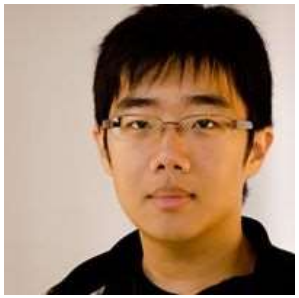
Finding and Evaluating the Performance Impact of Redundant Data Access for Applications that are Developed Using Object-Relational Mapping Frameworks

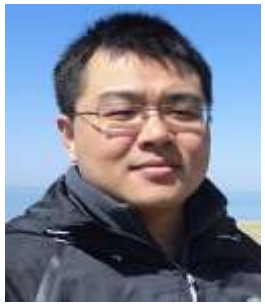


Tse-Hsun(Peter) Chen

Université Concordia University

Weiyi Shang

Université Concordia University



Zhen Ming Jiang

York University

Ahmed E. Hassan

Queen's University

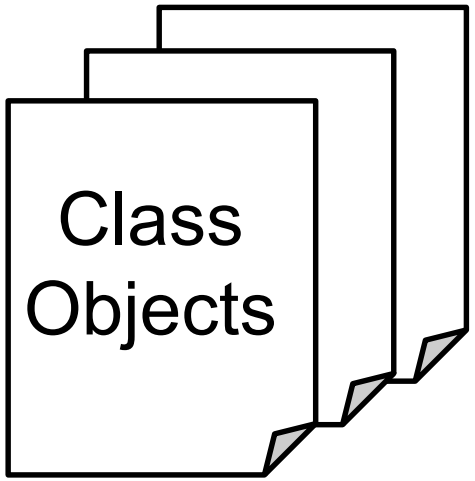
Mohamed Nasser, Parminder Flora

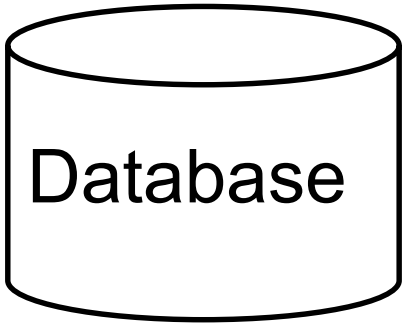
BlackBerry

$1.6 billion revenue loss for a one-second slowdown

Slow database access is often the performance bottleneck

Object-Relational Mapping abstracts relational database as objects





Benefits of using ORM

- Less **boilerplate code**
- Automatically **manage object-DB translations**

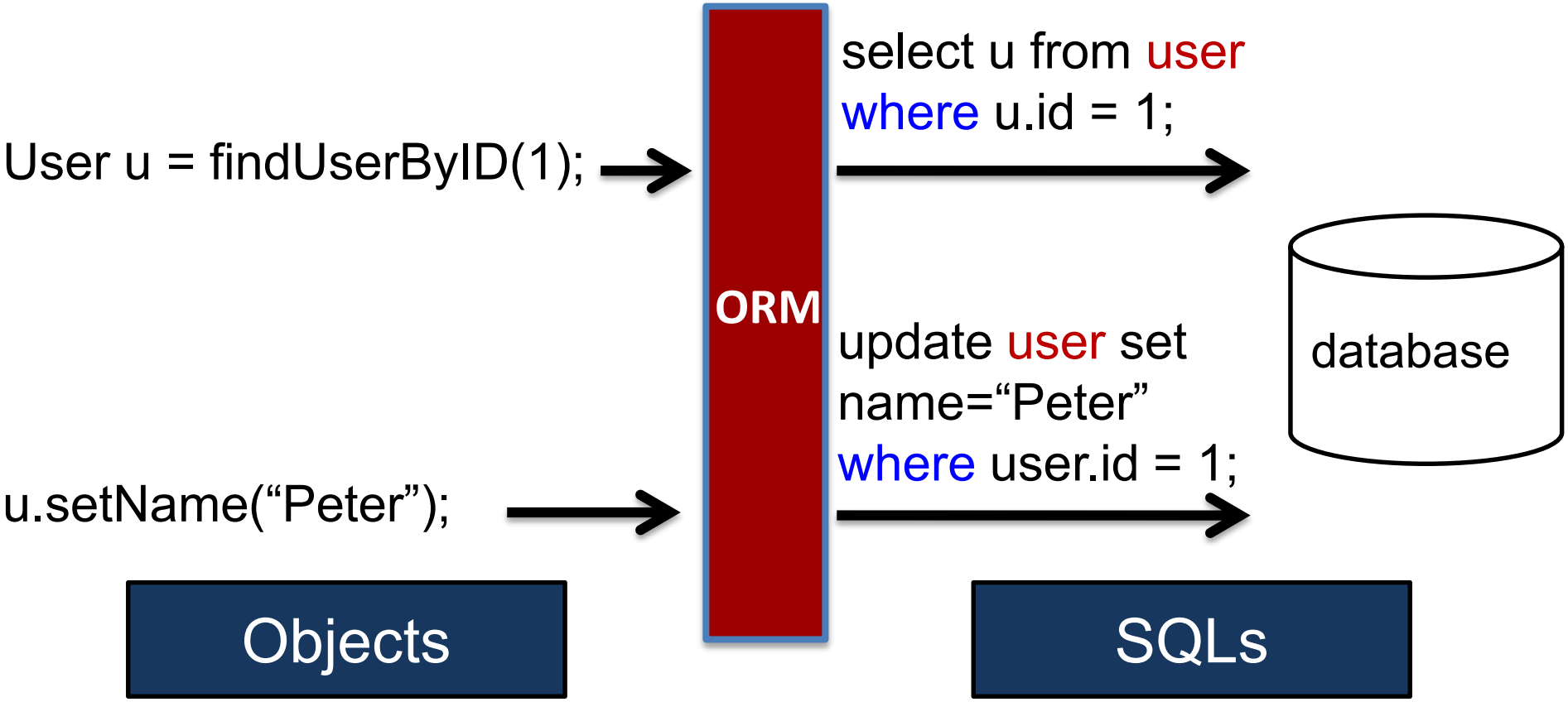
Using ORM results in much less code and shorter development time.

