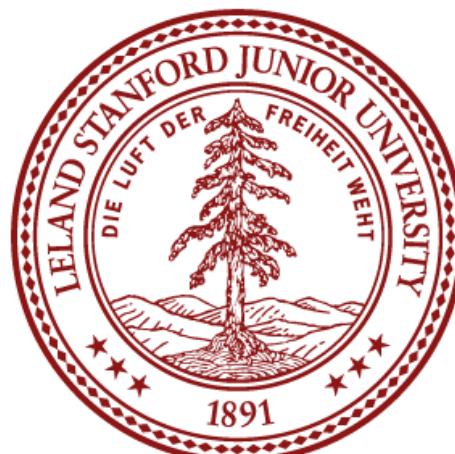


ACIDRain: Concurrency-Related Attacks on Database-Backed Web Applications

Todd Warszawski, Peter Bailis
Stanford University



A: Atomicity

C: Consistency

I: Isolation

D: Durability

Problems with concurrent operations:

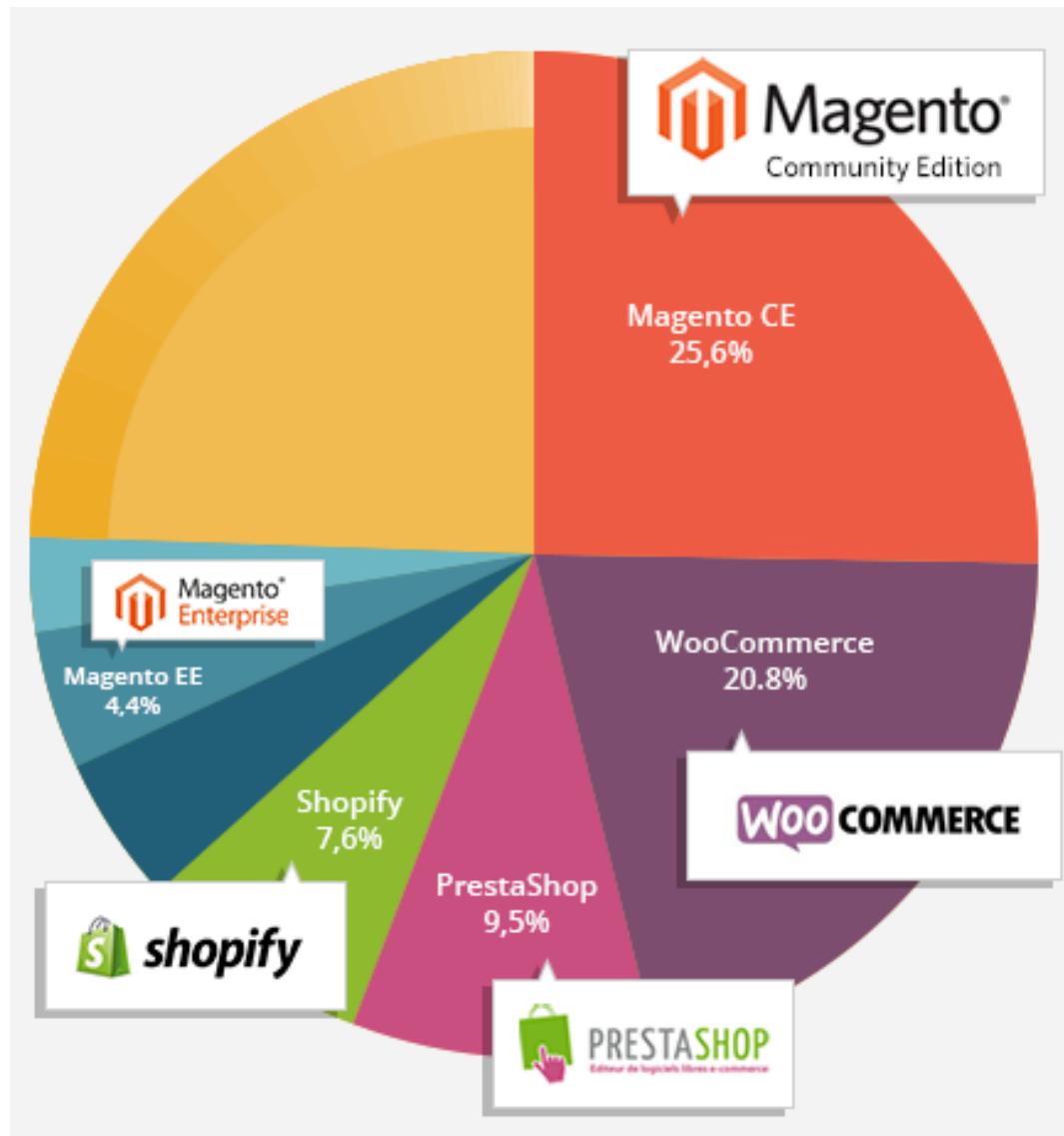
1. lost update
2. dirty read
3. non-repeatable read
4. phantom row

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

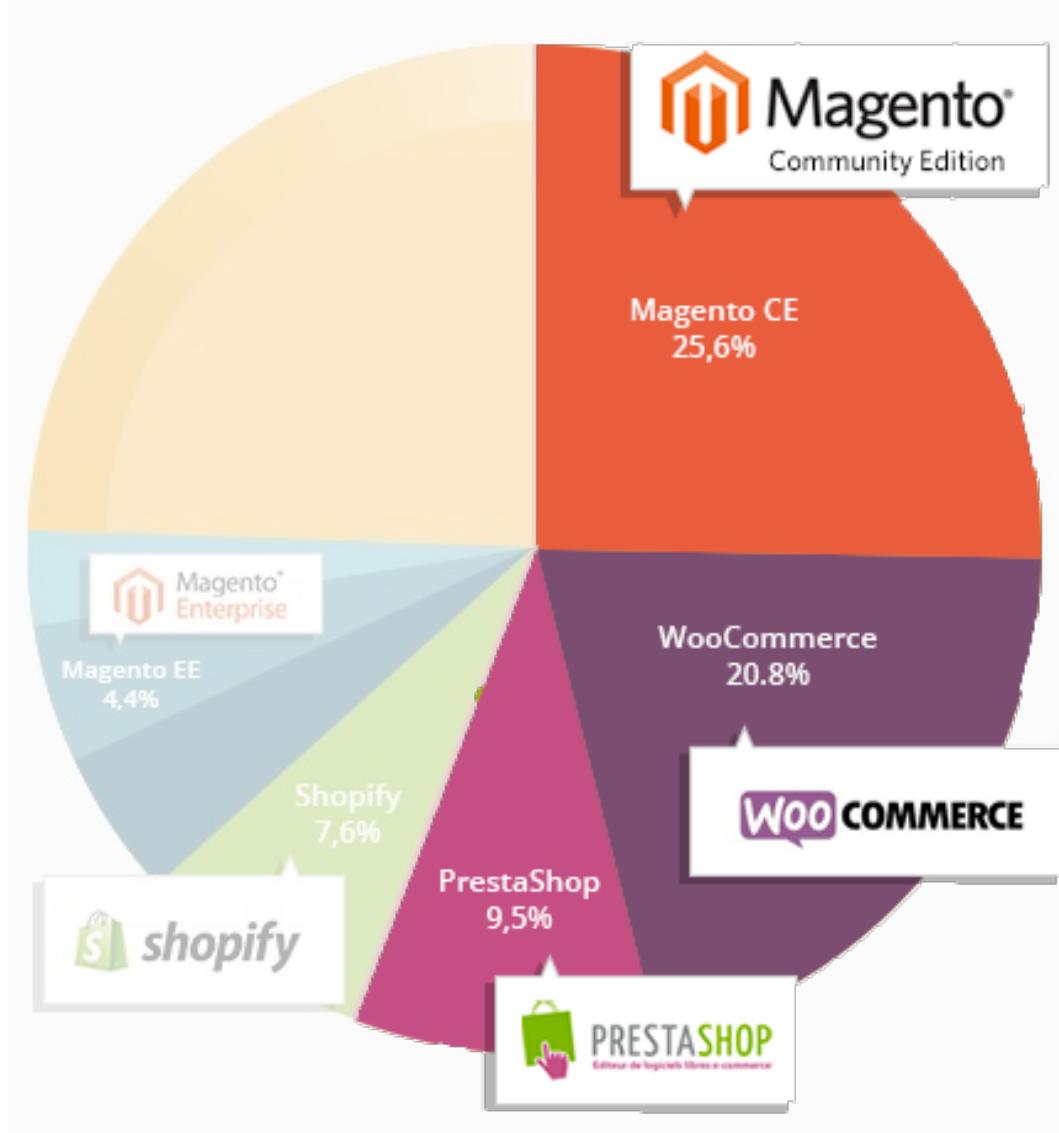
Explore Real World Transaction Usage

- 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



The Container Store®

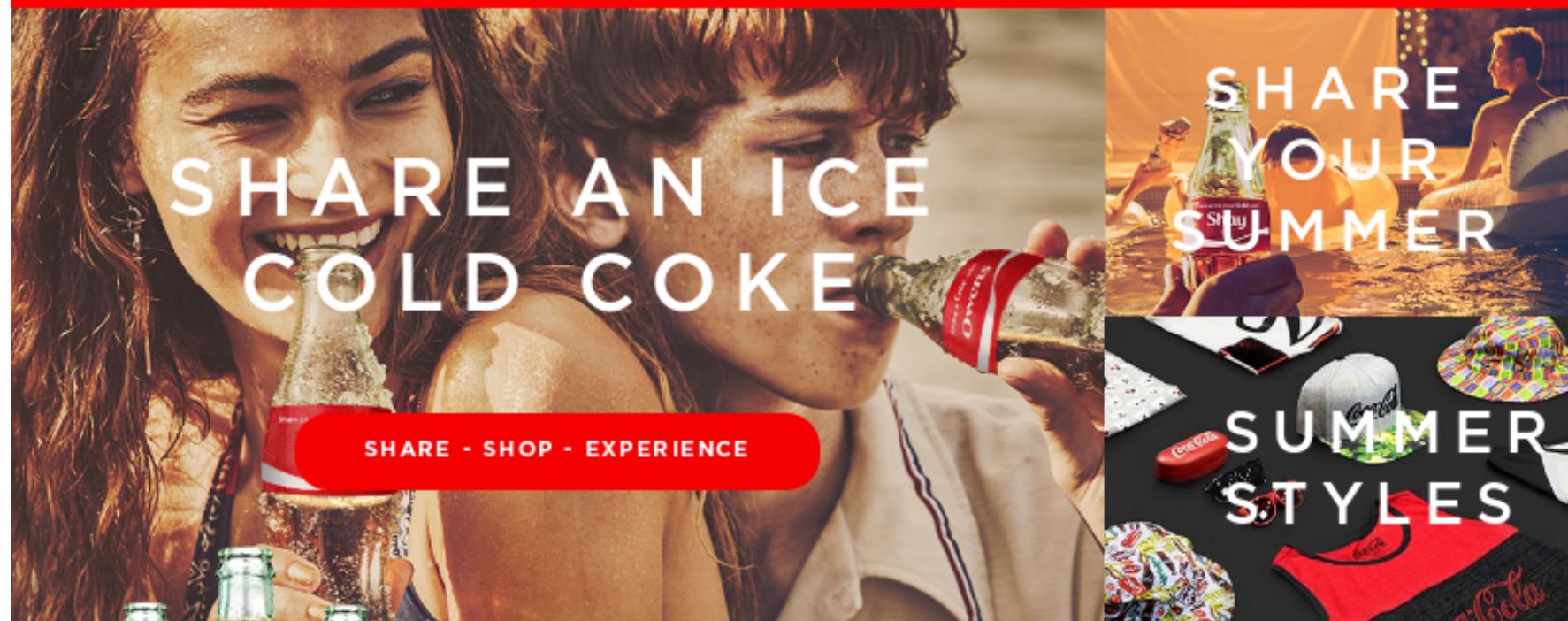


Coca-Cola



[SHOP](#)[SUMMER](#)[CUSTOMIZE](#) Search entire site

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

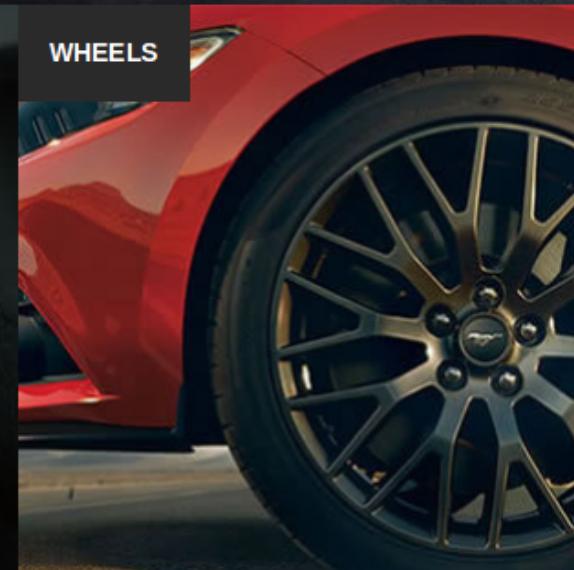
Find accessories that fit your Ford vehicle



INTERIOR



WHEELS



BED PRODUCTS



EXTERIOR



START BY SELECTING Your Vehicle

Model

Year

Go

ELECTRONICS





FREE SHIPPING
WHEN YOU SPEND \$75 OR MORE* ▾

Search now



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



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

EVERYTHING
kitchen



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 ultimately the Bitcoin community.

