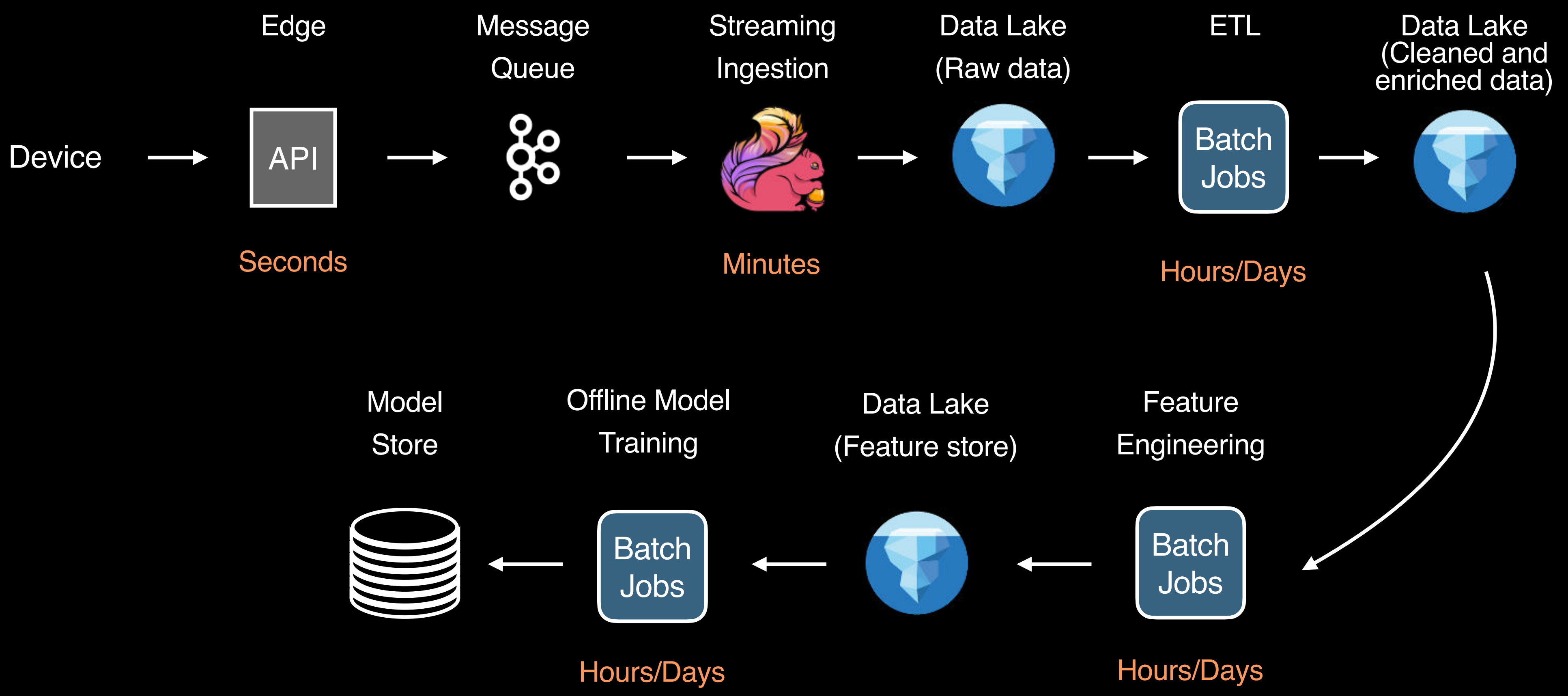
Streaming from Iceberg Data Lake

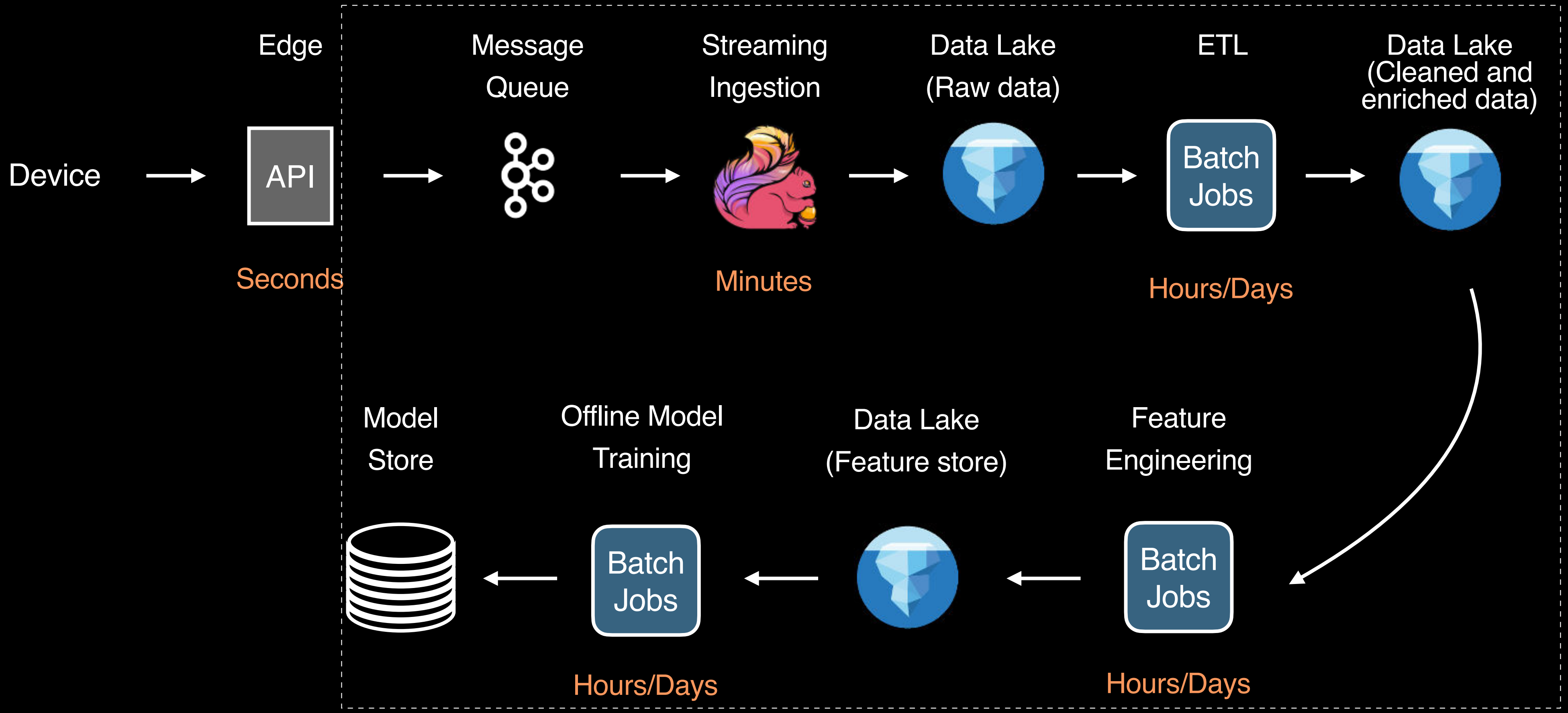
Steven Wu I Apple

THIS IS NOT A CONTRIBUTION

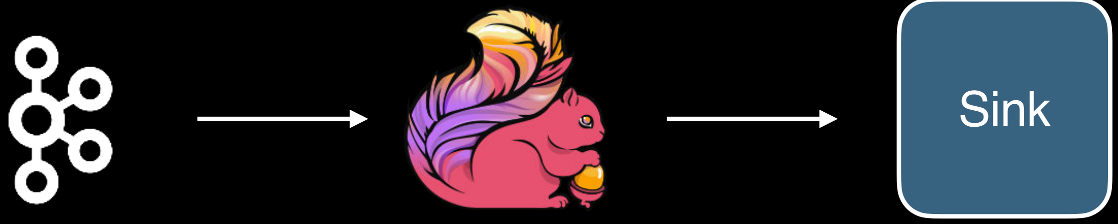
Traditional data pipelines are largely chained by batch jobs reading from data lake



Overall latency is hours to days



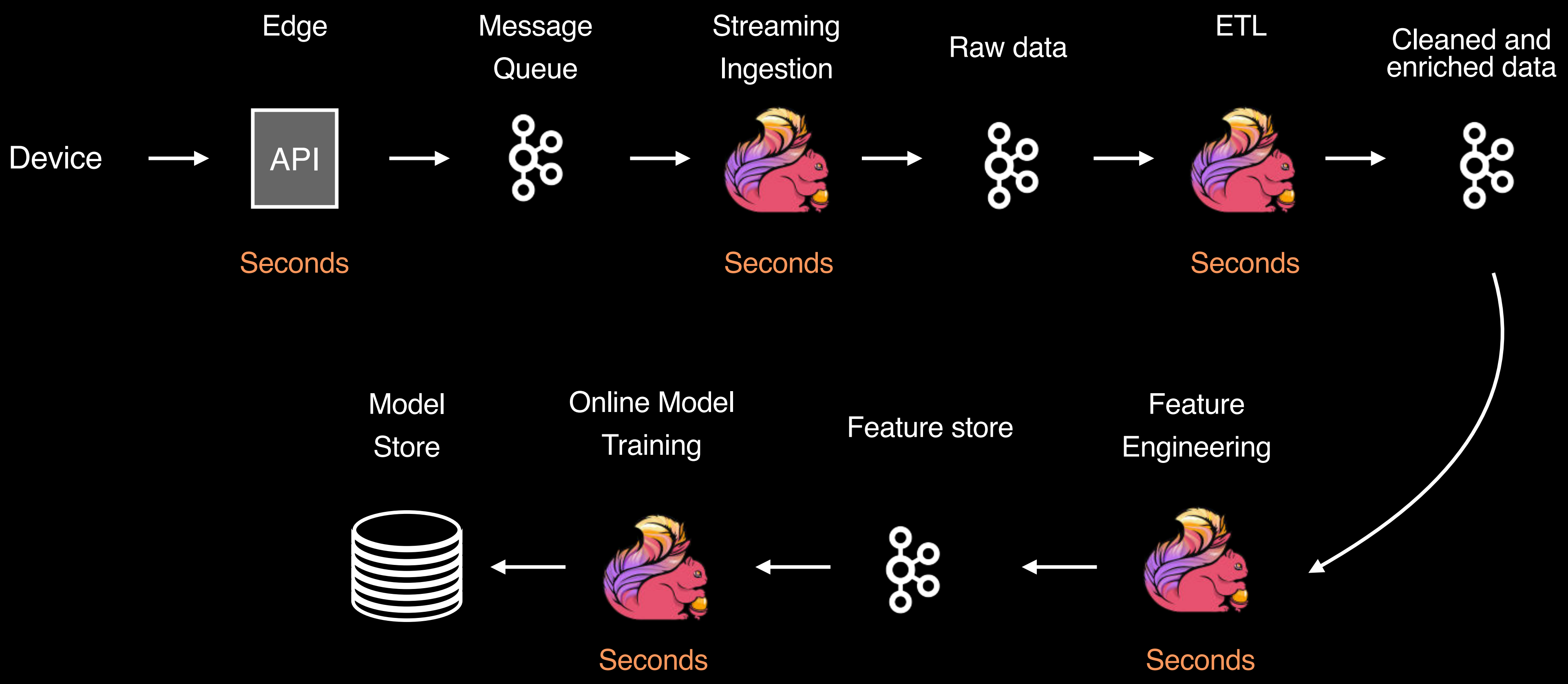
Flink streaming from Kafka is very popular



Sink

Flink Streaming Job

Switch everything to Flink streaming from Kafka



Kafka can achieve
sub-second read latency

But there are tradeoffs . . .

