In-Memory Computing SUMMIT | EUROPE 2018

SnappyData: Apache Spark Meets Embedded In-Memory Database

Masaki Yamakawa

UL Systems, Inc.

In-Memory Computing SUMMIT EUROPE 2018

Masaki Yamakawa

UL Systems, Inc.

Managing Consultant

{

“Sector” : ”**Financial**”

“Skills” : [“**Distributed system**”, “**In-memory computing**”]

“Hobbies” : ”**Marathon running**”

}

未来をHackする

Agenda

In-Memory Computing SUMMIT EUROPE 2018

1. Current Issues of Real-Time Analytics Solutions
2. SnappyData Features
3. Our SnappyData Case Study

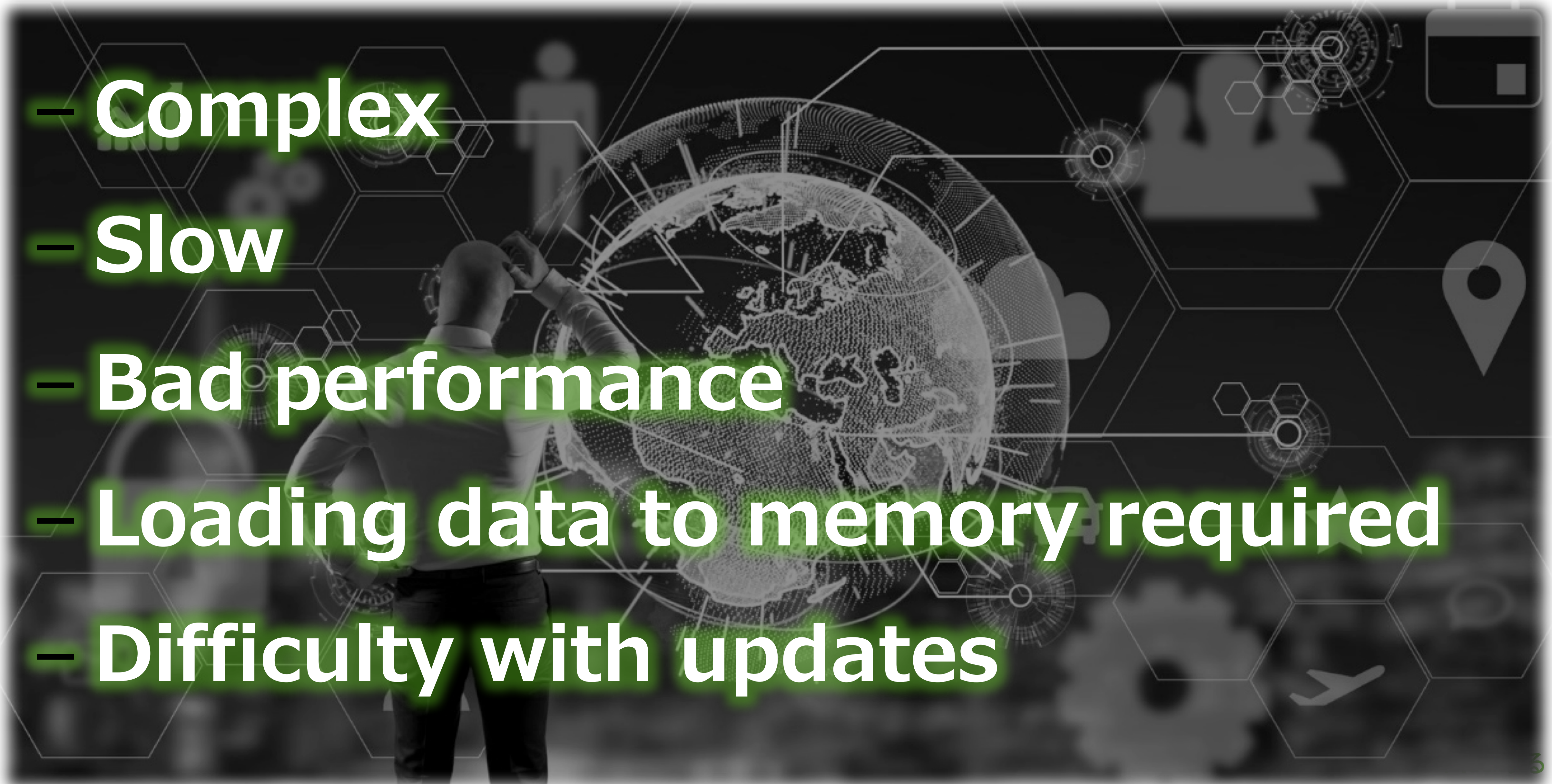
Current Issues of Real-Time Analytics Solutions

PART 1

In-Memory Computing SUMMIT | EUROPE 2018

Are you satisfied with real-time analytics solutions?

In-Memory Computing SUMMIT EUROPE 2018

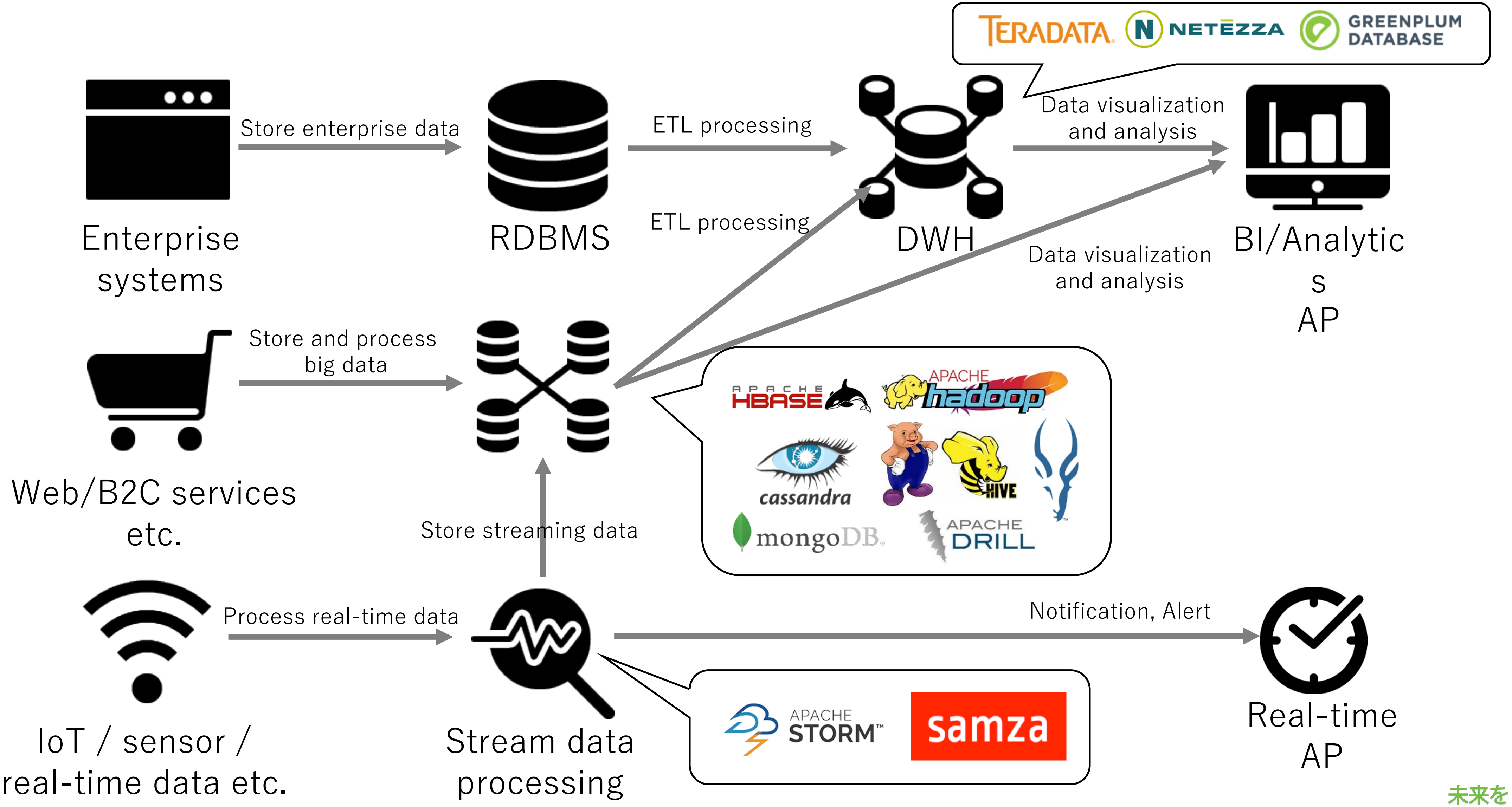


– Complex
– Slow
– Bad performance
– Loading data to memory required
– Difficulty with updates

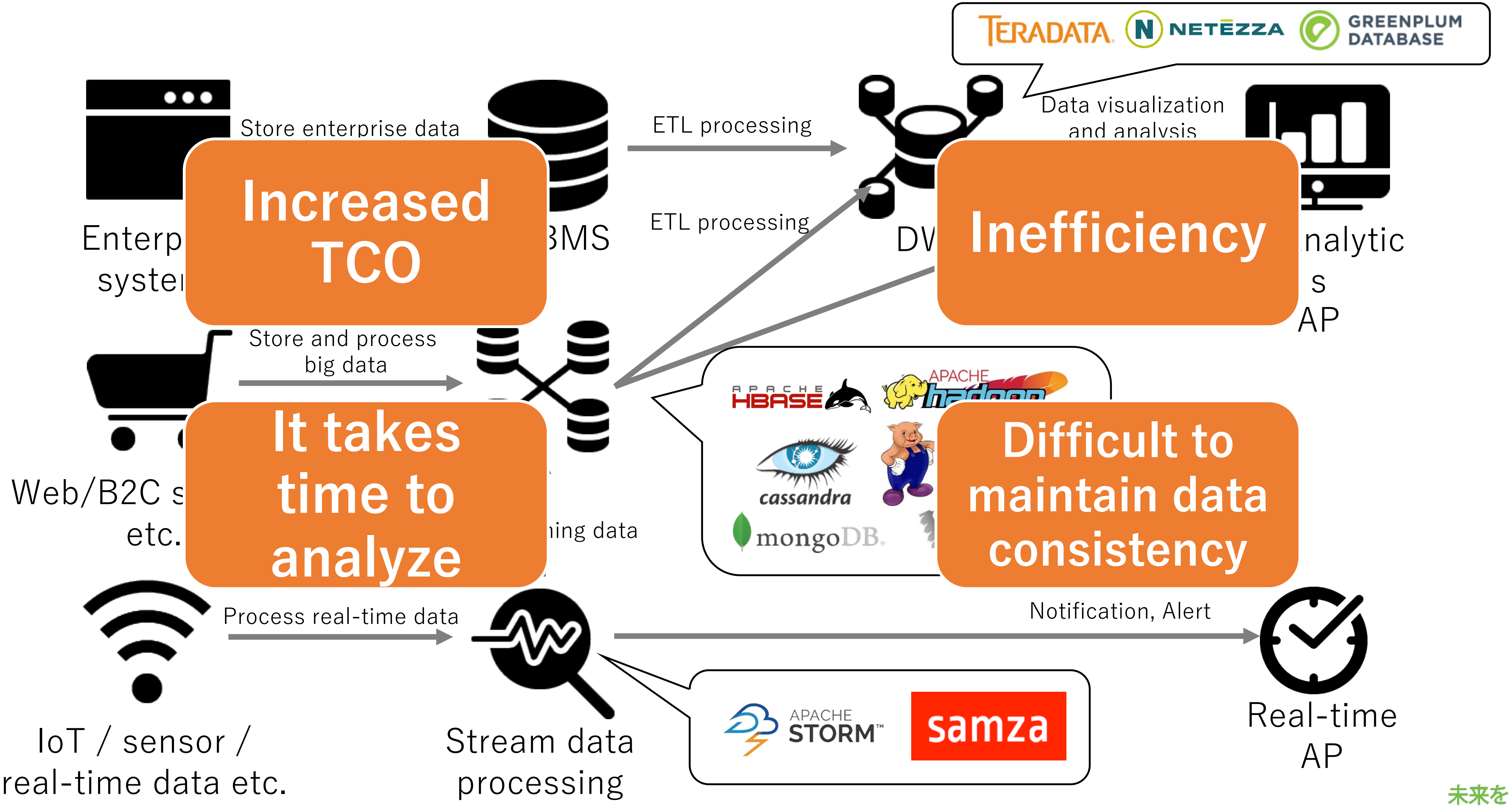
What are common demands for data processing platform?

