Enabling Persistent Memory Use in Java

Steve Dohrmann
Sr. Staff Software Engineer, Intel

Motivation

- **Java is a very popular language on servers, especially for databases, data grids, etc., e.g. Apache projects:**
  - **Cassandra**
  - **Ignite**
  - **HBase**
  - **Lucene**
  - **Spark**
  - **HDFS**
- **Want to offer benefits of persistent memory to such applications**

Java Access to Persistent Memory

- Volatile use
  - Allocation of Java Heap on Alternative Memory Devices,
    - whole heap: openjdk.java.net/jeps/316
    - new / old gen split: bugs.openjdk.java.net/browse/JDK-8202286
- Persistent or Volatile use
  - Persistent MappedByteBuffer, openjdk.java.net/jeps/8207851
  - Low-Level Persistence Library, github.com/pmem/llpl
  - Persistent Collections for Java, github.com/pmem/pcj

Low-Level Persistence Library (LLPL) and Persistent Collections for Java (PCJ)

	LLPL	PCJ
	github.com/pmem/llpl	github.com/pmem/pcj
Persistent data	memory blocks	Java collections and other objects
Memory management	manual	automatic
Thread-safe	no*	yes
Data consistency	assisted	built-in
granularity	user-defined	transactional methods
policy	user-defined	ACID

* Heap API used to manage blocks is thread-safe

Low-level Persistence Library

https://github.com/pmem/llpl

- Heap API to allocate and free blocks of persistent memory
- MemoryBlock API
  - set / get Java scalars and copy bytes between blocks and arrays
- Three kinds of heaps / memory blocks
  - TransactionalHeap - modifications roll back if interrupted
  - PersistentHeap - modifications are durable, opt. transactional
  - Heap - volatile use or persistent with custom consistency schemes
- User Transaction API to aggregate transactional modifications

LLPL Example Code


```
01 TransactionalHeap heap = TransactionalHeap.getHeap("/mnt/mem/heap1", 10000000L);
02
03 long ID_OFFSET = 0;
04 long NAME_OFFSET = 8;
05 long EMPLOYEE_SIZE = Long.BYTES + Integer.BYTES + 20;
06
07 // allocate and initialize an employee block
08 TransactionalMemoryBlock employee = Transaction.run(heap, () -> {
09   TransactionalMemoryBlock block = heap.allocateMemoryBlock(EMPLOYEE_SIZE);
10   long id = 12345;
11   byte[] nameBytes = "John Doe".getBytes();
12   block.setLong(ID_OFFSET, id);
13   block.setInt(NAME_OFFSET, nameBytes.length);
14   block.copyFromArray(nameBytes, 0, NAME_OFFSET + Integer.BYTES, nameBytes.length);
15   return block;
16 });
17
18 // allocate block to hold array of employee handles
19 TransactionalMemoryBlock employees = heap.allocateMemoryBlock(10 * Long.BYTES);
20 // store handle to this root data structure in the heaps root location
21 heap.setRoot(employees.handle());
22 // add employee to employee array
23 employees.setLong(0, employee.handle());
```

