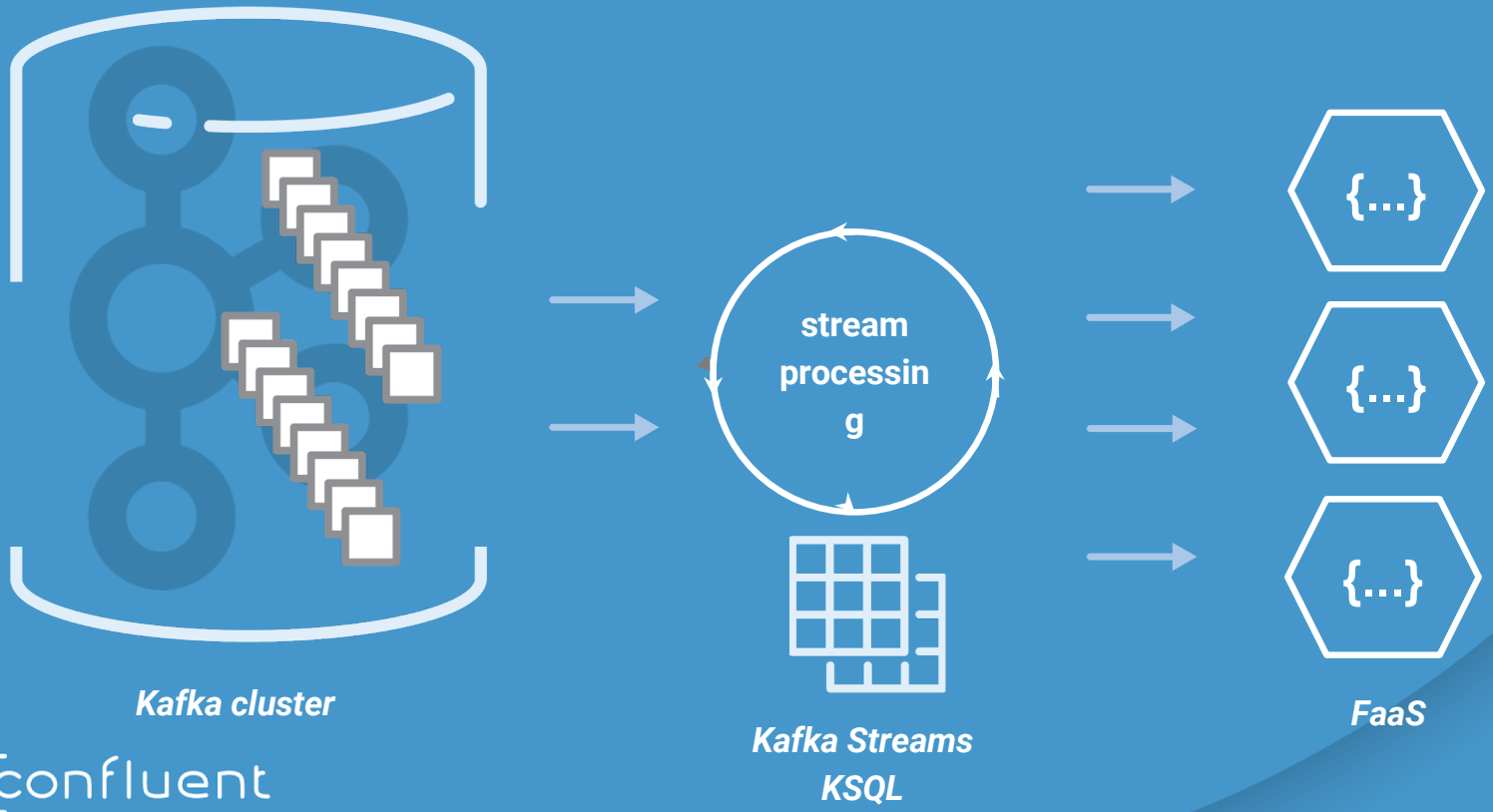
Event driven architectures with Kafka

confluent

A stack view

from stateful to stateless



Kafka cluster

Kafka Streams
KSQL

FaaS

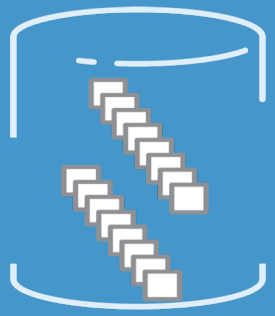
confluent

Stream processors versus FaaS processors

Kafka Streams & KSQL

- Stateful (and stateless)
- In band
- Dataflow - control plane
- Not ad hoc (except KSQL)
- Elastic

Think: data-model -->



confluent

FaaS - GCP Fn, Azure Fn, AWS Lambda etc

- Stateless
- Out of band (generally)
- Edge (in band)
- Ad hoc
- Super-elastic

Think: fire and forget -->

Events

What is an event?

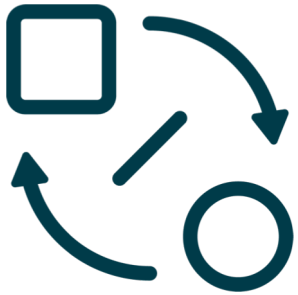
SOMETHING HAPPENED!

FACT!

Events

A Sale

An Invoice



A Trade

A Customer Experience

confluent

Events

Why do you care?

Loose coupling, autonomy, evolvability, scalability, resilience, traceability, replayability

...more importantly...

