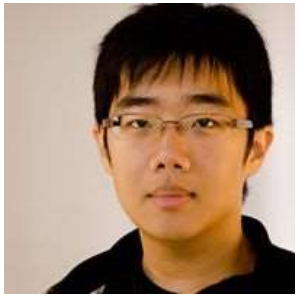
Improving the Quality of Large-Scale Database-Centric Software Systems by Analyzing Database Access Code

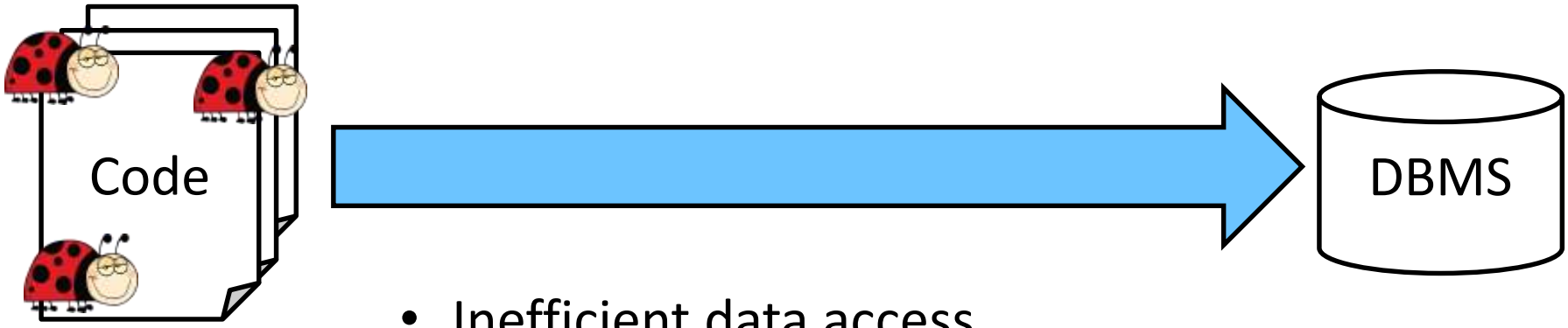


Tse-Hsun(Peter) Chen

Supervised by Ahmed E. Hassan

Queen's UNIVERSITY

Common performance problems in database-access code



- Inefficient data access

Inefficient data access

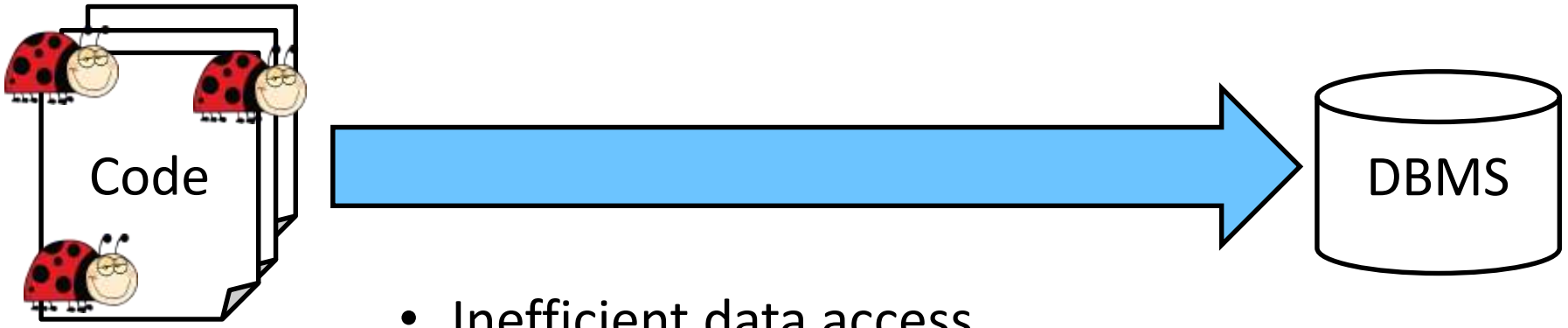
Code

```
for each userId{
    …
    executeQuery(“select … where
u.id = userId”);
}
```

SQL

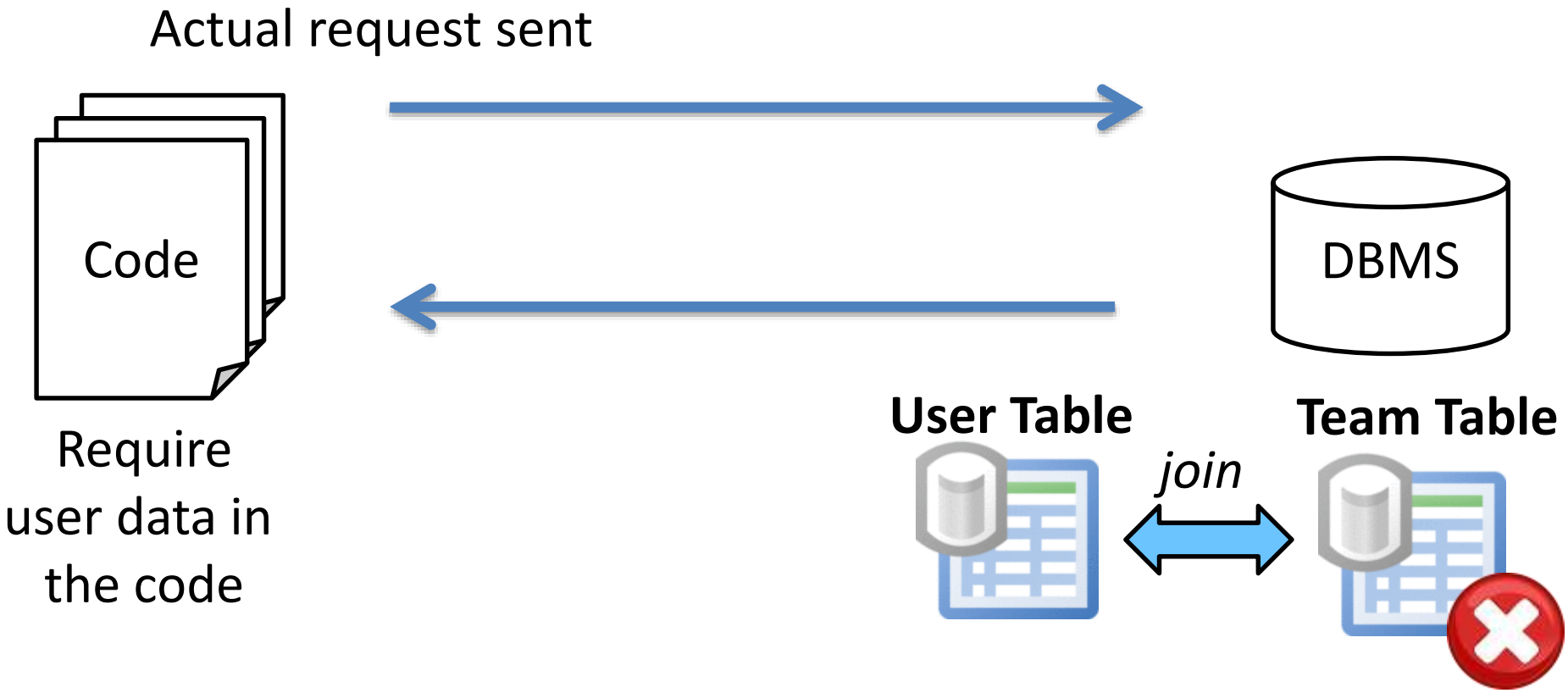
```
select … from user where u.id = 1
select … from user where u.id = 2
…
```