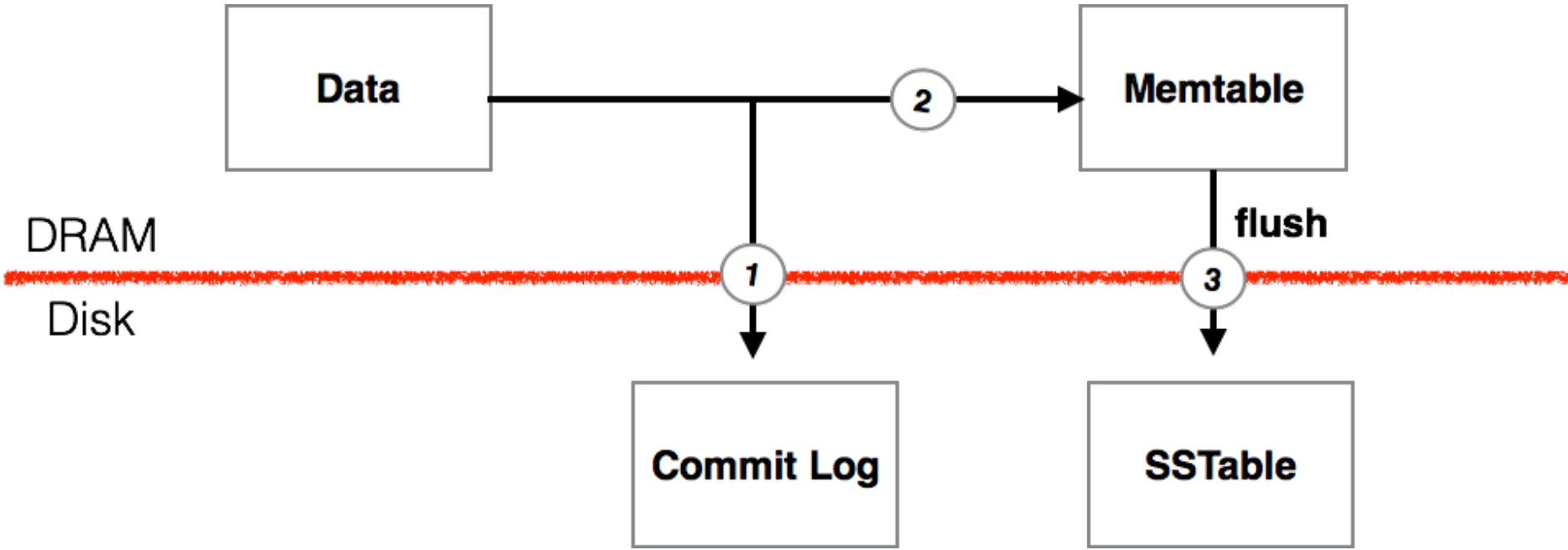
LLPL Example Code (continued)


```
01 // restart
02
03 // open heap
04 TransactionalHeap heap = TransactionalHeap.getHeap("/mnt/mem/heap1");
05
06 // retrieve handles to employee array and first employee
07 TransactionalMemoryBlock employees = heap.memoryBlockFromHandle(heap.getRoot());
08 TransactionalMemoryBlock employee = heap.memoryBlockFromHandle(employees.getLong(0));
09
10 // read employee data
11 byte[] nameBytes = new byte[employee.getInt(NAME_OFFSET)];
12 employee.copyToArray(NAME_OFFSET + Integer.BYTES, nameBytes, 0, nameBytes.length);
13 String name = new String(nameBytes);
14 System.out.format("id = %d, name = '%s'\n", employee.getLong(ID_OFFSET), name);
```

