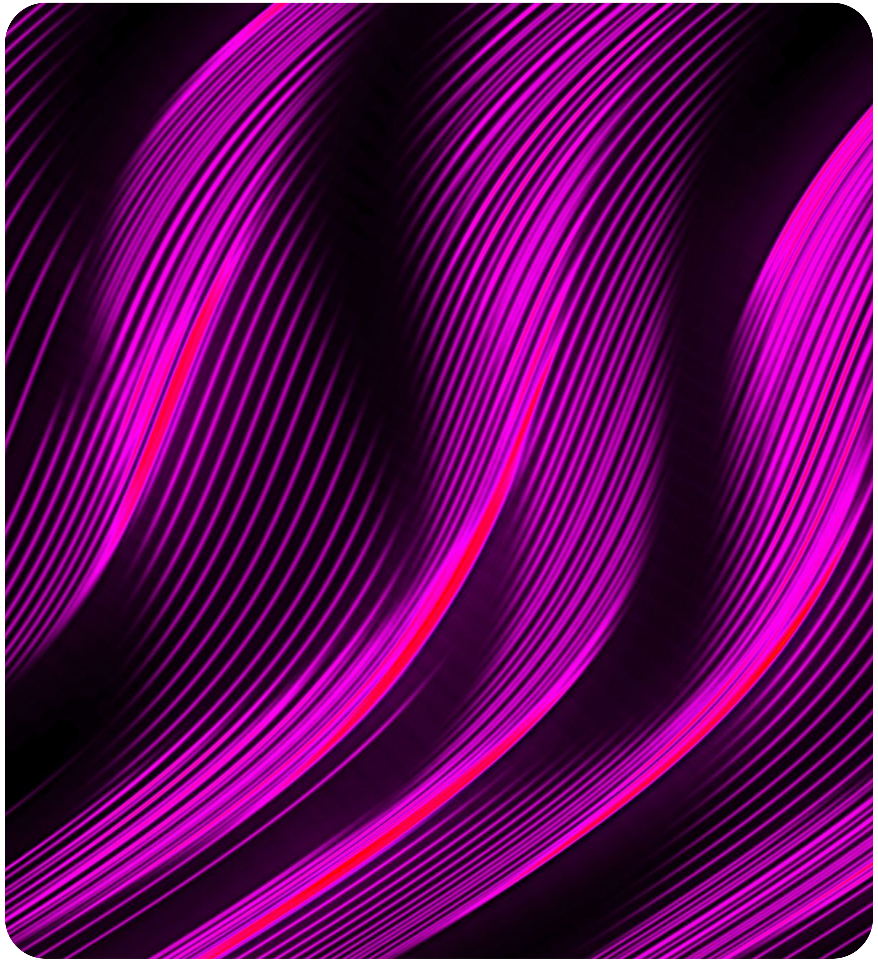
RTB HOUSE

25.03.2024

# Stream processing - part II

Piotr Jaczewski
**RTB HOUSE**



**Stream processing - part II**

# Agenda

1. intro
2. advantages of stream processing
3. basic concepts
4. design patterns
5. fault tolerance
6. lambda architecture

# What is stream processing?

“[stream processing is] a type of data processing engine that is designed with infinite data sets in mind” - Tyler Akidau

infinite == unbounded, always growing

everything is a stream

# Attributes of a data stream

Streams are:

- ordered
- immutable
- replayable

# Request-response vs batch vs stream

## Request-response

- low latency
- often blocking
- also known as OLTP - online transaction processing

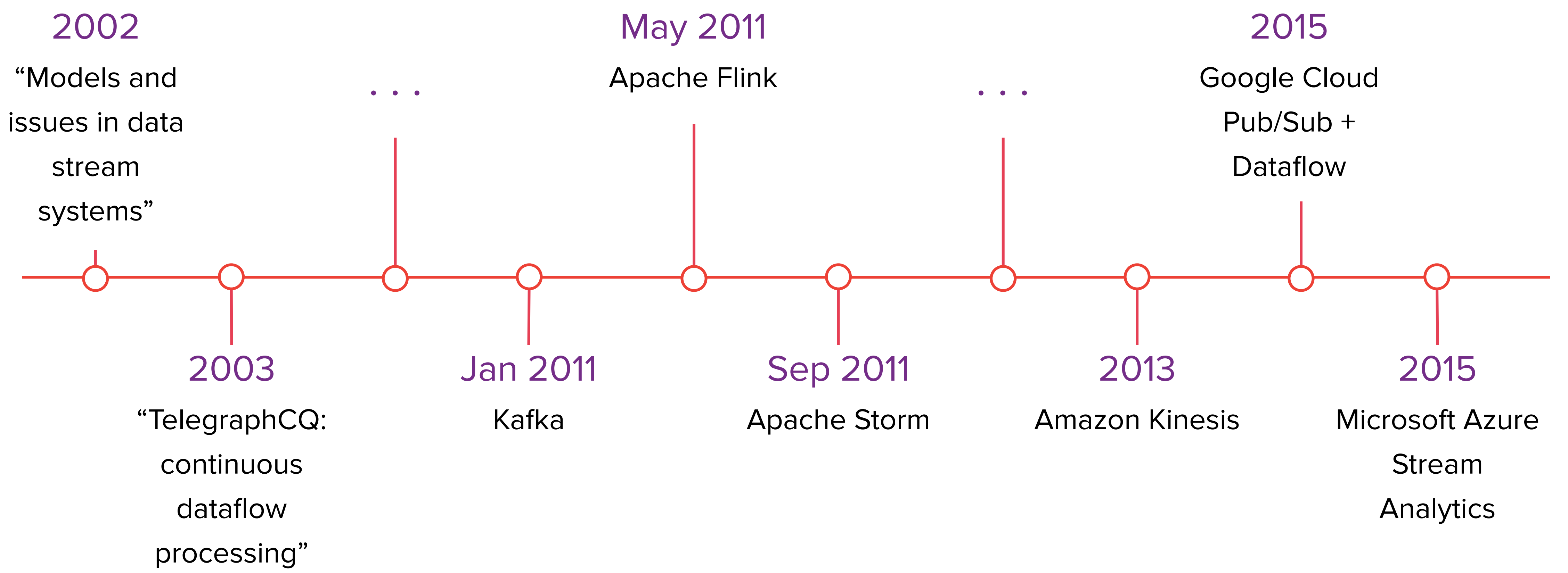
## Batch

- high latency
- high throughput
- great efficiency
- stale data

## Stream

- nonblocking
- reasonably low latency
- continuous and ongoing

# Historical timeline



# Typical use cases

1. real-time analytics
2. fraud detection
3. Internet of Things
4. log processing
5. financial trading
6. supply chain and logistics

# Real-time analysis

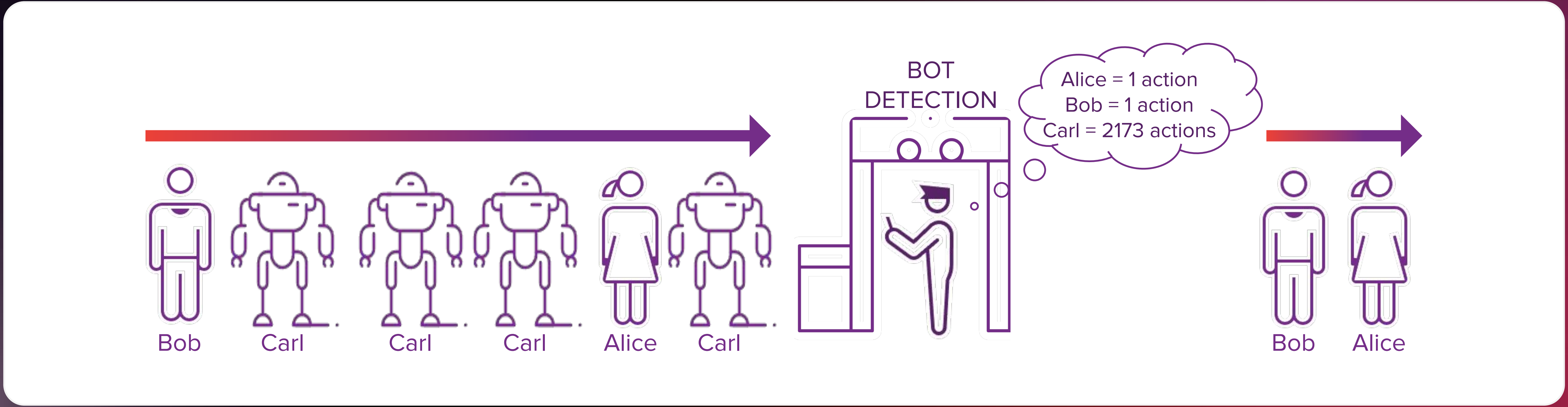
- faster decision making
- faster reaction to market fluctuations
- easier detection and addressing of issues
- improved customer experience
- increased agility and optimization

# Practical example: bot detector

certain bots are easy to detect (they make hundreds of actions per minute)

but difficult to eliminate (they live only for several minutes)

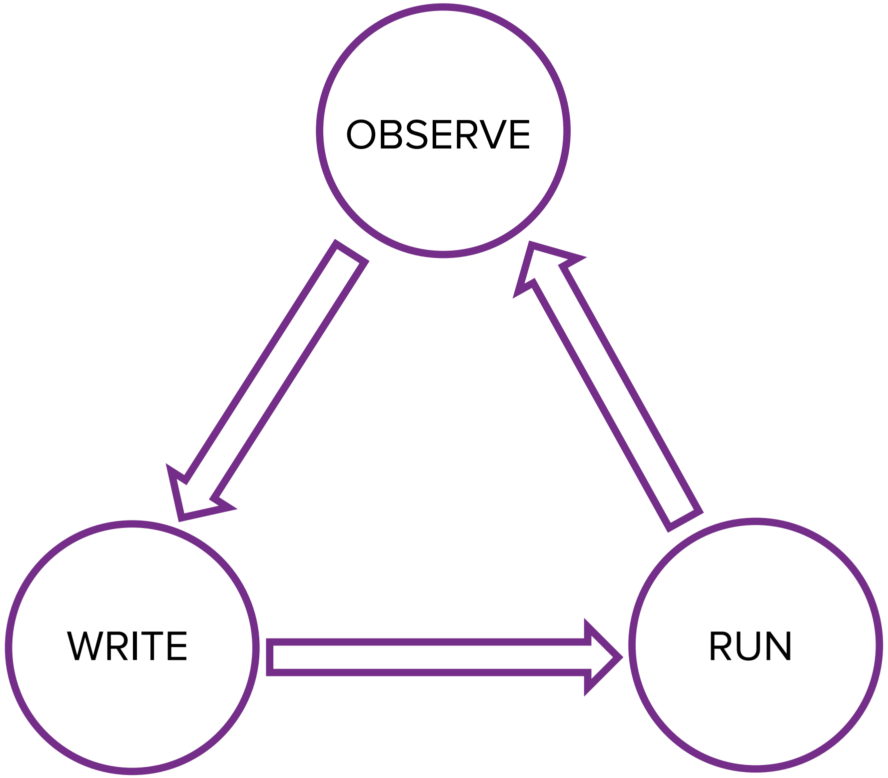
we can scan a stream of user actions and filter out users with too many actions



