THE UNIVERSITY
OF BRITISH COLUMBIA

ReSeSS
The Reliable, Secure, and Sustainable Software Lab

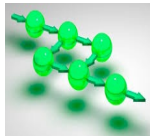
# *Dynamic Slicing with Trace-Based Alias Analysis*

Khaled Ahmed, Ph.D. candidate, UBC

# *Reliable, Secure, and Sustainable Software Lab (ReSeSS)*

**Mobile software**
(malware detection, efficient testing, energy-efficiency)

**Analysis for microservice-based cloud software**
(quality, security, performance, efficient deployment)



**Compositional software development**
(integration and upgrade management, automated repairs)

**Trustworthy AI**
(security, reliability, quality assurance, fairness, ethics)

# *Reliable, Secure, and Sustainable Software Lab (ReSeSS)*

**Mobile software**
(malware detection, efficient testing, energy-efficiency)



**Analysis for microservice-based cloud software**
(quality, security, performance, efficient deployment)

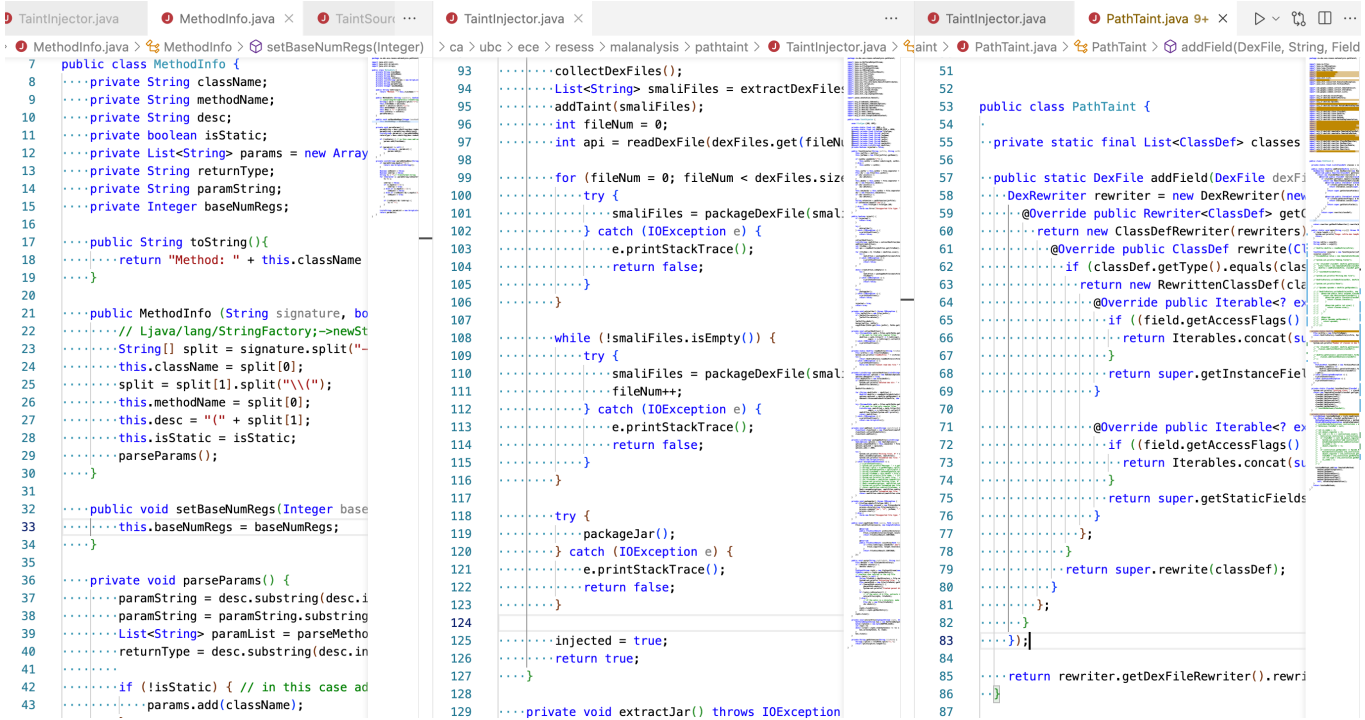


**Compositional software development**
(integration and upgrade management, automated repairs)

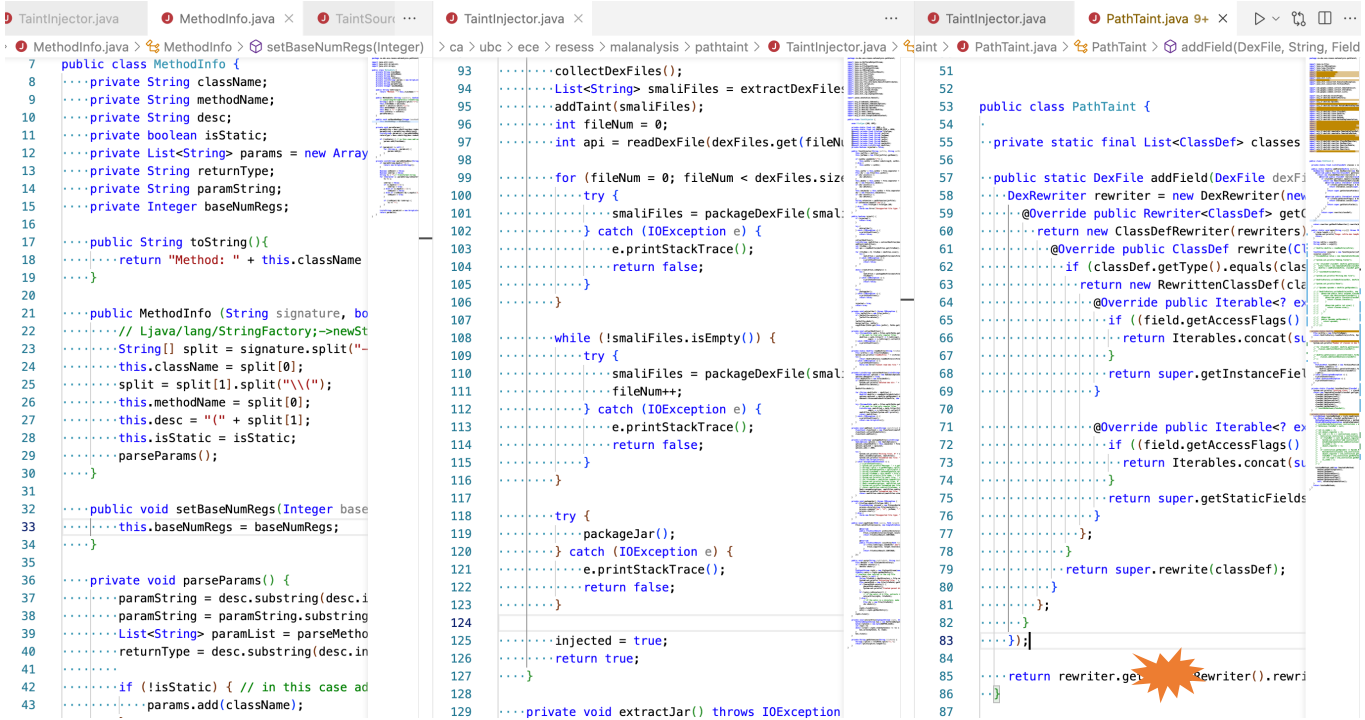


**Trustworthy AI**
(security, reliability, quality assurance, fairness, ethics)

# Debugging …

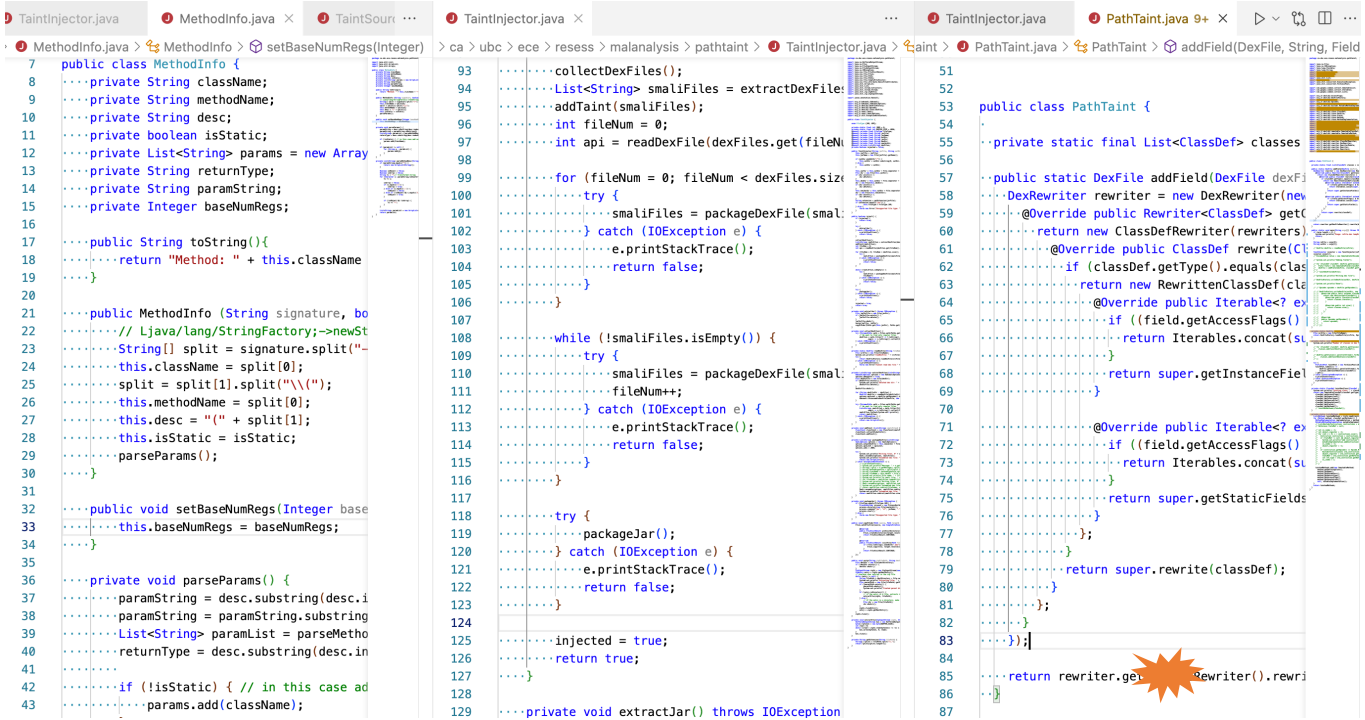

TaintInjector.java
MethodInfo.java
TaintSourc
MethodInfo.java > MethodInfo > setBaseNumRegs(Integer)
public class MethodInfo {
private String className;
private String methodName;
private String desc;
private boolean isStatic;
private List<String> params = new Array
private String returnType;
private String paramString;
private Integer baseNumRegs;
public String toString(){
return "Method: " + this.className
}
public MethodInfo (String signature, bo
// Ljava/lang/StringFactory;->newSt
String[] split = signature.split("-
this.className = split[0];
split = split[1].split("\\(");
this.methodName = split[0];
this.desc = "(" + split[1];
this.isStatic = isStatic;
parseParams();
}
public void setBaseNumRegs(Integer base