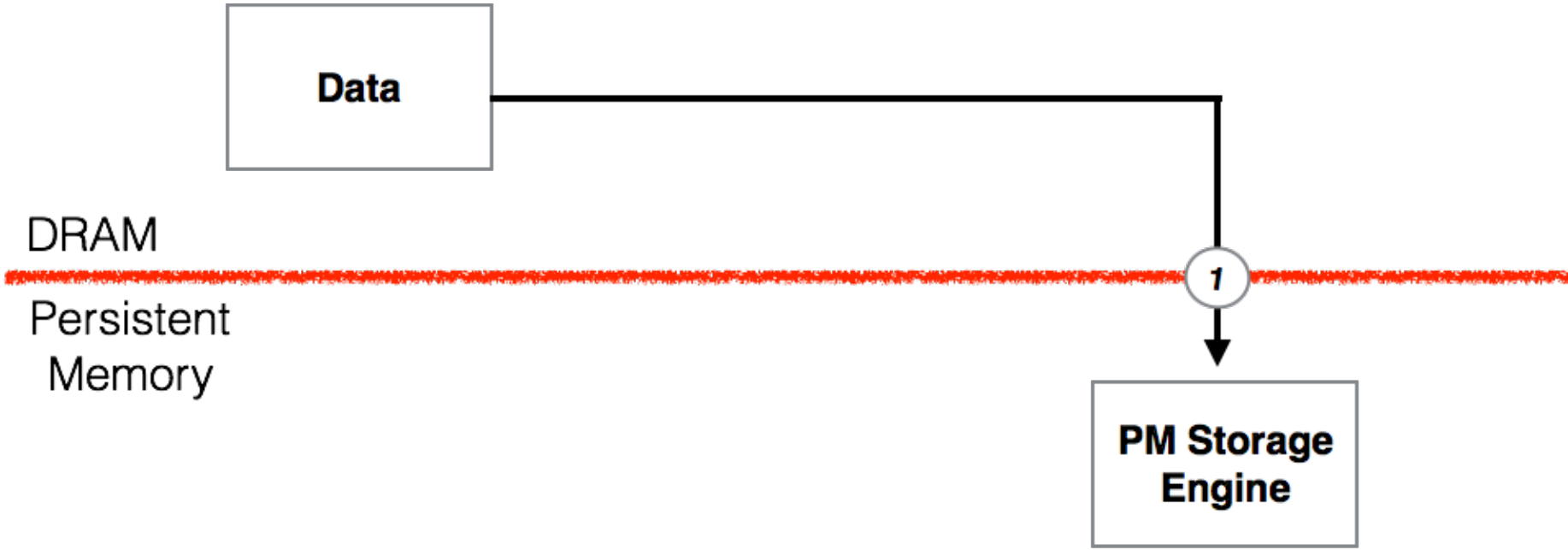
PM Storage Engine for Cassandra

- **Cassandra is a popular distributed NoSQL database written in Java**
- **Uses a storage engine based on a Log Structured Merge Tree with DRAM and disk levels**
- **Could persistent memory offer Cassandra opportunities for simpler code and improved performance?**

