Hands-On Stateful Serverless with Apache Flink Stateful Functions

Seth Wiesman / **@sjwiesman**
Apache Flink Committer & Solutions Architect, Ververica

What does a Stream Processor have to say about Serverless?

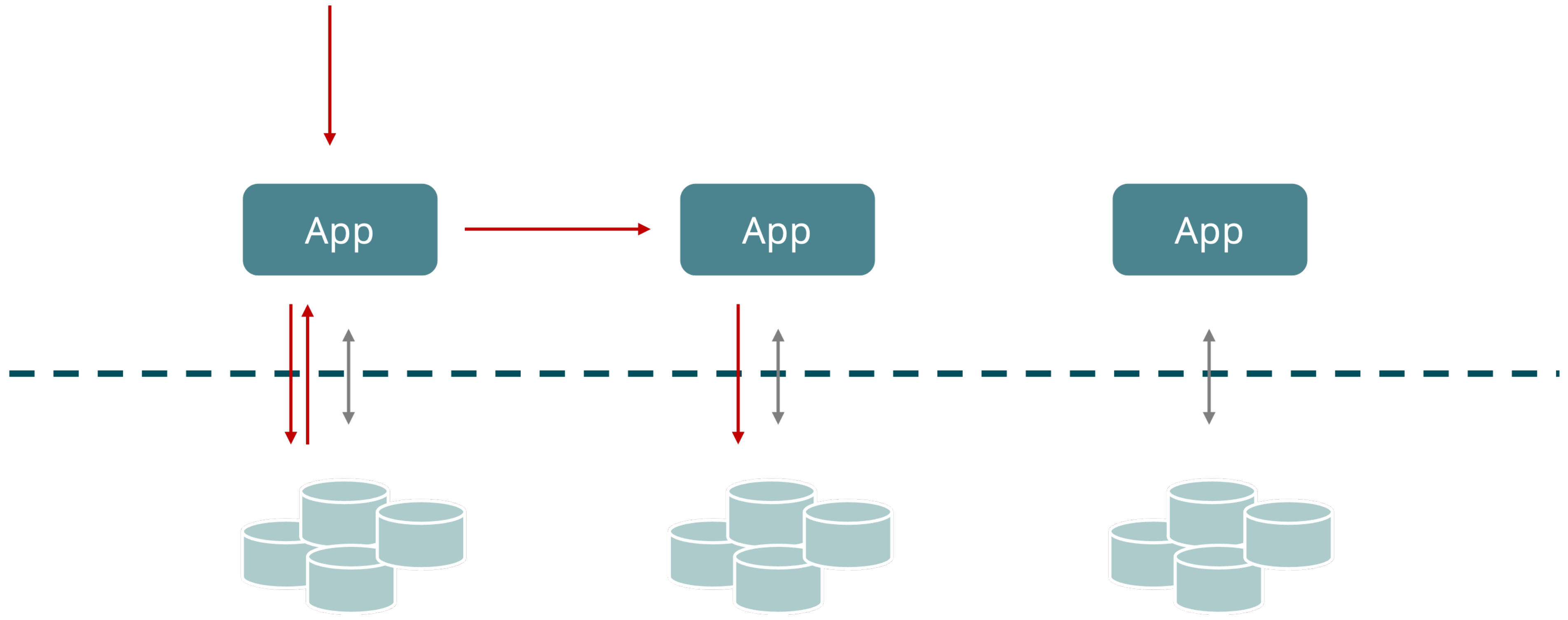
What is Stateful Serverless

- **Serverless** architectures are an application of modern infrastructure capabilities
  - Rapid Scalability
  - Scale to Zero
  - Zero Downtime Upgrades

- **Stateful Serverless** is about bringing these advances to the application layer **plus**
  - Consistent durable state
  - Cloud native fault tolerance
  - Simple messaging between systems
    - No service discovery
    - Strong ordering guarantees between messages