	Transaction	Analytics	Streaming
Traditional data processing	RDBMS	DWH TERADATA NETEZZA GREENPLUM DATABASE	
Bigdata processing	NoSQL cassandra APACHE HBASE mongoDB	SQL on Hadoop APACHE hadoop HIVE APACHE DRILL	Stream data processing APACHE STORM™ samza

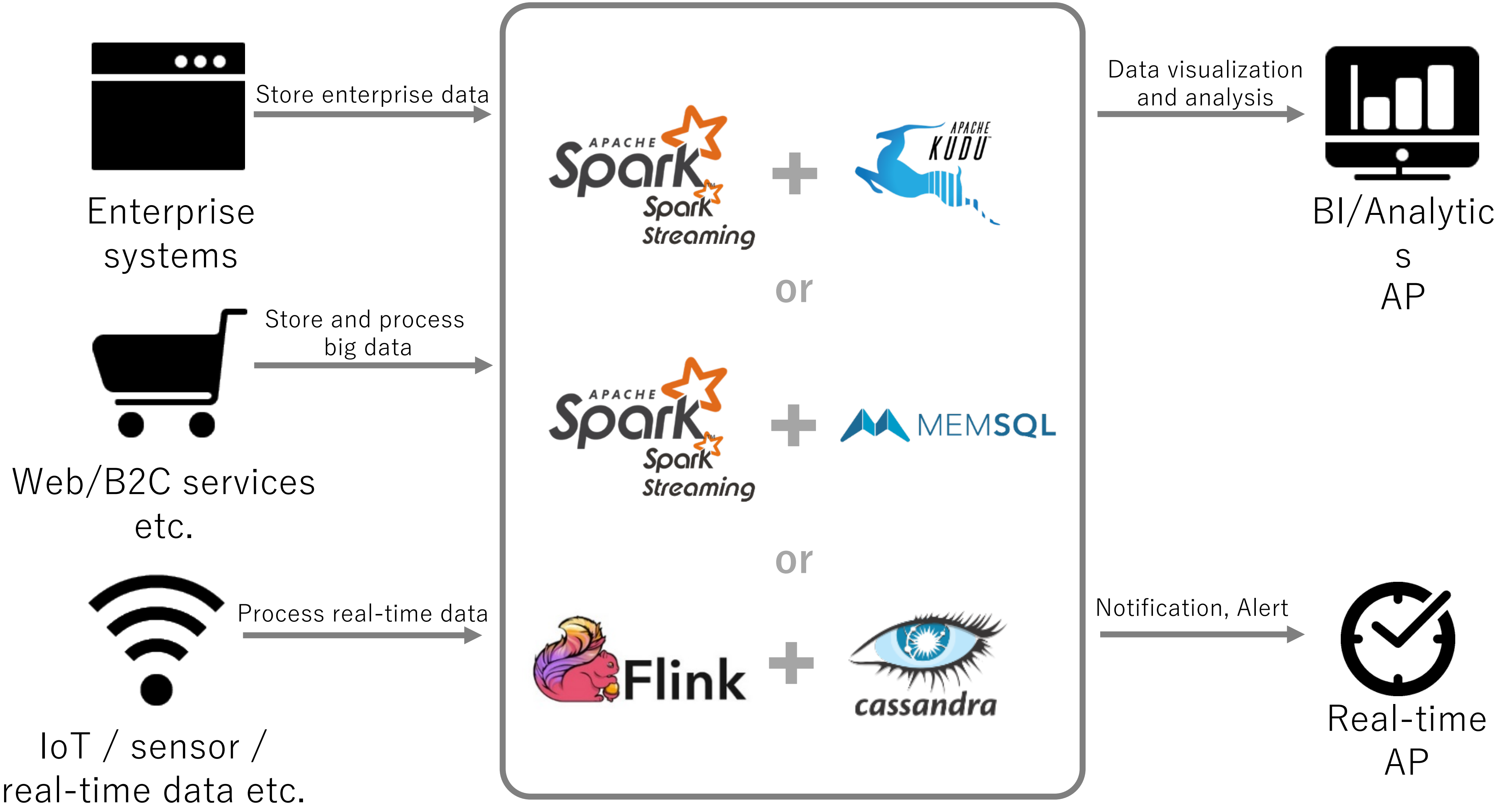
Tends to become complex system when integrates with multiple products



Tends to become complex system when integrates with multiple products



Although it became quite simple after Spark released…?



SnappyData can build simpler real-time analytics solutions!

In-Memory Computing SUMMIT EUROPE 2018

Enterprise systems

Store enterprise data

Web/B2C services etc.

Store and process big data

IoT / sensor / real-time data etc.

Process real-time data

SnappyData

Data visualization and analysis

BI/Analytics AP

Notification, Alert

Real-time AP

未来をHackする

SnappyData Features

PART 2

SnappyData is the Spark Database for Spark users

NEW SnappyData: Unifying Data Theory & Practice

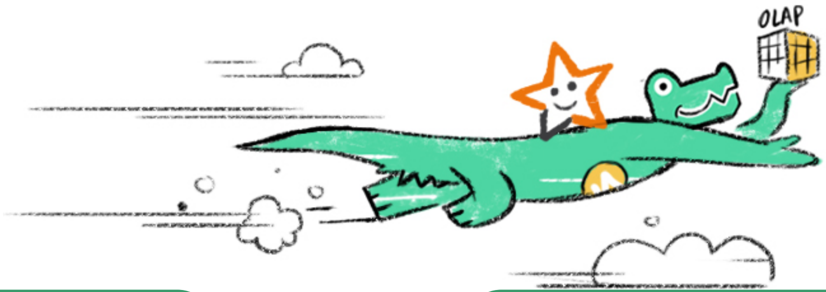
PRODUCT SOLUTIONS CLOUD RESOURCES BLOG COMMUNITY GET SNAPPY

SnappyData, the Spark Database.

SnappyData is a high performance in-memory data platform for mixed workload applications. Built on Apache Spark, SnappyData provides a unified programming model for streaming, transactions, machine learning and SQL Analytics in a single cluster. SnappyData unifies real time data sources with external data sources that have a Spark connector. SnappyData is 100% compatible with Apache Spark and is Open Sourced with an Apache V2 license.

Apache Spark+Distributed In-memory DB+Own features

SNAPPYDATA



Batch processing
Analytics
Stream processing

Distributed computing framework

Spark

+

Row database
Transaction

Distributed In-memory database

GemFire XD

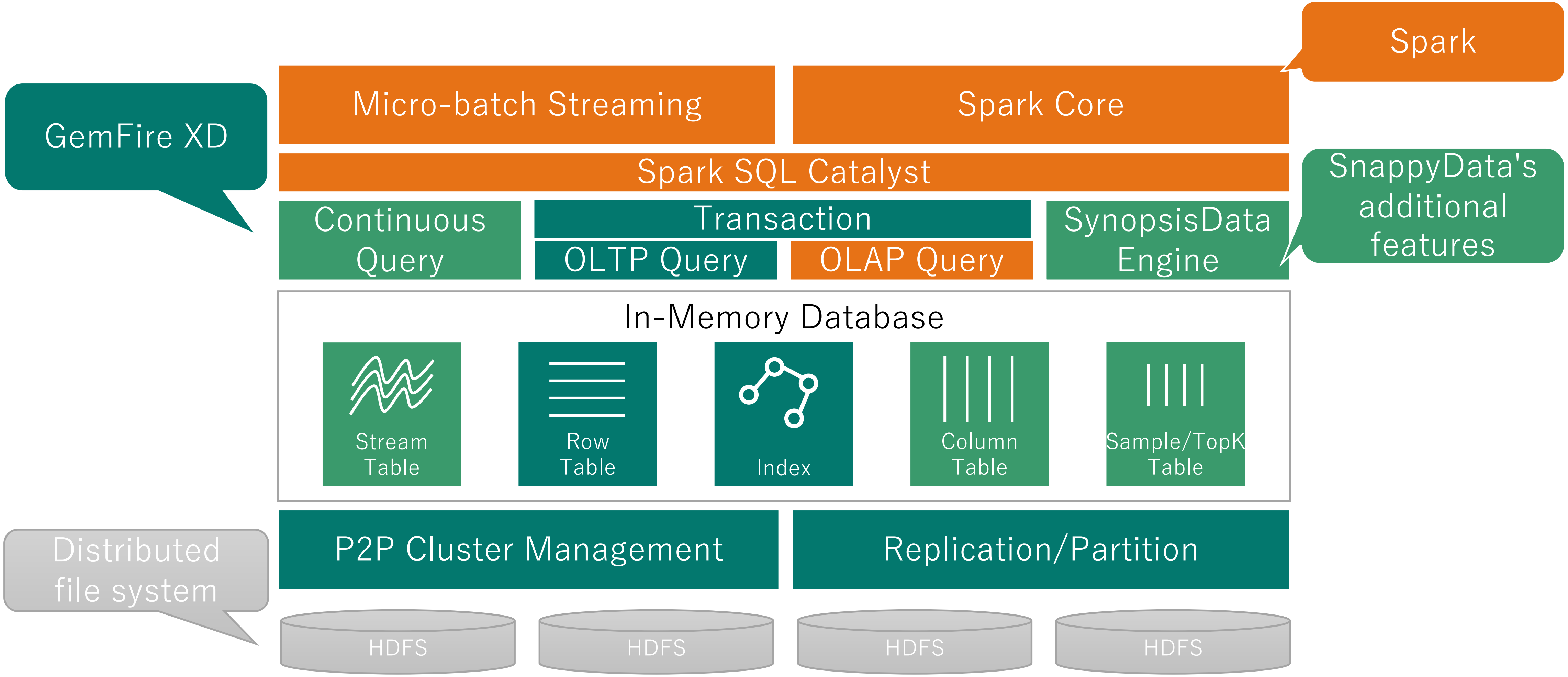
+

Columnar database
Synapsis Data Engine

SnappyData's own features

What is SnappyData's core component?