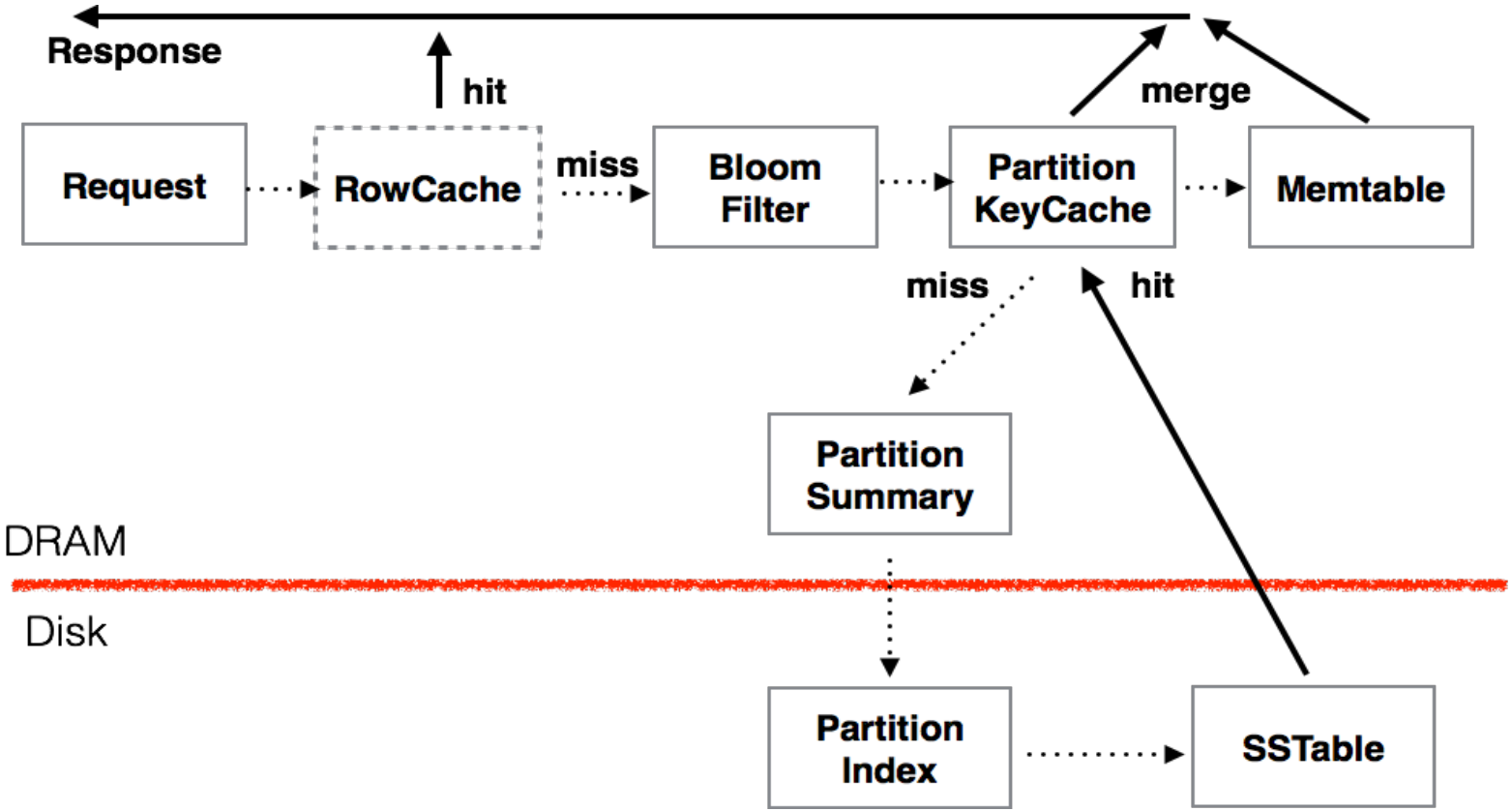
Cassandra Write Path



