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

Université Concordia University

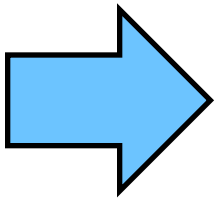
York University

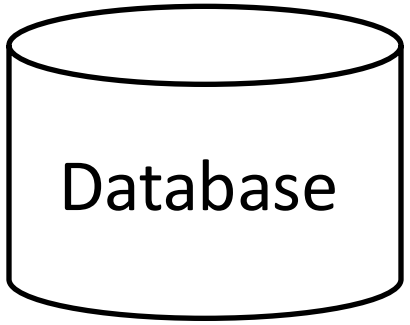
Queen's University

Mohamed Nasser, Parminder Flora

BlackBerry

Databases are essential in large-scale software systems

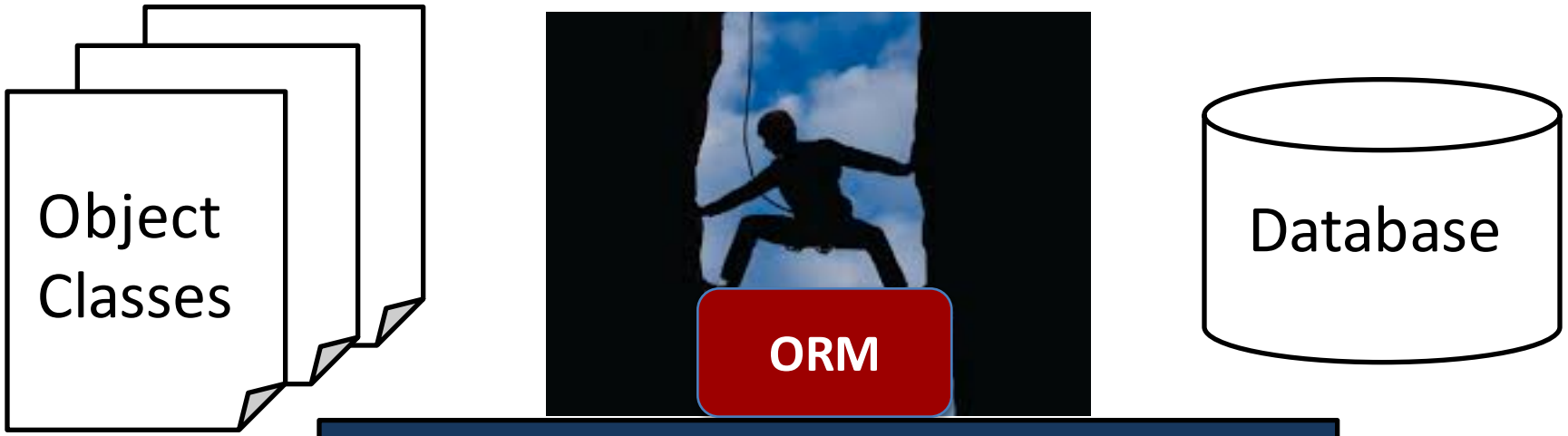




Application developers work with objects

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

- 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

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

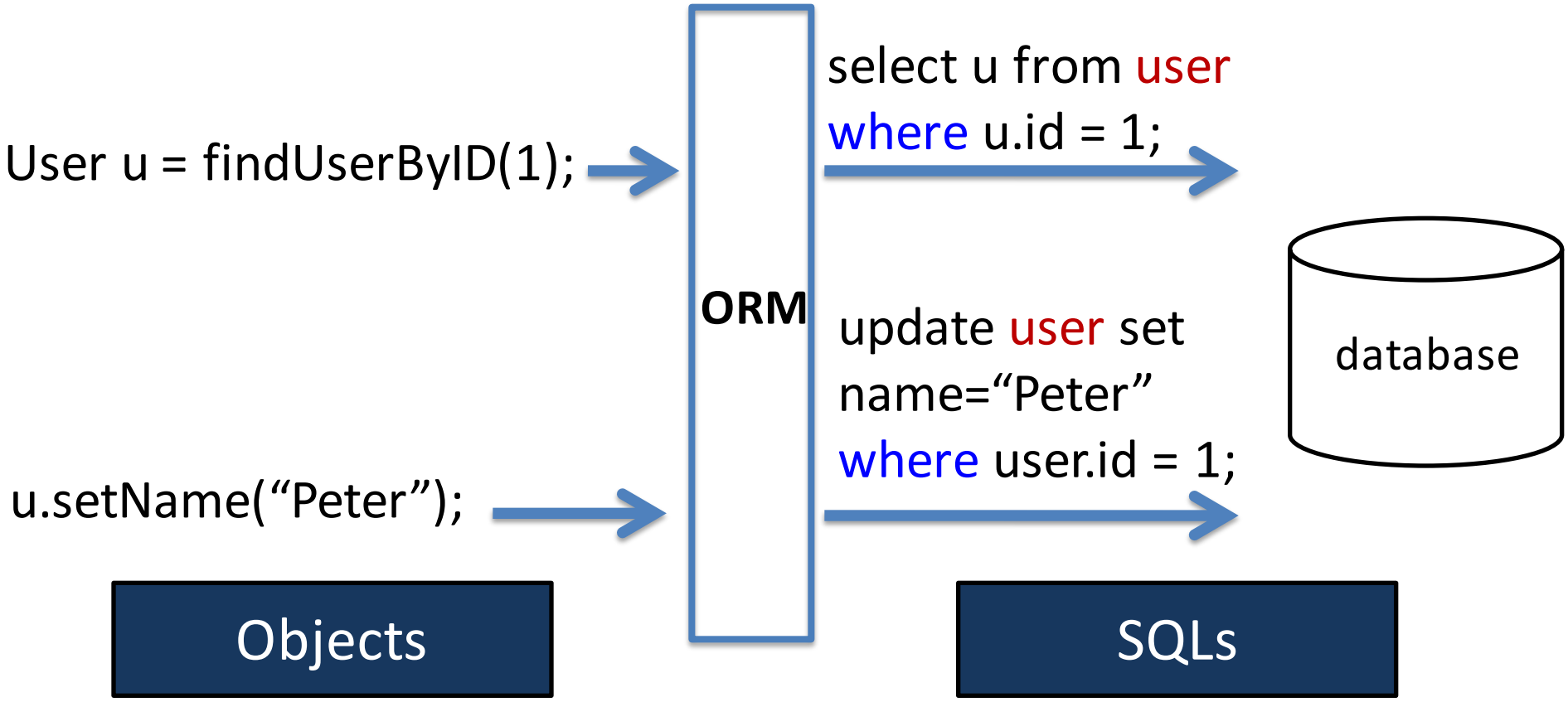
A user can belong to multiple teams

