SNIA DEVELOPER CONFERENCE

SDC 25

By Developers FOR Developers

Highly scalable, masterless, distributed file system at Rubrik

Abhinav Agarwal, Sr. Software Engineer at Rubrik
Avi Ganguli, Sr. Software Engineer at Rubrik

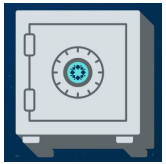
Rubrik at a glance

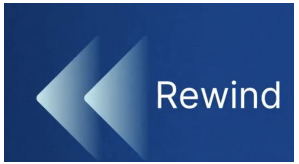
Scale

- Number of customers > 6k
- Front end bytes > 42 ExaByte+

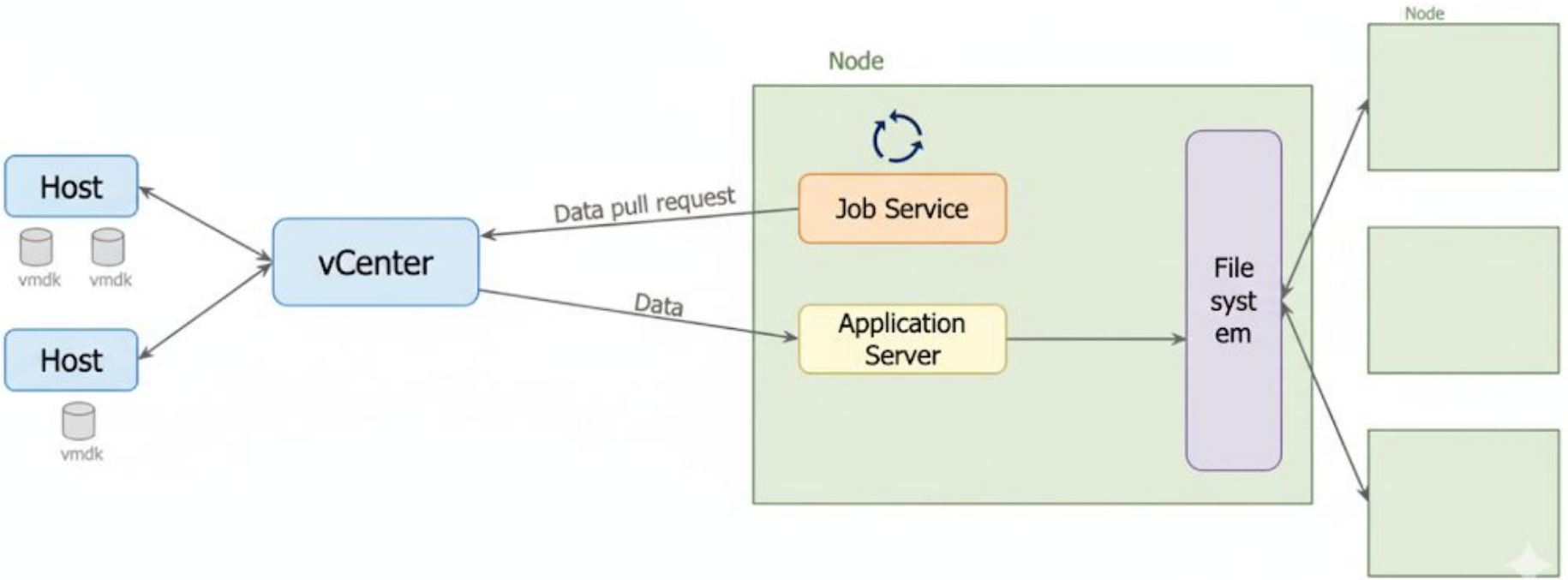
Offerings

- Live Mount
- Continuous Threat Monitoring
- Ransomware Recovery
- Agent Rewind
- Annapurna
- Replication
- Archival & more…

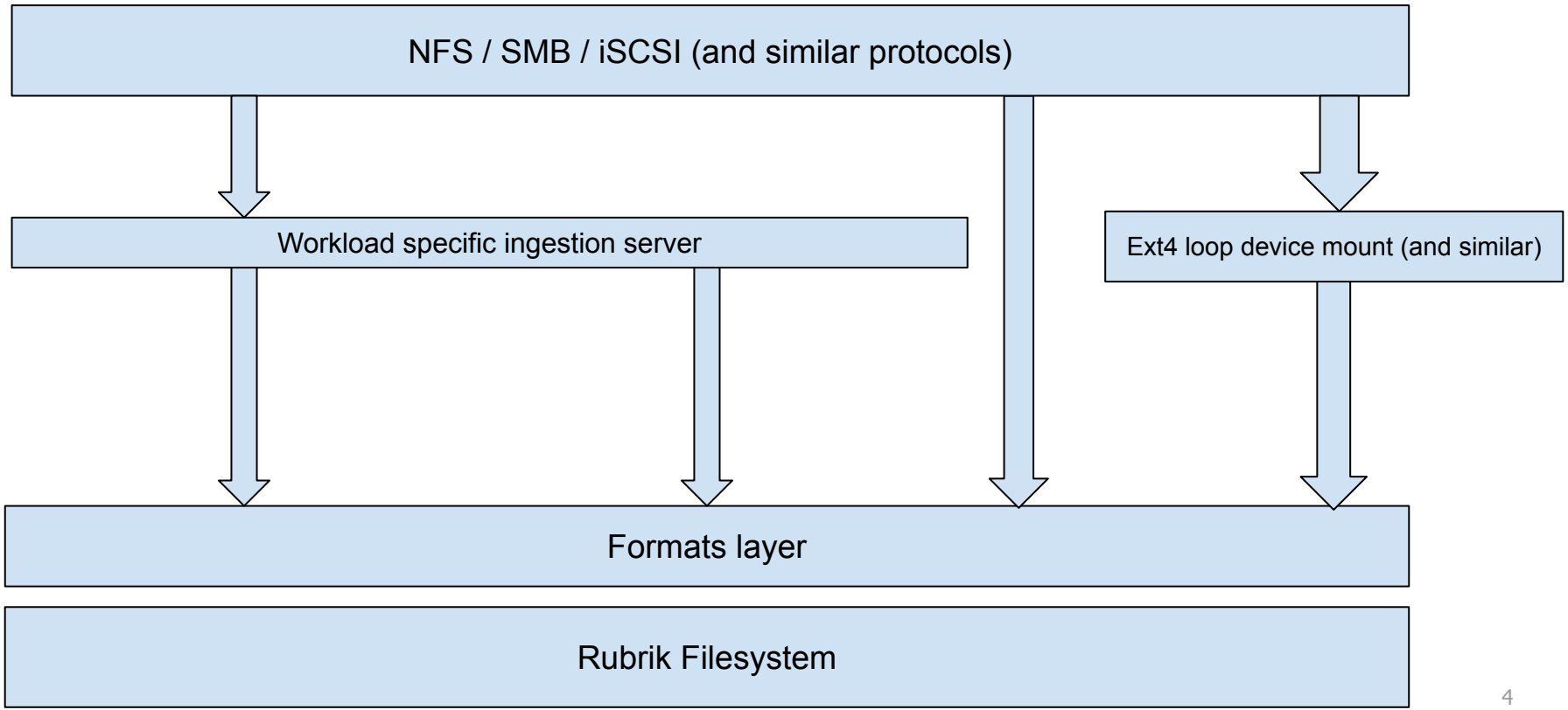




VMWare protection workflow



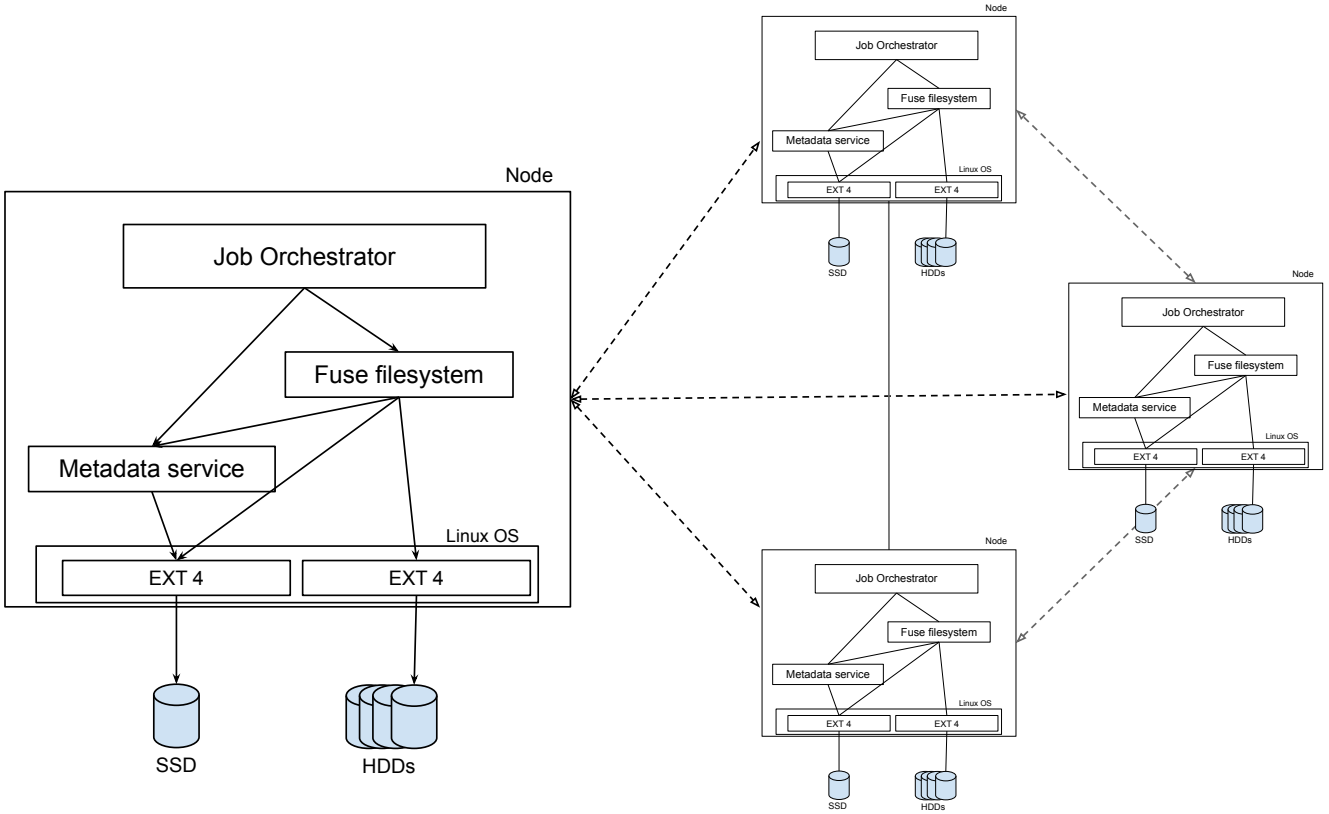
IO Path



High level requirements

- Support multiple hardware configs
- Various platforms: virtual, on-prem & cloud
- Use standard posix API
- Horizontal performance scaling
- Data resilience and integrity