- Seamless integration of Spark and in-memory database conponents



Key to Spark program's accelerations

1. In-memory database
2. In-memory data format
3. Unified cluster
4. Optimized SparkSQL

Key#1: Data exists in in-memory database

In-Memory Computing SUMMIT EUROPE 2018

In case of Spark

Spark program

Spark — In memory

HDFS — **On disk**

In case of SnappyData

Spark program

SNAPPYDATA

Spark

Distributed in-memory database

In memory

Key#1: Data access code example

In-Memory Computing SUMMIT EUROPE 2018

In case of Spark	In case of SnappyData

In case of Spark

```
// load data from HDFS
val df = spark.sqlContext.read.
         format("com.databricks.spark.csv").
         option("header", "true").load("hdfs://...")
df.createOrReplaceTempView("SparkTable")

// create new DataFrame using SparkSQL
val filteredDf =
  spark.sql("SELECT * FROM SparkTable WHERE ...")
val newDf = filteredDf. ....

// save processing results
newDf.write.
  format("com.databricks.spark.csv").
  option("header", "false").save("hdfs://...")
```

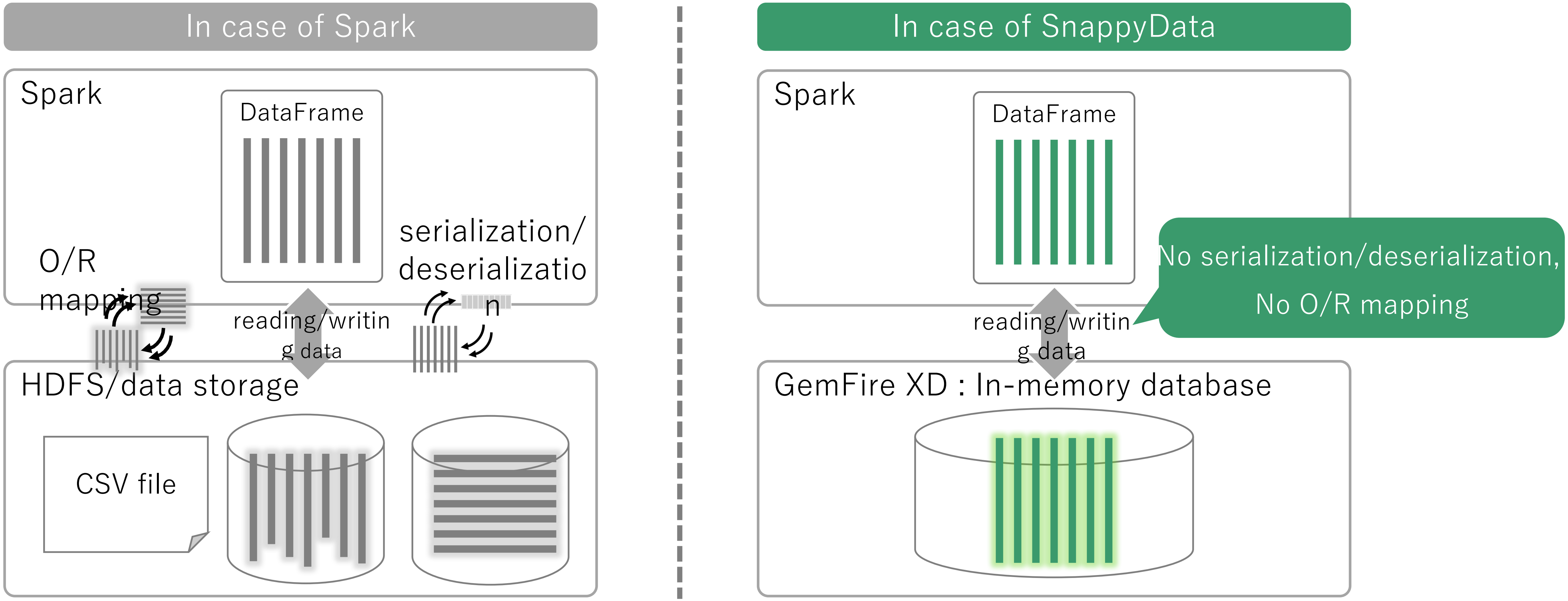
In case of SnappyData

```
// create SnappySession from SparkContext
val snappy = new org.apache.spark.sql.
                    SnappySession(spark.sparkContext)
```

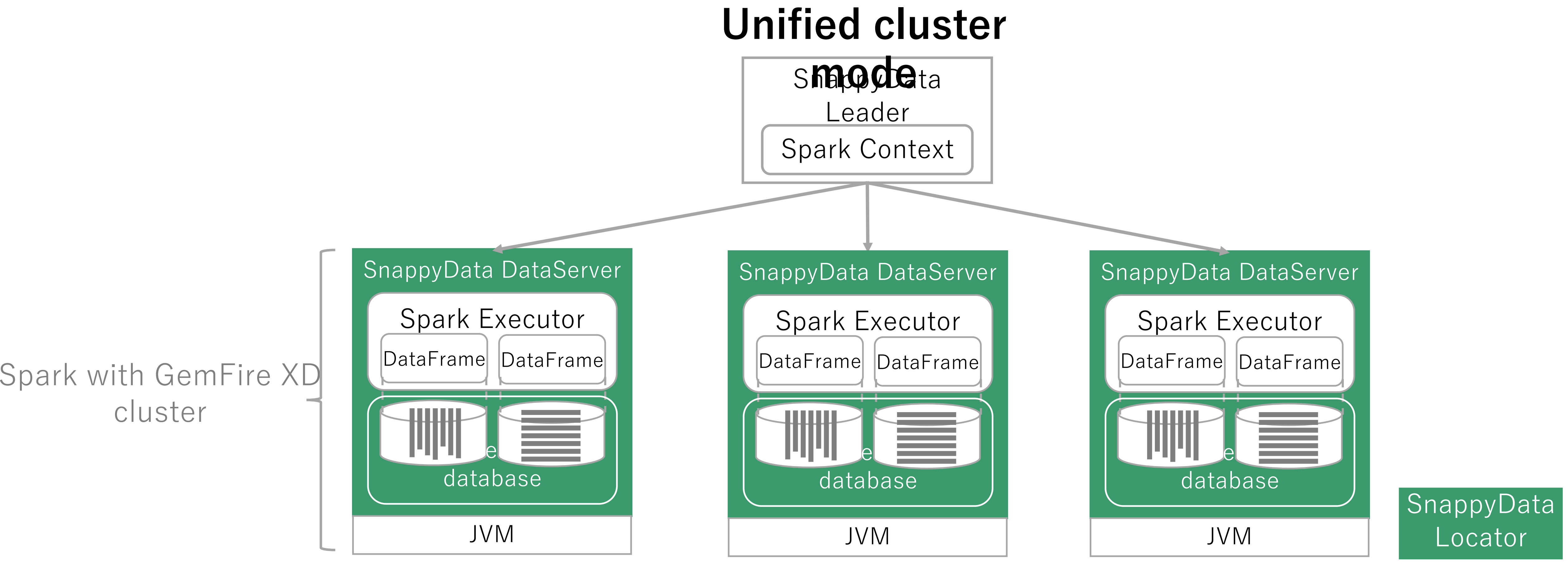
No need to load data

```
// create new DataFrame using SparkSQL
val filteredDf =
  snappy.sql("SELECT * FROM SnappyTable WHERE …")
val newDf = filteredDf. ....

// save processing results
newDf.write.insertInto("NewSnappyTable")
```

Key#2: SnappyData same data format as Spark's

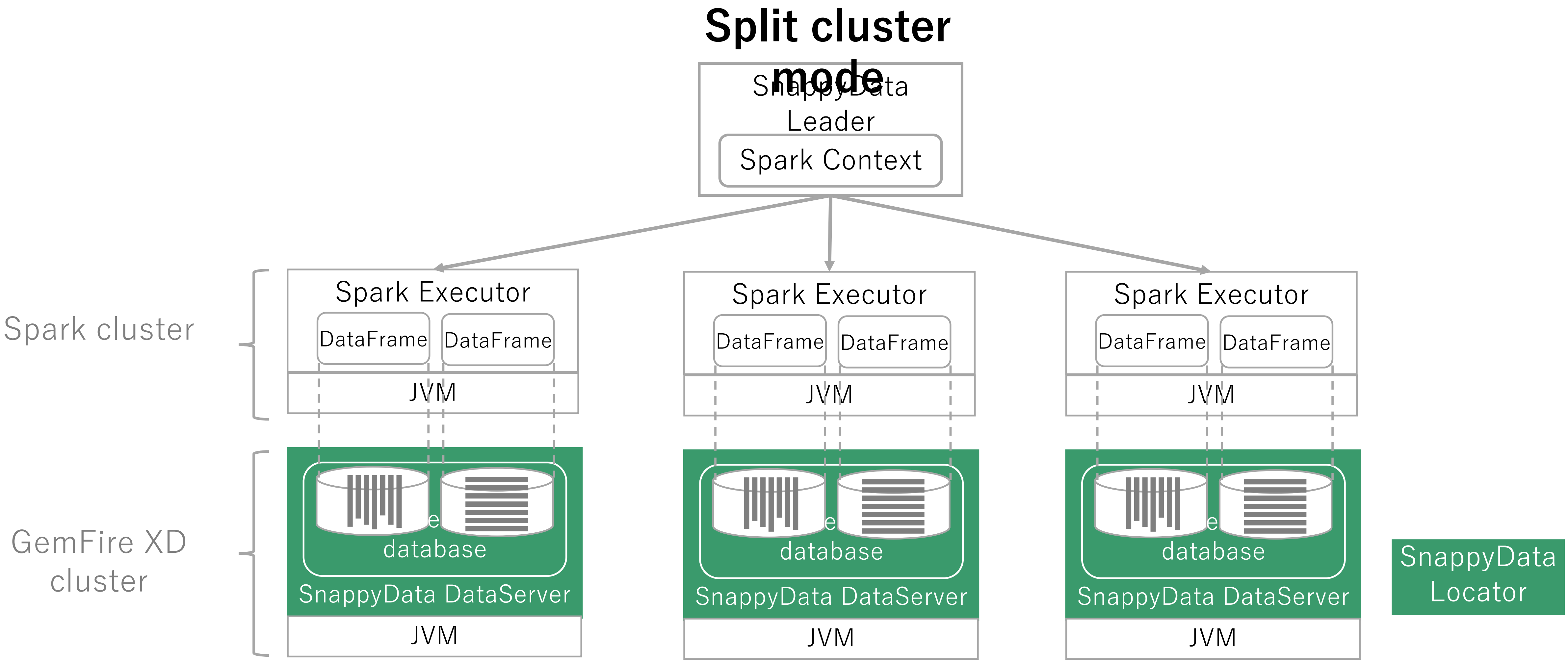


Key#3: Spark and GemFire XD cluster can be integrated



Key#3: Another cluster mode (for your reference)

