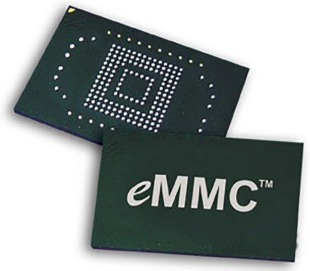
SSDFS + ZNS SSD: deterministic architecture decreasing TCO cost of data infrastructure

Viacheslav Dubeyko <slava@dubeyko.com>

What is SSDFS?

SSDFS is flash-friendly and ZNS natively compatible open-source Linux kernel-space file system



eMMC

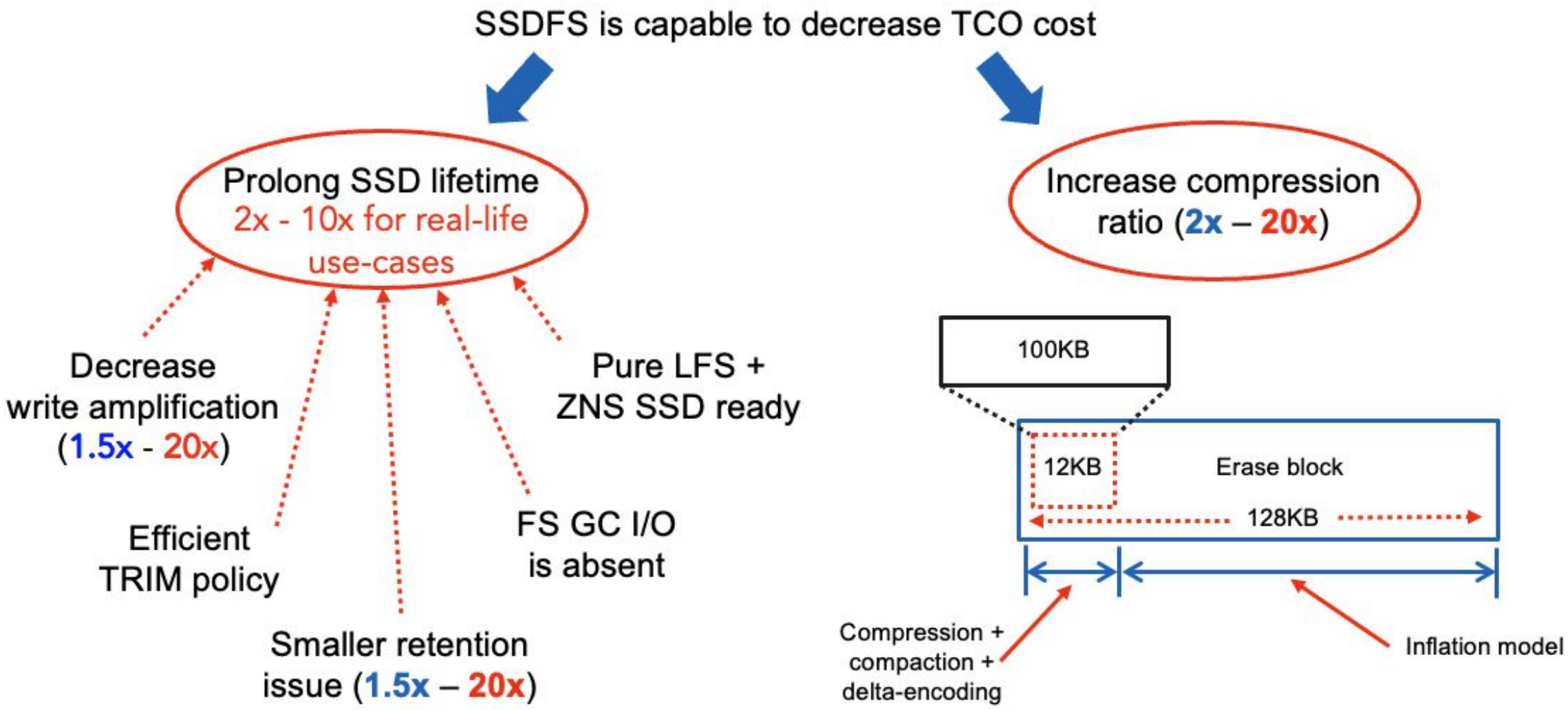
NVMe SSD

ZNS SSD

SMR HDD

https://www.indiegogo.com/projects/ssdfs-zns-ssd-native-file-system/x/34131201#/

What is the point of SSDFS?



https://www.indiegogo.com/projects/ssdfs-zns-ssd-native-file-system/x/34131201#/

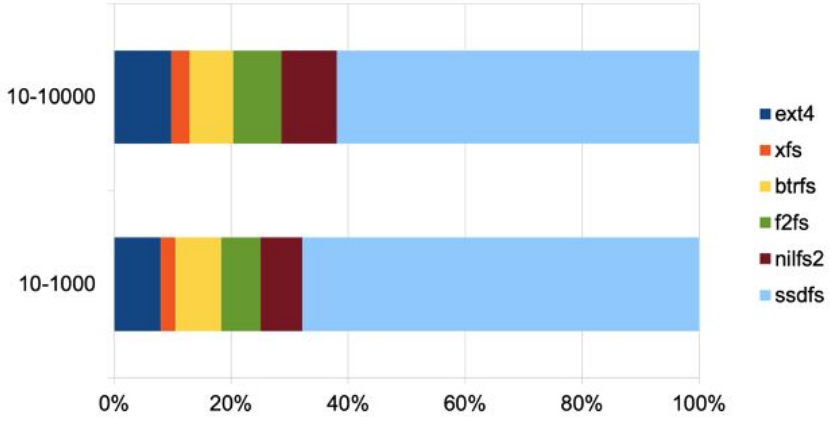
Decrease carbon footprint

Prolong SSD lifetime
Decrease number of write/read I/O requests
Eliminate GC activity
Store more data on the same device (increase compression ratio)

Decrease TCO cost
Decrease power consumption

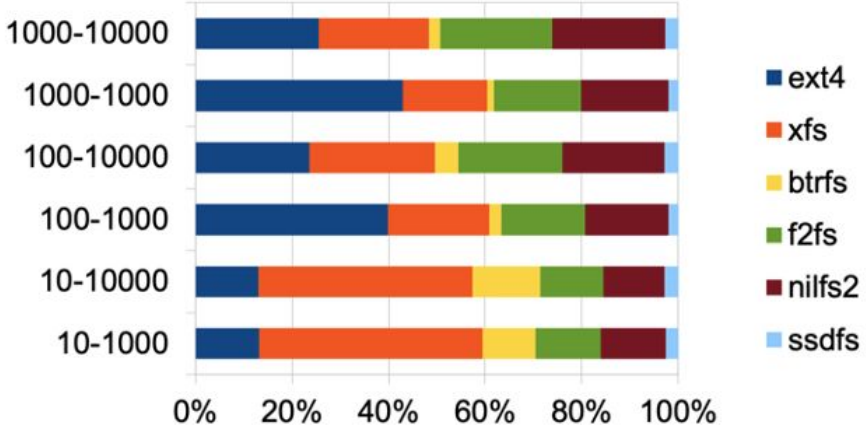
Decrease carbon footprint
Support "green" economy
Save the planet

Lifetime estimation

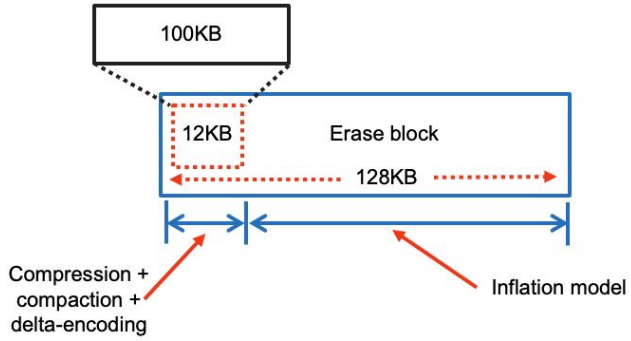


SSDFS can prolong SSD lifetime
2x - 10x for real-life use-cases

Write I/O requests



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



https://www.indiegogo.com/projects/ssdfs-zns-ssd-native-file-system/x/34131201#/

Why yet another file system?

