A stream processing pipeline for an online advertising platform

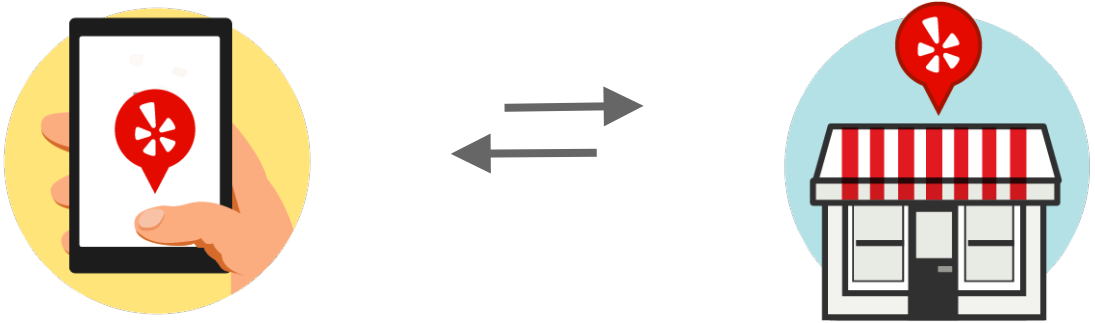
Yifan Wang

Amit Ramesh

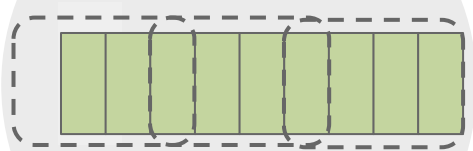
yelp

Yelp's Mission

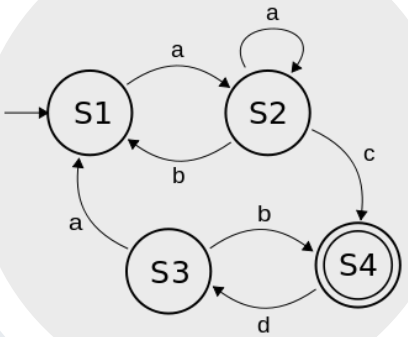
Connecting people with great local businesses.



Yelp Ads

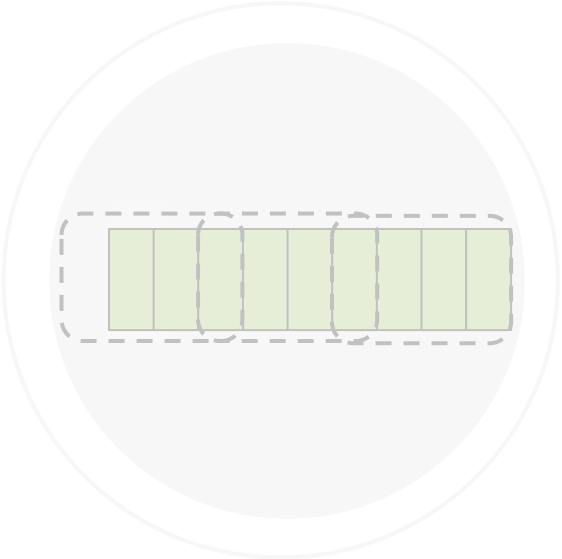


Sliding Window

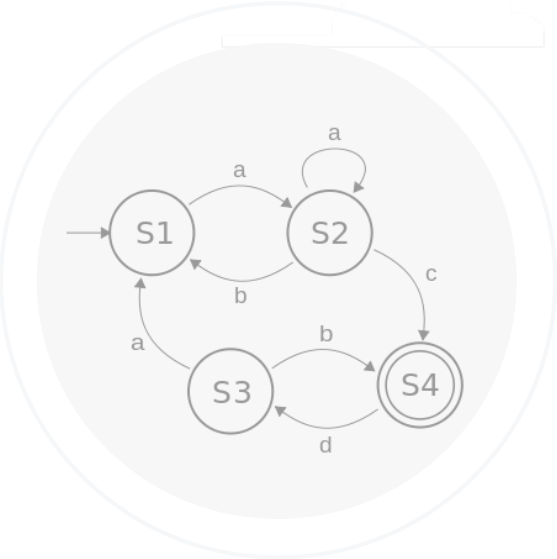


State Management

Yelp Ads

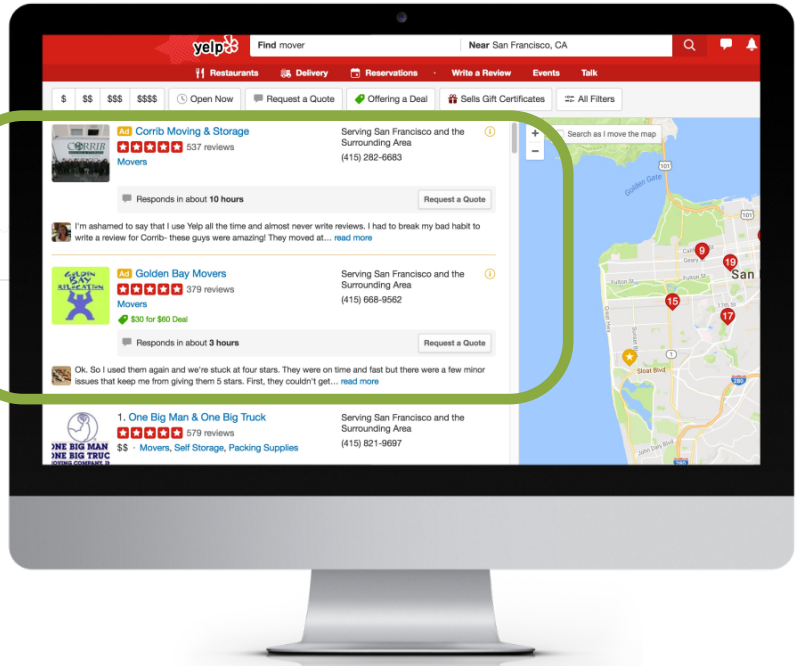


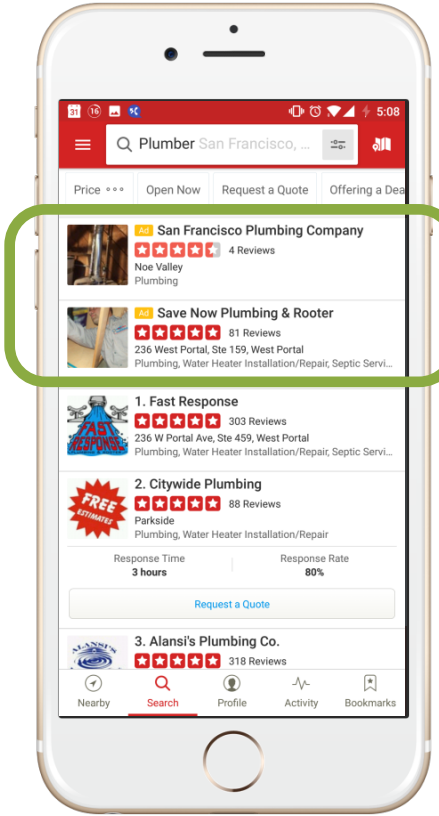
Sliding Window



State Management

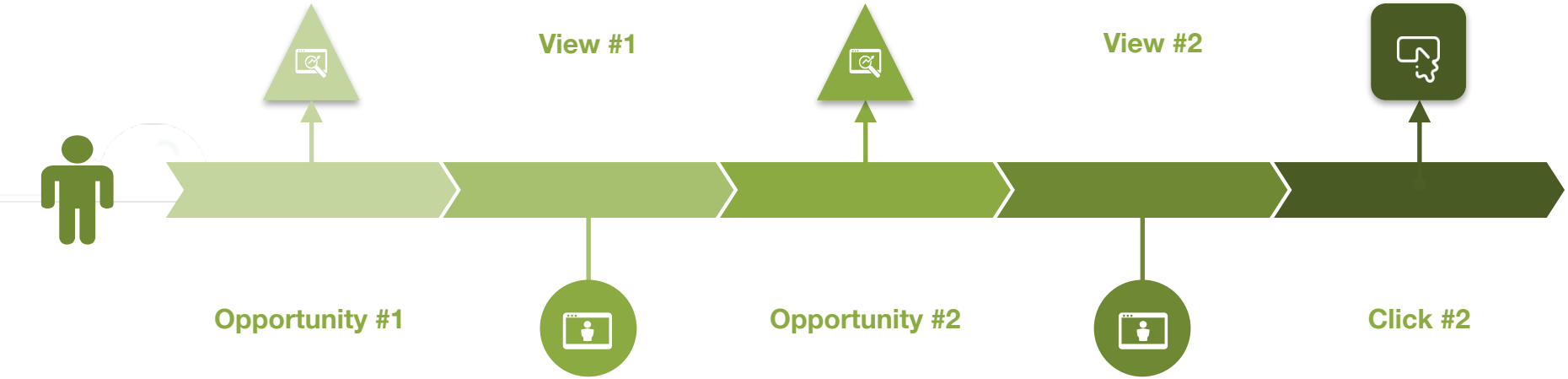
Yelp Ads





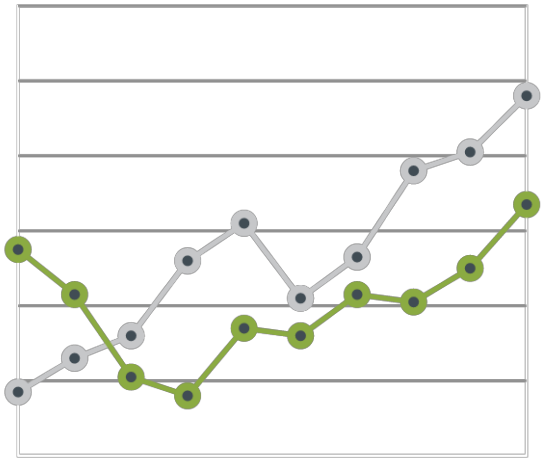
SPARK SUMMIT 2017

Events Timeline



- Opportunity
- View
- Click

Why Real-time?



