

# Discardable, In-Memory Materialized Query for Hadoop



Julian Hyde  
June 3<sup>rd</sup>, 2014

# About me

**Julian Hyde**

**Architect at Hortonworks**

## **Open source:**

- Founder & lead, Apache Optiq (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)

# Before we get started...

## The bad news

- This software is not available to download
- I am a database bigot
- I am a BI bigot

## The good news

- I believe in Hadoop
- Now is a great time to discuss where Hadoop is going

# Hadoop today

## Brute force

Hadoop brings a lot of CPU, disk, IO

Yarn, Tez, Vectorization are making Hadoop faster

How to use that brute force is left to the application

## Business Intelligence

Best practice is to pull data out of Hadoop

- Populate enterprise data warehouse
- In-memory analytics
- Custom analytics, e.g. Lambda architecture

## Ineffective use of memory

## Opportunity to make Hadoop smarter

# Brute force + diplomacy



# Hardware trends

## Typical Hadoop server configuration

Year	2009	2014
Cores	4 – 8	24
Memory	8 GB	128 GB
SSD	None	1 TB
Disk	4 x 1 TB	12 x 4 TB
Disk : memory	512 : 1	384 : 1

Lots of memory - but not enough for all data

More heterogeneous mix (disk + SSD + memory)

## What to do about memory?

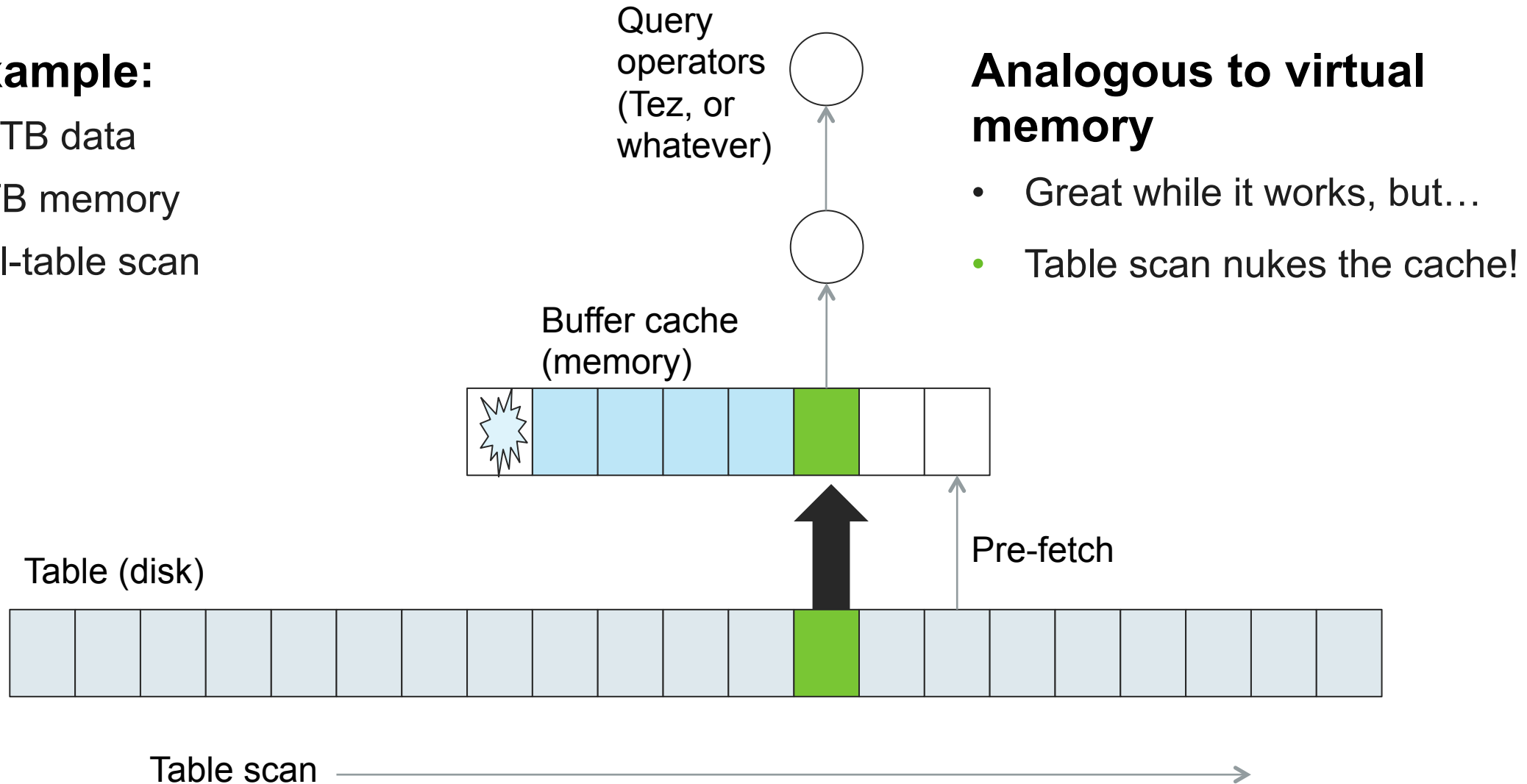
# Dumb use of memory - Buffer cache

## Example:

50 TB data

1 TB memory

Full-table scan

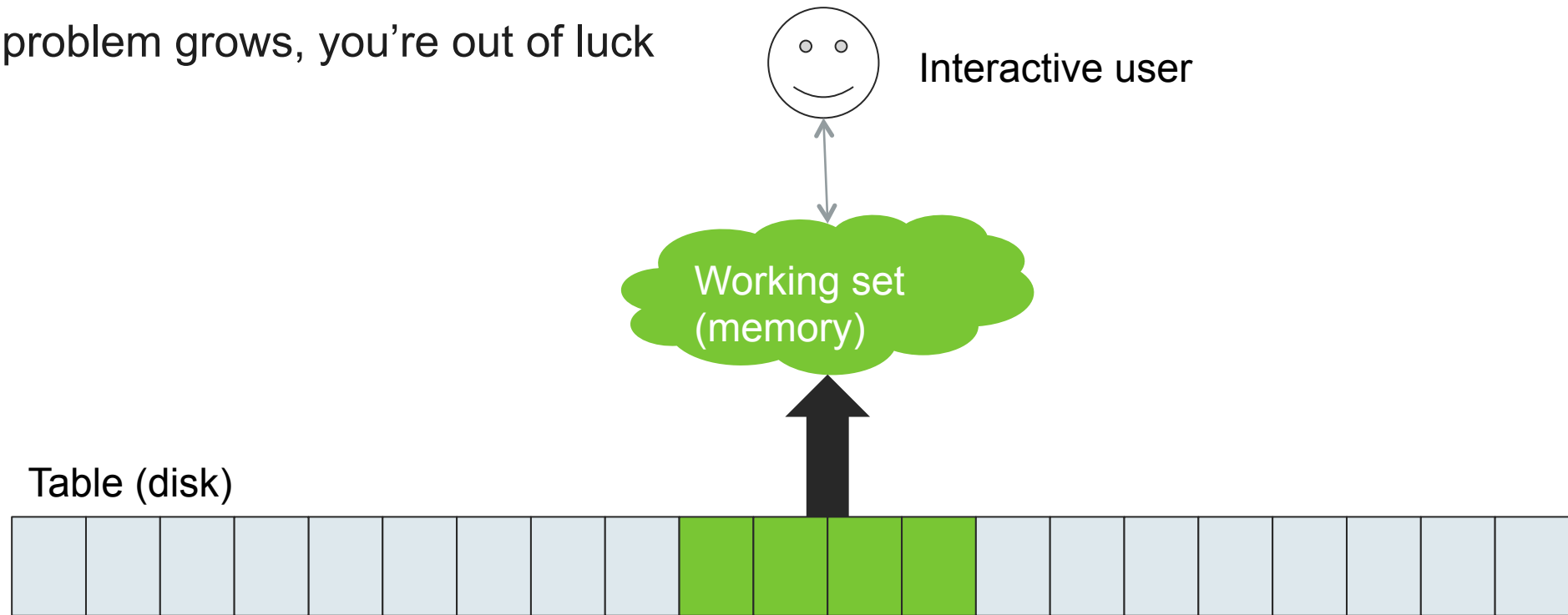


# “Document-oriented” analysis

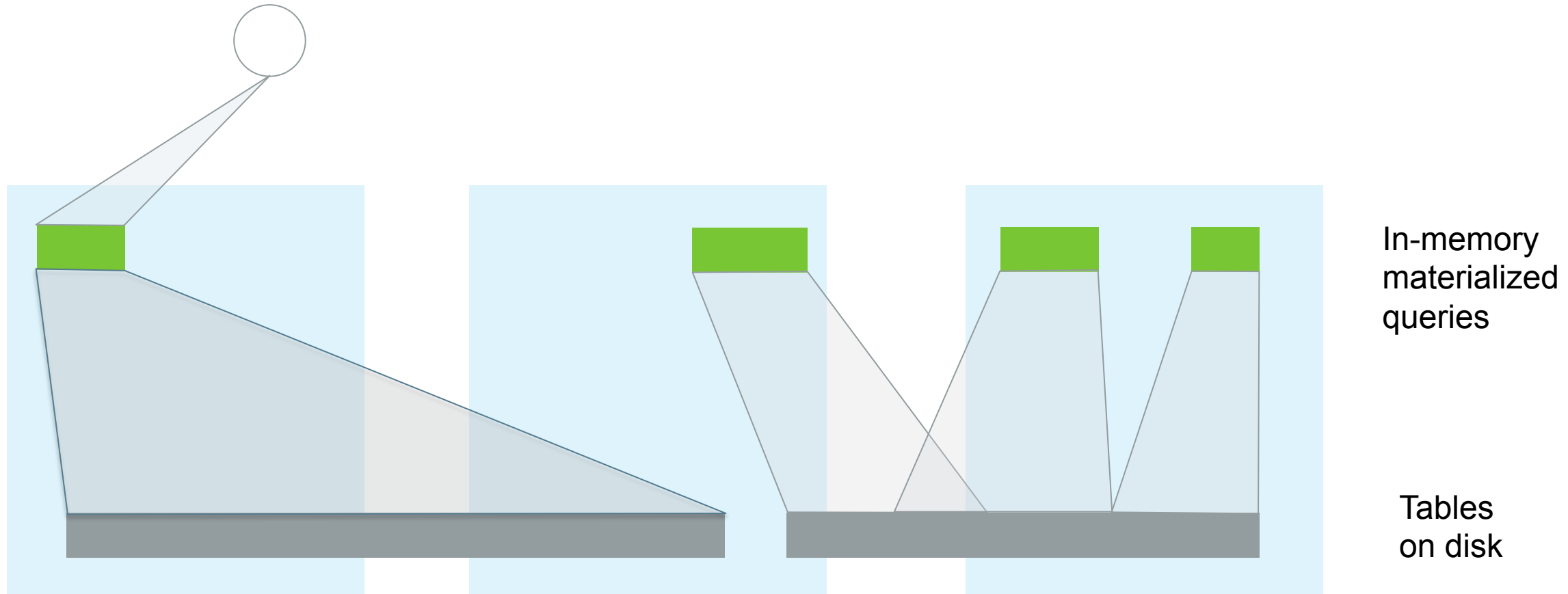
**Operate on working sets small enough to fit in memory**

**Analogous to working on a document (e.g. a spreadsheet)**

- Works well for problems that fit into memory (e.g. some machine-learning algorithms)
- If your problem grows, you’re out of luck



