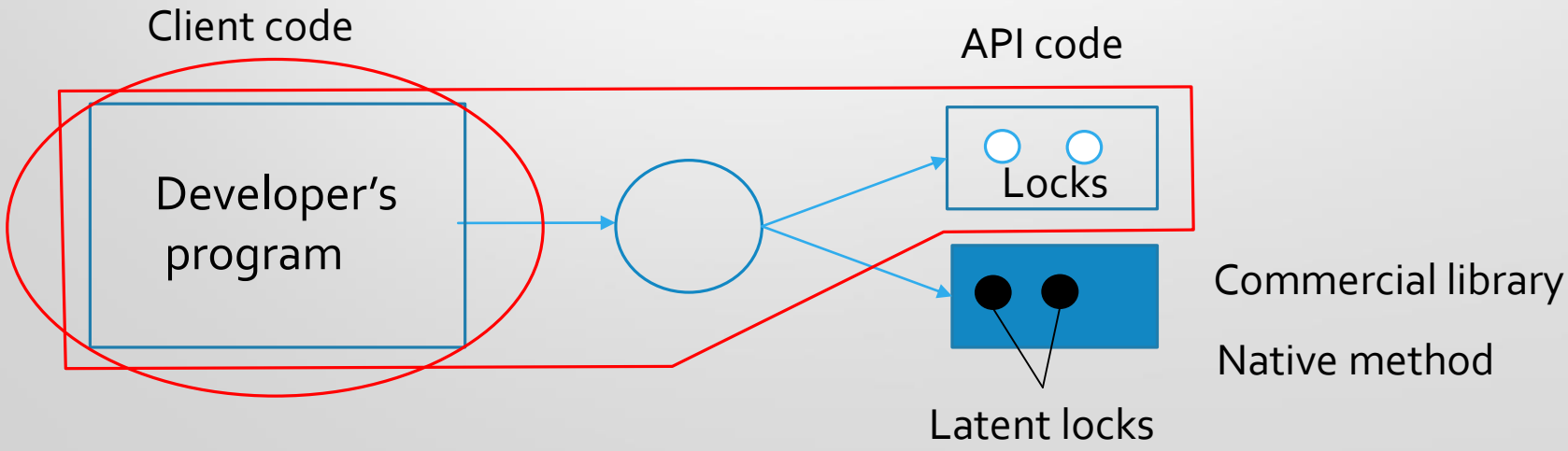
LockPeeker: Detecting Latent Locks in Java APIs

林子熠，钟浩，陈雨亭，赵建军

上海交通大学，日本九州大学

Introduction

- It is a hot research topic to detect deadlocks.
  - Most approaches focus on analyzing developers' program.
- What if locks in APIs whose code is unavailable?
  - We refer such locks as latent locks.



Motivating Example

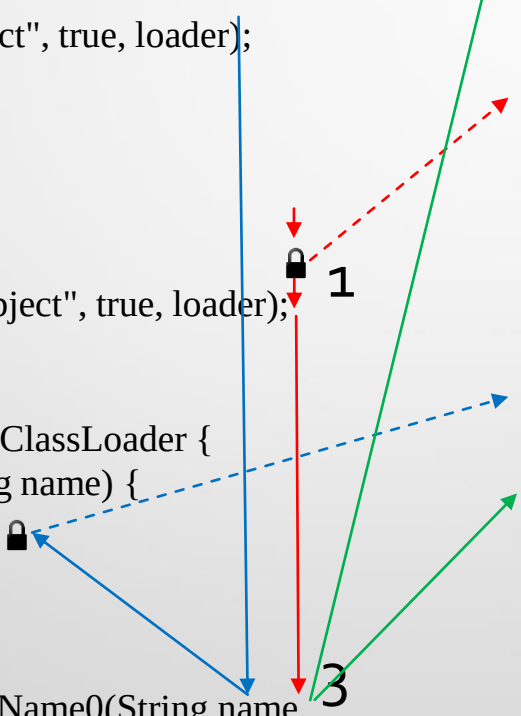
```
public class SimpleBirt287102 {
  private SimpleClassLoader loader;
  private Object obj;

class Thread1 extends Thread {
  public void run() {
    Class.forName("java.lang.Object", true, loader);
  }}

  class Thread0 extends Thread {
    public void run() {
      synchronized (obj) {
        Class.forName("java.lang.Object", true, loader);
  }}}

class SimpleClassLoader extends ClassLoader {
  public Class<?> loadClass(String name) {
    synchronized (obj) {...}
    return super.loadClass(name);
}}}

 private static native Class<?> forName0(String name,
boolean initialize, ClassLoader loader, Class<?> caller);
```



