
AWS
re:Invent
DECEMBER 2 – 6, 2024 | LAS VEGAS, NV

NFX303

Efficient incremental processing with Apache Iceberg at Netflix

Jun He

(he/him)
Staff Software Engineer
Netflix

Vaidy Krishnan

Strategic Account Manager
AWS

Data

Big data

Data accuracy

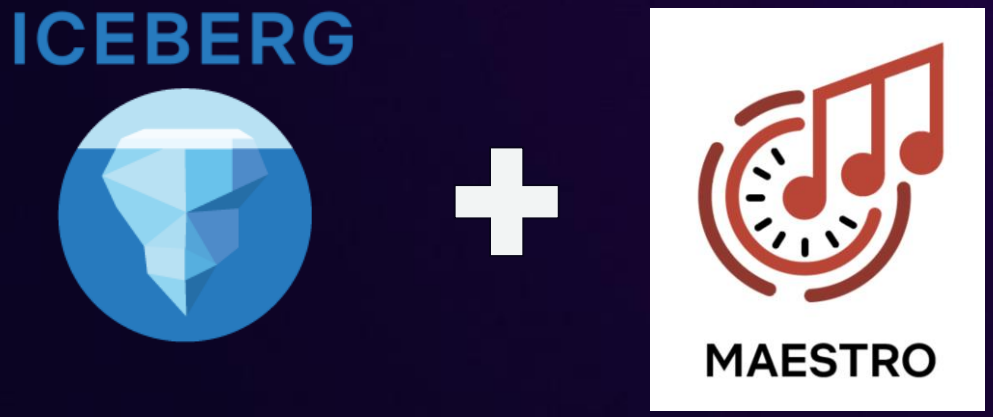
Data freshness

Cost efficiency

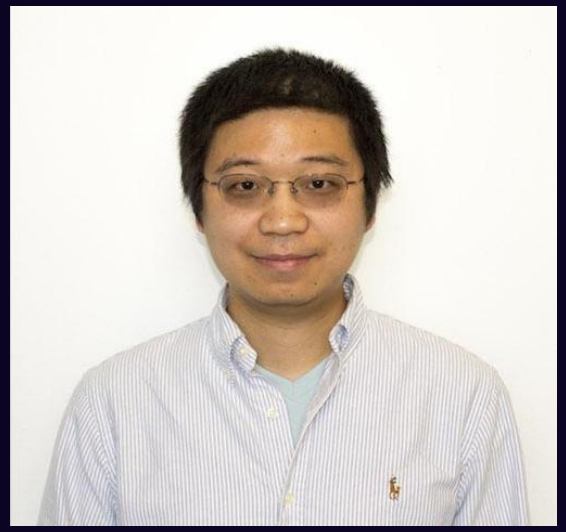
Apache Iceberg

Netflix

Netflix Maestro



About us



Jun He

(he/him)
Staff Software Engineer
Netflix

Vaidy Krishnan

Strategic Account Manager
AWS

Outline

01 Introduction

02 Architectural design

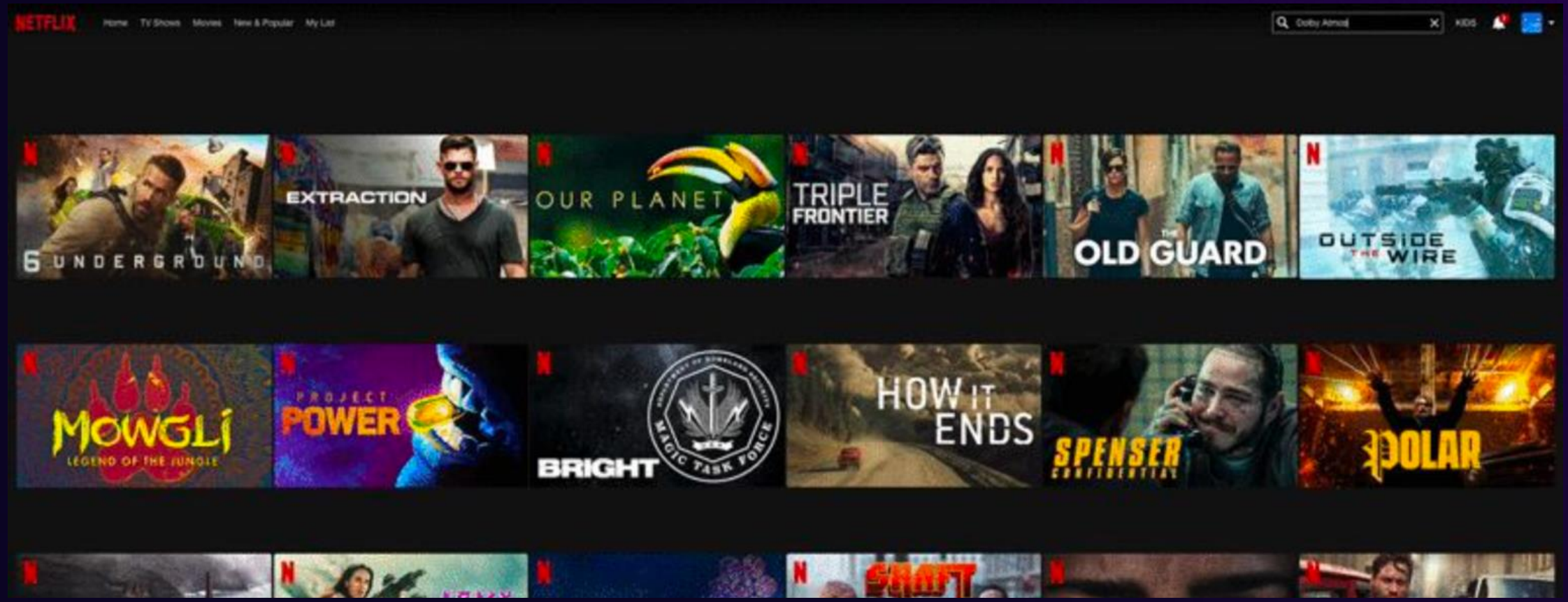
03 Use cases and examples

04 Takeaways and future improvements

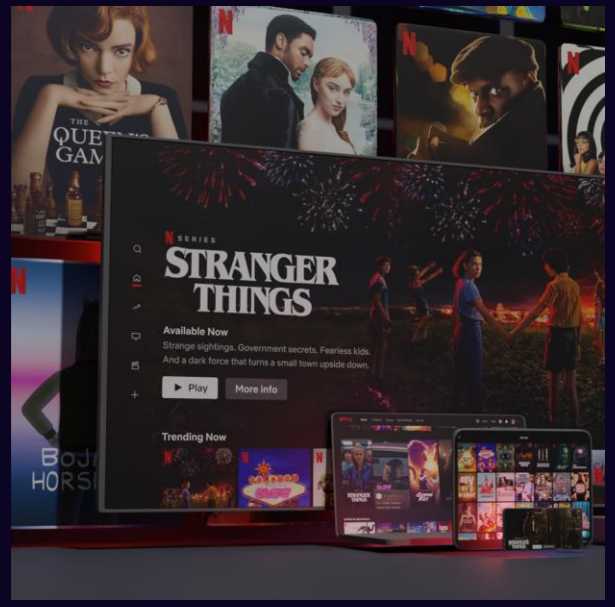
Introduction

Introduction

Landscape of data insights at Netflix

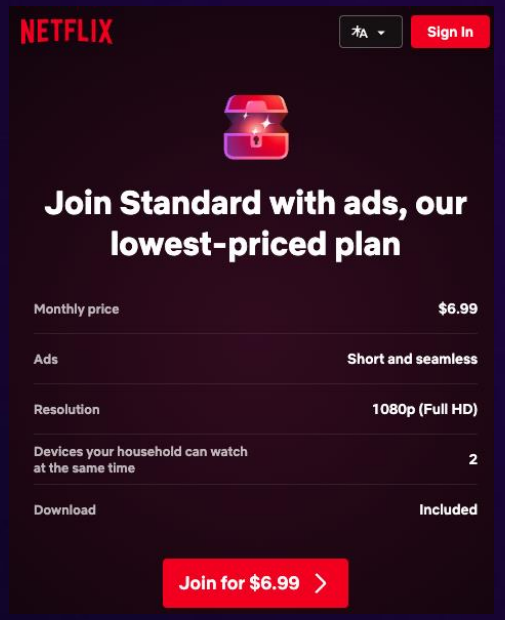


Data for business needs



Streaming

Games



Ads

Live

Common problems

Data accuracy

Exabyte data warehouse

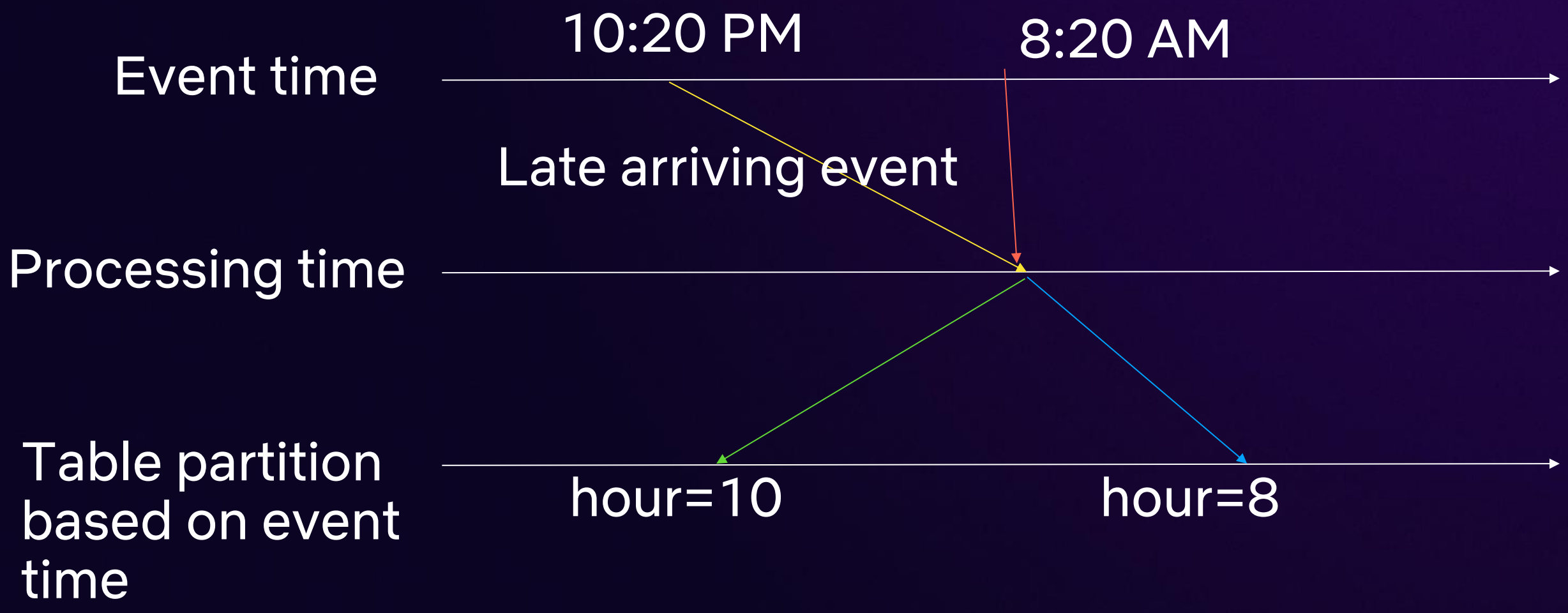
Data freshness

Business needs for new initiatives

Cost efficiency

More than $150M per year

Late arriving data



Big Data Analytics Platform

BDAP tech stack

and other BDAP internal services

Existing solutions

Lookback window

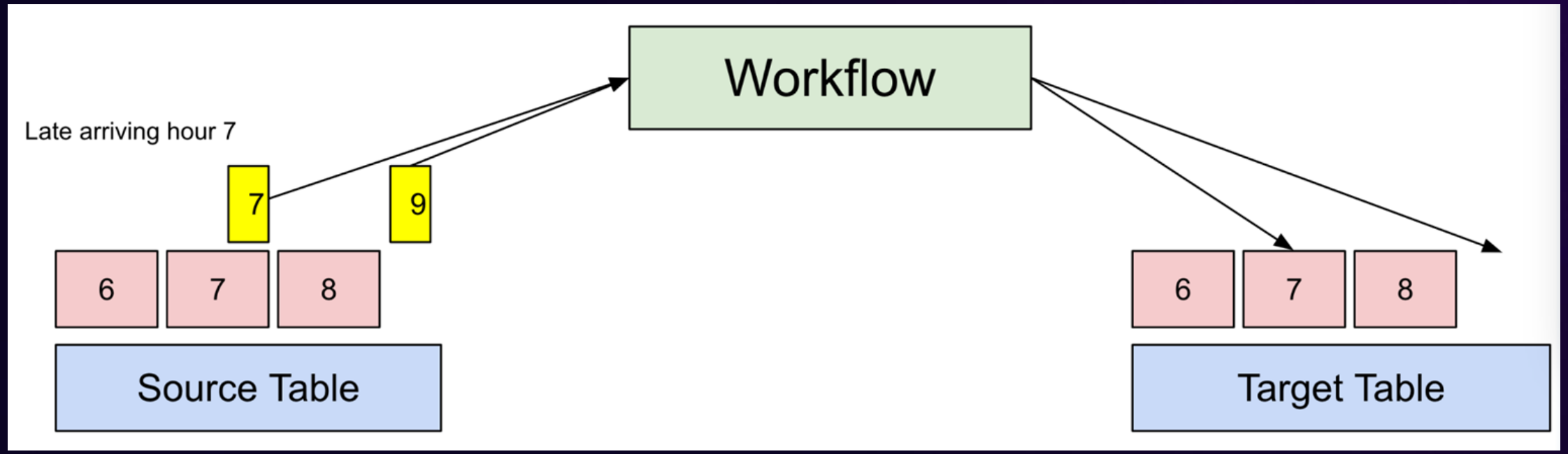
- Data accuracy

Ignore late arriving data

- Data freshness
- Cost efficiency

Wait then process

- Data accuracy
- Cost efficiency



Incremental processing

Incremental processing is an approach to processing data in batches — but only new or changed data

