IPADS
INSTITUTE OF PARALLEL AND DISTRIBUTED SYSTEMS

Automated Verification of Idempotence for Stateful Serverless Applications

Haoran Ding, Zhaoguo Wang, Zhuohao Shen, Rong Chen, Haibo Chen

Institute of Parallel and Distributed Systems, SEIEE, Shanghai Jiao Tong University

Serverless Computing

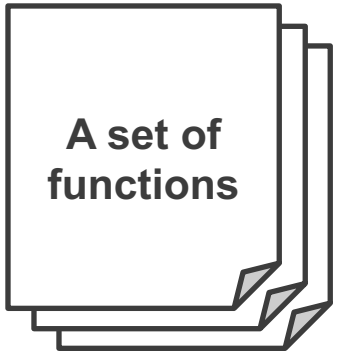
Developer

User

User

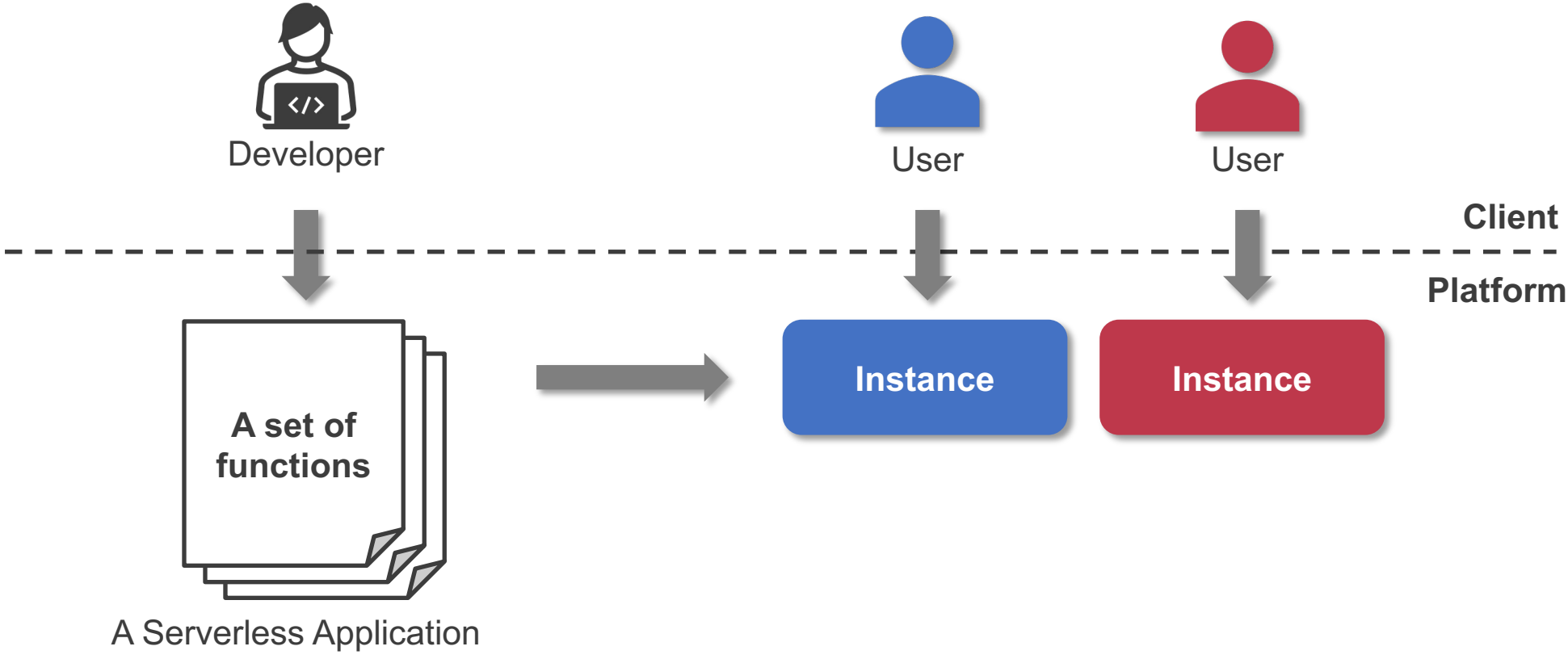
Client

Platform

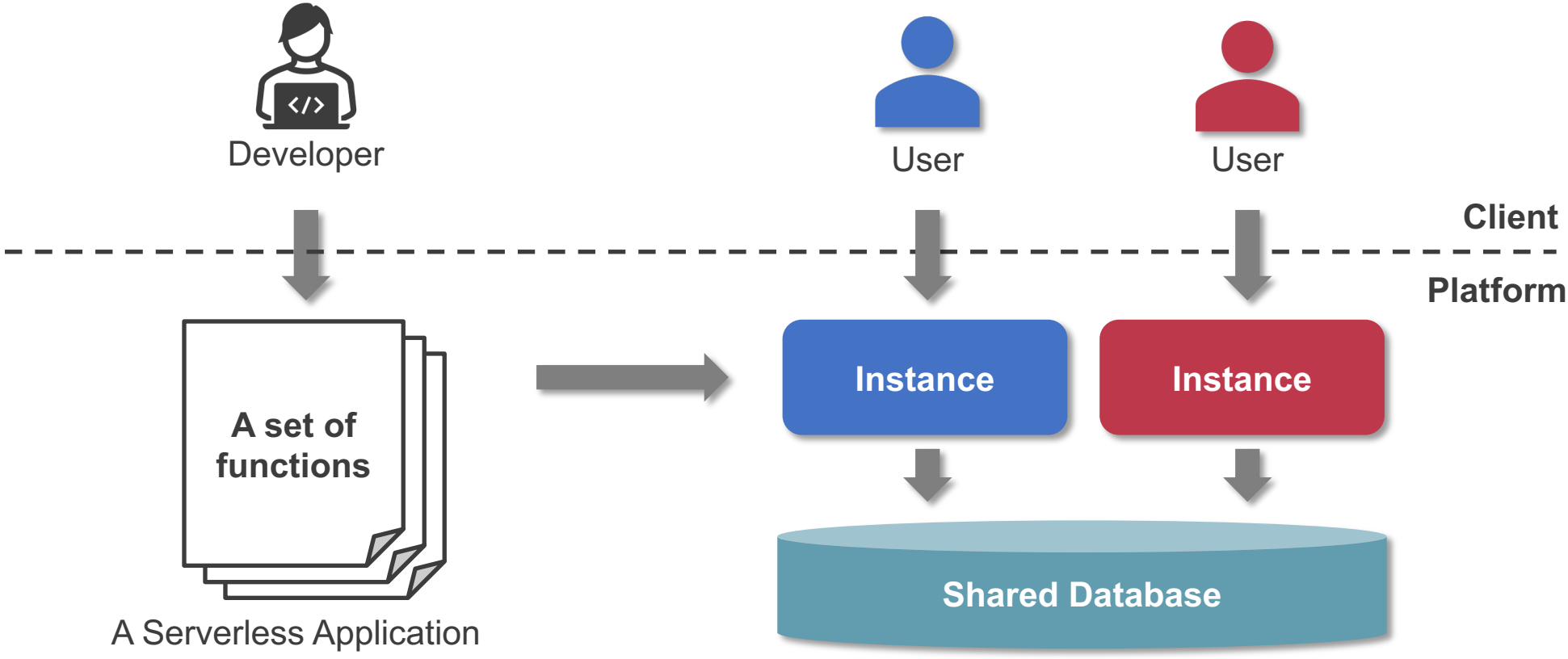


A Serverless Application

Serverless Computing



Serverless Computing



Problem of Retry-Based Fault Tolerance

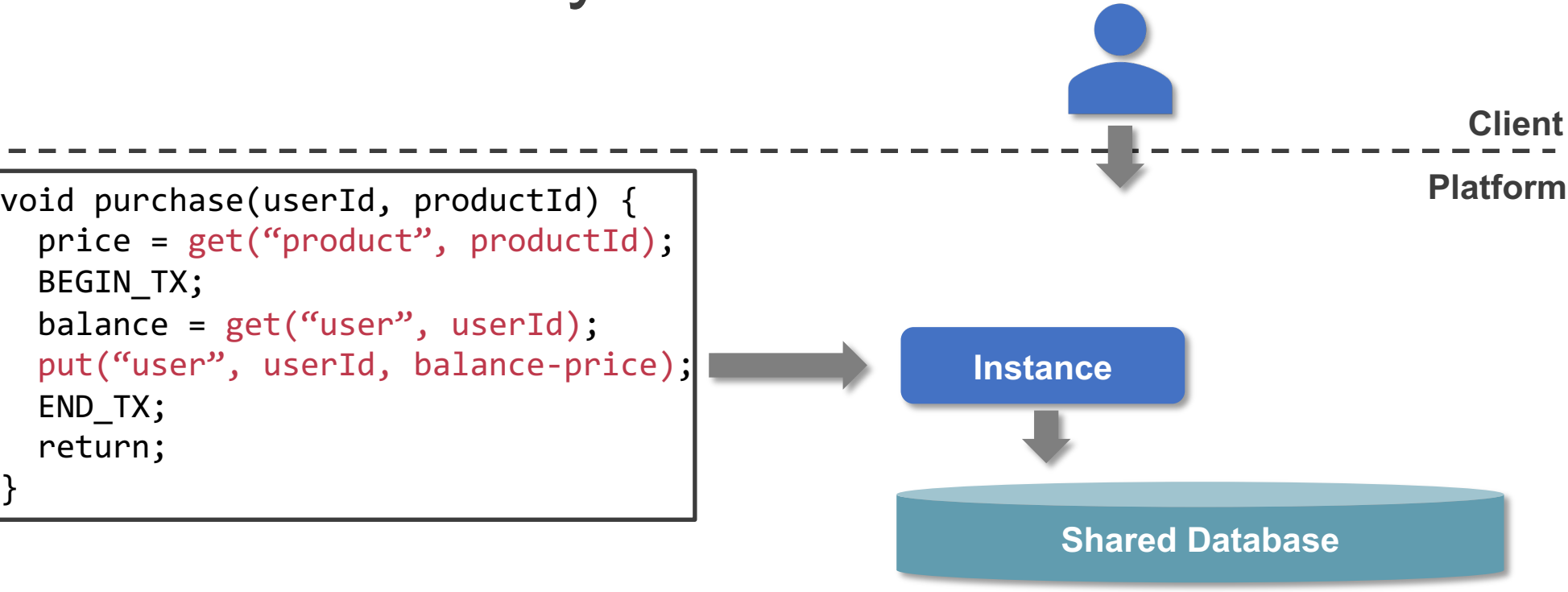
Client

Platform

```
void purchase(userId, productId) {
