Shadow Filesystems: Recovering from Filesystem Runtime Errors via Robust Alternative Execution

Jing Liu, Xiangpeng Hao

Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau, Tej Chajed

University of Wisconsin–Madison

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

Existing Filesystems:
Excels at Performance OR Correctness

Performance	Correctness
Caches, concurrency, parallelism, etc
Kernel filesystems (e.g., ext4, btrfs)
DevFS (FAST ’18)
LineFS (SOSP ’21)
uFS (SOSP ’21)

Existing Filesystems: Excels at Performance OR Correctness

Performance	Correctness
Caches, concurrency, parallelism, etc	Formally verified implementation
Kernel filesystems (e.g., ext4, btrfs)	FSCQ (SOSP '15)
DevFS (FAST '18)	Cogent (ASPLOS '16)
LineFS (SOSP '21)	Yggdrasil (OSDI '16)
uFS (SOSP '21)	AtomFS (SOSP '19)
	DaisyNFS (OSDI '22)
Correctness is difficult	**Performance is difficult**

Can we build a file system that has both high performance AND correctness

Can we build a file system
that has both
high performance AND correctness

use two filesystems

Idea: Two Filesystems to Achieve Both

Performance	Correctness
Caches, concurrency, parallelism, etc	Formally verified implementation
Kernel filesystems (e.g., ext4, btrfs)	FSCQ (SOSP '15)
DevFS (FAST '18)	Cogent (ASPLOS '16)
LineFS (SOSP '21)	Yggdrasil (OSDI '16)
uFS (SOSP '21)	AtomFS (SOSP '19)
	DaisyNFS (OSDI '22)
	uFS-Shadow

uFS (SOSP '21) → Performance AND Correctness ← uFS-Shadow

Idea: Two Filesystems to Achieve Both

Performance	Correctness
Caches, concurrency, parallelism, etc	Formally verified implementation
Kernel filesystems (e.g., ext4, btrfs)	FSCQ (SOSP '15)
DevFS (FAST '18)	Cogent (ASPLOS '16)
LineFS (SOSP '21)	Yggdrasil (OSDI '16)
uFS (SOSP '21)	AtomFS (SOSP '19)
	DaisyNFS (OSDI '22)
	uFS-Shadow

uFS → Performance AND Correctness ← uFS-Shadow

Robust Alternative Execution

RAE: Robust Alternative Execution

Two filesystems

- A base filesystem (common path)
  - High performance
- A shadow filesystem (alternative path)
  - Correctness
  - Handles the workload that triggers a bug in the base

RAE: Robust Alternative Execution

Two filesystems

- A base filesystem (common path)
  - High performance
- A shadow filesystem (alternative path)
  - Correctness
  - Handles the workload that triggers a bug in the base

Can even survive deterministic bugs in the base

Outline

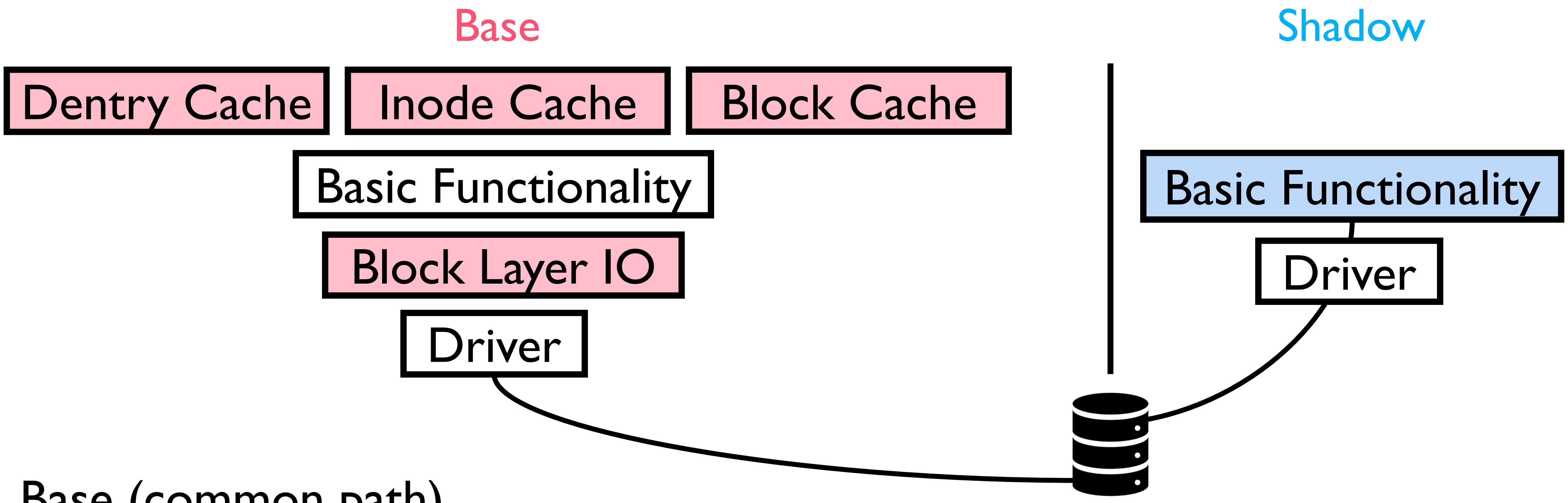
Introduction

Robust Alternative Execution (RAE)