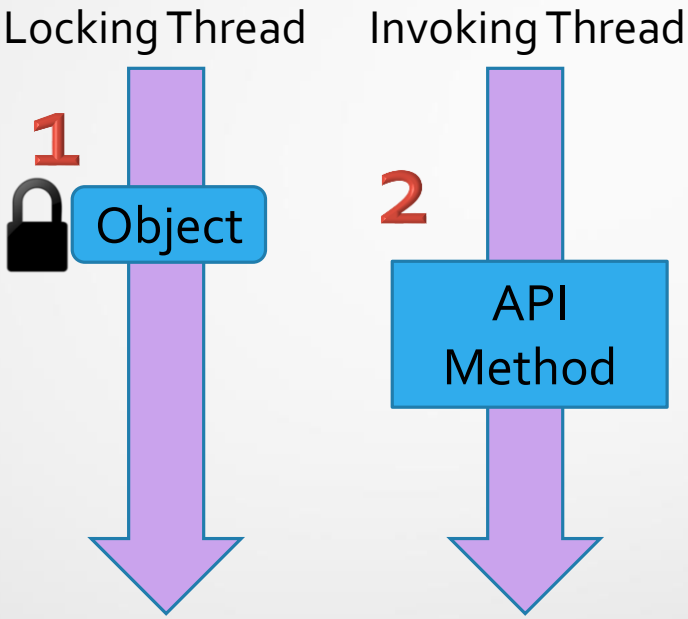
``Thread-1":
Waiting a latent lock here
at java.lang.Class.forName0(Native Method)
at java.lang.Class.forName(Class.java:348)
at SimpleBirt287102$Thread2.run(SimpleBirt287102.java:45)
- locked <0x00000000d5eee710> (a java.lang.Object)

``Thread-0":
at SimpleBirt287102$SimpleClassLoader.loadClass(SimpleBirt287102.java:64)
- waiting to lock <0x00000000d5eee710> (a java.lang.Object)
Acquire a latent lock here
at java.lang.Class.forName0(Native Method)
at java.lang.Class.forName(Class.java:348)
at SimpleBirt287102$Thread1.run(SimpleBirt287102.java:33)

How to find the latent lock in this native method?

Illustration of Idea

Locks in this paper are Java intrinsic locks, acquired by "synchronized" keyword



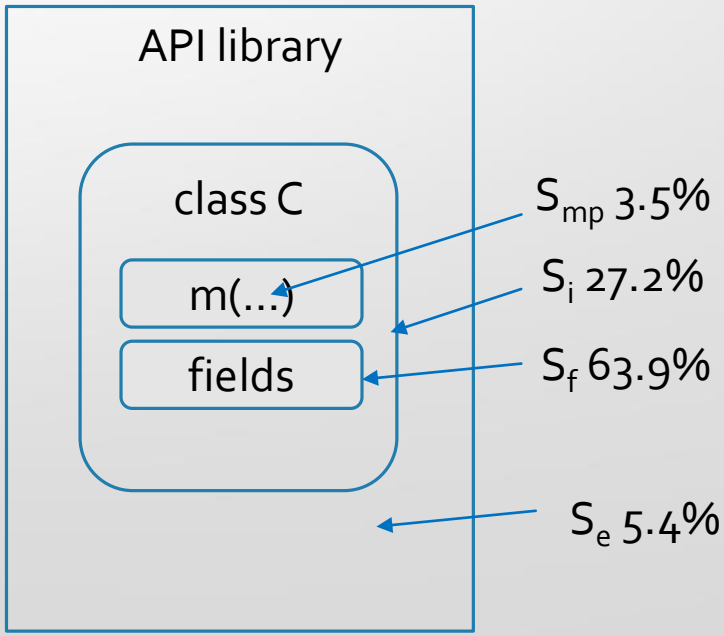
Which object could be locked?
How many locks in the API method?
What are the relations among the locks?
Are there any conditions for locking?

