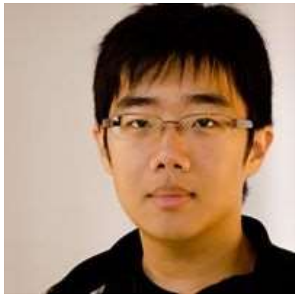
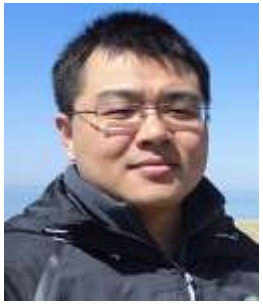
Detecting Performance Anti-patterns for Applications Developed Using Object-Relational Mapping





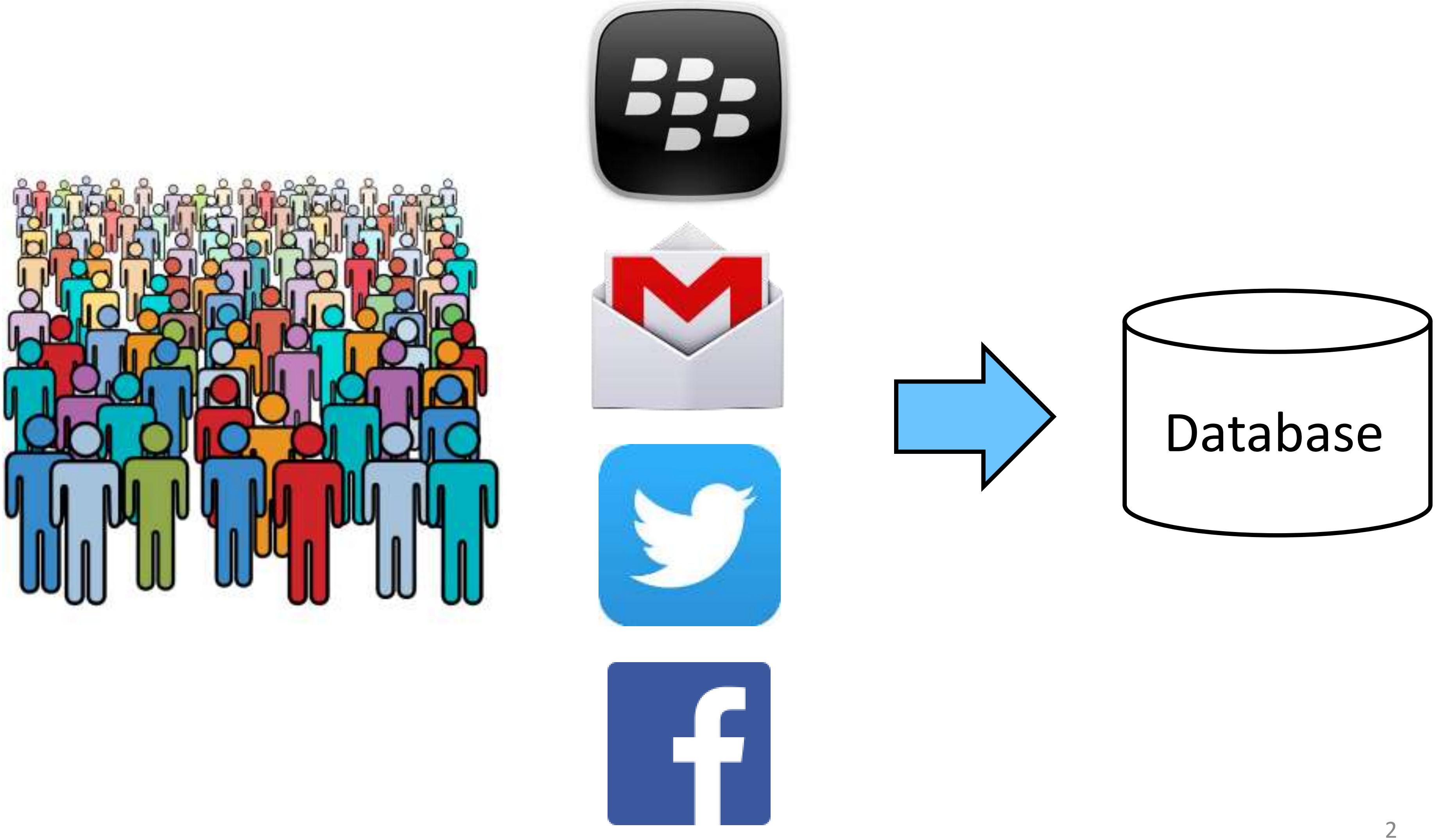
Tse-Hsun(Peter) Chen Weiyi Shang Zhen Ming Jiang Ahmed E. Hassan

Queen's University Queen's University York University Queen's University

Mohamed Nasser, Parminder Flora

BlackBerry

Databases are essential in large-scale software systems

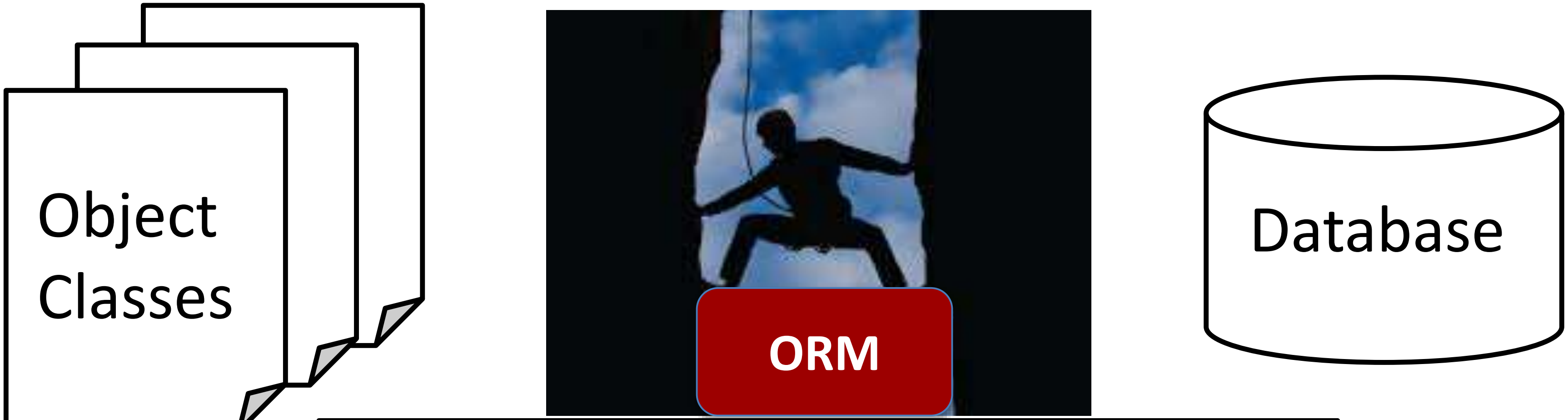


Database

Application developers work with objects

More intuitive if we can map objects directly to DB

Object-Relational Mapping eliminates the gap between objects and SQL



Much less code and shorter development time

Problem of using raw SQLs

- Lots of **boilerplate code**
- Need to **manage object-DB translations** manually

ORM is widely used in practice

HIBERNATE

- Java Hibernate has ***more than 8 million*** downloads
- In 2013, ***15% of the 17,000*** Java developer jobs require ORM experience (dice.com)

