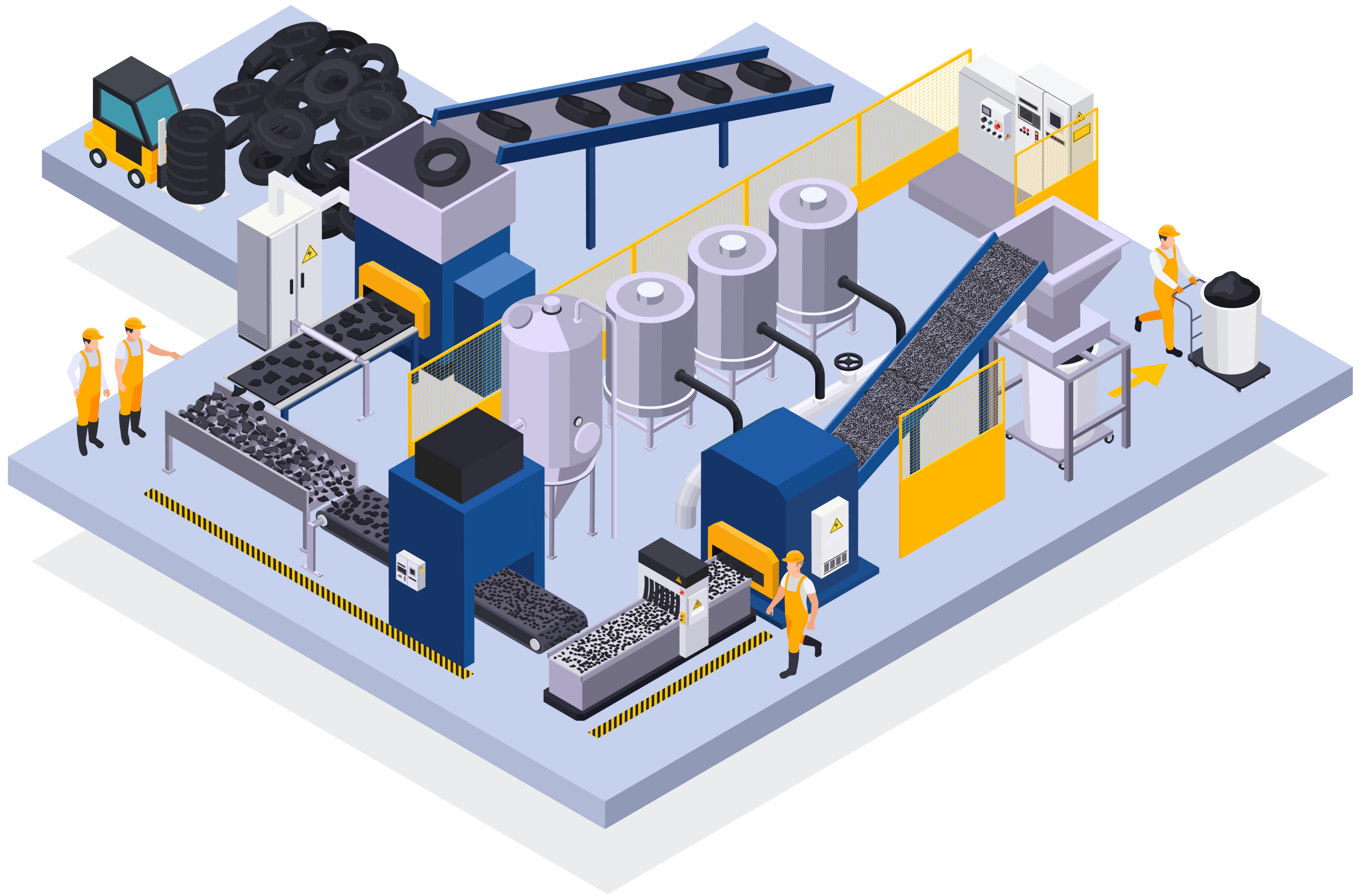
Efficient Kappa Architecture with Trino

Sanghyun Lee - SK Telecom

Manufacturing Data

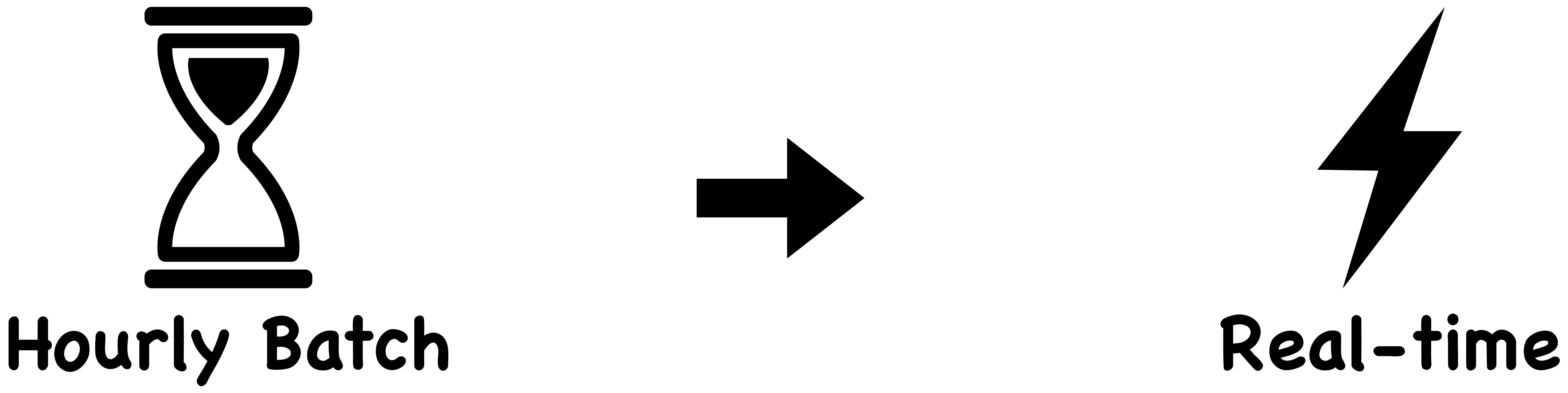


Generated at 3M TPS
Accumulated in PB

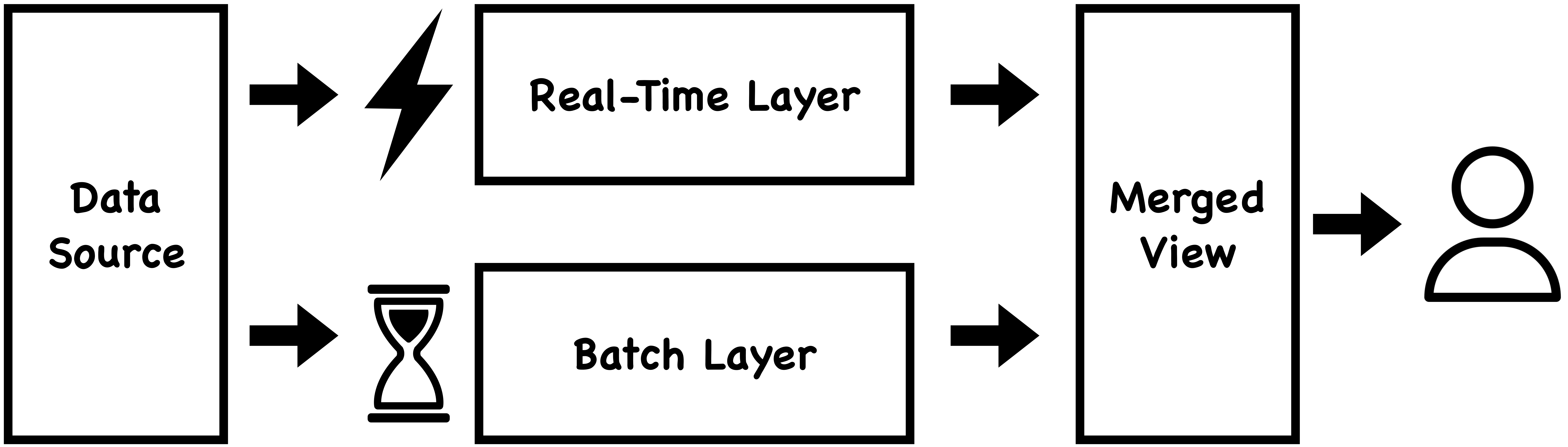
Trino Cluster

trino

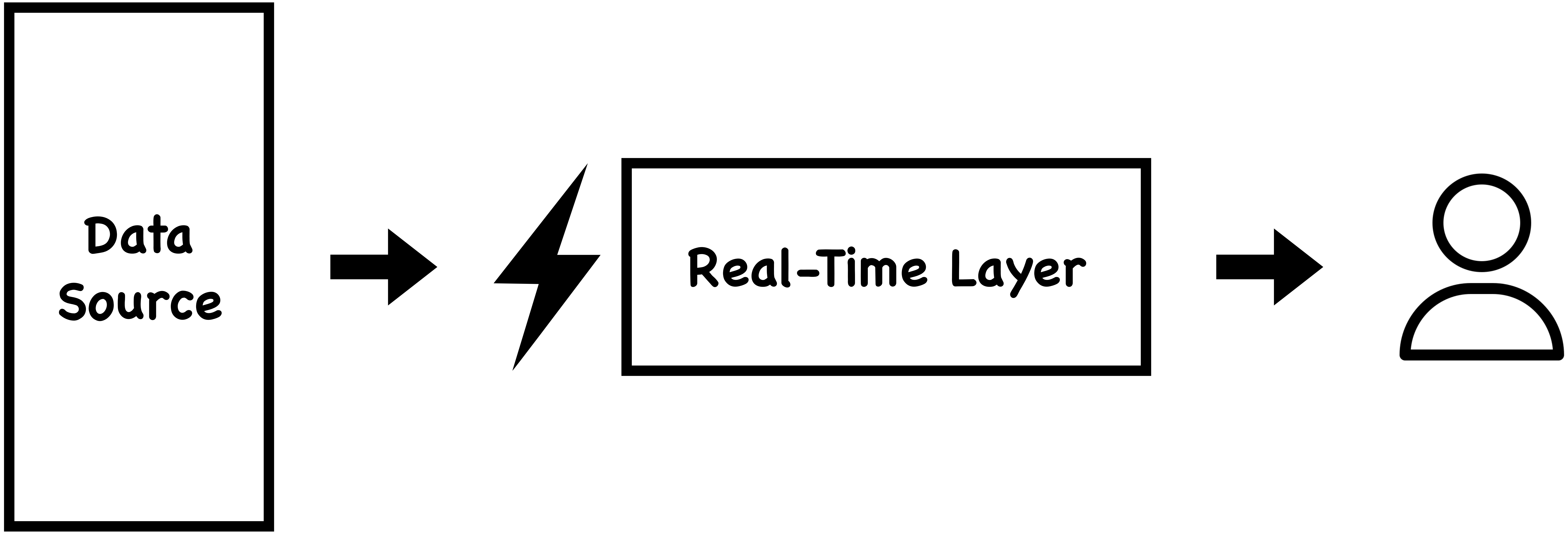
100+ of nodes
300+ queries per minute
TB size query input


