Leveraging Stateful Functions to Power the Next Generation of Event-Driven Applications

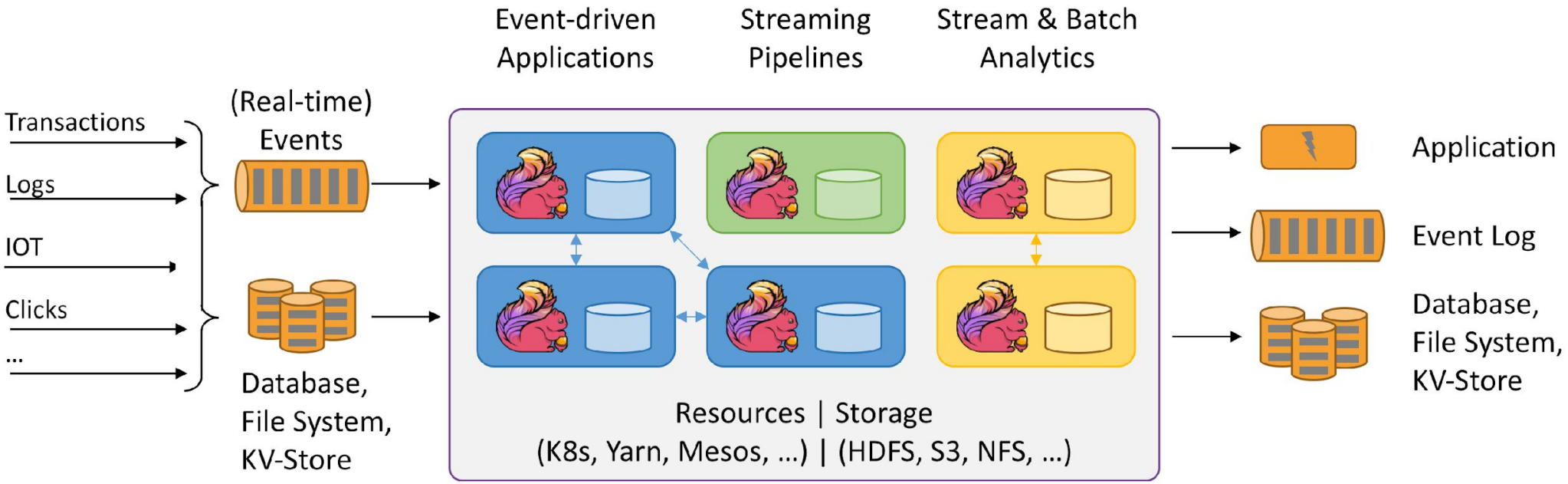
Seth Wiesman
@sjwiesman on most platforms

About Ververica

Original creators of
Apache Flink®

Enterprise
Stream Processing

Stateful Computations over Data Streams



Some Apache Flink Users

Sources: Powered by Flink, Speakers – Flink Forward San Francisco 2019, Speakers – Flink Forward Europe 2019

Apache Flink at Alibaba Group

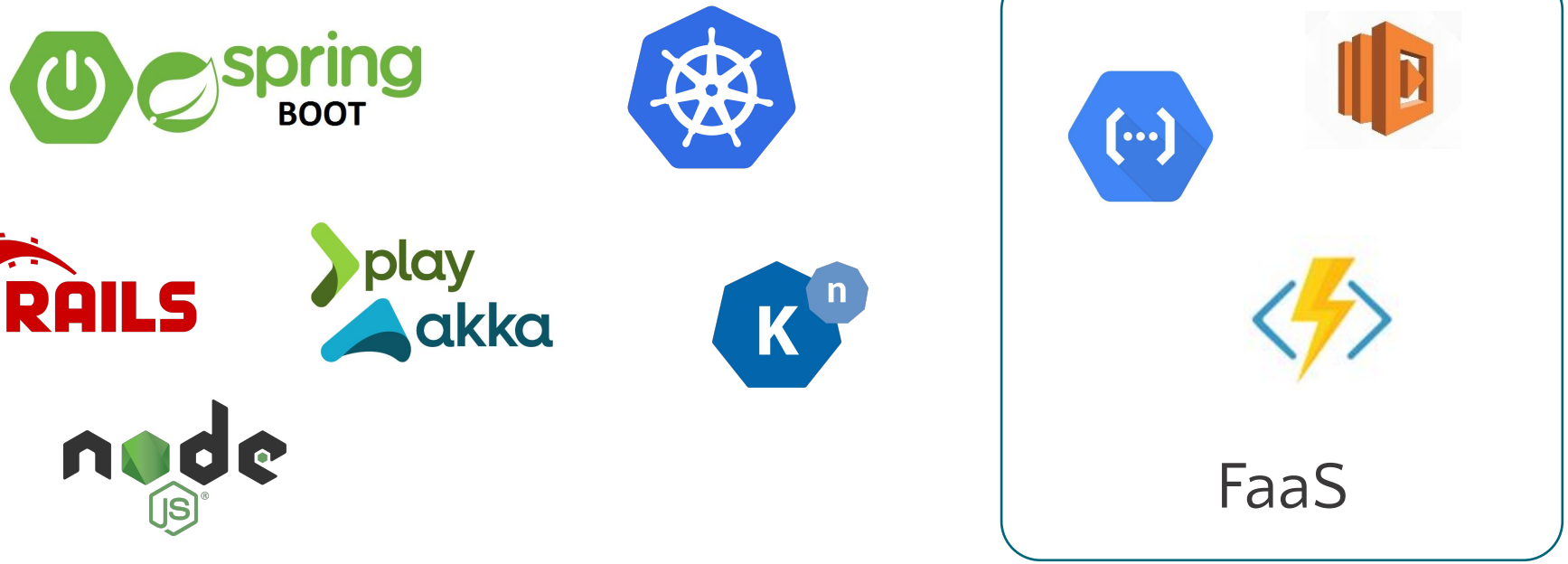
The "Singles Day" (11/11/2018)

machines	queries	throughput	latency	state size
10K	10K	1.7B events / sec	Sub-Second	100TB

Let’s look at building Applications

Building an Application Today

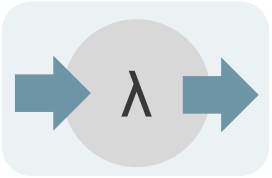
Building an Application Today



The big trend: Serverless

Functions as a Service

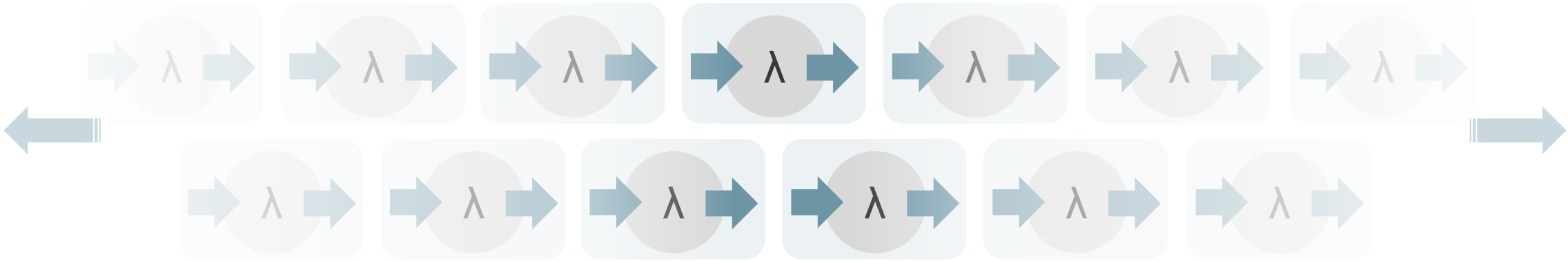
an event-driven function



Functions as a Service

elastically scalable

“lightweight resource footprint”



