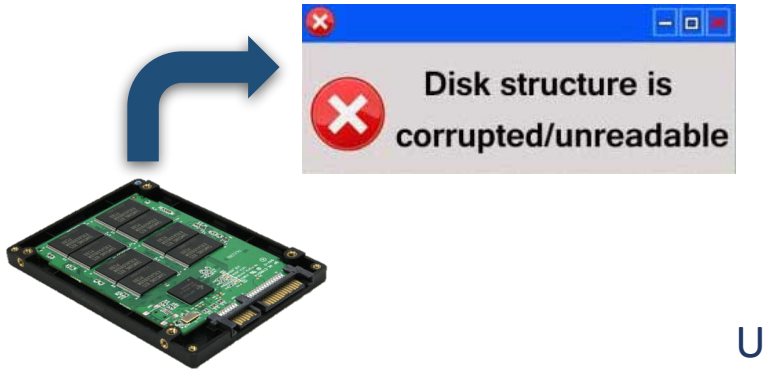
Evaluating File System Reliability on Solid State Drives

Shehbaz Jaffer, **Stathis Maneas**, Andy Hwang, Bianca Schroeder



Computer Science
UNIVERSITY OF TORONTO

USENIX ATC ‘19

Introduction & Motivation

- Storage landscape has changed:
  - ~~HDDs~~ -> SSDs.
  - What about their failure characteristics?
    - ➢Partial failures are a magnitude higher for SSDs!
    - ➢FTL is prone to bugs during power faults.
  - New/Evolved file systems:
    - ➢ext3 -> **ext4** (journaling).
    - ➢**Btrfs** (copy-on-write).
    - ➢**F2FS** (log-structured, tailored for flash).

- **Our goal:** How do these file systems deal with partial drive errors?

Research Questions & Methodology

Research Questions & Methodology

- **What we know** (IRON File Systems, 2005):
  - Only ext3, JFS, ReiserFS, NTFS (partial).
  - Hard disks only.
  - Does not consider file system checkers.

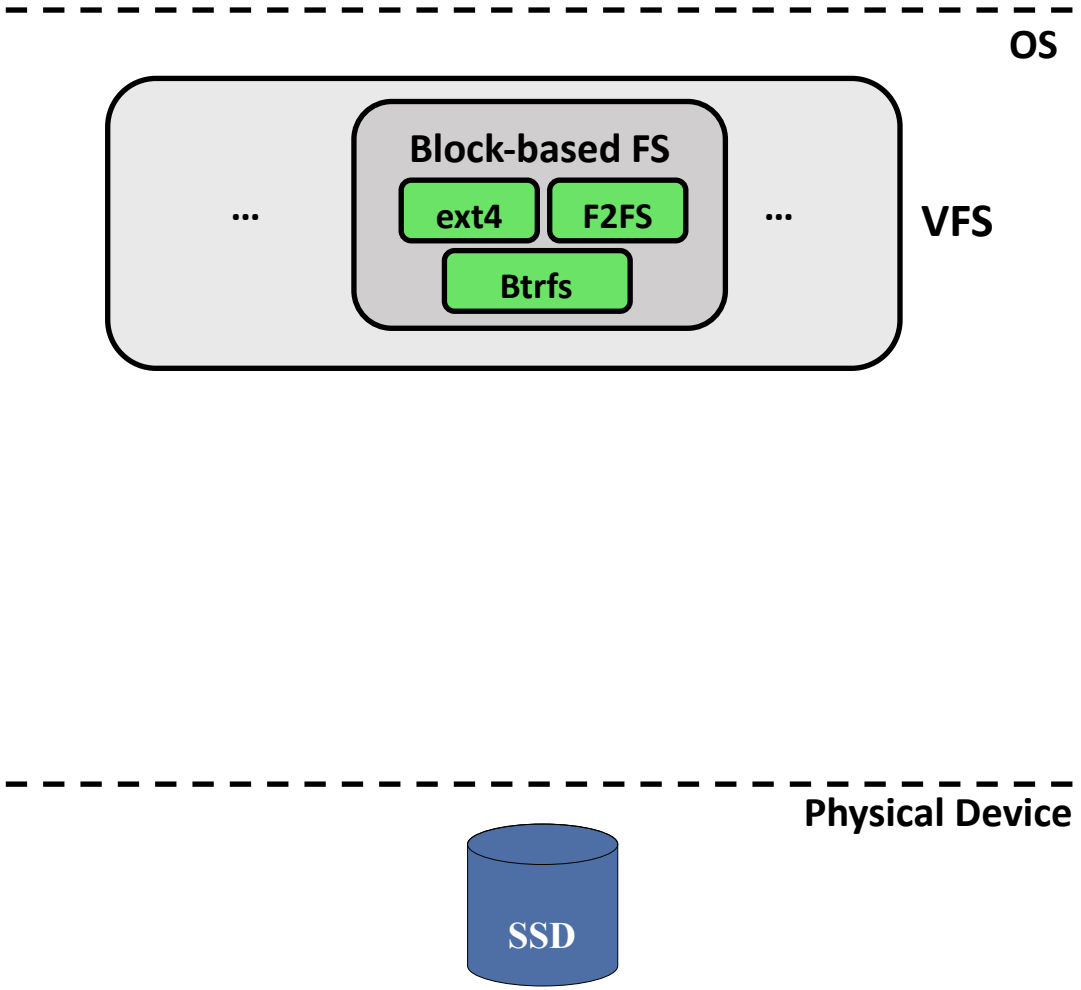
Research Questions & Methodology

- **What we know** (IRON File Systems, 2005):
  - Only ext3, JFS, ReiserFS, NTFS (partial).
  - Hard disks only.
  - Does not consider file system checkers.

- **What we *want* to know:**
  - Btrfs, ext4, F2FS.
  - Can they detect errors?
  - Can they recover from errors?
  - Can the system checker (fsck) fix errors?

SSD Faults and their Manifestation

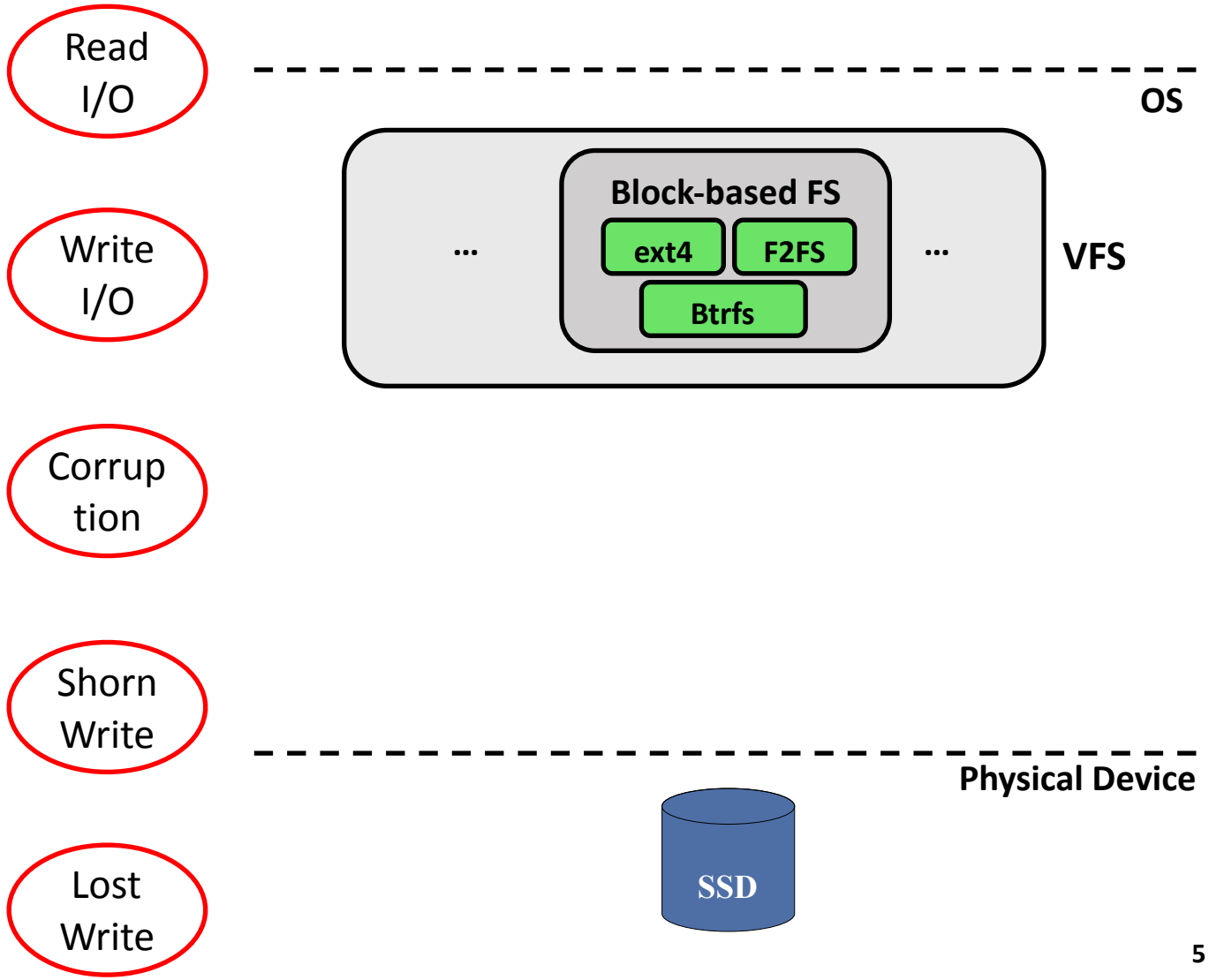
- What if the storage device starts misbehaving and generating errors?
- How exactly file systems deal with these errors?



SSD Faults and their Manifestation

SSD/Flash Faults

Manifestation

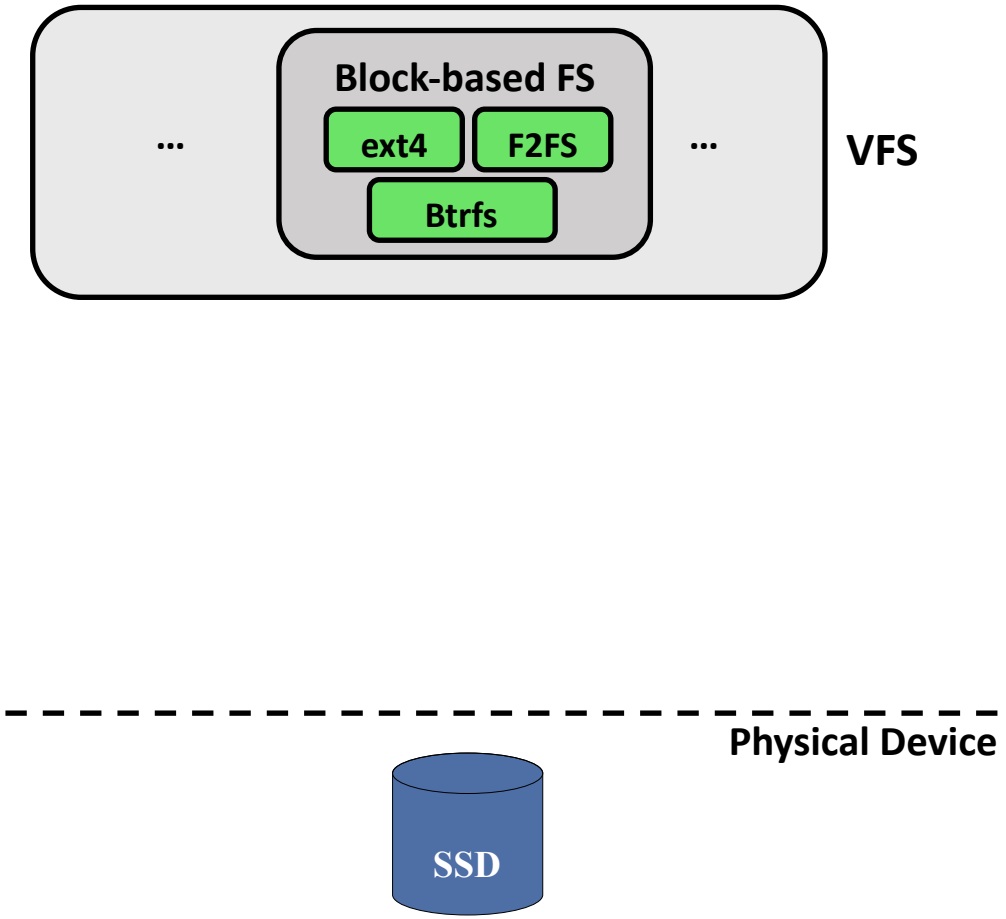


SSD Faults and their Manifestation

SSD/Flash Faults
Uncorrectable bit corruption

Manifestation

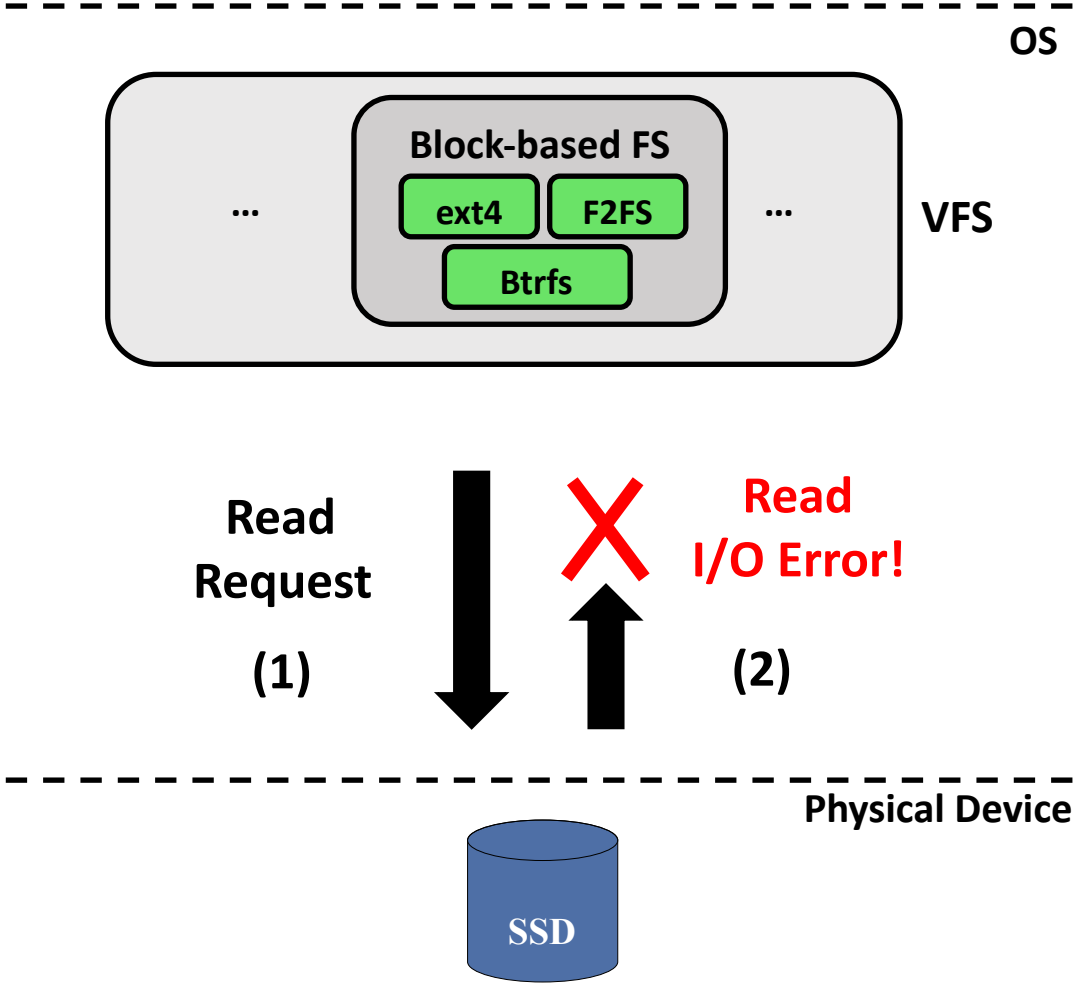
- Read I/O
- Write I/O
- Corruption
- Shorn Write
- Lost Write



SSD Faults and their Manifestation

SSD/Flash Faults
Uncorrectable bit corruption

Manifestation
Read I/O
Write I/O
Corruption
Shorn Write
Lost Write



Read I/O Error

- The SSD could not successfully complete the operation!

