Software Engineering

Paulo Borba
Informatics Center
Federal University of Pernambuco

pauloborba.cin.ufpe.br

What is static analysis?

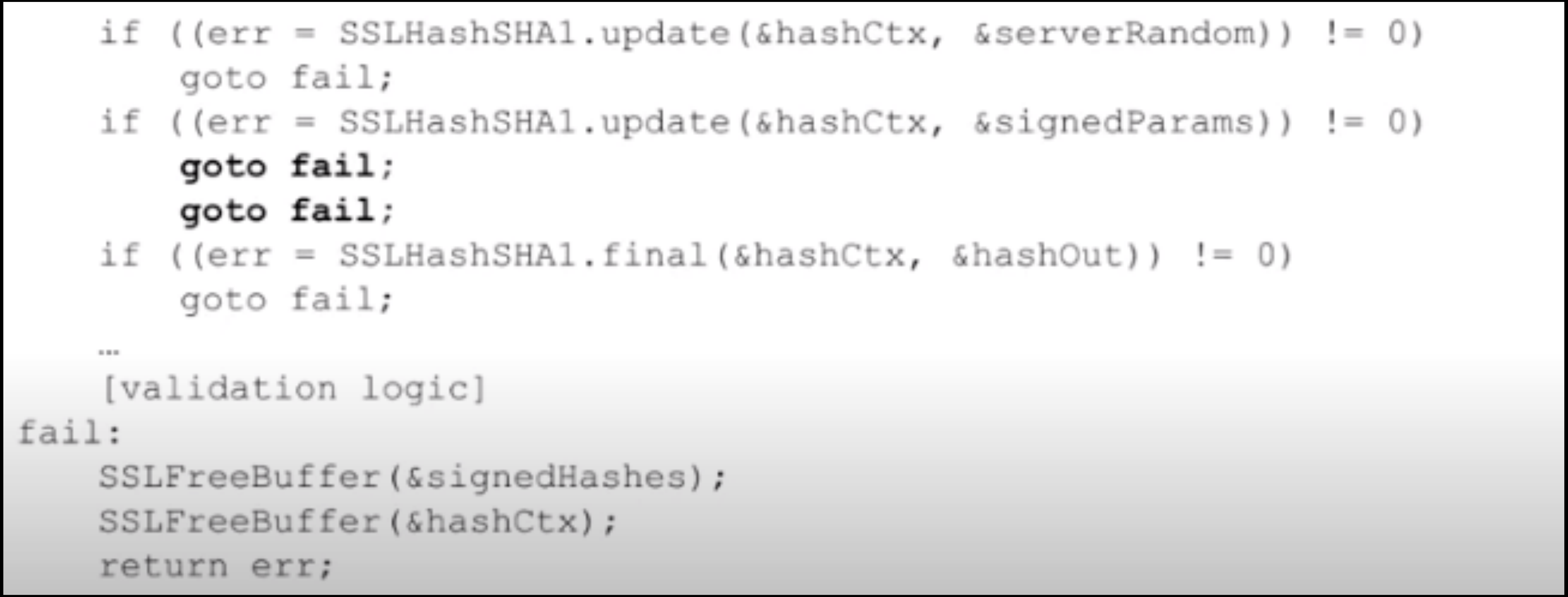
Process of inferring properties of a program without executing it

with the aim of finding issues

(defects, deviation from standard style, lack of documentation, bad smells, etc).

Apple goto bug!

could have been easily detect by a simple static analysis



```
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;
    …
    [validation logic]
fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;
```

Properties

of source code, configuration files, etc.

- if-else is used with block statements
- No two identical consecutive statements
- No string comparison with ==
- No dead code
- No NullPointerException
- Sensitive information does not leak
- Changes from a branch do not interfere with changes from another
- Every method has a Javadoc

How can we do that?

Depends on what we want to check

and how the program to be analyzed is represented

```
export class Student {
  name: string;
  cpf: string;
  email: string;
  goals: Map<string,string>;
  …
  clone(): Student {
    var student: Student = new Student();
    …
    return student;
  }
}
```

Program as a string

is the clone method ever called?

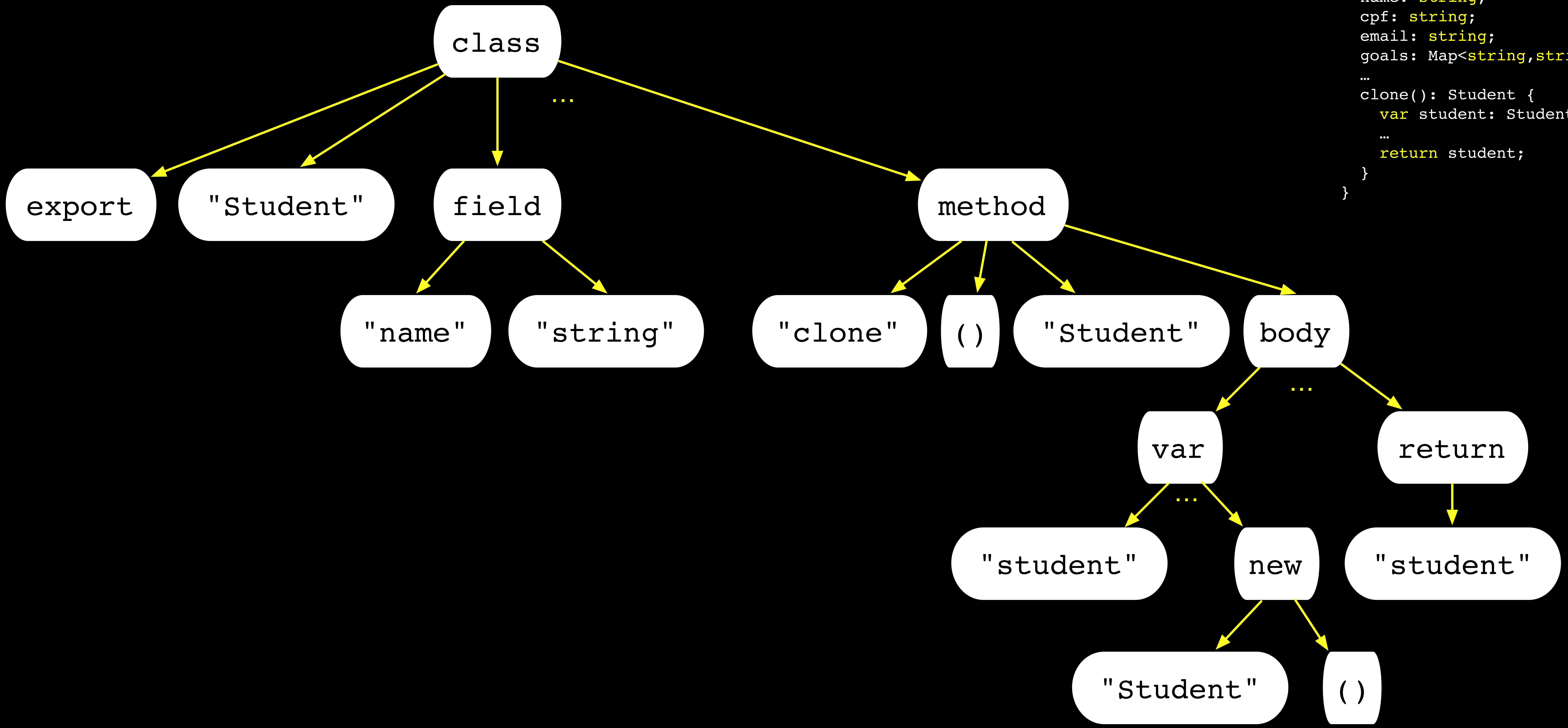
```
"export class Student {\n  name: string;\n  cpf: string;\n
 email: string;\n  goals: Map<string,string>;\n … \n
 clone(): Student {\n    var student: Student = new Student();\n
    …\n    return student;\n  }\n}"
```

```
export class Student {
  name: string;
  cpf: string;
  email: string;
  goals: Map<string,string>;
  …
  clone(): Student {
    var student: Student = new Student();
    …
    return student;
  }
}
```

Program as an AST (Abstract Syntax Tree)

is clone recursive? does clone initialises student?

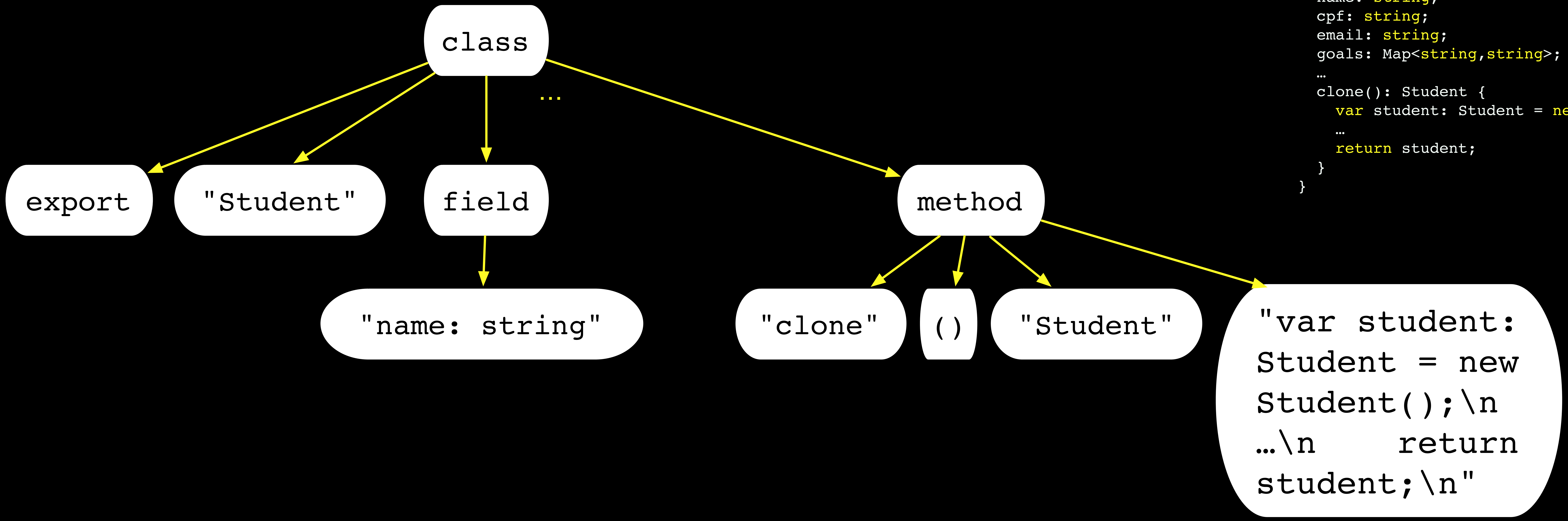
```
export class Student {
  name: string;
  cpf: string;
  email: string;
  goals: Map<string,string>;
  …
  clone(): Student {
    var student: Student = new Student();
    …
    return student;
  }
}
```



Partial ASTs might be enough

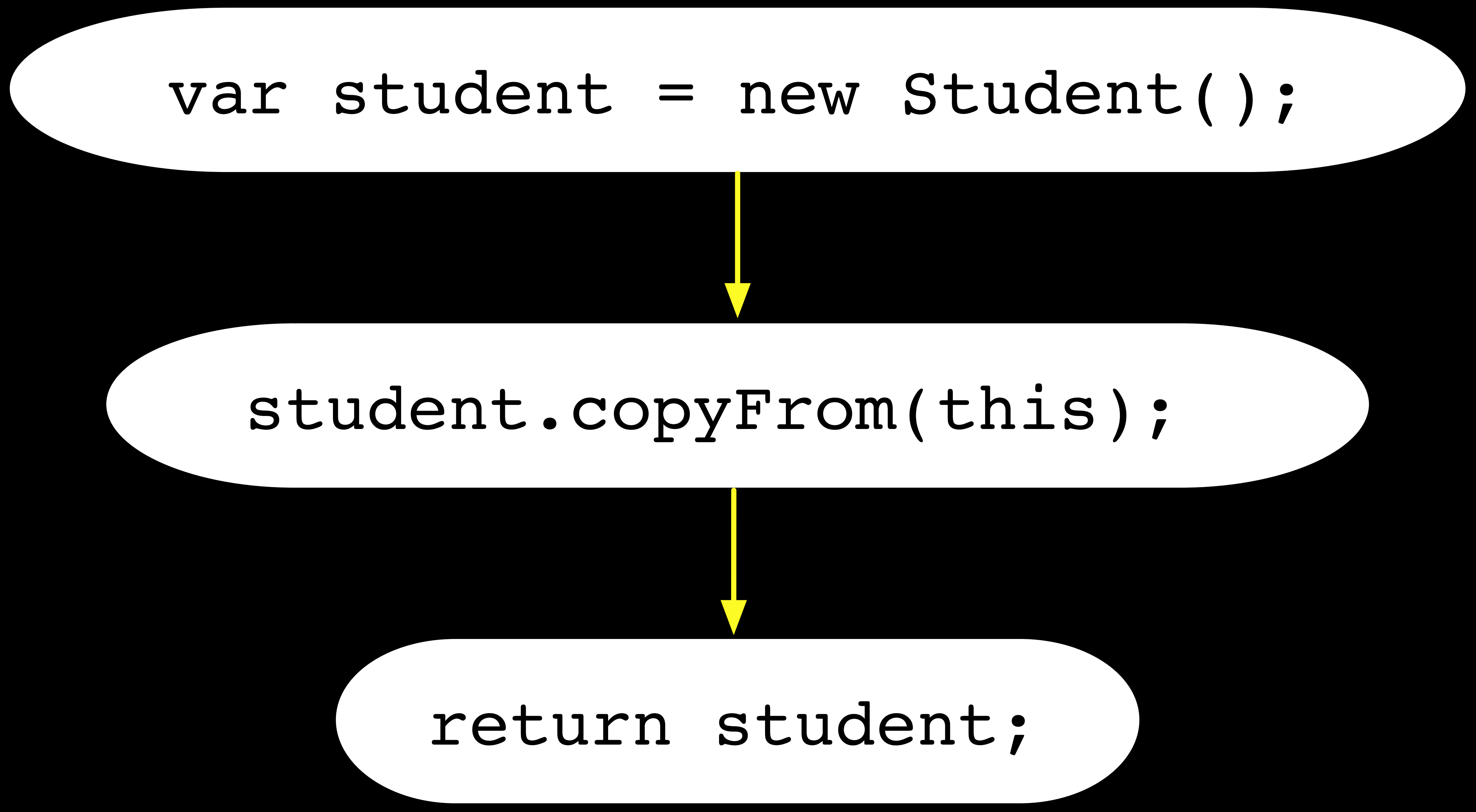
is name a field of Student?

```
export class Student {
  name: string;
  cpf: string;
  email: string;
  goals: Map<string,string>;
  …
  clone(): Student {
    var student: Student = new Student();
    …
    return student;
  }
}
```


class
…
export
"Student"
field
method
"name: string"
"clone"
()
"Student"
"var student: Student = new Student();\n …\n   return student;\n"

Method body as CFG (Control Flow Graph)

is student initialised before being used?