Prototype and Progress Status

Future Challenges

RAE: The Base and Shadow Filesystems



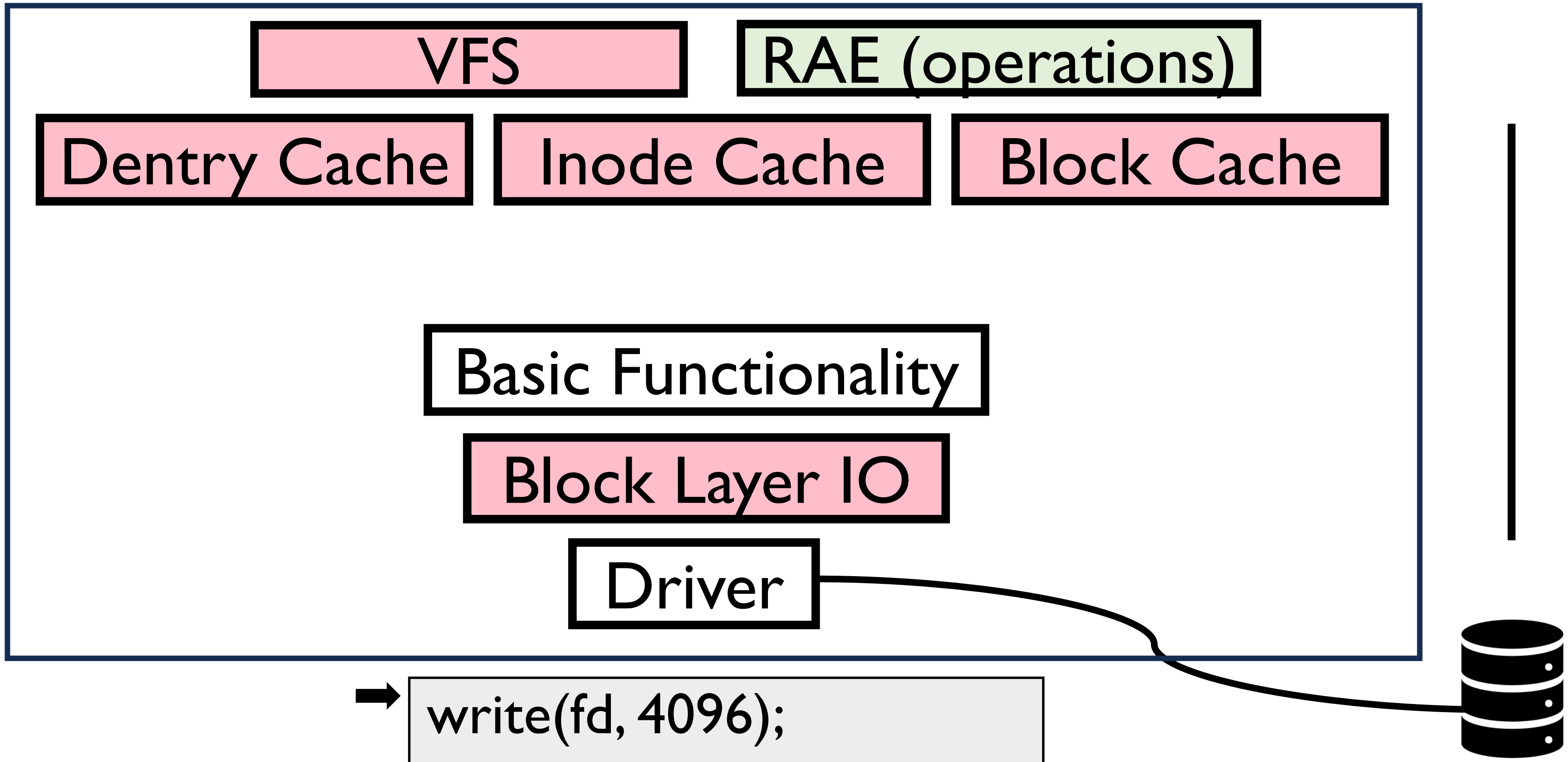
Base (common path)

- An existing filesystem optimized for performance

Shadow (alternative path)

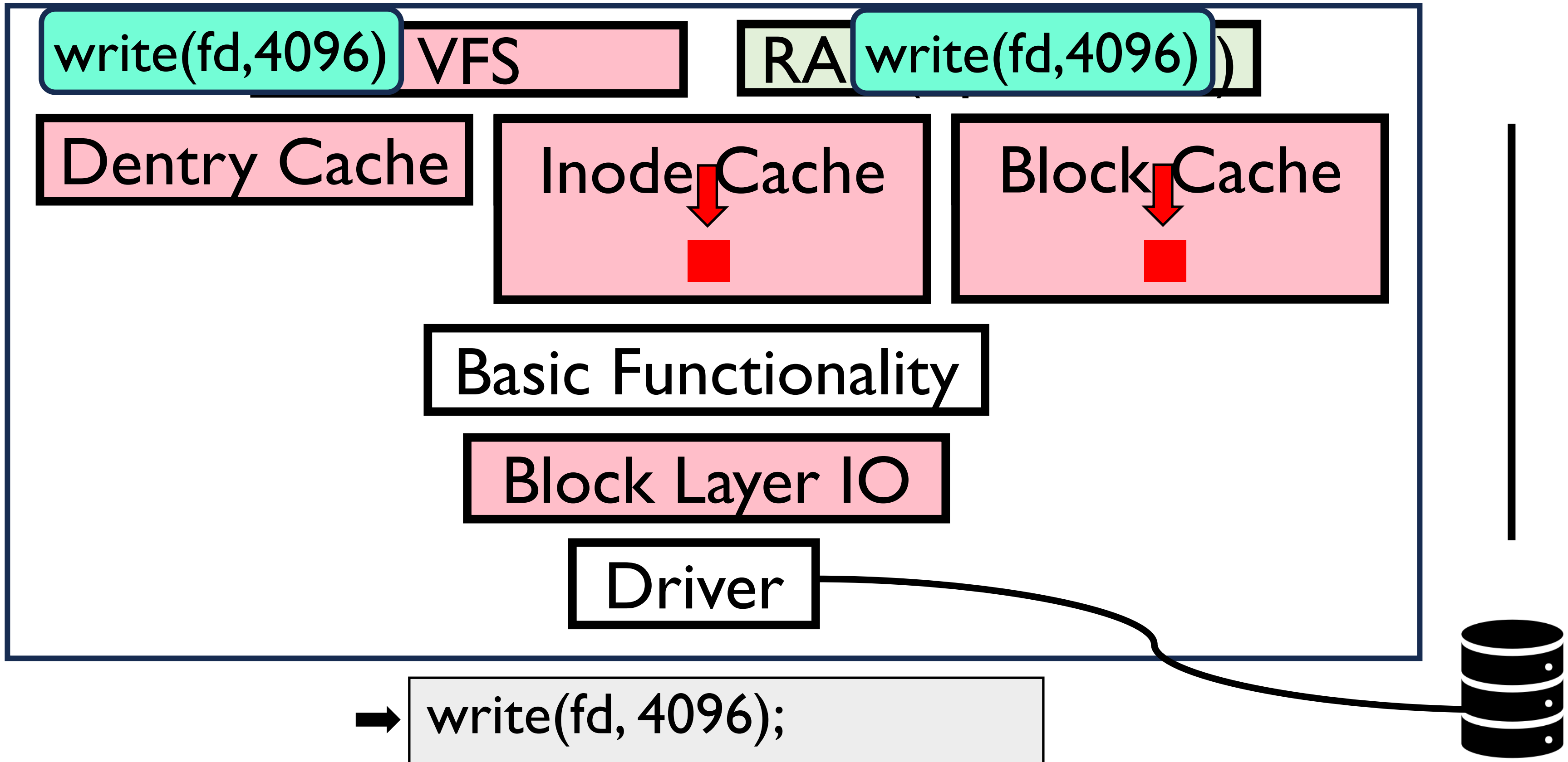
- A shadow filesystem that aims to be "bug-free"

