Silhouette: Leveraging Consistency Mechanisms to Detect Bugs in Persistent Memory-Based File Systems

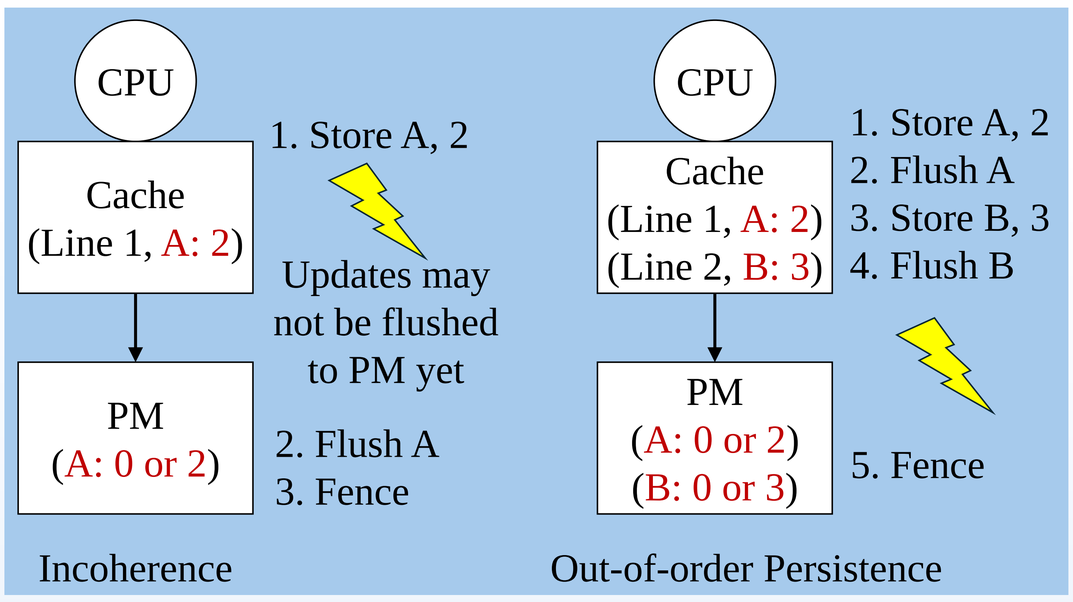
Bing Jiao — *Florida State University*
Ashvin Goel — *University of Toronto*
An-I Andy Wang — *Florida State University*

Problem

- Detecting crash-consistency bugs in PM file systems requires exploring all subsets of in-flight stores at fence operations
- Search space is large, N in-flight stores may lead to 2^N crash scenarios

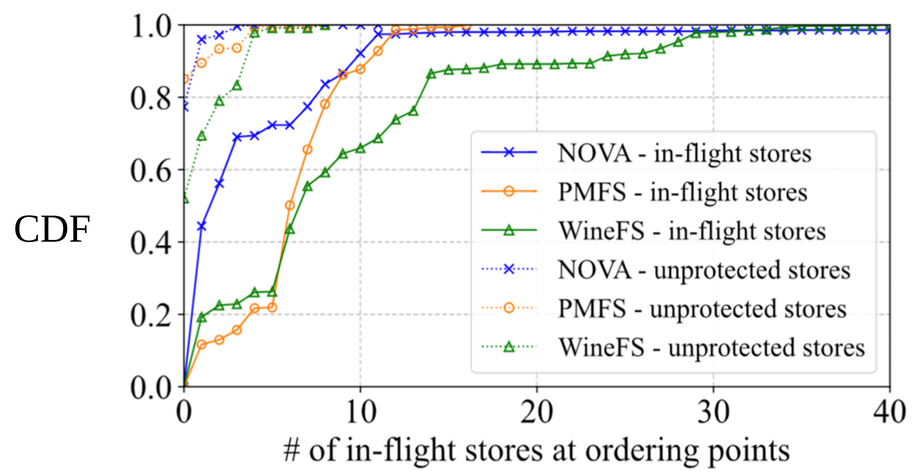
Background

PM programs are prone to crash-consistency bugs because they need to flush stores from CPU caches to PM and correctly order them using fence operations



