TODAY’S AGENDA

Background
Storage & Recovery Methods for NVM
Project #3 Code Review Guidelines
NON-VOLATILE MEMORY

Emerging storage technology that provide low latency read/writes like DRAM, but with persistent writes and large capacities like SSDs. → AKA Storage-class Memory, Persistent Memory

First devices will be block-addressable (NVMe) Later devices will be byte-addressable.
FUNDAMENTAL ELEMENTS OF CIRCUITS

Capacitor (ca. 1745)

Resistor (ca. 1827)

Inductor (ca. 1831)
In 1971, Leon Chua at Berkeley predicted the existence of a fourth fundamental element.

A two-terminal device whose resistance depends on the voltage applied to it, but when that voltage is turned off it permanently remembers its last resistive state.
FUNDAMENTAL ELEMENTS OF CIRCUITS

Capacitor (ca. 1745)

Resistor (ca. 1827)

Inductor (ca. 1831)

Memristor (ca. 1971)
A team at HP Labs led by Stanley Williams stumbled upon a nano-device that had weird properties that they could not understand.

It wasn’t until they found Chua’s 1971 paper that they realized what they had invented.
A team at HP Labs led by Stanley Williams stumbled upon a nano-device that had weird properties that they could not understand. It wasn't until they found Chua’s 1971 paper that they realized what they had invented.
A team at HP Labs led by Stanley Williams stumbled upon a nano-device that had weird properties that they could not understand. It wasn't until they found Chua's 1971 paper that they realized what they had invented.
MEMRISTOR – HYSTERESIS LOOP

Vacuum Circuits
(ca. 1948)

TWO CENTURIES OF MEMRISTORS
Nature Materials 2012
TECHNOLOGIES

Phase-Change Memory (PRAM)
Resistive RAM (ReRAM)
Magnetoresistive RAM (MRAM)
PHASE-CHANGE MEMORY

Storage cell is comprised of two metal electrodes separated by a resistive heater and the phase change material (chalcogenide).

The value of the cell is changed based on how the material is heated.
→ A short pulse changes the cell to a ‘0’.
→ A long, gradual pulse changes the cell to a ‘1’.
RESISTIVE RAM

Two metal layers with two TiO$_2$ layers in between. Running a current one direction moves electrons from the top TiO$_2$ layer to the bottom, thereby changing the resistance.

May be programmable storage fabric... → Bertrand Russell’s Material Implication Logic
MAGNETORESISTIVE RAM

Stores data using magnetic storage elements instead of electric charge or current flows.

Spin-Transfer Torque (STT-MRAM) is the leading technology for this type of NVM. → Supposedly able to scale to very small sizes (10nm) and have SRAM latencies.
TIMELINE

Intel announced that their **3D XPoint** drives will be available in 2016.
→ Rumors are that the 2017 Xeon ISA will include instructions for NVM DIMMs.

Samsung has recently **partnered** to develop their NVDIMM-P storage.

HP’s ReRam is always two years away...
MEMRISTOR
NON-VOLATILE STORAGE

1. Resistor
2. Capacitor
3. Inductor
4. Memristor

A resistor with memory

2006: HP Labs proves fourth fundamental element of electronic circuitry

2008: Development ready

FUTURE

RESEARCH CONTRIBUTION

Replace DRAM and hard drives, transistors

Source: Luke Kilpatrick
NVM FOR DATABASE SYSTEMS

Block-addressable NVM is not that interesting.

Byte-addressable NVM will be a game changer but will require some work to use correctly.
→ In-memory DBMSs will be better positioned to use byte-addressable NVM.
→ Disk-oriented DBMSs will initially treat NVM as just a faster SSD.

More significant for OLTP workloads.
STORAGE & RECOVERY METHODS

Understand how a DBMS will behave on a system that only has byte-addressable NVM.

Develop NVM-optimized implementations of standard DBMS architectures.

Based on the N-Store prototype DBMS.
SYNCHRONIZATION

Existing programming models assume that any write to memory is non-volatile. → CPU decides when to move data from caches to DRAM.

The DBMS needs a way to ensure that data is flushed from caches to NVM.
Existing programming models assume that any write to memory is non-volatile. → CPU decides when to move data from caches to DRAM.

