SOSCON

삼성 오픈소스 컨퍼런스
SAMSUNG OPEN SOURCE CONFERENCE

OPEN YOUR UNIVERSE WITH SOSCON

Extending Spark Streaming to Support Complex Event Processing

—

소속 삼성전자

이름 김병진, 권오찬

2015. 10. 27.

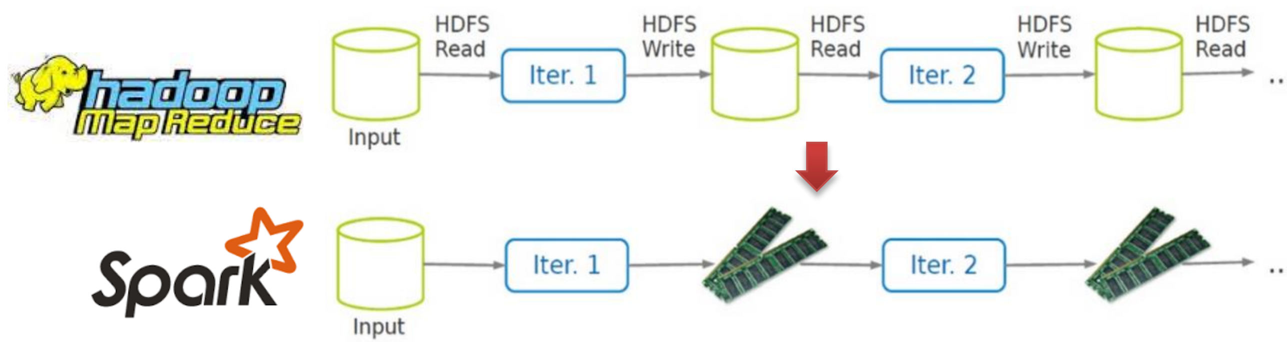
Agenda

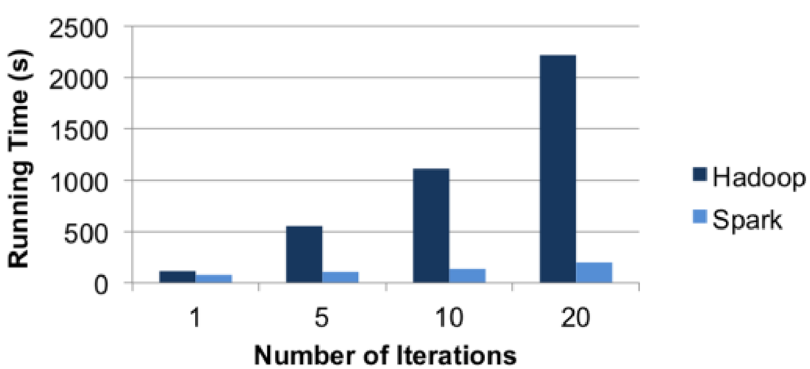
- Background
- Motivation
- Streaming SQL
- Auto Scaling
- Future Work

Background - Spark

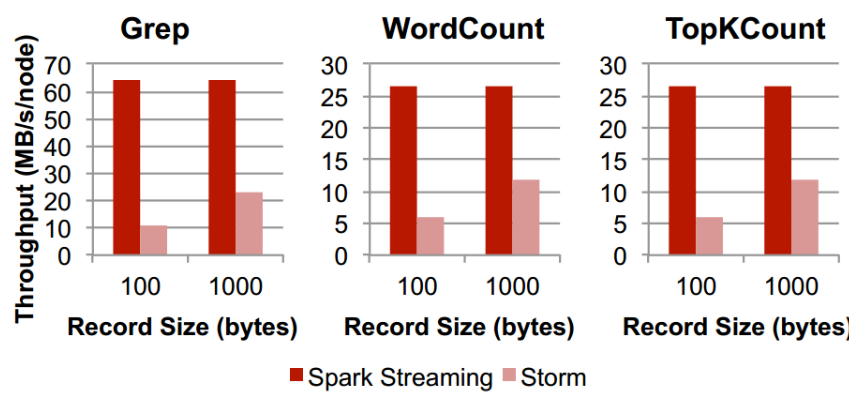
*RDD: Resilient Distributed Datasets

- **Apache Spark is a high performance data processing platform**
  - Use a distributed memory cache (RDD*)
  - Process batch and stream data in the common platform
  - Be developed by 700+ contributors





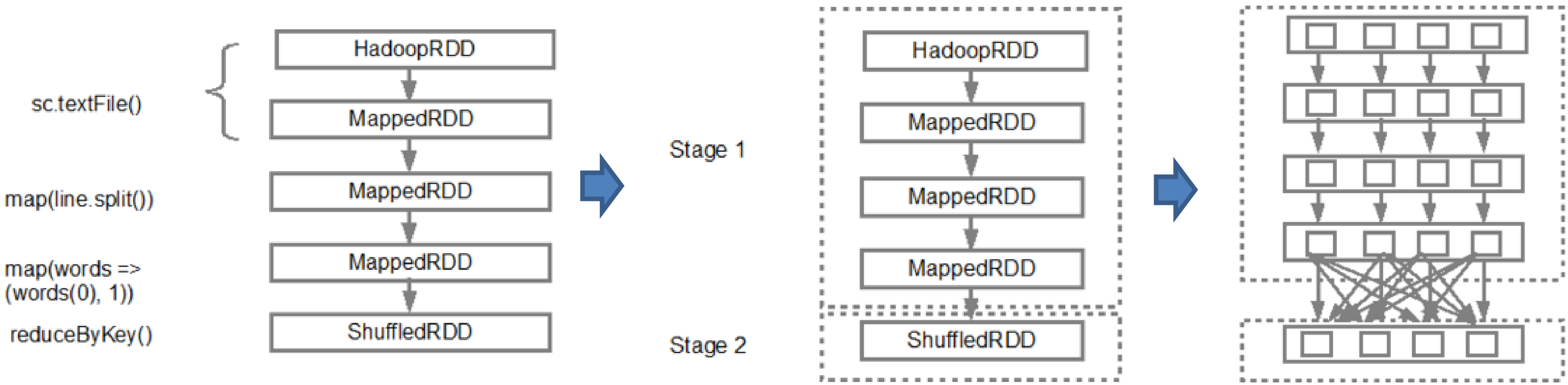
100x faster than Hadoop
(Batch, 50 Nodes, Clustering Alg.)



3x faster than Storm
(Stream, 30 Nodes)

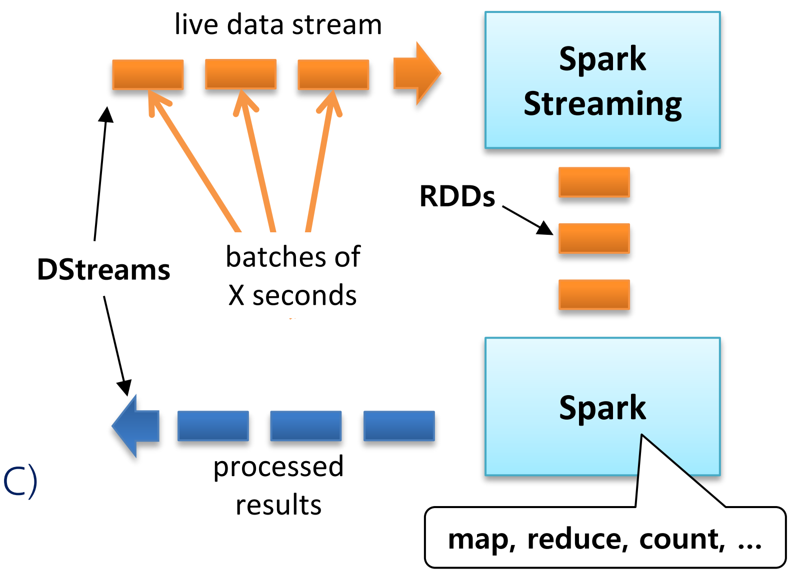
Background - Spark RDD

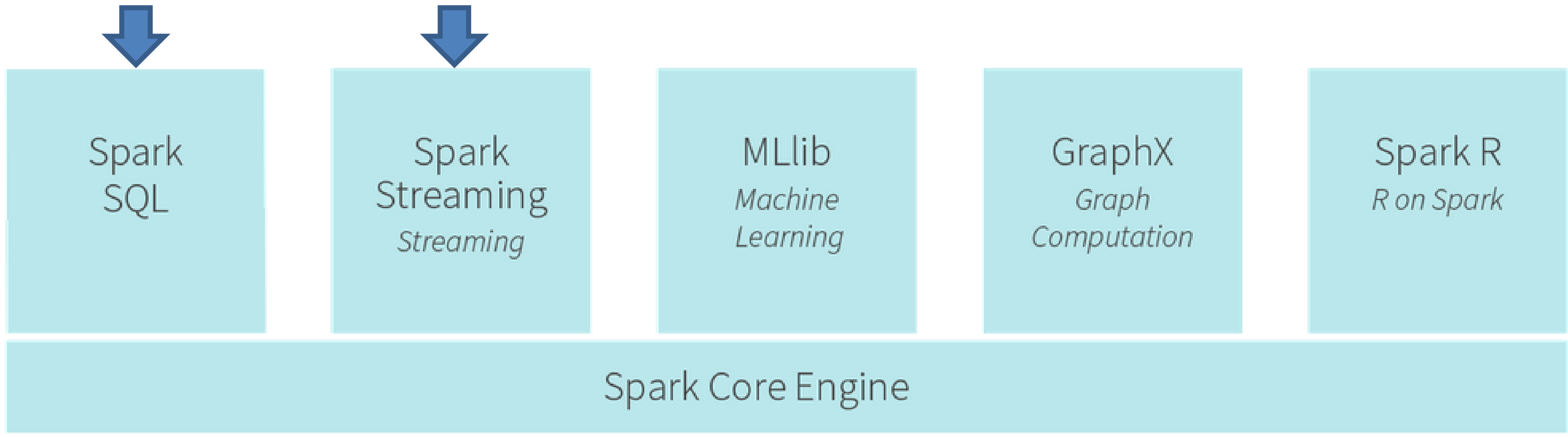
- **Resilient Distributed Datasets**
  - RDDs are immutable
  - Transformations are lazy operations which build a RDD's lineage graph
    - E.g.: map, filter, join, union
  - Actions launch a computation to return a value or write data
    - E.g.: count, collect, reduce, save
  - The scheduler builds a DAG of stages to execute
  - If a task fails, spark re-runs it on another node as long as its stage's parent are still available



