Lecture #13

Carnegie Mellon University

ADVANCED DATABASE SYSTEMS

Networking

@Andy_Pavlo // 15-721 // Spring 2019

ADMINISTRIVIA

Feb 27: Project #1 is due

Feb 27: Project #2 will be released

Mar 4: Extra Credit assignment will be released

Mar 6: Mid-term Exam

Mar 18: Project #2 Proposals

MID-TERM

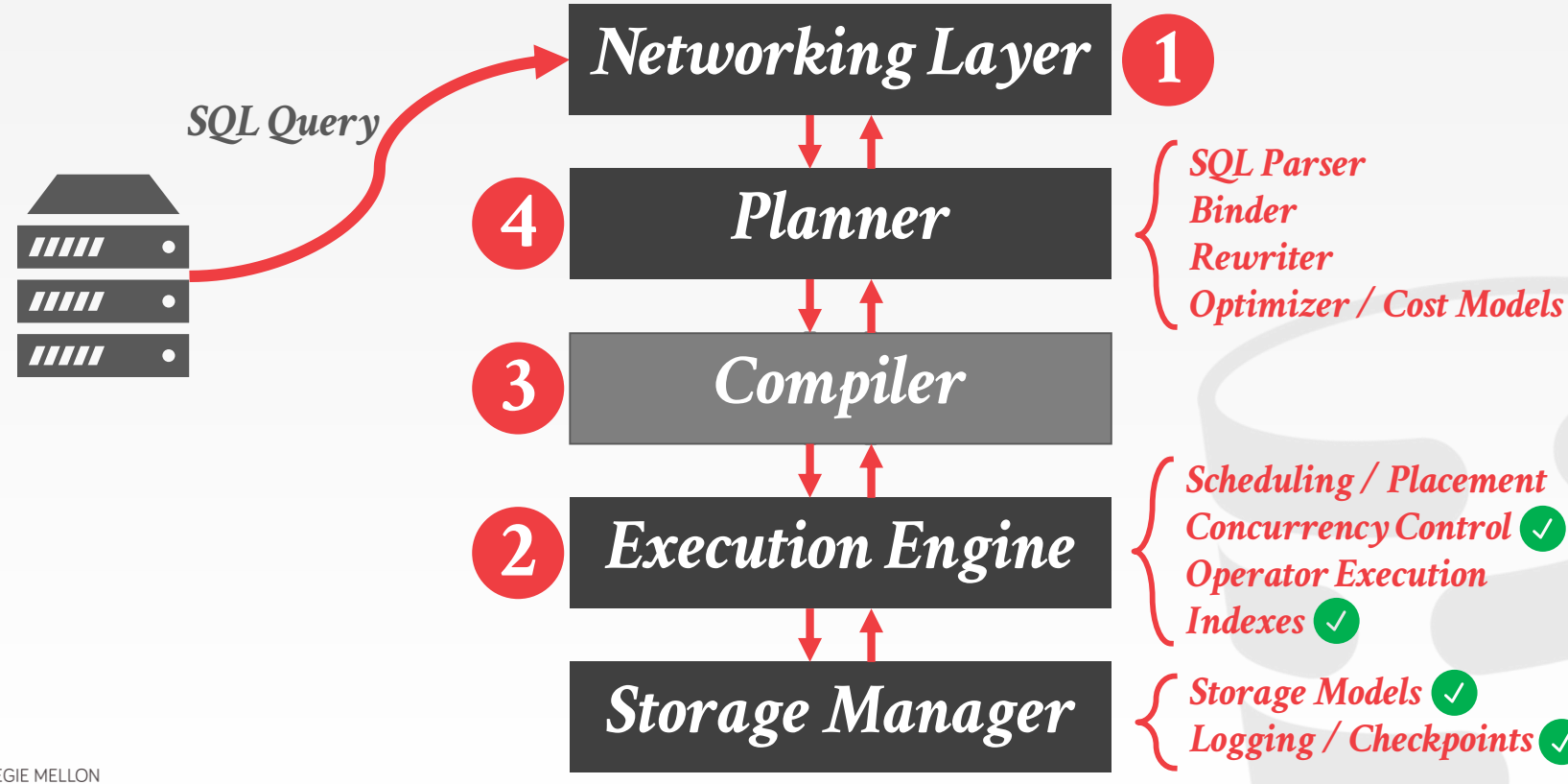
March 6th @ 3:00pm in this room.

Mix of multiple choice and short-answer questions

Closed notes. I don't care about paper minutiae.

