Cost-based Query Optimization

+

Maryann Xue (Intel)
Julian Hyde (Hortonworks)

Hadoop Summit, San Jose
June 2016

- @maryannxue
- Apache Phoenix PMC member
- Intel

- @julianhyde
- Apache Calcite VP
- Hortonworks

What is Apache Phoenix?

- A relational database layer for Apache HBase
  - Query engine
    - Transforms SQL queries into native HBase API calls
    - Pushes as much work as possible onto the cluster for parallel execution
  - Metadata repository
    - Typed access to data stored in HBase tables
  - Transaction support
  - Table Statistics
  - A JDBC driver

APACHE PHOENIX

Advanced Features

- Secondary indexes
- Strong SQL standard compliance
- Windowed aggregates
- Connectivity (e.g. remote JDBC driver, ODBC driver)

Created architectural pain… We decided to do it right!

APACHE PHOENIX

Example 1: Optimizing Secondary Indexes

```
CREATE TABLE Emps(empId INT PRIMARY KEY, name VARCHAR(100));

CREATE INDEX I_Emps_Name ON Emps(name);
```

How we match secondary indexes in Phoenix 4.8:

Q1
```
SELECT * FROM Emp ORDER BY name
```
→ `I_Emps_Name`

Q2
```
SELECT * FROM Emp WHERE empId > 100
```
→ `Emps`

What about both?

Q3
```
SELECT * FROM Emp
WHERE empId > 100 ORDER BY name
```
→ `?`

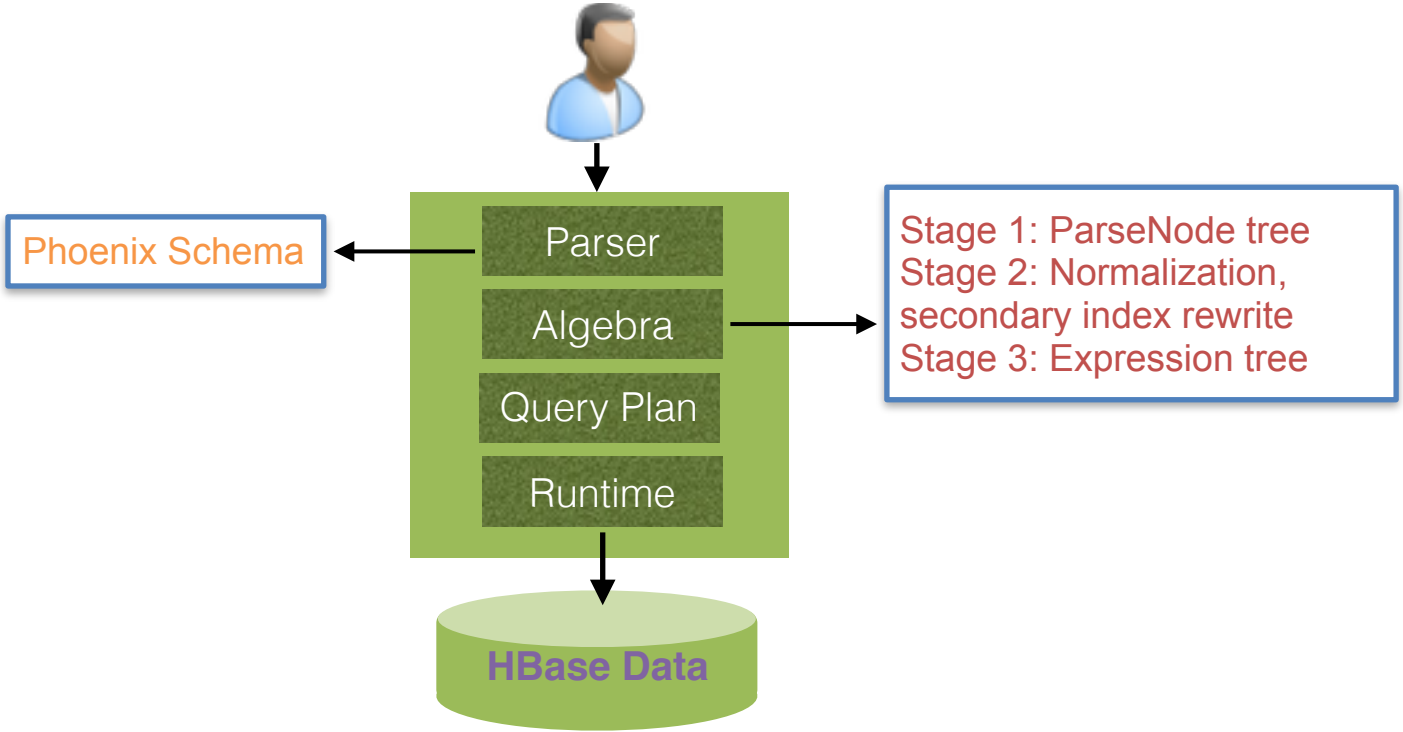
We need to make a **cost-based** decision! **Statistics** can help.

