Foundations of Software Engineering

Taint Analysis

Miguel Velez

institute for SOFTWARE RESEARCH

Learning goals

- Define taint analysis.
- Compare the dynamic and static approaches, as well as their benefits and limitations.
- Apply the analysis to several examples
- Understand how dynamic and static analyses can be combined to overcome the limitations of each other.

DYNAMIC ANALYSIS

Dynamic Analysis

- Learn about program's properties by executing it.
- Examine program state throughout/ after execution by gathering additional information.

Performance Analysis

How would you learn about method execution time?

```
void main(a) {
  if(a > 0) {
    sleep_ms(a);
  else {
    sleep_ms(1000);
  }
}
```

```
void main(a) {
  start("main");
  if(a > 0) {
    sleep_ms(a);
  else {
    sleep_ms(1000);
  }
  end("main");
}
```

Benefits

Benefits

- Analyzes the state of the program in a runtime environment.
- If the property we are looking for is found, we can be sure that it exists.
- Validate static analysis findings.

Limitations

Limitations

- Input dependent
- Cannot explore all paths
- Cost of tracking information
- Heisenbuggy behavior

STATIC ANALYSIS

Static Analysis

- Learn about program's properties without executing it.
- Systematic examination of an abstraction of a program

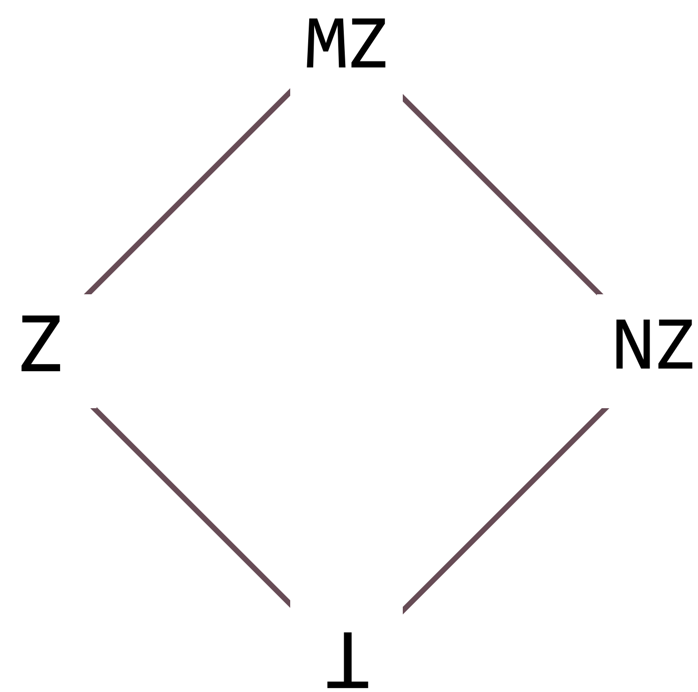
institute for SOFTWARE RESEARCH

Zero Analysis

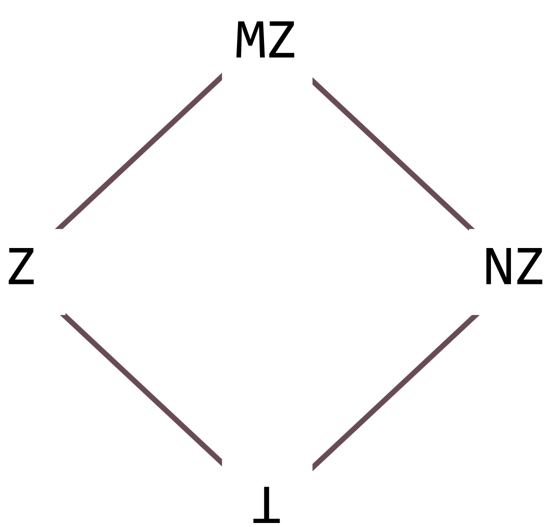
How would you learn if you divide by 0?

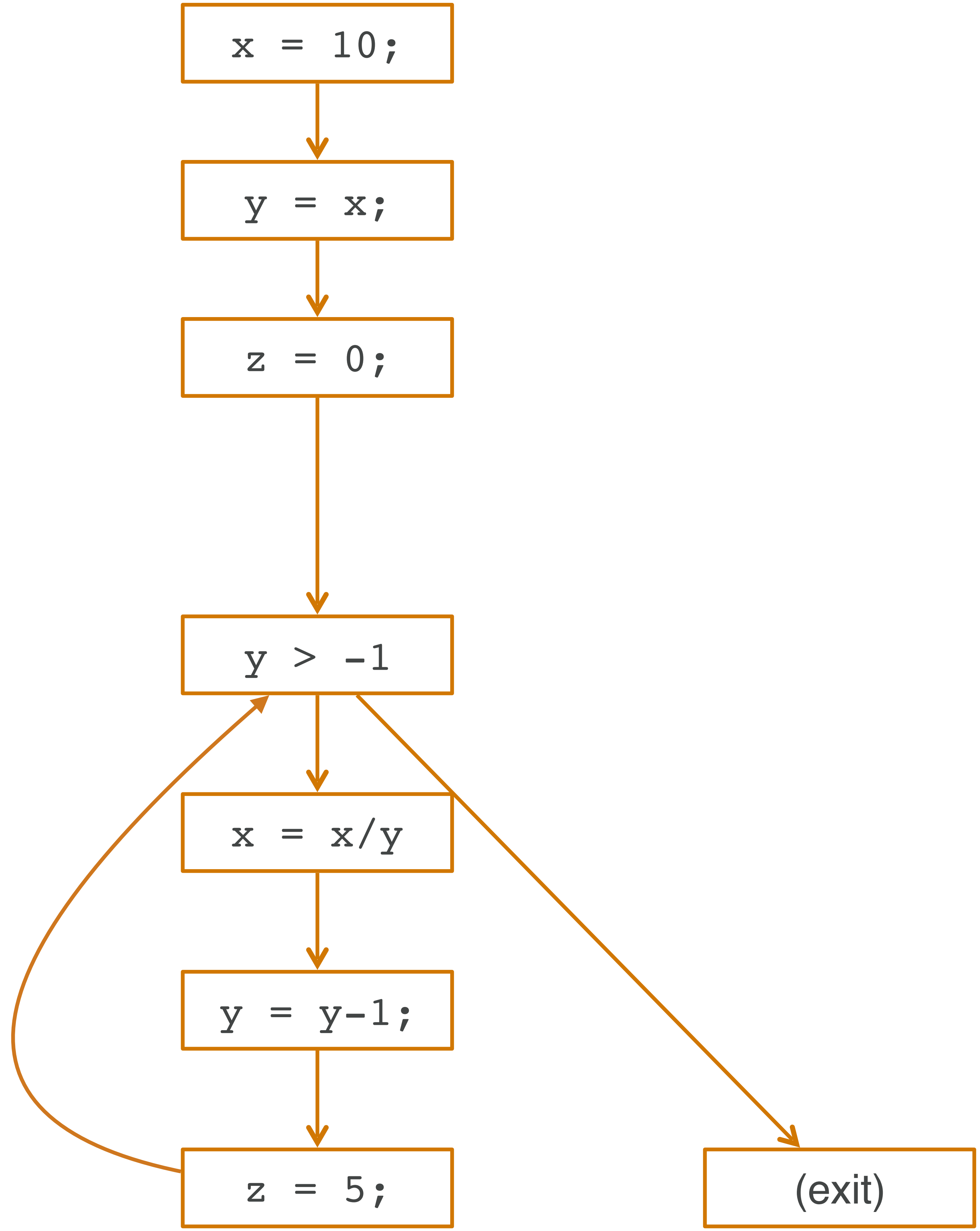
```
x = 10;
y = x;
z = 0;
while(y > -1) {
  x = x/y;
  y = y-1;
  Z = 5;
}
```

```
x = 10;
y = x;
z = 0;
while(y > -1) {
  x = x/y;
  y = y-1;
  Z = 5;
}
```

