50th International Conference on Very Large Databases

When Amnesia Strikes: Understanding and Reproducing Data Loss Bugs with Fault Injection

Maria Ramos, João Azevedo, Kyle Kingsbury*, José Pereira, Tânia Esteves, Ricardo Macedo and João Paulo

INESC TEC
University of Minho
Jepsen*

Crash consistency

Context

- **Efficient** data access and data **durability** are key issues for many systems (*e.g.* databases, key-value stores).
- Storage systems use caches to avoid disk accesses.
- Cached data is **lost** in the event of a power or OS failure.

Crash consistency

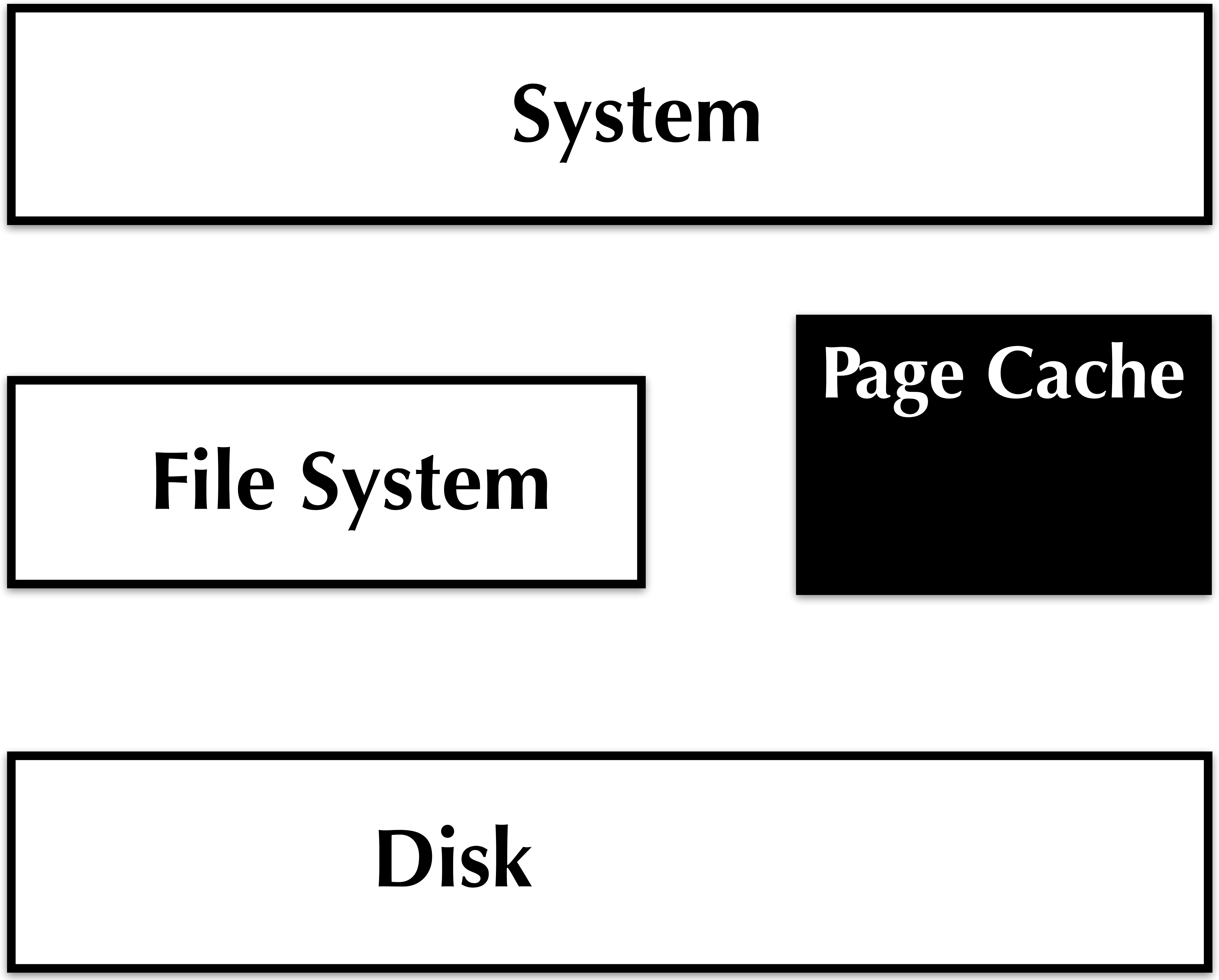
Context

- **Efficient** data access and data **durability** are key issues for many systems (*e.g.* databases, key-value stores).
- Storage systems use caches to avoid disk accesses.
- Cached data is **lost** in the event of a power or OS failure.

Trade-off between **performance** and **reliability**

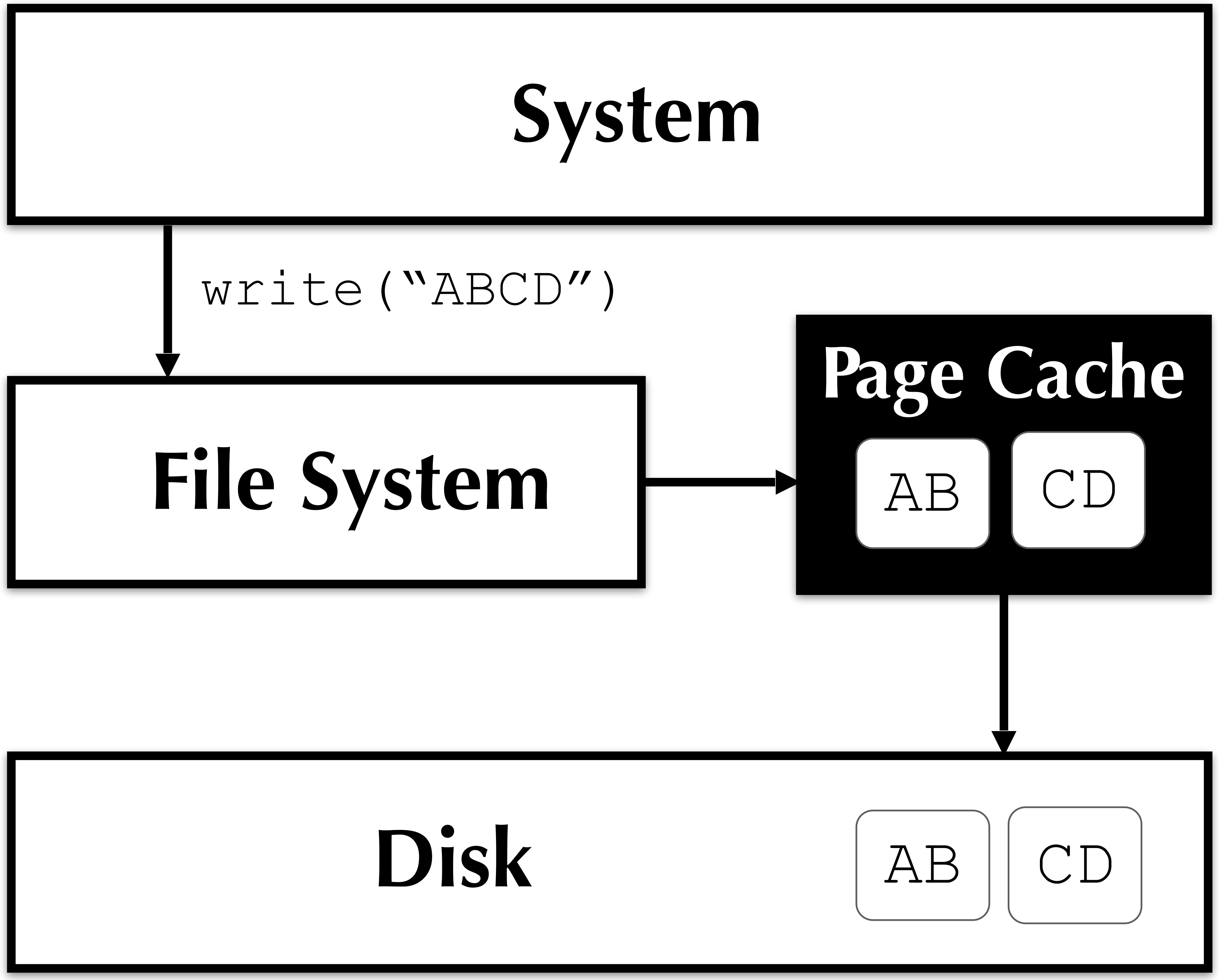
Crash consistency

Context



Crash consistency

Context

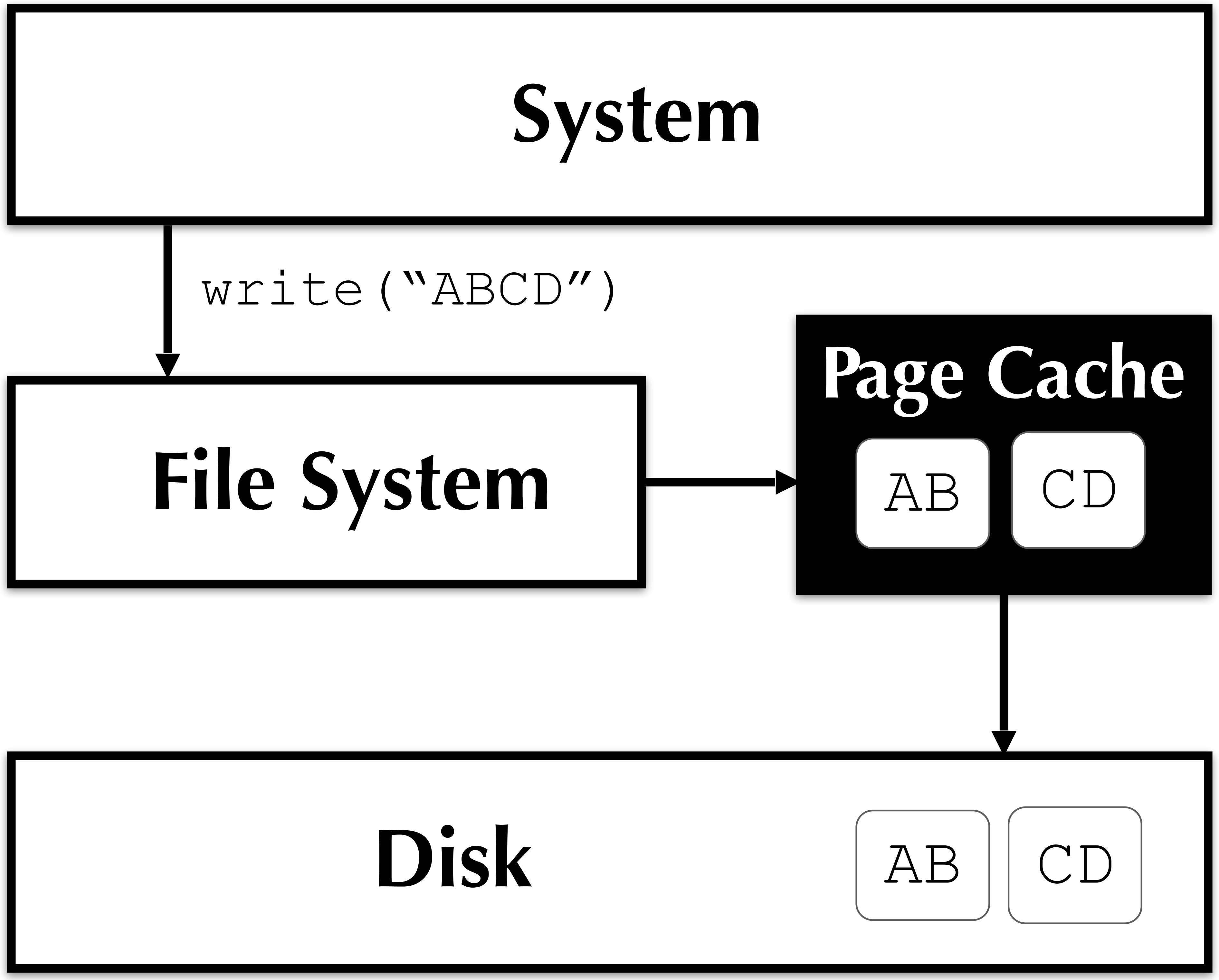


Cached data is flushed to disk by:

- OS pressure
- `fsync()` call

Crash consistency

Context



Cached data is flushed to disk by:

- OS pressure
- `fsync()` call

Writes can be **asynchronous**

Crash consistency

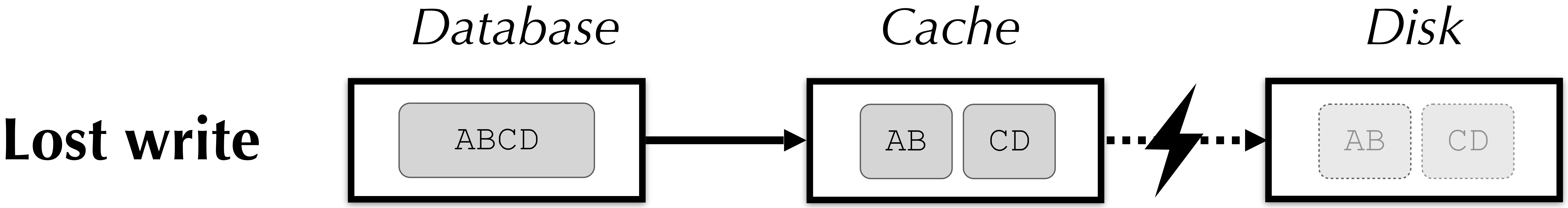
Problem

- Writes can be persisted **partially** and **out-of-order**.

Crash consistency

Problem

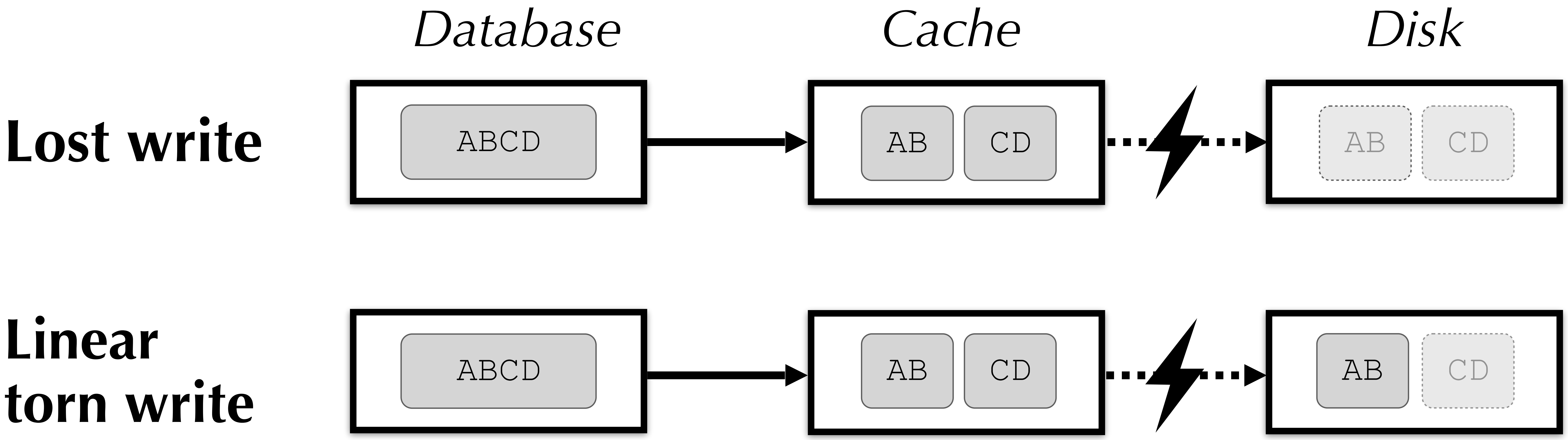
- Writes can be persisted **partially** and **out-of-order**.