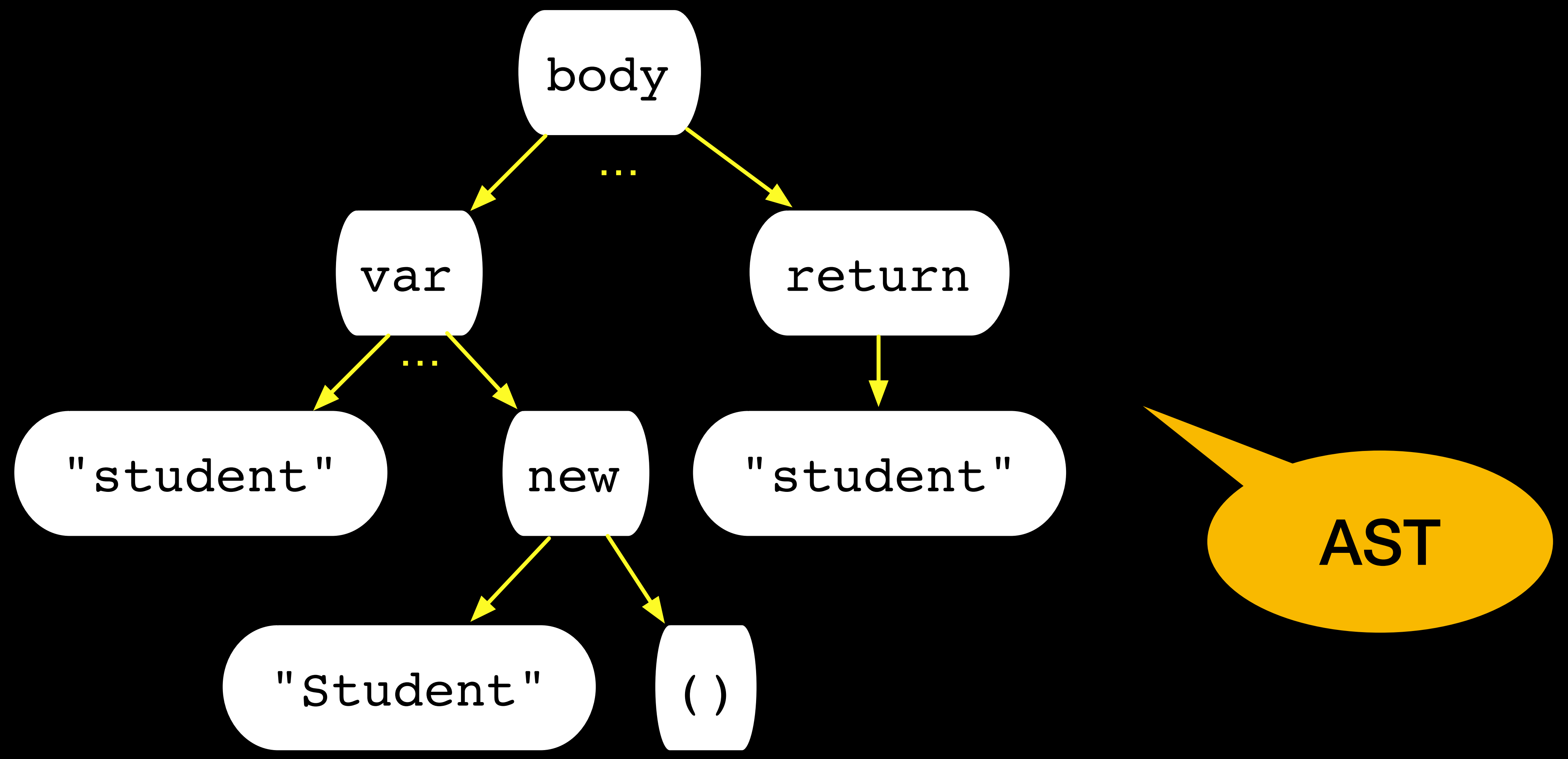


```
var student = new Student();
```

```
student.copyFrom(this);
```

```
return student;
```

```
var student: Student = new Student();
…
return student;
```



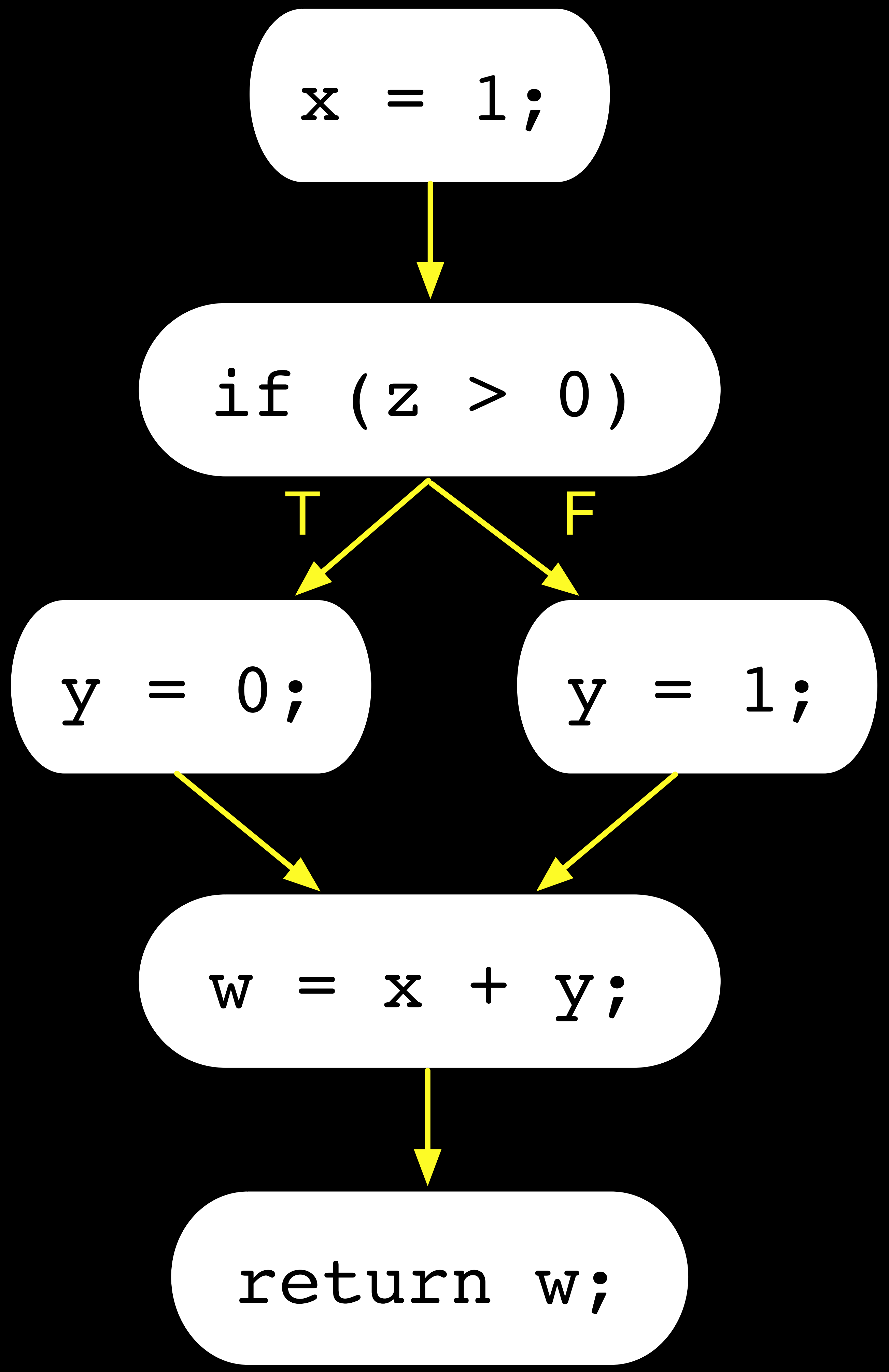
What’s the main difference between a CFG and an AST?

Can you think of a property that you would like to check that could be detected with an AST but not with a CFG? And vice-versa?

Representing conditionals in CFGs

is y initialised before it is used?

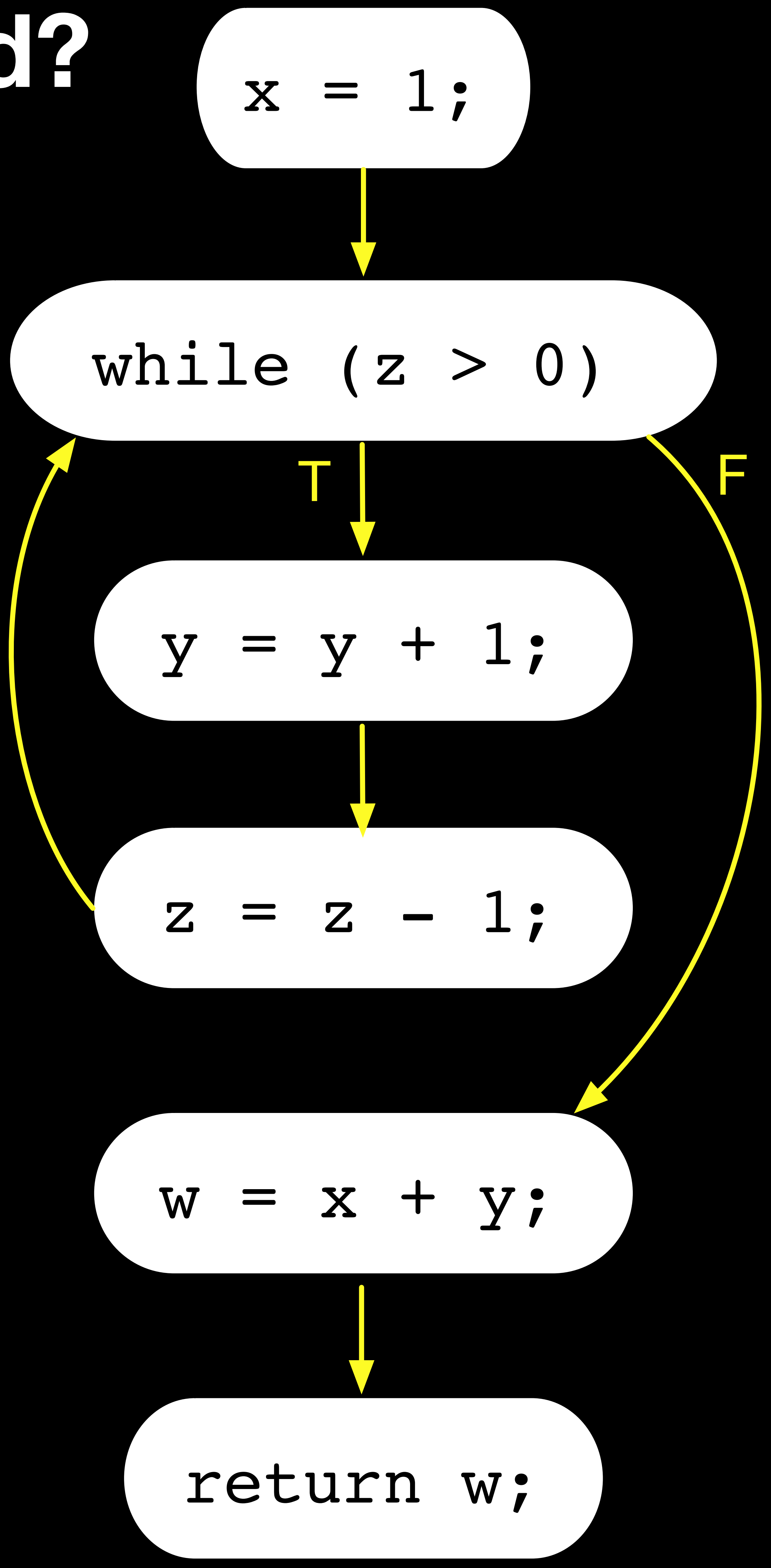
```
x = 1;
if (z > 0) {
   y = 0;
} else {
   y = 1;
}
w = x + y;
return w;
```



Representing loops in CFGs

is y initialised before it is used?

