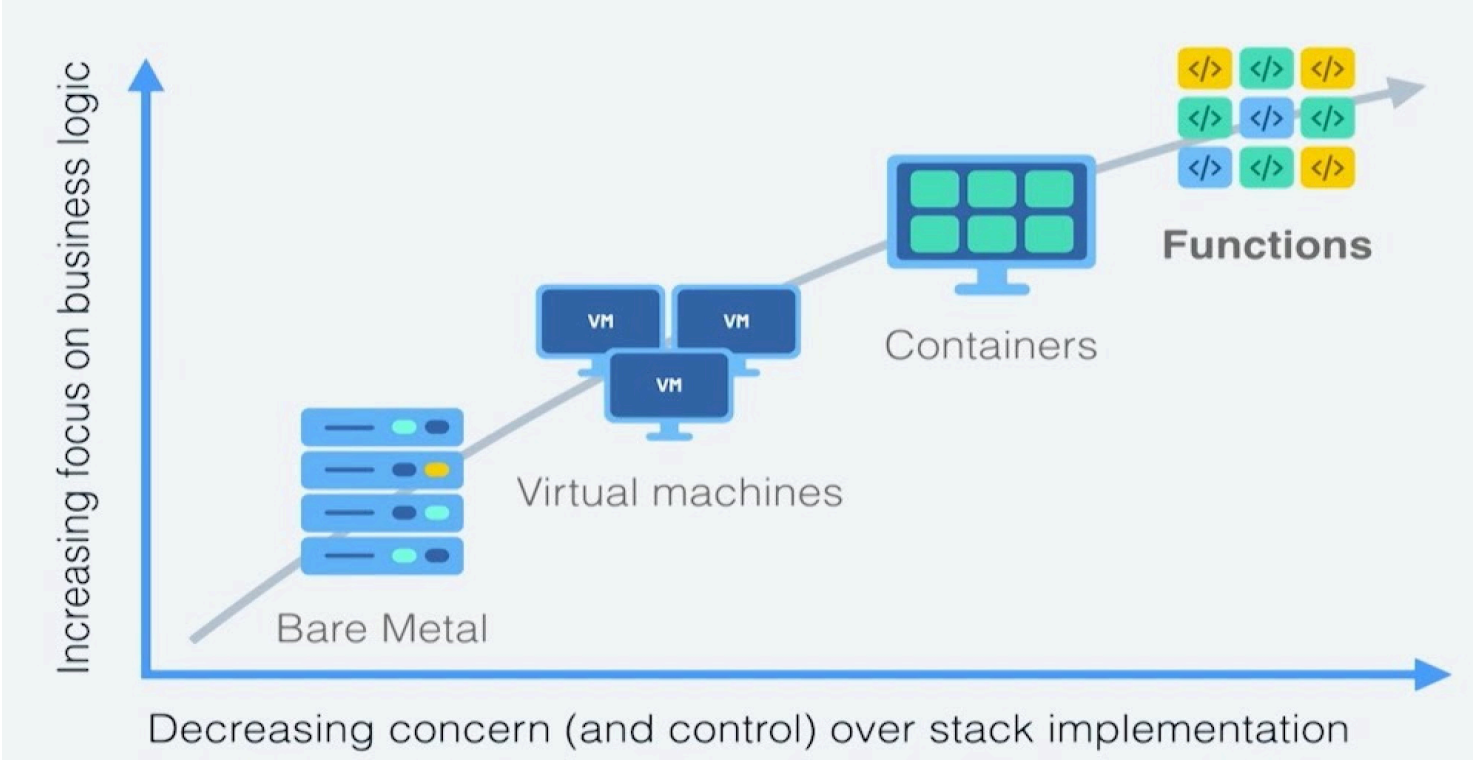
Stateful Functions 2.0

Stream Processing meets Serverless Apps

Serverless



Data Center

- Hardware based scaling
- Abstracts the physical hosting environment

IaaS

- Operating system based scaling
- Abstracts the hardware

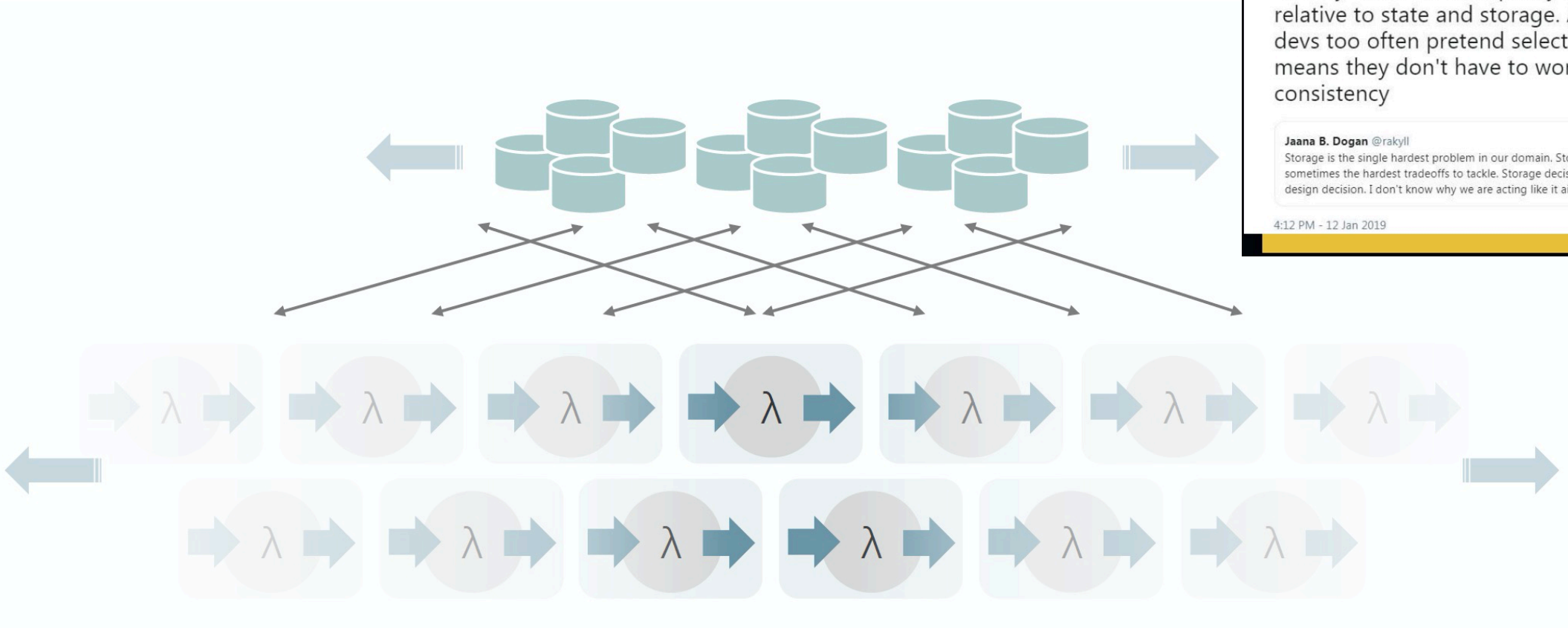
PaaS

- Application based scaling
- Abstracts the operating system

Serverless

- Function based scaling
- Abstracts the language runtime

Problem of data handling



tim gross
@0x74696d

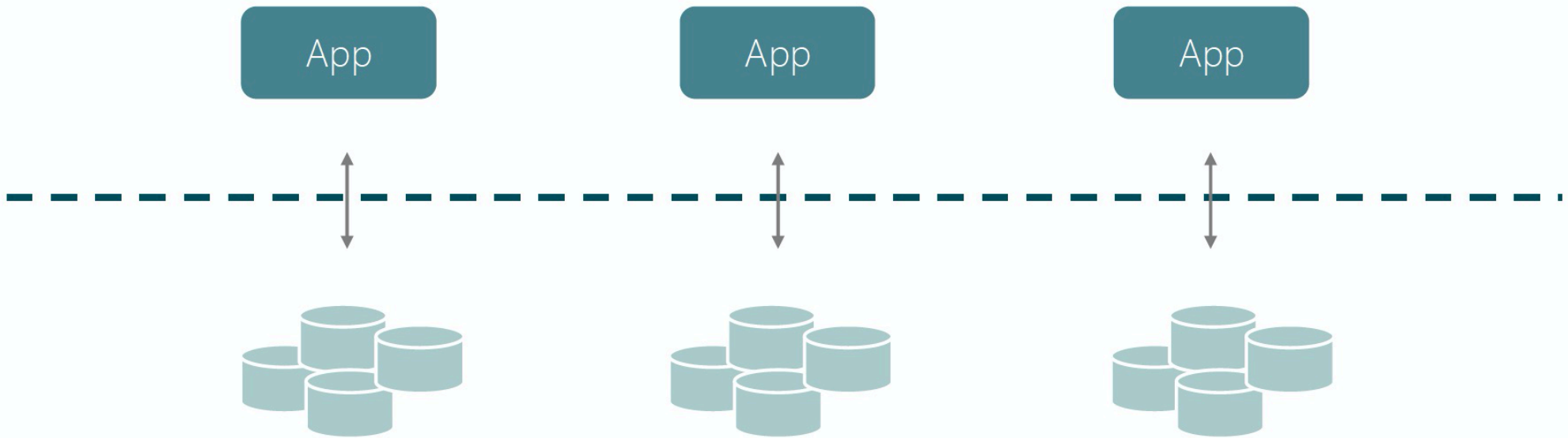
Follow