Operation is not easy

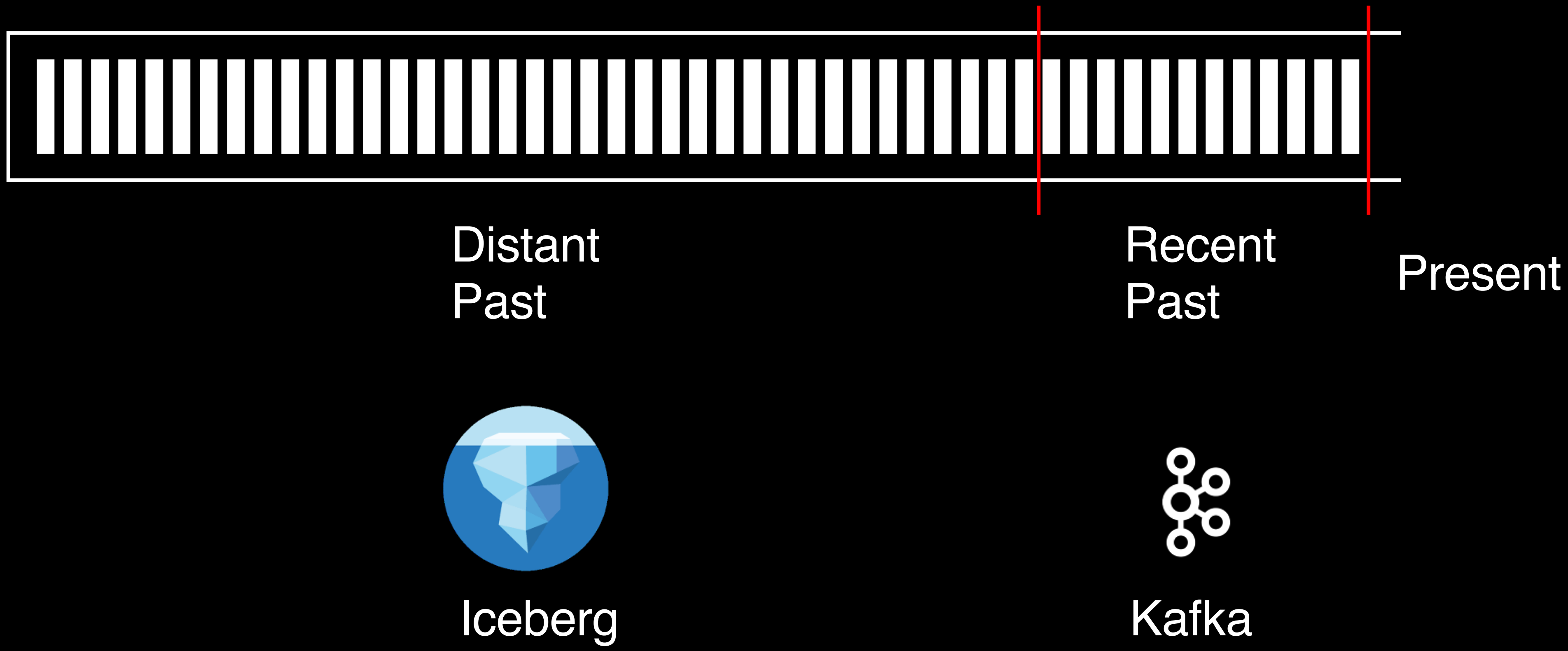
- Upgrading stateful system is painful
- Capacity planning
- Bursty workload and isolation
- Managed Kafka service in cloud can be more expensive

It is very expensive to store long-term data in Kafka

38x

Steven Wu & Sundaram Ananthanarayanan. Backfilling from Flink pipelines at frac. cost using Iceberg. Apache Flink Meetup Hosted by Netflix. Jan 20, 2021

Here comes tiered storage



Distant Past

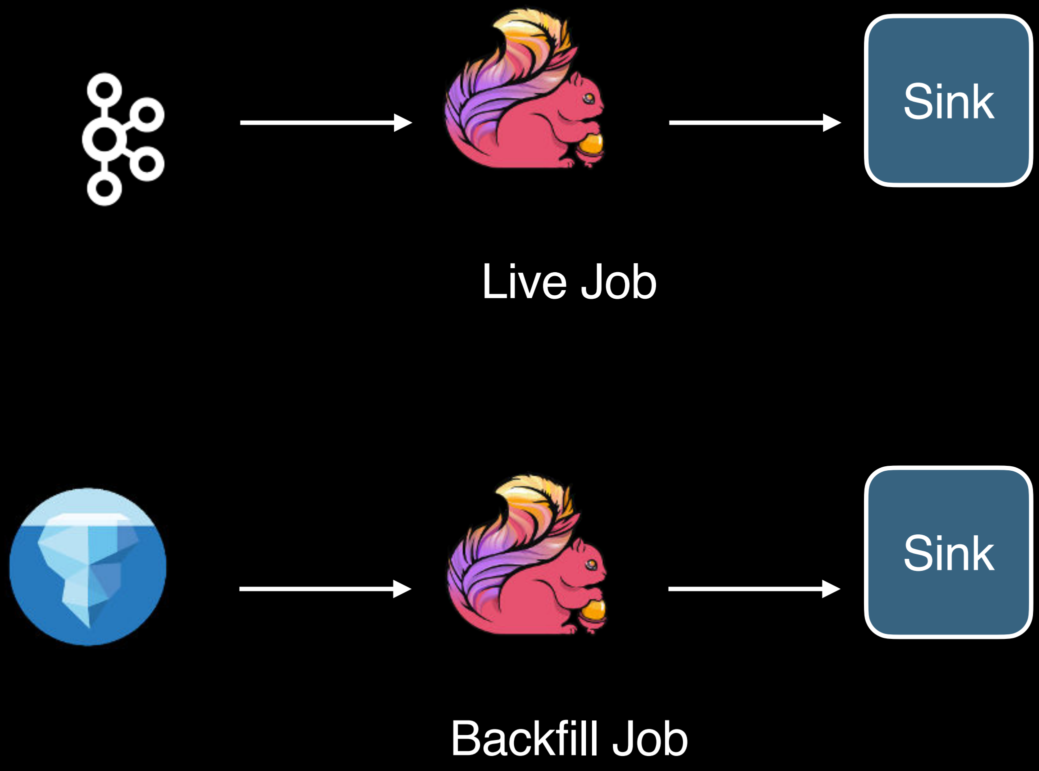
Recent Past

Present

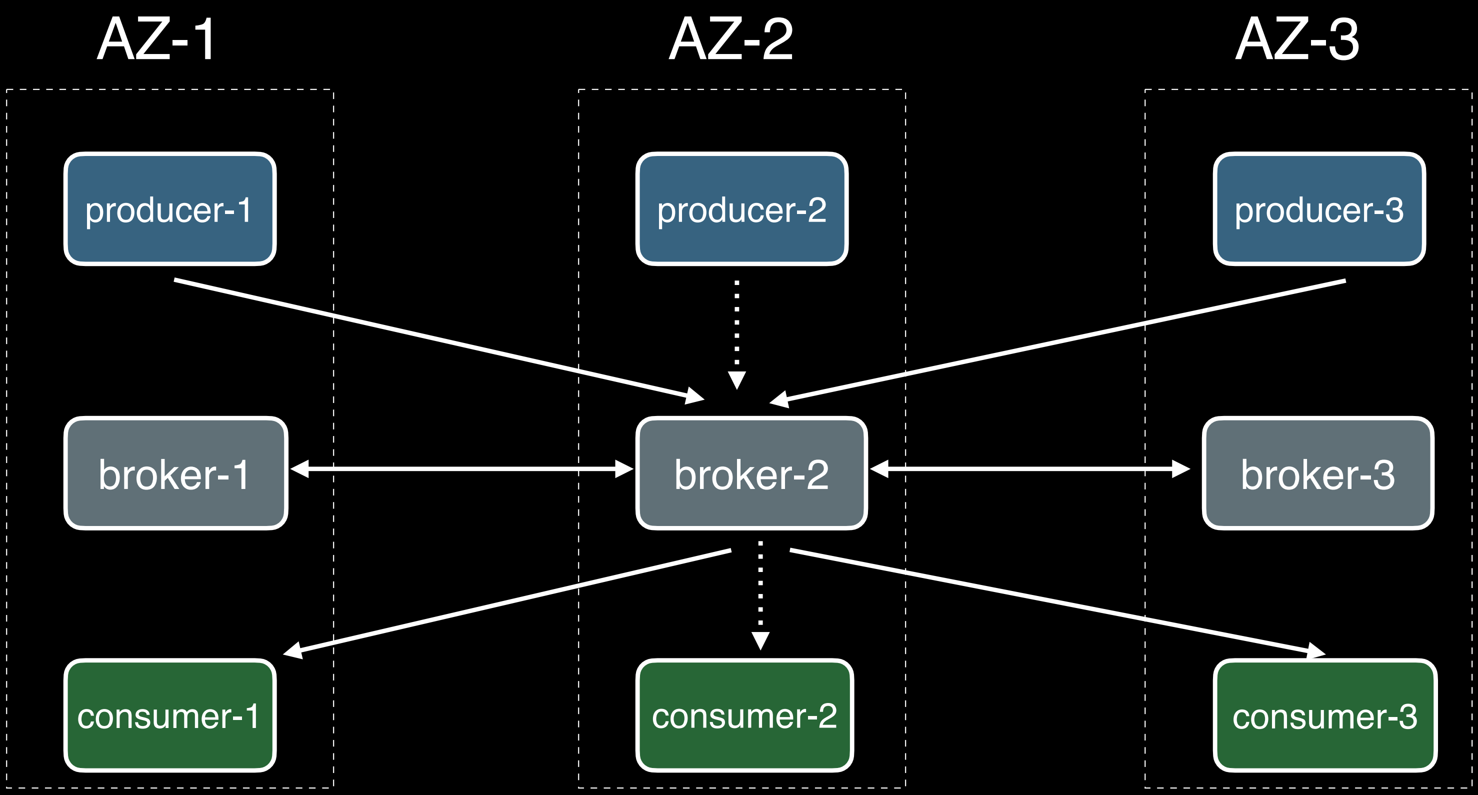
Iceberg

Kafka

Backfill jobs read data from Iceberg long-term storage

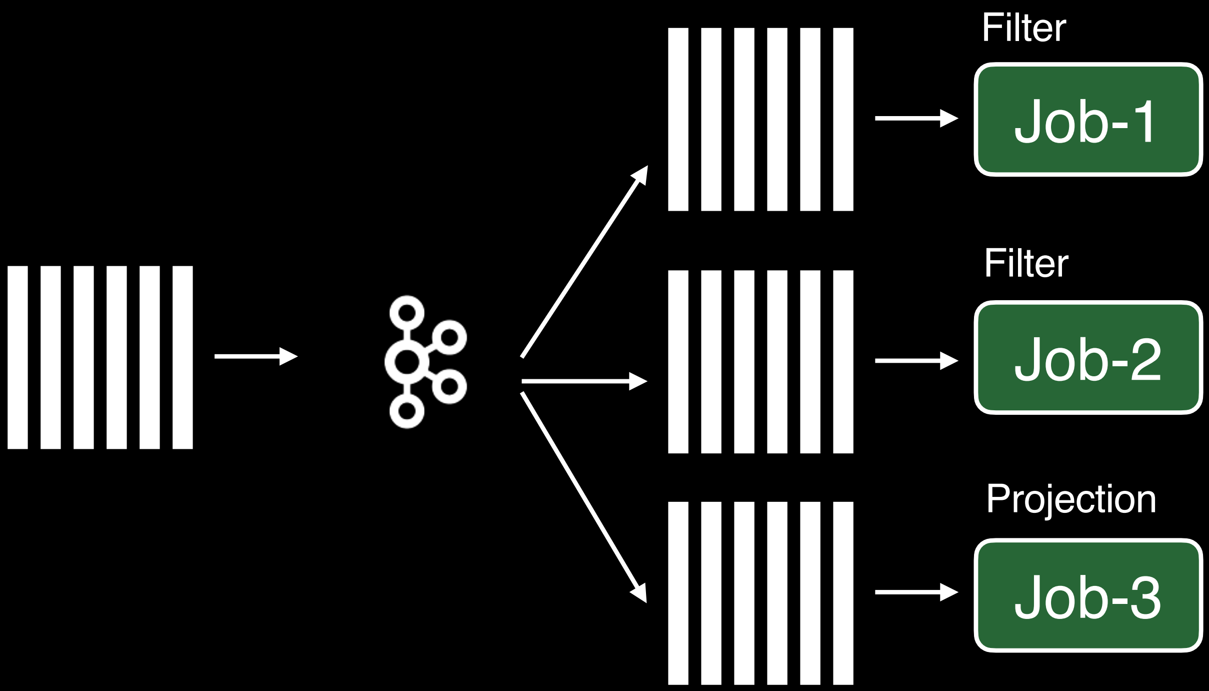


Cross-AZ network cost can be much higher than compute and storage cost for brokers