Approach

- Select potential locking objects
- Design stateful lock tree model to represent the locking structure in the method
- Synthesize locking structure
- Synthesize conditions

Selecting Locking Objects

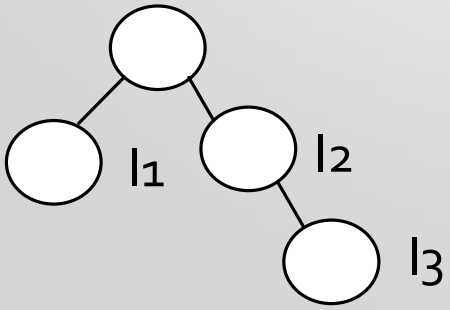
- Investigate 1,414 locks inside methods from 10 open source projects
- For an API method m in class C, the locking objects are in 4 sets:
  - S_{mp}: parameters of m
  - S_i: instance of C (this), or the class of C (C.class)
  - S_f: C's fields
  - S_e: other objects from environment
- In this paper, $S_c = S_{mp} + S_i + S_f$
- Primitive type objects are omitted



Stateful Lock Tree

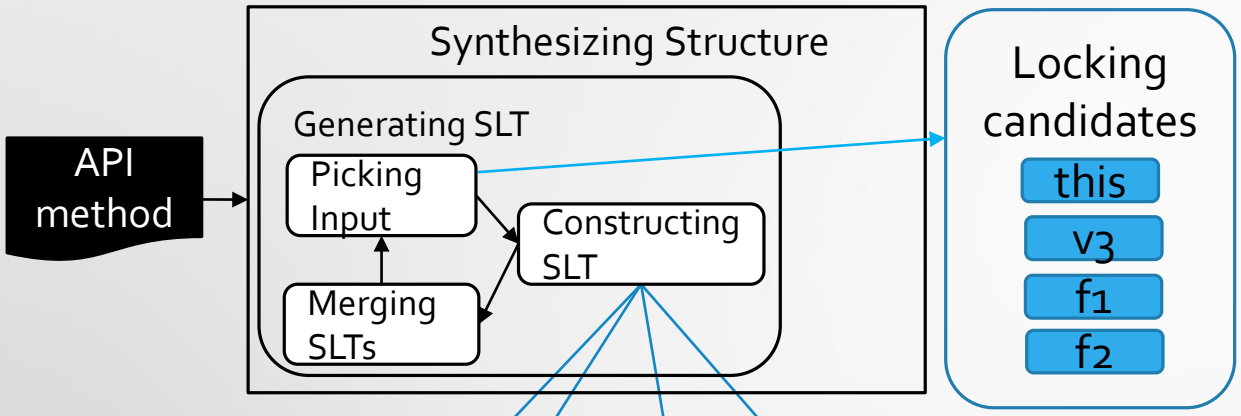
- Stateful lock tree (SLT) extends traditional lock tree to represent the locking structure in a method
  - Root denotes the API method
  - Test input values are stored in root
  - Node denotes lock
- Condition lock tree (CLT) adds condition in each node, denoting the condition to trigger the lock