BEING EVENT-FIRST CHANGES HOW YOU THINK ABOUT WHAT YOU ARE BUILDING

confluent

Events versus commands

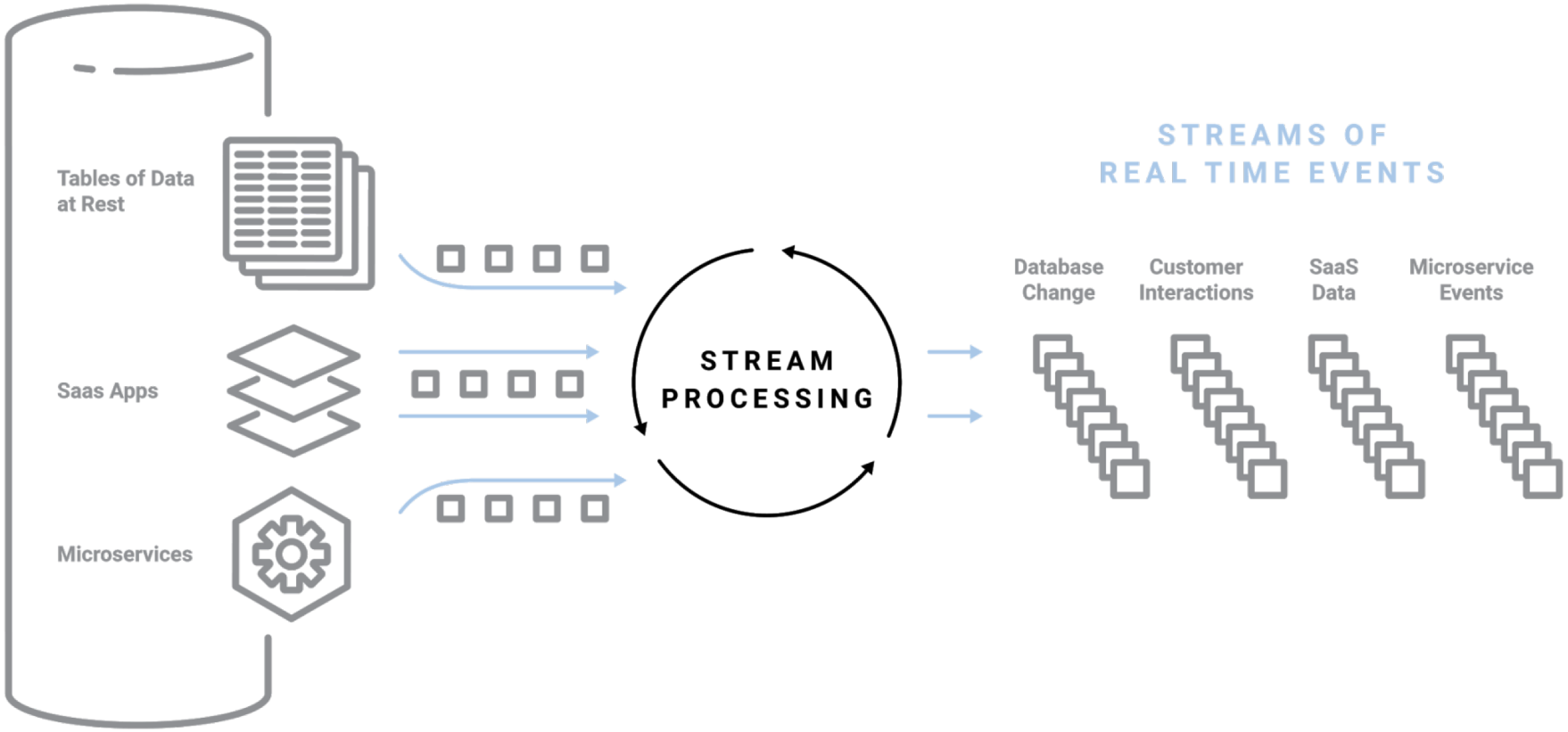
Events:
- Fact
- Something happen
- Immutable
- When
- Behavioural

Commands:
- Intent
- Contract
- Do something
- Coupling
- Structural

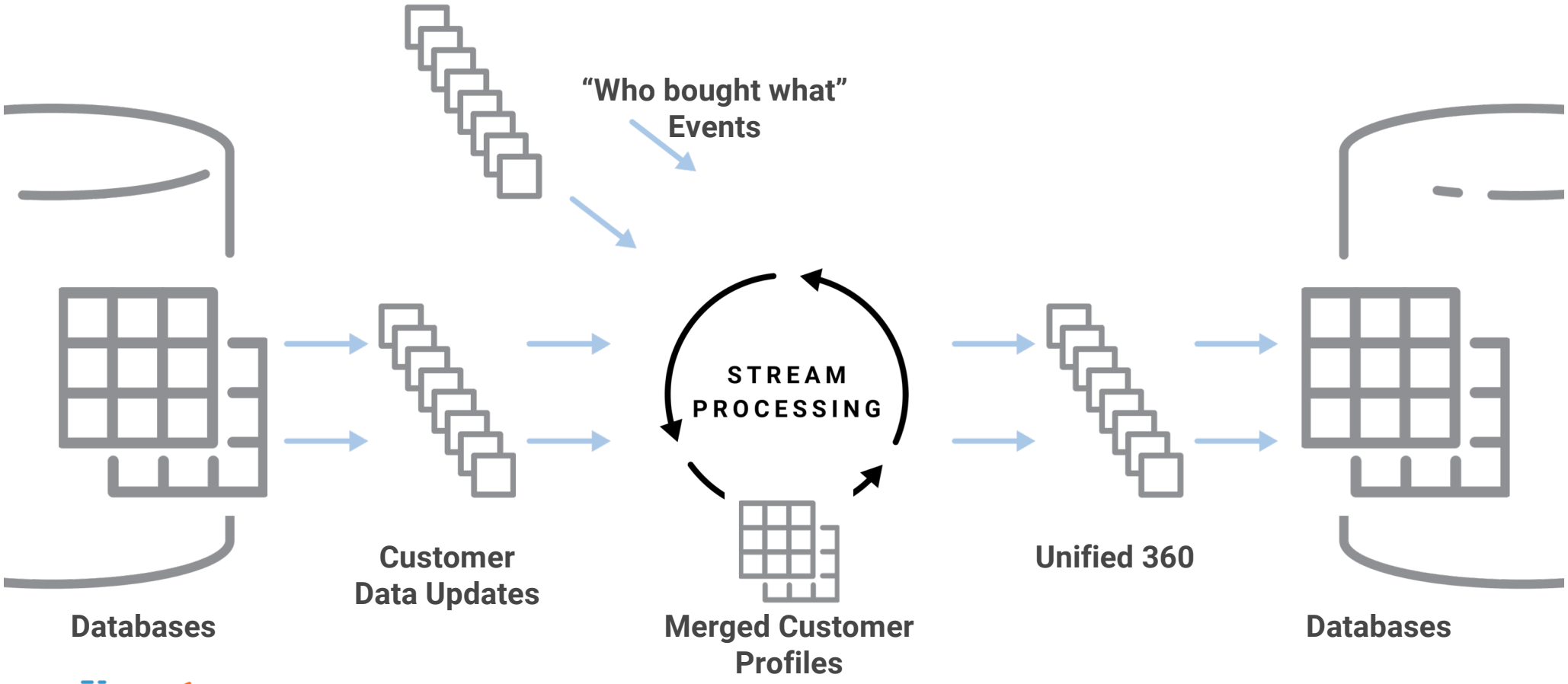
EVENTS ARE USED TO MODEL BEHAVIOUR IN A DOMAIN
EVENT-SOURCING CAPTURES THAT BEHAVIOUR