OMG yes! So much energy being poured into orchestrating stateless applications. That isn't *totally* trivial but it's pretty damn close relative to state and storage. And application devs too often pretend selecting a RDBMS means they don't have to worry about state consistency

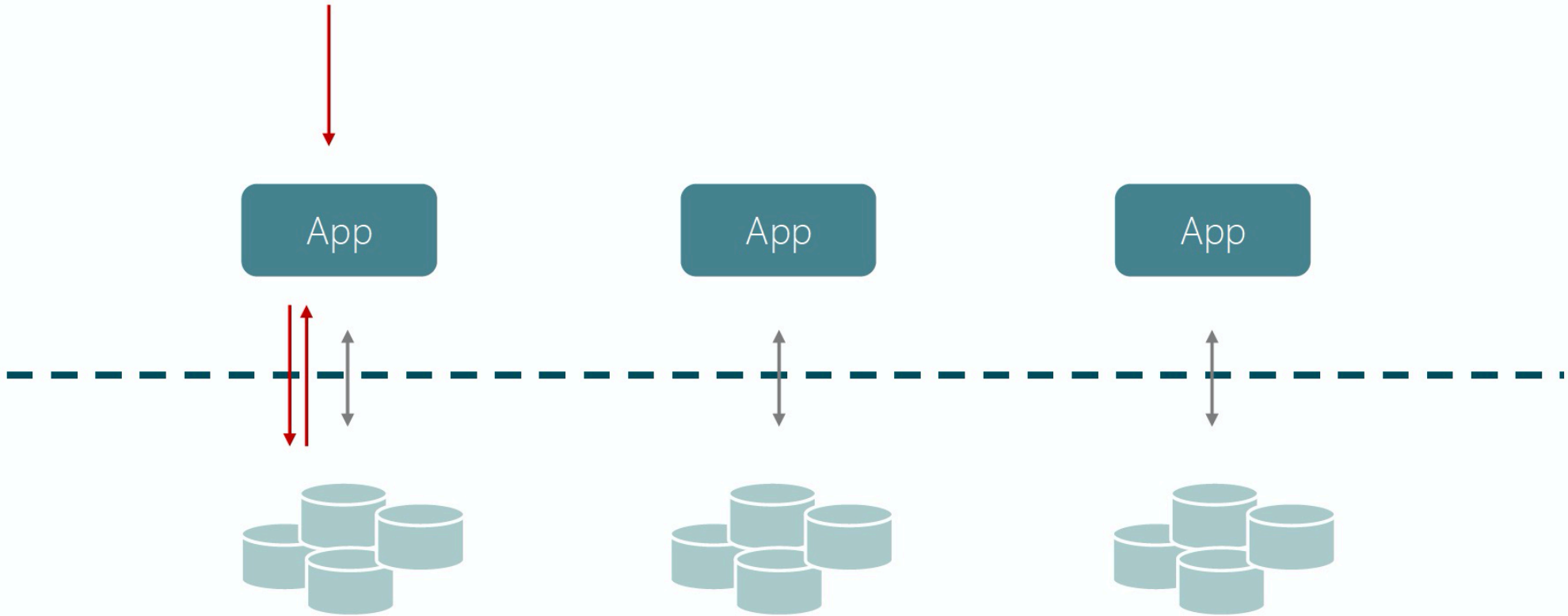
Jaana B. Dogan @rakyll
Storage is the single hardest problem in our domain. Storage related tradeoffs are sometimes the hardest tradeoffs to tackle. Storage decisions often impact every other design decision. I don't know why we are acting like it ain't so.

4:12 PM - 12 Jan 2019

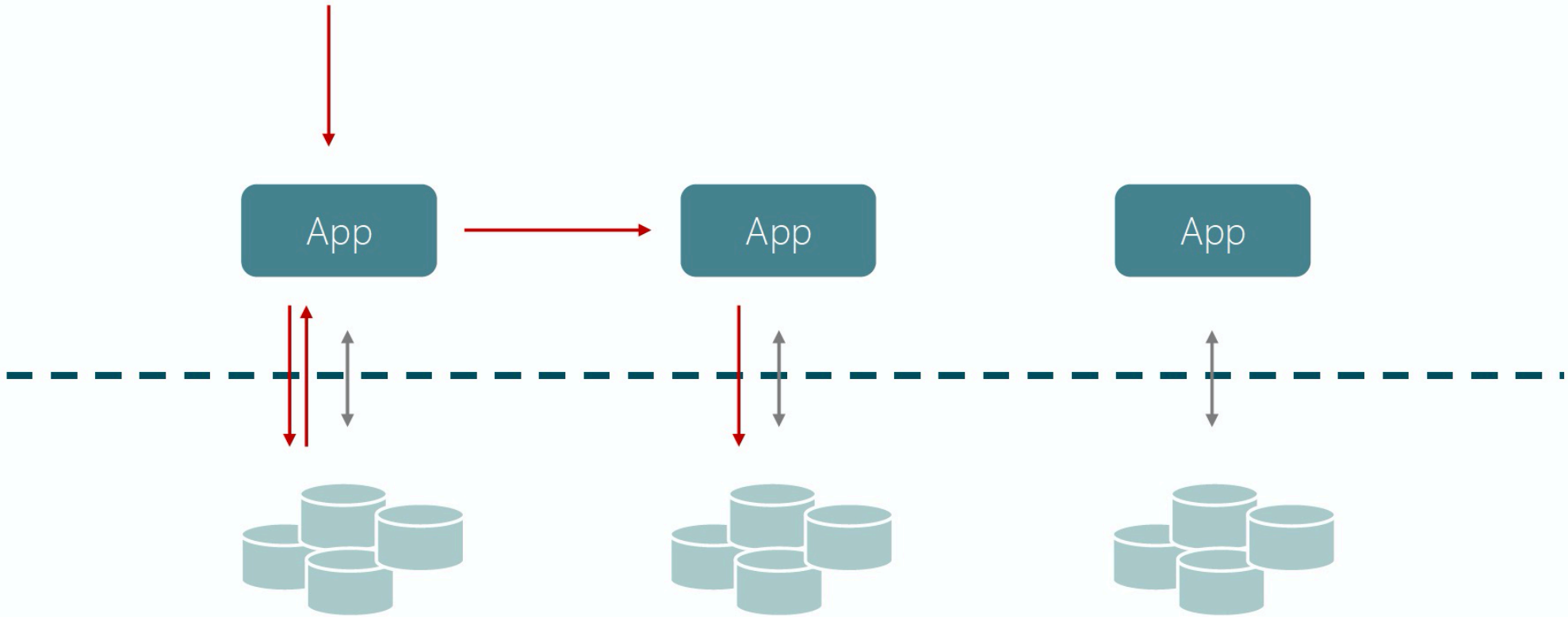
The Applications handles messaging (and consistency)