- Capturing incremental data changes
- Tracking their states (i.e., whether a change is processed by a workflow or not)

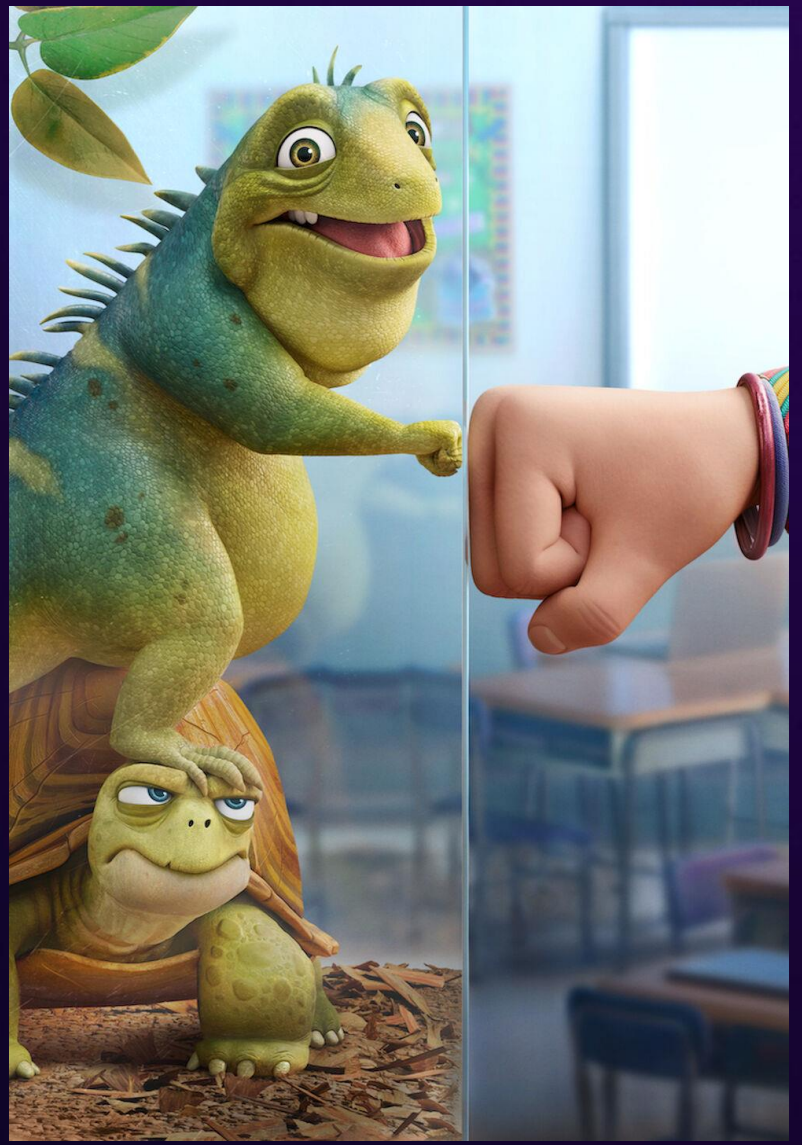
Incremental processing support (IPS)

Key enablers

ICEBERG

+

MAESTRO



Apache Iceberg

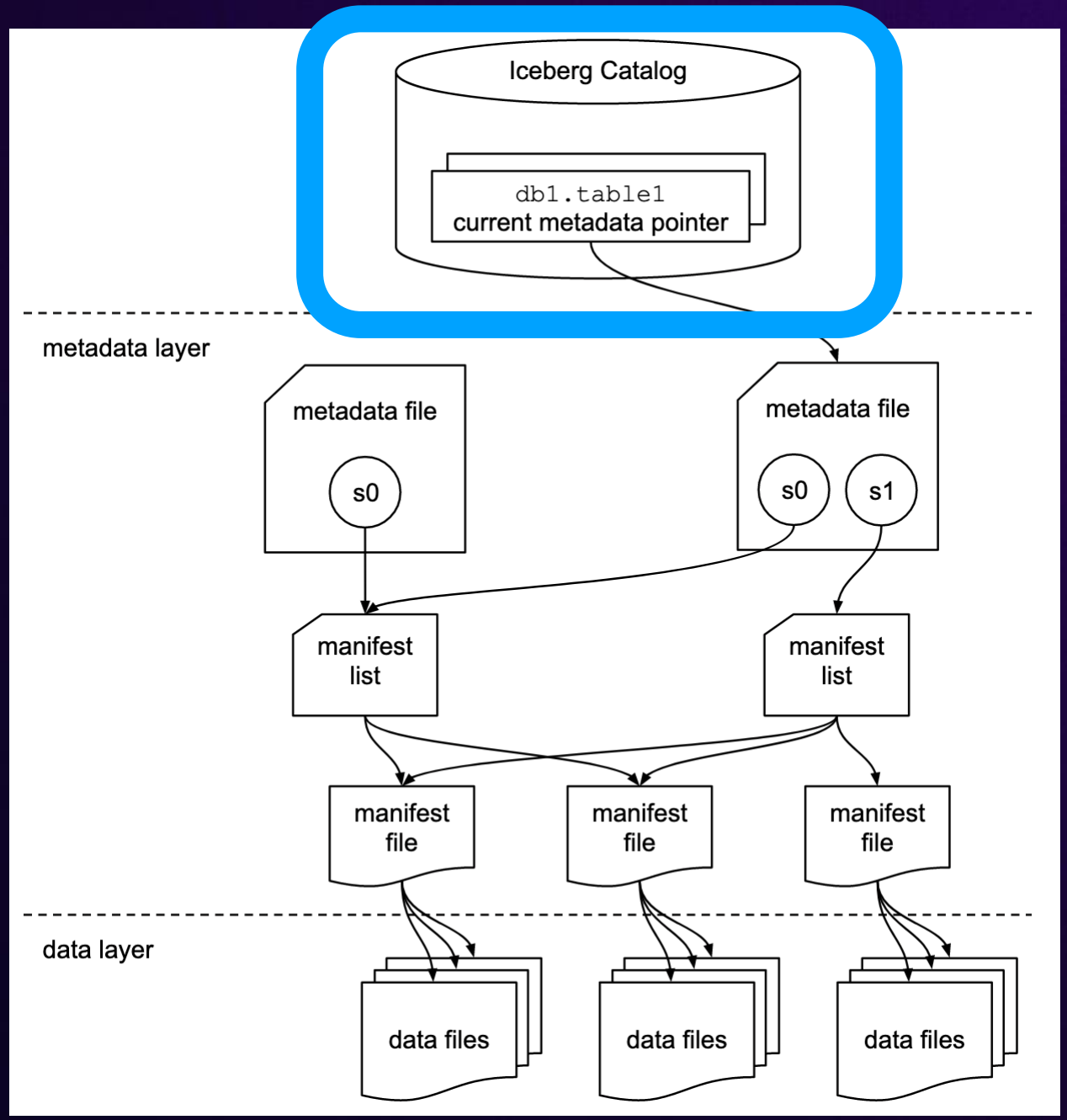
Iceberg is a high-performance format for huge analytic tables

- Open table format
- Open community standard
- Decentralized metadata
- Storage separation
- Schema and partition evolution
- Ease of data management

