Hello !!

CS 744: SCOPE

Shivaram Venkataraman

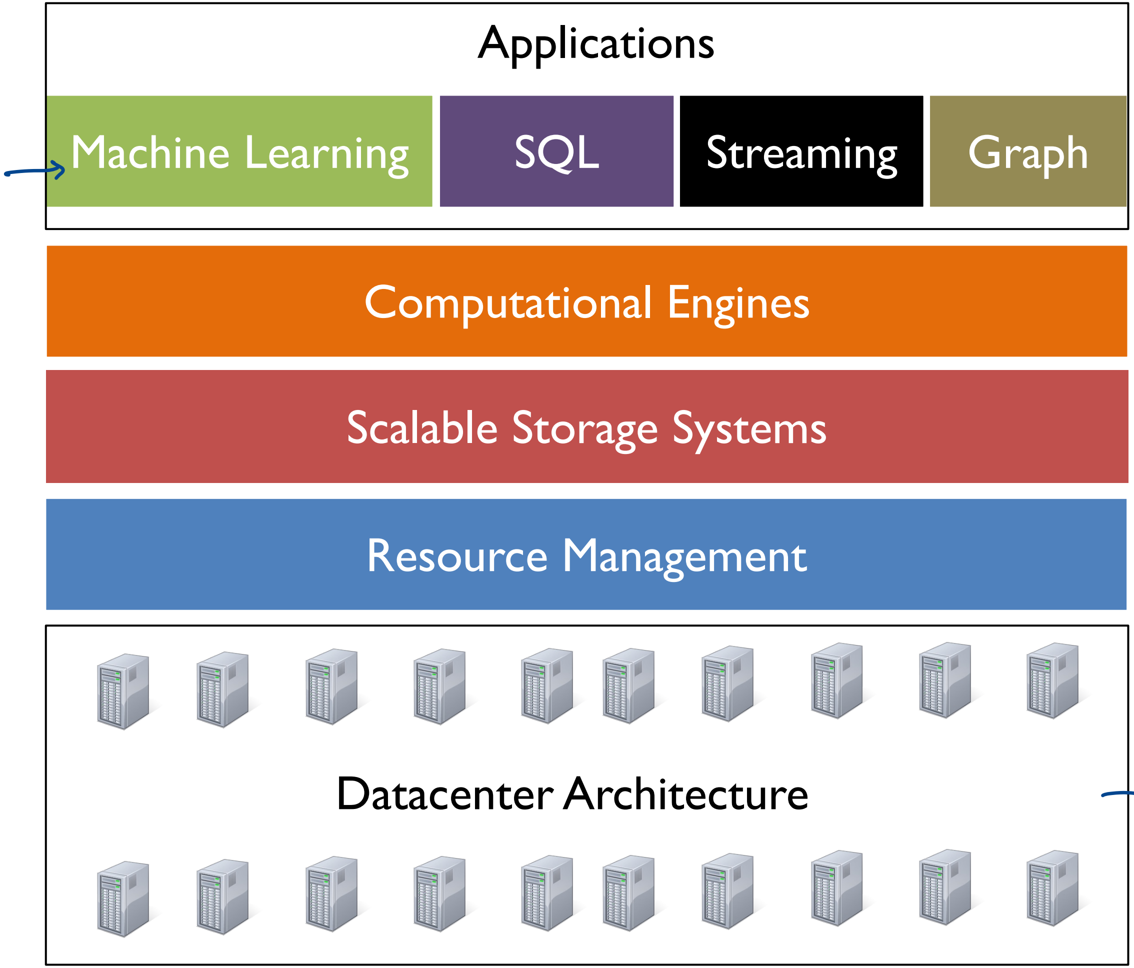
Spring 2024

ADMINISTRIVIA

- Course Project Proposal: Due soon!

- Midterm details are on Piazza. → one week

- No reviews for Tuesday (Snowflake)!