The Applications handles messaging (and consistency)

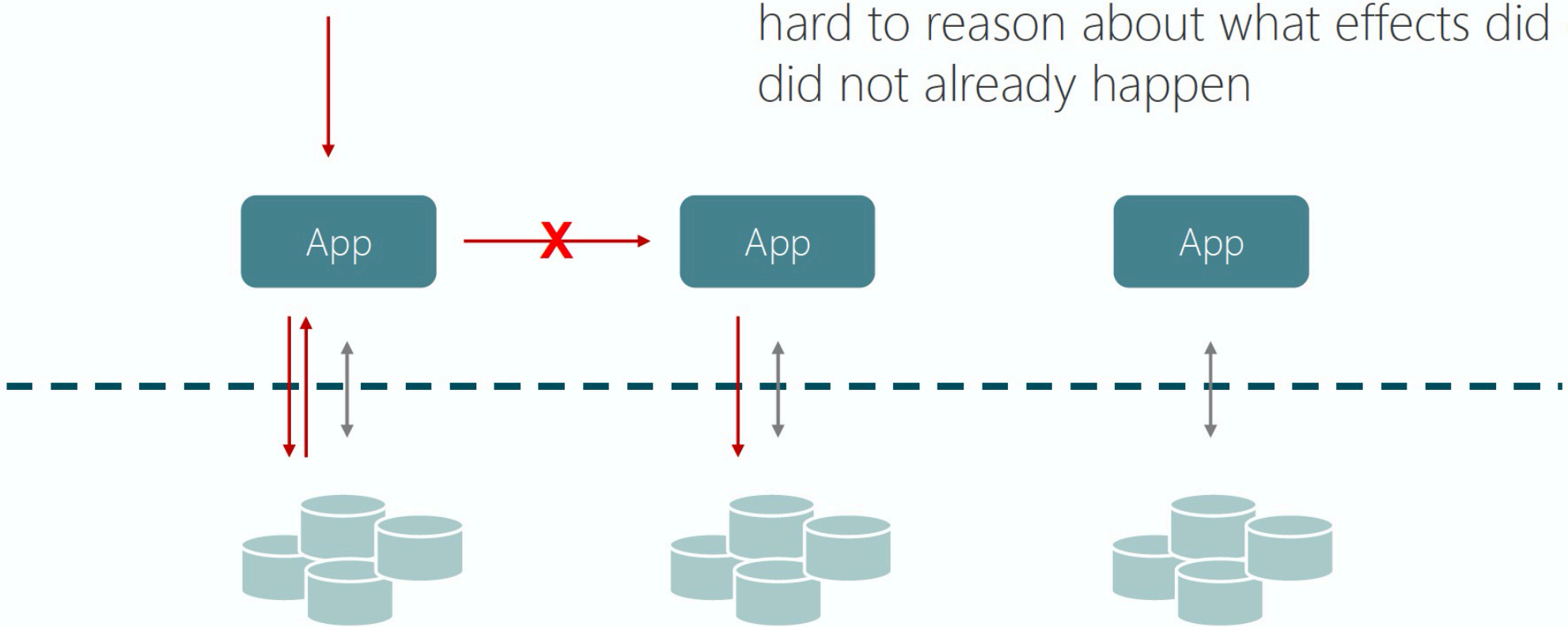


The Applications handles messaging (and consistency)

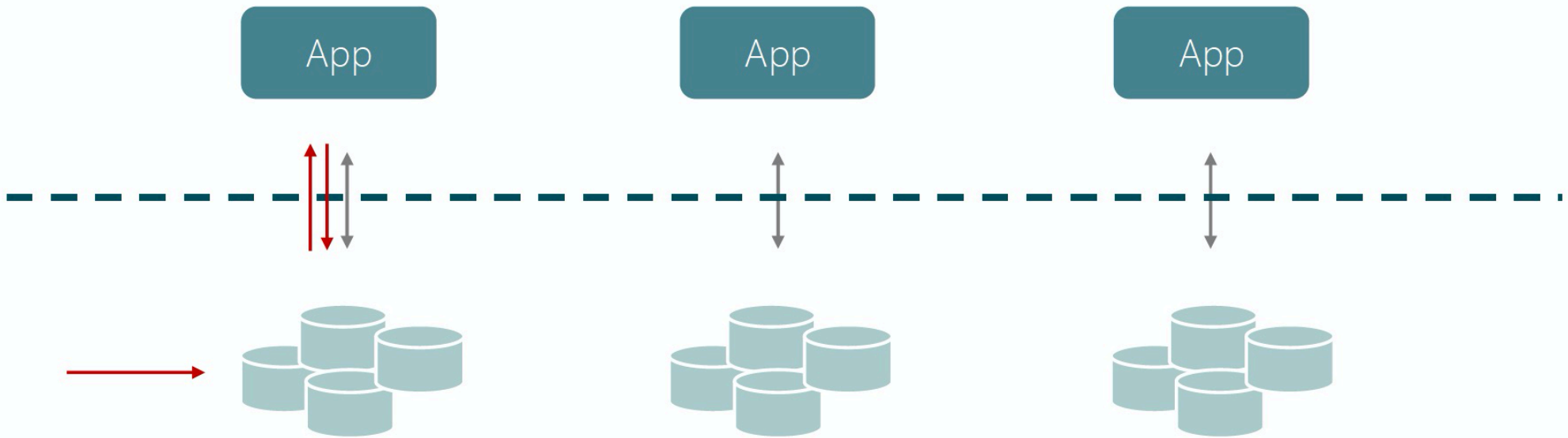


The Applications handles messaging (and consistency)

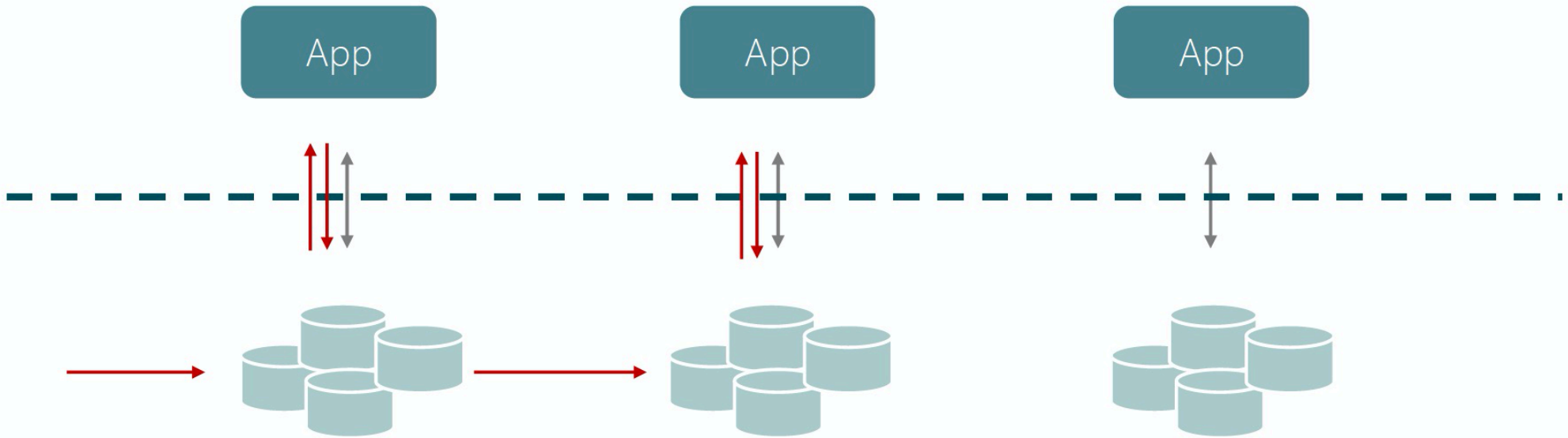
For any failure in any call, it becomes hard to reason about what effects did or did not already happen



