Continuously Optimizing Storage of Stream Histories

Lessons learned with Apache Druid

DEVIEW 2018

Jihoon Son

Imply

Outline

DEVIEW 2018

1. Introduction to Apache Druid
2. Optimizing Data Storage
3. Automatic Segment Compaction
4. Is Everything Going to be OK?
5. Towards Continuous Compaction

1.
Introduction to Apache Druid

DEVIEW 2018

What is Apache Druid?

DEVIEW 2018

High performance distributed analytics data store

What is Apache Druid?

DEVIEW 2018

High performance distributed analytics data store

- High ingest rates
- Low query latency

What is Apache Druid?

High performance distributed analytics data store

- Deployed in clusters
- Typically 10s–100s of nodes

What is Apache Druid?

DEVIEW 2018

High performance distributed analytics data store

- Optimized for analytics workloads

What is Apache Druid?

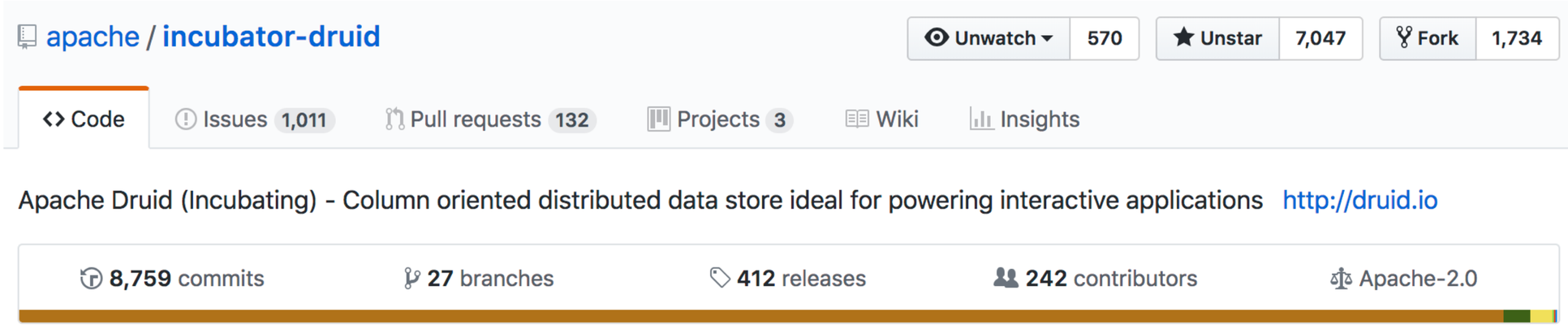
High performance distributed analytics data store

- The cluster stores a copy of your data

Brief History of Druid

DEVIEW 2018

- Introduced by Metamarkets in 2011
- Open-sourced in 2012
- Incubated to Apache in February, 2018
- Released 0.12.3 in September, 2018
- 0.13.0 will be available soon!

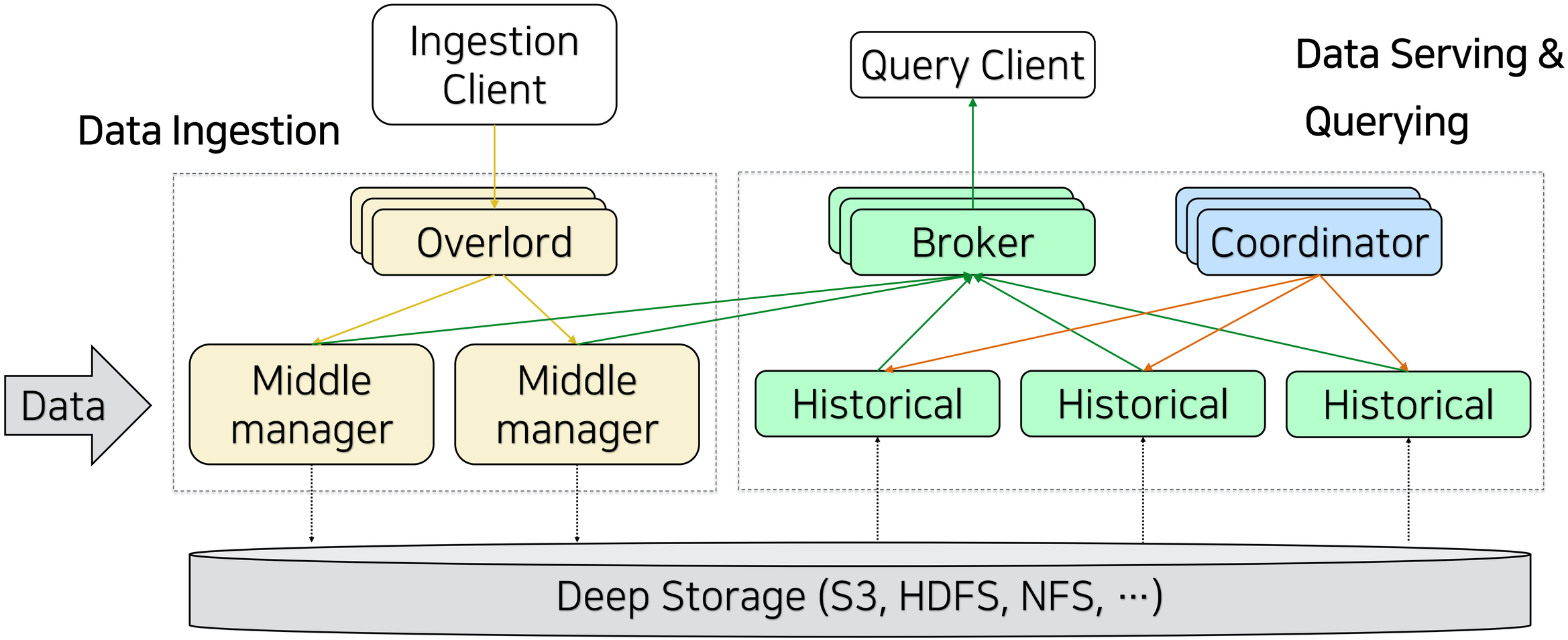


Powered by Druid

Source: http://druid.io/druid-powered.html

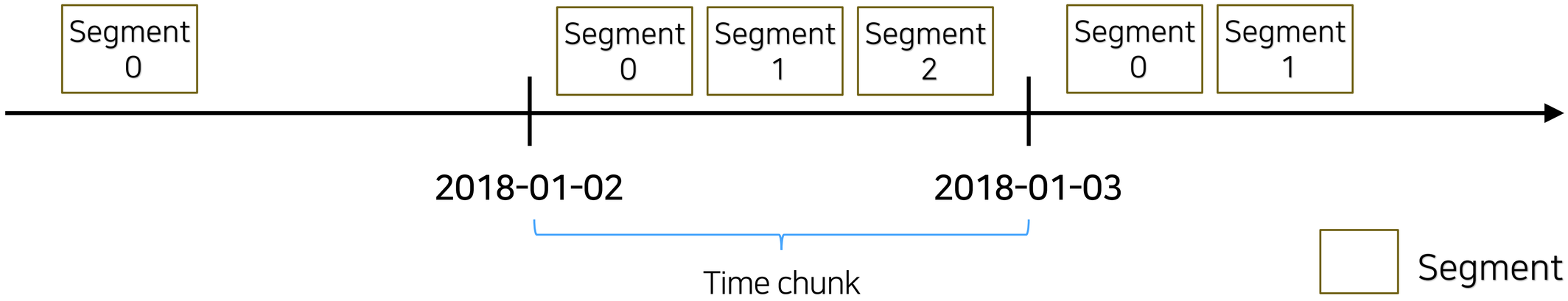
Not an endorsement

Architecture Overview



Datasource

- Similar to tables in a traditional RDBMS
- A datasource is primarily partitioned into *time chunks*
- A time chunk is partitioned into one or multiple *segments*
- Each segment is a single file



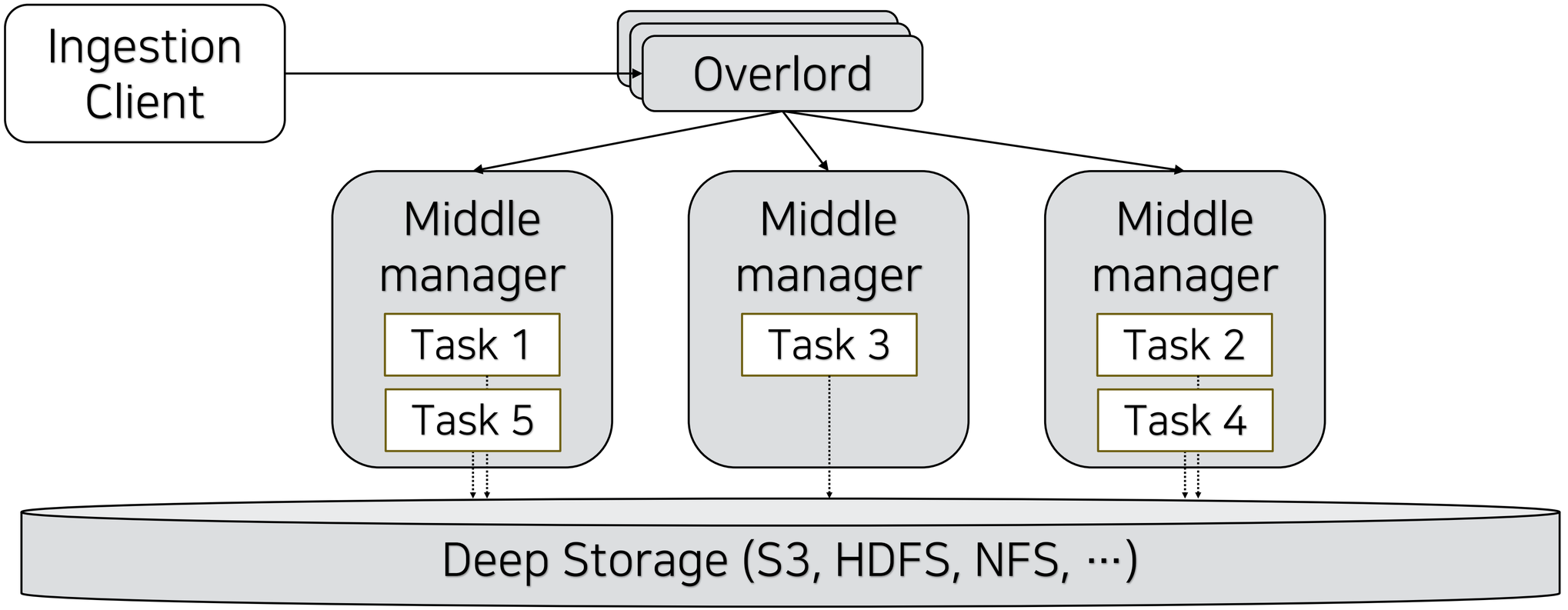
Data Ingestion

DEVIEW 2018

- Batch ingestion: native batch indexing, Hadoop/spark batch indexing
- Stream ingestion: Kafka, Flink, Spark streaming, …

Indexing Service

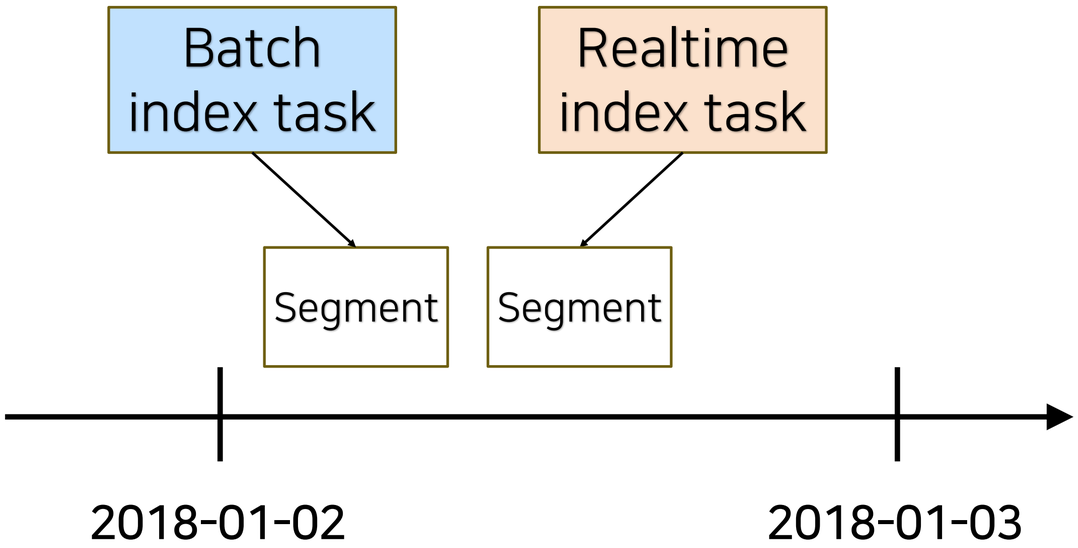
- Distributed service that runs *indexing tasks*
- Tasks read input data and generate segments