NILFS2 → reliability
- in-place update superblocks
- COW policy (LFS)
- user-space GC
- snapshots

F2FS → performance
- in-place update metadata area
- COW area
- kernel-space GC
- dual checkpoints
- transparent file compression
- file system level encryption

bcachefs → reliability + performance
- Copy on write (COW) - like zfs or btrfs
- COW b-trees + journal
- Copying garbage collection
- Full data and metadata checksumming
- compression
- Multiple devices
- Replication + Erasure coding
- encryption
- snapshots

SSDFS
- Pure LFS (COW policy) + ZNS SSD ready
- compression + delta-encoding + compaction scheme
- migration scheme + migration stimulation + noGC overhead
- deduplication (not fully implemented)
- post-deduplication delta-compression (planned)
- prolong SSD lifetime
- snapshots (not fully implemented)
- recoverfs (reconstruct file system state -> heavily corrupted volume)
- employ parallelism of multiple NAND dies

1. prolong SSD lifetime
2. reliability
3. performance

SSDFS

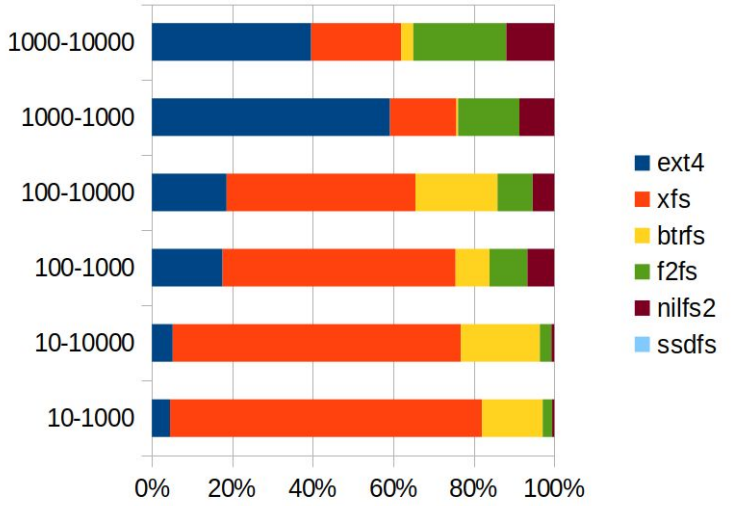
https://www.indiegogo.com/projects/ssdfs-zns-ssd-native-file-system/x/34131201#/

File systems landscape

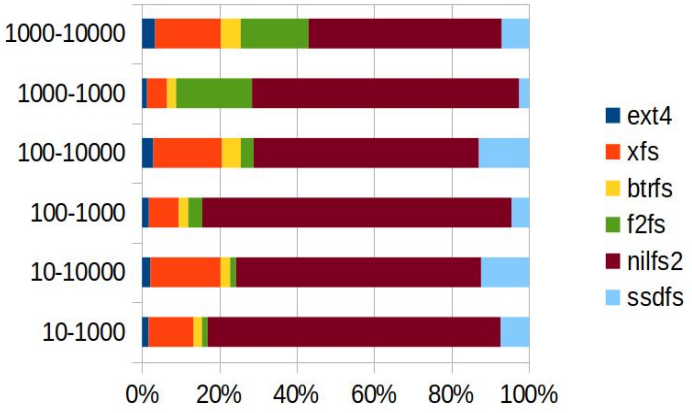
Current issues:

- FTL GC responsibility (in-place update)
- FS GC overhead (LFS/COW file systems)
- Significant write amplification
- No ZNS/SMR native compatibility
- Retention issue (not efficient TRIM/discard policy)
- Significant read disturbance

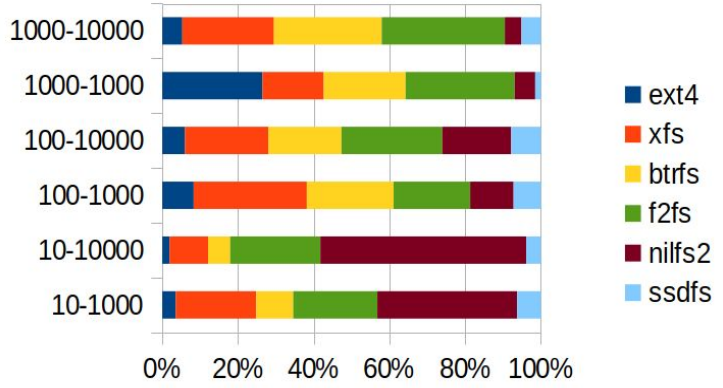
FTL responsibility



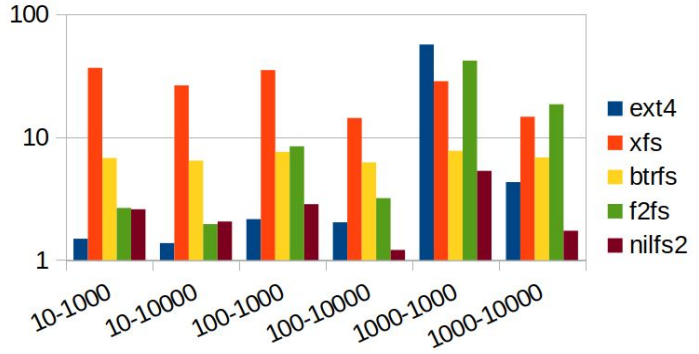
Read disturbance (estimation)



Retention issue (estimation)

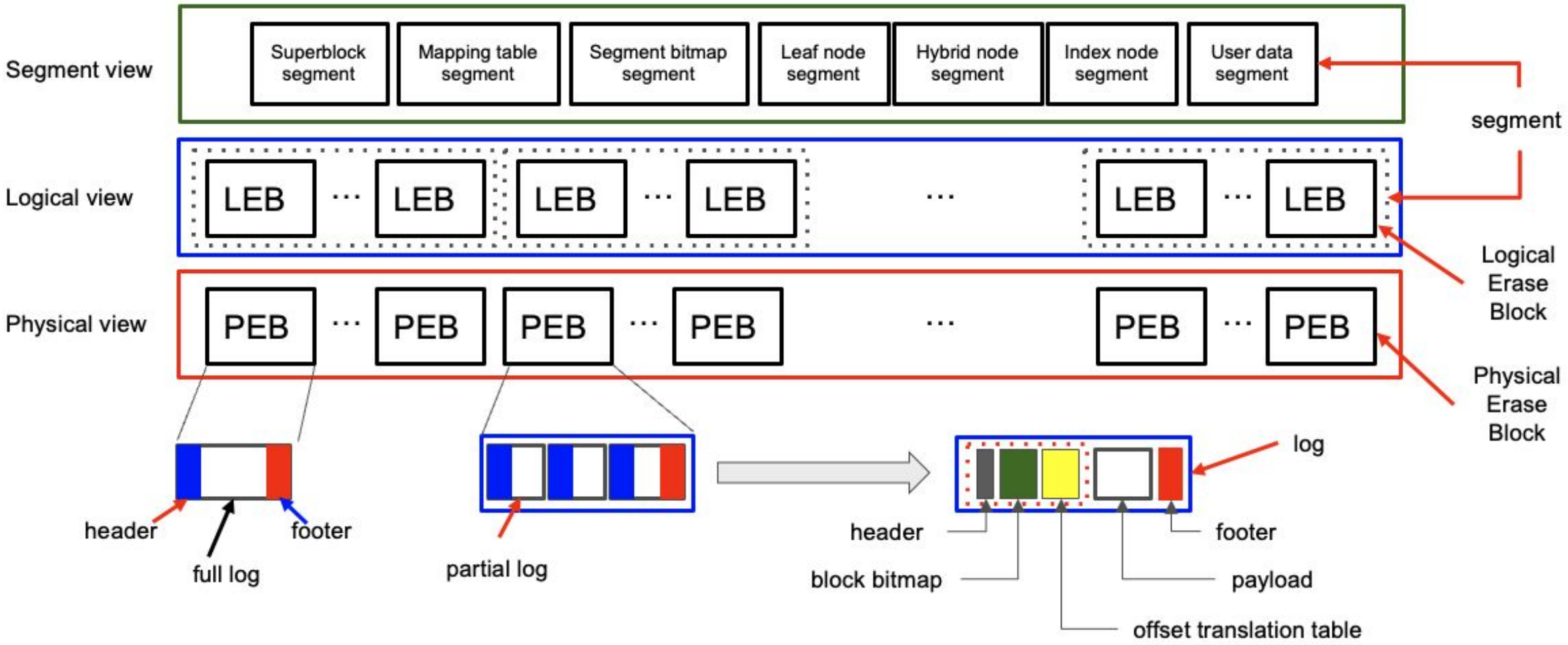


Write amplification (estimation)



