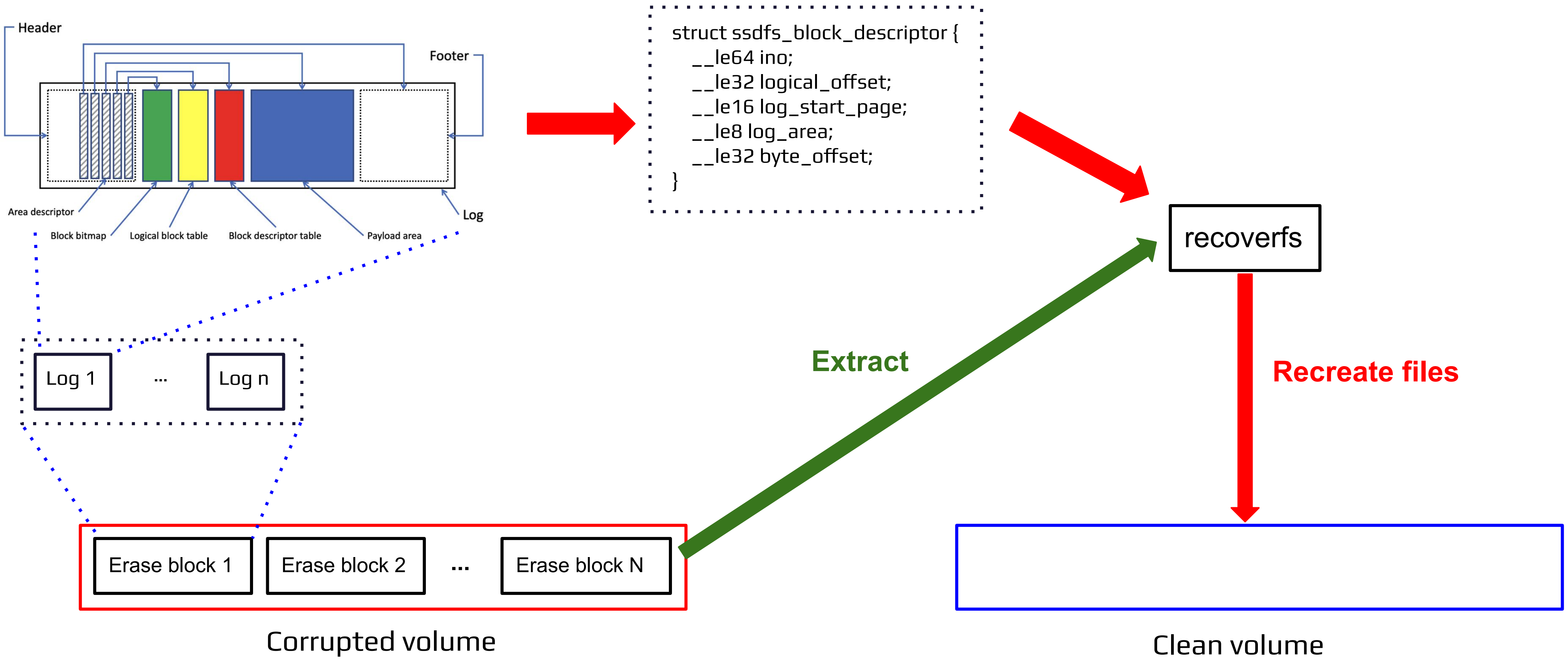
Erase block inflation model is capable of storing 1.5x - 12x more data than physical capacity of erase block.



