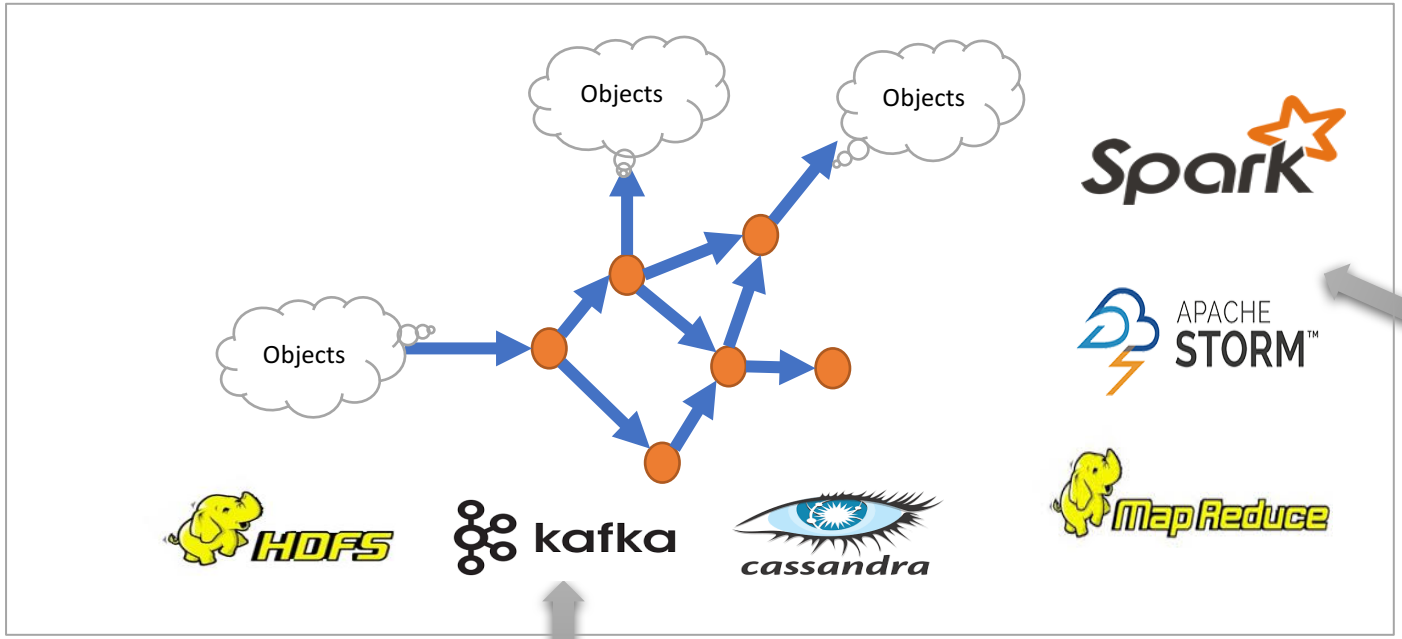
A GENERIC DURABLE OBJECT MODEL WITH DURABLE COMPUTING

Gary Wang – Intel
Johnu George, Debo Dutta – Cisco

CURRENT BIG DATA SYSTEM STACK

APACHE Incubating Mnemonic



GPU

TPU

FPGA

CPU

Compute

Storage

DRAM

3D Xpoint

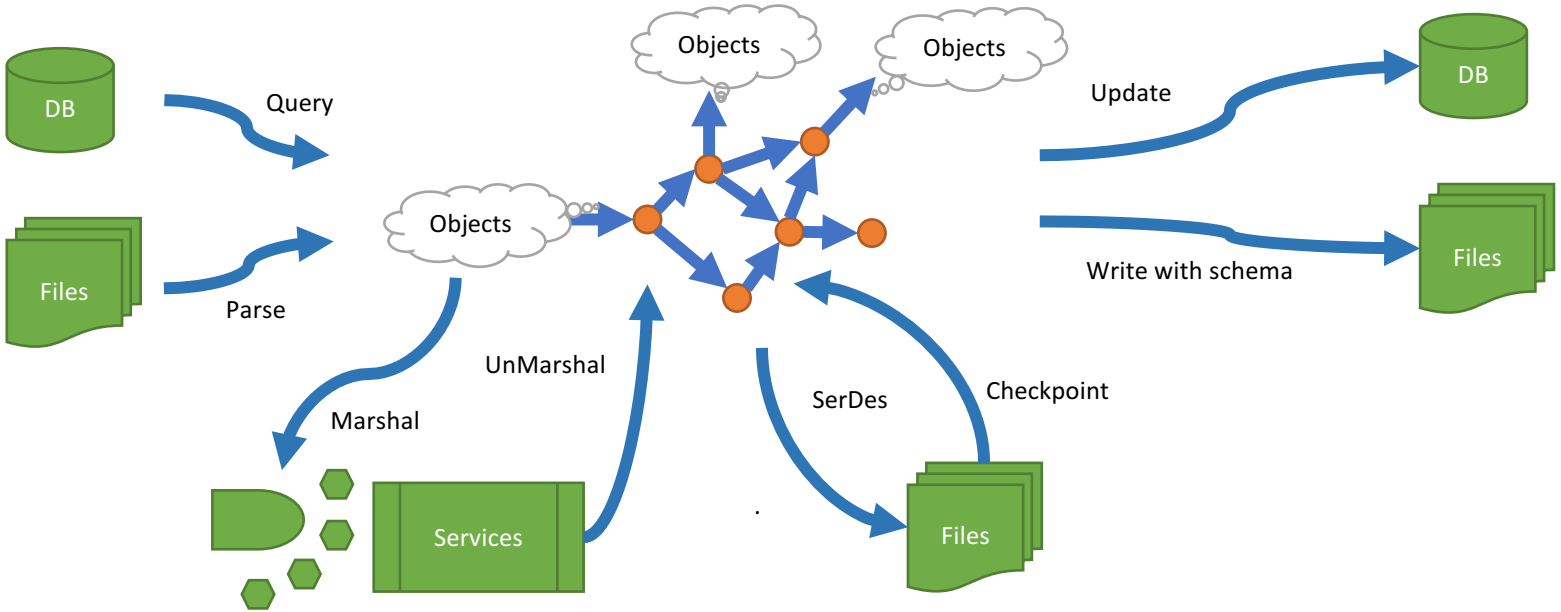
SSD

Questions?

