Failure is not fatal: what is your recovery story?

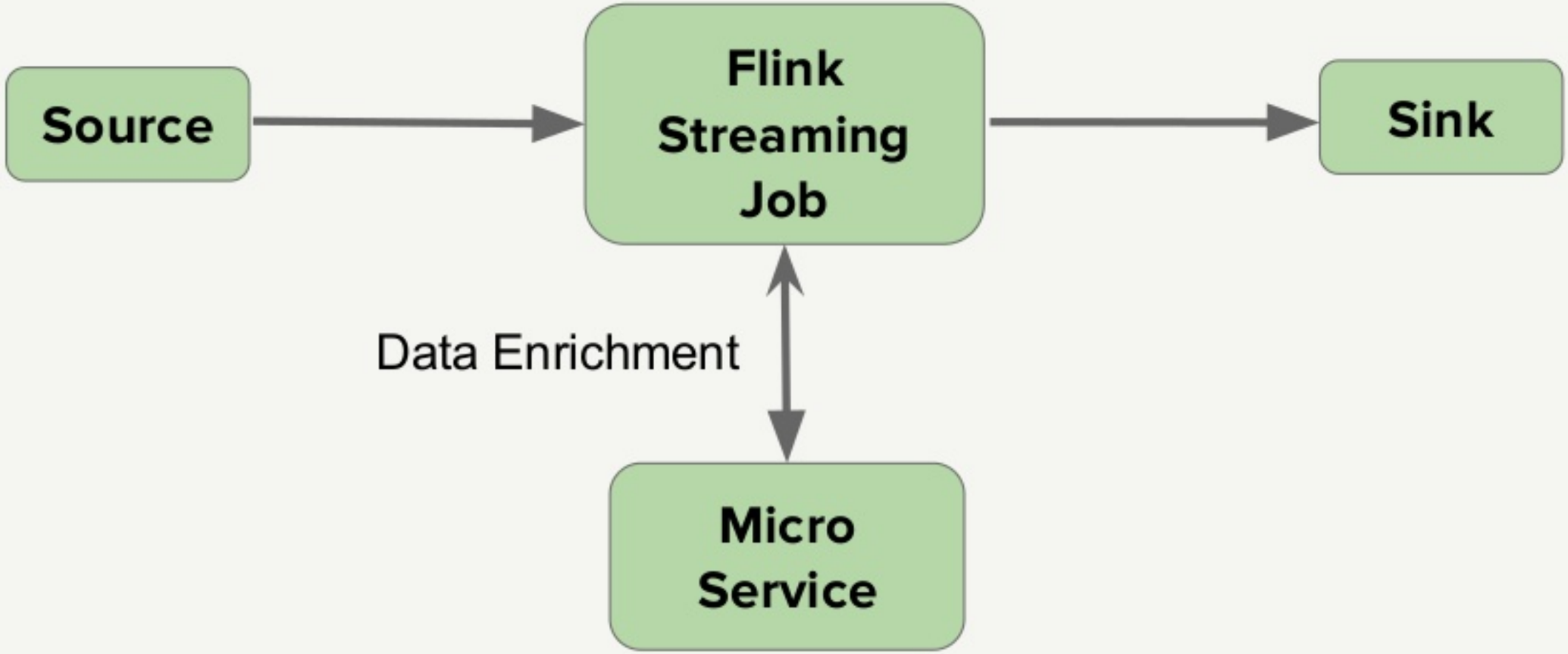
Steven Wu


@stevenzwu

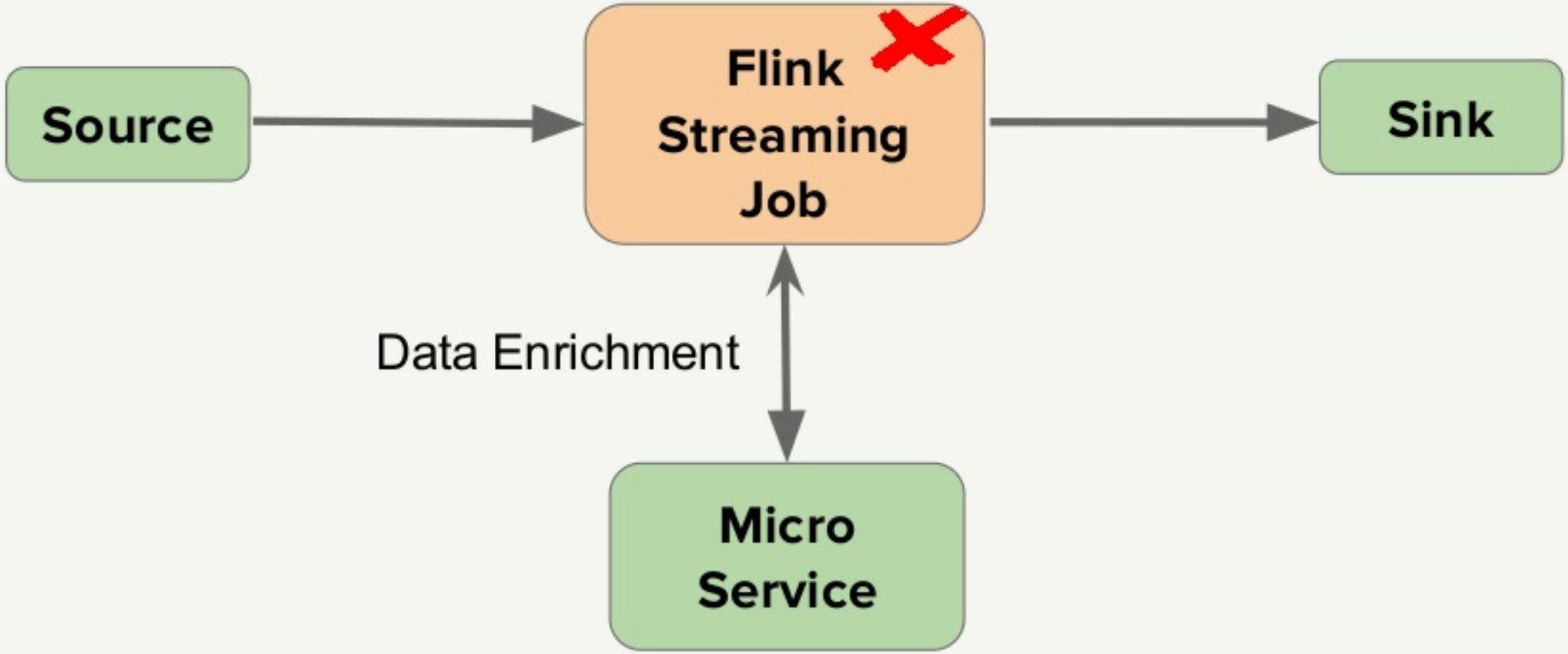
NETFLIX

Failures are part of life!

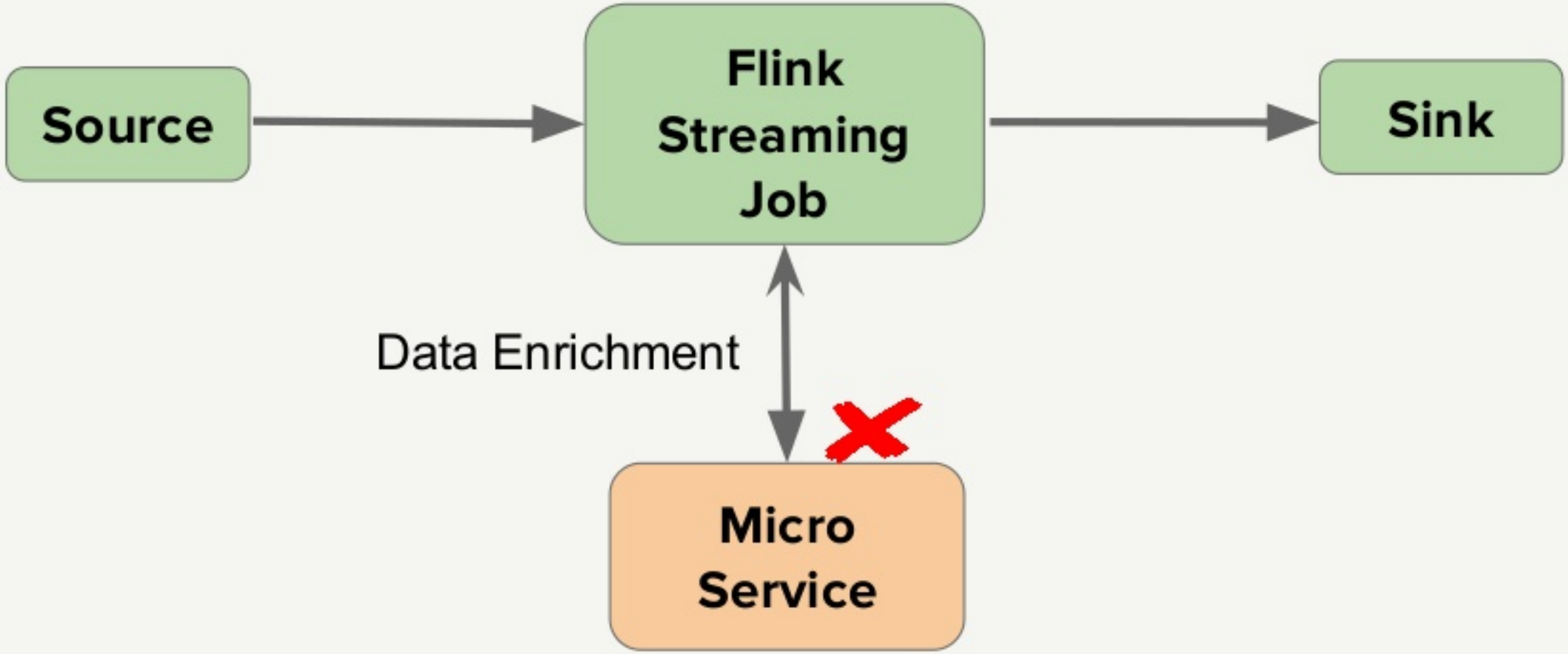
A streaming job can fail



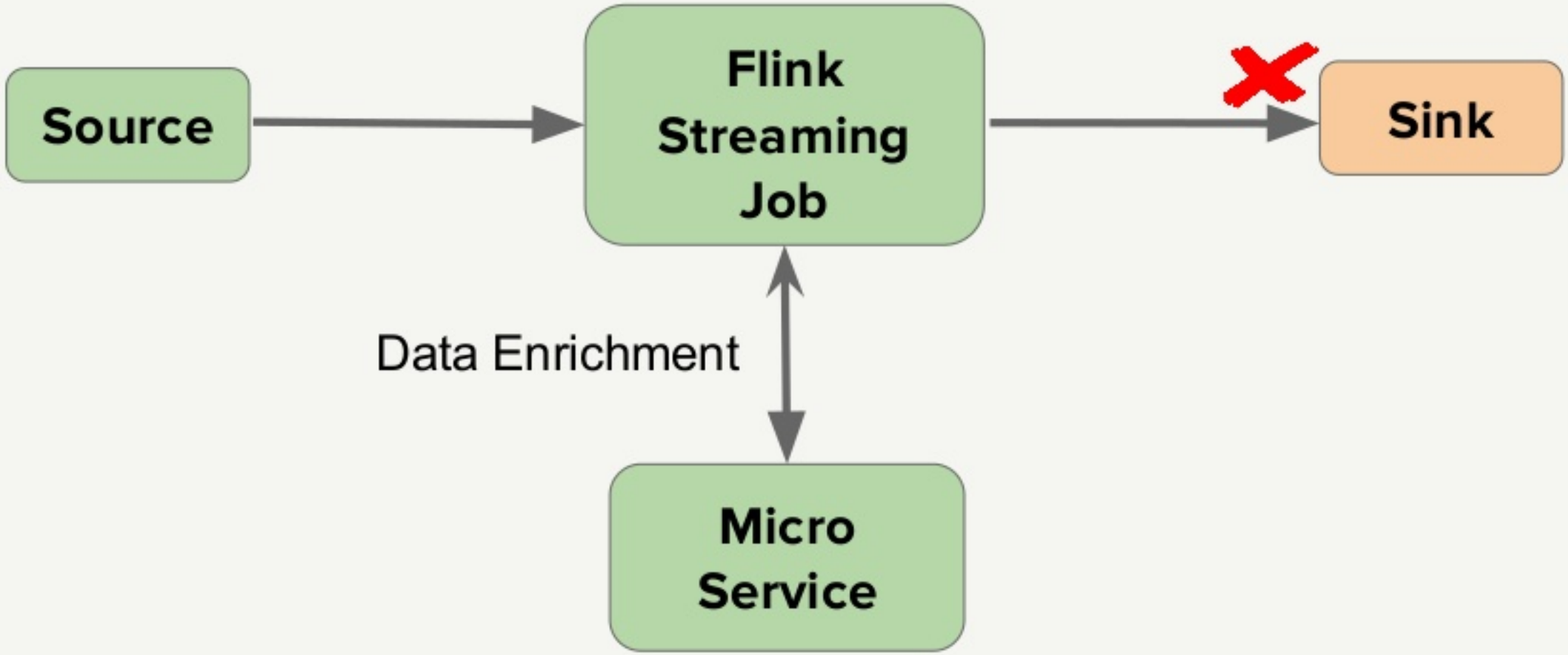
The application can have a bug



The dependency service may return bad data



The sink can fail



How can we recover?

Source: https://pixabay.com/en/man-working-what-to-do-311326/

Agenda

Hive
Backfill

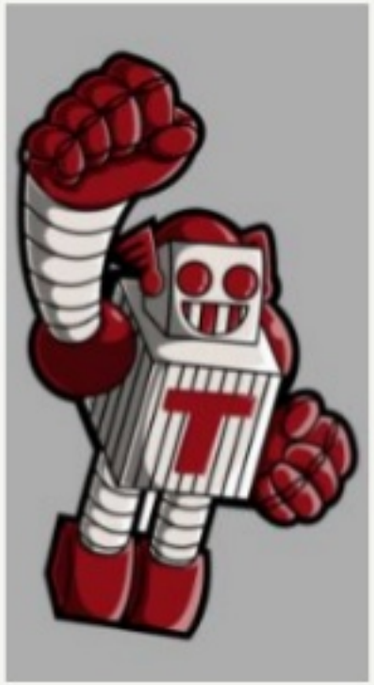
Flink
Rewind

Caveats

We are building a stream processing platform on top of Apache Flink

That integrates with Netflix ecosystem

Spinnaker

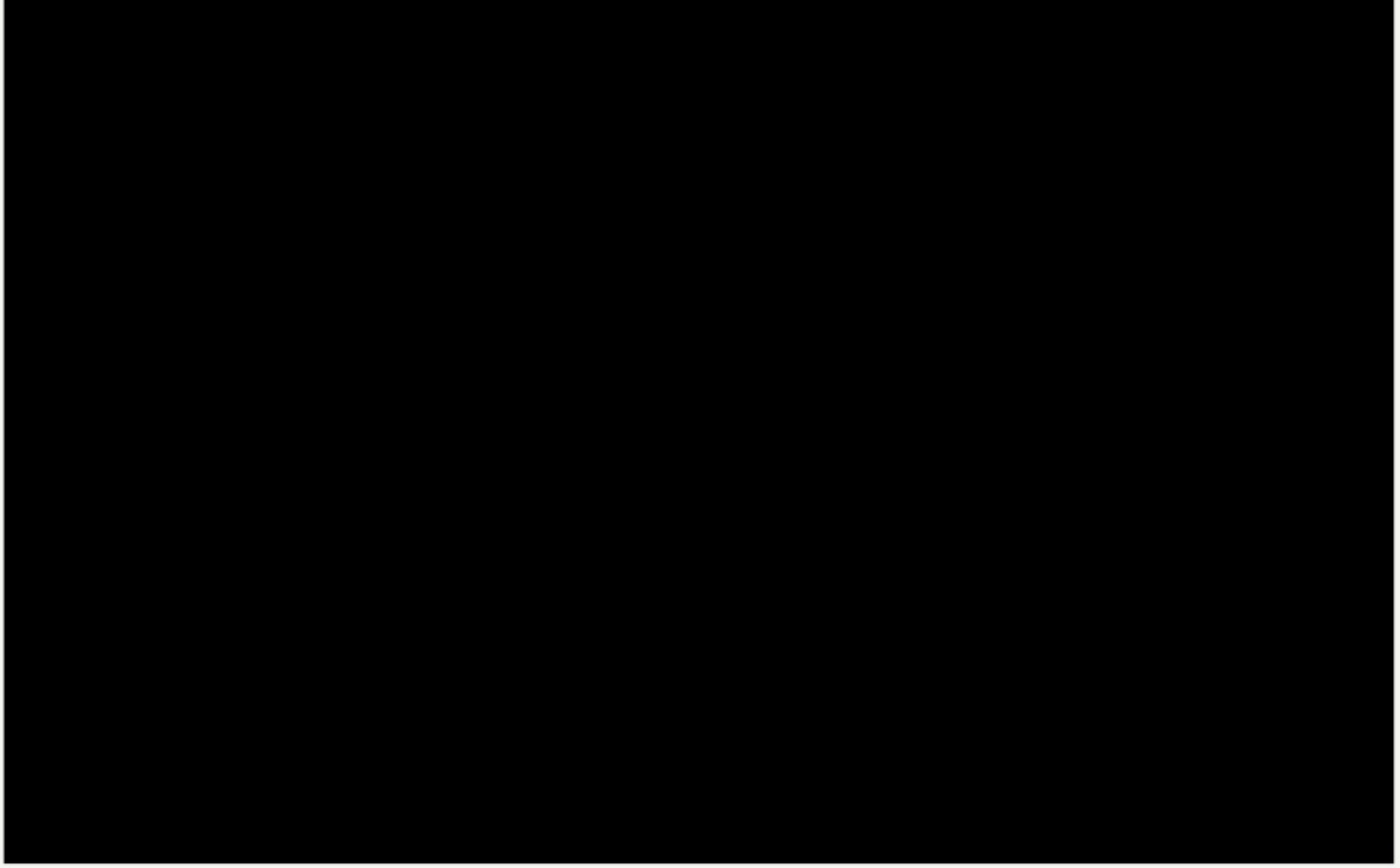


Titus

And
others
....

Geico caveman, https://memegenerator.net

Demo: how to bootstrap new project



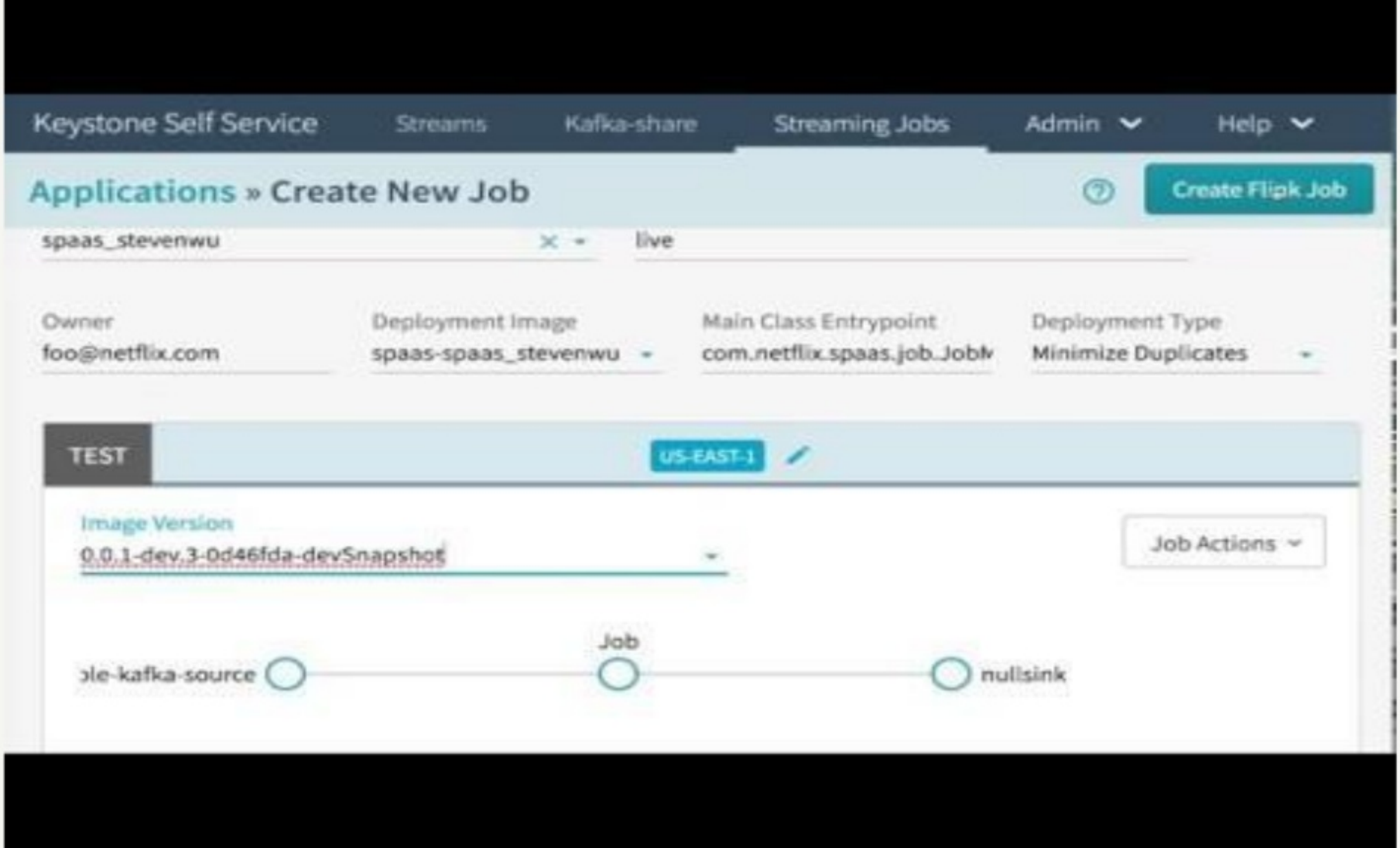
This is the generated skeleton code

```
createSource("example-kafka-source")
        .addSink(getSink("null-sink"))
        .name(“null-sink”);
```