RTB HOUSE

# Short feedback loop when coding

- faster development
- better productivity



RTB HOUSE

# Is stream processing always the best approach?

NO

# Stream processing concepts

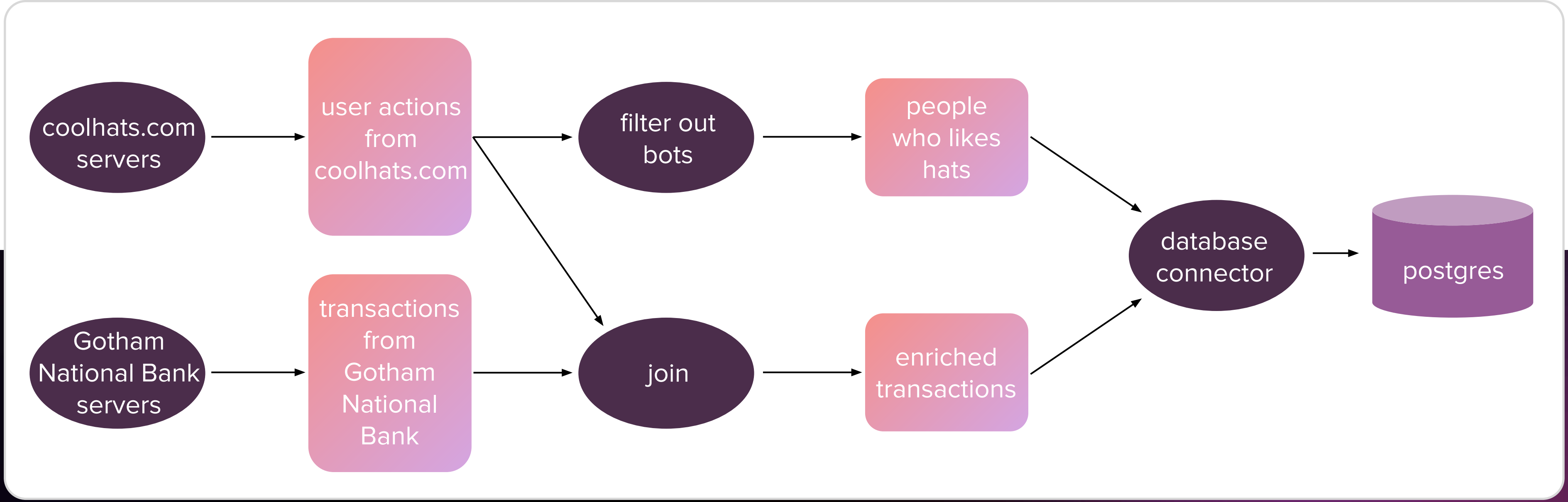
"

A **stream** is a continuous body of surface water[1] flowing within the bed and banks of a channel. Depending on its location or certain characteristics, a stream may be referred to by a variety of local or regional names. Long large streams are usually called rivers, while smaller, less voluminous and more intermittent streams are known as streamlets, brooks or creeks, etc.

The flow of a stream is controlled by three inputs – surface runoffs (from precipitation or meltwater),

Aubach (Wiehl) in North Rhine-Westphalia, Germany

# Topology

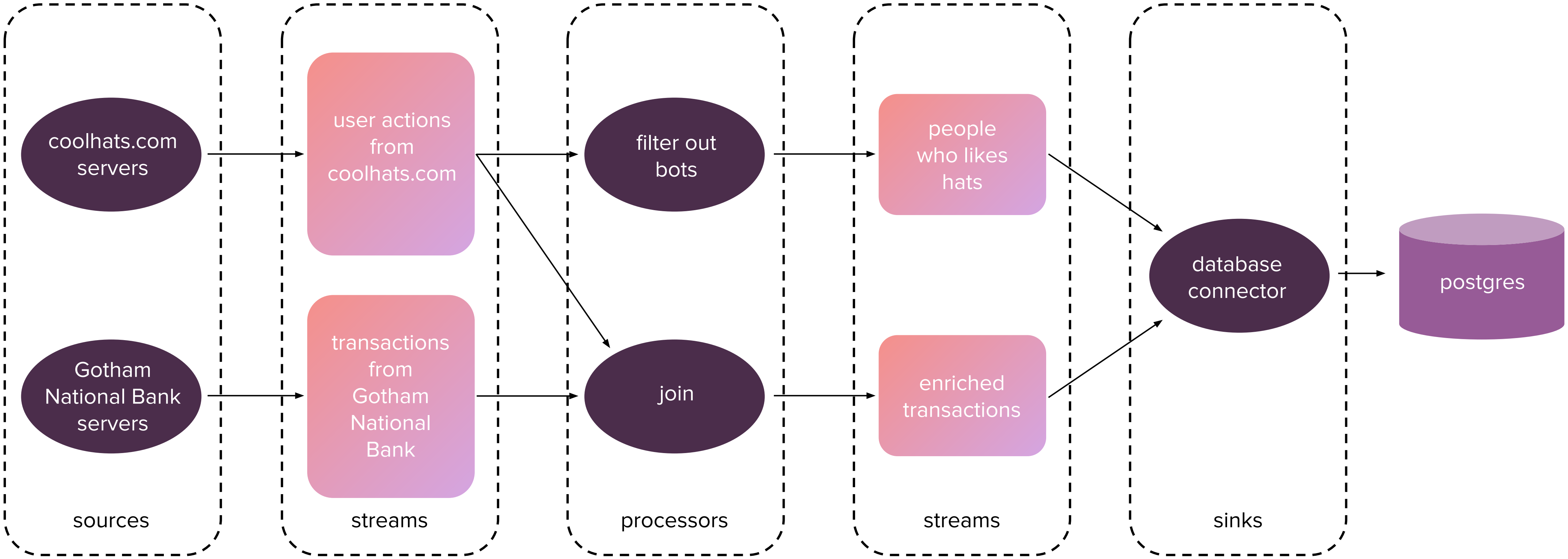



- logical abstraction for your stream processing code

- acyclic graph of sources, processors, and sinks

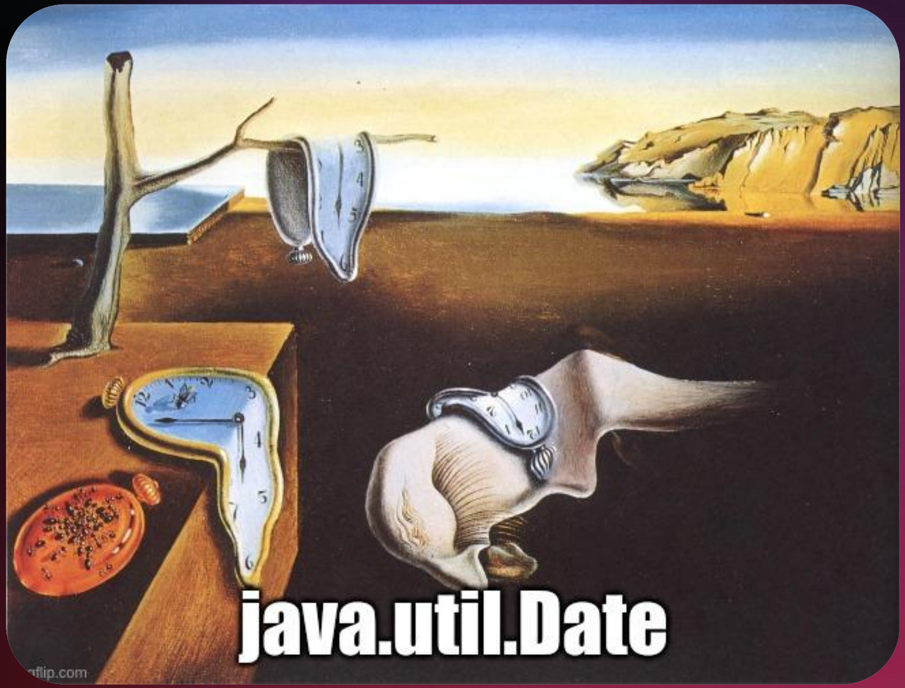
RTB HOUSE

# Topology



# Time

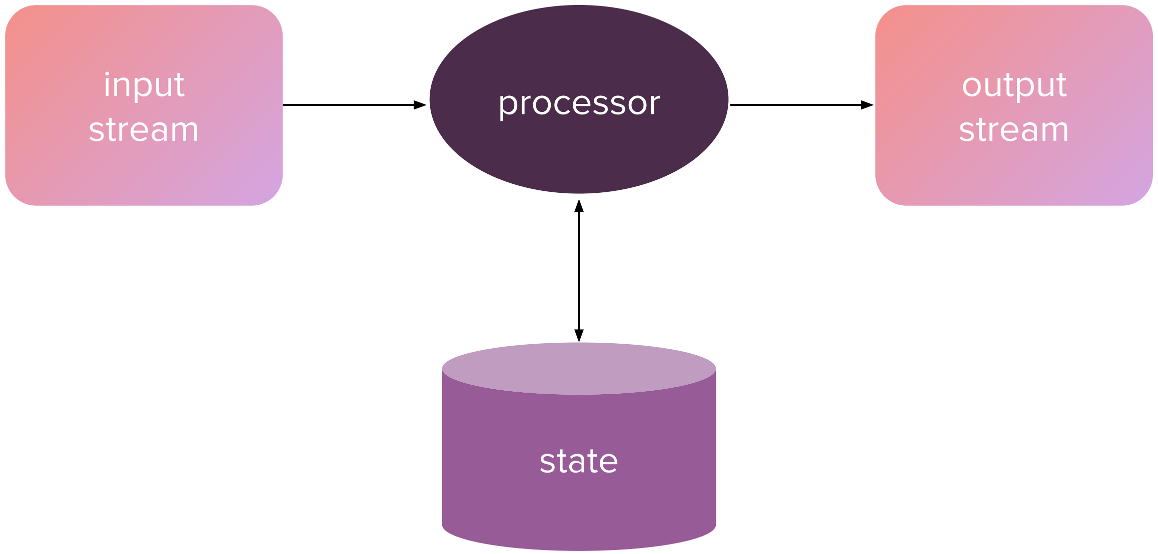
- ingestion time
- event time
- processing time
- sometimes even more (our record: 20 timestamps in a single event)



RTB HOUSE =

# State

- used to keep track of information about the events
- local: accessible only by a specific instance, within the application
- external: global; outside the application



# Stream - table duality

TABLE

| id | balance |
|---|---|
| 101 | 1200 |
| 102 | -300 |

UPDATE ... SET balance = 200 WHERE id = 101;
UPDATE ... SET balance = 3000 WHERE id = 101;
UPDATE ... SET balance = 700 WHERE id = 102;
UPDATE ... SET balance = 1200 WHERE id = 101;
UPDATE ... SET balance = -300 WHERE id = 102;

STREAM

| | |
|---|---|
| 101 | 200 |
| 101 | 3000 |
| 102 | 700 |
| 101 | 1200 |
| 102 | -300 |