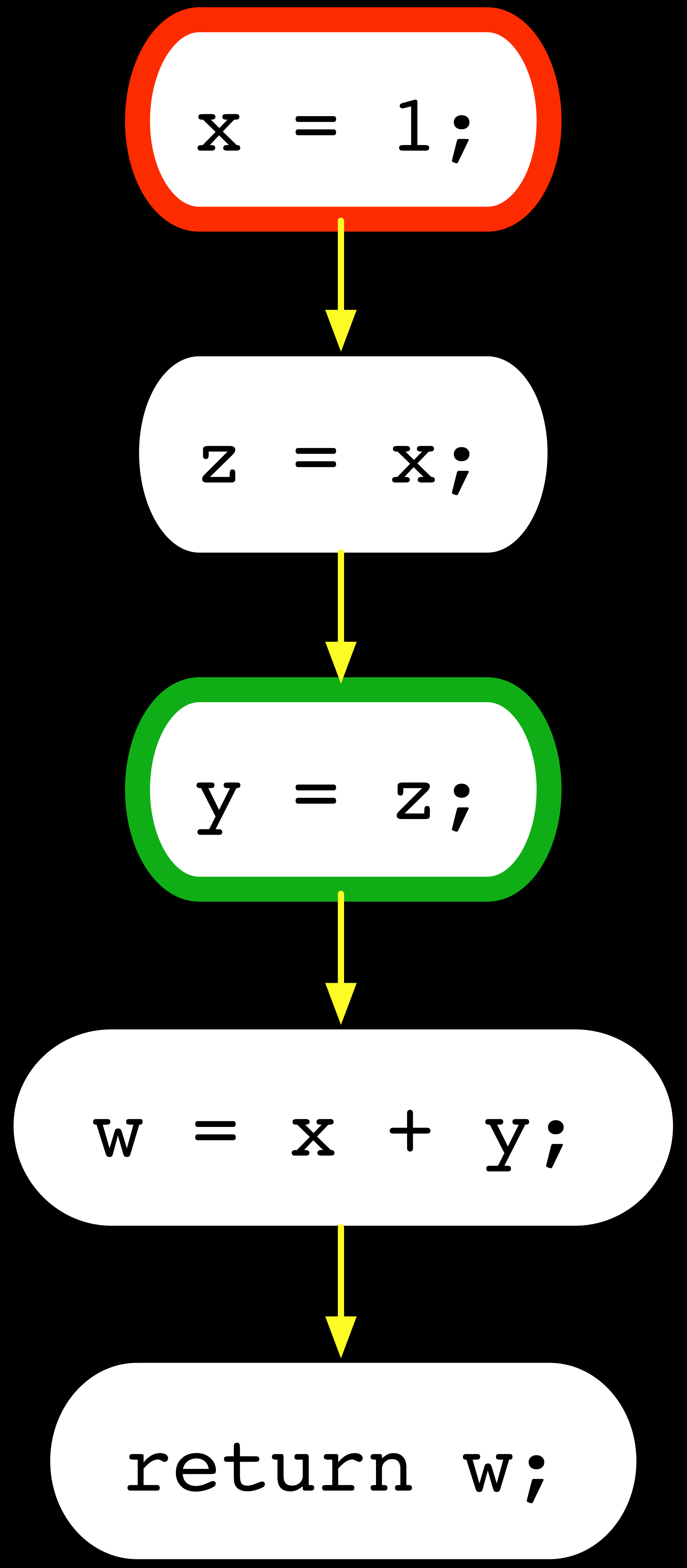
```
x = 1;
while (z > 0) {
  y = y + 1;
  z = z - 1;
}
w = x + y;
return w;
```



Taint analysis

does sensitive information (red) leaks (green)?

```
x = 1;
z = x;
y = z;
w = x + y;
return w;
```



Analysis abstraction

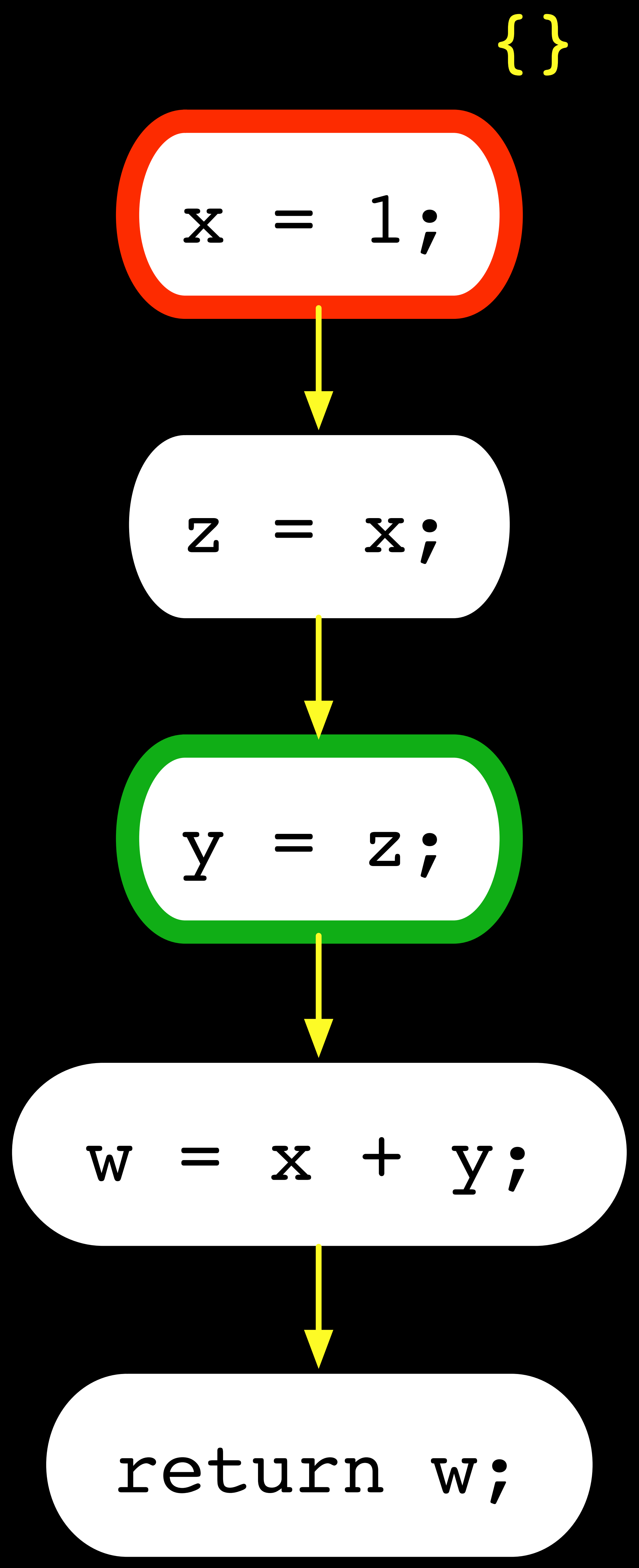
information computed by the analysis so that property can be checked

```
{x,z,y}
```

for taint analysis: set of tainted variables

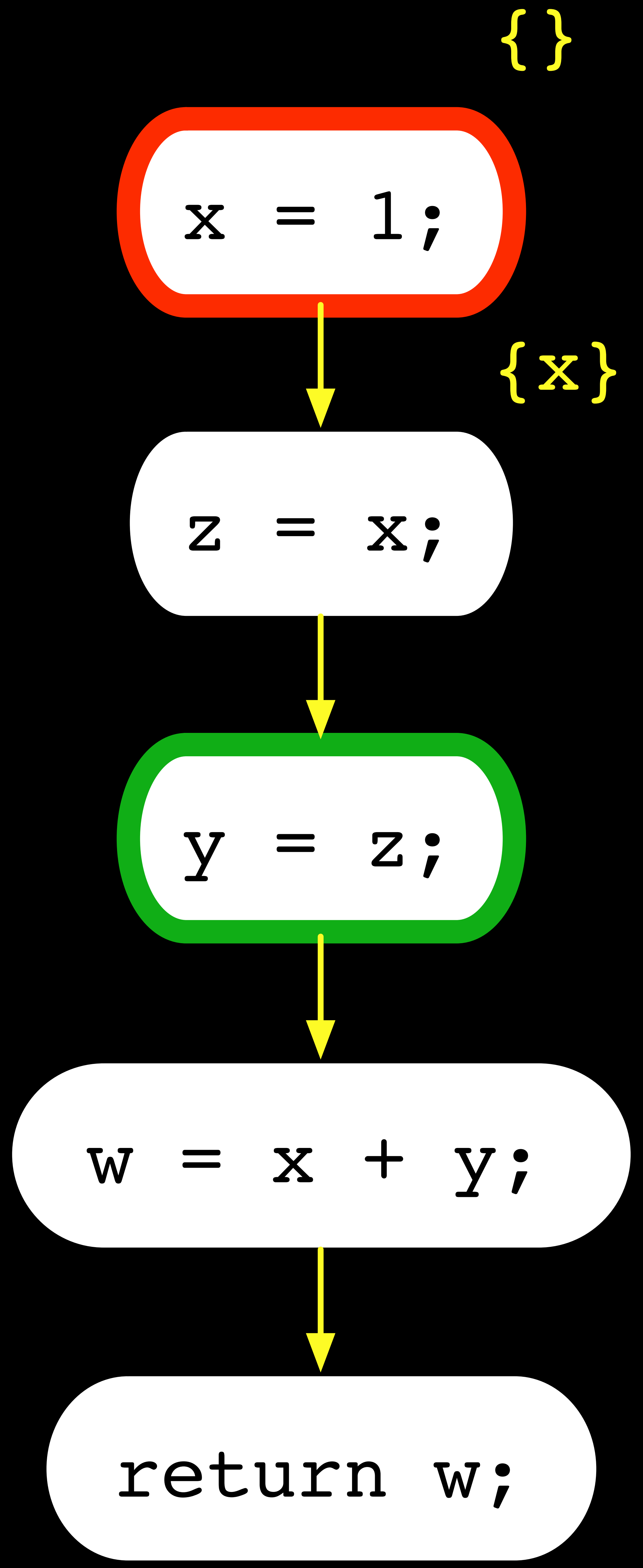
variables that can potentially contain sensitive information

Stepwise analysis process



{ }

```
x = 1;
z = x;
y = z;
w = x + y;
return w;
```



{}

x = 1;

