Why you care about relational algebra (even though you didn't know it)

Julian Hyde

@julianhyde

Enterprise Data World

Washington, DC

April 2nd, 2015

ENTERPRISE DATA WORLD

THE TRANSFORMATION TO DATA-DRIVEN BUSINESS STARTS HERE

GRAND HYATT · WASHINGTON, DC · MARCH 29-APRIL 3, 2015

Hortonworks

About me

mondrian™
pentaho analysis services™

Apache
Calcite

HIVE

APACHE
DRILL

Hortonworks

Why you should care about relational algebra

Why should you care?

- It is old
- It is as useful as ever
- Exposed in new products such as Hadoop
- New challenges

Agenda

- Is Hadoop a revolution for the database world?
- What is relational algebra?
- Examples of algebra in action
- Introducing Apache Calcite
- Adding data independence to Hadoop via materialized views

Old world, new world

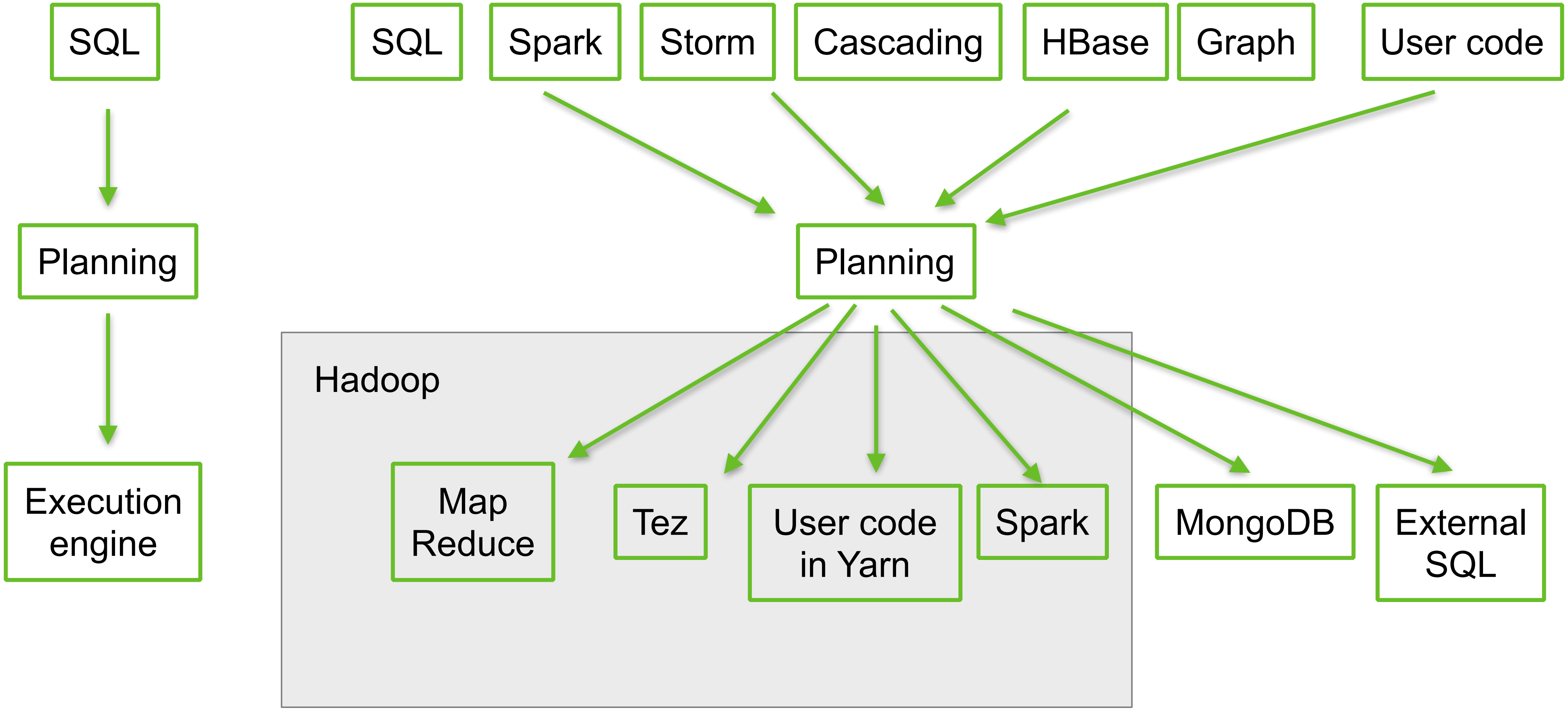
RDBMS

- Security
- Metadata
- SQL
- Query planning
- Data independence

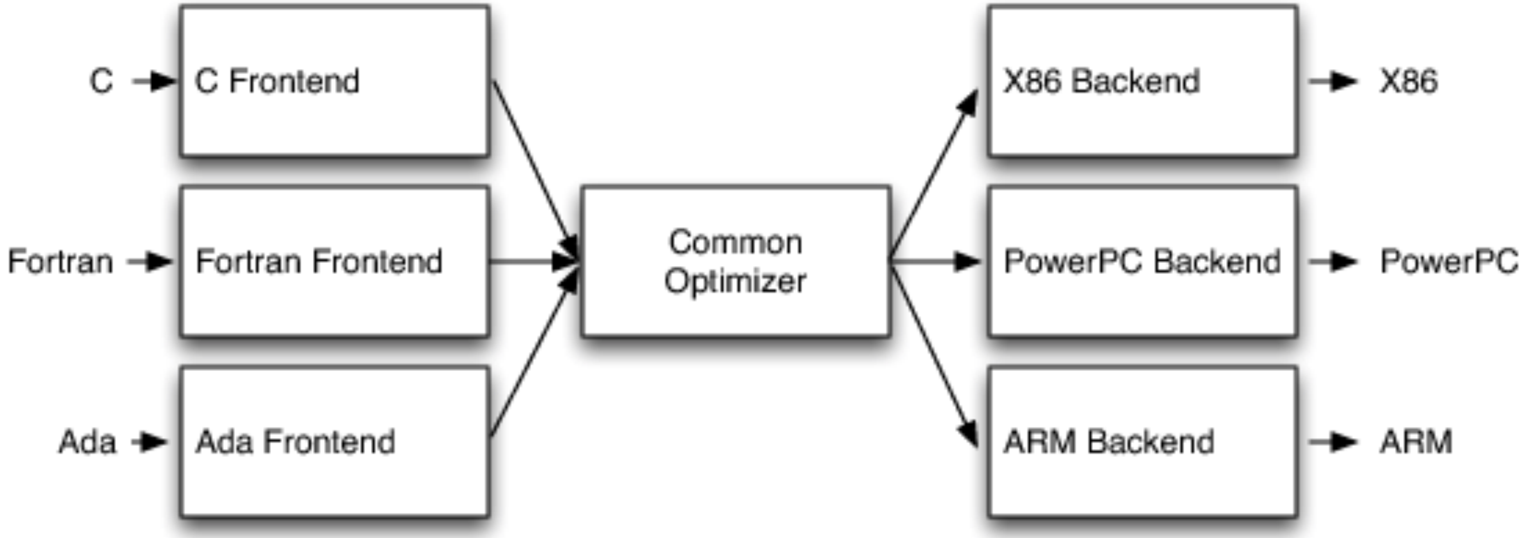
Hadoop

- Scale
- Late schema
- Choice of front-end
- Choice of engines
- Workload: batch, interactive, streaming, ML, graph, …