Hourly Batch
Real-time

Lambda Architecture



Kappa Architecture

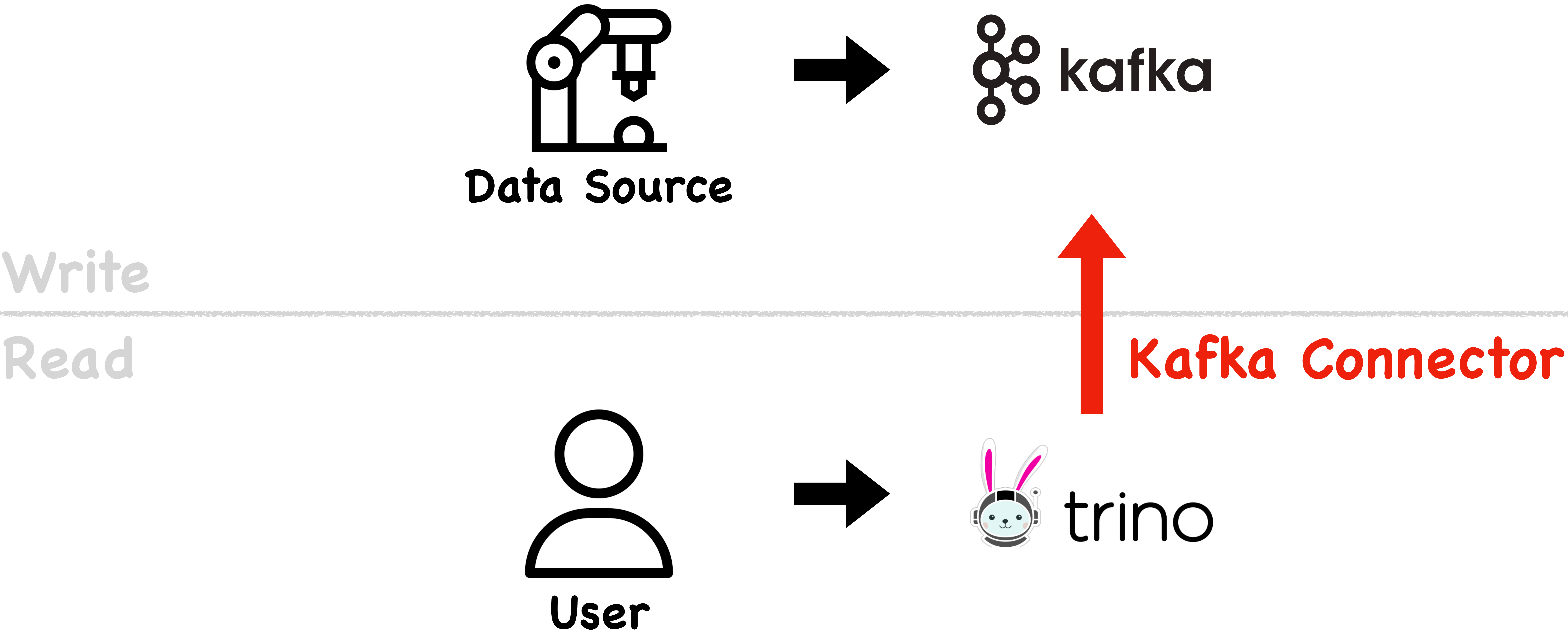


Kappa Architecture

- **Goals**
  - Exactly-once delivery
  - Low latency
  - High ingestion performance
  - High query performance

Kappa Architecture

Kappa Architecture



Kappa Architecture

- **Trino's Kafka connector**
  - Limited query performance
  - Predicate pushdown fields:
    - Kafka offset
    - Kafka timestamp
    - Kafka partition ID
  - No predicate pushdown for message ➔ **Full scan**

Kappa Architecture

Real-time Processing Engine

Spark (Structured Streaming)

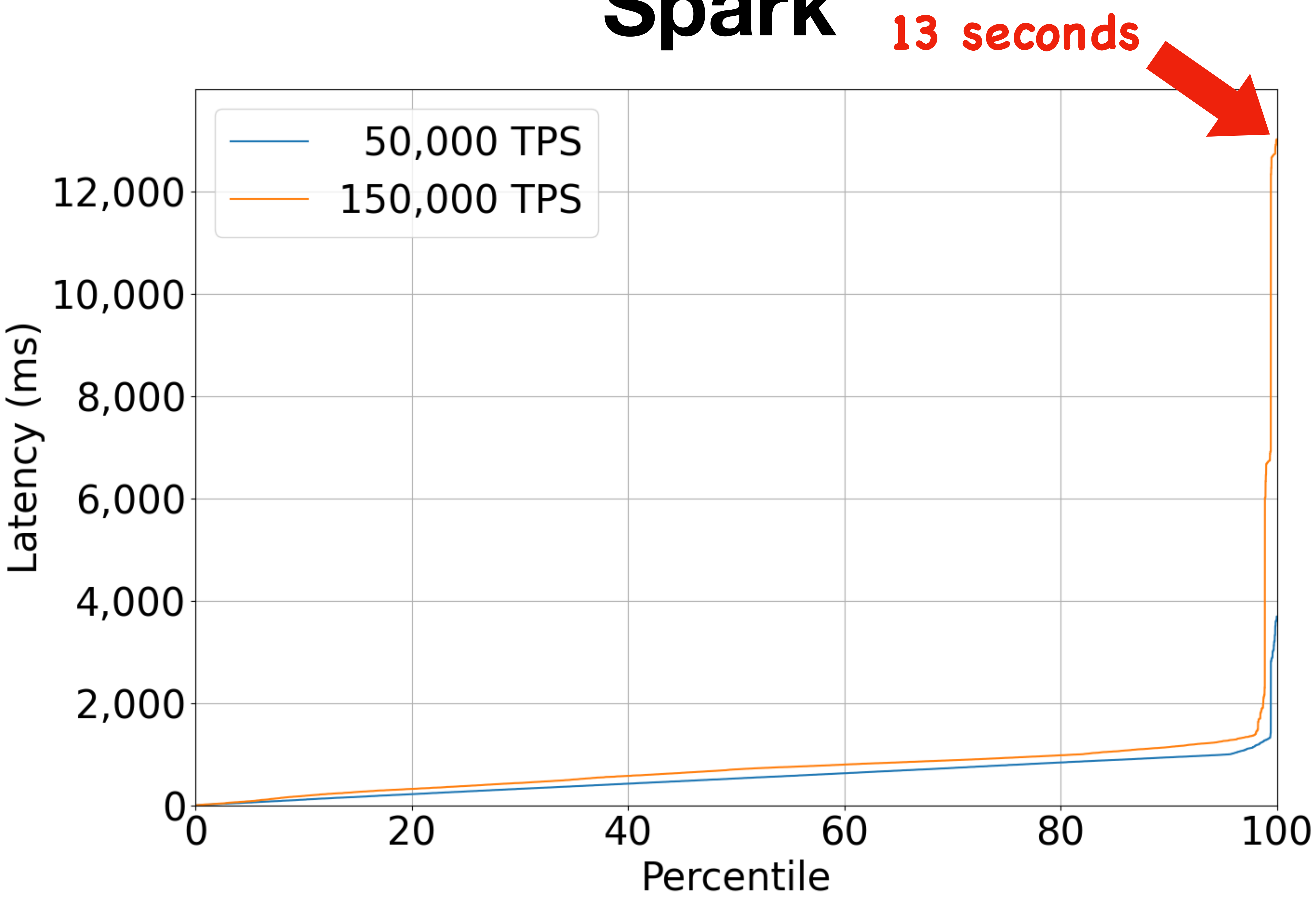
Flink

Real-time Processing Engine

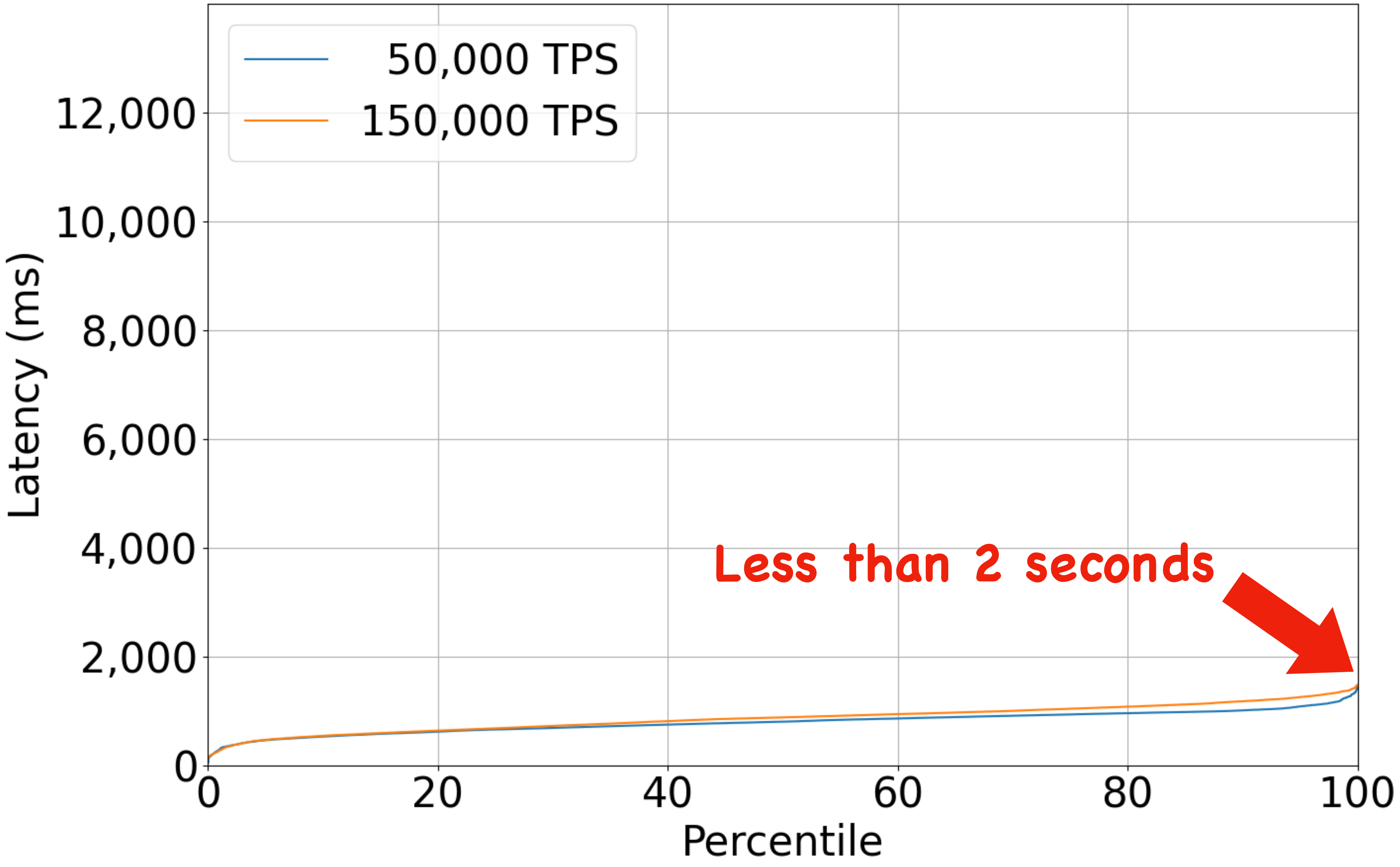
Processing Engine	Spark		Flink
Mode	Micro Batch	Continuous	
Exactly-once	✔	✘	✔
Low Latency	✘	✔	✔

Real-time Processing Engine

Spark



Flink



Streaming Benchmarks (Yahoo, https://github.com/yahoo/streaming-benchmarks)

Real-time Processing Engine

- **Spark**
  - Basic stream processing features
  - e.g. watermark, windowing, stream join
- **Flink**
  - Advanced stream processing features
  - e.g. custom window, custom trigger, evictor, side output

```
sparkDataFrame
    .withWatermark()
    .groupBy()
    .window()
    .agg()

flinkDataStream
    .assignTimestampsAndWatermarks()
    .keyBy()
    .window()
    .trigger()
    .evictor()
    .allowedLateness()
    .sideOutputLateData()
    .reduce/aggregate/apply()
```

Real-time Processing Engine

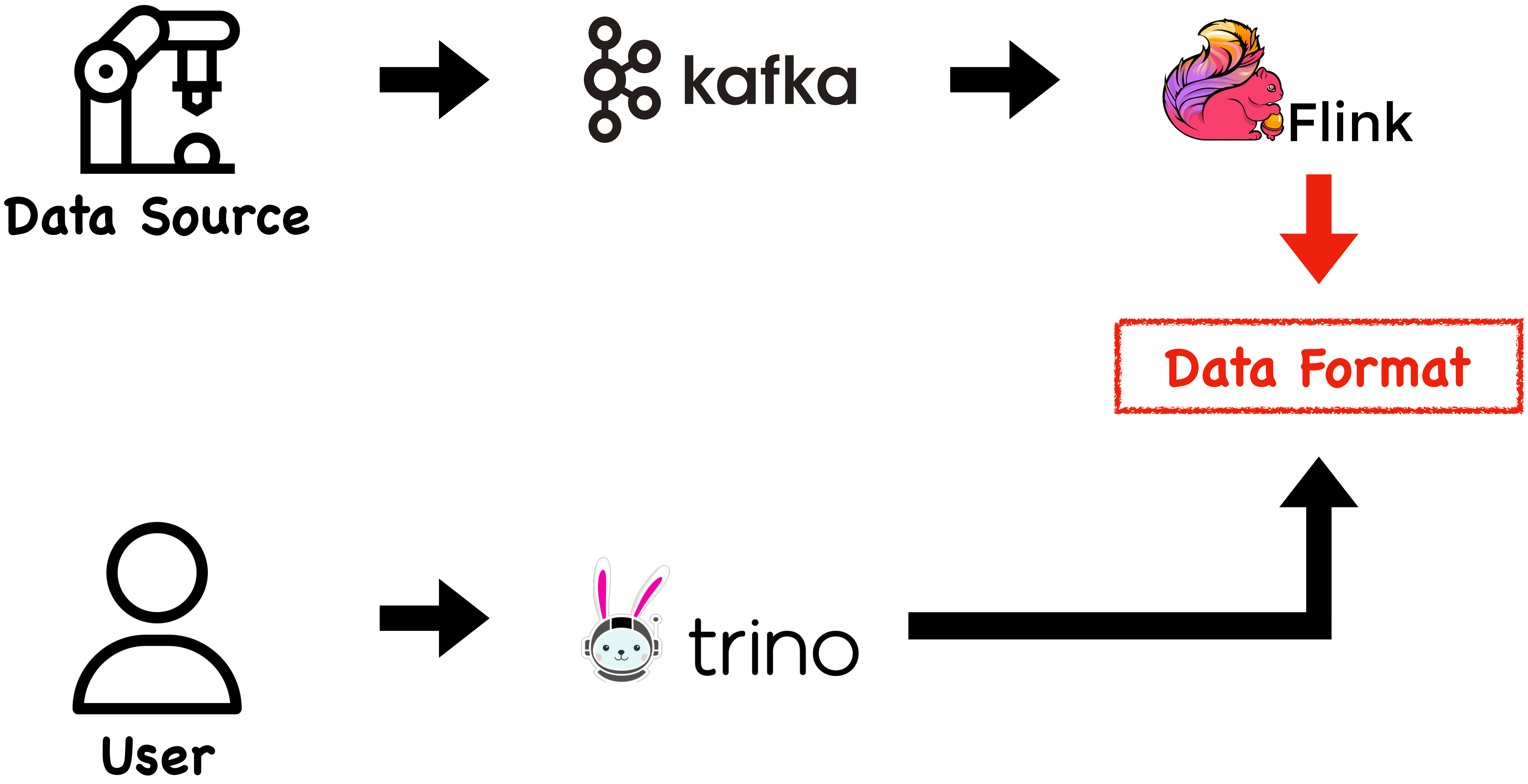
- Not sensitive to latency
- Only needs basic streaming features