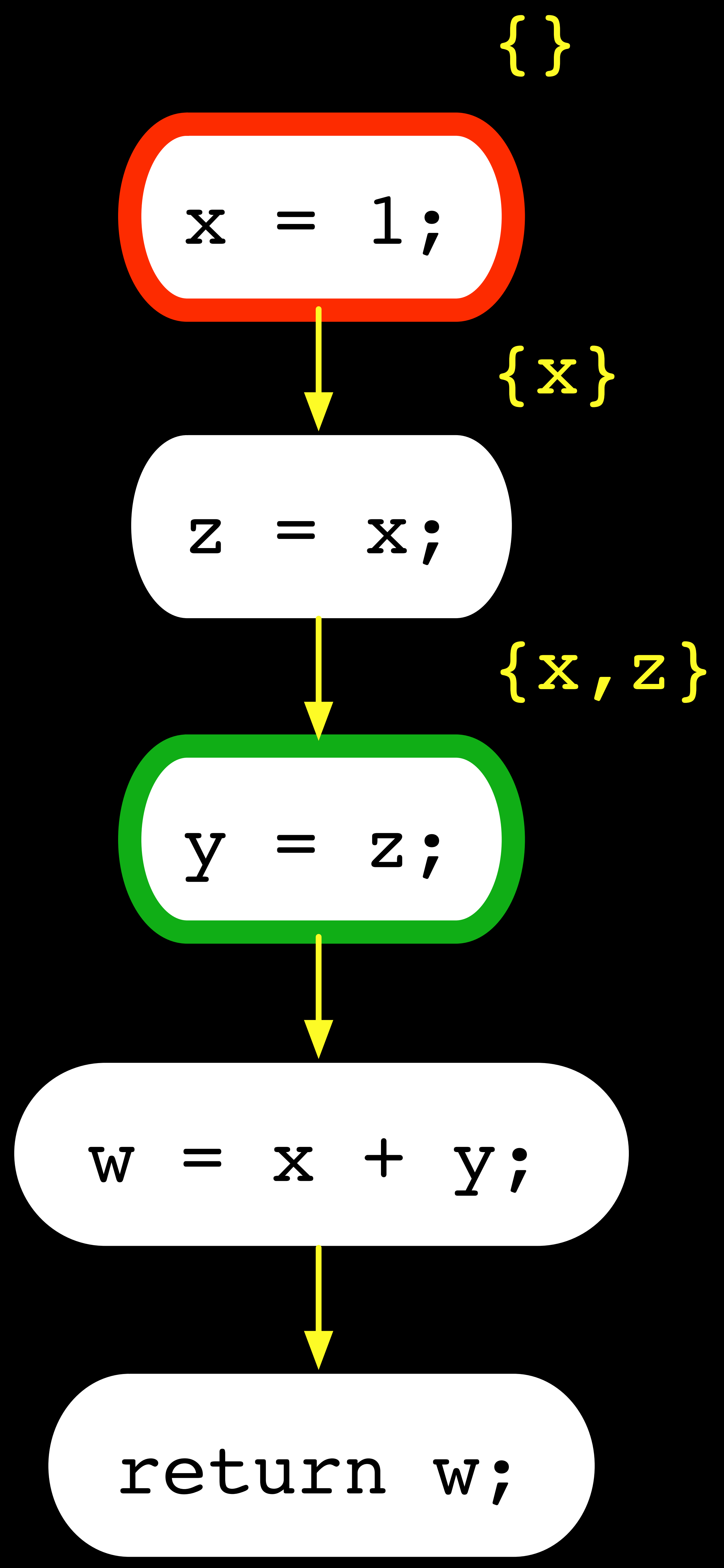
{x}

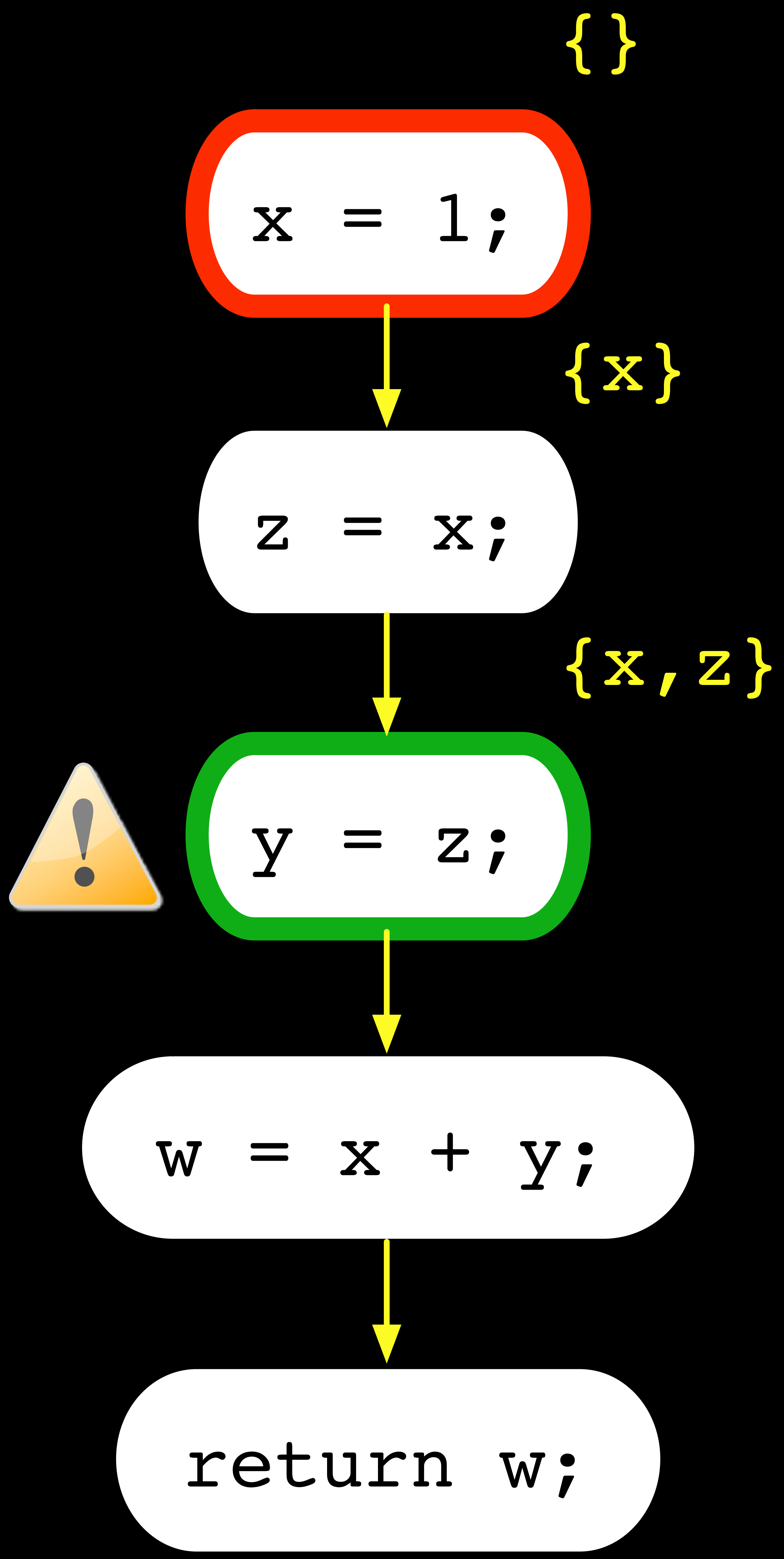
z = x;

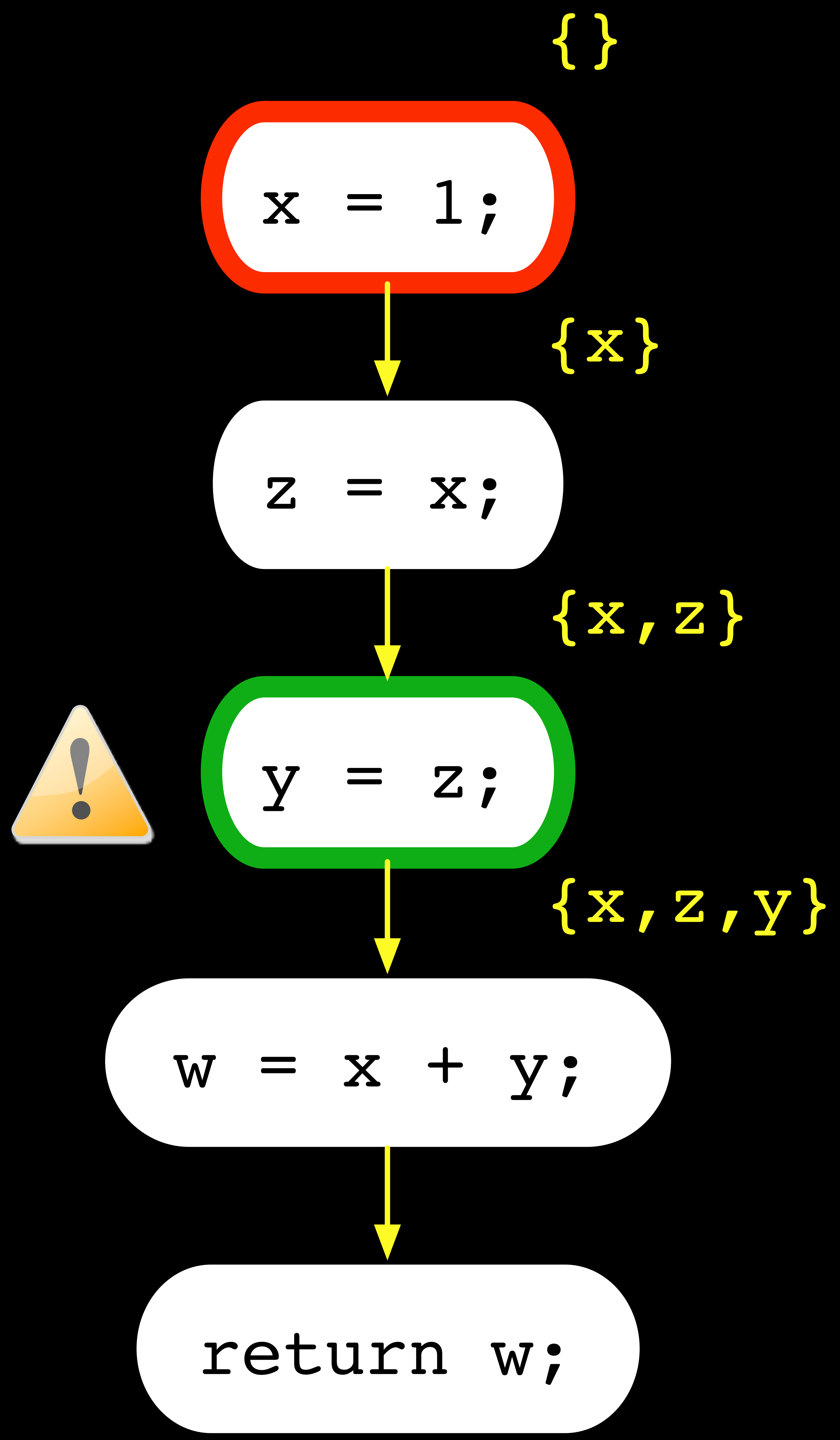
y = z;

w = x + y;

return w;


{}
x = 1;
{x}
z = x;
{x,z}
y = z;
w = x + y;
return w;


{}
x = 1;
{x}
z = x;
{x,z}
y = z;
w = x + y;
return w;



{}

x = 1;

{x}

z = x;

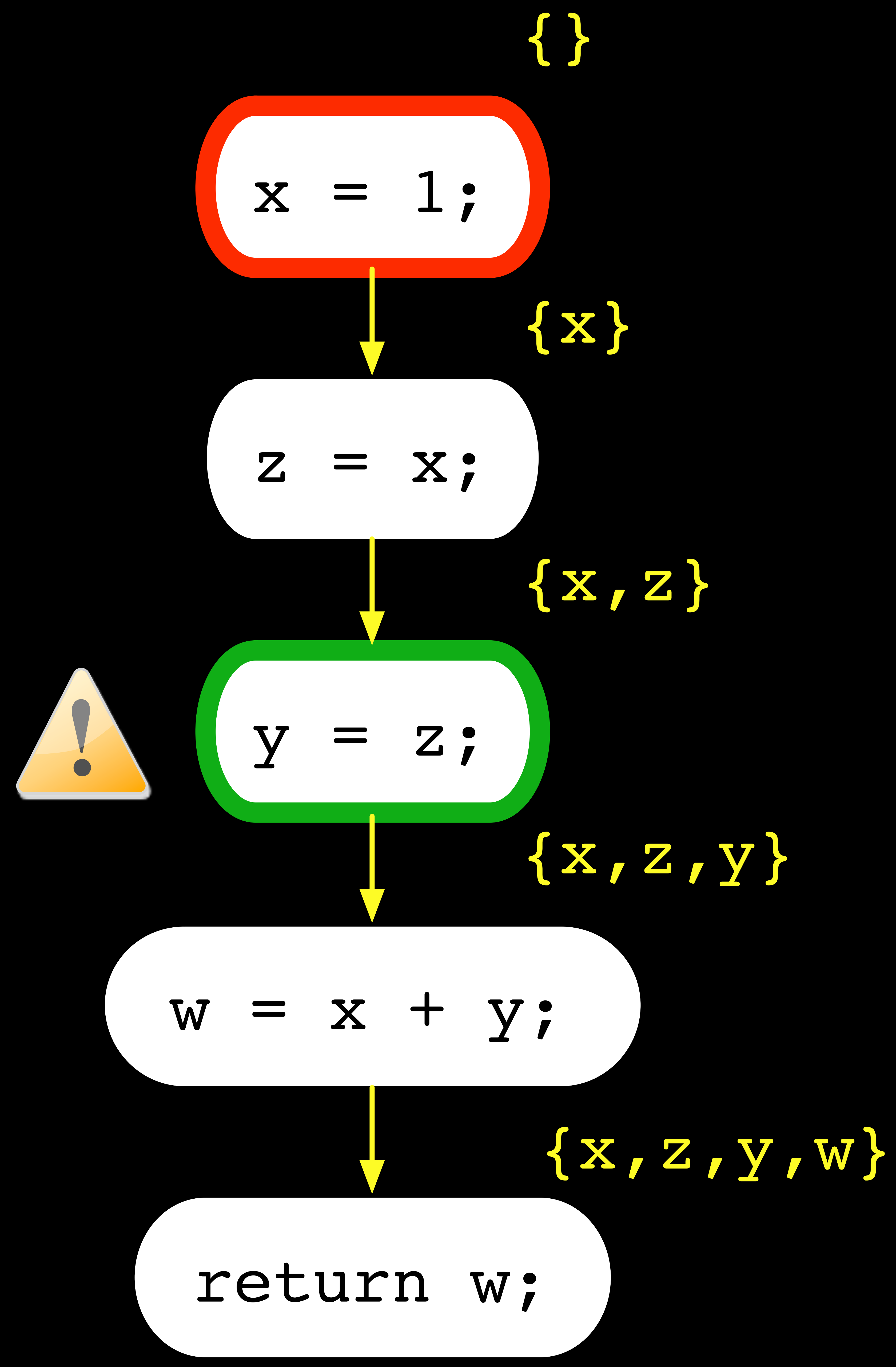
{x,z}

y = z;

{x,z,y}

w = x + y;

return w;



{}

x = 1;

{x}

z = x;