Component Design

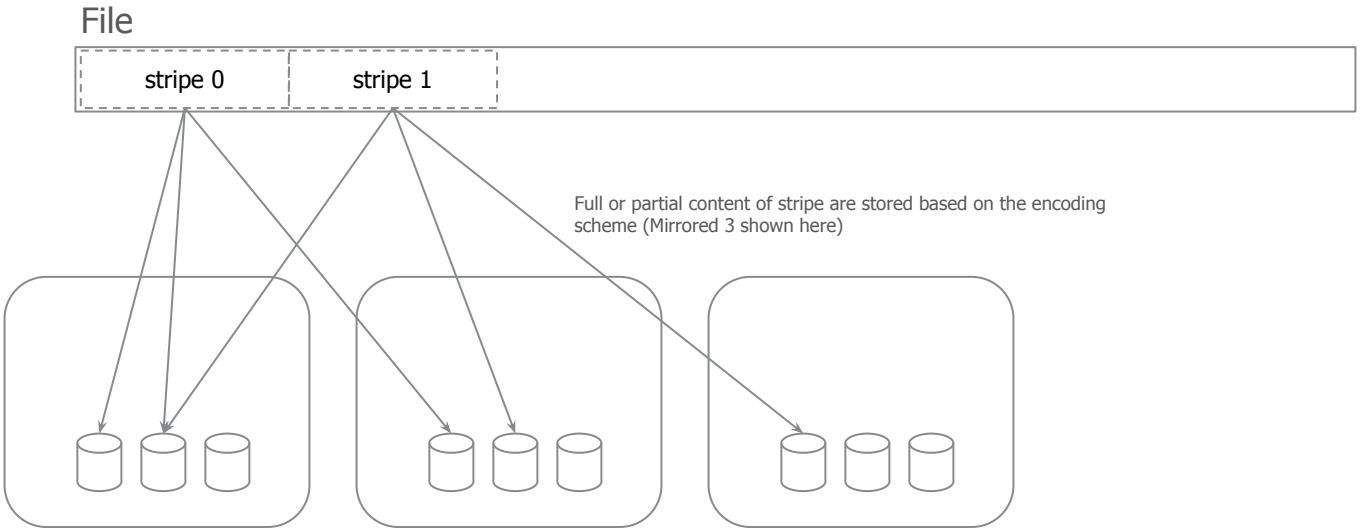


Filesystem requirements

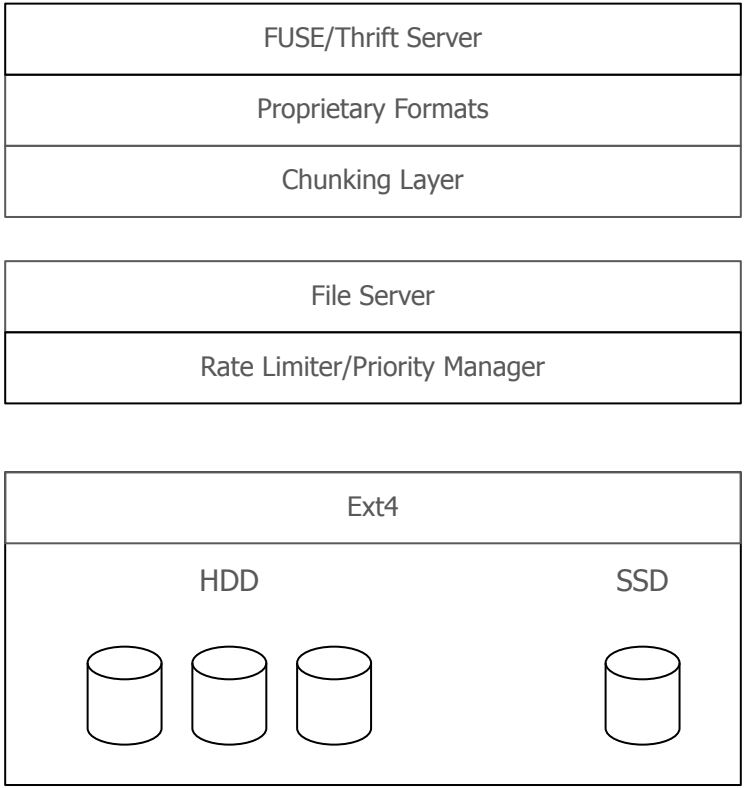
1. Data integrity
2. Data resilience
   a. Rack level failure
   b. Node level failure
   c. Disk level failure
2. Background operations
   a. Rebuild any corrupted chunks
   b. Up-replicate any lost chunks
   c. Up-replicate for higher resilience
   d. Space Rebalance
3. Online cluster ops
   a. Node add/remove
   b. Disk add/remove
4. IO prioritization
5. Advanced data services
   a. Optimal data storage
   b. Compression
   c. Deduplication
   d. Encryption

File Layout

- A file is made up of several fixed sized “stripes”.
- A stripe is laid out across the nodes in “chunk” files with additional metadata and checksums for data integrity and resilience.
- Layout depends on encoding: Mirrored, Reed Solomon etc.



Filesystem In Depth

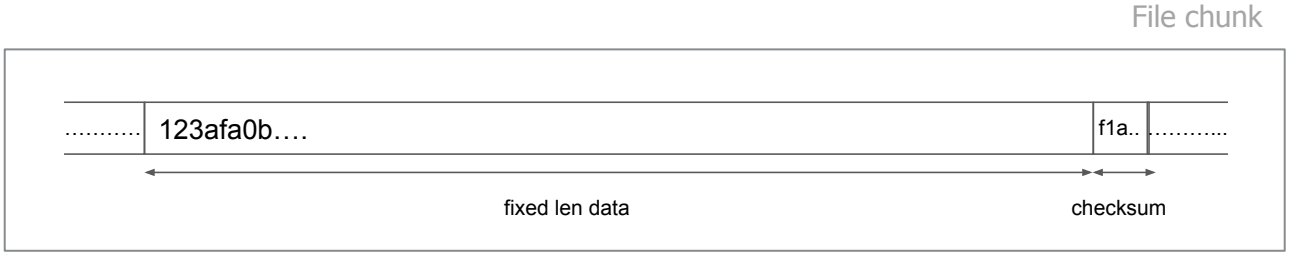


Data Integrity

We must ensure data integrity on every read (even for small random reads)

We deploy checksums at multiple layers:

- Inline checksum: Embedded in every file after every ~4KiB data. Validated on every read.

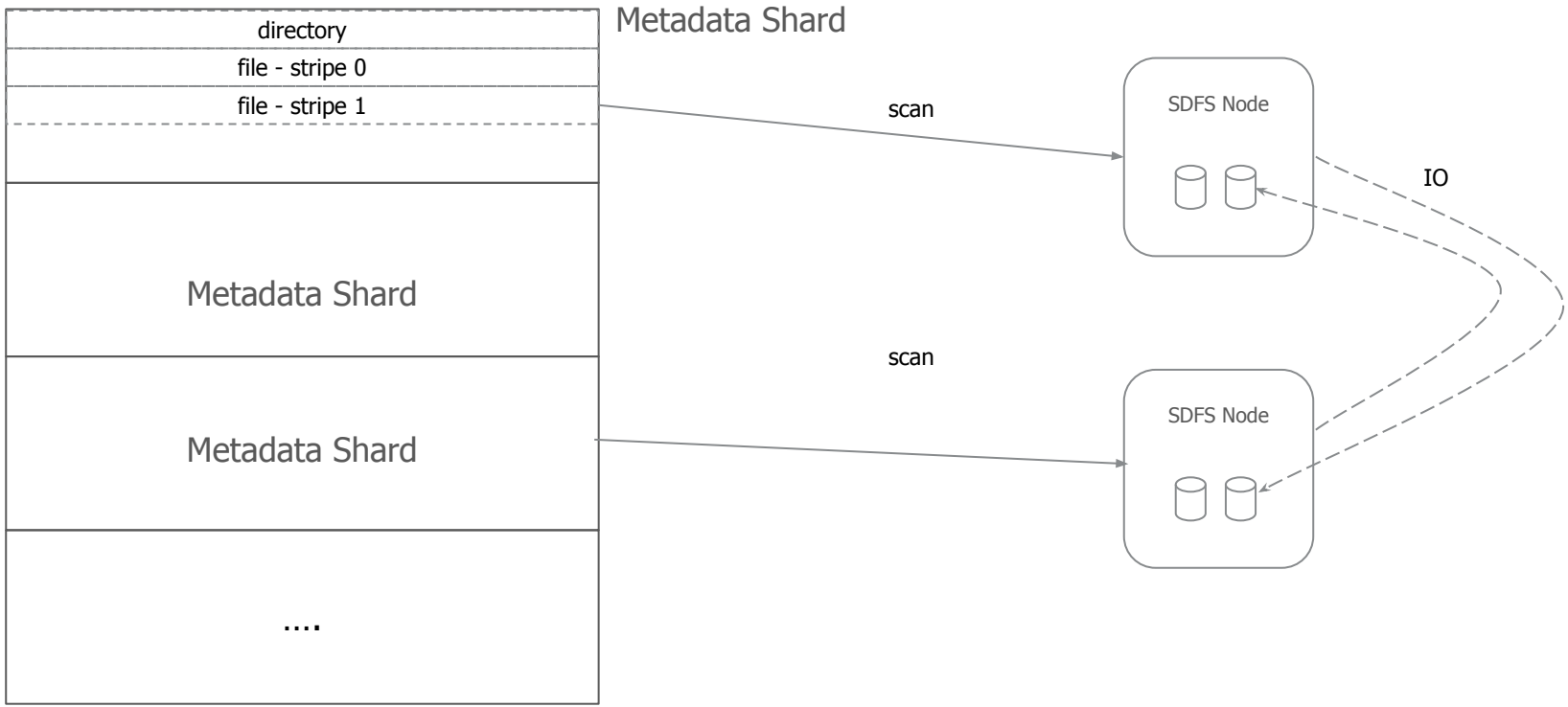


- Chunk checksum: Checksum of fixed size chunk (~32 MiB). Validated in background.
- Fingerprint: SHA1 of logical chunk (~64 KiB).
- This is in addition to checksums at higher layers.