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

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

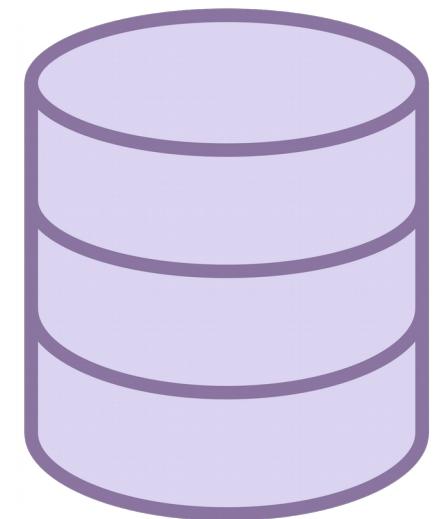
Overview

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

Problem Setup: Attacking Websites

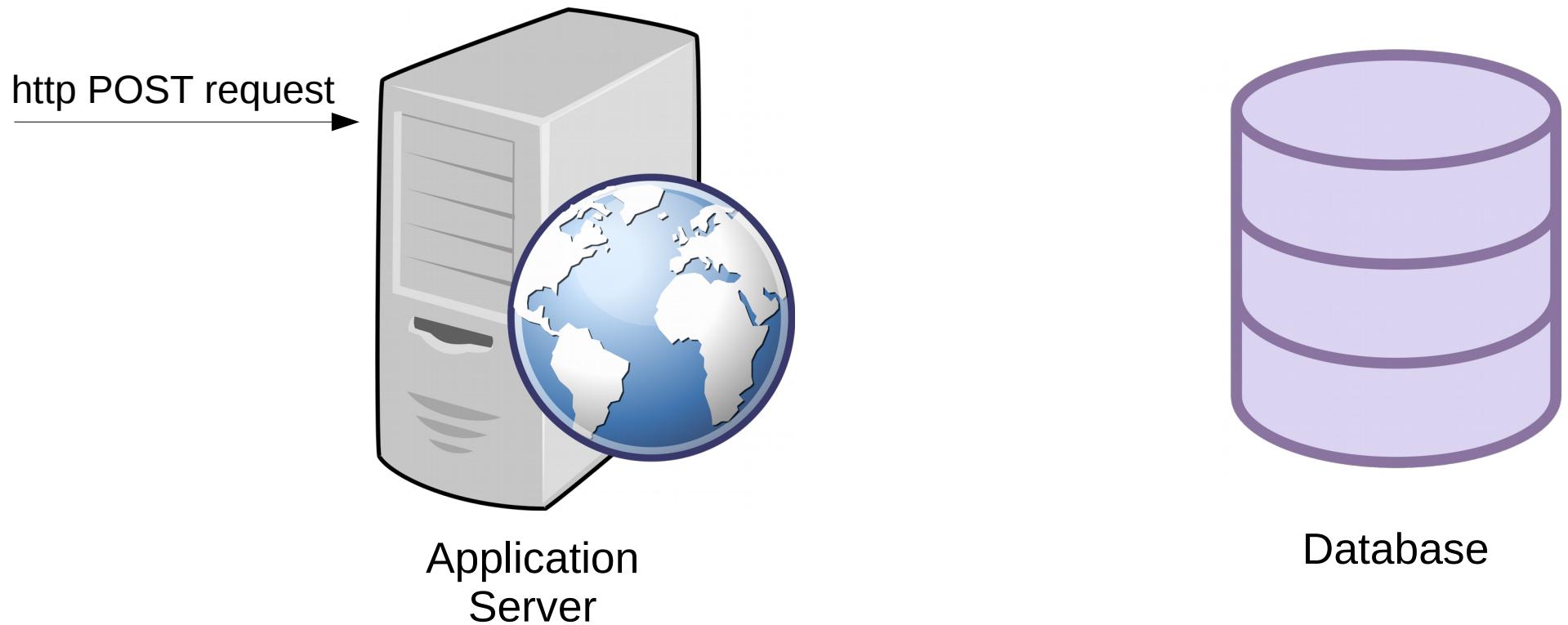


Application
Server

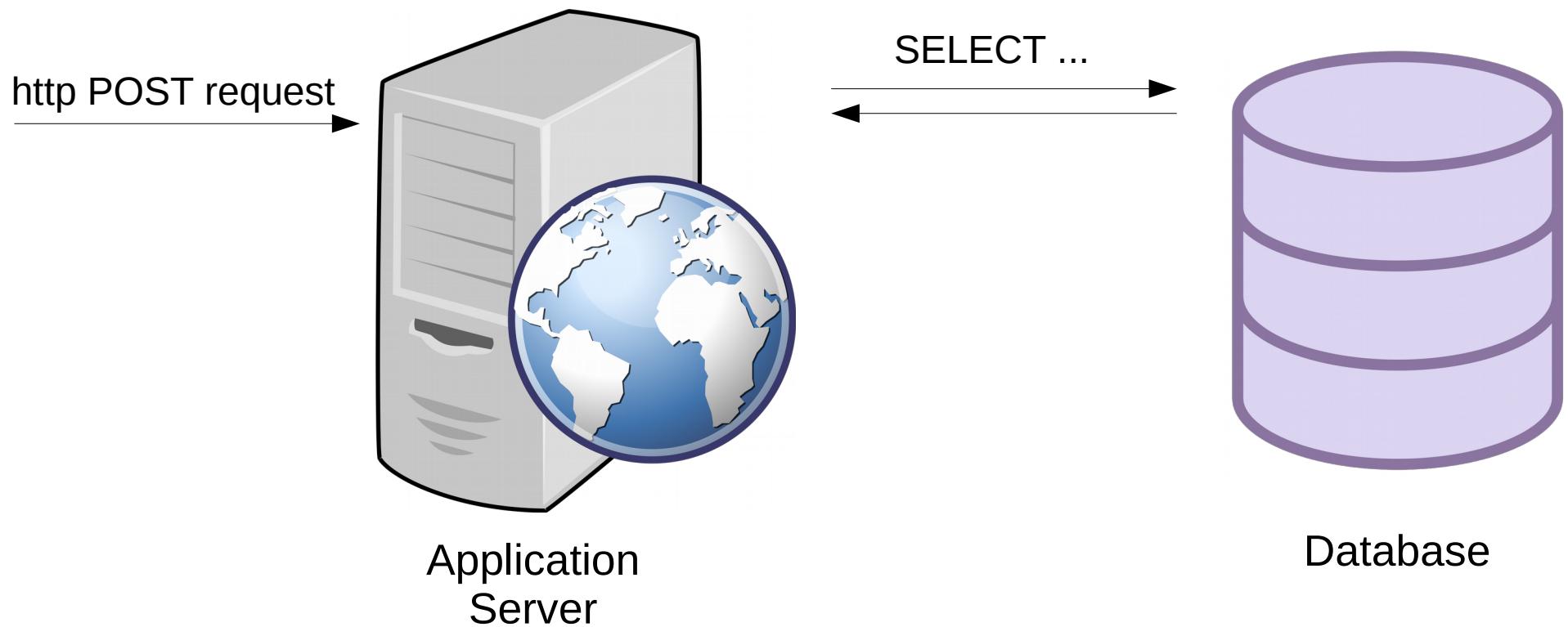


Database

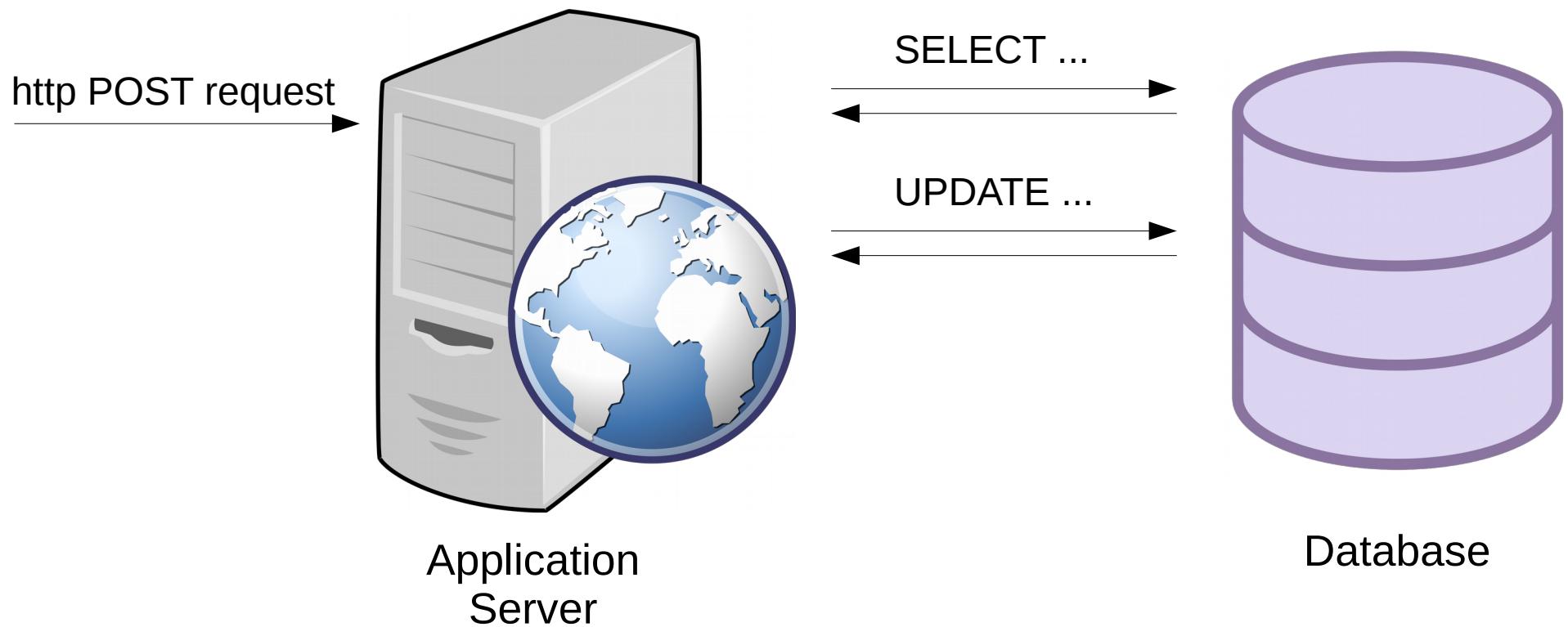
Problem Setup: Attacking Websites



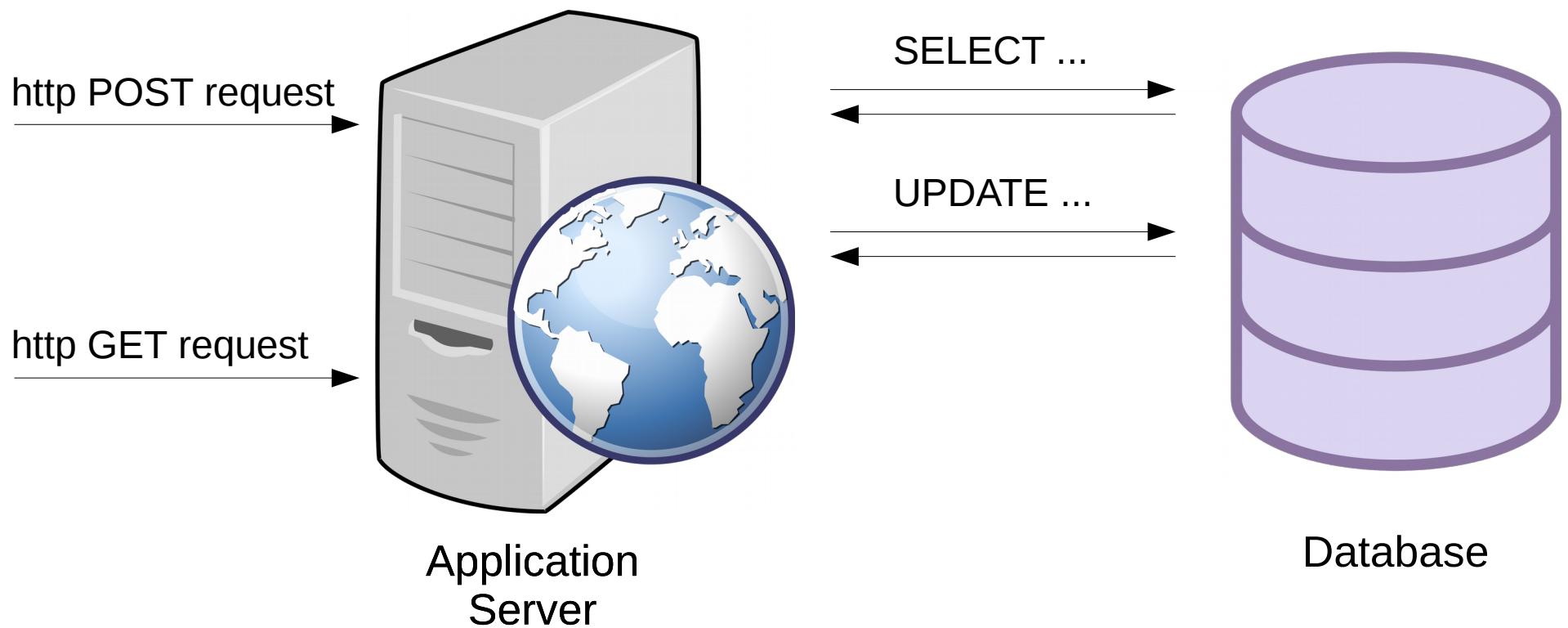
Problem Setup: Attacking Websites



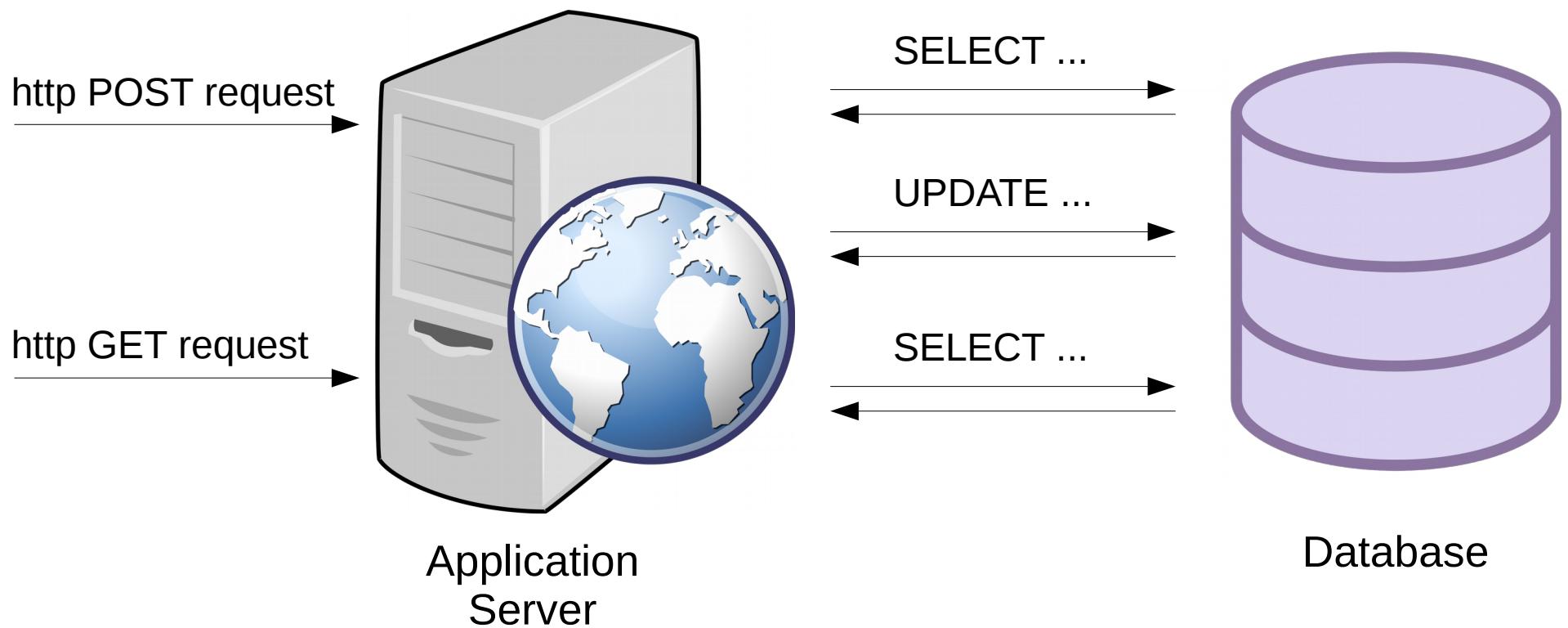
Problem Setup: Attacking Websites



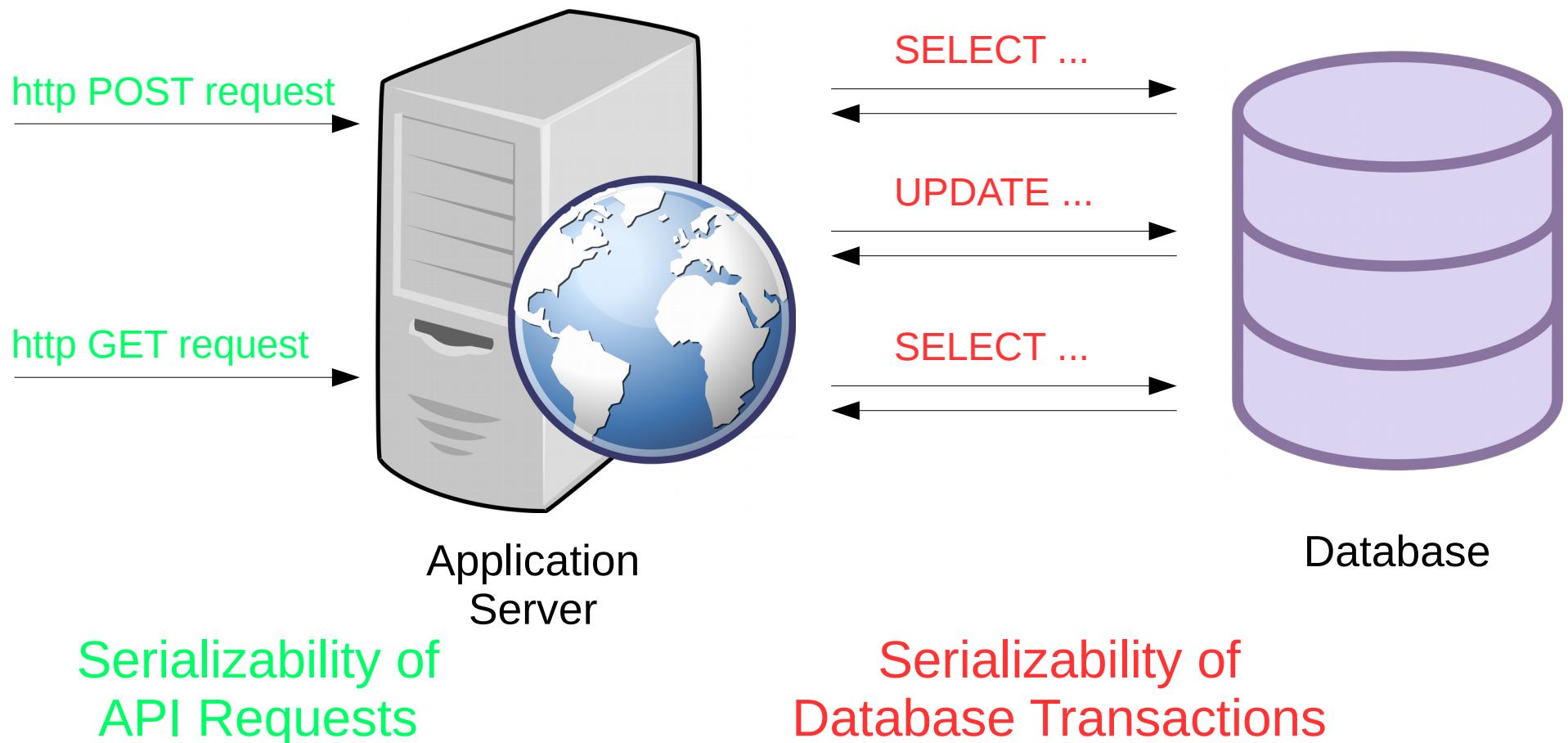
Problem Setup: Attacking Websites



Problem Setup: Attacking Websites



Problem Setup: Attacking Websites



Non-Transactional Implementation

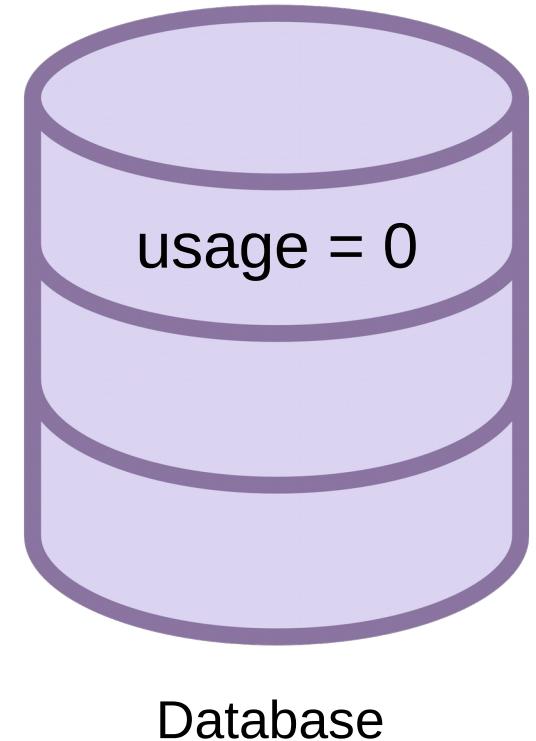
```
def checkVoucher(code):
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
```

```
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)
```

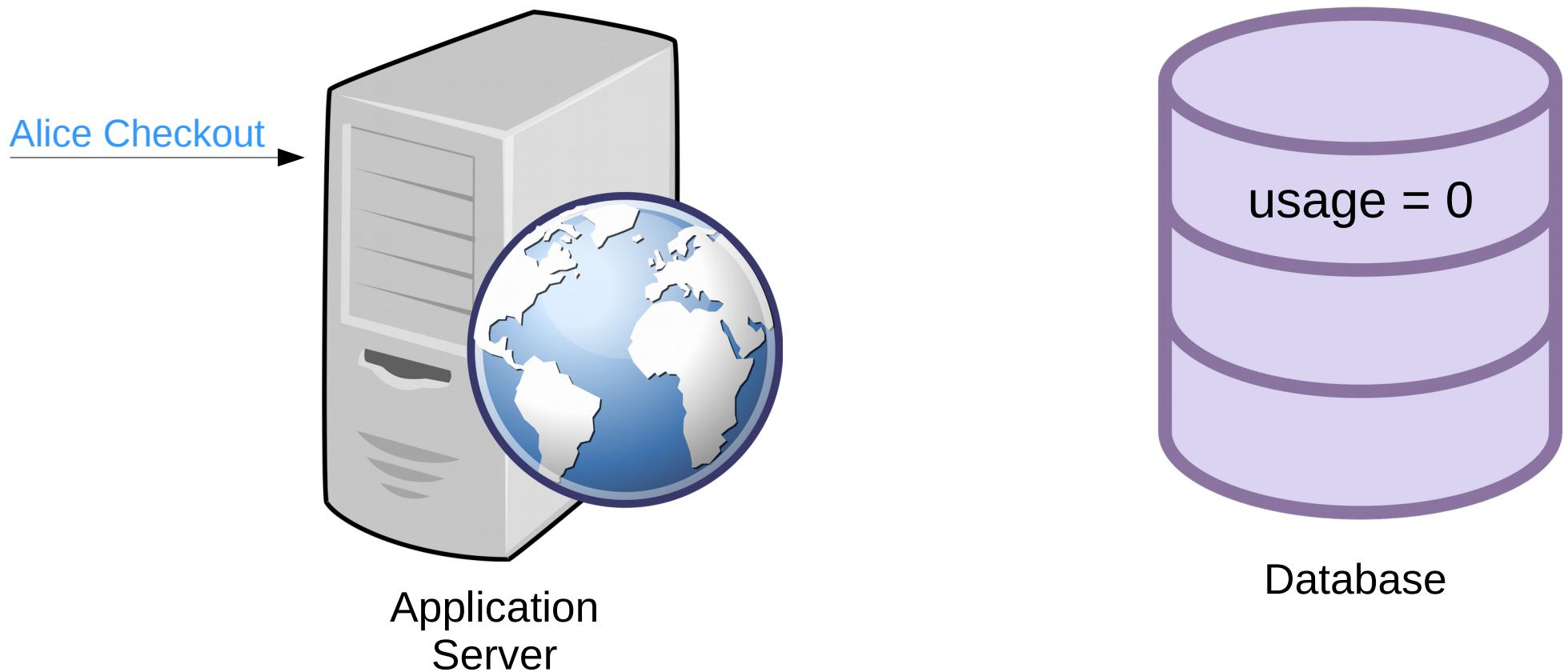
```
SELECT usage FROM voucher WHERE code = HNUHY  
UPDATE voucher SET usage = 1 WHERE code = HNUHY
```



Non-Transactional Implementation

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

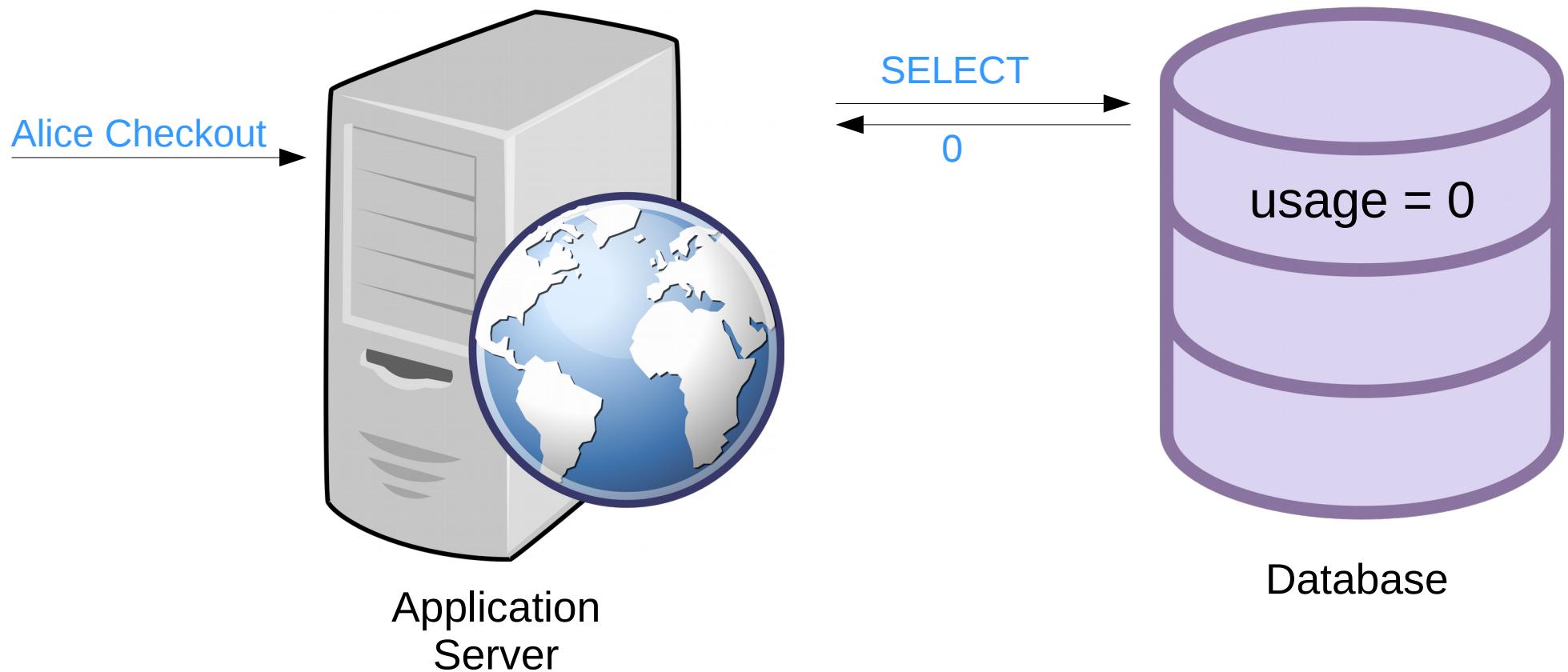
```
SELECT usage FROM voucher WHERE code = HNUHY  
UPDATE voucher SET usage = 1 WHERE code = HNUHY
```