User can add business logic

```
createSource("example-kafka-source")
        .keyBy(<key selector>)
        .window(TumblingProcessingTimeWindows.of(Time.seconds(5)))
        .reduce(<window function>);
        .addSink(getSink(“hive-sink"))
        .name(“hive-sink”);
```

Demo: how to deploy job


Keystone Self Service
Streams
Kafka-share
Streaming Jobs
Admin
Help
Applications » Create New Job
Create Flink Job
spaas_stevenwu
live
Owner
foo@netflix.com
Deployment Image
spaas-spaas_stevenwu
Main Class Entrypoint
com.netflix.spaas.job.JobM
Deployment Type
Minimize Duplicates
TEST
US-EAST-1
Image Version
0.0.1-dev.3-0d46fda-devSnapshot
Job Actions
Job
ole-kafka-source
nullsink

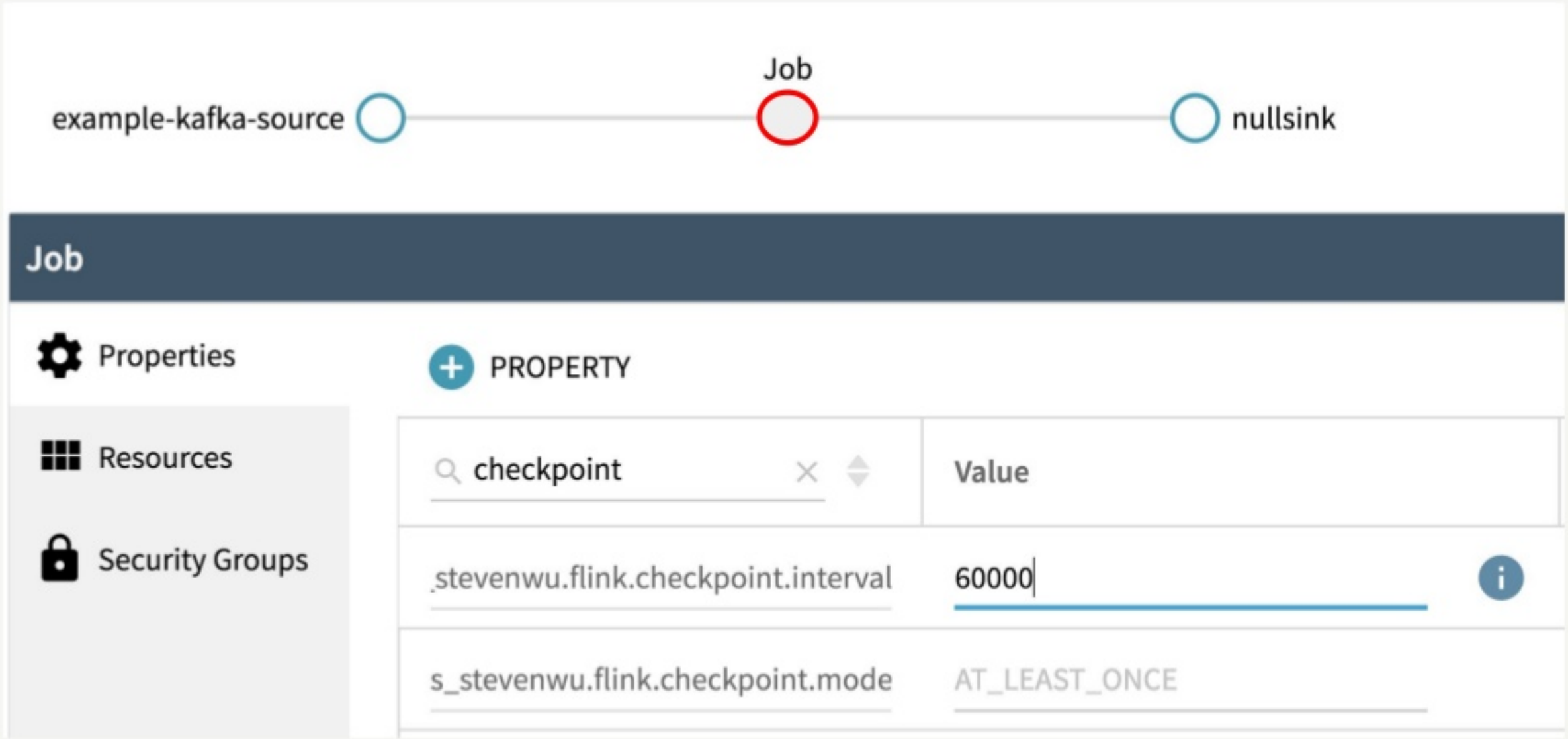
User can override source configuration

Kafka Source - example-kafka-source

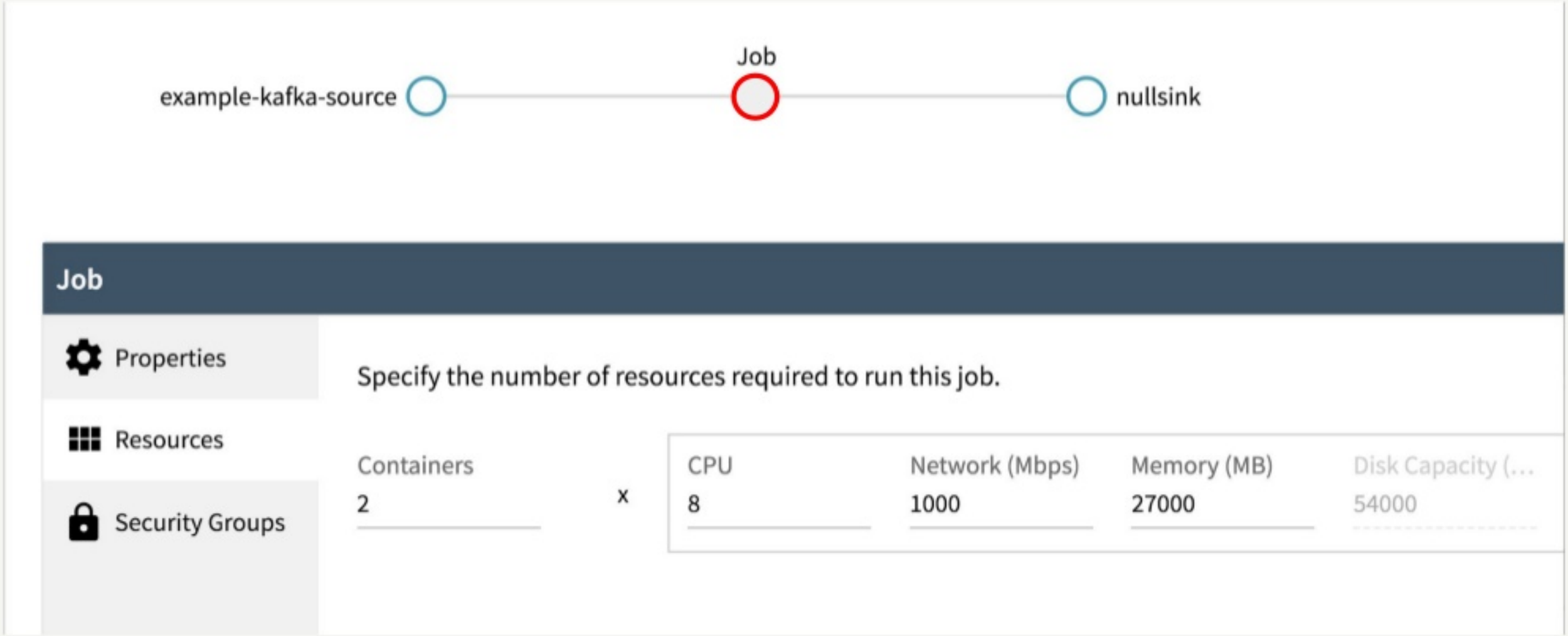
Name	Template Value	Optional Override
Topic Name	clevent_ihs	
Vip	kafka-test:2181	kafka-prod:2181

Override Kafka cluster VIP

User can override any job config



User can configure resources



Agenda

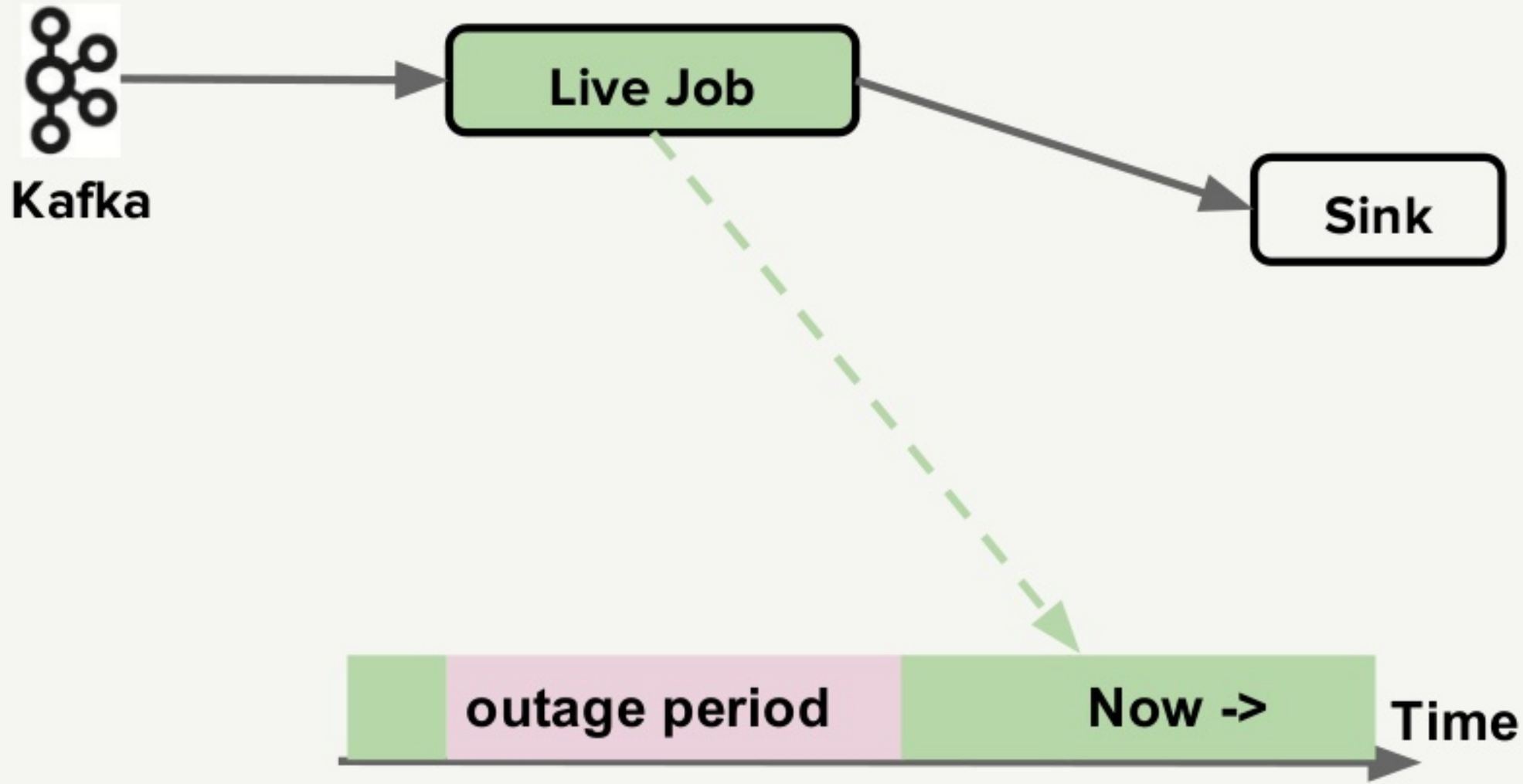
Hive
Backfill

Flink
Rewind

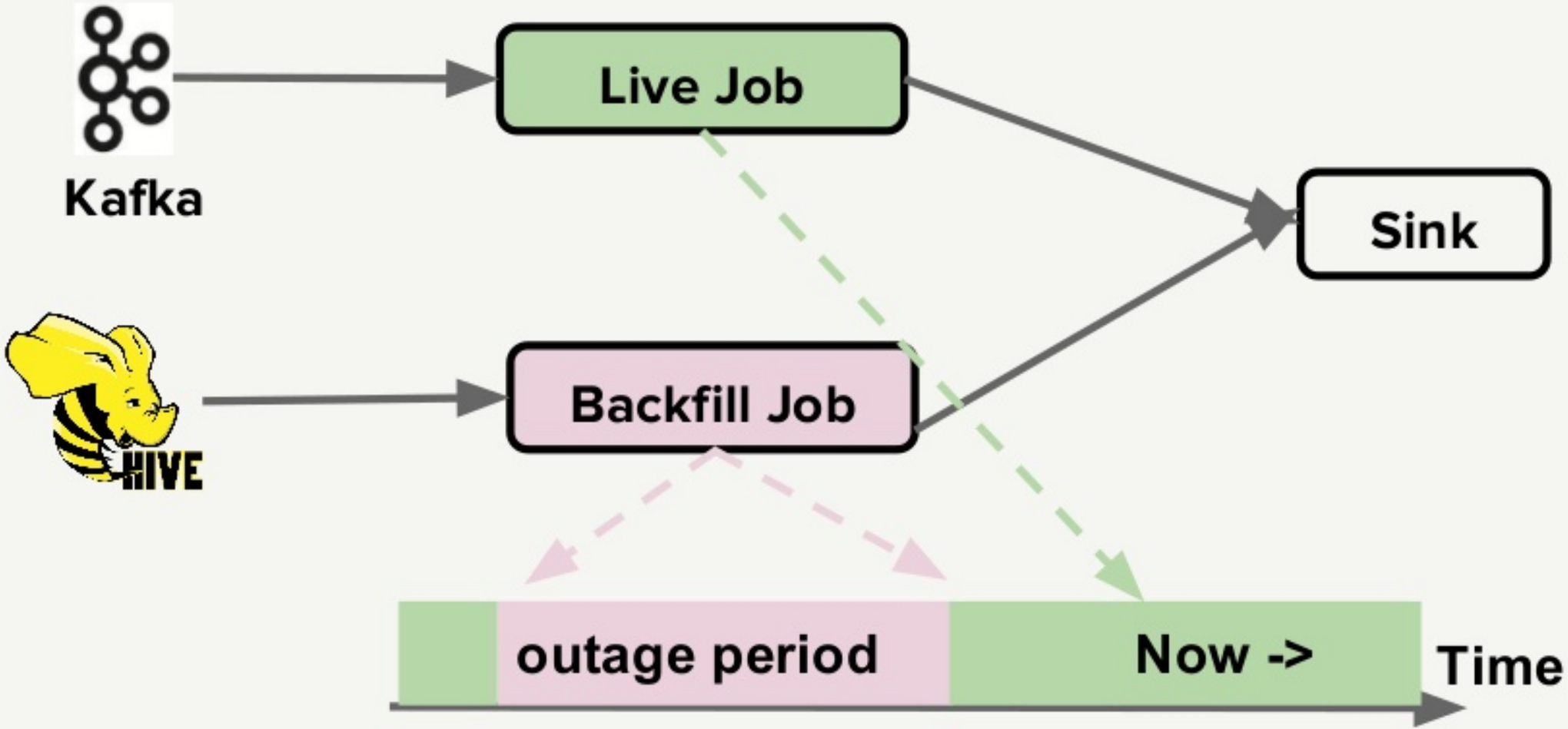
Caveats

Streaming data also go to Hive in addition to Kafka

Live job continues to run



User can start a parallel backfill job reading from Hive



We implemented a Hive source with DataStream API

```
public class HiveSource<OUT>
    extends RichParallelSourceFunction<OUT>
        implements CheckpointedFunction,
                   ResultTypeQueryable<OUT> {
            // ...
}
```

We provide dynamic source that allows user to switch from Kafka to Hive

We provide dynamic source that allows user to switch from Kafka to Hive

```
// build a kafka source
SingleOutputStreamOperator<Record<Map<String, Object>>> kafkaSource =
        getSourceBuilder().fromKafka("kafka").build();
```

(1)

```
// build a hive source
SingleOutputStreamOperator<Record<Map<String, Object>>> hiveSource =
        getSourceBuilder().fromHive("hive").build();
```

(2)

```
// dynamically pick the selected source at job launch time
SingleOutputStreamOperator<Record<Map<String, Object>>> selectedSource =
        getSourceBuilder()
            .fromDynamic("dynamicsource")
            .declareWith("kafka",kafkaSource )
            .or("hive", hiveSource)
            .build();
```

(3)

We provide dynamic source that allows user to switch from Kafka to Hive

```
// build a kafka source
SingleOutputStreamOperator<Record<Map<String, Object>>> kafkaSource =
        getSourceBuilder().fromKafka("kafka").build();
```

(1)

```
// build a hive source
SingleOutputStreamOperator<Record<Map<String, Object>>> hiveSource =
        getSourceBuilder().fromHive("hive").build();
```

(2)

```
// dynamically pick the selected source at job launch time
SingleOutputStreamOperator<Record<Map<String, Object>>> selectedSource =
        getSourceBuilder()
            .fromDynamic("dynamicsource")
            .declareWith("kafka",kafkaSource )
            .or("hive", hiveSource)
            .build();
```

(3)

We provide dynamic source that allows user to switch from Kafka to Hive

```
// build a kafka source
SingleOutputStreamOperator<Record<Map<String, Object>>> kafkaSource =
        getSourceBuilder().fromKafka("kafka").build();
```

(1)

```
// build a hive source
SingleOutputStreamOperator<Record<Map<String, Object>>> hiveSource =
        getSourceBuilder().fromHive("hive").build();
```

(2)

```
// dynamically pick the selected source at job launch time
SingleOutputStreamOperator<Record<Map<String, Object>>> selectedSource =
        getSourceBuilder()
            .fromDynamic("dynamicsource")
            .declareWith("kafka",kafkaSource )
            .or("hive", hiveSource)
            .build();
```

(3)

We provide dynamic source that allows user to switch from Kafka to Hive

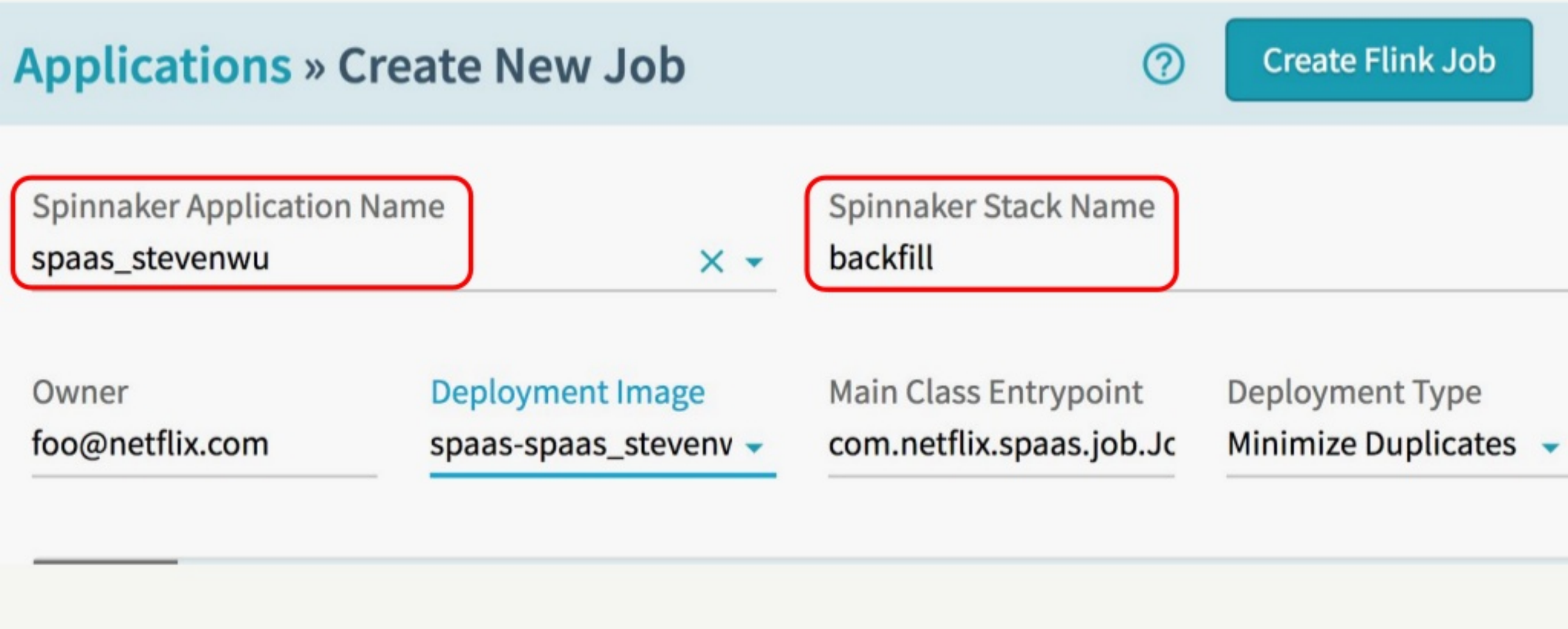
```
// build a kafka source
SingleOutputStreamOperator<Record<Map<String, Object>>> kafkaSource =
        getSourceBuilder().fromKafka("kafka").build();

// build a hive source
SingleOutputStreamOperator<Record<Map<String, Object>>> hiveSource =
        getSourceBuilder().fromHive("hive").build();

// dynamically pick the selected source at job launch time
SingleOutputStreamOperator<Record<Map<String, Object>>> selectedSource =
        getSourceBuilder()
            .fromDynamic("dynamicsource")
            .declareWith("kafka",kafkaSource )
            .or("hive", hiveSource)
            .build();
```

(1)

(2)

