SQL on everything, in memory

Apache Calcite

Strata + Hadoop World Conference

Julian Hyde

Strata, NYC

October 16th, 2014

Hortonworks

About me

Julian Hyde

Architect at Hortonworks

Open source:

- Founder & lead, Apache Calcite (query optimization framework)
- Founder & lead, Pentaho Mondrian (analysis engine)
- Committer, Apache Drill
- Contributor, Apache Hive
- Contributor, Cascading Lingual (SQL interface to Cascading)

Past:

- SQLstream (streaming SQL)
- Broadbase (data warehouse)
- Oracle (SQL kernel development)

SQL: in and out of fashion

1969 — CODASYL (network database)

1979 — First commercial SQL RDBMSs

1990 — Acceptance — transaction processing on SQL

1993 — Multi-dimensional databases

1996 — SQL EDWs

2006 — Hadoop and other “big data” technologies

2008 — NoSQL

2011 — SQL on Hadoop

2014 — Interactive analytics on {Hadoop, NoSQL, DBMS}, using SQL

SQL remains popular.

But why?

“SQL inside”

Implementing SQL well is hard

- System cannot just “run the query” as written
- Require relational algebra, query planner (optimizer) & metadata

…but it’s worth the effort

Algebra-based systems are more flexible

- Add new algorithms (e.g. a better join)
- Re-organize data
- Choose access path based on statistics
- Dumb queries (e.g. machine-generated)
- Relational, schema-less, late-schema, non-relational (e.g. key-value, document)

Apache Calcite

Apache Calcite

Apache incubator project since May, 2014

- Originally named Optiq

Query planning framework

- Relational algebra, rewrite rules, cost model
- Extensible

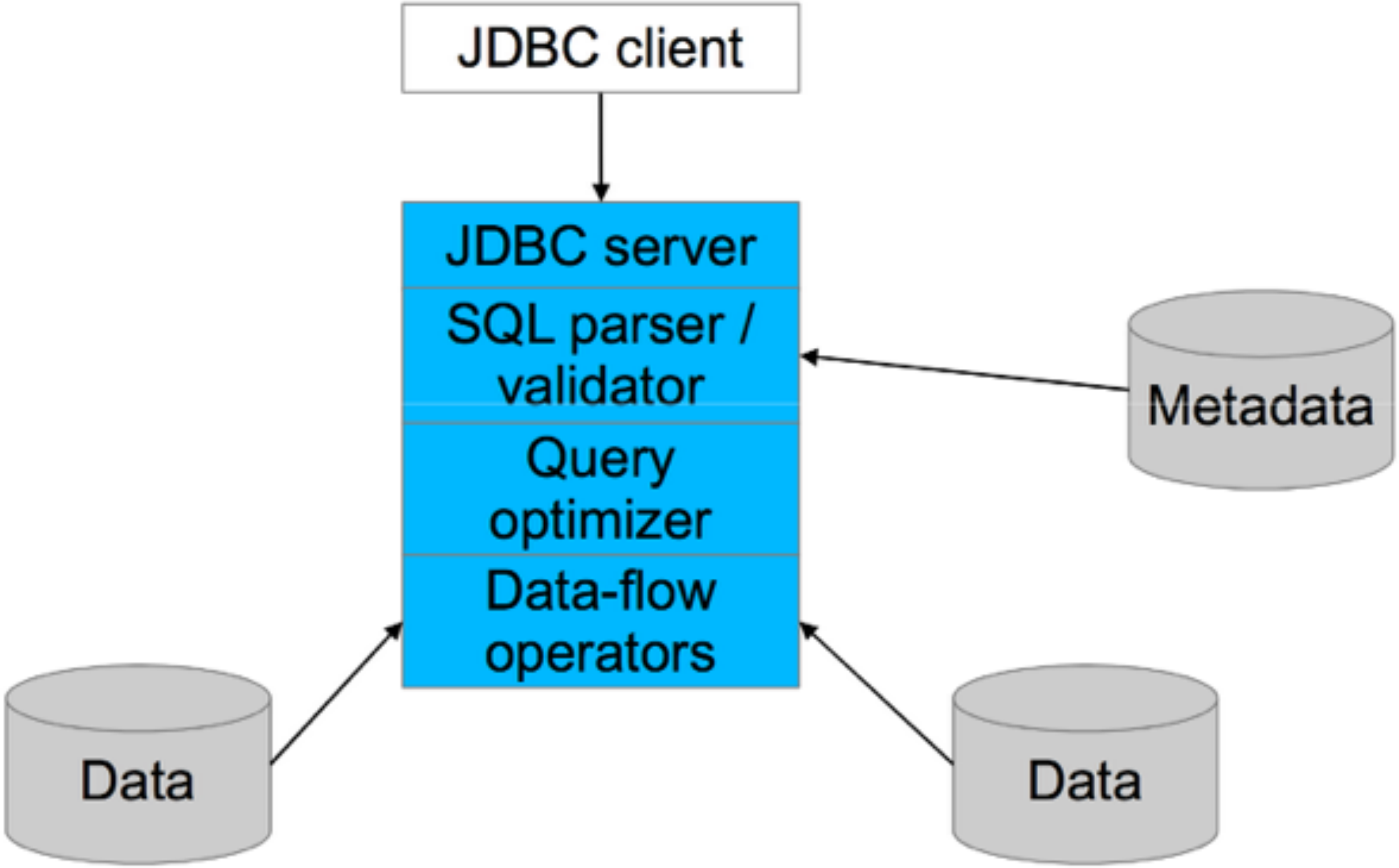
Packaging

- Library (JDBC server optional)
- Open source
- Community-authored rules, adapters