confluent

All your data is a stream of events

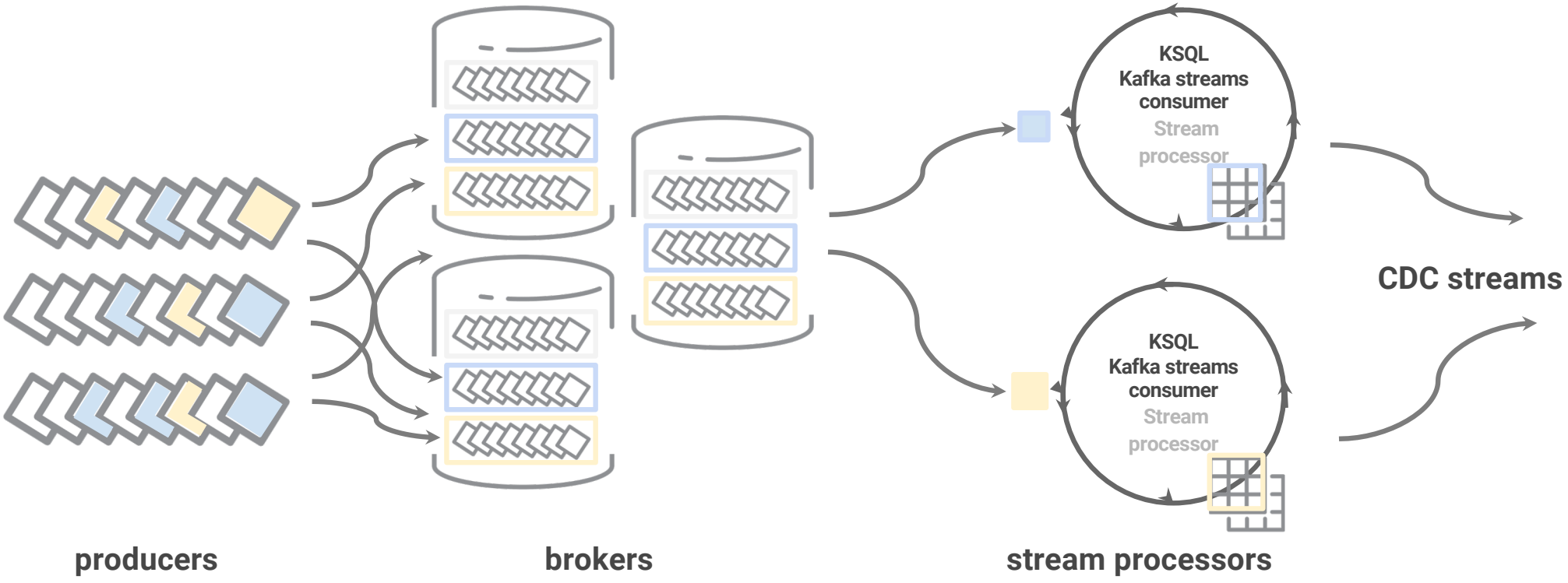


Events and data pipelines: Events as data



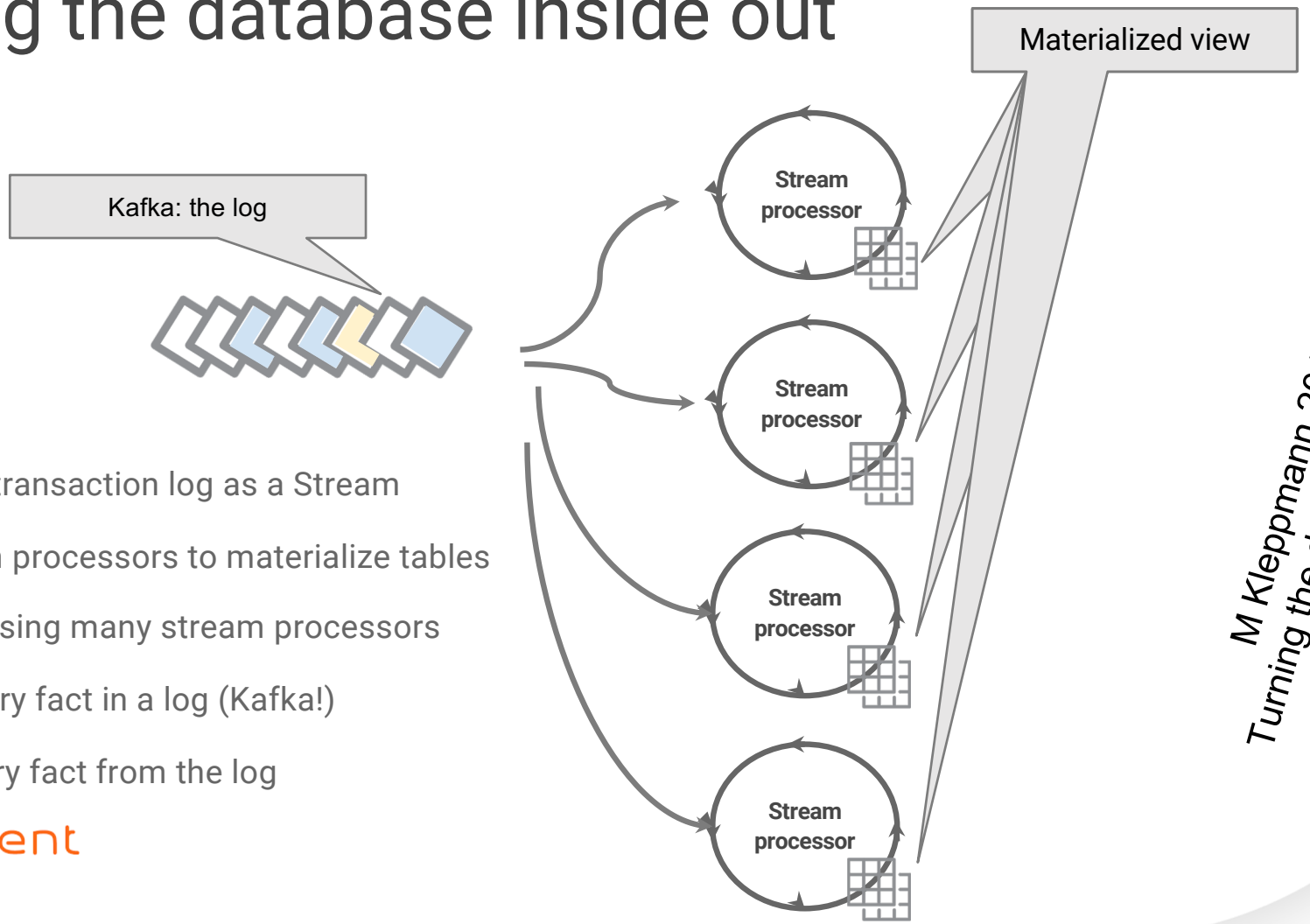
confluent

Events, Streams, Partitions, Tables



confluent

Turning the database inside out

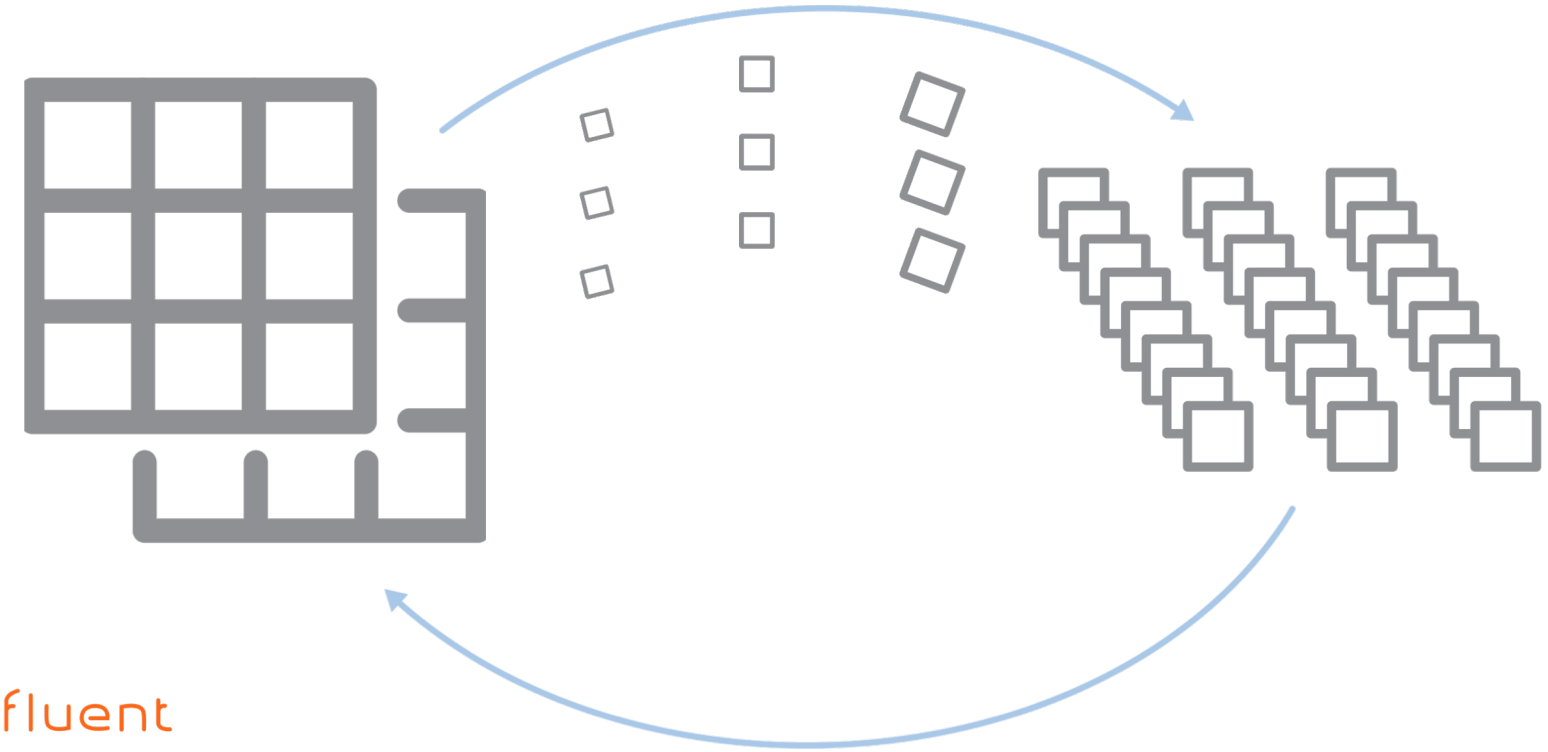