(3)

Create a Hive backfill job under the same application as live job



User needs to override selected source

User needs to override selected source

Configure Hive source

.hive_source.hive	database	default
.hive_source.hive	table	clevent
.hive_source.hive	where	dateint=20180201 and hour=0
.hive_source.type		hive

It is NOT a lambda architecture

- Single streaming code base
- Just switch source from Kafka to Hive

Hive backfill is likely not good for stateful jobs

- Warm-up issue
- Ordering issue

Hive backfill is likely not good for stateful jobs

- Warm-up issue
- Ordering issue

For stateful jobs, each input record is evaluated against application state accumulated over time

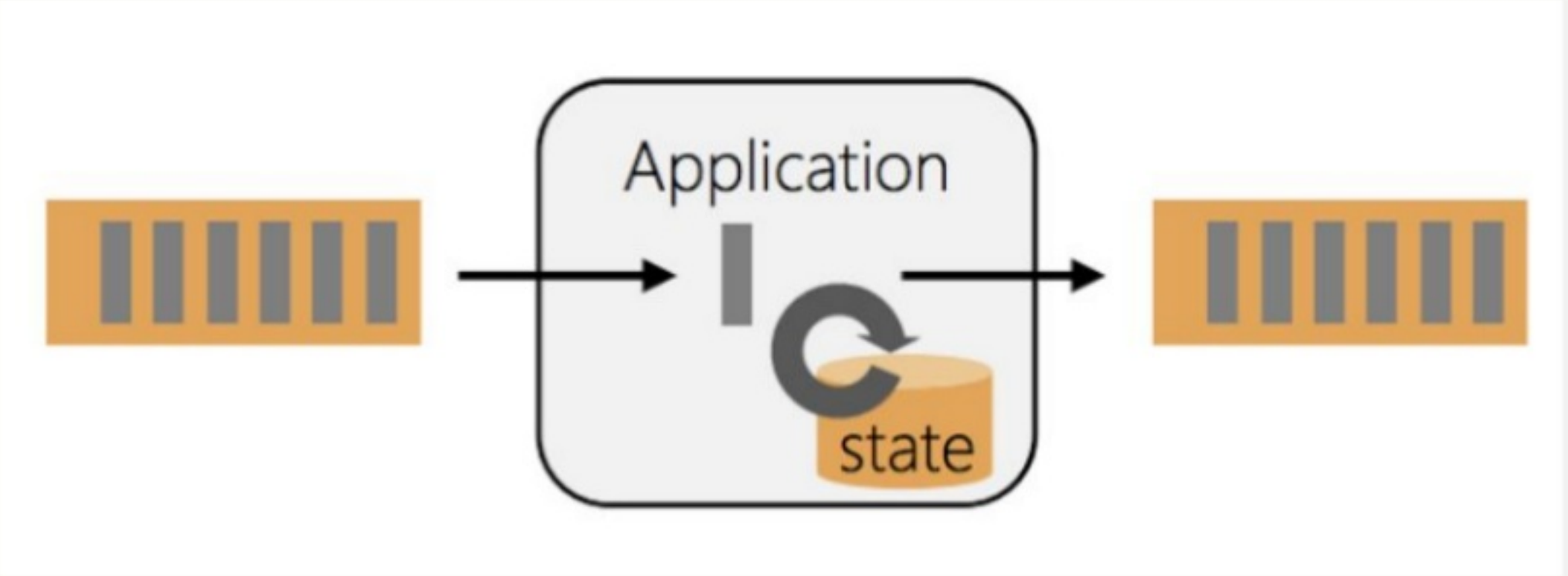
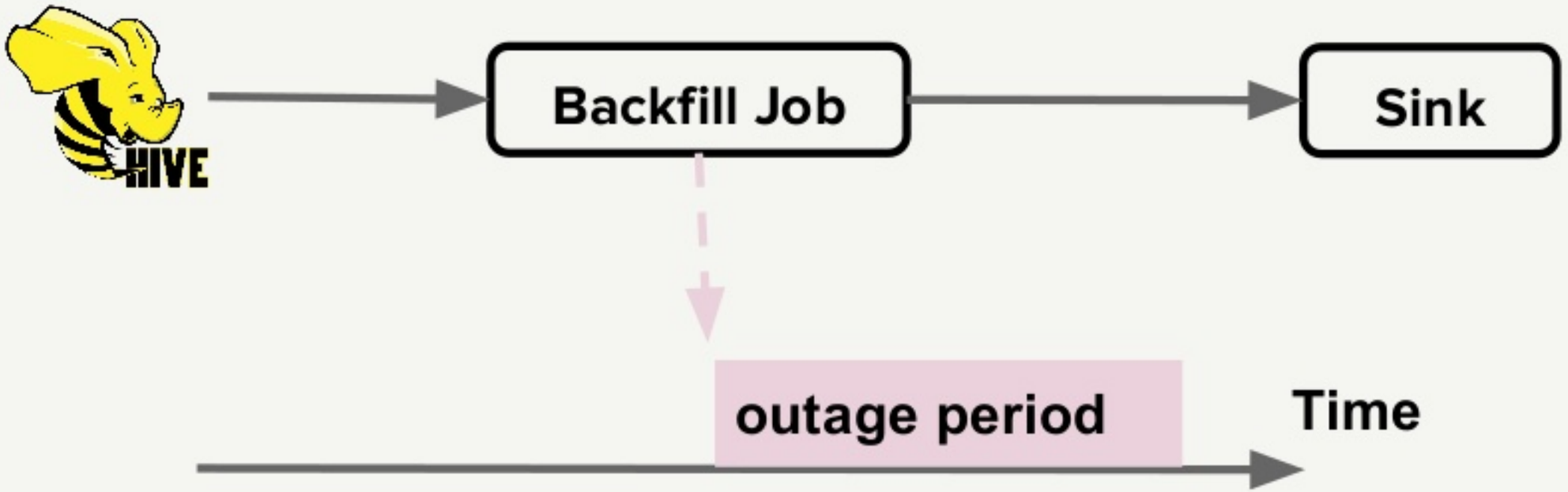
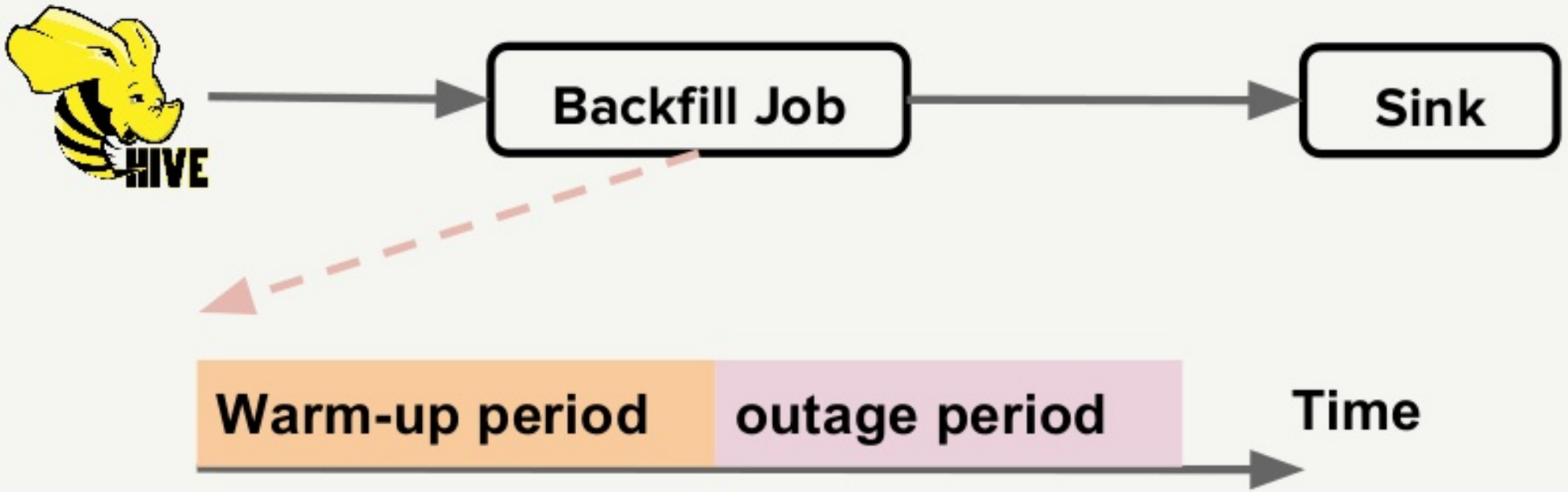


Image adapted from Stephen Ewen

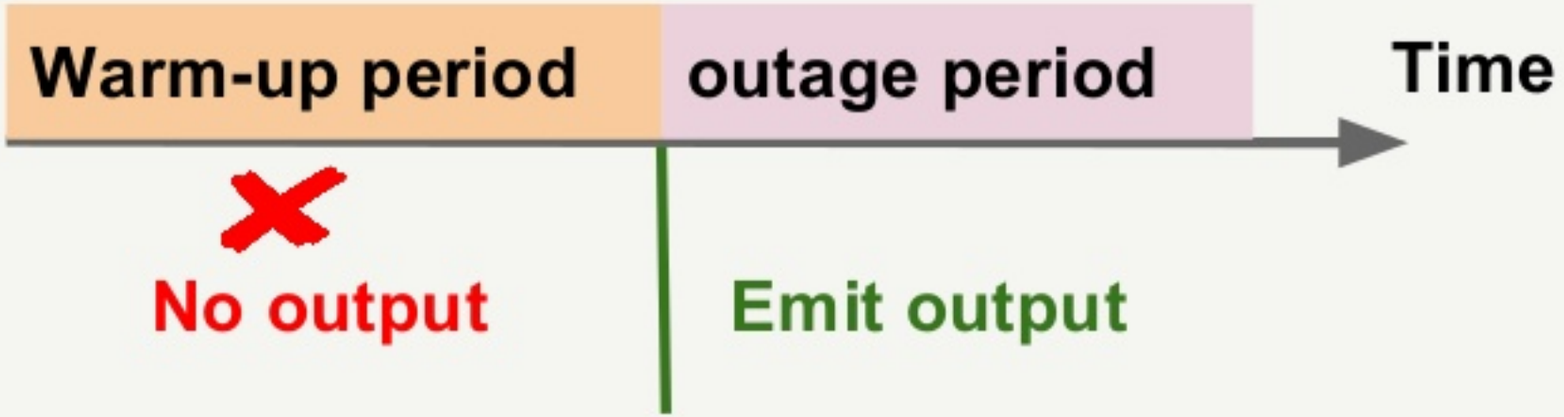
Backfill job started with an empty state



Why don’t we add a warm-up period to build up the proper state



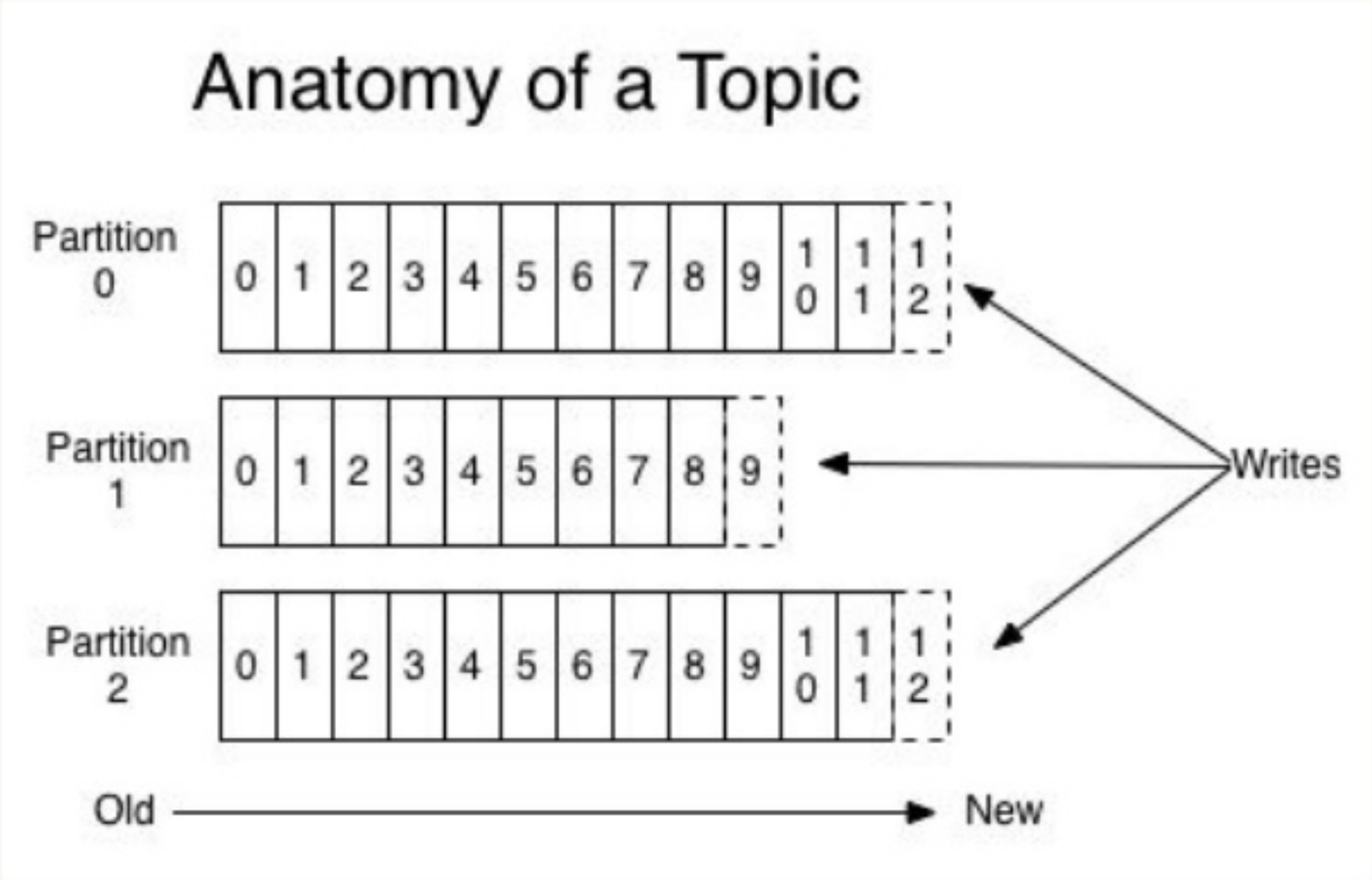
Need to avoid output during warm-up period