Applications

Machine Learning

SQL

Streaming

Graph

Computational Engines

Scalable Storage Systems

Resource Management

Datacenter Architecture

PyTorch
PipeDream
vLLM

Spark, MR

GFS

Mesos, Gavel
DRF InFaaS

SQL: STRUCTURED QUERY LANGUAGE

~ 1970s

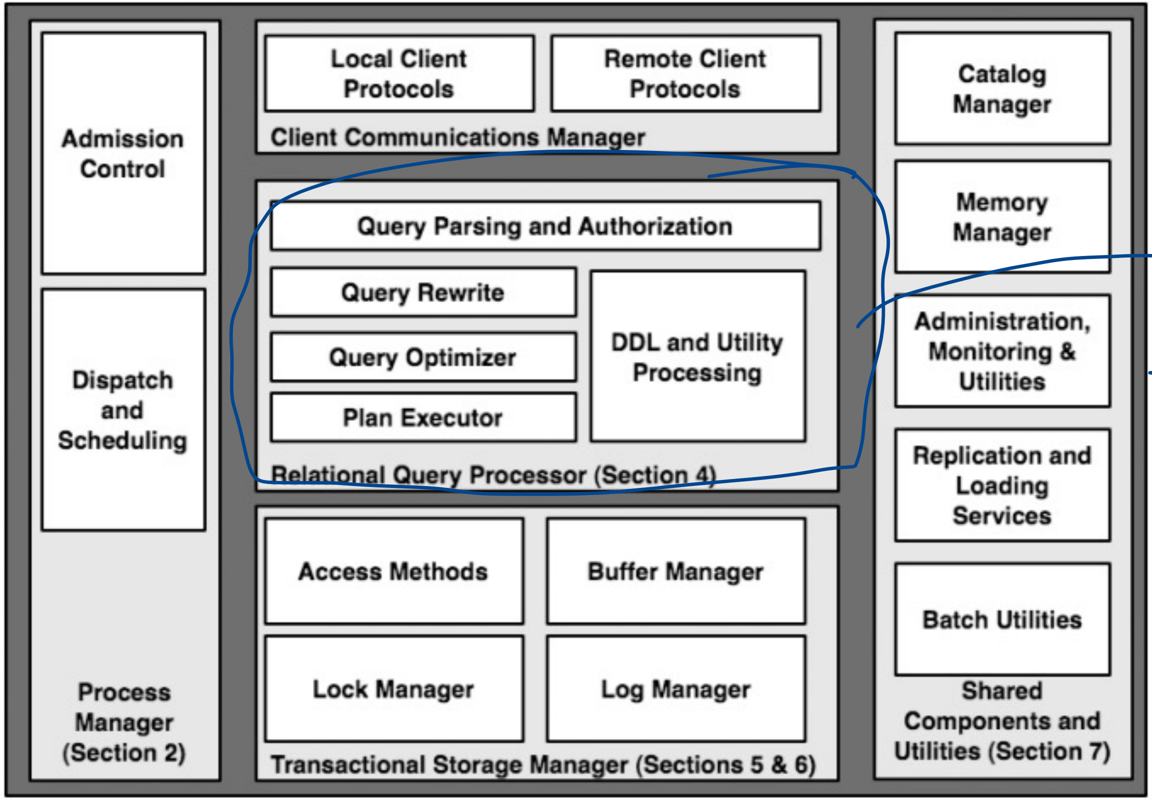
DATABASE SYSTEMS

short lived and perform modifications to DBMS

SQL queries

OLTP

↳ Transactions in an online setting web front



Local Client Protocols

Remote Client Protocols

Client Communications Manager

Query Parsing and Authorization

Query Rewrite

Query Optimizer

Plan Executor

DDL and Utility Processing

Relational Query Processor (Section 4)

Access Methods

Buffer Manager

Lock Manager

Log Manager

Transactional Storage Manager (Sections 5 & 6)

Admission Control

Dispatch and Scheduling

Process Manager (Section 2)

Catalog Manager

Memory Manager

Administration, Monitoring & Utilities

Replication and Loading Services

Batch Utilities

Shared Components and Utilities (Section 7)

Set of rows

OLAP

↳ Analytics

→ long queries and read only

PROCEDURAL VS. RELATIONAL

How the query is executed

don't

```
lines = sc.textFile("users")
csv = lines.map(x =>
    x.split(‘,’))
young = csv.filter(x =>
    x(1) < 21)
println(young.count())
```

string int

Name , age

.... , 10

.... , 23

.... , 45

Schema → metadata the organization of data

Name: String , Age: Int

```
SELECT COUNT(*)
FROM “users”
WHERE age < 21
```

→ more succinct

captured in 1 query

" Human friendly "