Many front ends, many engines



Analogy: LLVM

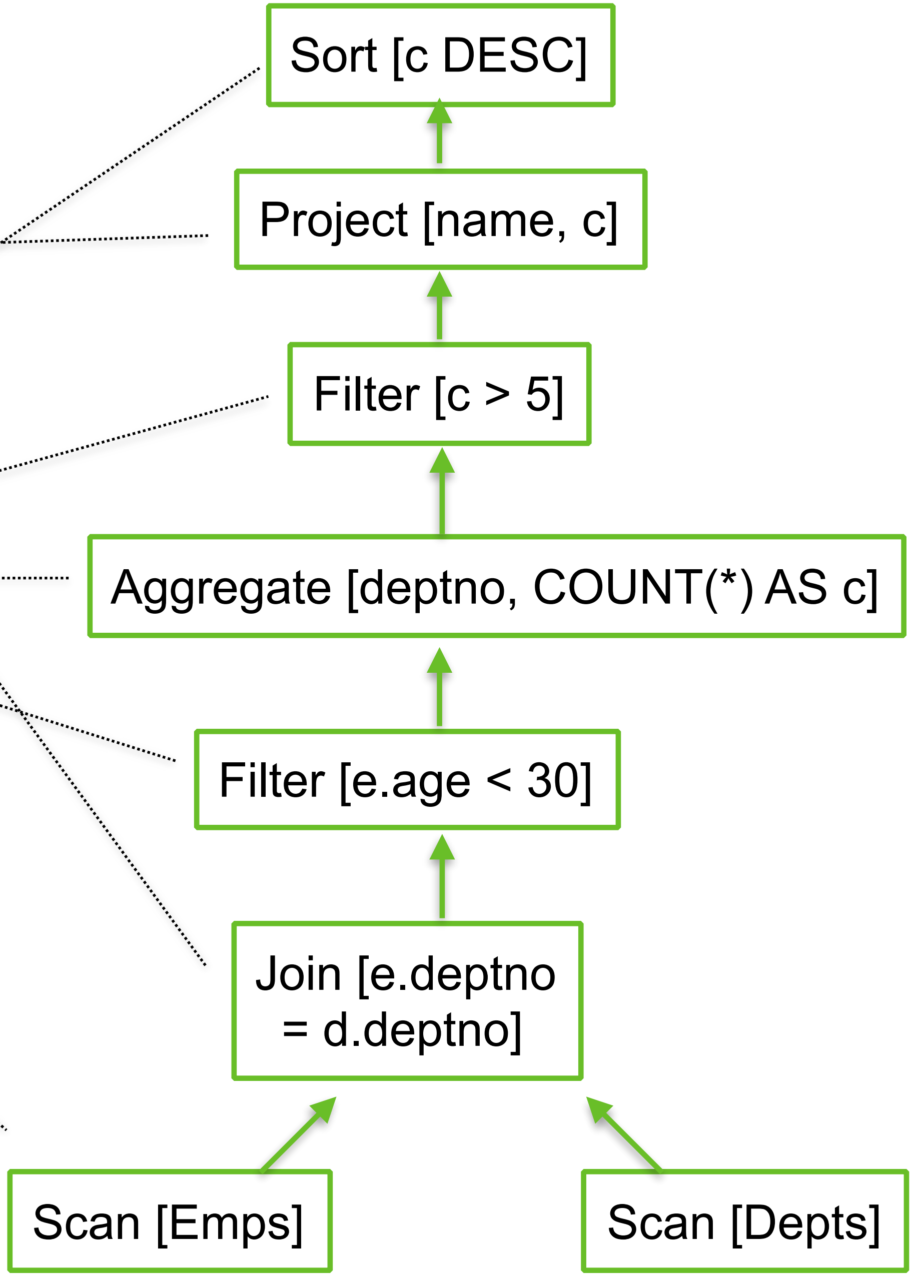


Lessons from the compiler community:

- **Writing a front end is hard**
- **Writing a back end is hard**
- **Writing an optimizer is *really* hard**
- **Most of the logic in the optimizer is independent of front end and back end**
  - **E.g. register assignment**
- **The optimizer is a collection of separate algorithms**
- **Common language between algorithms**

Relational algebra

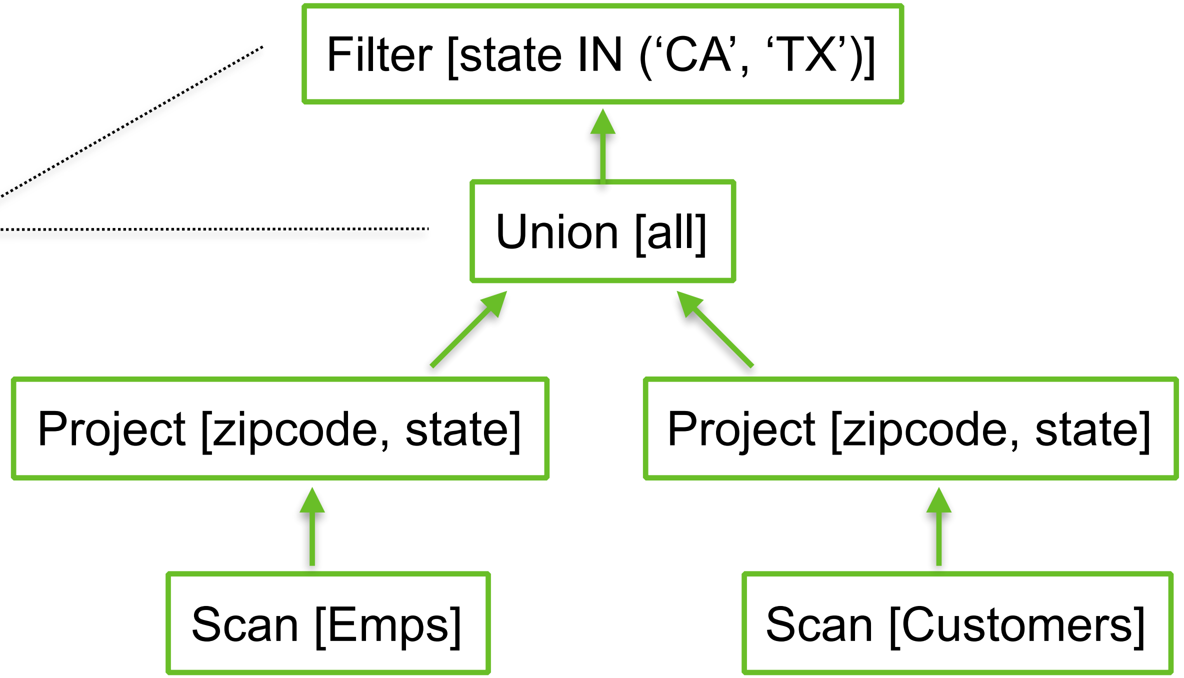
```
SELECT d.name, COUNT(*) AS c
FROM Emps AS e
  JOIN Depts AS d ON e.deptno = d.deptno
WHERE e.age < 30
GROUP BY d.deptno
HAVING COUNT(*) > 5
ORDER BY c DESC
```



(Column names are simplified. They would usually be ordinals, e.g. $0 is the first column of the left input.)

Relational algebra - Union and sub-query

```
SELECT * FROM (
  SELECT zipcode, state
  FROM Emps
  UNION ALL
  SELECT zipcode, state
  FROM Customers)
WHERE state IN ('CA', 'TX')
```



Relational algebra - Insert and Values

INSERT INTO Facts
VALUES (‘Meaning of life’, 42),
(‘Clever as clever’, 6)

Insert [Facts]

Values [[‘Meaning of life’, 42],
[‘Clever as clever’, 6]]

Relational algebra - Strict versus Pragmatic

"Strict" relational algebra

Introduced by E.F. Codd in "A relational model for large shared data banks" [1970]

Goal is mathematical elegance (ability to prove theorems)

Greek symbols: σ, π, ρ, U, ⋈

Relations cannot contain duplicates

Relations are not sorted

Column values are scalars

Only logical operators

Pragmatic relational algebra

Goal is to optimize queries, allow real-world data models, extensibility

Elegance still important

Verbs: Project, Filter, Union, Join

Relations may contain duplicates

Relations may be sorted

- But Sort is the only logical operator that guarantees order

Null values have 3-value semantics, as in SQL

Physical operators (e.g. HashJoin, MergeJoin)

Physical properties (sort, distribution)

Algebraic transformations

(R filter c1) filter c2 → R filter (c1 and c2)

(R1 union R2) join R3 on c → (R1 join R3 on C) union (R2 join R3 on c)

- **Compare distributive law of arithmetic: (x + y) * z → (x * z) + (y * z)**

(R1 join R2 on c) filter c2 → (R1 filter c2) join R2 on c **(provided C2 only depends on columns in E, and join is inner)**