Hive backfill is likely not good for stateful jobs

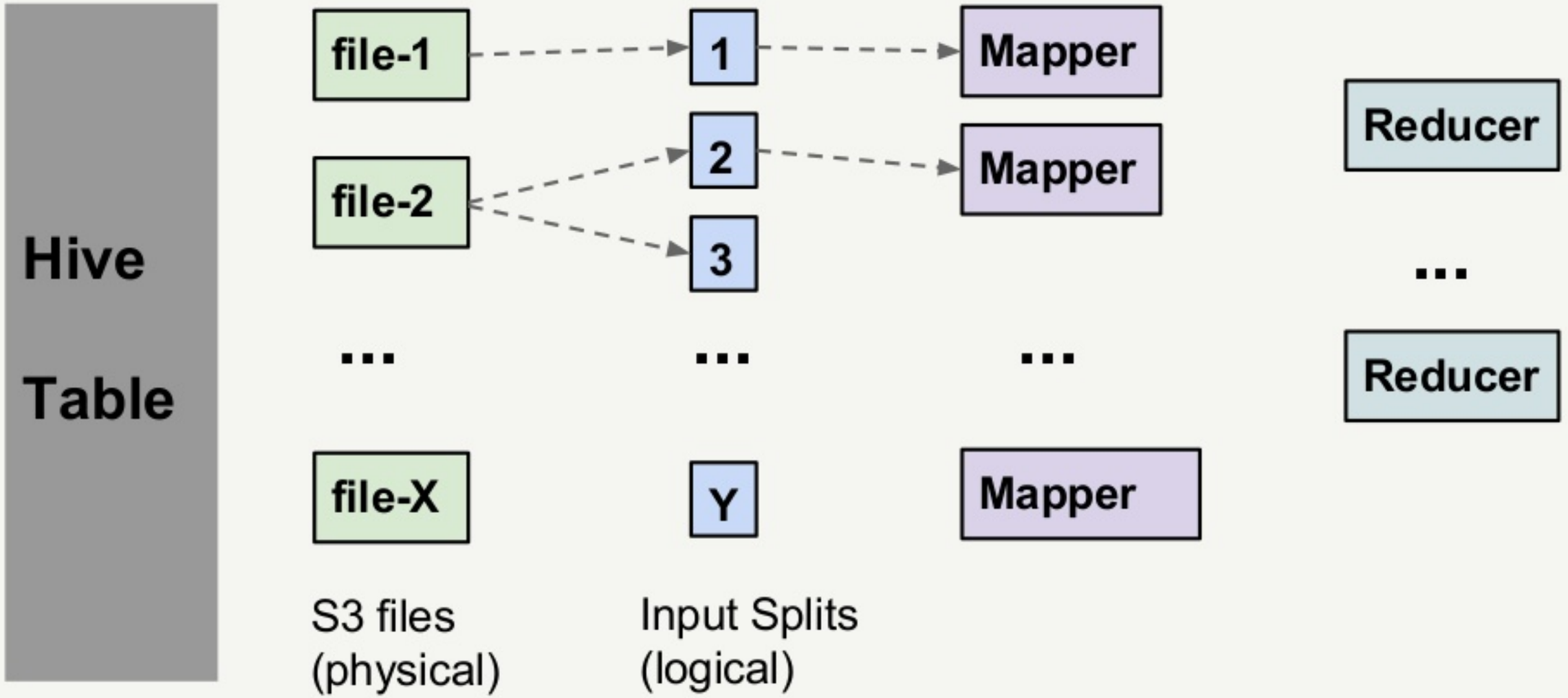
- Warm-up issue
- Ordering issue

Kafka messages are ordered within a partition

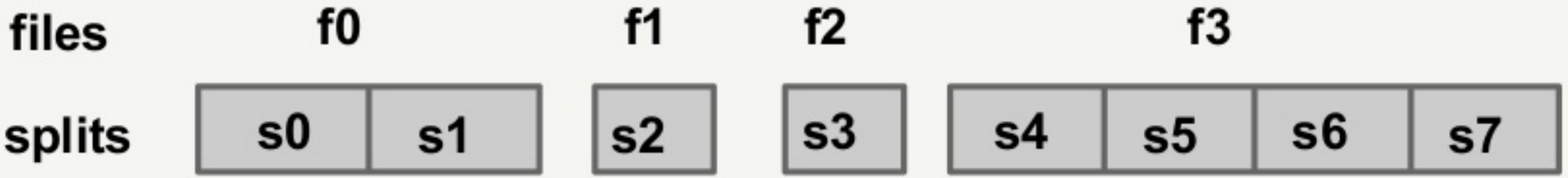


Source: kafka.apache.org

MapReduce data processing is driven by this concept of input splits



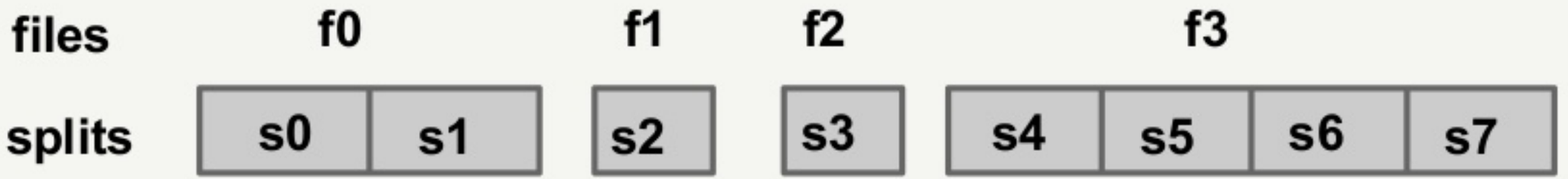
Job manager does split calculation

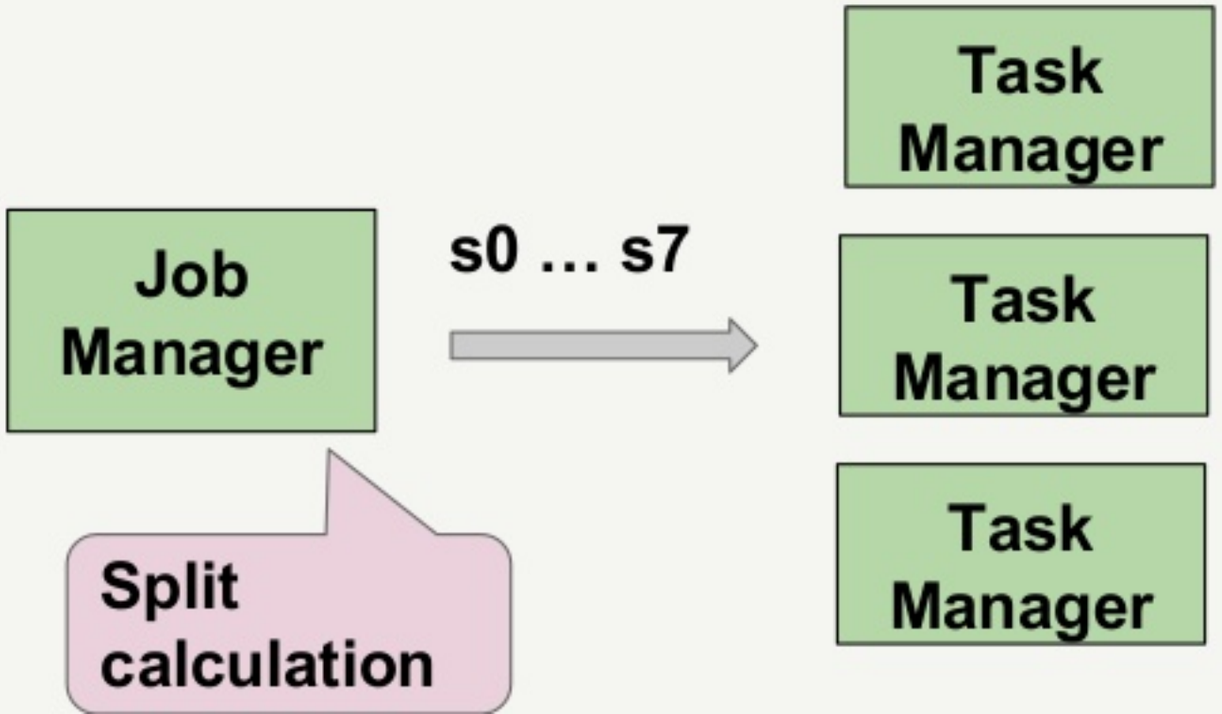


Job Manager

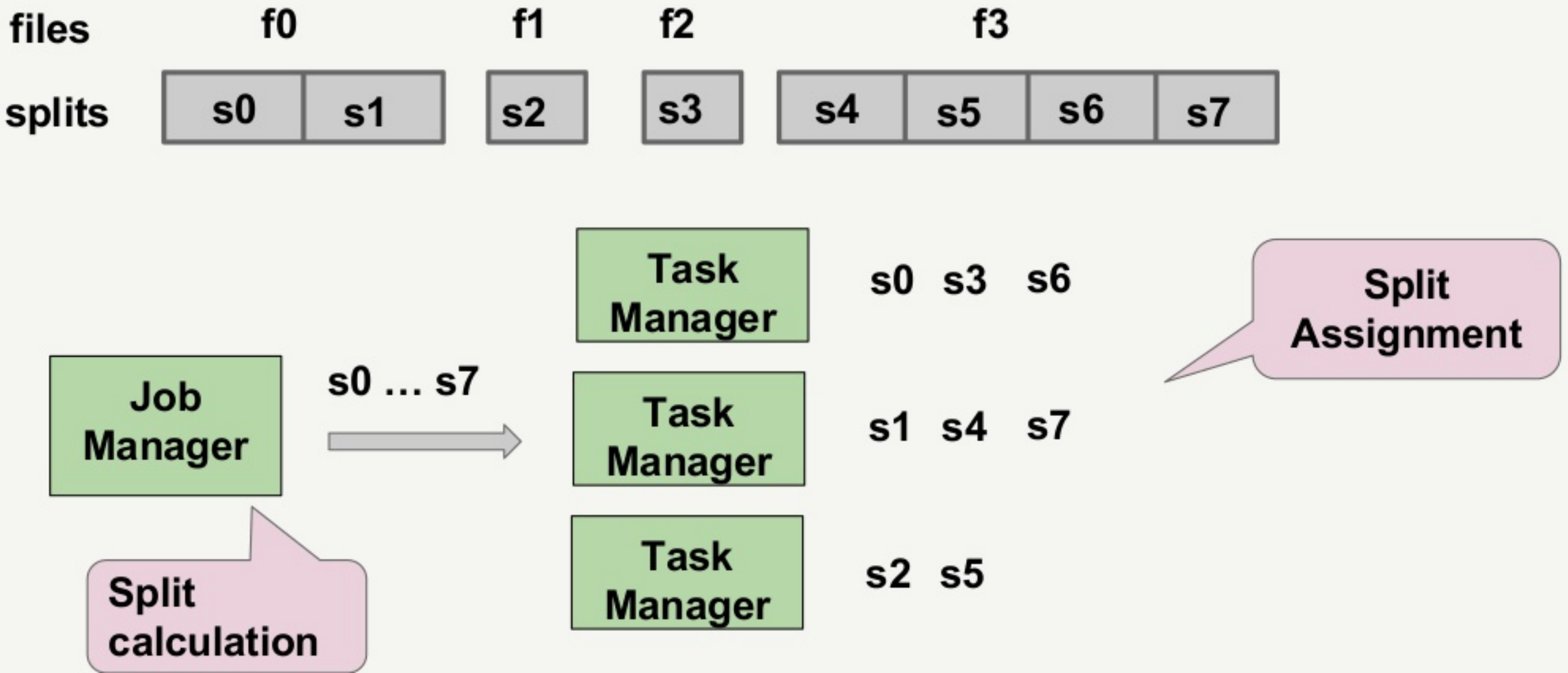
Split calculation

Job manager broadcasts input splits to all task managers

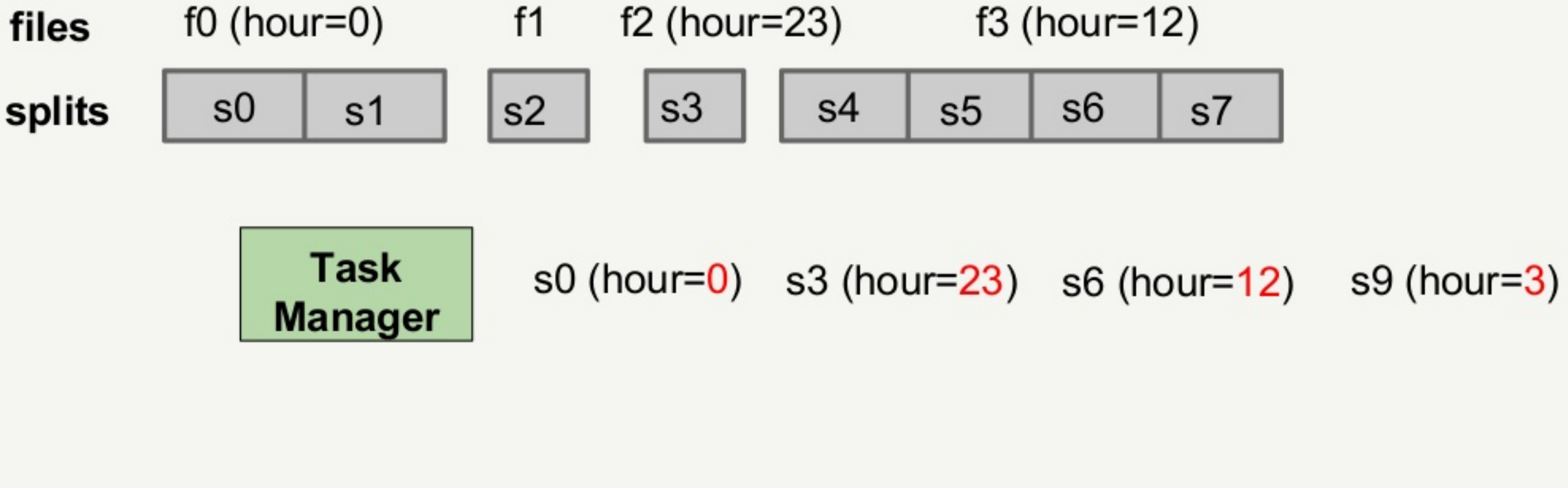




Task managers run the same split assignment algorithm



There is no guarantee of order for data files



Does ordering matter?

- Usually not for stateless jobs
- Probably important for stateful jobs

Late events can be dropped

- When watermark is past the end timestamp of the window
- Allowed lateness can give some extra time buffer

