A Tour of Apache Beam's New Iceberg Connector

Ahmed Abualsaud

BEAM SUMMIT
NYC 2025

Agenda

- What is Iceberg?
- Iceberg Sink
  - Dynamic destinations
  - Batch and streaming capabilities
  - Examples - Java, Python, YAML, SQL
- Iceberg Source
  - Examples - Java, Python, YAML, SQL
- Iceberg CDC Source
  - Capabilities
  - Batch and streaming examples
- Future Areas of Growth

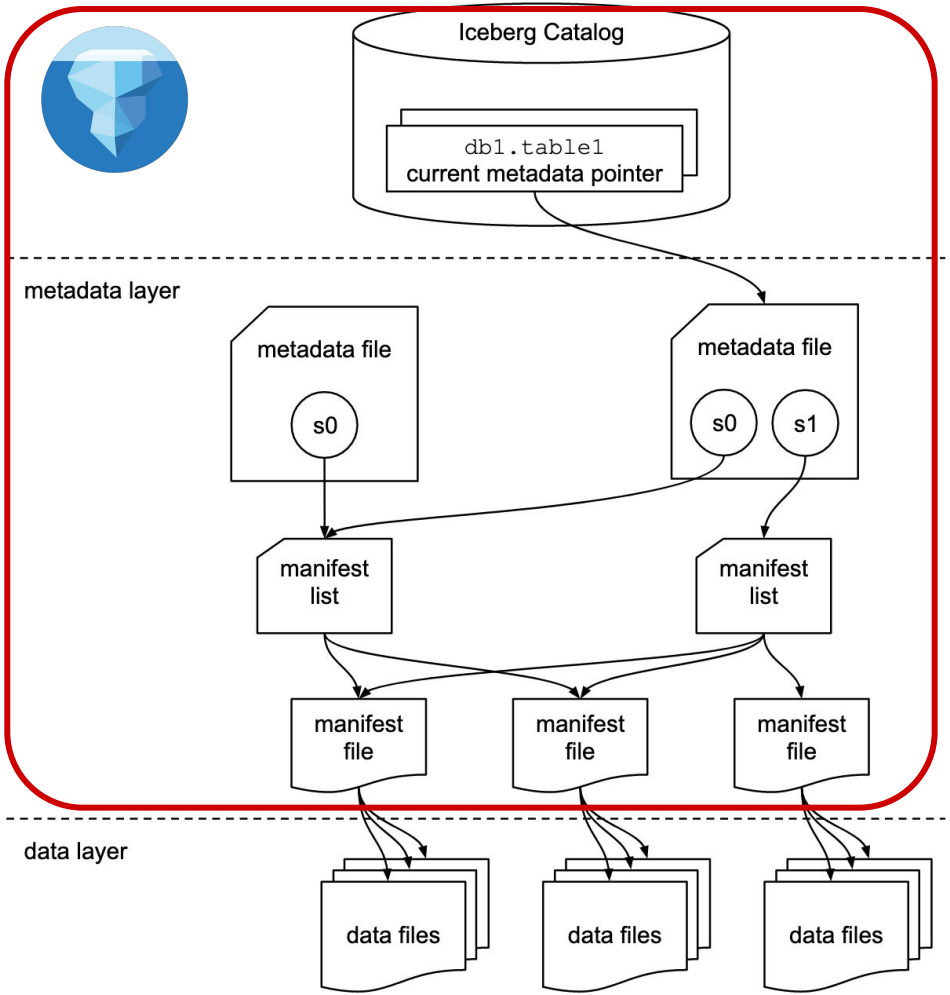
What is Iceberg?

What is Iceberg?

- **Open table format**
  - A specification/blueprint for organizing **data files** and **metadata files** to create a logical table
- Built to handle **massive datasets**
- **Sophisticated metadata layer** on top of raw data files
- Maintains a **versioned history of every change** to the table's:
  - **data** (inserts, updates, deletes)
  - **structure** (e.g. schema, partition spec)

Iceberg table structure

youtube.com/watch?v=TsmhRZElPvM



Why use Iceberg?

Highly interoperable (not tied to a single cloud provider or engine)

Highly scalable

Fast, efficient queries

Schema evolution

Time travel

ACID transactions - reliability and consistency

Active community – continuous improvements

Beam's Iceberg Connectors

Notice: Iceberg IOs are offered as Managed Transforms

Managed transforms - power of Beam without maintenance overheads

Jul-8 11:45-12:10 in Star Leaved Gum Add to Calendar

Configuration properties:

https://beam.apache.org/documentation/io/managed-io/

Beam's Iceberg Sink

Iceberg Sink

Writes to Iceberg table(s)

Creates namespaces and tables when needed

- Table schema is inferred from data schema
- User-specified partition spec

Supports writing to dynamic destinations

- i.e. an element’s table destination is determined by its field values

Iceberg Sink - Dynamic Destinations

Input record

```
{airlines="AA", origin="JFK",
destination="MAD", duration=430}
```