Eagerly retrieve associated teams when retrieving a user object

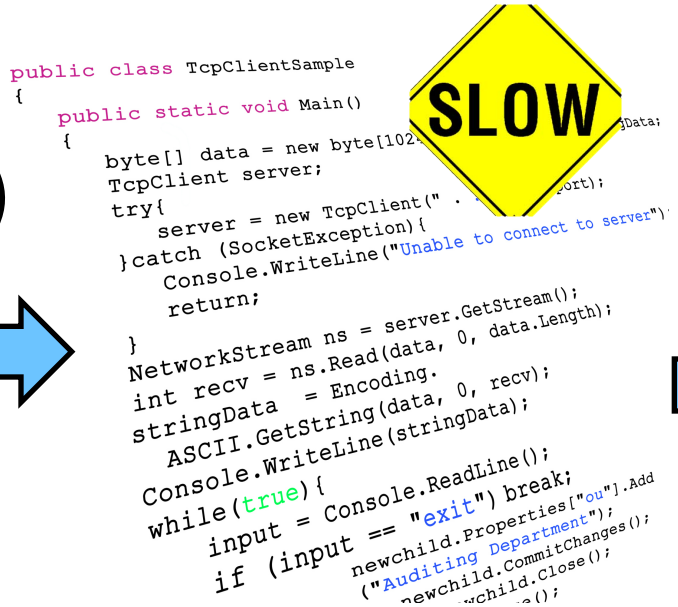
User.java

```
@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



Developers may not be aware of database access

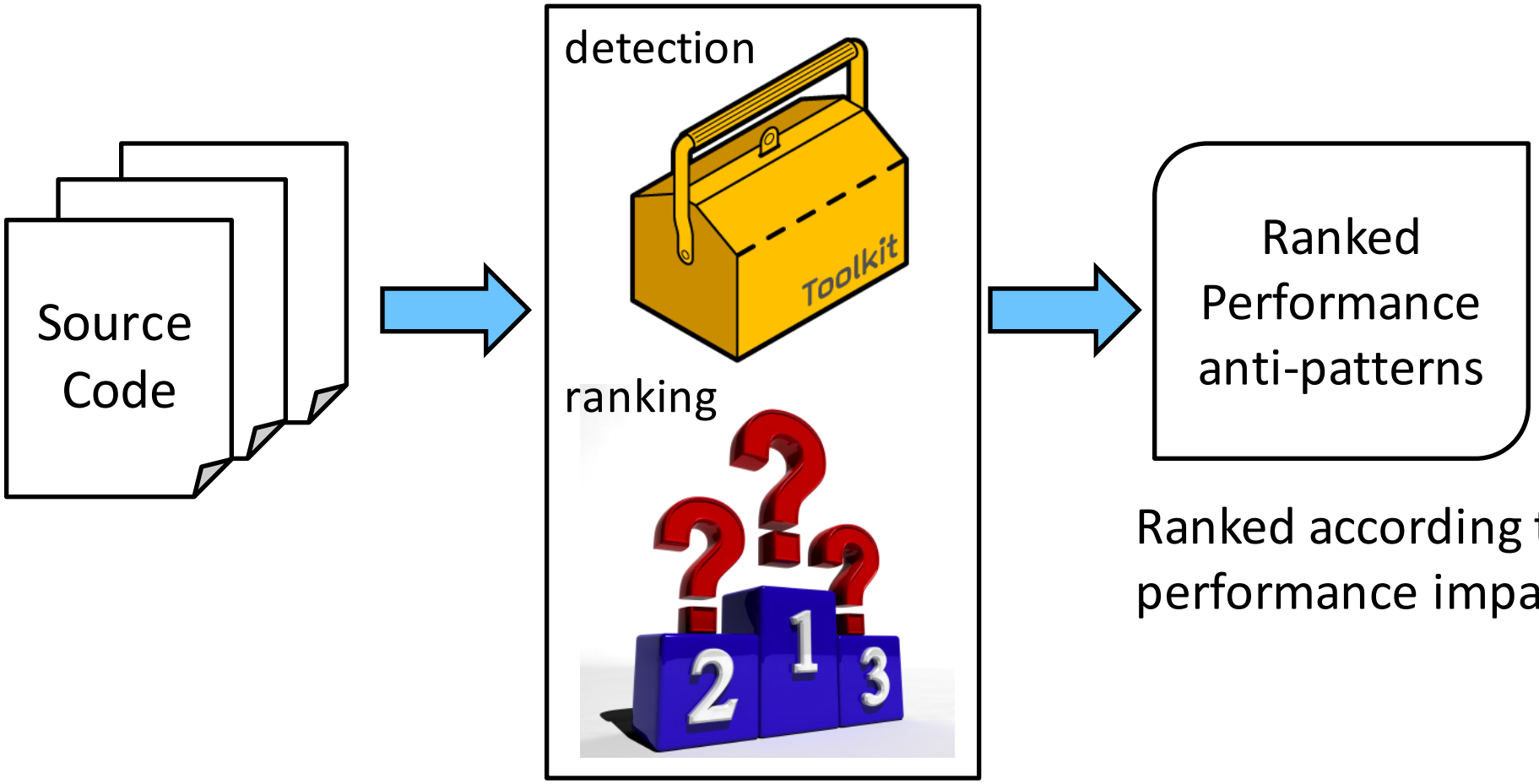


ORM code with performance anti-patterns

Bad system performance

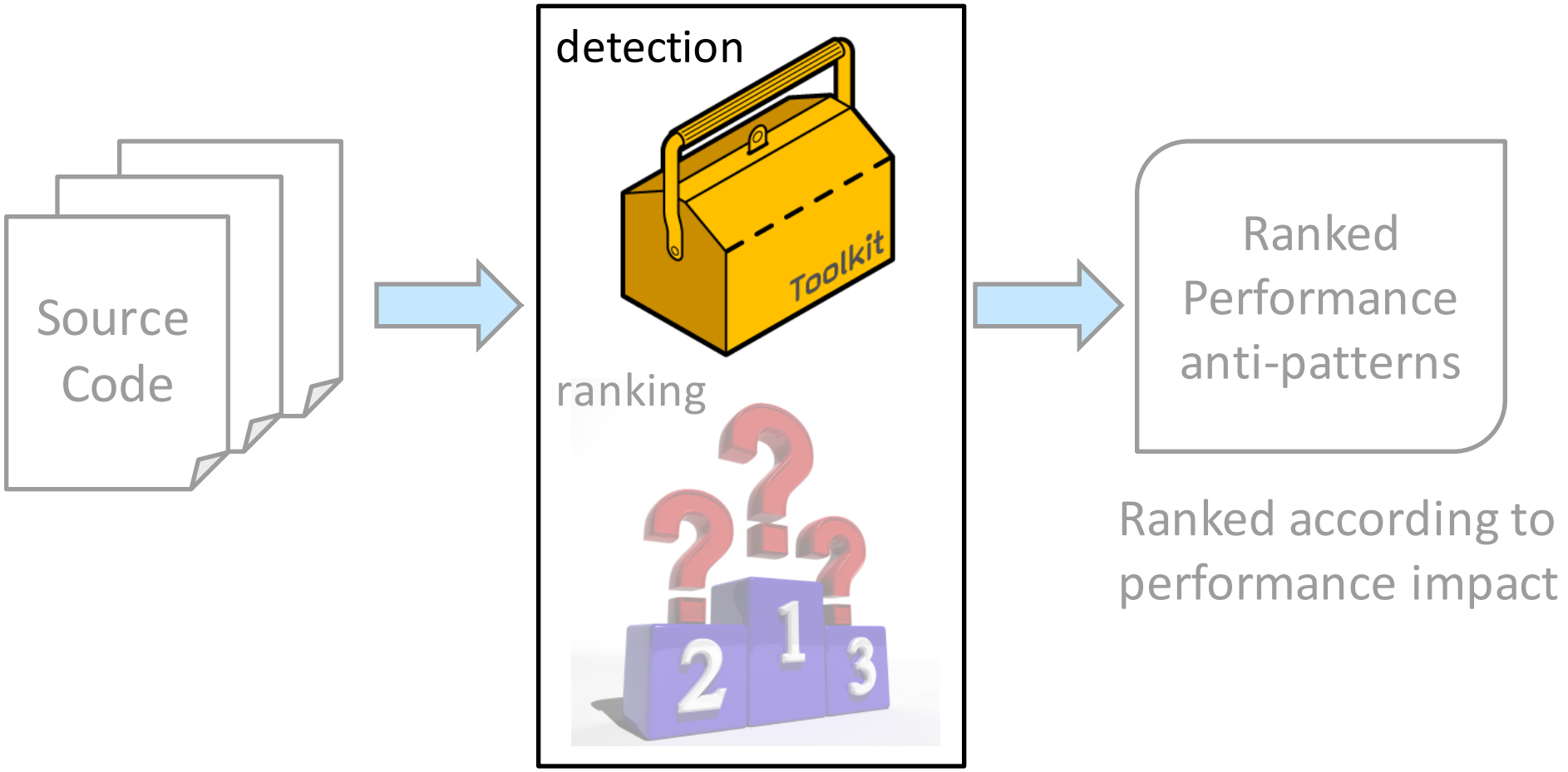
The performance difference can be LARGE!

Performance anti-pattern detection framework



Performance anti-pattern detection and ranking framework

Performance anti-pattern detection framework



Performance anti-pattern detection and ranking framework

ORM one-by-one processing anti-pattern