APACHE PHOENIX

Phoenix + Calcite

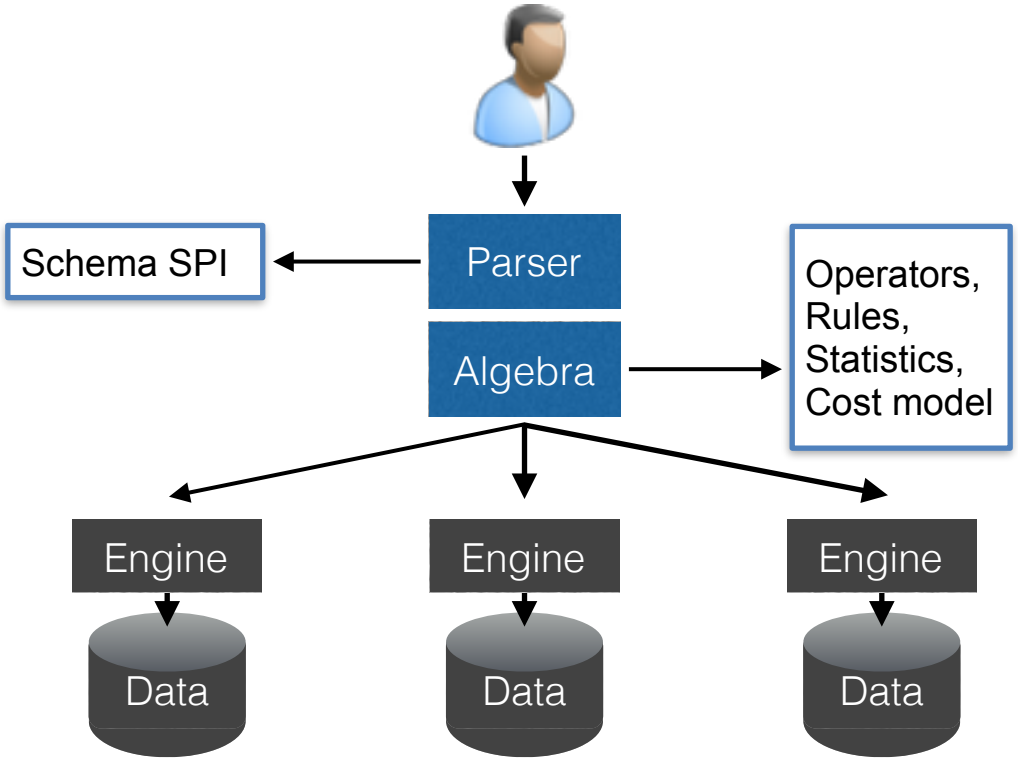
- Both are Apache projects
- Involves changes to both projects
- Work is being done on a branch of Phoenix, with changes to Calcite as needed
- Goals:
  - Remove code! (Use Calcite's SQL parser, validator)
  - Improve planning (Faster planning, faster queries)
  - Improve SQL compliance
  - Some "free" SQL features (e.g. WITH, scalar subquery, FILTER)
  - Close to full compatibility with current Phoenix SQL and APIs
- Status: beta, expected GA: late 2016

Current Phoenix Architecture



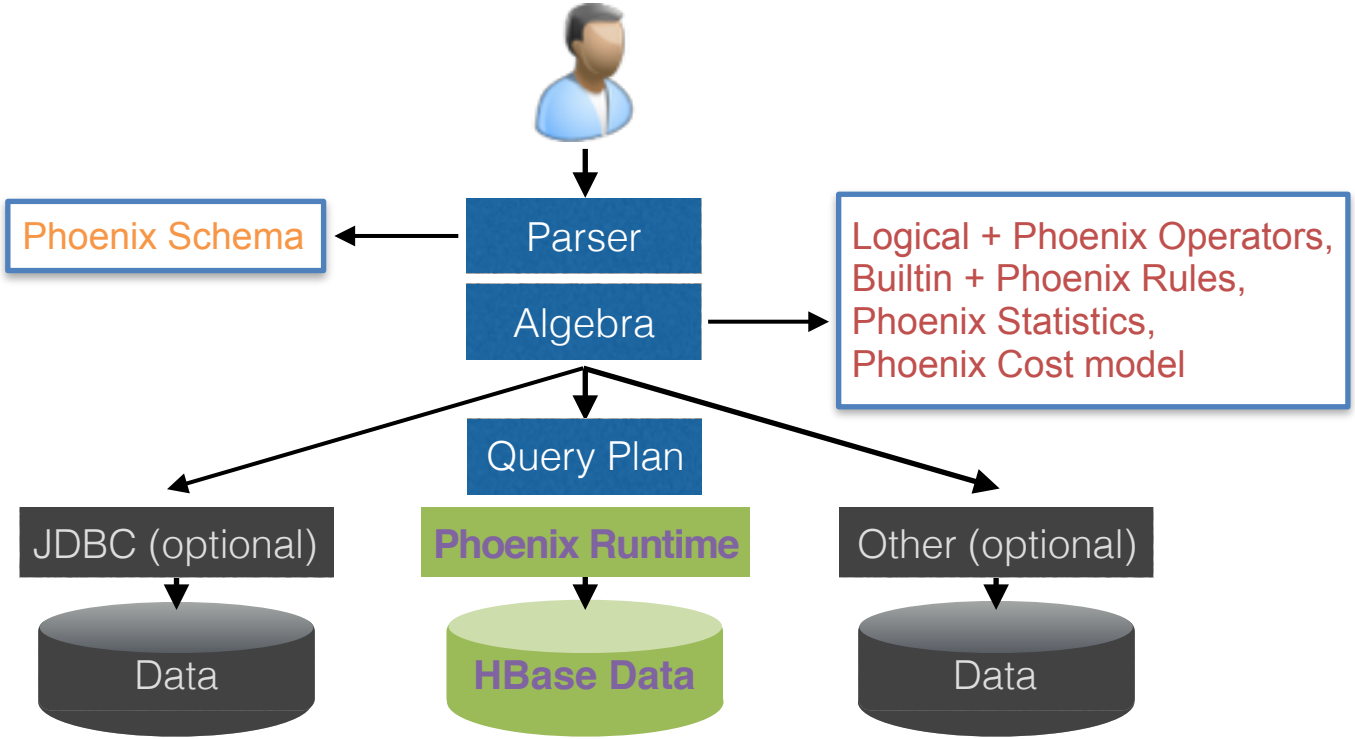
APACHE PHOENIX

Calcite Architecture



APACHE PHOENIX

Phoenix + Calcite Architecture



APACHE PHOENIX

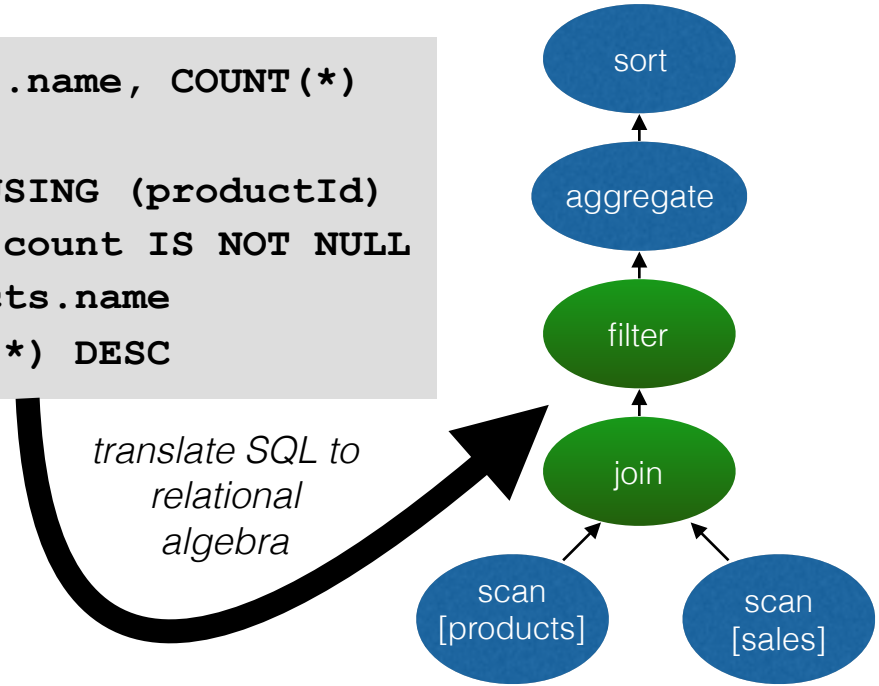
Cost-based Query Optimizer with Apache Calcite