Test input values



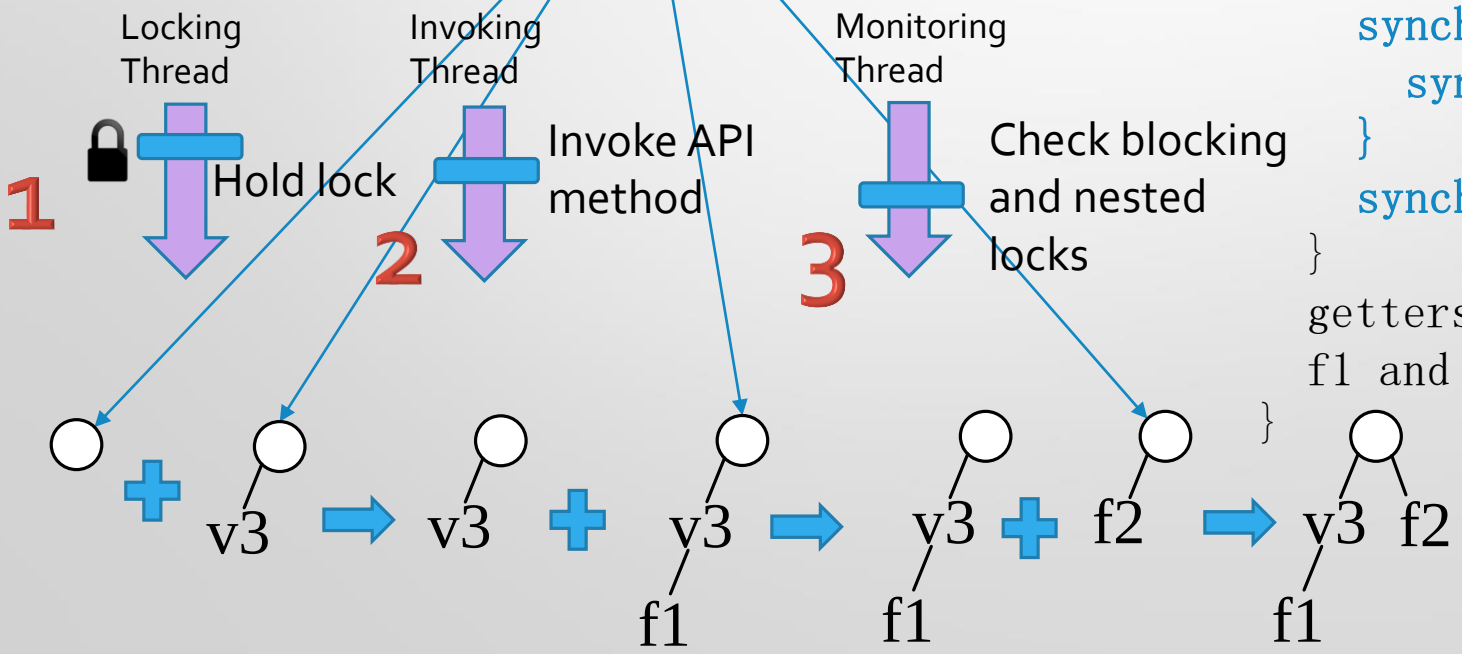

```
synchronized(l1){…}
synchronized(l2){
   synchronized(l3){…}
}
```

Synthesizing Locking Structure

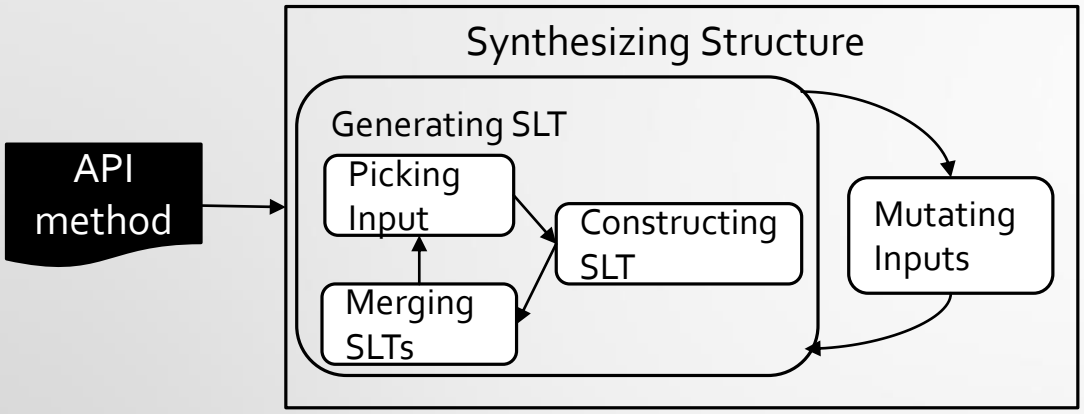



```
public class Example{
  private Object f1;
  private Object f2;

  foo(int v1, boolean v2,
      Object v3){
    synchronized(v3){
      synchronized(f1){...}
    }
    synchronized(f2){...}
  }
  getters and setters for
  f1 and f2
}
```

