Consistency and Correctness in Workflow Systems

Michael Stonebraker, Xinjing Zhou, Peter Kraft, **Qian Li**

DBOS Research Project (2020 - 2023)

Apiary: A DBMS-Integrated Transactional Function-as-a-Service Framework

Peter Kraft[1*], Qian Li[1*], Kostis Kaffes[1], Athinagoras Skiadopoulos[1], Deeptaanshu Kumar[3], Danny Cho[1], Jason Li[2], Robert Redmond[2], Nathan Weckwerth[2], Brian Xia[2], Peter Bailis[1], Michael Cafarella[2], Goetz Graefe[4], Jeremy Kepner[2], Christos Kozyrakis[1], Michael Stonebraker[2], Lalith Suresh[5], Xiangyao Yu[6], and Matei Zaharia[1]
[1]Stanford, [2]MIT, [3]CMU, [4]Google, [5]VMware, [6]University of Wisconsin-Madison

Abstract

Developers increasingly use function-as-a-service (FaaS) platforms for *data-centric* applications that perform low-latency and transactional operations on data, such as for microservices or web serving. Unfortunately, existing FaaS platforms support these applications poorly because they physically and logically separate application logic, executed in cloud functions, from data management, done in interactive transactions accessing remote storage. Physical separation harms performance while logical separation complicates efficiently providing transactional guarantees and fault tolerance.

This paper introduces Apiary, a novel DBMS-integrated FaaS platform for deploying and composing fault-tolerant transactional functions. Apiary physically co-locates and logically integrates function execution and data management by wrapping a distributed DBMS engine and using it as a unified runtime for function execution, data management, and operational logging, thus providing similar or stronger transactional guarantees as comparable systems while greatly improving

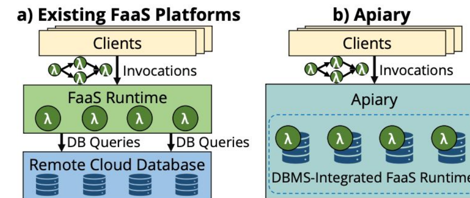


Figure 1: Existing FaaS platforms separate application logic, executed in cloud functions, from data management, done in interactive transactions accessing a remote database. Apiary instead tightly integrates application logic and data management, executing functions in DBMS stored procedures.

platforms support these applications poorly because they both physically and logically separate function execution from data management, adopting the architecture of Figure 1a and calling a remote DBMS once per data operation. Physical separation causes high communication overhead: in our experiments with OpenWhisk (Figure 2), communication accounts for as much as 98% of function runtime. Logical separation com-

doi:10.14778/3485450.3485454

Qian Li

Abstract

We propose T
bugging mod
Our critical in
and only acc
always-on tra
and data prov
transaction lo
retroactively
plify program
several resear
base and syst
between the

guts of
in data

rakis[1],

ant web and
erstand bugs
challenging
for applica-
utions. How-
atabases and
ed history of
at always-on
ces to locate
apply logged
nce TROD can
senbugs [11],
ause of com-
count of interactions
t stakeholders with
ntroduce limitations

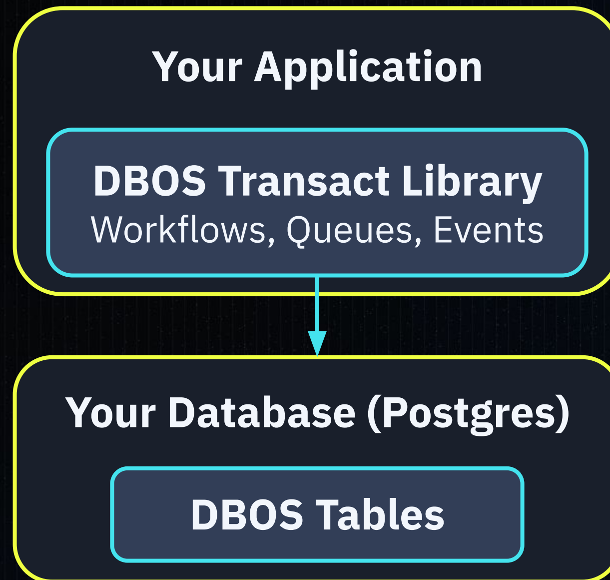
(ML), Internet of Things (IoT), and

(6) New programming models Users ideally want to pay for re-

Databases-Backed Workflows!