{x,z}

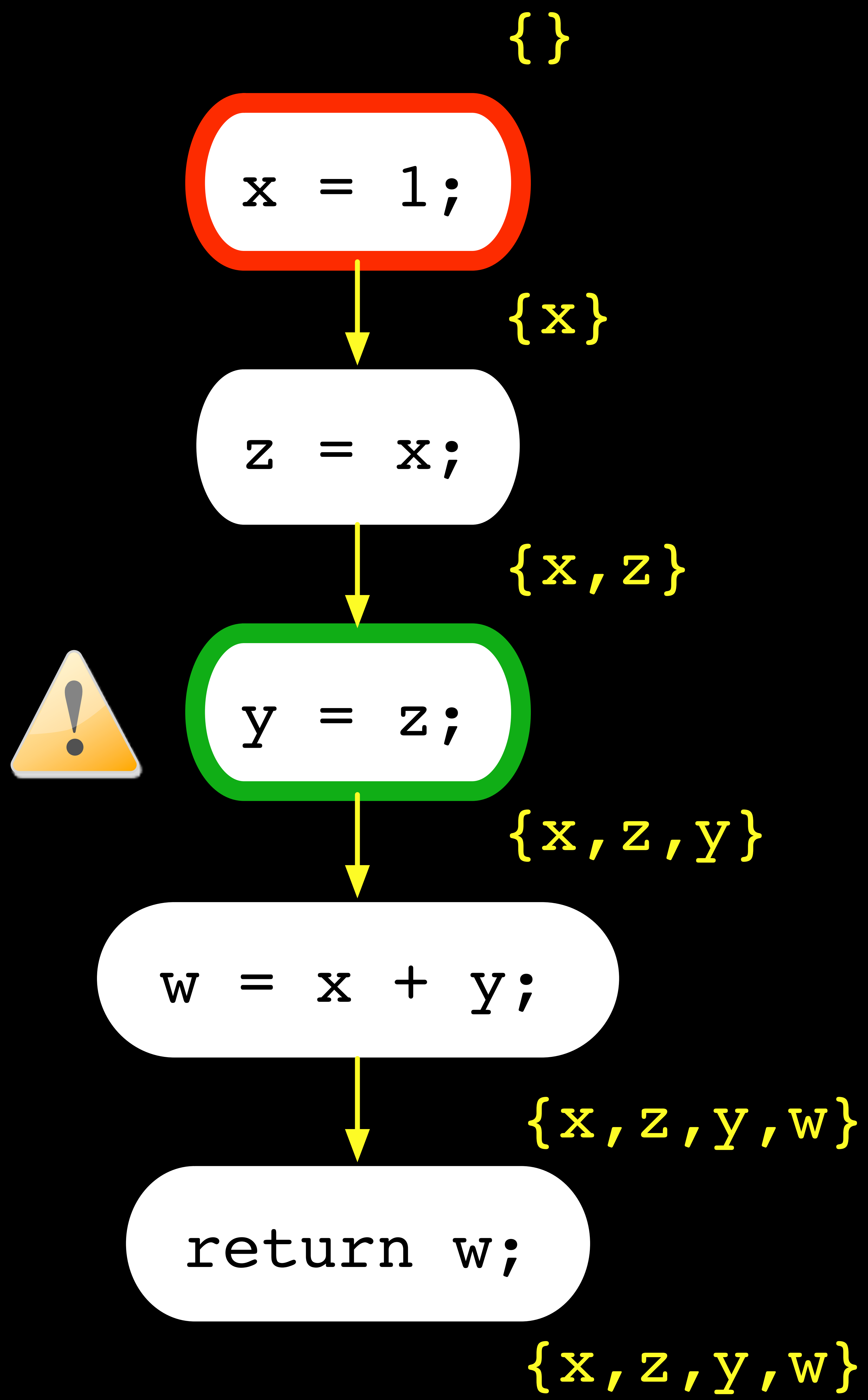
y = z;

{x,z,y}

w = x + y;

{x,z,y,w}

return w;



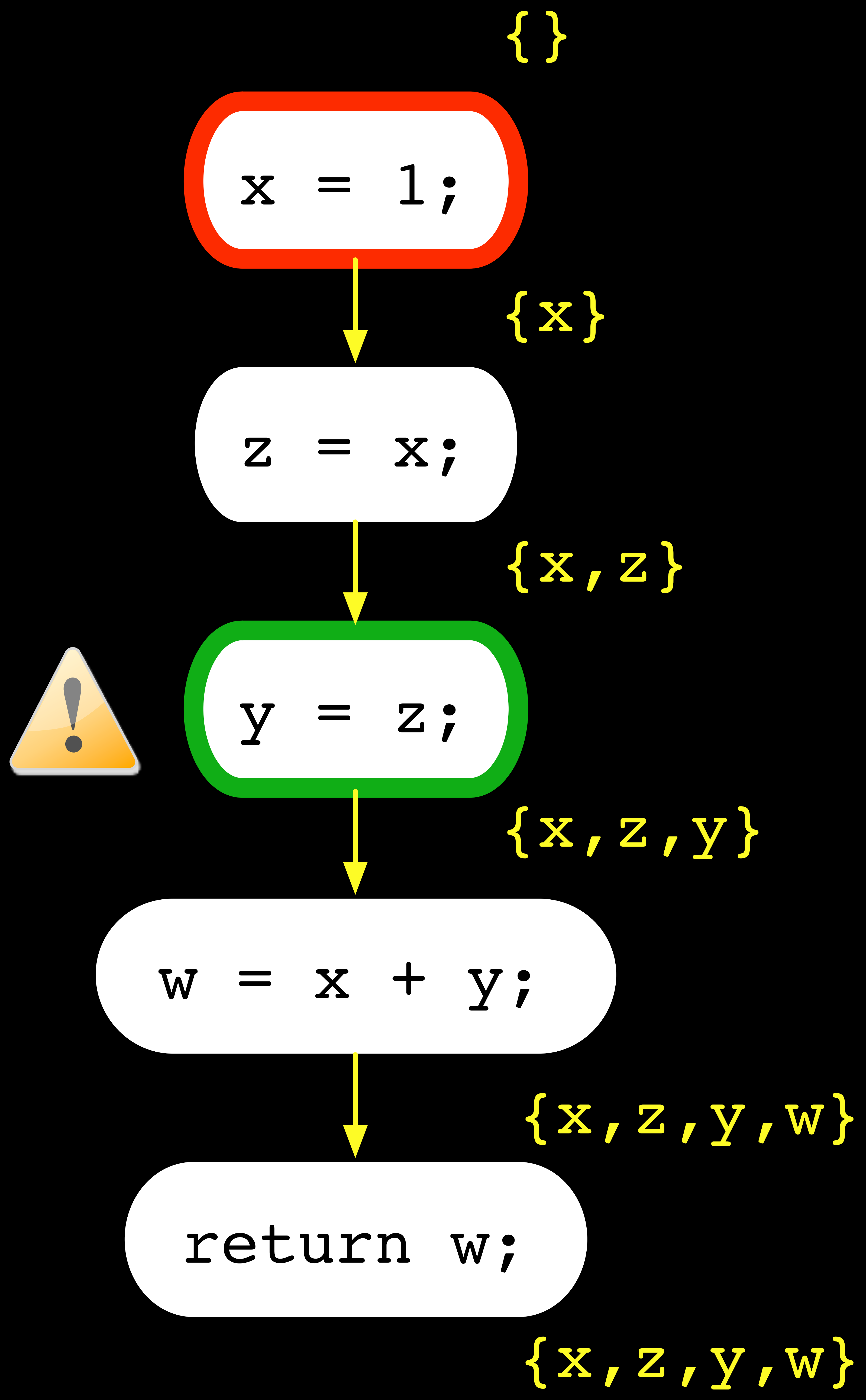
```
{}
x = 1;
{x}
z = x;
{x,z}
y = z;
{x,z,y}
w = x + y;
{x,z,y,w}
return w;
{x,z,y,w}
```

What if we had z = 0; in the second node on this CFG?

What would be the final computed abstraction?

Would a warning be raised?

Consider that the initial abstraction is still { }. Draw all intermediate abstractions.



{}

x = 1;

{x}

z = x;

{x,z}

y = z;

{x,z,y}

w = x + y;

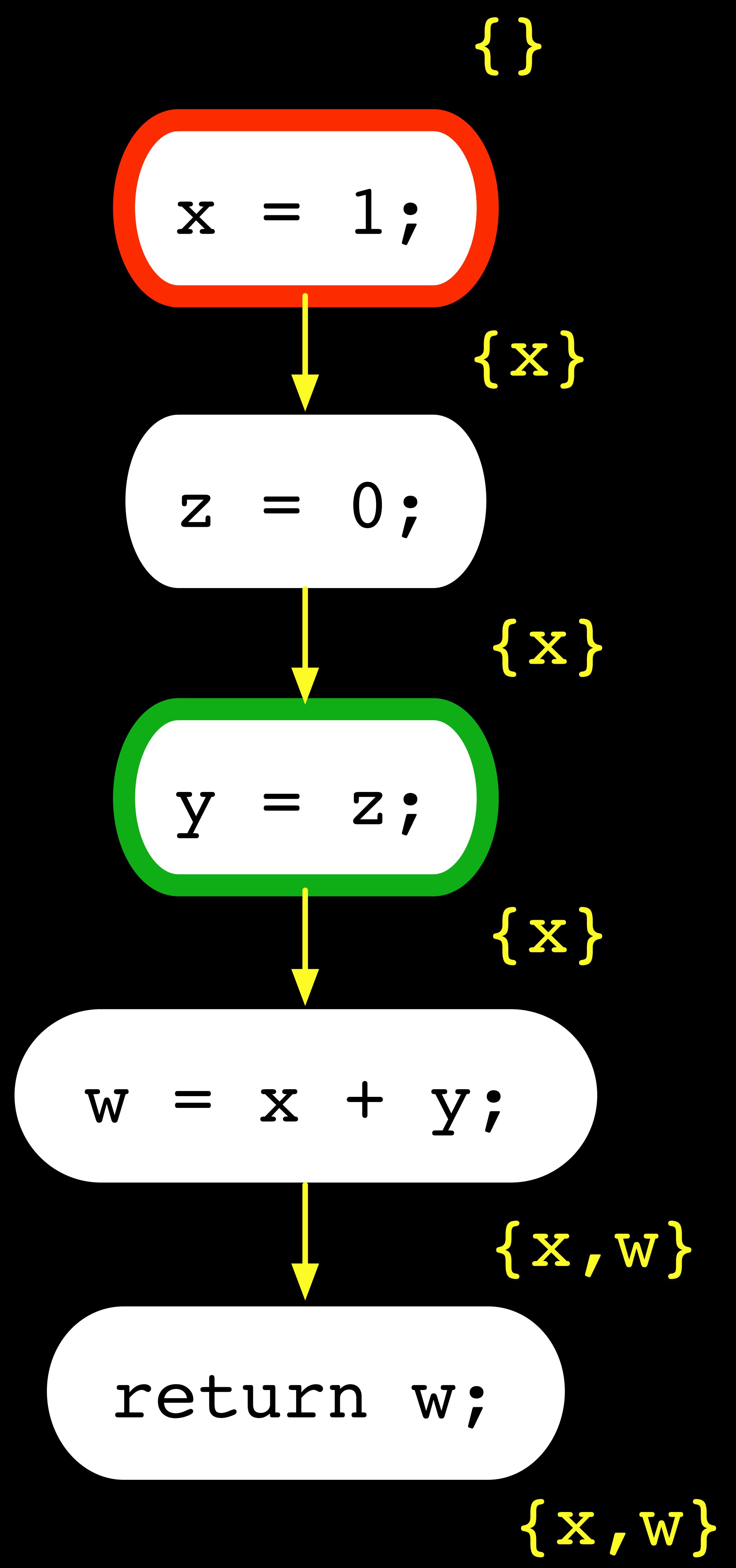
{x,z,y,w}

return w;

{x,z,y,w}

Generating the abstraction

gen

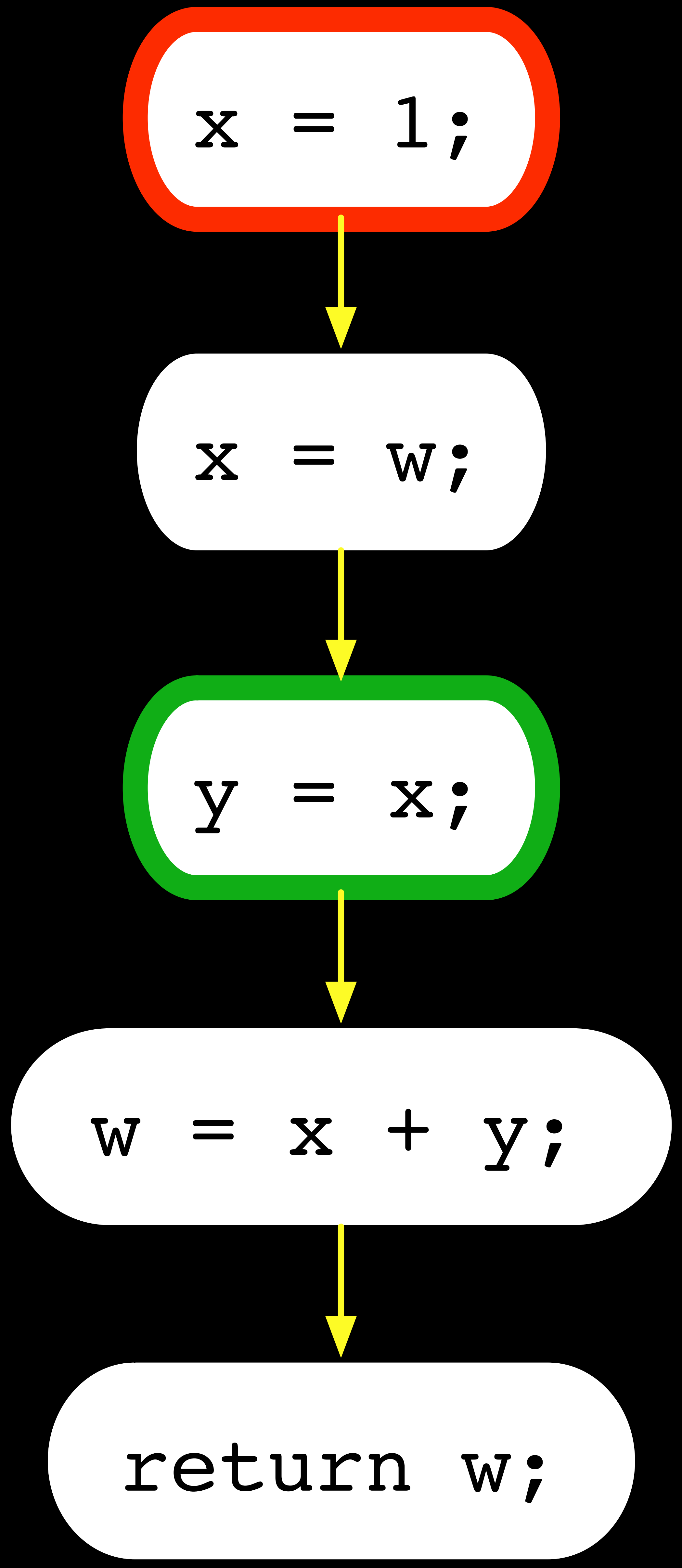


What are the abstractions for this CFG?