- Model the transaction log as a Stream
- Use stream processors to materialize tables
- Scale out using many stream processors
- Record every fact in a log (Kafka!)
- Replay every fact from the log

confluent

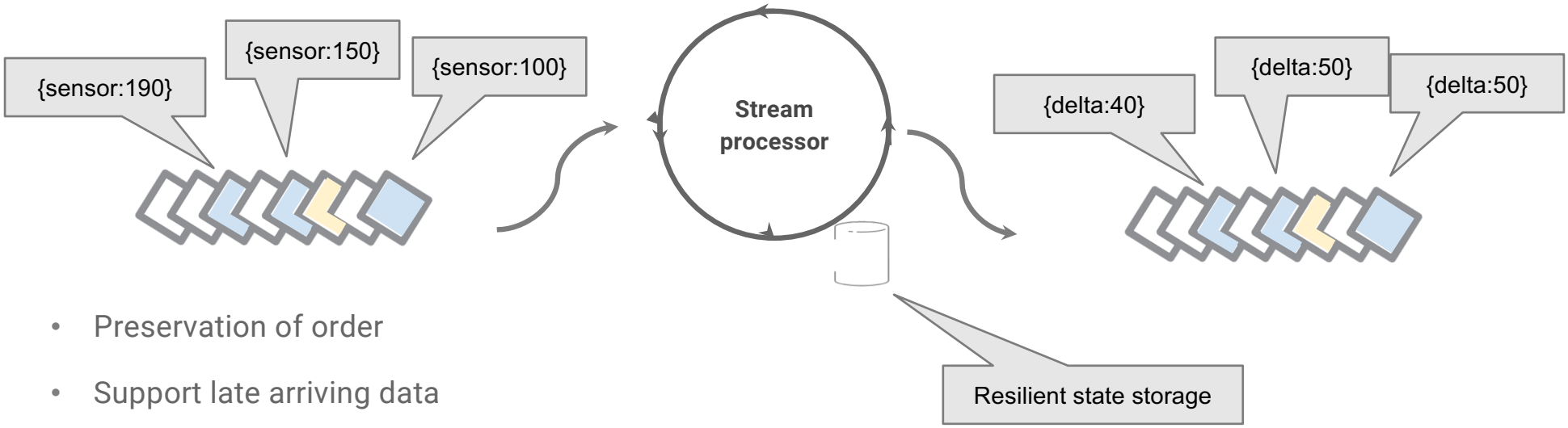
M Kleppmann 2015: Turning the database inside out