Crash consistency

Problem

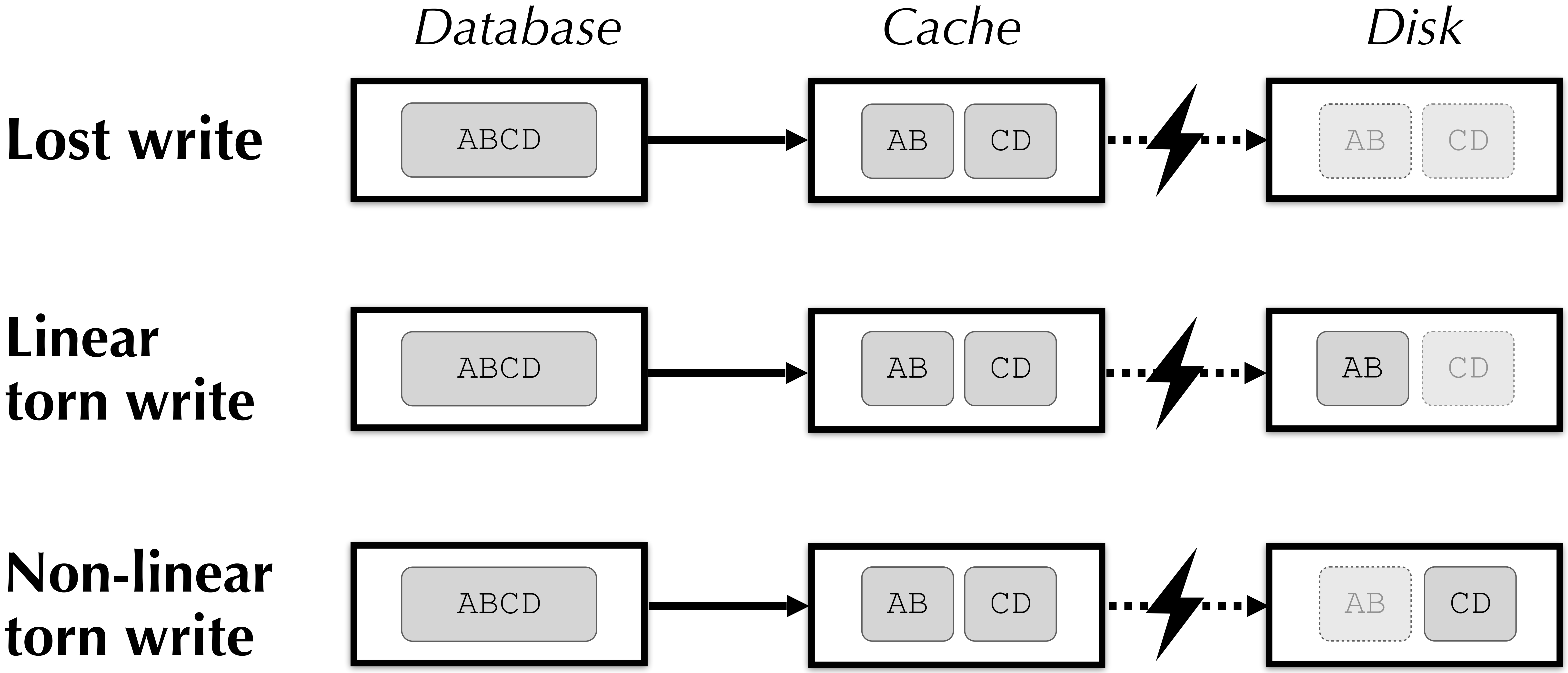
- Writes can be persisted **partially** and **out-of-order**.



Crash consistency

Problem

- Writes can be persisted **partially** and **out-of-order**.



Crash consistency bugs

Study

- Study of 12 reported crash consistency bugs:
  - **symptoms** reported
  - **reproduction** steps
  - applied/suggested **fixes**
- Studied systems:

etcd

Redis

LevelDB

PostgreSQL

Lightning Network Daemon

ZooKeeper

Study

Bug classification

- Known bugs

Study

Bug classification

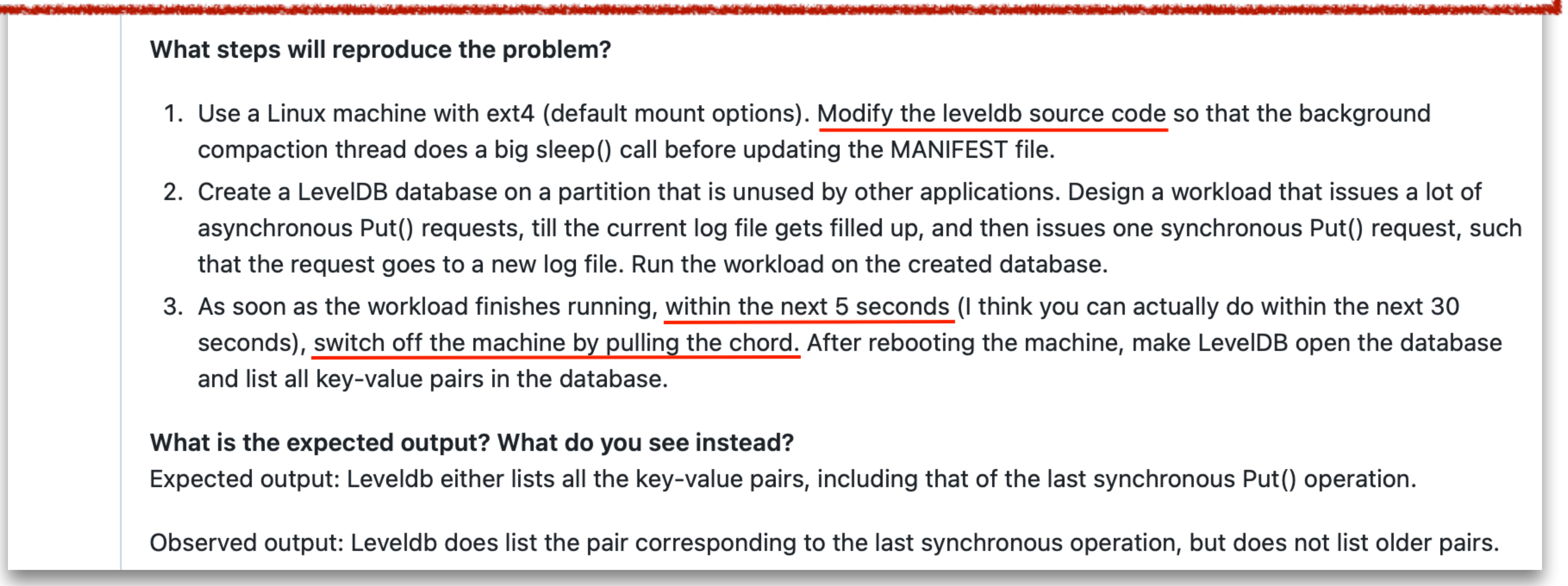
- Known bugs
- Ambiguous bugs

Study

Bug classification

- Known bugs
- Ambiguous bugs

"modify the leveldb source code"
"within the next 5 seconds, switch off the machine by pulling the cord"



What steps will reproduce the problem?

1. Use a Linux machine with ext4 (default mount options). Modify the leveldb source code so that the background compaction thread does a big sleep() call before updating the MANIFEST file.
2. Create a LevelDB database on a partition that is unused by other applications. Design a workload that issues a lot of asynchronous Put() requests, till the current log file gets filled up, and then issues one synchronous Put() request, such that the request goes to a new log file. Run the workload on the created database.
3. As soon as the workload finishes running, within the next 5 seconds (I think you can actually do within the next 30 seconds), switch off the machine by pulling the chord. After rebooting the machine, make LevelDB open the database and list all key-value pairs in the database.

What is the expected output? What do you see instead?
Expected output: Leveldb either lists all the key-value pairs, including that of the last synchronous Put() operation.

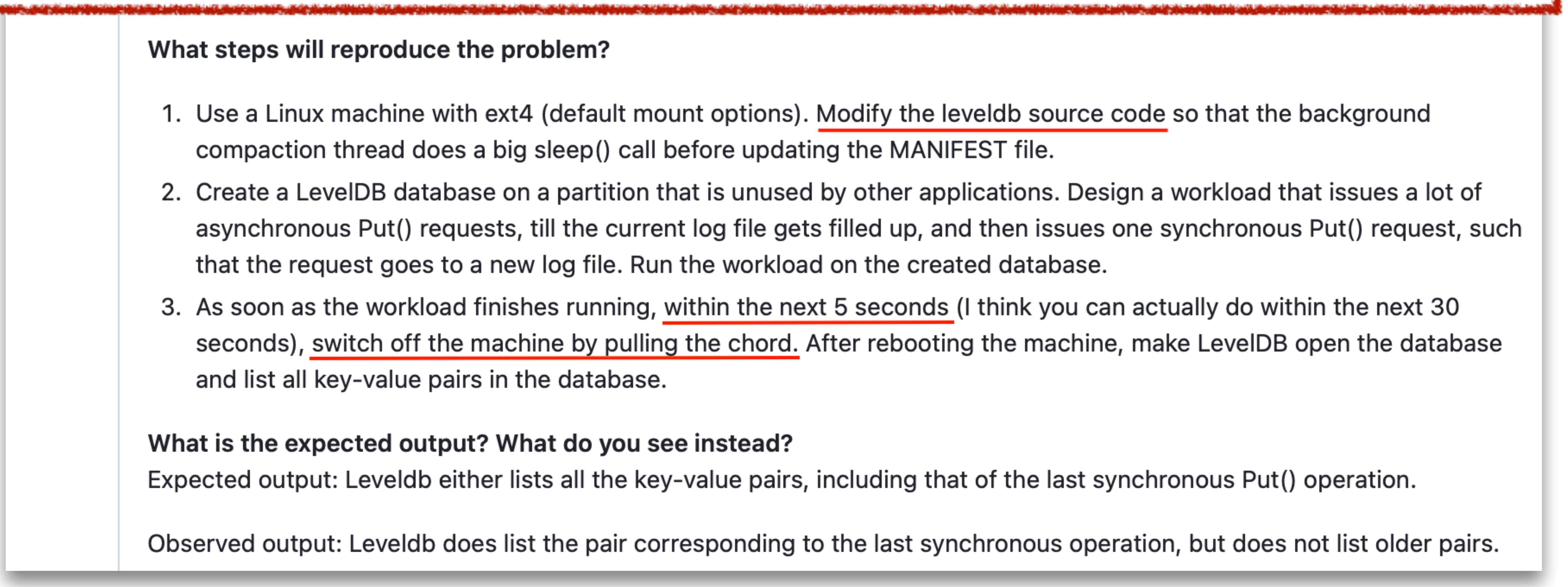
Observed output: Leveldb does list the pair corresponding to the last synchronous operation, but does not list older pairs.

Study

Bug classification

- Known bugs
- Ambiguous bugs

"modify the leveldb source code"
"within the next 5 seconds, switch off the machine by pulling the cord"



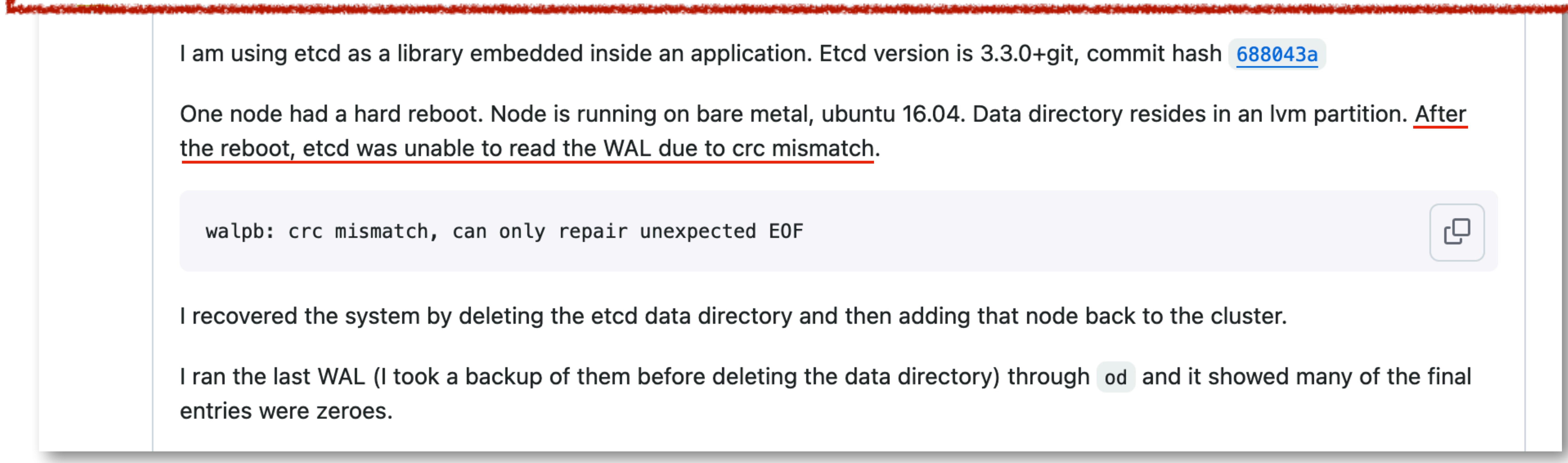
What steps will reproduce the problem?

1. Use a Linux machine with ext4 (default mount options). Modify the leveldb source code so that the background compaction thread does a big sleep() call before updating the MANIFEST file.
2. Create a LevelDB database on a partition that is unused by other applications. Design a workload that issues a lot of asynchronous Put() requests, till the current log file gets filled up, and then issues one synchronous Put() request, such that the request goes to a new log file. Run the workload on the created database.
3. As soon as the workload finishes running, within the next 5 seconds (I think you can actually do within the next 30 seconds), switch off the machine by pulling the chord. After rebooting the machine, make LevelDB open the database and list all key-value pairs in the database.

What is the expected output? What do you see instead?
Expected output: Leveldb either lists all the key-value pairs, including that of the last synchronous Put() operation.

Observed output: Leveldb does list the pair corresponding to the last synchronous operation, but does not list older pairs.

