جامعة نيويورك أبوظبي

NYU ABU DHABI

IFDS Framework

CS-UH 3260
Static Program Analysis

Karim Ali
@karimhamdanali

Previously

- Context sensitivity
- Call strings
- Functional approach

Interprocedural Finite Distributive Subset Problems

Thomas Reps, Susan Horwitz, and Mooly Sagiv. Precise Interprocedural Dataflow Analysis via Graph Reachability. ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '95), pages 49–61.

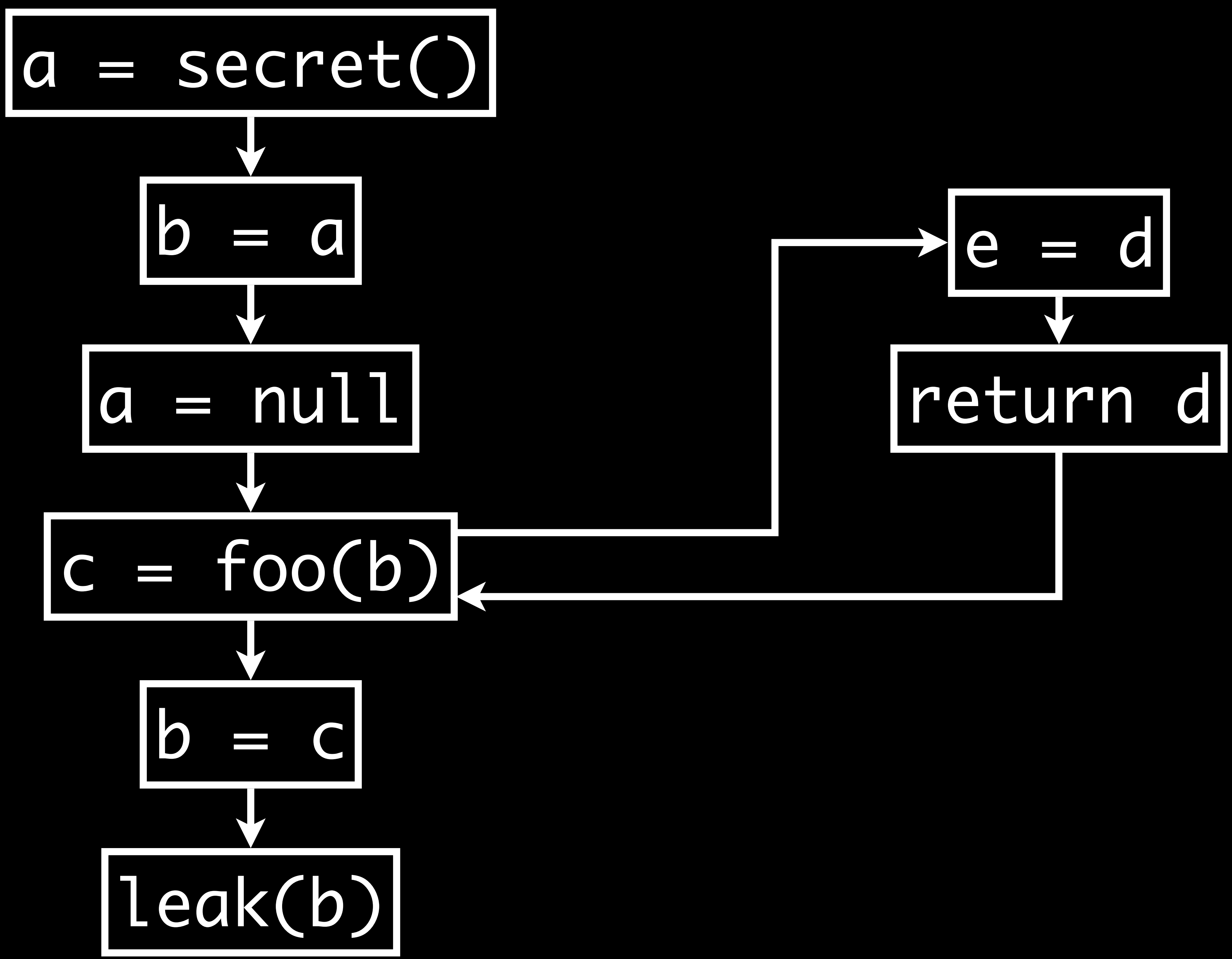
Interprocedural **F**inite **D**istributive **S**ubset Problems

Interprocedural

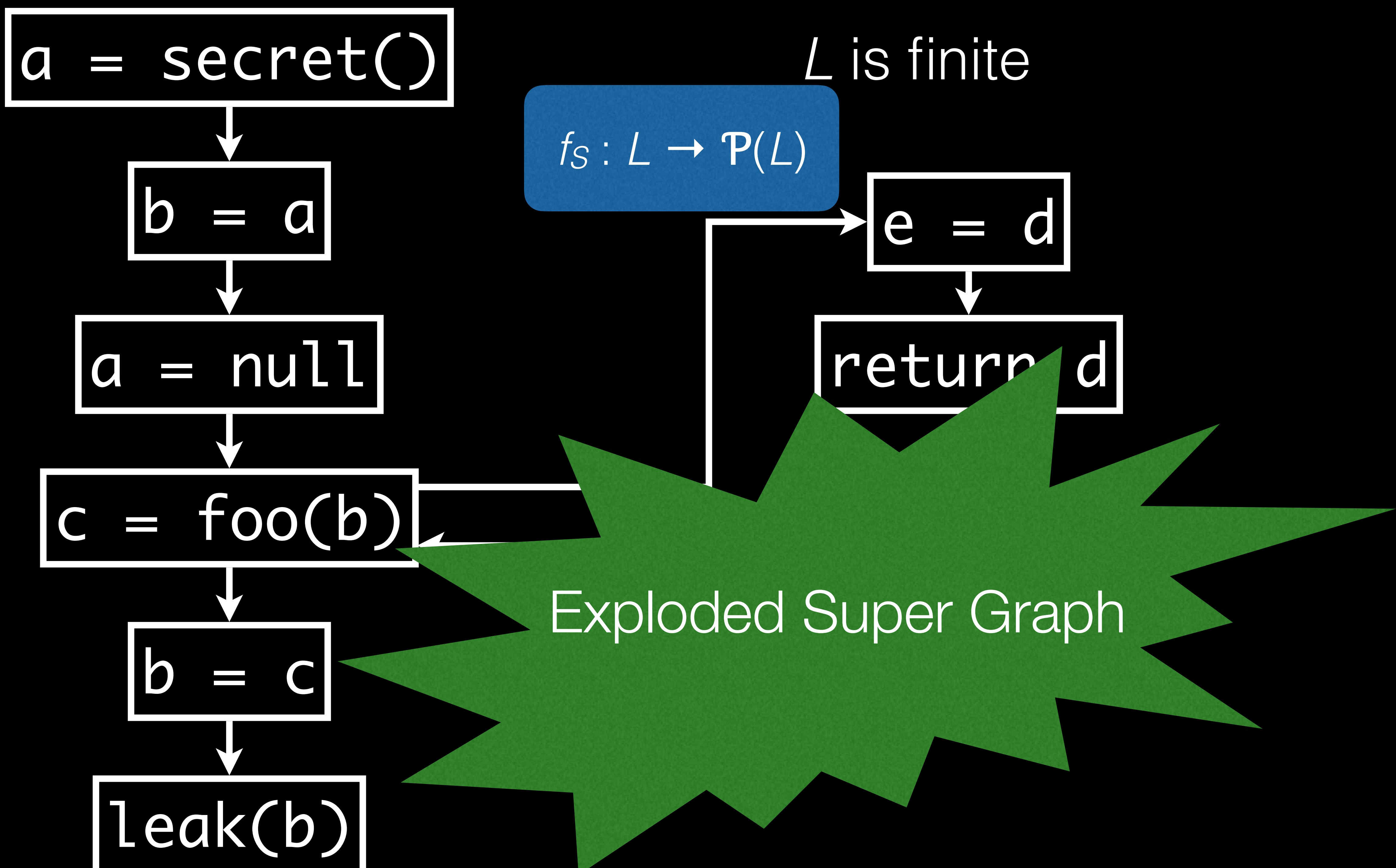
```
main() {
  a = secret();
  b = a;
  a = null;
  c = foo(b);
  b = c;
  leak(b);
}
```

```
foo(d) {
  e = d;
  return d;
}
```

Interprocedural

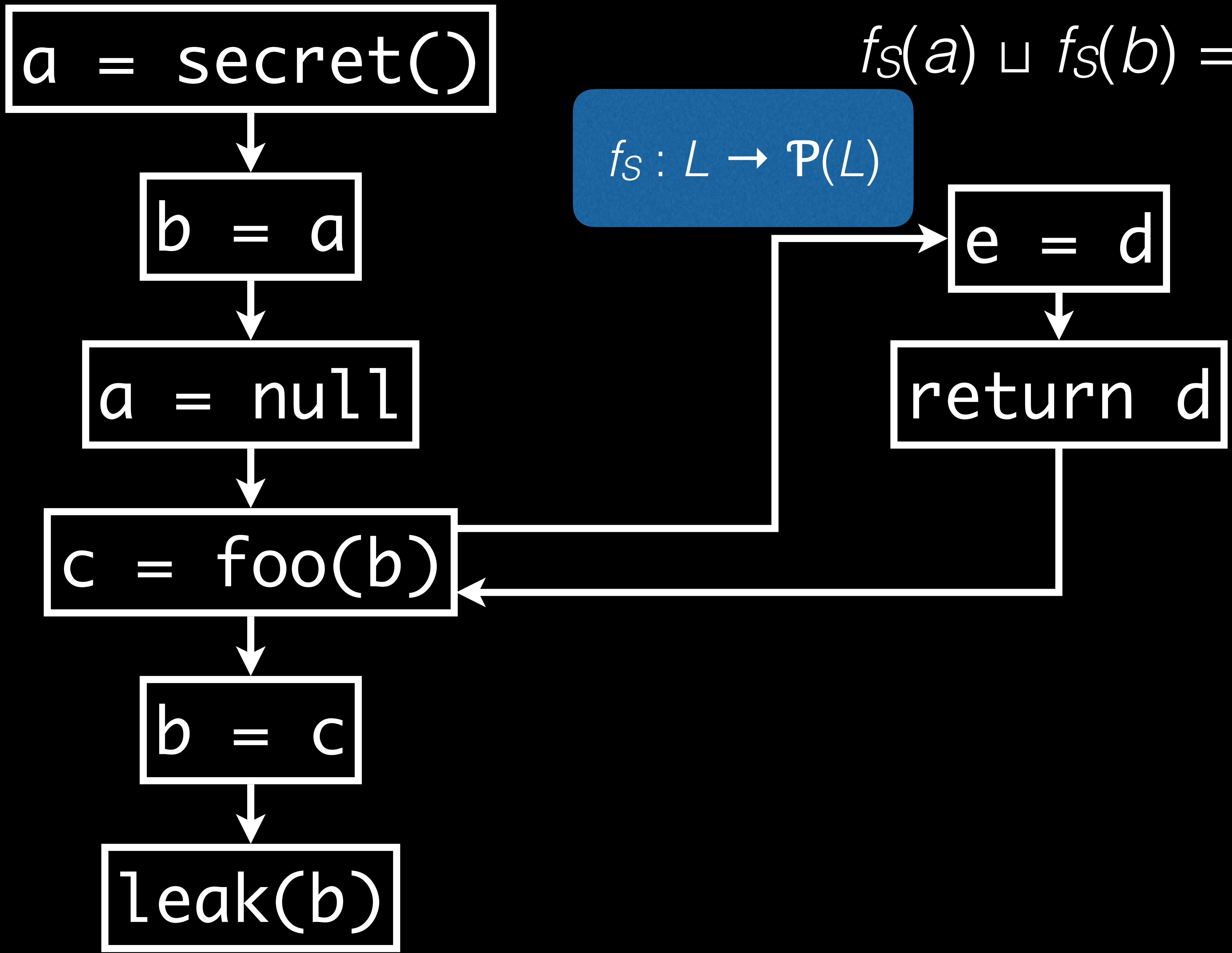


Finite


a = secret()
b = a
a = null
c = foo(b)
b = c
leak(b)
L is finite
$f_S : L \rightarrow \mathcal{P}(L)$
e = d
return d
Exploded Super Graph

