Lecture #01

Carnegie Mellon University

ADVANCED DATABASE SYSTEMS

In-Memory Databases

@Andy_Pavlo // 15-721 // Spring 2019

TODAY'S AGENDA

Course Logistics Overview
In-Memory DBMS Architectures
Early Notable In-Memory DBMSs

WHY YOU SHOULD TAKE THIS COURSE

DBMS developers are in demand and there are many challenging unsolved problems in data management and processing.

If you are good enough to write code for a DBMS, then you can write code on almost anything else.

[redacted]
salesforce

[redacted]
LinkedIn

[redacted]
Google

[redacted]
snowflake

[redacted]
salesforce

[redacted]
salesforce

[redacted]
amazon

[redacted]
cloudera

[redacted]
HyPer

[redacted]
NUODB

[redacted]
snowflake

[redacted]
salesforce

COURSE OBJECTIVES

Learn about modern practices in database internals and systems programming.

Students will become proficient in:

→ Writing correct + performant code
→ Proper documentation + testing
→ Code reviews
→ Working on a large code base

COURSE TOPICS

The internals of single node systems for in-memory databases. We will ignore distributed deployment problems.

We will cover state-of-the-art topics.
This is **not** a course on classical DBMSs.

COURSE TOPICS

Concurrency Control

Indexing

Storage Models, Compression

Parallel Join Algorithms

Networking Protocols

Logging & Recovery Methods

Query Optimization, Execution, Compilation

BACKGROUND

I assume that you have already taken an intro course on databases (e.g., 15-445/645).

We will discuss modern variations of classical algorithms that are designed for today's hardware.

Things that we will **not** cover:
SQL, Serializability Theory, Relational Algebra, Basic Algorithms + Data Structures.

COURSE LOGISTICS

Course Policies + Schedule:

→ Refer to course web page.

Academic Honesty:

→ Refer to CMU policy page.
→ If you're not sure, ask me.
→ I'm serious. Don't plagiarize or I will wreck you.

OFFICE HOURS

Before class in my office:

→ Mon/Wed: 2:00 – 3:00

→ Gates-Hillman Center 9019

Things that we can talk about:

→ Issues on implementing projects

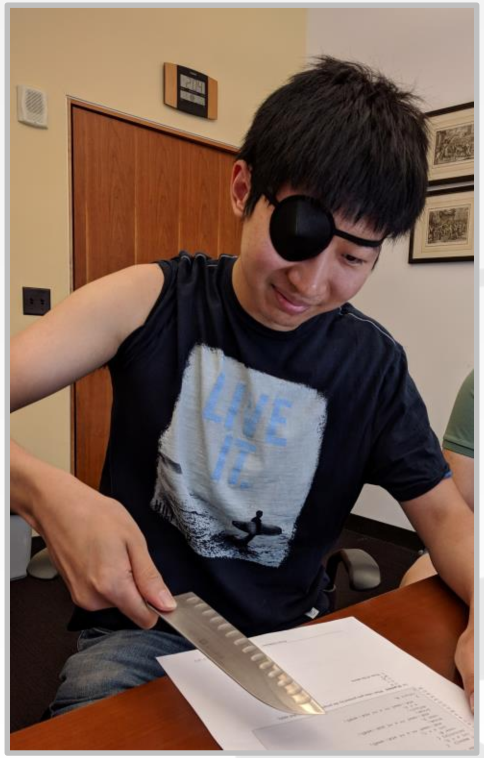
→ Paper clarifications/discussion

→ How to handle the police

TEACHING ASSISTANTS

Head TA: Lin Ma

→ 4th Year PhD Student (CSD)
→ PKU Undergrad
→ Lead architect/developer of self-driving components.
→ Dangerous.



COURSE RUBRIC

Reading Assignments

Programming Projects

Mid-term Exam

Final Exam

Extra Credit

READING ASSIGNMENTS

One mandatory reading per class (★). You can skip **four** readings during the semester.

You must submit a synopsis **before** class:
→ Overview of the main idea (three sentences).
→ System used and how it was modified (one sentence).
→ Workloads evaluated (one sentence).

Submission Form:
https://cmudb.io/15721-s19-submit

☠ PLAGIARISM WARNING ☠

Each review must be your own writing.