Key Idea

- PM file systems use well-known crash consistency mechanisms (e.g., journaling and log-structured writes) to provide atomicity and durability guarantees.
- We can check whether a file system implements its crash consistency mechanism correctly
- Then we only need to explore (unprotected) stores that are not associated with these mechanisms

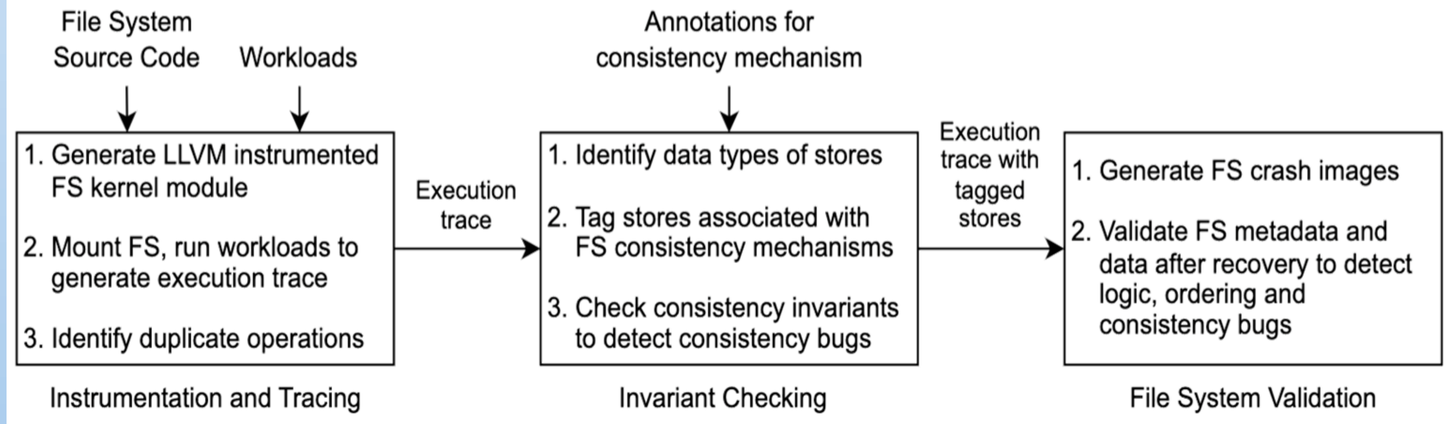


Cumulative Distribution Function of in-flight and unprotected stores in PMFS, NOVA, and WineFS under ACE seq-3 workload.

Silhouette

We propose **Silhouette,** a bug detection tool that combines static instrumentation and data-type-based dynamic analysis to check:

- Whether each crash-consistency mechanism protect its stores correctly
- Whether remaining unprotected stores are crash-consistent when reordered