# Smarter use of memory - Materialized queries



# Census data

**Census table – 300M records**

**Which state has the most males?**

**Brute force – read 150M records**

**Smarter – read 50 records**

```
CREATE TABLE Census (id, gender,
                        zipcode, state, age);
```

```
SELECT state, COUNT(*) AS c
FROM Census
WHERE gender = 'M'
GROUP BY state
ORDER BY c DESC LIMIT 1;
```

```
CREATE TABLE CensusSummary AS
SELECT state, gender, COUNT(*) AS c
FROM Census
GROUP BY state, gender;
```

```
SELECT state, c
FROM CensusSummary
WHERE gender = 'M'
ORDER BY c DESC LIMIT 1;
```

# Materialized view – Automatic smartness

**A materialized view is a table that is declared to be identical to a given query**

```
CREATE MATERIALIZED VIEW CensusSummary
  STORAGE (MEMORY, DISCARDABLE) AS
SELECT state, gender, COUNT(*) AS c
FROM Census
GROUP BY state, gender;
```

**Optimizer rewrites query on “Census” to use “CensusSummary” instead**

```
SELECT state, COUNT(*) AS c
FROM Census
WHERE gender = 'M'
GROUP BY state
ORDER BY c DESC LIMIT 1;
```

**Even smarter – read 50 records**

# Indistinguishable from magic

## Run query #1

```
SELECT state, COUNT(*) AS c
FROM Census
WHERE gender = 'M'
GROUP BY state
ORDER BY c DESC LIMIT 1;
```

## System creates materialized view in background

```
CREATE MATERIALIZED VIEW CensusSummary
  STORAGE (MEMORY, DISCARDABLE) AS
SELECT state, gender, COUNT(*) AS c
FROM Census
GROUP BY state, gender;
```

## Related query #2 runs faster

```
SELECT state,
  COUNT(NULLIF(gender, 'F')) AS males,
  COUNT(NULLIF(gender, 'M')) AS females
FROM Census
GROUP BY state
HAVING females > males;
```

# Materialized views - Classic

## Classic materialized view (Oracle, DB2, Teradata, MSSql)

1. A table defined using a SQL query
2. Designed by DBA
3. Storage same as a regular table
  1. On disk
  2. Can define indexes
4. DB populates the table
5. Queries are rewritten to use the table\*\*
6. DB updates the table to reflect changes to source data (usually deferred)\*

\*Magic required

```
CREATE MATERIALIZED VIEW SalesMonthZip AS
SELECT t.year, t.month,
       c.state, c.zipcode,
       COUNT(*), SUM(s.units), SUM(s.price)
FROM SalesFact AS s
JOIN TimeDim AS t USING (timeId)
JOIN CustomerDim AS c USING (customerId)
GROUP BY t.year, t.month,
         c.state, c.zipcode;
```

```
SELECT t.year, AVG(s.units)
FROM SalesFact AS s
JOIN TimeDim AS t USING (timeId)
GROUP BY t.year;
```

# Materialized views - DIMMQ

## **DIMMQs - Discardable, In-memory Materialized Queries**

### **Differences with classic materialized views**

1. May be in-memory
2. HDFS may discard – based on DDM (Distributed Discardable Memory)
3. Lifecycle support:
  1. Assume table is populated
  2. Don't populate & maintain
  3. User can flag as valid, invalid, or change definition (e.g. date range)
  4. HDFS may discard
4. More design options:
  1. DBA specifies
  2. Retain query results (or partial results)
  3. An agent builds MVs based on query traffic

# DIMMQ compared to Spark RDDs

	Spark RDD	DIMMQ
Access	By reference	By algebraic expression
Sharing	Within session	Across sessions
Execution model	Built-in	External
Recovery	Yes	No
Discard	Failure or GC	Failure or cost-based
Native organization	Memory	Disk
Language	Scala (or other JVM language)	Language-independent

## It's not “either / or”

- Spark-on-YARN, SQL-on-Spark already exist; Cascading-on-Spark common soon
- Hive-queries-on-RDDs, DIMMQs populated by Spark, Spark-on-Hive-tables are possible

# Data independence

## **This is not just about SQL standards compliance!**

Materialized views are supposed to be transparent in creation, maintenance and use.  
If not one DBA ever types “CREATE MATERIALIZED VIEW”, we have still succeeded

## **Data independence**

Ability to move data around and not tell your application

Replicas

Redundant copies

Moving between disk and memory

Sort order, projections (à la Vertica), aggregates (à la Microstrategy)

Indexes, and other weird data structures

# Implementing DIMMQs

## Relational algebra

Apache Optiq (just entered incubator – yay!)

