WeBridge: Synthesizing Stored Procedures for Large-Scale Real-World Web Applications

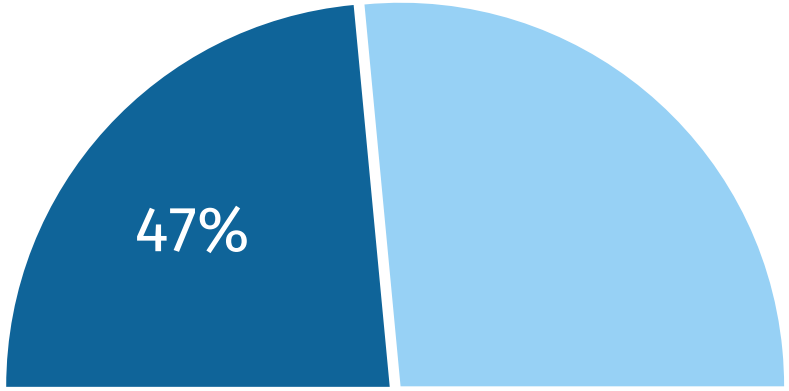
Gansen Hu, Zhaoguo Wang, **Chuzhe Tang**, Jiahuan Shen,
Zhiyuan Dong, Sheng Yao, and Haibo Chen

Institute of Parallel and Distributed Systems (IPADS),
Shanghai Jiao Tong University

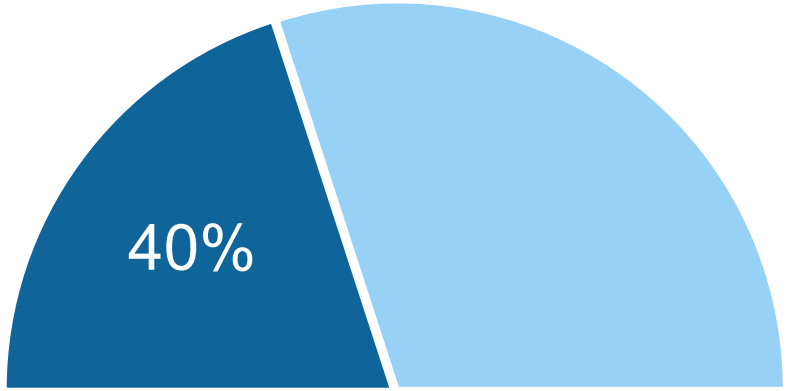
SHANGHAI JIAO TONG UNIVERSITY

IPADS
INSTITUTE OF PARALLEL AND DISTRIBUTED SYSTEMS

Latency is Critical to Web Applications



<2s page load time
expected by 47% users



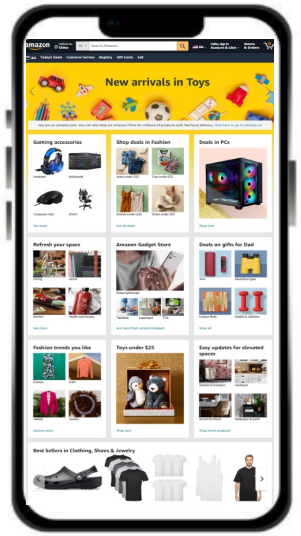
>3s page load time
causes 40% users to leave

"A 1 second page delay could potentially cost $2.5 million in lost sales every year"

* How Loading Time Affects Your Bottom Line. neilpatel.com

Web Apps Suffer from DB Round Trips

56% request processing time is spent on DB round trips!



Broadleaf's Add-Cart API

```
IN: pid, uid
item := ORM.getItem(pid)
user := ORM.getUser(uid)
cart := ORM.getCart(user.cart_id)
if not item.available:
  log := new Log("Item Unavailable")
  ORM.save(log)
else:
  cart.items.append(item)
  ORM.save(cart)
```

RT 1 `Select * From Items …`

RT 2 `Select * From Users …`

RT 3 `Select * From Carts …`

RT 4 `Insert Into Log Values (…)`
or
`Insert Into Items Values (…)`

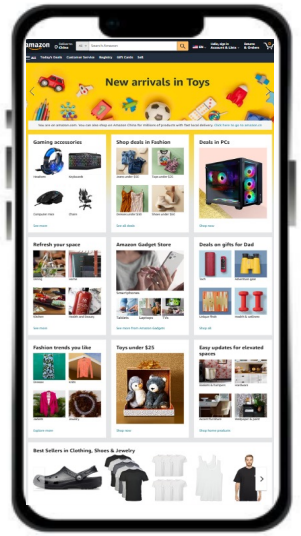
Database

Dynamically generated & Individually sent!

* Broadleaf Commerce, 1.7k Stars on GitHub, https://github.com/BroadleafCommerce

Web Apps Suffer from DB Round Trips

56% request processing time is spent on DB round trips!



Broadleaf's Add-Cart API

```
IN: pid, uid
item := ORM.getItem(pid)
user := ORM.getUser(uid)
cart := ORM.getCart(user.cart_id)
if not item.available:
  log := new Log("Item Unavailable")
  ORM.save(log)
else:
  cart.items.append(item)
  ORM.save(cart)
```

RT 1 `Select * From Items …`

RT 2 `Select * From Users …`

RT 3 `Select * From Carts …`

RT 4 `Insert Into Log Values (…)`
or
`Insert Into Items Values (…)`

Database

Dynamically generated & Individually sent!

Research Question: How to reduce DB round trips?

* Broadleaf Commerce, 1.7k Stars on GitHub, https://github.com/BroadleafCommerce

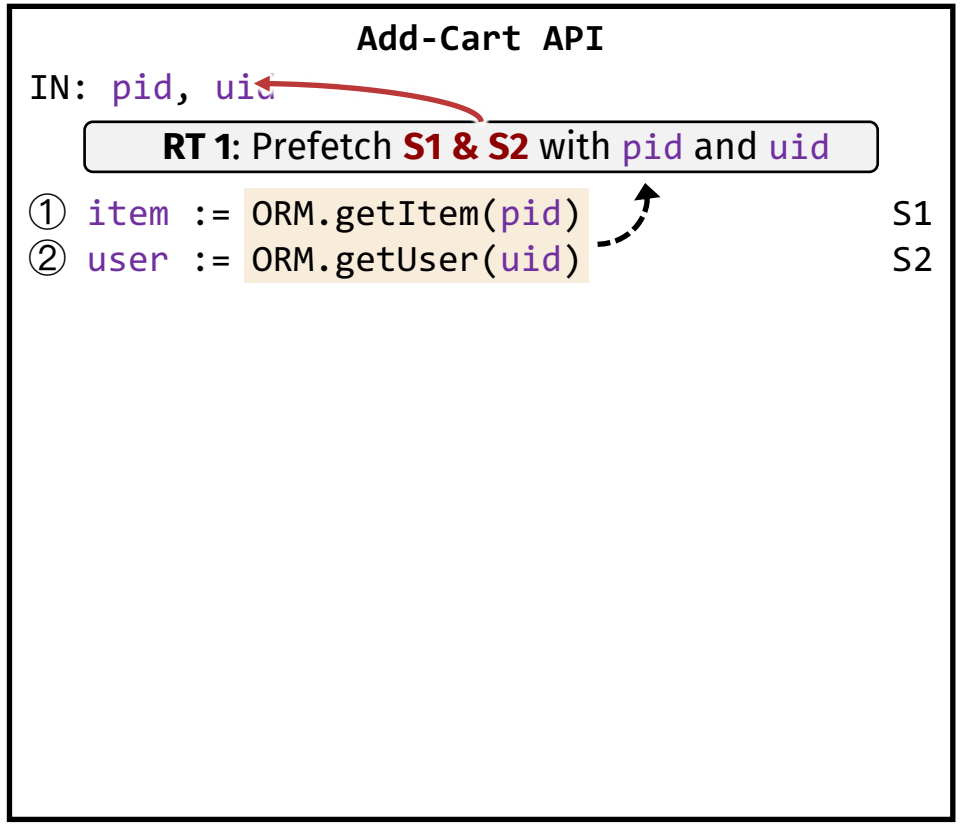
Existing Method: Prefetching

Prefetching (or, eager execution)
Execute statements as soon as parameters become ready.

Existing Method: Prefetching

Prefetching (or, eager execution)

Execute statements as soon as parameters become ready.