```
@Entity
@Table (name="company")
Class Company{
        List<Department> department;
}
```

Mapping a class to a DB table

Objects

```
for (Company c: companyList){
    for (Department d:c.getDepartments()){
        d.getDepartmentName();
    }
}
```

SQL

```
select department as d where d.companyID=1
select department as d where d.companyID=2
....
select department as d where d.companyID in (1,2,…)
```

Detecting one-by-one processing using static analysis


```
@Entity
@Table (name="company")
Class Company{
        List<Department>
department;
}
```

First find all the classes that are mapped to DB

```
for (Company c: companyList){
    for (Department d:c.getDepartments()){
        d.getDepartmentName();
    }
}
```

Identify all the loops

Check if any DB access is inside a loop

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

User Table

join

Team Table

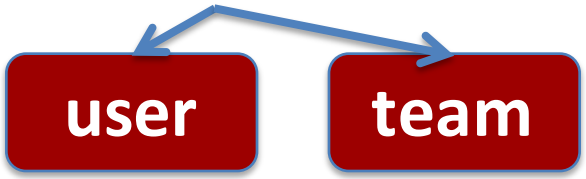
Team data is never used!

Detecting excessive data using static analysis


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

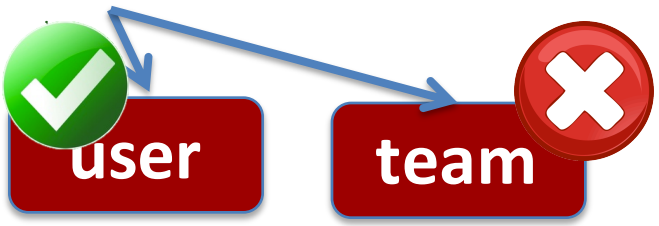
First find all the objects that eagerly retrieve data from DB

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



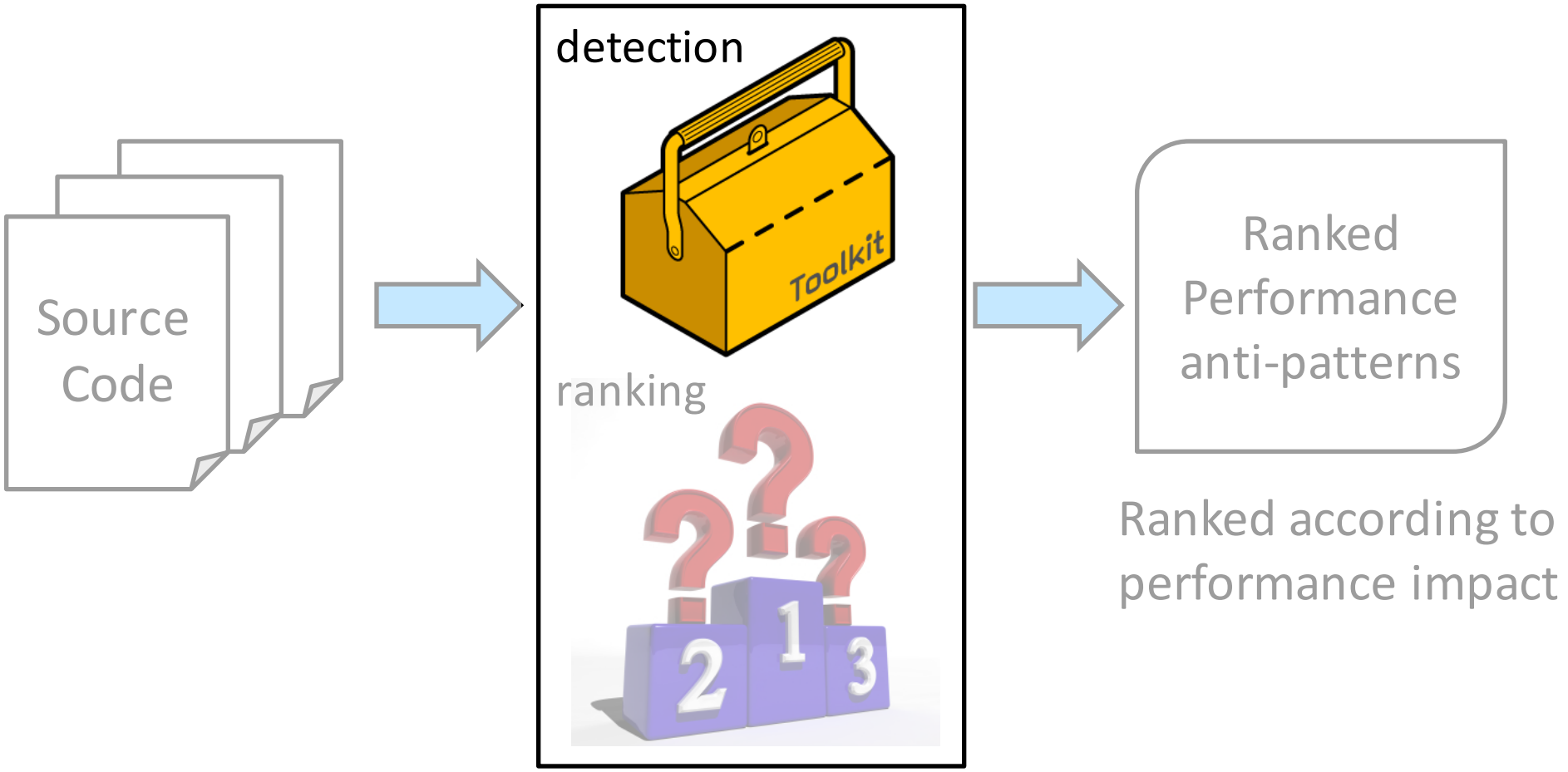
Identify all the data usages of objects

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



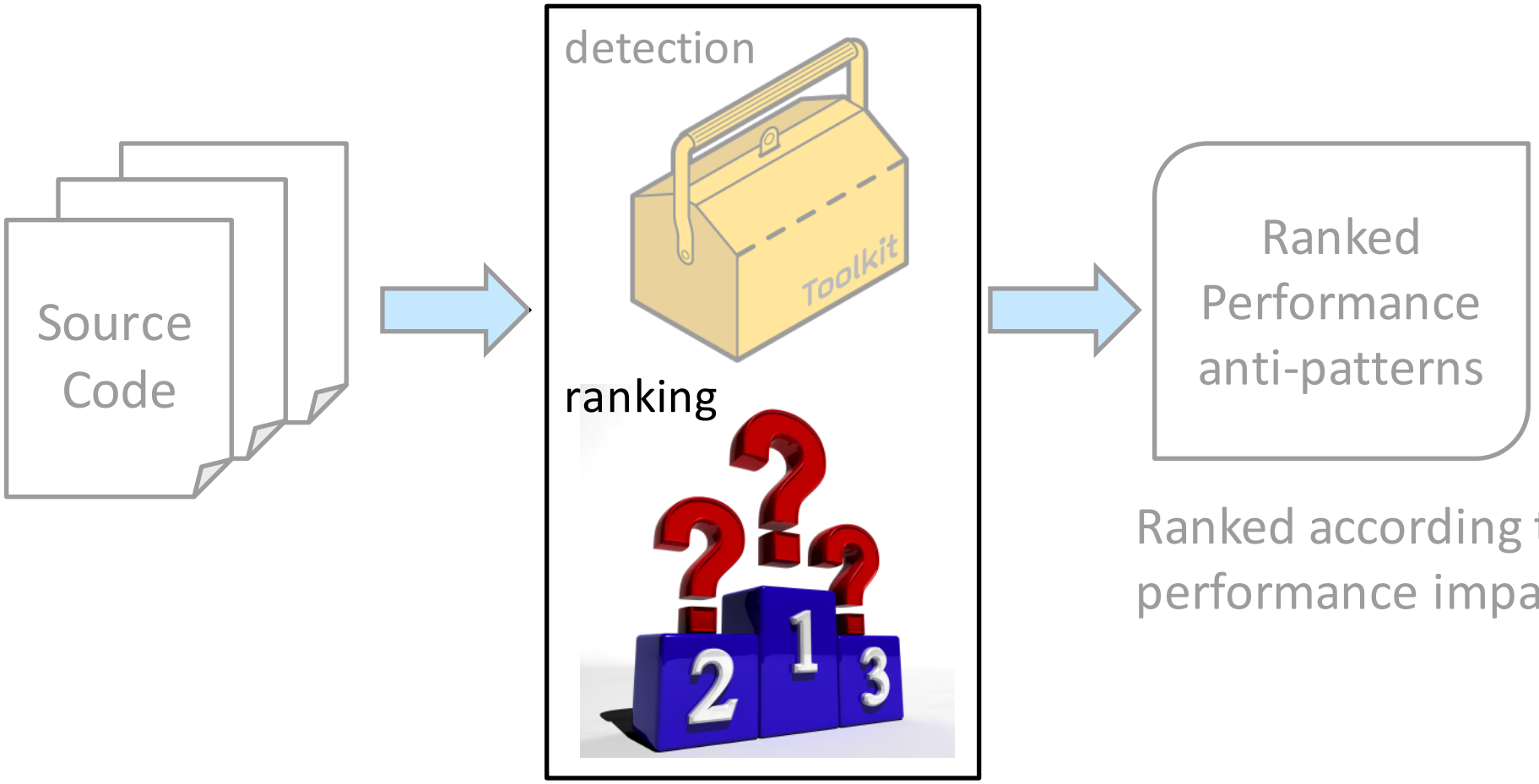
Check if the retrieved data is ever used

Performance anti-pattern detection framework



Performance anti-pattern detection and ranking framework

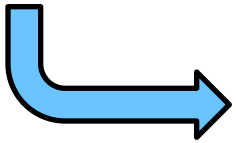
Performance anti-pattern detection framework



Performance anti-pattern detection and ranking framework