Rich information from the metadata layer

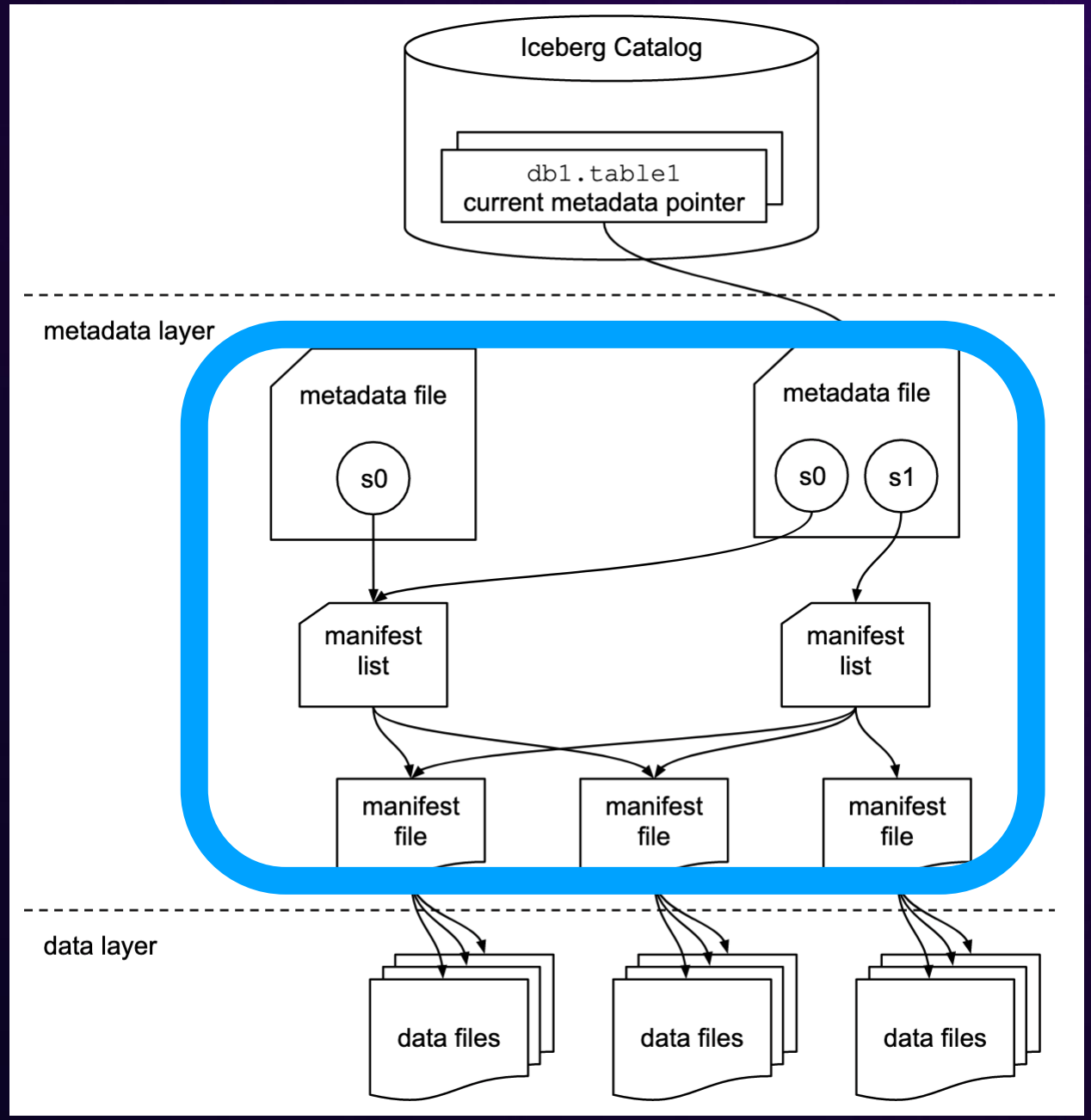
Iceberg concepts

- Catalog
- Table
- Snapshot
- Data file
- Partition



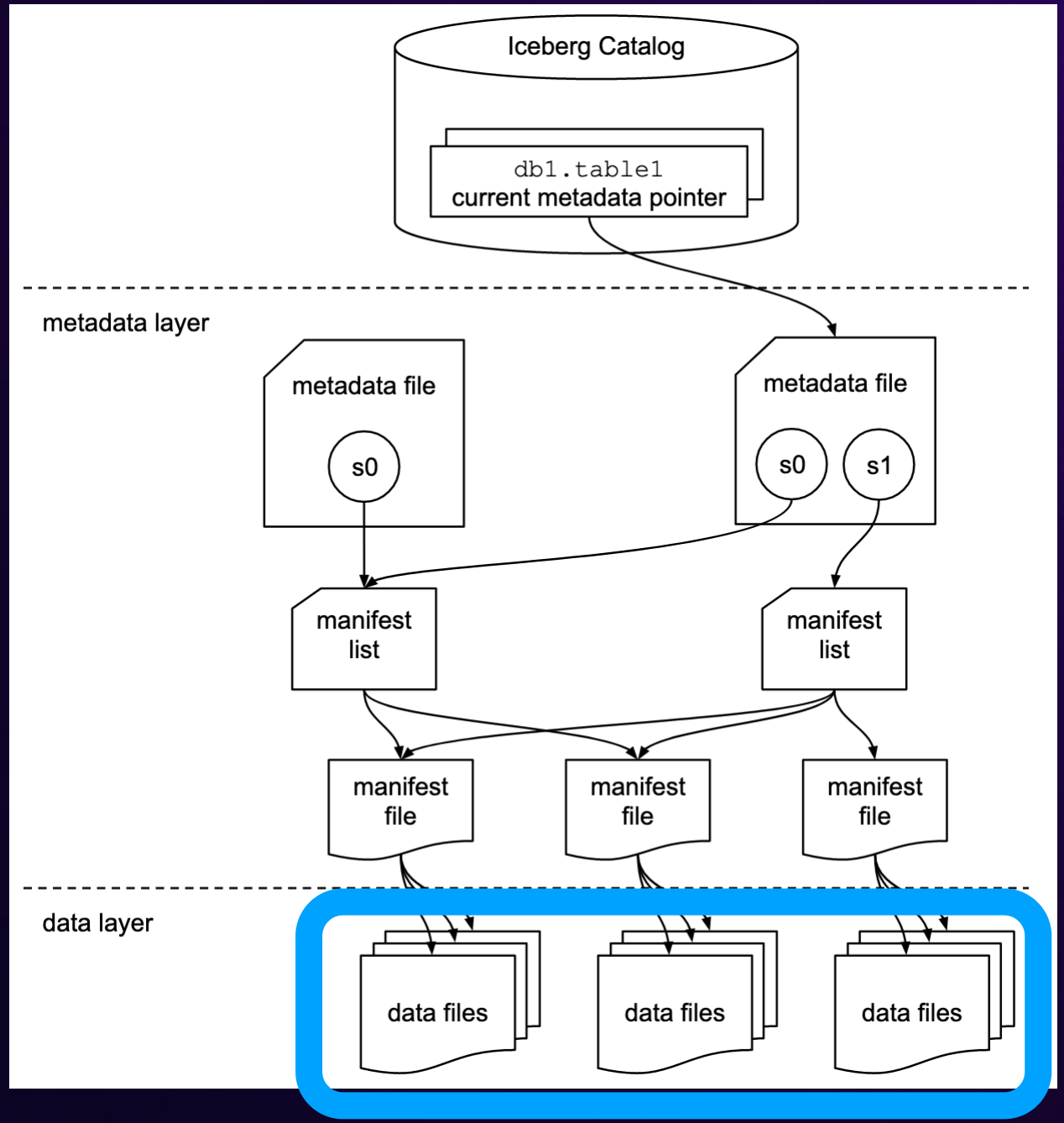
Iceberg concepts

- Catalog
- Table
- Snapshot
- Data file
- Partition



Iceberg concepts

- Catalog
- Table
- Snapshot
- Data file
- Partition



What is Maestro

- Horizontally scalable workflow orchestrator
- Oversee the entire lifecycle of a workflow
- Serverless execution from user perspective
- Support both acyclic and cyclic workflows
- Multiple out-of-the-box reusable patterns
- Designed for expansion and integrations with others

Maestro

Netflix data/ML workflow orchestrator

- Started in March 2020
- Alpha in September 2021
- Beta in April 2022
- Migrated all traffic in October 2023

- GA in November 2022
- Open to the public in July 2024
- New features in progress

Workflow Example

YAML definition

```yaml
Description: |
  This is a sample description to [a Maestro
workflow](https://github.com/Netflix/maestro)
 cron: '@daily'
Workflow:
 id: demo.pipeline
 my_query: INSERT OVERWRITE TABLE
${TARGET_TABLE} SELECT USER_ID, DATE_INT,
SUM(watch_time) FROM ${SOURCE_TABLE} WHERE
is_active=true AND DATE_INT > DATE_MINUS_7
group by USER_ID, DATE_INT;
 jobs:
   - job:
       id: job.1
       type: Spark
       spark:
         script: ${my_query}
```

```
$ scheduler push workflow.yaml
$ scheduler run demo.pipeline my_query="SELECT
1"

Workflow run has been successfully submitted.
View on data.netflix:
https://data.netflix.net/maestro/sandbox/demo.p
ipeline/instances/1
```

Workflow Example

YAML definition

```
Description: |
  This is a sample description to [a Maestro
Trigger:
 cron: '@daily'
 id: demo.pipeline
 my_query: INSERT OVERWRITE TABLE
${TARGET_TABLE} SELECT USER_ID, DATE_INT,
SUM(watch_time) FROM ${SOURCE_TABLE} WHERE
is_active=true AND DATE_INT > DATE_MINUS_7
group by USER_ID, DATE_INT;
 jobs:
   - job:
       id: job.1
       type: Spark
       spark:
         script: ${my_query}
```

```
$ scheduler push workflow.yaml
$ scheduler run demo.pipeline my_query="SELECT
1"

Workflow run has been successfully submitted.
View on data.netflix:
https://data.netflix.net/maestro/sandbox/demo.p
ipeline/instances/1
```

Workflow Example

YAML definition

```yaml
Description: |
  This is a sample description to [a Maestro
workflow](https://github.com/Netflix/maestro)
Trigger:
 cron: '@daily'
Workflow:
 id: demo.pipeline
 my_query: INSERT OVERWRITE TABLE
${TARGET_TABLE} SELECT USER_ID, DATE_INT,
SUM(watch_time) FROM ${SOURCE_TABLE} WHERE
is_active=true AND DATE_INT > DATE_MINUS_7
group by USER_ID, DATE_INT;
```

```yaml
   - job:
       id: job.1
       type: Spark
       spark:
         script: ${my_query}
```

```
$ scheduler push workflow.yaml
$ scheduler run demo.pipeline my_query="SELECT
1"

Workflow run has been successfully submitted.
View on data.netflix:
https://data.netflix.net/maestro/sandbox/demo.p
ipeline/instances/1
```

Workflow Example

YAML definition

```yaml
Description: |
  This is a sample description to [a Maestro
workflow](https://github.com/Netflix/maestro)
Trigger:
 cron: '@daily'
Workflow:
 id: demo.pipeline
 my_query: INSERT OVERWRITE TABLE
${TARGET_TABLE} SELECT USER_ID, DATE_INT,
SUM(watch_time) FROM ${SOURCE_TABLE} WHERE
is_active=true AND DATE_INT > DATE_MINUS_7
group by USER_ID, DATE_INT;
 jobs:
   - job:
       id: job.1
       type: Spark
       spark:
         script: ${my_query}
```

```
$ scheduler push workflow.yaml
$ scheduler run demo.pipeline my_query="SELECT
1"

Workflow run has been successfully submitted.
View on data.netflix:
https://data.netflix.net/maestro/sandbox/demo.p
ipeline/instances/1
```

Workflow Example

YAML definition

```yaml
Description: |
  This is a sample description to [a Maestro
workflow](https://github.com/Netflix/maestro)
Trigger:
 cron: '@daily'
Workflow:
 id: demo.pipeline
 my_query: INSERT OVERWRITE TABLE
${TARGET_TABLE} SELECT USER_ID, DATE_INT,
SUM(watch_time) FROM ${SOURCE_TABLE} WHERE
is_active=true AND DATE_INT > DATE_MINUS_7
group by USER_ID, DATE_INT;
 jobs:
   - job:
       id: job.1
       type: Spark
       spark:
         script: ${my_query}
```

```
$ scheduler push workflow.yaml
$ scheduler run demo.pipeline my_query="SELECT
1"
```

```
Workflow run has been successfully submitted.
View on data.netflix:
https://data.netflix.net/maestro/sandbox/demo.p
ipeline/instances/1
```

Workflow Example

YAML definition

```yaml
Description: |
  This is a sample description to [a Maestro
workflow](https://github.com/Netflix/maestro)
Trigger:
 cron: '@daily'
Workflow:
 id: demo.pipeline
 my_query: INSERT OVERWRITE TABLE
${TARGET_TABLE} SELECT USER_ID, DATE_INT,
SUM(watch_time) FROM ${SOURCE_TABLE} WHERE
is_active=true AND DATE_INT > DATE_MINUS_7
group by USER_ID, DATE_INT;
 jobs:
   - job:
       id: job.1
       type: Spark
       spark:
         script: ${my_query}
```

```
$ scheduler push workflow.yaml
$ scheduler run demo.pipeline my_query="SELECT
1"

Workflow run has been successfully submitted.
View on data.netflix:
https://data.netflix.net/maestro/sandbox/demo.p
ipeline/instances/1
```

Workflow orchestrator for everyone

Interfaces

YAML
Python
JAVA
REST
GraphQL
Maestro UI

Integrations

Shell scripts
Notebooks
Java
Docker
SQL

Dynamic

Parameterized workflows
SEL
Input/output Parameters

Extensible

Job templates
Bring your own compute
Async execution

IPS Product briefs

Provide a clean and easy-to-adopt solution for the efficient incremental processing

- Data freshness
- Data accuracy
- Cost efficiency

Architectural design

Design of incremental processing support (IPS)

Efficient incremental change capturing

- DO NOT read any real data from the table(s)
- Use Apache Iceberg metadata layer

Best user experiences

- Decoupling change capturing from business logics
- Clean interfaces over Maestro
- Language and engine agnostic

Design of incremental change capturing

Apache Iceberg metadata provides all the needed information related to the new/updated data files

- Upper and lower bounds of a given column (partitioned or unpartitioned)
- Data file size, location

Build a zero-data-copy table including only change data

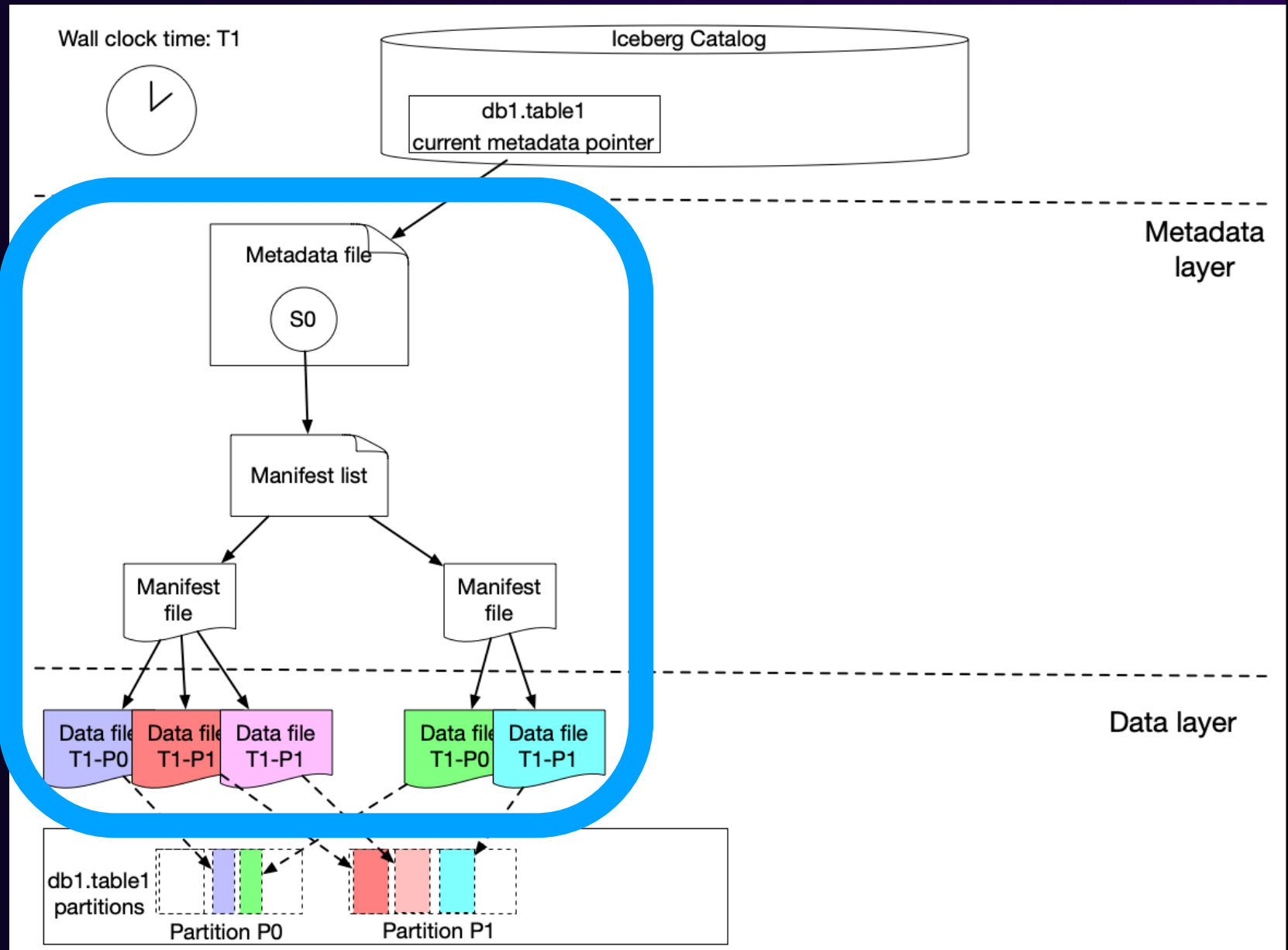
- No need for real data access
- No security concerns