- Base all query optimization decisions on cost
  - Filter push down; range scan vs. skip scan
  - Hash aggregate vs. stream aggregate vs. partial stream aggregate
  - Sort optimized out; sort/limit push through; fwd/rev/unordered scan
  - Hash join vs. merge join; join ordering
  - Use of data table vs. index table
  - All above (any many others) COMBINED
- Query optimizations are modeled as pluggable rules

APACHE PHOENIX

Calcite Algebra

```
SELECT products.name, COUNT(*)
FROM sales
JOIN products USING (productId)
WHERE sales.discount IS NOT NULL
GROUP BY products.name
ORDER BY COUNT(*) DESC
```

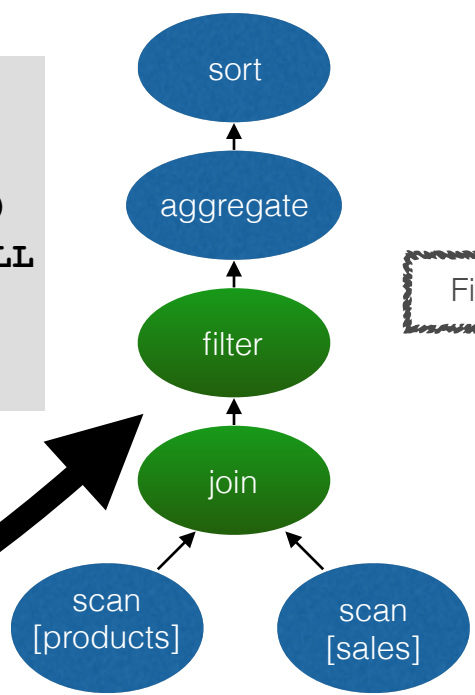


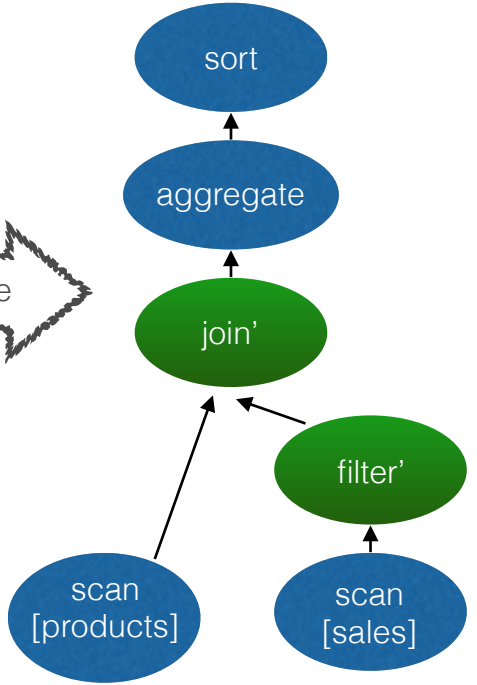
APACHE PHOENIX

Example 2: FilterIntoJoinRule

```
SELECT products.name, COUNT(*)
FROM sales
JOIN products USING (productId)
WHERE sales.discount IS NOT NULL
GROUP BY products.name
ORDER BY COUNT(*) DESC
```

translate SQL to relational algebra





APACHE PHOENIX

Example 3: Phoenix Joins

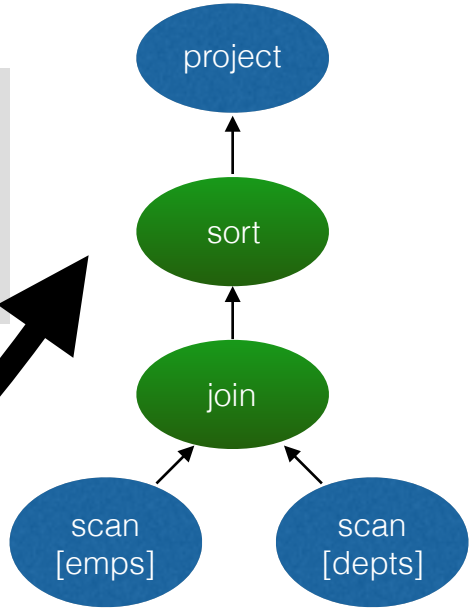
- Hash join vs. Sort merge join
  - Hash join good for: either input is small
  - Sort merge join good for: both inputs are big
  - Hash join downside: potential OOM
  - Sort merge join downside: extra sorting required sometimes
- Better to exploit the sortedness of join input
- Better to exploit the sortedness of join output

APACHE PHOENIX

Example 3: Calcite Algebra

```
SELECT empid, e.name, d.name, location
FROM emps AS e
JOIN depts AS d USING (deptno)
ORDER BY d.deptno
```