- a table is just a stream of updates

- a stream is just the change of a table over time

# Stream - table duality

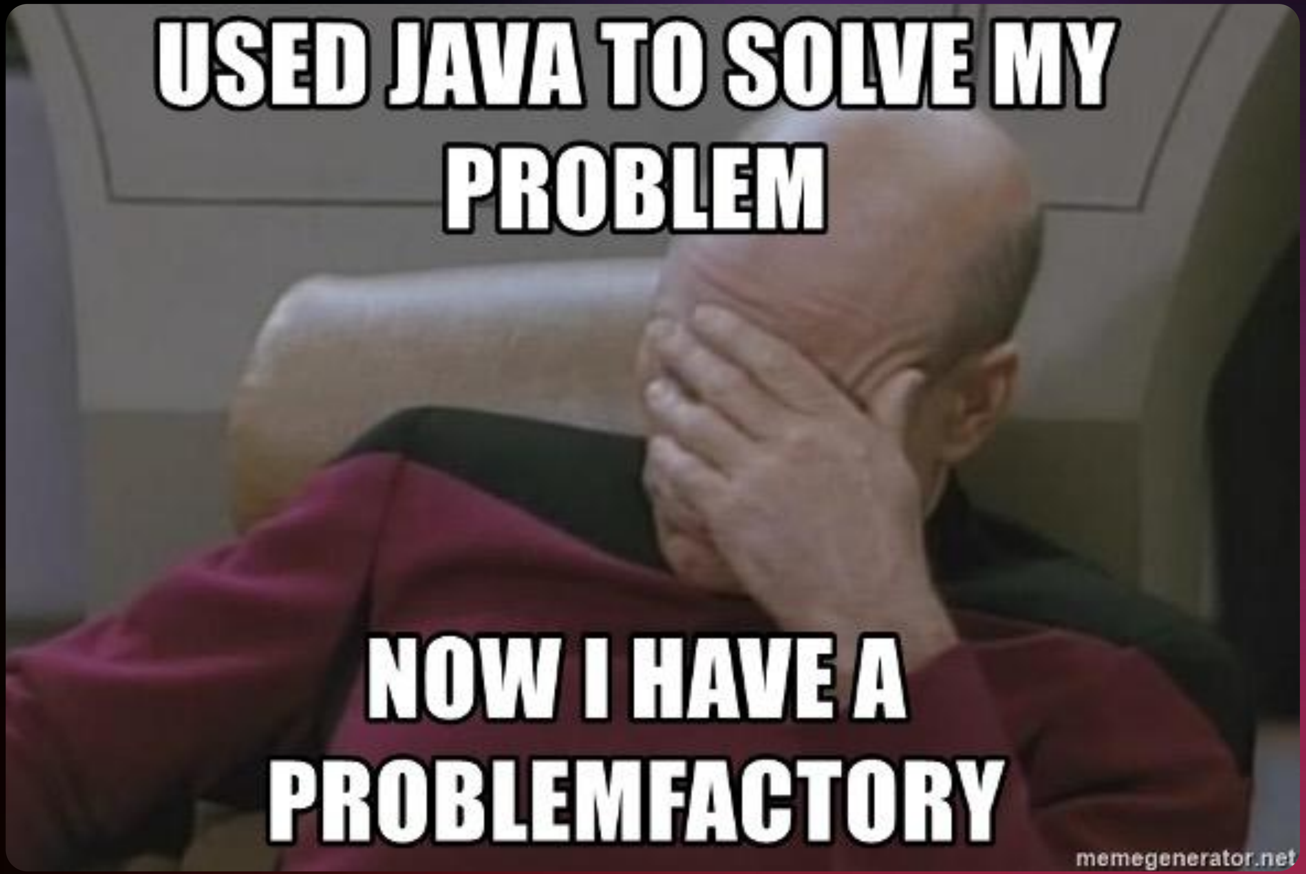
$$state(now) = \int_{t=0}^{now} stream(t)\ \mathrm{d}t \qquad stream(t) = \frac{\mathrm{d}\ state(t)}{\mathrm{d}t}$$

# Processing guarantees

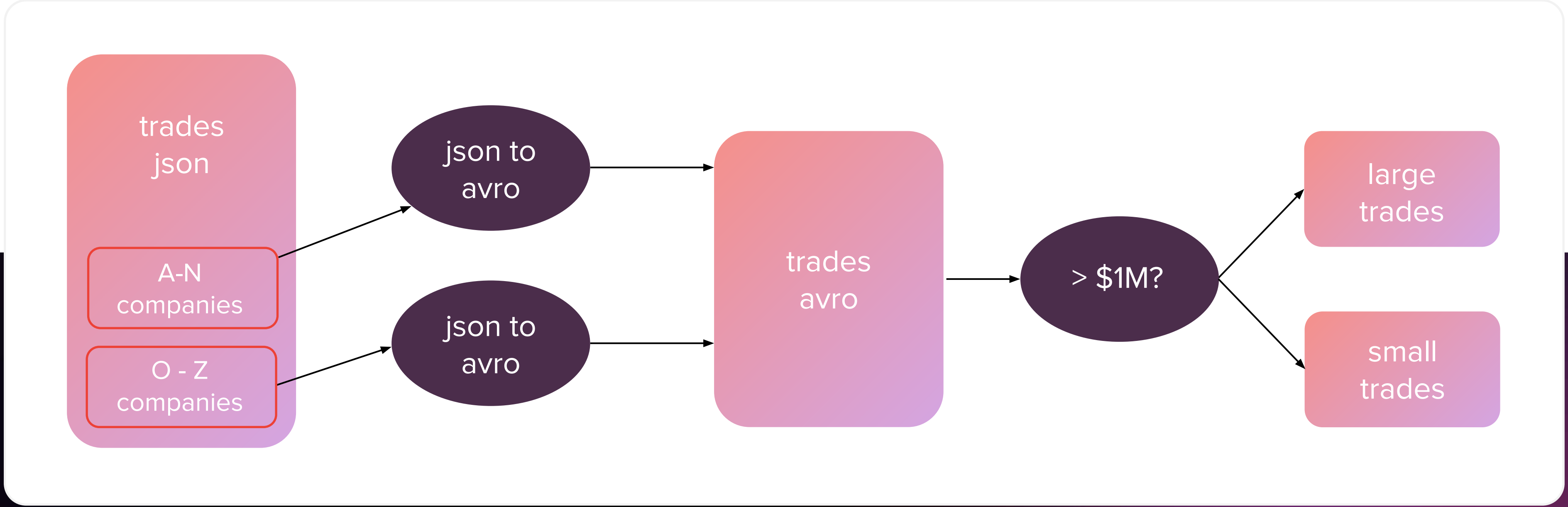
- at most once
- at least once
- exactly once

RTB HOUSE

# Design patterns



# Single-event processing

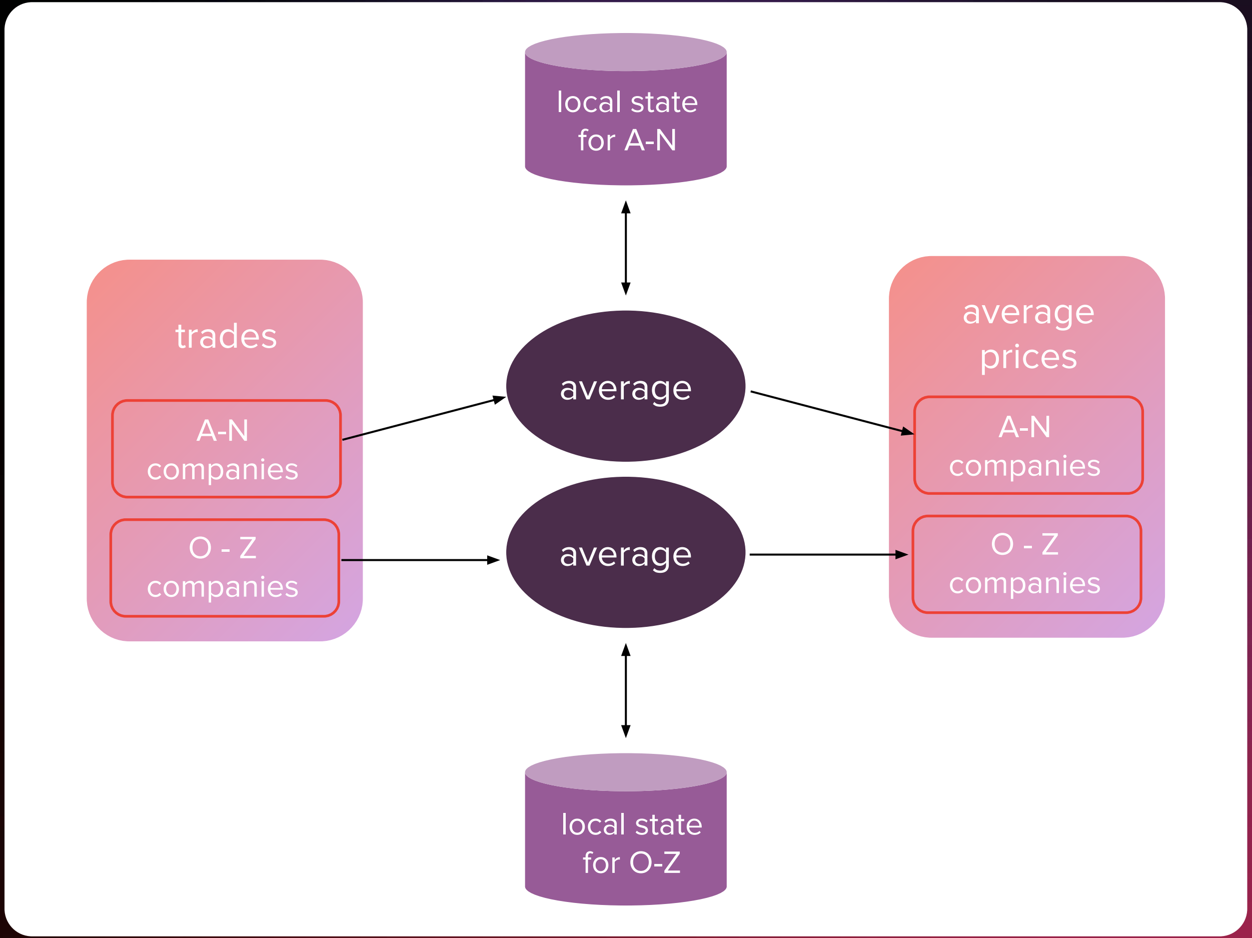



- aka map/filter pattern
- each event is handled independently
- no state == easy scaling & load balancing & failure recovery