SSD Faults and their Manifestation

SSD/Flash Faults
Uncorrectable bit corruption
Incomplete Program Operation

Manifestation

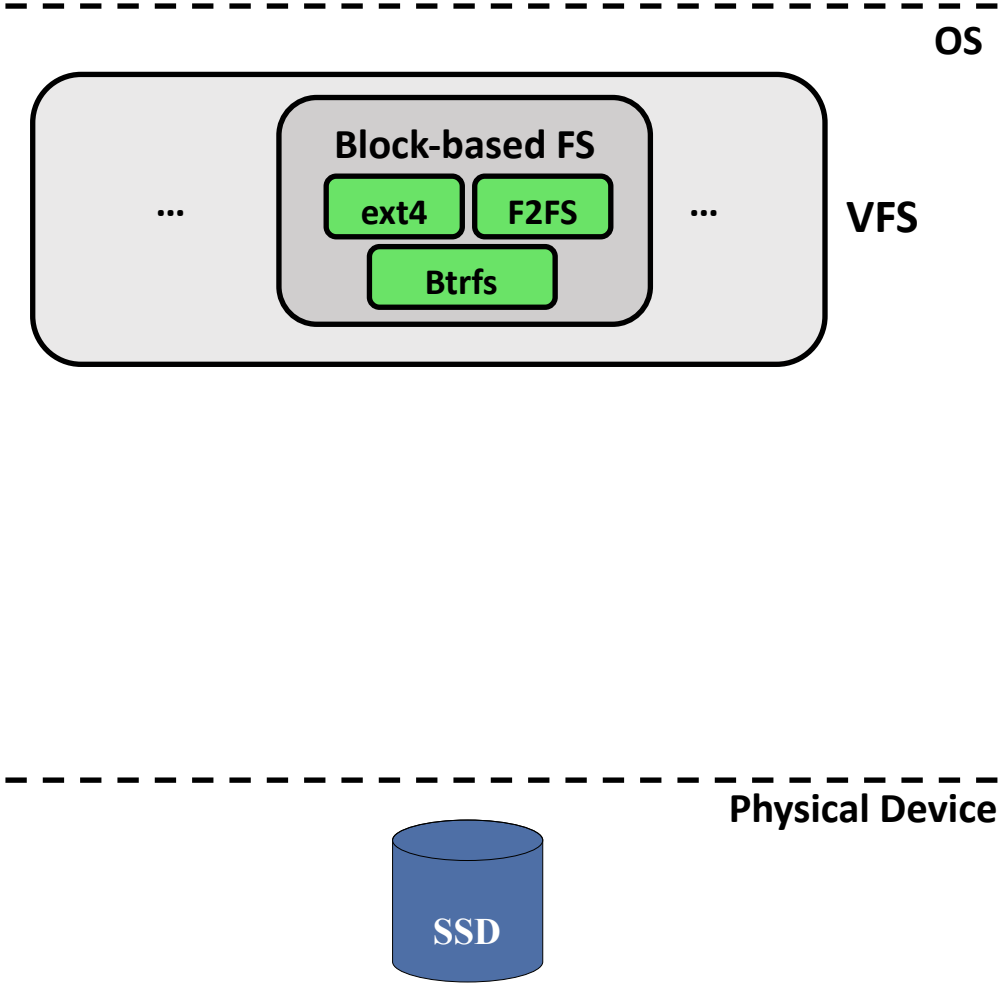
Read I/O

Write I/O

Corrup tion

Shorn Write

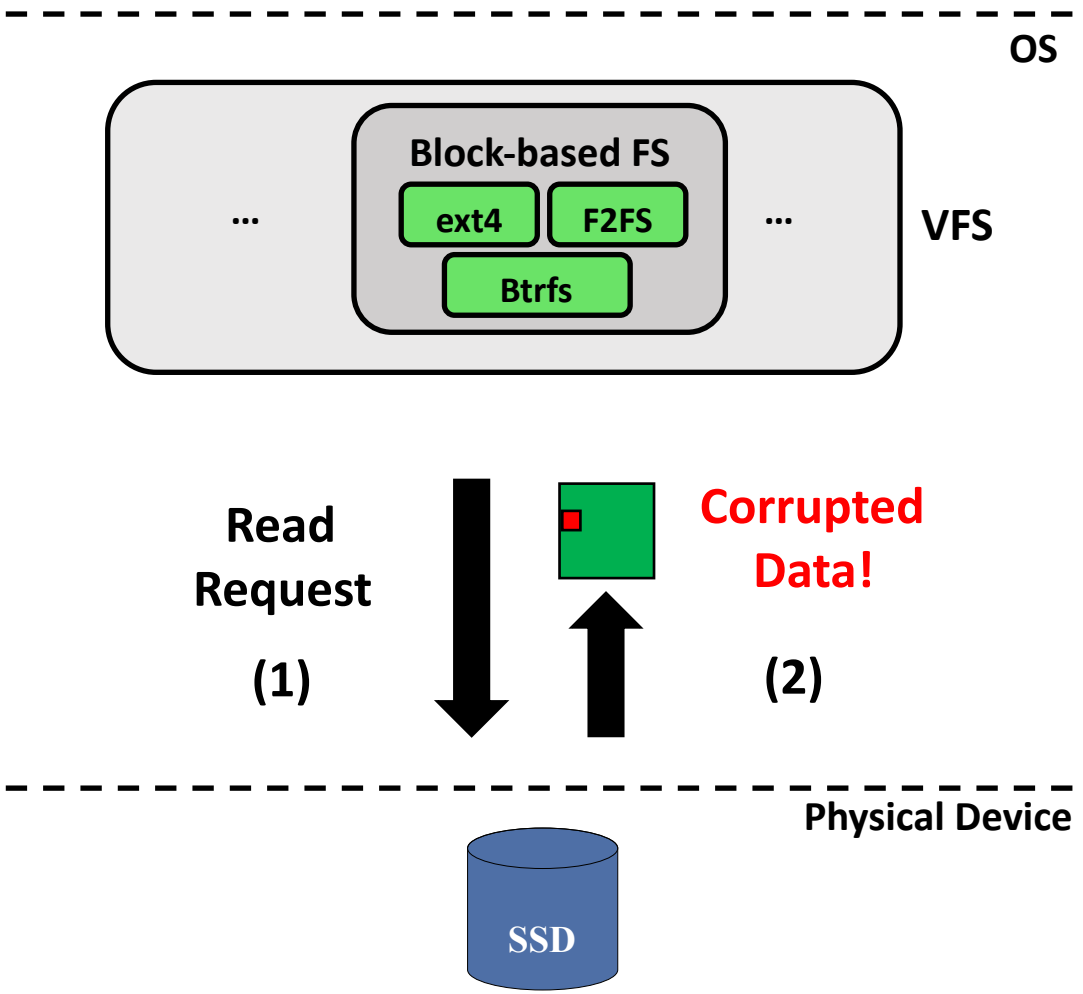
Lost Write



SSD Faults and their Manifestation

SSD/Flash Faults
Uncorrectable bit corruption
Incomplete Program Operation

Manifestation
Read I/O
Write I/O
Corruption
Shorn Write
Lost Write



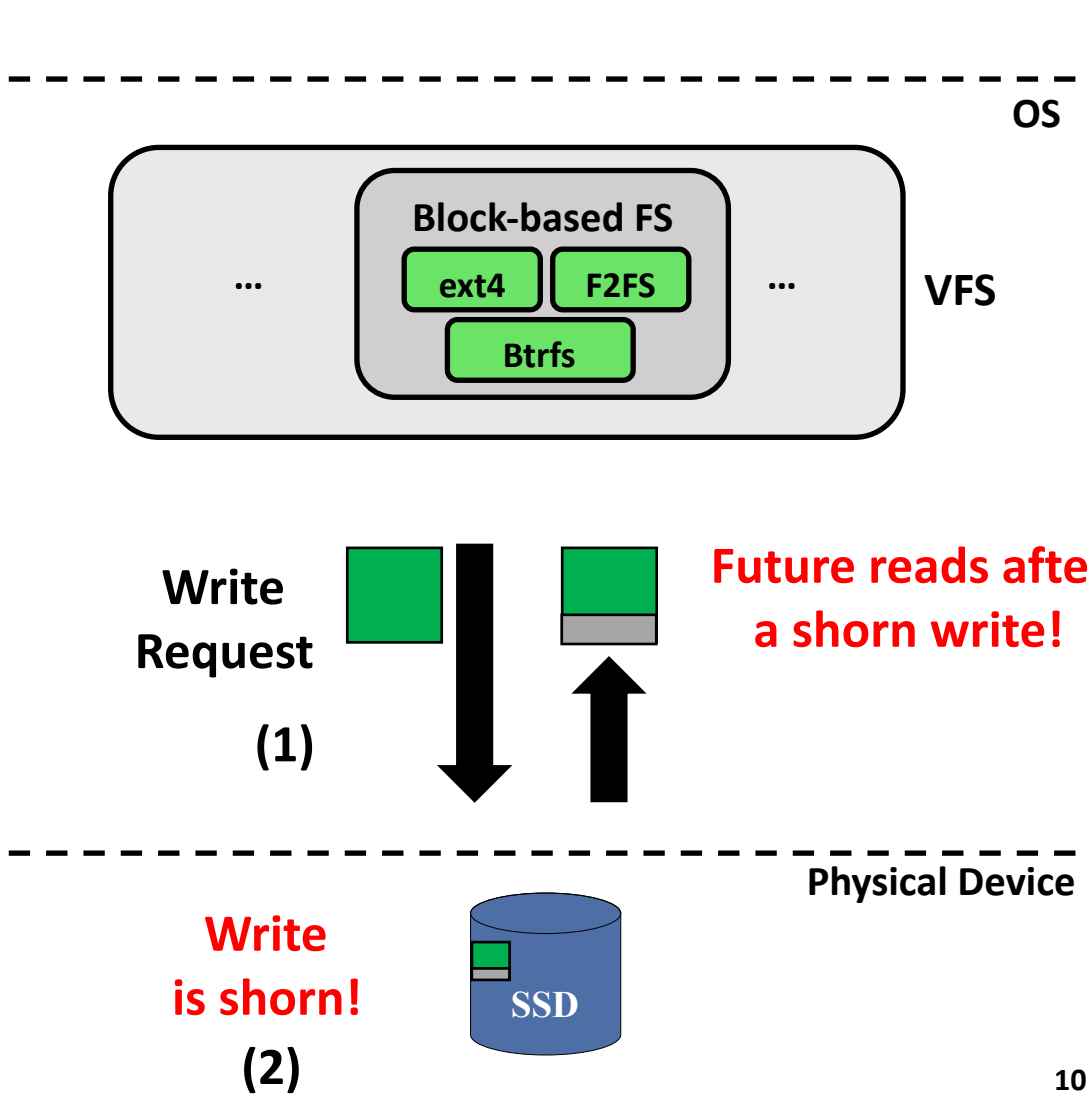
Corruption

- Silent! The SSD does not detect it!

SSD Faults and their Manifestation

SSD/Flash Faults
Uncorrectable bit corruption
Incomplete Program Operation

Manifestation
Read I/O
Write I/O
Corruption
Shorn Write
Lost Write

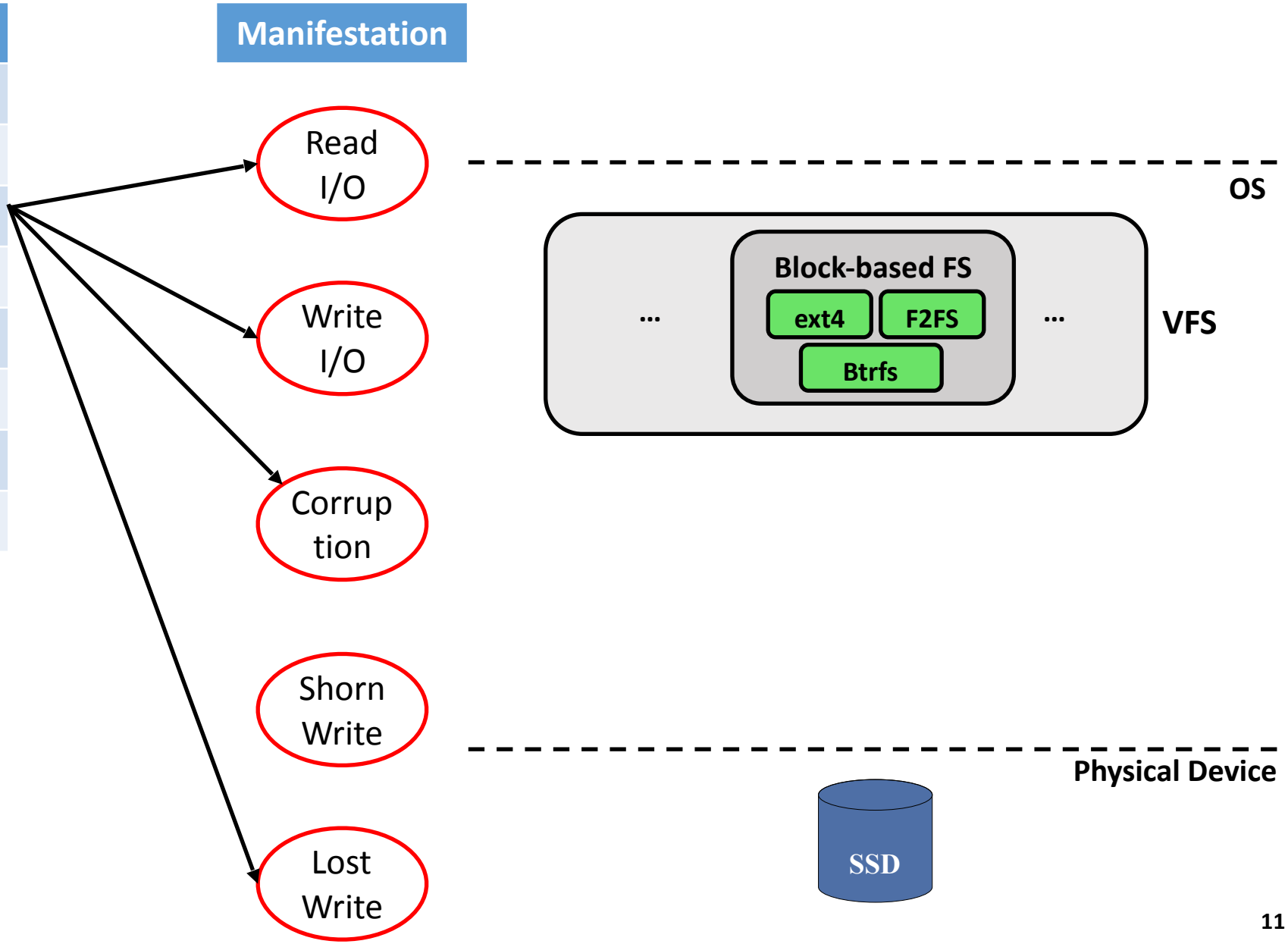


Shorn Write

- A write issued by the file system but partially persisted on the SSD.
  - Power fault (Zheng et *al.*, FAST '13).