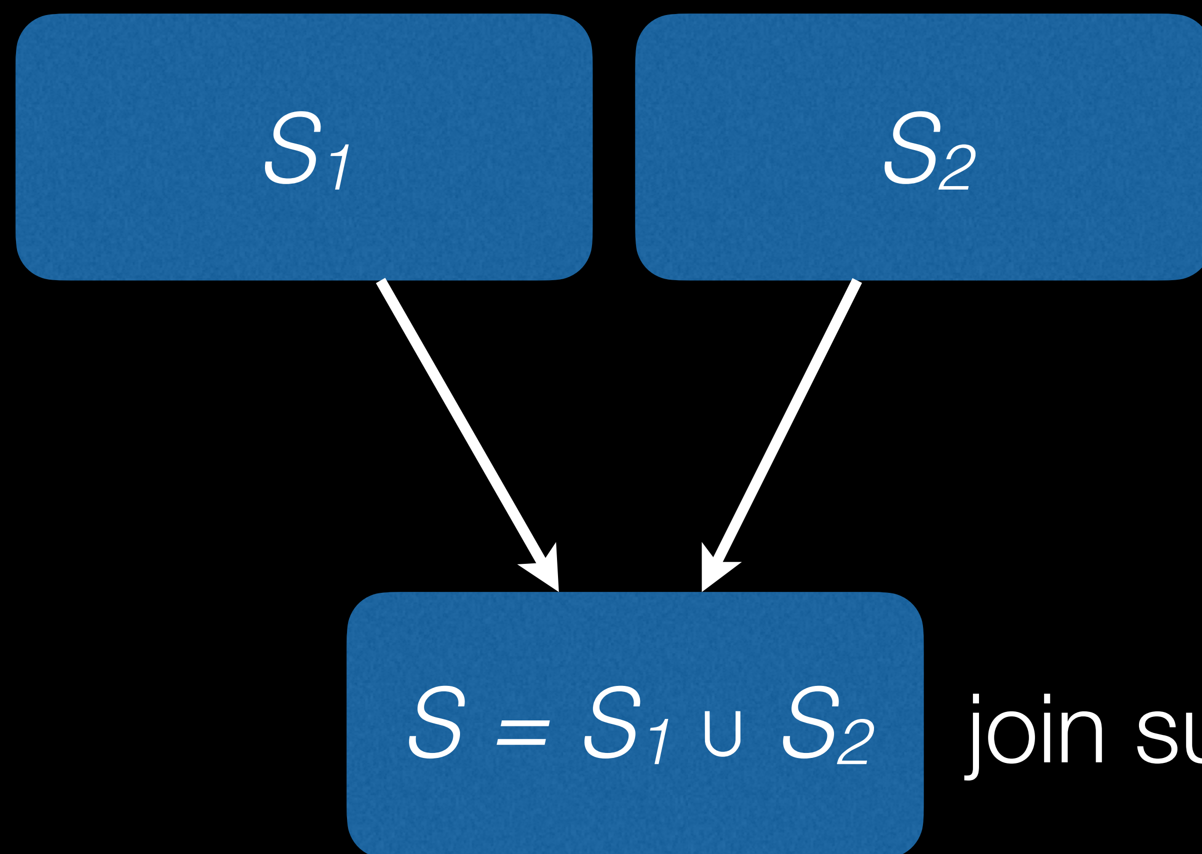
Distributive

$$f_S(a) \sqcup f_S(b) = f_S(a \sqcup b)$$



@karimhamdanali

Subset



join subsets

alias

taint

typestate

IFDS

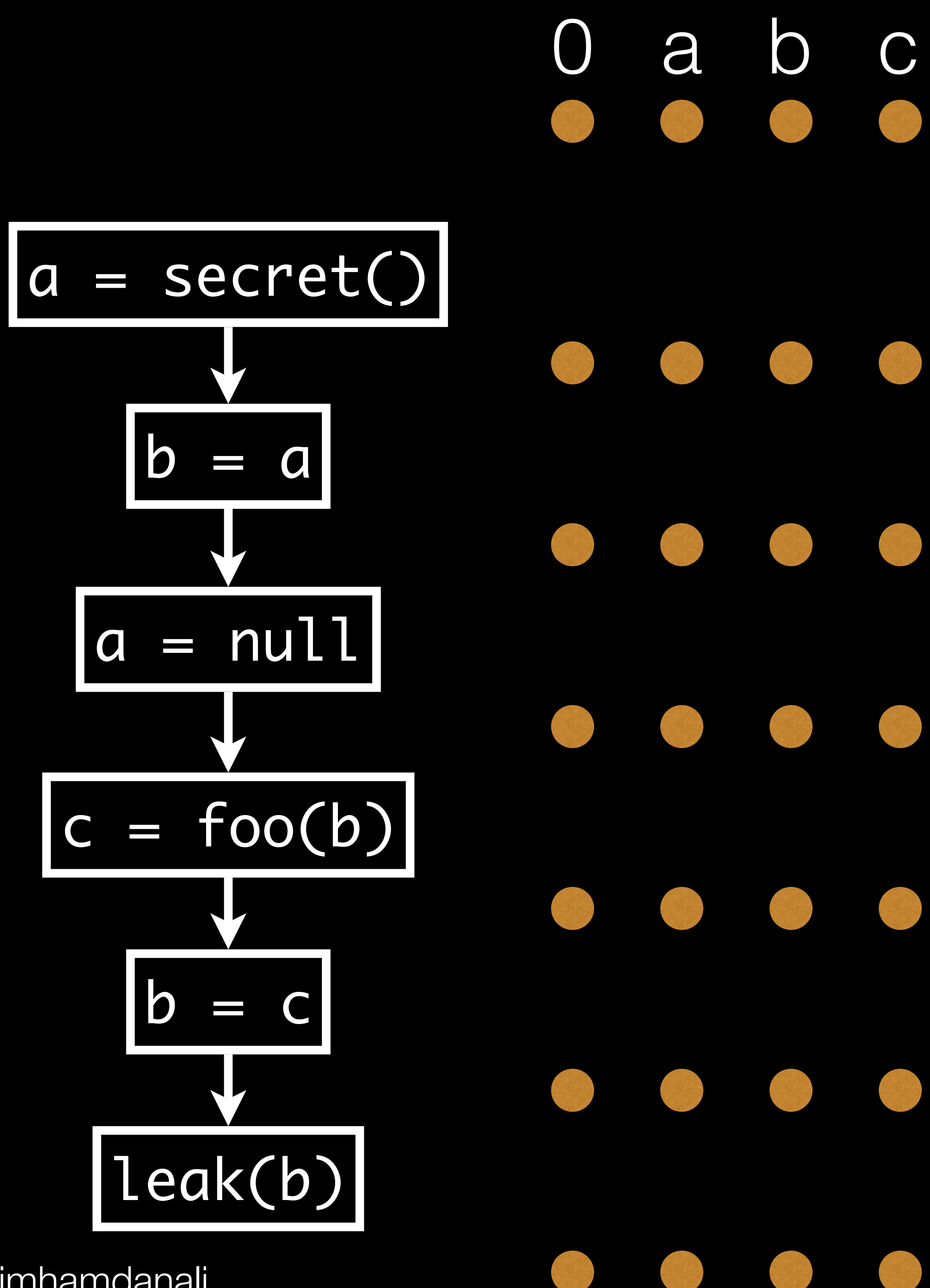
constant propagation

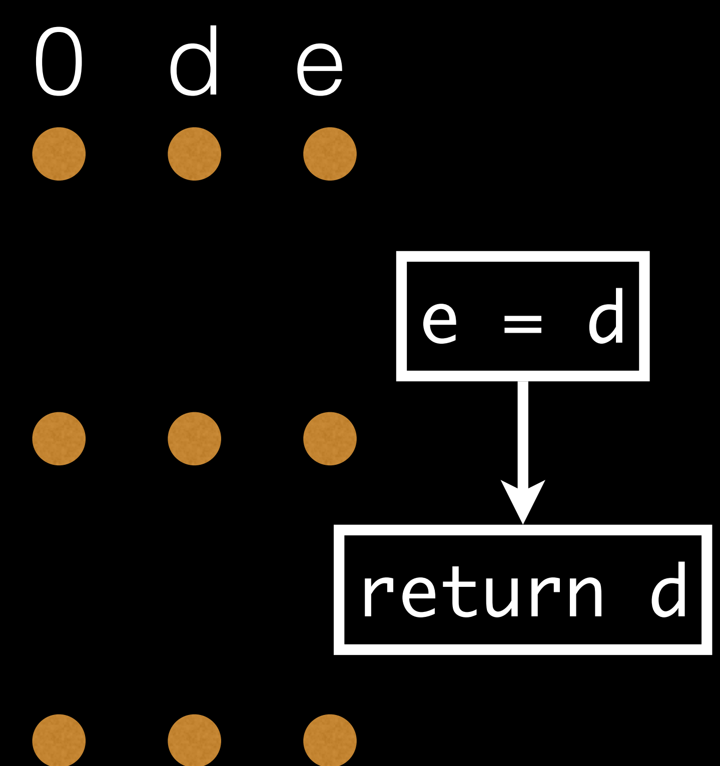
reaching definitions

uninitialized variables

Graph Reachability

Graph Reachability



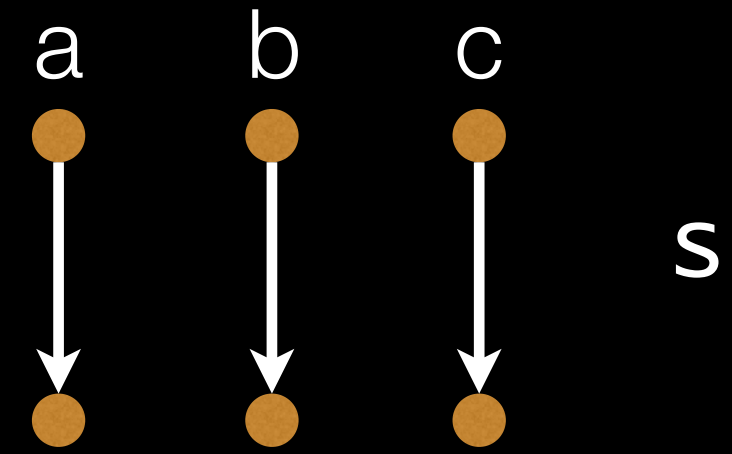


- each node is a fact
- fact holds at a statement => node is reachable in ESG

Examples of Flow Functions

Identity Flow Functions

in(**s**) = facts before **s**

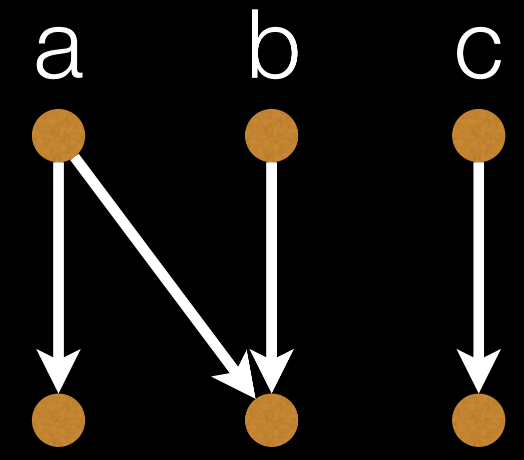


out(**s**) = facts after **s**

every fact is reachable iff it was previously reachable

Flow Functions

$$\text{out}(\mathbf{s}) = \begin{cases} \text{in}(\mathbf{s}) \cup \{b\} \text{ if } f \in \text{in}(\mathbf{s}) \\ \text{in}(\mathbf{s}),\ \textit{otherwise} \end{cases}$$



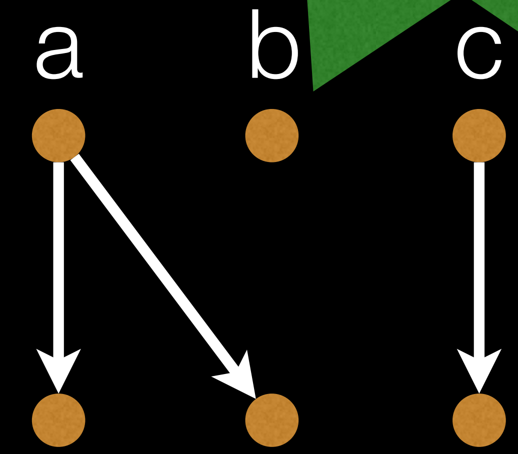
`s: b = a`

every fact is reachable iff it was previously reachable, and *b* is also reachable if *a* was reachable

Flow Functions

Taint Analysis

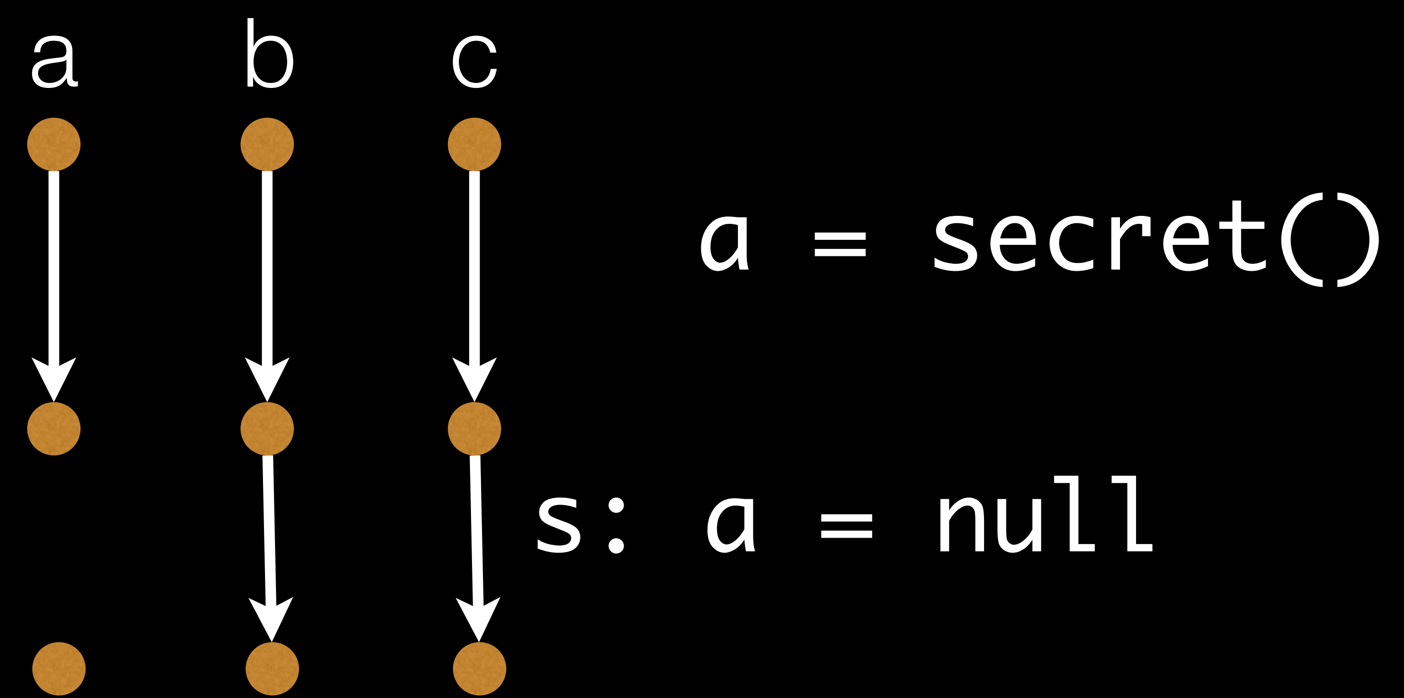
$$\text{out}(\mathbf{s}) = \begin{cases} \text{in}(\mathbf{s}) \cup \{b\} \text{ if } f \in \text{in}(\mathbf{s}) \\ \text{in}(\mathbf{s}) \setminus \{b\}, \textit{ otherwise} \end{cases}$$




```
s: b = a
```

every fact except *b* is reachable iff it was previously reachable; *b* is reachable if *a* was reachable