You need both tables and streams



confluent

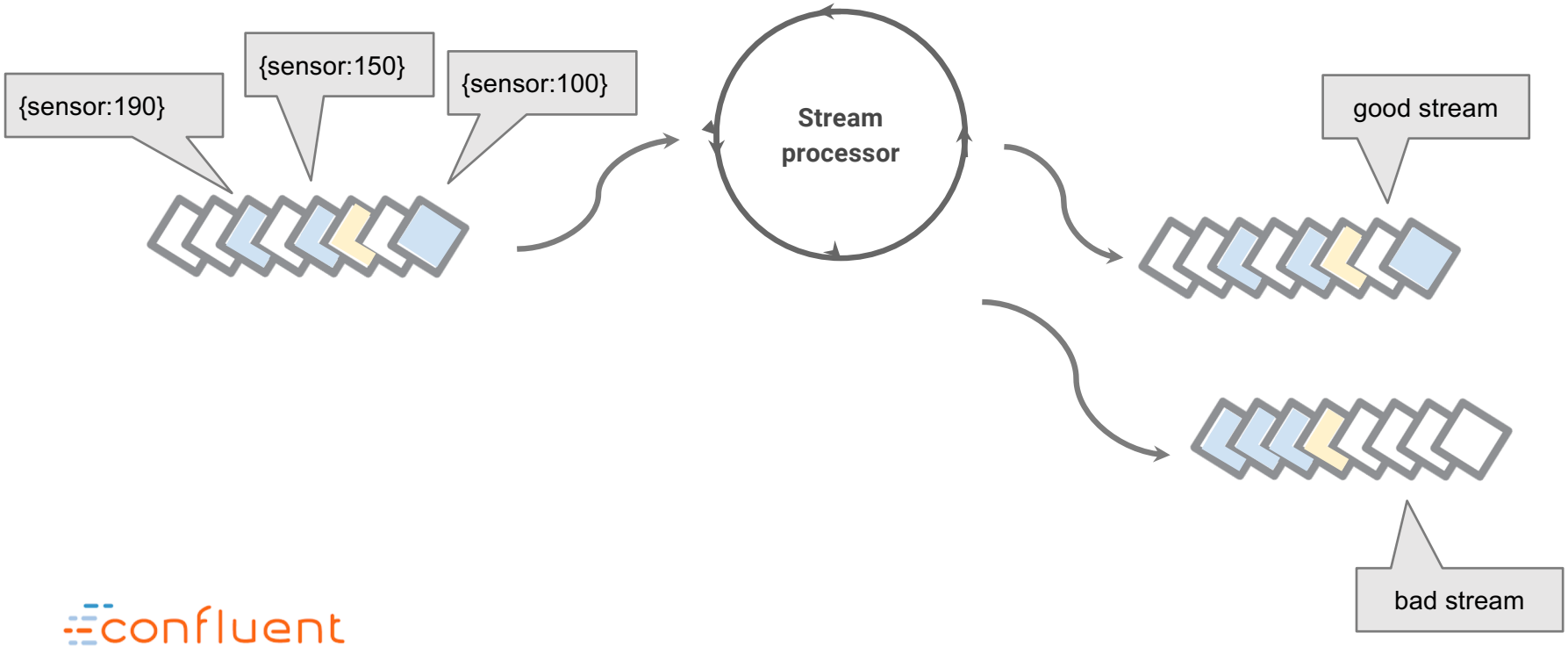
Streaming correctness (stateful)



- Preservation of order
- Support late arriving data
- Tolerance for out-of-sequence data
- Support machine failure scenarios
- Exactly once --- plus the ability to reason about time

confluent

Streaming correctness (stateless)



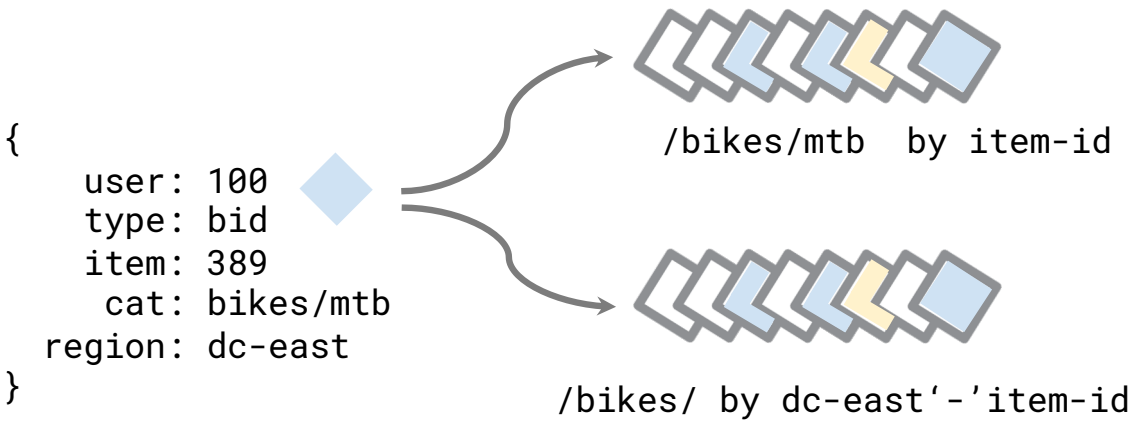
confluent

Data Modelling

Think events not commands,

-> streams of events,

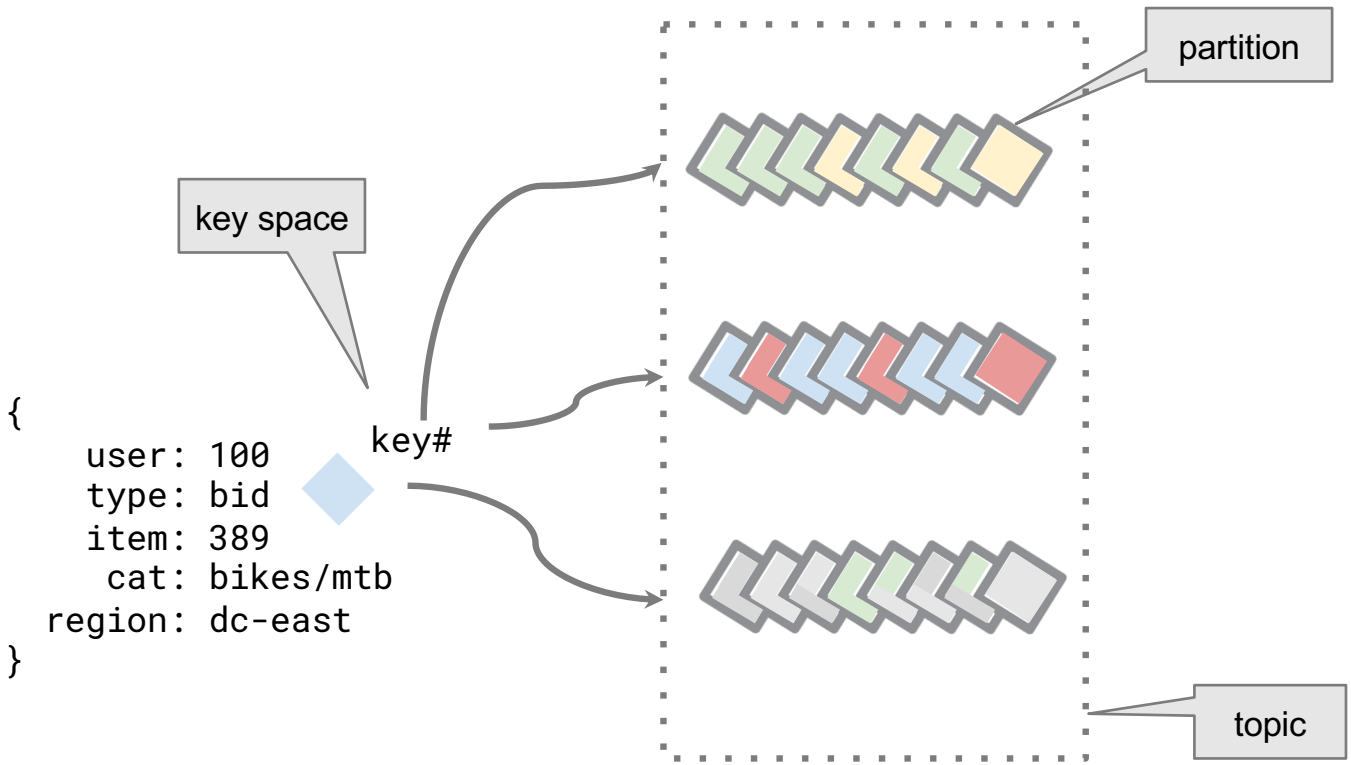
-> series of streams to model the domain