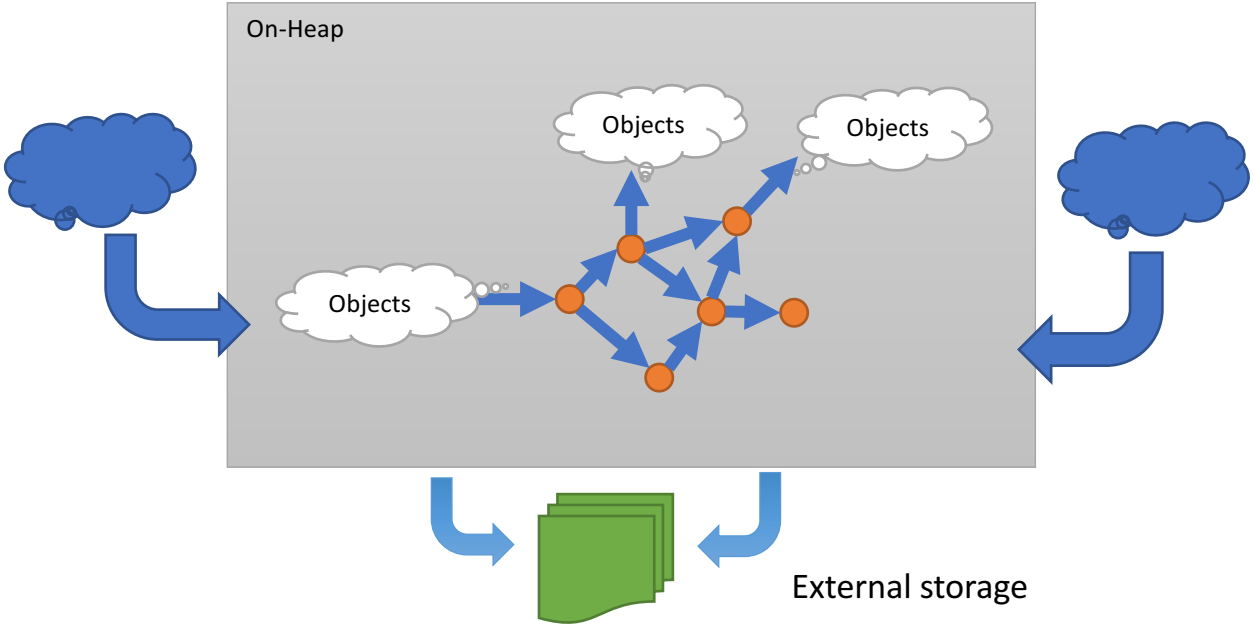
1. How to represent object graphs in software efficiently?

2. How to map software models to hardware efficiently?

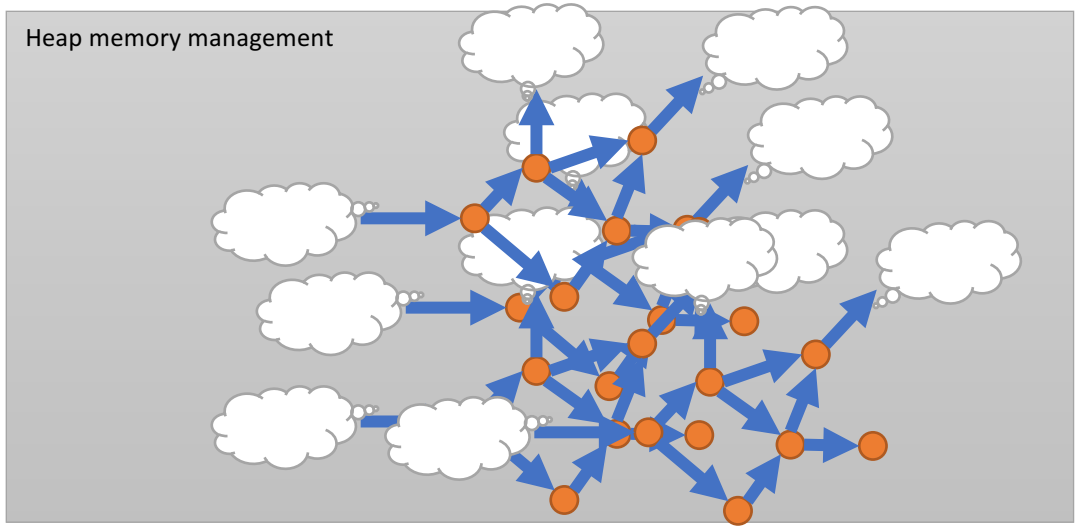
PROBLEM: OBJECT TRANSFORMATIONS -> HIGH SERDES COST