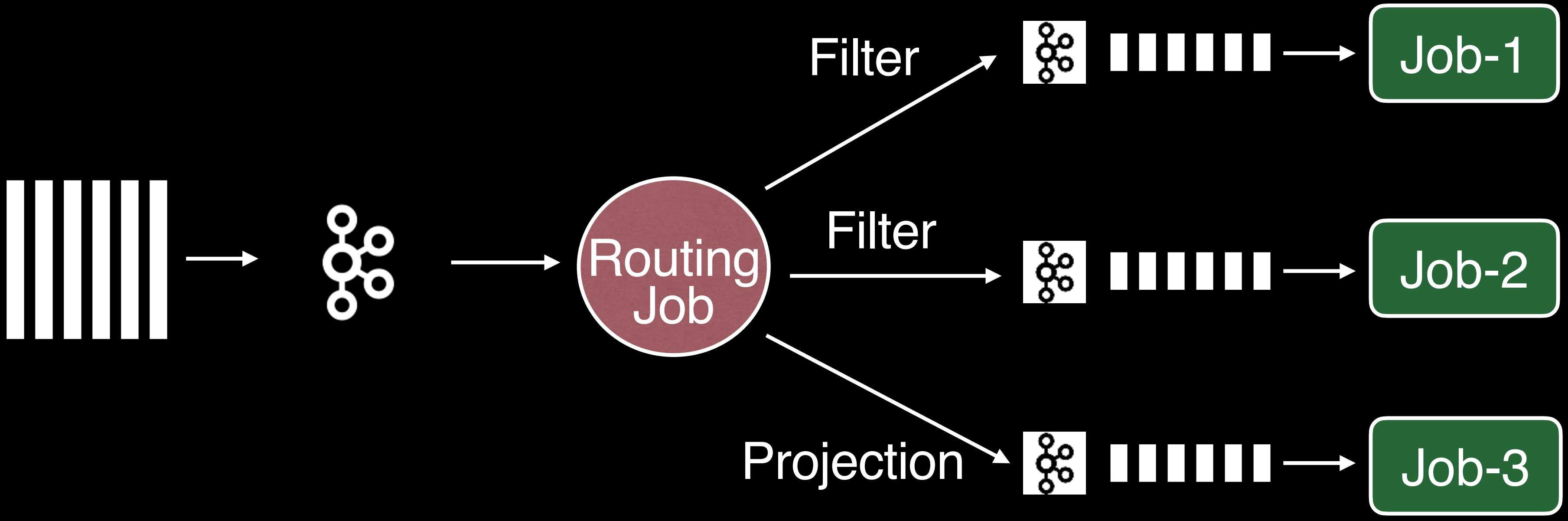


>10x

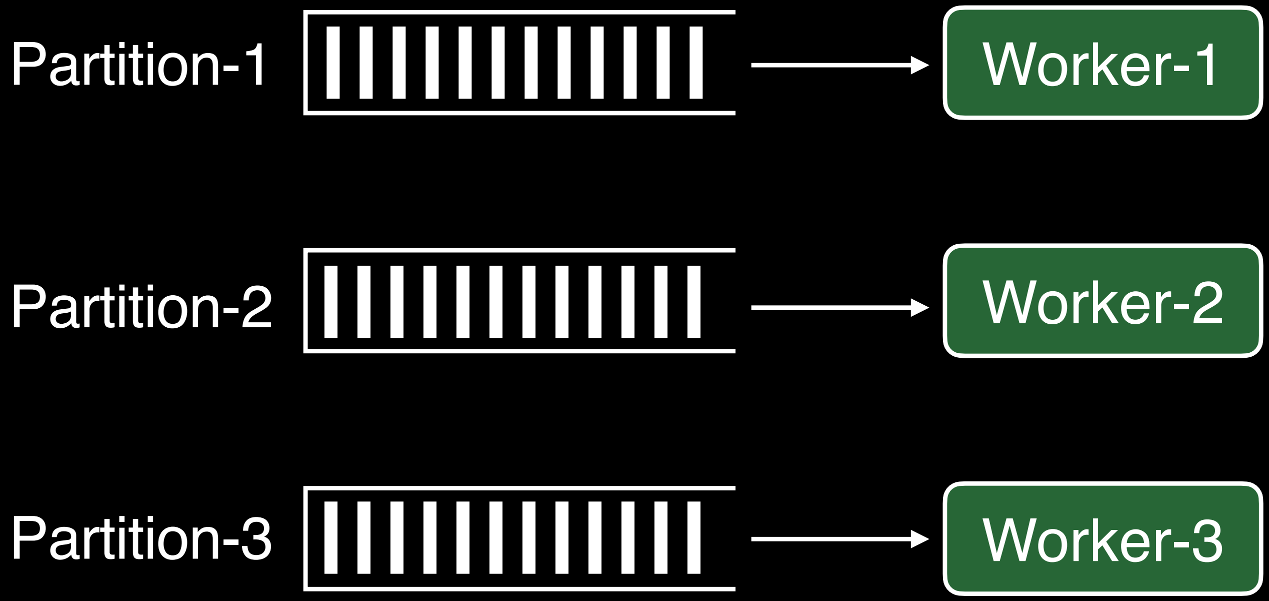
Kafka source doesn't support filtering or projection at broker side



Set up routing jobs just to filter or project data



Kafka source statically assigns partitions during



Partition-1

Worker-1

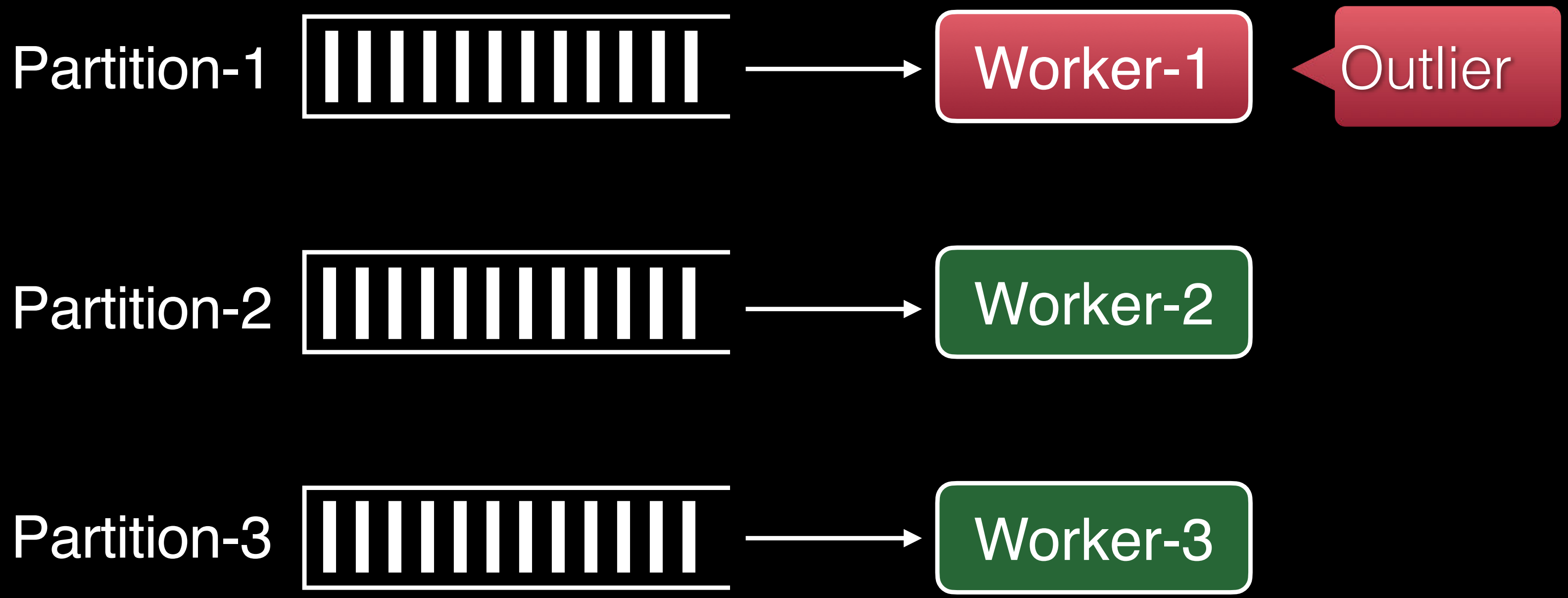
Partition-2

Worker-2

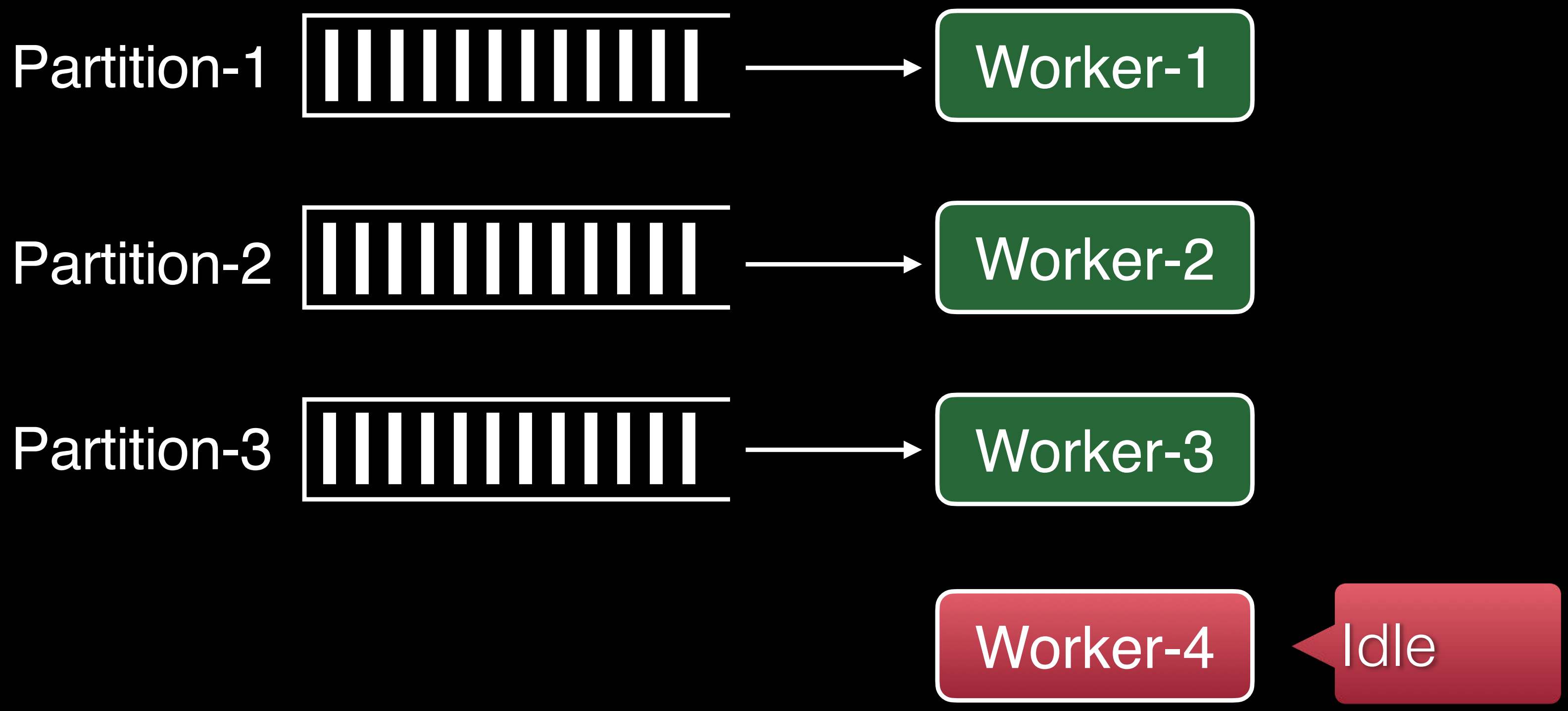
Partition-3

Worker-3

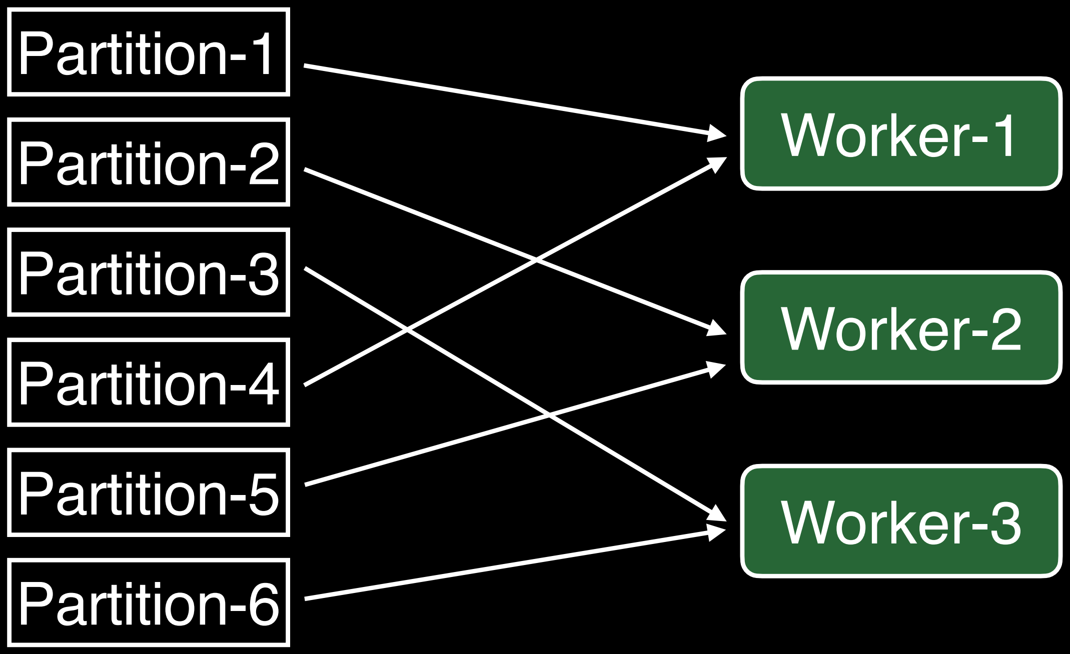
Other workers can’t pick up the slack from outlier



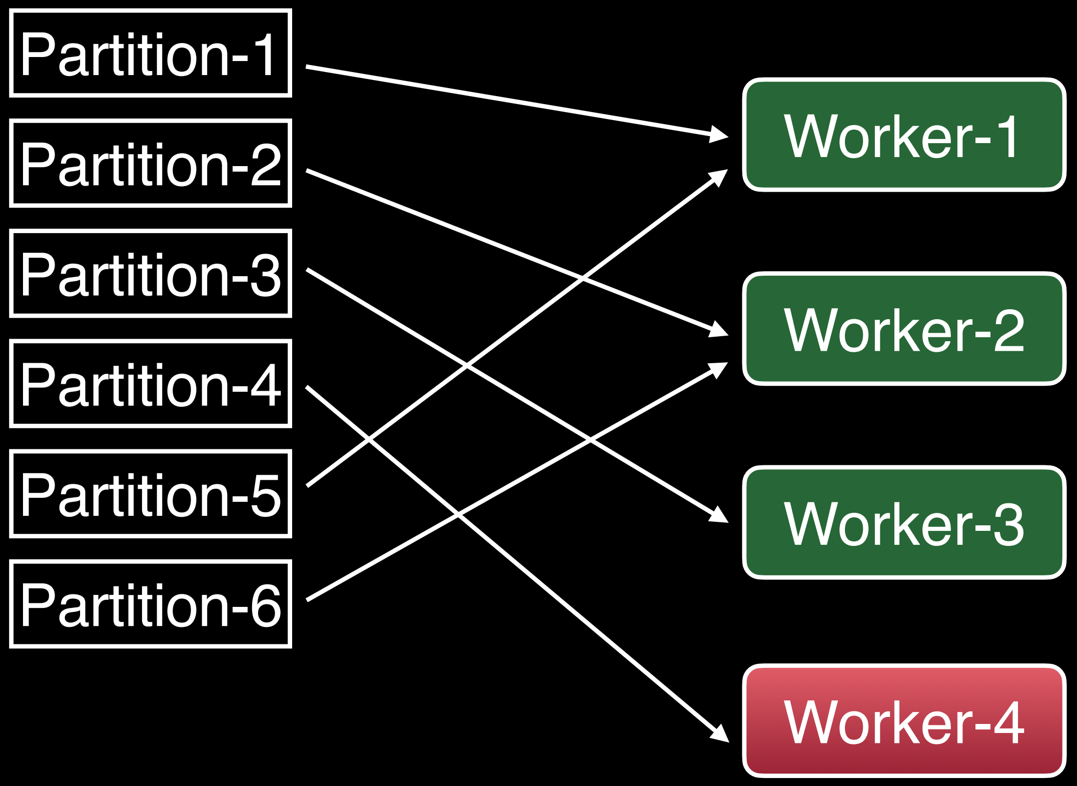
Source parallelism is limited by the number of partitions



May be difficult to get balanced partition assignment during autoscaling



May be difficult to get balanced partition assignment during autoscaling



Alternative streaming source?