this.baseNumRegs = baseNumRegs;
}
private void parseParams() {
paramString = desc.substring(desc.i
paramString = paramString.substring
List<String> paramList = parseMetho
returnType = desc.substring(desc.in
if (!isStatic) { // in this case ad
params.add(className);
TaintInjector.java
collectDexFiles();
List<String> smaliFiles = extractDexFiles
addTaint(smaliFiles);
int fileNum = 0;
int api = readDexFile(dexFiles.get(fileNu
for (fileNum = 0; fileNum < dexFiles.size
try {
smaliFiles = packageDexFile(smali
} catch (IOException e) {
e.printStackTrace();
return false;
}
}
while (!smaliFiles.isEmpty()) {
try {
smaliFiles = packageDexFile(smali
fileNum++;
} catch (IOException e) {
e.printStackTrace();
return false;
}
}
try {
packageJar();
} catch (IOException e) {
e.printStackTrace();
return false;
}
injected = true;
return true;
}
private void extractJar() throws IOException
TaintInjector.java
PathTaint.java 9+
ca > ubc > ece > resess > malanalysis > pathtaint > TaintInjector.java > aint > PathTaint.java > PathTaint > addField(DexFile, String, Field
public class PathTaint {
private static final List<ClassDef> classes
public static DexFile addField(DexFile dexFi
DexRewriter rewriter = new DexRewriter(new
@Override public Rewriter<ClassDef> getC
return new ClassDefRewriter(rewriters)
@Override public ClassDef rewrite(Cl
if (classDef.getType().equals(clas
return new RewrittenClassDef(cla
@Override public Iterable<? ex
if ((field.getAccessFlags()
return Iterables.concat(su
}
return super.getInstanceFiel
}
@Override public Iterable<? ex
if ((field.getAccessFlags()
return Iterables.concat(su
}
return super.getStaticFields
}
};
}
return super.rewrite(classDef);
}
};
}});
return rewriter.getDexFileRewriter().rewri
}

# Debugging …




What caused the crash?

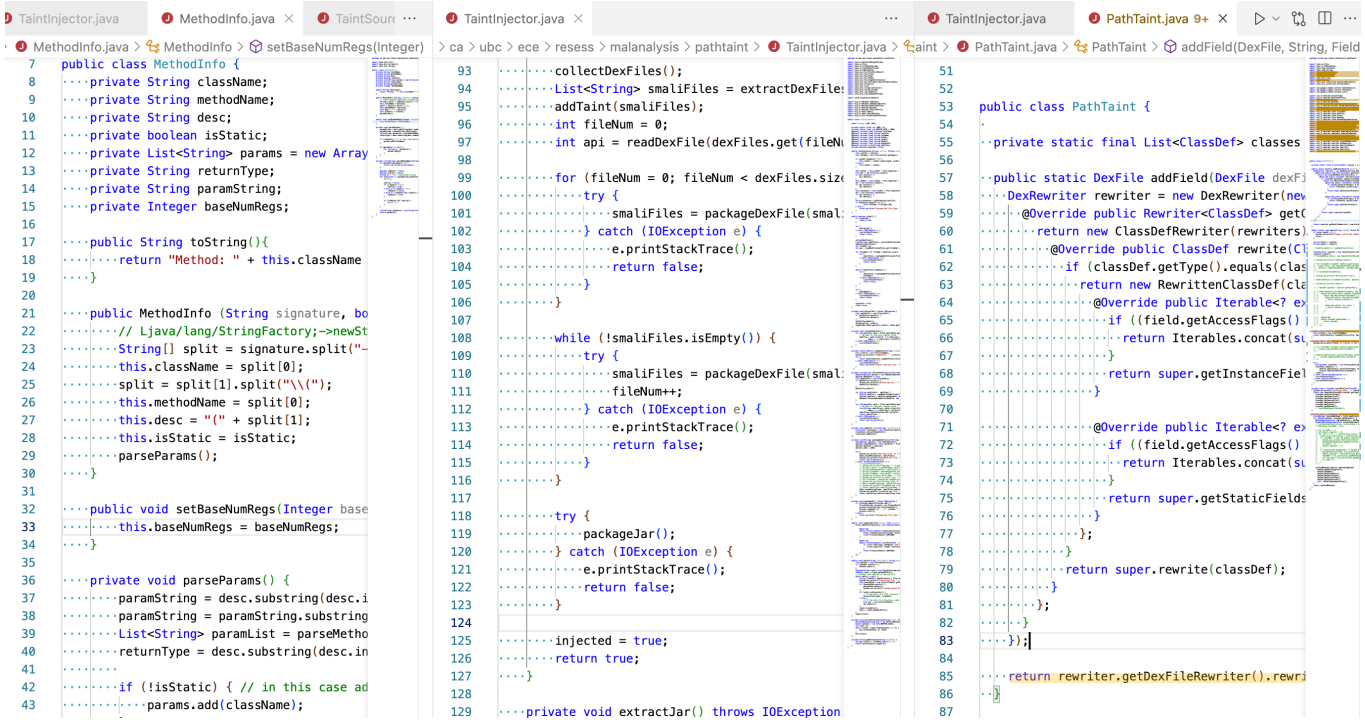
# Debugging …



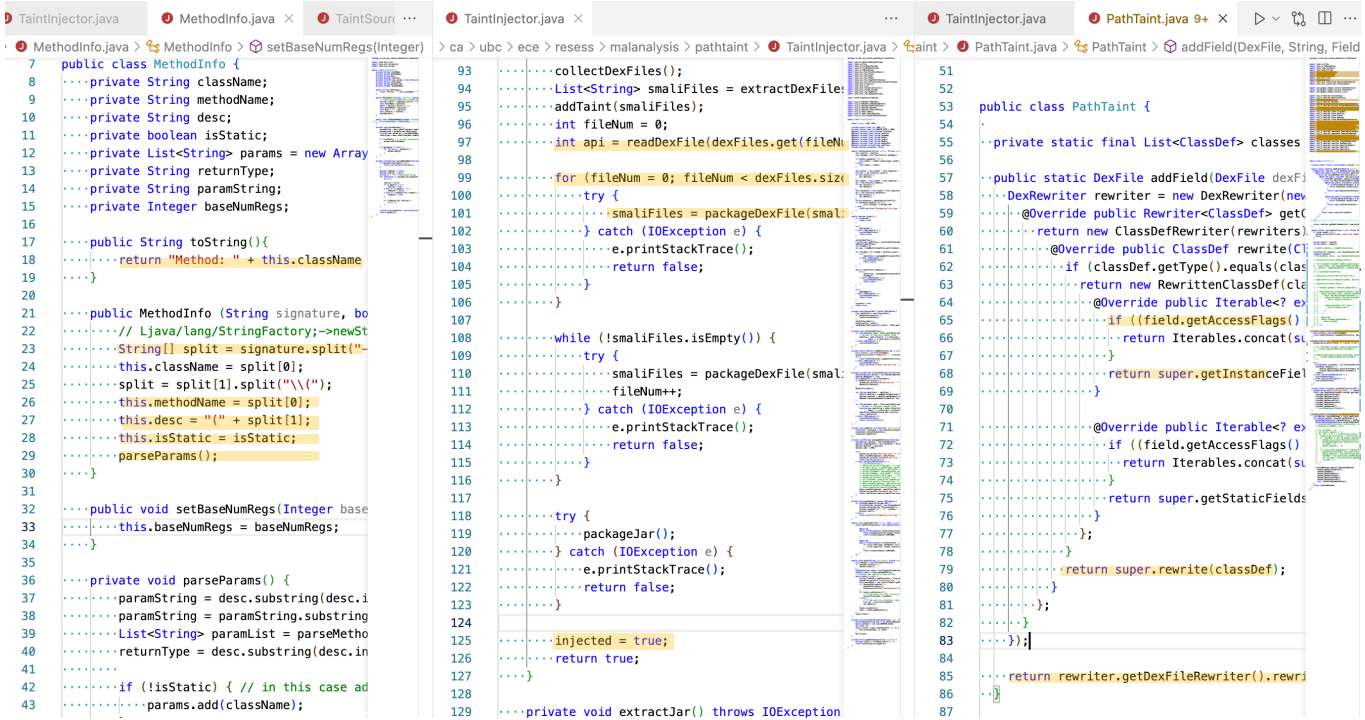

What caused the crash?

Do we need to read the full code?