Indexing Service

DEVIEW 2018

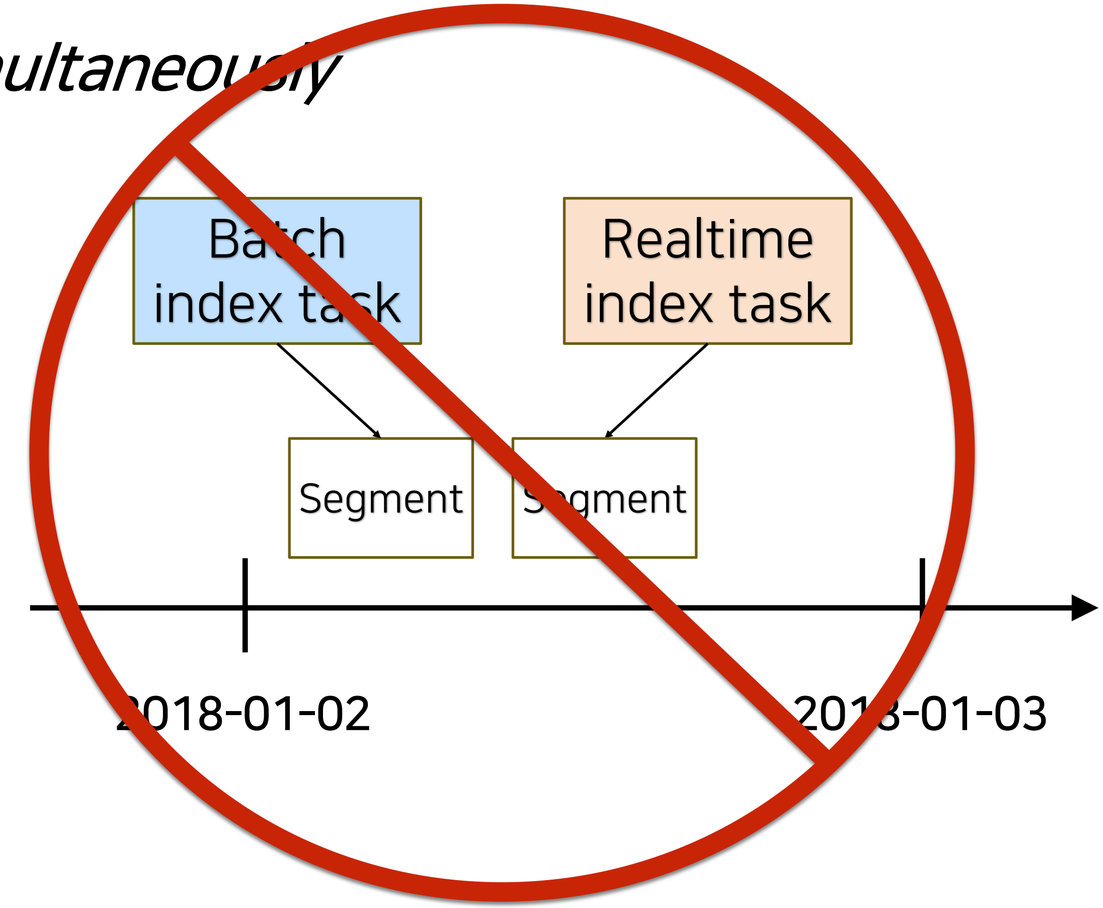
- Guarantees that *no two tasks write segments into the same time chunk of the same datasource simultaneously*



Indexing Service

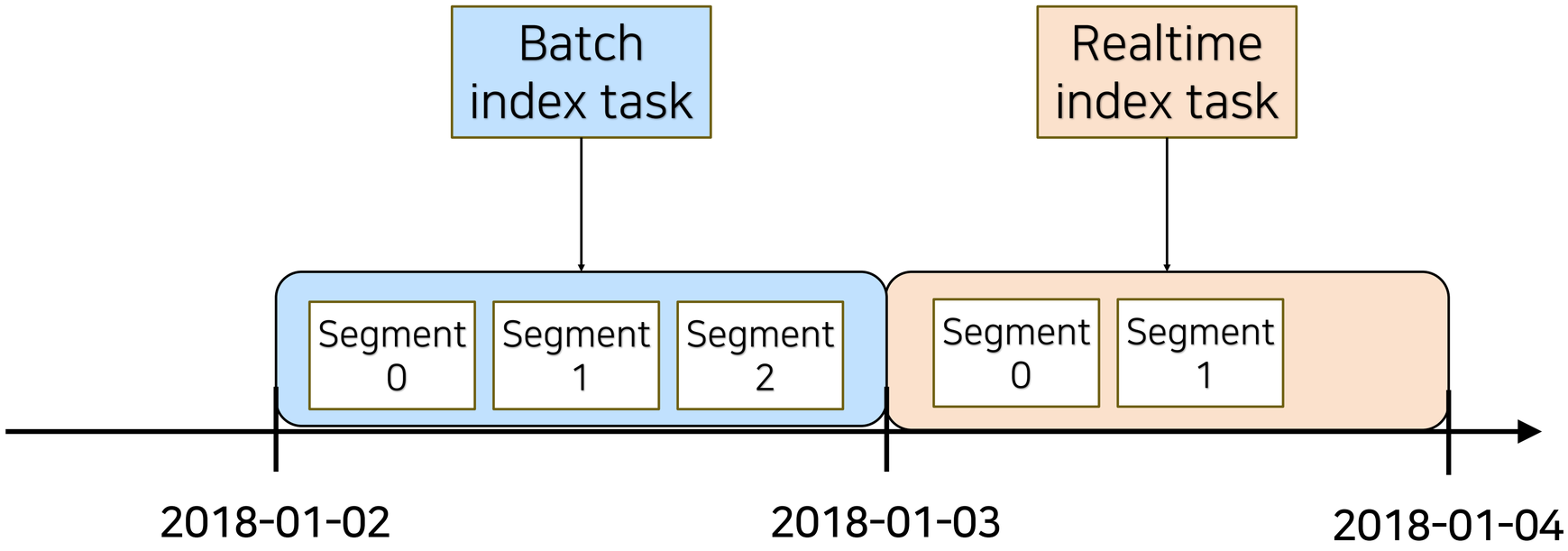
DEVIEW 2018

- Guarantees that *no two tasks write segments into the same time chunk of the same datasource simultaneously*



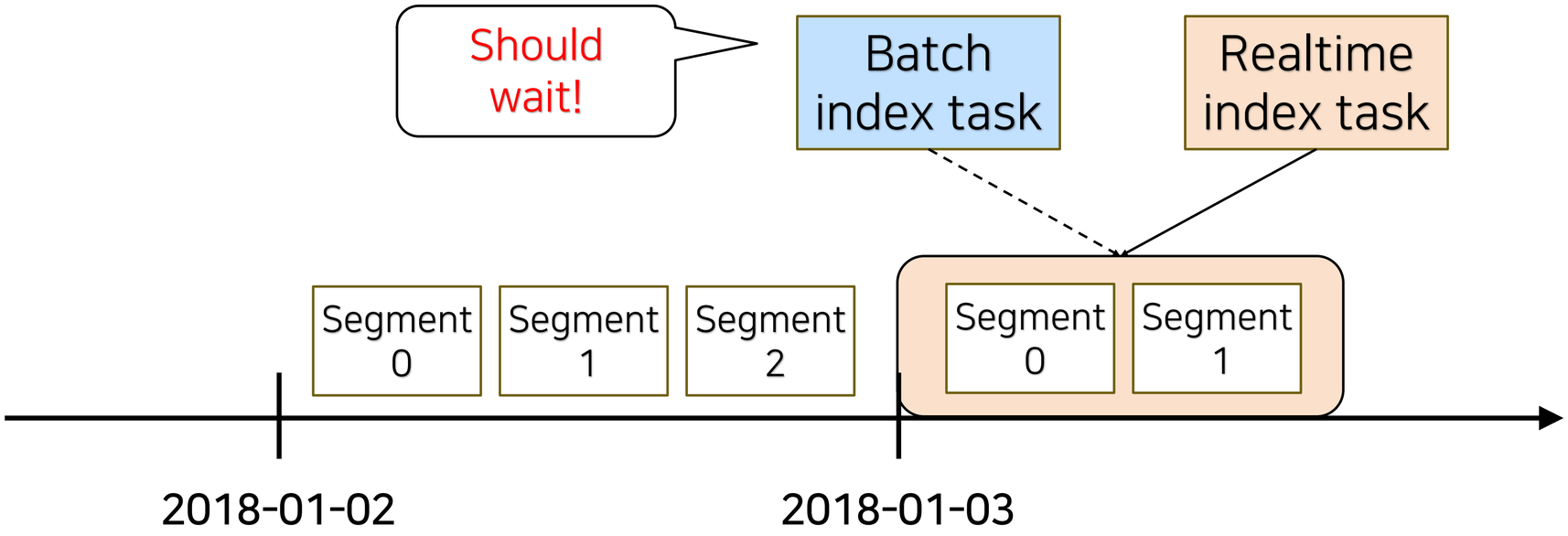
Time Chunk Locking

- A task should get a lock for a time chunk before reading/writing any segment



Time Chunk Locking

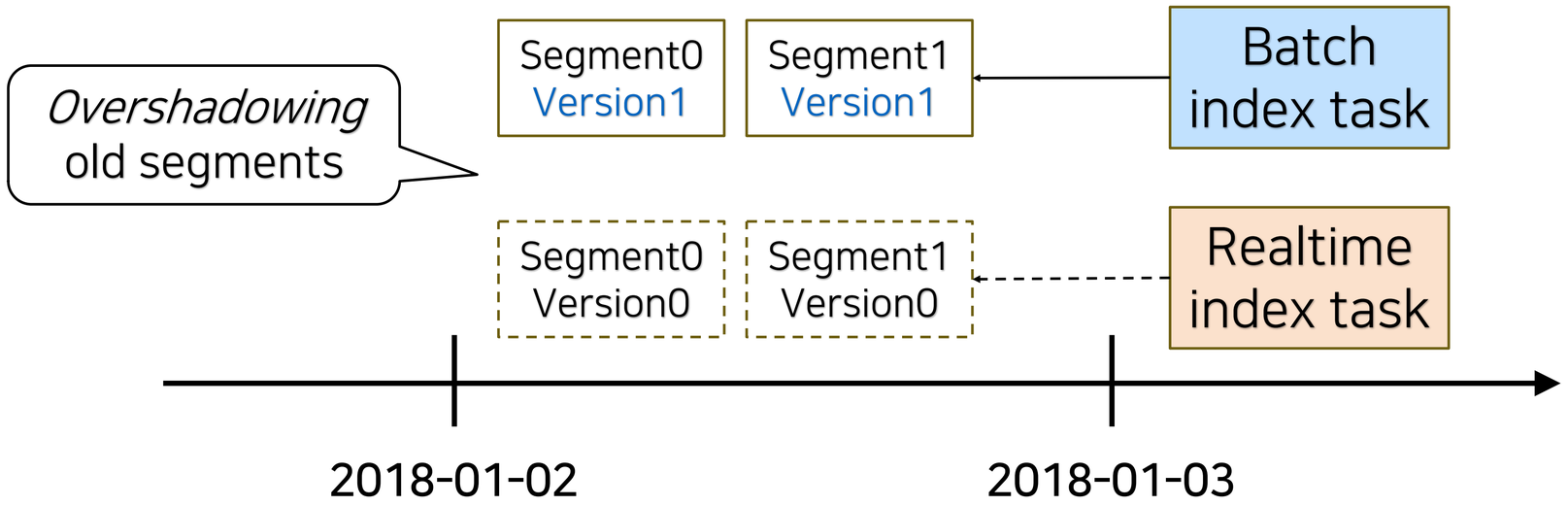
- If two or more tasks request a lock for the same time chunk, only one of them gets a lock
- Other tasks should wait



