Programming with Persistent Memory from Java

Persistent Memory for Java

Intel open-source libraries:

- Low-Level Persistence Library (LLPL)
- Java bindings to PMDK pmemkv library
- Persistent Collections for Java (PCJ) -- experimental

OpenJDK enhancements:

- JEP 352 Persistent MappedByteBuffer -- in JDK14
- JEP 370 OpenJDK java.foreign package -- incubator in JDK16

Links to these are on last slide

LLPL Overview

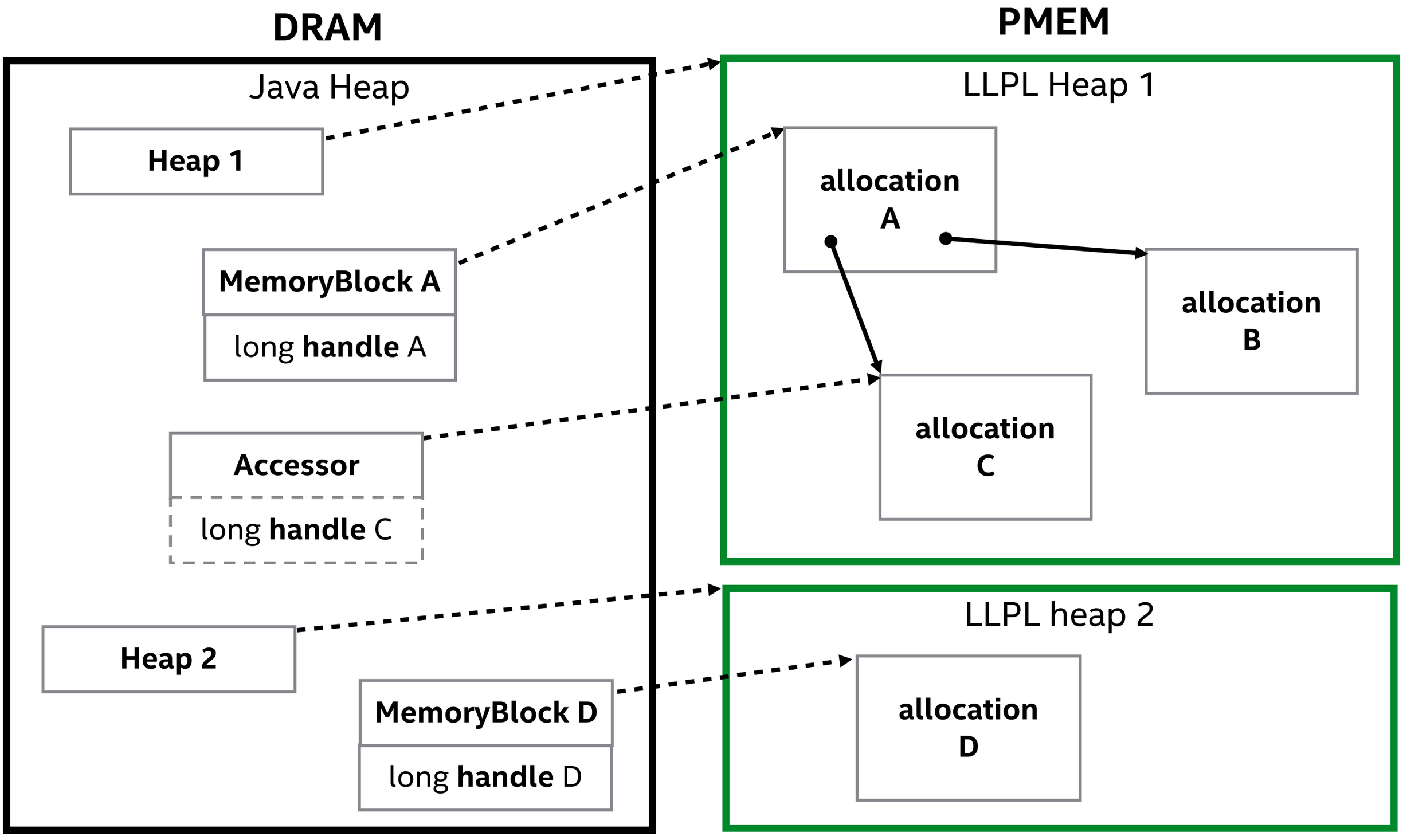
- Intel open-source Java library for persistent memory programming
- Compatible with JDK 8+
- A component of the Persistent Memory Development Kit (PMDK)
- Uses PMDK libraries (libpmem, libpmemobj)
- Version 1.1 on GitHub (github.com/pmem/llpl) and Maven Central
- Version 1.2 in-progress