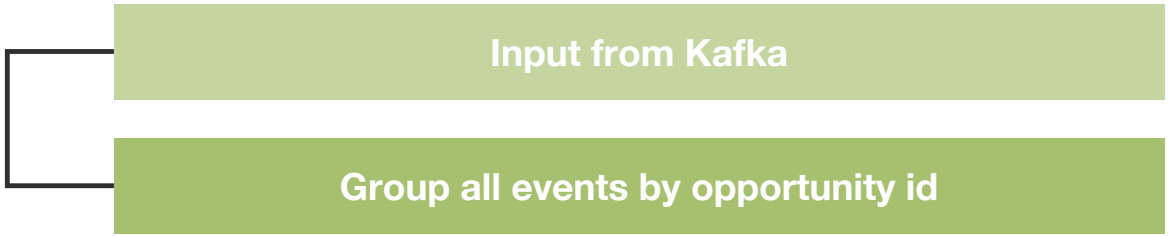
Real-time Metrics

Campaign Overspending

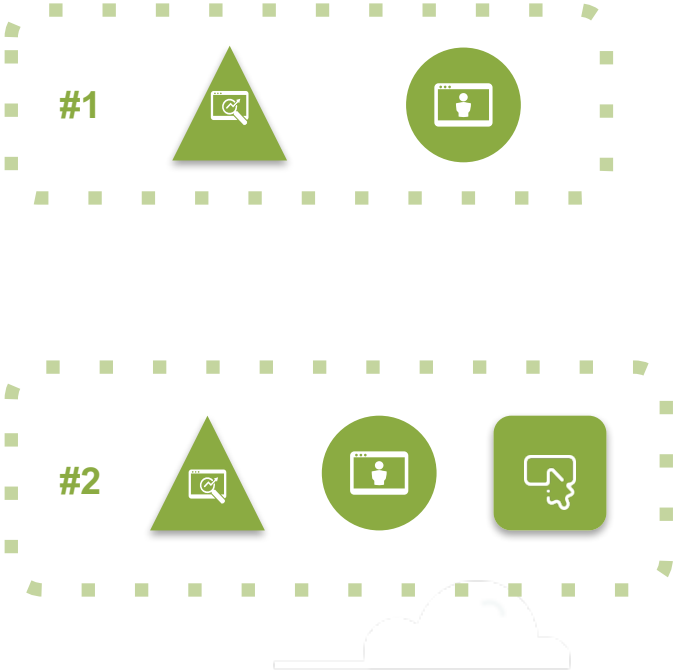
Overall Process

Input from Kafka

Overall Process

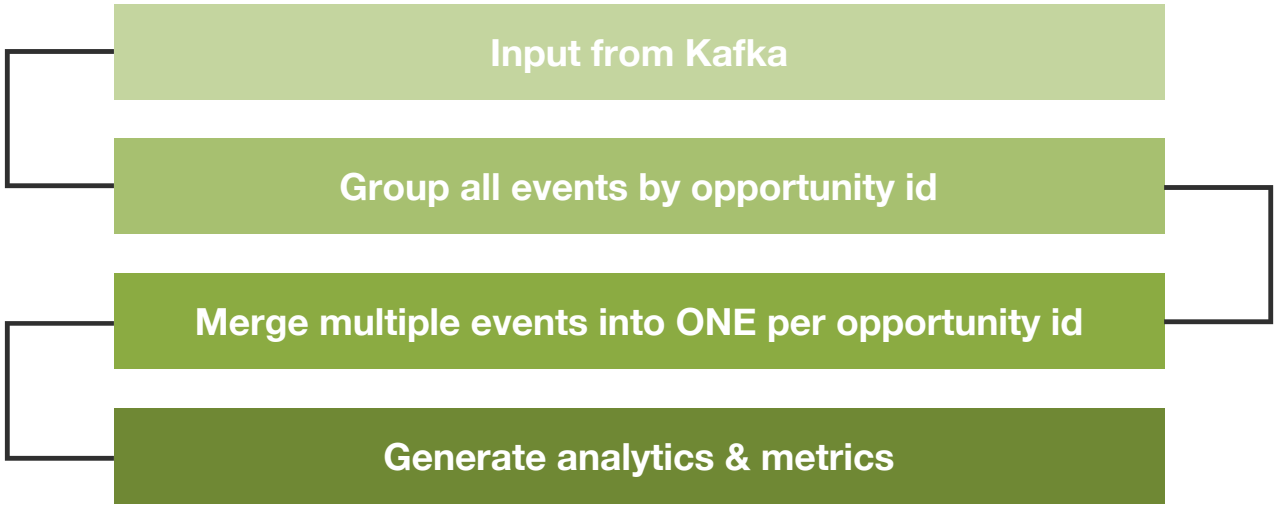


Group Events By Id

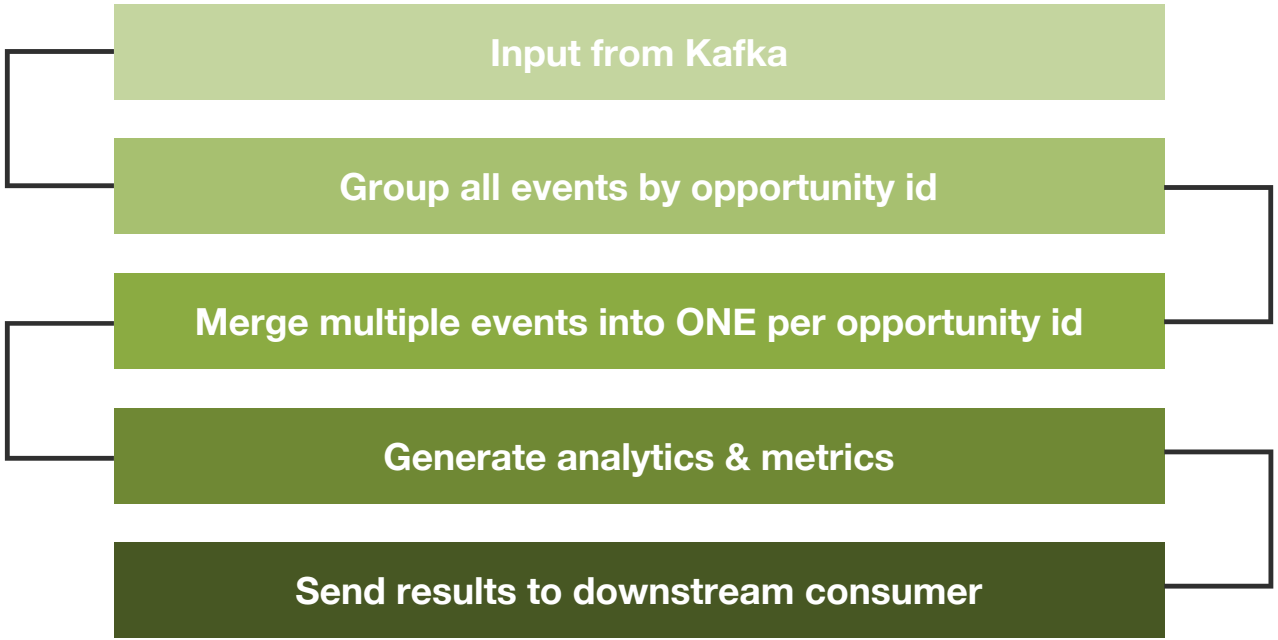


Overall Process

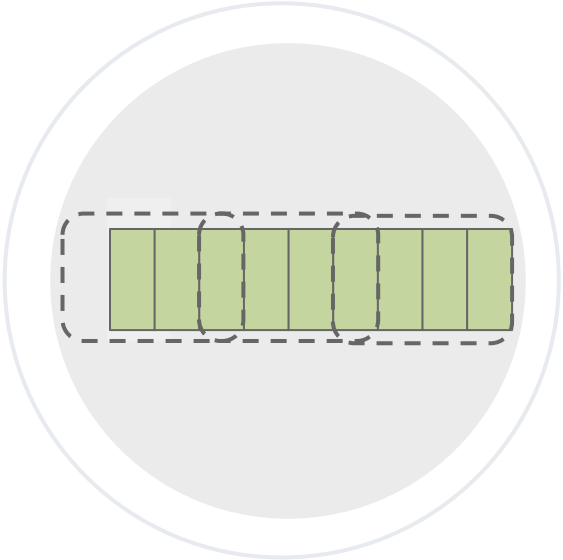
Overall Process



