Soot

Java程序分析与变换框架

陈逸凡 2020年11月4日

Soot是什么

Soot

Soot – A framework for analyzing and transforming Java and Android applications

What is Soot?

Originally, Soot started off as a Java optimization framework. By now, researchers and practitioners from around the world use Soot to analyze, instrument, optimize and visualize Java and Android applications.

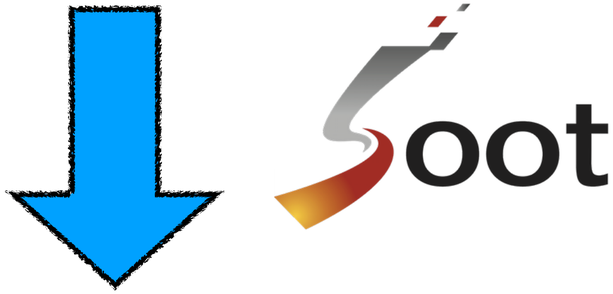
主页：https://soot-oss.github.io/soot/

Soot的开发历程

- Started in 1996-97 with the development of coffi by Clark Verbrugge and some first prototypes of Jimple IR by Clark and Raja Vallée-Rai.
- Originally developed by the **Sable Research Group** of McGill University.
- The first publication on Soot appeared at CASCON 1999.
- The current maintenance is driven by **Eric Bodden**'s Software Engineering Group at Heinz Nixdorf Institute of Paderborn University.
- Currently there are a bunch of extensions to Soot, including **Boomerang**, **FlowDroid** and **Soot-Scala.**

Soot的输入和输出

- Input：Java源码/字节码



- Output：程序分析的结果（如活跃变量）/ 程序的中间表示（如Jimple）

为什么要用Soot?

问题1：分析Java源代码的第一步?

- 直接当成字符串？（别笑，真有[1]）
  - 难以知晓代码结构信息
- 转为**中间表示**（IR）！
  - 保留源码信息（与源代码有明确映射关系）
  - 方便机器理解（更加简单，更加结构化）

[1] Code Completion with Statistical Language Models, Veselin Raychev, Martin Vechev, Eran Yahav, PLDI’14

为什么要用Soot?

问题2：使用什么中间表示?

- 直接使用Java bytecode?
  - 😰太贴近机器码（为执行而设计）
  - 😭语句类型～200种（至多有256条指令）
  - 😫基于栈的代码

- 基于栈的代码

```java
for (int i = 2; i < 1000; i++) {
    for (int j = 2; j < i; j++) {
        if (i % j == 0)
            continue outer;
    }
    System.out.println (i);
}
```

```
0:   iconst_2
1:   istore_1
2:   iload_1
3:   sipush  1000
6:   if_icmpge       44
9:   iconst_2
10:  istore_2
11:  iload_2
12:  iload_1
13:  if_icmpge       31
16:  iload_1
17:  iload_2
18:  irem
19:  ifne    25
22:  goto    38
25:  iinc    2, 1
28:  goto    11
31:  getstatic       #84; // Field java/lang/System.out:Ljava/io/PrintStream;
34:  iload_1
35:  invokevirtual   #85; // Method java/io/PrintStream.println:(I)V
38:  iinc    1, 1
41:  goto    2
44:  return
```