Background - Spark Modules

- **Spark Streaming**
  - Divide stream data into micro-batches
  - Compute the batches with fault tolerance
- **Spark SQL**
  - Support SQL to handle structured data
  - Provide a common way to access a variety of data sources (Hive, Avro, Parquet, ORC, JSON, and JDBC)





Motivation - Support CEP in Spark

*Complex Event Processing

- **Current spark is not enough to support CEP***
  - Do not support continuous query language to process stream data
  - Do not support auto-scaling to elastically allocate resources

- **We have solved the issues:**
  - Extend Intel's Streaming SQL package
    - Improve performance by optimizing time-based windowed aggregation
    - Support query chains by implementing "Insert Into" queries
  - Implement elastic-seamless resource allocation
    - Can do auto-scale in/out

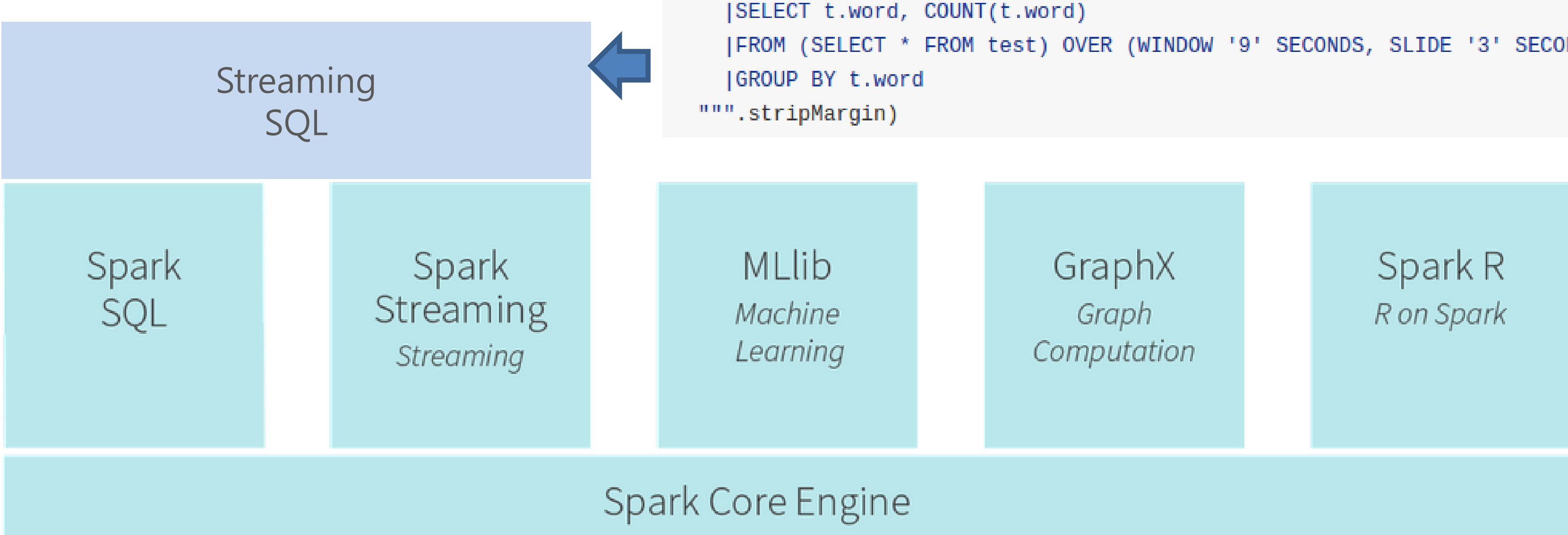
Streaming SQL

Streaming SQL - Intel

http://spark-packages.org/package/Intel-bigdata/spark-streamingsql

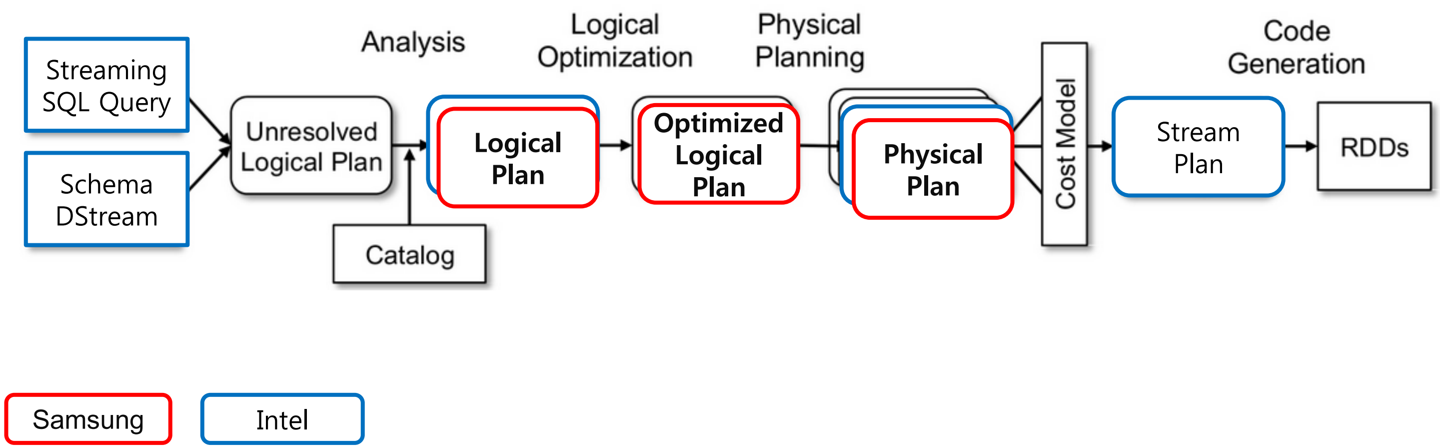
- **Streaming SQL is a third party library of Spark Packages**
    - Build on top of Spark Streaming and Spark SQL Catalyst
    - Manipulate stream data like static structured data in database
    - Process queries continuously
    - Support time based windowing join/aggregation queries

```
sql(
  """
    |SELECT t.word, COUNT(t.word)
    |FROM (SELECT * FROM test) OVER (WINDOW '9' SECONDS, SLIDE '3' SECONDS) AS t
    |GROUP BY t.word
  """.stripMargin)
```



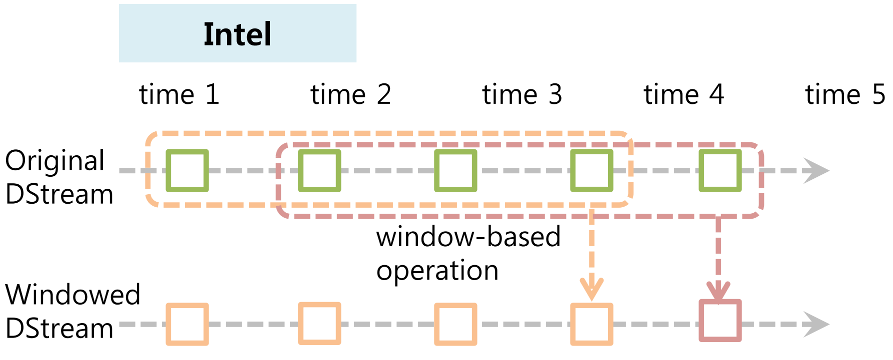
Streaming SQL - Plans

- **Logical Plan**
  - Modify streaming SQL optimizer
  - Add windowed aggregate logical plan
- **Physical Plan**
  - Add windowed aggregate physical plan to call new **WindowedStateDStream** class
  - Implement new expression functions (Count, Sum, Average, Min, Max, Distinct)
  - Develop **FixedSizedAggregator** for efficiency which is the concept of IBM InfoSphere Streams

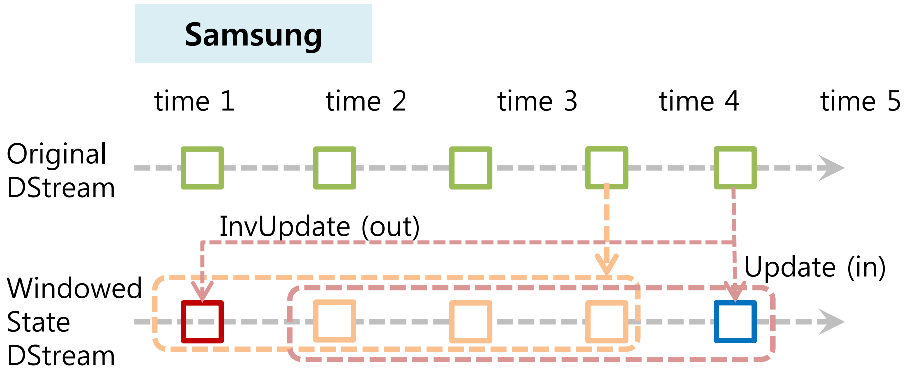


Streaming SQL - Windowed State

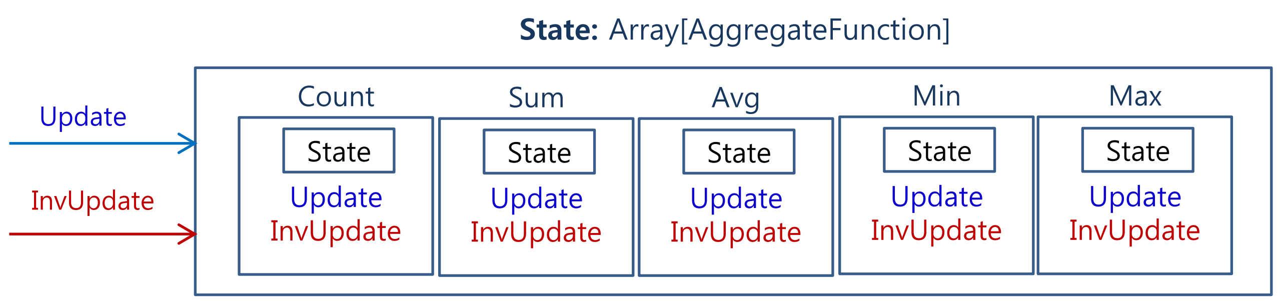
- **WindowedStateDStream class**
  - Modify StateDStream class to support windowed computing
  - Add inverse update function to evaluate old values
  - Add filter function to remove obsolete keys