Common performance problems in database-access code

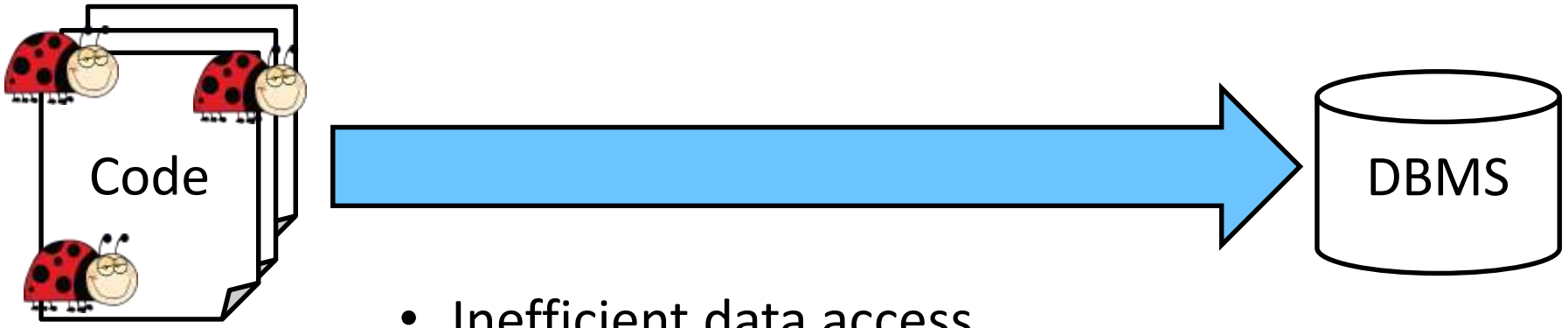


- Inefficient data access
- Unneeded data access

Unneeded data access

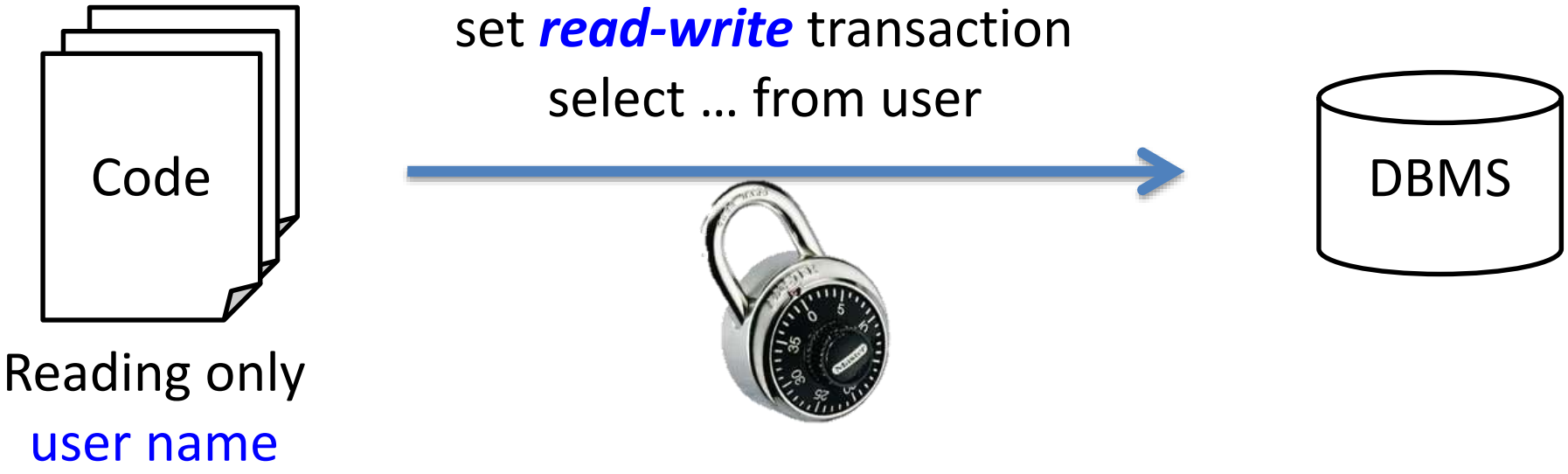


Common performance problems in database-access code



- Inefficient data access
- Unneeded data access
- Overly-strict isolation level

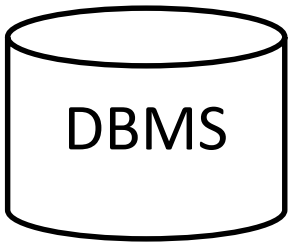
Overly-strict isolation level



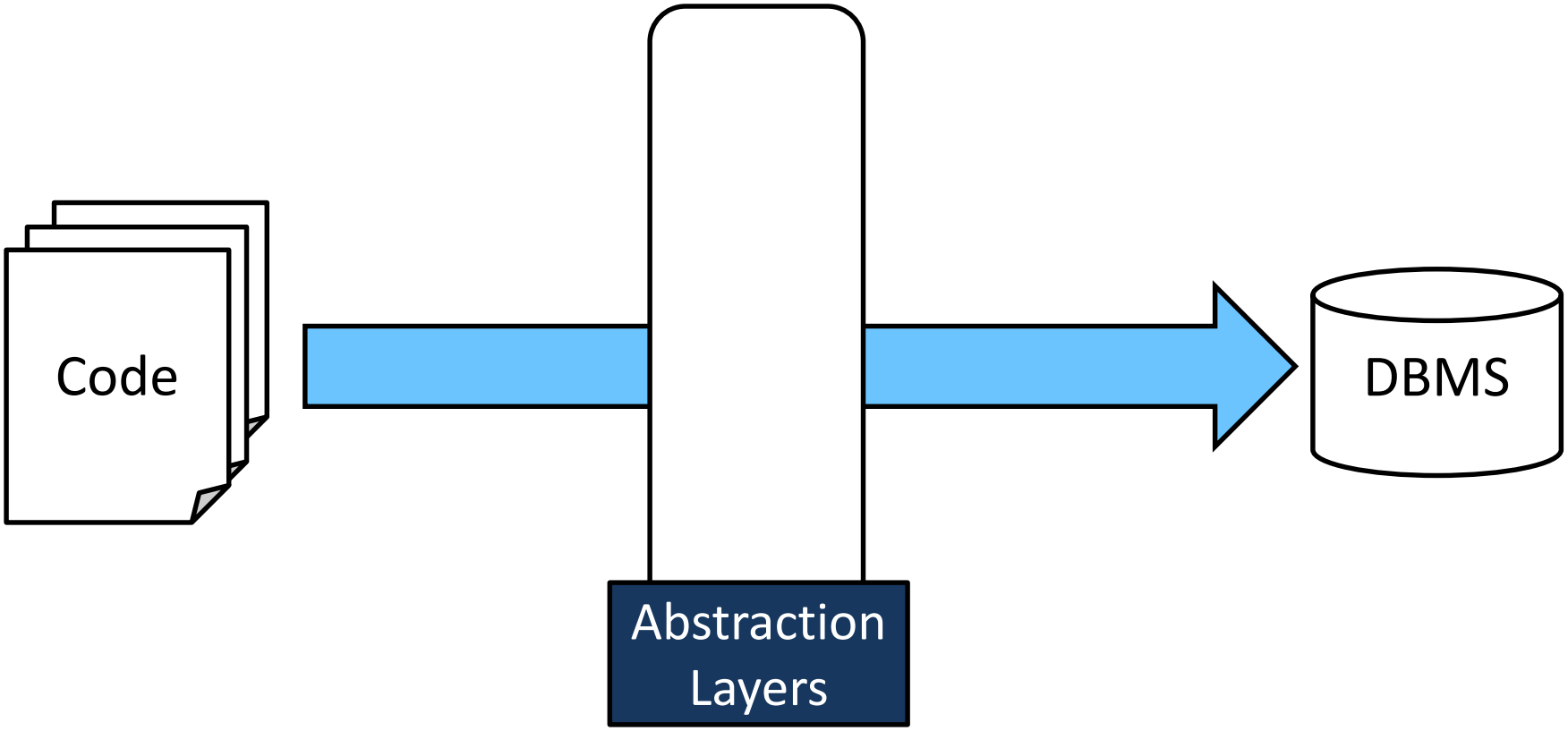
Cannot find problems by only looking at the DBMS side

select … from user where u.id = 1
select … from user where u.id = 2
…

```
for each userId{
   …
   executeQuery(“
select … where u.id = userId”);
}
```



Database accesses are abstracted



Problems become more complex and frequent after adding abstraction layers

Accessing data using ORM incorrectly

```
@Entity
Class User{
        …
}
```

A database entity class

Objects

```
for each userId{
   User u = findUserById(userId);
   u.getName();
}
```

SQL

```
select … from user where u.id = 1
select … from user where u.id = 2
…
```

Research statement

There are common problems in how developers write database-access code, and these problems are hard to diagnose by only looking at the DBMS. In addition, due to the use of different database abstraction layers, these problems become even harder to find.

By finding problems in the database access code and configuring the abstraction layers correctly, we can significantly improve the performance of database-centric systems.

Overview of the thesis

Detecting inefficient data access code	Detecting unneeded data access	Finding overly-strict isolation level
Finished work	Under submission	Future work