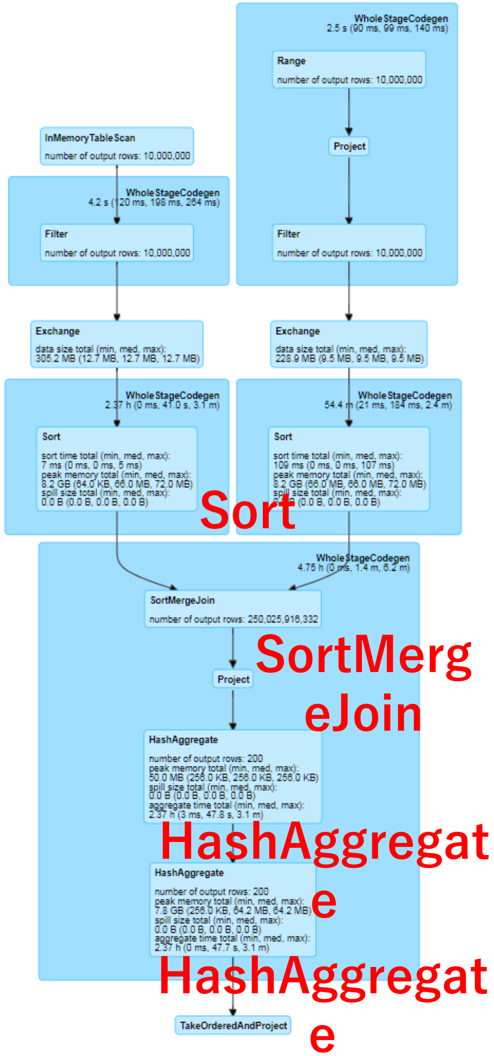
In-Memory Computing SUMMIT EUROPE 2018



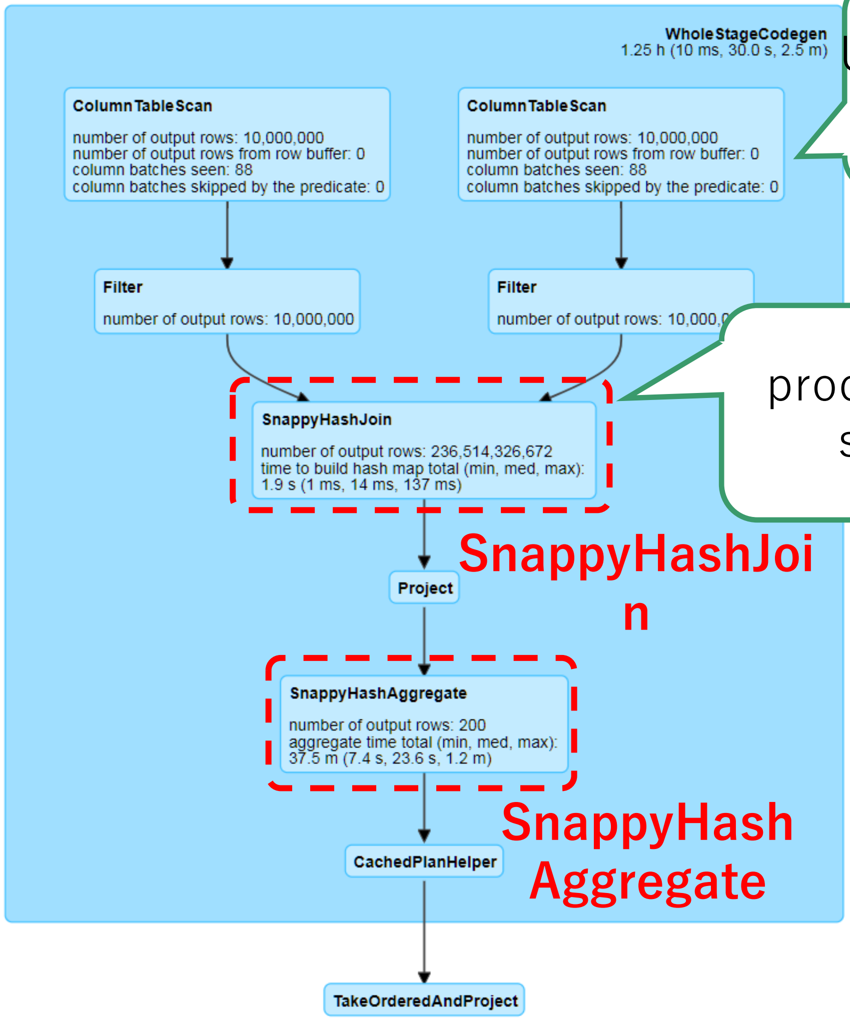
Key#4: SparkSQL Acceleration

In case of Spark

```
SELECT
  A.CardNumber,
  SUM(A.TxAmount)
FROM
  CreditCardTx1 A,
  CreditCardComm B
WHERE
  A.CardNumber=B.CardNumber AND
  A.TxAmount+B.Comm < 1000
GROUP BY
  A.CardNumber
ORDER BY
  A.CardNumber
```



In case of SnappyData



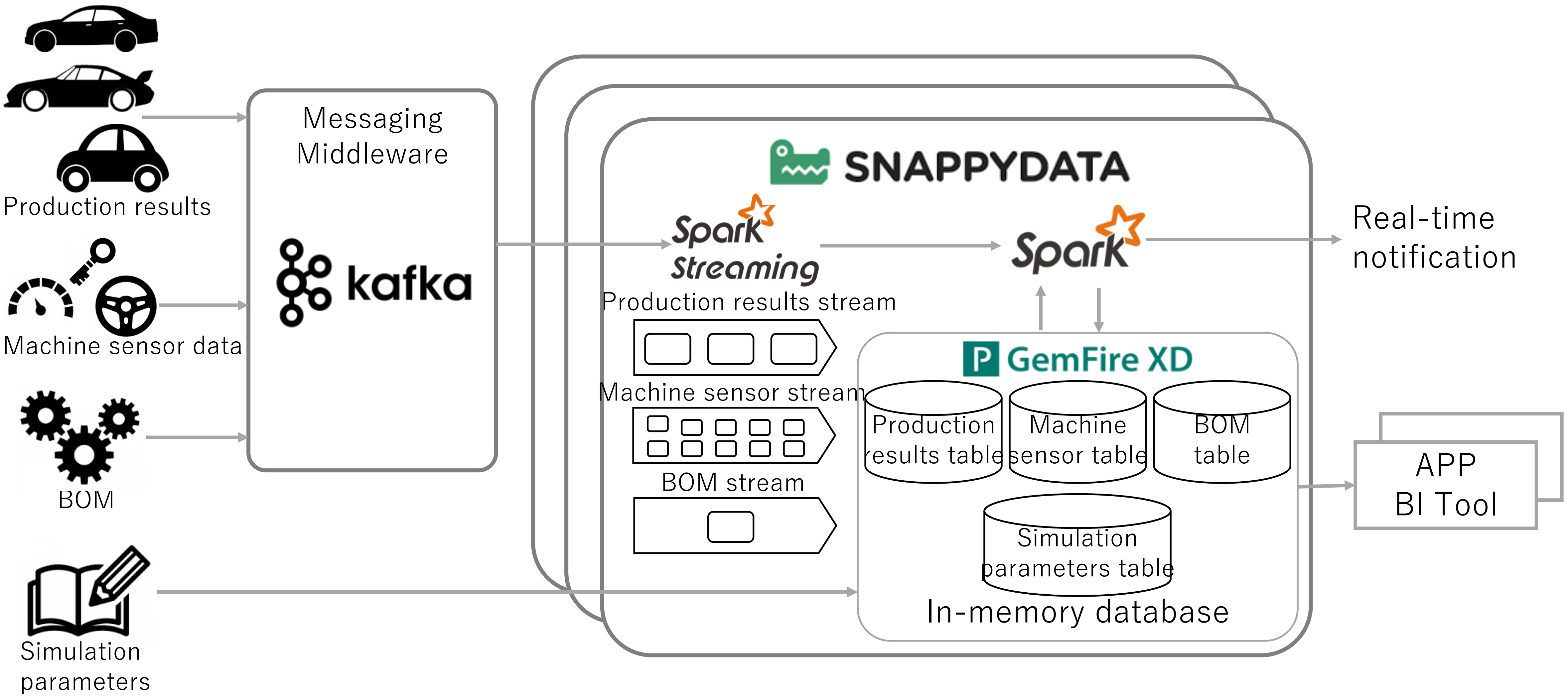
Unique DAG is generated, less shuffle and faster