Compute all elements in the window



Compute only Delta (old and new elements)



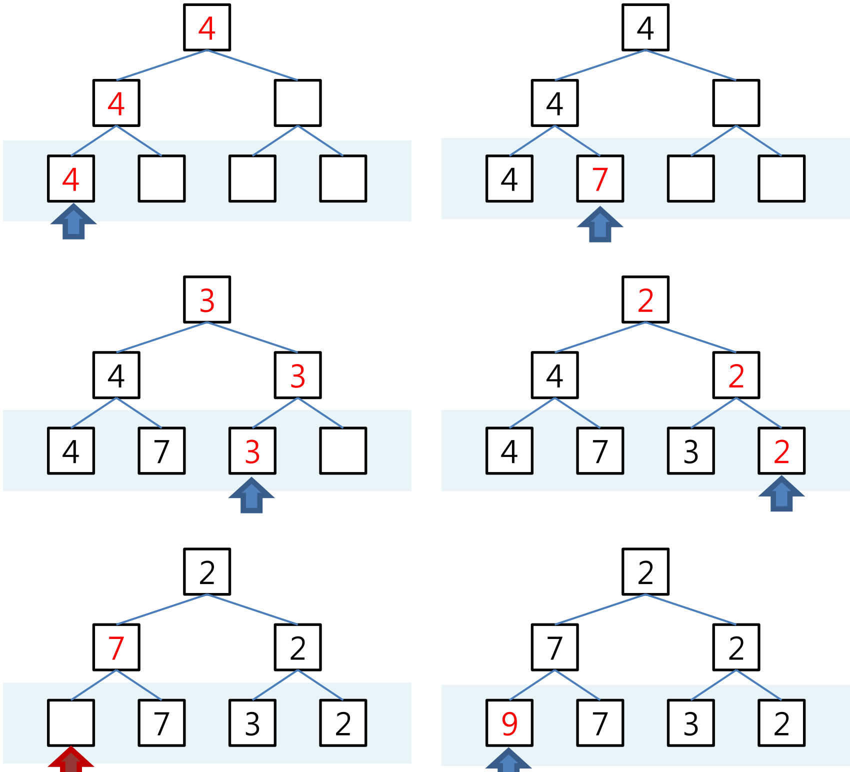
Streaming SQL - Fixed Sized Aggregator

*K. Tangwongsan, M. Hirzel, S. Schneider, K-L. Wu, "General incremental sliding-window aggregation," In VLDB, 2015.

- **Fixed-Sized Aggregator* of IBM InfoSphere Streams**
  - Use fixed sized binary tree
  - Maintain leaf nodes as a circular buffer using front and back pointers
  - Very efficient for non-invertable expressions (e.g. Min, Max)

- **Example - Min Aggregator**

Event	Window's Contents	FlatFAT's Slots $a[1]$	$a[2]$	$a[3]$	$a[4]$
1) 4 arrives	4	F 4 B	⊥	⊥	⊥
2) 7 arrives	4, 7	F 4	7 B	⊥	⊥
3) 3 arrives	4, 7, 3	F 4	7	3 B	⊥
4) 2 arrives	4, 7, 3, 2	F 4	7	3	2 B
5) 4 leaves	7, 3, 2	⊥	F 7	3	2 B
6) 9 arrives	7, 3, 2, 9	9 B	F 7	3	2



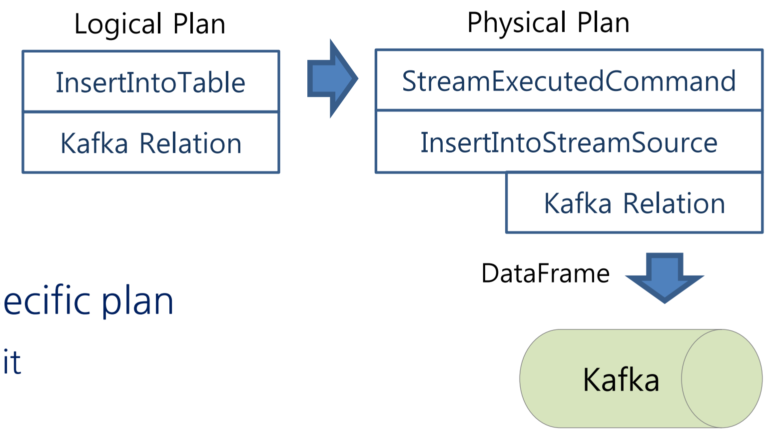
- **Windowed Aggregate Functions**
  - Add invUpdate function
  - Reduce objects for efficient serialization
  - Implement Fixed-Sized Aggregator for Min, Max functions

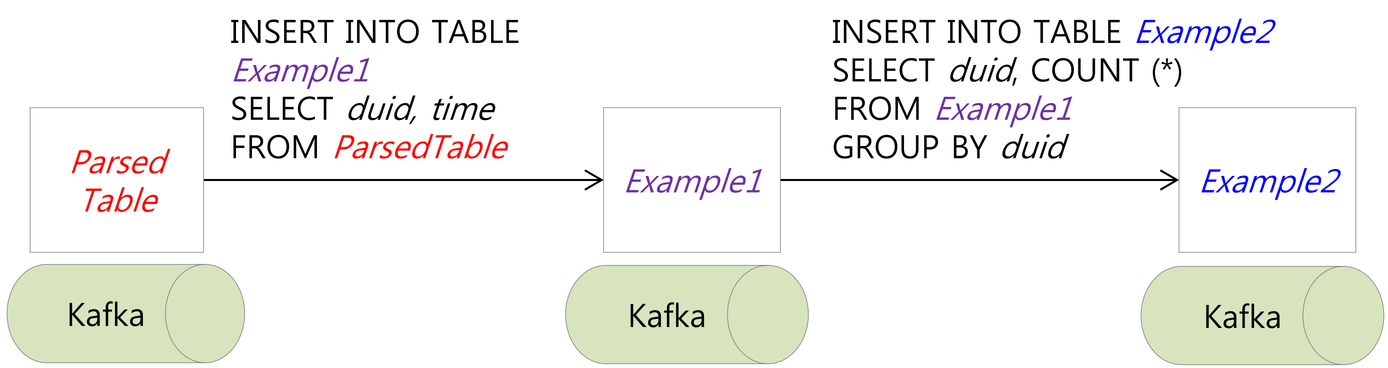
Aggregate Functions	State	Update	InvUpdate
Count	countValue: Long	Increase countValue	Decrease countValue
Sum	sumValue: Any	Add to sumValue	Subtract to sumValue
Average	sumValue: Any countValue: Long	Increase countValue Add to sumValue	Decrease countValue Subtract to sumValue
Min	fat: FixedSizedAggregator	Insert minimum to fat	Remove the oldest from fat
Max	fat: FixedSizedAggregator	Insert maximum to fat	Remove the oldest from fat
CountDistinct	distinctMap: mutable.HashMap[Row, Long]	Increase count of distinct value in map	Decrease count of distinct value in map

Streaming SQL - Insert Into

- **Support "Insert Into" query**
  - Implement Data Sources API
    - Implement the insert function in Kafka relation
    - Modified some physical plans to support streaming
  - Modified physical planning strategies to assign a specific plan
    - Convert current RDD to a DataFrame and then insert it
  - Support query chaining

    → The result a query can be reused by multiple queries





Streaming SQL - Evaluation

- **Experimental Environment**
  - Processing Node: Intel i5 2.67GHz, 4 Cores, 4GB per node

    Spark Cluster: 7 Nodes

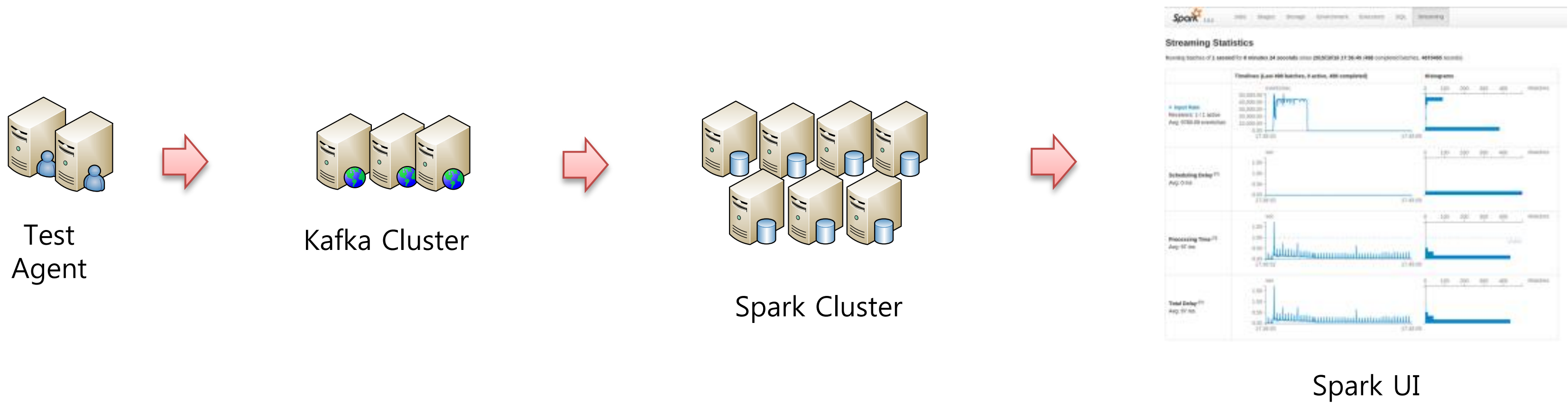
    Kafka Cluster: 3 Nodes
  - Test Query:

