ACIDRain: Concurrency-Related Attacks on Database-Backed Web Applications

Todd Warszawski, Peter Bailis
Stanford University

A: Atomicity

C: Consistency

I: Isolation

D: Durability

Problems with concurrent operations:

1. lost update

2. dirty read

3. non-repeatable read

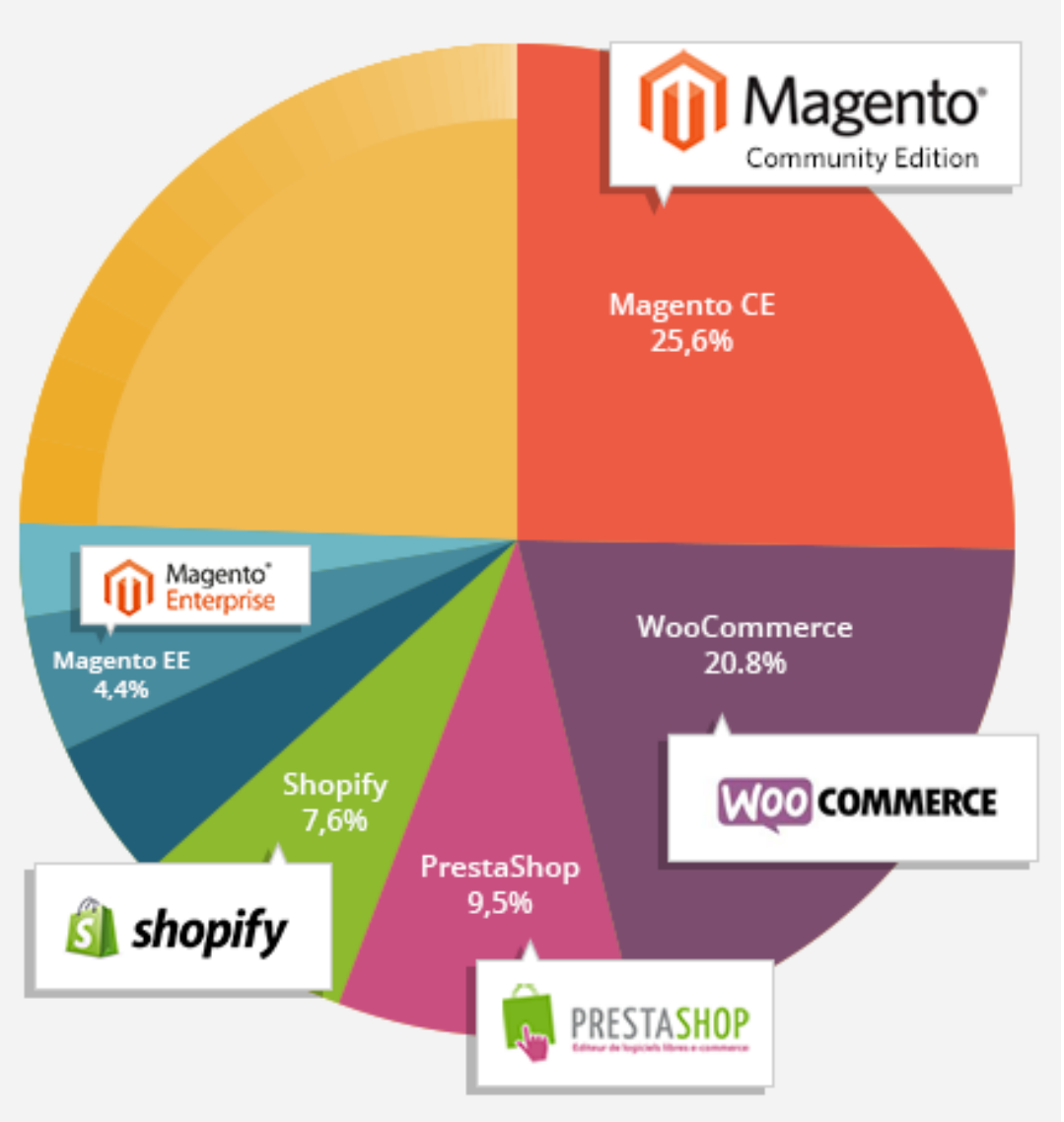
4. phantom row

concurrency / Isolation	lost update	dirty read	non-repeatable read	phantom row
Read Uncommitted	N	Y	Y	Y
Read Committed	N	N	Y	Y
Repeatable Read	N	N	N	Y
Serializable	N	N	N	N

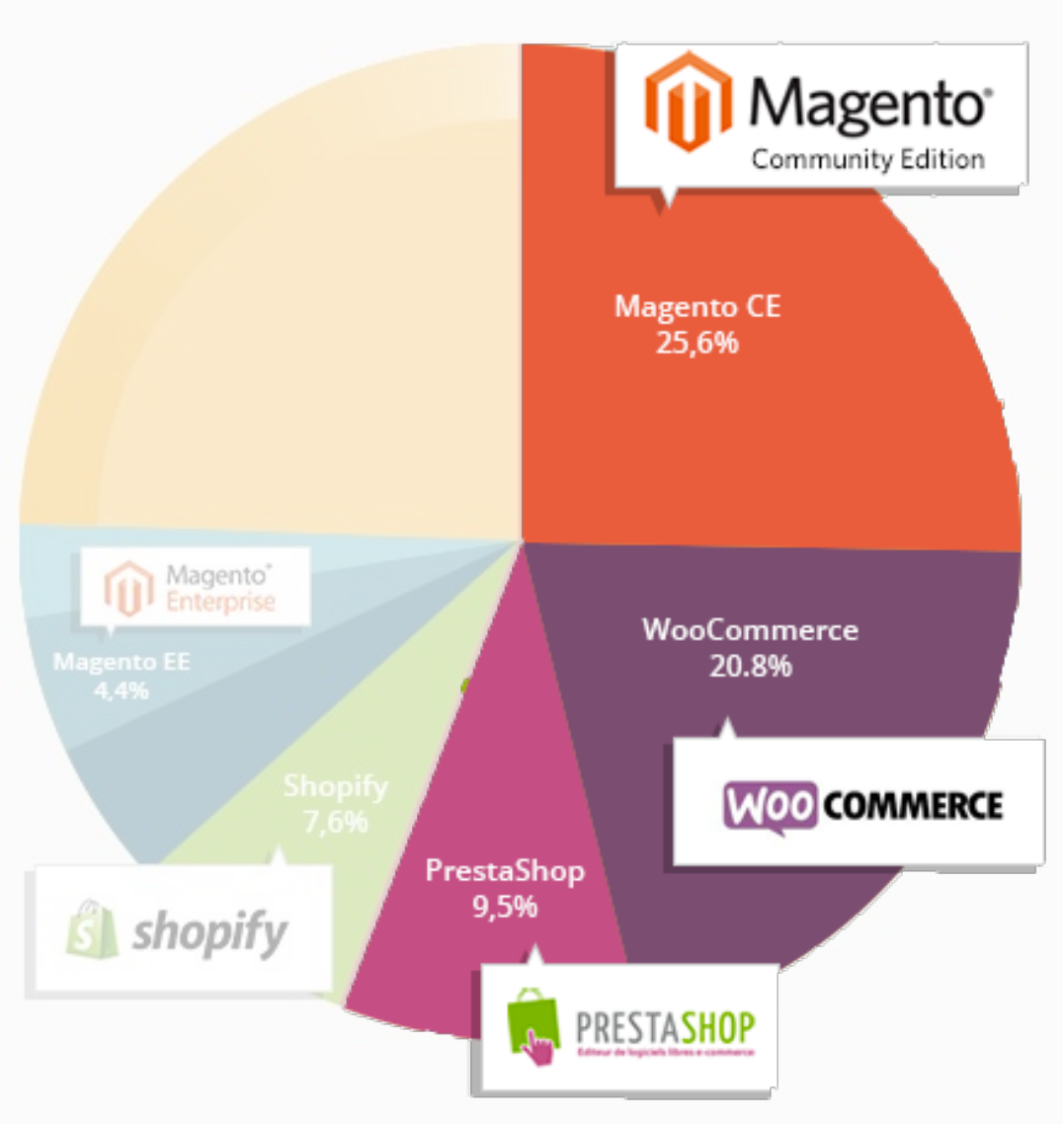
Explore Real World Transaction Usage

- Do programmers use transactions correctly?
- This paper: 22 new critical vulnerabilities due to incorrect transaction usage
  - Corrupt store inventory, overspend giftcards, steal items
- 50% of eCommerce sites (2M+) at risk

Analyzed Popular Sites



Analyzed Popular Sites



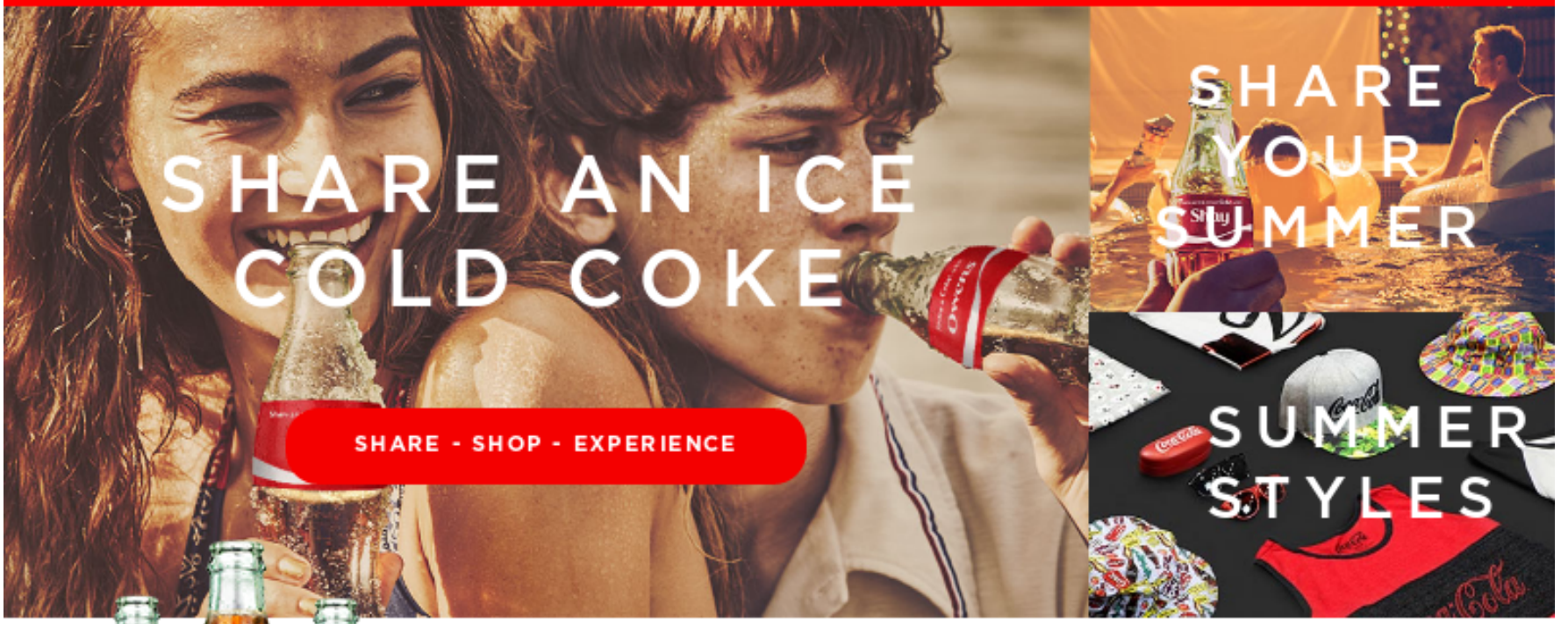
Plus 9 more!

Companies Using These Applications

Coca-Cola

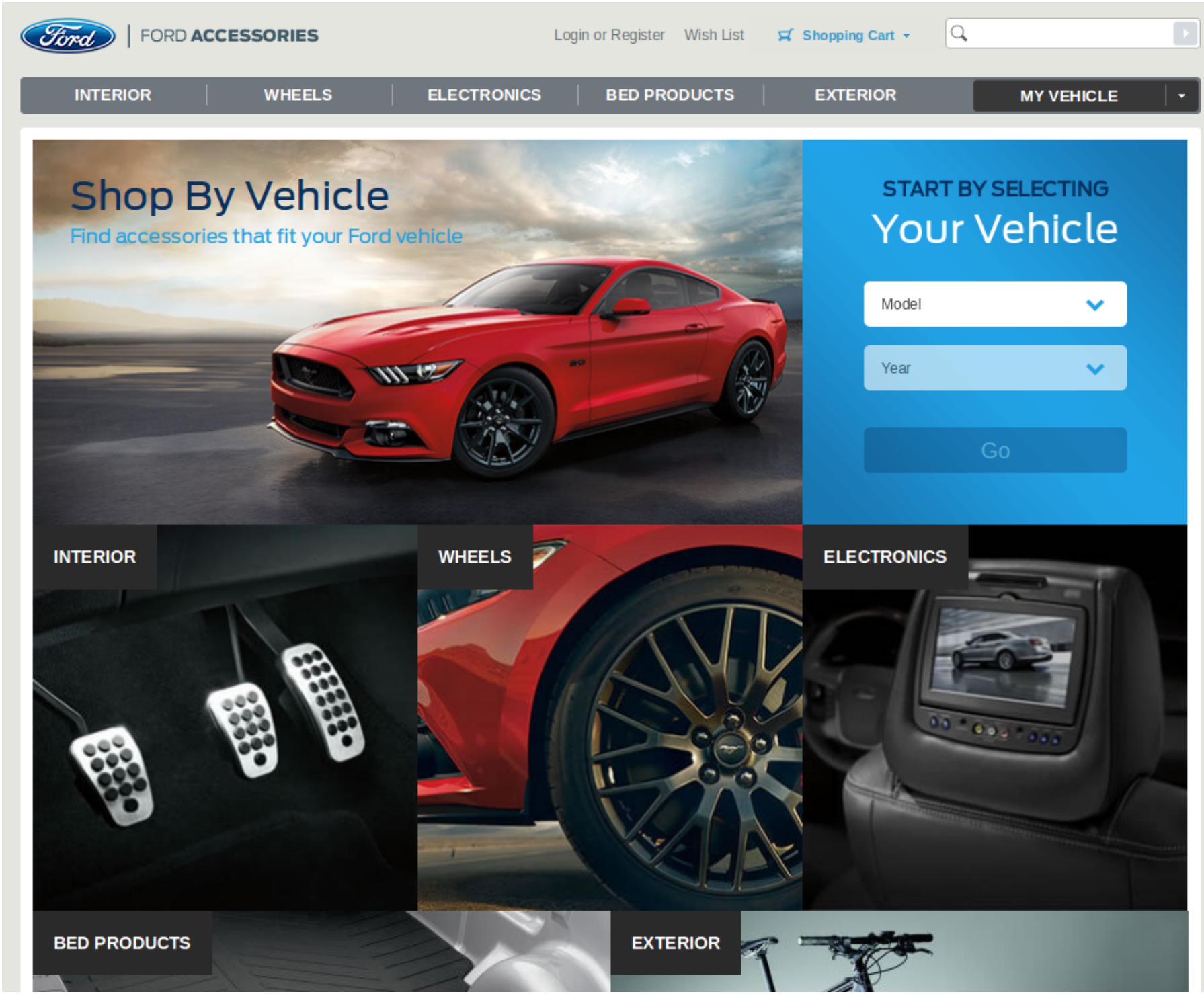
SHOP SUMMER CUSTOMIZE Search entire s

Save 10% off our new globally inspired styles through May 21 with code WORLD10 | Use code SHIP6 and get free shipping off 6 or more personalized bottles



Enter a name or phrase to create a custom 8 oz. bottle

CUSTOMIZE


FORD ACCESSORIES
Login or Register
Wish List
Shopping Cart
INTERIOR
WHEELS
ELECTRONICS
BED PRODUCTS
EXTERIOR
MY VEHICLE
Shop By Vehicle
Find accessories that fit your Ford vehicle
START BY SELECTING
Your Vehicle
Model
Year
Go
INTERIOR
WHEELS
ELECTRONICS
BED PRODUCTS
EXTERIOR

EXPLORE: TCS CLOSETS BLOGS CONTAINED HOME

CONTACT LOCATIONS POP! SIGN IN CART SHIP TO

The Container Store®

FREE SHIPPING
WHEN YOU SPEND $75 OR MORE*

Search now

SHOP ALL SALE elfa CLOSET STORAGE KITCHEN BATH TRAVEL SHELVING OFFICE GARAGE LAUNDRY TRASH & RECYCLING

TCS CLOSETS®

LIMITED TIME OFFER
4 FREE HOURS OF PROFESSIONAL ORGANIZING

LEARN MORE

HURRY!

15% OFF all closet essentials
Ends Soon!

Over 750 closet essentials on SALE

What do these vulnerabilities look like?

Flexcoin is shutting down. (March 3 2014)

On March 2nd 2014 Flexcoin was attacked and robbed of all coins in the hot wallet. The attacker made off with 896 BTC, dividing them into these two addresses:

1NDkevapt4SWYFEmquCDBSf7DLMTNVggdu

1QFcC5JitGwpFKqRDd9QNH3eGN56dCNgy6

As Flexcoin does not have the resources, assets, or otherwise to come back from this loss, we are closing our doors immediately.

Users who put their coins into cold storage will be contacted by Flexcoin and asked to verify their identity. Once identified, cold storage coins will be transferred out free of charge. Cold storage coins were held offline and not within reach of the attacker. All other users will be directed to Flexcoin's **"Terms of service"** located at "Flexcoin.com/118.html" a document which was agreed on, upon signing up with Flexcoin.

Flexcoin will attempt to work with law enforcement to trace the source of the hack.

Updates will be posted on **twitter** as soon as they become available.

Update (March 4 2014)

During the investigation into stolen funds we have determined that the extent of the theft was enabled by a flaw within the front-end.

The attacker logged into the flexcoin front end from IP address 207.12.89.117 under a newly created username and deposited to address **1DSD3B3uS2wGZjZAwa2dqQ7M9v7Ajw2iLy**

The coins were then left to sit until they had reached 6 confirmations.

The attacker then successfully exploited a flaw in the code which allows transfers between flexcoin users. By sending thousands of simultaneous requests, the attacker was able to "move" coins from one user account to another until the sending account was overdrawn, before balances were updated.

This was then repeated through multiple accounts, snowballing the amount, until the attacker withdrew the coins. (**Here** and **Here**)

Flexcoin has made every attempt to keep our servers as secure as possible, including regular testing. In our ~3 years of existence we have successfully repelled thousands of attacks. But in the end, this was simply not enough.

Having this be the demise of our small company, after the endless hours of work we've put in, was never our intent. We've failed our customers, our business, and ultimatley the Bitcoin community.

Please direct any and all questions to admin(at)flexcoin(dot)com and we will reply to you as soon as possible.

flexcoin the bitcoin bank

FAQ Bitcoin Bank Register Activate Login

Flexcoin is shutting down. (March 3 2014)

On March 2nd 2014 Flexcoin was attacked and robbed of all coins in the hot wallet. The attacker made off with 896 BTC, dividing them into these two addresses:

1NDkevapt4SWYFEmquCDBSf7DLMTNVggdu

1QFcC5JitGwpFKqRDd9QNH3eGN56dCNgy6

Update (March 4 2014)

During the investigation into stolen funds we have determined that the extent of the theft was enabled by a flaw within the front-end.

The attacker logged into the flexcoin front end from IP address 207.12.89.117 under a newly created username and deposited to address 1DSD3B3uS2wGZjZAwa2dqQ7M9v7Ajw2iLy

"By sending thousands of simultaneous requests, the attacker was able to 'move' coins from one user account to another until the sending account was overdrawn, before balances were updated."

What's Happening?

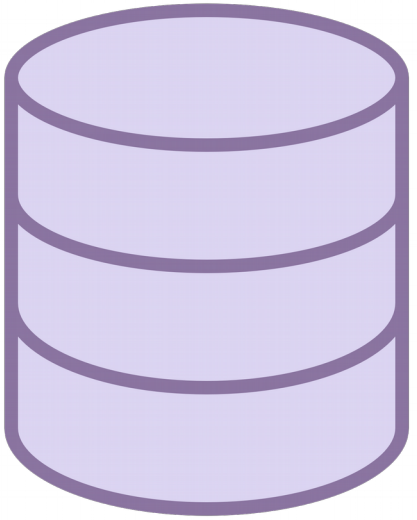
- Race condition – application exhibits behavior under concurrent execution **not possible under serial execution**
- Can we exploit these behaviors?
- Yes! We call this exploitation of non-serializable API behavior an ACIDRain attack

Overview

- Problem setup
- New method for detecting latent potential for non-serializable behavior
- Evaluation – analysis of 12 eCommerce platforms