- Low-level because:
  - Pmem is off-Java-heap memory
  - Manual memory management
  - Manual layout of data
  - LLPL uses Unsafe internally

LLPL in One Slide

- **Heap –** pool of pmem along with an allocator for it; named with a path, create and reopen, thread-safe
- **handle** – a Java long value that refers to an allocation on a heap
- **MemoryBlock** – wraps an allocation handle with an API to read and write values to / from the allocation, not thread-safe
- **Accessor** – same API as MemoryBlock but repositioned (repeatedly) to point at any allocation on a heap; not thread-safe
- Writing to pmem
  - **volatile** -- no expectation of recovering memory contents after JVM exits
  - **durable --** can access memory after a restart, data is consistent if writes aren't interrupted
  - **transactional --** can access memory after a restart, data is consistent, at transaction granularity, even after a crash or power failure

Example -- Java Heap and LLPL Heaps



Access API – MemoryBlocks and Accessors

Write methods:

- setByte
- setShort
- setInt
- setLong
- setMemory
- copyFromArray
- copyFromMemory

Read methods:

- getByte
- getShort
- getInt
- getLong
- copyToArray

Other methods:

- free
- handle
- isValid

Three Kinds of Heaps and Access Objects

Class	**volatile write**	**durable write**	**transactional write**
Heap MemoryBlock Accessor	✓	✓*	✓*
PersistentHeap PersistentMemoryBlock PersistentAccessor	**X**	✓	✓
TransactionalHeap TransactionalMemoryBlock TransactionalAccessor	**X**	**X**	✓
abstract AnyHeap abstract AnyMemoryBlock abstract AnyAccessor	✓	✓	✓
	uses default write of actual concrete class		

✓ default write kind

* manual flush() for durable or manual addToTransaction() for transactional

LLPL Code Examples

Today's workshop

1. IntArray
2. IntArray ("consistency-generic")
3. Array<T> (references)
4. List (minimize construction)
5. RecordLog

Primer examples in repository

1. getting started
2. sizing heaps
3. using other heaps
4. more on transactions
5. wrapping memory blocks

Other examples in repository

1. adaptive radix tree
2. other arrays
3. linked list (more complete)

New Features Coming to LLPL

- MemoryPools
  - each presented as a single space –- no allocator built in
  - access API is similar to MemoryBlock / Accessor
  - sharable between JVM instances
  - no transaction support