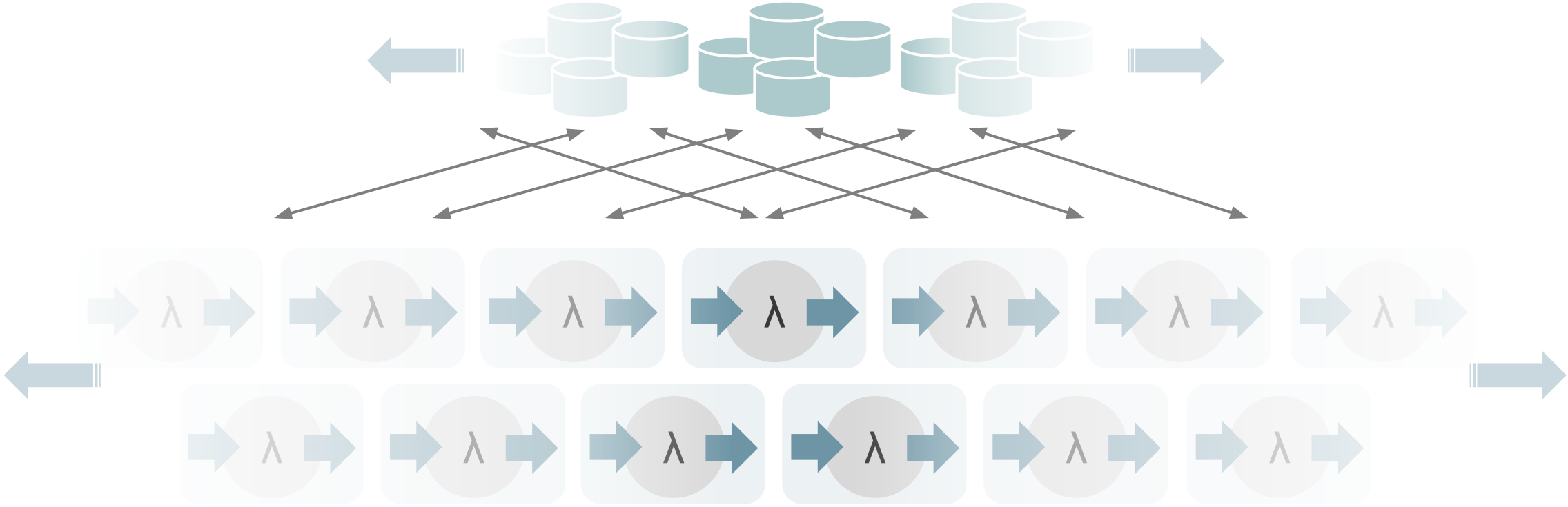
Functions as a Service – Handling State in Applications

state consistency ?

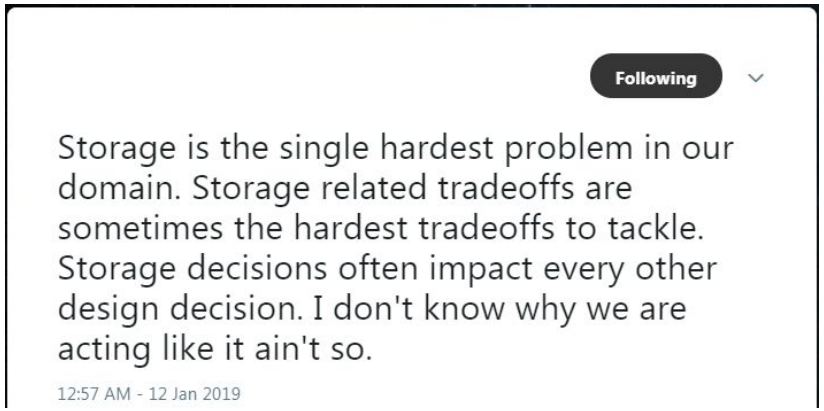
often bottlenecked by state access & I/O

scaling the database?

connections, request rates, ...



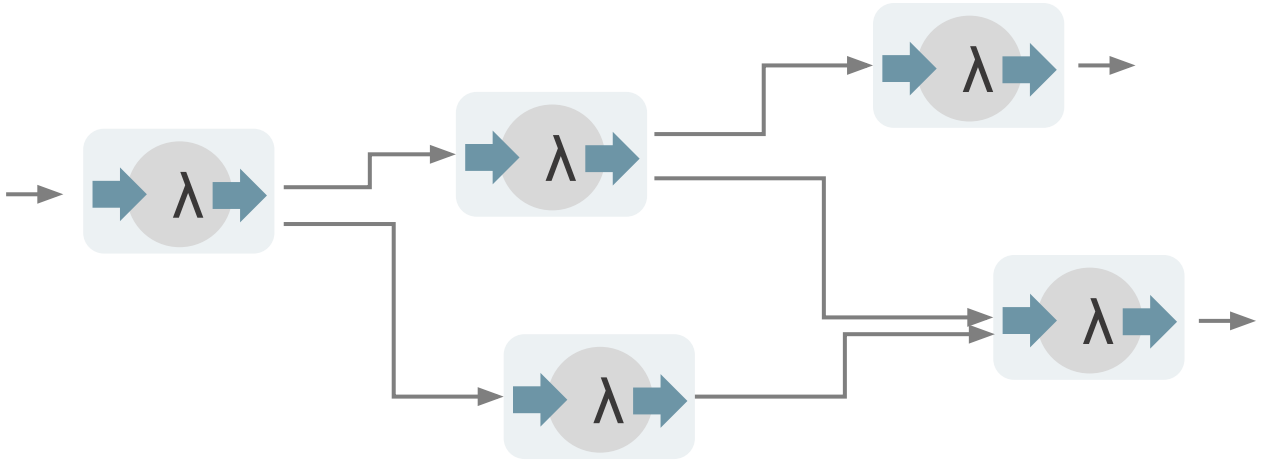
Functions as a Service – Handling State in Applications



Composition of Functions

Not straightforward to build more complex applications

Lack of messaging / composition primitives

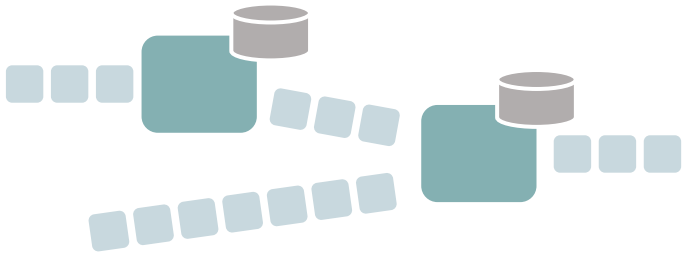


workflows of functions as a workaround, but not a general solution

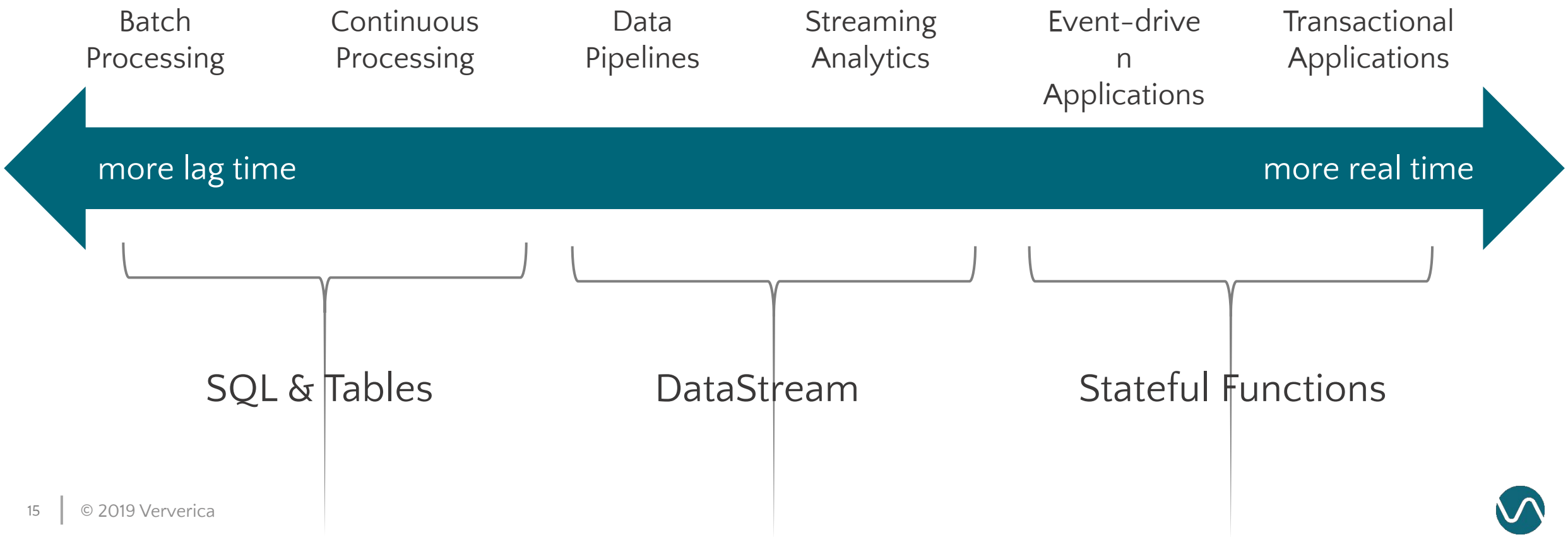
event-driven composable state management

...that sound like...