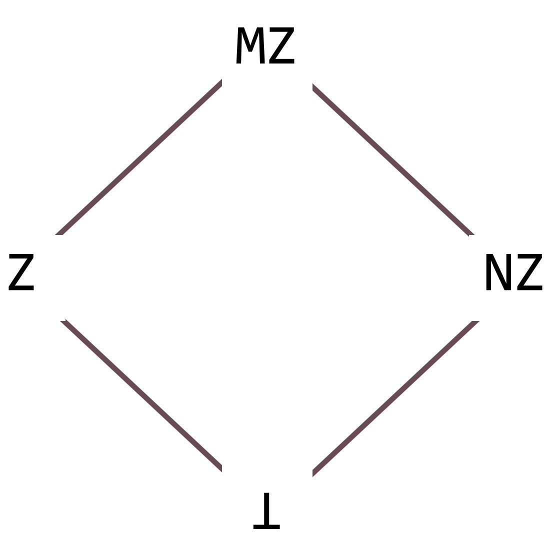


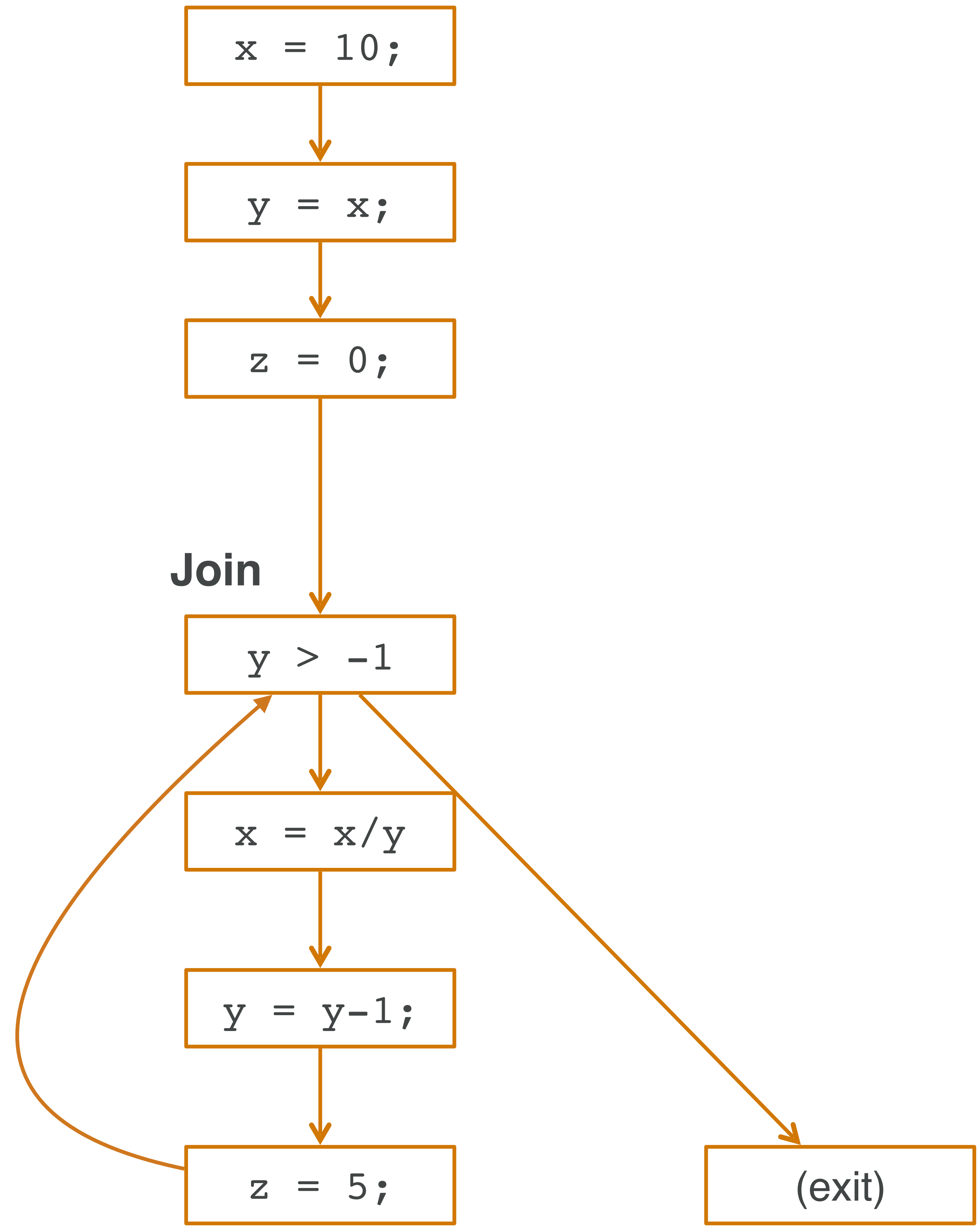
```
x = 10;
y = x;
z = 0;
while(y > -1) {
  x = x/y;
  y = y-1;
  Z = 5;
}
```



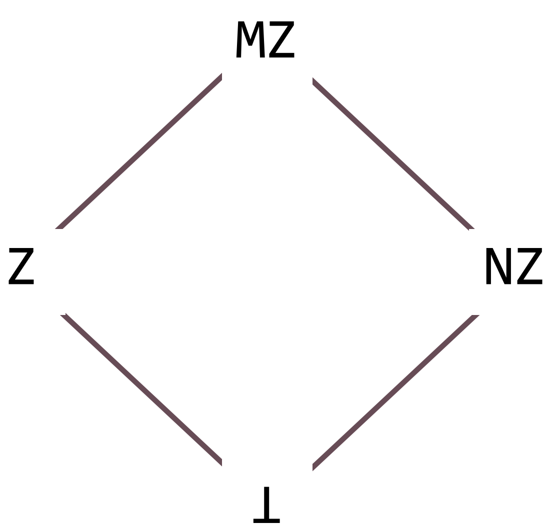


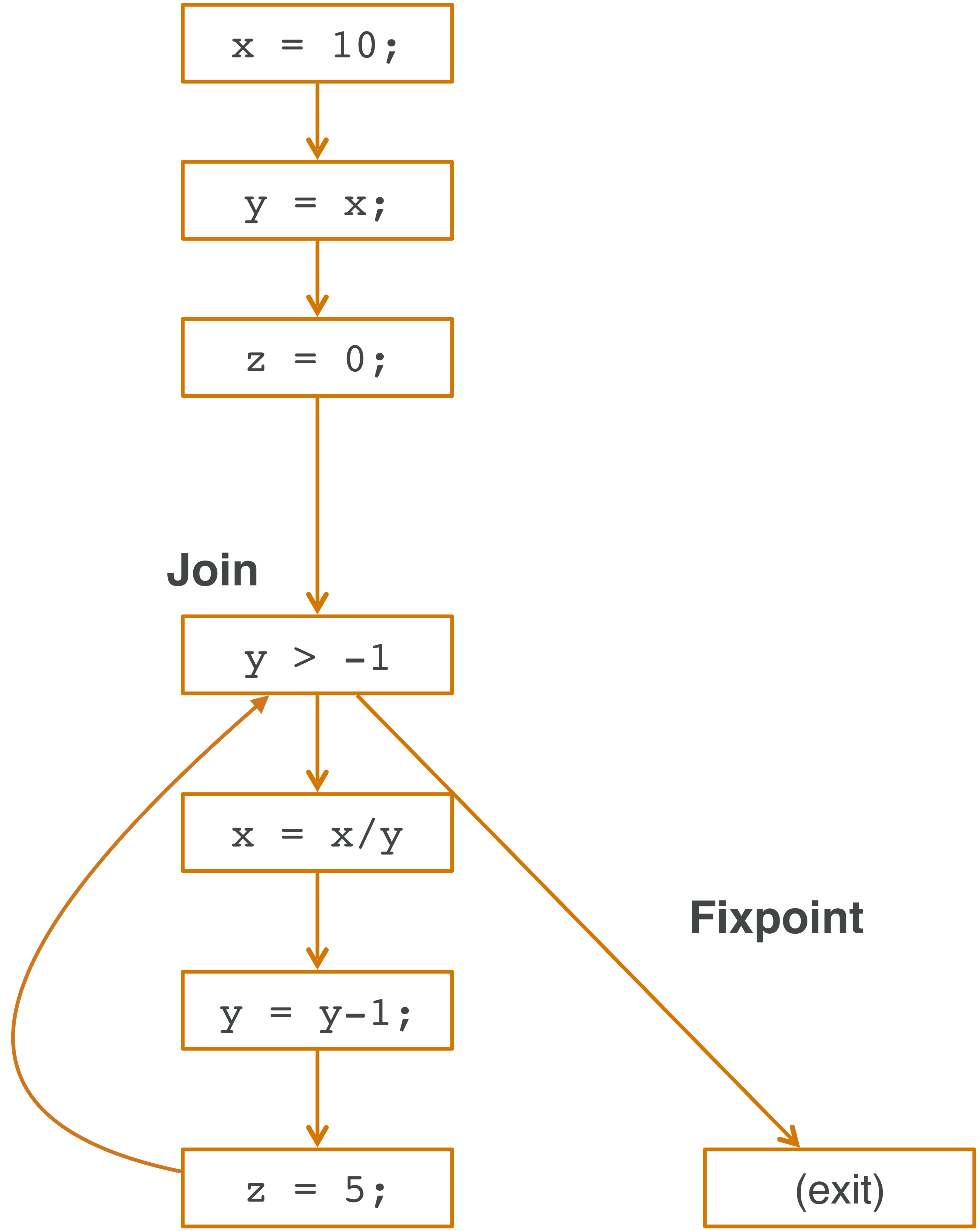
```
x = 10;
y = x;
z = 0;
while(y > -1) {
  x = x/y;
  y = y-1;
  Z = 5;
}
```