Accelerate the processing by modifying some workload of SparkSQL

Our SnappyData Case Study:
How to use SnappyData

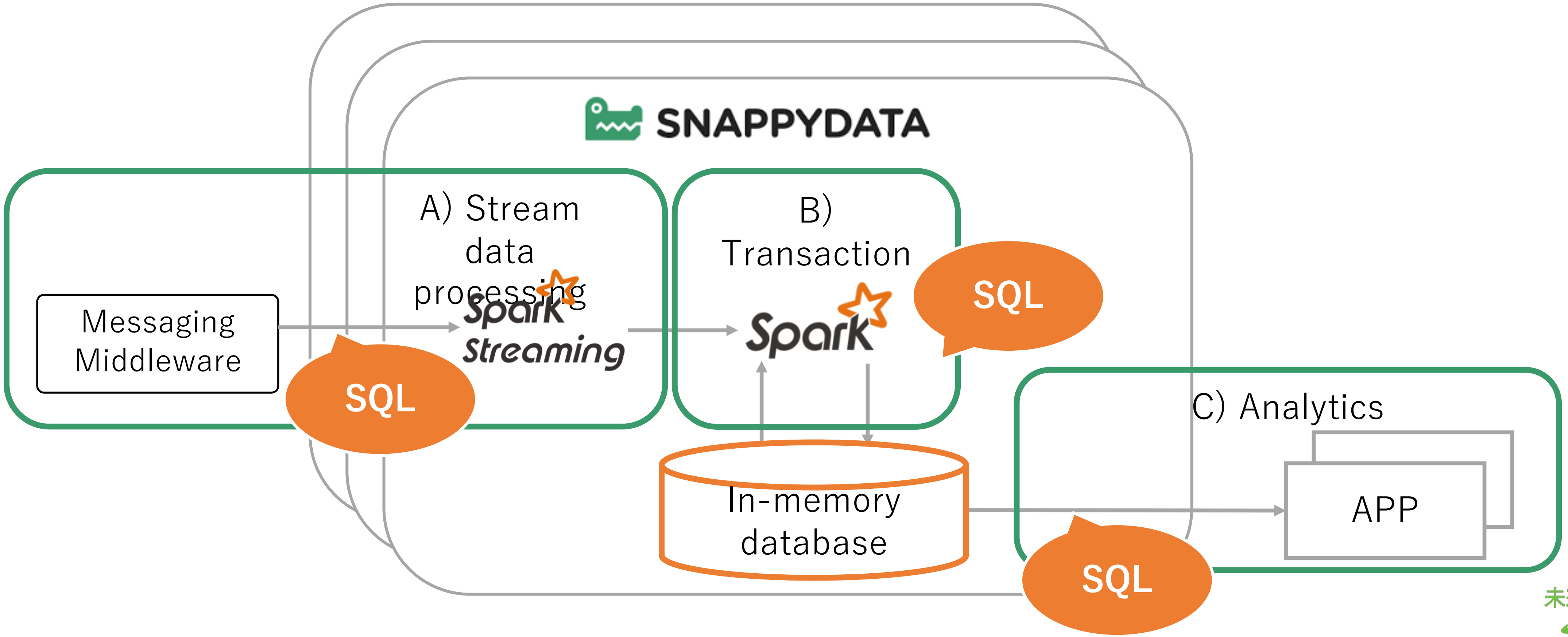
PART 3

Example of use: Production plan simulation system



Architecture with SnappyData

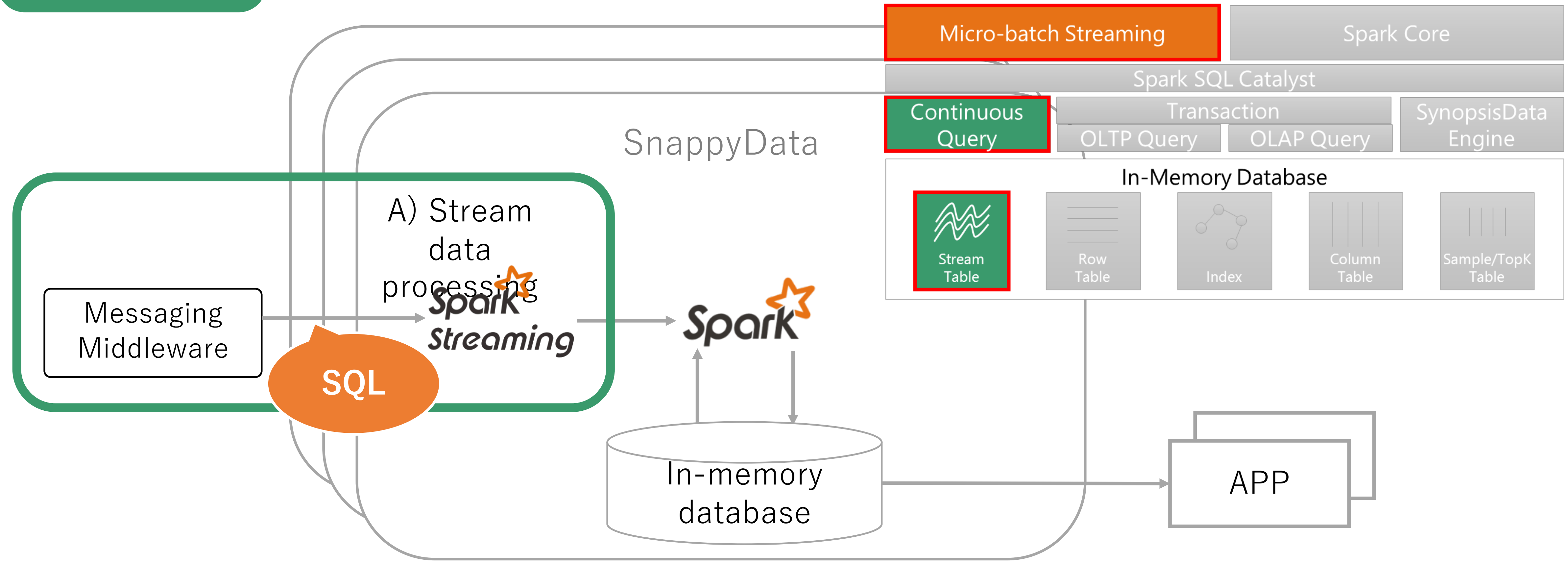
- Use SnappyData to realize all data processings such as stream processings, transactions, analytics
- The key is that it includes in-memory database and can be processed by SQL



A) Stream Data Processing

Difference from plain Spark

- The stream data is inserted into the table
- Stream data processing can be executed by SQL



SnappyData implements stream data processing using SQL

In-Memory Computing SUMMIT EUROPE 2018

Stream table

Sensorld	VIN	MachineNo	Point	Value	Timestamp
1	11AA	111	1	28.0760	2017/11/05 10:10:01
2	22BB	222	37	60.069	2017/11/05 10:10:20
3	11AA	111	2	37.528	2017/11/05 10:10:21
4	33CC	333	25	1.740	2017/11/05 10:11:05
5	11AA	111	3	88.654	2017/11/05 10:11:15
6	11AA	111	4	394.390	2017/11/05 10:11:16

Process（Continuous Query）

```
SELECT
  *
FROM
  MachineSensorStream
    WINDOW
    (DURATION 10 SECONDS,
     SLIDE 2 SECONDS)
WHERE
  Point=1;
```

Only specifies stream data source info in table definition


```
CREATE STREAM TABLE MachineSensorStream
  (SensorId      long,
   VIN           string,
   MachineNo     int,
   Point         long
   Value         double,
   Timestamp     timestamp)
USING KAFKA_STREAM
 OPTIONS
   (storagelevel  'MEMORY_AND_DISK_SER_2',
    rowConverter  'uls.snappy.KafkaToRowsConverter',
    kafkaParams   'zookeeper.connect->localhost:2181;xx',
    topics        'MachineSensorStream');
```

Streaming data source other than Kafka

- TWITTER_STREAM
- DIRECTKAFKA_STREAM
- RABBITMQ_STREAM
- SOCKET_STREAM
- FILE_STREAM

Streaming data source

Storage level (Spark setting)

Stream data row converter class

Setting for each streaming data source

Implements StreamToRowsConverter and converts to table format

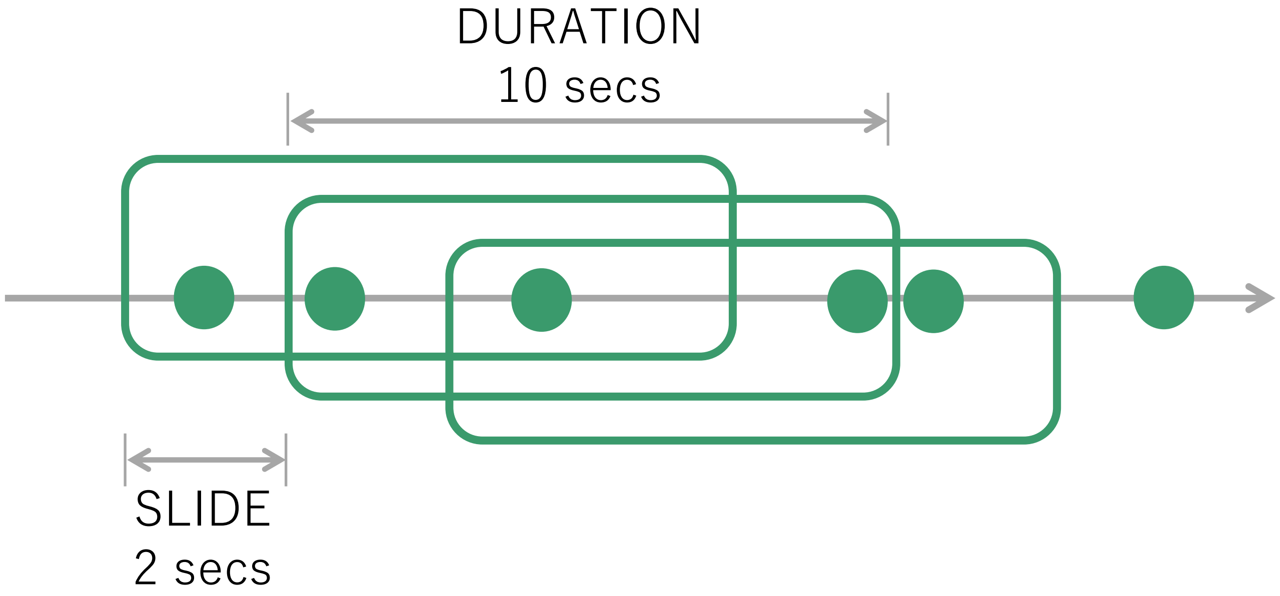

```
class KafkaToRowsConverter extends StreamToRowsConverter with Serializable {

  override def toRows(message: Any): Seq[Row] = {
    val sensor: MachineSensorStream = message.asInstanceOf[MachineSensorStream]

    Seq(Row.fromSeq(Seq(sensor.getSensorId,
                        sensor.getVin,
                        sensor.getMachineNo,
                        sensor.getPoint,
                        sensor.getValue,
                        sensor.getTimestamp)))
  }
}
```

Data for one row of stream table

Stream data processing using SQL

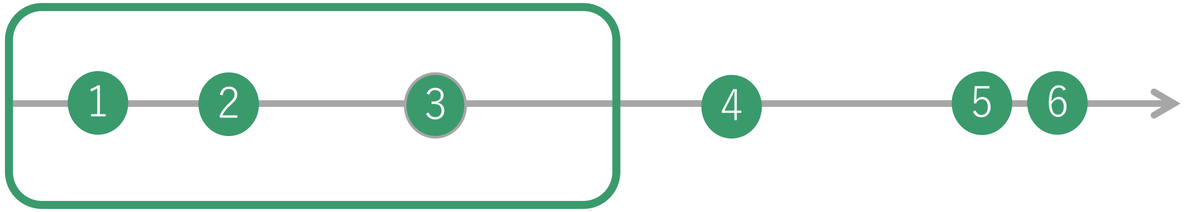
```
SELECT
  *
FROM
  MachineSensorStream
    WINDOW
    (DURATION 10 SECONDS,
     SLIDE 2 SECONDS)
WHERE
  Point=1;
```

