ACCELERATE YOUR SPARK USING INTEL® OPTANE™ DC PERSISTENT MEMORY

Cheng Xu, Intel

cheng.a.xu@intel.com

Nov, 2018

Legal Disclaimer & Optimization Notice

AGENDA

Challenges in Data Analytics

Intel® Optane™ DC Persistent Memory (DCPMM)

Introduction to PMDK library

Spark DCPMM optimizations

Future And Other DCPMM Optimization Work

A long time ago in a galaxy far, far away...

Let’s talk about Storage

Let’s talk about Memory

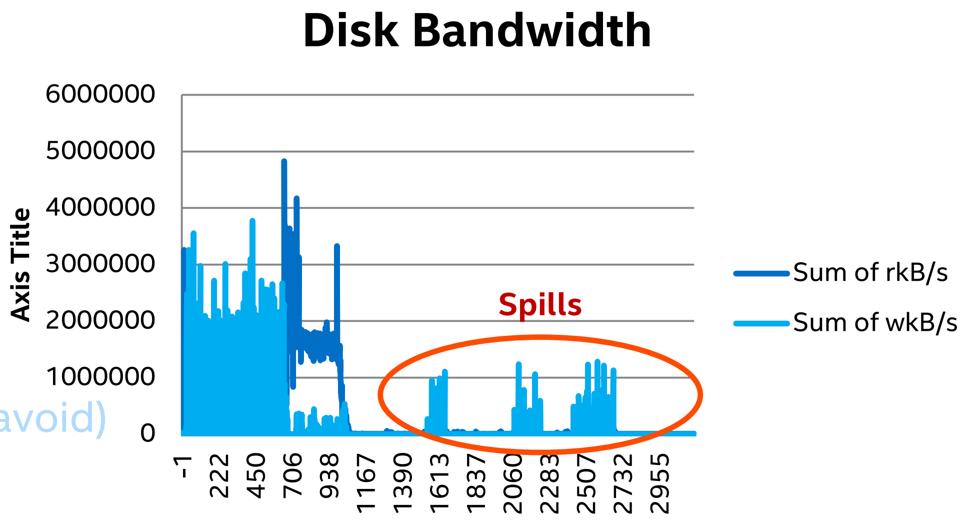
CHALLENGES IN DATA ANALYTICS

Data Analytic is MEMORY sensitive because it's critical for

- **Performance :** Spark SQL is already blazing fast but sometimes

- Memory bound (e.g. Unnecessary spills for Q67* - Data skew)

- Or even faster with extra DRAM as I/O cache (e.g. I/O cache for Q44* or avoid)



Summary Metrics for 2592 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	2 ms	6 ms	8 ms	0.1 s	36 min
Scheduler Delay	7 ms	0.3 s	0.4 s	0.6 s	0.8 s
Task Deserialization Time	2 ms	5 ms	7 ms	95 ms	0.2 s
GC Time	0 ms	0 ms	0 ms	0 ms	4.1 min
Result Serialization Time	0 ms	0 ms	1 ms	1 ms	14 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	64.0 KB	64.0 KB	64.0 KB	64.0 KB	14.1 GB
Shuffle Read Blocked Time	0 ms	0 ms	0 ms	0 ms	0.5 s
Shuffle Read Size / Records	0.0 B / 0	0.0 B / 0	0.0 B / 0	0.0 B / 0	11.8 GB / 192677917
Shuffle Remote Reads	0.0 B	0.0 B	0.0 B	0.0 B	11.7 GB
Shuffle spill (memory)	0.0 B	0.0 B	0.0 B	0.0 B	65.8 GB
Shuffle spill (disk)	0.0 B	0.0 B	0.0 B	0.0 B	19.4 GB

TPC-DS 30 TB SF, 10 nodes cluster, 64 CPU cores, 384GB DRAM, 4 NVMe SSD)

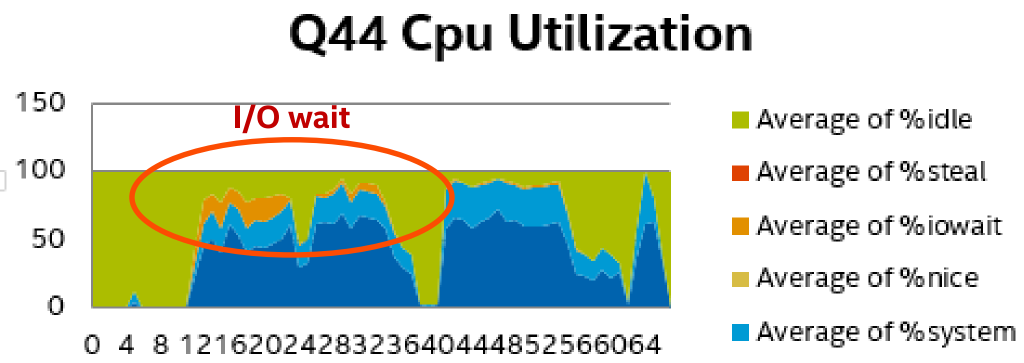
CHALLENGES IN DATA ANALYTICS

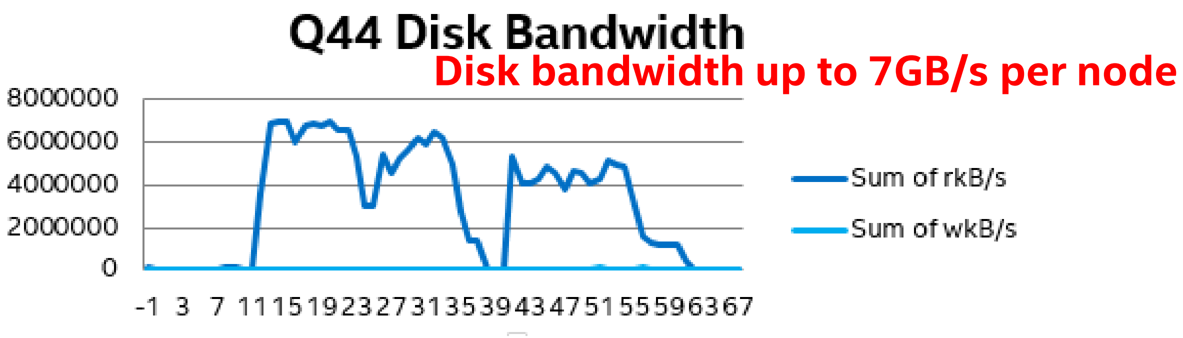
Data Analytic is MEMORY sensitive because it's critical for

- **Performance :** Spark SQL is already blazing fast but sometimes

- Memory bound (e.g. Unnecessary spills for Q67* - Data skew)

- Or even faster with extra DRAM as I/O cache (e.g. I/O cache for Q44*)





TPC-DS 30 TB SF, 10 nodes cluster, 64 CPU cores, 384GB DRAM, 4 NVMe SSD)

CHALLENGES IN DATA ANALYTICS

Data Analytics is facing **DILEMMA** and tradeoff for

- **Performance VS. Durable:**

  - Checkpoint for iterative computation (e.g. Persisted checkpoint in Spark)
  - Or Recovery Log Flush Frequency (e.g. Kafka recovery log flush frequency)

  → Persistent! Performance!

- **Scale out VS. Scale up:**

  - Better TCO
  - Extra cost for scale out, sometimes lower utilization

  → Cheaper!