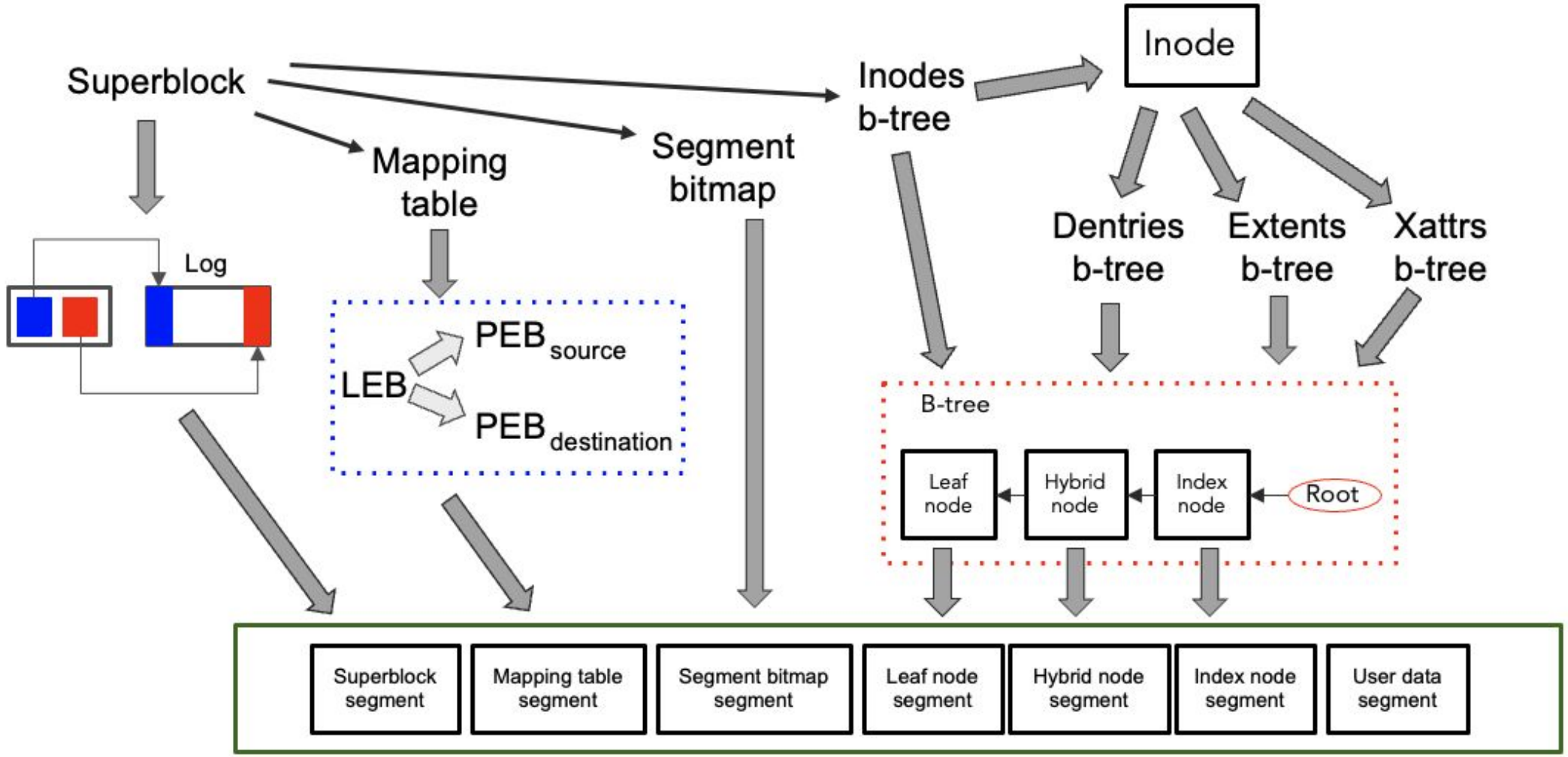
https://www.indiegogo.com/projects/ssdfs-zns-ssd-native-file-system/x/34131201#/

SSDFS architecture (logical vs. physical view)



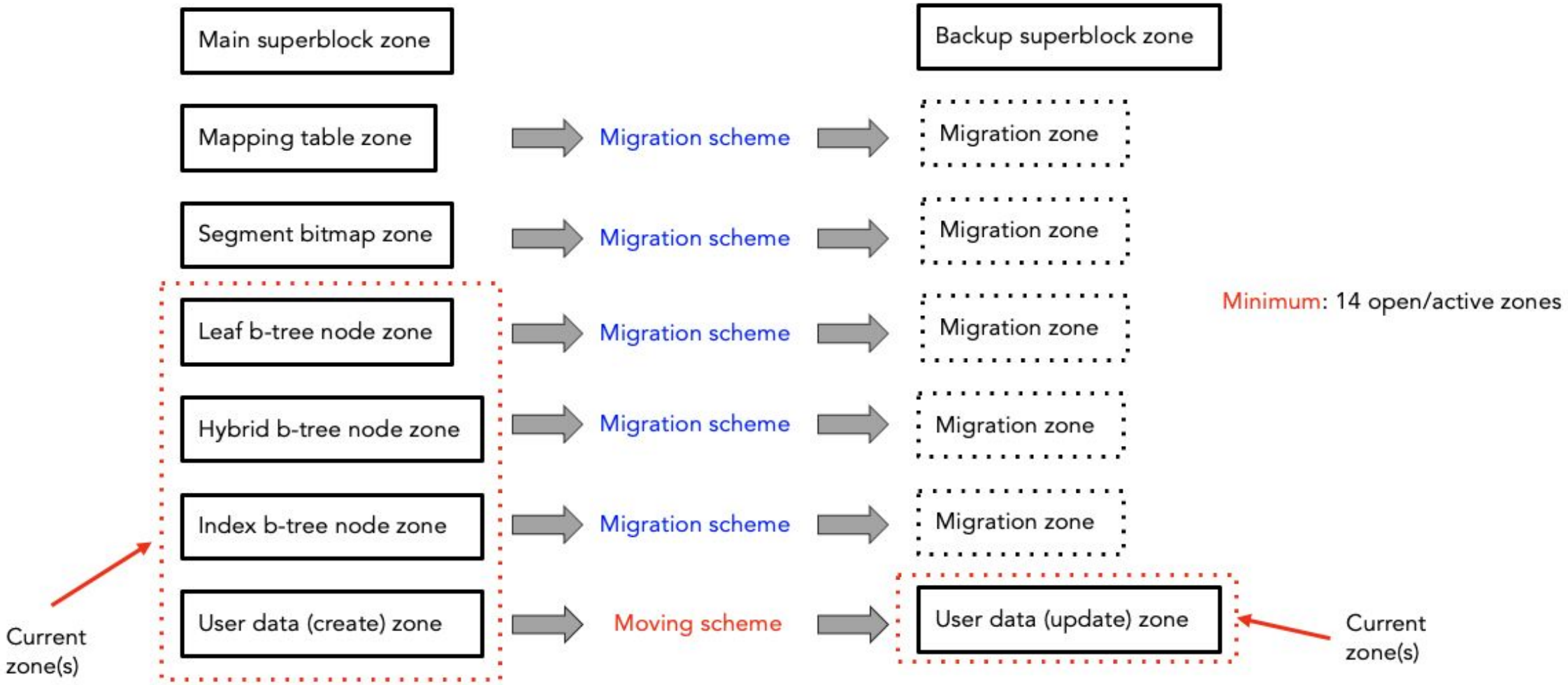
https://www.indiegogo.com/projects/ssdfs-zns-ssd-native-file-system/x/34131201#/