➔ **Spark**

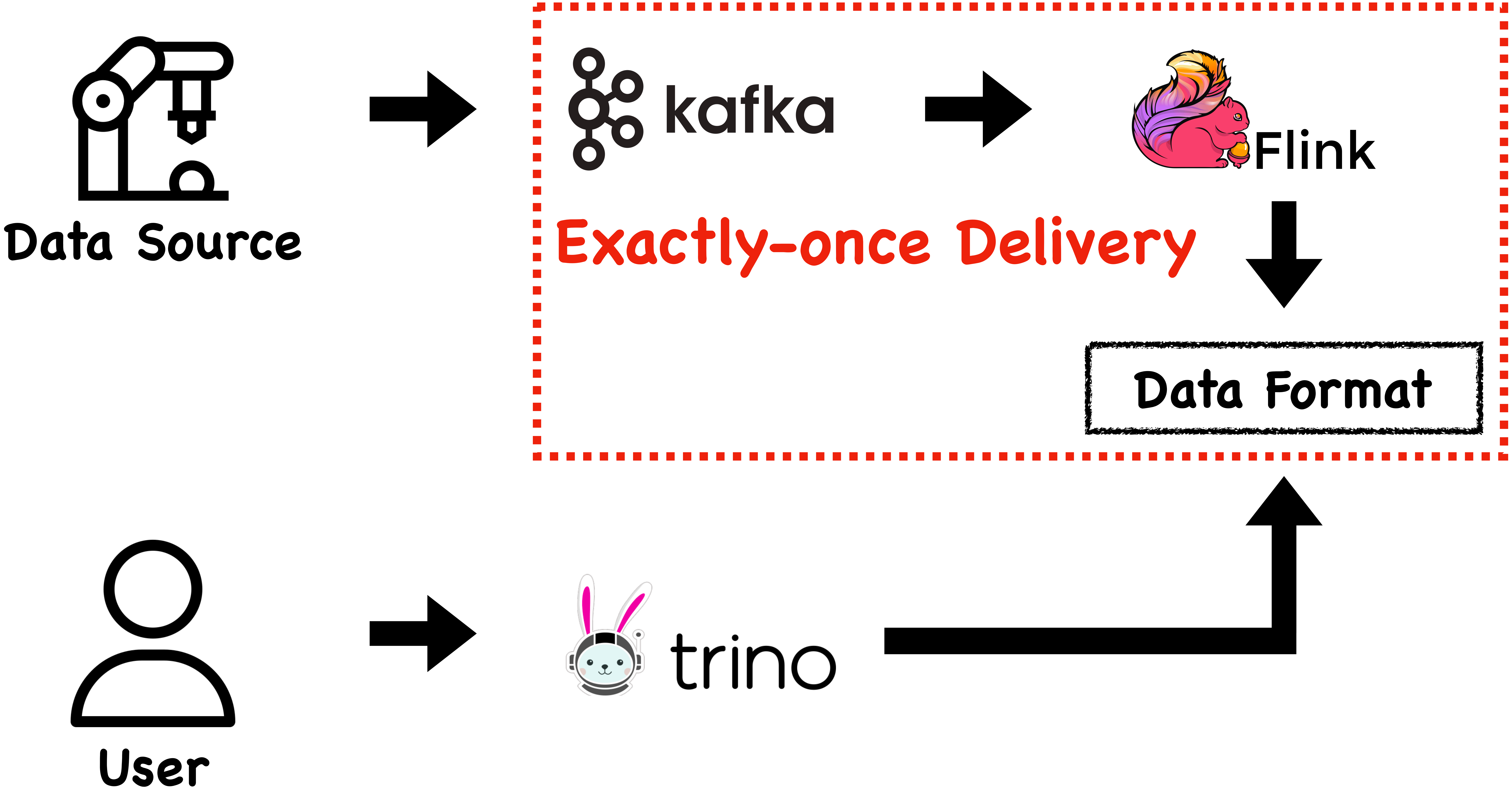
- Latency is important
- Needs advanced streaming features

➔ **Flink**

Kappa Architecture



Kappa Architecture



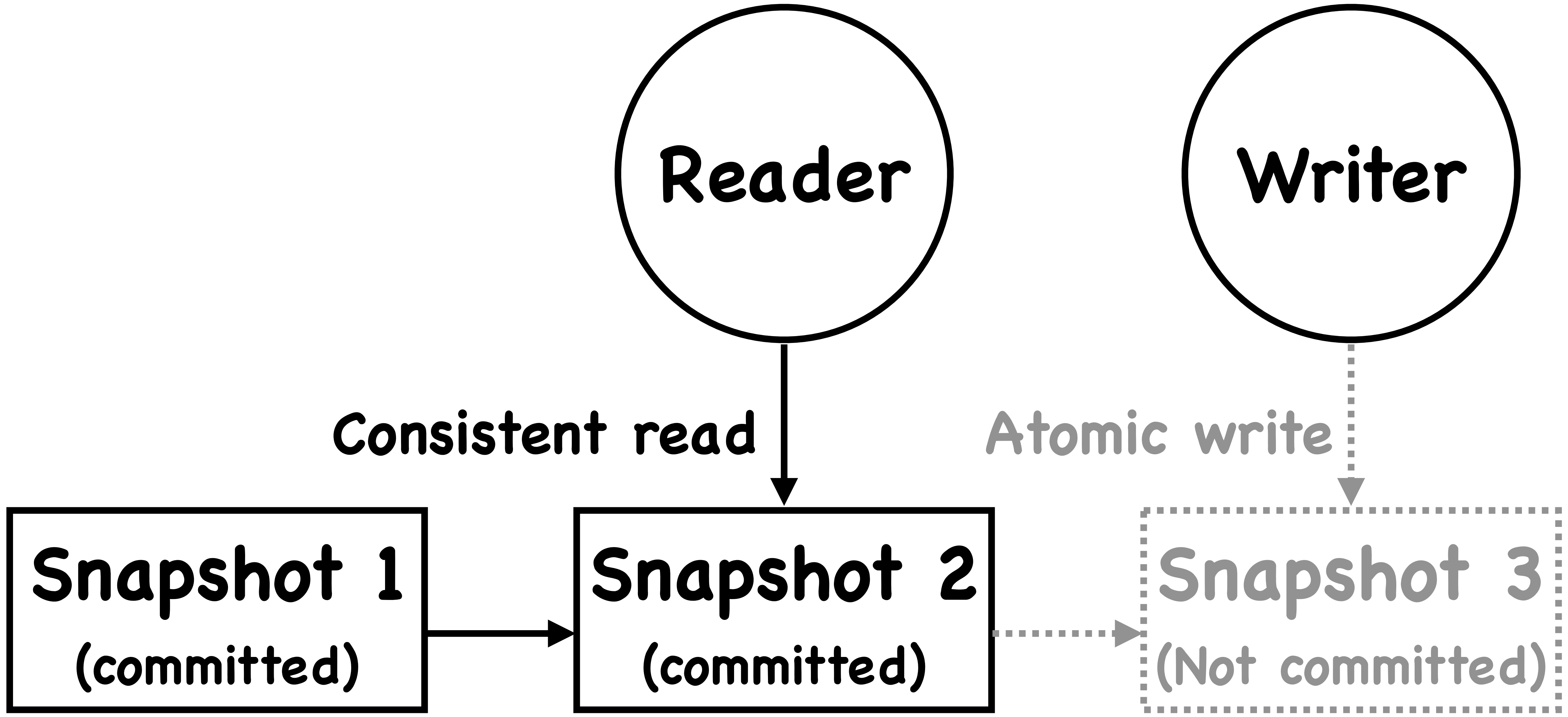
Exactly-once Delivery

- Three conditions for exactly-once delivery
  - Processing engine that supports exactly-once semantics
  - Replayable source (e.g. Kafka)
  - Transactional sink (=Transactional table)

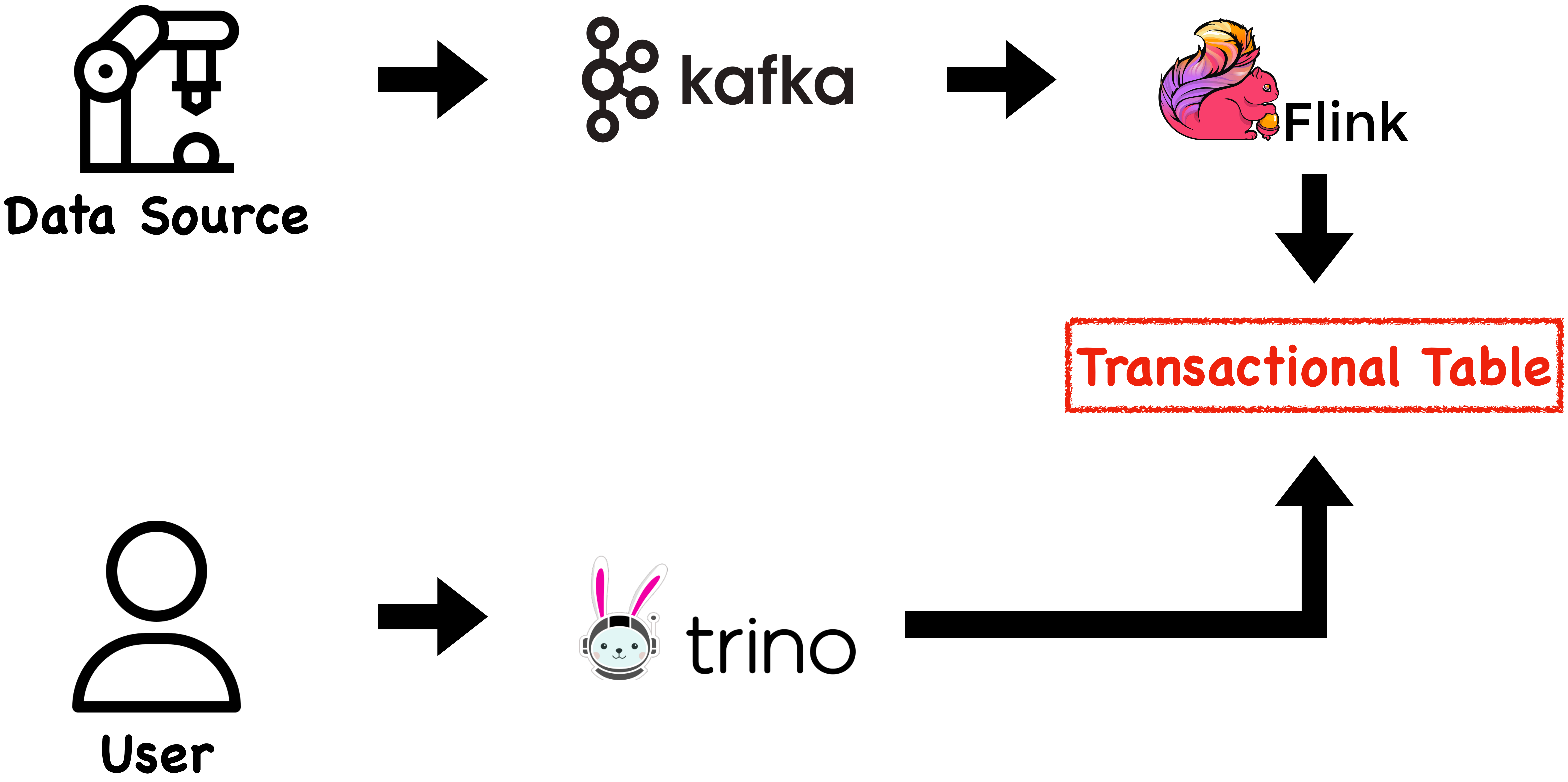
➔ **We need transactional table** (to achieve exactly-once delivery)

