Università degli Studi di Modena e Reggio Emilia
Dipartimento di Ingegneria "Enzo Ferrari"
Corso di Laurea Magistrale in Ingegneria Informatica

Big Data streaming processing engines under the umbrella of the Apache Foundation: benchmark and industrial applications

Relatore:
Prof. Sonia Bergamaschi

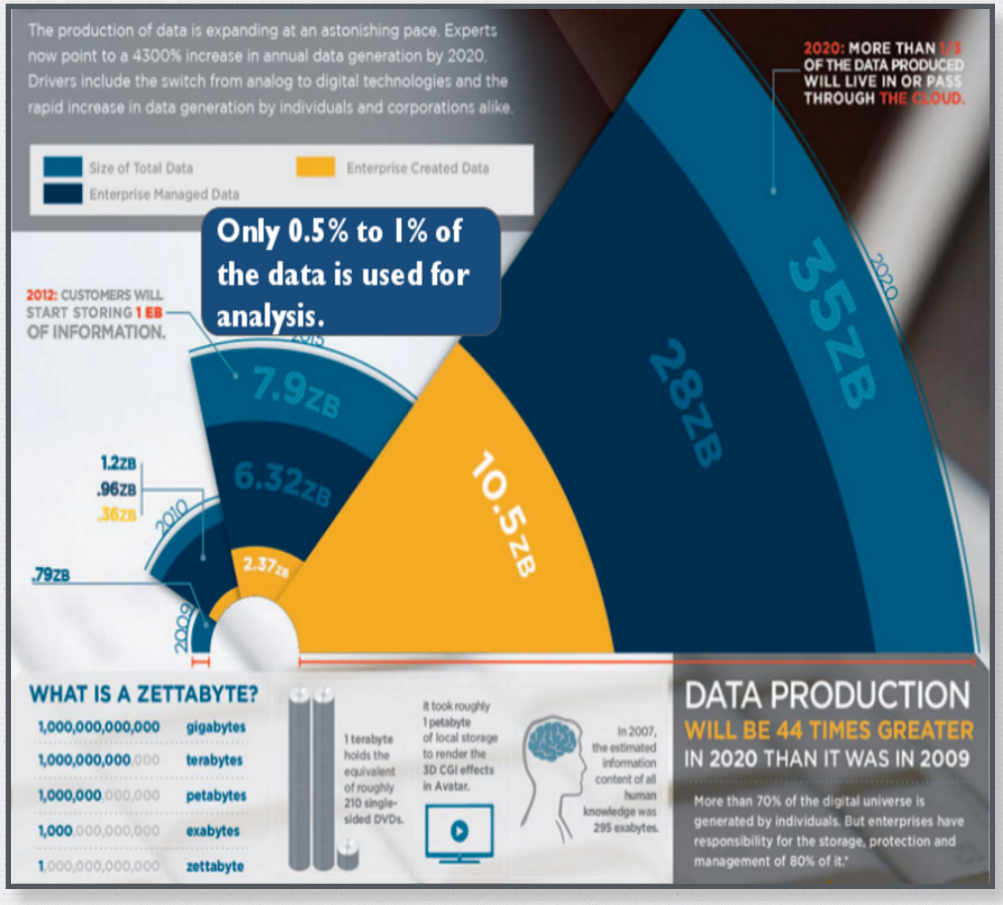
Correlatore:
Dr. Tilmann Rabl

Tesi di laurea di:
Guido Mazza

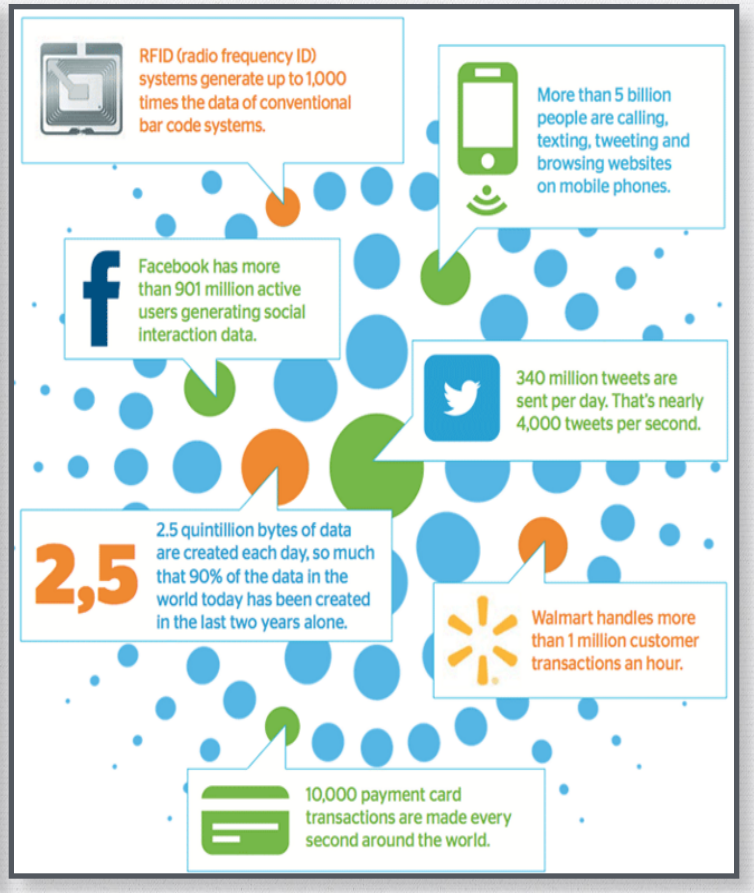
Collaboration with TU Berlin

Work jointly conducted by the **DB Group** at **University of Modena and Reggio Emilia** and the **Technische Universitat in Berlin (TUB)**, precisely at the **Datenbanksysteme u. Informationsmanagement Department (DIMA)**.

Big Data modern challenges

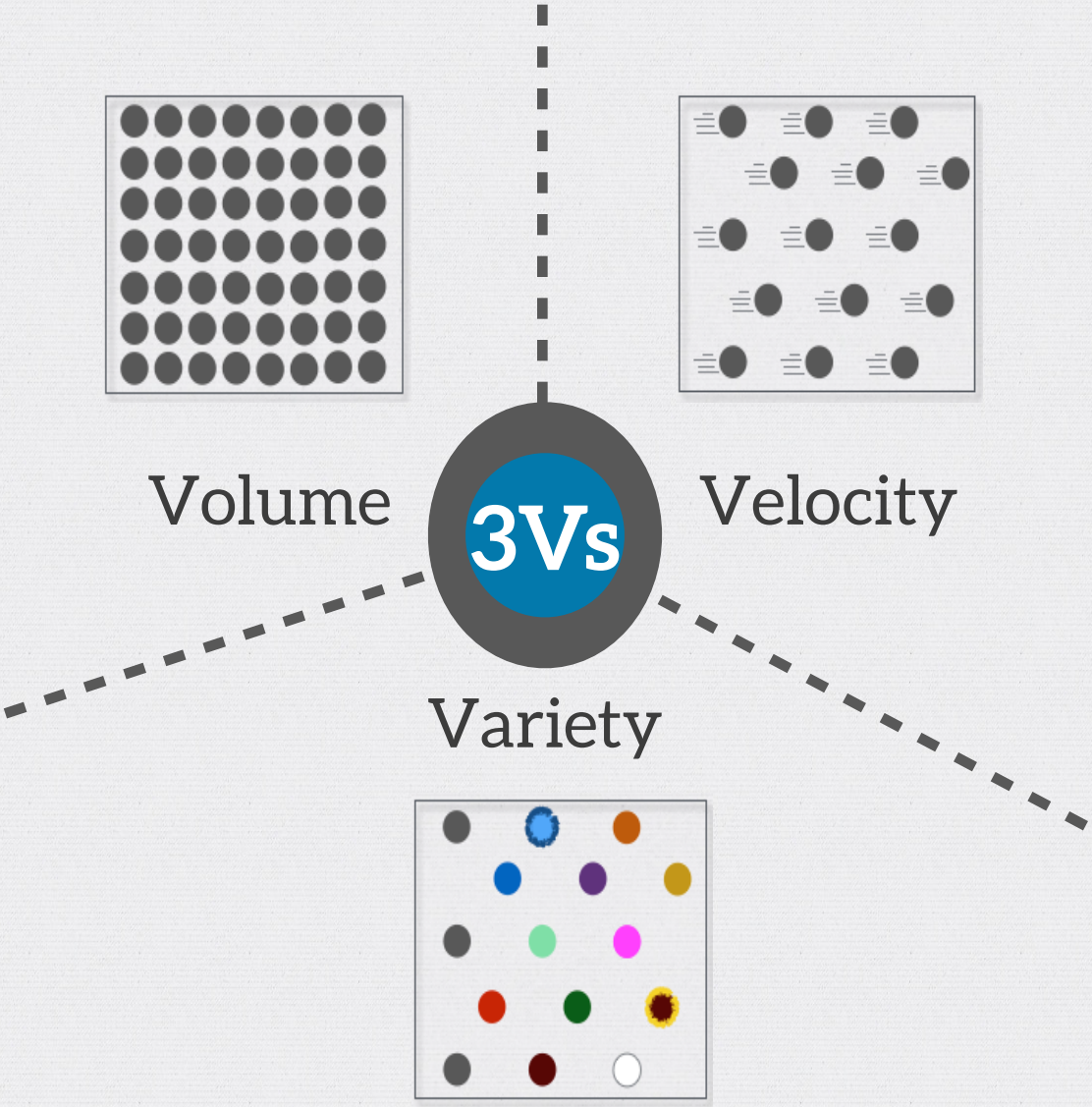


Big Data universe beginning to explode.
http://www.csc.com/insights/flxwd/78931-big_data_growth_just_beginning_to_explode