translate SQL to relational algebra

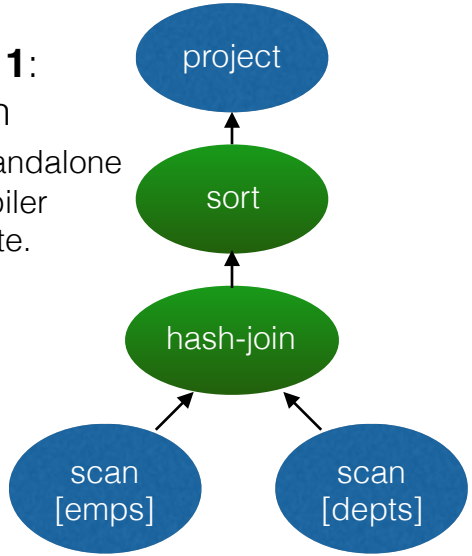


APACHE PHOENIX

Example 3: Plan Candidates

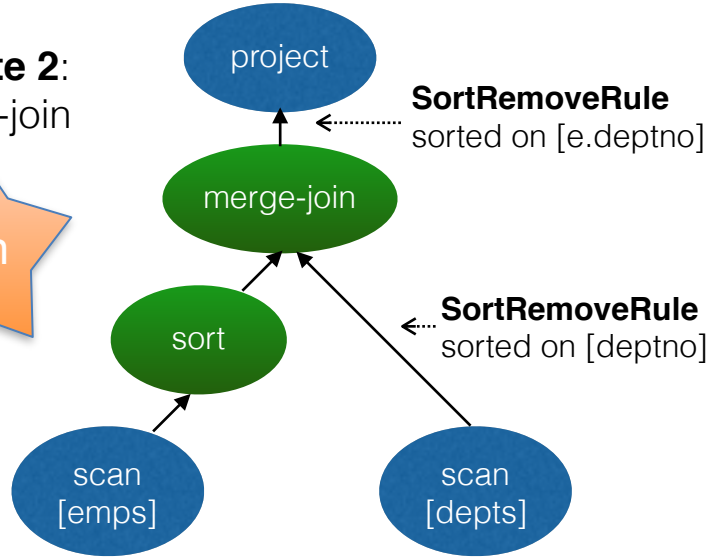
Candidate 1:
hash-join

*also what standalone Phoenix compiler would generate.



Candidate 2:
merge-join

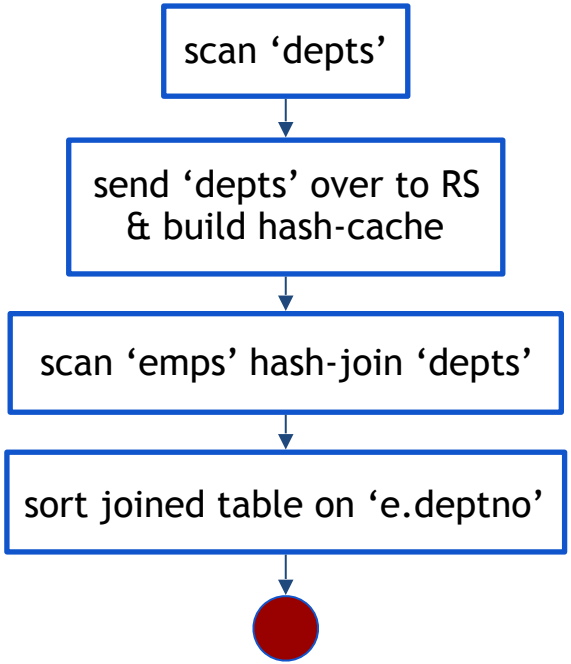
Win



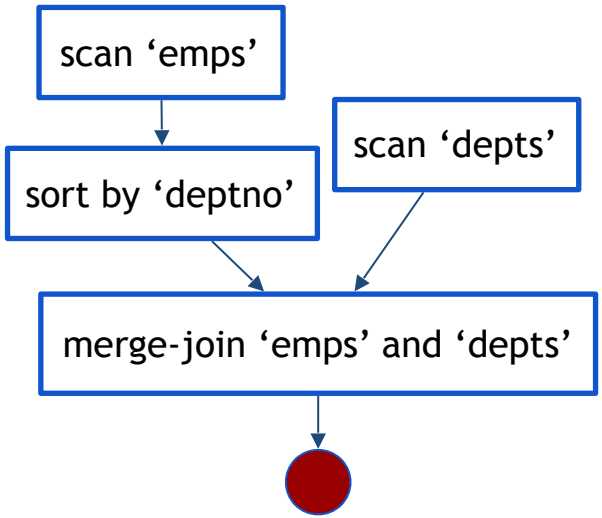
1. Very little difference in all other operators: project, scan, hash-join or merge-join
2. Candidate 1 would sort “emps join depts”, while candidate 2 would only sort “emps”

APACHE PHOENIX

Example 3: Improved Plan



Old vs. New



1. Exploited the sortedness of join input
2. Exploited the sortedness of join output

APACHE PHOENIX

(and now, a brief look at Calcite)

Apache Calcite

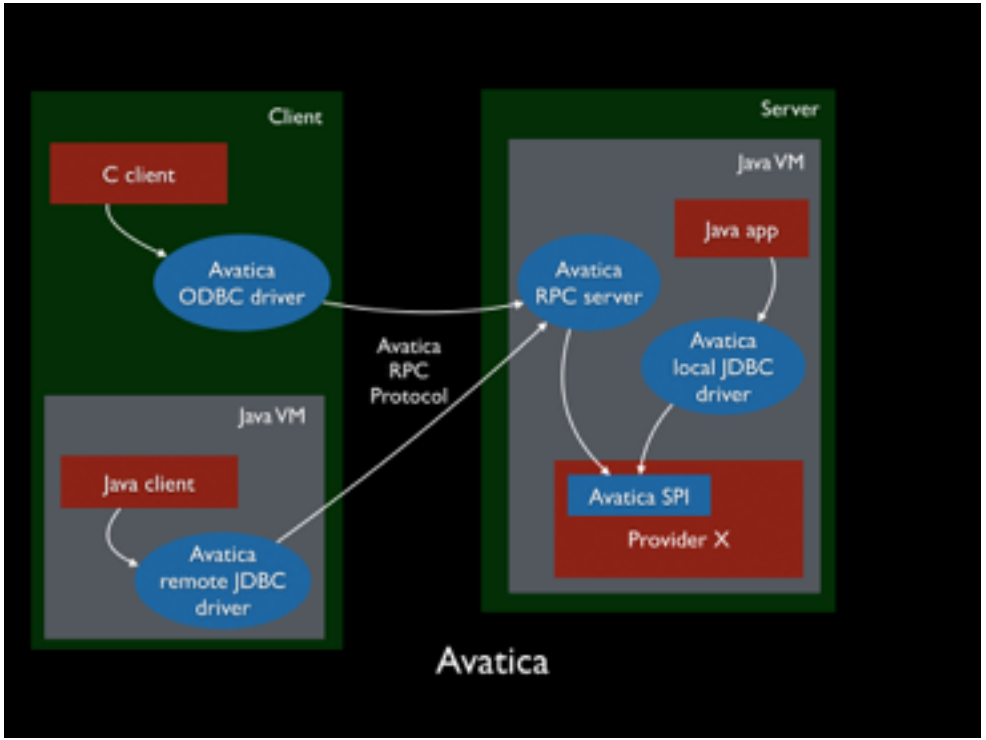
- Apache top-level project since October, 2015
- Query planning framework
  - Relational algebra, rewrite rules
  - Cost model & statistics
  - Federation via adapters
  - Extensible
- Packaging
  - Library
  - Optional SQL parser, JDBC server
  - Community-authored rules, adapters