SSDFS architecture (metadata)



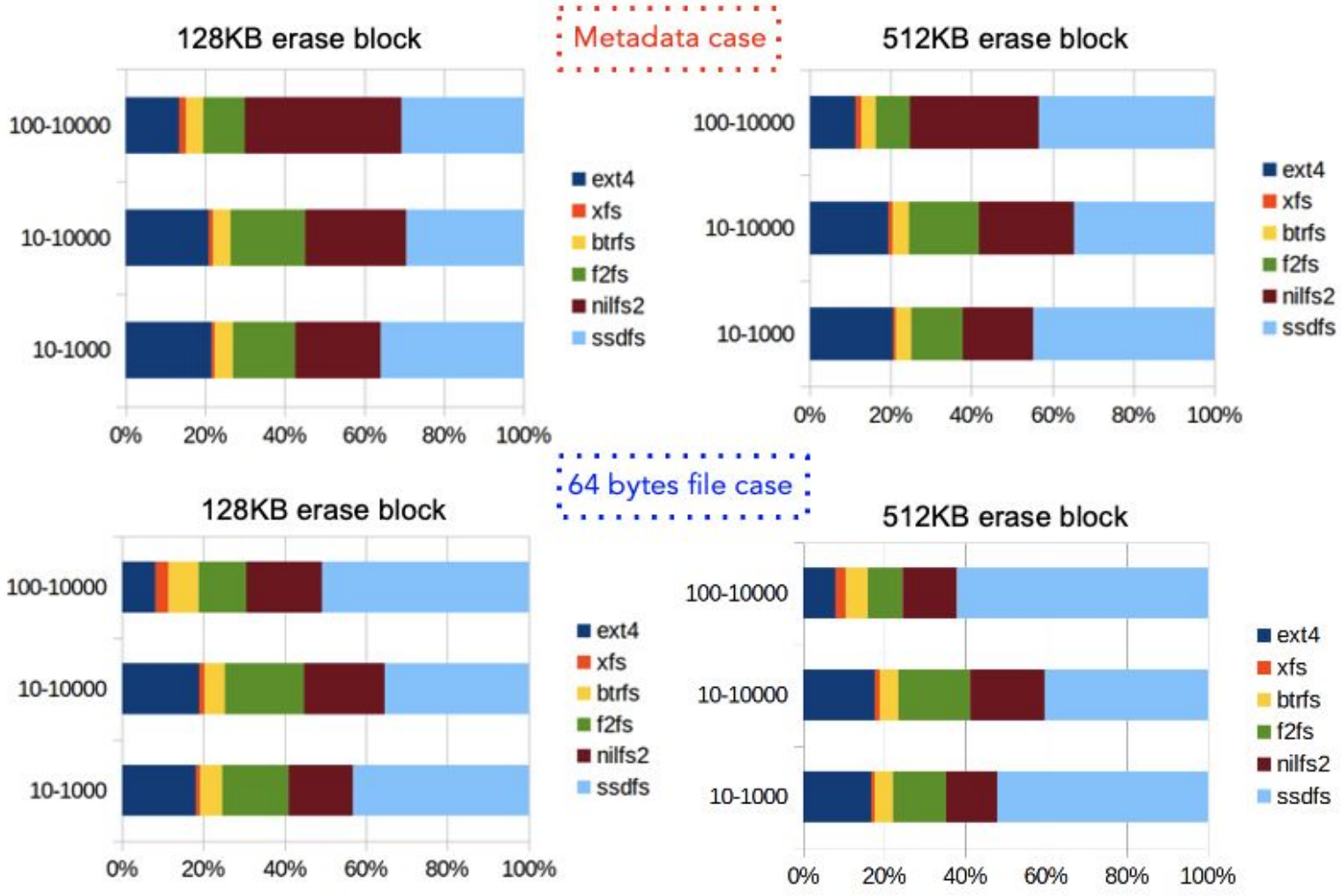
https://www.indiegogo.com/projects/ssdfs-zns-ssd-native-file-system/x/34131201#/

Current zones



https://www.indiegogo.com/projects/ssdfs-zns-ssd-native-file-system/x/34131201#/

SSD lifetime (create + update + delete)



Metadata case

	ext4	xfs	btrfs	f2fs	nilfs2
128KB	1.4x – 2.2x	17x – 44x	6.6x – 7.8x	1.5x – 2.9x	0.7x – 1.6x
512KB	1.7x – 3.8x	30x – 67x	8.5x – 12x	2x – 5x	1.3x – 2.5x

64 bytes file case

	ext4	xfs	btrfs	f2fs	nilfs2
128KB	1.8x – 6.2x	15x – 40x	6.8x – 7.9x	1.8x – 4.3x	1.7x – 2.7x
512KB	2.3x – 7.8x	24x – 60x	8.7x – 11x	2.2x – 7.2x	2.2x – 4.6x

SSDFS is capable to prolong SSD lifetime:

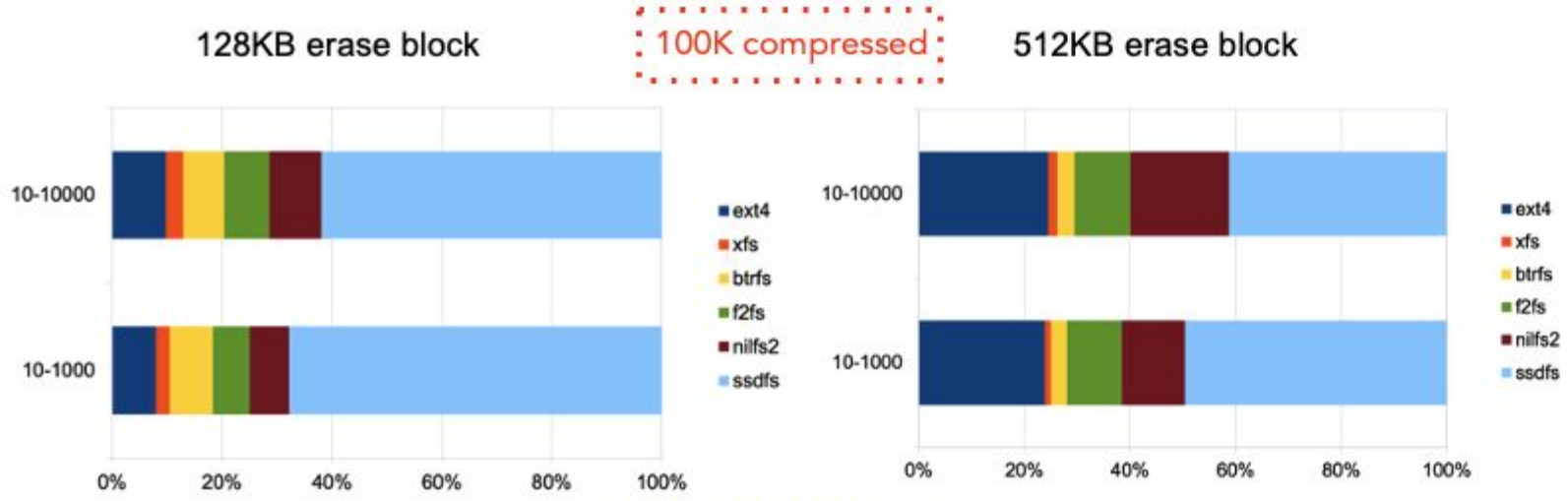
- (1.4x - 7.8x) compared with ext4
- (15x - 60x) compared with xfs
- (6x - 12x) compared with btrfs
- (1.5x - 7x) compared with f2fs
- (1x - 4.6x) compared with nilfs2