Different ORM technologies

OpenJPA

eclipselink

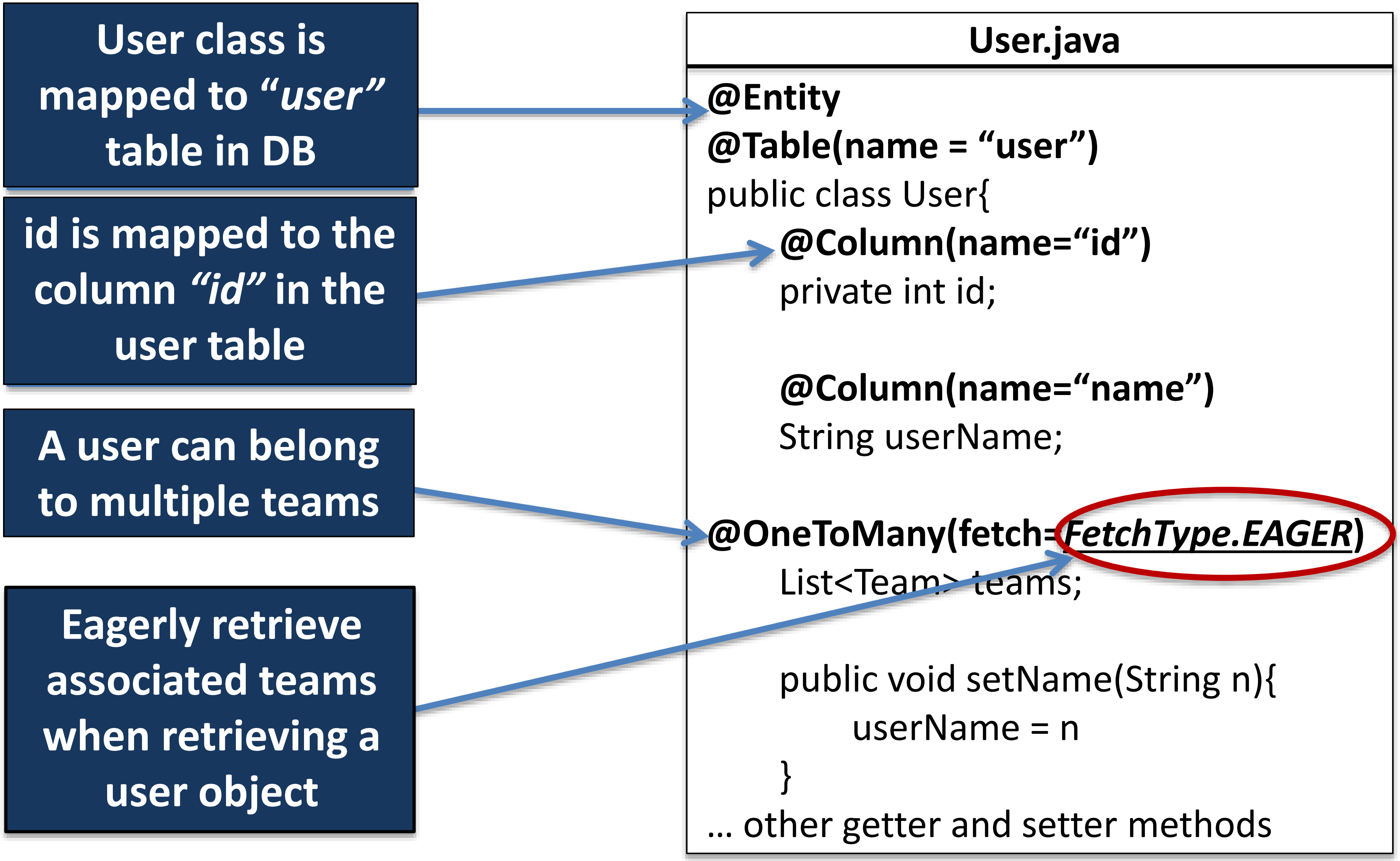
WebSphere

django

Rails

NHibernate

An example class with ORM code



User class is mapped to *"user"* table in DB

id is mapped to the column *"id"* in the user table

A user can belong to multiple teams

Eagerly retrieve associated teams when retrieving a user object

User.java

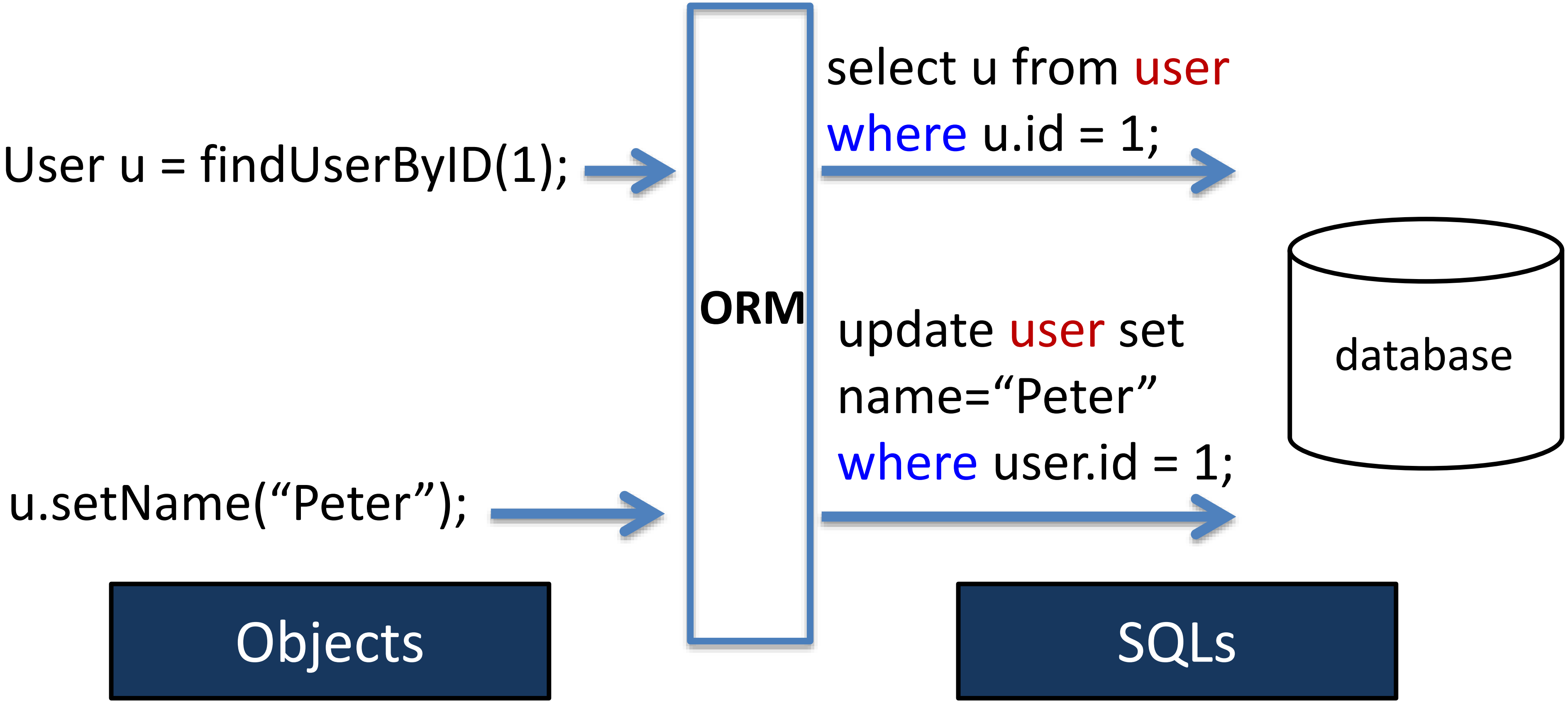
```
@Entity
@Table(name = "user")
public class User{
    @Column(name="id")
    private int id;

    @Column(name="name")
    String userName;

@OneToMany(fetch=FetchType.EAGER)
    List<Team> teams;

    public void setName(String n){
        userName = n
    }
... other getter and setter methods
```

Accessing the database using ORM



User u = findUserByID(1);

u.setName(“Peter”);

ORM

select u from user
where u.id = 1;

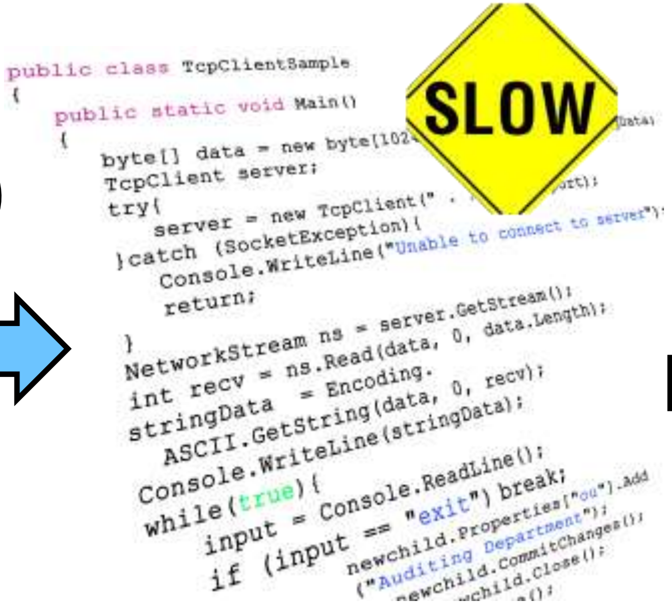
update user set
name=“Peter”
where user.id = 1;