Over 67% of Java developers use Object-Relational Mapping (Hibernate) to access databases

Two most popular Java database access frameworks

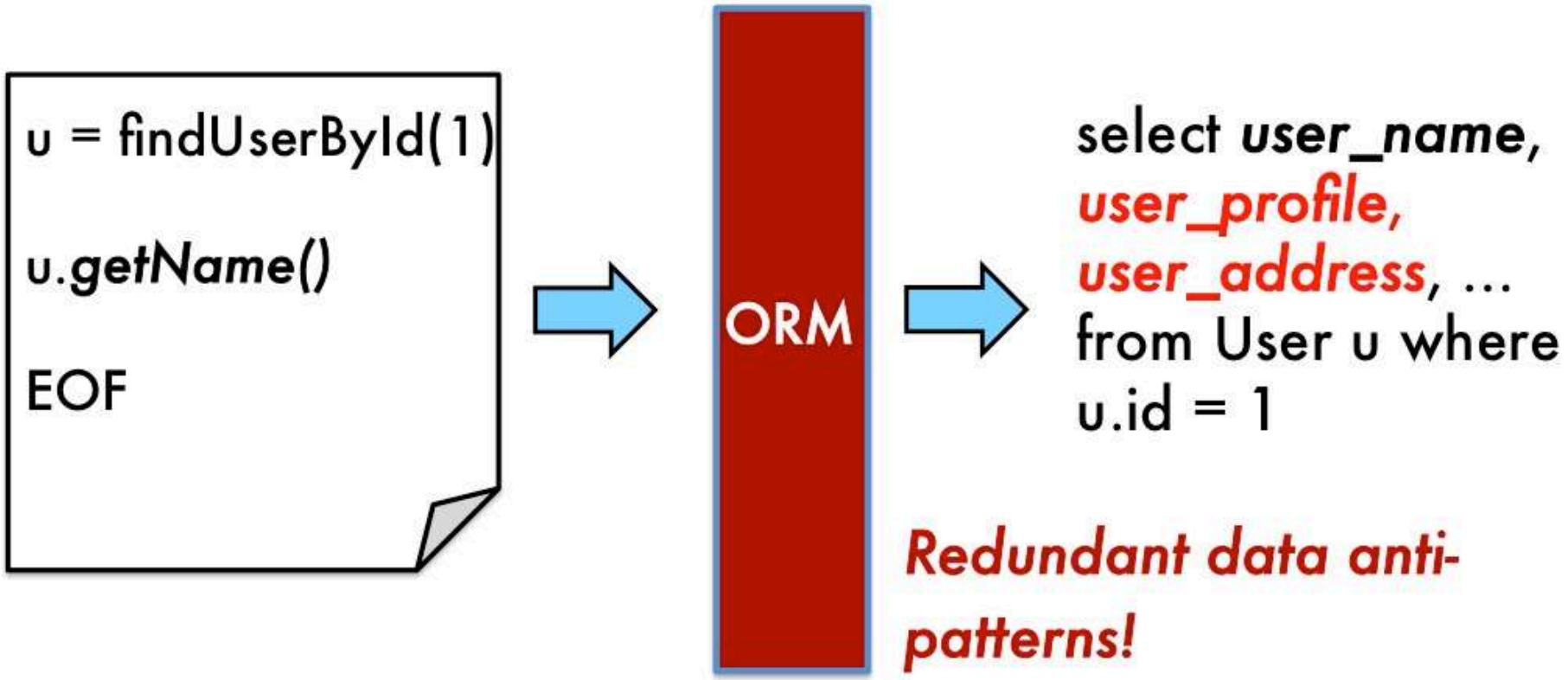
2001~Now	1997~Now
	
67%	22%

Accessing the database using ORM



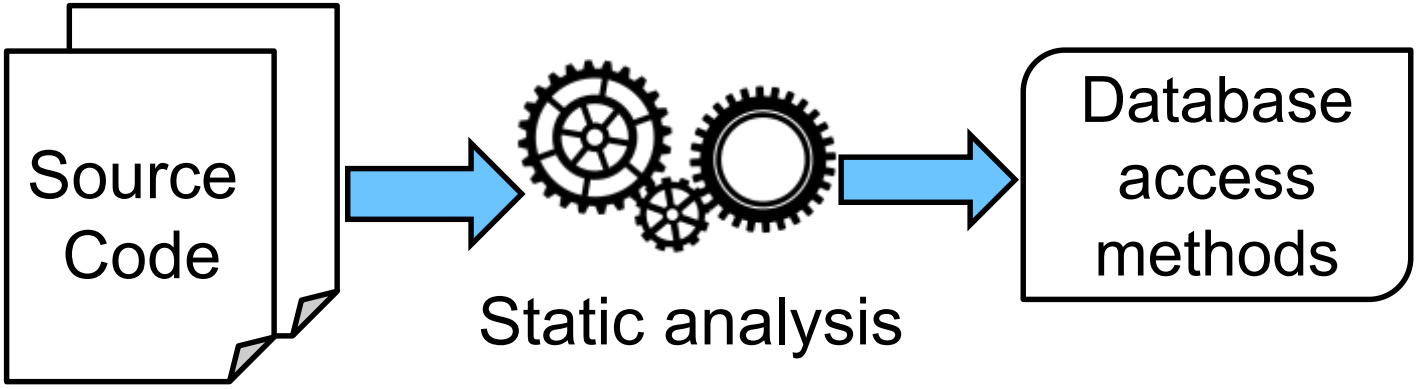