Embedded	Adapters	Streaming
Apache Drill	Apache Cassandra*	Apache Flink*
Apache Hive	Apache Spark	Apache Samza
Apache Kylin	CSV	Apache Storm
Apache Phoenix*	In-memory	
Cascading	JDBC	
Lingual	JSON	
	MongoDB	
	Splunk	
	Web tables	

APACHE PHOENIX

Apache Calcite Avatica

- Database connectivity stack
- Self-contained sub-project of Calcite
- Fast, open, stable
- Powers Phoenix Query Server



APACHE PHOENIX

Calcite – APIs and SPIs

Relational algebra

RelNode (operator)
- TableScan
- Filter
- Project
- Union
- Aggregate
- …

RelDataType (type)
RexNode (expression)
RelTrait (physical property)
- RelConvention (calling-convention)
- RelCollation (sortedness)
- TBD (bucketedness/distribution)

SQL parser

SqlNode
SqlParser
SqlValidator

Metadata

Schema
Table
Function
- TableFunction
- TableMacro

Lattice

JDBC driver (Avatica)

Transformation rules

RelOptRule
- MergeFilterRule
- PushAggregateThroughUnionRule
- 100+ more

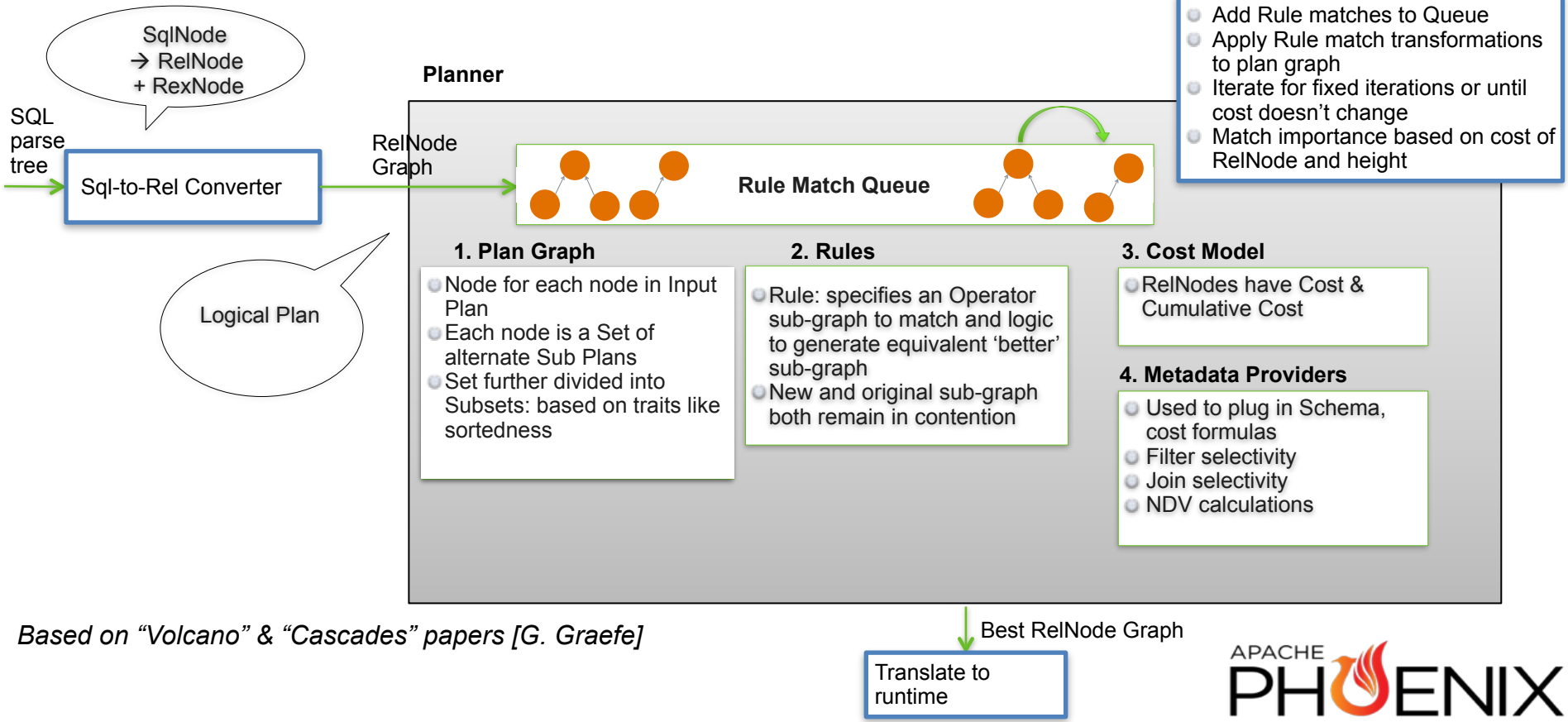
Global transformations
- Unification (materialized view)
- Column trimming
- De-correlation

Cost, statistics

RelOptCost
RelOptCostFactory
RelMetadataProvider
- RelMdColumnUniquensss
- RelMdDistinctRowCount
- RelMdSelectivity

APACHE PHOENIX

Calcite Planning Process



Based on "Volcano" & "Cascades" papers [G. Graefe]