Iceberg Sink - Dynamic Destinations

table: table destination template

Input record

```
{airlines="AA", origin="JFK",
destination="MAD", duration=430}
```

```
table=
"flights_{airlines}.from_{origin}"
```

Iceberg Sink - Dynamic Destinations

table: table destination template

Input record

```
{airlines="AA", origin="JFK",
destination="MAD", duration=430}
```

```
table=
"flights_{airlines}.from_{origin}"
```

Written to Iceberg table:

`"flights_AA.from_JFK"`

Iceberg Sink - Dynamic Destinations

table: table destination template

keep: exclusively keeps columns and drops others before writing

Input record

```
{airlines="AA", origin="JFK",
destination="MAD", duration=430}
```

```
keep=["duration",
"destination"]
```

```
table=
"flights_{airlines}.from_{origin}"
```

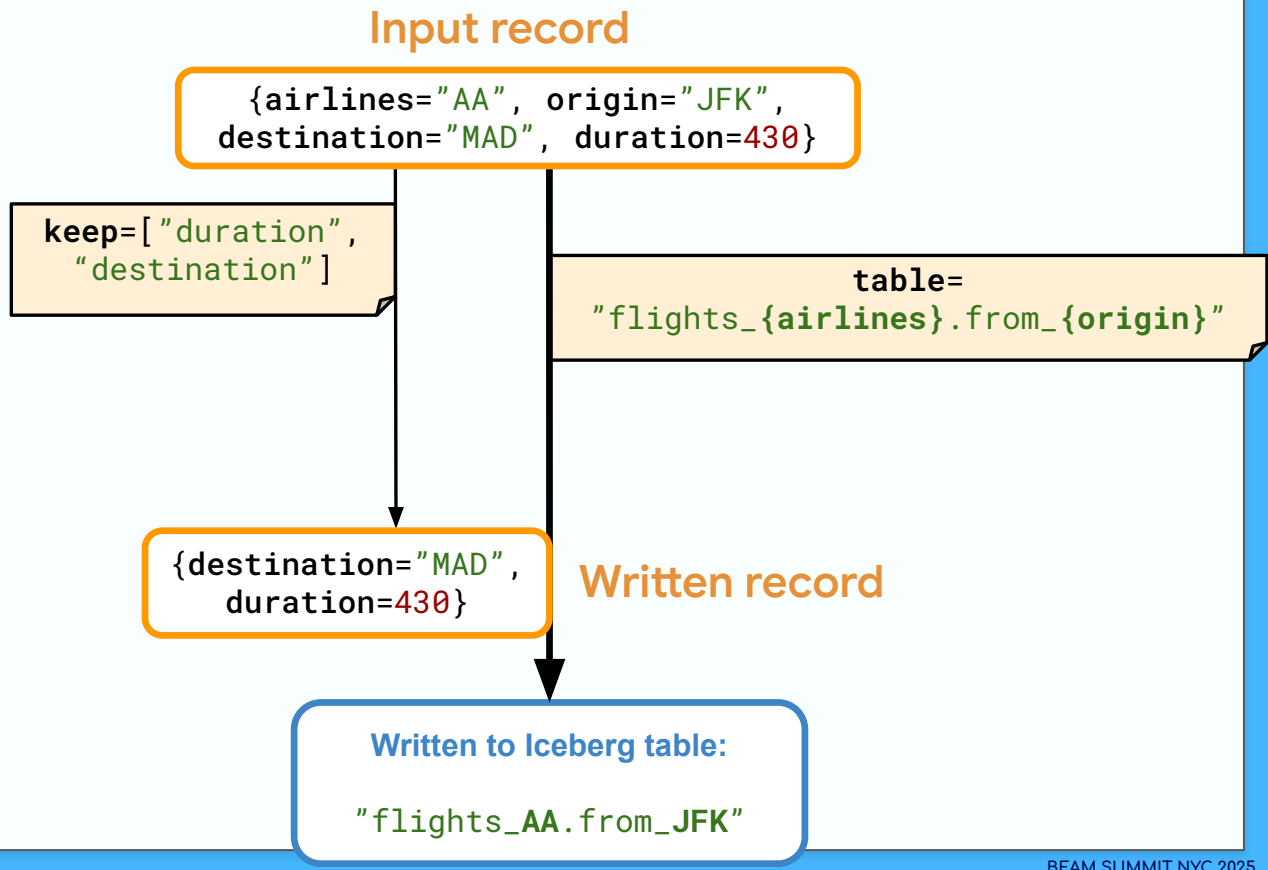
Written to Iceberg table:

```
"flights_AA.from_JFK"
```