Redundant data is common across ORMs!

Redundant data is caused by ORM's lack of knowledge on the business logic

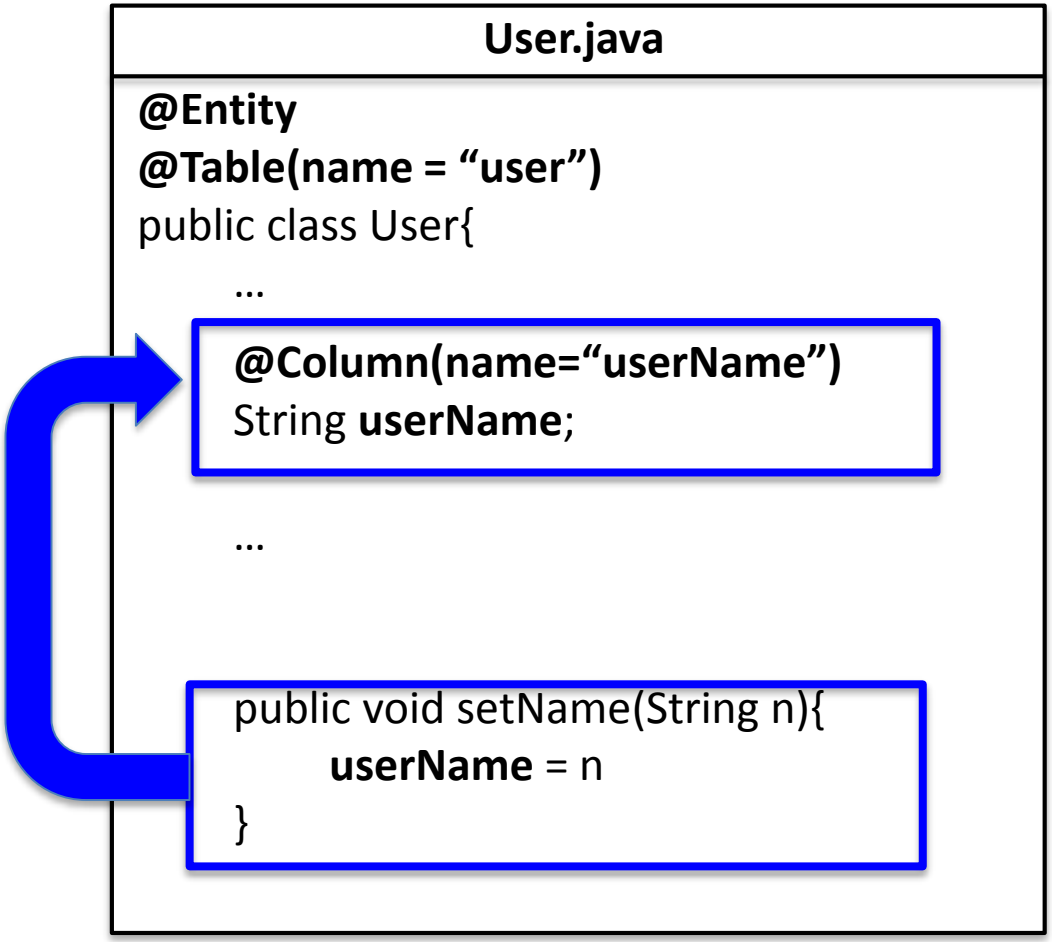


We need to understand how ORM APIs are used in the application in order to detect redundant data anti-patterns.

