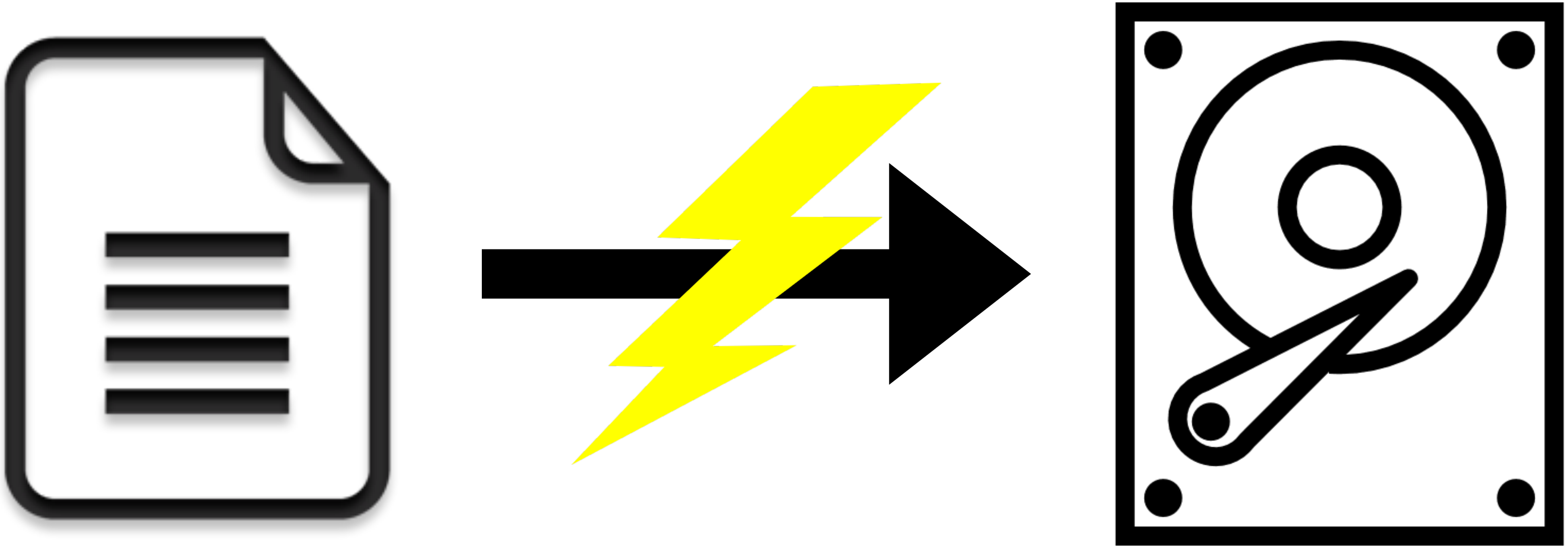
Finding File System Crash Consistency Bugs Through Fuzzing and Verification

Hayley LeBlanc, Vijay Chidambaram, James Bornholt, Isil Dillig

The University of Texas at Austin



Example crash consistency bug

```
mkdir(A);fsync(A);CRASH!
```

Output of `ls –l` in parent directory:

Expected:

```
total 0
drwxr-xr-x 2 root root 4096 Nov  9 08:23 A
```

Actual:

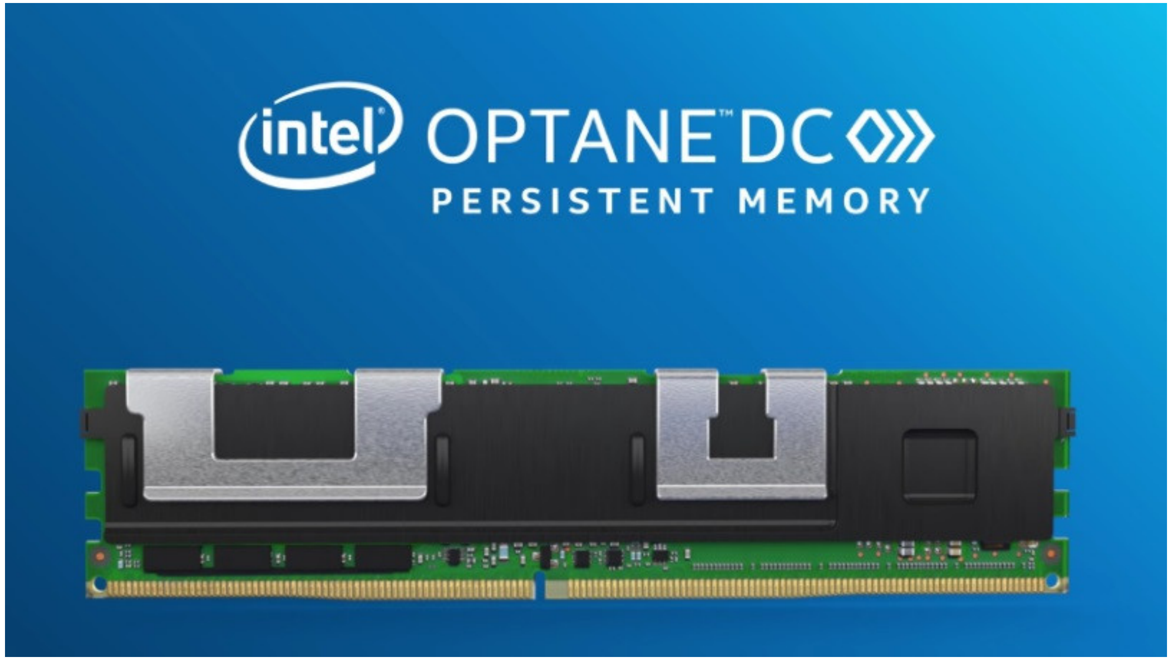
```
ls: cannot access 'A': Input/output error
total 0
d????????? ? ? ? ? ? A
```