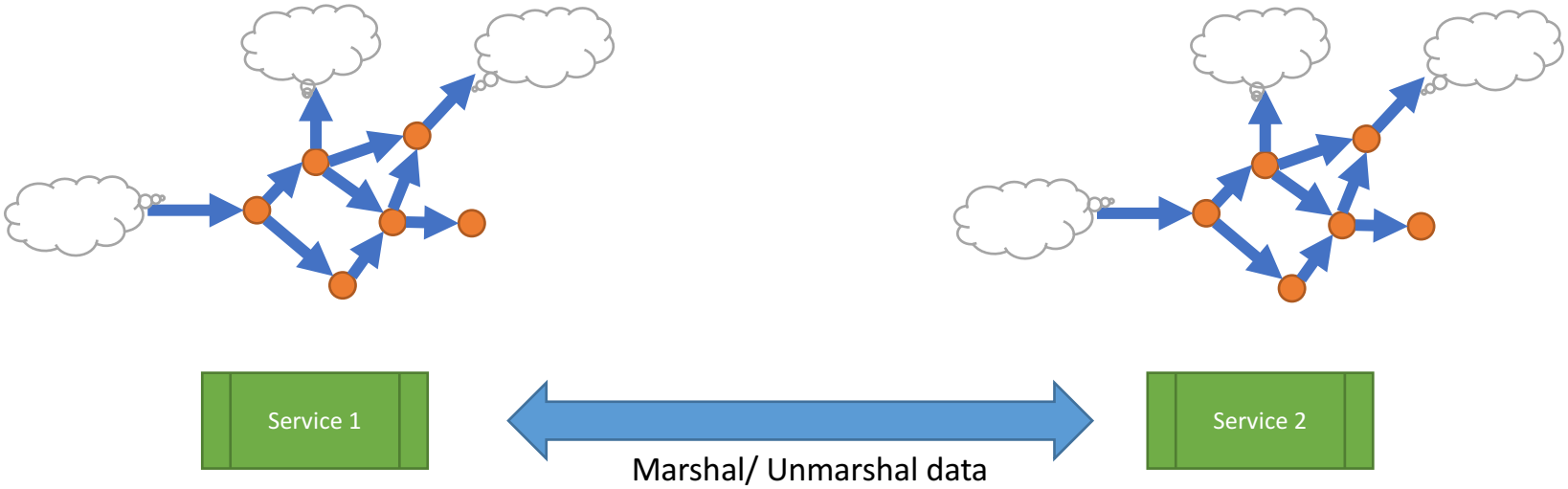
PROBLEM: LIMITED HEAP SPACE



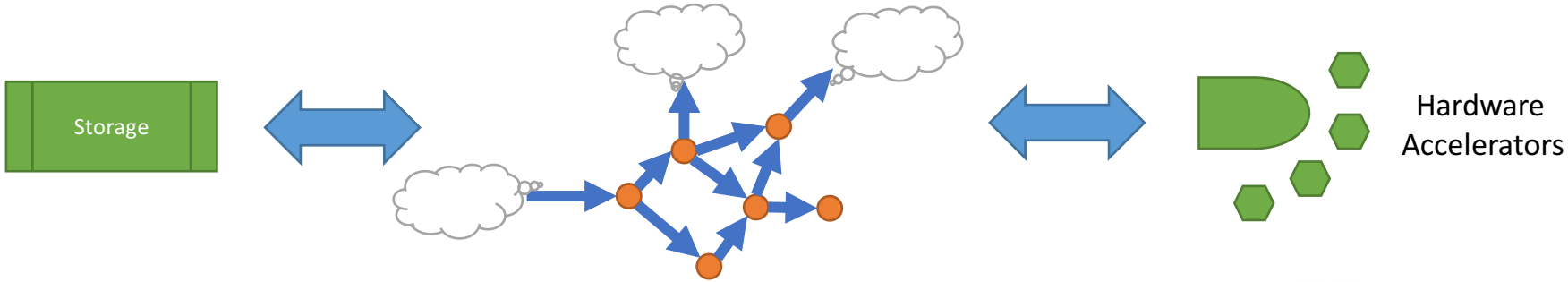
PROBLEM: HIGH MEMORY MANAGEMENT OVERHEADS

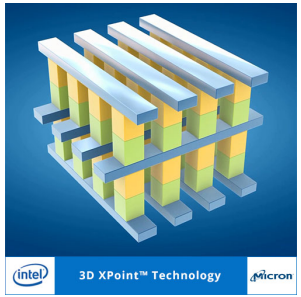


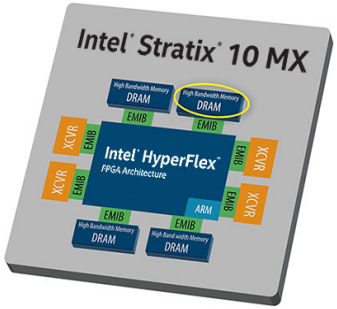
PROBLEM: HIGH COMMUNICATION OVERHEADS



SOLUTION?: LEVERAGE SMART HARDWARE







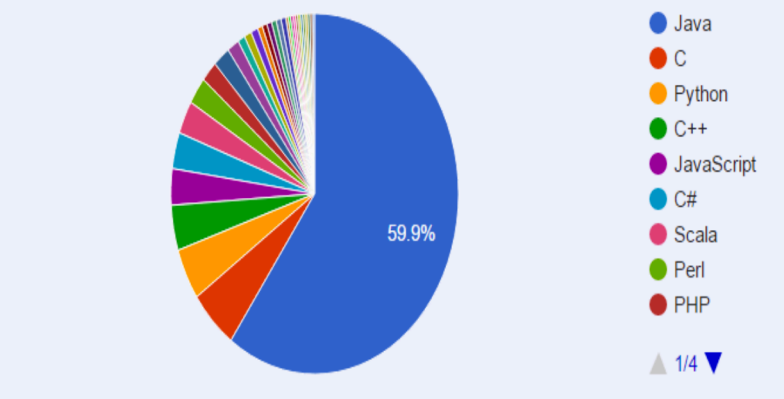
TLDR: CURRENT BIGDATA WORKLOADS

- JVM performance costs
  - Serialization/Deserialization
  - JNI invocation cost

- Complex
  - Own caching schemes
  - Own memory management schemes
  - Own transformation schemes

- Poor Performance of JVM based stacks
  - Higher memory usage
  - Frequent GC pauses

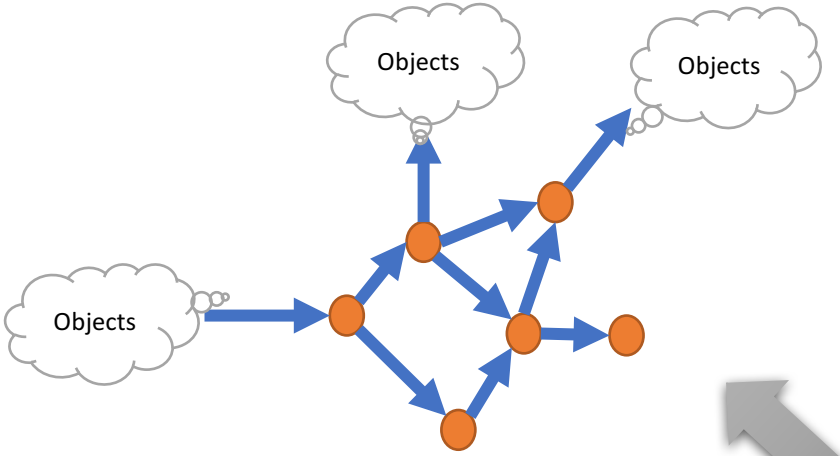
MNEMONIC: INTRODUCTION



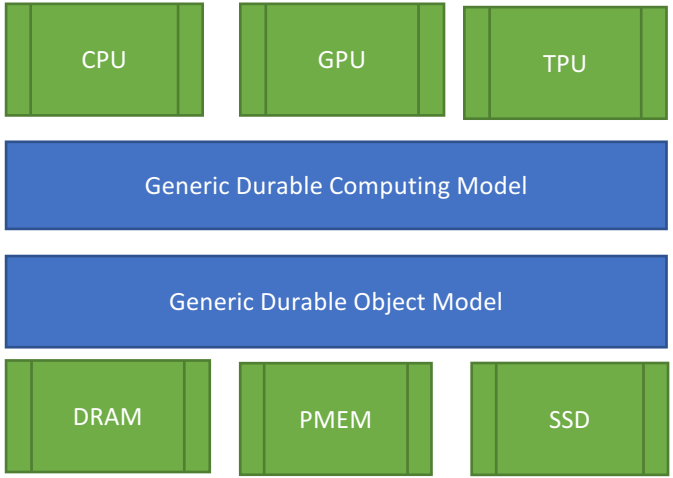
- Evolving new storage technologies – NVM, 3D Xpoint
  - Relative performance of system layers changed
  - Changes required in software programming models
- The most popular PL among Apache projects - Java
  - Approx. 60% of TLP are java based projects
- Need for a Java based non-volatile programming model that provides an unified view of storage technologies – DRAM, PMEM, SSD etc.

=> Mnemonic

WHAT IS MNEMONIC?