RTB HOUSE

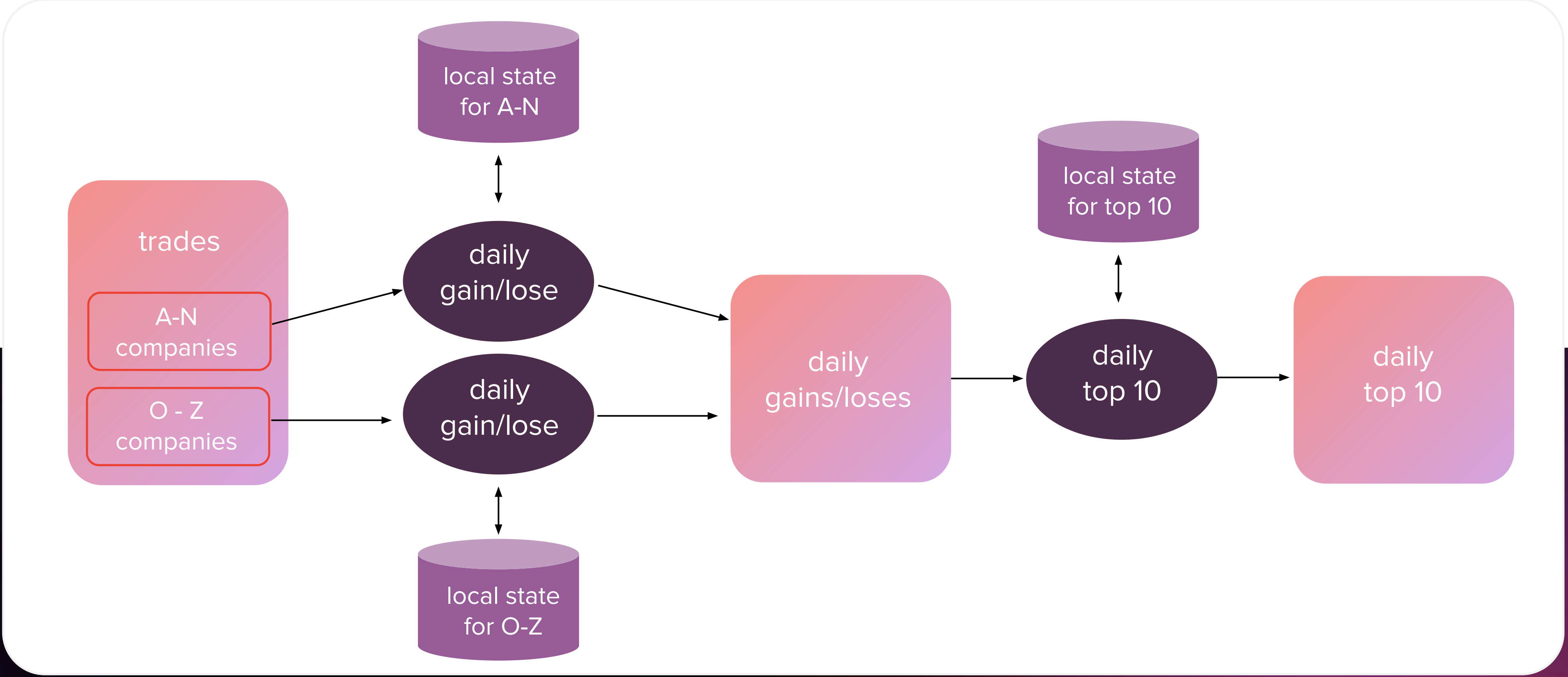
# Local state

- used to aggregate information
- local state == aggregation per partition



RTB HOUSE

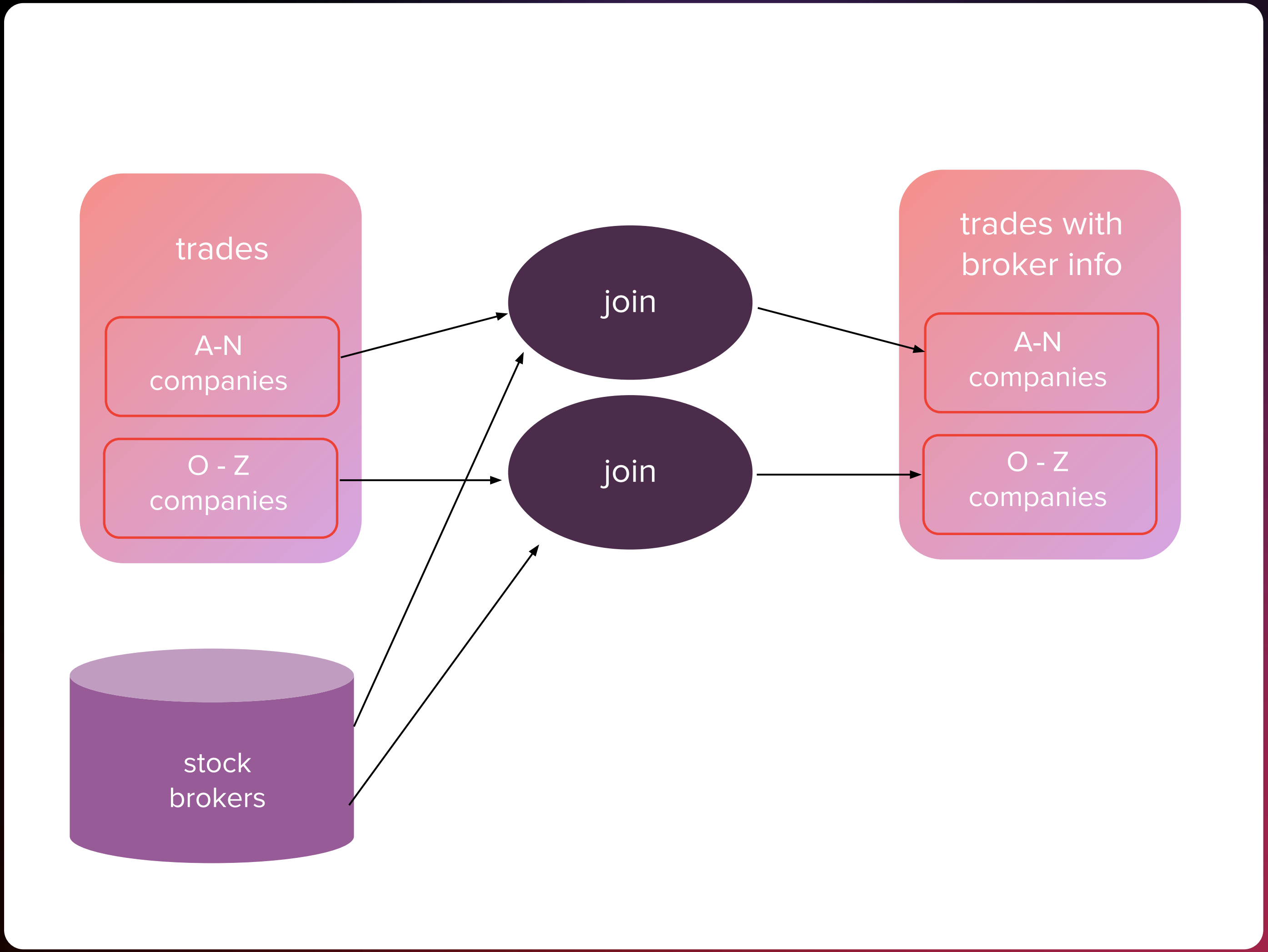
# Multiphase processing




- complex problem often require complex topologies
- similar to multiple MapReduce phases

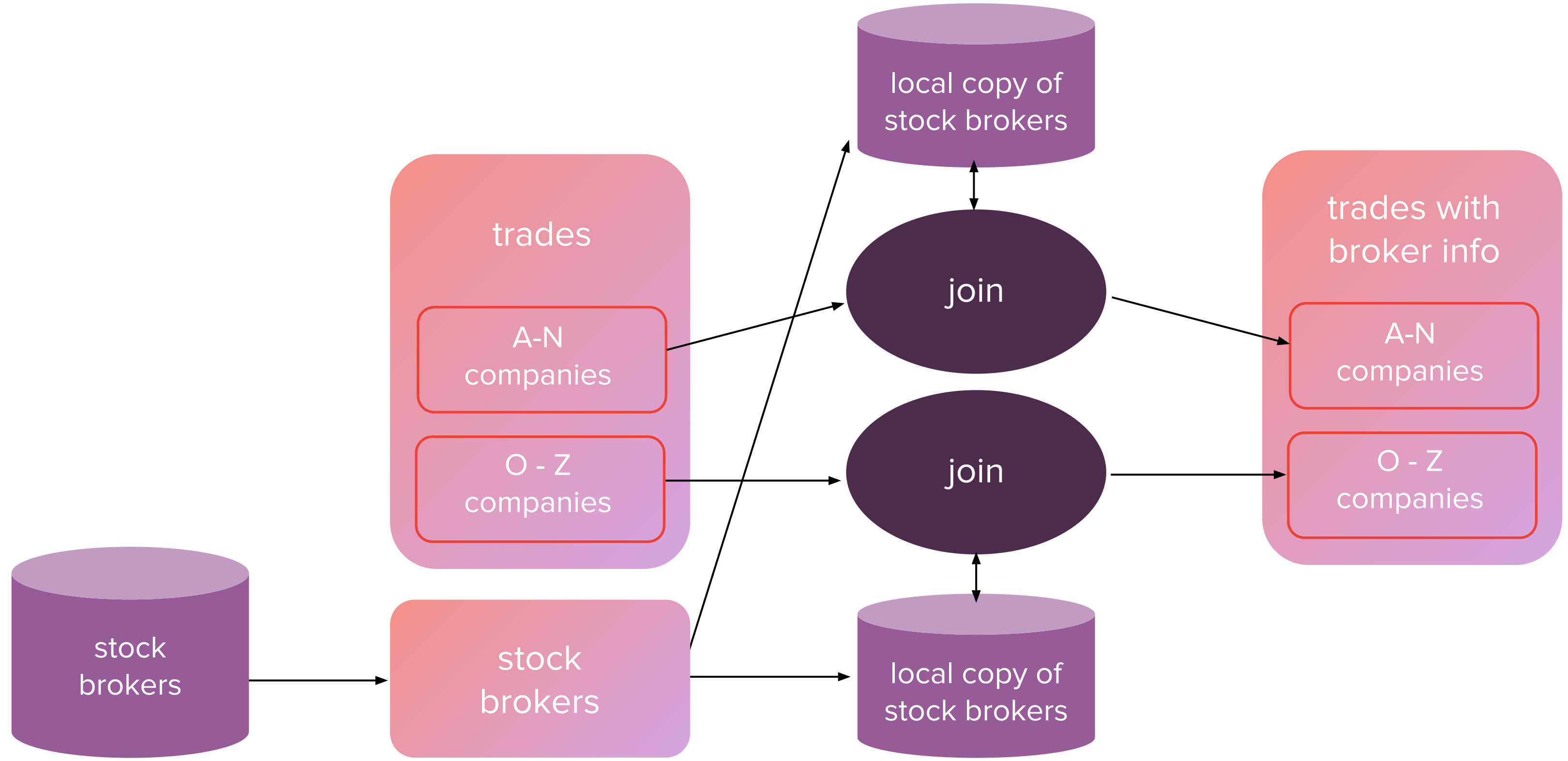
# External lookup (stream - table join)

- enriching stream data with database data



RTB HOUSE

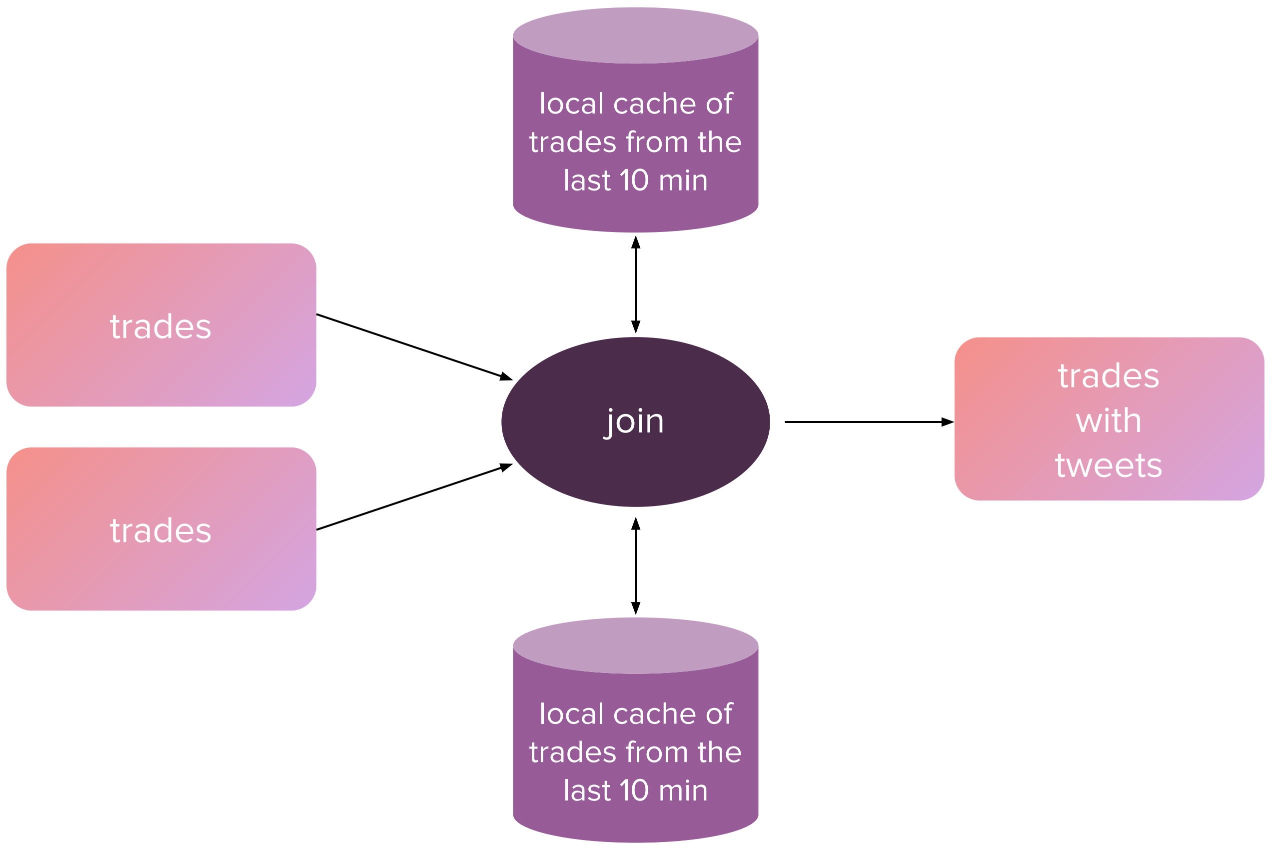
# External lookup (stream - table join)