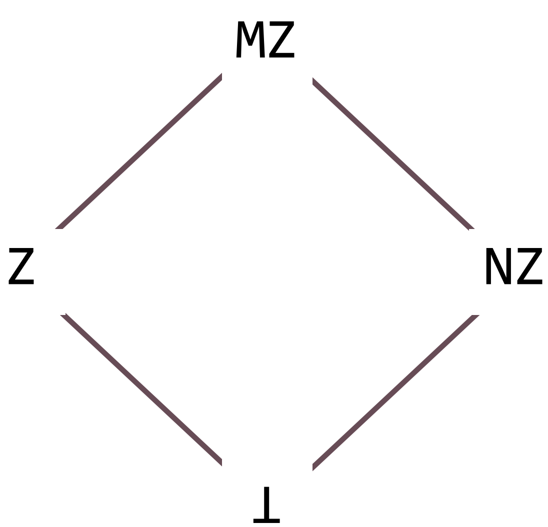


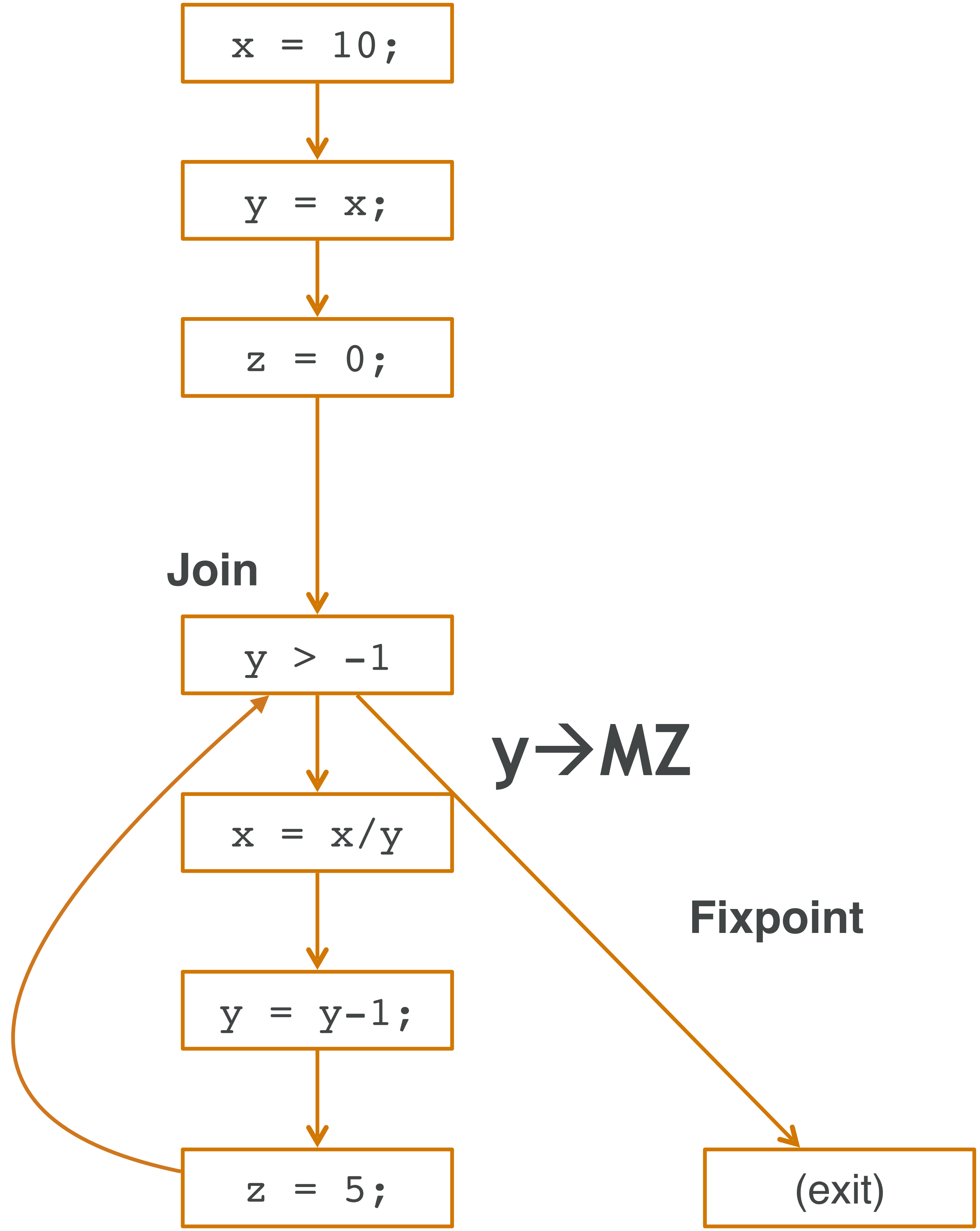
```
x = 10;
y = x;
z = 0;
while(y > -1) {
  x = x/y;
  y = y-1;
  Z = 5;
}
```





```
x = 10;
y = x;
z = 0;
while(y > -1) {
  x = x/y;
  y = y-1;
  Z = 5;
}
```





Benefits

Benefits

- Analyzes all possible executions of the program.
- Pinpoint in code where issues occur.
- Detects issues in the early stages of development.

Limitations

Limitations

- Rice's Theorem: Every static analysis is necessarily incomplete or unsound or undecidable (or multiple of these).
- Difficult to track runtime properties.
- Can analyze parts of the program that are never executed.

TAINT ANALYSIS

Taint Analysis

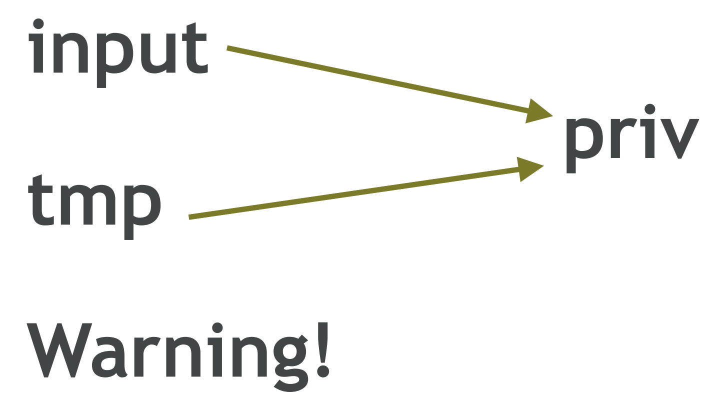
- Information flow analysis.
- Used in the security domain.
- Tracking how private information flows through the program and if it is leaked to public observers.

Example

```
input = get_input();
tmp = "select …" + input;
query(tmp);
log(tmp);
```

Example

```
input = get_input();
tmp = “select …” + input;
query(tmp);
log(tmp);
```



Terminology

- Sources
  - Private data of interest

- Sinks
  - Locations of interest
  - Check taints of incoming information
  - Determines if there is a leak in the program.

Example

```
input = get_input();
tmp = “select …” + input;
query(tmp);
log(tmp);
```

Example

```
input = Source();
tmp = “select …” + input;
Sink(tmp);
log(tmp);
```

Example

```
1. input = Source();
2. tmp = "select …" + input;
3. Sink(tmp);
4. log(tmp);
```

input

tmp

priv

Warning!

Example

```
input = Source();
tmp = “select …” + input;
tmp = encode(tmp)
Sink(tmp);
log(tmp);
```

input → priv

tmp → …

OK

DYNAMIC TAINT ANALYSIS

Dynamic Taint Analysis

- Track what are the taints that are influencing the values of the program.

Example

```
x = get_input();
y = 1;
z = x;
w = y + z;
print(w);
```

Example

```
x = Source(0);
y = 1;
z = x;
w = y + z;
Sink(w);
```

Example

```
1. x = Source(0);
2. y = 1;
3. z = x;
4. w = y + z;
5. Sink(w);
```

$x \longrightarrow 0 \rightarrow T$

Example

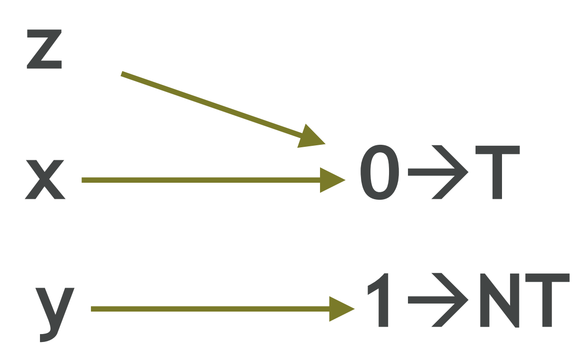
```
1. x = Source(0);
2. y = 1;
3. z = x;
4. w = y + z;
5. Sink(w);
```

x → 0→T

y → 1→NT

Example

```
x = Source(0);
y = 1;
z = x;
w = y + z;
Sink(w);
```



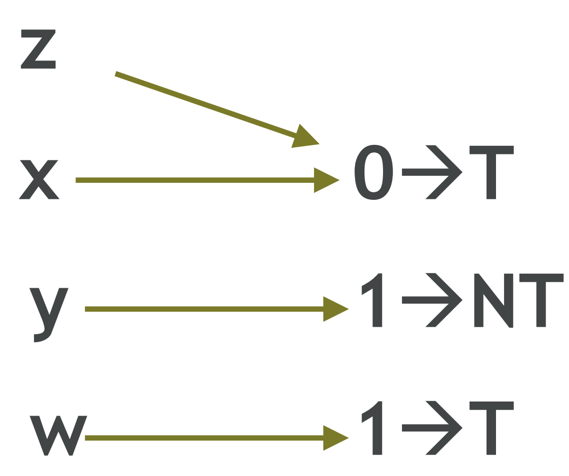
Example

```
1. x = Source(0);
2. y = 1;
3. z = x;
4. w = y + z;
5. Sink(w);
```

z → 0→T
x → 0→T
y → 1→NT
w → 1→T

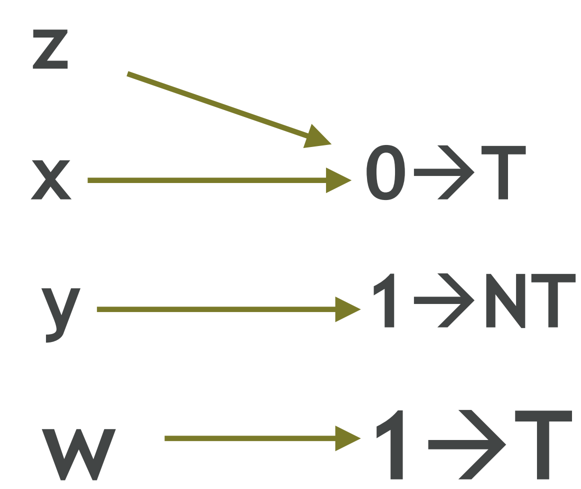
Example

```
1. x = Source(0);
2. y = 1;
3. z = x;
4. w = y + z;
5. Sink(w);
```



Example

```
1. x = Source(0);
2. y = 1;
3. z = x;
4. w = y + z;
5. Sink(w);
```



Leak in the program!