    SELECT *t.word*, COUNT(*t.word*), SUM(*t.num*), AVG(*t.num*), MIN(*t.num*), MAX(*t.num*)

    FROM (SELECT * FROM *t_kafka*) OVER (WINDOW *'x'* SECONDS, SLIDE '1' SECONDS) AS *t*

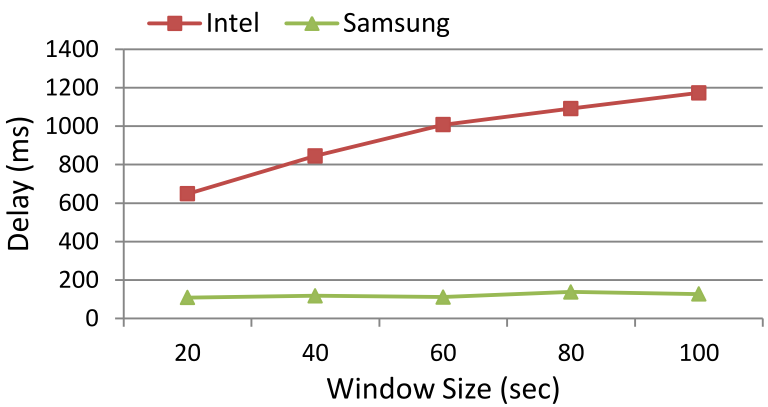
    GROUP BY *t.word*
  - Default Set:

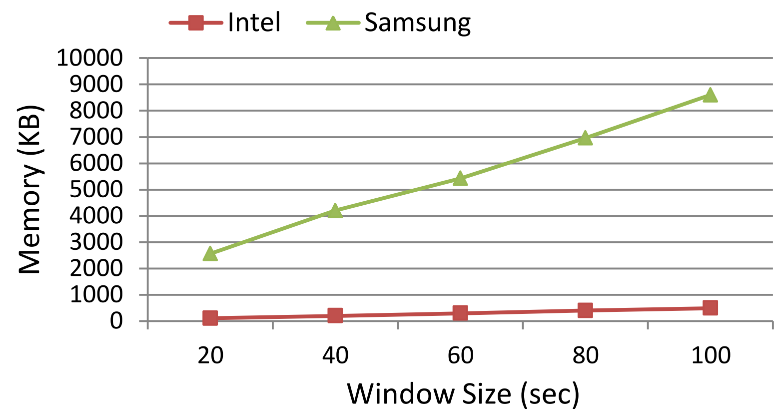
    100 EPS, 100 Keys, 20 sec Window, 1 sec Slide, 10 Executors, 10 Reducers

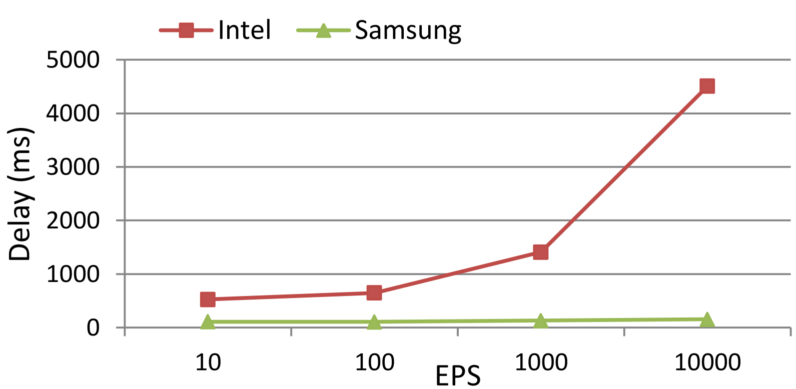


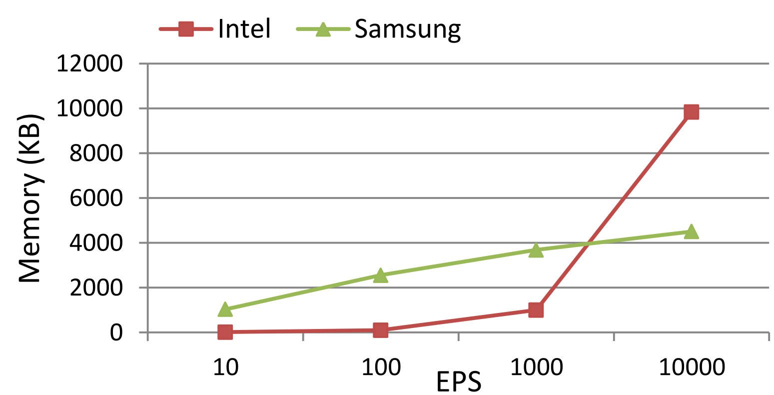
Streaming SQL - Evaluation

- **Test Result**
  - Show low processing delays despite of heavy event loads or large-sized windows
  - Need memory optimization





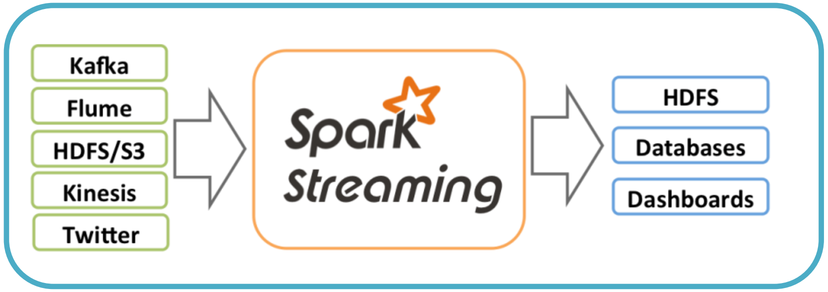


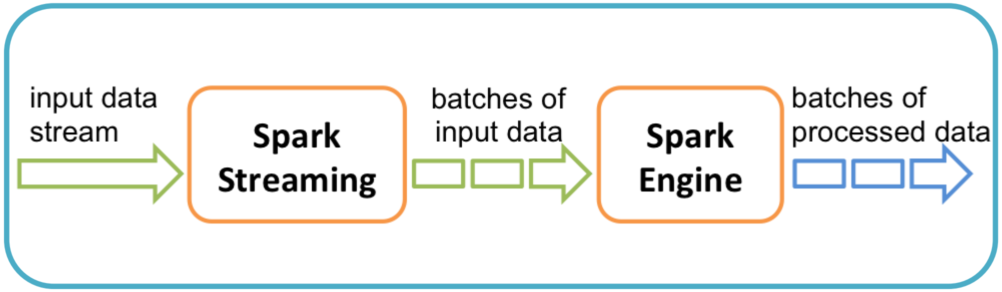


Auto Scaling

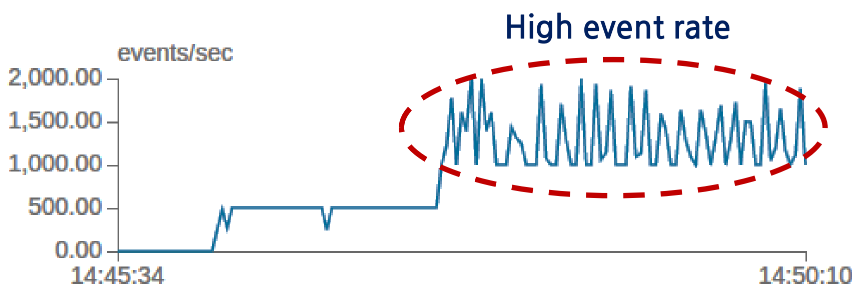
Time-varying Event Rate in Real World

- **Spark Streaming**
  - Data can be ingested from many sources like Kafka, Flume, Twitter, etc.
  - Live input data streams are divided into batches, and they are processed

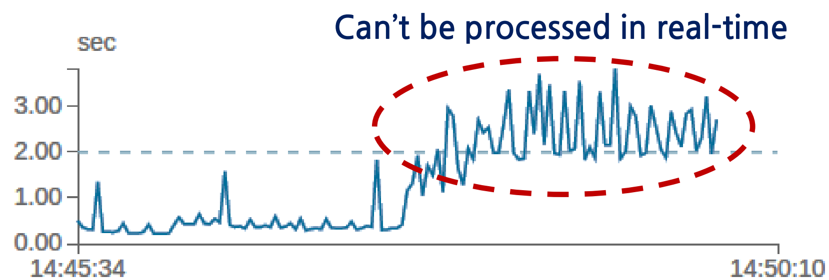




- **In streaming application, event rate may change frequently over time**
- **How can this be dealt with?**



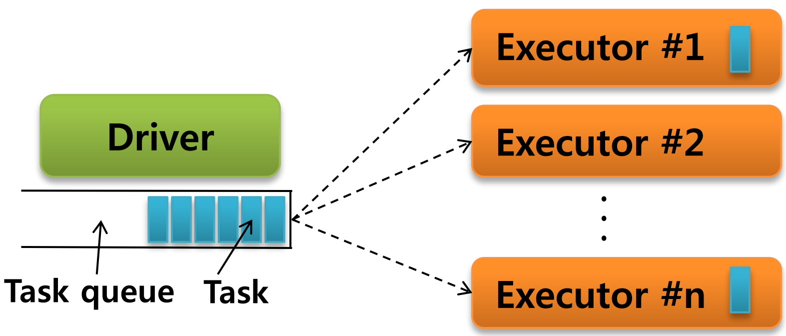
〈 Event Rate 〉



〈 Batch Processing Time〉

Spark AS-IS

- **Spark currently supports dynamic resource allocation on Yarn (SPARK-3174) and coarse-grained Mesos (SPARK-6287)**