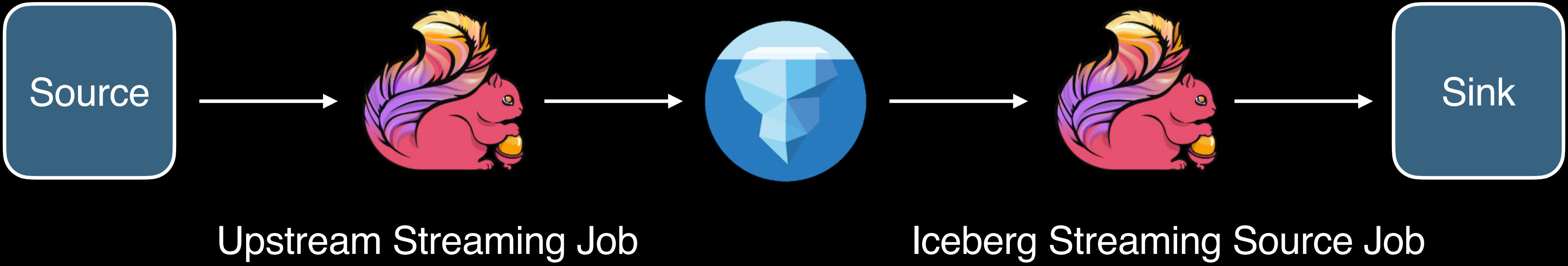
Agenda

Motivation

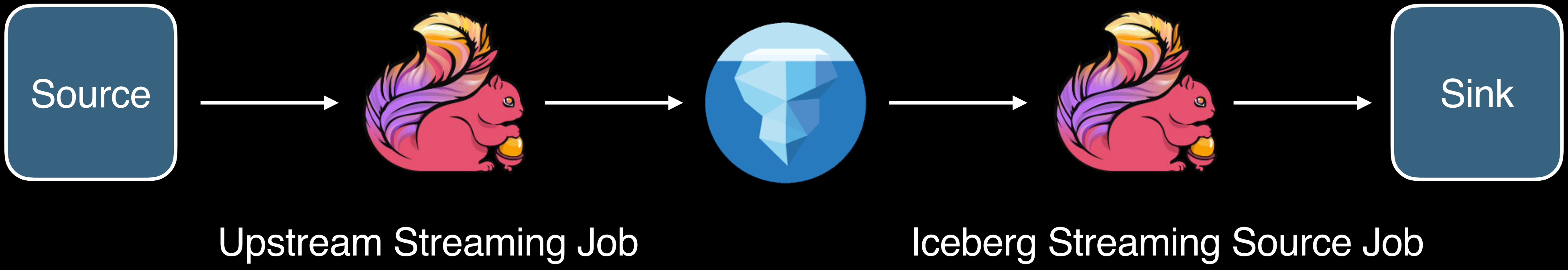
Streaming from Iceberg

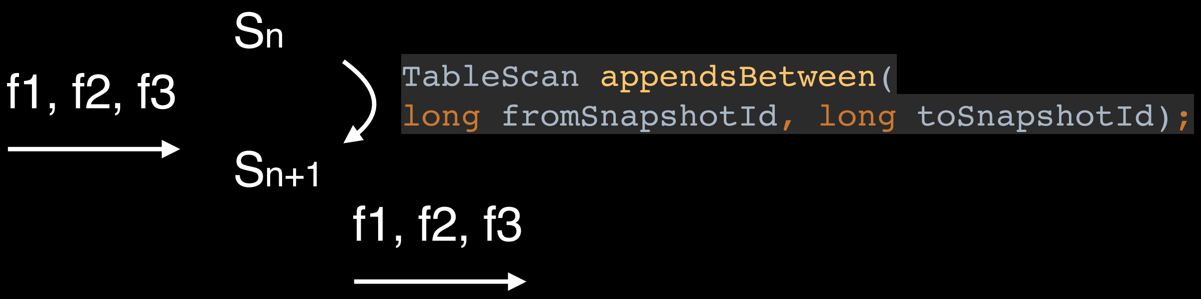
Evaluation results

Can Flink stream data from Iceberg as they are appended to the table by upstream?



Iceberg supports scan of incremental changes between snapshots



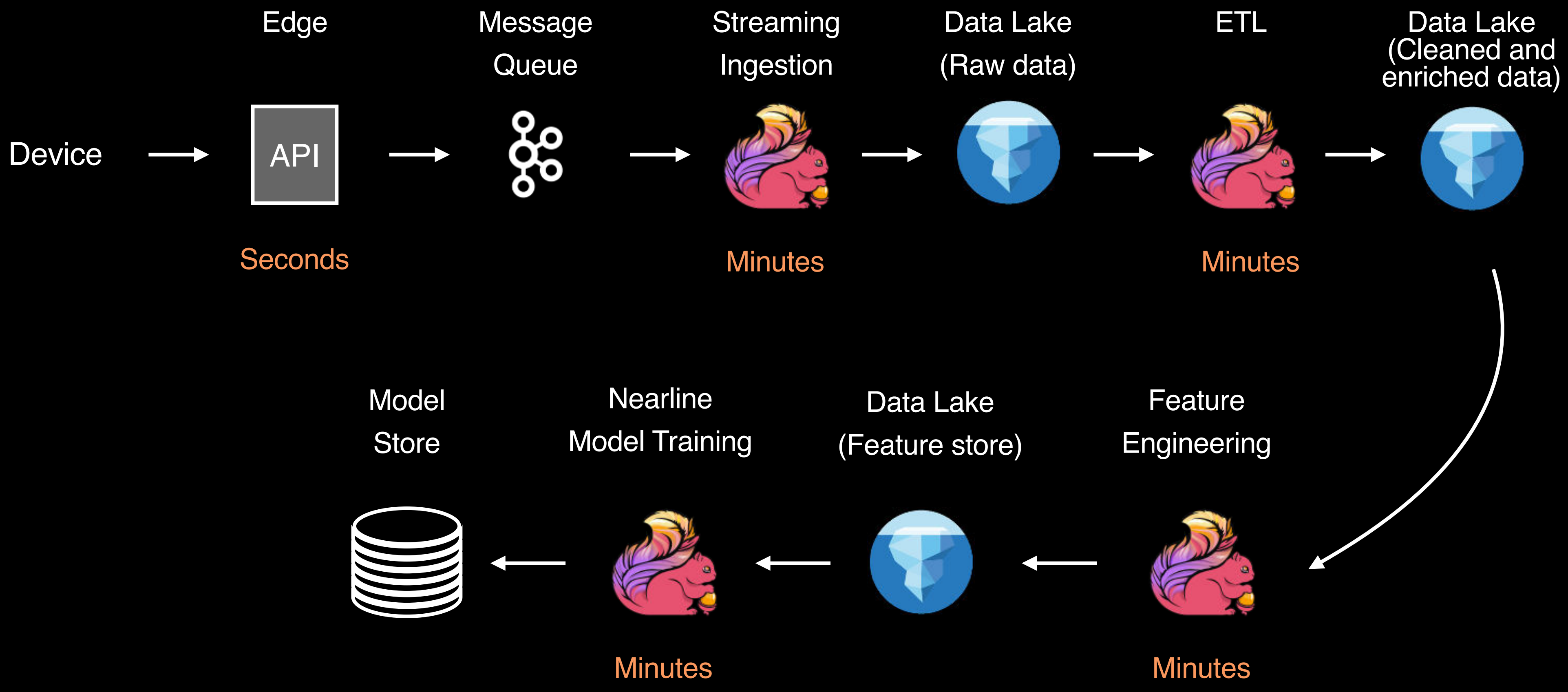


```
TableScan appendsBetween(
long fromSnapshotId, long toSnapshotId);
```

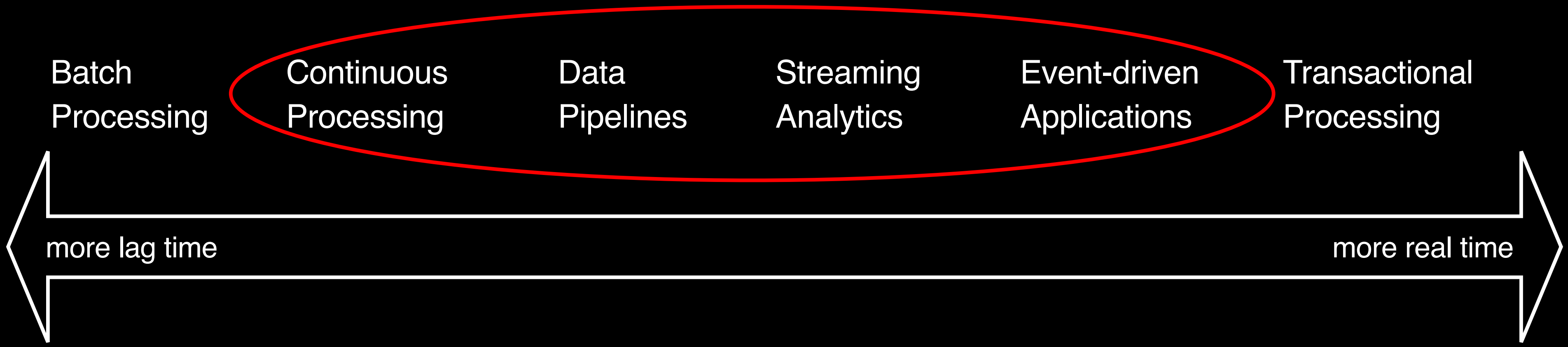
This cycle continues forever

Many streaming use cases are good with minute-level latency

Build low-latency data pipelines chained by Flink jobs streaming from Iceberg



Where does stream processing fit in the spectrum of data processing applications



Batch Processing

Continuous Processing

Data Pipelines

Streaming Analytics

Event-driven Applications

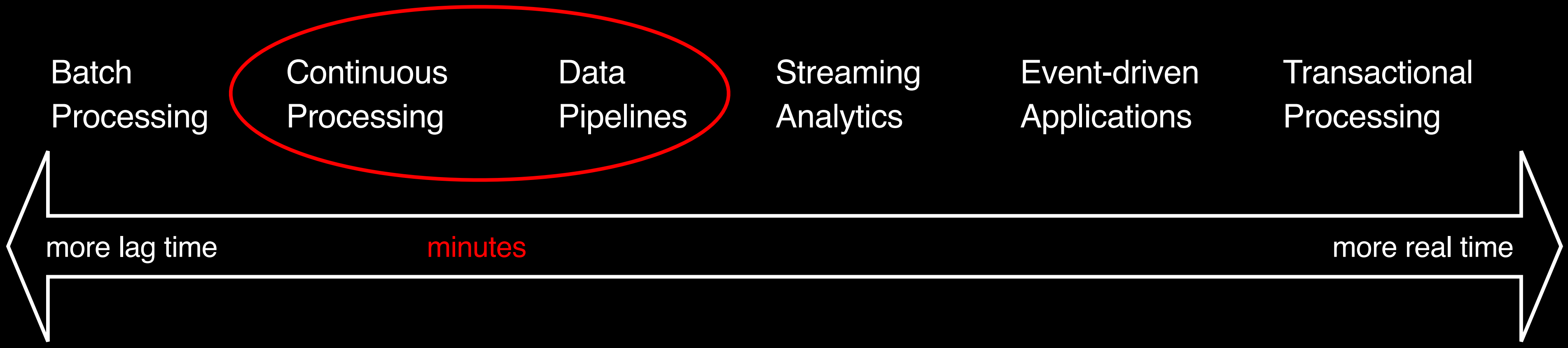
Transactional Processing

more lag time

more real time

Stephan Ewen & Xiaowei Jiang & Robert Metzger. From Stream Processing to Unified Data Processing System. Flink Forward. April 1-2, 2019. San Francisco

Flink Iceberg streaming source fits well for data pipelines and continuous processing