Object Graph



Compute

Storage

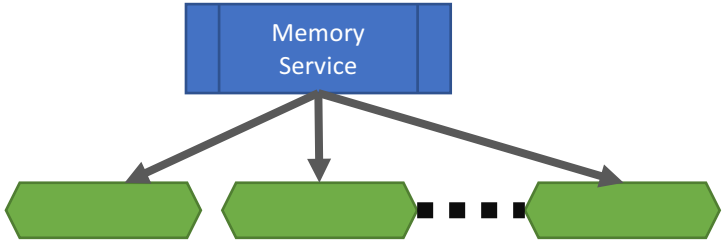
GENERIC DURABLE OBJECT MODEL

- Simple but not simplistic
- Generic
- Durable
- Auto-Reclaimable
- Lazily Restorable
- Memory Agnostic
- No SerDes
- Native Identifiable
- Optional Handler Store

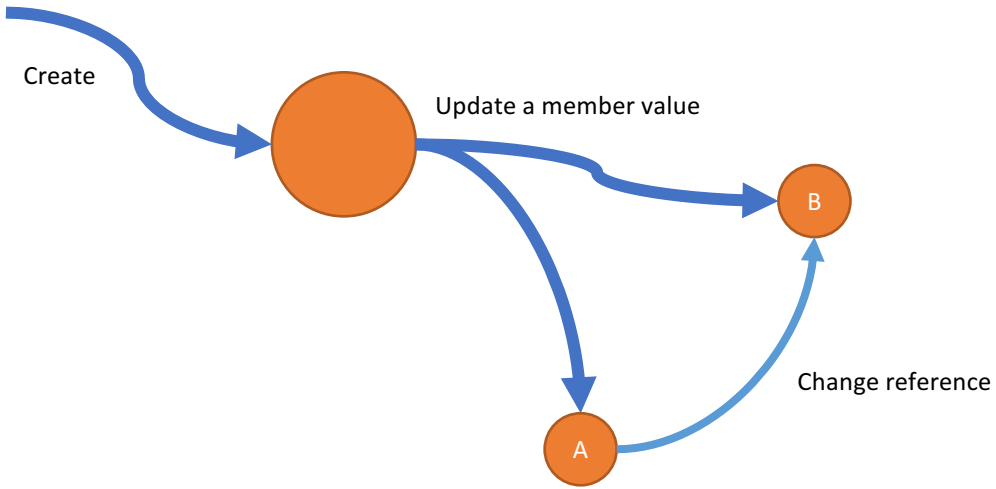
Supported Modes		Data Type		
		Block	Object	Generic
Storage Type	Volatile	✓	✓	✓
	NonVolatile	✓	✓	✓

AVAILABLE MEMORY SERVICES

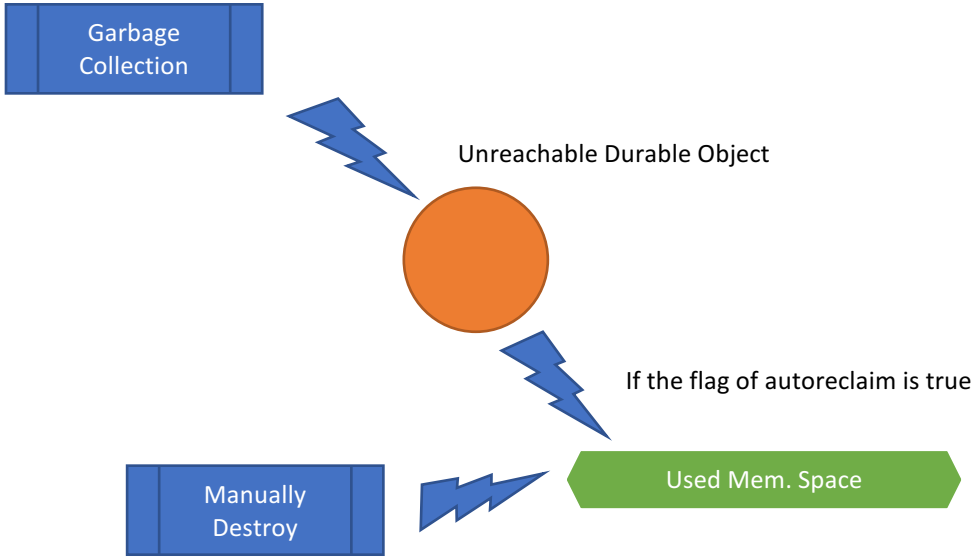
- NVML-VMEM
- PMALLOC
- NVML-PMEM
- SYS-VMEM