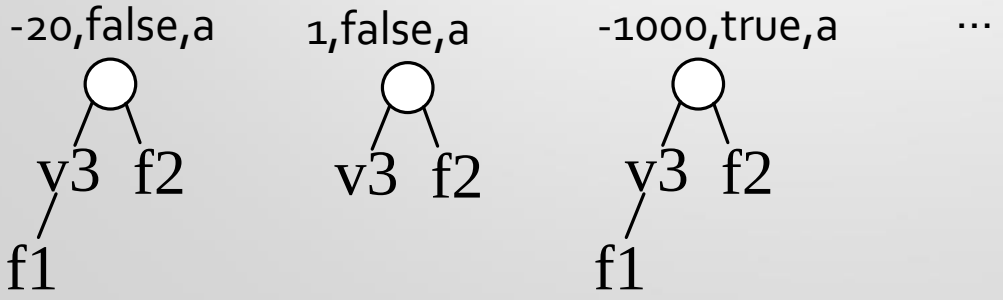


Generating More SLTs

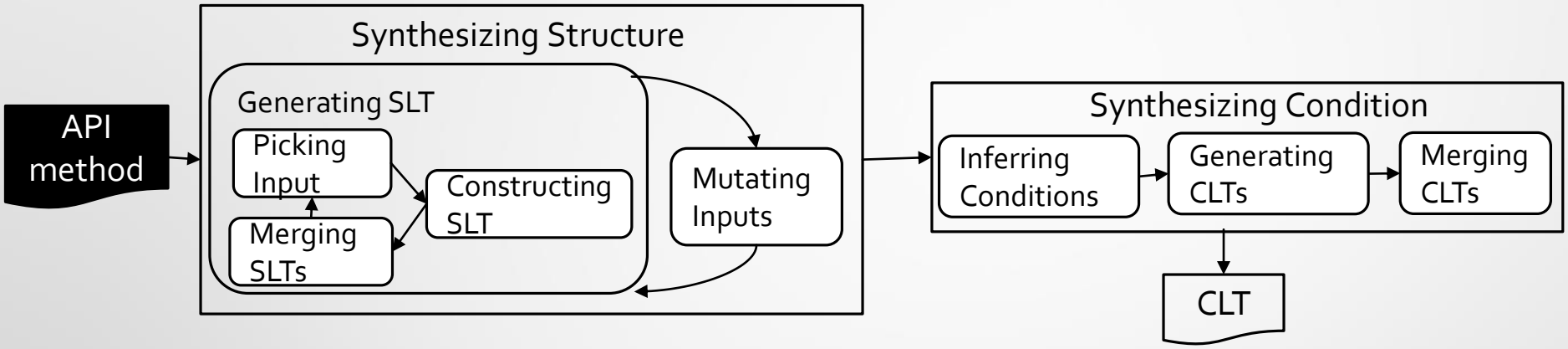



```
public class Example{
  private Object f1;
  private Object f2;

  foo(int v1, boolean v2,
      Object v3){
    synchronized(v3){
      if(v1<=0||v2)
      synchronized(f1){...}
    }
    synchronized(f2){...}
  }
  getters and setters for
  f1 and f2
}
```