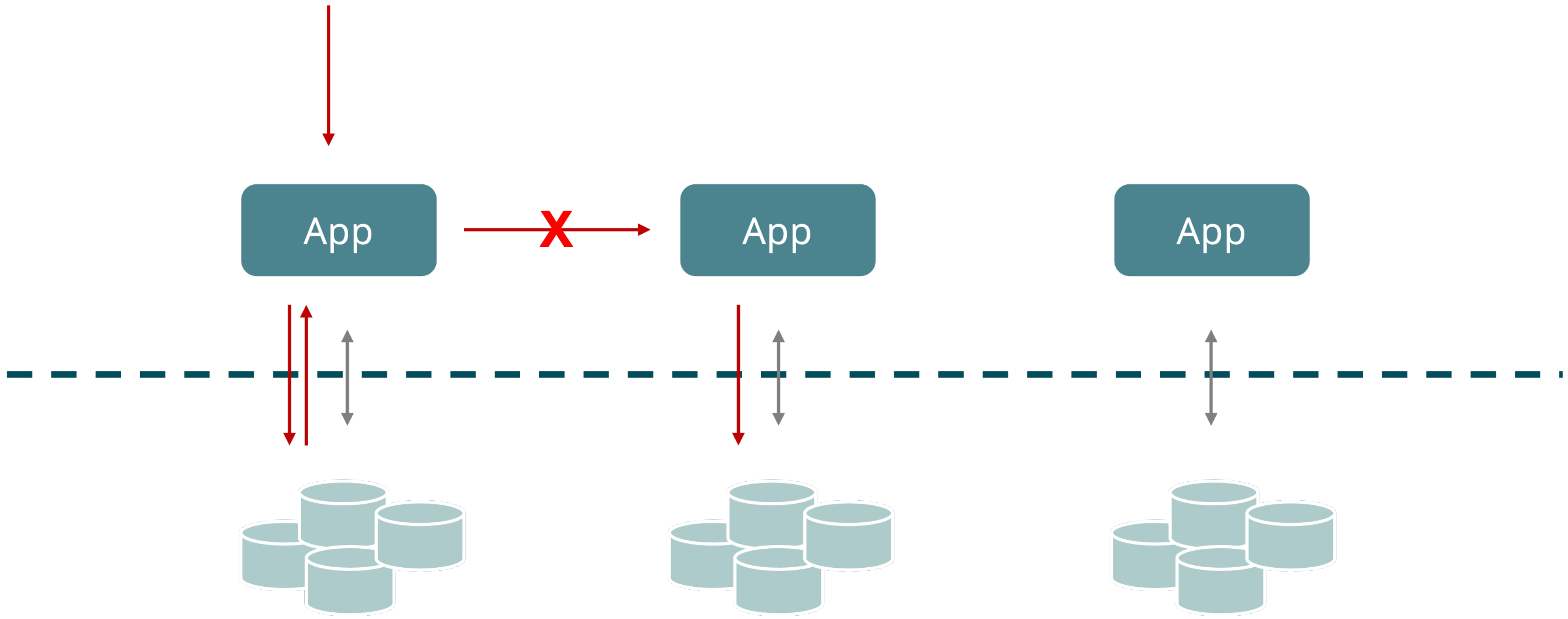
When the application handles messaging

and consistency . . .