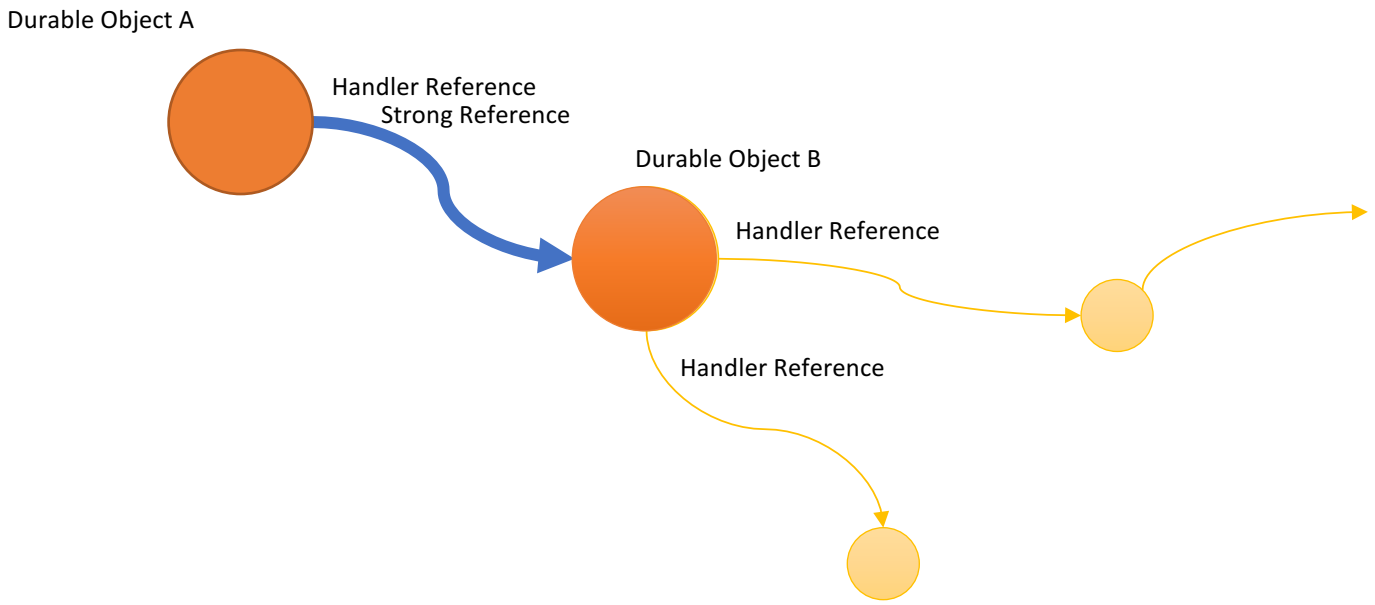
BENEFIT: IN PLACE UPDATES



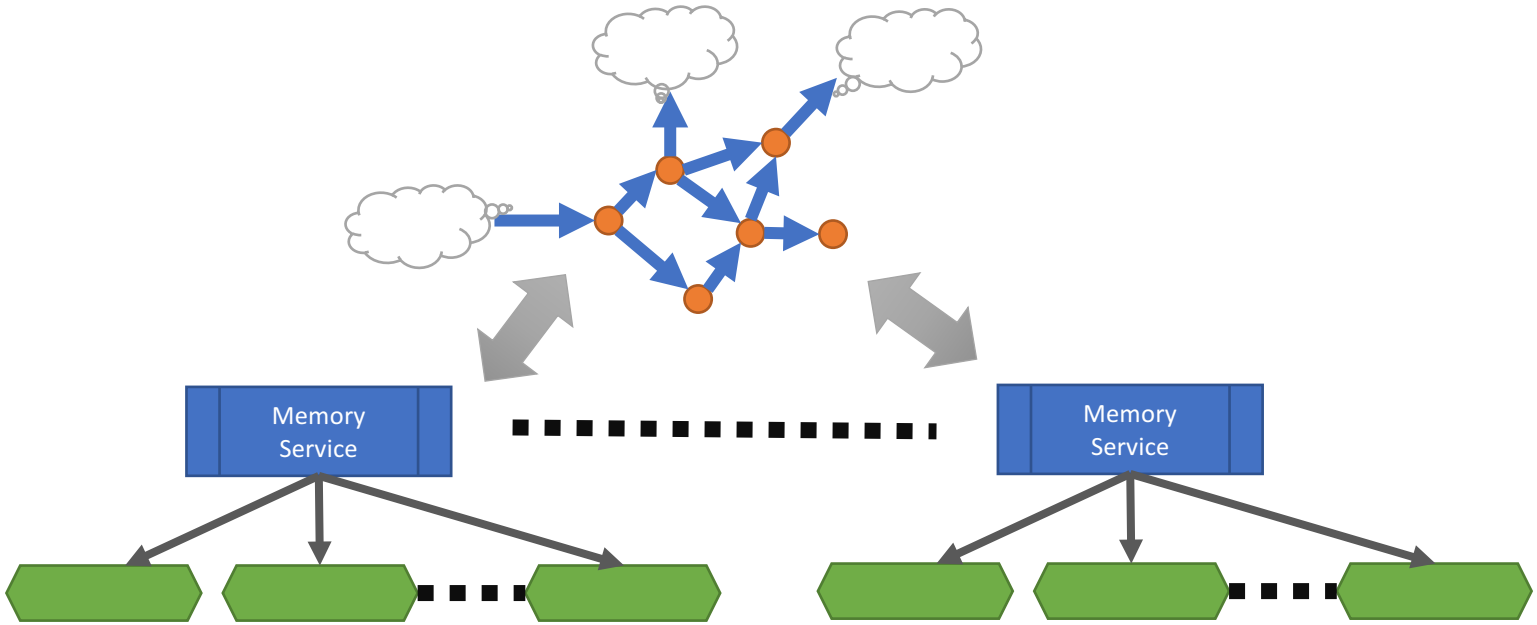
BENEFIT: AUTO RECLAIM



BENEFIT: LAZILY RESTORABLE

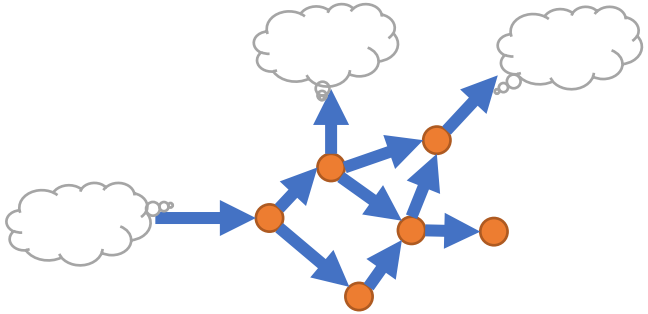


BENEFIT: MEMORY AGNOSTIC

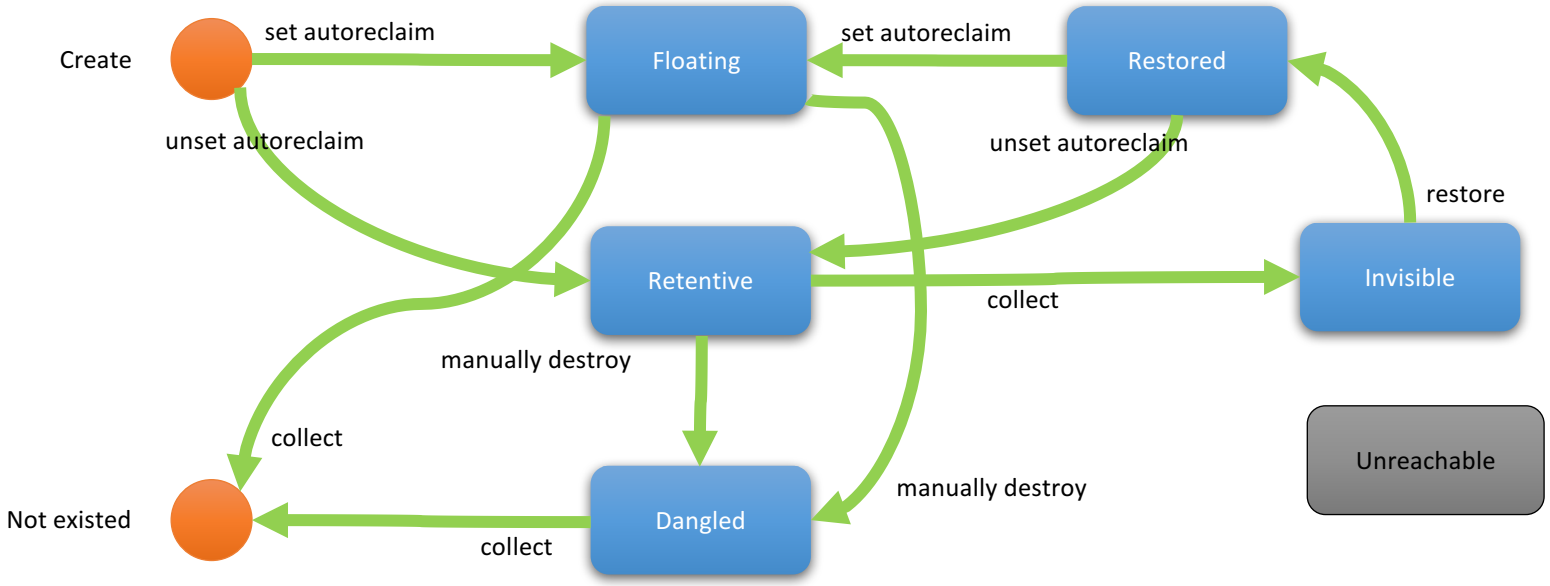


CONSISTENCY

- None
- Close
- Flush
- Single Object Persist
- Object Deep Persist
- Atomic Operation
- Transaction