What would be the final computed abstraction?

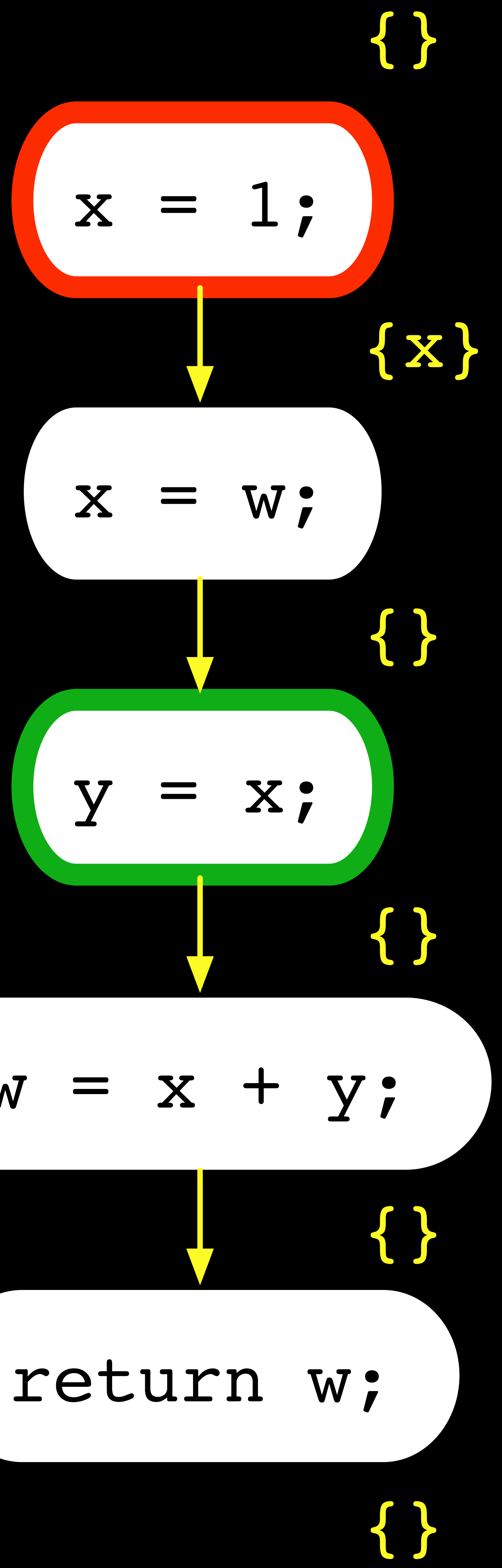
Would a warning be raised?

Consider that the initial abstraction is still { }. Draw all intermediate abstractions.



Killing the abstraction

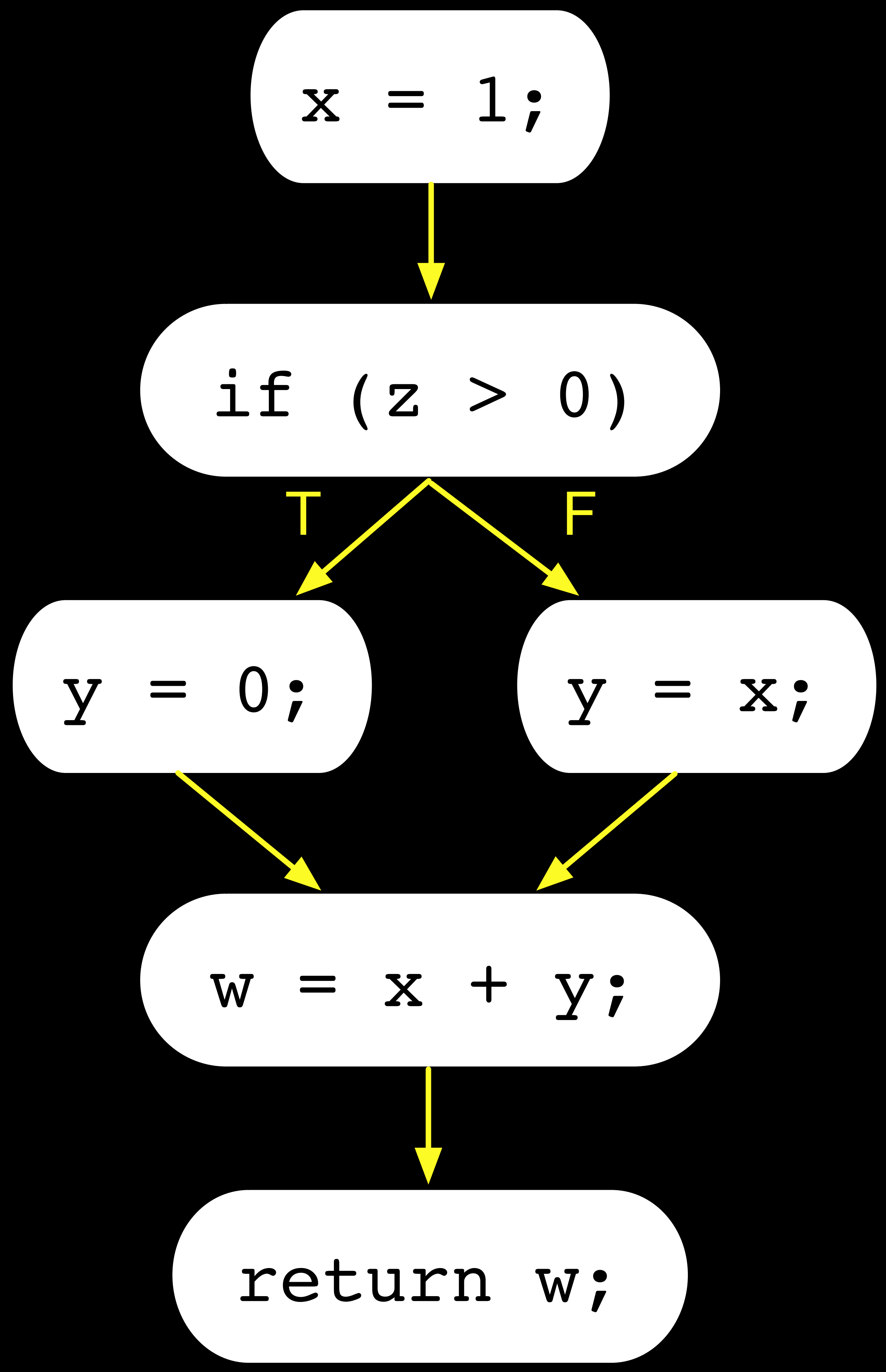
kill



Conservative analysis

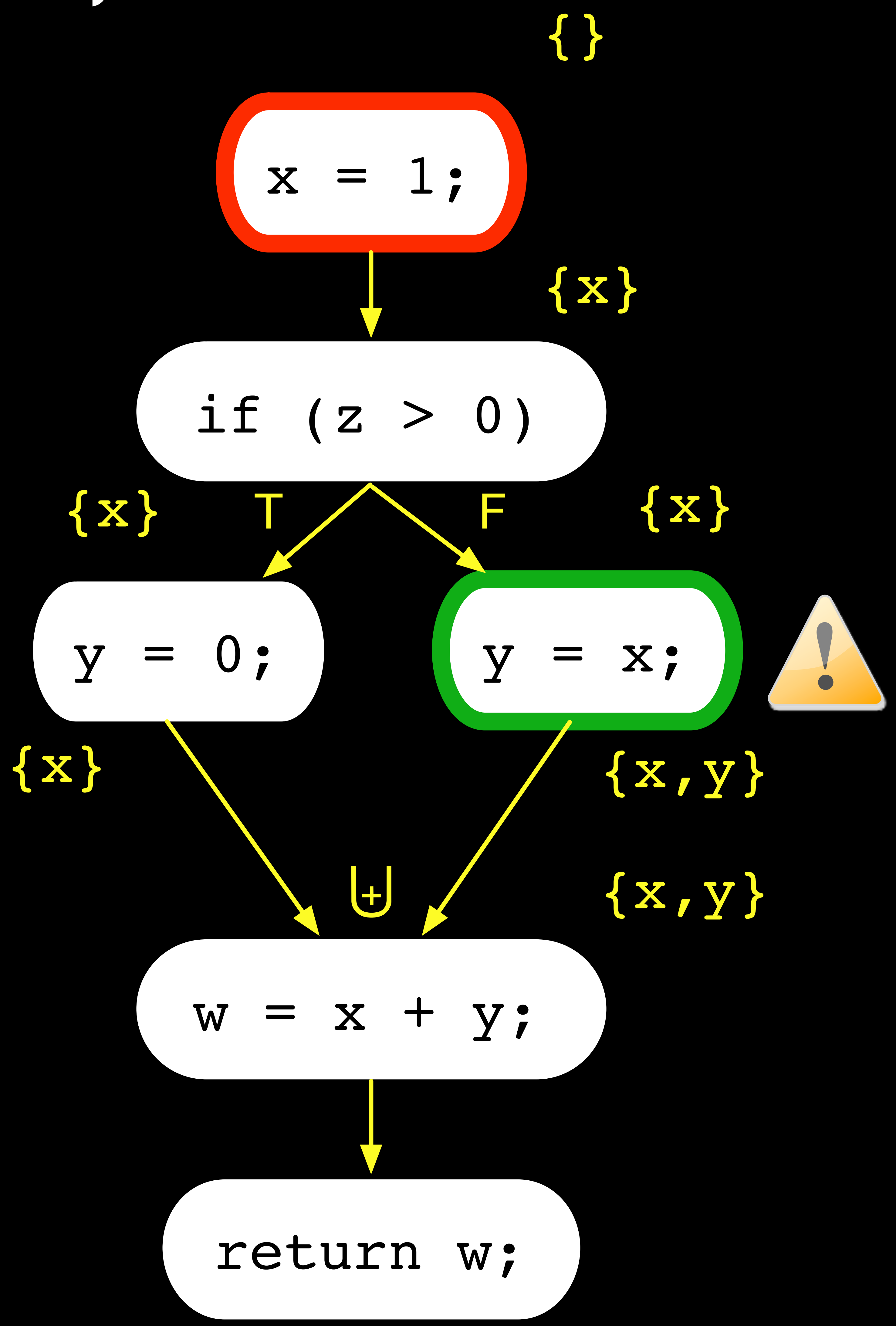
will y be tainted? leads to false positives!

```
x = 1;
if (z > 0) {
   y = 0;
} else {
   y = x;
}
w = x + y;
return w;
```