DBOS.DEV (2023 - now)

- Durable workflows and queues in a database-backed library

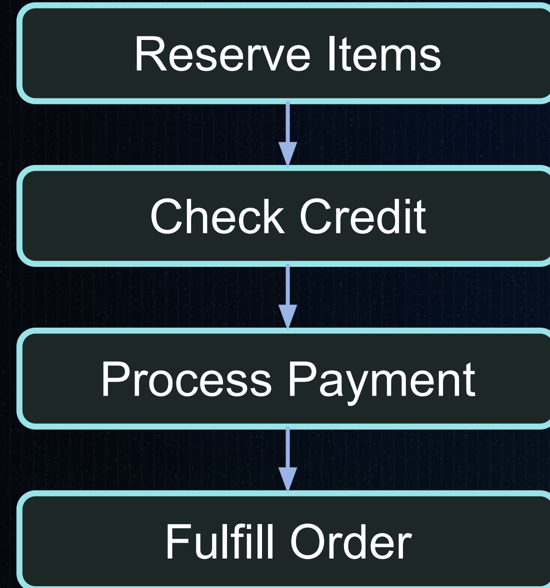


The Workflow Problem

- What guarantees existing systems provide
- What future systems should provide

Example: An Online Checkout Workflow

- Each step may involve **external systems**
- Contains **both read and write**



Reliability is Hard: Any Step Can Break!

- Process crashes (e.g., out-of-memory, bugs)
- User takes too long to respond/timeouts
- Unreliable external APIs
- Re-deploys

AI makes everything harder

Try try try again?

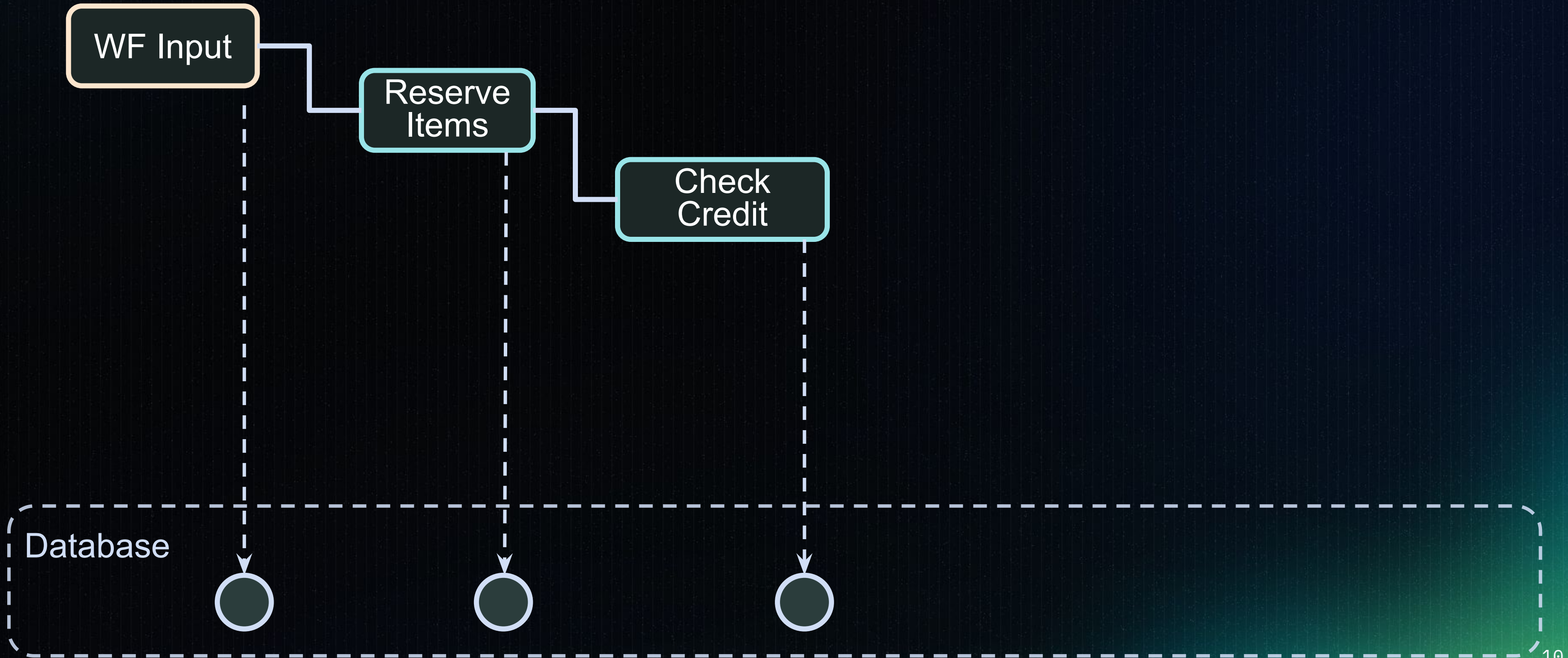
- Data corruption
- Duplication
- Wasted compute resources
- Sl.o..w...