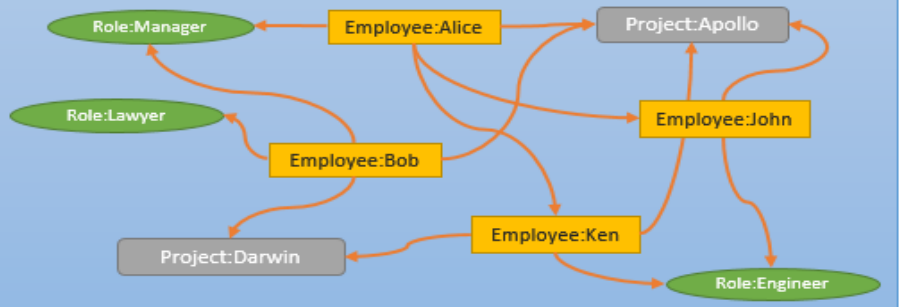
DURABLE OBJECT STATES



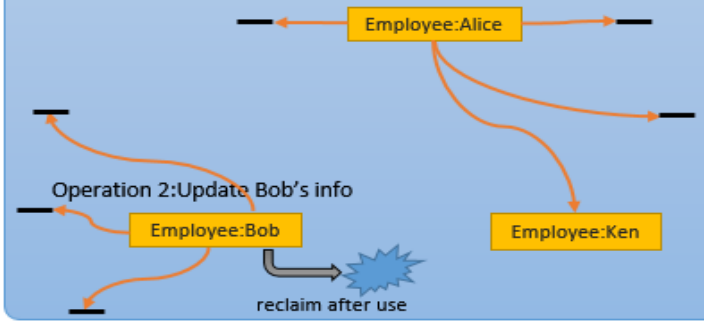
DURABLE OBJECT GRAPH EXAMPLE

APACHE Incubating Mnemonic

Traditional Object Graph View (on Heap)
Operation: Search Alice's team member Ken

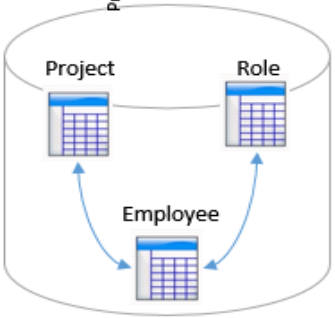


Mnemonic Non-volatile Object Graph View (on Heap)
Operation 1: Search Alice's team member Ken



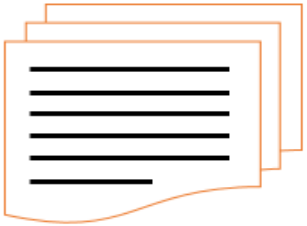
Process Communication

- Normalize
- Move
- Join
- Index
- Search
- Type Mapping
- SerDe



Sequential Read/Write

- Move
- Cache
- Pack & Unpack
- Load all or none
- Bursty traffic
- break encapsulation
- Involve filesystem Cache

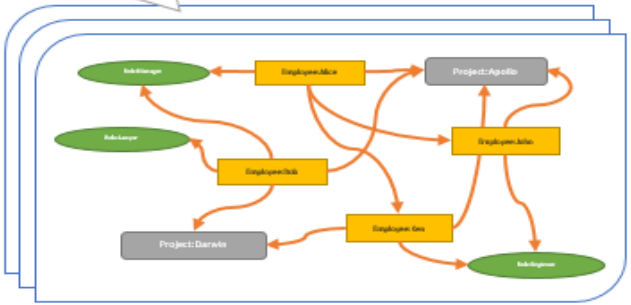


Shadow

- Load on demand
- Reclaim once used
- Share for multiple processes
- In-place creation & update
- No need to normalize
- Hybrid Memory-like device backed
- Pluggable customized allocators as service

Possible Storage Media

- Off-Heap Memory
- Persistent Memory
- NVMe/SSD



DURABLE COLLECTIONS

APACHE Incubating Mnemonic

- Simple Memory Abstraction for Common Data Structures
  - DurableChunk
  - DurableBuffer
  - DurableList
  - DurableMap
  - DurableTree
  - DurableArray
  - DurableSet (WIP)
- Distributed Collections (WIP)

EXAMPLES

Durable Maps

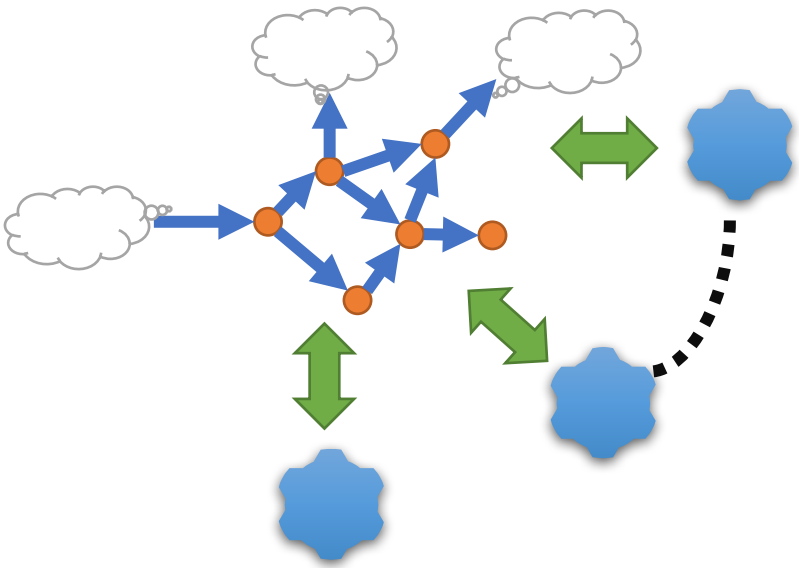
```
DurableHashMap<String, Integer> map = DurableHashMapFactory.create(allocator);
map.put("hello", 1);
map.put("world", 2);
map.put("hello", 1);
```

Durable Arrays

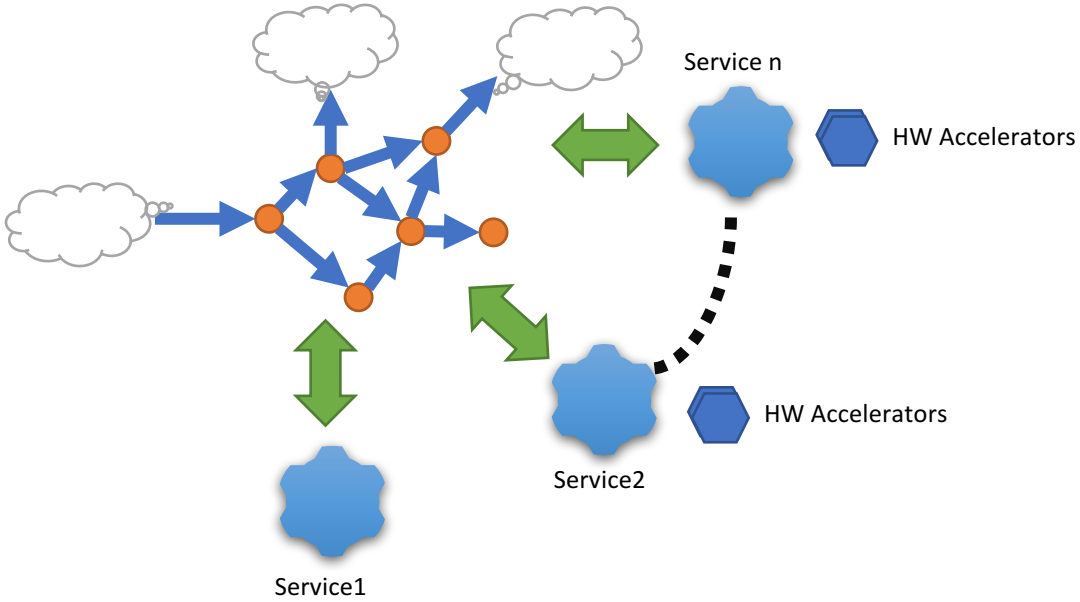
```
DurableArray<String> array = DurableArrayFactory.create(allocator,100);
array.set(1, "string0");
array.get(1);
```