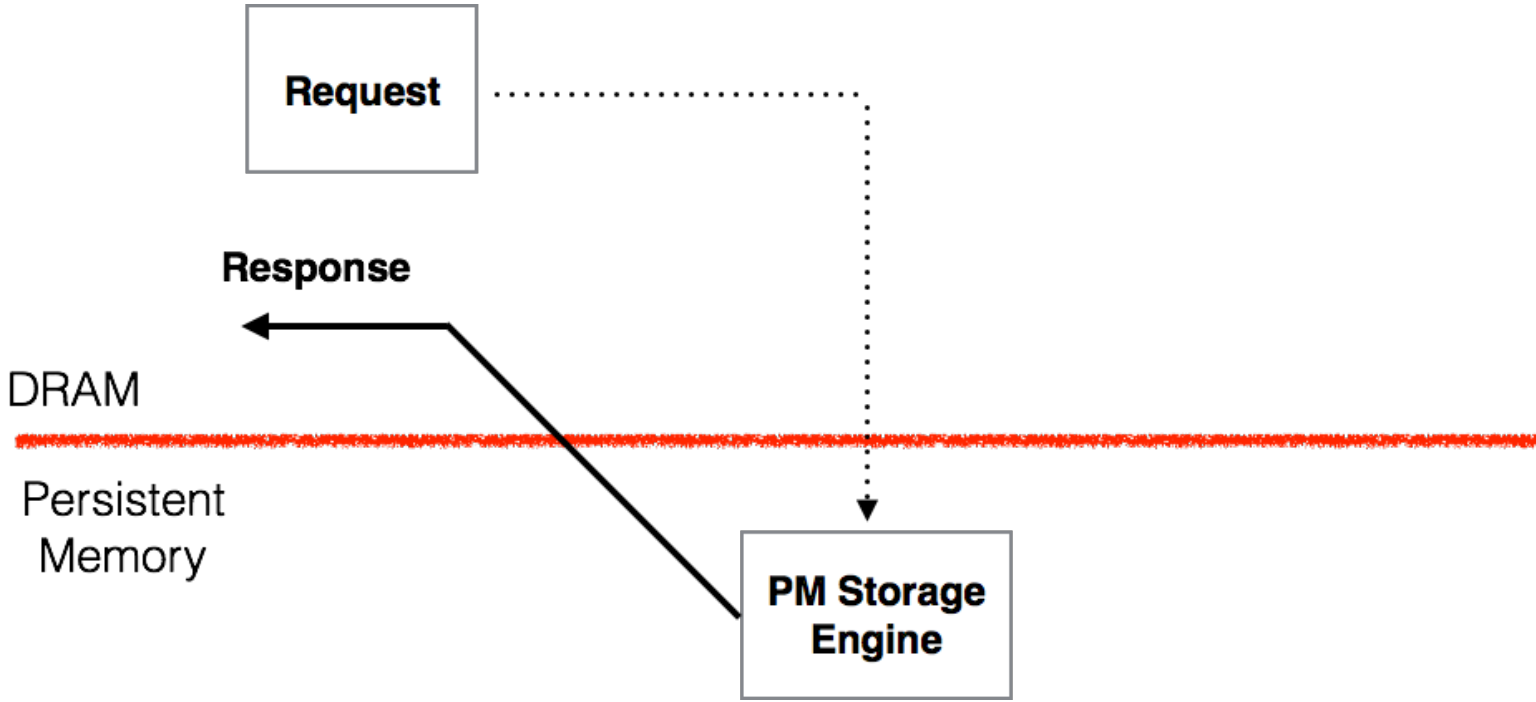
Cassandra Write Path – PM Storage Engine