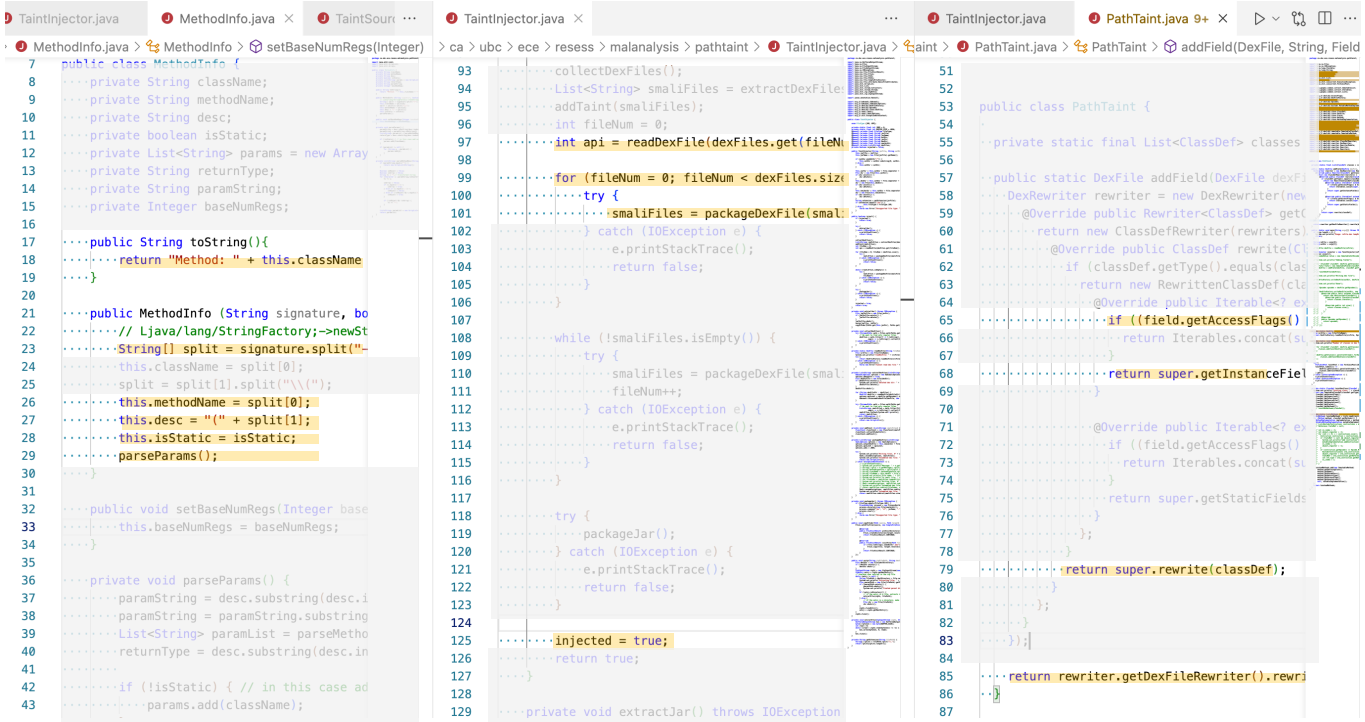
# Making debugging faster



# Making debugging faster

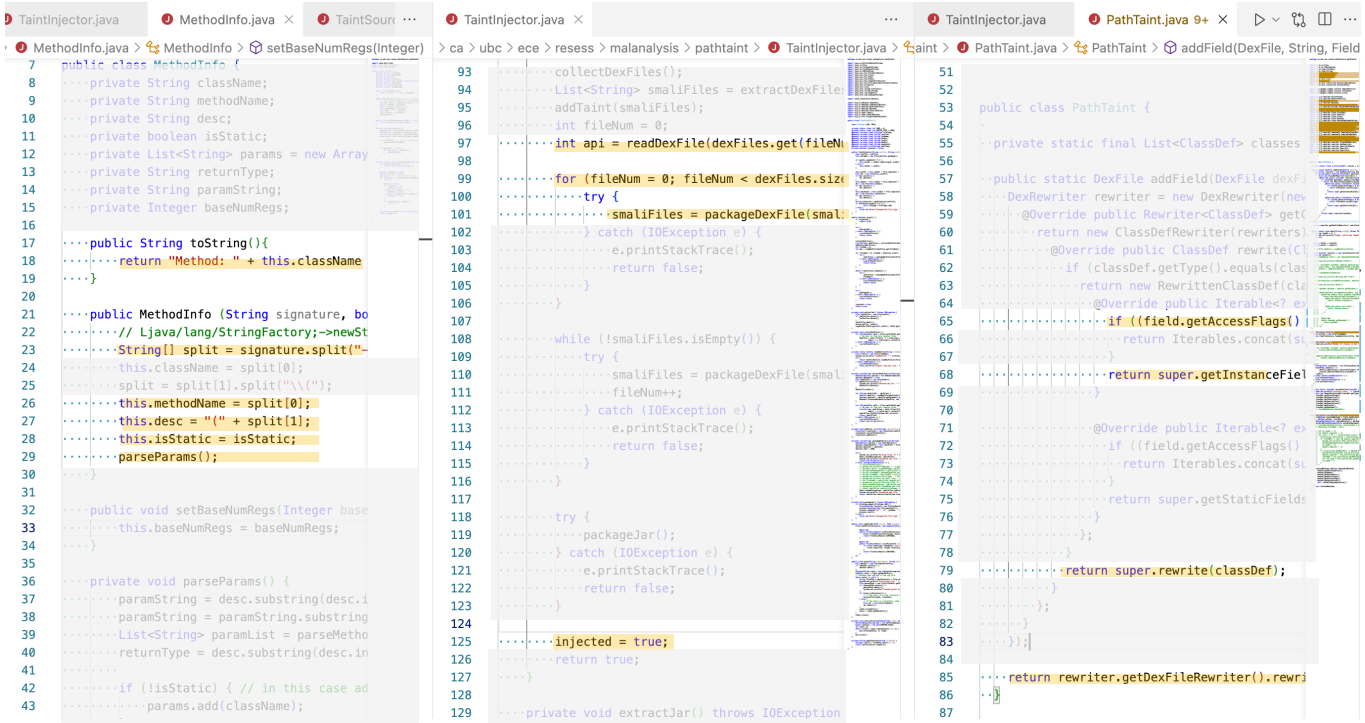


# Making debugging faster




Highlight statements that affect the crash

# Making debugging faster




Highlight statements that affect the crash

Program slicing

# Program slicing

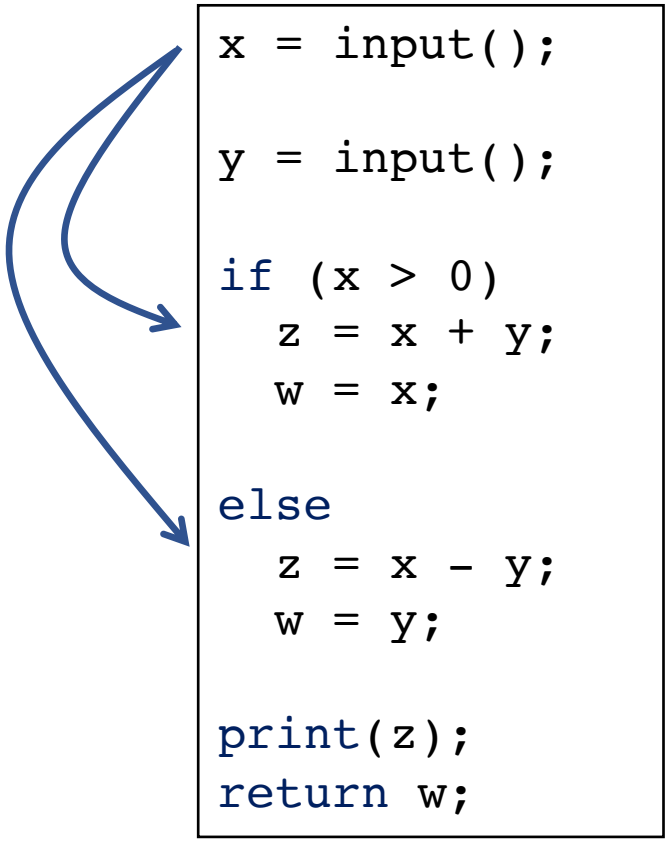
```
x = input();

y = input();

if (x > 0)
  z = x + y;
  w = x;

else
  z = x - y;
  w = y;

print(z);
return w;
```

# Program slicing

```
x = input();

y = input();

if (x > 0)
   z = x + y;
   w = x;

else
   z = x - y;
   w = y;

print(z);
return w;
```