Erase block based deduplication



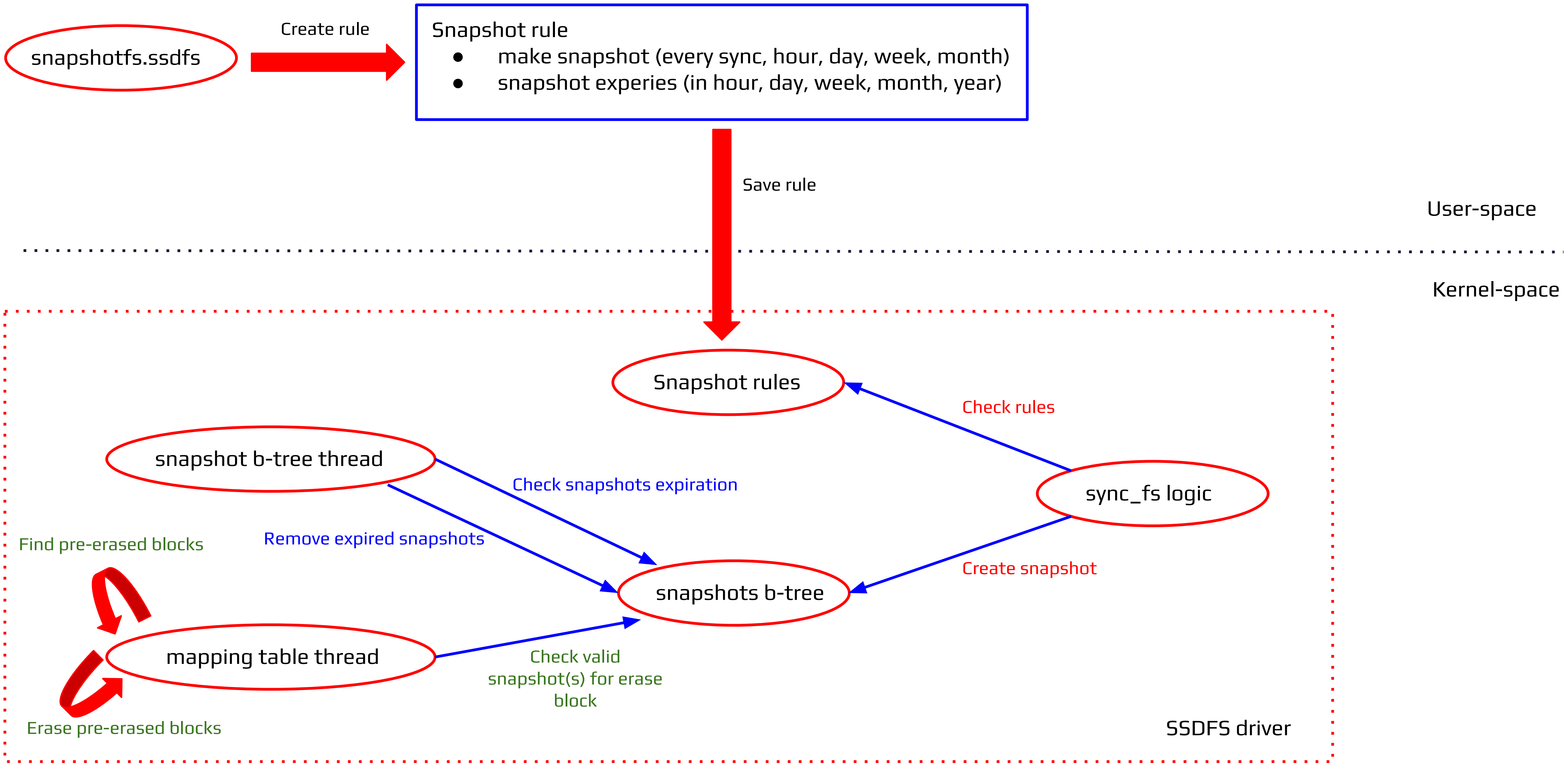
The recoverfs tool



```
struct ssdfs_block_descriptor {
    __le64 ino;
    __le32 logical_offset;
    __le16 log_start_page;
    __le8 log_area;
    __le32 byte_offset;
}
```

Copy data from corrupted volume as a last resort

Snapshot rules



snapshotfs.ssdfs

Create rule

Snapshot rule
- make snapshot (every sync, hour, day, week, month)
- snapshot experies (in hour, day, week, month, year)

Save rule

User-space

Kernel-space

Snapshot rules

Check rules

snapshot b-tree thread

Check snapshots expiration

sync_fs logic

Find pre-erased blocks

Remove expired snapshots

Create snapshot

snapshots b-tree

mapping table thread

Check valid snapshot(s) for erase block

Erase pre-erased blocks

SSDFS driver

Microbenchmarking: environment

```
Linux 6.10.0 #34 SMP PREEMPT_DYNAMIC Tue Aug 13 18:50:14
MSK 2024 x86_64 x86_64 x86_64 GNU/Linux

11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz
MemTotal:      32452532 kB

Model Family:        Silicon Motion based SSDs
Device Model:        TS128GSSD230S
User Capacity:       128,035,676,160 bytes [128 GB]
Sector Size:  512 bytes logical/physical
Rotation Rate:       Solid State Device
Form Factor:         2.5 inches
TRIM Command:        Available, deterministic, zeroed
ATA Version is:   ACS-3 T13/2161-D revision 5
SATA Version is:  SATA 3.3, 6.0 Gb/s (current: 6.0 Gb/s)
```

CREATE:

```
for (i = 0; i < file_number; i++) {
   touch <file_name>
   dd if=./pattern1.bin of=<file_name> conv=notrunc oflag=append bs=4096 count=1
}
sync
```