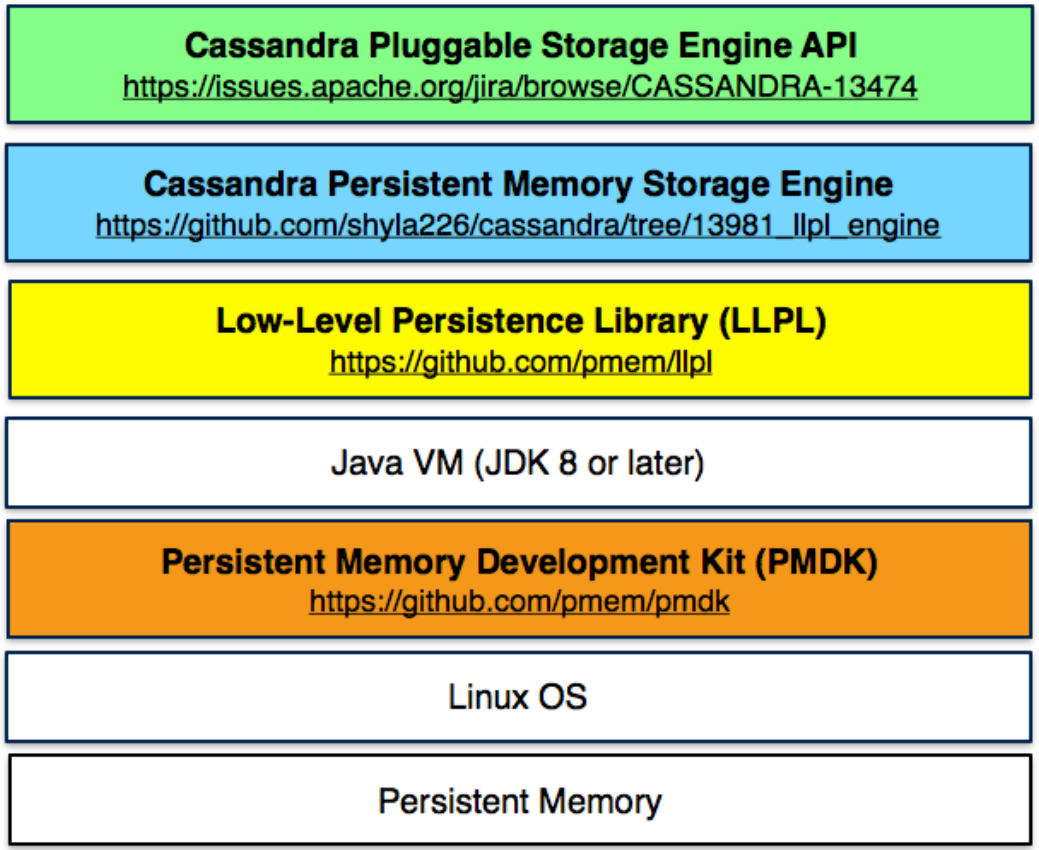
Cassandra Read Path



Cassandra Read Path – PM Storage Engine

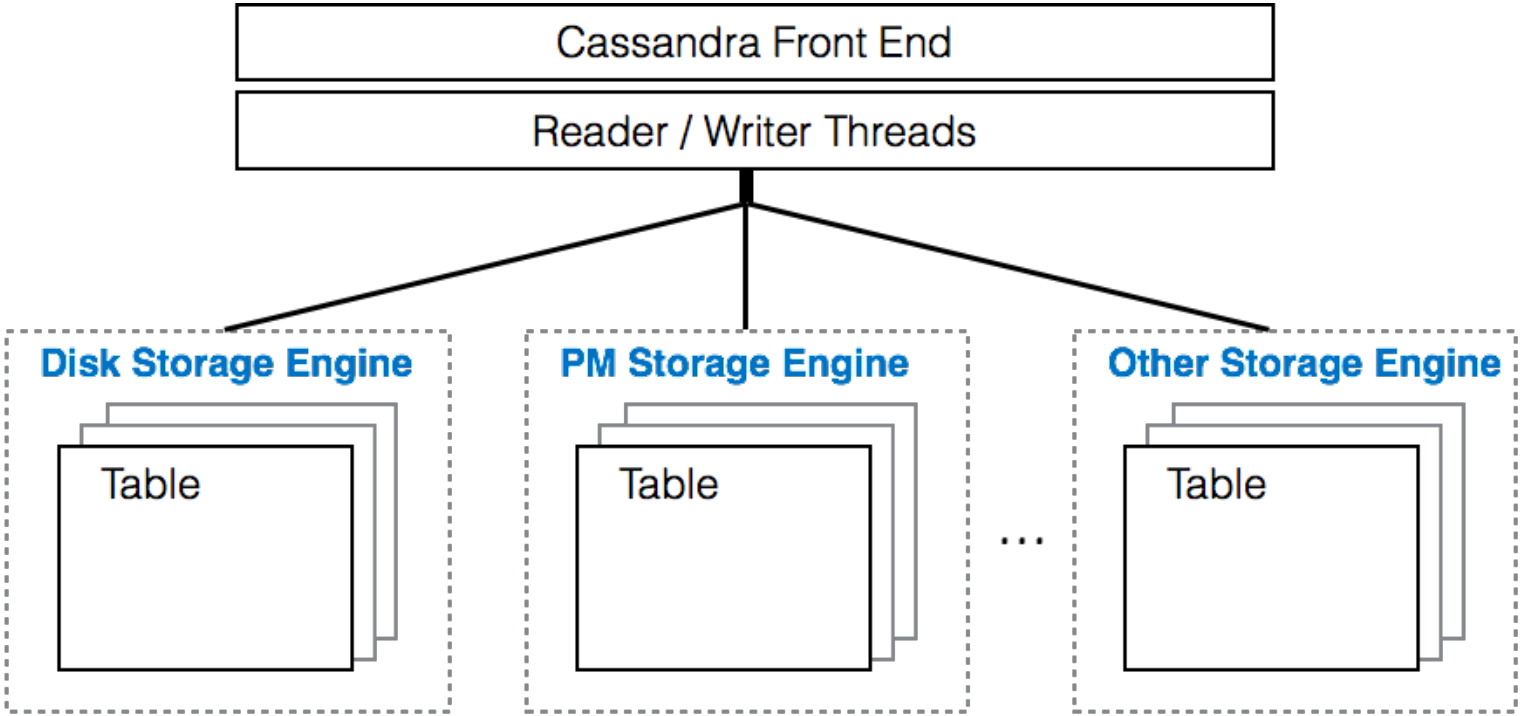


Software - Persistent Memory Storage Engine



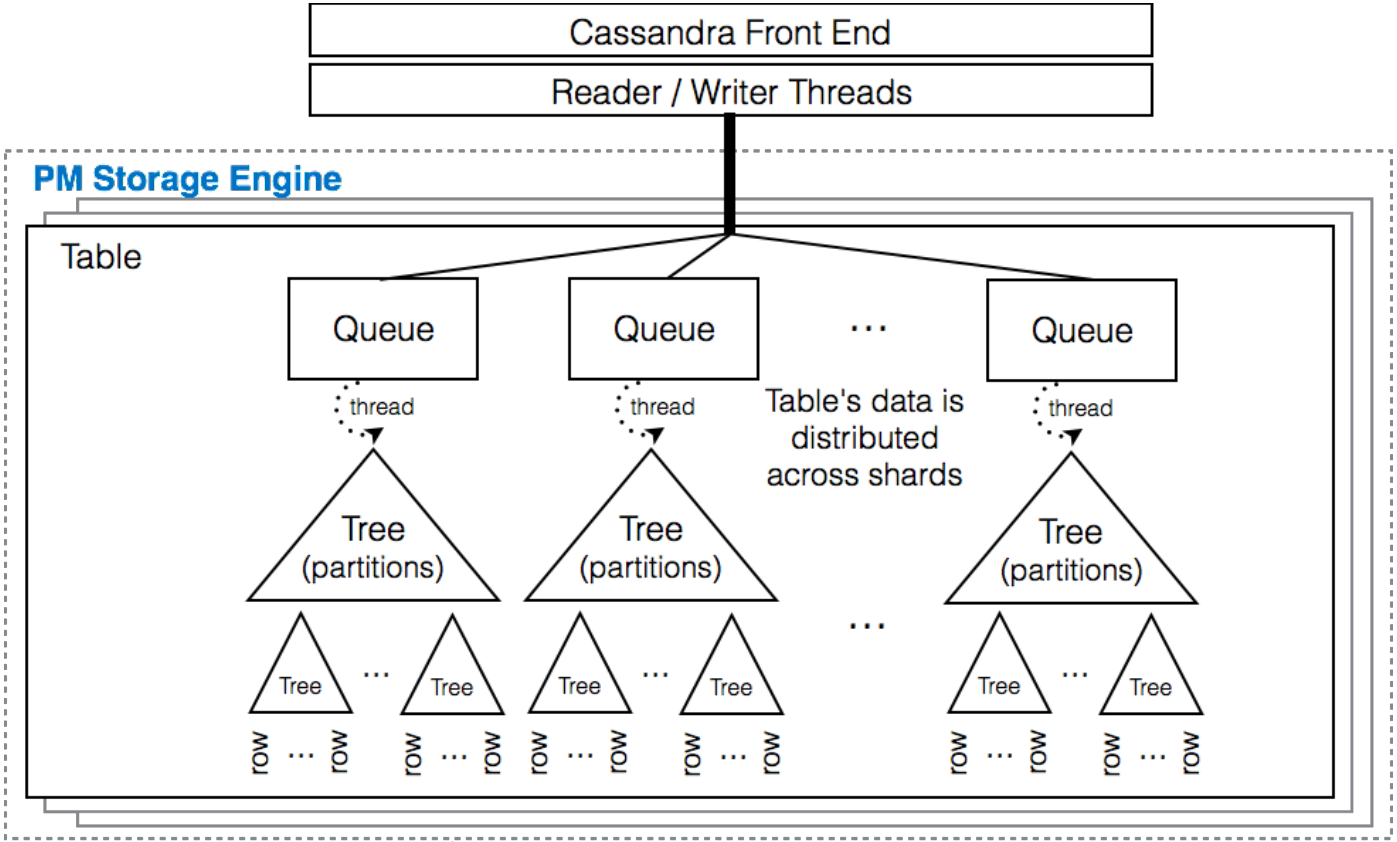
Cassandra Pluggable Storage Engine

https://issues.apache.org/jira/browse/CASSANDRA-13474



Alternate engines or mixture of engines at table granularity

Shard-based Storage Engine



Summary

- **Low-Level Persistence Library and other efforts let Java developers program persistent memory today**
- **Pluggable storage APIs can enable alternate, compatible storage back-ends going forward**
- **Cassandra persistent memory storage engine is an example of a pluggable design that shows promising simplicity and performance**

Backup

Persistent Collections for Java

https://github.com/pmem/pcj

- Library of persistent classes
- Object state stored on a persistent heap in object-layout form
- Instances behave like regular Java objects, just longer-lived
- Garbage collected, reachability-based lifetime
- Changes to persistent state (including setting of fields) are done using transactional Java methods
- API for declaring custom persistent classes
- No change to developer tool-chain
- Transaction API to aggregate already transactional operations