"After the reboot, etcd was unable to read the WAL due to crc mismatch"



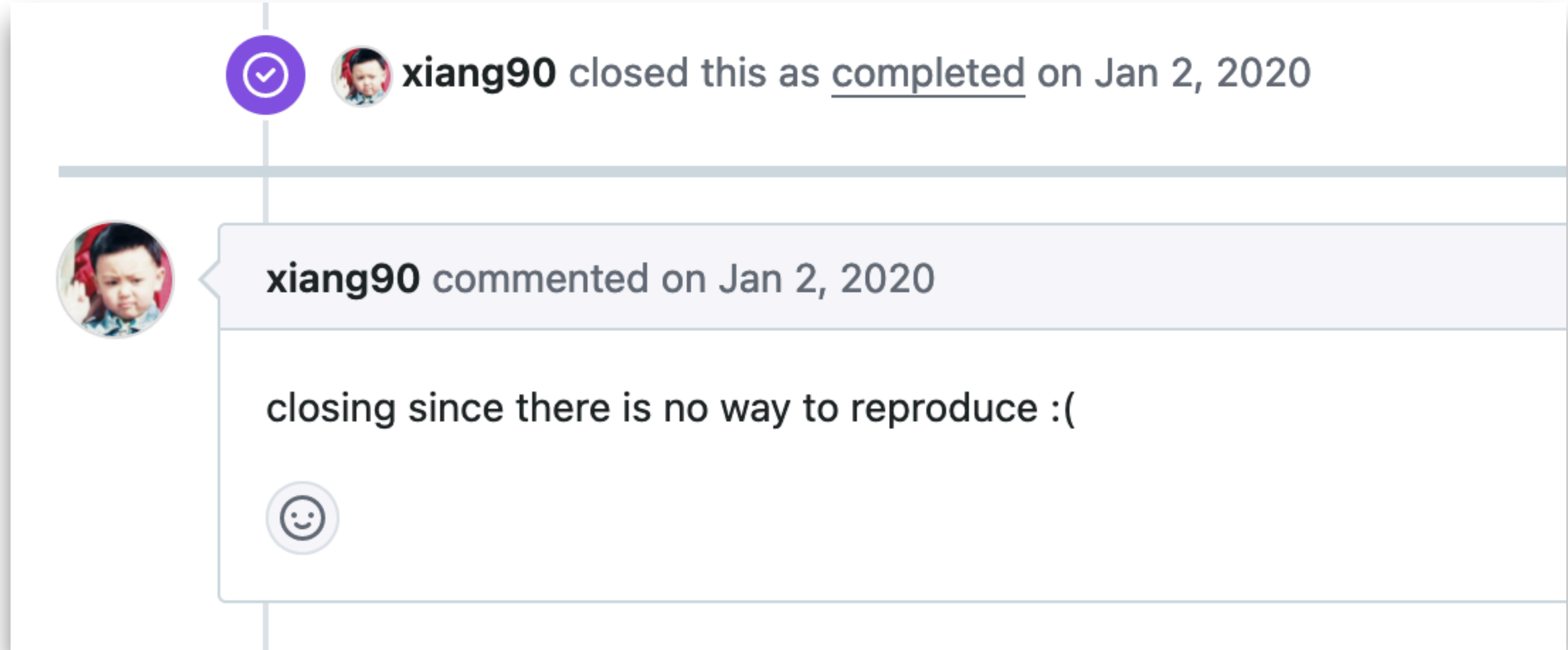
I am using etcd as a library embedded inside an application. Etcd version is 3.3.0+git, commit hash `688043a`

One node had a hard reboot. Node is running on bare metal, ubuntu 16.04. Data directory resides in an lvm partition. After the reboot, etcd was unable to read the WAL due to crc mismatch.

```
walpb: crc mismatch, can only repair unexpected EOF
```

I recovered the system by deleting the etcd data directory and then adding that node back to the cluster.

I ran the last WAL (I took a backup of them before deleting the data directory) through `od` and it showed many of the final entries were zeroes.



xiang90 closed this as completed on Jan 2, 2020

xiang90 commented on Jan 2, 2020

closing since there is no way to reproduce :(

Study findings

Reproducibility

- Bugs are time-sensitive (*e.g.*, switching off machines in specific time windows)
- Modification of systems' codebases (*e.g.*, add `sleep()` call)
- Specific and complex deployments (*e.g.*, restore from cold backup)
- Lack of means to validate fixes

Study findings

Understanding ambiguous bugs

- Sometimes dismissed by developers
- Hard to associate reported errors with type of fault
- Developers lean on external tools (*e.g.*, strace)
- Data loss and corruption are common symptoms
- Similar error messages and affected files across different systems (*e.g.*, checksum errors and log files, respectively)

Goals

- **Non-intrusive** and **automated** way to reproduce storage-level data loss bugs.

  ✗ code changes ✗ power off machine ✗ counting time ✗ specific setup

- Provides insightful information for understanding the **root cause of bugs,** such as data that can be lost or torn.

- Users can use it directly or it can be used as a module of other testing systems.

Goals

- **Non-intrusive** and **automated** way to reproduce storage-level data loss bugs.

  ✗ code changes ✗ power off machine ✗ counting time ✗ specific setup

- Provides insightful information for understanding the **root cause of bugs,** such as data that can be lost or torn.

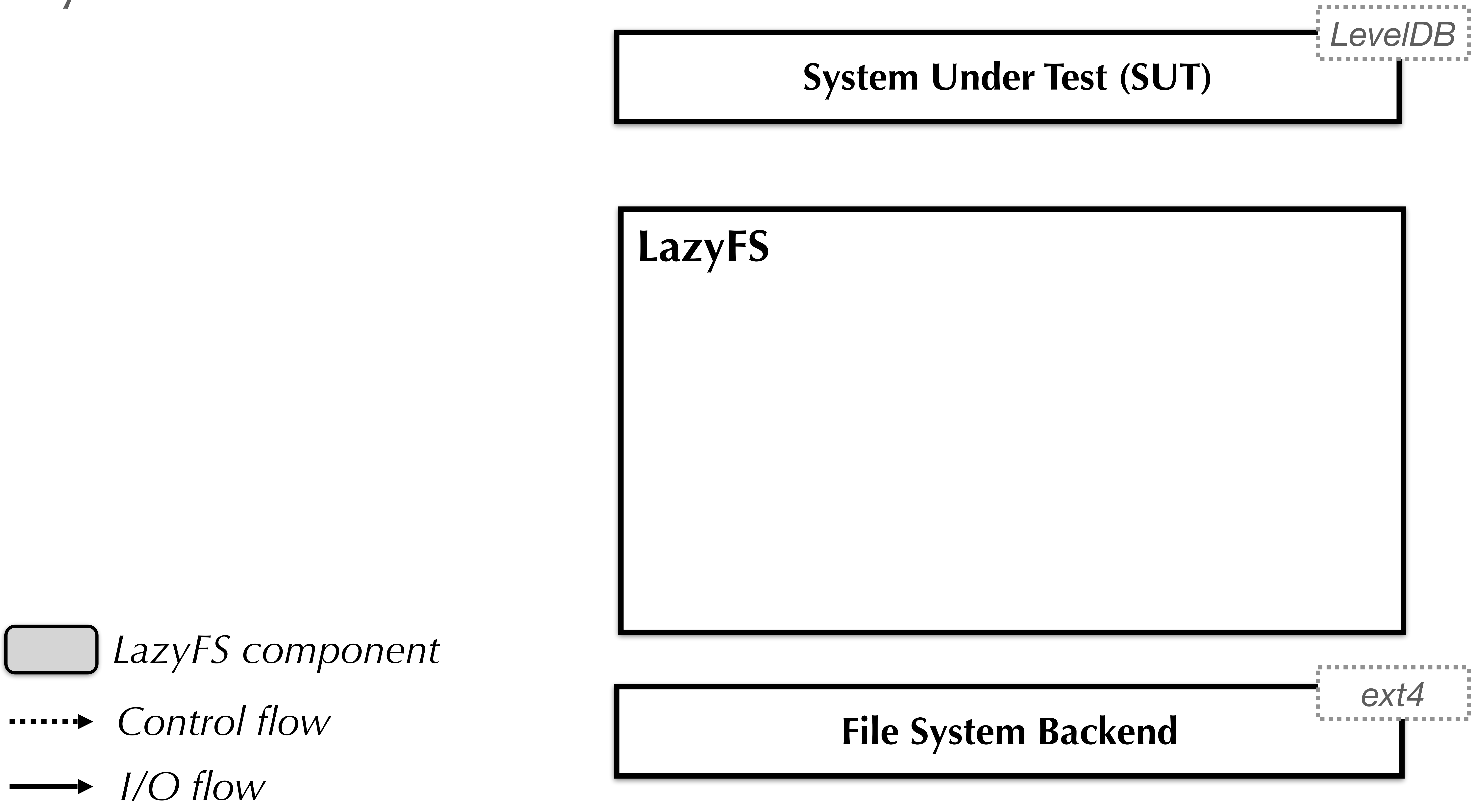
- Users can use it directly or it can be used as a module of other testing systems.

LazyFS

Software-based tool for injecting lost and torn write faults at the file system level.