Batch Processing

Continuous Processing

Data Pipelines

Streaming Analytics

Event-driven Applications

Transactional Processing

more lag time

minutes

more real time

Stephan Ewen & Xiaowei Jiang & Robert Metzger. From Stream Processing to Unified Data Processing System. Flink Forward. April 1-2, 2019. San Francisco

What about incremental batch processing

- Schedule batch runs every a few minutes
- Each run discovers and processes incremental data files
- The line becomes blurry as scheduling intervals are shortened

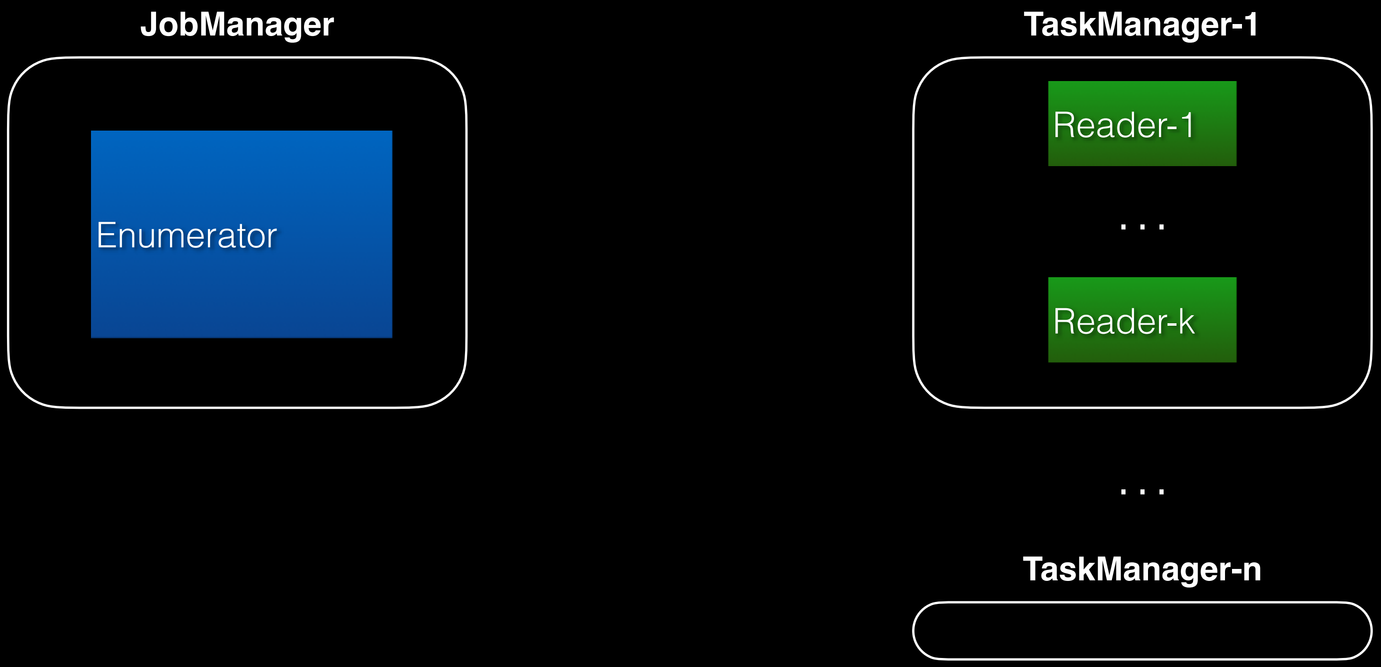
Limitations of incremental batch processing

- May be more expensive to tear down and start the batch runs when scheduling intervals are small
- Operational burden can be too high
- Intermediate results for stateful processing are lost after each run and recomputed in the next run

Implement a Flink Iceberg source based on the FLIP-27 source interface from Flink

https://github.com/apache/iceberg/projects/23

Flink FLIP-27 source interface separates work discovery with reading



JobManager

Enumerator

TaskManager-1

Reader-1

. . .

Reader-k

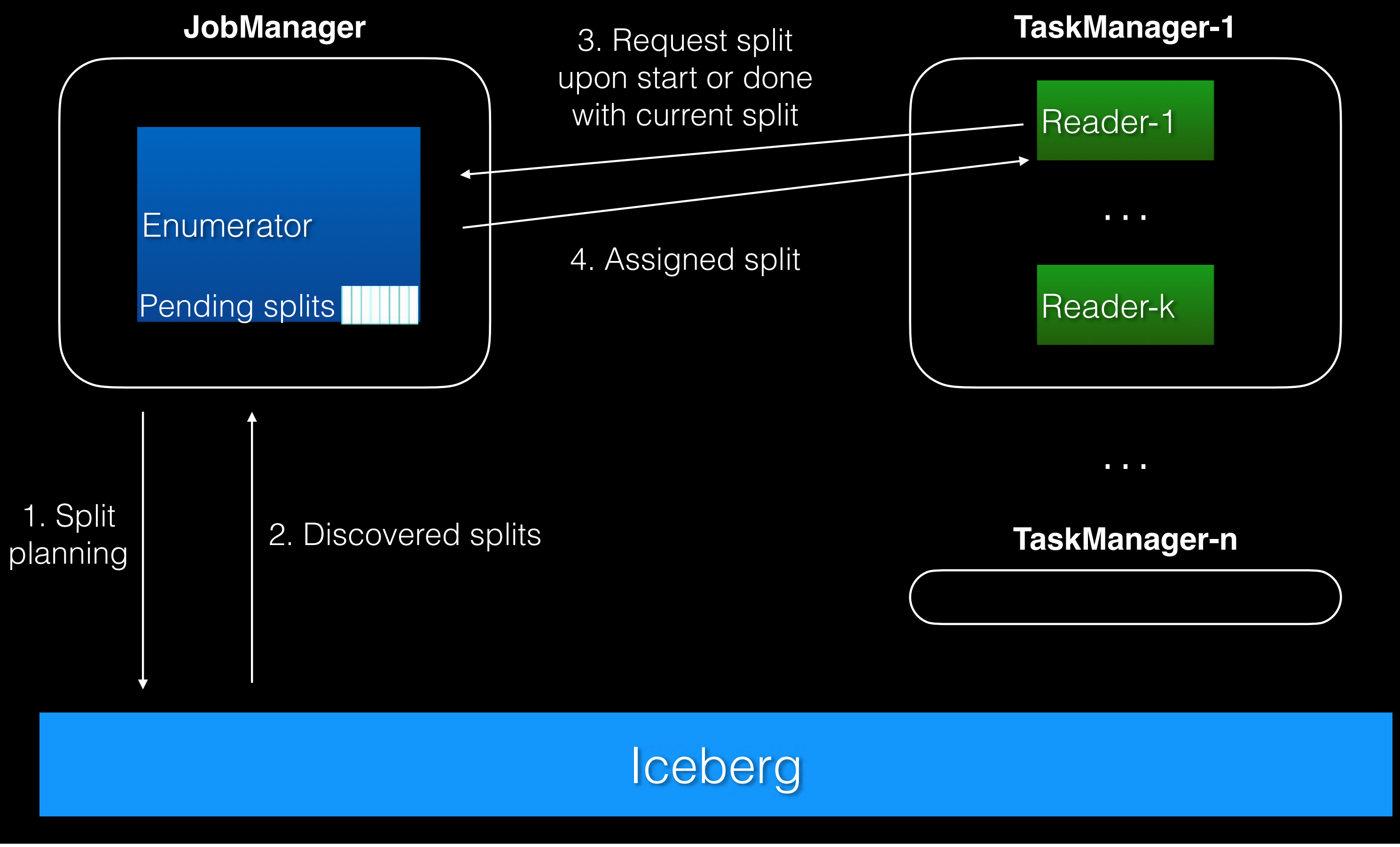
. . .

TaskManager-n

A unit of work is defined as split

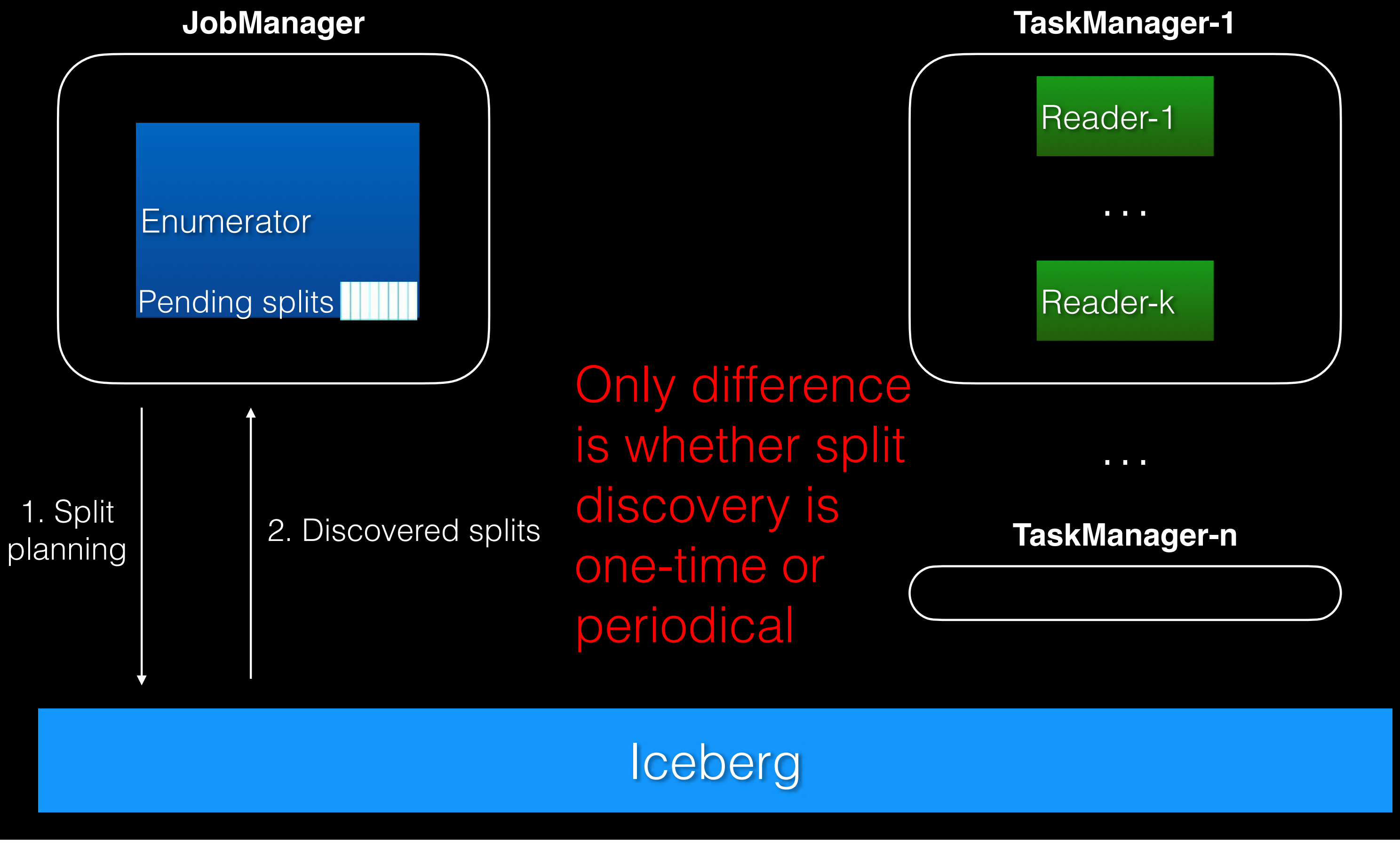
- In Kafka source, a split is a partition
- In Iceberg source, a split is a file, a slice of a large file, or a group of small files
- A split can be unbounded (Kafka) or bounded (Iceberg)

Iceberg source dynamically assign splits to readers with pull based model



Reader requests one split a time

FLIP-27 unifies batch and streaming sources



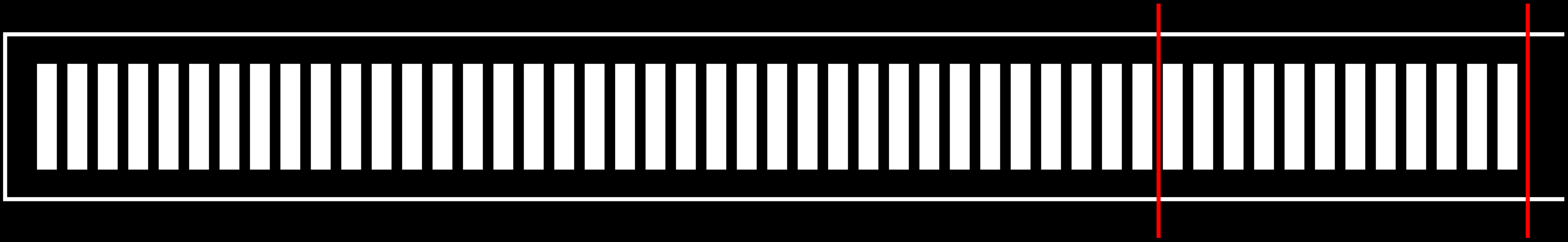
Only difference is whether split discovery is one-time or periodical

Benefits of Iceberg streaming source?