Our approach for finding redundant data anti-patterns

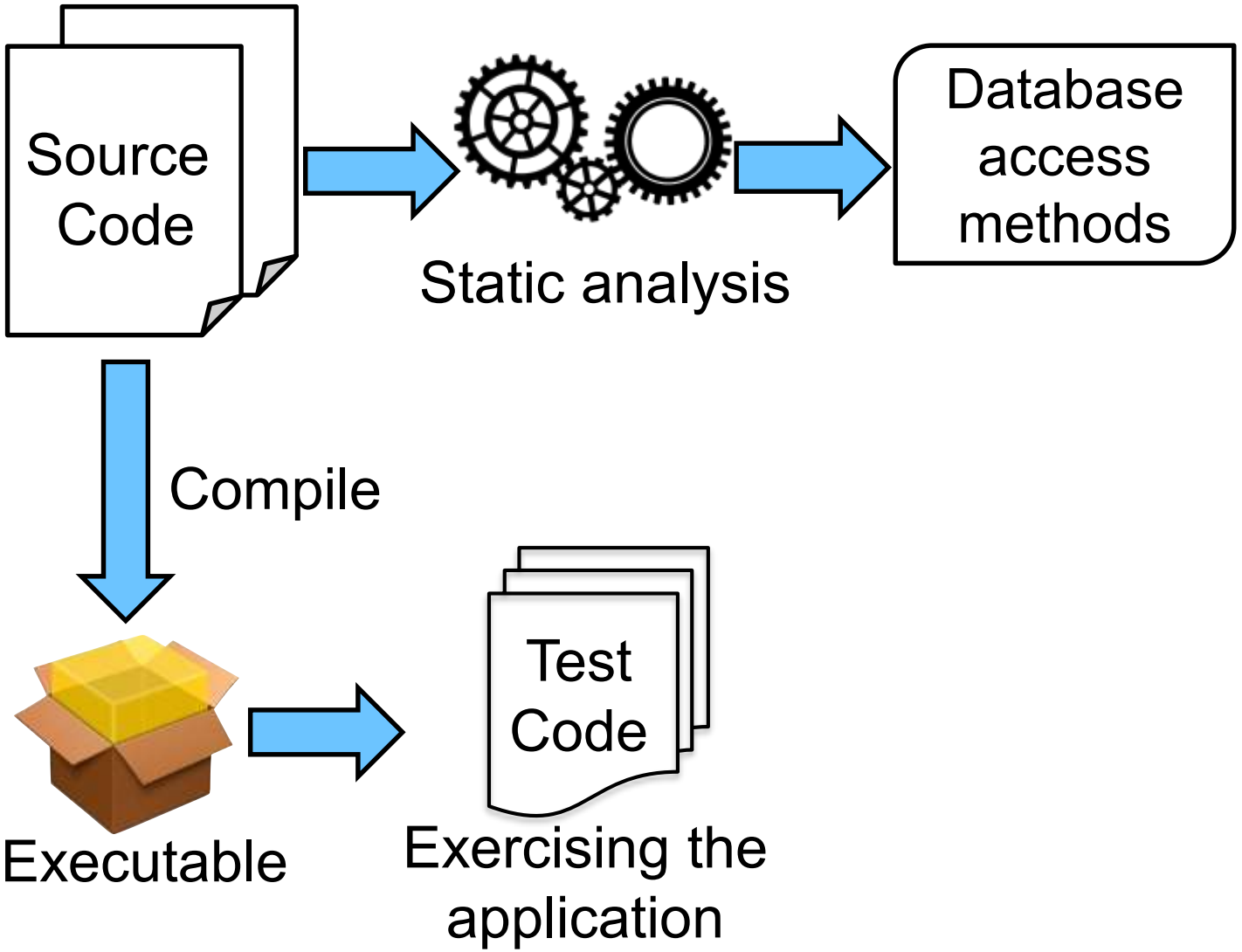


Finding database access methods

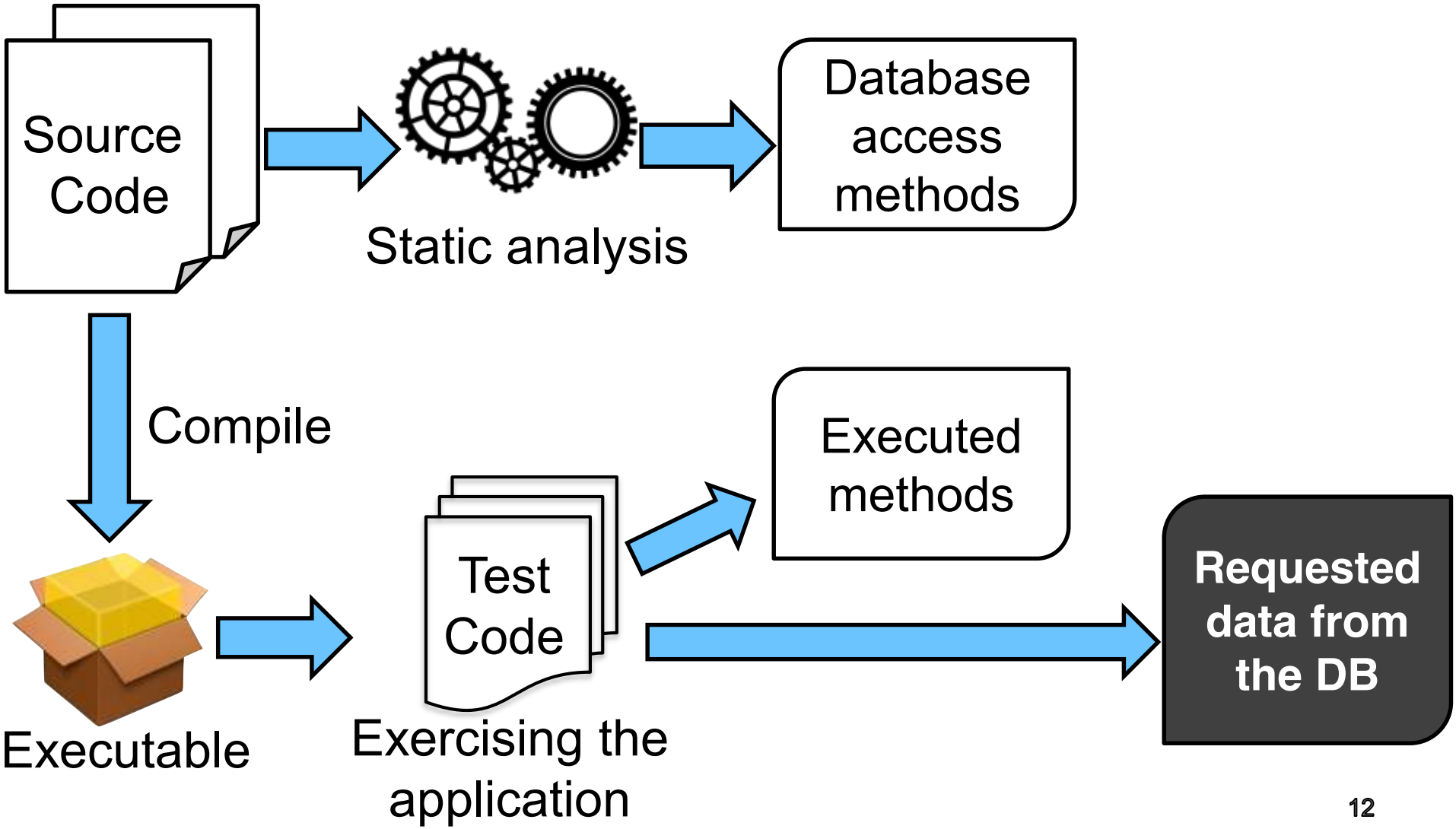