  price = get("product", productId);
  BEGIN_TX;
  balance = get("user", userId);
  put("user", userId, balance-price);
  END_TX;
  return;
}
```

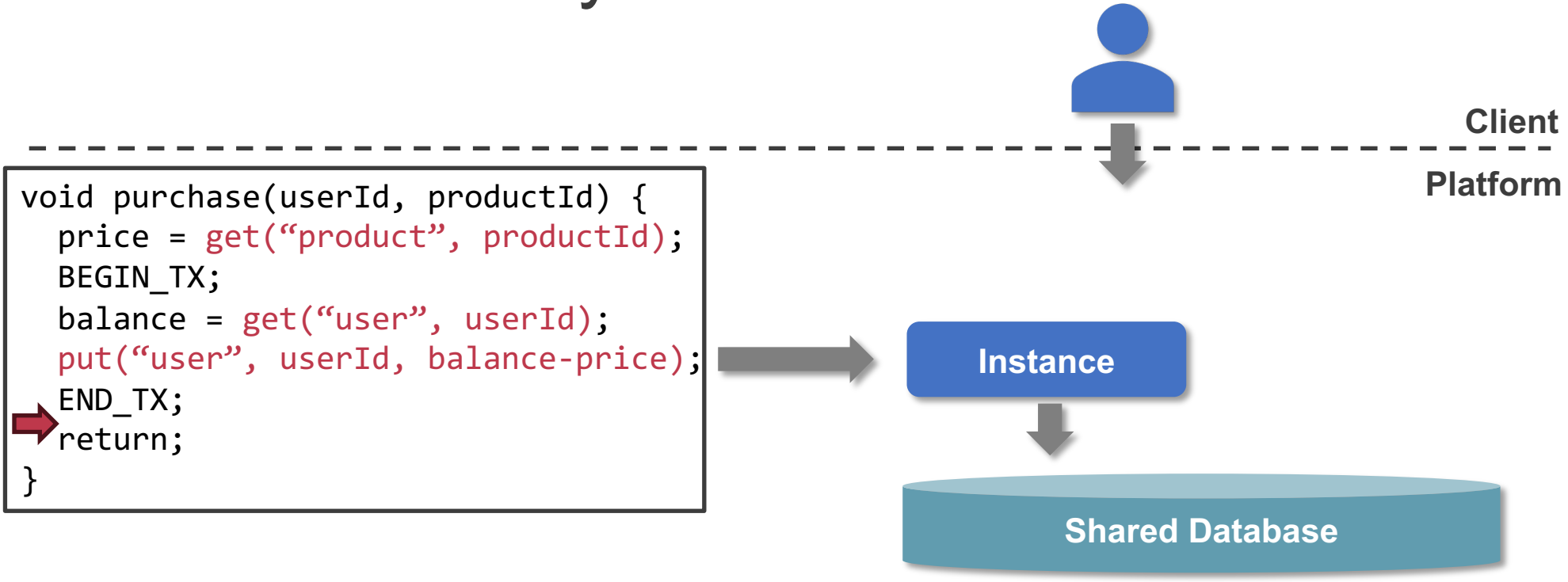
Problem of Retry-Based Fault Tolerance

```
void purchase(userId, productId) {
  price = get("product", productId);
  BEGIN_TX;
  balance = get("user", userId);
  put("user", userId, balance-price);
  END_TX;
  return;
}
```



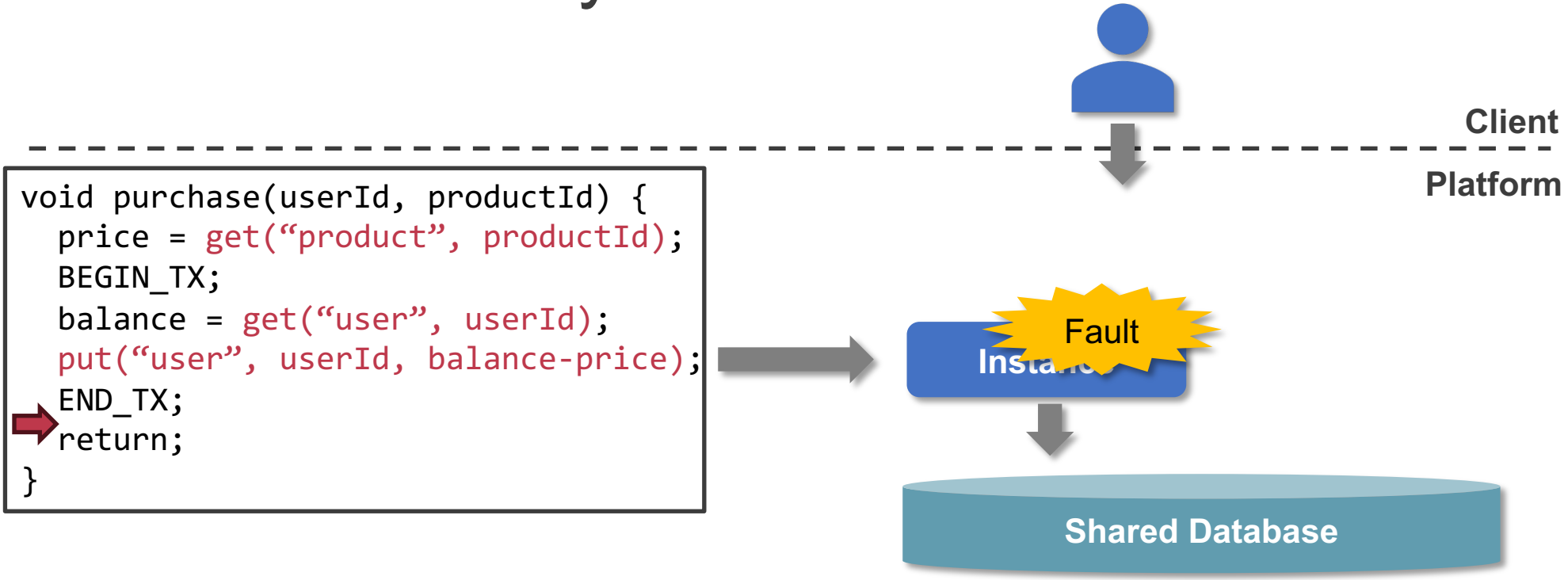
Problem of Retry-Based Fault Tolerance

```
void purchase(userId, productId) {
  price = get("product", productId);
  BEGIN_TX;
  balance = get("user", userId);
  put("user", userId, balance-price);
  END_TX;
  return;
}
```