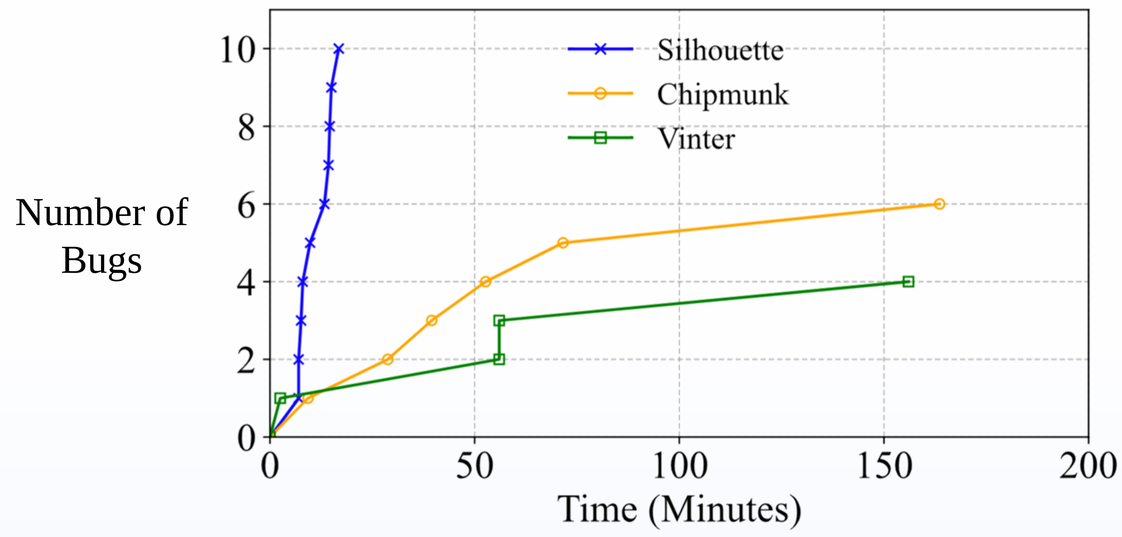


The Silhouette architecture.

Evaluation

We tested Silhouette on NOVA, PMFS, and WineFS and found 15 new bugs (refer to the paper for the full list):

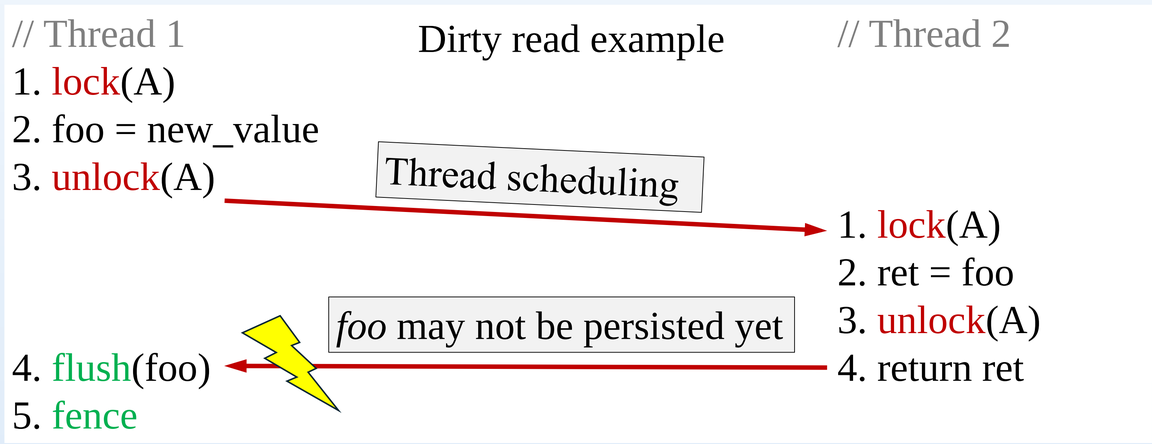
- **Segfault** due to incorrect pointer persistence in NOVA
- **Data leak** since *truncate* is not atomic in NOVA
- **Data loss** due to reusing *inodes* in orphan list in PMFS and WineFS



Bug finding time of Silhouette, Chipmunk, and Vinter.

On-going Work

Dirty Reads: PM programs may have dirty read bugs when a thread reads data that has been modified but not persisted or committed by another thread



Durinn [OSDI'22] and PMRace [ASPLOS'22] have explored such bugs but their approaches are inaccurate or inefficient because they rely on heuristics or fuzzing

Ideas:

- Reading unpersisted data is a special kind of data race
- Lockset algorithm is good at detecting data races
- Adopt Lockset algorithm for PM programs

Lock interval of A: [1, 3] { 1. lock(A), 2. store foo, 3. unlock(A) } } Persist interval of *foo*: [2, 5] { 4. flush(foo), 5. fence }

In Thread 1: lockset when writing *foo*: $[2,5] \not\subset [1,3] \Rightarrow \{\emptyset\}$
In Thread 2: lockset when reading *foo*: $\{A\}$
$\{\emptyset\} \cap \{A\} \Rightarrow \{\emptyset\} \Rightarrow$ PM Data Race

Other challenges:

- How to detect data that is persisted but not yet committed
- How to detect happen-before-induced dirty reads
- How to avoid or detect false positives
- How to validate bugs efficiently

Scan to access Silhouette source code ⟹