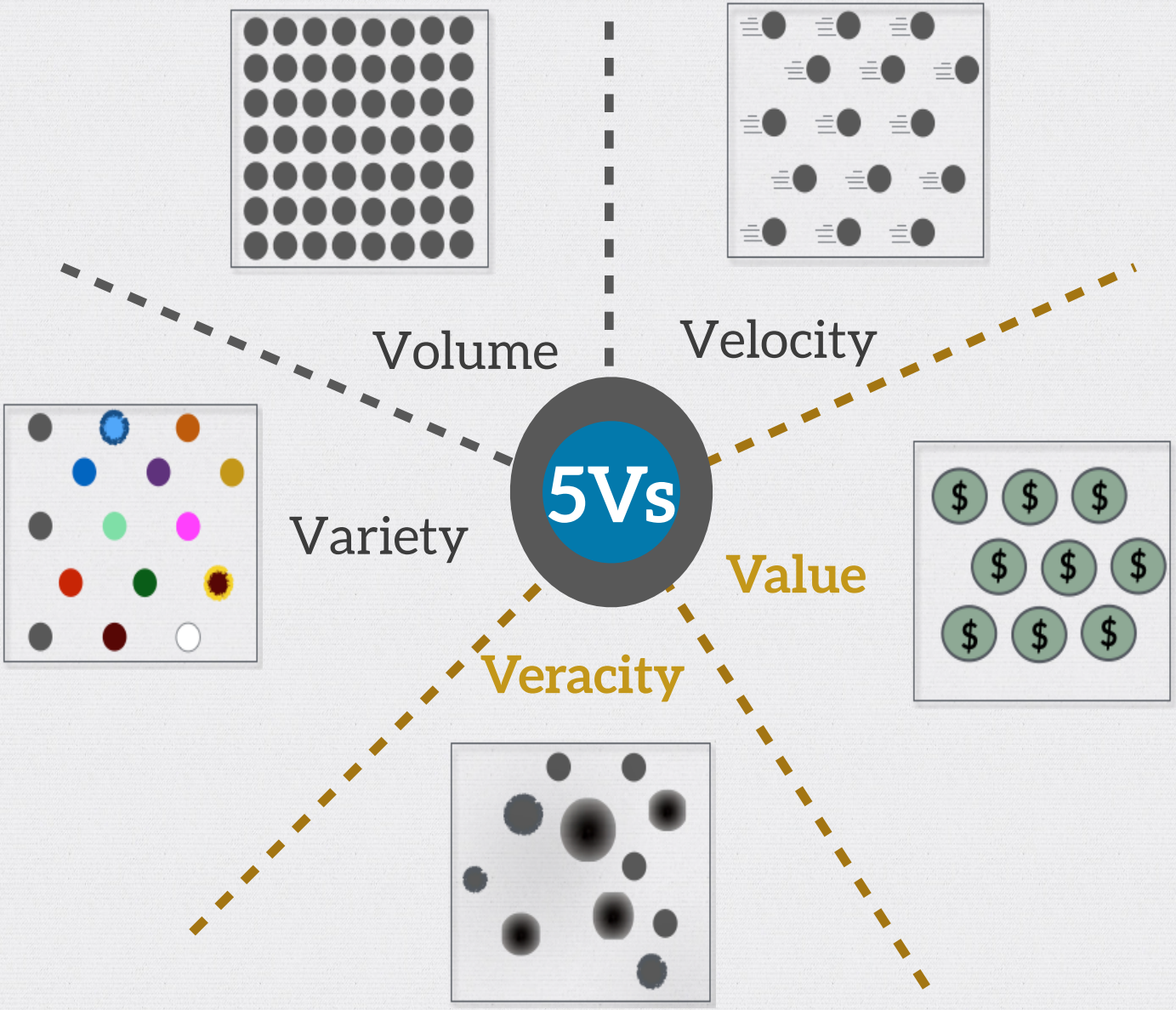


Big data "Where we at?", June 2013.
http://www.centrodeinnovacionbbva.com/en/innovation-edge/big-data/big-data-where-we