Materials Covered:
→ Lecture #01 to #12 (inclusive)

ARCHITECTURE OVERVIEW



TODAY'S AGENDA

Database Access APIs
Database Network Protocols
Kernel Bypass Methods
Project #2 Topics

DATABASE ACCESS

All of the demos in the class have been through a terminal client.

→ SQL queries are written by hand.
→ Results are printed to the terminal.

Real programs access a database through an API:

→ Direct Access (DBMS-specific)
→ Open Database Connectivity (ODBC)
→ Java Database Connectivity (JDBC)

OPEN DATABASE CONNECTIVITY

Standard API for accessing a DBMS. Designed to be independent of the DBMS and OS.

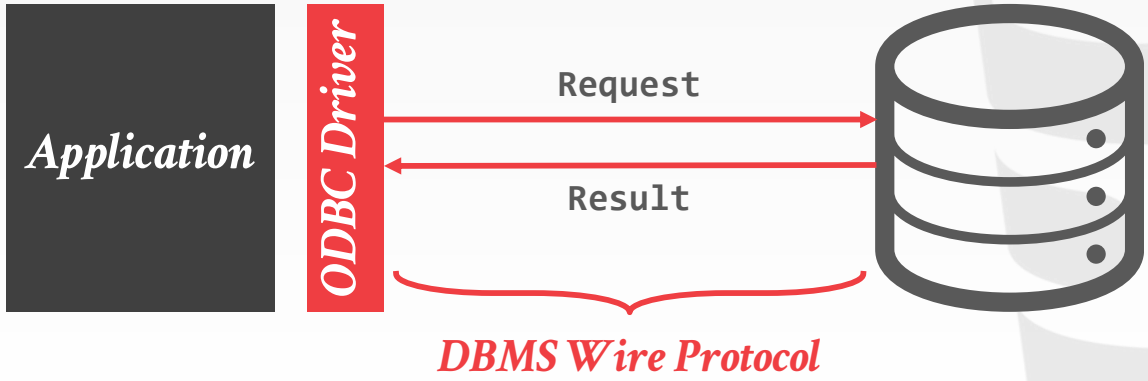
Originally developed in the early 1990s by Microsoft and Simba Technologies.

Every major relational DBMS now has an ODBC implementation.

OPEN DATABASE CONNECTIVITY

ODBC is based on the device driver model.

The **driver** encapsulates the logic needed to convert a standard set of commands into the DBMS-specific calls.



JAVA DATABASE CONNECTIVITY

Developed by Sun Microsystems in 1997 to provide a standard API for connecting a Java program with a DBMS.

JDBC can be considered a version of ODBC for the programming language Java instead of C.

JAVA DATABASE CONNECTIVITY

Approach #1: JDBC-ODBC Bridge
→ Convert JDBC method calls into ODBC function calls.

Approach #2: Native-API Driver
→ Convert JDBC method calls into native calls of the target DBMS API.

Approach #3: Network-Protocol Driver
→ Driver connects to a middleware that converts JDBC calls into a vendor-specific DBMS protocol.

Approach #4: Database-Protocol Driver
→ Pure Java implementation that converts JDBC calls directly into a vendor-specific DBMS protocol.

DATABASE NETWORKING PROTOCOLS

All major DBMSs implement their own proprietary wire protocol over TCP/IP.

A typical client/server interaction:

→ Client connects to DBMS and begins authentication process. There may be an SSL handshake.
→ Client then sends a query.
→ DBMS executes the query, then serializes the results and sends it back to the client.

EXISTING PROTOCOLS

Most newer systems implement one of the open-source DBMS wire protocols. This allows them to reuse the client drivers without having to develop and support them.

Just because on DBMS "speaks" another DBMS's wire protocol does not mean that it is compatible.

→ Need to also support catalogs, SQL dialect, and other functionality.

EXISTING PROTOCOLS

PROTOCOL DESIGN SPACE

Row vs. Column Layout

Compression

Data Serialization

String Handling

DON'T HOLD MY DATA HOSTAGE: A CASE FOR CLIENT PROTOCOL REDESIGN
VLDB 2017

ROW VS. COLUMN LAYOUT

ODBC/JDBC are inherently row-oriented APIs.

→ The DBMS packages tuples into messages one tuple at a time.

→ The client has to deserialize data one tuple at a time.

But modern data analysis software operates on matrices and columns.

One potential solution is to send data in vectors.

→ Batch of rows organized in a column-oriented layout.