Transactional Table

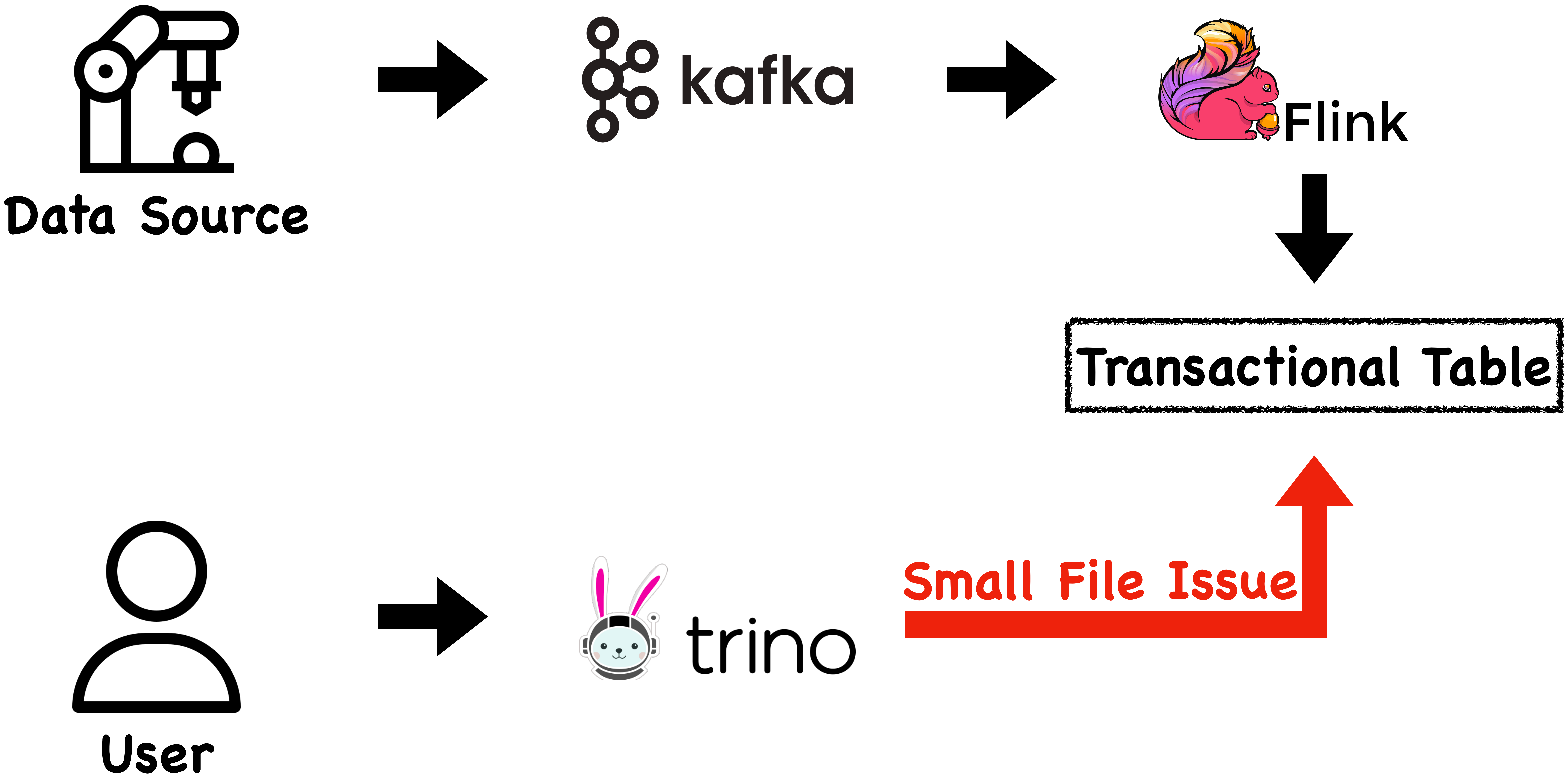
- Snapshot isolation
- Atomic write
- Consistent read



Kappa Architecture



Kappa Architecture



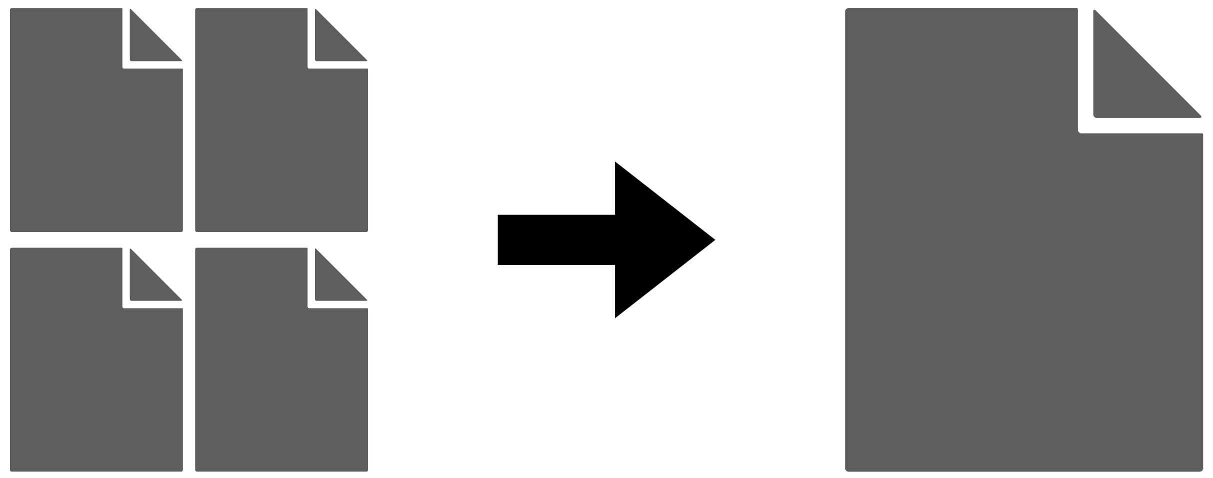
Small File Issue

- Problem occurs when processing a large number of small files
  - Large number of files ➔ High coordinator load
  - Small file size ➔ Ineffective data skipping
- Real-time data accelerates small file issue

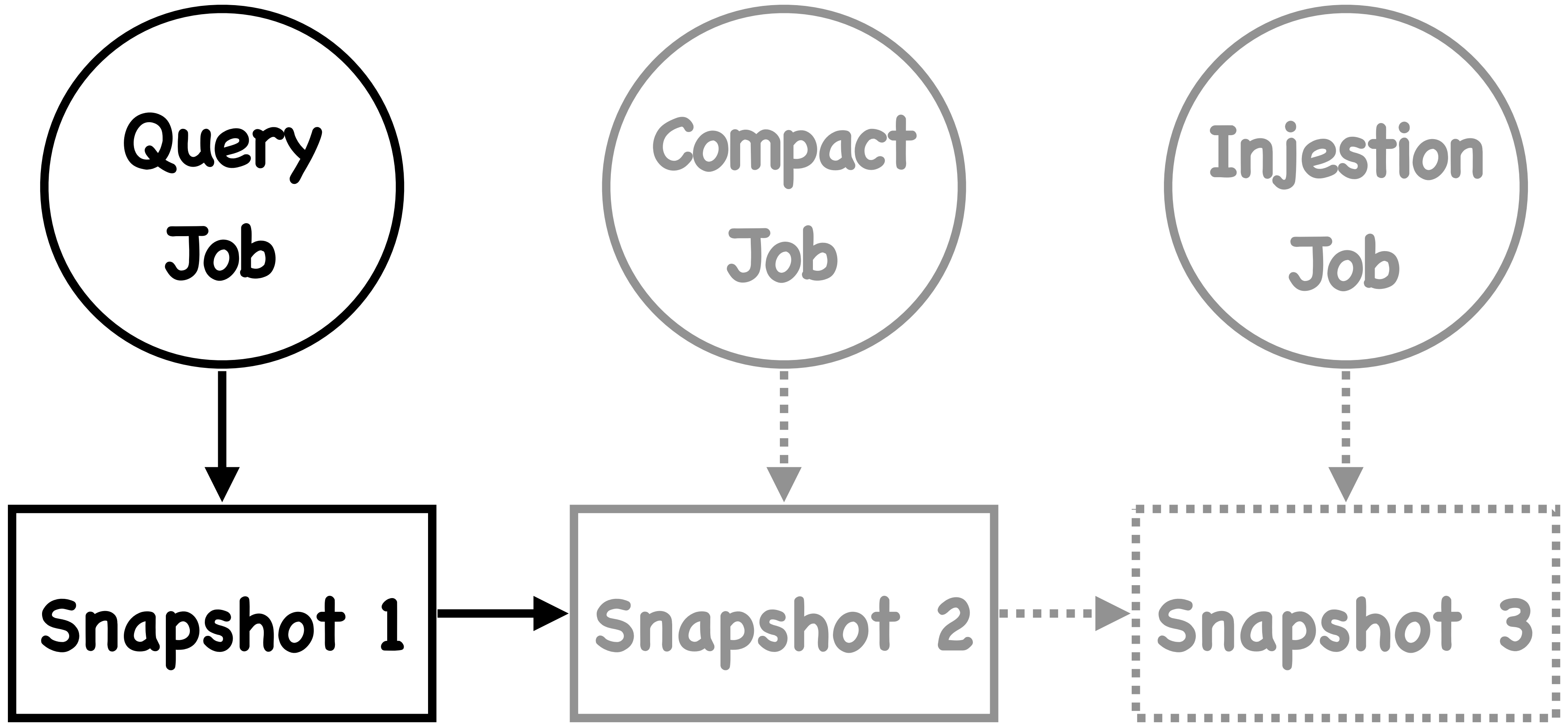
➔ **We need compaction**

Compaction

- Combines small data files into one large file

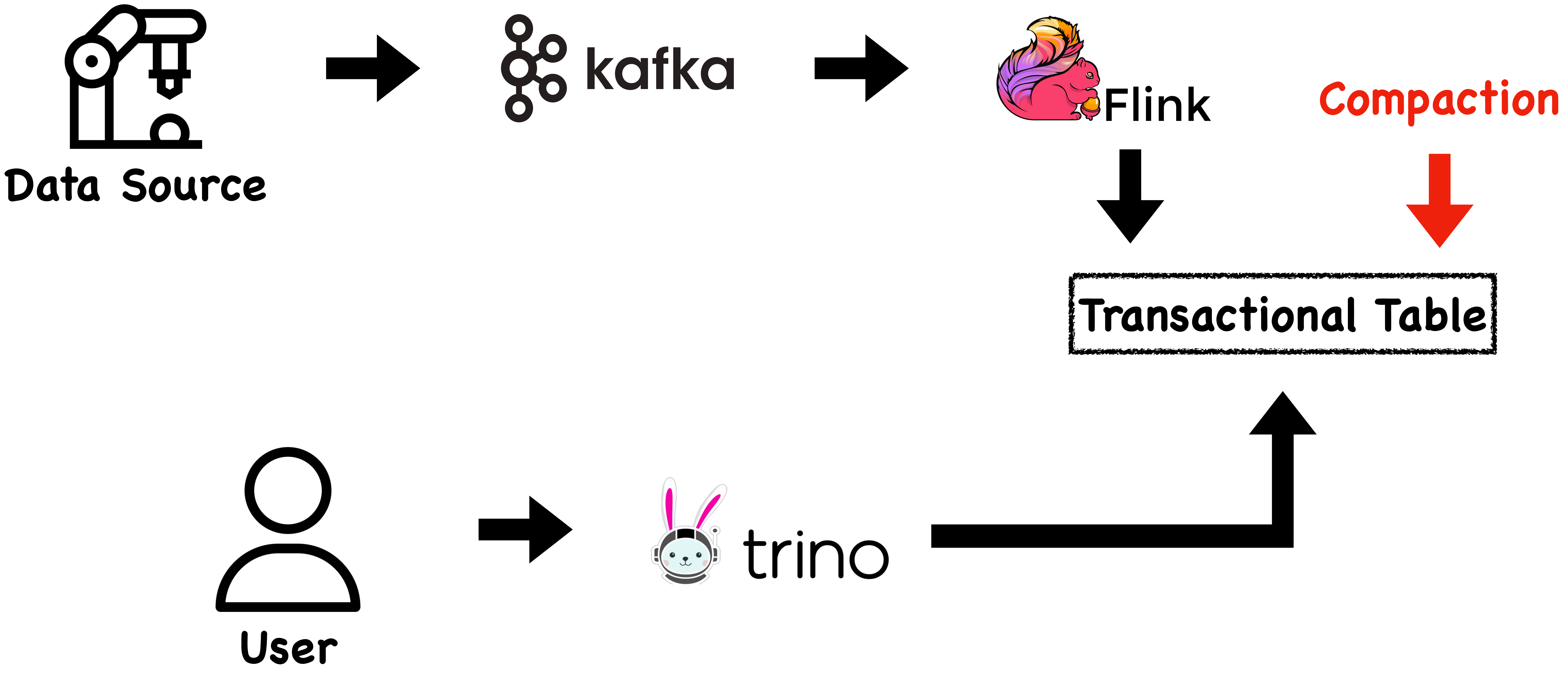


- Transactional table allows jobs to use different snapshots
  - Ingestion job
  - Compaction job
  - Query job