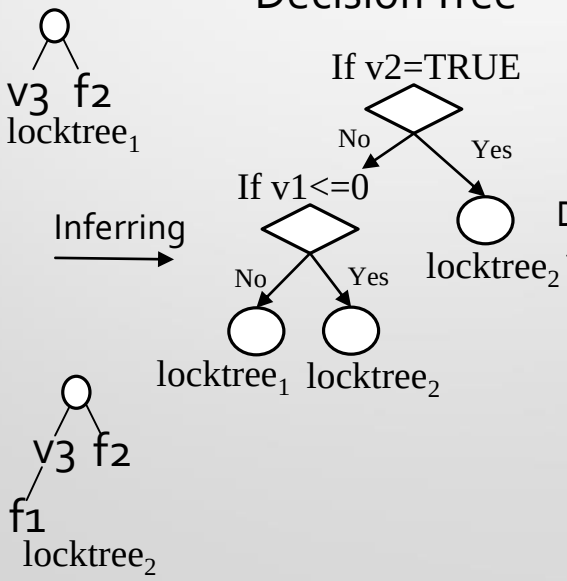
Synthesizing Condition



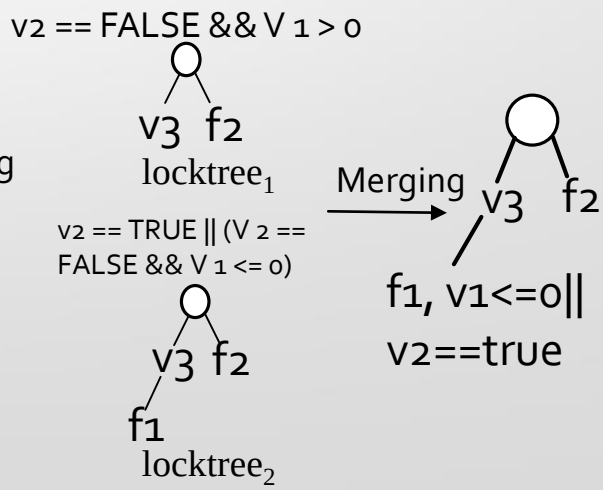
Encoded SLTs

Attributes			Class
V1	v2	v3	
-20	FALSE	a	$locktree_2$
-1000	TRUE	a	$locktree_2$
100	TRUE	a	$locktree_2$
1	FALSE	a	$locktree_1$
89978	FALSE	a	$locktree_1$
9989776	FALSE	a	$locktree_1$
0	TRUE	a	$locktree_2$
0	FALSE	a	$locktree_2$
...			

Decision Tree



Condition Lock Trees (CLT)



Evaluation

- **RQ1.** How effective is LockPeeker in revealing locks in Java API methods ?
- **RQ2.** What kinds of deadlocks can be detected, if our detected latent locks are integrated ?
- **RQ3.** What is the significance of LockPeeker's threshold ?
- **RQ4.** What are the essential test instance variables that may trigger locks ?

Evaluation: Subjects

Project	LOC	M	EM	L	EL	Version
DBCP	5,792	6	6	6	6	1.2
Derby	357,575	535	34	581	37	10.5.1.1
FtpServer	12,039	7	7	8	8	1.0.6
Groovy	119,586	42	22	44	23	1.7.9
HsqlDB	165,787	97	30	105	32	2.3.3
Log4j	15,615	39	13	43	15	1.2.15
Lucene	45,842	104	21	126	24	2.9.3
Pool	1,891	13	10	13	10	1.2
Tomcat	218,882	417	33	464	35	8.0.29
Xalan	12,039	23	13	24	13	2.7.2
Total	**955,084**	**1,283**	**189**	**1,414**	**203**	

- Subjects are from our previous work [1]
- Searched "synchronized" keyword inside methods and manually checked locking objects to categorize them into 4 sets as we have talked previously
- We used 203 locks in the evaluation

[1]Ziyi Lin, Darko Marinov, Hao Zhong, Yuting Chen and Jianjun Zhao. *JaConTeBe: A Benchmark Suite of Real-World Java Concurrency Bugs.* In Proc. ASE, 2015

Evaluation

- **RQ1:** How effective is LockPeeker in revealing locks in Java API methods ?
- We manually compare the detected locks with the locks in API methods
  - The lock is detected
  - Relations among locks are detected
  - Branching specifications are detected