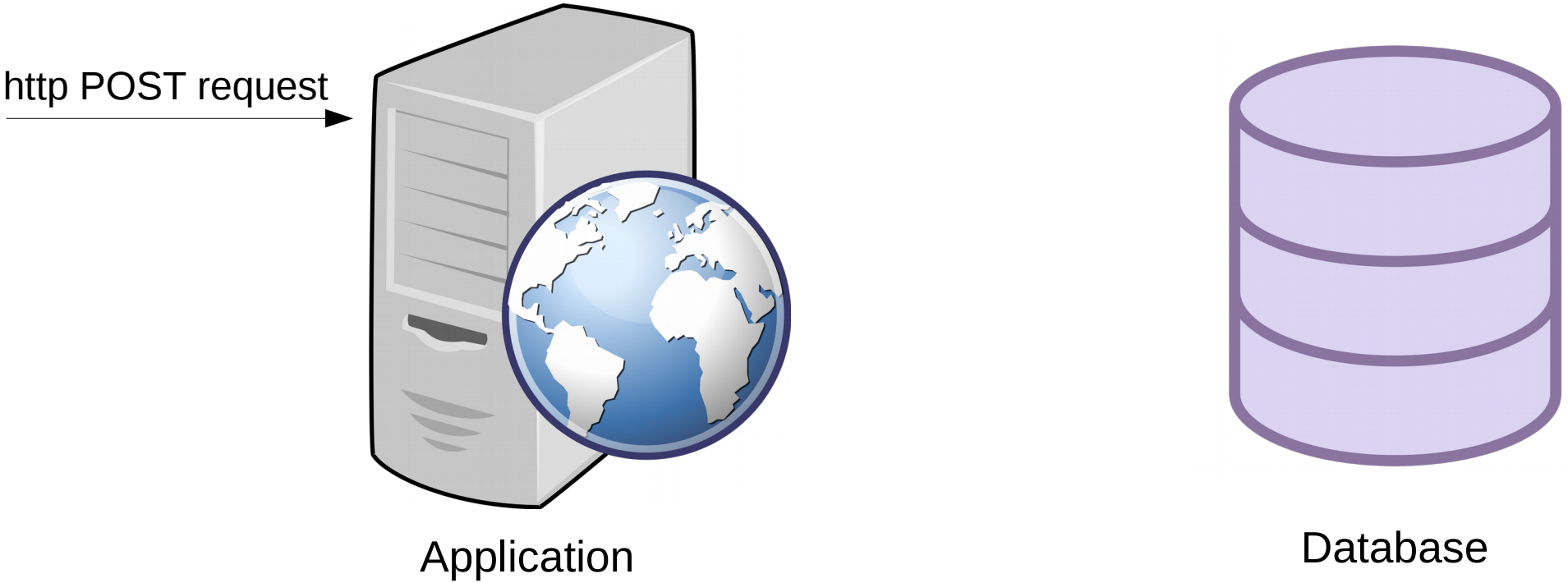
Problem Setup: Attacking Websites

Application Server

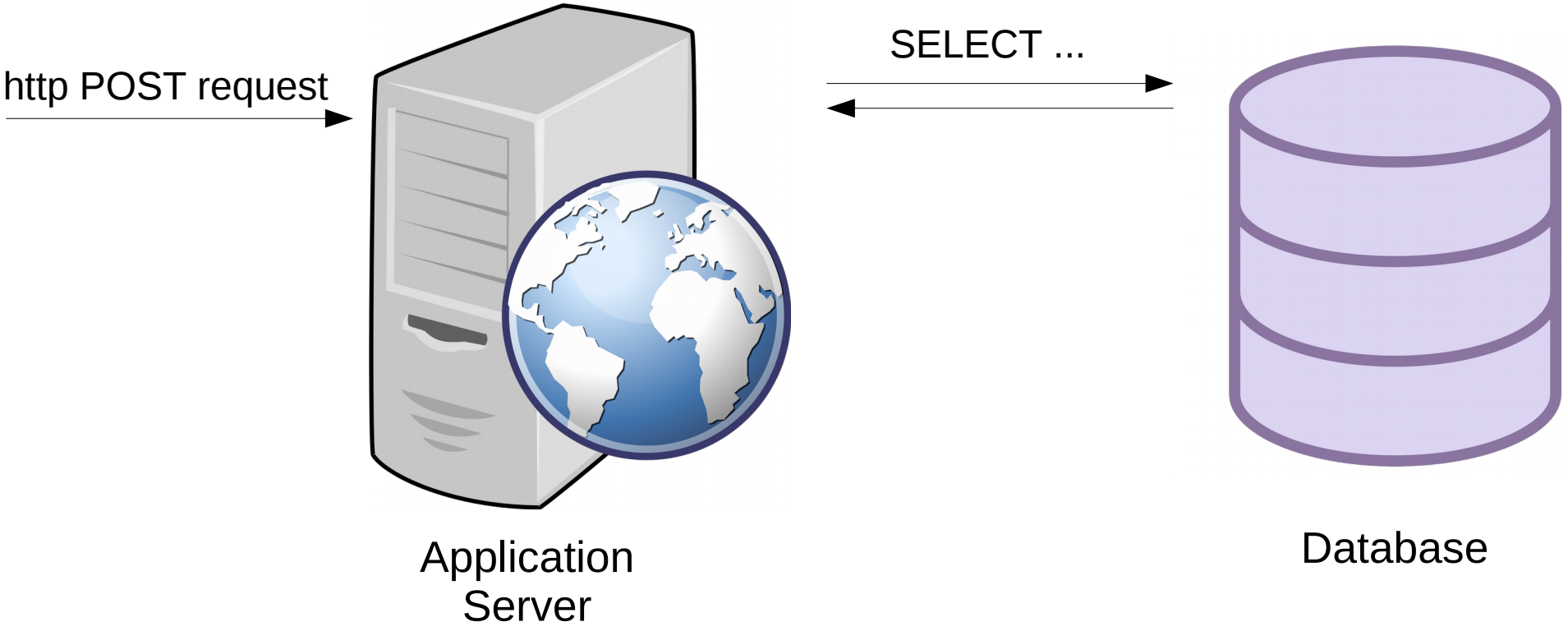


Database

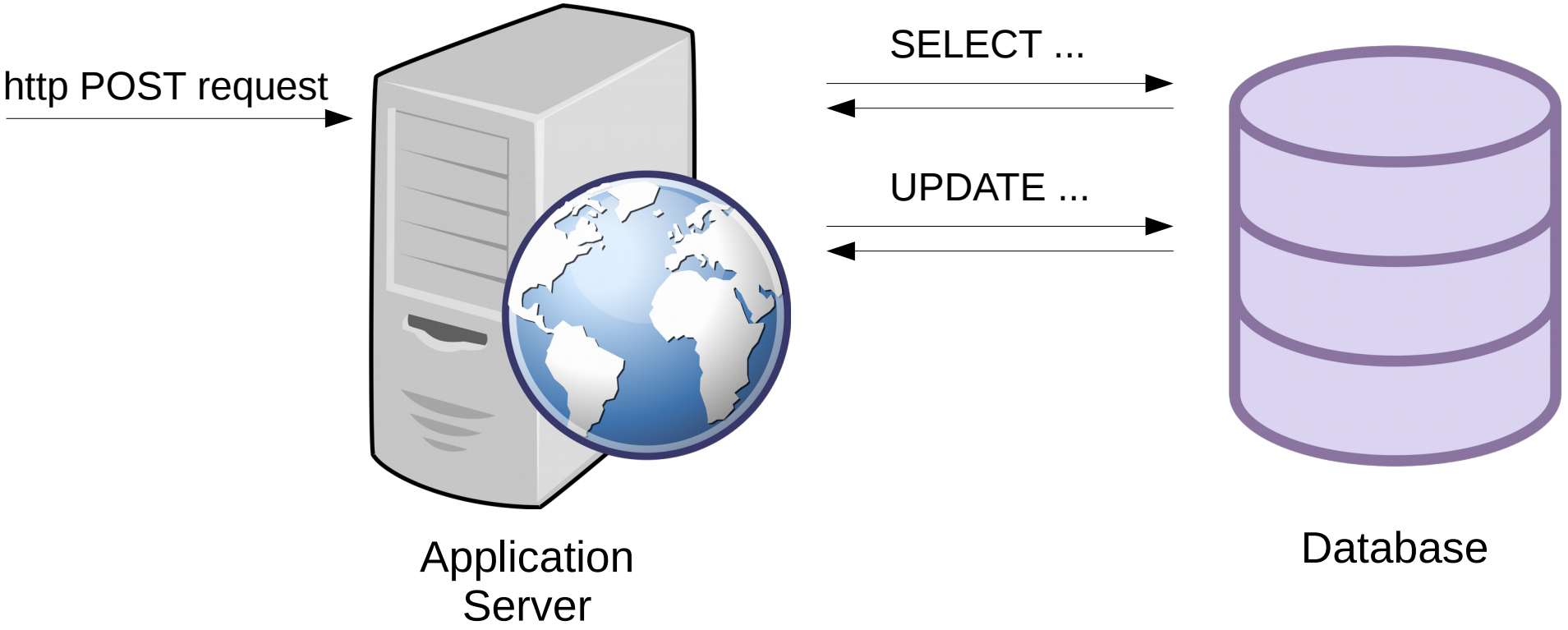
Problem Setup: Attacking Websites



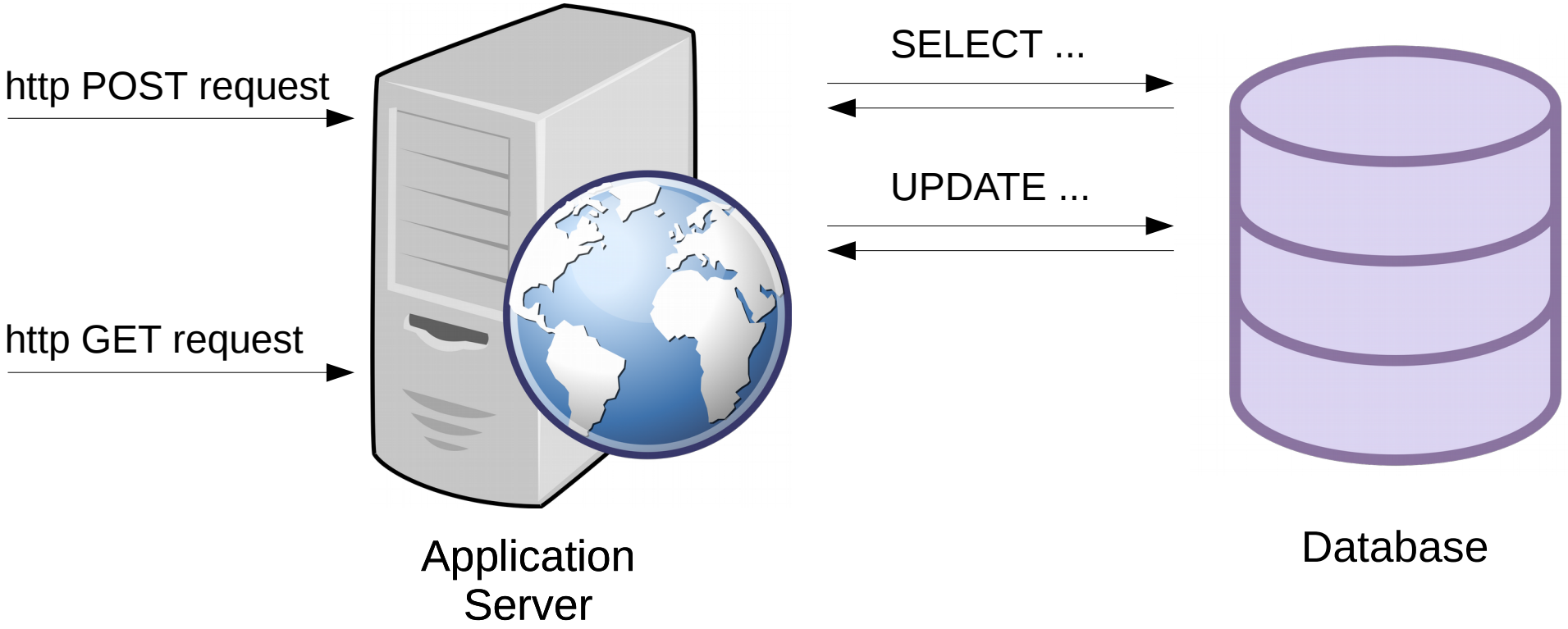
Problem Setup: Attacking Websites



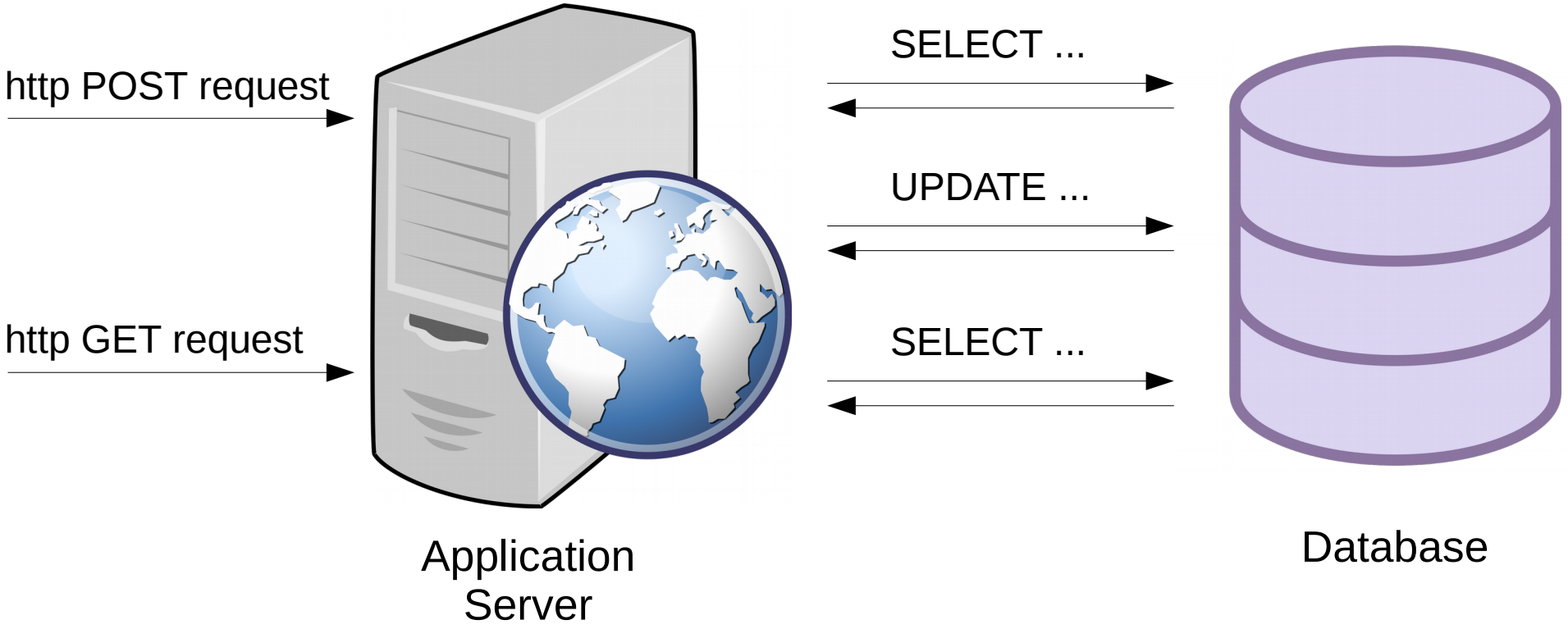
Problem Setup: Attacking Websites



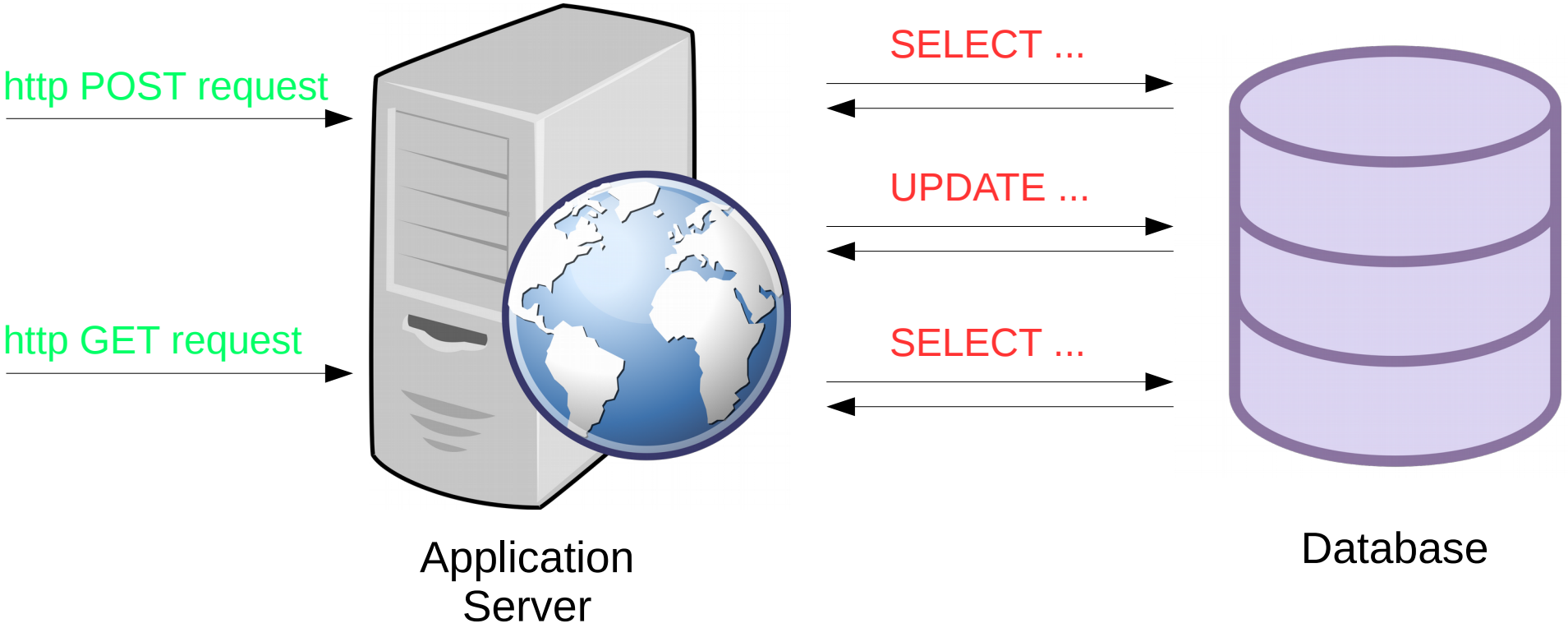
Problem Setup: Attacking Websites



Problem Setup: Attacking Websites



Problem Setup: Attacking Websites



Serializability of API Requests

Serializability of Database Transactions

Non-Transactional Implementation

```
def checkVoucher(code):
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
```

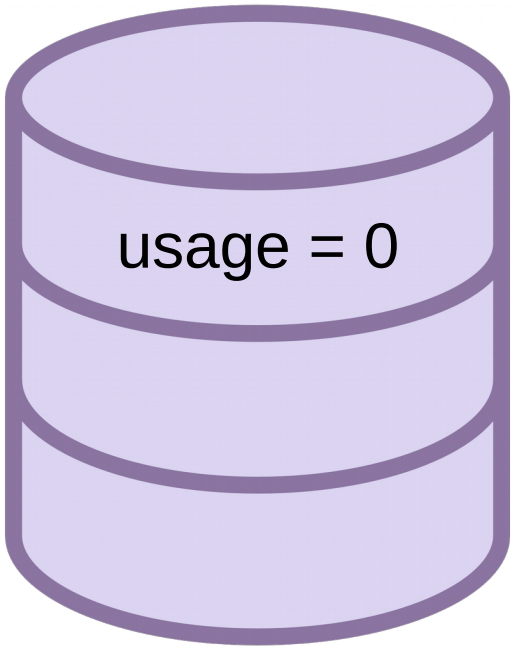
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY

Non-Transactional Implementation

```
def checkVoucher(code):
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
```

SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY

Application Server



Database

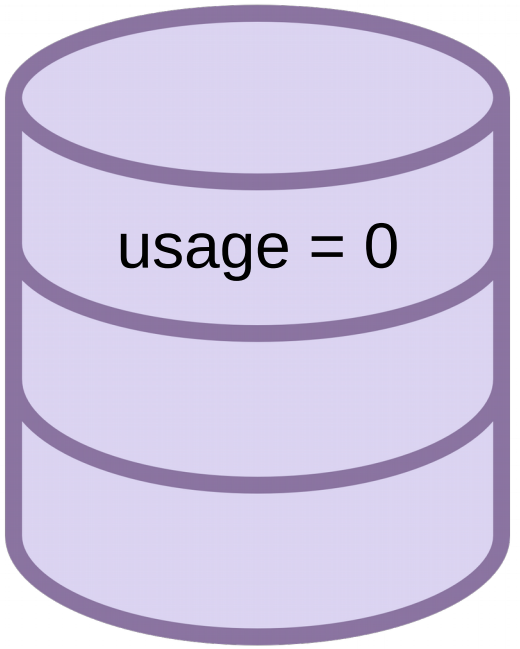
Non-Transactional Implementation

```
def checkVoucher(code):
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
```

SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY

Alice Checkout

Application Server

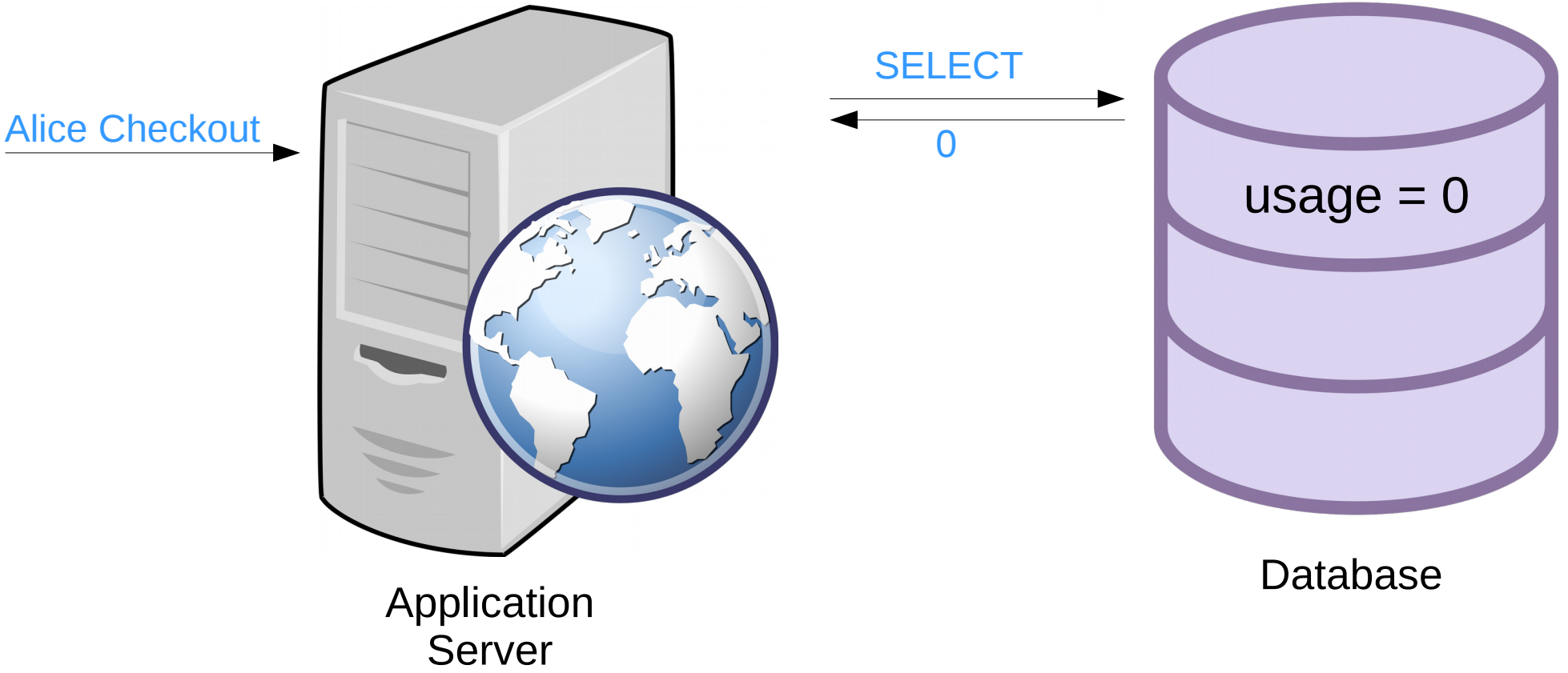


Database

Non-Transactional Implementation

```
def checkVoucher(code):
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
```

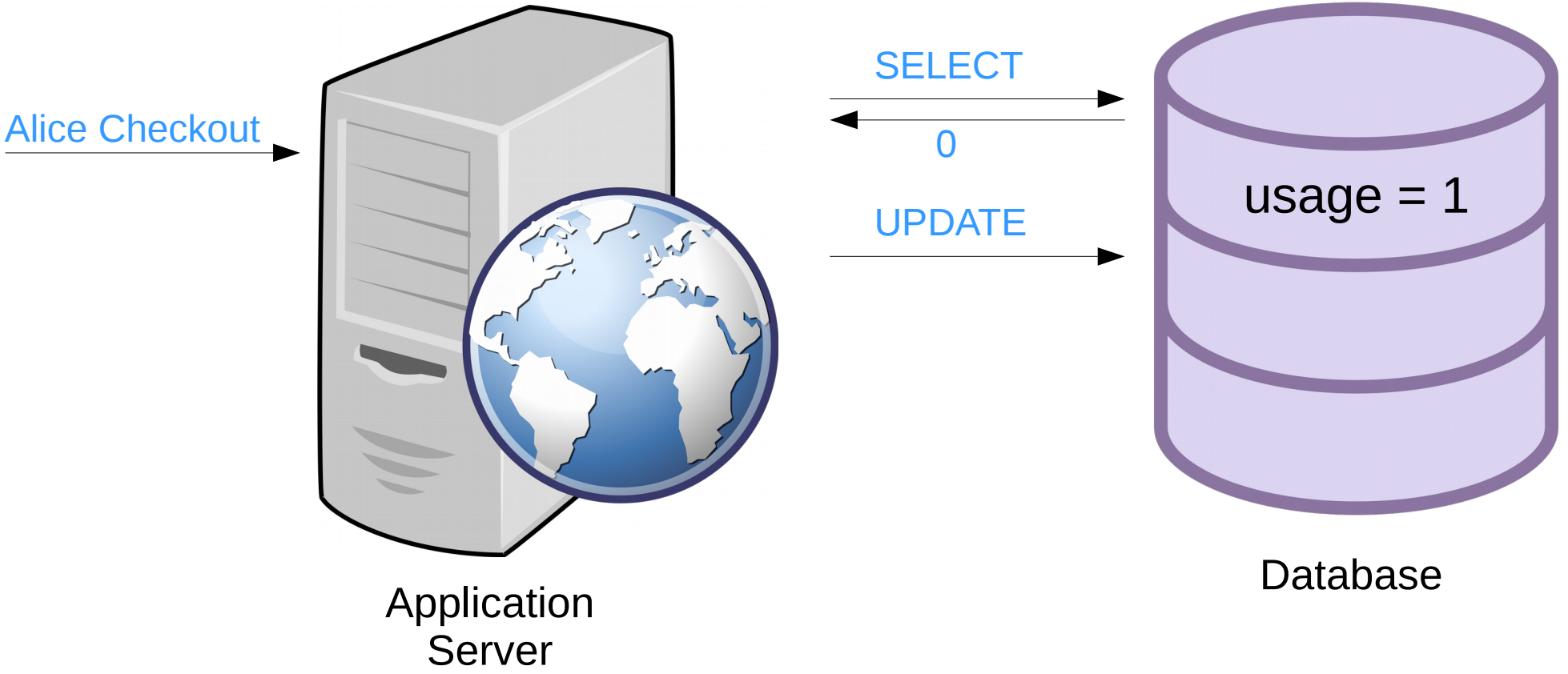
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY



Non-Transactional Implementation

```
def checkVoucher(code):
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
```

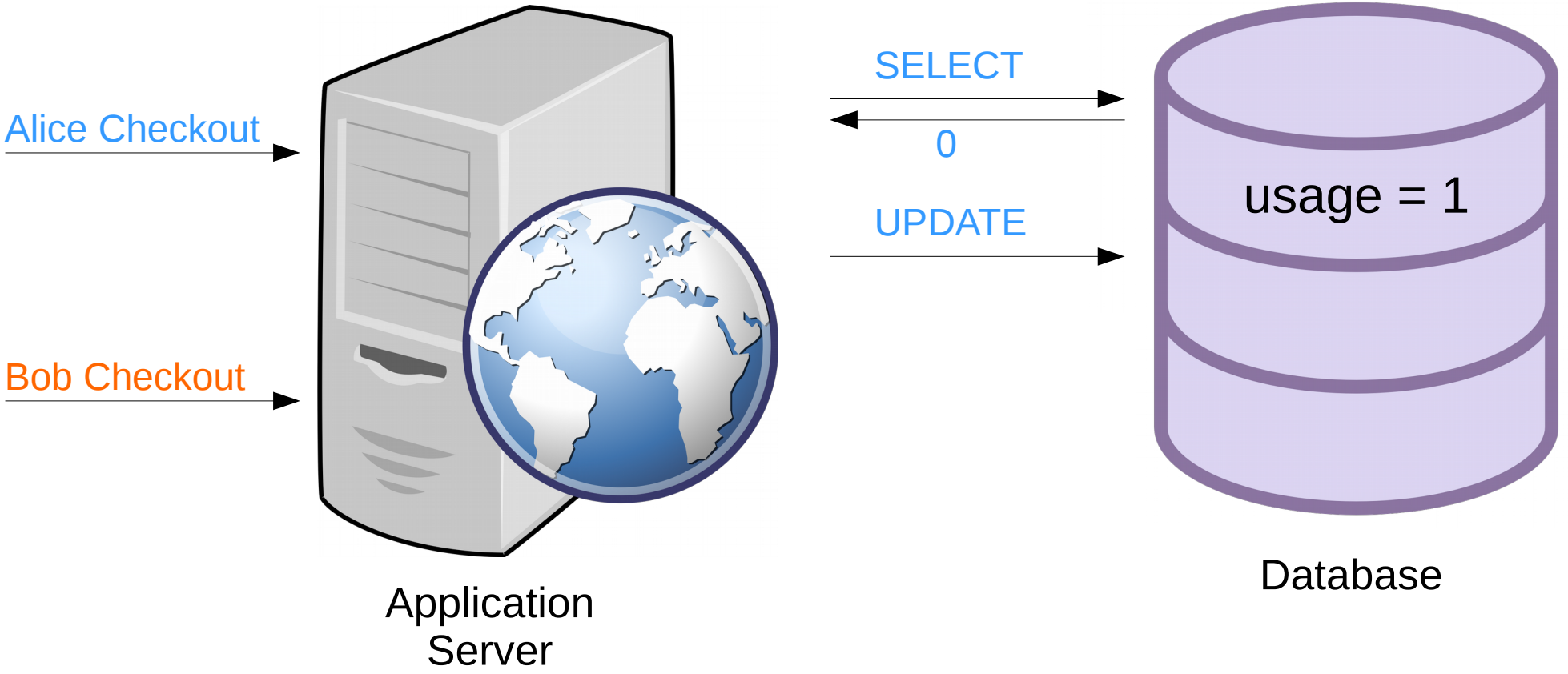
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY



Non-Transactional Implementation

```
def checkVoucher(code):
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
```

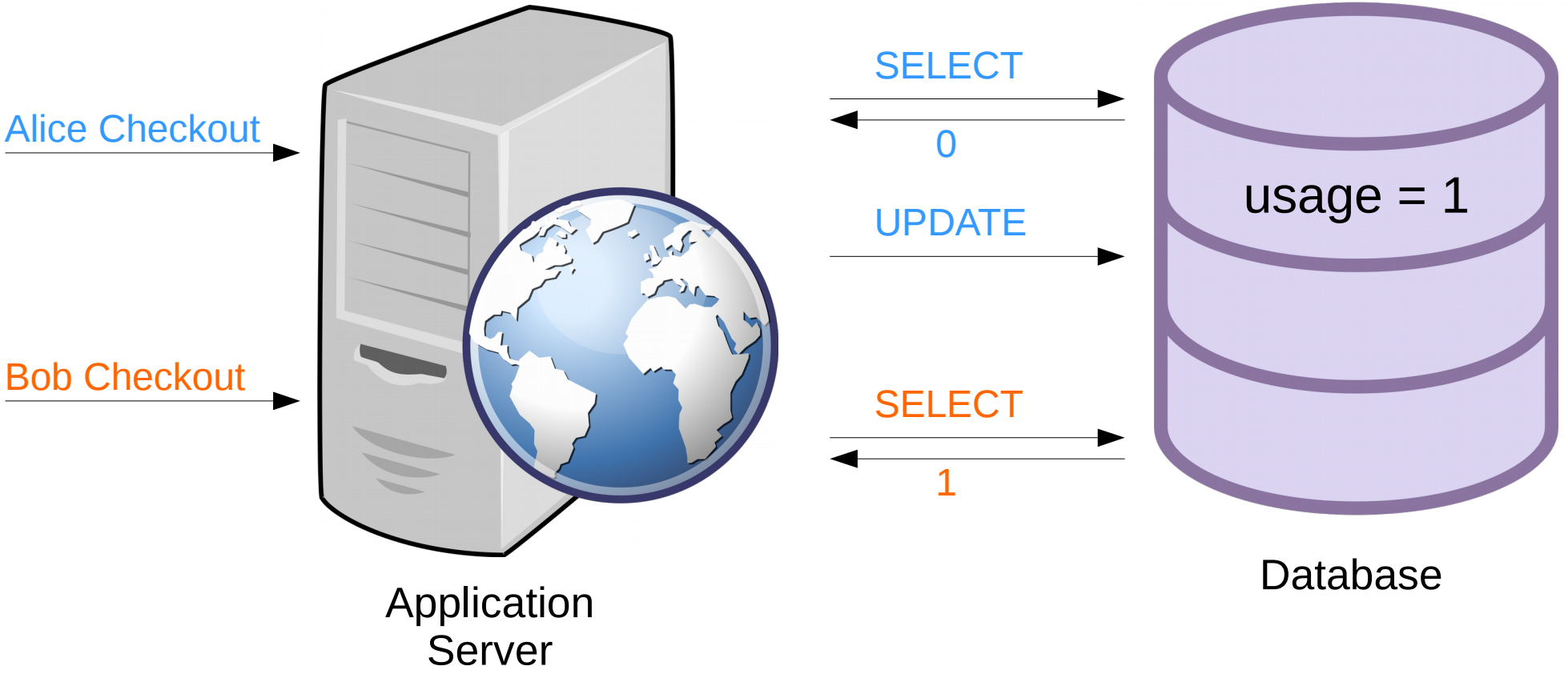
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY



Non-Transactional Implementation

```
def checkVoucher(code):
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
```

SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY

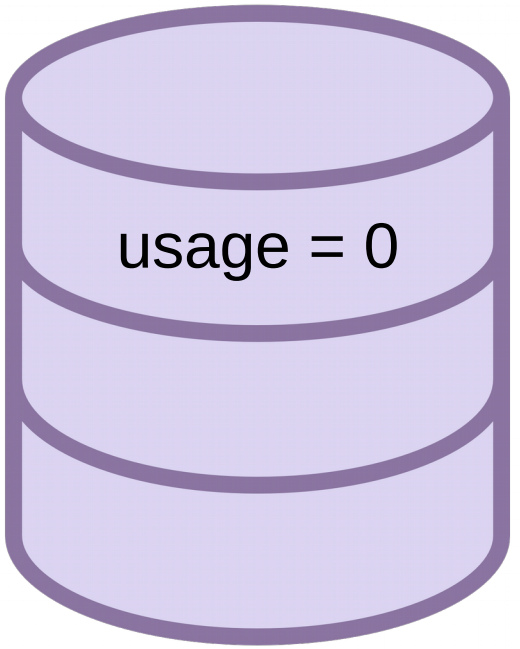


Non-Transactional Implementation

```
def checkVoucher(code):
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
```

SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY

Application Server



Database

Non-Transactional Implementation

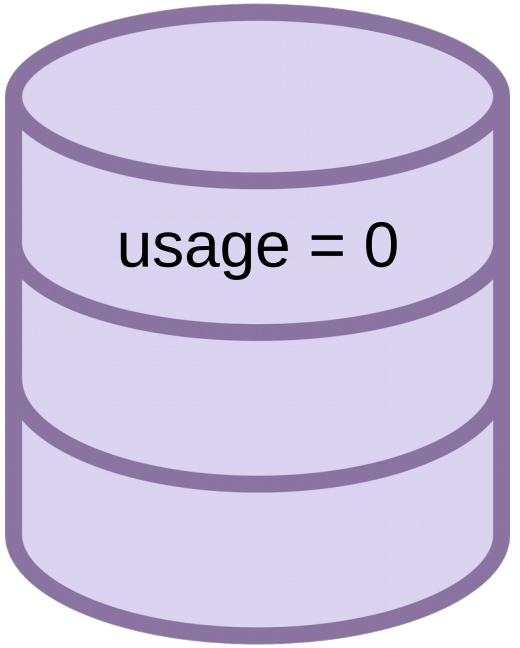
```
def checkVoucher(code):
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
```

SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY

Alice Checkout

Bob Checkout

Application Server

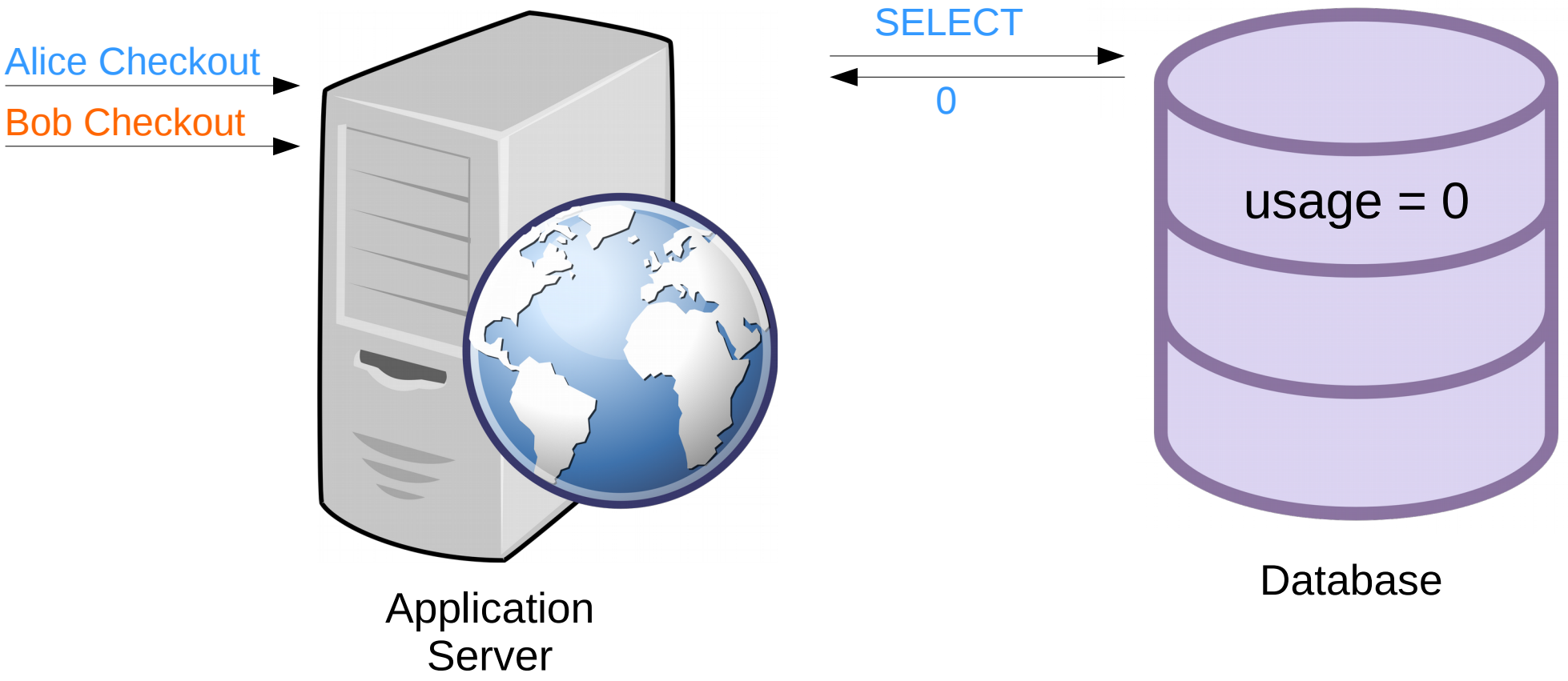


Database

Non-Transactional Implementation

```
def checkVoucher(code):
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
```

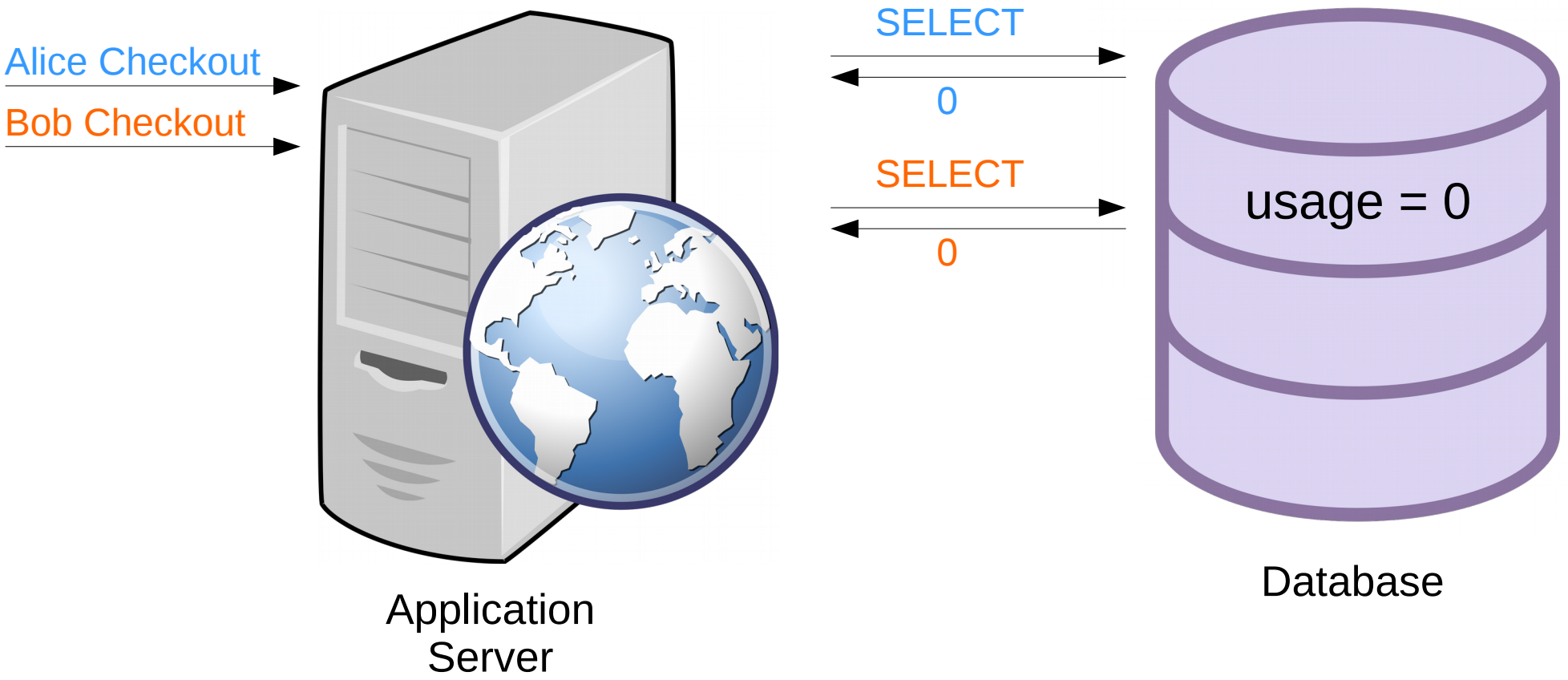
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY



Non-Transactional Implementation

```
def checkVoucher(code):
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
```

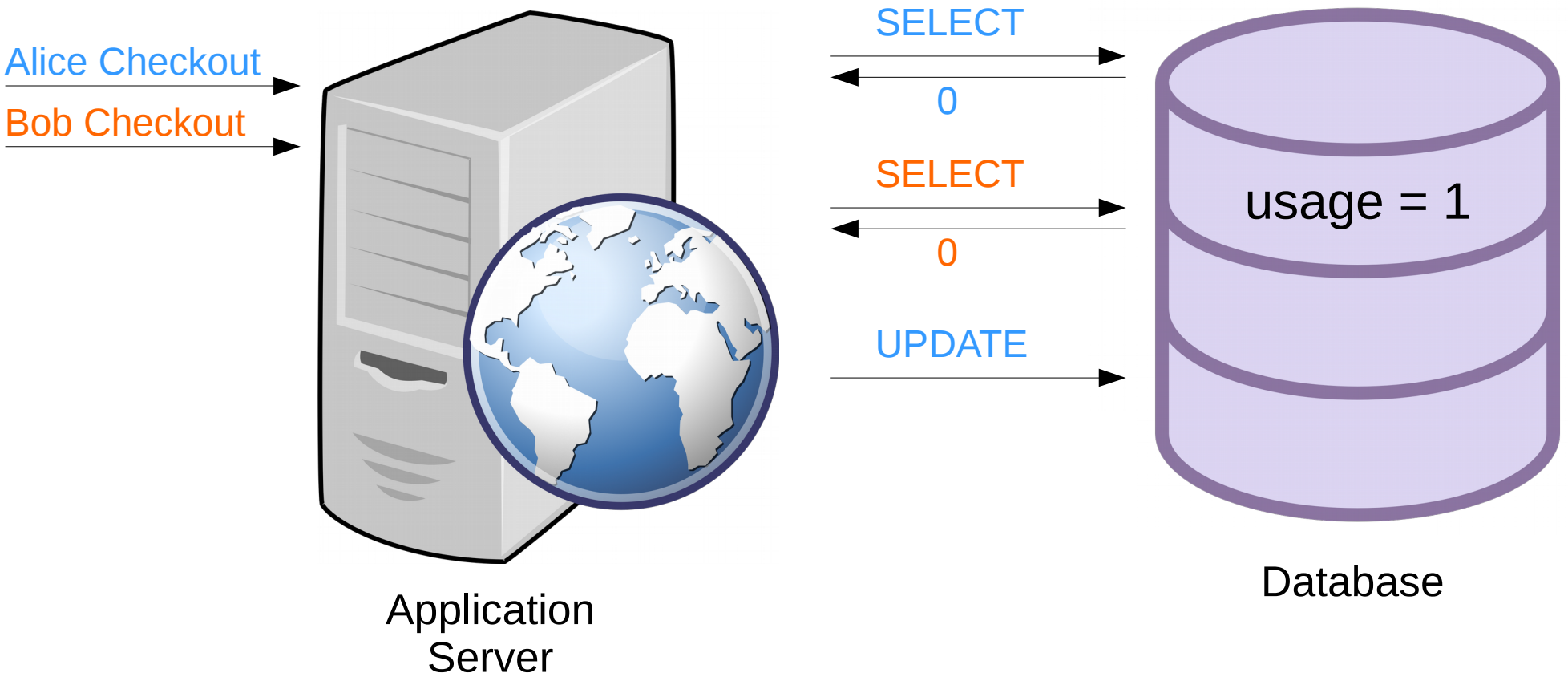
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY



Non-Transactional Implementation

```
def checkVoucher(code):
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
```

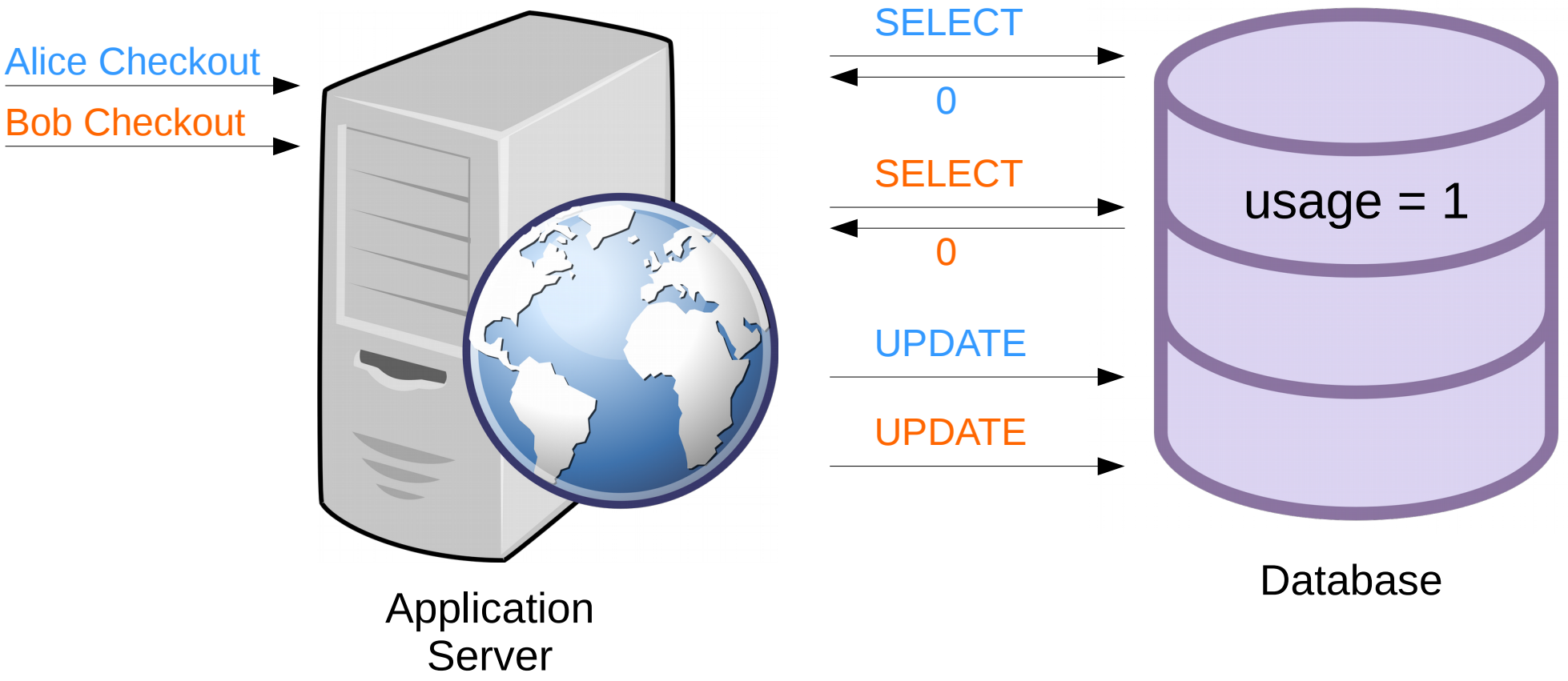
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY



Non-Transactional Implementation

```
def checkVoucher(code):
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
```

SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY



Transactional Implementation

```
def checkVoucher(code):
    beginTxn()
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
    commit()
```

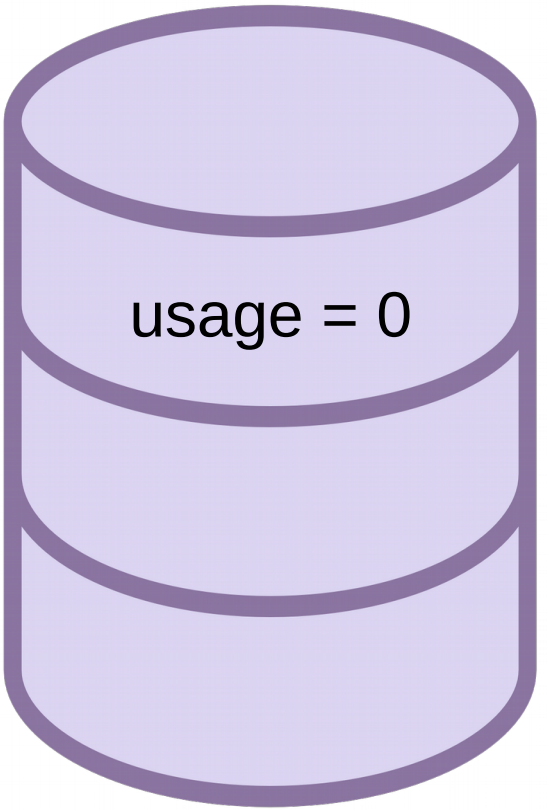
```
BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT
```

Transactional Implementation

```
def checkVoucher(code):
    beginTxn()
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
    commit()
```

```
BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT
```

Application Server



Database

Transactional Implementation

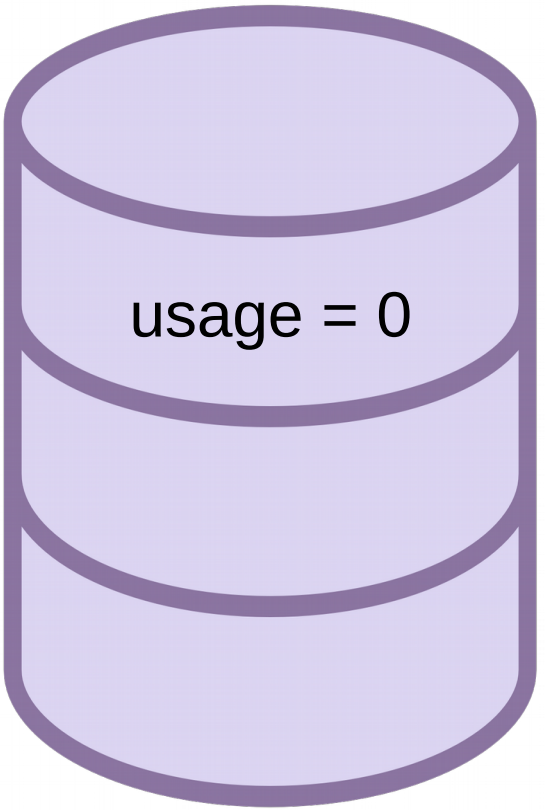
```
def checkVoucher(code):
    beginTxn()
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
    commit()
```

```
BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT
```

Alice Checkout

Bob Checkout

Application Server

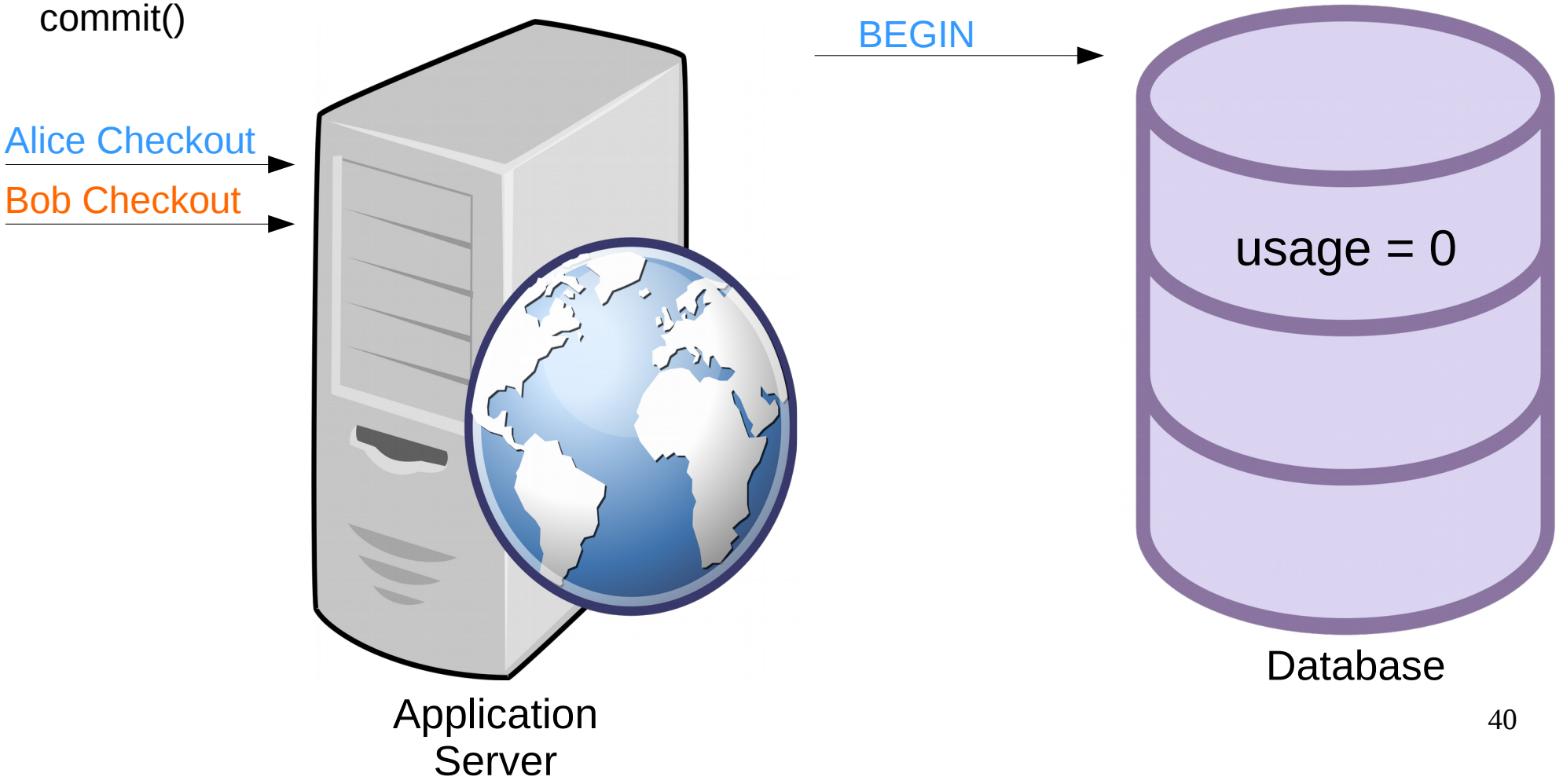


Database

Transactional Implementation

```
def checkVoucher(code):
    beginTxn()
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
    commit()
```

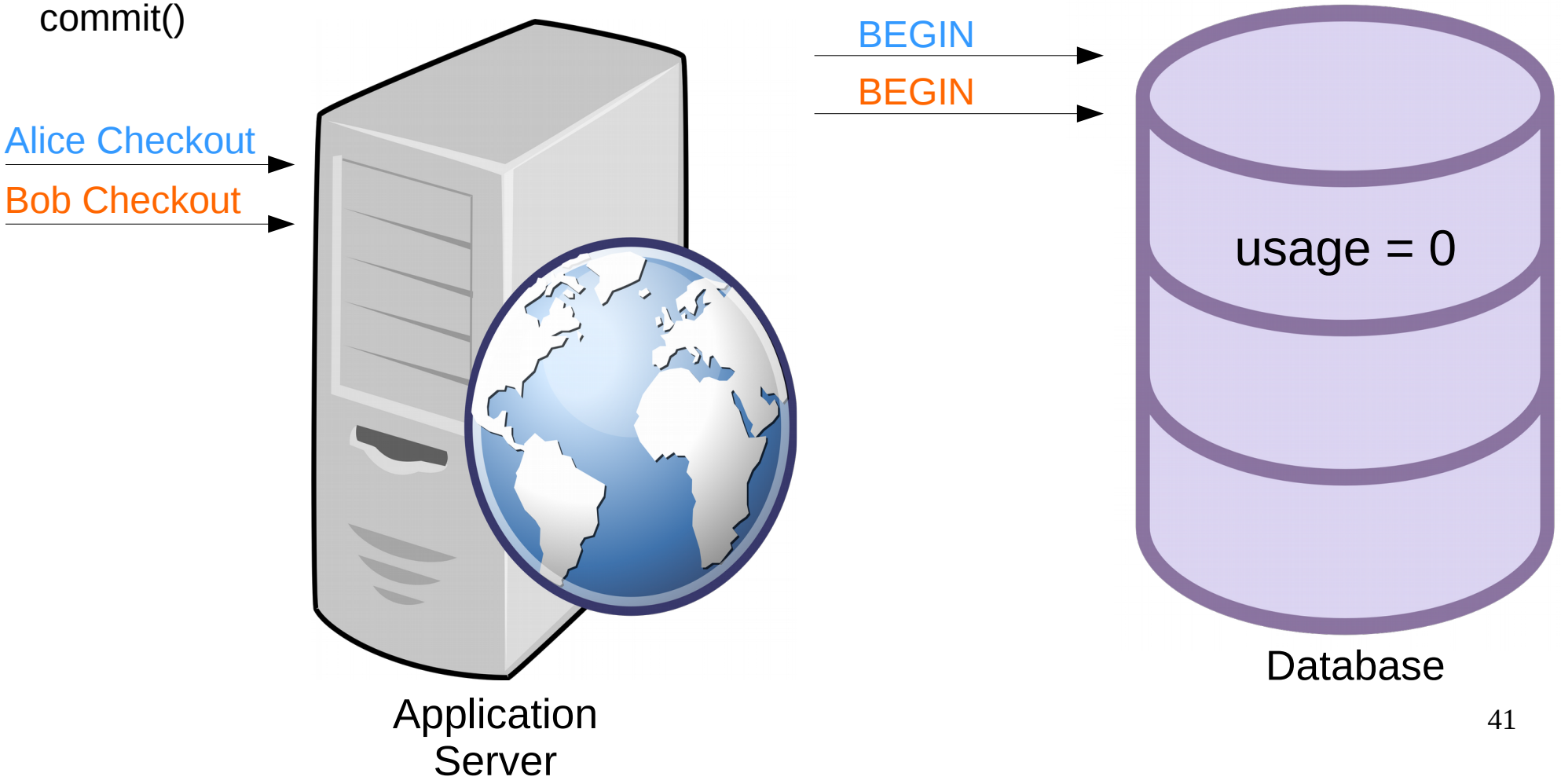
```
BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT
```