PCJ - Examples of Persistent Classes

- **Primitive arrays** (e.g. PersistentByteArray, mutable and immutable)
- **PersistentArray**<E extends AnyPersistent> (mutable and immutable)
- **PersistentArrayList**<E extends AnyPersistent>
- **PersistentHashMap**<K extends AnyPersistent, V extends AnyPersistent>
- **PersistentSkipListMap**<K extends AnyPersistent, V extends AnyPersistent>
- **ObjectDirectory** - root map of <String, T extends AnyPersistent>
- **Primitive types** (as field and array element values, no separate class)
- **PersistentString**
- **PersistentByteBuffer**

PCJ Example Code


```
01 PersistentIntArray a = new PersistentIntArray(1024);   // Ints are allocated on persistent heap
02 a.set(0, 123);                                         // 4-byte int value written to persistent heap
03 a = null;                                              // Array is unreachable. Object will be collected
04
05
06 PersistentIntArray data = new PersistentIntArray(1024);
07 ObjectDirectory.put("Application_data", data);         // no serialization, reference to array is written
08 data.set(0, 123);
09
10 // restart
11
12 PersistentIntArray data1 = ObjectDirectory.get("Application_data", PersistentIntArray.class);
13 assert(data.get(0) == 123);
14
15
16
17 PersistentArrayList<PersistentString> movies = new PersistentArrayList<>();
18 PersistentArrayList<PersistentString> movieIndex = new PersistentArrayList<>();
19
20 public void addMovie(PersistentString movie) {
21    Transaction.run(() -> {
22       movies.add(movie);
23       movieIndex.add(movie);
24    });
25 }
```

Transactions

Use of Persistent Memory Development Kit

- PCJ and LLPL libraries depend on PMDK for:
  - provisioning pools of memory
  - memory allocation
  - cache flushing
  - transaction support
- We wrote a small C JNI library to get access to the above functionality in PMDK

Data Dependencies, Lifetimes, and Integrity

- Data dependencies drive lower bounds for lifetime of dependent data
- Want to be able to say something clear about the usability of data after a given event
- Stronger statements about after-event usability require stronger read / write mechanics

Events and After-event State

Example events

- hardware failure
- power failure
- kernel crash
- process crash
- unhandled exception
- handled exception
- controlled process exit

Example after-event state

- undefined
- reinitialized / available, e.g. Java DRAM heap
- before-event state, e.g. via transactions
- not affected by event, e.g. immutable data