database

Objects

SQLs

Developers may not be aware of database access



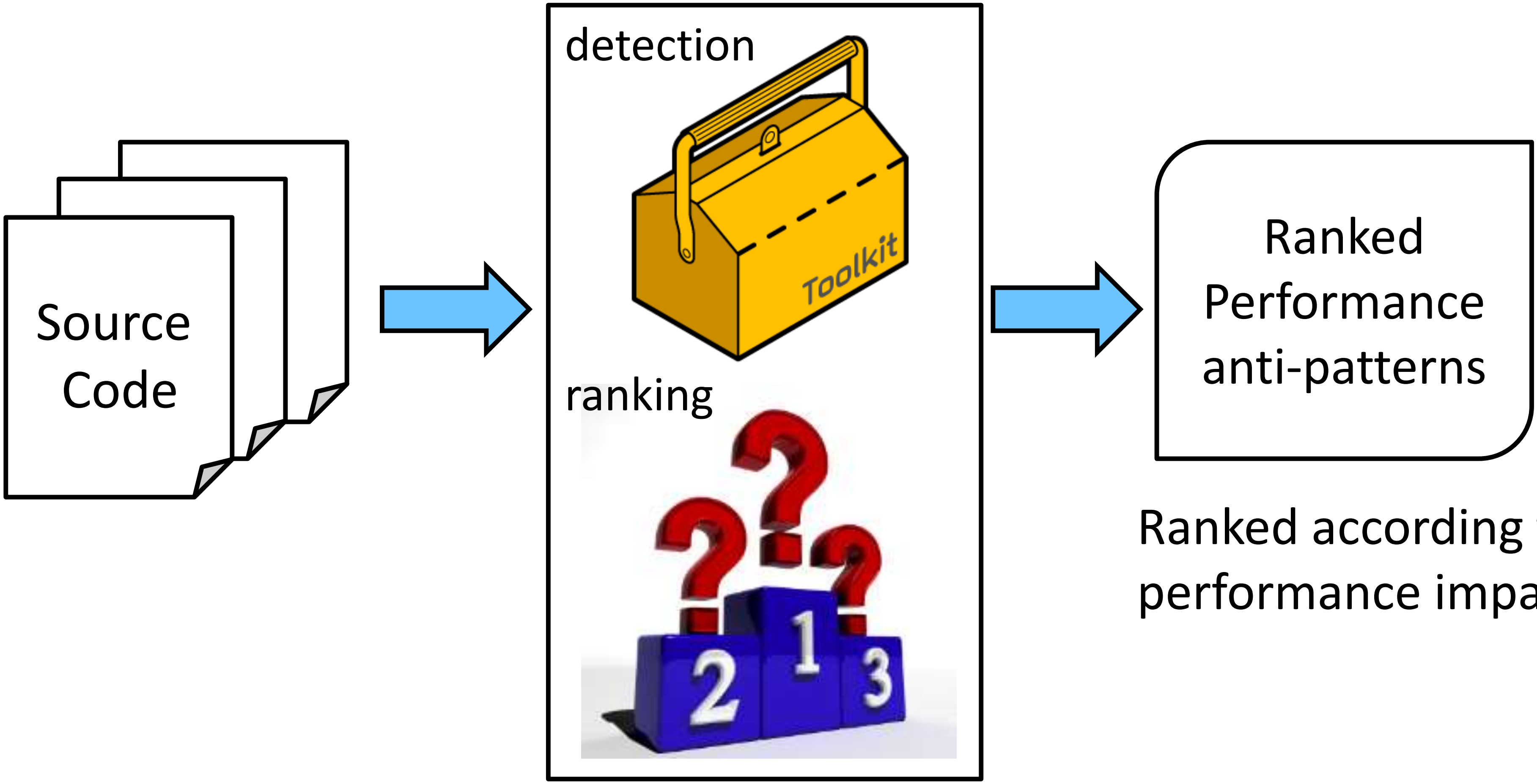
```
public class TcpClientSample
{
    public static void Main()
    {
        byte[] data = new byte[102
        TcpClient server;
        try{
            server = new TcpClient("  .
        }catch (SocketException){
            Console.WriteLine("Unable to connect to server");
            return;
        }
        NetworkStream ns = server.GetStream();
        int recv = ns.Read(data, 0, data.Length);
        stringData   = Encoding.
            ASCII.GetString(data, 0, recv);
        Console.WriteLine(stringData);
        while(true){
            input = Console.ReadLine();
            if (input == "exit") break;
                newchild.Properties["ou"].Add
                ("Auditing Department");
                newchild.CommitChanges();
                newchild.Close();
```

ORM code with performance anti-patterns

Bad system performance

The performance difference can be LARGE!

Performance anti-pattern detection framework



Source Code

detection

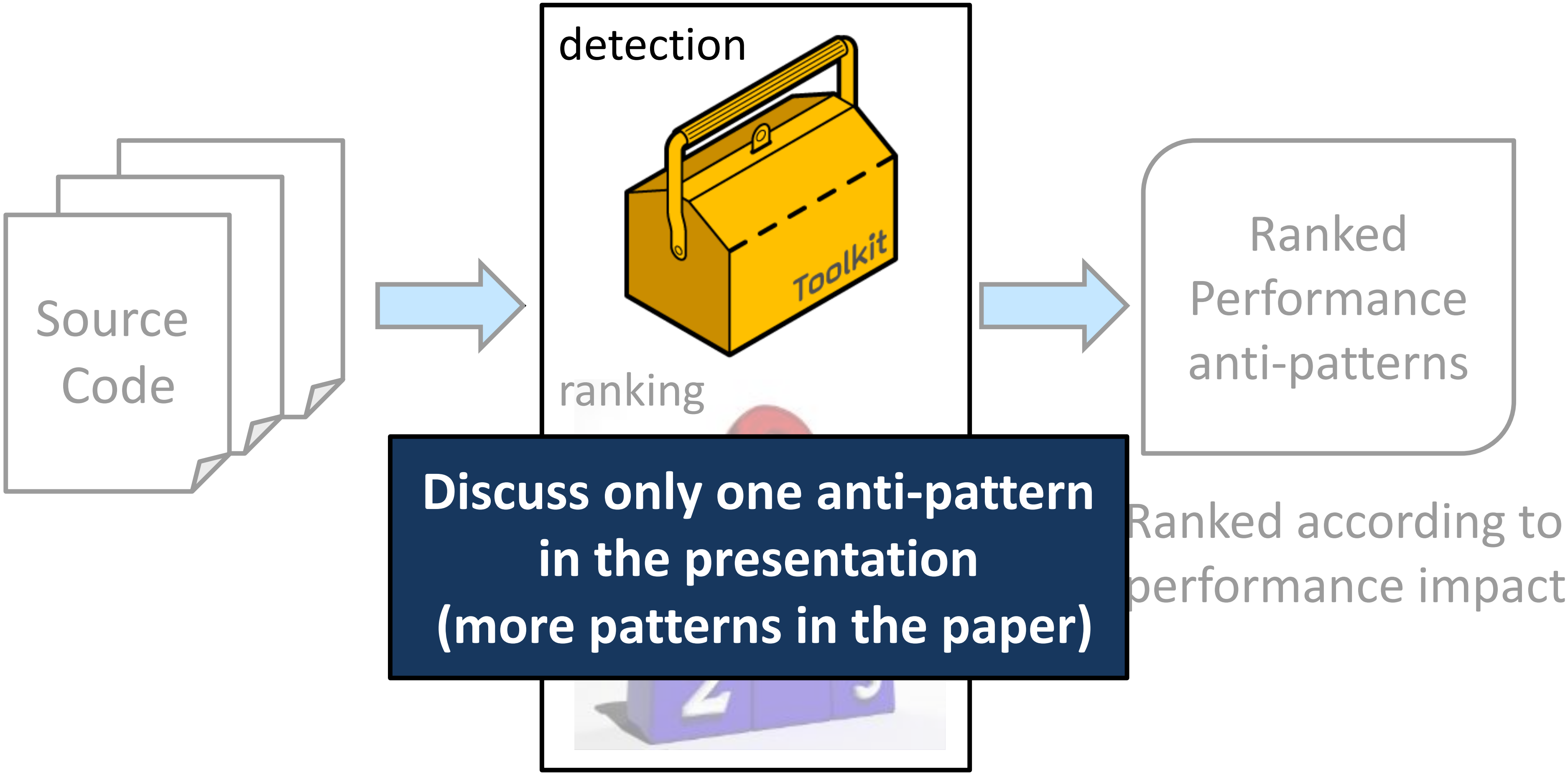
ranking

Ranked Performance anti-patterns

Ranked according to performance impact

Performance anti-pattern detection and ranking framework

Performance anti-pattern detection framework



Source Code

detection

ranking

Ranked Performance anti-patterns

Ranked according to performance impact

Discuss only one anti-pattern in the presentation (more patterns in the paper)