You may **not** copy text from the papers or other sources that you find on the web.

Plagiarism will **not** be tolerated.
See CMU's Policy on Academic Integrity for additional information.

PROGRAMMING PROJECTS

Projects will be implemented in CMU's new DBMS "**name to be determined**".

→ In-memory, hybrid DBMS
→ Modern code base (C++17, Multi-threaded, LLVM)
→ Strict coding / documentation standards
→ Open-source / MIT License
→ Postgres-wire protocol compatible

PROGRAMMING PROJECTS

Do all development on your local machine.

→ Peloton only builds on Linux + OSX.

→ We will provide a Vagrant configuration.

Do all benchmarking using Amazon EC2.

→ We will provide details later in semester.

PROJECT #1

Optimize a core component of the DBMS.
We will teach you how to profile the system.

Project #1 will be completed individually.

PROJECT #2

Each group (3 people) will choose a project that is:
→ Relevant to the materials discussed in class.
→ Requires a significant programming effort from all team members.
→ Unique (i.e., two groups cannot pick same idea).
→ Approved by me.

You don't have to pick a topic until after you come back from Spring Break.
We will provide sample project topics.

☠ PLAGIARISM WARNING ☠

These projects must be all of your own code.

You may **not** copy source code from other groups or the web.

Plagiarism will **not** be tolerated.
See CMU's Policy on Academic Integrity for additional information.

PROJECT #3

Project deliverables:

→ Proposal / Specification
→ Project Update / Documentation
→ Code Reviews
→ Testing / Performance Analysis
→ Final Presentation
→ Code Drop

MID-TERM EXAM

Written long-form examination on the mandatory readings and topics discussed in class. Closed notes.

Exam will be given on the last day of class before spring break (Wednesday March 6th).

FINAL EXAM

Take home exam. Harder than the mid-term. Written long-form examination on the mandatory readings and topics discussed in class.

Will be given out on the last day of class (Wednesday May 1st) in this room.

EXTRA CREDIT

We are writing an encyclopedia of DBMSs. Each student can earn extra credit if they write an entry about one DBMS.

→ Must provide citations and attributions.

Additional details will be provided later.

This is optional.

☠ PLAGIARISM WARNING ☠

The extra credit article must be your own writing. You may **not** copy text/images from papers or other sources that you find on the web.

Plagiarism will **not** be tolerated.
See CMU's Policy on Academic Integrity for additional information.

GRADE BREAKDOWN

Reading Reviews (10%)
Project #1 (20%)
Project #2 (50%)
Mid-term Exam (10%)
Final Exam (10%)
Extra Credit (+10%)

COURSE MAILING LIST

On-line Discussion through Piazza:
https://piazza.com/cmu/spring2019/15721

If you have a technical question about the projects, please use Piazza.
→ Don't email me or TAs directly.

All non-project questions should be sent to me.

SPECIAL THANKS

SAP®

IN-MEMORY DATABASES

BACKGROUND

Much of the development history of DBMSs is about dealing with the limitations of hardware.

Hardware was much different when the original DBMSs were designed:
→ Uniprocessor (single-core CPU)
→ RAM was severely limited.
→ The database had to be stored on disk.
→ Disks were even slower than they are now.

BACKGROUND

But now DRAM capacities are large enough that most databases can fit in memory.

→ Structured data sets are smaller.
→ Unstructured or semi-structured data sets are larger.

So why not just use a "traditional" disk-oriented DBMS with a really large cache?

DISK-ORIENTED DBMS

The primary storage location of the database is on non-volatile storage (e.g., HDD, SSD).

→ The database is organized as a set of fixed-length blocks called **slotted pages**.

The system uses an in-memory (volatile) buffer pool to cache blocks fetched from disk.

→ Its job is to manage the movement of those blocks back and forth between disk and memory.