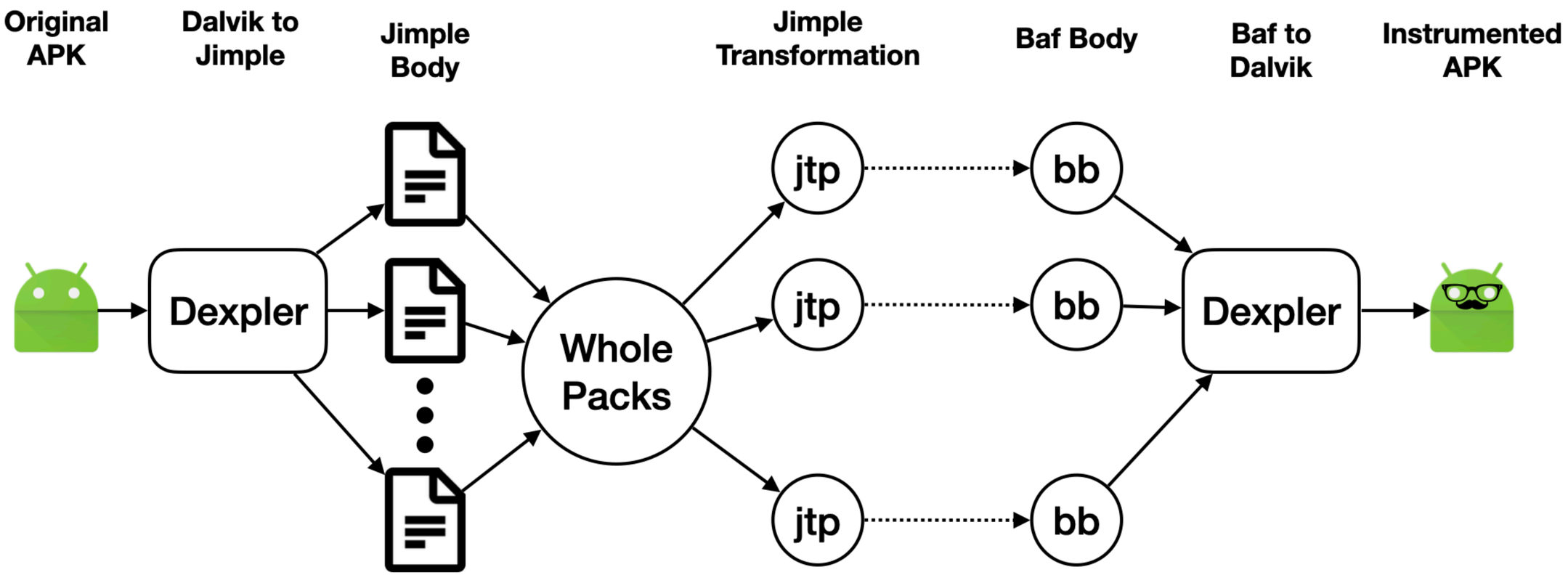
为什么要用Soot?

Soot的中间表示格式

- Soot有四种适合不同程序分析的中间表示：
  - Baf: a streamlined representation of bytecode which is simple to manipulate.
  - **Jimple: a typed 3-address intermediate representation suitable for optimization.**
  - Shimple: an SSA variation of Jimple.
  - Grimp: an aggregated version of Jimple suitable for decompilation and code inspection.

Jimple

=Java+Simple



.

Jimple

=Java+Simple

- 😄Jimple只有15种指令

- Core statements:
  `NopStmt`
  `DefinitionStmt: IdentityStmt, AssignStmt`
- Intraprocedural control-flow:
  `IfStmt`
  `GotoStmt`
  `TableSwitchStmt,LookupSwitchStmt`
- Interprocedural control-flow:
  `InvokeStmt`
  `ReturnStmt, ReturnVoidStmt`

- `ThrowStmt`
  throws an exception
- `RetStmt`
  not used; returns from a JSR
- `MonitorStmt: EnterMonitorStmt, ExitMonitorStmt`
  mutual exclusion