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

$$\text{Lifetime} = \frac{\text{Erase}_{\text{limit}}}{\text{Erase}_{\text{total}}}$$

https://www.indiegogo.com/projects/ssdfs-zns-ssd-native-file-system/x/34131201#/

SSD lifetime (create + update + delete)

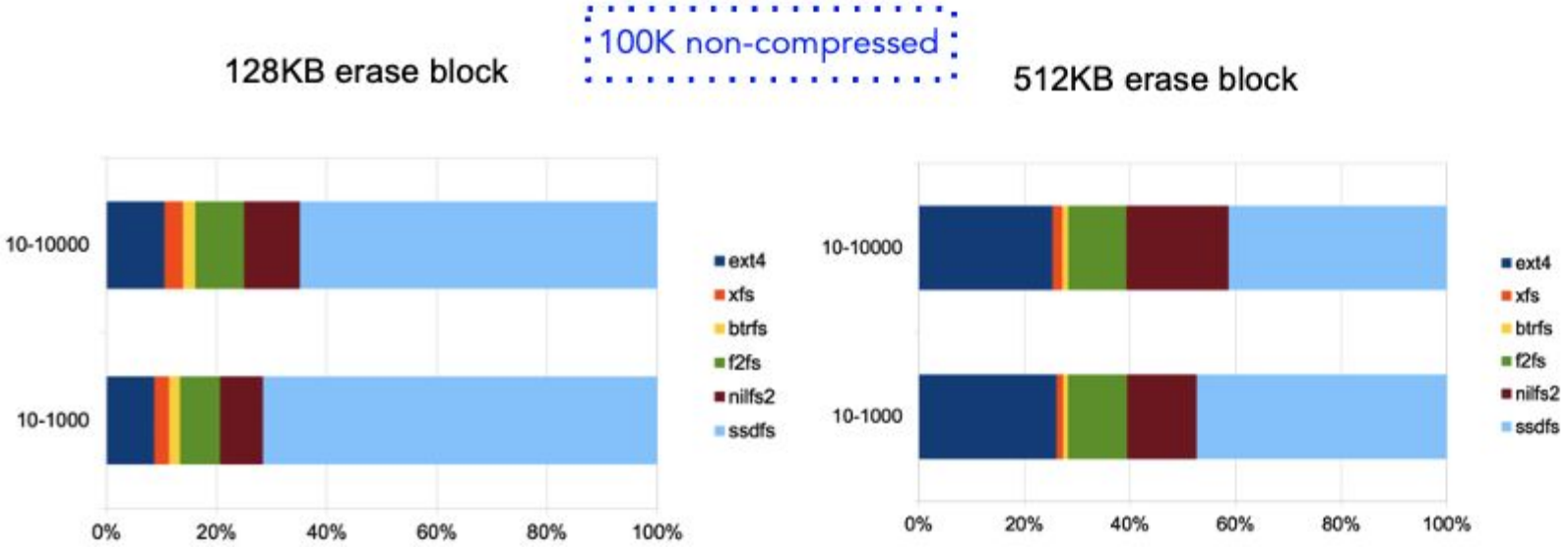


100K compressed

	ext4	xfs	btrfs	f2fs	nilfs2
128KB	6x - 8x	19x - 26x	8.3x - 8.6x	7.5x - 10x	6.5x - 9x
512KB	1.6x - 2x	23x - 41x	12x - 16x	3.8x - 4.8x	2.2x - 4x

100K non-compressed

	ext4	xfs	btrfs	f2fs	nilfs2
128KB	6.2x - 8.3x	18x - 26x	28x - 37x	7.3x - 9.7x	6x - 9x
512KB	1.6x- 1.8x	22x - 36x	38x - 57x	3.7x - 4.2x	2x - 3.6x



SSDFS is capable to prolong SSD lifetime:

- (1.6x - 8.3x) compared with ext4
- (18x - 40x) compared with xfs
- (8x - 50x) compared with btrfs
- (3x - 10x) compared with f2fs
- (2x - 9x) compared with nilfs2

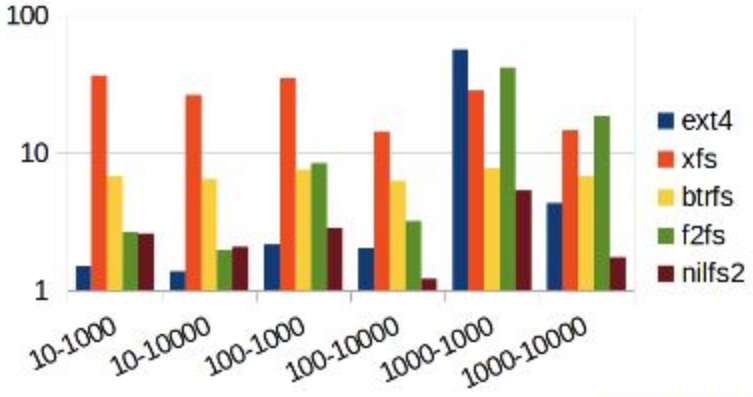
SSDFS can prolong SSD lifetime
2x - 10x for real-life use-cases

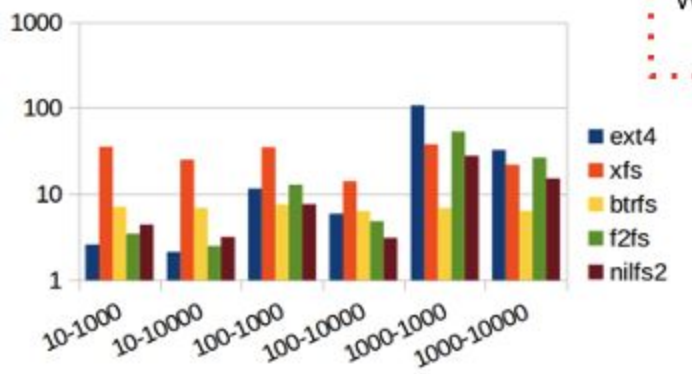
$$\text{Lifetime} = \frac{\text{Erase}_{\text{limit}}}{\text{Erase}_{\text{total}}}$$

https://www.indiegogo.com/projects/ssdfs-zns-ssd-native-file-system/x/34131201#/

Write amplification (create + update + delete)

Metadata case





$$\text{Write Amplification}_{\text{ratio}} = \frac{\text{FS(Write I/O + FS GC I/O)}}{\text{SSDFS(Write I/O + FS GC I/O)}}$$