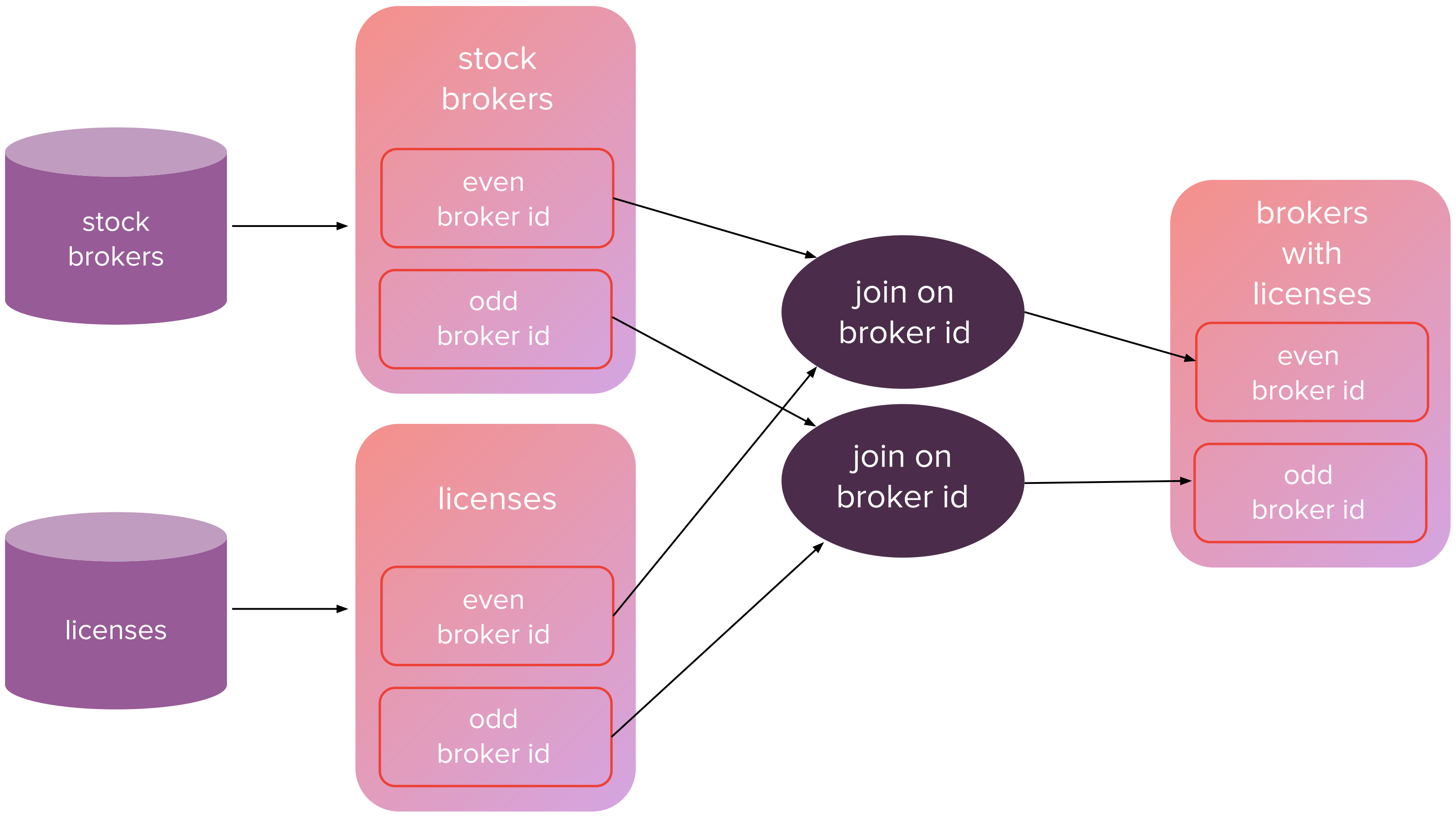

- capturing changes to the database as events
- better performance and availability

RTB HOUSE

# Streaming join

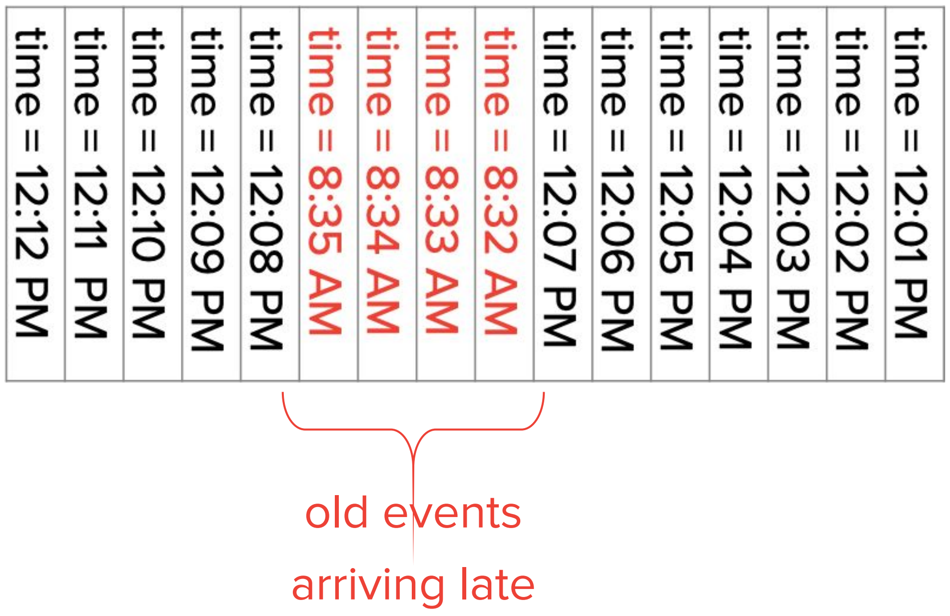


# Table – table join

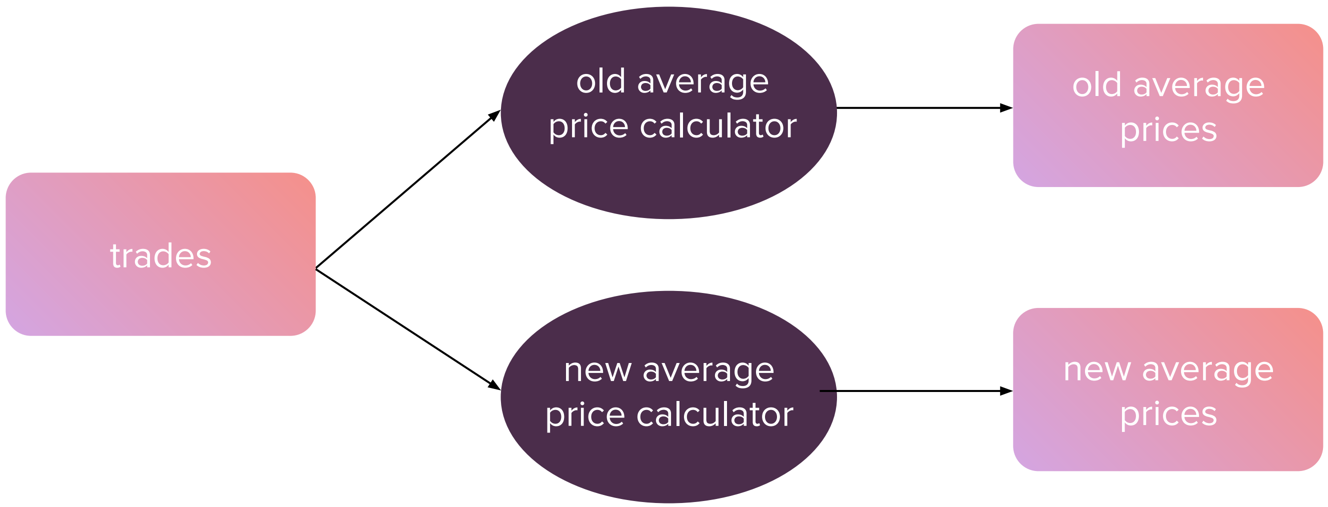


# Out of sequence events

- example: mobile device reconnects after 5h in airplane mode
- no easy solution



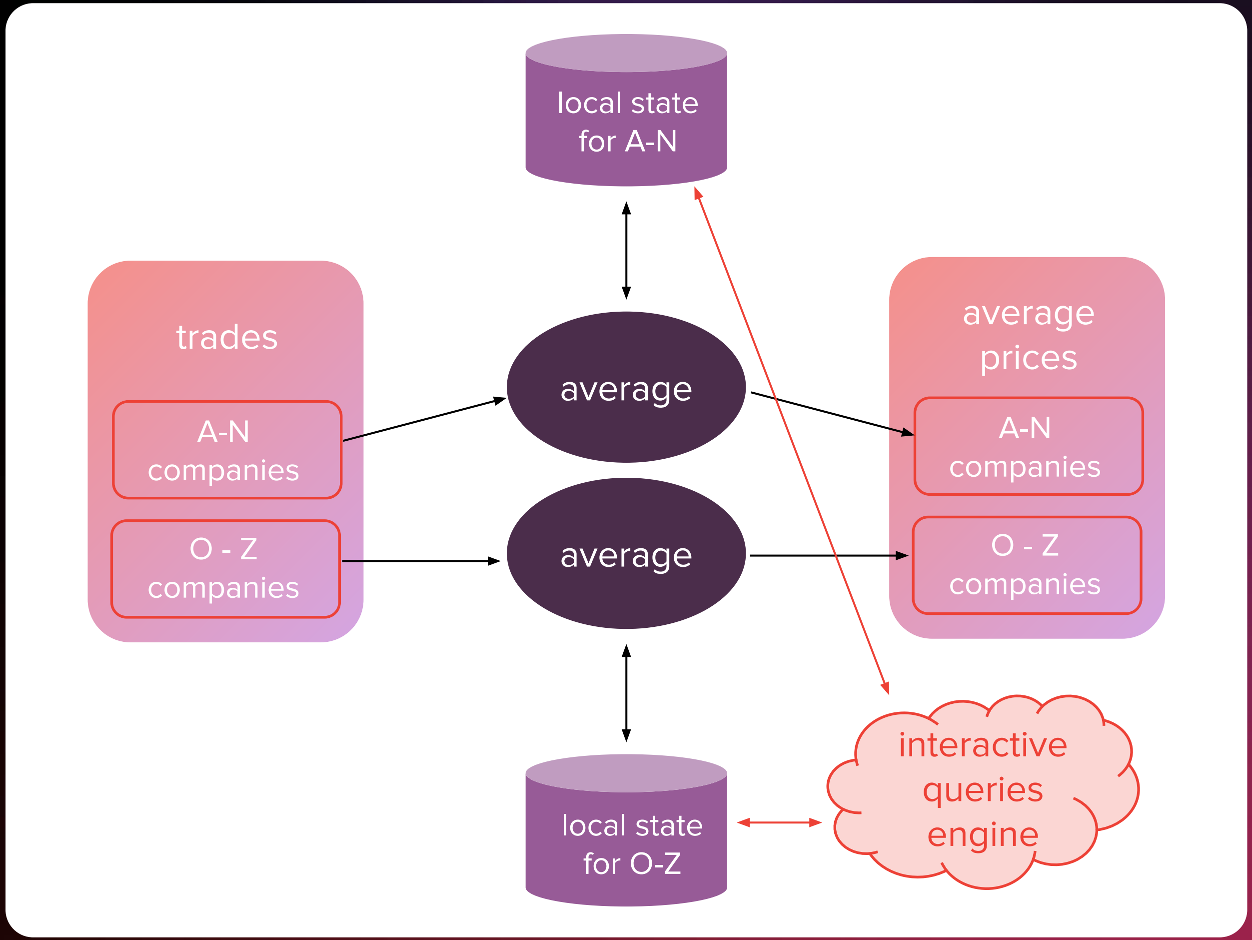
# Reprocessing




- running two versions of the application along each other
- possible thanks to stream properties: replayability, immutability, and ordering