Overview of the thesis

Detecting inefficient data access code

Finished work

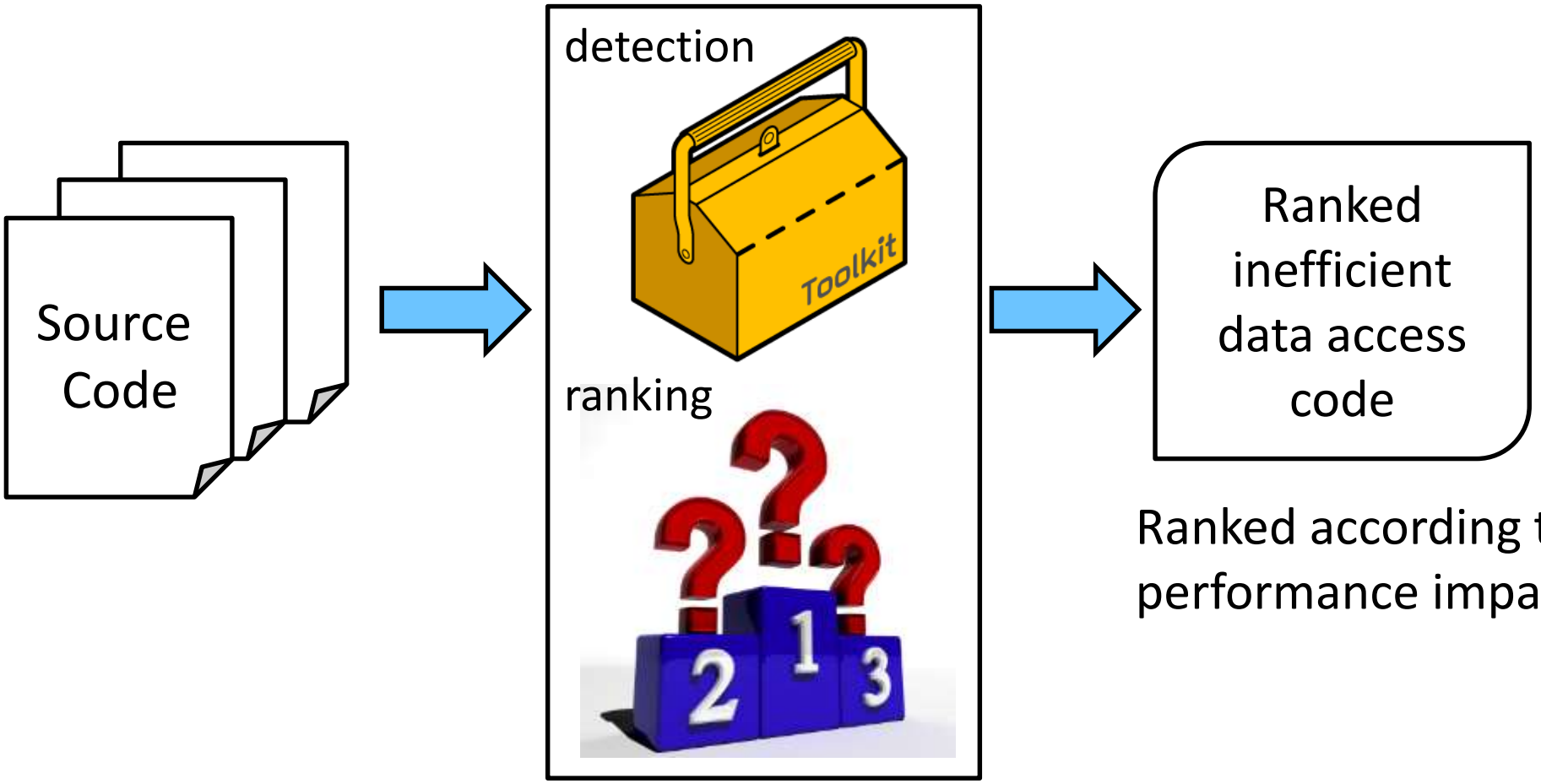
Detecting unneeded data access

Under submission

Finding overly-strict isolation level

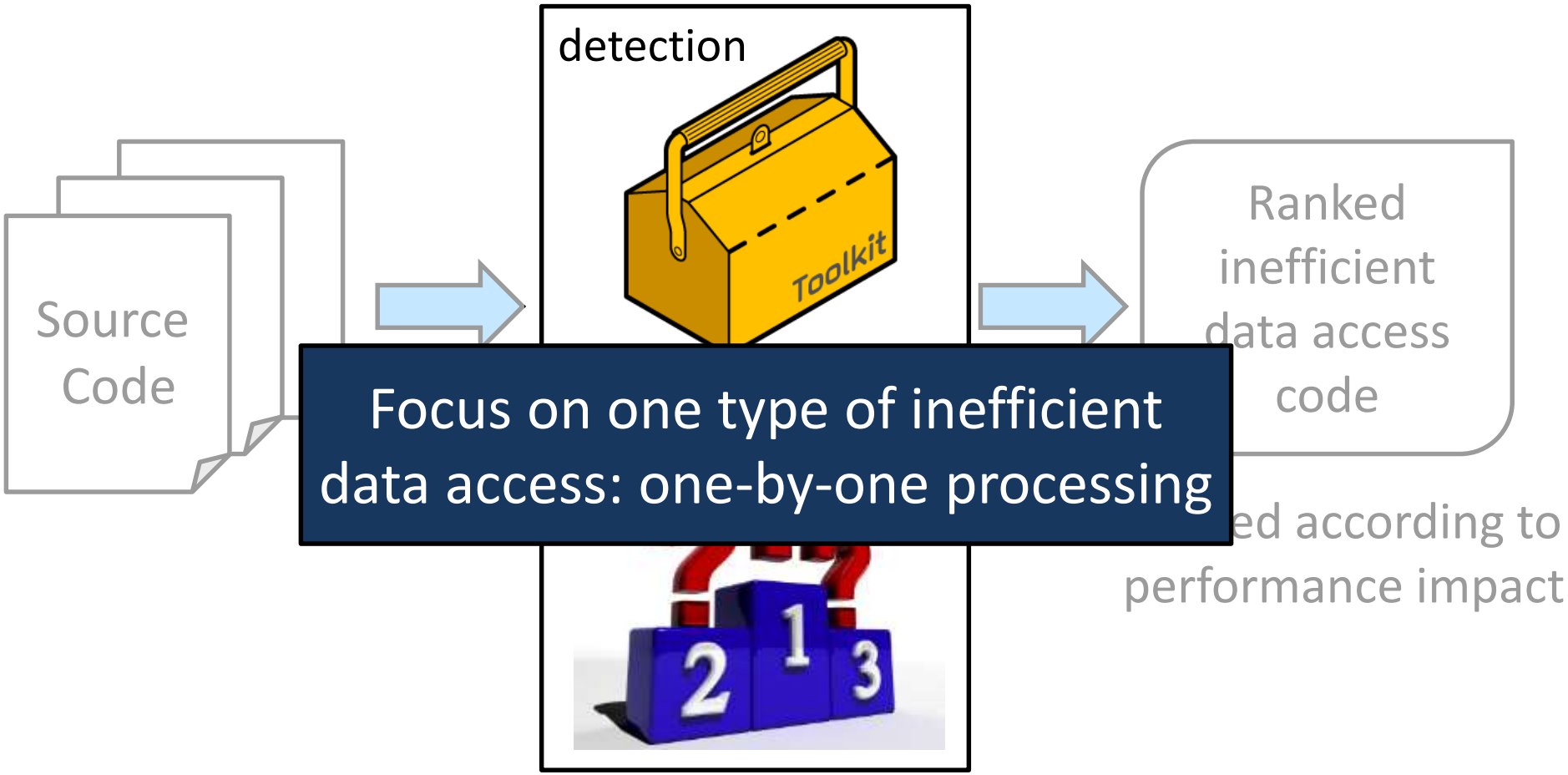
Future work

Inefficient data access detection framework



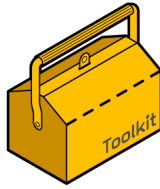
Inefficient data access detection and ranking framework

Inefficient data access detection framework



Inefficient data access detection and ranking framework

Detecting one-by-one processing using static analysis



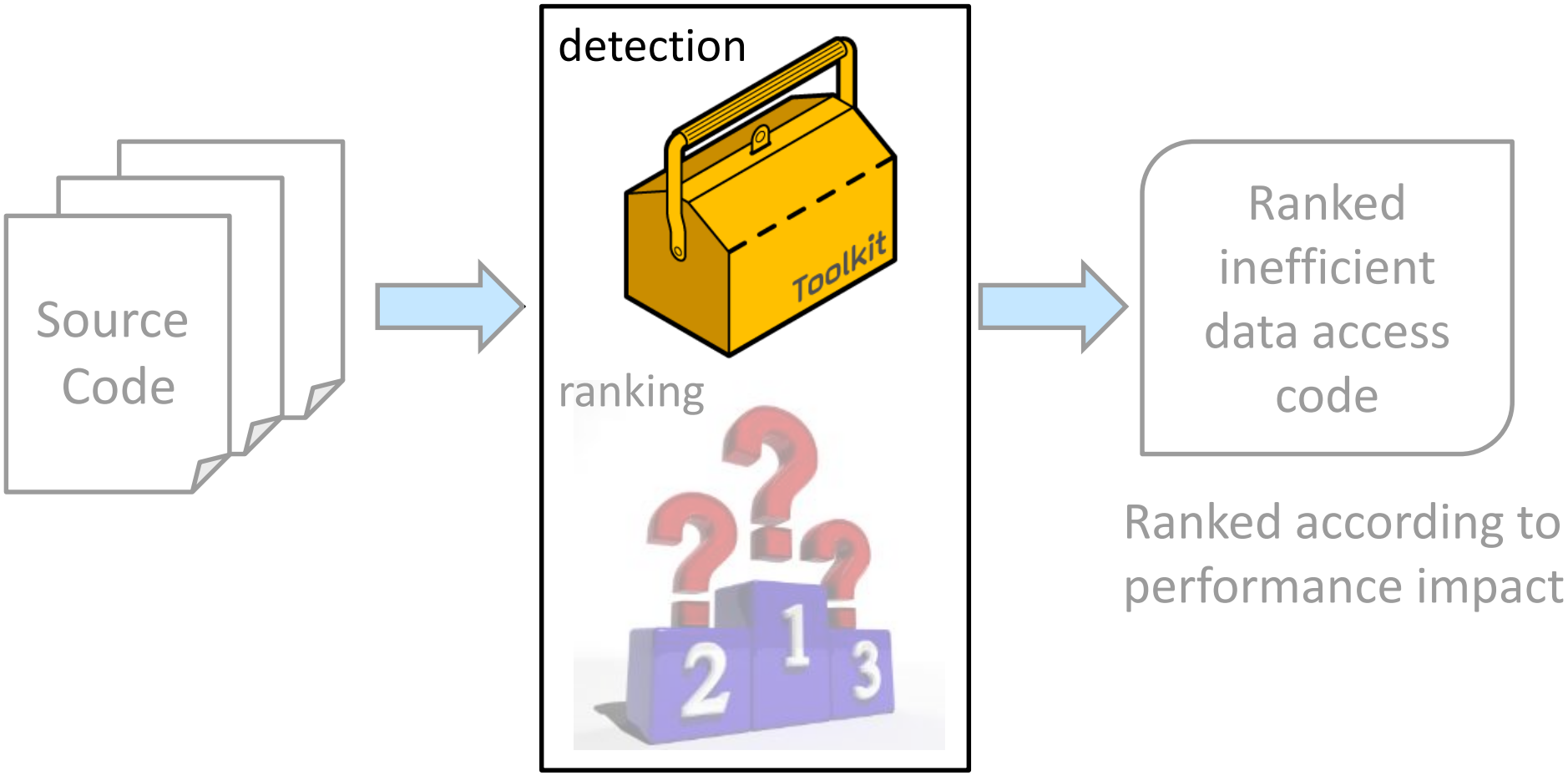

```
Class User{
    getUserById()…
    getUserAddress()…
}

for each userId{
    foo(userId)
}

foo (userId){
    getUserById(userId)
}
```