Stream Processing



The Spectrum of Streaming Data Use Cases



Disclaimer

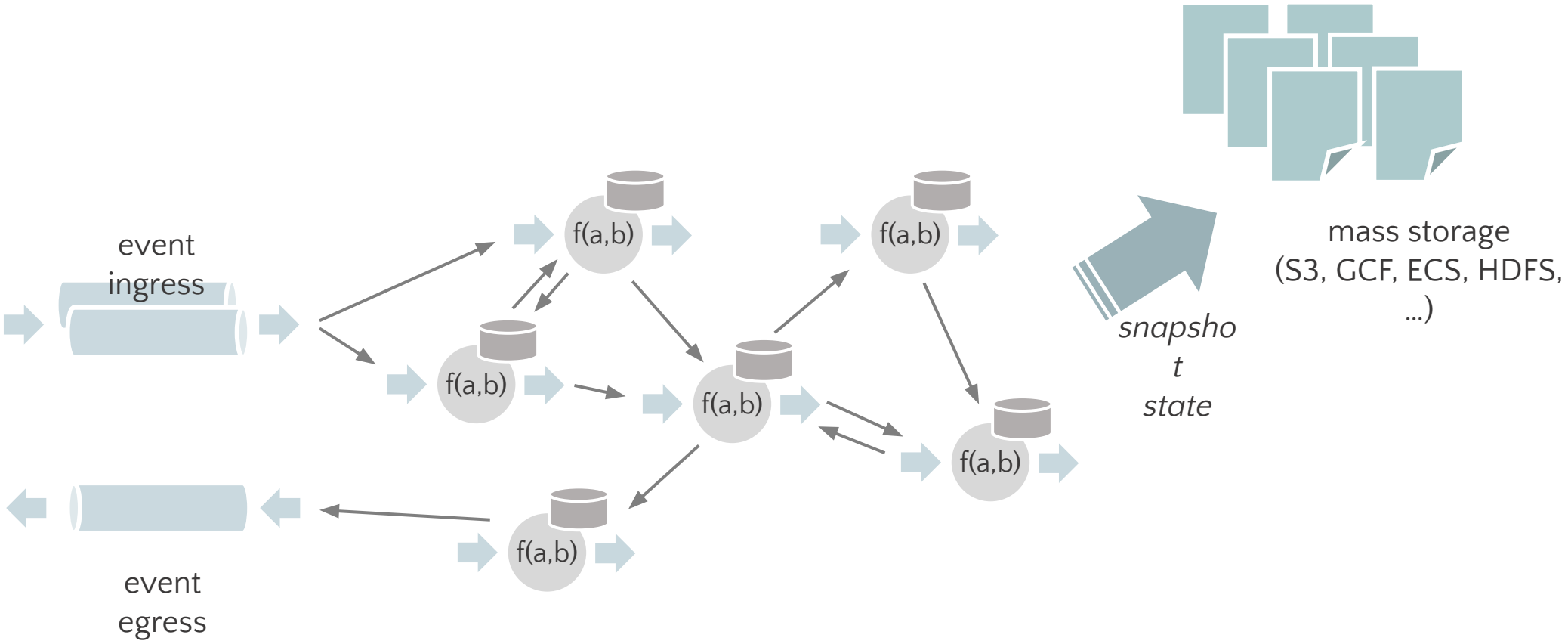
Stateful Functions is currently a standalone project

https://statefun.io/
https://github.com/ververica/stateful-functions

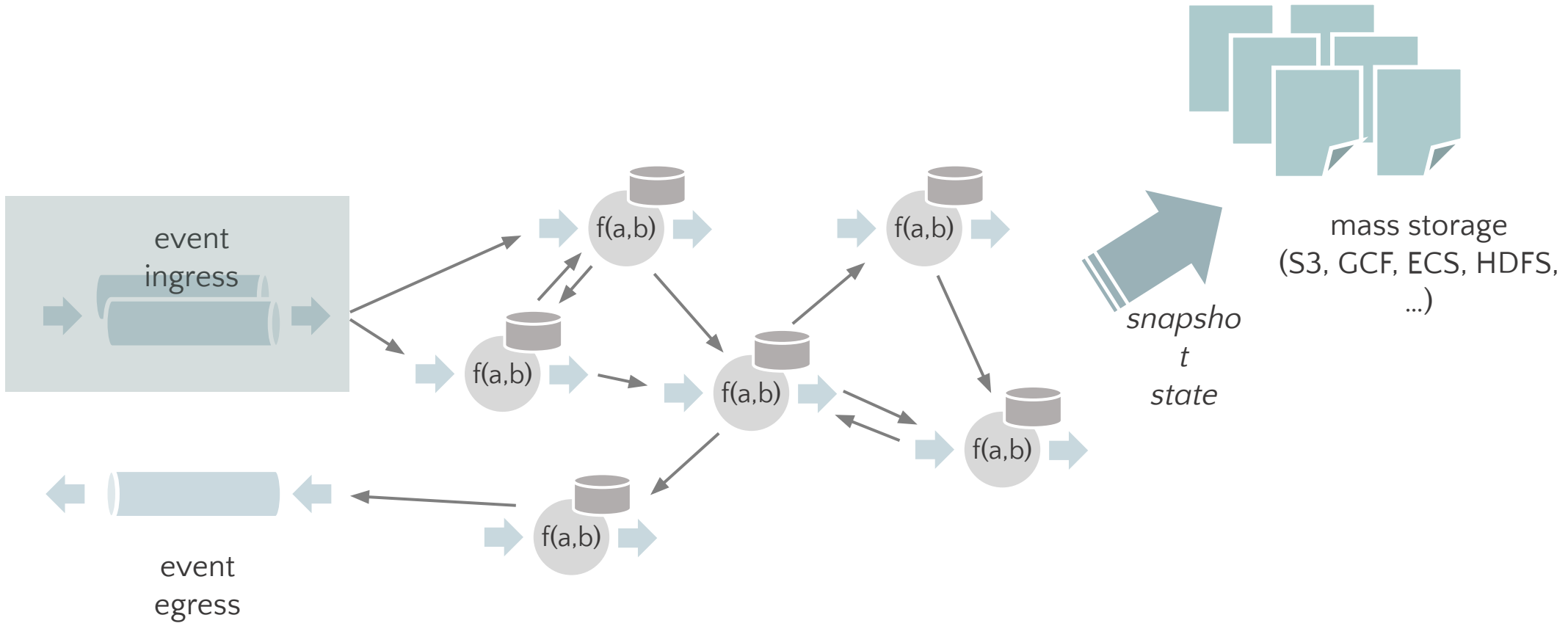
The Apache Flink community has voted to adopt

The project is still new and dynamic.
A good time to get involved to get traction ;-)

