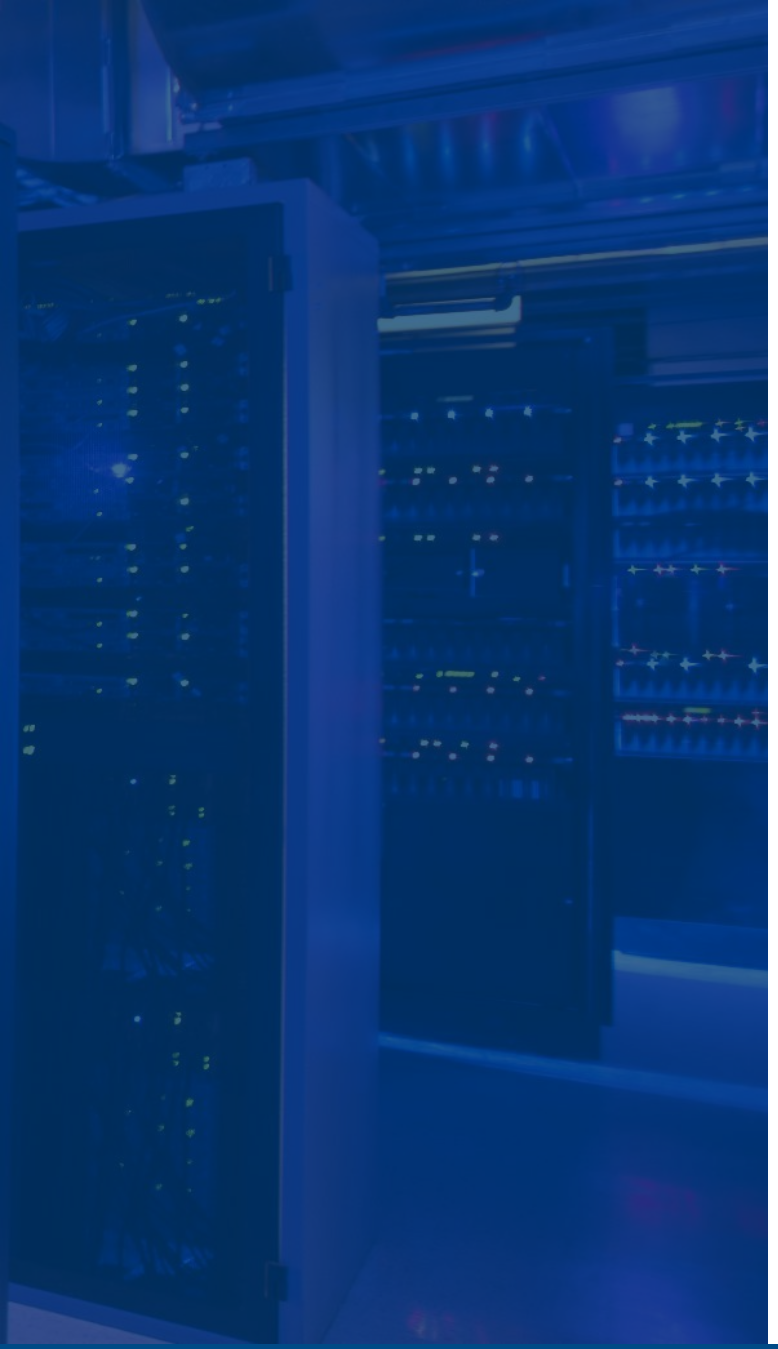
Building Java Solutions with Persistent Memory

Olasoji Denloye
Software Engineer
Intel

SPDK, PMDK, Intel® Performance Analyzers | Virtual Forum



Agenda

1. Motivation
2. LLPL Overview
3. A Simple Array
   Sample Code
4. Future Work

Motivation

This is a sample text. Insert your desired text here.