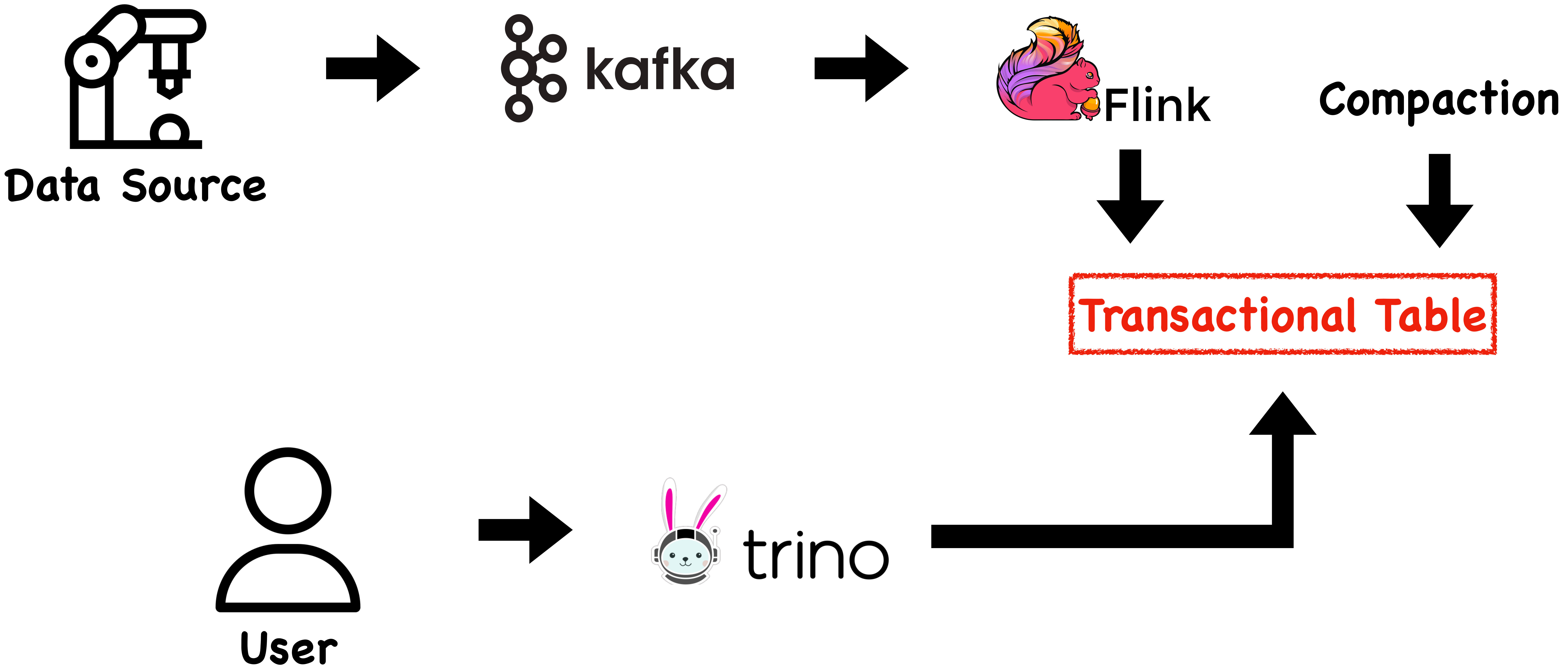


- **Compaction + Transactional table → Solve small file issue**

Kappa Architecture



Kappa Architecture



Transactional Table Formats

HIVE DELTA LAKE Apache hudi ICEBERG

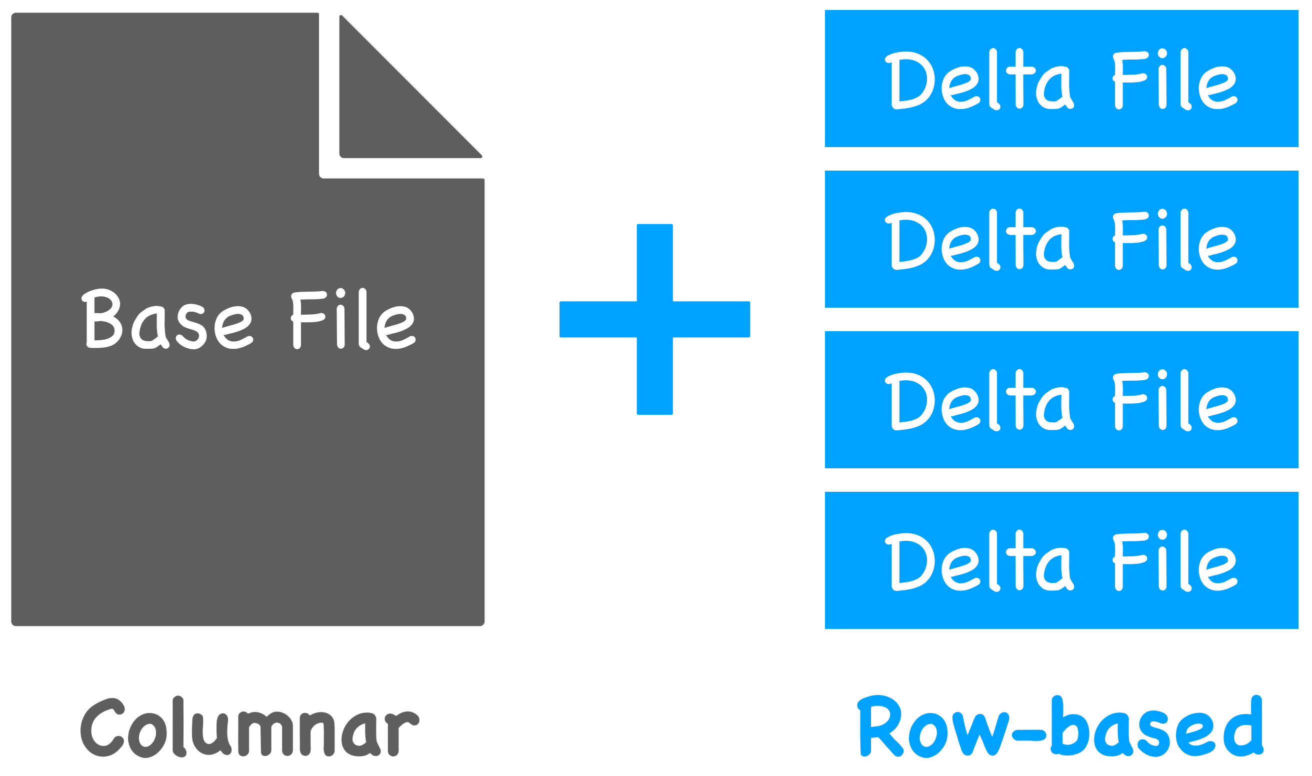
Transactional Table Formats

Apache hudi

ICEBERG

Hudi vs Iceberg

- Hudi provides lower latency (than Iceberg)
  - Columnar base file + Row-based delta file
  - Faster write (append/update)
- Hudi provide auto compaction (that Iceberg does not)
  - No code for compaction
  - No scheduling for compaction jobs