Source: flink.apache.org

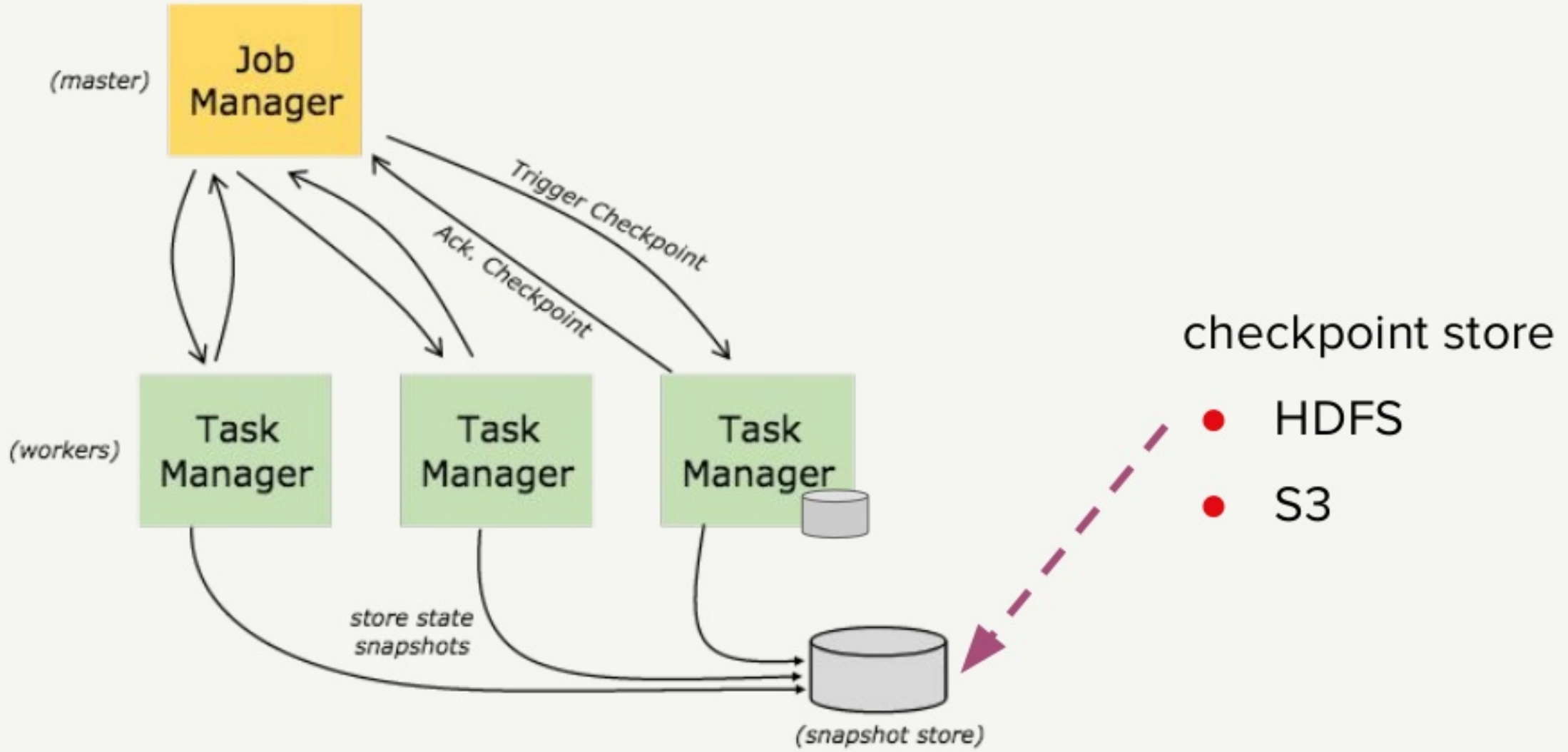
Agenda

Hive
Backfill

Flink
Rewind

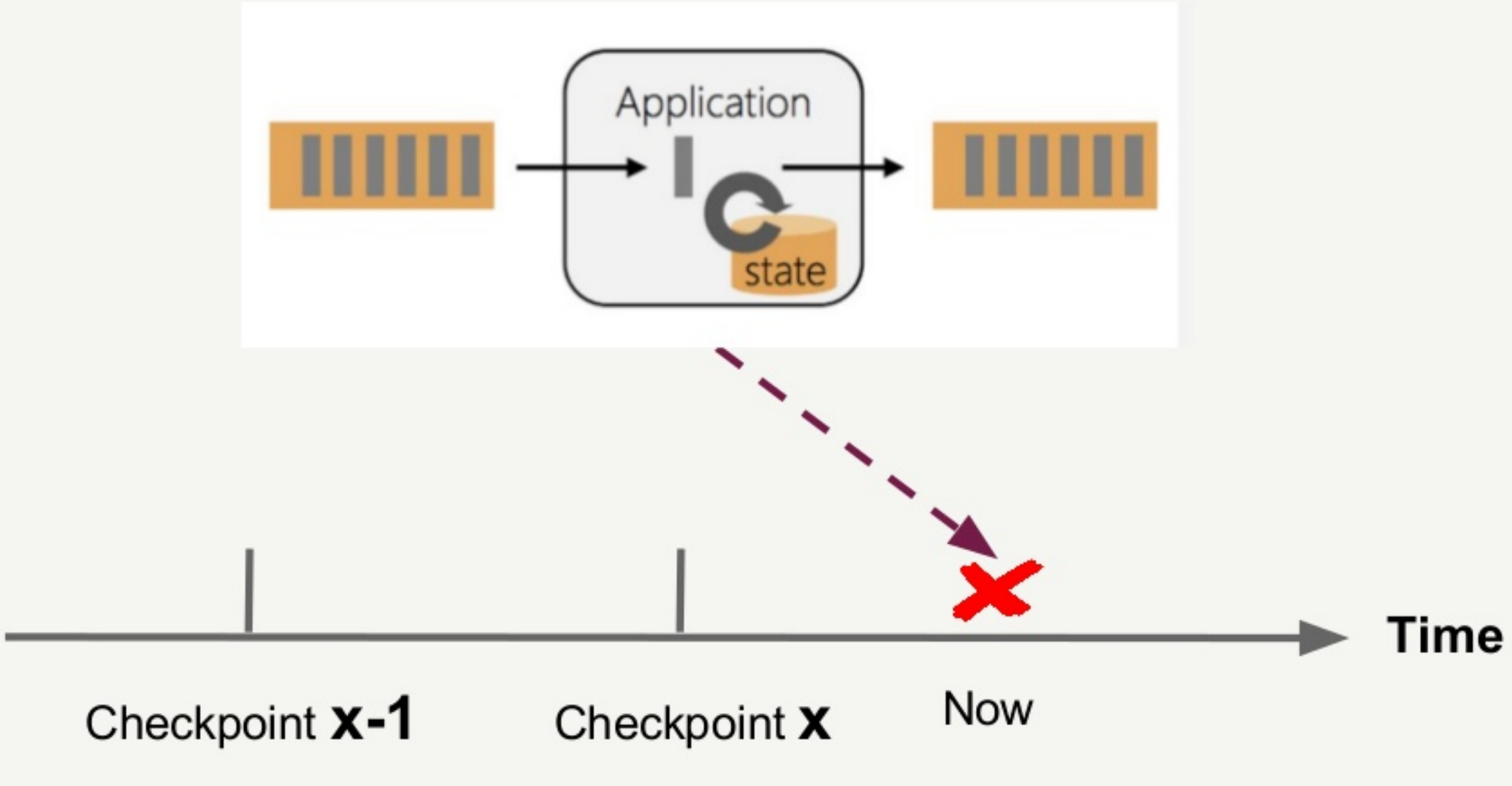
Caveats

Checkpoint snapshots and uploads state to DFS

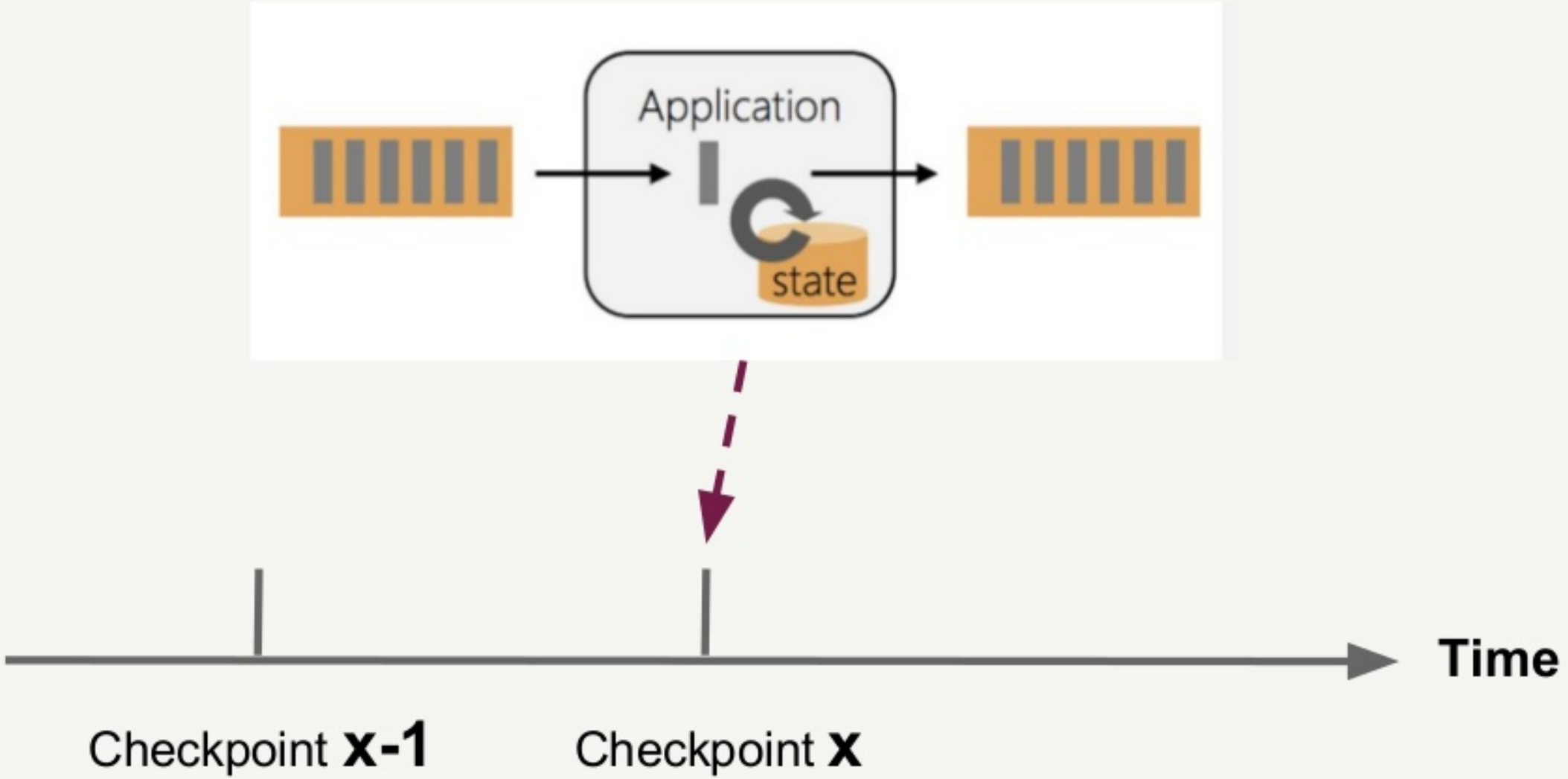


Source: http://flink.apache.org/

Checkpoint achieves fault tolerance



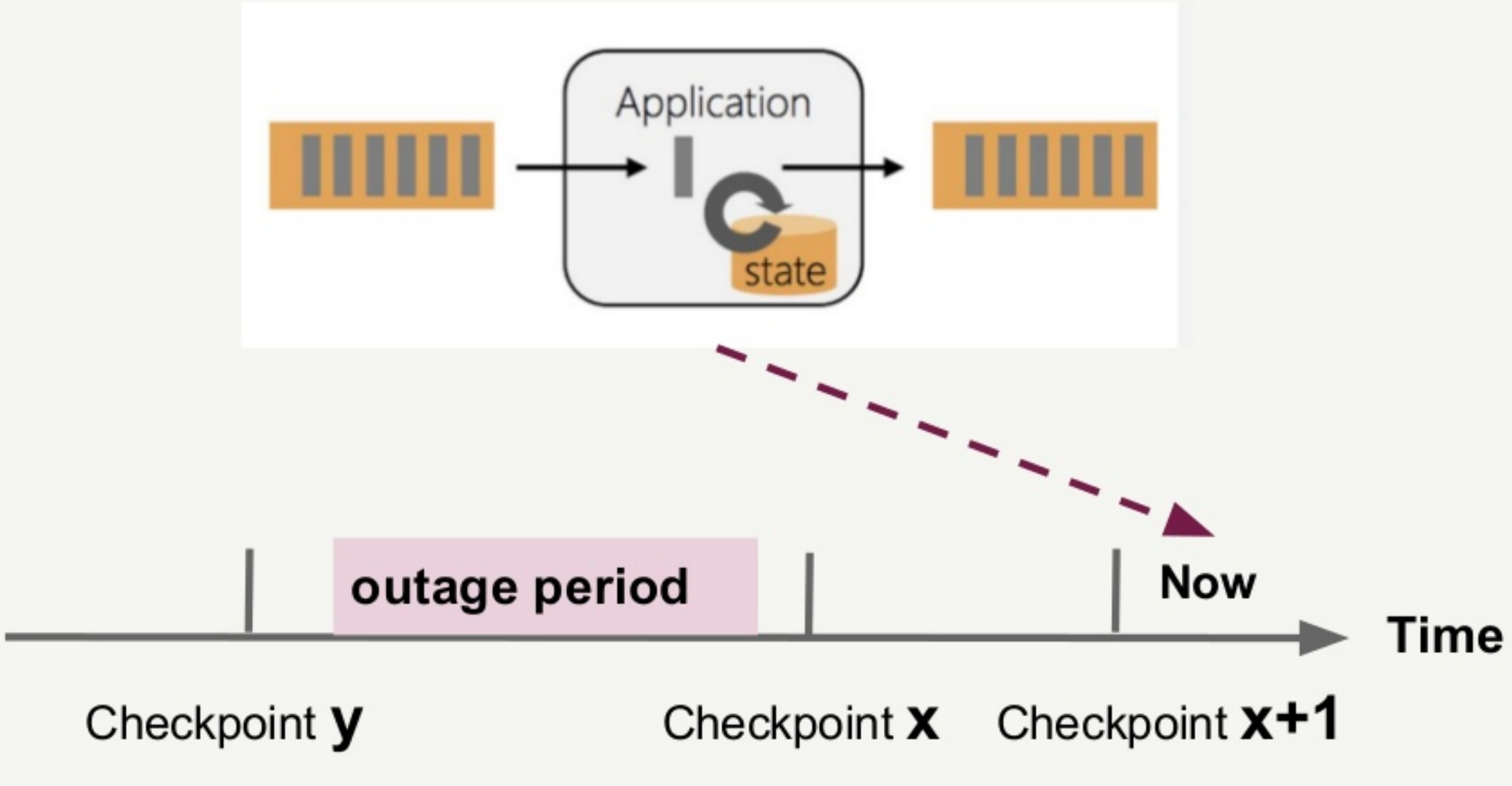
Checkpoint achieves fault tolerance



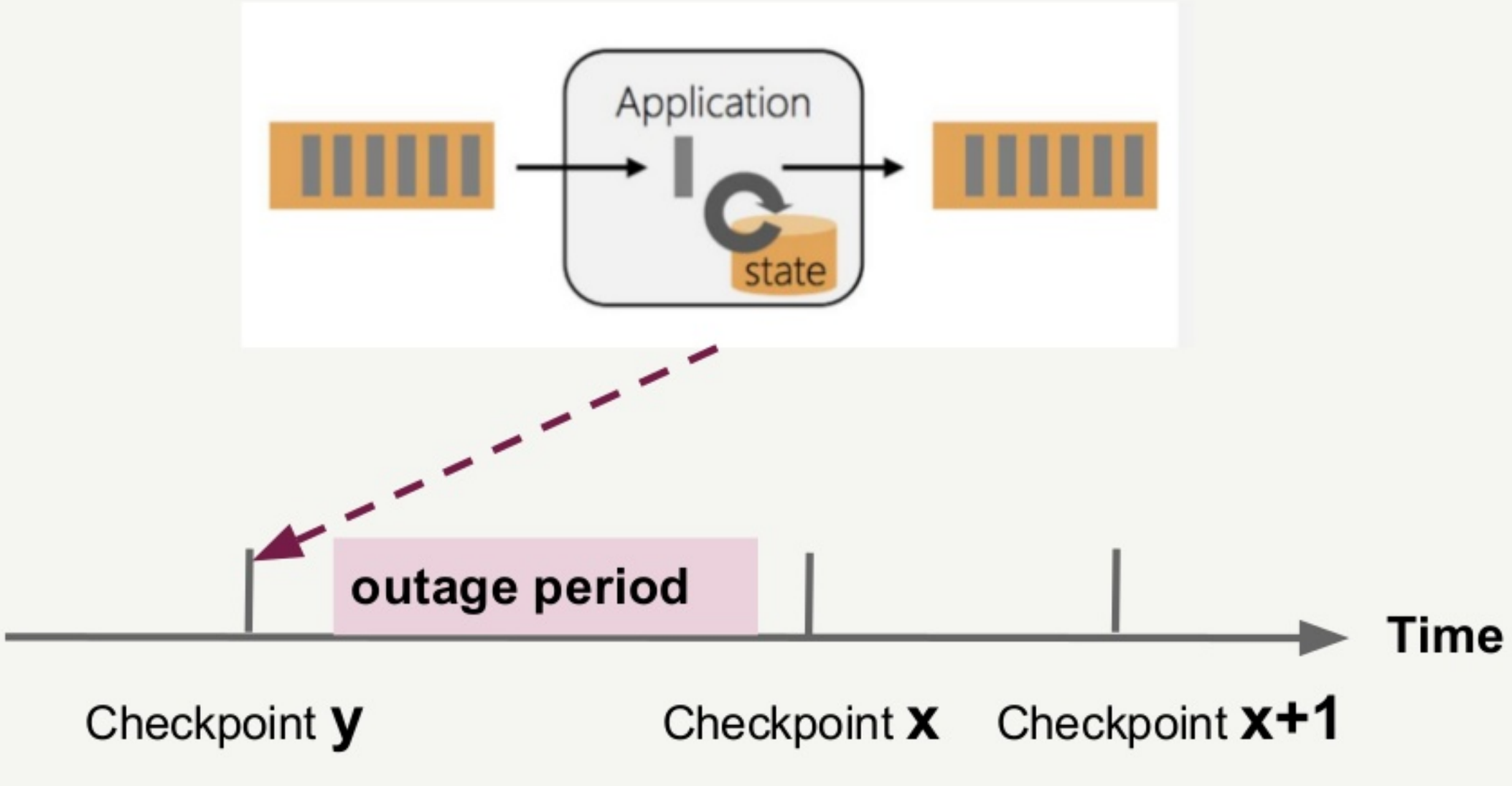
Enable external checkpoint

```
CheckpointConfig config = env.getCheckpointConfig();
config.enableExternalizedCheckpoints(
    ExternalizedCheckpointCleanup.RETAIN_ON_CANCELLATION);
```

Rewind job to a checkpoint before outage period



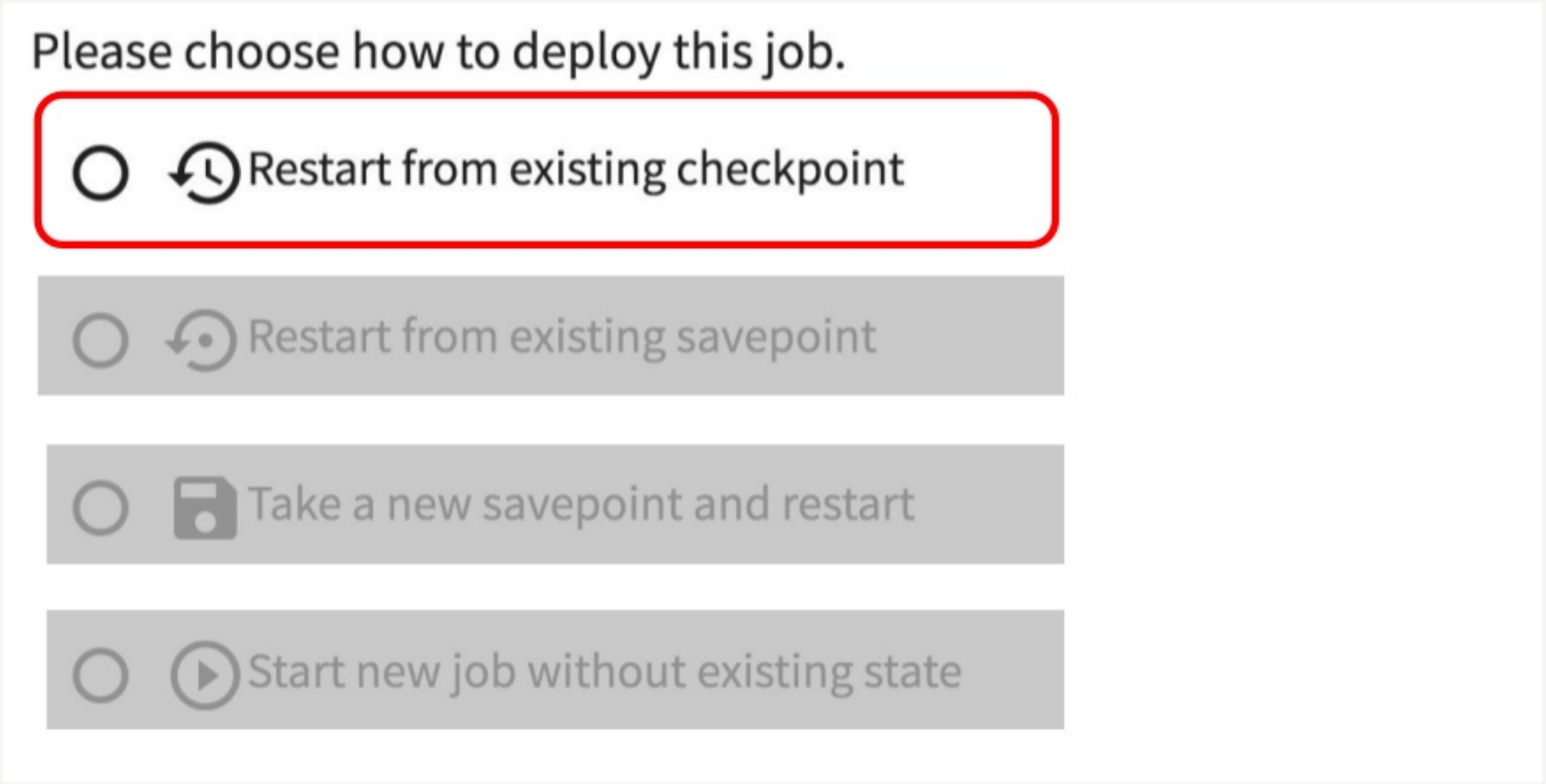
Rewind job to a checkpoint before outage period



There are no warmup and ordering issues with Flink rewind

- Application state is correct after rewind
- Flink job is still reading the same Kafka source

Choose the external checkpoint option



Choose a checkpoint

Choose an existing checkpoint to deploy from:

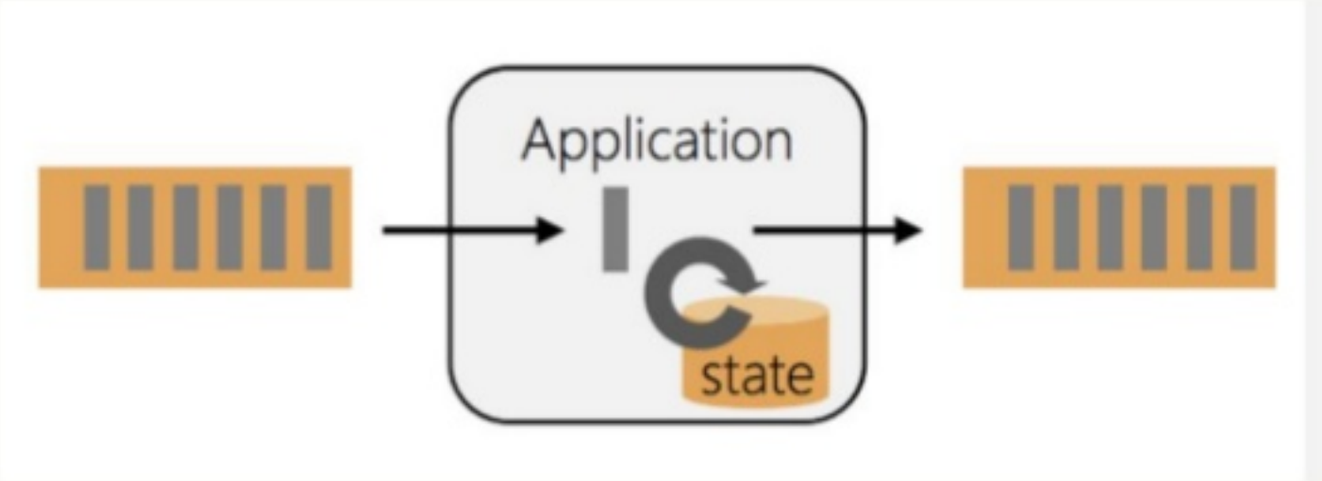
Time	Path
05/10/2018 11:33:31 AM	s3://us-east-1.spaas.test/spaas_perf-state_dev/extCheckpoints/checkpoint_metadata-b07e66947b75
05/10/2018 11:28:32 AM	s3://us-east-1.spaas.test/spaas_perf-state_dev/extCheckpoints/checkpoint_metadata-73384146ec0c
05/10/2018 11:23:32 AM	s3://us-east-1.spaas.test/spaas_perf-state_dev/extCheckpoints/checkpoint_metadata-70e04092ff14
05/10/2018 11:03:33 AM	s3://us-east-1.spaas.test/spaas_perf-state_dev/extCheckpoints/checkpoint_metadata-39f9975040f0

Resume from checkpoint

s3://us-east-1.spaas.test/spaas_perf-state_dev/extCheckpoints/checkpoint_metadata-b07e66947b75

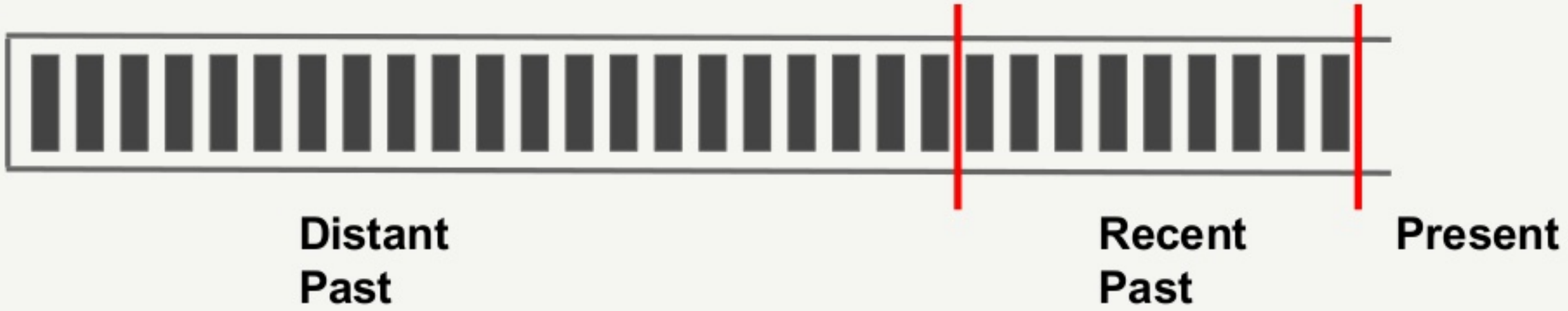
Cancel Deploy

Kafka retention matters



Can we have 10 days of Kafka retention?

Anatomy of data stream



Adapted from “An elastic batch-and stream-processing stack with Pravega and Apache Flink” by Stephan Ewen and Flavio Junqueira

Anatomy of data stream

Distant Past

Recent Past

Present

Can’t keep 10 days of data in local disk

- **d2.8xl** : 10 Gbps of network, 48 TB of disk
- Assuming 2 Gbps ingestion rate per instance, 10 days of data requires 216 TB of disk

EBS is more expensive than S3

EBS is more expensive than S3

	Cost (per month)
EBS: throughput optimized HDD	$0.045 per GB

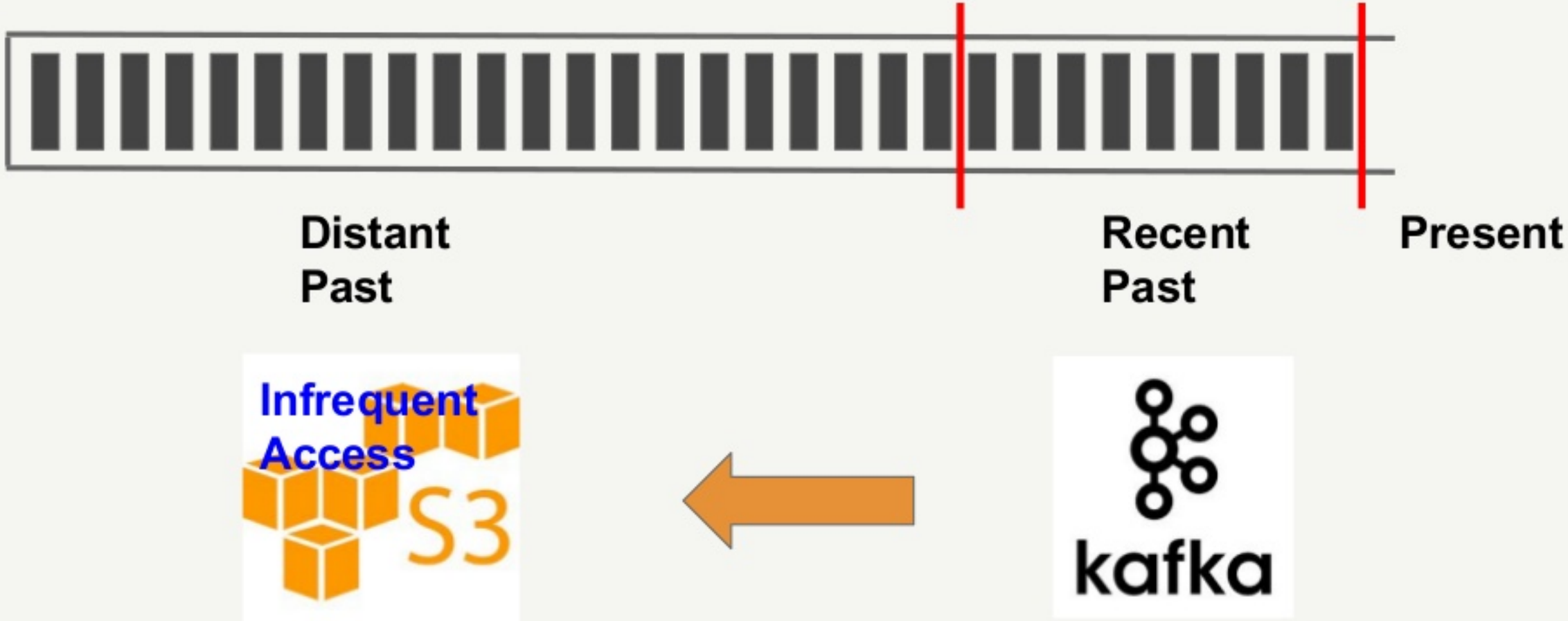
EBS is more expensive than S3

	Cost (per month)
EBS: throughput optimized HDD	$0.045 per GB
S3 standard	$0.021 per GB

EBS is more expensive than S3

	Cost (per month)
EBS: throughput optimized HDD	$0.045 per GB
S3 standard	$0.021 per GB
S3 Standard-Infrequent Access	$0.013 per GB

What if Kafka offloads historical data to S3 infrequent access tier



Here are the benefits of tiered storage

- Only deal with Kafka source
- Support 10-day retention cost efficiently

There are systems implemented tiered storage

Hive Backfill

v.s.

Flink Rewind

	Hive backfill	**Flink rewind**
Warm-up issue	Yes	No
Ordering issue	Yes	No

	Hive backfill	**Flink rewind**
Warm-up issue	Yes	No
Ordering issue	Yes	No
Applicability	Stateless	Stateless and stateful

	Hive backfill	**Flink rewind**
Warm-up issue	Yes	No
Ordering issue	Yes	No
Applicability	Stateless	Stateless and stateful
Data retention	Weeks or months	Hours or days

Pros for Hive backfill

- Long-term storage
- No delay for processing latest events
- Can achieve fast recovery