Using static analysis to find which database column a method is reading or modifying.

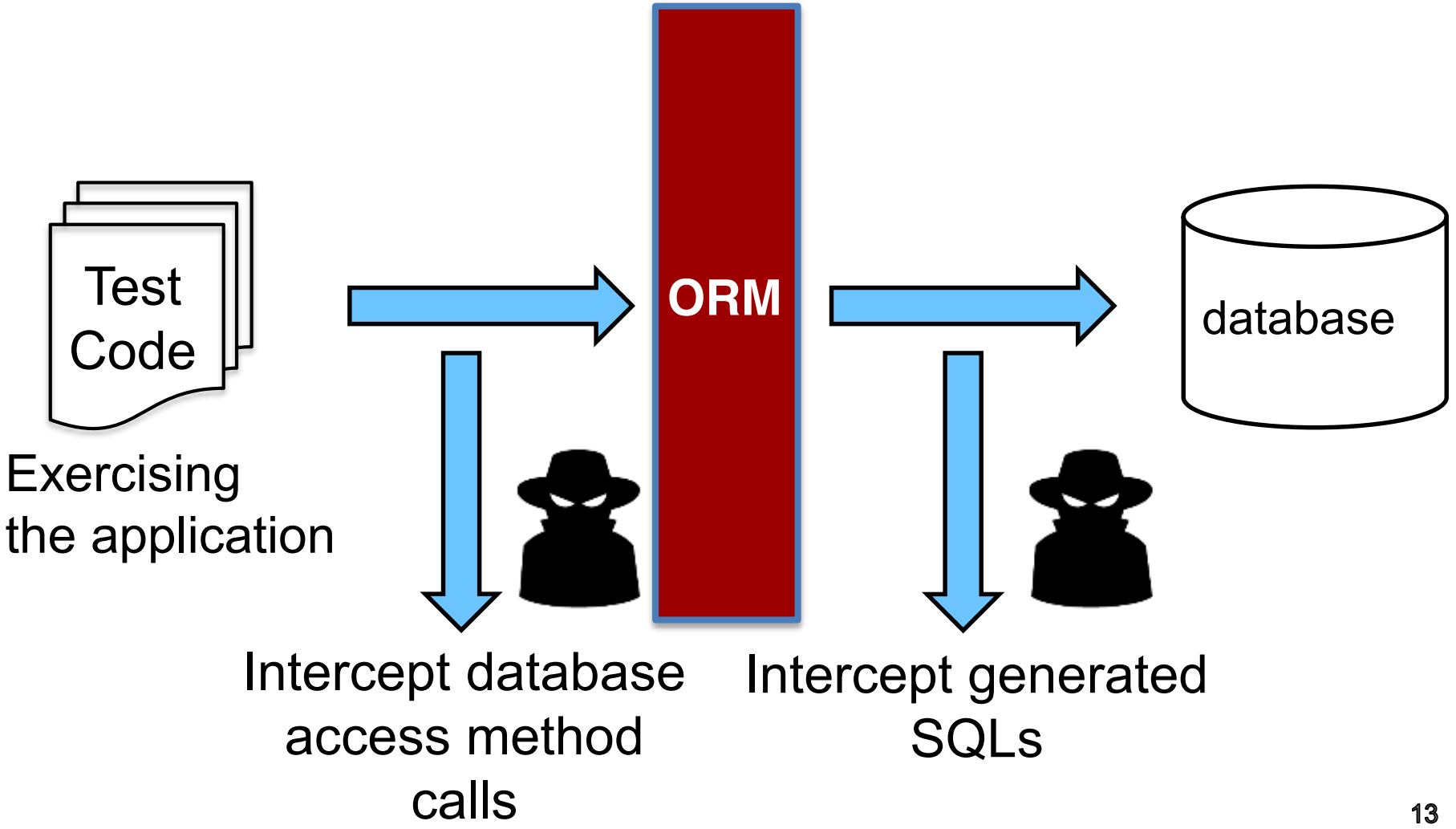
Our approach for finding redundant data anti-patterns



