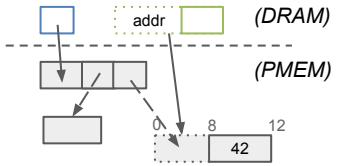
J-NVM:
Off-heap Persistent Objects in Java

Authors
Anatole Lefort
Yohan Pipereau
Kwabena Amponsem
Pierre Sutra
Gaël Thomas

PMEM in Java
Background

- **Java** language used in many **data stores** and **processing frameworks**
- **NVMM = byte-addressable non-volatile** memory (persistent + DRAM speed)
- **Filesystem** or **JNI** are **not efficient** enough to access NVMM
- Prior works for managed language runtimes propose **orthogonal data persistence**, leading to **inefficiencies** and **difficulties** in programming NVMM
- No solution for **garbage collection:** language runtimes **cannot scale** to persistent dataset size

Decoupling illustrated




```
Map root = JNVM.root();
Simple s = root.get("Simple");
s.setX(42);
```

Off-heap Persistent Objects
Decoupling = persistent data structure + volatile proxy

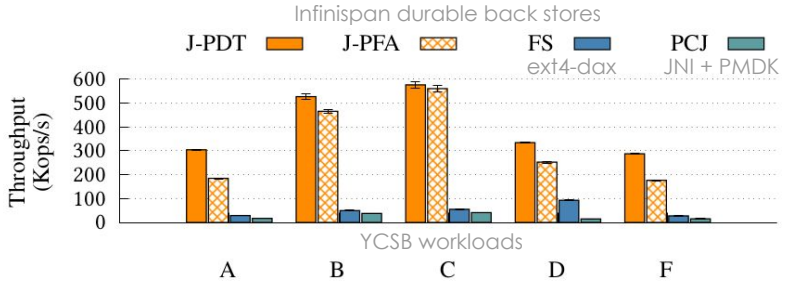
- Persistent **data structure** allocated **off-heap** (NVMM), **unmanaged** by the language runtime
- **Proxy** object instantiated **lazily** on-heap (DRAM), **managed** by the language runtime, **intermediate** the access to data structure (methods), **re-constructed** when dereferencing a persistent pointer
- **Explicit deallocation** of the persistent data structure
- **Recovery-time GC** to allow non-crash-consistent NVMM management
- Objects are **alive** as long as they are **reachable** from a root object.
- Dynamic **root object** definition using **naming** in a **global registry** (persistent map)

J-NVM: high-level API
Implementation:
A java library and framework

- **Code-generator:** automated conversion of POJOs
- **J-PFA:** generic crash consistent data manipulation through failure-atomic blocks of code
- **J-PDT:** hand-made efficient persistent data types, including drop-in replacement for some of the JDK classes (e.g., collections)
- **Low-level API:** custom proxy building with direct memory access intrinsics for fine-grained persistence and performance

Efficient PMEM access
Evaluation:
YCSB and TPC-B like benchmarks

- Up-to **10.5x faster** than **FS**-based persistence on **NVRAM**
- **No** need for a **volatile cache**
- **5x faster recovery** time for 10M objects
- Around **50% slower** than the **DRAM** baseline
- **J-PDT** up to **65%** faster than **J-PFA**



Hardware: 4 Intel CLX 6230 HT (80-core), 128GB DDR4, 4*128GB Optane DC (gen1)

Contact [firstname].[surname][at]telecom-sudparis.eu

github.com/jnvm-project/jnvm