Performance anti-pattern detection and ranking framework

ORM excessive data anti-pattern

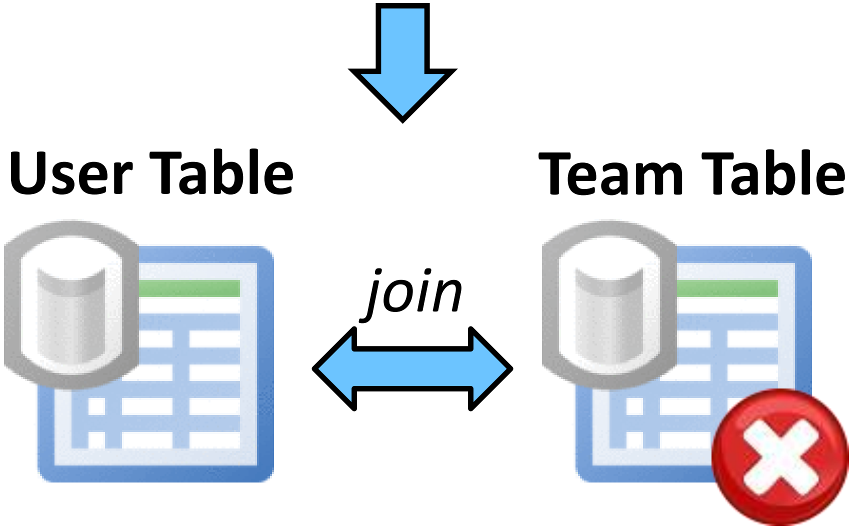
```
Class User{
        @EAGER
        List<Team> teams;
}
```

Eagerly retrieve teams from DB

Objects

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

SQL



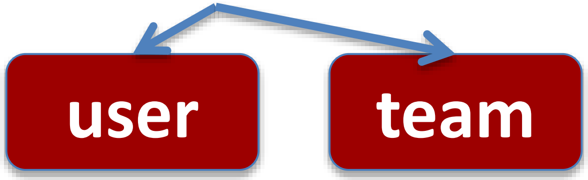
Team data is never used!

Detecting excessive data using static analysis

```
Class User{
        @EAGER
        List<Team> teams;
}
```

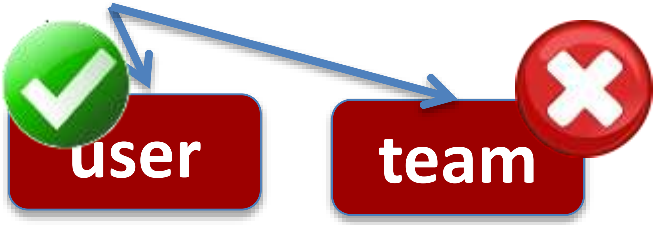
First find all the objects that eagerly retrieve data from DB

```
User user = findUserByID(1);
```



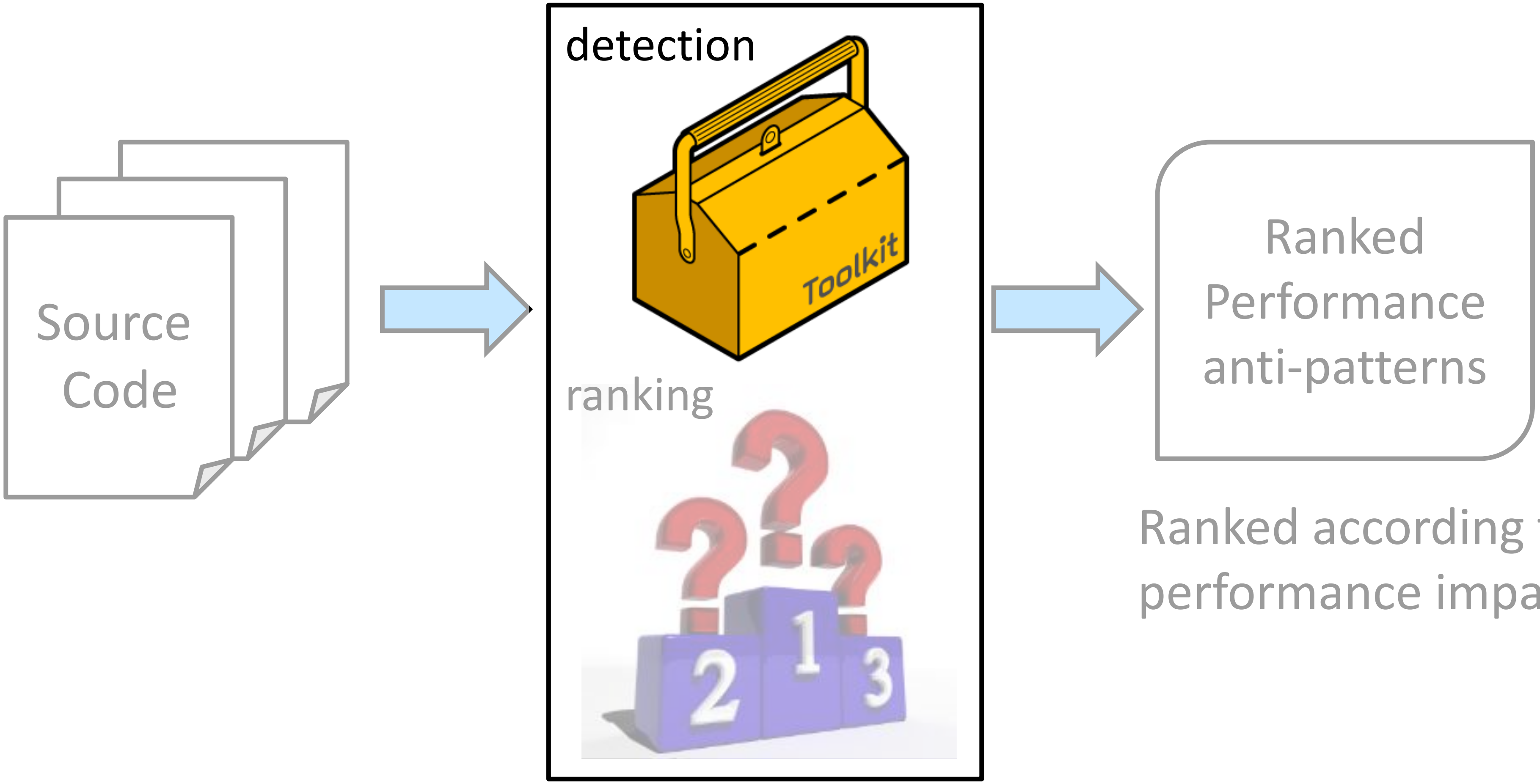
Identify all the data usages of objects

```
user.getName();
```