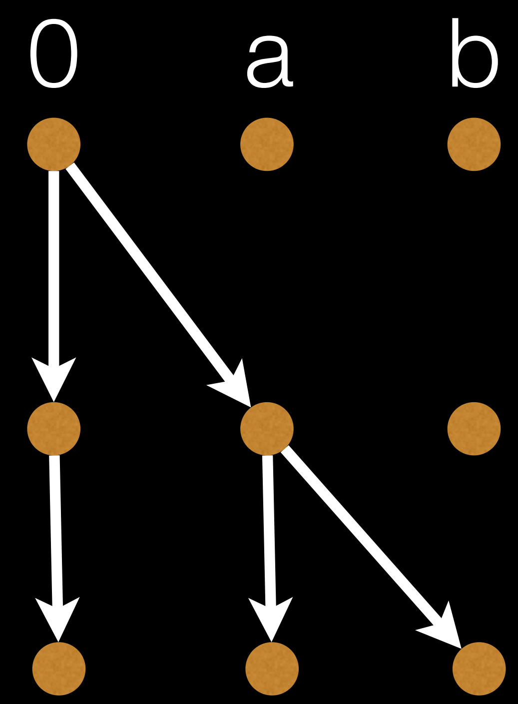
“Killing” Facts



out(**s**) = in(**s**) \ {*a*}

a is not reachable, even if it was before

Generating Facts



$out(\mathbf{s}) = in(\mathbf{s}) \cup \{a\}$

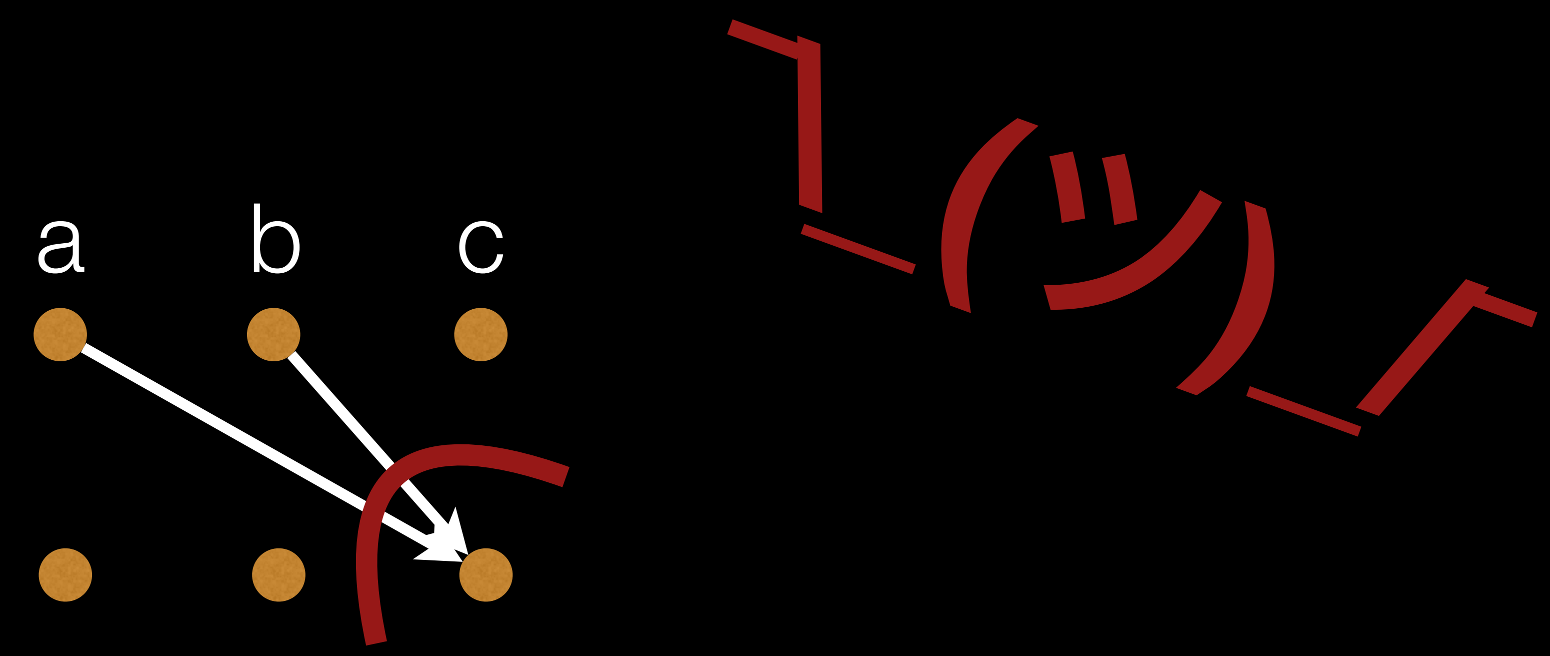
```
a = secret()
```

$out(\mathbf{s}) = in(\mathbf{s}) \cup \{b\}$

```
b = a
```

0 is the tautological fact => always reachable

Non-Distributive
Flow Functions



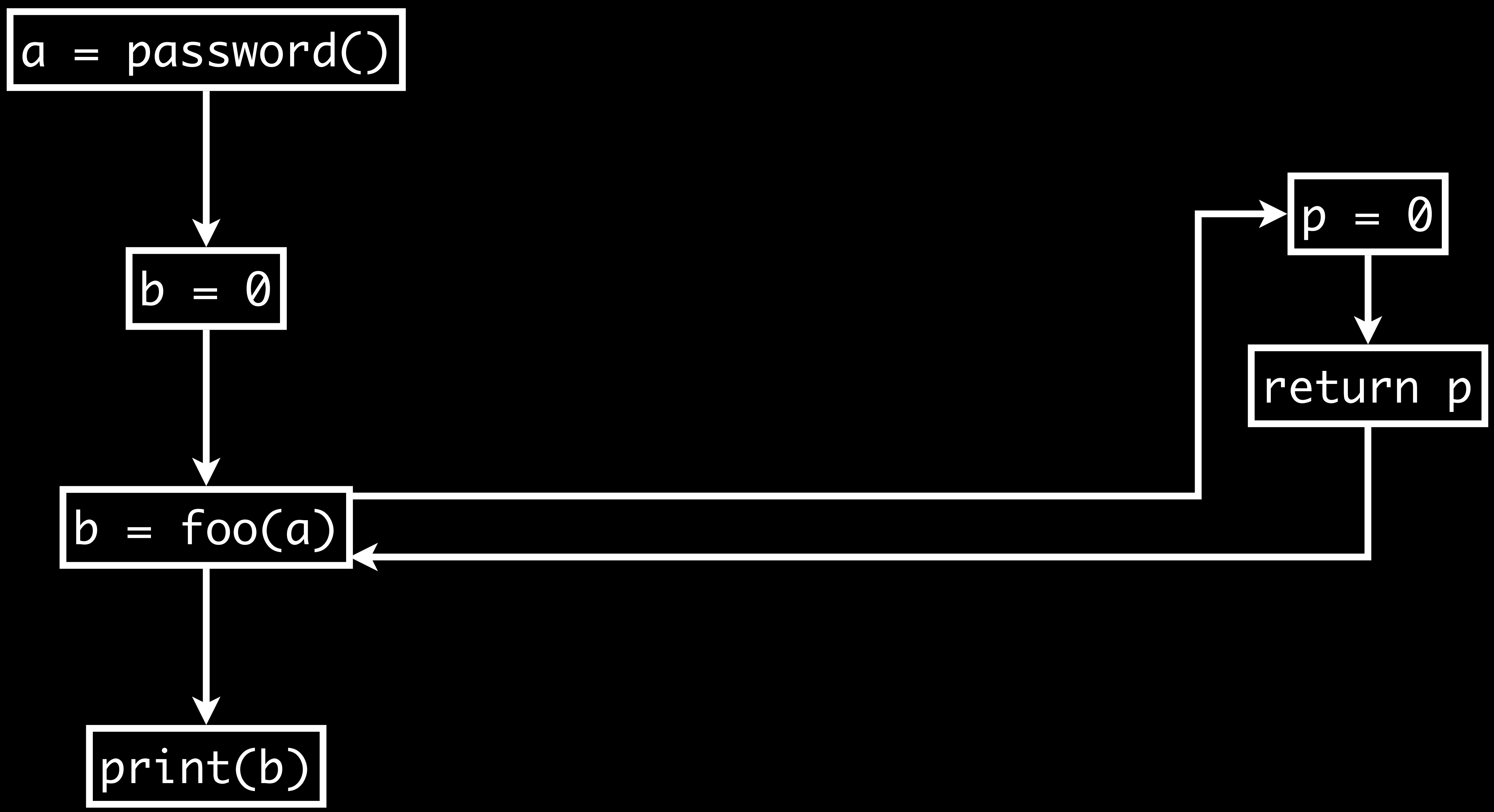
e.g., full constant propagation
c = a + b

Taint Analysis Example

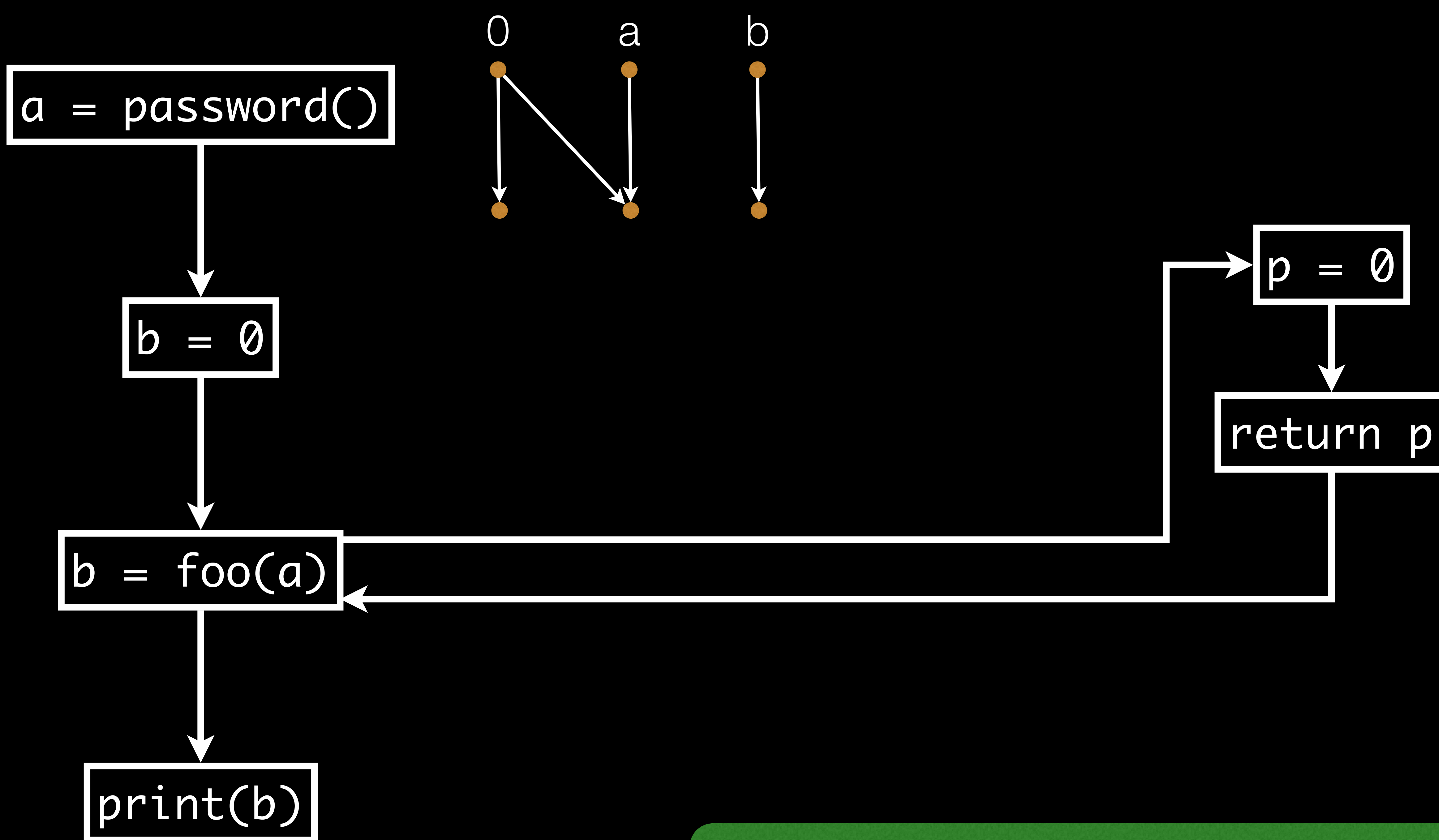
```
void main() {
    int a = password();
    int b = 0;

    b = foo(a);

    print(b);
}
```

```
int foo(int p) {
    p = 0;

    return p;
}
```