DURABLE COMPUTING MODEL

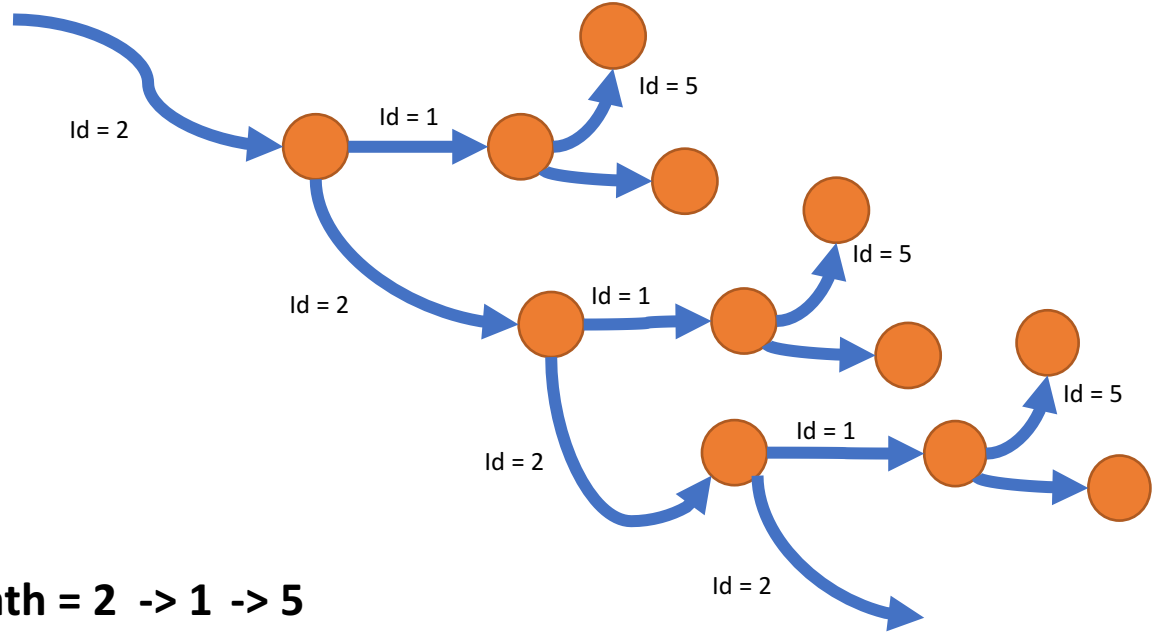
- Simple but not simplistic
- Across VM boundary
- No Marshal/Un-marshalling
- Shared Viewpoint
- Schema-less
- Enable Smart Hardware