BUFFER POOL

When a query accesses a page, the DBMS checks to see if that page is already in memory:

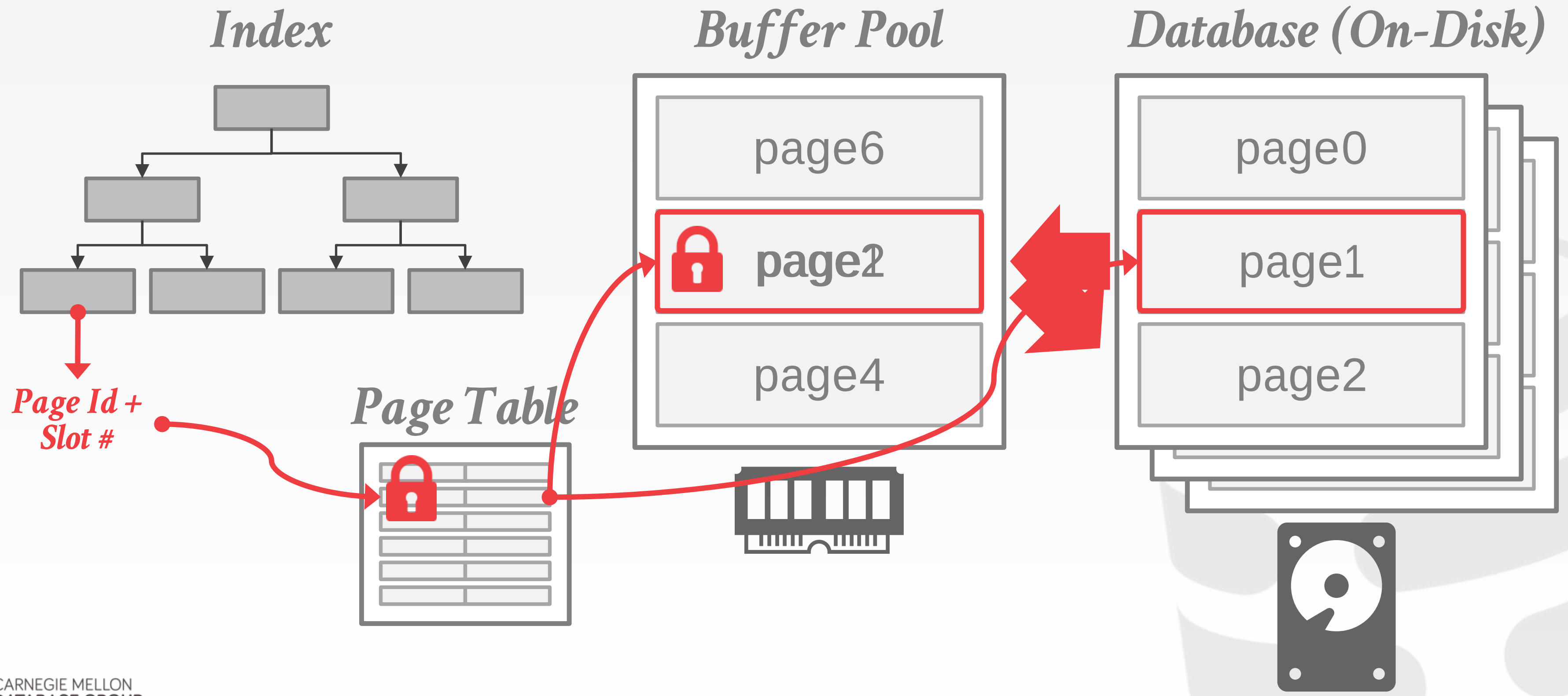
→ If it's not, then the DBMS has to retrieve it from disk and copy it into a frame in its buffer pool.

→ If there are no free frames, then find a page to evict.

→ If the page being evicted is dirty, then the DBMS has to write it back to disk.

Once the page is in memory, the DBMS translates any on-disk addresses to their in-memory addresses.

DISK-ORIENTED DATA ORGANIZATION



BUFFER POOL

Every tuple access has to go through the buffer pool manager regardless of whether that data will always be in memory.

→ Always have to translate a tuple's record id to its memory location.

→ Worker thread has to **pin** pages that it needs to make sure that they are not swapped to disk.

CONCURRENCY CONTROL

In a disk-oriented DBMS, the systems assumes that a txn could stall at any time when it tries to access data that is not in memory.

Execute other txns at the same time so that if one txn stalls then others can keep running.
→ Has to set locks to provide ACID guarantees for txns.
→ Locks are stored in a separate data structure to avoid being swapped to disk.