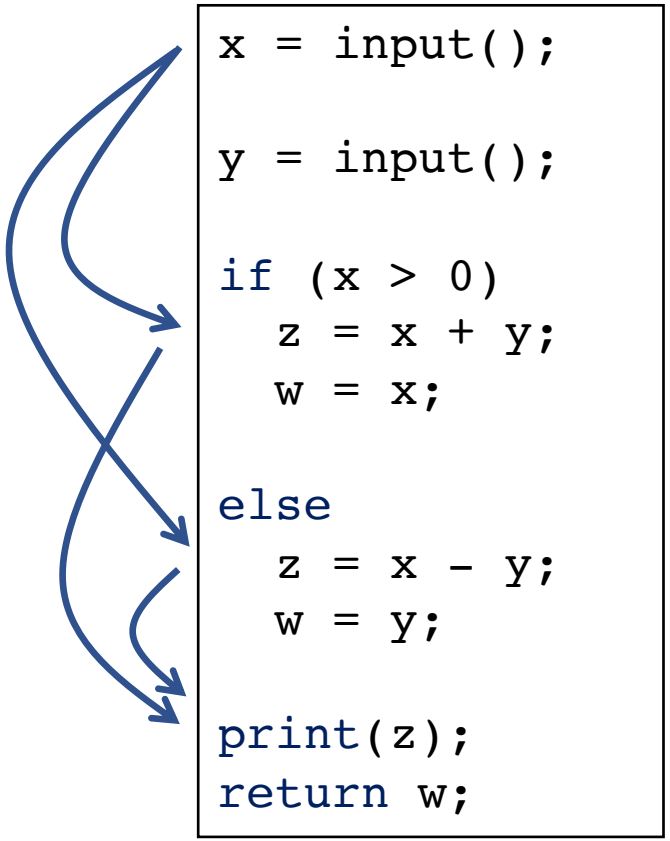
Dataflow 

# Program slicing



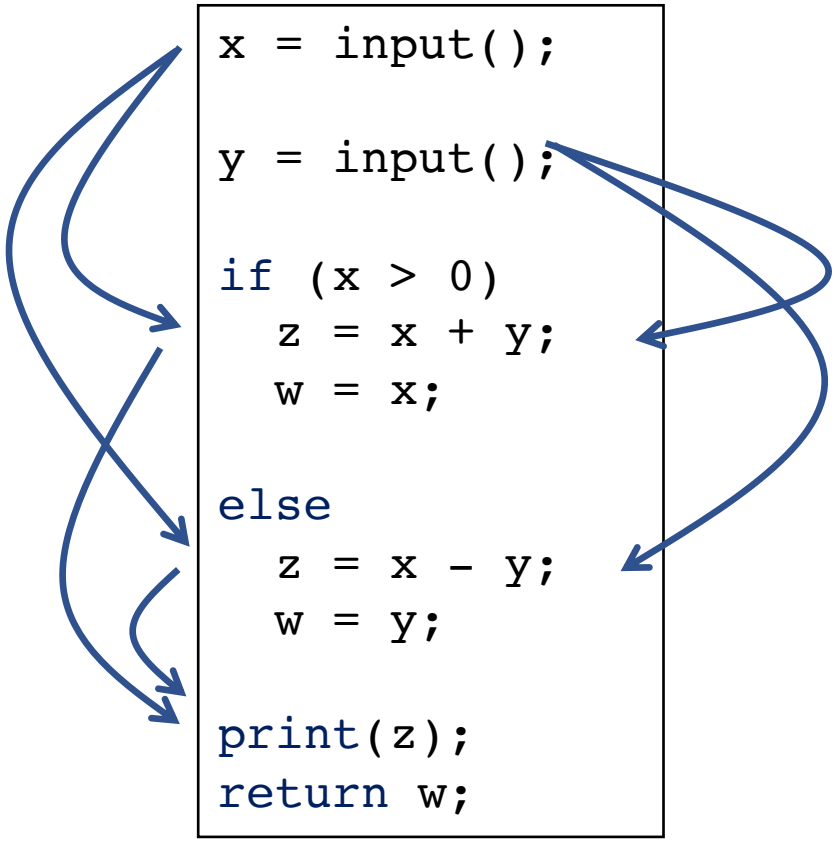

Dataflow 

# Program slicing



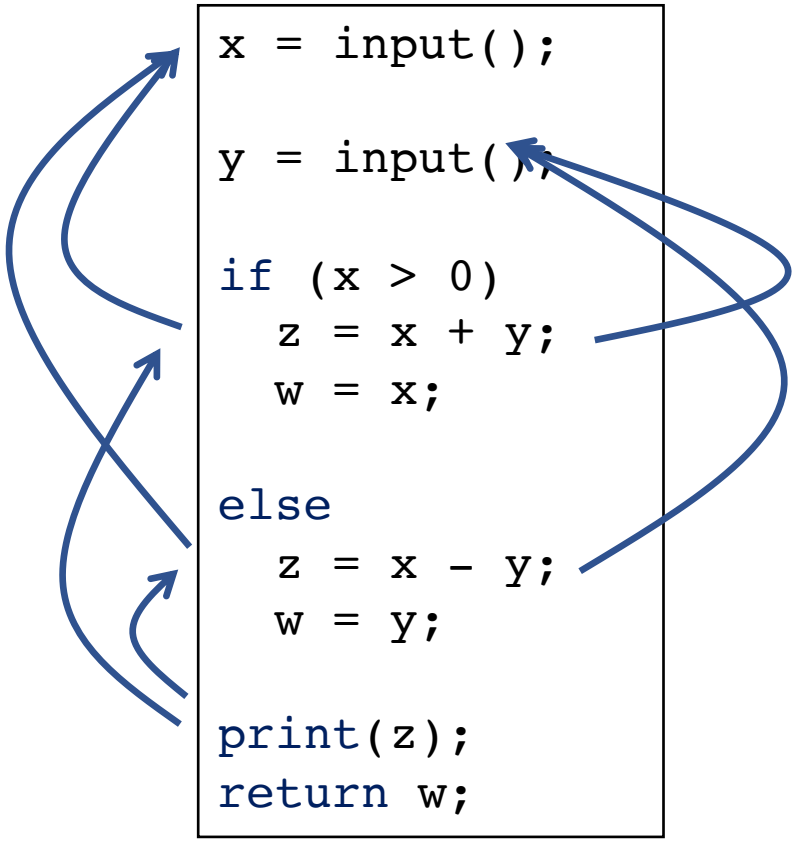

Dataflow →

# Program slicing



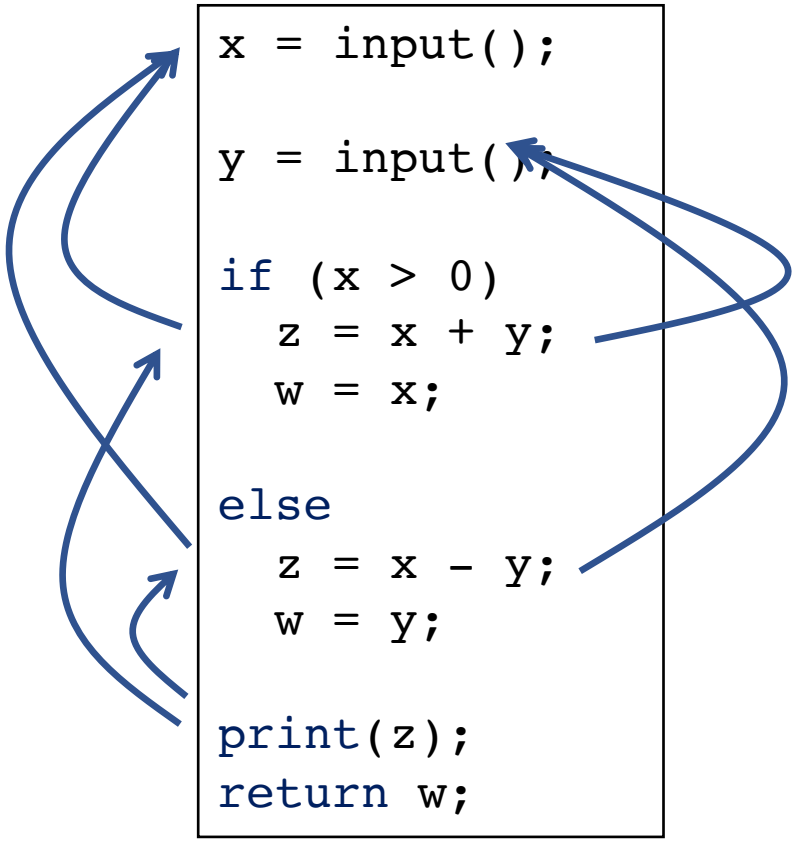

Dataflow →

# Program slicing




Dataflow dependence →

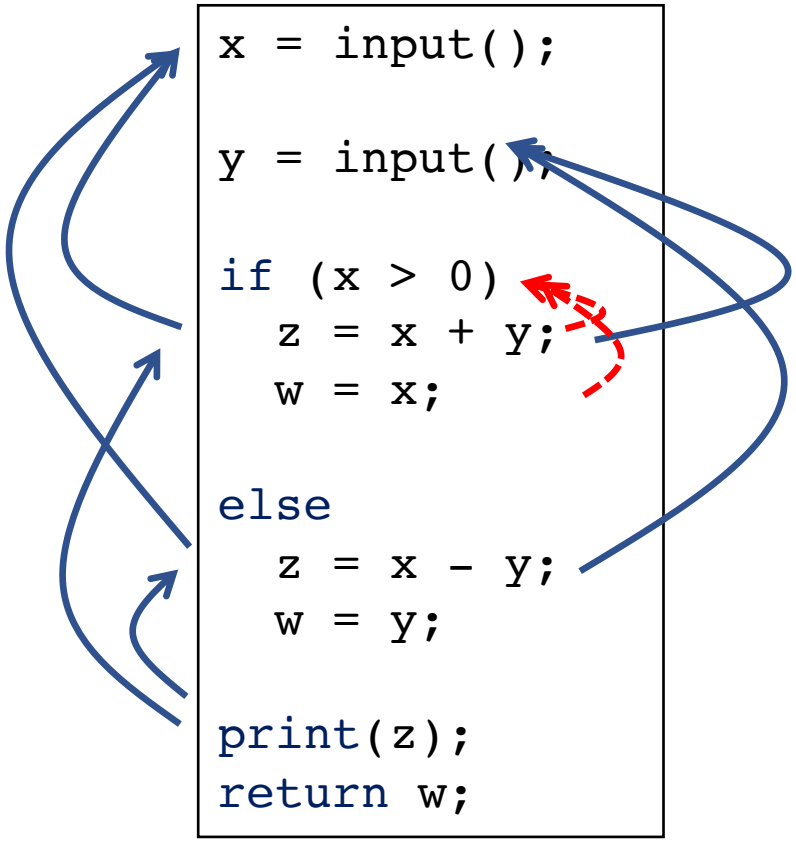
# Program slicing




Dataflow dependence

Control dependence

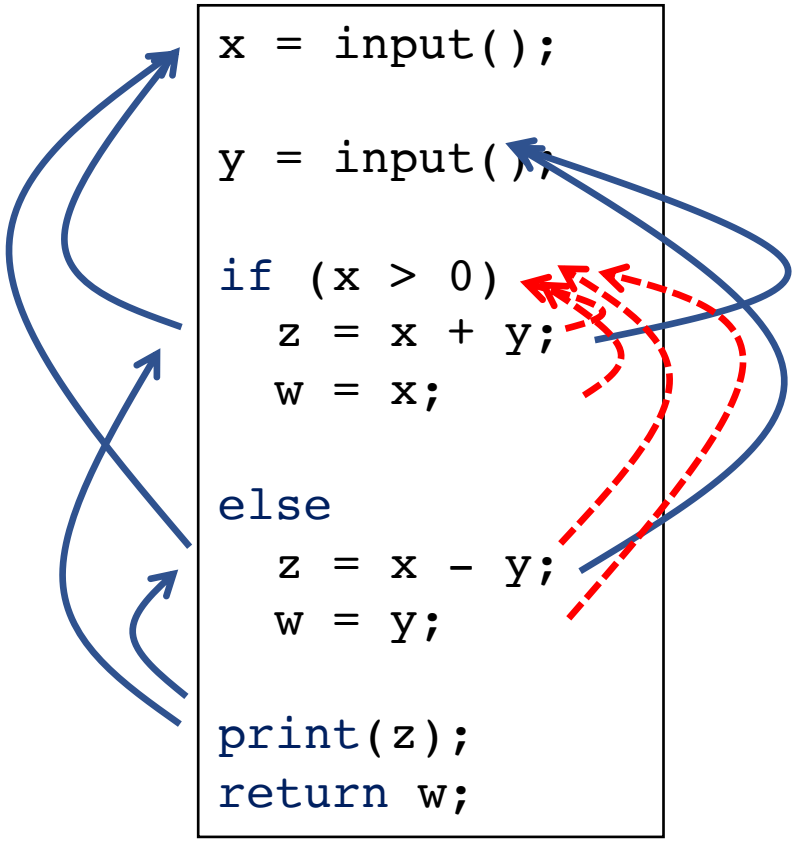
# Program slicing




Dataflow dependence

Control dependence

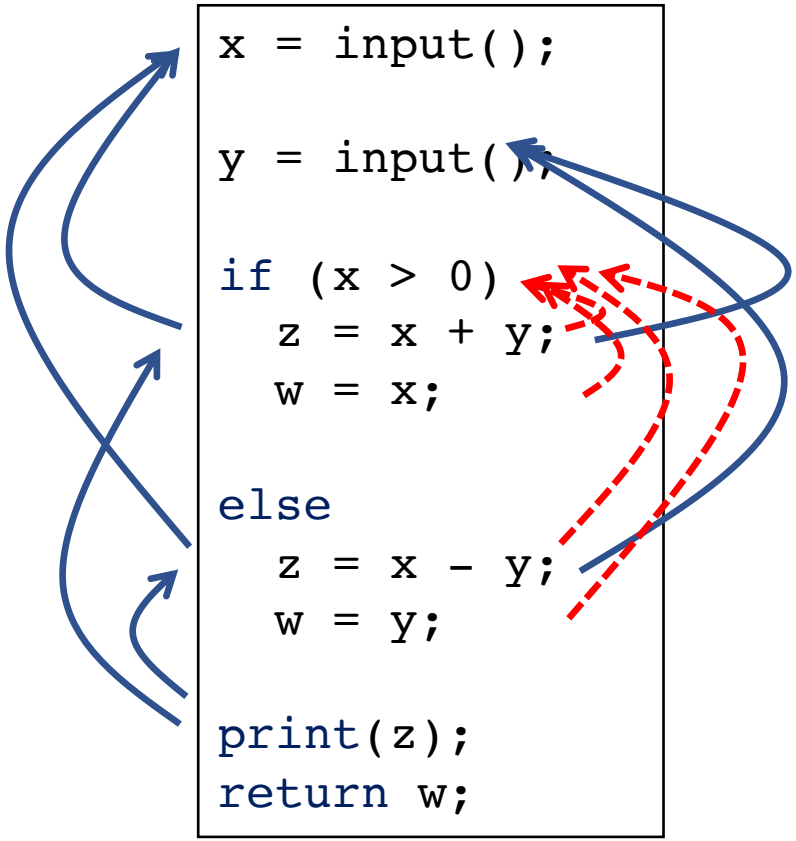
# Program slicing




Dataflow dependence

Control dependence

# Program slicing

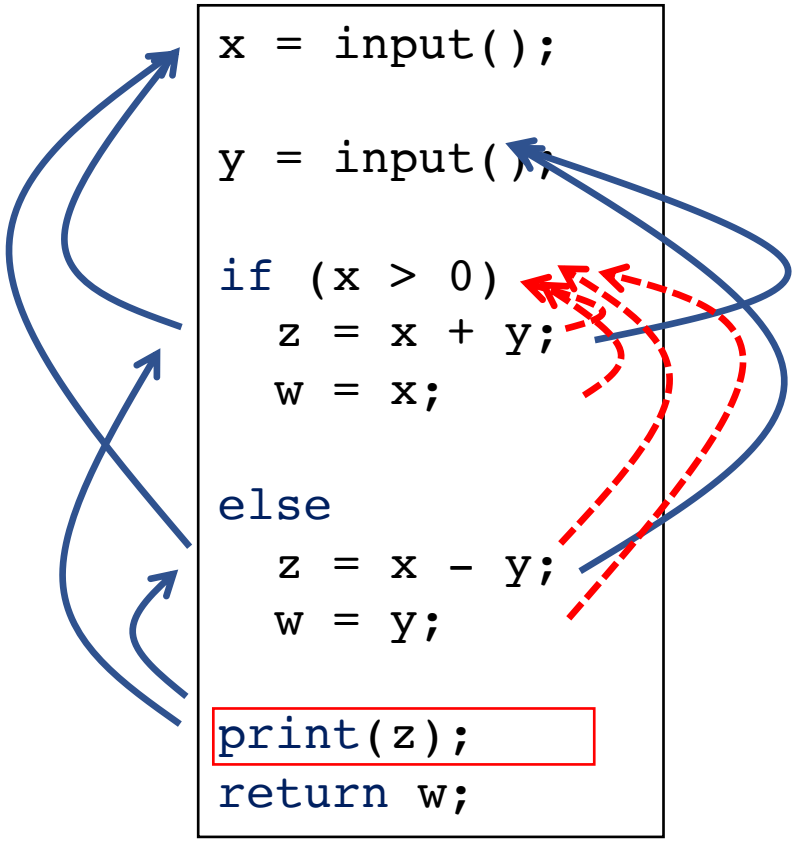



Dataflow dependence

Control dependence

Slicing criterion

# Program slicing

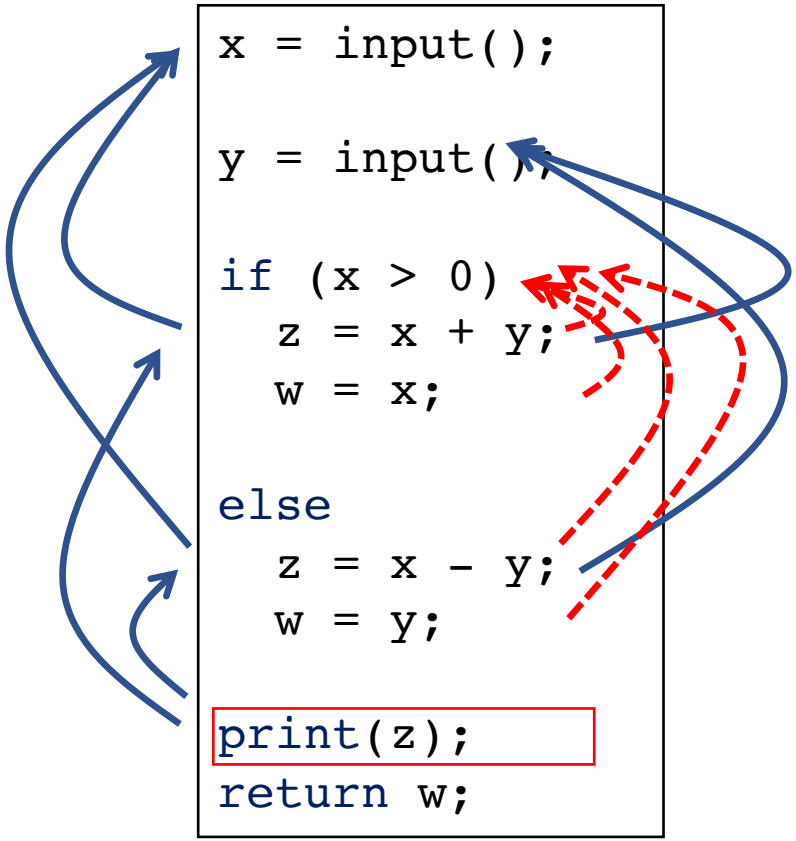



Dataflow dependence

Control dependence

Slicing criterion

# Program slicing



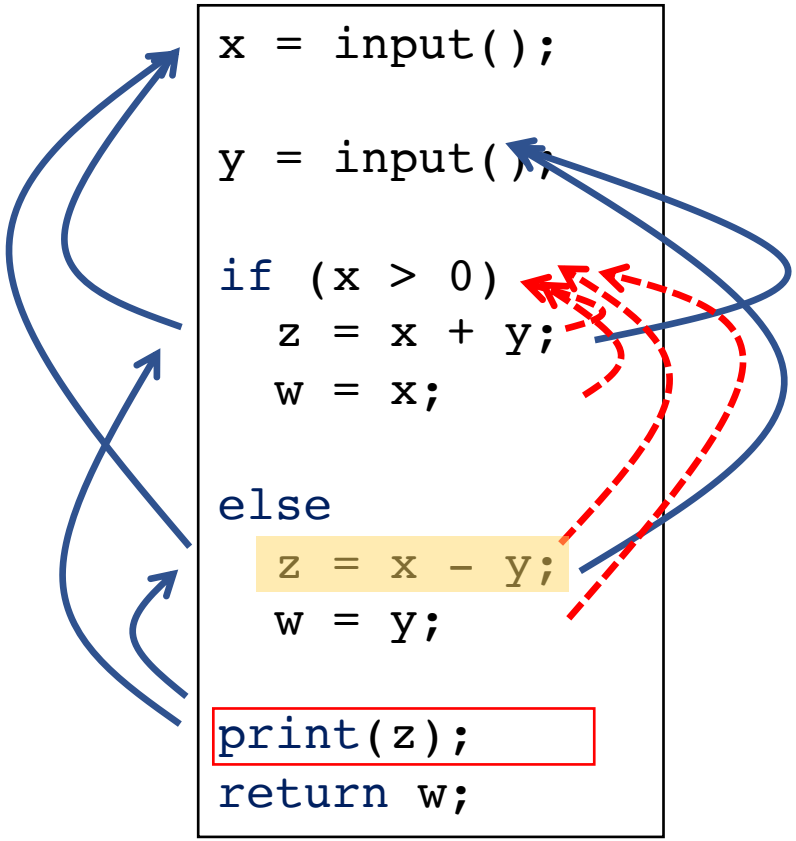

Dataflow dependence

Control dependence

Slicing criterion

Slice

# Program slicing



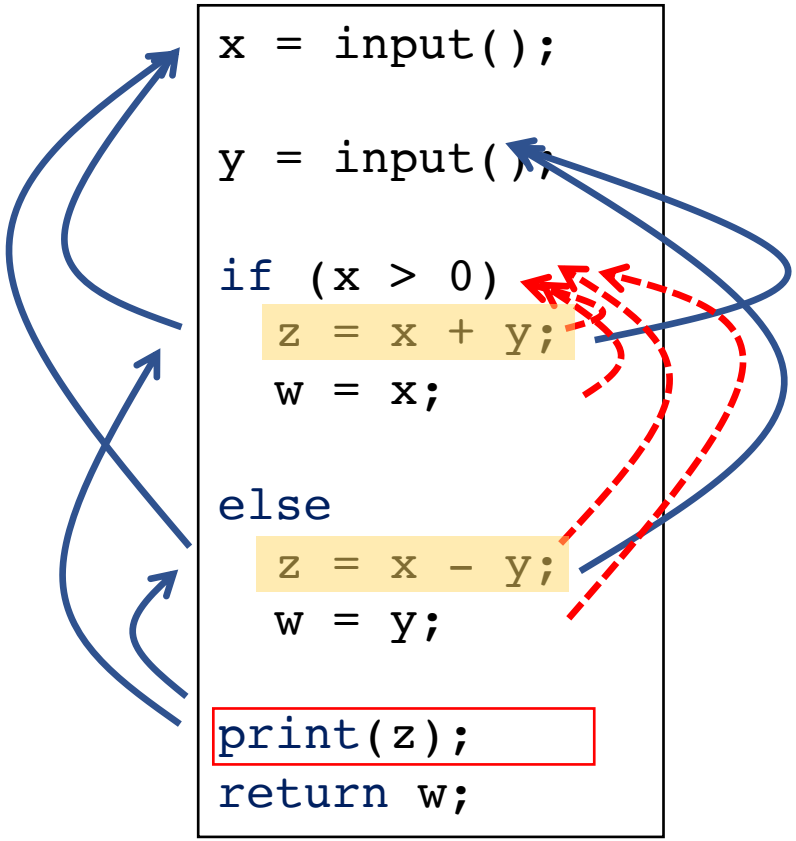

Dataflow dependence

Control dependence

Slicing criterion

Slice

# Program slicing



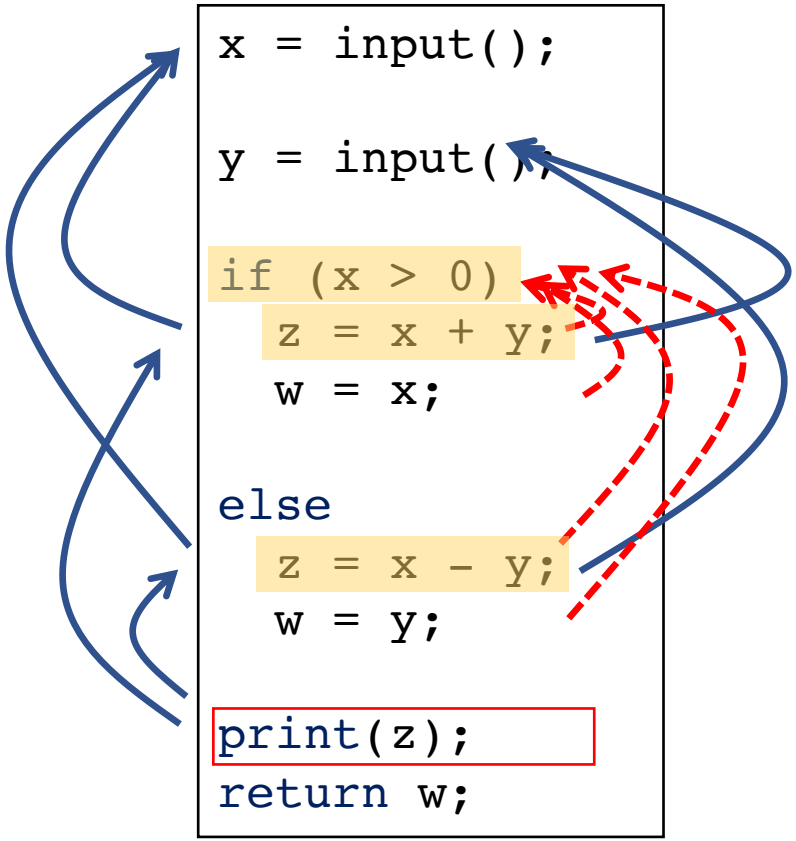

Dataflow dependence

Control dependence

Slicing criterion

Slice

# Program slicing



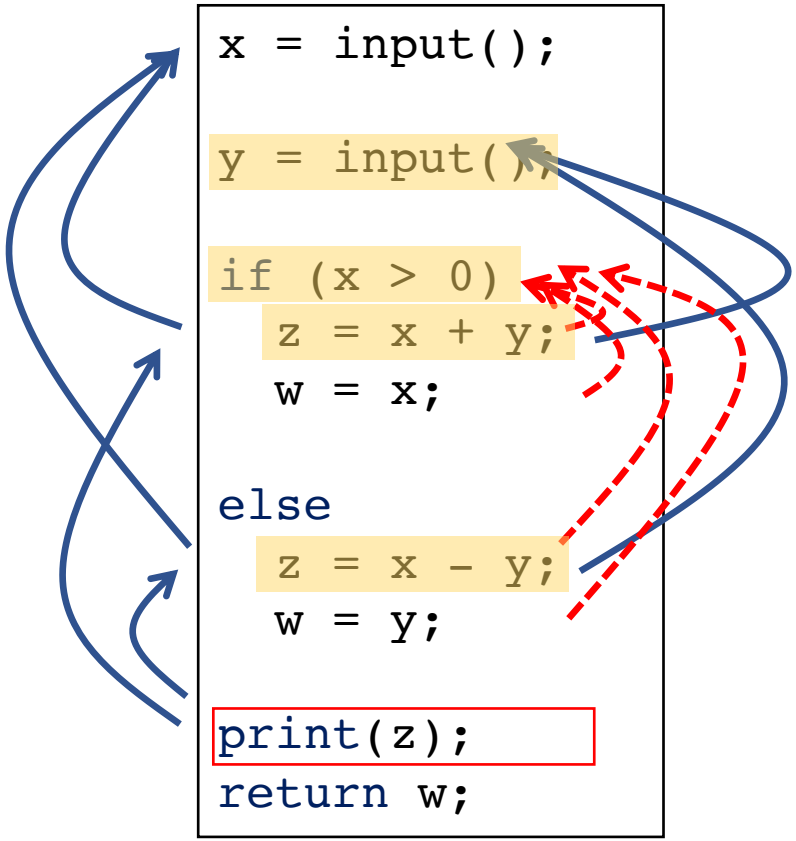

Dataflow dependence

Control dependence

Slicing criterion

Slice

# Program slicing



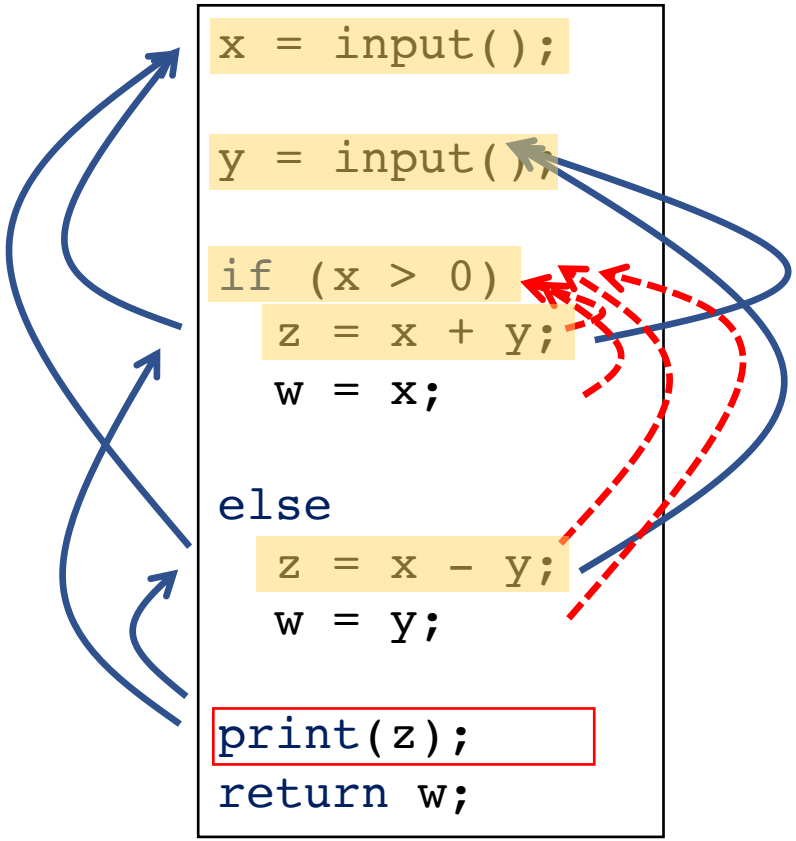

Dataflow dependence

Control dependence

Slicing criterion

Slice

# Program slicing




Dataflow dependence

Control dependence

Slicing criterion

Slice

# Static slicing: Analyze the program without executing it

```
x = input();

y = input();

if (x > 0)
   z = x + y;
   w = x;

else
   z = x - y;
   w = y;

print(z);
return w;
```

**Pros**

- All execution paths
- No runtime overhead

**Cons**

- Includes infeasible executions
- Cannot analyze dynamically-loaded code
- Scalability for complex code

# *Dynamic slicing: Analyze specific executions of the program*

```
x = input();

y = input();

if (x > 0)
   z = x + y;
   w = x;

else
   z = x - y;
   w = y;

print(z);
return w;
```