Base Executes the Workload



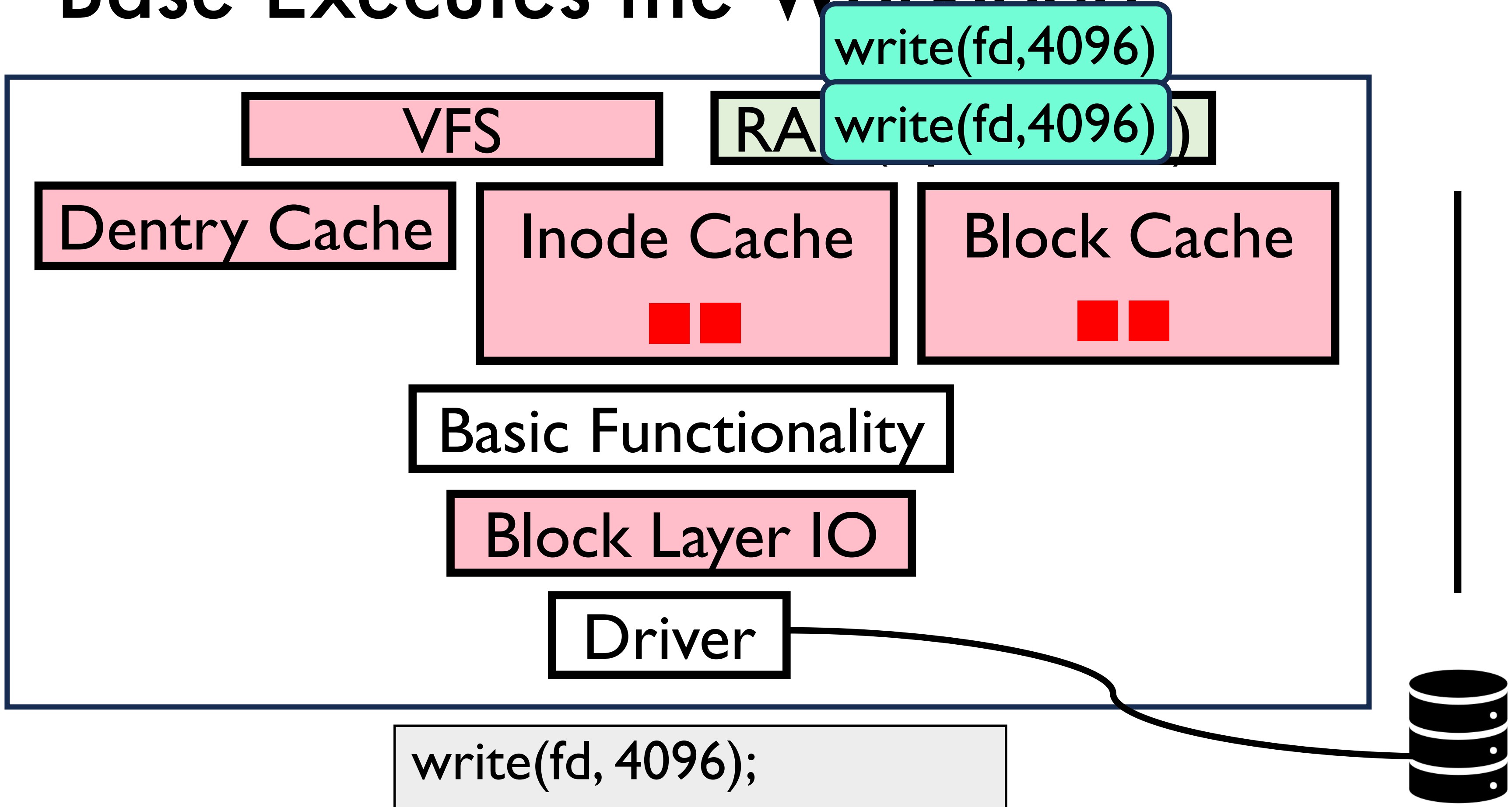

```
write(fd, 4096);
write(fd, 4096);
pread(fd, offset=0);
close(fd);
```

Base Executes the Workload



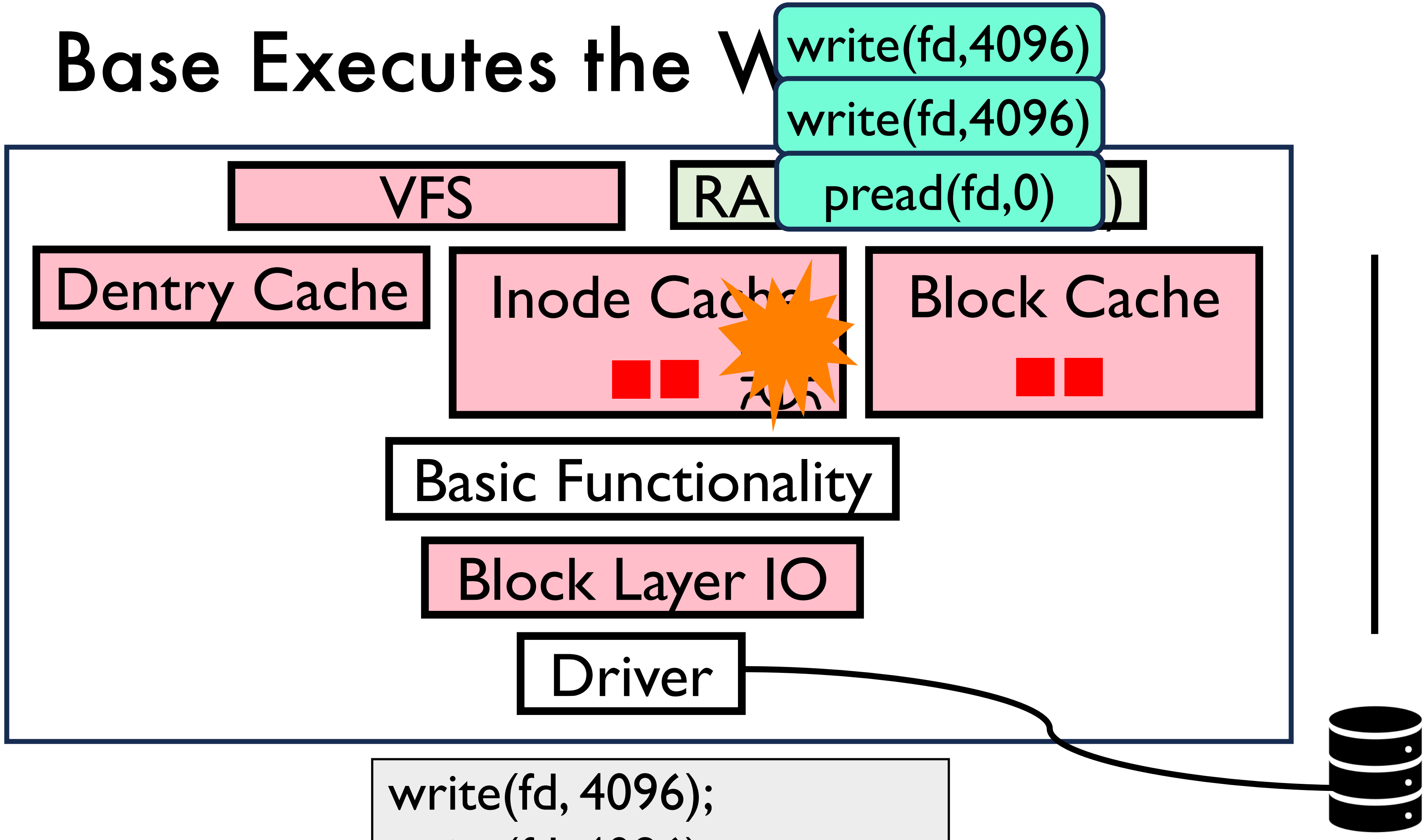

```
write(fd, 4096);
write(fd, 4096);
pread(fd, offset=0);
close(fd);
```