- **Upper/lower bound for the number of executors**
  - spark.dynamicAllocation.minExecutors
  - spark.dynamicAllocation.maxExecutors
- **Scale-out condition**
  - schedulerBacklogTimeout < staying time of a task in task queue
- **Scale-in condition**
  - executorIdleTimeout < staying time of an executor in idle state

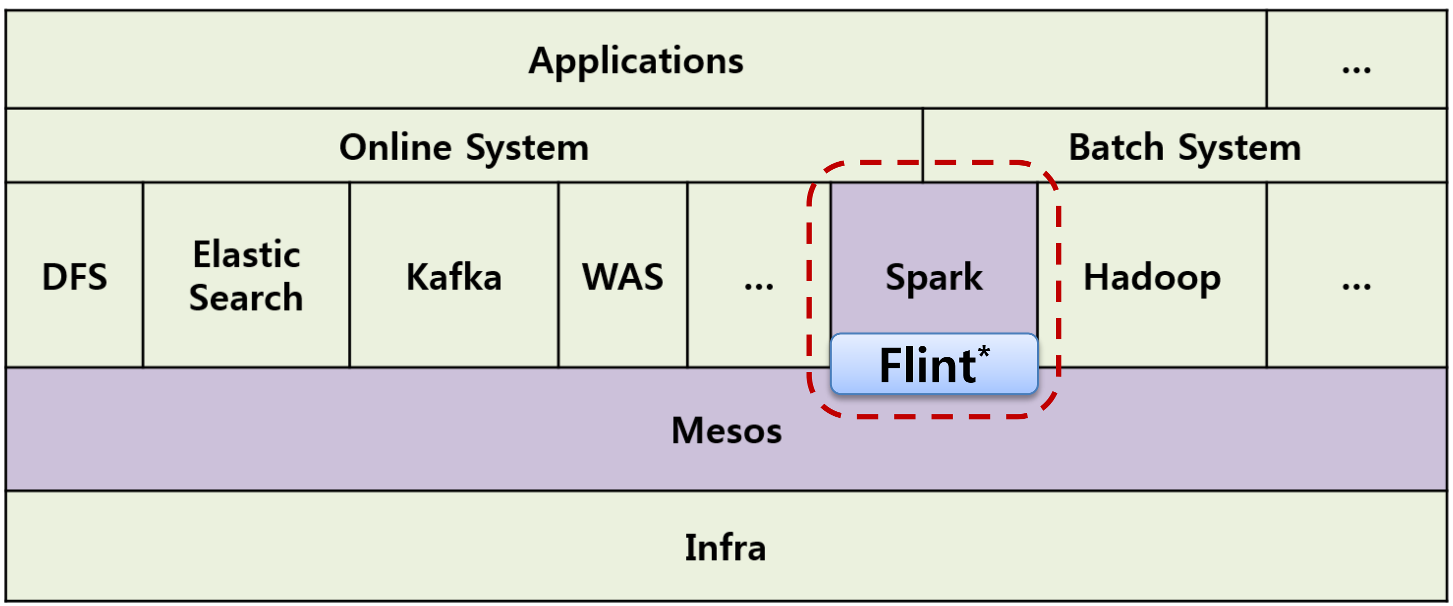
  - Existing Dynamic Resource Allocation is not optimized for streaming
    - Even though event rate becomes smaller, executors may not be removed due to scheduling policy
  - It is difficult to determine appropriate configuration parameters

- **Backpressure (SPARK-7398)**
  - Enable the Spark streaming to control the receiving rate dynamically for handling bursty input streams
  - Not real-time

Elastic-seamless Resource Allocation

- **Goal**
  - Allocate resources to streaming applications dynamically as the rate of incoming events varies over time
  - Enable the applications to meet real-time deadlines
  - Utilize the resource efficiently

- **Cloud Architecture**



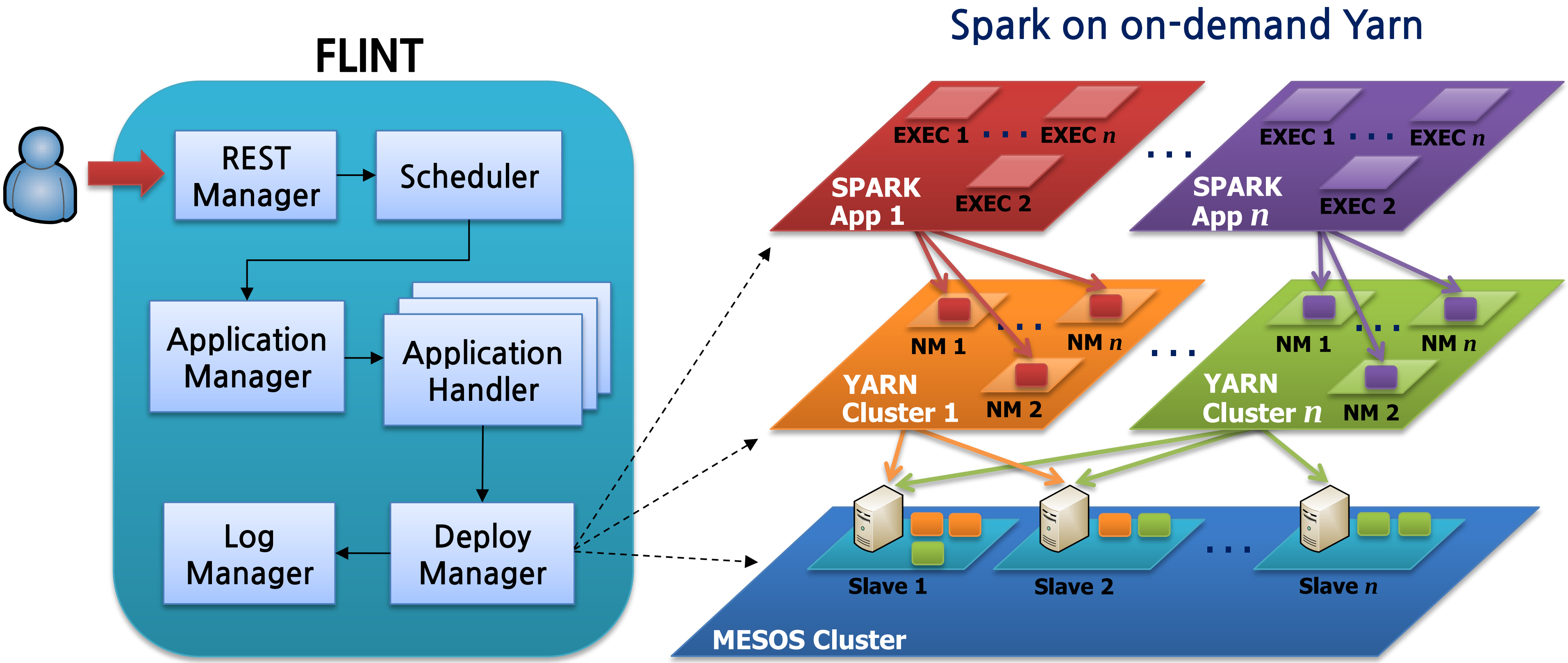
*Flint: our spark job manager

Spark Deployment Architecture

- **Spark currently supports three cluster managers**
  - Standalone, Apache Mesos, Hadoop Yarn

- **Spark on Mesos**
  - Fine-grained mode
    - Each Spark task runs as a separate Mesos task.
    - Launching overhead is big, so it is not suitable for streaming.
  - Coarse-grained mode
    - Launch only one long-running Spark task on each Mesos machine
      - Cannot scale-out more than the number of Mesos machines
    - Cannot control executor resource
      - Only available for total resource
  - Both of two modes have some problems to achieve our goal