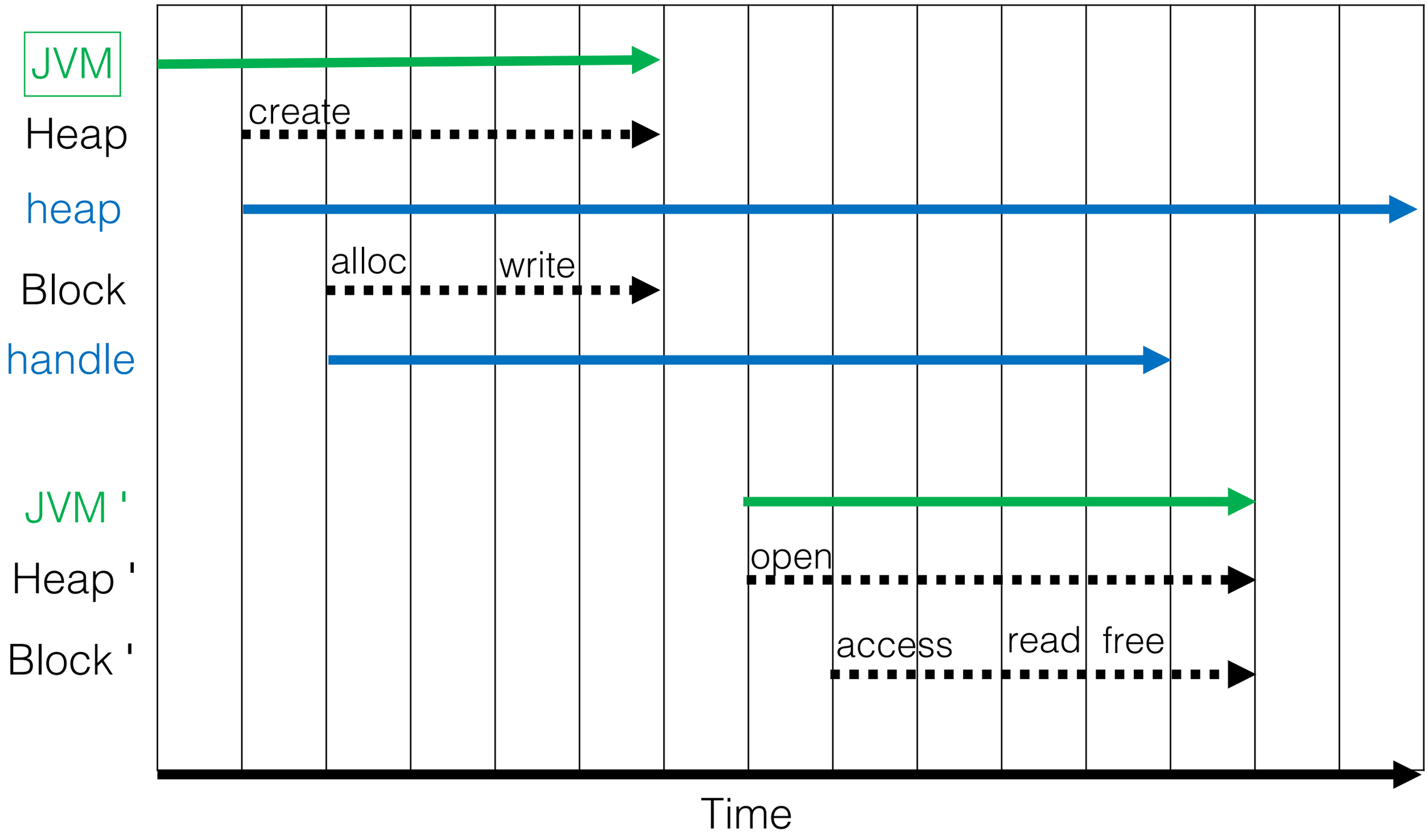
- Production-quality prebuilt data structures
  - Concurrent Adaptive Radix Tree (CART)
  - Arrays
  - List

Backup

Comparison of Libraries

	LLPL	Java bindings to pmemkv	PCJ	Mapped ByteBuffer	java.foreign
Status	release 1.1	release 1.0	experimental	JDK14	incubator JDK16
Compatibility	JDK 8+	JDK 8+	JDK 8+	JDK14+	JDK14
Persistent data	heaps of memory blocks	key-value store	Java collections and other classes	ByteBuffers	structs, unions, arrays, etc.
Memory mgmt.	manual	manual	automatic	manual	manual
Thread-safe	heap: yes blocks: no	yes - optional	yes	no	default - yes
Data integrity / consistency	developer-defined	developer-defined	ACID objects	developer-defined	developer-defined
Transactions	yes	yes - on puts	yes	no	not built-in

Java Object Lifetime vs Pmem Block Lifetime



Heap API

Static methods:

- createHeap
- openHeap
- createAccessor

Allocation methods:

- allocateMemoryBlock
- allocateMemory
- exists
- size

- Other Methods
  - memoryBlockFromHandle
  - execute

Two Kinds of Errors, Three Heap Classes

Three kinds of writes:

- volatile: write data, in CPU cache, not necessarily in memory module
- durable: write data, **flush data from CPU cache to memory module**
- transactional: **add data range to transaction**, write data, flush data

Two new kinds of programming errors:

1. durable write: forget to flush data from cache
2. transactional write: forget to add range to transaction before writing