DURABLE COMPUTING SERVICE



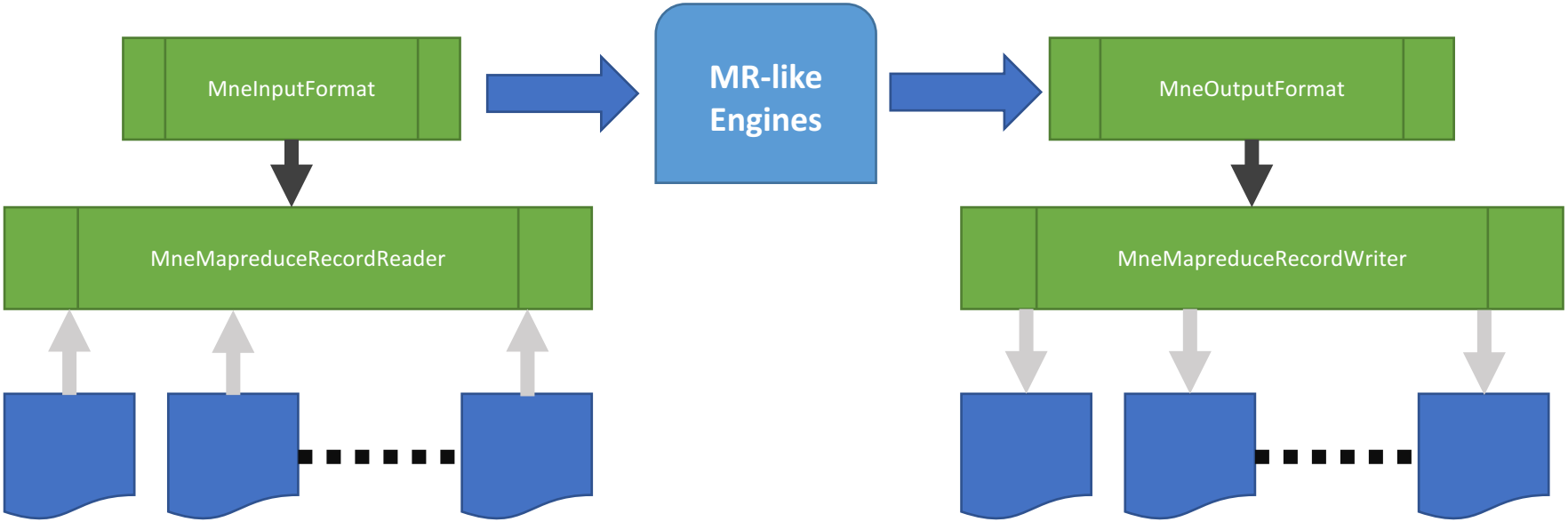
COLLECTED OBJECTS



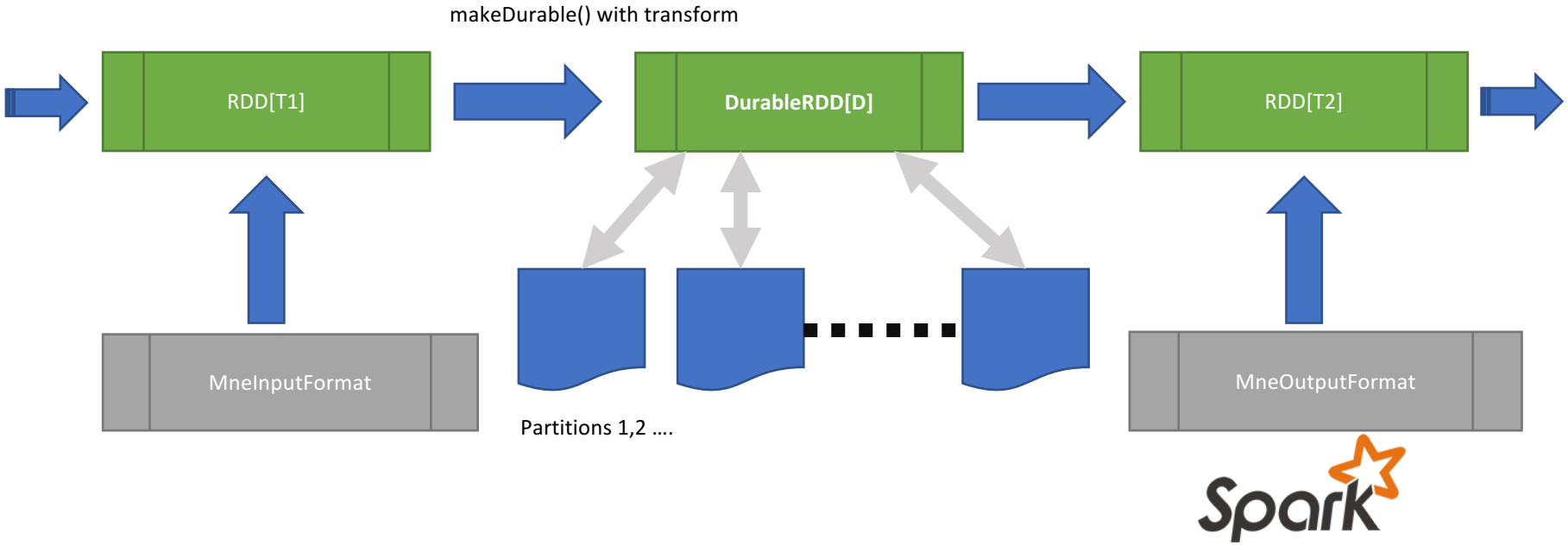
Path = 2 -> 1 -> 5

MNEMONIC INTEGRATION

MNEMONIC – HADOOP



MNEMONIC - SPARK

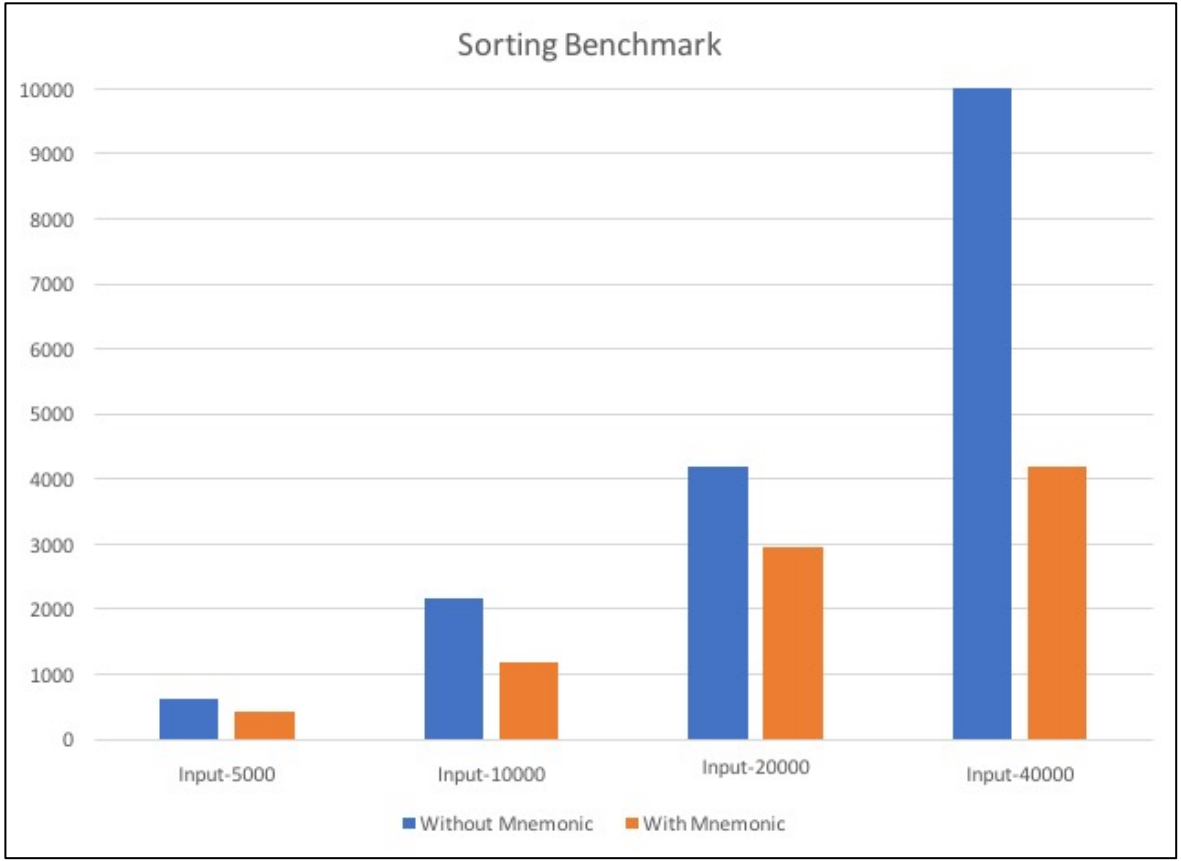


REDUCE GC ACTIVITIES

Exp: GC Sum of executors	Default	Mnemonic	Mnemonic/Default
Total number of FullGC	23	12	0.522
Total FullGC User (s)	1716.82	800.06	0.466
Total FullGC Sys (s)	13.67	6.52	0.477
Total FullGC Real (s)	129.36	59.52	0.460
	Default	Mnemonic	Mnemonic/Default
Total number of YoungGC	1310	864	0.660
Total YoungGC User (s)	6799.45	3892.05	0.572
Total YoungGC Sys (s)	833.54	740.81	0.889
Total YoungGC Real (s)	225.98	158.79	0.703
Total GC pauses (s)	355.34	218.31	0.614
	Default	Mnemonic	Mnemonic/Default
Total FullGC Real (s)	129.36	59.52	0.460
Total YoungGC Real (s)	225.98	158.79	0.703
Total GC pauses (s)	355.34	218.31	0.614

Lesser GC for Mnemonic applications

BUBBLE SORTING EXPERIMENTS



CONCLUSION

Without Mnemonic

- JVM performance costs
  - SerDes
  - JNI invocation
- Complex
  - Own caching schemes
  - Own memory management schemes
  - Own structure transformations
- Poor Performance of JVM based stacks
  - Higher memory usage
  - Frequent GC pauses

With Mnemonic

- Durable Object Model
  - Unified view of storage devices
  - No SerDes cost
  - Less memory usage
  - Reduced JC pauses
  - Handles memory management
- Durable Computing model
  - Bypass JNI calls
  - Better performance

Thanks!

Join Mnemonic community

dev-subscribe@mnemonic.incubator.apache.org

Gang(Gary) Wang
Gang1.wang@intel.com
garyw@apache.org

Johnu George
johnugeo@cisco.com
johnu@apache.org

Debo Dutta
dedutta@cisco.com
ddutta@apache.org