Non-Transactional Implementation

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

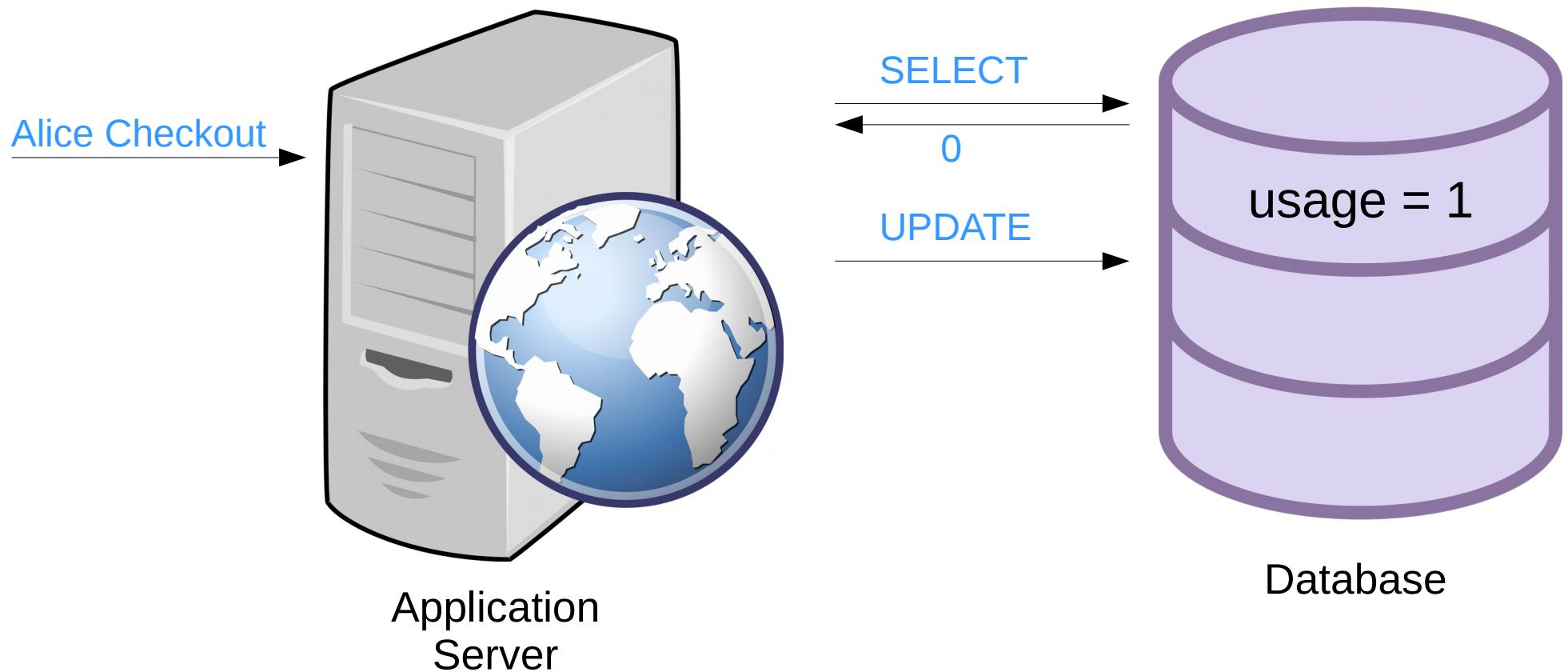
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY



Non-Transactional Implementation

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

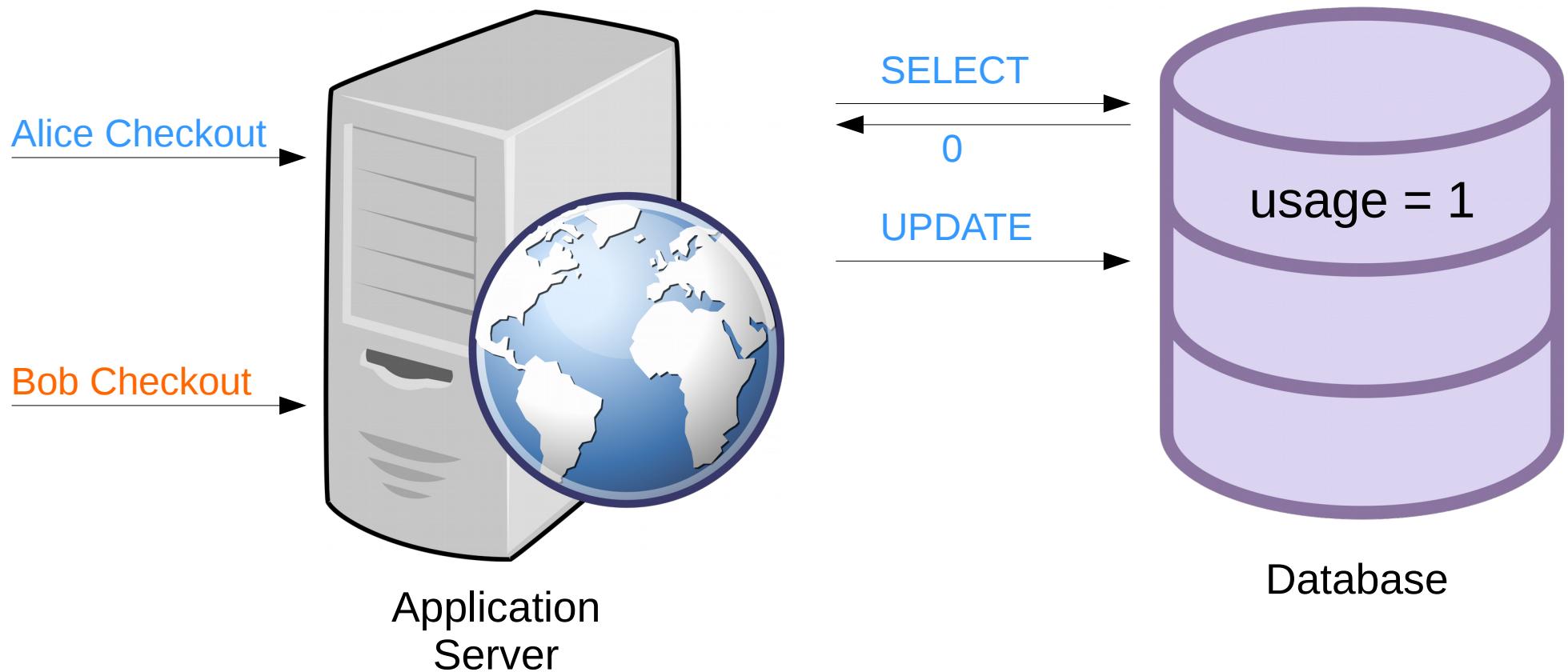
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY



Non-Transactional Implementation

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

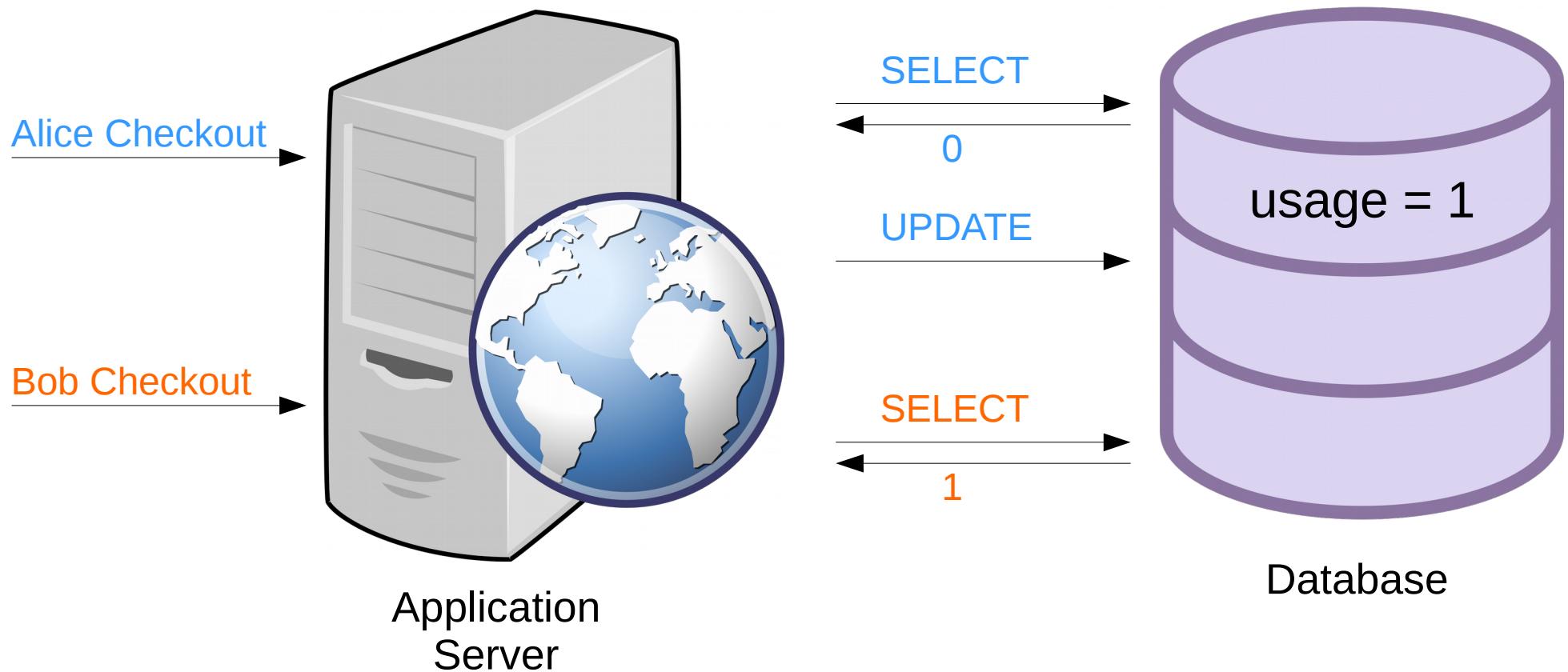
```
SELECT usage FROM voucher WHERE code = HNUHY  
UPDATE voucher SET usage = 1 WHERE code = HNUHY
```



Non-Transactional Implementation

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

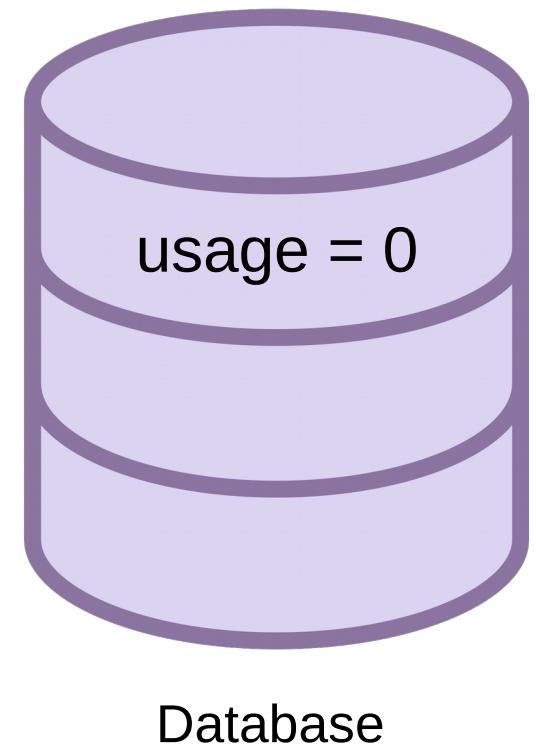
```
SELECT usage FROM voucher WHERE code = HNUHY  
UPDATE voucher SET usage = 1 WHERE code = HNUHY
```



Non-Transactional Implementation

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

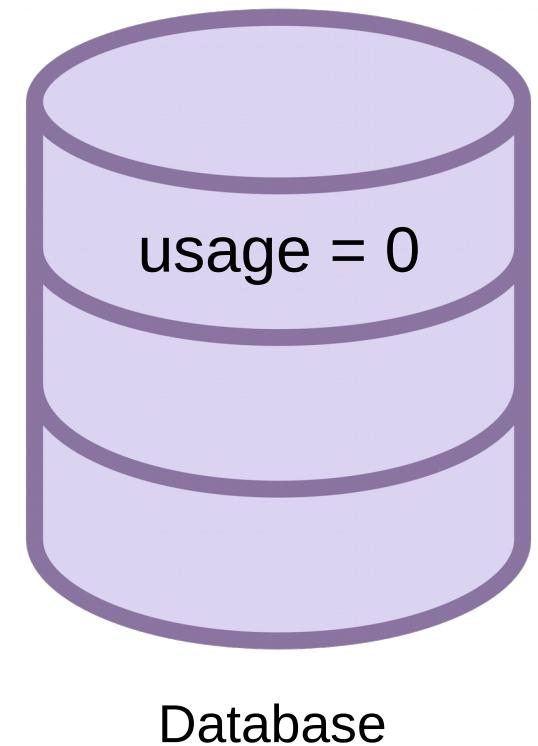
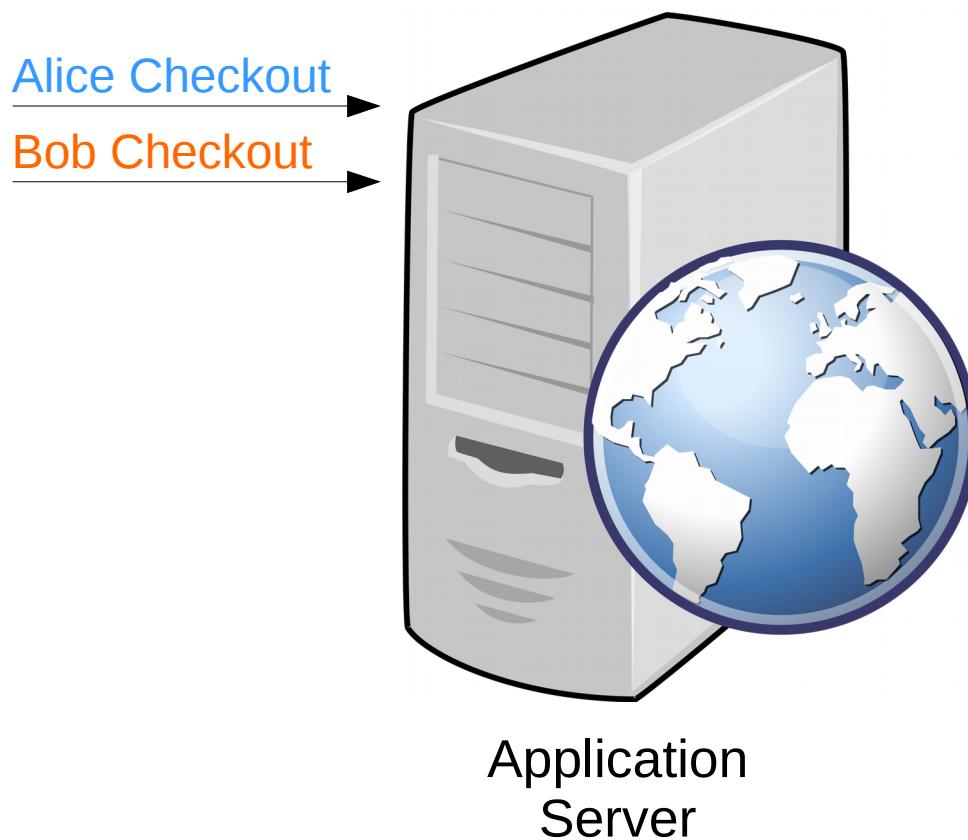
```
SELECT usage FROM voucher WHERE code = HNUHY  
UPDATE voucher SET usage = 1 WHERE code = HNUHY
```



Non-Transactional Implementation

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

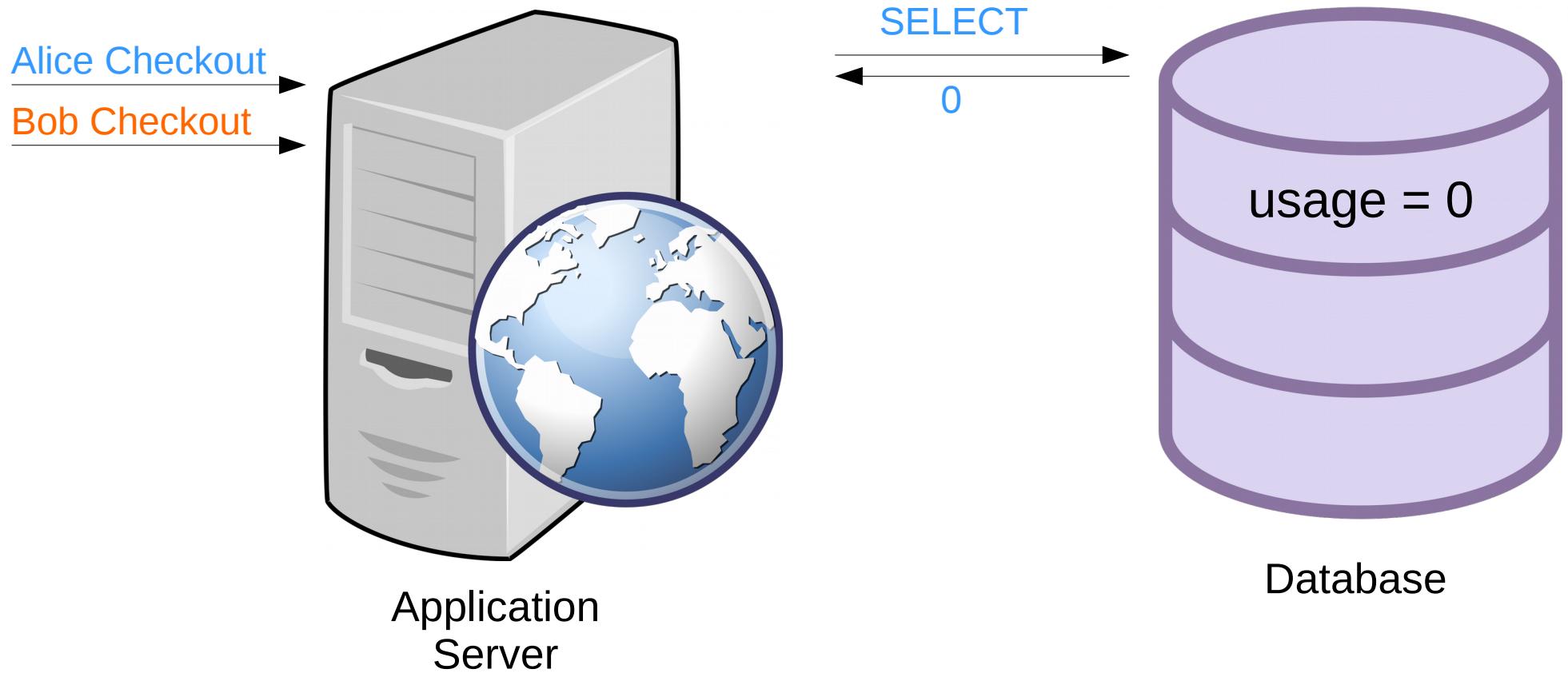
```
SELECT usage FROM voucher WHERE code = HNUHY  
UPDATE voucher SET usage = 1 WHERE code = HNUHY
```



Non-Transactional Implementation

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

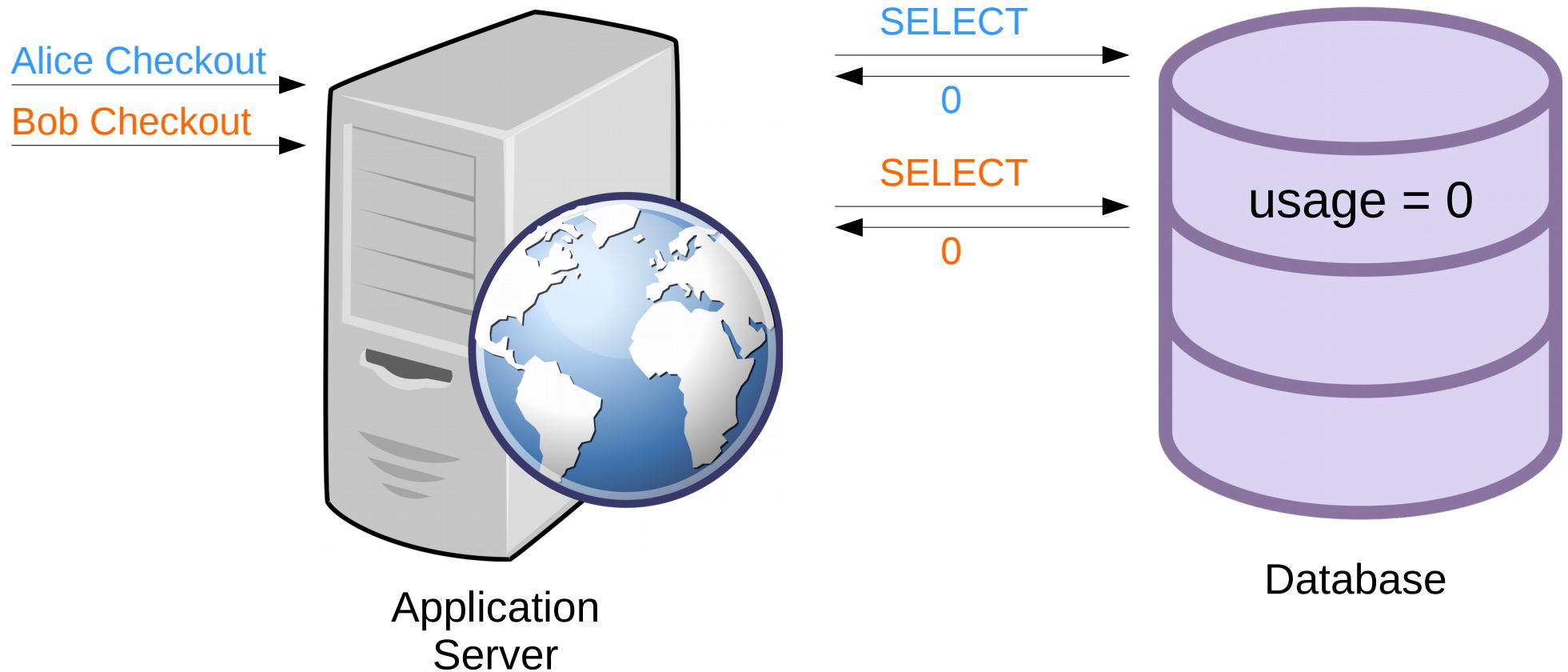
```
SELECT usage FROM voucher WHERE code = HNUHY  
UPDATE voucher SET usage = 1 WHERE code = HNUHY
```