Point acquires “1" data in
2 secs sliding window



In Continuous Query, only data included in WINDOW is acquired

In-Memory Computing SUMMIT EUROPE 2018

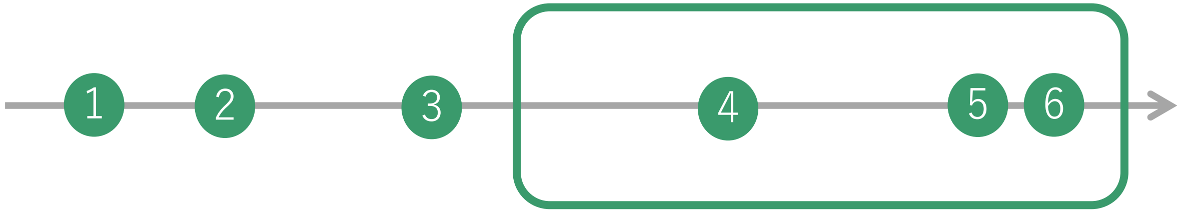


Sensor Id	VIN	Machine No	Point	Value	Timestamp
1	11AA	111	1	28.0760	2017/11/05 10:10:01
2	22BB	222	37	60.069	2017/11/05 10:10:20
3	11AA	111	2	37.528	2017/11/05 10:10:21

After 2 seconds

Sensor Id	VIN	Machine No	Point	Value	Timestamp
3	11AA	111	2	37.528	2017/11/05 10:10:21
4	33CC	333	25	1.740	2017/11/05 10:11:05

After 2 seconds



Sensor Id	VIN	Machine No	Point	Value	Timestamp
4	33CC	333	25	1.740	2017/11/05 10:11:05
5	11AA	111	3	88.654	2017/11/05 10:11:15
6	11AA	111	4	394.390	2017/11/05 10:11:16

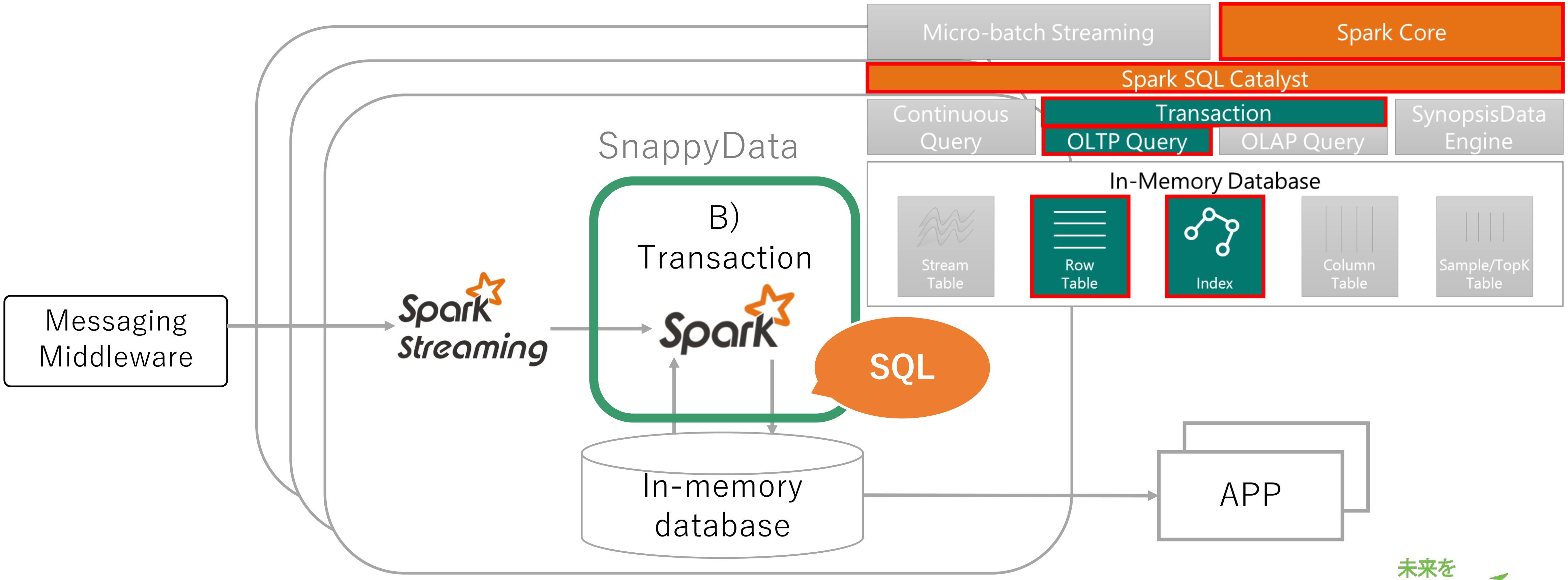
Stream data processing code example

```
// create SnappyStreamingContext from SparkStreamingContext
val snappy = new SnappyStreamingContext(sc, 10)

// register Continuous Query
val machineSensorStream : SchemaDStream = snappy.registerCQ(s"""
 SELECT
    SensorId, VIN, MachineNo, Point, Value, Timestamp
  FROM MachineSensorStream
    WINDOW
      (DURATION 10 SECONDS, SLIDE 2 SECONDS)
  WEHERE
    Point=1
""")

// process stream data
machineSensorStream.foreachDataFrame(df => {
  ...
  df.write.insertInto("MachineSensorHistory")
  ...
})
```

B) Transaction

Difference from plain Spark

- Insert, update, and delete to DataFrame are reflected in In-memory database
- SnappyData has the same function as RDBMS



Data insert, update, delete code example

In-Memory Computing SUMMIT EUROPE 2018

```
bomStream.foreachRDD(rdd => {
  val streamDS = rdd.toDS()

  // Delete from BOM table
  streamDS.where("ACTION = 'DELETE'").write.deleteFrom("BOM")
  // Insert/Update to BOM table
  streamDS.where("ACTION = 'INSERT'").write.putInto("BOM")
})

machineSensorStream.foreachRDD(rdd => {
  val streamDS = rdd.toDS()

  // Create BOM table DataFrame
  val bom = snappy.table("BOM")

  // Register join result in faulty parts table
  val faultyParts = streamDS.join(bom, $"PartsNo" === $"PartsNo", "leftsemi")
  val faulty = faultyParts.select("SensorId", "VIN", "MachineNo", "PartsNo", "Timestamp")
  faulty.write.insertInto("FaultyParts")
})
```

Possible to insert, update, and delete in standard SQL using SnappySession

Possible to use different table formats depending on data characteristics

In-Memory Computing SUMMIT EUROPE 2018

Row table
(For master data / transaction data)

Parts No	Parts Type	Effective Date	…
999999999	1	2020/12/01	
876543210	1	2019/04/02	
213757211	2	2020/02/02	
555444777	1	2018/08/13	
	…		
987654321	2	2022/09/30	

■frequently inserted or updated
■lookup by key

Column table
(For aggregate / analysis data)

Sensor Id	VIN	MachineNo	Point	Value	…
1	11AA	111	1	28.0760	
2	22BB	222	37	60.069	
3	11AA	111	2	37.528	
4	33CC	333	25	1.740	
	…				
6	11AA	111	4	394.390	

■aggregate or group by specified columns

未来をHackする

You can create tables with DDL like RDB

- Additional settings for data distribution and persistence are required

Row table

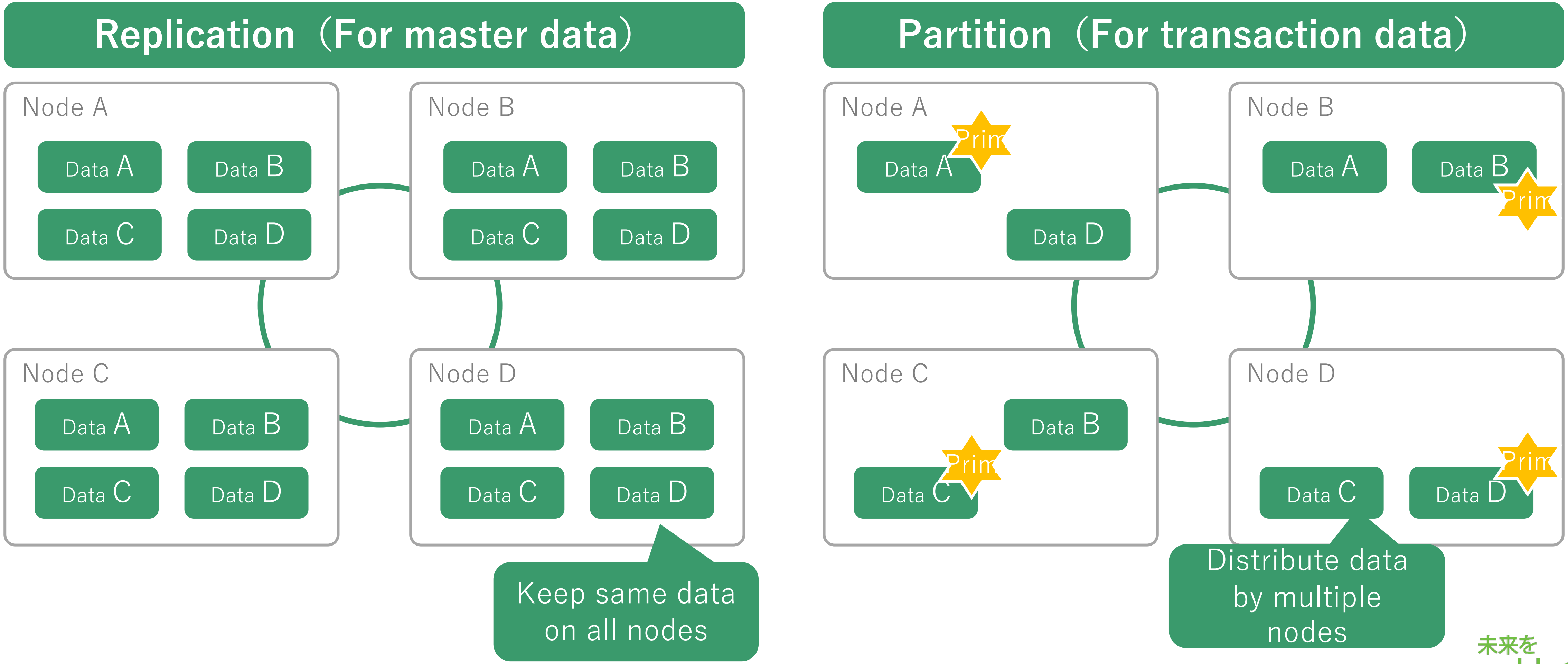
```
CREATE TABLE BOM
 (PartsNo         CHAR(16)    NOT NULL PRIMARY KEY,
  PartsType       CHAR(1)     NOT NULL,
  EffectiveDate   DATE        NOT NULL,
  …               CHAR(3)             ,
  …               CHAR(1)             ,
  …               DATE                ,
  …               DECIMAL(9,2))
USING ROW                                   -- Database engine
  OPTIONS
    (PARTITION_BY    'PartsNo',             -- Data distribution setting
     COLOCATE_WITH   'PartsType',
     REDUNDANCY      '1',
     EVICTION_BY     'LRUMEMSIZE 10240',    -- Persistence setting
     OVERFLOW        'true',
     DISKSTORE       'LOCALSTORE',
     PERSISTENCE     'ASYNC',
     EXPIRE          '86400');              -- Expire option
```