上机实践1

从Java到Jimple

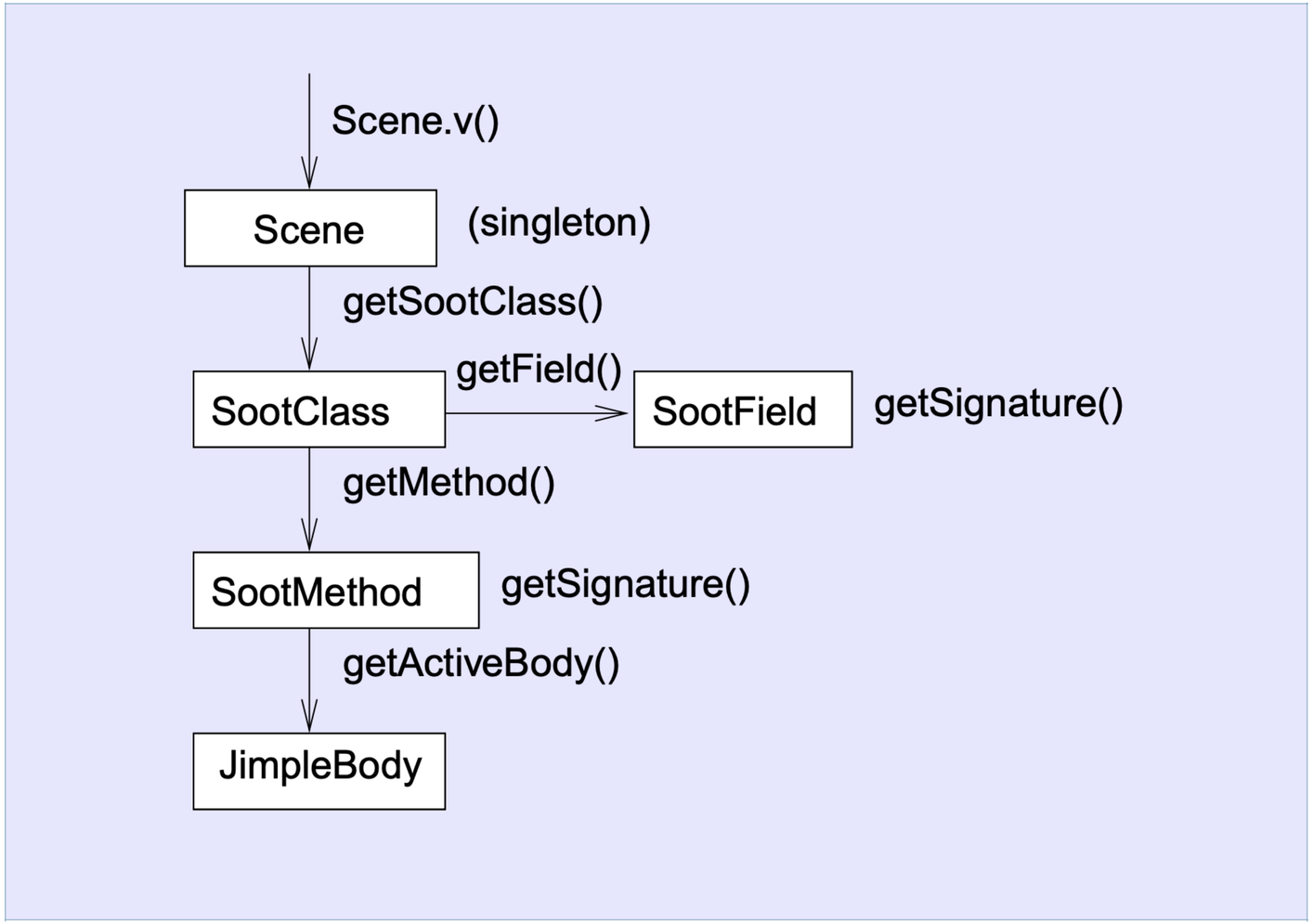
- Soot as a command-line tool:

```
java -cp <path/to/soot> soot.Main -cp <path/to/rt.jar>:. -f J HelloWorld
```