- **On heap VS. off heap**

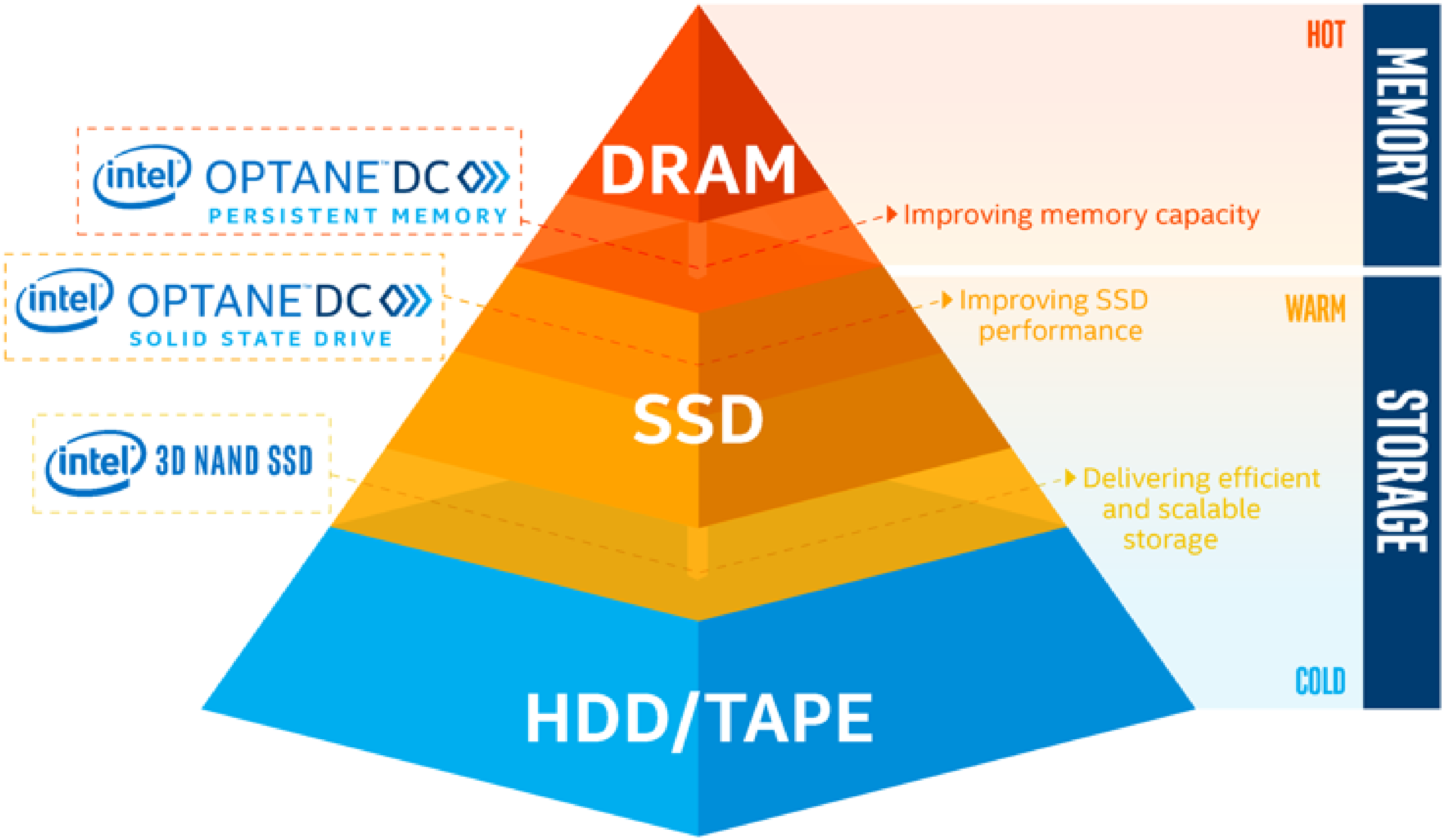
  - GC VS. Self-managed memory

  → Easy to use!

TPC-DS 30 TB SF, 10 nodes cluster, 64 CPU cores, 384GB DRAM, 4 NVMe SSD)

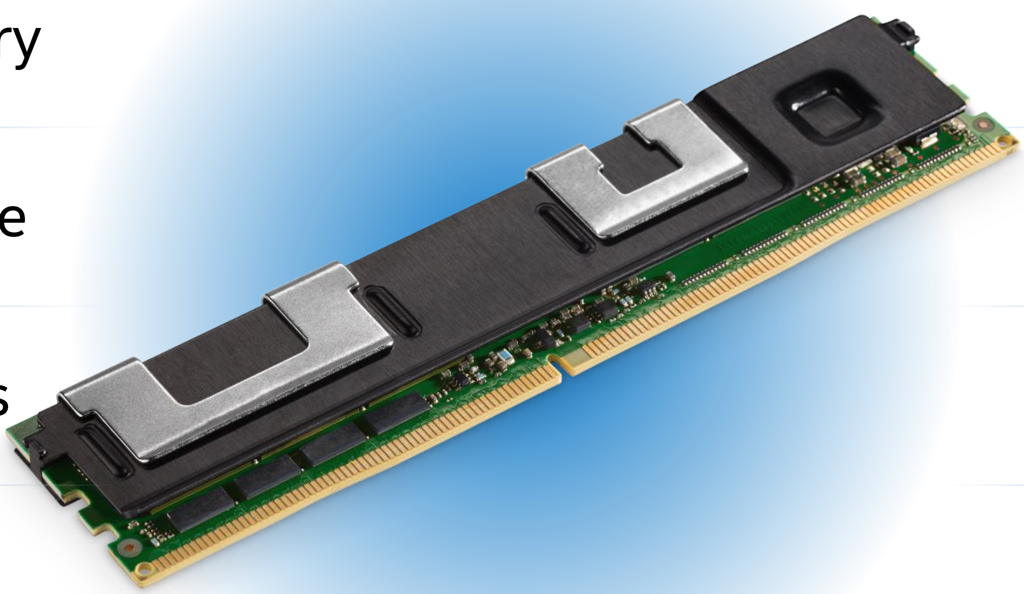
Answer?