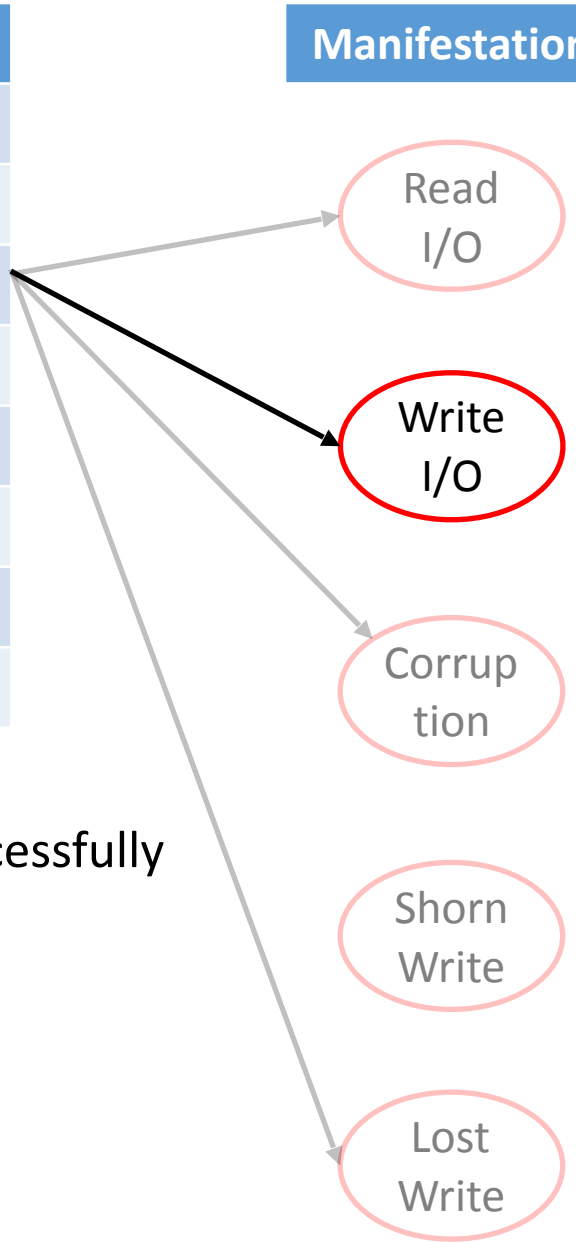
SSD Faults and their Manifestation

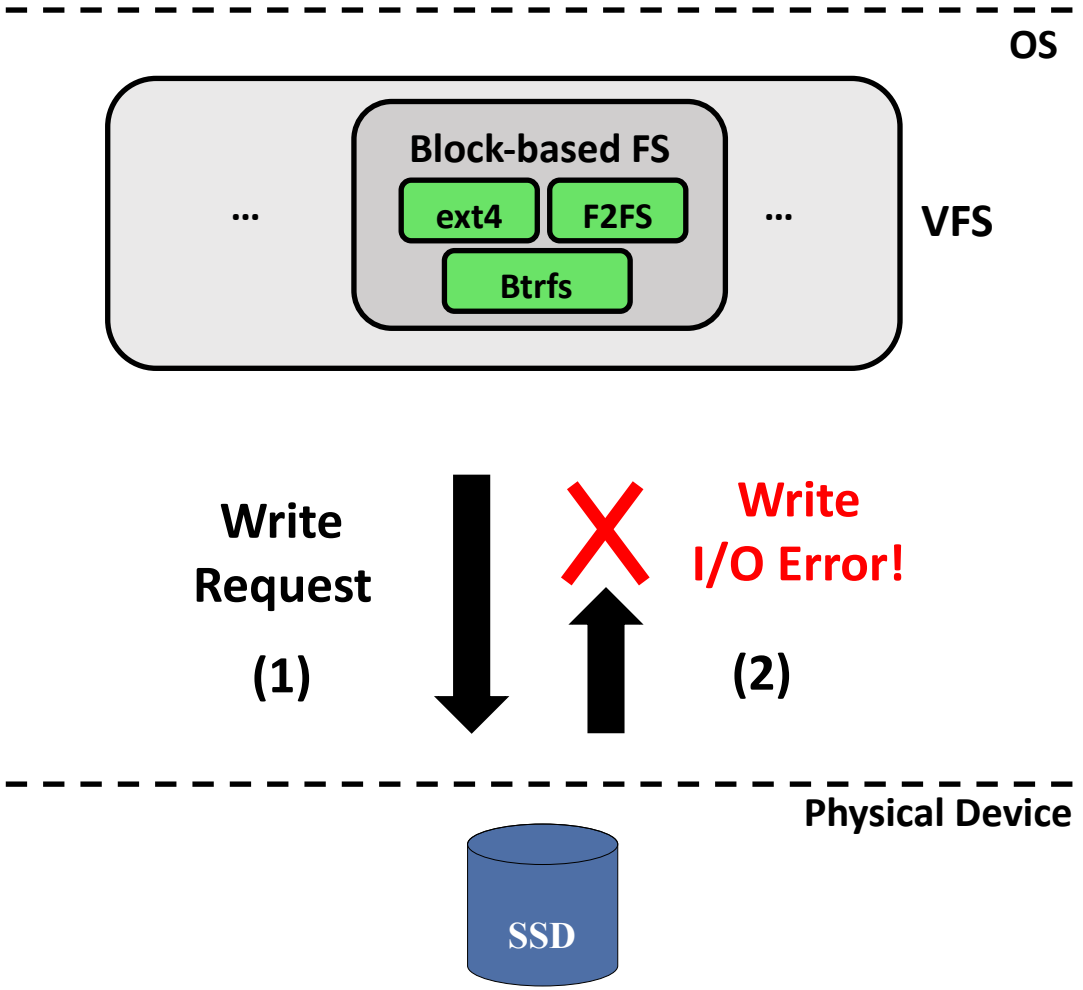
SSD/Flash Faults
Uncorrectable bit corruption
Incomplete Program Operation
Dropped Writes



SSD Faults and their Manifestation

SSD/Flash Faults
Uncorrectable bit corruption
Incomplete Program Operation
Dropped Writes





Write I/O Error

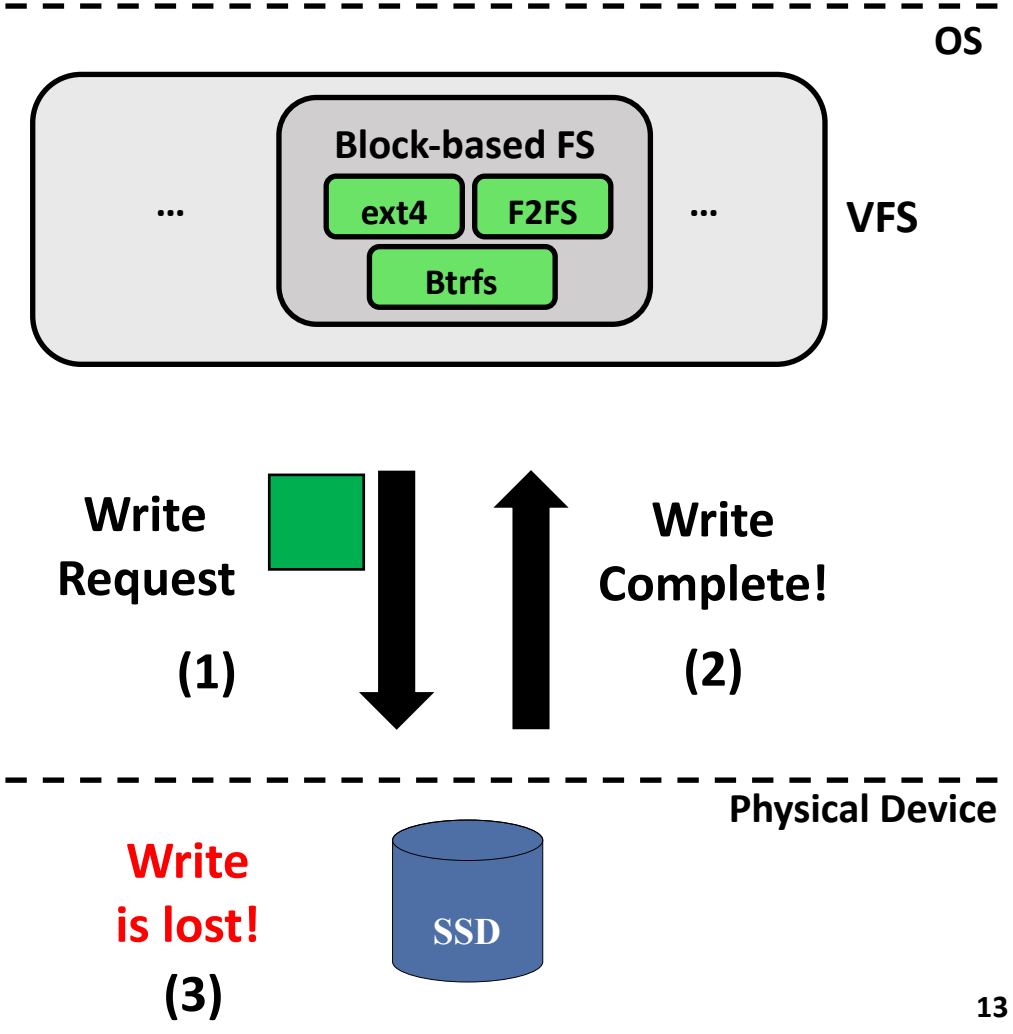
- The SSD could not successfully complete the operation!

SSD Faults and their Manifestation

SSD/Flash Faults
Uncorrectable bit corruption
Incomplete Program Operation
Dropped Writes