Why Java

Java is a popular language for building data center applications such as databases

Why Persistent Memory

Persistent memory offers new ways to program with long-lived data

Why LLPL

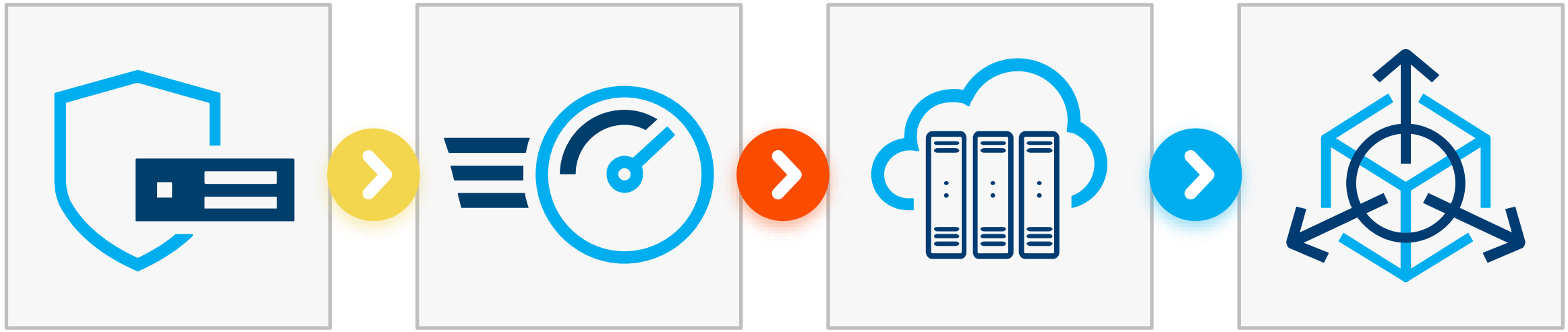
Enables Persistent Memory programming in Java

LLPL Overview

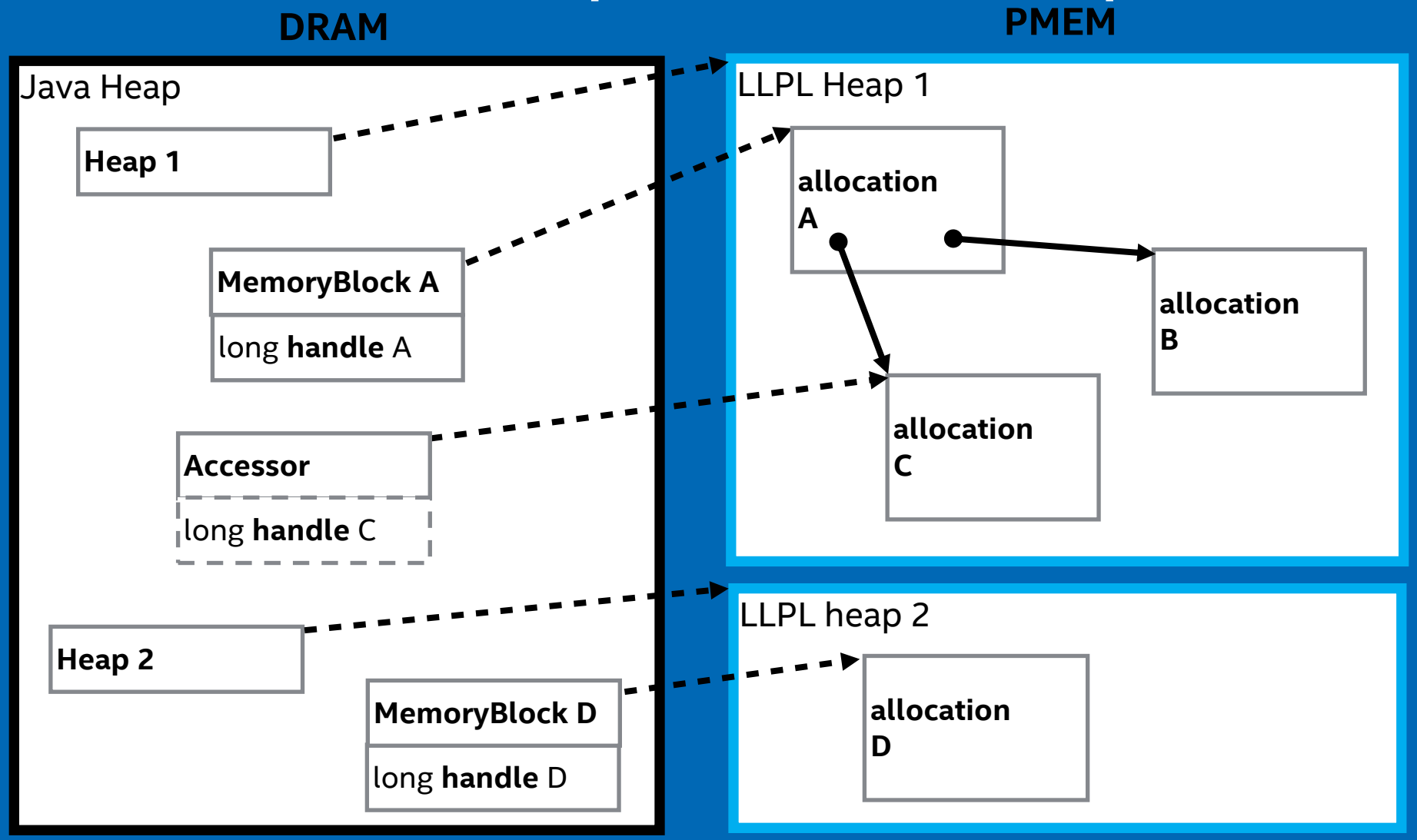
This is a sample text. Insert your desired text here.