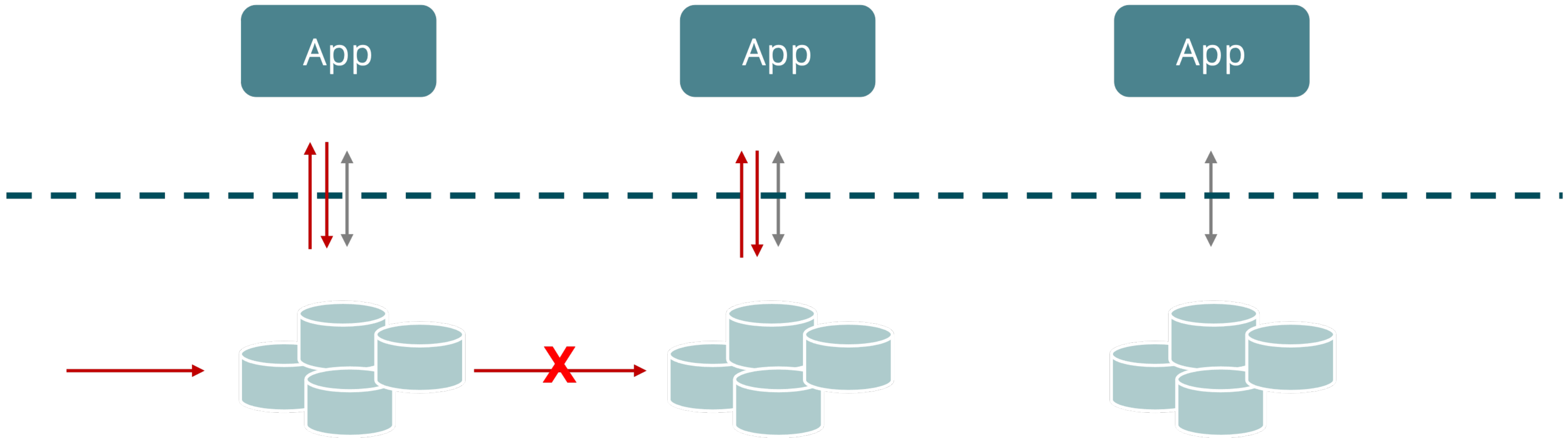
When the application handles messaging

and consistency, the application becomes complex



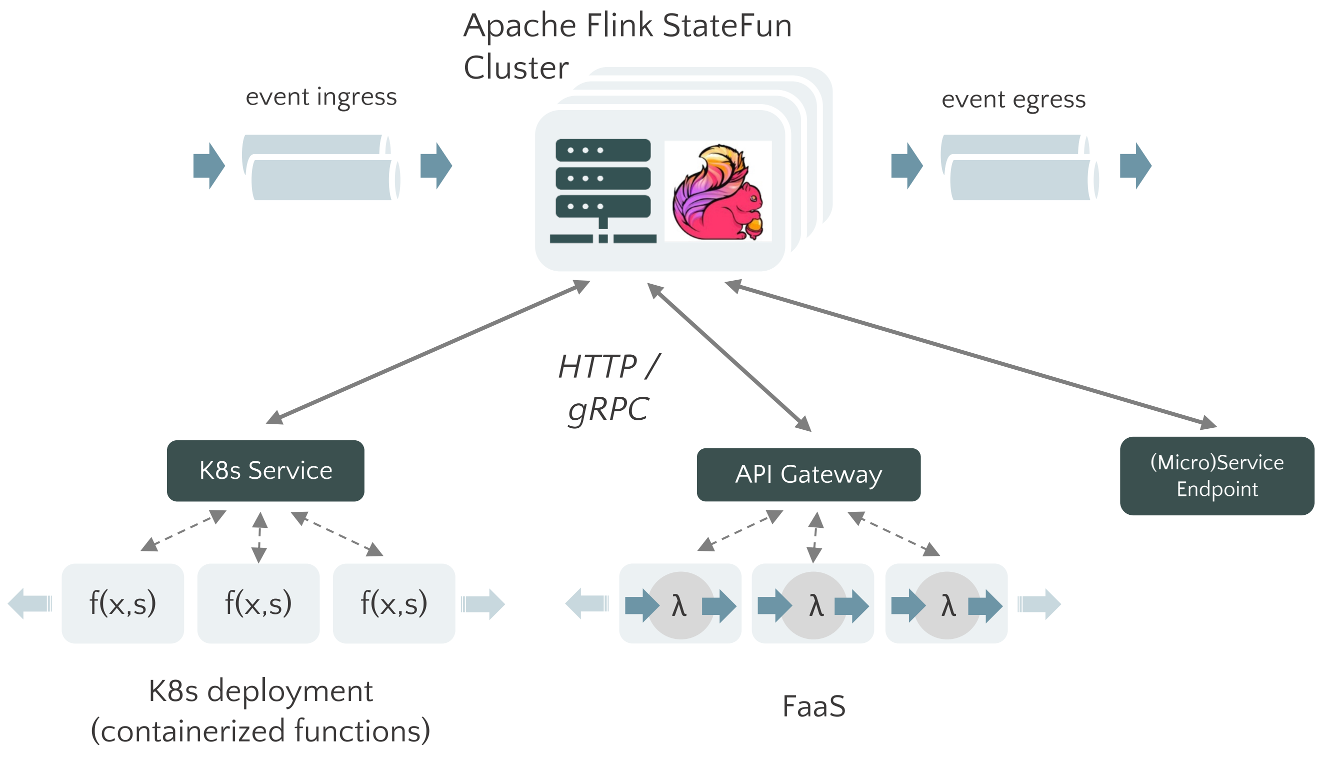
What if the state layer handles messaging

and consistency?