Iceberg Sink - Dynamic Destinations

table: table destination template

keep: exclusively keeps columns and drops others before writing

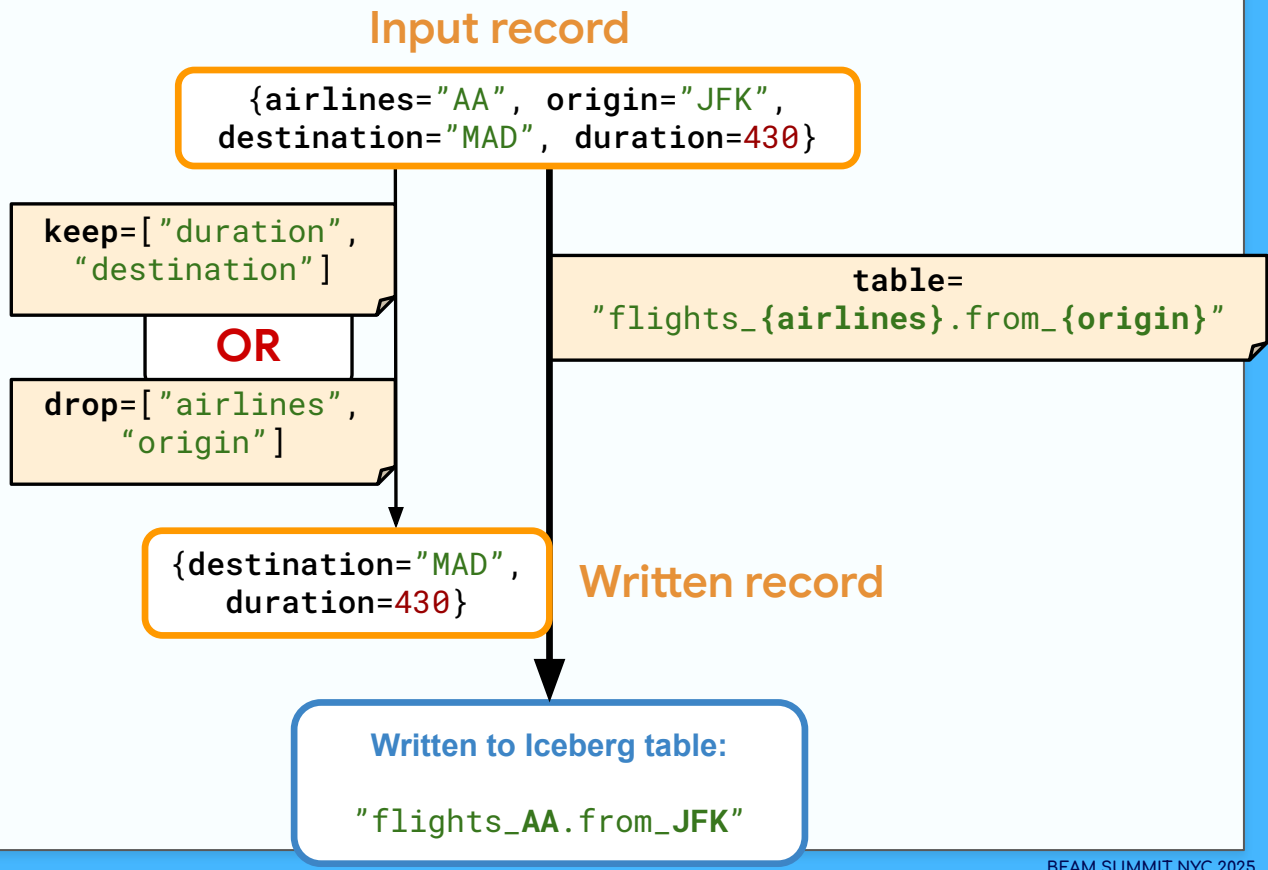


Iceberg Sink - Dynamic Destinations

table: table destination template

keep: exclusively keeps columns and drops others before writing

drop: drops columns before writing



Iceberg Sink - example (Java)

```
Map<String, Object> config = Map.of(
    "table", "flights_{airlines}.from_{origin}",
    "drop", Arrays.asList("airlines", "origin"),
    "partition_fields", Arrays.asList(
        "plane_model", "month(departure)", "bucket(destination, 3)"),
    "catalog_properties", Map.of(...));

PCollection<Row> input = ...;
input.apply(Managed.write("iceberg").withConfig(config)));
```

Iceberg Sink - example (Java)

```
Map<String, Object> config = Map.of(
    "table", "flights_{airlines}.from_{origin}",
    "drop", Arrays.asList("airlines", "origin"),
    "partition_fields", Arrays.asList(
        "plane_model", "month(departure)", "bucket(destination, 3)"),
    "catalog_properties", Map.of(...));

PCollection<Row> input = ...;
input.apply(Managed.write("iceberg").withConfig(config)));
```

Iceberg Sink - example (Java)

```
Map<String, Object> config = Map.of(
    "table", "flights_{airlines}.from_{origin}",
    "drop", Arrays.asList("airlines", "origin"),
    "partition_fields", Arrays.asList(
        "plane_model", "month(departure)", "bucket(destination, 3)"),
    "catalog_properties", Map.of(...));

PCollection<Row> input = ...;
input.apply(Managed.write("iceberg").withConfig(config)));
```

Supports partitioning transforms!

