ADMINISTRIVIA

April 29: Guest Speaker (Live)

May 4: Code Review #2 Submission

May 5: Final Presentations (Live)

May 13: Final Exam Due Date

May 16: Hack-a-Thon (Extra Credit, Optional)
Administrivia

Course Evaluation

→ Please tell me what you really think of me.
→ I take your feedback in consideration.
→ Take revenge on next year's students.

https://cmu.smartevals.com/
People have been thinking about using hardware to accelerate DBMSs for decades.

**1980s:** Database Machines

**2000s:** FPGAs + Appliances

**2010s:** FPGAs + GPUs

**2020s:** PM + FPGAs + GPUs + **CSAs** + More!
TODAY’S AGENDA

Persistent Memory
GPU Acceleration
Hardware Transactional 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, Non-Volatile Memory

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

Capacitor (1745)

Resistor (1827)

Inductor (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 (1745)

Resistor (1827)

Inductor (1831)

Memristor (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.
MEMRISTOR
NON-VOLATILE STORAGE

A resistor with memory

RESEARCH CONTRIBUTION

› 2006: HP Labs proves fourth fundamental element of electronic circuitry

FUTURE

› 2008: Development ready

› Replace DRAM and hard drives, transistors

Source: Luke Kilpatrick
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 TiO₂ layers in between. Running a current one direction moves electrons from the top TiO₂ layer to the bottom, thereby changing the resistance.

Potential 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 PM.
→ Supposedly able to scale to very small sizes (10nm) and have SRAM latencies.
Industry has agreed to standard technologies and form factors (JDEC).

Linux and Microsoft added support for PM in their kernels (DAX).

Intel added new instructions for flushing cache lines to PM (CLFLUSH, CLWB).
WHY THIS IS FOR REAL

Industry has agreed to standard technologies and form factors (JDEC).

Linux and Microsoft added support for PM in their kernels.

Intel added new instructions for flushing cache lines to PM (CLFLUSH, CLWB).
PM CONFIGURATIONS

DRAM as Hardware-Managed Cache

PM Next to DRAM

Source: Ismail Oukid
PM FOR DATABASE SYSTEMS

Block-addressable PM is not that interesting.

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

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

Develop PM-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 PM.
NAMING

If the DBMS process restarts, we need to make sure that all the pointers for in-memory data point to the same data.
NAMING

If the DBMS process restarts, we need to make sure that all the pointers for in-memory data point to the same data.
If the DBMS process restarts, we need to make sure that all the pointers for in-memory data point to the same data.
PM-AWARE MEMORY ALLOCATOR

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

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

Tuple Delta

Snapshots
IN-PLACE UPDATES ENGINE

In-Memory Index

In-Memory Table Heap

1. Durable Storage
   - Write-Ahead Log
     - Tuple Delta
   - Snapshots

2. Tuple #00
   - Tuple #01 (!)
   - Tuple #02
IN-PLACE UPDATES ENGINE

In-Memory Index

In-Memory Table Heap

Durable Storage

1. Write-Ahead Log
   Tuple Delta

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

3. Snapshots
   Tuple #01 (!)
IN-PLACE UPDATES ENGINE

In-Memory - In-Memory - Durable

⚠️ Duplicate Data

⚠️ Recovery Latency
PM-OPTIMIZED ARCHITECTURES

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

The DBMS only must 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 PM at the time of commit.
PM IN-PLACE UPDATES ENGINE

**PM Index**

**PM Table Heap**
- Tuple #00
- Tuple #01
- Tuple #02

**PM Storage**
- Write-Ahead Log
  - Tuple Pointers

1
PM IN-PLACE UPDATES ENGINE

PM Index

PM Table Heap

1. Write-Ahead Log
   Tuple Pointers

2. Tuple Heap
   Tuple #00
   Tuple #01 (!)
   Tuple #02

PM Storage
COPY-ON-WRITE ENGINE

Master Record

Current Directory

Leaf 1

Page #00

Leaf 2

Page #01
COPY-ON-WRITE ENGINE

Master Record

Current Directory

Leaf 1

Leaf 2

Updated Leaf 1

Page #00

Page #01

Page #00

15-721 (Spring 2020)
COPY-ON-WRITE ENGINE

Master Record

Current Directory

Dirty Directory

Leaf 1

Leaf 2

Updated Leaf 1

Page #00

Page #01

Page #00
COPY-ON-WRITE ENGINE

Master Record

Current Directory

Dirty Directory

Leaf 1

Page #00

Leaf 2

Page #01

Updated Leaf 1

Page #00
COPY-ON-WRITE ENGINE

Expensive Copies

Current Directory

Dirty Directory

Leaf 1

Leaf 2

Updated Leaf 1

Page #00

Page #01

Page #00
PM COPY-ON-WRITE ENGINE

**Master Record**

**Current Directory**

- **Leaf 1**
  - Tuple #00

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

Master Record

Leaf 1
Tuple #00

Leaf 2
Tuple #01

Updated Leaf 1
Tuple #00 (!)