First find all the functions that read/write from/to DBMS

Identify the positions of all loops

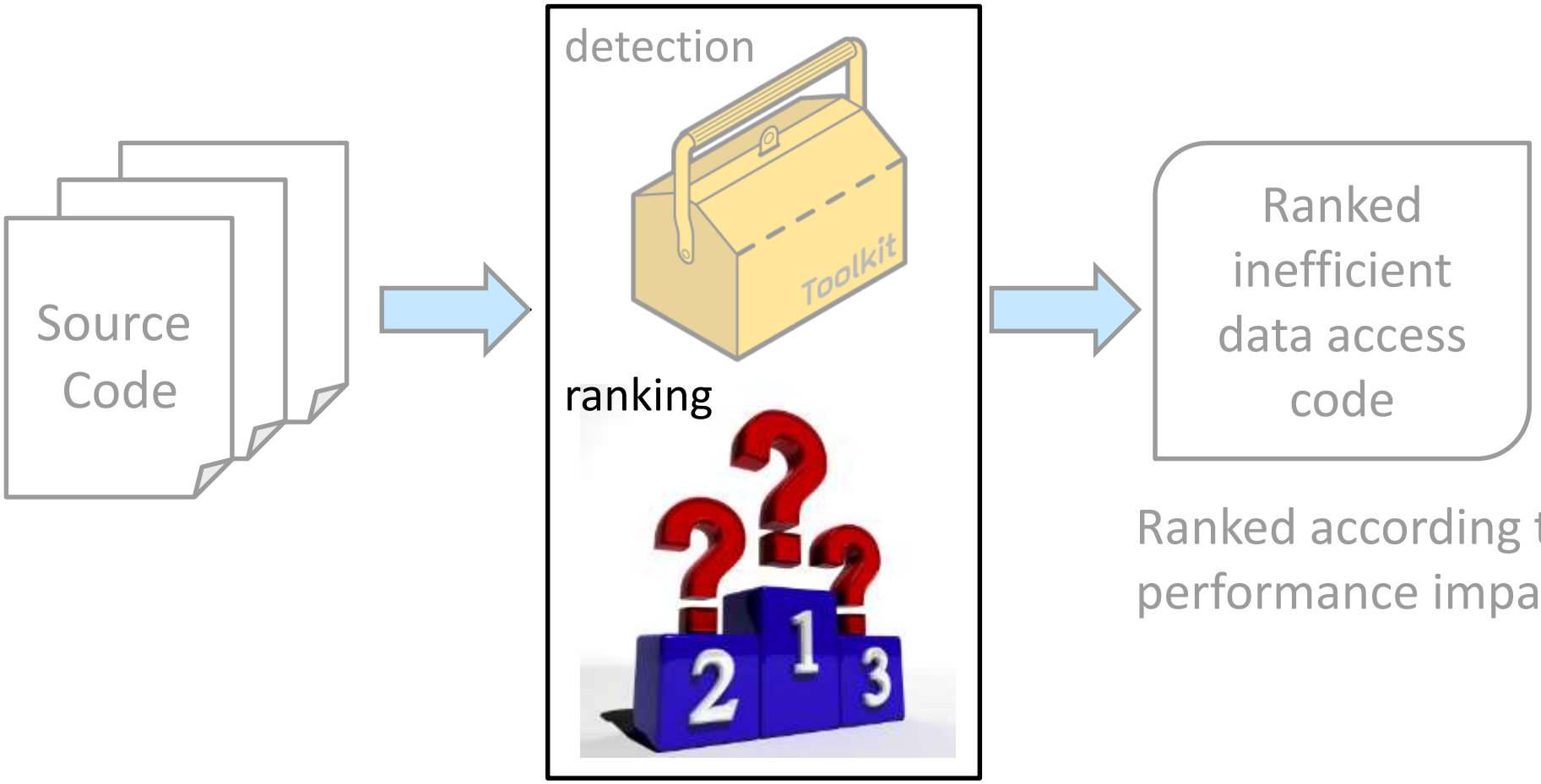
Check if the loop contains any database-accessing function

Inefficient data access detection framework



Inefficient data access detection and ranking framework

Inefficient data access detection framework



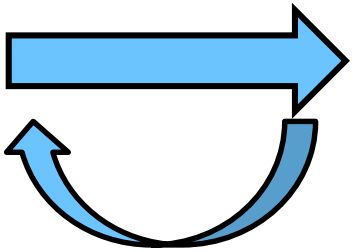
Inefficient data access detection and ranking framework

Assessing inefficient data access impact by fixing the problem

```
for u in users{
   update u
}
```