REIMAGINING THE DATA CENTER MEMORY AND STORAGE HIERARCHY



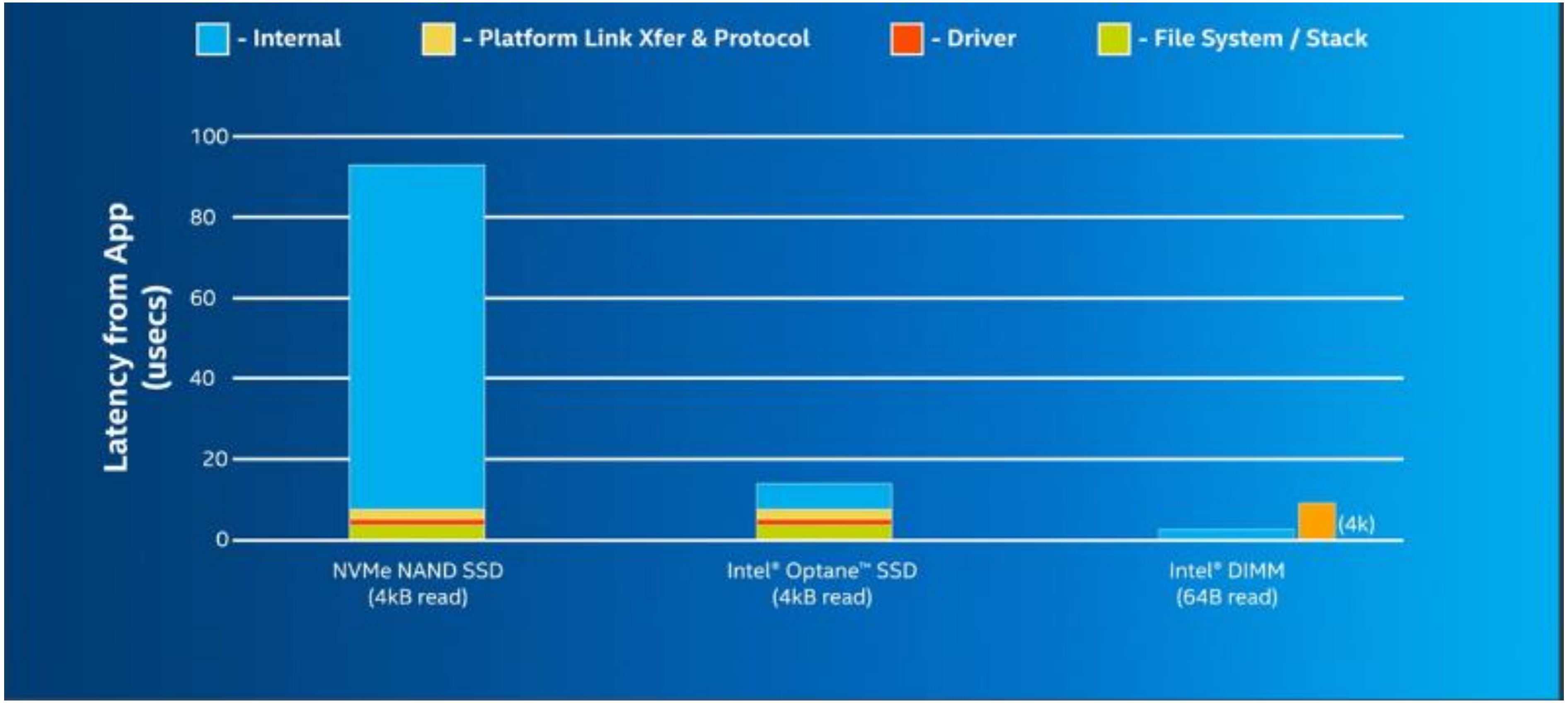
INTEL OPTANE DC PERSISTENT MEMORY

- Big and Affordable Memory
- High Performance Storage
- Direct Load/Store Access
- Native Persistence



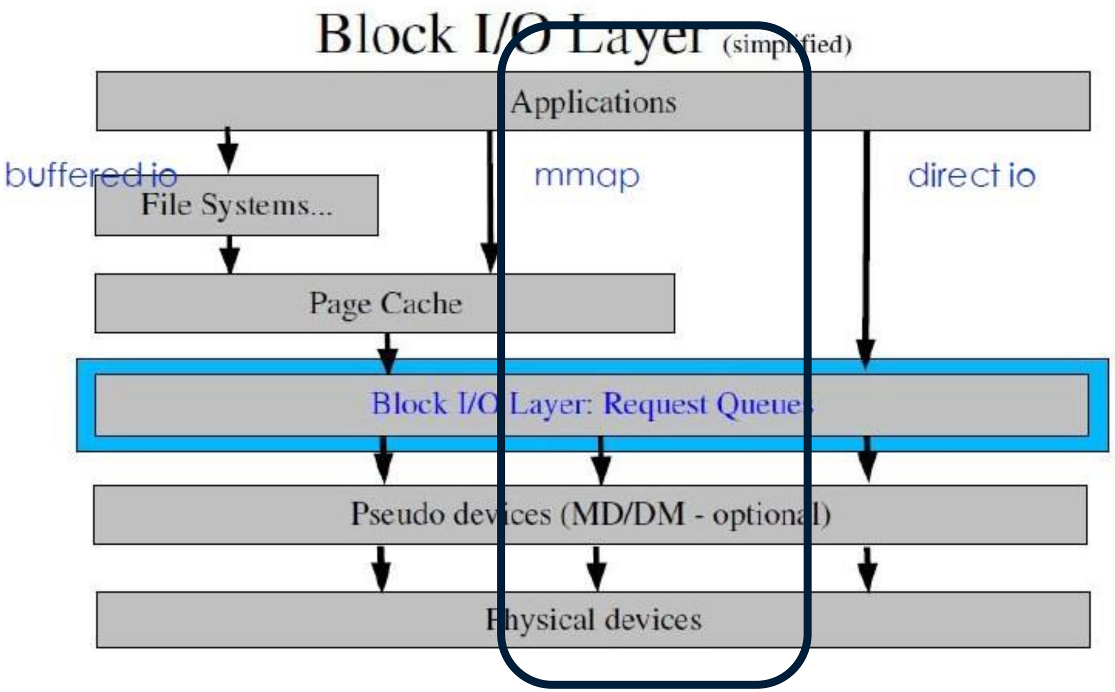
- 128, 256, 512GB
- DDR4 Pin Compatible
- Hardware Encryption
- High Reliability

Performance of DCPMM vs. NAND

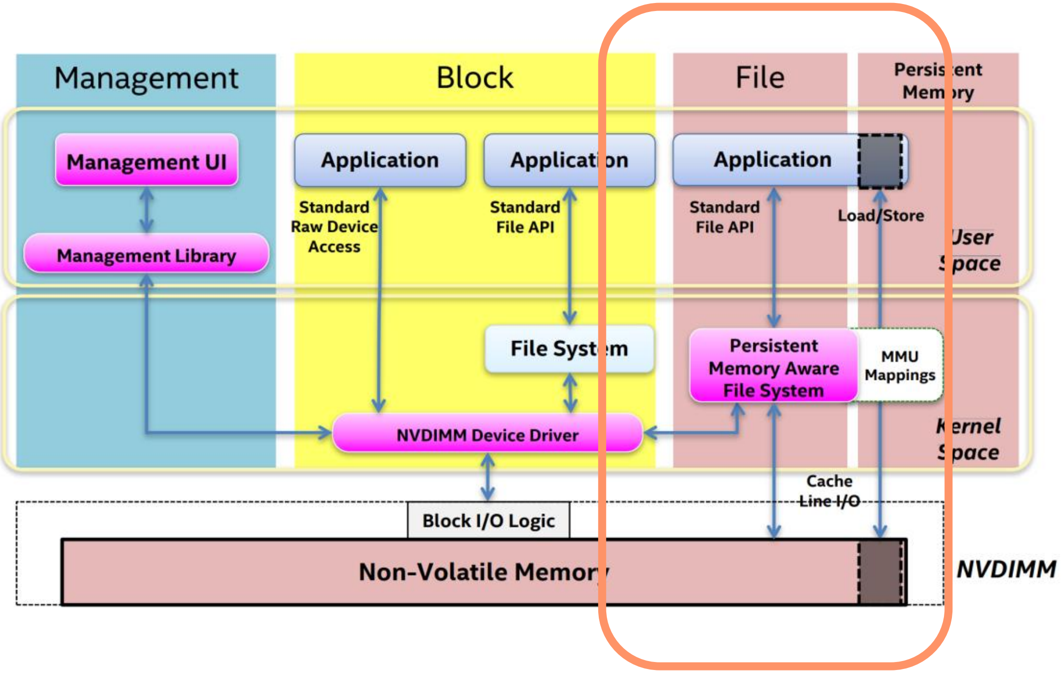


Then how to use it?