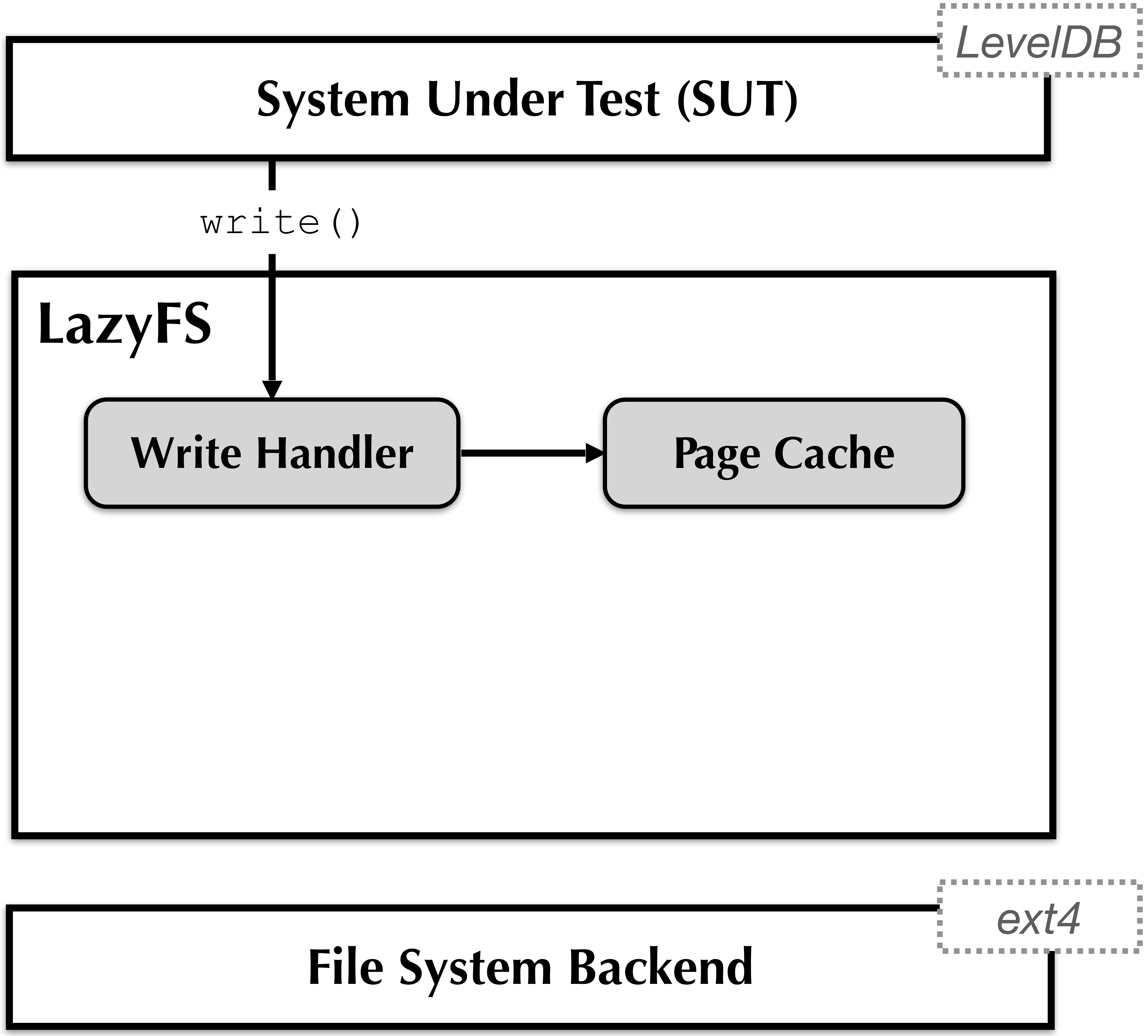
LazyFS

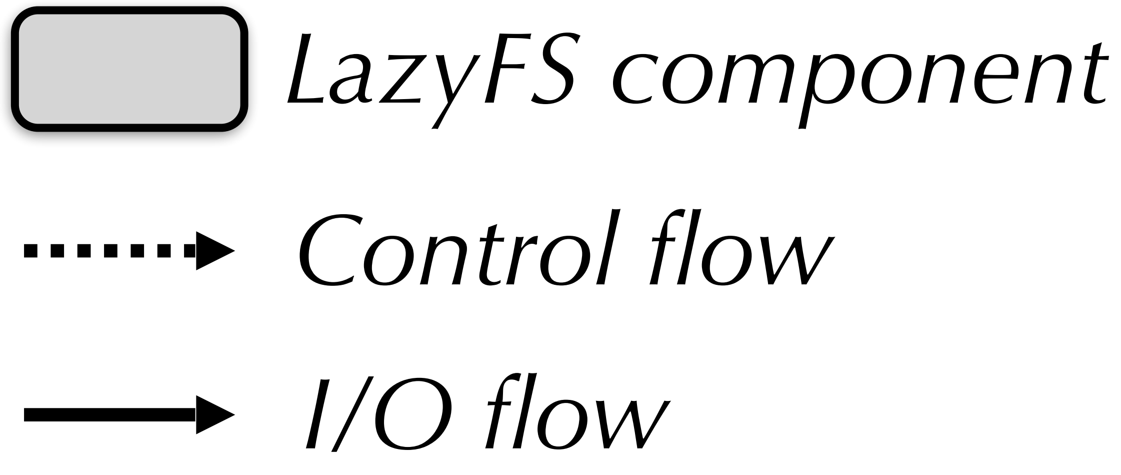
System overview



LazyFS

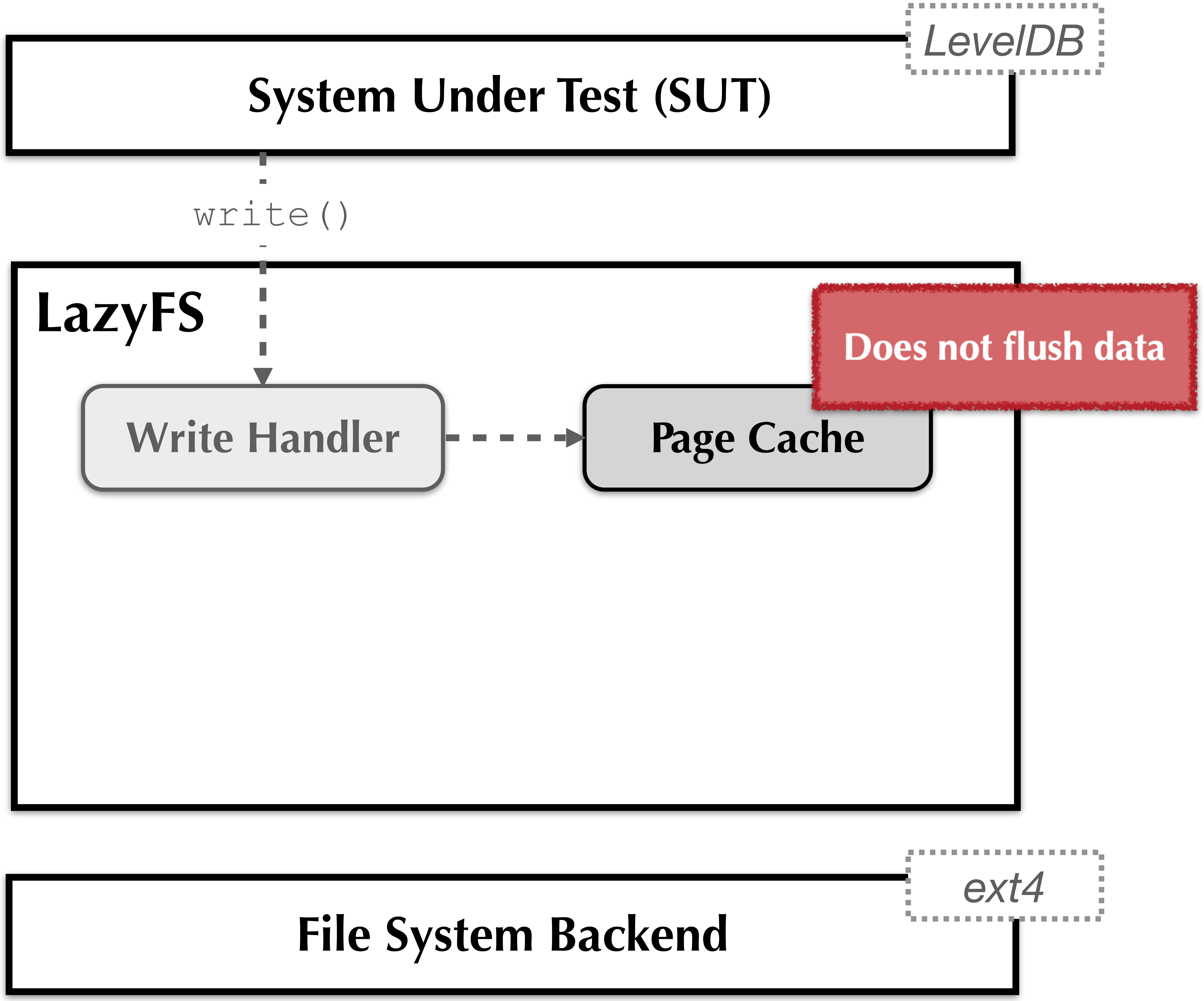
System overview





LazyFS

System overview



▢ *LazyFS component*

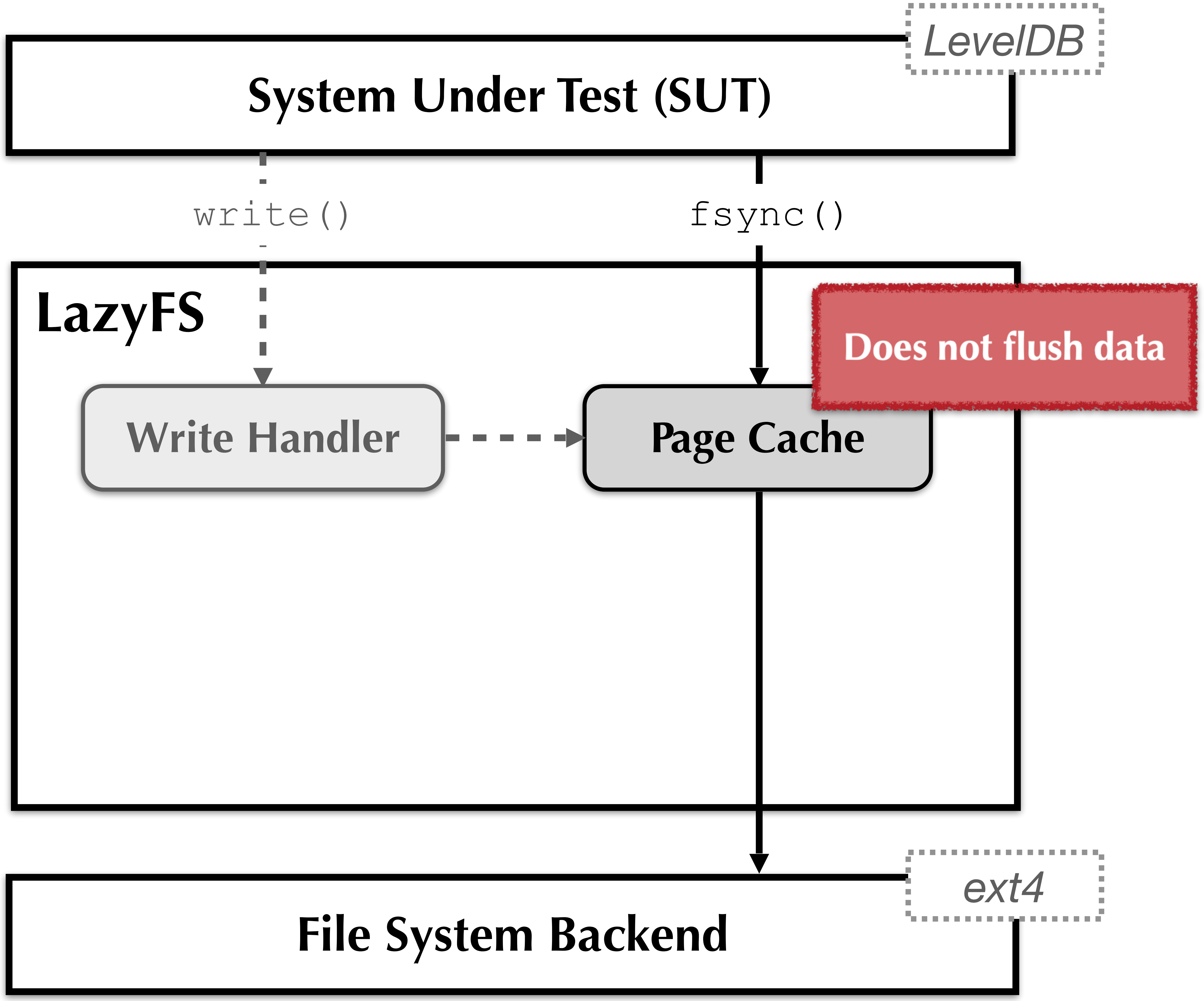
⋯⋯→ *Control flow*

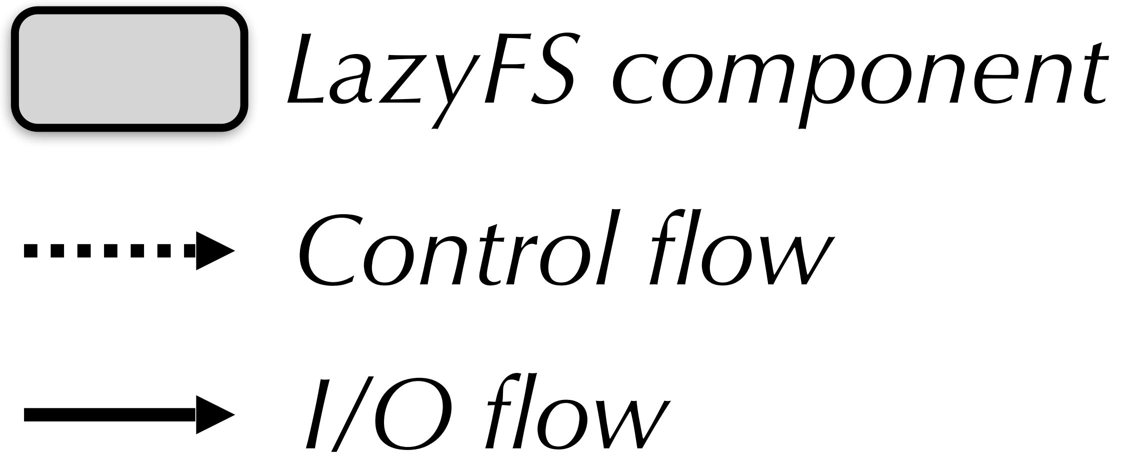
⟶ *I/O flow*

LazyFS

System overview

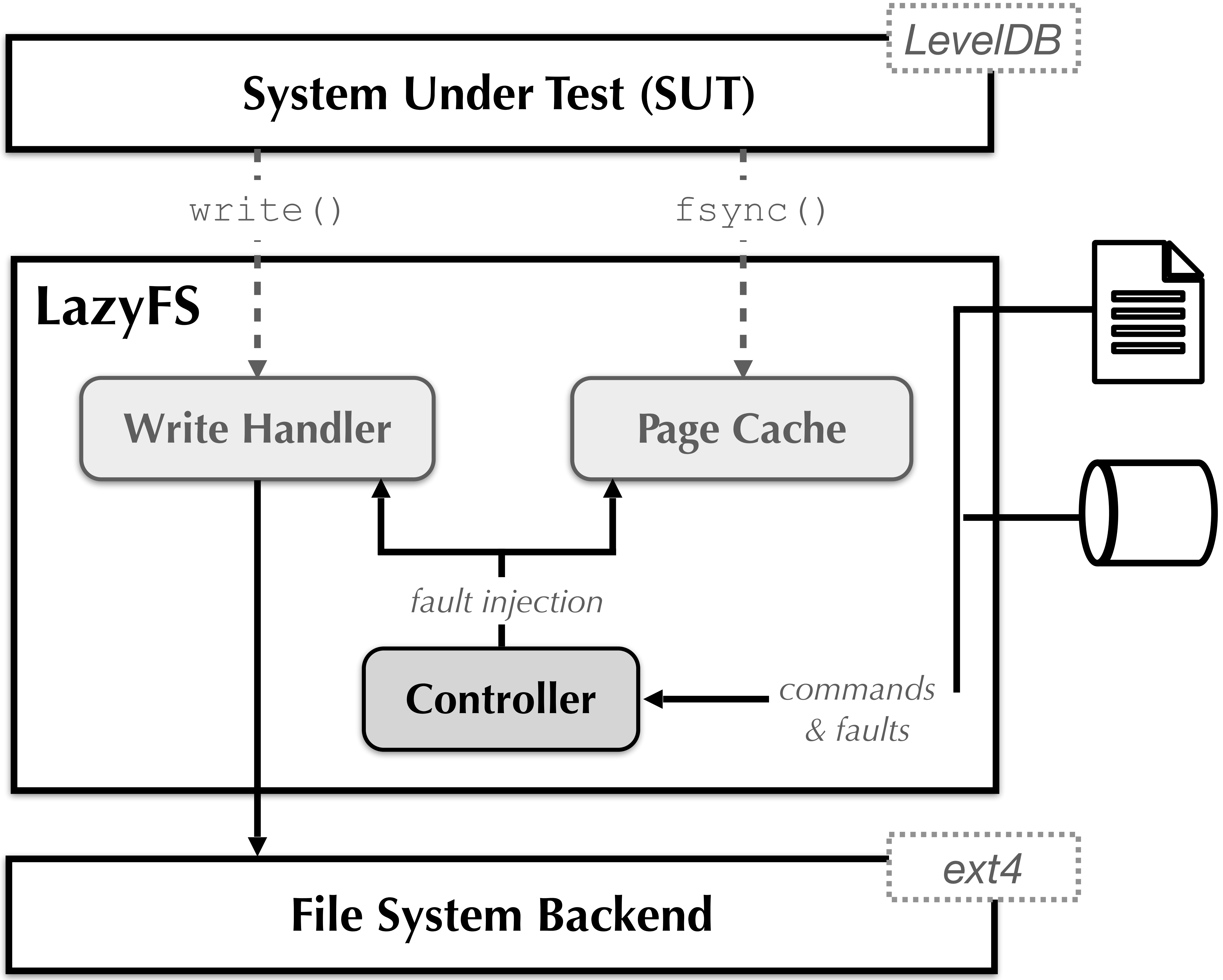
Controlling when data is written to disk allows to mimic the behavior of lost and torn writes