Here is our recommendation to users

Stateless	Hive Backfill
Stateful	Flink Rewind

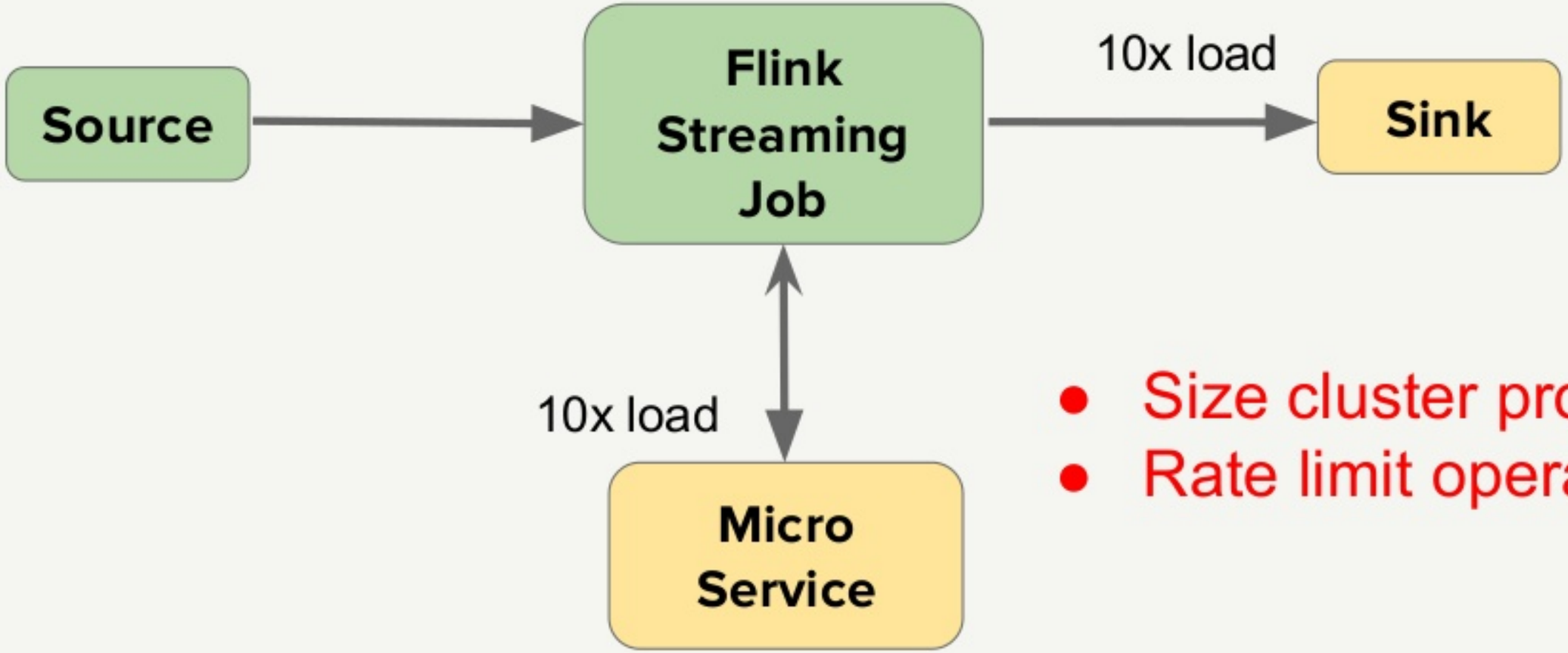
Agenda

Hive Backfill

Flink Rewind

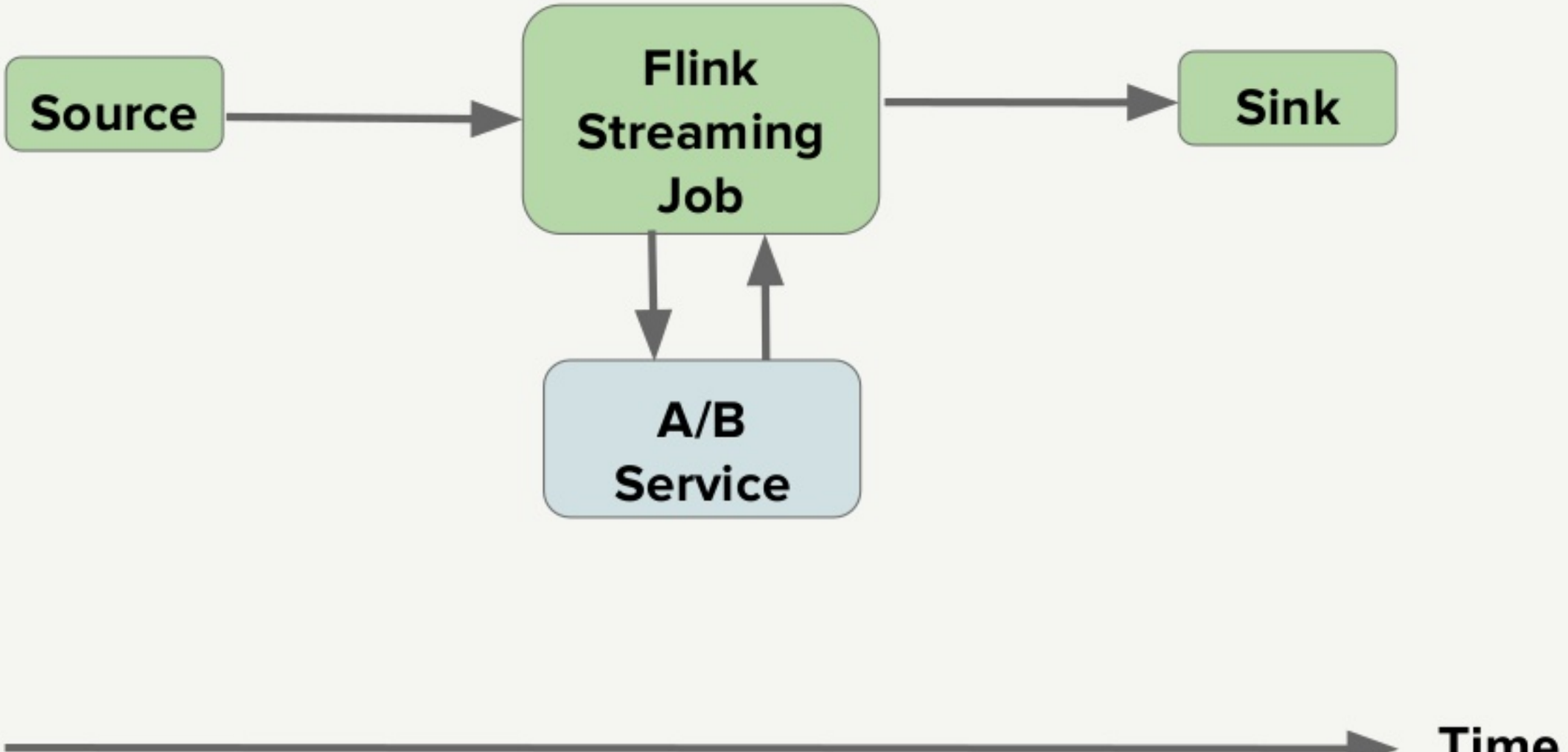
Caveats

Caveat 1: Don’t overwhelm external services

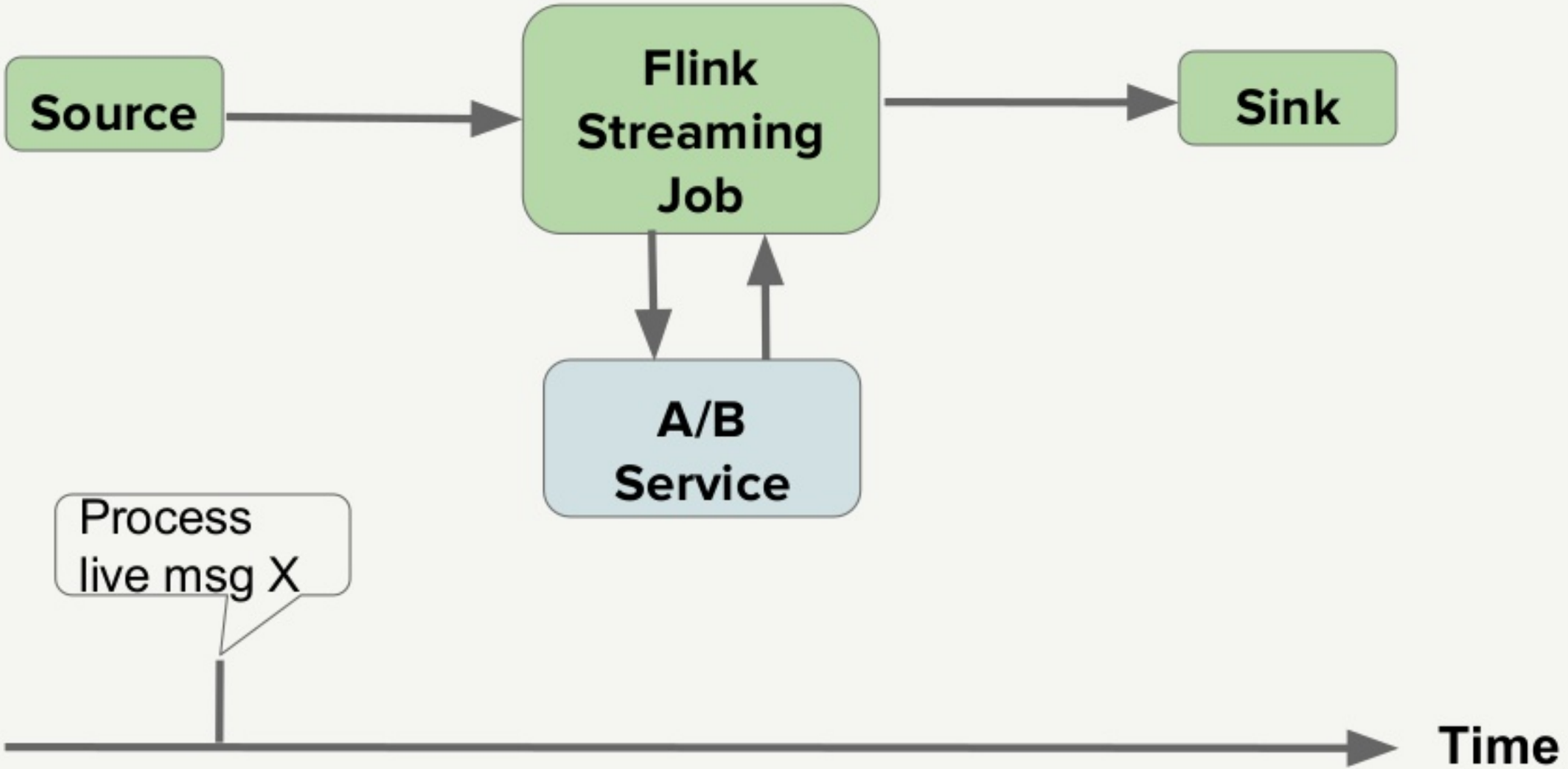


- Size cluster properly
- Rate limit operator

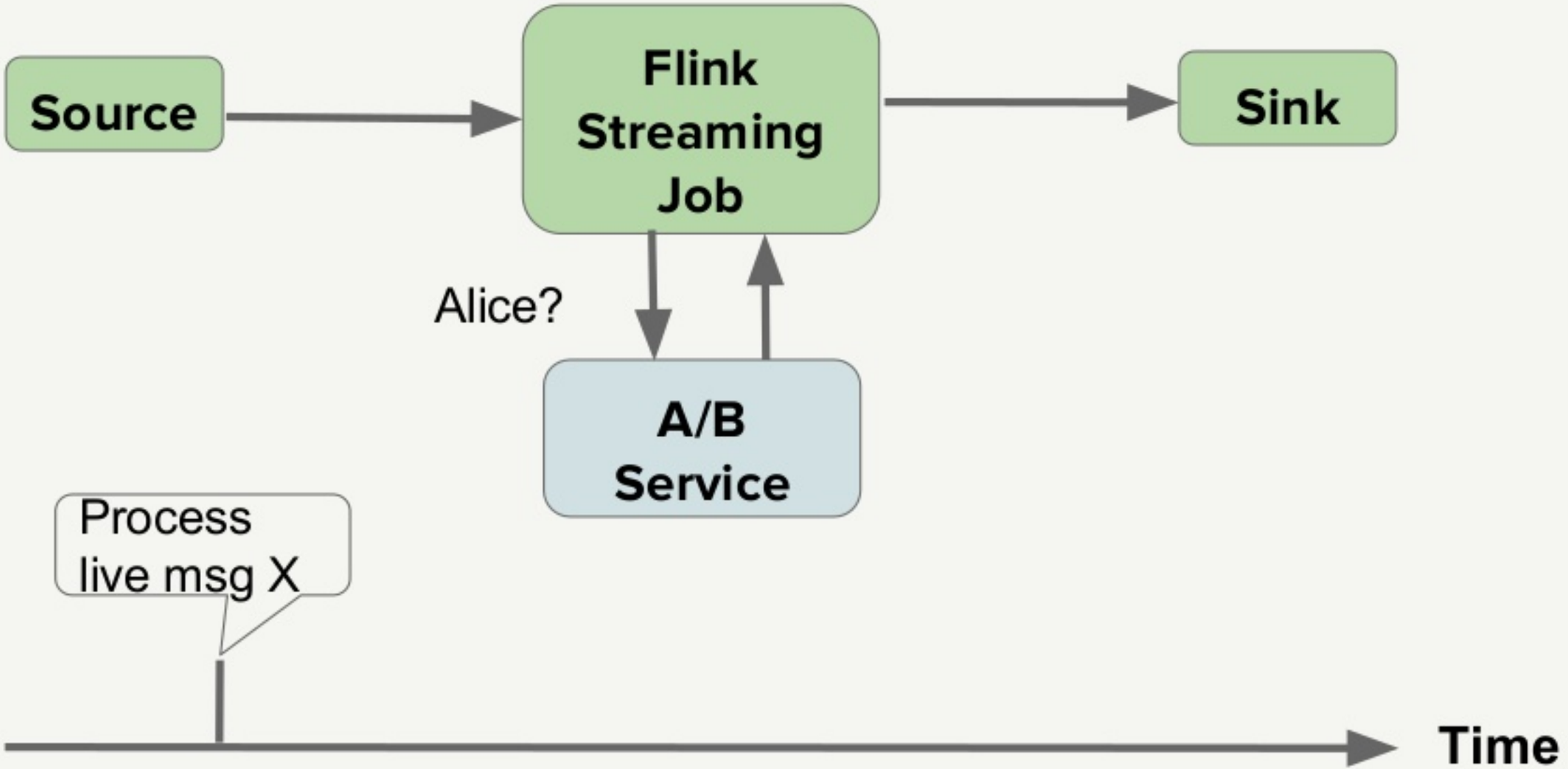
Caveat 2: Your dependency may not participate in rewind



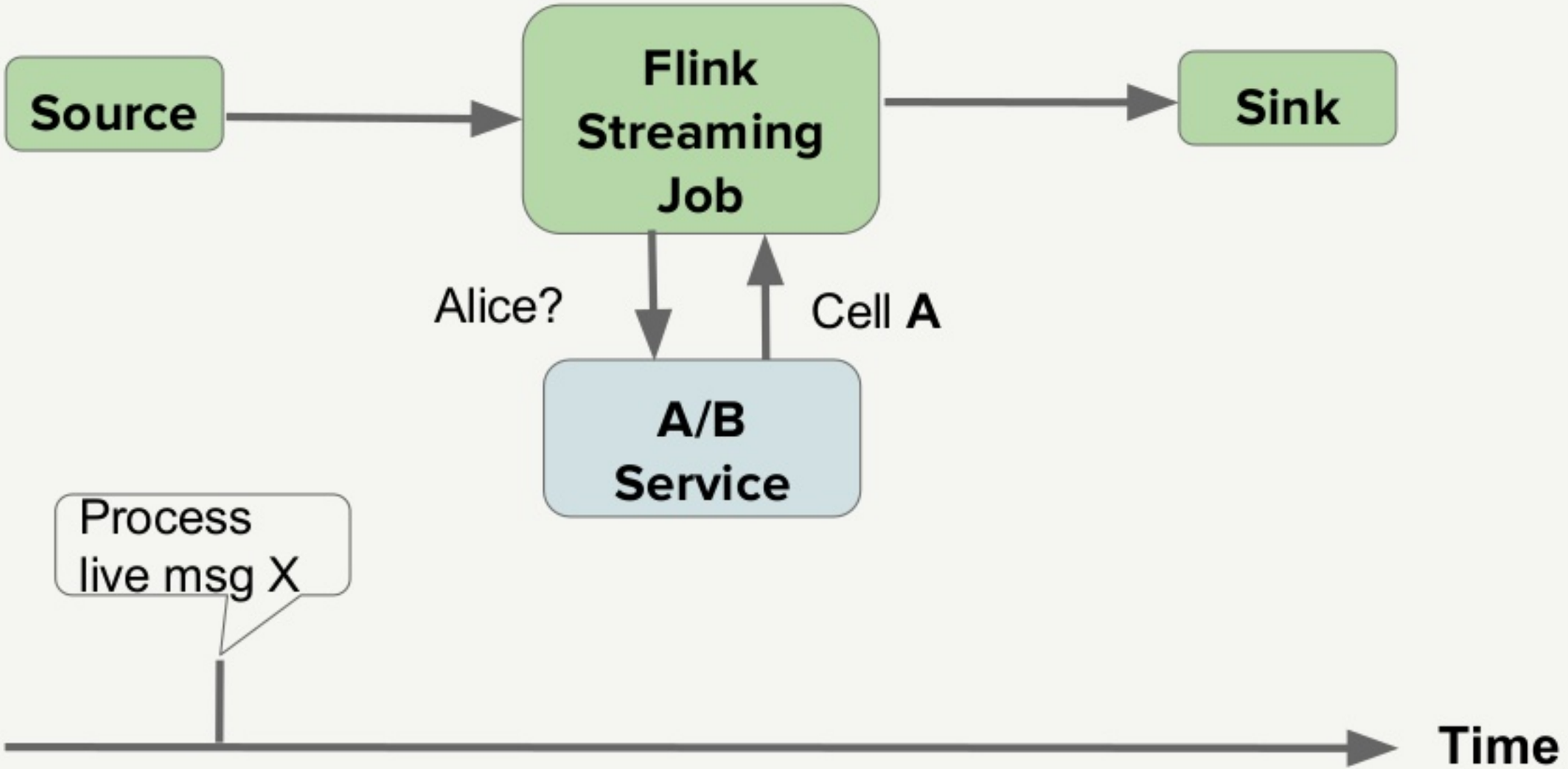
Caveat 2: Your dependency may not participate in rewind



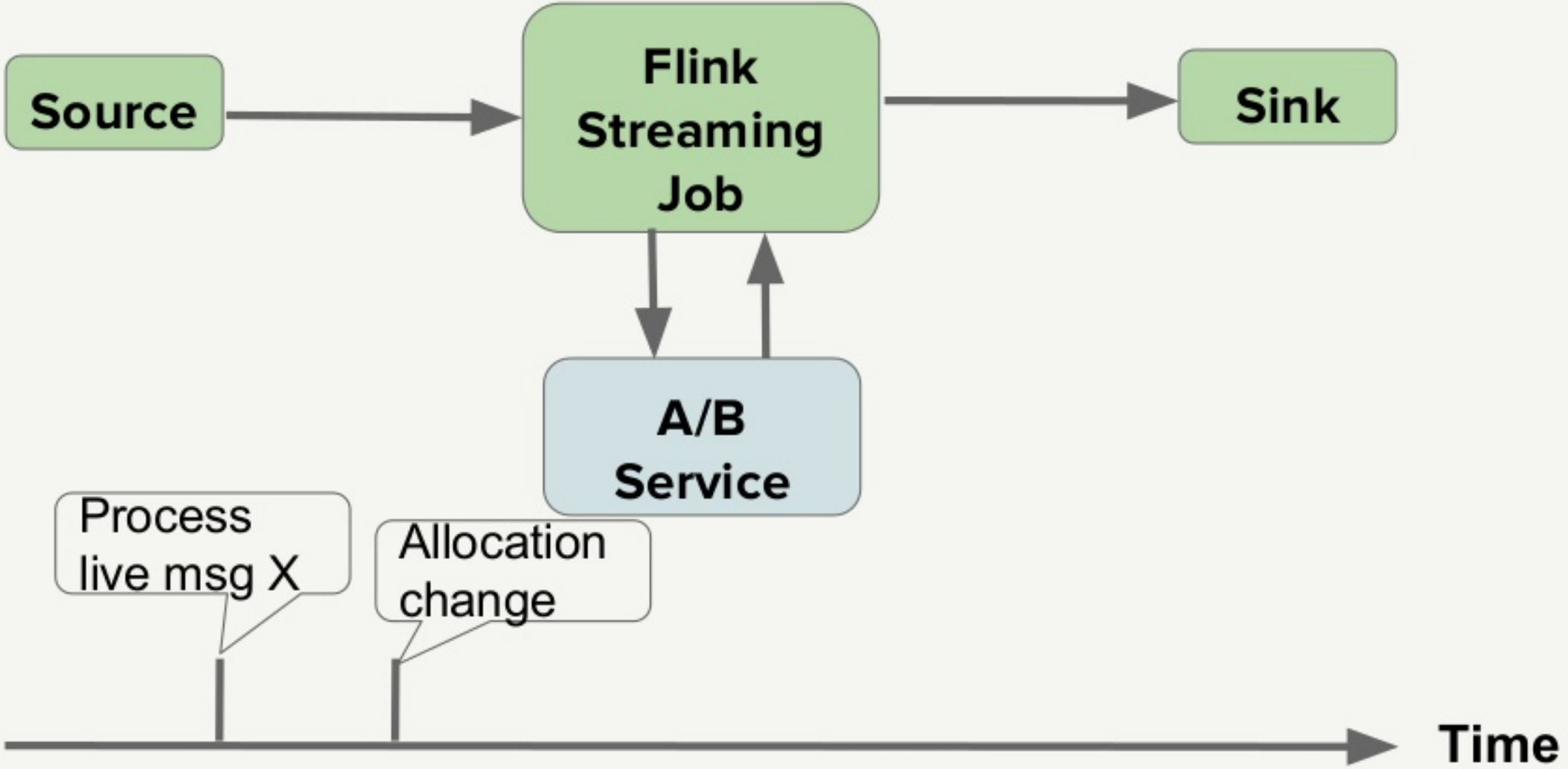
Caveat 2: Your dependency may not participate in rewind



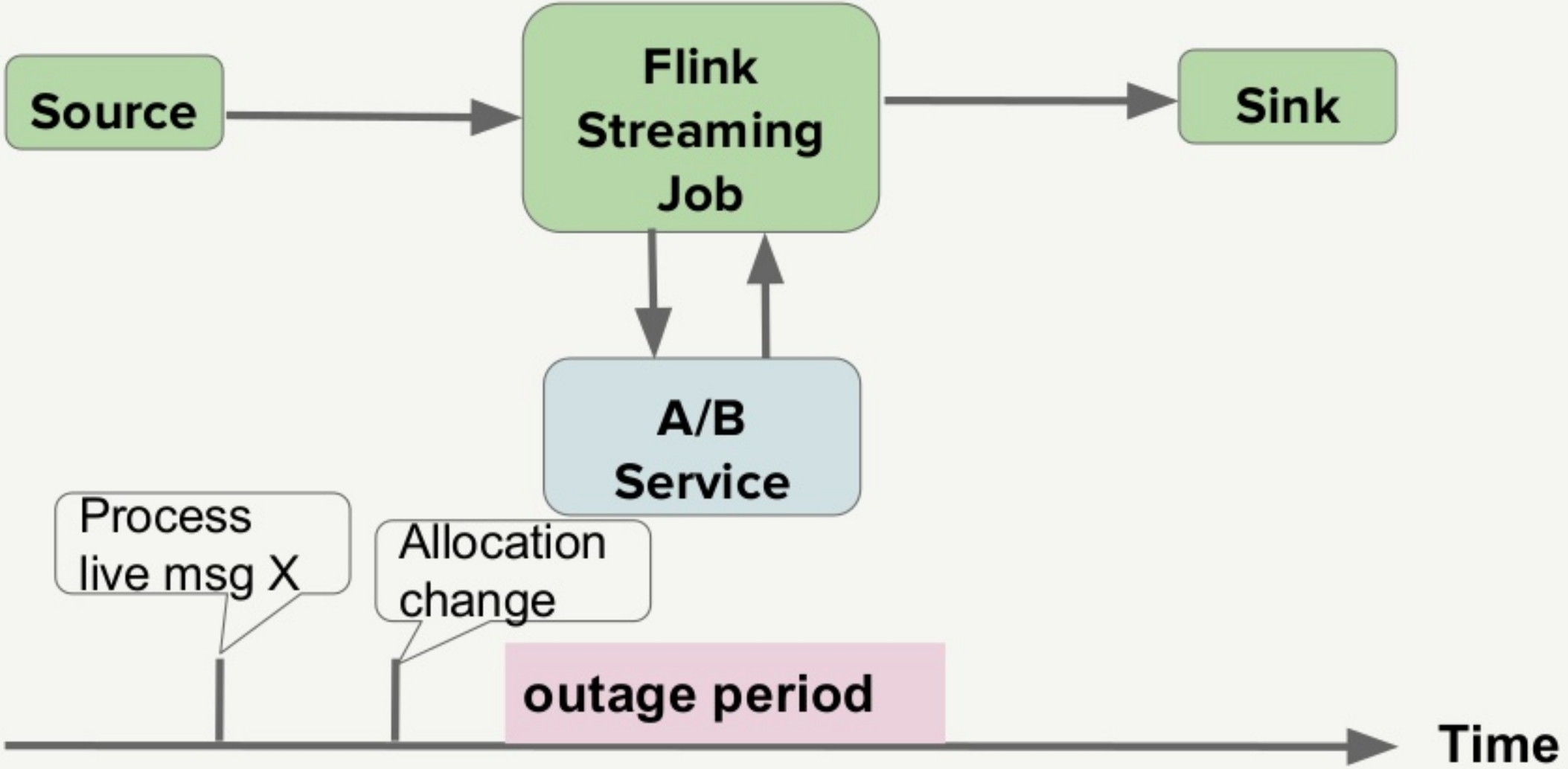
Caveat 2: Your dependency may not participate in rewind