Non-Transactional Implementation

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

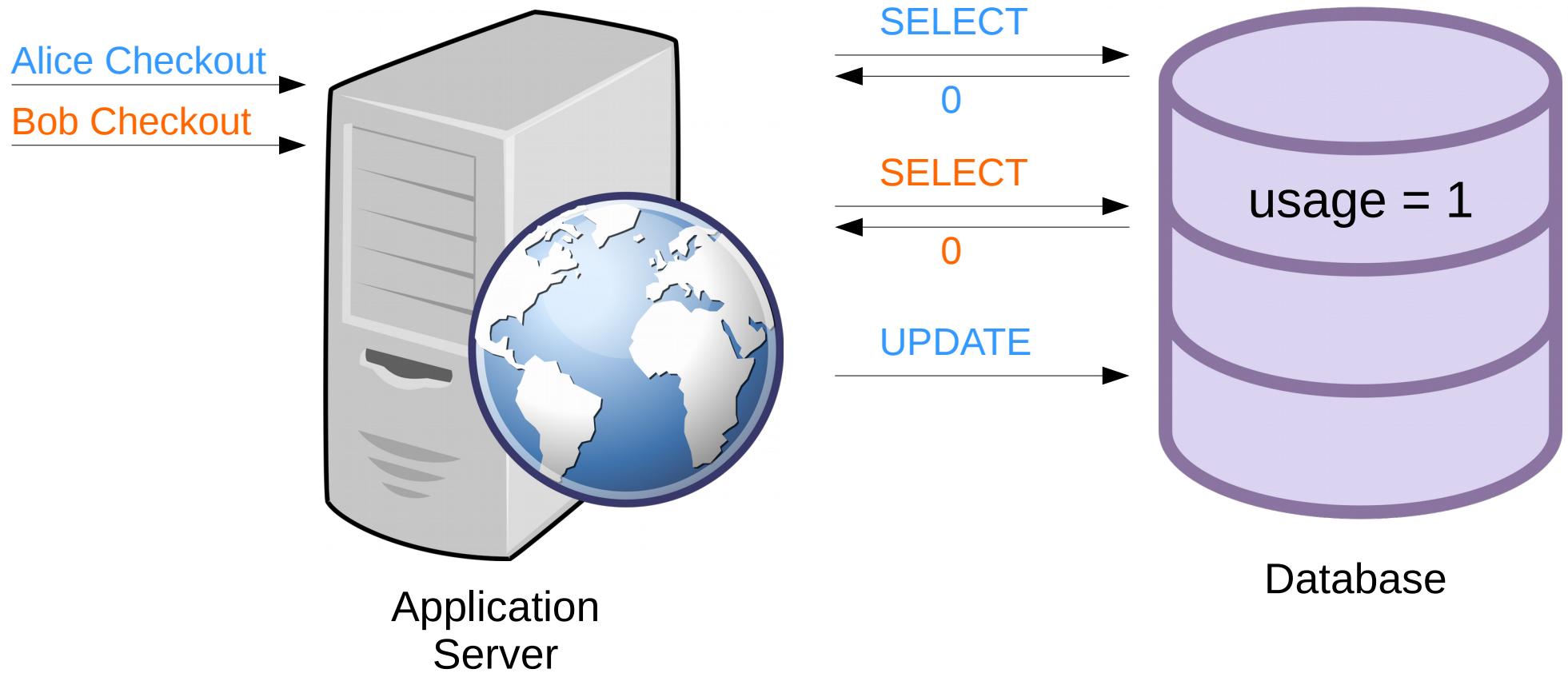
```
SELECT usage FROM voucher WHERE code = HNUHY  
UPDATE voucher SET usage = 1 WHERE code = HNUHY
```



Non-Transactional Implementation

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

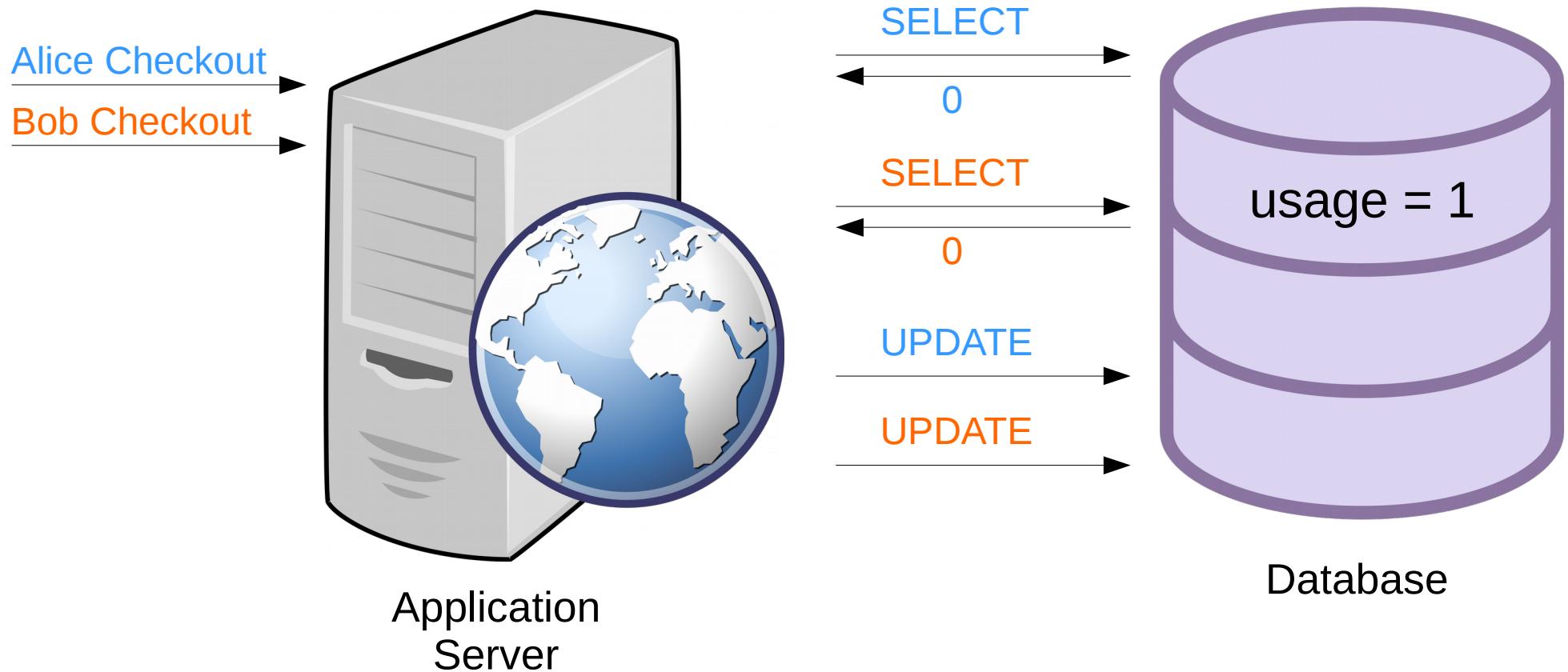
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY



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
```

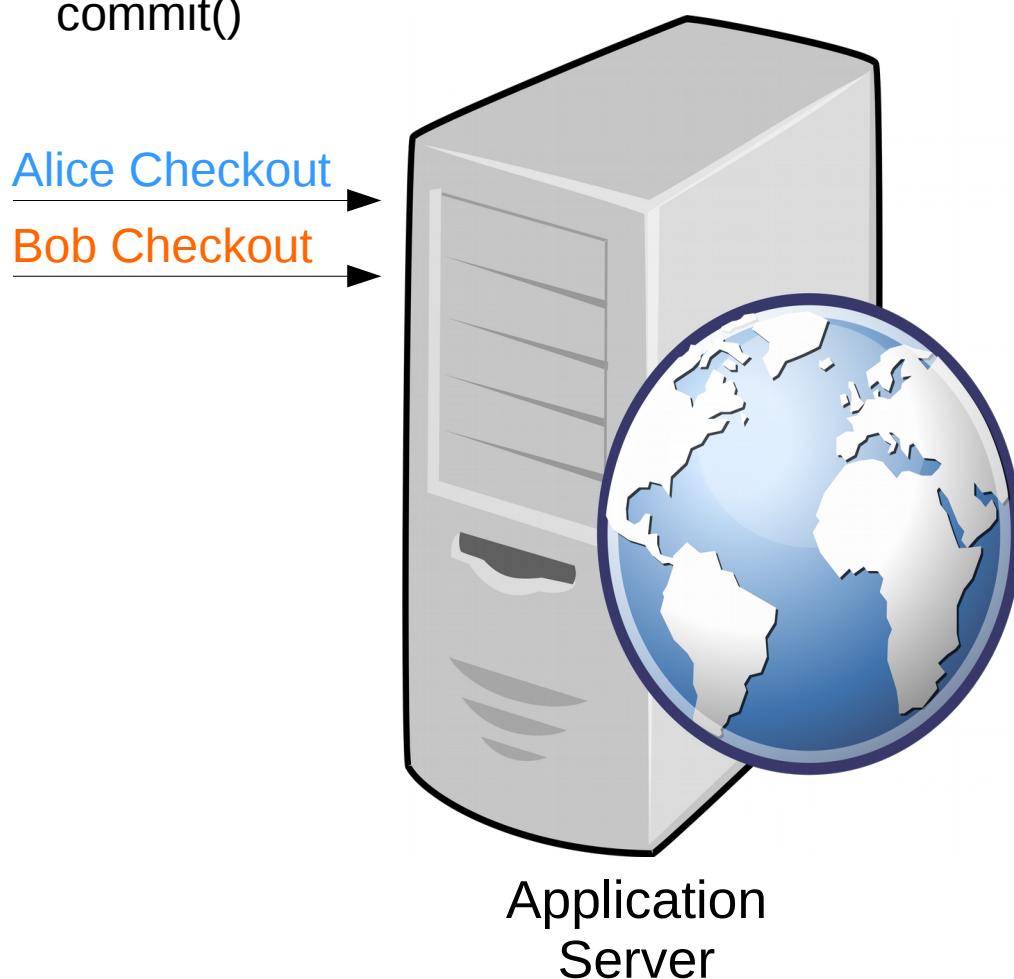


Database

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
```

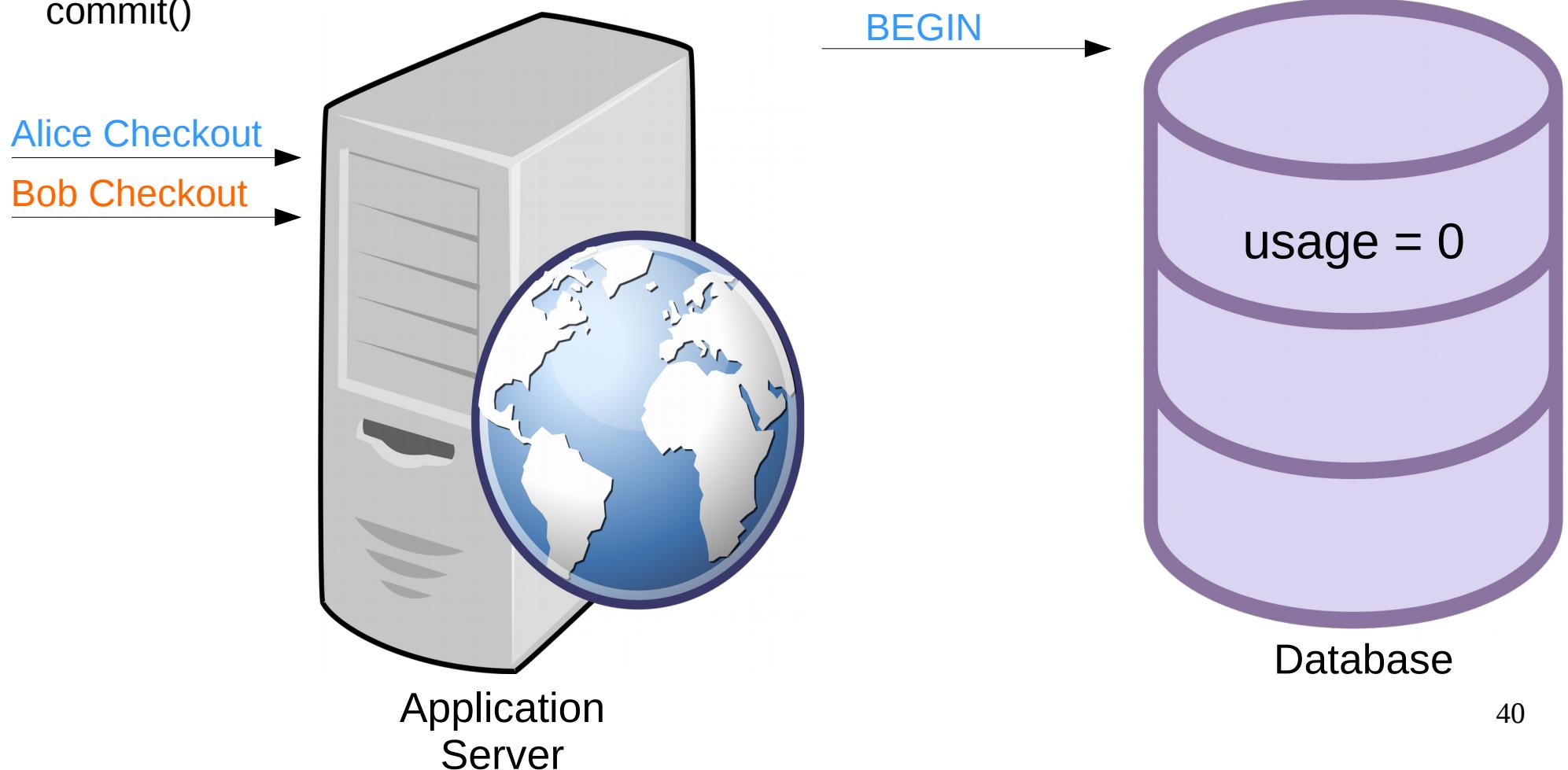


Database

Transactional Implementation

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

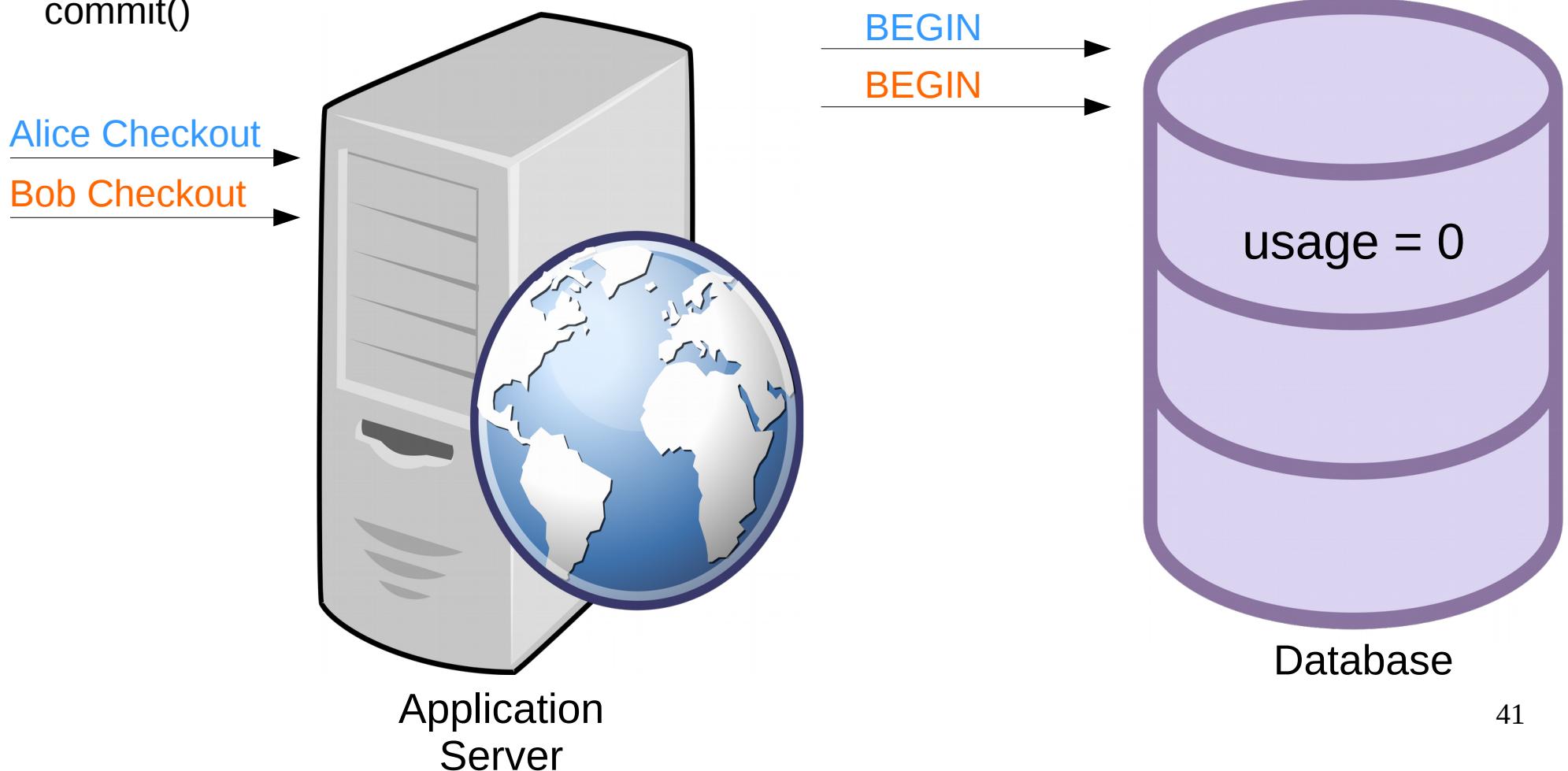
```
BEGIN TRANSACTION  
SELECT usage FROM voucher WHERE code = HNUHY  
UPDATE voucher SET usage = 1 WHERE code = HNUHY  
COMMIT
```



Transactional Implementation

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

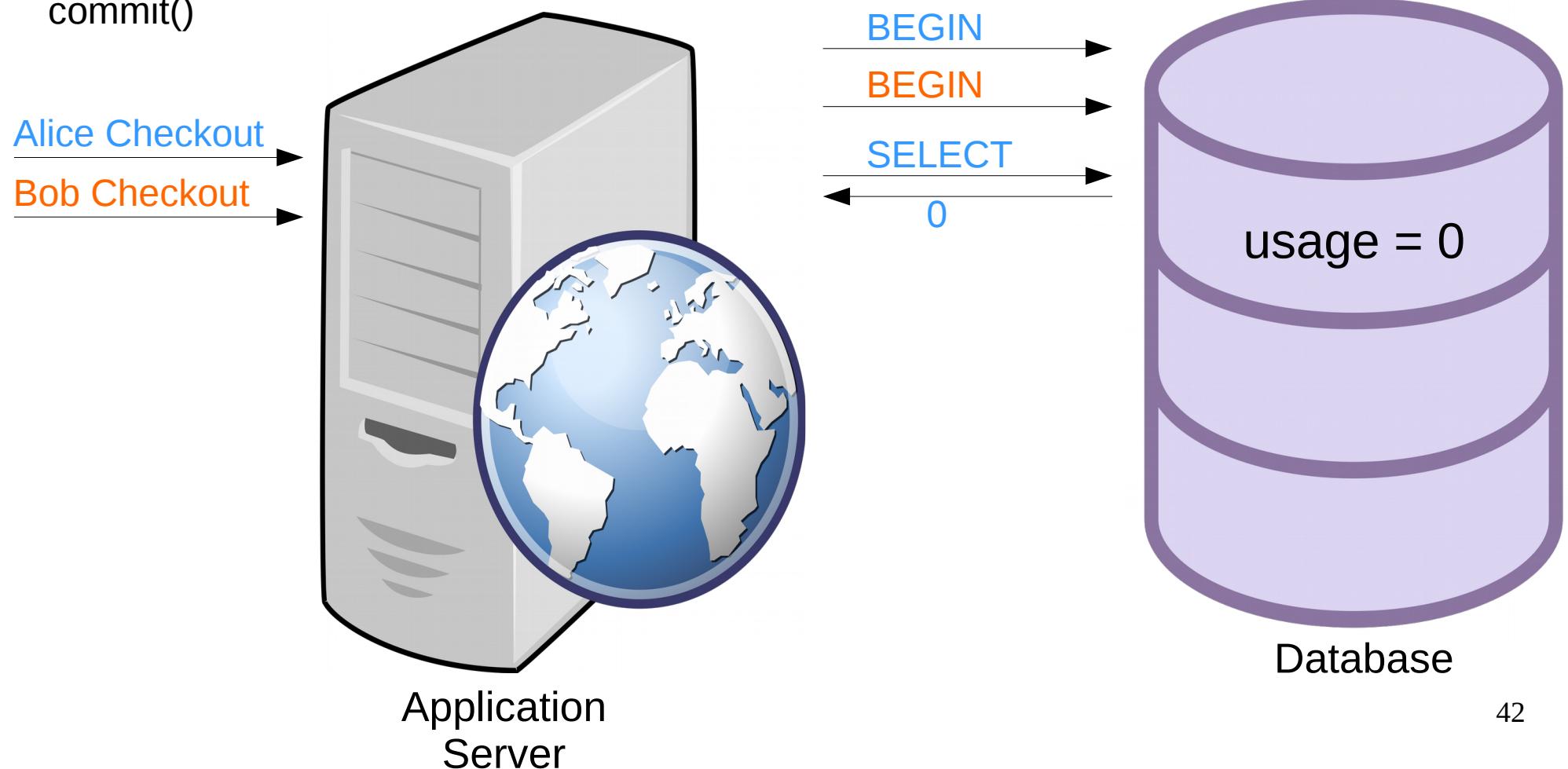
```
BEGIN TRANSACTION  
SELECT usage FROM voucher WHERE code = HNUHY  
UPDATE voucher SET usage = 1 WHERE code = HNUHY  
COMMIT
```