The DBMS needs a way to ensure that data is flushed from caches to NVM.
If the DBMS process restarts, we need to make sure that all of the pointers for in-memory data point to the same data.
If the DBMS process restarts, we need to make sure that all of the pointers for in-memory data point to the same data.
If the DBMS process restarts, we need to make sure that all of the pointers for in-memory data point to the same data.
NVM-AWARE MEMORY ALLOCATOR

Feature #1: Synchronization
→ The allocator writes back CPU cache lines to NVM using the CLFLUSH instruction.
→ It then issues a SFENCE instruction to wait for the data to become durable on NVM.

Feature #2: Naming
→ The allocator ensures that virtual memory addresses assigned to a memory-mapped region never change even after the OS or DBMS restarts.
Choice #1: In-place Updates
→ Table heap with a write-ahead log + snapshots.
→ Example: VoltDB

Choice #2: Copy-on-Write
→ Create a shadow copy of the table when updated.
→ No write-ahead log.
→ Example: LMDB

Choice #3: Log-structured
→ All writes are appended to log. No table heap.
→ Example: RocksDB
IN-PLACE UPDATES ENGINE

In-Memory Index

In-Memory Table Heap
- Tuple #00
- Tuple #01
- Tuple #02

Durable Storage
- Write-Ahead Log
- Snapshots
IN-PLACE UPDATES ENGINE

In-Memory Index

In-Memory Table Heap

Tuple #00

Tuple #01

Tuple #02

Durable Storage

Write-Ahead Log

Snapshots
IN-PLACE UPDATES ENGINE

In-Memory Index

In-Memory Table Heap
- Tuple #00
- Tuple #01
- Tuple #02

Durable Storage
- Write-Ahead Log
  - Tuple Delta
- Snapshots
IN-PLACE UPDATES ENGINE

In-Memory Index

In-Memory Table Heap

Tuple #00

Tuple #01 (!)

Tuple #02

Durable Storage

Write-Ahead Log

Tuple Delta

Snapshots
IN-PLACE UPDATES ENGINE

In-Memory Index

In-Memory Table Heap

Tuple #00
Tuple #01 (!)
Tuple #02

Durable Storage

Write-Ahead Log
Tuple Delta

Snapshots
Tuple #01 (!)
Duplicate Data

Tuple #02

Snapshots

Tuple #01 (!)
IN-PLACE UPDATES ENGINE

In-Memory Table Heap

Tuple #00

Durable Storage

Write-Ahead Log

Tuple Delta

In-Memory Index

Tuple #01

Snapshots

Tuple #01 (!)

1

2

3

Duplicate Data

Recovery Latency
NVM-OPTIMIZED ARCHITECTURES

Leverage the allocator’s non-volatile pointers to only record what changed rather than how it changed.

The DBMS only has to maintain a transient UNDO log for a txn until it commits.  
→ Dirty cache lines from an uncommitted txn can be flushed by hardware to the memory controller.  
→ No REDO log because we flush all the changes to NVM at the time of commit.
NVM IN-PLACE UPDATES ENGINE