CONCEPT - DAX



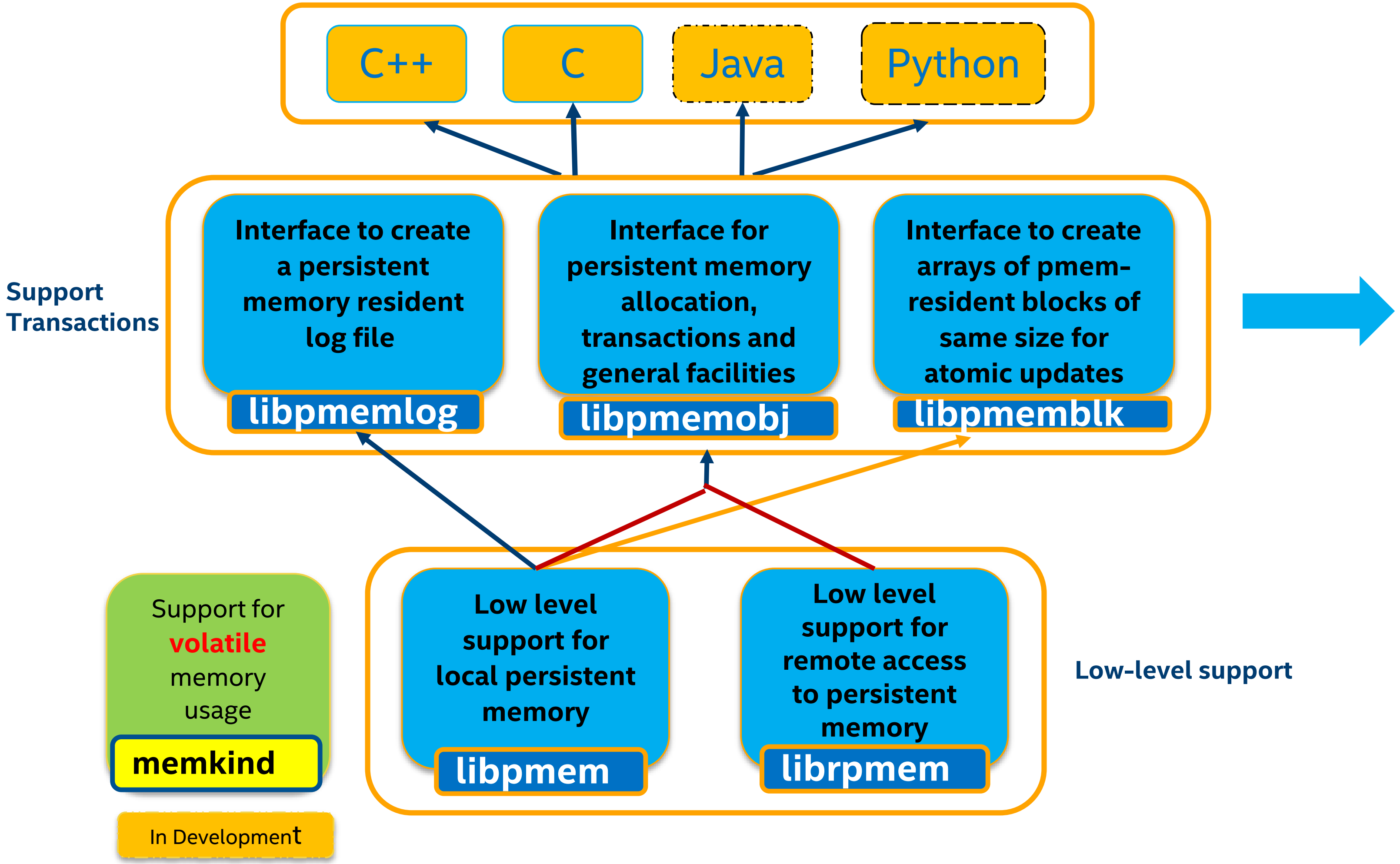
Normal I/O Path

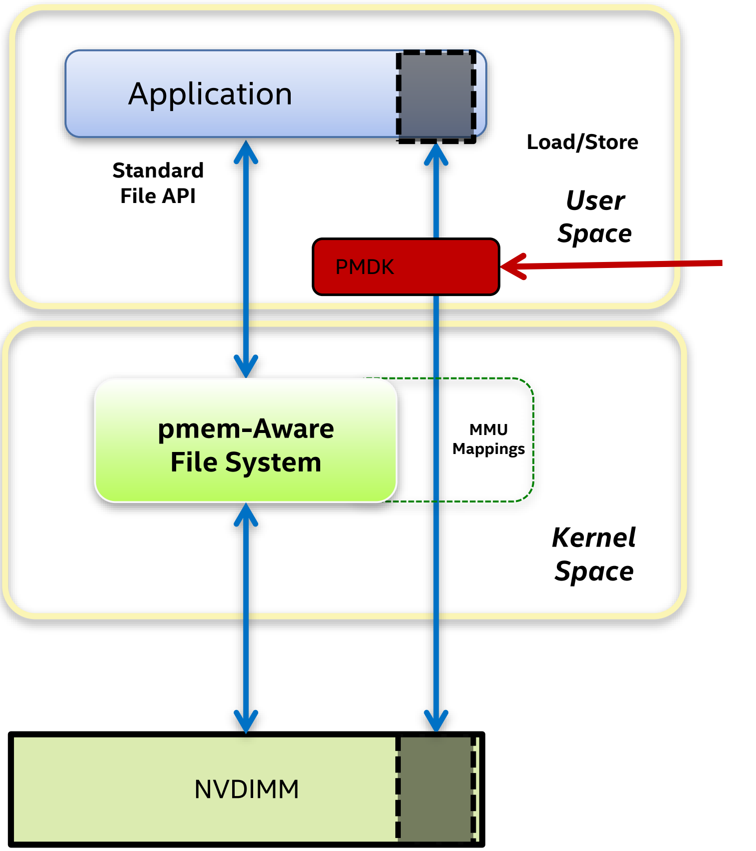


PMDK I/O Path

PMDK : A SUITE OF OPEN SOURCE OF LIBRARIES

Link to Open Source :
http://pmem.io/PMDK/





Link to Intel Developer Zone:
https://software.intel.com/en-us/persistent-memory

MEMKIND LIBRARY

- **Memkind** supports the traditional *malloc/free* interfaces on a memory mapped file

- Use persistent memory as volatile memory

- Old name was libmem

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <libvmem.h>

int
main(int argc, char *argv[])
{
VMEM *vmp;
char *ptr;

/* create minimum size pool of memory */
if ((vmp = vmem_create("/pmem-fs",
VMEM_MIN_POOL)) == NULL) {
perror("vmem_create");
exit(1);
}

if ((ptr = vmem_malloc(vmp, 100)) == NULL) {

perror("vmem_malloc");
exit(1);
}

strcpy(ptr, "hello, world");

/* give the memory back */
vmem_free(vmp, ptr);