Transactional Implementation

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

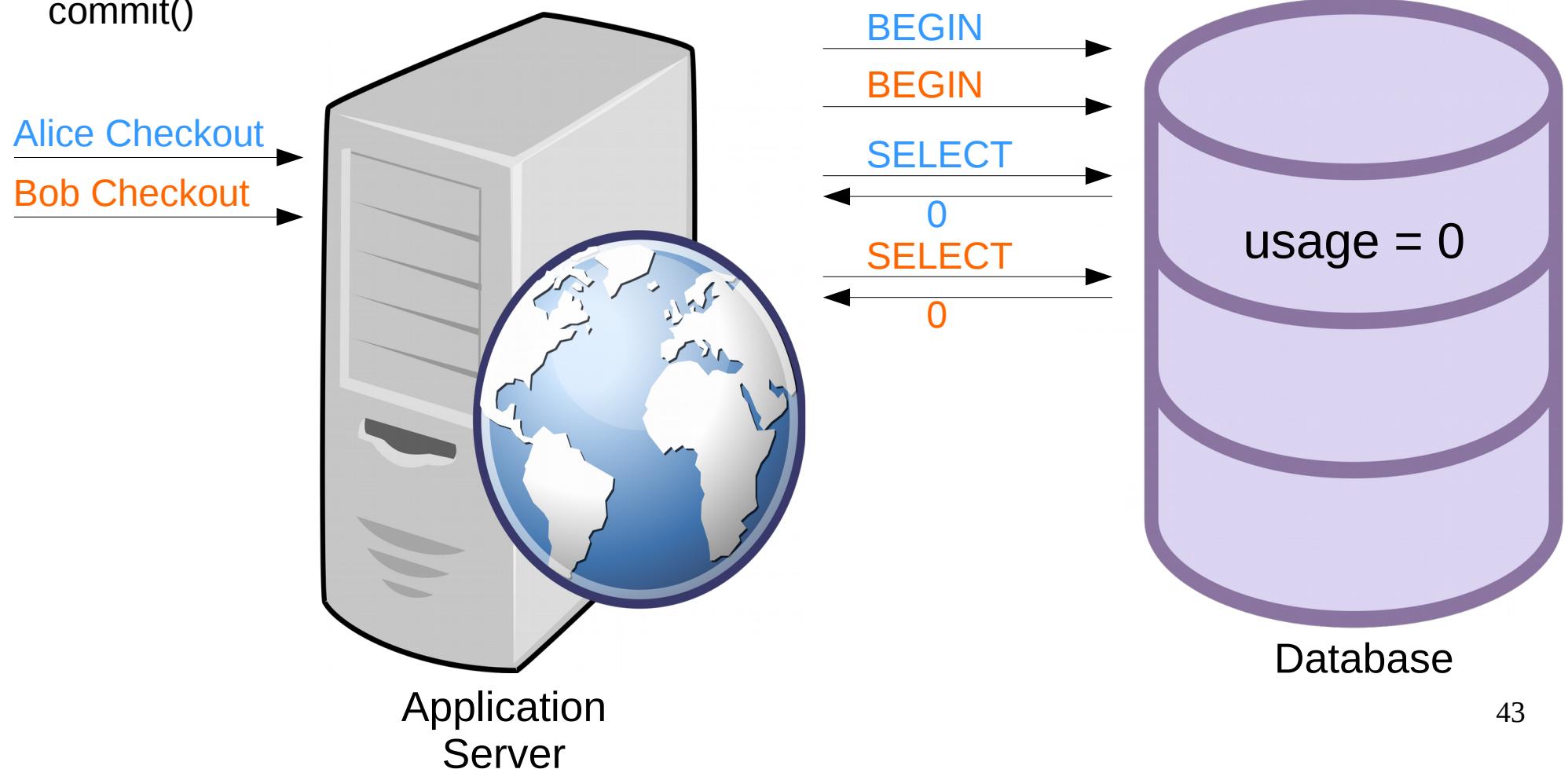
```
BEGIN TRANSACTION  
SELECT usage FROM voucher WHERE code = HNUHY  
UPDATE voucher SET usage = 1 WHERE code = HNUHY  
COMMIT
```



Transactional Implementation

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

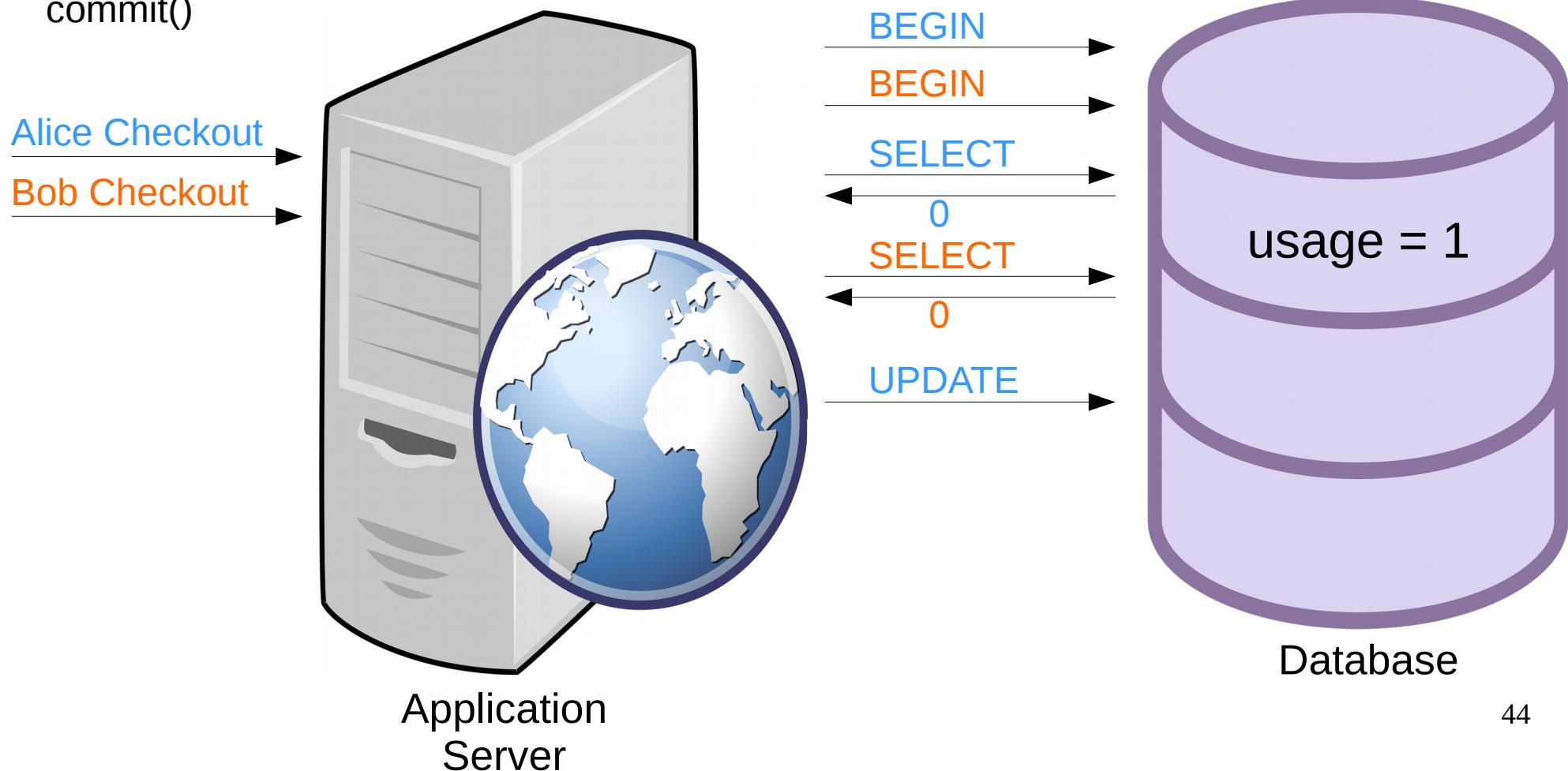
```
BEGIN TRANSACTION  
SELECT usage FROM voucher WHERE code = HNUHY  
UPDATE voucher SET usage = 1 WHERE code = HNUHY  
COMMIT
```



Transactional Implementation

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

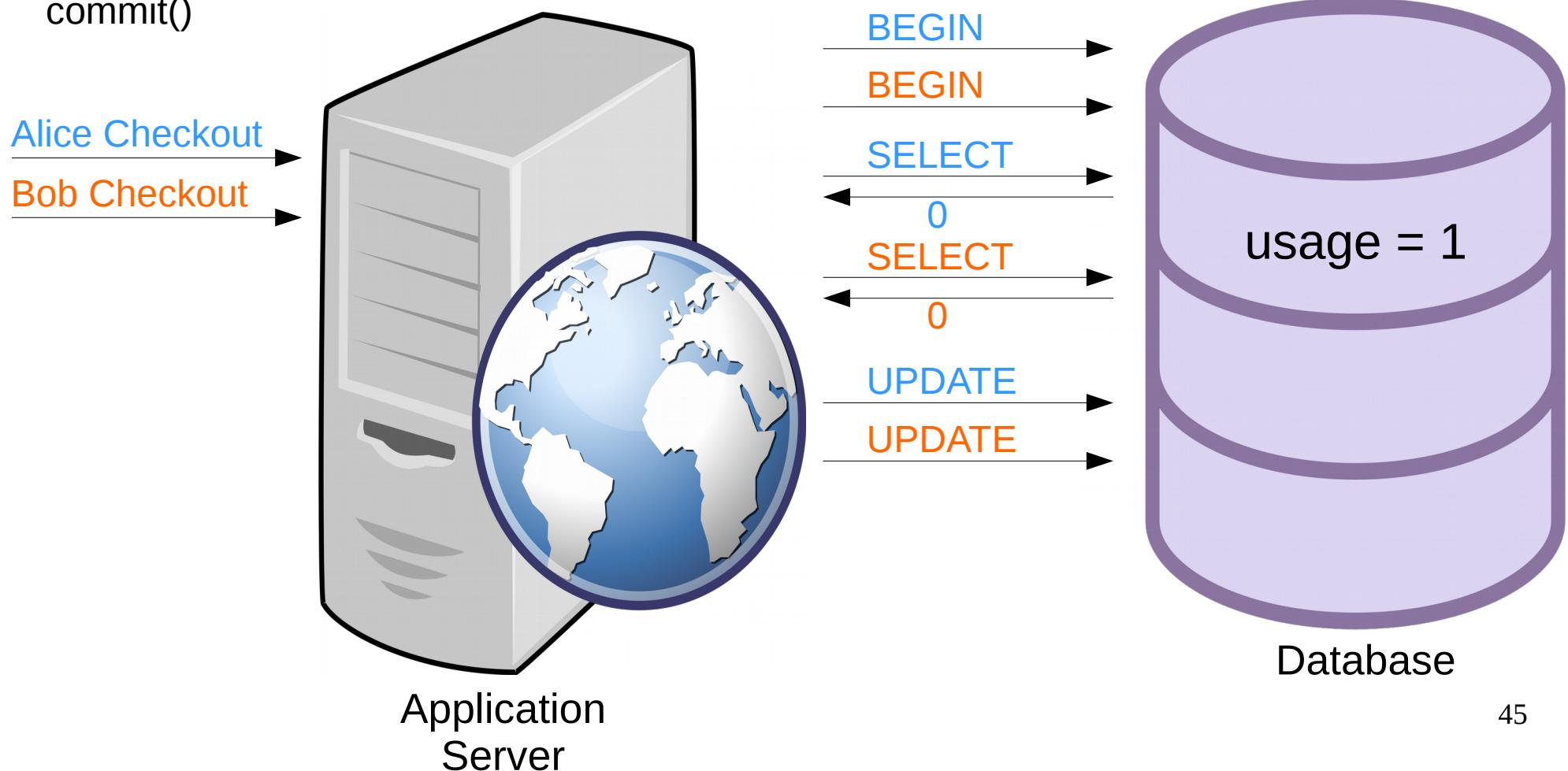
```
BEGIN TRANSACTION  
SELECT usage FROM voucher WHERE code = HNUHY  
UPDATE voucher SET usage = 1 WHERE code = HNUHY  
COMMIT
```



Transactional Implementation

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

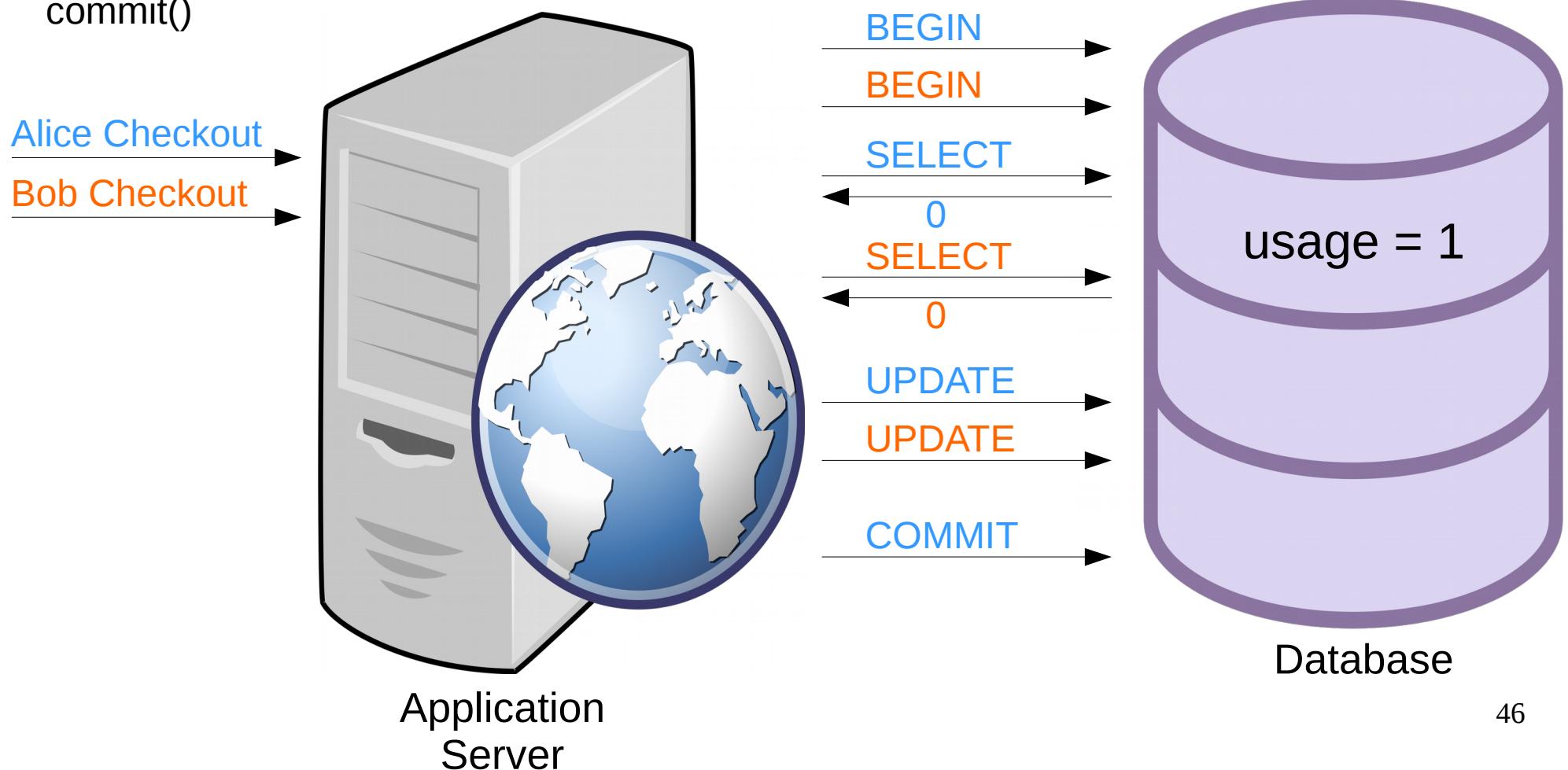
```
BEGIN TRANSACTION  
SELECT usage FROM voucher WHERE code = HNUHY  
UPDATE voucher SET usage = 1 WHERE code = HNUHY  
COMMIT
```



Transactional Implementation

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

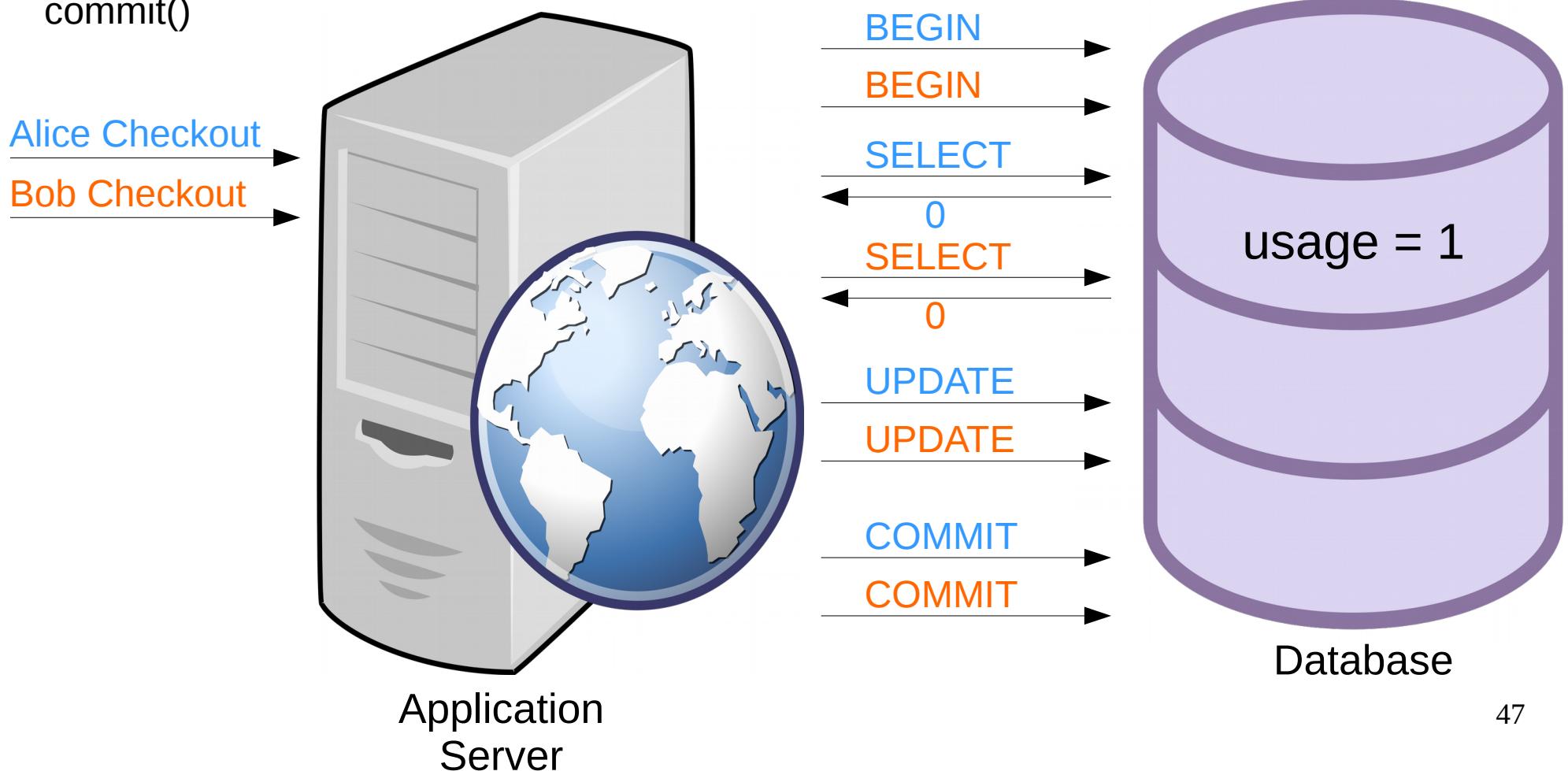
```
BEGIN TRANSACTION  
SELECT usage FROM voucher WHERE code = HNUHY  
UPDATE voucher SET usage = 1 WHERE code = HNUHY  
COMMIT
```