Big Data modern challenges

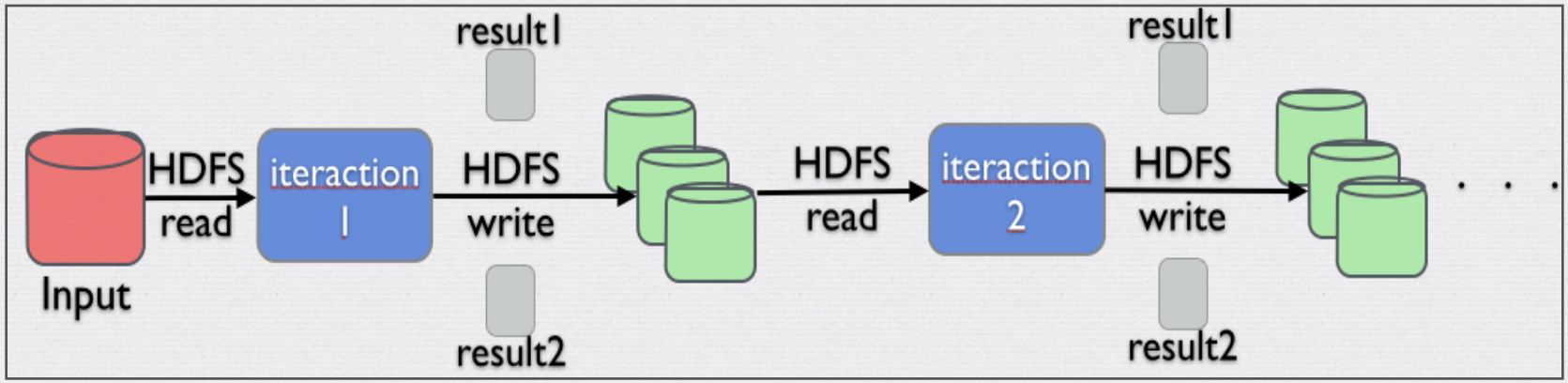


Big Data modern challenges

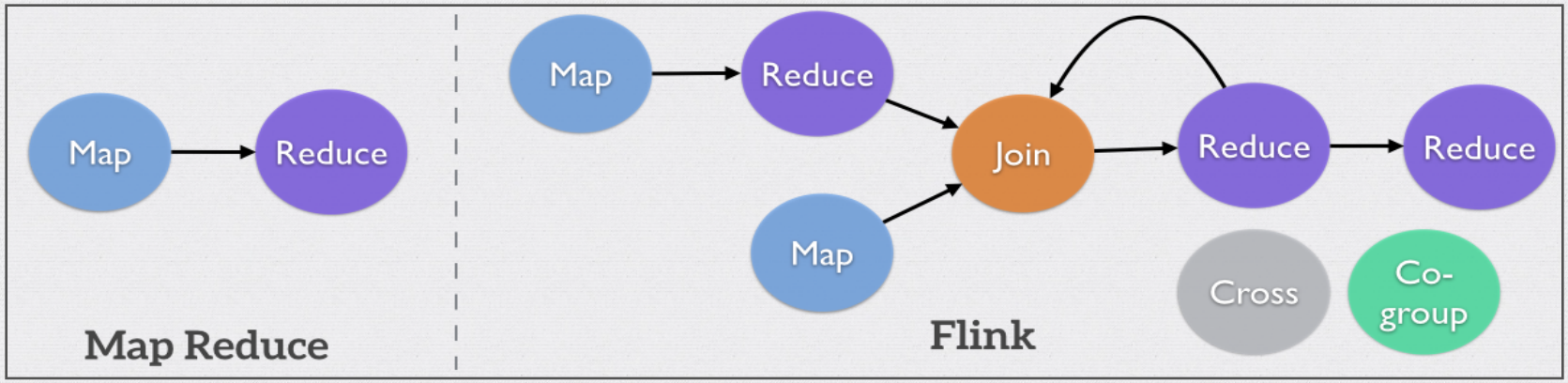


Beyond the MapReduce' shortcomings

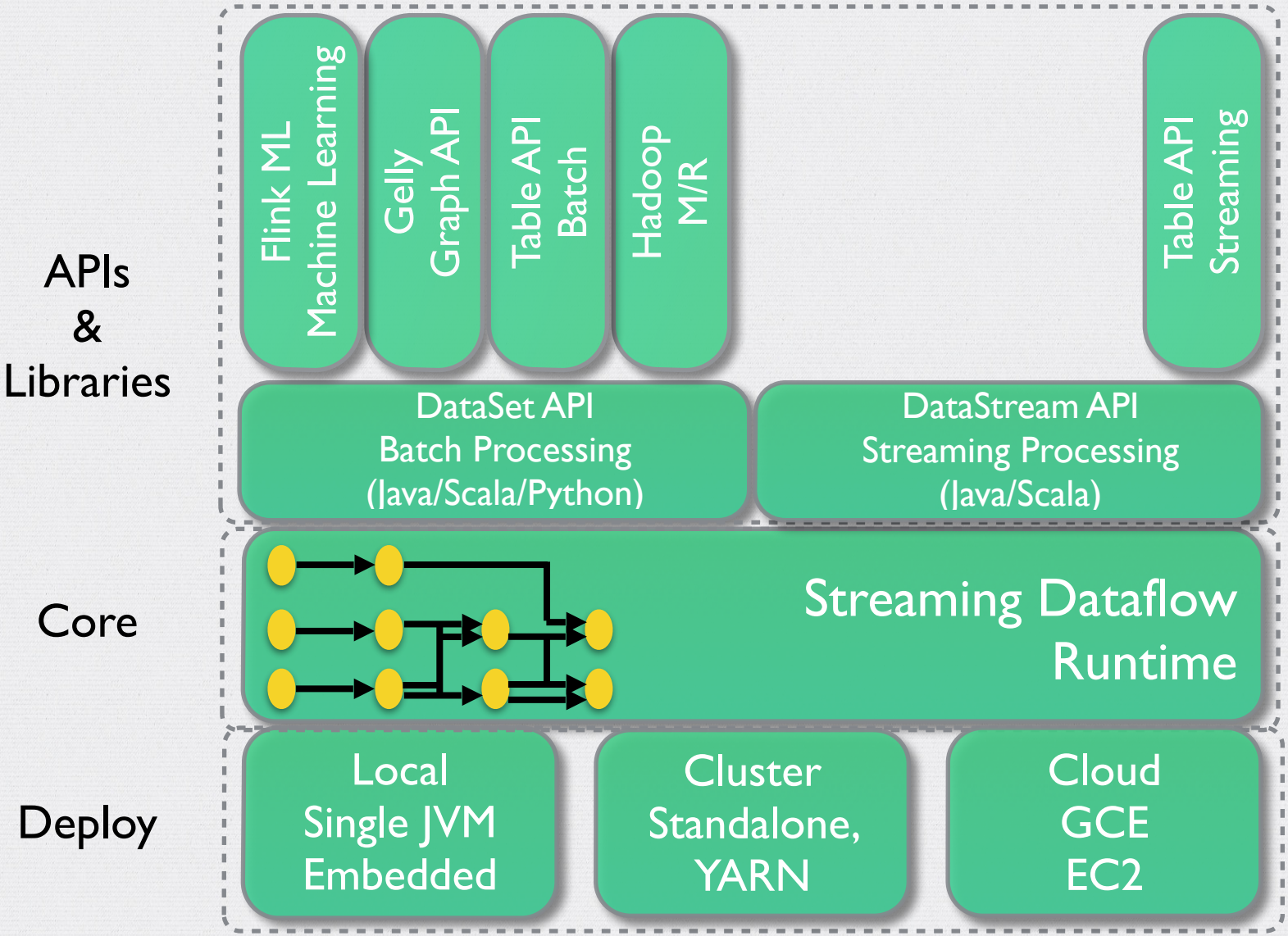
- Memory management



- Inability to express complex logic & low-level programming

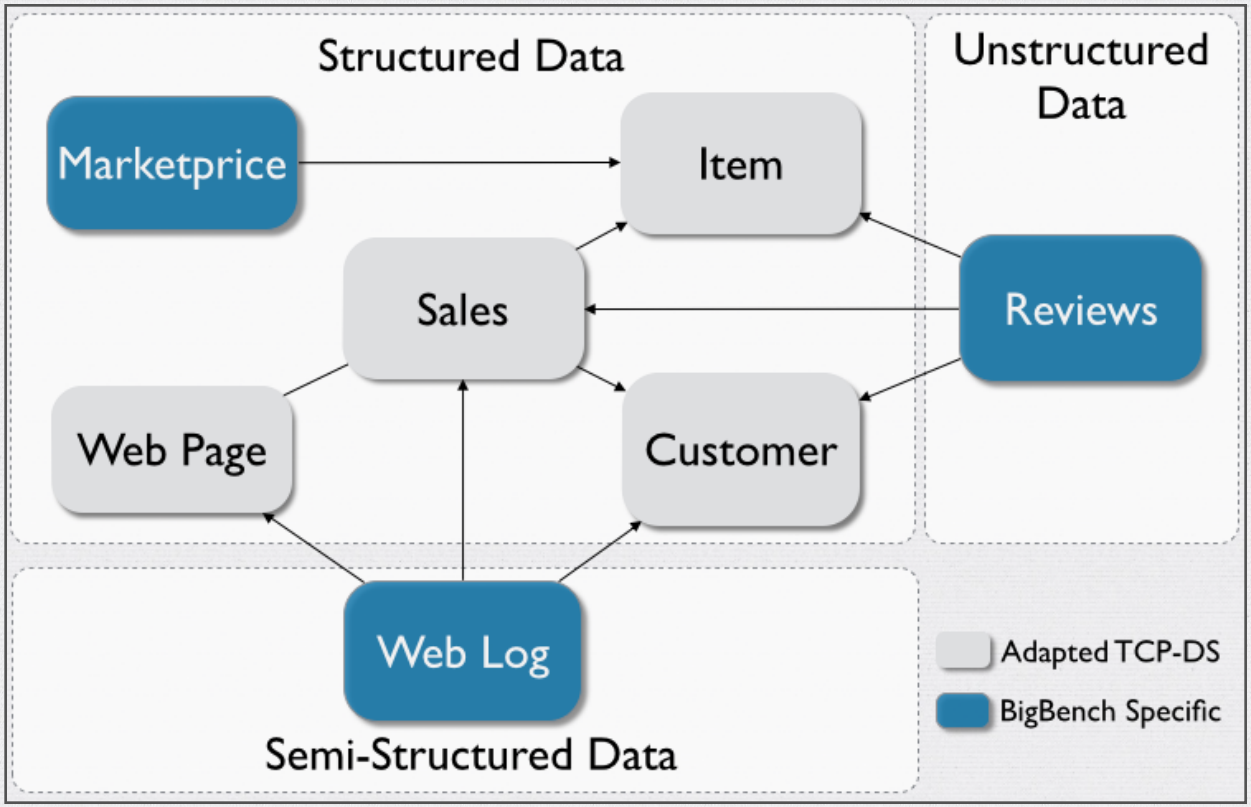


Apache Flink stack



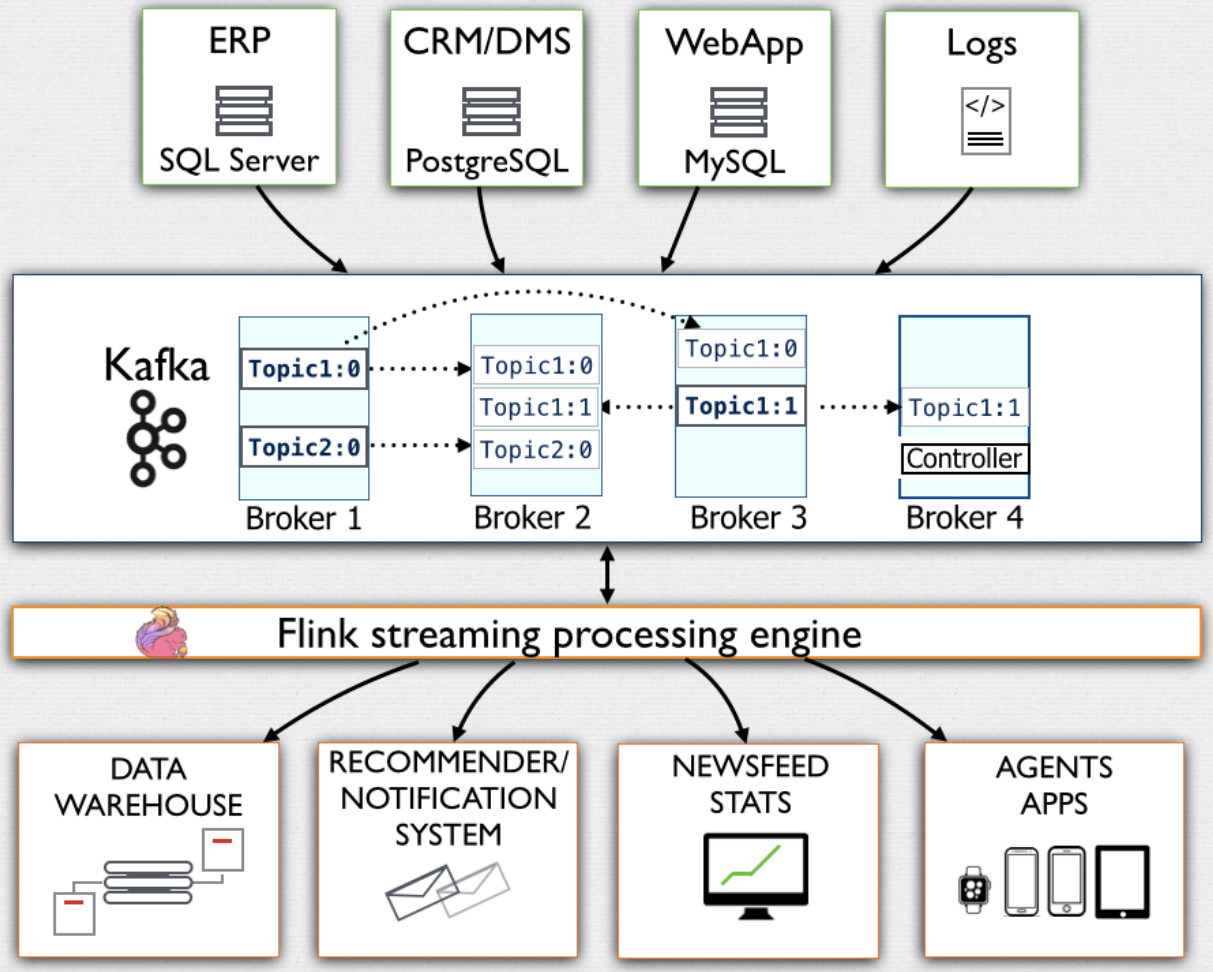
Thesis Road Map - Benchmark

- **Big Bench:** an end-to-end big data benchmark developed by **Oracle**, **TeraData**, **Intel** in collaboration with the Middleware Systems Research Group, University of Toronto, Canada
  - **Big Bench Data Model**