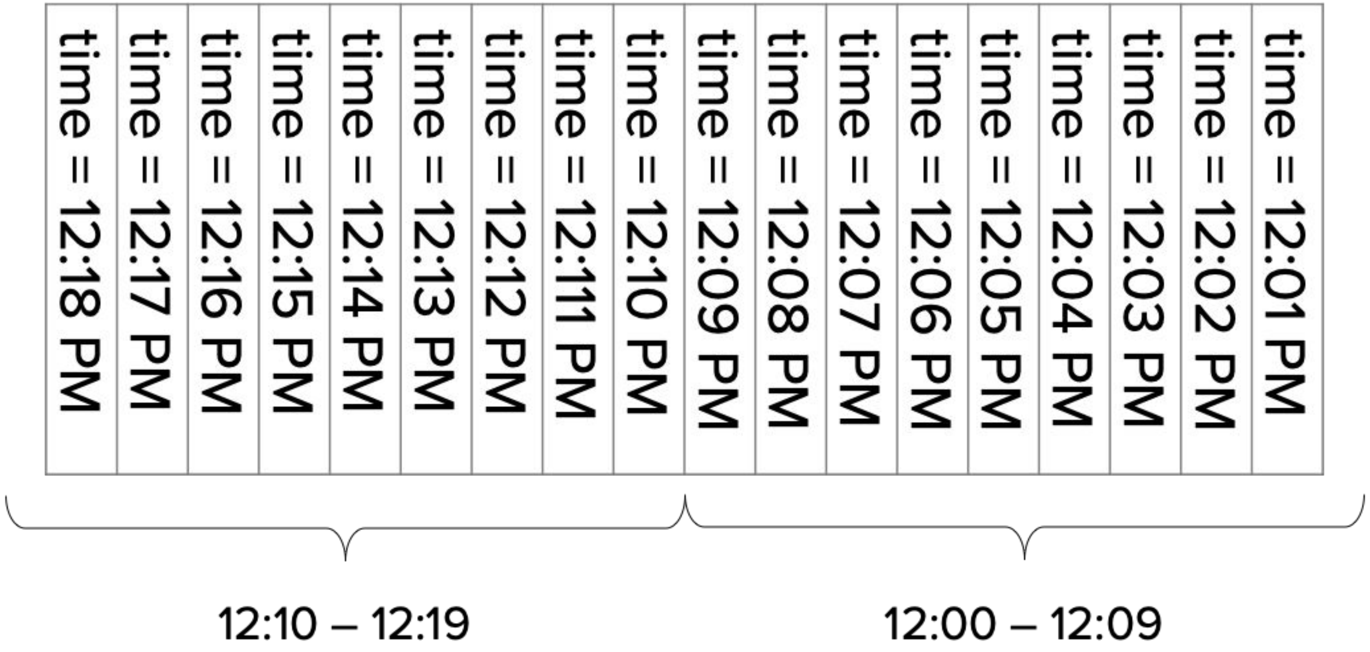
# Interactive queries

- read the results directly from the state instead of querying the database or output topic

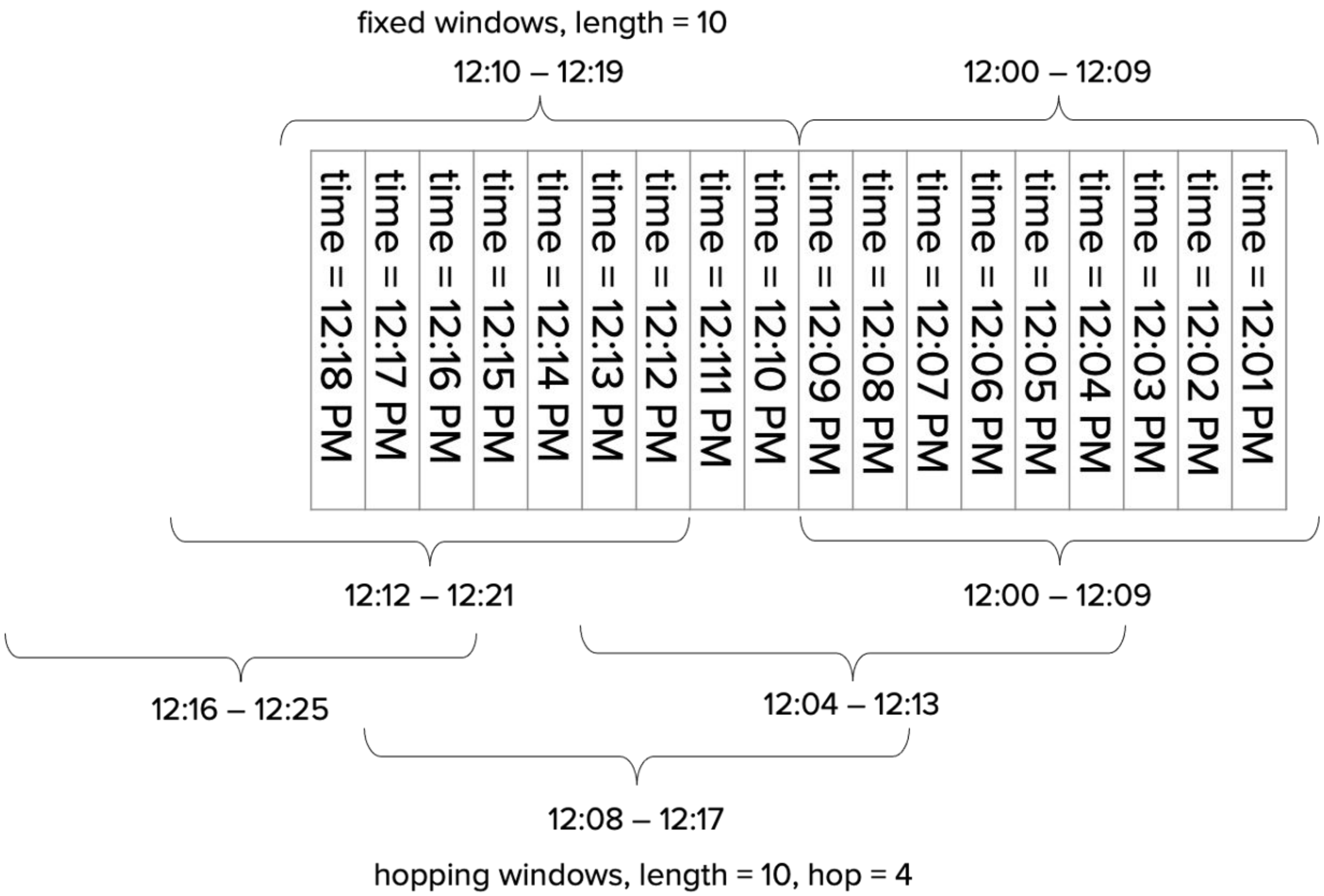


# Time windows

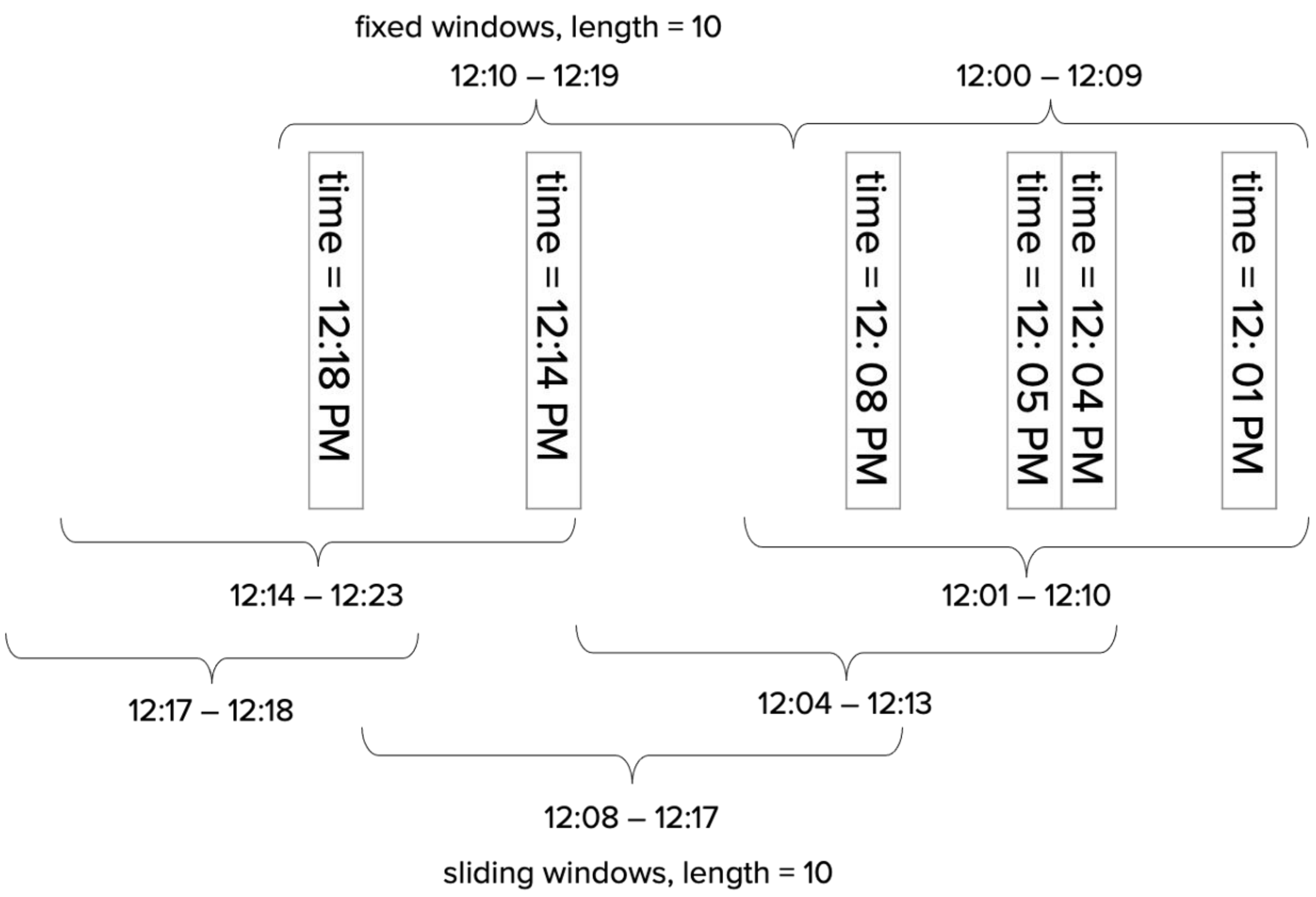
# Tumbling (fixed)