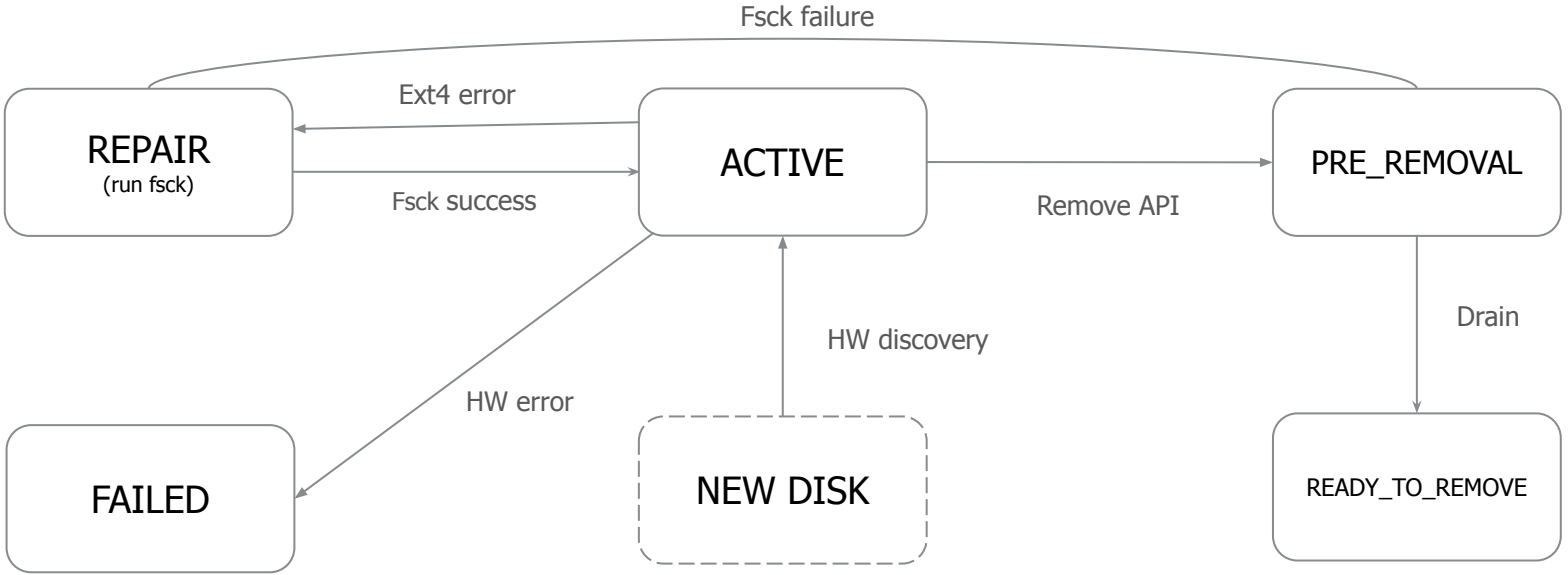
Additional Background Operations

- Garbage collection
- Reporting filesystem health
- Metadata validation
- Compaction
- Dedup

Background Scan Layout



Online Cluster Operations

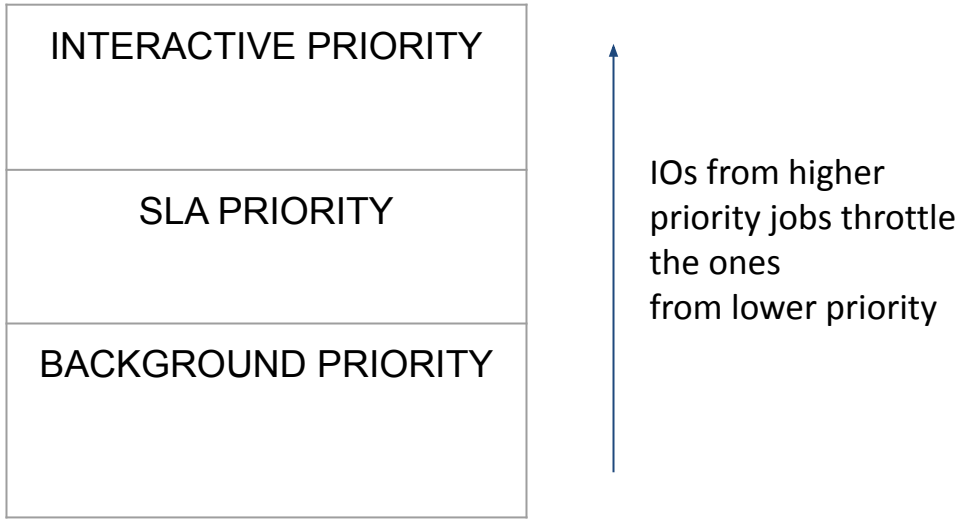


IO Priority

At any time, there can be several jobs running concurrently

1. Background jobs like data rebalance
2. SLA driven jobs like object protection or indexing
3. Manually triggered (interactive) jobs like restore

IO Priority



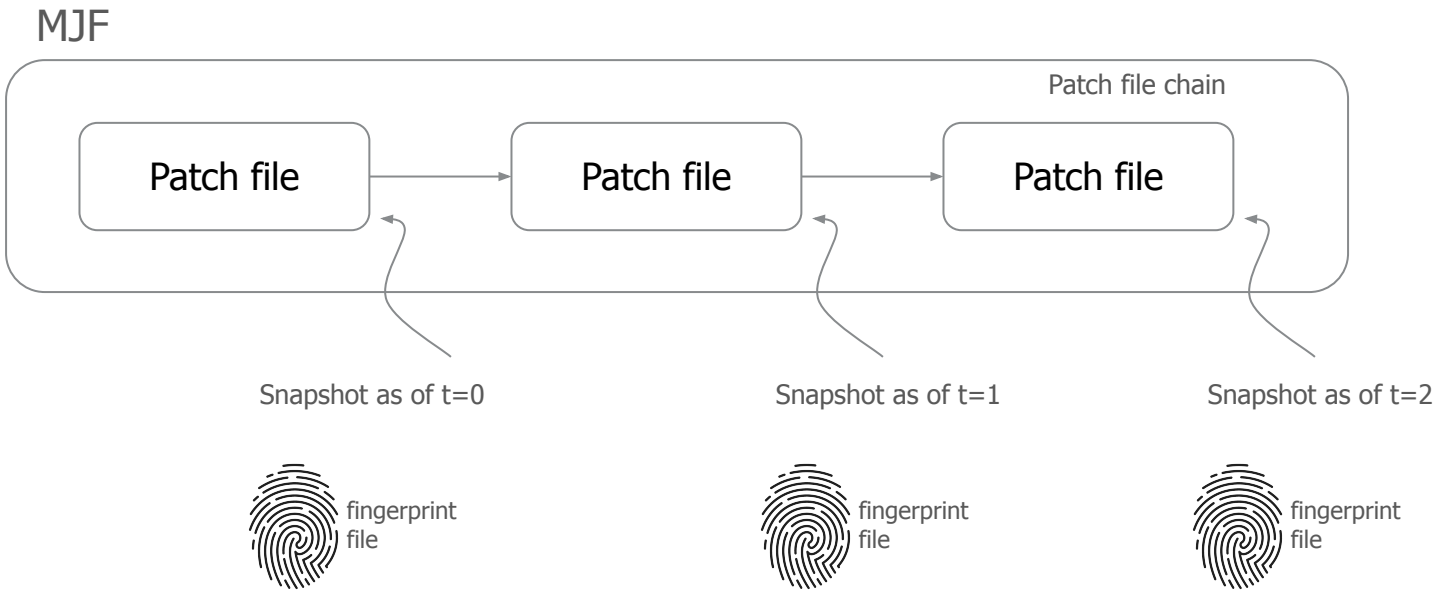
Data Formats Layer

To efficiently store periodic backups, our filesystem supports advanced data structures.

1. Patch File: A key value store (similar to SSTable) optimized for on-disk reads. We store offset → data mapping in this file.
2. Fingerprint File: This contains checksum (SHA1) in 64 KiB chunks of logical data. This is used for data integrity, de-duplication and computing incremental.
3. Journal File: This is a simple log-structured file where we append (offset, data). This should be later converted to patch file for optimized reads.