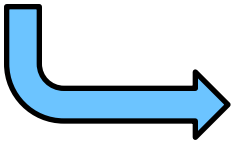
Performance anti-patterns have different impacts

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

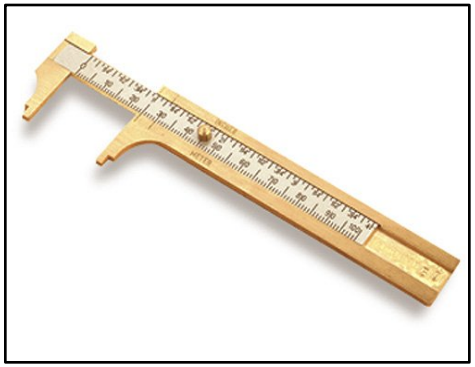
Measuring the impact using repeated measurements and effect sizes

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



Effect sizes =

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

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

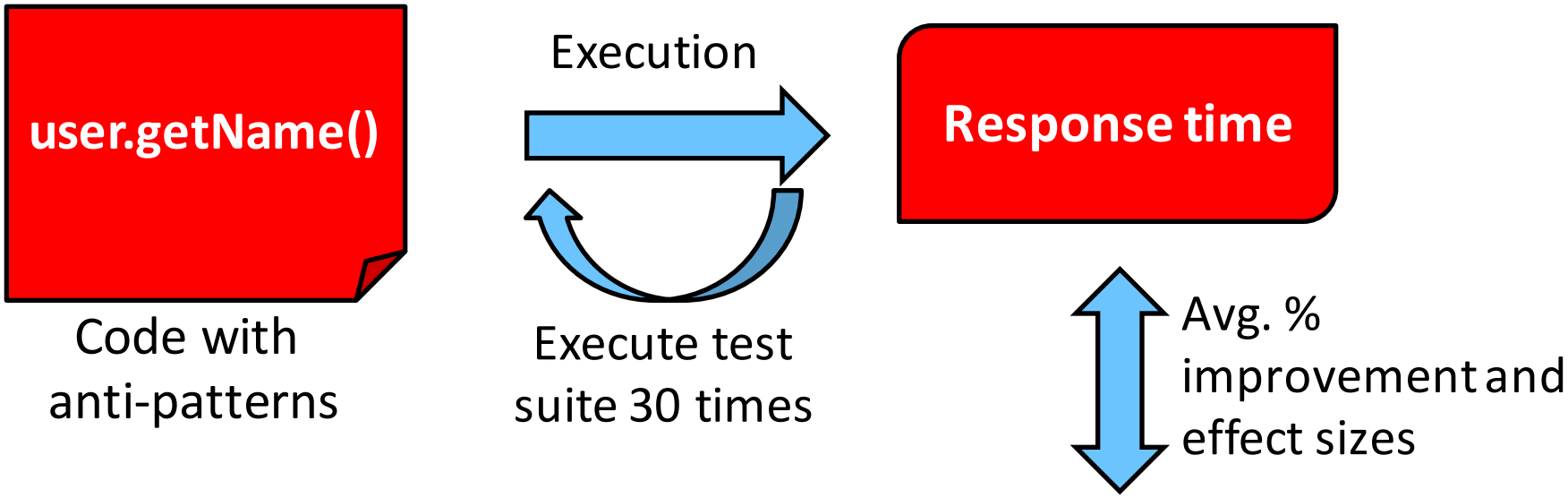
Ranks of the anti-patterns at different scales

Research questions

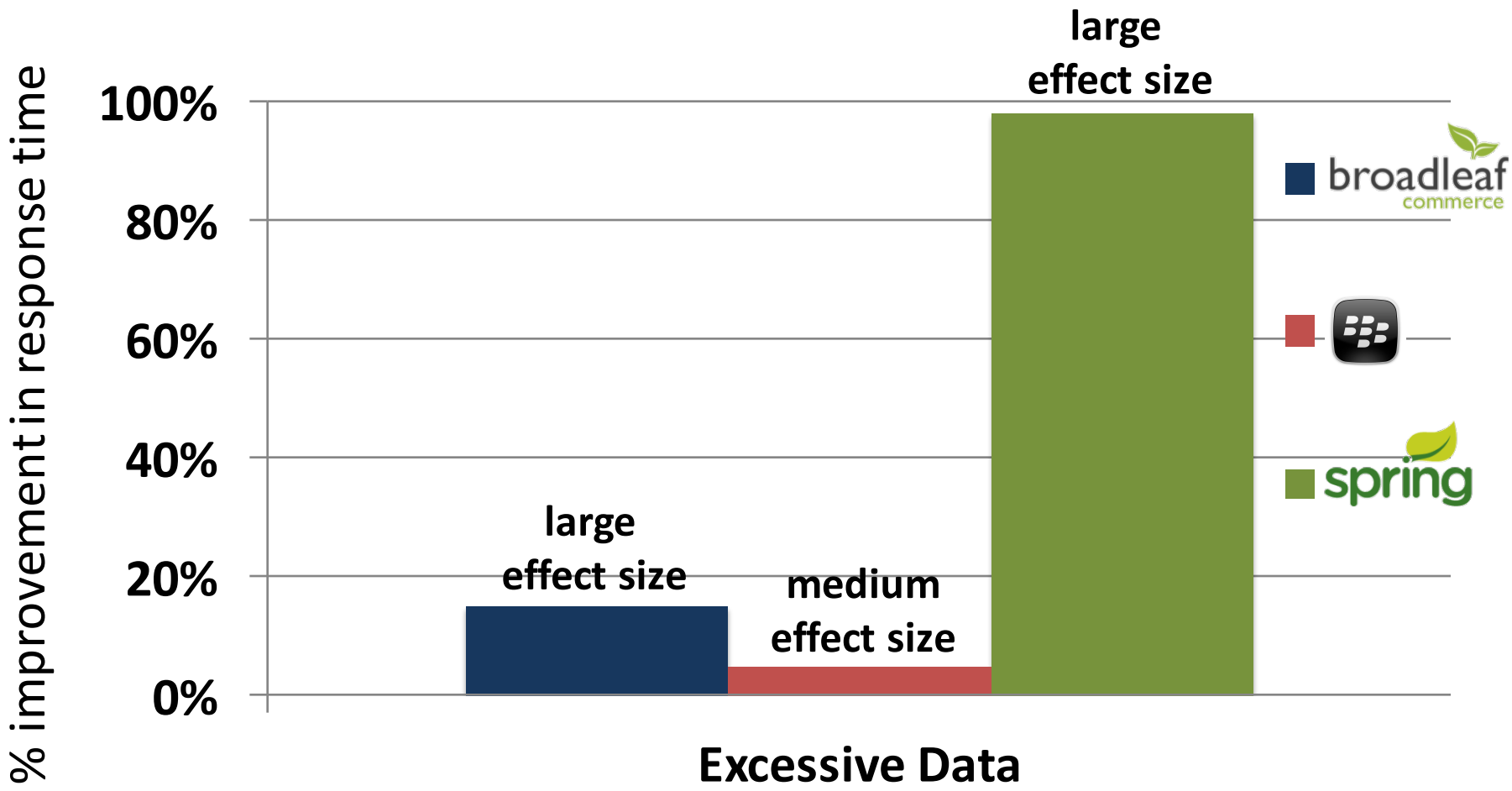
Performance impact

Ranks of the anti-patterns at different scales

Assessing anti-pattern impact by fixing the anti-patterns



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

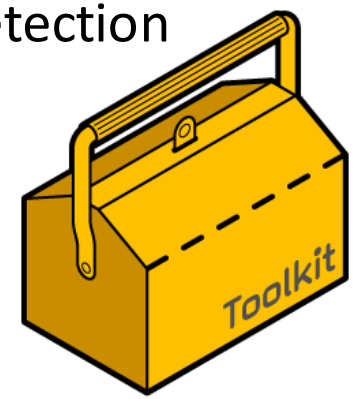
We should first fix the anti-patterns that have larger effects at smaller scales

