# The Serverless Trilemma

## (Function Composition for Serverless Computing)

TCSS562: Group 3
Anisha Agarwal
Chhaya Choudhary
Sanchya Bhagat

UNIVERSITY *of* WASHINGTON

## Talk Outline

The key points of the talk:

- > **Serverless composition-as-function problem**
- > **The Core (Reactive) Model - Using Apache OpenWhisk**
- > **Problem: The serverless trilemma**
- > **Solution: Trilemma-Safe Sequential Composition**
- > **Critique**

## Paper Overview

**The problem?**

Composition-as-functions must violate at least one of the 3 constraints:

- **Functions should be considered as black boxes;**
- **Function composition should obey a substitution principle with respect to synchronous invocation.**
- **Invocations should not be double-billed.**

**Why it is problem?**

Economics, performance, and synchronous composition.

## Paper Overview

- **Composition via Reflection:**
  - f1 followed by f2
  - running time of f1 will be billed twice: once as f1 and once as part of f2
- **Composition via Fusion**
  - f3 is a function that inlines the code of the sequence members
  - violate a black box constraint; e.g. they assume availability of source code, and that functions are monoglot (written in the same language)
- **Composition with Asyncs**
  - fire-and-forget model of composition
  - violate a substitution principle: f3 is no longer a composable serverless function
- **Composition on the Client**
  - follows a client-scheduled structure these compositions cannot be nested inside of other compositions that are unaware of that client

## Introduction

**Trilemma-Safe Sequential Composition**

Serverless core must offer more than actions, rules, and triggers to satisfy all the three constraints

> **Overview of the OpenWhisk Invocation Flow**
- Handling of invocations
- Consists of 4 components: Controller, Invokers, Message Queue and System of Record.

> **Realizing Completion Triggers with "active ack"**
- Microarchitectural strategy of pipeline bypass known as active ack
- Notion of completion triggers
- Used to reduce the latency of request-response invocations, orchestrate and optimize invocations.
- Reduction of overhead by blocking calls by 18X.

## Introduction

> **ST-Safe Sequences with active ack**
- Active ack strategy to schedule sequences
- **Includes 2 changes:**
  - Specifying the action to be of type Sequence and component OpenWhisk actions to form the composition.
  - The controller must handle the invocation of a Sequence action differently.
- User does not get double billed
- Very less system overhead by avoiding the use of heavy weight resources for action invocation.

## Key Contributions

- **A formulation of the serverless trilemma**
- **A programming model to build new serverless functions**
- **A solution to the trilemma for the sequential composition of functions**
- **The implementation in Apache OpenWhisk, an open source serverless runtime**
- **Improvement in Latency reduction**
  - New latency for result passing from the invoker to the controller: 1-2ms on average.
  - Old latency for storing and then fetching a document with the system of record: 26ms and 10ms on average.

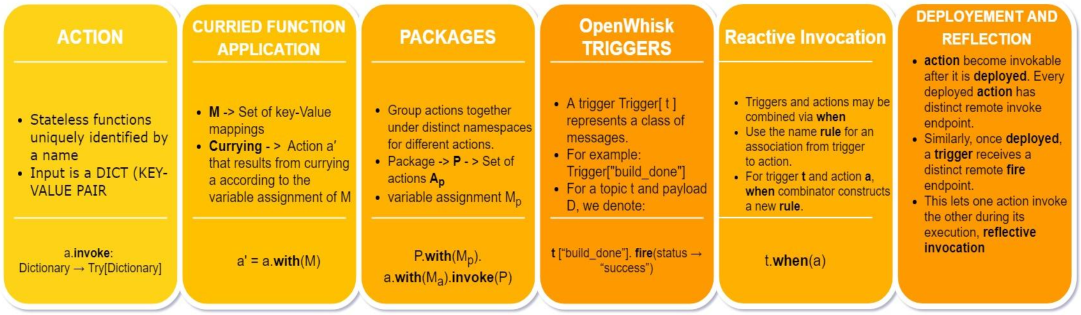
## Background/Related Work

**Serverless Computing:**

- **Functions as a Service**
  - Micro-services are offered as separated “actions” or “functions”.
  - One function generates an output ( example JSON) that acts as input to any other function.
- **Event-driven invocations**
  - The function should invoked based on events.
    For example: When a function build completes, it “triggers” the other function(s).
- **Function composition**
  - Rather than create a single monolithic function, it is often desirable to separate the concerns of schema alignment and notification.