Manifestation

- Read I/O
- Write I/O
- Corruption
- Shorn Write
- Lost Write

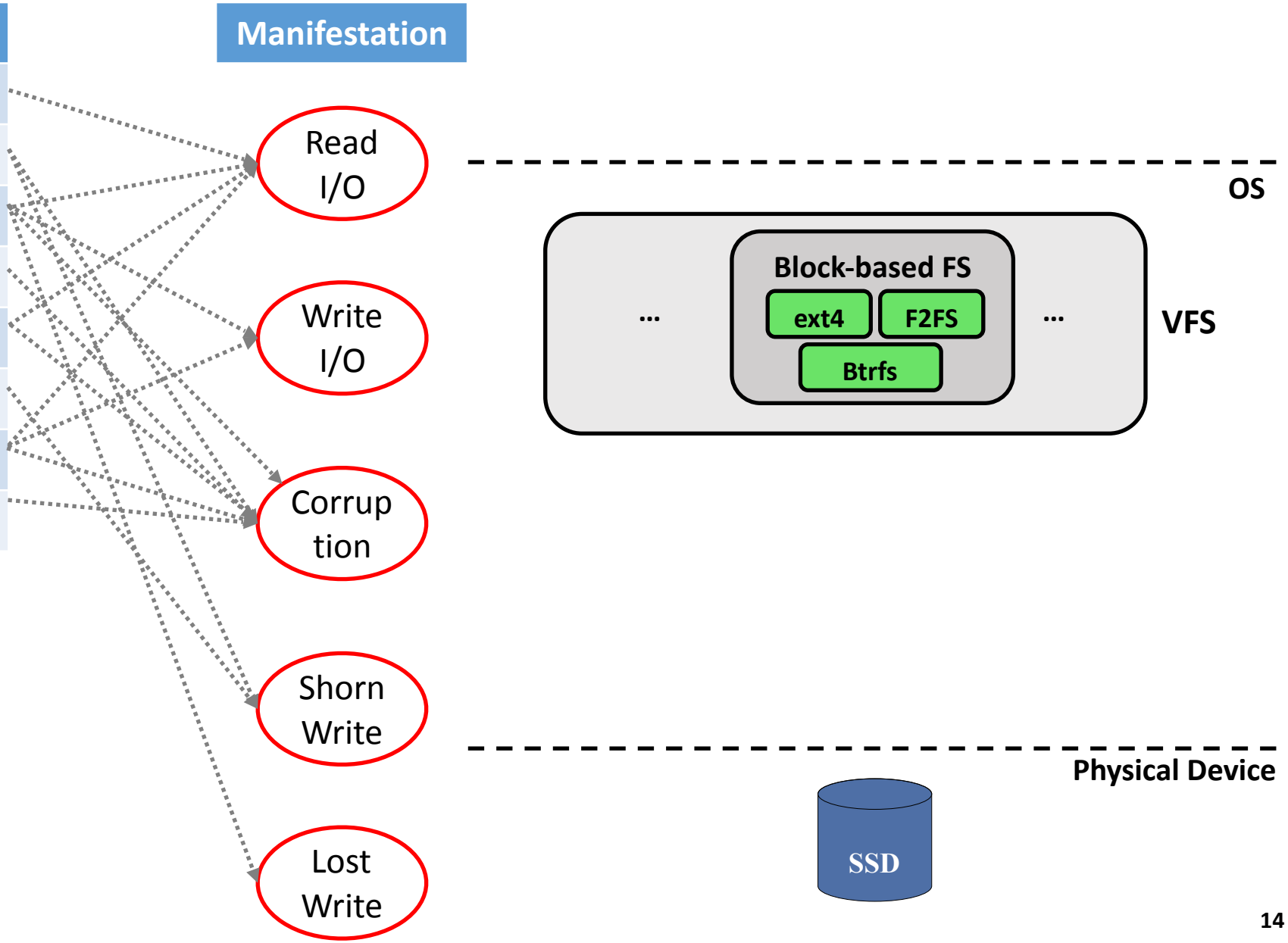


Lost Write

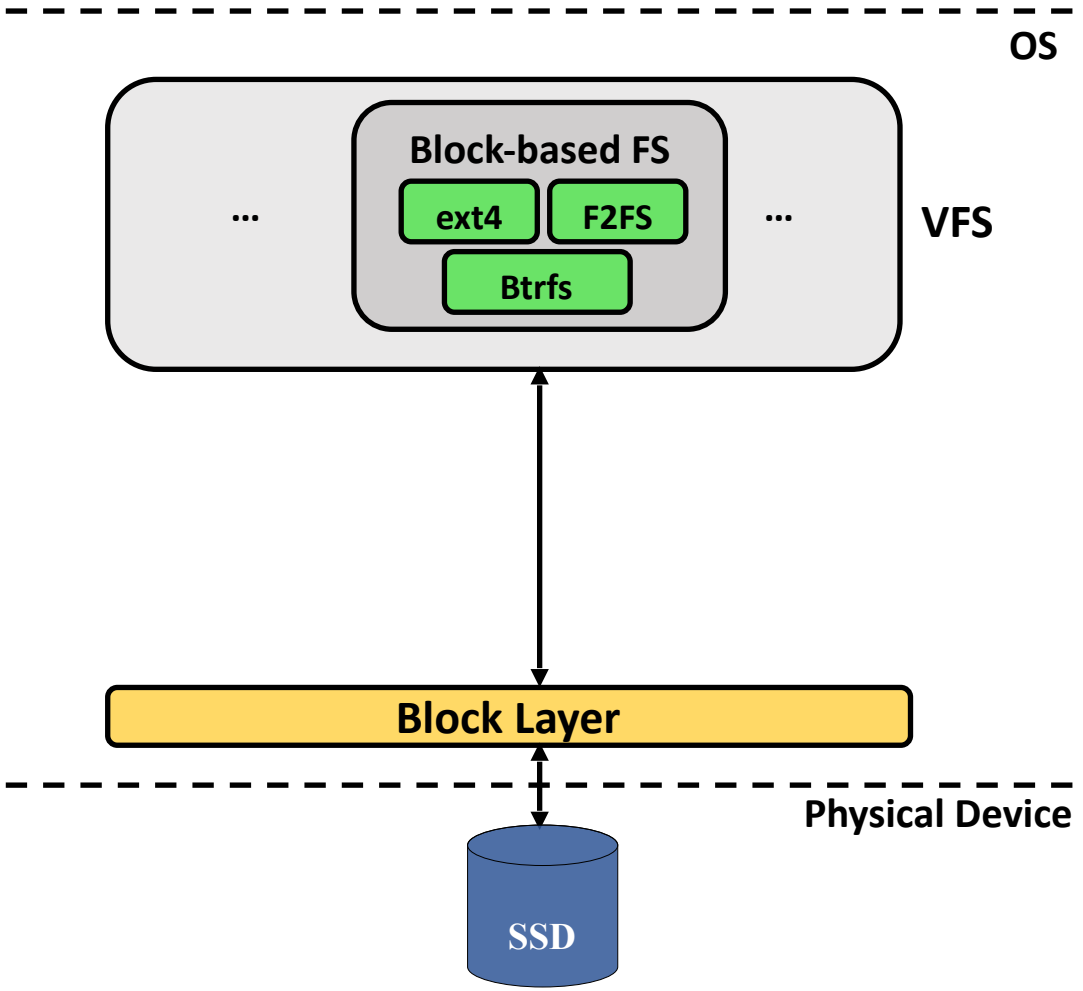
- No error is returned!

SSD Faults and their Manifestation

SSD/Flash Faults
Uncorrectable bit corruption
Incomplete Program Operation
Dropped Writes
Silent Bit Corruption
Misdirected Writes
Shorn Writes
FTL Metadata Corruption
Incomplete Erase Operation

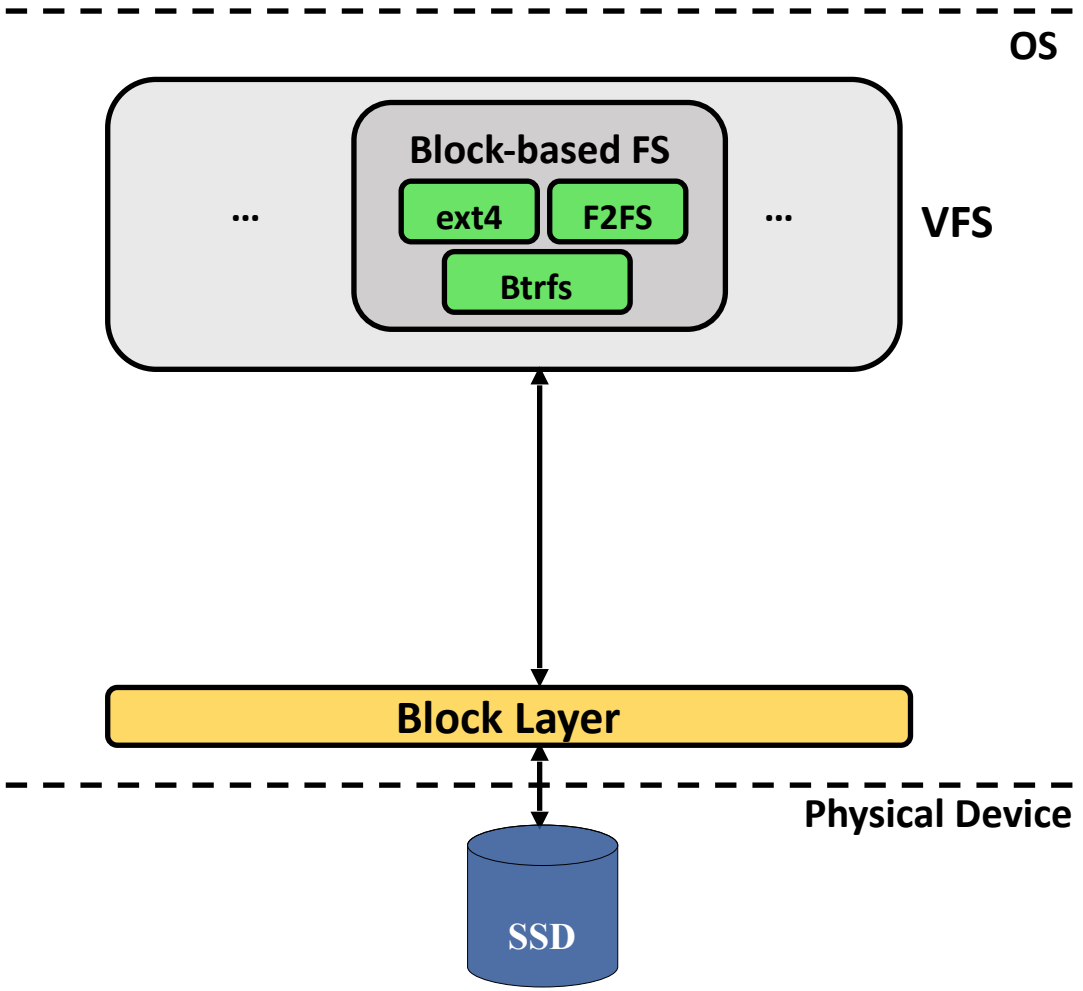


How to test the resiliency of file systems against SSD faults?



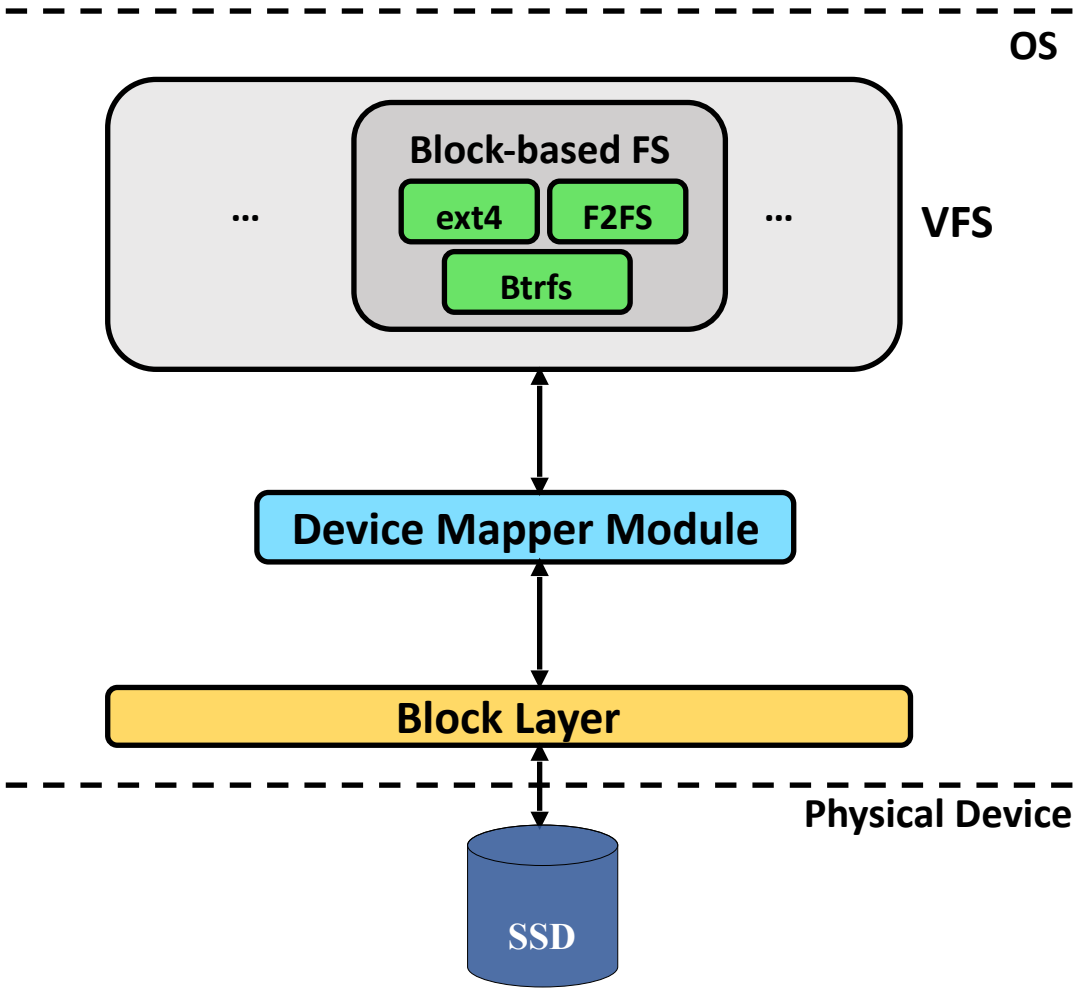
How to test the resiliency of file systems against SSD faults?

- Emulate the faults' manifestation!
  - Inject errors at the Block Layer.



How to test the resiliency of file systems against SSD faults?

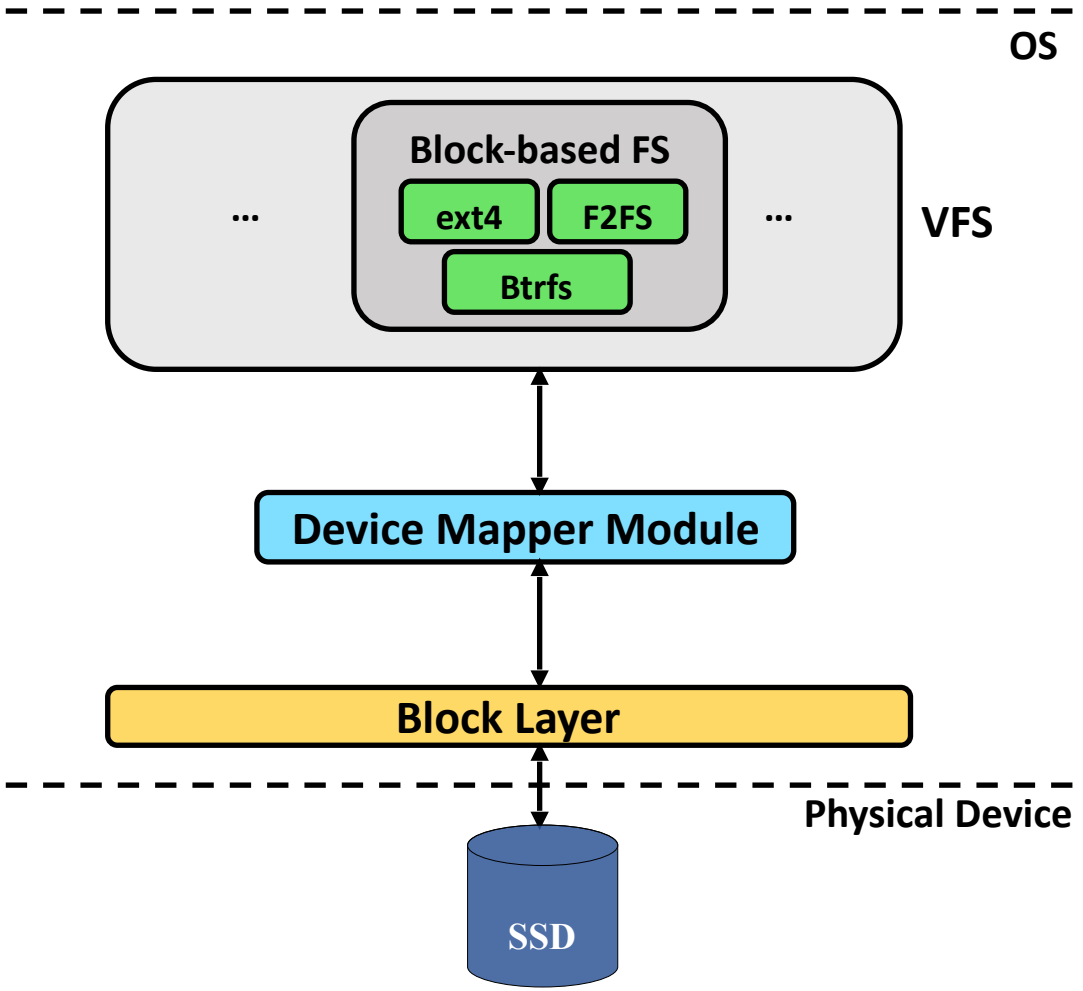
- Emulate the faults' manifestation!
  - Inject errors at the Block Layer.



How to test the resiliency of file systems against SSD faults?

- Emulate the faults' manifestation!
  - Inject errors at the Block Layer.

- Device Mapper Module:
  - Intercept every I/O request.
  - Fail a request & return an error.
  - Silently drop a request.
  - Alter block contents online.