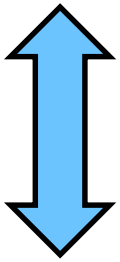
Code with inefficient data access

Execution



Execute test suite 30 times

Response time

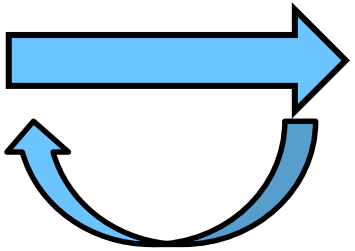


Avg. % improvement

```
users.updateAll()
```

Code ***without*** inefficient data access

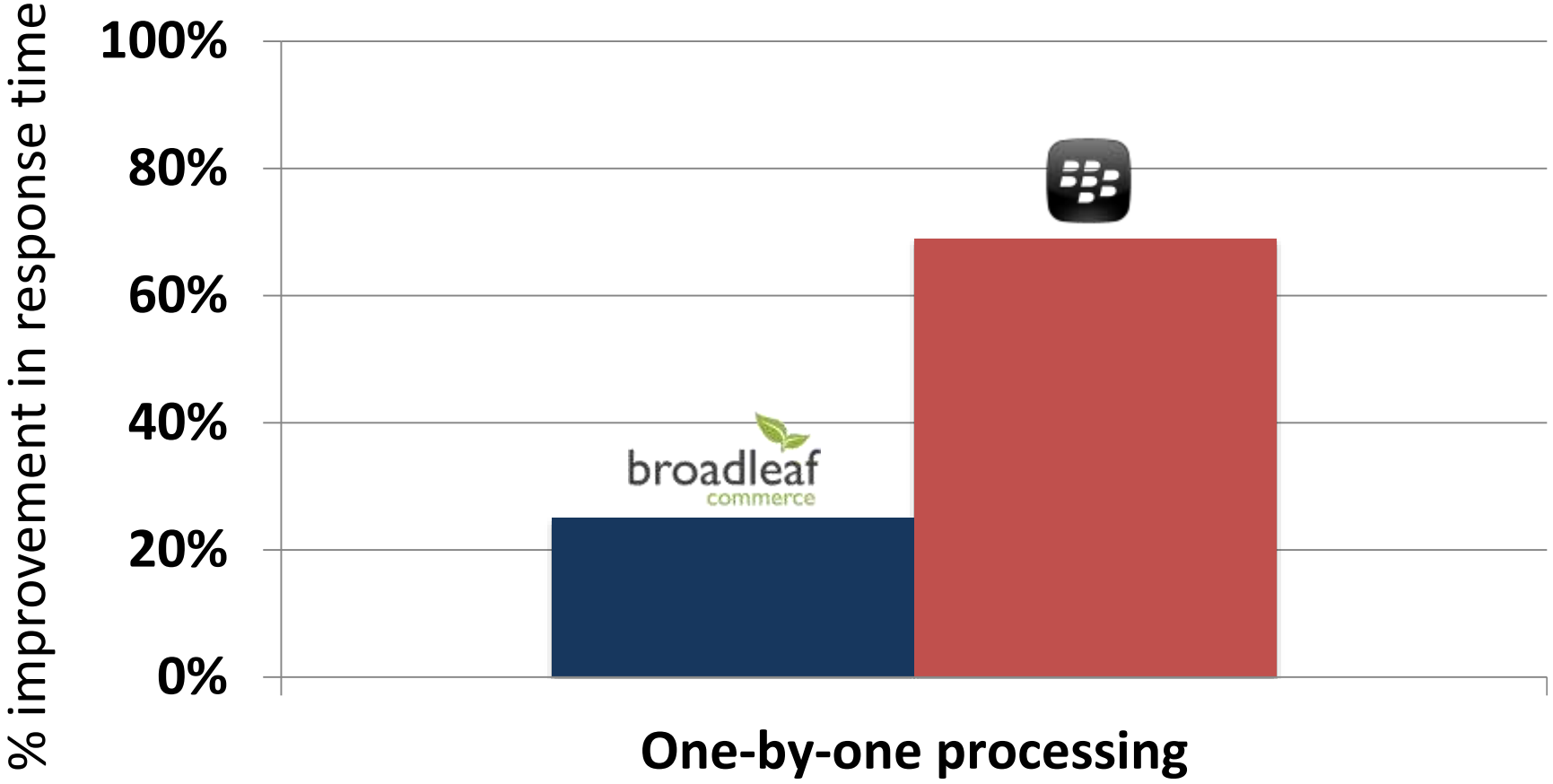
Execution



Execute test suite 30 times

Response time *after the fix*

Inefficient data access causes large performance impacts



Overview of the thesis

Detecting inefficient data access code

Finished work

Detecting unneeded data access

Under submission

Finding overly-strict isolation level

Future work

Overview of the thesis

Detecting inefficient data access code

Finished work

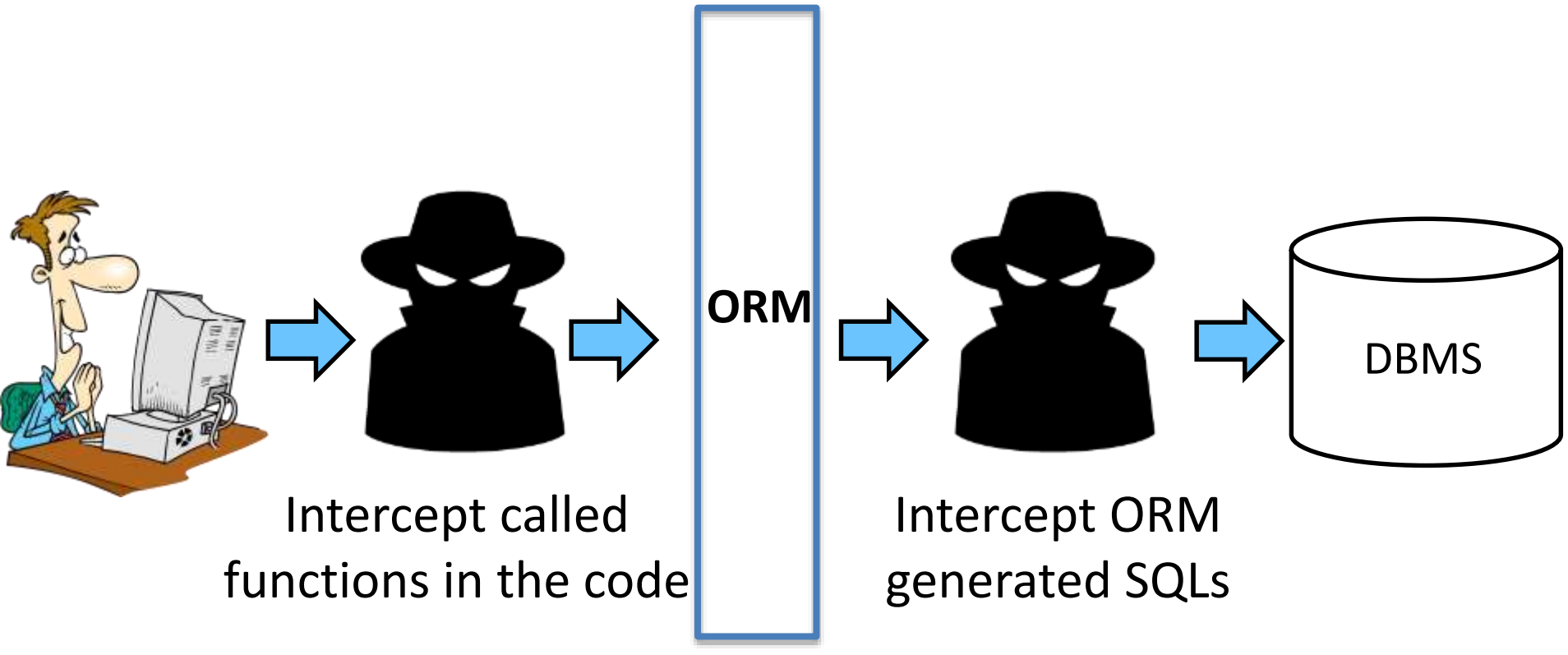
Detecting unneeded data access

Under submission

Finding overly-strict isolation level

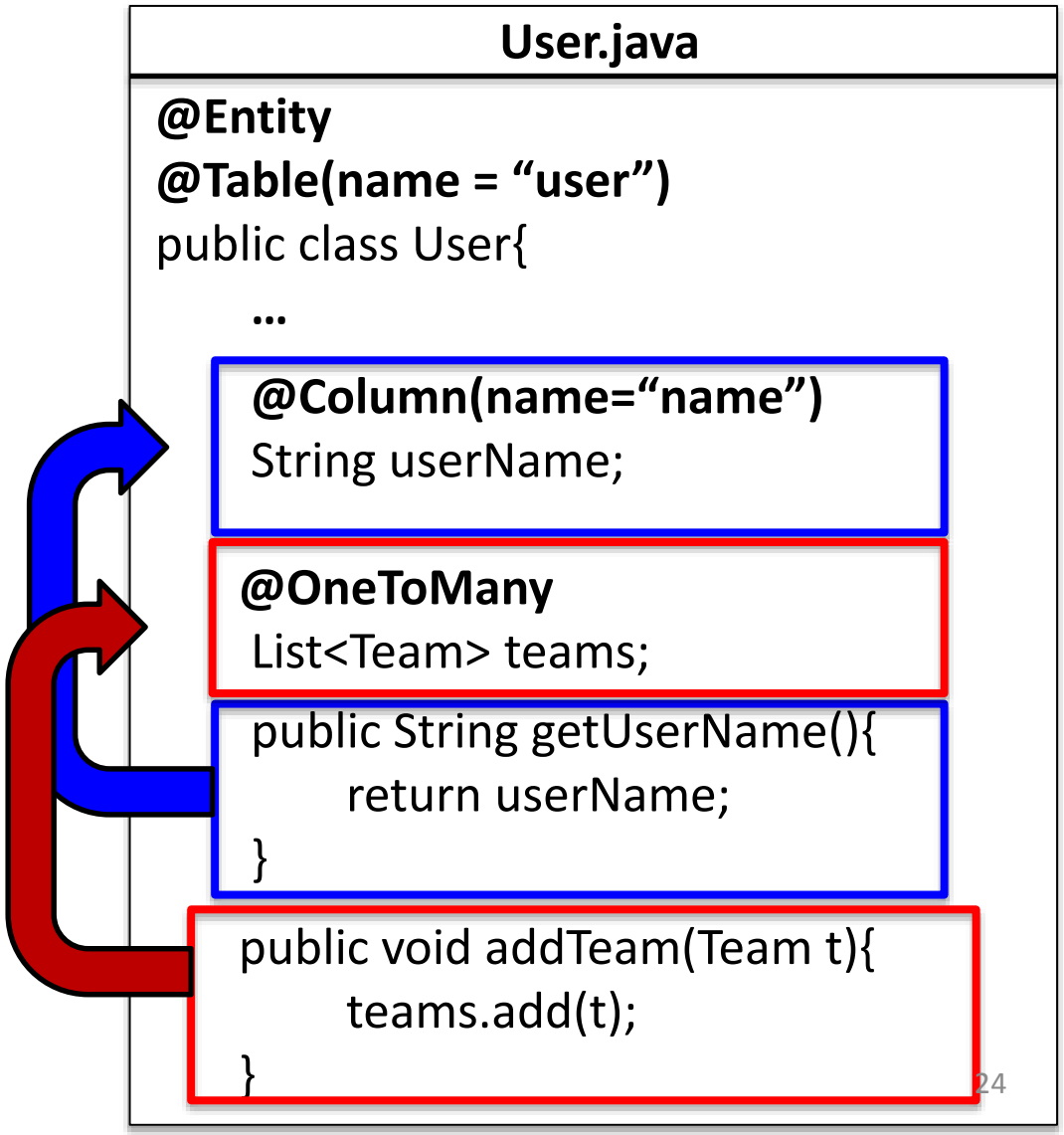
Future work

Intercepting system executions using byte code instrumentation



Mapping data access to called functions

We apply static analysis to find which database column a function is reading/modifying



Identifying unneeded data access from intercepted data

```
<transaction>
        <functionCall>
                user.getUserName()
        </functionCall>
        <sql>
                select u.id, u.name, u.address,
                u.phone number from User u
                where u.id=1
        </sql>
</transaction>
```