LazyFS

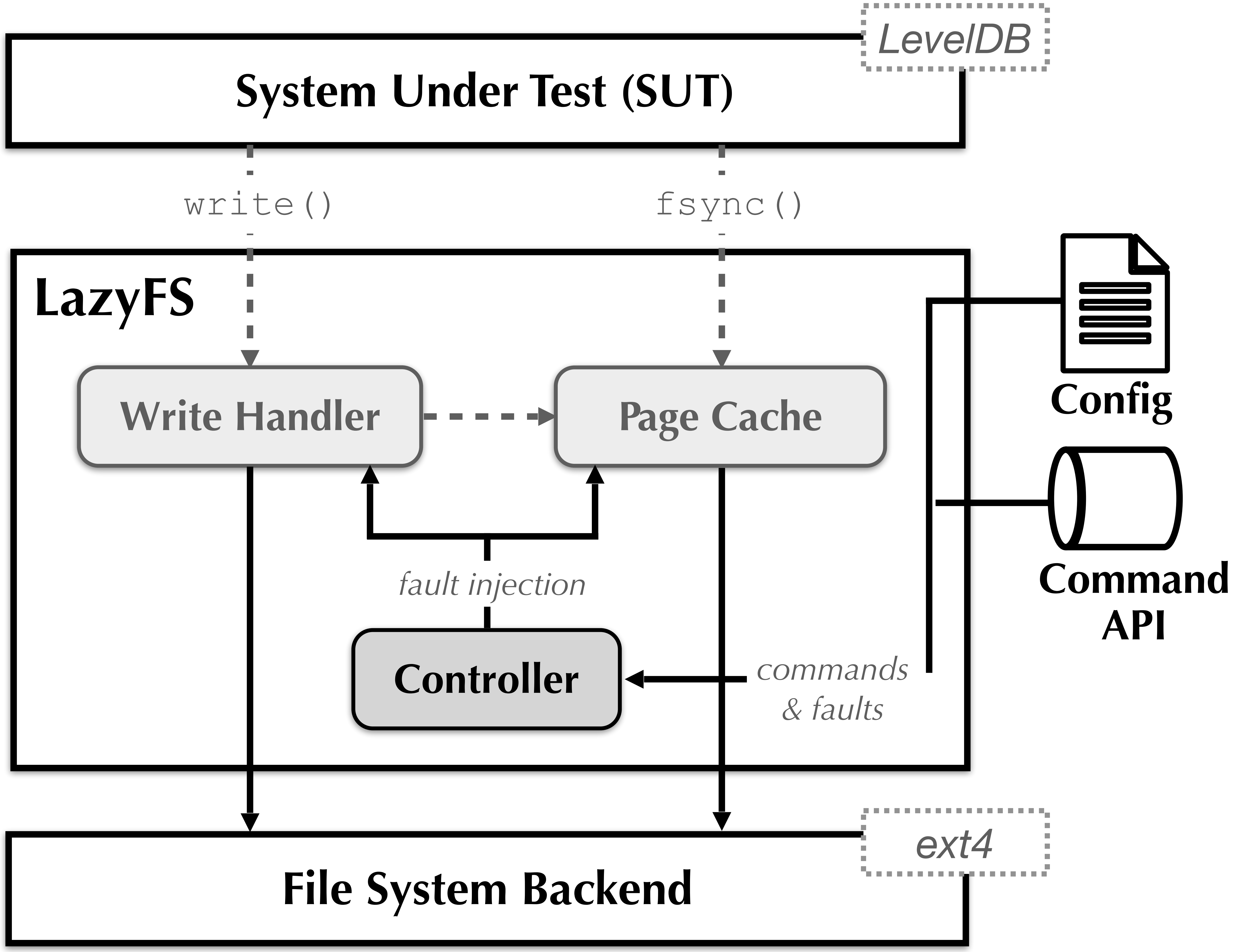
System overview



- ▢ *LazyFS component*
- ⋯→ *Control flow*
- → *I/O flow*

LazyFS

System overview



▢ *LazyFS component*

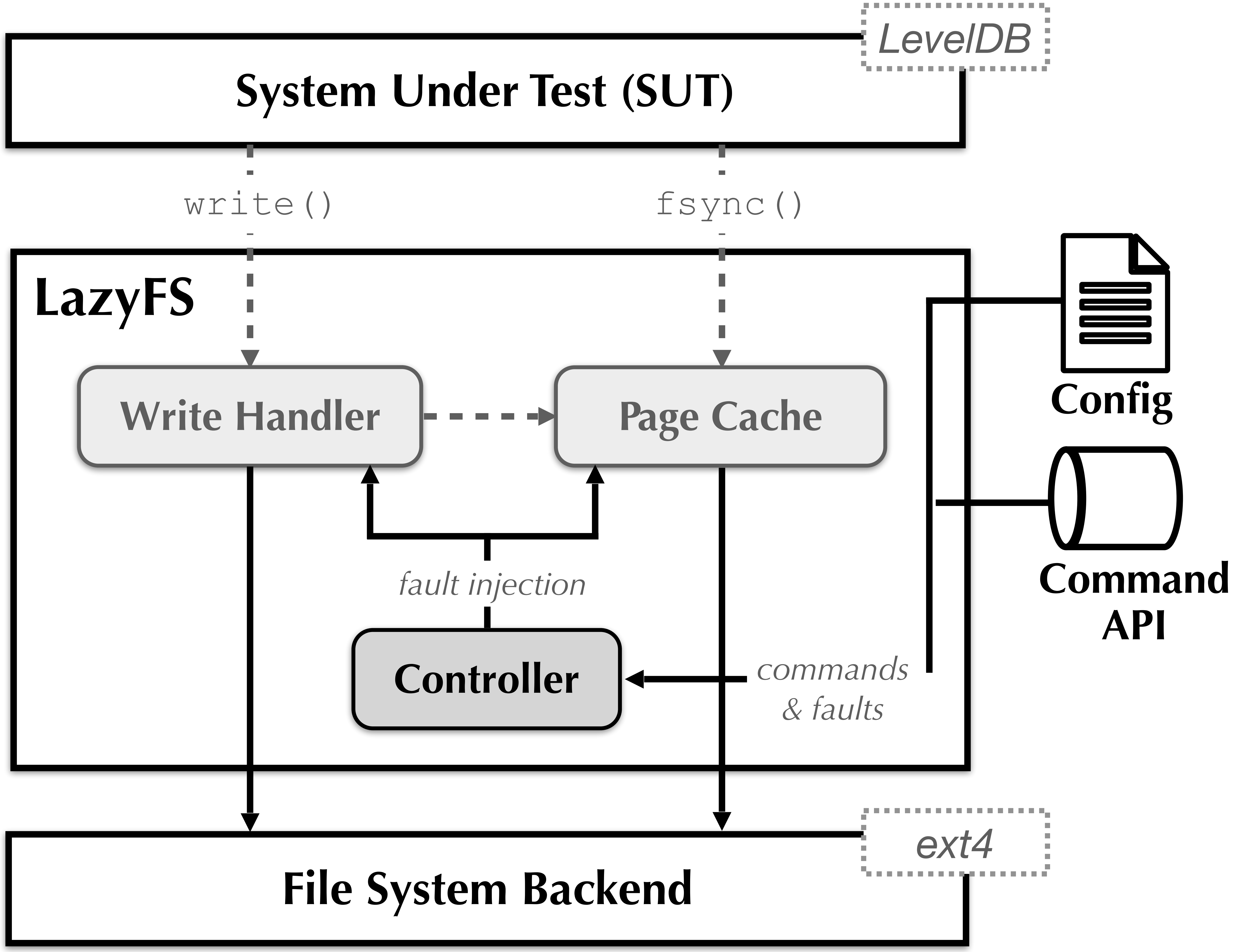
⋯⋯▸ *Control flow*

⟶ *I/O flow*

LazyFS

System overview

Example:
Inject a lost write fault after renaming file *wal*



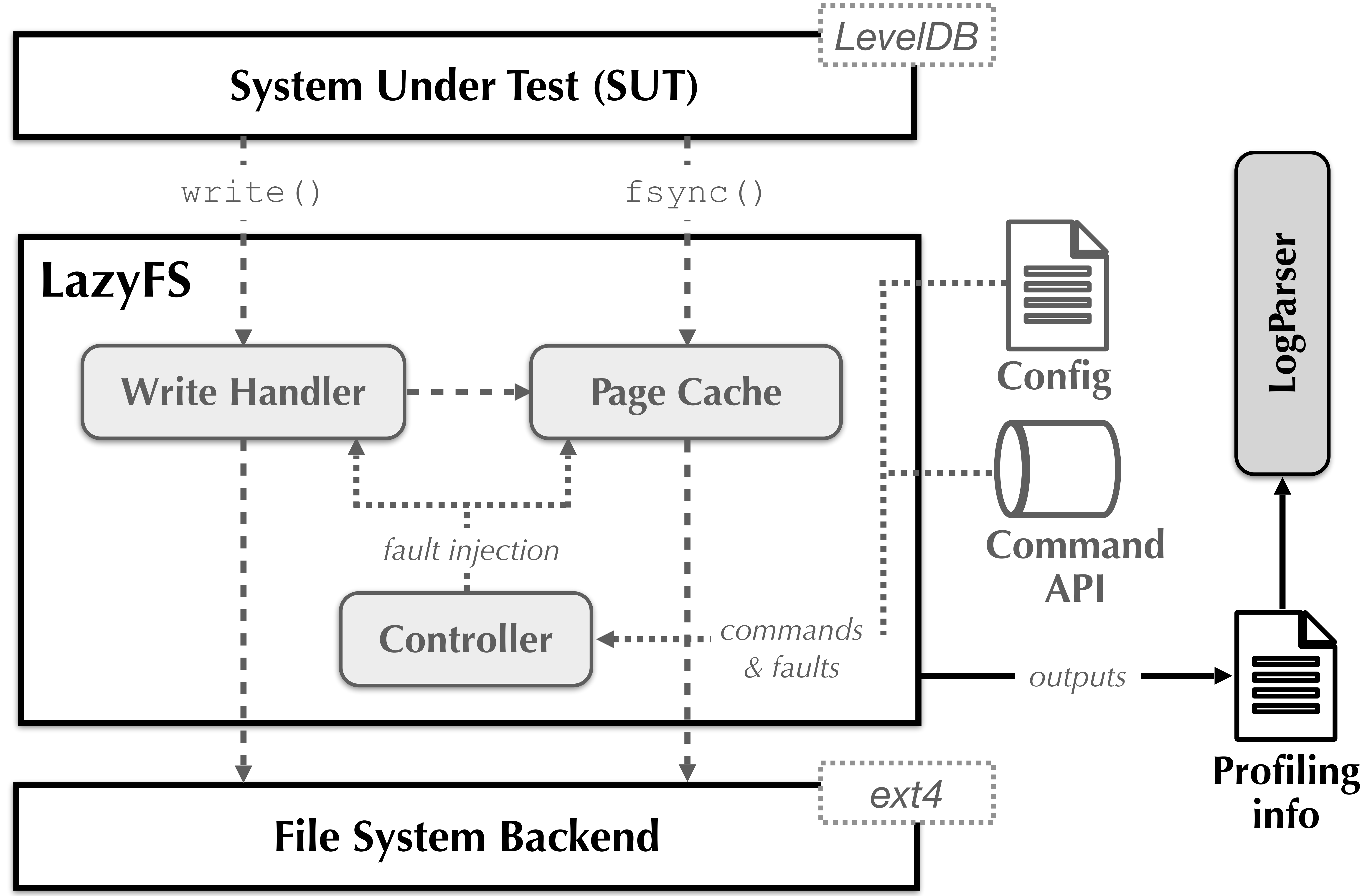
▢ *LazyFS component*

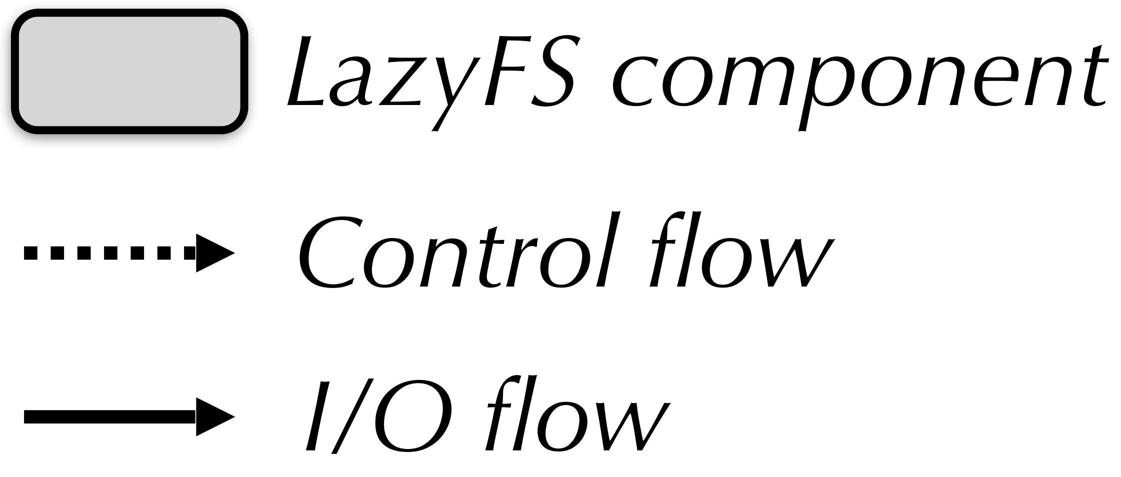
⇢ *Control flow*

→ *I/O flow*

LazyFS

System overview





Evaluation

Overview

	Number of bugs	Impact	Bugs in recent versions
PostgreSQL	1	🟧	1
LevelDB	4	🟦 🟧	2
ZooKeeper	3	🟦	3
Redis	1	🟪	0
Lightning N.	1	🟧	1
etcd	2	🟦	0
PebblesDB	3	🟦 🟧 🟪	3
etcd	5	🟦 🟩	4
Total	**20**		

🟦 13 Unavailability

🟧 4 Data corruption/loss

🟪 2 Silent data loss

🟩 1 Data inconsistency

Evaluation

Overview

		Number of bugs	Impact	Bugs in recent versions
Known	**PostgreSQL**	1	🟧	1
	LevelDB	4	🟦 🟧	2
	ZooKeeper	3	🟦	3
	Redis	1	🟪	0
	Lightning N.	1	🟧	1
	etcd	2	🟦	0
	PebblesDB	3	🟦 🟧 🟪	3
	etcd	5	🟦 🟩	4
	Total	**20**		

13 Unavailability

4 Data corruption/loss

2 Silent data loss

1 Data inconsistency

Evaluation

Overview

		Number of bugs	Impact	Bugs in recent versions
Known	PostgreSQL	1	🟧	1
	LevelDB	4	🟦 🟧	2
Ambiguous	ZooKeeper	3	🟦	3
	Redis	1	🟪	0
	Lightning N.	1	🟧	1
	etcd	2	🟦	0
	PebblesDB	3	🟦 🟧 🟪	3
	etcd	5	🟦 🟩	4
	Total	20		

🟦 13	Unavailability
🟧 4	Data corruption/loss
🟪 2	Silent data loss
🟩 1	Data inconsistency

Evaluation

Overview

		Number of bugs	Impact	Bugs in recent versions
Known	PostgreSQL	1	🟧	1
	LevelDB	4	🟦 🟧	2
Ambiguous	ZooKeeper	3	🟦	3
	Redis	1	🟪	0
	Lightning N.	1	🟧	1
	etcd	2	🟦	0
New	PebblesDB	3	🟦 🟧 🟪	3
	etcd	5	🟦 🟩	4
	Total	**20**		

- 🟦 13 Unavailability
- 🟧 4 Data corruption/loss
- 🟪 2 Silent data loss
- 🟩 1 Data inconsistency

Evaluation

Overview

		Number of bugs	Impact	Bugs in recent versions
Known	PostgreSQL	1	🟧	1
	LevelDB	4	🟦 🟧	2
Ambiguous	ZooKeeper	3	🟦	3
	Redis	1	🟪	0
	Lightning N.	1	🟧	1
	etcd	2	🟦	0
New	PebblesDB	3	🟦 🟧 🟪	3
	etcd	5	🟦 🟩	4
	Total	**20**		

🟦 13	Unavailability
🟧 4	Data corruption/loss
🟪 2	Silent data loss
🟩 1	Data inconsistency

Evaluation

Overview

		Number of bugs	Impact	Bugs in recent versions
Known	**PostgreSQL**	1	🟧	1
	LevelDB	4	🟦 🟧	2
Ambiguous	**ZooKeeper**	3	🟦	3
	Redis	1	🟪	0
	Lightning N.	1	🟧	1
	etcd	2	🟦	0
New	**PebblesDB**	3	🟦 🟧 🟪	3
	etcd	5	🟦 🟩	4
	Total	**20**		