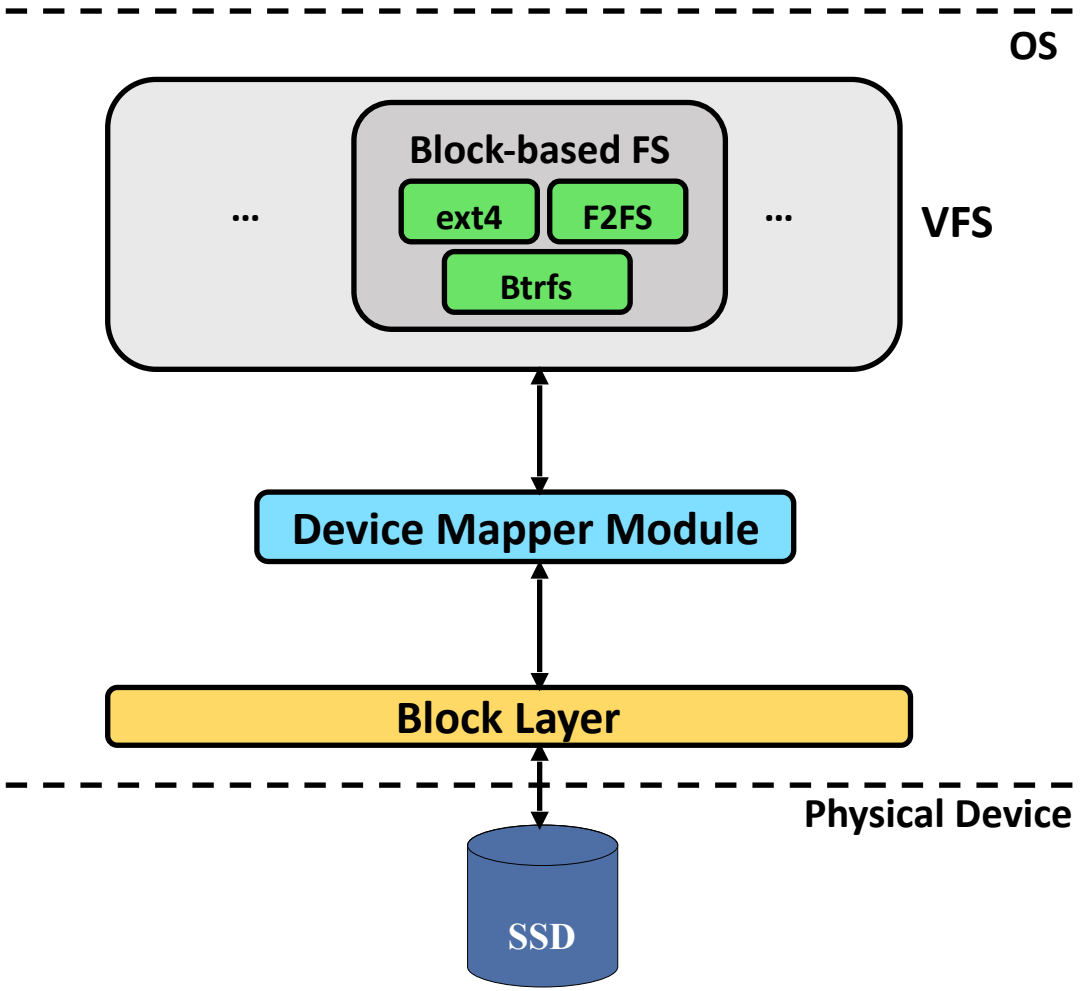


Targeted Error Injection

- Understand the effect of every injected error.

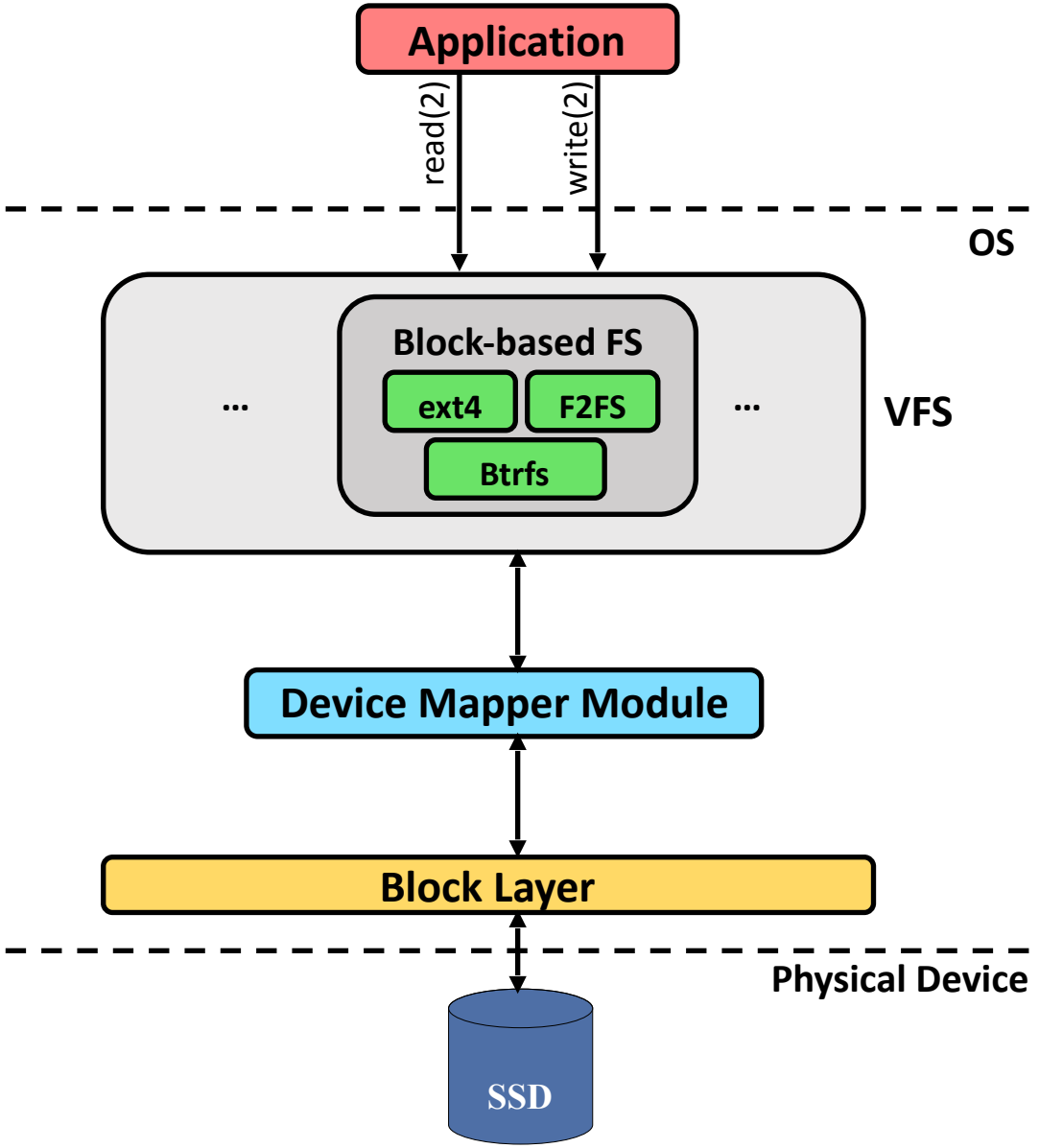
- Identify block types and specific data structures within each block!

- Target specific data structures and fields within them:
  - Trace all I/O requests (*blktrace*).
  - Logic inside our device mapper module.
  - FS tools, such as dump, to inspect the disk image offline.



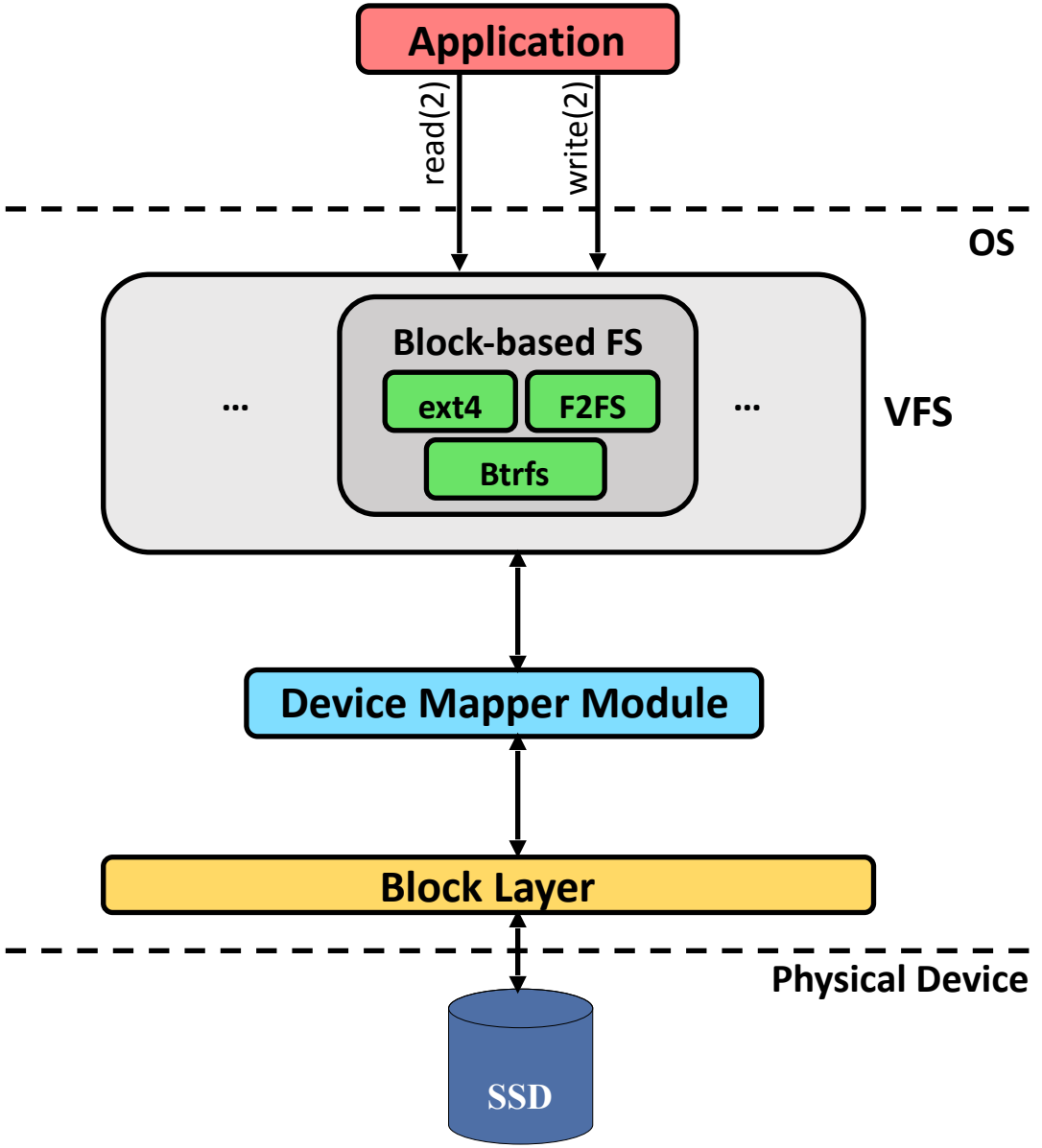
How do file systems detect and recover from errors?

- Each application focuses on one particular operation:
  - *mkdir*, *creat*, *etc.*
- Run an application and collect all accessed blocks.
- Targeted error injection:
  - Repeat the execution and inject a single error into each accessed block.
  - Target one block at a time.
- Better isolation and characterization of the file system's reaction to every injected error!



How do file systems detect and recover from errors?

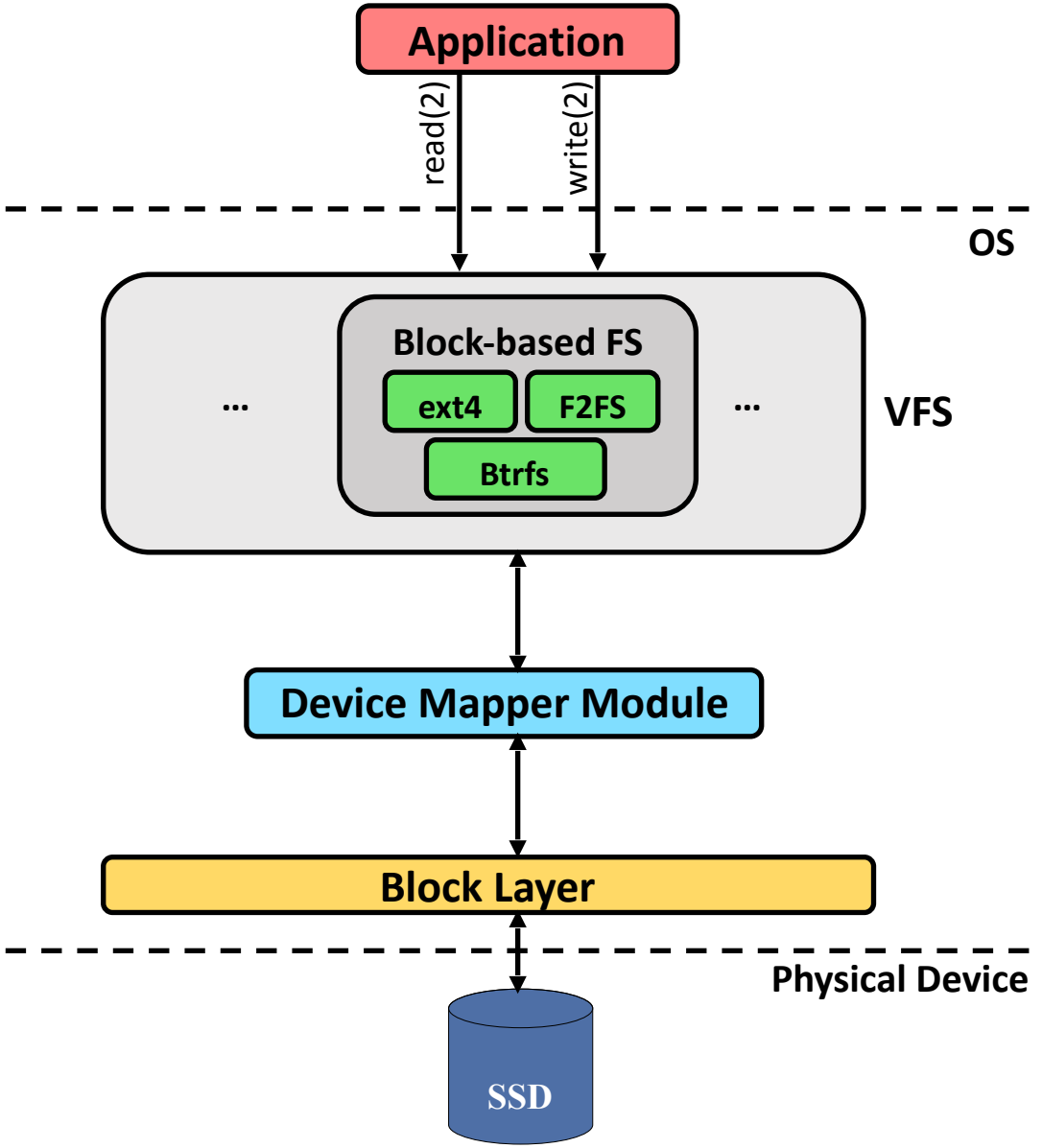
- Categorize each file system's detection and recovery policies:
  - Across all visible aspects, such as *logs*, *return codes*, etc.
  - Check how effectively ***fsck*** recovers the file system.



How do file systems detect and recover from errors?

- Categorize each file system's detection and recovery policies:
  - Across all visible aspects, such as *logs*, *return codes*, etc.
  - Check how effectively ***fsck*** recovers the file system.