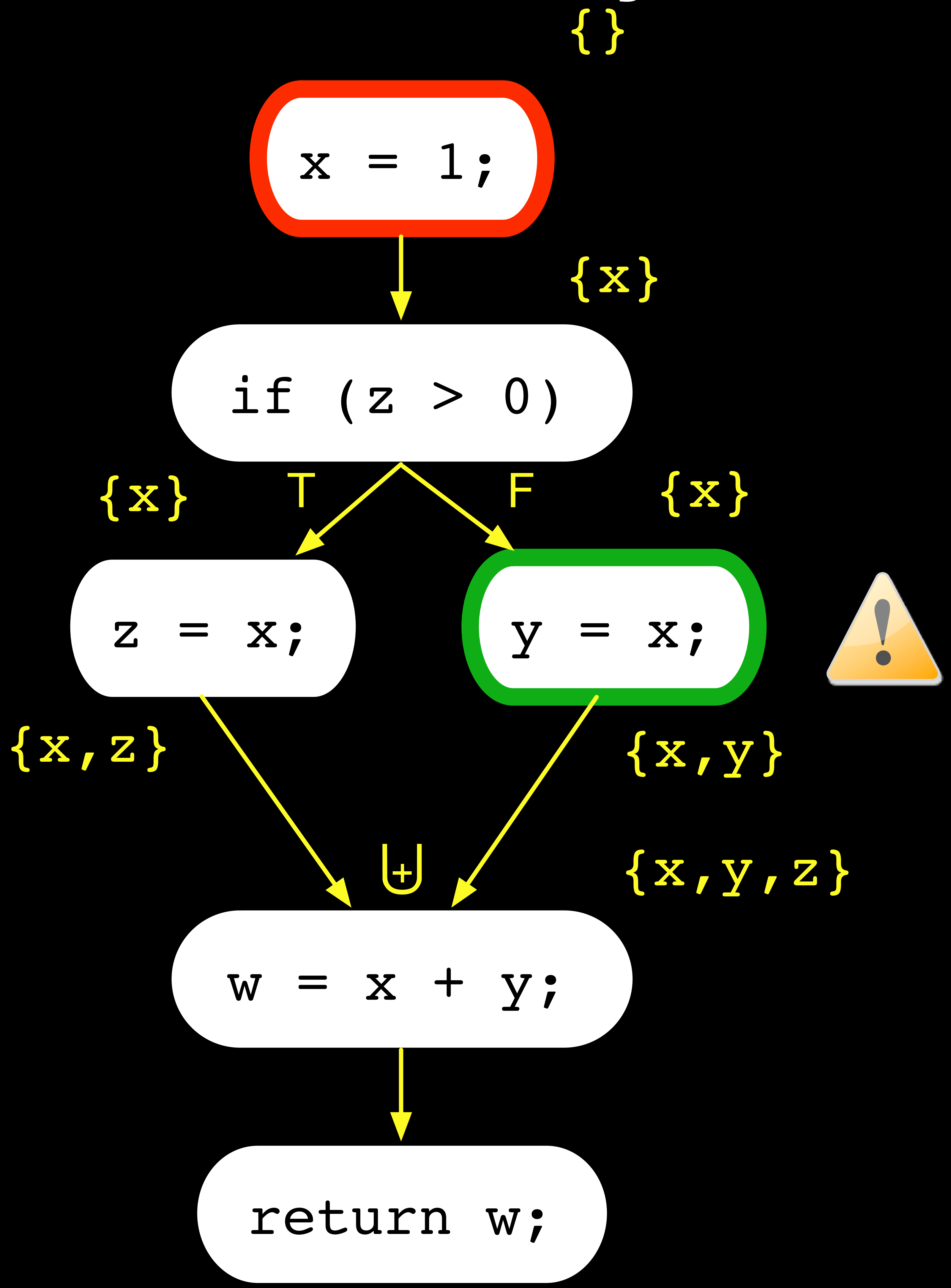
Merging

for taint analysis, the union of the abstractions



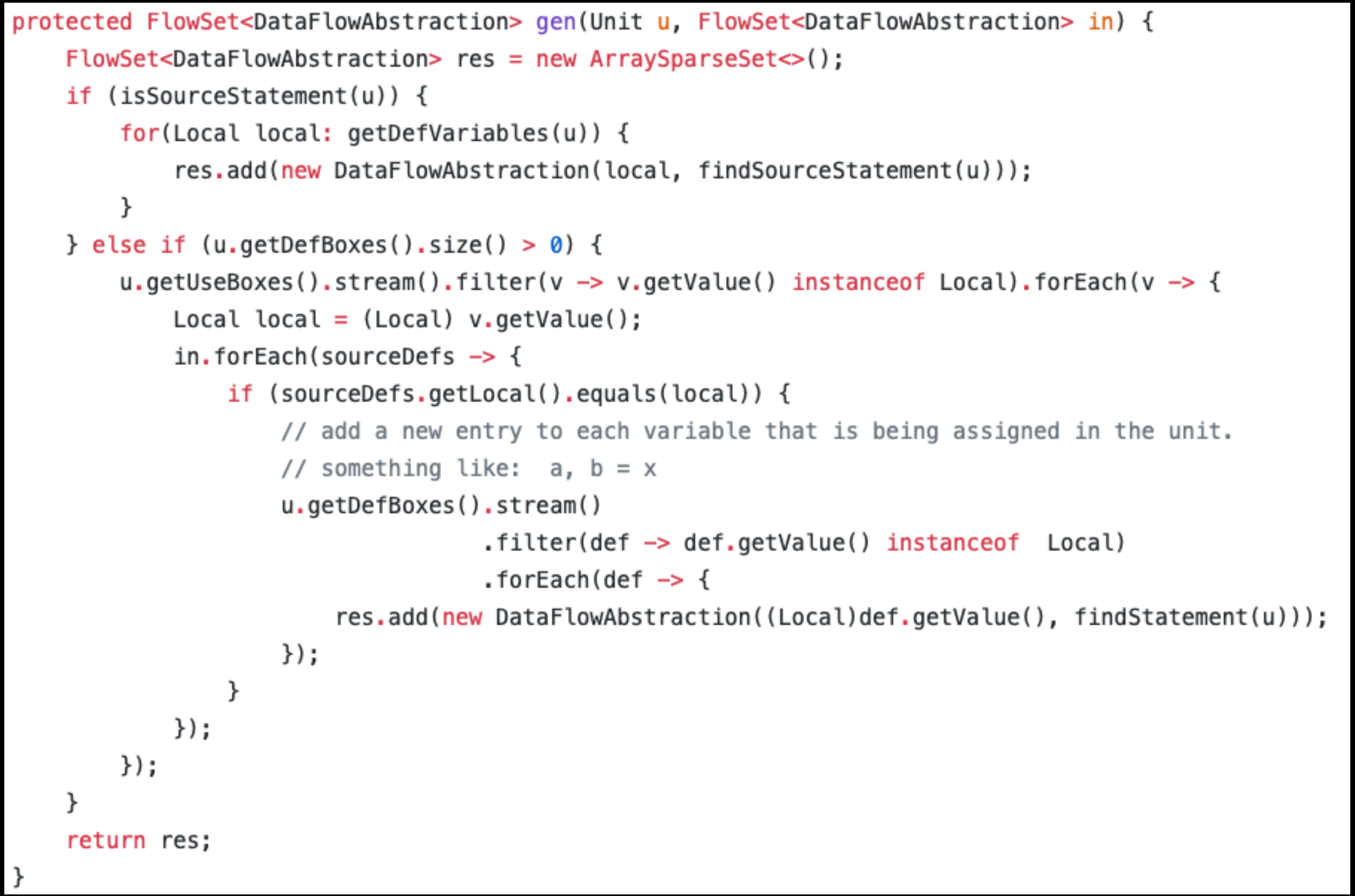
Overestimation

considers that both z and y will be tainted

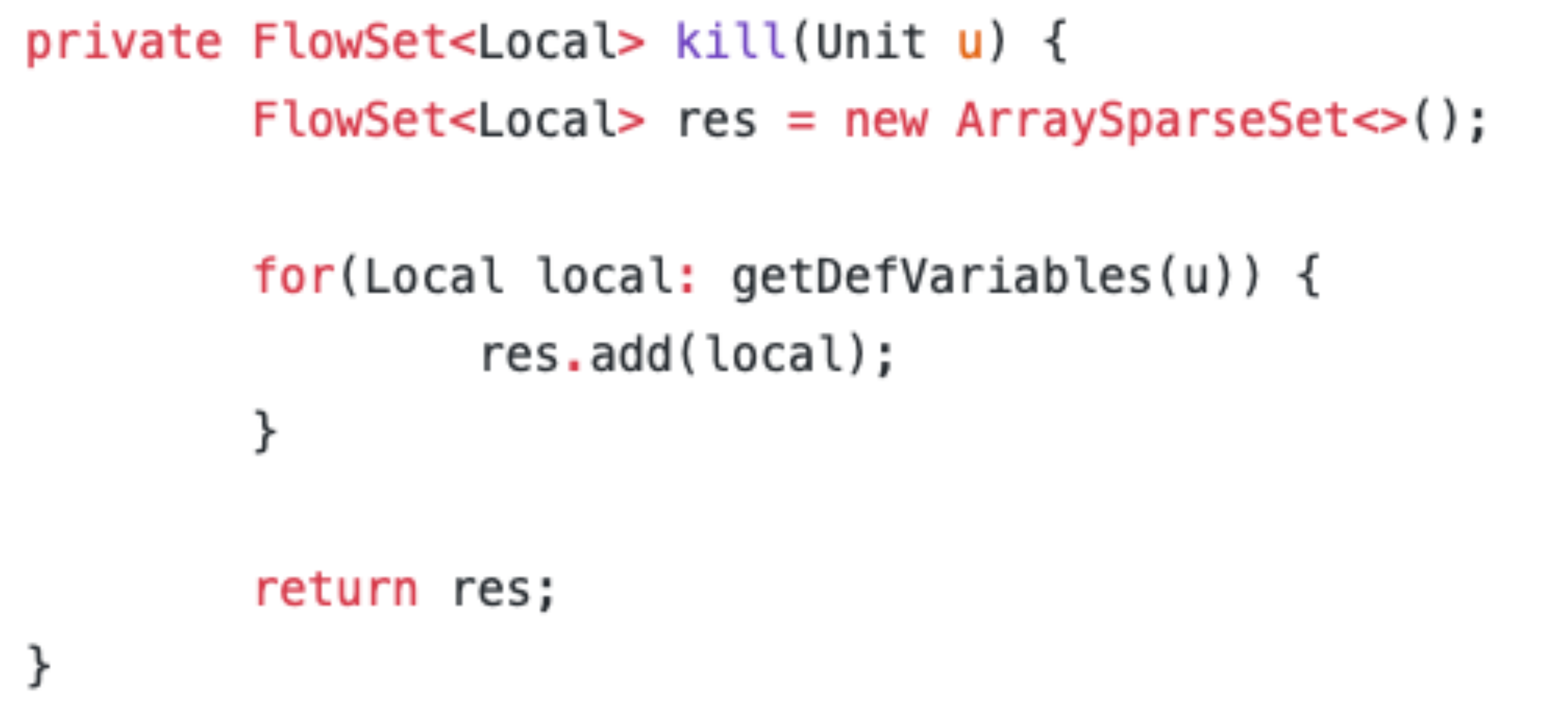


Implementation

SOOT framework



```java
protected FlowSet<DataFlowAbstraction> gen(Unit u, FlowSet<DataFlowAbstraction> in) {
    FlowSet<DataFlowAbstraction> res = new ArraySparseSet<>();
    if (isSourceStatement(u)) {
        for(Local local: getDefVariables(u)) {
            res.add(new DataFlowAbstraction(local, findSourceStatement(u)));
        }
    } else if (u.getDefBoxes().size() > 0) {
        u.getUseBoxes().stream().filter(v -> v.getValue() instanceof Local).forEach(v -> {
            Local local = (Local) v.getValue();
            in.forEach(sourceDefs -> {
                if (sourceDefs.getLocal().equals(local)) {
                    // add a new entry to each variable that is being assigned in the unit.
                    // something like:  a, b = x
                    u.getDefBoxes().stream()
                                   .filter(def -> def.getValue() instanceof  Local)
                                   .forEach(def -> {
                        res.add(new DataFlowAbstraction((Local)def.getValue(), findStatement(u)));
                    });
                }
            });
        });
    }
    return res;
}
```

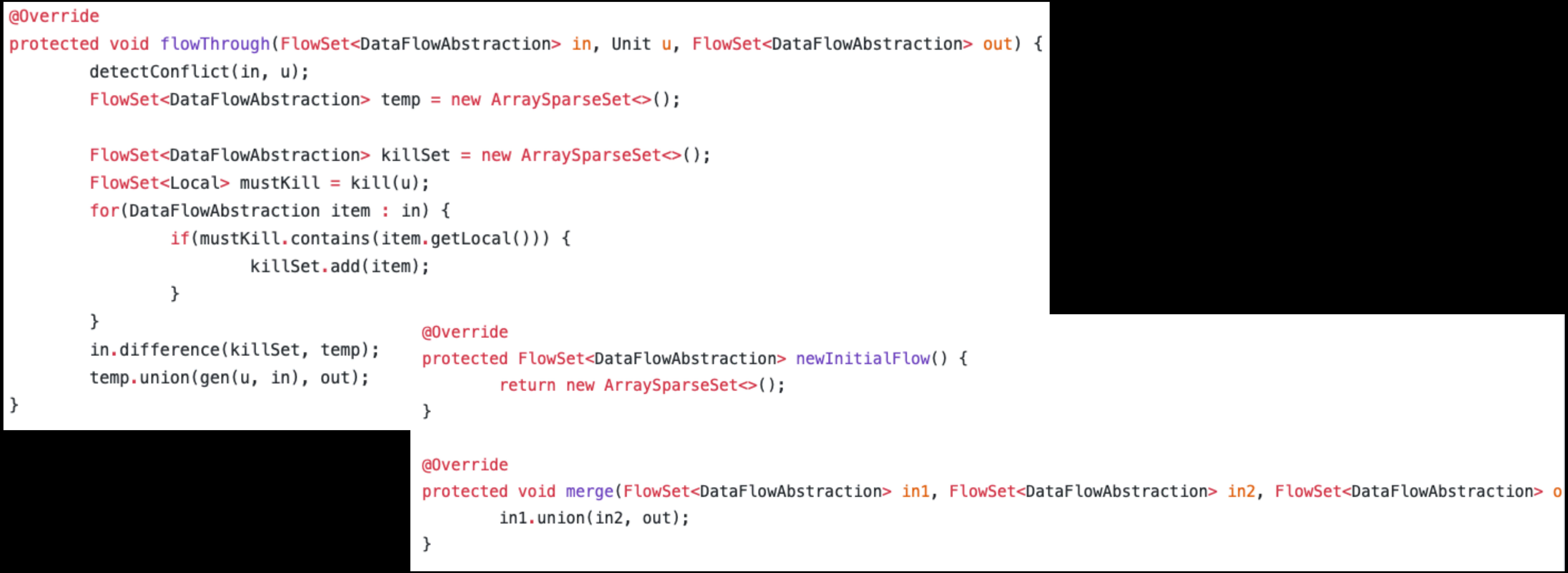


```java
private FlowSet<Local> kill(Unit u) {
        FlowSet<Local> res = new ArraySparseSet<>();

        for(Local local: getDefVariables(u)) {
                res.add(local);
        }

        return res;
}
```