Stateful Functions

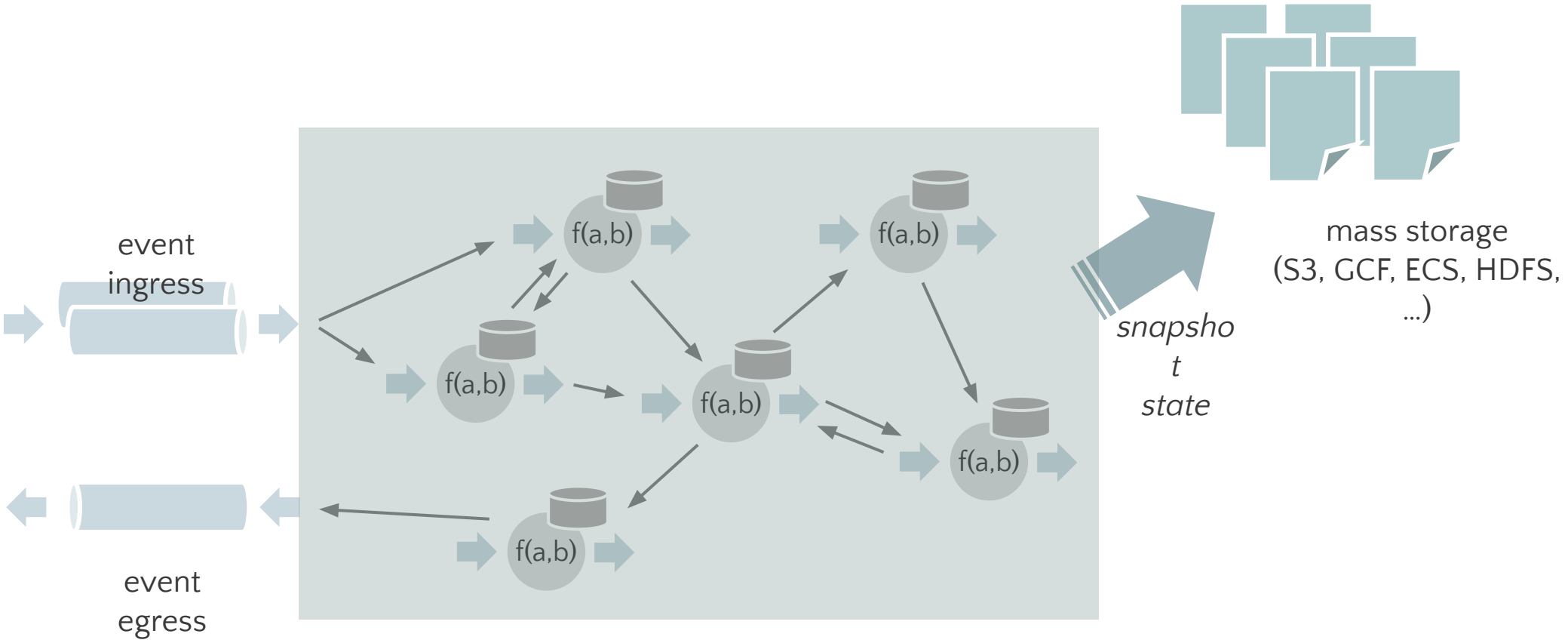


Stateful Functions



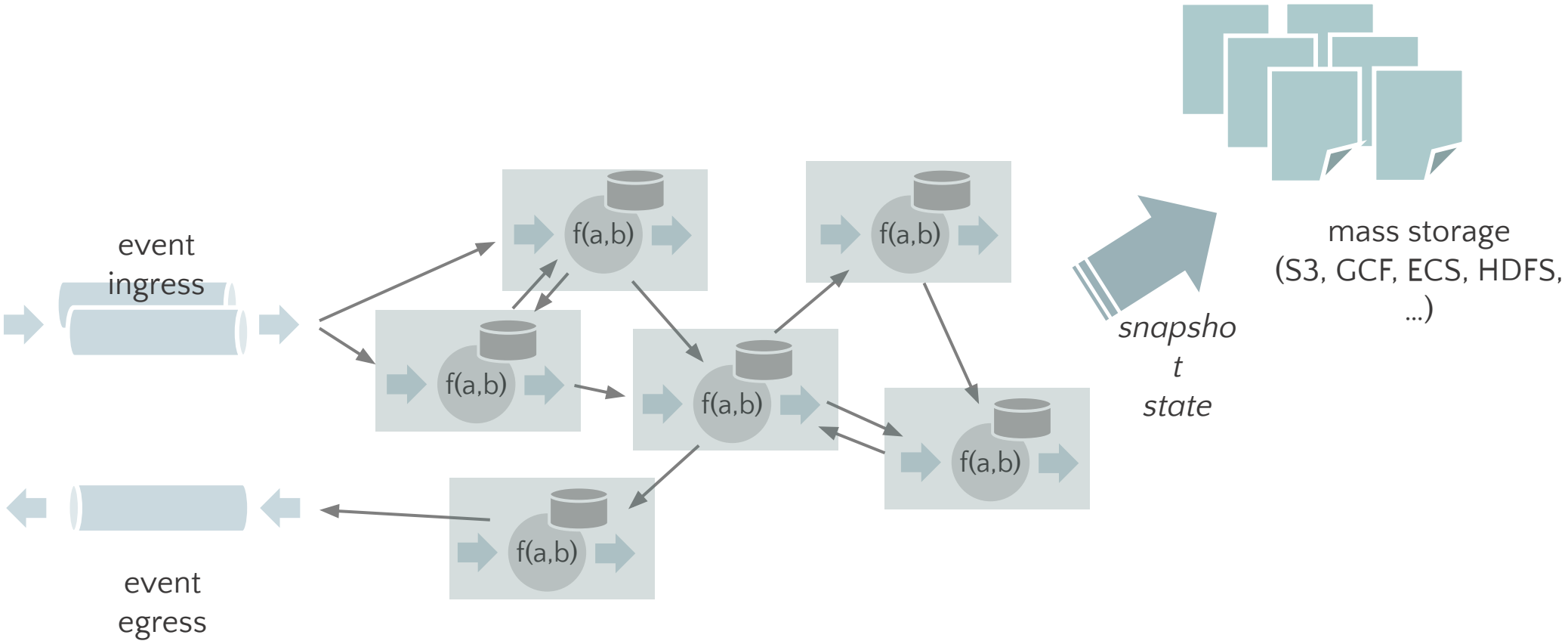
Event ingresses supply events that trigger functions

Stateful Functions



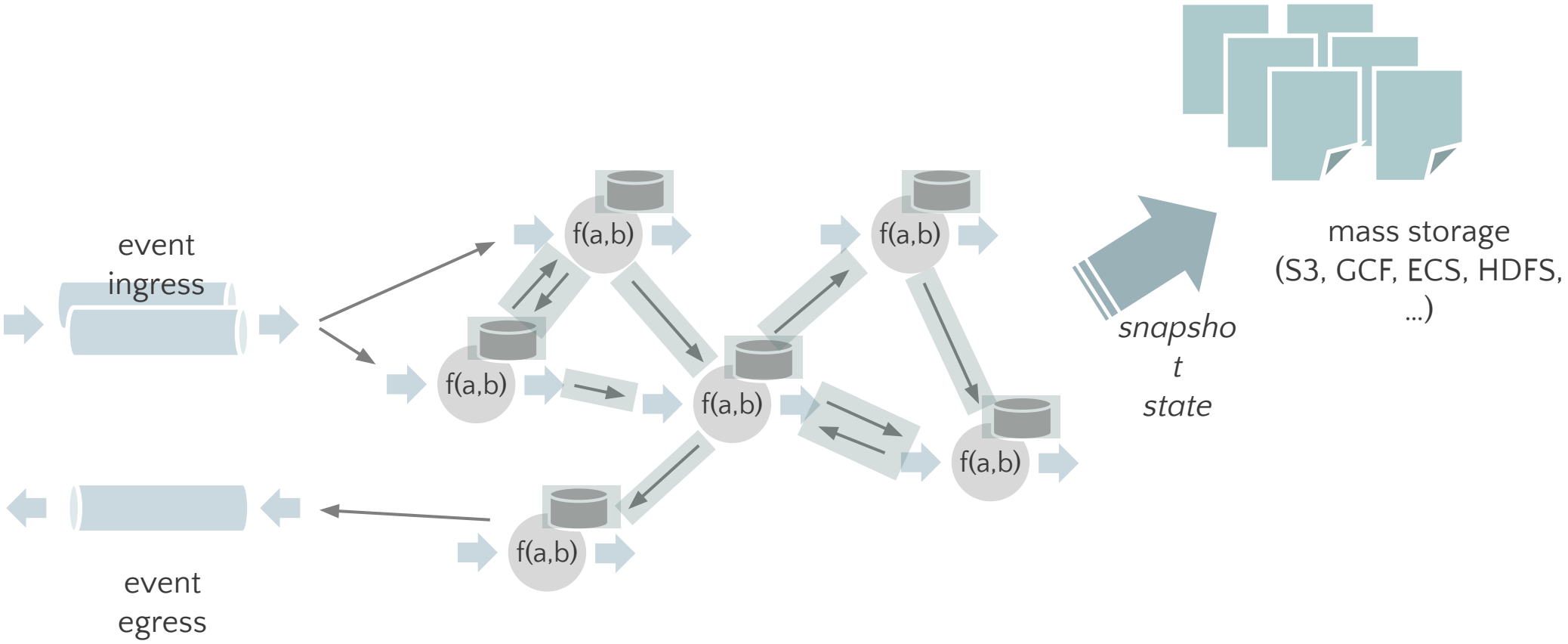
Multiple functions send event to each other
Arbitrary addressing, no restriction to DAG