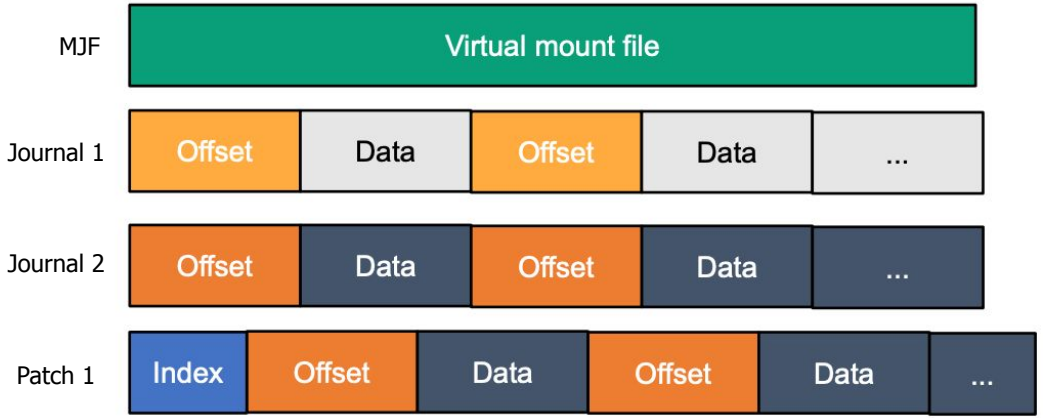
16

Data Formats Layer

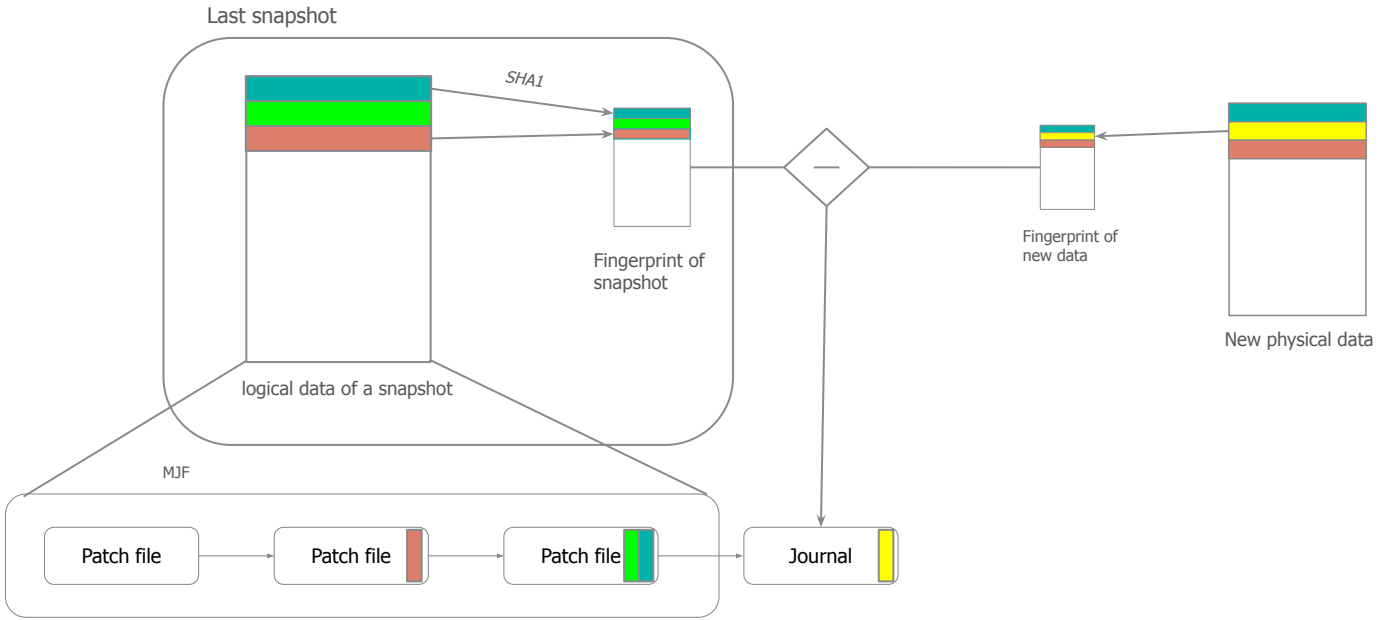


Read across chain

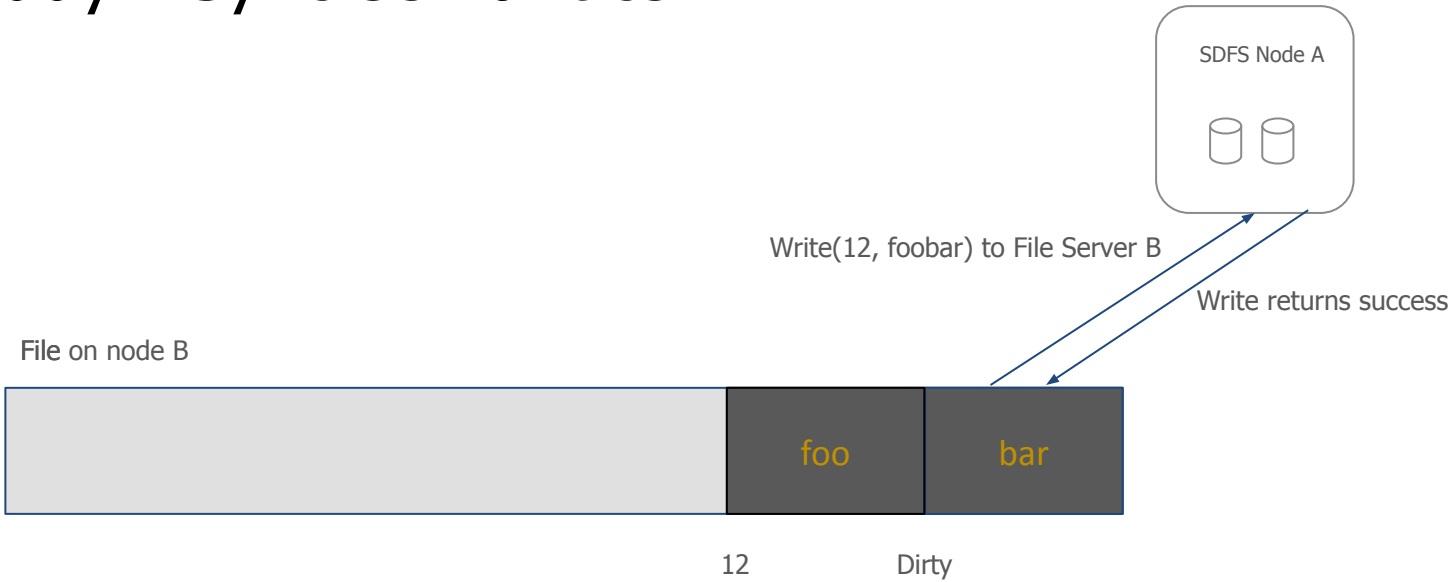
Read may have to traverse down the patch file chain