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
```



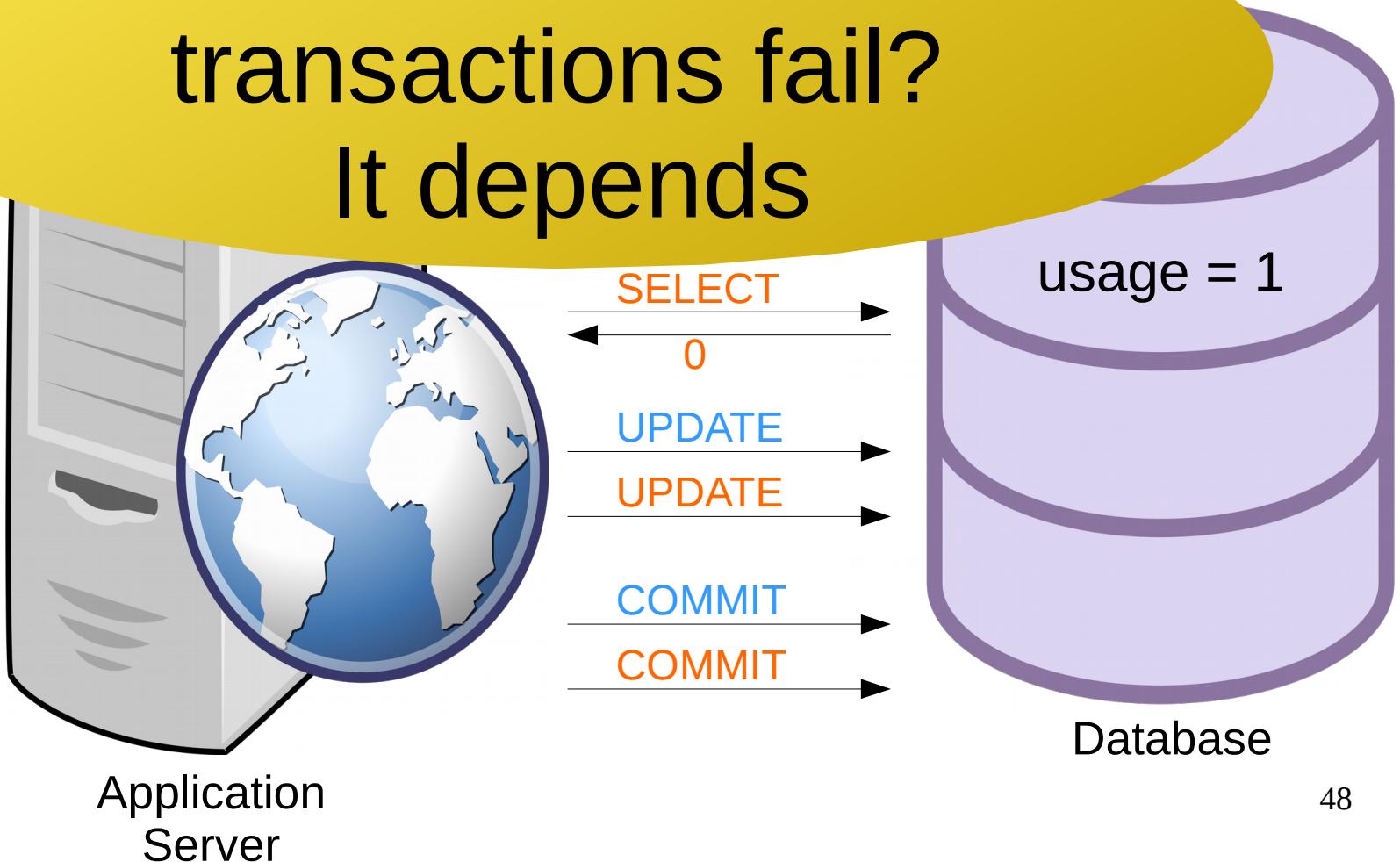
Transactional Implementation

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

Alice Checkout
Bob Checkout

```
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

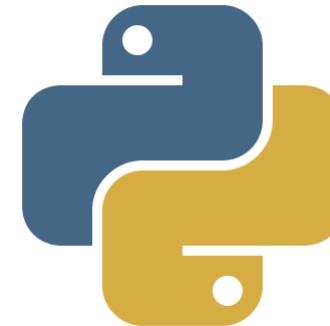
```
def checkVoucher(code):
    usage = readUsage(code)
    if (usage == 0):
        markUsed(code)
```

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)

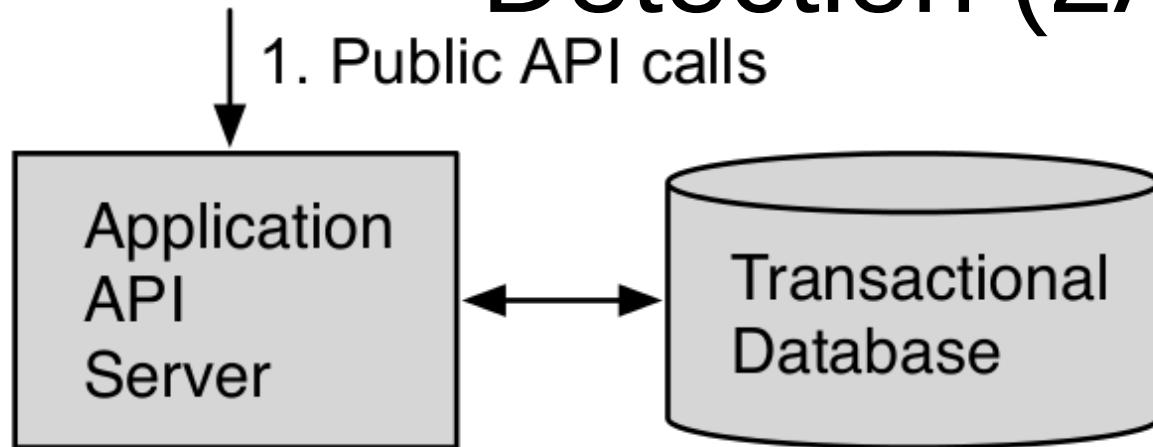
1. Collect (possibly serial) logs from database
2. Build compact representation of history (abstract history graph)
3. Search abstract history for cycles to generate possible anomalous API calls

Approach: Abstract Anomaly Detection (2AD)

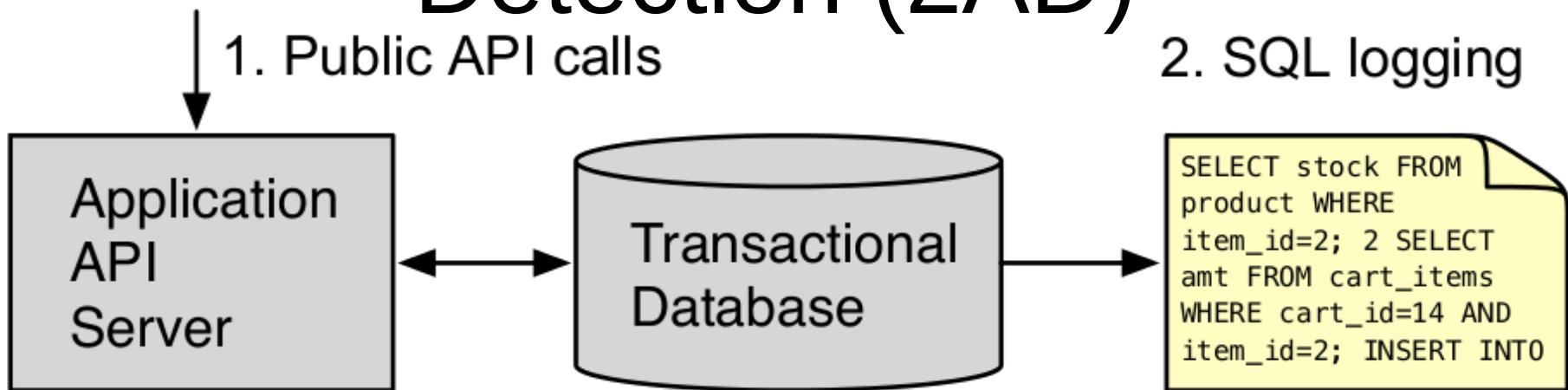


Application
API
Server

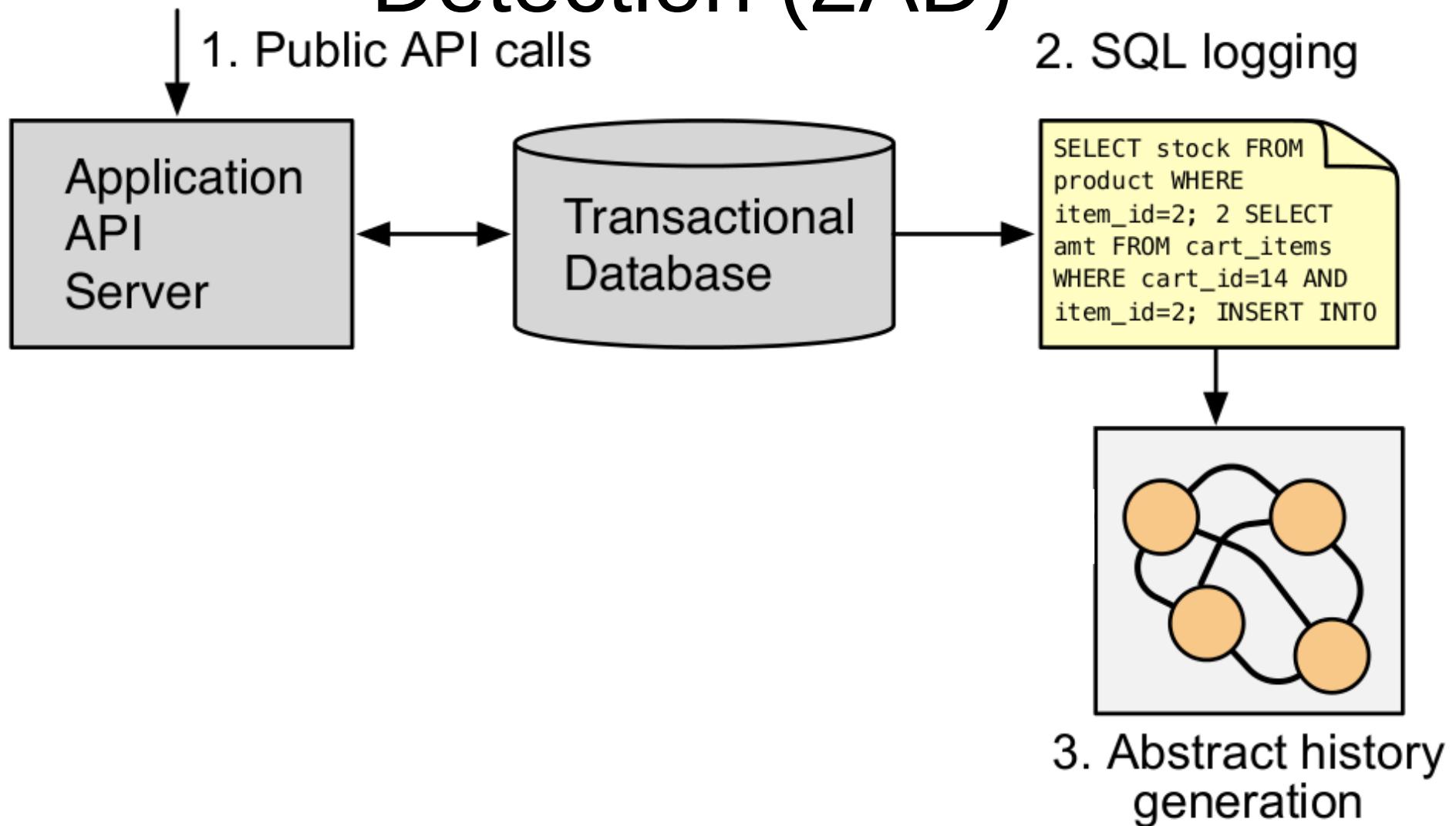
Approach: Abstract Anomaly Detection (2AD)



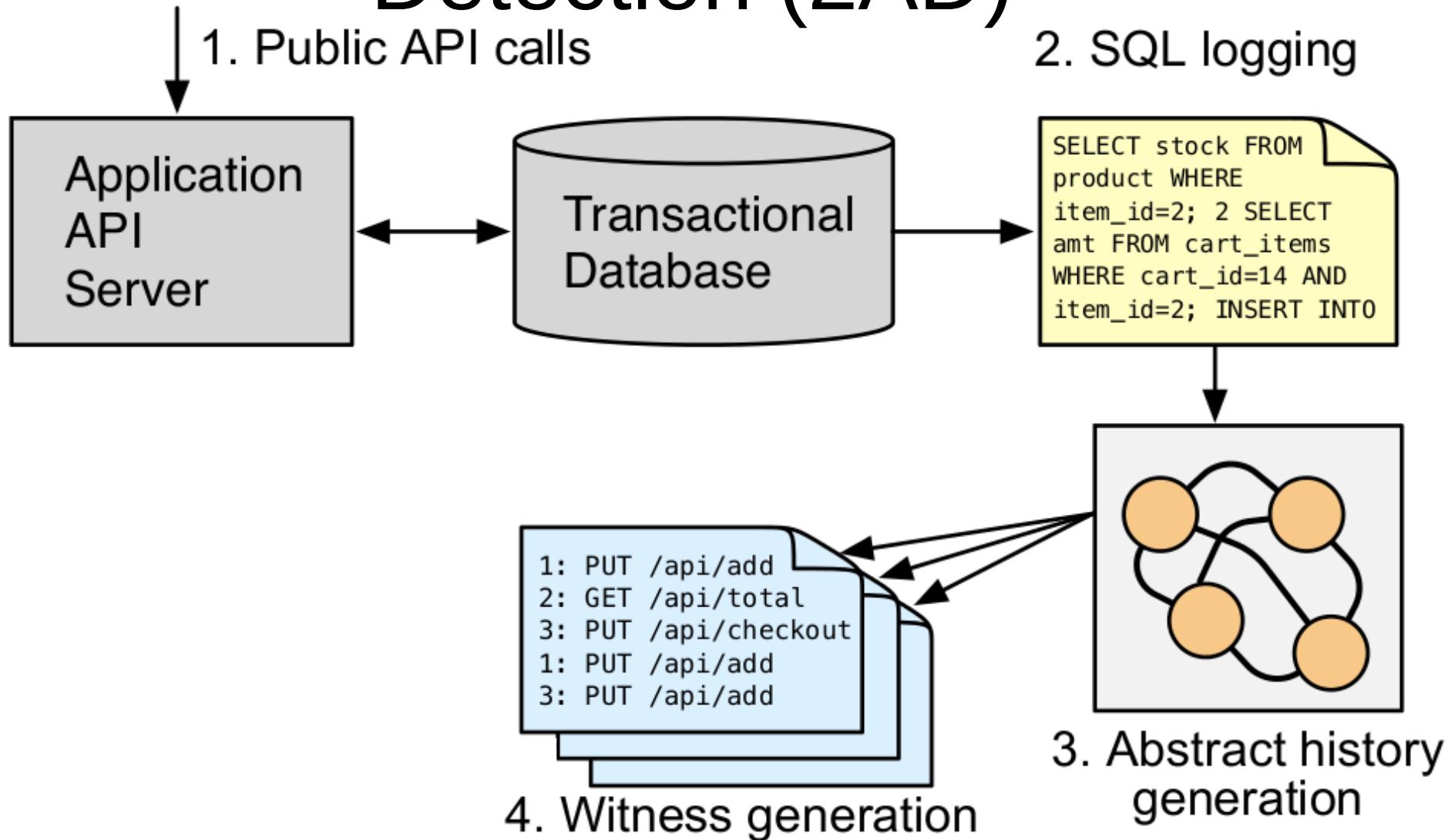
Approach: Abstract Anomaly Detection (2AD)



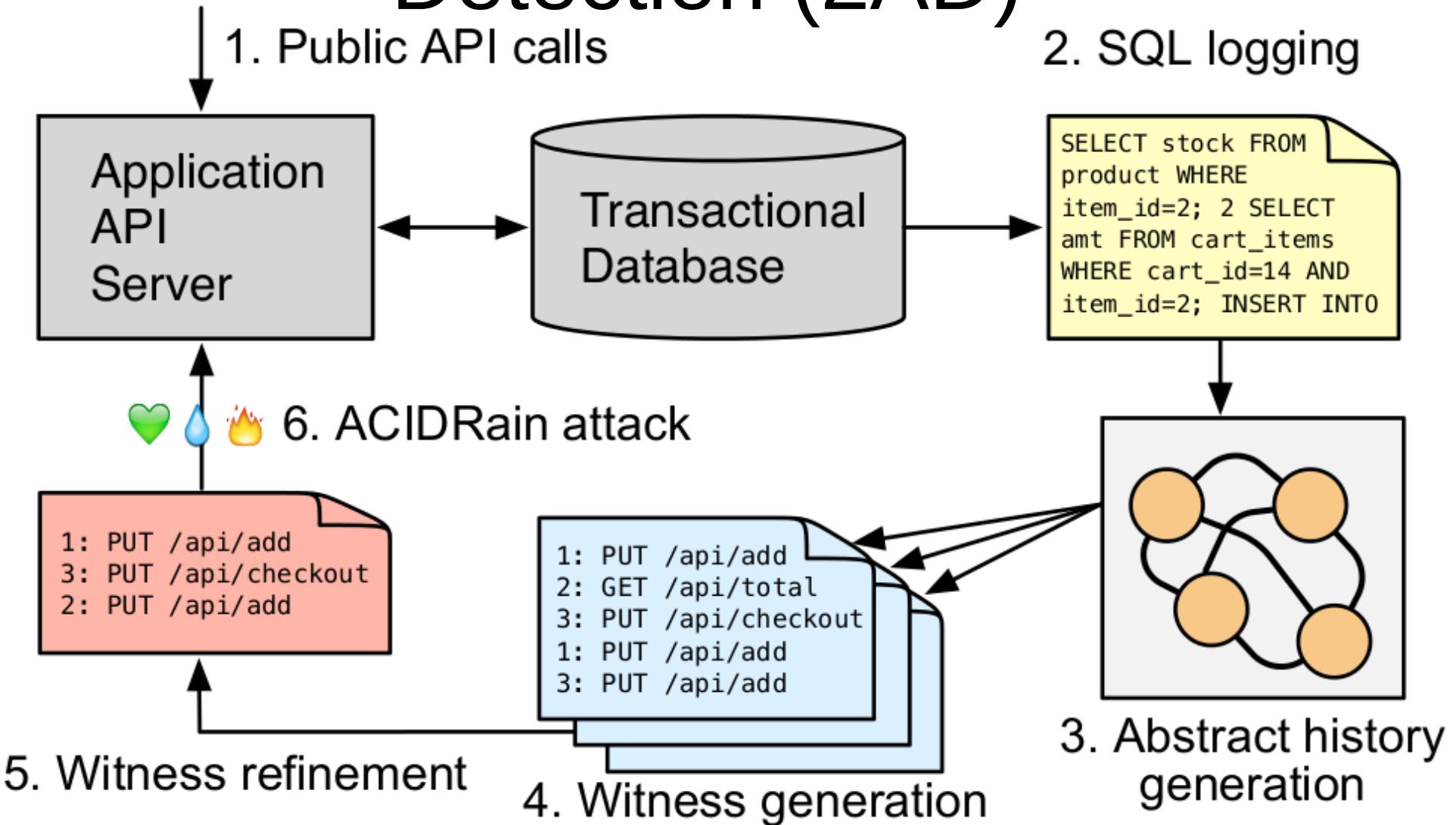
Approach: Abstract Anomaly Detection (2AD)



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
```

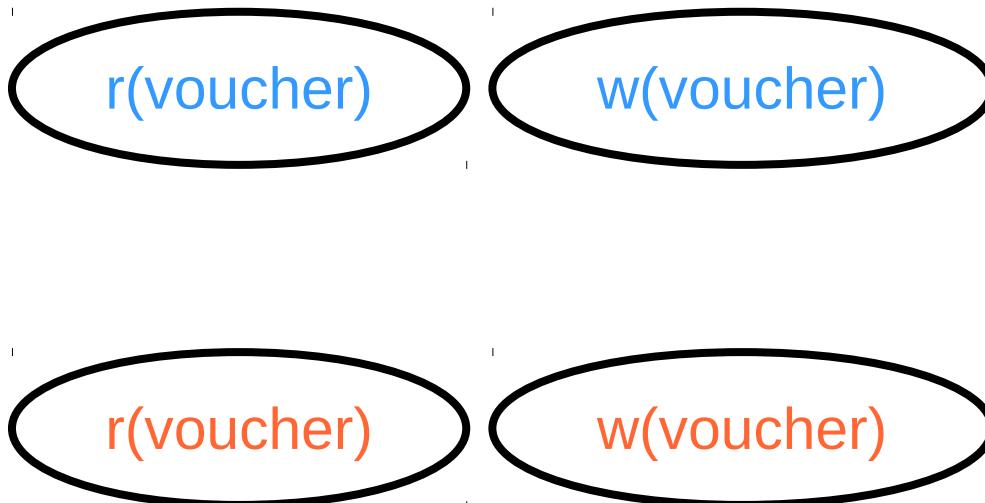
```
COMMIT
```

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

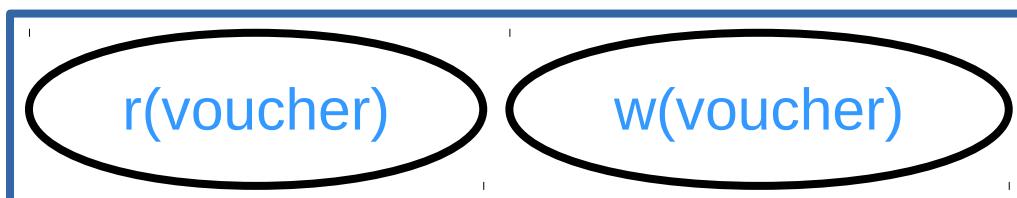
1. Add node for each operation



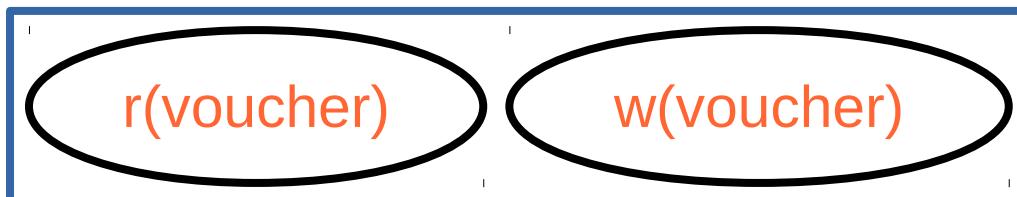
Abstract History Graph