Needed user name in code

Requested id, name, address, phone number

Only *user n*ame is needed in the application logic

Overview of the thesis

Detecting inefficient data access code

Finished work

Detecting unneeded data access

Under submission

Finding overly-strict isolation level

Future work

Overview of the thesis

Detecting inefficient data access code

Finished work

Detecting unneeded data access

Under submission

Finding overly-strict isolation level

Future work

A real life example of transaction abstraction problem

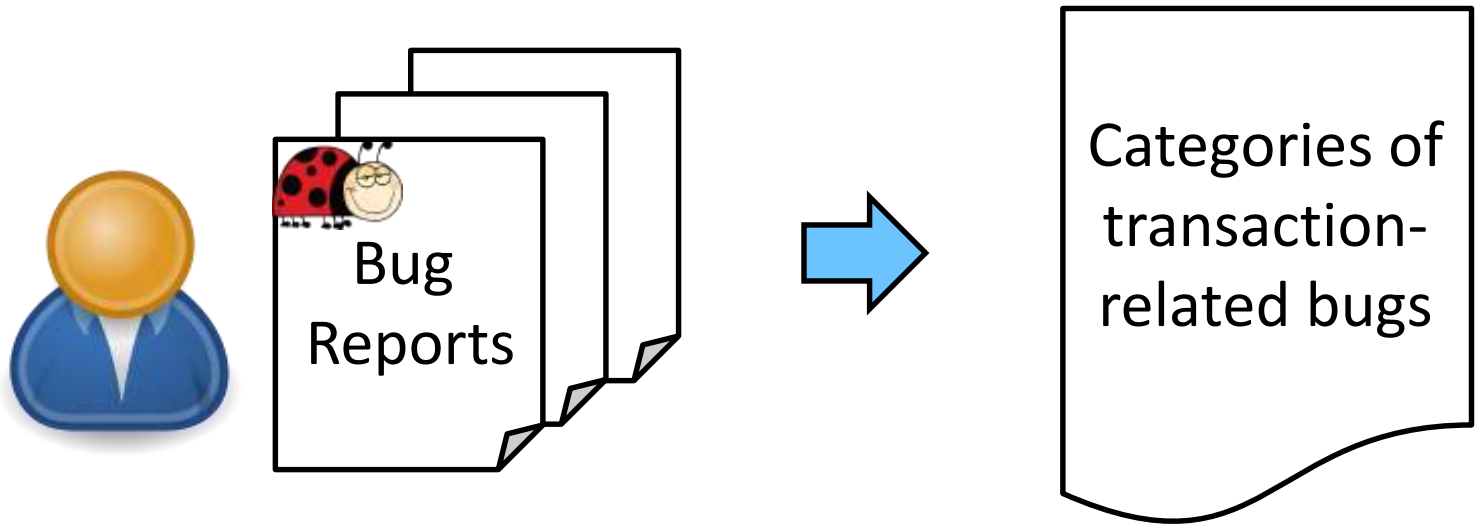
```
@Transactional
public class foo{

    int getFooVal(){
        return foo.val;
    }
    ...
}
```

All functions in *foo* will be executed in transactions

This function ***does not*** need to be executed in transactions

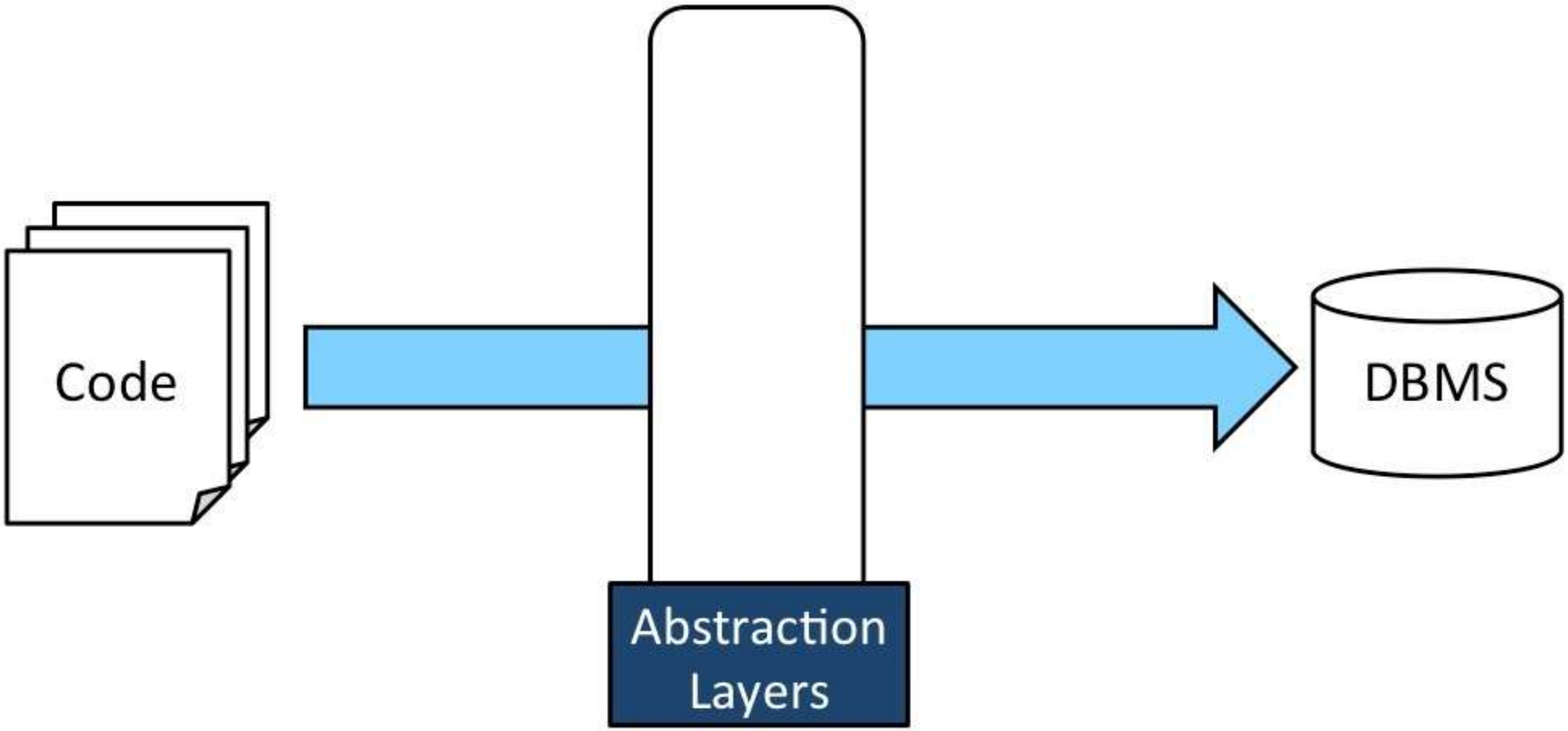
Where to put the transaction may affect system behavior and performance

Plans for finding transaction problems related abstraction



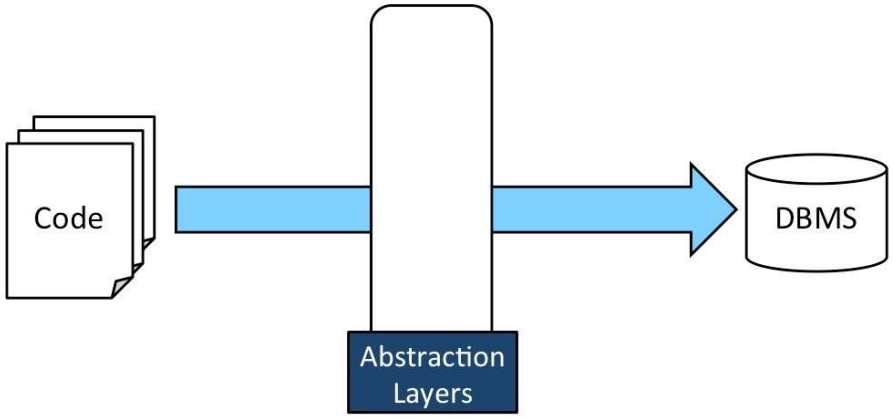
We plan to empirically study bug reports to find root causes and implement a detection framework