Is there a leak? Why? Why not?

```
x = Source(0);
y = x;
if(y == 0) {
  z = 2
}
else {
  z = 1
}
Sink(z);
```

Implicit Flows

- Tainted data affects the value of another variable indirectly.
- Needed for sound analysis.

Implicit Flows

```
x = Source(0);
y = x;
if(y == 0) {
   z = 2
}
else {
   z = 1
}
Sink(z);
```

Explicit information flow

Implicit information flow

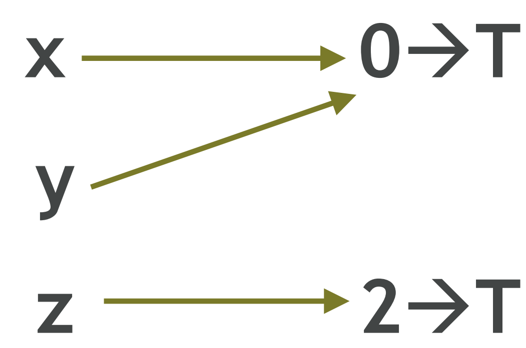
Implicit Flows

```
x = Source(0);
y = x;
if(y == 0) {
  z = 2
}
else {
  z = 1
}
Sink(z);
```

$$\begin{array}{l} x \longrightarrow 0 \to T \\ y \nearrow \end{array}$$

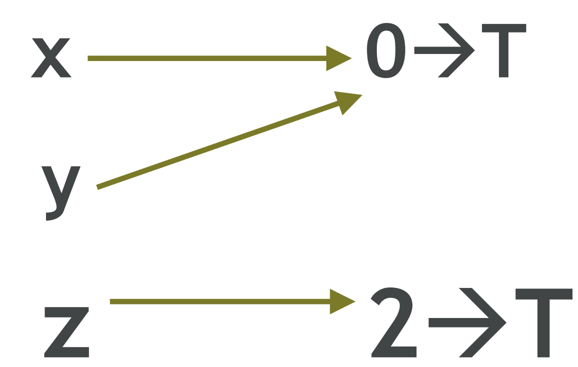
Implicit Flows

```
x = Source(0);
y = x;
if(y == 0) {
  z = 2
}
else {
  z = 1
}
Sink(z);
```



Implicit Flows

```
x = Source(0);
y = x;
if(y == 0) {
   z = 2
}
else {
   z = 1
}
Sink(z);
```



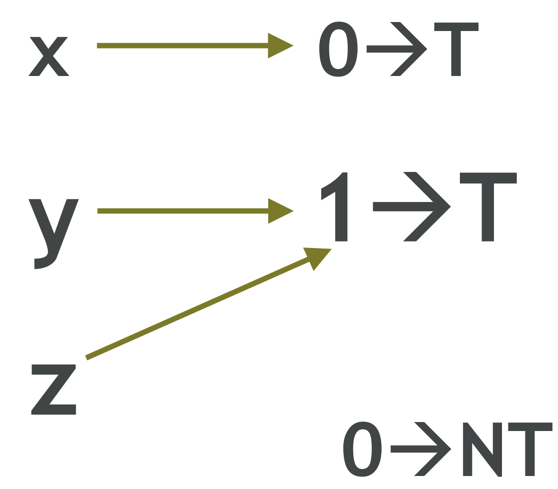
Leak in the program!

Try it yourself

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

Try it yourself

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



Leaks in the program!

Limits of Dynamic Analysis

- Results are input dependent.
- Implicit flows needed for sound analysis, but difficult to track*.

- *Stayed tuned for the end of lecture.

STATIC TAINT ANALYSIS

Static Taint Analysis

- Track, at each instruction, what are the taints that are influencing the variables of the program.

Example

```
x = Source(i);
y = 1;
z = x;
w = y + z;
Sink(w);
```

Example

`1. x = Source(i);`	x→T
`2. y = 1;`	x→T
`3. z = x;`	x→T, z→T
`4. w = y + z;`	x→T, z→T, w→T
`5. Sink(w);`	x→T, z→T, w→T

Example

`x = Source(i);`	x→T
`y = 1;`	x→T
`z = x;`	x→T, z→T
`w = y + z;`	x→T, z→T, w→T
`Sink(w);`	x→T, z→T, **w→T**

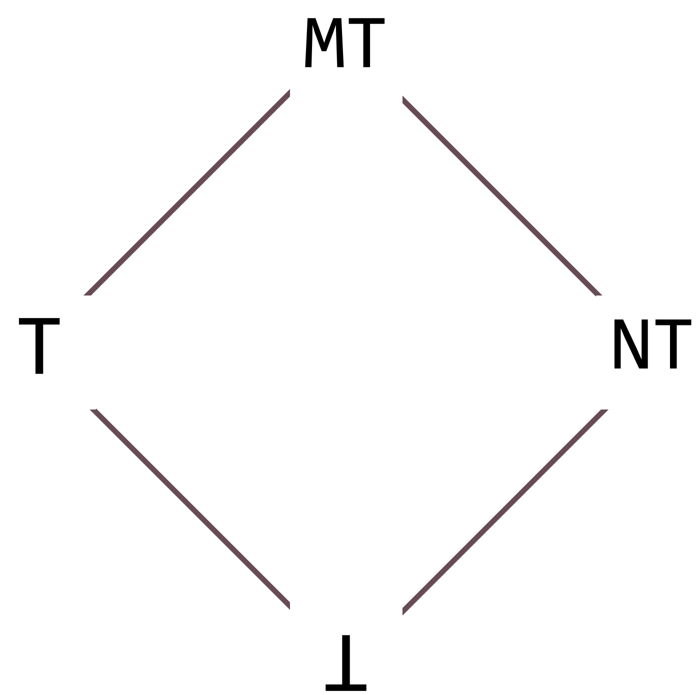
Leak in the program!