Offload operational burden to cloud blob storage

- Managed service
- Scalable
- Cost effective

Simplify the architecture with unified storage



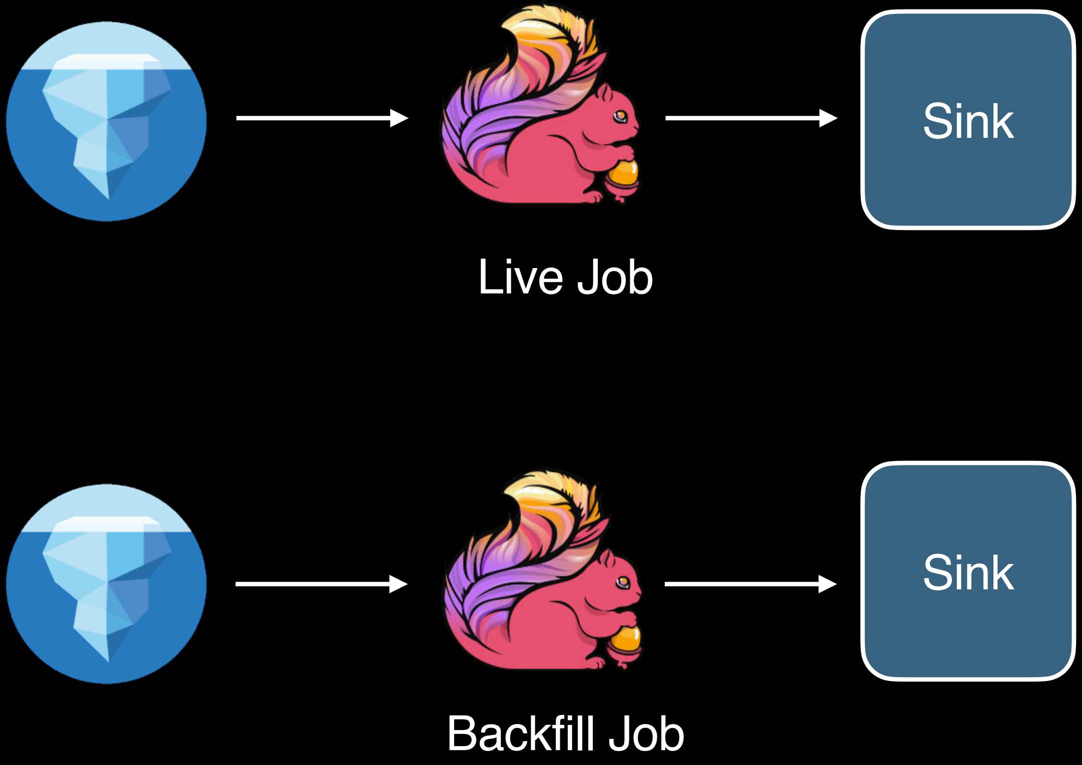
Distant
Past

Recent
Past

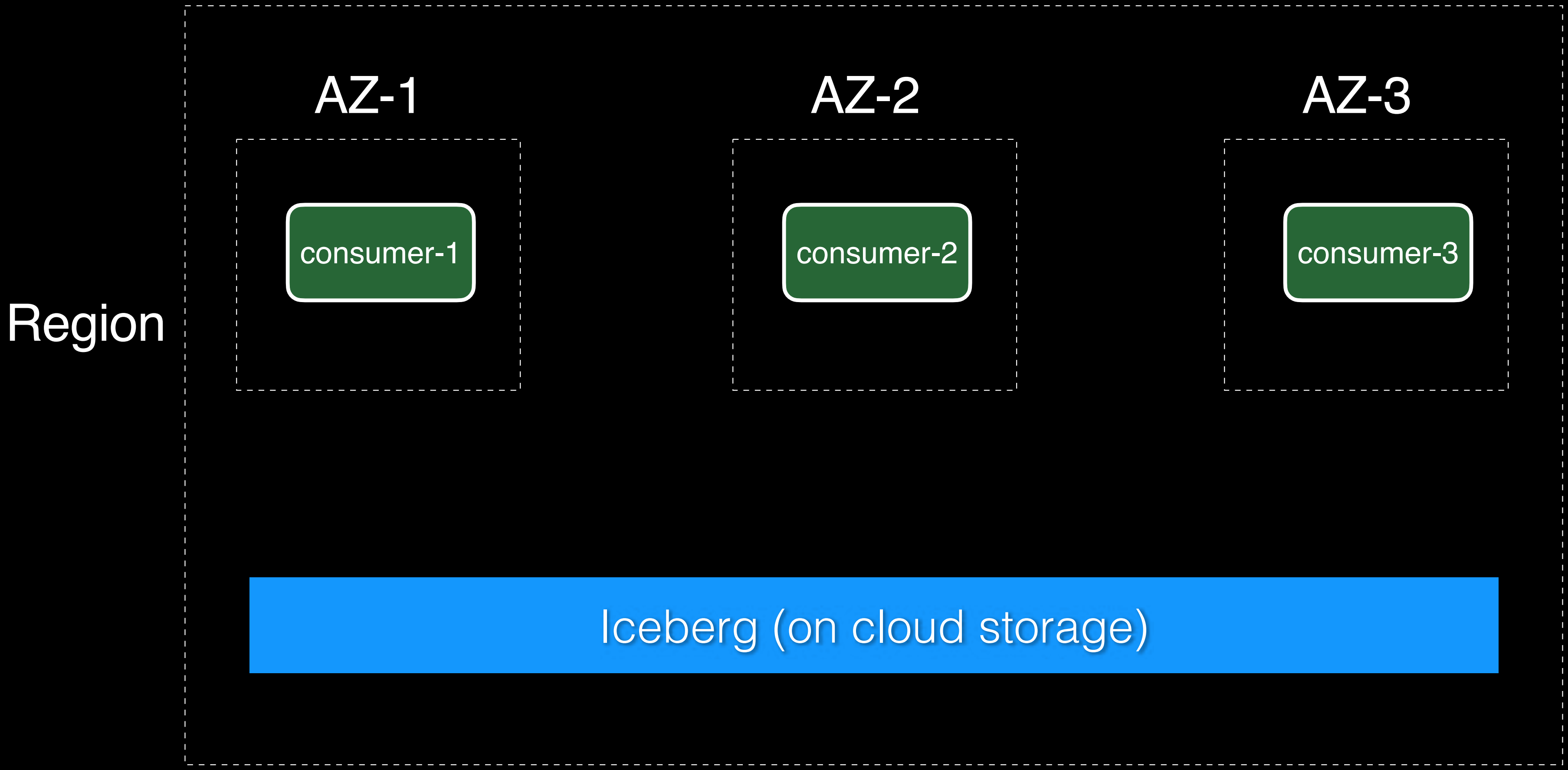
Present

Iceberg

Unify the live and backfill sources to Iceberg



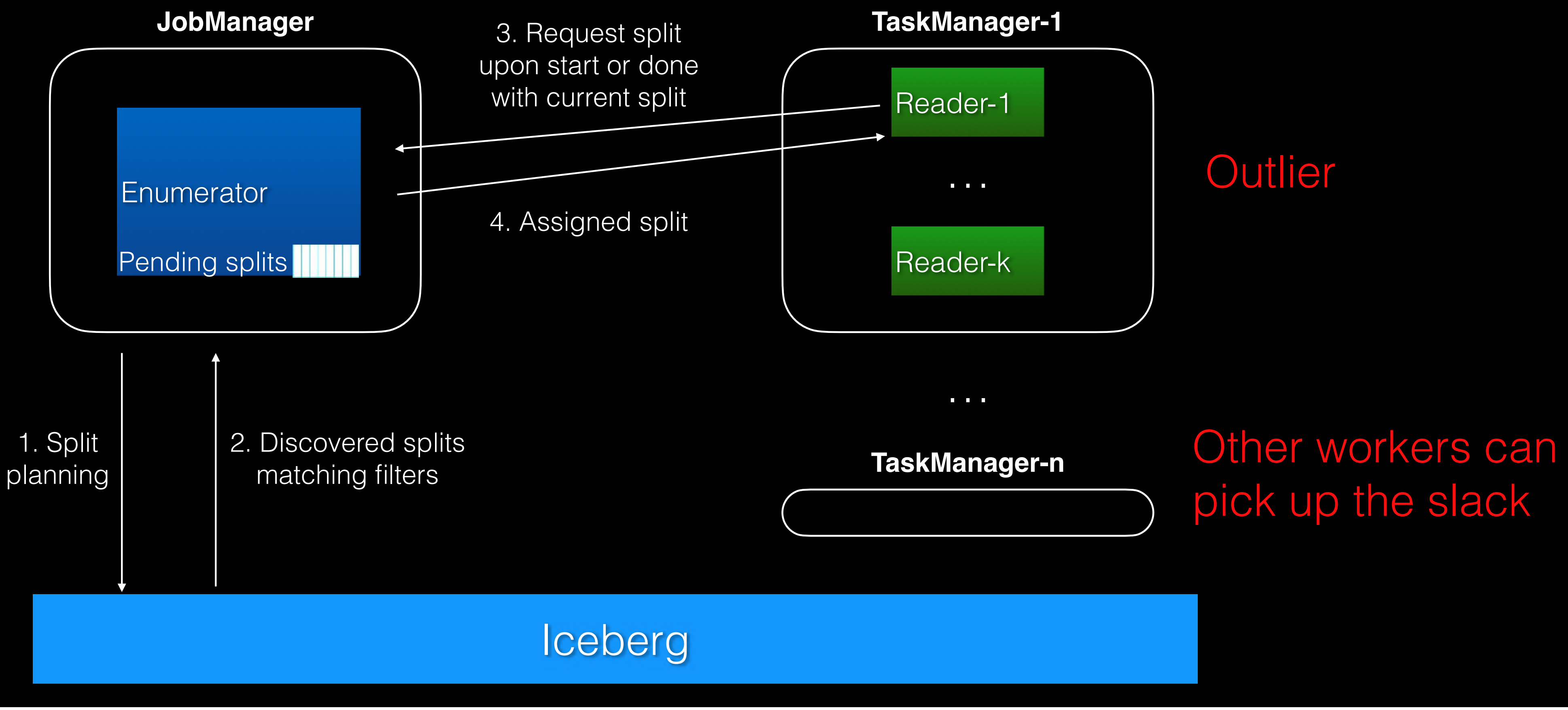
Cloud storage doesn’t charge network cost within a



Support advanced data pruning

- **File pruning (predicate pushdown)**
- **Column projection**

Dynamic pull-based split assignment allows other worker to pick up the slack



Outlier

Other workers can pick up the slack

It is more operationally friendly

- Have a lot more file segments than the number of Kafka partitions
- Can support higher parallelism
- Is more autoscaling friendly

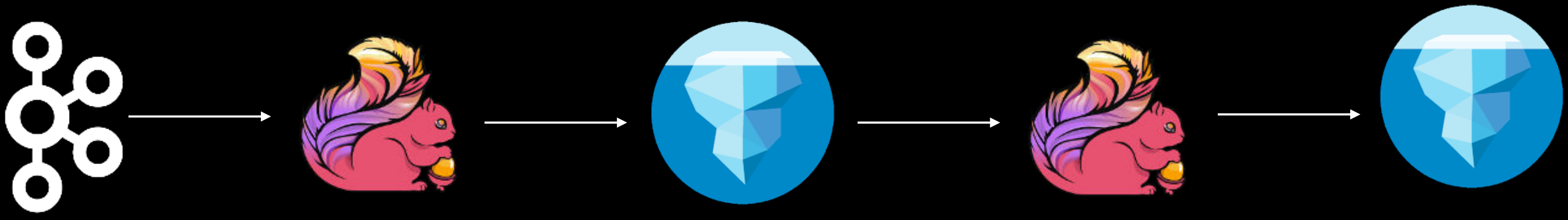
Agenda

Motivation

Streaming from Iceberg

Evaluation results

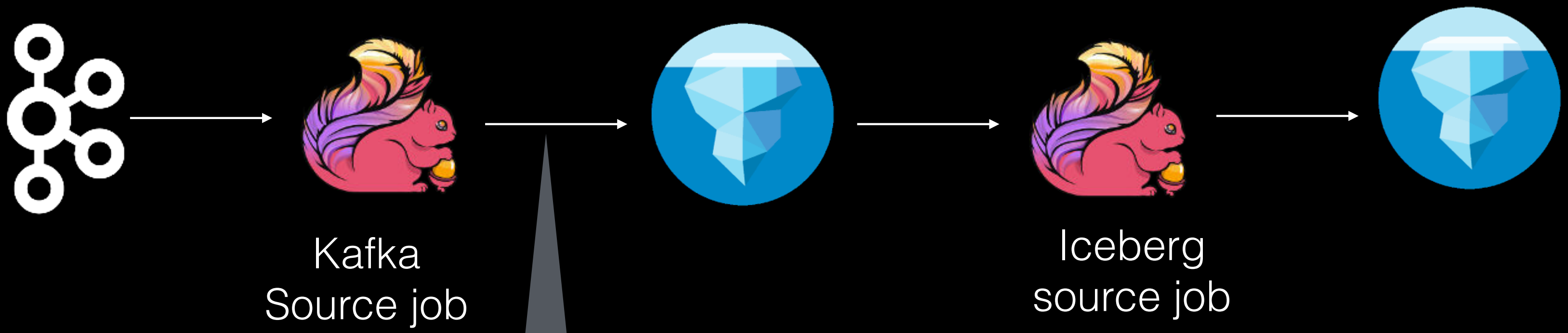
Test pipeline setup



Kafka
Source job

Iceberg streaming
source job

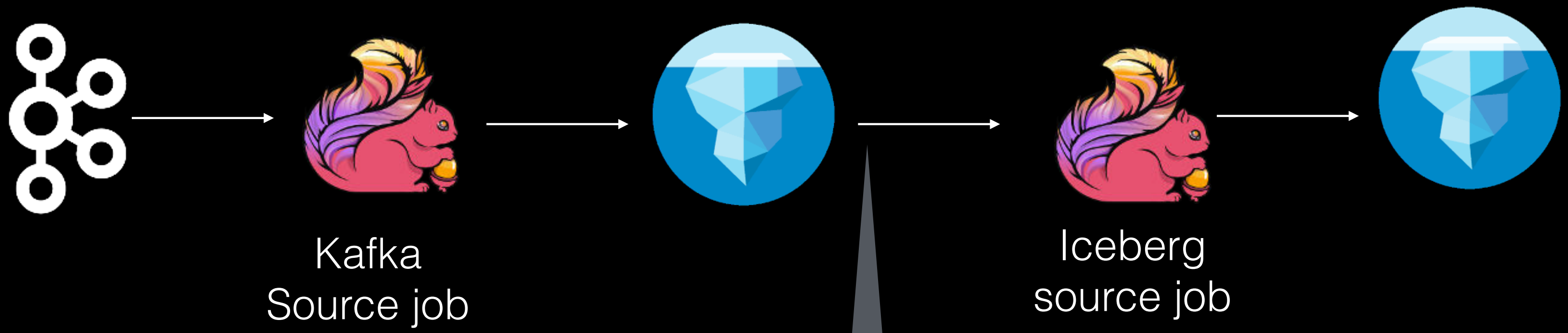
Test pipeline setup



Commit after checkpoint

- 1-10 minutes are pretty common
- Committing too often (like 1s) can overwhelm Iceberg with too many metadata files
- Committing too infrequent (like 1 hour) can lead to delay and bursty consumption for the downstream