**Pros**

- Very accurate for the executed code
- Can analyze dynamically loaded code

**Cons**

- Triggered paths only
- Runtime overhead

# Usefulness of slicing

```
x = input();

y = input();

if (x > 0)
   z = x + y;
   w = x;

else
   z = x - y;
   w = y;

print(z);
return w;
```

# Usefulness of slicing

```
x = input();

y = input();

if (x > 0)
   z = x + y;
   w = x;

else
   z = x - y;
   w = y;

print(z);
return w;
```

- Aids manual analysis

# Usefulness of slicing

```
x = input();

y = input();

if (x > 0)
   z = x + y;
   w = x;

else
   z = x - y;
   w = y;

print(z);
return w;
```

- Aids manual analysis
- Core of may automated techniques
  - Fault localization
  - Malware detection
  - Refactoring
  - …

# *Usefulness of slicing*

```
x = input();

y = input();

if (x > 0)
   z = x + y;
   w = x;

else
   z = x - y;
   w = y;

print(z);
return w;
```

- Aids manual analysis
- Core of may automated techniques
  - Fault localization
  - Malware detection
  - Refactoring
  - …

In the rest of this talk:
we focus on dynamic slicing

# Big challenge: dataflow analysis

```
1  jane.age = 15
2  john     = jane
3  john.age = 25
4  res = jane.age
```

# *Big challenge: dataflow analysis*

```
1   jane.age = 15
2   john     = jane
3   john.age = 25
4   res = jane.age
```

*use*

# Big challenge: dataflow analysis

```
1   jane.age = 15
2   john     = jane
3   john.age = 25
4   res = jane.age
```

*use*

# Big challenge: dataflow analysis

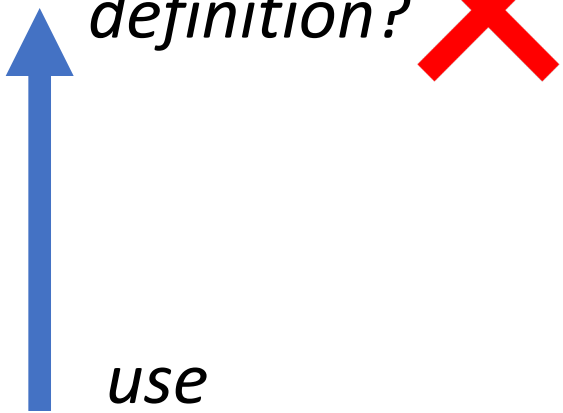
```
1   jane.age = 15
2   john     = jane
3   john.age = 25
4   res = jane.age
```

*definition?*

*use*

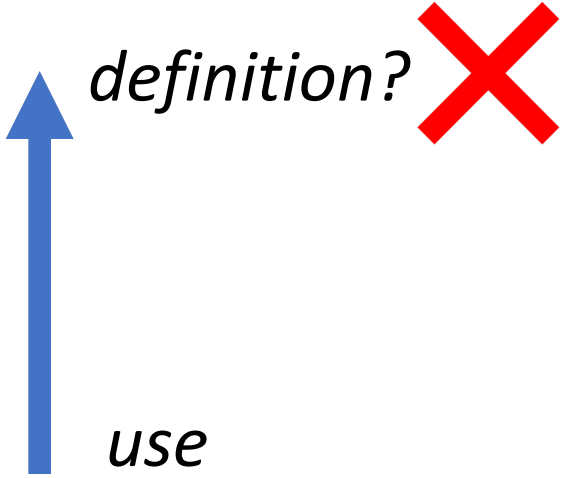
# Big challenge: dataflow analysis

```
1   jane.age = 15
2   john     = jane
3   john.age = 25
4   res = jane.age
```



# Big challenge: dataflow analysis

```
1   jane.age = 15
2   john     = jane
3   john.age = 25
4   res = jane.age
```




Needs alias analysis

# *Dynamic dataflow analysis*

```
1  jane.age = 15

2  john     = jane

3  john.age = 25

4  res = jane.age
```

# Dynamic dataflow analysis

```
1  jane.age = 15
     print([jane.age])
2  john     = jane

3  john.age = 25
     print([john.age])
4  res = jane.age