Goal: Once a workflow starts, it should complete durably even across failures

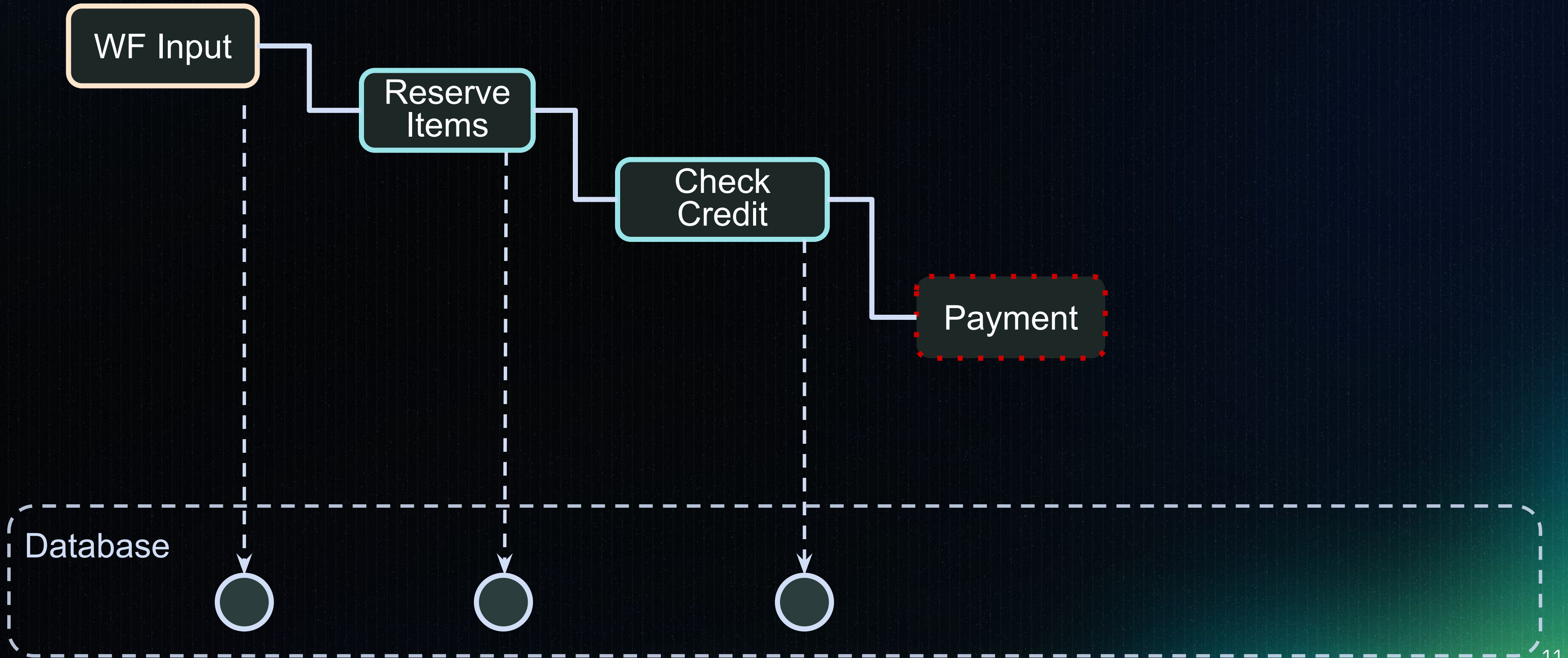
How Durable Workflows Can Help

- **D**urability in ACID
- Implementation is typically by logging/replay (DBMS techniques)