Column table

```
CREATE TABLE MachineSensor
 (SensorId        BIGINT              ,
  VIN             CHAR(20)            ,
  MachineNo       CHAR(16)            ,
  Value           DECIMAL(15,2)       ,
  Point           CHAR(2)             ,
  …               DATE                ,
  …               DECIMAL(9,2))
USING COLUMN                                -- Database engine
  OPTIONS
    (PARTITION_BY    'SensorID',            -- Data distribution setting
     COLOCATE_WITH   'PartsType',
     REDUNDANCY      '1',
     EVICTION_BY     'LRUMEMSIZE 10240',    -- Persistence setting
     OVERFLOW        'true',
     DISKSTORE       'LOCALSTORE',
     PERSISTENCE     'ASYNC');
```

Needs to use replication and partition properly

- Replication can be used instead of broadcast variable
- Partition can be used instead of RDD

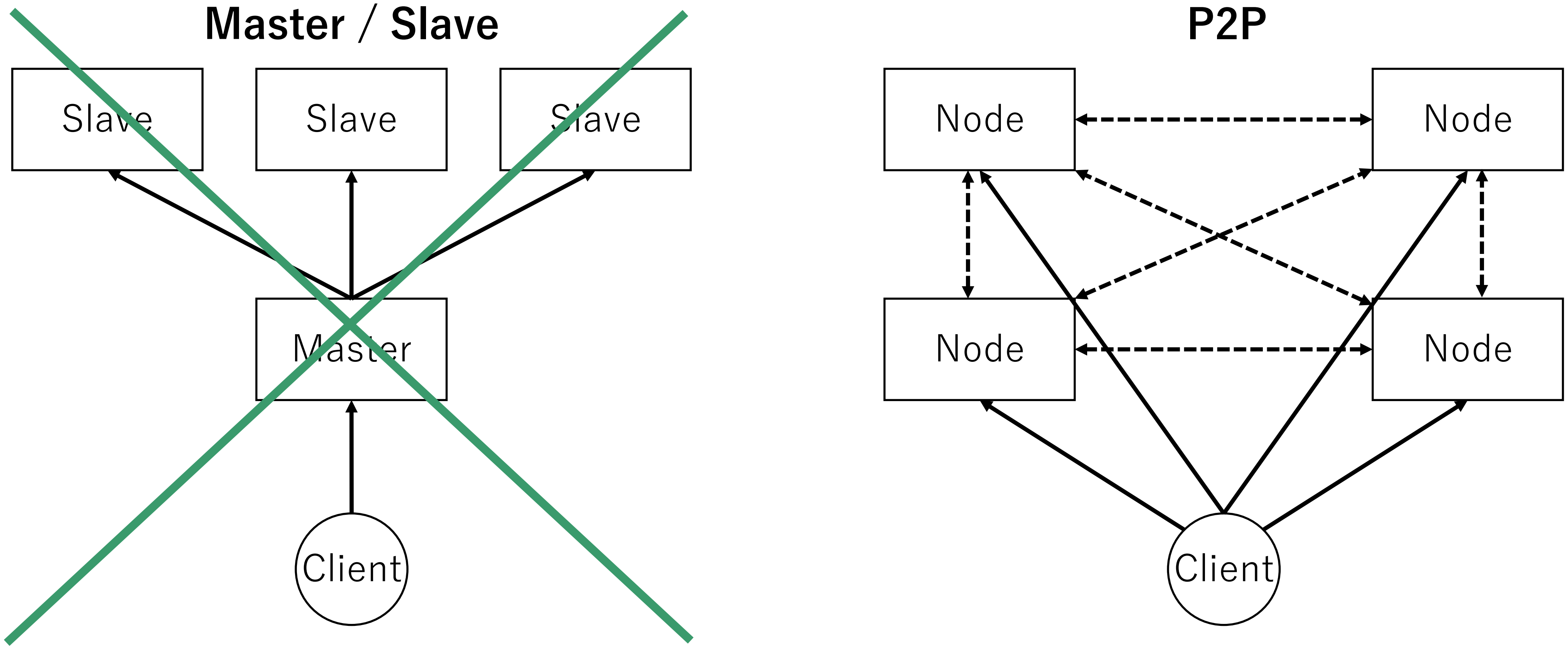


Transactions can be used like RDBMS

Supported transaction isolation level	READ UNCOMMITTED	×
	READ COMMITTED	○
	REPEATABLE READ	○
	SERIALIZABLE	×
SELECT FOR UPDATE	Conflict exception occurs at COMMIT when data is updated during transaction	
COMMIT/ROLLBACK	If the cluster member goes down during a transaction, an exception is raised that COMMIT failed	
LOCK TABLE	Not supported	
CASCADE DELETE	Not supported	

P2P Architecture

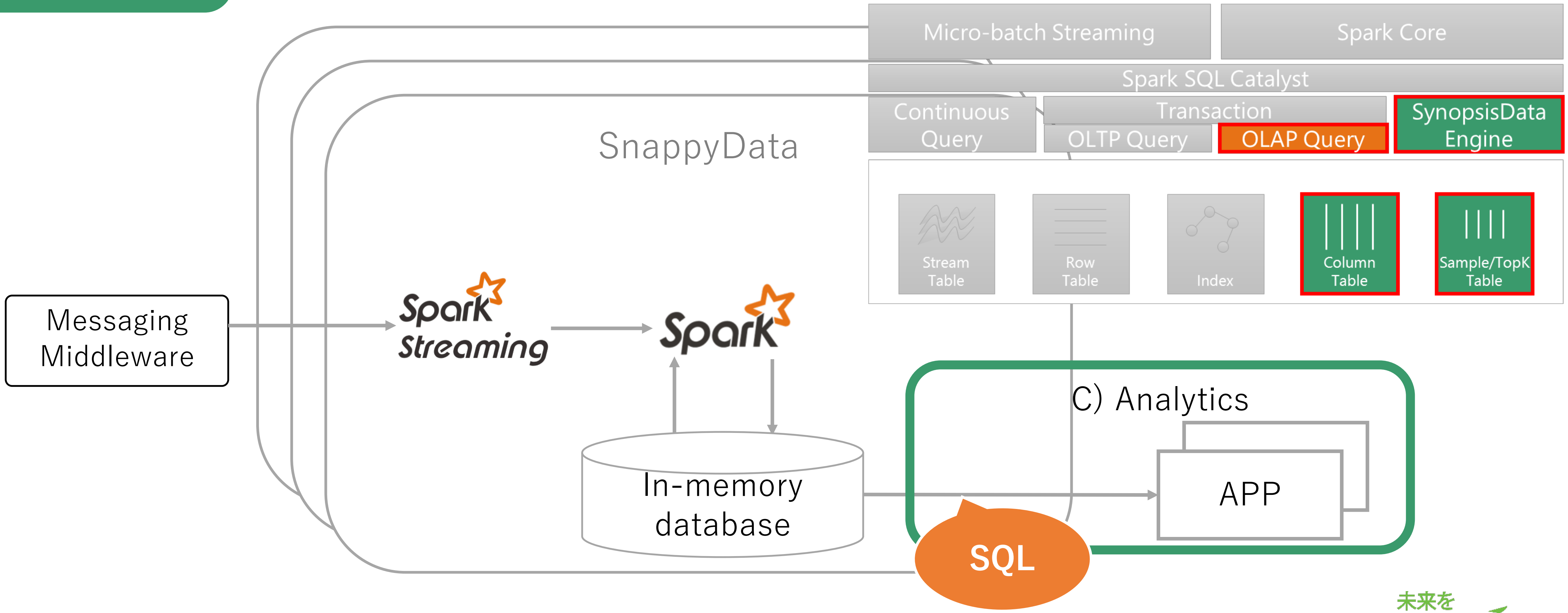
- SnappyData is masterless
- Possible to scale out not only the reading processes but also the writing processes



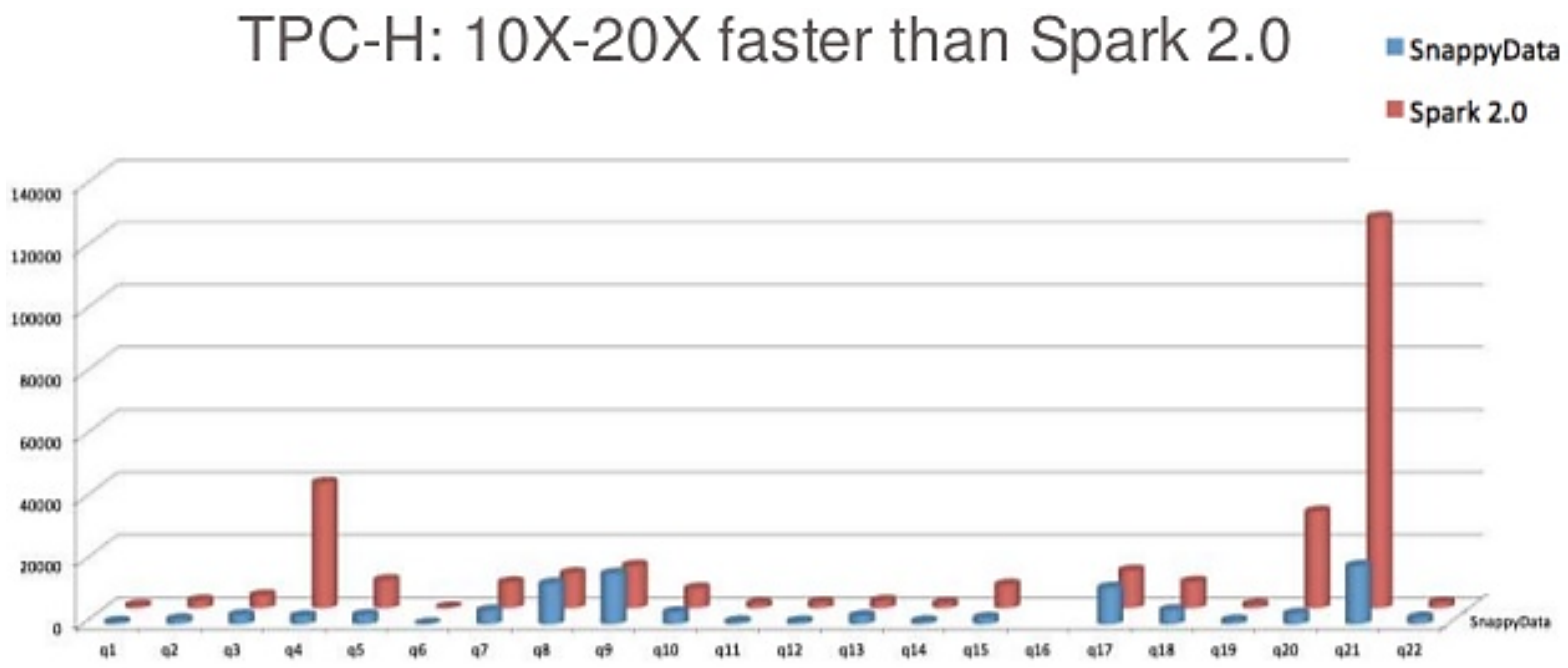
C) Analytics

Difference from plain Spark

- SnappyData has changed SparkSQL to become faster
- Approximate query, TOP K table can be used