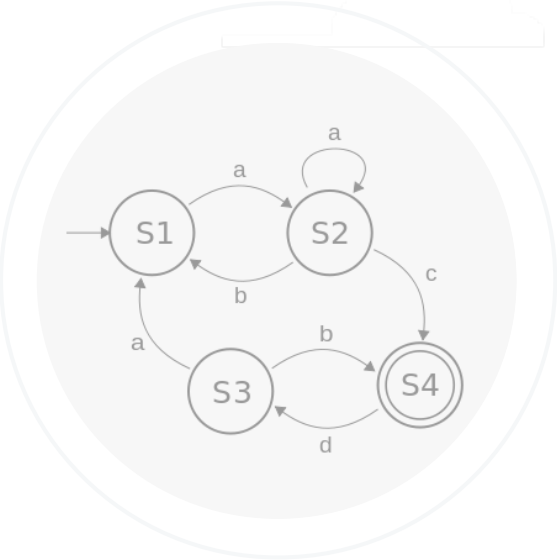
Overall Process



Yelp Ads

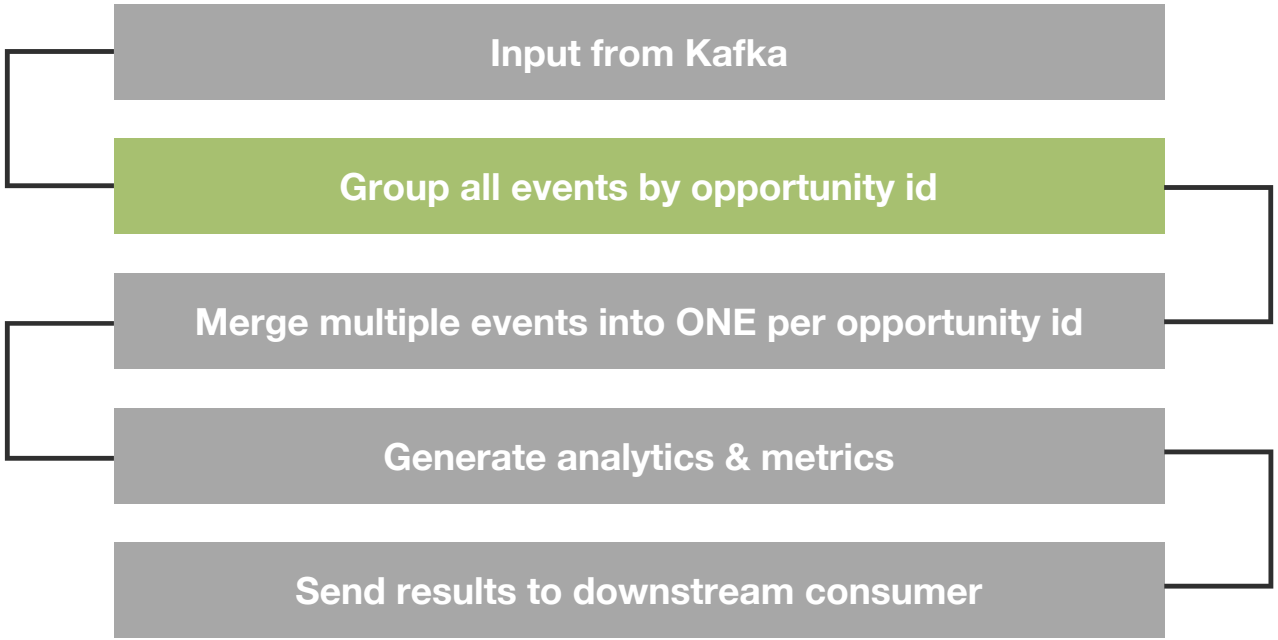


Sliding Window



State Management

Overall Process



Sliding Window in Spark

Spark API:

- Window()
- countByWindow()
- reduceByWindow()
- reduceByKeyAndWindow()
- countByValueAndWindow()

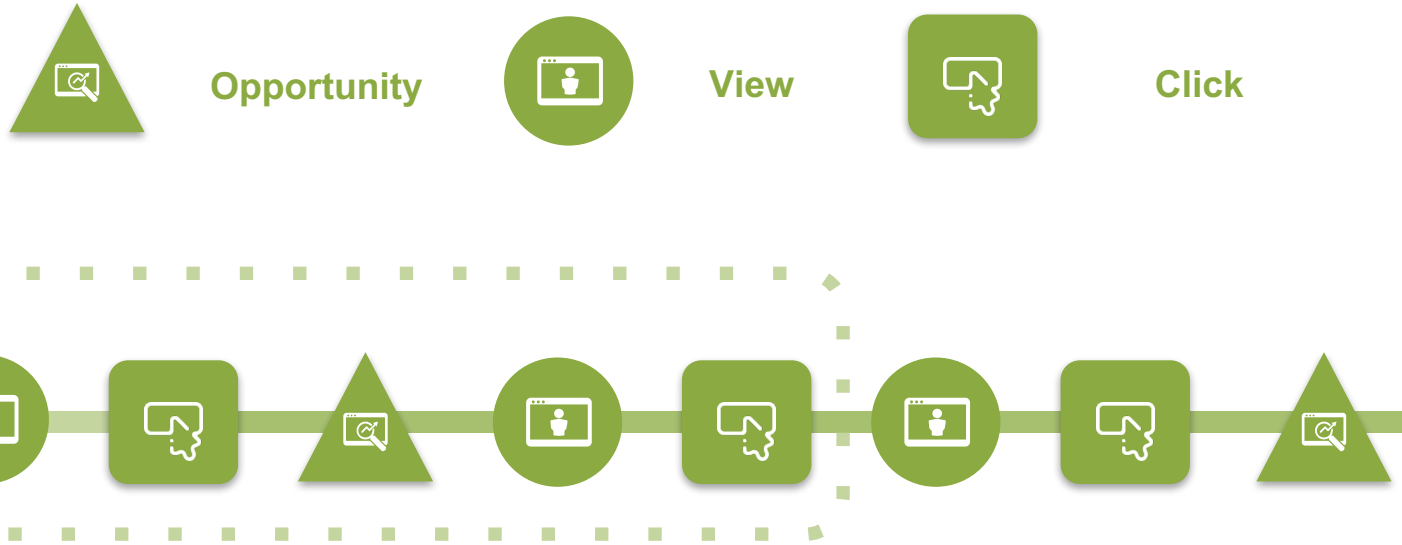
Sliding Window in Spark

?