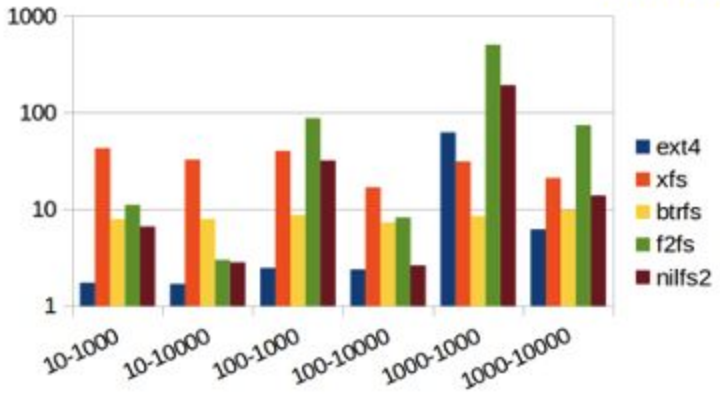
Metadata case

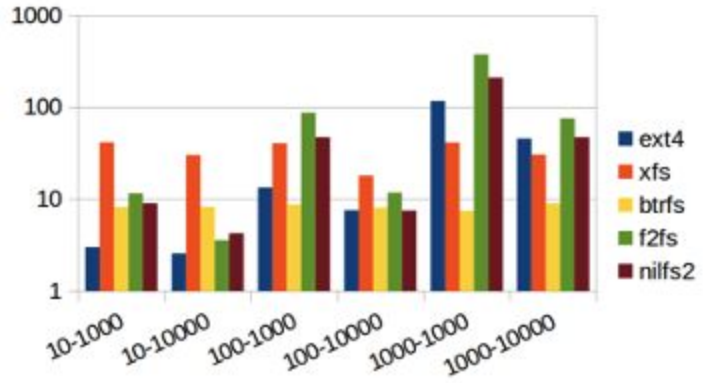
	ext4	xfs	btrfs	f2fs	nilfs2
128KB	1.3x – 56x	14x – 35x	6x – 7.7x	1.9x – 41x	1.2x – 5.3x
512KB	1.5x – 61x	18x – 41	7.4x – 8.7x	2.3x – 93x	1.6x – 13x
8MB	1.6x – 61x	16x – 42x	7.3x – 9.8x	2.9x – 502x	2.6x – 190x

w/o GC I/O

	f2fs	nilfs2
128KB	1.5x – 29x	0.6x – 2.1x
512KB	1.7x – 31x	0.8x – 2.3x
8MB	1.8x – 31x	0.7x – 2.3x

64 bytes file case





64 bytes file case

	ext4	xfs	btrfs	f2fs	nilfs2
128KB	2x – 107x	14x – 37x	6.3x – 7.5x	2.4x – 53x	3x – 27x
512KB	2.4x – 116x	18x – 40x	7.4x – 8.7x	2.9x – 108x	3.7x – 52x
8MB	2.5x – 116x	17x – 41x	7.4x – 8.9x	3.5x – 371x	4.2x – 208x

w/o GC I/O

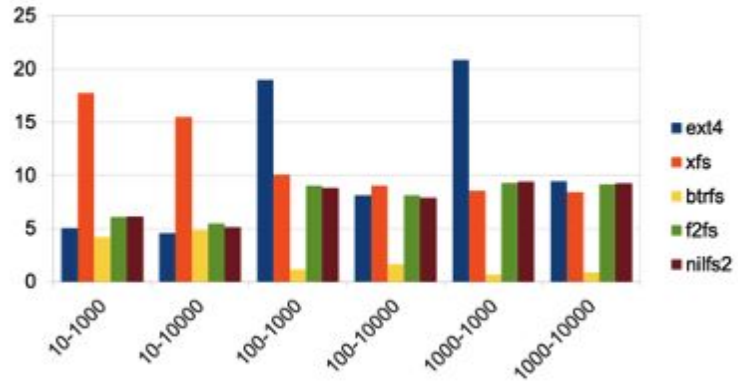
	f2fs	nilfs2
128KB	1.7x – 38x	1.7x – 20x
512KB	2x – 41x	1.9x – 22x
8MB	2.1x – 41x	2x – 22x

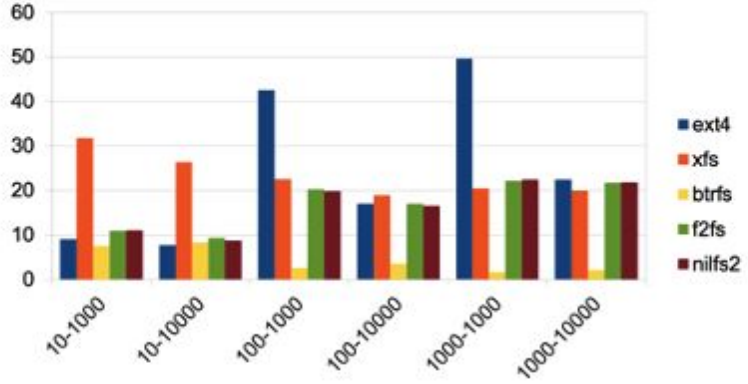
SSDFS is capable to decrease a write amplification issue 1.5x - 20x comparing with other file systems.

https://www.indiegogo.com/projects/ssdfs-zns-ssd-native-file-system/x/34131201#/

Write amplification (create + update + delete)

100K compressed





$$\text{Write Amplification}_{\text{ratio}} = \frac{\text{FS(Write I/O + FS GC I/O)}}{\text{SSDFS(Write I/O + FS GC I/O)}}$$

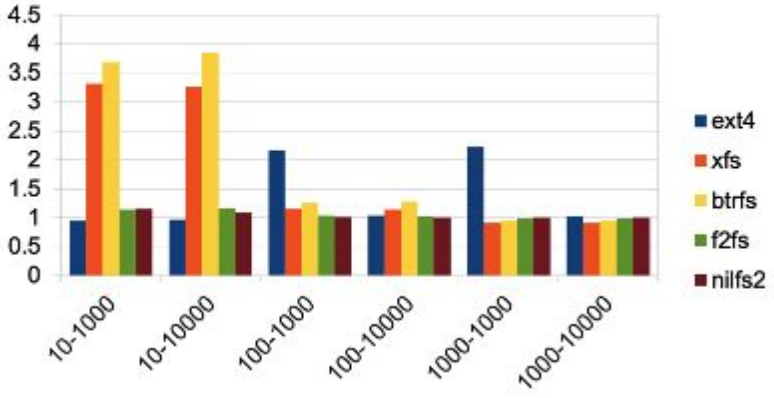
100K compressed

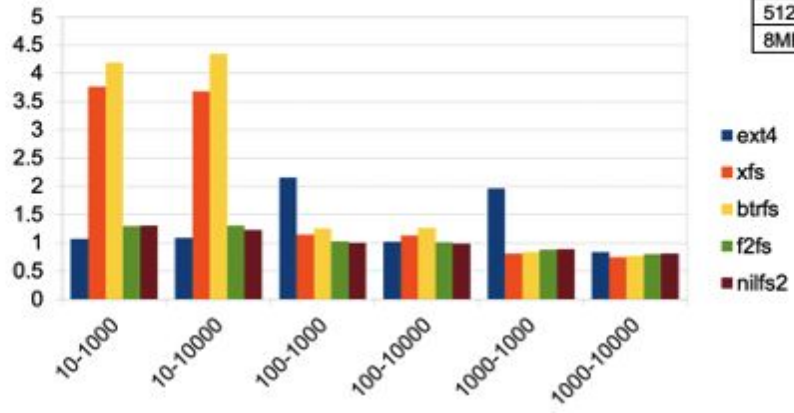
	ext4	xfs	btrfs	f2fs	nilfs2
128KB	4.5x - 20x	8x - 17x	0.6x - 4.8x	5x - 9x	5x - 9x
512KB	7x - 39x	16x - 27x	1.2x - 7x	8x - 17x	7x - 17x
8MB	7x - 49x	18x - 31x	1.5x - 8x	9x - 22x	8x - 22x

w/o GC I/O

	f2fs	nilfs2
128KB	4.5x - 8x	4x - 8x
512KB	6x - 16x	6x - 16x
8MB	7x - 20x	7x - 20x

100K non-compressed





100K non-compressed

	ext4	xfs	btrfs	f2fs	nilfs2
128KB	0.9x - 2x	0.9x - 3.3x	0.9x - 3.8x	0.9x - 1.1x	0.9x - 1.1x
512KB	1x - 2.4x	0.9x - 3.6x	1x - 4x	1x - 1.2x	1x - 1.2x
8MB	0.8x - 2x	0.7x - 3.7x	0.7x - 4x	0.8x - 1.3x	0.8x - 1.3x