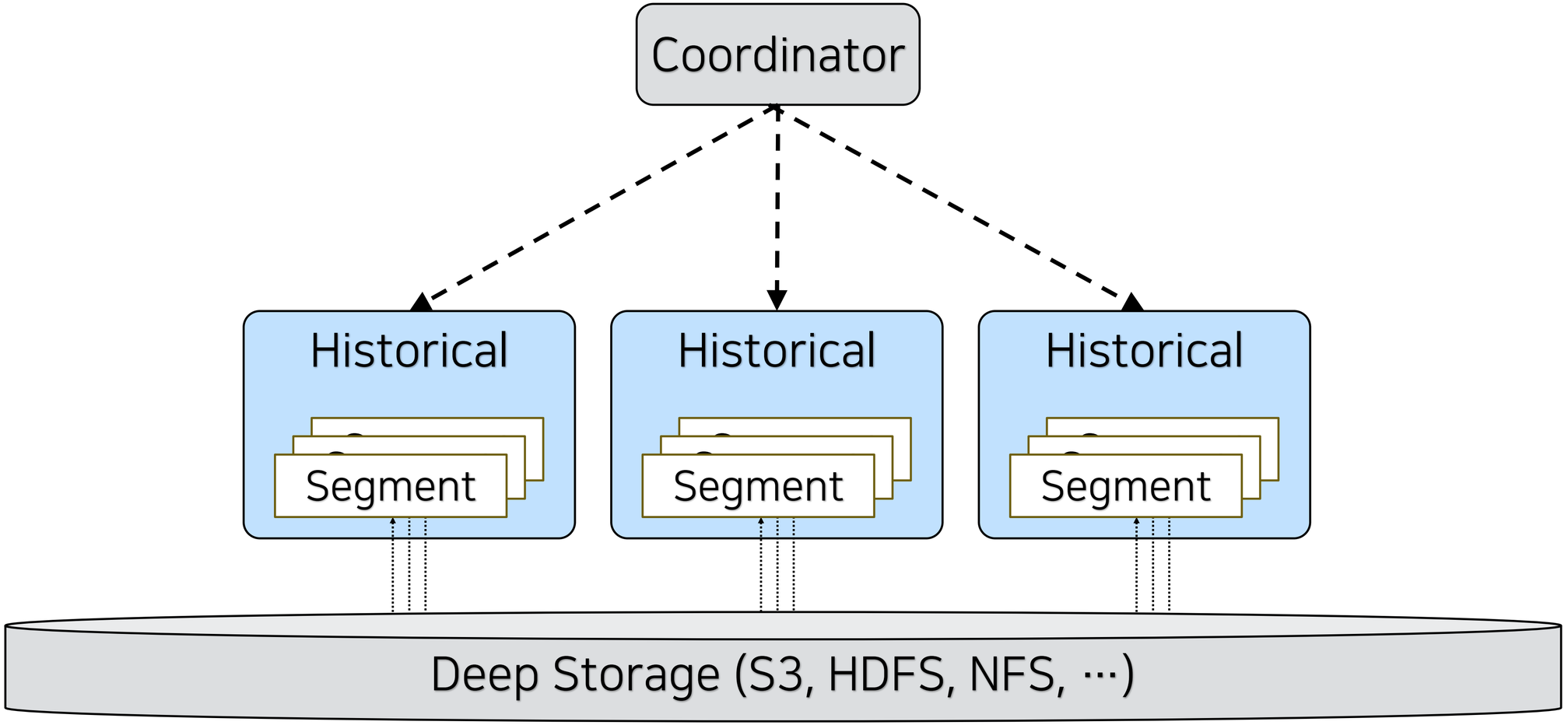
Segment Versioning

DEVIEW 2018

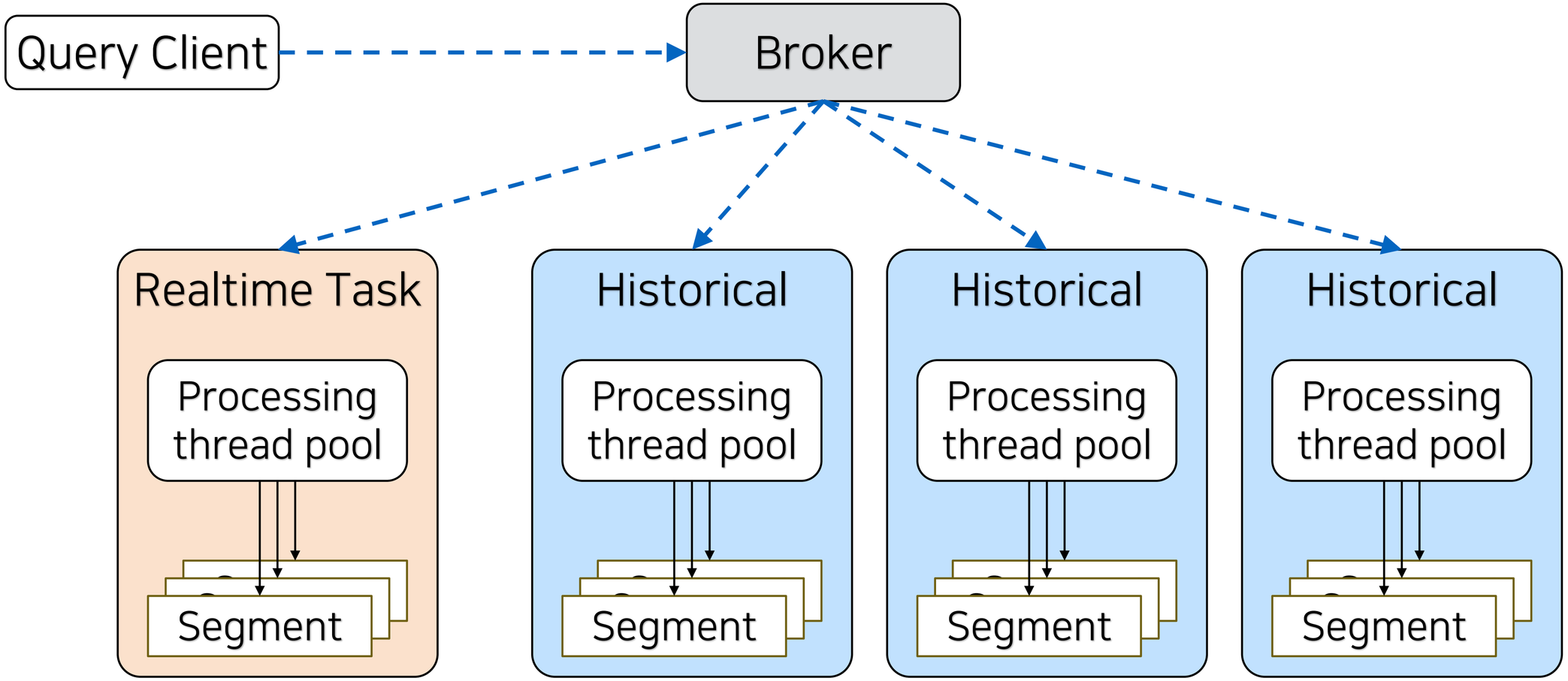
- Each segment has a version
- Usually, recent segments have the higher version
- Segments of the highest version *overshadow* others in the same time chunk