+2 crash consistency mechanisms

🟦 13 Unavailability

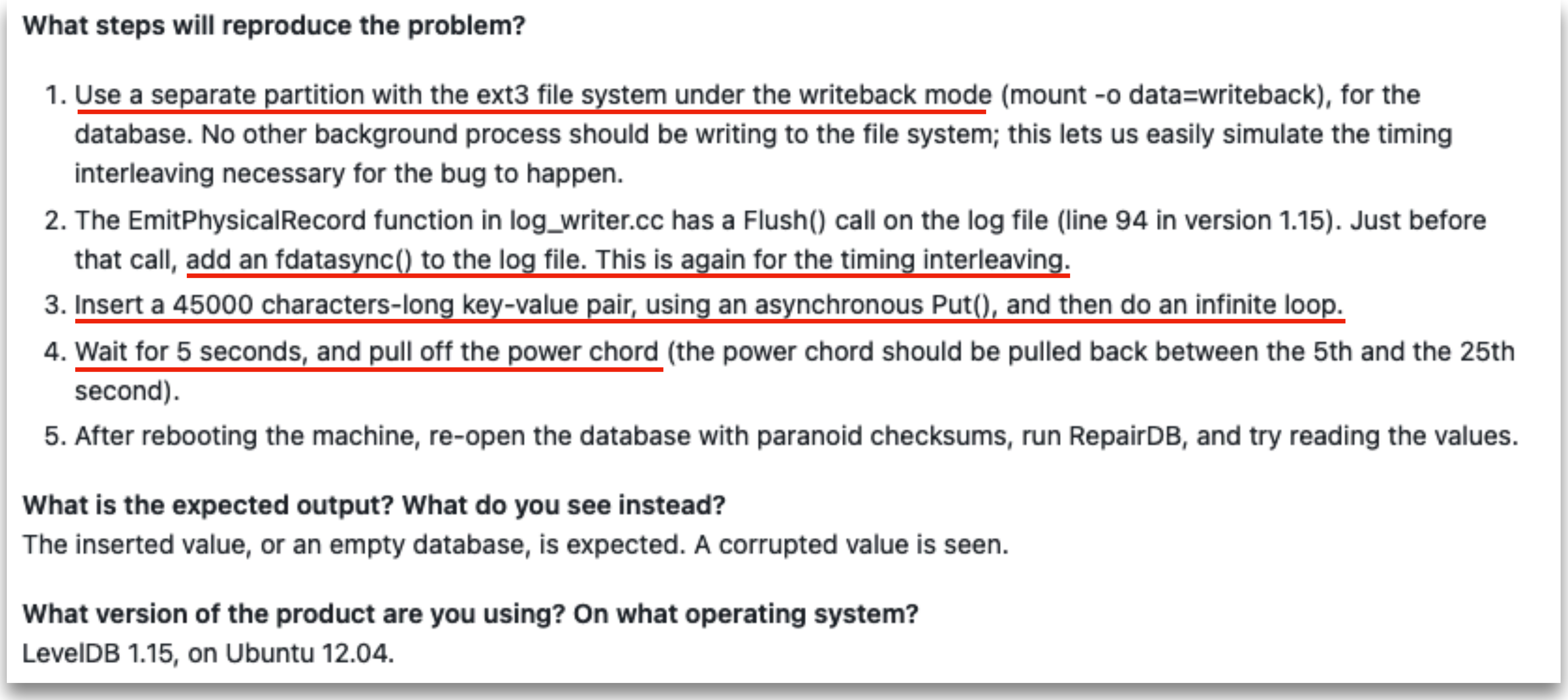
🟧 4 Data corruption/loss

🟪 2 Silent data loss

🟩 1 Data inconsistency

Known bug

LevelDB *Bug #6*



What steps will reproduce the problem?

1. Use a separate partition with the ext3 file system under the writeback mode (mount -o data=writeback), for the database. No other background process should be writing to the file system; this lets us easily simulate the timing interleaving necessary for the bug to happen.
2. The EmitPhysicalRecord function in log_writer.cc has a Flush() call on the log file (line 94 in version 1.15). Just before that call, add an fdatasync() to the log file. This is again for the timing interleaving.
3. Insert a 45000 characters-long key-value pair, using an asynchronous Put(), and then do an infinite loop.
4. Wait for 5 seconds, and pull off the power chord (the power chord should be pulled back between the 5th and the 25th second).
5. After rebooting the machine, re-open the database with paranoid checksums, run RepairDB, and try reading the values.

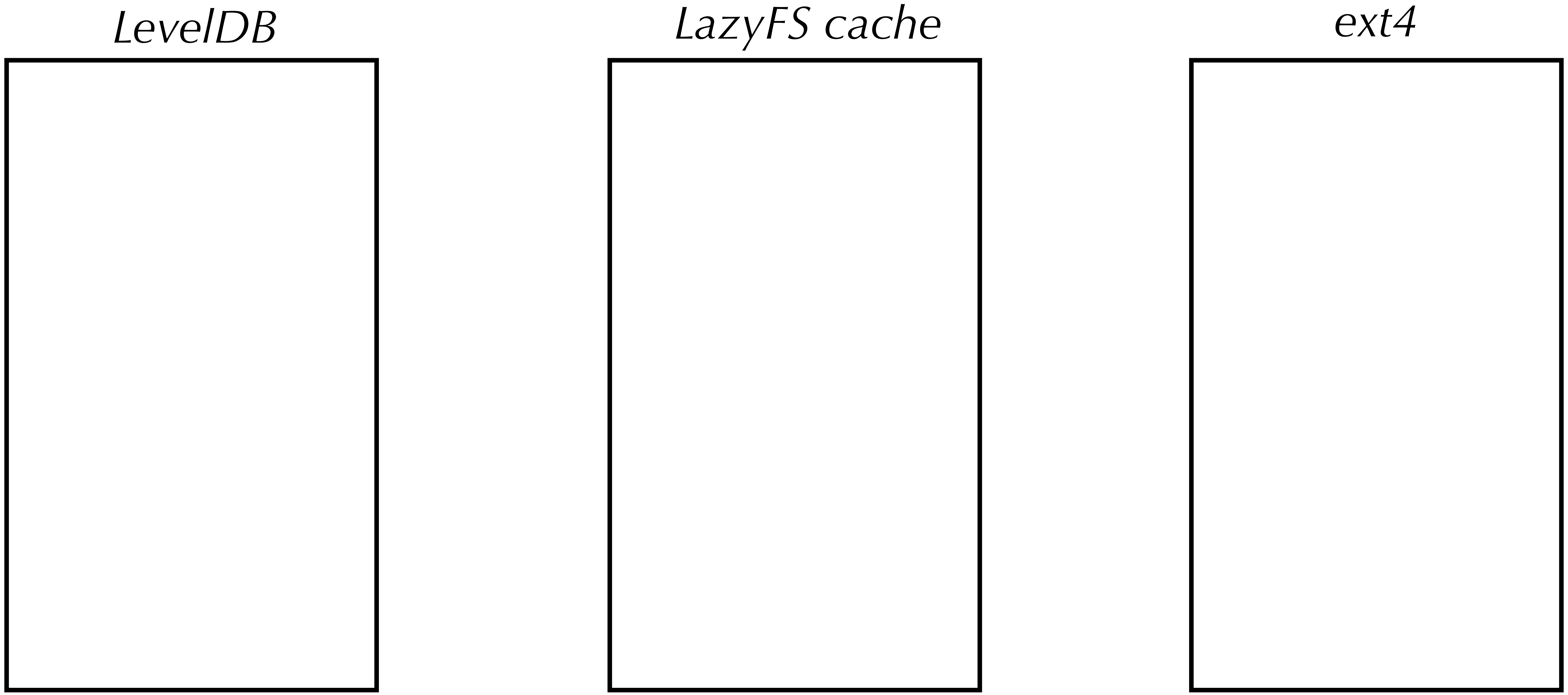
What is the expected output? What do you see instead?
The inserted value, or an empty database, is expected. A corrupted value is seen.

What version of the product are you using? On what operating system?
LevelDB 1.15, on Ubuntu 12.04.

1. Use ext3 file system in writeback mode in a separate partition.
2. Add `fsync()` in function of source code.
3. Insert a 45000 characters-long key-value pair and do an infinite loop.
4. Wait 5 seconds and pull off the power chord.