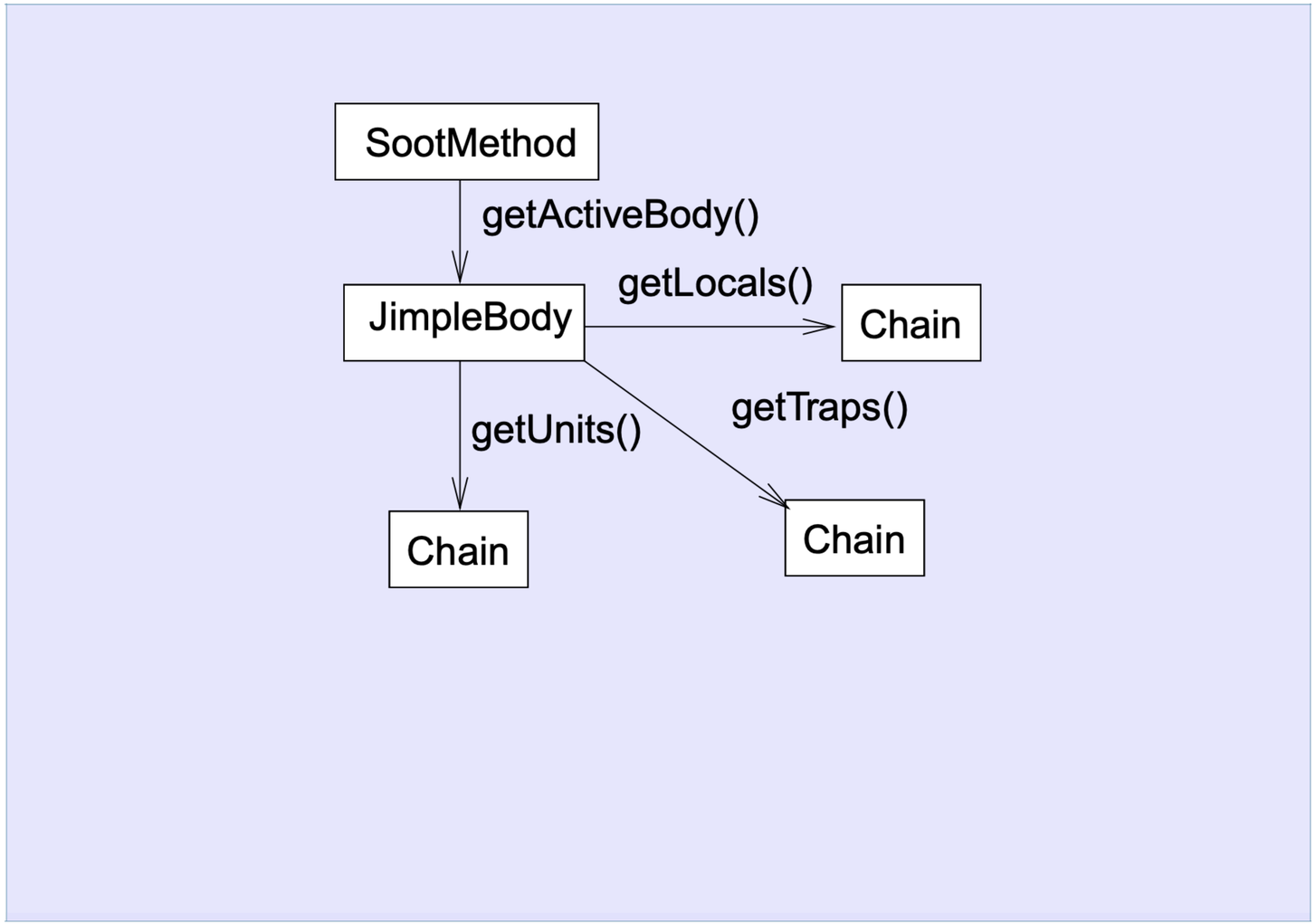
- Soot as a library:

```
soot.Main.main(args)
```

Soot的数据结构


Scene.v()
Scene
(singleton)
getSootClass()
SootClass
getField()
SootField
getSignature()
getMethod()
SootMethod
getSignature()
getActiveBody()
JimpleBody

Soot的数据结构


SootMethod
getActiveBody()
JimpleBody
getLocals()
Chain
getUnits()
getTraps()
Chain
Chain