The State Layer handles messaging (and consistency)

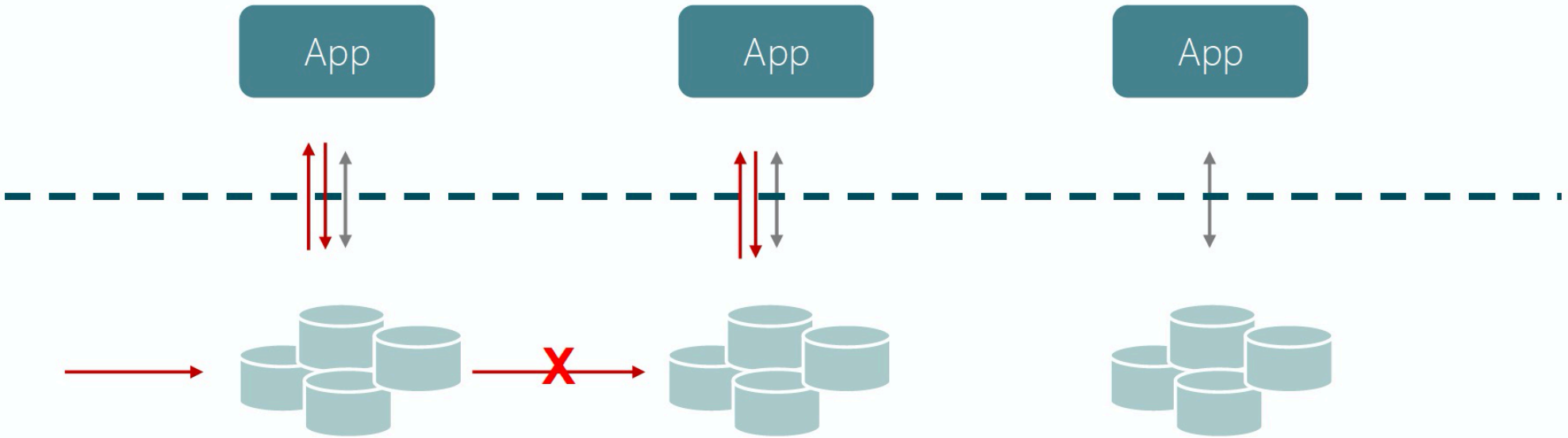


The State Layer handles messaging (and consistency)



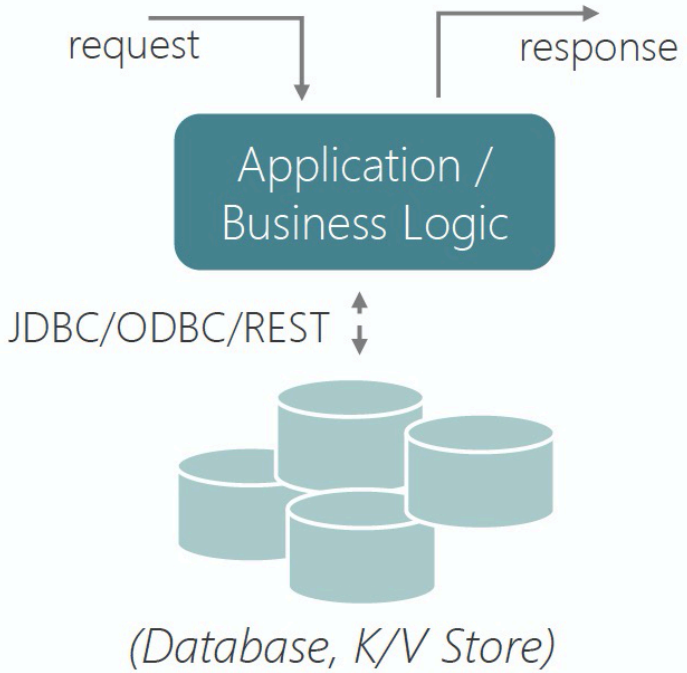
The State Layer handles messaging (and consistency)

application logic is pure/stateless,
so re-tries are always idempotent

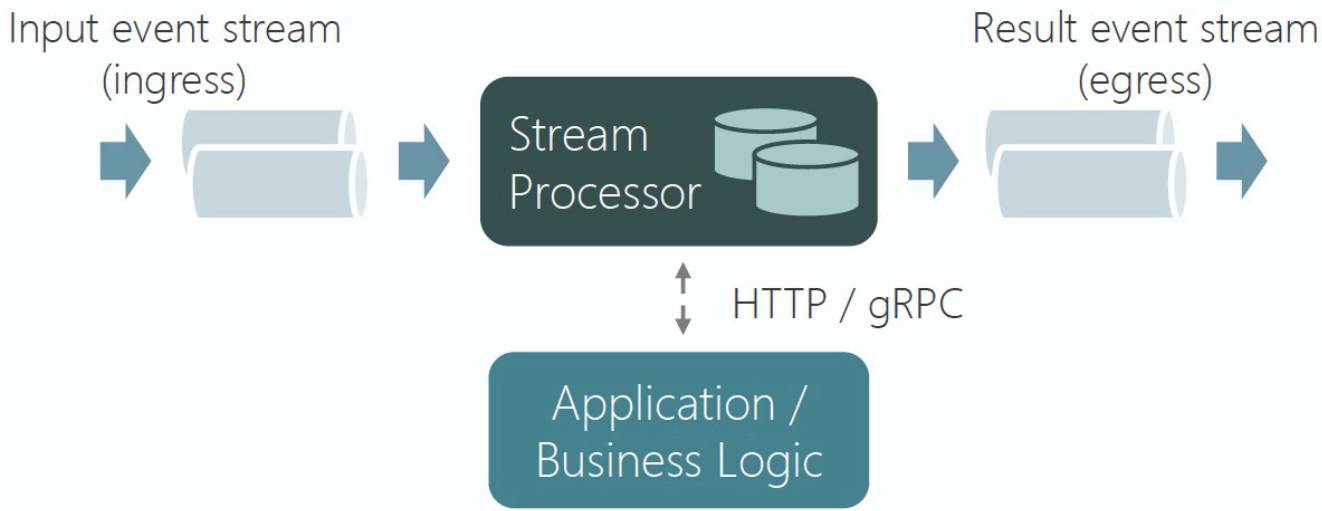


state updates and messaging are
easily made “atomic”