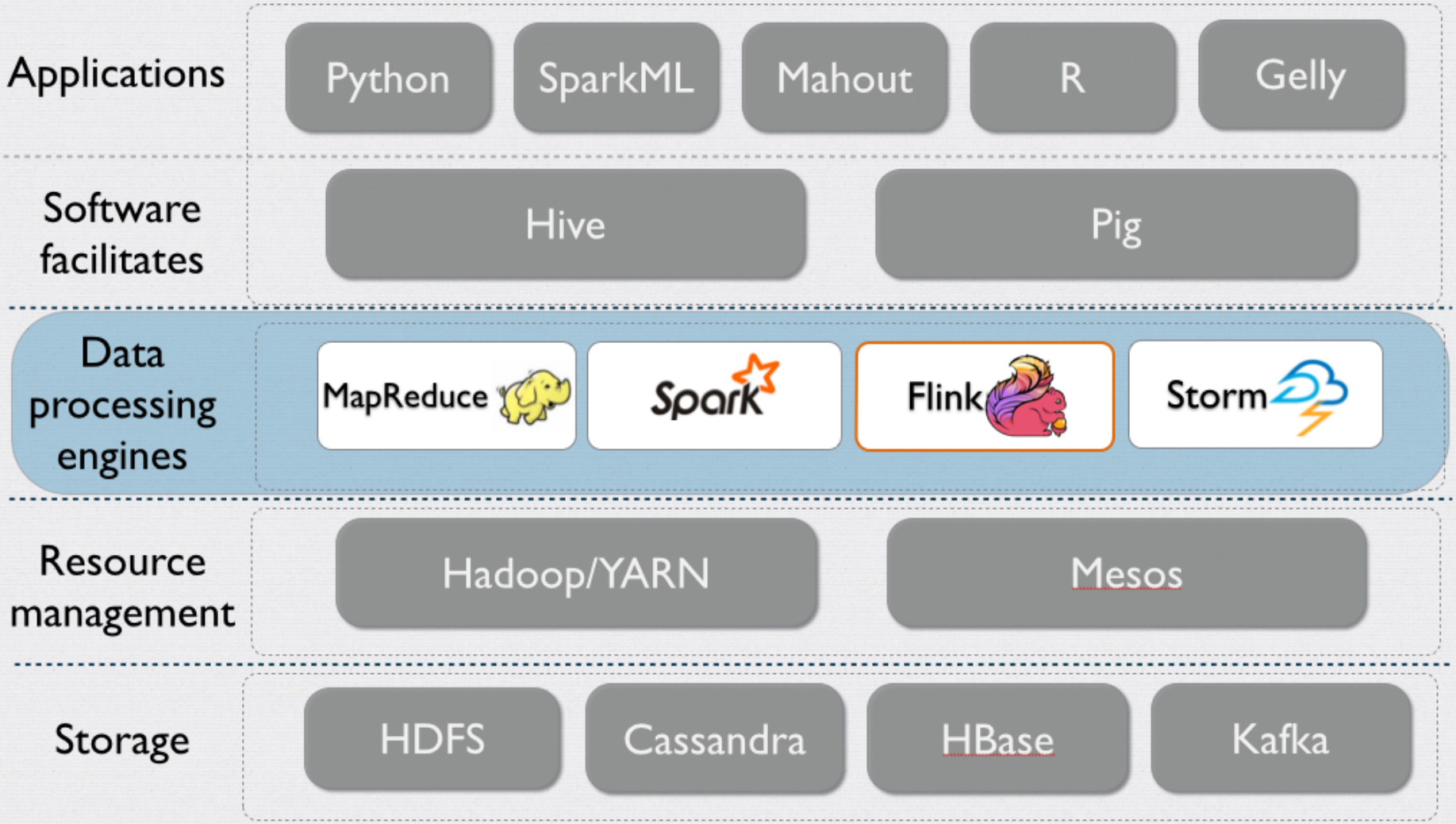


Thesis Road Map - Streaming

- **Data integration & Streaming processing**: beyond the traditional batch processing model.



An open source data management infrastructure



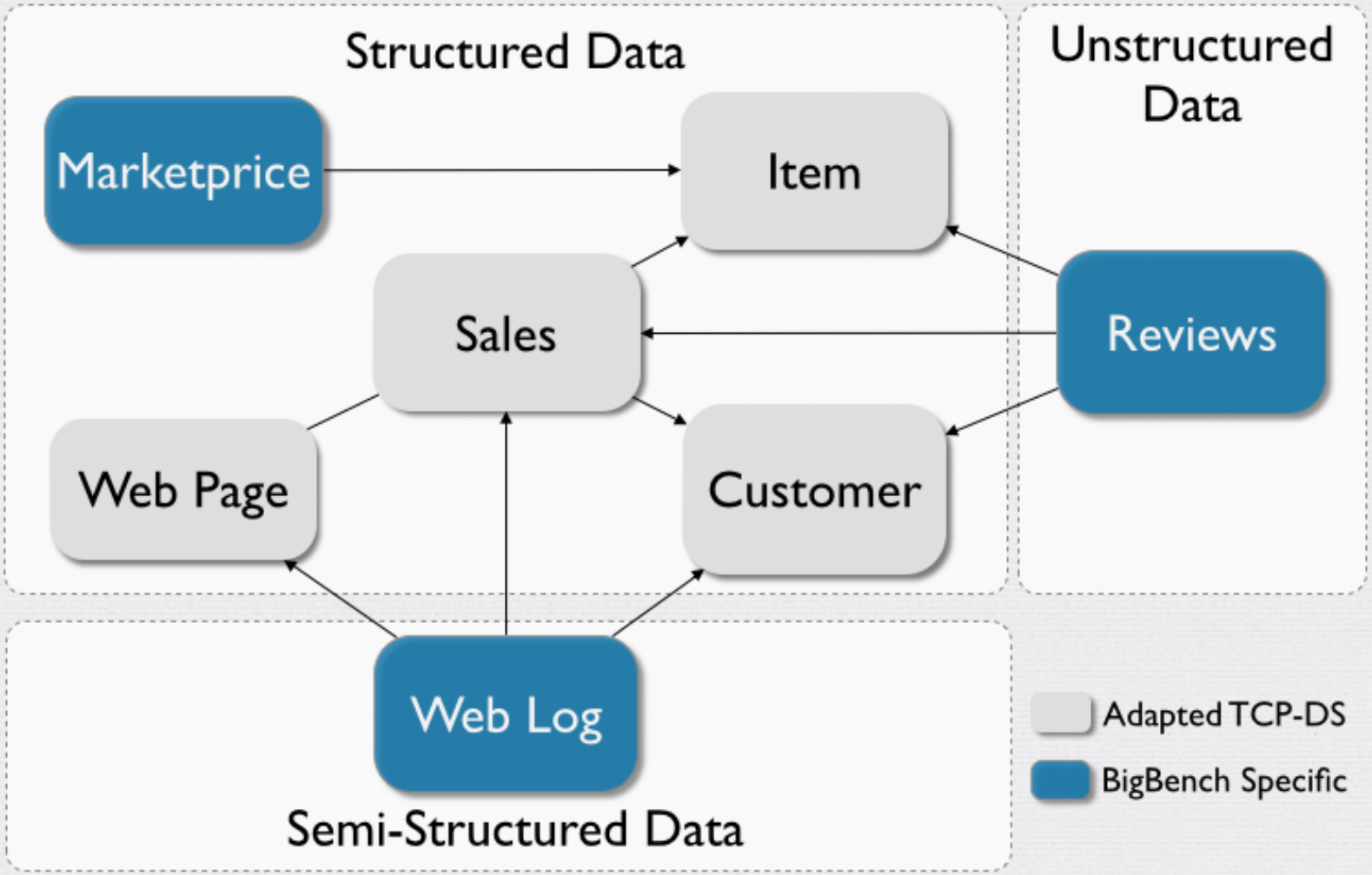
Flink Forward 2015

"A Year in Review" MeetUp, Berlin 16/12/2015.
References: https://flink.apache.org/news/2015/12/18/a-year-in-review.html

Big Bench

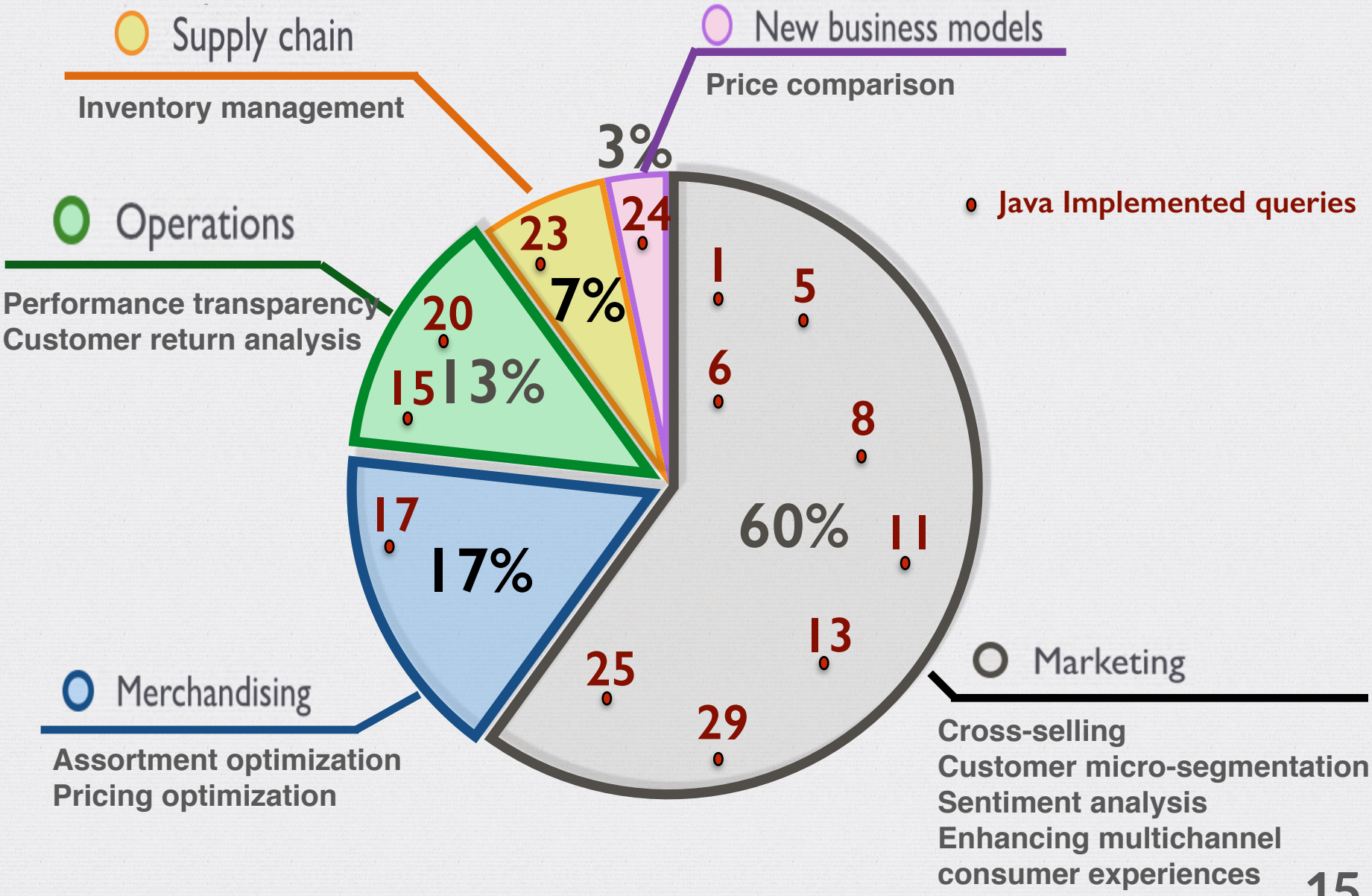
github.com/intel-hadoop/Big-Data-Benchmark-for-Big-Bench