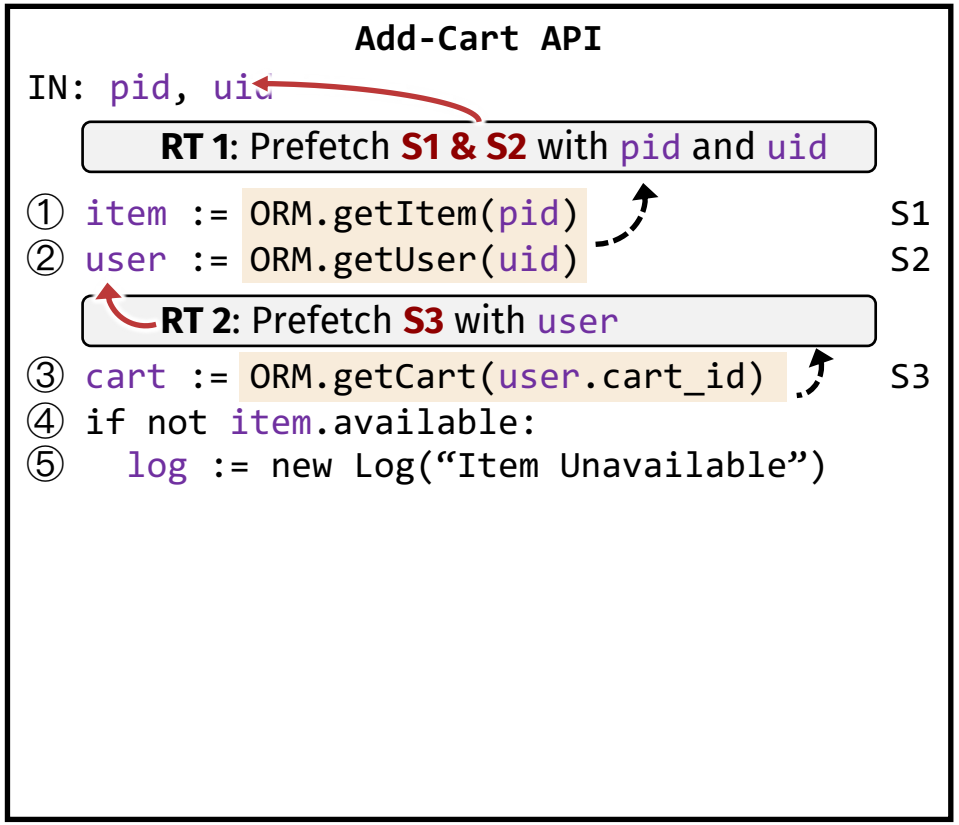


Data Dep. Control Dep. Get Results From

Existing Method: Prefetching

Prefetching (or, eager execution)

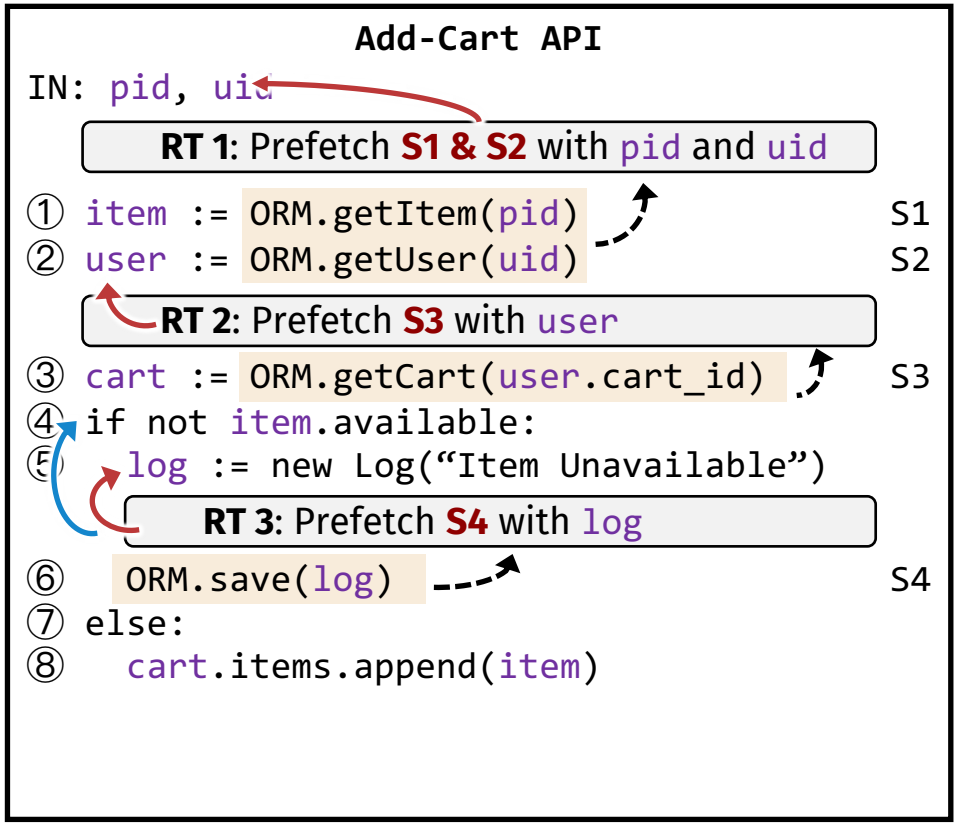
Execute statements as soon as parameters become ready.



Data Dep. Control Dep. Get Results From

Existing Method: Prefetching

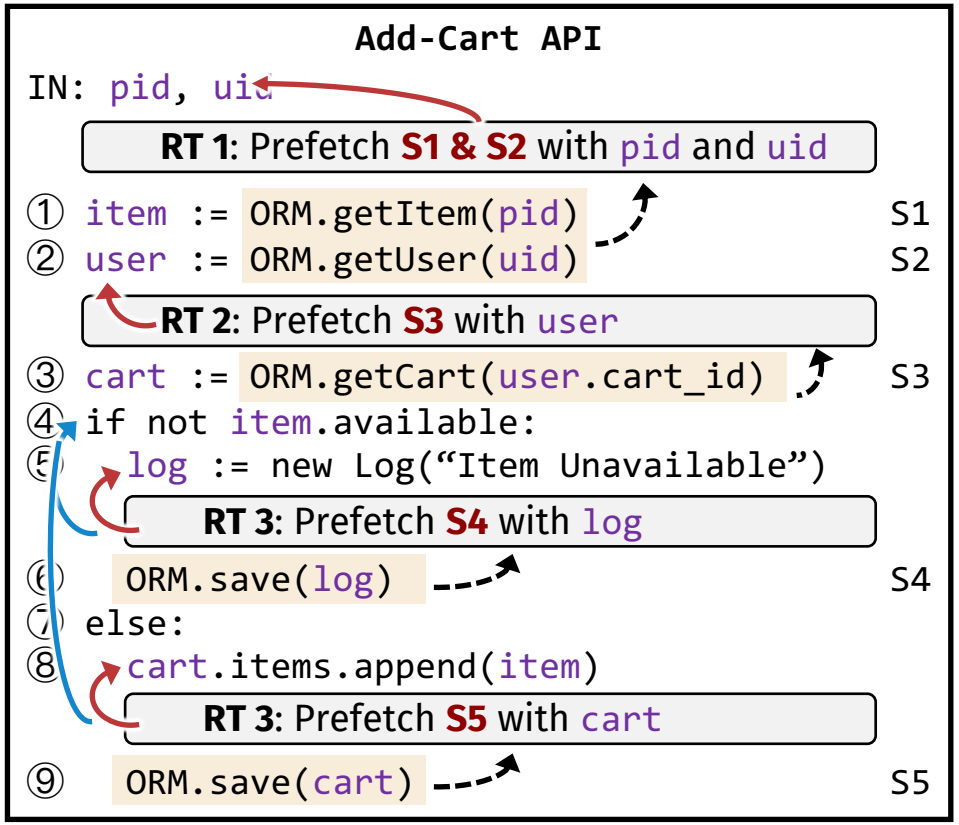
Prefetching (or, eager execution)
Execute statements as soon as parameters become ready.