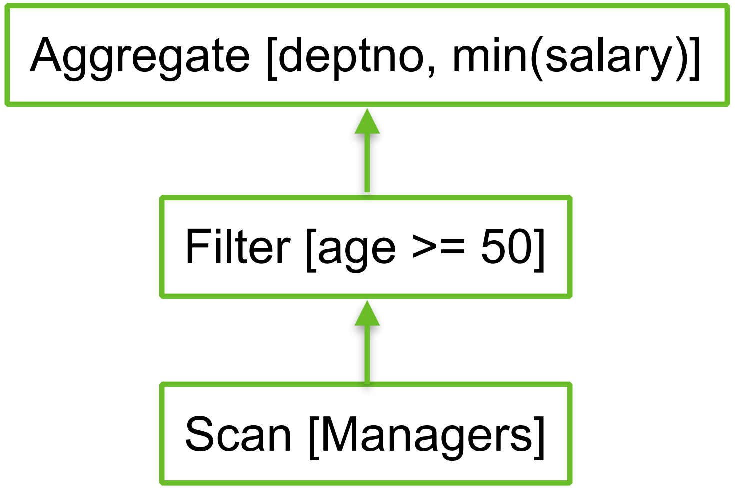
(R1 join R2 on c) → (R2 join R2 on c) project [R1.*, R2.*]

(R1 join R2 on c) join R3 on c2 → R1 join (R2 join R3 on c2) on c **(provided c, c2 have the necessary columns)**

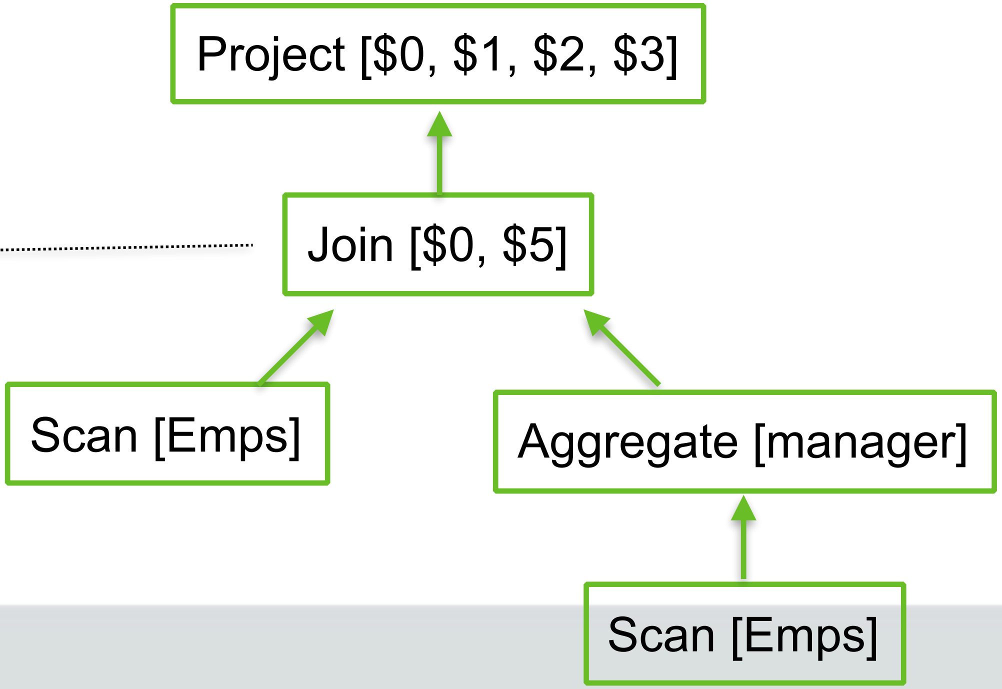
Many, many others…

Query using a view

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



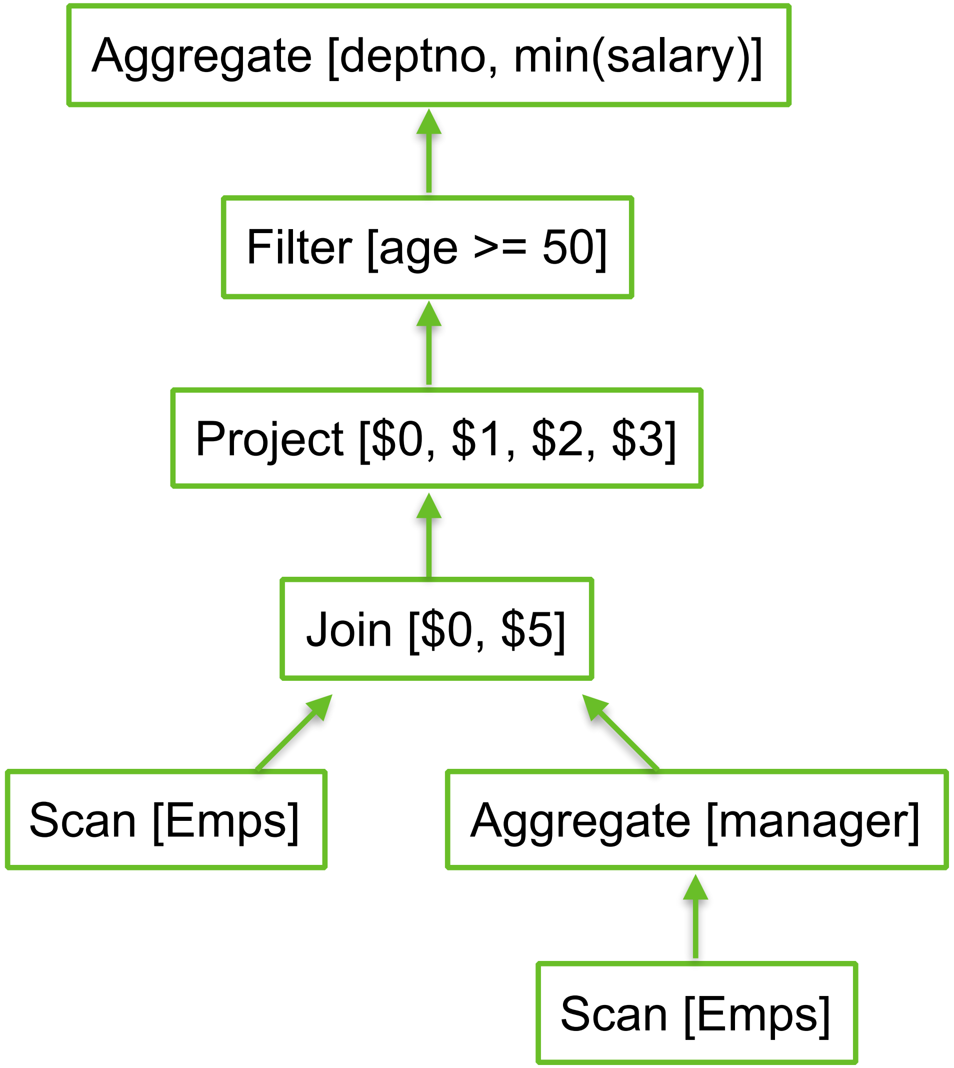

```
CREATE VIEW Managers AS
SELECT *
FROM Emps
WHERE EXISTS (
  SELECT *
  FROM Emps AS underling
  WHERE underling.manager = emp.id)
```