institute for SOFTWARE RESEARCH

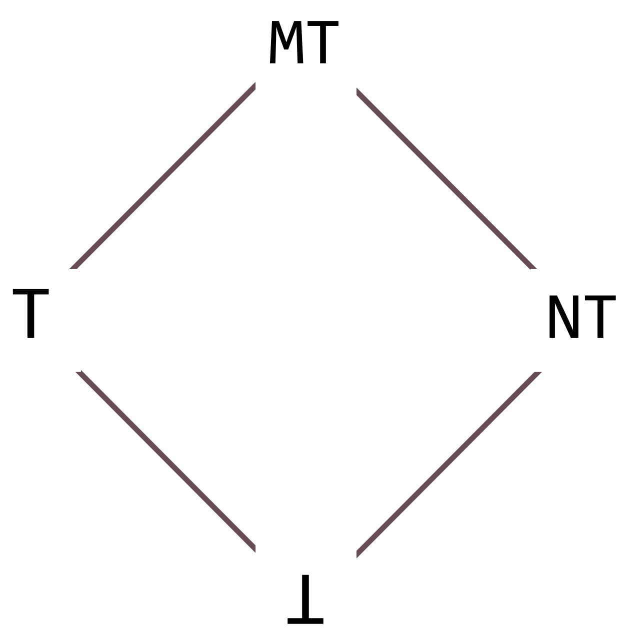
Implicit Flows

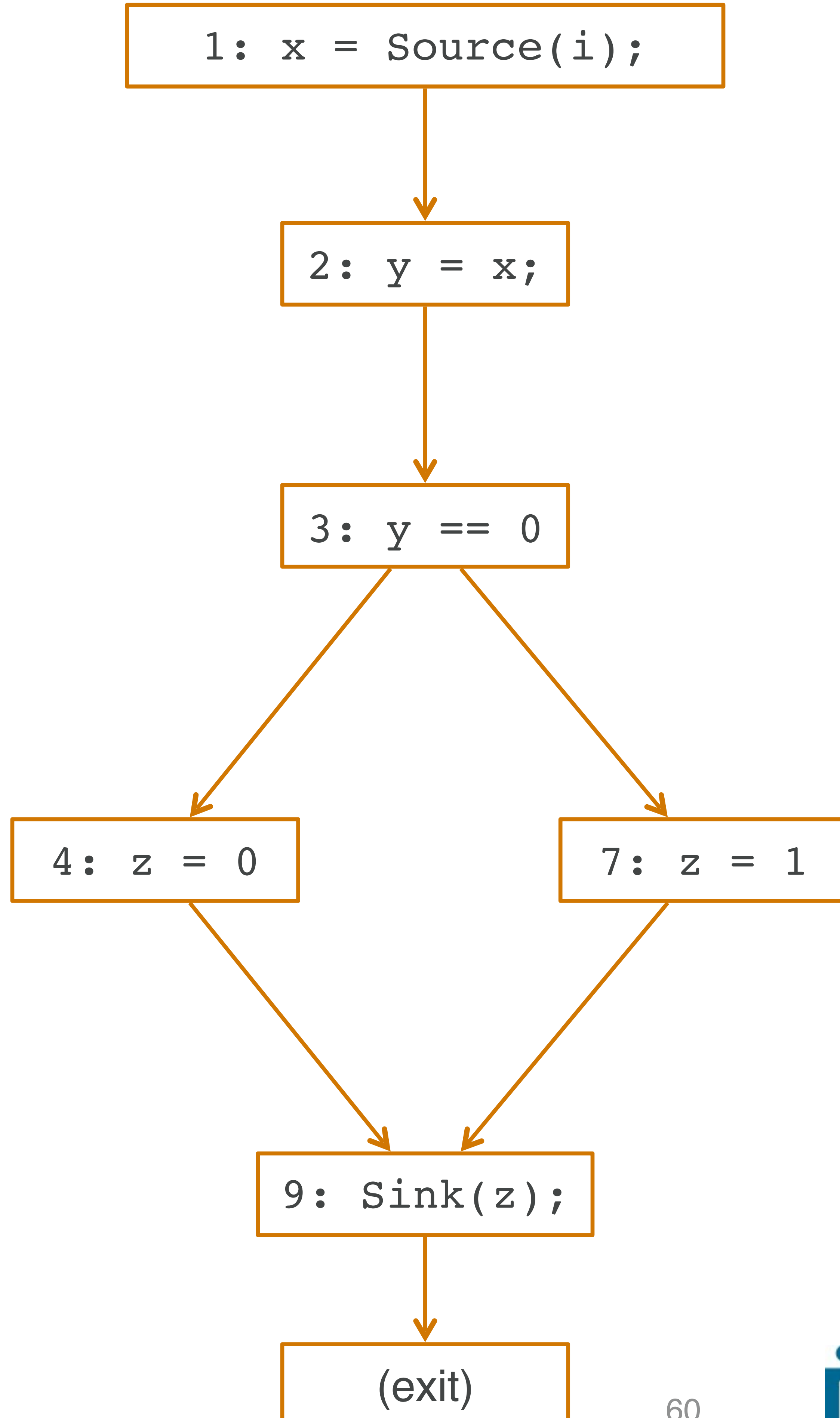
```
x = Source(i);
y = x;
if(y == 0) {
  z = 0
}
else {
  z = 1
}
Sink(z);
```

```
x = Source(i, “A”);
y = x;
if(y == 0) {
  z = 0
}
else {
  z = 1
}
Sink(z);
```



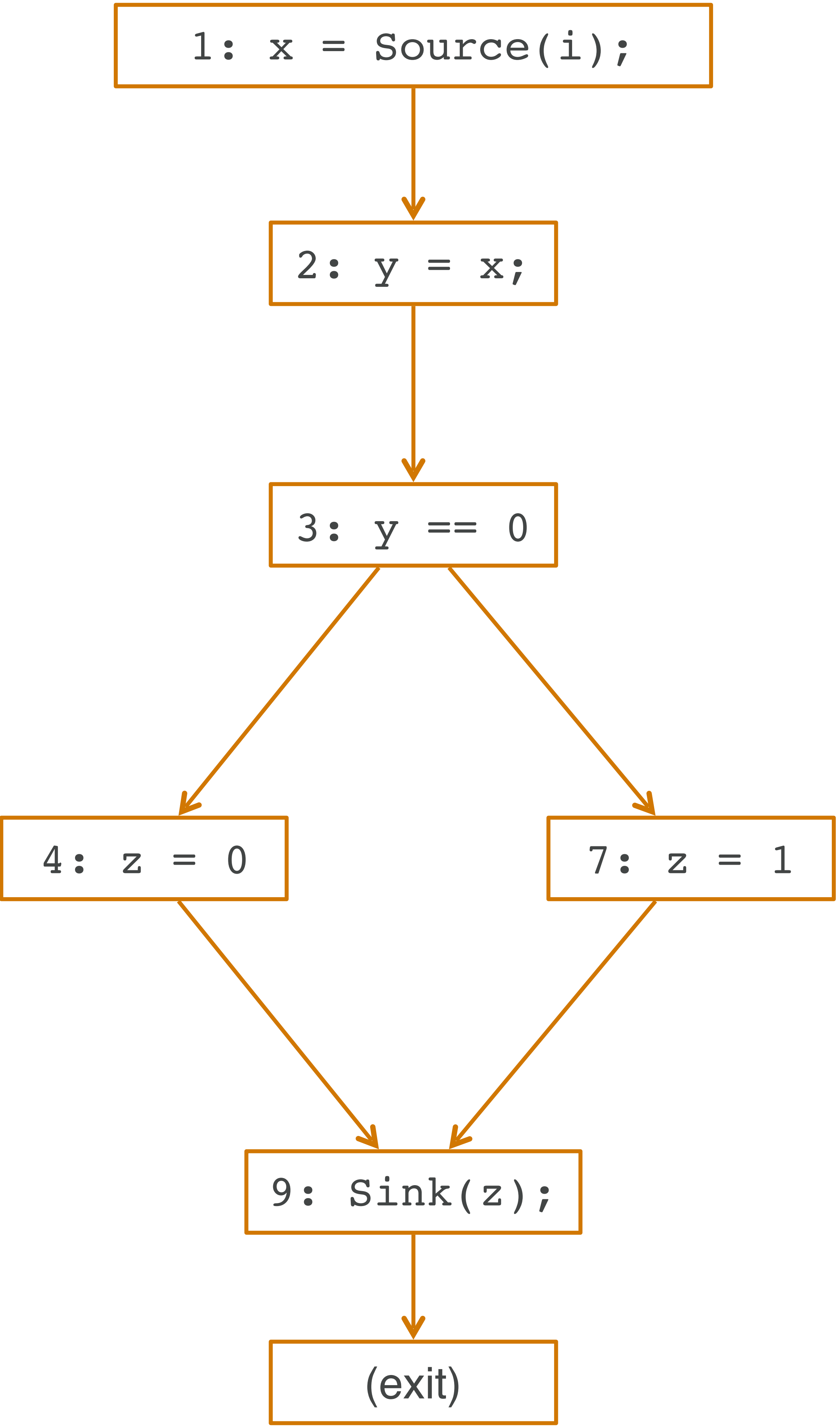
```
x = Source(i);
y = x;
if(y == 0) {
  z = 0
}
else {
  z = 1
}
Sink(z);
```