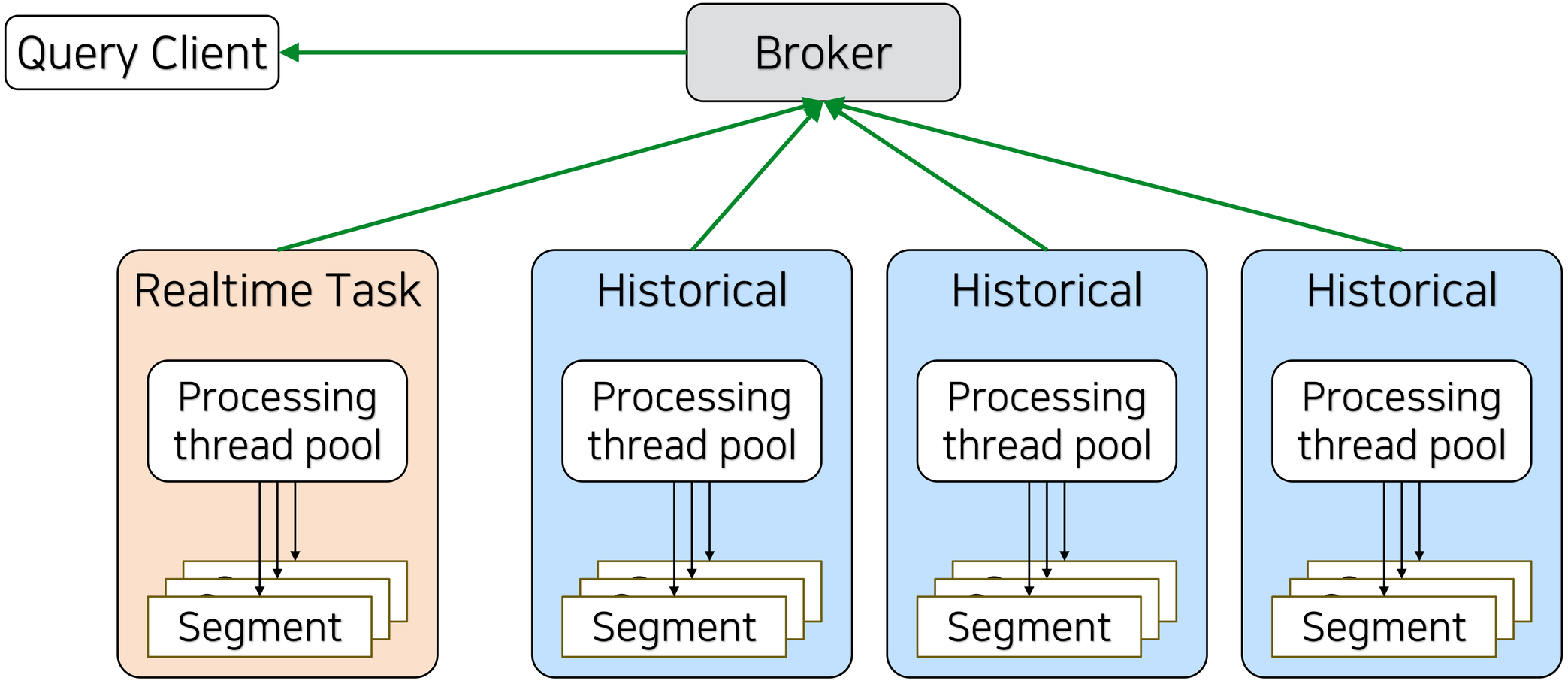
Segment Assignment



Distributed Querying

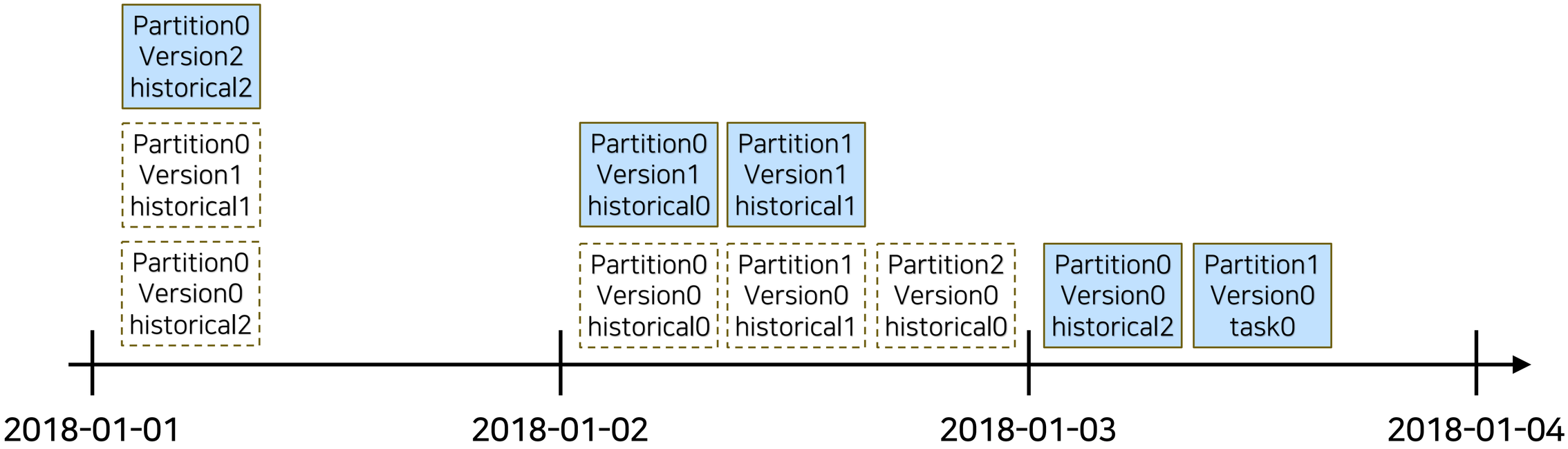


Distributed Querying



Server View

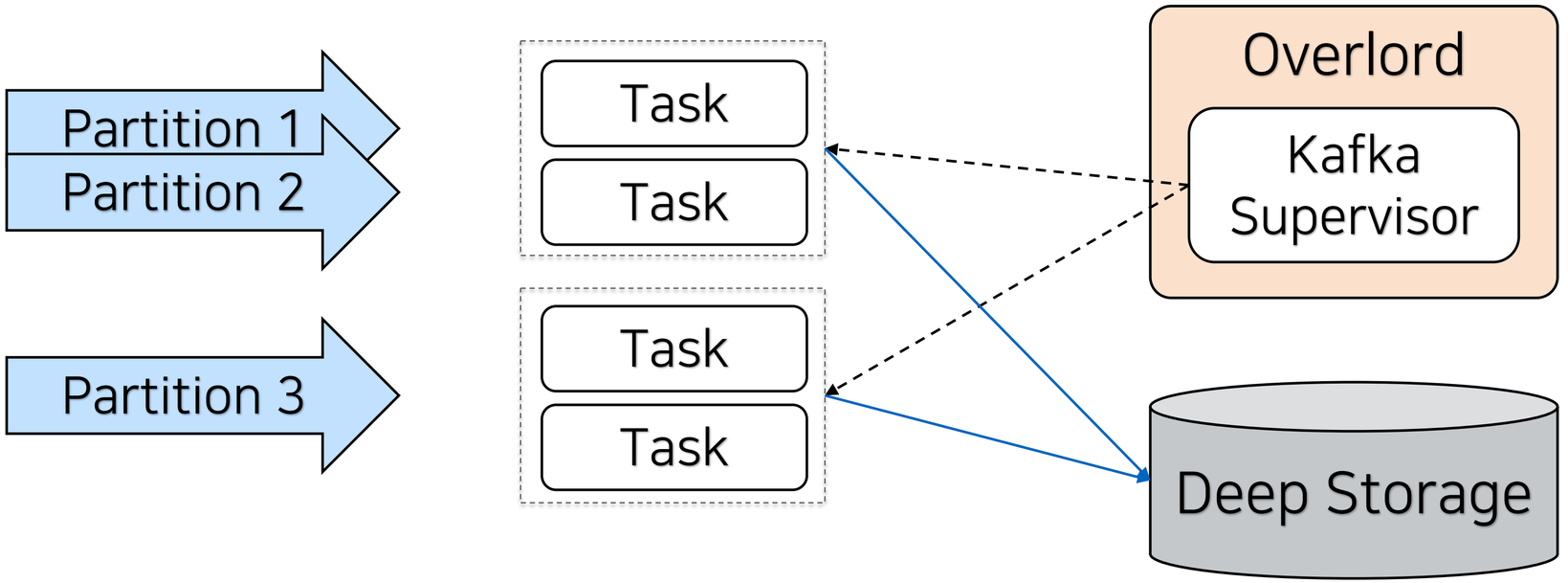
- In-memory structure representing what historicals/tasks serve what segments
- Providing the *timeline* consisting of the latest segments



Example Use Case: Analyzing Stream Data

Kafka Indexing Service

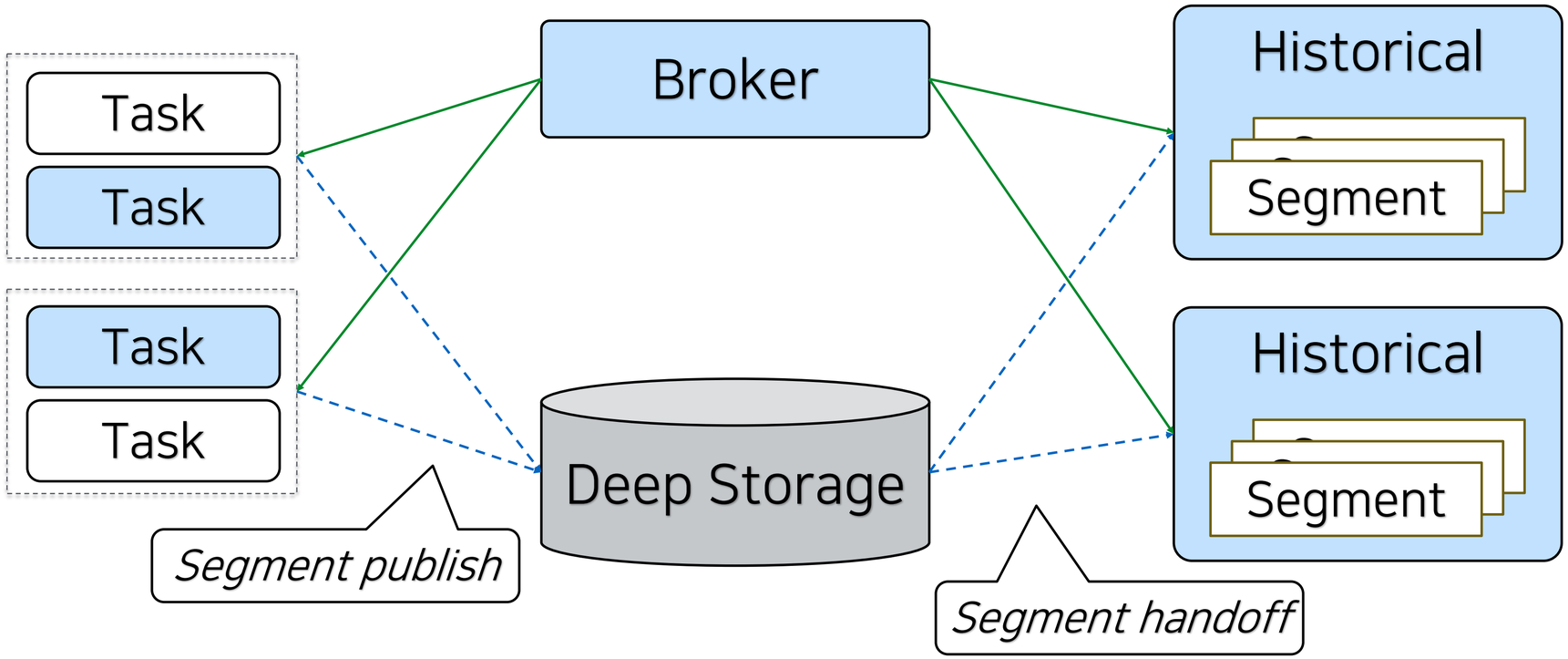
- Data ingestion from Kafka with exactly-once guarantee



Example Use Case: Analyzing Stream Data

Kafka Indexing Service

- Data ingestion from Kafka with exactly-once guarantee



2.
Optimizing Data Storage

DEVIEW 2018

Optimizing Segments

- For both better disk space utilization and better query performance
- Segment size (# of rows): better parallelizing query processing
- Sort order: better compression ratio

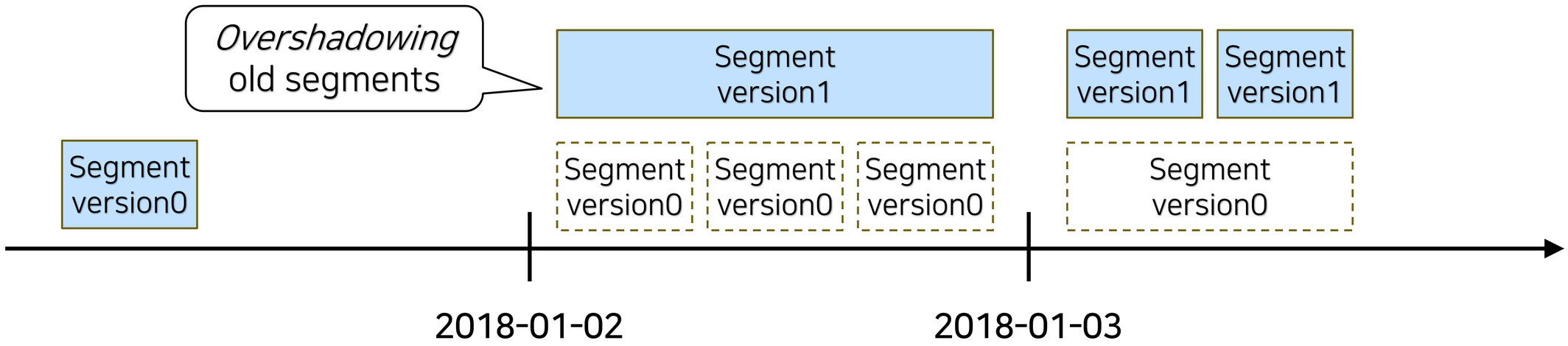
Optimizing Segments

DEVIEW 2018

- Optimizing segments in the first place is not easy especially for stream ingestion

Segment Compaction

- Segment compaction is merging/splitting segments
- Sort order can be changed



Segment Compaction

DEVIEW 2018

General guideline

- Segment size: 300MB ~ 700MB
- # of rows per segment: 5M ~ 20M
- Sort order: Highly compressible dimensions first

Example: Segment Compaction

DEVIEW 2018

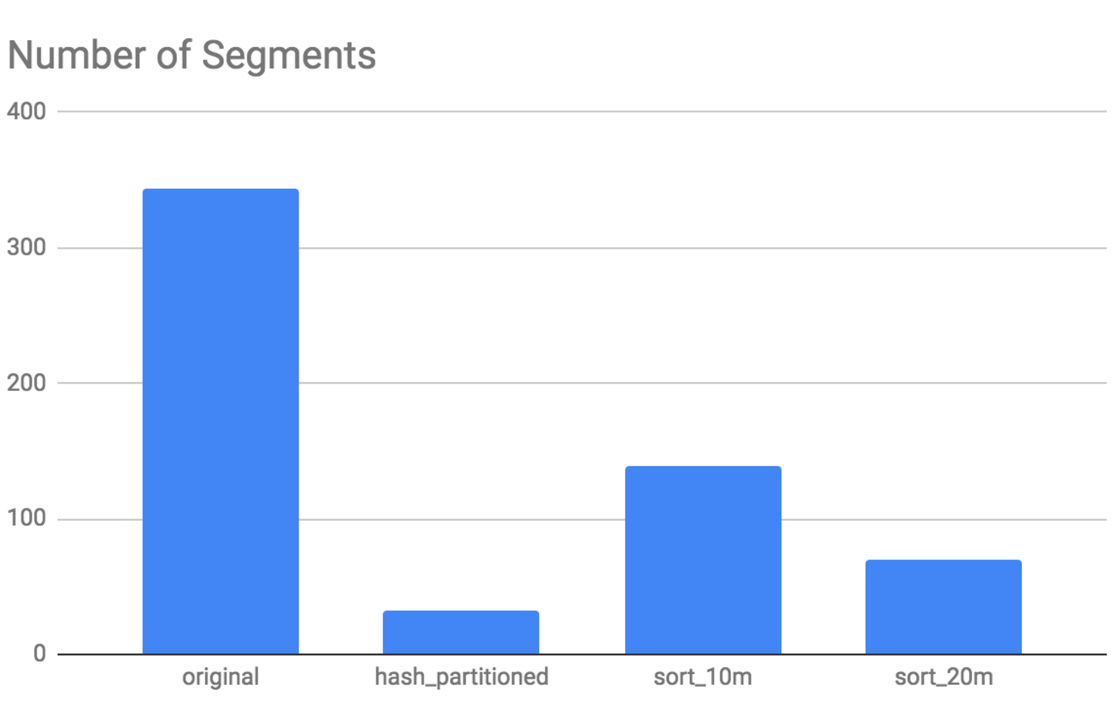
Simple benchmark

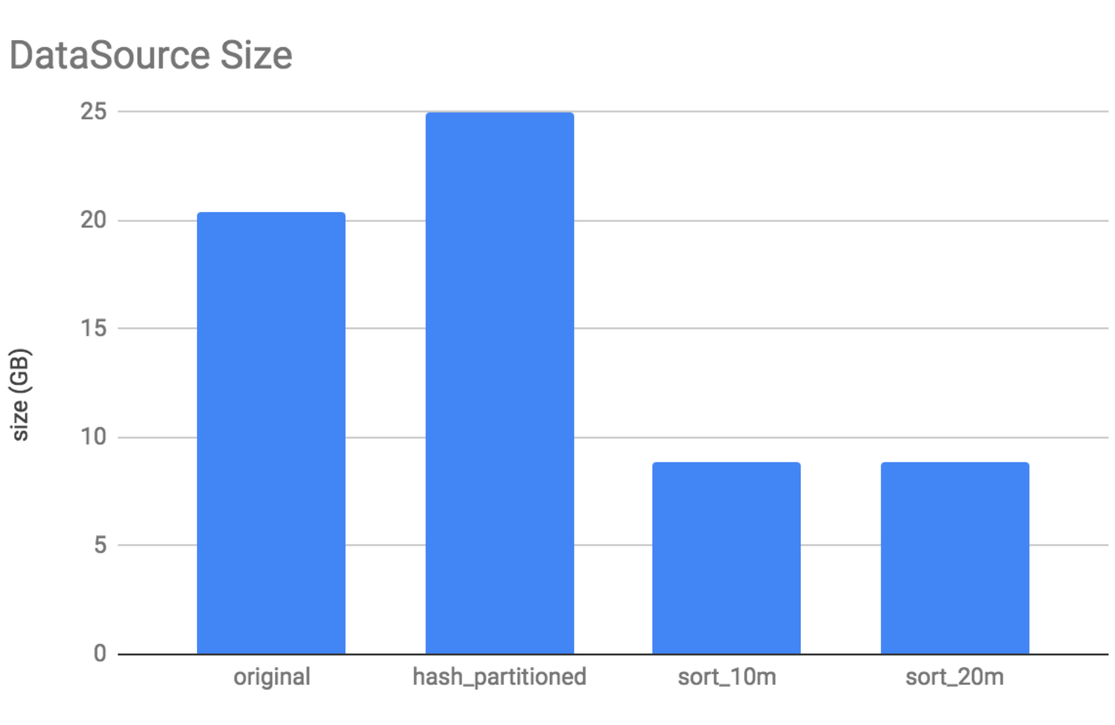
- Original datasource: 20GB of a single date data (57MB each segment), 5 columns
- Compaction with hash partitioning: hash partition by all columns
- Compaction with sorting (10m and 20m rows): kafka partition key, filter column first
- Local cluster: 1 broker + 4 historicals (2 processing threads)