Iceberg Sink - Batch

Writes data files to the appropriate partitions

e.g.

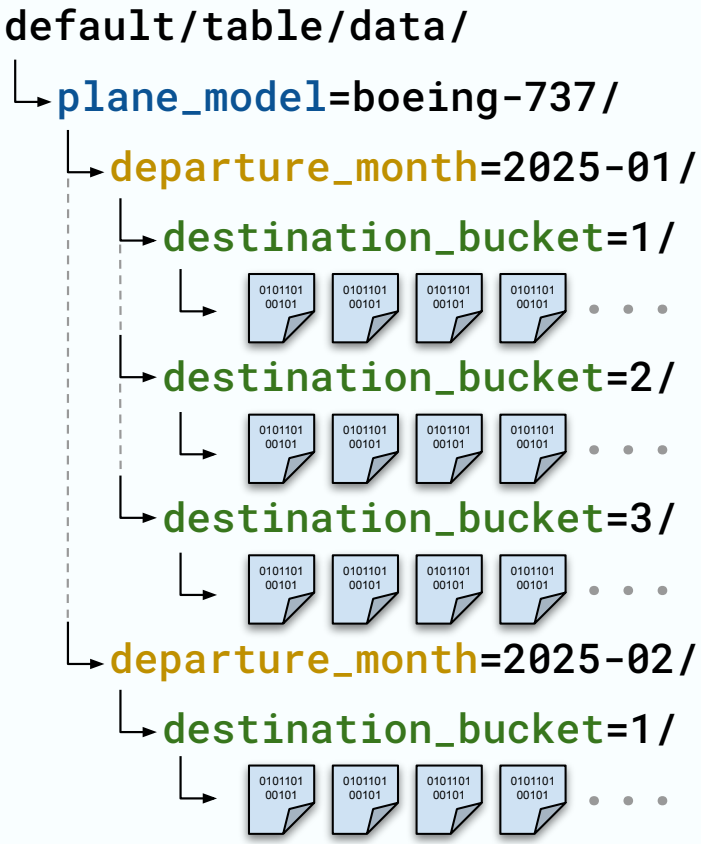
```
"plane_model",
"month(departure)",
"bucket(destination, 3)"
```

Iceberg Sink - Batch

Writes data files to the appropriate partitions

e.g.

```
"plane_model",
"month(departure)",
"bucket(destination, 3)"
```



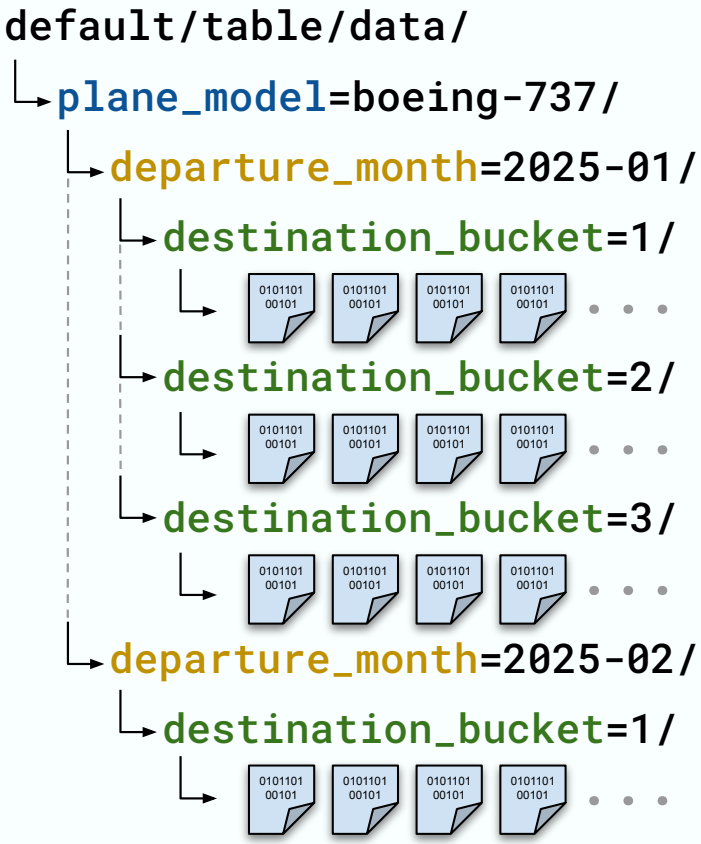
Iceberg Sink - Batch

Writes data files to the appropriate partitions

e.g.

```
"plane_model",
"month(departure)",
"bucket(destination, 3)"
```

Single table commit for all data files (one snapshot)



Iceberg Sink - Streaming

Enable streaming writes by specifying a triggering frequency:

```
triggering_frequency_seconds=60
```

Buffers records in state, then writes and commits data files according to the triggering frequency.