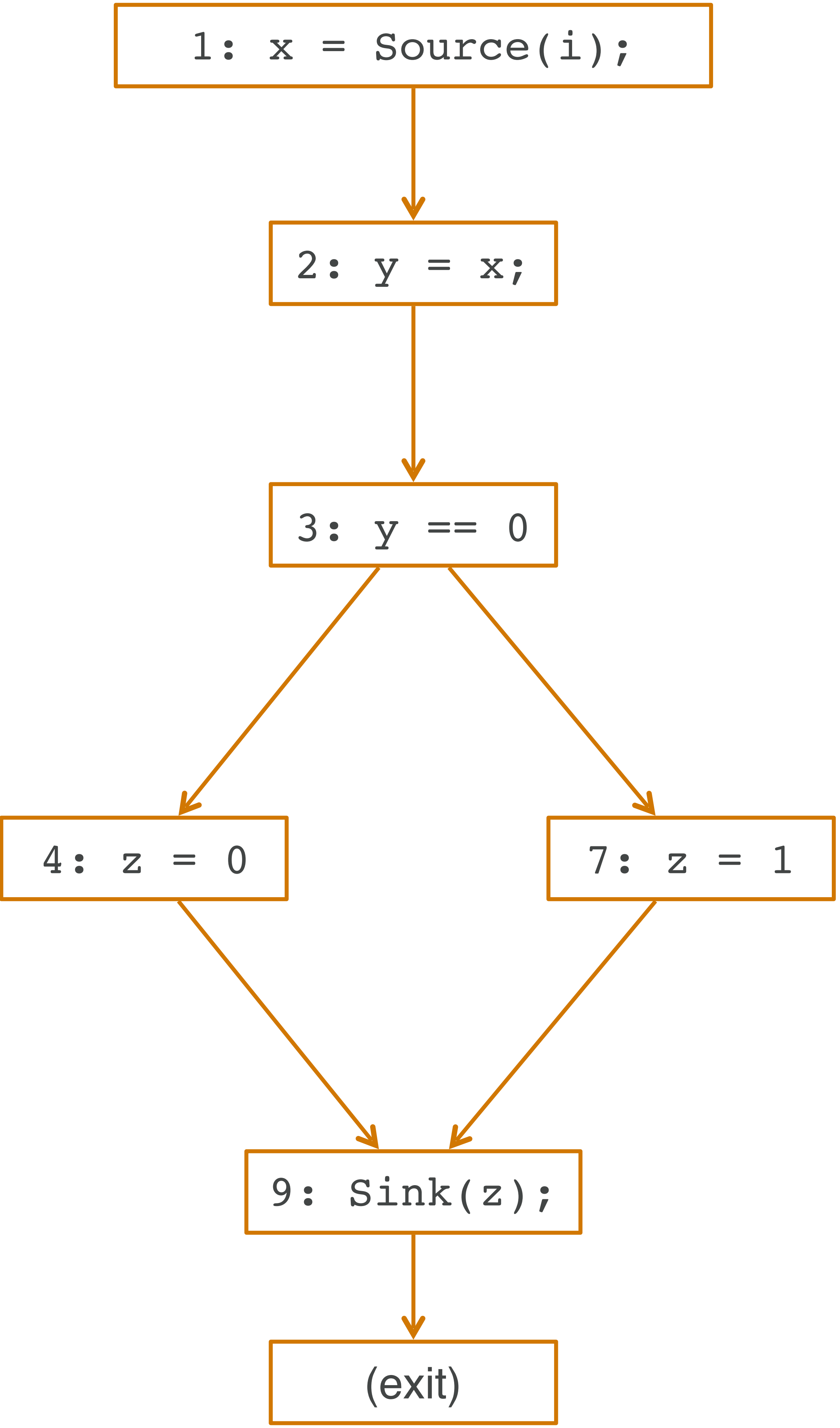
Kildall's Worklist Algorithm

```
for Instruction i in program
    input[i] = ⊥
input[firstInstruction] = initialDataflowInformation
worklist = { firstInstruction }

while worklist is not empty
    take an instruction i off the worklist
    output = flow(i, input[i])
    for Instruction j in succs(i)
        if output ⋢ input[j]
            input[j] = input[j] ⊔ output
            add j to worklist
```



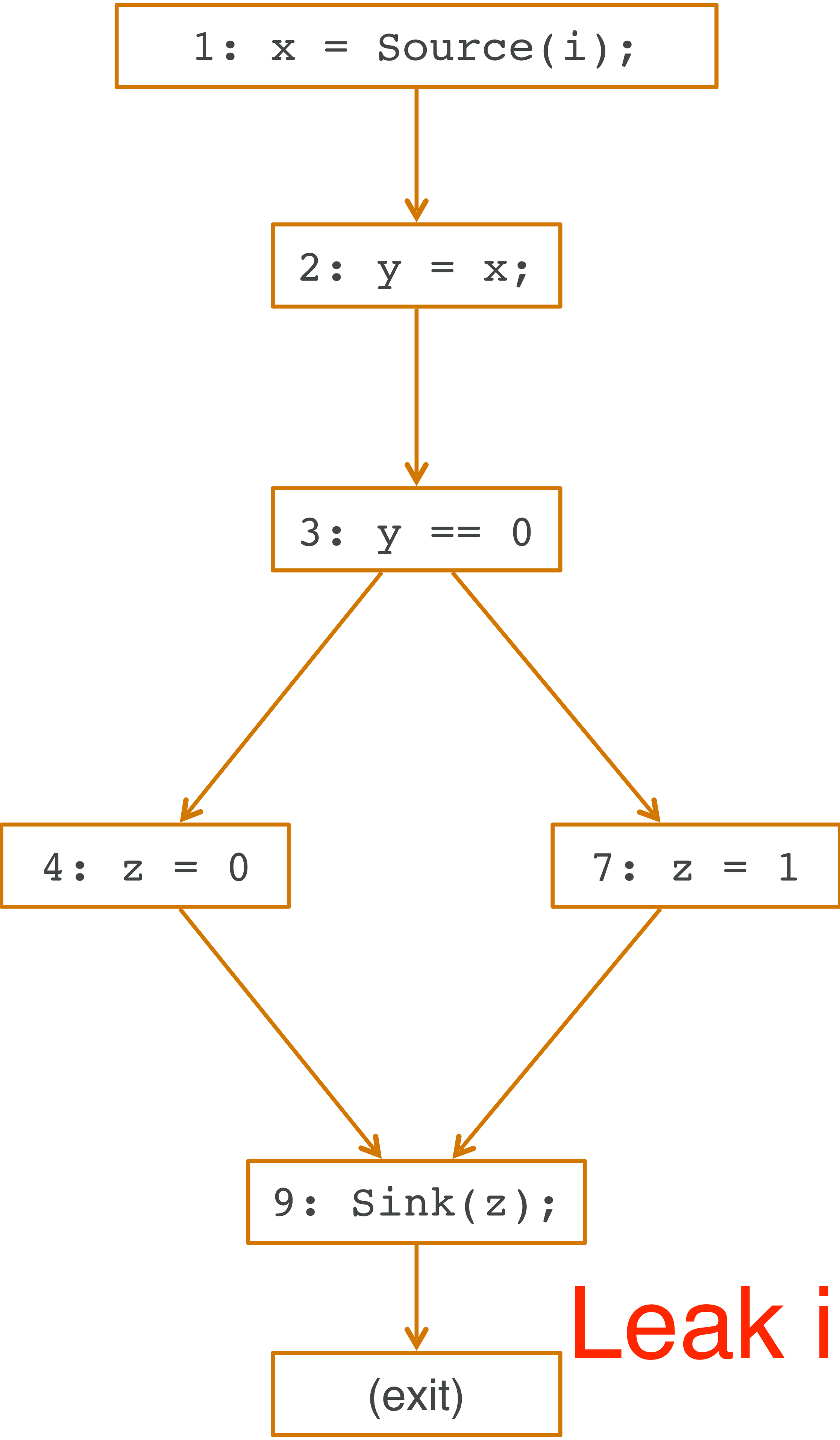
	Input		
Stmt	x	y	z
1	$\perp$	$\perp$	$\perp$
2	$\perp$	$\perp$	$\perp$
3	$\perp$	$\perp$	$\perp$
4	$\perp$	$\perp$	$\perp$
7	$\perp$	$\perp$	$\perp$
9	$\perp$	$\perp$	$\perp$

Stmt	Worklist	x	y	z



	Input		
Stmt	x	y	z
1	NT	NT	NT
2	T	NT	NT
3	T	T	NT
4	T	T	NT
7	T	T	NT
9	T	T	T

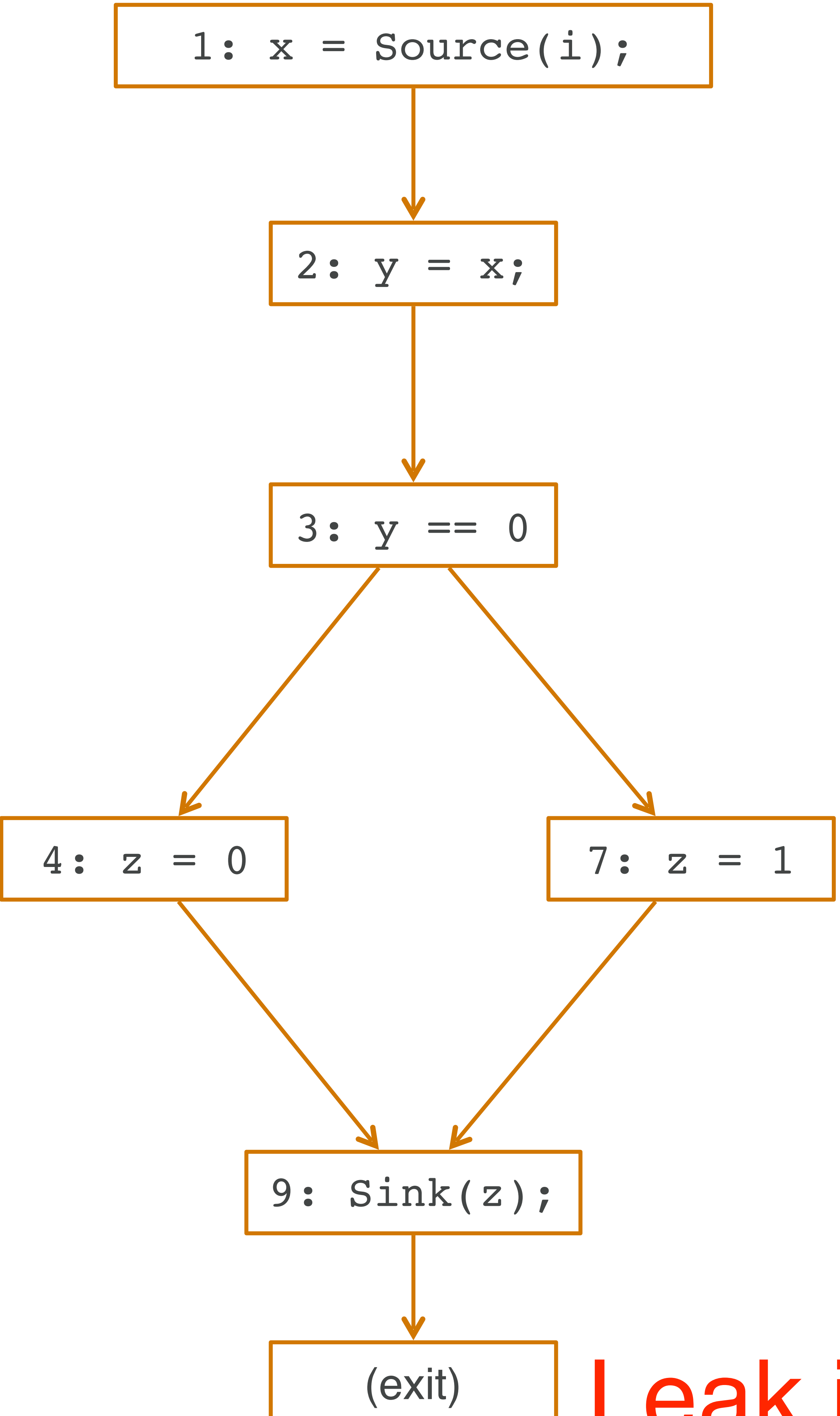
Stmt	Worklist	x	y	z
1	2	T	NT	NT
2	3	T	T	NT
3	4,7	T	T	NT
4	7,9	T	T	T
7	9	T	T	T
9		T	T	T