Comparing ranked anti-patterns at different data scales

Small size input

Large size input

detection



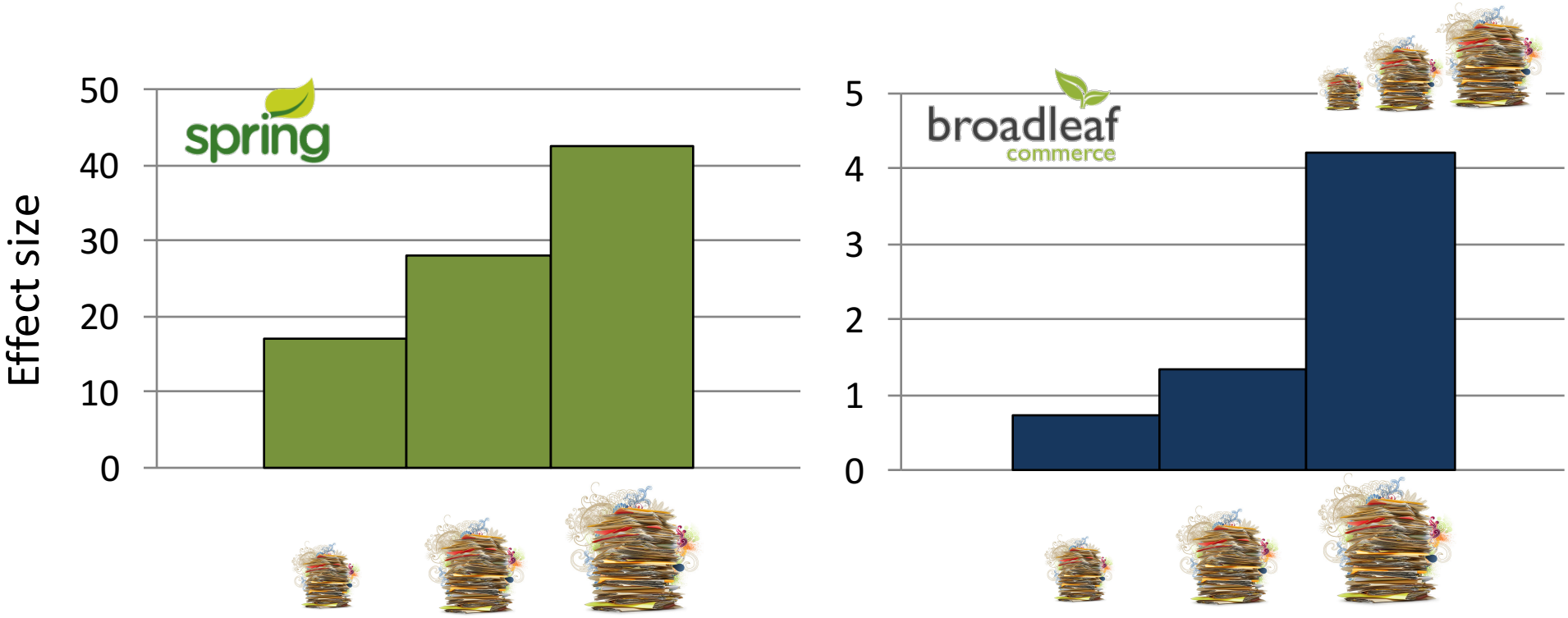
ranking

Ranked Performance anti-patterns from small data

=?

Ranked Performance anti-patterns from large data

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%

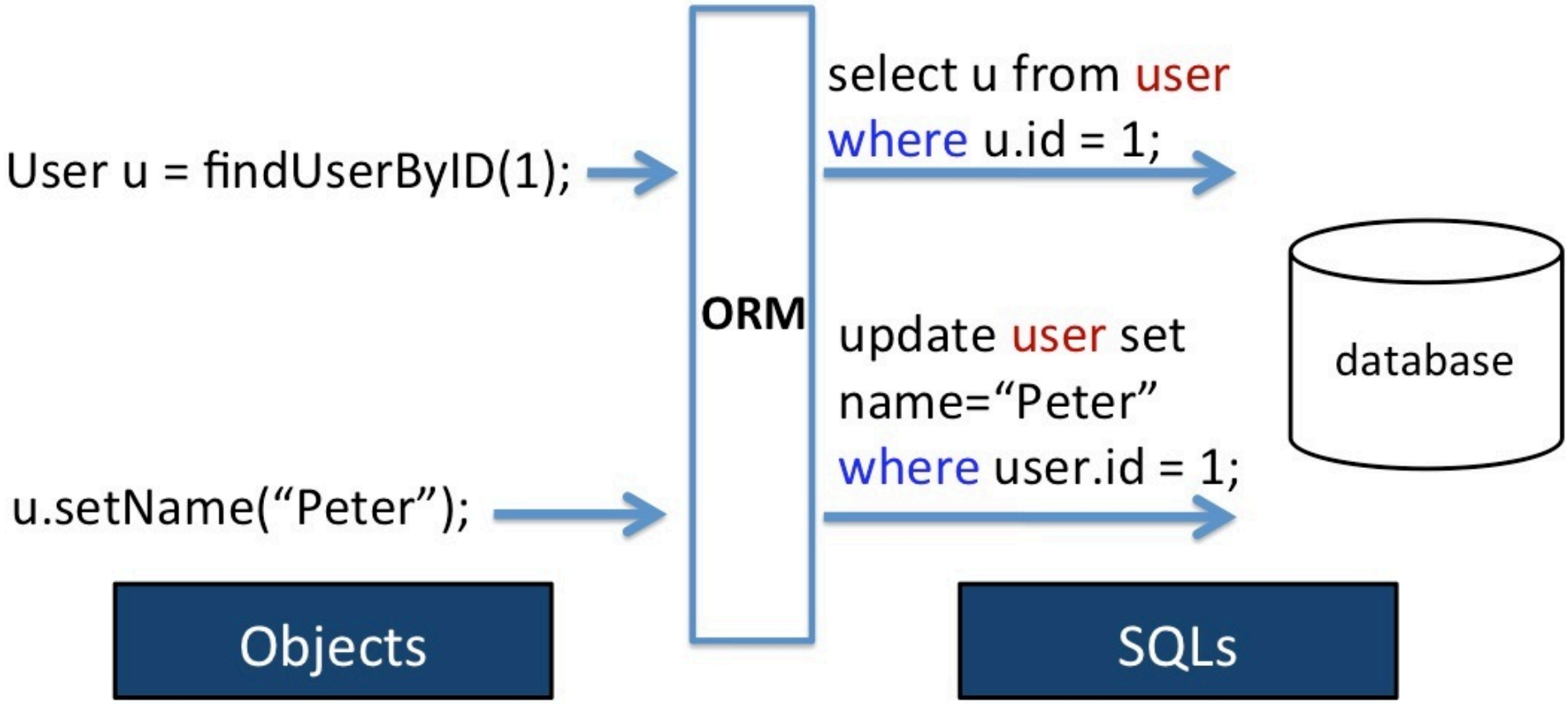
Ranks of the anti-patterns are consistent in different data scales

Object-Relational Mapping eliminates the gap between objects and SQL

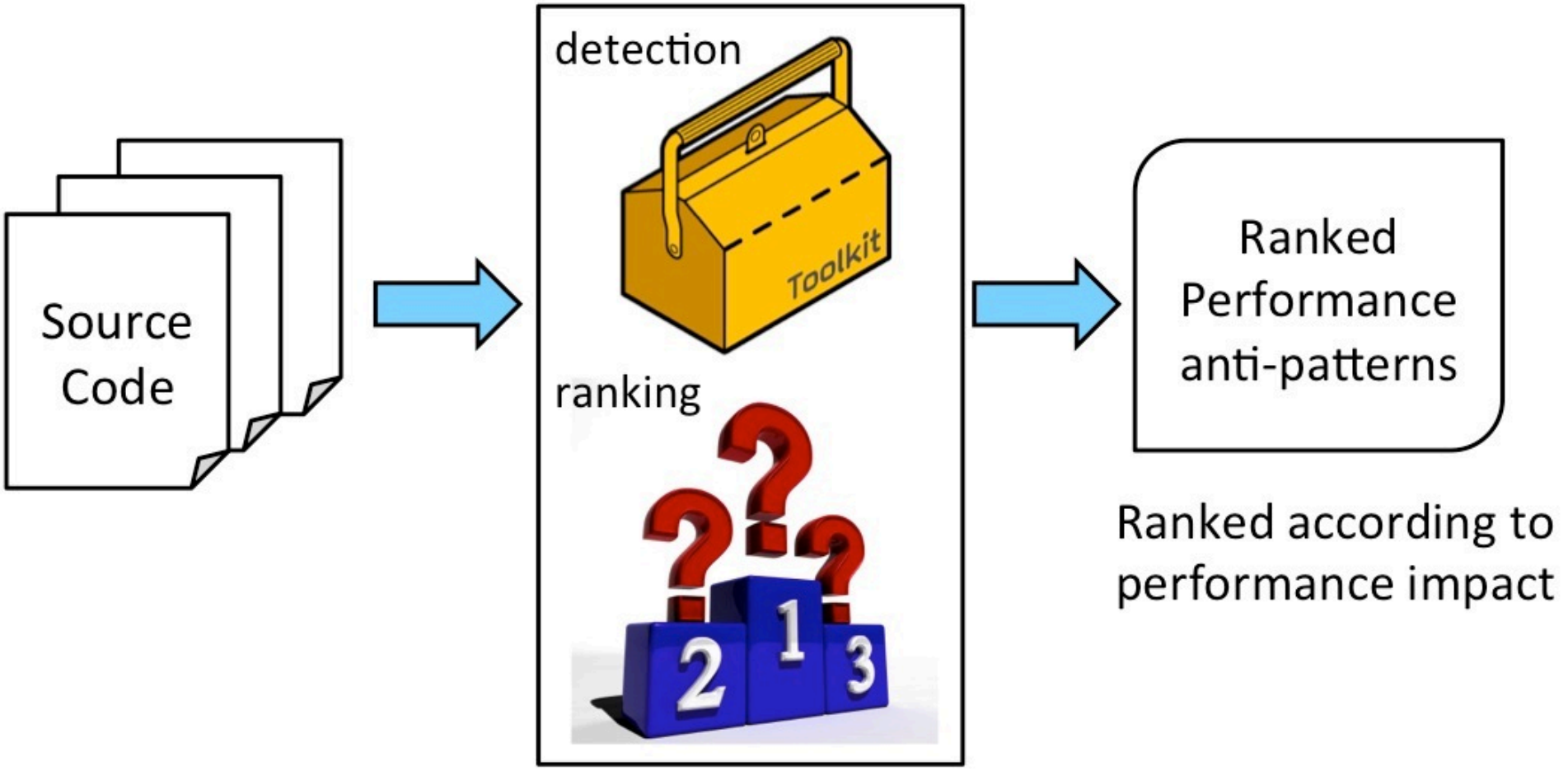
Problem of using raw SQLs

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

Accessing the database using ORM



Performance anti-pattern detection framework



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

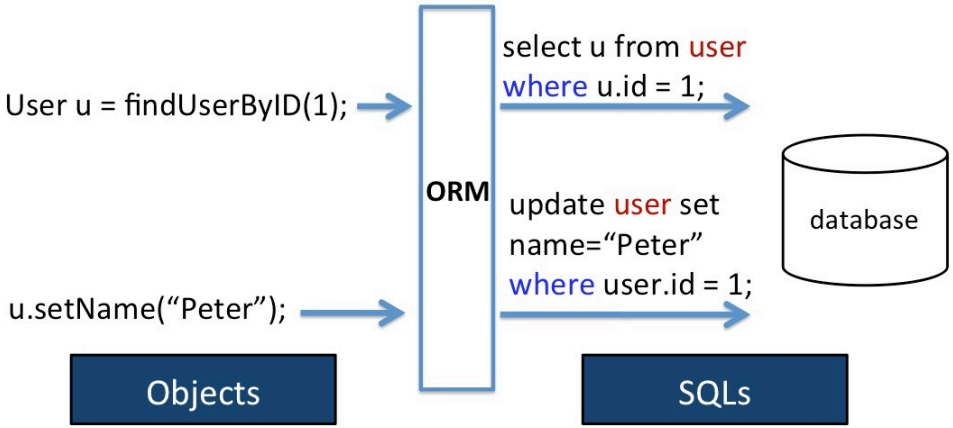
Object-Relational Mapping eliminates the gap between objects and SQL