Taint Analysis Example

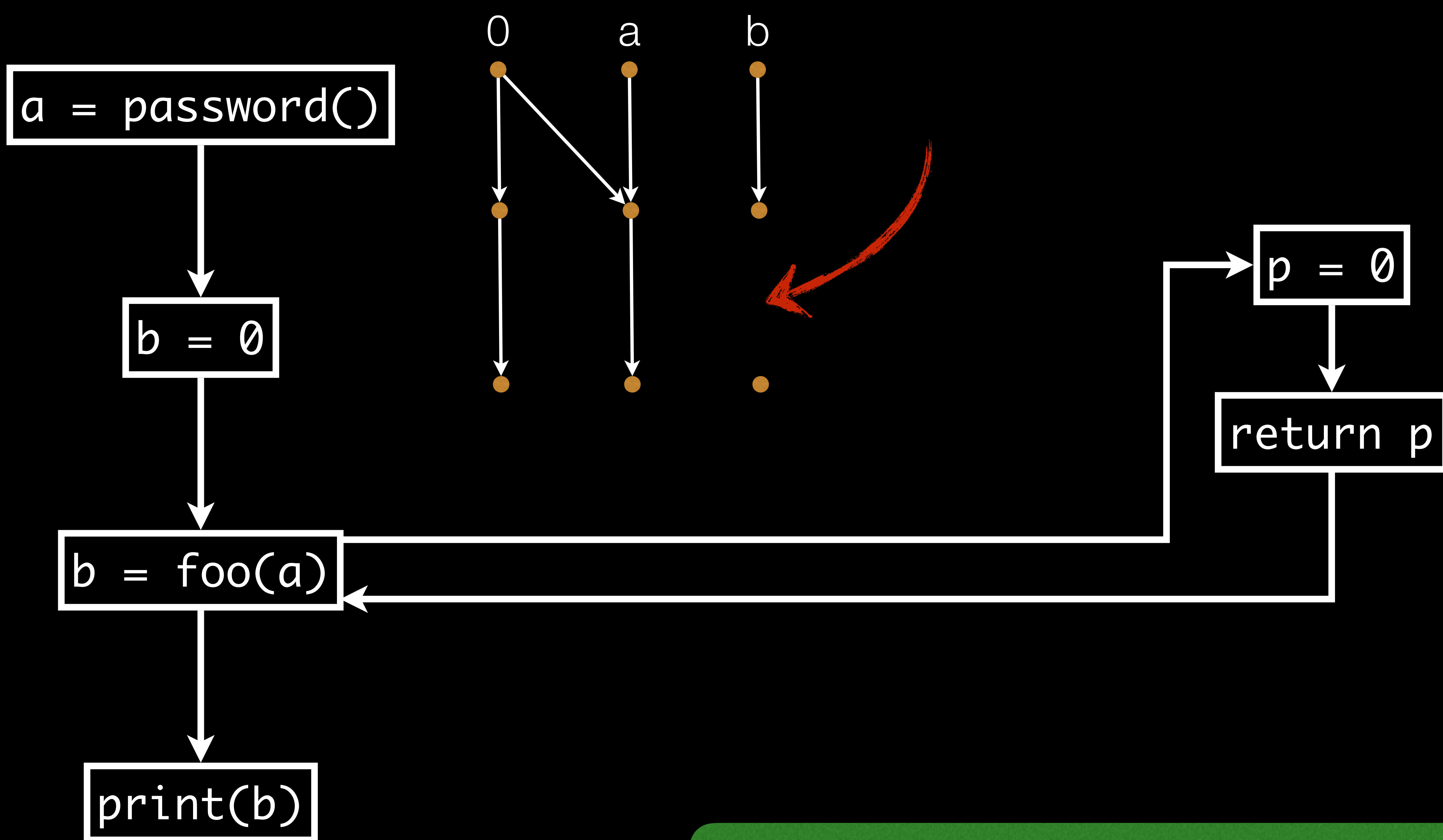


Taint Analysis Example



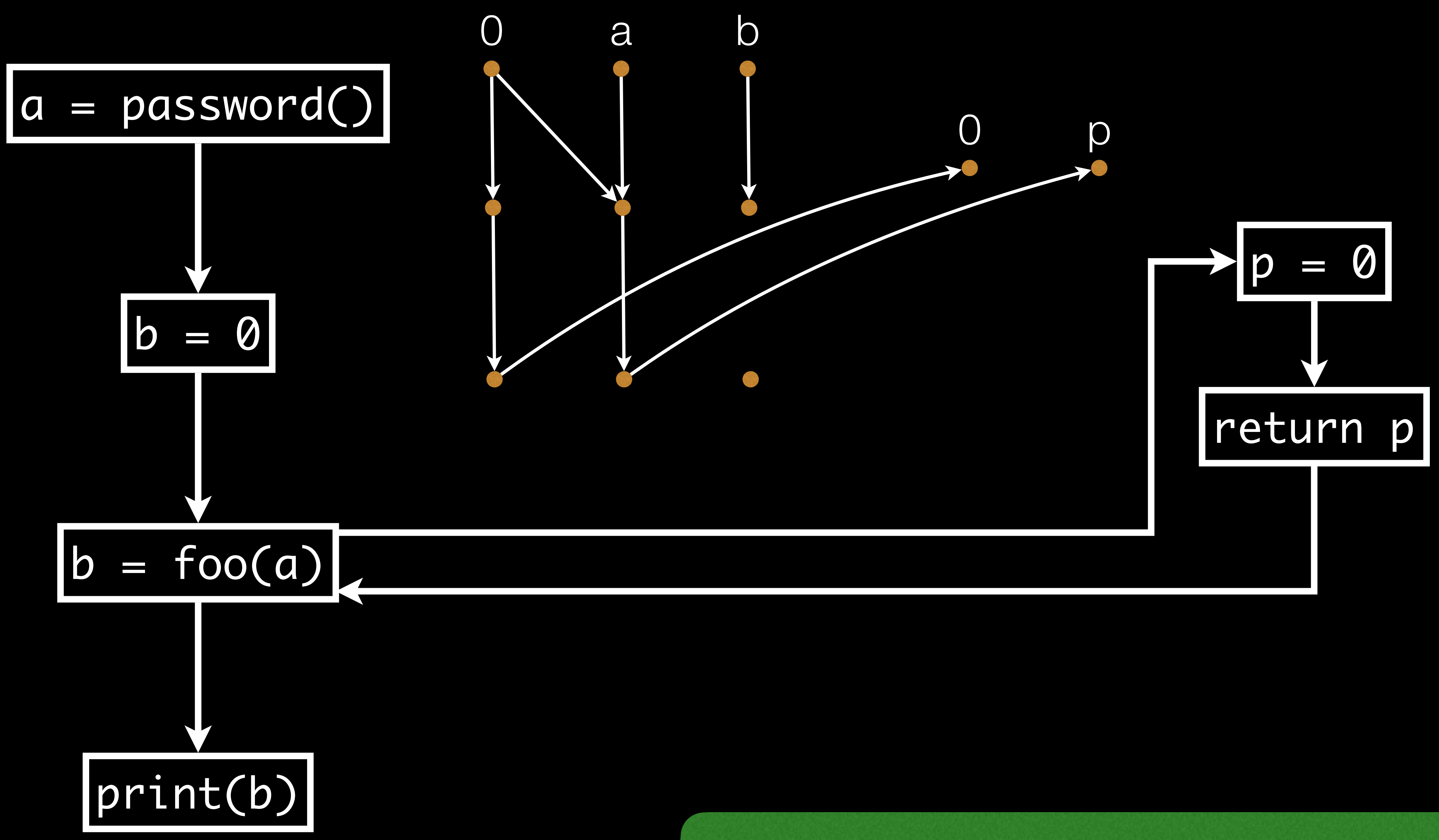
Normal-Flow Function

Taint Analysis Example



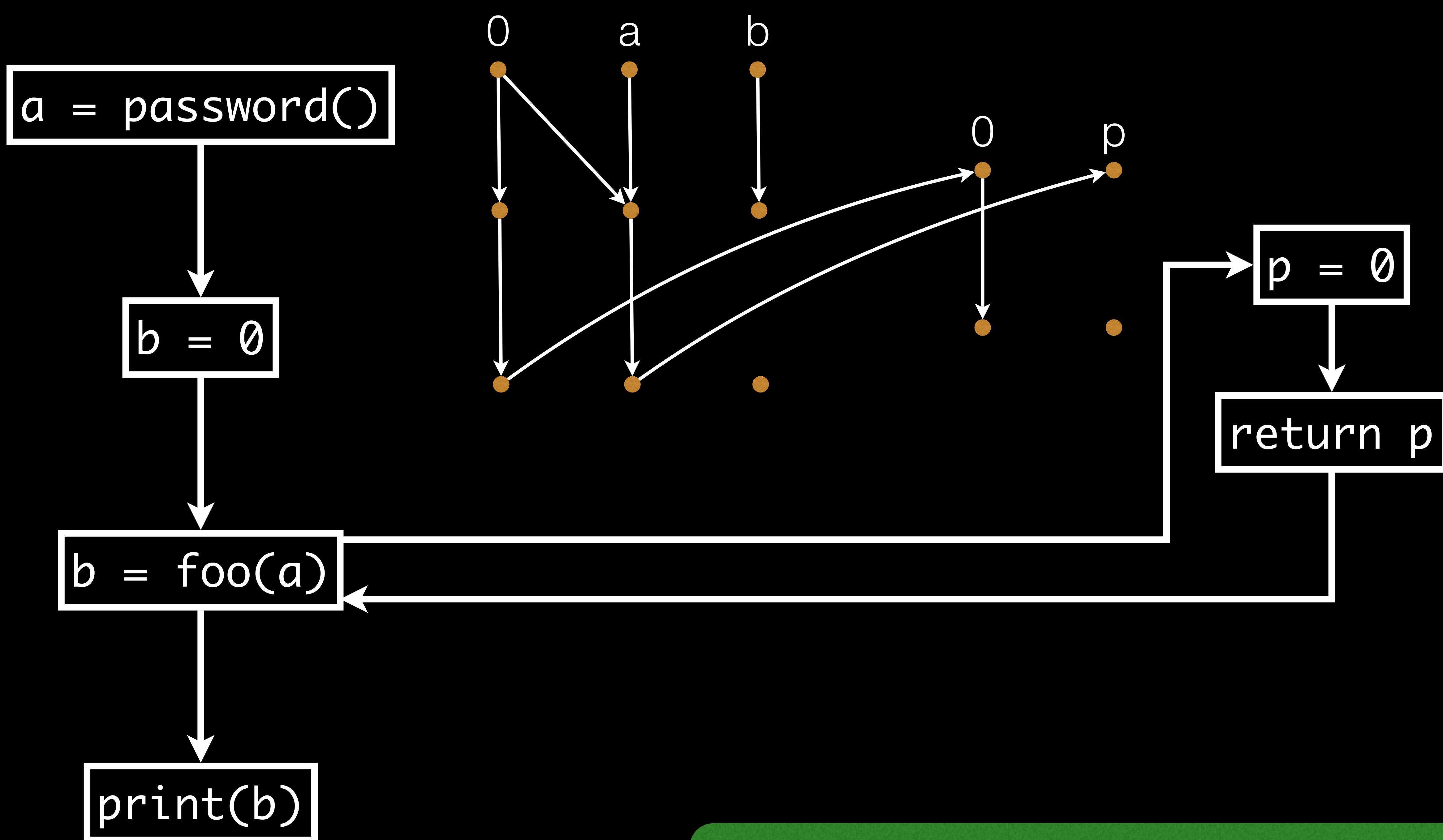
still Normal-Flow Function

Taint Analysis Example



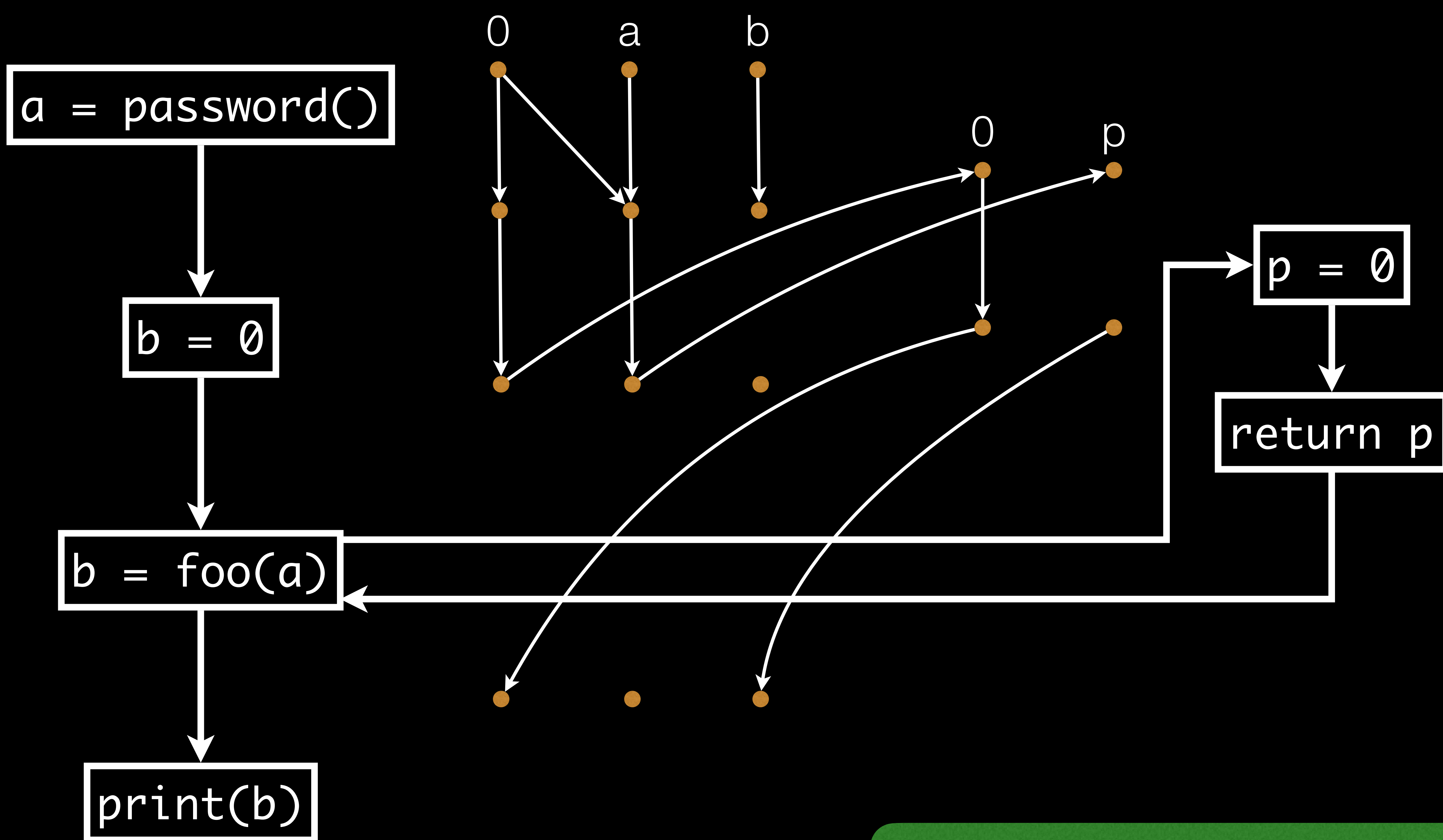
Call-Flow Function