Check if the retrieved data is ever used

Performance anti-pattern detection framework



Source Code

detection

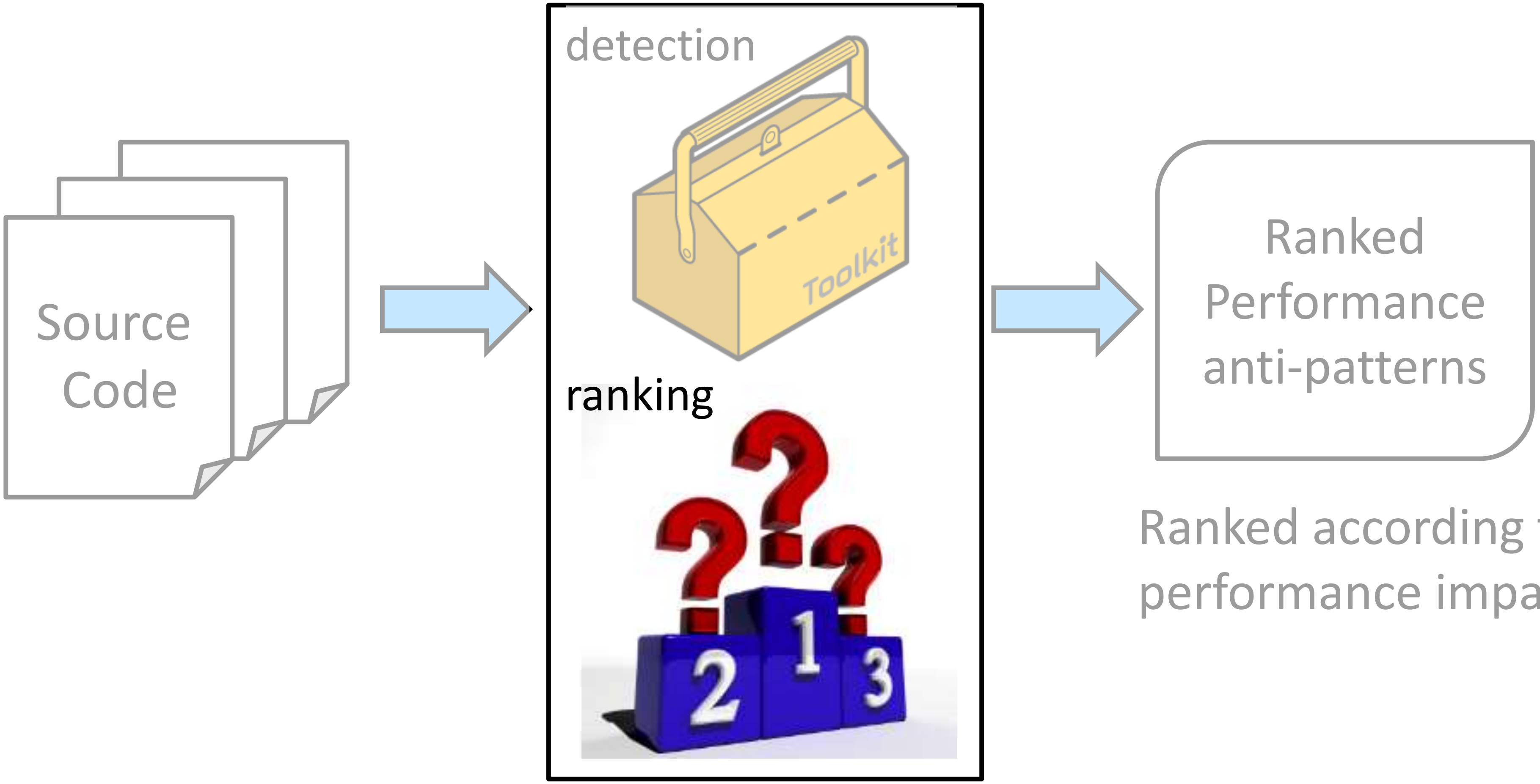
ranking

Ranked Performance anti-patterns

Ranked according to performance impact

Performance anti-pattern detection and ranking framework

Performance anti-pattern detection framework



Ranked according to performance impact

Performance anti-pattern detection and ranking framework

Performance anti-patterns have different impacts

```
User user_in_1_team = findUserByID(1);
```

Retrieving ***1 user*** and ***1 team***

```
User user_in_100_teams = findUserByID(100);
```

Retrieving ***1 user*** and ***100 teams!***

One can only reveal performance impact by execution

Measuring the impact using repeated measurements and effect sizes

Performance measurements are unstable:

We repeat each test 30 times to obtain stable measurement results

Size of performance impact is not defined:

We use ***effect sizes (Cohen's D)*** to measure the performance impact

$$\text{Effect sizes} = \begin{cases} \text{trivial} & \text{if } \mathit{Cohen's}\ d \leq 0.2 \\ \text{small} & \text{if } 0.2 < \ \mathit{Cohen's}\ d \leq 0.5 \\ \text{medium} & \text{if } 0.5 < \ \mathit{Cohen's}\ d \leq 0.8 \\ \text{large} & \text{if } 0.8 < \ \mathit{Cohen's}\ d \end{cases}$$

Studied systems and detection results

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

482 excessive data

Enterprise system
> 3,000 files
> 300K LOC

> 10 excessive data

Spring open-source system
Online system for a pet clinic
51 files
3.3K LOC

10 excessive data

Research questions

Performance impact

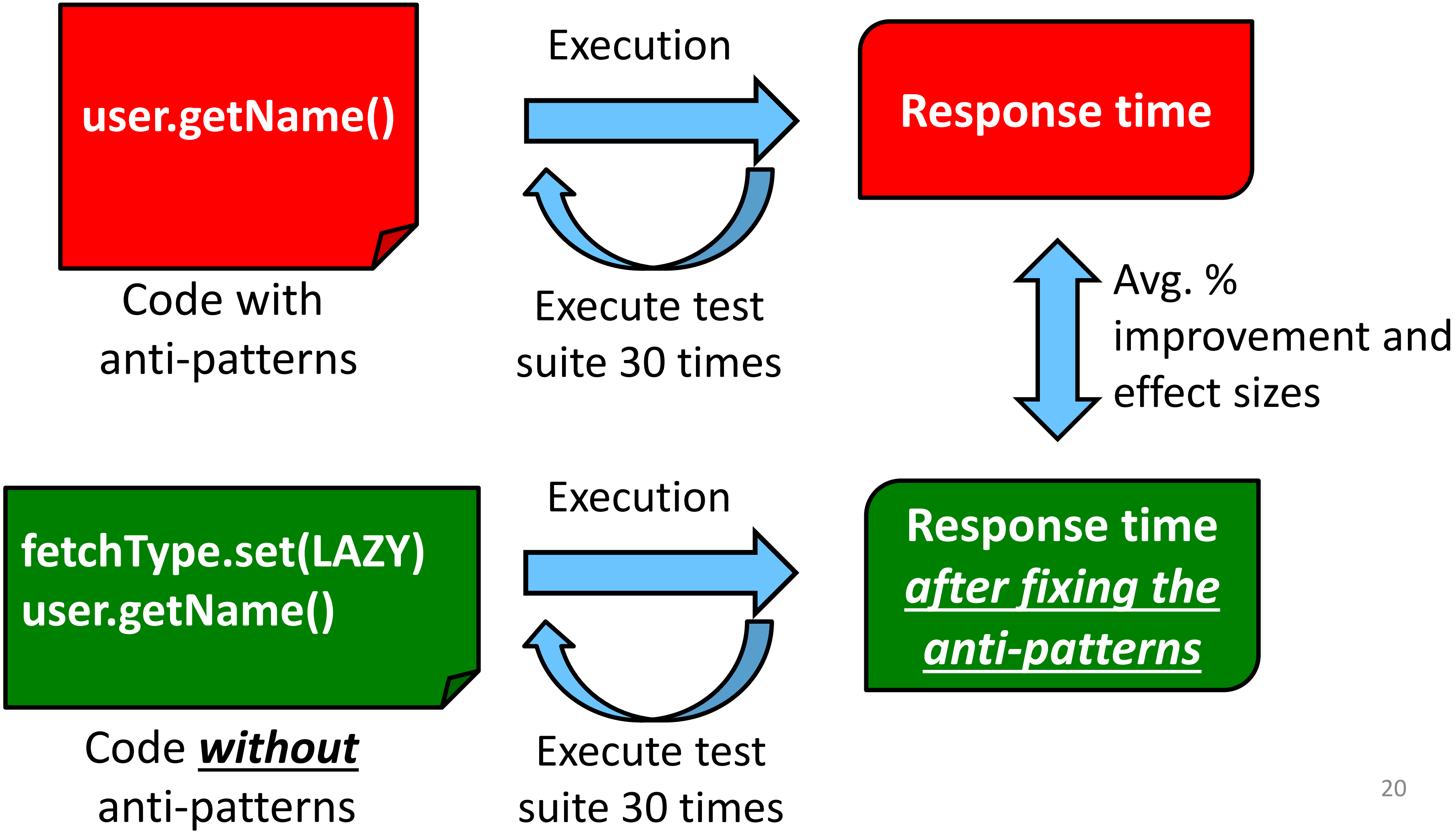
Ranks of the anti-patterns at different scales

Research questions

Performance impact

Ranks of the anti-patterns at different scales

Assessing anti-pattern impact by fixing the anti-patterns



user.getName()

Code with anti-patterns

Execution

Execute test suite 30 times

Response time

Avg. % improvement and effect sizes

fetchType.set(LAZY)
user.getName()