Stateful Functions



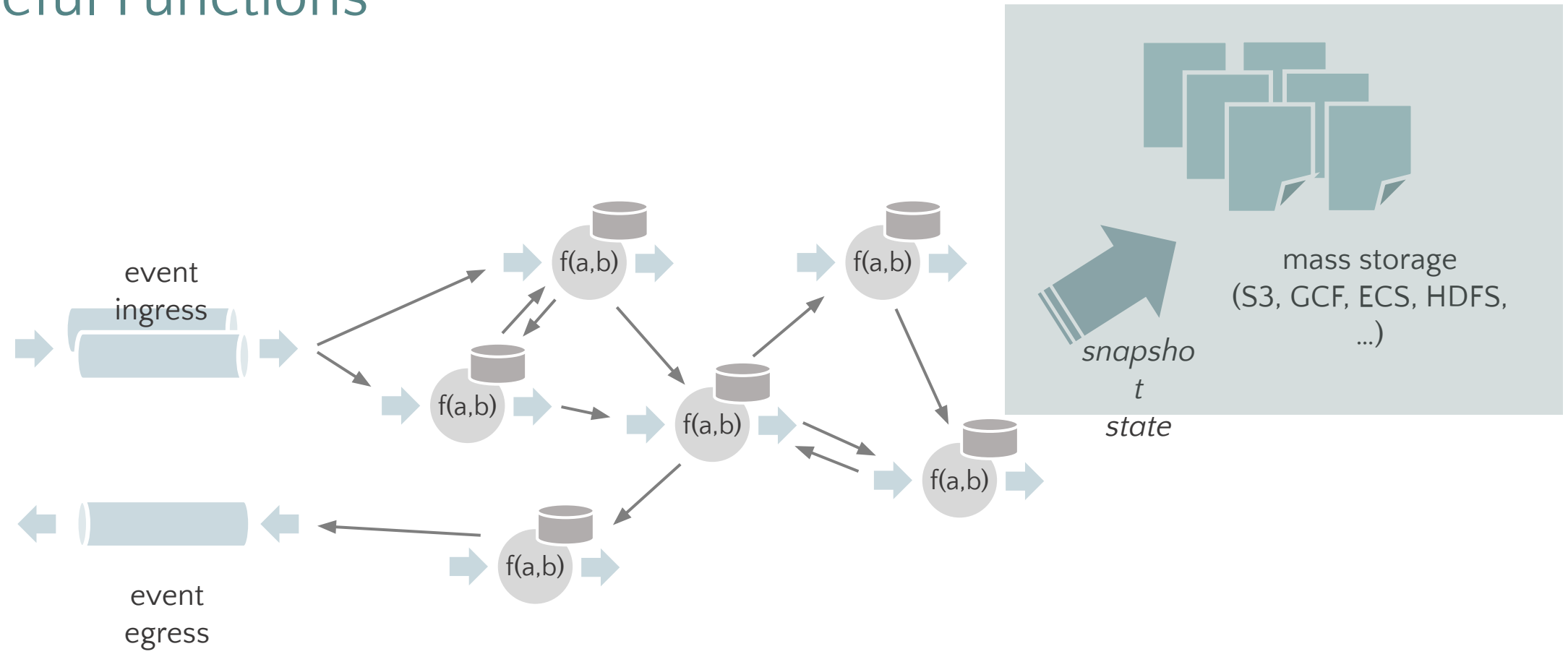
Functions have locally embedded state

Stateful Functions



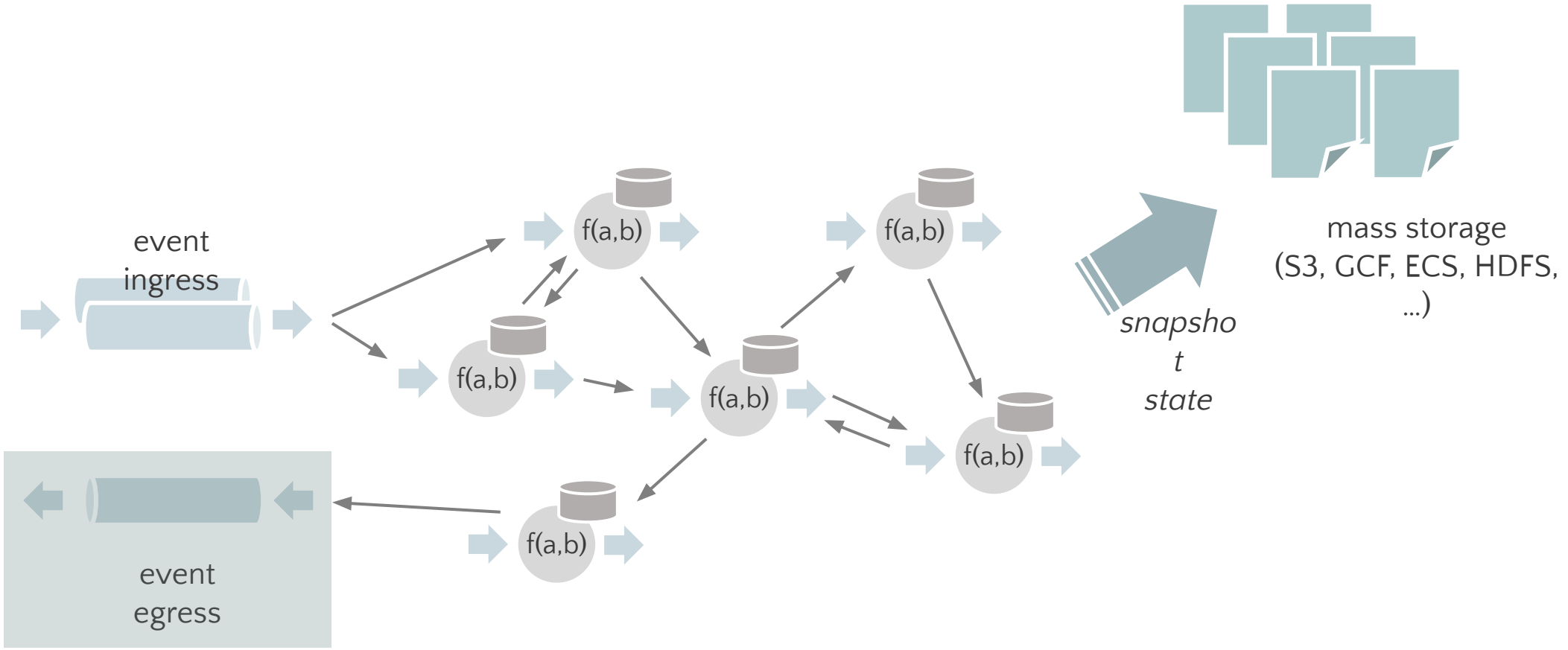
State and messaging are consistent
with exactly-once semantics

Stateful Functions



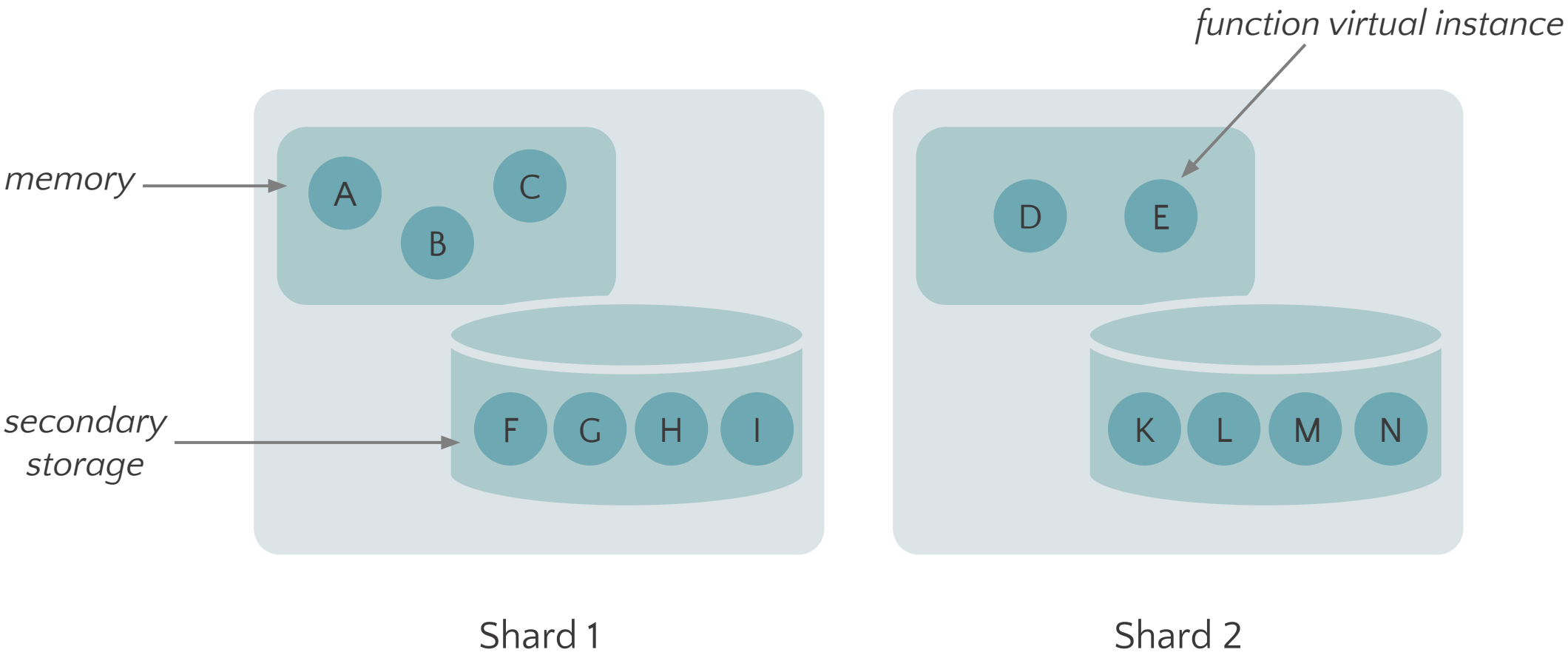
No database required
All persistence goes directly to blob storage