Apache Flink Stateful Functions

An API that simplifies building **distributed stateful applications** with a runtime build for **serverless architectures.**



Cloud Native

- Can be combined with capabilities of modern serverless platforms (e.g. AWS Lambda)

“Stateless” Operation

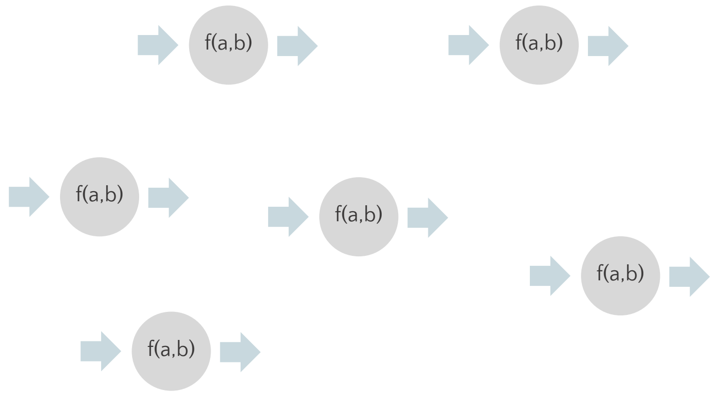
- State access / updates is part of the invocations / responses
- Function deployments have benefits of stateless processes – rapid scalability, scale-to-zero, zero-downtime upgrades

What is Stateful Serverless

- **Serverless** architectures are an application of modern infrastructure capabilities
  - Rapid Scalability ✓
  - Scale to Zero ✓
  - Zero Downtime Upgrades ✓

- **Stateful Serverless** is about bringing these advances to the application layer **plus**
  - Consistent durable state
  - Cloud native fault tolerance
  - Simple messaging between systems
    - No service discovery
    - Strong ordering guarantees between messages

What is Stateful Functions?



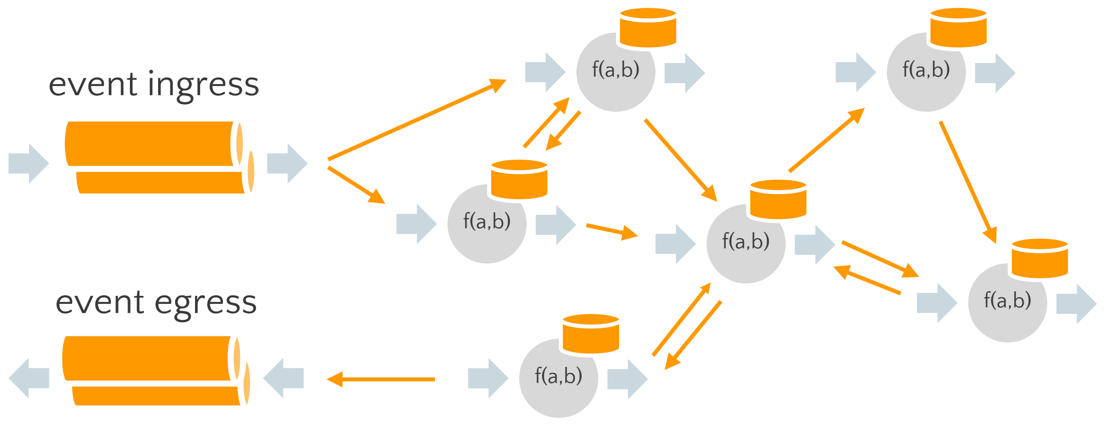
Building block: Functions

- Small piece of logic that represents entities
- Invokable through messages
- Inactive functions don't consume resources

Multi-language Support

- Can be implemented in any programming language that handles HTTP requests or gRPC

What is Stateful Functions?