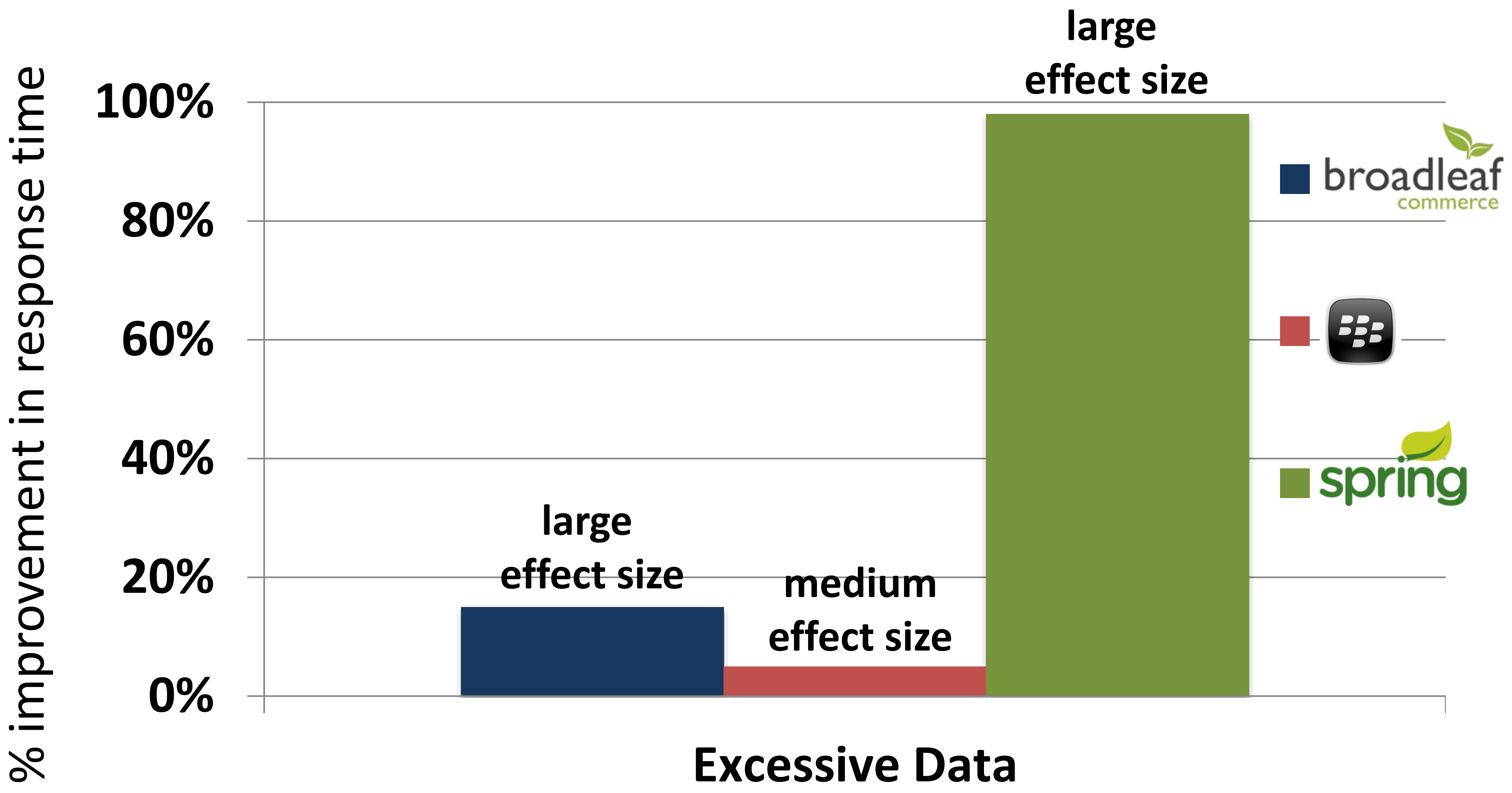
Code ***without*** anti-patterns

Execution

Execute test suite 30 times

Response time *after fixing the anti-patterns*

Performance anti-patterns have medium to large effect sizes



Research questions

Performance impact

Ranks of the anti-patterns at different scales

Removing anti-pattern improves response by ~35%

Research questions

Performance impact

Ranks of the anti-patterns at different scales

Removing anti-pattern improves response by ~35%

Performance problems usually arise under large load

Performance problems revealed at small scales may be more serious

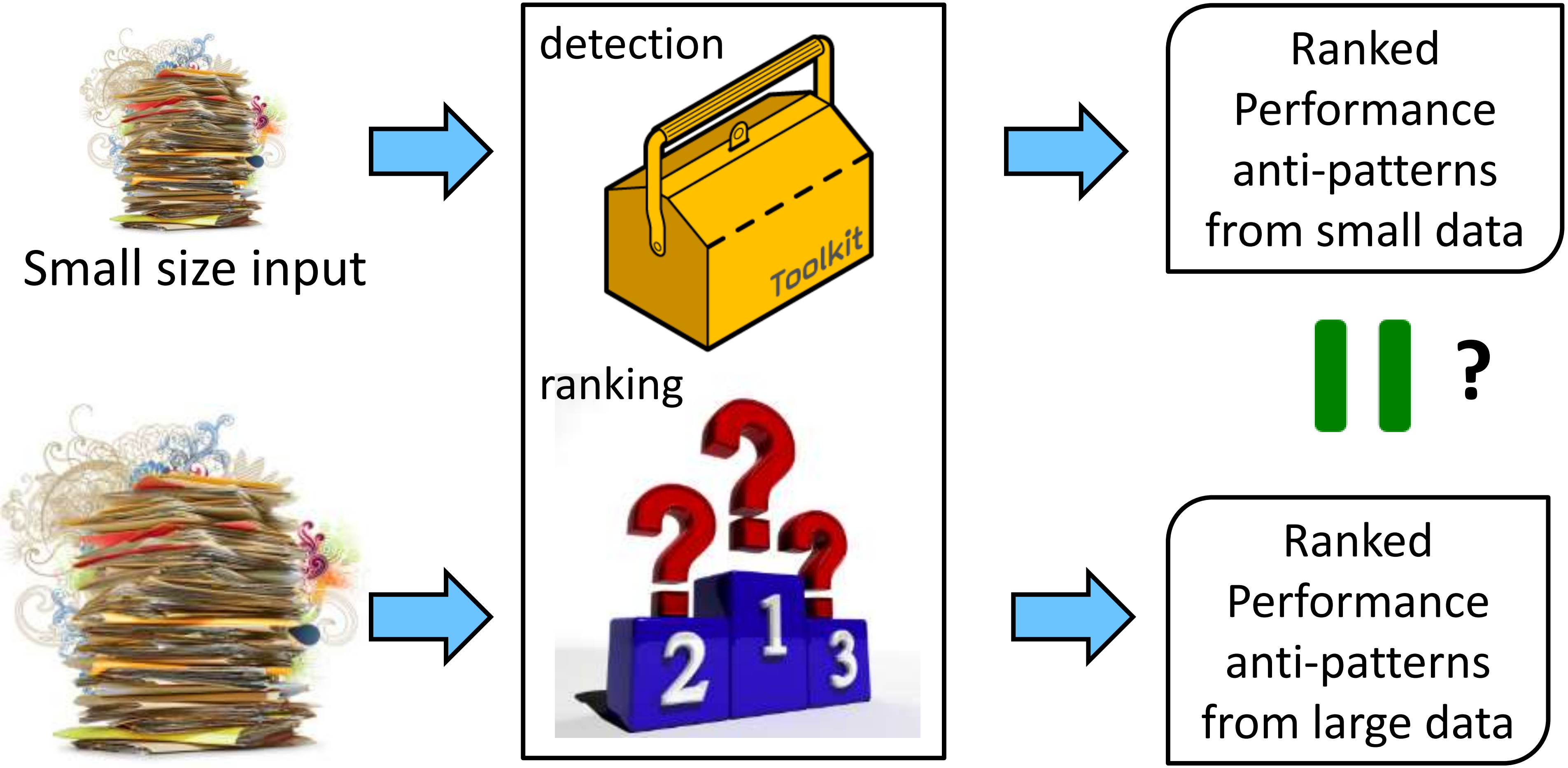
Input scales may have ***exponential*** effects on performance