Database accesses are abstracted



Problems become more complex and frequent after adding abstraction layers

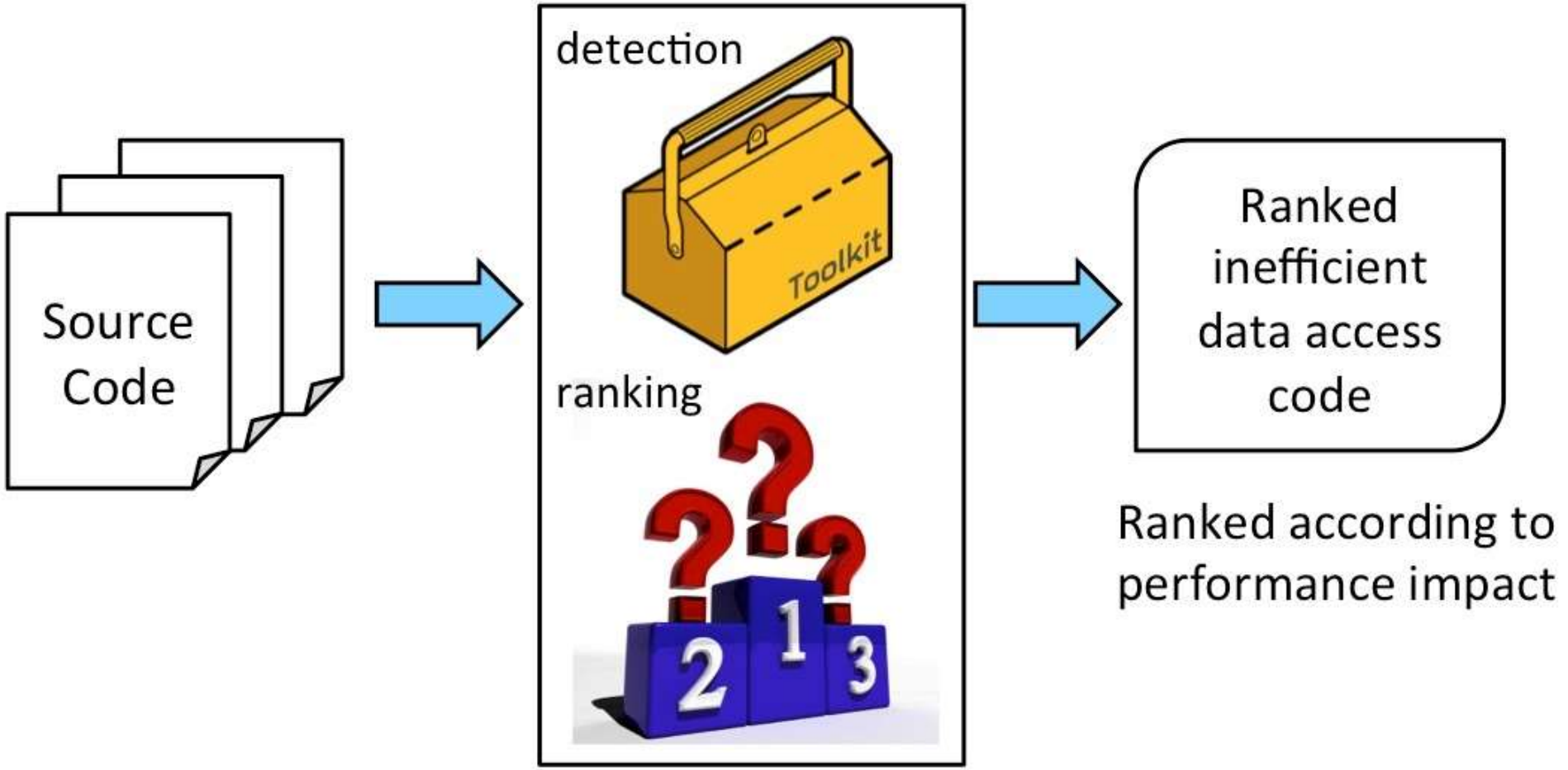
Database accesses are abstracted



Problems become more complex and frequent after adding abstraction layers

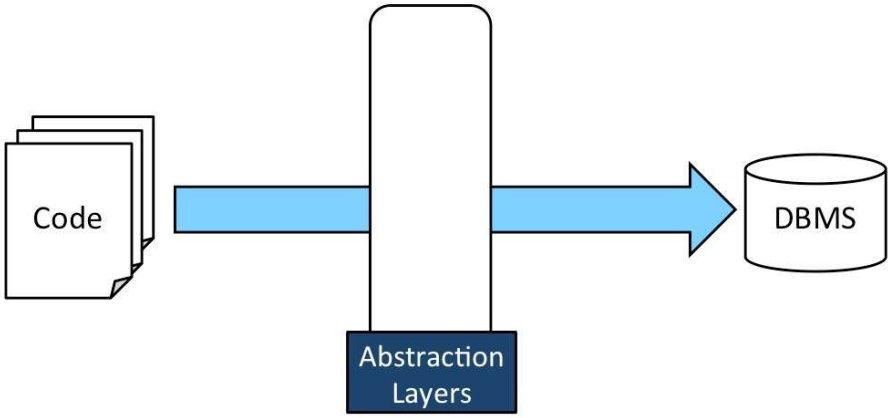
9

Inefficient data access detection framework



Inefficient data access detection and ranking framework

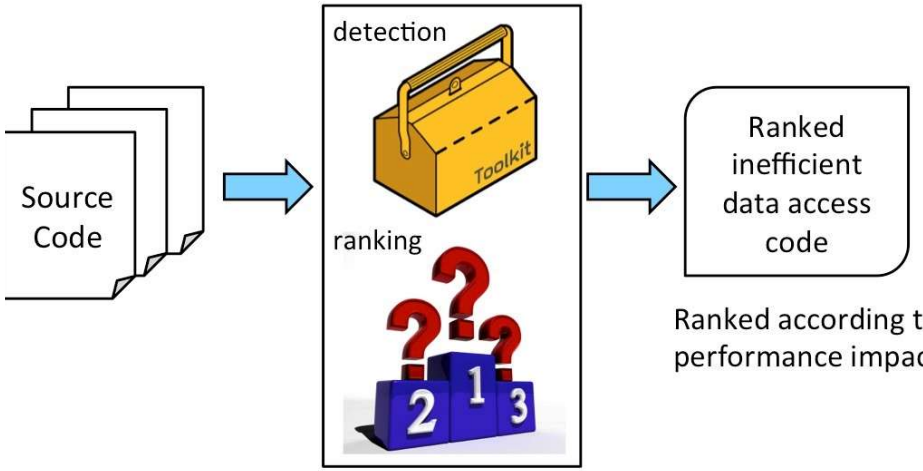
Database accesses are abstracted



Problems become more complex and frequent after adding abstraction layers

9

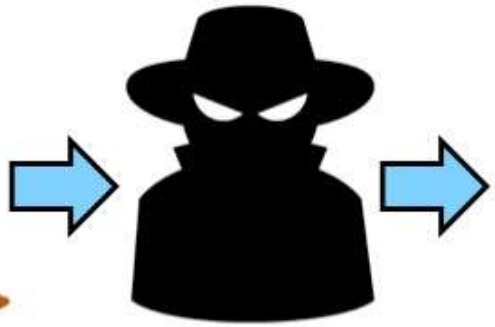
Inefficient data access detection framework



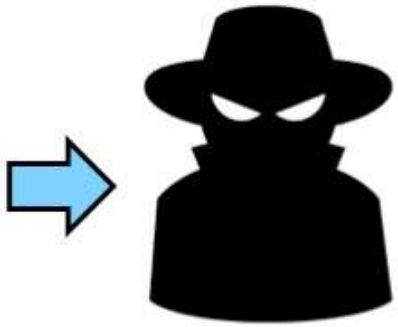
Inefficient data access detection and ranking framework