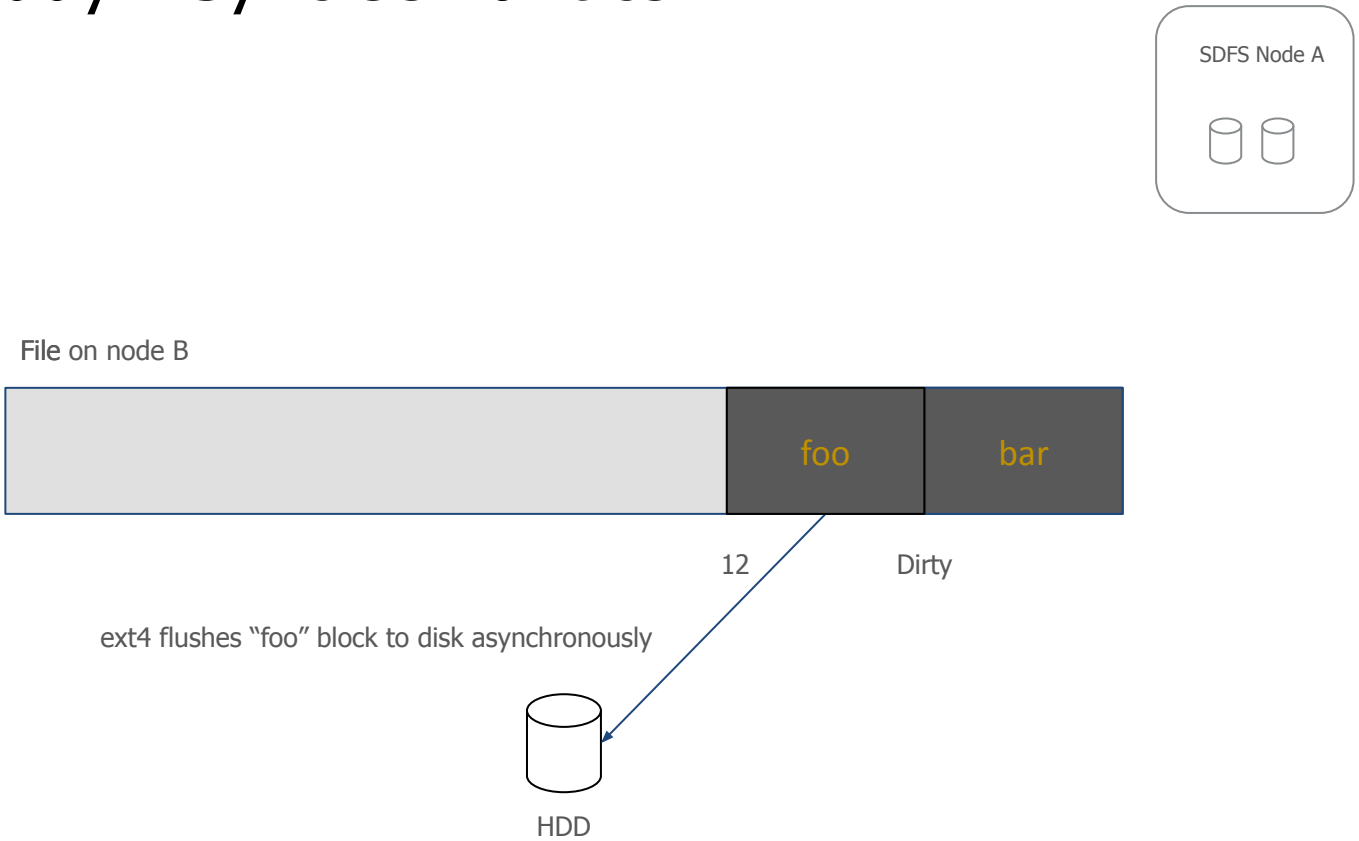
Computing incremental



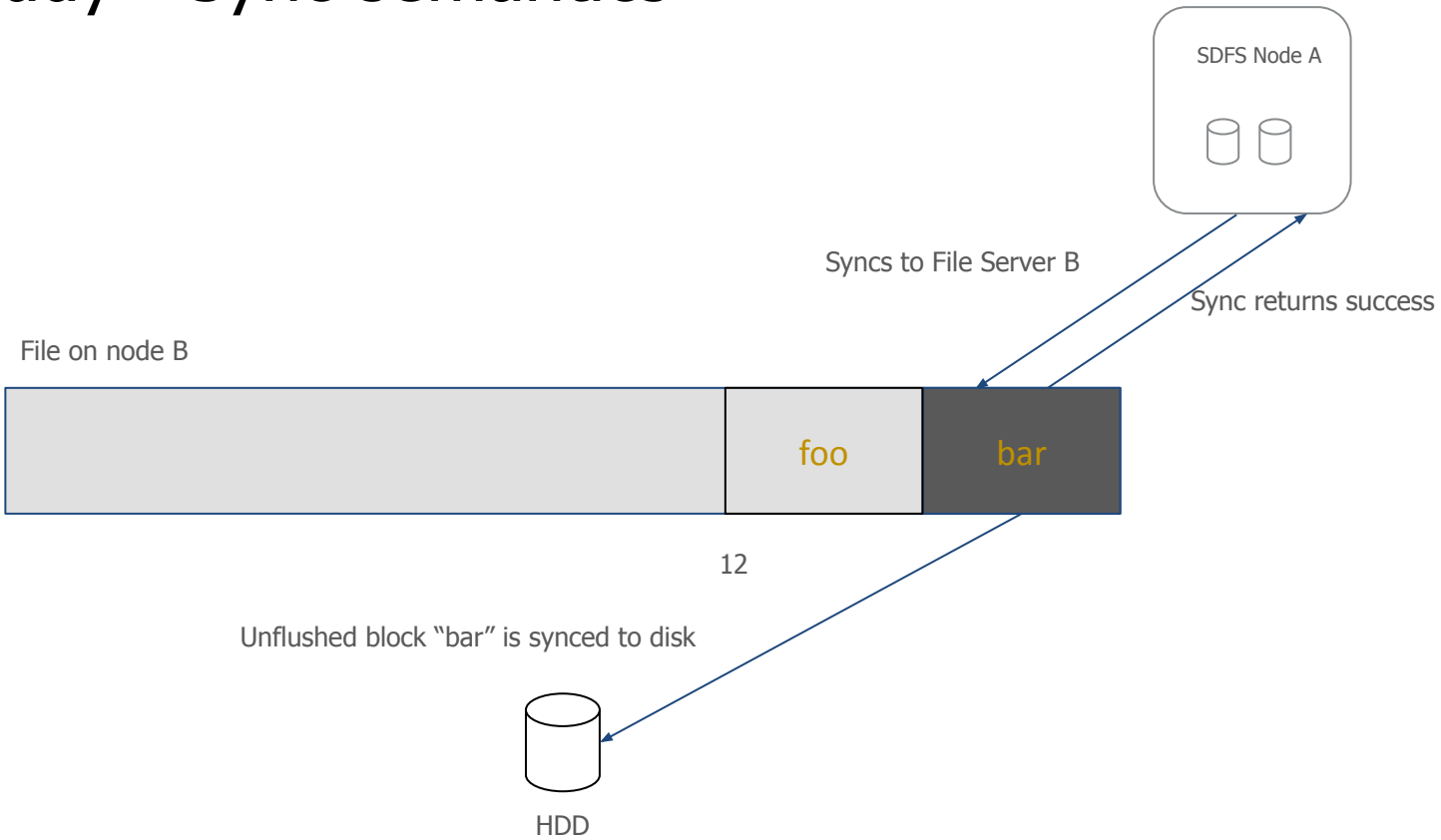
Case Study - Sync semantics



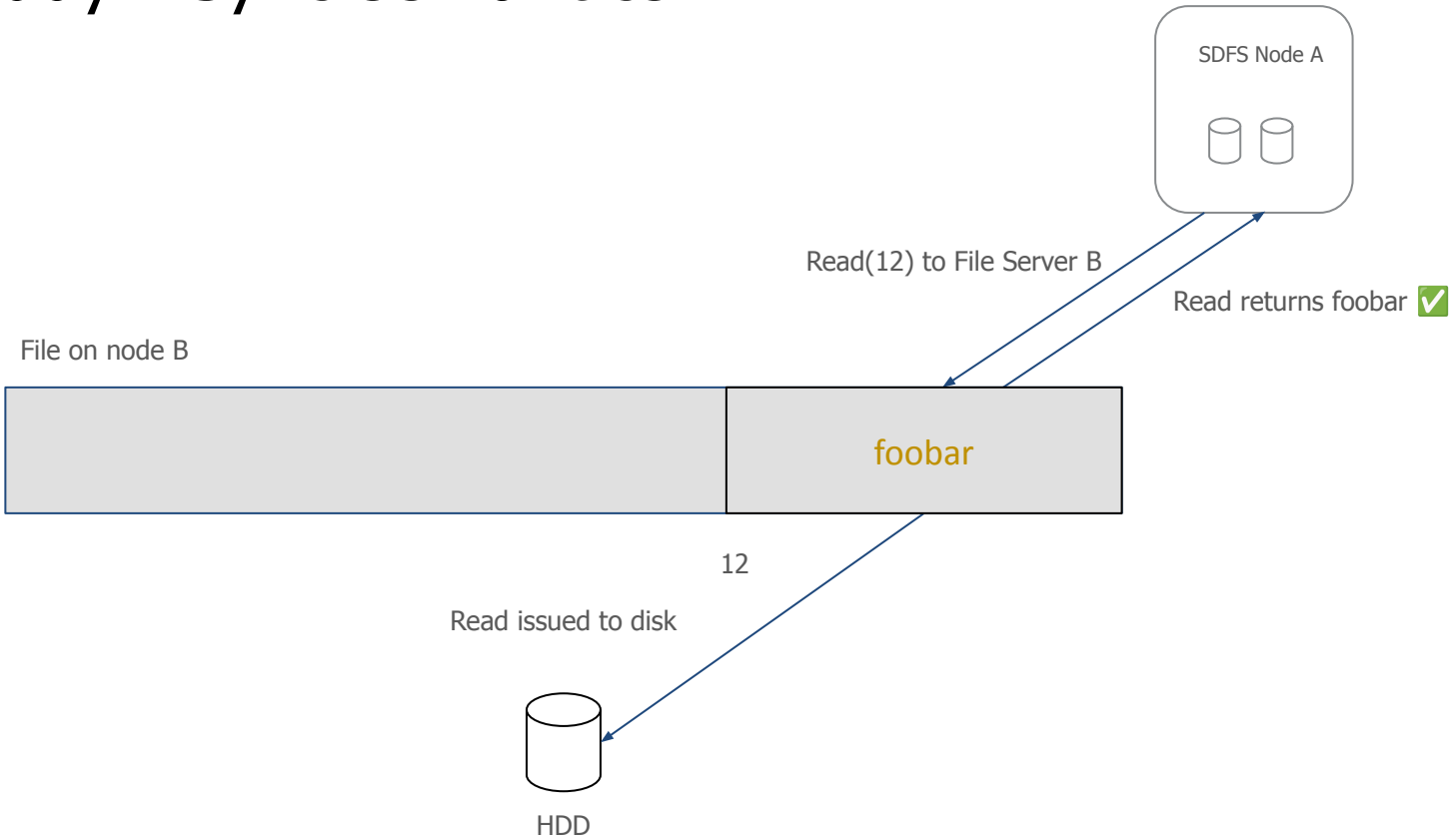
Case Study - Sync semantics



Case Study - Sync semantics

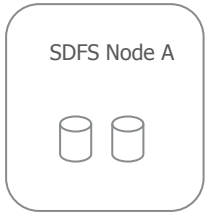


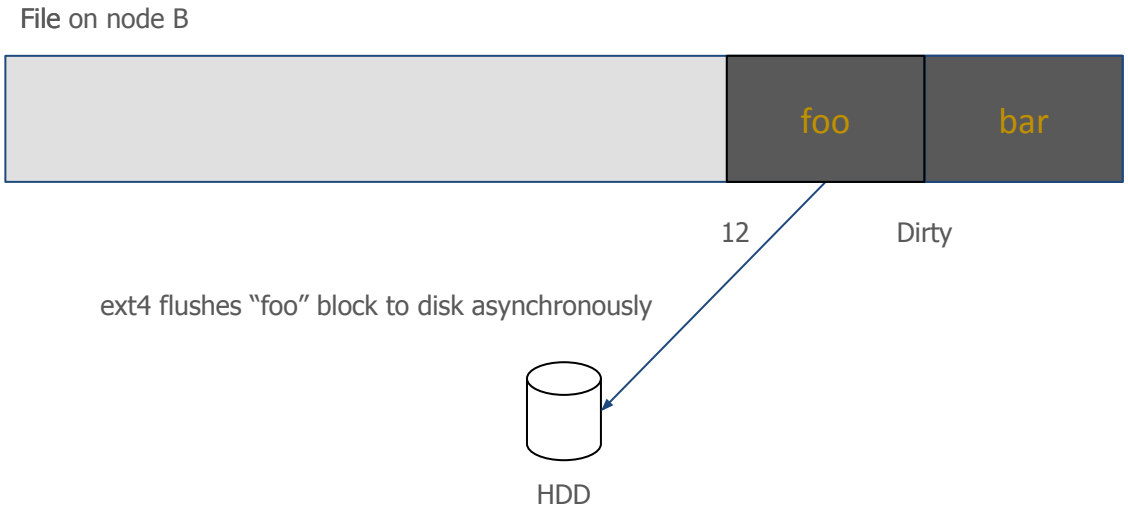
Case Study - Sync semantics



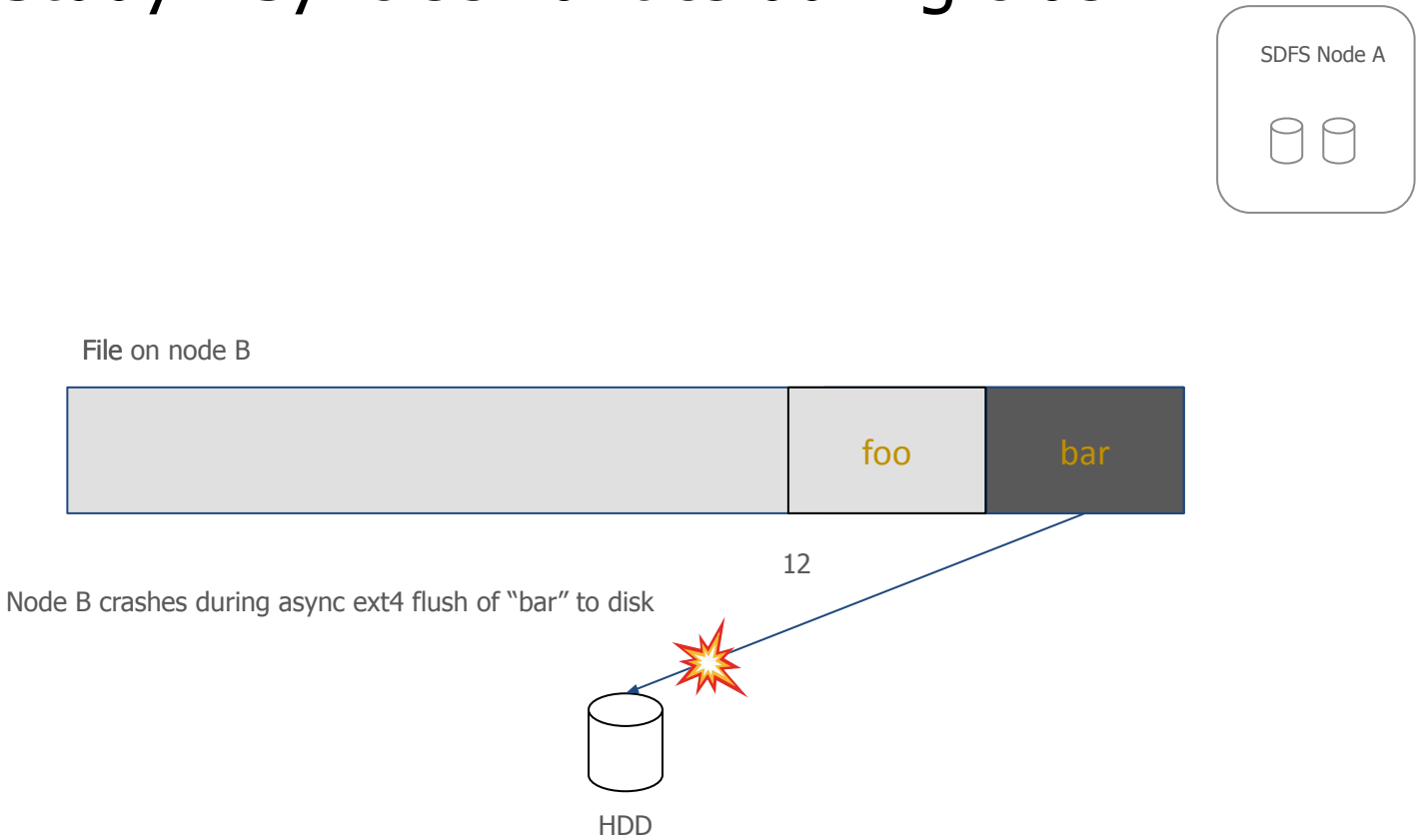
Case Study - Sync semantics during crash