Projects

1. Fuzzing for persistent memory file system crash consistency bugs
2. Ext4 journal verification

Persistent memory (PM)

- Non-volatile
- Similar performance to DRAM
- Byte-addressable
- High capacity
- File systems: NOVA, SplitFS, Strata, ext4-DAX…



General approach

- Based on CrashMonkey+ACE (OSDI'18)
- Record-and-replay approach
  1. Run a workload that accesses the file system
  2. Record writes to persistent media via file system
  3. Replay writes up to simulated crash point
  4. Check consistency
- Some new challenges…

Challenges

- How to log writes?
  - CrashMonkey: intercept block I/O by mounting FS on wrapper block device
- PM writes are made via memory load/store interface
- Data must be explicitly flushed or bypass cache with special assembly instructions to guarantee durability Key issue: **no software layer at which to intercept writes**

Challenges

- What types of programs should we test on?
- Few known bugs, little existing work

The NOVA team was aware of **one** crash-consistency bug in their file system

Current approach

- Observation: most writes/flushes to PM in file systems are made by a small set of central library functions
- Loadable kernel module to automatically instrument PM writer functions
- Baseline: tests from ACE
- Found 4 bugs in NOVA with ACE-generated test cases!
  - All confirmed and fixed in main NOVA repo

NOVA crash consistency bugs

- NULL pointer dereference in recovery procedure on 1 core due to bug in per-CPU metadata access
  - **Made crash recovery impossible** on single core machines
- **`fsync`'ed directory inaccessible** due to missing flush on an inode field after `mkdir`
- **File system unwritable** due to lack of flush of updated inumber information after `mkdir`
- **New directory unreadable and undeletable** due to lack of flush on inode valid field after `mkdir`

Next steps

- Fuzzing
  - Generate new test programs based on past programs that exposed bugs
  - Syzkaller (Linux kernel fuzzer) for generating syntactically valid sequences of file system calls

Projects

1. Fuzzing for persistent memory file system crash consistency bugs
2. Ext4 journal verification

File system verification

- Can we formally prove that a file system has no crash consistency bugs?
- Some prior work
  - FSCQ (SOSP '15)
  - Yggdrasil (OSDI '16)
- Problem: no work on verifying *existing* file systems

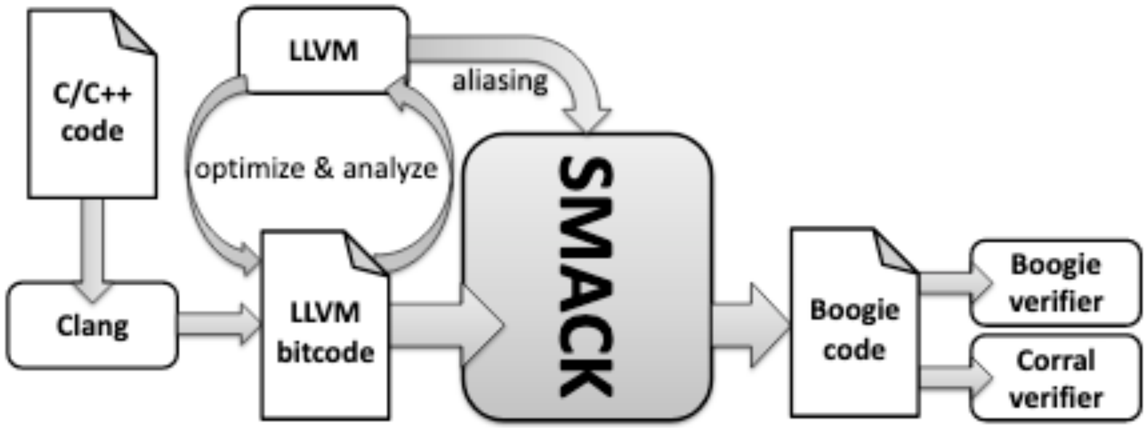
Very quick verification background

- Hoare triples: {P} S {Q}
  - For a precondition P, a statement S, and a postcondition Q: assume P and execute S.
  - If Q always holds, the triple is *valid*
- Hoare triples form basis of deductive program verification
- Can specify a program using Hoare triples and check correctness using SMT solver