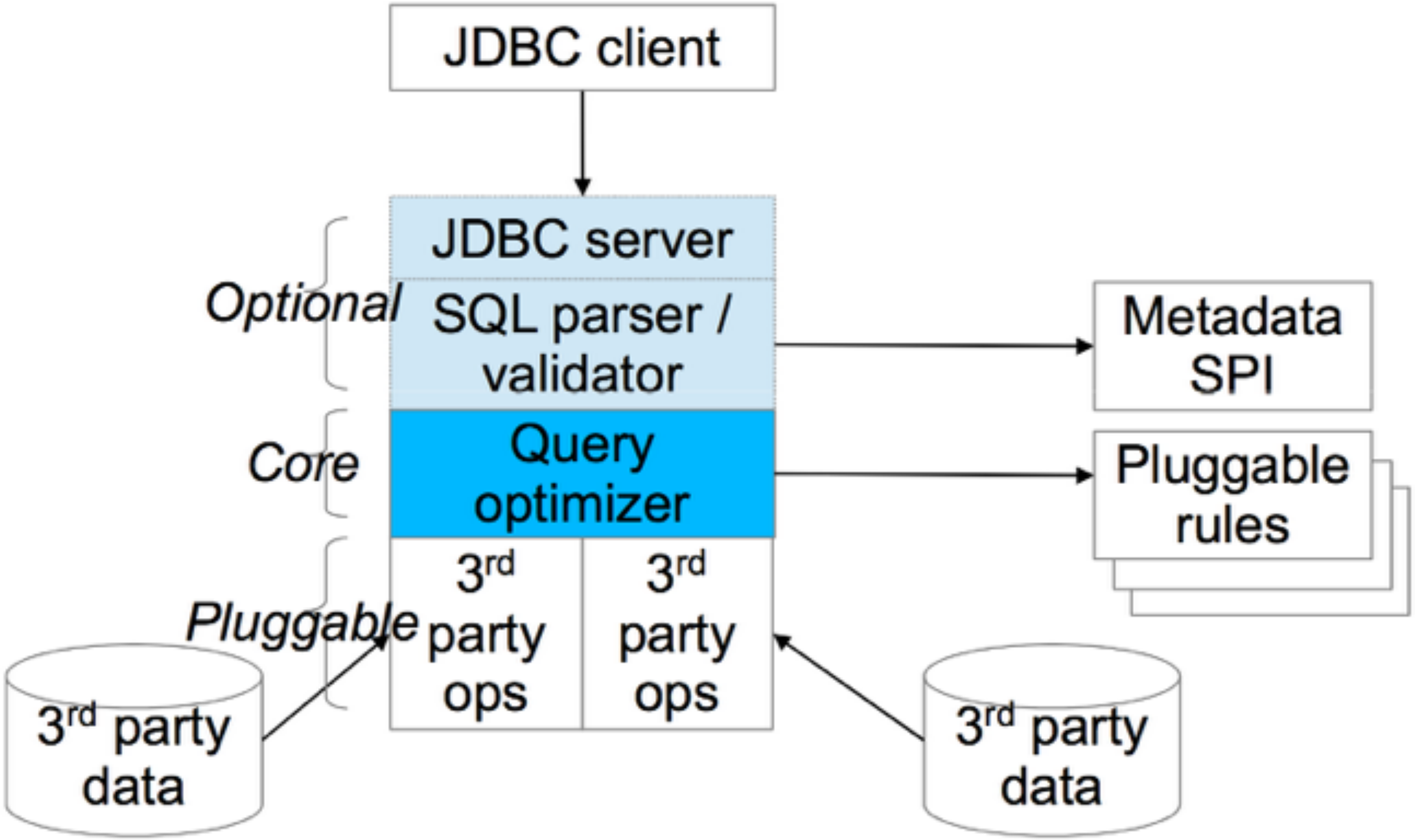
Adoption

- **Embedded**: Lingual (SQL interface to Cascading), Apache Drill, Apache Hive, Kylin OLAP
- **Adapters**: Splunk, Spark, MongoDB, JDBC, CSV, JSON, Web tables, In-memory data

Conventional DB architecture



Calcite architecture

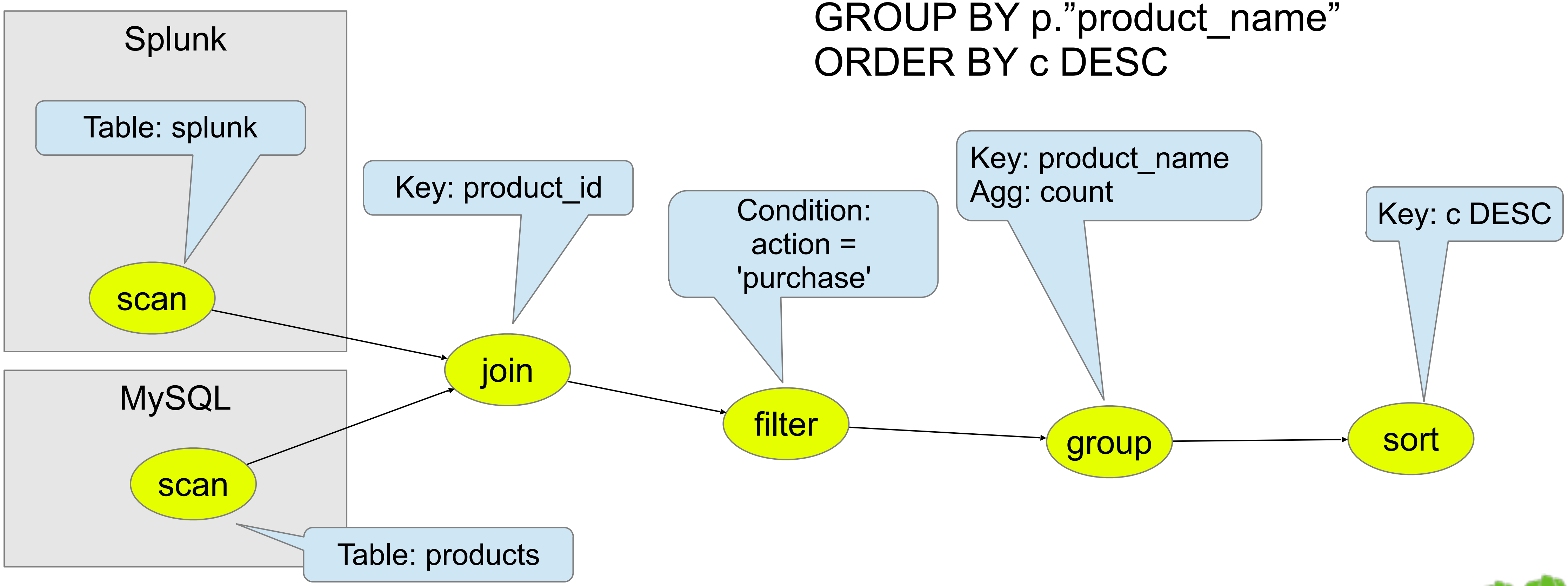


Demo

{sqlline, apache-calcite-0.9.1, .csv}

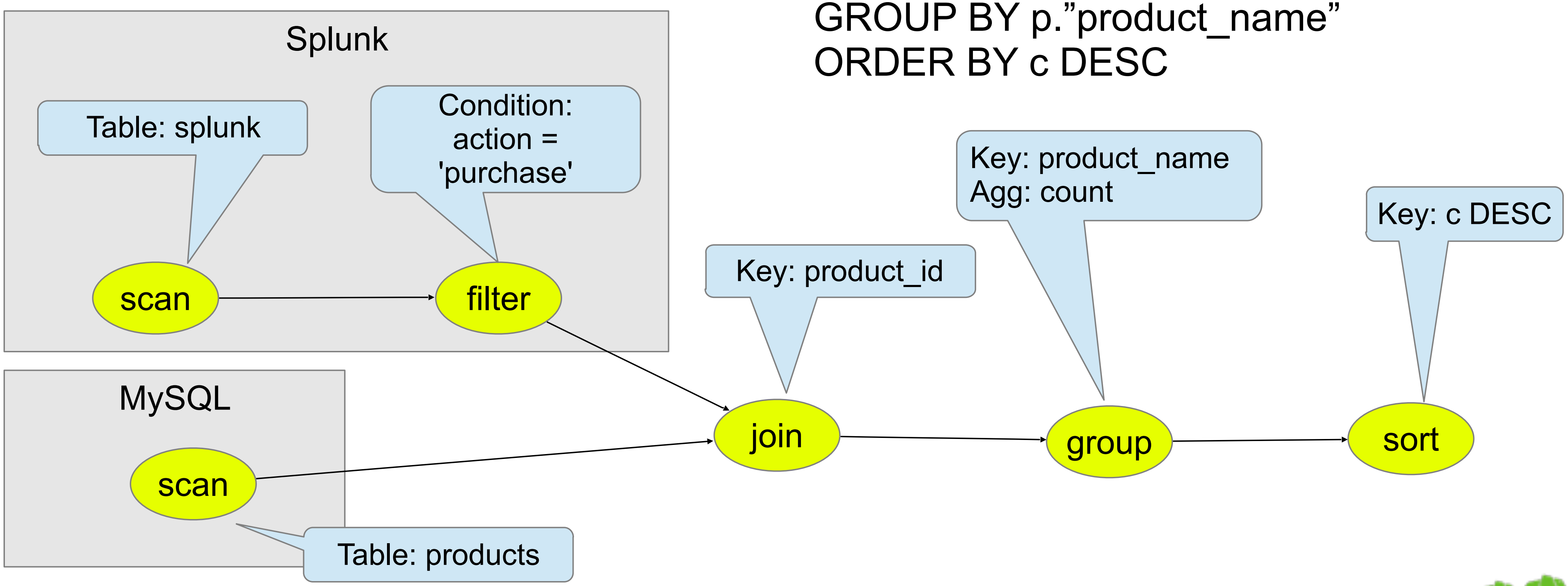
Expression tree

SELECT p.“product_name”, COUNT(*) AS c
FROM “splunk”.”splunk” AS s
JOIN “mysql”.”products” AS p
ON s.”product_id” = p.”product_id”
WHERE s.“action” = 'purchase'
GROUP BY p.”product_name”
ORDER BY c DESC