Compute / State Separation

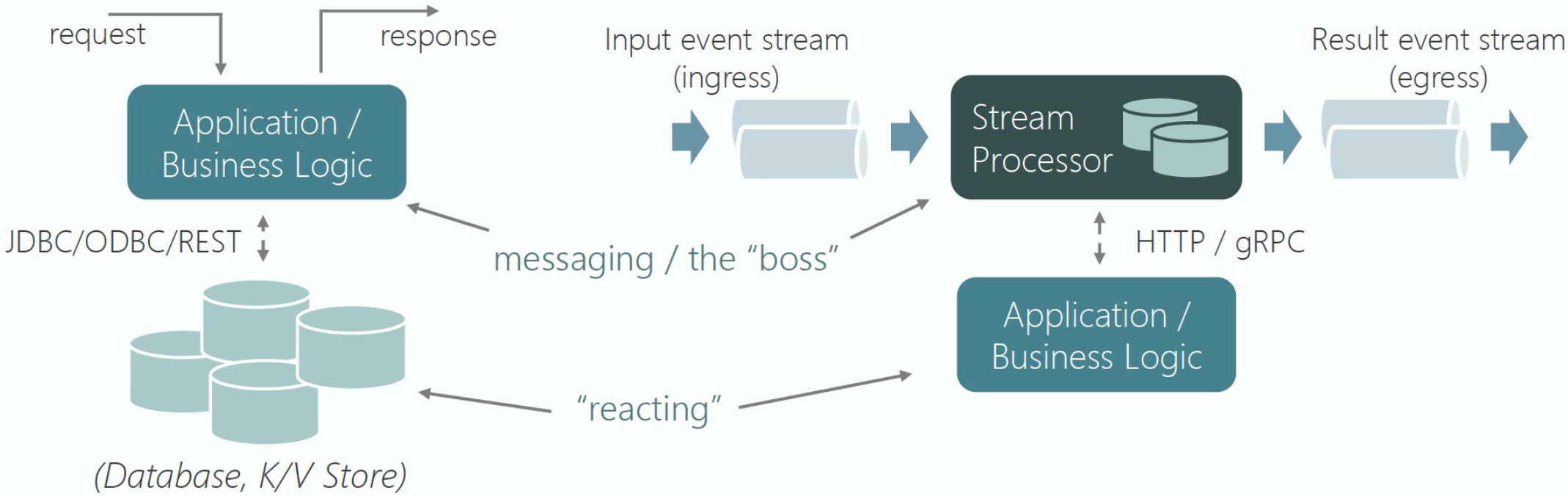


CRUD and Batch

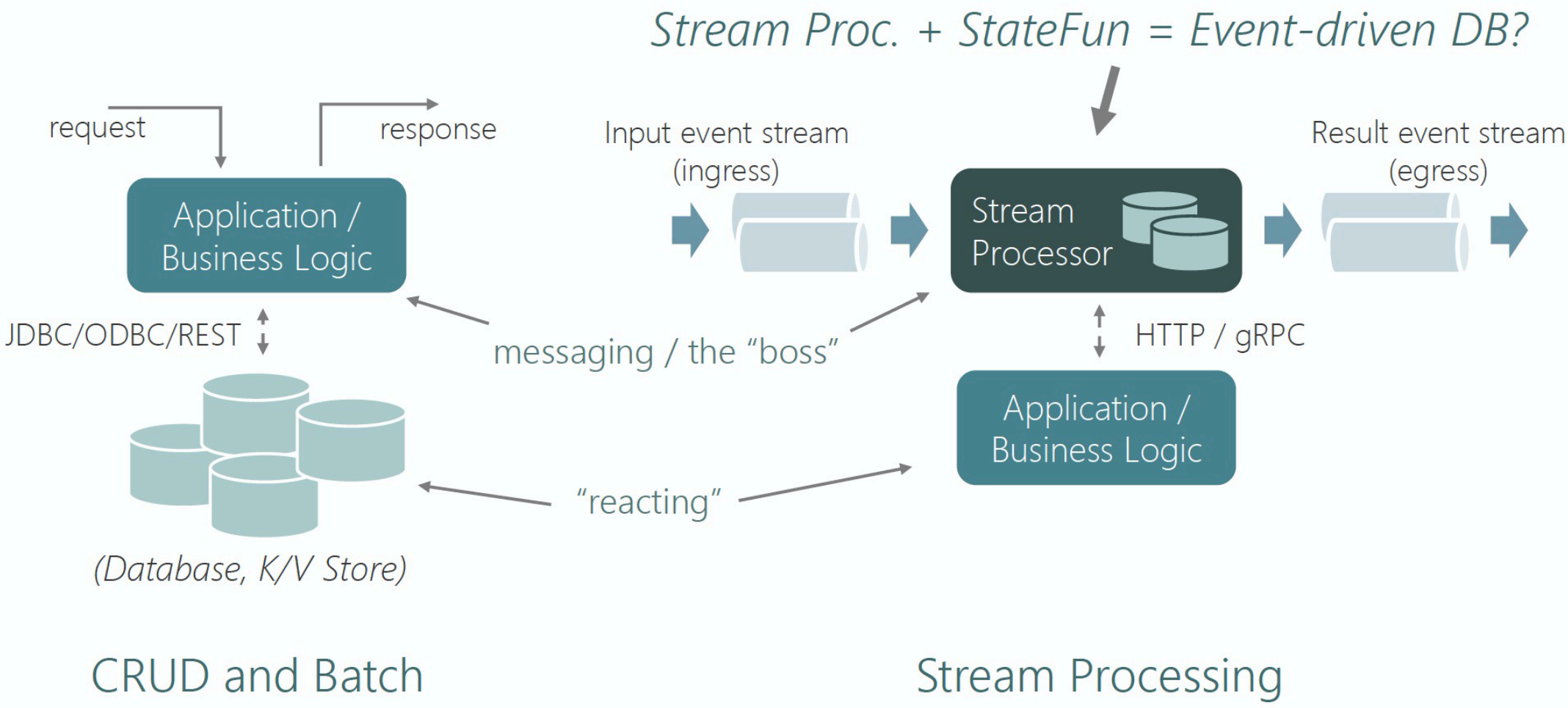


Stream Processing

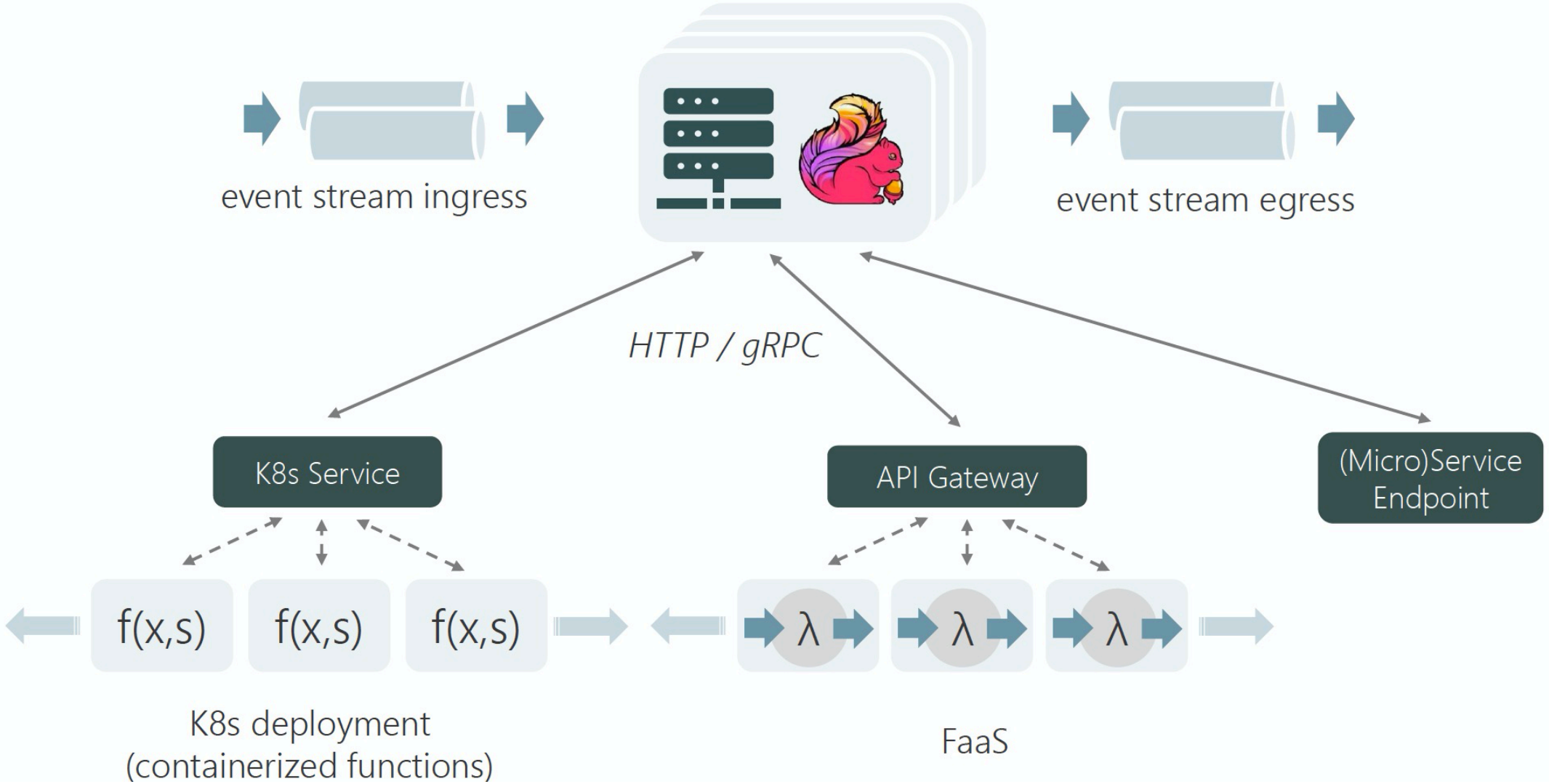
Compute / State Separation