Taint Analysis Example



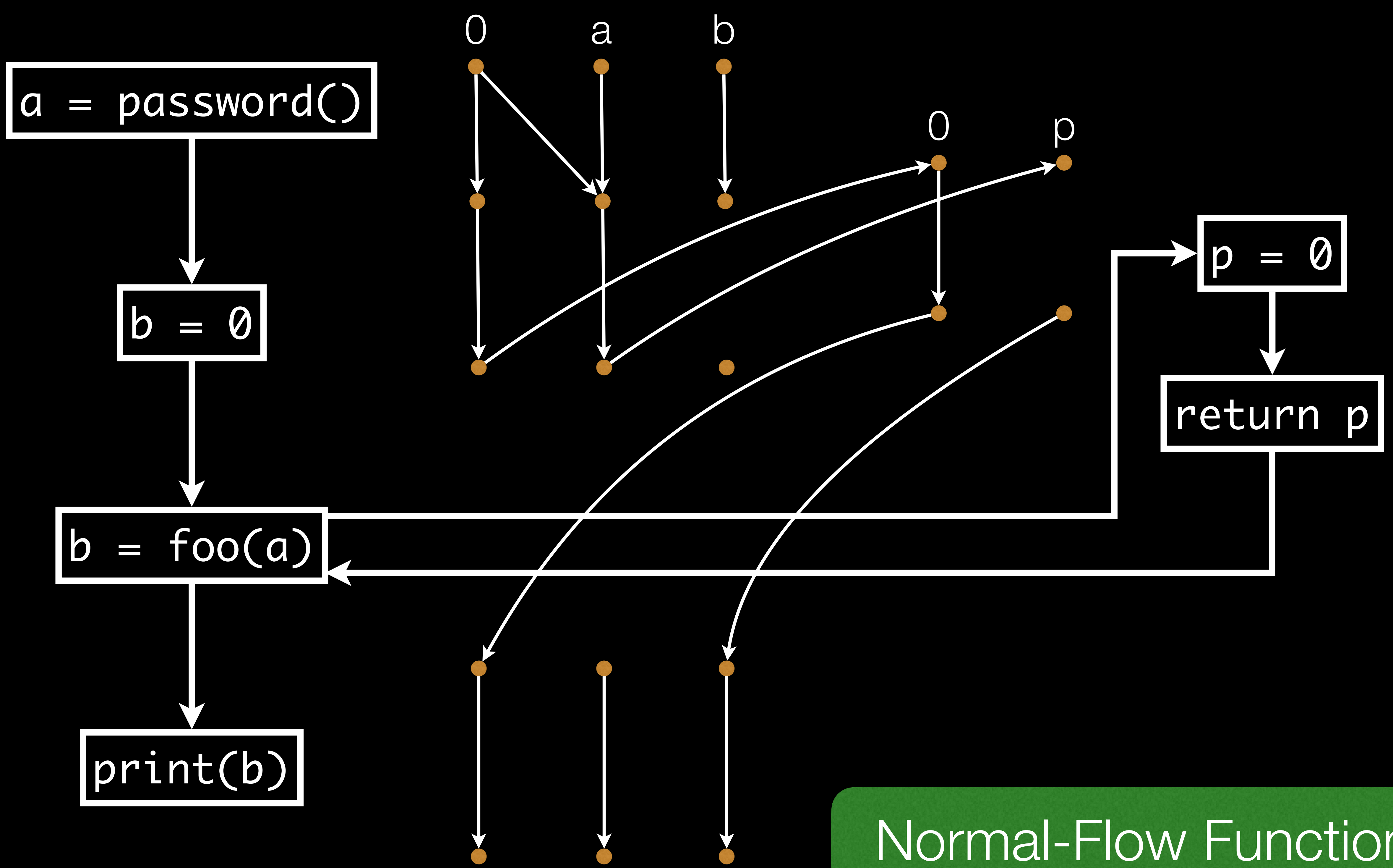
Normal-Flow Function

Taint Analysis Example



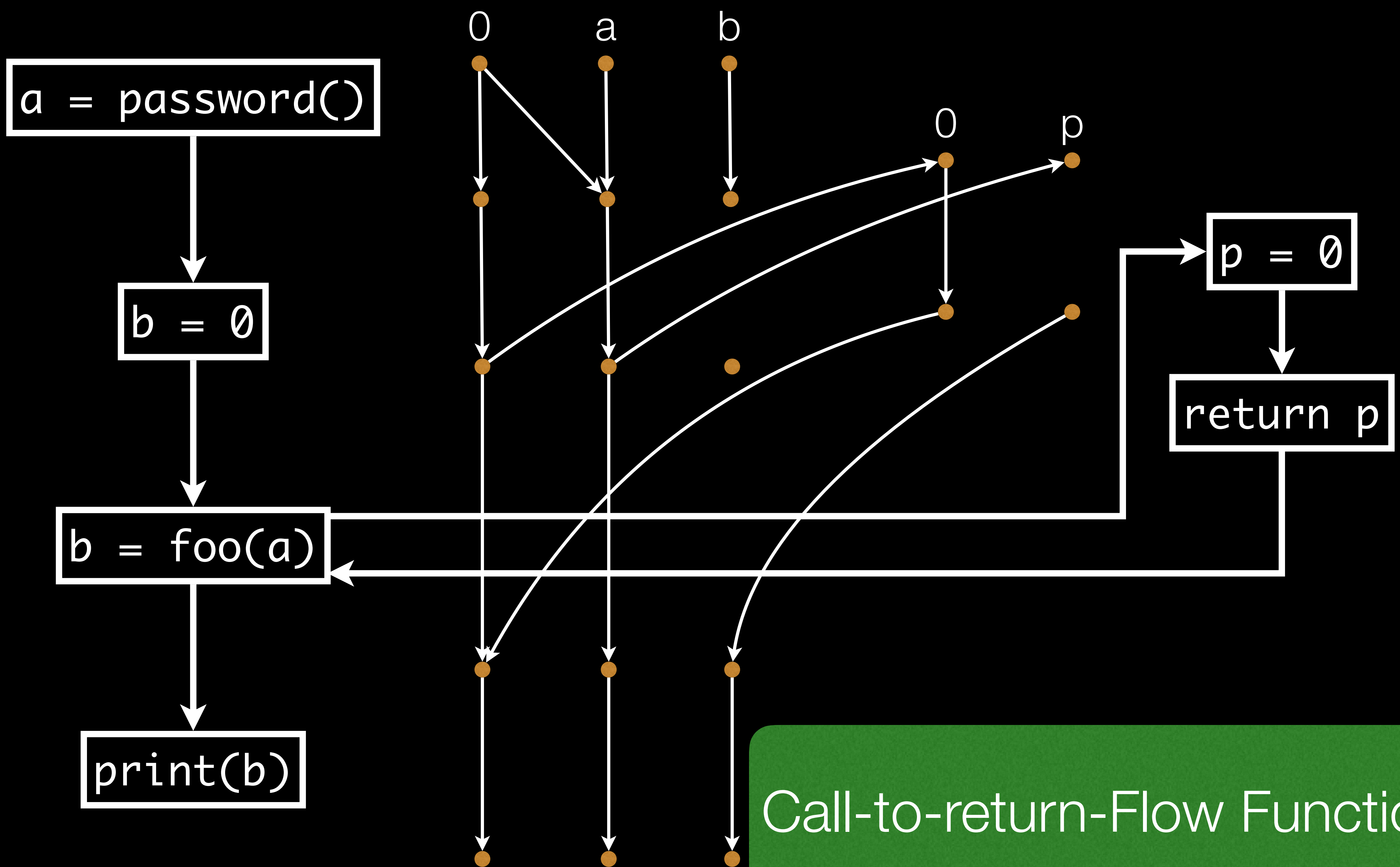
Return-Flow Function

Taint Analysis Example



Normal-Flow Function

Taint Analysis Example



Call-to-return-Flow Function

Flow Functions

Call-Flow Function