COMPRESSION

Approach #1: Naïve Compression

Approach #2: Columnar-Specific Encoding

More heavyweight compression is better when the network is slow.

Better compression ratios for larger message chunk sizes.

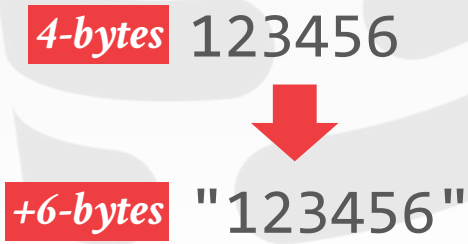
DATA SERIALIZATION

Approach #1: Binary Encoding

→ Have to handle endian conversion on client.
→ The closer the serialized format is to the DBMS's binary format, then the lower the overhead to serialize.
→ DBMS can implement its own format or rely on existing libraries (ProtoBuf, Thrift).

Approach #2: Text Encoding

→ Convert all binary values into strings (`atoi`).
→ Do not have to worry about endianness.



STRING HANDLING

Approach #1: Null Termination

→ Store a null byte (`'\0'`) to denote the end of a string.
→ Client has to scan the entire string to find end.

Approach #2: Length-Prefixes

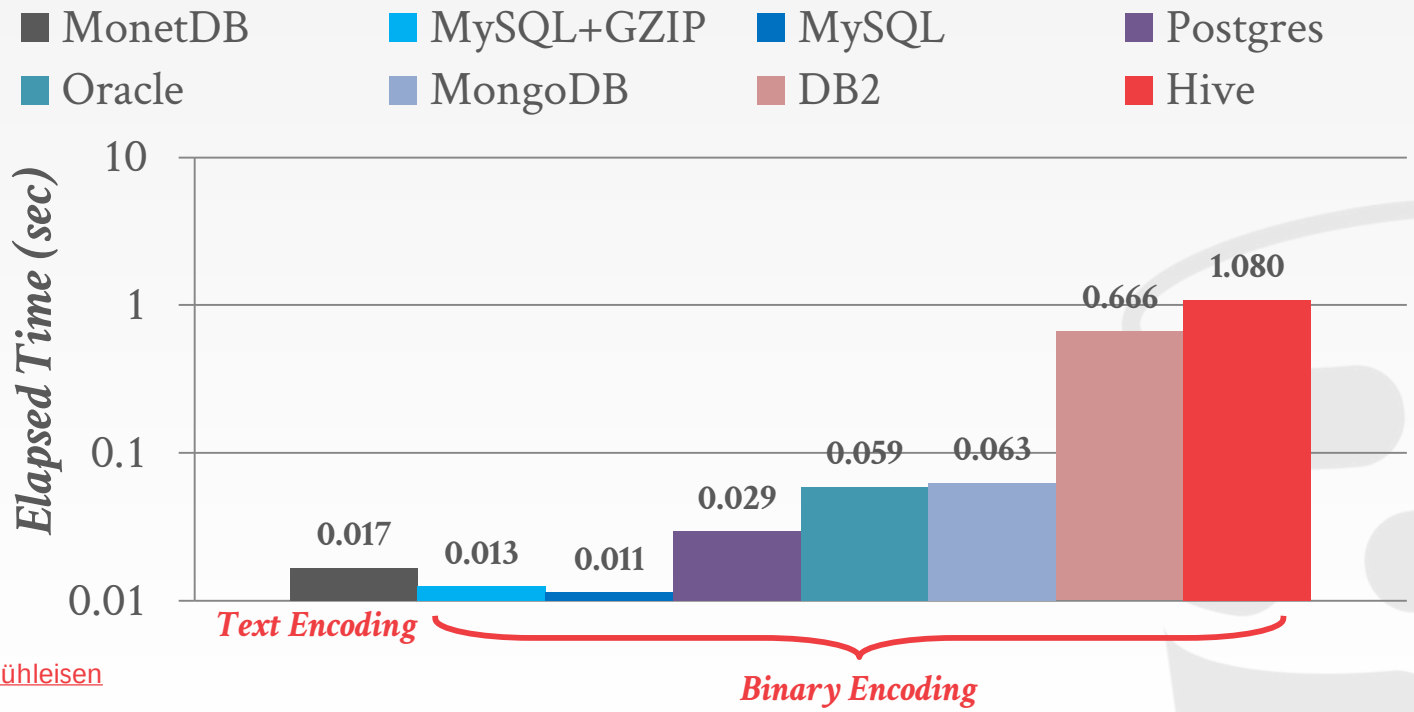
→ Add the length of the string at the beginning of the bytes.