	Input		
Stmt	x	y	z
1	NT	NT	NT
2	T	NT	NT
3	T	T	NT
4	T	T	NT
7	T	T	NT
9	T	T	T

Stmt	Worklist	x	y	z
1	2	T	NT	NT
2	3	T	T	NT
3	4,7	T	T	NT
4	7,9	T	T	T
7	9	T	T	T
9		T	T	T

Leak in the program!



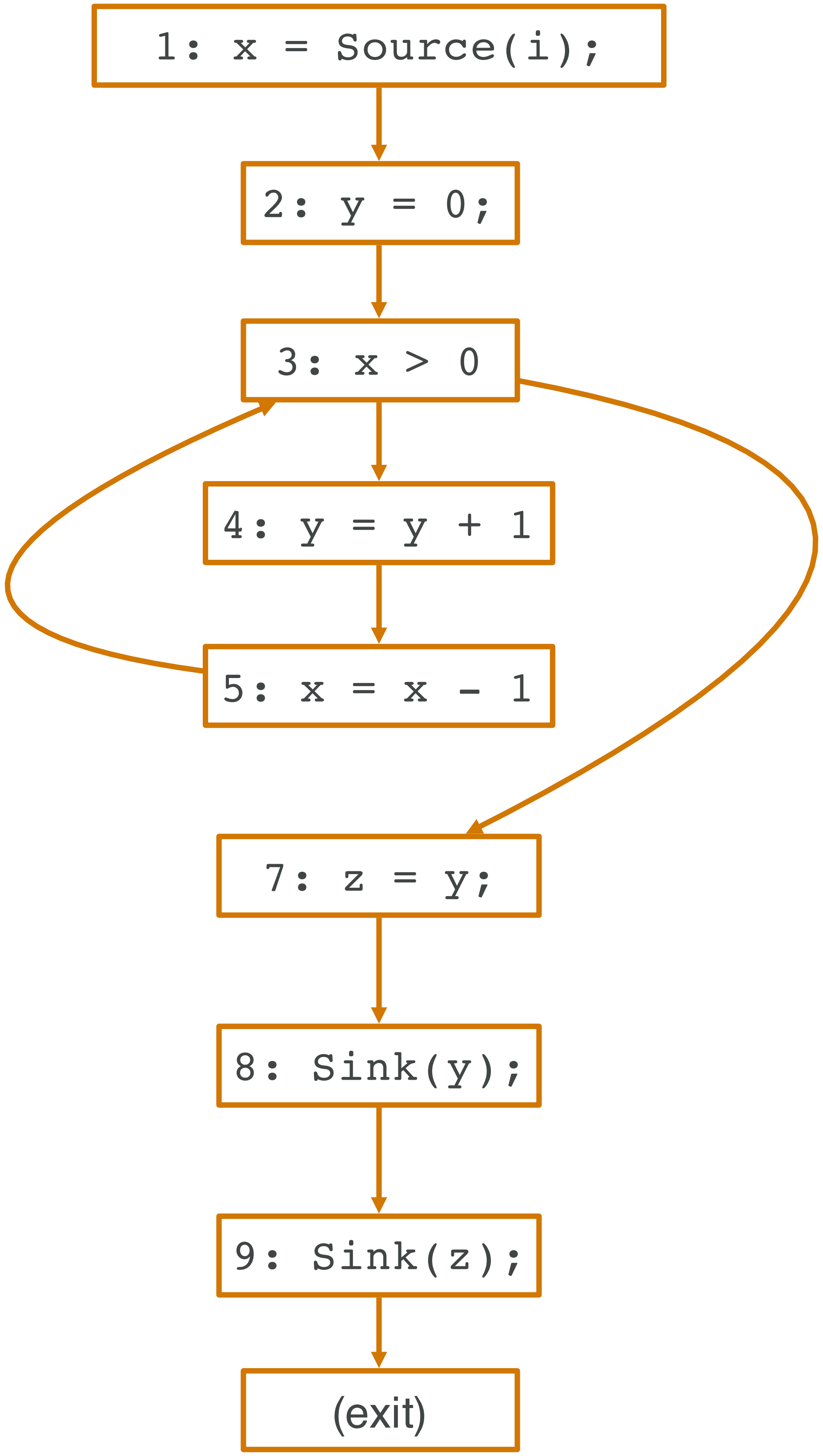
	Input		
Stmt	x	y	z
1	MT	MT	MT
2	T	MT	MT
3	T	T	MT
4	T	T	MT
7	T	T	MT
9	T	T	T

Stmt	Worklist	x	y	z
1	2	T	MT	MT
2	3	T	T	MT
3	4,7	T	T	MT
4	7,9	T	T	T
7	9	T	T	T
9		T	T	T

Leak in the program!

Try it yourself

```
x = Source(i);
y = x;
if(y == 0) {
  z = 0
}
else {
  z = 1
}
Sink(z);
```



	Input		
Stmt	x	y	z
1	NT	NT	NT
2	T	NT	NT
3	T	MT	NT
4	T	MT	NT
5	T	MT	NT
7	T	MT	NT
8	T	MT	MT
9	T	MT	MT

Stmt	Worklist	x	y	z
1	2	T	NT	NT
2	3	T	NT	NT
3	4,7	T	NT	NT
4	5,7	T	T	NT
5	3,7	T	T	NT
3	4,7	T	MT	NT
4	5,7	T	MT	NT
5	7	T	MT	NT
7	8	T	MT	MT
8	9	T	**MT**	MT
9		T	MT	**MT**

Possible leak in the program!

Limits of Static Analysis

- Do not know what values might cause the leak.
- Overtainting

Overtainting anti-patterns

```
1. x = Source(args[0]);
2. Object o = foo();
3. v = o.equals(x);
```

Overtainting anti-patterns

```
1. x = Source(args[0]);
2. Object o = foo();
3. v = o.equals(x);
```

All implementation of equals analyzed!

Overtainting anti-patterns

```
x = Source(args[0]);
if(Math.max(1, x) == 0) {
  Sink(x);
}
```

Overtainting anti-patterns

```
x = Source(args[0]);
if(Math.max(1, x) == 0) {
  Sink(x);      x→T
}
```

Overtainting anti-patterns

```
i = foo();
j = i + 1;
a[i] = Source();
a[j] = 0;
Sink(a);
Sink(a[i]);
Sink(a[j]);
```

Overtainting anti-patterns

```
i = foo();
j = i + 1;
a[i] = Source();          Taints the whole array
a[j] = 0;
Sink(a);                  a→T
Sink(a[i]);               a[i]→T
Sink(a[j]);               a[j]→T
```

COMBINING DYNAMIC AND STATIC ANALYSIS

Implicit Flows in Dynamic Analysis

```
x = Source(0);
y = x;
if(y == 0) {
   z = 2
}
else {
   z = 1
}
Sink(z);
```

x

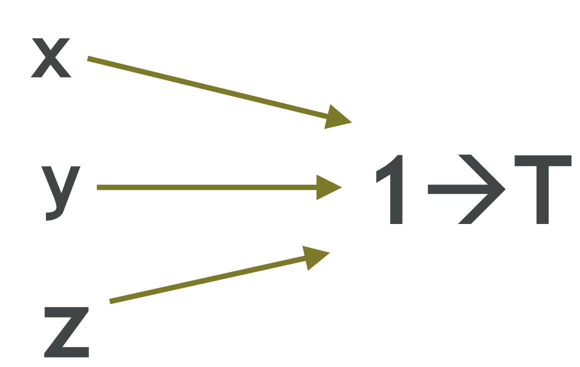
y → 0→T

z

Leak in the program!

Implicit Flows in Dynamic Analysis

```
x = Source(3);
y = x;
if(y == 0) {
  z = 2
}
else {
  z = 1
}
Sink(z);
```



Leak in the program!