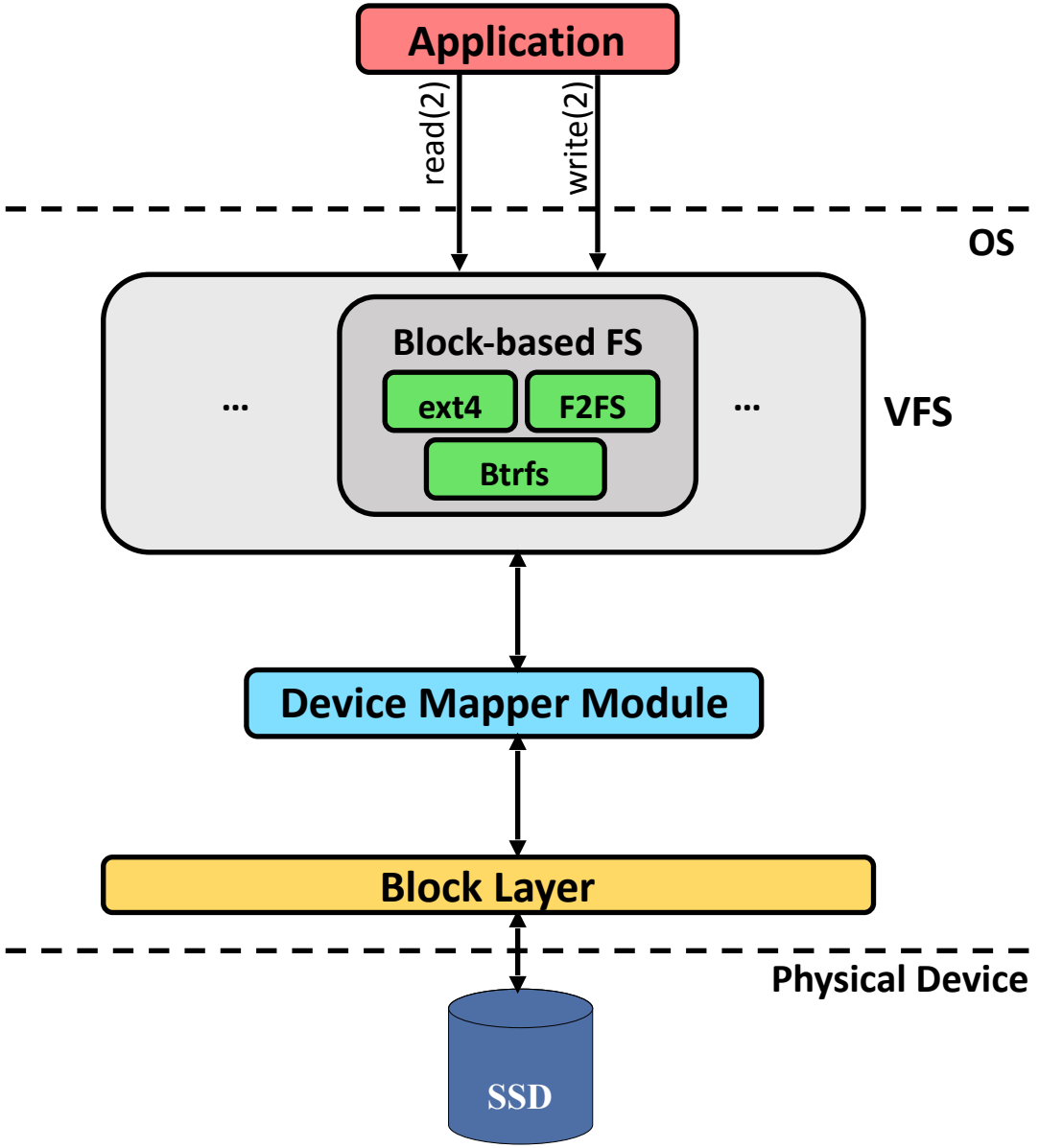
Detection	Recovery	Fsck
Error Code		
Sanity		



How do file systems detect and recover from errors?

- Categorize each file system's detection and recovery policies:
  - Across all visible aspects, such as *logs*, *return codes*, etc.
  - Check how effectively ***fsck*** recovers the file system.

Detection	Recovery	Fsck
Error Code	*Retry*	
Sanity	*Propagate*	

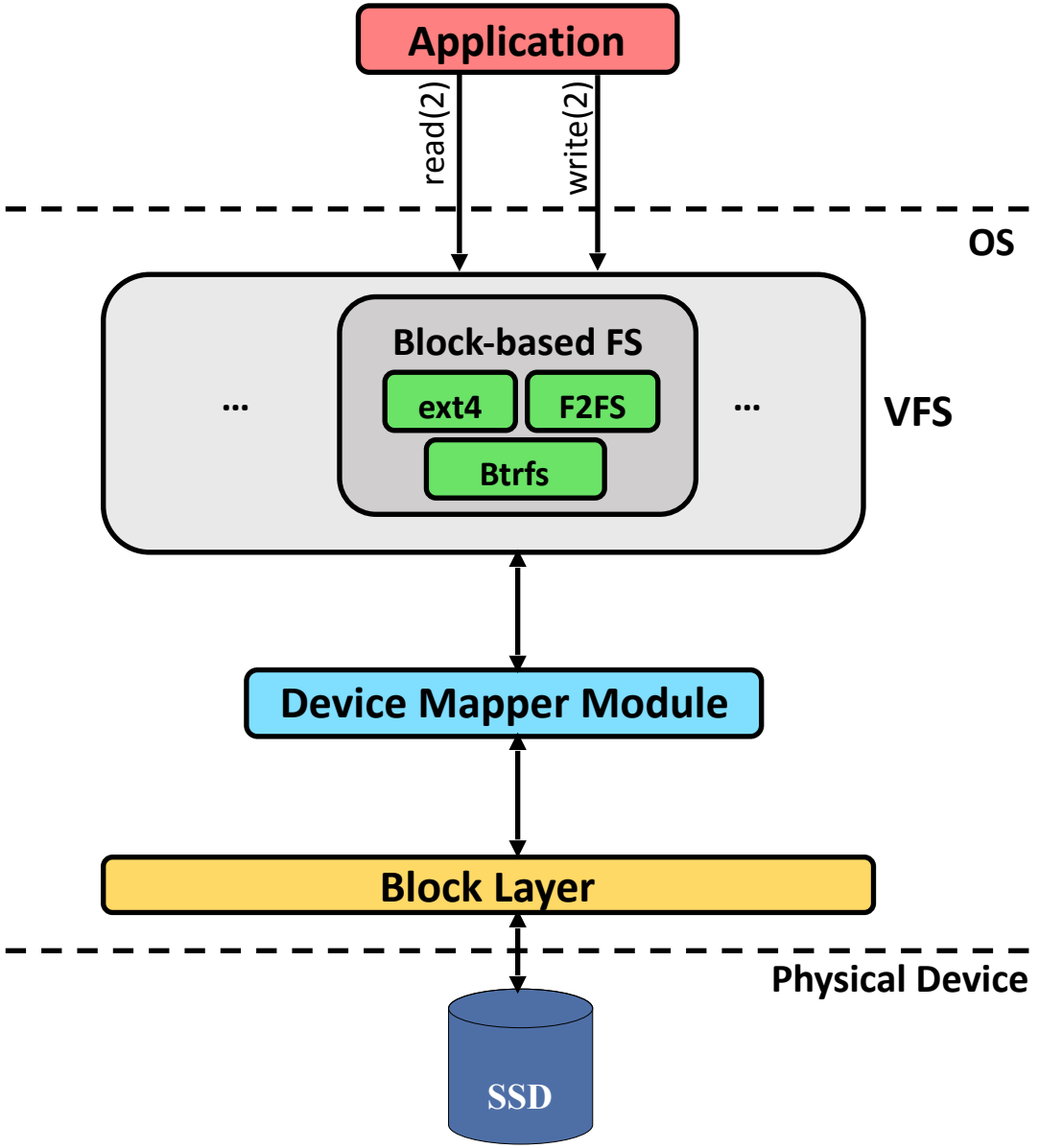


How do file systems detect and recover from errors?

- Categorize each file system's detection and recovery policies:
  - Across all visible aspects, such as *logs*, *return codes*, etc.
  - Check how effectively ***fsck*** recovers the file system.

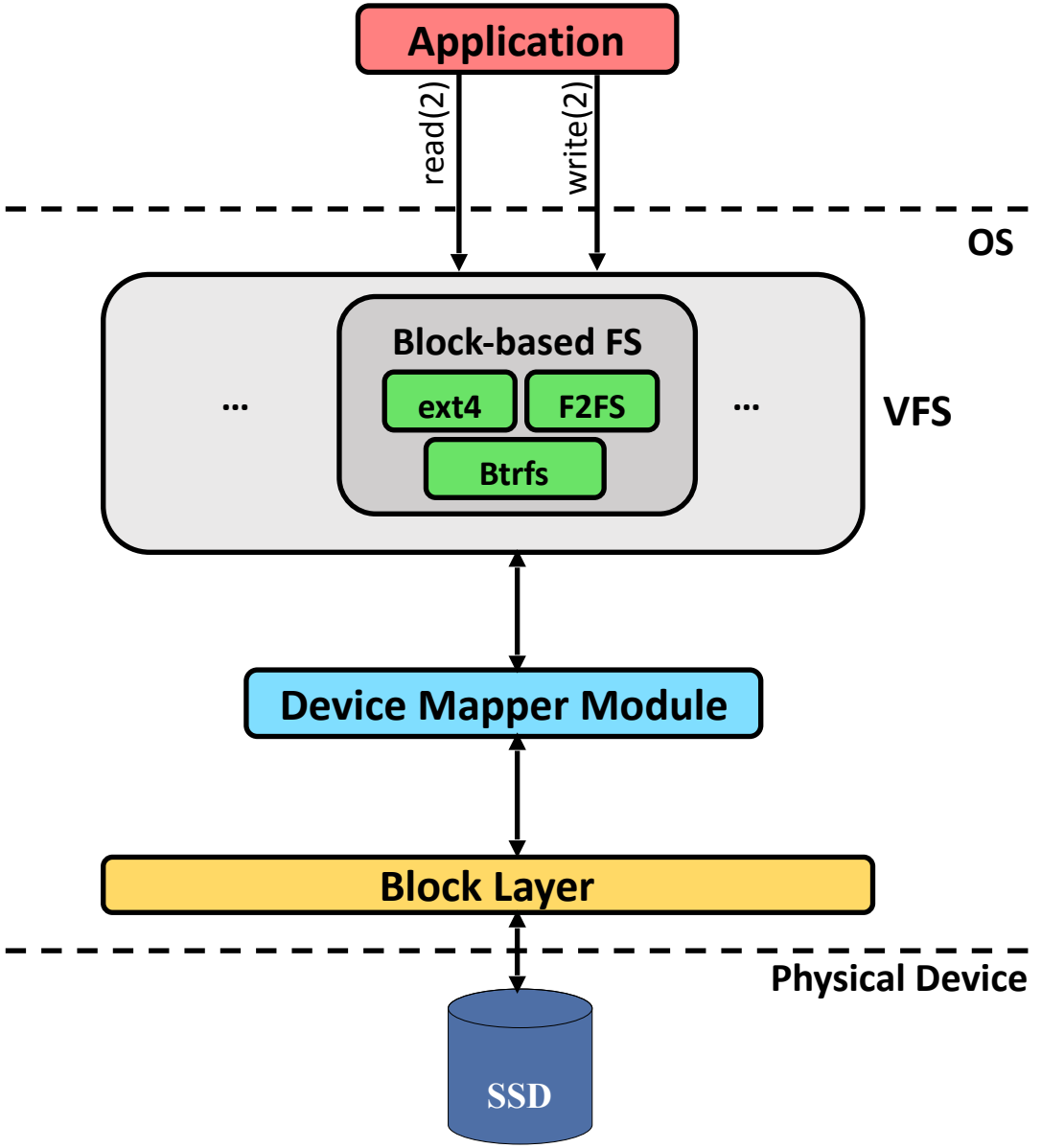
Detection	Recovery	Fsck*
Zero	*Zero*	*Crash/Fail*
Error Code	*Retry*	*Fail*
Sanity	*Propagate*	*Partial*
Redundancy	*Previous*	*Original*
Fsck	*Stop*	*Full*

* Colors indicate severity.



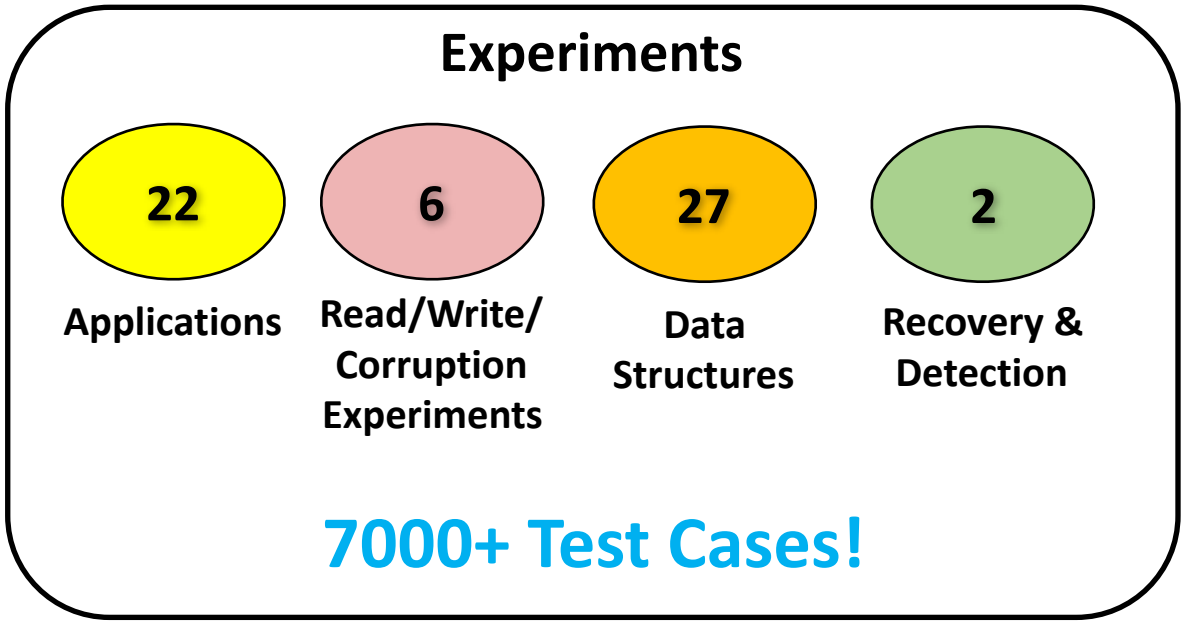
How do file systems detect and recover from errors?