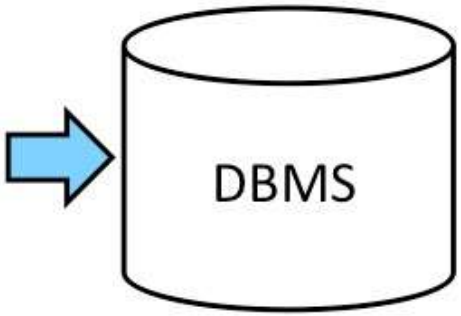
14

Intercepting system executions using byte code instrumentation



ORM

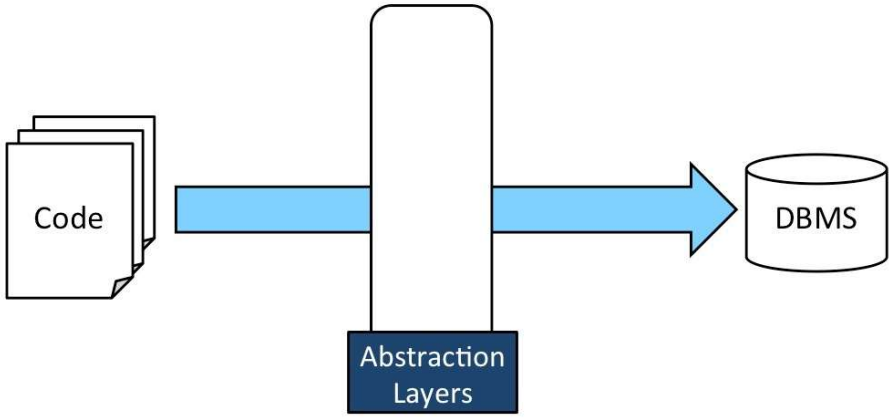




Intercept called functions in the code

Intercept ORM generated SQLs

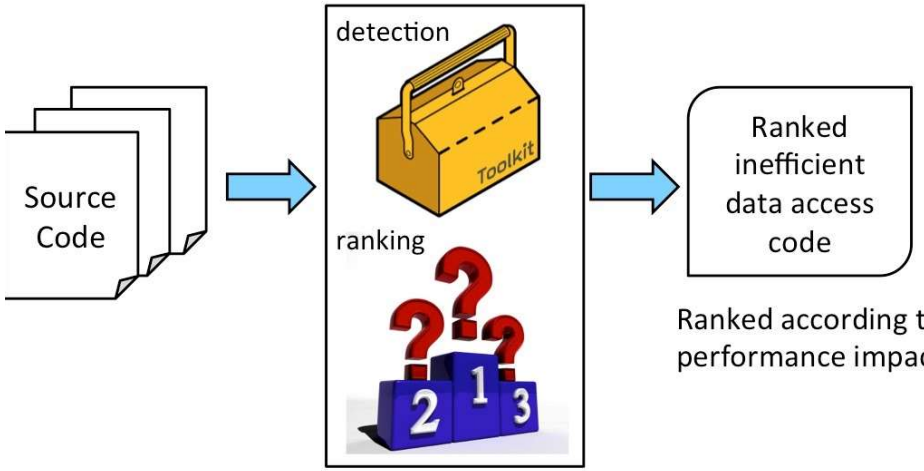
Database accesses are abstracted



Problems become more complex and frequent after adding abstraction layers

9

Inefficient data access detection framework

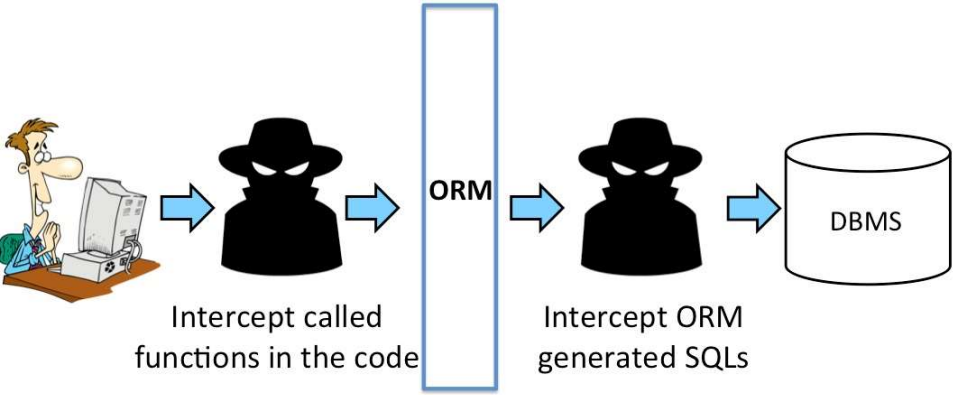


Ranked according to performance impact

Inefficient data access detection and ranking framework

14

Intercepting system executions using byte code instrumentation

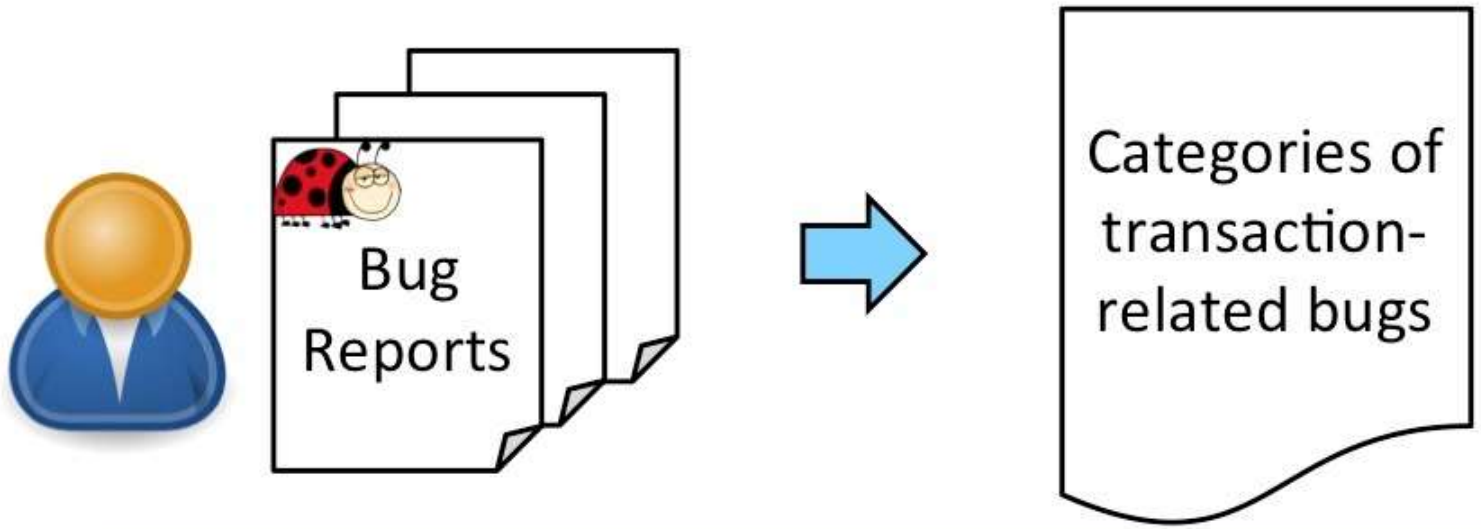


Intercept called functions in the code

Intercept ORM generated SQLs

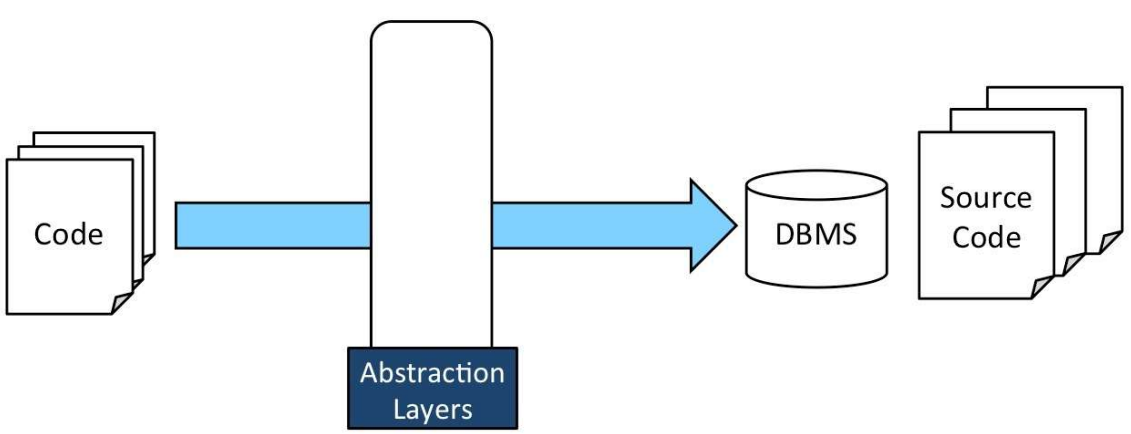
23

Plans for finding transaction problems caused by abstraction



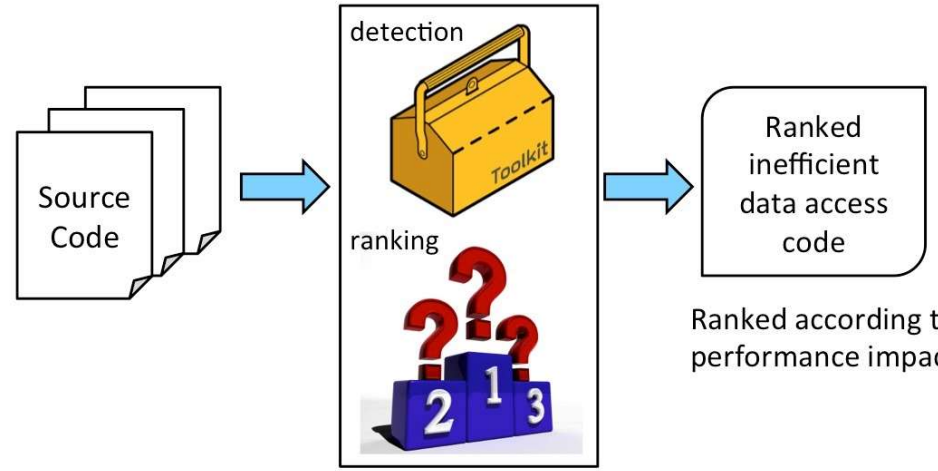
We plan to empirically study bug reports to find root causes and implement a detection framework

Database accesses are abstracted



Problems become more complex and frequent after adding abstraction layers

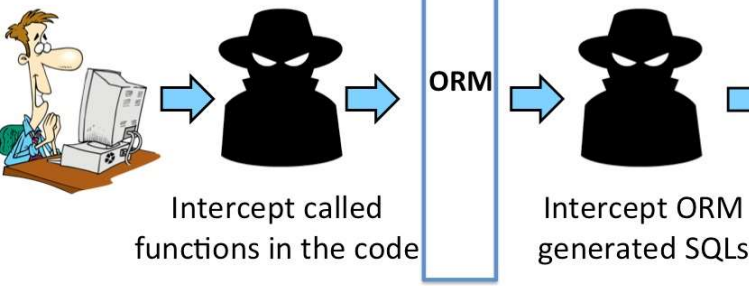
Inefficient data access detection framework



Inefficient data access detection and

http://petertsehsun.github.io

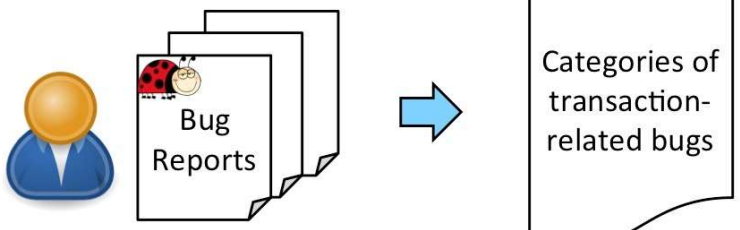
Intercepting … byte co… …ransaction abstraction



Intercept called functions in the code

Intercept ORM generated SQLs

Needed data ⟺ *diff* Requested data



We plan to empirically study bug reports to find root causes and implement a detection framework