Big Bench Data Model



Parallel Data Generation Function (PDGF)

Big Bench Workload



Workload

- Market Basket Analysis
- Tracking customers' behavior
- Map and Python Reduce Function
- Pearson correlation coefficient of sentiments in reviews on monthly revenues
- Customers K-Means clustering
- Data model generation for Logistic Regression analysis in classification
- Detecting flat or declining sale trends
- Finding the ratio of sold items according to proposed promotions
- Customer segmentation for return analysis
- Items' coefficient of variation analysis

Github Repository: github.com/guidom300/MasterThesisFink

Test

Single query running time
Aggregate time analysis
Time savings percentage
Detailed use case

Cluster environment

IBM DIMA Departmental Cluster - 8 nodes configuration

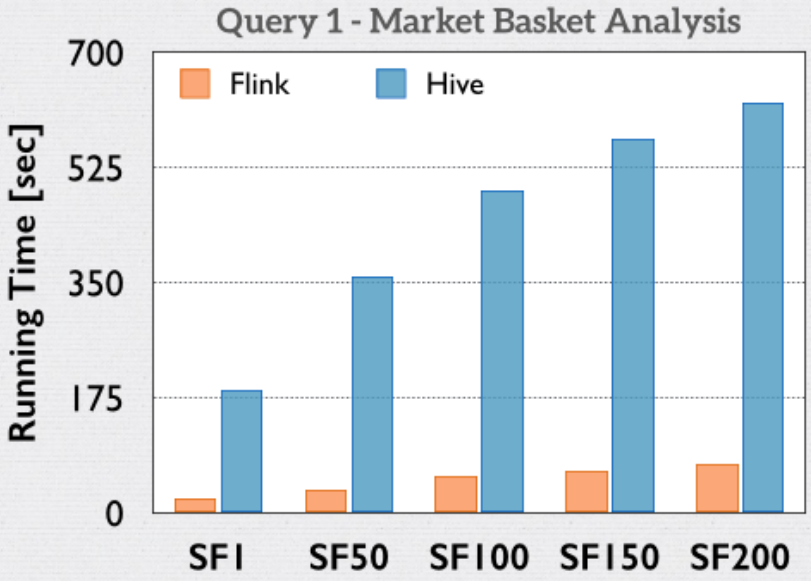
Single node remarkable specs

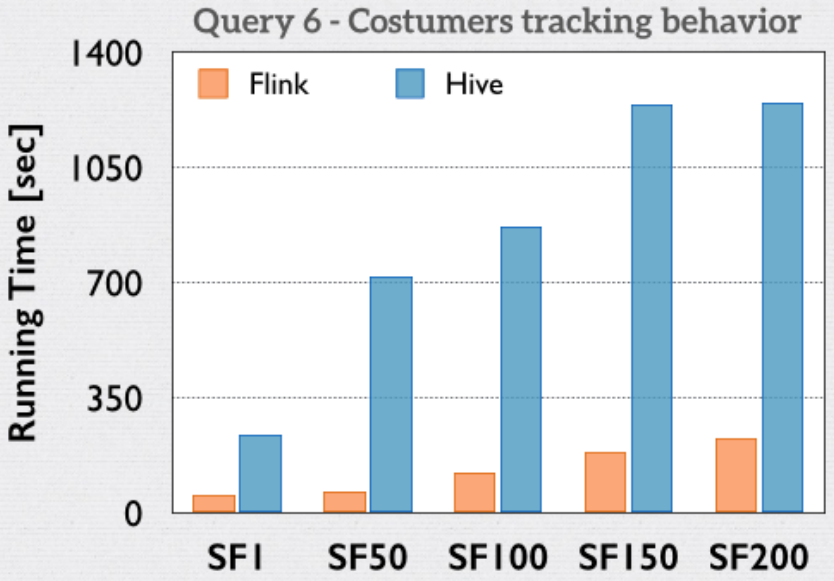
- Architecture: PPC64
- CPU(s): 48
- Single processor clock: 3.7 GHz
- Thread(s) per core: 4
- Socket(s): 12
- Core(s) per socket: 1
- NUMA node(s): 2 (0-23, 24-47)
- L1d cache: 32 KB
- L1i cache: 32 KB
- L2 cache: 256 KB
- L3 cache: 4096 KB
- Total memory: 50 GB

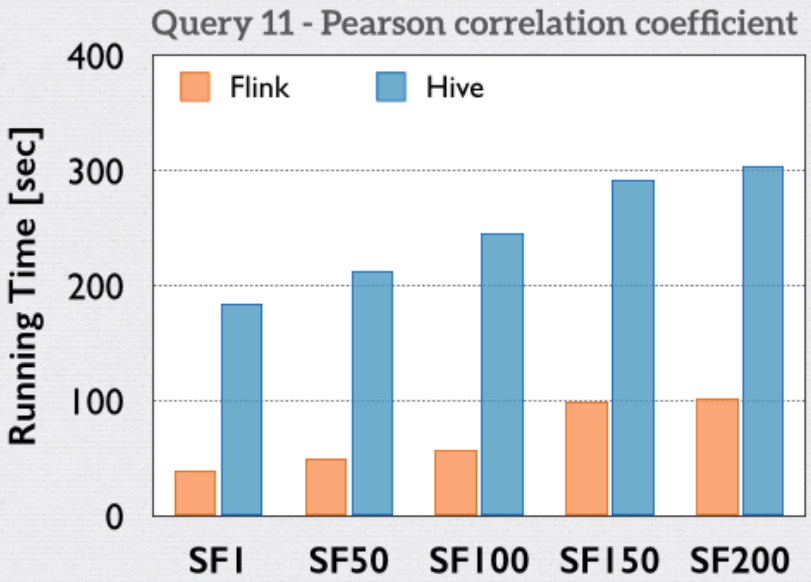
System requirements

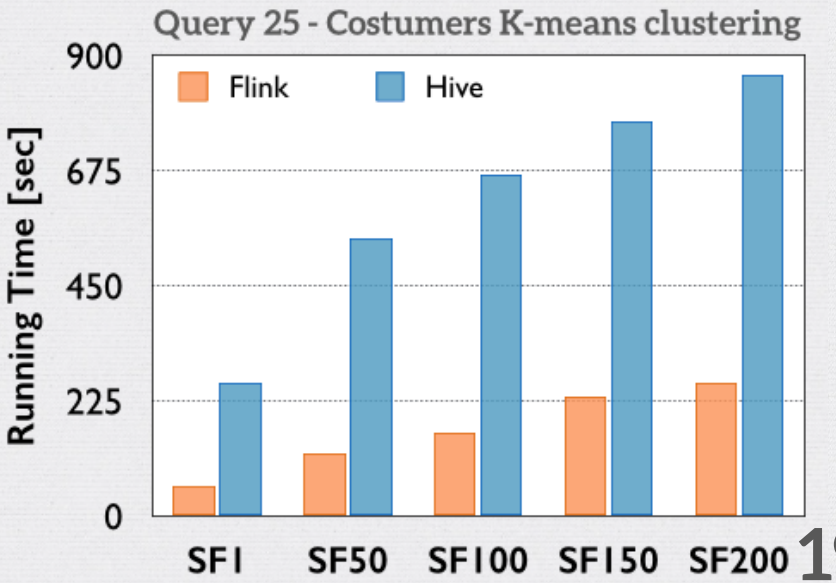
- Apache Flink 0.10 (October 2015)
- Apache Hive 1.2.1
- Apache Spark 1.6.0 (spark-ml package)
- Apache Hadoop 2.4.1