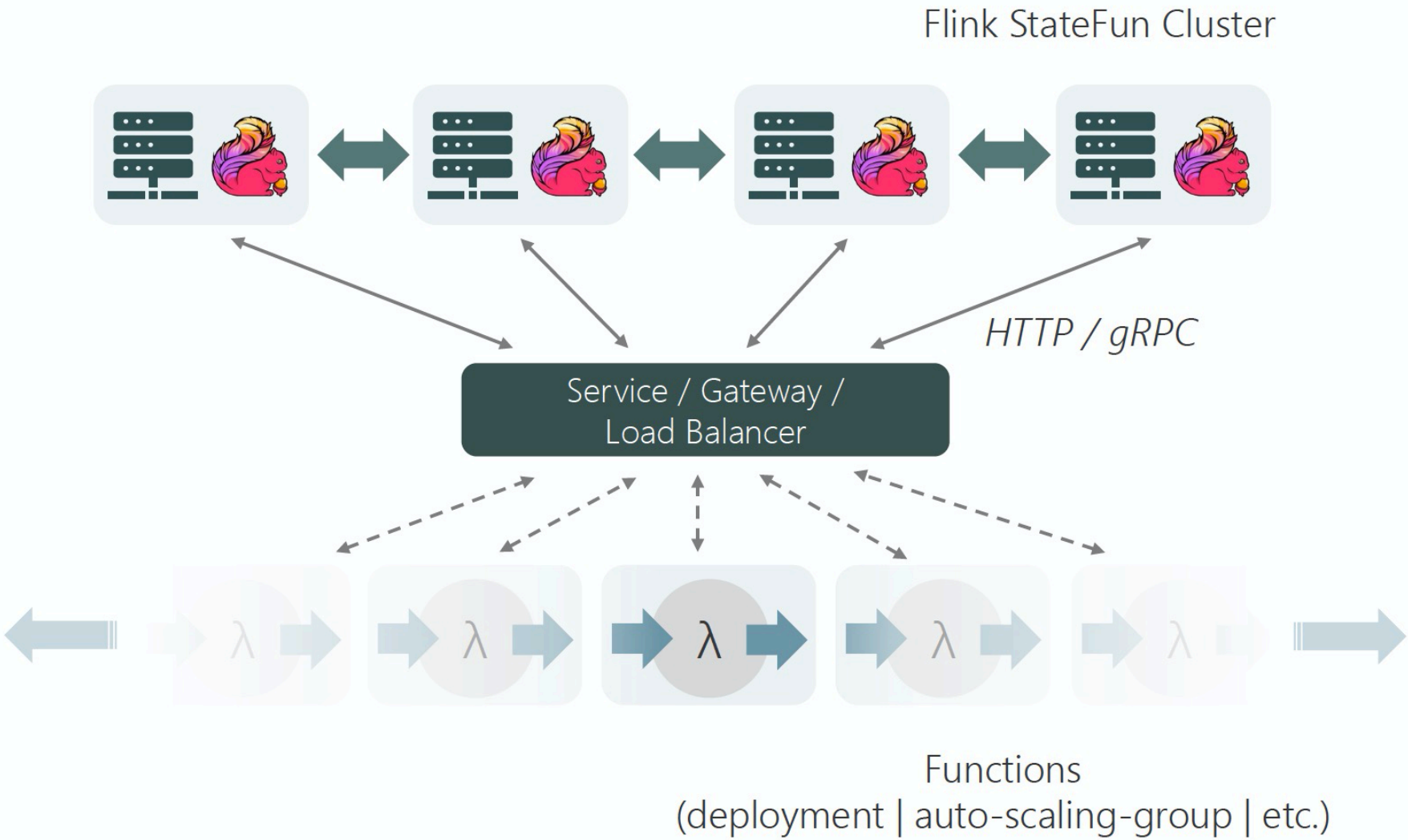
Compute / State Separation



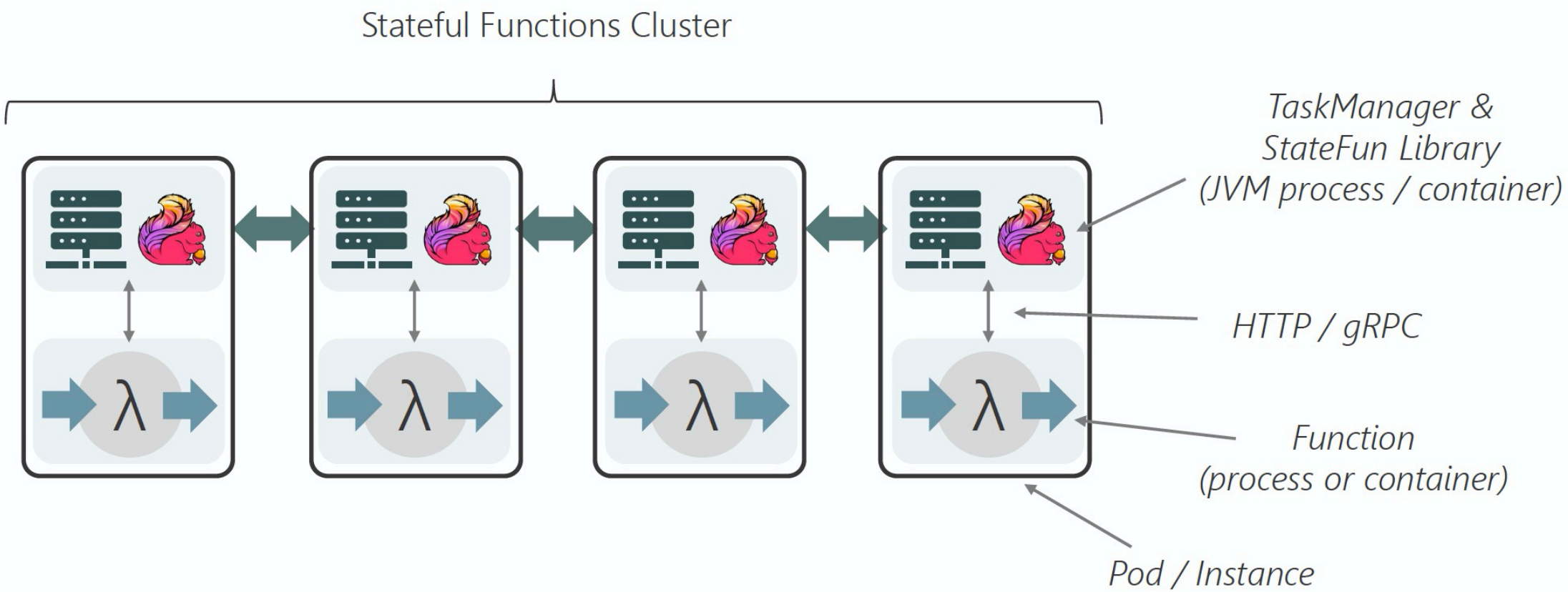
Stateful Functions 2.0



Remote Functions



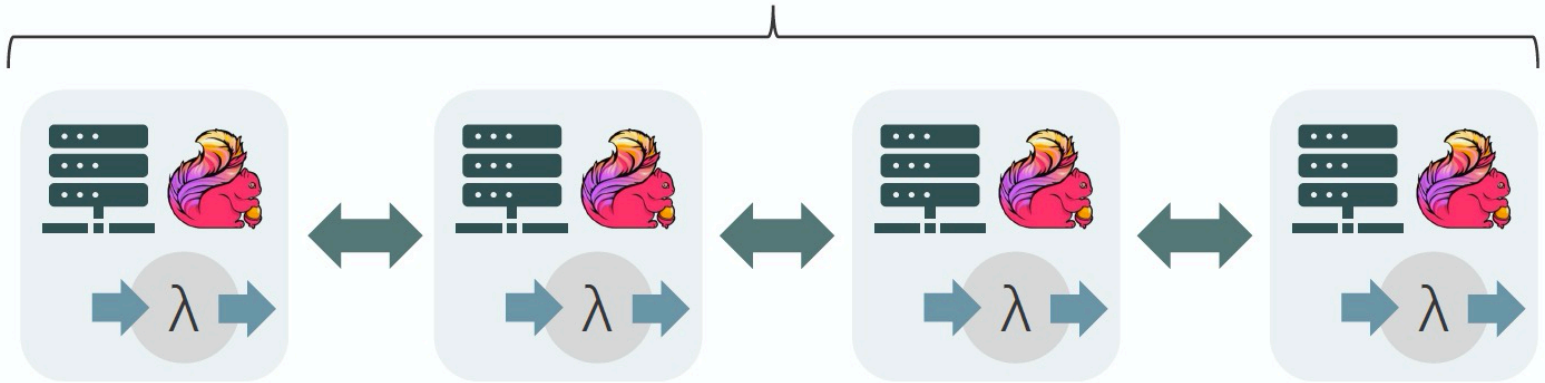
Co-located Functions