```
{ 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

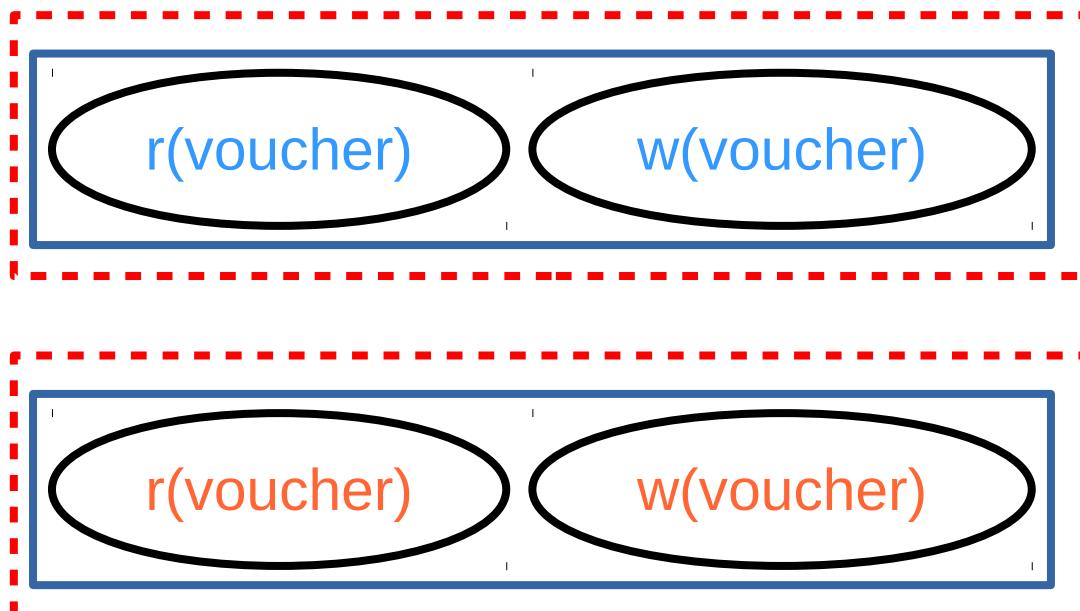
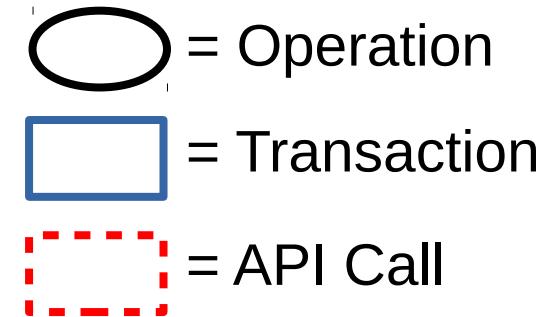


1. Add node for each operation
2. Add supernode for each transaction



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
```

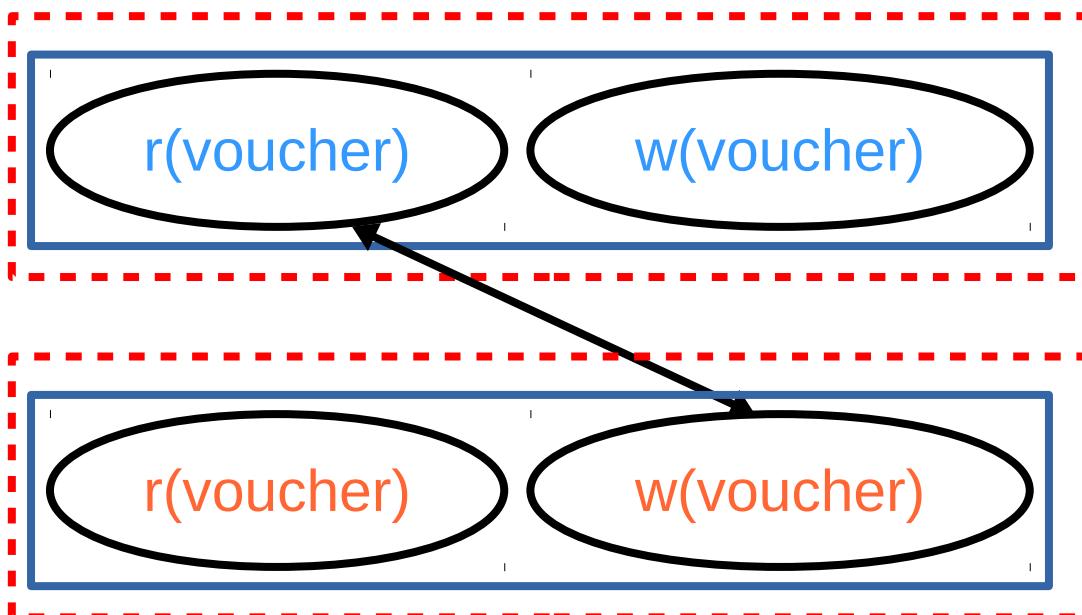
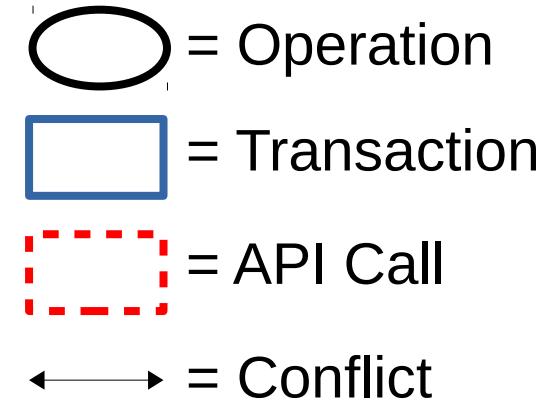


1. Add node for each operation
2. Add supernode for each transaction
3. Add super-supernode for each API call

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
```

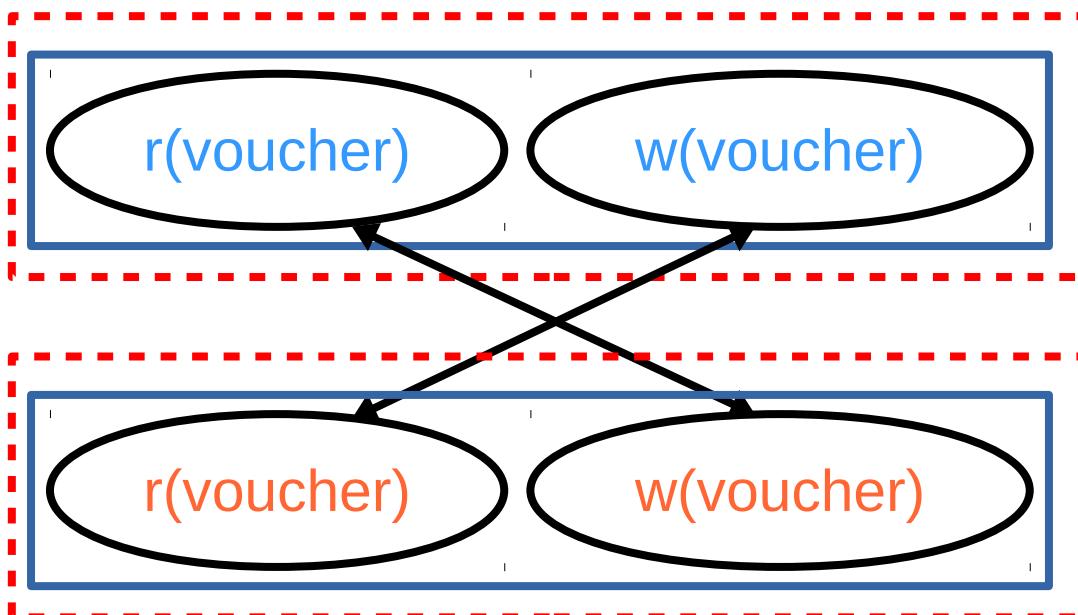
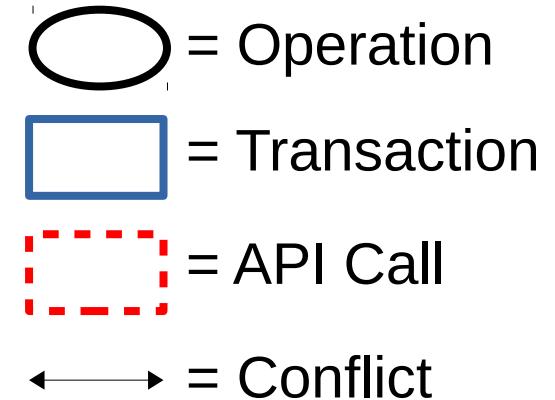


1. Add node for each operation
2. Add supernode for each transaction
3. Add super-supernode for each API call
4. Add edge for each conflict

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
```

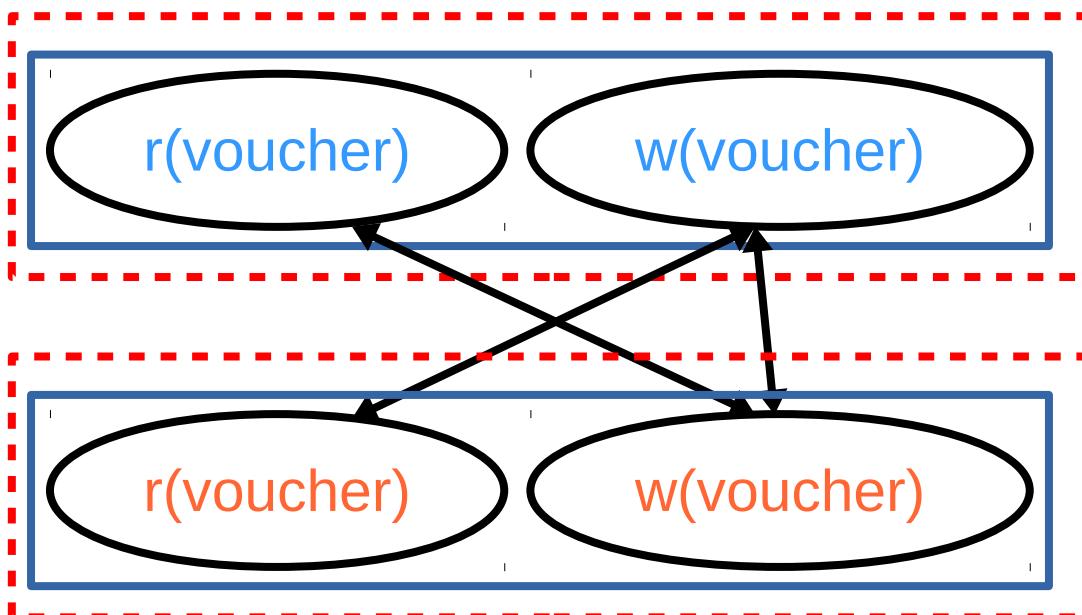
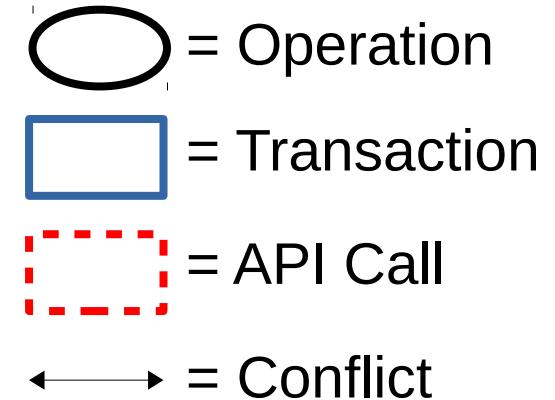


1. Add node for each operation
2. Add supernode for each transaction
3. Add super-supernode for each API call
4. Add edge for each conflict

Abstract History Graph

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

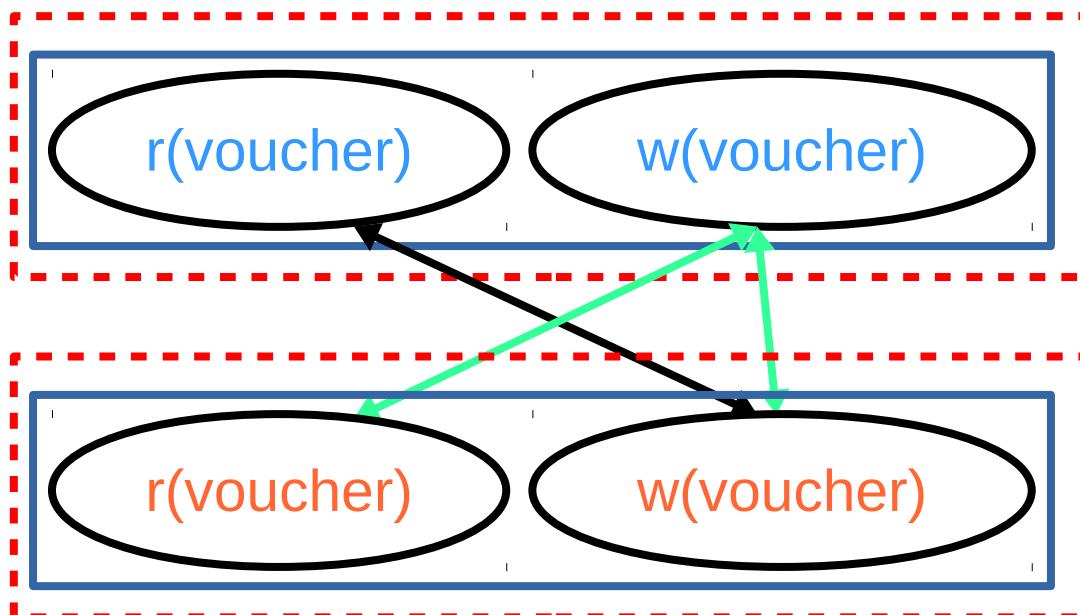
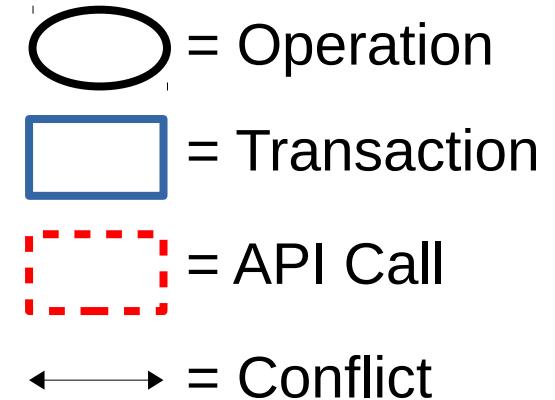
BEGIN TRANSACTION
SELECT usage FROM voucher WHERE code = HNUHY
UPDATE voucher SET usage = 1 WHERE code = HNUHY
COMMIT
```



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

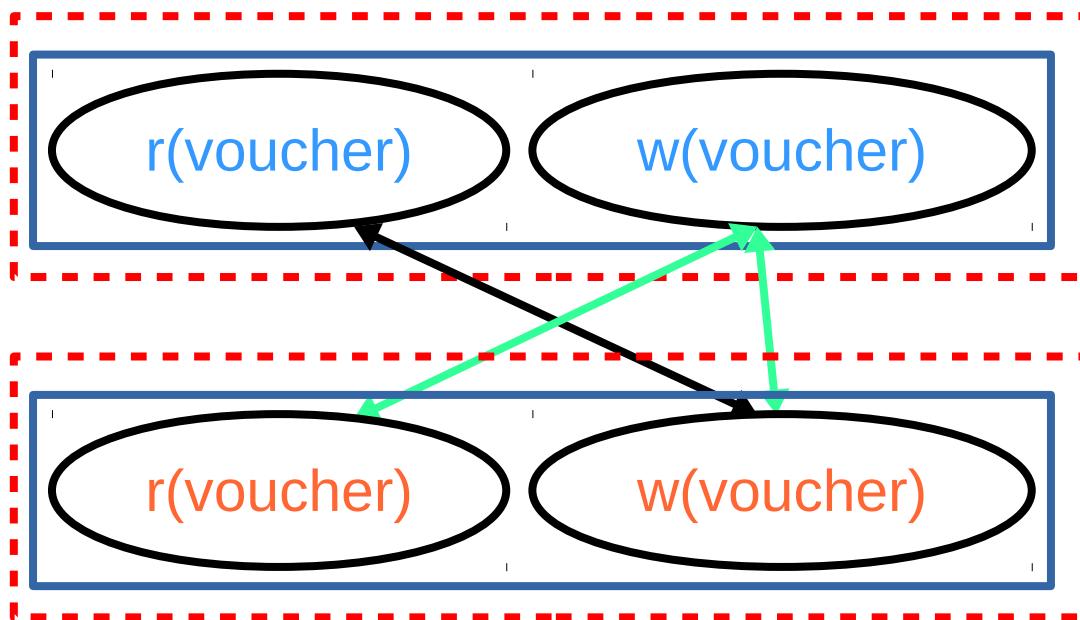
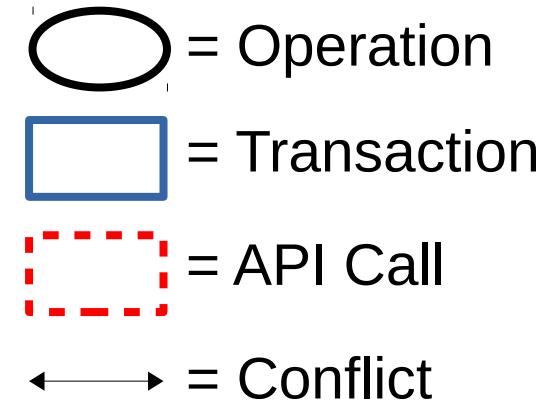
```
BEGIN TRANSACTION  
SELECT usage FROM voucher WHERE code = HNUHY  
UPDATE voucher SET usage = 1 WHERE code = HNUHY  
COMMIT  
  
BEGIN TRANSACTION  
SELECT usage FROM voucher WHERE code = HNUHY  
UPDATE voucher SET usage = 1 WHERE code = HNUHY  
COMMIT
```



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



Magento®
Community



shopizer

broadleaf
commerce

spreeTM
commerce



Presta**Shop**



ror-e.com

Woo **COMMERCE**

\$ \$ SHOPPE

Ecommerce Platforms



PrestaShop

Interested in 3
key invariants:

1. Inventory
2. Voucher
3. Cart



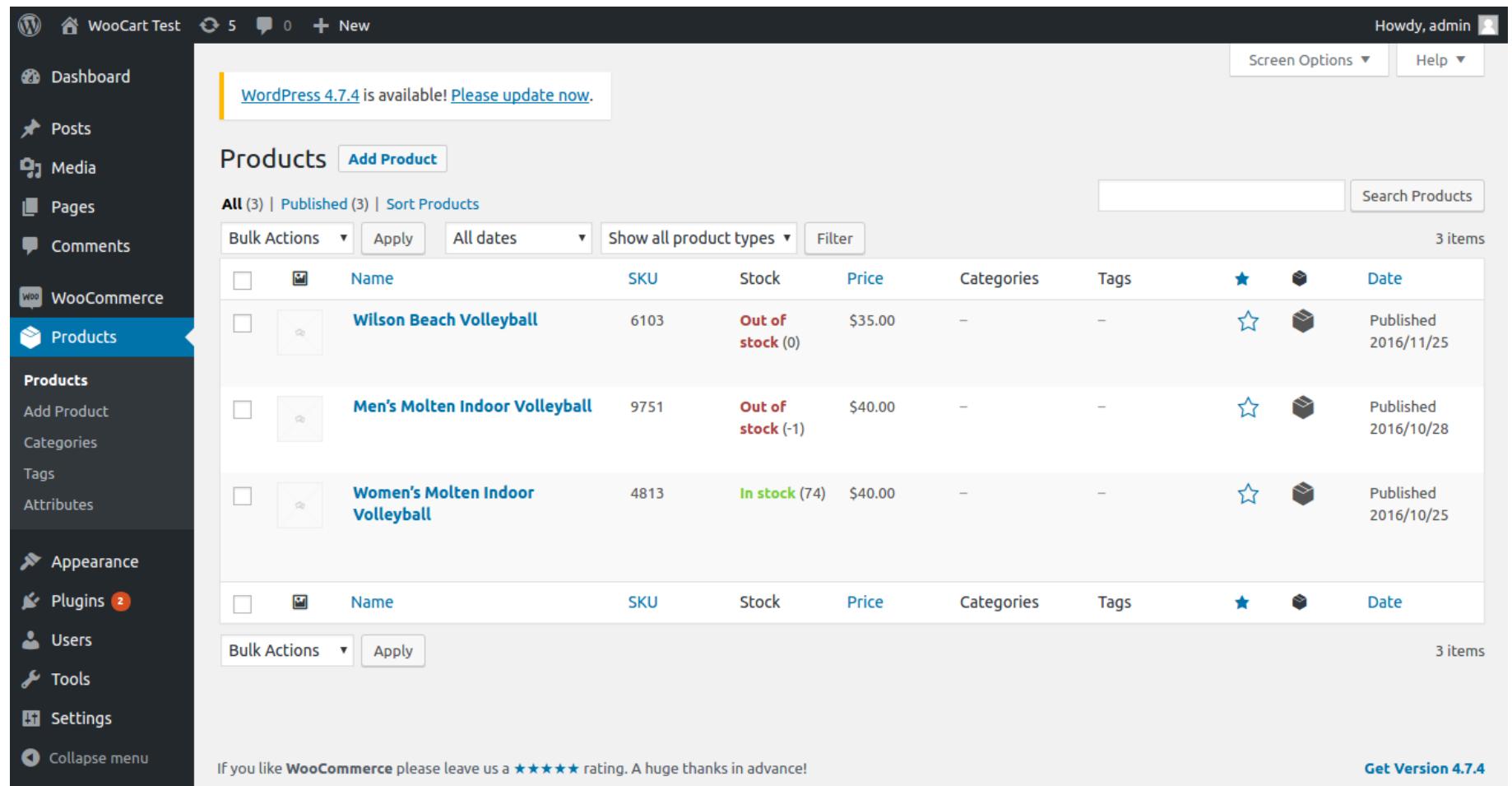
.com

OMMERCE

PPPE

Inventory Invariant

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



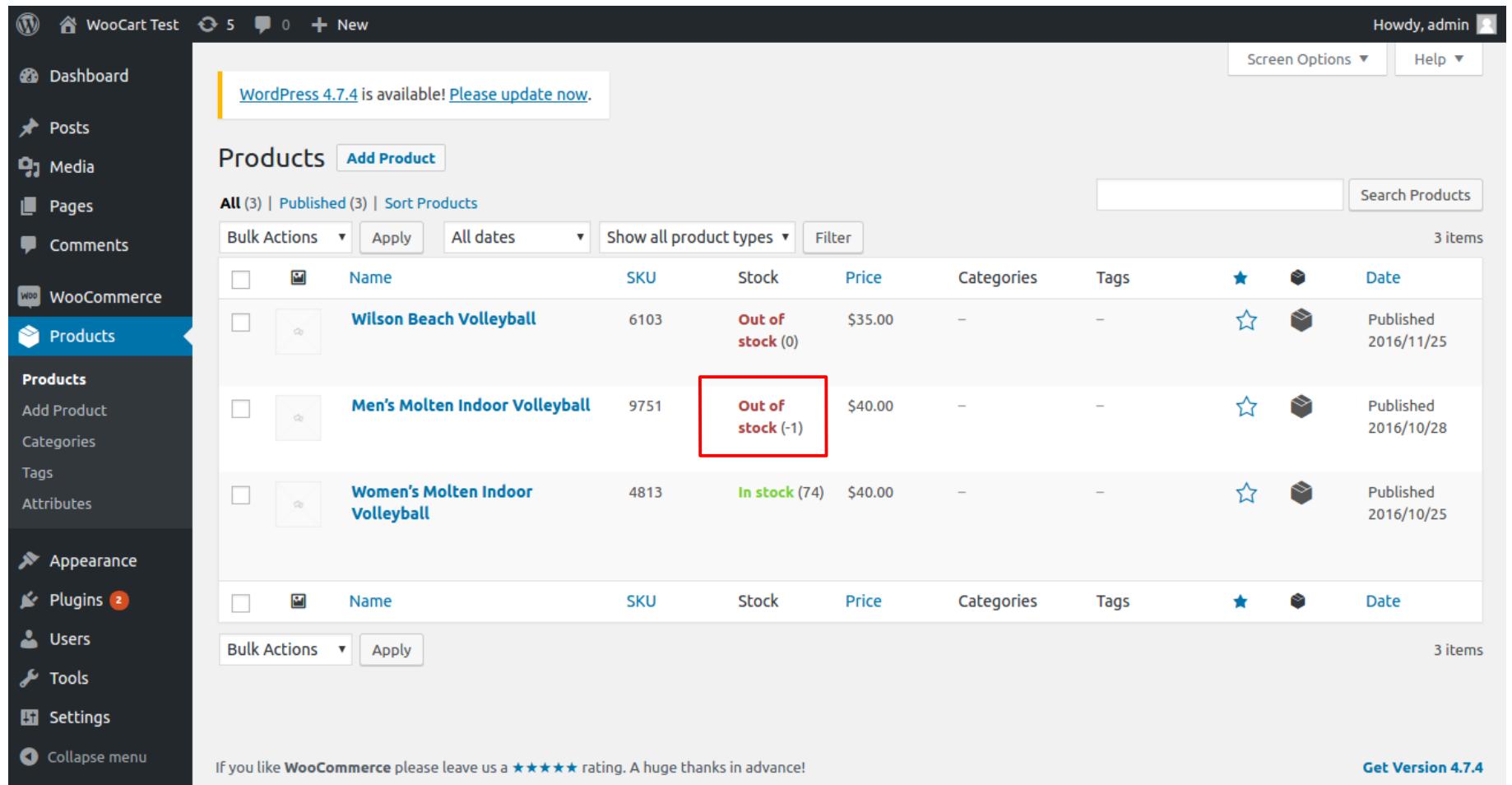
The screenshot shows the WordPress WooCommerce Products dashboard. The sidebar on the left includes links for Dashboard, Posts, Media, Pages, Comments, WooCommerce, Products, Appearance, Plugins, Users, Tools, and Settings. The main content area displays a list of products:

Bulk Actions	Name	SKU	Stock	Price	Categories	Tags	Date
<input type="checkbox"/>	Wilson Beach Volleyball	6103	Out of stock (0)	\$35.00	—	—	Published 2016/11/25
<input type="checkbox"/>	Men's Molten Indoor Volleyball	9751	Out of stock (-1)	\$40.00	—	—	Published 2016/10/28
<input type="checkbox"/>	Women's Molten Indoor Volleyball	4813	In stock (74)	\$40.00	—	—	Published 2016/10/25

At the bottom of the page, there is a message: "If you like WooCommerce please leave us a ★★★★ rating. A huge thanks in advance!" and a link to "Get Version 4.7.4".

Inventory Invariant

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



The screenshot shows the WooCommerce Products dashboard. The sidebar on the left includes links for Dashboard, Posts, Media, Pages, Comments, WooCommerce, Products, Appearance, Plugins, Users, Tools, Settings, and a Collapse menu. The main area displays a list of products:

	Name	SKU	Stock	Price	Categories	Tags	Published	Date
<input type="checkbox"/>	Wilson Beach Volleyball	6103	Out of stock (0)	\$35.00	—	—	★ 📦	Published 2016/11/25
<input type="checkbox"/>	Men's Molten Indoor Volleyball	9751	Out of stock (-1)	\$40.00	—	—	★ 📦	Published 2016/10/28
<input type="checkbox"/>	Women's Molten Indoor Volleyball	4813	In stock (74)	\$40.00	—	—	★ 📦	Published 2016/10/25

At the bottom of the page, there is a message: "If you like WooCommerce please leave us a ★★★★ rating. A huge thanks in advance!" and a link to "Get Version 4.7.4".

Inventory Invariant

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

The screenshot shows the WooCommerce Products dashboard in WordPress. The sidebar on the left includes links for Dashboard, Posts, Media, Pages, Comments, WooCommerce, Products, Appearance, Plugins, Users, Tools, and Settings. The main area displays a list of three products: 'Wilson Beach Volleyball', 'Men's Molten Indoor Volleyball', and 'Women's Molten Indoor Volleyball'. The 'Women's Molten Indoor Volleyball' row is highlighted with a red box and contains the text 'Out of stock (-1)'. The dashboard also features a notice about WordPress 4.7.4, search filters, and a message at the bottom encouraging ratings.

Name	SKU	Stock	Price	Categories	Tags	Date
Wilson Beach Volleyball		-			★	Published 2016/11/25
Men's Molten Indoor Volleyball		-			★	Published 2016/10/28
Women's Molten Indoor Volleyball		-			★	Published 2016/10/25

If you like WooCommerce please leave us a ★★★★ rating. A huge thanks in advance!

Get Version 4.7.4

Voucher Invariant

Vouchers should not be spent past their intended limit

The screenshot shows the oscar CMS interface with a dark theme. At the top, there is a navigation bar with links for Dashboard, Catalogue, Fulfilment, Customers, Offers, Content, and Reports. The user is logged in as `twarszaw@stanford.edu`. Below the navigation bar, the URL `Dashboard / Vouchers / v5` is visible.

The main content area displays a table titled "Voucher details" for a voucher named "v5". The table includes the following information:

Name	v5
Code	V5
Start datetime	3 Aug 2016, 10:30 a.m.
End datetime	4 Nov 2016, 3:45 a.m.
Usage	Can be used once by one customer
Discount	£10.00 discount on site

Below this is another table titled "Voucher performance" showing usage statistics:

Number of basket additions	2
Number of orders	1
Total discount	£10.00

Finally, there is a table titled "Recent orders" showing two recent purchases:

Order number	Order total	Discount	Date placed
100020	£21.42	£10.00	10 Aug 2016, 10:47 p.m.
100019	£52.84	£10.00	10 Aug 2016, 10:47 p.m.

At the bottom left, there are buttons for "Edit" (blue), "Delete" (red), and "cancel".

Voucher Invariant

Vouchers should not be spent past their intended limit

The screenshot shows the oscar CMS interface with a navigation bar at the top. The main content area displays a voucher named 'v5' with the following details:

Voucher details	
Name	v5
Code	V5
Start datetime	3 Aug 2016, 10:30 a.m.
End datetime	4 Nov 2016, 3:45 a.m.
Usage	Can be used once by one customer
Discount	£10.00 discount on site

Below this, the 'Voucher performance' section shows:

Number of basket additions	2
Number of orders	1
Total discount	£10.00

The 'Recent orders' section lists two recent orders:

Order number	Order total	Discount	Date placed
100020	£21.42	£10.00	10 Aug 2016, 10:47 p.m.
100019	£52.84	£10.00	10 Aug 2016, 10:47 p.m.

At the bottom, there are buttons for 'Edit' and 'Delete'.

Voucher Invariant

Vouchers should not be spent past their intended limit

The screenshot shows the oscar CMS interface. At the top, there is a navigation bar with links for Dashboard, Catalogue, Fulfilment, Customers, Offers, Content, and Reports. The user is logged in as `twarszaw@stanford.edu`. Below the navigation bar, the URL `Dashboard / Vouchers / v5` is visible.

The main content area displays a voucher detail page for "v5". A large red box highlights the "Voucher details" section, which contains the text: "Can be used once by one customer".

Below this, another red box highlights the "Voucher performance" section, which includes a chart showing "Number of basket additions" over time.

A third red box highlights the "Recent orders" section, which lists two recent orders:

Order number	Order total	Discount	Date placed
100020	£21.42	£10.00	10 Aug 2016, 10:47 p.m.
100019	£52.84	£10.00	10 Aug 2016, 10:47 p.m.

At the bottom of the page, there are buttons for "Edit" and "Delete".

Cart Invariant

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

LFS

admin | Logout

Shop Catalog Properties HTML Customers Marketing Utils Help

Pages Start End Name State All Submit

OVERVIEW DELETE ORDER RESEND ORDER STATE: Submitted

46
45
44
43
42
41
40
39
38
37
36
35
34
33
32
31
30
29
28

Order

General

Order number:	46	Date:	May 12, 2017, 8:37 p.m.
E-mail:	twarszaw@stanford.edu	State:	Submitted (May 12, 2017, 8:37 p.m.)
Phone:		Total:	\$40.00

Shipping address

Todd Warszawski 94305 Stanford 121 Campus Drive CA USA 2674413738 twarszaw@stanford.edu	Todd Warszawski 94305 Stanford 121 Campus Drive CA USA 2674413738 twarszaw@stanford.edu	Standard	Cash on delivery
---	---	----------	------------------

Invoice address

Shipping method	Payment method
-----------------	----------------

Items

Image	SKU	Name	Amount	Price	Included VAT	Total
	1	Molten Men's Indoor Volleyball	10	\$40.00	\$0.00	\$400.00

Shipping (Standard) 1 \$0.00 \$0.00 \$0.00 \$40.00
Payment (Cash on delivery) 1 \$0.00 \$0.00 \$0.00 \$0.00
Inclusive VAT: \$0.00

Powered by LFS — Lightning Fast Shop
© 2009 - 2017 by Kai Diefenbach and contributors — All rights reserved
Distributed under the BSD-License

Cart Invariant

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

LFS

admin | Logout

Shop Catalog Properties HTML Customers Marketing Utils Help

Pages Start End Name State All Submit

OVERVIEW DELETE ORDER RESEND ORDER STATE: Submitted

46
45
44
43
42
41
40
39
38
37
36
35
34
33
32
31
30
30
29
28

Order

General

Order number:	46	Date:	May 12, 2017, 8:37 p.m.
E-mail:	twarszaw@stanford.edu	State:	Submitted (May 12, 2017, 8:37 p.m.)
Phone:		Total:	\$40.00

Shipping address

Todd Warszawski 94305 Stanford 121 Campus Drive CA USA 2674413738 twarszaw@stanford.edu	Todd Warszawski 94305 Stanford 121 Campus Drive CA USA 2674413738 twarszaw@stanford.edu	Standard	Cash on delivery
---	---	----------	------------------

Invoice address

Shipping method

Payment method

Items

Image	SKU	Name	Amount	Price	Included VAT	Total
	1	Molten Men's Indoor Volleyball	10	\$40.00	\$0.00	\$400.00
			1	\$0.00	\$0.00	\$0.00
			1	\$0.00	\$0.00	\$0.00
						\$40.00
						Inclusive VAT: \$0.00

Powered by LFS — Lightning Fast Shop
© 2009 - 2017 by Kai Diefenbach and contributors — All rights reserved
Distributed under the BSD-License

Cart Invariant

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

The screenshot shows the LFS (Lightning Fast Shop) administration interface. On the left, there's a sidebar with a list of numbers from 46 down to 28. The main area has a navigation bar with links like Shop, Catalog, Properties, HTML, Customers, Marketing, Utils, and Help. The user is logged in as 'admin'. Below the navigation is a search bar with fields for Start, End, Name, and State, and buttons for T, Y, W, and a dropdown menu. There are also 'OVERVIEW' and 'DELETE' buttons.

The main content area displays an order detail. On the left, there are sections for 'General' (Order number: 1, E-mail: twarszaw@stanford.edu, Phone: 2674413738), 'Shipping address' (Todd Warszawski, 94305 Stanford, 121 Campus Drive, CA, USA, 2674413738, twarszaw@stanford.edu), and 'Image' (a small thumbnail of an NCAA logo).

A large red box highlights the order summary table. The table has columns: Amount, Price, Included VAT, and Total. It shows one row with an Amount of 10, a Price of \$40.00, and a Total of \$400.00. Below the table, a note says 'Inclusive VAT: \$0.00'. Two red arrows point from the 'Total' column of the table to this note, indicating that the total amount charged for the order is equal to the sum of the item prices.

At the bottom of the page, there's a footer with the text: 'Powered by LFS — Lightning Fast Shop', '© 2009 - 2017 by Kai Diefenbach and contributors — All rights reserved', and 'Distributed under the BSD-License'.

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

The screenshot shows a GitHub issue page for the repository `drhenner/ror_ecommerce`. The issue, titled "Can create a submitted order with total charged not matching items associated with order. #174", is marked as **Closed** by TWarszawski on Sep 8, 2016, with 3 comments.

Comments:

- TWarszawski commented on Sep 8, 2016**
 - When a customer begins check out and finishes checkout in a different window concurrently, the order total may not accurately reflect the products in the order.**
 - Steps to reproduce:**
 1. Start site, create a customer, create/pick test product.
 2. Have the customer add the product to the cart and go through checkout up until the complete order step.
 3. In a separate tab or window, log in again as the customer and go to the cart page.
 4. Have the customer on the cart page update the quantity in the cart.
 5. Begin checkout from the cart page and finish checkout (click the Complete Order button) as close to the same time as possible.
 - Expected Result:**

The amount charged matches the sum of the totals for each item in the cart.
 - Actual Result:**

The amount charged reflects the total of the original items in the cart, but when examining the order page the updated items are shown to be associated with the order.

drhenner commented on Sep 8, 2016

@TWarszawski Thanks for the details. I'm in over my head with work but I'll try to get this solved as soon as possible.

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.

Even more preferred would be to checkout with the items you are currently viewing but that could be difficult. If it isn't too hard I'll go with this option.

Assignees: No one assigned

Labels: None yet

Projects: None yet

Milestone: No milestone

Notifications: You're receiving notifications because you were mentioned.

2 participants: drhenner, TWarszawski

8 confirmed

Developer Response

The screenshot shows a GitHub issue page for the repository `drhenner/ror_ecommerce`. The issue, titled "Can create a submitted order with total charged not matching items associated with order. #174", is marked as **Closed** by TWarszawski on Sep 8, 2016, with 3 comments.

Comments:

- TWarszawski commented on Sep 8, 2016**
 - When a customer begins check out and finishes checkout in a different window concurrently, the order total may not accurately reflect the products in the order.**
 - Steps to reproduce:**
 1. Start site, create a customer, create/pick test product.
 2. Have the customer add the product to the cart and go through checkout up until the complete order step.
 3. In a separate tab or window, log in again as the customer and go to the cart page.
 4. Have the customer on the cart page update the quantity in the cart.
 5. Begin checkout from the cart page and finish checkout (click the Complete Order button) as close to the same time as possible.
 - Expected Result:**

The amount charged matches the sum of the totals for each item in the cart.
 - Actual Result:**

The amount charged reflects the total of the original items in the cart, but when examining the order page the updated items are shown to be associated with the order.

drhenner commented on Sep 8, 2016

@TWarszawski Thanks for the details. I'm in over my head with work but I'll try to get this solved as soon as possible.

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.

Even more preferred would be to checkout with the items you are currently viewing but that could be difficult. If it isn't too hard I'll go with this option.

Assignees: No one assigned

Labels: None yet

Projects: None yet

Milestone: No milestone

Notifications: You're receiving notifications because you were mentioned.

Participants: drhenner, TWarszawski

8 confirmed

Developer Response

The screenshot shows a GitHub repository page for `drhenner/ror_ecommerce`. The top navigation bar includes links for `Pull requests`, `Issues`, and `Gist`. The repository stats show 75 issues, 1,114 stars, 334 forks, and 8 confirmed. The main content is an issue titled "Can create a submitted order with total charged not matching items associated with order." with ID `#174`. The issue is marked as `Closed` by TWarszawski on Sep 8, 2016, with 3 comments. A comment from TWarszawski describes a bug where the order total may not reflect the products in the order due to concurrent checkout. It lists steps to reproduce: starting the site, creating a customer, adding a product to the cart, and completing the checkout. The right sidebar shows assignees, labels, and projects.

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.

8 confirmed

  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!

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.

8 confirmed

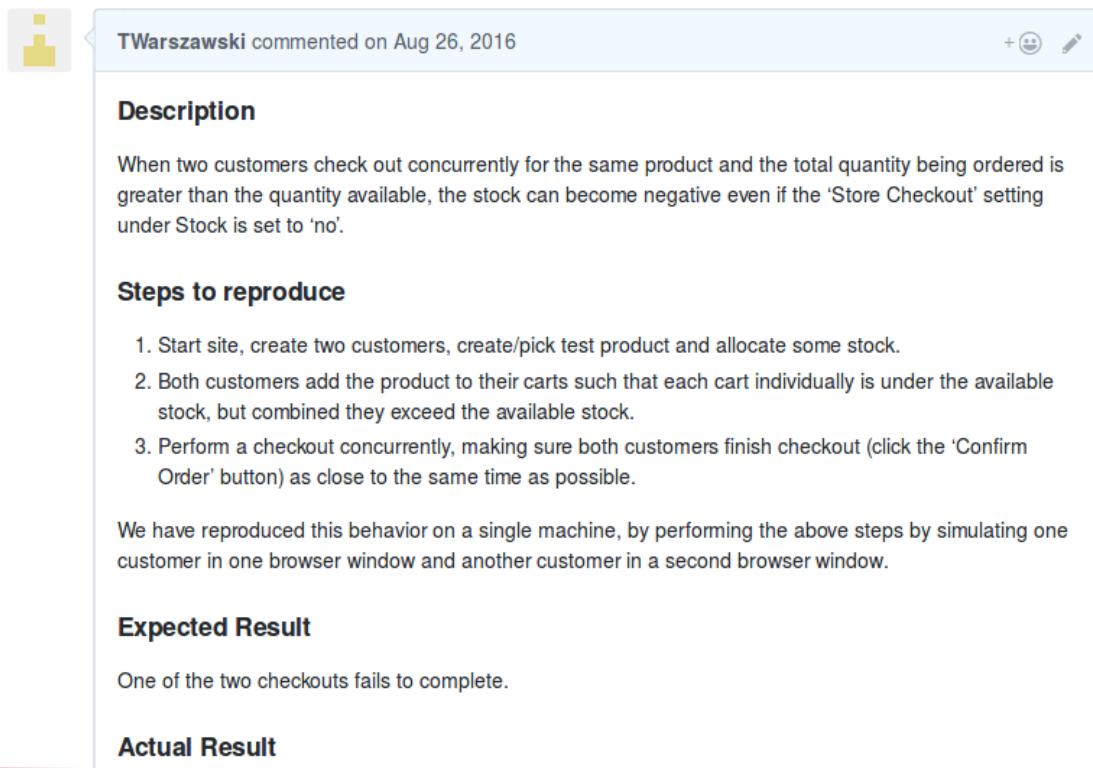
  danielkerr closed this on Aug 28, 2016

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!

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 ...](https://twitter.com/pbailis/status/...)

[github.com/opencart/openc ...](https://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

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



jberrymann 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



jberrymann 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>