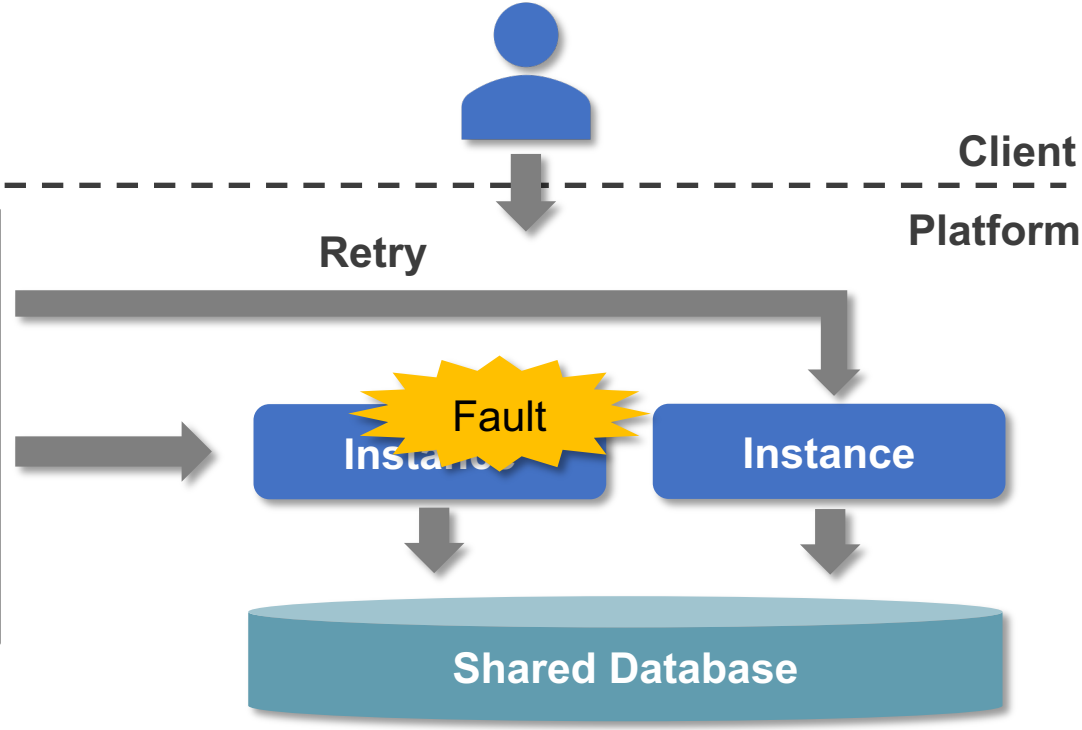
Problem of Retry-Based Fault Tolerance

```
void purchase(userId, productId) {
  price = get("product", productId);
  BEGIN_TX;
  balance = get("user", userId);
  put("user", userId, balance-price);
  END_TX;
  return;
}
```



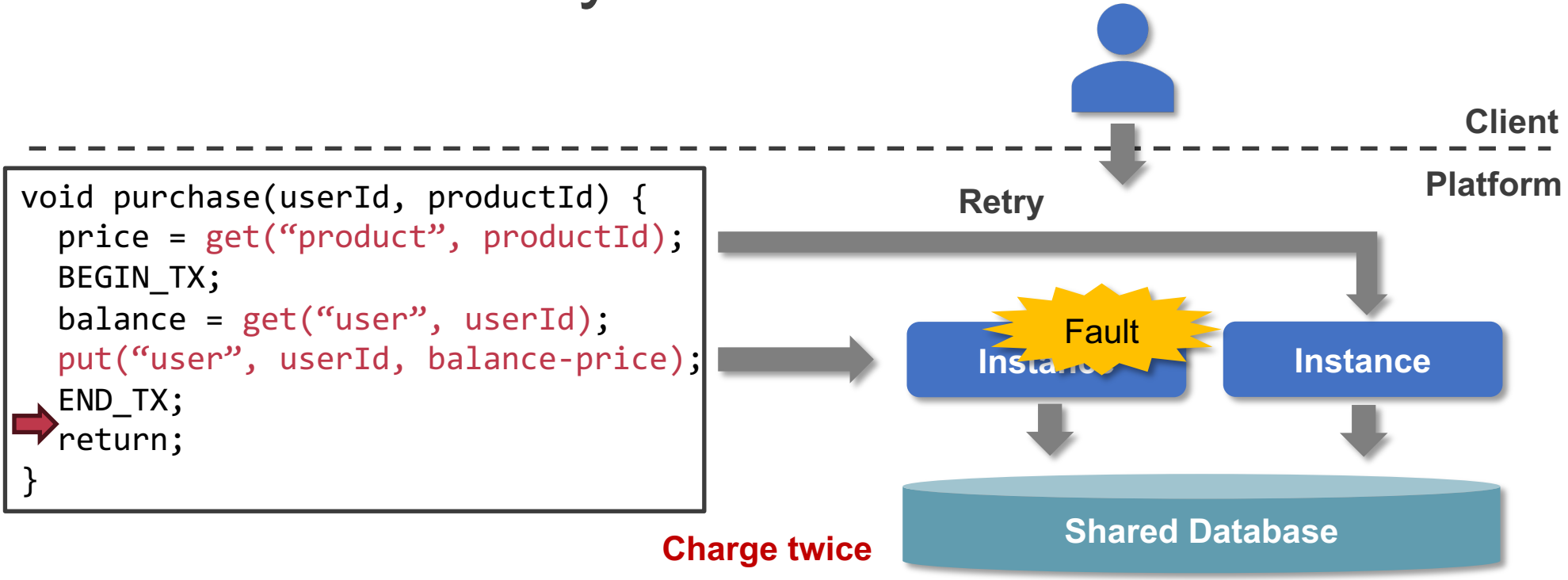
Problem of Retry-Based Fault Tolerance

```
void purchase(userId, productId) {
  price = get("product", productId);
  BEGIN_TX;
  balance = get("user", userId);
  put("user", userId, balance-price);
  END_TX;
  return;
}
```



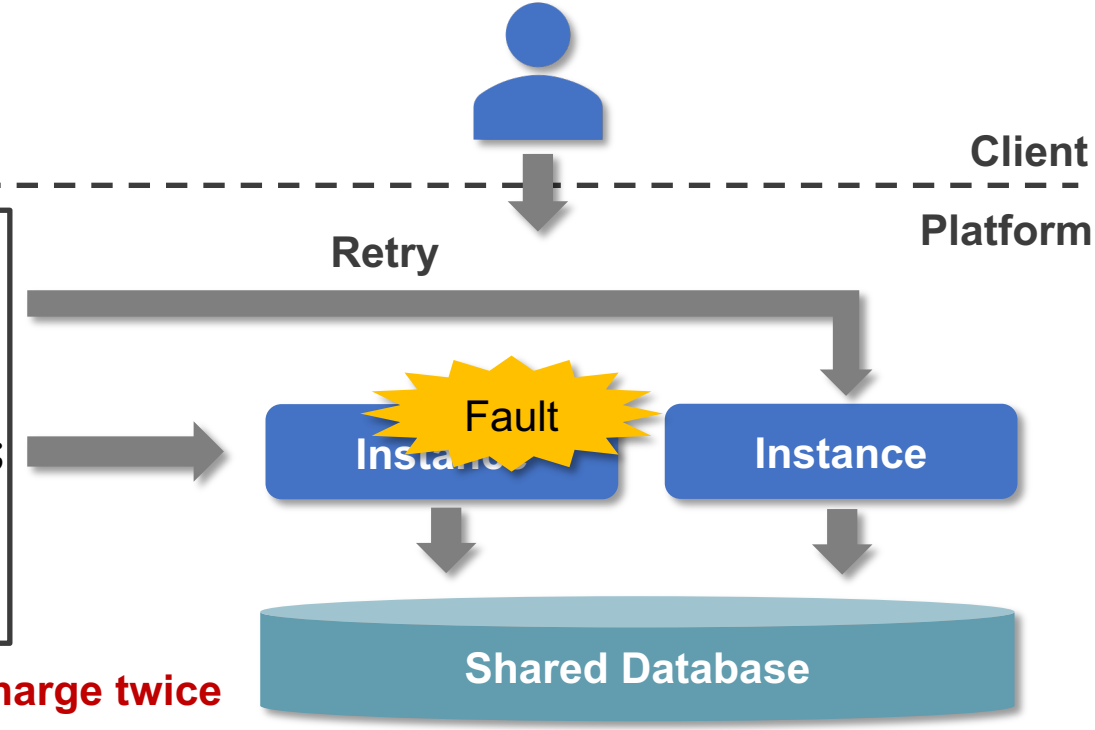
Problem of Retry-Based Fault Tolerance

```
void purchase(userId, productId) {
  price = get("product", productId);
  BEGIN_TX;
  balance = get("user", userId);
  put("user", userId, balance-price);
  END_TX;
  return;
}
```