Case Study - Sync semantics during crash

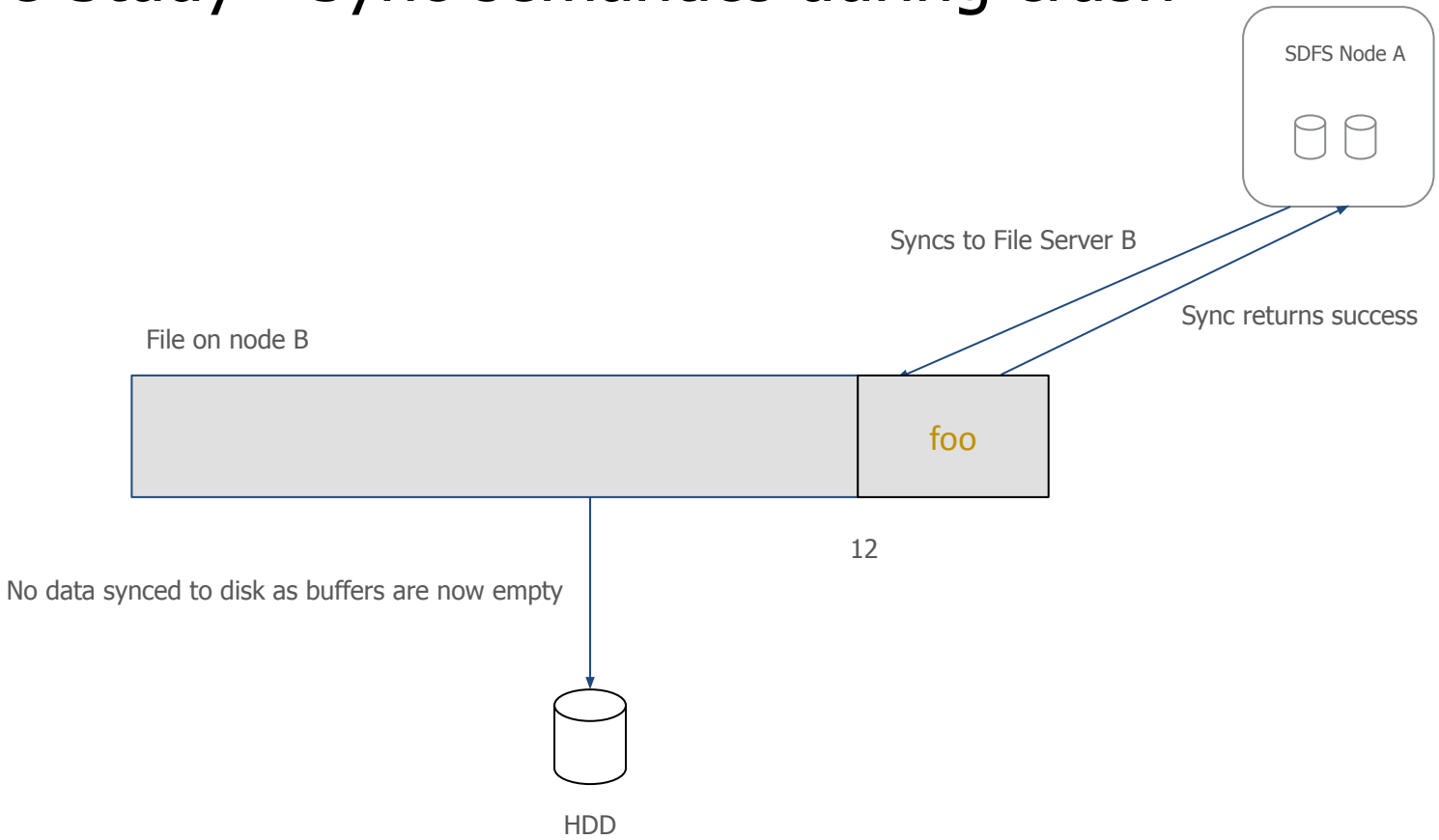




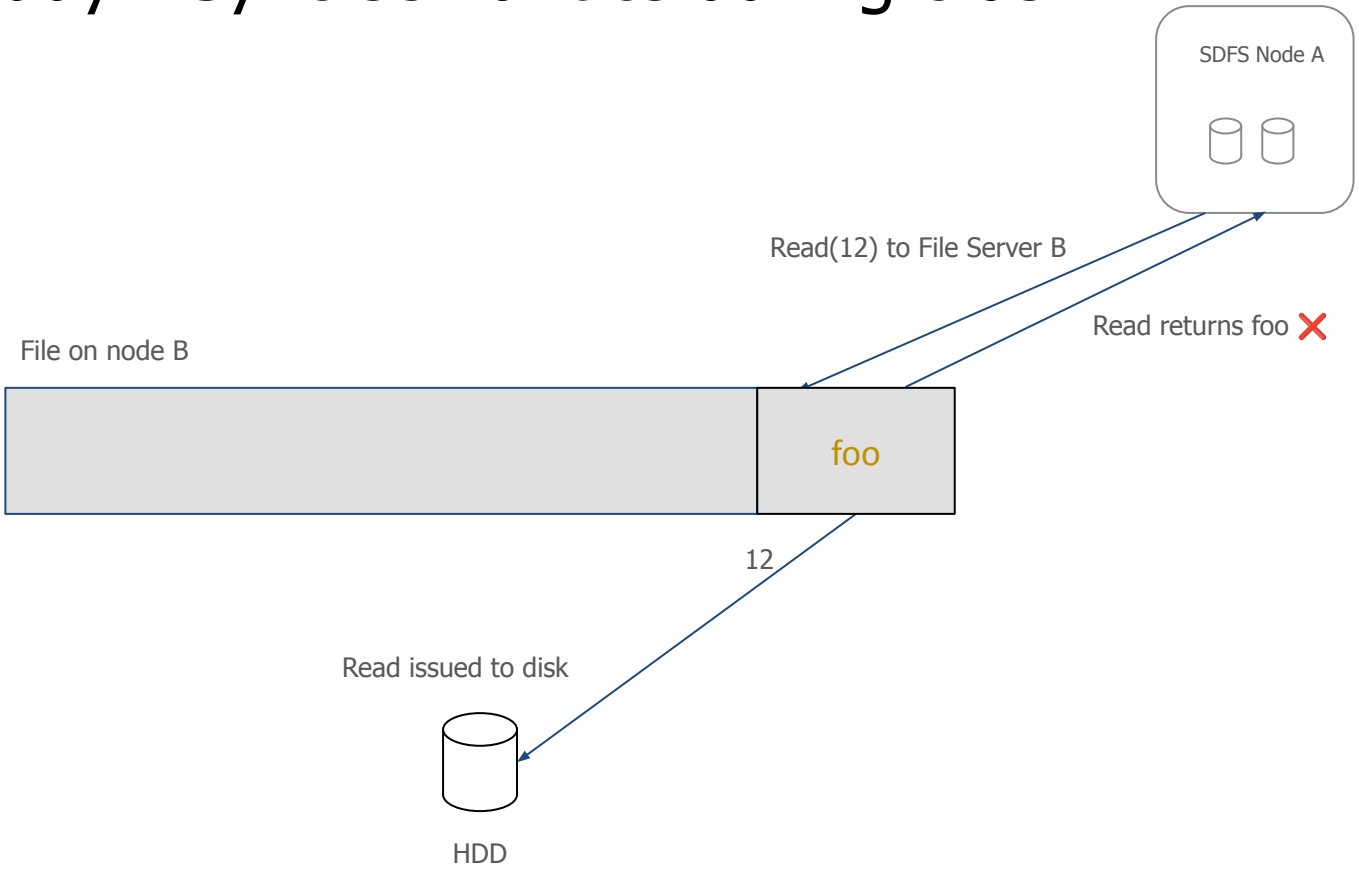
Case Study - Sync semantics during crash



Case Study - Sync semantics during crash



Case Study - Sync semantics during crash



Potential solutions

Keep all File Descriptors in memory at File Server

- ✅ Simple management
- ❌ Not horizontally scalable
- ❌ Unpredictable resource usage

Potential solutions

Keep all File Descriptors in memory at File Server

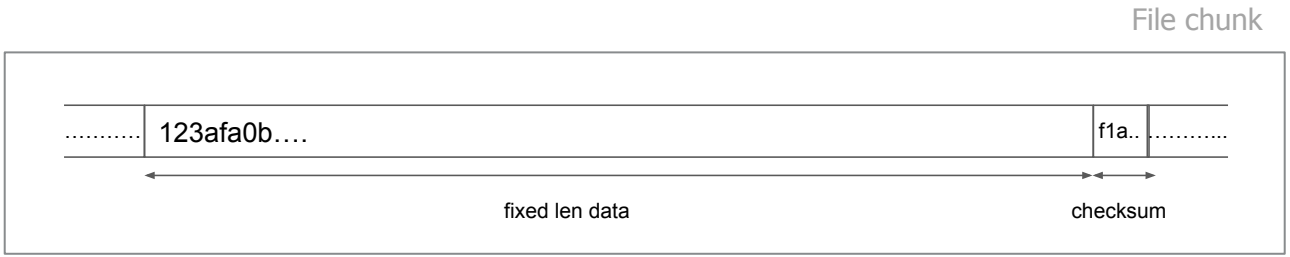
- ✅ Simple management
- ❌ Not horizontally scalable
- ❌ Unpredictable resource usage