# Background -> OpenWhisk

**Overview of Key terms:**



**ACTION**

- Stateless functions uniquely identified by a name
- Input is a DICT (KEY-VALUE PAIR

a.**invoke**:
Dictionary → Try[Dictionary]

**CURRIED FUNCTION APPLICATION**

- **M** -> Set of key-Value mappings
- **Currying** - > Action a' that results from currying a according to the variable assignment of M

a' = a.**with**(M)

**PACKAGES**

- Group actions together under distinct namespaces for different actions.
- Package **->** **P** - > Set of actions $\mathbf{A_p}$
- variable assignment $M_p$

P.**with**($M_p$).
a.**with**($M_a$).**invoke**(P)

**OpenWhisk TRIGGERS**

- A trigger Trigger[ t ] represents a class of messages.
- For example: Trigger["build_done"]
- For a topic t and payload D, we denote:

**t** ["build_done"]. **fire**(status → "success")

**Reactive Invocation**

- Triggers and actions may be combined via **when**
- Use the name **rule** for an association from trigger to action.
- For trigger **t** and action **a**, **when** combinator constructs a new **rule**.

t.**when**(a)

**DEPLOYEMENT AND REFLECTION**

- **action** become invokable after it is **deployed**. Every deployed **action** has distinct remote invoke endpoint.
- Similarly, once **deployed**, a **trigger** receives a distinct remote **fire** endpoint.
- This lets one action invoke the other during its execution, **reflective invocation**

Source: http://openwhisk.incubator.apache.org/

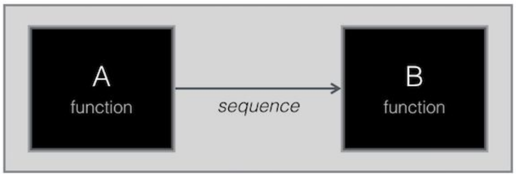
# Related Work

> *OpenWhisk relies heavily on prior work for lightweight isolated execution environments. The current implementation exploits technologies developed for Linux Containers [3].*

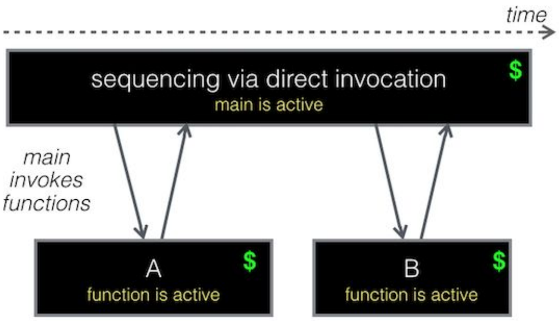
> *AWS Step Functions [4] is an example of composing functions as steps, and describing a state machine for the overall orchestration of a large application.*

## The Serverless Trilemma

> This desired sequential combinator as then i.e. a.then(b).then(c) => c (b (a()))



> **Composition by Reflective Action Invocation** (Double Billing Constraint)



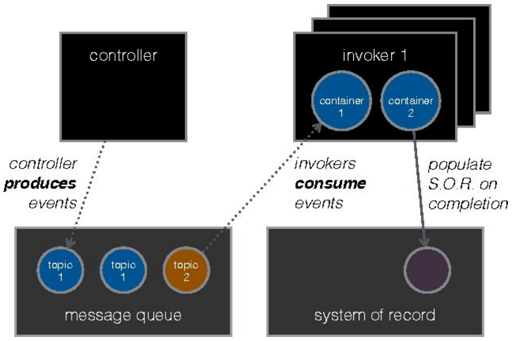
## The Serverless Trilemma

> **Composition by Fusion of Actions** (Black-Box/Polyglot Constraint)
- To avoid the double billing, we can infuse all functions in one source code.
- Challenges: The source to every action is available, and in the same language

> **Interlude: The Serverless Substitution Principle**
- Compositions-as-actions conform to the the JSON in, JSON out protocol of actions
- Implies a single entry-single exit structure
- Replace it with async/await pattern

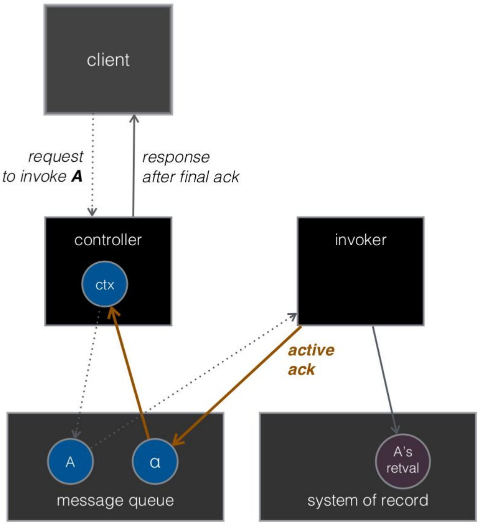
> **Client-Side Scheduling (Abnegation)**
- since it runs on the Client side scheduler not implemented as an action
- There is no black box, no double billing
- Satisfies substitution
- The approach doesn’t work always