Test pipeline setup



Periodically discover files appended between last enumerated snapshot X and current table snapshot Y

Traffic volume

- Throughput: ~3.9K msgs/sec
- Message size: ~1 KB

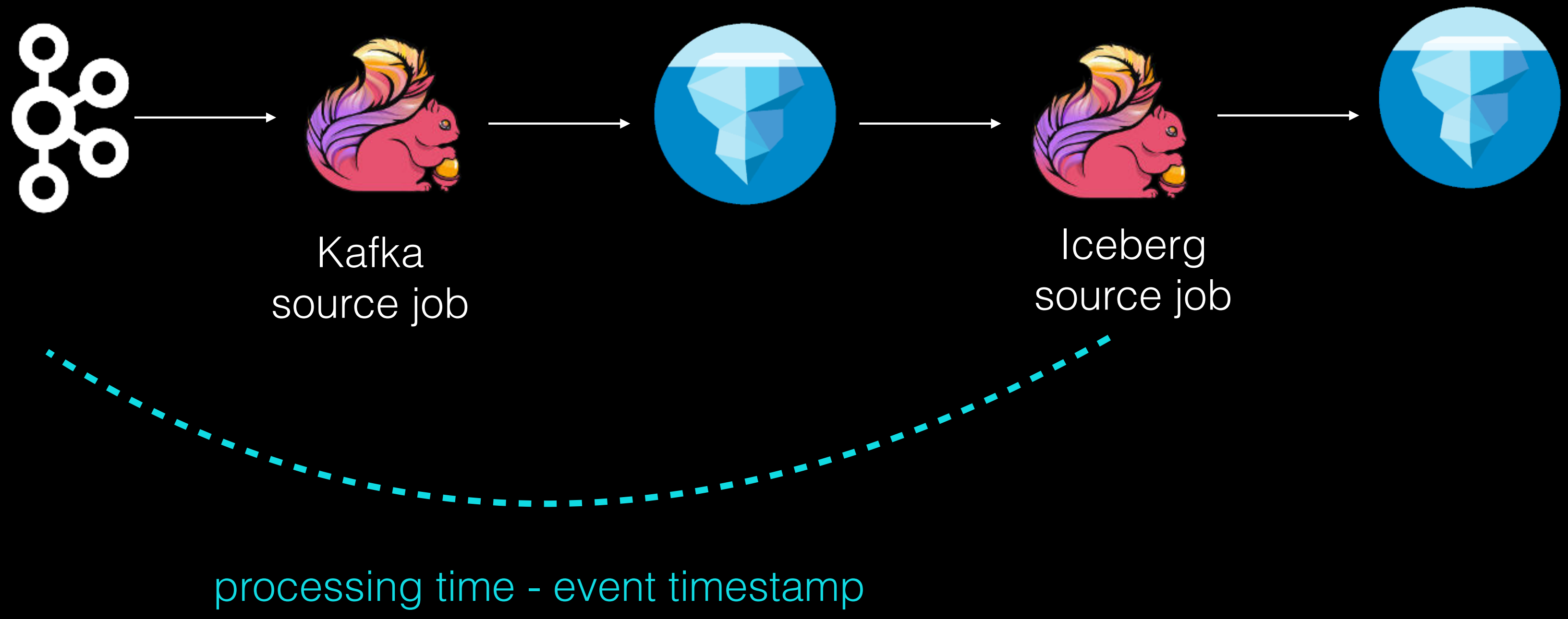
Container resource dimensions

- JobManager: 1 CPU, 4 GB memory
- TaskManager: 1 CPU, 4 GB memory

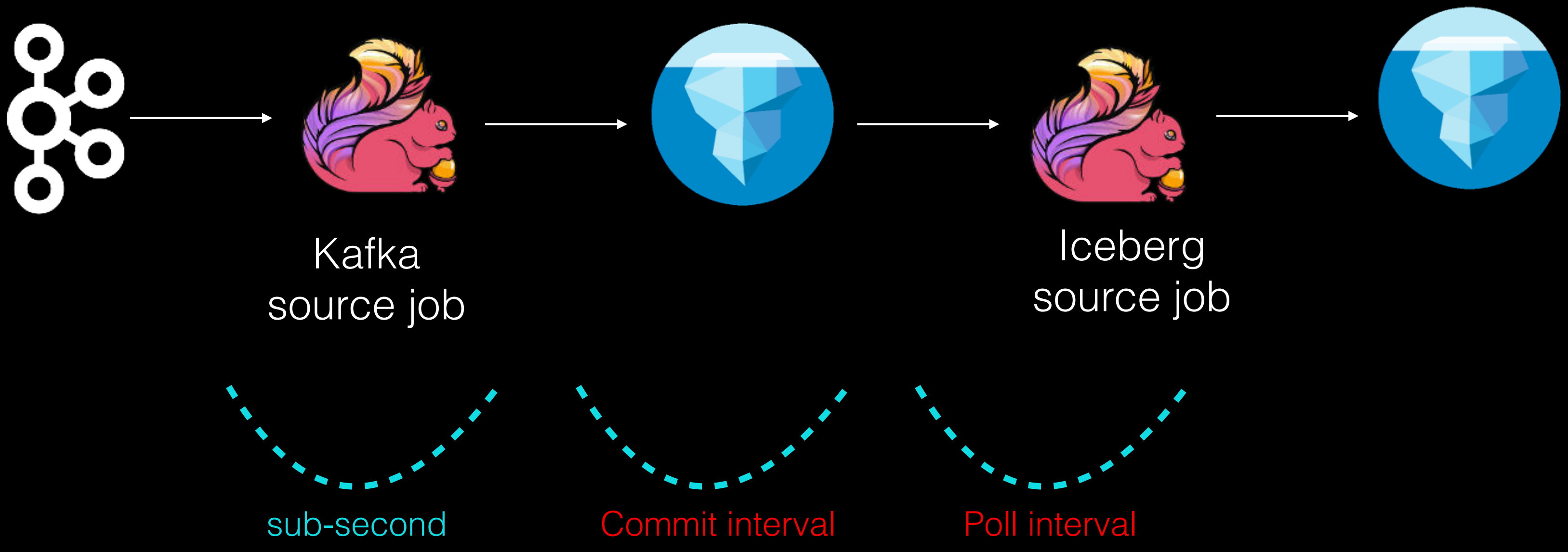
What are we evaluating

- Processing delay
- How upstream commit interval affects the bursty consumption
- CPU util comparison btw Kafka and Iceberg source

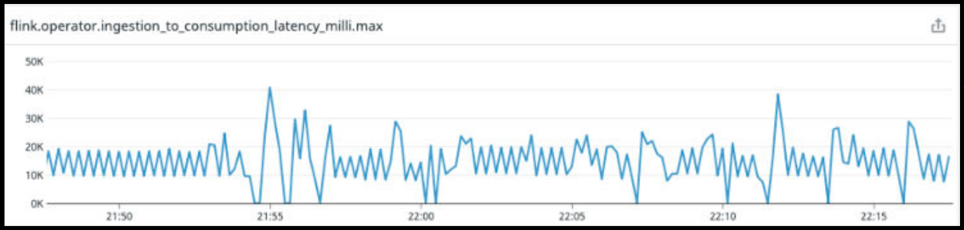
Measure the latency from Kafka to Iceberg source



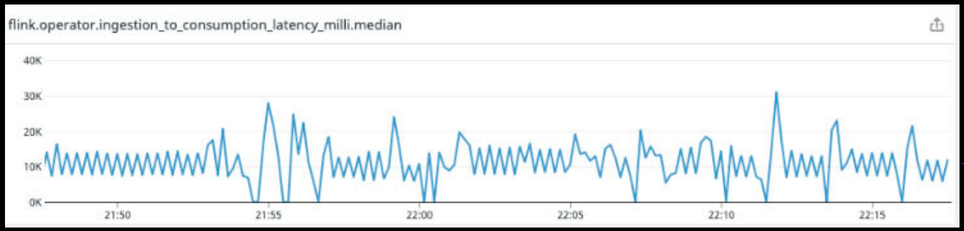
Latency is mostly decided by commit and poll interval



Latency histogram is within expected range for 10s commit and 5s poll interval



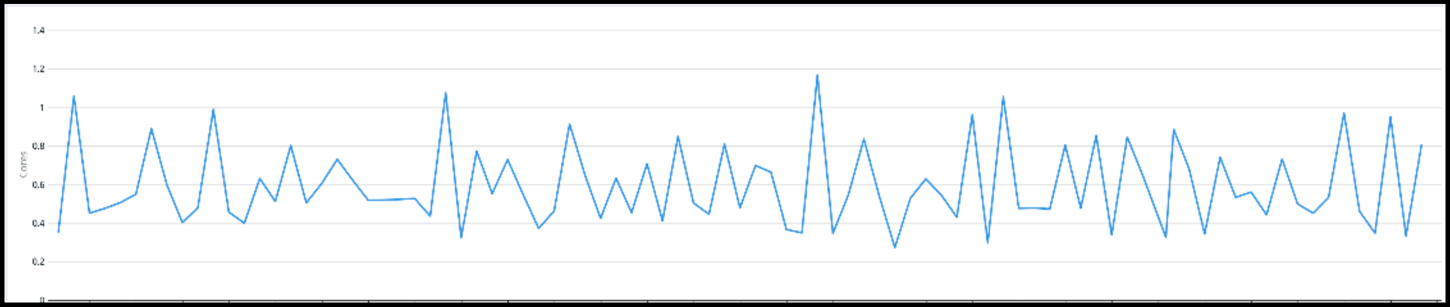
Max < 40s



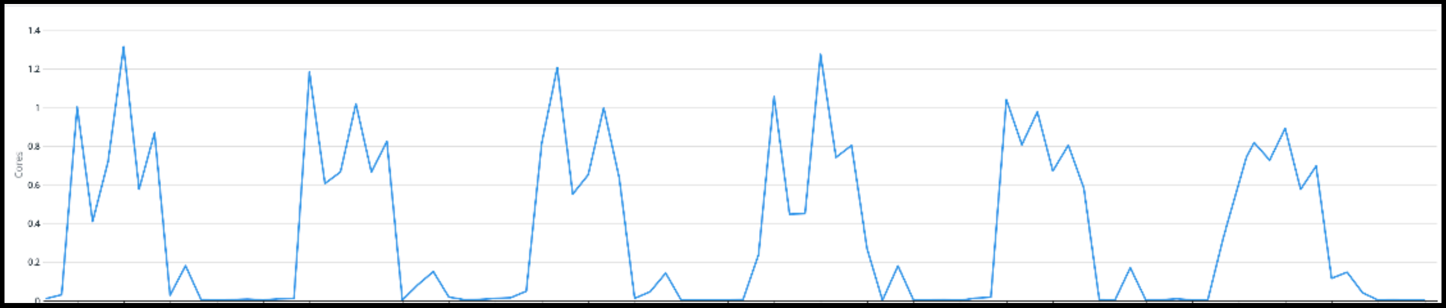
Median fluctuates around 10s

Transactional commit in upstream ingestion leads to bursty stop-and-go consumption as expected