READ:

```
for (i = 0; i < file_number; i++) {
   md5sum <file_name>
}
```

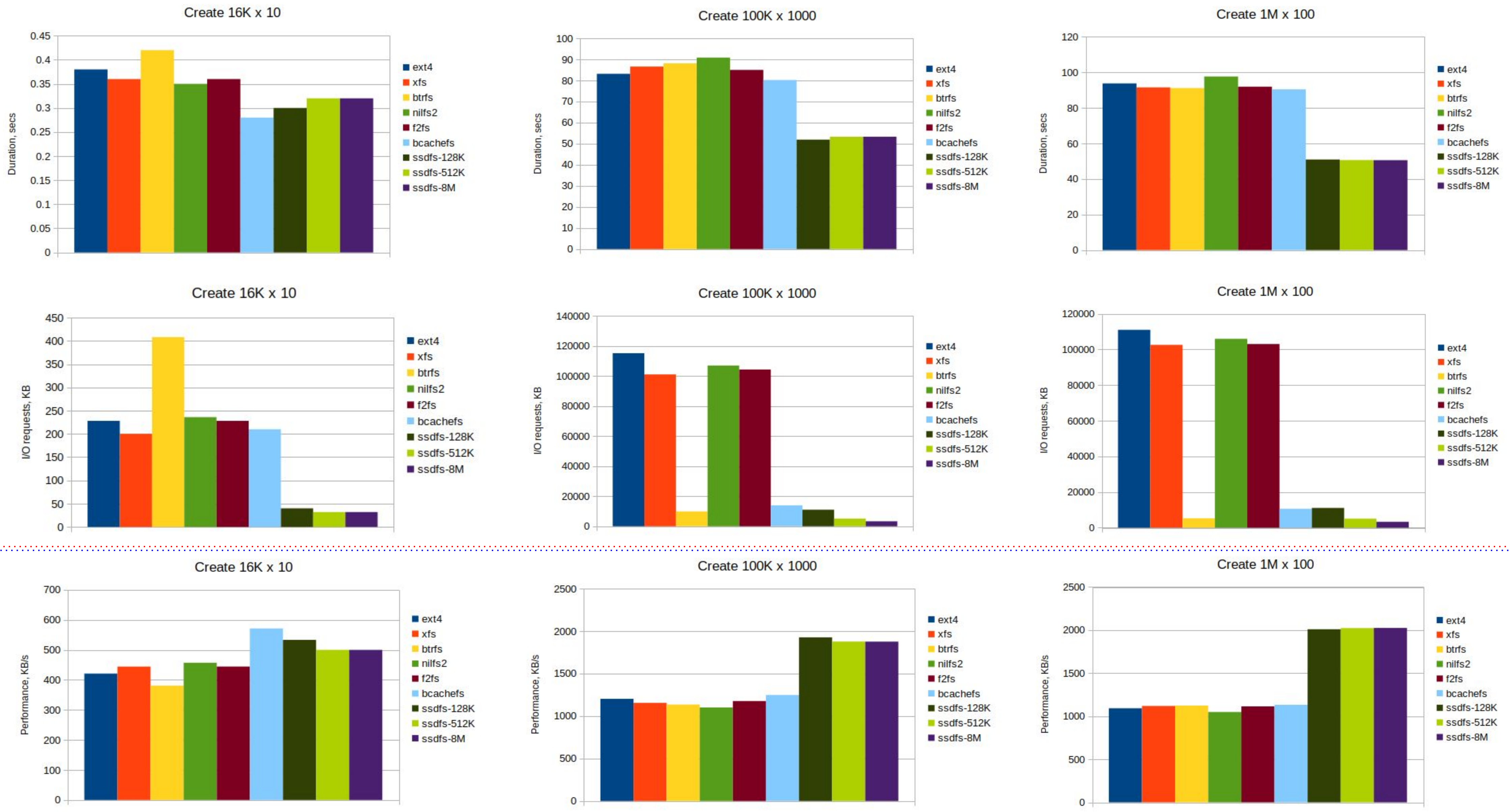
UPDATE:

```
for (i = 0; i < file_number; i++) {
   dd if=./pattern2.bin of=<file_name> conv=notrunc seek=offset bs=4096 count=1
   offset += 4096
}
sync
```