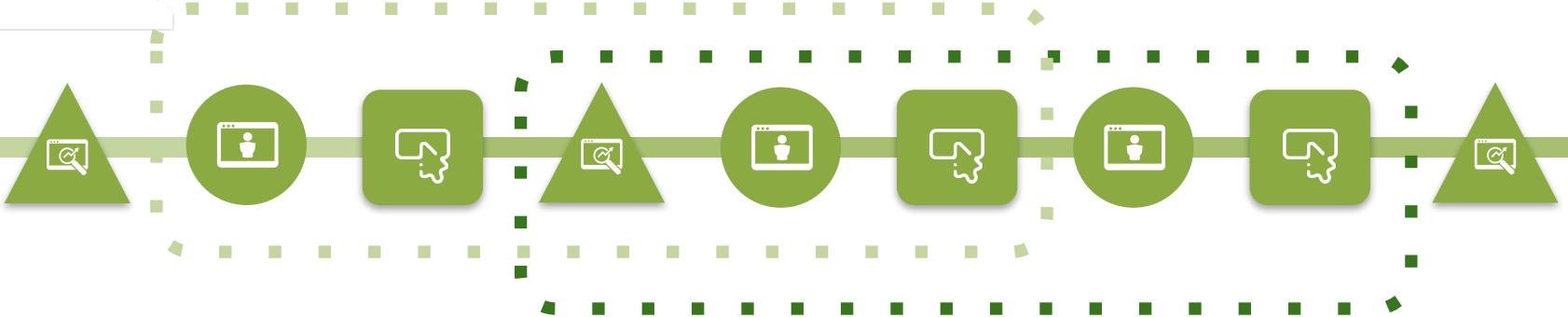
Challenges with existing API

- No state tracking
- Do not support complex customized business logic

Sliding Window



Sliding Window



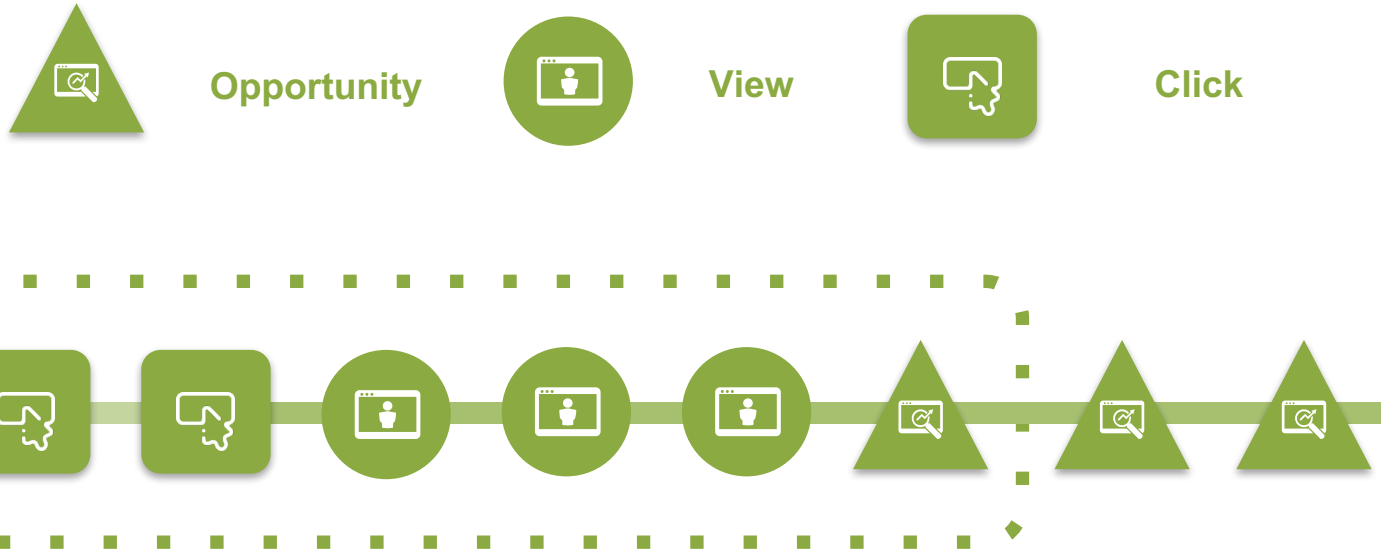
Sliding Window in Spark

?

Challenges with existing API

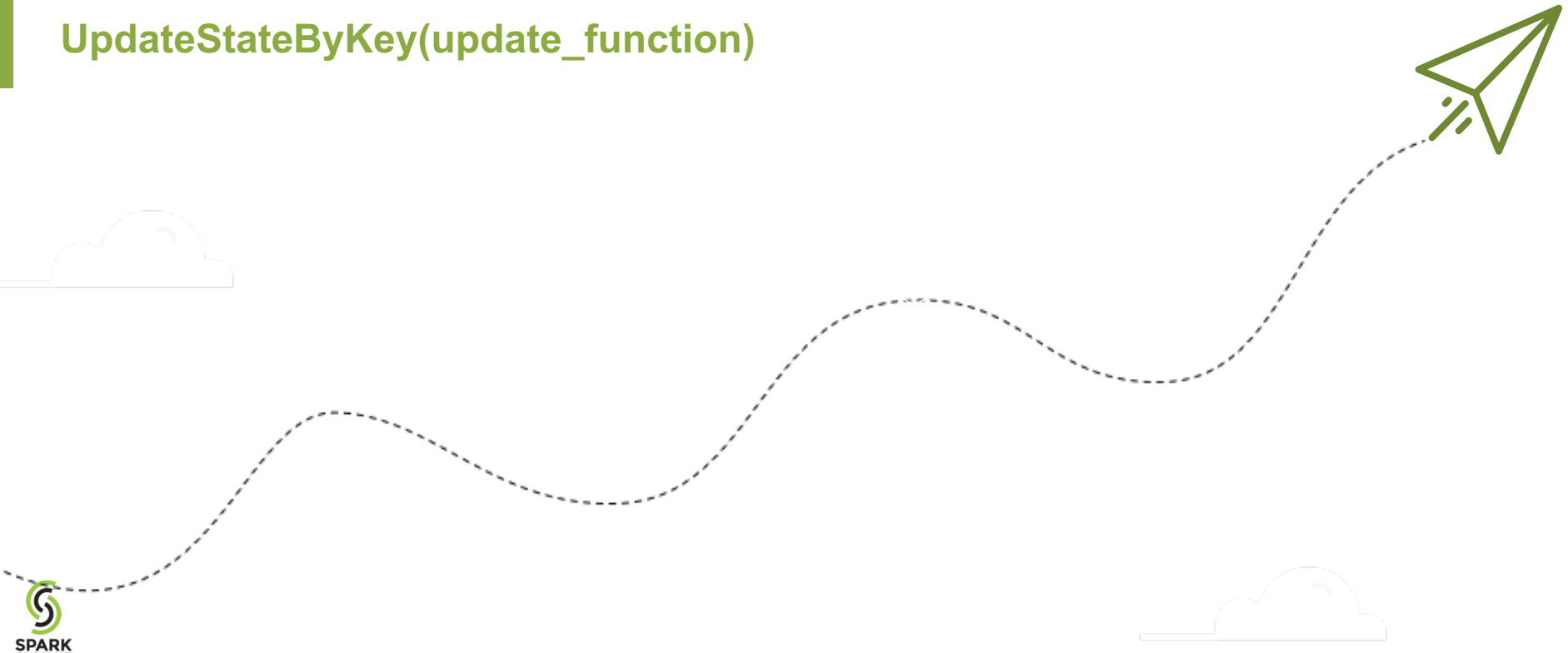
- No state tracking
- Do not support complex customized business logic

Sliding Window



Our solution

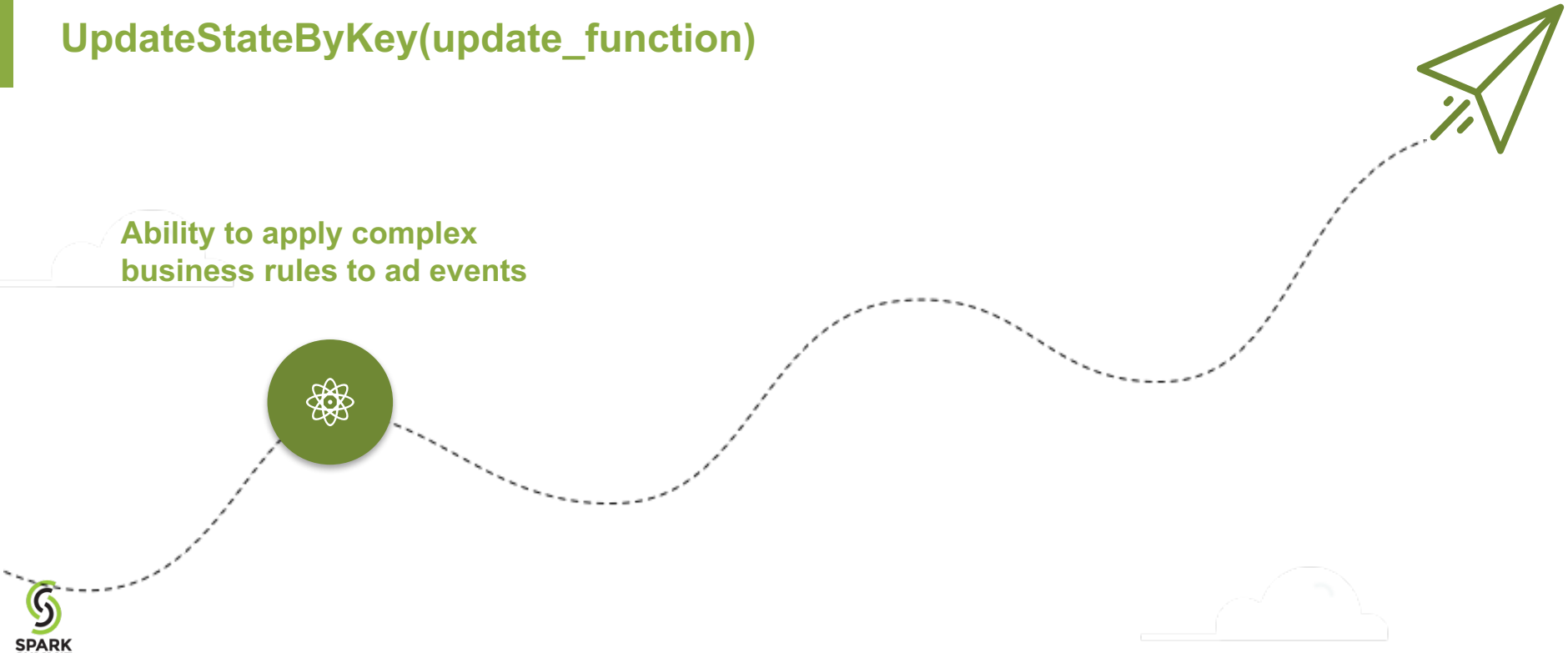
UpdateStateByKey(update_function)



Our solution

UpdateStateByKey(update_function)