Persist dirty information

- ✅ Can limit resource usage to scale
- ✅ Up-replication from last synced offset possible
- ❌ Performance challenges to sync dirty info

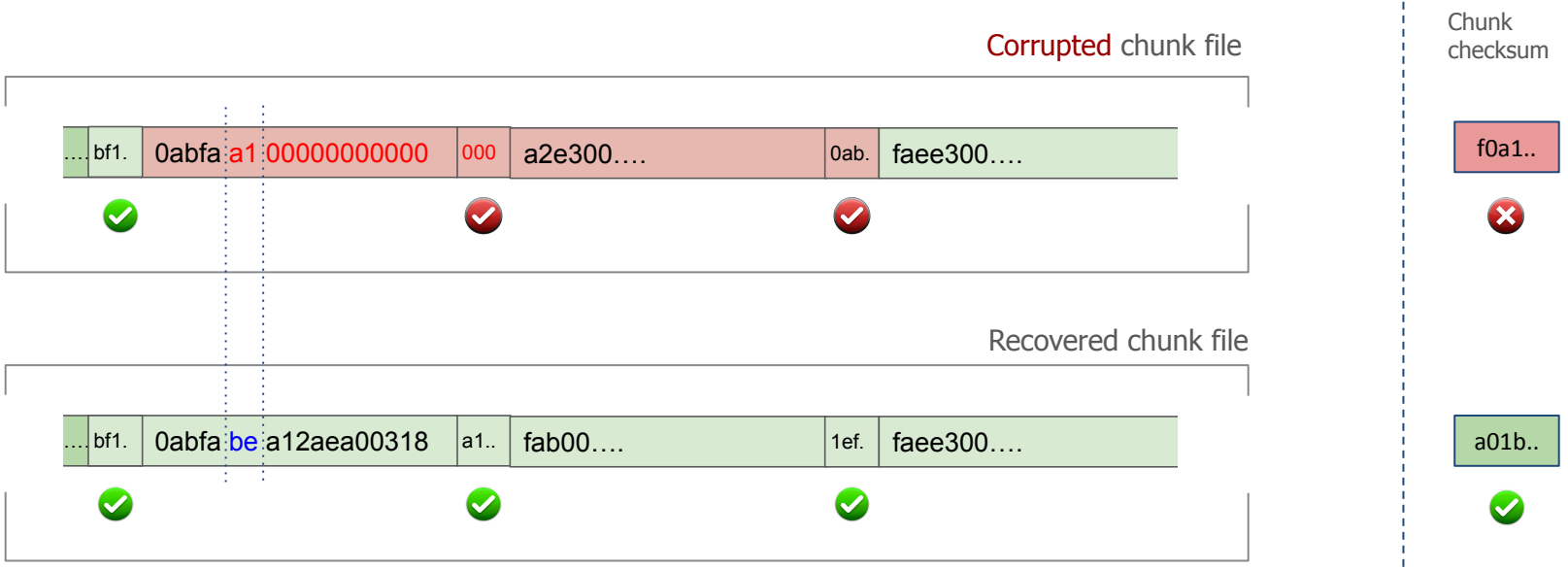
Case Study - Data Integrity

4B of crc32c embedded inline after every 4092B of data



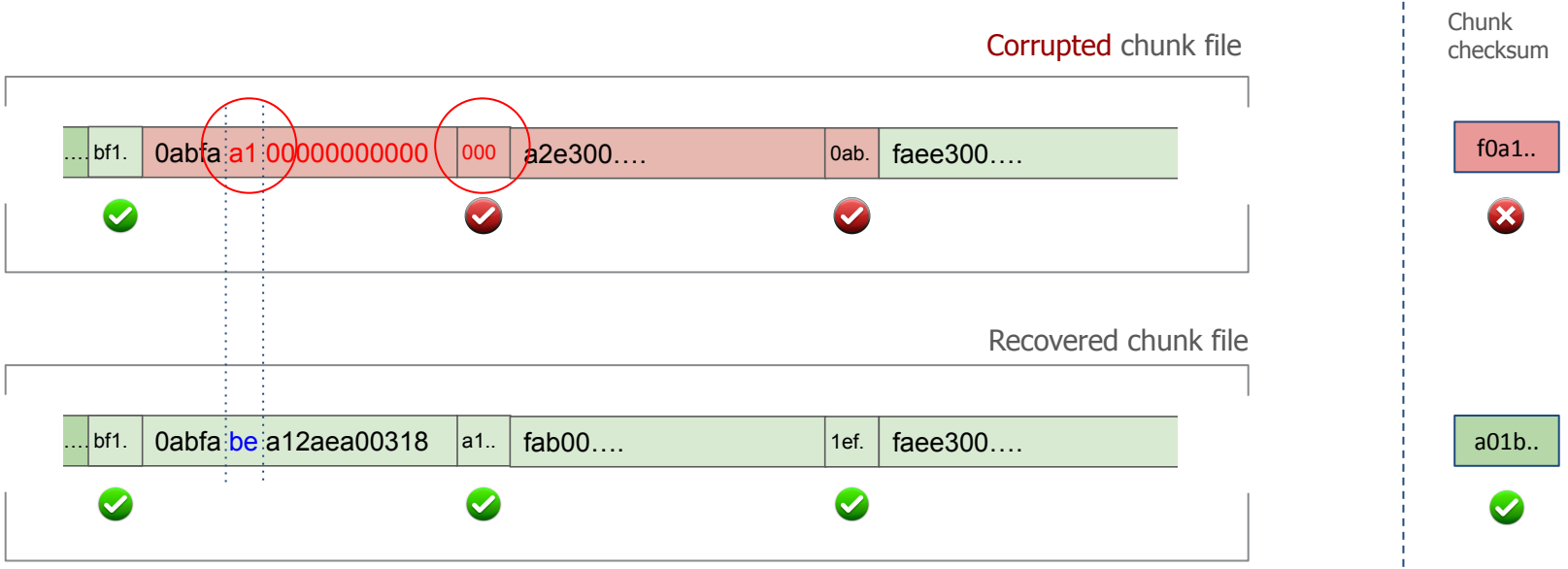
Case Study - Data Integrity

We saw several instances where corruptions bypassed our inline embedded checksums protection



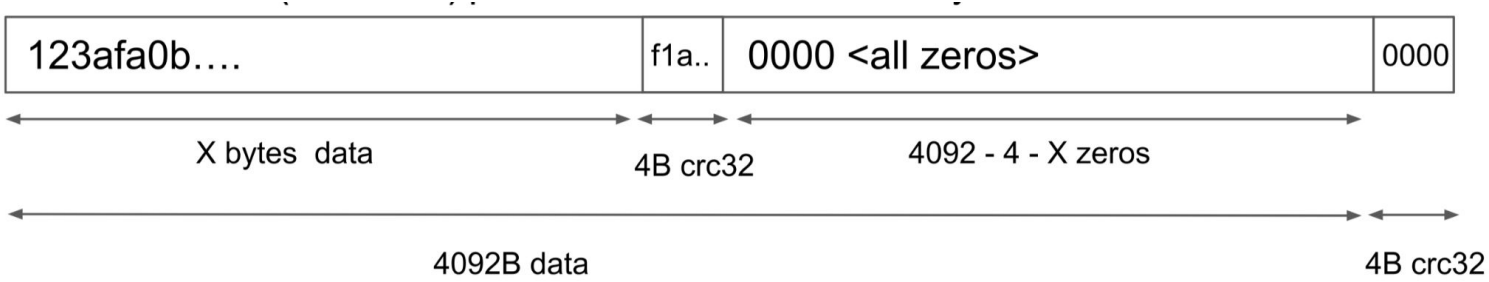
Case Study - Data Integrity

A 2 byte corruption followed by all zeros.
The crc32c is also 0 and correct.