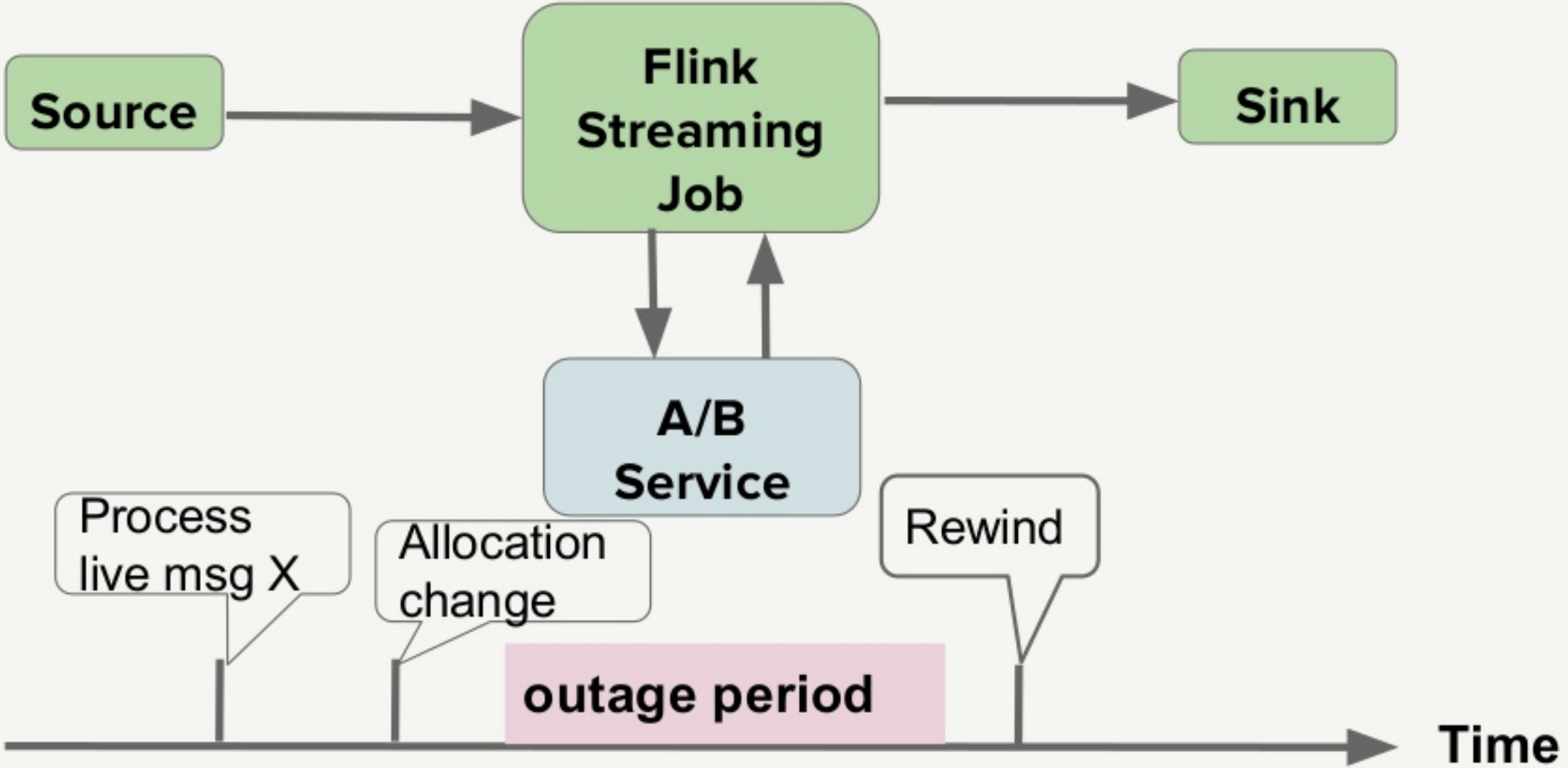
Caveat 2: Your dependency may not participate in rewind



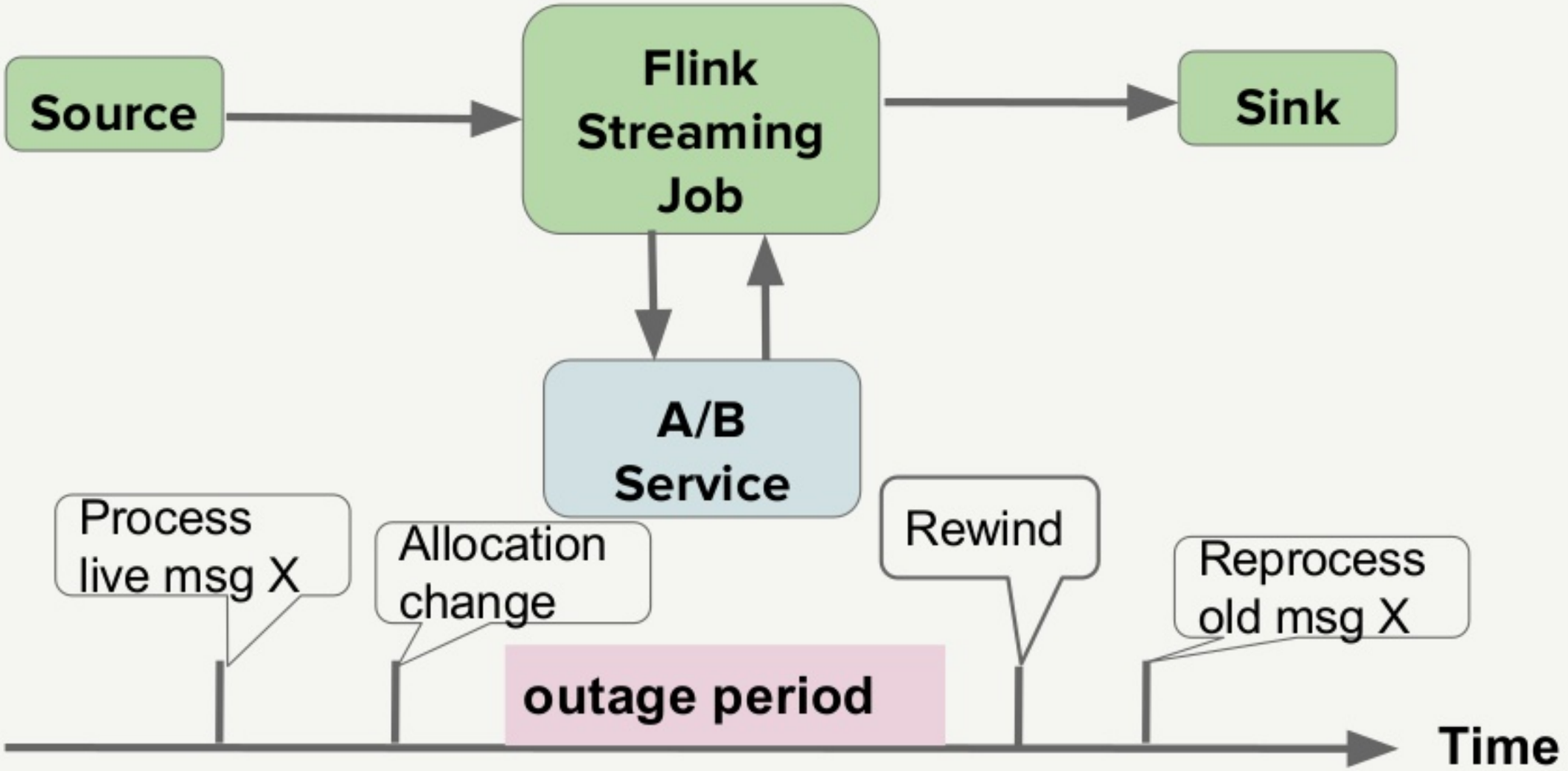
Caveat 2: Your dependency may not participate in rewind



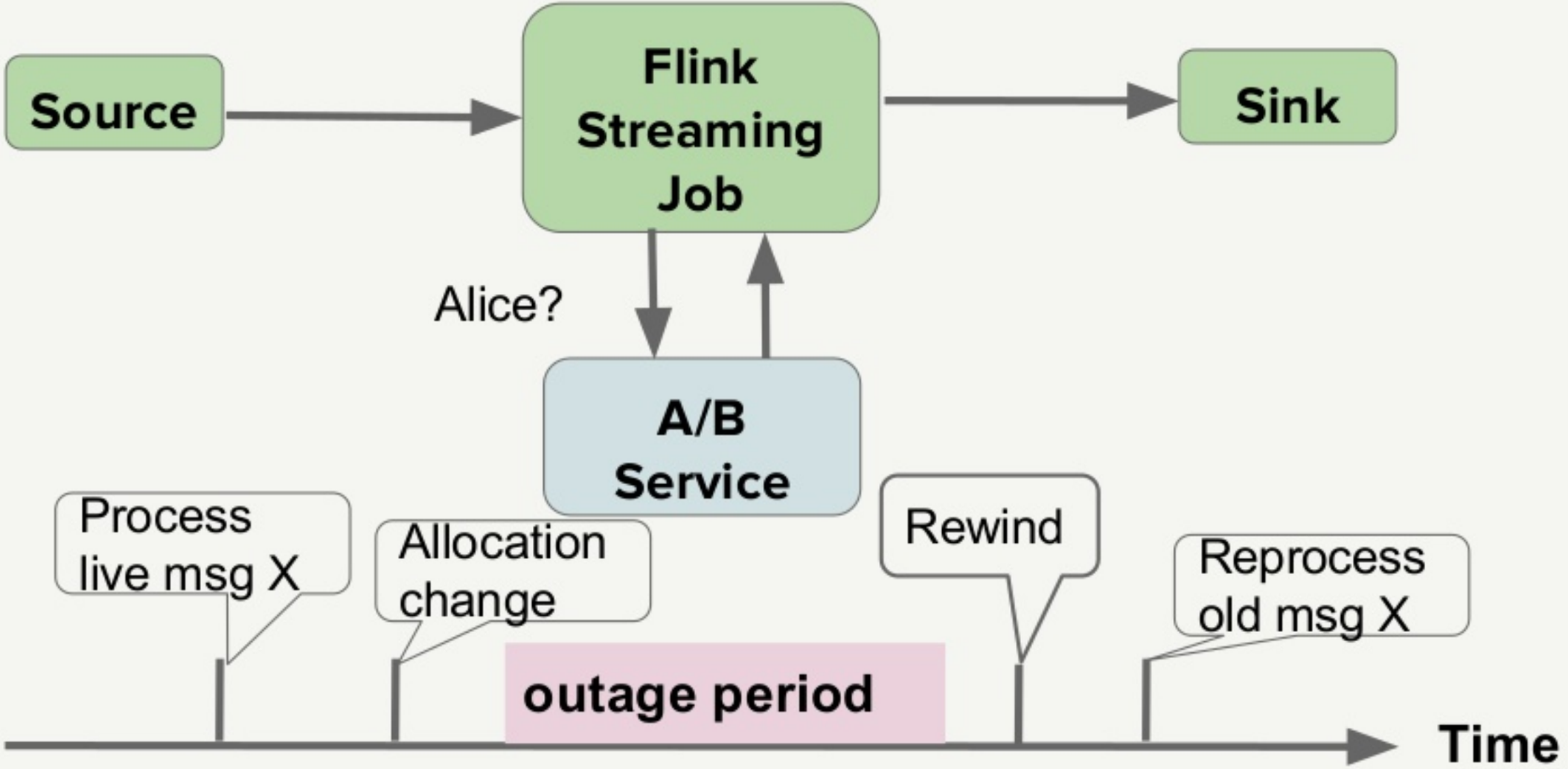
Caveat 2: Your dependency may not participate in rewind



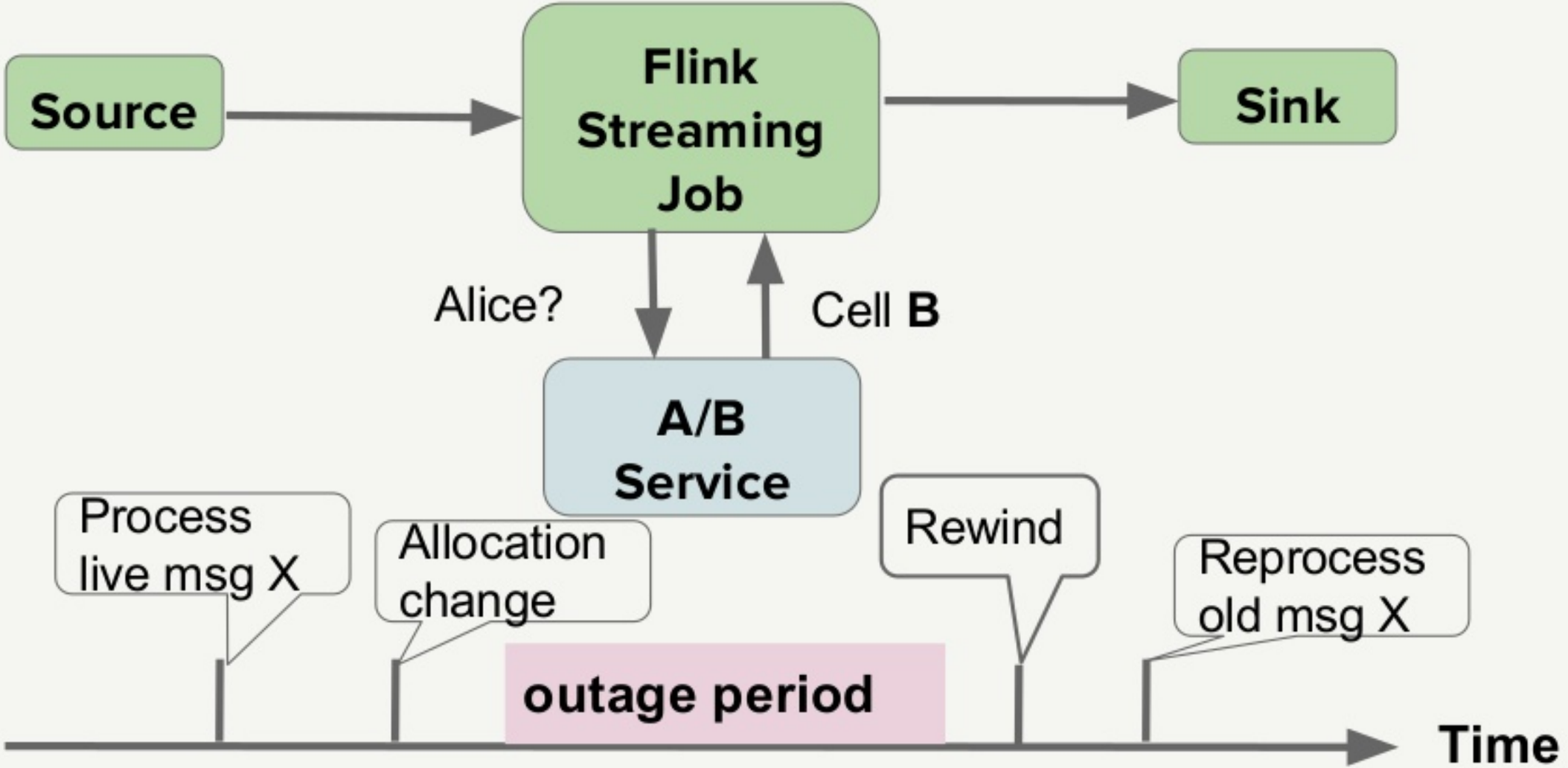
Caveat 2: Your dependency may not participate in rewind



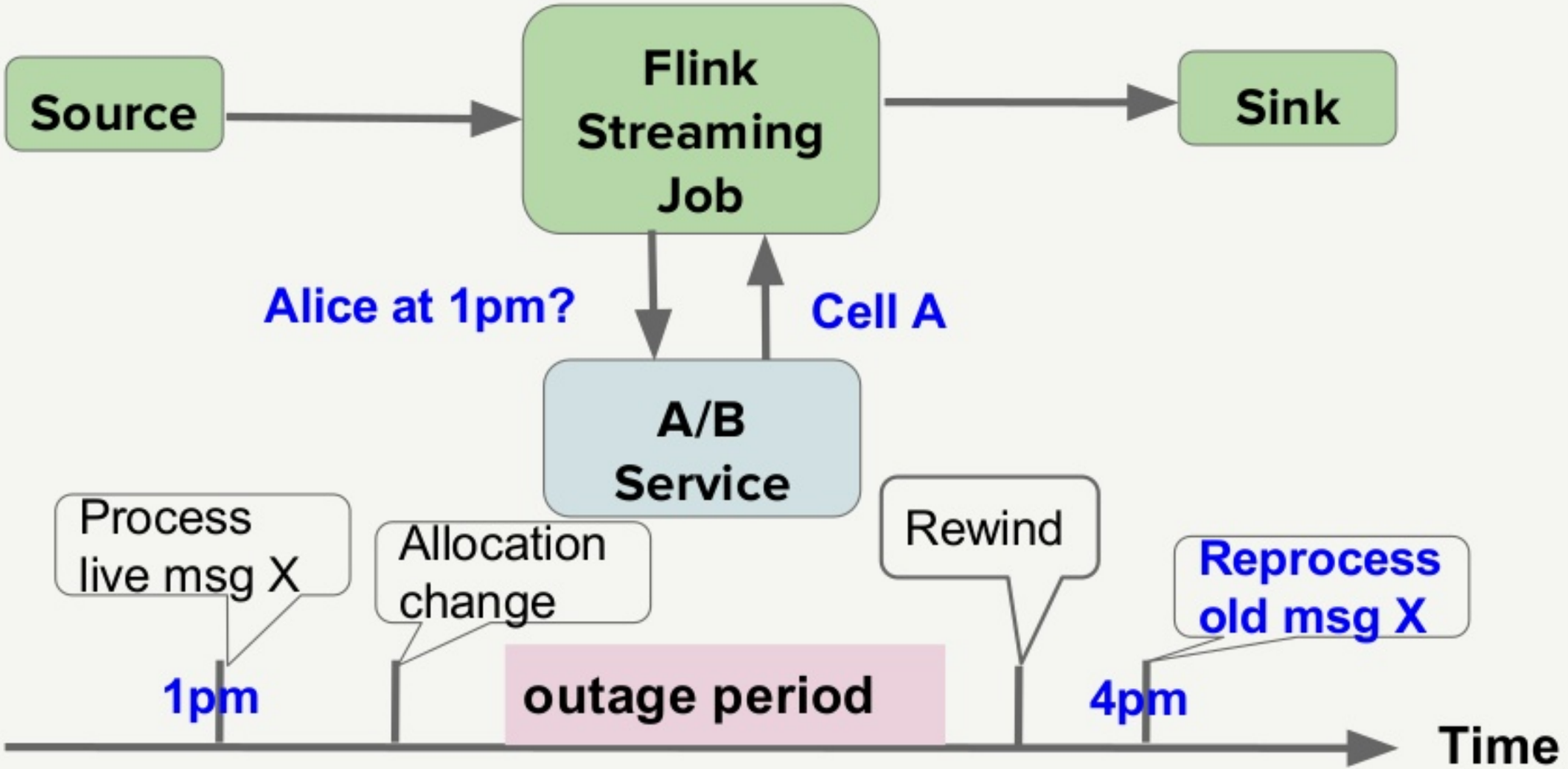
Caveat 2: Your dependency may not participate in rewind



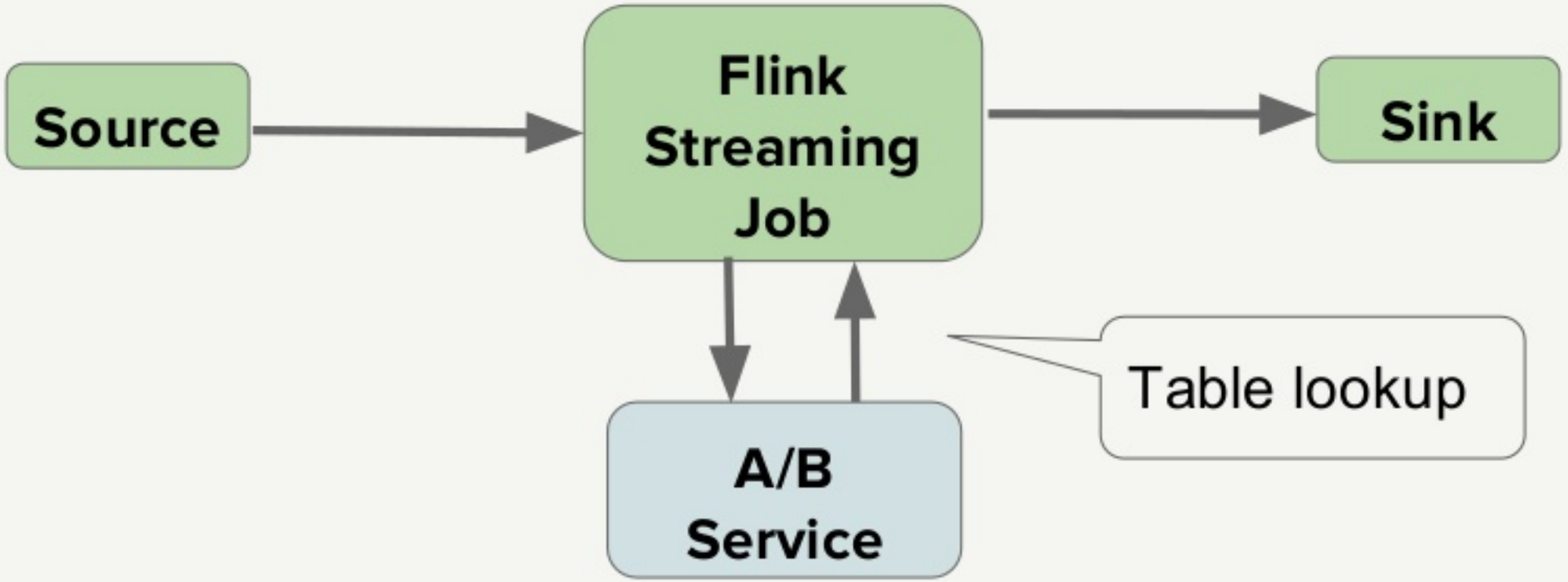
Caveat 2: Your dependency may not participate in rewind