Only Copy Pointers
PM COPY-ON-WRITE ENGINE

Current Directory

Leaf 1

Tuple #00

Leaf 2

Tuple #01

Dirty Directory

Master Record

Updated Leaf 1

Only Copy Pointers

Tuple #00 (!)
LOG-STRUCTURED ENGINE

**MemTable**

- Write-Ahead Log
- Tuple Delta

**SSTable**

- Bloom Filter

15-721 (Spring 2020)
LOG-STRUCTURED ENGINE

MemTable

- Write-Ahead Log
- Tuple Delta

SSTable

- Bloom Filter
- Tuple Delta
- Tuple Data
LOG-STRUCTURED ENGINE

⚠️ Duplicate Data

⚠️ Compactions

Tuple Data

15-721 (Spring 2020)
PM LOG-STRUCTURED ENGINE

MemTable

1. Write-Ahead Log
   Tuple Delta

SSTable

2. Bloom Filter
3. Tuple Delta
4. Tuple Data
PM LOG-STRUCTURED ENGINE

MemTable

1. Write-Ahead Log
   Tuple Delta

2. SSTable
   Bloom Filter

3. Tuple Delta
   Tuple Data
PM LOG-STRUCTURED ENGINE

MemTable

Write-Ahead Log

Tuple Delta

1
**OBSERVATION**

WAL serves two purposes
→ Transform random writes into sequential log writes.
→ Support transaction rollback.
→ Design makes sense for disks with slow random writes.

But PM supports fast random writes
→ Directly write data to the multi-versioned database.
→ Only record meta-data about committed txns in log.
WRITE-BEhind LOGGING

PM-centric logging protocol that provides instant recovery and minimal duplication overhead.
→ Directly propagate changes to the database.
→ Only record meta-data in log.

Recover the database almost instantaneously.
→ Need to record meta-data about in-flight transactions.
→ In case of failure, ignore their effects.
The diagram illustrates the concept of write-behind logging. It shows a SQL update statement:

```
UPDATE table SET val=ABC
WHERE id=123
```

The diagram is divided into two sections: DRAM and PM.

- **DRAM** section:
  - **Table Heap**: The current state of the table heap before the update.

- **PM** section:
  - **Table Heap**: The current state of the table heap after the update.
  - **Log**: The log records of the update operation.
WRITE-BEHIND LOGGING

\[
\text{UPDATE table SET val=ABC WHERE id=123}
\]
WRITE-BEHIND LOGGING

DBMS assigns timestamps to transactions
→ Get timestamps within same group commit timestamp range to identify and ignore effects of in-flight txns.

Use failed group commit timestamp range:
→ DBMS uses range during tuple visibility checks.
→ Ignores tuples created or updated within this range.
→ UNDO is implicitly done via visibility checks.
WRITE-BEHIND LOGGING

Recovery consists of only analysis phase
→ The DBMS can immediately start processing transactions after restart with explicit UNDO/REDO phases.

Garbage collection eventually kicks in to remove the physical versions of uncommitted transactions.
→ Using timestamp range information in write-behind log.
→ After this finishes, no need to do extra visibility checks.
METADATA FOR INSTANT RECOVERY

Use group commit timestamp range to ignore effects of transactions in failed group commit.
→ Maintain list of failed timestamp ranges.
WRITE-BEHIND LOGGING – RECOVERY

Replay Log with 1m TPC-C Transactions
PM 2× Latency Relative to DRAM

- Write-Ahead
- Write-Behind

<table>
<thead>
<tr>
<th>Storage Type</th>
<th>Recovery Time (sec)</th>
<th>Latency Relative to DRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard Disk Drive</td>
<td>1,000 ×</td>
<td>1000×</td>
</tr>
<tr>
<td>Solid State Drive</td>
<td>1,000 ×</td>
<td>1000×</td>
</tr>
<tr>
<td>Persistent Memory</td>
<td>1,000 ×</td>
<td>1000×</td>
</tr>
</tbody>
</table>
WRITE-BEHIND Logging – Runtime

TPC-C Transactions (Eight Warehouses)
PM 2× Latency Relative to DRAM

Throughput (txn/sec)

