Paper title: Durinn: Adversarial Memory and Thread Interleaving for Detecting Durable Linearizability Bugs
Paper authors: Xinwei Fu, Dongyoon Lee, Changwoo Min
Your name: Parth Kansara (115135130)

What problem does the paper address? How does it relate to and improve upon previous work in its domain?
Concurrent data structures used in systems based on the non-volatile memory are required to uphold linearizability even in case of a crash. This durable linearizability needs to be integrated into the data structure manually by the programmer, by flushing the cache lines and using the store fence instructions. This makes it prone to errors and a challenging task since there is a gap between the point where the operations on a data structure become visible to the system (linearization point) and the point where the changes performed by the operation are persisted in the NVM (durability point). This is aggravated by the lack of sufficient testing solutions for the DL of these concurrent data structures.

Several models exist that perform consistency checking on file systems, but Durinn is the first model for testing concurrent data structures in NVM systems. It uses adversarial crash states and thread interleaving construction to reduce the test space. Among the file system models, **CrashMonkey** leverages a bug study to create heuristics that can limit the search space, while **EXPLODE** and **FiSC** perform checks in the original model. Several database systems, such as **Salt** and **Hekaton** use *buffered durable linearizability*, which delays the persistence of changes as long as one of the old consistent states can be used to resume. **Line-up**, **Round-up** and **Pradel et al.** are some of the linearizability checkers, performing checks for deterministic linearizability, non-deterministic linearizability and concurrency bugs respectively.Several softwares exist for checking NVM bugs but none of them are suited for checking DL. **PMDebugger** checks universal bugs while **Yat**, **PMTest**, **XFDetector** and **Agamotto** check application-specific bugs and depend on user-defined checkers. **Jaaru** only detects bugs that have visible manifestation, while **Witcher** is limited to single-threaded NVM programs.

What are the key contributions of the paper?
The paper addresses the challenges in upholding durable linearizability and based on their analysis, present three types of bug patterns that violate the DL. These bug patterns are based on when the crash occurs relative to the gap in the linearization point and the durability point. If the crash occurs after DP, the crashed operation should provide the guarantee that it will completed and persisted. This is known as the “all” semantic guarantee. Violation of this may give rise to an **Incompletely-Durable bug (ID)**. If the crash occurs before DP, either before or after LP, the crashed operation must guarantee that none of the changes are visible. Violation of this gives rise to an **Uncovered-Durable bug (UD)**. Lastly, in case of concurrent operations, if a crash occurs between LP and DP, the crashing operation may be visible to the concurrent operation, but its changes are not durable. This results in a **Visible-But-Not-Durable bug (VD)**.

Based on these bug patterns, the authors present the first DL checker for concurrent NVM data structures called **Durinn**. The testing space for detecting these bug patterns is very huge as a crash may occur at any point resulting in the eviction of cache lines in a random order. Further, the number of thread interleavings is also exponentially high. To combat this, Durinn uses an adversarial technique to construct NVM states and thread interleaving that may result in the bug patterns. For adversarial NVM state construction, Durinn decides the persistence of the store operations before the crash based on which DL bug is to be triggered. For the ID bug, Durinn does not persist as many preceding stores as possible while for the UD bug, as many preceding stores as possible are persisted. For the VD bug, Durinn constructs a thread interleaving such that a concurrent operation is scheduled between LP and DP of another operation. The above operations require knowledge of the location of the LP. Durinn detects likely-LPs using static program analysis to detect atomic operations, memory initialization after which the memory region is made visible to other threads and the read or write of a guardian flag. Based on the publish-after-initialization idiom, all the store operations on newly allocated memory regions are excluded from likely-LPs. Once these likely-LPs are inferred and NVM crash traces & thread schedules are generated, they are validated to detect the DL bugs.

Briefly describe how the paper's experimental methodology supports the paper's conclusions.
Durinn detected 27 DL bugs out of which 15 were new and 7 of which were confirmed by developers. It inferred 2000 likely-LPs out of 15.3K store instructions, which reduced the test space by almost 87%. Although Druinn infers almost twice the number of likely-LPs as actually present, it manages to detect the same number of DL bugs as done using the manually detected actual LPs. Durinn was able to detect 10 VD bugs unlike the state-of-the-art tool Witcher. In terms of the test space reduction, Durinn performs better reduction as compared to both Yat and Witcher, while detecting more bugs. These evaluations indeed conclude that Durinn is an effective DL checker for concurrent NVM data structures.

Write down one question you may want to ask the authors.
Can we measure the overhead of the extra likely-LPs detected by Durinn and could there be any heuristics to eliminate them?