Expression tree (optimized)

SELECT p.“product_name”, COUNT(*) AS c
FROM “splunk”.”splunk” AS s
    JOIN “mysql”.”products” AS p
    ON s.”product_id” = p.”product_id”
WHERE s.“action” = 'purchase'
GROUP BY p.”product_name”
ORDER BY c DESC



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

Transformation rules

RelOptRule

- MergeFilterRule
- PushAggregateThroughUnionRule
- 100+ more

Global transformations

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

Cost, statistics

RelOptCost
RelOptCostFactory
RelMetadataProvider

- RelMdColumnUniquenss
- RelMdDistinctRowCount
- RelMdSelectivity

Calcite Planning Process

SqlNode → RelNode + RexNode

SQL parse tree → Sql-to-Rel Converter → RelNode Graph

Logical Plan

Planner

- Add Rule matches to Queue
- Apply Rule match transformations to plan graph
- Iterate for fixed iterations or until cost doesn't change
- Match importance based on cost of RelNode and height

Rule Match Queue

1. Plan Graph

- Node for each node in Input Plan
- Each node is a Set of alternate Sub Plans
- Set further divided into Subsets: based on traits like sortedness

2. Rules

- Rule: specifies an Operator sub-graph to match and logic to generate equivalent 'better' sub-graph
- New and original sub-graph both remain in contention

3. Cost Model

- RelNodes have Cost & Cumulative Cost

4. Metadata Providers

- Used to plug in Schema, cost formulas
- Filter selectivity
- Join selectivity
- NDV calculations

Best RelNode Graph → Translate to runtime

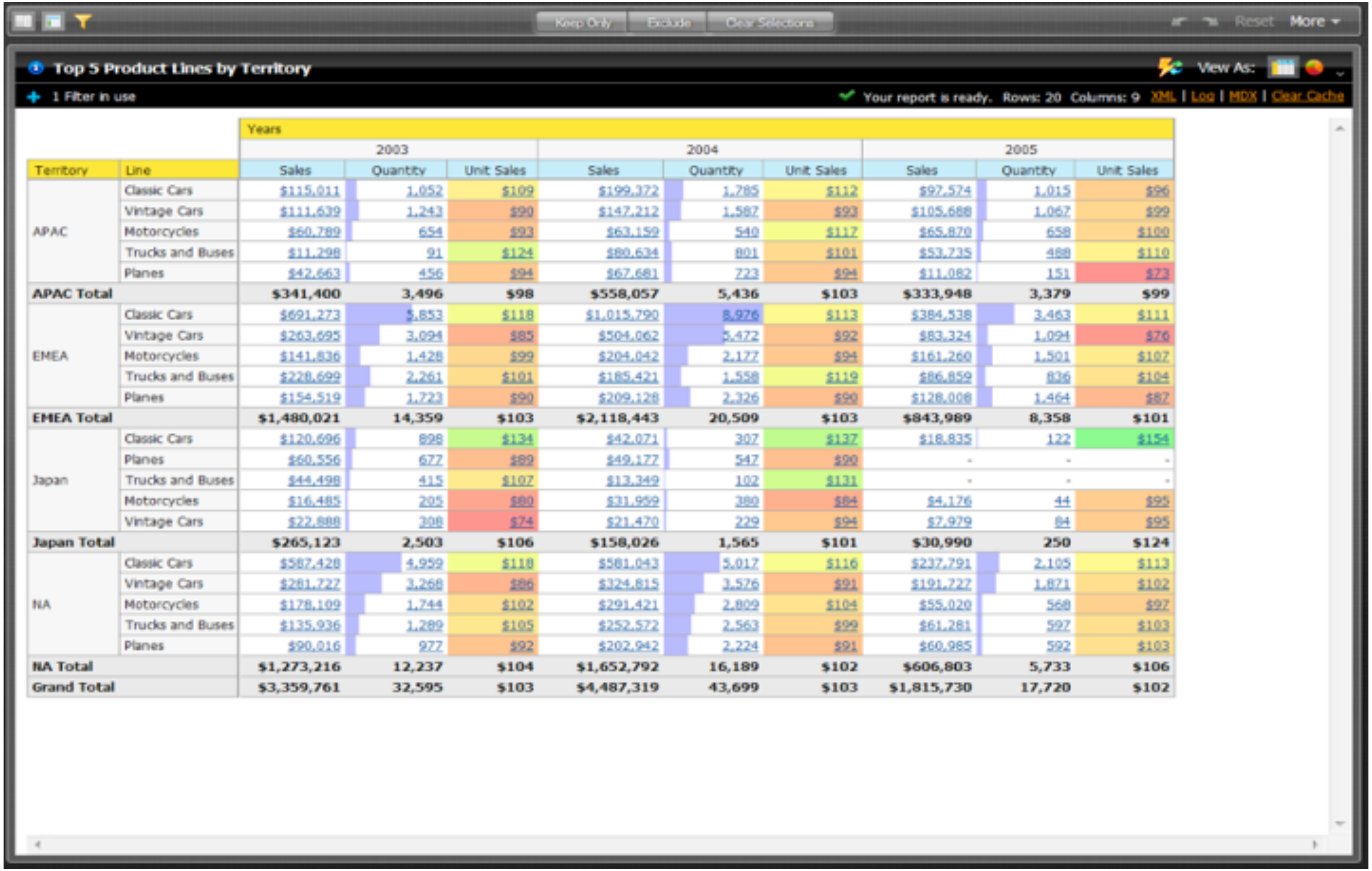
Based on "Volcano" & "Cascades" papers [G. Graefe]

Demo

{sqlline, apache-calcite-0.9.1, .csv, CsvPushProjectOntoTableRule}

Analytics

Mondrian OLAP (Saiku user interface)



Years										
		2003			2004			2005		
Territory	Line	Sales	Quantity	Unit Sales	Sales	Quantity	Unit Sales	Sales	Quantity	Unit Sales
APAC	Classic Cars	$115,011	1,052	$109	$199,372	1,785	$112	$97,574	1,015	$96
	Vintage Cars	$111,639	1,243	$90	$147,212	1,587	$93	$105,688	1,067	$99
	Motorcycles	$60,789	654	$93	$63,159	540	$117	$65,870	658	$100
	Trucks and Buses	$11,298	91	$124	$80,634	801	$101	$53,735	488	$110
	Planes	$42,663	456	$94	$67,681	723	$94	$11,082	151	$73
APAC Total		**$341,400**	**3,496**	**$98**	**$558,057**	**5,436**	**$103**	**$333,948**	**3,379**	**$99**
EMEA	Classic Cars	$691,273	5,853	$118	$1,015,790	8,976	$113	$384,538	3,463	$111
	Vintage Cars	$263,695	3,094	$85	$504,062	5,472	$92	$83,324	1,094	$76
	Motorcycles	$141,836	1,428	$99	$204,042	2,177	$94	$161,260	1,501	$107
	Trucks and Buses	$228,699	2,261	$101	$185,421	1,558	$119	$86,859	836	$104
	Planes	$154,519	1,723	$90	$209,128	2,326	$90	$128,008	1,464	$87
EMEA Total		**$1,480,021**	**14,359**	**$103**	**$2,118,443**	**20,509**	**$103**	**$843,989**	**8,358**	**$101**
Japan	Classic Cars	$120,696	898	$134	$42,071	307	$137	$18,835	122	$154
	Planes	$60,556	677	$89	$49,177	547	$90	-	-	-
	Trucks and Buses	$44,498	415	$107	$13,349	102	$131	-	-	-
	Motorcycles	$16,485	205	$80	$31,959	380	$84	$4,176	44	$95
	Vintage Cars	$22,888	308	$74	$21,470	229	$94	$7,979	84	$95
Japan Total		**$265,123**	**2,503**	**$106**	**$158,026**	**1,565**	**$101**	**$30,990**	**250**	**$124**
NA	Classic Cars	$587,428	4,959	$118	$581,043	5,017	$116	$237,791	2,105	$113
	Vintage Cars	$281,727	3,268	$86	$324,815	3,576	$91	$191,727	1,871	$102
	Motorcycles	$178,109	1,744	$102	$291,421	2,809	$104	$55,020	568	$97
	Trucks and Buses	$135,936	1,289	$105	$252,572	2,563	$99	$61,281	597	$103
	Planes	$90,016	977	$92	$202,942	2,224	$91	$60,985	592	$103
NA Total		**$1,273,216**	**12,237**	**$104**	**$1,652,792**	**16,189**	**$102**	**$606,803**	**5,733**	**$106**
Grand Total		**$3,359,761**	**32,595**	**$103**	**$4,487,319**	**43,699**	**$103**	**$1,815,730**	**17,720**	**$102**