Ability to apply complex
business rules to ad events

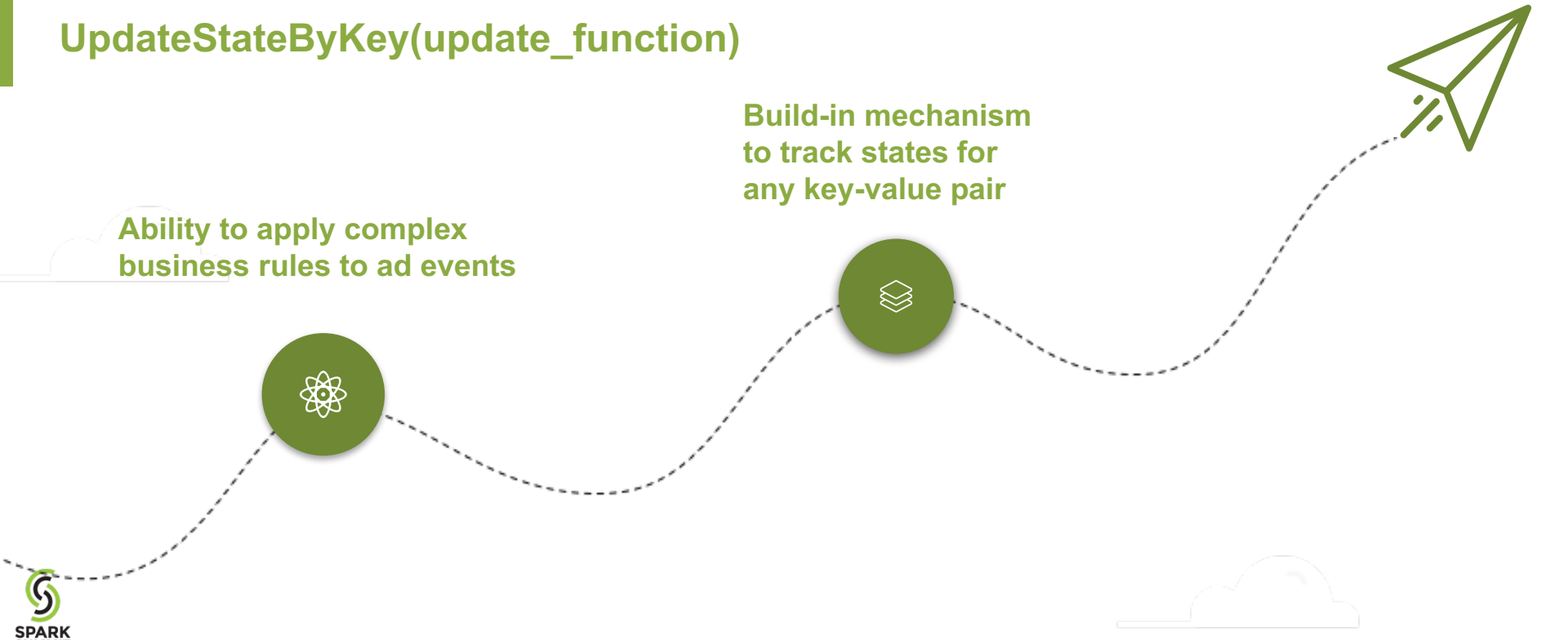


Our solution

UpdateStateByKey(update_function)

Build-in mechanism
to track states for
any key-value pair

Ability to apply complex
business rules to ad events



Our solution

1

Attach expire date/time when events are first seen & state is initialized

2

Drop the state if it expires

3

Apply business logic to new events + current state

Our solution

1

Attach expire date/time when events are first seen & state is initialized

2

Drop the state if it expires

3

Apply business logic to new events + current state

Our solution

1

Attach expire date/time when events are first seen & state is initialized

2

Drop the state if it expires

3

Apply business logic to new events + current state

Pseudo code:

```
def update_function(new_events, current_state):
```

Pseudo code:

```
def update_function(new_events, current_state):

    if current_state is None:
        current_state = init_state()
        attach_expire_datetime(current_state)
```

Pseudo code:

```
def update_function(new_events, current_state):

    if current_state is None:
        current_state = init_state()
        attach_expire_datetime(current_state)


    ......
    if is_expired(current_state):
        return None
```

Pseudo code:

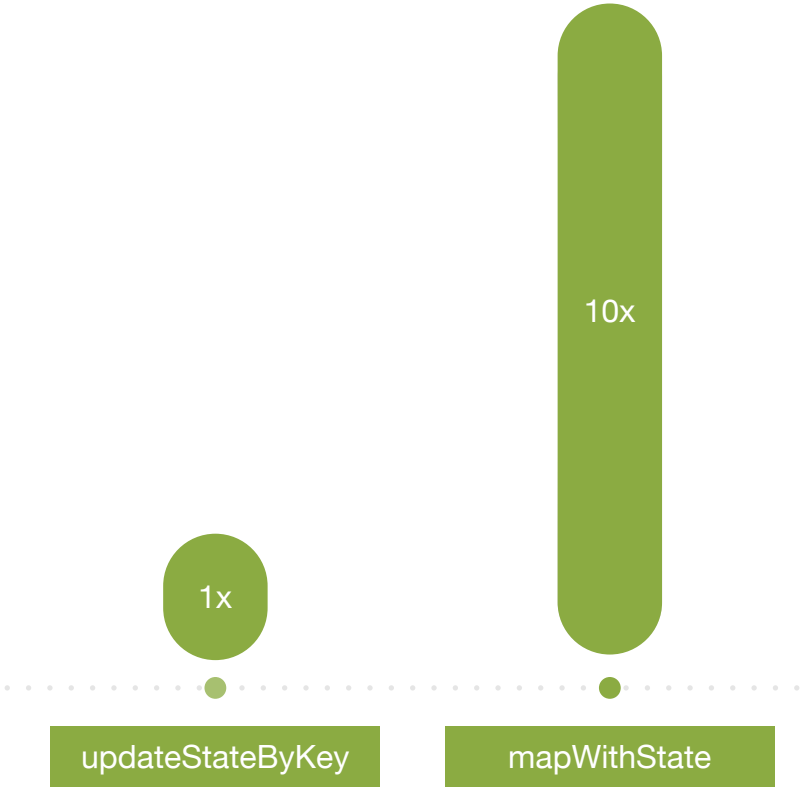
```
def update_function(new_events, current_state):

    if current_state is None:
        current_state = init_state()
        attach_expire_datetime(new_events)


    ......

    if is_expired(current_state):
        return None


    ......

    if new_events:
        apply_business_logic(new_events, current_state)
```

Downside

Spark will apply the state update function for all existing keys

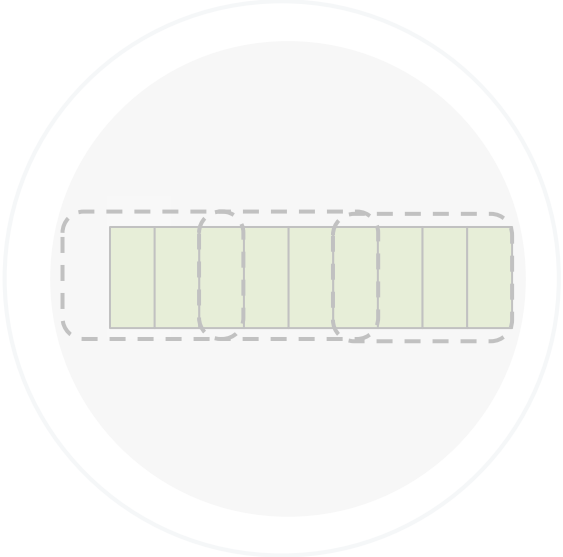
Return early in your update_function()

Improvement in Spark 1.6+

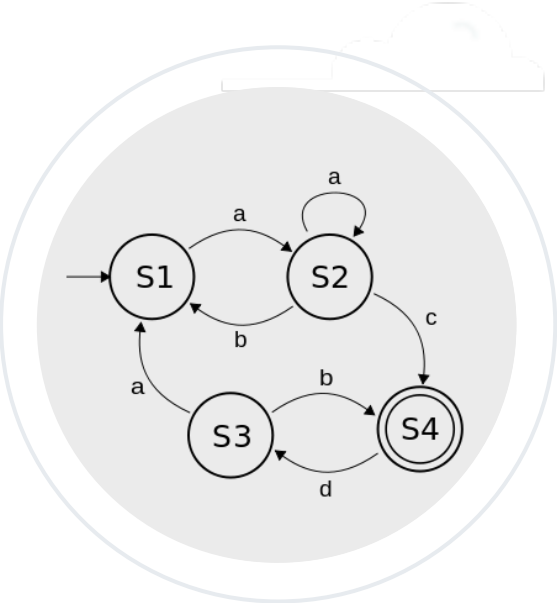


https://databricks.com/blog/2016/02/01/faster-stateful-stream-processing-in-apache-spark-streaming.html