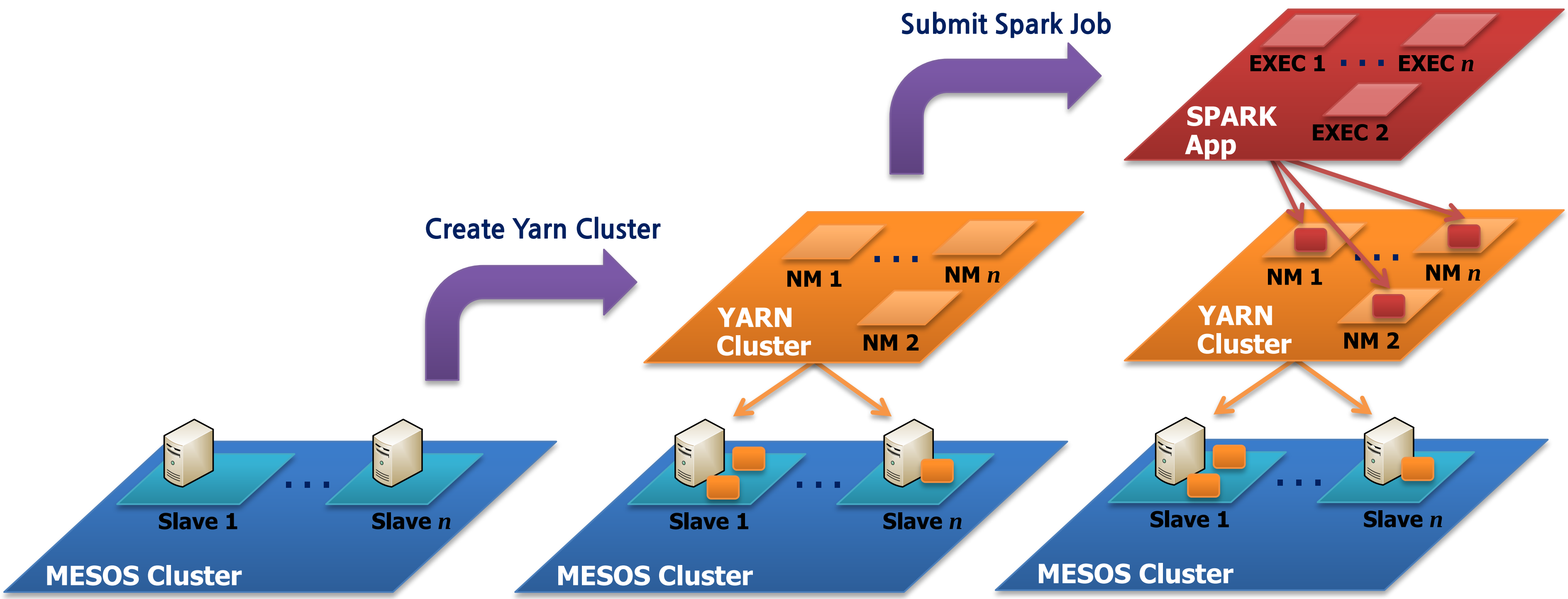
Flint Architecture Overview



- Marathon, Zookeeper, ETCD, HDFS

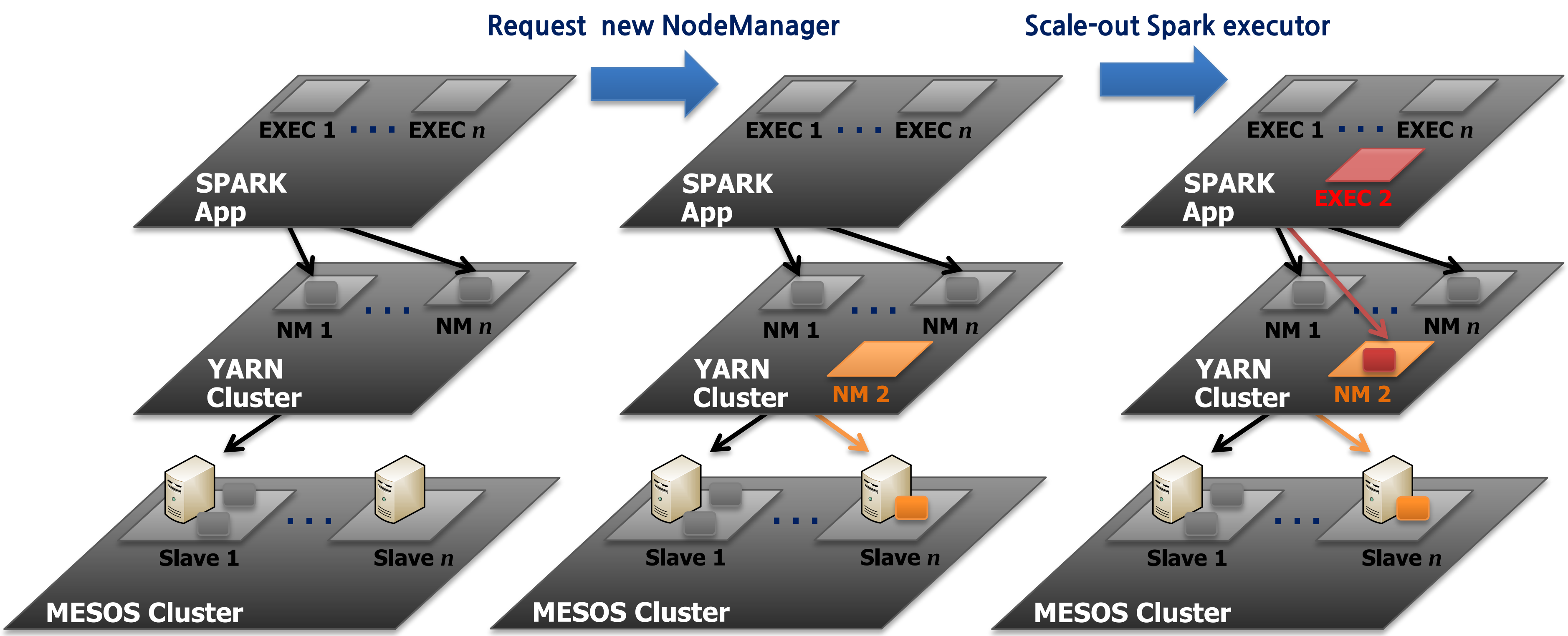
Flint Architecture: Job Submission

- **Job submission process**
  1. Request to deploy dockerized YARN via Marathon
  2. Launch ResourceManager, NodeManagers for Spark driver/executors
  3. Check ResourceManager status
  4. Submit Spark Job and check Job status
  5. Watch Spark driver status and get Spark endpoint



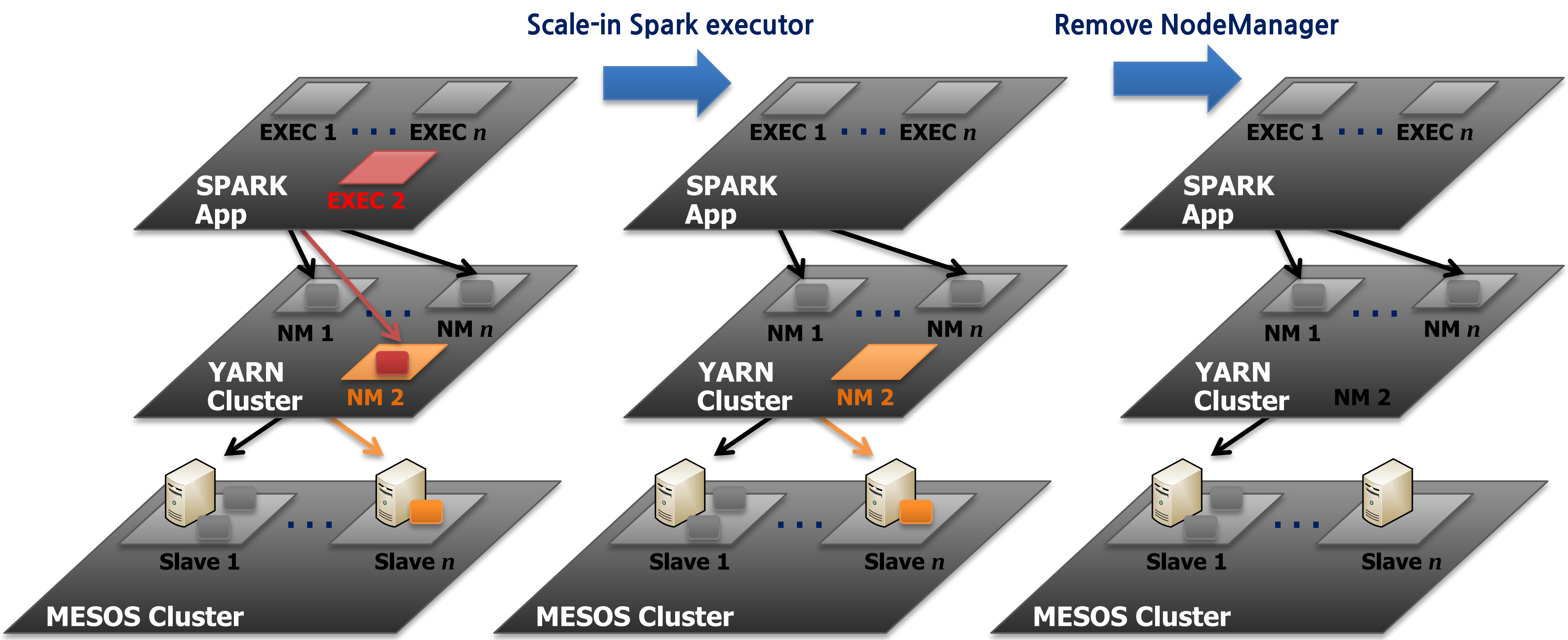
Flint Architecture: Scale-out

- **Scale-out Process**
  1. Request to increase the instance of NodeManager via Marathon
  2. Launch new NodeManager
  3. Scale-out Spark executor



Flint Architecture: Scale-in

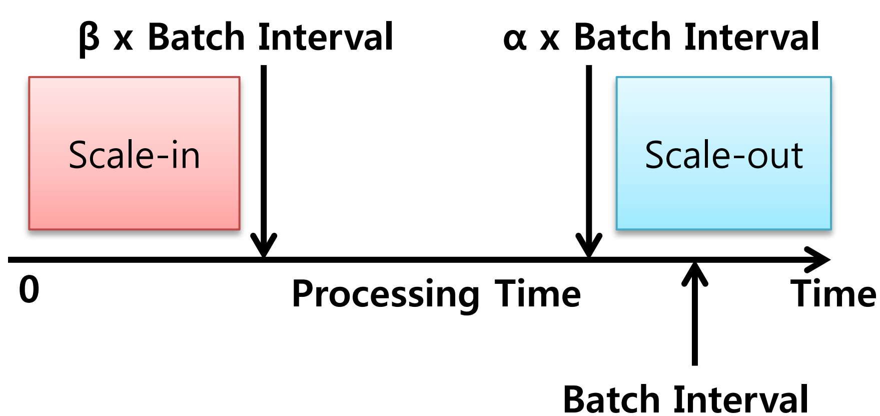
- **Scale-in Process**
  1. Get Executor's info and select spark victims
  2. Inactivate spark victims and kill them after *n* x batch interval
  3. Get Yarn victims and decommission NodeManager via ResourceManager
  4. Get mesos task id of Yarn victims and kill mesos tasks of victims