Interactive queries on NoSQL

Typical requirements

NoSQL operational database (e.g. HBase, MongoDB, Cassandra)

Analytic queries aggregate over full scan

Speed-of-thought response (< 5 seconds first query, < 1 second second query)

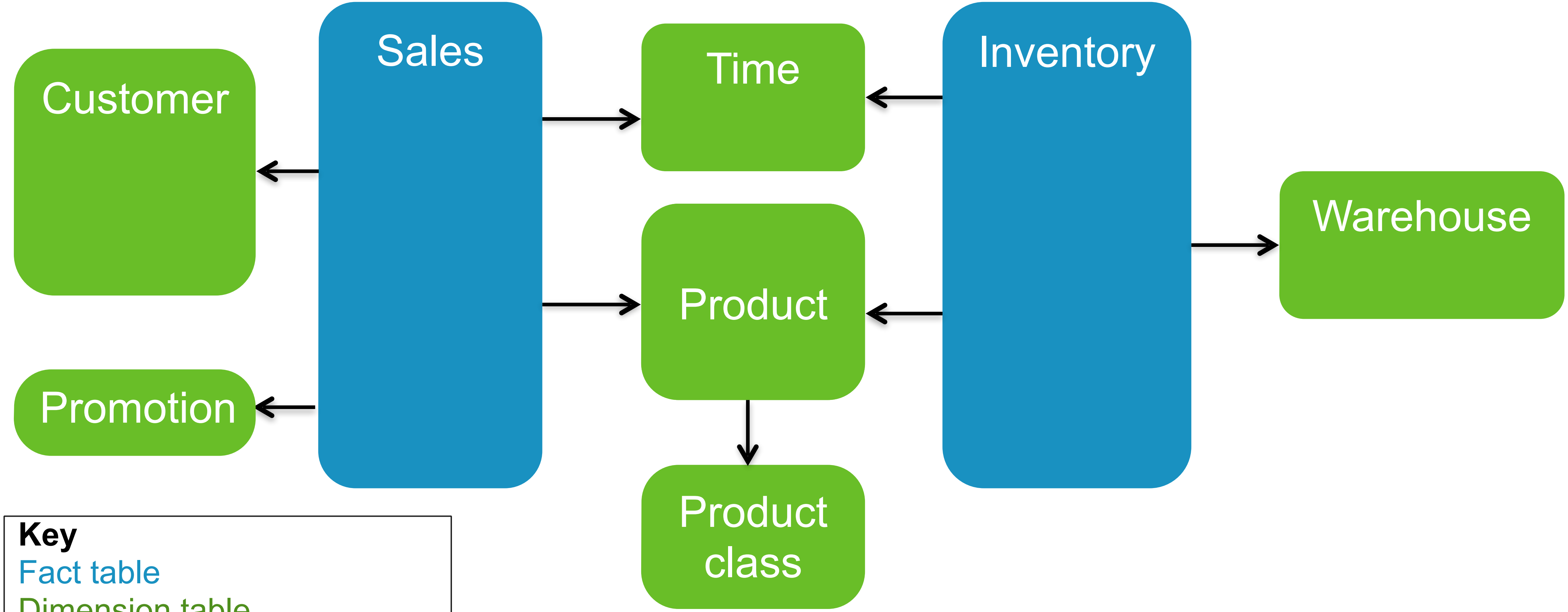
Data freshness (< 10 minutes)

Other requirements

Hybrid system (e.g. Hive + HBase)

Star schema

Star schema



Key
Fact table
Dimension table
→ Many-to-one relationship

Simple analytics problem?

System

100M US census records

1KB each record, 100GB total

4 SATA 3 disks, total read throughput 1.2GB/s

Requirement

Count all records in < 5s

Solution #1

It's not possible! It takes 80s just to read the data

Solution #2

Cheat!

How to cheat

Multiple tricks

Compress data

Column-oriented storage

Store data in sorted order

Put data in memory

Cache previous results

Pre-compute (materialize) aggregates

Common factors

Make a copy of the data

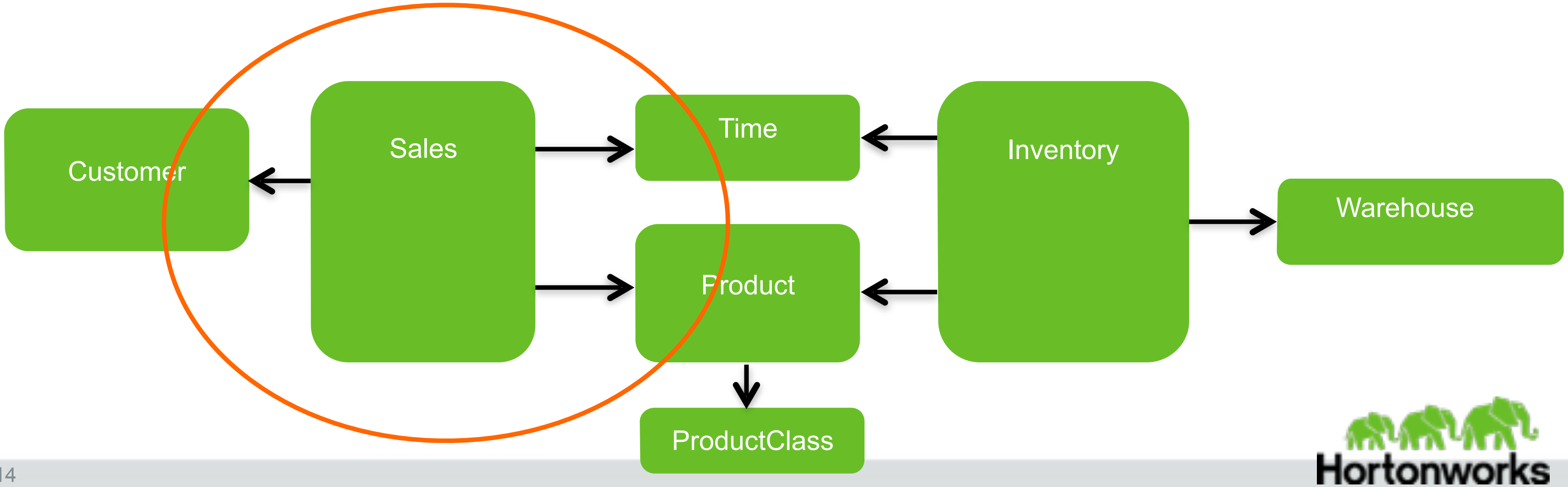
Organize it in a different way

Optimizer chooses the most suitable data organization

SQL query is unchanged

Filter-join-aggregate query

SELECT product.id, sum(sales.units), sum(sales.price), count(*)
FROM sales …
JOIN customer ON …
JOIN time ON …
JOIN product ON …
JOIN product_class ON …
WHERE time.year = 2014
AND time.quarter = ‘Q1’
AND product.color = ‘Red’
GROUP BY …



Materialized view, lattice, tile

(FAKE SYNTAX)

Materialized view

A table whose contents are guaranteed to be the same as executing a given query.

