This technical report presents the durable algorithms with selective logging and lazy persistency. All updates on the new nodes created within a transaction are log-free and eager persistent.

LL: The backward pointer of the linked list node is log-free and eager persistent. The head pointer to the incoming node is logged but lazy persistent. The recovery applies the log to the corrupted data. It then fixes the inconsistency between forward and backward pointers by detecting and removing jump pointers from a node to another node that is not next to it.

Hash: Each bucket stores a pointer to a directly linked list. The pointer is null if there is no key-value pair assigned to the bucket. The pointer is logged and lazy persistent if it was not null before the update. Otherwise, the pointer is log-free and eager persistent. The setup ensures that the recovery can find all key-value pairs assigned to a bucket by scanning from the pointer stored in the bucket.

BST: Each node contains two pointers to children. The pointers are log-free and eager persistent. The recovery detects dangling pointers with the help of a crash-consistent memory allocator. It set those pointers to null.

LRU: On the LRU stack hit, all pointers are log-free and eager persistent except for the forward pointer to the node to access. The pointer is logged and eager persistent, such that the recovery can infer the correct order of nodes on the linked list. On LRU stack miss, the setup of the hash table is the same as *Hash*. All pointers are log-free and eager persistent. The recovery first recovers the hash table. It validates whether the head pointer refers to unallocated memory space. In such a case, the crash interrupts an LRU stack miss. The recovery recovers the head pointer with backward scanning the linked list from the tail. The tail pointer may be null. In this case, the recovery finds the node present in the hash table but not the linked list. It sets the node as the tail.

SkipList: The forward pointer in the lowest level is logged and eager persistent. All other pointers are log-free. The recovery detects and fixes the inconsistency of upper-level linked lists with the lowest level linked list.

BPTree: The data structure is a variant of a prior proposal[1]. The proposal uses multiple fences and flushes operations. This proposal uses transactions with selective logging, incurring only one fence on transaction commit. Specifically, the data at the place where the new key will be inserted and the last key of every cache line are logged and eager persistent. All other keys are log-free and lazy persistent. The recovery detects and removes redundant keys by shifting the keys of each cache line. It recovers the last key of every cache line with the log records.

[1] Hwang, D., Kim, W. H., Won, Y., & Nam, B. (2018). Endurable transient inconsistency in byte-addressable persistent b+-tree. In *16th USENIX Conference on File and Storage Technologies (FAST 18)* (pp. 187-200).