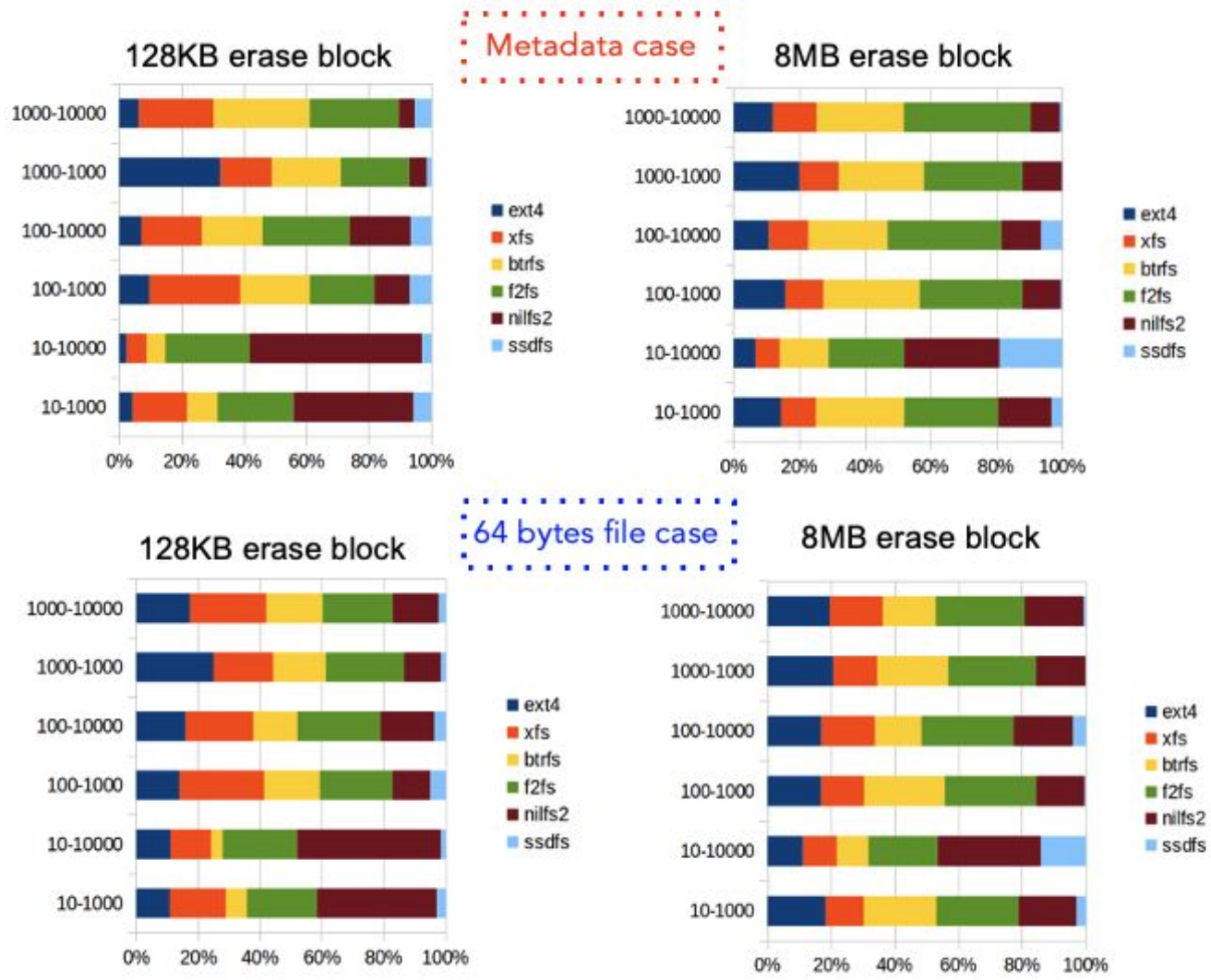
w/o GC I/O

	f2fs	nilfs2
128KB	0.9x	0.9x
512KB	1x	1x
8MB	0.7x - 1x	0.7x - 1.1x

SSDFS is capable to decrease a write amplification issue 1.5x - 20x comparing with other file systems (compressed case).

https://www.indiegogo.com/projects/ssdfs-zns-ssd-native-file-system/x/34131201#/

Payload (create + update + delete) - erase blocks



$$Payload_{ratio} = \frac{FS_{payload}}{SSDFS_{payload}}$$

Metadata case

	ext4	xfs	btrfs	f2fs	nilfs2
128KB	0.7x – 21x	2.1x – 10x	1.7x – 14x	2.9x – 14x	0.9x – 18x
512KB	0.4x – 80x	1x – 33x	1x – 39x	3.7x – 63x	2x – 11x
8MB	0.3x – 315x	0.3x – 189x	0.7x – 409x	1.2x – 472x	1.5x - 189x