Our goal: formally verify Linux's JBD2 journaling system

Current approach

- Exploring both bounded model checking and full deductive verification
- One possible workflow:
  - Boogie: intermediate verification language
  - Corral: bounded verifier
  - SMACK: C → Boogie translator



Challenges

- How best to model on disk state?
- How to reduce amount of manual effort?
- Is bounded verification feasible?
- Is full verification feasible?

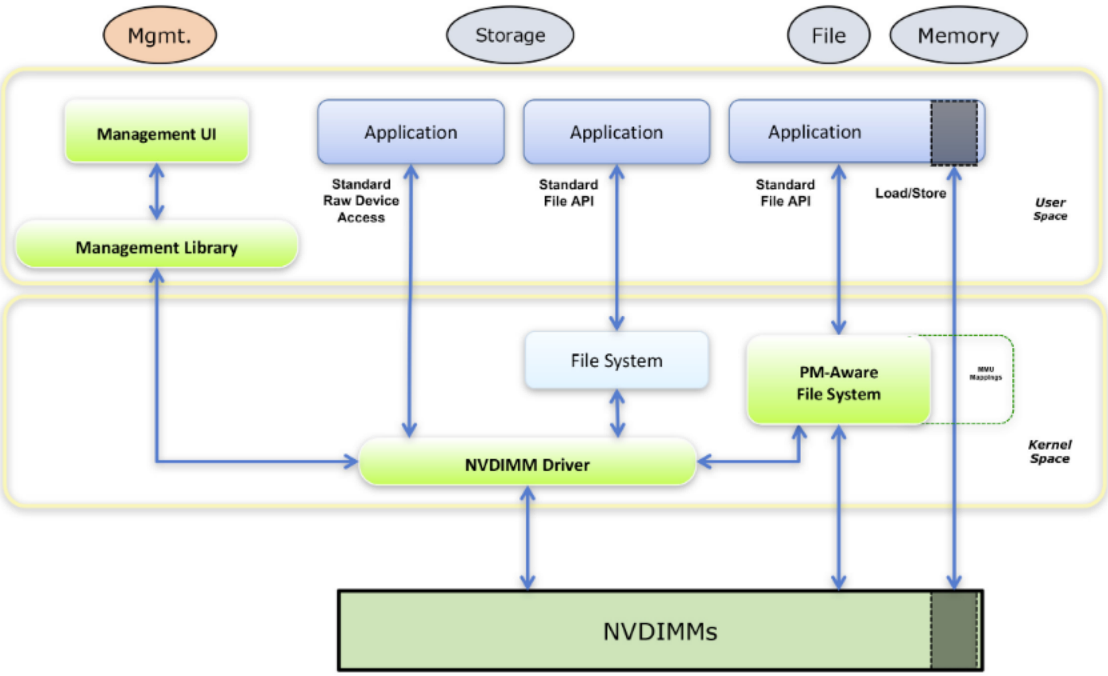
Conclusion

- Crash consistency bugs can have serious consequences in real file systems
  - PM file systems
  - Mature systems like ext4
- Exploring 2 approaches to finding bugs
  - Record-and-replay + fuzzing for PM file systems
  - Formal verification of ext4 journal

Supplemental slides

Writing to PM

- x86
  - CLWB: flush a cache line to persistent memory
  - SFENCE: enforces order in which memory stores become globally visible
  - CLWB+SFENCE: enforces order in which data is made durable in PM



No CLWB → data is not guaranteed to be persisted!

Very quick ext4 background

- Ext4: most widely used Linux file system
- Uses a *journal* (JBD2) to ensure crash consistency
  - Make a note of new operations in journal before actually executing them
  - If we crash, replay journal onto the main FS

Transaction 1 start	Some data	Some more data	Transaction 1 end	Transaction 2 start	…

Corral

- Reachability problem: for a control flow graph, does there exist a path from the initial state to the error state?
    - I.e., is there an execution that establishes the presence of an error?
    - In general, recursively enumerable and undecidable
- Reachability is decidable for *bounded* programs

Corral

- Takes a recursion bound from the user
- Statically inlines loops and recursive procedures up to provided bound
- Inlined program can be verified as though it is a program with no loops
  - Makes verification decidable because all possible executions can now be explored