NVM Index

NVM Table Heap

Tuple #00
Tuple #01
Tuple #02

NVM Storage

Write-Ahead Log
NVM IN-PLACE UPDATES ENGINE

NVM Index

NVM Table Heap
- Tuple #00
- Tuple #01
- Tuple #02

NVM Storage

Write-Ahead Log
NVM IN-PLACE UPDATES ENGINE

NVM Index

NVM Table Heap

1. Tuple Pointers

NVM Storage

Write-Ahead Log

Tuple #00

Tuple #01

Tuple #02
NVM IN-PLACE UPDATES ENGINE

NVM Index

NVM Table Heap

1. Tuple Pointers

NVM Storage

Write-Ahead Log

2. Tuple #01 (!)

Tuple #00

Tuple #01

Tuple #02
COPY-ON-WRITE ENGINE

Master Record

Current Directory

Leaf 1

Leaf 2

Slotted Page #00

Slotted Page #01
COPY-ON-WRITE ENGINE

Master Record

Current Directory

Leaf 1

Leaf 2

Slotted Page #00

Slotted Page #01
COPY-ON-WRITE ENGINE

Master Record

Current Directory

Leaf 1

Leaf 2

Updated Leaf 1

Slotted Page #00

Slotted Page #01

Slotted Page #00
COPY-ON-WRITE ENGINE

Master Record

Current Directory

Dirty Directory

Leaf 1

Leaf 2

Updated Leaf 1

Slotted Page #00

Slotted Page #01

Slotted Page #00
COPY-ON-WRITE ENGINE

Expensive Copies

Current Directory

Dirty Directory

Leaf 1

Leaf 2

Updated Leaf 1

Slotted Page #00

Slotted Page #01

Slotted Page #00
NVM COPY-ON-WRITE ENGINE

Master Record

Current Directory

Leaf 1

Tuple #00

Leaf 2

Tuple #01
NVM COPY-ON-WRITE ENGINE

Current Directory

Master Record

Leaf 1
- Tuple #00

Leaf 2
- Tuple #01

Updated Leaf 1
- Only Copy Pointers
- Tuple #00 (!)
NVM COPY-ON-WRITE ENGINE

- **Current Directory**
  - Leaf 1
    - Tuple #00
  - Leaf 2
    - Tuple #01

- **Dirty Directory**
  - Updated Leaf 1
    - Only Copy Pointers
    - Tuple #00 (!)

- **Master Record**
  - 3
LOG-STRUCTURED ENGINE

MemTable

SSTable

Write-Ahead Log

Bloom Filter
LOG-STRUCTURED ENGINE

**MemTable**

- Write-Ahead Log
  - Tuple Delta

**SSTable**

- Bloom Filter
LOG-STRUCTURED ENGINE

MemTable

Write-Ahead Log

Tuple Delta

SSTable

Bloom Filter

Tuple Delta

Tuple Data
LOG-STRUCTURED ENGINE

Duplicate Data

1. Write-Ahead Log
   Tuple Delta

2. Tuple Delta
   Tuple Data

3. Tuple Data
LOG-STRUCTURED ENGINE

⚠️ Duplicate Data

⚠️ Compactions

Tuple Data
**NVM LOG-STRUCTURED ENGINE**

- **MemTable**
  - Write-Ahead Log
  - Tuple Delta

- **SSTable**
  - Bloom Filter
  - Tuple Delta
  - Tuple Data

1. Write-Ahead Log
2. Tuple Delta
3. Tuple Data
NVM LOG-STRUCTURED ENGINE

**MemTable**

1. Write-Ahead Log
   - Tuple Delta

**SSTable**

2. Bloom Filter
3. Tuple Data
NVM LOG-STRUCTURED ENGINE

MemTable

Write-Ahead Log

Tuple Delta
SUMMARY

Storage Optimizations
→ Leverage byte-addressability to avoid unnecessary data duplication.

Recovery Optimizations
→ NVM-optimized recovery protocols avoid the overhead of processing a log.
→ Non-volatile data structures ensure consistency.
EVALUATION

N-Store DBMS testbed with pluggable storage manager architecture.
→ H-Store-style concurrency control

Intel Labs NVM Hardware Emulator
→ NVM latency = 2x DRAM latency

Yahoo! Cloud Serving Benchmark
→ 2 million records + 1 million transactions
→ 10% Reads / 90% Writes
→ High-skew setting
RUNTIME PERFORMANCE

YCSB Workload – 10% Reads / 90% Writes
NVRAM – 2x DRAM Latency

- Traditional
- NVM-Optimized

Throughput (txn/sec)

<table>
<thead>
<tr>
<th></th>
<th>Traditional</th>
<th>NVM-Optimized</th>
</tr>
</thead>
<tbody>
<tr>
<td>In-Place</td>
<td>400000</td>
<td>1200000</td>
</tr>
<tr>
<td>Copy-on-Write</td>
<td>800000</td>
<td>900000</td>
</tr>
<tr>
<td>Log-Structured</td>
<td>1200000</td>
<td>1200000</td>
</tr>
</tbody>
</table>
WRITE ENDURANCE

YCSB Workload – 10% Reads / 90% Writes
NVRAM – 2x DRAM Latency

- Traditional
- NVM-Optimized

NVM Stores (M)

- In-Place
- Copy-on-Write
- Log-Structured
WRITE ENDURANCE

YCSB Workload – 10% Reads / 90% Writes
NVRAM – 2x DRAM Latency

Traditional   NVM-Optimized

In-Place  ↓40%
Copy-on-Write  ↓25%
Log-Structured  ↓20%
RECOVERY LATENCY

Elapsed time to replay log on recovery
NVRAM – 2x DRAM Latency

Recovery Time (ms)