Kafka Source job



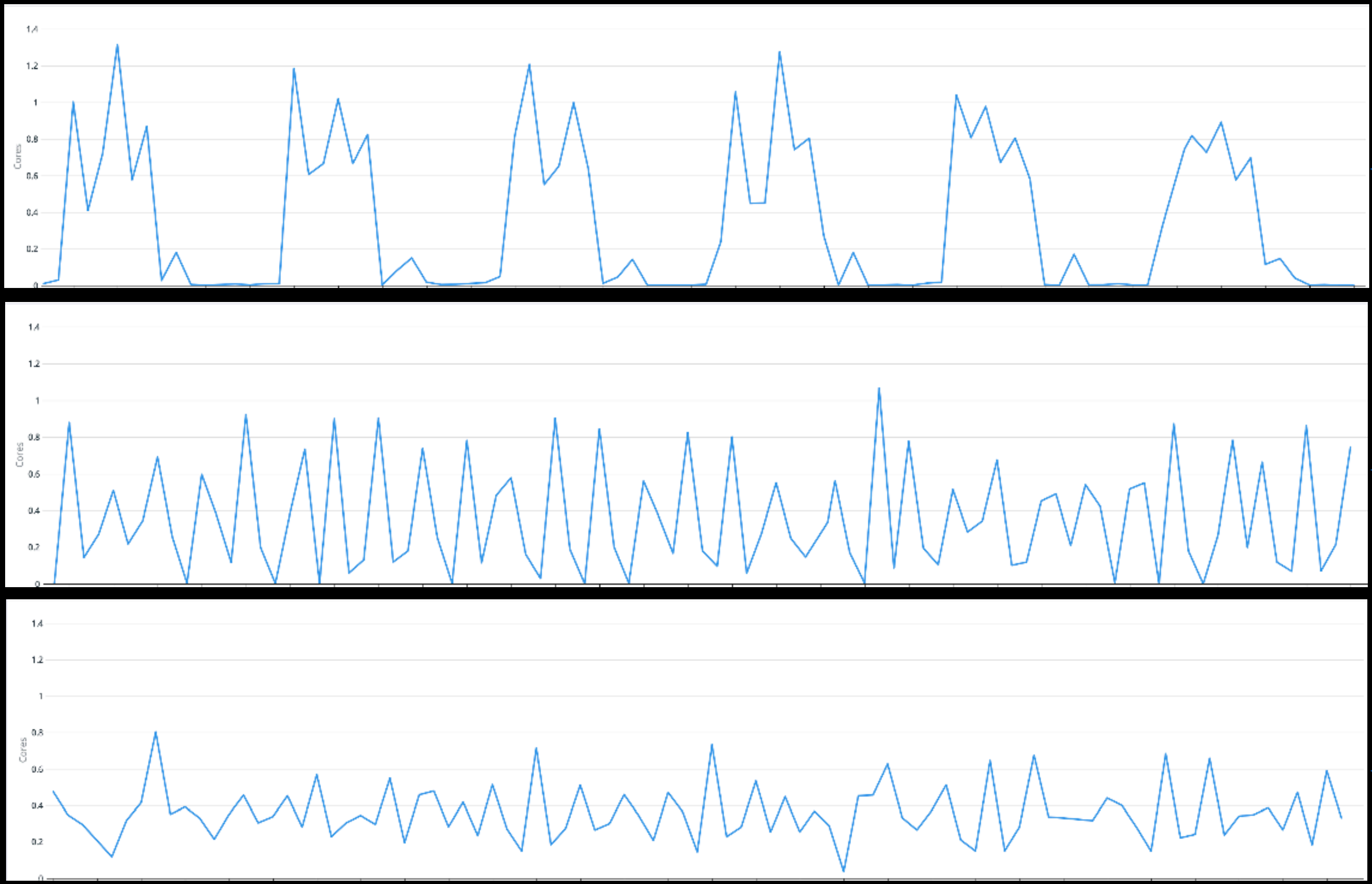
Iceberg source job



300s commit and 30s poll interval

30-minute graphing window

CPU usage becomes smoother as we shorten the upstream commit interval and Iceberg source poll interval



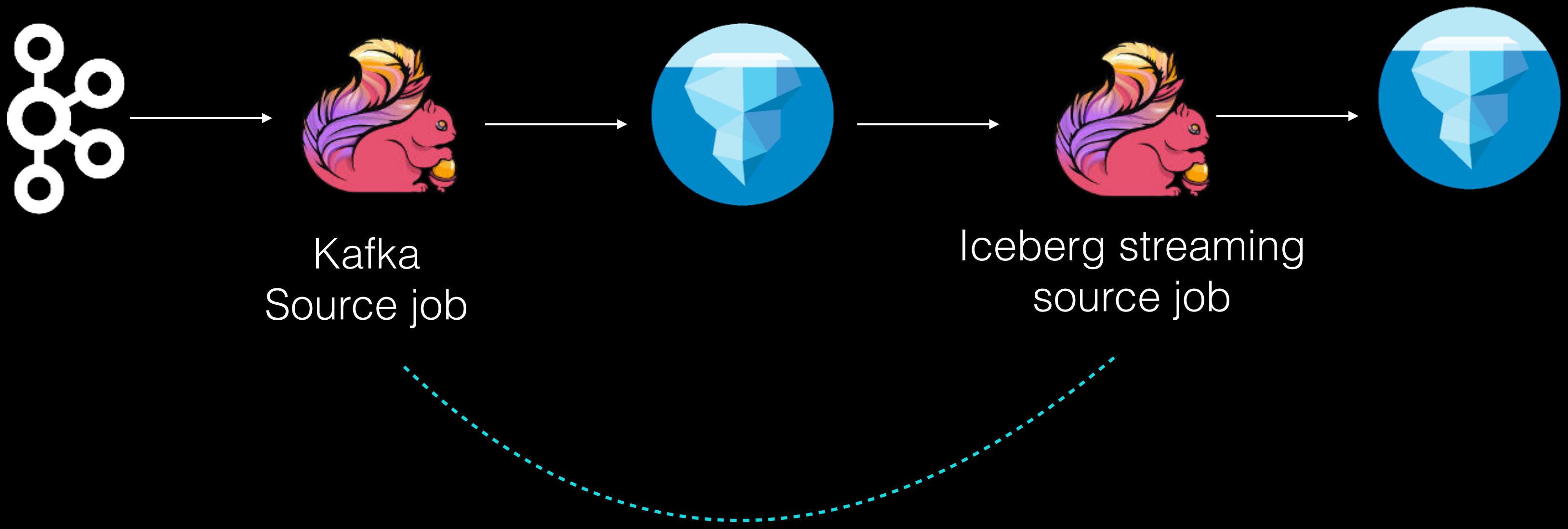
300s commit and 30s poll interval

60s commit and 10s poll interval

10s commit and 5s poll interval

30-minute graphing window

How does Iceberg source compare to Kafka source in CPU cost



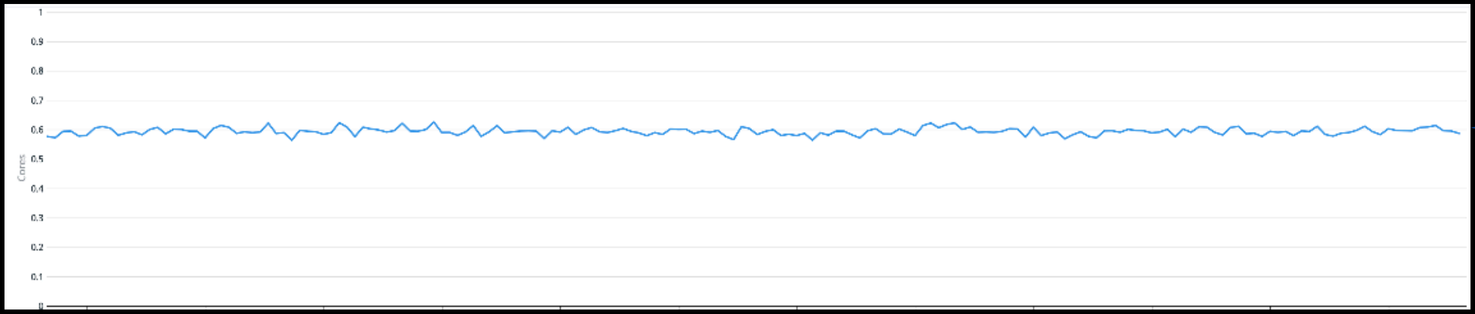
Kafka
Source job

Iceberg streaming
source job

The only difference is the streaming source: Kafka vs Iceberg

Here is the CPU usage comparison btw Kafka and Iceberg source after applying the smooth function

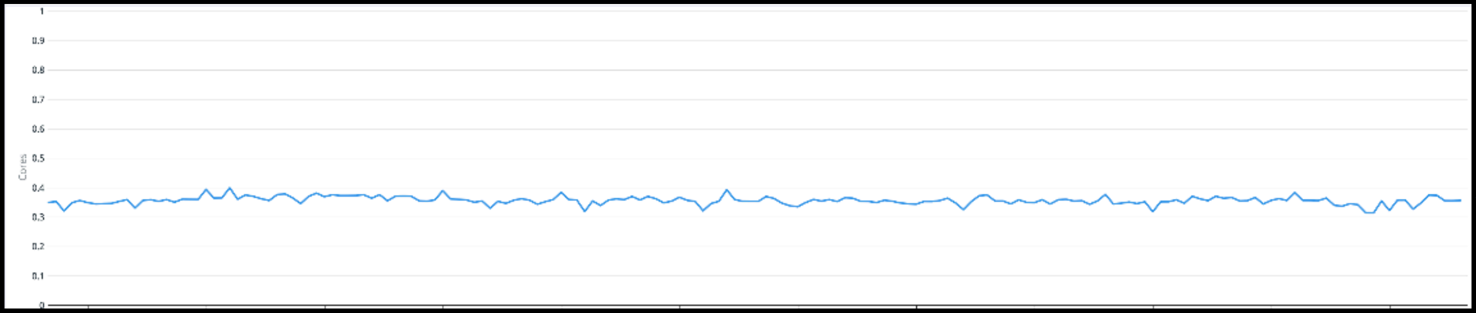
Kafka source job



~60%

Iceberg source job

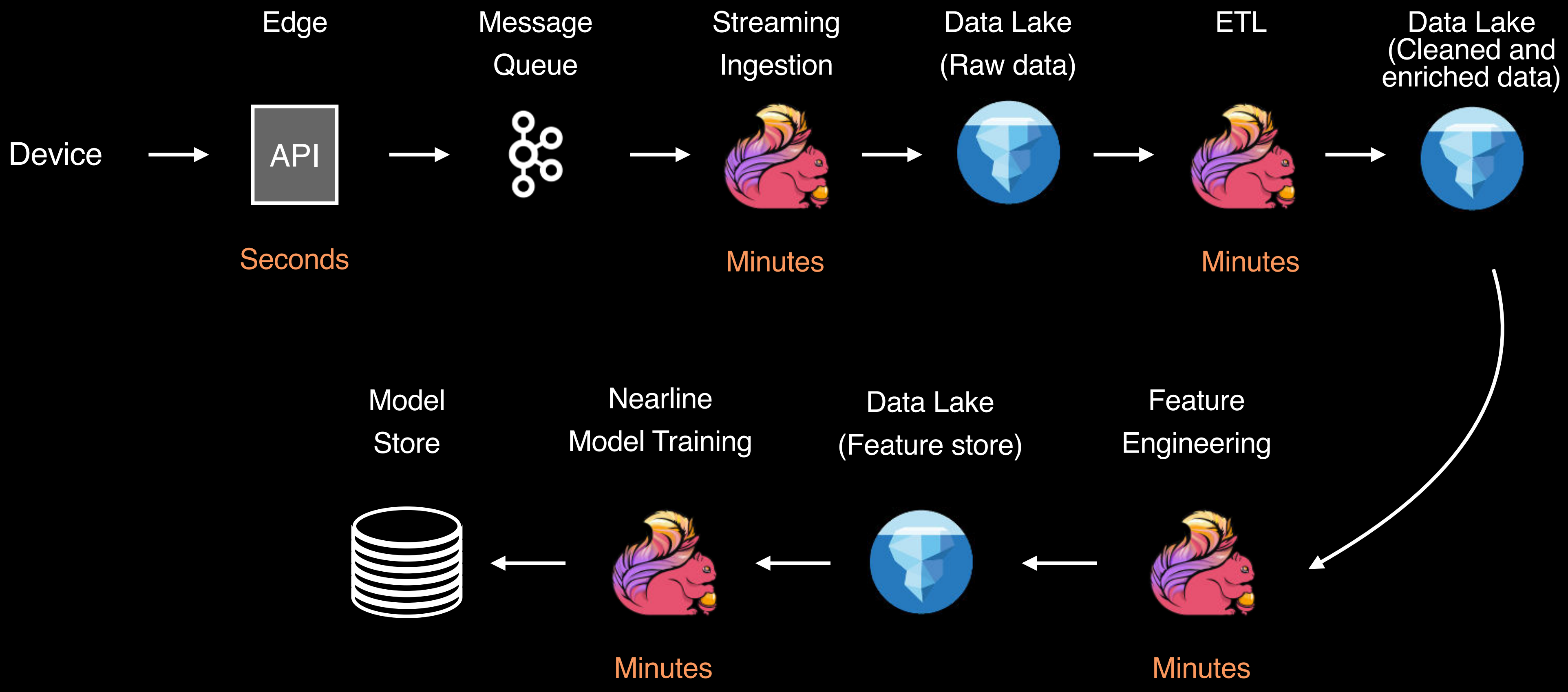
(60s commit and 10s poll intervals)



~36%

60-minute graphing window

Build low-latency data pipelines chained by Flink jobs streaming from Iceberg



Device → Edge: API (Seconds) → Message Queue → Streaming Ingestion (Minutes) → Data Lake (Raw data) → ETL (Minutes) → Data Lake (Cleaned and enriched data)

→ Feature Engineering (Minutes) → Data Lake (Feature store) → Nearline Model Training (Minutes) → Model Store