DELETE:

```
for (i = 0; i < file_number; i++) {
   rm <file_name>
}
sync
```

Microbenchmarking: create operation



Duration

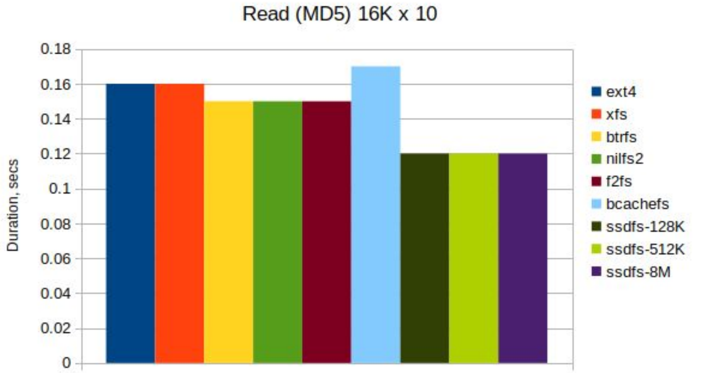
I/O requests

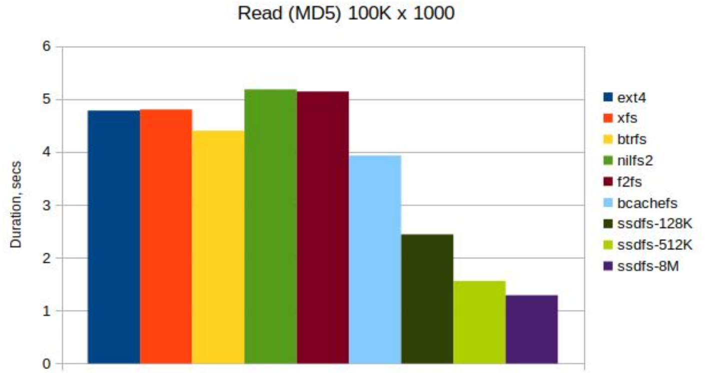
The lesser the better

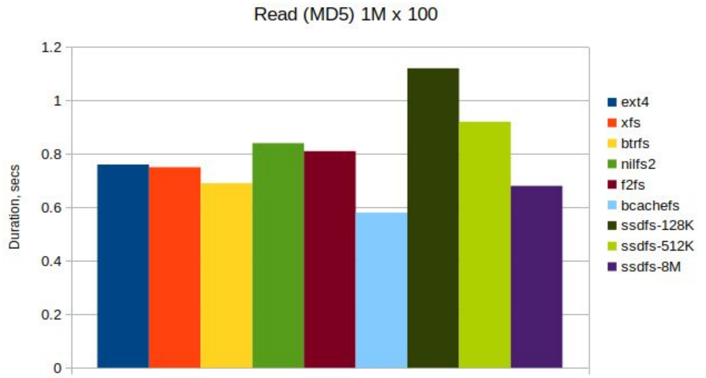
Performance

The bigger the better

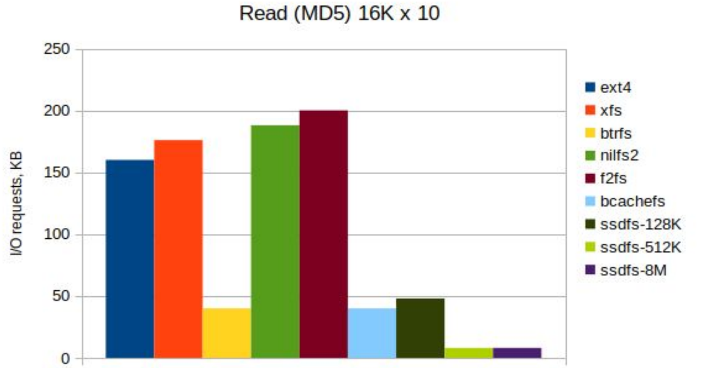
Microbenchmarking: read (MD5) operation

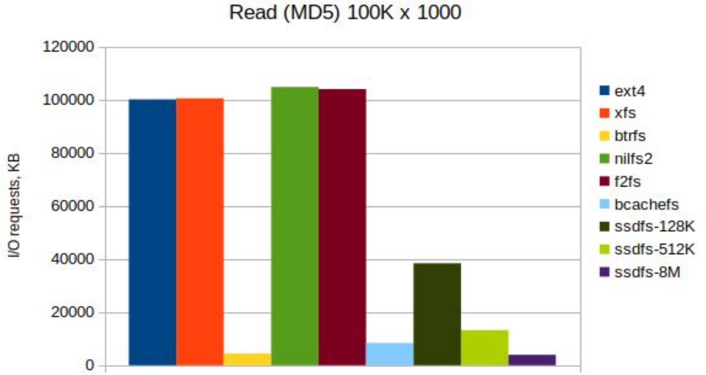

Read (MD5) 16K x 10

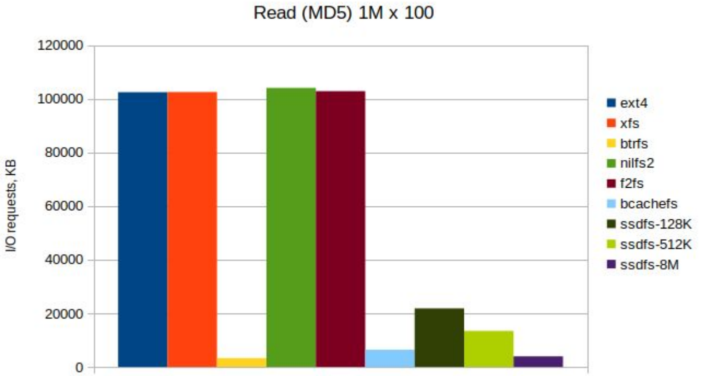

Read (MD5) 100K x 1000


Read (MD5) 1M x 100

Duration

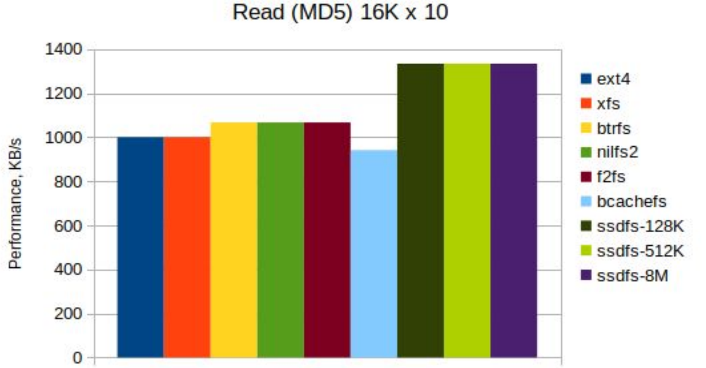

Read (MD5) 16K x 10

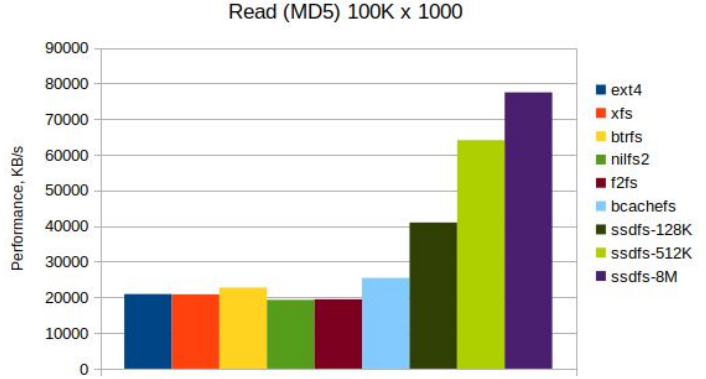

Read (MD5) 100K x 1000

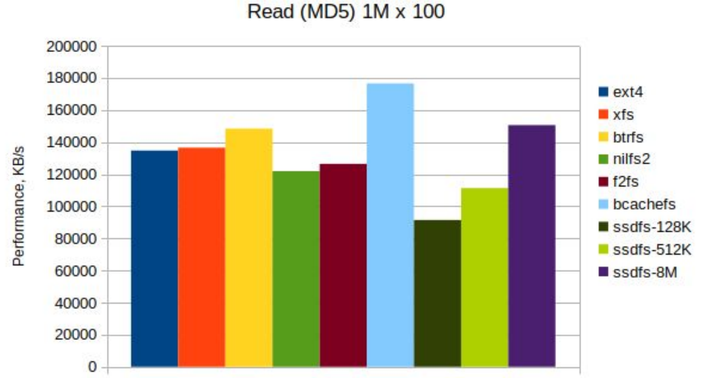

Read (MD5) 1M x 100

I/O requests

The lesser the better


Read (MD5) 16K x 10

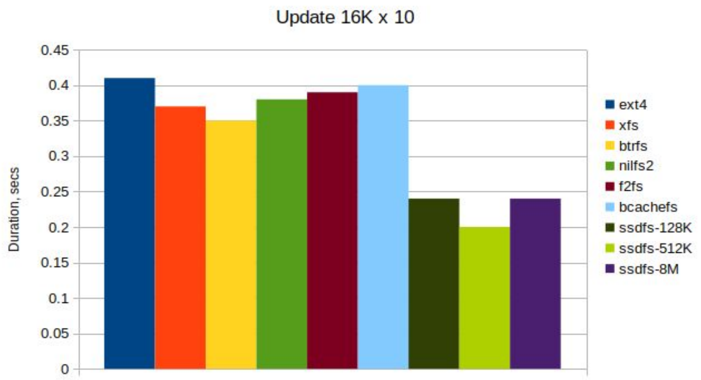

Read (MD5) 100K x 1000