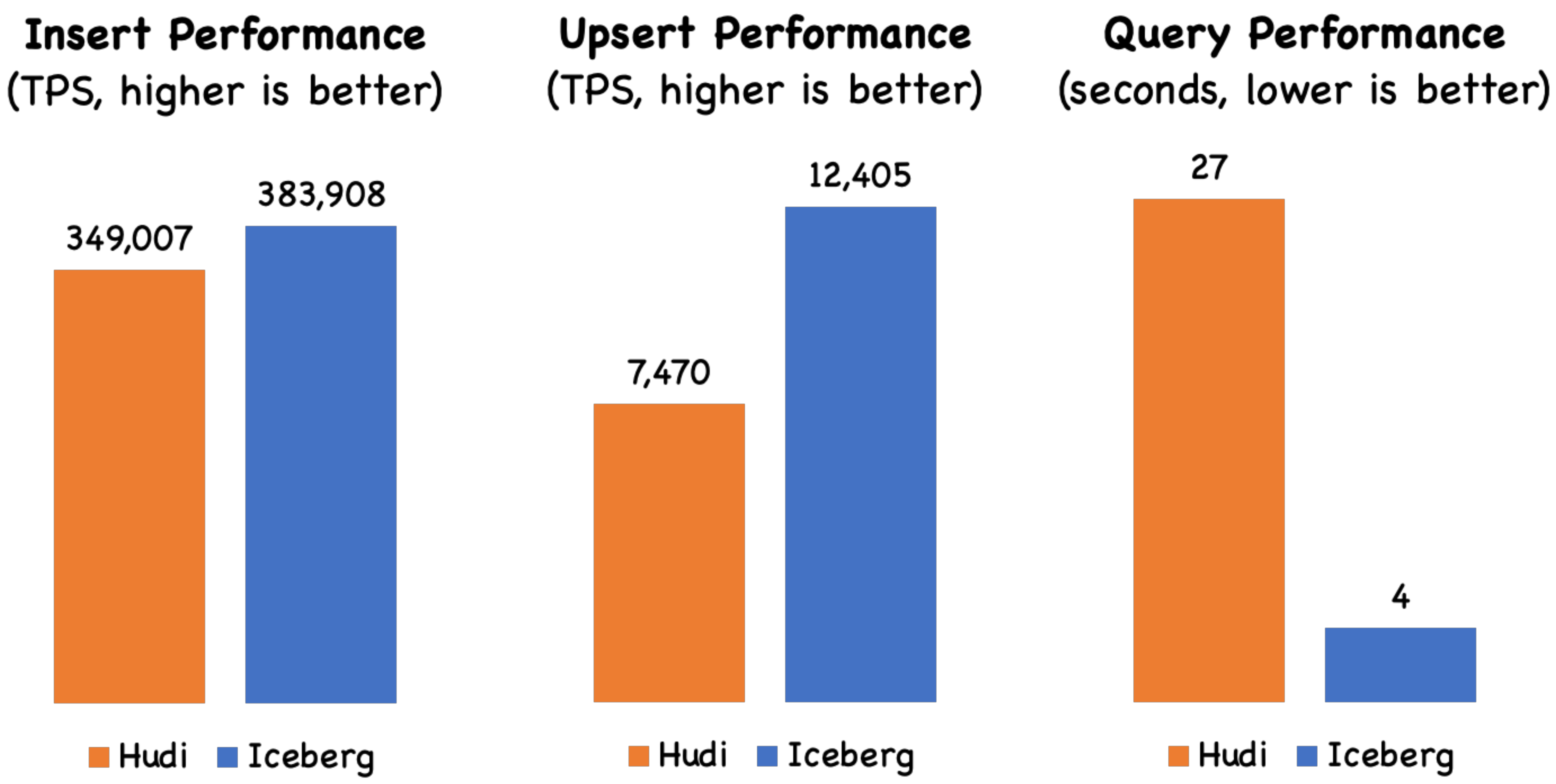


Hudi vs Iceberg

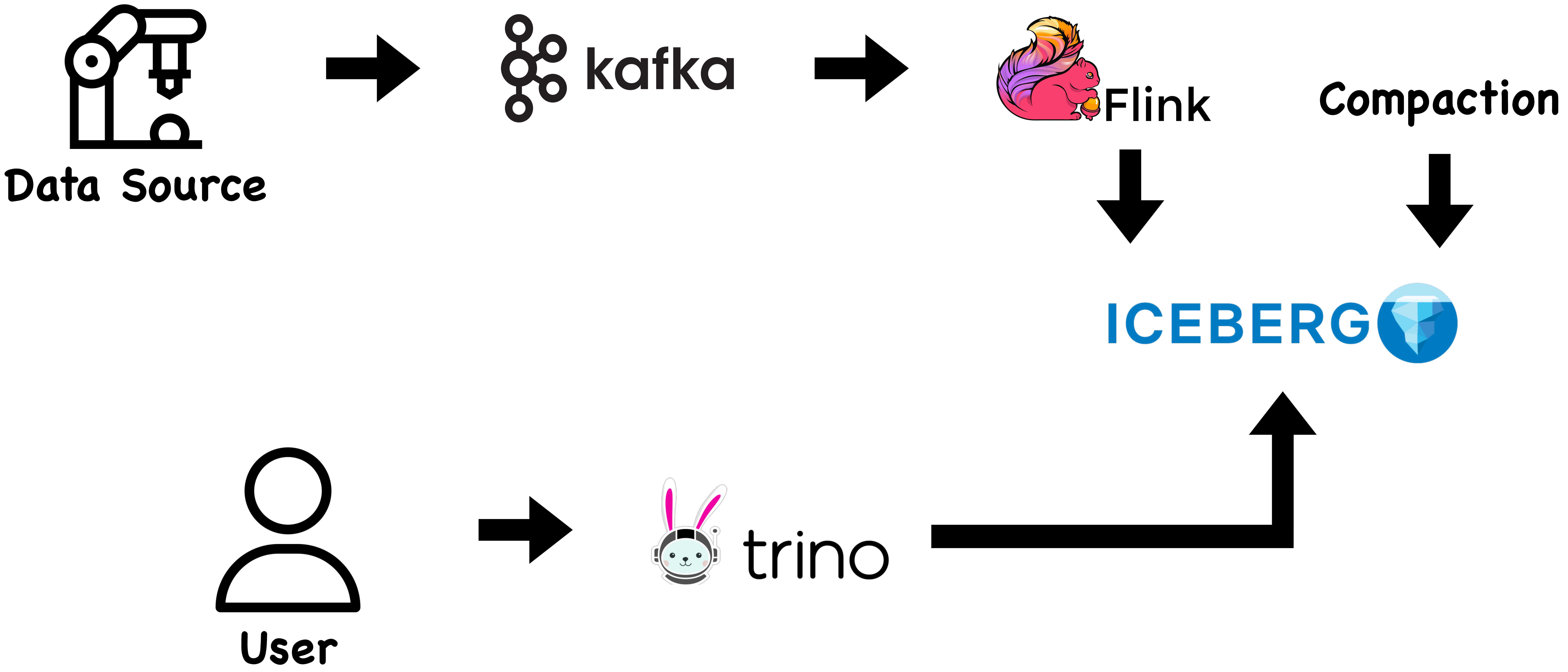
- Trino can not read Hudi's delta files

  ➔ Can not get low latency on Trino

- Hudi had lower performance
  - Insert was 9% slower
  - Upsert was 40% slower
  - Query was 6 times slower



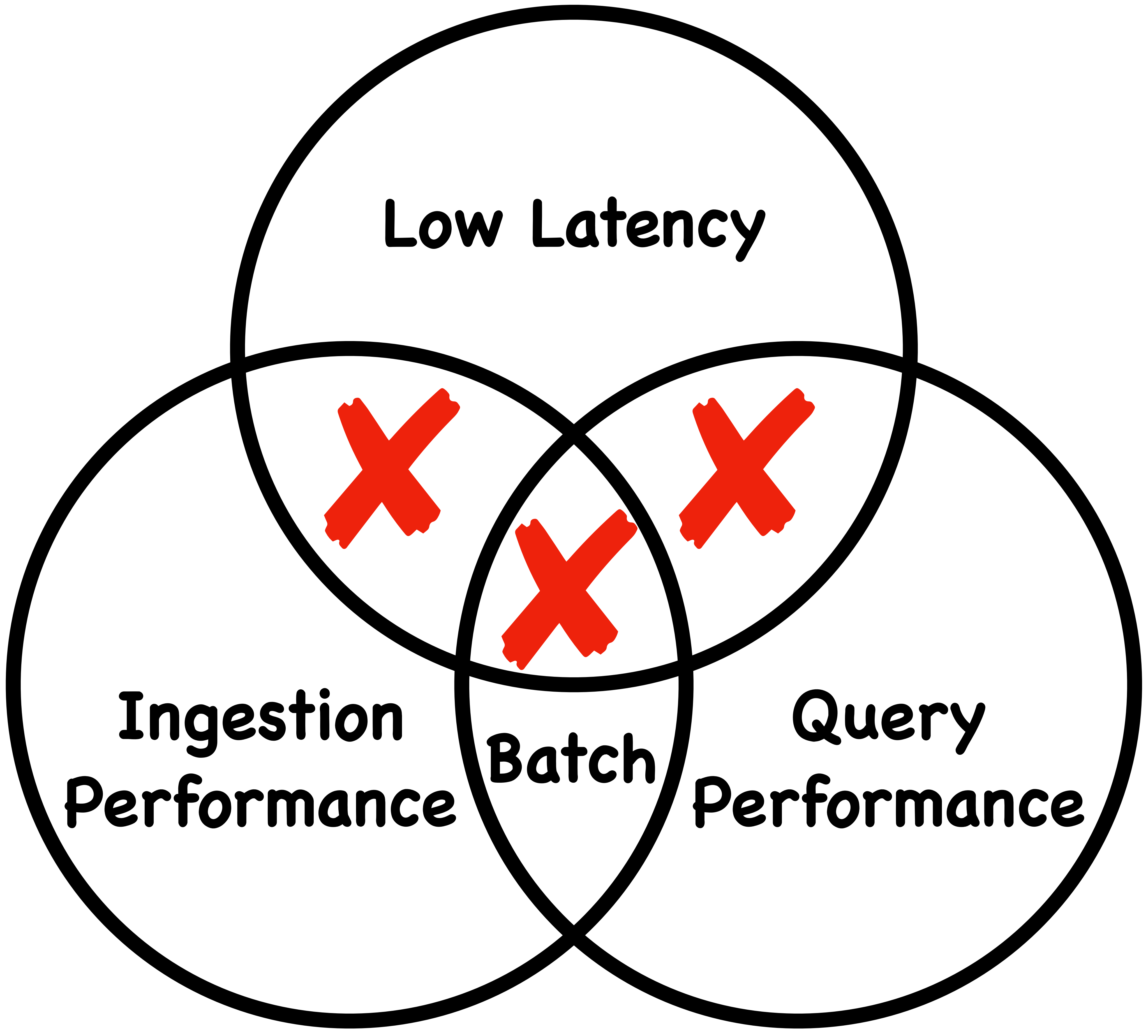
Kappa Architecture



Performance Goals

- Low latency
- High ingestion performance
- High query performance

→ **There is a trade-off here**



Fine Tuning Guidelines

1. Low latency is expensive

2. How to set parallelism

3. How to optimize compaction

4. Why should we expire snapshots

Fine Tuning Guidelines

1. **Low latency is expensive**
2. How to set parallelism
3. How to optimize compaction
4. Why should we expire snapshots

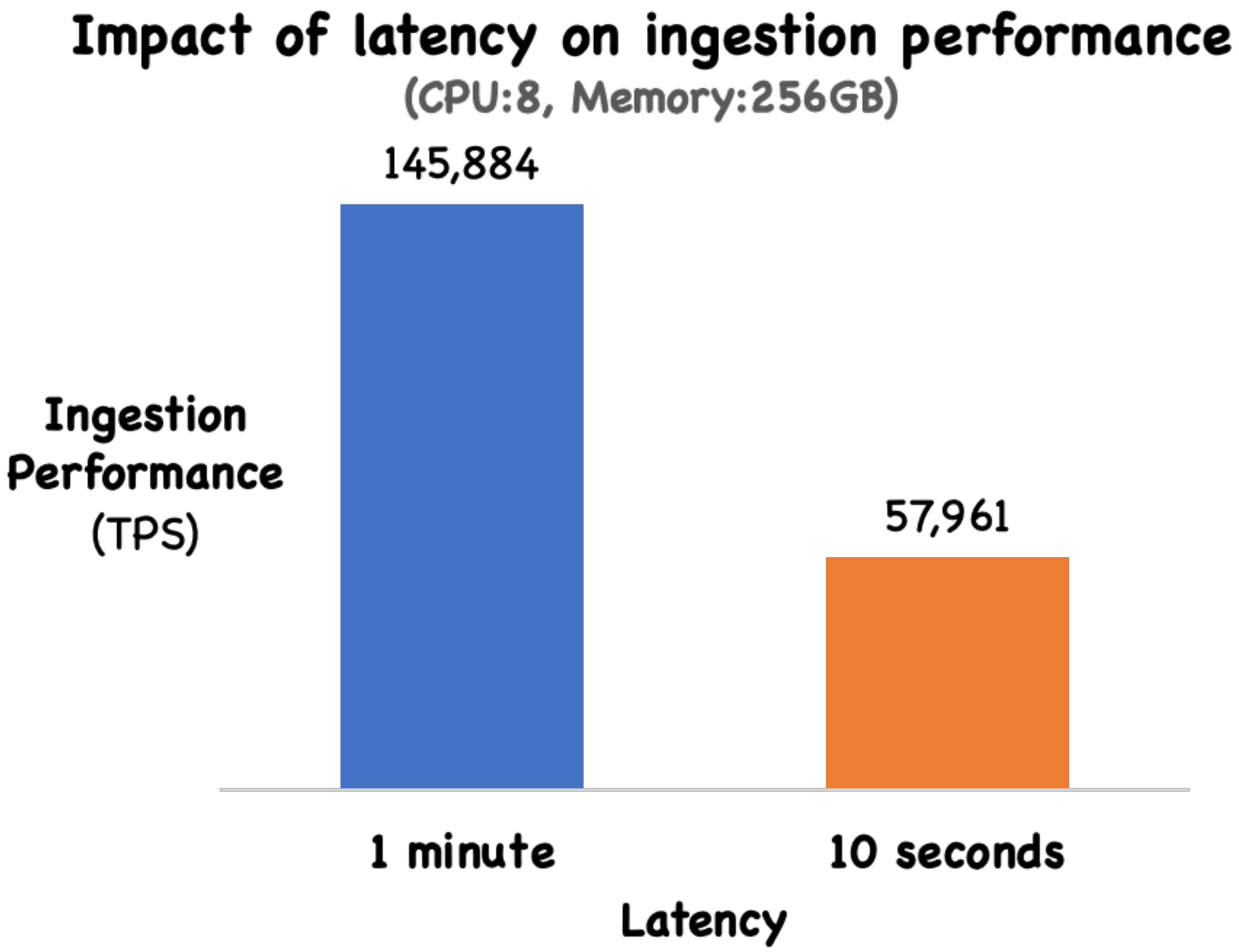
1. Low Latency is Expensive

- What is Flink checkpoint?
  - At each checkpoint, workers commit records
  - Users can only query committed records

➔ **Checkpoint interval == Latency**

1. Low Latency is Expensive

- Costs of low latency
  - Low ingestion performance
  - Small file issue
  - Expensive compaction