- Keyspace
- Throughput (events per sec)
- Throughput (historic)
- Parallelism
- Replicas
- Retention
- Data evolution

confluent

Data Modelling




```
{
    user: 100
    type: bid
    item: 389
     cat: bikes/mtb
  region: dc-east
}
```

/bikes/mtb by item-id

- Keyspace
- Throughput (events per sec)
- Throughput (historic)
- Parallelism
- Replicas
- Retention
- Data evolution

confluent

Stateless versus Stateful

Stateless

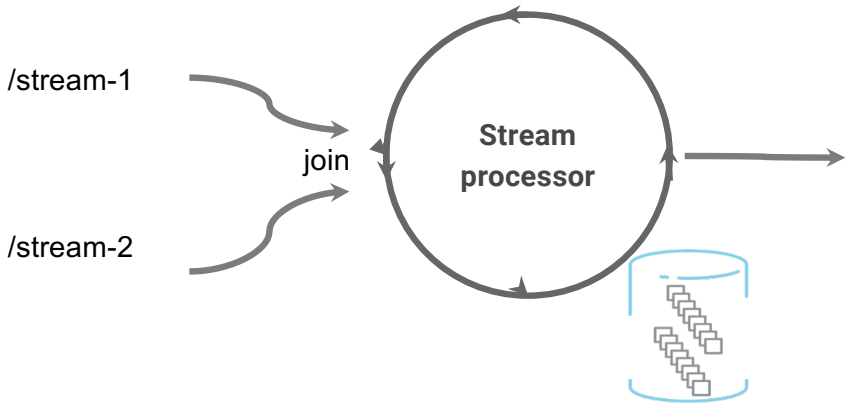
- Filtering
- Transform
- Projection

Think: fire and forget!

Stateful

- Window
- Aggregate
- Join (stream-table, stream-stream etc)

Think: remembers events!



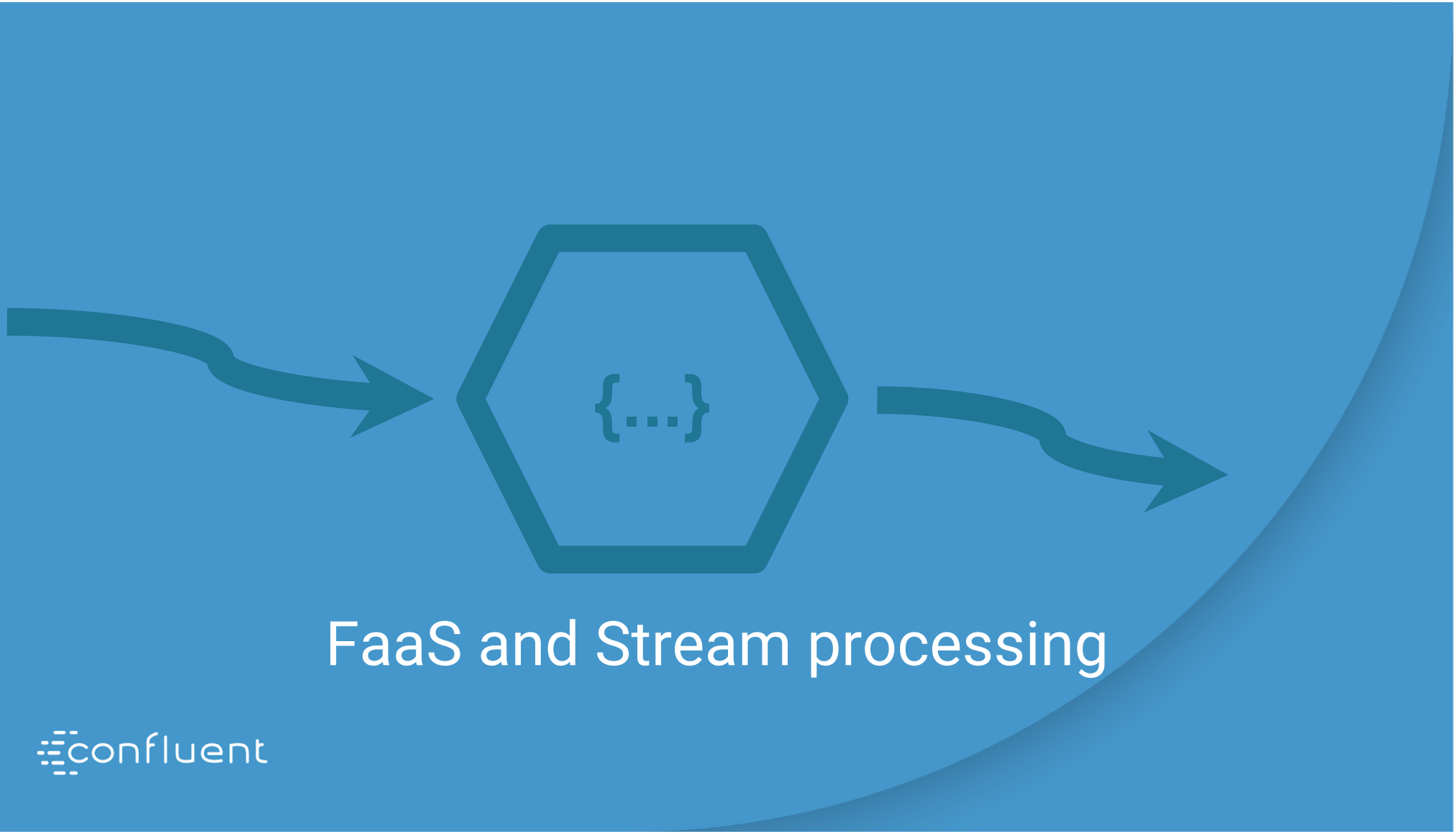
confluent

Winning with Event-driven streaming architectures

the secret sauce

the log, event sourcing, source of truth, CQRS, event collaboration, replayability, at-least once, exactly one, evolutionary architectures, data-virtualization and more

confluent



FaaS and Stream processing

confluent

What is FaaS?