Streaming writes adapt to changes in the table partition spec

Iceberg Sink - Streaming

Streaming sinks inevitably run into the infamous **small-files issue**:

Iceberg queries are slower when the data is dispersed over **many small** files

- longer time to plan queries
- many file open operations

Iceberg Sink - Streaming

Mitigation: writing over longer intervals (i.e. larger `triggering_frequency_seconds`)

Larger triggering frequency	Larger, fewer data files	More data buffered in state
Smaller triggering frequency	Smaller, more numerous data files	Less data buffered in state

Ultimately, the standard solution is **continuous file compaction**.

Iceberg Sink - Java example

```
Map<String, Object> config = Map.of(
    "table", "flights_{airlines}.from_{origin}",
    "drop", Arrays.asList("airlines", "origin"),
    "partition_fields", Arrays.asList(
        "plane_model", "month(departure)", "bucket(destination, 3)"),
    "triggering_frequency_seconds", 10,
    "catalog_properties", Map.of(...));

PCollection<Row> input = ...;
input.apply(Managed.write("iceberg").withConfig(config)));
```

Iceberg Sink - Python example

```python
with beam.Pipeline() as p:
  input_rows = ...

  input_rows | beam.managed.Write(
    "iceberg", config={
      "table": "flights_{airlines}.from_{origin}",
      "drop": ["airlines", "origin"],
      "partition_fields": [
          "plane_model",
          "month(departure)",
          "bucket(destination, 3)"],
      "triggering_frequency_seconds": 10,
      "catalog_properties": {...}
    }
  )
```

Iceberg Sink - YAML example

```yaml
pipelines:
  - pipeline:
      type: chain
      transforms:
        - type: Create
          config:
            elements:
              - ...
        - type: WriteToIceberg
          config:
            table: "flights_{airlines}.from_{origin}"
            drop: ["airlines", "origin"]
            partition_fields:
              - "plane_model"
              - "month(departure)"
              - "bucket(destination, 3)"
            triggering_frequency_seconds: 10
            catalog_properties: ...
```

Iceberg Sink - SQL example

```
CREATE CATALOG my_catalog
TYPE 'iceberg'
PROPERTIES (
  'foo'='bar',
  ...
)


USE CATALOG my_catalog
```

```
CREATE EXTERNAL TABLE test_table(
    id BIGINT,
    name VARCHAR
)
TYPE 'iceberg'
PARTITIONED BY(
  'plane_model',
  'month(departure)',
  'bucket(destination, 3)'
)
LOCATION 'flights_AA.from_JFK'
TBLPROPERTIES {
  “triggering_frequency_seconds”: 10
}


INSERT INTO test_table
SELECT * FROM other_table
```