→ what needs to be executed

SCOPE / SparkSQL

↳ best of both worlds

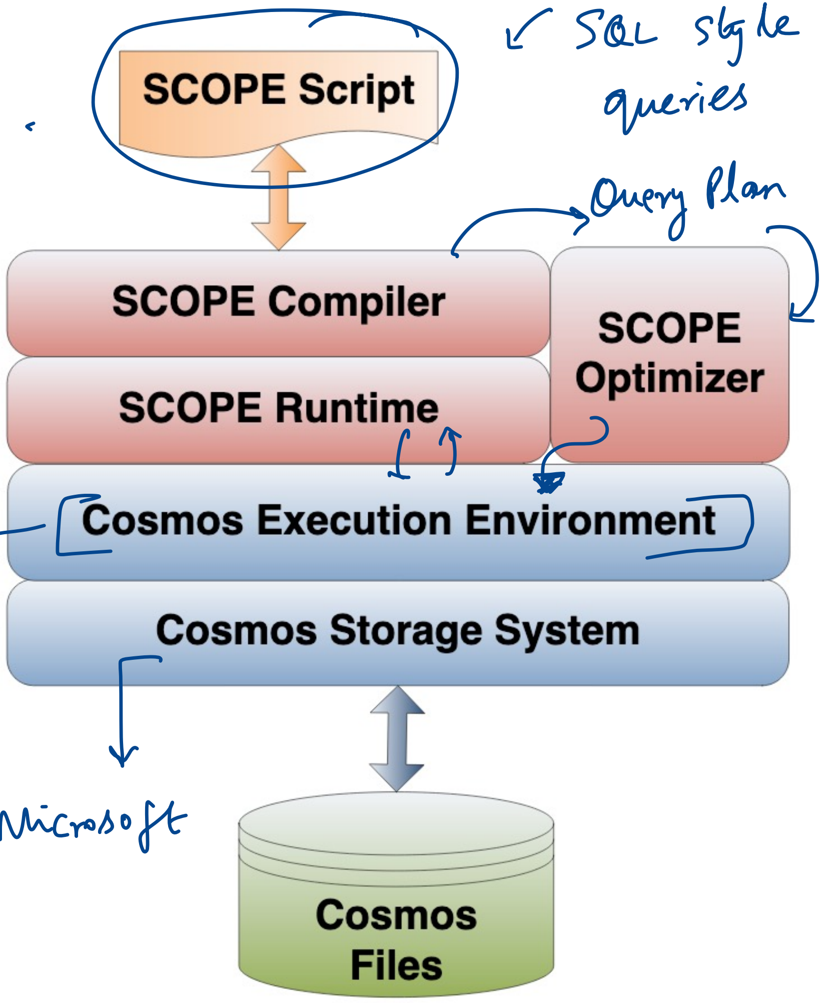
```
SELECT query, COUNT(*)
AS count
FROM "search.log"

USING LogExtractor
GROUP BY query
HAVING count > 1000
ORDER BY count DESC;
```

similar to SQL

familiar to users

MR



↙ SQL style queries

SCOPE OPERATORS

Raw Files in FS

↳ Structured table used by rest of the query

Input reading: What is different?

Schema

```
EXTRACT column[:<type>][, ...]
FROM <input_stream(s) >
USING <Extractor> [(args)]
[HAVING <predicate>]
```

files

filter out data at the source

Custom user defined class that parses one row from the file

SQL OPERATORS

Select – read rows that satisfy some predicate

Join – Support for Inner and Outer join

Subset of SQL Standard

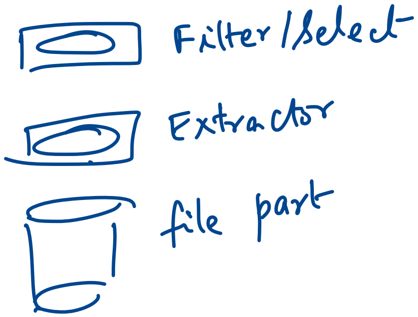
GroupBy – Group by some column

OrderBy – Sorting the output

Aggregations – COUNT, SUM, MAX etc.

natively implemented in the system

Ease to use for data analysts



LANGUAGE INTEGRATION

Language Integrated Queries (LINQ)

plus

C# Trim function

```
R1 = SELECT A+C AS ac, B.Trim() AS B1
FROM R
WHERE StringOccurs(C,“xyz”) > 2
```

C # function defined inline

```
#CS
public static int StringOccurs(string str, string ptrn){
 int cnt=0; int pos=-1;
 while (pos+1 < str.Length) {
    pos = str.IndexOf(ptrn, pos+1);
    if (pos < 0) break;
    cnt++;
  }
  return cnt;
}
#ENDCS
```

import

black box to the query optimizer

compile binary

ship this to all tasks run. Format/ deserialize so that functions can run.