Incremental change capturing

Iceberg table snapshot

Immutable data files

Hidden partitions

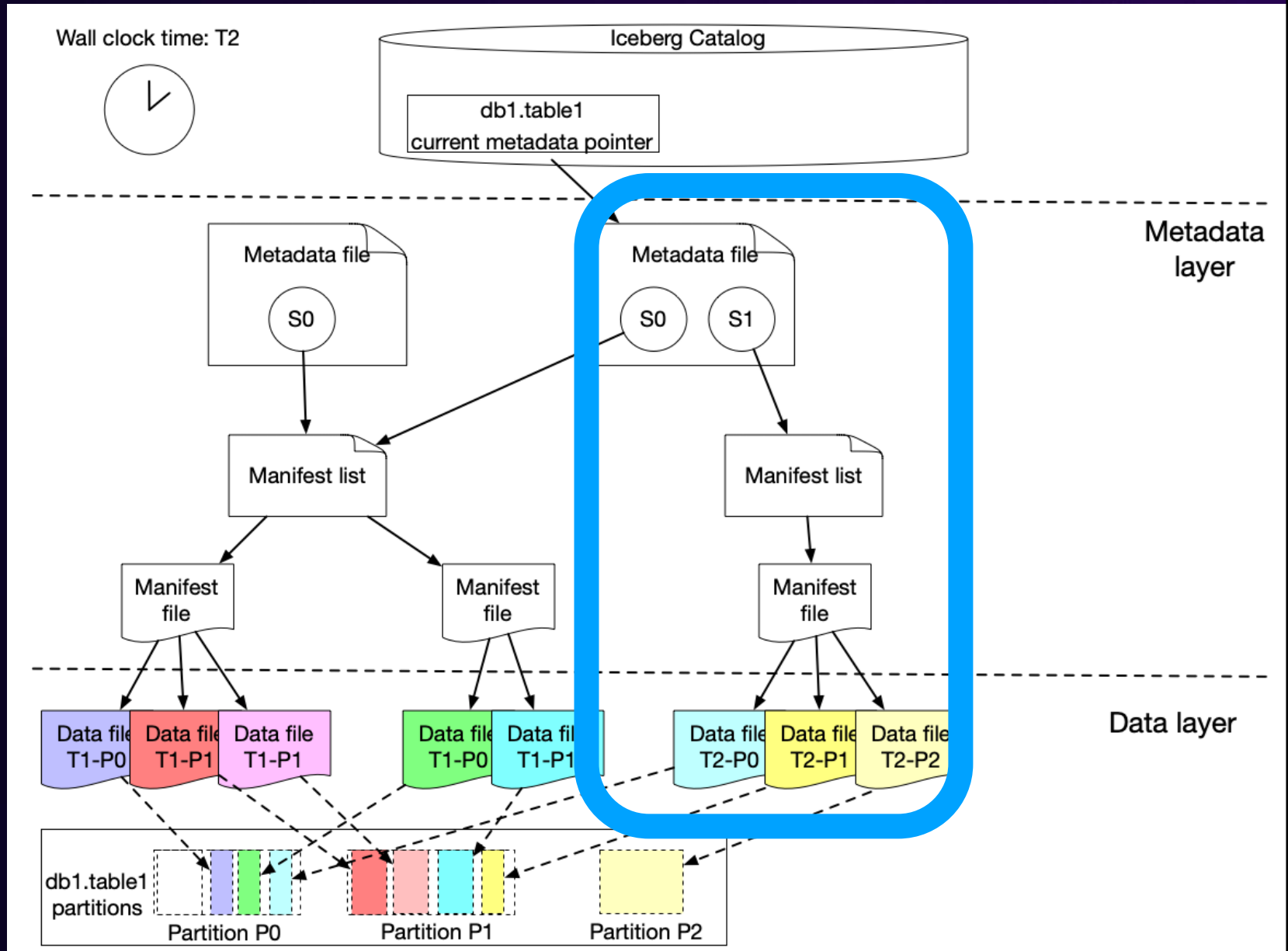


Incremental change capturing

New “append” snapshot

Added immutable data files

Mutated partitions

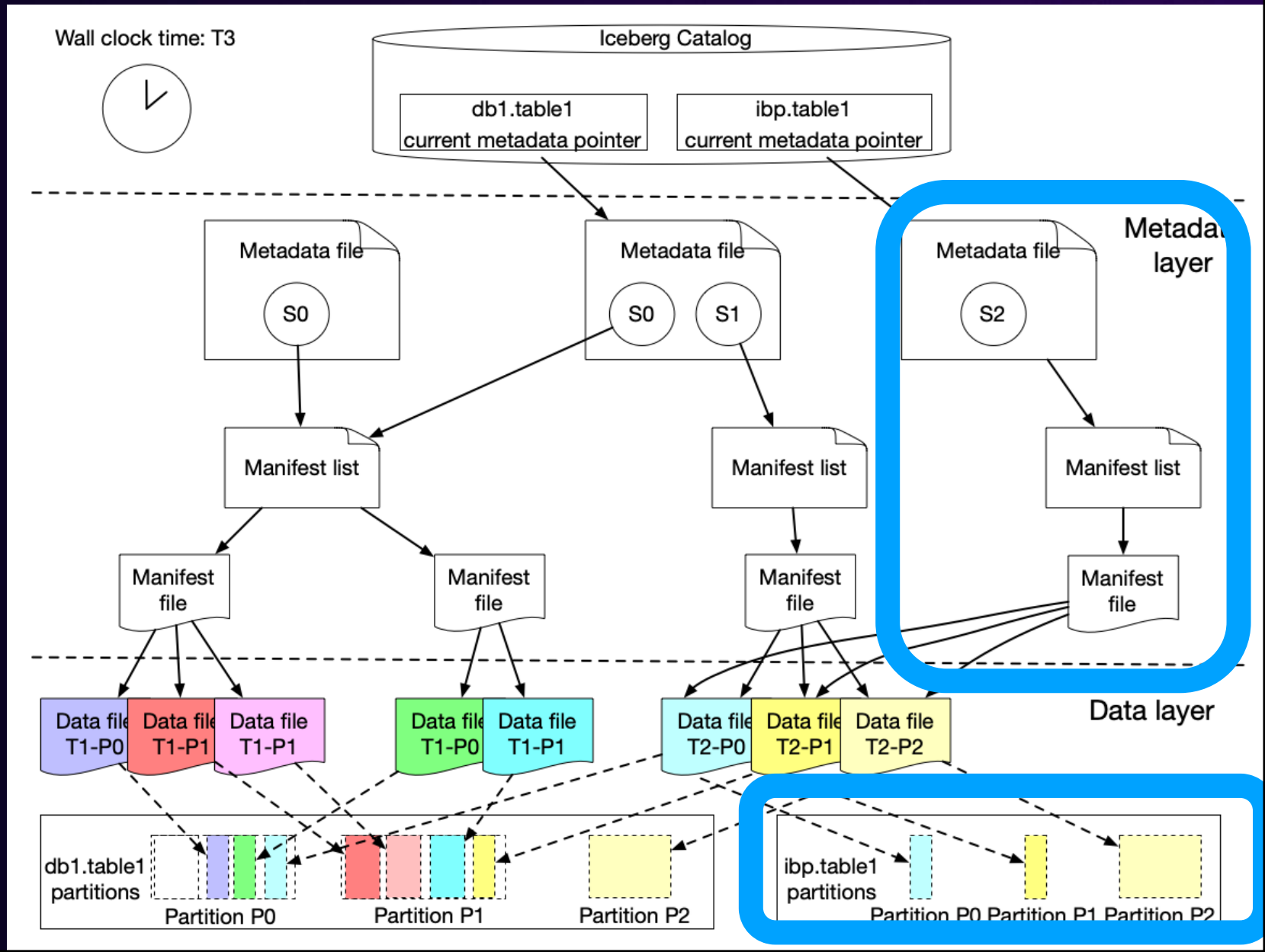


Incremental change capturing

Empty table with the same schema

Add data file references to the change table

- Zero data copy
- No data access
- Same partition layout



Alternatives

- Apache Iceberg + Flink
- Apache Iceberg + Spark Procedure (create_changelog_view)
- Apache Iceberg + Spark Structure Streaming

Show me the code

```
Table source = catalog.loadTable(sourceTableIdentifier);
List<Snapshot> snapshots = StreamSupport.stream(
      SnapshotUtil.ancestorsBetween(source, toSnapshotIdInclusive, fromSnapshotIdExclusive).spliterator(), false)
   .filter(snapshot -> snapshot.operation().equals(DataOperations.APPEND))
   .collect(Collectors.toList());
```

Get a list of append snapshots

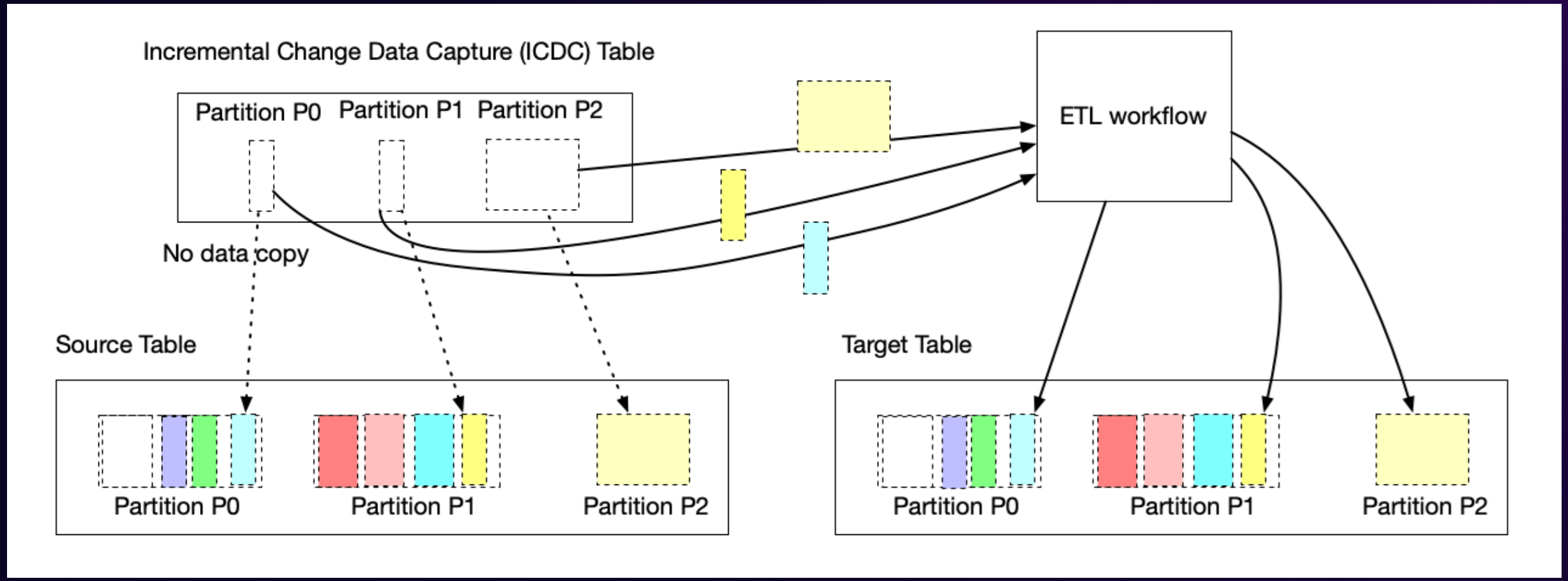
```
Table icdcTable = createChangeTable(catalog, source, icdcTableName));
AppendFiles af = icdcTable.newAppend();
for (Snapshot snapshot : snapshots) {
  for (DataFile df : snapshot.addedDataFiles(source.io())) {
    af.appendFile(df);
  }
}
af.commit();
```

Add data files to the ICDC table

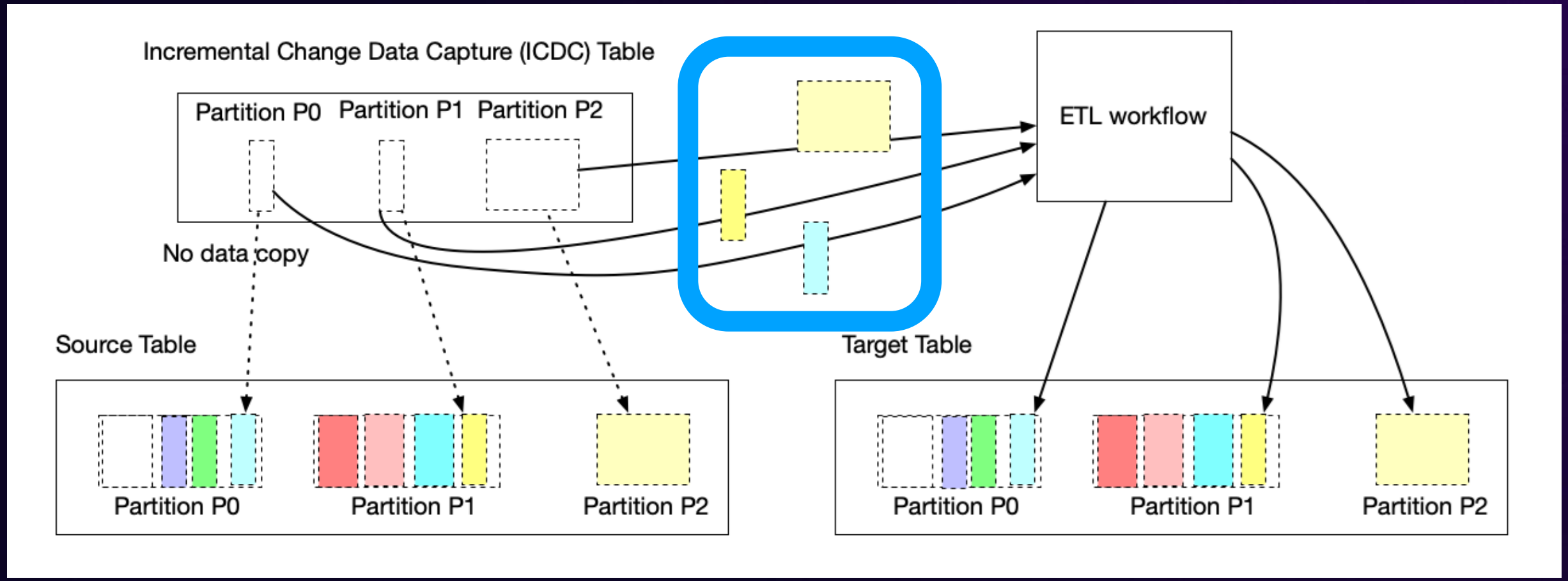
Emerging patterns

- Incrementally process the captured incremental change data and directly append it to the target table
- Use captured incremental change data as the row level filter list to remove unnecessary transformation
- Use the captured range parameters in the business logic