➔ Set latency as low as you really need

Fine Tuning Guidelines

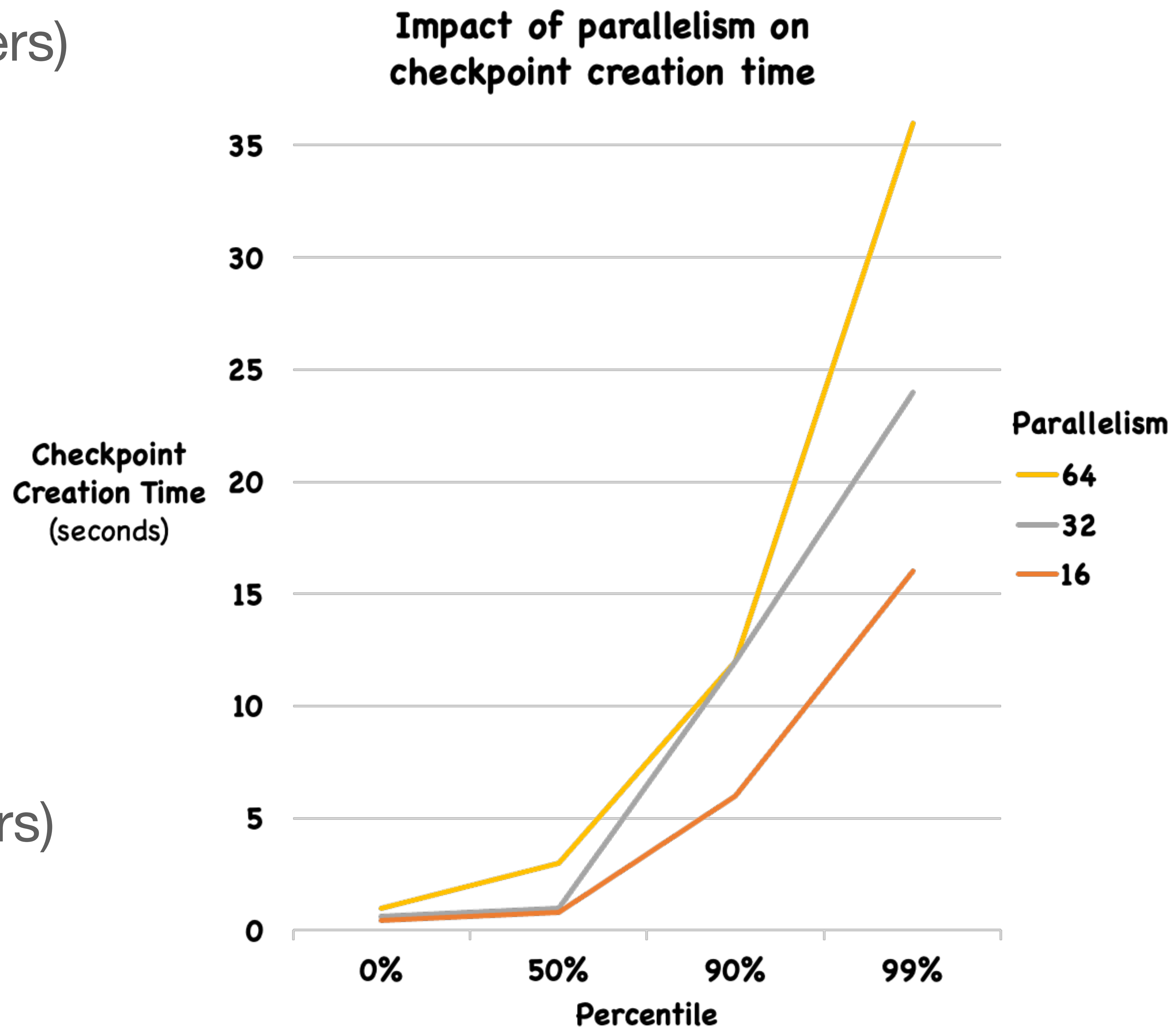
1. Low latency is expensive
2. **How to set parallelism**
3. How to optimize compaction
4. Why should we expire snapshots

2. How to set parallelism

- Large number of small workers? (High parallelism)
- Small number of large workers? (Low parallelism)
- Set equals to the number of Kafka partitions?

2. How to set parallelism

- High parallelism (large number of small workers)
  - High checkpoint creation time
    - ➔ Low ingestion performance
    - ➔ High latency
  - Small file issue
- Low parallelism (small number of large workers)
  - Long failure recovery time



Fine Tuning Guidelines

1. Low latency is expensive
2. How to set parallelism
3. **How to optimize compaction**
4. Why should we expire snapshots

3. How to optimize compaction

- **How compaction works**
  1. Read data file list
  2. Group data files by partition
  3. Re-group data files into file groups (with max file group size)
  4. Read and sort each file group
  5. Write into new data files
  6. Add new Snapshot
  7. Commit

3. How to optimize compaction

- **How to optimize compaction**
  - Enable partial commit (to prevent commit conflict)
  - Apply time-based partition
  - Compact after partition is complete (to prevent commit conflict)

(Continued on next slide)

3. How to optimize compaction

- **How to optimize compaction**
  - Sort data files
    - Do not use default bin-packing
    - Otherwise, file pruning will not work well
  - Choose right sort strategy
    - Basic sort vs Z-order sort
    - Basic sort is better for most use cases (including our case)

Fine Tuning Guidelines

1. Low latency is expensive
2. How to set parallelism
3. How to optimize compaction
4. **Why should we expire snapshots**

4. Why should we expire snapshots

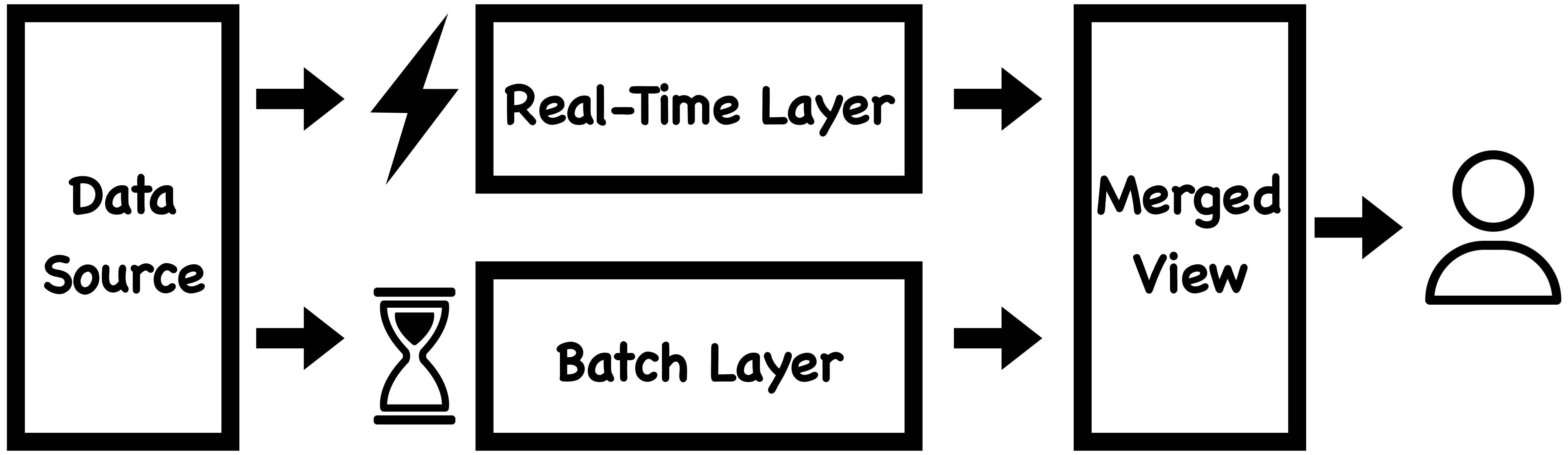
- Checkpoint and compaction job adds a new snapshot
- Too many snapshot cause
  - Large metadata ➔ Reduce query performance
  - Too many unnecessary data files
- We should expire unused snapshots

Let's Recap

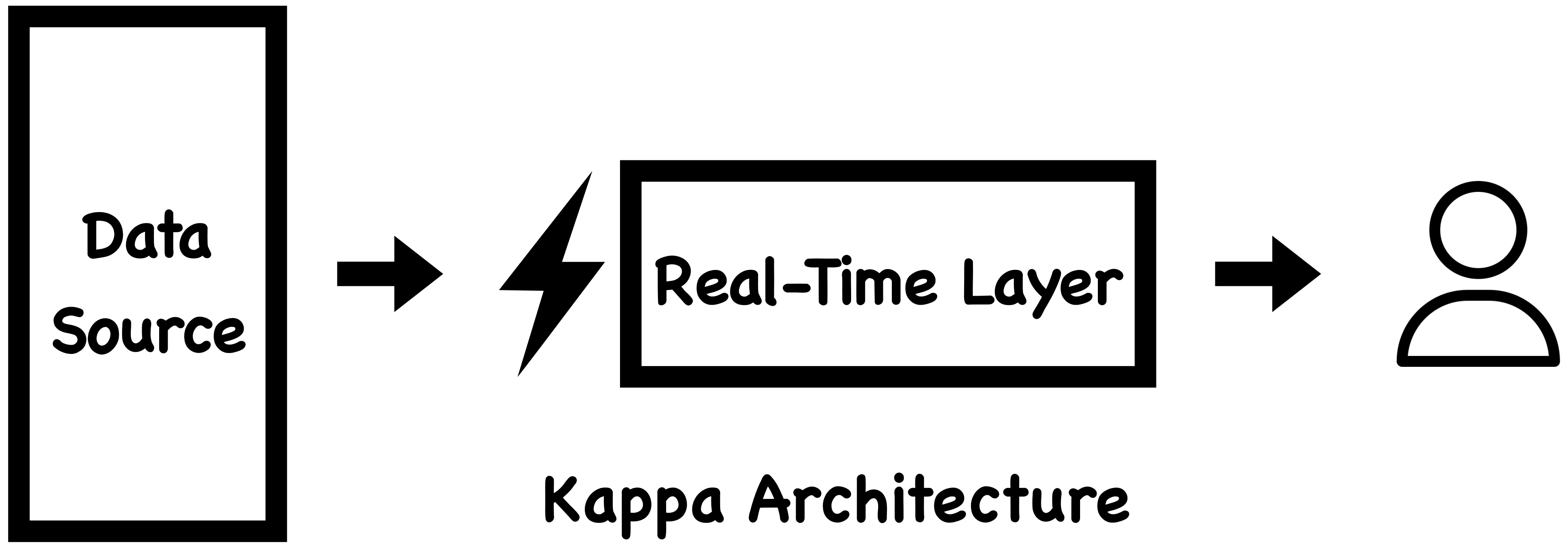
- Lambda vs Kappa
- Trino's Kafka Connector
- Real-time Processing Engine
- Exactly-once Delivery
- Small File Issue ➔ Compaction
- Transactional Table
- Fine Tuning Guidelines

Let’s Recap

- **Lambda vs Kappa**
- Trino’s Kafka Connector
- Real-time Processing Engine
- Exactly-once Delivery
- Small File Issue ➔ Compaction
- Transactional Table
- Fine Tuning Guidelines



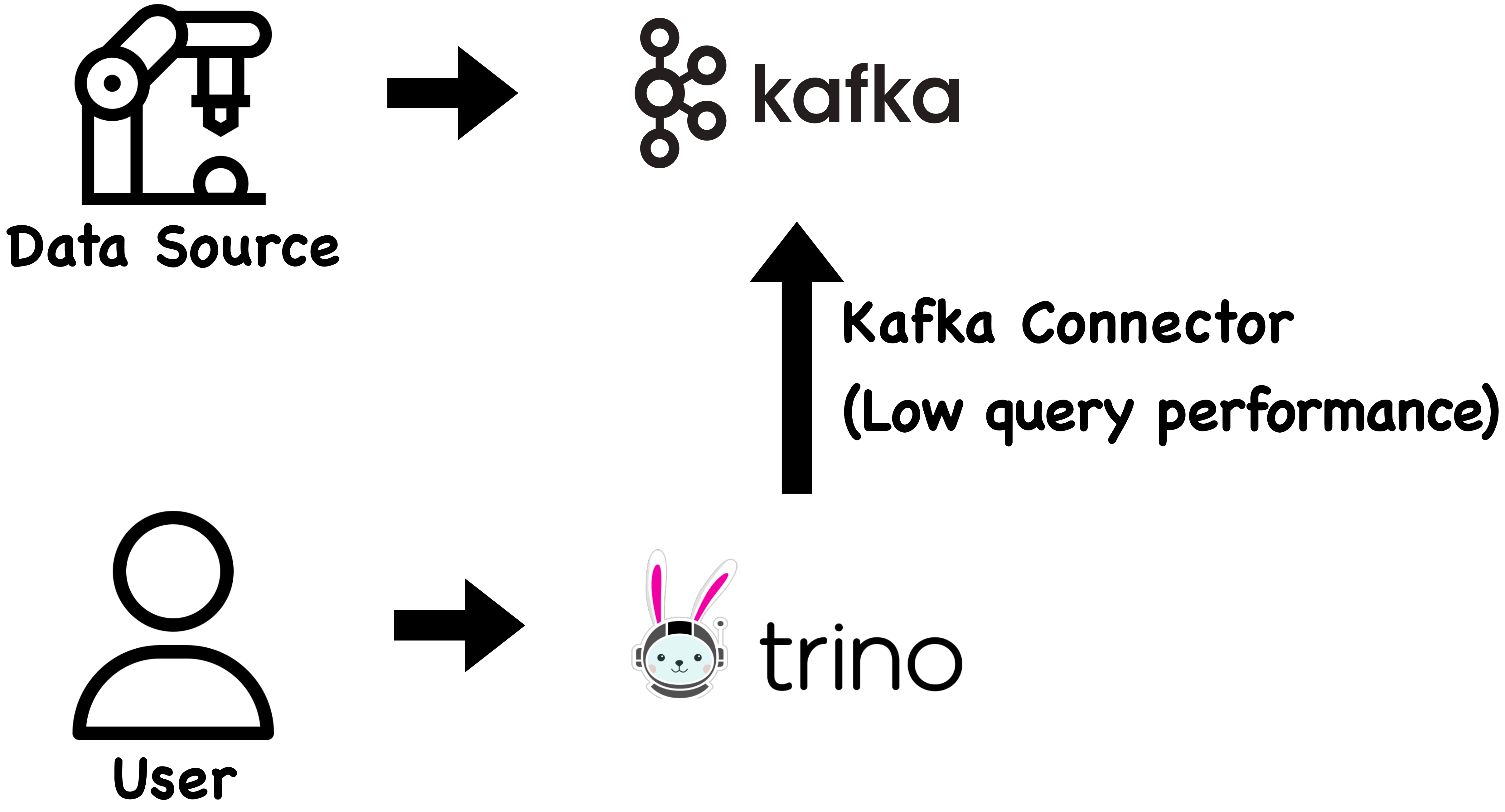
Lambda Architecture



Kappa Architecture

Let's Recap

- Lambda vs Kappa
- **Trino's Kafka Connector**
- Real-time Processing Engine
- Exactly-once Delivery
- Small File Issue ➔ Compaction
- Transactional Table
- Fine Tuning Guidelines