Embedded Functions

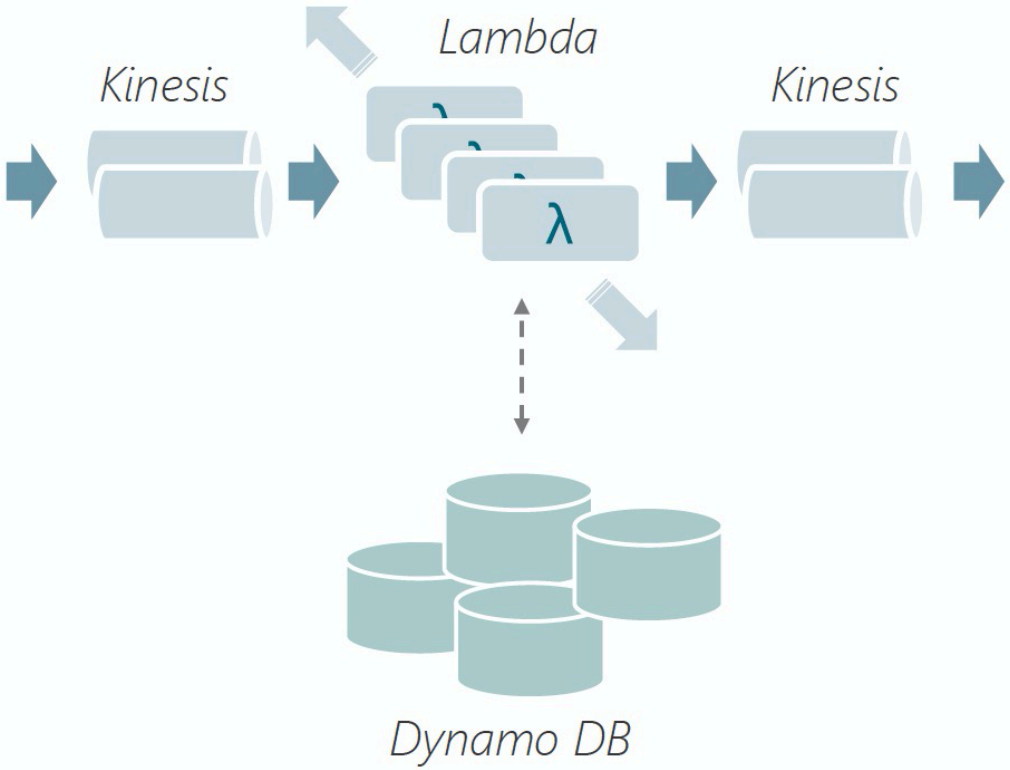
Stateful Functions 1.0

Stateful Functions Cluster

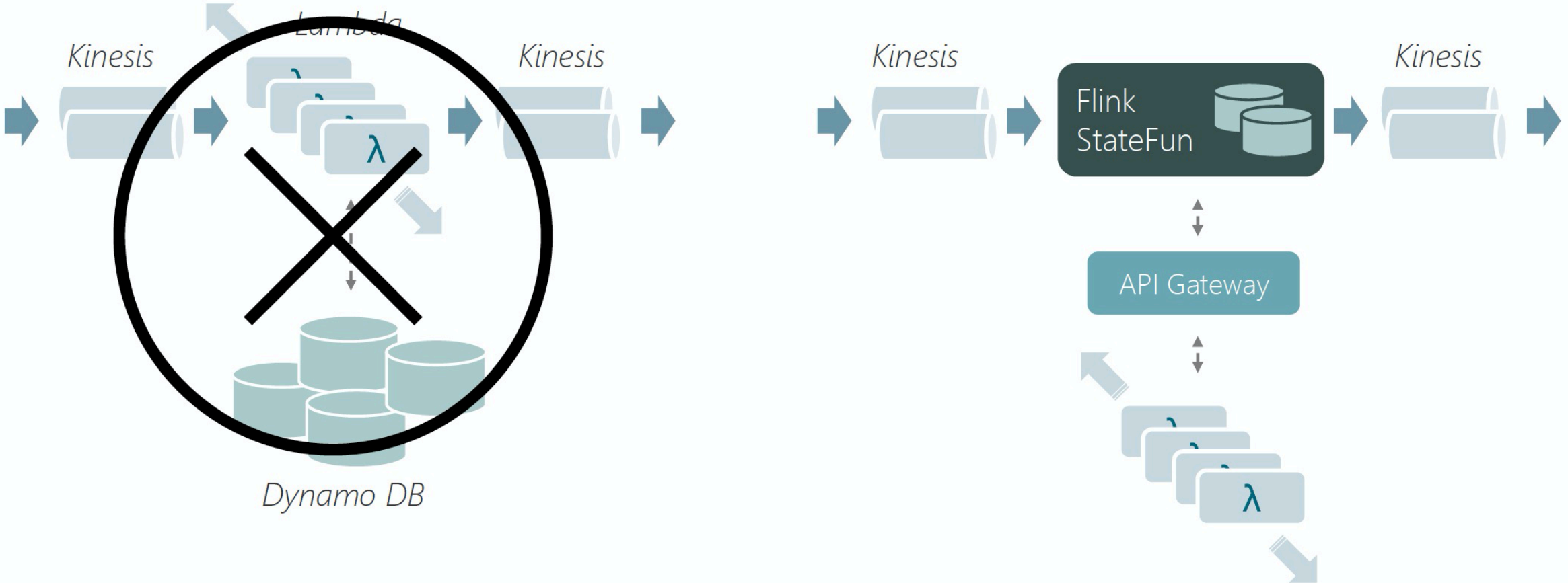


TaskManager &
StateFun Library
(JVM process / container)