Different input scales

Performance at different input scales

We should first fix the anti-patterns that have larger effects at smaller scales

Comparing ranked anti-patterns at different data scales



Small size input

detection

ranking

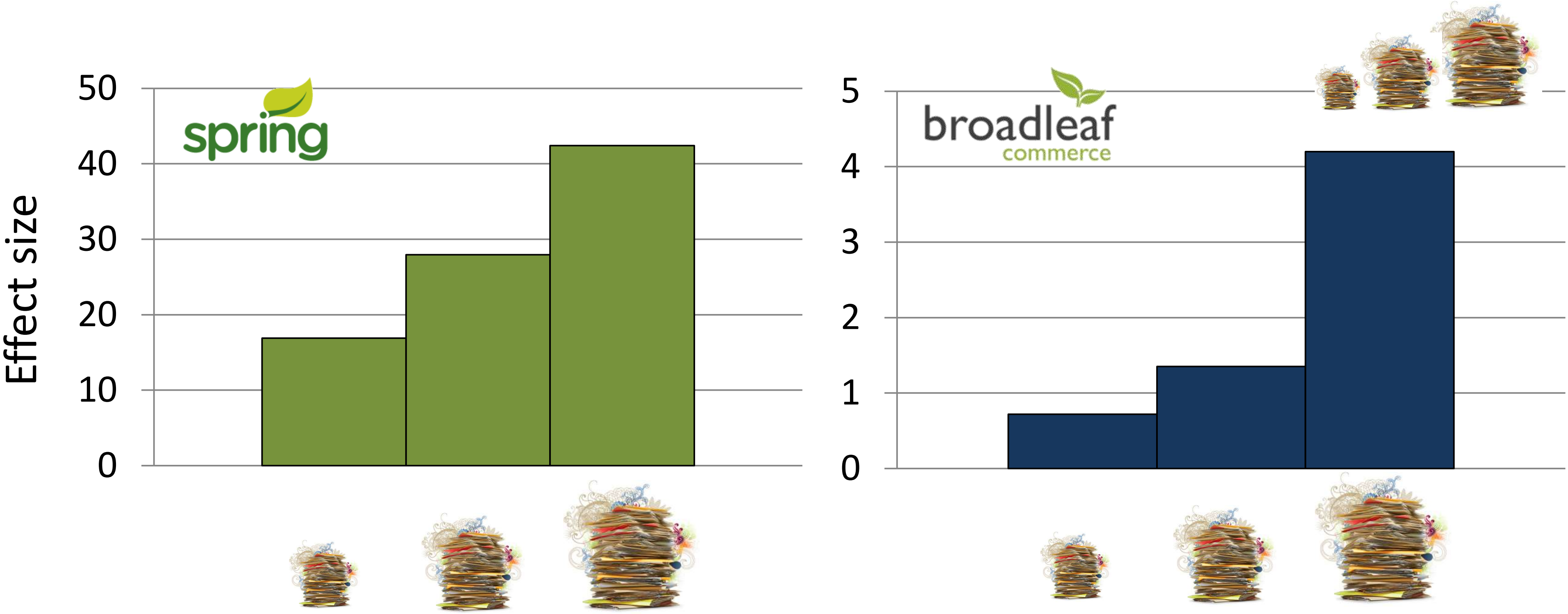
Ranked Performance anti-patterns from small data

= ?

Ranked Performance anti-patterns from large data

Large size input

Anti-patterns have large effects on performance even at smaller data scales



Effect sizes and the ranks of the anti-patterns are consistent in different data scales

Research questions

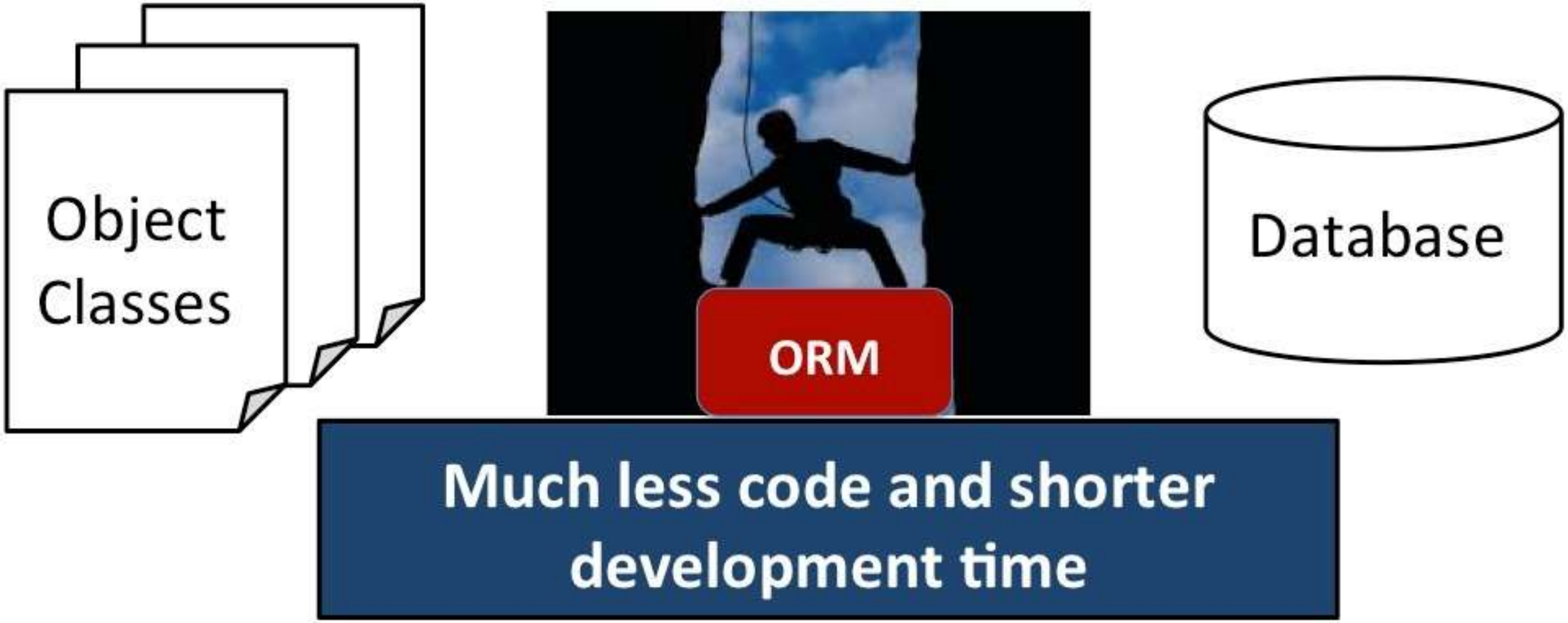
Performance impact

Ranks of the anti-patterns at different scales

Removing anti-pattern improves response by ~35%

Ranks of the anti-patterns are consistent in different data scales

Object-Relational Mapping eliminates the gap between objects and SQL



Object Classes

ORM

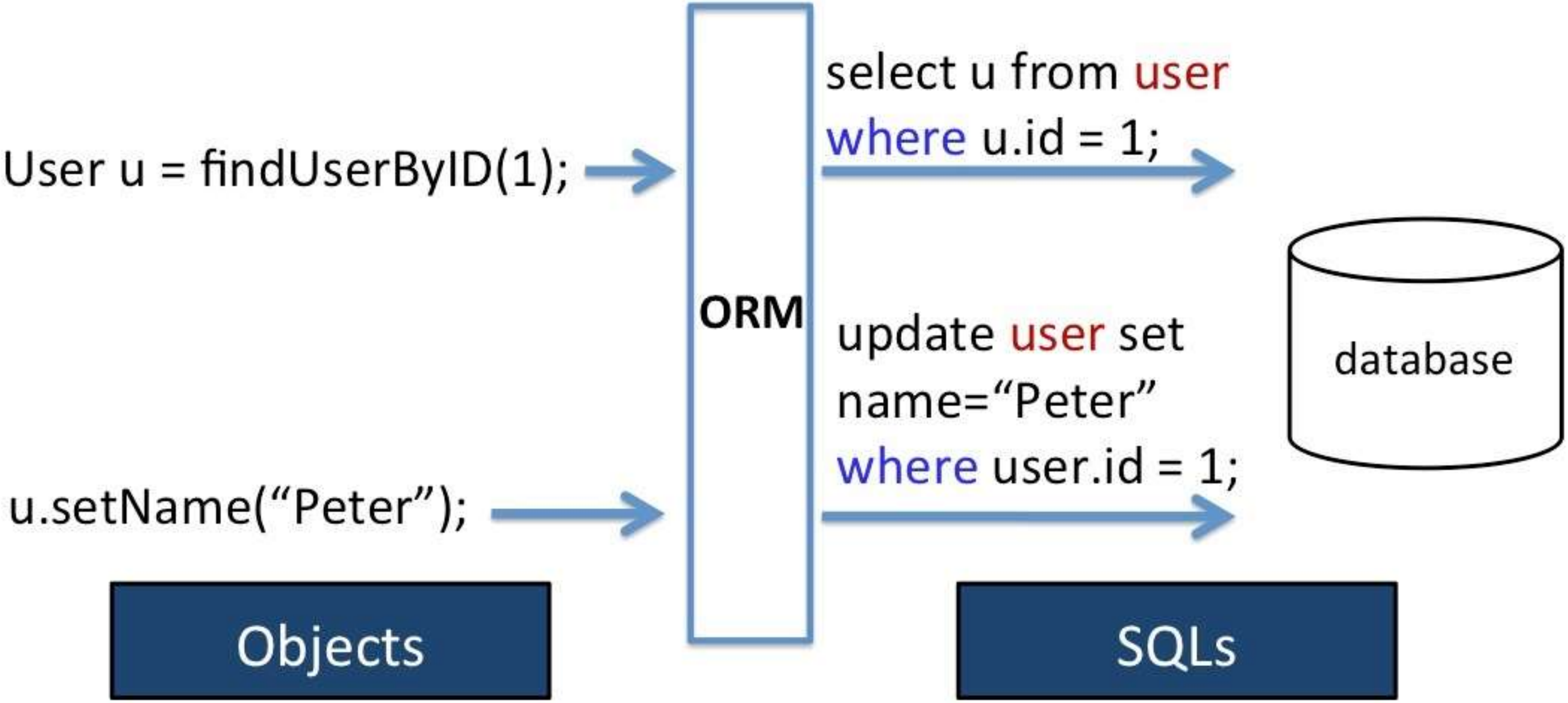
Database

Much less code and shorter development time

Problem of using raw SQLs

- Lots of **boilerplate code**
- Need to **manage object-DB translations** manually