Problem of Retry-Based Fault Tolerance

```
void purchase(userId, productId) {
  price = get("product", productId);
  BEGIN_TX;
  balance = get("user", userId);
  put("user", userId, balance-price);
  END_TX;
  return;
}
```



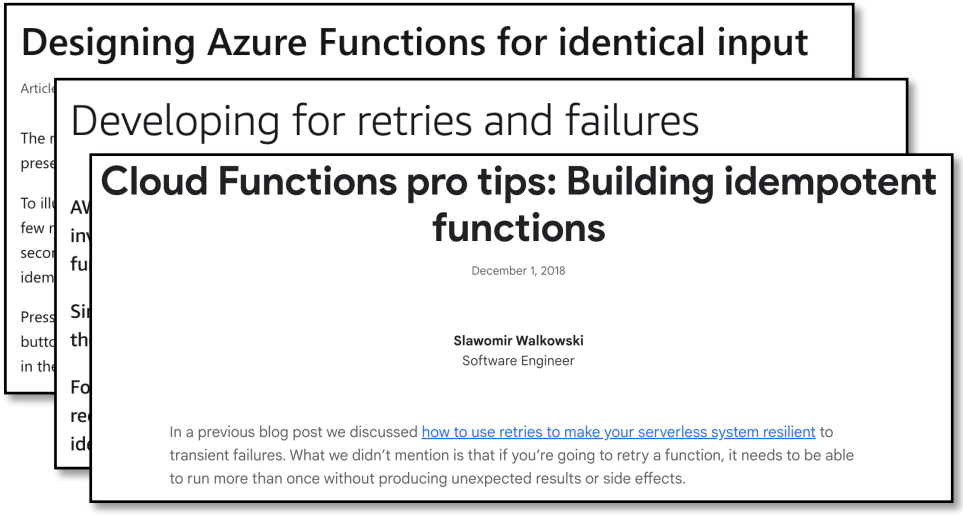
Repeated execution ➔ incorrect execution results

We need idempotence, but at what cost?

Idempotence

An application should expose the same behavior regardless of retry

Currently, developers are mandated to ensure idempotence



Developer

1. Is the application idempotent?
2. How to fix idempotence issues?

Existing Work

AWS Powertool

Targets retries caused by only exception thrown by functions

- It cannot guarantee idempotence under other cases, such as hardware crash

FSCQ, Yggdrasil, Perennial, GoJournal, …

Verify idempotence of only a sequential function

- However, serverless functions usually run concurrently

Beldi, Boki, …

Ensure the exactly-once execution of all database operations

- However, their mechanisms introduce heavy performance cost

Flux

Goal	Automatically verify and ensure idempotence of serverless applications

Challenge #1

A formal definition of idempotence for concurrent functions is desired but missing

Challenge #2

Concurrency and arbitrary failure hinder automated verification

Challenge #3

Ensuring idempotence while introducing minor performance cost is difficult

Flux

Goal Automatically verify and ensure idempotence of serverless applications

1 **Formal definition of idempotence: idempotence consistency**
- Support concurrent functions

2 **Automated verifier of idempotence consistency**
- Ensure soundness

3 **Accurately identify root causes of issues via verification and repair them**
- Ensure idempotence consistency while reducing unnecessary performance cost

Reuse existing definition of idempotence?

Idempotence definition for a sequential function

A function always produces the same program state and return value regardless of retry

Idempotent sequential code 1

```
void write(string key, int val) {
  put("data", key, val);
  return;
}
```

Perform the same update regardless of retry

Idempotent sequential code 2

```
int read(string key) {
  int val = get("data", key);
  return val;
}
```

No side effect

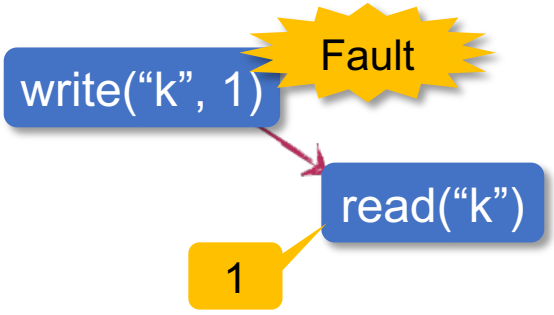
Limitation of the Existing Definition

write(“k”, 1)