APACHE PHOENIX

Views and materialized views

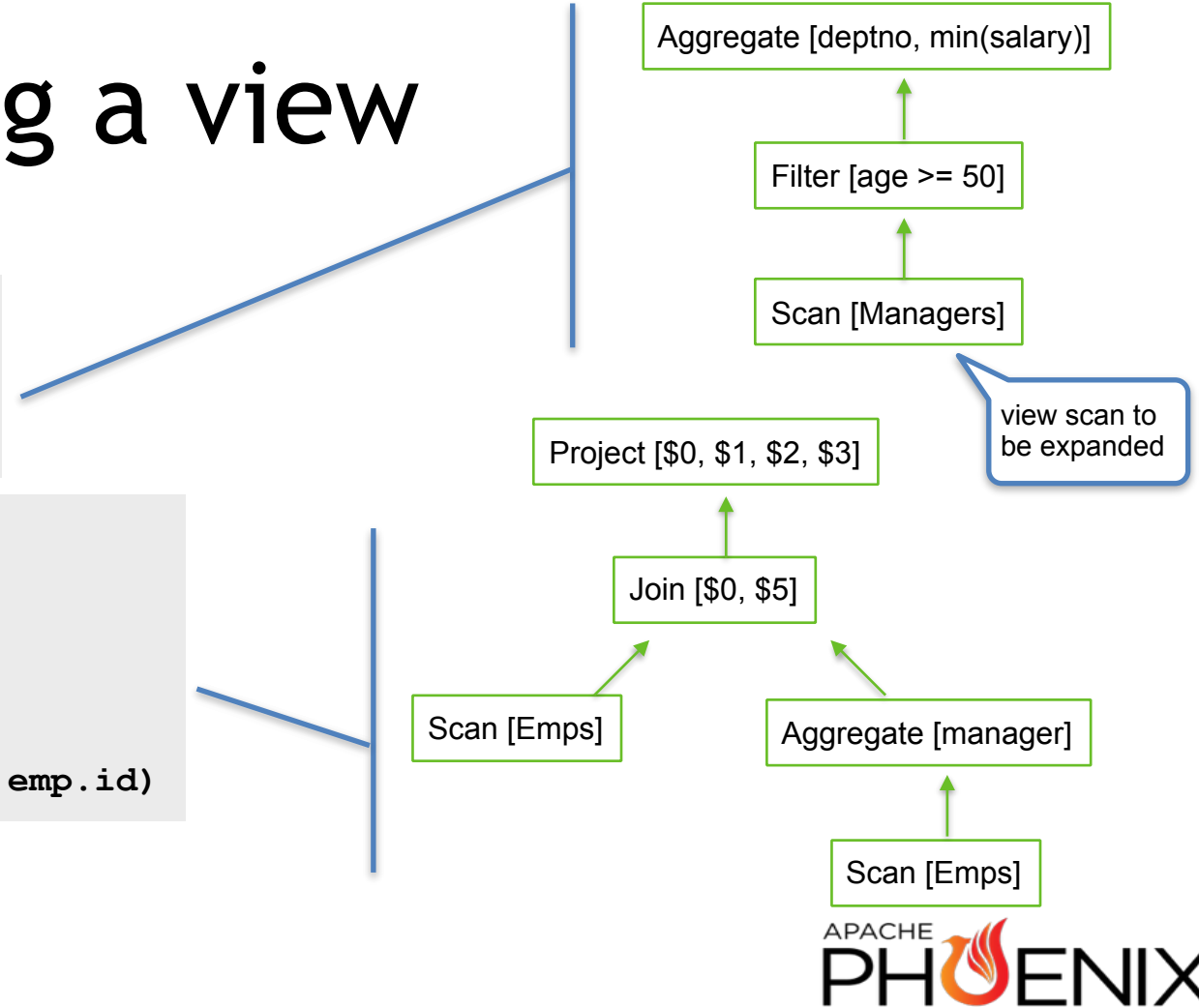
- A ***view*** is a named relational expression, stored in the catalog, that is expanded while planning a query.
- A ***materialized view*** is an equivalence, stored in the catalog, between a table and a relational expression.

  The planner substitutes the table into queries where it will help, even if the queries do not reference the materialized view.

Query using a view

```
SELECT deptno, min(salary)
FROM Managers
WHERE age >= 50
GROUP BY deptno
```

```
CREATE VIEW Managers AS
SELECT *
FROM Emps
WHERE EXISTS (
  SELECT *
  FROM Emps AS underling
  WHERE underling.manager = emp.id)
```

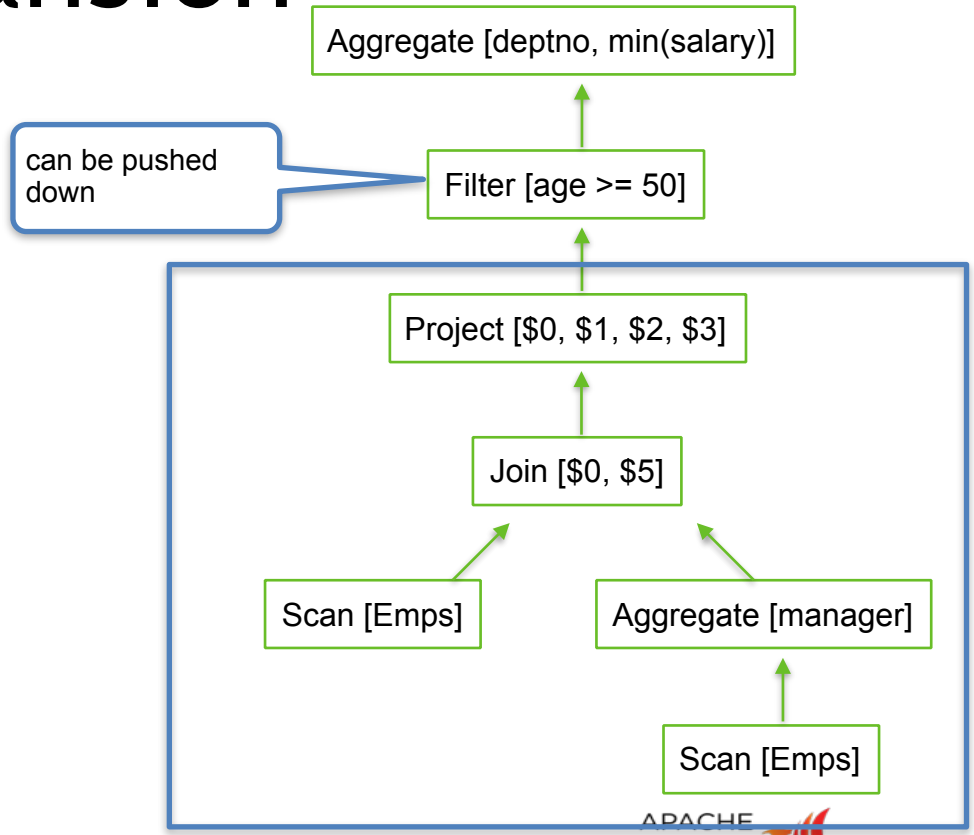


APACHE PHOENIX

After view expansion

```
SELECT deptno, min(salary)
FROM Managers
WHERE age >= 50
GROUP BY deptno
```

```
CREATE VIEW Managers AS
SELECT *
FROM Emps
WHERE EXISTS (
  SELECT *
  FROM Emps AS underling
  WHERE underling.manager = emp.id)
```