Base Executes the Workload



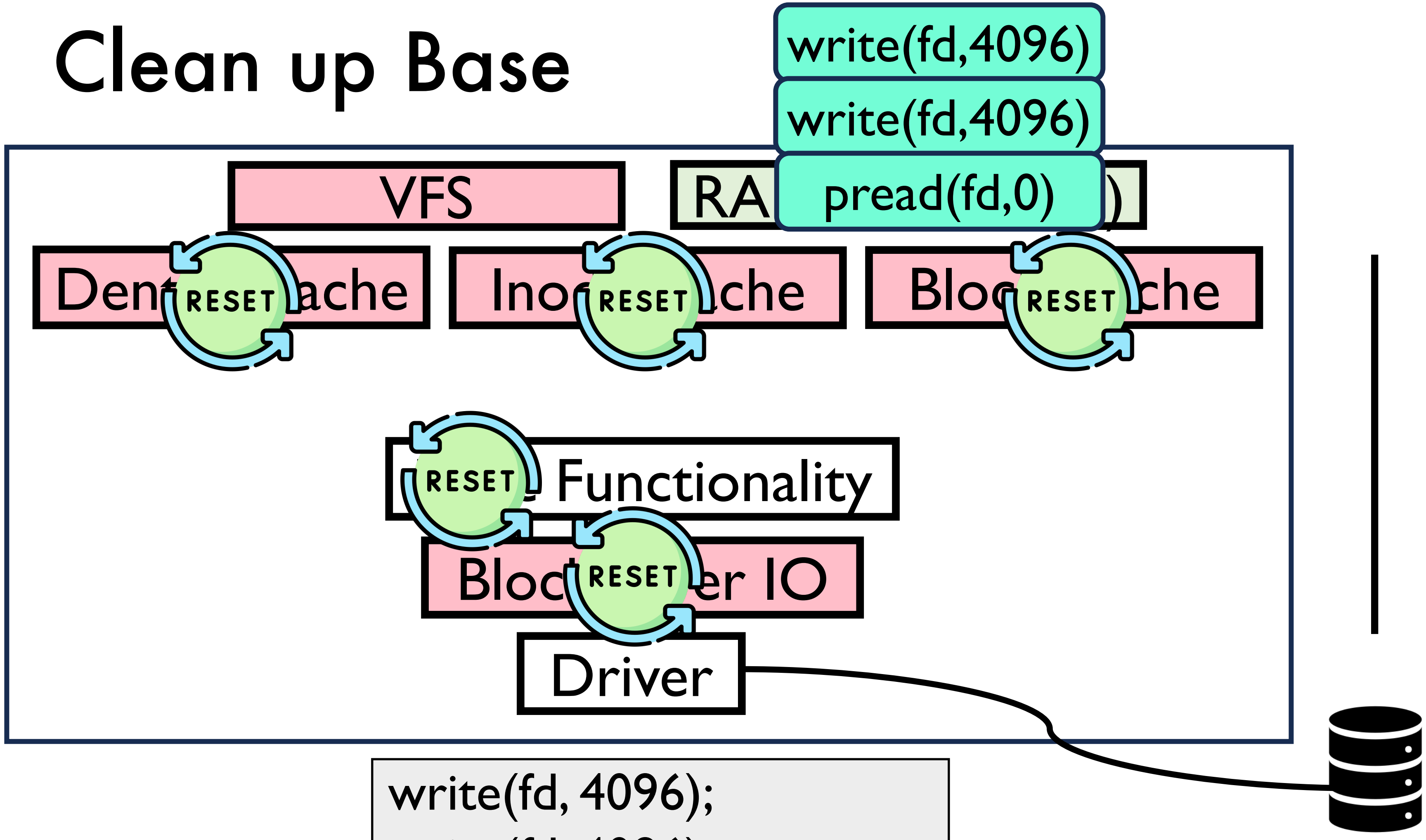

```
write(fd, 4096);
write(fd, 4096);
pread(fd, offset=0);
close(fd);
```

Base Executes the W



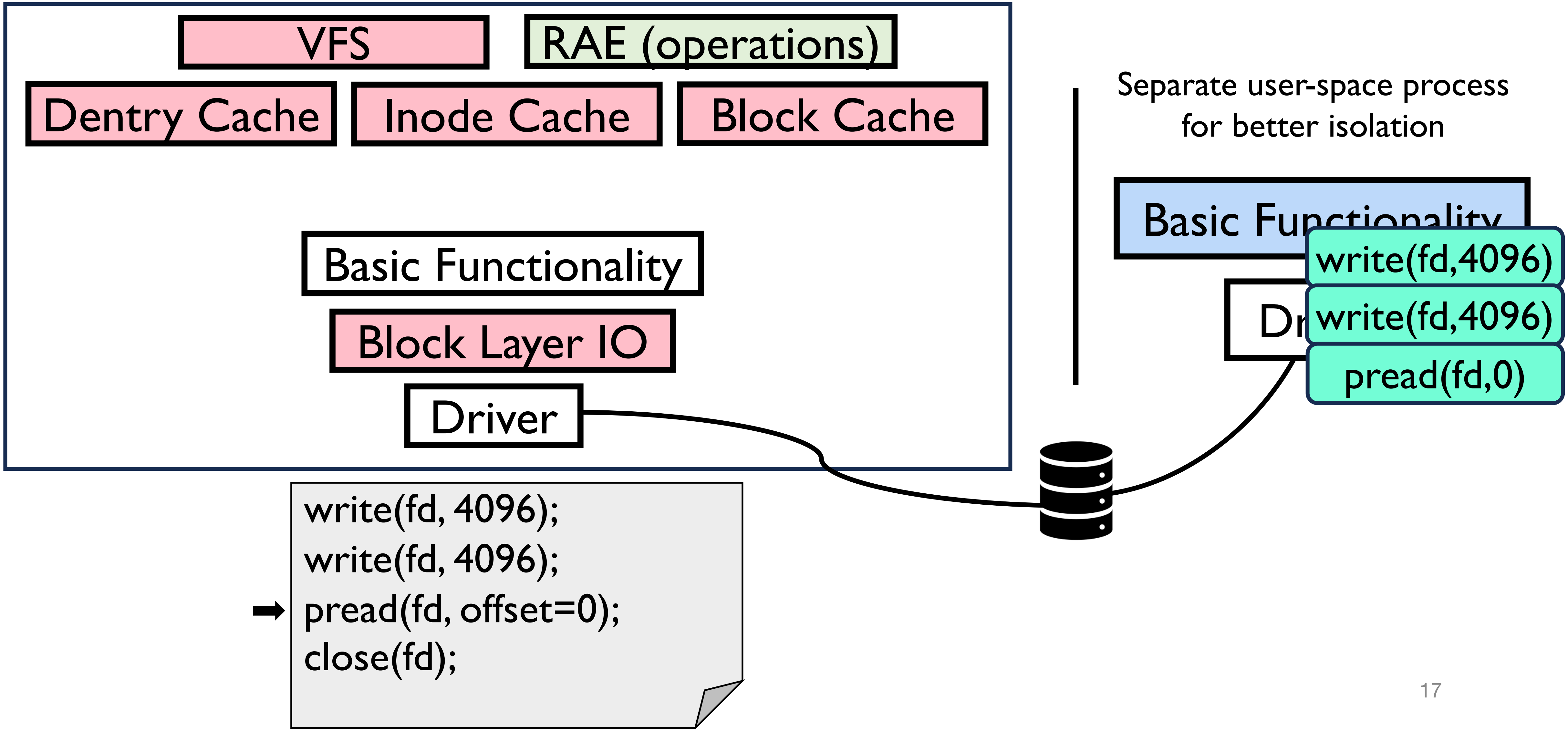

```
write(fd, 4096);
write(fd, 4096);
➡ pread(fd, offset=0);
close(fd);
```