Happen before

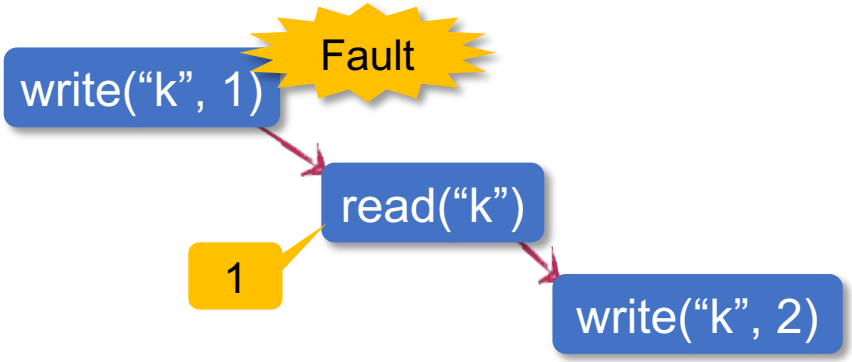
Limitation of the Existing Definition

Limitation of the Existing Definition



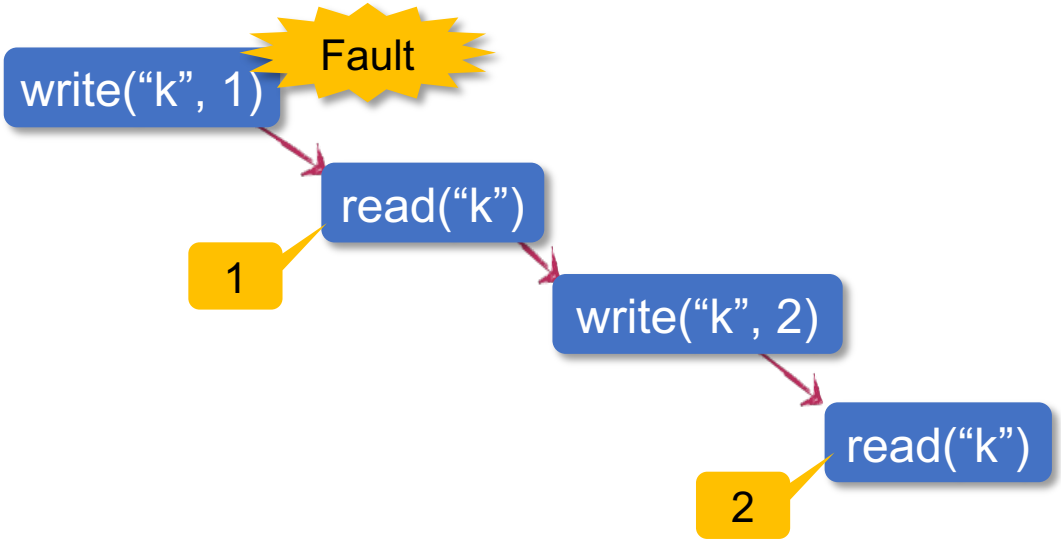
Happen before

Limitation of the Existing Definition



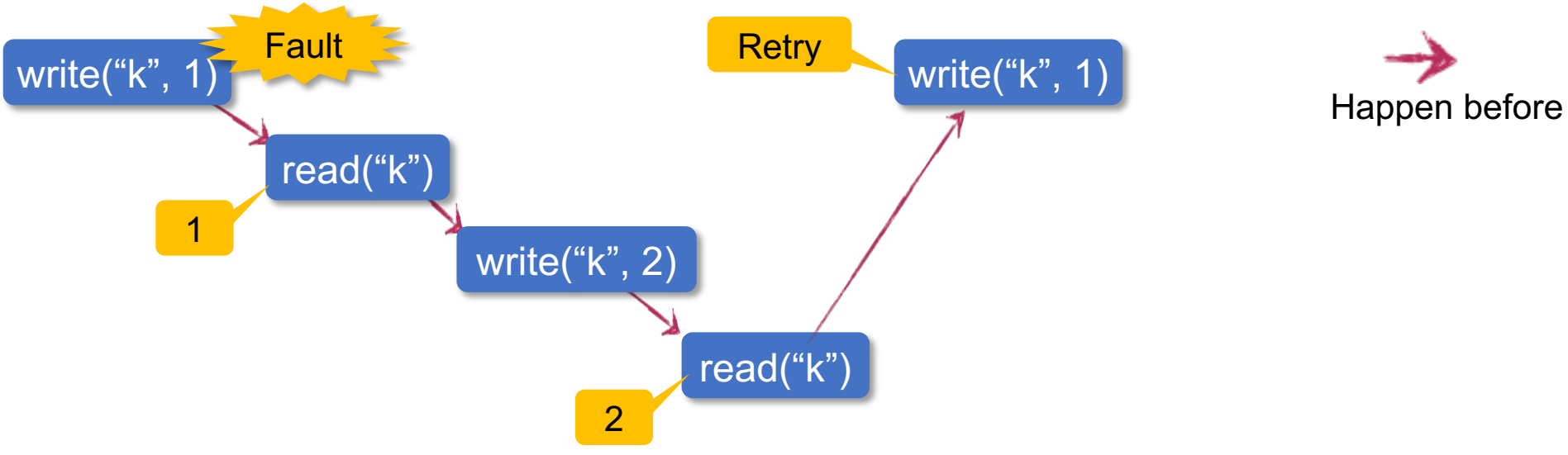
Happen before

Limitation of the Existing Definition

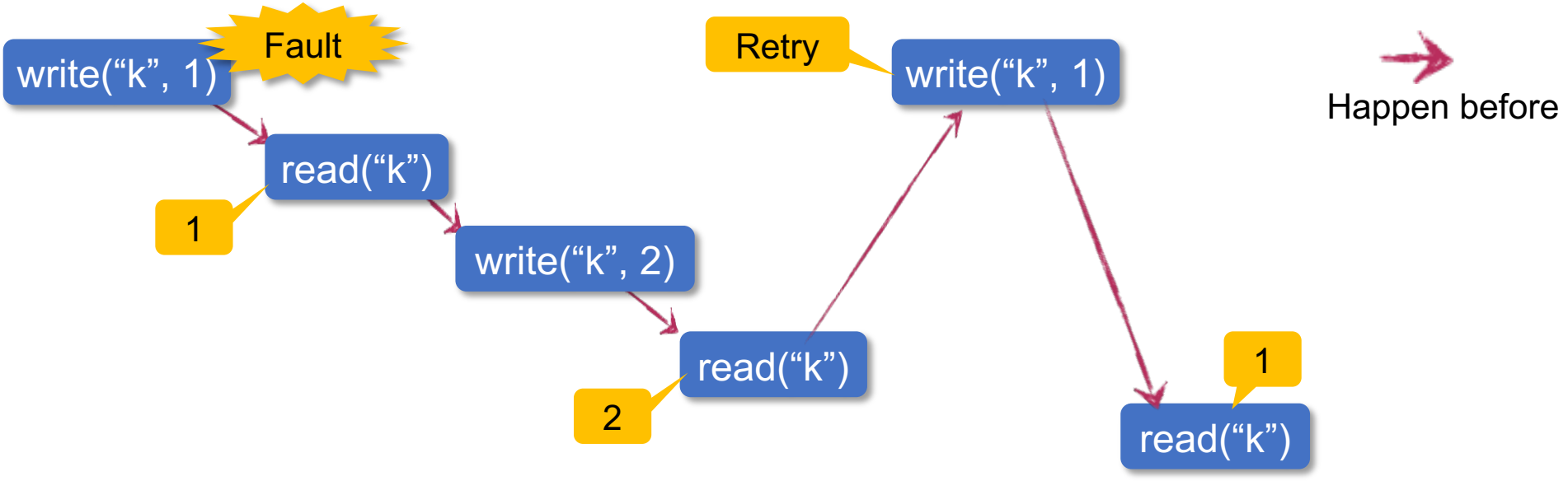


Happen before

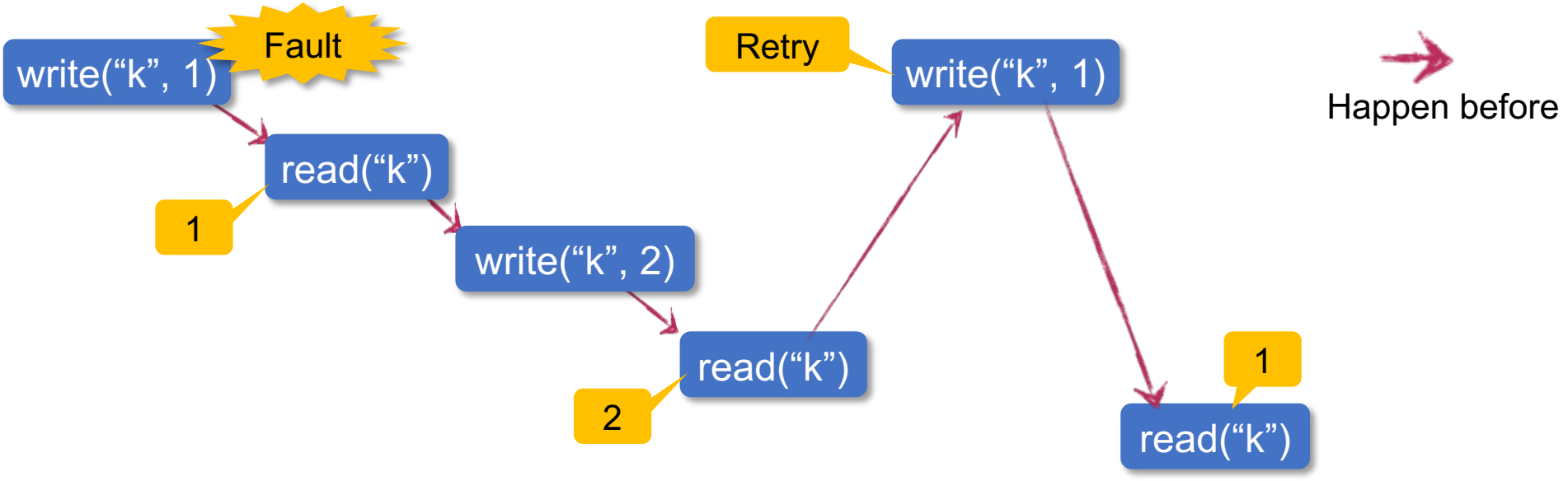
Limitation of the Existing Definition



Limitation of the Existing Definition

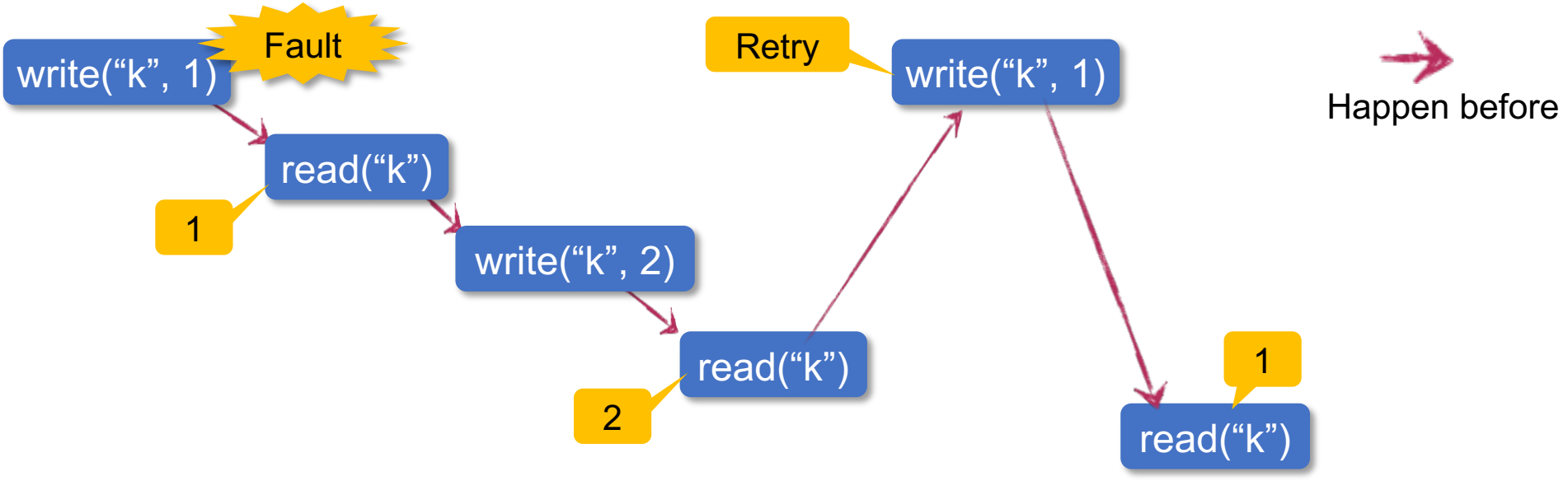


Limitation of the Existing Definition



From the view of clients, there are only two requests for “write”, but the value is flipped for three times

Limitation of the Existing Definition



From the view of clients, there are only two requests for "write", but the value is flipped for three times

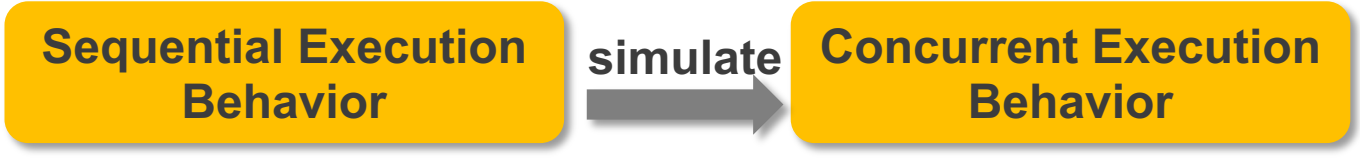
Idempotence is not ensured under concurrency

Rethink the Definition of Idempotence

The requirement for the definition of idempotence

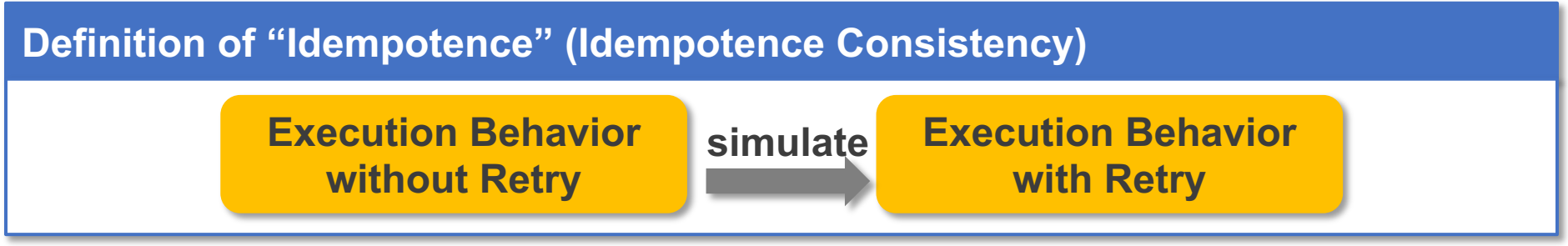
Define the acceptable behavior of a concurrent execution under retry

Definition of "Serializability" (Consistency Model)



Inspired by "serializability", we define "idempotence" as a consistency model

Our Definition: Idempotence Consistency



Automated Reasoning of Concurrency

Challenge	There are infinite interleavings to consider
Basic Idea	**Compositional proof technique** **Each function satisfies P** → Implies → **The application satisfies idempotence consistency** func1 ✓ func2 ✓ func3 ✓

? **What is the property “P”?**

Idempotence Simulation

f^* models the function f running with failure and retry