- **Write-Ahead**
- **Write-Behind**

Hard Disk Drive
Solid State Drive
Persistent Memory

<table>
<thead>
<tr>
<th>Storage Type</th>
<th>Throughput (txn/sec)</th>
<th>Latency Relative to DRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard Disk Drive</td>
<td>10×</td>
<td></td>
</tr>
<tr>
<td>Solid State Drive</td>
<td>10×</td>
<td>10×</td>
</tr>
<tr>
<td>Persistent Memory</td>
<td>1.2×</td>
<td>10×</td>
</tr>
</tbody>
</table>

1.2× Latency Relative to DRAM
PM SUMMARY

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

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

GPUs excel at performing (relatively simple) repetitive operations on large amounts of data over multiple streams of data.

Target operations that do not require blocking for input or branches:
→ Good: Sequential scans with predicates
→ Bad: B+Tree index probes

AFAIK, GPU memory is **not** cache coherent with CPU memory.
GPU ACCELERATION
GPU ACCELERATION

- PCIe Bus (~16 GB/s)
- DDR4 (~40 GB/s)
- NVLink (~25 GB/s)
GPU ACCELERATION

Choice #1: Entire Database
→ Store the database in the GPU(s) VRAM.
→ All queries perform massively parallel seq scans.

Choice #2: Important Columns
→ Return the offsets of records that match the portion of the query that accesses GPU-resident columns.
→ Must materialize full results in CPU.

Choice #3: Streaming
→ Transfer data from CPU to GPU on the fly.
GPU ACCELERATION

Choice #1: Entire Database
→ Store the database in the GPU(s) VRAM.
→ All queries perform massively parallel seq scans.

Choice #2: Important Columns
→ Return the offsets of records that match the portion of the query that accesses GPU-resident columns.
→ Must materialize full results in CPU.

Choice #3: Streaming
→ Transfer data from CPU to GPU on the fly.
HARDWARE TRANSACTIONAL MEMORY

Create critical sections in software that are managed by hardware.

→ Leverages same cache coherency protocol to detect transaction conflicts.
→ Intel x86: Transactional Synchronization Extensions

Read/write set of transactions must fit in L1 cache.

→ This means that it is not useful for general purpose txns.
→ It can be used to create latch-free indexes.
HTM PROGRAMMING MODEL

Hardware Lock Elision (HLE)
→ Optimistically execute critical section by *eliding* the write to a lock so that it appears to be free to other threads.
→ If there is a conflict, re-execute the code but take locks the second time.

Restricted Transactional Memory (RTM)
→ Like HLE but with an optional fallback codepath that the CPU jumps to if the txn aborts.
HTM LATCH ELISION

Insert Key 25

TSX-START {
  LATCH A
  Read A
  LATCH C
  Read C
  LATCH F
  UNLATCH C
}

TSX-COMMIT
Insert 25
UNLATCH F
HTM LATCH ELISION

Insert Key 25

Insert 25
PARTING THOUGHTS

Byte-addressable PM is going to be a game changer when it comes out.

We are likely to see many new computational components that DBMSs can use in the next decade.
→ The core ideas / algorithms will still be the same.
FINAL PARTING THOUGHTS

You now are aware of the major topics involved in building a modern, single-node DBMS.

You have a foundation for reasoning about systems in order to discern whether claims are legitimate or marketing hype.
You now are aware of the major topics involved in building a modern, single-node DBMS. You have a foundation for reasoning about systems in order to discern whether claims are legitimate or marketing hype.

### Final Parting Thoughts

#### TerminusDB: The Difference

<table>
<thead>
<tr>
<th>Feature</th>
<th>TerminusDB</th>
<th>Stardog</th>
<th>neo4j</th>
<th>mongoDB</th>
<th>Oracle Database</th>
<th>IBM DB2</th>
<th>Lox</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web/ Cloud Native</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Schematic Quality Control</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Geo-Temporal Query</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>A1 Code Generation</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ACID</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>Auto-Sharding</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
</tr>
</tbody>
</table>
FINAL PARTING THOUGHTS

TerminusDB: The Difference

bearjaws 12 days ago [-]

"AI Code Generation"

I see this mentioned in the product comparison chart, but no mention of what that actually means.

reply

chekovcodes 12 days ago [-]

Yes, this is an unfortunately buzzwordy phrase. In this case it does have some meaning though - we generate what we call class-frames from the AI - simple logical javascript programs which know how to render documents and talk to the API, but definitely AI Code generation is not a good phrase.

reply

ggleason 12 days ago [-]

Frankly I think this should be removed. Marketing sometimes get overzealous in trying to present what makes us special.

reply