After view expansion

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

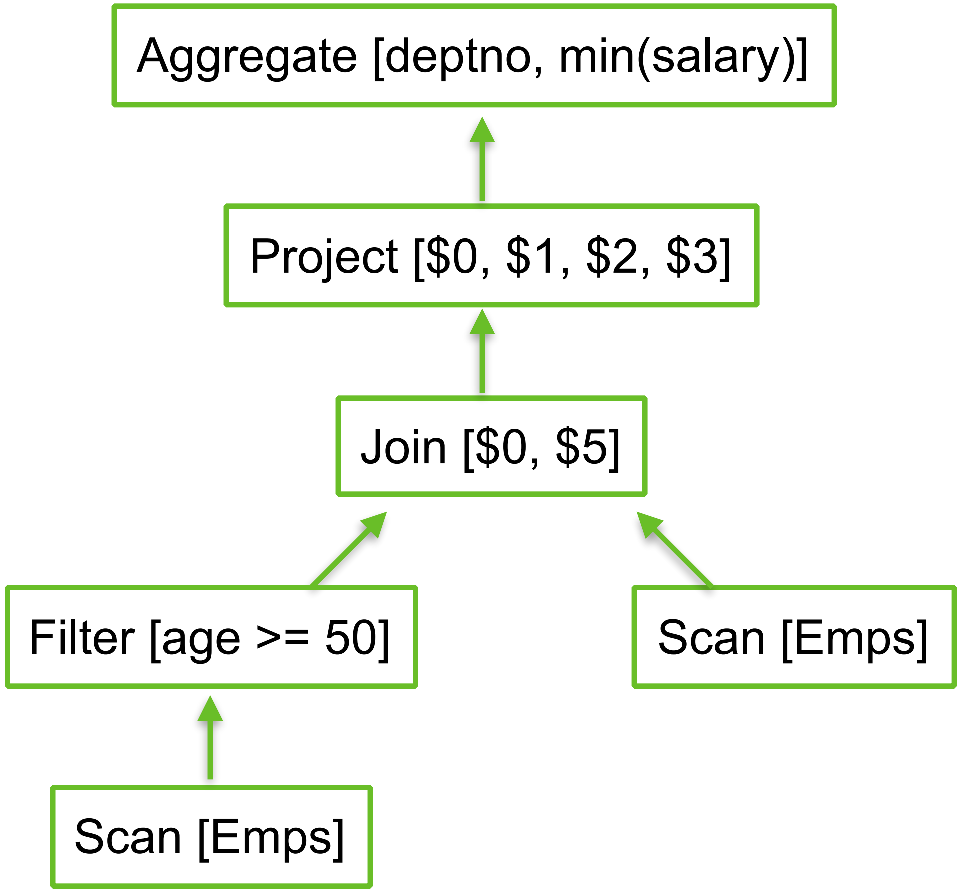
```
CREATE VIEW Managers AS
SELECT *
FROM Emps
WHERE EXISTS (
  SELECT *
  FROM Emps AS underling
  WHERE underling.manager = emp.id)
```



After pushing down filter

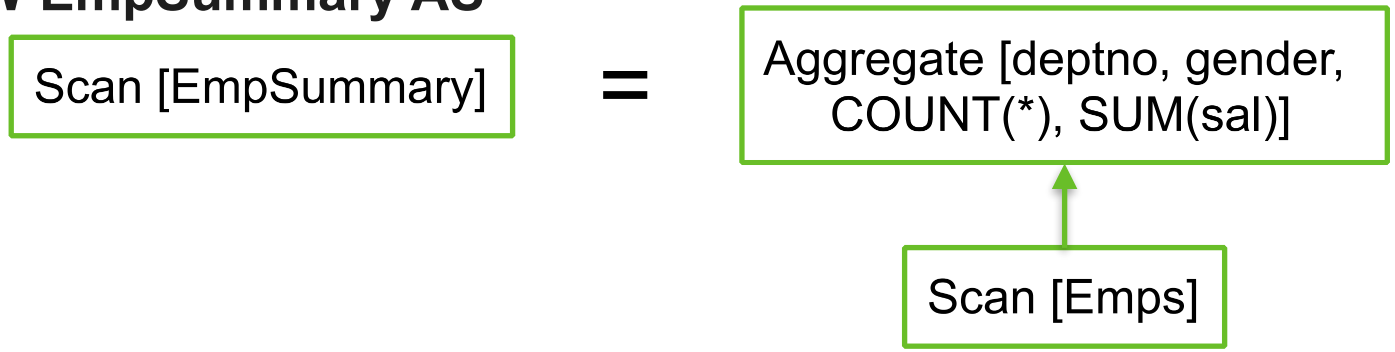
SELECT deptno, min(salary)
FROM Managers
WHERE age >= 50
GROUP BY deptno

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

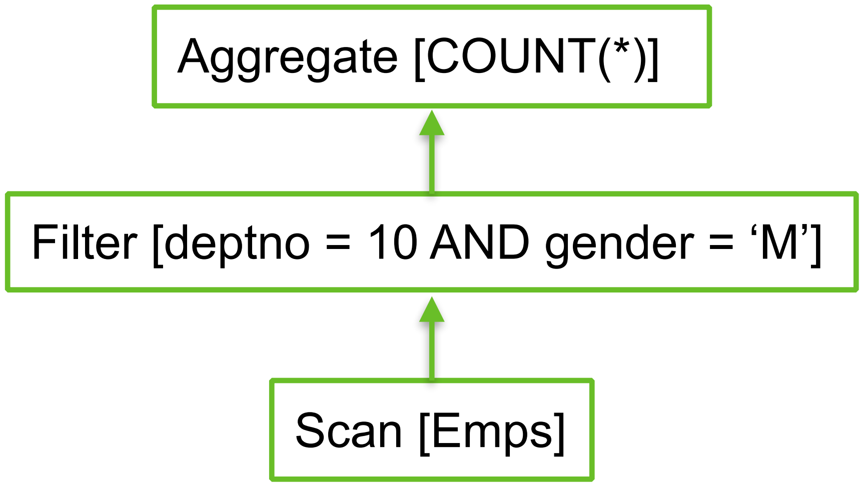


Materialized view

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

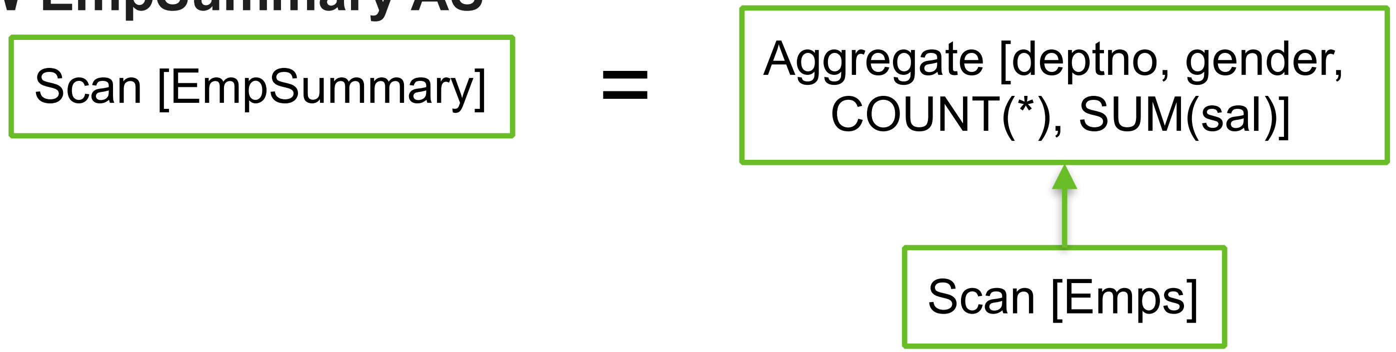



```
SELECT COUNT(*)
FROM Emps
WHERE deptno = 10
AND gender = ‘M’
```