Beam's Iceberg Source

Iceberg Source

Data filtering with predicate pushdown

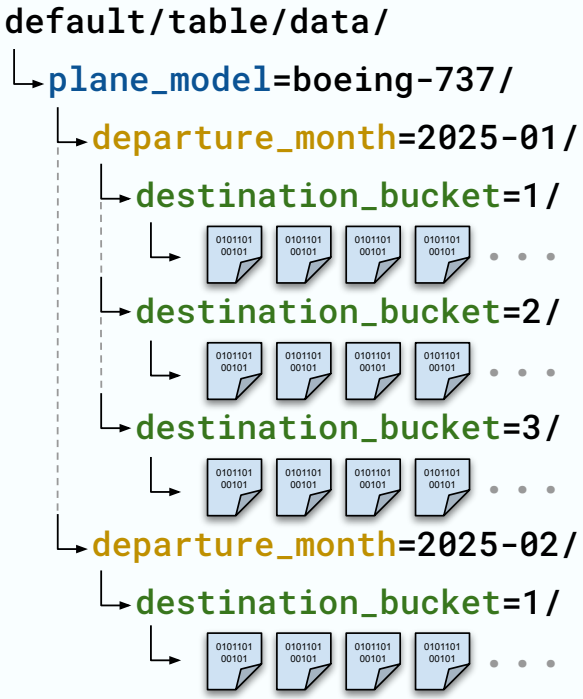
Column pruning with projection pushdown

Iceberg Source

Data filtering with predicate pushdown

Column pruning with projection pushdown

Example: looking for departures in February 2025 →

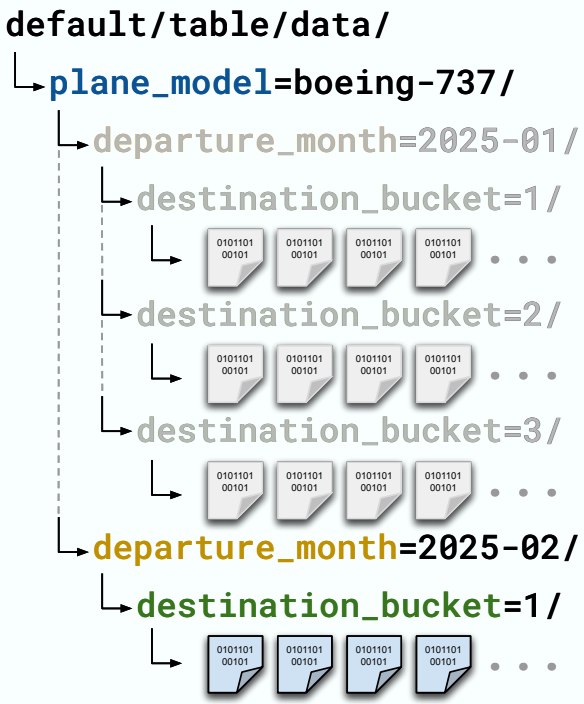


Iceberg Source

Data filtering with predicate pushdown

Column pruning with projection pushdown

Example: looking for departures in February 2025 →



Iceberg Source - Java example

```
Map<String, Object> config = Map.of(
    "table", "flights_AA.from_JFK",
    "filter", "departure > '2025-03-01' AND destination = 'MAD'",
    "keep", ["plane_model", "duration"],
    "catalog_properties", Map.of(...));


PCollection<Row> input = pipeline.apply(
    Managed.read("iceberg").withConfig(config)));
```

Iceberg Source - Python example

```python
with beam.Pipeline() as p:

  input_rows = beam.managed.Read(
    "iceberg", config={
      "table": "flights_AA.from_JFK",
      "filter", "departure > '2025-03-01' AND destination = 'MAD'",
      "keep", ["plane_model", "duration"],
      "catalog_properties": {...}
    }
  )
```

Iceberg Source - YAML example

```yaml
pipelines:
  - pipeline:
      type: chain
      transforms:
        - type: ReadFromIceberg
          config:
            table: "flights_AA.from_JFK"
            filter: "departure > '2025-03-01' AND
                     destination = 'MAD'"
            keep: ["plane_model", "duration"]
            catalog_properties: ...
```