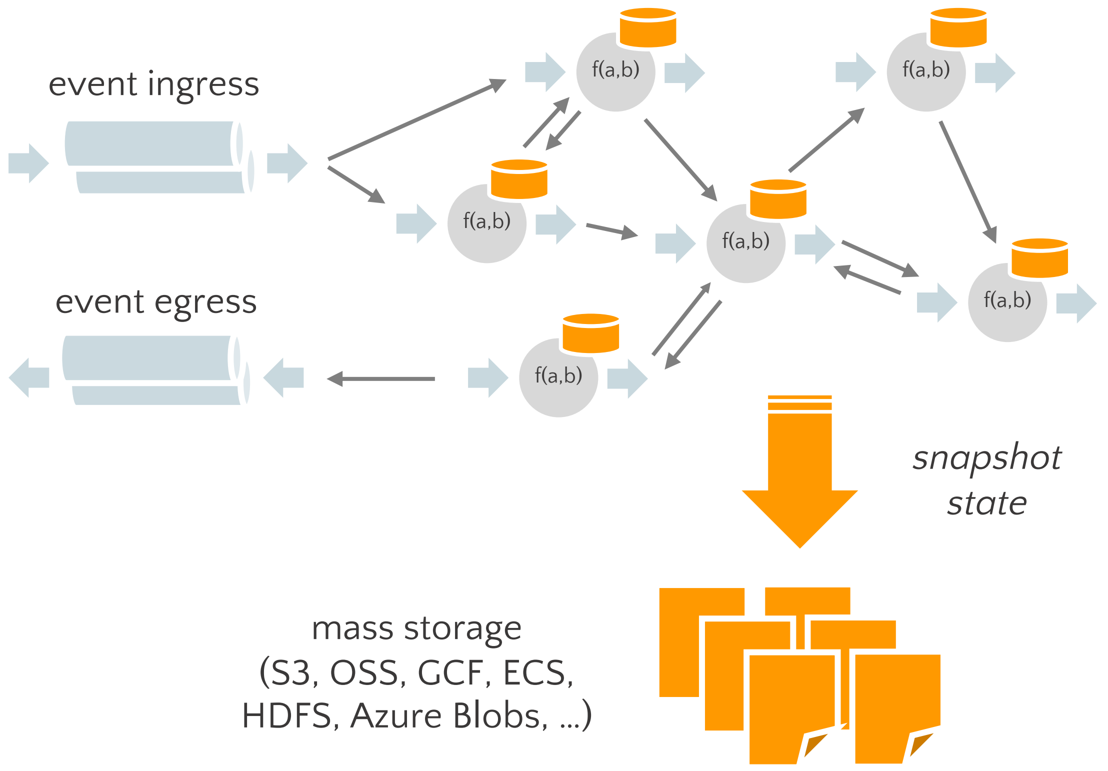
Dynamic messaging

- Arbitrary communication between functions
- Functions message each other by logical addresses – no service discovery needed

Consistent state

- Functions keep local state that is persistent and integrated with messaging
- Out-of-box exactly-once state access / updates & messaging

What is Stateful Functions?



No Database Required

- Uses Flink’s distributed snapshots model for state durability and fault tolerance
- Requires only a simple blob storage tier to store state snapshots

Application Development Walkthrough

Types in Remote Functions

- Since remote functions may be implemented with any language that handles HTTP requests, functions implemented with different languages may message each other arbitrarily

- All messages sent to / from remote functions are required to **uniformly be the Protobuf Any type**

    - **Any** messages wrap a Protobuf message in its serialized form, plus an URL describing the type
    - Receiving functions may "unpack" the **Any** messages they receive to specific types using their language-specific Protobuf libraries

Types in Remote Functions

- The same goes for state types – they must be the Protobuf **Any** type
    - This allows state written by arbitrary languages to be uniformly maintained in Flink
    - Flink simply stores state in backends in their serialized form, as wrapped in the **Any**
- The Python SDK (for remote functions) provides utility methods to:
    - Pack / Unpack messages and state objects
    - Allows users to develop only against specific Protobuf types

Python SDK: Stateful Functions

with Python (remote function):

```
def greet(context, input: GreetRequest):
   name = context.address.identity

   greeting = create_personalized_greeting(
      name,
      context,
   )

   context.pack_and_send(
       "demo/email_sender",
       input.reply_email,
       greeting
   )
```

Python SDK: Stateful Functions

with Python (remote function):

```
def greet(context, input: GreetRequest):
   name = context.address.identity

   greeting = create_personalized_greeting(
      name,
      context,
   )

   context.pack_and_send(
       "demo/email_sender",
       input.reply_email,
       greeting
   )
```

- Each function instance is associated with a **function type** + **ID**, together forming the instance's unique **Address**

Python SDK: Stateful Functions

with Python (remote function):

```
def greet(context, input: GreetRequest):
    name = context.address.identity

    greeting = create_personalized_greeting(
        name,
        context,
    )

    context.pack_and_send(
        "demo/email_sender",
        input.reply_email,
        greeting
    )
```