Clean up Base

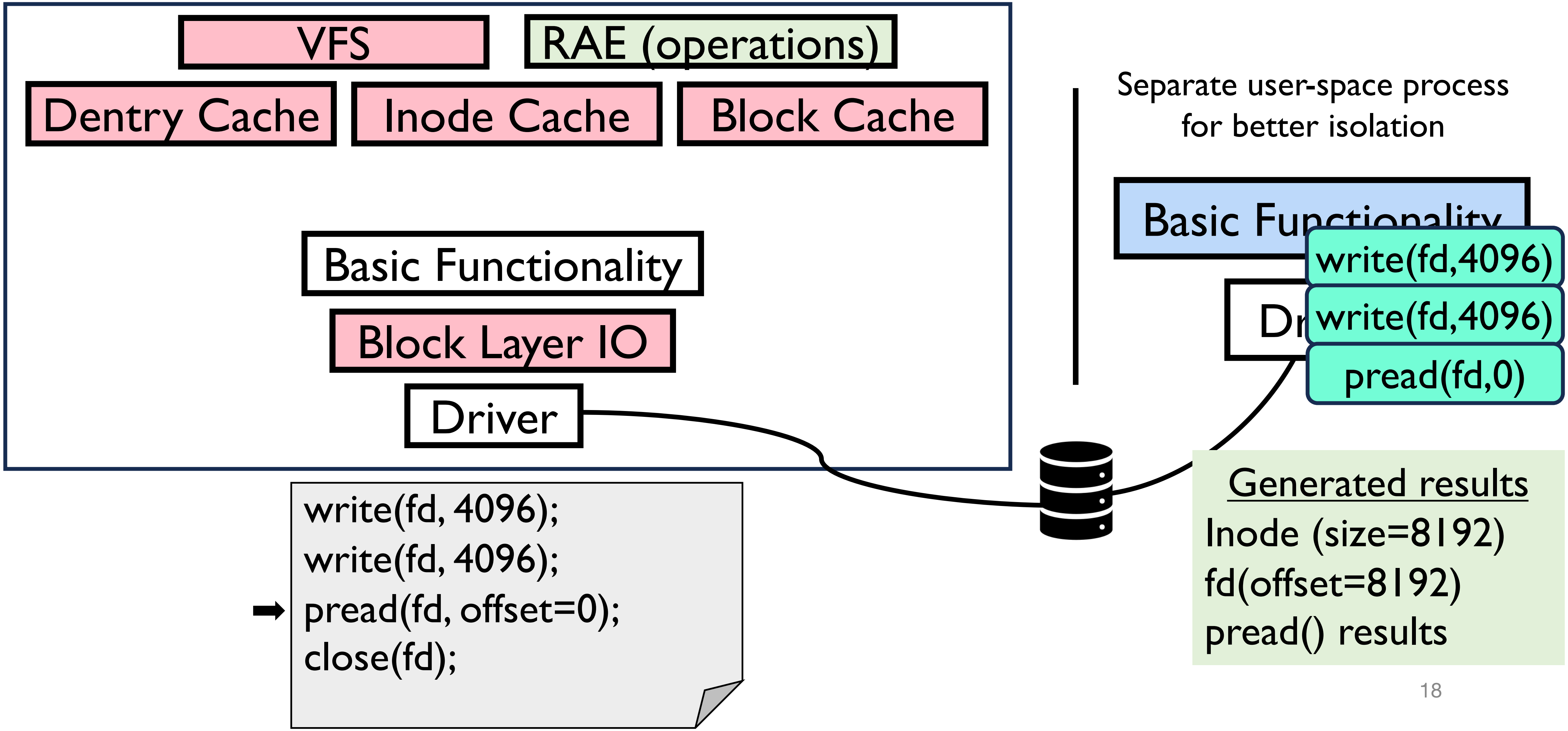



```
write(fd, 4096);
write(fd, 4096);
➡ pread(fd, offset=0);
close(fd);
```