Data Dep. Control Dep. Get Results From

Existing Method: Prefetching

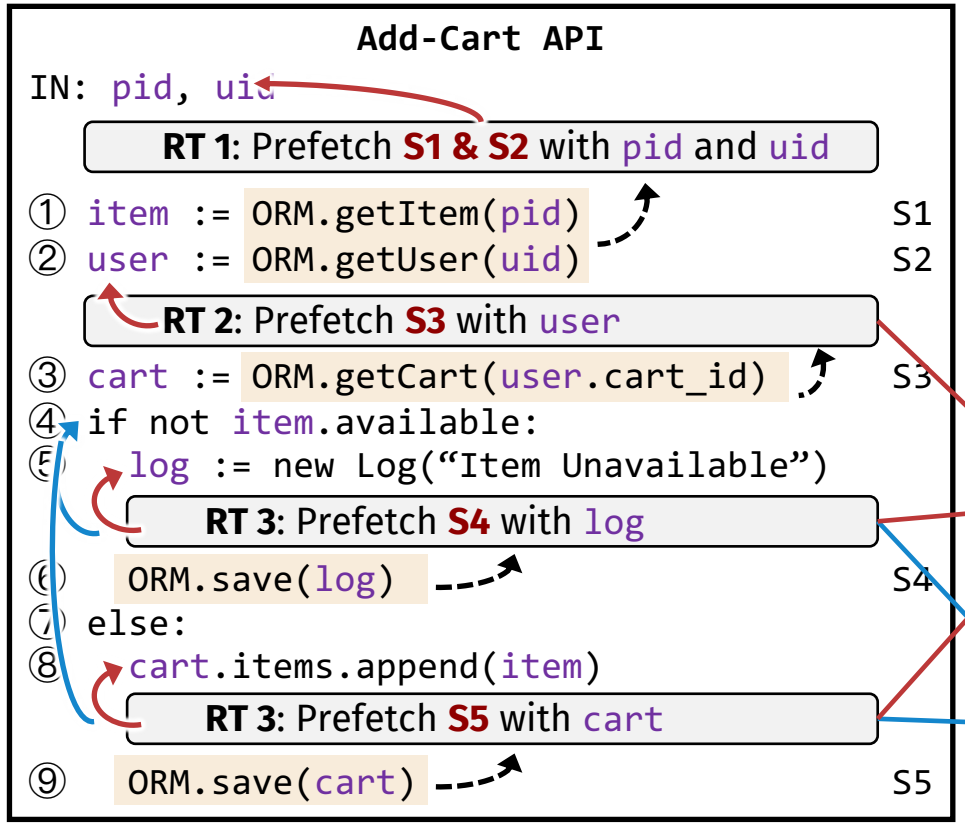
Prefetching (or, eager execution)
Execute statements as soon as parameters become ready.



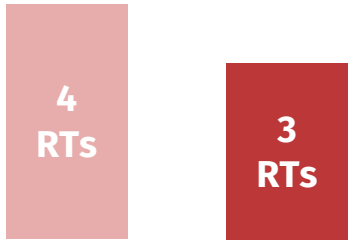
Data Dep. Control Dep. Get Results From

Existing Method: Prefetching

Prefetching (or, eager execution)
Execute statements as soon as parameters become ready.



Number of Round Trips

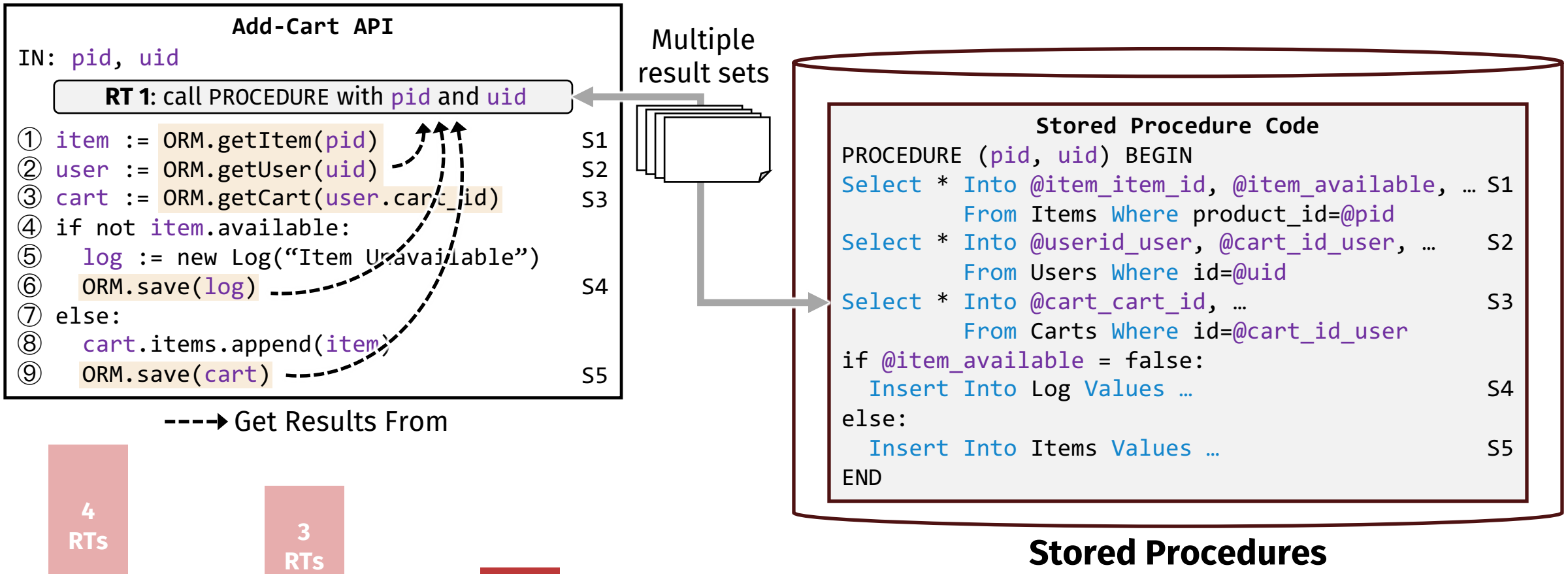


Data Dependency
Prefetching of S3 must wait until variable user is finalized.

Control Dependency
Prefetching of S4/5 must wait until branching at ④ is decided.

Data Dep. Control Dep. Get Results From

Our Approach: Shipping Dependencies to DB



Our Approach: Shipping Dependencies to DB

Add-Cart API

```
IN: pid, uid
    RT 1: call PROCEDURE with pid and uid
① item := ORM.getItem(pid)                    S1
② user := ORM.getUser(uid)                    S2
③ cart := ORM.getCart(user.cart_id)           S3
④ if not item.available:
⑤   log := new Log("Item Unavailable")
⑥   ORM.save(log)                             S4
⑦ else:
⑧   cart.items.append(item)
⑨   ORM.save(cart)                            S5
```

Multiple result sets

Stored Procedure Code

```
PROCEDURE (pid, uid) BEGIN
Select * Into @item_item_id, @item_available, … S1
         From Items Where product_id=@pid
Select * Into @userid_user, @cart_id_user, …    S2
         From Users Where id=@uid
Select * Into @cart_cart_id, …                  S3
         From Carts Where id=@cart_id_user
if @item_available = false:
  Insert Into Log Values …                      S4
else:
  Insert Into Items Values …                    S5
END
```