Known bug

LevelDB *Bug #6*



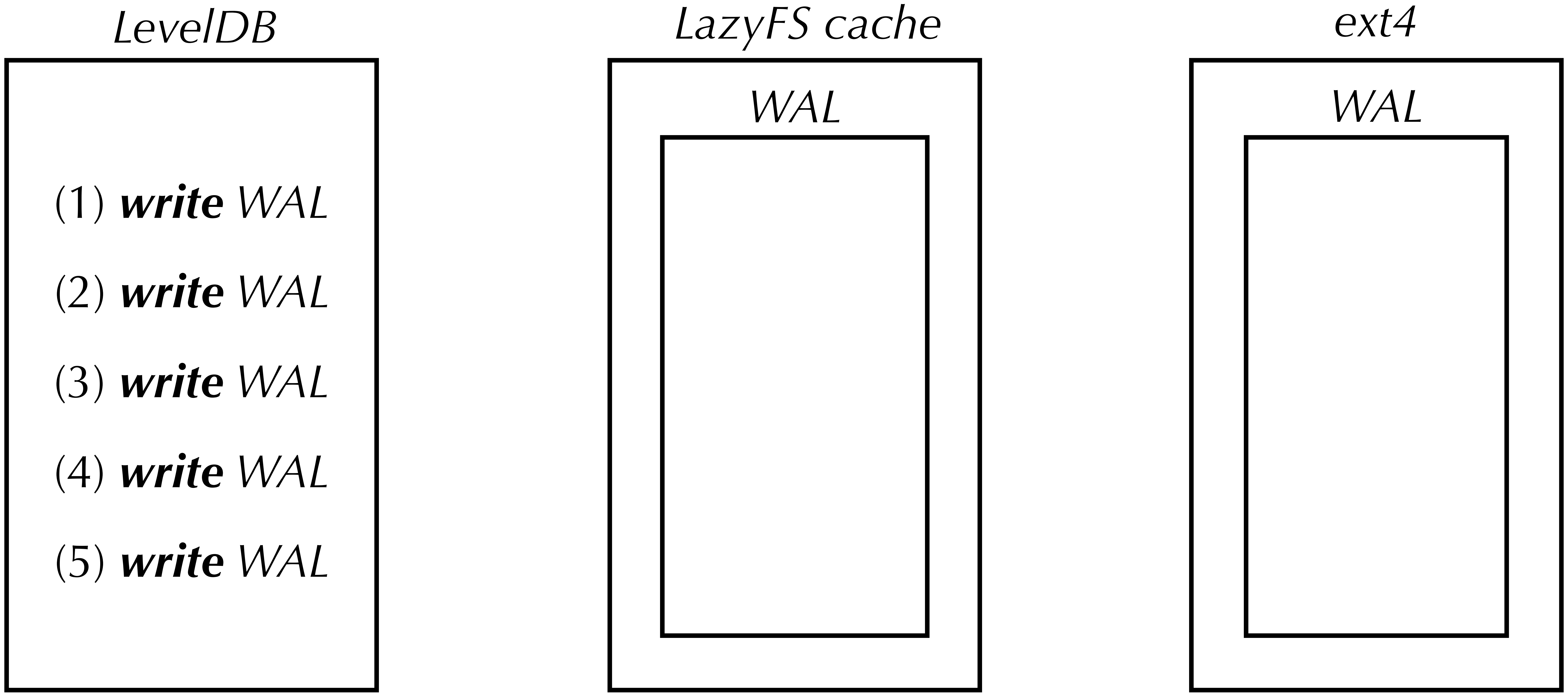
LevelDB

LazyFS cache

ext4

Known bug

LevelDB *Bug #6*



LevelDB

(1) ***write*** *WAL*

(2) ***write*** *WAL*

(3) ***write*** *WAL*

(4) ***write*** *WAL*

(5) ***write*** *WAL*

LazyFS cache

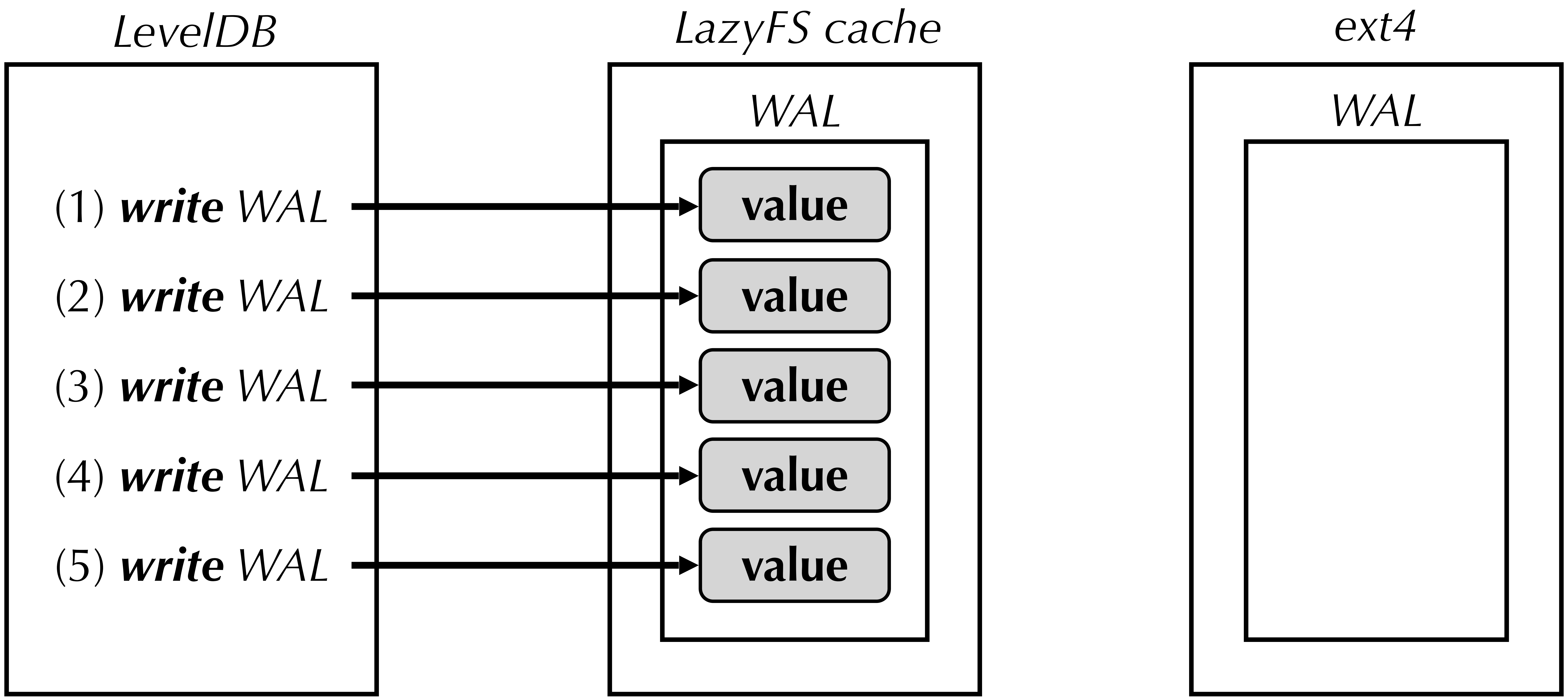
WAL

ext4

WAL

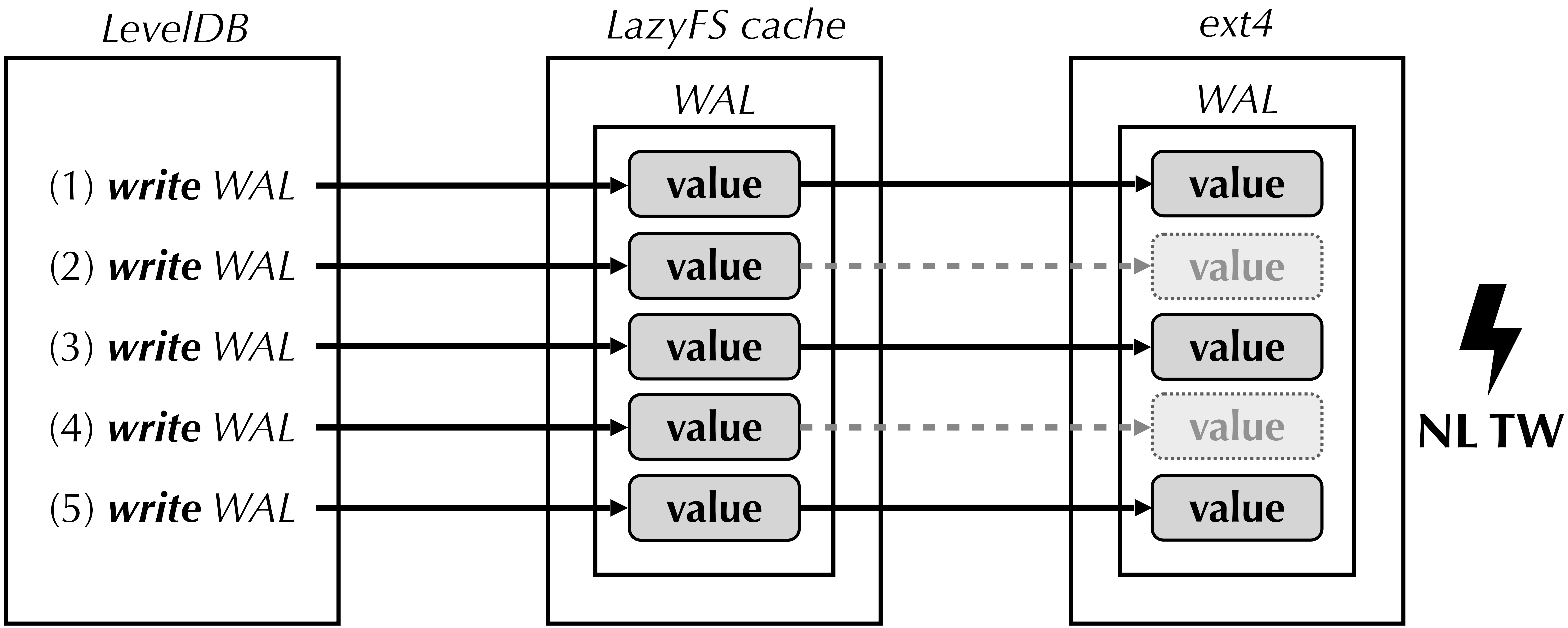
Known bug

LevelDB *Bug #6*



Known bug

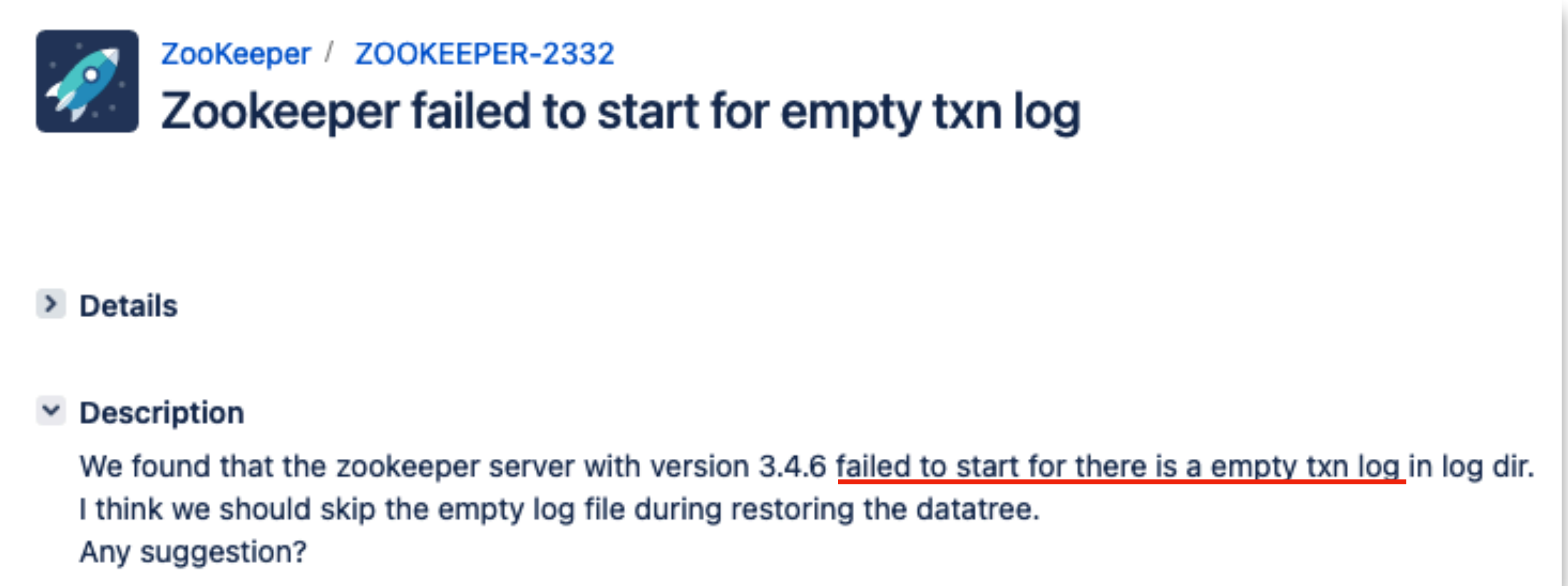
LevelDB *Bug #6*



NL TW→Non-Linear Torn Write

Ambiguous bugs

ZooKeeper *Bug #7*



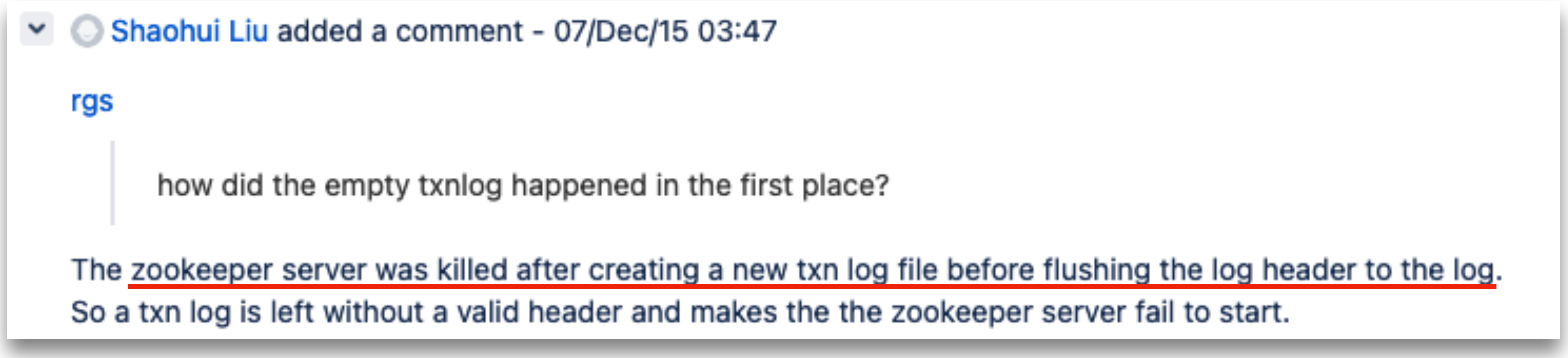
ZooKeeper / ZOOKEEPER-2332

Zookeeper failed to start for empty txn log

Details

Description

We found that the zookeeper server with version 3.4.6 failed to start for there is a empty txn log in log dir.
I think we should skip the empty log file during restoring the datatree.
Any suggestion?



Shaohui Liu added a comment - 07/Dec/15 03:47

rgs

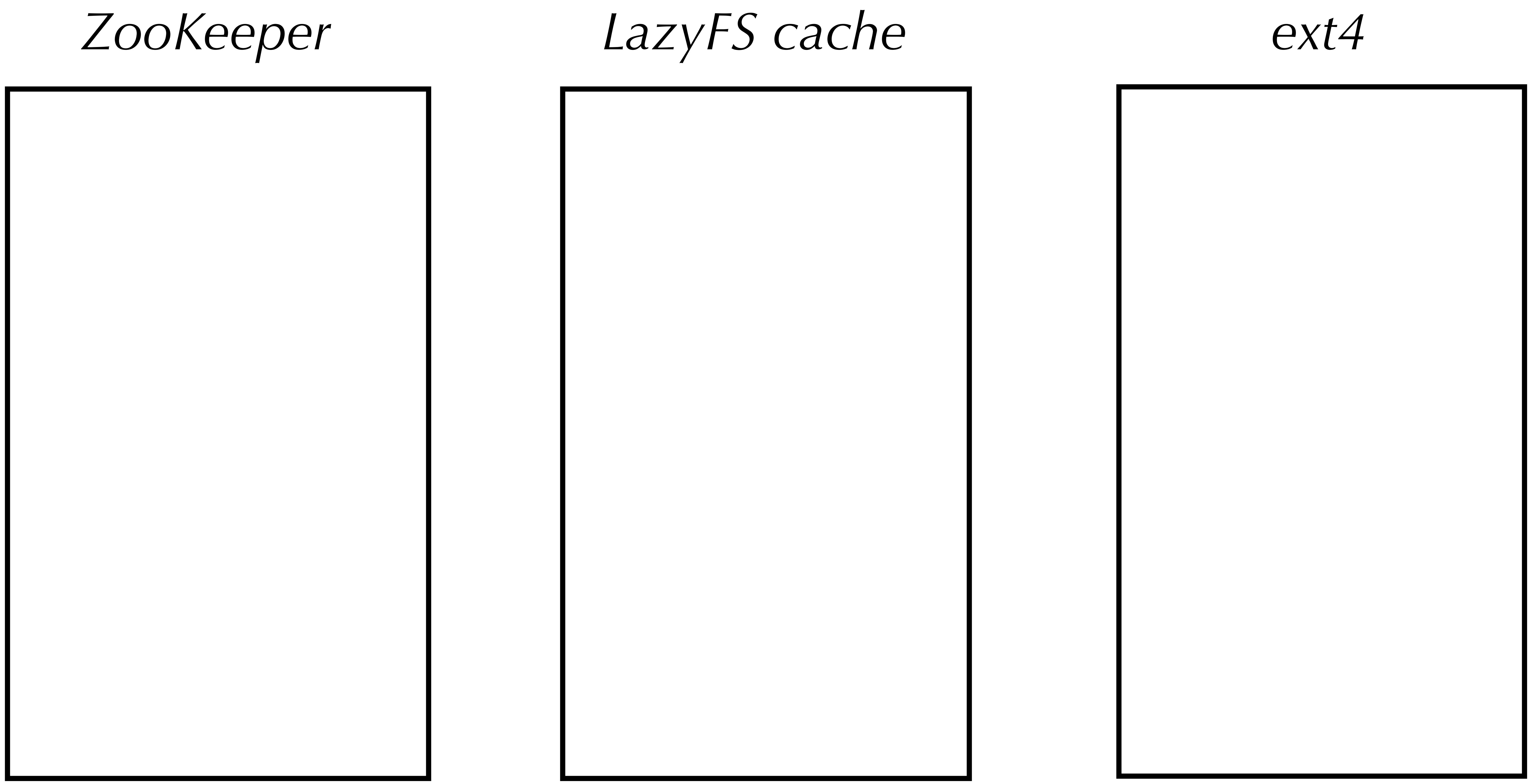
> how did the empty txnlog happened in the first place?

The zookeeper server was killed after creating a new txn log file before flushing the log header to the log.
So a txn log is left without a valid header and makes the the zookeeper server fail to start.

- Fails to start with empty log file.
- ZooKeeper server killed after creating log file but before flushing log header.