Lattice

Recommends, builds, and recognizes summary materialized views (tiles) based on a star schema.

A query defines the tables and many:1 relationships in the star schema.

Tile

A summary materialized view that belongs to a lattice.

A tile may or may not be materialized.

Materialization methods:

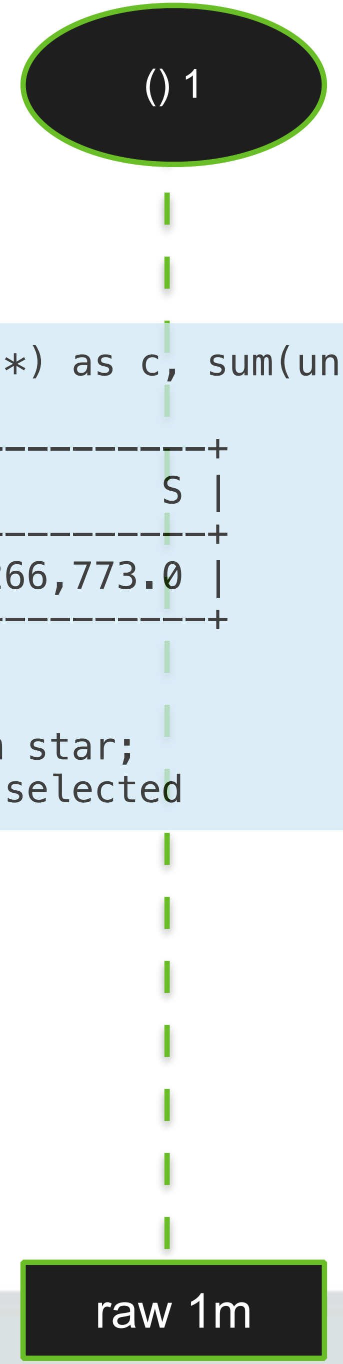
- Declare in lattice
- Generate via recommender algorithm
- Created in response to query

```
CREATE MATERIALIZED VIEW t AS
SELECT * FROM emps
WHERE deptno = 10;

CREATE LATTICE star AS
SELECT *
FROM sales_fact_1997 AS s
JOIN product AS p ON …
JOIN product_class AS pc ON …
JOIN customer AS c ON …
JOIN time_by_day AS t ON …;

CREATE MATERIALIZED VIEW zg IN star
SELECT gender, zipcode,
  COUNT(*), SUM(unit_sales)
FROM star
GROUP BY gender, zipcode;
```

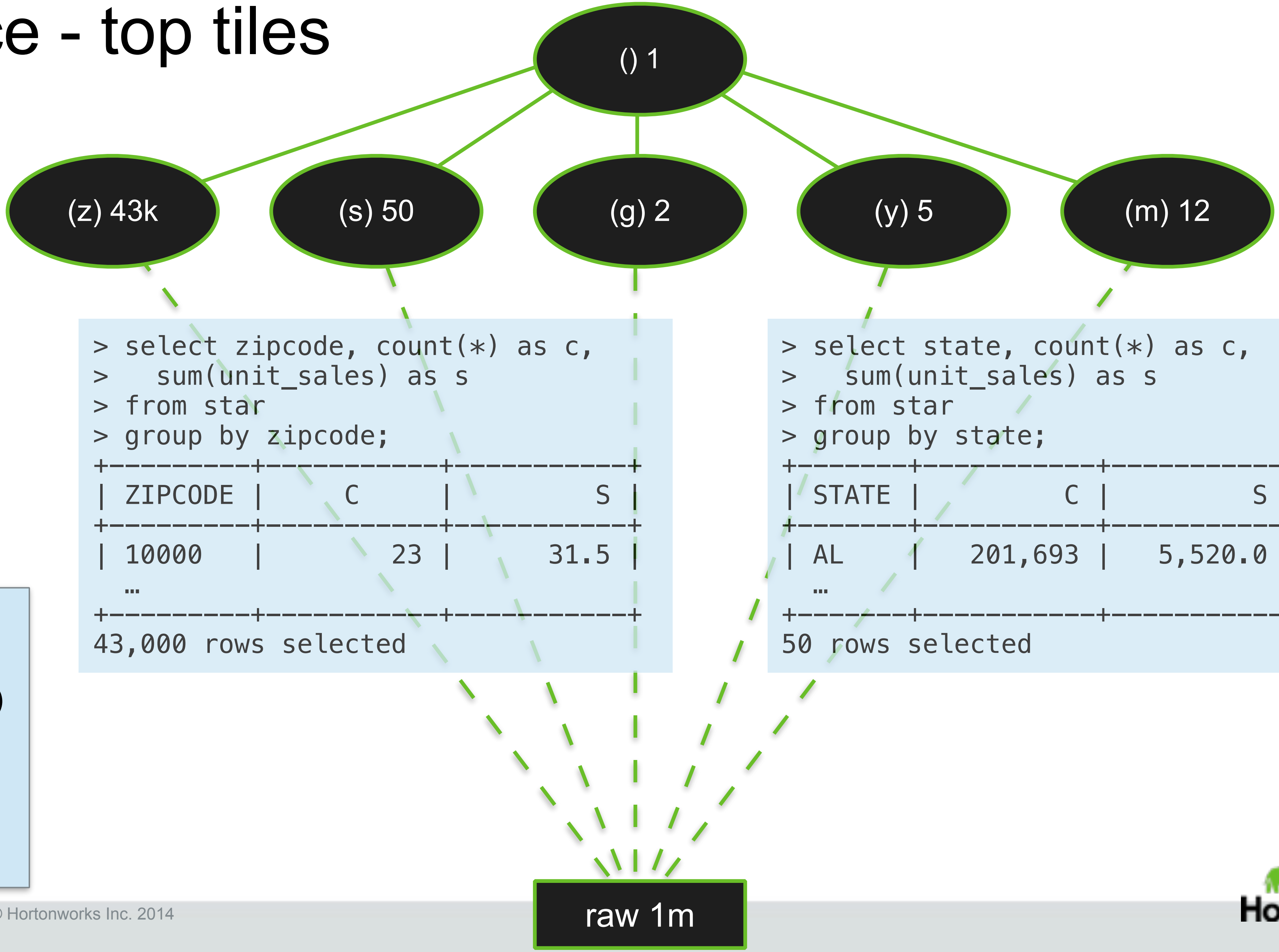
Lattice




```
> select count(*) as c, sum(unit_sales) as s
> from star;
+-----------+-----------+
|     C     |         S |
+-----------+-----------+
| 1,000,000 | 266,773.0 |
+-----------+-----------+
1 row selected

> select * from star;
1,000,000 rows selected
```