- Bring your own compute
- Elastic
- Pay per use
- Stateless
- Cloud native

Any language, concurrent, sync or async

confluent

FaaS apps

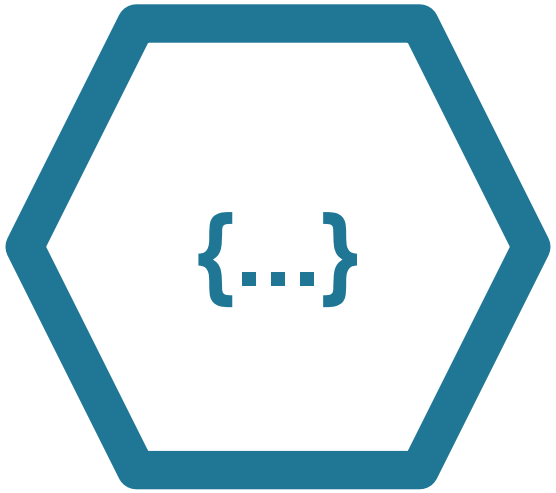
1. Single site, events in any order

   Async chained, **stateless**

2. **Non-stream** oriented, **non-time** critical

3. **Edge** processing (in or out)

4. Enrich **incoming** events (stateless etl)
   or
   outgoing - email users, enrich image, ad-hoc requests



confluent

FaaS BUTs

- Sync chaining FaaS anti-pattern
- Complicated on different platforms
- Really really granular - like ~~micro~~nano-services
- Not cross cloud interoperable yet (without kafka or svrless fwk)
- Testability sucks
- Automation sucks

… but …

FaaS requires the event-driven paradigm in order to be successful.

You cannot throw away 25 years of event-driven legacy and go back to rpc (fail!)

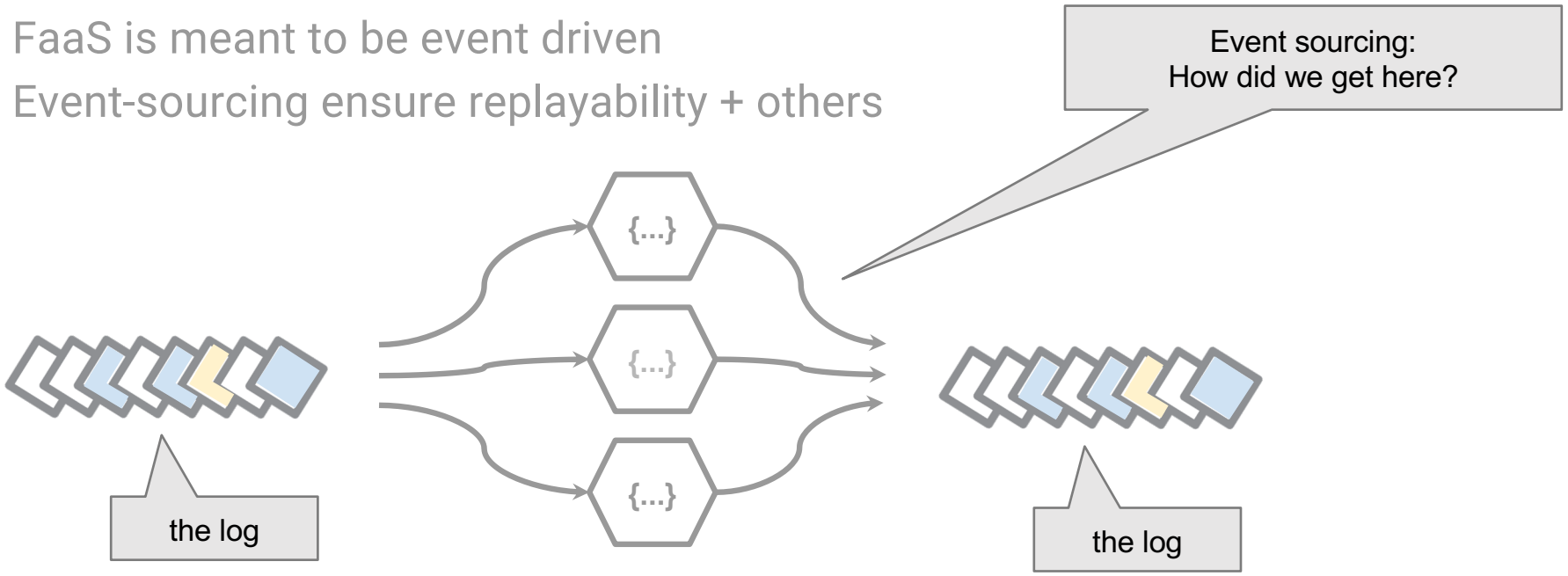
It's not 'event driven' unless you use event-sourcing

confluent

FaaS and event-sourcing

Recap:

- Events are behavioural
- FaaS is meant to be event driven
- Event-sourcing ensure replayability + others