     print([jane.age])
```

# Dynamic dataflow analysis

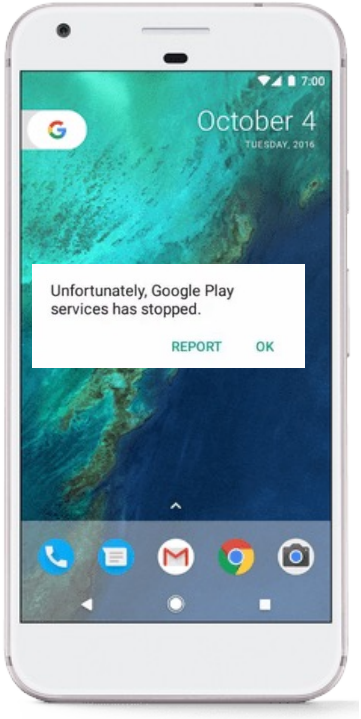
```
1  jane.age = 15                [jane.age] = 0xA
     print([jane.age])
2  john     = jane

3  john.age = 25                [john.age] = 0xA
     print([john.age])
4  res = jane.age               [jane.age] = 0xA
     print([jane.age])
```

# *Dynamic dataflow analysis*

```
1  jane.age = 15                [jane.age] = 0xA
     print([jane.age])
2  john     = jane

3  john.age = 25                [john.age] = 0xA
     print([john.age])
4  res = jane.age               [jane.age] = 0xA
     print([jane.age])
```

Instrumentation (even a very smart one)
→ High overhead

# *High overhead is bad*




On a mobile device:
system will kill slow apps

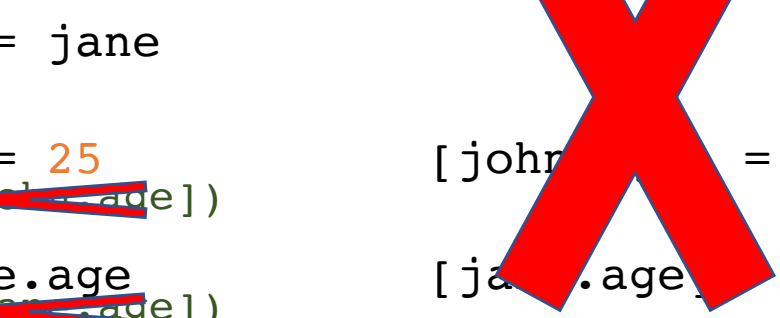
# Efficient dataflow analysis

```
1  jane.age = 15                [jane.age] = 0xA
     print([jane.age])
2  john     = jane

3  john.age = 25                [john.age] = 0xA
     print([john.age])
4  res = jane.age               [jane.age] = 0xA
     print([jane.age])
```

# Efficient dataflow analysis

```
1  jane.age = 15
     print([jane.age])
2  john     = jane

3  john.age = 25
     print([john.age])
4  res = jane.age
     print([jane.age])
```

```
[ja    age    0xA

[john       = 0xA

[ja  .age]  0xA
```



Don’t record addresses

# *Efficient dynamic slicing for mobile**

* Khaled Ahmed, Mieszko Lis, and Julia Rubin. MANDOLINE: Dynamic Slicing of Android Applications with Trace-Based Alias Analysis. ICST, **Distinguished Paper Award**, 2021

# *Efficient dynamic slicing for mobile**

1. Light instrumentation ➔ only basic blocks, not every statement, no fields

* Khaled Ahmed, Mieszko Lis, and Julia Rubin. MANDOLINE: Dynamic Slicing of Android Applications with Trace-Based Alias Analysis. ICST, **Distinguished Paper Award**, 2021

# *Efficient dynamic slicing for mobile**

1. Light instrumentation ➔ only basic blocks, not every statement, no fields
2. Post-execution analysis of the trace

* Khaled Ahmed, Mieszko Lis, and Julia Rubin. MANDOLINE: Dynamic Slicing of Android Applications with Trace-Based Alias Analysis. ICST, **Distinguished Paper Award**, 2021