LOGGING & RECOVERY

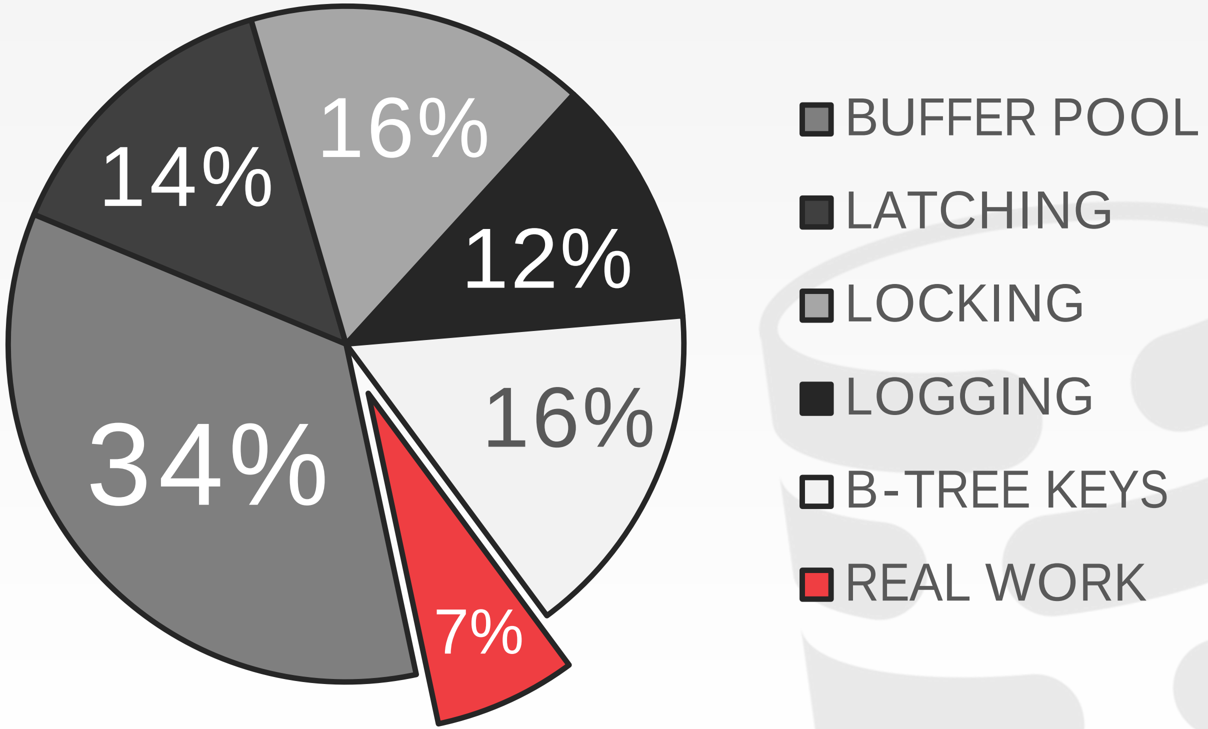
Most DBMSs use **STEAL** + **NO-FORCE** buffer pool policies, so all modifications have to be flushed to the WAL before a txn can commit.

Each log entry contains the before and after image of record modified.

Lots of work to keep track of LSNs all throughout the DBMS.

DISK-ORIENTED DBMS OVERHEAD

Measured CPU Instructions



OLTP THROUGH THE LOOKING GLASS, AND WHAT WE FOUND THERE
SIGMOD 2008

IN-MEMORY DBMSS

Assume that the primary storage location of the database is **permanently** in memory.

Early ideas proposed in the 1980s but it is now feasible because DRAM prices are low and capacities are high.

BOTTLENECKS

If I/O is no longer the slowest resource, much of the DBMS's architecture will have to change account for other bottlenecks:

→ Locking/latching
→ Cache-line misses
→ Pointer chasing
→ Predicate evaluations
→ Data movement & copying
→ Networking (between application & DBMS)

STORAGE ACCESS LATENCIES

	L3	*DRAM*	*SSD*	*HDD*
Read Latency	~20 ns	60 ns	25,000 ns	10,000,000 ns
Write Latency	~20 ns	60 ns	300,000 ns	10,000,000 ns