Low-Level Persistence Library (LLPL)

- Intel open-source Java library for persistent memory programming
- A component of the Persistent Memory Development Kit (PMDK)



Java Heap vs LLPL Heap



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

3 kinds of Heaps and writes

TRANSACTIONAL

DURABLE

VOLATILE

TRANSACTIONAL HEAP

PERSISTENT HEAP

HEAP

A Simple Array

Sample Code

```
public class IntArray {
  private static final long SIZE_OFFSET = 0;
  private static final long HEADER_SIZE = 8;
  private final TransactionalMemoryBlock arrayBlock;

  public IntArray(TransactionalHeap heap, long size) {
    this.arrayBlock = heap.allocateMemoryBlock(HEADER_SIZE + size * Integer.BYTES);
    arrayBlock.setLong(SIZE_OFFSET, size);
  }

  public static IntArray fromHandle(TransactionalHeap heap, long handle) {
    TransactionalMemoryBlock arrayBlock = heap.memoryBlockFromHandle(handle);
    return new IntArray(arrayBlock);
  }

  private IntArray(TransactionalMemoryBlock arrayBlock) {
    this.arrayBlock = arrayBlock;
  }

  public void set(long index, int value) {
    arrayBlock.setInt(HEADER_SIZE + Integer.BYTES * index, value);
  }

  public int get(long index) {
    return arrayBlock.getInt(HEADER_SIZE + Integer.BYTES * index);
  }

  public long size() {
    return arrayBlock.getLong(SIZE_OFFSET);
  }

  public long handle() {
    return arrayBlock.handle();
  }

  public void free() {
    arrayBlock.free();
  }
}
```

```
import com.intel.pmem.llpl.Transaction;
import com.intel.pmem.llpl.TransactionalHeap;
import com.intel.pmem.llpl.TransactionalMemoryBlock;
import workshop.util.Util;

public class IntArrayExample{
   public static void main(String[] args) {
      String heapName = Util.pmemHome() + "A_intarray";
      long heapSize = 20_000_000L;
      boolean firstRun = !TransactionalHeap.exists(heapName);
      TransactionalHeap heap = firstRun
                        ? TransactionalHeap.createHeap(heapName, heapSize)
                        : TransactionalHeap.openHeap(heapName);
      if (firstRun) {
         long size = 10;
         System.out.println("A) Creating New Array of size " + size);
         Transaction.create(heap, ()-> {
            IntArray array = new IntArray(heap, size);
            heap.setRoot(array.handle());
            array.set(5, 10);
            array.set(7, 20);
         });
      }
      else {
         IntArray array = IntArray.fromHandle(heap, heap.getRoot());
         System.out.println("A) Retrieved IntArray of size " + array.size());
         for (long i = 0; i < array.size(); i++) {
            int val = array.get(i);
            System.out.println("  IntArray[" + i + "] = " + val);
         }
      }
   }
}
```

Notes

- Heap Provisioning
- Root object
- Transaction
- Reconstruction

Future Work

New Features

- Memory Pools
- Production Quality Data structures

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


intel®
SPDK, PMDK, Intel® Performance Analyzers
Virtual Forum

BACKUP

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