# *Efficient dynamic slicing for mobile**

1. Light instrumentation ➔ only basic blocks, not every statement, no fields
2. Post-execution analysis of the trace
   - Recovers dataflows

* Khaled Ahmed, Mieszko Lis, and Julia Rubin. MANDOLINE: Dynamic Slicing of Android Applications with Trace-Based Alias Analysis. ICST, **Distinguished Paper Award**, 2021

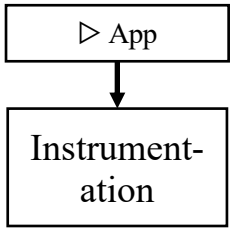
# *Efficient dynamic slicing for mobile**

1. Light instrumentation ➔ only basic blocks, not every statement, no fields
2. Post-execution analysis of the trace
   - Recovers dataflows
   - Static, but analyzes one execution path only

* Khaled Ahmed, Mieszko Lis, and Julia Rubin. MANDOLINE: Dynamic Slicing of Android Applications with Trace-Based Alias Analysis. ICST, **Distinguished Paper Award**, 2021

# *Efficient dynamic slicing for mobile**

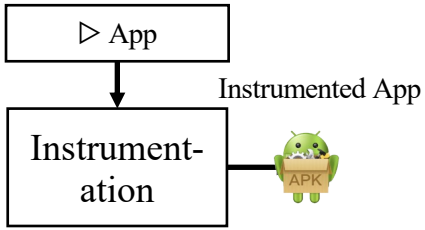
1. Light instrumentation ➔ only basic blocks, not every statement, no fields
2. Post-execution analysis of the trace
   - Recovers dataflows
   - Static, but analyzes one execution path only




* Khaled Ahmed, Mieszko Lis, and Julia Rubin. MANDOLINE: Dynamic Slicing of Android Applications with Trace-Based Alias Analysis. ICST, **Distinguished Paper Award**, 2021

# *Efficient dynamic slicing for mobile**

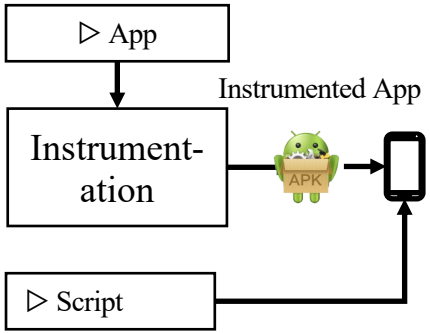
1. Light instrumentation ➔ only basic blocks, not every statement, no fields
2. Post-execution analysis of the trace
   - Recovers dataflows
   - Static, but analyzes one execution path only




* Khaled Ahmed, Mieszko Lis, and Julia Rubin. MANDOLINE: Dynamic Slicing of Android Applications with Trace-Based Alias Analysis. ICST, **Distinguished Paper Award**, 2021

# *Efficient dynamic slicing for mobile**

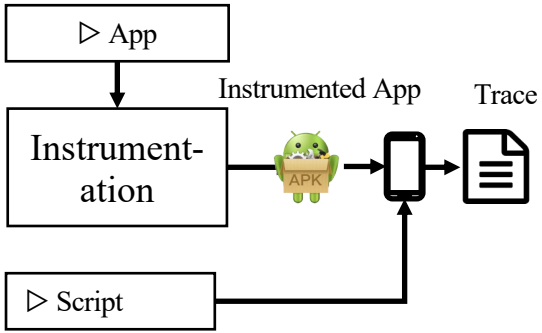
1. Light instrumentation ➔ only basic blocks, not every statement, no fields
2. Post-execution analysis of the trace
   - Recovers dataflows
   - Static, but analyzes one execution path only




* Khaled Ahmed, Mieszko Lis, and Julia Rubin. MANDOLINE: Dynamic Slicing of Android Applications with Trace-Based Alias Analysis. ICST, **Distinguished Paper Award**, 2021

# *Efficient dynamic slicing for mobile**

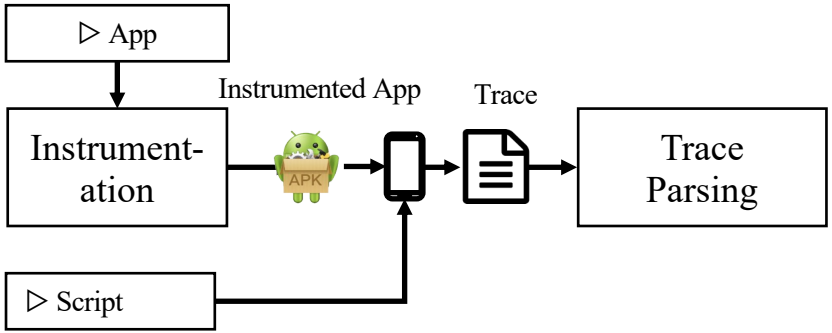
1. Light instrumentation ➔ only basic blocks, not every statement, no fields
2. Post-execution analysis of the trace
   - Recovers dataflows
   - Static, but analyzes one execution path only




* Khaled Ahmed, Mieszko Lis, and Julia Rubin. MANDOLINE: Dynamic Slicing of Android Applications with Trace-Based Alias Analysis. ICST, **Distinguished Paper Award**, 2021

# *Efficient dynamic slicing for mobile**

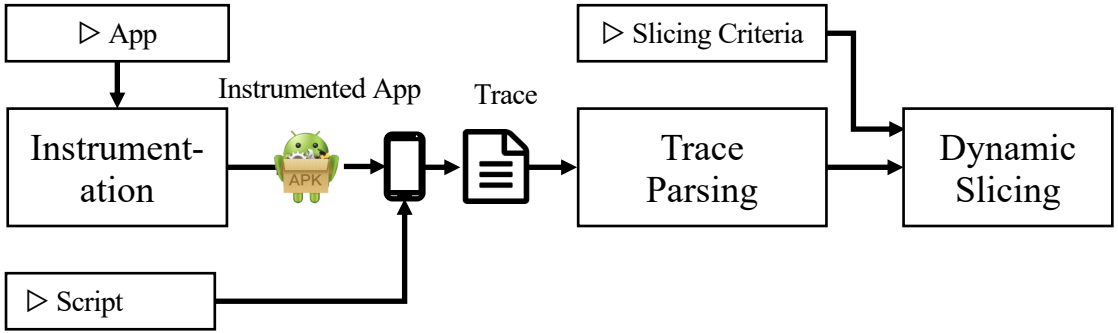
1. Light instrumentation ➔ only basic blocks, not every statement, no fields
2. Post-execution analysis of the trace
   - Recovers dataflows
   - Static, but analyzes one execution path only




* Khaled Ahmed, Mieszko Lis, and Julia Rubin. MANDOLINE: Dynamic Slicing of Android Applications with Trace-Based Alias Analysis. ICST, **Distinguished Paper Award**, 2021

# *Efficient dynamic slicing for mobile**

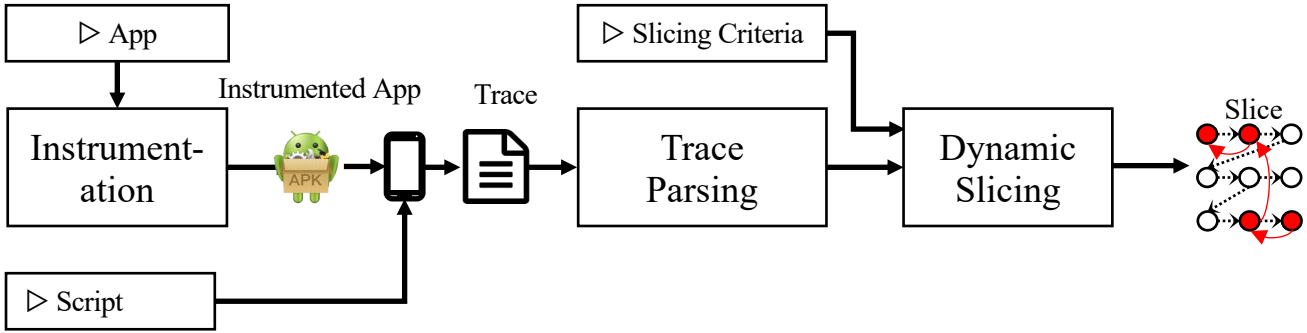
1. Light instrumentation ➔ only basic blocks, not every statement, no fields
2. Post-execution analysis of the trace
   - Recovers dataflows
   - Static, but analyzes one execution path only




* Khaled Ahmed, Mieszko Lis, and Julia Rubin. MANDOLINE: Dynamic Slicing of Android Applications with Trace-Based Alias Analysis. ICST, **Distinguished Paper Award**, 2021

# *Efficient dynamic slicing for mobile**

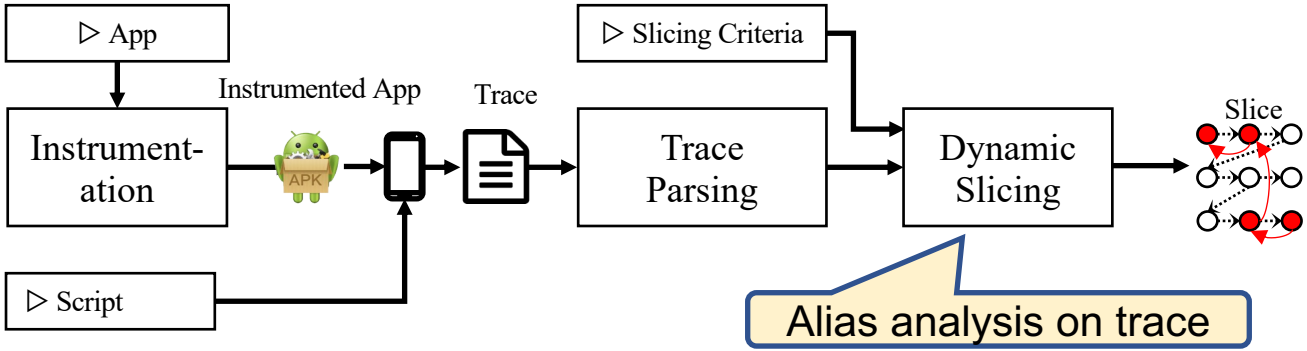
1. Light instrumentation ➔ only basic blocks, not every statement, no fields
2. Post-execution analysis of the trace
   - Recovers dataflows
   - Static, but analyzes one execution path only




* Khaled Ahmed, Mieszko Lis, and Julia Rubin. MANDOLINE: Dynamic Slicing of Android Applications with Trace-Based Alias Analysis. ICST, **Distinguished Paper Award**, 2021

# *Efficient dynamic slicing for mobile**

1. Light instrumentation ➔ only basic blocks, not every statement, no fields
2. Post-execution analysis of the trace
   - Recovers dataflows
   - Static, but analyzes one execution path only




* Khaled Ahmed, Mieszko Lis, and Julia Rubin. MANDOLINE: Dynamic Slicing of Android Applications with Trace-Based Alias Analysis. ICST, **Distinguished Paper Award**, 2021