Stateful Functions

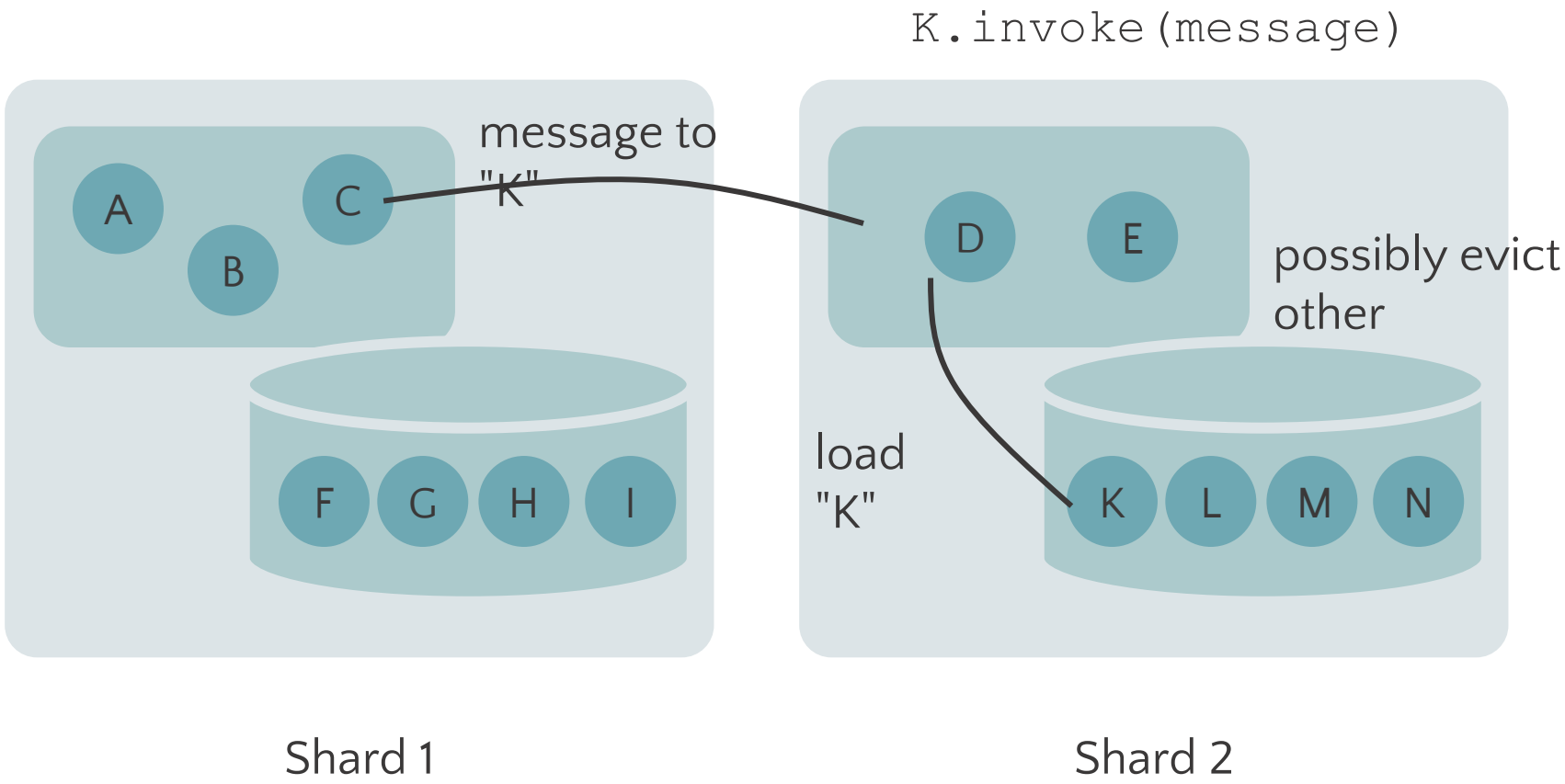


Event egresses to respond via event streams

Logical/Virtual Instances



Logical/Virtual Instances

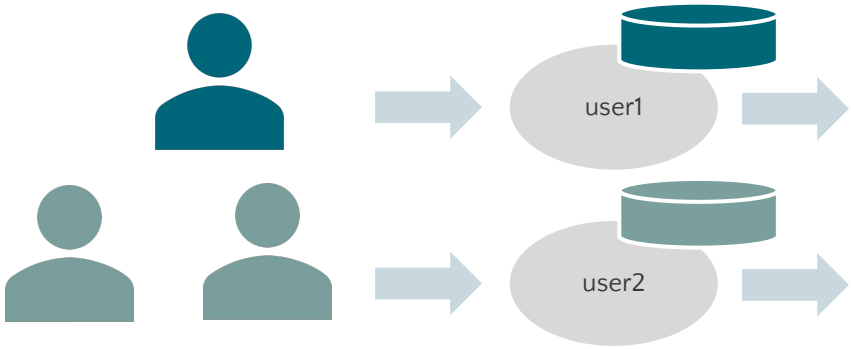


SDK Concepts

- Each function is associated with a **FunctionType** and **id**
  - **FunctionType** + **id** ⇒ **Address**

- An **Address** is logical not physical
  - No service discovery required

FunctionType ⇒ Greeter
id ⇒ User Id

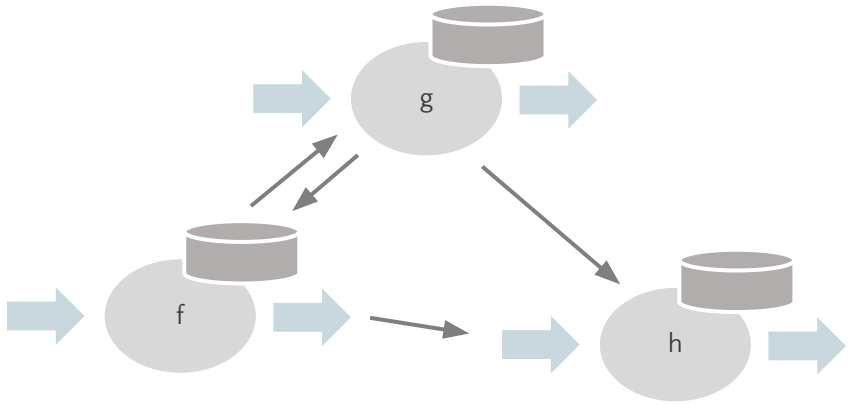


SDK Concepts

```
public class GreetRouter implements Router<GreetRequest> {

        @Override
        public void route(GreetRequest message, Downstream<GreetRequest> downstream) {
                Address address = new Address(GreetFunction.TYPE, message.getUserId());
                downstream.forward(address, message);
        }
}
```


SDK Concepts

- Applications are bundles of **StatefulFunction**'s
- A stateful function reacts to incoming events and can:
  - Perform a local computation
  - Access & modify local state
  - Send a message to any other stateful function
  - Send a message to external systems
  - Send a message with delay
  - Complete an asynchronous request



SDK Concepts

```
public class GreeterFunction implements StatefulFunction {

        public static FunctionType TYPE = new FunctionType("ververica", "greeter");

        @Persisted
        PersistedValue<Integer> seenCount = PersistedValue.of("seen", Integer.class);

        @Override
        public void invoke(Context ctx, Object message) {
                String userId = ctx.self().id();
                int seen = seenCount.getOrDefault(0) + 1;
                seenCount.set(seen);

                String greeting = createGreeting(userId, seen);
                ctx.send(Identifier.greetings, greeting)
        }
}
```


SDK Concepts

```
public class GreeterFunction implements StatefulFunction {

    public static FunctionType TYPE = new FunctionType("ververica", "greeter");

    @Persisted
    PersistedValue<Integer> seenCount = PersistedValue.of("seen", Integer.class);

    @Override
    public void invoke(Context ctx, Object message) {
        String userId = ctx.self().id();
        int seen = seenCount.getOrDefault(0) + 1;
        seenCount.set(seen);

        String greeting = createGreeting(userId, seen);
        ctx.send(Identifier.greetings, greeting)
    }
}
```


SDK Concepts

```
public class GreeterFunction implements StatefulFunction {

        public static FunctionType TYPE = new FunctionType("ververica", "greeter");

        @Persisted
        PersistedValue<Integer> seenCount = PersistedValue.of("seen", Integer.class);

        @Override
        public void invoke(Context ctx, Object message) {
                String userId = ctx.self().id();
                int seen = seenCount.getOrDefault(0) + 1;
                seenCount.set(seen);

                String greeting = createGreeting(userId, seen);
                ctx.send(Identifier.greetings, greeting)
        }
}
```


SDK Concepts

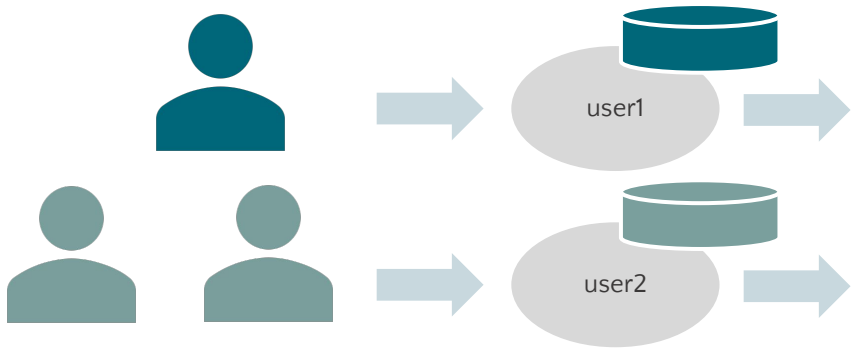
```
public class GreeterFunction implements StatefulFunction {

        public static FunctionType TYPE = new FunctionType("ververica", "greeter");

        @Persisted
        PersistedValue<Integer> seenCount = PersistedValue.of("seen", Integer.class);

        @Override
        public void invoke(Context ctx, Object message) {
                String userId = ctx.self().id();
                int seen = seenCount.getOrDefault(0) + 1;
                seenCount.set(seen);

                String greeting = createGreeting(userId, seen);
                ctx.send(Identifier.greetings, greeting)
        }
}
```