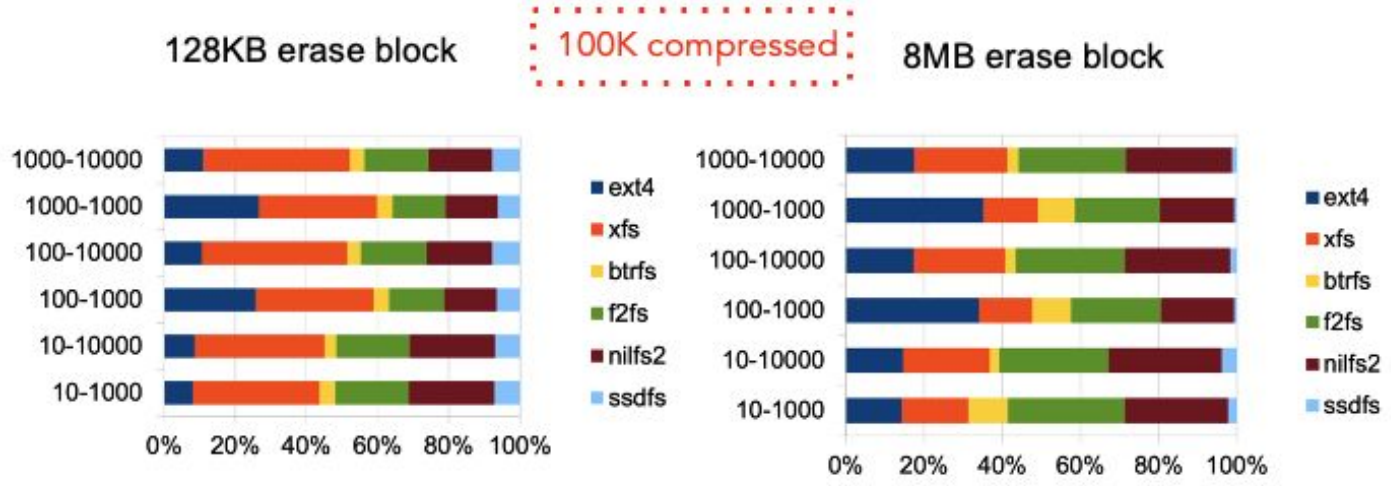
64 bytes file case

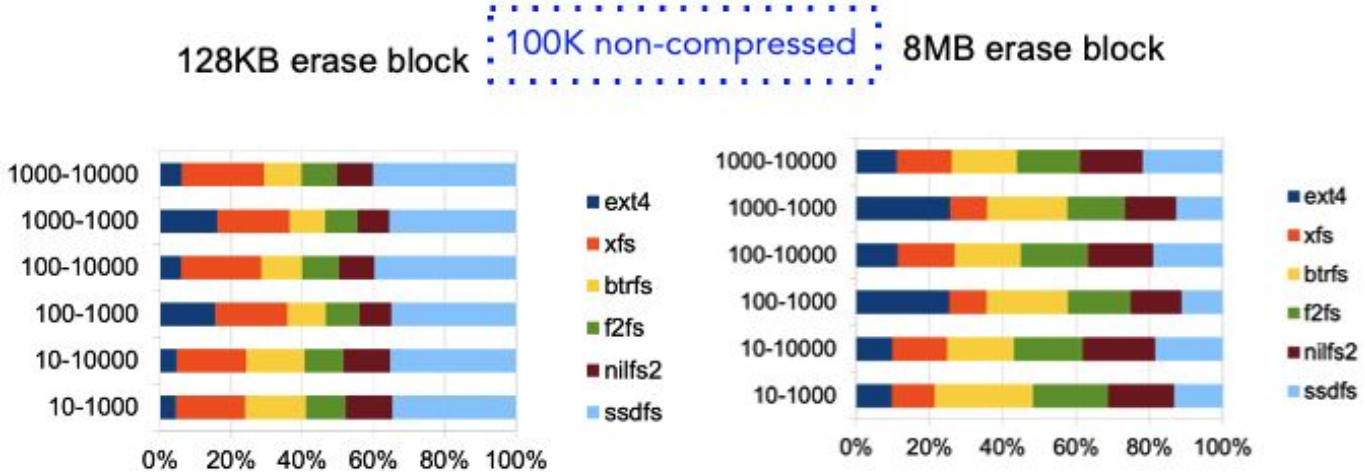
	ext4	xfs	btrfs	f2fs	nilfs2
128KB	2.7x – 15x	5.4x – 12x	2.3x – 10x	4.7x – 16x	2.4x – 29x
512KB	2.5x – 64x	3.4x – 40x	1.4x – 27x	5.4x – 68x	8.4x – 31x
8MB	0.7x – 248x	0.7x – 165x	0.7x – 268x	1.5x – 330x	2.2x – 186x

SSDFS is capable to create smaller (2x - 20x) payload. However, SSDFS can generate more payload for some use-cases (for example, 10-10000, 100-10000) compared with ext4, xfs, btrfs.

https://www.indiegogo.com/projects/ssdfs-zns-ssd-native-file-system/x/34131201#/

Payload (create + update + delete) - erase blocks





$$Payload_{ratio} = \frac{FS_{payload}}{SSDFS_{payload}}$$

100K compressed

	ext4	xfs	btrfs	f2fs	nilfs2
128KB	1.1x – 4.2x	4.9x – 5.3x	0.4x – 0.7x	2.3x – 2.8x	2.2x – 3.4x
512KB	5x – 27x	9x – 15x	0.7x – 2.2x	12x – 17x	12x - 17x
8MB	3.8x – 50x	5.7x – 18x	0.6x – 13x	7x – 31x	7.5x – 27x

100K non-compressed

	ext4	xfs	btrfs	f2fs	nilfs2
128KB	0.1x – 0.4x	0.5x	0.2x – 0.4x	0.2x – 0.3x	0.2x – 0.3x
512KB	0.5x - 1.9x	0.8x – 0.9x	0.9x – 1x	1x – 1.1x	1x – 1.1x
8MB	0.5x – 2.2x	0.7x – 0.9x	0.8x – 2x	0.7x – 1.5x	0.7x – 1.3x

SSDFS is capable to create smaller (2x - 20x) payload. However, SSDFS can generate more payload for some use-cases, data type, and erase block sizes.

https://www.indiegogo.com/projects/ssdfs-zns-ssd-native-file-system/x/34131201#/

Read disturbance (create + update + delete)

Metadata case

	ext4	xfs	btrfs	f2fs	nilfs2
128KB	0.1x – 0.5x	1.3x – 2.4x	0.2x – 0.9x	0.1x – 7.8x	4.5x – 27x
512KB	0.1x – 0.8x	0.8x – 4.3x	0.1x – 1.4x	0.06x – 12x	2.8x – 44x
8MB	0.05x – 0.9x	0.4x – 4x	0.06x – 1.8x	0.03x – 15x	1.5x – 53x

64 bytes file case

	ext4	xfs	btrfs	f2fs	nilfs2
128KB	0.2x – 3.4x	1.7x – 4.8x	0.2x – 1x	0.1x – 6.9x	4.9x – 27x
512KB	0.1x – 5x	0.9x – 7.3x	0.1x – 1.5x	0.08x – 10x	3.5x – 41x
8MB	0.05x – 5.6x	0.4x – 8x	0.06x – 1.6x	0.03x – 11x	1.5x – 45x

SSDFS generates smaller amount of read I/O
- (1.5x - 50x) compared with nilfs2
- (1x - 8x) compared with xfs

SSDFS generates bigger amount of read I/O:
- (1x - 20x) compared with ext4
- (1x - 16x) compared with btrfs
- (1x - 26x) compared with f2fs

SSDFS generates **more read I/O** for bigger erase blocks with smaller partial logs. **Offsets translation table** is the main contributor to this issue.
Solution: store full offset translation table in every log + compress offset translation table.

https://www.indiegogo.com/projects/ssdfs-zns-ssd-native-file-system/x/34131201#/

Current status

- Mount logic – **stable**
- Mapping table – **stable**
- Segment bitmap – **stable**
- Migration scheme – **stable**
- Inodes tree – **stable**
- Dentries tree – **mostly stable**
- Extents tree – **not fully stable**
- Xattrs tree – **not fully stable**
- Delta-encoding – **not fully stable**

- 8K/16K/32K logical block support – **not stable**
- Multiple erase blocks in segment – **not stable**
- Moving scheme – **not stable**

- Folio adoption - **not fully implemented**
- PEB inflation model – **not fully implemented**
- Deduplication – **not fully implemented**
- Snapshots – **not fully implemented**
- Recoverfs – **not fully implemented**

- Fsck – **not implemented**
- Post-deduplication delta-encoding – **not implemented**

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

Support open-source project

https://www.indiegogo.com/projects/ssdfs-zns-ssd-native-file-system/x/34131201#/

THANK YOU

QUESTIONS???

slava@dubeyko.com

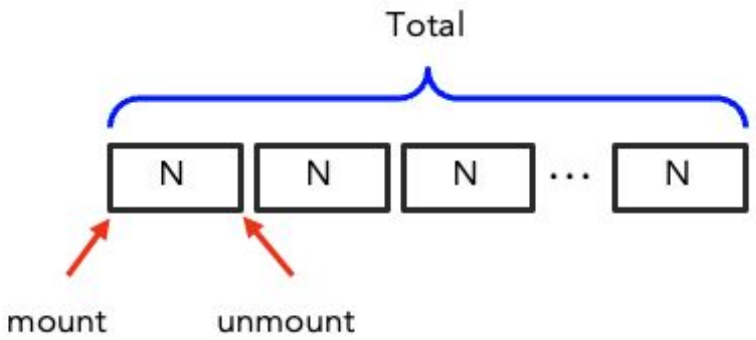
https://www.indiegogo.com/projects/ssdfs-zns-ssd-native-file-system/x/34131201#/

Testing use-cases

Metadata	User data	
Create empty file	Create file	64 bytes
		16KB
		100KB
Update empty file	Update file	64 bytes
		16KB
		100KB
Delete empty file	Delete file	64 bytes
		16KB
		100KB

SSDFS	
Erase block size	128KB
	512KB
	8MB

N	Total
10	1000
10	10000
100	1000
100	10000
1000	1000
1000	10000



Testing sequence:
- format partition (mkfs - default settings)
- blktrace <partition>
- while (iterations < (Total/N)) {
  mount();
  while (items < N) {
    execute_use_case();
  }
  unmount();
 }
- stop blktrace

Methodology

$$Lifetime = \frac{Erase_{limit}}{Erase_{total}}$$

$$Erase_{limit} = Capacity_{EB} * Erase\ Block_{limit}$$

$$Erase_{total} = Erase_{FTL\ GC} + Erase_{TRIM} + Erase_{FS\ GC} + Erase_{read\ disturbance} + Erase_{retention}$$

$$Erase_{FTL\ GC} = Write^{I/O}_{EB} - Erase\ Block_{unique}$$

$$Erase_{FS\ GC} = Payload_{EB} - Valid\ Data_{EB}$$

$$Erase_{read\ disturbance} = \frac{Read^{I/O}_{EB}}{Threshold_{disturbance}}$$

$$Payload_{EB} = Erase\ Block_{unique} - TRIM_{EB}$$

$$Erase_{retention} = \frac{Time_{use\text{-}case}}{3\ months} * Payload_{EB}$$