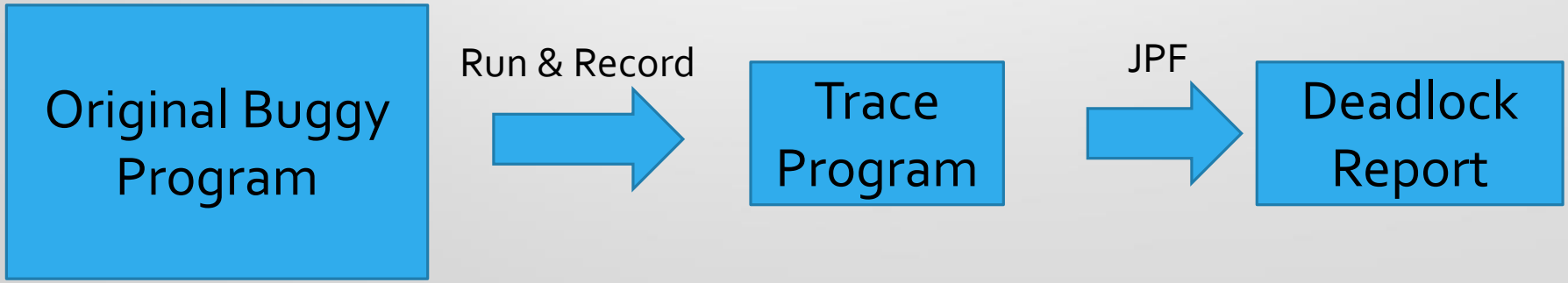
  } Loosely detected

  } Strictly detected
- RC1=strictly_detected/total, RC2=loosely_detected/total

Projects	Parameter (S_{mp})			Field (S_f)			Receiver (S_i)			Total		
	RC1	RC2	#Locks	RC1	RC2	#Locks	RC1	RC2	#Locks	RC1	RC2	#Locks
DBCP	N/A	N/A	0	25%	25%	4	100%	100%	2	50%	50%	6
Derby	78.60%	78.60%	14	84.60%	84.60%	13	80%	80%	10	81.10%	81.10%	37
FtpServer	25%	25%	4	50%	50%	4	N/A	N/A	0	37.50%	37.50%	8
Groovy	0%	0%	3	90%	90%	10	70%	70%	10	69.60%	69.60%	23
HsqlDB	75%	75%	12	80%	100%	10	80%	100%	10	78.10%	90.60%	32
Log4j	0%	0%	1	83.30%	83.30%	12	0%	100%	2	66.70%	80%	15
Lucene	100%	100%	1	75%	83.30%	12	18.20%	54.50%	11	50%	70.80%	24
Pool	N/A	N/A	0	N/A	N/A	0	80%	90%	10	80%	90%	10
Tomcat	71.40%	71.40%	14	63.60%	63.60%	11	80%	90%	10	71.40%	74.30%	35
Xalan	N/A	N/A	0	40%	40%	10	100%	100%	3	53.80%	53.80%	13
Total	**65.30%**	**65.30%**	**49**	**70.90%**	**74.40%**	**86**	**67.60%**	**82.40%**	**68**	**68.50%**	**74.90%**	**203**

Evaluation

- **RQ2:** What kinds of deadlocks can be detected, if our detected latent locks are integrated?
- Evaluation subject: 4 real-world deadlocks caused by latent locks
- Evaluation tool: deadlock detection tool CheckMate[2]
- CheckMate does not detect any deadlock alone, but detects all together with LockPeeker



[2] Pallavi Joshi, Mayur Naik, Koushik Sen, David Gay: An effective dynamic analysis for detecting generalized deadlocks. FSE 2010

Evaluation

```
public class SimpleBirt287102 {
  private SimpleClassLoader loader;
  private Object obj;

class Thread1 extends Thread {
  public void run() {
    synchronized(loader){
      Class.forName("java.lang.Object", true, loader);
   }
  }}

  class Thread2 extends Thread {
    public void run() {
      synchronized (obj) {
        synchronized(loader){
          Class.forName("java.lang.Object", true, loader);
        }
  }}}

class SimpleClassLoader extends ClassLoader {
  public Class<?> loadClass(String name) {
    synchronized (obj) {...}
    return super.loadClass(name);
}}}
private static native Class<?> forName0(String name,
boolean initialize, ClassLoader loader, Class<?> caller);
```

Run and Record

```
public class TraceProgram {
   static Object obj1 = new Object();
   static Object obj2 = new Object();
   static Thread t1 = new Thread(){
      public void run() {
        synchronized (obj2) {
           synchronized (obj1) {}
        }
      }};
   static Thread t2 = new Thread() {
       public void run() {
          synchronized (obj1) {
             synchronized (obj2) {}
          } }};
   public static void main(String[] args) {
       t1.start();
       t2.start();
}}
```

JPF

Deadlock Report

Conclusion & Future Work

- Locks can be latent in API methods, which are not rare, but difficult to be detected
- LockPeeker dynamically checks latent locks from close-sourced Java API methods, even native methods which are implemented by other languages
- There are still many works need to take care in the future to detect latent locks in complicated methods:
  - Call sequences and inputs
  - Repeated locks
  - Complicated conditions
  - Missing the locks on objects in S_e