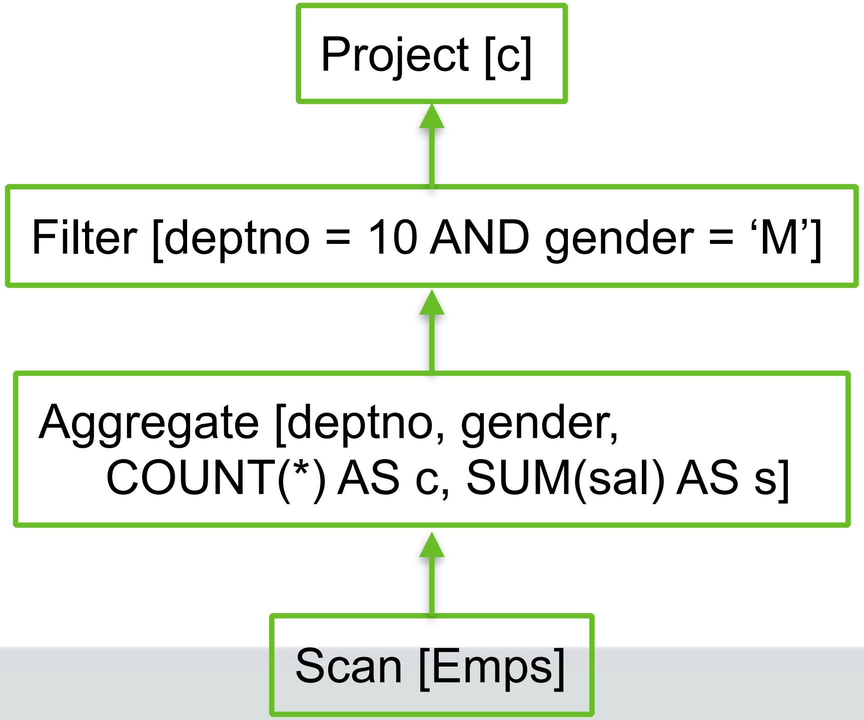


Materialized view, step 2: Rewrite query to match

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

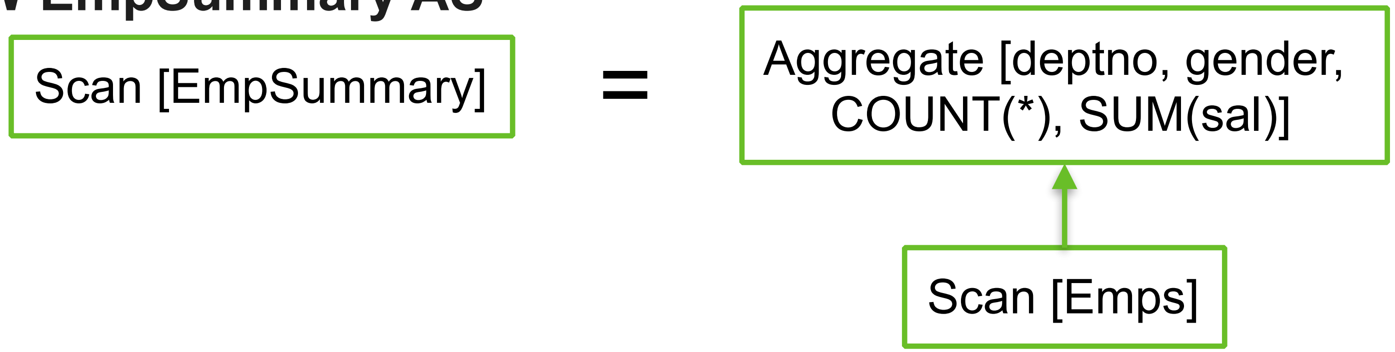


SELECT COUNT(*)
FROM Emps
WHERE deptno = 10
AND gender = ‘M’

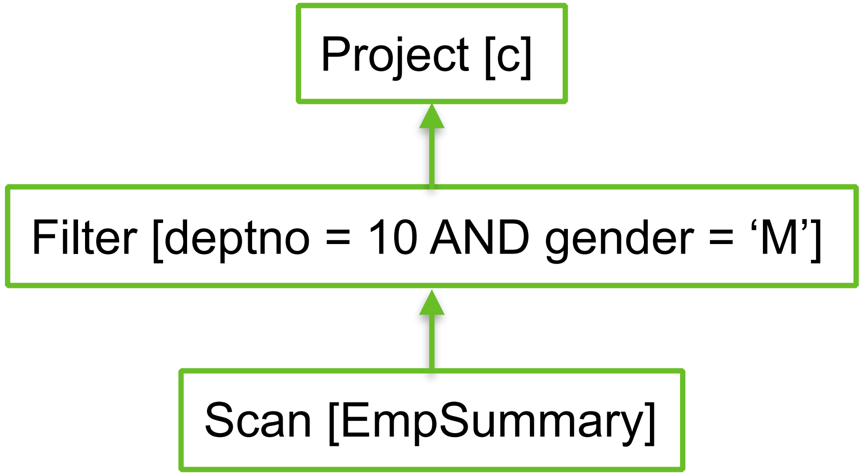


Materialized view, step 3: Substitute table

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

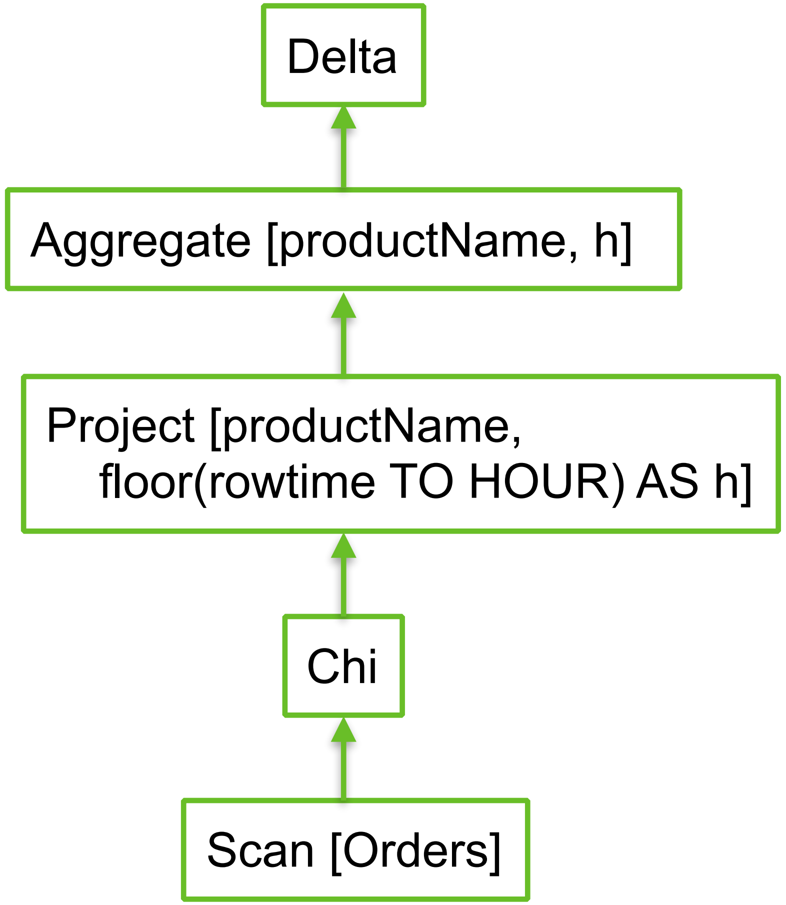



```
SELECT COUNT(*)
FROM Emps
WHERE deptno = 10
AND gender = ‘M’
```



Streaming

SELECT STREAM DISTINCT productName,
floor(rowtime TO HOUR) AS h
FROM Orders



Delta

Converts a table to a stream

Each time a row is inserted into the table, a record appears in the stream

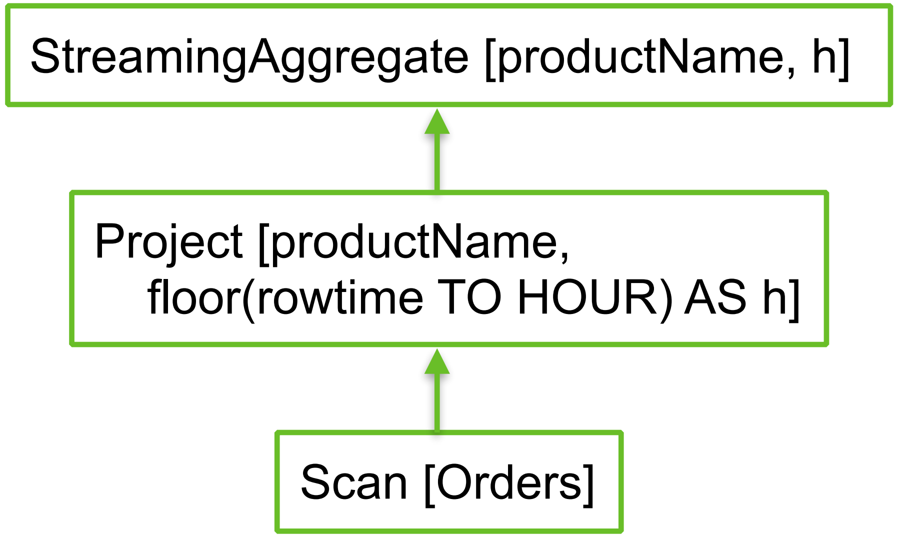
Chi

Converts a stream into a table

Often we can safely narrow the table down to a small time window

Streaming - efficient implementation

SELECT STREAM DISTINCT productName,
floor(rowtime TO HOUR) AS h
FROM Orders



Can create efficient implementation:

- **Input is sorted by timestamp**
- **Only need to aggregate an hour at a time**
- **Output timestamp tracks input timestamp**
- **Therefore it is safe to cancel out the Chi and Delta operators**

Algebraic transformations - streaming

delta(filter(c, R)) → filter(delta(c, R))

delta(project(e1, …, en, R) → project(delta(e1, …, en, R))

delta(union(R1, R2)) → union(delta(R1), delta(R2))

(f + g)' = f' + g'

delta(join(R1, R2, c)) → union(join(R1, delta(R2), c),
join(delta(R1), R2), c)

(f . g)' = f.g' + f'.g

Delta behaves like "differentiate" in differential calculus,
Chi like "integrate".