Flint Architecture: Auto-scaling

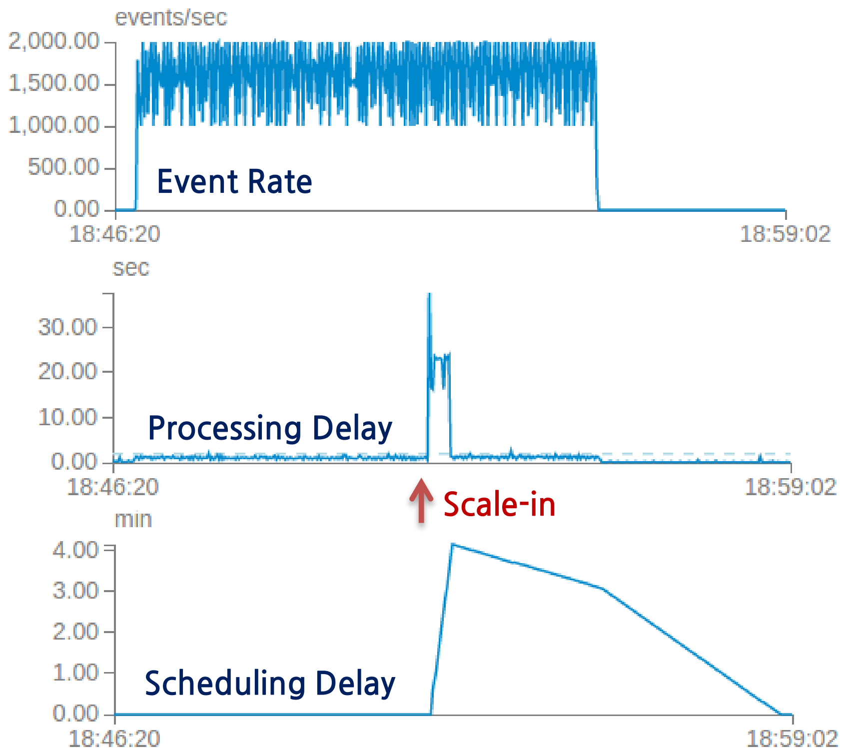
- **Auto-scaling Mechanism**
  - Real-time constraint
    - Batch processing time < Batch interval

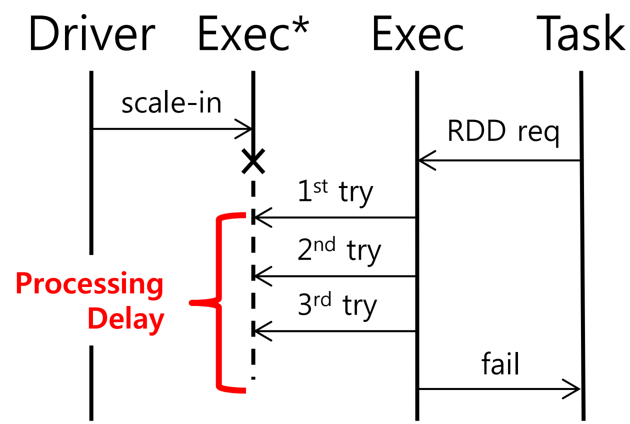
  - **Scale-out condition**
    - α x batch interval < batch processing delay
  - **Scale-in condition**
    - β x batch interval > batch processing delay ($0 < \beta < \alpha \leq 1$)



Spark Issues: Scale-in (1/3)

- **Timeout & Retry for Killed Executor**
  - Although Spark executor is killed by admin, Spark re-tries to connect it.

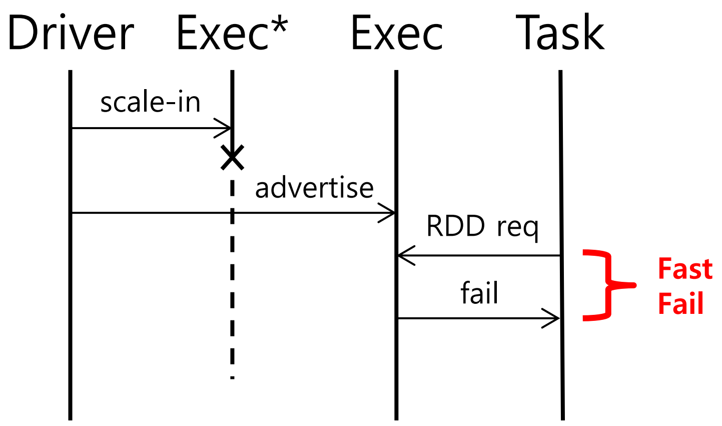




- spark.shuffle.io.maxRetries = 3
- spark.shuffle.io.retryWait = 5s

Spark Issues: Scale-in (2/3)

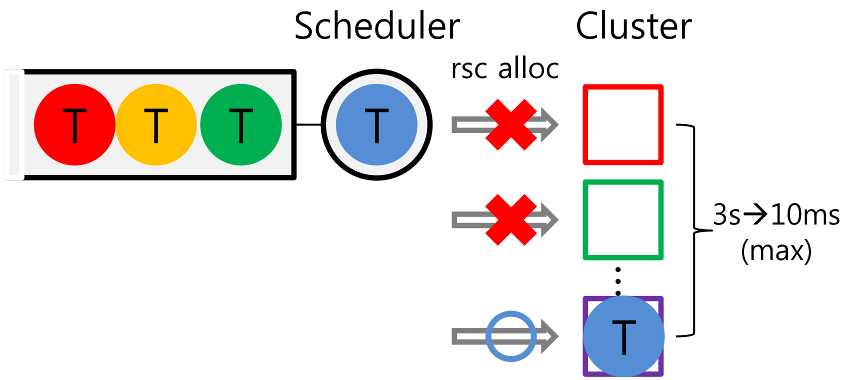
- **Timeout & Retry for Killed Executor**
  - Advertisement for killed executor



  - Inactivate executor before scale-in
    - Inactivate executor and kill them after n x batch interval
    - During inactivating stage, the executor does not receive any task from driver.

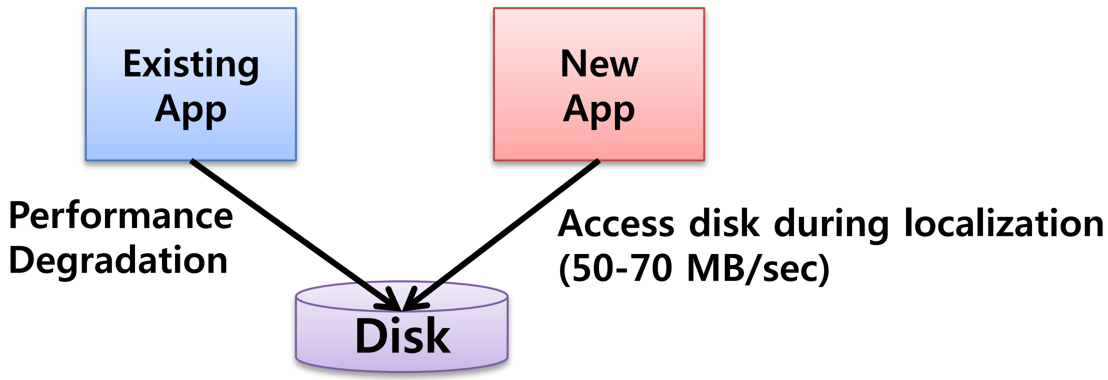
Spark Issues: Scale-in (3/3)

- **Data Locality**
  - Spark scheduler consider data locality
    - Processing/Node/Rack locality
    - spark.locality.wait = 3s
  - However, waiting time for locality is big burden to streaming applications
    - The waiting time should be much less than batch interval for streaming
    - Otherwise, streaming may not be processed in real-time
  - Thus, Flint overrides "spark.locality.wait" to very small value if application type is streaming



Spark Issues: Scale-out (1/2)

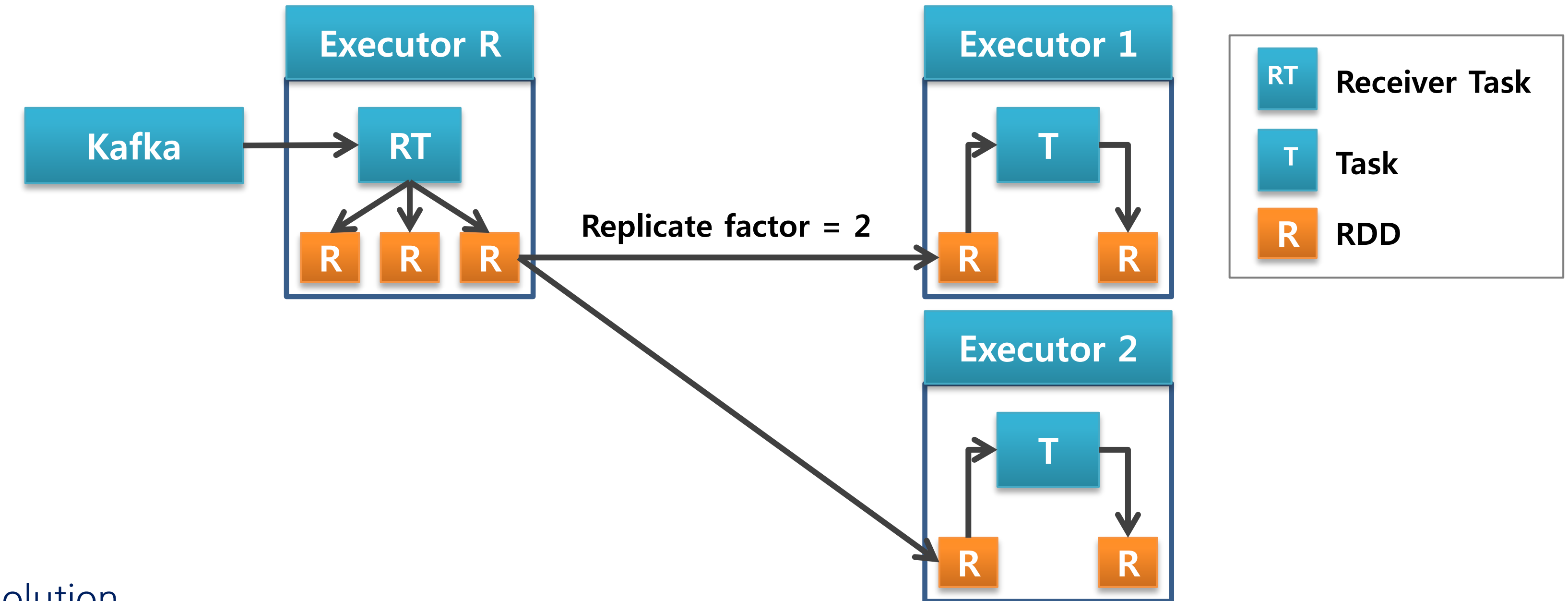
- **Data Localization**
  - During scale-out process, new Yarn container performs to localize some data (e.g. spark jar, application jar)
  - Data localization incurs high disk I/O, so