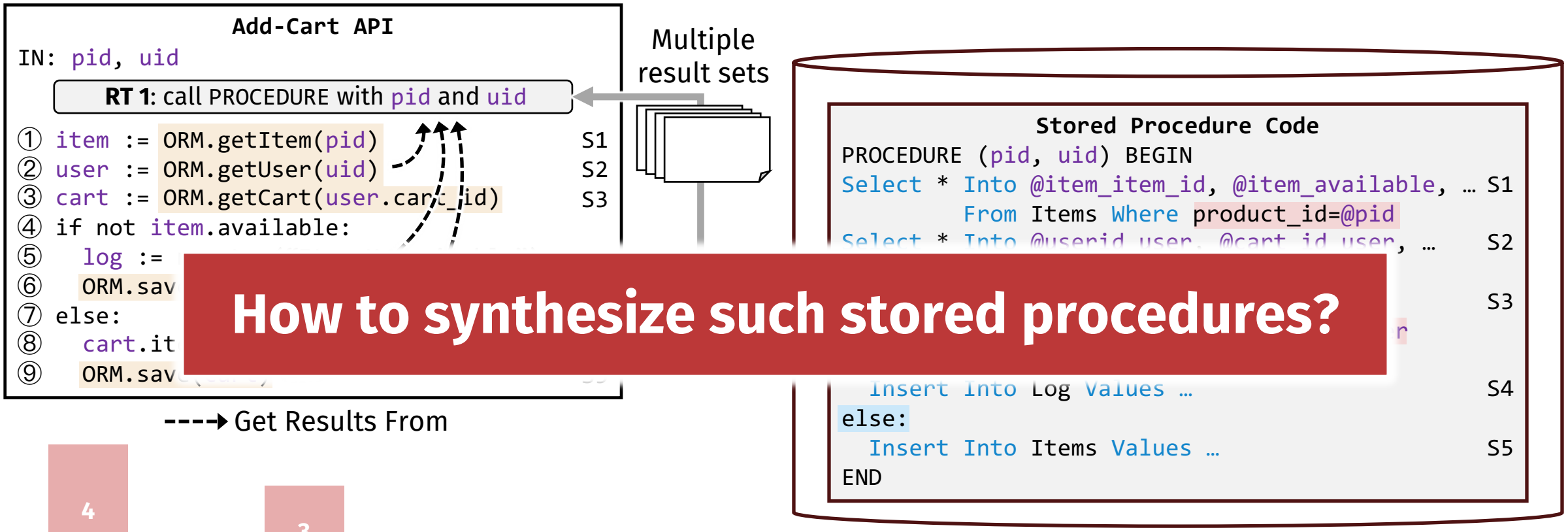
Stored Procedures

- Can express **data dep** via local variables
- Can express **control dep** via branching commands

----→ Get Results From

Number of Round Trips

Our Approach: Shipping Dependencies to DB



- Can express **data dep** via local variables
- Can express **control dep** via branching commands

Practical Challenge

- **Strawman Solution: static trans-compiling**
  - Statically build an IR, then compile into stored procedures.

Practical Challenge

- **Strawman Solution: static trans-compiling**
  - Statically build an IR, then compile into stored procedures.

- **Challenge: Language dynamism**
  - The language feature that **alter program behaviors at run time** (e.g., Java's reflection), commonly used by web frameworks.

```
entityManager.find(Entity.class, id)
```

1. Issue `Select` with `Inner Join`
2. Issue `Select` with `Left Outer Join`
3. Defer `Select` until returned object is used
4. Do nothing
5. ...

Practical Challenge

- **Strawman Solution: static trans-compiling**
  - Statically build an IR, then compile into stored procedures.

- **Challenge: Language dynamism**
  - The language feature that **alter program behaviors at run time** (e.g., Java's reflection), commonly used by web frameworks.

```
entityManager.find(Entity.class, id)
```

1. Issue `Select` with `Inner Join`
2. Issue `Select` with `Left Outer Join`
3. Defer `Select` until returned object is used
4. Do nothing
5. ...

  - As a result, we **cannot statically determine** what and when SQL statements will be issued at run time.

Opportunities

- **Concolic execution (concrete + symbolic execution)**
  - Able to **accurately analyze dynamic languages** like Java.
  - However, it only analyzes the program paths triggered by given inputs.

Opportunities

- **Concolic execution (concrete + symbolic execution)**
  - Able to **accurately analyze dynamic languages** like Java.
  - However, it only analyzes the program paths triggered by given inputs.

- **Pareto principle in program path hotness distribution**
  - Most requests are handled in a few distinct program paths.
    - In Broadleaf*, 2 hottest paths account for **96.3%** requests
  - By collecting inputs that trigger these hot paths, we can still **optimize for the most common cases**

* Broadleaf Commerce, 1.7k Stars, https://github.com/BroadleafCommerce

Key Idea

Synthesize and call stored procedures to cover hot path requests

Safely fall back to normal execution for cold path requests

Synthesizing Stored Procedures

```
IN: uid
① user := ORM.getUser(uid)                S1
② if user != null:
③   cart := ORM.getCart(user.cart_id)     S2
```

Original Code

Synthesizing Stored Procedures

```
IN: uid
① user := ORM.getUser(uid)                S1
② if user != null:
③   cart := ORM.getCart(user.cart_id)     S2
```

Original Code

```
uid:     233
S1_ret: {id:233, …}
S2_ret: /* cart obj */
```

Trace (taking `if` branch)

Synthesizing Stored Procedures

```
IN: uid
① user := ORM.getUser(uid)                    S1
② if user != null:
③    cart := ORM.getCart(user.cart_id)        S2
```

Original Code

```
uid:     233
S1_ret:  {id:233, …}
S2_ret:  /* cart obj */
```

Trace (taking `if` branch)

Concolically Replay

SQL node (S1)

Template:	"Select … Where id={uid}"
Condition:	true
In:	uid
Out:	user

SQL node (S2)

Template:	"Select… id={user_cart_id}"
Condition:	user != null
In:	user_cart_id
Out:	cart

Dependency Graph

Synthesizing Stored Procedures

```
IN: uid
① user := ORM.getUser(uid)                    S1
② if user != null:
③    cart := ORM.getCart(user.cart_id)        S2
```

Original Code

```
uid:     233
S1_ret:  {id:233, …}
S2_ret:  /* cart obj */
```

Trace (taking `if` branch)

Concolically Replay

SQL node (S1)

Template:	"Select … Where id={uid}"
Condition:	true
In:	uid
Out:	user

SQL node (S2)

Template:	"Select… id={user_cart_id}"
Condition:	user != null
In:	user_cart_id
Out:	cart

Dependency Graph

Translate

```
if true:
  Select * Into @user From Users
        Where id=@uid
if @user != null:
  Select * Into @cart From Carts
        Where id=@user_cart_id
```

Stored Procedure

Integrating into Web Apps

Add-Cart API

```
IN: pid, uid


① item := ORM.getItem(pid)                    S1
② user := ORM.getUser(uid)                    S2
③ cart := ORM.getCart(user.cart_id)           S3
④ if not item.available:
⑤   log := new Log("Item Unavailable")
⑥   ORM.save(log)                             S4
⑦ else:
⑧   cart.items.append(item)
⑨   ORM.save(cart)                            S5
```

Stored Procedure Code

```
PROCEDURE (pid, uid) BEGIN
Select * Into @item_item_id, @item_available, …  S1
         From Items Where product_id=@pid
Select * Into @userid_user, @cart_id_user, …     S2
         From Users Where id=@uid
Select * Into @cart_cart_id, …                   S3
         From Carts Where id=@cart_id_user
if @item_available = true:
  Insert Into Items Values …                     S5
END
```

Integrating into Web Apps

Add-Cart API

```
IN: pid, uid

① item := ORM.getItem(pid)                    S1
② user := ORM.getUser(uid)                    S2
③ cart := ORM.getCart(user.cart_id)           S3
④ if not item.available:
⑤   log := new Log("Item Unavailable")
⑥   ORM.save(log)                             S4
⑦ else:
⑧   cart.items.append(item)
⑨   ORM.save(cart)                            S5
```

Interception layer

Result Buffer

Stored Procedure Code

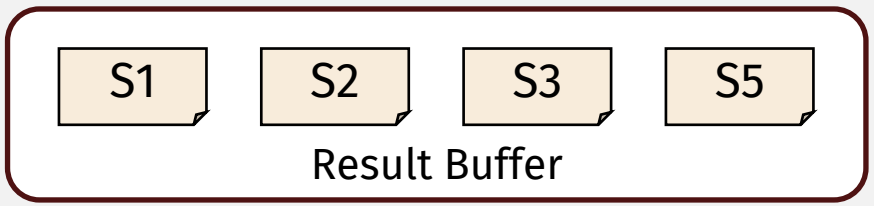
```
PROCEDURE (pid, uid) BEGIN
Select * Into @item_item_id, @item_available, …  S1
         From Items Where product_id=@pid
Select * Into @userid_user, @cart_id_user, …     S2
         From Users Where id=@uid
Select * Into @cart_cart_id, …                   S3
         From Carts Where id=@cart_id_user
if @item_available = true:
  Insert Into Items Values …                     S5
END
```