/* ... */
}
```

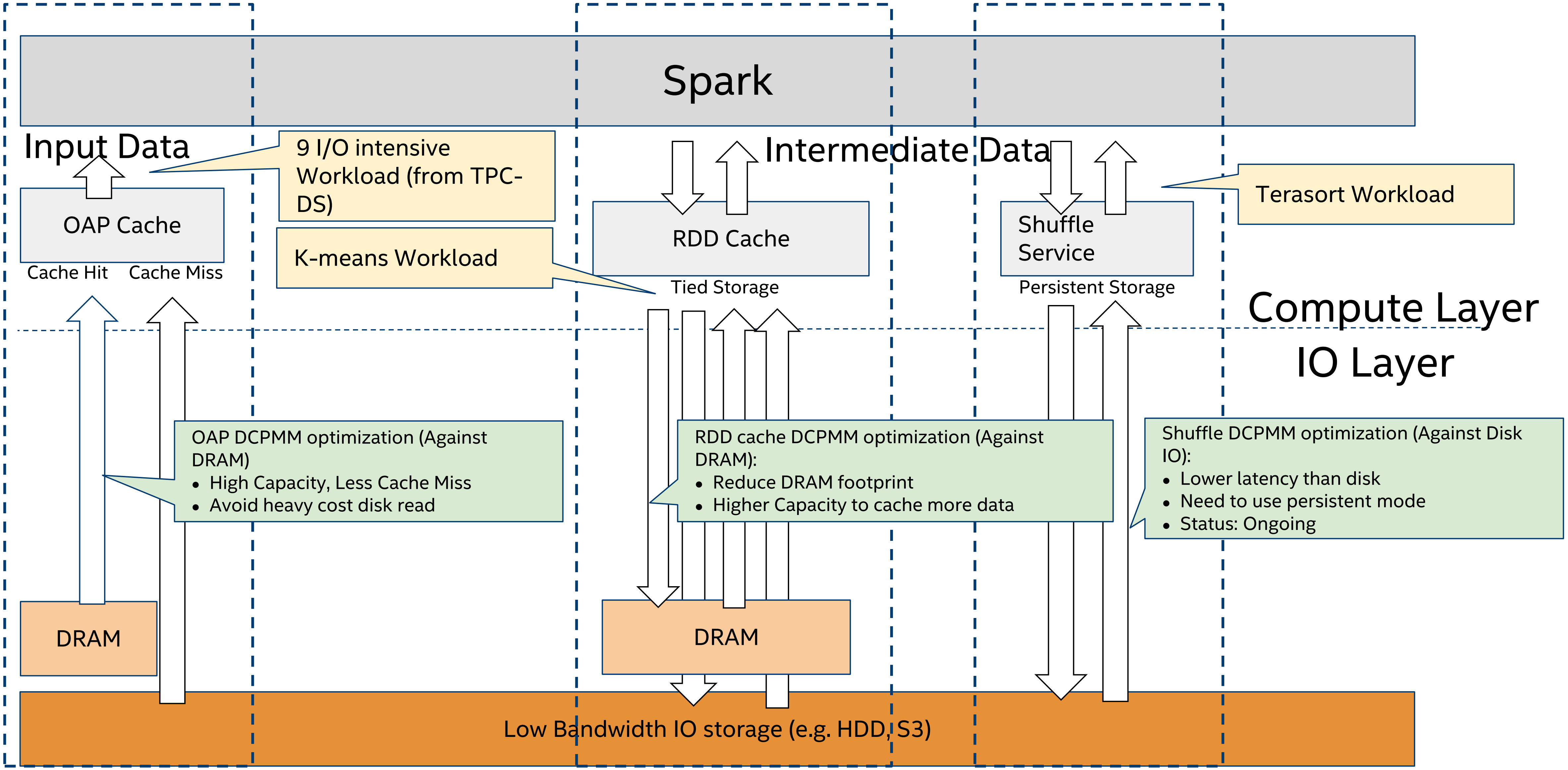
WHY PMDK

- Built on top of SNIA Programming Model
- Simplifies/Facilitates Persistent Memory Programming Adoption with Higher Level Language Support
  - C, C++, Java
    - No Changes to Compiler or Programming Language
  - Abstracts details about
    - Types of Flush commands supported by CPU
    - Size of Atomic Stores

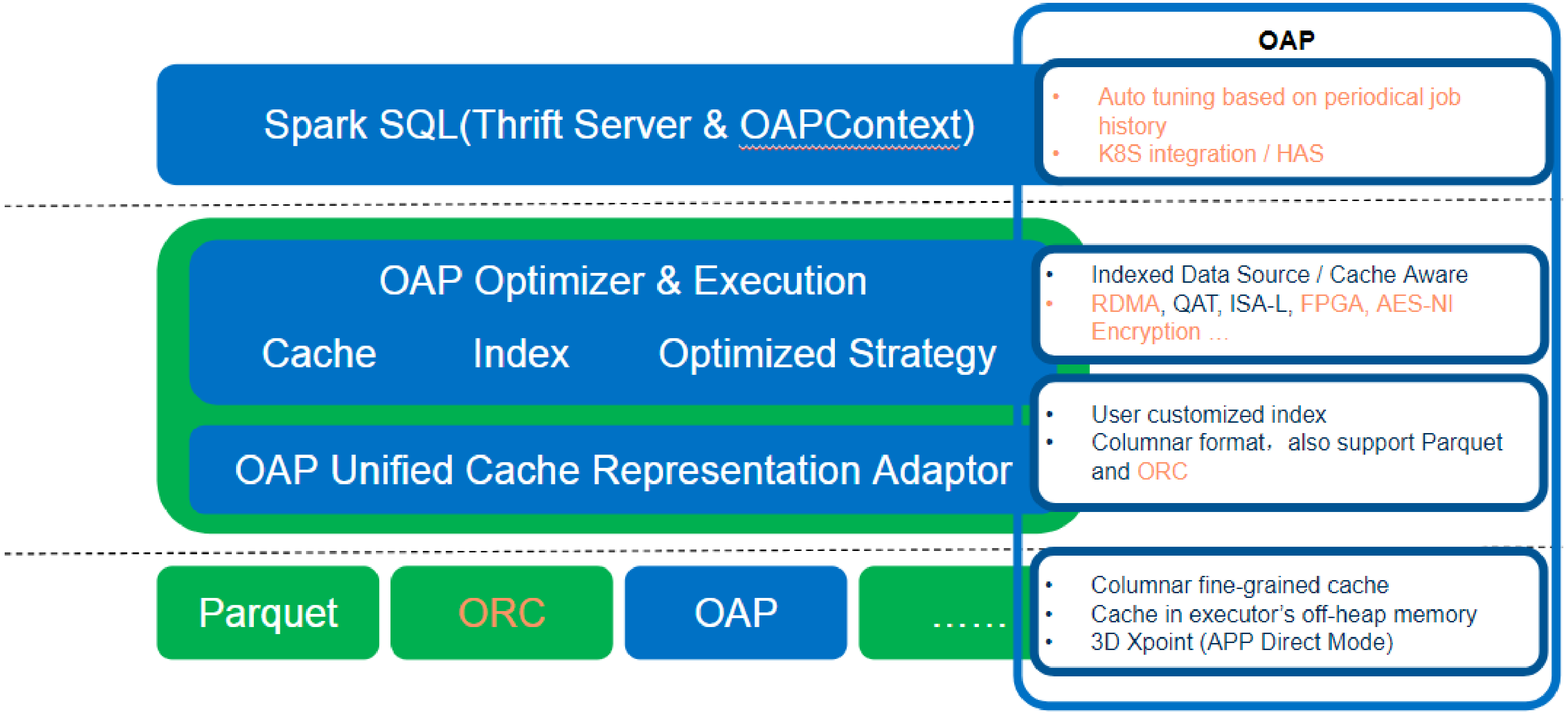
- Provides API to
  - Allocate/Manage Persistent Memory Pools
    - Uses memory-mapping
    - In-place update
  - Transactional Operations