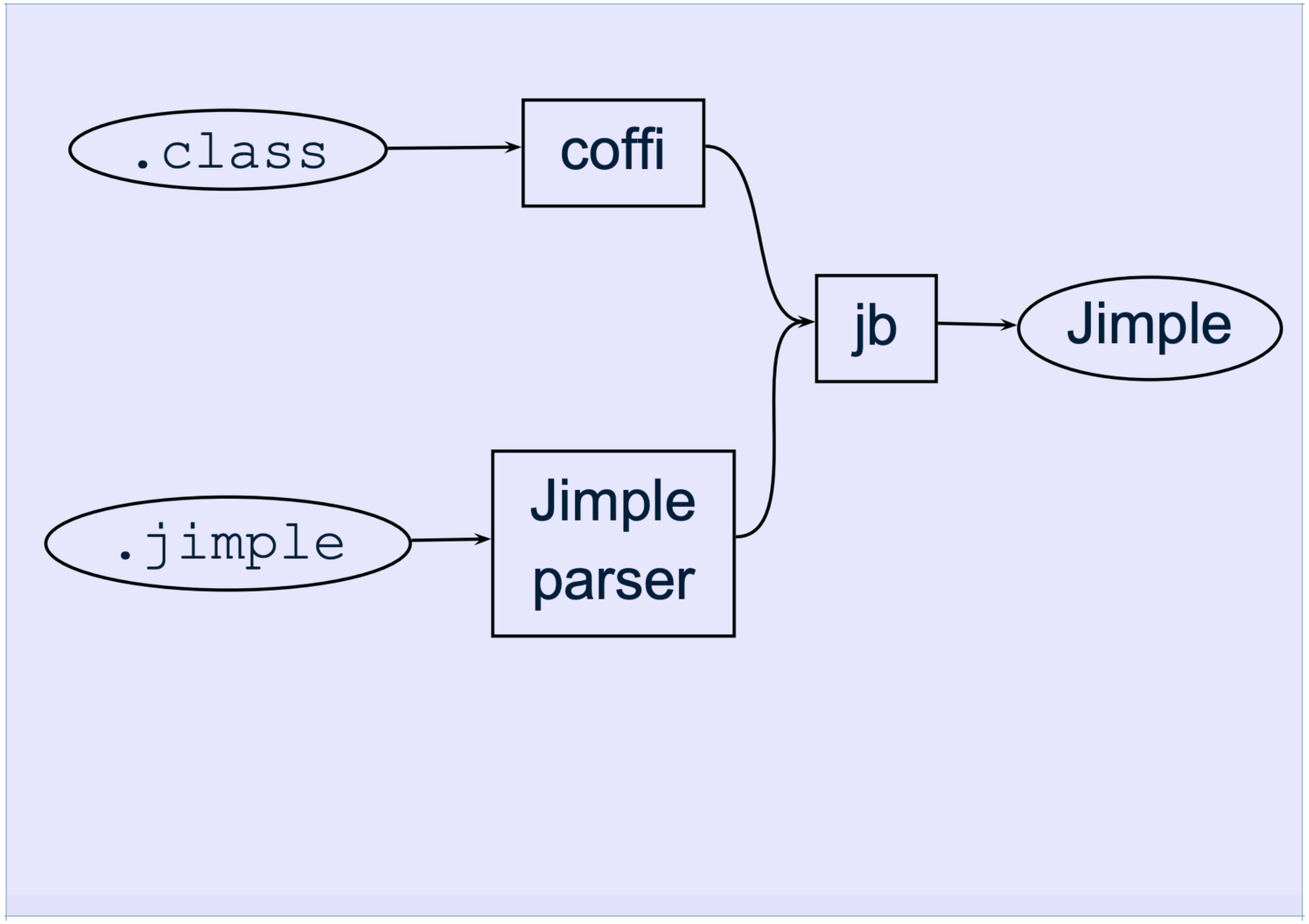
上机实践2

遍历程序结构

- 阅读并运行Traverse.java
- 了解每条语句的效果

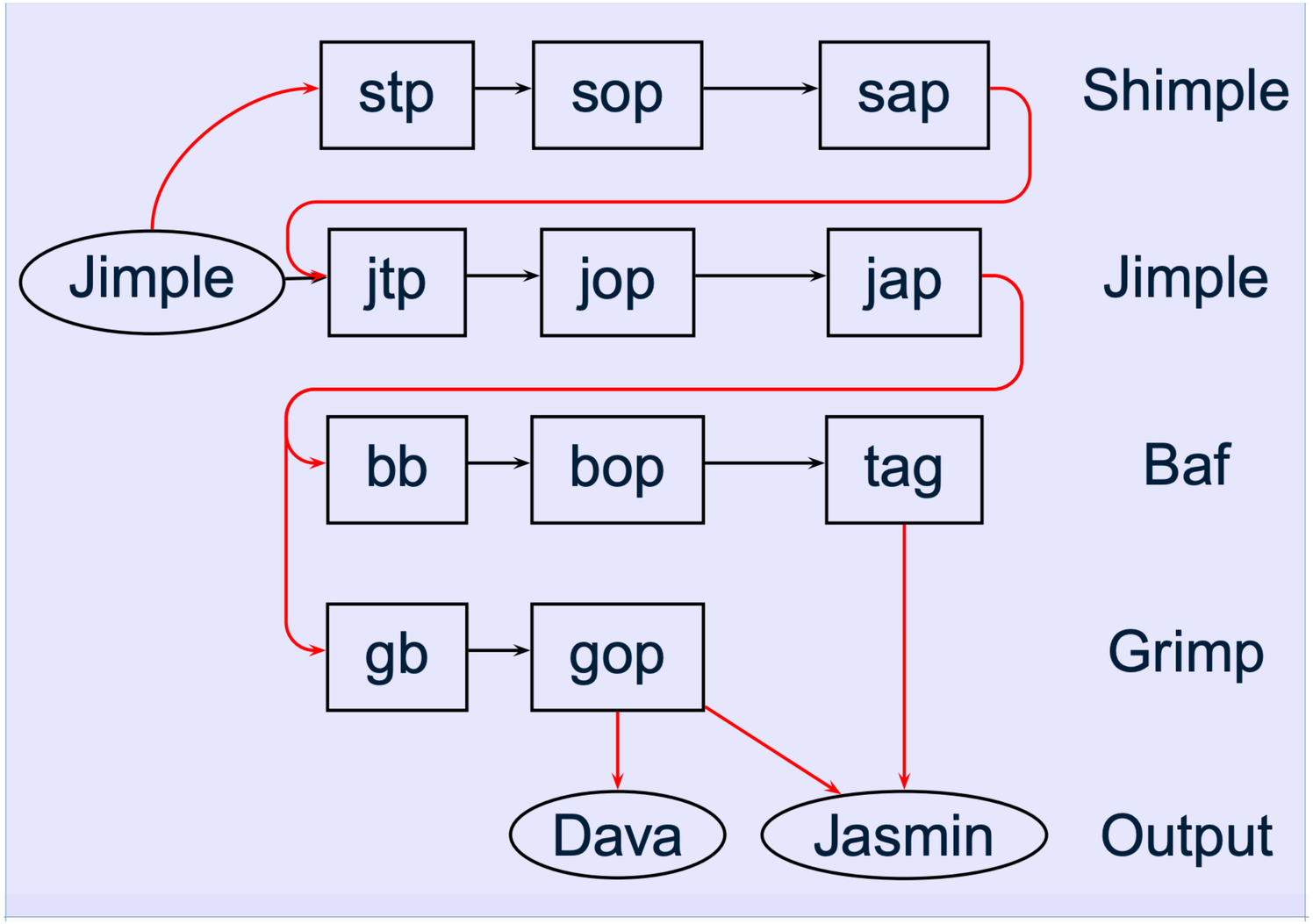
Soot的执行流程

Pack & Phase



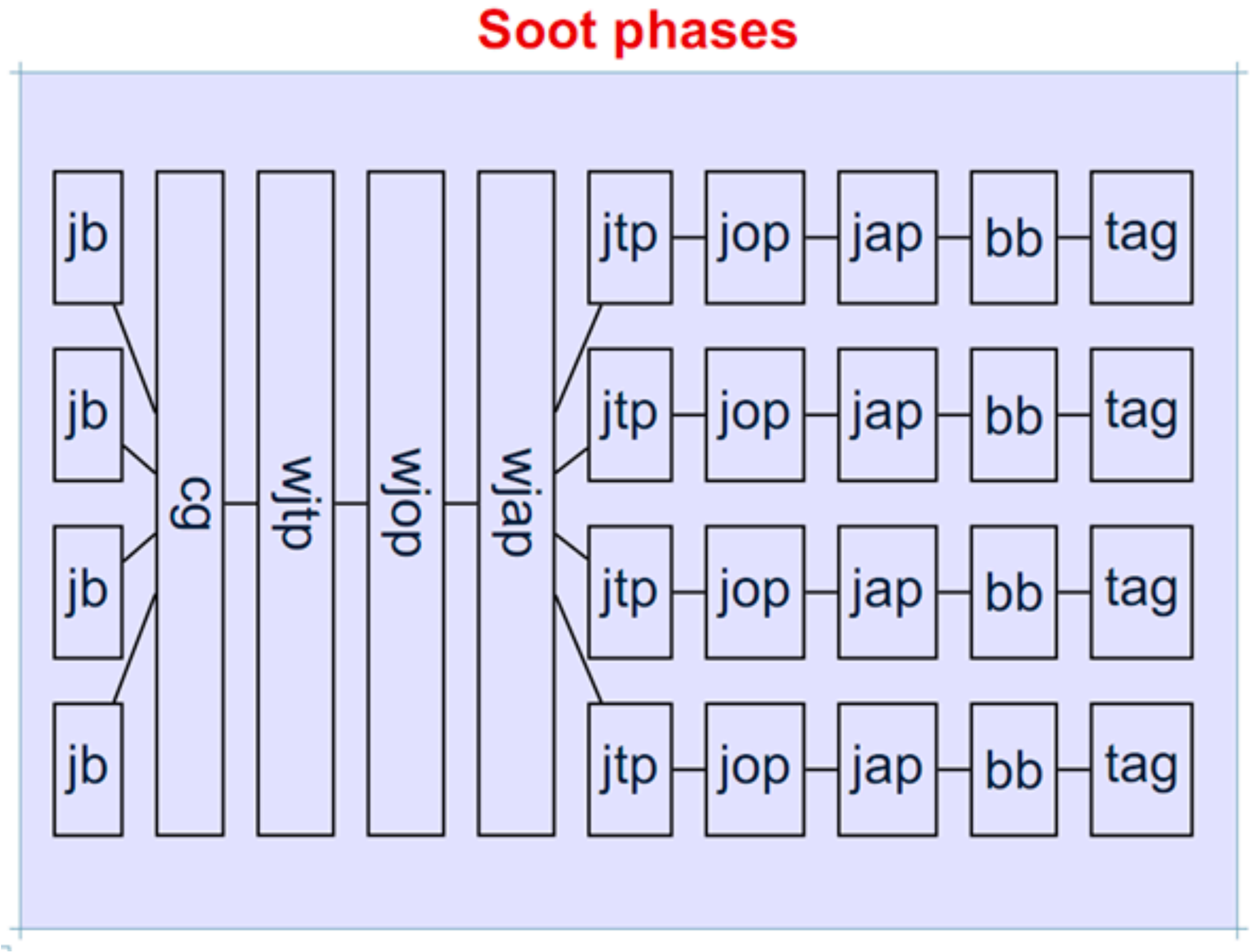
Soot的执行流程

Pack & Phase



Soot的执行流程

Pack & Phase



上机练习3

插入phase

- 尝试运行Fetch.java
- 尝试修改以获取更多信息

参考资料

- A Survivor's Guide to Java Program Analysis with Soot. Arni Einarsson and Janus Dam Nielsen. https://www.brics.dk/SootGuide/
- Analyzing Java Programs with Soot. Bruno Dufour. http://www.iro.umontreal.ca/~dufour/cours/ift6315/docs/soot-tutorial.pdf
- Home - soot-oss/soot Wiki - GitHub. https://github.com/soot-oss/soot/wiki
- noidsirius/SootTutorial. https://github.com/noidsirius/SootTutorial

Q & A