Transactional Implementation

```
def checkVoucher(code):
    beginTxn()
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
    commit()
```

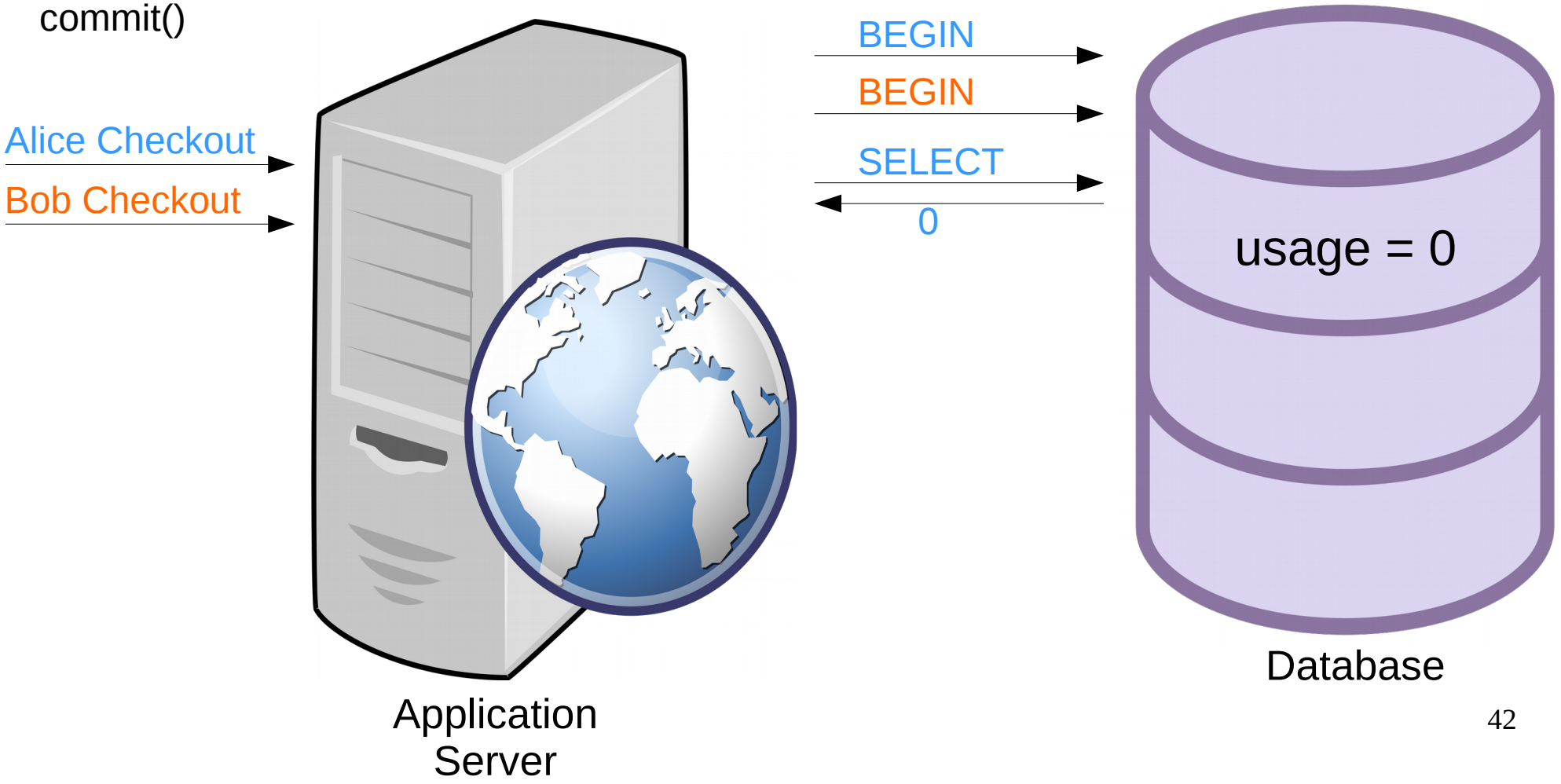
```
BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT
```



Transactional Implementation

```
def checkVoucher(code):
    beginTxn()
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
    commit()
```

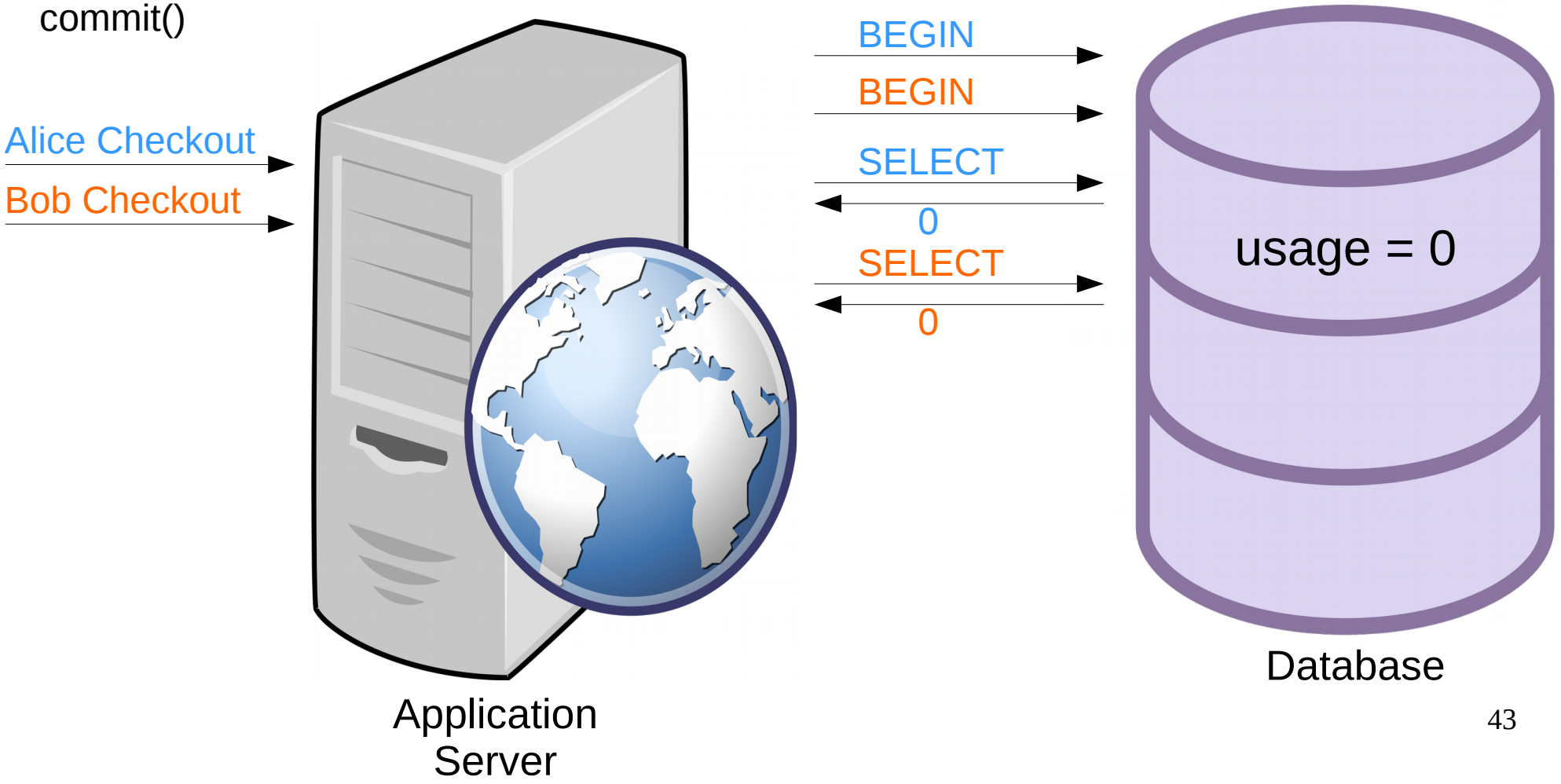
```
BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT
```



Transactional Implementation

```
def checkVoucher(code):
    beginTxn()
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
    commit()
```

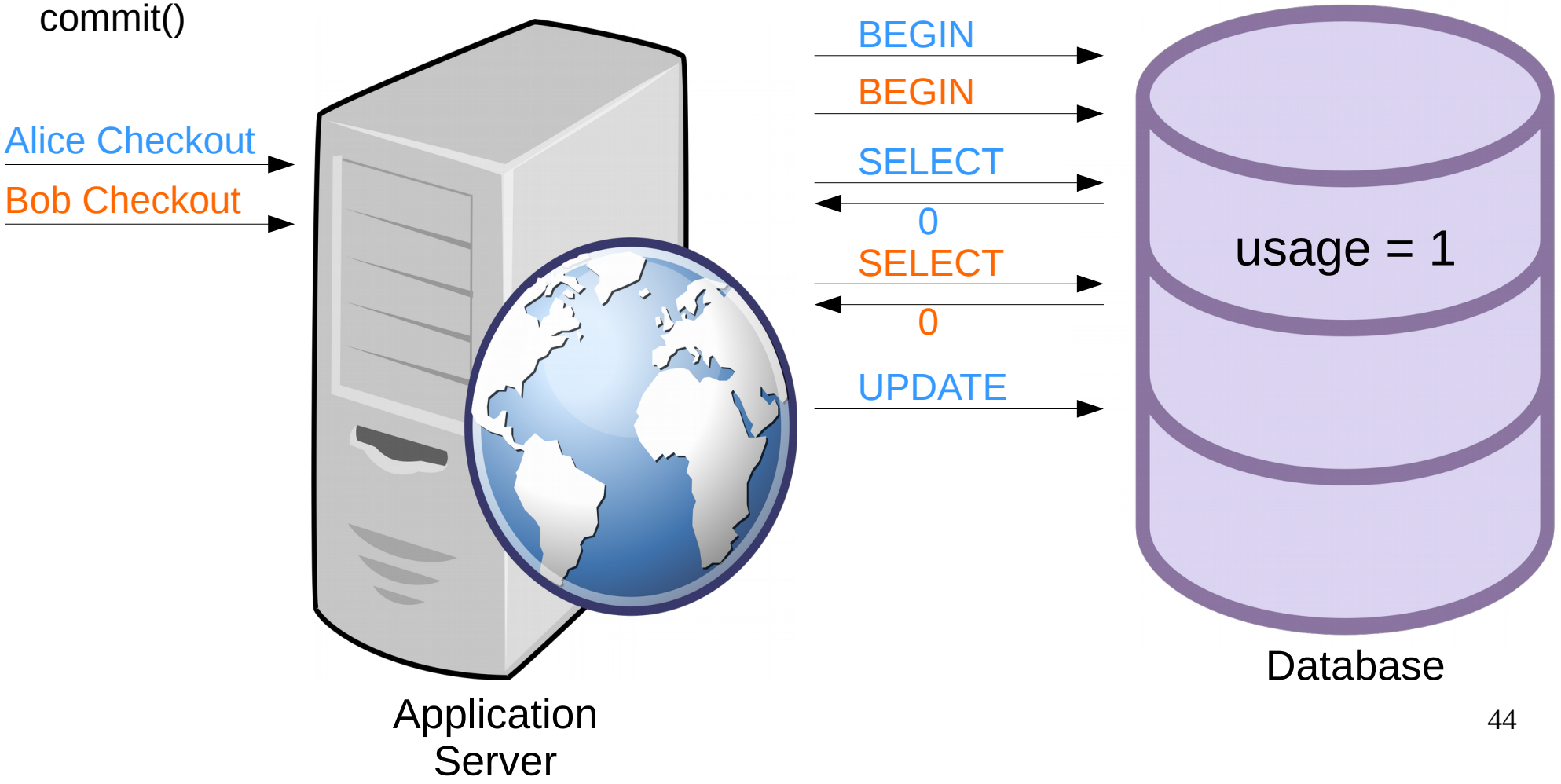
```
BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT
```



Transactional Implementation

```
def checkVoucher(code):
    beginTxn()
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
    commit()
```

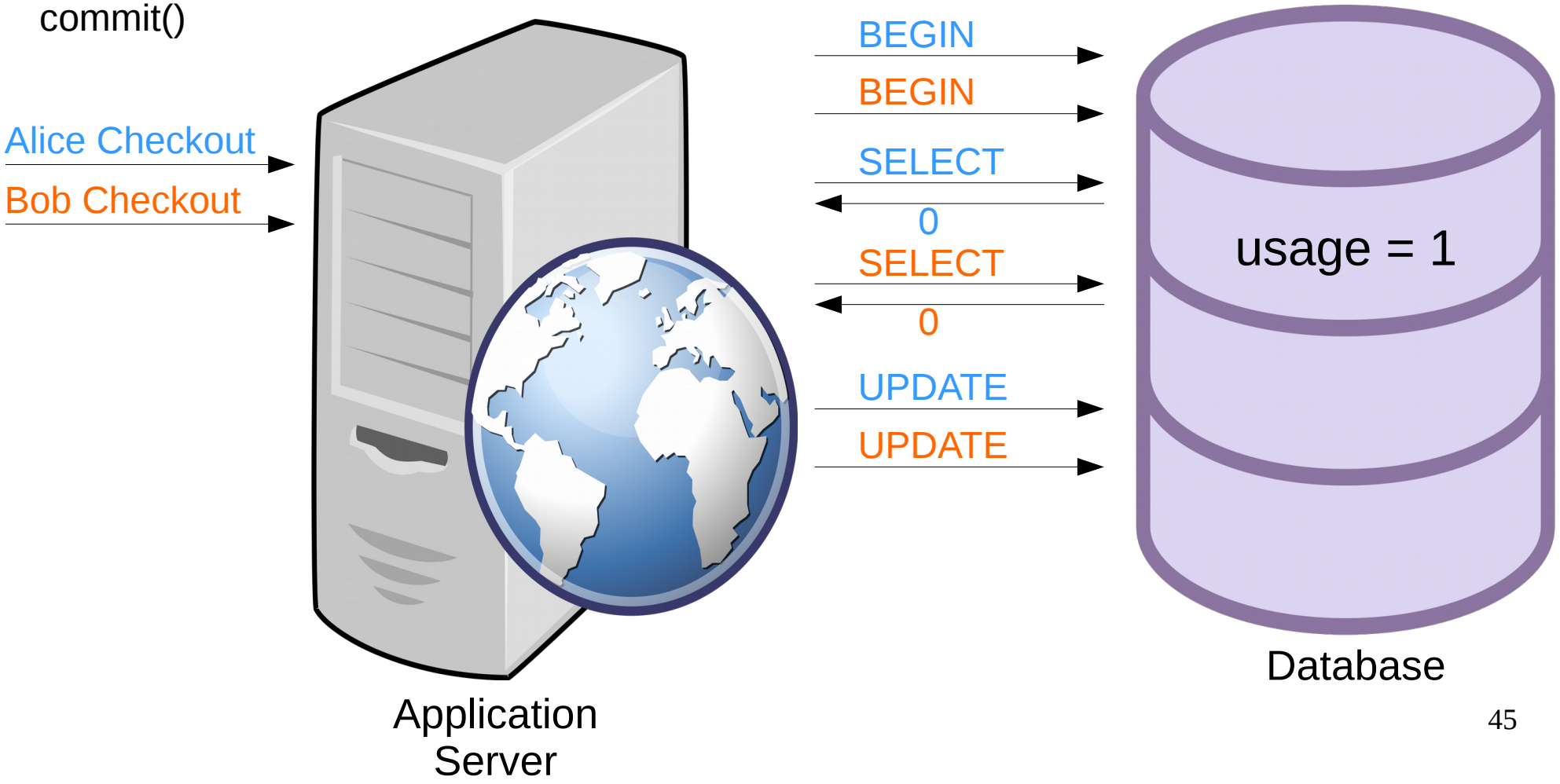
```
BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT
```



Transactional Implementation

```
def checkVoucher(code):
    beginTxn()
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
    commit()
```

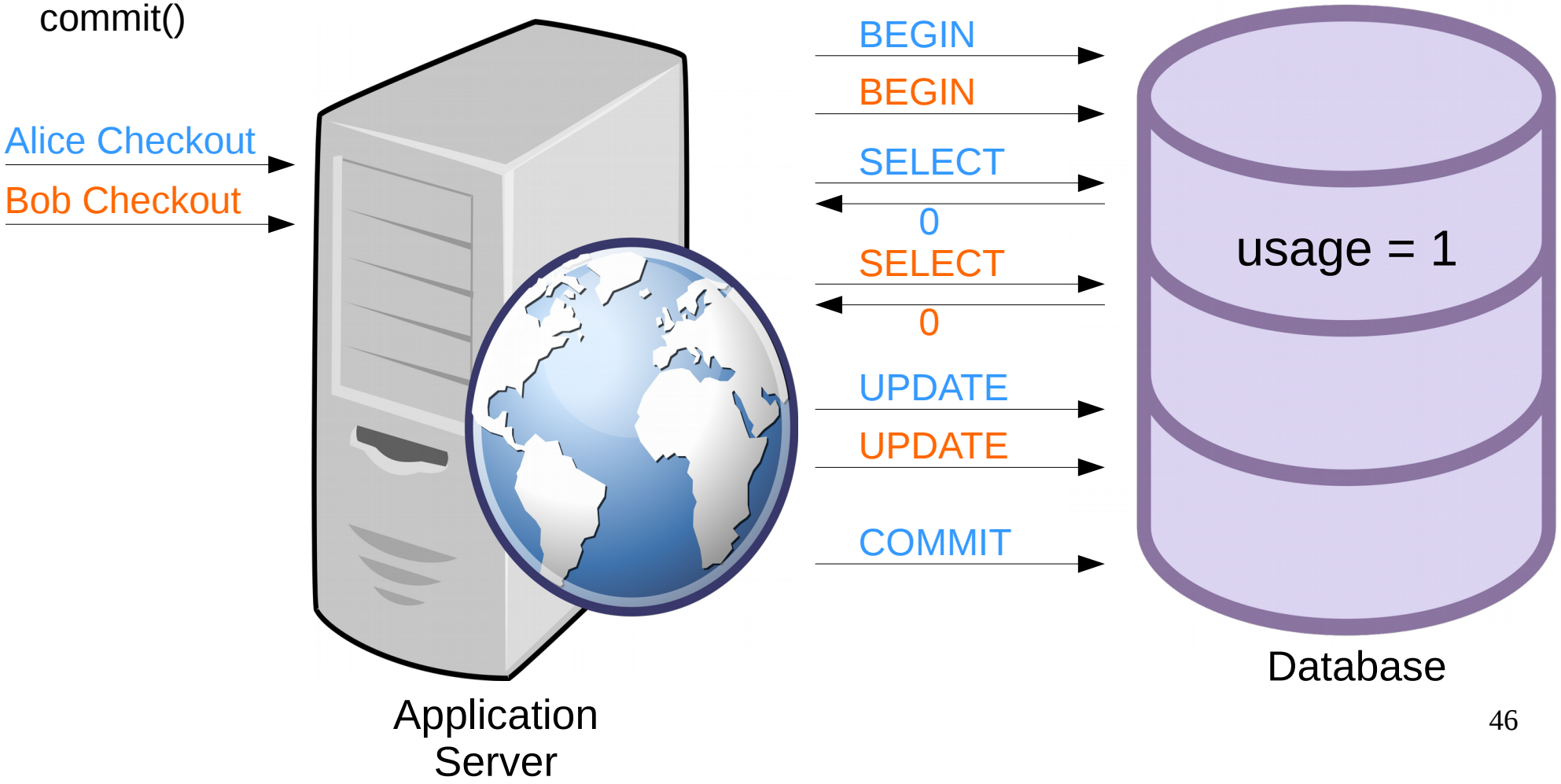
```
BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT
```



Transactional Implementation

```
def checkVoucher(code):
    beginTxn()
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
    commit()
```

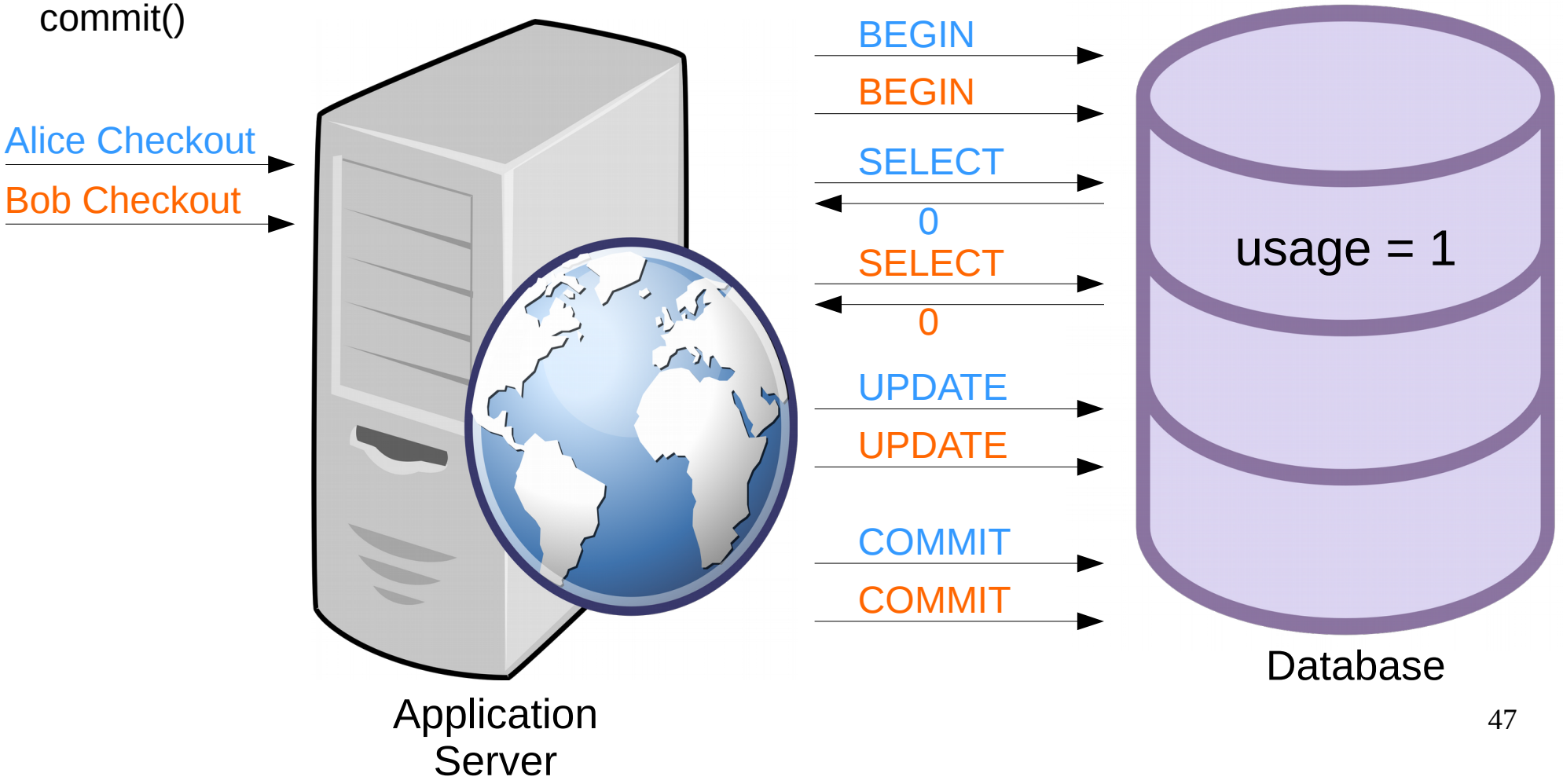
```
BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT
```



Transactional Implementation

```
def checkVoucher(code):
    beginTxn()
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
    commit()
```

```
BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT
```

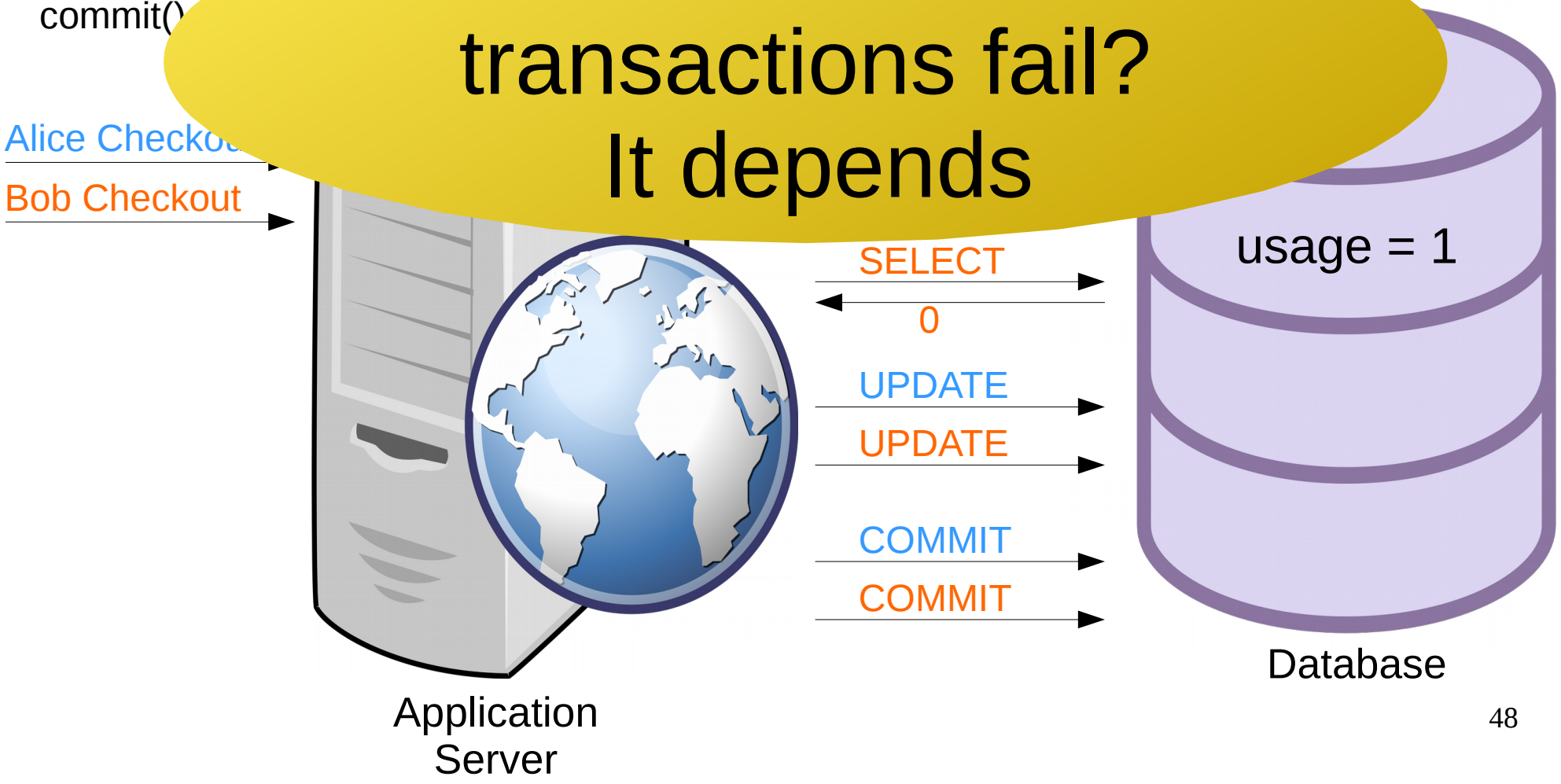


Transactional Implementation

```
def checkVoucher(code):
    beginTxn()
    usage = readUsage(code)
    if (usage == 0):
        markUsed(
    commit()
```

```
BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
```