Integrating into Web Apps

Add-Cart API

```
IN: pid, uid
    RT 1: call PROCEDURE with pid and uid
① item := ORM.getItem(pid)                    S1
② user := ORM.getUser(uid)                    S2
③ cart := ORM.getCart(user.cart_id)           S3
④ if not item.available:
⑤   log := new Log("Item Unavailable")
⑥   ORM.save(log)                             S4
⑦ else:
⑧   cart.items.append(item)
⑨   ORM.save(cart)                            S5
```

Interception layer



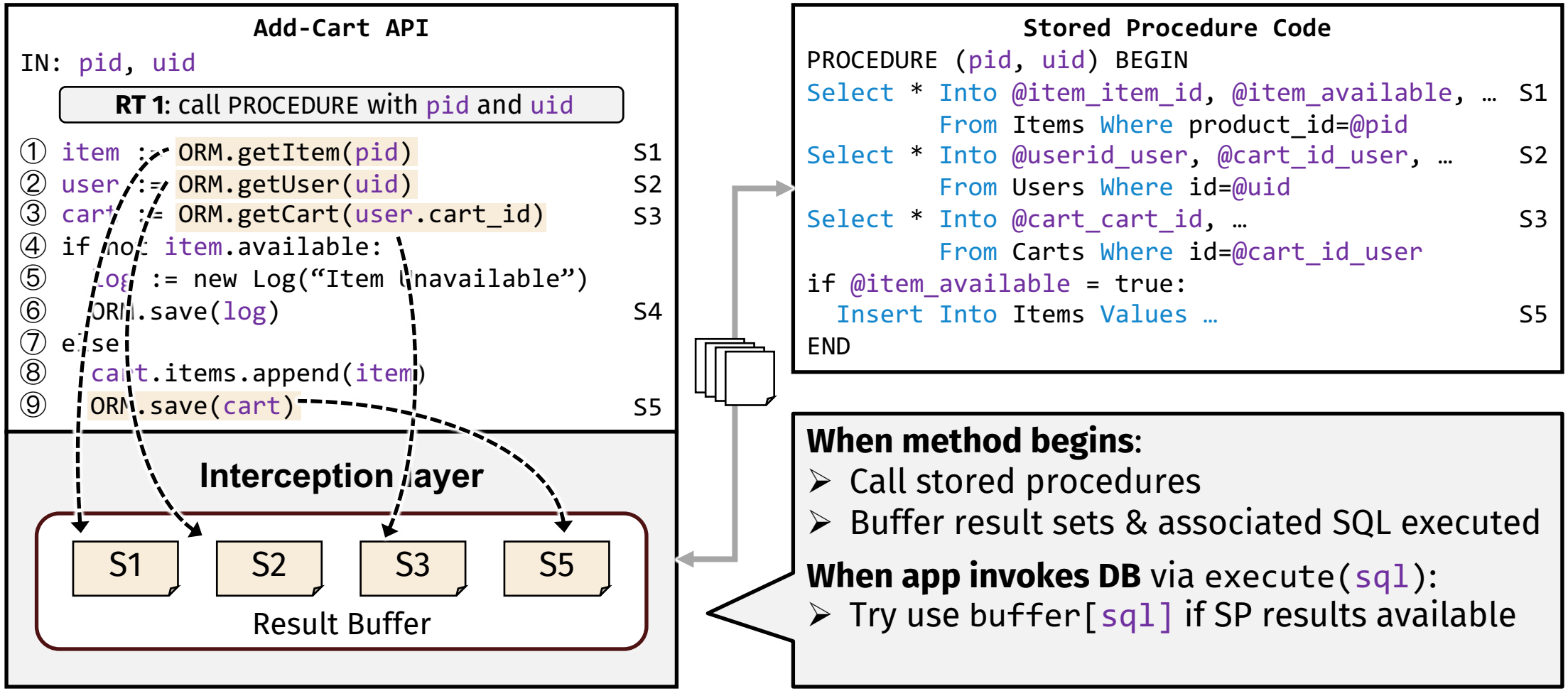
Stored Procedure Code

```
PROCEDURE (pid, uid) BEGIN
Select * Into @item_item_id, @item_available, …  S1
         From Items Where product_id=@pid
Select * Into @userid_user, @cart_id_user, …     S2
         From Users Where id=@uid
Select * Into @cart_cart_id, …                   S3
         From Carts Where id=@cart_id_user
if @item_available = true:
  Insert Into Items Values …                     S5
END
```

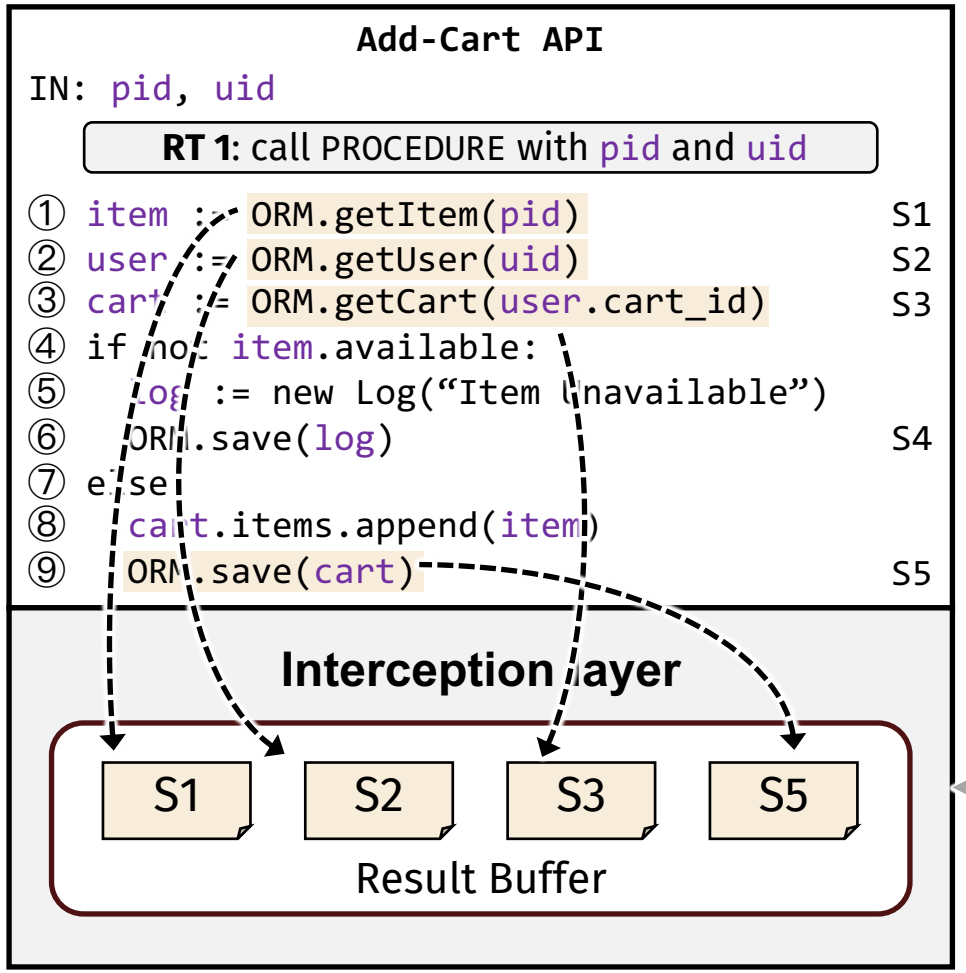
When method begins:

- Call stored procedures
- Buffer result sets & associated SQL executed

Integrating into Web Apps



Integrating into Web Apps



```
Stored Procedure Code
PROCEDURE (pid, uid) BEGIN
Select * Into @item_item_id, @item_available, … S1
         From Items Where product_id=@pid
Select * Into @userid_user, @cart_id_user, …    S2
         From Users Where id=@uid
Select * Into @cart_cart_id, …                  S3
         From Carts Where id=@cart_id_user
if @item_available = true:
  Insert Into Items Values …                    S5
END
```

When method begins:

- Call stored procedures
- Buffer result sets & associated SQL executed

When app invokes DB via `execute(sql)`:

- Try use `buffer[sql]` if SP results available
- On miss, send `sql` to DB & clean buffer