LET'S TALK ABOUT STORAGE & RECOVERY METHODS FOR
NON-VOLATILE MEMORY DATABASE SYSTEMS
SIGMOD 2015

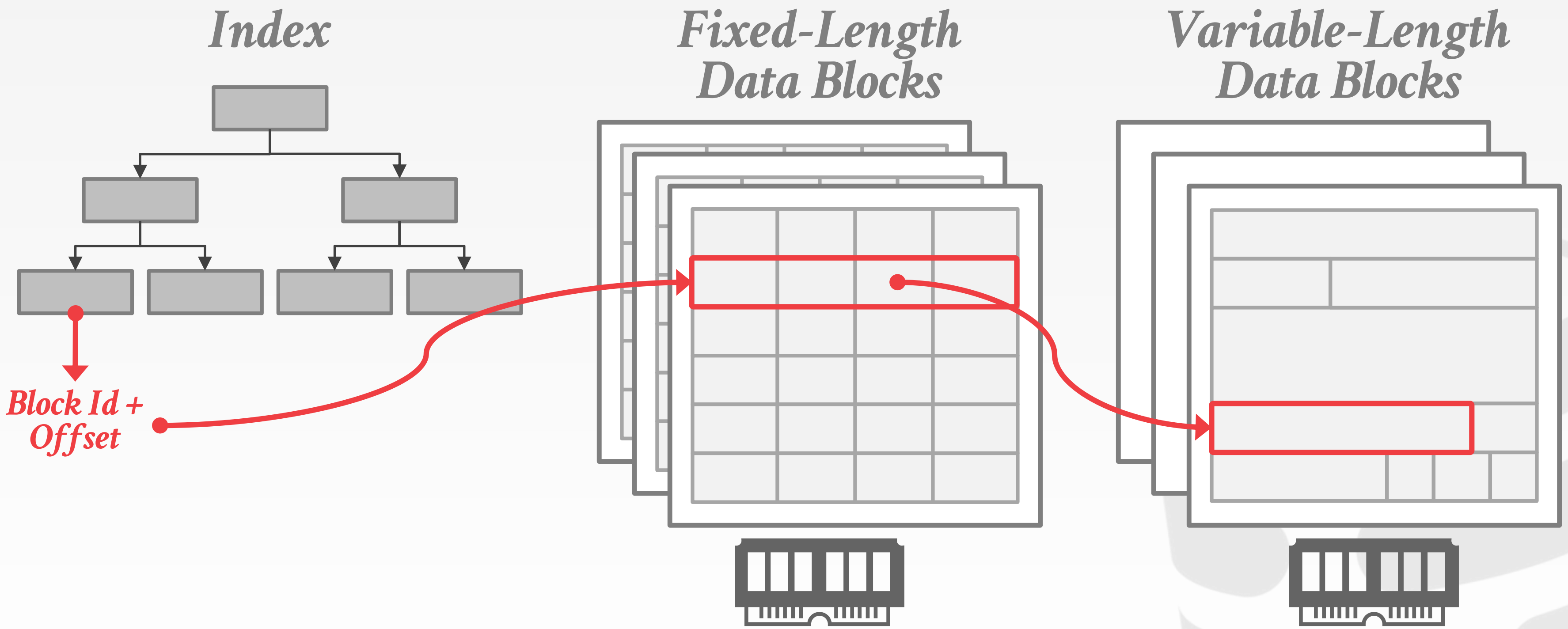
DATA ORGANIZATION

An in-memory DBMS does not need to store the database in slotted pages but it will still organize tuples in blocks/pages:

- → Direct memory pointers vs. record ids
- → Fixed-length vs. variable-length data pools
- → Use checksums to detect software errors from trashing the database.

The OS organizes memory in pages too. We will cover this later.

IN-MEMORY DATA ORGANIZATION



WHY NOT MMAP?

Memory-map (**mmap**) a database file into DRAM and let the OS be in charge of swapping data in and out as needed.

Use **madvise** and **msync** to give hints to the OS about what data is safe to flush.

Notable **mmap** DBMSs:

→ MongoDB (pre WiredTiger)
→ MonetDB
→ LMDB
→ MemSQL

WHY NOT MMAP?

Using **`mmap`** gives up fine-grained control on the contents of memory.

→ Cannot perform non-blocking memory access.
→ The "on-disk" representation has to be the same as the "in-memory" representation.
→ The DBMS has no way of knowing what pages are in memory or not.
→ Various mmap-related syscalls are not portable.