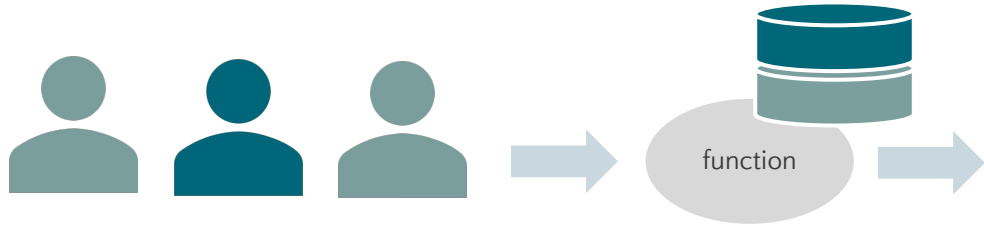

SDK Concepts

```
public class GreeterFunction implements StatefulFunction {

        public static FunctionType TYPE = new FunctionType("ververica", "greeter");

        @Persisted
        PersistedValue<Integer> seenCount = PersistedValue.of("seen", Integer.class);

        @Override
        public void invoke(Context ctx, Object message) {
                String userId = ctx.self().id();
                int seen = seenCount.getOrDefault(0) + 1;
                seenCount.set(seen);

                String greeting = createGreeting(userId, seen);
                ctx.send(Identifier.greetings, greeting)
        }
}
```

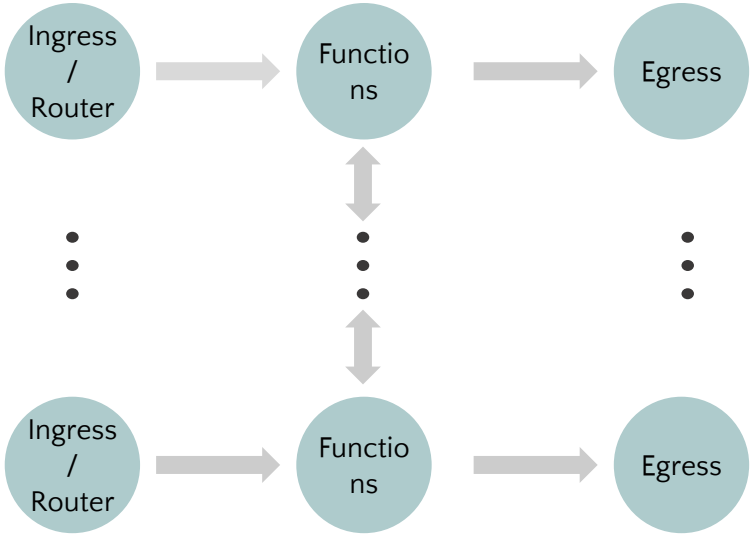

Execution Model

- There is a single (logical) instance of a function per **Address**
- Each **StatefulFunction** is created on demand, transparently by the runtime
- Messages sent to an **Address** are processed by a single thread
- Messages sent from function A to function By are always received in FIFO order