  - Solution
    - Prefetch if possible
    - Disk isolation, SSD

Spark Issues: Scale-out (2/2)

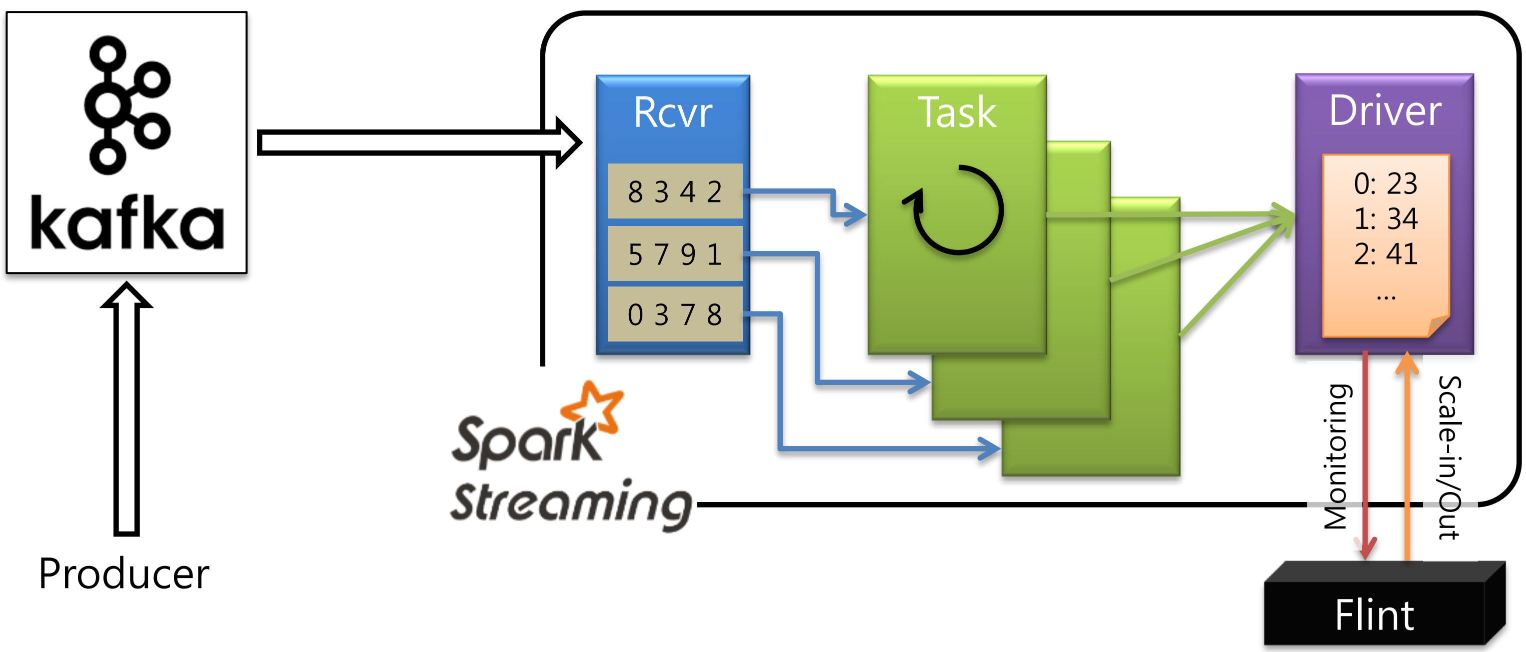
- **RDD Replication**
  - When a new executor is added, a receiver requests to replicate received RDD blocks to the executor
  - New executor does not ready to receive RDD blocks during an initialization
  - At this situation, the receiver waits until new executor is ready



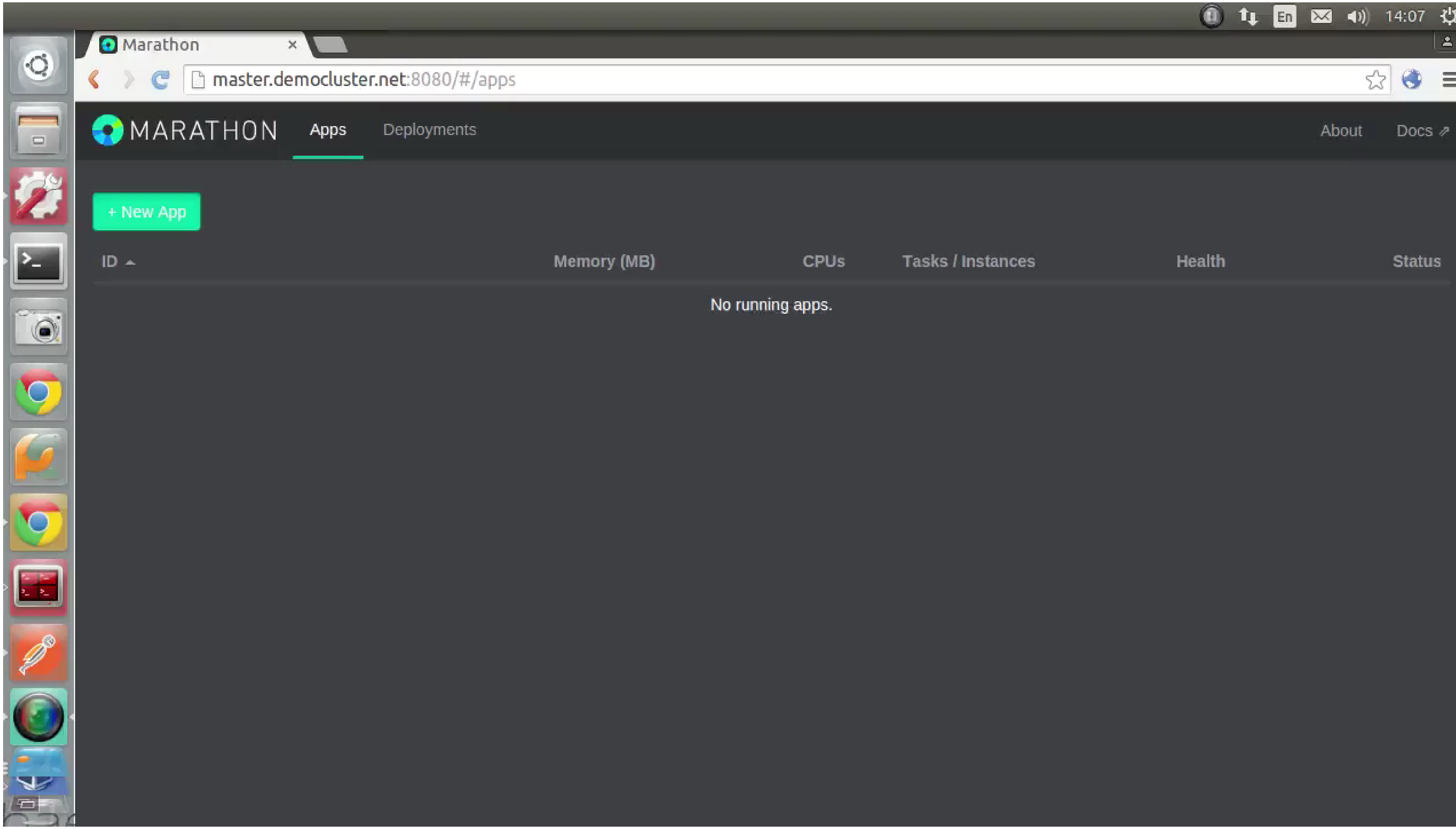
  - Solution
    - A receiver does not replicate RDD blocks to new executor during initialization
    - E.g., spark.blockmanager.peer.bootstrap = 10s

Demo (1/2)

- **Demo Environment**
  - Data source: Kafka
  - Auto-scaling parameter
    - α=0.4, β=0.9
  - SQL query
    - SELECT t.word, COUNT(DISTINCT t.num), SUM(t.num), AVG(t.num), MIN(t.num), MAX(t.num) FROM (SELECT * FROM t_kafka) OVER (WINDOW 300 SECONDS, SLIDE 3 SECONDS) AS t GROUP BY t.word



Demo (2/2)


14:07
Marathon
master.democluster.net:8080/#/apps
MARATHON
Apps
Deployments
About
Docs
+ New App
ID
Memory (MB)
CPUs
Tasks / Instances
Health
Status
No running apps.

Future Work

- **Streaming SQL**
  - Apply Tungsten Framework
    - Small-sized object, code gen, GC free memory management
  - Share RDDs
    - Map stream tables to RDD

- **Auto Scaling**
  - Support to dynamically allocate resources for batch (non-streaming) applications
  - Support unified schedulers for heterogeneous applications
    - Batch, streaming, ad-hoc

THANK YOU!

https://github.com/samsung/spark-cep