Will one of the transactions fail?
It depends



Many Databases Allow This Anomaly

Database	Default Isolation	Maximum Isolation
Actian Ingres 10.0/10S	✓	✓
Aerospike	✗	✗
Akiban Persistit	✓	✓
Clustrix CLX 4100	✓	✓
Greenplum 4.1	✗	✓
IBM DB2 10 for z/OS	✓	✓
MySQL 5.6	✗	✓
MemSQL 1b	✗	✗
MS SQL Server 2012	✗	✓
NuoDB	✓	✓
Oracle 11g	✗	✓
Oracle Berkeley DB	✓	✓
Oracle Berkeley DB JE	✓	✓
Postgres 9.2.2	✗	✓
SAP HANA	✗	✓
ScaleDB 1.02	✗	✗
VoltDB	✓	✓

✓ = prevents anomaly

✗ = exhibits anomaly

Two Sources of Vulnerabilities

- Databases providing weak isolation may exhibit non-serializable behavior

```
def checkVoucher(code):
    beginTxn()
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
    commit()
```

- Programmers may code transactions incorrectly

```
def checkVoucher(code):
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
```

Overview

- Problem setup
- New method for detecting latent potential for non-serializable behavior
- Evaluation – analysis of 12 eCommerce platforms

Analysis Challenges

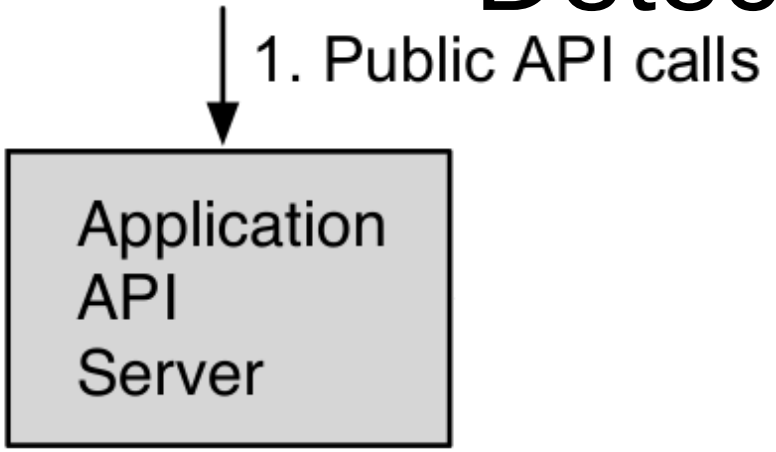
- Want to analyze web applications written in multiple languages and frameworks


- Anomalies only occur under concurrent execution, but website activity is often serial

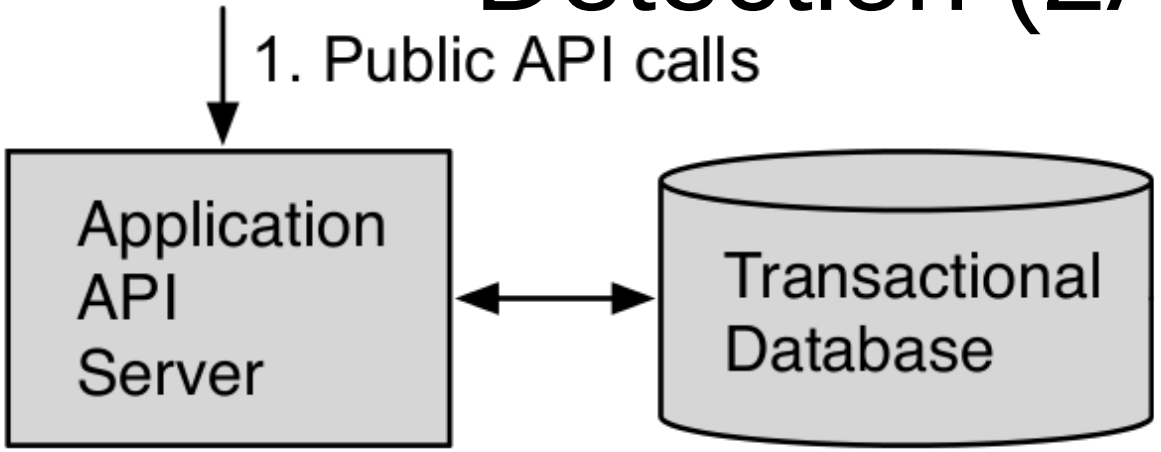
Approach: Abstract Anomaly Detection (2AD)

1. Collect (possibly serial) logs from database
2. Build compact representation of history (abstract history graph)
3. Search abstract history for cycles to generate possible anomalous API calls

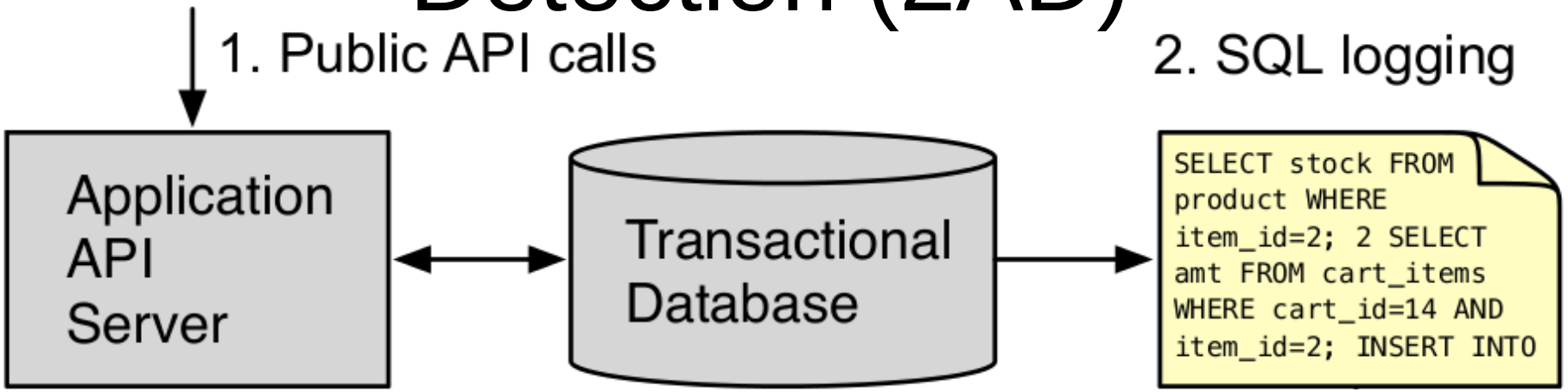
Approach: Abstract Anomaly Detection (2AD)



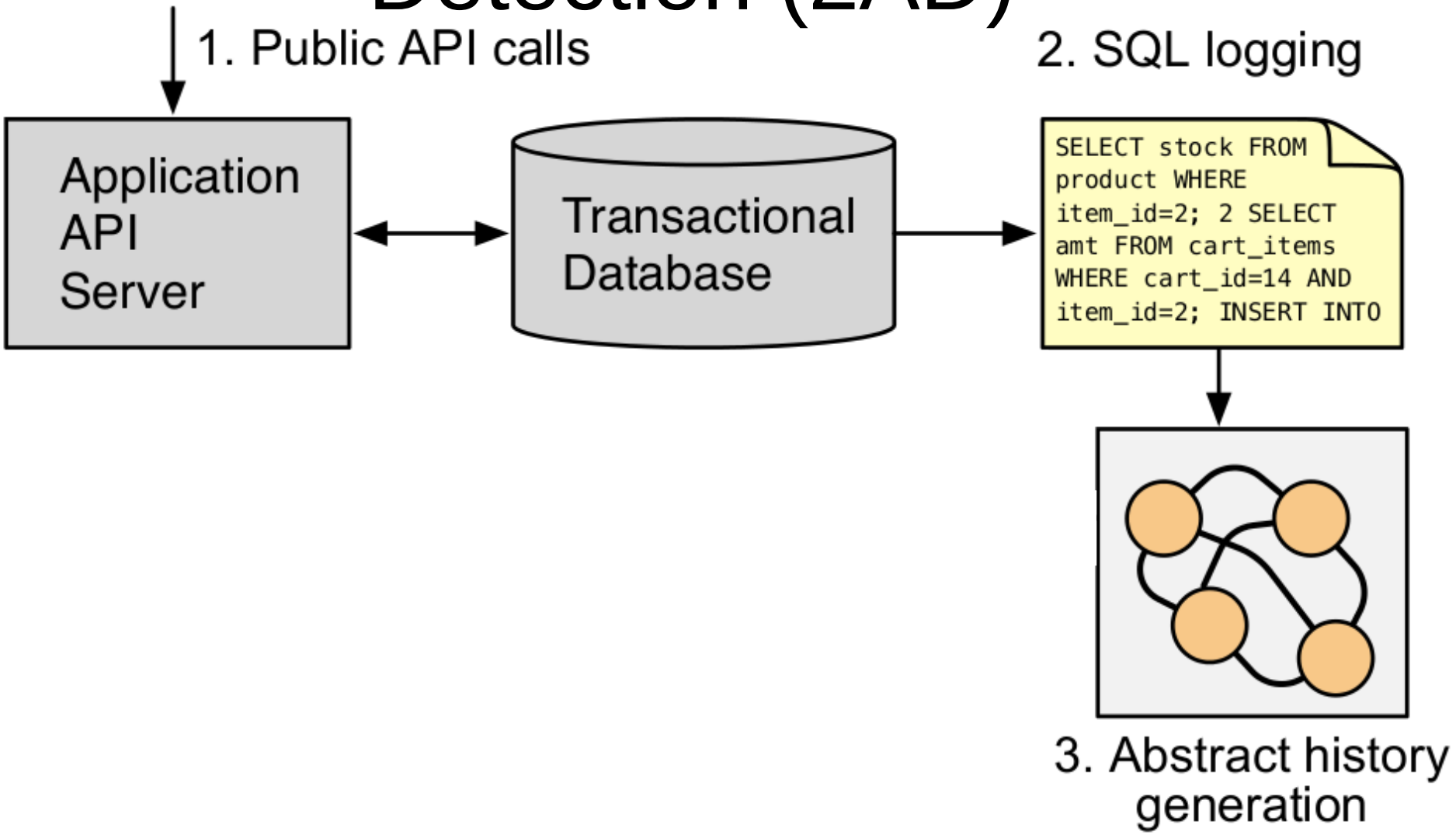
Approach: Abstract Anomaly Detection (2AD)



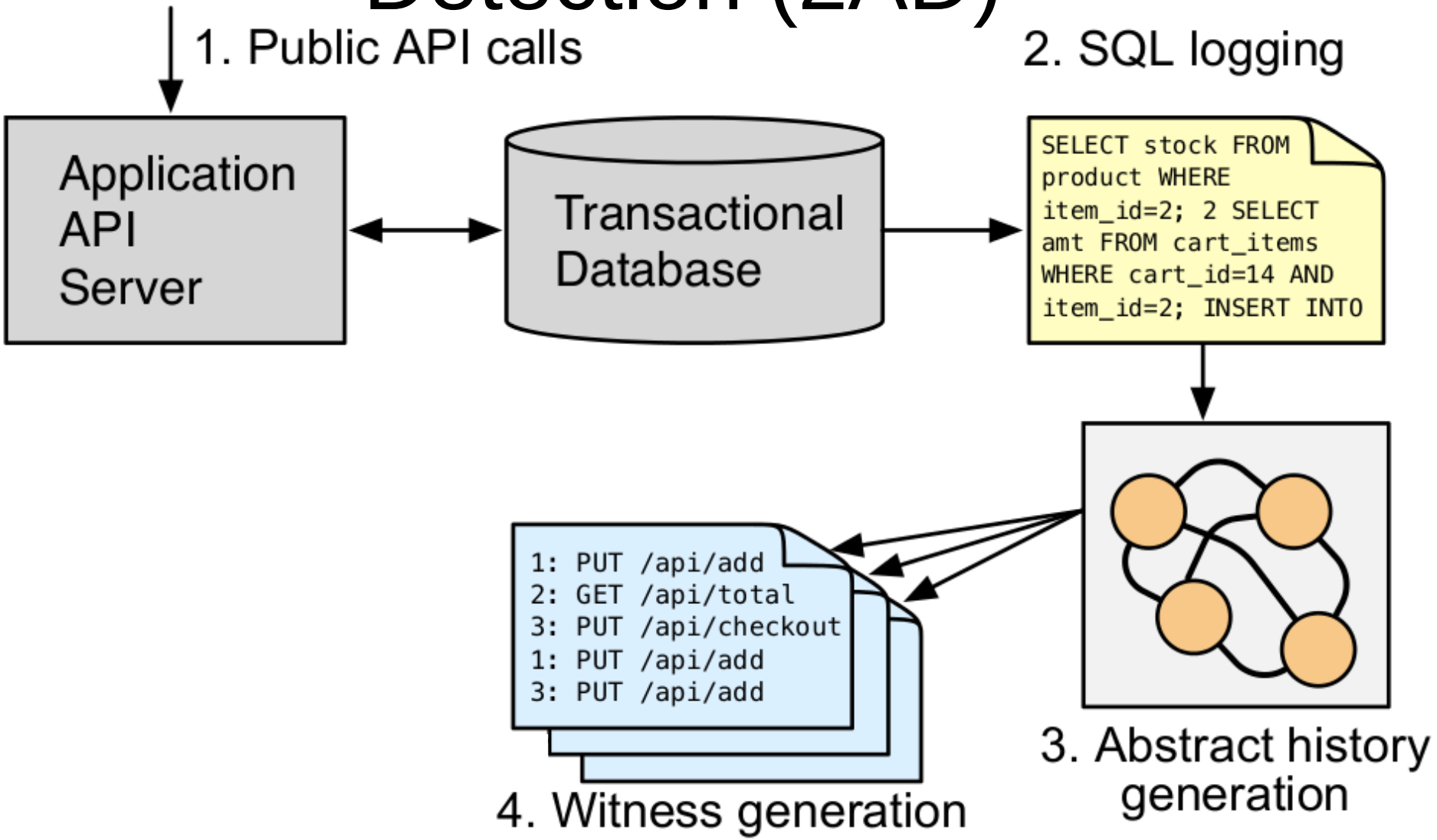
Approach: Abstract Anomaly Detection (2AD)



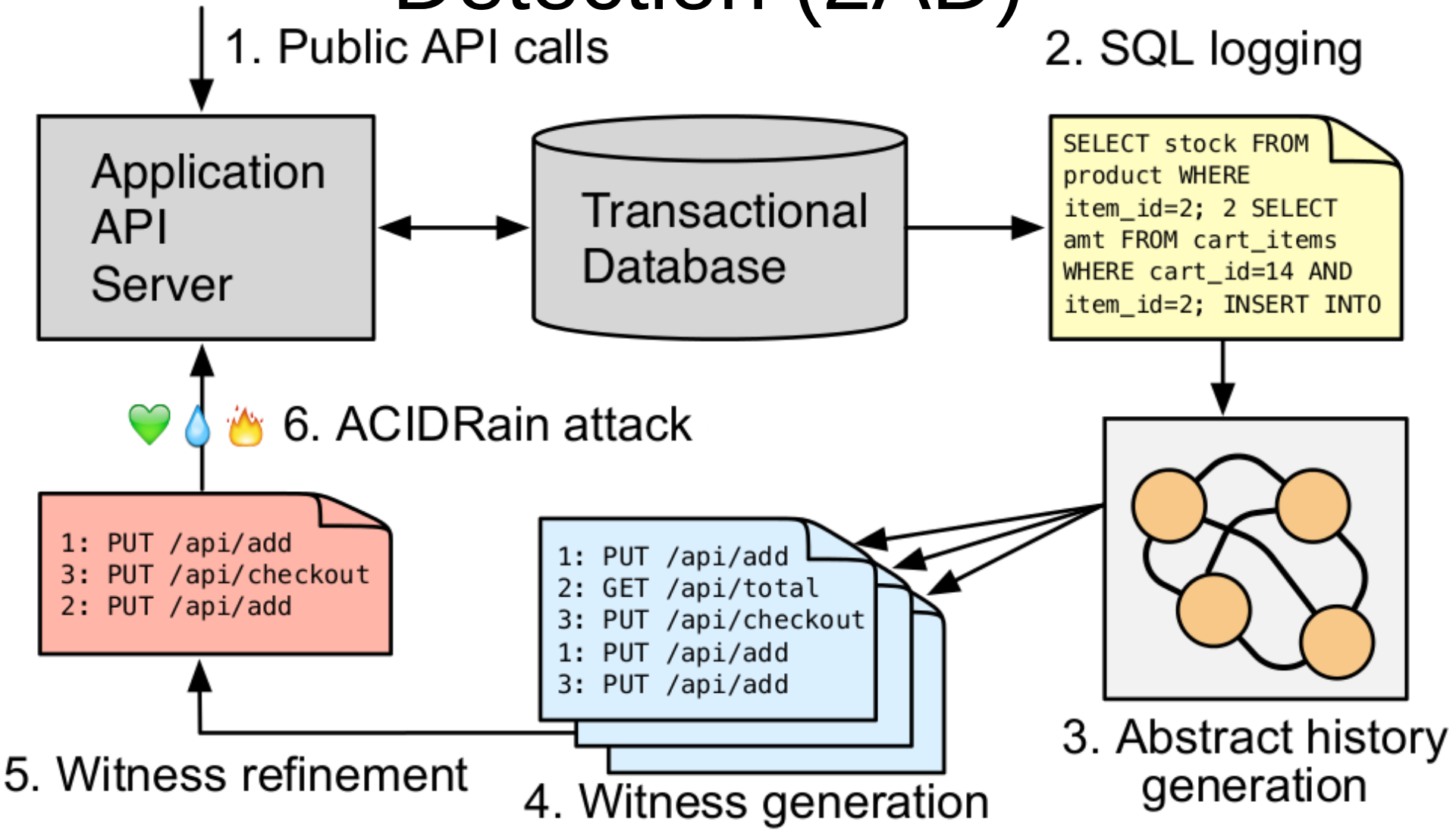
Approach: Abstract Anomaly Detection (2AD)



Approach: Abstract Anomaly Detection (2AD)



Approach: Abstract Anomaly Detection (2AD)



Approach: Abstract Anomaly Detection (2AD)

1. Collect (possibly serial) logs from database
2. Build compact representation of history (abstract history graph)
3. Search abstract history for cycles to generate possible anomalous API calls

Abstract History Graph

```
def checkVoucher(code):
    beginTxn()
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
    commit()
```

BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT
BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT

Abstract History Graph

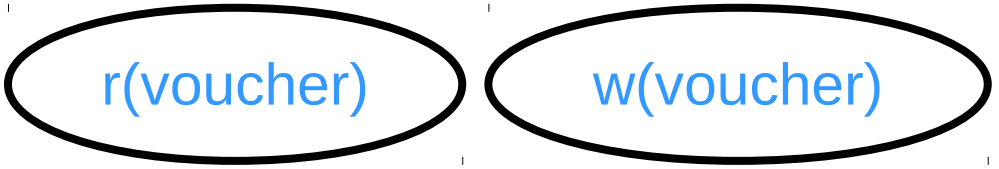
BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT
BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT

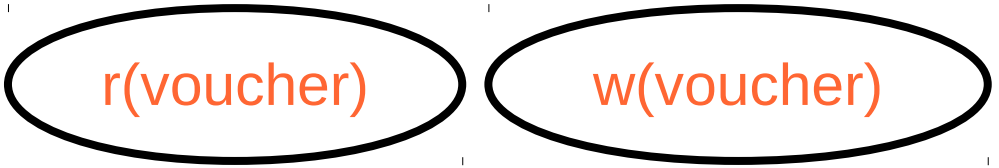
Abstract History Graph

BEGIN TRANSACTION
→ SELECT usage FROM voucher WHERE code = HNUHY
→ UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT
BEGIN TRANSACTION
→ SELECT usage FROM voucher WHERE code = HNUHY
→ UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT

 = Operation

1. Add node for each operation





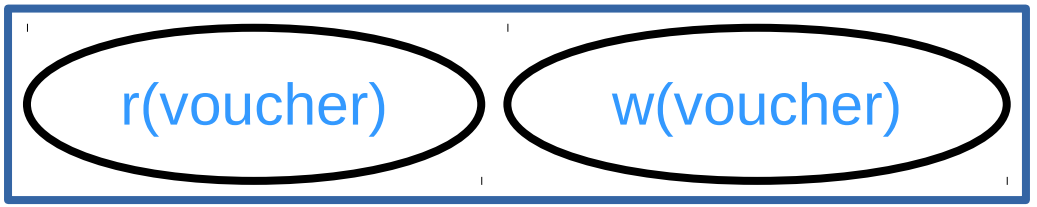
Abstract History Graph

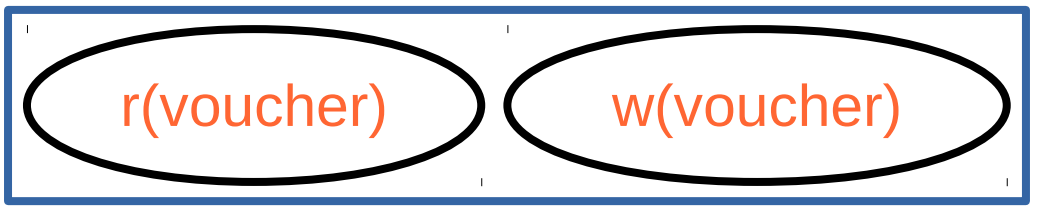
BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT

BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT


= Operation


= Transaction

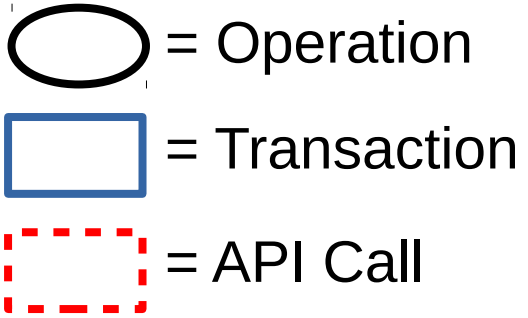


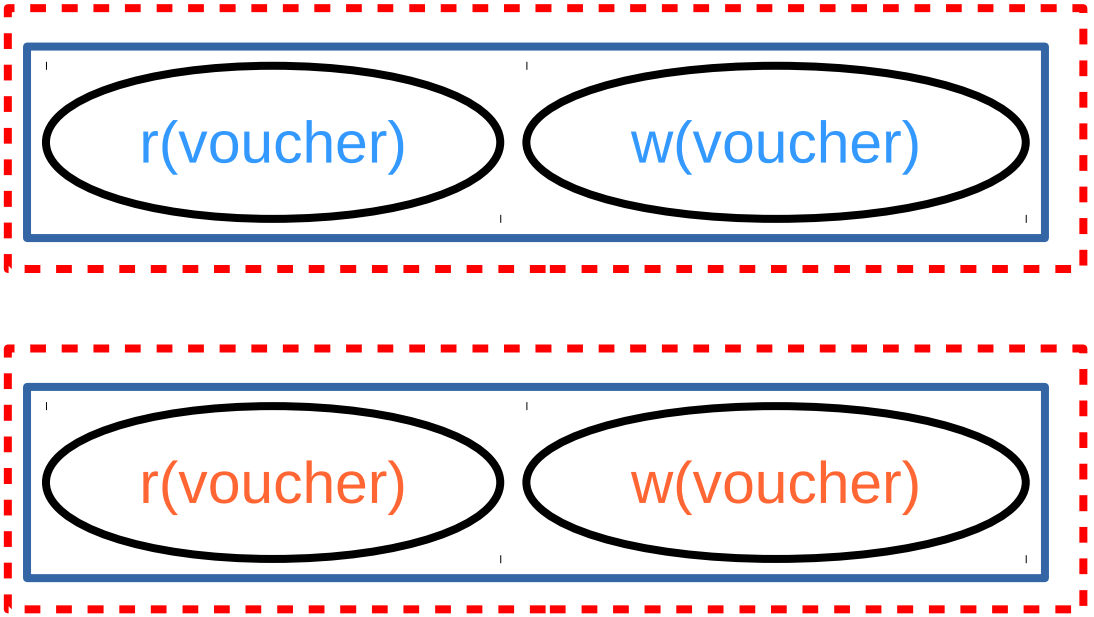


1. Add node for each operation
2. Add supernode for each transaction

Abstract History Graph

BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT
BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT

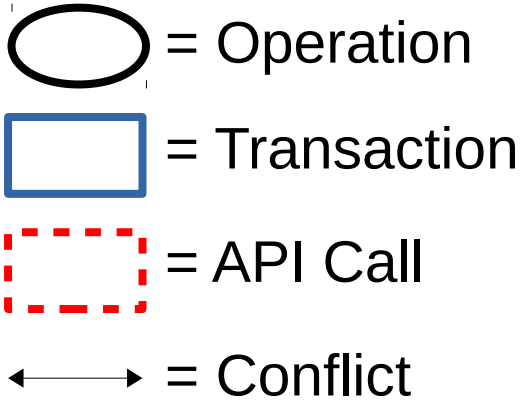


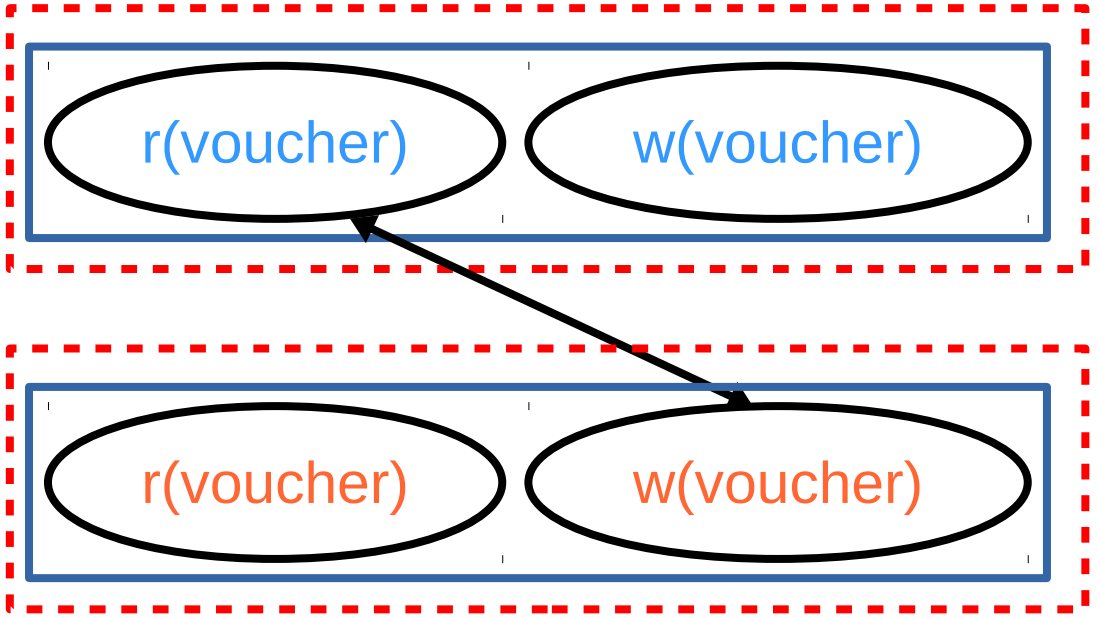


1. Add node for each operation
2. Add supernode for each transaction
3. Add super-supernode for each API call

Abstract History Graph

BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT
BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT

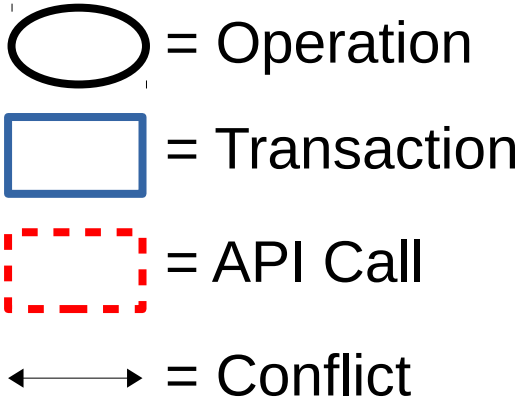


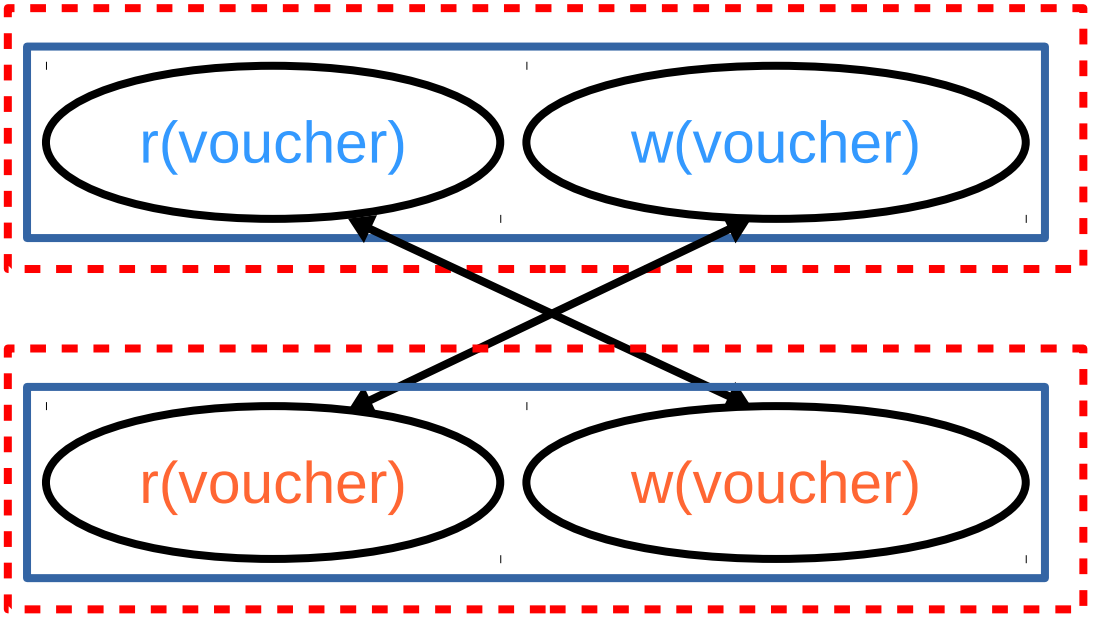


1. Add node for each operation
2. Add supernode for each transaction
3. Add super-supernode for each API call
4. Add edge for each conflict

Abstract History Graph

BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT
BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT

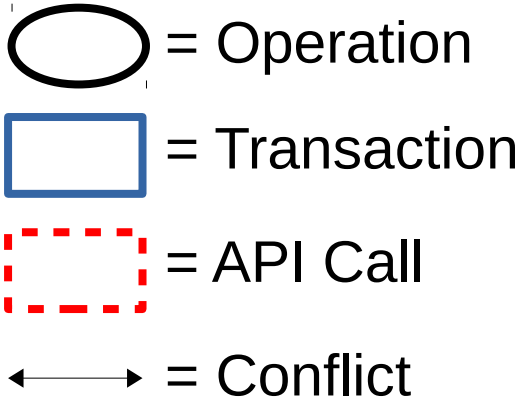


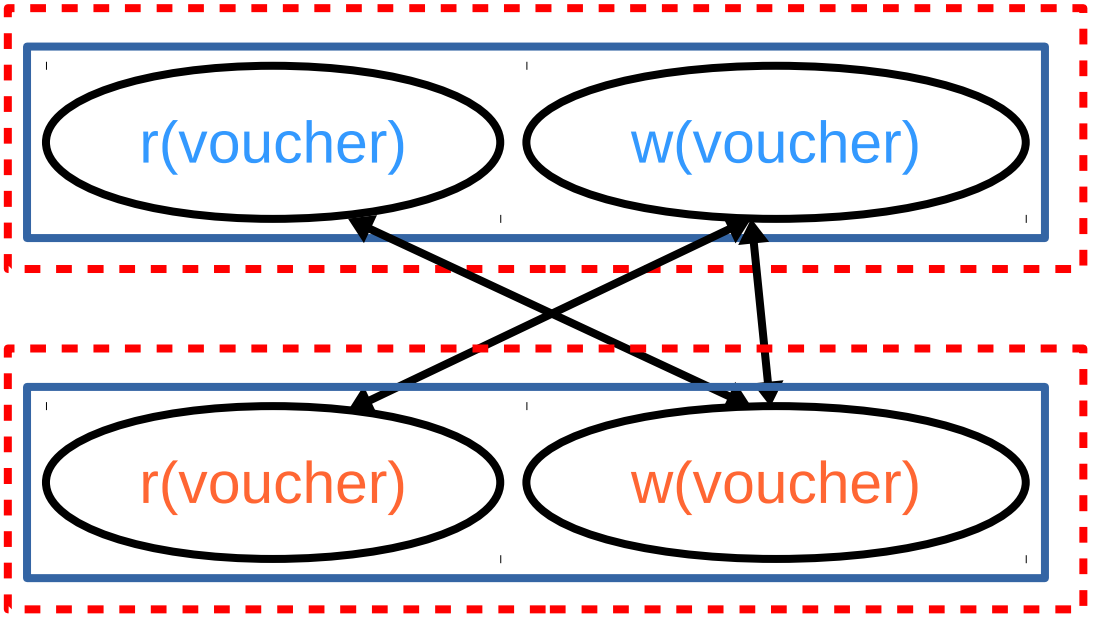


1. Add node for each operation
2. Add supernode for each transaction
3. Add super-supernode for each API call
4. Add edge for each conflict

Abstract History Graph

BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT
BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT

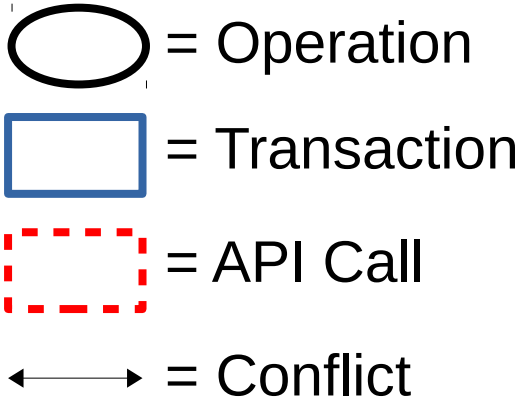


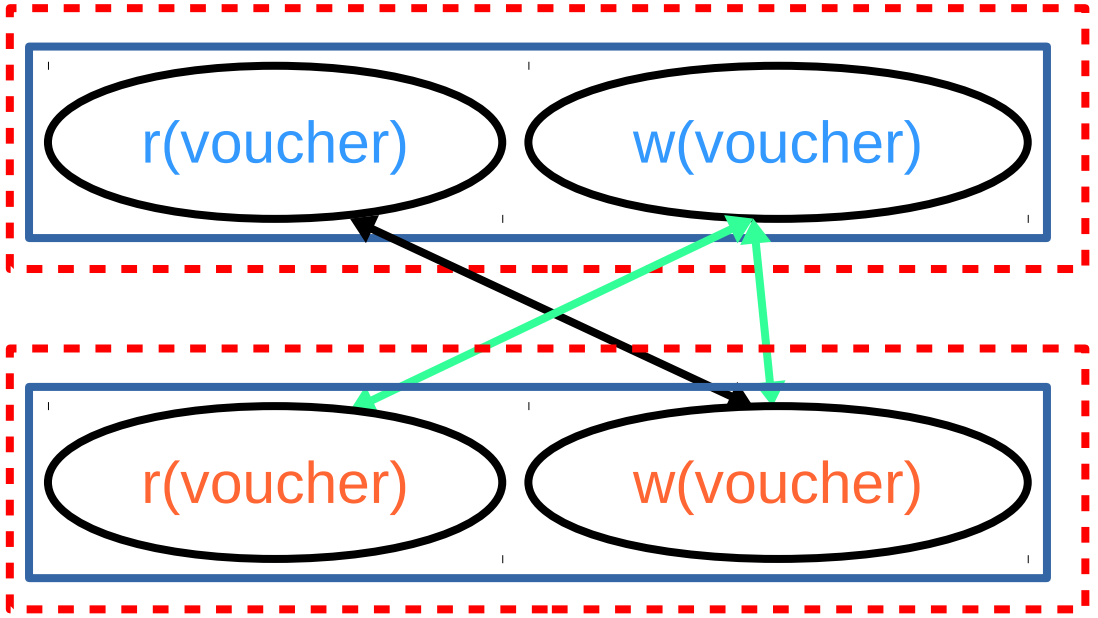


1. Add node for each operation
2. Add supernode for each transaction
3. Add super-supernode for each API call
4. Add edge for each conflict

Abstract History Graph

BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT
BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT

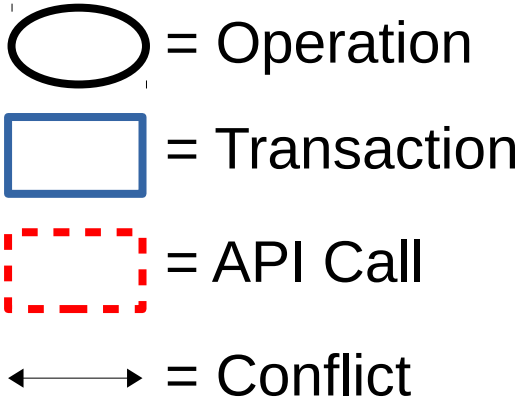


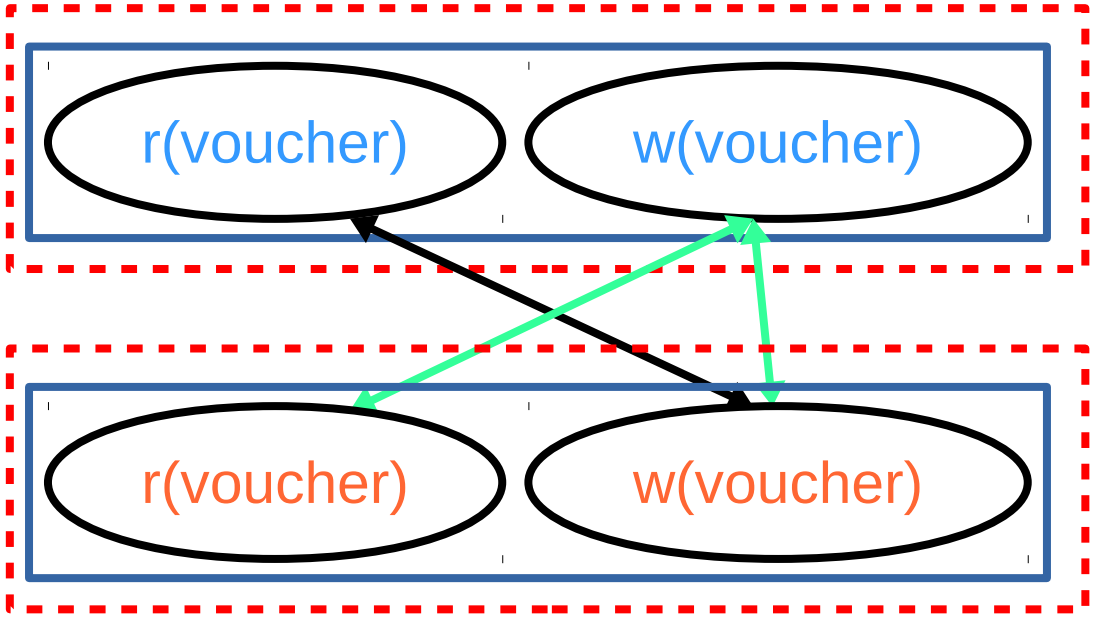


1. Add node for each operation
2. Add supernode for each transaction
3. Add super-supernode for each API call
4. Add edge for each conflict
5. Search for cycles in the graph

Abstract History Graph

BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT





1. Add node for each operation
2. Add supernode for each transaction
3. Add super-supernode for each API call
4. Add edge for each conflict
5. Search for cycles in the graph

Completeness Guarantees

Thm: Given a set of API calls, there exists an anomalous execution of the API calls if and only if there is a cycle in the abstract history.

- Completeness: if there is a potential anomalous execution, this approach will find it
- Soundness: discussion in paper

Limitations

- Does not take into account user level (i.e., "feral" [Bailis et al. 2015]) concurrency control

```
def checkVoucher(code):
    beginTxn()
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
    commit()
```

Limitations

- Does not take into account user level (i.e., "feral" [Bailis et al. 2015]) concurrency control

```
def checkVoucher(code):
    beginTxn()
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
    commit()
```

```
def checkVoucher(code):
    appLock.lock()
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
    appLock.release()
```

Overview

- Problem setup
- New method for detecting latent potential for non-serializable behavior
- Evaluation – analysis of 12 eCommerce platforms