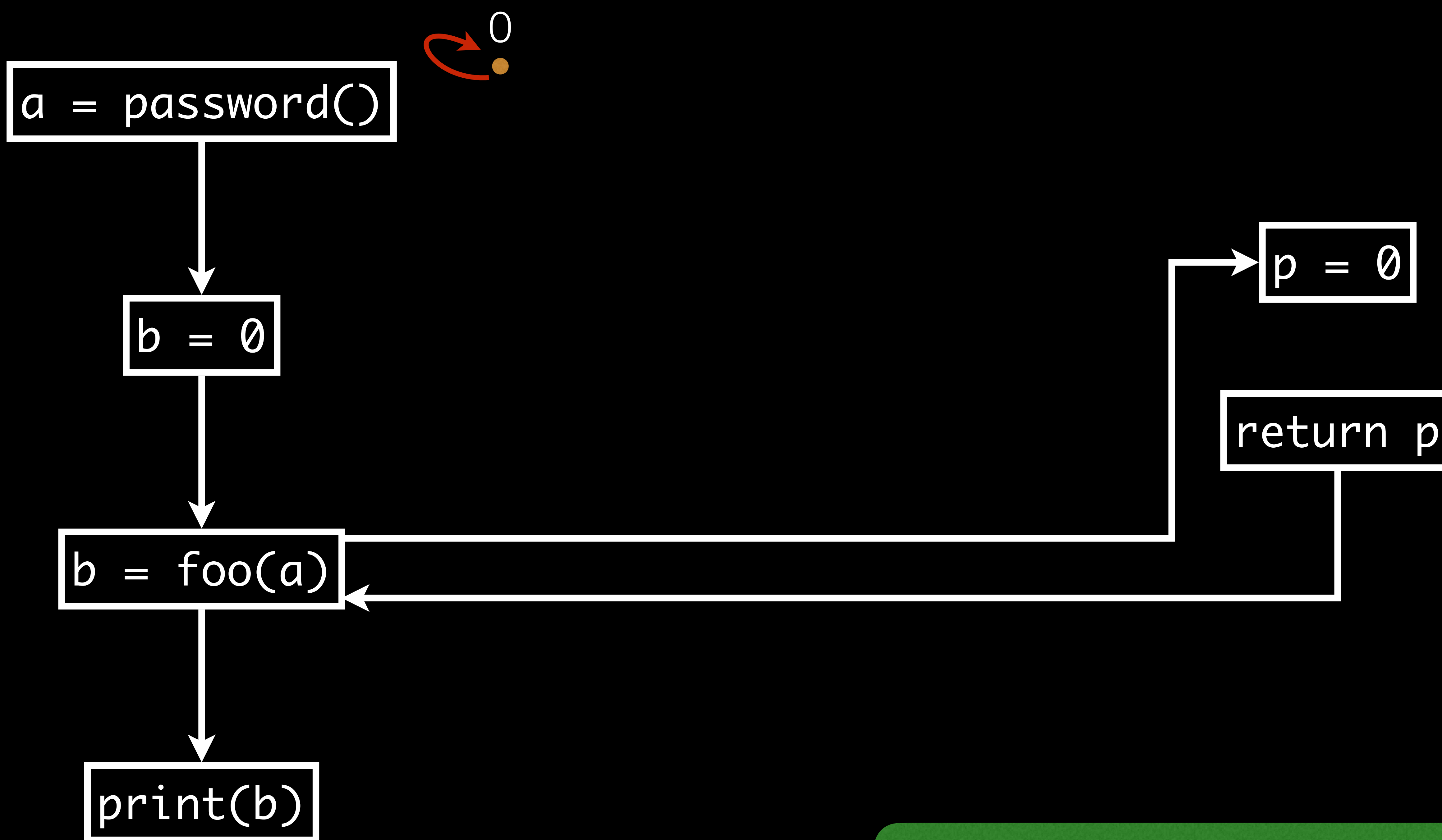
Normal-Flow Function

Return-Flow Function

id → Call-to-return-Flow Function

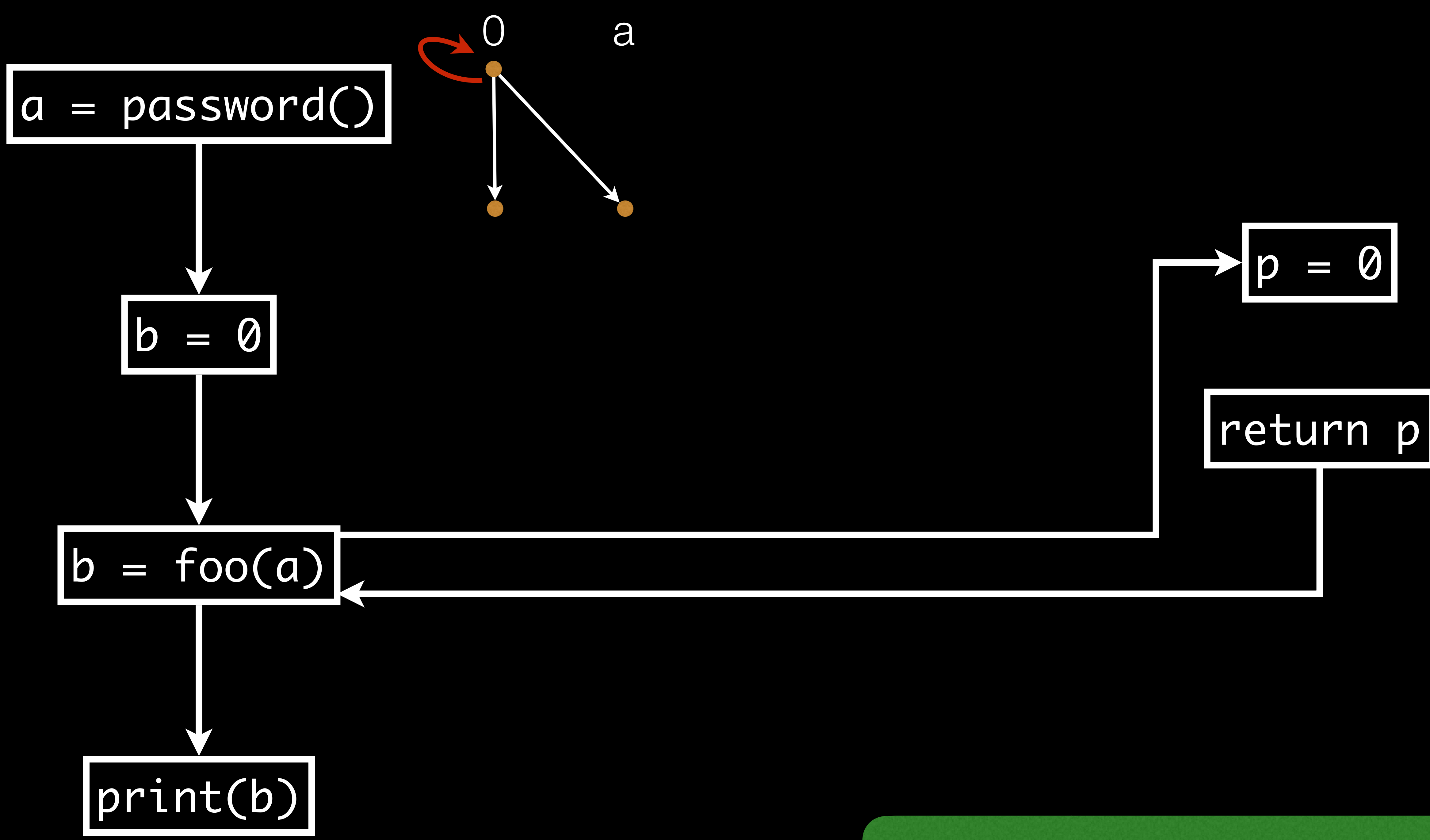
On-the-fly ESG

On-the-fly ESG



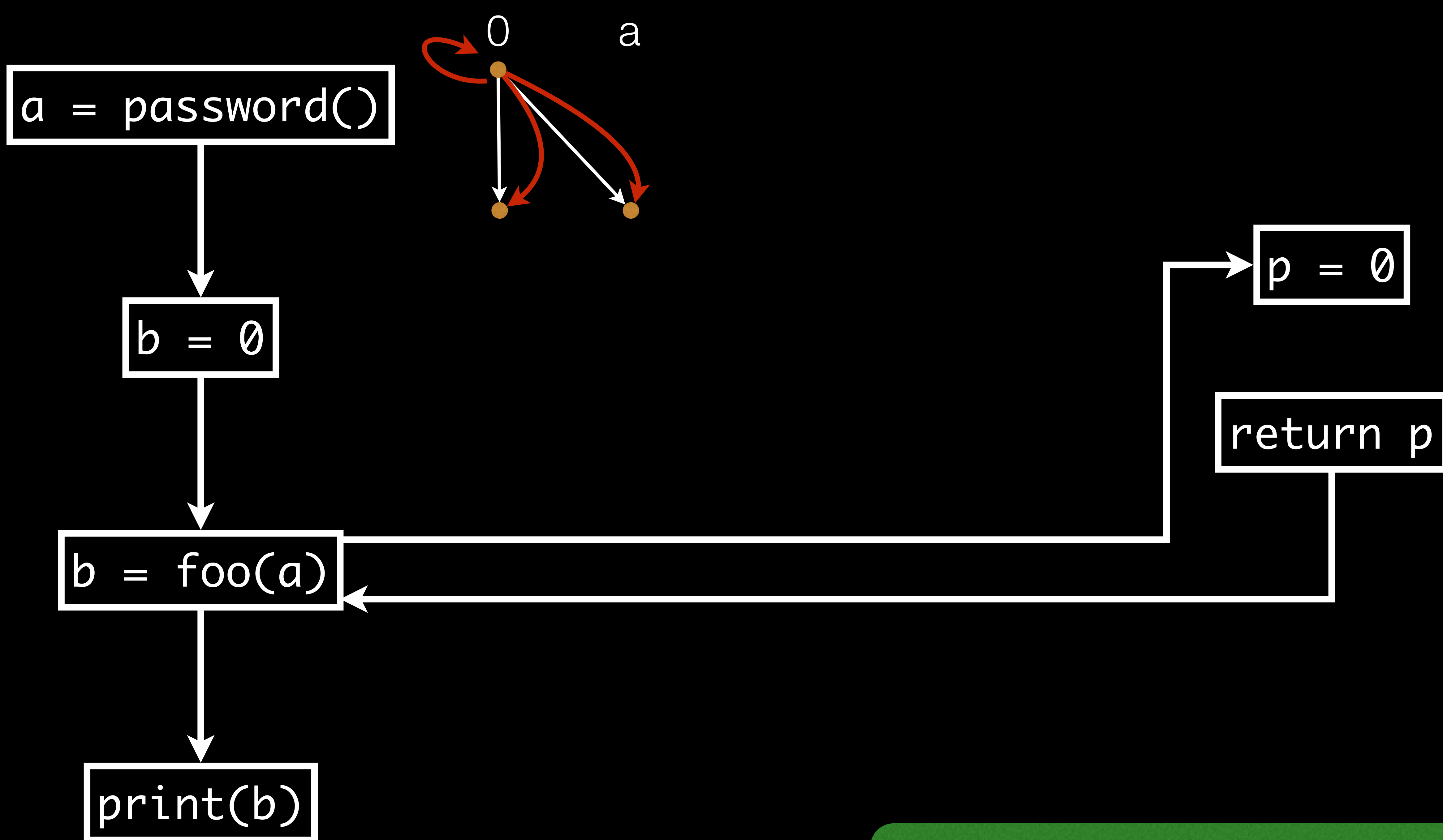
initial self-loop edge

On-the-fly ESG



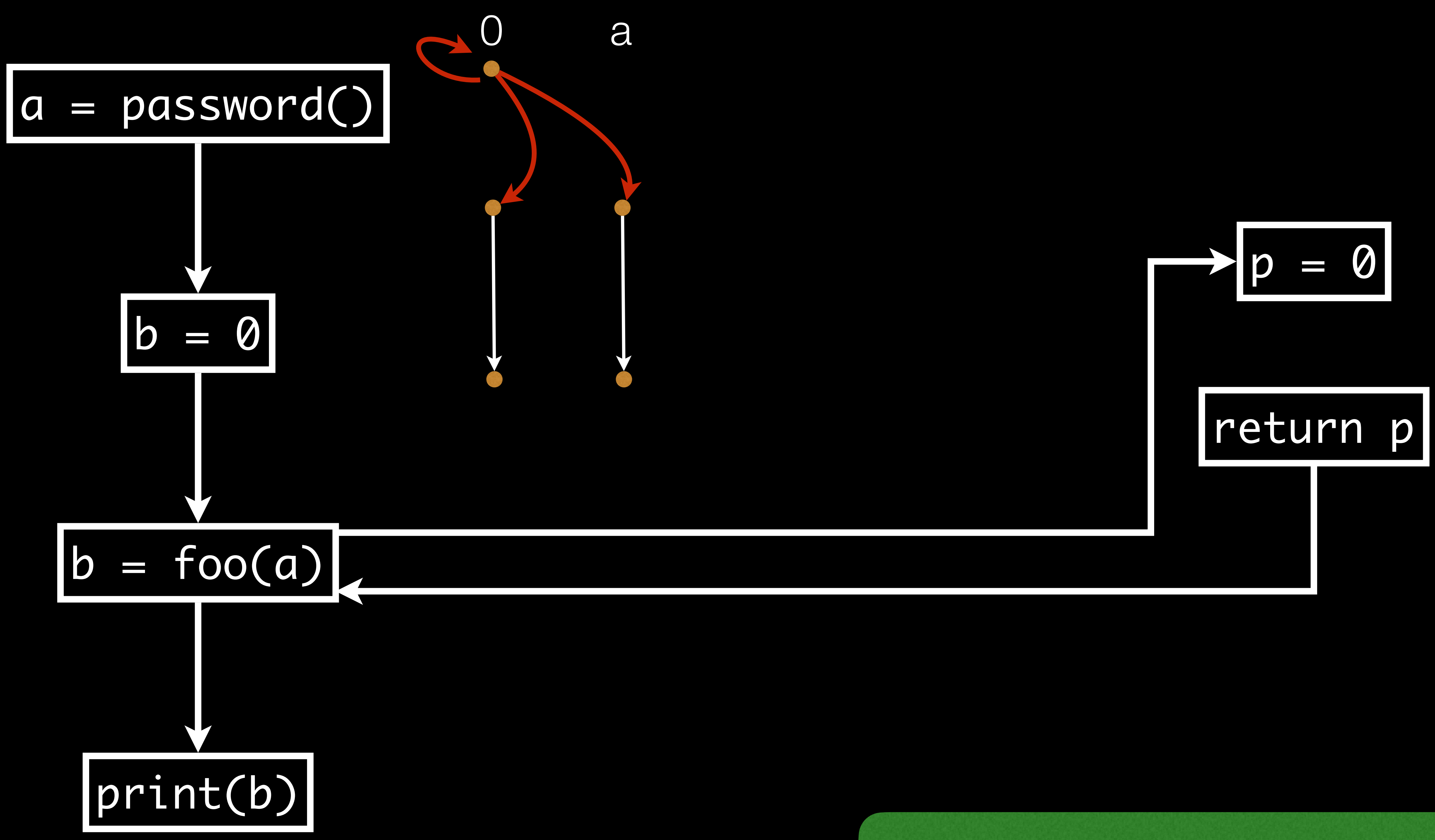
Normal-Flow

On-the-fly ESG