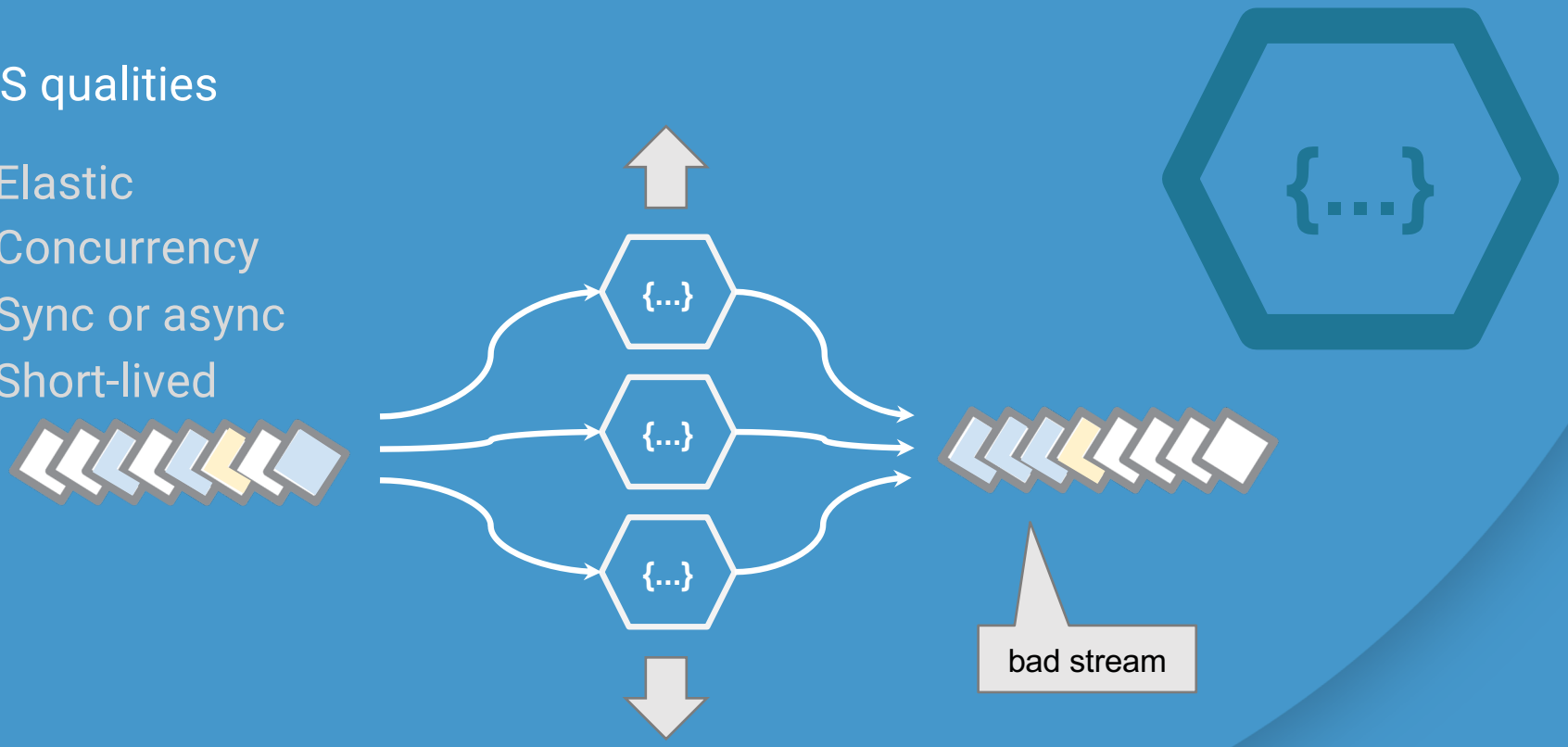


confluent

FaaS and streaming correctness

FaaS qualities

- Elastic
- Concurrency
- Sync or async
- Short-lived



confluent

FaaS for Stream processing

Needs...

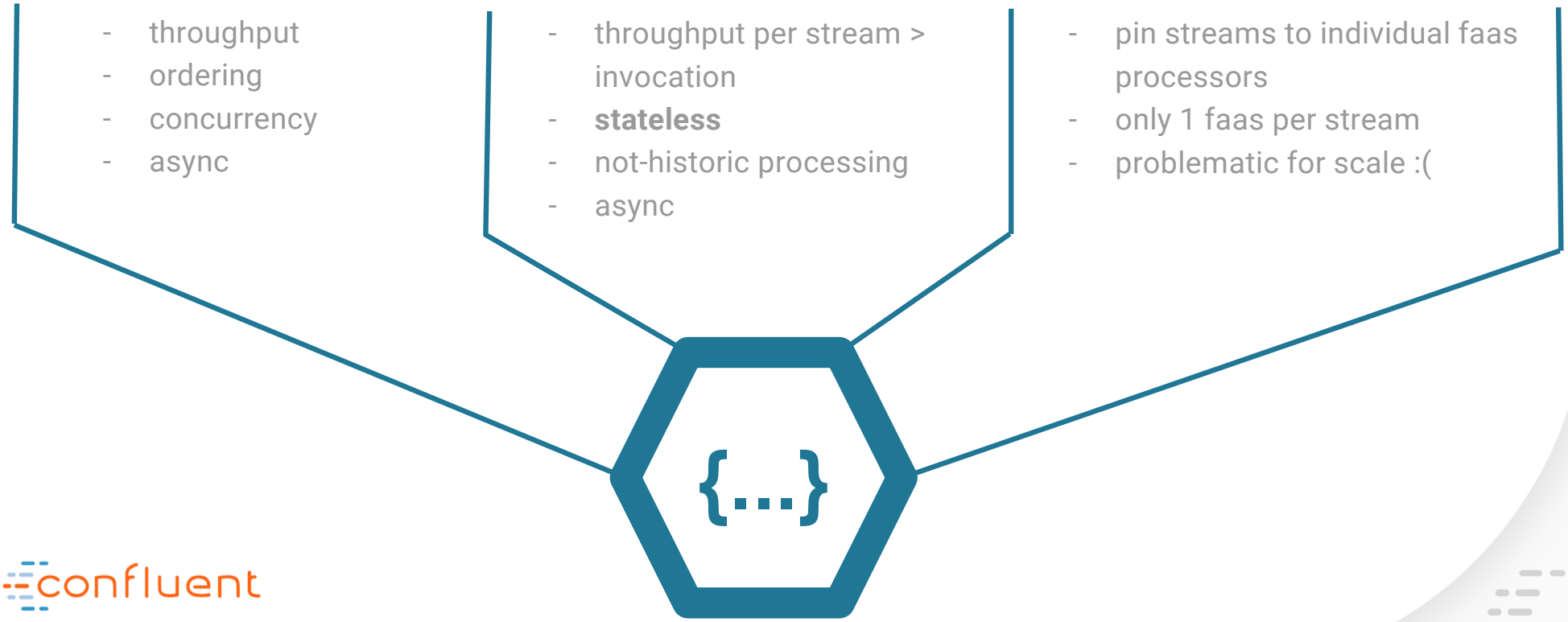
- throughput
- ordering
- concurrency
- async

Works when...

- throughput per stream > invocation
- **stateless**
- not-historic processing
- async

Otherwise...

- pin streams to individual faas processors
- only 1 faas per stream
- problematic for scale :(



confluent

FaaS and the auction platform

- Enrich users on signup (address validation)
- Geo-enrich items on placement (city, state, lat-lon cell identification)
- Notify users on item sold or reserve not met
- Perform analytics on auction when item ‘completes’`

In-band but edge

- Notify user of items-of-interest from their history when browsing
- Ad-placement analytics (watched items, interested items, users-purchased

ad-hoc

- Monte-carlo auction simulations to guide users and calc item trending scores

Out of band, edge

confluent

Patterns for Infrastructure

1. **Ops:** (Observability, instrumentation, metrification) = monitoring patterns, dead-letter-queues, error-queues, audit-logs, application-logging, data-lineage

2. **Stream-based:** Worker Queue (compute grid or faas), event-backbone

3. **Data-based:** K/V store, Queryable Data fabric,
   Data virtualization (via connect cdc streams) → Data Fabric

4. **FaaS**: fire and forget, event sourcing, unit-of-work

confluent

Key takeaways:

- **Model**: Events as the API, model the use-cases, model for scale, evolve the data-model : DDD
- **As Streams**: Streams are the database, tables materialized views, architect for evolutionary apps by using events
- **App Infra**: Build patterns to underpin higher order models (metrics, k/v etc)
- **FaaS**: Ad-hoc and edge stream processing or pinned processors