- Automatically unpacks input messages as specified Protobuf message type

Python SDK: Stateful Functions

with Python (remote function):

```
def greet(context, input: GreetRequest):
   name = context.address.identity

   greeting = create_personalized_greeting(
      name,
      context,
   )

   context.pack_and_send(
       "demo/email_sender",
       input.reply_email,
       greeting
   )
```

- Automatically unpacks input messages as specified Protobuf message type
- `pack_and_send` packs output Protobuf messages as `Any` before sending

Python SDK: Stateful Functions

with Python (remote function):

```
def greet(context, input: GreetRequest):
    name = context.address.identity

    greeting = create_personalized_greeting(
        name,
        context,
    )

    context.pack_and_send(
        "demo/email_sender",
        input.reply_email,
        greeting
    )
```

- Automatically unpacks input messages as specified Protobuf message type
- `pack_and_send` packs output Protobuf messages as `Any` before sending
- To invoke a function, simply send a message to its address
- In the Python SDK, function types are defined with strings, with the format:

  `"<namespace>/<name>"`

Python SDK: Persisted State

with Python (remote function):

```
def create_personalized_greeting(name, context):
   seen = context.state("seen-count").unpack(SeenCount)
   if not seen:
       seen = SeenCount()
       seen.count = 1
   else:
       seen.count += 1
   context.state("seen-count").pack(seen)

   text = greetText(name, seen.count)
   greeting = PersonalizedGreeting()
   greeting.text = text
   return greeting
```

Python SDK: Persisted State

with Python (remote function):

```
def create_personalized_greeting(name, context):
   seen = context.state("seen-count").unpack(SeenCount)
   if not seen:
       seen = SeenCount()
       seen.count = 1
   else:
       seen.count += 1
   context.state("seen-count").pack(seen)

   text = greetText(name, seen.count)
   greeting = PersonalizedGreeting()
   greeting.text = text
   return greeting
```

- State is accessed / updated using the invocation context
- Use **unpack / pack** to work against specific Protobuf types

Python SDK: Exposing Functions

```
functions = StatefulFunctions()

@functions.bind("demo/greet")
def greet(context, message: GreetRequest): // …

@functions.bind("demo/email_sender")
def sendEmail(context, message: PersonalizedGreeting): // ...

handler = RequestReplyHandler(functions)

app = Flask(__name__)

@app.route('/statefun', methods=['POST'])
def handle():
   response_data = handler(request.data)
   response = make_response(response_data)
   return response

if __name__ == "__main__":
   app.run()
```

Python SDK: Exposing Functions

```
functions = StatefulFunctions()

@functions.bind("demo/greet")
def greet(context, message: GreetRequest): // …

@functions.bind("demo/email_sender")
def sendEmail(context, message: PersonalizedGreeting): // ...

handler = RequestReplyHandler(functions)

app = Flask(__name__)

@app.route('/statefun', methods=['POST'])
def handle():
   response_data = handler(request.data)
   response = make_response(response_data)
   return response

if __name__ == "__main__":
   app.run()
```

- Bind multiple function types with their function definition

Python SDK: Exposing Functions

```
functions = StatefulFunctions()

@functions.bind("demo/greet")
def greet(context, message: GreetRequest): // …

@functions.bind("demo/email_sender")
def sendEmail(context, message: PersonalizedGreeting): // ...

handler = RequestReplyHandler(functions)

app = Flask(__name__)

@app.route('/statefun', methods=['POST'])
def handle():
   response_data = handler(request.data)
   response = make_response(response_data)
   return response

if __name__ == "__main__":
   app.run()
```