Ecommerce Platforms

opencart

oscar

Magento® Community

shopizer

broadleaf commerce

spree™ commerce

ror-e.com

LFS

WOO COMMERCE

PrestaShop

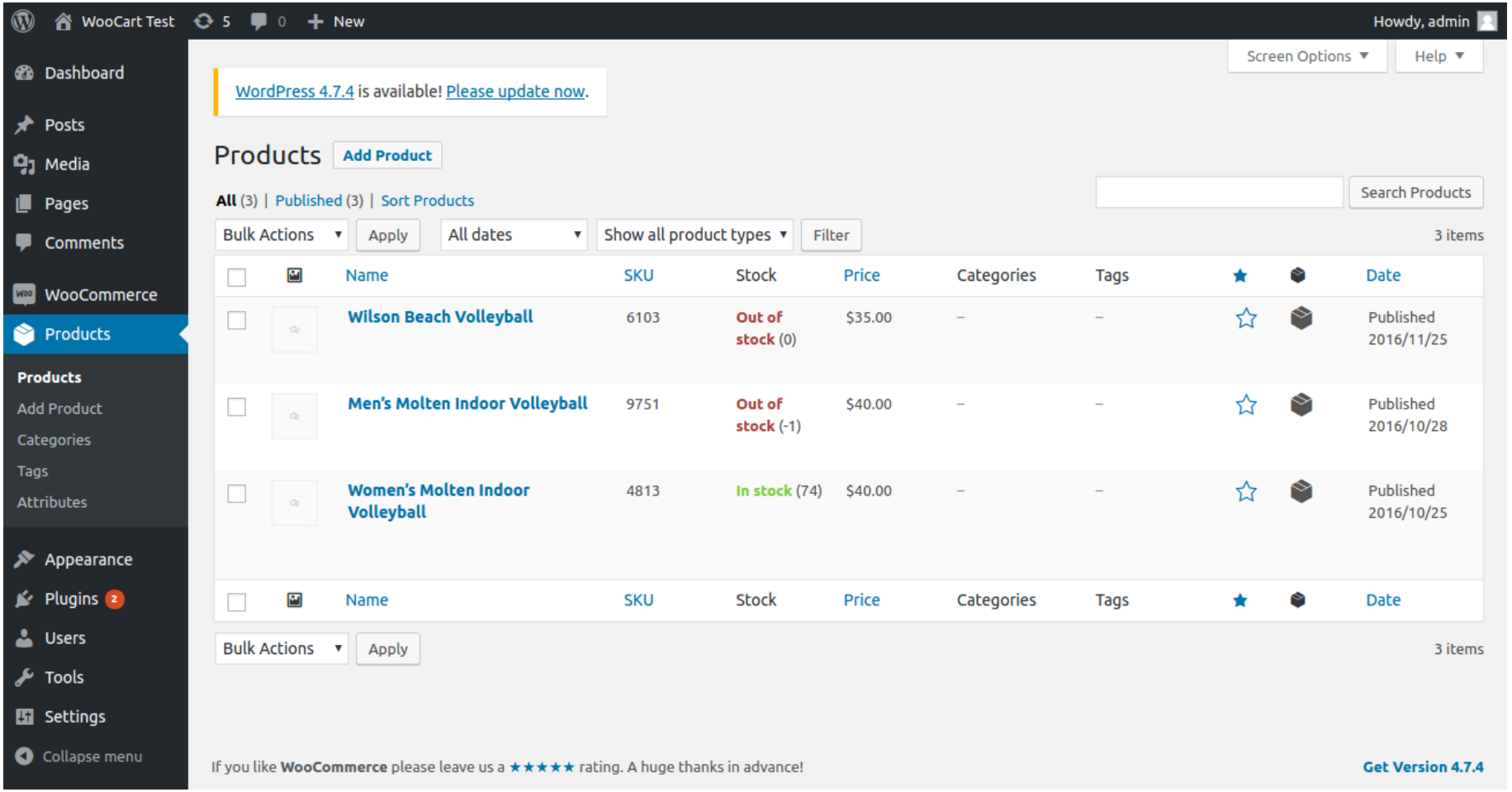
SHOPPE

Ecommerce Platforms

Interested in 3
key invariants:
1. Inventory
2. Voucher
3. Cart

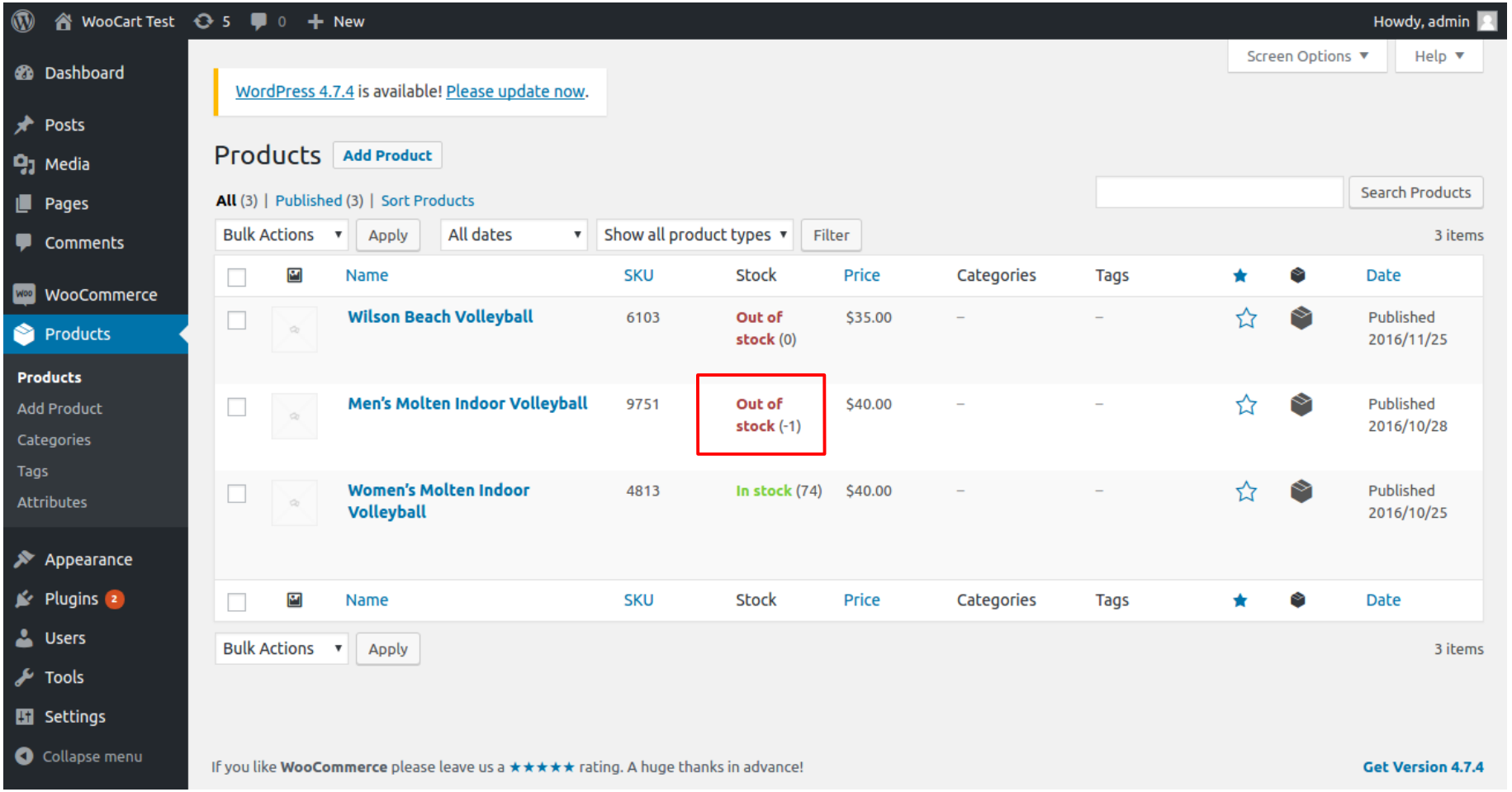
Inventory Invariant

Stock should not go below 0 and should reflect all orders placed



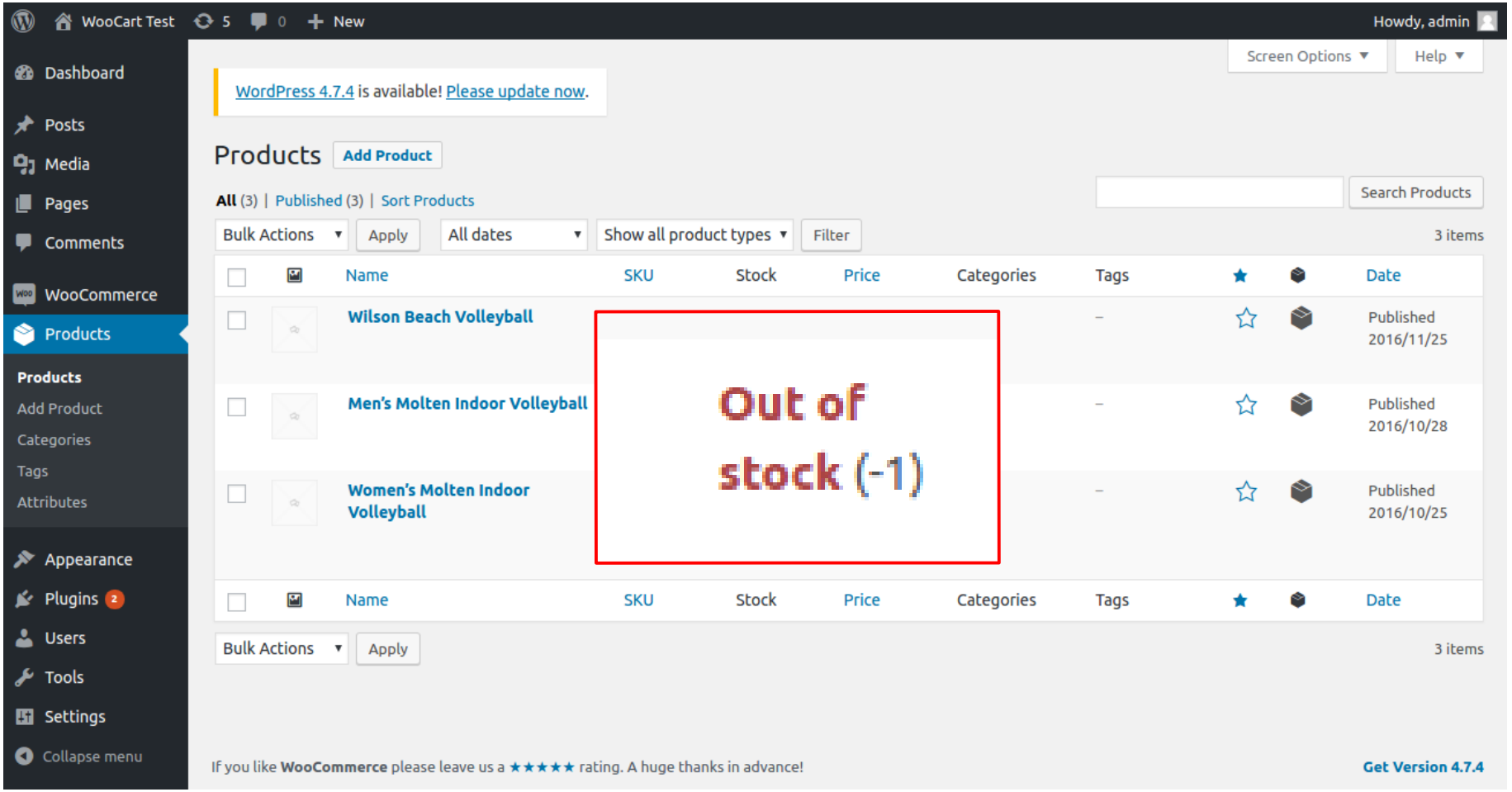
Inventory Invariant

Stock should not go below 0 and should reflect all orders placed



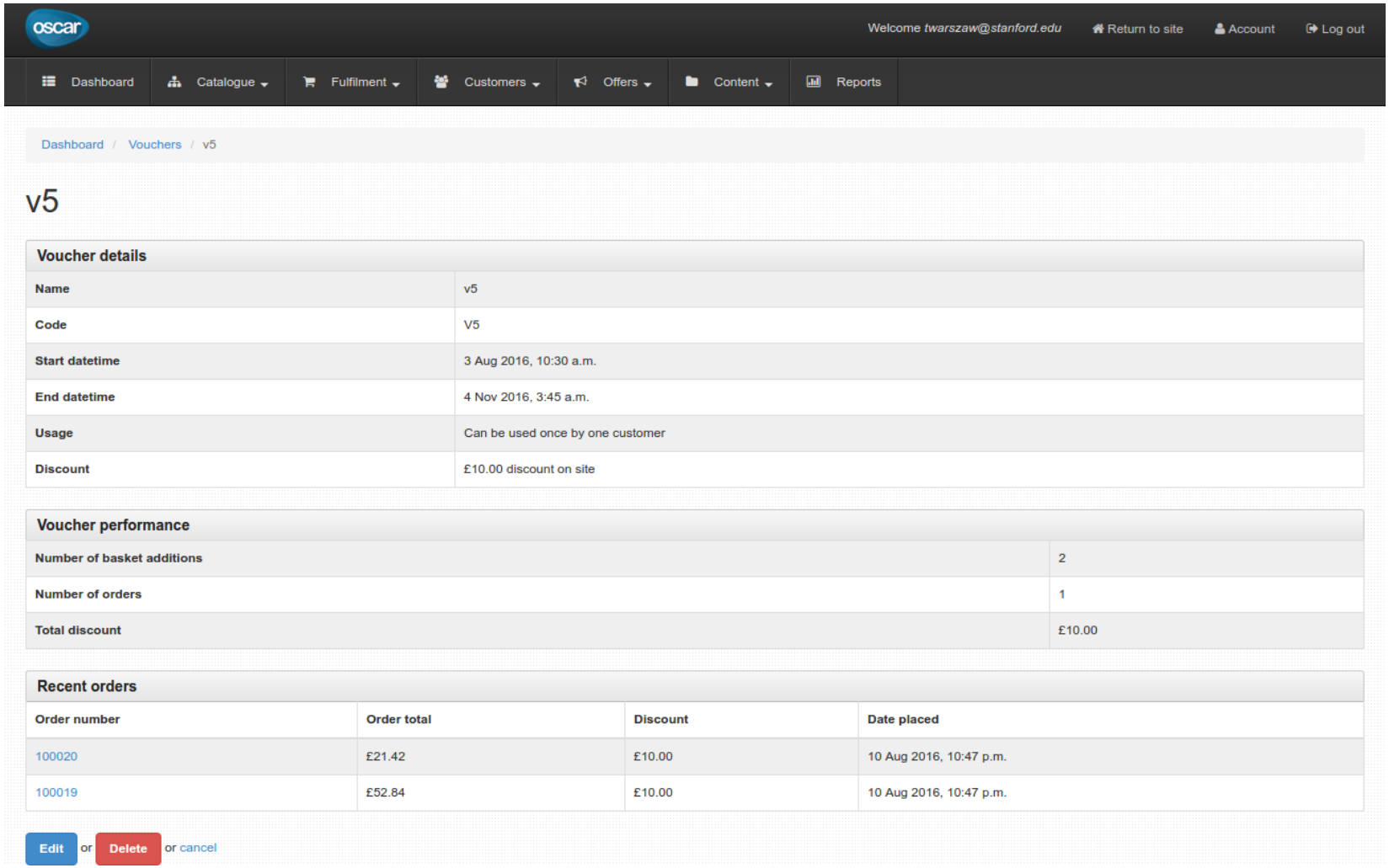
Inventory Invariant

Stock should not go below 0 and should reflect all orders placed



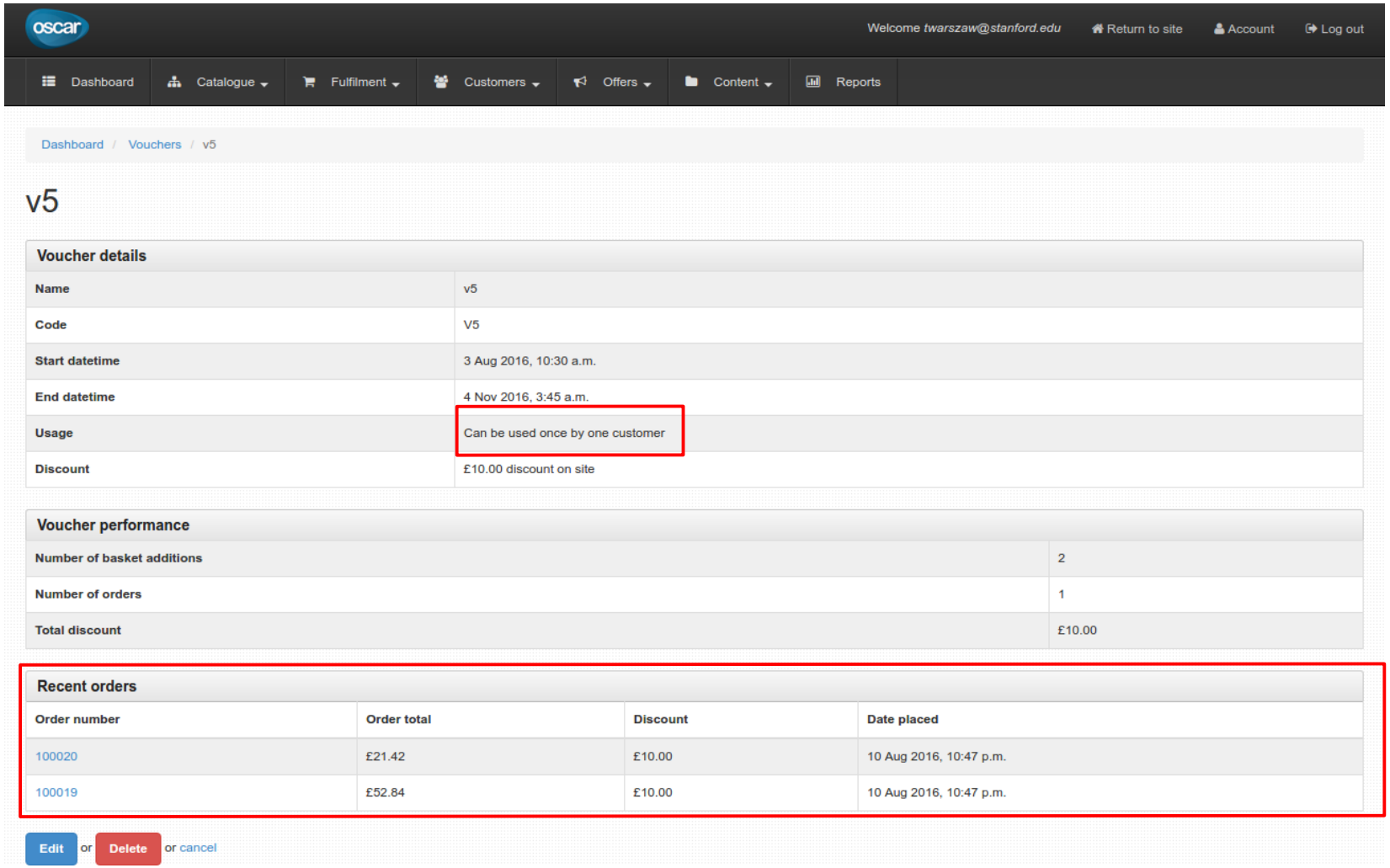
Voucher Invariant

Vouchers should not be spent past their intended limit



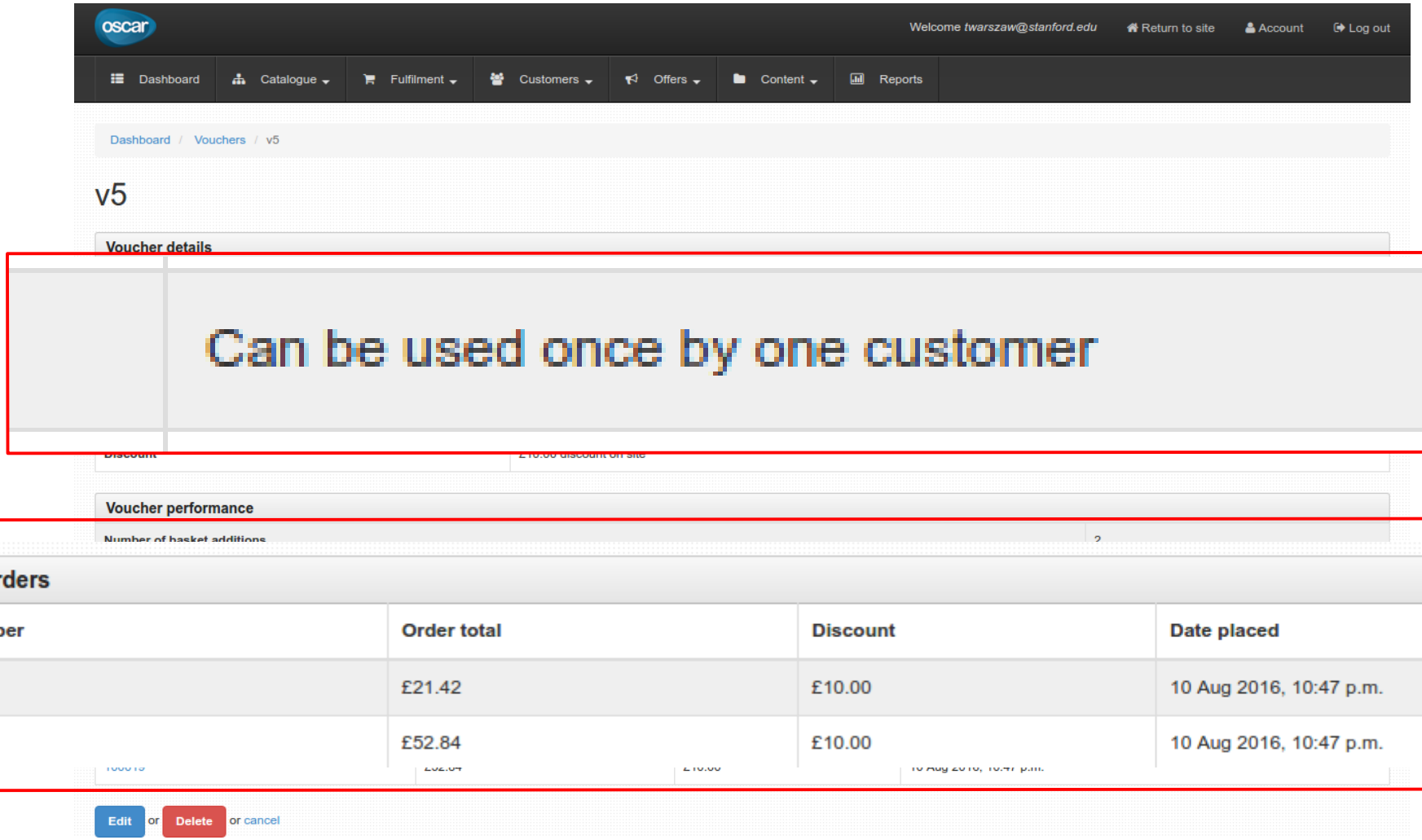
Voucher Invariant

Vouchers should not be spent past their intended limit



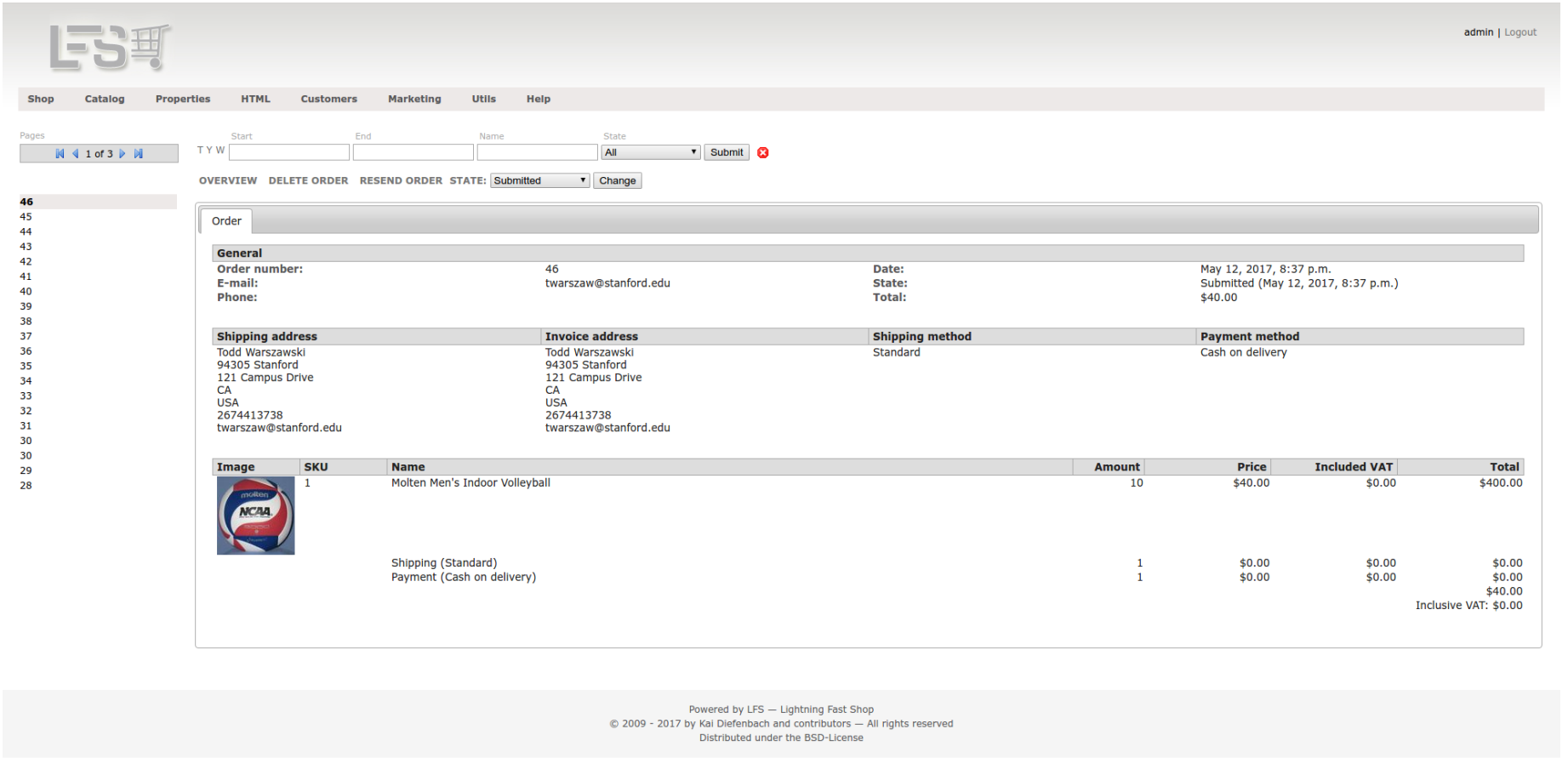
Voucher Invariant

Vouchers should not be spent past their intended limit



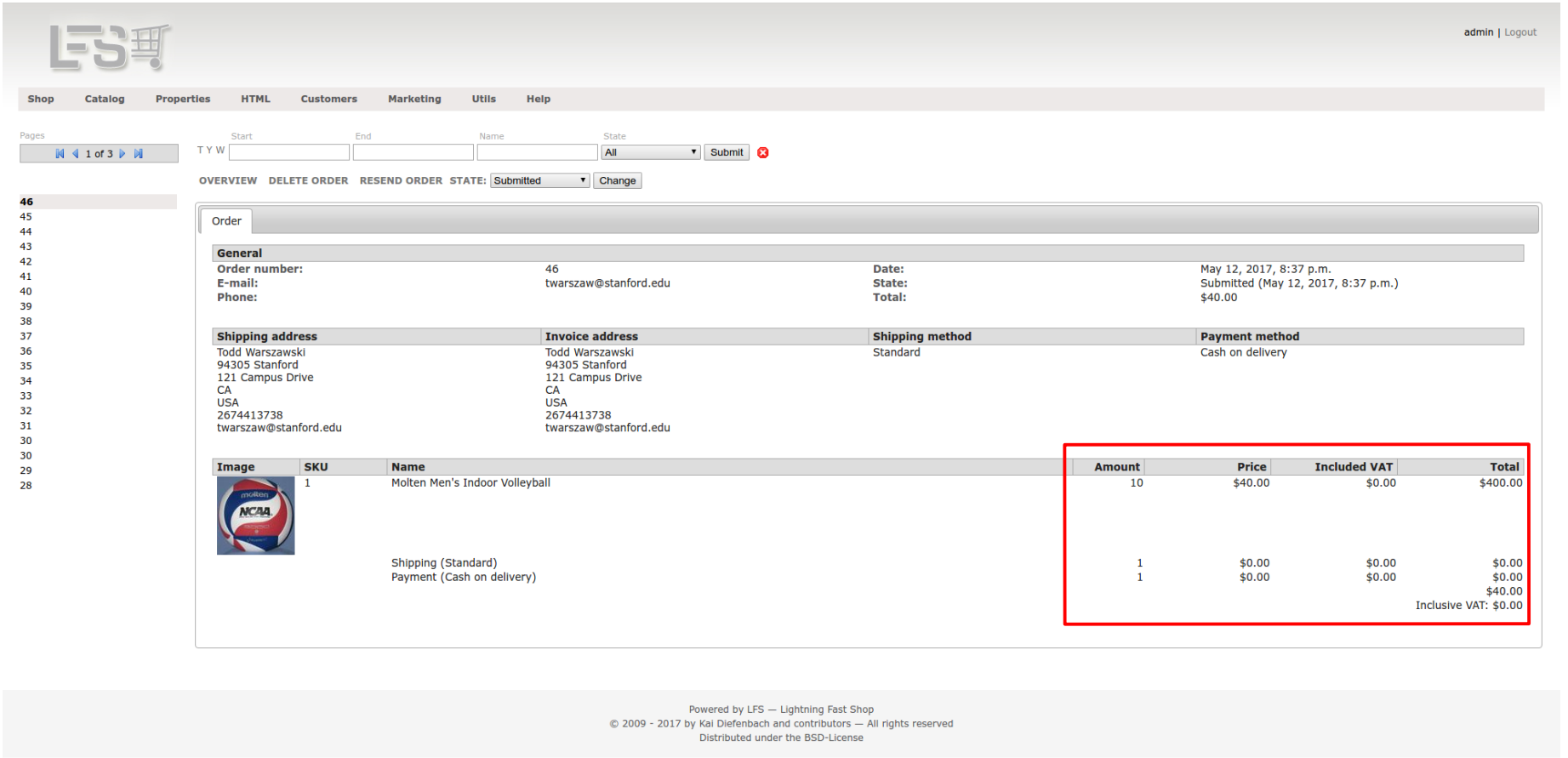
Cart Invariant

Total charged for an order should be equal to the value of items associated with the order



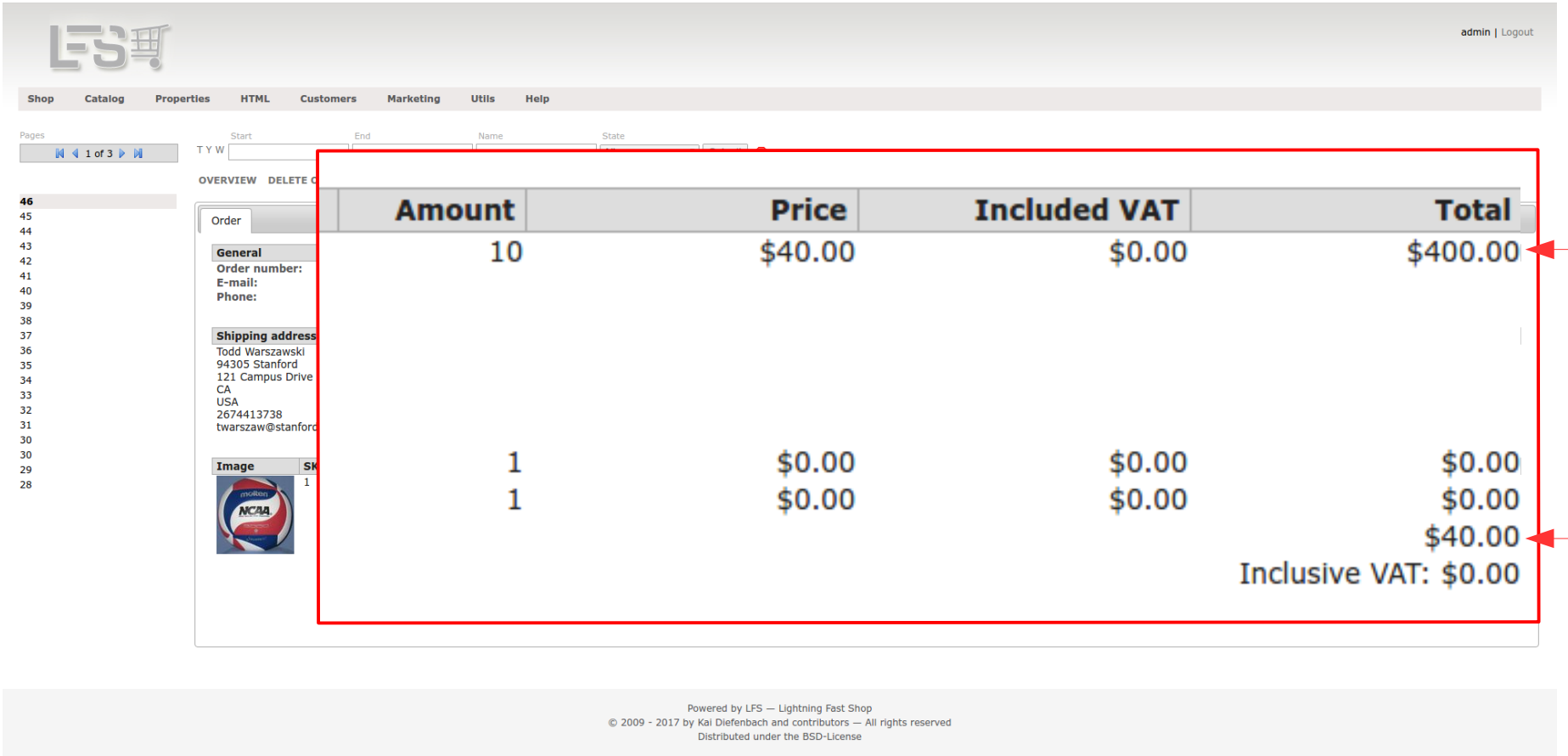
Cart Invariant

Total charged for an order should be equal to the value of items associated with the order



Cart Invariant

Total charged for an order should be equal to the value of items associated with the order



Analysis Results

22 new vulnerabilities!

✗ = vulnerable, ✔ = not vulnerable

Application	Language	Inventory	Voucher	Cart
Opencart	PHP	✗	✗	✔
Prestashop	PHP	✗	✗	✔
Magento	PHP	✗	✗	✔
WooCommerce	PHP	✗	✗	✔
Spree	Ruby on Rails	✔	✔	✔
Ror_ecommerce	Ruby on Rails	✗	N/A	✗
Shoppe	Ruby on Rails	✗	N/A	✗
Oscar	Python (Django)	✗	✗	✔
LFS	Python (Django)	✗	✗	✗
Saleor	Python (Django)	✗	✗	N/A
Broadleaf	Java (Spring)	N/A	✗	✗
Shopizer	Java (Spring)	N/A	N/A	✗

Analysis Results

22 new vulnerabilities!

✗ = vulnerable, ✔ = not vulnerable

Application	Language	Inventory	Voucher	Cart
Opencart	PHP	✗	✗	✔
Prestashop	PHP	✗	✗	✔
Magento	PHP	✗	✗	✔
WooCommerce	PHP	✗	✗	✔
Spree	Ruby on Rails	✔	✔	✔
Ror_ecommerce	Ruby on Rails	✗	N/A	✗
Shoppe	Ruby on Rails	✗	N/A	✗
Oscar	Python (Django)	✗	✗	✔
LFS	Python (Django)	✗	✗	✗
Saleor	Python (Django)	✗	✗	N/A
Broadleaf	Java (Spring)	N/A	✗	✗
Shopizer	Java (Spring)	N/A	N/A	✗

2M+ sites at risk

Analysis Results

22 new vulnerabilities!

✗ = vulnerable, ✔ = not vulnerable

Application	Language	Inventory	Voucher	Cart
Opencart	PHP	✗	✗	✔
Prestashop	PHP	✗	✗	✔
Magento	PHP	✗	✗	✔
WooCommerce	PHP	✗	✗	✔
Spree	Ruby on Rails	✔	✔	✔
Ror_ecommerce	Ruby on Rails	✗	N/A	✗
Shoppe	Ruby on Rails	✗	N/A	✗
Oscar	Python (Django)	✗	✗	✔
LFS	Python (Django)	✗	✗	✗
Saleor	Python (Django)	✗	✗	N/A
Broadleaf	Java (Spring)	N/A	✗	✗
Shopizer	Java (Spring)	N/A	N/A	✗

2M+ sites at risk

4 different languages

Analysis Results

22 new vulnerabilities!

✗ = vulnerable, ✔ = not vulnerable

Application	Language	Inventory	Voucher	Cart
Opencart	PHP	✗	✗	✔
Prestashop	PHP	✗	✗	✔
Magento	PHP	✗	✗	✔
WooCommerce	PHP	✗	✗	✔
Spree	Ruby on Rails	✔	✔	✔
Ror_ecommerce	Ruby on Rails	✗	N/A	✗
Shoppe	Ruby on Rails	✗	N/A	✗
Oscar	Python (Django)	✗	✗	✔
LFS	Python (Django)	✗	✗	✗
Saleor	Python (Django)	✗	✗	N/A
Broadleaf	Java (Spring)	N/A	✗	✗
Shopizer	Java (Spring)	N/A	N/A	✗

2M+ sites at risk

4 different languages

5 errors due to DB default isolation

Analysis Results

22 new vulnerabilities!

