ORACLE®

NVM Killer Apps without the splatter

Garret Swart

SNIA NVM Summit

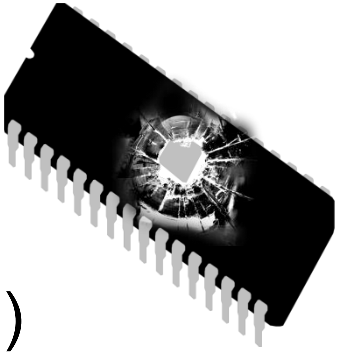
Technology driven innovation

- DRAM compatible NVM technologies are being developed
  - Could eclipse DRAM in density, endurance, and access times
  - Sooner if we can converge on a technology and concentrate demand
- Storage APIs on NVM are great
  - Existing applications can exploit it immediately
- But what about mapping NVM into application address spaces?
  - What are the killer apps? How do they benefit?
  - What issues do we need to solve to make this real?

NVM Taxonomy: **Focus of talk**

- Device Connectivity:
  - **Memory Channel**, Coherency Link (e.g. QPI), PCIe, SAS
  - Why? Highest off-chip bandwidth. Gives up dual-porting
- Device Form Factor:
  - MCM, **DIMM**, PCIe card, 2.5” SFF, Blade, Rack unit
  - Why? Multiple DIMMs/processor, DDR4 standard
    - But: Limits device size & power, no hot swap or easy access.
- SW Abstraction
  - Storage (LUN, Object Store, File System), **Memory Mapped**
  - Why: Lowest latency access
- Device Access Logic: (Addressing, wear leveling)
  - **ASIC**, Firmware, Processor HW, Driver SW, Application SW
  - Why: Looks like DRAM ➔ less changes to processor & SW

Killer App 1: In Memory Databases



- Relational OLTP
  - High update rate, short running queries
- Key-value Store (memcached, ZooKeeper, …)
  - Mutable Map of objects
- Relational Data Warehouse
  - Low update rate, Long running queries
  - Incorporate new data, reorganize and compress
- Full Text Index
  - Maintain a list of hits for each term
  - Combine term hits to answer queries
- Partition and distribute for scale-out
- Replicate for HA and hot spot handling

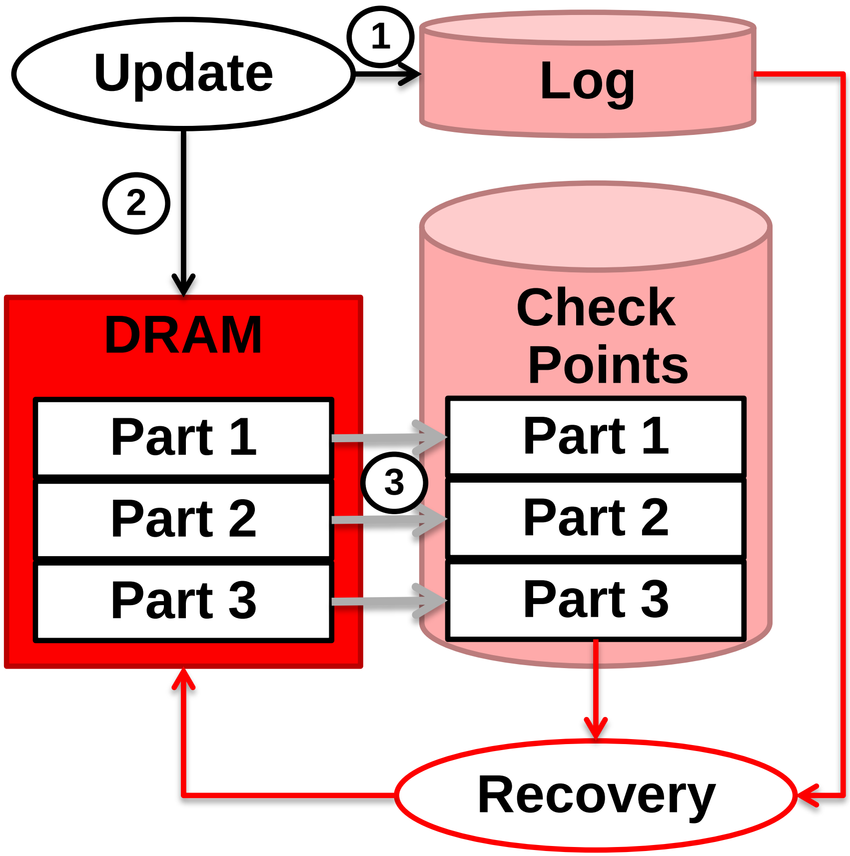
How have we lived without NVM?