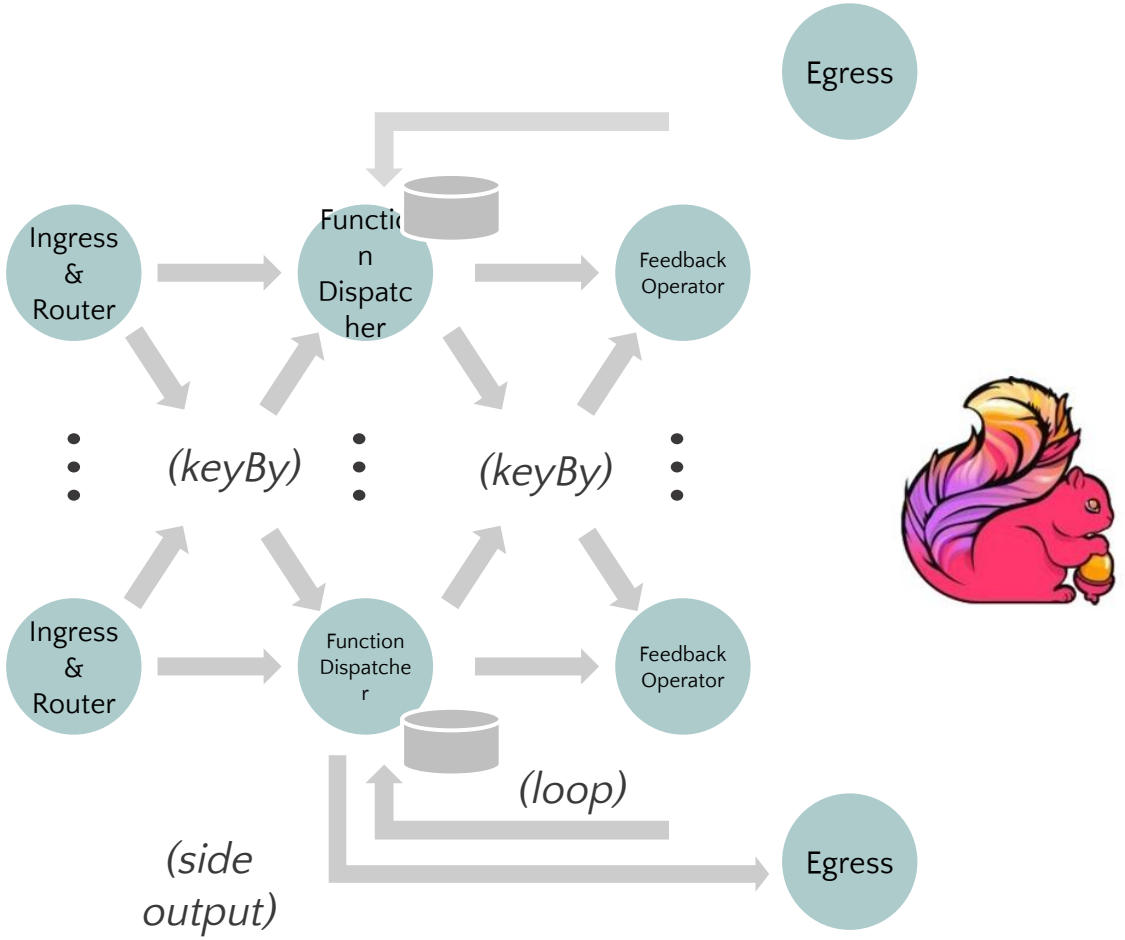




Apache Flink is the State and Event Streaming Fabric

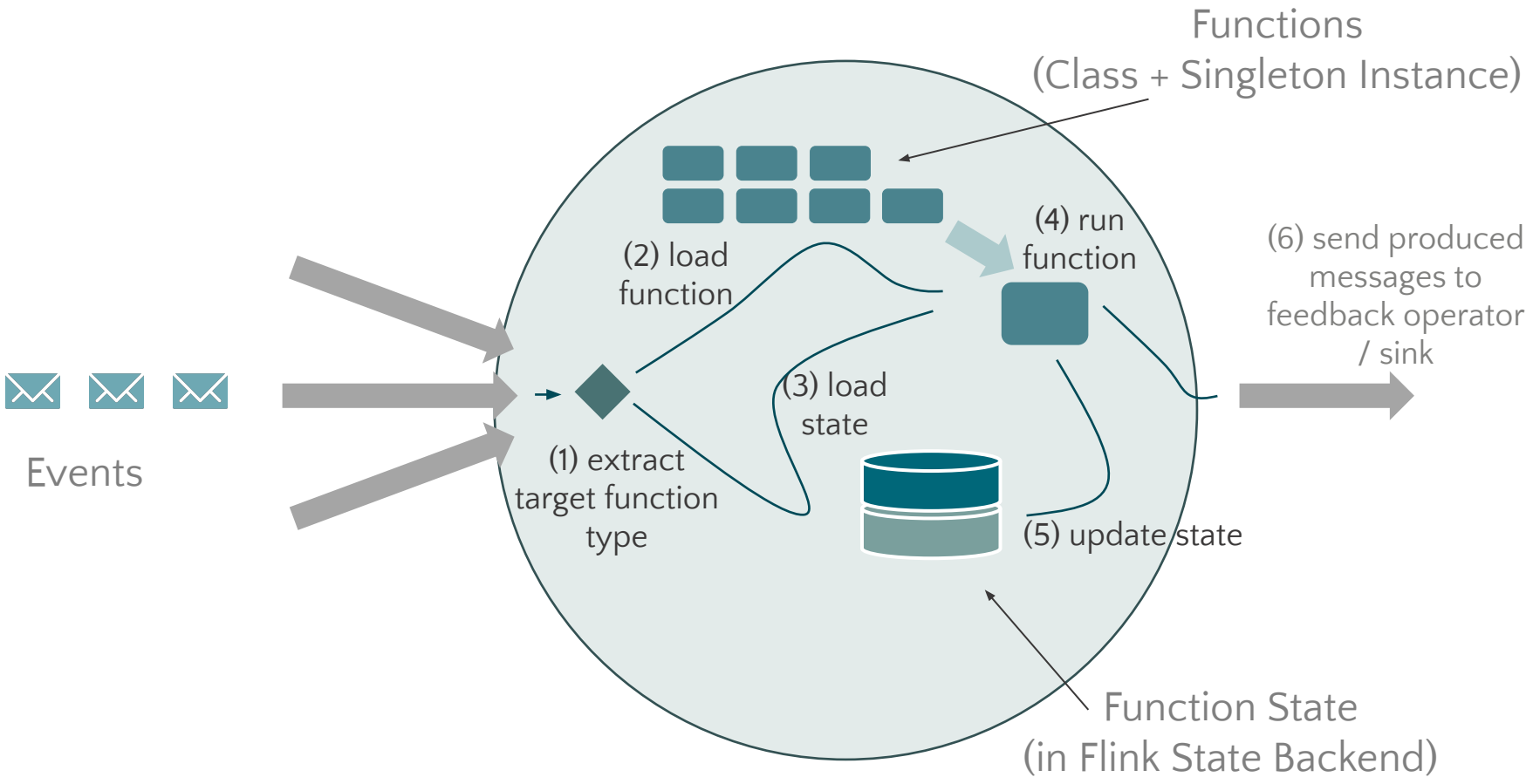


Conceptual Dataflow

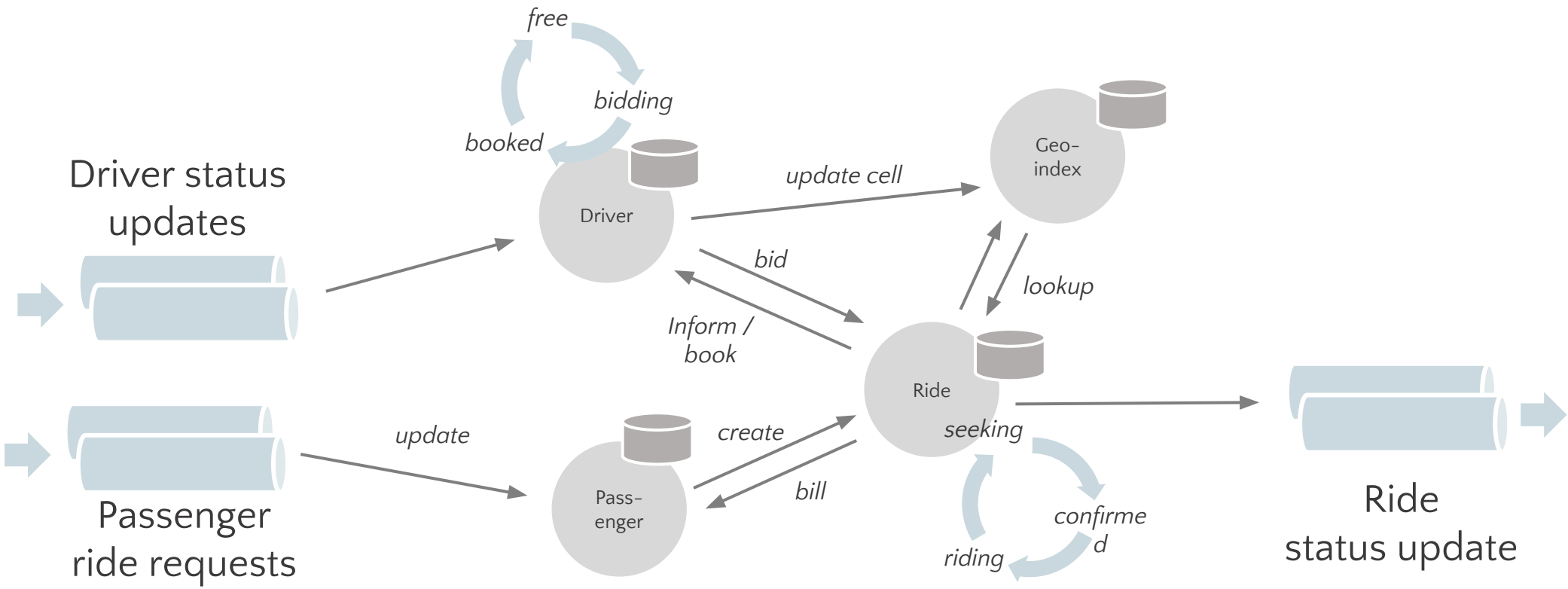


Apache Flink Dataflow Graph

A Peek Under The Hood



Example: Ride Sharing App



Stream Processing
Streaming SQL

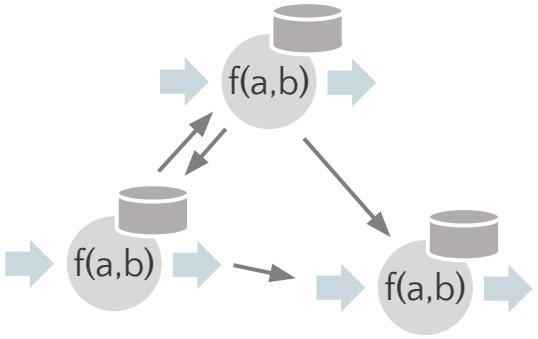
event/stream-centric

data preparation
combining knowledge/information

filtering, enriching,
aggregating, joining events

Stateful Functions

state-centric

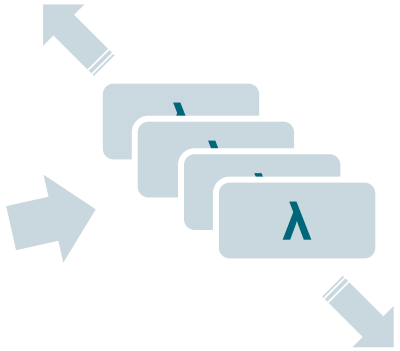


coordination,
(interacting) state machines

complex event/state
interactions

F-a-a-S

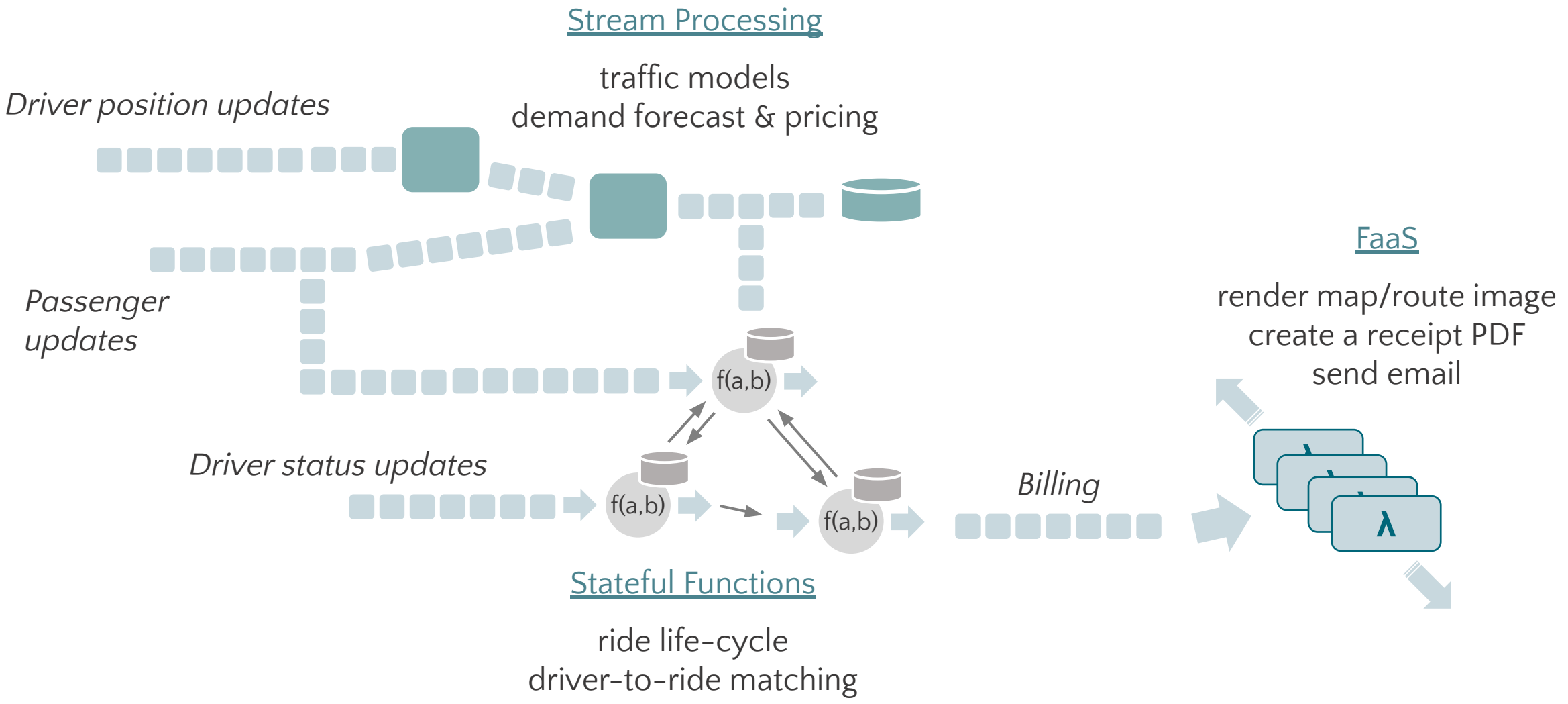
stateless / compute-centric



"occasional" actions or
spiky loads

compute-intensive
or blocking

Putting it all together: Ridesharing again



THANK YOU!

www.statefun.io
@statefun_io
github.com/ververica/stateful-functions