Example: Segment Compaction

DEVIEW 2018

Datasource size

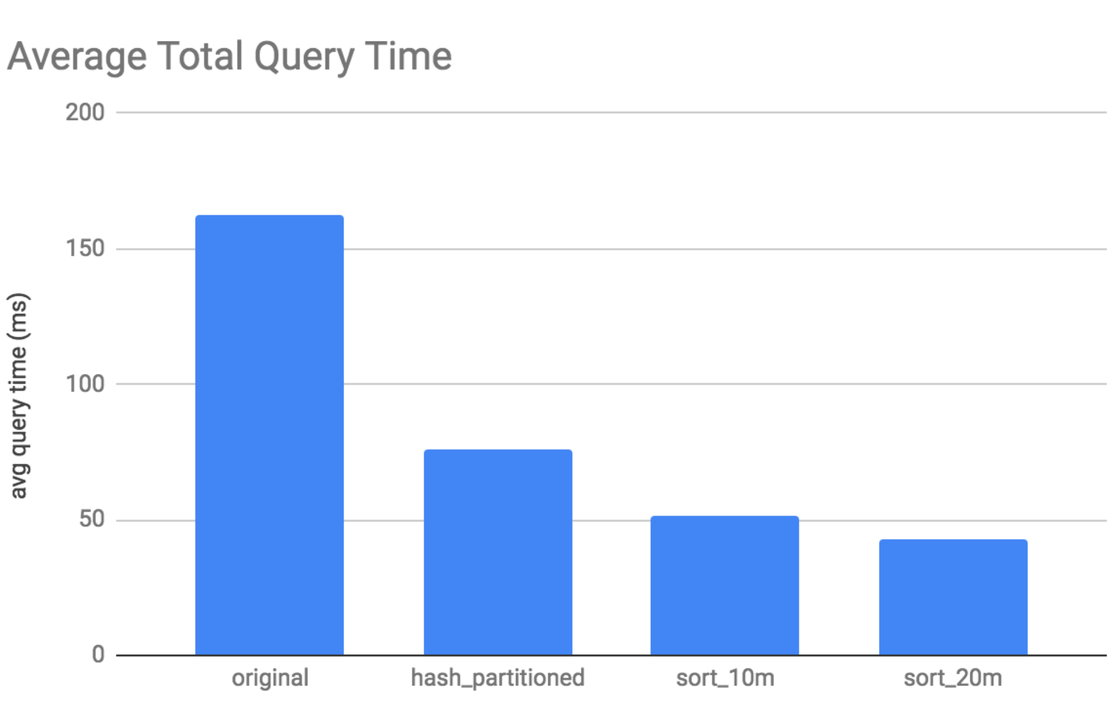


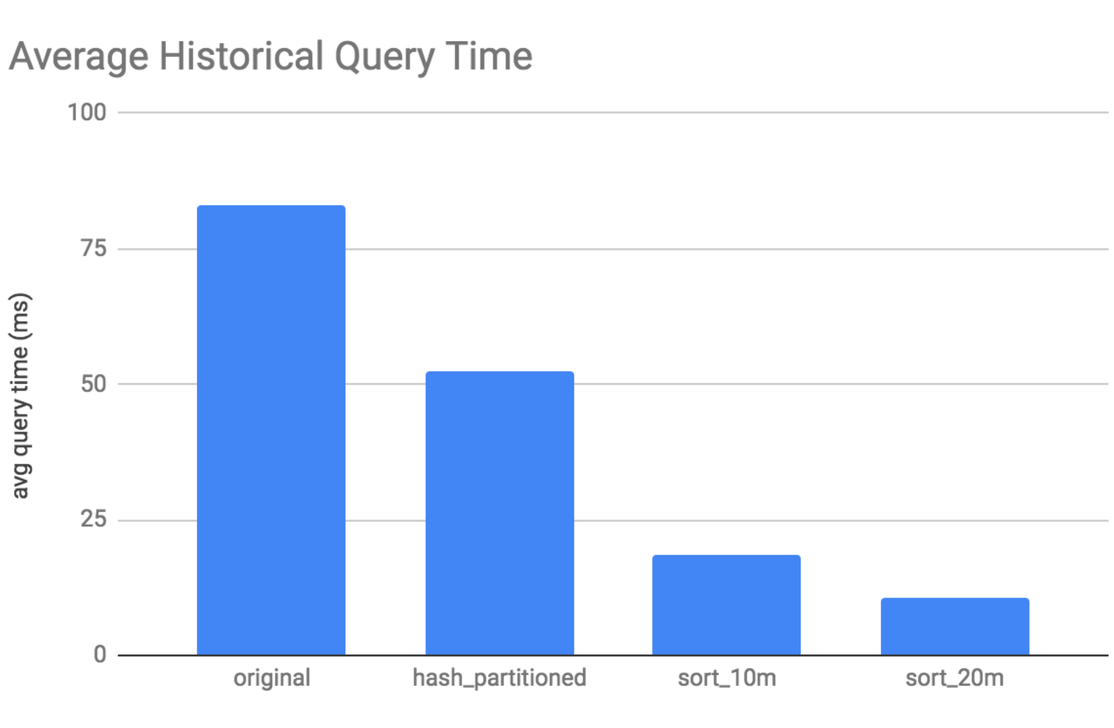


Example: Segment Compaction

DEVIEW 2018

Query performance





Automatic Background Compaction

DEVIEW 2018

Goals

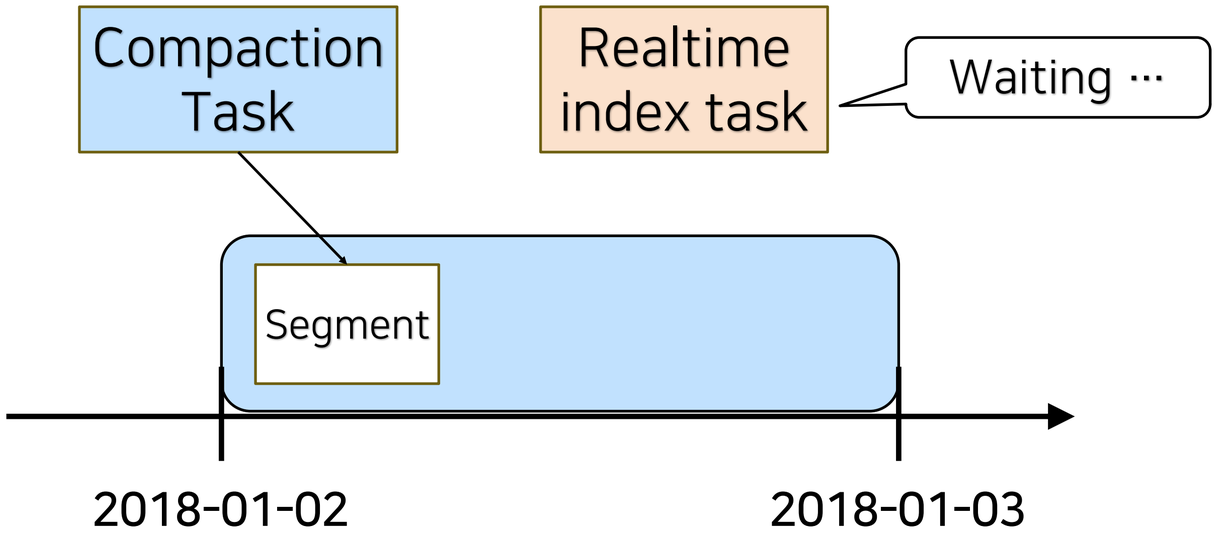
- Compaction should be available for very recent data (recent data is more important!)
- Compaction and other ingestion shouldn't block each other, especially for stream ingestion

Challenges in Automatic Compaction

- Compaction requires to updating existing segments while appending new data at the same time
- What if a new segment is appended into the same time chunk while compacting it?