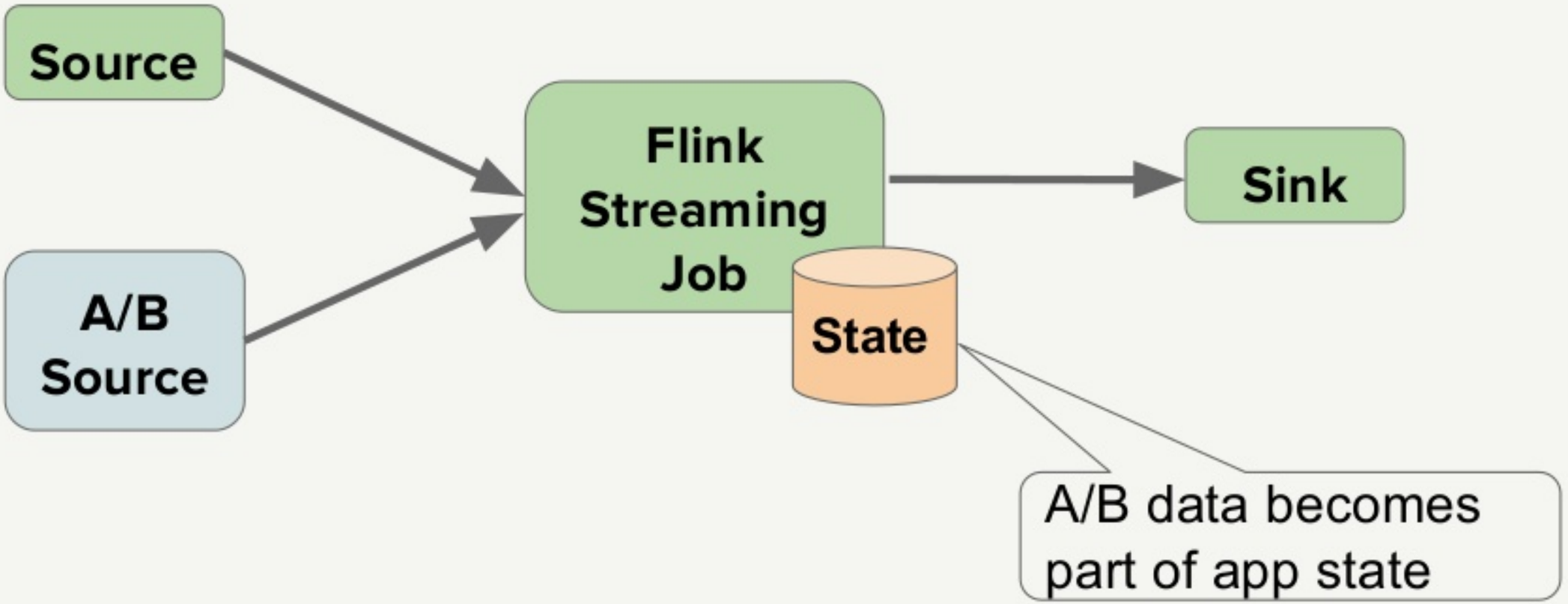
Solution 1: Support lookup with historical view



Solution 2: Convert table lookup to a streaming source



Solution 2: Convert table lookup to a streaming source



Caveat #3: watch out for the impact to downstream consumers

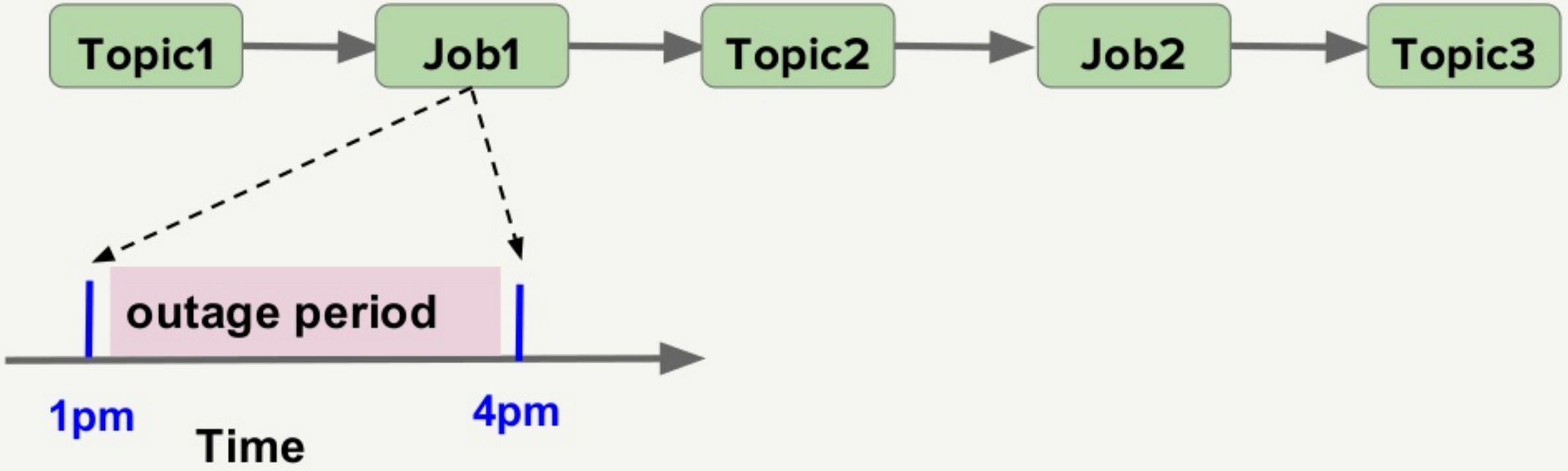
- Idempotent sink
  - ElasticSearch, Cassandra

Caveat #3: watch out for the impact to downstream consumers

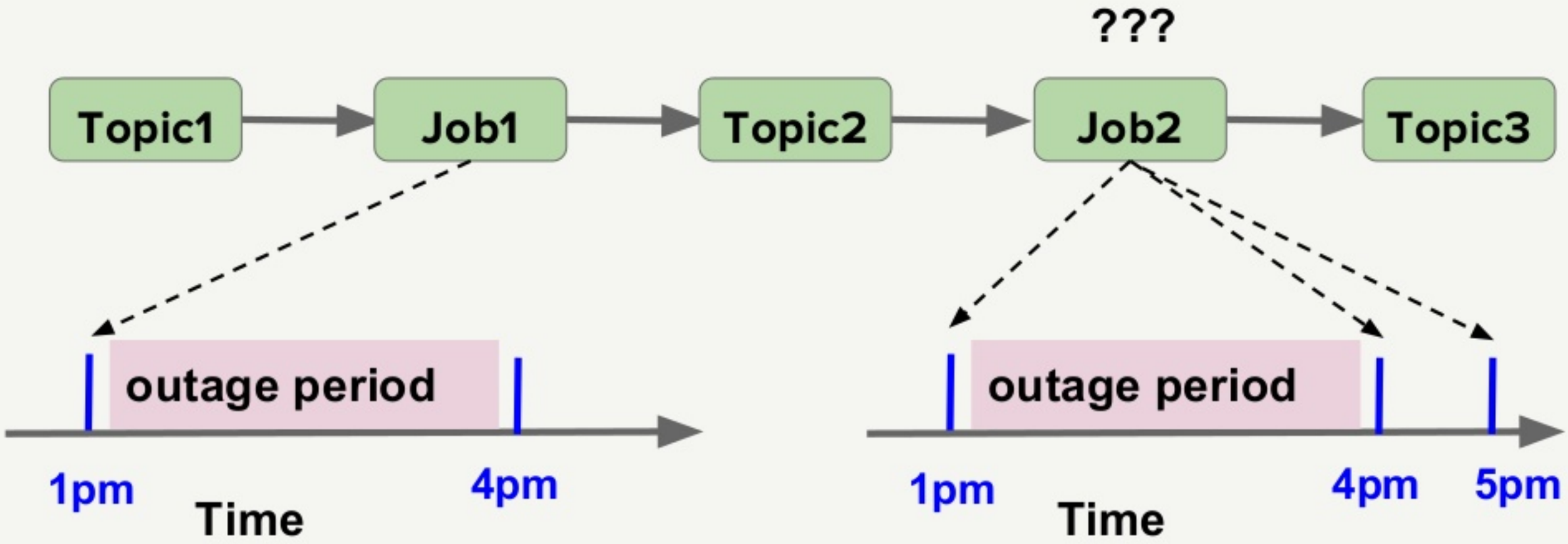
- Idempotent sink
  - ElasticSearch, Cassandra
- Resettable sink
  - Drop Hive partition with bad data

Here is a more complicated case with Kafka sink

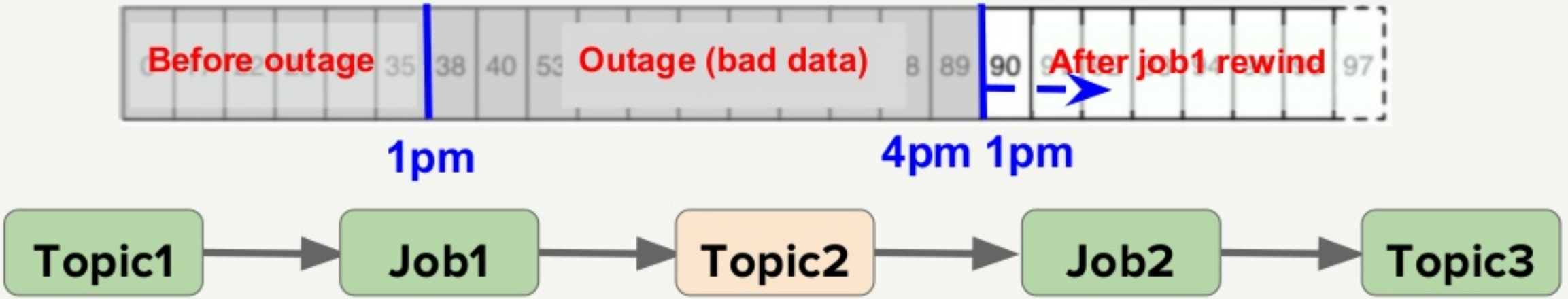
At 4pm, job1 rewinded to checkpoint taken at 1pm



What should job2 do?

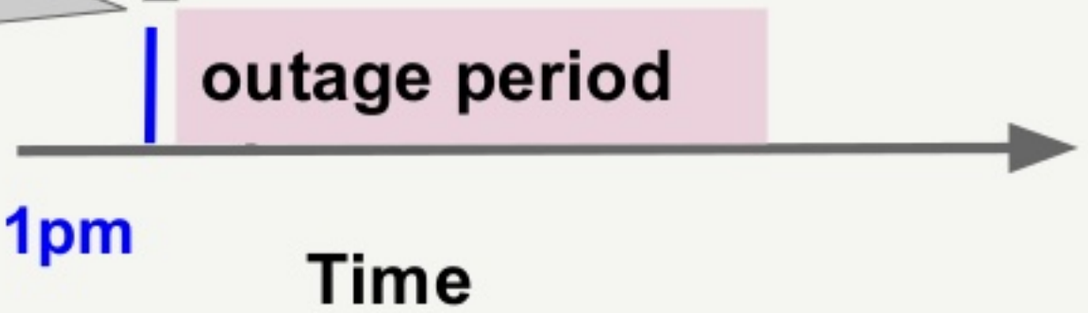


Anatomy of topic2

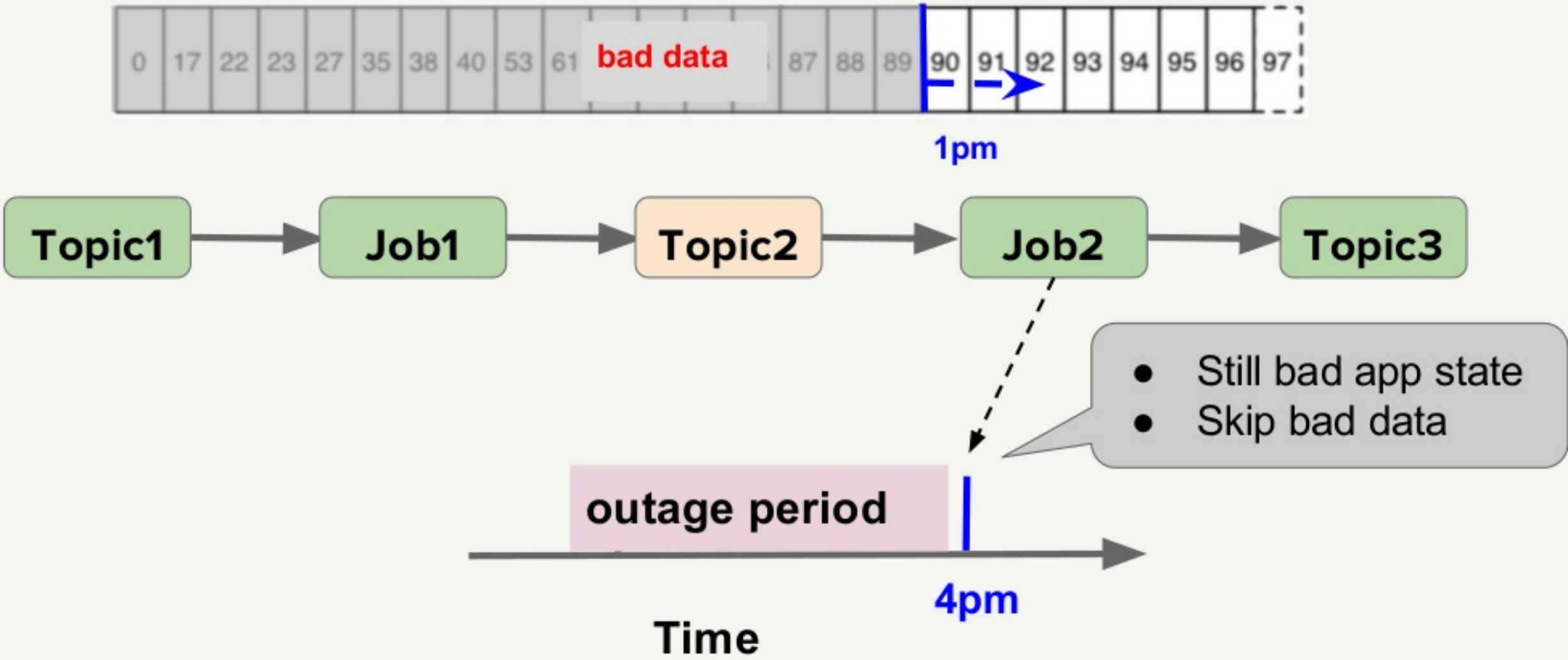


Rewind job2 to 1pm

- Correct app state
- Still reprocess bad data



Rewind job2 to 4pm

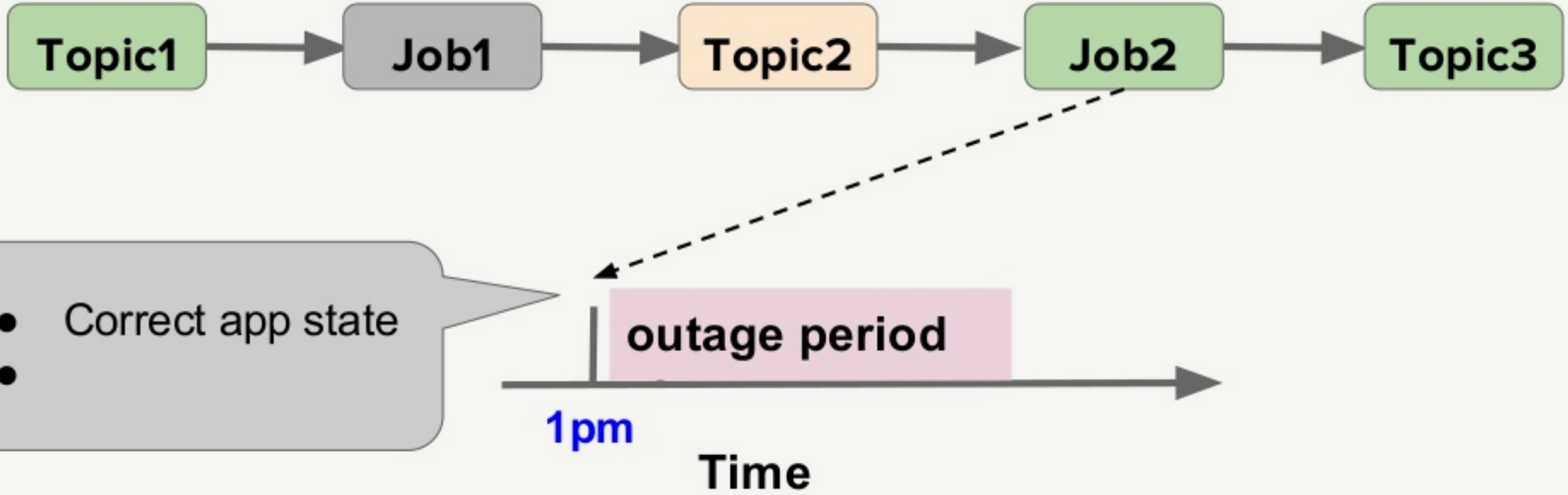


Stop job1 and job2 first

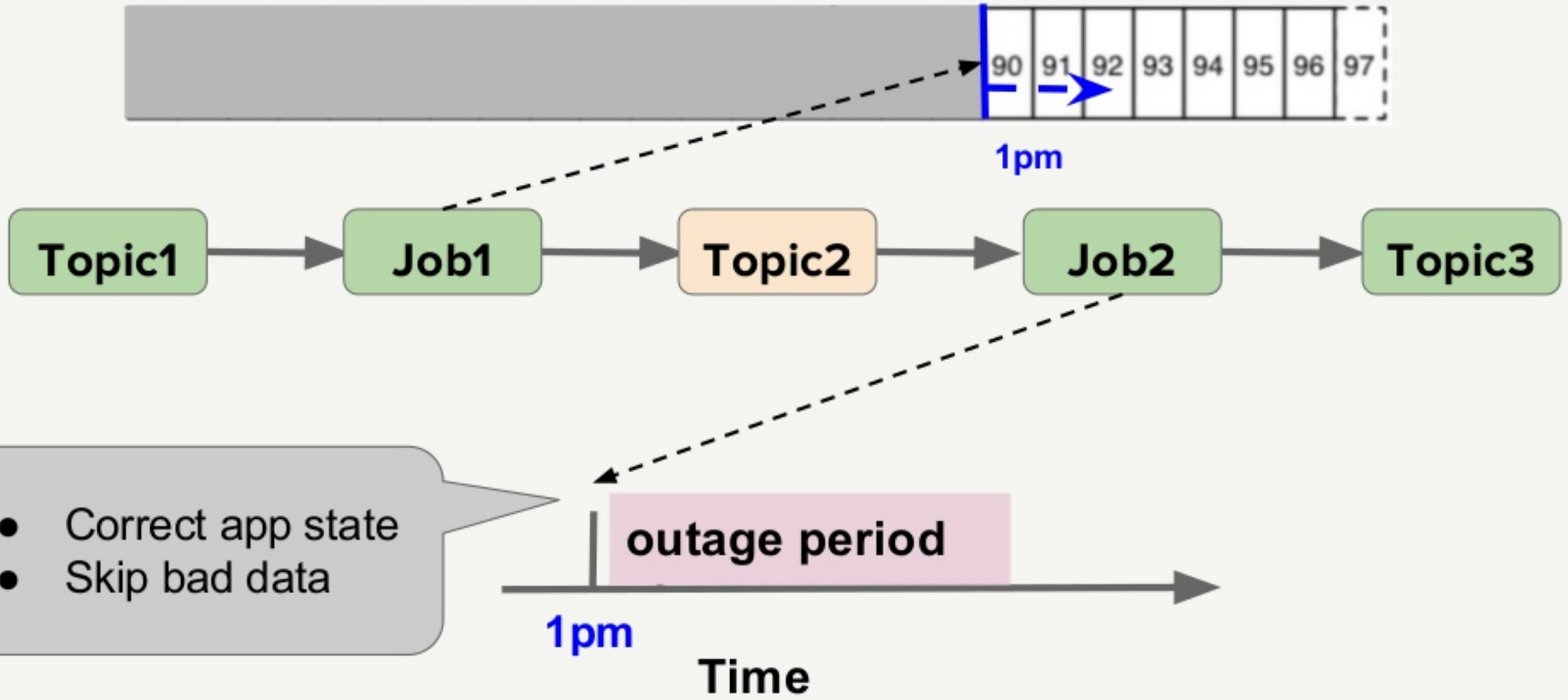
Wipe out all messages from topic2

All bad data are gone!

Rewind job2 to 1pm checkpoint



Rewind job1 to 1pm checkpoint



It is difficult to execute

- Very involved process
- Need coordination btw job1 and job2

Caveats recap

- Don’t overwhelm external services
- Your dependency may not participate in rewind
- Watch out for the impact to downstream consumers

What defines us is
how well we rise
after falling

Steven Wu
NETFLIX

@stevenzwu