Single query running time

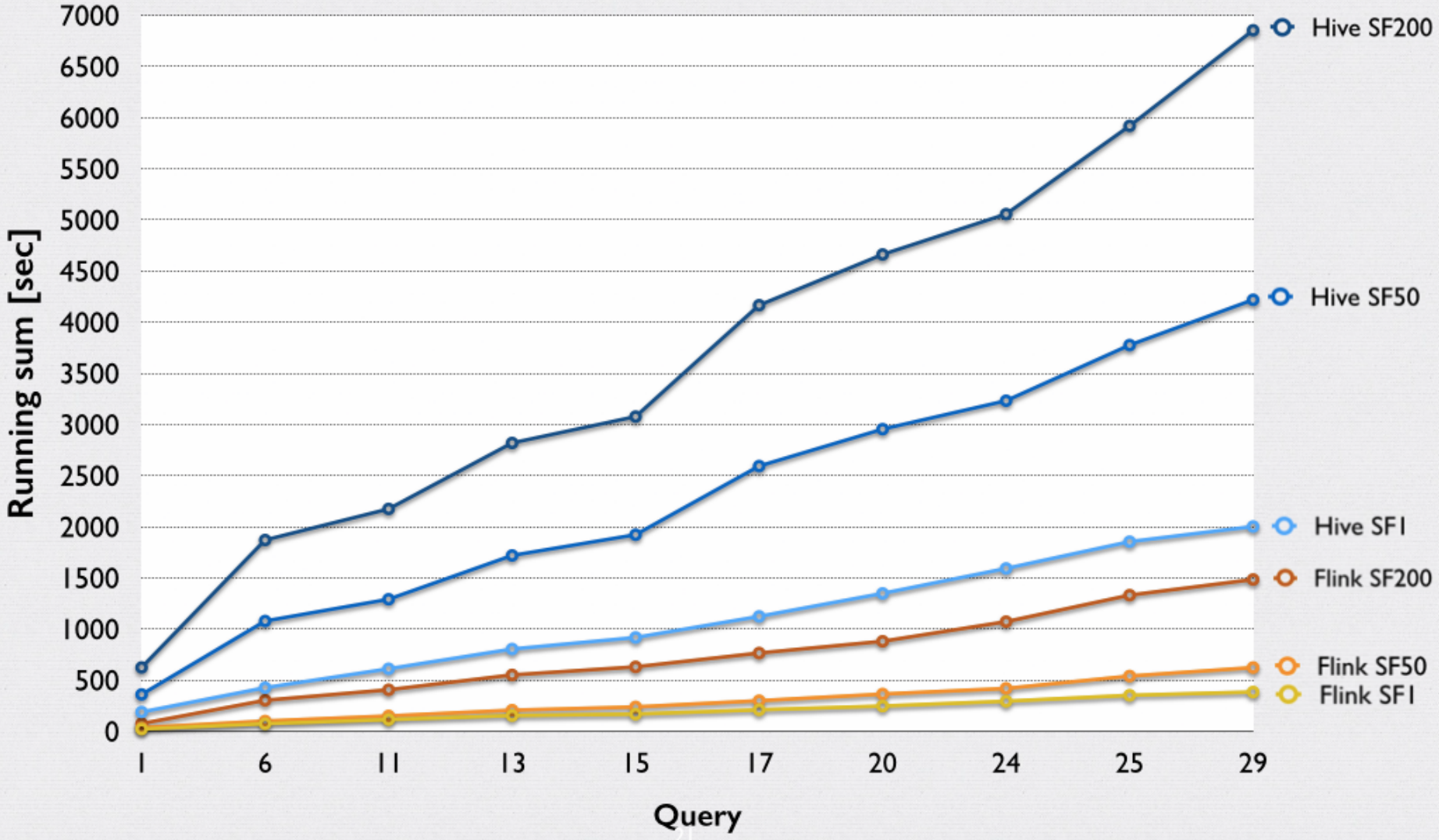








Total running time



Time savings percentage

Query	SF 1	SF 50	SF 100	SF 150	SF 200
1	88.3%	90.0%	88.8%	88.7%	88.0%
6	78.2%	91.1%	86.0%	85.2%	81.7%
11	78.4%	76.5%	76.8%	65.8%	66.1%
13	79.3%	86.9%	84.0%	78.4%	77.5%
15	87.6%	84.2%	77.6%	68.5%	69.4%
17	79.6%	90.9%	89.4%	90.1%	87.8%
20	83.6%	82.0%	81.1%	80.4%	76.8%
24	81.1%	80.2%	71.6%	73.2%	61.4%
25	77.5%	77.8%	75.7%	69.8%	70.0%
29	78.4%	89.8%	89.4%	87.3%	83.6%
Total	**80.0%**	**86.1%**	**83.7%**	**81.4%**	**78.3%**

Query 6 - Hive code

Identifies customers shifting their purchase habit from store to web sales

```
-- customer store sales
CREATE VIEW ${hiveconf:TEMP_TABLE1} AS
SELECT ss_customer_sk AS customer_sk,
       sum( case when (d_year = ${hiveconf:q06_YEAR})
       THEN
            (((ss_ext_list_price-ss_ext_wholesale_cost-ss_ext_discount_amt)+ss_ext_sales_price)/2)
       ELSE
            0
       END
       ) first_year_total,
       sum( case when (d_year = ${hiveconf:q06_YEAR}+1)
       THEN
            (((ss_ext_list_price-ss_ext_wholesale_cost-ss_ext_discount_amt)+ss_ext_sales_price)/2)
       ELSE
            0
       END
       ) second_year_total
FROM  store_sales
     ,date_dim
WHERE ss_sold_date_sk = d_date_sk
AND   d_year BETWEEN ${hiveconf:q06_YEAR} AND ${hiveconf:q06_YEAR} +1
GROUP BY ss_customer_sk
HAVING first_year_total > 0  -- required to avoid division by 0, because later we will divide by this value
```

Query 6 - Hive code

Identifies customers shifting their purchase habit from store to web sales

```
-- customer web sales
CREATE VIEW ${hiveconf:TEMP_TABLE2} AS
SELECT ws_bill_customer_sk AS customer_sk,
       sum( case when (d_year = ${hiveconf:q06_YEAR})
       THEN
            (((ws_ext_list_price-ws_ext_wholesale_cost-ws_ext_discount_amt)+ws_ext_sales_price)/2)
       ELSE
            0
       END
       ) first_year_total,
       sum( case when (d_year = ${hiveconf:q06_YEAR}+1)
       THEN
            (((ws_ext_list_price-ws_ext_wholesale_cost-ws_ext_discount_amt)+ws_ext_sales_price)/2)
       ELSE
            0
       END
       ) second_year_total
FROM  store_sales
     ,date_dim
WHERE ws_sold_date_sk = d_date_sk
AND   d_year BETWEEN ${hiveconf:q06_YEAR} AND ${hiveconf:q06_YEAR} +1
GROUP BY ws_bill_customer_sk
HAVING first_year_total > 0  -- required to avoid division by 0, because later we will divide by this value
```

Query 6 - Hive code

Identifies customers shifting their purchase habit from store to web sales

```
-- the real query part
INSERT INTO TABLE ${hiveconf:RESULT_TABLE}
SELECT
      (web.second_year_total / web.first_year_total) AS web_sales_increase_ratio,
      c_customer_sk,
      c_first_name,
      c_last_name,
      c_preferred_cust_flag,
      c_birth_country,
      c_login,
      c_email_address
FROM ${hiveconf:TEMP_TABLE1} store,
     ${hiveconf:TEMP_TABLE2} web,
     customer c
WHERE store.customer_sk = web.customer_sk
AND   web.customer_sk = c_customer_sk
AND   (web.second_year_total / web.first_year_total)  >  (store.second_year_total / store.first_year_total)
ORDER BY
  web_sales_increase_ratio DESC,
  c_customer_sk,
  c_first_name,
  c_last_name,
  c_preferred_cust_flag,
  c_birth_country,
  c_login
LIMIT ${hiveconf:q06_LIMIT};
```

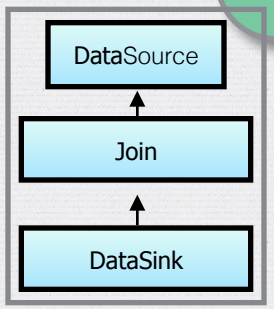
Query 6 - Flink code & execution

```
DataSet<Sales> store_sales = getStoreSalesDataSet(env);
DataSet<DateDim> date_dim = getDateDimDataSet(env);
DataSet<Sales> web_sales = getWebSalesDataSet(env);
DataSet<Customer> customers = getCustomersDataSet(env);

//customer_sk, first_year_total, first_year_total
DataSet<Tuple3<Long, Double, Double>> customer_store_sales =
        store_sales
                .join(date_dim, BROADCAST_HASH_SECOND).with(new JoinHelper())
                .groupBy(0)
                .reduceGroup(new getCustomerSales());

//customer_sk, first_year_total, first_year_total
DataSet<Tuple3<Long, Double, Double>> customer_web_sales =
        web_sales
                .join(date_dim, BROADCAST_HASH_SECOND).with(new JoinHelper_2())
                .groupBy(0)
                .reduceGroup(new getWebCustomerSales());

//web_sales_increase_ratio, c_customer_sk, c_first_name, c_last_name, c_preferred_cust_flag,
c_birth_country,c_login, c_email_address
DataSet<Tuple8<Double, Long, String, String, String, String, String, String>> results =
        customer_store_sales
                .join(customer_web_sales, BROADCAST_HASH_FIRST).with(new SSJoinWS())
                .filter(new FilterTotals())
                .join(customers, BROADCAST_HASH_FIRST).with(new WebStoreJoinCustomer())
                .sortPartition(on [0:DESC,1:ASC,2:ASC,3:ASC,4:ASC,5:ASC,6:ASC,7:ASC]).setParallelism(1)
                .first(100);

results.writeAsCsv(hdfs_output_path).setParallelism(1);
```