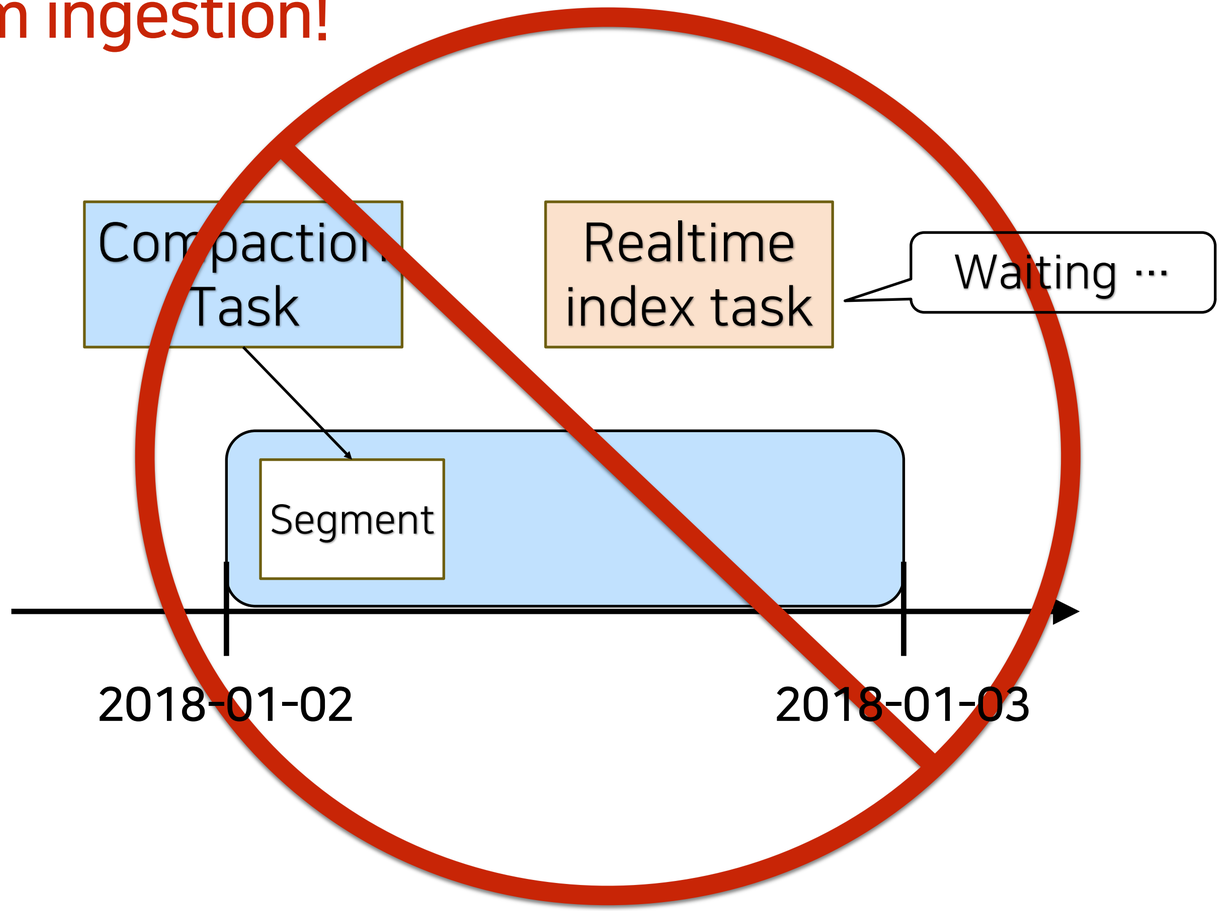
Challenges in Automatic Compaction

- In the current locking system, the later task should wait



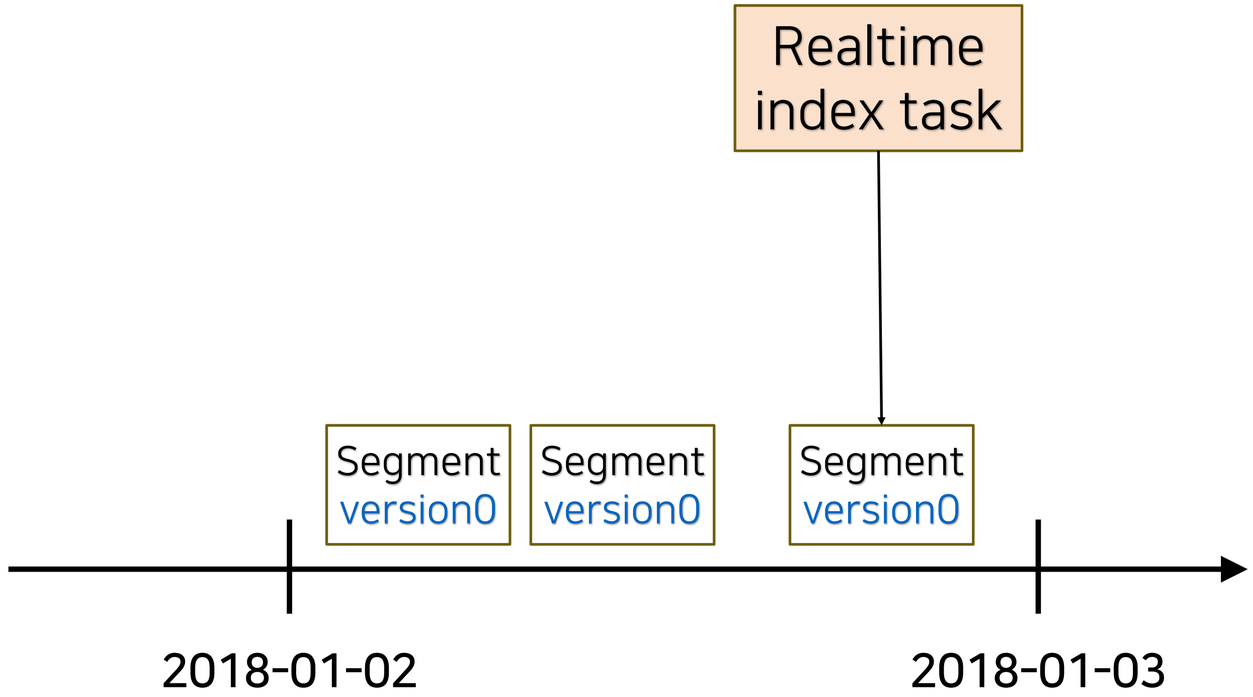
Challenges in Automatic Compaction

- In the current locking system, the later task should wait
- This can block stream ingestion!



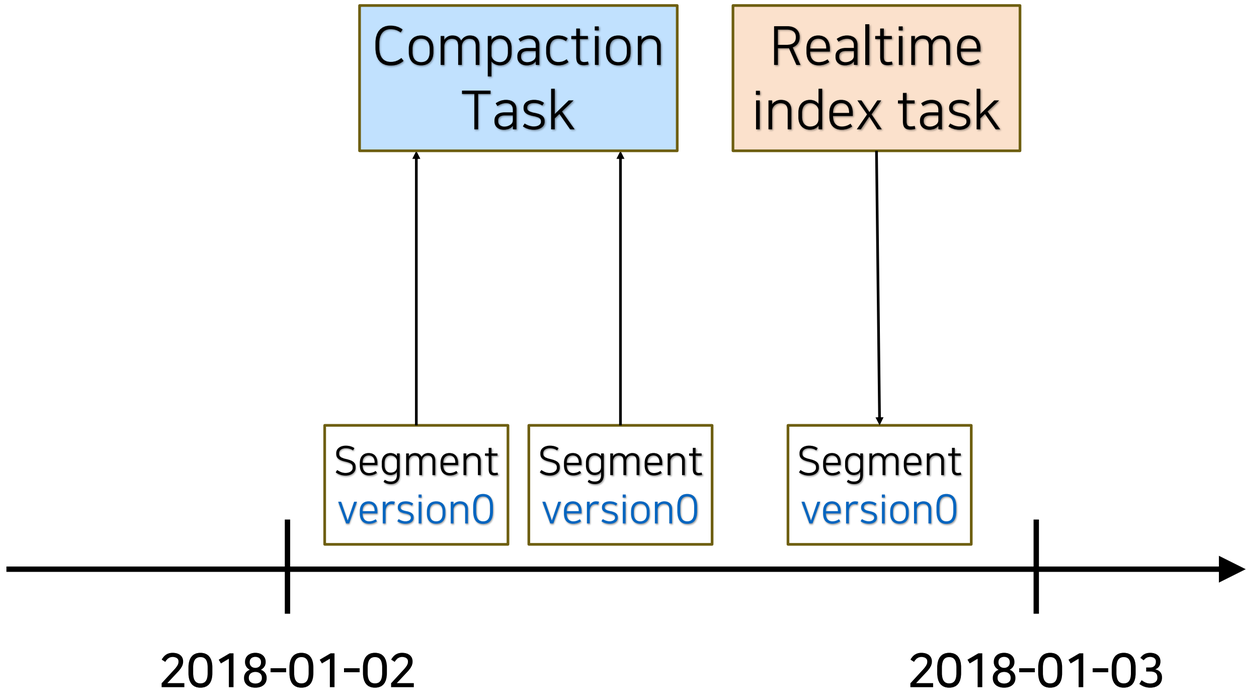
Challenges in Automatic Compaction

- If we allow for compaction task and Realtime index task to write segments to the same time chunk, the timeline will be broken!



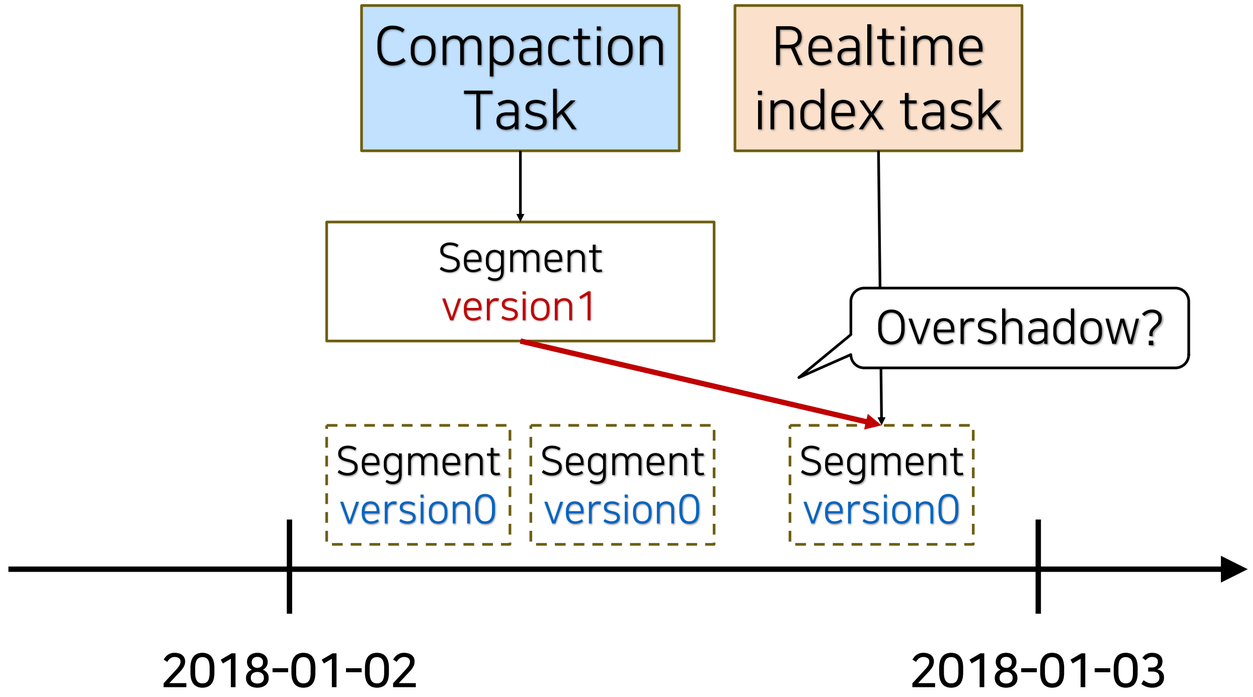
Challenges in Automatic Compaction

- If we allow for compaction task and Realtime index task to write segments to the same time chunk, the timeline will be broken!



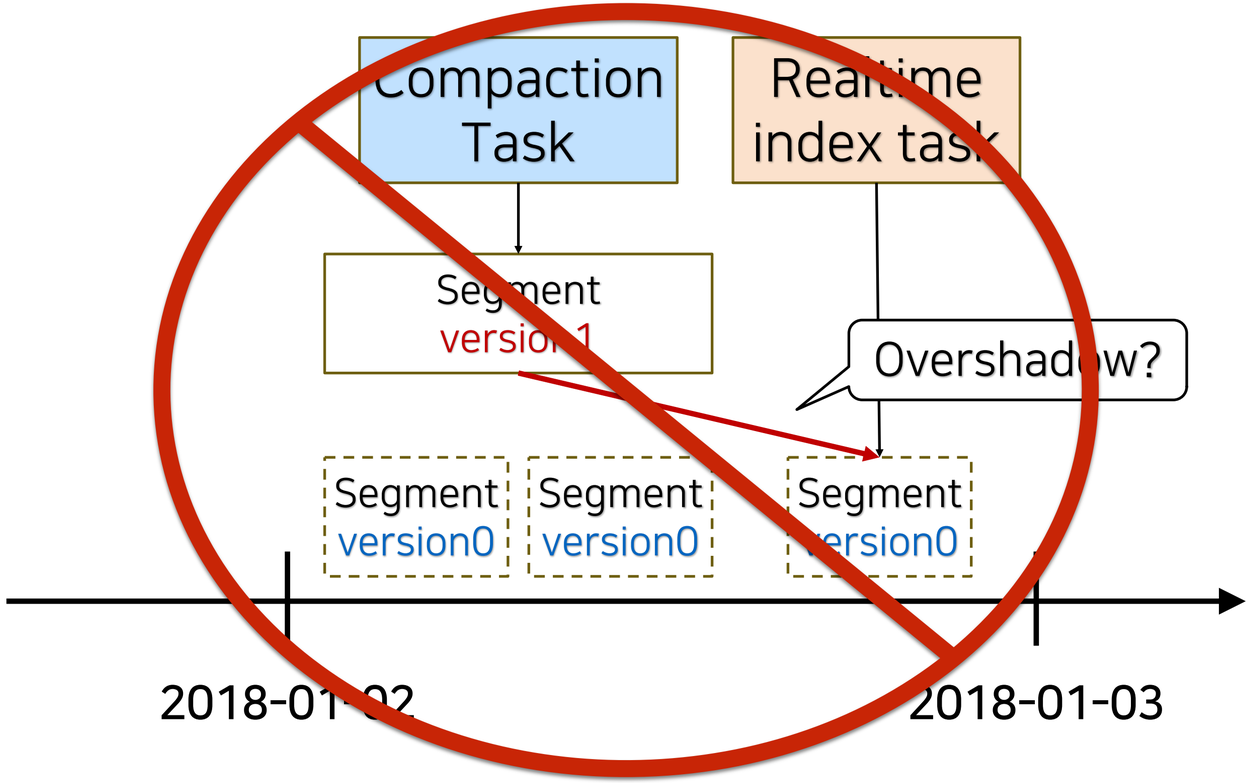
Challenges in Automatic Compaction

- If we allow for compaction task and Realtime index task to write segments to the same time chunk, the timeline will be broken!



Challenges in Automatic Compaction

- If we allow for compaction task and Realtime index task to write segments to the same time chunk, the timeline will be broken!