<table>
<thead>
<tr>
<th>Recovery Time (ms)</th>
<th>Traditional</th>
<th>NVM-Optimized</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^3$</td>
<td>100</td>
<td>0.1</td>
</tr>
<tr>
<td>$10^4$</td>
<td>1000</td>
<td>0.1</td>
</tr>
<tr>
<td>$10^5$</td>
<td>10000</td>
<td>0.1</td>
</tr>
</tbody>
</table>

In-Place

Copy-on-Write

Log-Structured
**RECOVERY LATENCY**

*Elapsed time to replay log on recovery  
NVRAM – 2x DRAM Latency*

- **Traditional**
- **NVM-Optimized**

**No Recovery Needed**

<table>
<thead>
<tr>
<th>Recovery Time (ms)</th>
<th>In-Place</th>
<th>Copy-on-Write</th>
<th>Log-Structured</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^3$</td>
<td>$10^4$</td>
<td>$10^5$</td>
<td>$10^3$</td>
</tr>
<tr>
<td>$10^4$</td>
<td>$10^5$</td>
<td>$10^5$</td>
<td>$10^4$</td>
</tr>
<tr>
<td>$10^5$</td>
<td></td>
<td></td>
<td>$10^5$</td>
</tr>
</tbody>
</table>
PARTING THOUGHTS

Designing for NVM is important → Non-volatile data structures provide higher throughput and faster recovery

Byte-addressable NVM is going to be a game changer when it comes out.
CODE REVIEWS

Every group will perform a code review of another group.
→ Dev group will send a pull request on Github.
→ Review group will write comments on that request.
→ You will need to send me your pull request URL

We will provide a write-up later this week.

Due Date: May 8th @ 11:59pm

Please be helpful and courteous.
GENERAL TIPS

The dev team should provide you with a summary of what files/functions the reviewing team should look at.

Review fewer than 400 lines of code at a time and only for at most 60 minutes.

Use a checklist to outline what kind of problems you are looking for.
CHECKLIST – GENERAL

Does the code work?
Is all the code easily understood?
Is there any redundant or duplicate code?
Is the code as modular as possible?
Can any global variables be replaced?
Is there any commented out code?
Is it using proper debug log functions?

Source: Gareth Wilson
CHECKLIST – DOCUMENTATION

Do comments describe the intent of the code?
Are all functions commented?
Is any unusual behavior described?
Is the use of 3rd-party libraries documented?
Is there any incomplete code?
CHECKLIST – TESTING

Do tests exist and are they comprehensive?
Are the tests actually testing the feature?
Are they relying on hardcoded answers?
What is the code coverage?

Source: Gareth Wilson
## LCOV - code coverage report

### Current view: top level

**Test:** Peloton-0.2 Code Coverage  
**Date:** 2016-04-15  
**Legend:** Rating: low: < 75%  medium: 75% - 90%  high: >= 90%