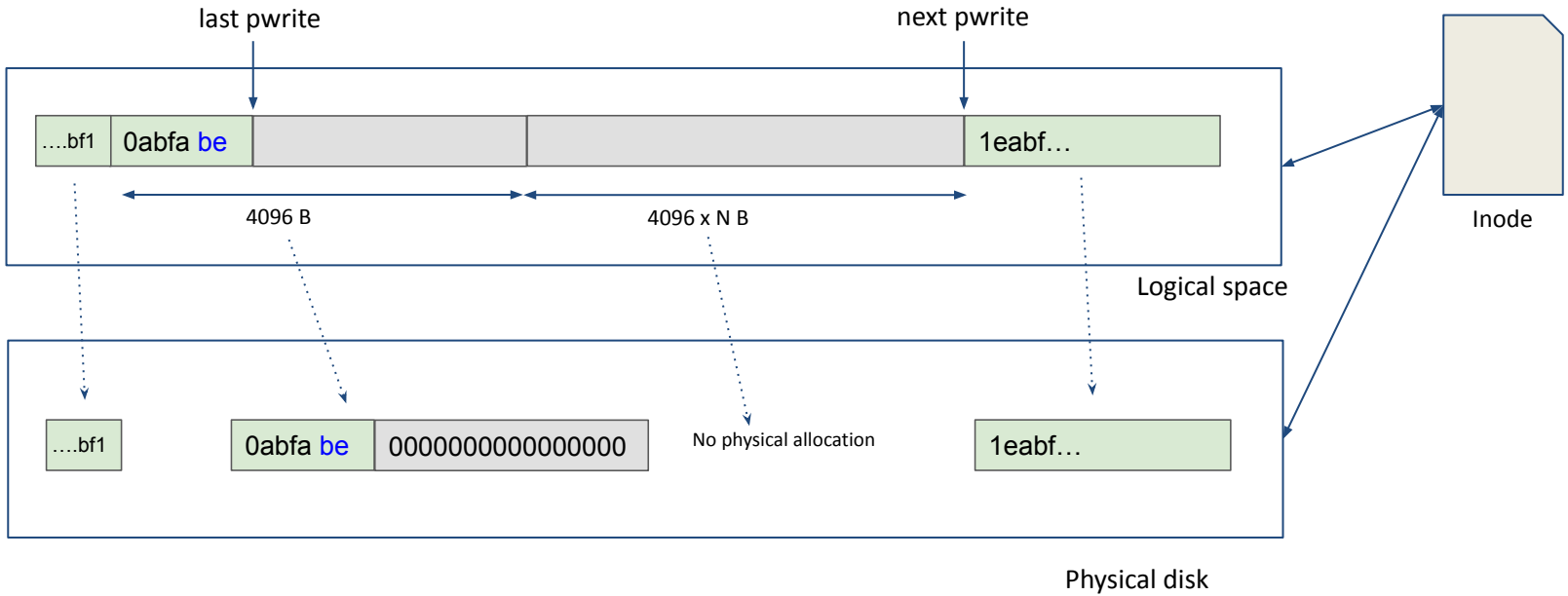
Case Study - Data Integrity

Zero checksum coincidence 🤯

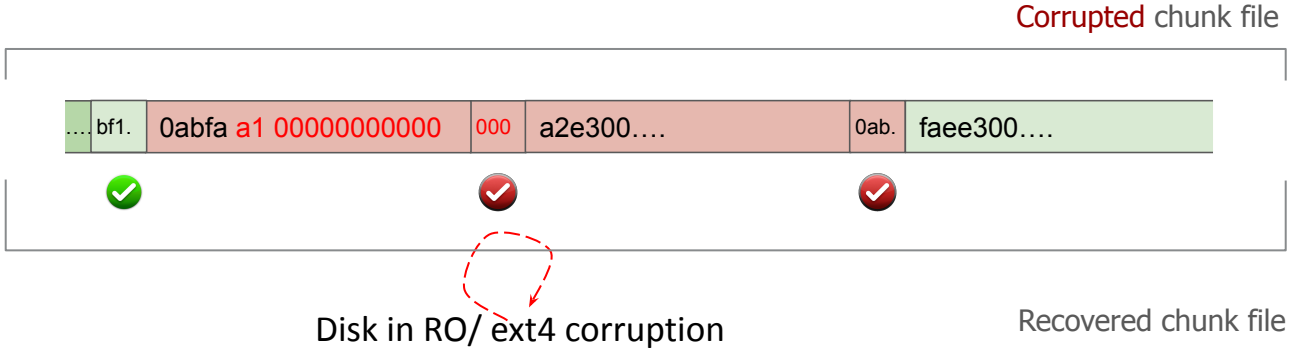


Case Study - Data Integrity

Ext4 sparse file



Case Study - Data Integrity

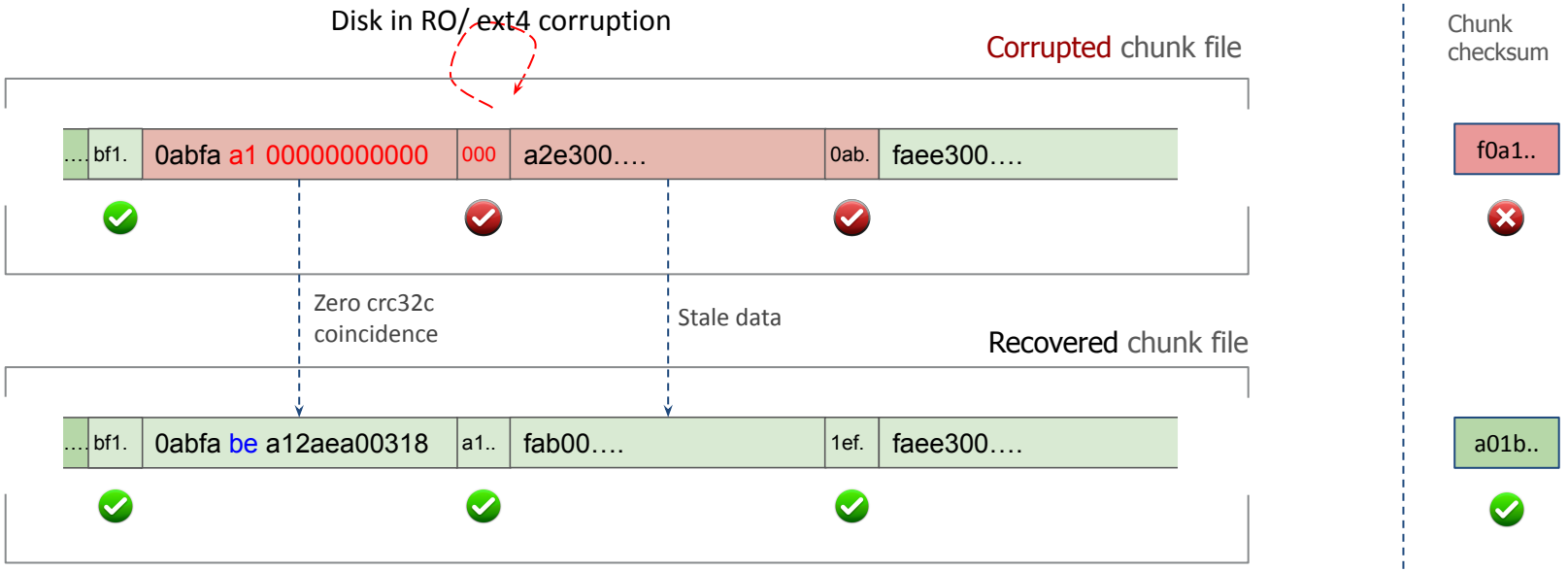



```
EXT4-fs error (device sdd1): ext4_validate_block_bitmap:385: comm sdfs_main: bg 12405: bad block bitmap checksum
EXT4-fs (sdd1): Remounting filesystem read-only
EXT4-fs error (device sdd1) in ext4_free_blocks:4984: Filesystem failed CRC
EXT4-fs error (device sdd1) in ext4_ext_remove_space:3193: IO failure
EXT4-fs error (device sdd1) in ext4_reserve_inode_write:5971: Journal has aborted
EXT4-fs error (device sdd1) in ext4_truncate:4581: Journal has aborted
EXT4-fs error (device sdd1): ext4_evict_inode:307: comm sdfs_main: couldn't truncate inode 241697901 (err -30)
blk_update_request: I/O error, dev loop6, sector 477809792 op 0x9:(WRITE_ZEROES) flags 0x1000800 phys_seg 0 prio class 0
```


36

Case Study - Data Integrity

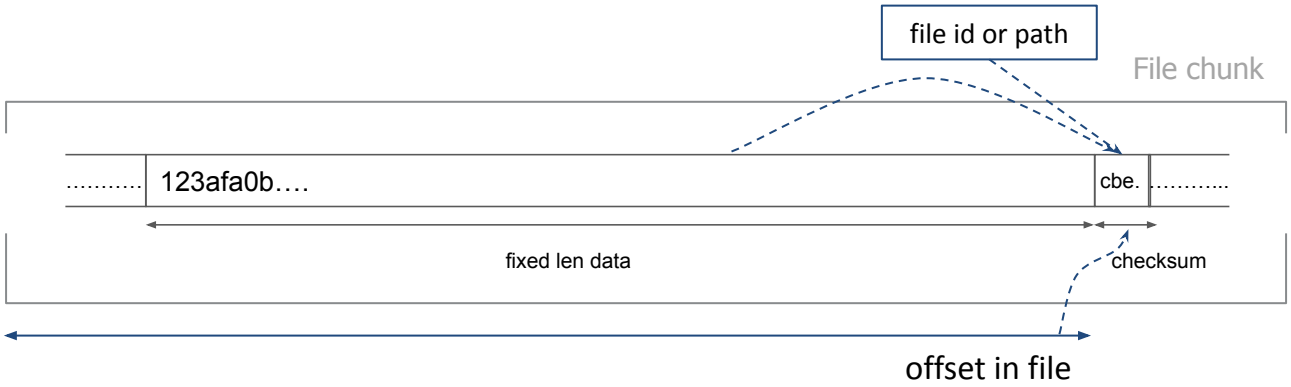
Summing it all up



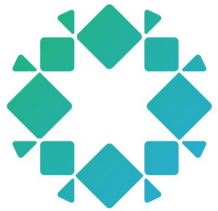
Context aware inline checksum

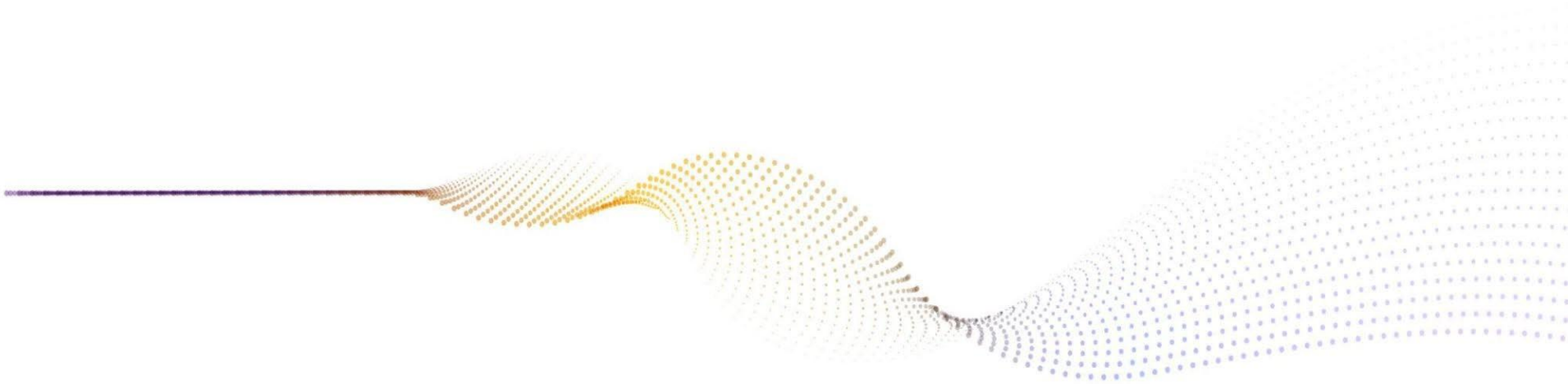
To avoid reading stale data, we add context to our inline checksums by including file path and offset.

```
Checksum(data block of a file) = Function(data block,
                                          unique file ID of the data block,
                                          offset of data block inside that file)
```



Don’t Backup. Go Forward.





Thank you for attending!

Please remember to rate this session. You get access the presentations at
http://sniadeveloper.org/conference