Yelp Ads

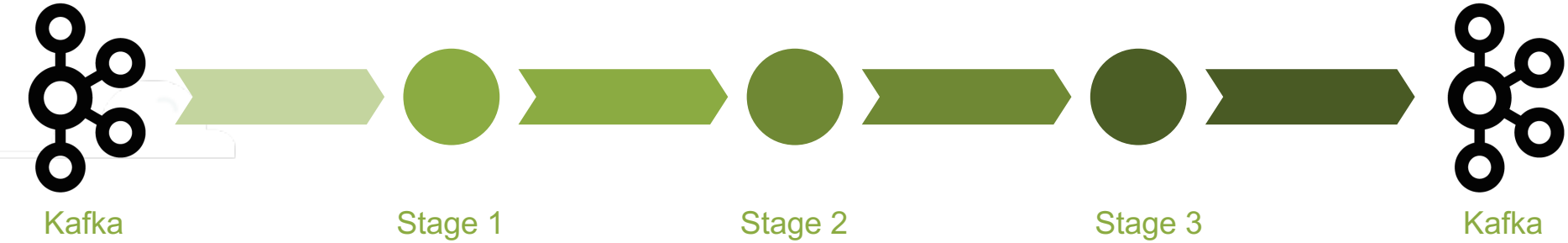


Sliding Window

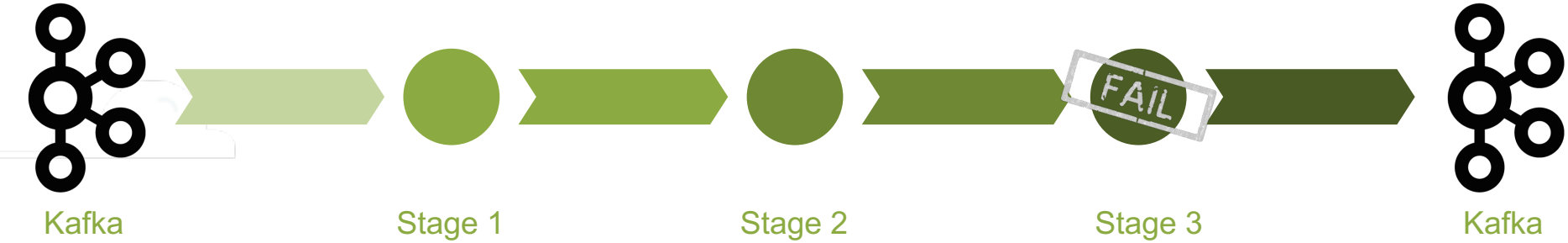


State Management

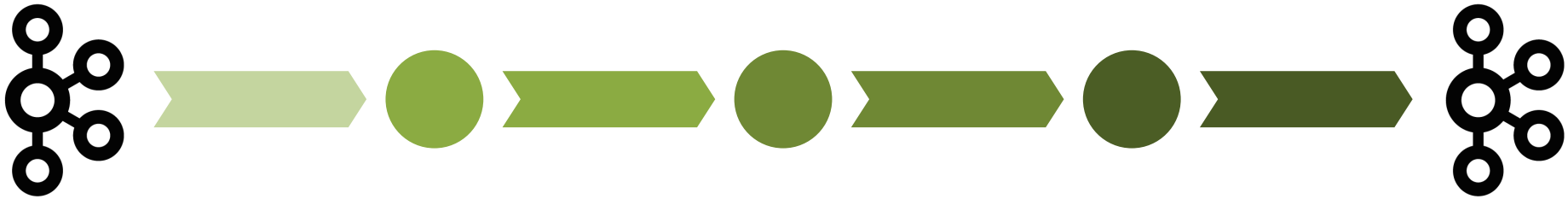
Dependency graph



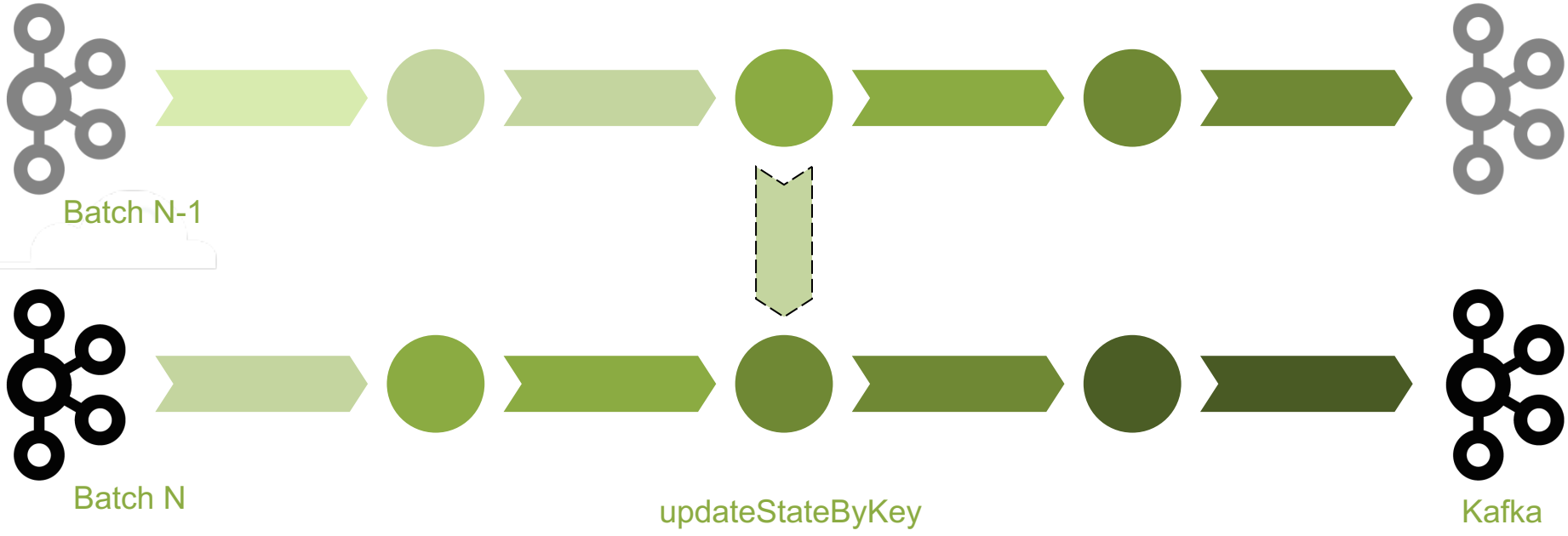
Dependency graph



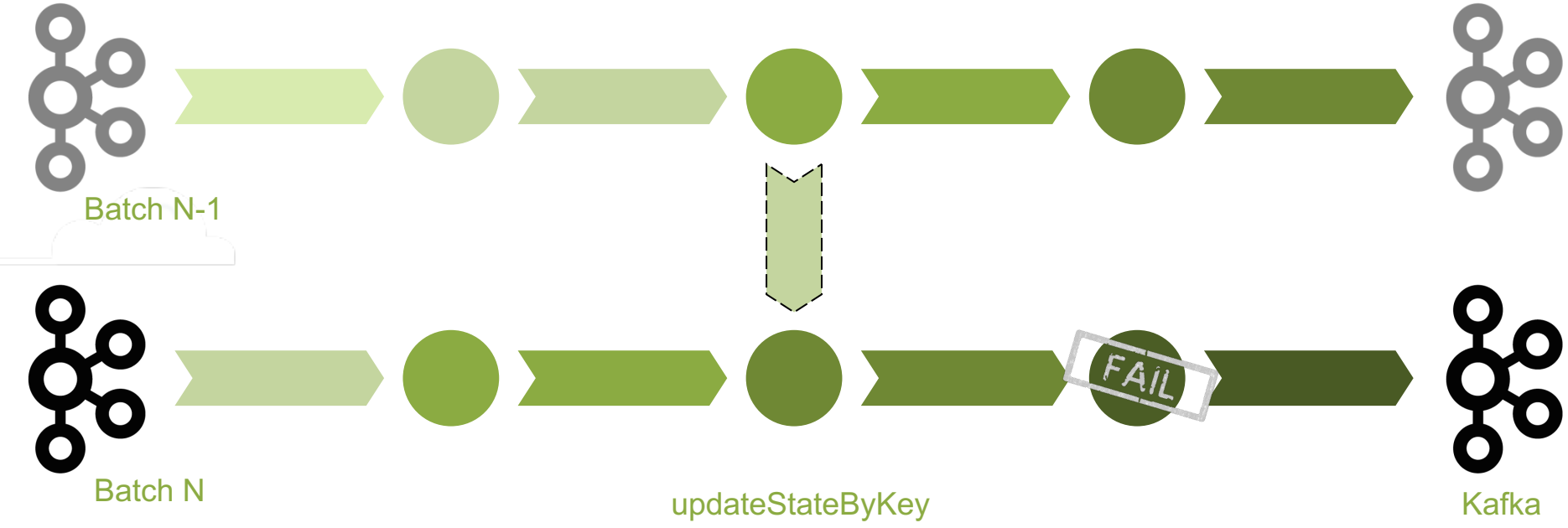
Dependency graph (stateful)