# Trilemma-safe Sequential Composition



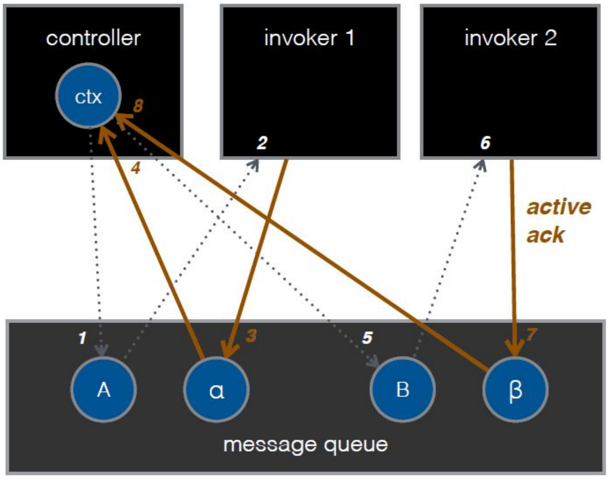
1. Overview

# Trilemma-safe Sequential Composition



2. Active-ack scheme

## Trilemma-safe Sequential Composition



3. ST-Safe Sequences with active ack

## Author’s Conclusions

- **Event - driven core of serverless**
  - Not yet expressive enough to implement compositions of functions, as serverless functions.
- **Continuation-passing style of invocation**
  - Cannot be expressed against the purely reactive core programming model that serverless platforms currently offer
- **Extension of core to implement sequential composition of functions.**
  - Available in open-source project Apache OpenWhisk.

## Critique: Strengths

- **Primary strengths of the new approach**
  - ST-Safe sequence composition
  - Optimization strategies to reduce the impact of cold start
  - Reduces Overhead
  - Better performance
  - Cost effective
  - Scalable
  - Secure

- **Strengths of the evaluation**
  - Use of three constraints: black boxes, substitution principle and double-billing

## Critique: Weaknesses

- **No reference to the "state of the art"**
- **Explanation missing for disregarding Composition on the Client as serverless**
- **Function composition -**
  - Is it a standard or a hypothesis for the sake of this paper?
  - Are there any other function composition(s) which could have been explored?
- **Comparison of performance and cost with other function-as-a-services would have been helpful**

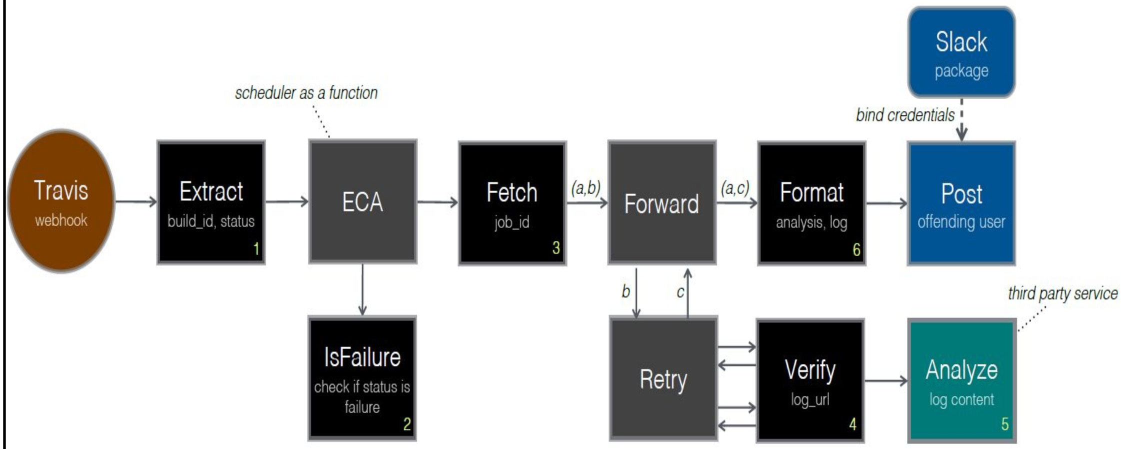
# Critique: Evaluation

- **Paper’s evaluation is satisfactory.**
- **Proof for serverless trilemma is missing.**
- **Less information on performance and cost metrics used.**
- **Results are hard to believe without proof and numbers.**
- **Enough information is not available to repeat /reproduce tests.**

# Future Work

- **Provide proofs of the serverless trilemma**
- **To extend the core to handle a larger class of compositions.**
- **To describe the classes of expressivity in serverless.**
- **Expansion of sequences for composition patterns:**
  - Addition of three combinators: Event-Condition-Action (ECA), retry, and data forwarding.
  - ECA: Static Composition versus Combinator
  - Retry as Metaprogram
  - Forward as Metaprogram

## Looking to the future: New Combinators



Case Study: the full Travis-to-Slack application includes three new composition patterns

## References

1. Baldini, Ioana, et al. "The serverless trilemma: function composition for serverless computing." Proceedings of the 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software. ACM, 2017.
2. https://medium.com/openwhisk/composing-functions-into-applications-70d3200d0fac
3. Dirk Merkel. 2014. Docker: Lightweight Linux Containers for Consistent Development and Deployment. Linux J. 2014, 239, Article 2 (March2014).
4. Amazon. 2016. AWS Step Functions. (2016). https://aws.amazon.com/step-functions/

## Questions

Questions ?