https://github.com/spgroup/conflict-static-analysis

Implementation

SOOT framework

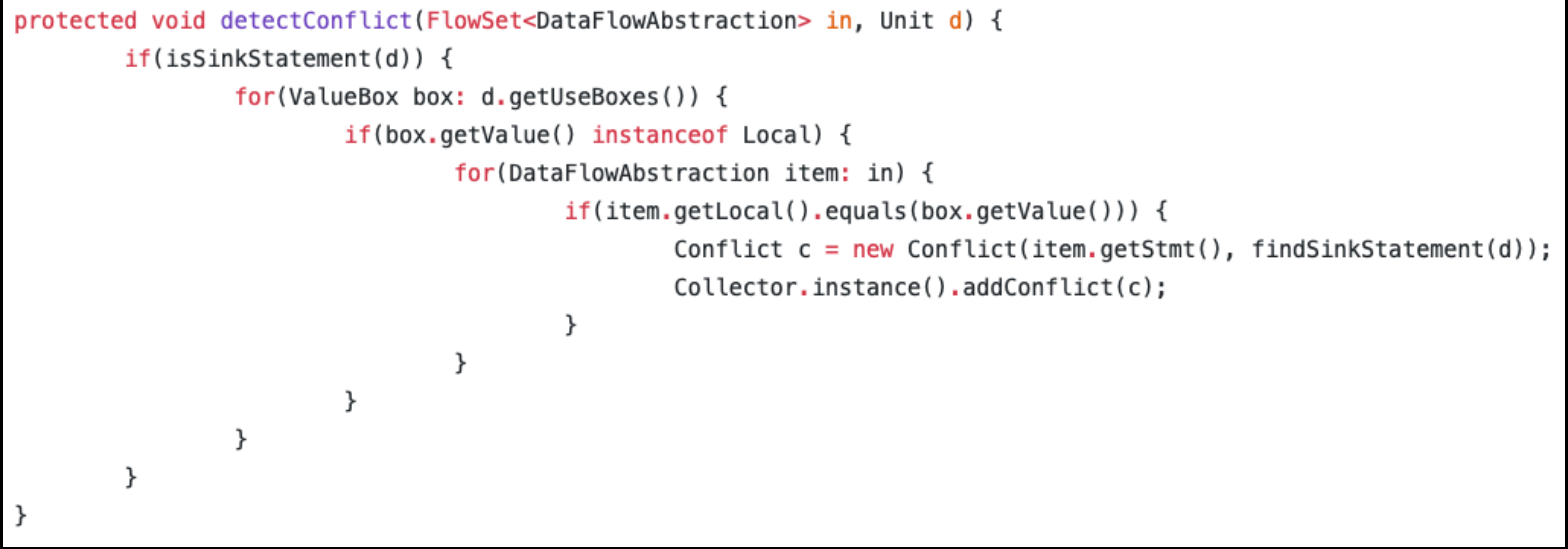


```
@Override
protected void flowThrough(FlowSet<DataFlowAbstraction> in, Unit u, FlowSet<DataFlowAbstraction> out) {
        detectConflict(in, u);
        FlowSet<DataFlowAbstraction> temp = new ArraySparseSet<>();

        FlowSet<DataFlowAbstraction> killSet = new ArraySparseSet<>();
        FlowSet<Local> mustKill = kill(u);
        for(DataFlowAbstraction item : in) {
                if(mustKill.contains(item.getLocal())) {
                        killSet.add(item);
                }
        }
        in.difference(killSet, temp);
        temp.union(gen(u, in), out);
}
```

```
@Override
protected FlowSet<DataFlowAbstraction> newInitialFlow() {
        return new ArraySparseSet<>();
}

@Override
protected void merge(FlowSet<DataFlowAbstraction> in1, FlowSet<DataFlowAbstraction> in2, FlowSet<DataFlowAbstraction> o
        in1.union(in2, out);
}
```

Implementation

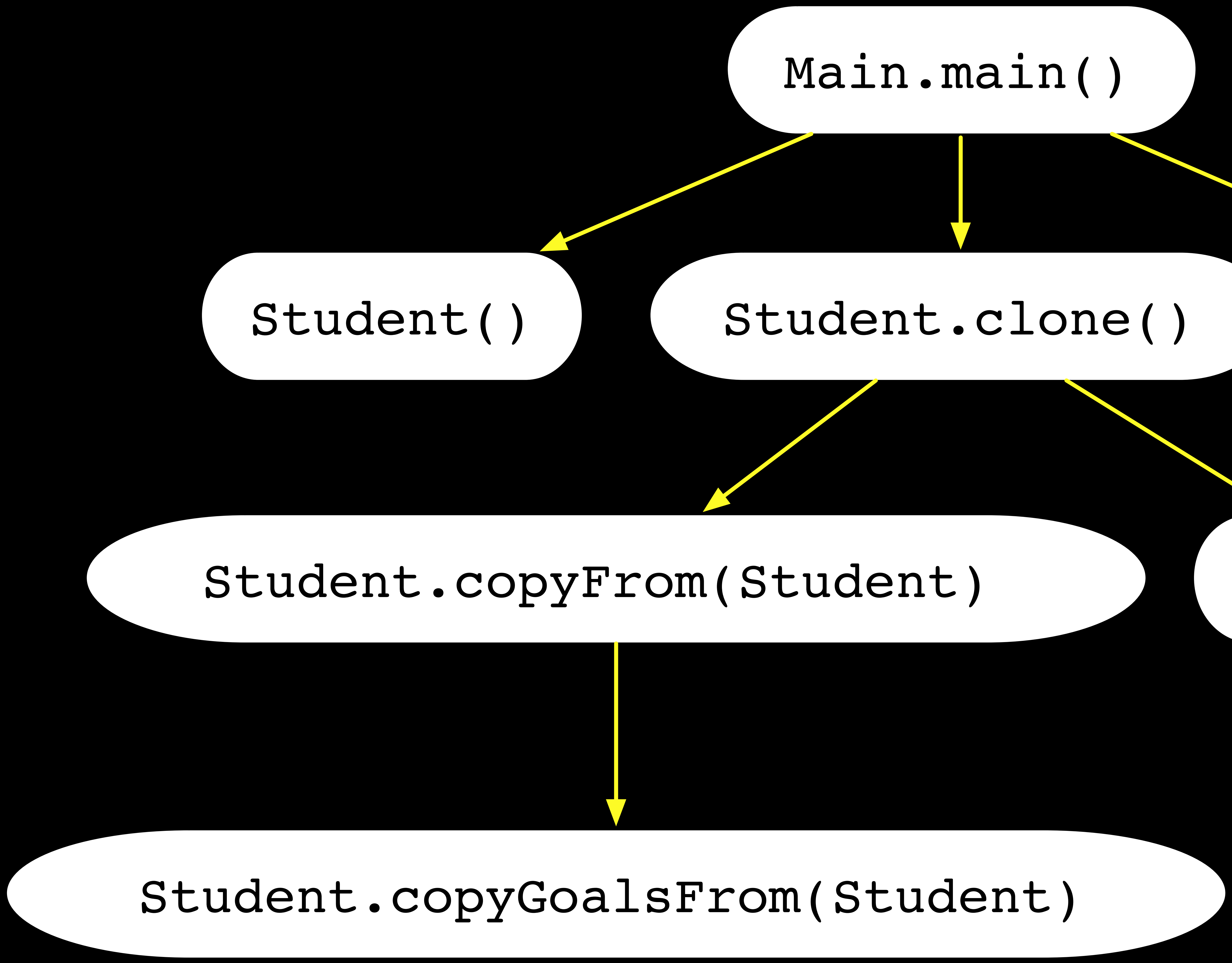
SOOT framework



```
protected void detectConflict(FlowSet<DataFlowAbstraction> in, Unit d) {
        if(isSinkStatement(d)) {
                for(ValueBox box: d.getUseBoxes()) {
                        if(box.getValue() instanceof Local) {
                                for(DataFlowAbstraction item: in) {
                                        if(item.getLocal().equals(box.getValue())) {
                                                Conflict c = new Conflict(item.getStmt(), findSinkStatement(d));
                                                Collector.instance().addConflict(c);
                                        }
                                }
                        }
                }
        }
}
```

Program as a Call Graph

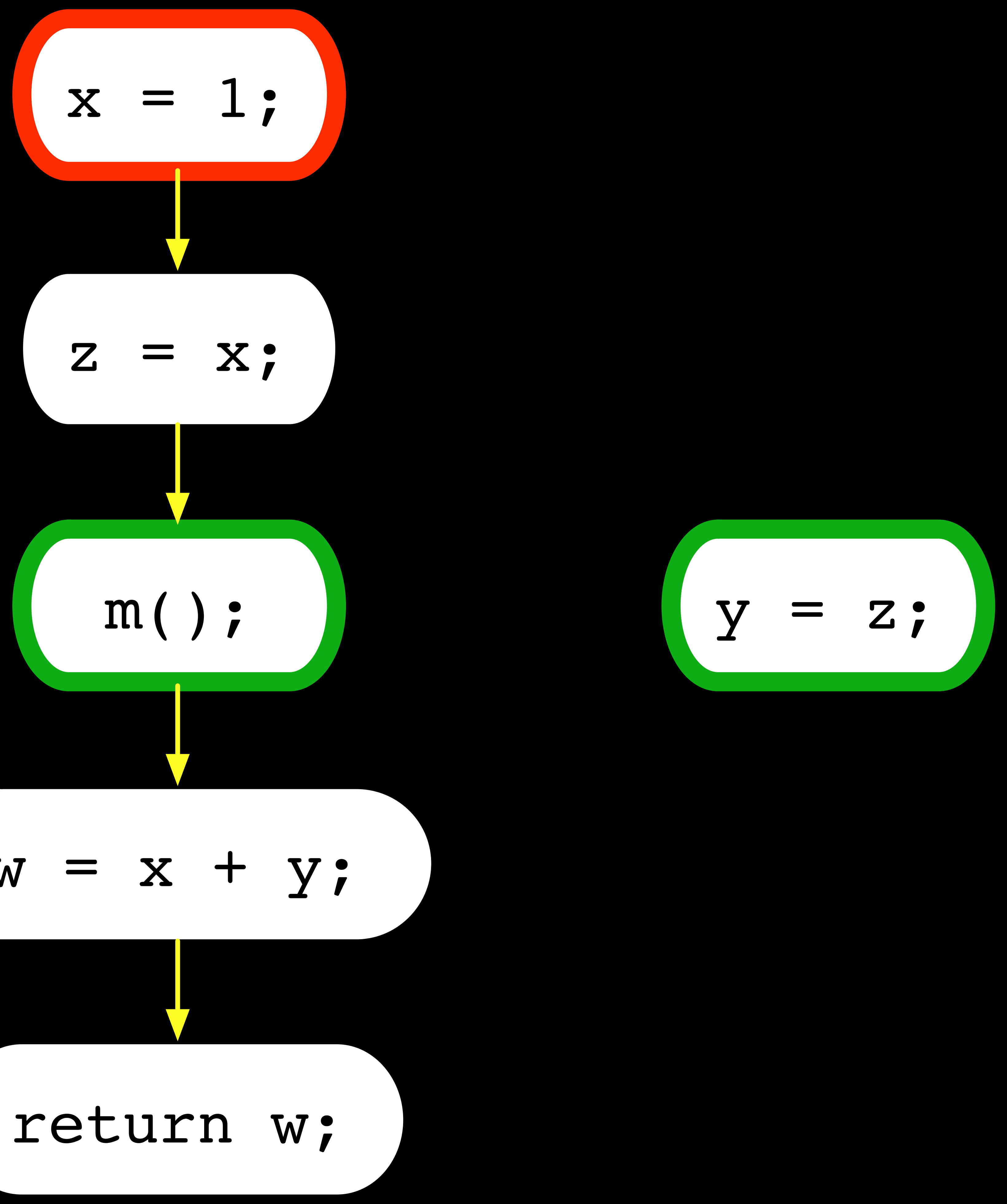
can method clone be reached during the execution of main?



Inter vs intra procedural analysis

how method calls are handled?

```
x = 1;
z = x;
m();
w = x + y;
return w;
```



```
void m() {
  y = z;
}
```

could also have all those representations
with extra information:
types, dataflows, etc.

Static Analysis at Google

cost-benefit tradeoff

- Scalability
  - 100M+ LoC, 10K+ engineers, 10K+ code reviews per day
  - incremental, simple, compositional analysis (>200)
- Usability
  - automatic fixes (>3K per day)
  - reduced “perceived” false positives (distinction between new and old bugs, monitored)
  - integrated to code review (no nightly run, warnings database, no team opening bug reports)

Beyond Code: New Signals for Static Analysis
Caitlin Sadowski. SOAP 20020.

```
public class Test {
```

- Lint
  Java
  1:02 AM, Aug 21

Missing a Javadoc comment.

Please fix
Not useful

```
  public boolean foo() {
    return getString() == "foo".toString();
```

- ErrorProne
  StringEquality
  1:03 AM, Aug 21

String comparison using reference equality instead of value equality
(see http://code.google.com/p/error-prone/wiki/StringEquality)

Please fix

Suggested fix attached: show

Not useful

```
  }

  public String getString() {
    return new String("foo");
  }
}
```

https://www.youtube.com/watch?v=81V2ifmW-4c&feature=youtu.be ~5:06

Beyond Code: New Signals for Static Analysis
Caitlin Sadowski. SOAP 20020.

Static Analysis at Google

beyond code

- Check documentation language
- Check vulnerabilities of imported projects
- Check translation files
- Check binary size (compare before and after change)
- Check conformance of configuration files and source code
- Check if deleted artifact is referenced in documents, etc.

Beyond Code: New Signals for Static Analysis
Caitlin Sadowski. SOAP 20020.

Dynamic analysis, with testing, is an alternative approach for some of the properties

Analysis approximations, accuracy	False positives	False negatives
Static analysis	X (approximations)	x (reflection)
Dynamic analysis	x (flakiness)	X (uncovered inputs)

Software Engineering

Paulo Borba
Informatics Center
Federal University of Pernambuco

pauloborba.cin.ufpe.br