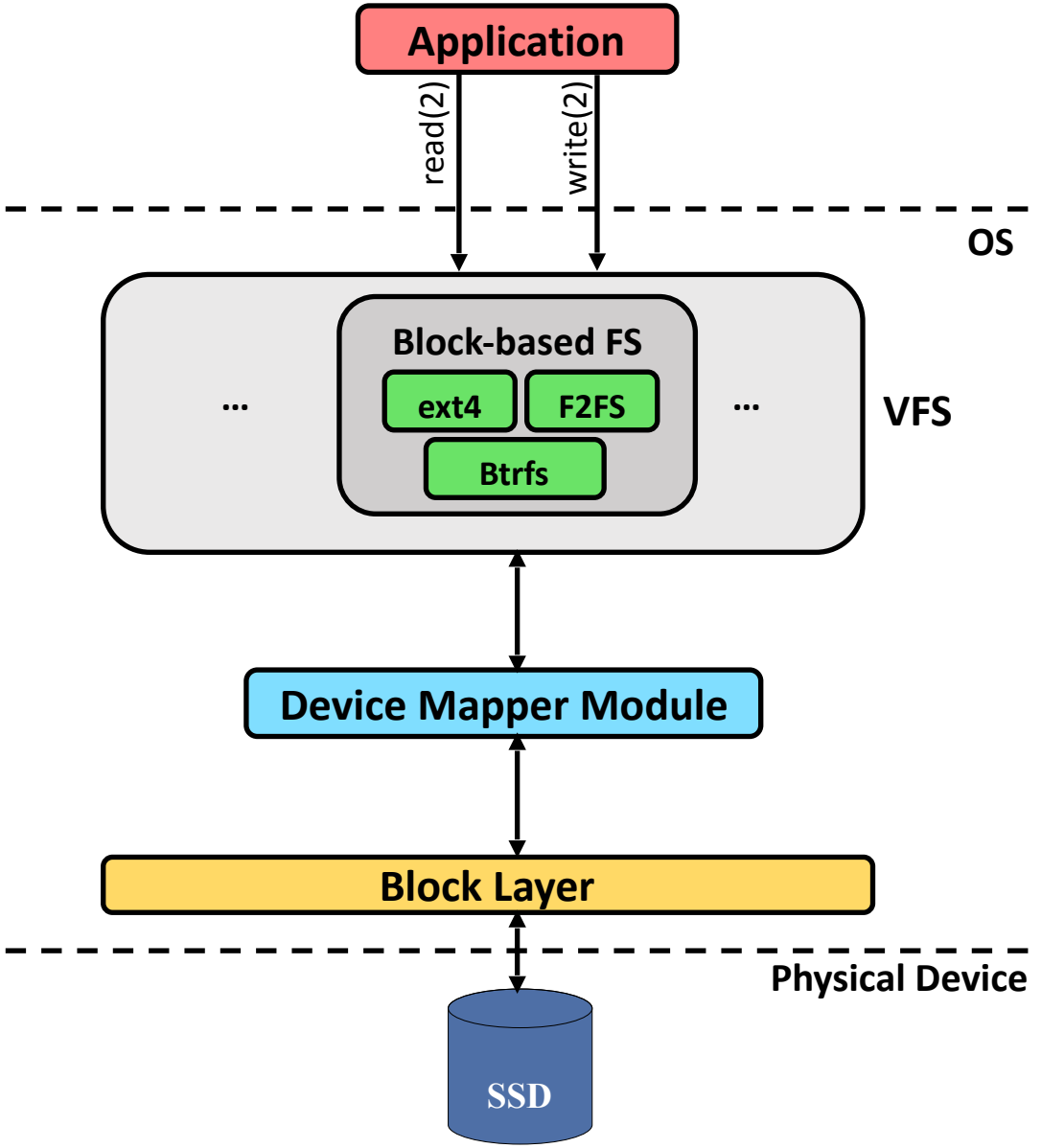
- Categorize each file system's detection and recovery policies:
  - Across all visible aspects, such as *logs*, *return codes*, etc.
  - Check how effectively ***fsck*** recovers the file system.



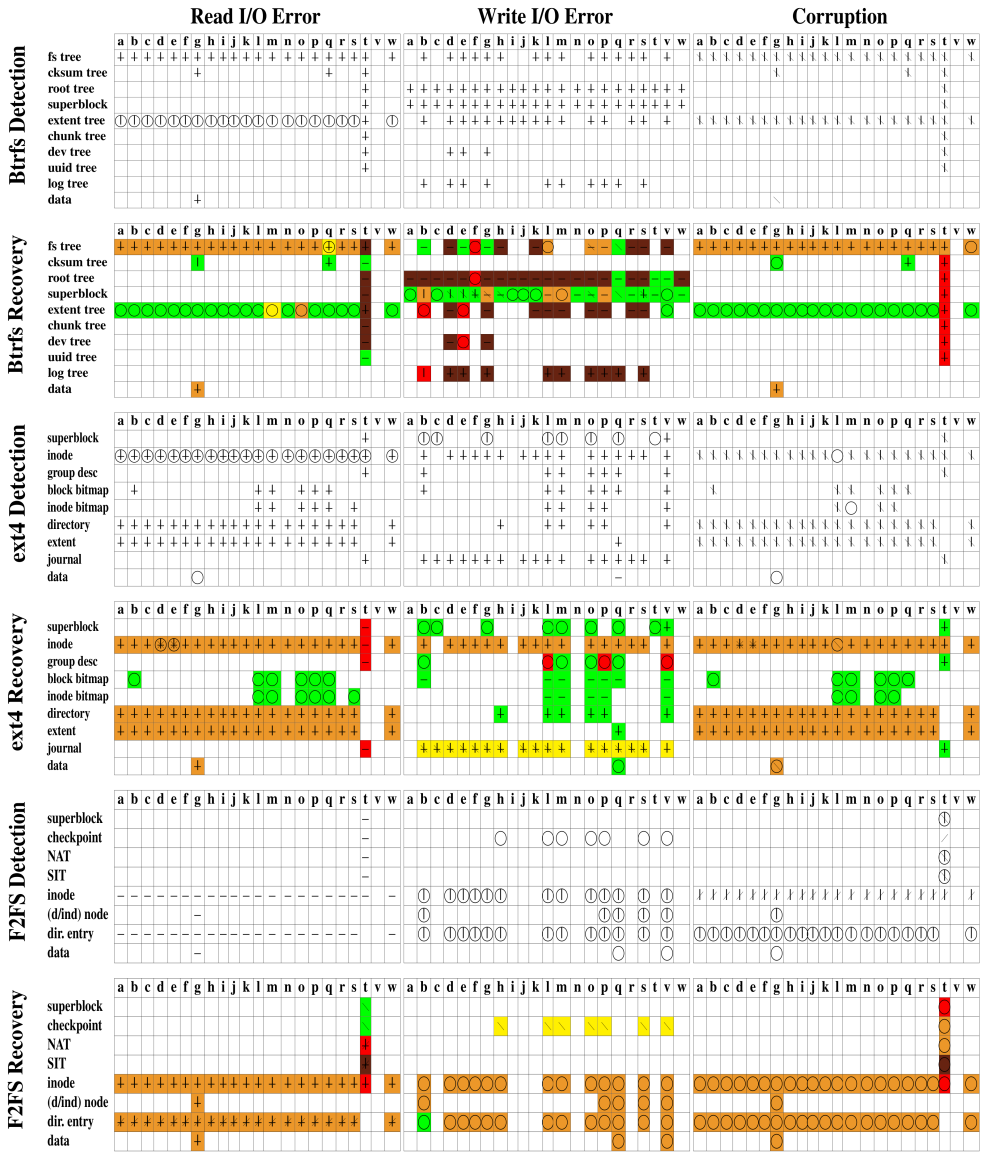
How do file systems detect and recover from errors?

- Categorize each file system's detection and recovery policies:
  - Across all visible aspects, such as *logs*, *return codes*, etc.
  - Check how effectively ***fsck*** recovers the file system.

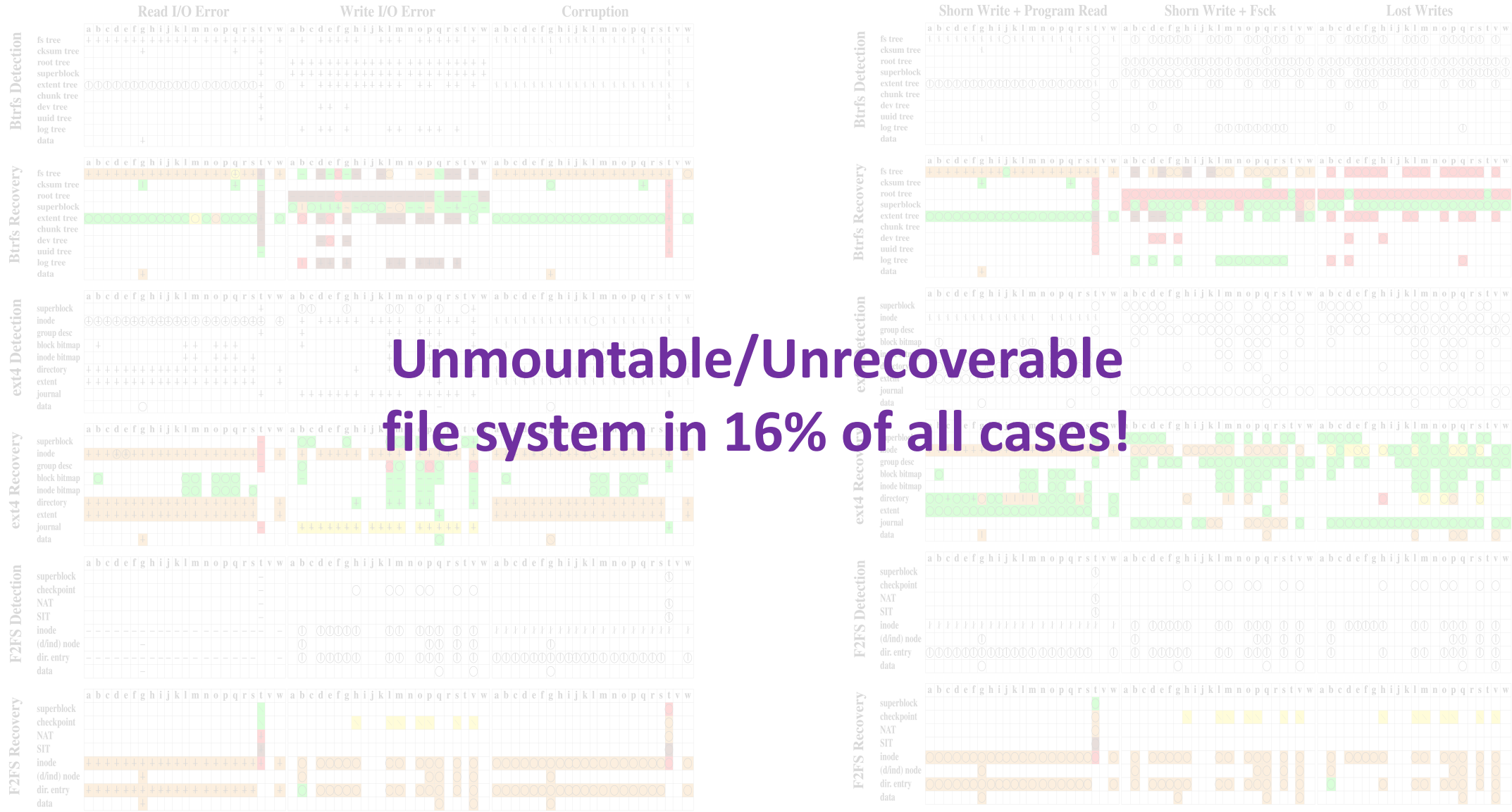




Results – Overview (1/2)



Results – Overview (1/2)



Unmountable/Unrecoverable file system in 16% of all cases!

Results – Overview (2/2)

File System	Detection	Recovery

Results – Overview (2/2)

File System	Detection	Recovery
ext4		

Results – Overview (2/2)

File System	Detection	Recovery
ext4		
Btrfs		

Results – Overview (2/2)

File System	Detection	Recovery
ext4		
Btrfs		
F2FS		

ext4 Results

The Good News

- Capable of recovering from a large range of fault scenarios.
- Little use of checksums:
  - Still, it can deal with corruption and shorn writes due to a very rich set of sanity checks.
- System checker capable of reconstructing several data structures:
  - Inode bitmaps, block bitmaps, group descriptor block.

The Bad News

- Lost and Shorn writes:
  - A few data structures **cannot** be recovered:
    - ➢Inode block → data loss.

The overall reliability of ext4 is significantly better compared to ext3!

Btrfs Results

The Good News

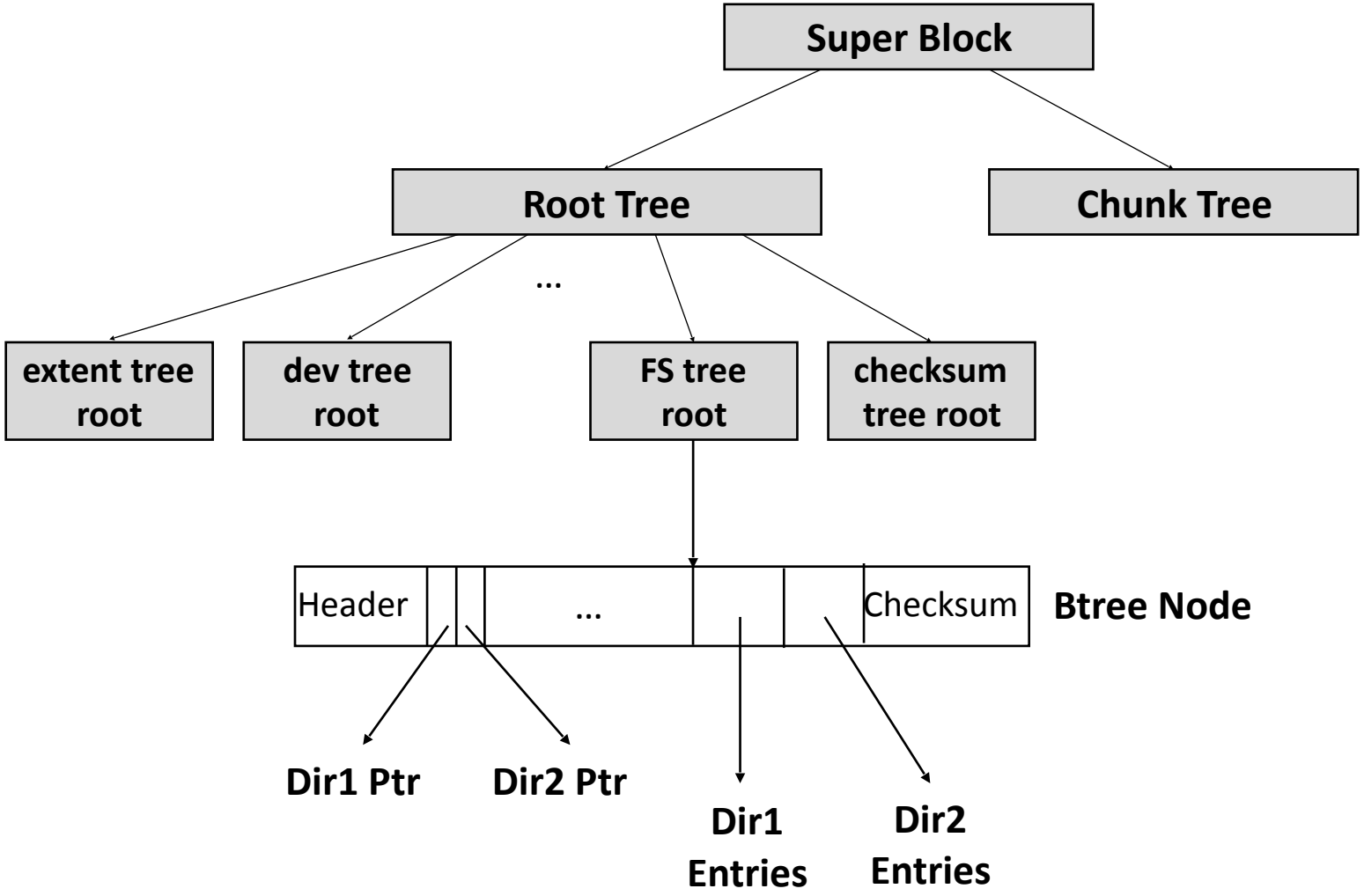
- Consistently detects all I/O errors, as well as corruption events (due to checksums).