Algebra, rewrite rules, cost model

## Metadata

Hive: “CREATE MATERIALIZED VIEW”

Definitions of materialized views in HCatalog

## HDFS - Discardable Distributed Memory (DDM)

Off-heap data in memory-mapped files

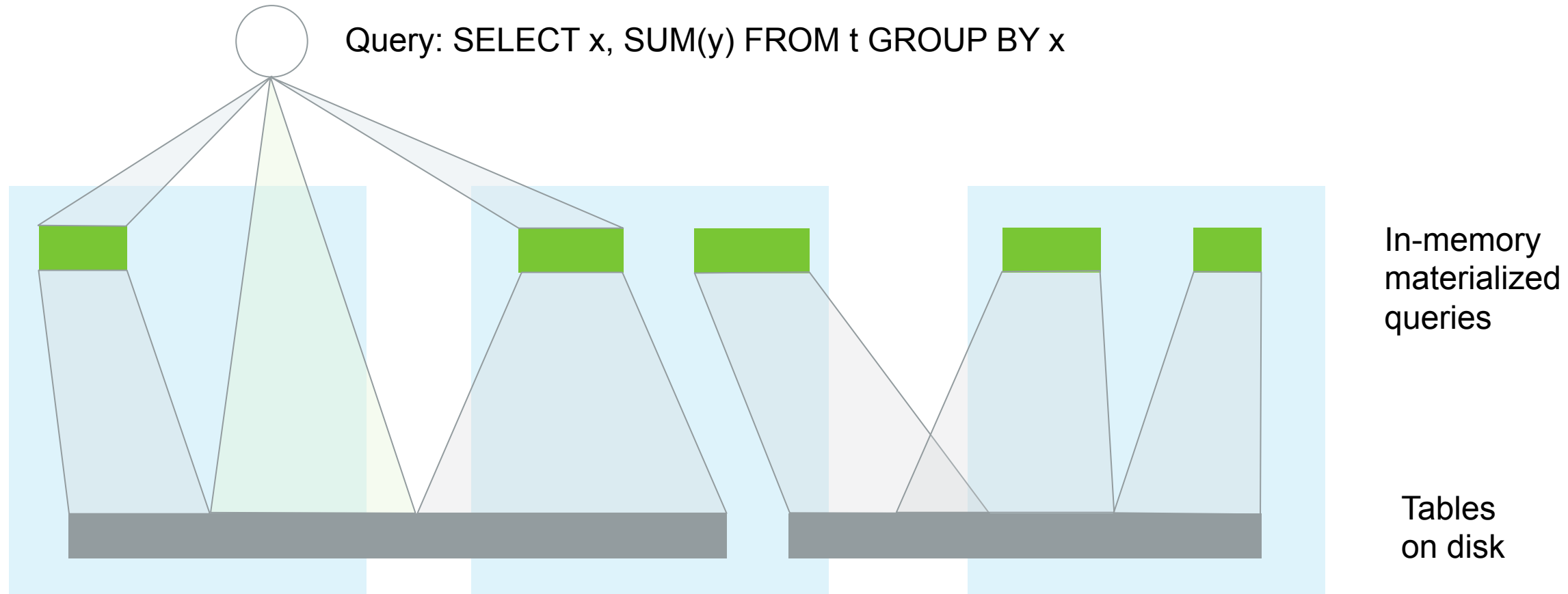
Discard policy

Build in-memory, replicate to disk; or vice versa

Central namespace