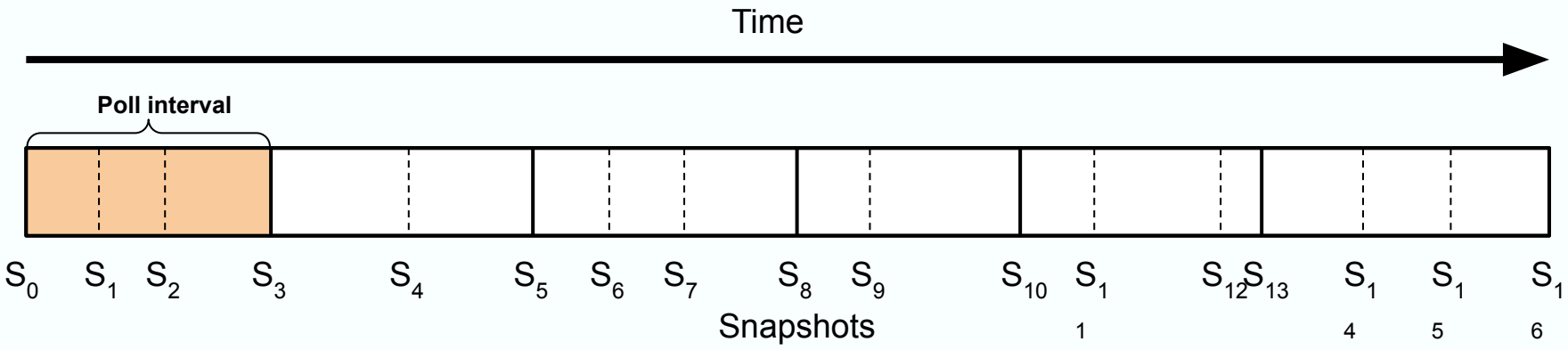
>= 2.67.0

Iceberg Source - SQL example

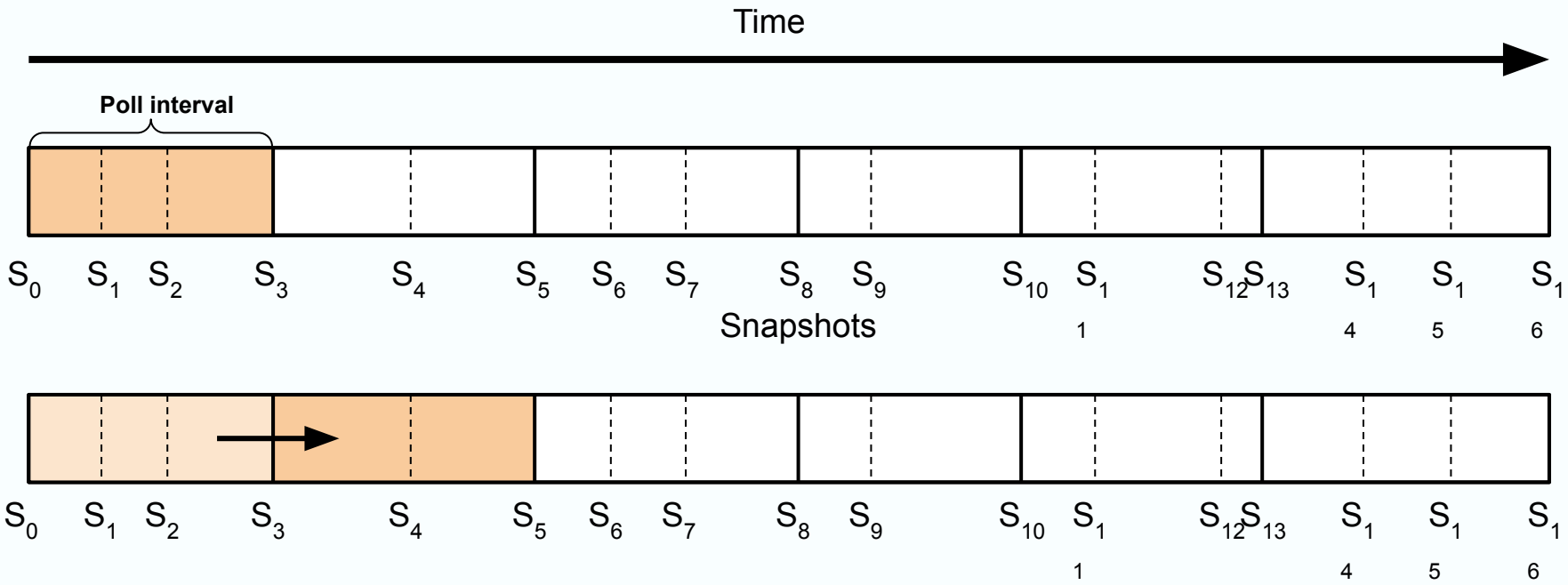
```sql
SELECT plane_model, duration
FROM test_table
WHERE departure > '2025-03-01'
AND destination = 'MAD'
```