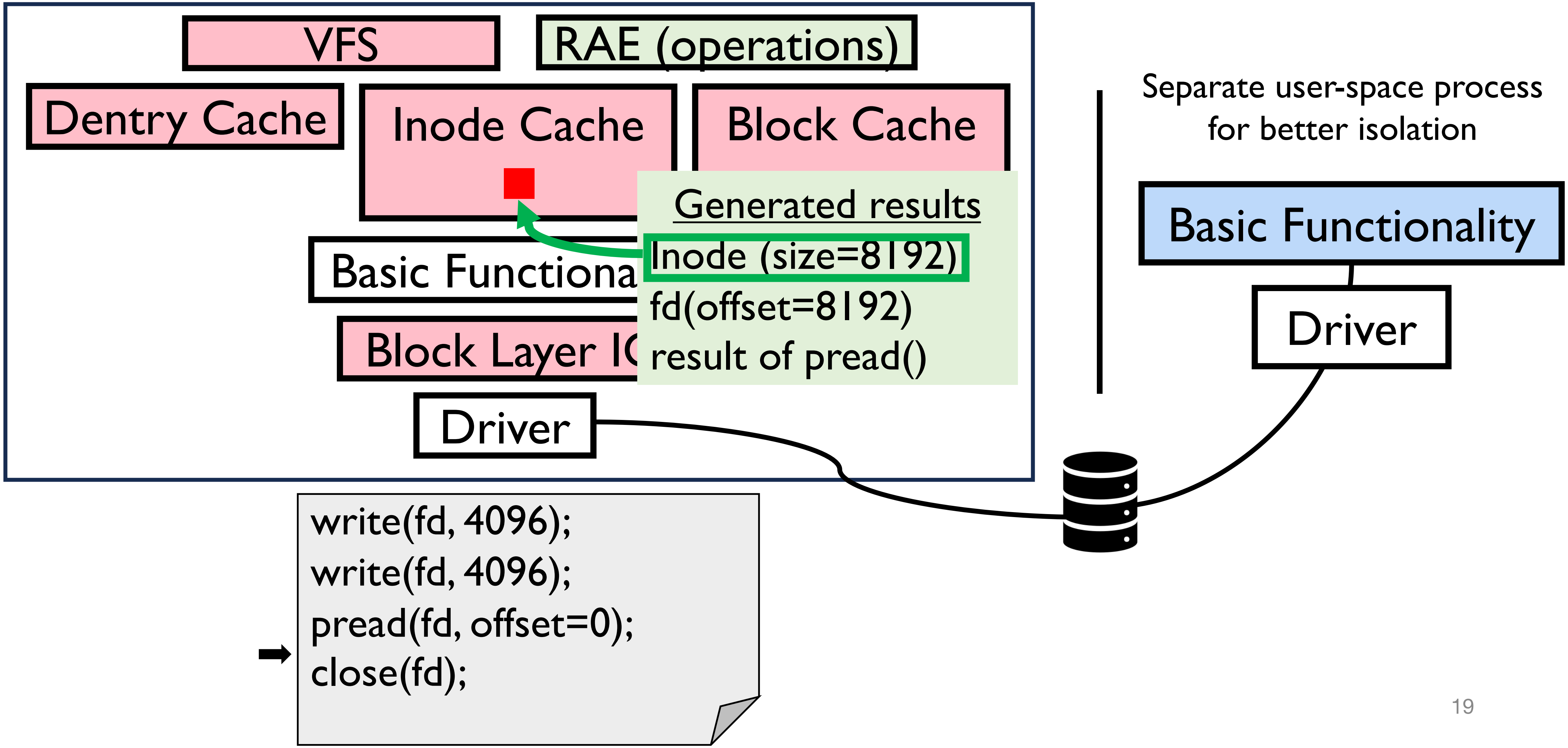
Hand-off to Shadow



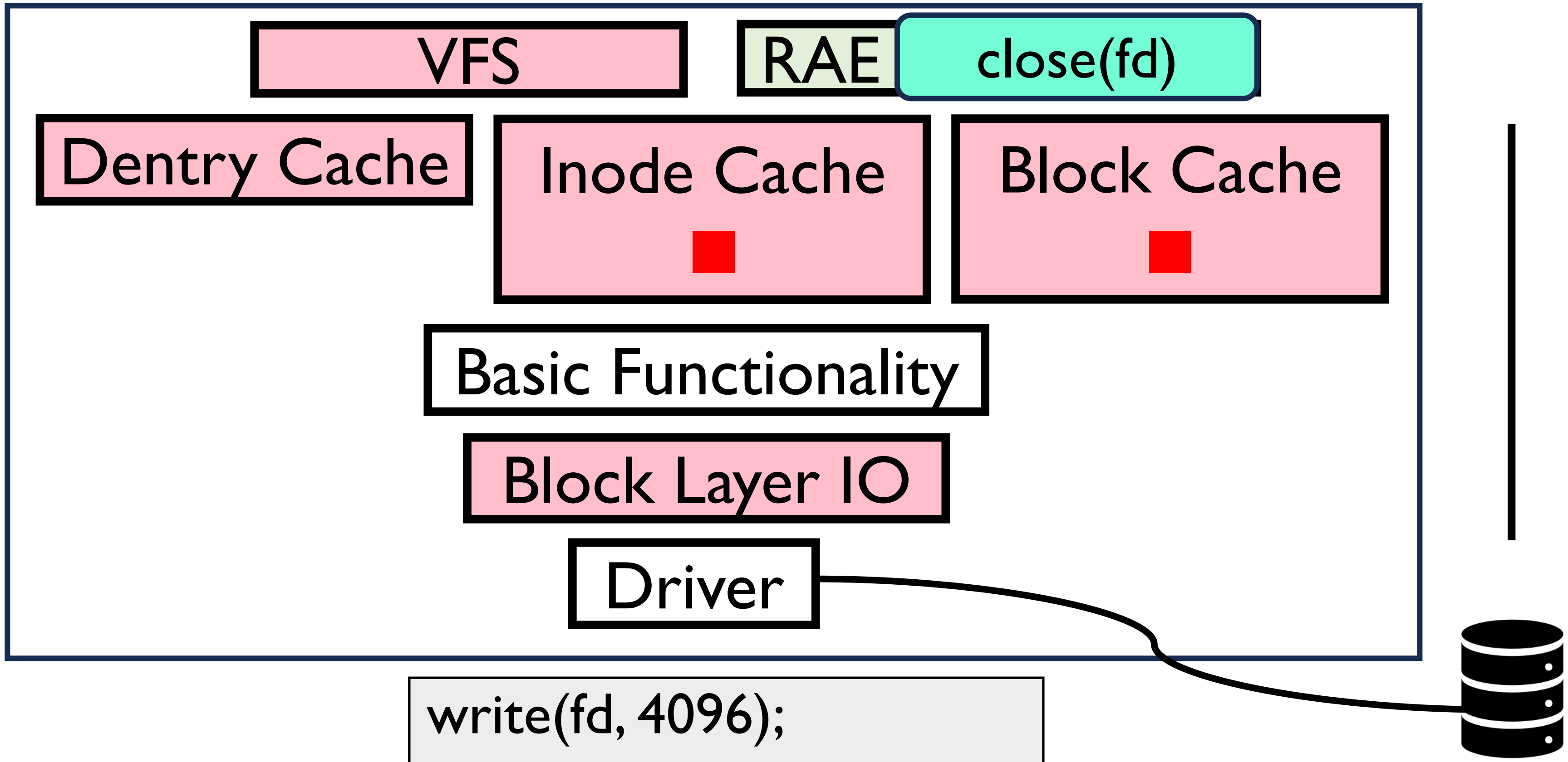
Shadow Executes the Workload, Correctly!