    - Keeps Data Consistent and Durable during Application Crashes
      - Flushes processor caches
    - Power Fail Atomicity
  - Builds on DAX capabilities in both Linux and Windows

How About Spark?

Spark DCPMM Optimization Overview

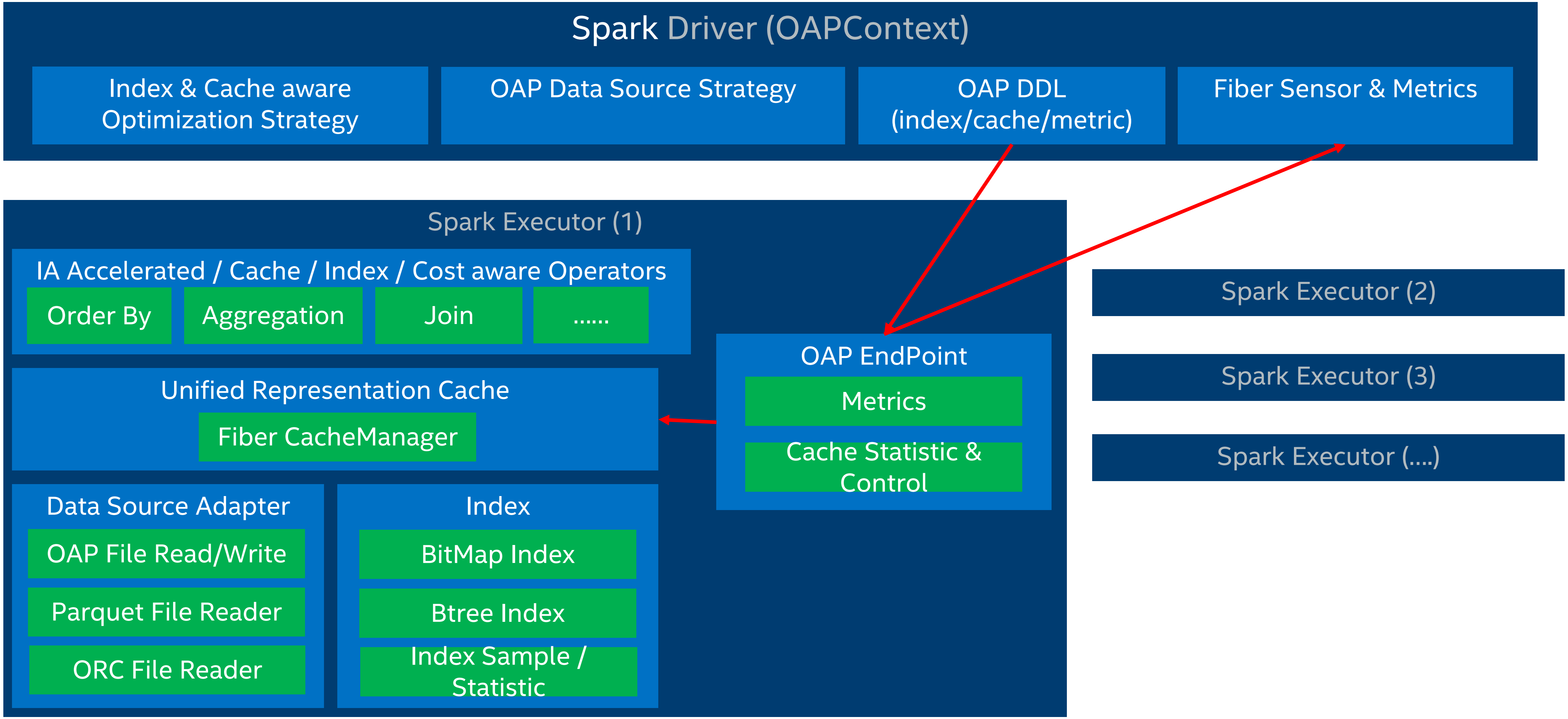


OAP Overview

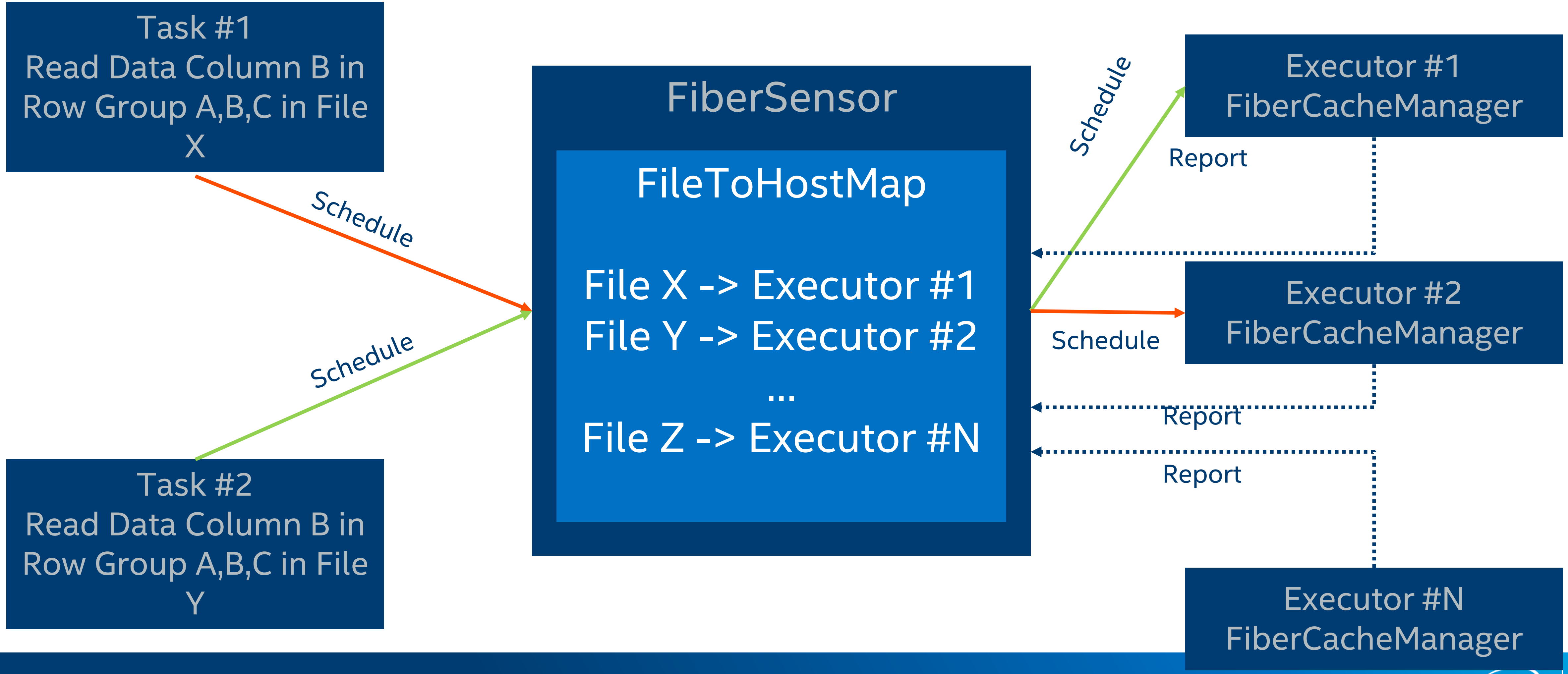


- Collaborating with Baidu, Intel invented OAP in 2016 and open source in 2017
- OAP provides optimizations like cache and index to accelerate Spark SQL
- In Baidu's Phoenix Hive advertising system, based on a trillion daily clicks and ad effectiveness analysis. OAP raised query performance 5x compared with native Spark SQL.