Materialized view

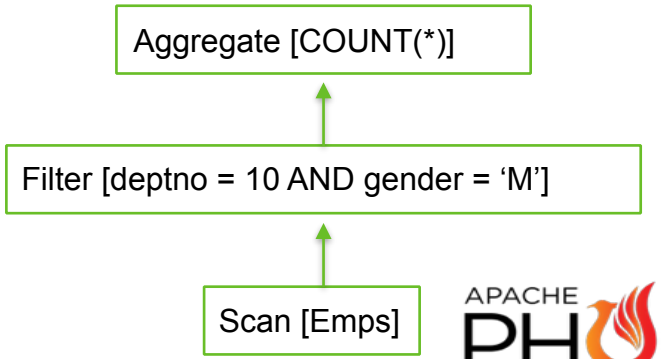
Scan [EmpSummary] = Aggregate [deptno, gender, COUNT(*), SUM(sal)]

Aggregate [deptno, gender, COUNT(*), SUM(sal)]

Scan [Emps]

```
CREATE MATERIALIZED VIEW EmpSummary AS
SELECT deptno,
  gender,
  COUNT(*) AS c,
  SUM(sal) AS s
FROM Emps
GROUP BY deptno, gender
```

```
SELECT COUNT(*)
FROM Emps
WHERE deptno = 10
AND gender = 'M'
```

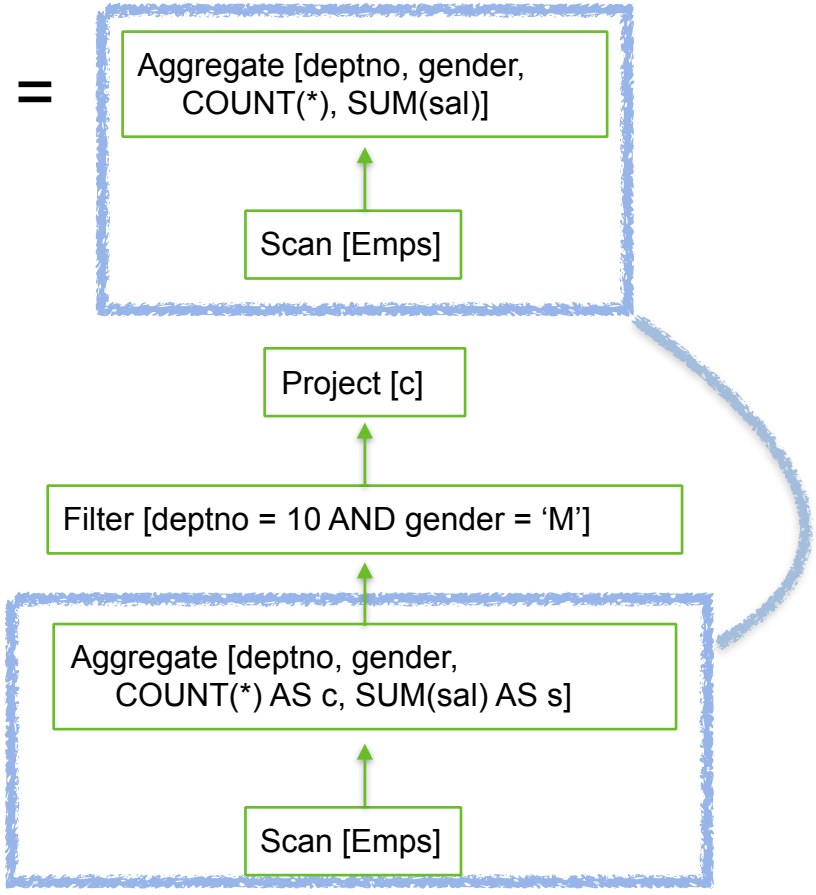


APACHE PHOENIX

Materialized view, step 2: Rewrite query to match

Scan [EmpSummary]

=



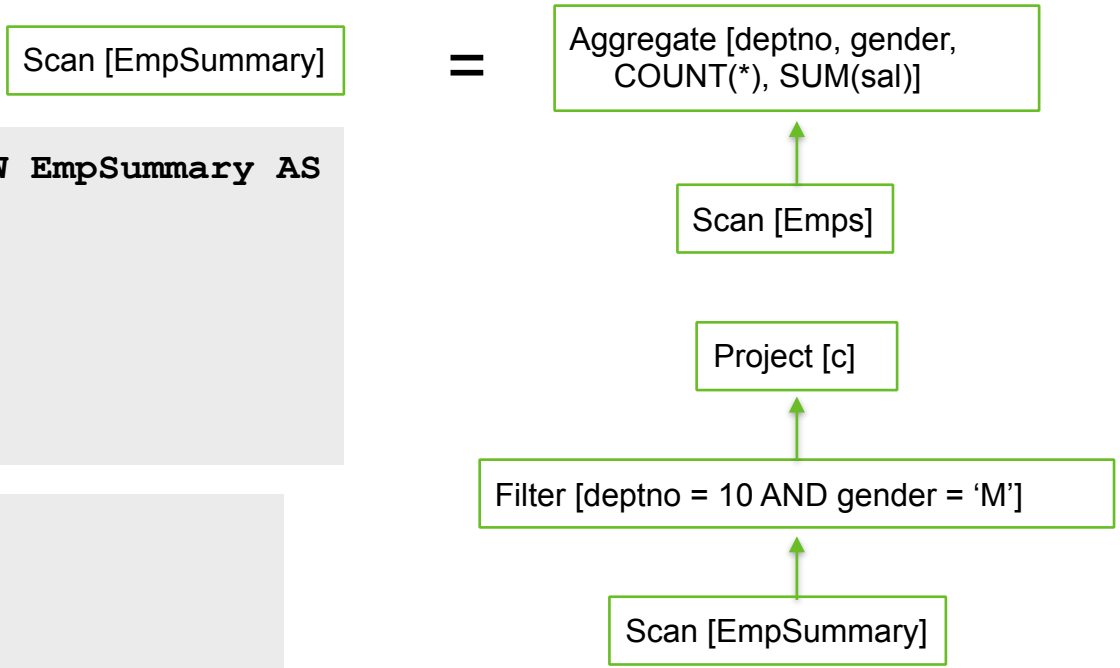

```
CREATE MATERIALIZED VIEW EmpSummary AS
SELECT deptno,
  gender,
  COUNT(*) AS c,
  SUM(sal) AS s
FROM Emps
GROUP BY deptno, gender
```

```
SELECT COUNT(*)
FROM Emps
WHERE deptno = 10
AND gender = 'M'
```

Materialized view, step 3: Substitute table

```
CREATE MATERIALIZED VIEW EmpSummary AS
SELECT deptno,
  gender,
  COUNT(*) AS c,
  SUM(sal) AS s
FROM Emps
GROUP BY deptno, gender
```

```
SELECT COUNT(*)
FROM Emps
WHERE deptno = 10
AND gender = 'M'
```