## Evolution of existing components

# Tiled queries in distributed memory



# An adaptive system

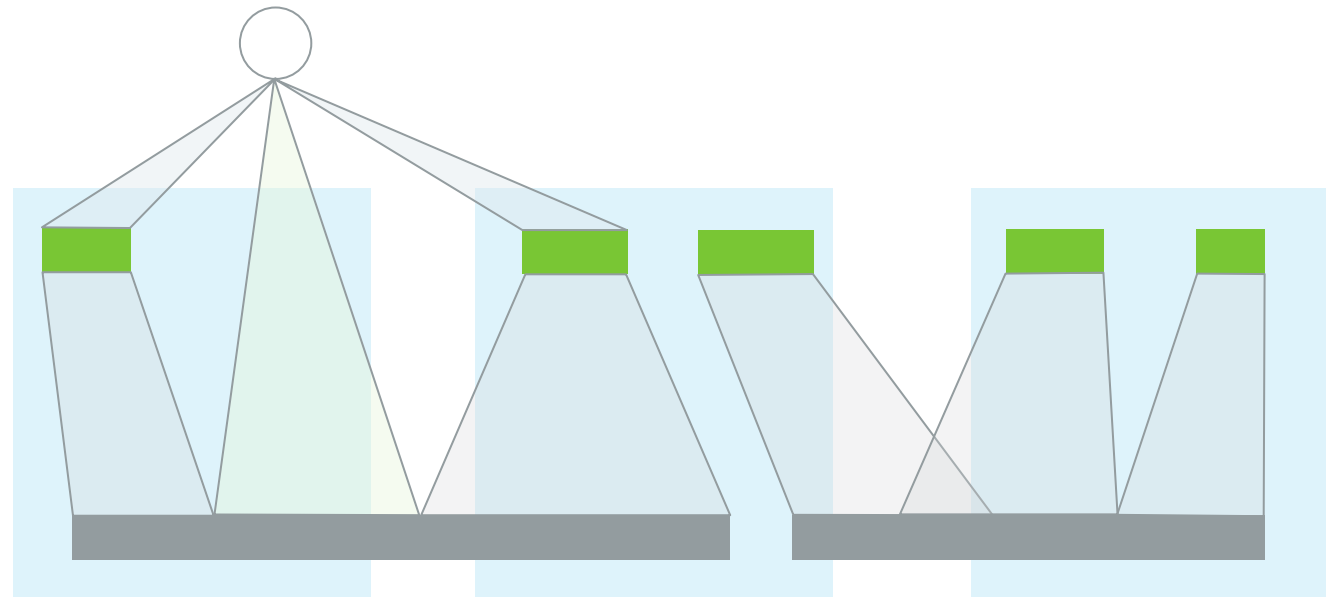
## Ongoing activities:

- Agent suggests new MVs
- MVs are built in background
- Ongoing query activity uses MVs
- User marks MVs as invalid due to source data changes
- HDFS throws out MVs that are not pulling their weight