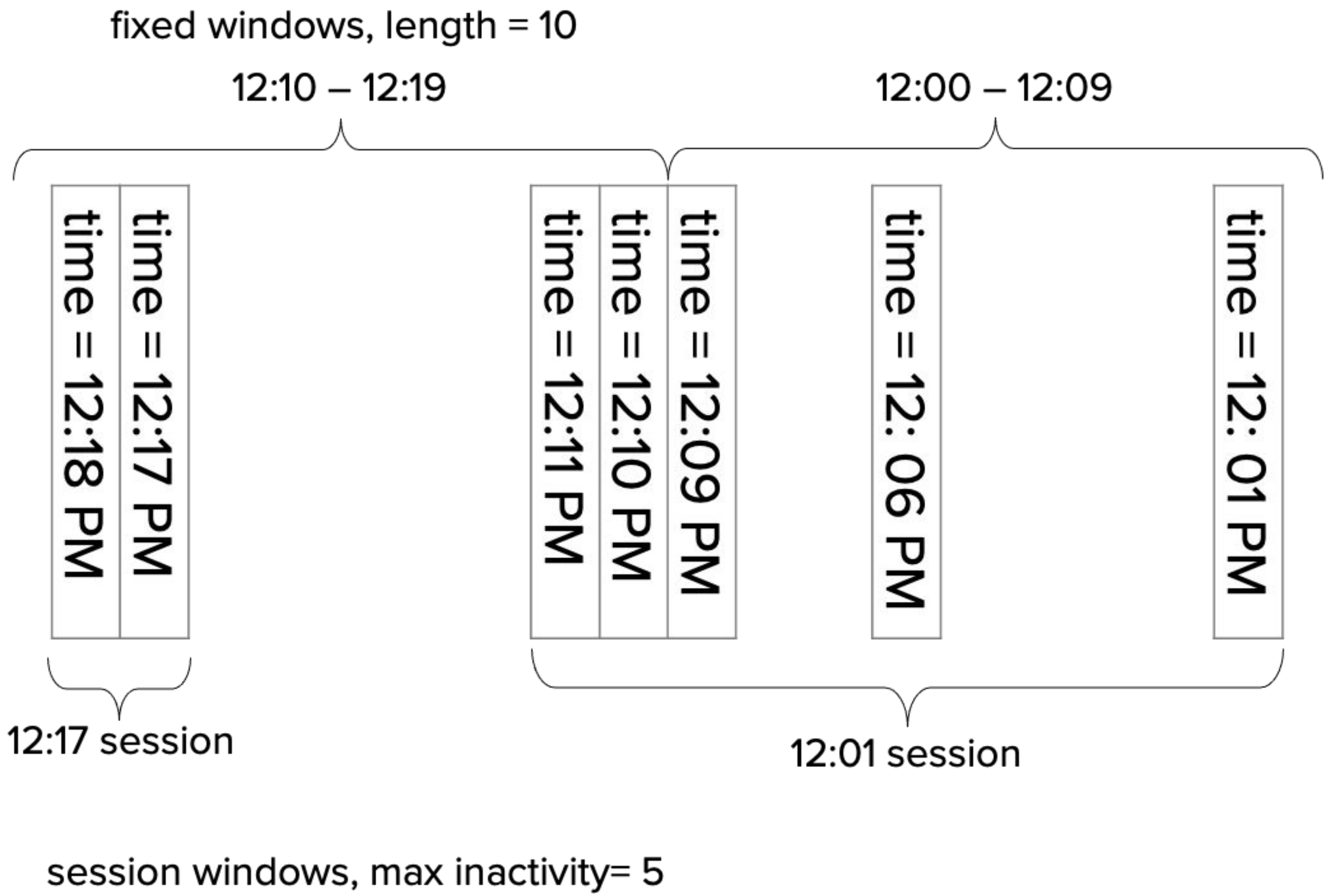
# Hopping


fixed windows, length = 10
12:10 – 12:19
12:00 – 12:09
time = 12:18 PM
time = 12:17 PM
time = 12:16 PM
time = 12:15 PM
time = 12:14 PM
time = 12:13 PM
time = 12:12 PM
time = 12:11 PM
time = 12:10 PM
time = 12:09 PM
time = 12:08 PM
time = 12:07 PM
time = 12:06 PM
time = 12:05 PM
time = 12:04 PM
time = 12:03 PM
time = 12:02 PM
time = 12:01 PM
12:12 – 12:21
12:00 – 12:09
12:16 – 12:25
12:04 – 12:13
12:08 – 12:17
hopping windows, length = 10, hop = 4

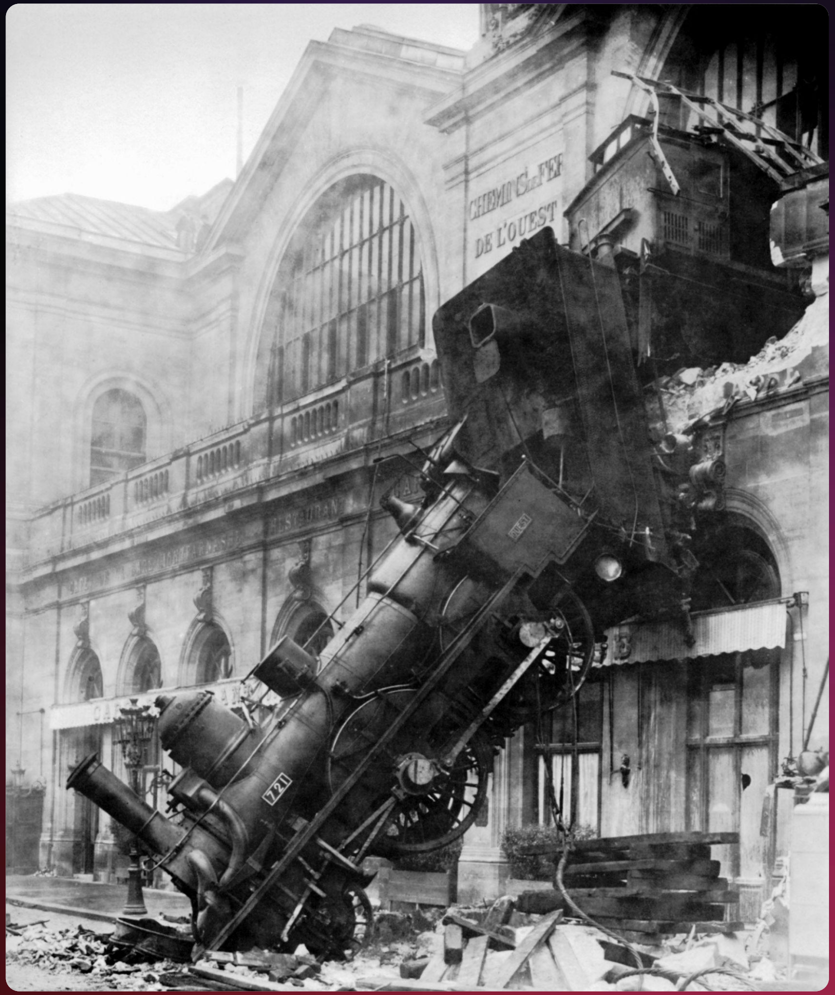
# Sliding


fixed windows, length = 10
12:10 – 12:19
12:00 – 12:09
time = 12:18 PM
time = 12:14 PM
time = 12: 08 PM
time = 12: 05 PM
time = 12: 04 PM
time = 12: 01 PM
12:14 – 12:23
12:01 – 12:10
12:17 – 12:18
12:04 – 12:13
12:08 – 12:17
sliding windows, length = 10

# Session

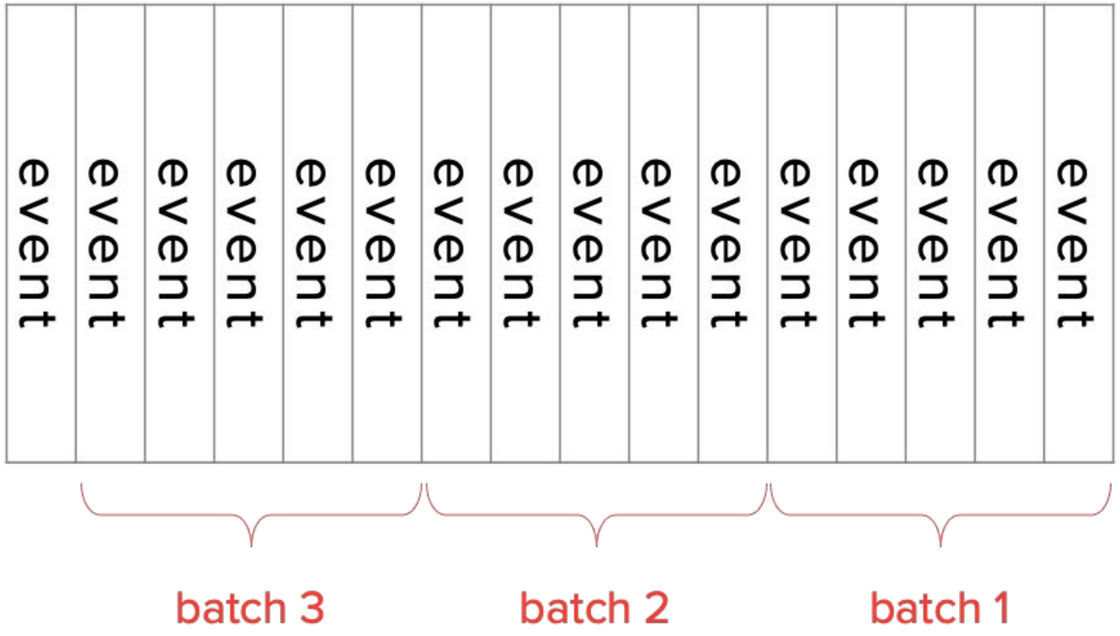

fixed windows, length = 10
12:10 – 12:19
12:00 – 12:09
time = 12:18 PM
time = 12:17 PM
time = 12:11 PM
time = 12:10 PM
time = 12:09 PM
time = 12: 06 PM
time = 12: 01 PM
12:17 session
12:01 session
session windows, max inactivity= 5

# Fault tolerance



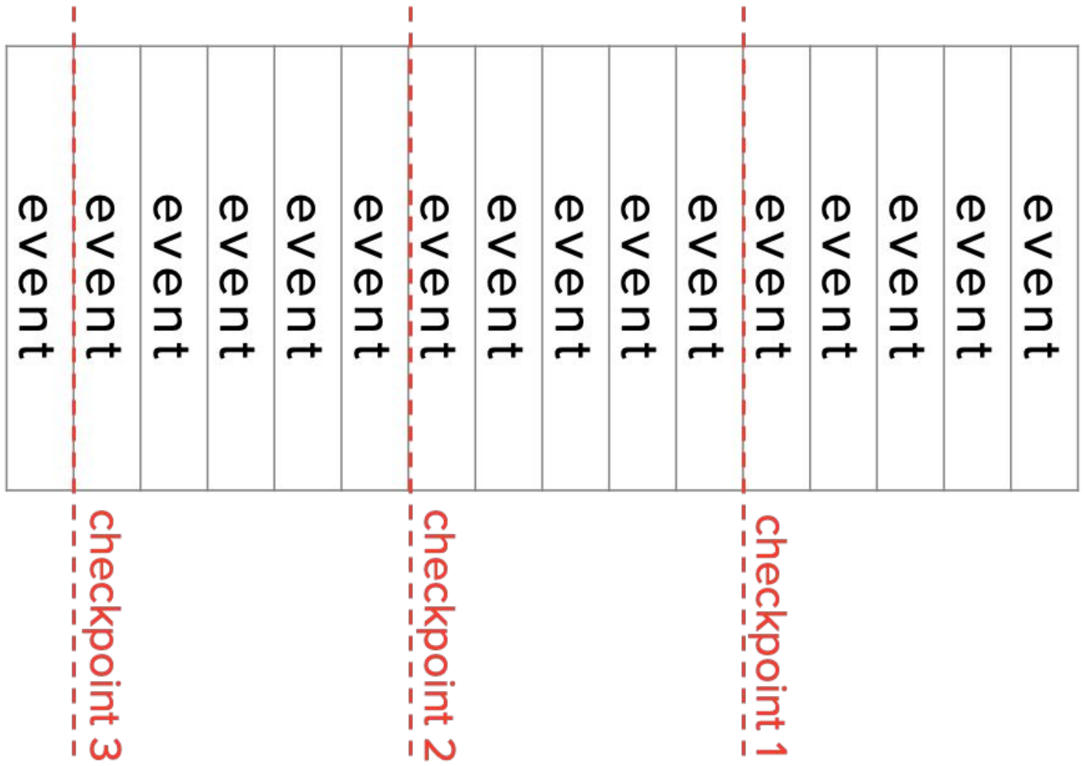
# Microbatching

break the stream into small blocks, and treat each block as a batch process



# Checkpointing

periodically generate a checkpoint of state and write it to persistent storage



# Atomic commit

X/Open XA (eXtended architecture): global transactions across heterogeneous components, based on two-phase commit protocol