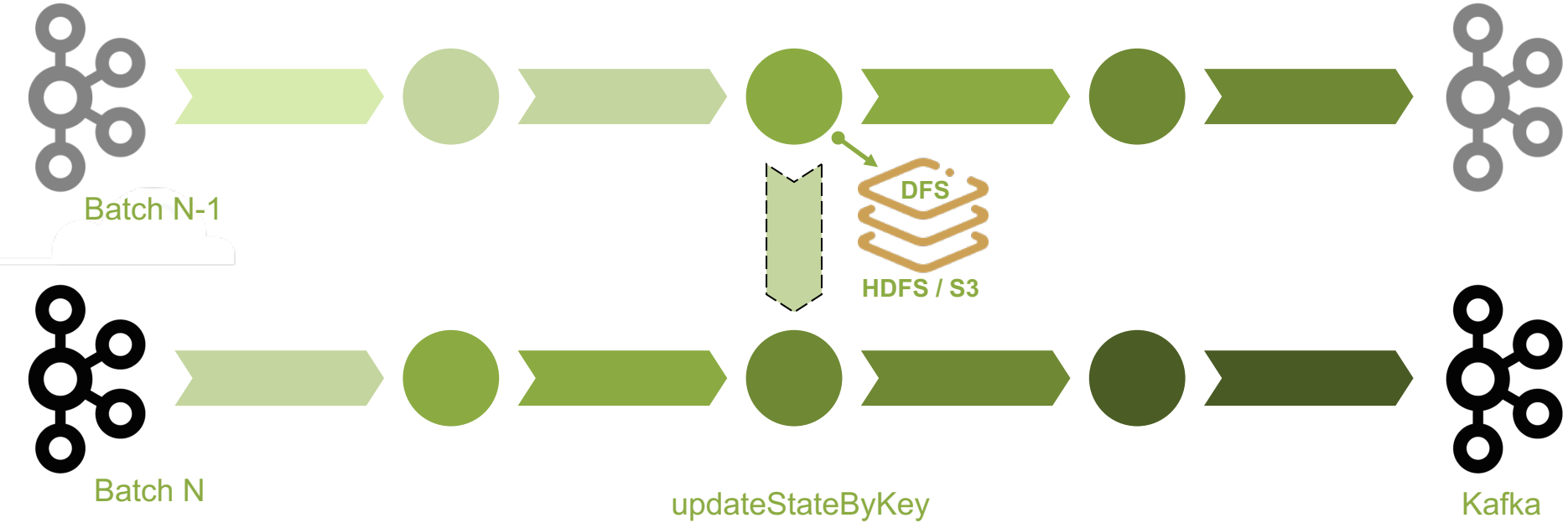
Dependency graph (stateful)



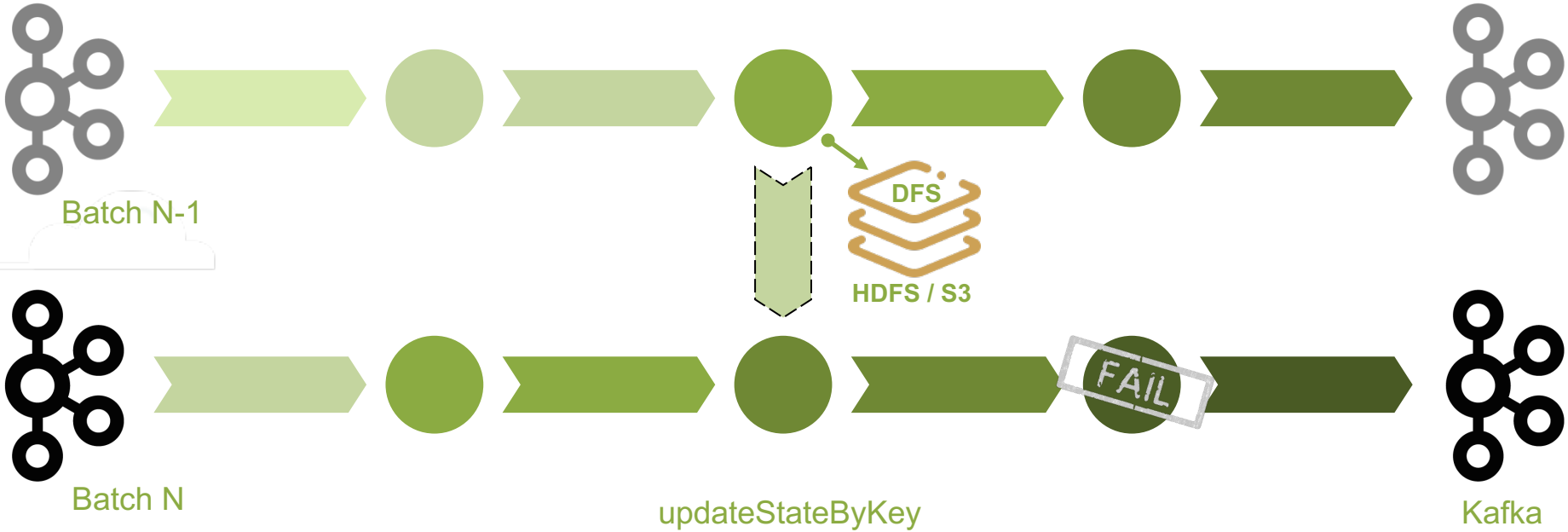
Dependency graph (stateful)



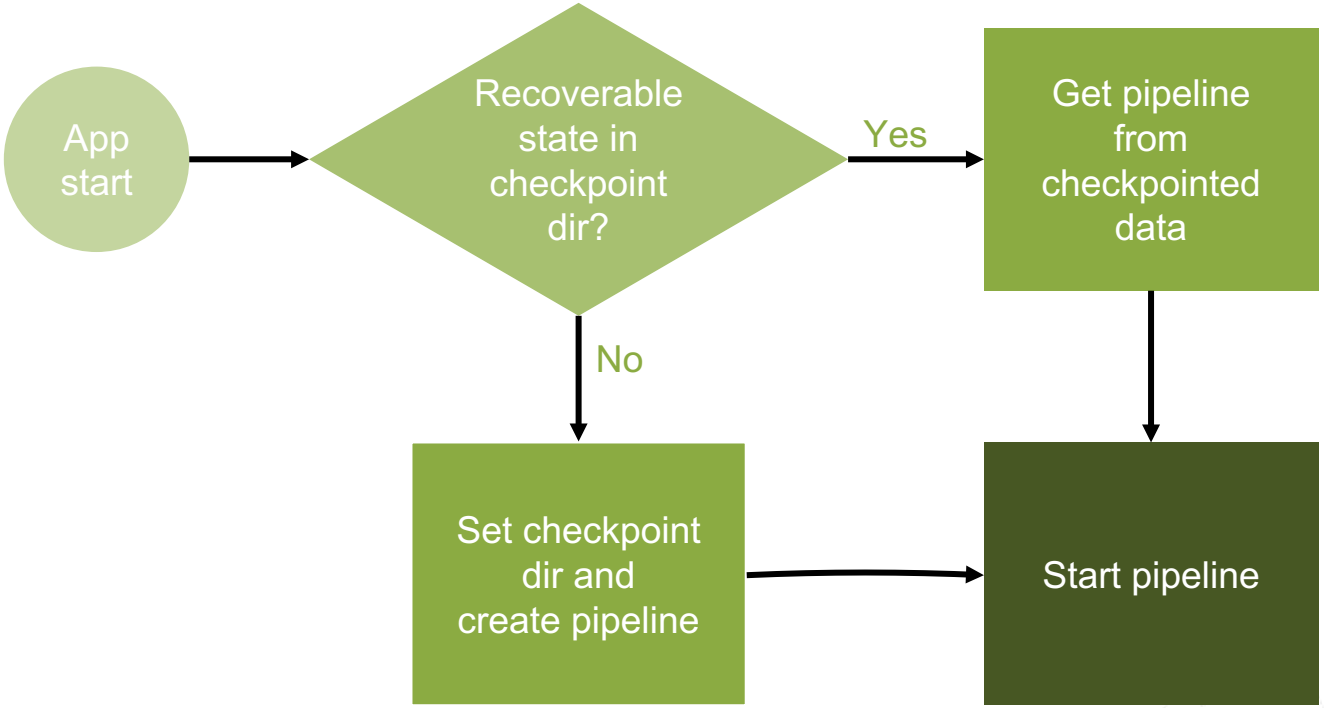
Checkpointing



Checkpointing



Checkpoint recovery

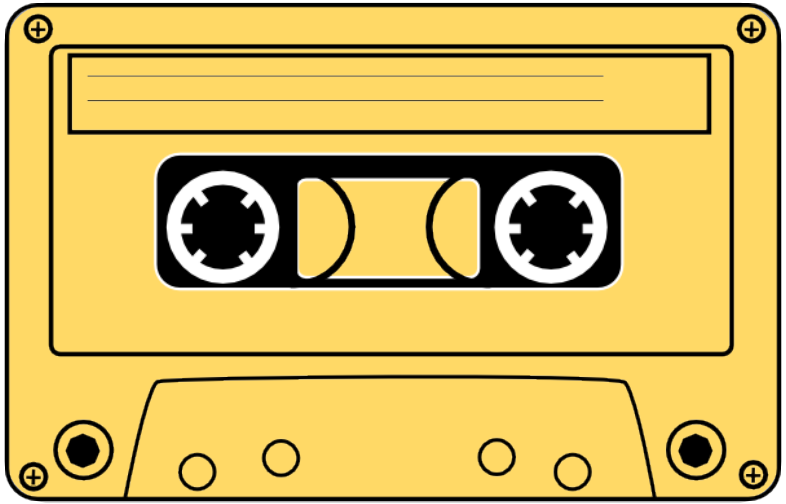


```
if __name__ == '__main__':
    ssc = StreamingContext.getOrCreate(CHECKPOINT_DIR, create_ssc)
    ssc.start()
    ssc.awaitTermination()
```

```
if __name__ == '__main__':
    ssc = StreamingContext.getOrCreate(CHECKPOINT_DIR, create_ssc)
    ssc.start()
    ssc.awaitTermination()


def create_ssc():
    sc = SparkContext()
    ssc = StreamingContext(sc, BATCH_INTERVAL)
    ssc.checkpoint(CHECKPOINT_DIR)
    setup_pipeline(ssc)
    return ssc
```