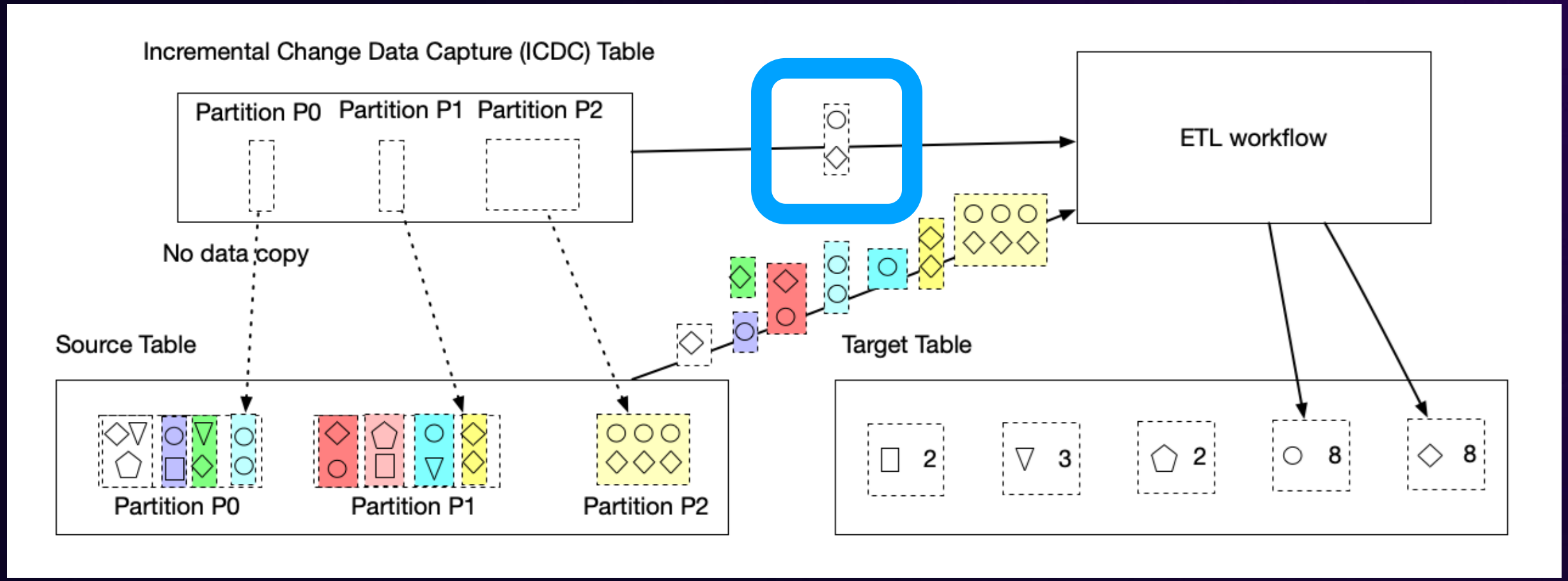
Pattern 1: Incrementally append captured change data



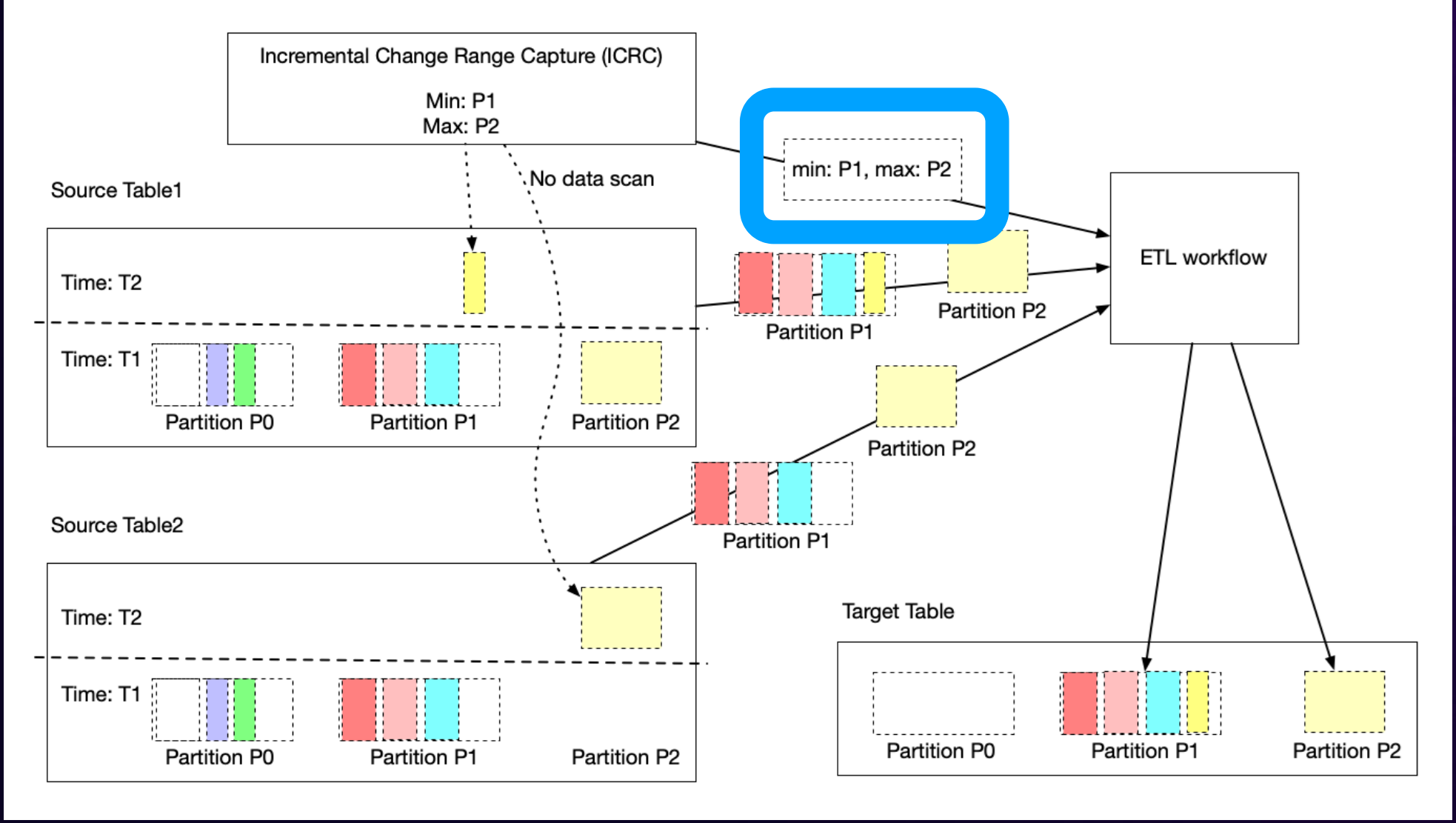
Pattern 1: Incrementally append captured change data



Pattern 2: Use captured change data as row level filters



Pattern 3: Use the captured range parameters



Onboarding concerns

Compatibility with existing workflows

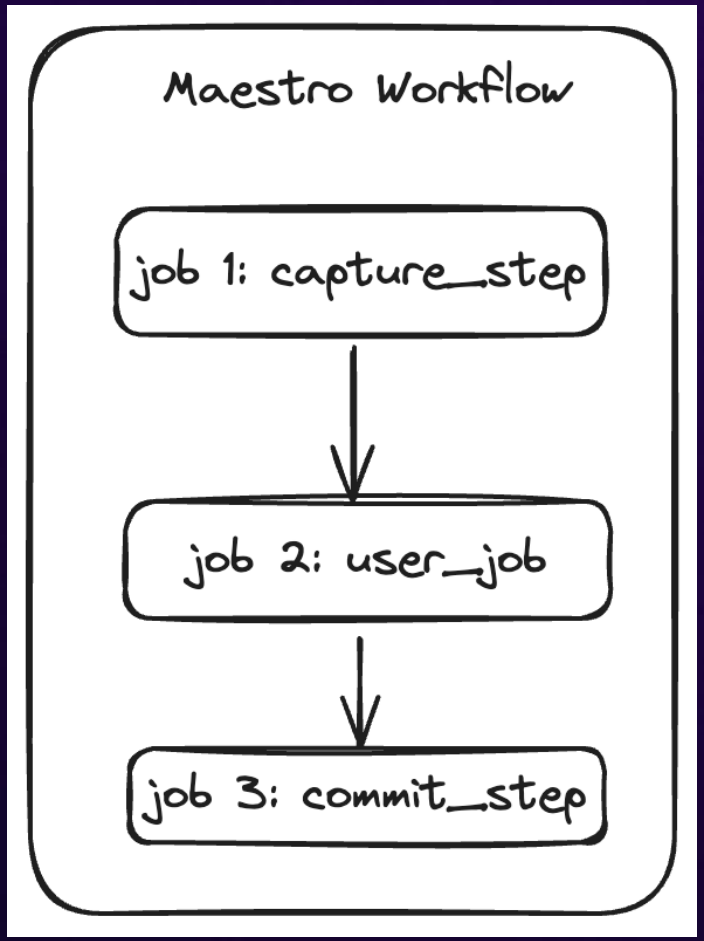
Extra costs

- Development cost
- Operational cost
- Maintenance cost

Maestro incremental processing solution (IPS)

Interfaces

- Table(s)
  - Table name(s) passed to user business logic over Maestro parameters
- IpCapture step type
  - Capture changes from last checkpoint
- IpCommit step type
  - Commit checkpoint for this run