- Insert the following code after each database operation in f

```
bool retry = random();
if (retry) {
  reset_local_state();
  goto BEGIN;
}
```

Model failure

Model retry

Idempotence simulation: each execution of f^* can be simulated by an execution of f

Theorem

Given a function set F, if each function f in F satisfies idempotence simulation, then F satisfies idempotence consistency.

Automated Reasoning of Failure

Challenge	Arbitrary failures also cause infinite interleavings

Basic Idea

Failure reduction: Assume failure occurs at most once

```
bool retry = random();
if (retry && (retry_num == 0)) {
  retry_num = 1;
  reset_local_state();
  goto BEGIN;
}
```

Retry is allowed only when it has not occurred before

Theorem

If a function satisfies idempotence simulation with at most one failure, then it satisfies idempotence simulation with arbitrary failure.

How to repair inconsistent applications?

Existing work ensures exactly-once execution of all operations via logs

- Each get always returns the same value as the first execution on retry
- Each put does nothing on retry

Key observation

- Logging only some of database operations can still ensure idempotence consistency

```
void purchase(userId, productId) {
  price = get("product", productId);
  BEGIN_TX;
  balance = get("user", userId);
  put("user", userId, balance-price);
  END_TX;
  return;
}
```

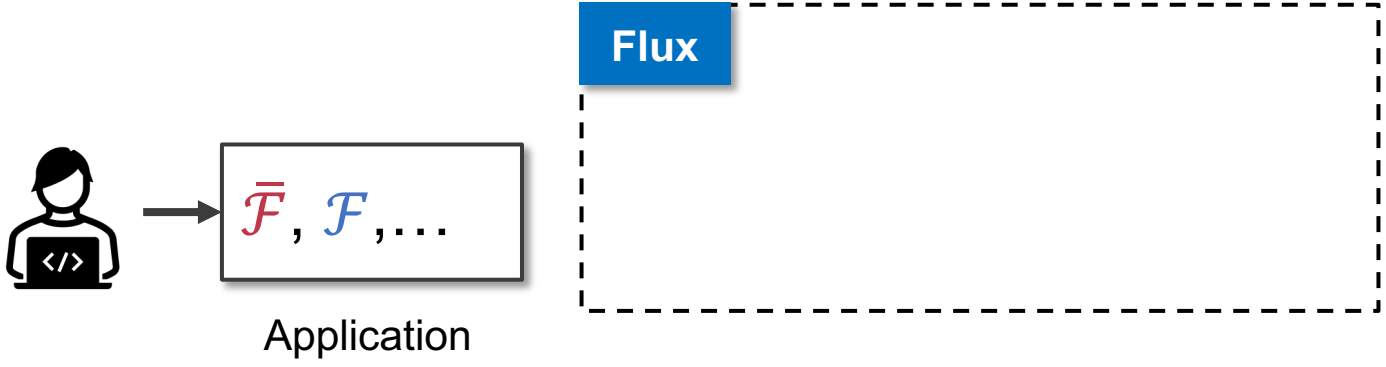
Not logging "get" also
ensures idempotence consistency

Advisor

Basic Idea

Find a set of operation such that

- Logging all operations in the set can ensure idempotence consistency
- Unnecessary logs are not in the set



Serverless Platform (e.g., AWS Lambda, Azure, GCP)