Lattice - top tiles



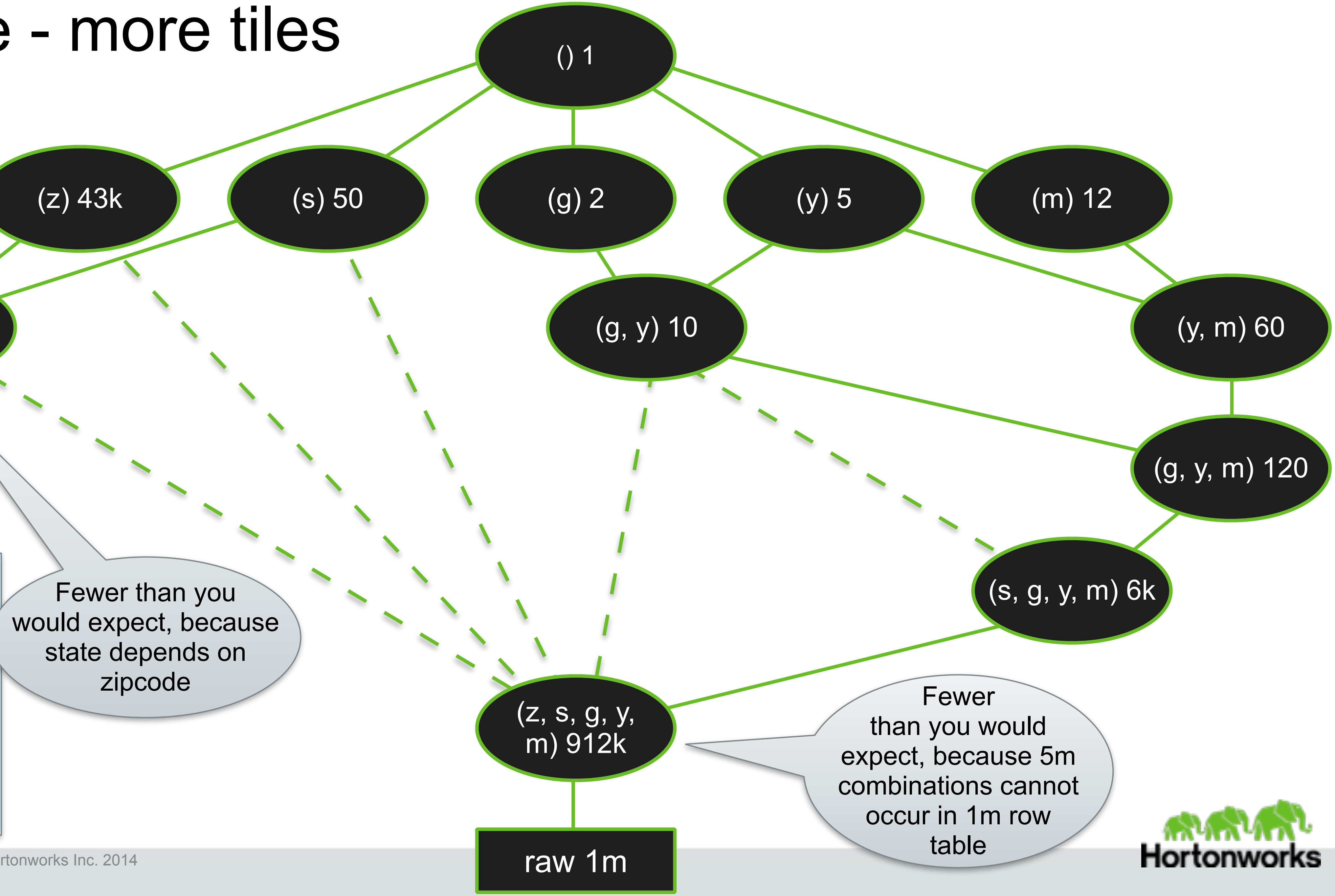

```
> select zipcode, count(*) as c,
>   sum(unit_sales) as s
> from star
> group by zipcode;
+---------+-----------+-----------+
| ZIPCODE |     C     |         S |
+---------+-----------+-----------+
| 10000   |        23 |      31.5 |
  …
+---------+-----------+-----------+
43,000 rows selected
```

```
> select state, count(*) as c,
>   sum(unit_sales) as s
> from star
> group by state;
+-------+-----------+-----------+
| STATE |         C |         S |
+-------+-----------+-----------+
| AL    |   201,693 |   5,520.0 |
  …
+-------+-----------+-----------+
50 rows selected
```

Key

z zipcode (43k)
s state (50)
g gender (2)
y year (5)
m month (12)

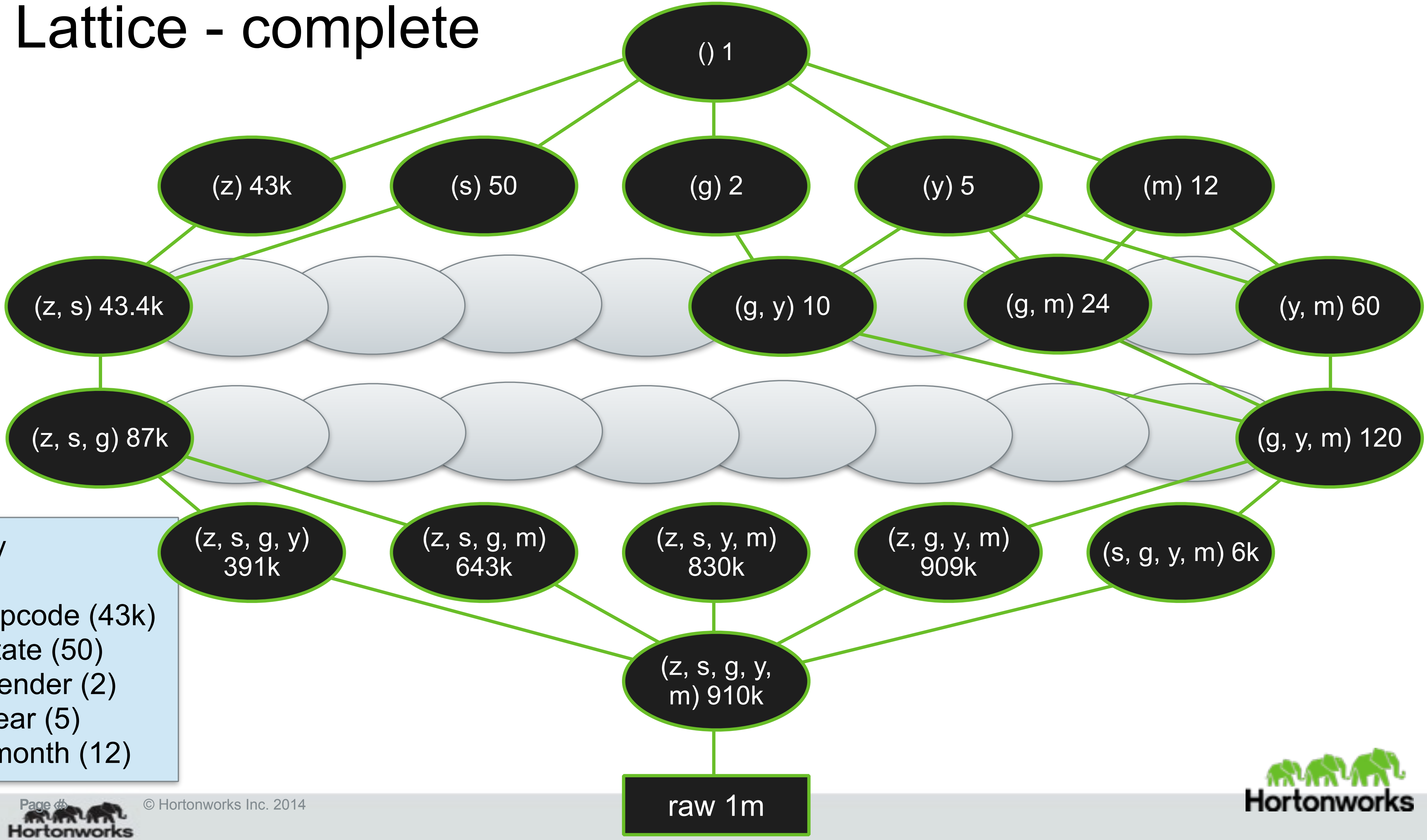
Lattice - more tiles



Key

z zipcode (43k)
s state (50)
g gender (2)
y year (5)
m month (12)