Base Obtains the Results



Base Continues




```
write(fd, 4096);
write(fd, 4096);
pread(fd, offset=0);
close(fd);
```

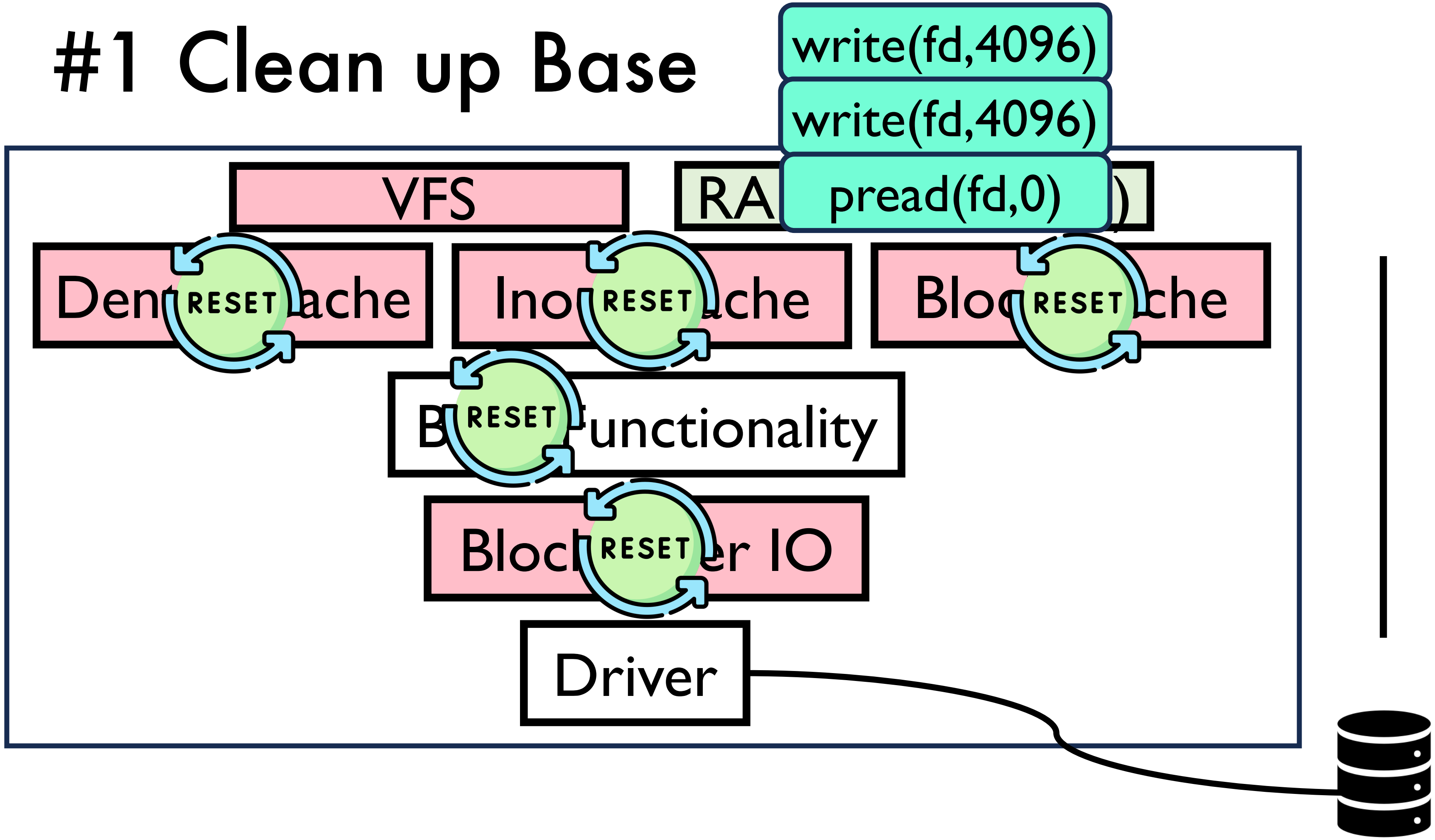
Challenges

#1 Clean up the base

#2 Correctness of the shadow

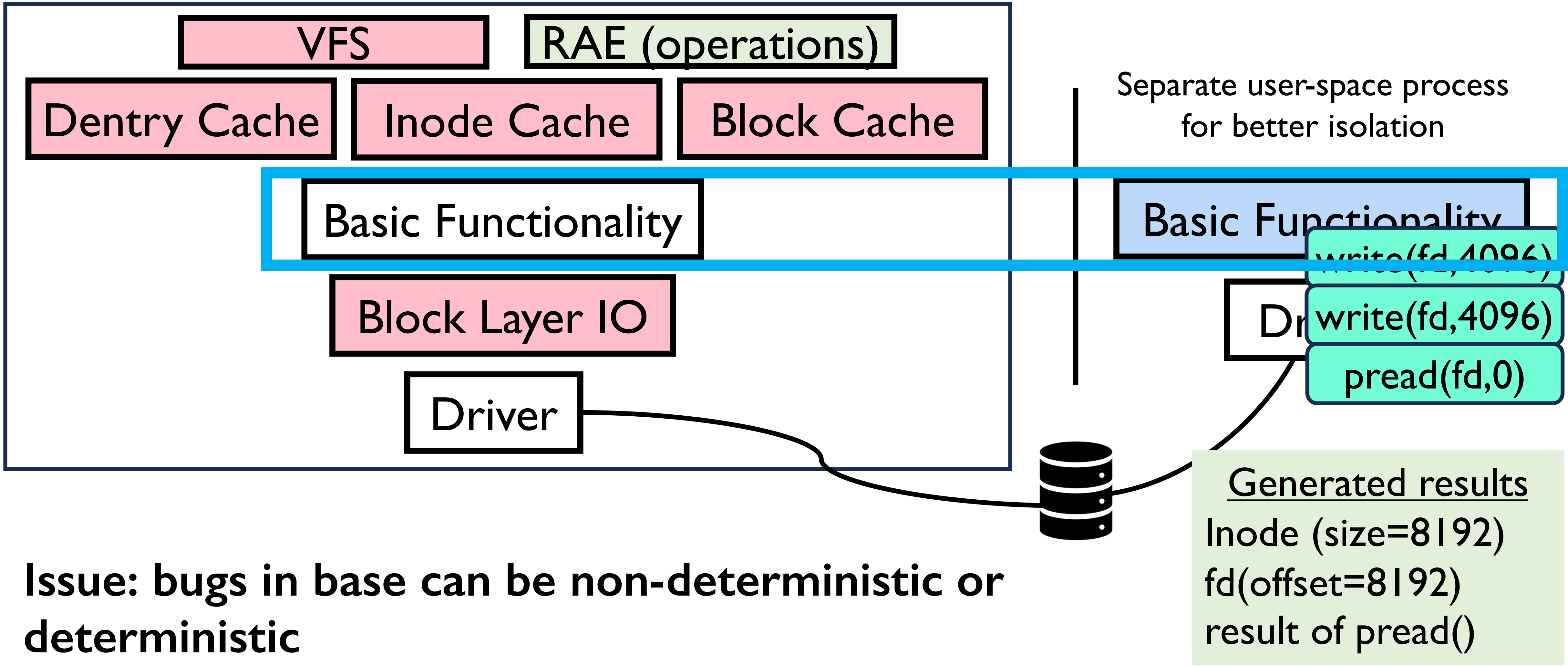
#3 Base obtains the results

#1 Clean up Base



Issue: reset the base components without restarting the OS to clean up buggy states