✗ = vulnerable, ✔ = not vulnerable

Application	Language	Inventory	Voucher	Cart
Opencart	PHP	✗	✗	✔
Prestashop	PHP	✗	✗	✔
Magento	PHP	✗	✗	✔
WooCommerce	PHP	✗	✗	✔
Spree	Ruby on Rails	✔	✔	✔
Ror_ecommerce	Ruby on Rails	✗	N/A	✗
Shoppe	Ruby on Rails	✗	N/A	✗
Oscar	Python (Django)	✗	✗	✔
LFS	Python (Django)	✗	✗	✗
Saleor	Python (Django)	✗	✗	N/A
Broadleaf	Java (Spring)	N/A	✗	✗
Shopizer	Java (Spring)	N/A	N/A	✗

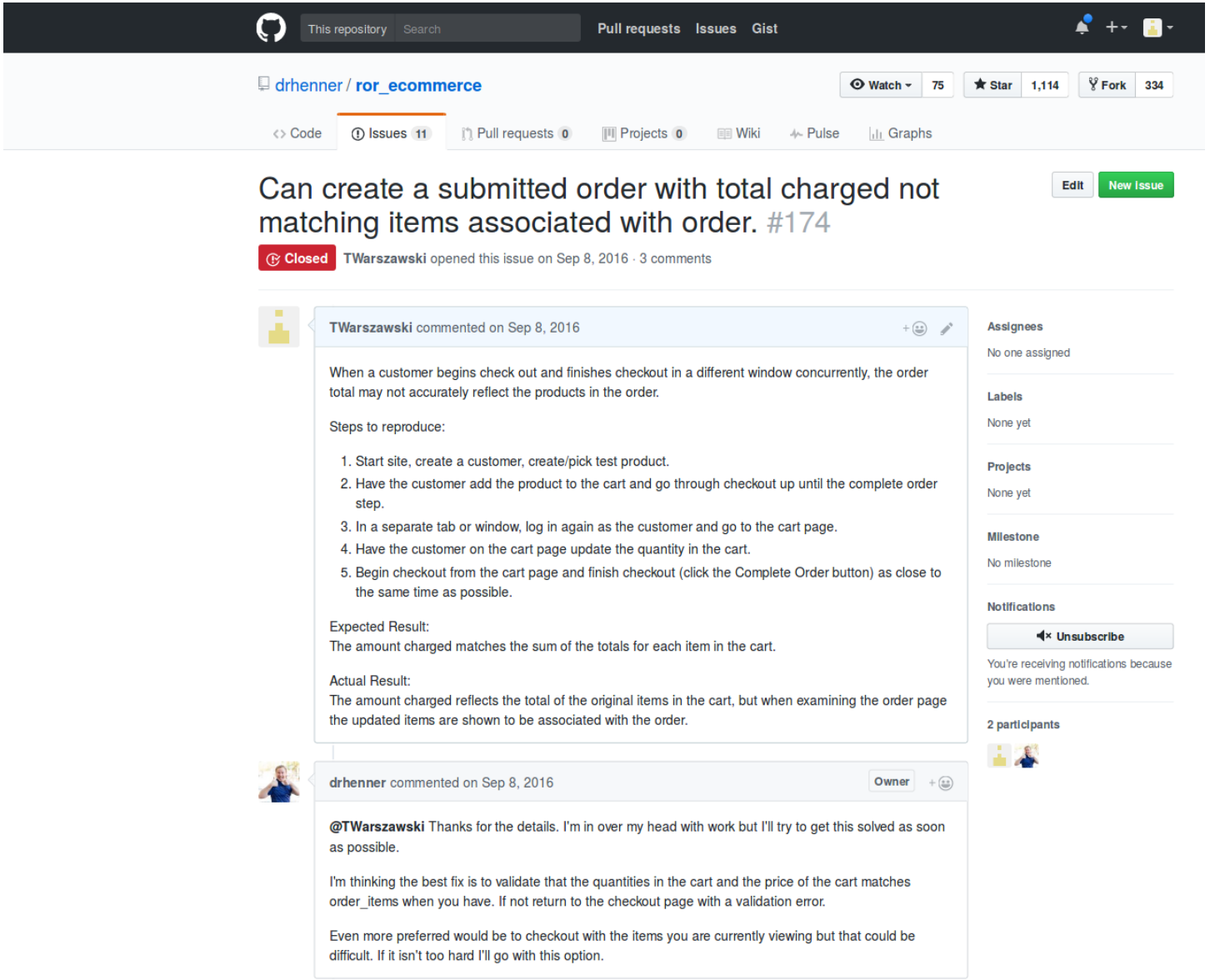
2M+ sites at risk

4 different languages

5 errors due to DB default isolation

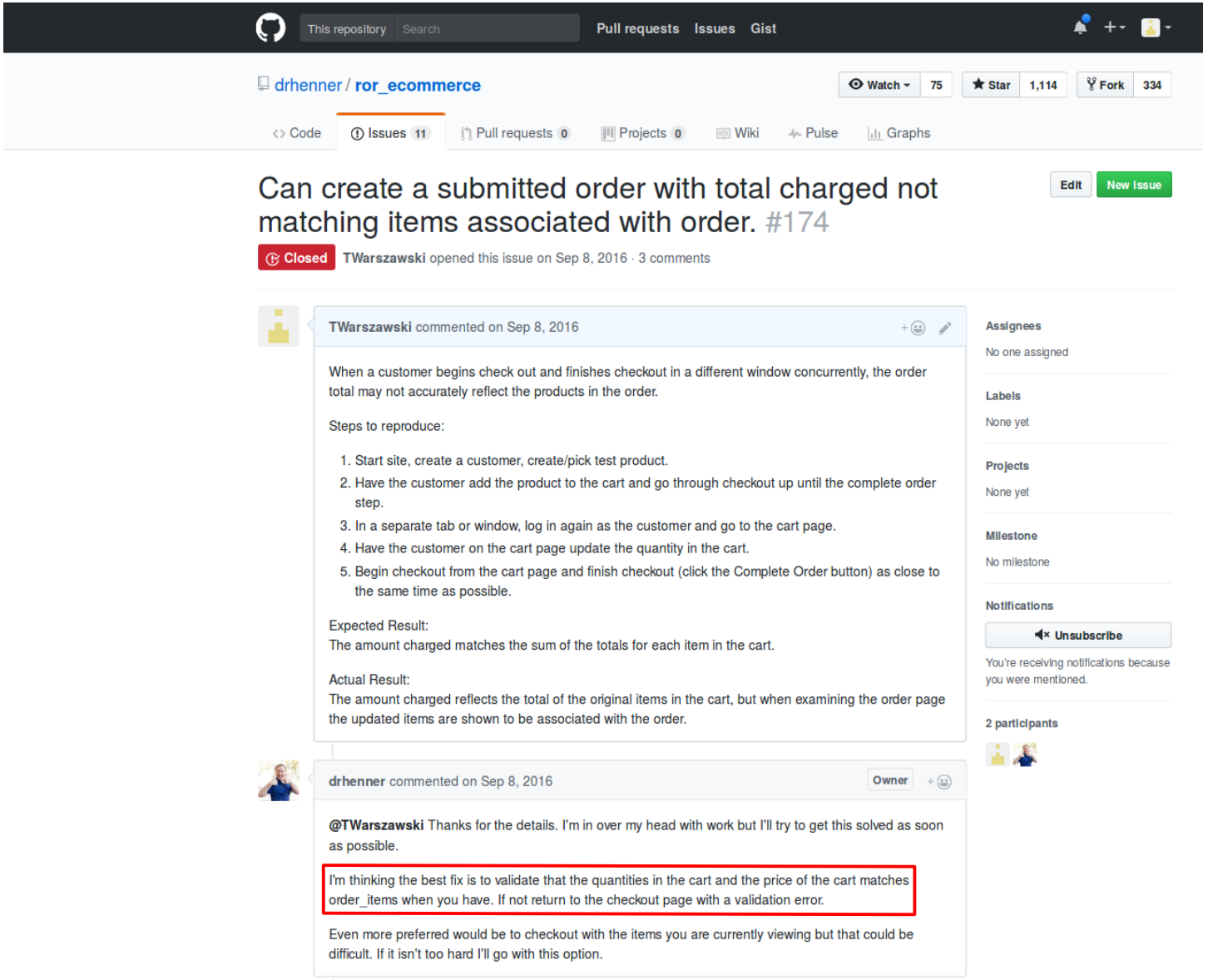
17 errors due to improper transaction usage

Developer Response



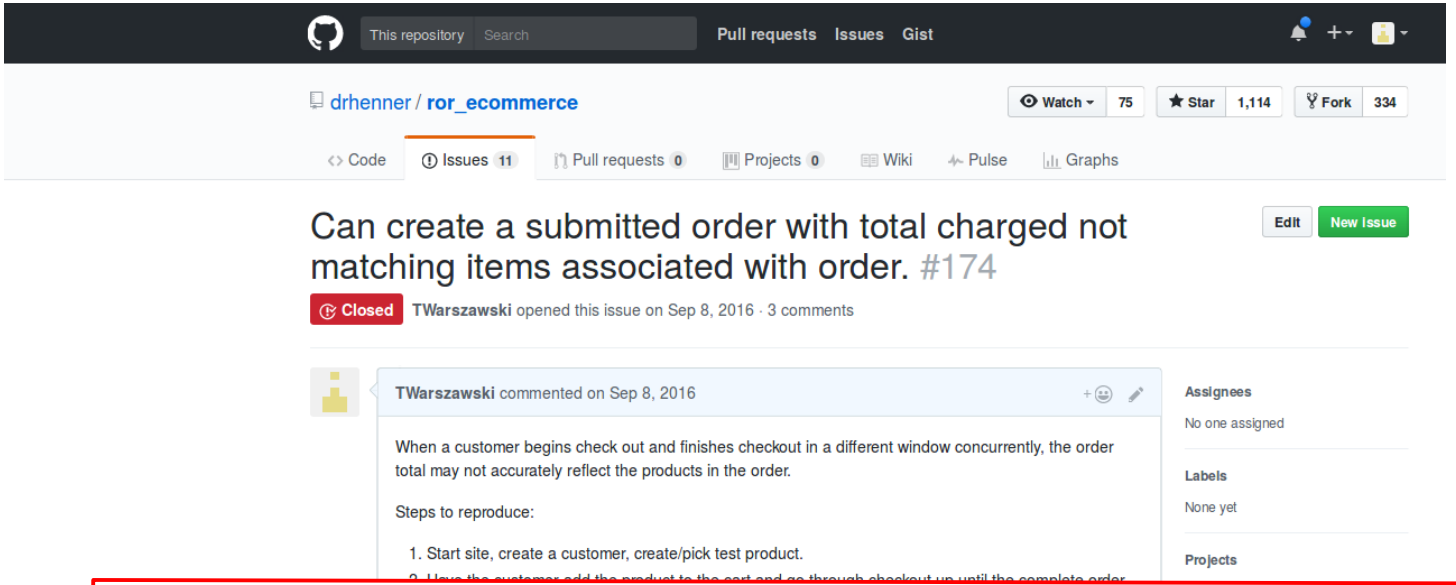
8 confirmed

Developer Response



8 confirmed

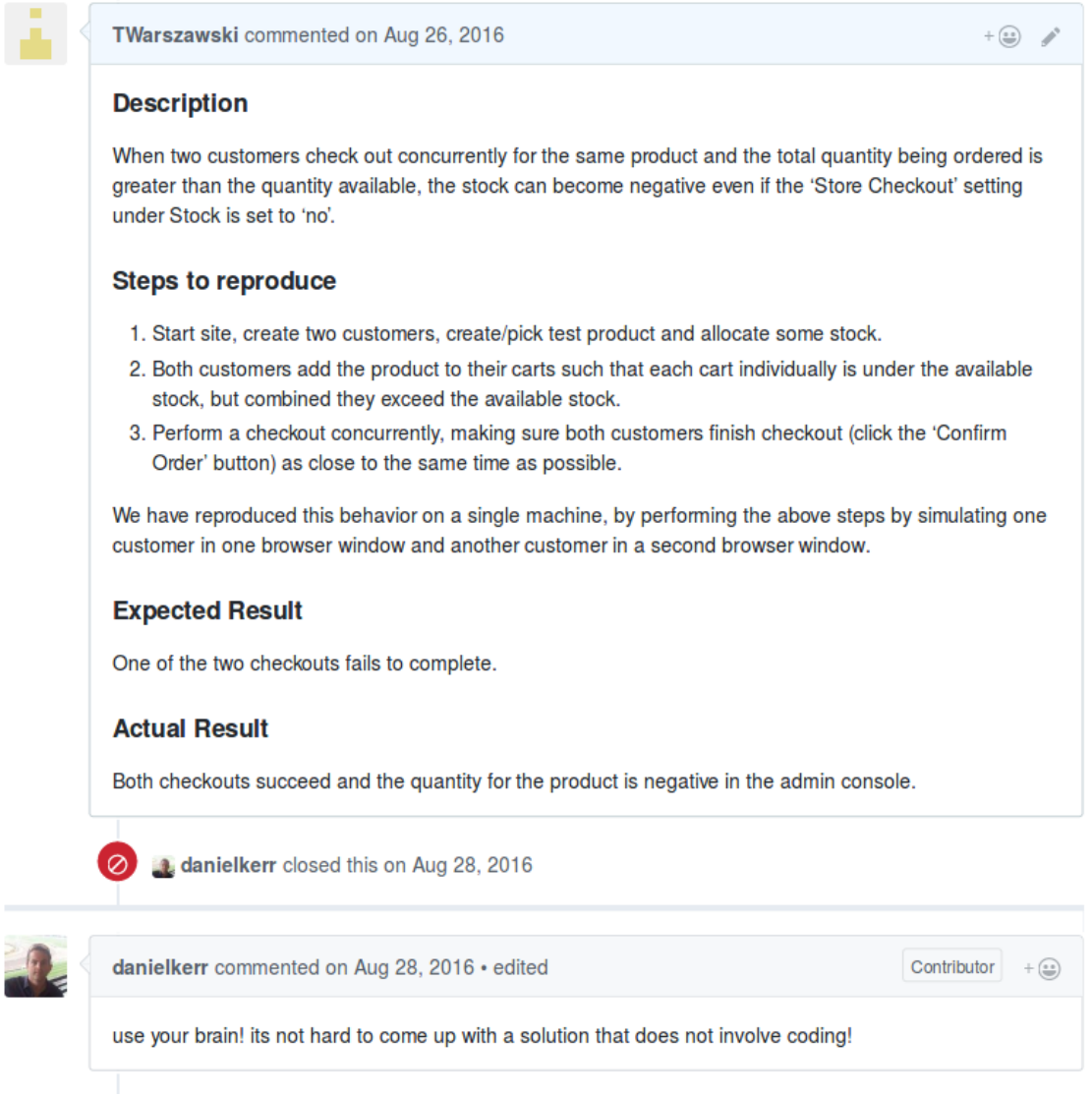
Developer Response



8 confirmed

I'm thinking the best fix is to validate that the quantities in the cart and the price of the cart matches order_items when you have. If not return to the checkout page with a validation error.

Developer Response



TWarszawski commented on Aug 26, 2016

Description

When two customers check out concurrently for the same product and the total quantity being ordered is greater than the quantity available, the stock can become negative even if the 'Store Checkout' setting under Stock is set to 'no'.

Steps to reproduce

1. Start site, create two customers, create/pick test product and allocate some stock.
2. Both customers add the product to their carts such that each cart individually is under the available stock, but combined they exceed the available stock.
3. Perform a checkout concurrently, making sure both customers finish checkout (click the 'Confirm Order' button) as close to the same time as possible.

We have reproduced this behavior on a single machine, by performing the above steps by simulating one customer in one browser window and another customer in a second browser window.

Expected Result

One of the two checkouts fails to complete.

Actual Result

Both checkouts succeed and the quantity for the product is negative in the admin console.

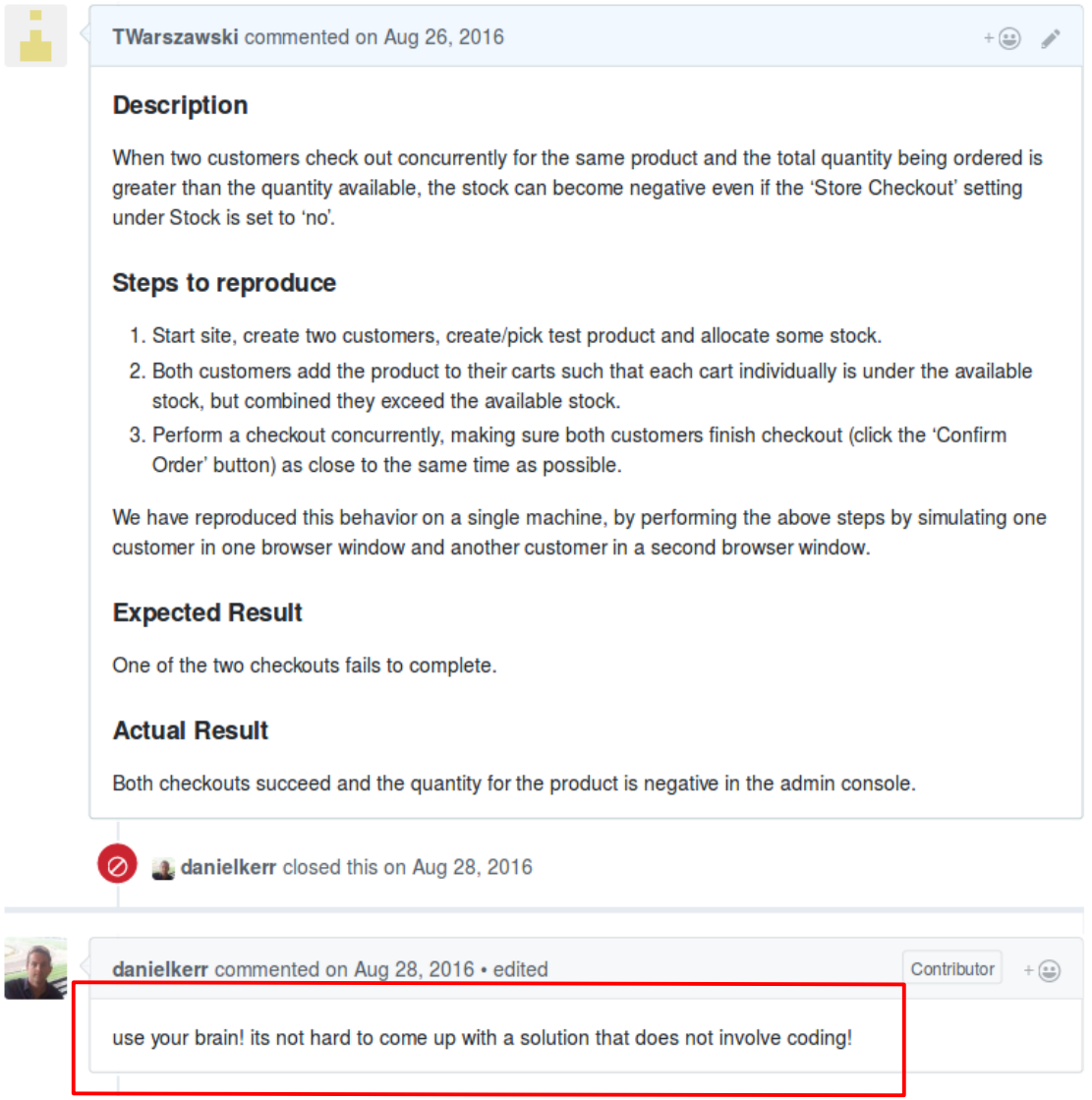
danielkerr closed this on Aug 28, 2016

danielkerr commented on Aug 28, 2016 • edited Contributor

use your brain! its not hard to come up with a solution that does not involve coding!

8 confirmed

Developer Response



TWarszawski commented on Aug 26, 2016

Description

When two customers check out concurrently for the same product and the total quantity being ordered is greater than the quantity available, the stock can become negative even if the 'Store Checkout' setting under Stock is set to 'no'.

Steps to reproduce

1. Start site, create two customers, create/pick test product and allocate some stock.
2. Both customers add the product to their carts such that each cart individually is under the available stock, but combined they exceed the available stock.
3. Perform a checkout concurrently, making sure both customers finish checkout (click the 'Confirm Order' button) as close to the same time as possible.

We have reproduced this behavior on a single machine, by performing the above steps by simulating one customer in one browser window and another customer in a second browser window.

Expected Result

One of the two checkouts fails to complete.

Actual Result

Both checkouts succeed and the quantity for the product is negative in the admin console.

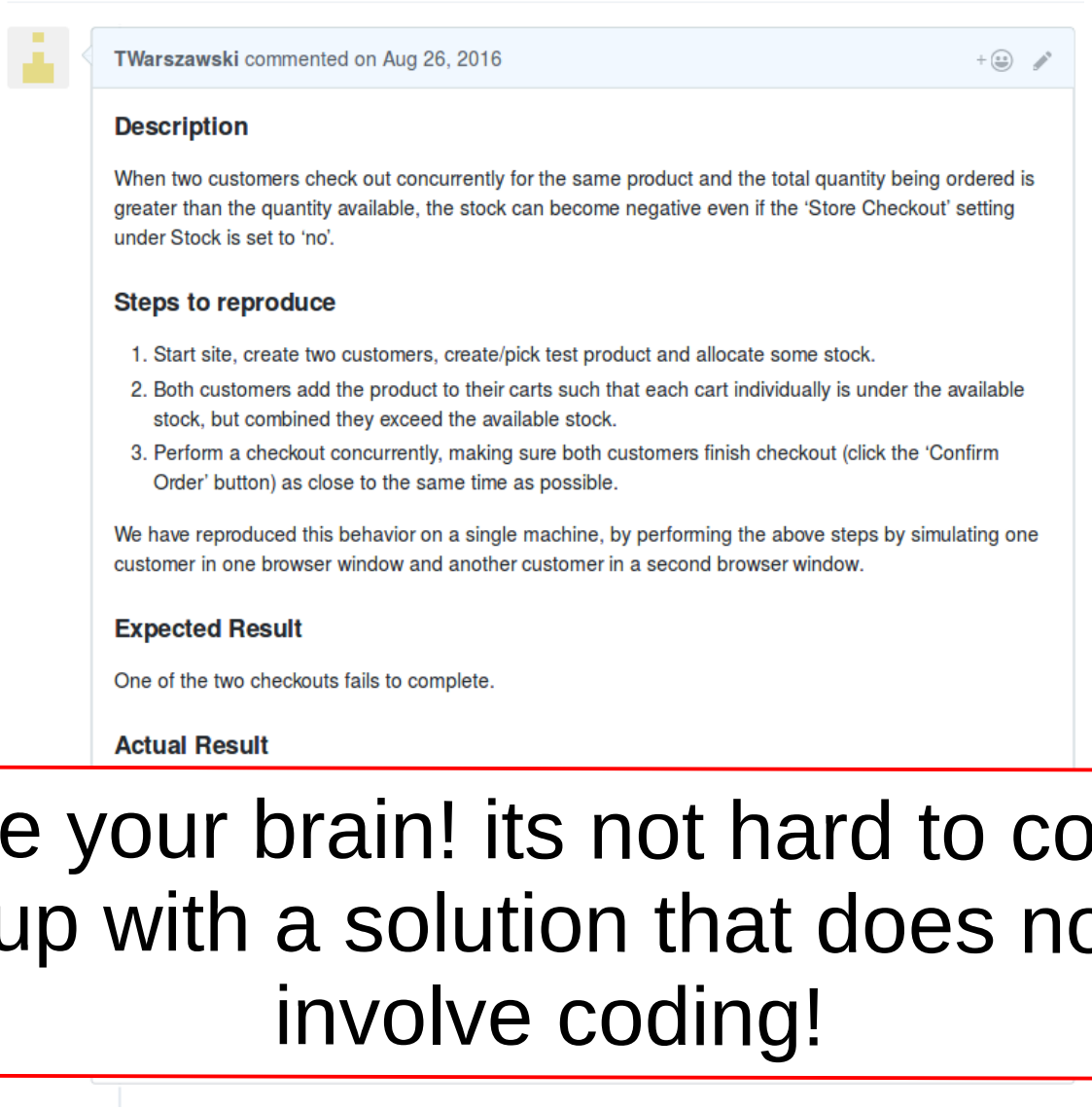
danielkerr closed this on Aug 28, 2016

danielkerr commented on Aug 28, 2016 • edited Contributor

use your brain! its not hard to come up with a solution that does not involve coding!

8 confirmed

Developer Response



TWarszawski commented on Aug 26, 2016

Description

When two customers check out concurrently for the same product and the total quantity being ordered is greater than the quantity available, the stock can become negative even if the 'Store Checkout' setting under Stock is set to 'no'.

Steps to reproduce

1. Start site, create two customers, create/pick test product and allocate some stock.
2. Both customers add the product to their carts such that each cart individually is under the available stock, but combined they exceed the available stock.
3. Perform a checkout concurrently, making sure both customers finish checkout (click the 'Confirm Order' button) as close to the same time as possible.

We have reproduced this behavior on a single machine, by performing the above steps by simulating one customer in one browser window and another customer in a second browser window.

Expected Result

One of the two checkouts fails to complete.

Actual Result

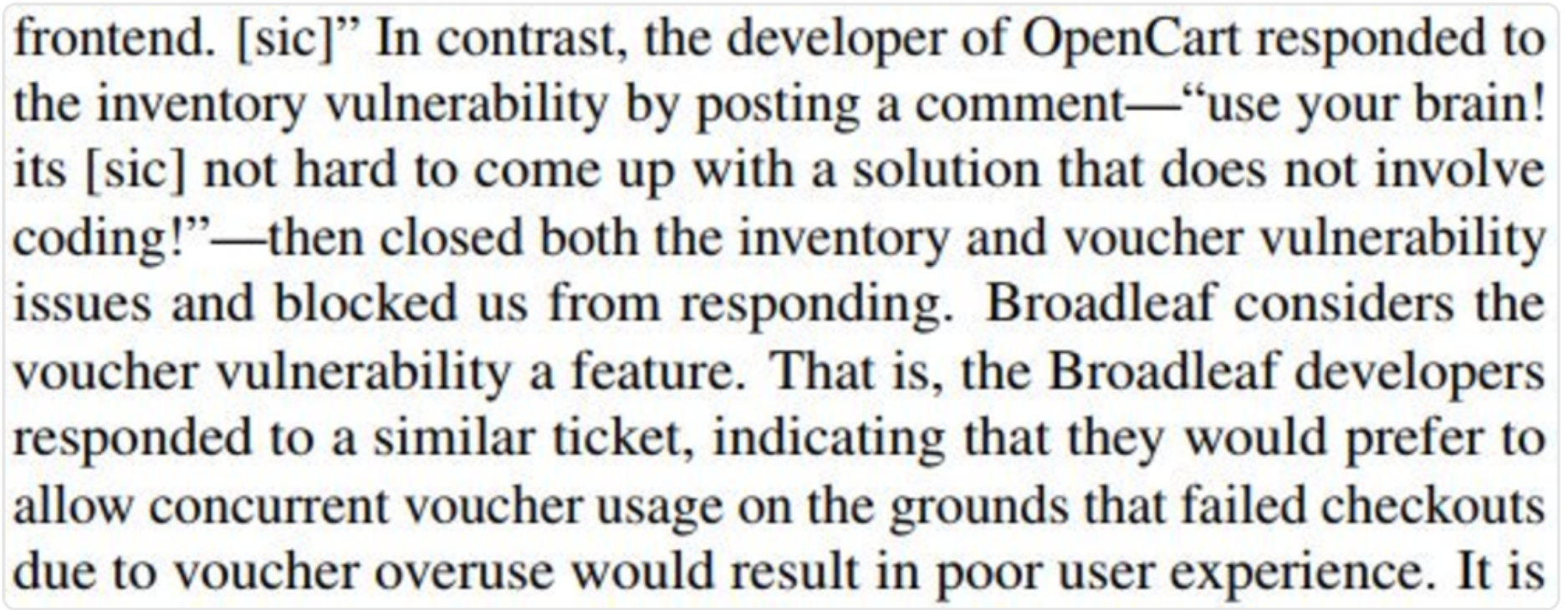
8 confirmed

use your brain! its not hard to come up with a solution that does not involve coding!

Hector Martin
@marcan42

Follow

Now you know which eCommerce platforms to stay away from. I don't even.
twitter.com/pbailis/status …
github.com/opencart/openc …



frontend. [sic]" In contrast, the developer of OpenCart responded to the inventory vulnerability by posting a comment—"use your brain! its [sic] not hard to come up with a solution that does not involve coding!"—then closed both the inventory and voucher vulnerability issues and blocked us from responding. Broadleaf considers the voucher vulnerability a feature. That is, the Broadleaf developers responded to a similar ticket, indicating that they would prefer to allow concurrent voucher usage on the grounds that failed checkouts due to voucher overuse would result in poor user experience. It is

RETWEETS LIKES

Hector Martin

Follow

coding!"—then closed both the inventory and voucher vulnerability issues and blocked us from responding. Broadleaf considers the voucher vulnerability a feature. That is, the Broadleaf developers responded to a similar ticket, indicating that they would prefer to allow concurrent voucher usage on the grounds that failed checkouts due to voucher overuse would result in poor user experience. It is

6 14 20

Hector Martin
@marcan42

Follow

"It's not a vuln if I ban you from the issue tracker"

RETWEETS 12 LIKES 30

5:48 AM - 24 Mar 2017

2 12 30

RETWEETS LIKES

Developer Response

jberryman commented on Apr 12

FYI from **@TWarszawski** and Peter Bailis' recent paper, concerning the voucher vulnerability #4812 which has been closed and comments blocked:

> For example, in Magento [6], OpenCart [7], and Oscar [8], users can buy a single gift card, then spend it an unlimited number of times by concurrently issuing checkout requests.

The responses to these two tickets is baffling (and should be concerning to OpenCart users). I guess it will make for an entertaining slide at SIGMOD in a few weeks though.

Developer Response

jberryman commented on Apr 12

FYI from **@TWarszawski** and Peter Bailis' recent paper, concerning the voucher vulnerability #4812 which has been closed and comments blocked:

The responses to these two tickets is baffling (and should be concerning to OpenCart users). I guess it will make for an entertaining slide at SIGMOD in a few weeks though.

Related Work

- [Bailis et al. 2015] Study user level (Feral) invariants in Ruby on Rails applications
- [Jorwekar et al. 2007] Provide analysis methods for detecting potential anomalies in transaction programs for Snapshot Isolation
- [Fekete et al. 2009] Quantify Read Committed and Snapshot Isolation anomalies
- Our focus is on any non-serializable behavior in API based web applications as observed in practice

Conclusions

- Many popular eCommerce applications do not use transactions correctly
- 2AD: a new, cross-language analysis tool to check for potential anomalies
- Using 2AD, we find 22 new vulnerabilities due to incorrect transaction usage affecting up to 2M+ eCommerce sites

Conclusions

- Many popular eCommerce applications do not use transactions correctly
- 2AD: a new, cross-language analysis tool to check for potential anomalies
- Using 2AD, we find 22 new vulnerabilities due to incorrect transaction usage affecting up to 2M+ eCommerce sites

Thanks!

twarszaw@stanford.edu

https://github.com/stanford-futuredata/acidrain