MAPREDUCE-LIKE?

Process (Map) UDF that takes in input rows → One or more output rows (with schema)

Reduce ↳ grouped data

All rows that belong to group → One or more rows (with schema)

Combine → NOT THE SAME

```
COMBINE S1 WITH S2
ON S1.A==S2.A AND S1.B==S2.B AND S1.C==S2.C
USING MultiSetDifference
PRODUCE A, B, C
```

Two tables which are co-partitioned.
one output table

EXECUTION: COMPILER

```
SELECT query, COUNT() AS count
FROM "search.log"
USING LogExtractor
GROUP BY query
HAVING count > 1000
ORDER BY count DESC;
```

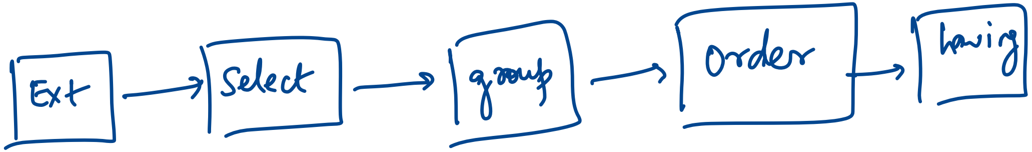
Check syntax, resolve names

Checks if columns have been defined

Result: Internal parse tree

check column types match up

logical query plan



Logical → Optimized plan that can be executed

OPTIMIZER

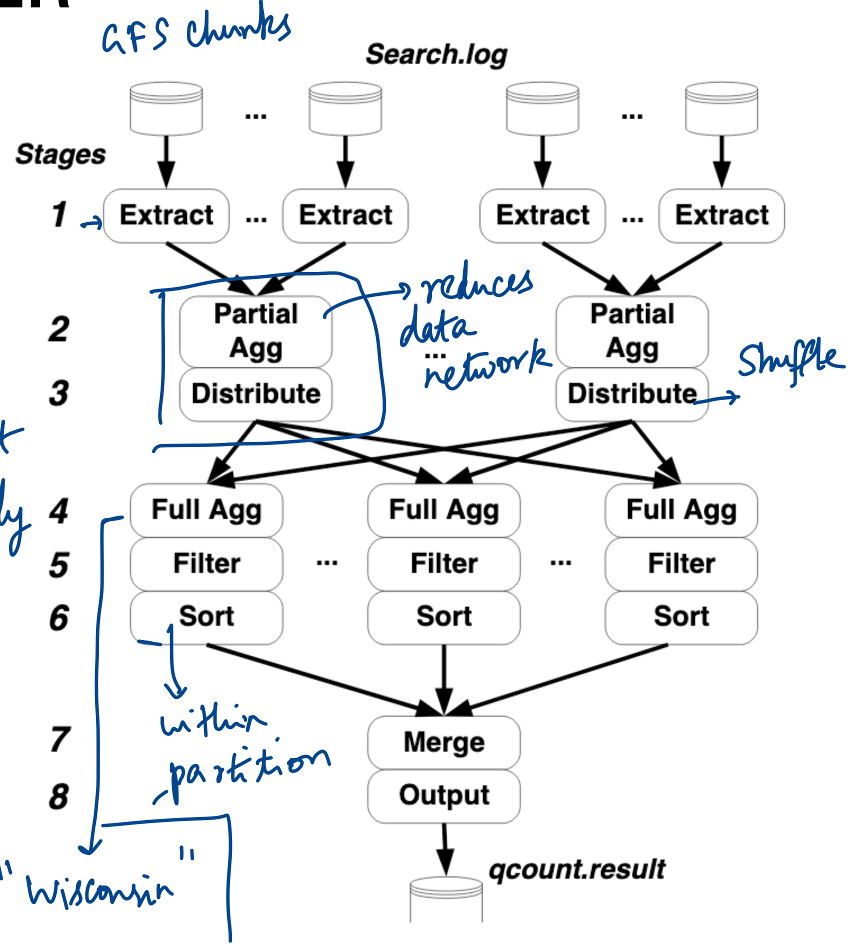
→ Area in DBMS

Rewrite the query expression → lowest cost

Examples:

- Removing unnecessary columns → don't read
- Pushing down selection predicates → filter out rows early
- Pre-aggregating

```
SELECT query, COUNT() AS count
FROM "search.log" →
USING LogExtractor
GROUP BY query
HAVING count > 1000
ORDER BY count DESC;
```



GFS chunks

→ reduces data network

→ Shuffle

within partition

"Wisconsin"

RUNTIME OPTIMIZATIONS

↳ Change the query plan at runtime

Hierarchical aggregation

↳ aggr within a machine

↳ within a rack

Locality-sensitive task placement

↳ extractors on the same machine input exists

↳ operator close to where its inputs exist

SUMMARY, TAKEAWAYS

Relational API

- Enables rich space of optimizations
- Easy to use, integration with C#

Scope Execution

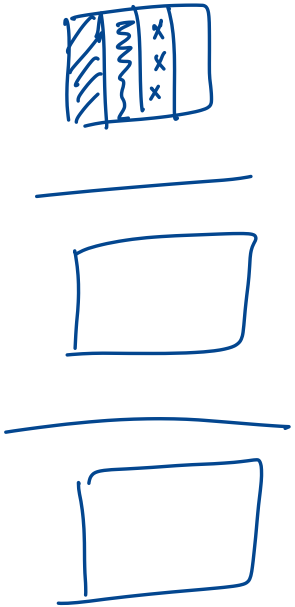
- Compiler to check for errors, generate DAG
- Optimizer to accelerate queries (static + dynamic)

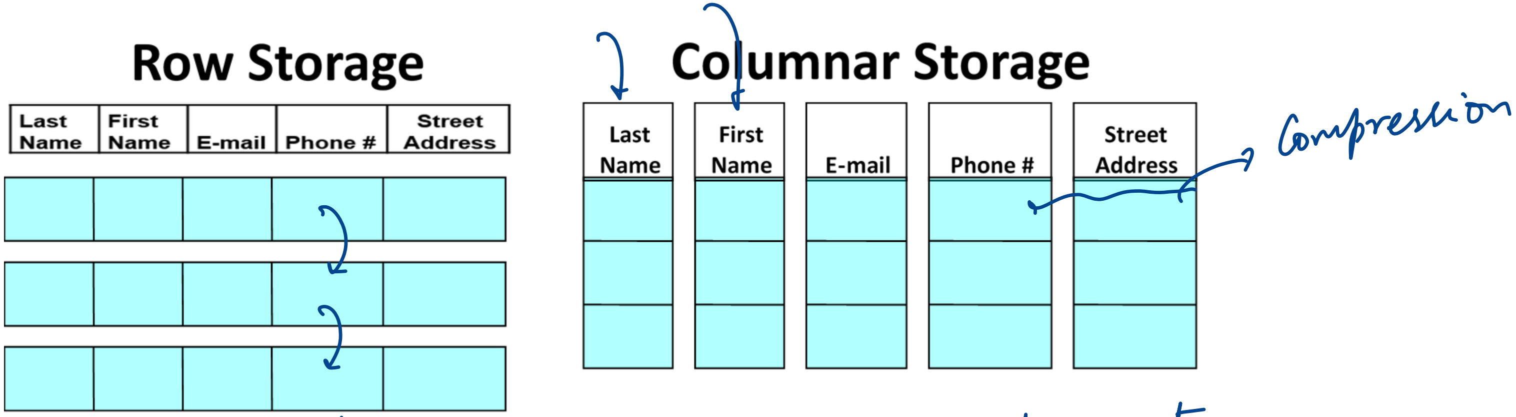
Precursor to systems like SparkSQL

DISCUSSION

https://forms.gle/D7D1b1g3VoQSJxBQ6

Consider you have a column-oriented data layout on your storage system (Example below). What are some reasons that a SCOPE query might be faster than running equivalent MR program?





→ Optimization for operators

→ data compression

→ fewer bytes read

→ Extractor - filter out columns that are not used

http://dbmsmusings.blogspot.com/2017/10/apache-arrow-vs-parquet-and-orc-do-we.html

Does SCOPE-like Optimizer help ML workloads? Consider the code in your Assignment2. What parts of your code would benefit and what parts would not?

Distributed ML

↳ Insert Scatter / Gather automatically
Reduce / Broadcast

↳ Hierarchical AllReduce
based on network topology etc.

I/O → input reading / Data loader is faster

NEXT STEPS

Next class: Elastic Data Warehousing with SnowFlake

Project proposals due soon!

Midterm: next week