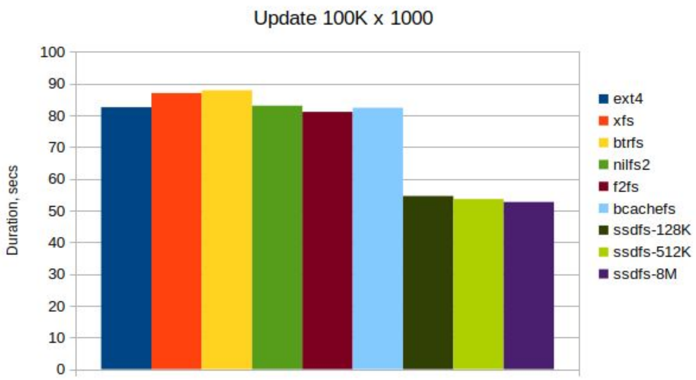

Read (MD5) 1M x 100

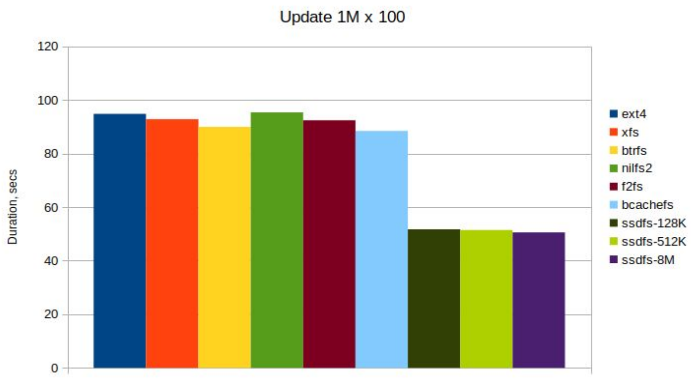
Performance

The bigger the better

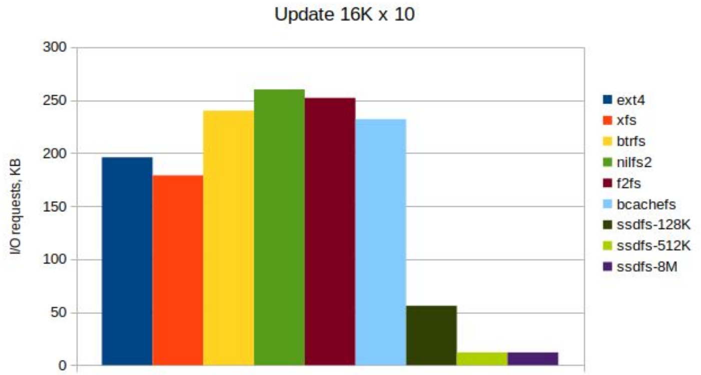
Microbenchmarking: update operation

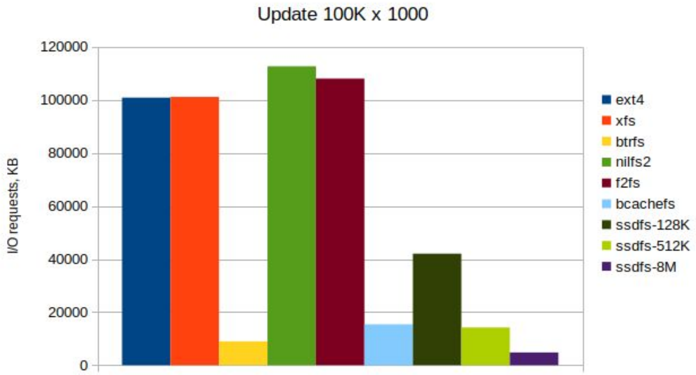


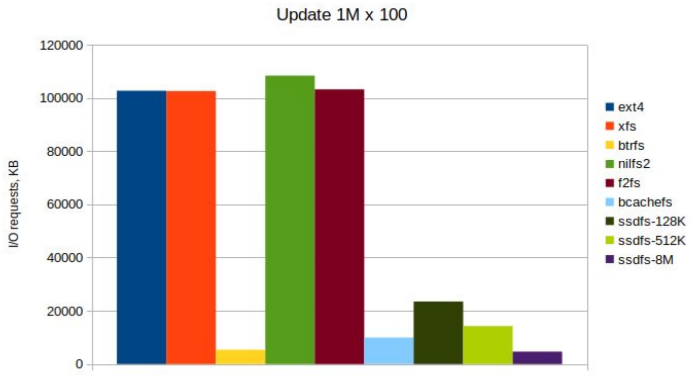




Duration

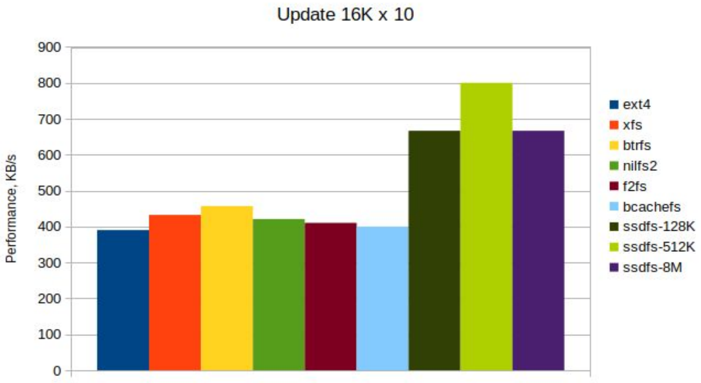


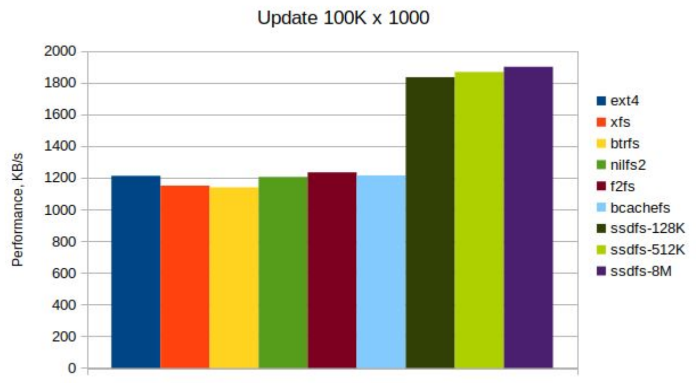


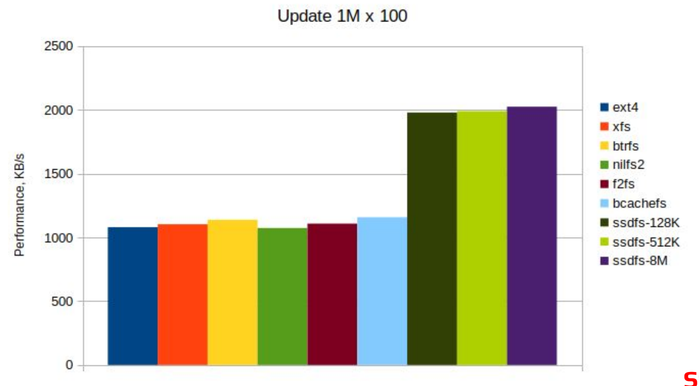


I/O requests

The lesser the better



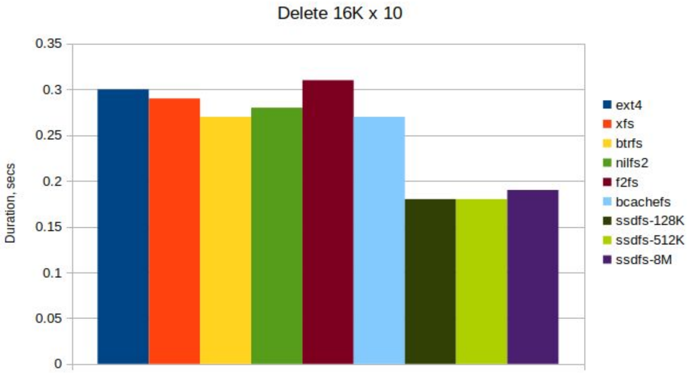


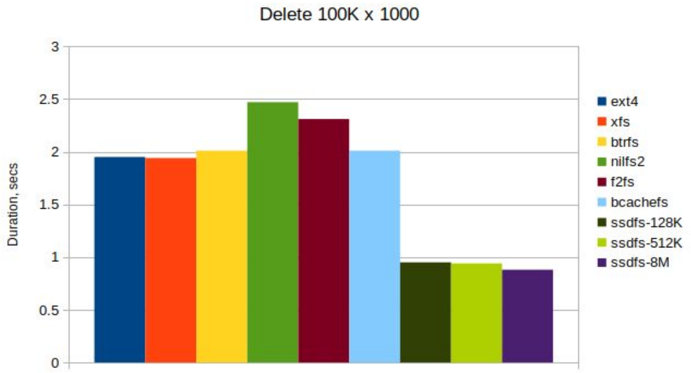


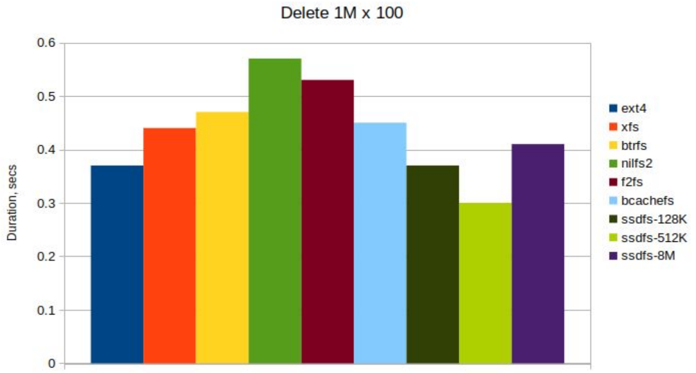
Performance

The bigger the better

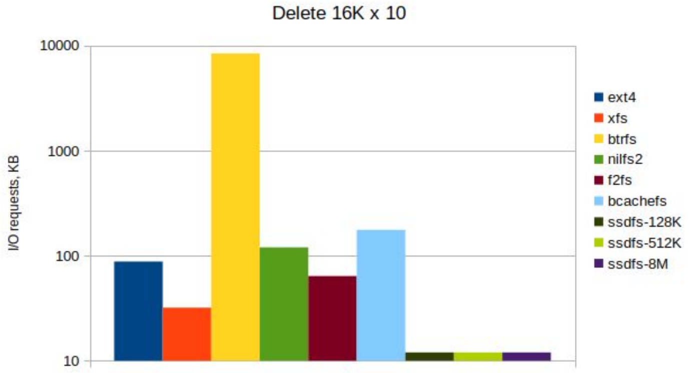
Microbenchmarking: delete operation

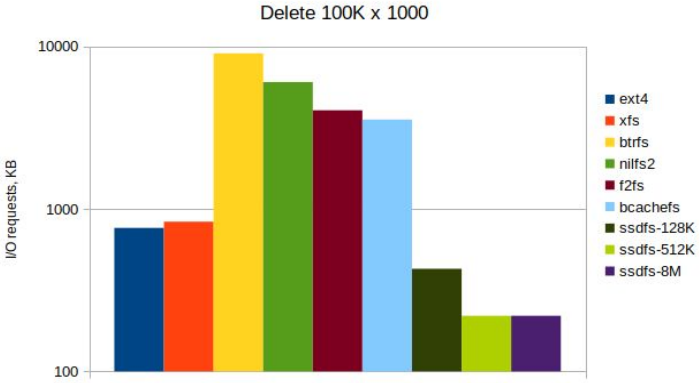


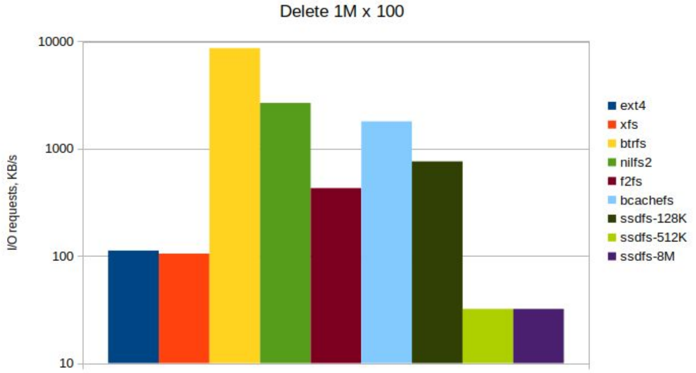




Duration

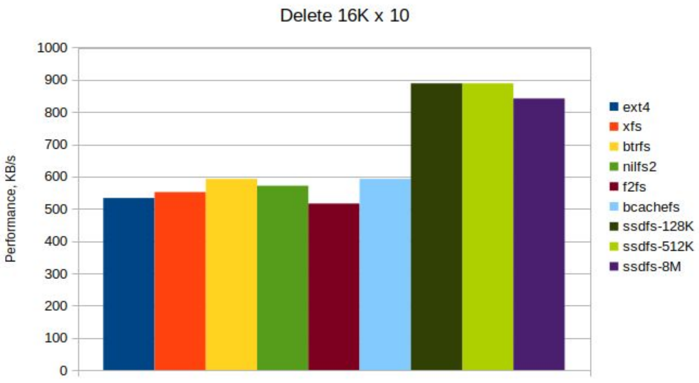