#2 Correctness of the Shadow



Issue: bugs in base can be non-deterministic or deterministic

A Deterministic Bug in ext4 (CVE 2022-1184)

```
#/bin/bash
mount -o loop tmp32.img mnt  # a corrupted image
mv mnt/foo/bar mnt/foo/YzoUYCy4vTth45i7... ZIOFz
mv mnt/foo/YzoUYCy4vTth45i7... ZIOFz  mnt/foo/AIdkBBulG0Pp5lbV... 7oF
```

> A use-after-free flaw was found in fs/ext4/namei.c:dx_insert_block() in the Linux kernel's filesystem sub-component. This flaw allows a local attacker with a user privilege to cause a denial of service.

Deterministic bugs are challenging to recover from

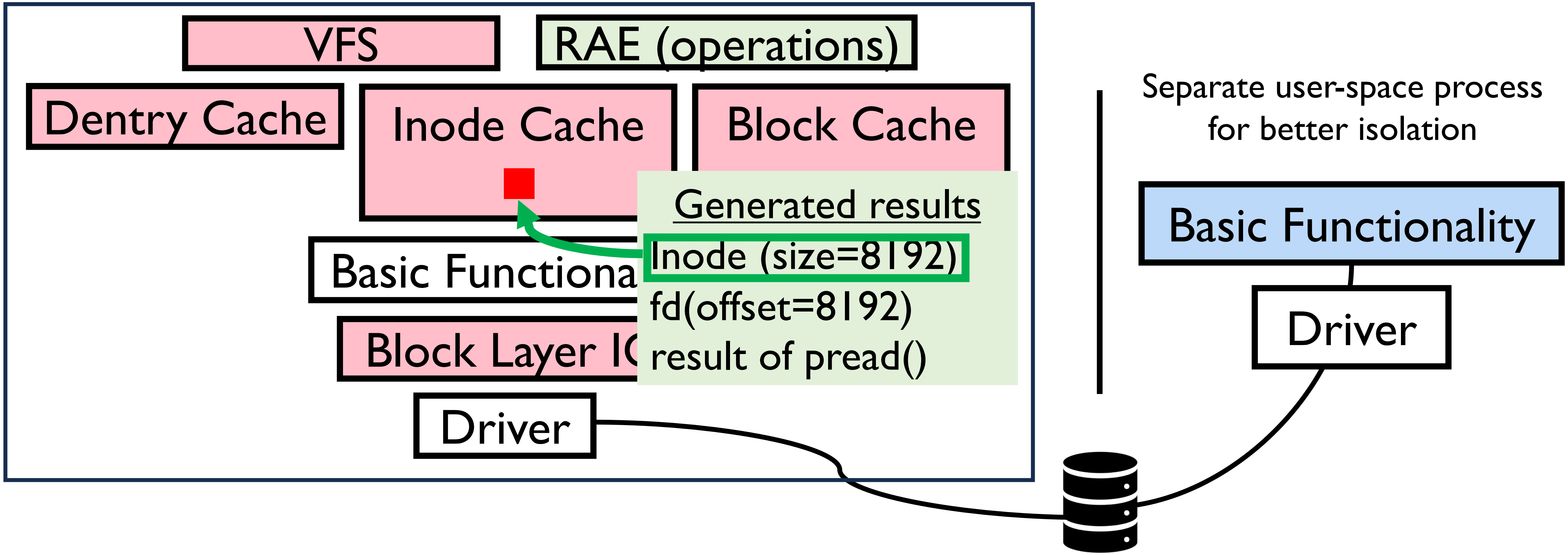
- retry by the base will fail again
- shadow's benefit

#2 Correctness of the Shadow

Techniques

- **A much simpler implementation from scratch**
  - Only basic functionality
  - Without any performance component
- **Fully-verified implementation is practical**
  - “Simple” enough for verification
  - Implementation from scratch makes verification easy

#3 Base Obtains the Results



Issue: base needs to continue with shadow's output

#3 Base Obtains the Results

Techniques

- Metadata downloading
  - Base directly reads the results from known directory (e.g., in /tmp/inodes), but not from disk
    - Shadow never writes to disk
  - Base exposes APIs to read shadow's output
    - E.g., InitInodeCache(path=/tmp/inodes)

Three Challenges

#1 Clean up the base

- reset all components in the base without restarting the OS

#2 Correctness of the shadow

- simple implementation from scratch and fully-verified

#3 Base obtains the results

- new API in base to read from (in-memory) temporary files

Prototype and Progress Status

Prototyping in uFS

- A high-performance microkernel style filesystem (SOSP '21)
- Clean up the base
  - restart the process is enough
- Correctness of the shadow
  - 35K Loc C++ (base) vs. 2.5K Loc Rust (shadow)
  - Verification of the rust implementation is in progress
    - Verus: automatic prover for rust language

Outline

Introduction

Robust Alternative Execution (RAE)

Prototype and Progress Status

Future Challenges

Future Challenges

Testing the discrepancies

- Given a workload, what if shadow and base produce different results?

Future Challenges

Testing the discrepancies

Trusted code

- The interaction between base and shadow
  - Hand-off
  - Downloading

Future Challenges

Testing the discrepancies

Trusted code

Design the shadow to be friendly to verify

- Interesting issues due to Rust's interaction with driver (i.e., C code)
- On-disk format is within the specification
  - E.g., handle crafted image

Future Challenges

Testing the discrepancies

Trusted code

Design the shadow to be friendly to verify

Maintain the shadow while the base evolves

- Shadow can be a “simple enough spec.” to evolve as well
- An up-to-date document

Future Challenges

Testing the discrepancies

Trusted code

Design the shadow to be friendly to verify

Maintain the shadow while the base evolves

Linux kernel filesystems

- “Reset the base without restarting the OS” and “Metadata downloading” are more challenging
- Each base (ext4, btrfs) needs one shadow

Summary

Robust Alternative Execution

Two filesystems to achieve both high performance and correctness

- An existing base: optimized for performance
- Build a shadow
  - From scratch
  - Avoid any performance optimization
  - Fully-verified implementation
- Coordination between base and shadow

Summary

Robust Alternative Execution

Two filesystems to achieve both high performance and correctness

- An existing base: optimized for performance
- Build a shadow
  - From scratch
  - Avoid any performance optimization
  - Fully-verified implementation
- Coordination between base and shadow

Thank you!