Approach #3: Fixed Width

→ Pad every string to be the max size of that attribute.

NETWORK PROTOCOL PERFORMANCE

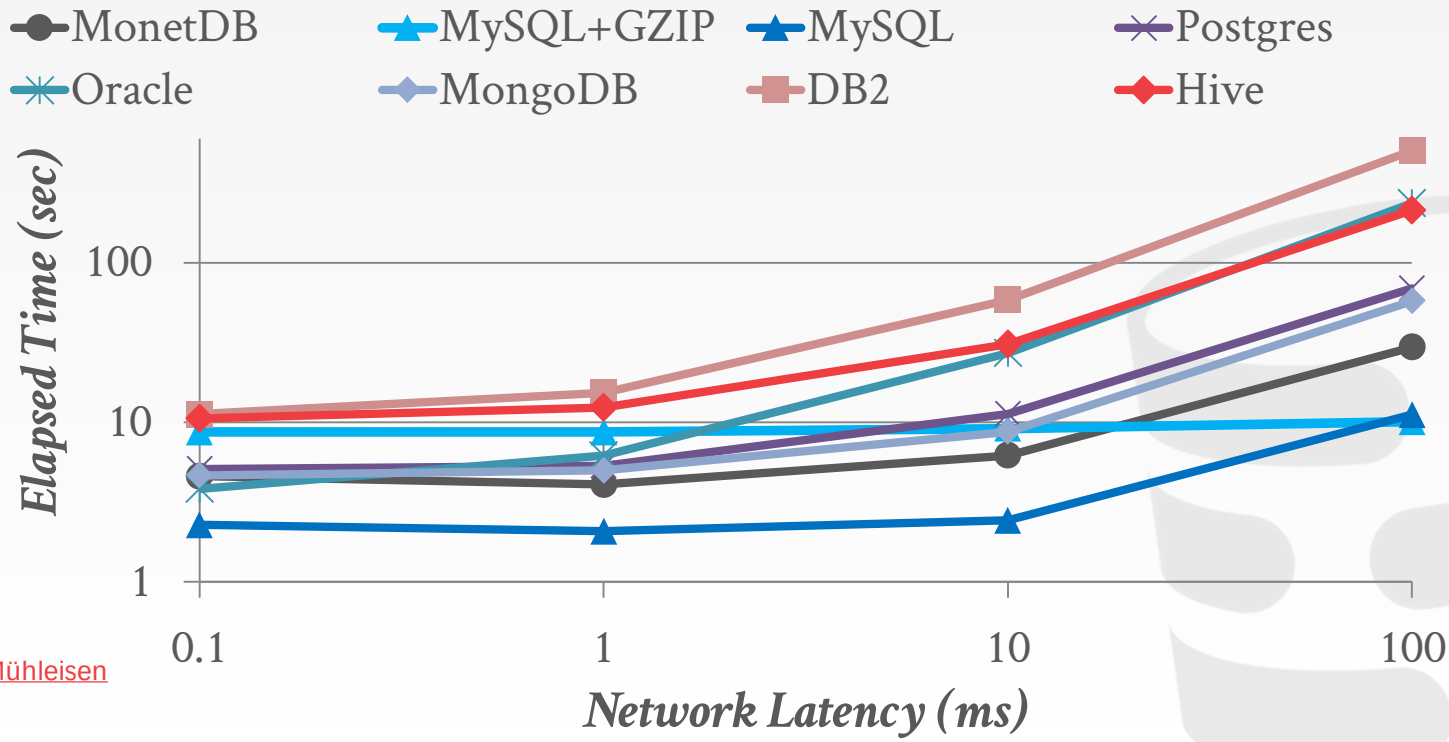
Transfer One Tuple from TCP-H LINEITEM



Source: Hannes Mühleisen

NETWORK PROTOCOL PERFORMANCE

Transfer 1m Tuples from TCP-H LINEITEM



Source: Hannes Mühleisen

OBSERVATION

The DBMS's network protocol implementation is not the only source of slowdown.

The OS's TCP/IP stack is slow…
→ Expensive context switches / interrupts
→ Data copying
→ Lots of latches in the kernel

KERNEL BYPASS METHODS

Allows the system to get data directly from the NIC into the DBMS address space.

→ No unnecessary data copying.
→ No OS TCP/IP stack.