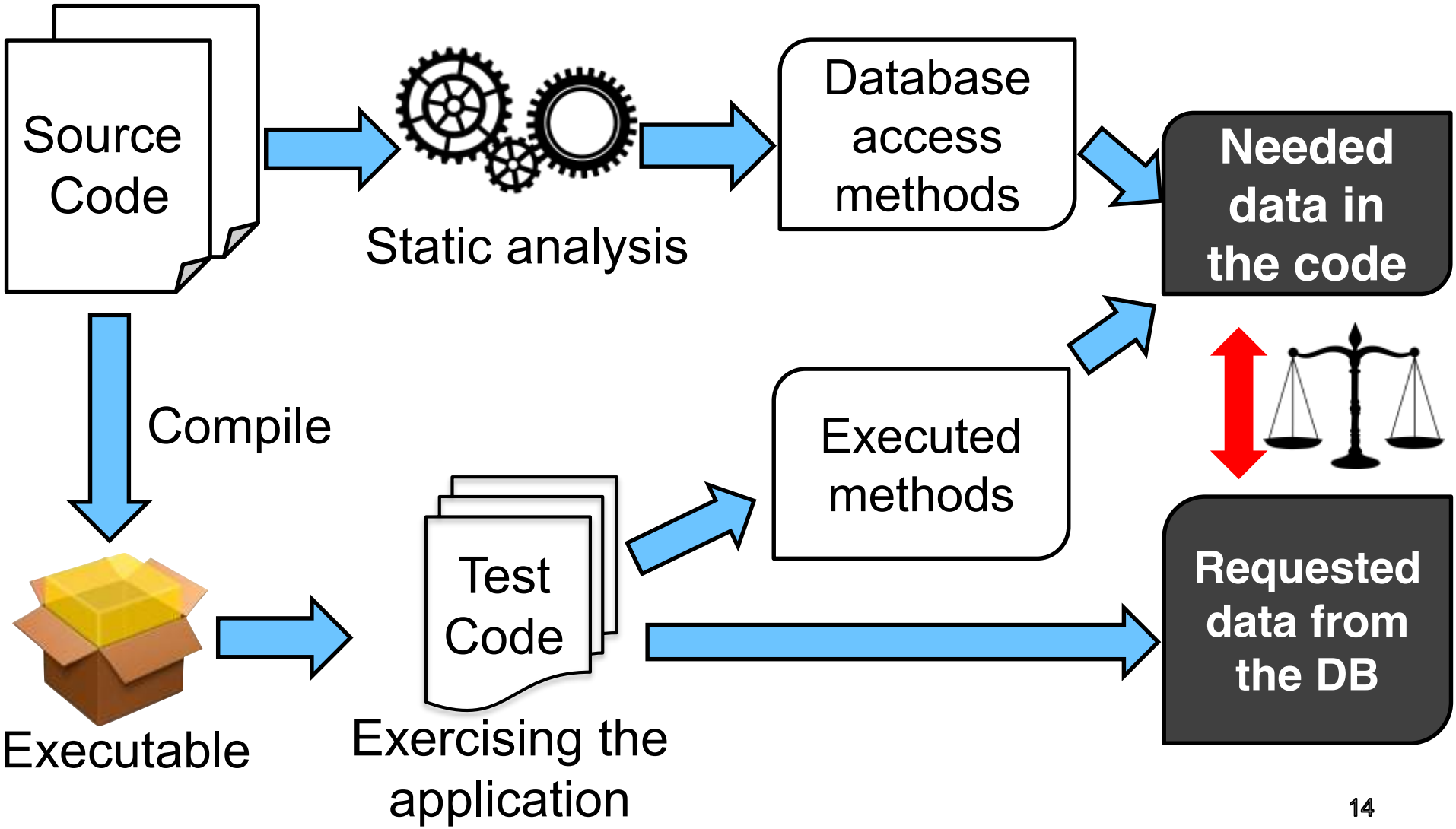
Our approach for finding redundant data anti-patterns