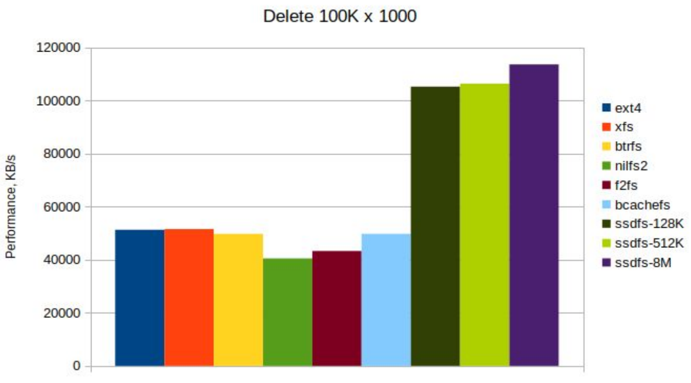


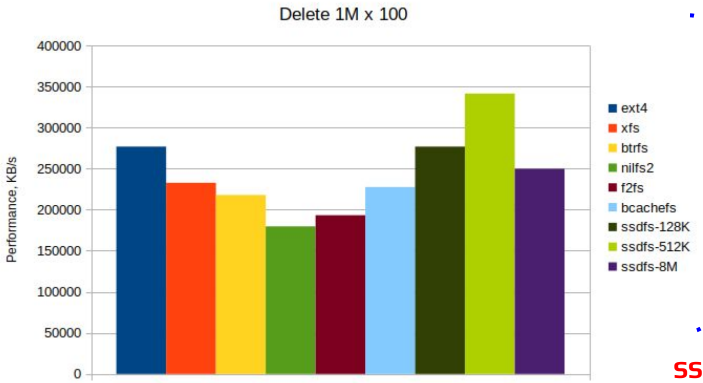


I/O requests

The lesser the better







Performance

The bigger the better

Microbenchmarking: conclusion

SSDFS is capable to demonstrate a **better performance** for data with good compression ratio:

create operation:

- ext4: 1.2x - 1.8x
- xfs: 1.2x - 1.8x
- btrfs: 1.3x - 1.8x
- nilfs2: 1.1x - 1.9x
- f2fs: 1.2x - 1.8x
- bcachefs: 0.9x - 1.7x

read operation:

- ext4: 1.1x - 3.7x
- xfs: 1.1x - 3.7x
- btrfs: 1x - 3.4x
- nilfs2: 1.2x - 4x
- f2fs: 1.2x - 3.9x
- bcachefs: 0.8x - 3x

update operation:

- ext4: 1.5x - 1.8x
- xfs: 1.5x - 1.8x
- btrfs: 1.4x - 1.7x
- nilfs2: 1.5x - 1.8x
- f2fs: 1.5x - 1.8x
- bcachefs: 1.5x - 1.7x

delete operation:

- ext4: 1.2x - 2.2x
- xfs: 1.4x - 2.2x
- btrfs: 1.5x - 2.2x
- nilfs2: 1.5x - 2.8x
- f2fs: 1.7x - 2.6x
- bcachefs: 1.5x - 2.2x

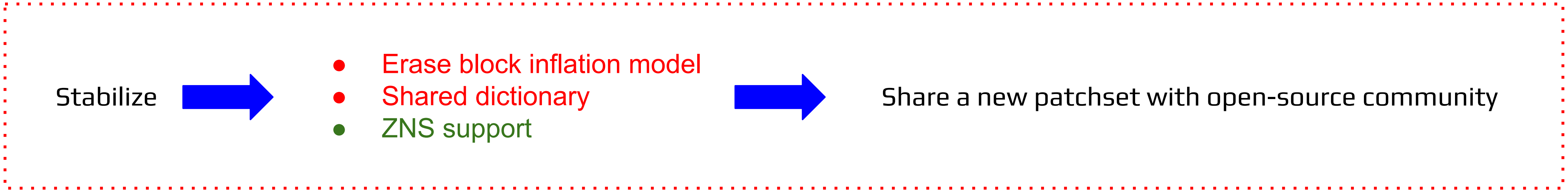