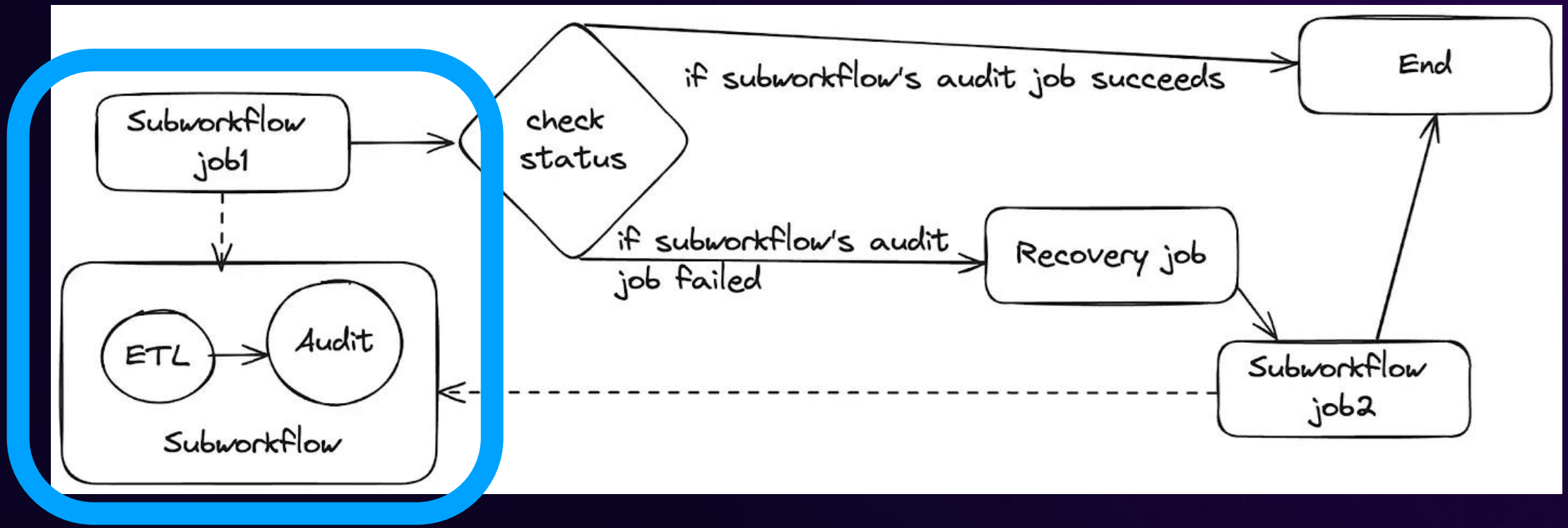


Best user experiences

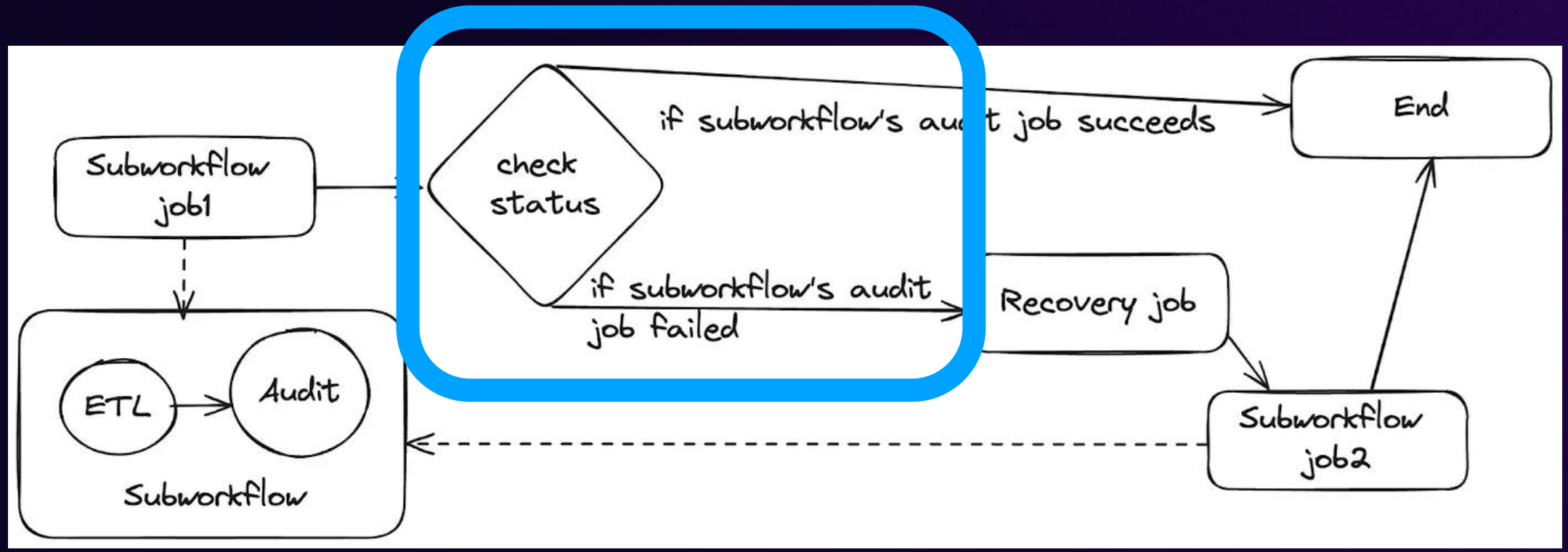
Users love to have freedom to use their own approaches to solve their problems

- Fully compatible with existing workflows
- Don’t ask to use specific languages
- Don’t ask to choose a specific compute engine
- Don’t ask to rewrite the business logic
- Don’t ask to re-architecture the existing data pipelines
- Low code solution

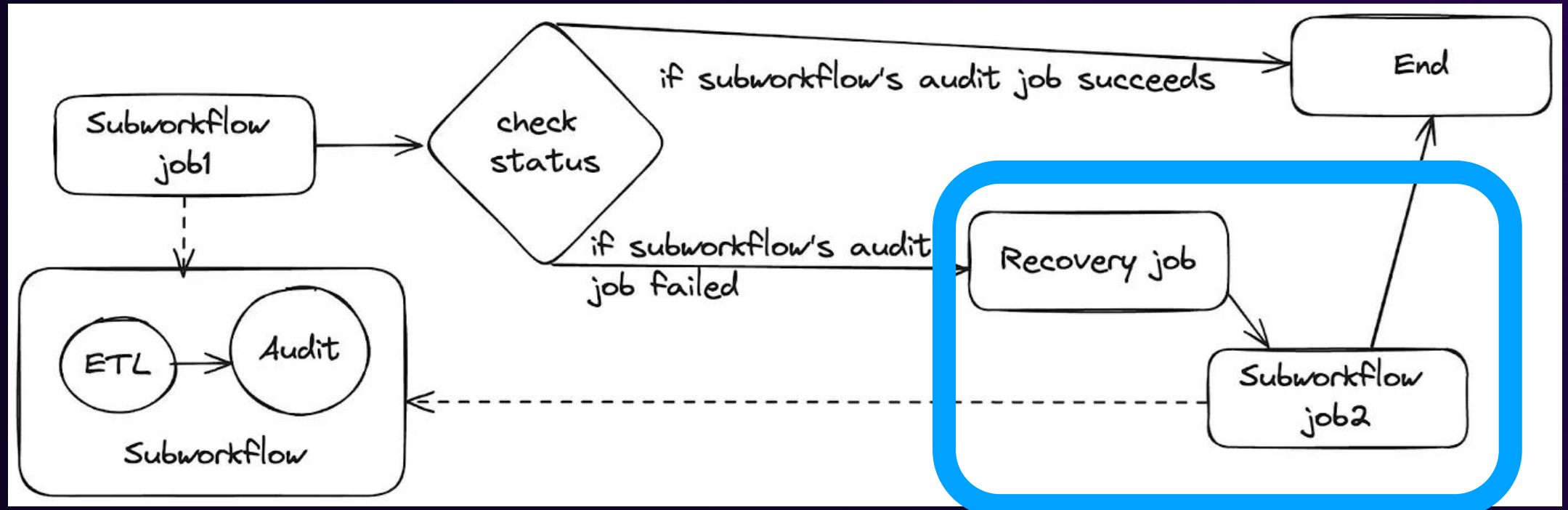
Auto remediation workflow



Auto remediation workflow



Auto remediation workflow



Auto remediation workflow

Workflow definition

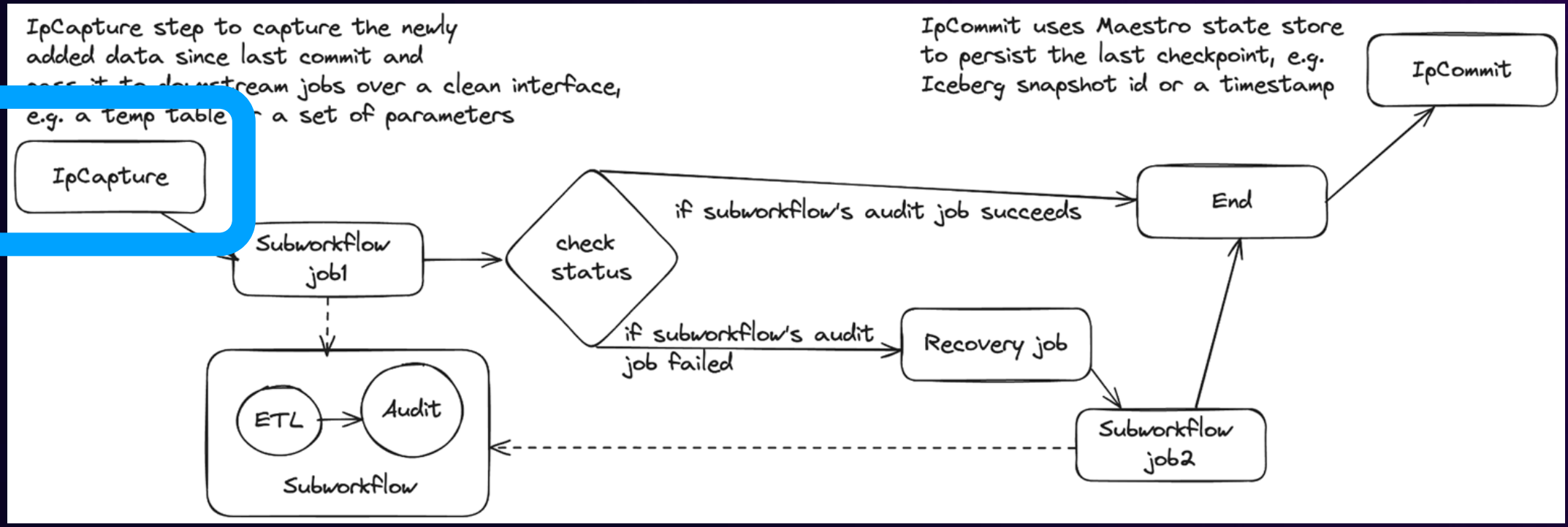
```
Workflow:
 id: auto.remediation.pipeline
 jobs:
    - subworkflow:
        id: normal_workflow
        workflow_id: demo.pipeline
        failure_mode: IGNORE_FAILURE
    - job:
        id: check_status
        type: NoOp
    - subworkflow:
        id: recovery
        workflow_id: example.recovery.wf
    - subworkflow:
        id: re_run
        workflow_id: demo.pipeline
    - job:
        id: end
        type: NoOp
 dag:
    - normal_workflow -> check_status
    - check_status:
        IF params.getFromStep('normal_workflow',
'MAESTRO_STEP_STATUS') ==
'COMPLETED_WITH_ERROR': recovery -> re_run ->
end
        OTHERWISE: end
```

Auto remediation workflow

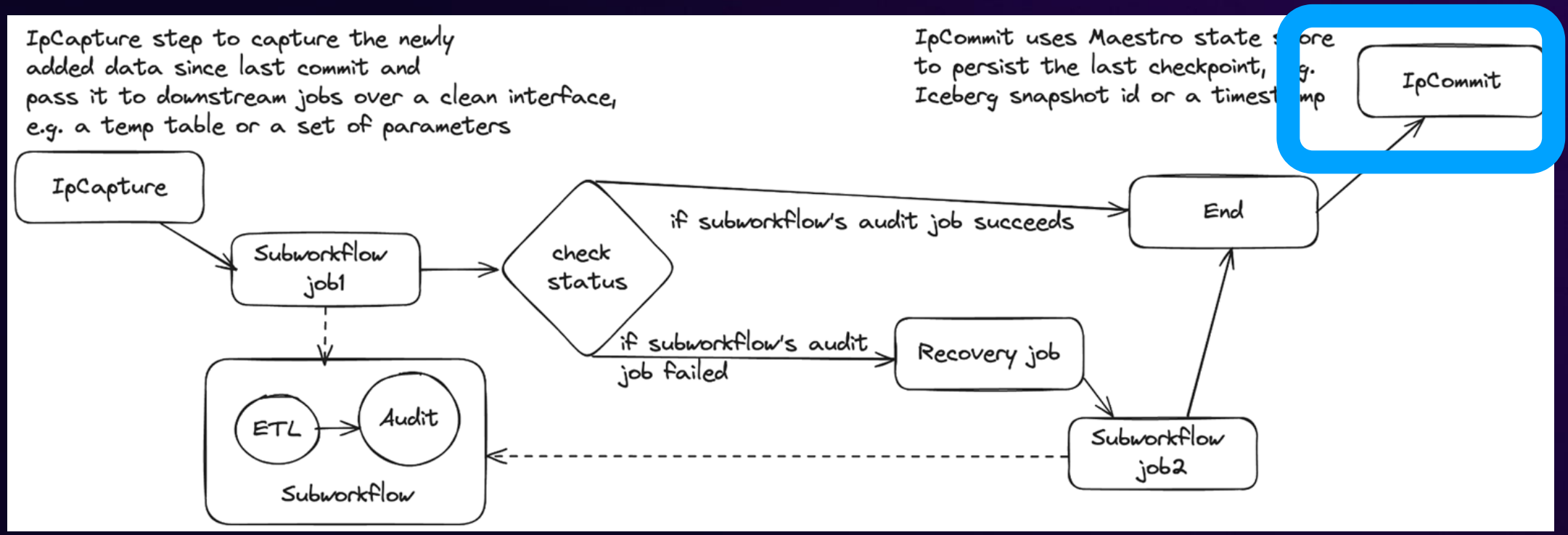
Workflow definition

```
Workflow:
 id: auto.remediation.pipeline
 jobs:
   - subworkflow:
       id: normal_workflow
       workflow_id: demo.pipeline
       failure_mode: IGNORE_FAILURE
   - job:
       id: check_status
       type: NoOp
   - subworkflow:
       id: recovery
       workflow_id: example.recovery.wf
   - subworkflow:
       id: re_run
       workflow_id: demo.pipeline
   - job:
       id: end
 dag:
   - normal_workflow -> check_status
   - check_status:
       IF params.getFromStep('normal_workflow',
'MAESTRO_STEP_STATUS') ==
'COMPLETED_WITH_ERROR': recovery -> re_run ->
end
       OTHERWISE: end
```

Auto remediation workflow with IPS



Auto remediation workflow with IPS



Auto remediation workflow with IPS

Workflow definition

```
Description: |
  This is a sample description to [a Maestro
workflow](https://github.com/Netflix/maestro)
Trigger:
 cron: '@daily'
Workflow:
 id: incremental.processing.pipeline
 jobs:
   - job:
       id: capture_change
       type: IpCapture
       ip_capture:
         source_tables:
           - membership_table
         capture_mode: ICDC
         data_operations:
           append
   - subworkflow:
       id: auto.remediation.pipeline
       workflow_id: auto.remediation.pipeline
       SOURCE_TABLE: icdc.membership_table.01
       my_query: $S3{my_script.sql}
   - job:
       id: commit_checkpoint
       type: IpCommit
       ip_commit:
         capture_step: capture_change
dag: sequential
```