Remote Functions with FaaS



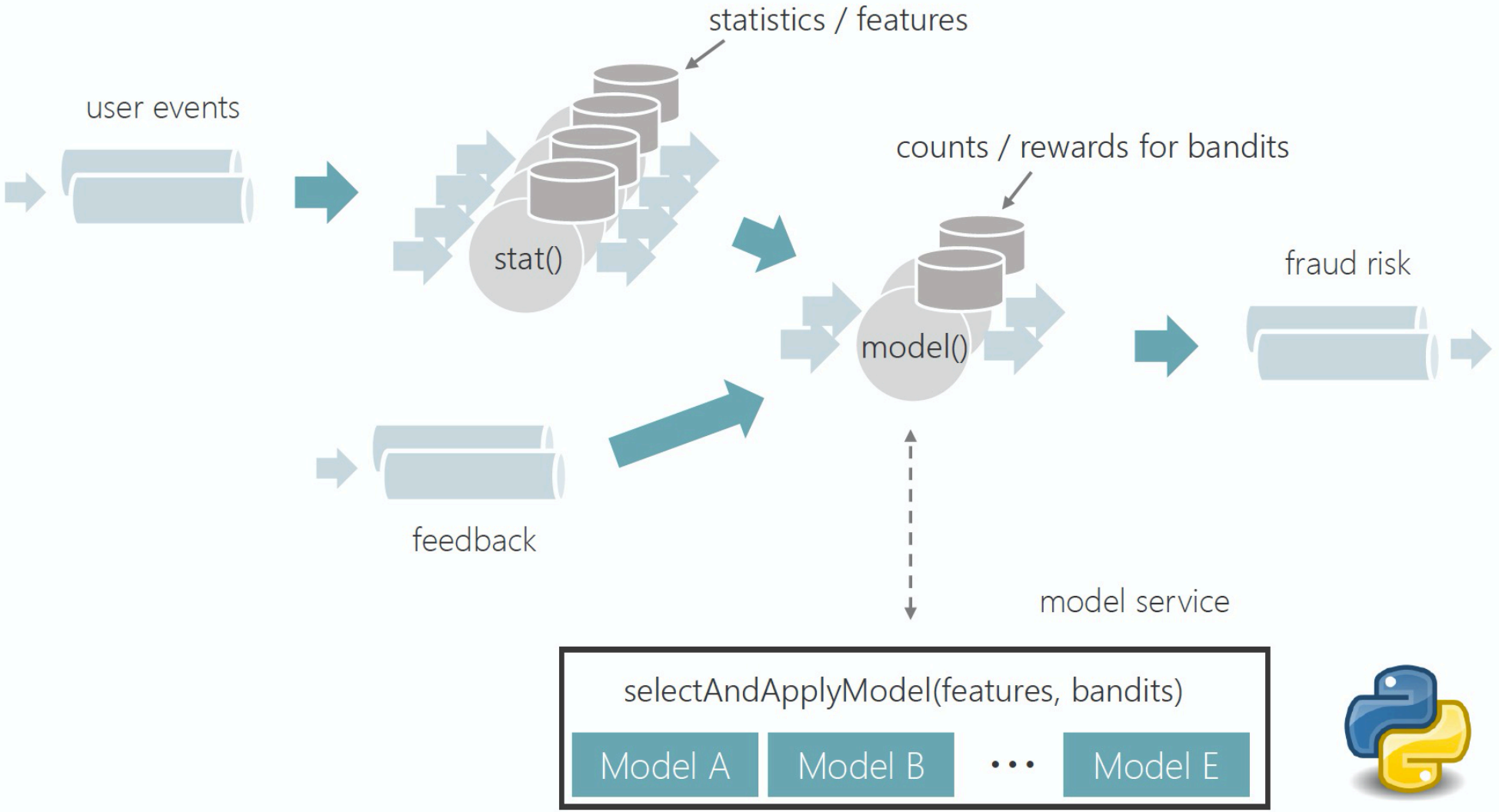
Remote Functions with FaaS



➔ Solves problem of consistent & scalable state

➔ Solves messaging / composition

Demo: Machine Learning - Feature Vectors & K-Bandits



Demo: Machine Learning - Feature Vectors & K-Bandits

```yaml
version: "1.0"
module:
  meta:
    type: remote
  spec:
    functions:
      - function:
          meta:
            kind: http
            type: demo/model
          spec:
            endpoint: http://python-worker:8000/statefun
            states:
              - arms
            maxNumBatchRequests: 500
            timeout: 2min
```

```python
@functions.bind("demo/model")
def model(context, message):
    if message.Is(SparseVector.DESCRIPTOR):
        handle_prediction_request(context, message)
    elif message.Is(PredictionFeedback.DESCRIPTOR):
        handle_prediction_feedback(context, message)
    else:
        raise ValueError('unknown message type ' + message)


def handle_prediction_request(context, message):
    # select the best model so far for this user
    bandit = get_from_state(context)
    model_index = bandit.choose_arm()

    # use the selected model to make a prediction
    result = PredictionResult()
    result.from_model = model_index
    result.prediction = MODELS[model_index].predict(message)

    # emit the prediction result to a Kafka topic
    out_record = kafka_egress_record(topic="predictions", value=result)
    context.pack_and_send_egress("demo/kafka", out_record)
```

StateFun means to open Flink to Event-driven App Use-cases

Streaming Analytics

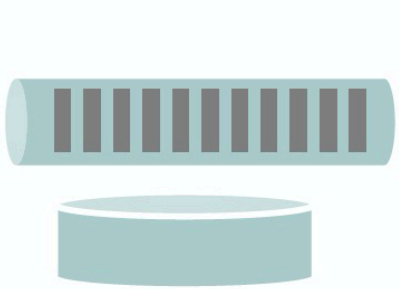
SQL and Tables

Stateful
Stream Processing

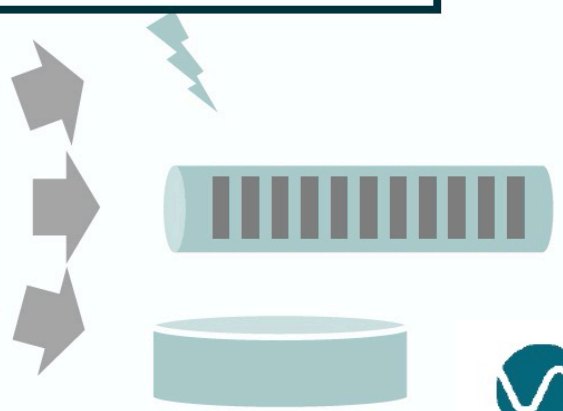
Streams, State, Time

Event-driven
Applications

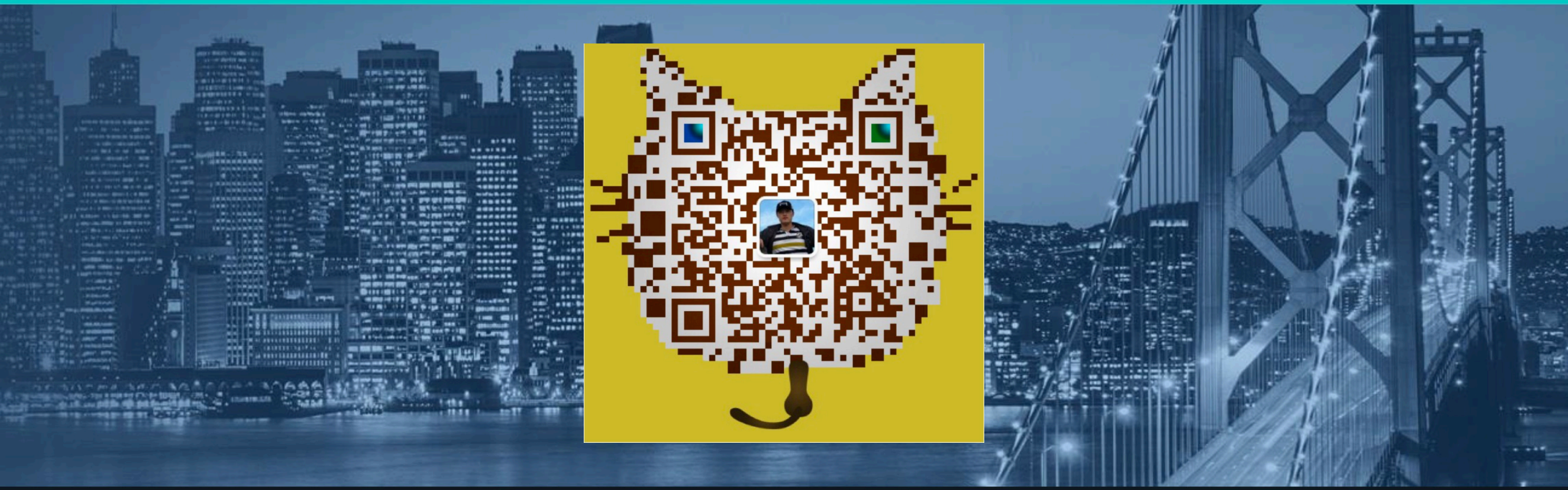
Stateful Functions



Flink Runtime
Stateful Computations over Data Streams



FLINK FORWARD



We hope you enjoy the first Virtual Flink Forward.