Approach #1: Data Plane Development Kit

Approach #2: Remote Direct Memory Access

DATA PLANE DEVELOPMENT KIT (DPDK)

Set of libraries that allows programs to access NIC directly. Treat the NIC as a bare metal device.

Requires the DBMS code to do more to manage memory and buffers.

→ No data copying.
→ No system calls.

Example: ScyllaDB

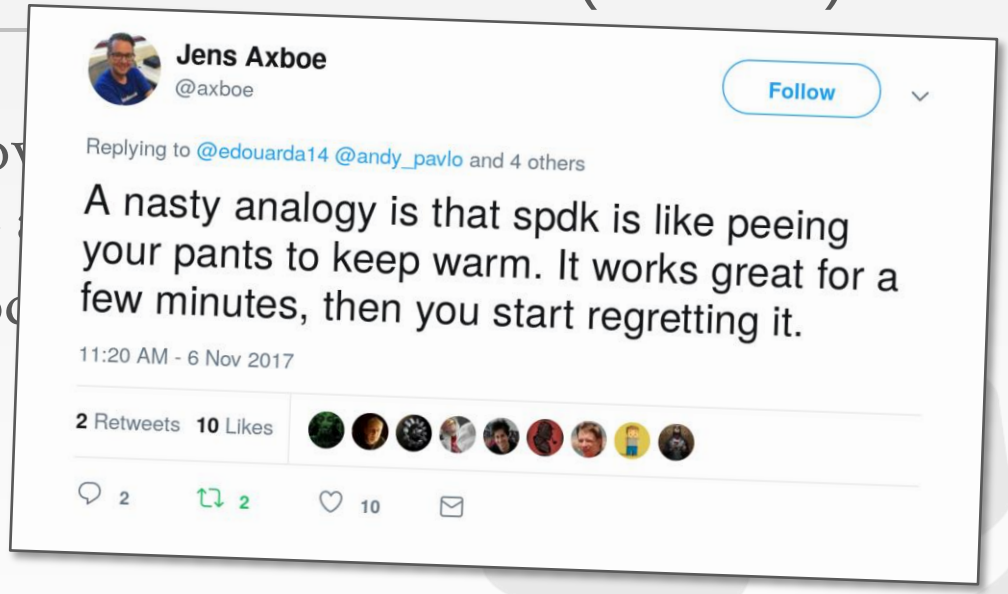
DATA PLANE DEVELOPMENT KIT (DPDK)

Set of libraries that allo
directly. Treat the NIC

Requires the DBMS co
memory and buffers.

→ No data copying.
→ No system calls.

Example: ScyllaDB



REMOTE DIRECT MEMORY ACCESS

Read and write memory directly on a remote host without going through OS.

→ The client needs to know the correct address of the data that it wants to access.

→ The server is unaware that memory is being accessed remotely (i.e., no callbacks).

Example: Oracle RAC, Microsoft FaRM

PARTING THOUGHTS

A DBMS's networking protocol is an often overlooked bottleneck for performance.

Kernel bypass methods greatly improve performance but require more bookkeeping.
→ Probably more useful for internal DBMS communication.

PROJECT #2

Group project to implement some substantial component or feature in a DBMS.

Projects should incorporate topics discussed in this course as well as from your own interests.

Each group must pick a project that is unique from their classmates.

PROJECT #2

Project deliverables:

→ Proposal
→ Status Update
→ Design Document
→ Code Review
→ Final Presentation
→ Code Drop

PROJECT #2 – PROPOSAL

Five minute presentation to the class that discusses the high-level topic.

Each proposal must discuss:

→ What files you will need to modify.
→ How you will test whether your implementation is correct.
→ What workloads you will use for your project.

PROJECT #2 – STATUS UPDATE

Five minute presentation to update the class about the current status of your project.

Each presentation should include:

→ Current development status.
→ Whether your plan has changed and why.
→ Anything that surprised you during coding.

PROJECT #2 – DESIGN DOCUMENT

As part of the status update, you must provide a design document that describes your project implementation:

→ Architectural Design
→ Design Rationale
→ Testing Plan
→ Trade-offs and Potential Problems
→ Future Work

PROJECT #2 – CODE REVIEW

Each group will be paired with another group and provide feedback on their code.

There will be two separate code review rounds.

Grading will be based on participation.

PROJECT #2 – FINAL PRESENTATION

10 minute presentation on the final status of your project during the scheduled final exam.

You'll want to include any performance measurements or benchmarking numbers for your implementation.