3.
Automatic Segment Compaction

DEVIEW 2018

Compaction Task

DEVIEW 2018

- https://github.com/apache/incubator-druid/pull/4985 (0.12.0)
- Merging/splitting the segments of the given interval

```
{
  "type" : "compact",
  "dataSource" : "wikipedia",
  "interval" : "2017-01-01/2018-01-01"
}
```

Prioritized Locking

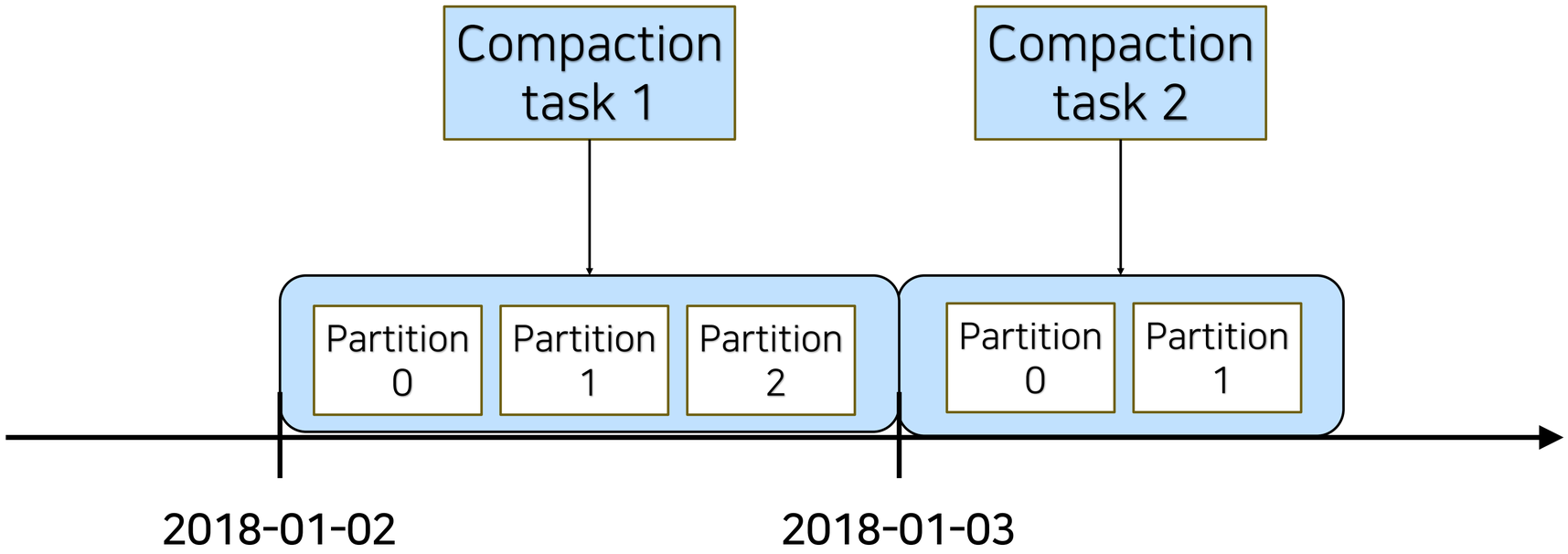
- https://github.com/apache/incubator-druid/pull/4550 (0.12.0)
- Each task has a priority
- A task of a higher priority can revoke the locks of another task of a lower priority

Task type	Default priority
Realtime tasks	75
Batch tasks	50
Merge/Append/Compaction tasks	25
Others	0

Prioritized Locking

DEVIEW 2018

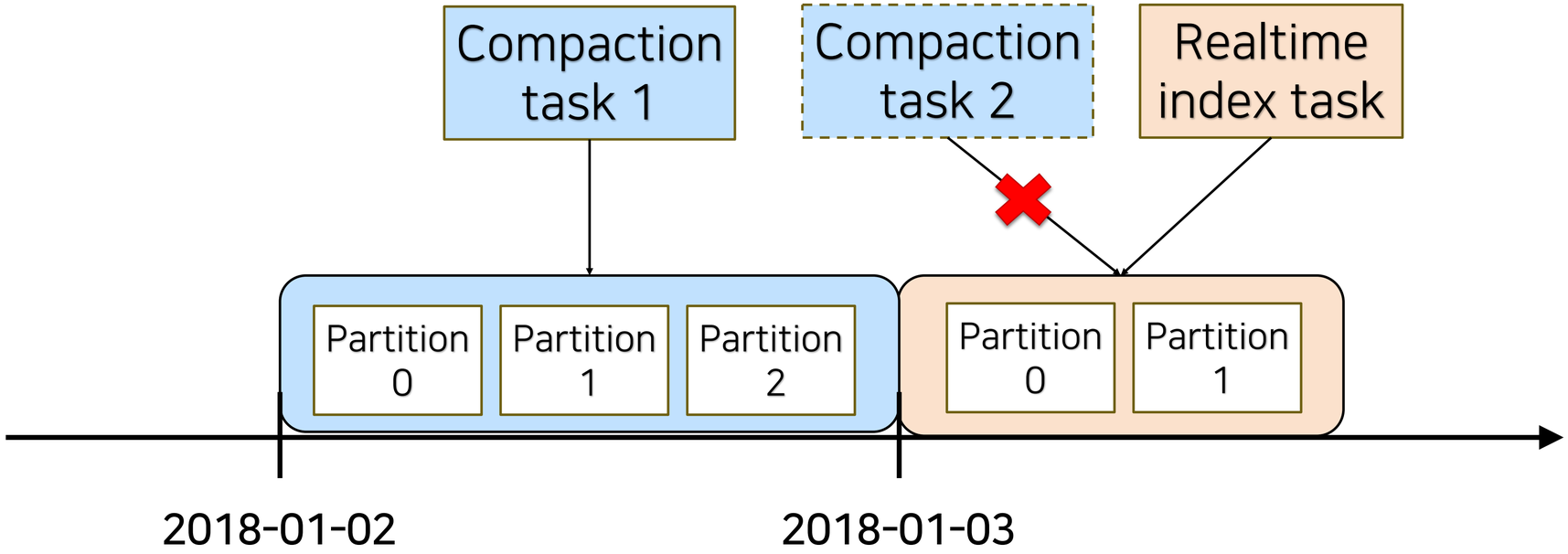
- CompactionTask and other tasks can run for the same dataSource at the same time if they are not writing to the same time chunk



Prioritized Locking

DEVIEW 2018

- CompactionTask and other tasks can run for the same dataSource at the same time if they are not writing to the same time chunk



Prioritized Locking

DEVIEW 2018

- Realtime tasks can beat compaction tasks
- No blocking for stream ingestion
- No accidentally overshadowing segments

Automatic Compaction by the Coordinator

- https://github.com/apache/incubator-druid/pull/5102 (0.13.0)
- The coordinator periodically finds sub-optimized segments and compacts them
- Checking segments from the latest to the oldest
- Currently it automatically optimizes only segment size
- Other configurations (e.g., sort order) can be manually set

Automatic Compaction by the Coordinator

DEVIEW 2018

- Example automatic compaction spec

```
{
  "dataSource": "twitter",
  "keepSegmentGranularity": true,
  "targetCompactionSizeBytes": 419430400,
  "skipOffsetFromLatest": "P1D"
}
```

2018-08-07T04	25	27 MB
2018-08-07T03	26	31.3 MB
2018-08-07T02	25	29.1 MB
2018-08-07T01	25	28.8 MB
2018-08-07T00	1	25.5 MB
2018-08-06T23	1	26 MB
2018-08-06T22	1	25.2 MB
2018-08-06T21	1	23.8 MB

4.
Is Everything Going to be OK?

DEVIEW 2018

Case: Late Data

- Long trickle of late data
- Most data for a particular day comes in real time
- But small amounts of late data come in over the next 30 days
- Impossible to run compaction tasks until late data stops coming in

Case: Backfill

DEVIEW 2018

- Loading historical data through Kafka
- Usually the historical data comes in no particular order
- Kafka index tasks would generate lots of waves of small segments for the entire time chunks
- Impossible to run compaction tasks until data loading finishes

5.
Towards Continuous Compaction

DEVIEW 2018

Requirements for Continuous Compaction

- A segment should be available for compaction immediately once it gets published.
- Overwriting tasks (including compaction task) and appending tasks shouldn't interfere each other
- More generally speaking, two tasks shouldn't block each other if they read/write from/to different segments

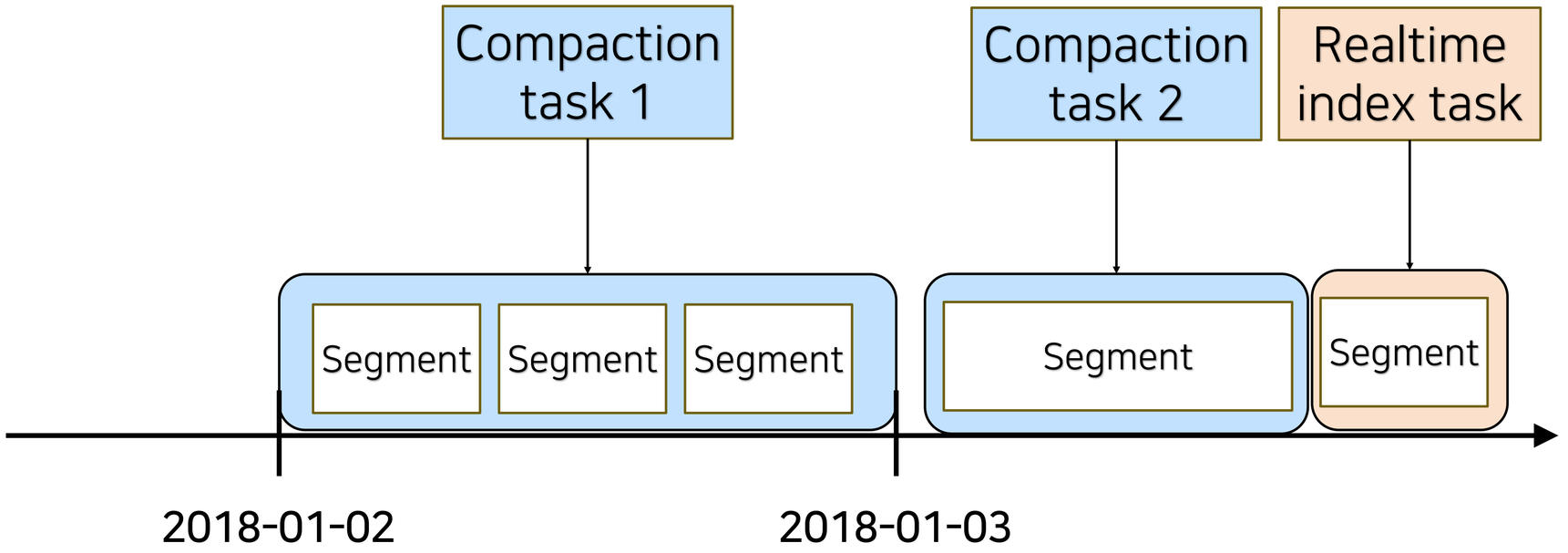
More Granular Locking

DEVIEW 2018

- If a task can lock individual segments,
- A compaction task can run for the very latest segment as soon as it gets published
- There’s no lock contention among tasks if they read/write from/to different segments

Segment Locking

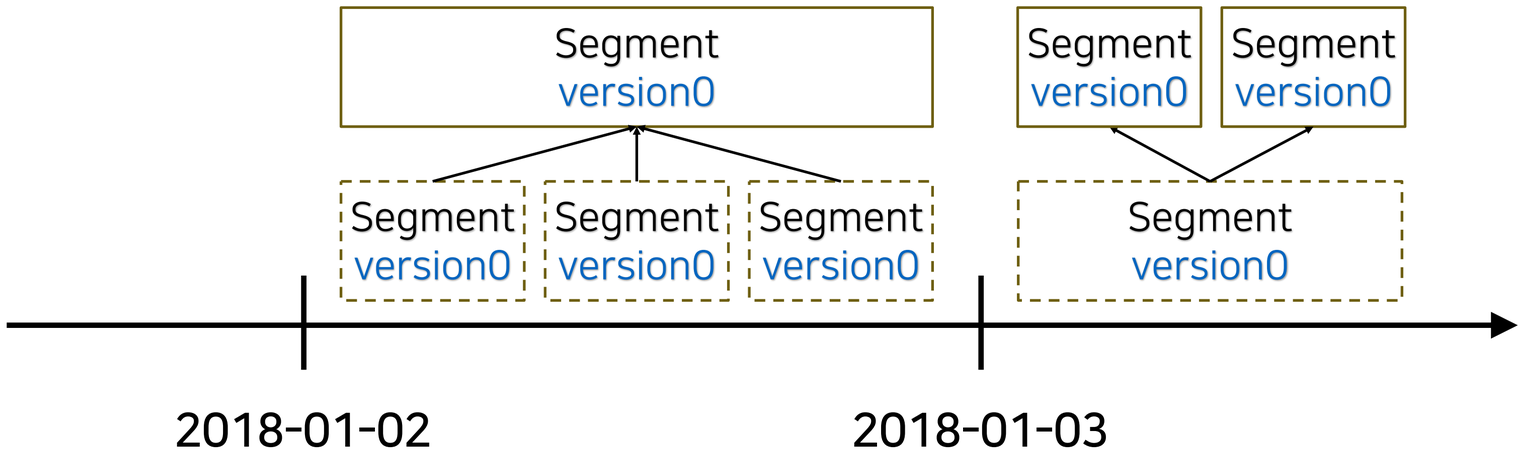
- https://github.com/apache/incubator-druid/issues/6319
- Tasks get locks for individual segments
- *Segment locking doesn't change the segment version!*



Server View Maintenance

Overshadowed Segments

- The segments generated by overwriting tasks (including compaction task) store the references to the input segments
- This is called *overshadowed segments*