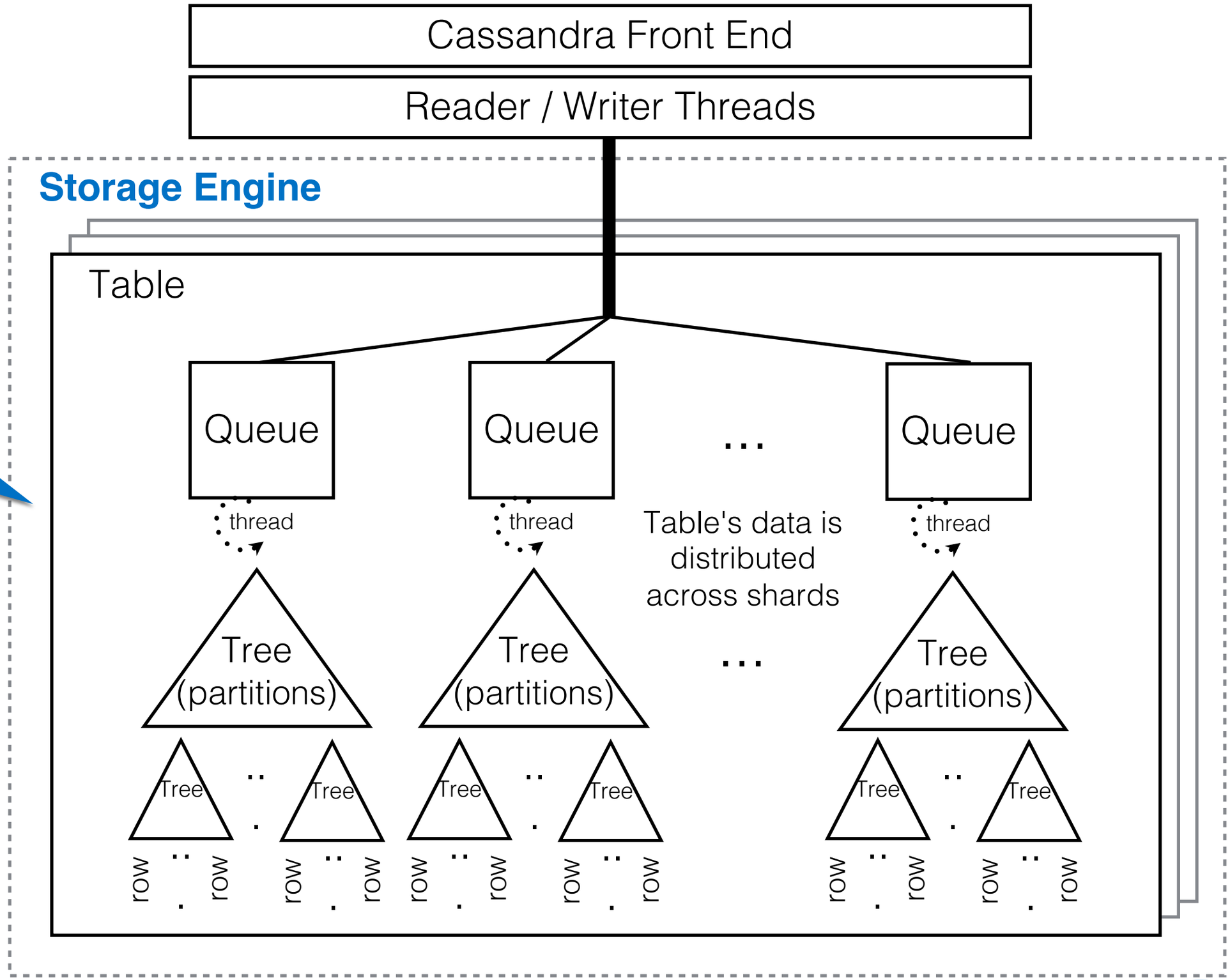
Three kinds of LLPL heaps:

- Heap: most flexible but both errors are possible
- PersistentHeap: durable writes are flushed -- if code compiles error #1 is not present
- TransactionalHeap: all writes are transactional -- if code compiles neither error is present

Example Use: Cassandra Pmem Storage Engine

https://github.com/intel/cassandra-pmem

All persistent data structures implemented using LLPL



△ = Adaptive Radix Tree: https://db.in.tum.de/%7Eleis/papers/ART.pdf

Links

Intel® Optane™ DC persistent memory

https://www.intel.com/content/www/us/en/products/memory-storage/optane-dc-persistent-memory.html

Low Level Persistence Library (LLPL)

https://github.com/pmem/llpl

Java bindings to PMDK pmemkv library

https://github.com/pmem/pmemkv-java

Persistent Memory Development Kit (PMDK)

https://github.com/pmem/pmdk

JEP 393 - java.foreign Memory Access API

https://openjdk.java.net/jeps/393

JEP 352 -- Non-Volatile Mapped Byte Buffers

https://openjdk.java.net/jeps/352

JEP 316 -- Allocation of Java Heap on Alt. Memory Devices

https://openjdk.java.net/jeps/316

Cassandra persistent memory storage engine

https://github.com/intel/cassandra-pmem

Persistent Collections for Java (PCJ) [Experimental]

https://github.com/pmem/pcj