Auto remediation workflow with IPS

Workflow definition

```
Description: |
  This is a sample description to [a Maestro
workflow](https://github.com/Netflix/maestro)
Trigger:
 cron: '@daily'
Workflow:
 id: incremental.processing.pipeline
 jobs:
   - job:
       id: capture_change
       type: IpCapture
       ip_capture:
         source_tables:
           - membership_table
         capture_mode: ICDC
         data_operations:
           - append
   - subworkflow:
       id: auto.remediation.pipeline
       workflow_id: auto.remediation.pipeline
       SOURCE_TABLE: icdc.membership_table.01
       my_query: $S3{my_script.sql}
   - job:
       id: commit_checkpoint
       type: IpCommit
       ip_commit:
         capture_step: capture_change
 dag: sequential
```

Auto remediation workflow with IPS

Workflow definition

```
Description: |
  This is a sample description to [a Maestro
workflow](https://github.com/Netflix/maestro)
Trigger:
 cron: '@daily'
Workflow:
 id: incremental.processing.pipeline
 jobs:
   - job:
       id: capture_change
       type: IpCapture
       ip_capture:
         source_tables:
           - membership_table
         capture_mode: ICDC
         data_operations:
           - append
   - subworkflow:
       id: auto.remediation.pipeline
       workflow_id: auto.remediation.pipeline
       SOURCE_TABLE: icdc.membership_table.01
       my_query: $S3{my_script.sql}
   - job:
       id: commit_checkpoint
       type: IpCommit
       ip_commit:
         capture_step: capture_change
```

IPS highlights

Efficient incremental change capturing

Handle late arriving data

Language and engine agnostic

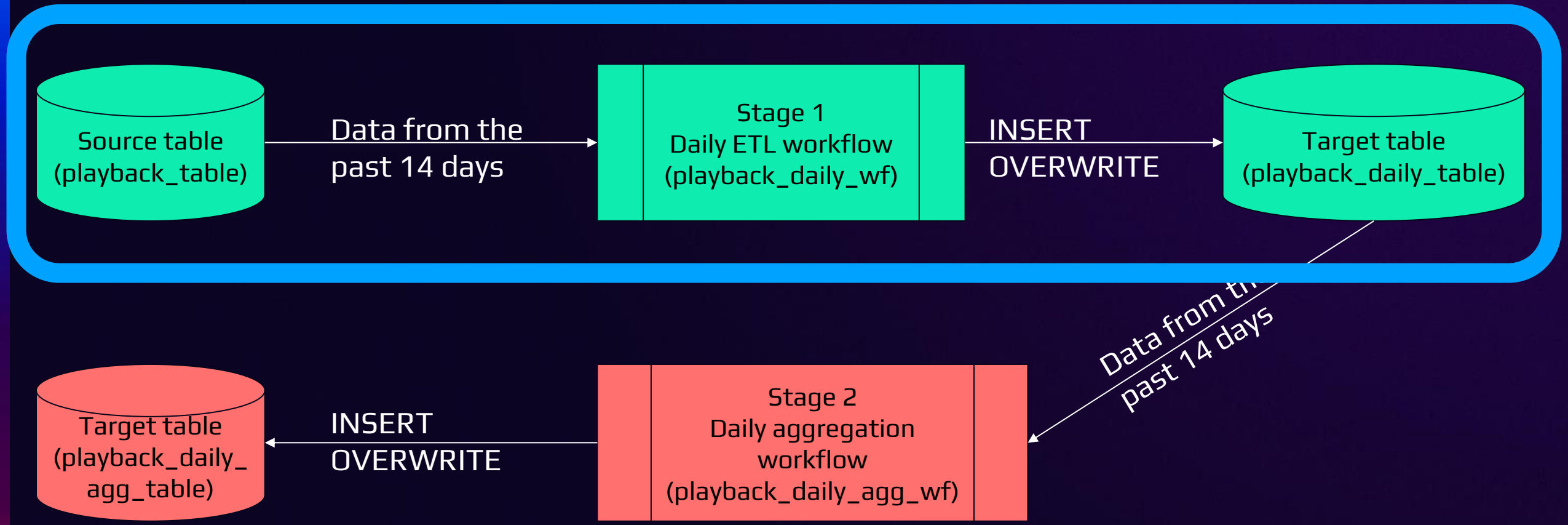
Clean interface

Compatible with existing workflows

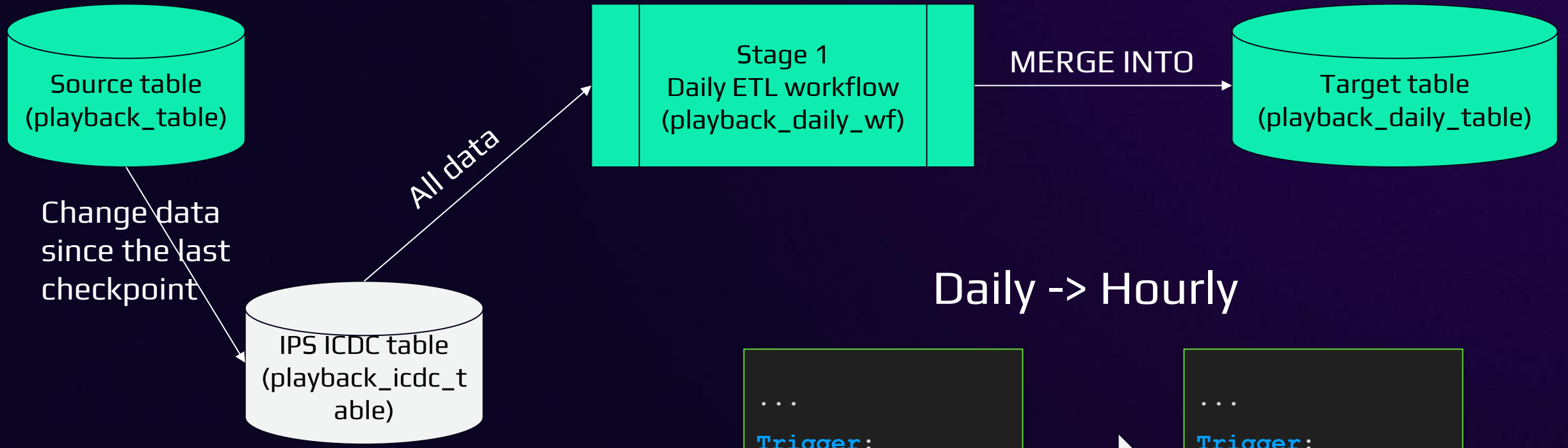
Low onboarding cost

Use cases and examples

Two-stage pipeline example



Stage 1 workflow with IPS



Daily -> Hourly

```
...
Trigger:
 cron: '@daily'
...
```

```
...
Trigger:
 cron: '@hourly'
...
```

Changes of write job on stage 1

Original SQL query in playback_daily_workflow:

```
INSERT OVERWRITE
TABLE playback_daily_table
SELECT
        ${business_logic}
FROM
        playback_table
```

New SQL query in ips_playback_daily_workflow:

```
MERGE INTO
TABLE playback_daily_table
SELECT
        ${business_logic}
FROM
        playback_icdc_table
WHEN MATCHED
        ${deduplication_logic}
```

Changes of write job on stage 1

Original SQL query in playback_daily_workflow:

```
Unset
INSERT OVERWRITE
        TABLE playback_daily_table
SELECT
        ${business_logic}
FROM
        playback_table
```

New SQL query in ips_playback_daily_workflow:

```
Unset
MERGE INTO
        TABLE playback_daily_table
SELECT
        ${business_logic}
FROM
        playback_icdc_table

        ${deduplication_logic}
```

Changes of write job on stage 1

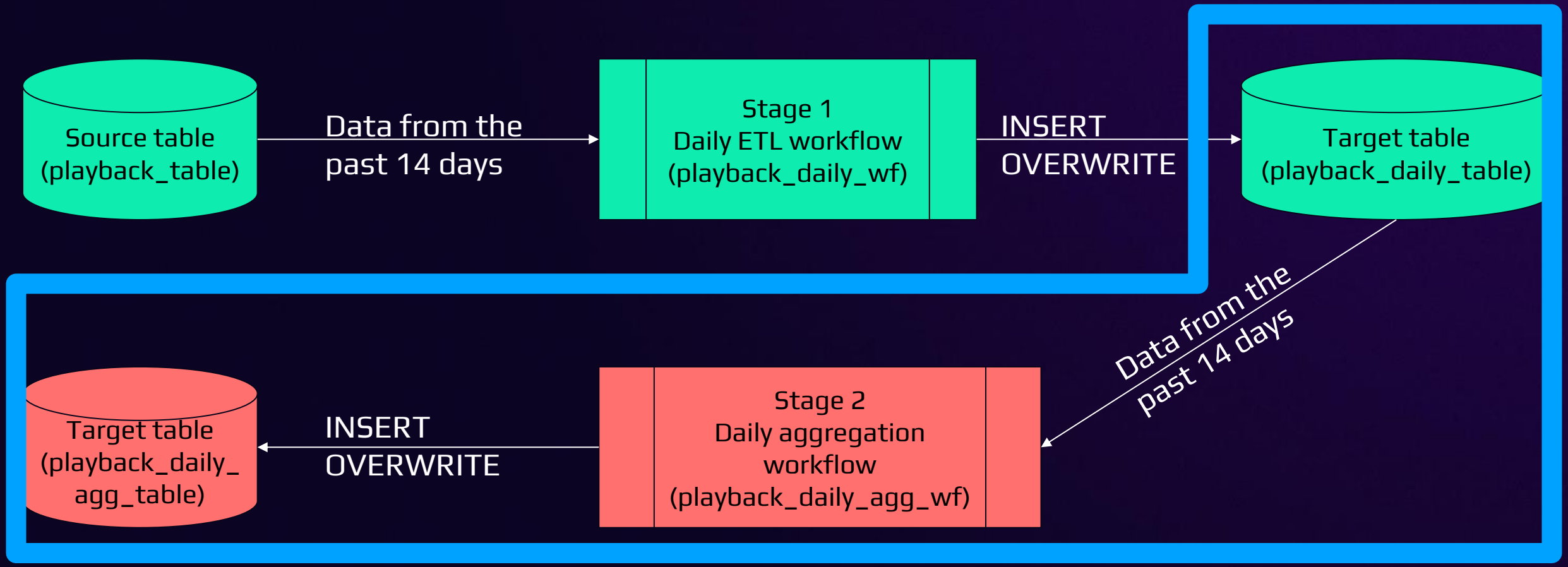
Original SQL query in playback_daily_workflow:

```
Unset
INSERT OVERWRITE
        TABLE playback_daily_table
SELECT
        ${business_logic}
FROM
        playback_table
```

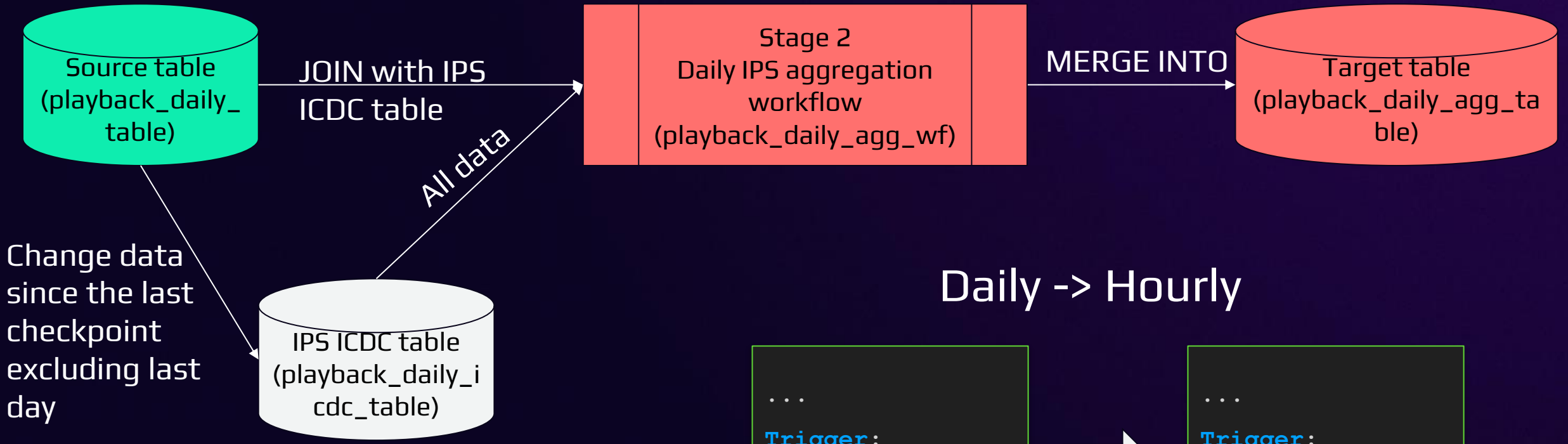
New SQL query in ips_playback_daily_workflow:

```
Unset
MERGE INTO
        TABLE playback_daily_table
SELECT
        ${business_logic}
FROM
        [illegible]
WHEN MATCHED
        ${deduplication_logic}
```