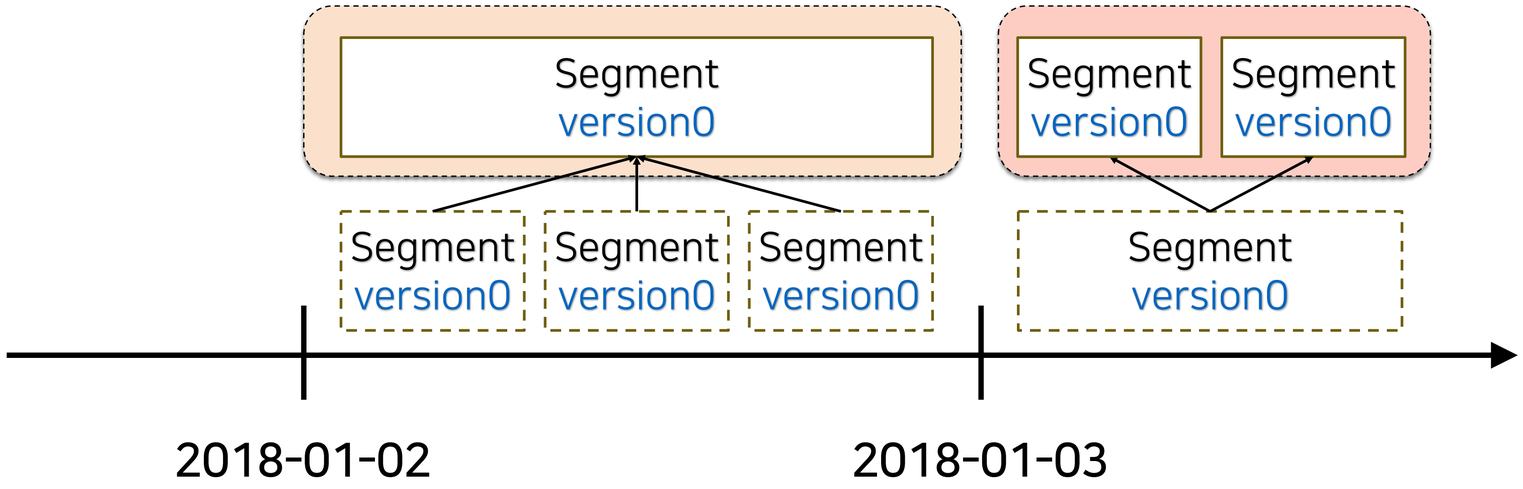


Server View Maintenance

DEVIEW 2018

Atomic Update Group

- The segments store the references to all segments generated together
- This is called *atomic update group*

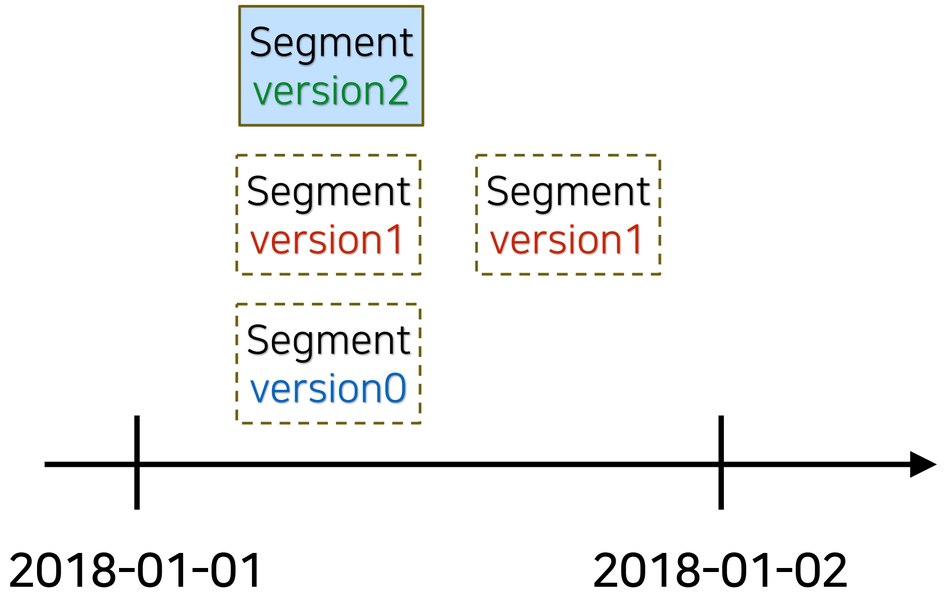


Server View Maintenance

DEVIEW 2018

Two-level management

- In a time chunk, segments of the highest version overshadows others

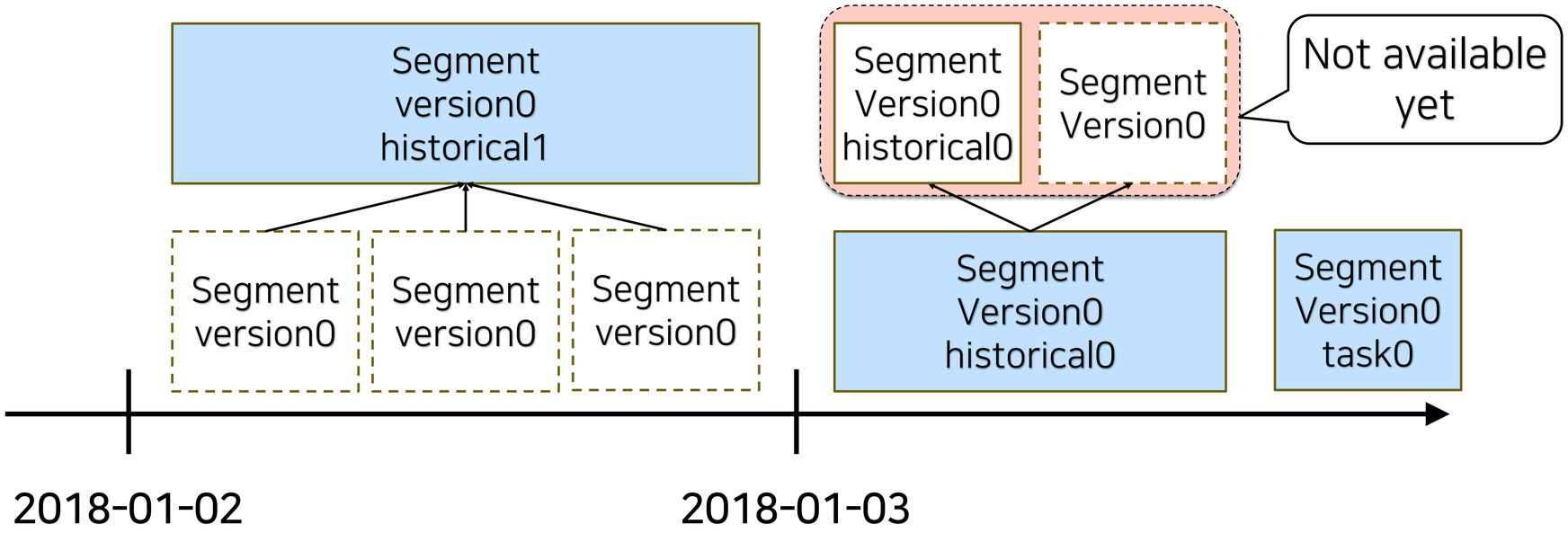


Server View Maintenance

DEVIEW 2018

Two-level management

- If two segments have the same version, a segment overshadows another if that is in the overshadowed segments
- All segments in the atomic update group should be available at the same time

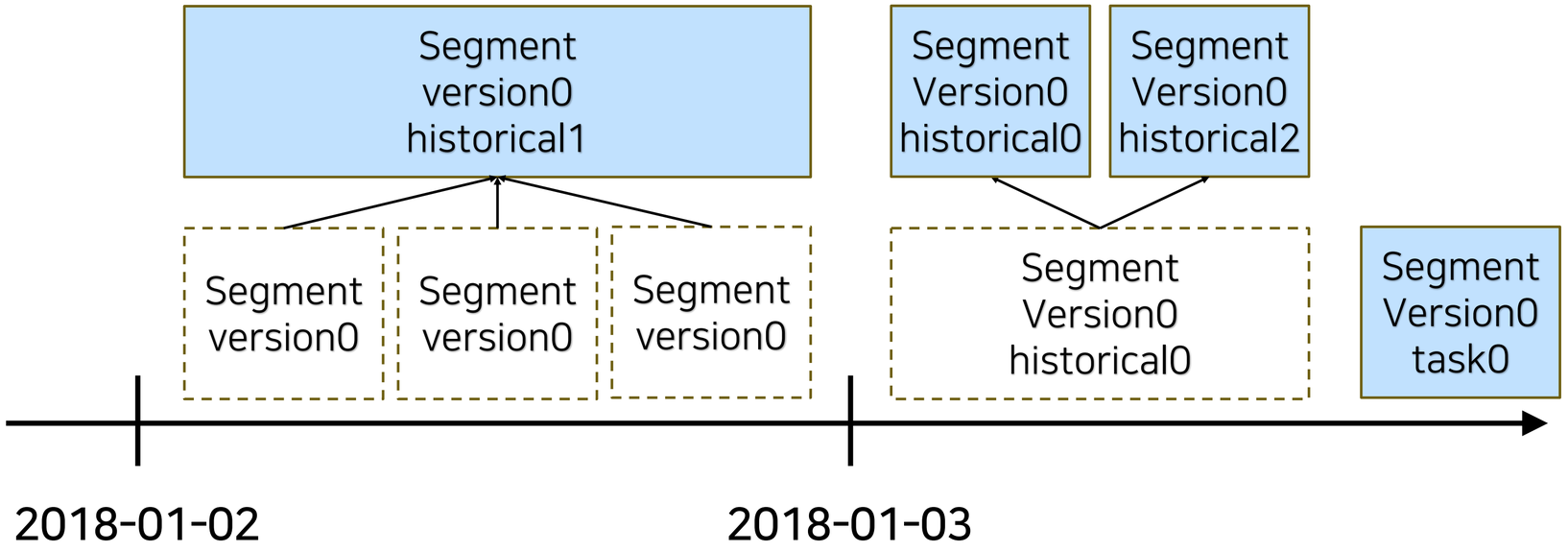


Server View Maintenance

DEVIEW 2018

Two-level management

- If two segments have the same version, a segment overshadows another if that is in the overshadowed segments
- All segments in the atomic update group should be available at the same time



Future Work

DEVIEW 2018

- Compaction based on parallel index task types
- Automatic optimization based on the query pattern
- Considering different factors other than segment size (secondary partitioning, sort order)

Q & A

TRACK 3

Thank you

Compaction: hash-33

DEVIEW 2018

```
{
  "type" : "index",
  "spec" : {
    "dataSchema" : {
      "dataSource" : ”hash_33",
      "parser" : {
        "type" : "noop",
        "parseSpec" : {
          "dimensionsSpec" : {
            "dimensions" : [
              { "name" : ”kafka_part_key" },
              { "name" : ”dim1" },
              { "name" : ”dim2" },
              { "type" : "double", "name" : ”met1” }
            ]
          }
        }
      },
      "granularitySpec" : {
        "type" : "arbitrary",
        "queryGranularity" : { "type" : "none" },
        "intervals" : [ "2017-12-29/2017-12-30" ]
      }
    },
    "ioConfig" : {
      "type" : "index",
      "firehose" : {
        "type" : "ingestSegment",
        "dataSource" : "original_datasource",
        "interval" : "2017-12-29/2017-12-30",
        "dimensions" : [ ”kafka_part_key", ”dim1", ”dim2", ”met1" ],
      }
    },
    "tuningConfig" : {
      "type" : "index",
      "numShards": 33,
      "forceGuaranteedRollup" : true
    }
  }
}
```

Compaction: Sort-20m

DEVIEW 2018

```
{
  "type" : "index",
  "spec" : {
    "dataSchema" : {
      "dataSource" : ”sort_20m",
      "parser" : {
        "type" : "noop",
        "parseSpec" : {
          "dimensionsSpec" : {
            "dimensions" : [
              { "name" : ”kafka_part_key" },
              { "name" : ”dim1" },
              { "name" : ”dim2" },
              { "type" : "double", "name" : ”met1" },
              { "type" : "long", "name" : "raw_timestamp" }
            ]
          }
        }
      },
      "granularitySpec" : {
        "type" : "uniform",
        "segmentGranularity" : "DAY",
        "queryGranularity" : "DAY",
        "intervals" : [ "2017-12-29/2017-12-30" ]
      },
      "transformSpec" : {
        "transforms": [
          {
            "type": "expression",
            "name": "raw_timestamp",
            "expression": "timestamp(timestamp)"
          }
        ]
      }
    },
    "ioConfig" : {
      "type" : "index",
      "firehose" : {
        "type" : "ingestSegment",
        "dataSource" : "original_datasource",
        "interval" : "2017-12-29/2017-12-30",
        "dimensions" : [ ”kafka_part_key", ”dim1", ”dim2", ”met1", "raw_timestamp" ]
      }
    },
    "tuningConfig" : {
      "type" : "index",
      "targetPartitionSize" : 20000000
    }
  }
}
```