Query 6 - Flink code & execution

Flink program

Job Graph



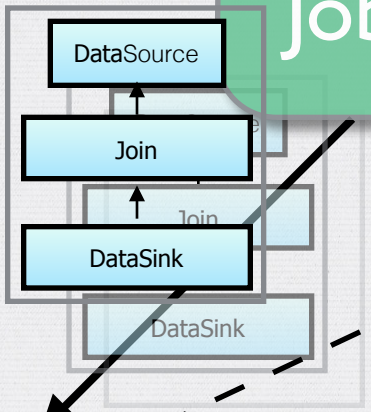
Client

Type extraction

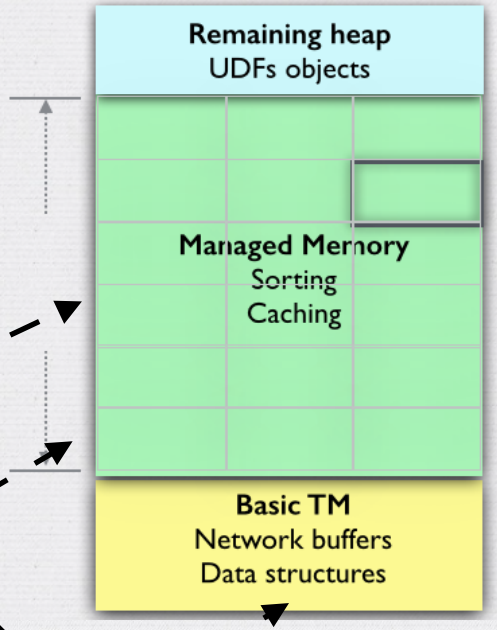
Optimizer

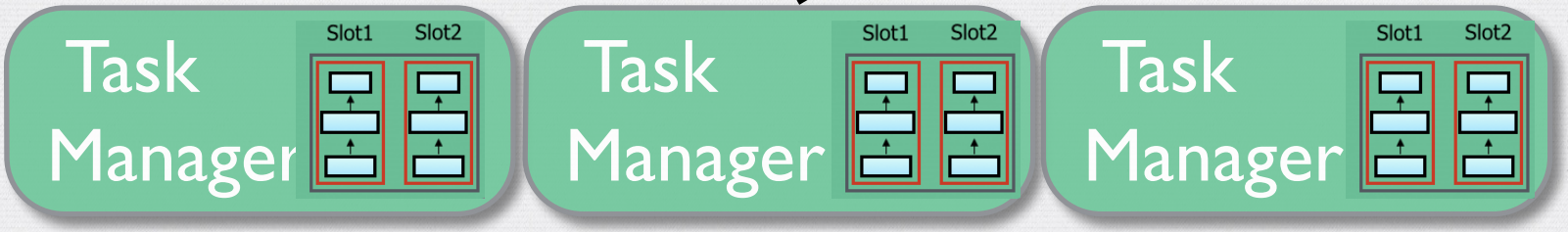
Job Manager

Execution Graph

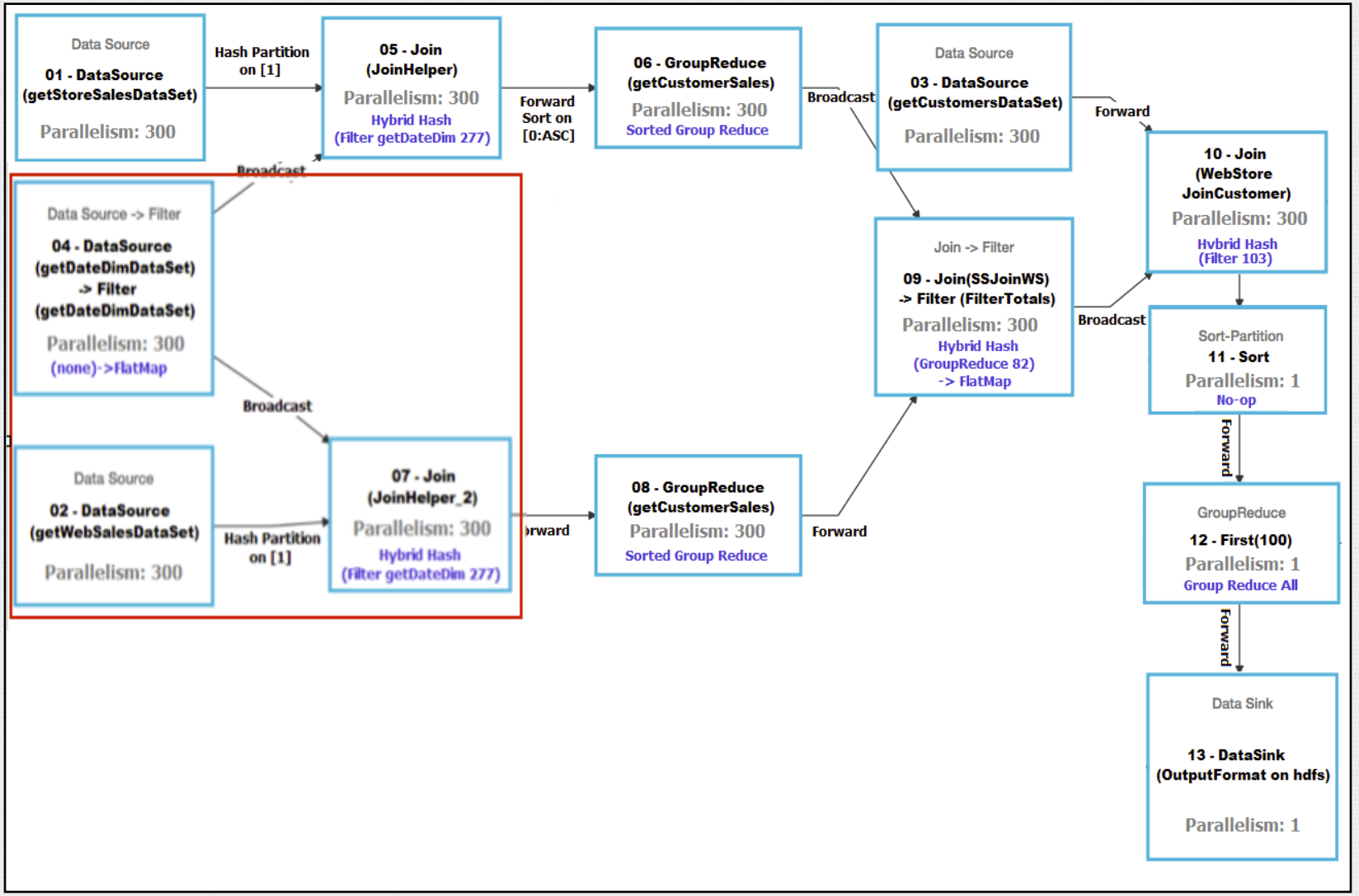


Task Manager JVM heap

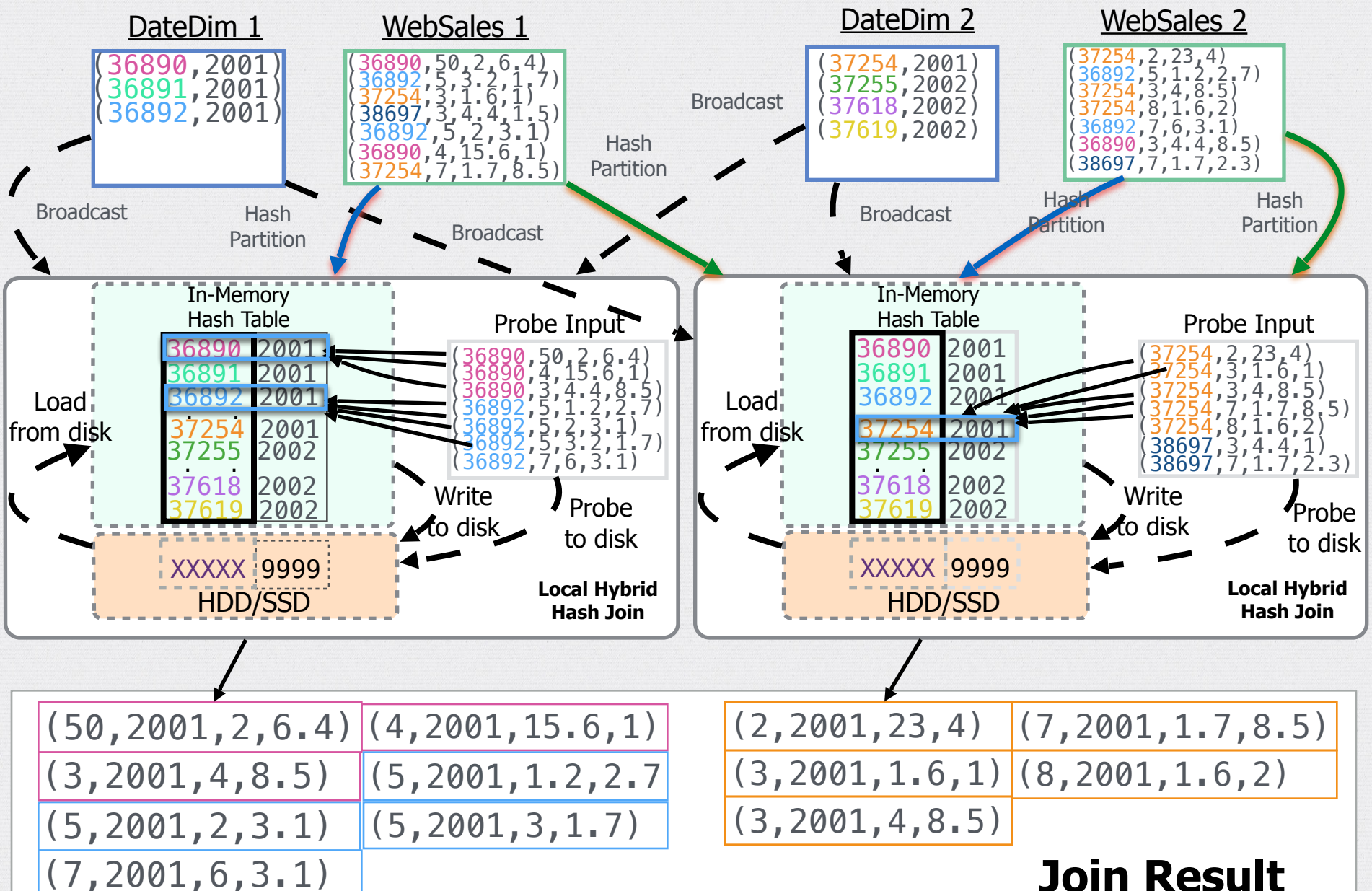




Query 6 - Execution plan



Query 6 - Hybrid Hash Join



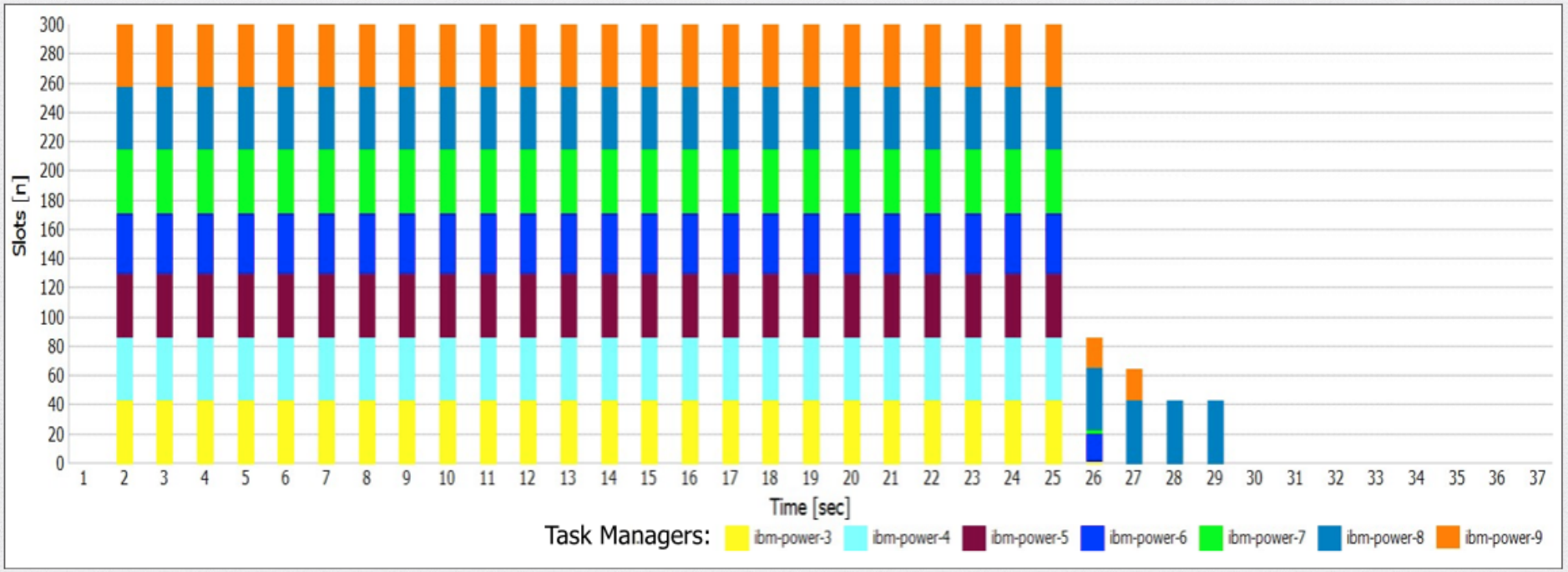
(50,2001,2,6.4)	(4,2001,15.6,1)
(3,2001,4,8.5)	(5,2001,1.2,2.7
(5,2001,2,3.1)	(5,2001,3,1.7)
(7,2001,6,3.1)	

(2,2001,23,4)	(7,2001,1.7,8.5)
(3,2001,1.6,1)	(8,2001,1.6,2)