Two-stage pipeline example



Stage 2 workflow with IPS



Daily -> Hourly

```
...
Trigger:
 cron: '@daily'
...
```

```
...
Trigger:
 cron: '@hourly'
...
```

Changes of write job on stage 2

Original SQL query:

```
INSERT OVERWRITE
SELECT
        ${business_logic}
FROM
        playback_daily_table
```

New SQL query:

```
MERGE INTO
SELECT
        ${business_logic}
FROM
        playback_daily_table
JOIN
        playback_daily_icdc_table
ON
        ${aggregation_group_by_keys}
WHEN MATCHED
        ${deduplicate_logic}
```

Changes of write job on stage 2

Original SQL query:

```
Unset
INSERT OVERWRITE
       TABLE playback_daily_agg_table
SELECT
       ${business_logic}
FROM
       playback_daily_table
```

New SQL query:

```
Unset
MERGE INTO
       TABLE playback_daily_agg_table
SELECT
       ${business_logic}
FROM
       playback_daily_table
JOIN
       playback_daily_icdc_table
ON
       ${aggregation_group_by_keys}
       ${deduplicate_logic}
```

Changes of write job on stage 2

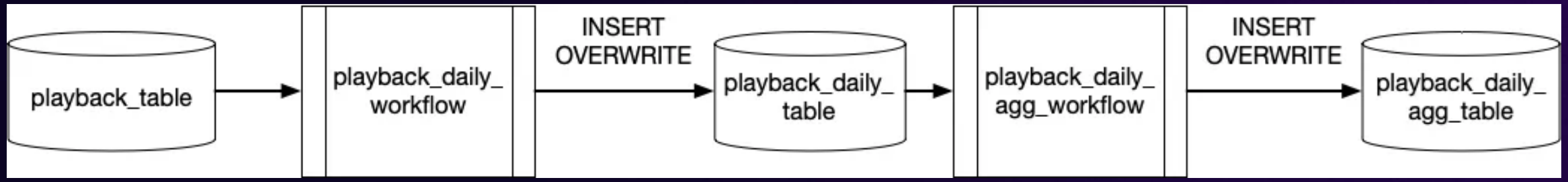
Original SQL query:

```
Unset
INSERT OVERWRITE
       TABLE playback_daily_agg_table
SELECT
       ${business_logic}
FROM
       playback_daily_table
```

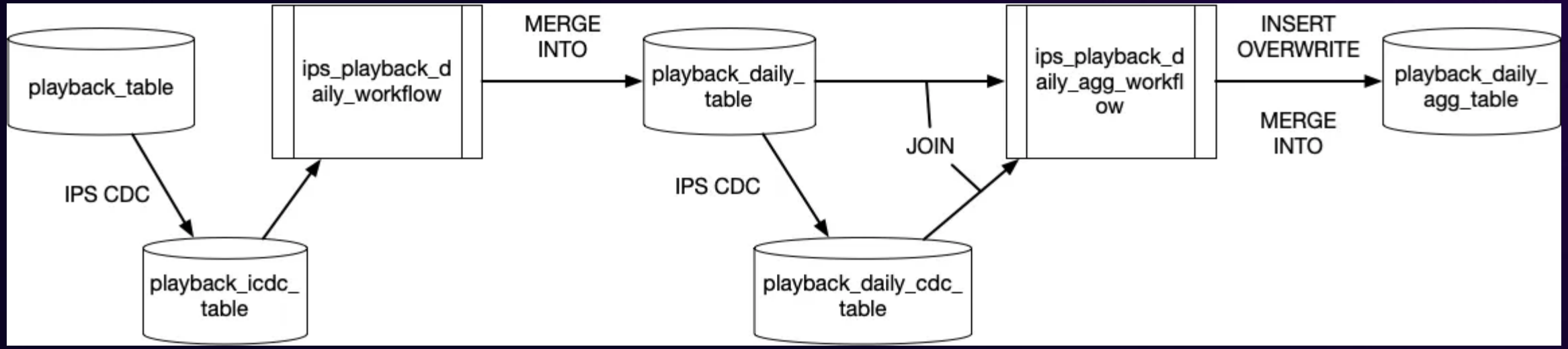
New SQL query:

```
Unset
MERGE INTO
       TABLE playback_daily_agg_table
SELECT
       ${business_logic}
FROM
       playback_daily_table
JOIN
       playback_daily_icdc_table
ON
       [illegible]
WHEN MATCHED
       ${deduplicate_logic}
```

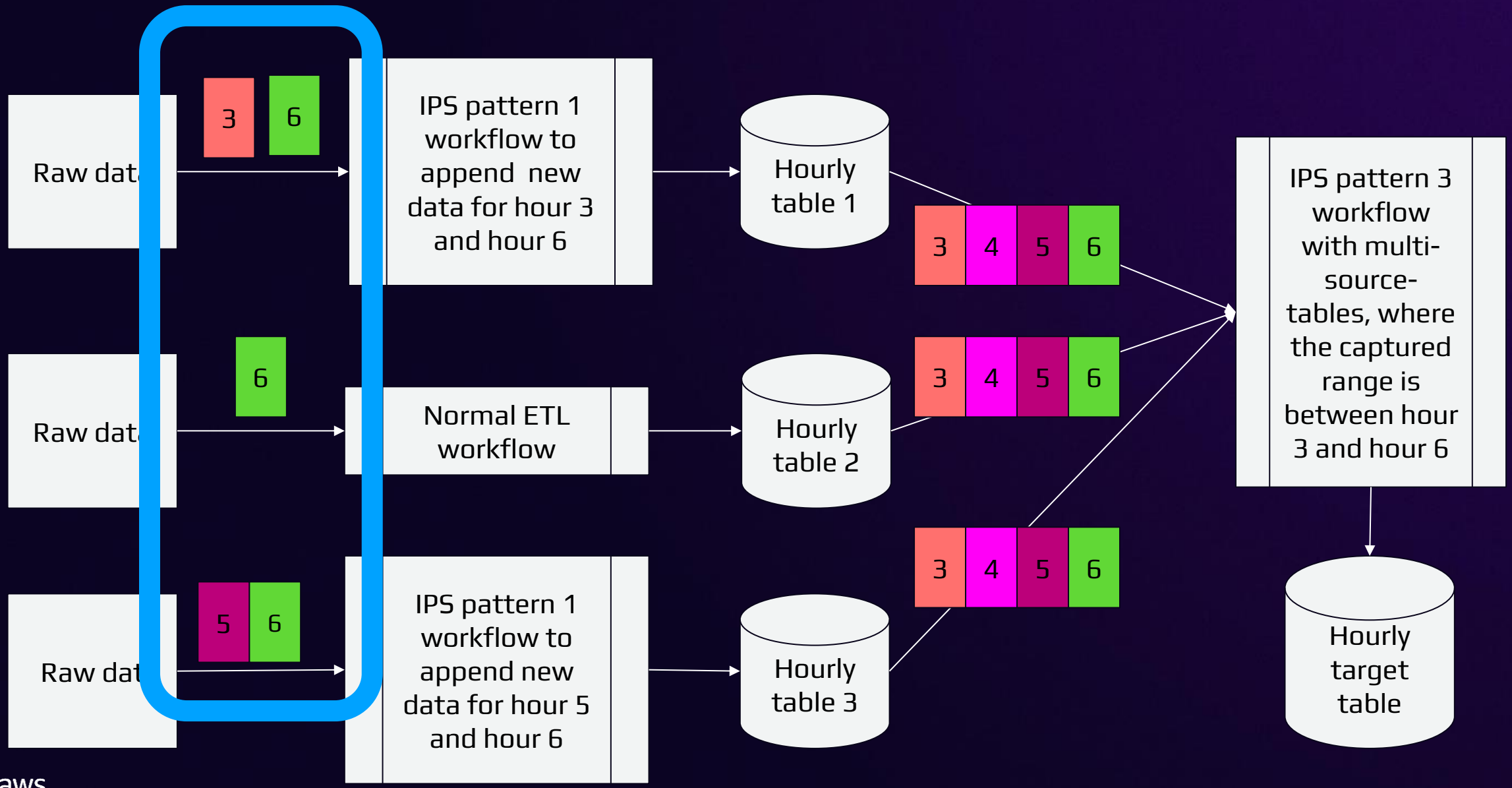
Multi-stage workflow with IPS



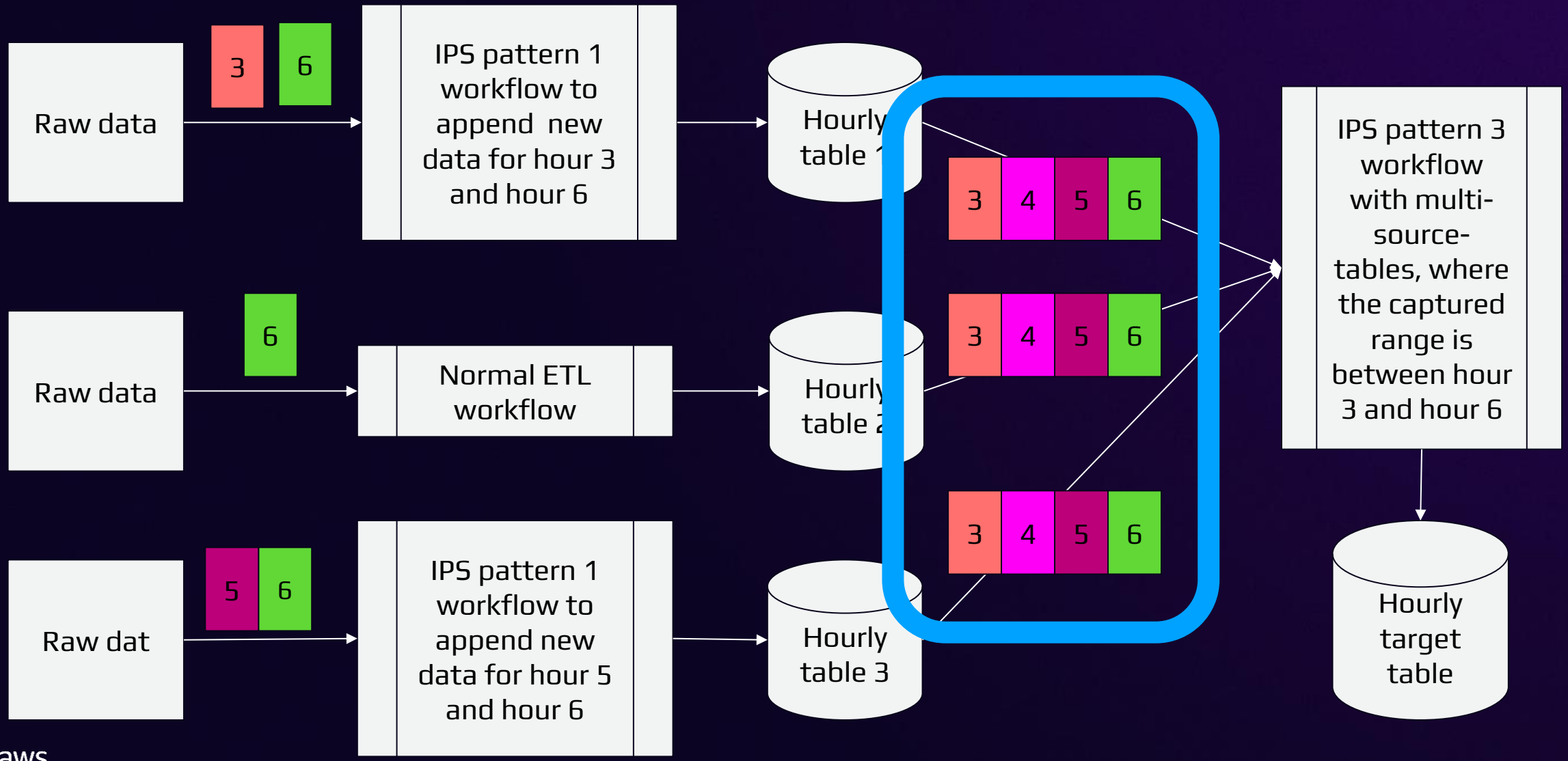
Only 10% of original cost!



Multi-table use case



Multi-table use case



Takeaways and future improvements

Key takeaways

Efficient capturing: Iceberg metadata enables incremental processing without accessing data

Simplified adoption: Decoupling change capture reduces complexity

Improved experience: Clean interfaces improve usability

New patterns: IPS patterns apply to many scenarios

Future improvements

Move from Table to View to reduce maintenance

Support other types of snapshots with the new version of Iceberg table specs

Work with Iceberg community to share the approach in Iceberg cookbooks

Multi-stage data backfill support

Acknowledgment

Drew Goya

Andy Chu

Charles Smith

Eva Tse

Jun He

Kyoko Shimada

Netflix team

Ely Spears

Yingyi Zhang

Anjali Norwood

Zhuoran Dong

Brittany Truong

Prashanth Ramdas

Abhinaya Shetty

Bharath Mummadisetty

Thank you!

Please complete the session survey in the mobile app

Jun He

Staff Software Engineer
Netflix
linkedin.com/in/jheua

Vaidy Krishnan

Strategic Account Manager
AWS