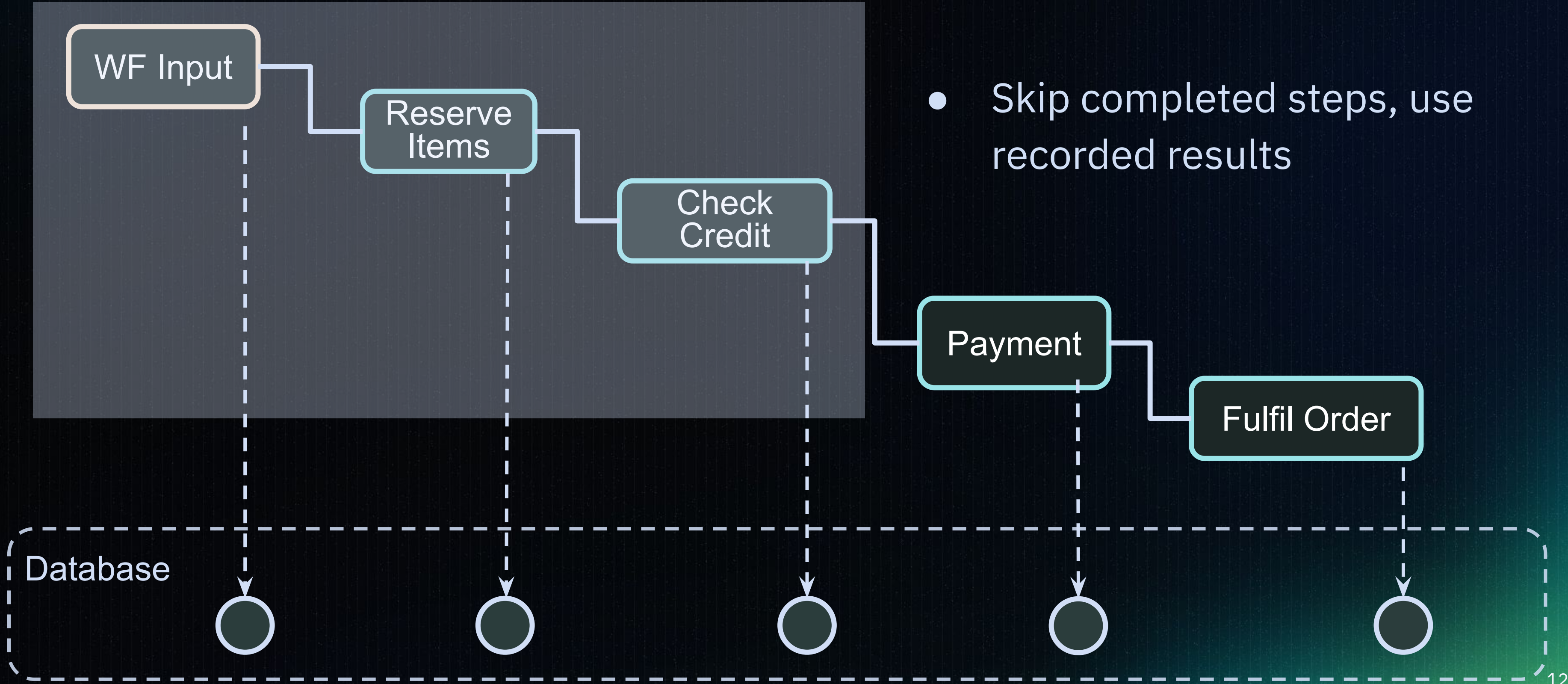
How It Works: Persist State in a Database



How It Works: Persist State in a Database



How It Works: Resume from Where Left Off



Durability Is Not Enough!

What happens if...

- A step fails in the middle of a workflow?
- Multiple concurrent workflows conflict?

Workflows Need To Be...