Handling Cold Path Requests

Add-Cart API

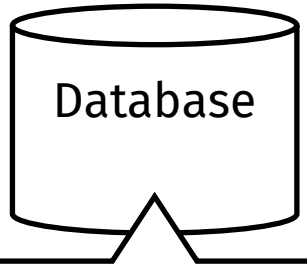
```
IN: pid, uid
```

RT 1: call PROCEDURE with pid and uid

```
① item := ORM.getItem(pid)                    S1
② user := ORM.getUser(uid)                    S2
③ cart := ORM.getCart(user.cart_id)           S3
④ if not item.available:
⑤   log := new Log("Item Unavailable")
⑥   ORM.save(log)                             S4
⑦ else:
⑧   cart.items.append(item)
⑨   ORM.save(cart)                            S5
```

Interception layer

Result Buffer



Stored Procedure Code

```
PROCEDURE (pid, uid) BEGIN
Select * Into @item_item_id, @item_available, …  S1
         From Items Where product_id=@pid
Select * Into @userid_user, @cart_id_user, …     S2
         From Users Where id=@uid
Select * Into @cart_cart_id, …                   S3
         From Carts Where id=@cart_id_user
if @item_available = true:
  Insert Into Items Values …                     S5
END
```

Handling Cold Path Requests

Request adding unavailable item

Add-Cart API

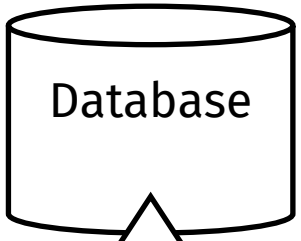
```
IN: pid, uid
```

RT 1: call PROCEDURE with pid and uid

```
① item := ORM.getItem(pid)                    S1
② user := ORM.getUser(uid)                    S2
③ cart := ORM.getCart(user.cart_id)           S3
④ if not item.available:
⑤   log := new Log("Item Unavailable")
⑥   ORM.save(log)                             S4
⑦ else:
⑧   cart.items.append(item)
⑨   ORM.save(cart)                            S5
```

Interception layer

Result Buffer

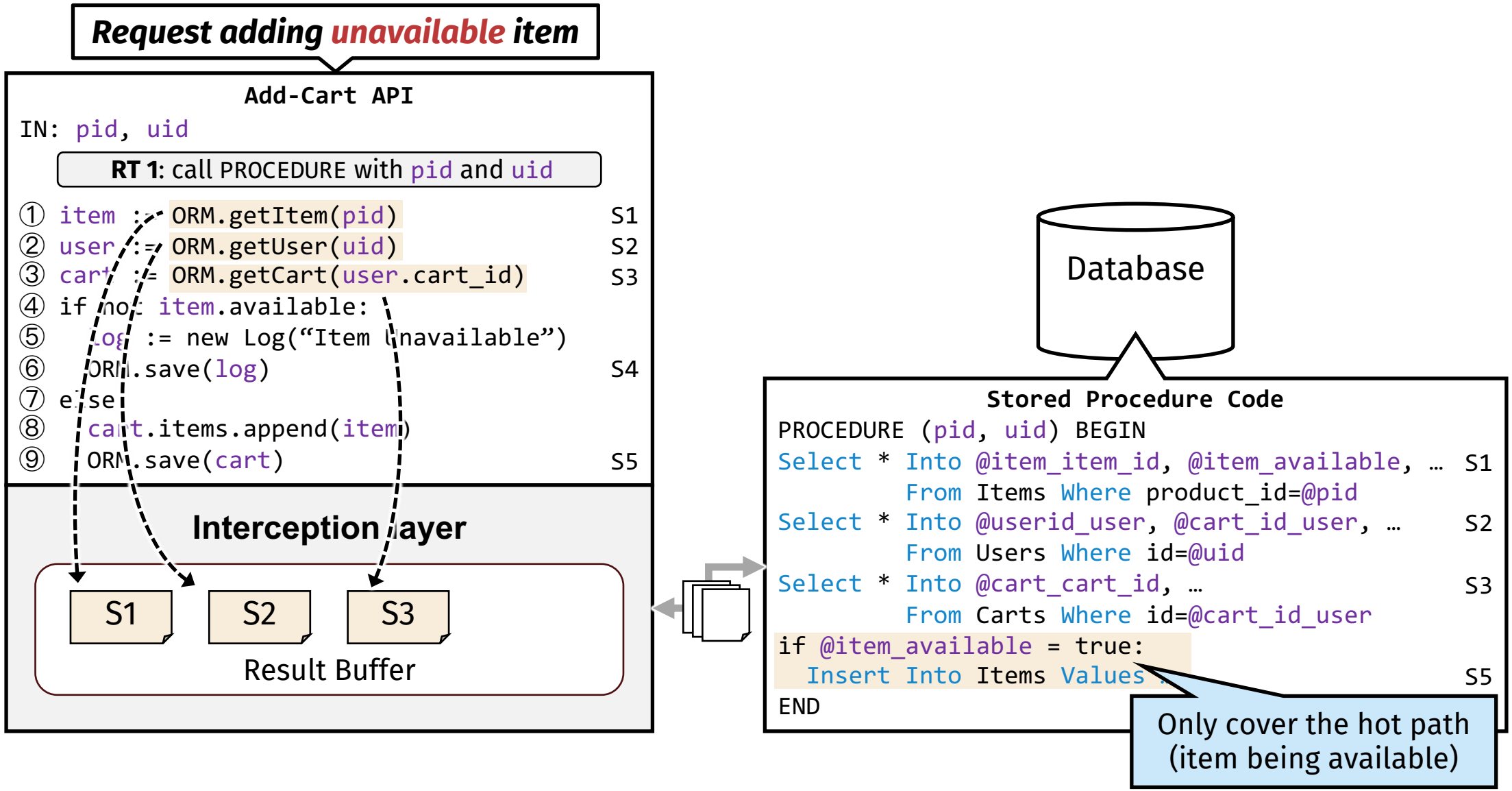


Stored Procedure Code

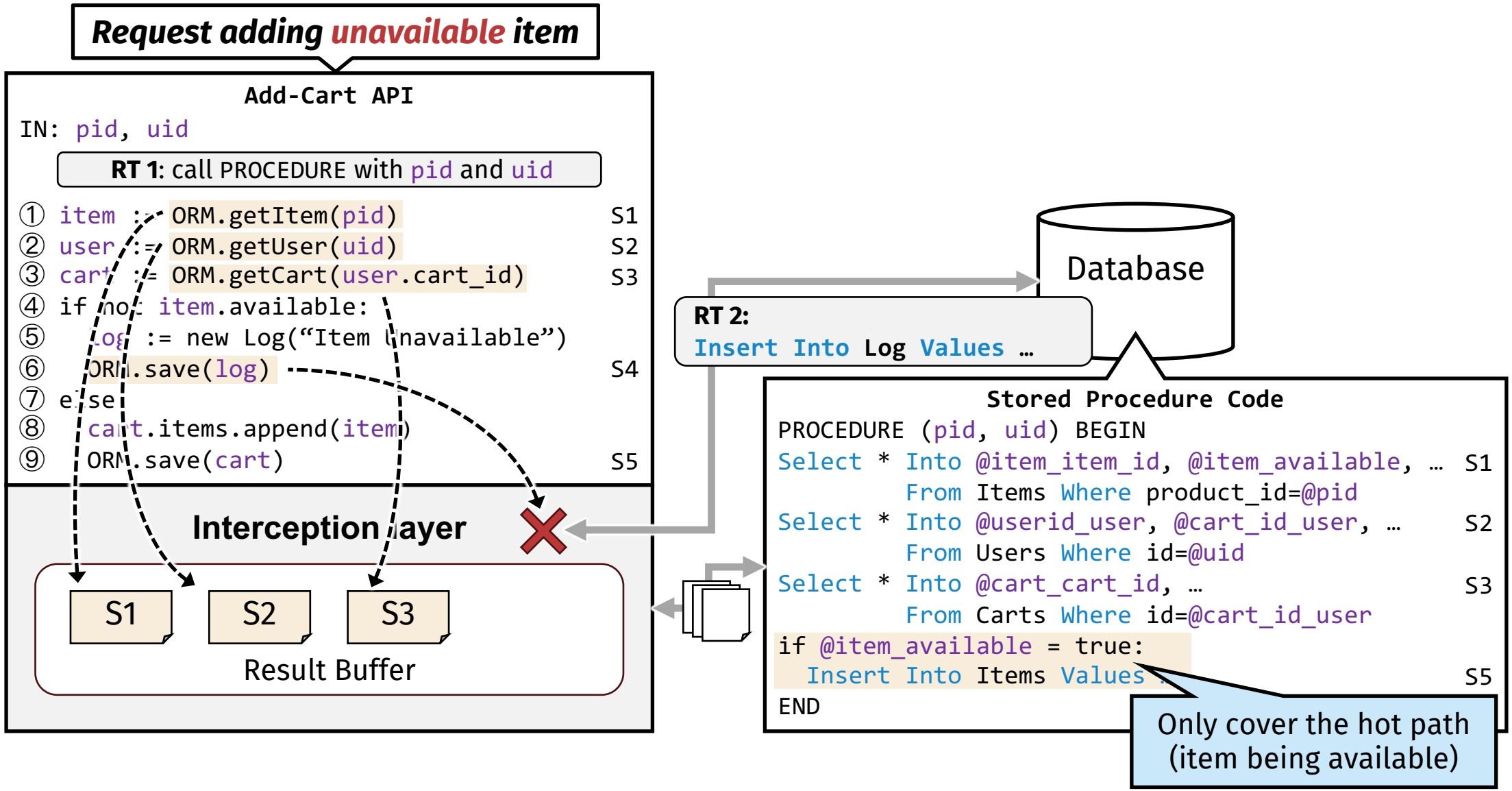
```
PROCEDURE (pid, uid) BEGIN
Select * Into @item_item_id, @item_available, …  S1
         From Items Where product_id=@pid
Select * Into @userid_user, @cart_id_user, …     S2
         From Users Where id=@uid
Select * Into @cart_cart_id, …                   S3
         From Carts Where id=@cart_id_user
if @item_available = true:
  Insert Into Items Values …                     S5
END
```