Understanding application execution using bytecode instrumentation



Our approach for finding redundant data anti-patterns



Finding redundant data anti-patterns by comparing needed and requested data

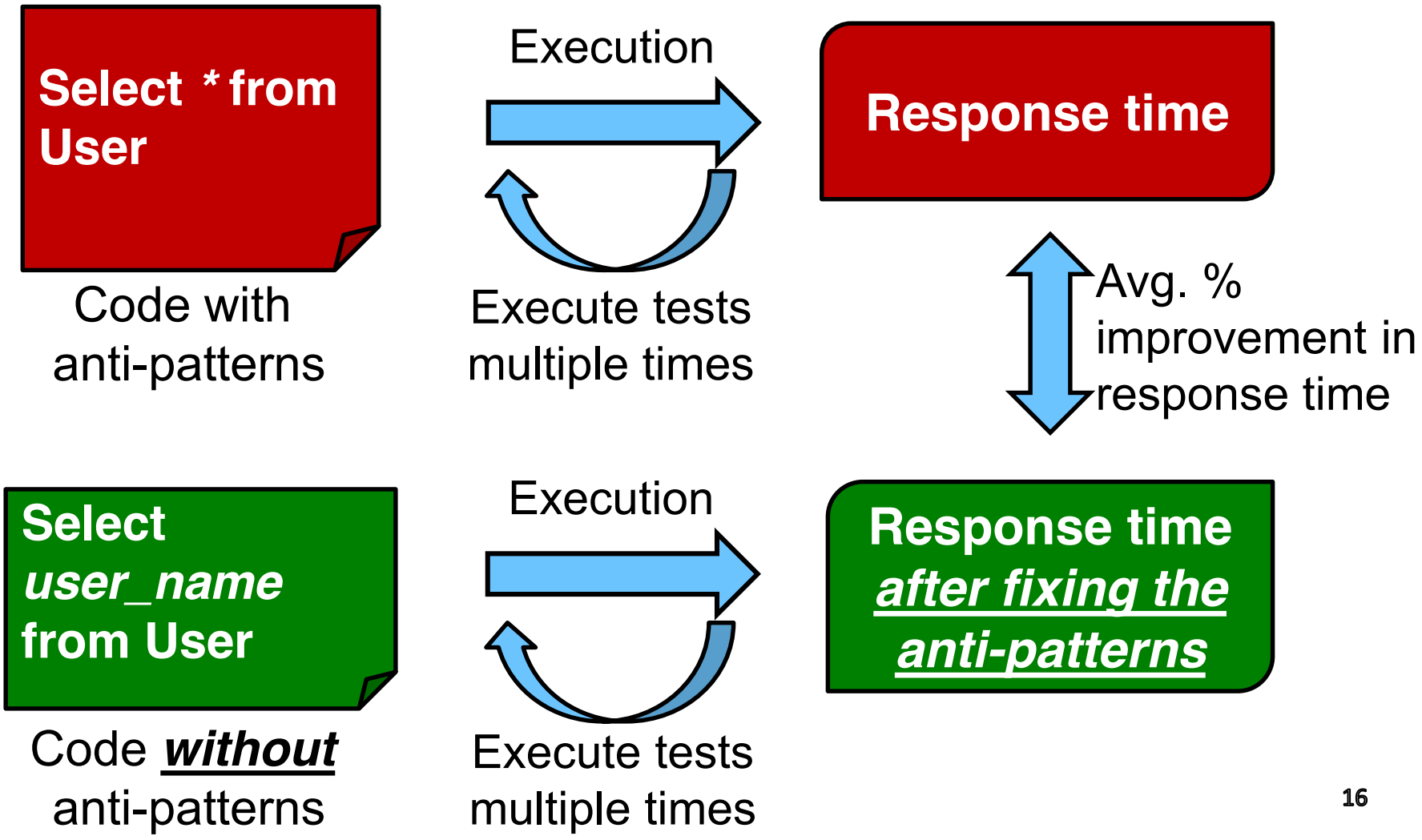
```
<transaction>
    <methodCall>
        user.getUserName()
    </methodCall>

    <ORM-generated SQL>
        select u.id, u.name, u.address, u.phone_number from
User u where u.id = 1
    </ORM-generated SQL>
</transaction>
```

Only needs *name* in the application logic.

The requested id, address, phone number are not needed.

We detect redundant data anti-patterns for all types of database access (e.g., select, update, join).