Apache Calcite

Apache Calcite

Apache incubator project since May, 2014

- Originally named Optiq

Query planning framework

- Relational algebra, rewrite rules, cost model
- Extensible

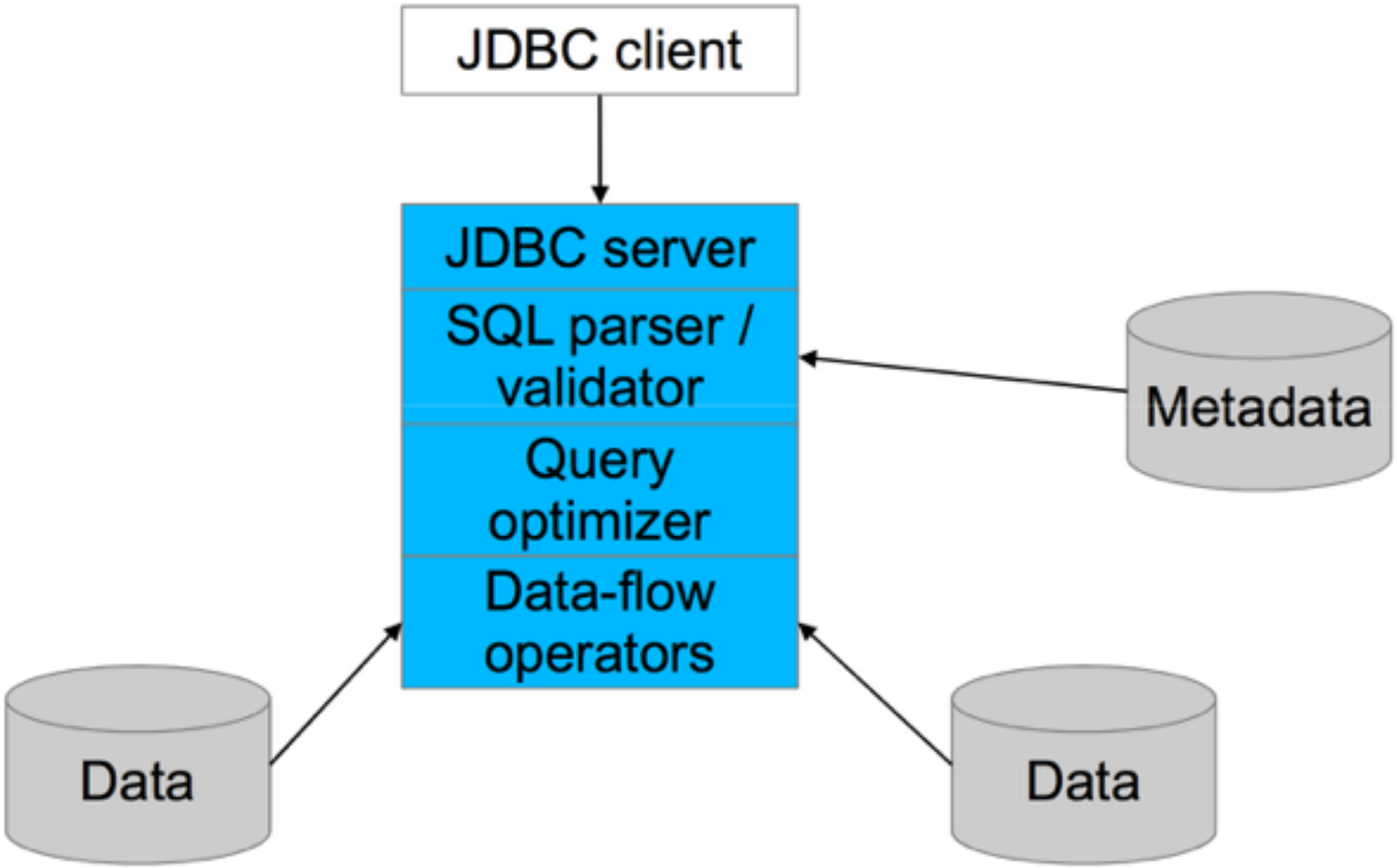
Packaging

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

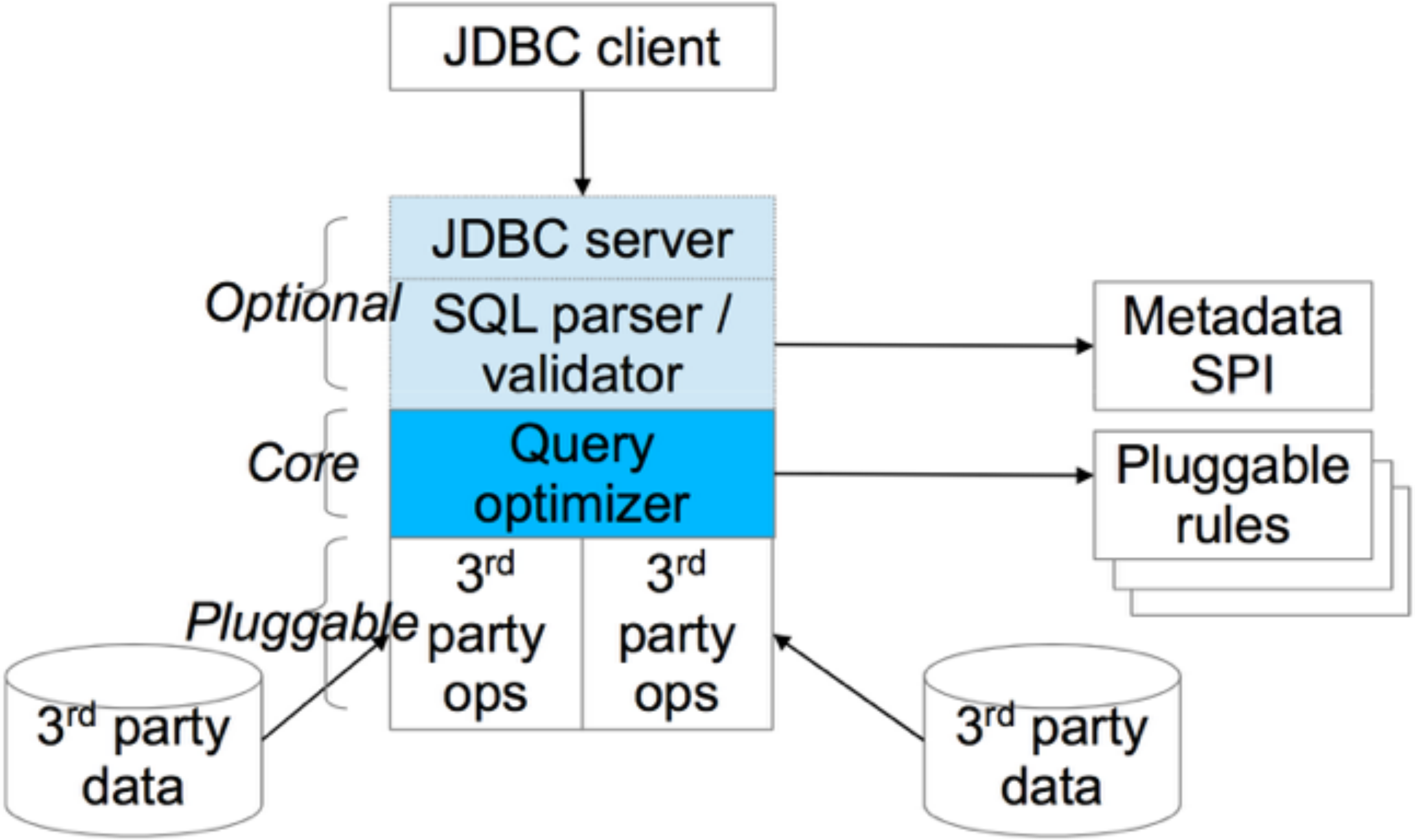
Adoption

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

Conventional DB architecture



Calcite architecture



Calcite – APIs and SPIs

Relational algebra

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

RelDataType (type)
RexNode (expression)
RelTrait (physical property)
- RelConvention (calling-convention)
- RelCollation (sort-order)
- RelDistribution (partitions)