Only cover the hot path (item being available)

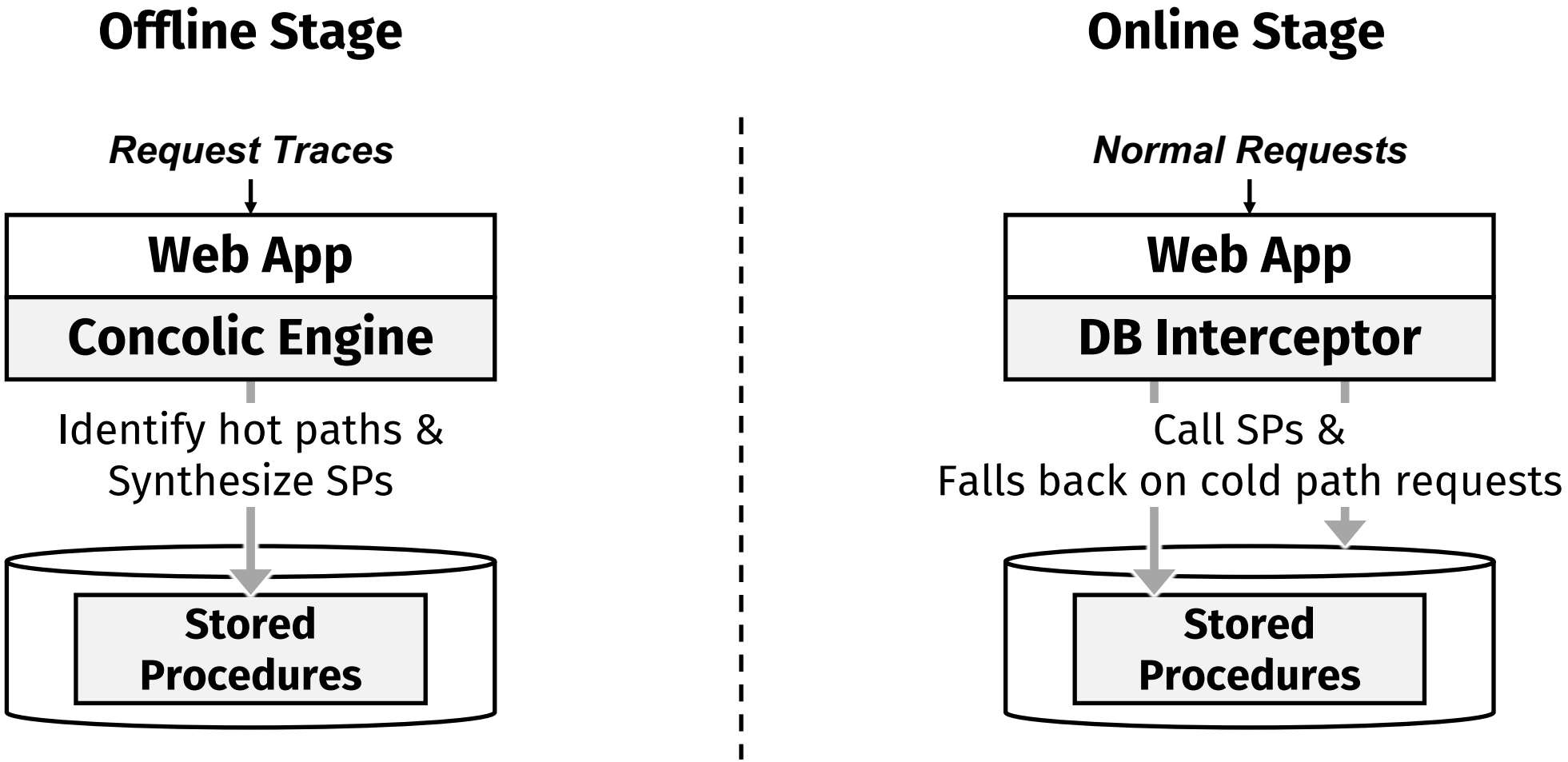
Handling Cold Path Requests



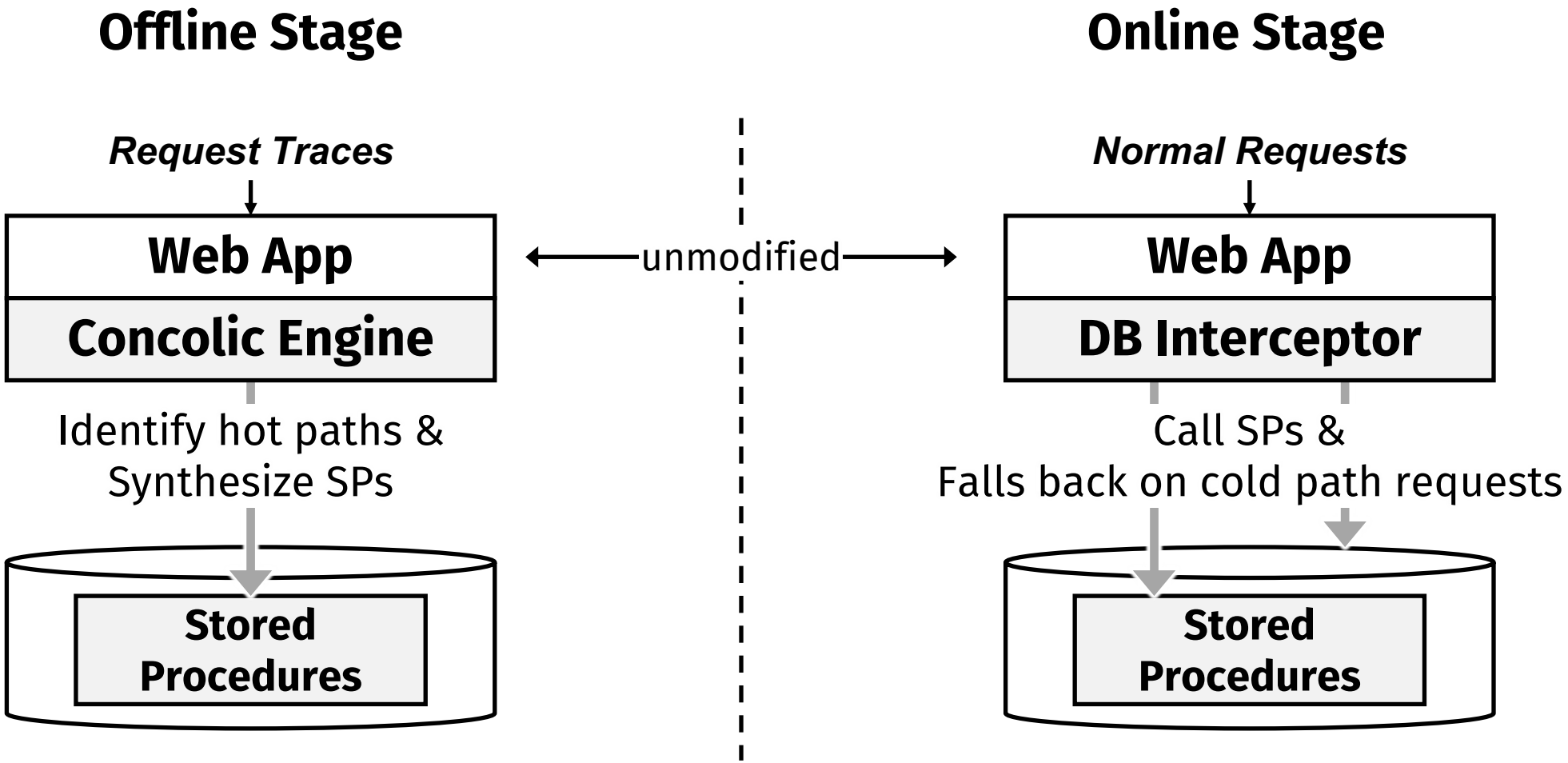
Handling Cold Path Requests



System Overview

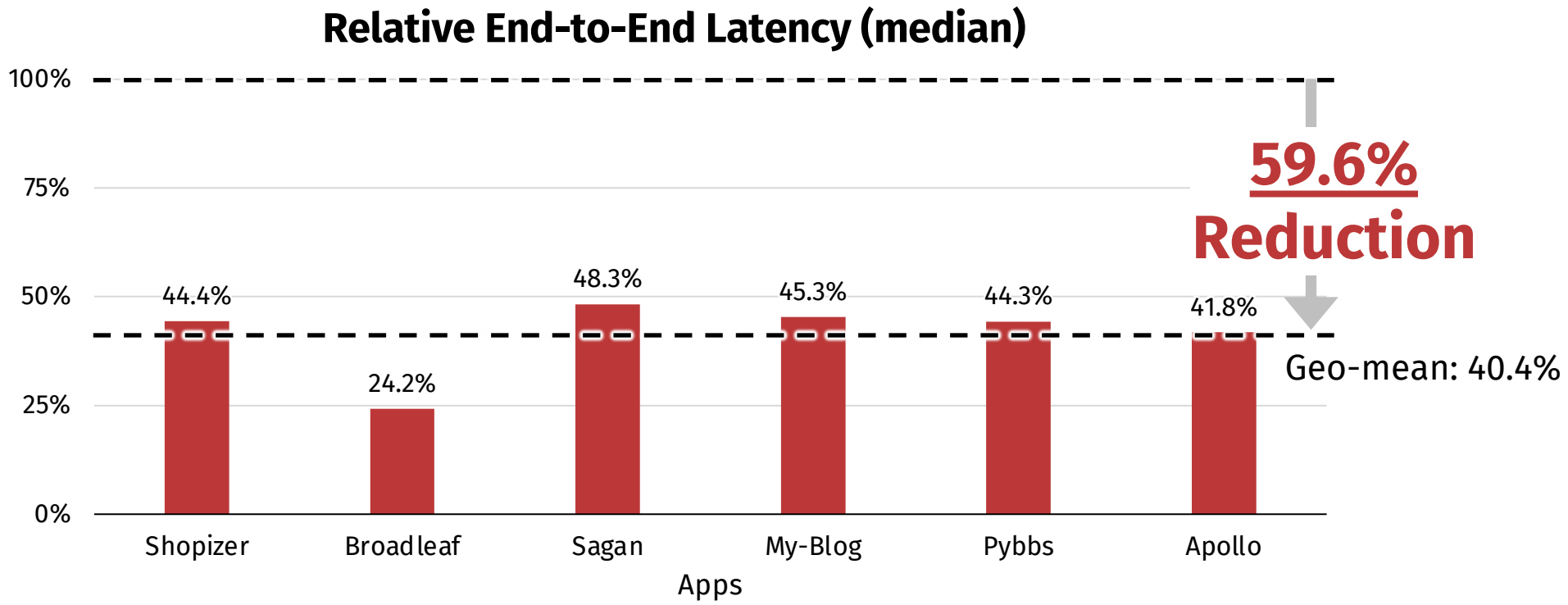


System Overview



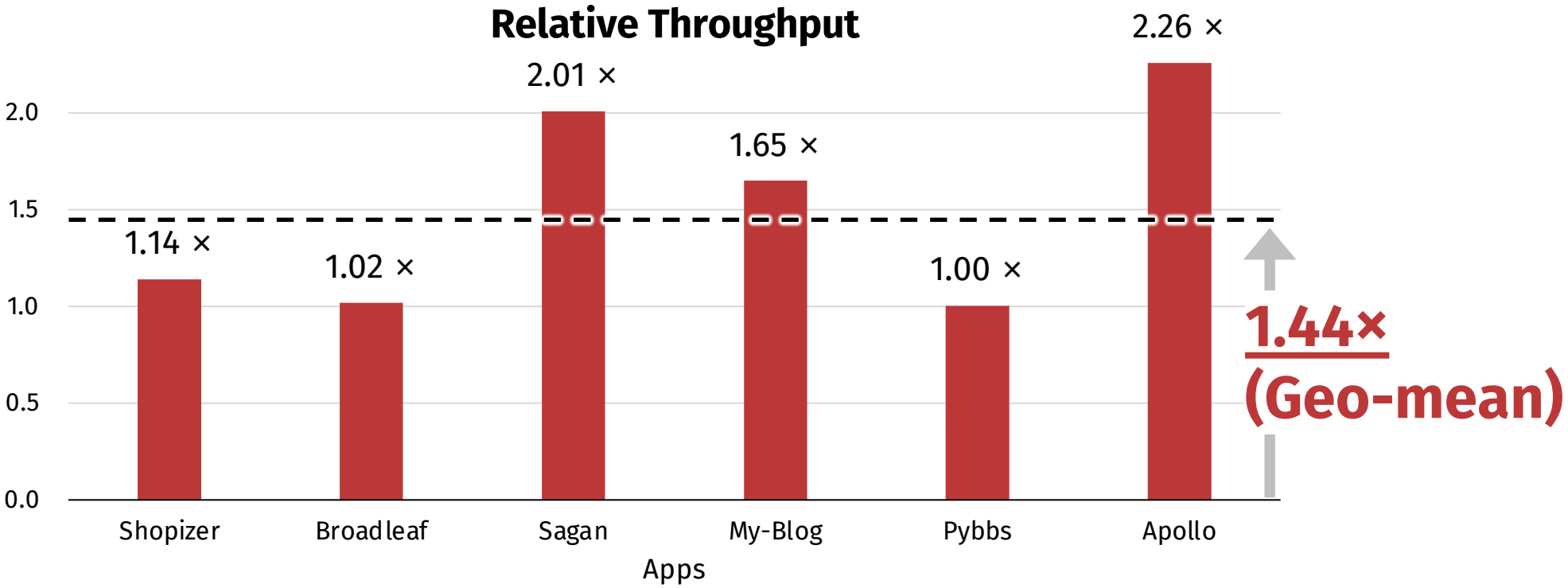
Evaluation: Latency

- **Over 6 open-source apps with realistic workloads. (6.8k ★/app)**
- **WeBridge achieves 59.6% end-to-end latency reduction.**



Evaluation: Throughput

- **WeBridge can also increase near half throughput, by**
  - Avoid DB repeatedly parsing interactive statements; and
  - Shorten lock-holding time of conflicting transactions



Summary and Q/A

- **WeBridge synthesize stored procedures to pre-execute SQL statements to reduce DB round trips.**
- **Using concolic execution, data & control dependencies are accurately shipped to DB.**
- **For end-users, WeBridge reduces 59.6% latency and achieves 1.44× relative throughput.**

- **Check out our paper and source code!**
  - Design details; optimizations; correctness proof, etc.

Paper

Code