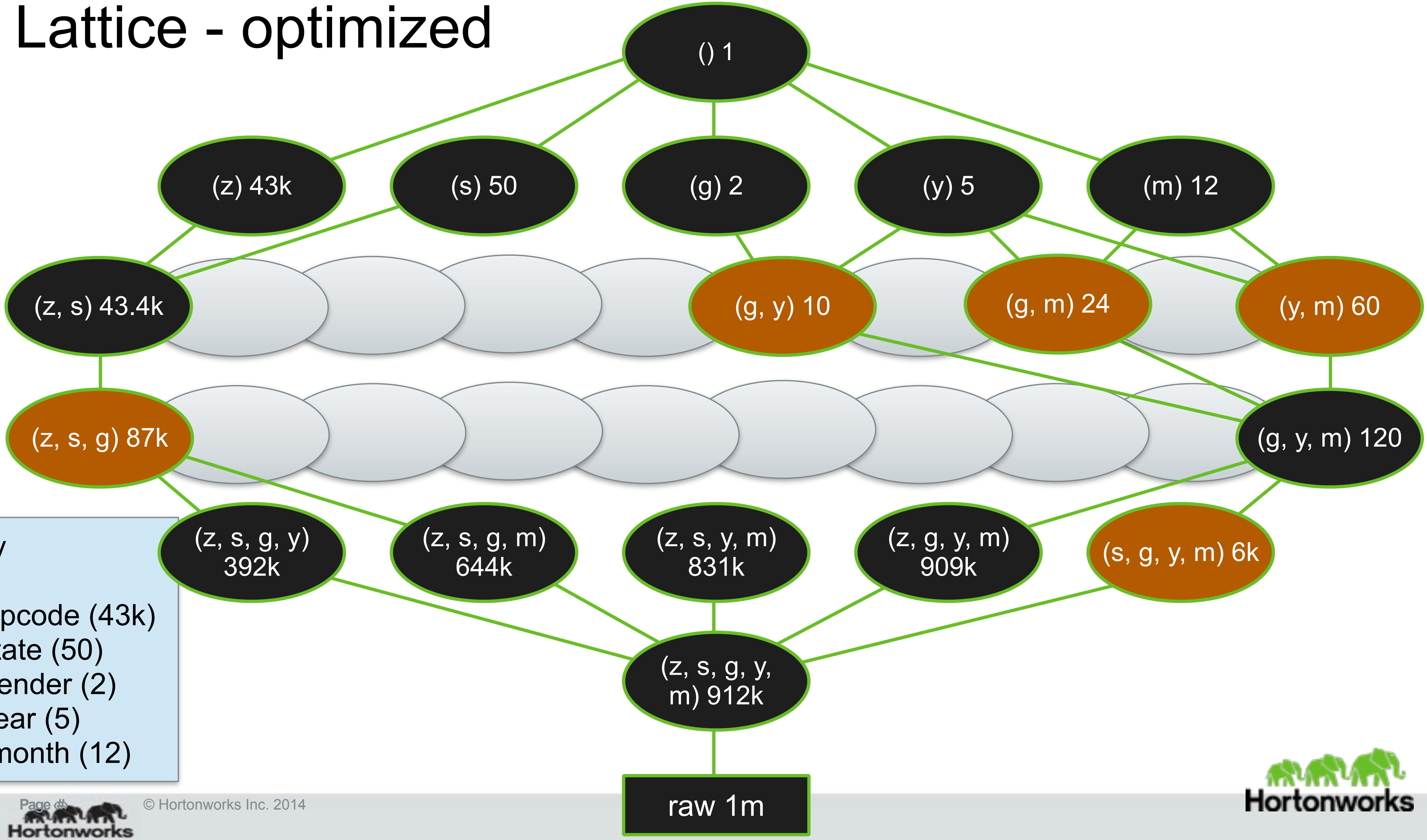
Lattice - complete



Key

z zipcode (43k)
s state (50)
g gender (2)
y year (5)
m month (12)

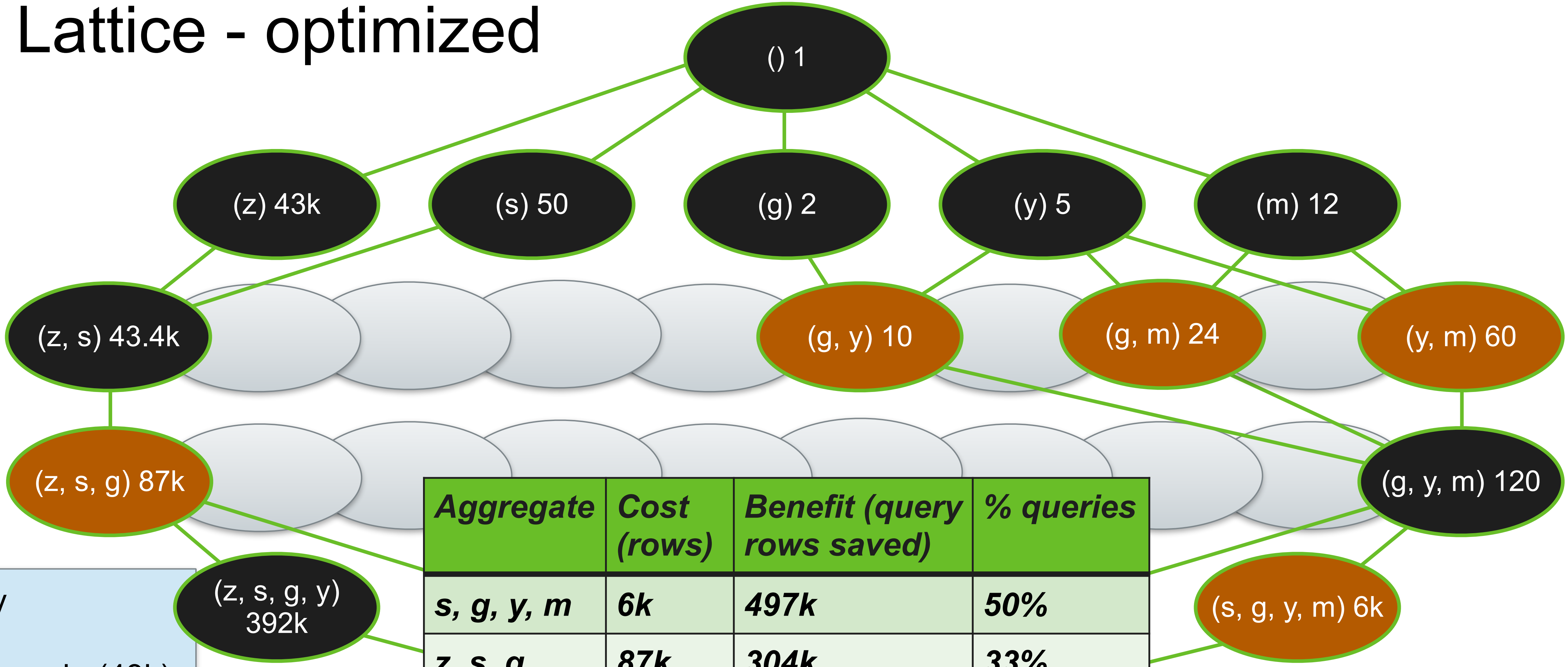
Lattice - optimized



Key

z zipcode (43k)
s state (50)
g gender (2)
y year (5)
m month (12)