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
- Join ordering

Cost, statistics

RelOptCost
RelOptCostFactory
RelMetadataProvider
- RelMdColumnUniquensss
- RelMdDistinctRowCount
- RelMdSelectivity

Data independence

A core principle of data management

Data independence is a contract:

- **Applications do not make assumptions about the location or organization of data**
- **The DBMS chooses the most efficient access path**

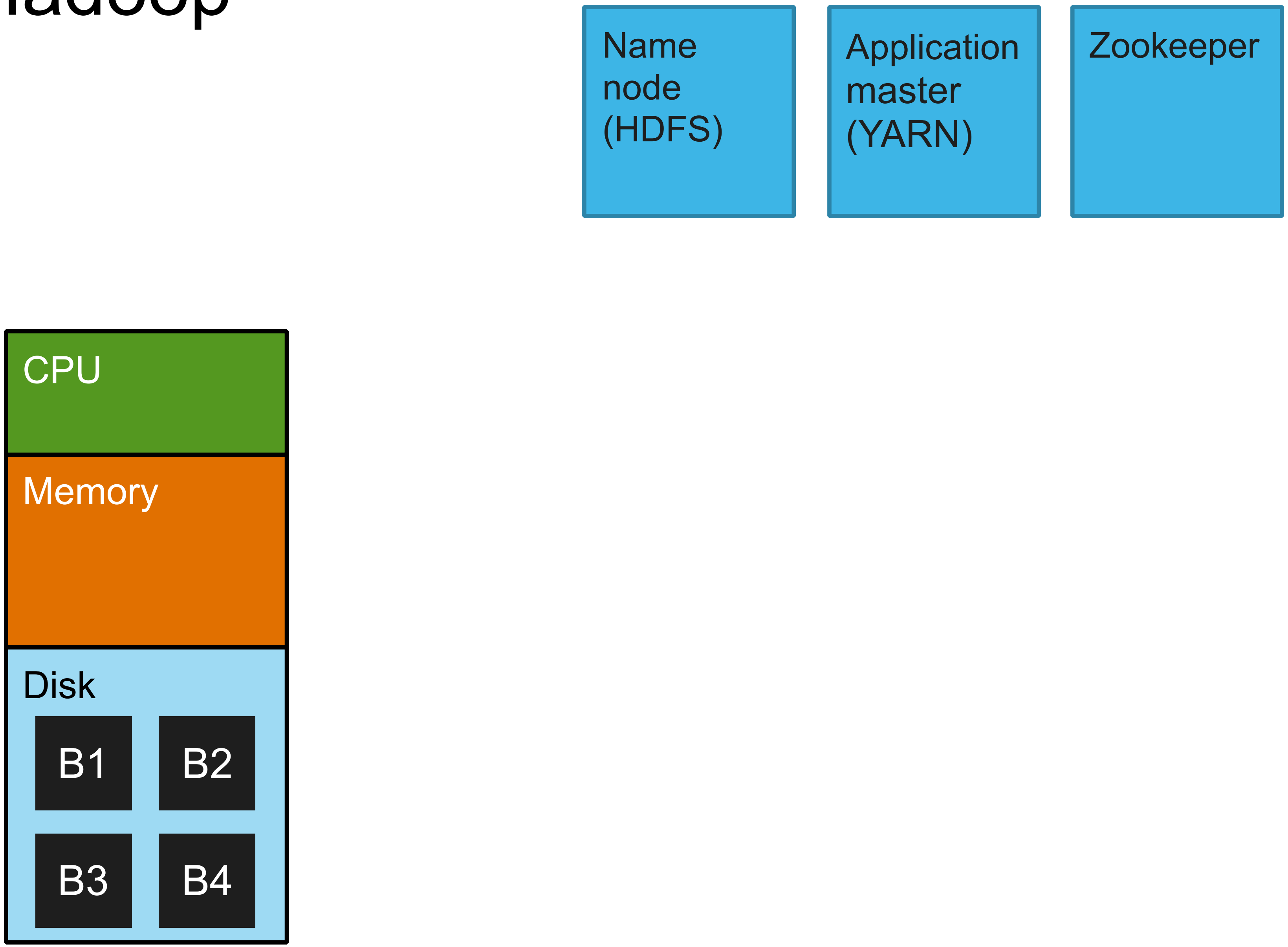
Requires:

- **Declarative query language**
- **Query planner**

Allows:

- **The DBMS (or administrator) can re-organize the data without breaking the application**
- **Redundant copies of the data (indexes, materialized views, replicas)**
- **Novel algorithms**
- **Novel data formats and organizations (e.g. b-tree, r-tree, column store)**