<table>
<thead>
<tr>
<th>Directory</th>
<th>Line Coverage</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src</td>
<td>0.0 % 0 / 347</td>
<td>0.0 % 0 / 33</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/backup</td>
<td>100.0 % 121 / 121</td>
<td>88.9 % 24 / 27</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/backup/backup</td>
<td>3.1 % 6 / 195</td>
<td>28.6 % 2 / 7</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/backup/backup/executor</td>
<td>1.5 % 5 / 182</td>
<td>26.2 % 2 / 7</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/backup/backup/executor/executor</td>
<td>1.0 % 5 / 182</td>
<td>26.2 % 2 / 7</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/backup/backup/executor/executor/exec</td>
<td>30.0 % 6 / 200</td>
<td>47.7 % 4 / 8</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/backup/backup/executor/executor/exec/exec</td>
<td>1.7 % 21 / 1249</td>
<td>31.4 % 33 / 104</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/backup/backup/executor/executor/exec/exec/exec</td>
<td>0.4 % 1 / 233</td>
<td>33.1 % 2 / 6</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/backup/backup/executor/executor/exec/exec/exec/catalog</td>
<td>73.2 % 232 / 317</td>
<td>74.0 % 71 / 96</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/backup/backup/executor/executor/exec/exec/exec/catalog/catalog</td>
<td>43.8 % 1524 / 3482</td>
<td>57.4 % 316 / 551</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/backup/executor</td>
<td>84.5 % 1939 / 2295</td>
<td>88.4 % 205 / 232</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/executor</td>
<td>88.6 % 1711 / 1931</td>
<td>85.9 % 274 / 319</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/executor/executor</td>
<td>11.9 % 321 / 2869</td>
<td>10.3 % 129 / 1253</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/executor/executor/exec</td>
<td>35.5 % 200 / 563</td>
<td>8.1 % 67 / 827</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/executor/executor/exec/exec</td>
<td>33.3 % 91 / 273</td>
<td>44.7 % 34 / 76</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/executor/executor/exec/exec/exec</td>
<td>19.9 % 43 / 216</td>
<td>47.1 % 8 / 17</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/executor/executor/exec/exec/exec/executor</td>
<td>10.2 % 81 / 791</td>
<td>27.8 % 20 / 72</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/executor/executor/exec/exec/exec/executor/executor</td>
<td>33.8 % 48 / 142</td>
<td>50.0 % 18 / 36</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/executor/executor/exec/exec/exec/executor/executor/exec</td>
<td>11.1 % 1 / 9</td>
<td>50.0 % 2 / 4</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/executor/executor/exec/exec/exec/executor/executor/exec/exec</td>
<td>13.7 % 1182 / 8618</td>
<td>18.7 % 304 / 1626</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/executor/executor/exec/exec/exec/executor/executor/exec/exec/exec</td>
<td>46.0 % 190 / 413</td>
<td>52.3 % 104 / 199</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/executor/executor/exec/exec/exec/executor/executor/exec/exec/exec/exec</td>
<td>55.2 % 853 / 1544</td>
<td>71.7 % 172 / 240</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/executor/executor/exec/exec/exec/executor/executor/exec/exec/exec/exec/executor</td>
<td>81.8 % 27 / 33</td>
<td>90.5 % 19 / 21</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/executor/executor/exec/exec/exec/executor/executor/exec/exec/exec/exec/executor/exec</td>
<td>100.0 % 40 / 40</td>
<td>88.9 % 8 / 9</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/executor/executor/exec/exec/exec/executor/executor/exec/exec/exec/exec/executor/exec/exec/exec</td>
<td>95.8 % 1176 / 1227</td>
<td>96.9 % 186 / 192</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/executor/executor/exec/exec/exec/executor/executor/exec/exec/exec/exec/executor/exec/exec/exec/concurrency</td>
<td>91.3 % 701 / 766</td>
<td>91.5 % 108 / 118</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/executor/executor/exec/exec/exec/executor/executor/exec/exec/exec/exec/executor/exec/exec/exec/concurrency/concurrency</td>
<td>98.7 % 2049 / 2077</td>
<td>94.8 % 289 / 305</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/executor/executor/exec/exec/exec/executor/executor/exec/exec/exec/exec/executor/exec/exec/exec/concurrency/concurrency/concurrency</td>
<td>99.1 % 343 / 346</td>
<td>97.3 % 72 / 74</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/executor/executor/exec/exec/exec/executor/executor/exec/exec/exec/exec/executor/exec/exec/exec/concurrency/concurrency/concurrency/concurrency/cpu</td>
<td>97.1 % 135 / 139</td>
<td>96.3 % 52 / 54</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/executor/executor/exec/exec/exec/executor/executor/exec/exec/exec/exec/executor/exec/exec/exec/concurrency/concurrency/concurrency/concurrency/cpu/exec</td>
<td>98.4 % 243 / 247</td>
<td>93.0 % 40 / 43</td>
</tr>
<tr>
<td>/var/lib/ranking/job/Peloton/workspace/src/backup/executor/executor/exec/exec/exec/executor/executor/exec/exec/exec/exec/executor/exec/exec/exec/concurrency/concurrency/concurrency/concurrency/cpu/exec/exec</td>
<td>100.0 % 34 / 34</td>
<td>88.9 % 16 / 18</td>
</tr>
</tbody>
</table>
## Group Assignments

<table>
<thead>
<tr>
<th></th>
<th>Logging</th>
<th>Multi-Threaded Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Constraints</td>
<td>Garbage Collection</td>
</tr>
<tr>
<td></td>
<td>UDFs</td>
<td>Memcache</td>
</tr>
<tr>
<td></td>
<td>Query Planning</td>
<td>Concurrency Control</td>
</tr>
<tr>
<td></td>
<td>Statistics</td>
<td>Query Compilation</td>
</tr>
</tbody>
</table>
NEXT CLASS

Final Exam Review
Ankur Goyal (CMU’15 / MemSQL)