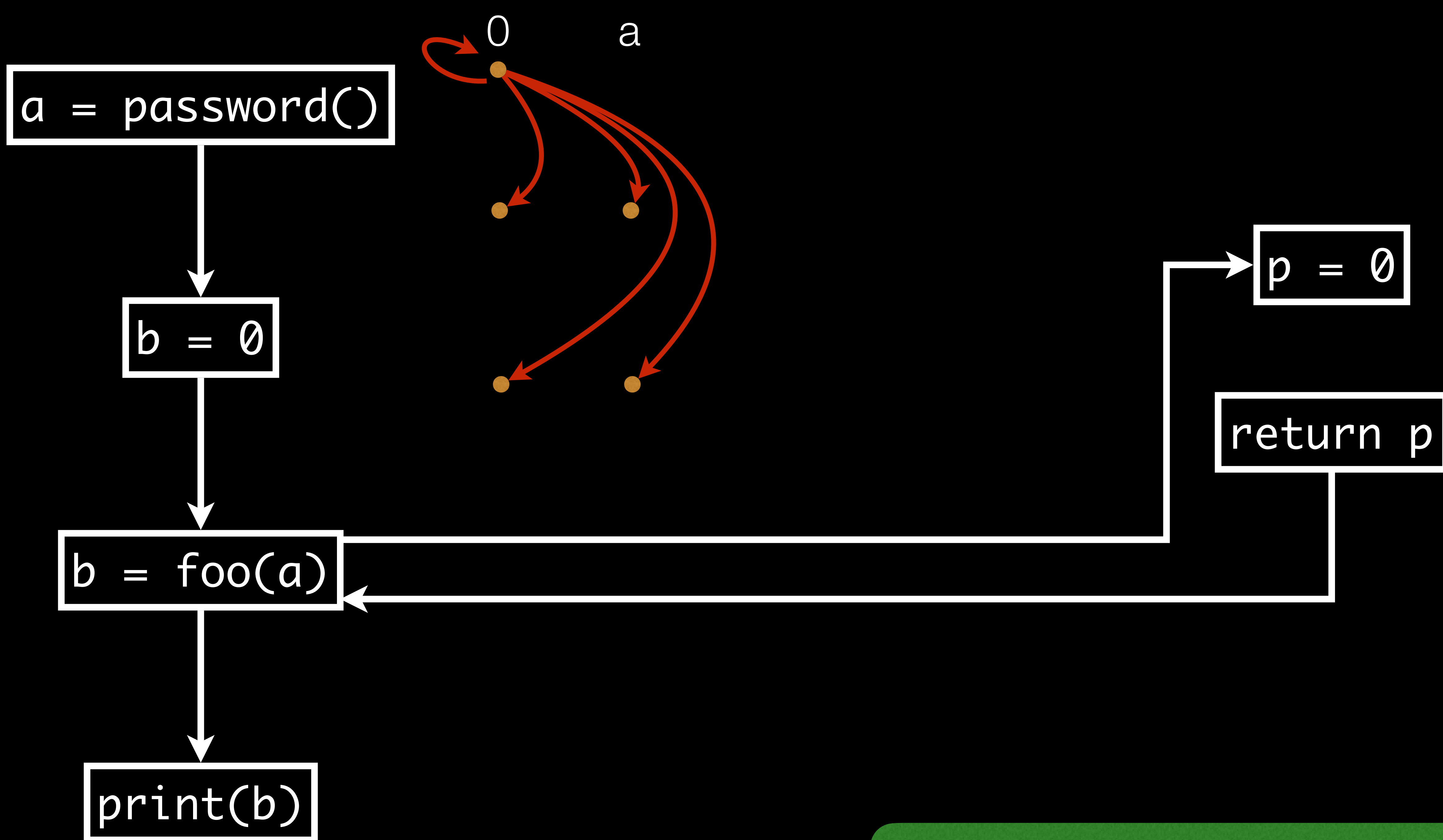
Path Edges

On-the-fly ESG



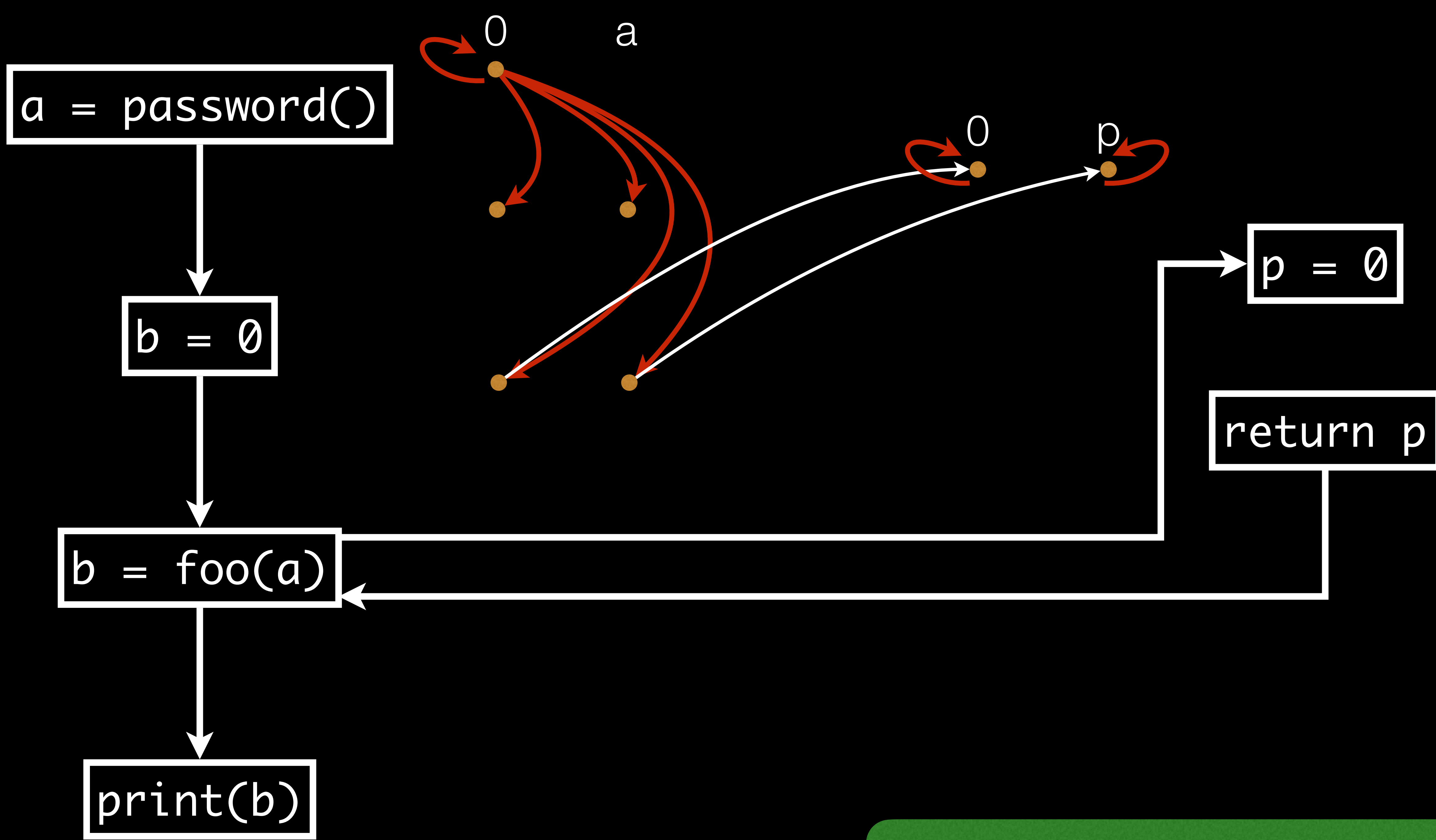
Normal-Flow

On-the-fly ESG



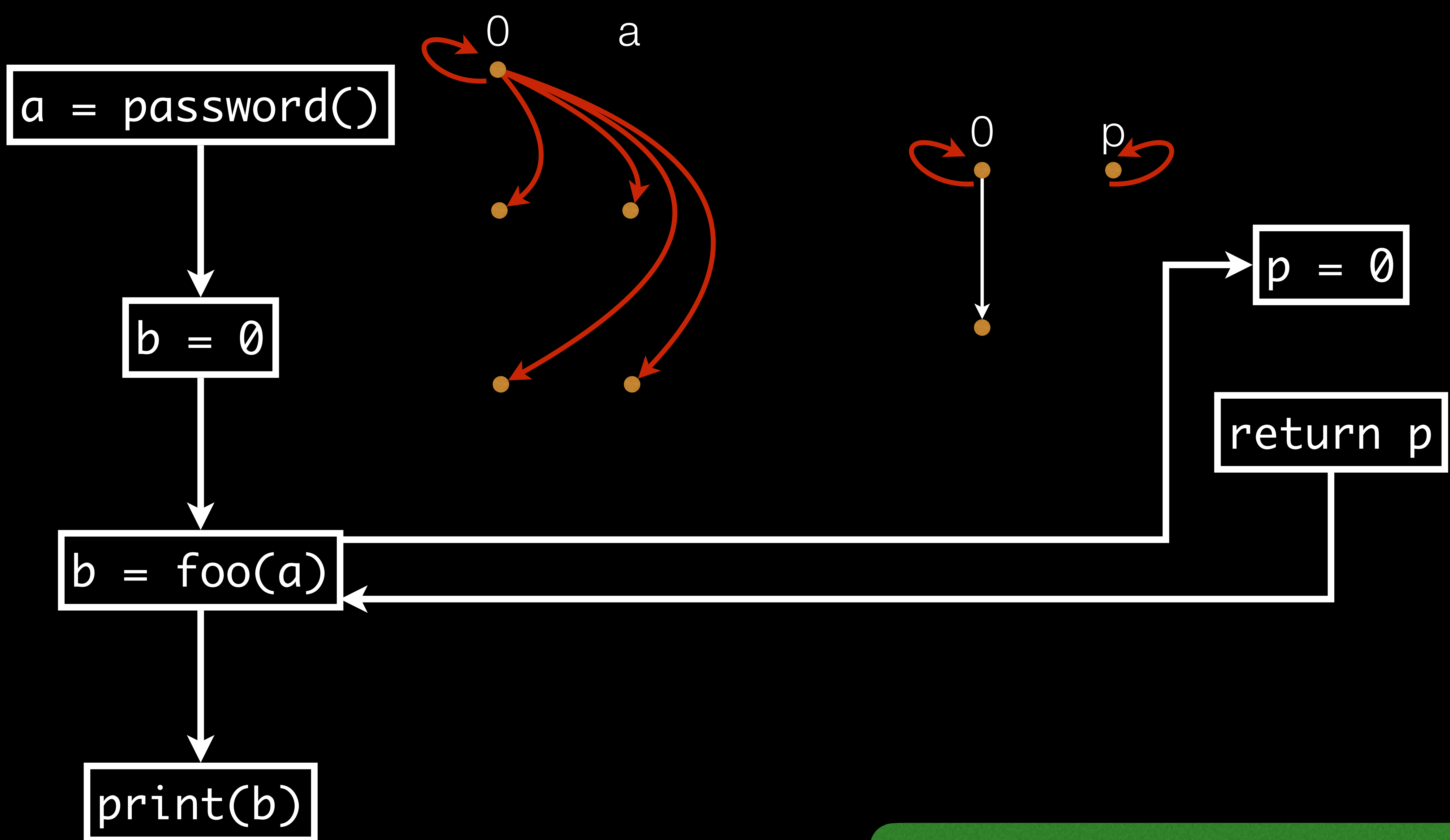
extend path edges

On-the-fly ESG



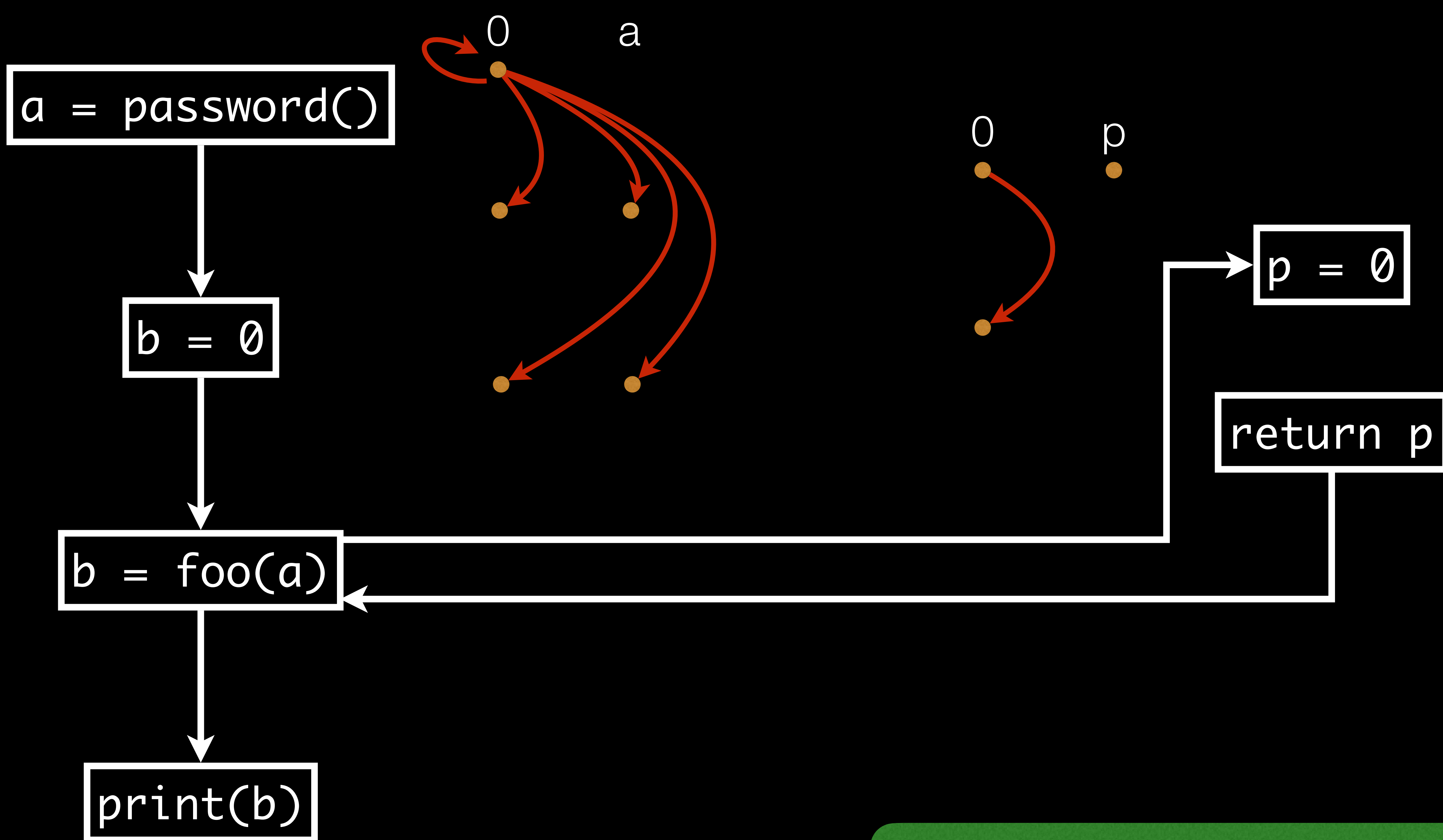
Call-Flow

On-the-fly ESG



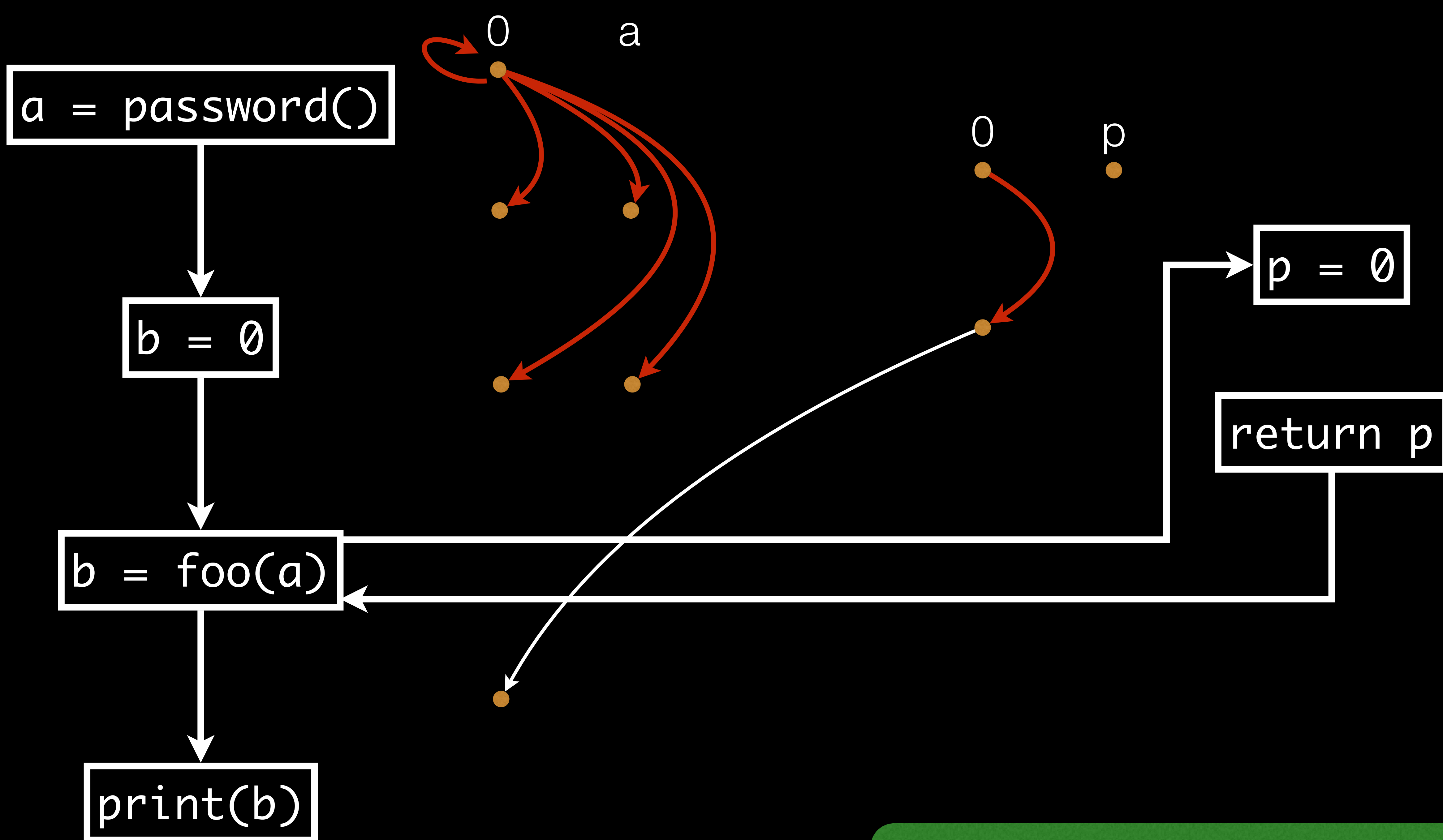