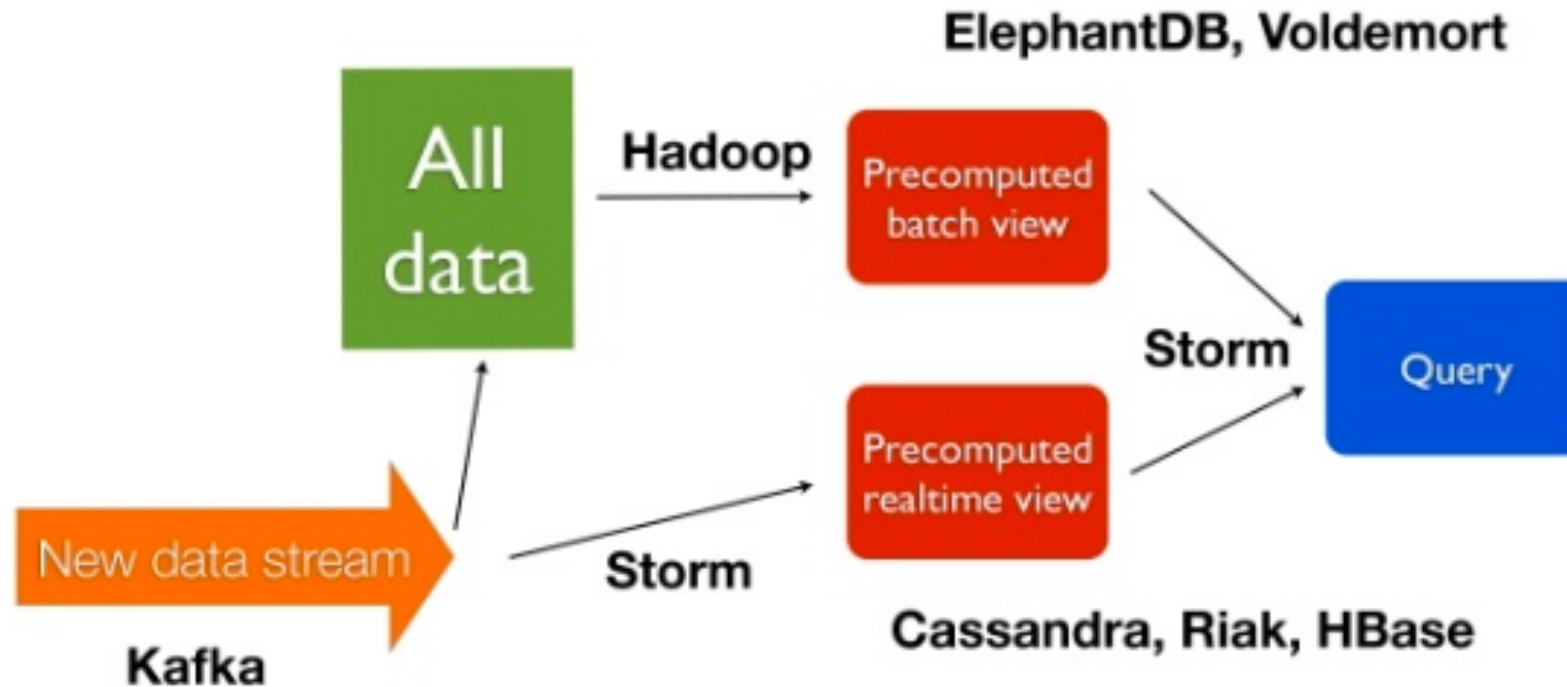
## Dynamic equilibrium

DIMMQs continually created & destroyed

System moves data around to adapt to changing usage patterns

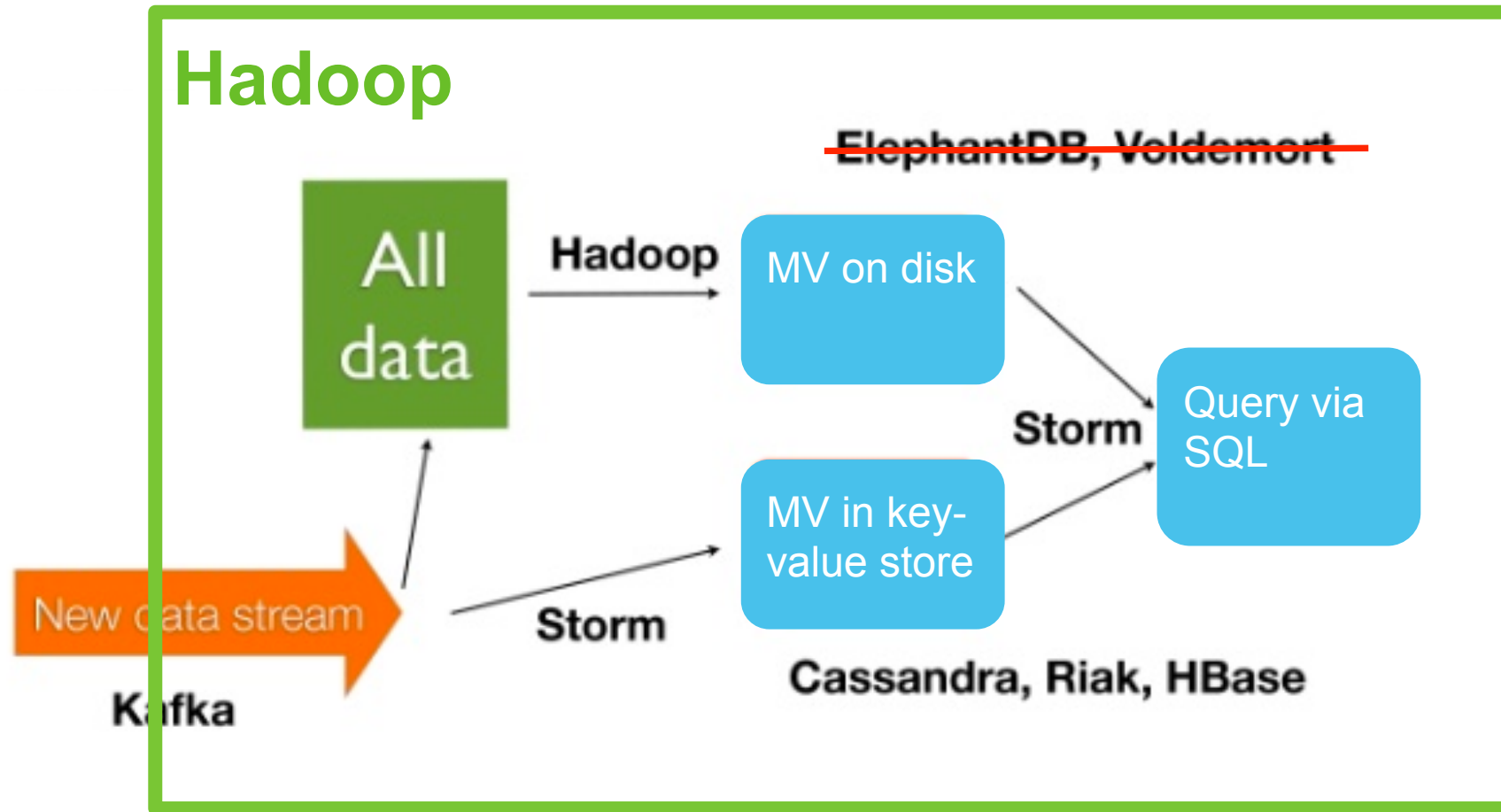


# Lambda architecture



From “Runaway complexity in Big Data and a plan to stop it” (Nathan Marz)

# Lambda architecture in Hadoop via DIMMQs



**Use DIMMQs for materialized historic & streaming data**

# Variations on a theme

**Materialized queries don't have to be in memory**

**Materialized queries don't need to be discardable**

**Materialized queries don't need to be accessed via SQL**

**Materialized queries allow novel data structures to be described**

**Maintaining materialized queries - build on Hive ACID**

**Fine-grained invalidation**

**Streaming into DIMMQs**

**In-memory tables don't have to be materialized queries**

**Data aging – Older data to cheaper storage**

Discardable

In-memory

Algebraic

# Lattice

## Space of possible materialized views

A star schema, with mandatory many-to-one relationships

## Each view is a projected, filtered aggregation

- Sales by zipcode and quarter in 2013
- Sales by state in Q1, 2012

## Lattice gathers stats

- “I used MV  $m$  to answer query  $q$  and avoided fetching  $r$  rows”
- Cost of MV = construction effort + memory \* time
- Utility of MV = query processing effort saved

## Recommends & builds optimal MVs

```
CREATE LATTICE SalesStar AS
SELECT *
FROM SalesFact AS s
JOIN TimeDim AS t USING (timeId)
JOIN CustomerDim AS c USING (customerId);
```

# Conclusion

**Broadening the tent – batch, interactive BI, streaming, iterative**

**Declarative, algebraic, transparent**

**Seamless movement between memory and disk**

**Adding brains to complement Hadoop's brawn**

# Thank you!

**My next talk:**  
**“Cost-based query optimization in Hive”**  
**4:35pm Wednesday**

**@julianhyde**

**<http://hortonworks.com/blog/dmmq/>**

**<http://hortonworks.com/blog/ddm/>**

**<http://incubator.apache.org/projects/optiq.html>**