Checkpointing and recovery

Restoring state via data replay

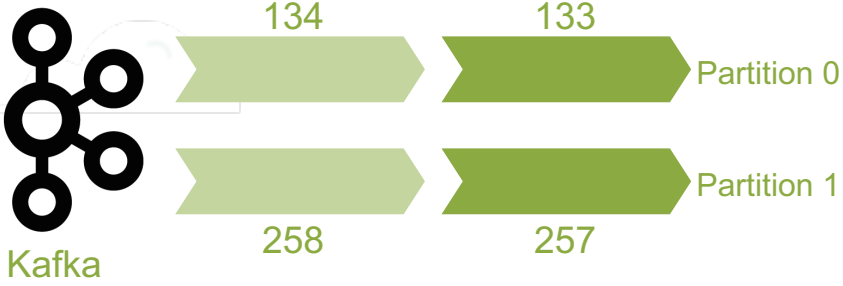


Replay how much?

updateStateByKey

```
{
  ‘key_1’: [
    {‘event’: ‘event_15’},
    {‘event’: ‘event_20’},
  ],
  ‘key_2’: [
    {‘event’: ‘event_23’},
    {‘event’: ‘event_13’},
  ],
}
```

Replay from where?



updateStateByKey

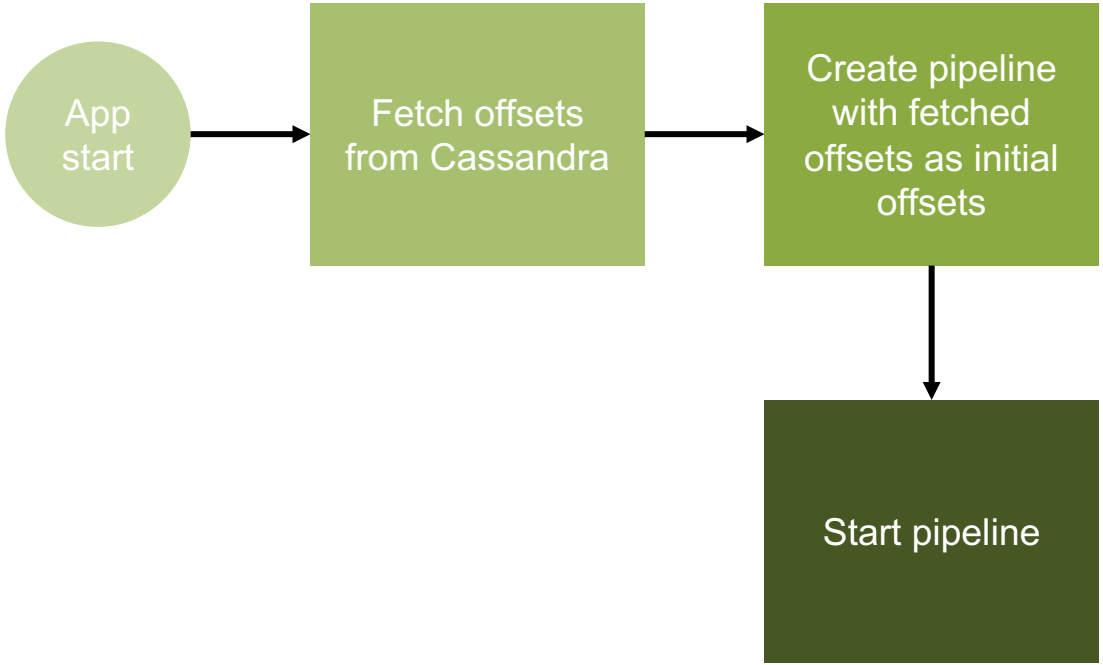
```
{
  ‘key_1’: [
    {‘event’: ‘...’, ‘partition’: 0, ‘offset’: 130},
    {‘event’: ‘...’, ‘partition’: 1, ‘offset’: 230},
  ],
  ‘key_2’: [
    {‘event’: ‘...’, ‘partition’: 0, ‘offset’: 119},
    {‘event’: ‘...’, ‘partition’: 1, ‘offset’: 251},
  ],
}
```

Externalizing partial state

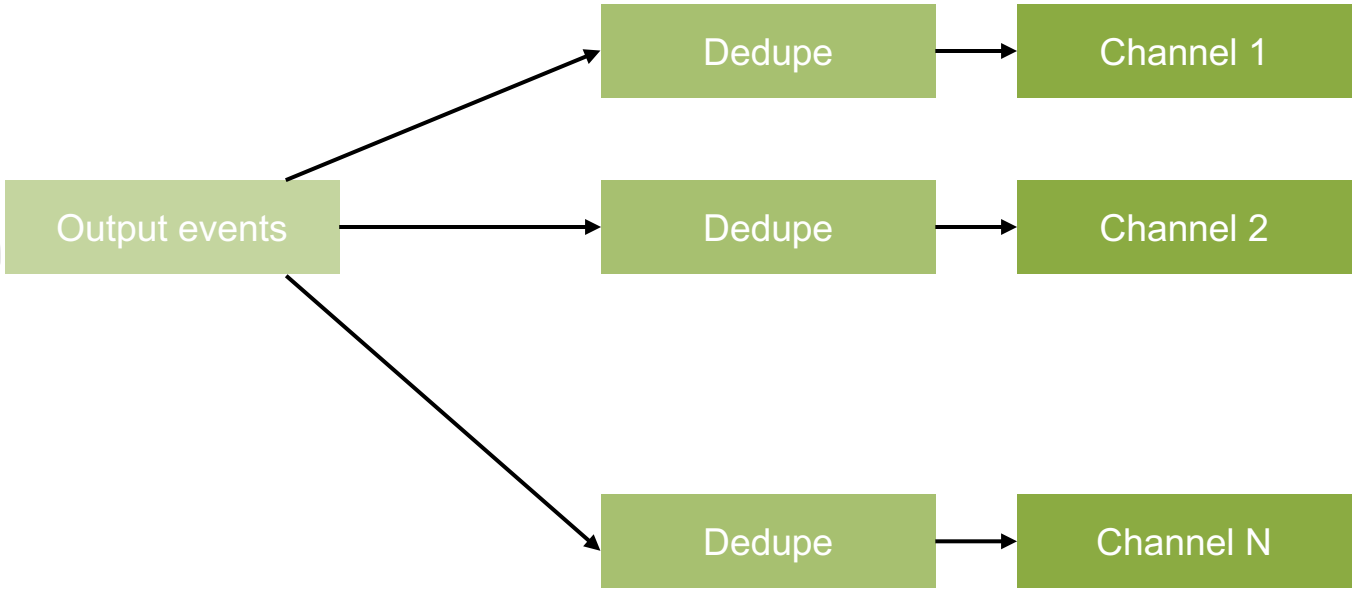
updateStateByKey

```
{
  ‘key_1’: [
    {‘event’: ‘...’, ‘partition’: 0, ‘offset’: 130},
    {‘event’: ‘...’, ‘partition’: 1, ‘offset’: 230},
  ],
  ‘key_2’: [
    {‘event’: ‘...’, ‘partition’: 0, ‘offset’: 119},
    {‘event’: ‘...’, ‘partition’: 1, ‘offset’: 251},
  ],
}
```

Recovery via replay



Dealing with duplicates



Downsides

- Need to handle large data volume after deployments
- Size of state is limited by ability to playback

Looking forward...

- Possibly externalize state entirely
- Structured streaming may fix this… eventually

Conclusion

- Use mapWithState / updateStateByKey to deal with out-of-order events or complex logic needs
- State is not carried over after code changes
- Structured streaming may provide a better solution in the future


We're Hiring!
www.yelp.com/careers/

fb.com/YelpEngineers

@YelpEngineering

engineeringblog.yelp.com

github.com/yelp

Thank you!

- Yelp Ad Targeting at Scale with Apache Spark
  - Tomorrow (June 7th) @ 11.00 am
- Visit us in the expo hall at the yelp booth (T3)