Much less code and shorter development time

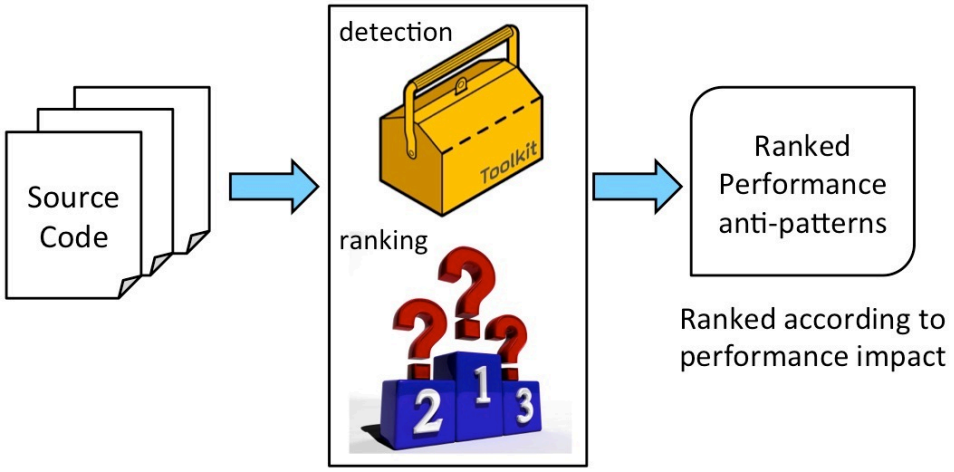
Problem of using raw SQLs

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

Accessing the database using ORM



Performance anti-pattern detection framework



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

Review anti-pattern (what you should NOT do)

Obvious results: Students are asked to rethink the validity of such a critique had they not read the paper (since quite often things look quite obvious once an elegant and simple solution has been proposed).

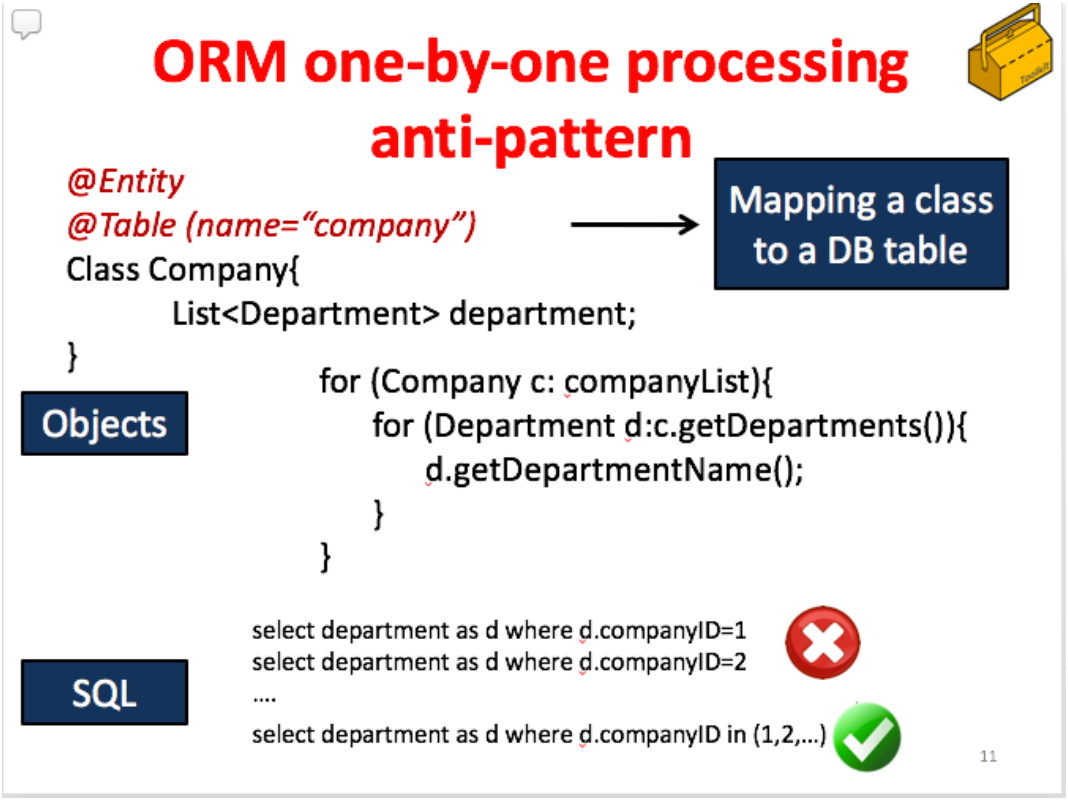
N+1 systems: Students are asked to think deeper about the goal of our field – Are we in search for a unifying theory that unifies knowledge across the whole field (and hence a considerable amount of systems must be studied for each paper)? Or are we more case study focused and are concerned about showing that the work can help at least a few systems (that are possibly impacting the lives of many)?

Industry vs. open source: A classic critique is to complain that the studied projects are open source ones or industrial ones. Students are asked to go beyond the specific projects and to think about the overall impact of the results and on the availability/rarity of the studied data.

Not novel (e.g., Replication studies): This is rarely raised once students are halfway through their assignment.

Un-addressable critiques: Students are asked to combine their critique with a realistic way to address it.

Assignment



Implement a tool to detect this pattern.

And test on one of the releases in BroadLeaf

https://github.com/BroadleafCommerce/BroadleafCommerce/releases