# Trace based alias analysis

```
1   jane.age = 15
2   john     = jane
3   john.age = 25
4   res = jane.age
```

john aliases jane
john.age aliases jane.age

# Trace based alias analysis

```
1  jane.age = 15
2  john     = jane
3  john.age = 25
4  res = jane.age
```

# Trace based alias analysis

```
1   jane.age = 15
2   john     = jane
3   john.age = 25
4   res = jane.age
```

*use*

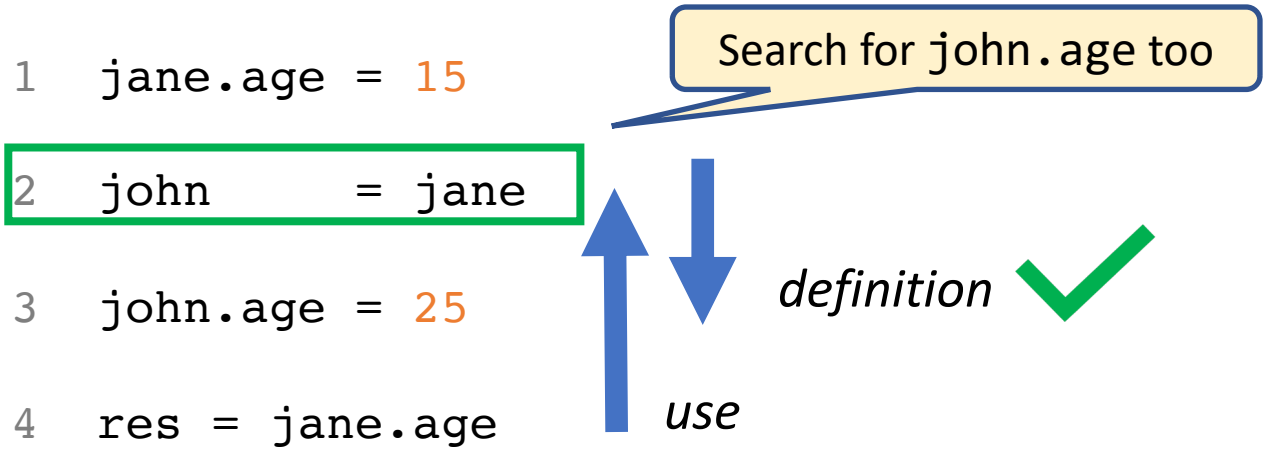
# Trace based alias analysis

```
1   jane.age = 15
2   john     = jane
3   john.age = 25
4   res = jane.age
```

Search for `john.age` too

*use*

# Trace based alias analysis



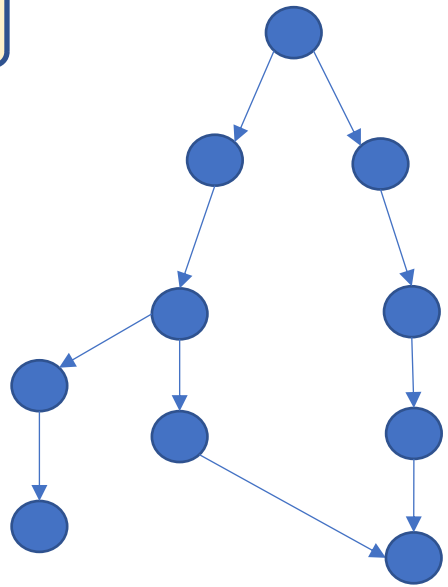
# Trace based alias analysis

```
1   jane.age = 15
2   john     = jane
3   john.age = 25
4   res = jane.age
```

Search for john.age too

*use*

*definition*



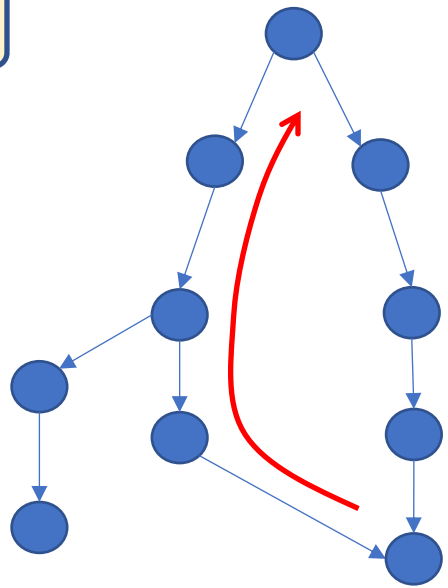
# Trace based alias analysis

```
1   jane.age = 15
2   john     = jane
3   john.age = 25
4   res = jane.age
```

Search for `john.age` too

*use*

*definition*



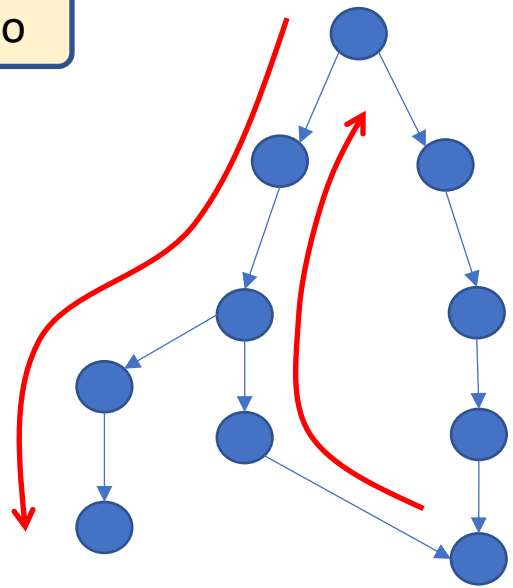
# Trace based alias analysis

```
1   jane.age = 15
2   john     = jane
3   john.age = 25
4   res = jane.age
```

Search for john.age too

*definition*

*use*



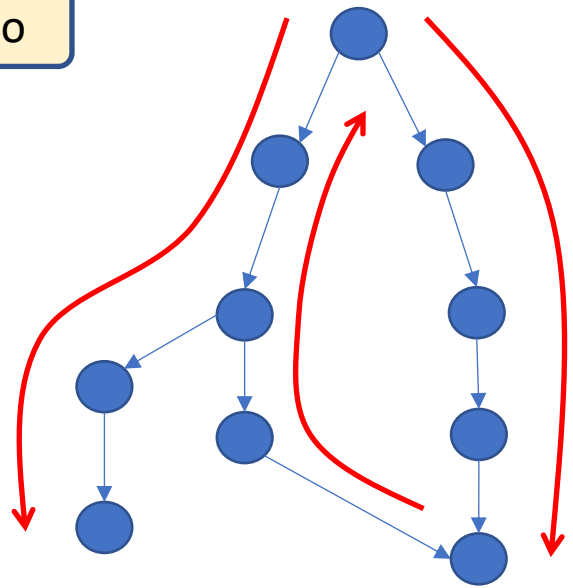
# Trace based alias analysis

```
1   jane.age = 15
2   john     = jane
3   john.age = 25
4   res = jane.age
```

Search for john.age too

*definition*

*use*



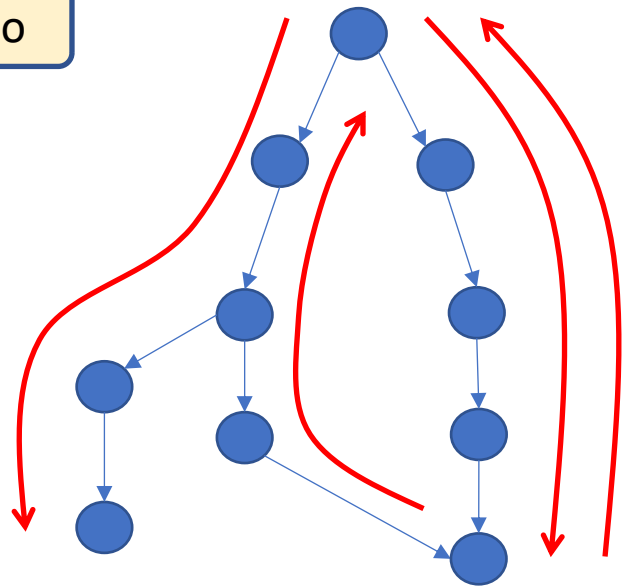
# Trace based alias analysis

```
1   jane.age = 15
2   john     = jane
3   john.age = 25
4   res = jane.age
```

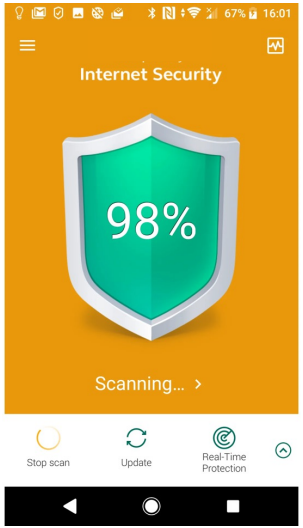
Search for `john.age` too

*definition*

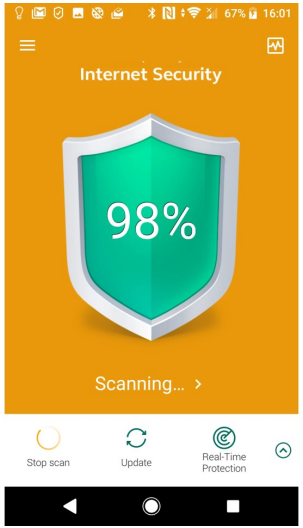
*use*



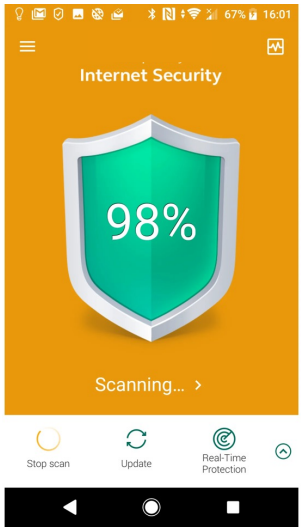
# Application in malware analysis

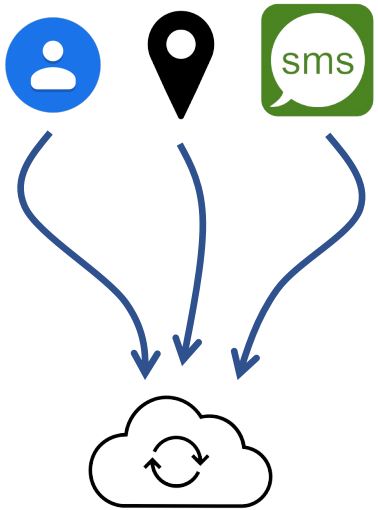


# Application in malware analysis




# Application in malware analysis




Slice separates malware from benign

# Where we are

## *MANDOLINE: Dynamic slicing for Android*


- Paper: Mandoline: Dynamic Slicing of Android Applications with Trace-Based Alias Analysis. ICST, Distinguished Paper Award, 2021
- Code: https://github.com/resess/Mandoline

## *Slicer4J: Dynamic slicing for Java*


- Paper: Slicer4J: A Dynamic Slicer for Java. ESEC/FSE, tools track, 2021
- Code: https://github.com/resess/Slicer4J

## *IntelliJ plugin for Slicer4J: Capstone project*

- Integeration of slicing, slice visualization, and slice navigation in the most popular IDE!

# What's next

## *Automated malware summarization / detection*

- Using slices to summarize the behavior of malware / detect new samples

## *Slicing for software maintainance*

- Slicing to compare versions of code

## *Interested? Join us!*

*Contact* : Khaled Ahmed, https://khaled-e-a.github.io
khaledea@ece.ubc.ca
Julia Rubin, https://people.ece.ubc.ca/mjulia/,
mjulia@ece.ubc.ca