$\bar{\mathcal{F}}$:Non-idempotent func

$\mathcal{F}$:Idempotent func

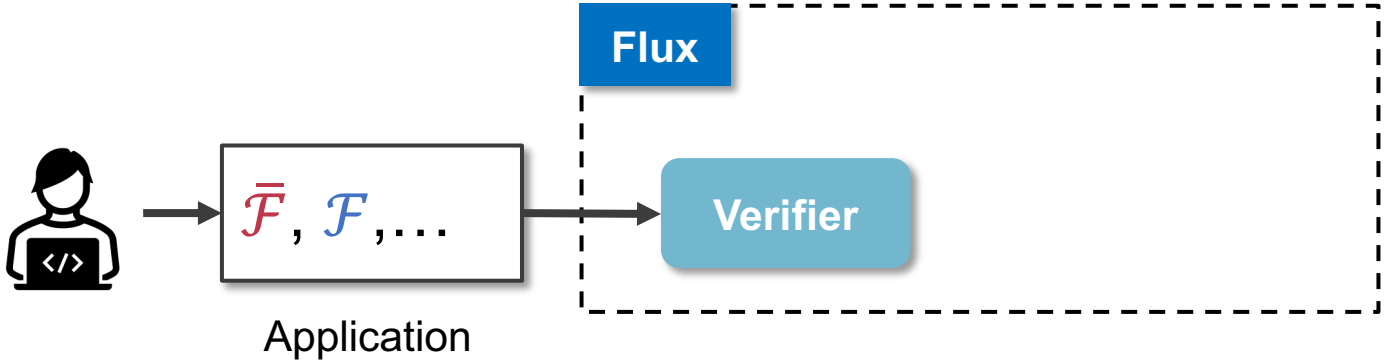
$[\mathcal{F}]$: Func with logging

Advisor

Basic Idea

Find a set of operation such that

- Logging all operations in the set can ensure idempotence consistency
- Unnecessary logs are not in the set



Serverless Platform (e.g., AWS Lambda, Azure, GCP)

$\bar{\mathcal{F}}$:Non-idempotent func

$\mathcal{F}$:Idempotent func

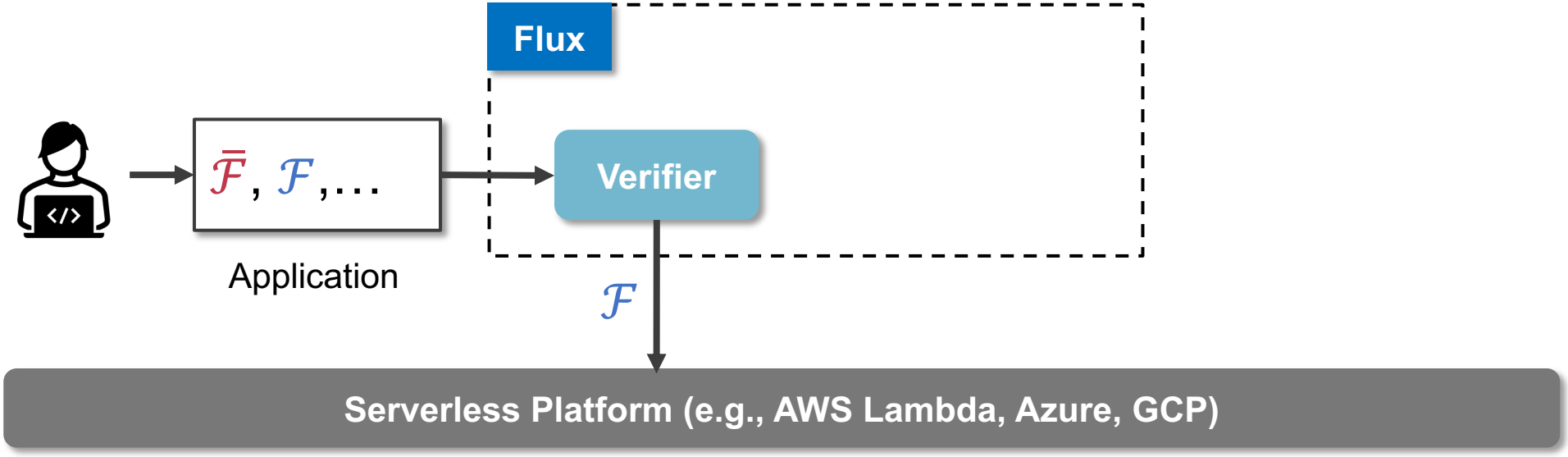
$[\mathcal{F}]$: Func with logging

Advisor

Basic Idea

Find a set of operation such that

- Logging all operations in the set can ensure idempotence consistency
- Unnecessary logs are not in the set



$\bar{\mathcal{F}}$:Non-idempotent func

$\mathcal{F}$:Idempotent func

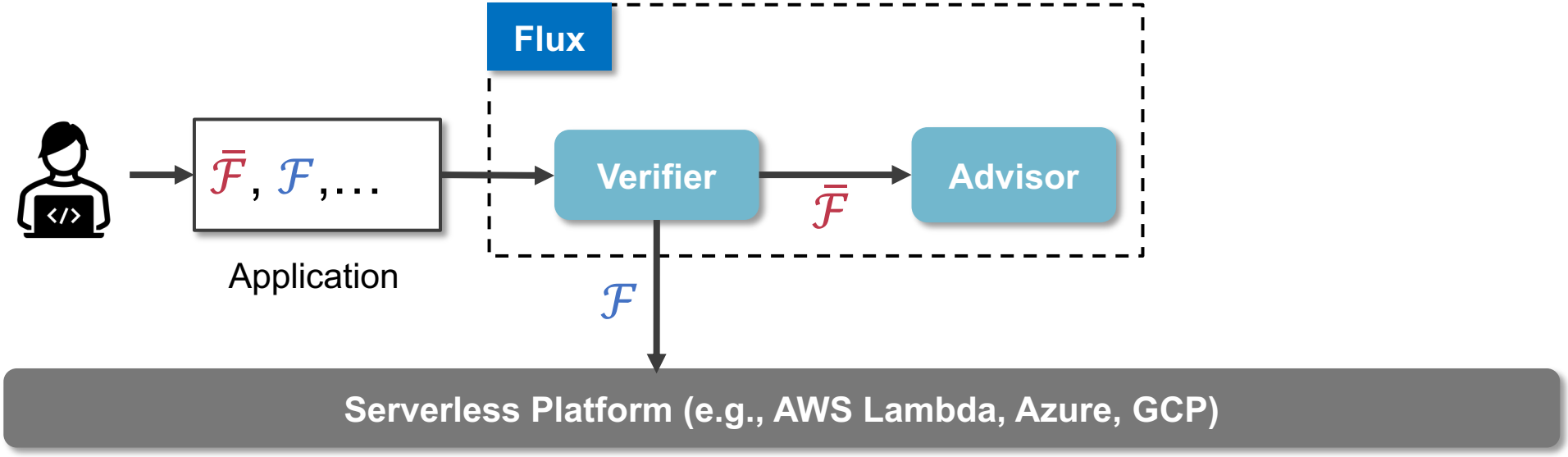
$[\mathcal{F}]$: Func with logging

Advisor

Basic Idea

Find a set of operation such that

- Logging all operations in the set can ensure idempotence consistency
- Unnecessary logs are not in the set



$\bar{\mathcal{F}}$:Non-idempotent func

$\mathcal{F}$:Idempotent func

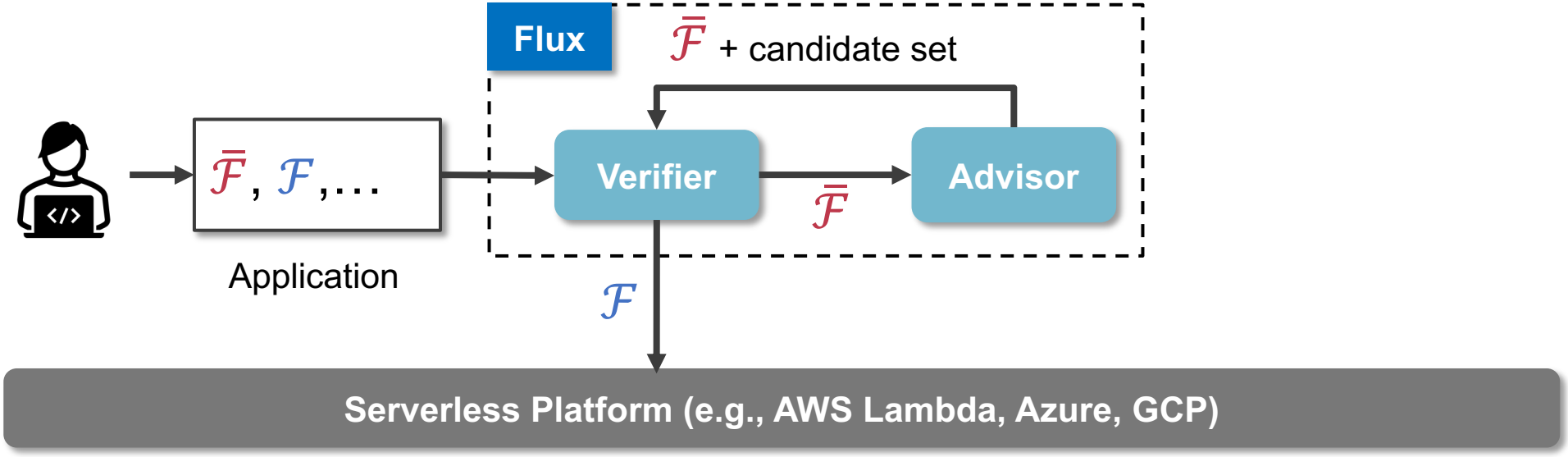
$[\mathcal{F}]$: Func with logging

Advisor

Basic Idea

Find a set of operation such that

- Logging all operations in the set can ensure idempotence consistency
- Unnecessary logs are not in the set