Kafka does not support XA

# Idempotence

idempotent operation performed multiple times has the same effect as performed once

example: “UPDATE table SET processed = true WHERE event_id = 101”

at least once + idempotent operations = effectively once

# Rebuilding state

applications must be able to restore the state after a failure

## Option 1

keep the state in a remote database

## Option 2

keep the state local, and replicate it periodically

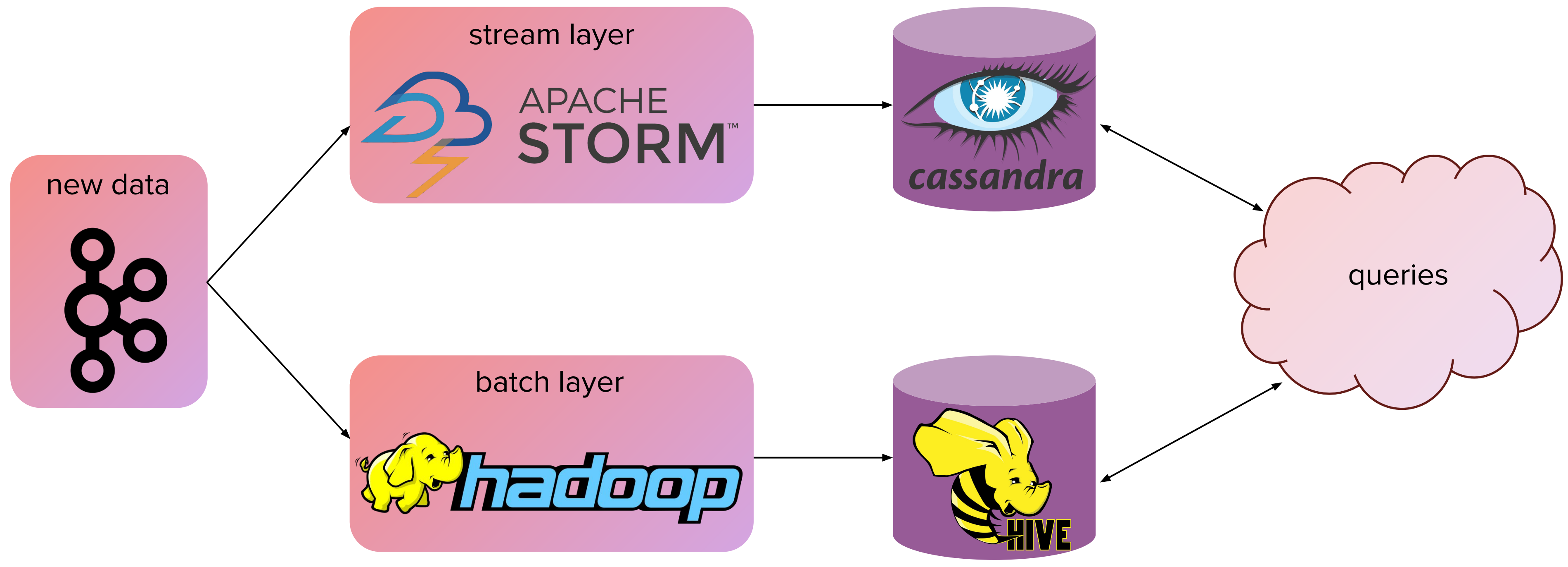
## Option 3

do nothing, and after a failure simply replay the input stream from start and rebuild the state

# Streaming Architecture

# Lambda architecture

“How to beat the CAP theorem” - Nathan Marz, 2011



# Benefits & challenges

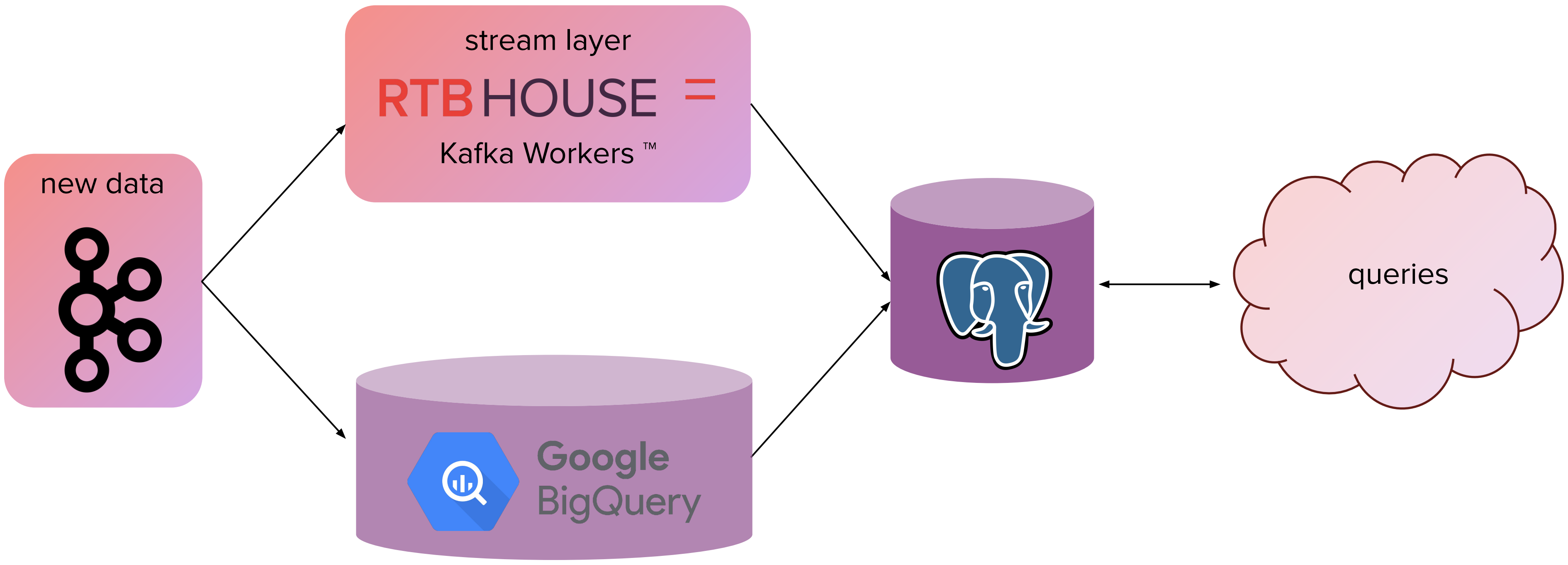
## Positives:

- best of two worlds: low latency and possibility to process historical data
- algorithmic flexibility
- easy ad-hoc analysis

## Negatives:

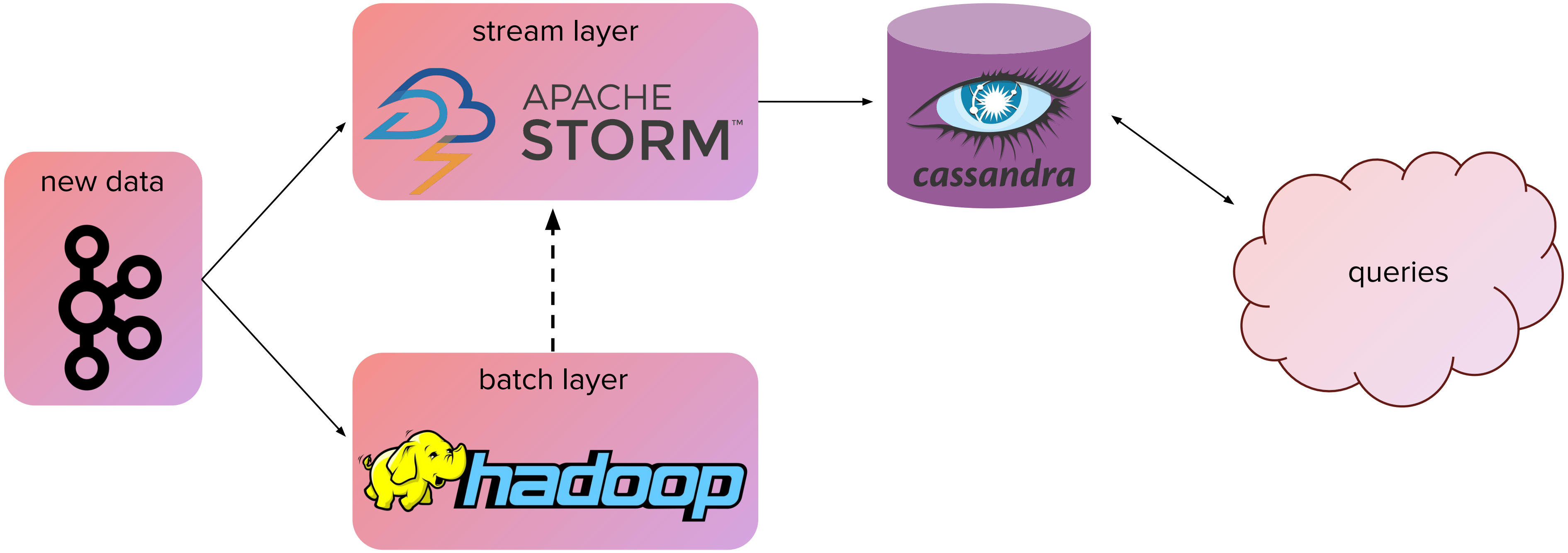
- inherent complexity
- maintaining two codebases
- synchronization between the layers is needed

# Practical example: stats counter



# Kappa architecture

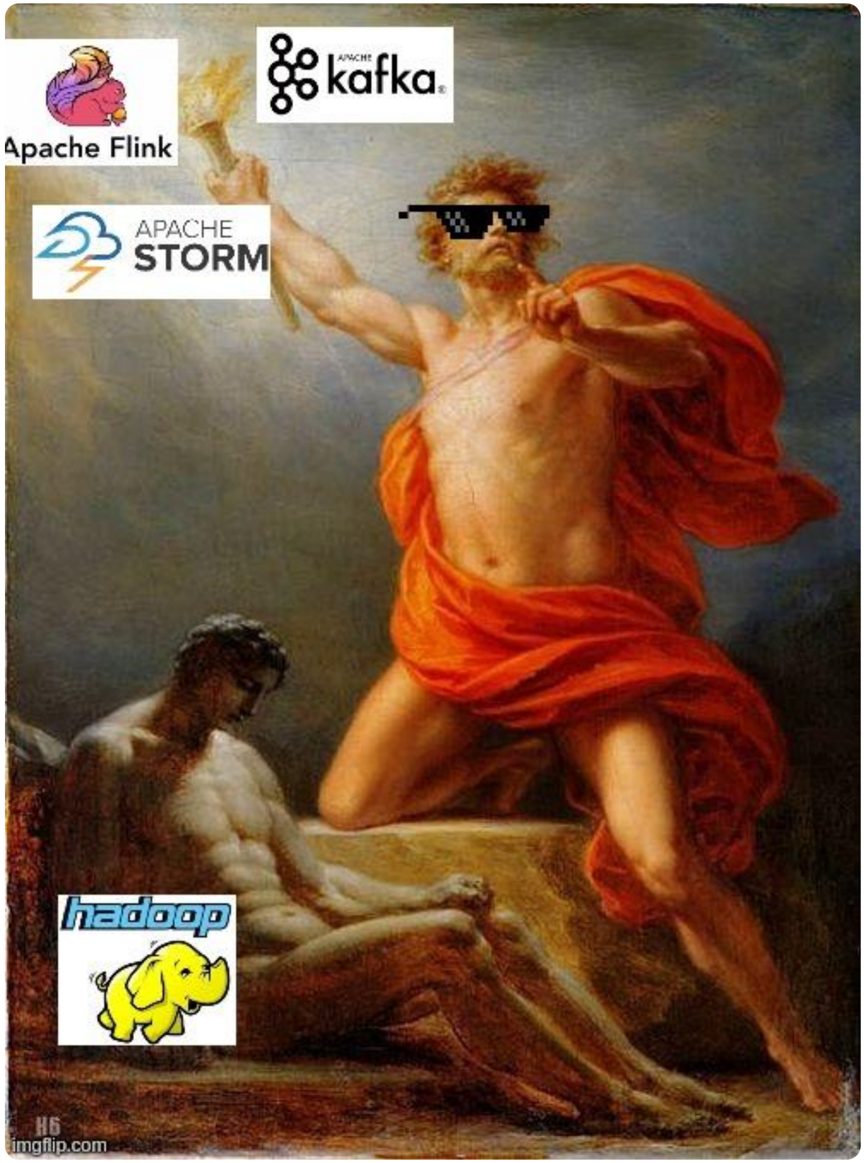
“Questioning the Lambda Architecture” - Jay Kreps, 2014



# Summary

Two things to remember:

- strengths and weaknesses of data streams
- basic concepts and techniques



RTB HOUSE

# Thank you.

Piotr Jaczewski