Assessing redundant data anti-pattern impact by fixing the anti-patterns



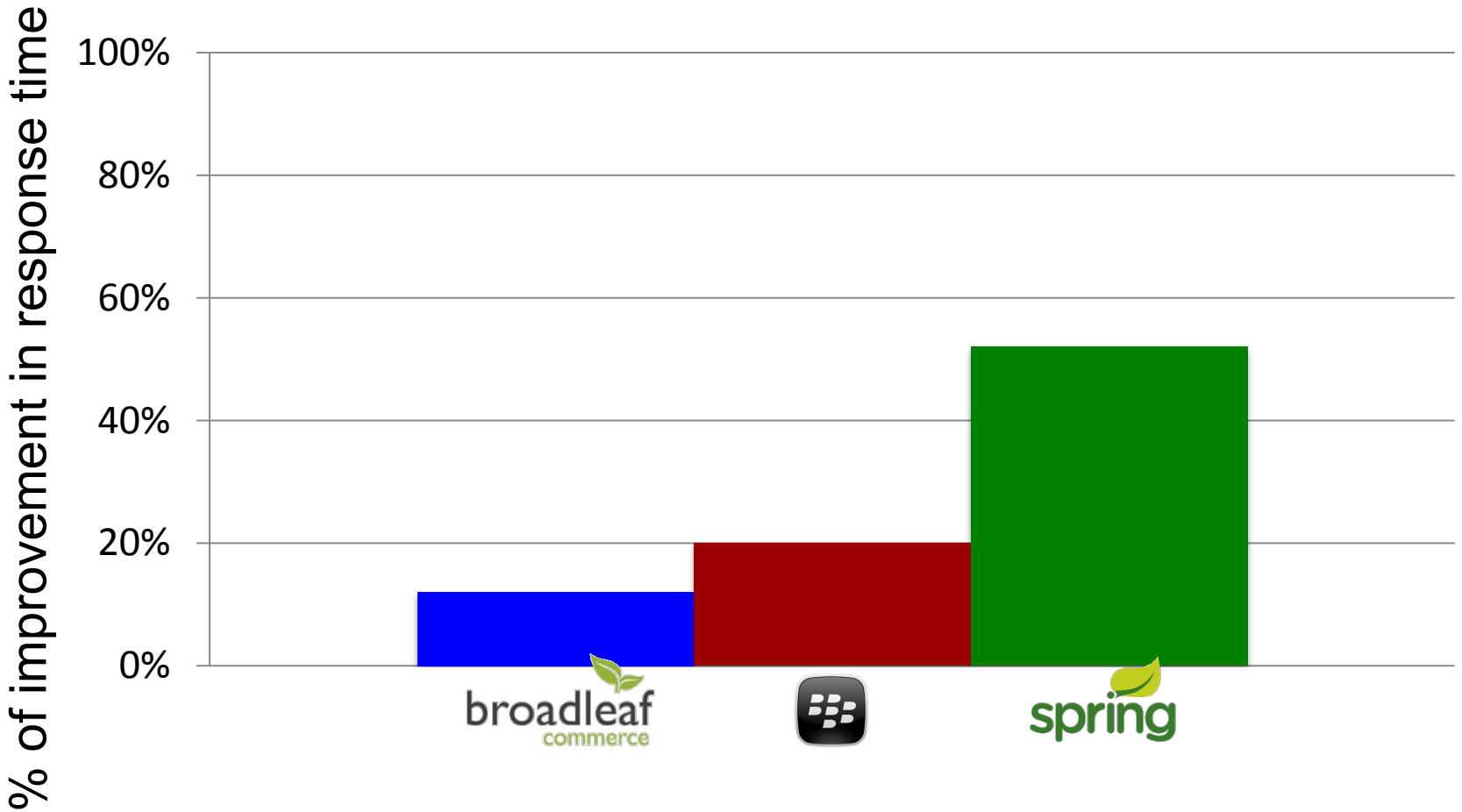
Studied applications

Large open-source
e-commence system
> 1,700 files
> 206K LOC

Enterprise application
> 5,000 files
> 10M LOC

Open-source pet clinic
application
51 files
3.3K LOC

Removing redundant data anti-patterns improves response time



Slow database access is often the performance bottleneck

Redundant data is caused by ORM's lack of knowledge on the business logic

```
u = findUserById(1)
u.getName()
EOF
```

ORM

```
select user_name,
user_profile,
user_address, ...
from User u where
u.id = 1
```

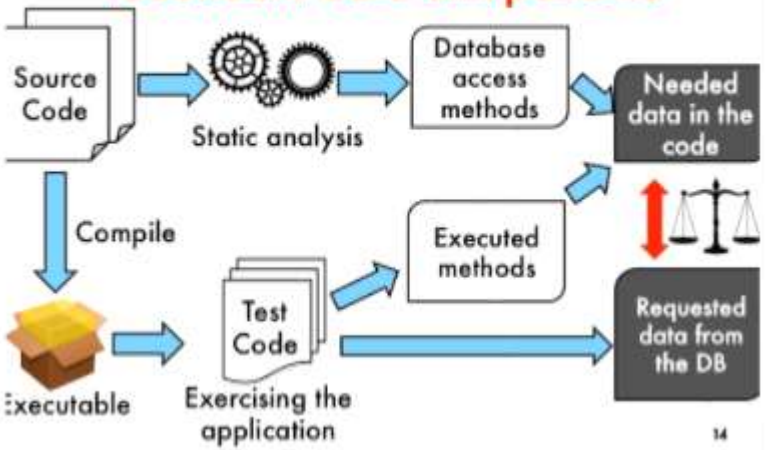
Redundant data anti-patterns!

We need to understand how ORM APIs are used in the application in order to detect redundant data anti-patterns.

http://petertsehsun.github.io/

@petertsehsun

Our approach for finding redundant data anti-patterns



Removing redundant data anti-patterns improves response time