SnappyData is 10-20X faster than SparkSQL



Implement unique features that enable real-time analysis

- Real-time analytics can't wait more than 10 seconds for one query

Synopsis Data Engine

Approximate query	Stratified Sampling / Random Sampling	Query against sampled data
TOP K table		Use Min / Max to detect outliers

Possible to sample based on cardinality of specific column

In-Memory Computing SUMMIT EUROPE 2018

- Specify information for sampling in standard DDL

Sample table

```
CREATE SAMPLE TABLE
        MachineSensorSample
 ON MachineSensor
   OPTIONS(
     qcs      'PartsNo, Year, Month',
     fraction '0.03')
 AS (SELECT * FROM MachineSensor)
```

Base table
MachineSensor

Sr Id	VIN	Year	Month	Value	…
1	111	2017	11	10,000	…
2	222	2017	11	3,980	…
3	111	2017	11	5,130	…
4	222	2017	11	323,456	…
5	111	2017	11	1,980	…
6	111	2017	11	23,456	…

It is sampled based on cardinality of the specified column

Create Sample table

Sample table
MachineSensorSample

Sr Id	VIN	Year	Month	Value	…
1	111	2017	11	10,000	…
2	222	2017	11	3,980	…
5	111	2017	11	1,980	…

For random sampling, no need to create sample tables

- Specify sampling information in SQL

Approximate Query

```
SELECT
  VIN,
  AVG(Value)
FROM
  MachineSensor
GROUP BY
  VIN
ORDER BY
  VIN
WITH
  ERROR 0.10
  CONFIDENCE 0.95
```

Base table
MachineSensor

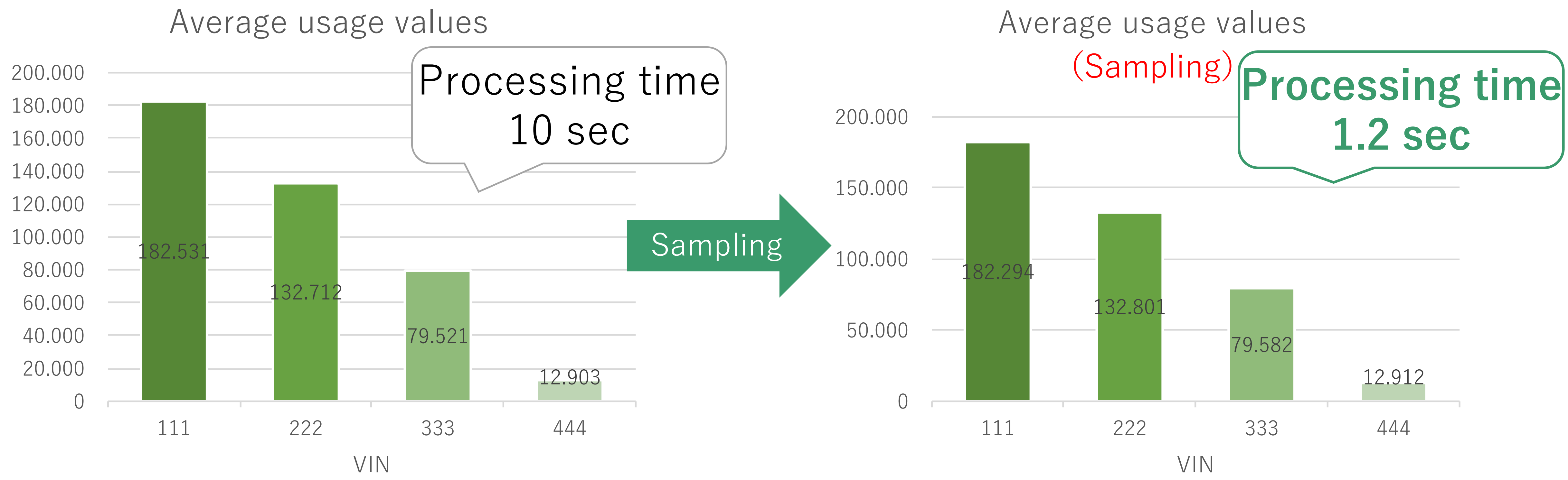
Sr Id	VIN	Year	Month	Tx Amount	…
1	111	2017	11	¥10,000	…
2	222	2017	11	¥3,980	…
3	111	2017	11	¥5,130	…
4	222	2017	11	¥323,456	…
5	111	2017	11	¥1,980	…
6	111	2017	11	¥23,456	…

Randomly sampled

SQL

Query results

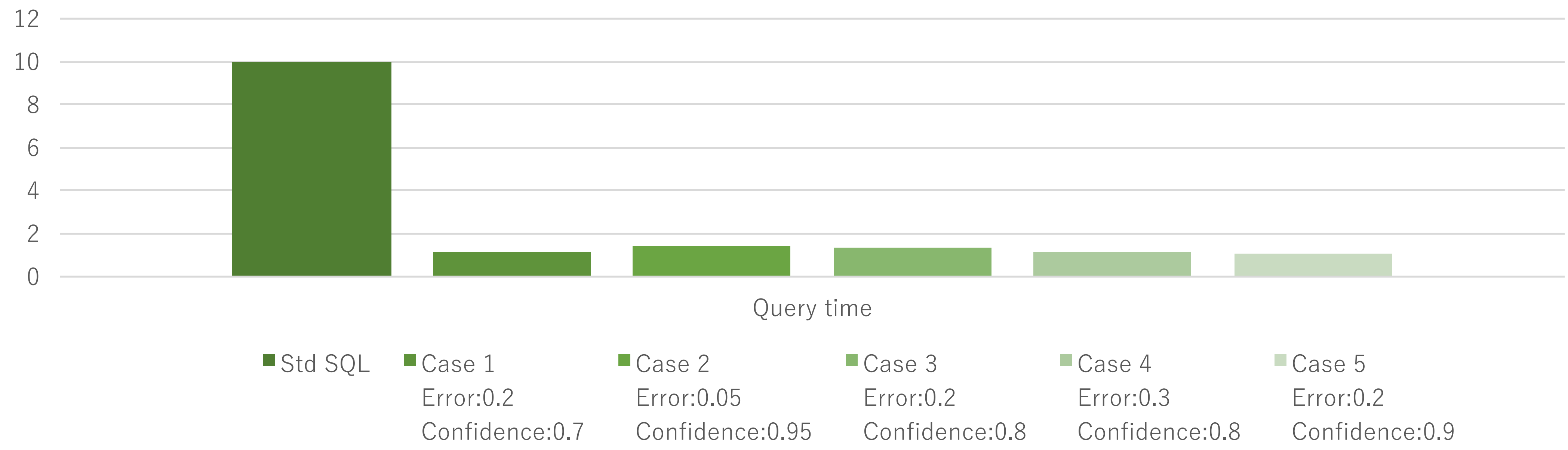
Faster query by utilizing sampling technique



Comparing confidence of query results and query performance...

In-Memory Computing SUMMIT EUROPE 2018

	Std SQL	Case 1	Case 2	Case 3	Case 4	Case 5
Error	-	0.20	0.05	0.20	0.30	0.20
Confidence	-	0.70	0.95	0.80	0.80	0.90
Query time	10.0 sec	1.2 sec	1.4 sec	1.3 sec	1.2 sec	1.1 sec



TOP K table

Collecting the top 50 cases with higher value at 1 minute intervals

TOP K table

```
CREATE TOPK TABLE HighestSensorValue
  on MachineSensor
OPTIONS (
  key              'MachineNo',
  timeInterval     '60000ms' ,
  size             '50'
  frequencyCol     'Value',
  timeSeriesColumn 'Timestamp '
)
```

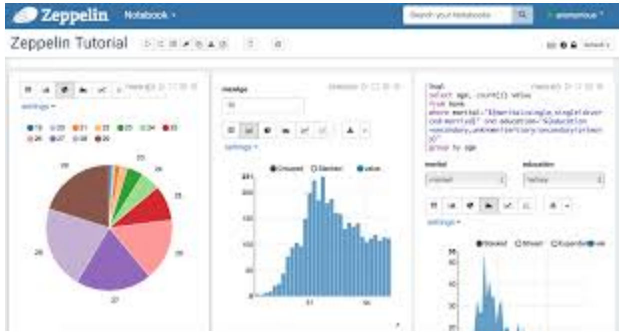
MachineNo	Value	…
666666666	1,2345,678	…
197654532	10,000,000	…
197654532	5,048,600	…
・・・	・・・	…

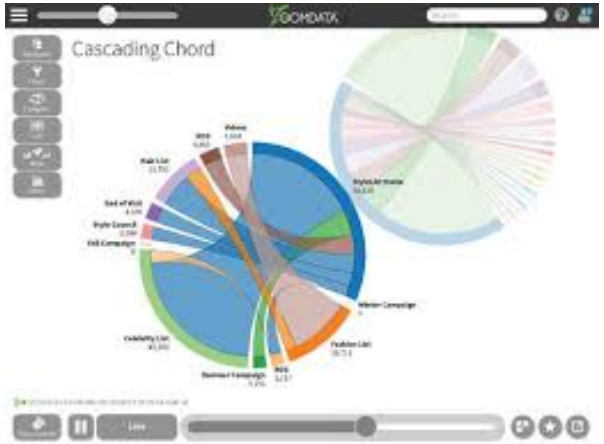
Connect to SnappyData from BI tool

Apache Zeppelin

tableau

ZOOMDATA





Code example: Connect to SnappyData from application

In-Memory Computing SUMMIT EUROPE 2018

```
// connect to SnappyData
val conn = DriverManager.getConnection("jdbc:snappydata://localhost:1527/APP")

// insert data
val psInsert = conn.
      prepareStatement("INSERT INTO MachineSensor VALUES(?, ?, ?, ?, …)")
psInsert.setString(1, "1000200030004000")
psInsert.setBigDecimal(2, java.math.BigDecimal.valueOf(100.2))
…
psInsert.execute()

// select data
val psSelect = conn.prepareStatement( "SELECT * FROM MachineSensor WHERE PartsNo=?")
psSelect.setString(1, "1000200030004000")
ResultSet rs = psSelect.query()

// disconnect from SnappyData
conn.close()
```

SnappyData Summary

PART 4

SnappyData Summary

In-Memory Computing SUMMIT EUROPE 2018

100% compatible with Spark

Spark and In-memory database integrated and simple

Unified with table and SQL

Well designed for faster performance

Thanks !

UL Systems, Inc.

SNAPPYDATA

Contact Information

mailto: info@ulsystems.co.jp

http://www.ulsystems.co.jp

twitter: @MasakiYamakawa