- The bigger erase block size is the better.
- Small erase block size (for example, 128KB, 256KB) could be the reason of bigger amount of metadata in the logs.
- SSDFS can be more efficient with small files.
- Read operation with big files looks like not very efficient for the case of SSDFS file system.
- Create, update, and delete operations look pretty efficient for the most cases.

Current status, issues, and plans

Stable features:
- Mount logic – **stable**
- Mapping table – **stable**
- Segment bitmap – **stable**
- Migration scheme – **stable**
- Inodes tree – **stable**
- Dentries tree – **stable**
- Extents tree – **stable**
- Folio support – **stable**
- 8K/16K/32K logical block – **stable**
- Erase block based deduplication - **stable**

NOT stable features:
- Erase block inflation model - **not fully stable**
- ZNS support - **not stable**
- Shared dictionary - **not stable**
- recoverfs - **not stable**
- Xattrs tree – **not fully stable**
- Delta-encoding – **not fully stable**
- Multiple erase blocks in segment – **not stable**

Under implementation:
- Deduplication – **not fully implemented**
- Snapshots – **not fully implemented**
- Fsck – **not fully implemented**



Stabilize

- Erase block inflation model
- Shared dictionary
- ZNS support

Share a new patchset with open-source community

SSDFS tools: https://github.com/dubeyko/ssdfs-tools.git
SSDFS driver: https://github.com/dubeyko/ssdfs-driver.git
Linux kernel with SSDFS support: https://github.com/dubeyko/linux.git

Planned new features

- FDP support
- Delta-encoding similar logical blocks
- Online fsck + offline fsck
- Scrubbing
- Multi-drive support
- Erasure coding scheme
- File-based deduplication
- Snapshot access + management

THANK YOU

QUESTIONS???