A well-written DBMS **always** knows best.

CONCURRENCY CONTROL

Observation: The cost of a txn acquiring a lock is the same as accessing data.

In-memory DBMS may want to detect conflicts between txns at a different granularity.

→ **Fine-grained locking** allows for better concurrency but requires more locks.

→ **Coarse-grained locking** requires fewer locks but limits the amount of concurrency.

CONCURRENCY CONTROL

The DBMS can store locking information about each tuple together with its data.
→ This helps with CPU cache locality.
→ Mutexes are too slow. Need to use CAS instructions.

New bottleneck is contention caused from txns trying access data at the same time.

INDEXES

Specialized main-memory indexes were proposed in 1980s when cache and memory access speeds were roughly equivalent.

But then caches got faster than main memory:
→ Memory-optimized indexes performed worse than the B+trees because they were not cache aware.

Indexes are usually rebuilt in an in-memory DBMS after restart to avoid logging overhead.

QUERY PROCESSING

The best strategy for executing a query plan in a DBMS changes when all of the data is already in memory.

→ Sequential scans are no longer significantly faster than random access.

The traditional **tuple-at-a-time** iterator model is too slow because of function calls.

→ This problem is more significant in OLAP DBMSs.

LOGGING & RECOVERY

The DBMS still needs a WAL on non-volatile storage since the system could halt at anytime.

→ Use **group commit** to batch log entries and flush them together to amortize `fsync` cost.

→ May be possible to use more lightweight logging schemes (e.g., only store redo information).

But since there are no "dirty" pages, there is no need to maintain LSNs all throughout the system.

LOGGING & RECOVERY

The system also still takes checkpoints to speed up recovery time.

Different methods for checkpointing:

→ Maintain a second copy of the database in memory that is updated by replaying the WAL.
→ Switch to a special "copy-on-write" mode and then write a dump of the database to disk.
→ Fork the DBMS process and then have the child process write its contents to disk.

LARGER-THAN-MEMORY DATABASES

DRAM is fast, but data is not accessed with the same frequency and in the same manner.
→ Hot Data: OLTP Operations
→ Cold Data: OLAP Queries

We will study techniques for how to bring back disk-resident data without slowing down the entire system.

NOTABLE IN-MEMORY DBMSs

- ➡ Oracle TimesTen
- ➡ Dali / DataBlitz
- Altibase
- ➡ P*TIME
- SAP HANA
- VoltDB / H-Store
- Microsoft Hekaton
- Harvard Silo
- TUM HyPer
- MemSQL
- IBM DB2 BLU
- Apache Geode

TIMESTEN

Originally SmallBase from HP Labs in 1995.

Multi-process, shared memory DBMS.

→ Single-version database using two-phase locking.
→ Dictionary-encoded columnar compression.

Bought by Oracle in 2005.

Can work as a cache in front of Oracle DBMS.

ORACLE TIMESTEN: AN IN-MEMORY DATABASE FOR ENTERPRISE APPLICATIONS
VLDB 2004

DALI / DATABLITZ

Developed at AT&T Labs in the early 1990s.

Multi-process, shared memory storage manager using memory-mapped files.

Employed additional safety measures to make sure that erroneous writes to memory do not corrupt the database.

→ Meta-data is stored in a non-shared location.

→ A page's checksum is always tested on a read; if the checksum is invalid, recover page from log.

DALI: A HIGH PERFORMANCE MAIN MEMORY STORAGE MANAGER
VLDB 1994

P*TIME

Korean in-memory DBMS from the 2000s.

Performance numbers are still impressive.

Lots of interesting features:

→ Uses differential encoding (XOR) for log records.
→ Hybrid storage layouts.
→ Support for larger-than-memory databases.

Sold to SAP in 2005. Now part of HANA.

P*TIME: HIGHLY SCALABLE OLTP DBMS FOR MANAGING UPDATE-INTENSIVE STREAM WORKLOAD
VLDB 2004

PARTING THOUGHTS

The design of a in-memory DBMS is significantly different than a disk-oriented system.

The world has finally become comfortable with in-memory data storage and processing.

Never use `mmap` for your DBMS.

NEXT CLASS

Transaction Programming Models

Isolation Levels

Modern Concurrency Control

Make sure that you submit the first reading review