Lattice - optimized



Aggregate	*Cost (rows)*	*Benefit (query rows saved)*	*% queries*
s, g, y, m	***6k***	***497k***	***50%***
z, s, g	***87k***	***304k***	***33%***
g, y	***10***	***1.5k***	***25%***
g, m	***24***	***1.5k***	***25%***
s, g	***100***	***1.5k***	***25%***
y, m	***60***	***1.5k***	***25%***

Key

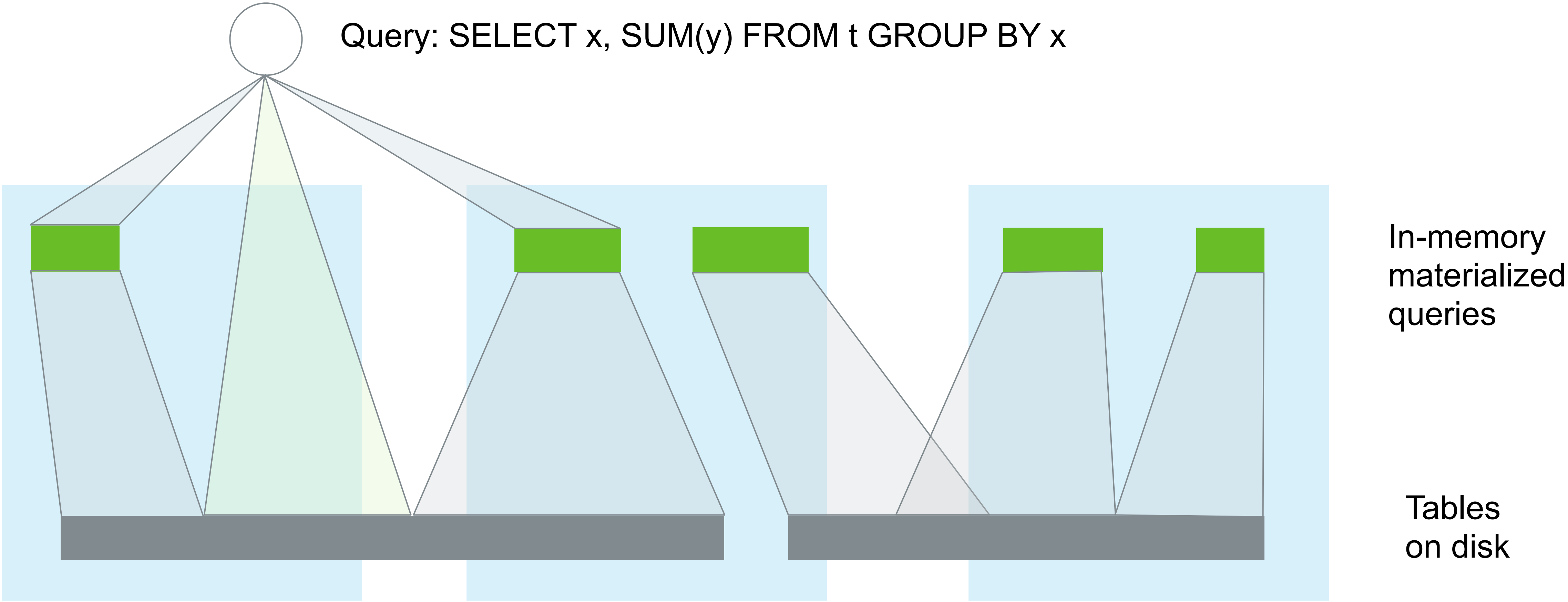
z zipcode (43k)
s state (50)
g gender (2)
y year (5)
m month (12)

Demo

{mysql-foodmart-lattice-model.json}

Hortonworks

Tiled, in-memory materializations

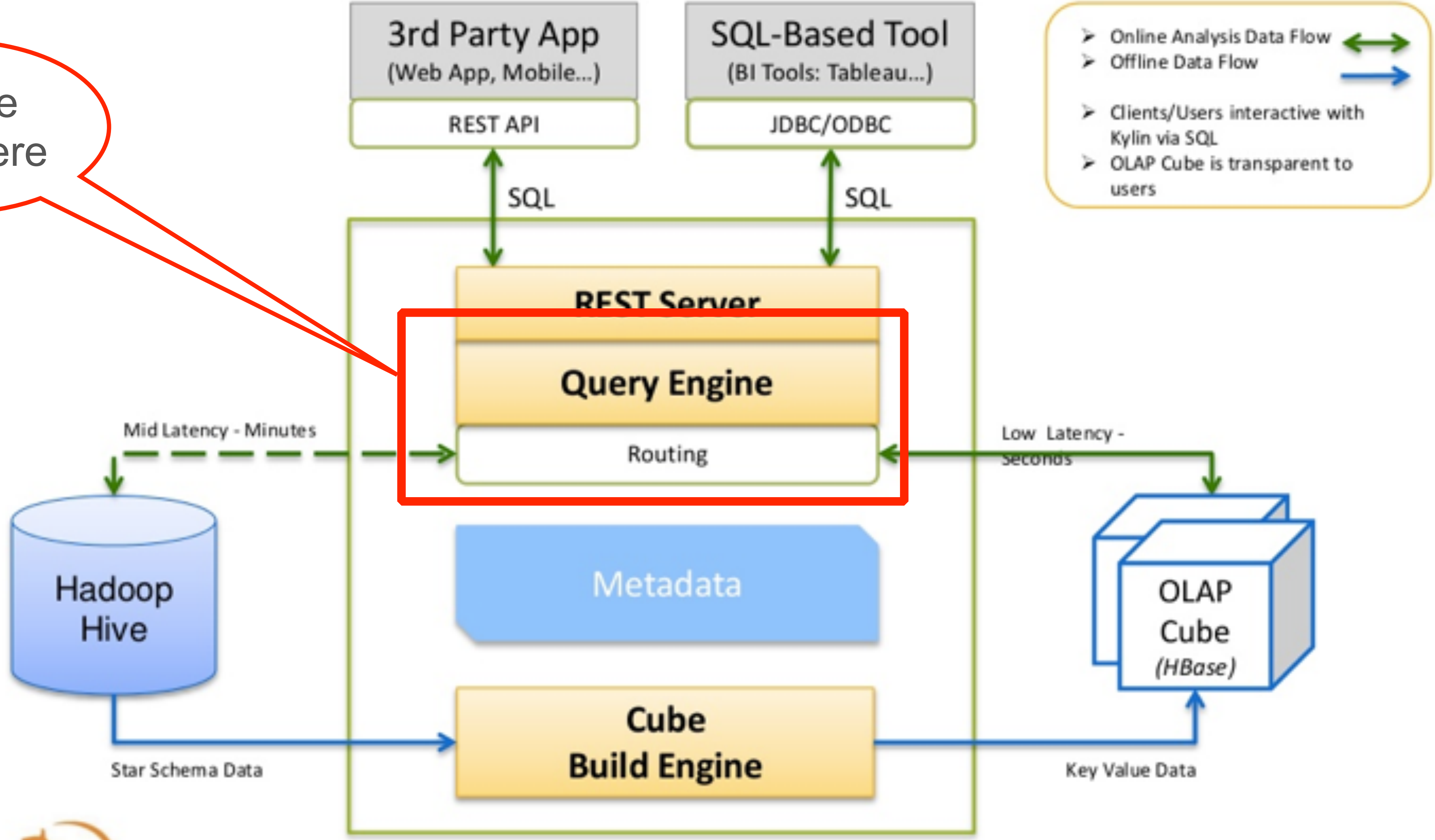


Where we're going… smart, distributed memory cache & compute framework

http://hortonworks.com/blog/dmmq/

Kylin OLAP engine

Calcite used here



Thank you!

@julianhyde

http://calcite.incubator.apache.org

http://www.kylin.io