Demos are always hot too…

PROJECT #2 – CODE DROP

A project is **not** considered complete until:

→ The code can merge into the master branch without any conflicts.
→ All comments from code review are addressed.
→ The project includes test cases that correctly verify that implementation is correct.
→ Source code contains clear documentation / comments.

We will select the merge order randomly.

COMPUTING RESOURCES

We will provide additional Amazon AWS credits.

Submitting a PR to our repo will invoke builds on Travis and local Jenkins cluster.

Let me know if you think you need special hardware.

DISCLAIMER

The DBMS is a major work-in-progress.

We do not support a bunch of things yet.

We should work together to add in features / components that we all need.

PROJECT TOPICS

Query Optimizer

Add/Drop Index

Metric Collection

Settings + Triggers

Checkpoints + Recovery

Unified GC

HANA-style Interval GC

Performance Opt.

LLVM Engine

LLVM Index Scans

QUERY OPTIMIZER

We have a sophisticated query optimizer based on the Cascades model.

Project: Expand features in DBMS optimizer

→ Outer Joins
→ Expression Rewriting
→ Cost Models
→ Note: You have to send me your CV if you choose this project because companies want to hire you. Seriously.

ADD/DROP INDEXES

We need to support building indexes in a transactionally consistent manner.

Project: Correct index creation/deletion

→ Maintain a delta storage for capturing changes made to table while the index is being built.
→ Temporarily halt txns when the index is built and then apply missed changes.
→ Bonus: Support building indexes with multiple threads.

METRIC COLLECTION

We need to collect internal metrics about the DBMS's runtime performance for the self-driving components.

Project: Build infrastructure for metrics.
→ Add hooks for txns + storage
→ Collect metrics during execution.
→ We have an previous implementation in Peloton.

SETTINGS + INTERNAL TRIGGERS

The system should store all of its configuration knobs in the catalog. But a table interface is not ideal for C++ components.

Project: Build new SettingsManager

→ Mimic Postgres catalog structure (`pg_settings`)
→ Changing the catalog should invoke an internal trigger that reconfigures the system.
→ Need to support scaling up/down threads.
→ Keep track of deployment times.

CHECKPOINTS + RECOVERY

We currently support a WAL scheme without checkpoints. We can replay log upon restart but not re-install catalogs.

Project: Implement full recovery from checkpoints + WAL.

→ First store catalog table in checkpoint
→ Add recovery from checkpoint + WAL.
→ Implement consistent checkpoints for data tables.

UNIFIED GARBAGE COLLECTION

We have a separate epoch-based GC mechanisms for data tables and the Bw-Tree.

Project: Build a unified GC component.

→ Examine existing GC methods in Bw-Tree
→ Expose epoch management in index layer.
→ Push epoch updates to index during runtime.

INTERVAL GARBAGE COLLECTION

We currently only implement basic min-timestamp MVCC garbage collection. Interval GC is necessary for OLAP queries.

Project: Implement HANA-style Interval GC

→ Low-overhead interval identification.
→ Will not match exactly with the scheme described in the SAP paper.
→ May need to work together with the checkpoint team.

PERFORMANCE ANALYSIS + OPTIMIZATION

We have not performed a thorough evaluation of the other parts of the DBMS.
If you liked Project #1, then you can go deeper…

Project: Analyze + Optimize System
→ Identify additional scalability bottlenecks.
→ Expand our multi-threaded testing framework to include complex insert, update, and delete workloads.
→ Integrate tests into performance regression framework.

LLVM ENGINE INTEGRATION

We have a new execution engine in separate repository. It uses the LLVM to compile query plans into machine code for fast execution.

Project: Port LLVM Engine to DBMS

→ Convert plan nodes + expressions to engine's DSL.
→ Modify engine to interface with storage manager.
→ Build out multi-threading execution infrastructure.
→ We don't have an optimizer yet, so you will have to test with hand crafted query plans loaded from JSON.

LLVM INDEX SCAN

Current LLVM engine only support sequential scans. It does not know how to access indexes.

Project: Support Index Scans in LLVM Engine
→ Extend engine's DSL to support index scans.
→ Have to infer key format from catalog + index schema.
→ Support both point queries + range scans.
→ You will have to load the indexes manually.

HOW TO START

Form a team.

Meet with your team and discuss potential topics.

Look over source code and determine what you will need to implement.

I am able during Spring Break for additional discussion and clarification of the project idea.

NEXT CLASS

Let's start to talk about how to execute queries!