Is there a leak? Why? Why not?

```
x = Source(3);
y = x;
z = 1;
if(y == 0) {
  z = 2
}
Sink(z);
```

Is there a leak? Why? Why not?

```
x = Source(3);
y = x;
z = 1;
if(y == 0) {
  z = 2
}
Sink(z);
```

x → 1→T
y → 1→T
z → 1→NT

No leak in the program!

Different result for Semantically the same Program?

```
x = Source(3);
y = x;
if(y == 0) {
  z = 3
}
else {
  z = 1
}
Sink(z);
```

Leak!

```
x = Source(3);
y = x;
z = 1;
if(y == 0) {
  z = 2
}
Sink(z);
```

No Leak!

Fundamental Issue

- In dynamic taint analysis, some implicit flows are hard to track
- If the code is not executed, we do not track its information.

How would you solve this issue?

```
x = Source(3);
y = x;
if(y == 0) {
  z = 2
}
else {
  z = 1
}
Sink(z);
```

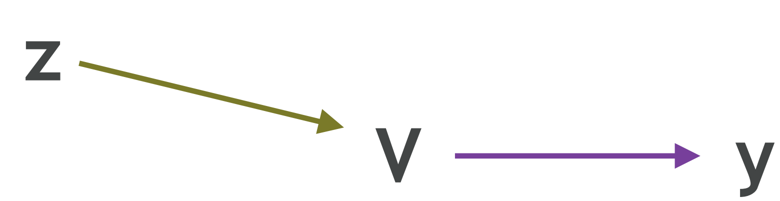
Leak!

```
x = Source(3);
y = x;
z = 1;
if(y == 0) {
  z = 2
}
Sink(z);
```

No Leak!

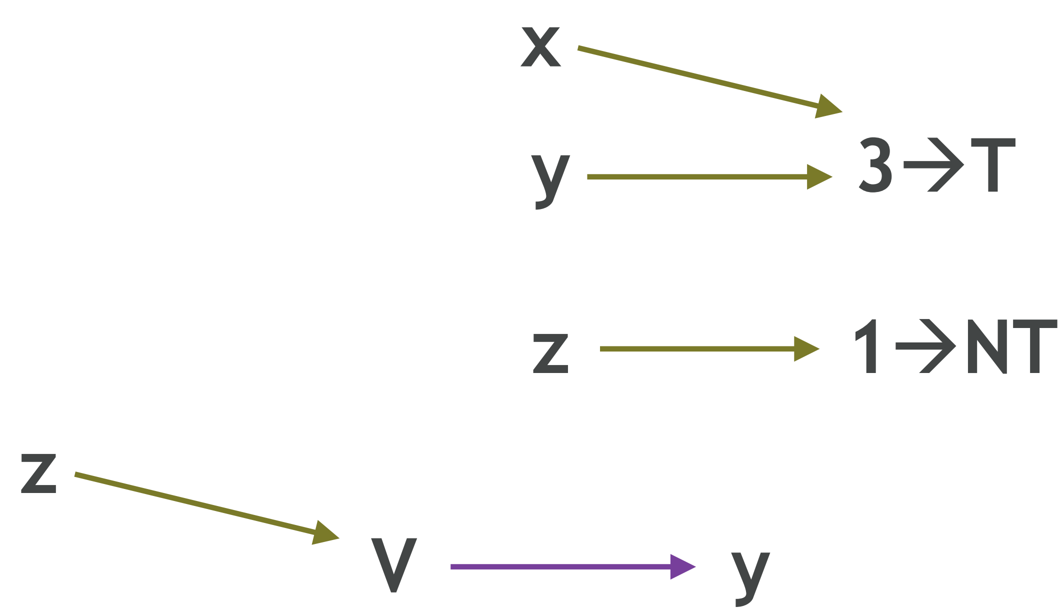
Branch-not-taken Analysis

```
x = Source(i);
y = x;
z = 1;

if(y == 0) {
  z = 2
}
Sink(z);
```



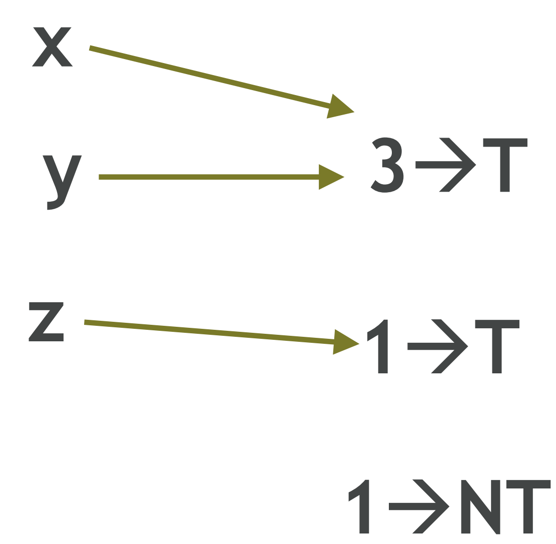
Branch-not-taken Analysis

```
x = Source(3);
y = x;
z = 1;

if(y == 0) {
  z = 2
}
Sink(z);
```



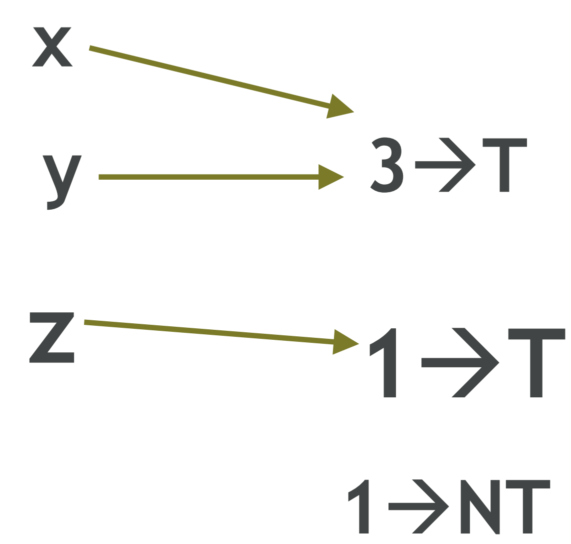
Branch-not-taken Analysis

```
x = Source(3);
y = x;
z = 1;

if(y == 0) {
  z = 2
}
Sink(z);
```



Branch-not-taken Analysis

```
x = Source(3);
y = x;
z = 1;

if(y == 0) {
  z = 2
}
Sink(z);
```



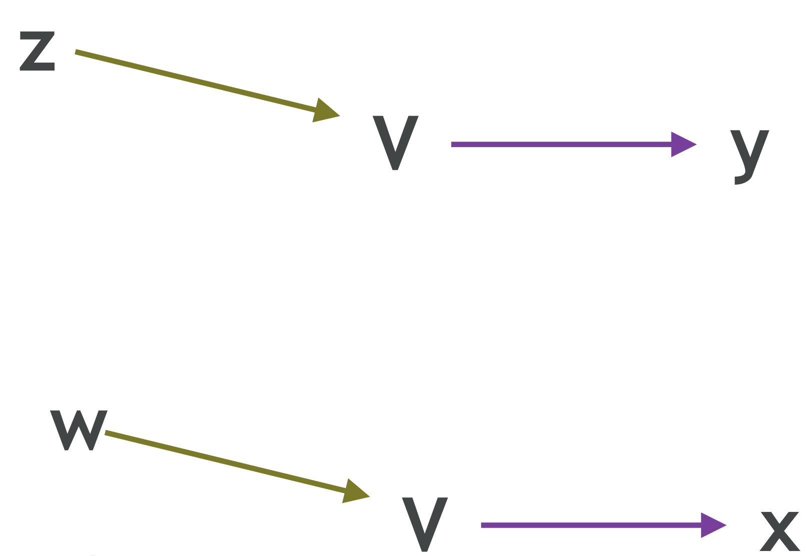
Leak in the program!

Is there a leak? Why? Why not?

```
x = Source(3);
y = x;
z = 1;
w = 1;
if(y == 0) {
   z = 2
   if(x == 0) {
     w = 0;
   }
}
Sink(w);
```

Limits of Branch-not-taken Analysis

```
x = Source(3);
y = x;
z = 1;
w = 1;

if(y == 0) {
    z = 2

    if(x == 0) {
      w = 0;
    }
  }
  Sink(w);
```



INTERPROCEDURAL ANALYSIS

Interprocedural Analysis

```
main() {
   x = Source(1);
   y = 1;
   z = foo(x);
   Sink(z);
   z = foo(y);
   Sink(z);
}
```

```
foo(x) {
   y = x * 2;
   return x;
}
```

Interprocedural Analysis

```
main() {
    x = Source(1);
    y = 1;
    z = foo(x);
    Sink(z);
    z = foo(y);
    Sink(z);
}
```

```
foo(x) {
    y = x * 2;
    return x;
}
```

Information with
context T

institute for SOFTWARE RESEARCH

Interprocedural Analysis

```
main() {
    x = Source(1);
    y = 1;
    z = foo(x);
    Sink(z);
    z = foo(y);
    Sink(z);
}
```

```
foo(x) {
    y = x * 2;
    return x;
}
```

Information with context T

Information with context NT

Summary

- Taint analysis is an information flow analysis to detect if private data is leaked in the program.
- Compare benefits and limitations of dynamic and static approaches.
- Can be combined to overcome the limitations of the other.