Accessing the database using ORM



```
User u = findUserByID(1);
```

```
select u from user
where u.id = 1;
```

ORM

```
u.setName("Peter");
```

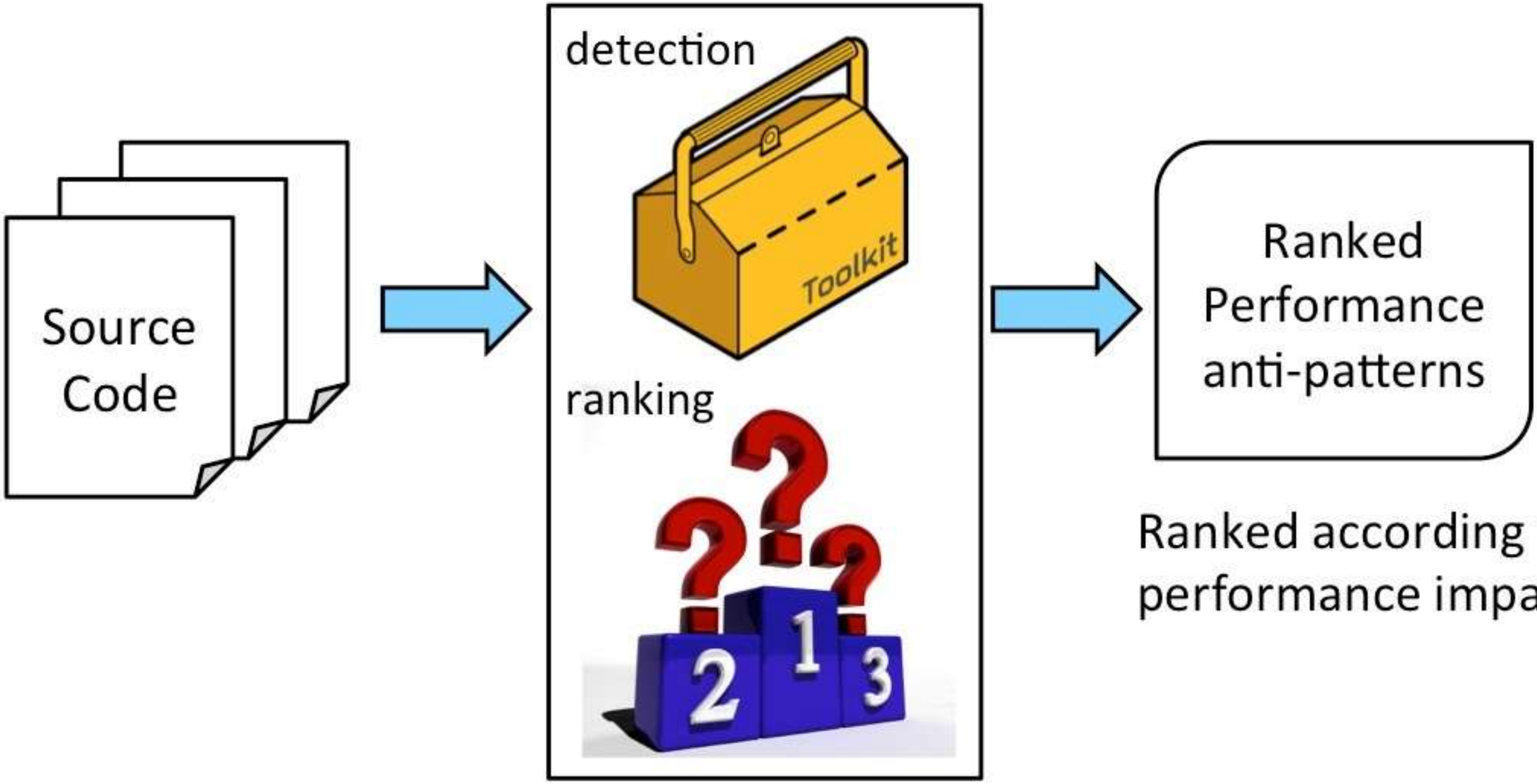
```
update user set
name="Peter"
where user.id = 1;
```

database

Objects

SQLs

Performance anti-pattern detection framework



Source Code

detection

ranking

Ranked Performance anti-patterns

Ranked according to performance impact

Performance anti-pattern detection and ranking framework

Research questions

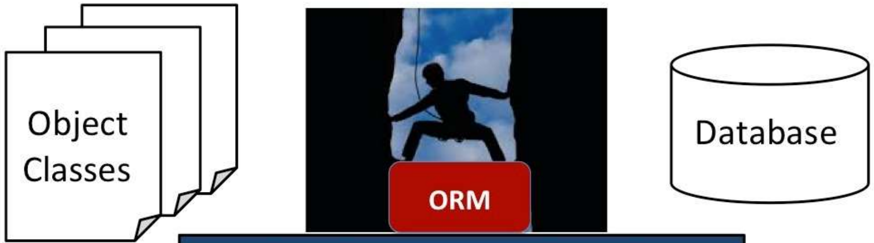
Performance impact

Ranks of the anti-patterns at different scales

Removing anti-pattern improves response by ~35%

Ranks of the anti-patterns are consistent in different data scales

Object-Relational Mapping eliminates the gap between objects and SQL



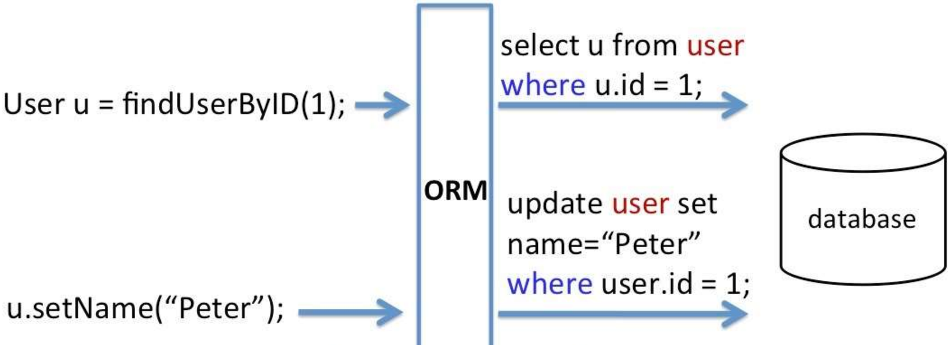
Much less code and shorter development time

Problem of using raw SQLs

- Lots of **boilerplate code**
- Need to manage object-DB

tr

Accessing the database using ORM



```
User u = findUserByID(1);
select u from user
where u.id = 1;

u.setName("Peter");
update user set
name="Peter"
where user.id = 1;
```

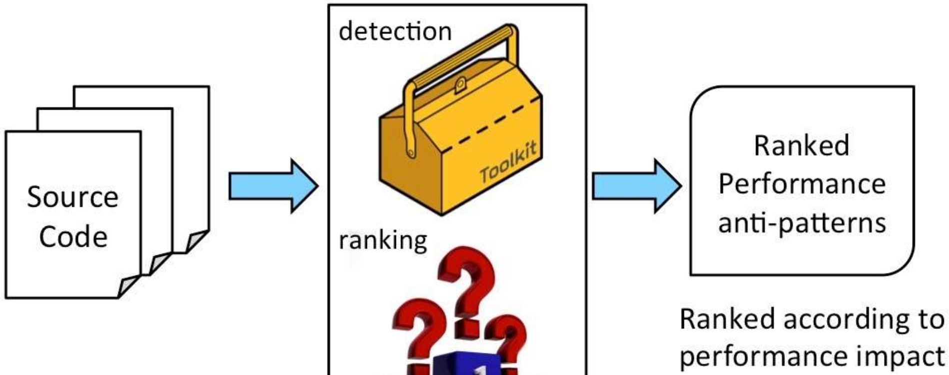
Objects

SQLs

tsehsun@cs.queensu.ca

Perform

detection framework


Source Code
detection
Toolkit
ranking
2
1
3
Ranked Performance anti-patterns
Ranked according to performance impact

Performance anti-pattern detection and ranking framework

estions

Performance impact

Ranks of the anti-patterns at different scales

Removing anti-pattern improves response by ~35%

Ranks of the anti-patterns are consistent in different data scales