(3,2001,4,8.5)	

Join Result

Query 6 - Join parallelism

07- Join (JoinHelper_2) - Task Managers' occupation

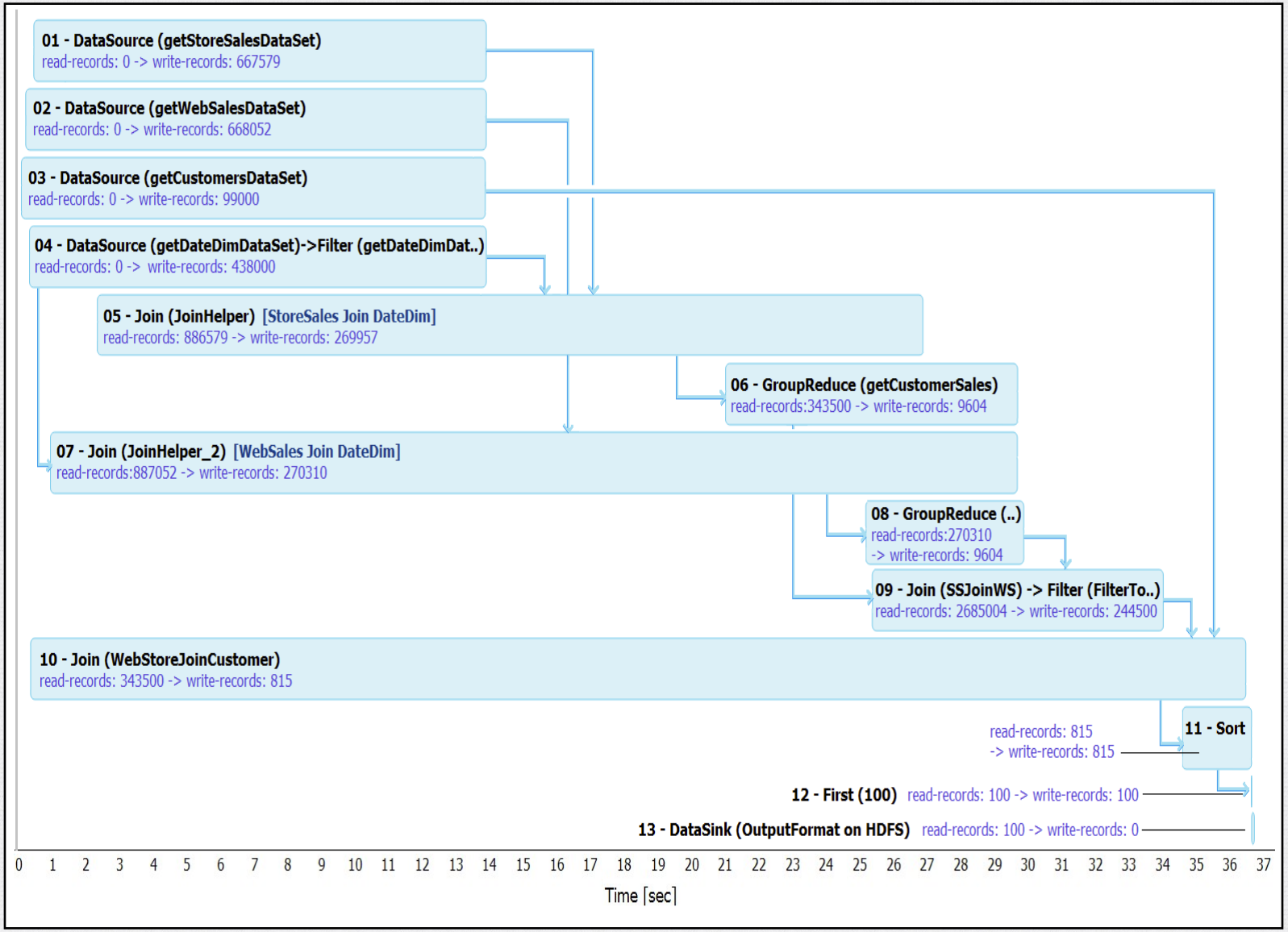


slots = nTaskManagers • slotsOnTaskManager = 7 • 48 = 336

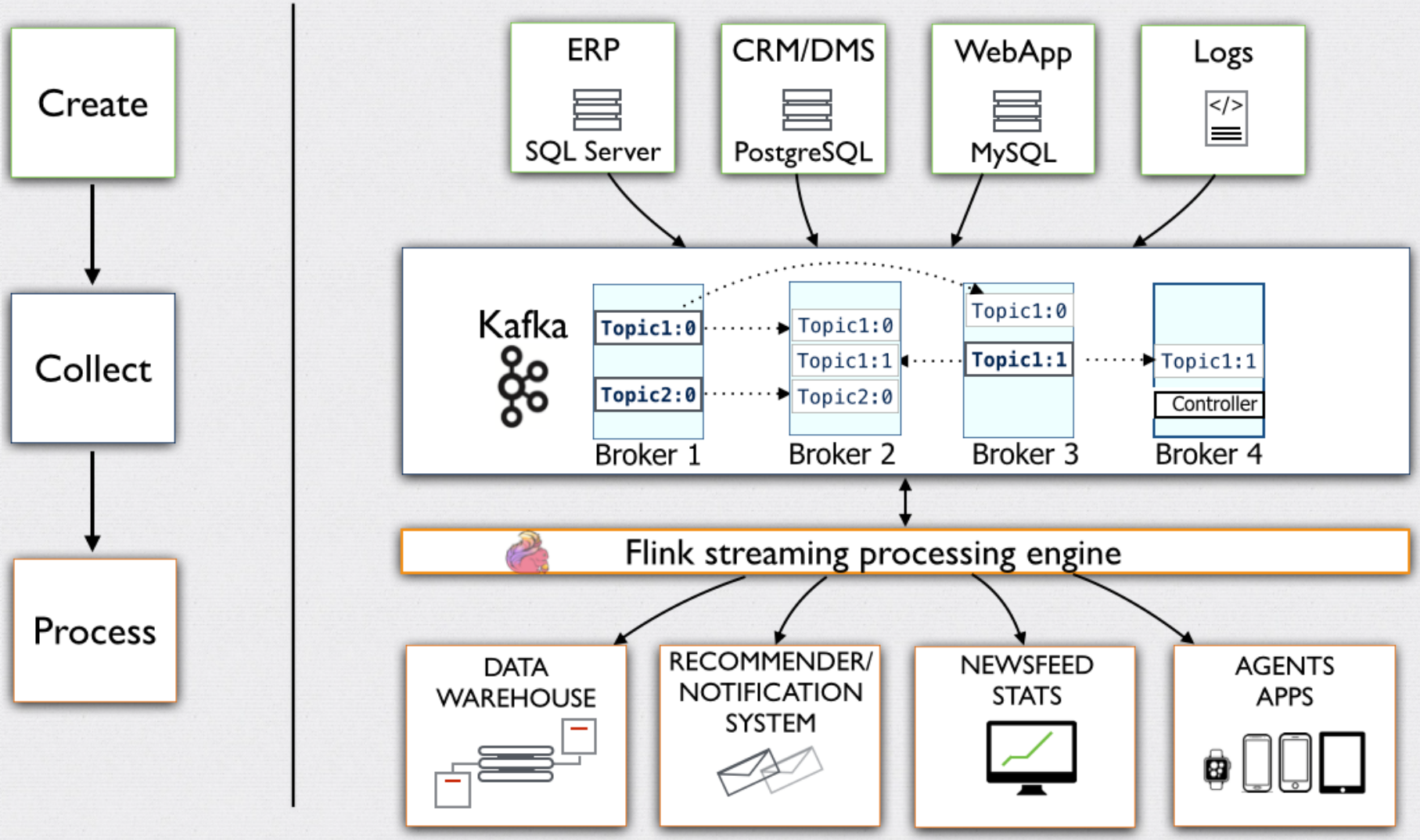
Streaming core

True streaming capabilities: higher throughput by running multiple operators' instances **concurrently** and pushing data between then as soon as they are processed, directly producing output from streams.

Query 6 - Timeline



Streaming and Flink + Kafka scenario



Thank you for your attention

Vielen Dank für Ihre Aufmerksamkeit