- **A**tomic (all or nothing)
- **C**onsistent (for compensation within a workflow)
- **D**urable (to avoid redoing work)
- **C**orrect (for compensation across concurrent workflows)

ACID → AC/DC

What Happens on a Workflow Error?

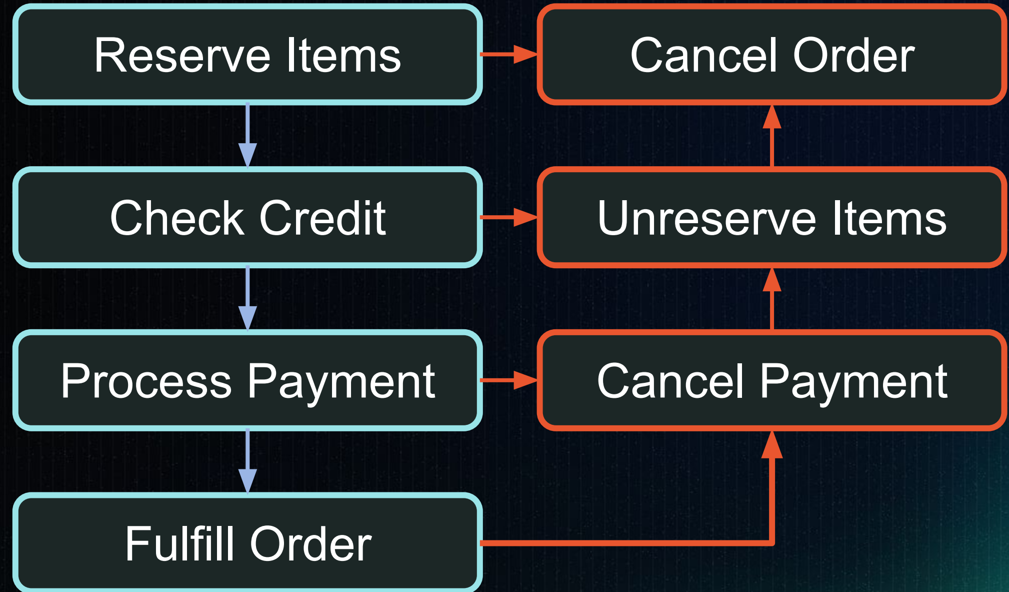
- Steps may fail:
  - Out of stock
  - User has bad credit
  - Bad address for fulfillment
  - …
- Must deal with errors – by an **alternative sequence** of steps, or **undo** some steps

The Reverse (Error) Path is 10x Harder

- Many ways a step can fail, only one way it can succeed
- Error handling is app specific; difficult to automate

Workflows Must Be Atomic

- Fully committed or fully rolled back
- Otherwise state is inconsistent



Support for Workflow Atomicity

- **Option 1:** Transactional locking
  - But requires holding long-term locks
  - Doesn't work for external API calls
- **Option 2:** Compensation / Sagas (Sigmod 1987 -- Garcia-Molina and Salem)
- Long running workflows are more efficient with compensation (see the paper)

Support for Workflow Atomicity w/o Concurrency

- Has to guarantee compensation is self **consistent** – i.e. compensation performs the correct inverse

- i.e. the compensation for increment is decrement and a 5% increase is a 4.76190% decrease

Support for Workflow Atomicity with Concurrency

- Compensation is tricky when someone else may have changed the state
- One must remove the side effect of the errant workflow without conflicting with other concurrent workflows
- Details in the paper along with the notion of **correctness**

Other Considerations

- Irreversible steps (withdraw money from an ATM)
- User stalls (human in the loop)
- External systems (Paypal et. al.)
    - Need to support idempotence and compensation

Future Work

- Tighten up the AC/DC definitions, formalize **correctness**
  - Similar to ANSI SQL isolation levels, but for workflows
- Explore AC/DC applications to security and privacy
  - E.g., delete all PII under GDPR
- Explore implementations for higher isolation and without sacrificing too much performance
- Speculative "what-if" analysis and automated bug detection

Thanks!

- Learn more: dbos.dev
- Open source: github.com/dbos-inc ⭐
- Prototype of AC/DC: github.com/DBOS-project/dbos-transact-py