Hadoop

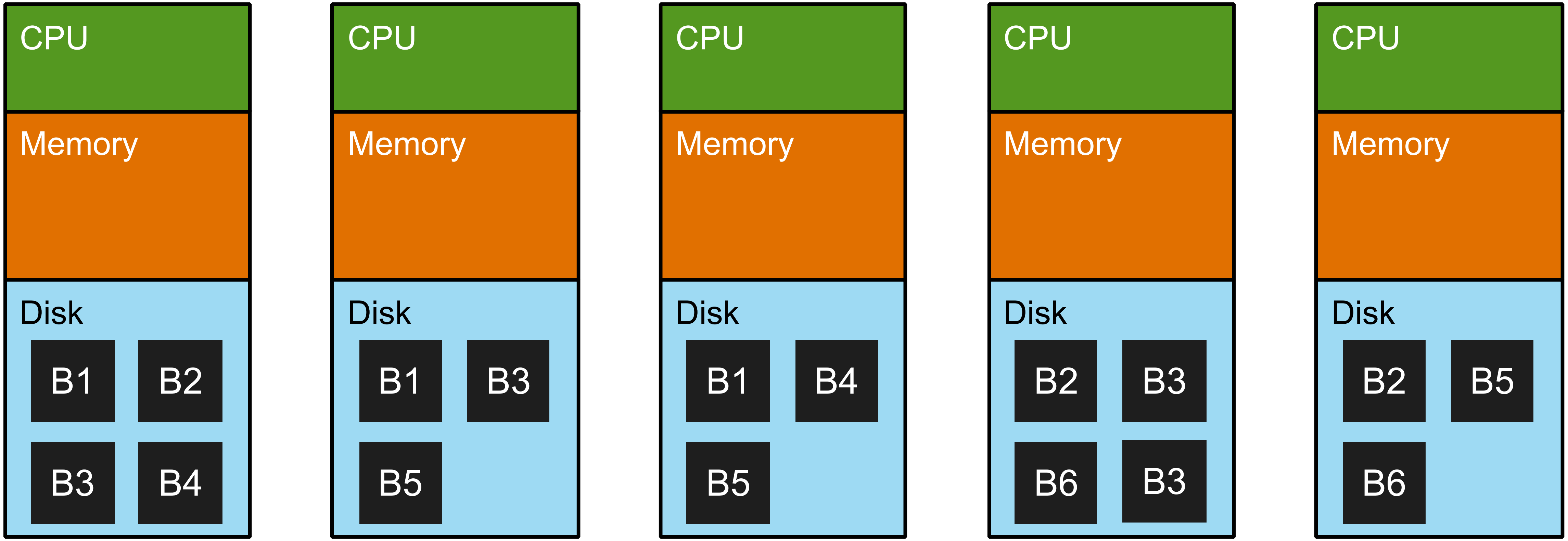

Name node (HDFS)
Application master (YARN)
Zookeeper
CPU
Memory
Disk
B1
B2
B3
B4

Hadoop scales

Commodity hardware

Storage, memory and CPU all scale as you add nodes

N replicas of each block (typically 3) give redundancy & scheduling flexibility

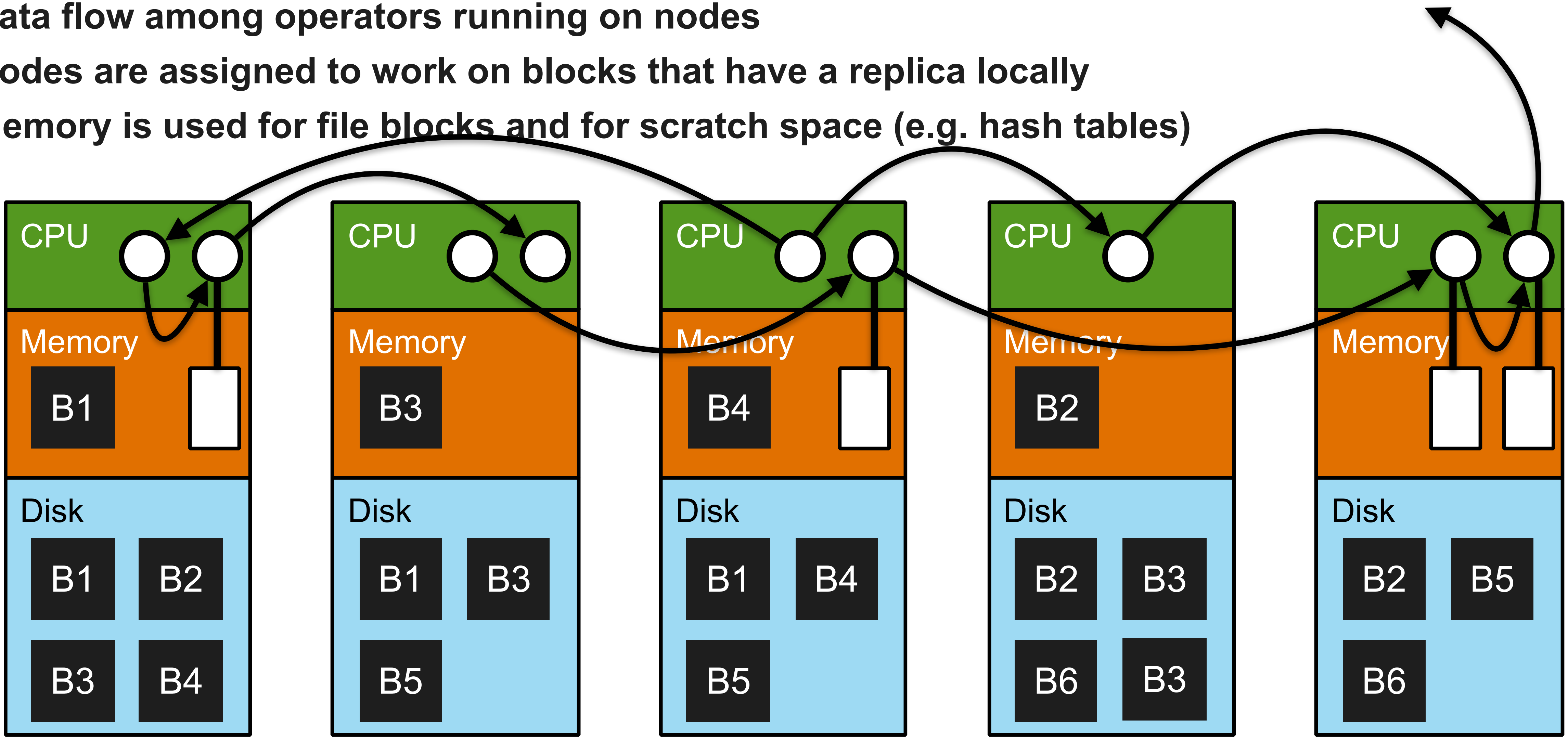


Hadoop query execution

Data flow among operators running on nodes

Nodes are assigned to work on blocks that have a replica locally

Memory is used for file blocks and for scratch space (e.g. hash tables)



Hortonworks

Data independence and Hadoop

Hadoop is very flexible when data is loaded

That flexibility has made it hard for the system to optimize access

Materialized views are an opportunity to “crack” the data, and create copies in other formats

Calcite: Lattices and tiles

(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:

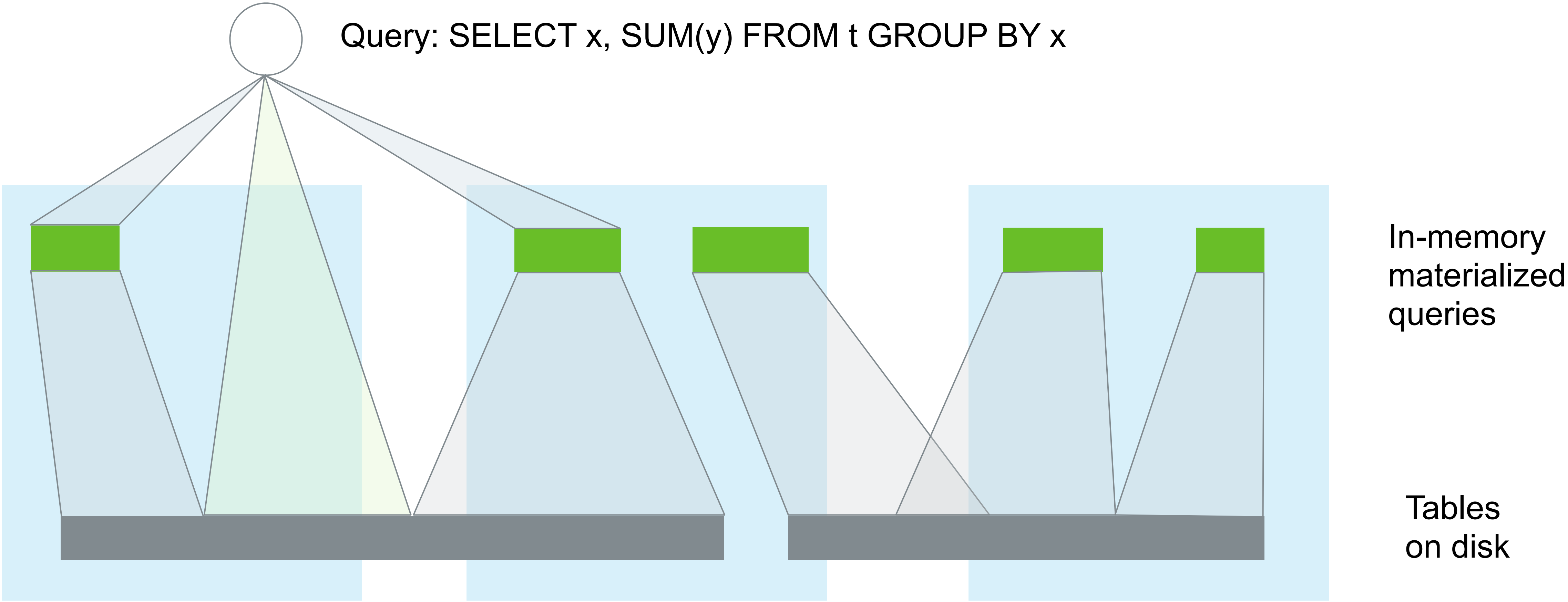
- Declare in lattice
- Generate via recommender algorithm
- Created in response to query

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

CREATE LATTICE star AS
SELECT *
FROM Sales AS s
JOIN Products AS p ON …
JOIN ProductClasses AS pc ON …
JOIN Customers AS c ON …
JOIN Time AS t ON …;

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

Tiled, in-memory materializations



Where we’re going… algebraic cache: http://hortonworks.com/blog/dmmq/

Summary

1. **Relational algebra allows us to reason about queries, and is the foundation of query planning**
2. **Hadoop is deconstructing the DBMS, and enabling new languages, engines and data formats**
3. **Data independence is more important than ever**
4. **Apache Calcite - an implementation of relational algebra**

Thank you!

@julianhyde

http://calcite.incubator.apache.org