$\bar{\mathcal{F}}$:Non-idempotent func

$\mathcal{F}$:Idempotent func

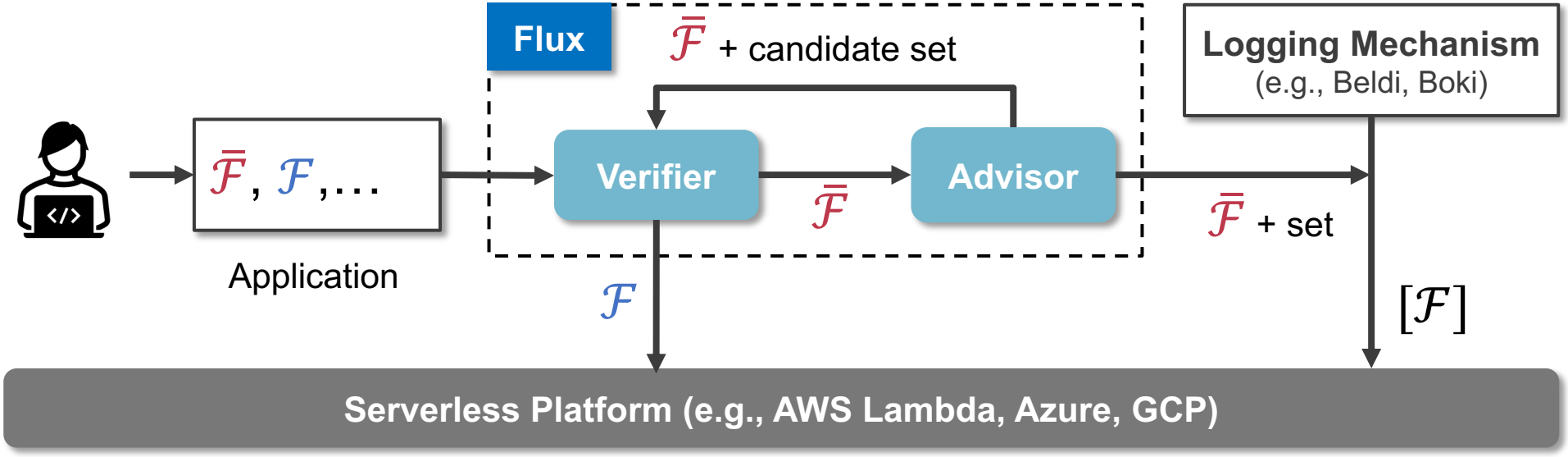
$[\mathcal{F}]$: Func with logging

Advisor

Basic Idea

Find a set of operation such that

- Logging all operations in the set can ensure idempotence consistency
- Unnecessary logs are not in the set



$\bar{\mathcal{F}}$:Non-idempotent func

$\mathcal{F}$:Idempotent func

$[\mathcal{F}]$: Func with logging

Evaluation

1. **What is the cost of verifier and advisor?**

2. **How much performance can Flux improve?**

Applications	AWS serverless applications repository, popular GitHub repository, … • 27 serverless applications • 79 serverless functions

What is the cost of verifier and advisor?

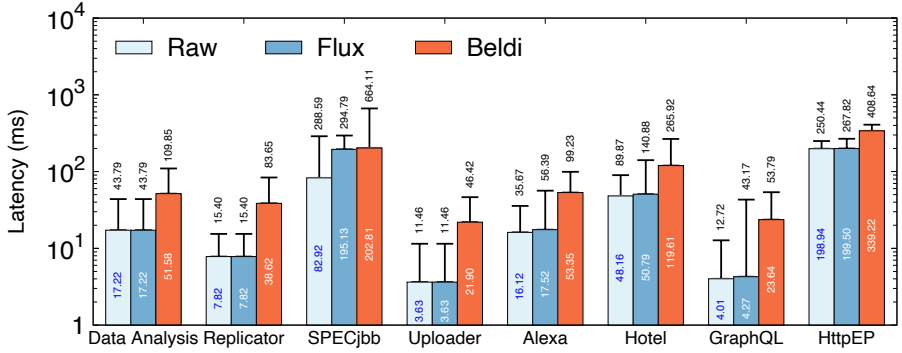
Verifier

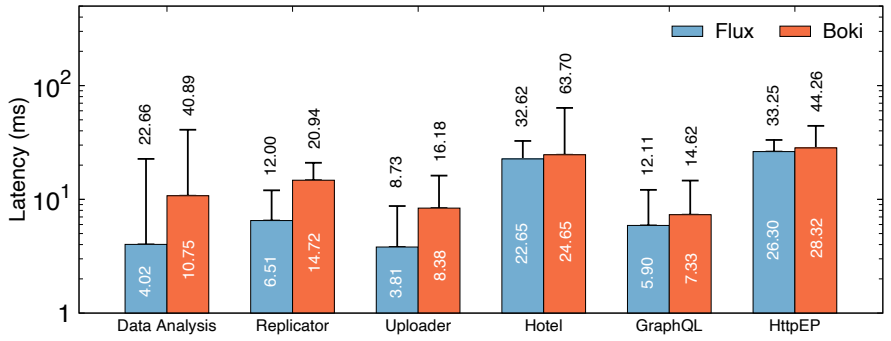
- The verification time < 110s
- Find all idempotence issues

Advisor

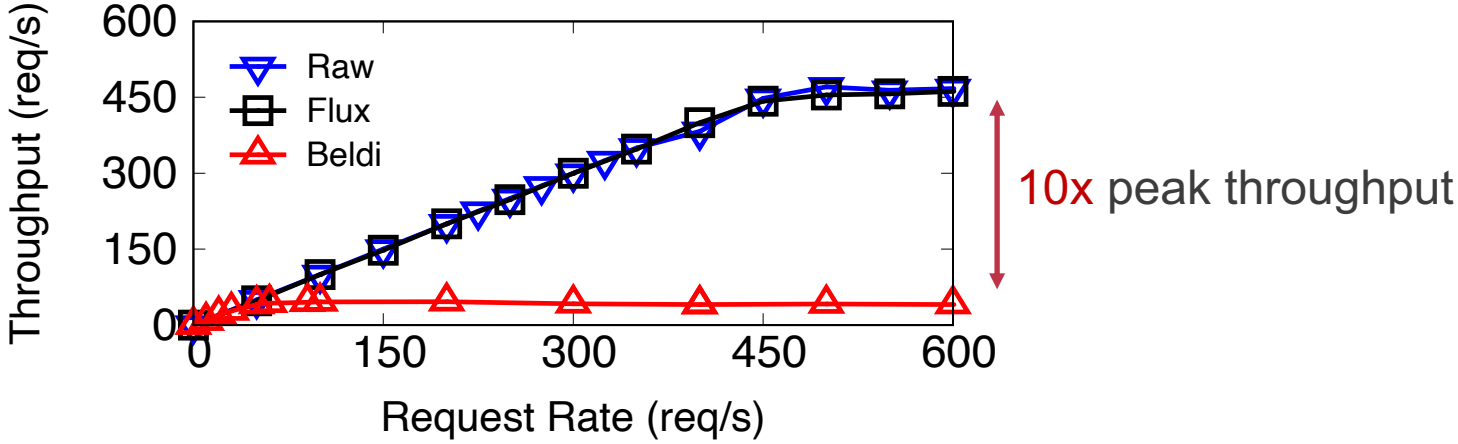
- The optimization time < 90s

How much performance can Flux improve?





6x lower latency



Summary

Flux

- A new definition of idempotence for stateful serverless applications
- The first automated verifier of idempotence consistency
- An algorithm to reduce performance cost of fault tolerance based on verification

Thank You