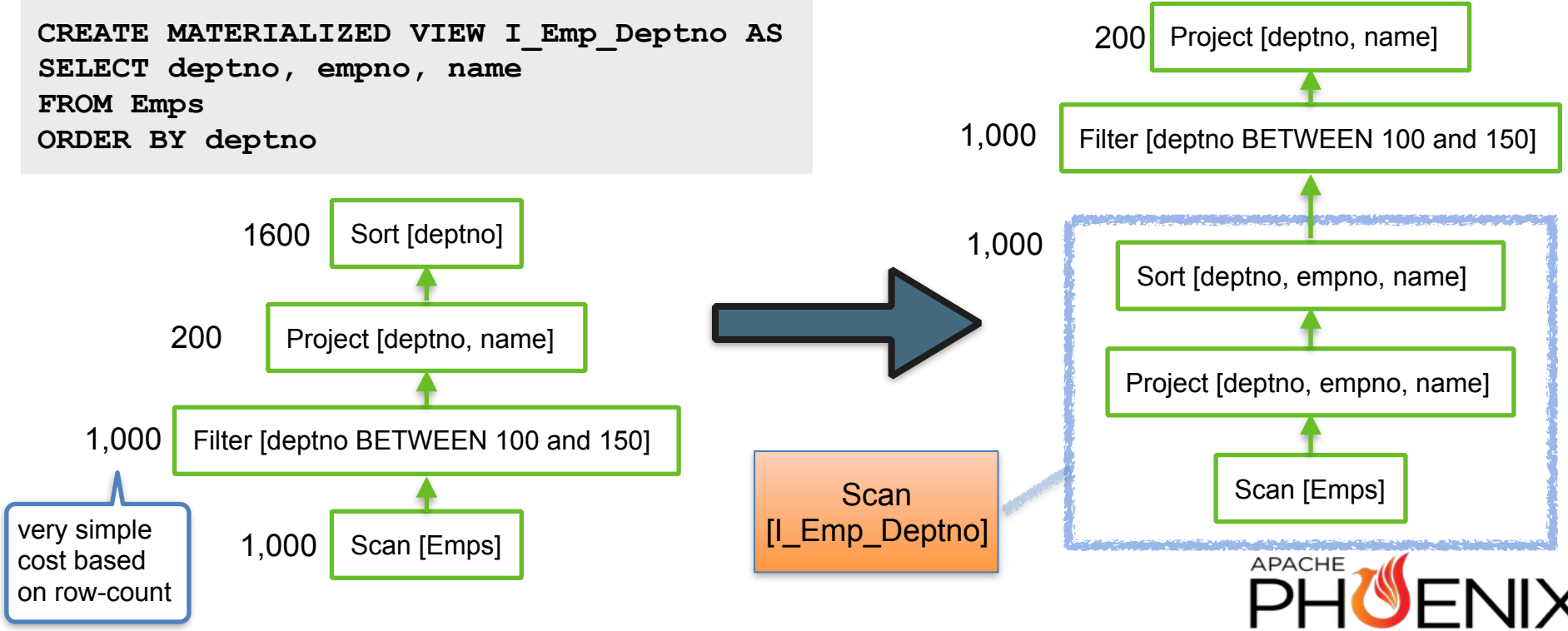
APACHE PHOENIX

(and now, back to Phoenix)

Example 1, Revisited: Secondary Index

Optimizer internally creates a mapping (query, table) equivalent to:

```
CREATE MATERIALIZED VIEW I_Emp_Deptno AS
SELECT deptno, empno, name
FROM Emps
ORDER BY deptno
```



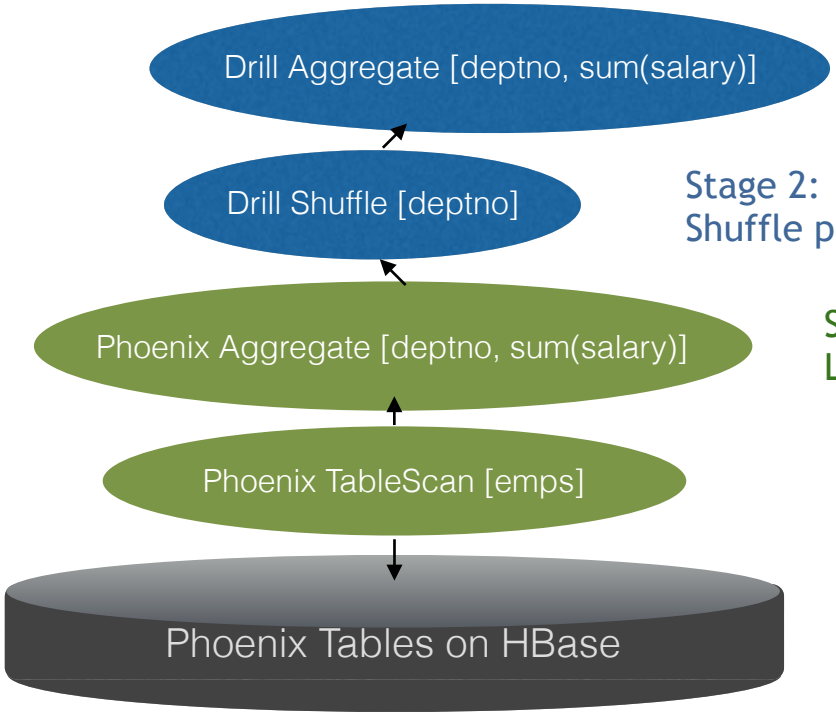
Beyond Phoenix 4.8 with Apache Calcite

- Get the missing SQL support
  - WITH, UNNEST, Scalar subquery, etc.
- Materialized views
  - To allow other forms of indices (maybe defined as external), e.g., a filter view, a join view, or an aggregate view.
- Interop with other Calcite adapters
  - Already used by Drill, Hive, Kylin, Samza, etc.
  - Supports any JDBC source
  - Initial version of Drill-Phoenix integration already working

APACHE PHOENIX

Drillix: Interoperability with Apache Drill

```
SELECT deptno, sum(salary) FROM emps GROUP BY deptno
```



Stage 3:
Final aggregation

Stage 2:
Shuffle partial results

Stage 1:
Local Partial aggregation

APACHE PHOENIX

Thank you! Questions?

@maryannxue
@julianhyde

http://phoenix.apache.org
http://calcite.apache.org