Beam's Iceberg CDC Source

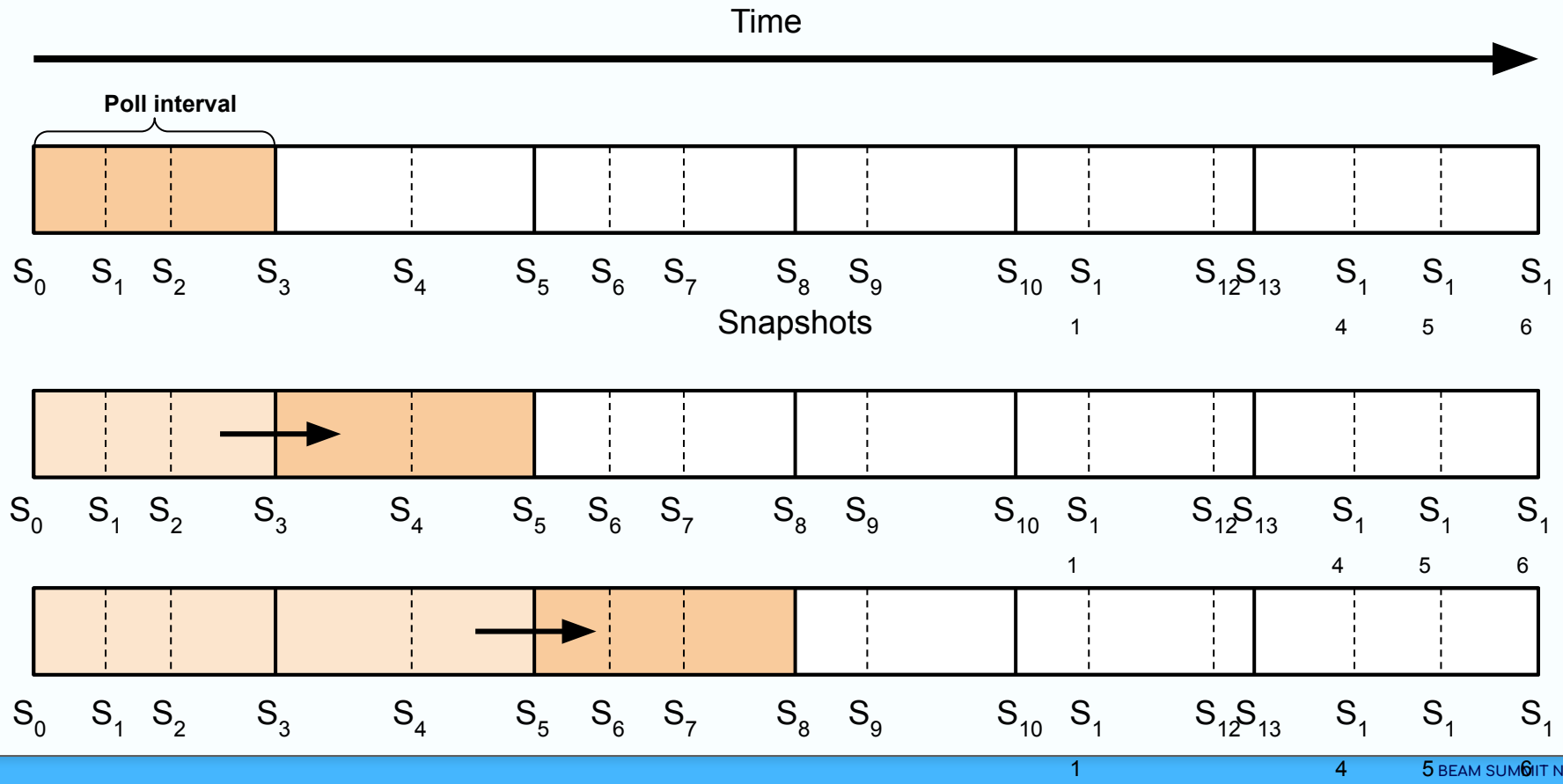
Iceberg CDC Source



Iceberg CDC Source



Iceberg CDC Source



Iceberg CDC Source

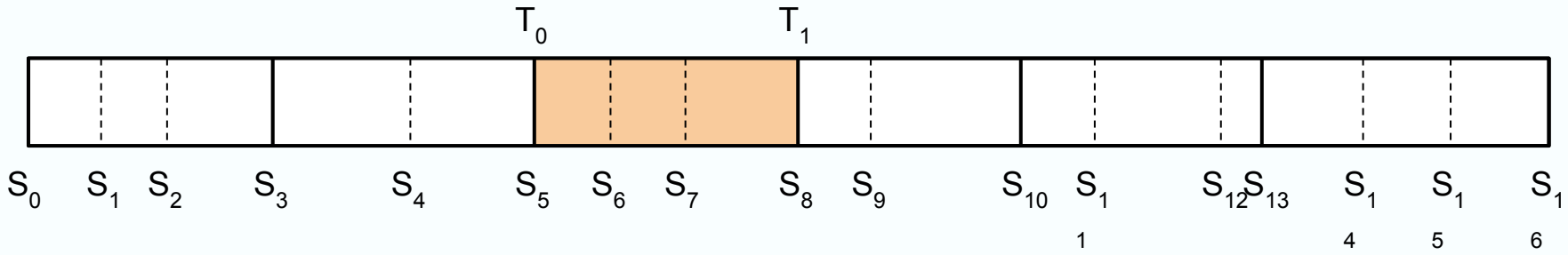
Available for batch and streaming reads (append-only snapshots)

Can read within a specified snapshot or timestamp boundary

Iceberg CDC Source

Available for batch and streaming reads (append-only snapshots)

Can read within a specified snapshot or timestamp boundary



Iceberg CDC Source - Java example

Batch

```
Map<String, Object> config = Map.of(
    "table", "flights_AA.from_JFK",
    "keep", ["plane_model", "duration"],
    "filter", "destination = 'MAD'",
    "from_snapshot", 1234567898765 43L,
    "to_snapshot", 975149332612356L,
    "catalog_properties", Map.of(...));


PCollection<Row> input = pipeline.apply(
    Managed.read("iceberg_cdc").withConfig(config)));
```

Iceberg CDC Source - Java example

Streaming

```
Map<String, Object> config = Map.of(
    "table", "flights_AA.from_JFK",
    "keep", ["plane_model", "duration"],
    "filter", "destination = 'MAD'",
    "streaming", true,
    "poll_interval_seconds", 5,
    "from_timestamp", 1751910395L,
    "catalog_properties", Map.of(...));


PCollection<Row> input = pipeline.apply(
    Managed.read("iceberg_cdc").withConfig(config)));
```

Future Areas of Growth

Full CDC writes and reads

Batch compaction

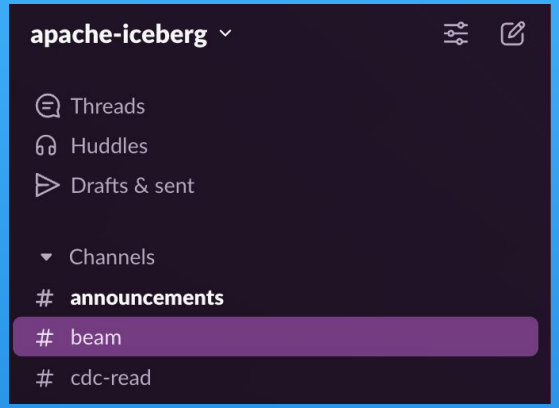
Continuous streaming compaction

Other admin/maintenance functions

Performance benchmarks

Ahmed Abualsaud

QUESTIONS?



a.abualsaud98@gmail.com

linkedin.com/in/ahmedabu98

github.com/ahmedabu98

BEAM SUMMIT
NYC 2025