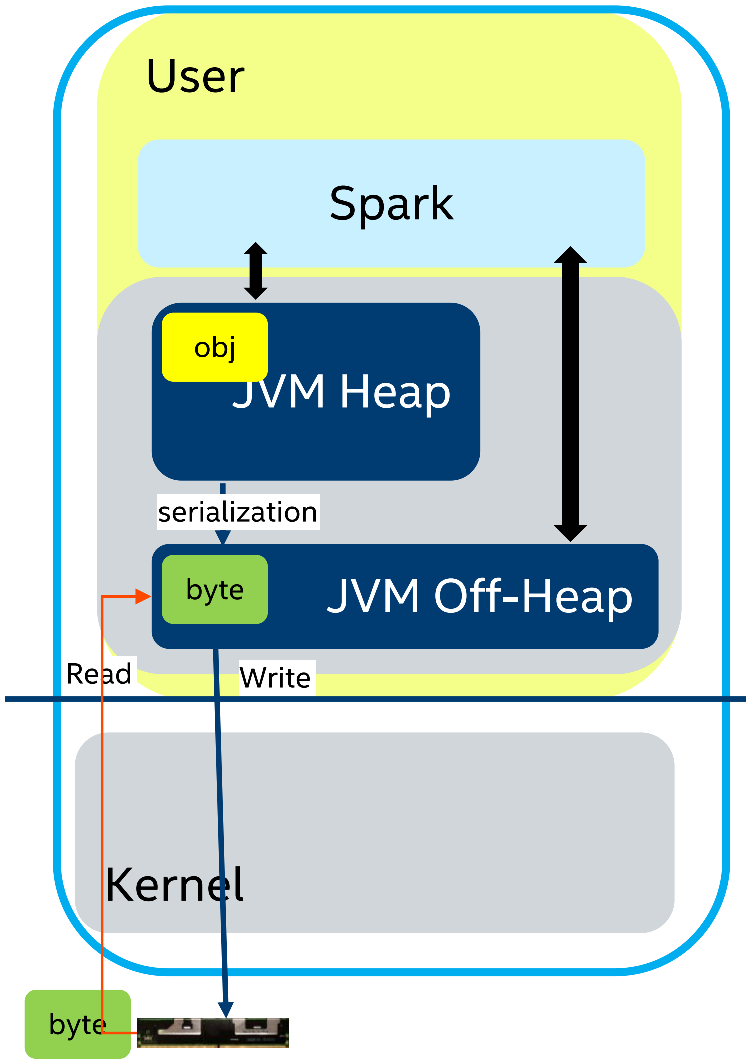
OAP Architecture

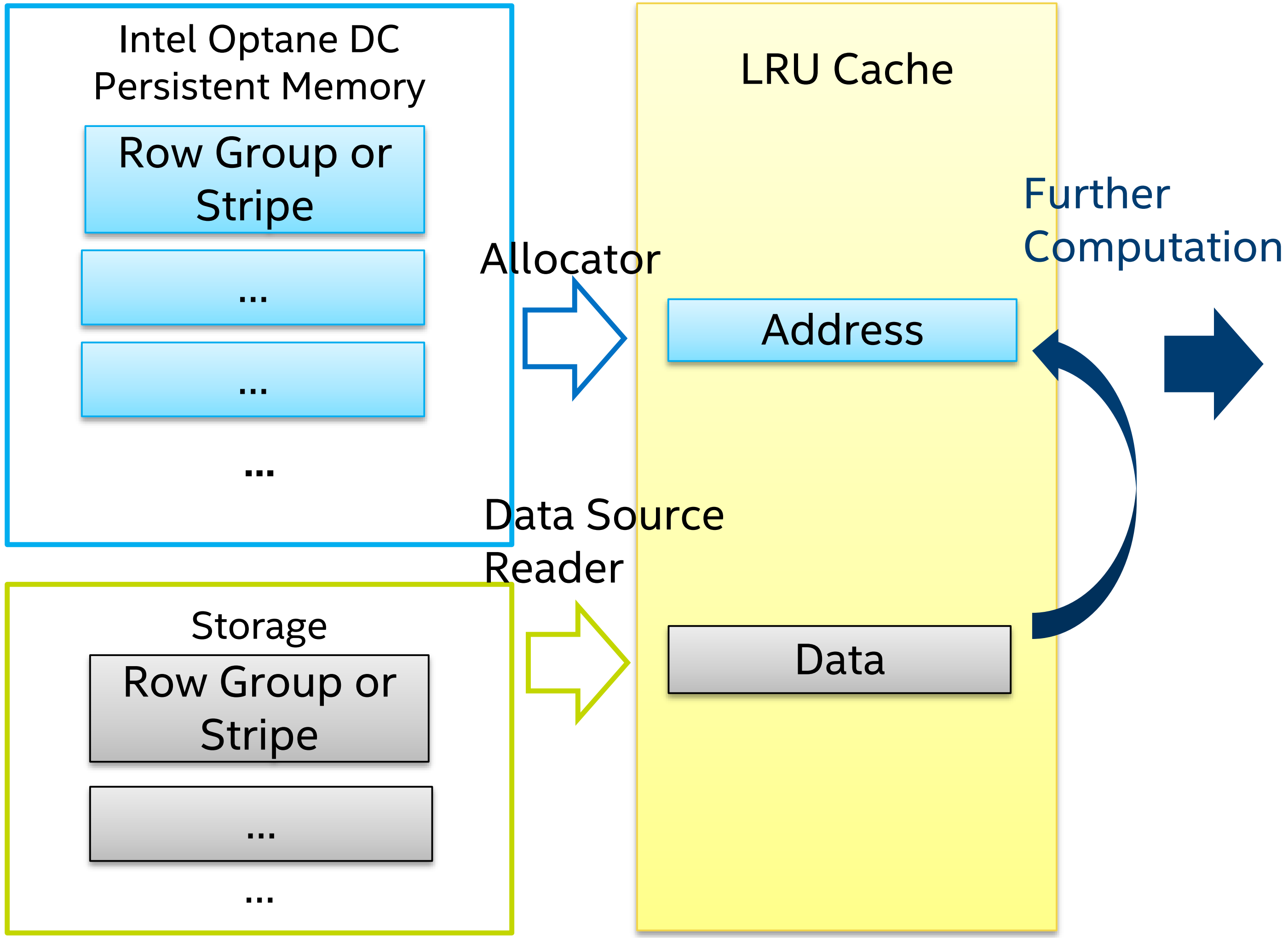


OAP Components – Cache (Report & Schedule)

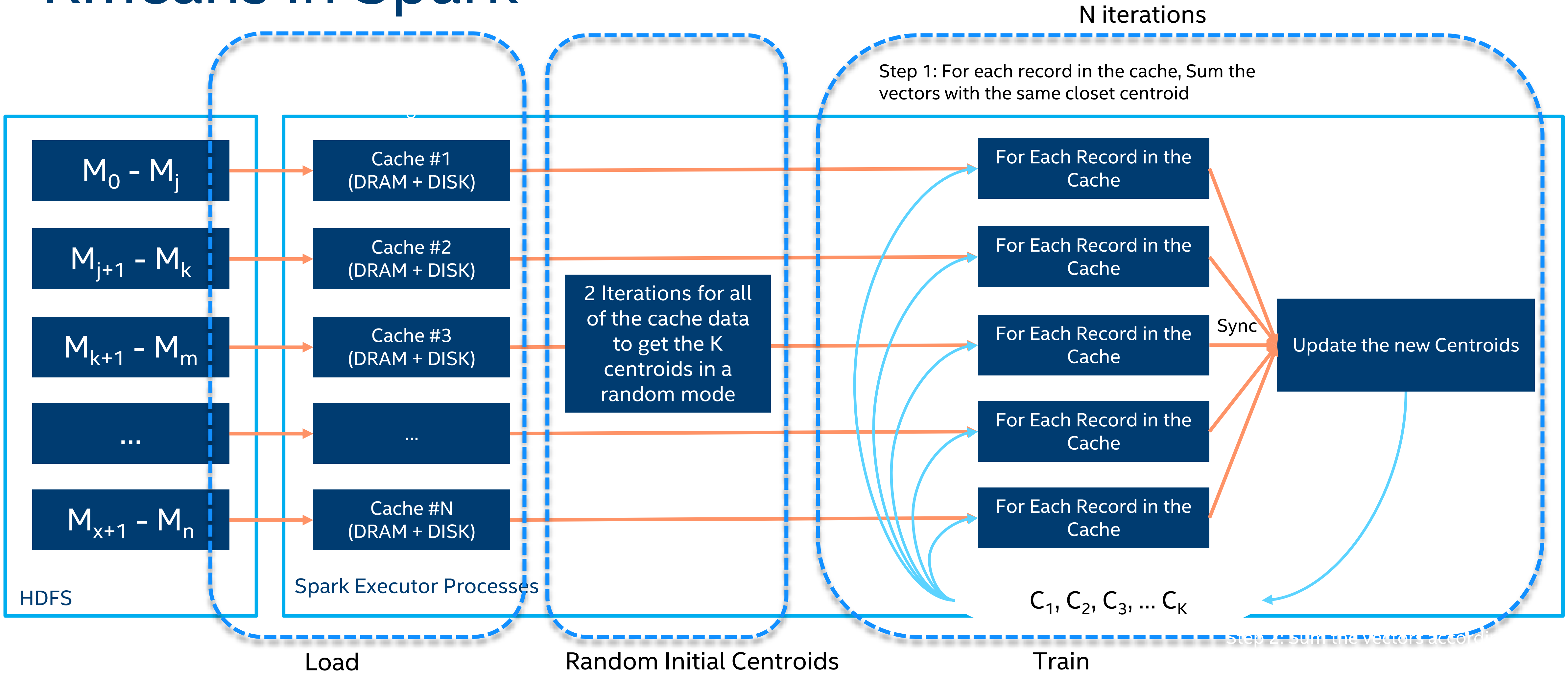


DCPMM Enabling For OAP I/O Cache



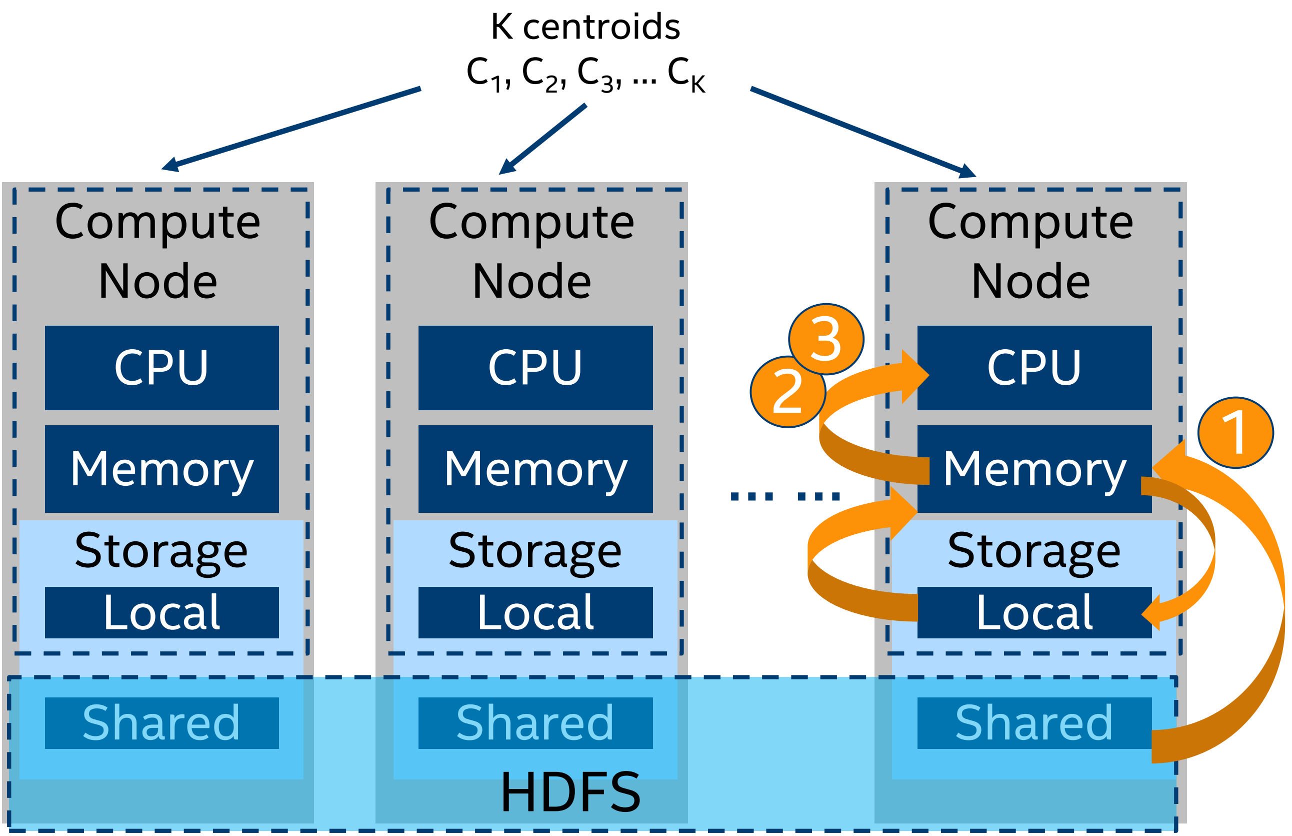


Kmeans in Spark



Load the data from HDFS to DRAM (and AEP / SSD if DRAM cannot hold all of the data)(**Load**), after that, the data will not be changed, and will be iterated repeatedly in **Initialization** and **Train** stages.

Kmeans Basic Data Flow



1 Load