Let's Recap

- Lambda vs Kappa
- Trino's Kafka Connector
- **Real-time Processing Engine**
- Exactly-once Delivery
- Small File Issue ➔ Compaction
- Transactional Table
- Fine Tuning Guidelines

APACHE Spark™

High latency

Basic features

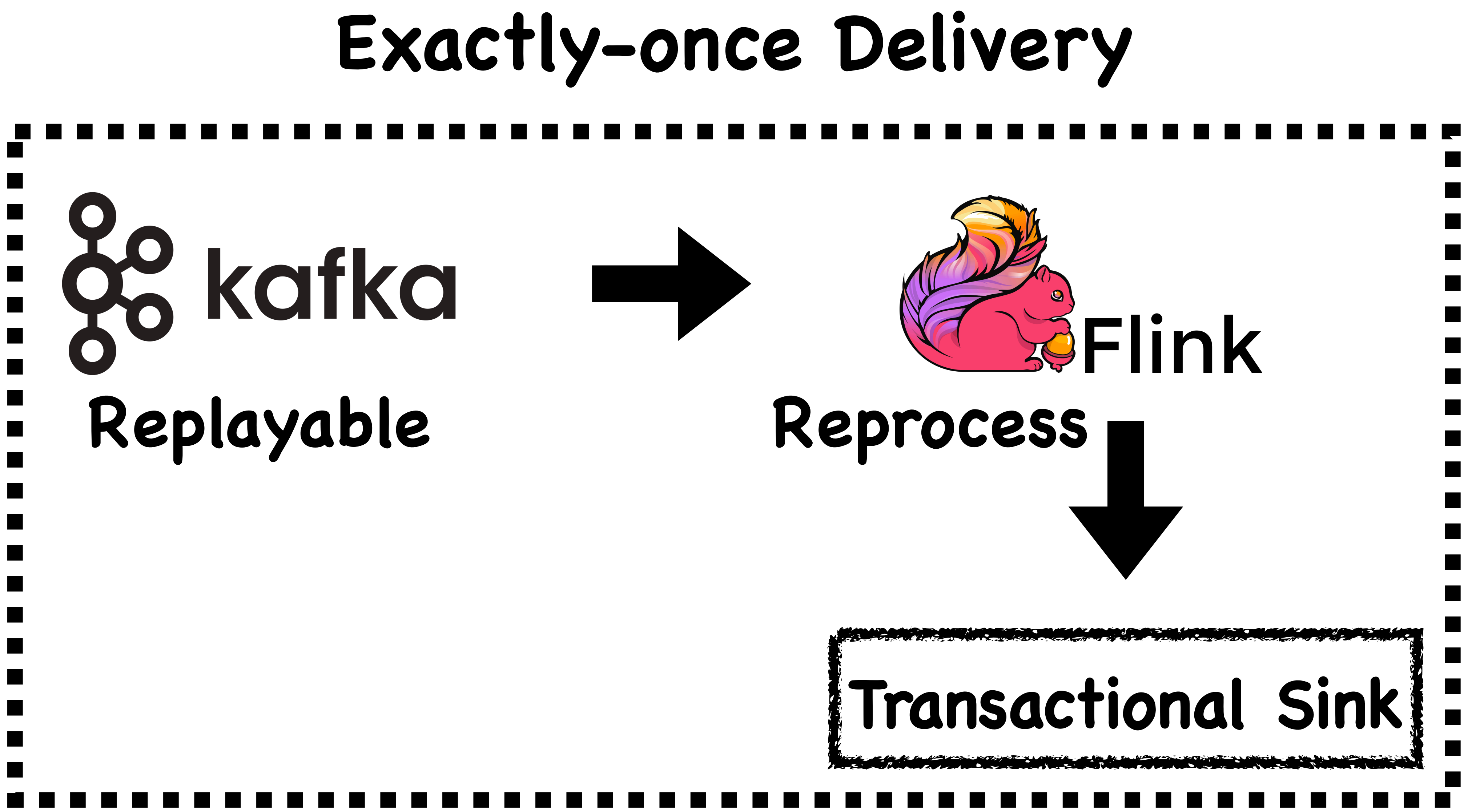
Flink

Low latency

Advanced features

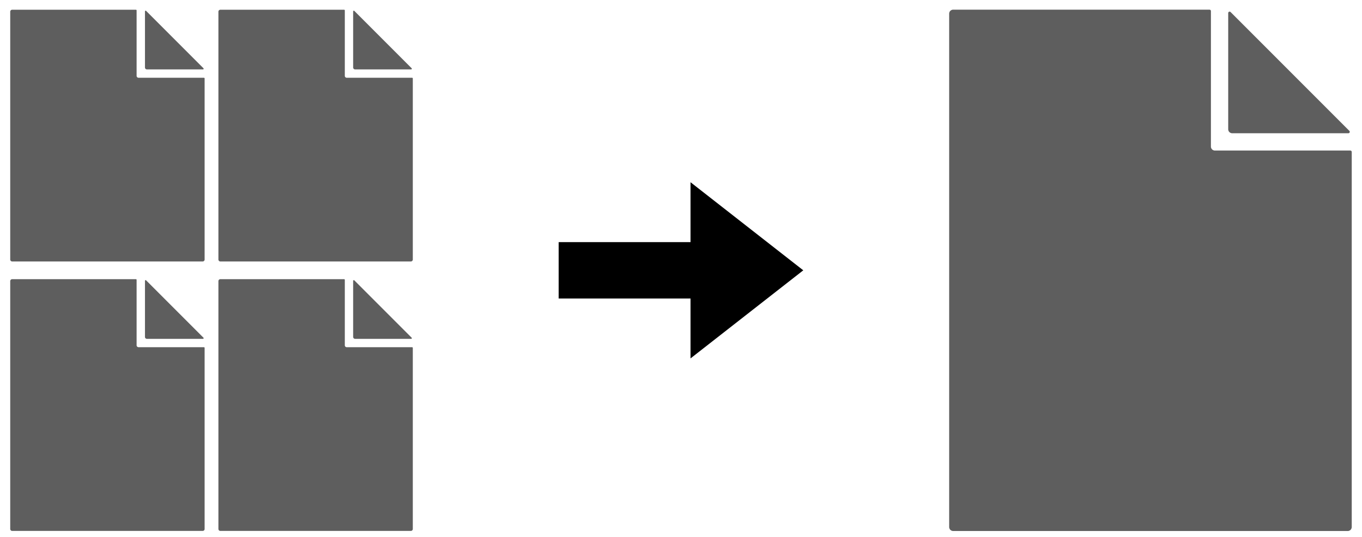
Let's Recap

- Lambda vs Kappa
- Trino's Kafka Connector
- Real-time Processing Engine
- **Exactly-once Delivery**
- Small File Issue ➔ Compaction
- Transactional Table
- Fine Tuning Guidelines



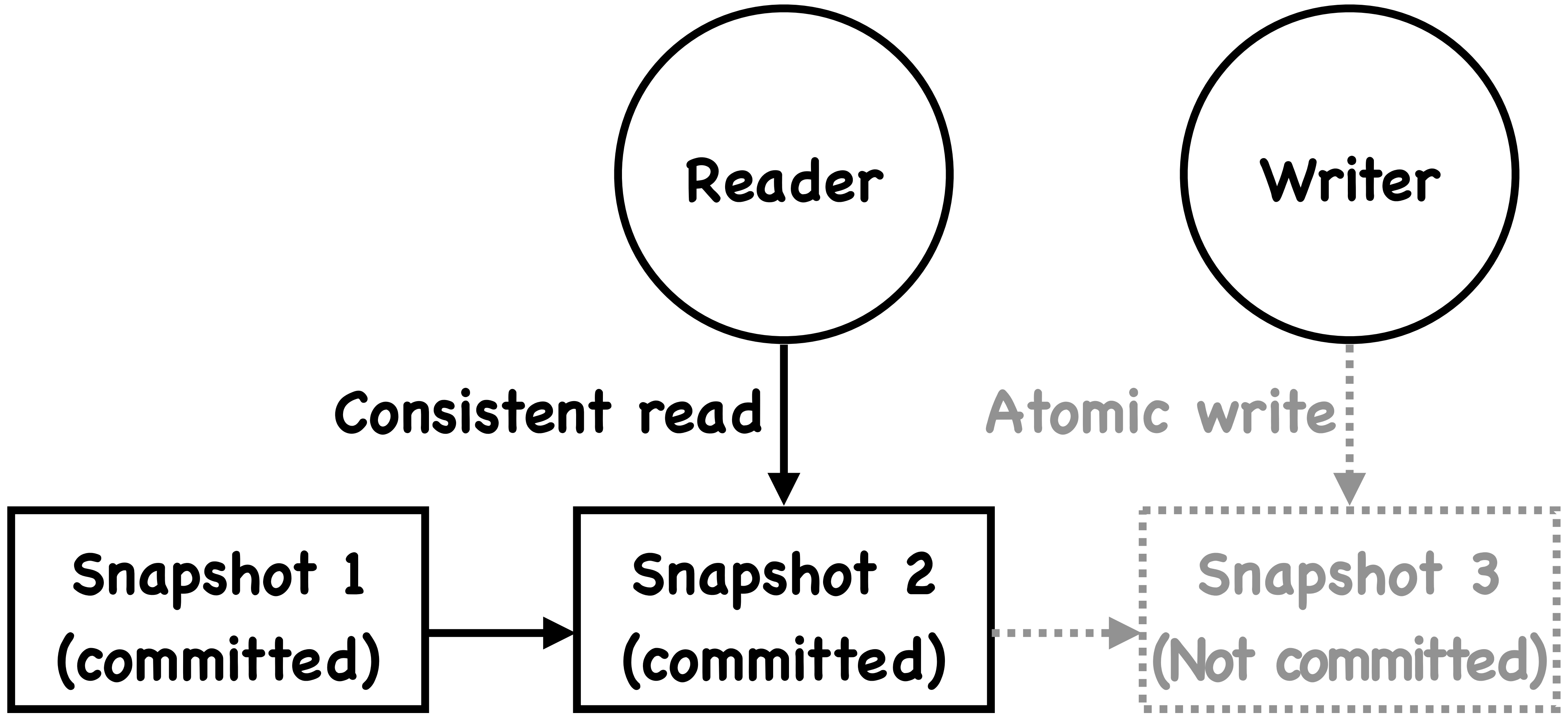
Let’s Recap

- Lambda vs Kappa
- Trino’s Kafka Connector
- Real-time Processing Engine
- Exactly-once Delivery
- **Small File Issue → Compaction**
- Transactional Table
- Fine Tuning Guidelines



Let's Recap

- Lambda vs Kappa
- Trino's Kafka Connector
- Real-time Processing Engine
- Exactly-once Delivery
- Small File Issue ➔ Compaction
- **Transactional Table**
- Fine Tuning Guidelines



Apache hudi

Low performance (Ingestion, Query)

ICEBERG

High performance (Ingestion, Query)

Let's Recap

- Lambda vs Kappa
- Trino's Kafka Connector
- Real-time Processing Engine
- Exactly-once Delivery
- Small File Issue ➔ Compaction
- Transactional Table
- **Fine Tuning Guidelines**

Fine-tuning Guidelines

1. Low latency is expensive
2. How to set parallelism
3. How to optimize compaction
4. Why should we expire snapshots

Performance Test Results

- **Ingestion performance**
  - Parallelism : 60
  - CPU : 60
  - Memory : 180GB
  - TPS : 1M
- **Query performance**
  - Trino Worker : 20
  - Count 2B : 4.6s
  - Aggregate 2B : 3.6s

Q & A

shyun9417@sk.com