The Bad News

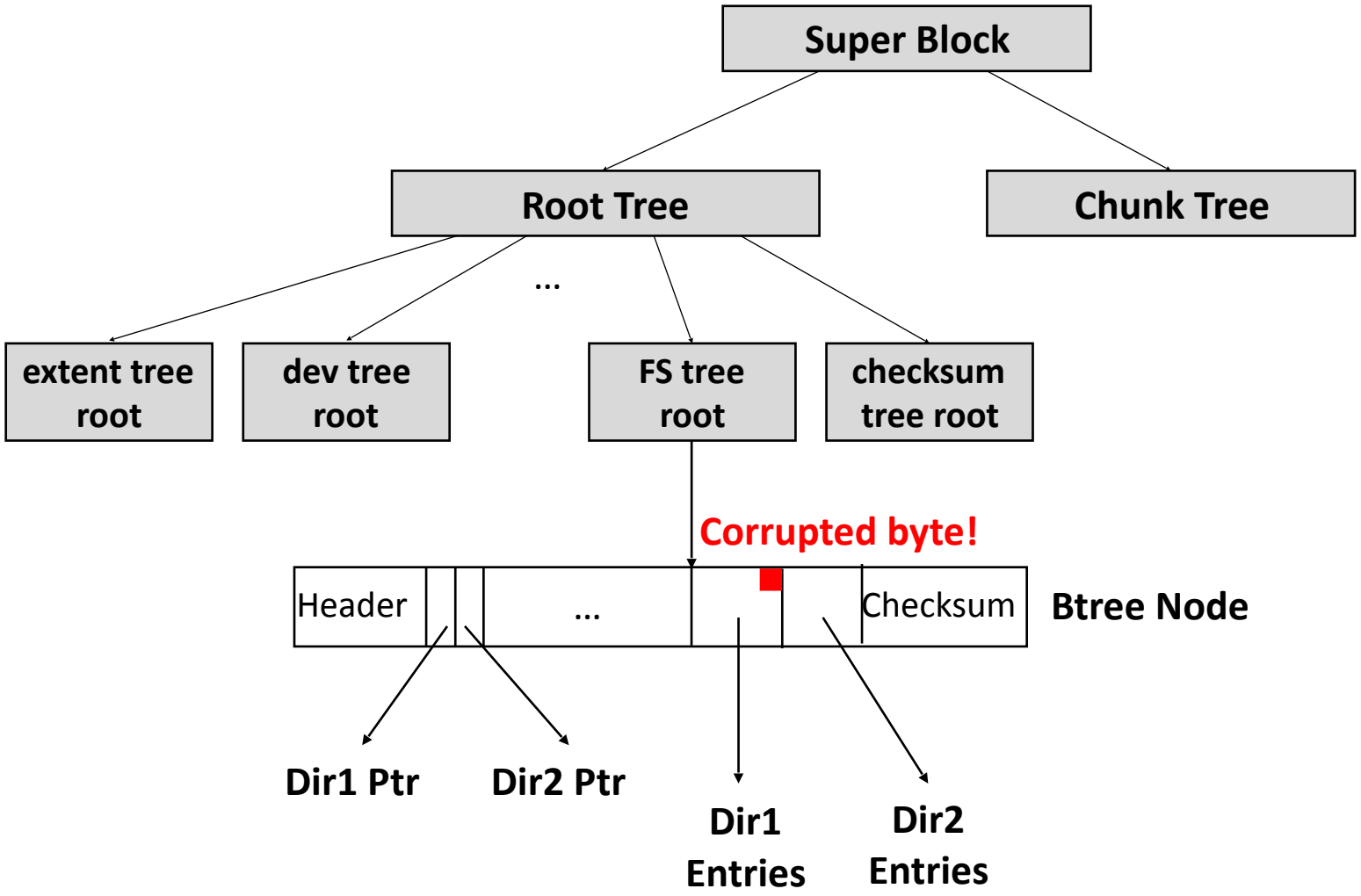
- Metadata replication is disabled for SSDs*!
- Makes use of node level checksums…

* https://btrfs.wiki.kernel.org/index.php/Manpage/mkfs.btrfs

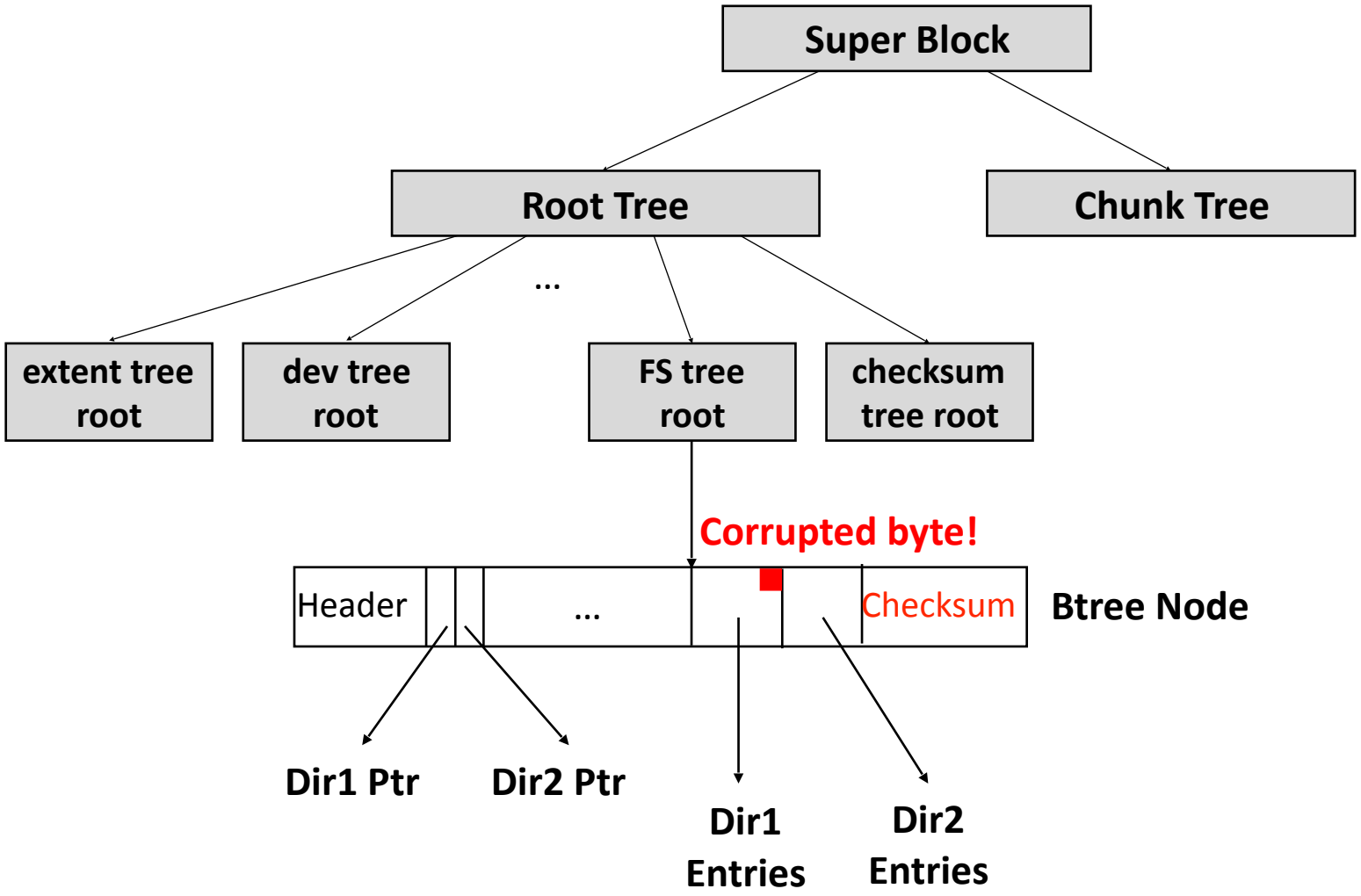
Btrfs Corruption Scenario



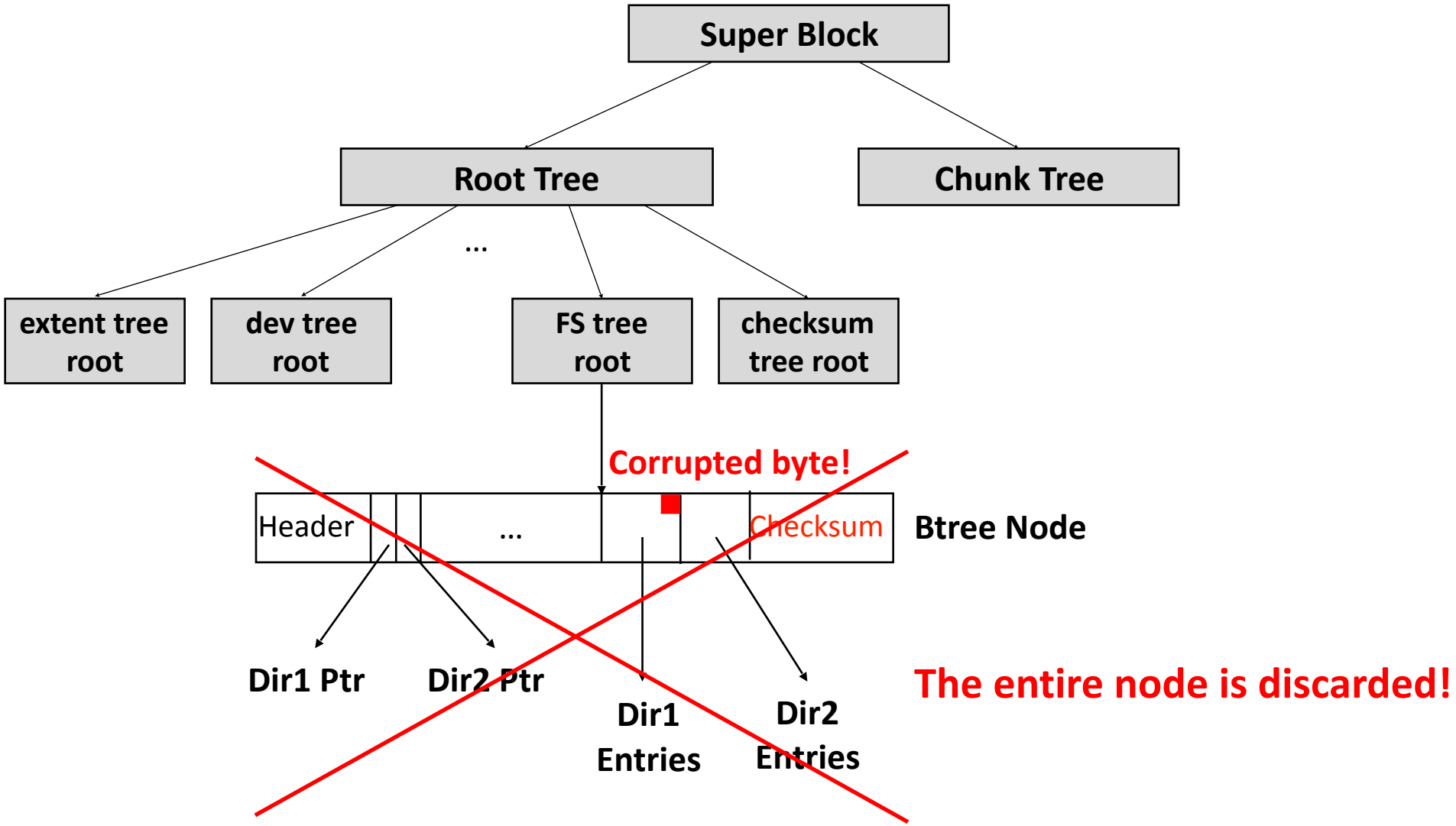
Btrfs Corruption Scenario



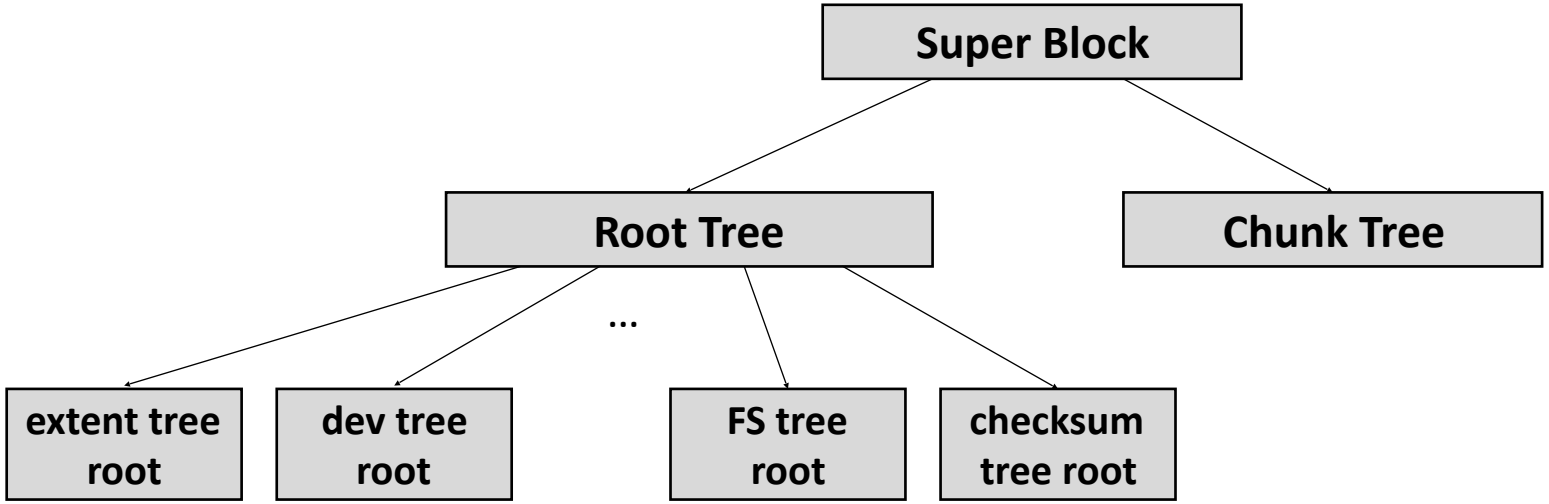
Btrfs Corruption Scenario



Btrfs Corruption Scenario



Btrfs Corruption Scenario



Data Loss!

Btrfs Results

The Good News

- Consistently detects all I/O errors, as well as corruption events (due to checksums).

The Bad News

- Metadata replication is disabled for SSDs!
- Makes use of node level checksums → an entire node is removed even if a single byte becomes corrupted!
- Does not always make use of the existing redundancy:
  - Two independent data structures for a directory:
    - ➢*DIR_ITEM* and *DIR_INDEX*.
  - If one becomes corrupted, the other is not used for recovery!
- Several cases of unmountable file system; a few crashes:
  - The file system cannot be mounted even after *btrfsck* is invoked.

F2FS Results

The Good News

- Read errors are detected and appropriately propagated in nearly all cases.
- Inodes and checkpoints are protected using checksums.
- The file system checker can bring the file system to a consistent state in some cases!

The Bad News

- Consistently fails to detect and report any write errors!
- Cannot deal with *lost* and *shorn* writes effectively → data loss.
- Corruption events can have severe repercussions.

Implications

- Verify the correctness of metadata through sanity checks, especially when metadata is not protected against corruption.

- Checksums can be a double-edged sword:
  - Increase error detection.
  - Coarse granularity checksums can lead to severe data loss.

- A few **key data structures** cause maximum recovery failures:
  - **ext4:** the journal's superblock and the inode of the root directory.
  - **Btrfs:** the root node of *fstree*.
  - **F2FS:** the inode of the root directory.

Thank you!

Questions?

Github: https://github.com/uoftsystems/dm-inject