- Load data from HDFS to memory.
- Spill over to local storage

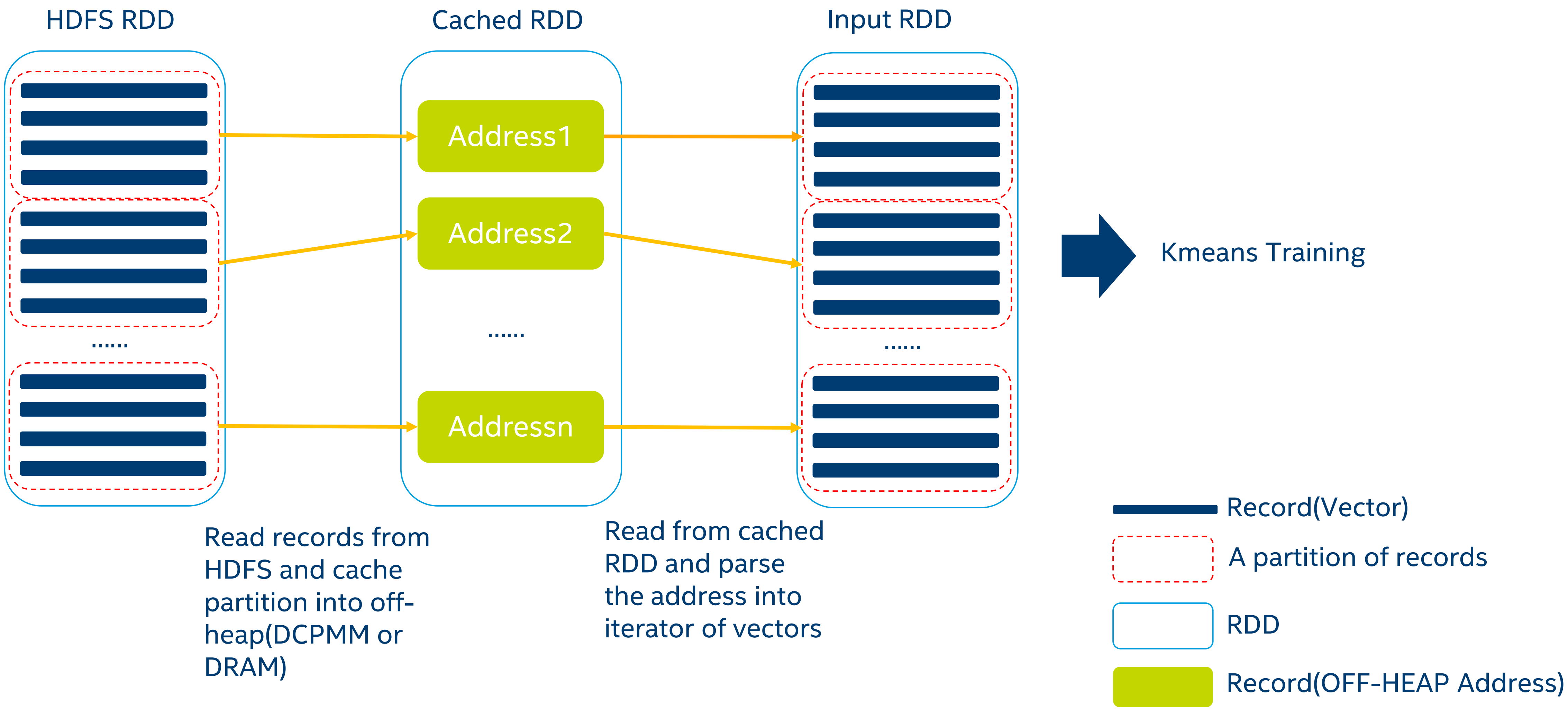
2 Initialization

- Compute using initial centroid based on data in memory or local storage

3 Train

- Compute iterations based on local data

Implement Details For DCPMM Enabling



Future Or Other related Work

Future or Other DCPMM Optimization Work

- Spark
  - Intel Optane DC persistent based checkpoint
  - Broadcast join
  - Push based shuffle DCPMM optimization
- Hadoop
  - DCPMM based cache
- HBase
  - WALess with DCPMM
  - Block cache
- Kudu
  - block cache using DCPMM