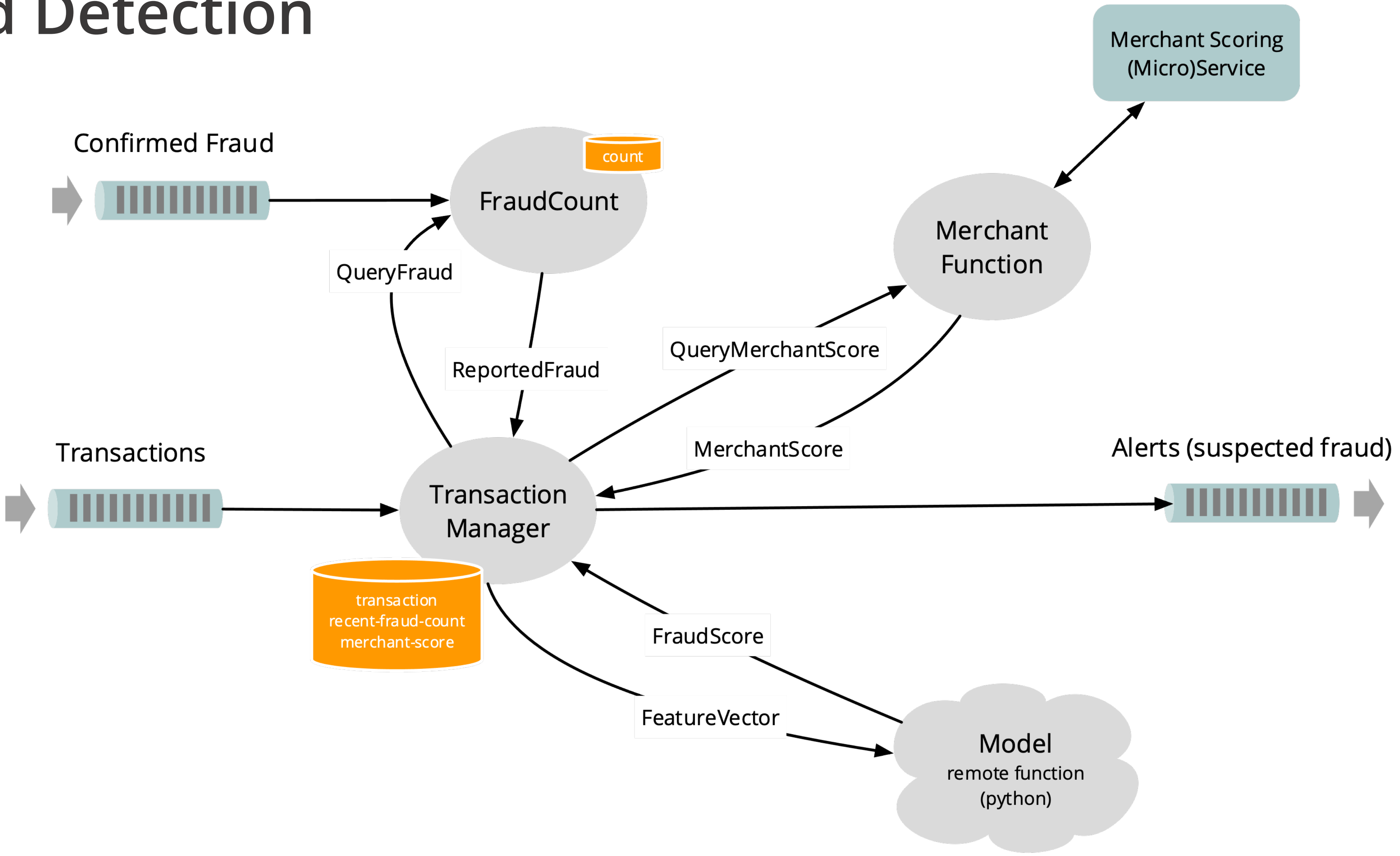
- Bind multiple function types with their function definition
- SDK ships a `RequestReplyHandler` which:
  - dispatches invocation requests via HTTP to bound functions
  - Ecodes their side effects (resulting outgoing messages and state updates) as an HTTP response

Python SDK: Exposing Functions

```
functions = StatefulFunctions()

@functions.bind("demo/greet")
def greet(context, message: GreetRequest): // …

@functions.bind("demo/email_sender")
def sendEmail(context, message: PersonalizedGreeting): // ...

handler = RequestReplyHandler(functions)

app = Flask(__name__)

@app.route('/statefun', methods=['POST'])
def handle():
   response_data = handler(request.data)
   response = make_response(response_data)
   return response

if __name__ == "__main__":
   app.run()
```

- Bind multiple function types with their function definition
- SDK ships a `RequestReplyHandler` which:
  - dispatches invocation requests via HTTP to bound functions
  - Ecodes their side effects (resulting outgoing messages and state updates) as an HTTP response
- Expose the handler with your favorite HTTP web framework

Fraud Detection



Thank You

@statefun_io

@sjwiesman