- Write ahead logging
  - Append a description of the change to the log
  - Apply change to working copy in DRAM
  - Periodically checkpoint working copy to storage
  - To recover:
    - Read most recent checkpoints into DRAM
    - Apply committed log entries

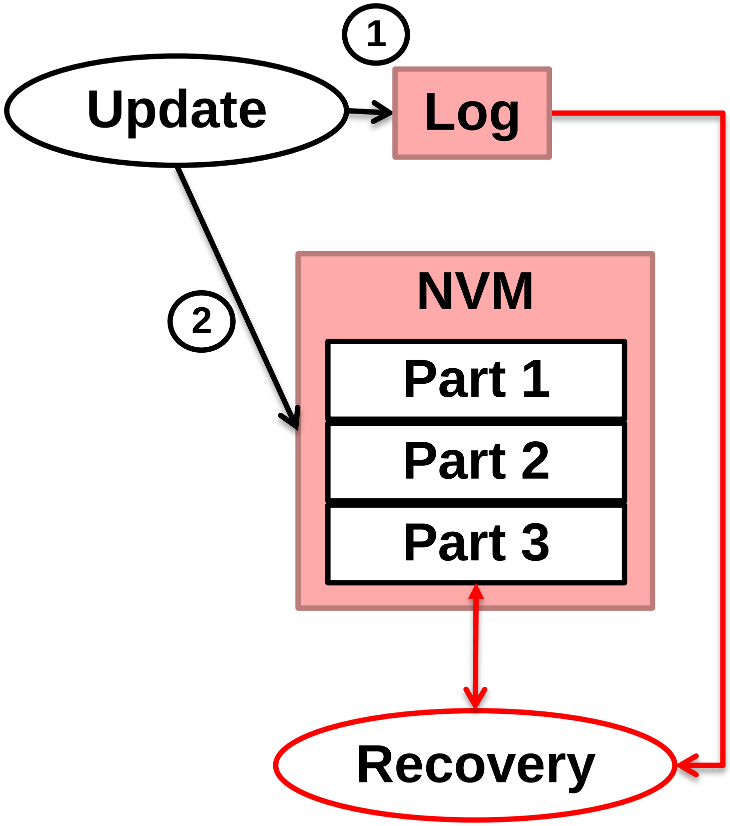
- Today's Costs
  - Writing log to storage: Increases latency on update
  - Writing checkpoints: Interferes with forward progress
  - Recovery: Delays restart if log is long
- NVM Promise
  - Fast writes to log
  - Combine the checkpoint and working copy in NVM
  - Recover only active transactions

In Memory Database

DRAM



NVM



Advantages of Checkpoints

- Checkpoints can be a serialization rather than a copy of memory
  - Makes checkpoint more expensive to make but ...
- Recovery has a side effect of compacting the heap
  - Mitigates entropy
- DRAM can use machine data types and pointers
- Checkpoints can use portable types and foreign keys
- Checkpoints make good backups
- DRAM corruptions can be discovered (and fixed) during checkpointing

Mitigating the Checkpoint advantage in NVM segments

- Backup memory mapped files: not process or machine images
  - Only back up persistent data: Not in-flight data. Failure should not cause a backup to record an update that has not committed
- Periodic partition reorganization
  - Create local replica of partition
  - Not tied to recovery time, tied to memory entropy
    - Should be much less often
- COW data structures in NVM
  - Stores history in an accessible way
  - Makes data corruption less likely

Killer App 2: Data Caches

- Many datasets are too big to fit in memory
  - If there is a skewed access pattern, caching can help
- Big DRAM caches are expensive to rebuild at restart
  - What to cache: Distinguishing cool and cold is difficult
  - Loading data: Data transfer
- OS managed NVM caches not optimal
  - Move active blocks of a file to NVM
  - Read/write means overhead on every access
    - DRAM caching above file system hides hits from file system and can cause hot blocks to be evicted from NVM
  - mmap means VM churn as pages come in and out of NVM
    - VM address space maintenance doesn’t scale

Killer App 2: In-App Caches!

- Application Managed NVM Cache
  - NVM resident memory mapped file used as cache of data stored on file system
  - LD/ST access to cached data
  - Read/Write access to file system data
  - Cache deserialized data structures, not bytes
    - No parsing: use machine native representations
- Issues
  - Sizing multiple application caches sharing same NVM
  - Validating cache against base files
  - Synchronizing multiple caches of same base file
  - Ensuring write-back cache atomicity
    - Update Data and meta-data atomically

Managing persistent memory

- Love those Logs!
  - Memory mapped NVM is best for small entries
  - Library support for finding complete entries and managing log replay
- Replication
  - Physical (NVM files) vs. Logical
    - Physical faster but logical adds resiliency
  - HA requires replica on separate failure domain
    - Failure domain is always on a different system: No such thing as a dual ported DIMM
    - RDMA to memory mapped NVM:
      - Implicit or explicit msync()?

msync() API issues

- **int msync(void ****addr***, size_t** ***length*****, int** ***flags*****);**
- **API requires address range**
  - **Adds overhead to track the ranges**
  - **Maximizing flush parallelism using msync()**
    - **Interferes with abstractions**
    - **Requires two calls per range**
  - **range1 = Btree_update()**
  - **range2 = Hash_update()**
  - **msync(range1, MS_ASYNC)**
  - **msync(range2, MS_ASYNC)**
  - **msync(range1, MS_SYNC)**
  - **msync(range2, MS_SYNC)**
- **Fix: Thread based? Flush on Failure?**