Ambiguous bugs

ZooKeeper *Bug #7*



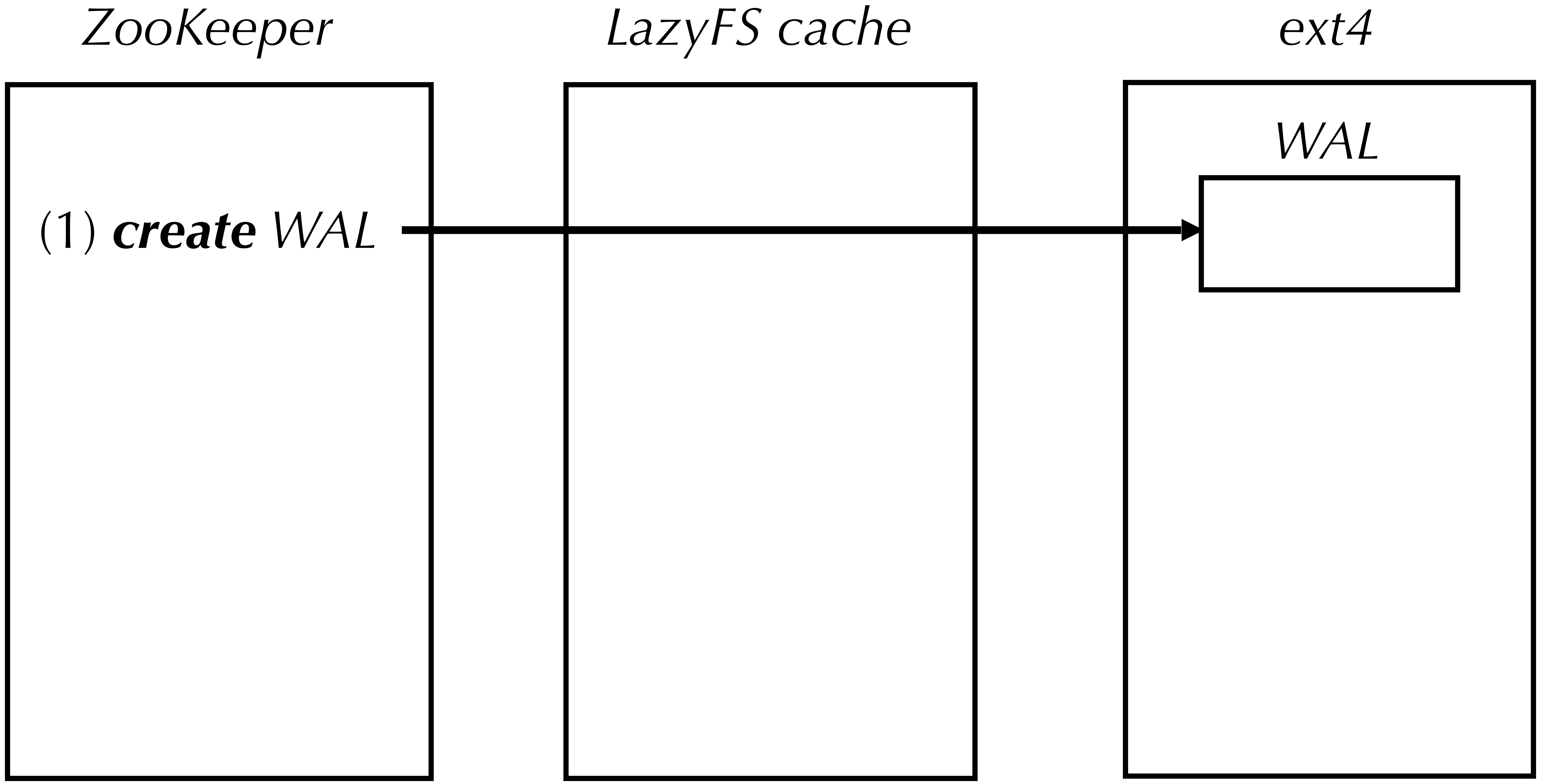
ZooKeeper

LazyFS cache

ext4

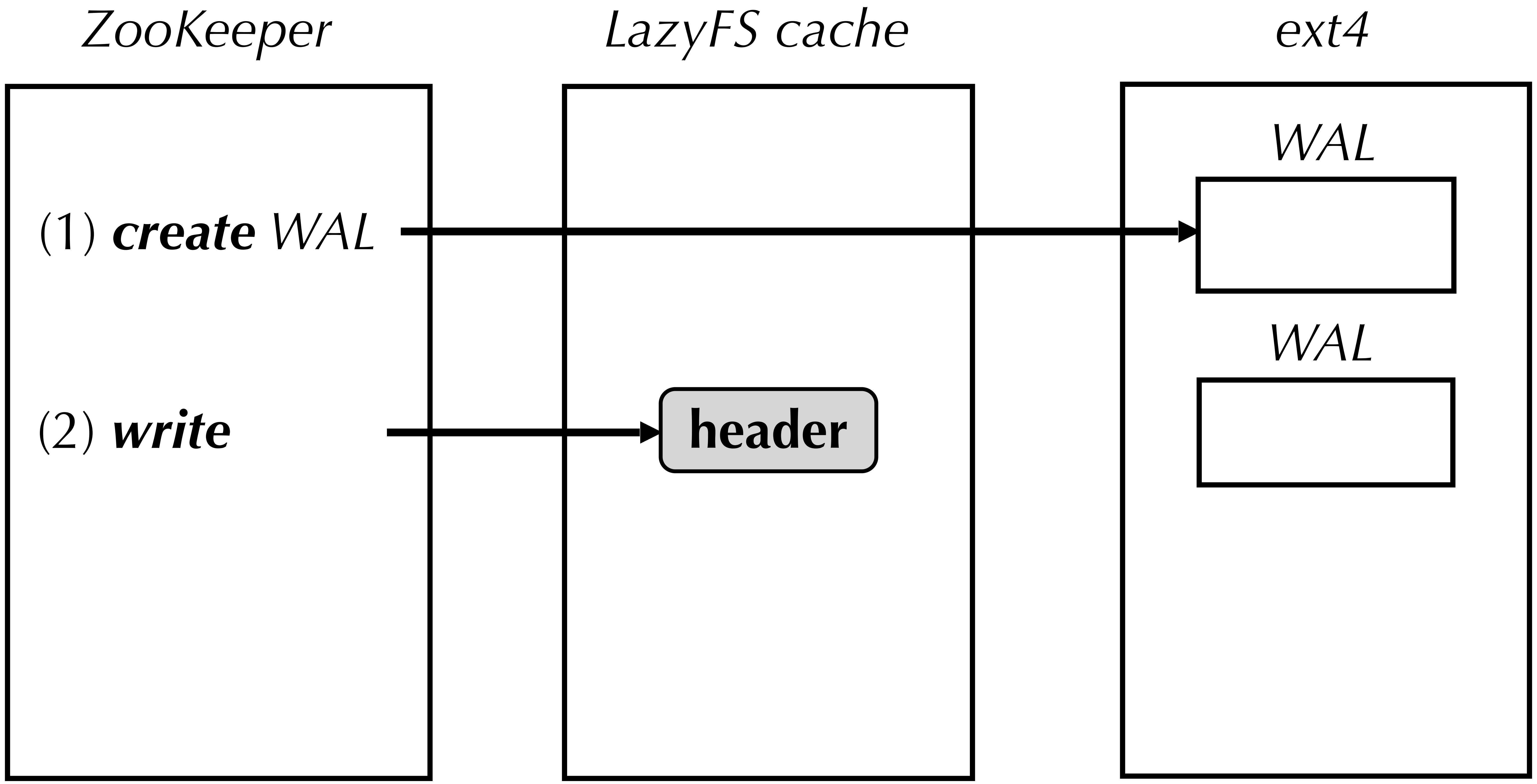
Ambiguous bugs

ZooKeeper *Bug #7*



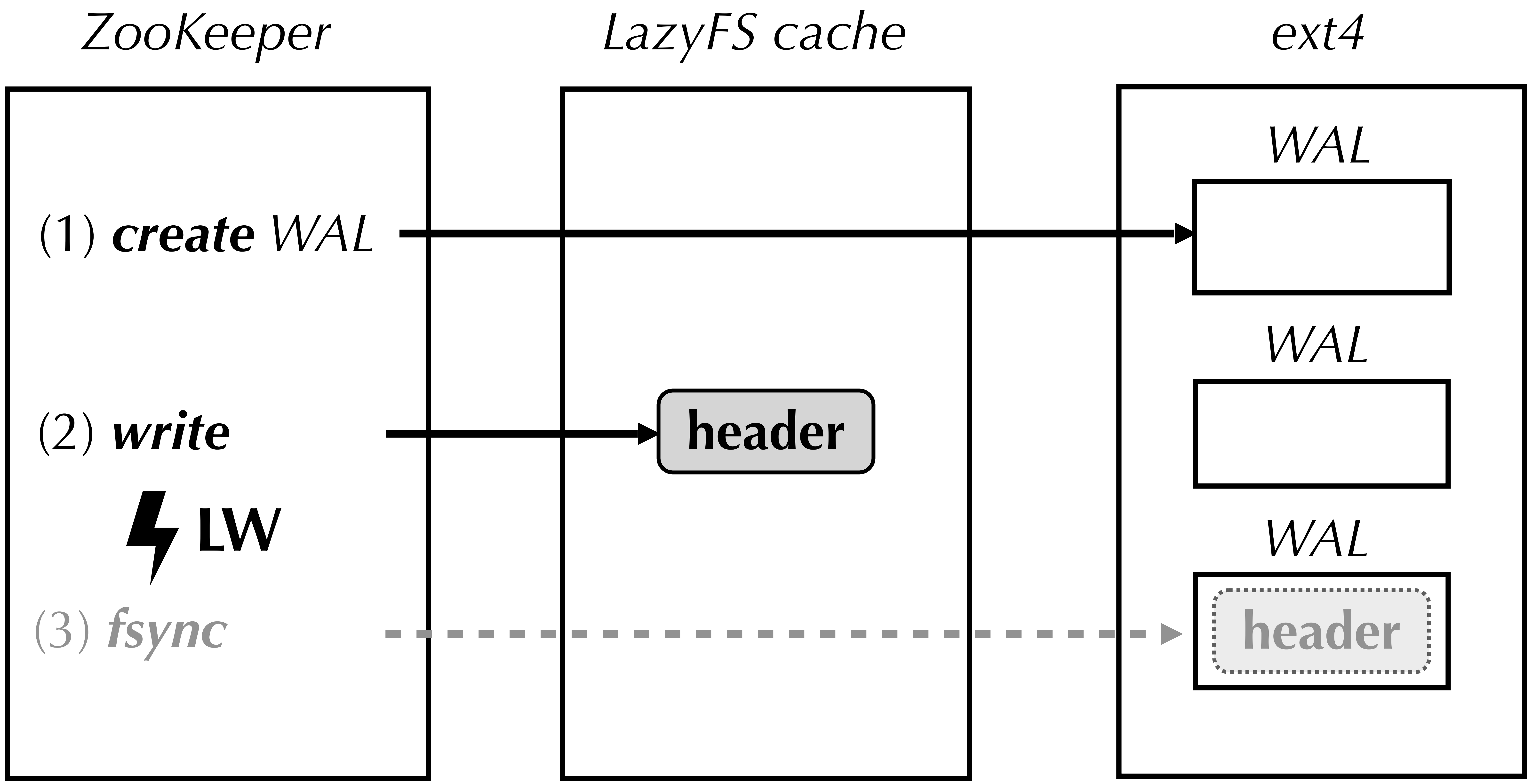
Ambiguous bugs

ZooKeeper *Bug #7*



Ambiguous bugs

ZooKeeper *Bug #7*



LW→Lost Write

New bugs

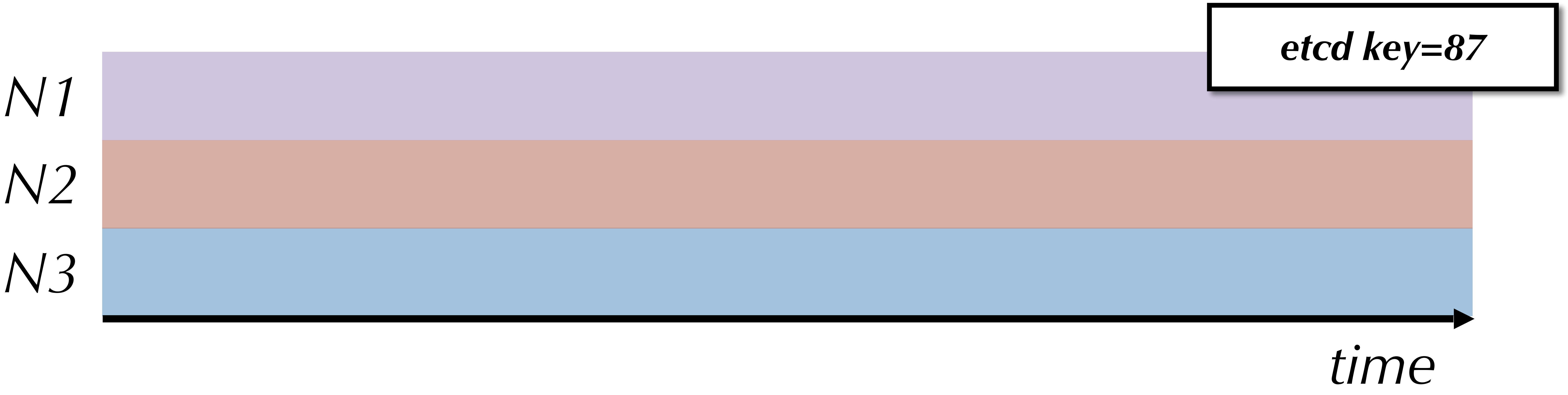
etcd *Bug* #20

- **Jepsen** is a framework to test the reliability of distributed systems.
- Integration with LazyFS finds a **split-brain** scenario in etcd.

New bugs

etcd *Bug* #20

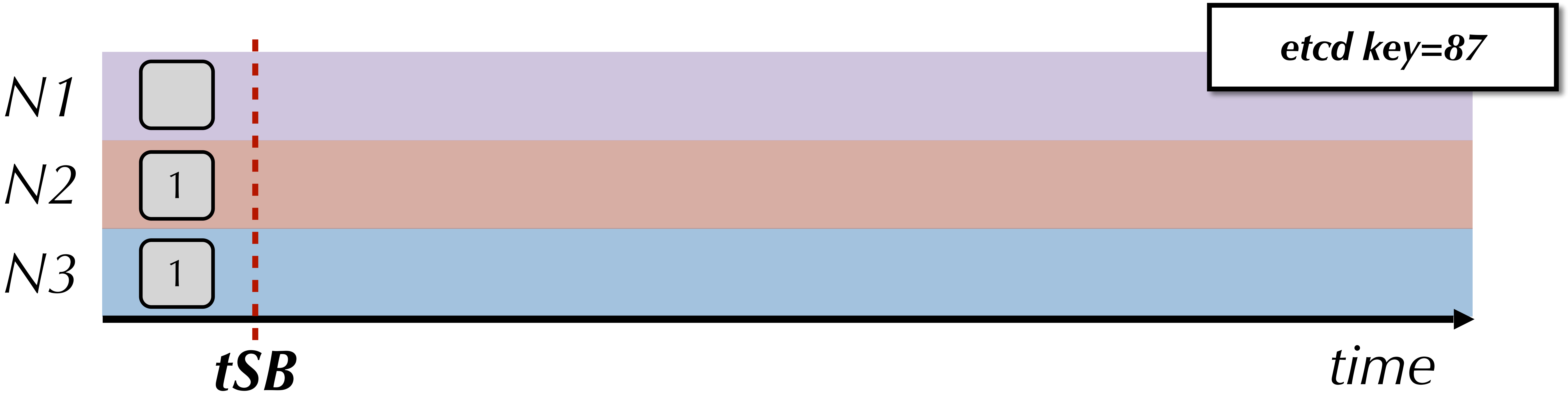
- **Jepsen** is a framework to test the reliability of distributed systems.
- Integration with LazyFS finds a **split-brain** scenario in etcd.



New bugs

etcd *Bug #20*

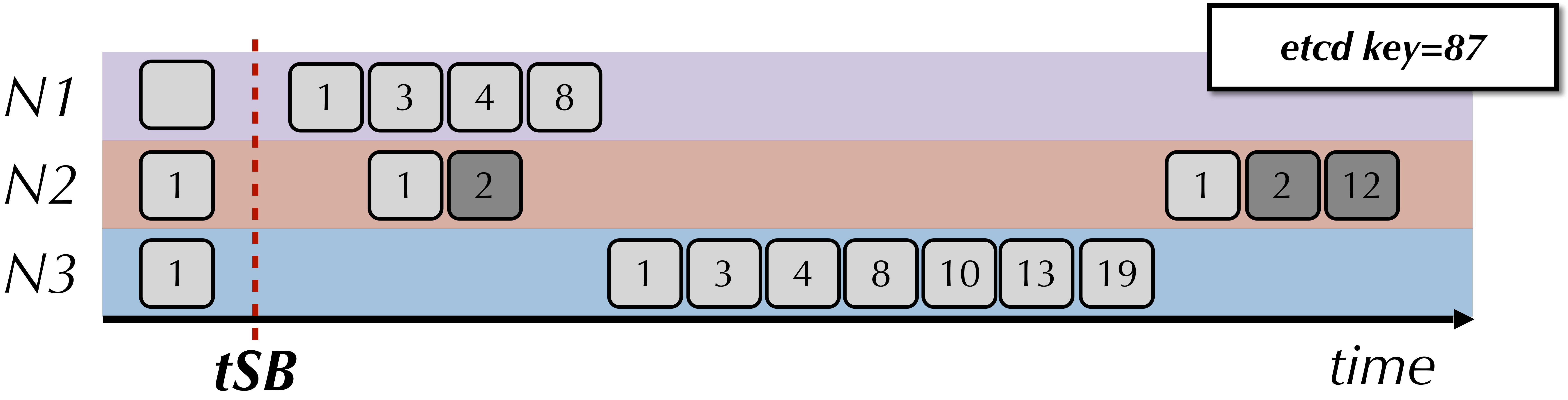
- **Jepsen** is a framework to test the reliability of distributed systems.
- Integration with LazyFS finds a **split-brain** scenario in etcd.



New bugs

etcd *Bug #20*

- **Jepsen** is a framework to test the reliability of distributed systems.
- Integration with LazyFS finds a **split-brain** scenario in etcd.



Conclusion

- Widely used systems are **still affected** by crash consistency bugs.
- LazyFS provides a way to **reproduce bugs caused by lost** and **torn writes.**
- LazyFS helps to **understand** the root cause of bugs.
- LazyFS helps to **validate** crash consistency mechanisms.

Known systems that used LazyFS:

◆ **PostgreSQL** ◆ **etcd** ◆ **MongoDB**

When Amnesia Strikes: Understanding and Reproducing Data Loss Bugs with Fault Injection

dsrhaslab/lazyfs

maria.j.ramos@inesctec.pt