Normal-Flow

On-the-fly ESG



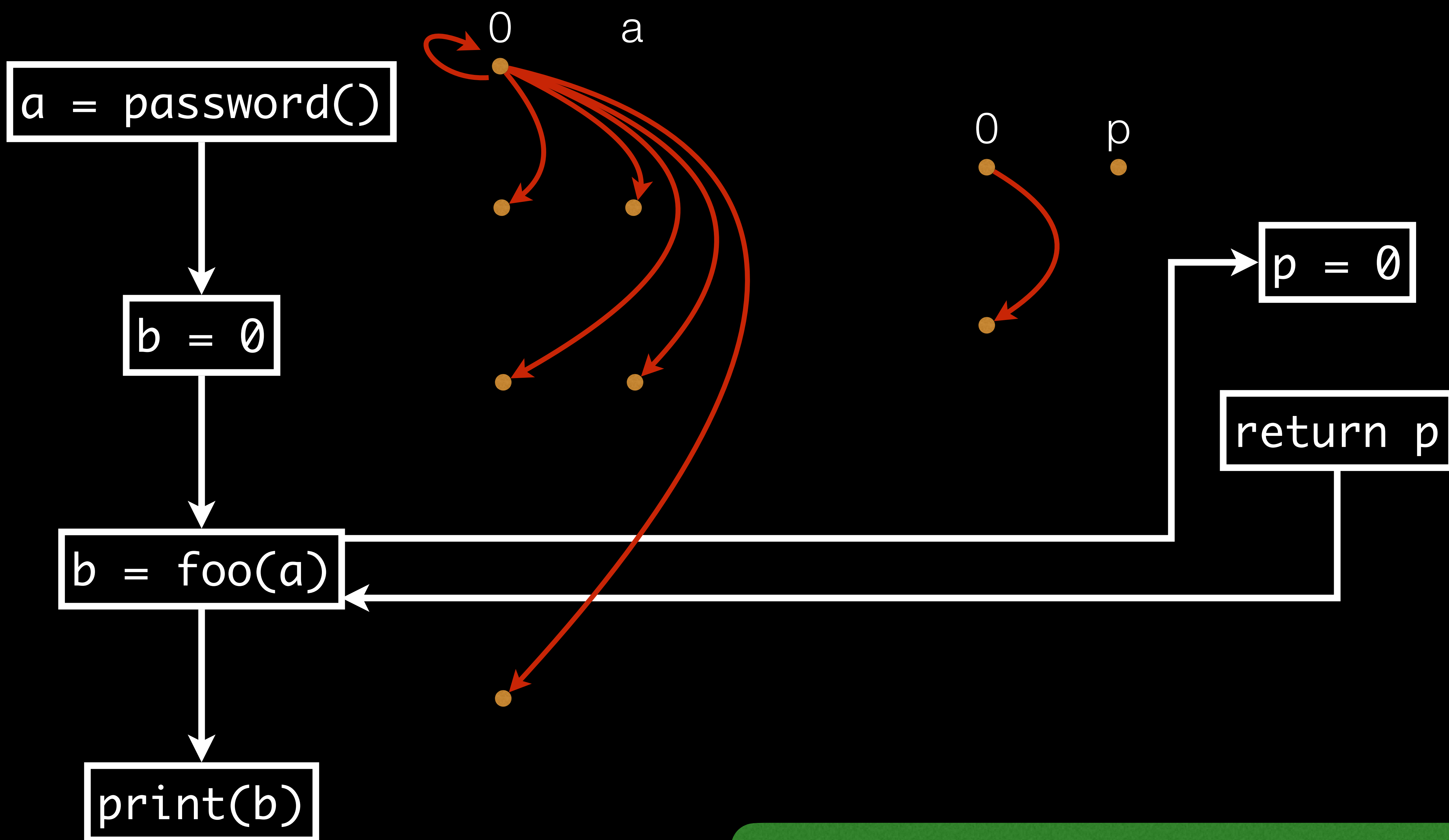
Method Summary

On-the-fly ESG



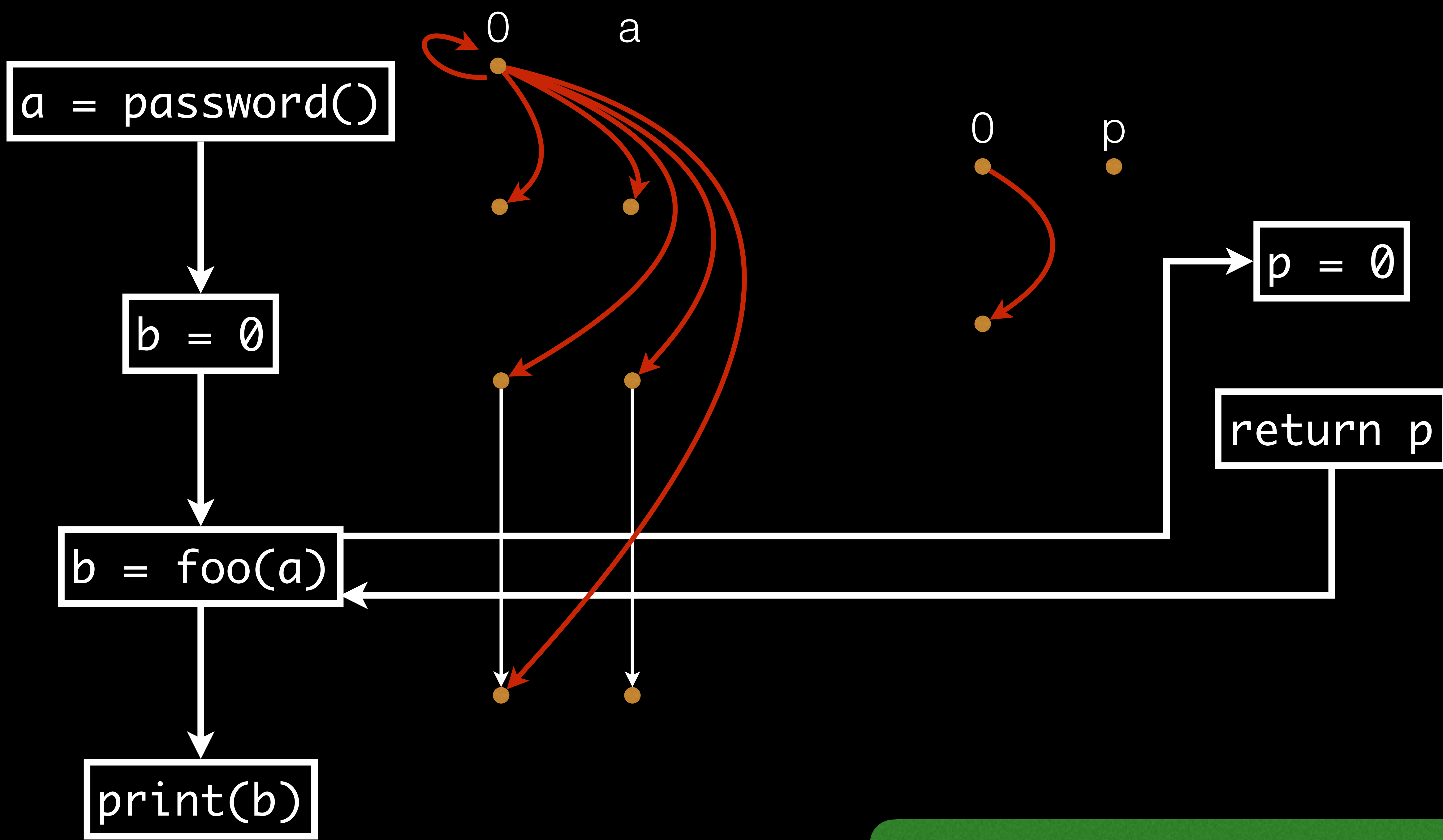
Return-Flow

On-the-fly ESG



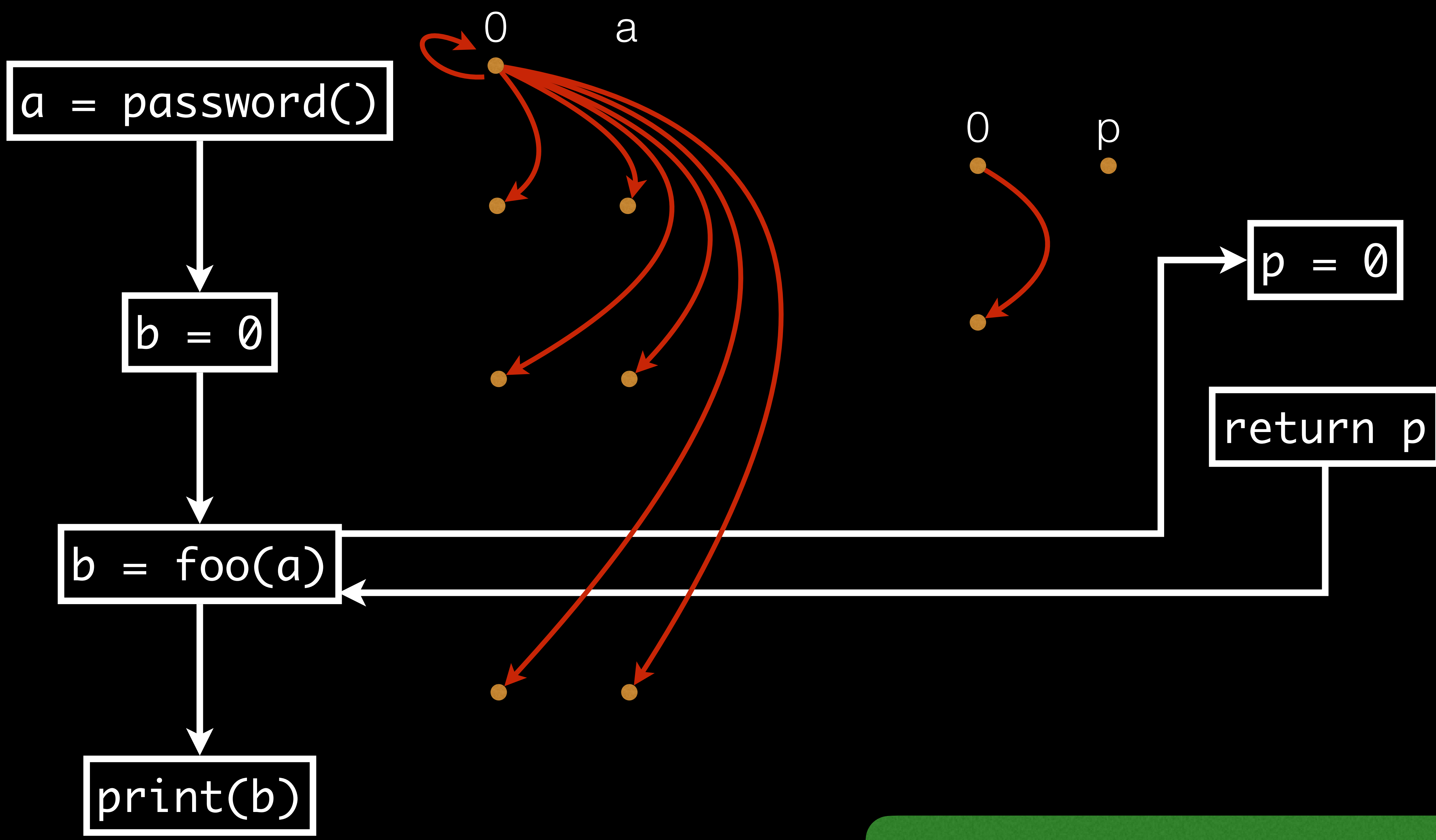
extend path edges in caller

On-the-fly ESG



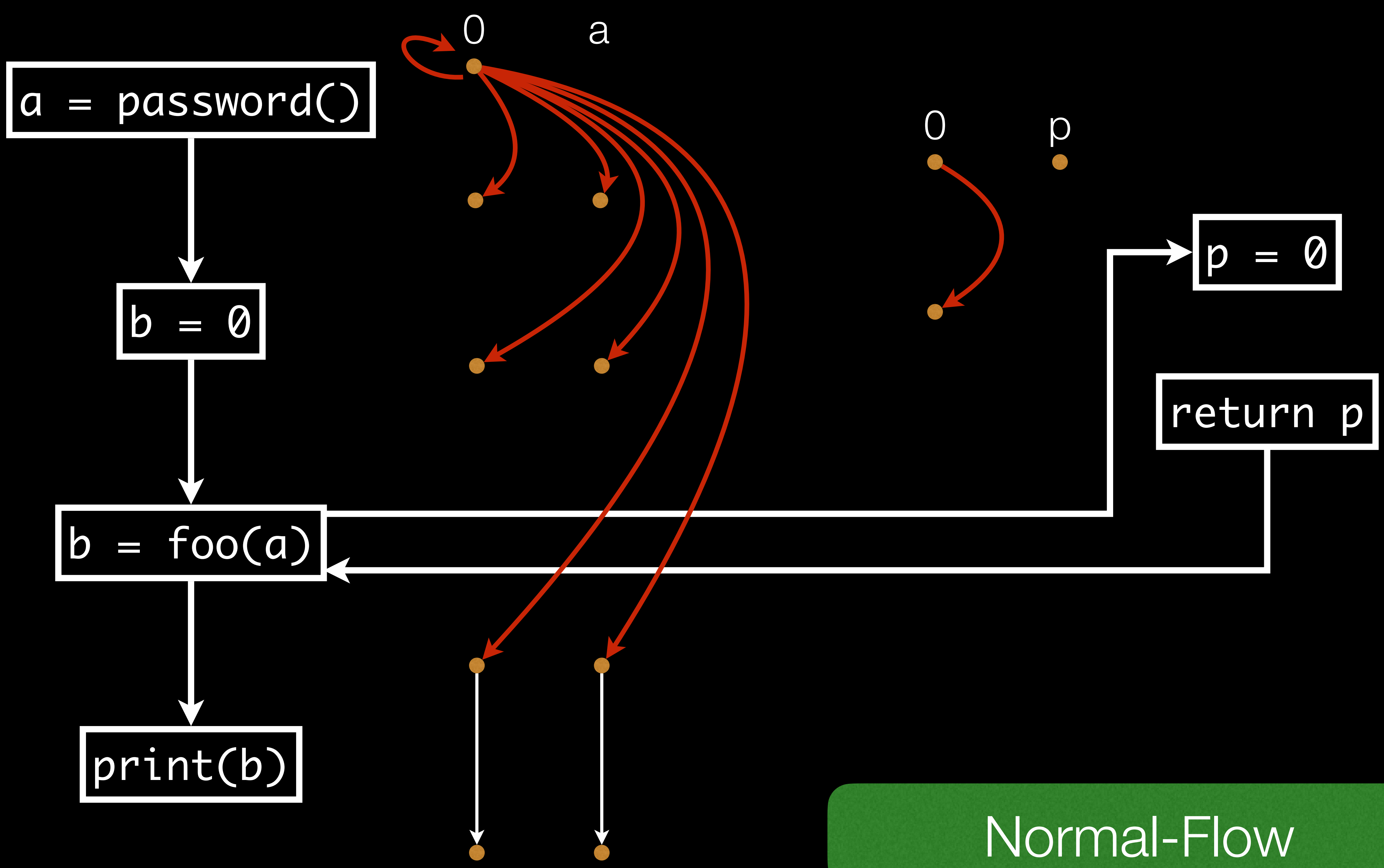
Call-to-return

On-the-fly ESG



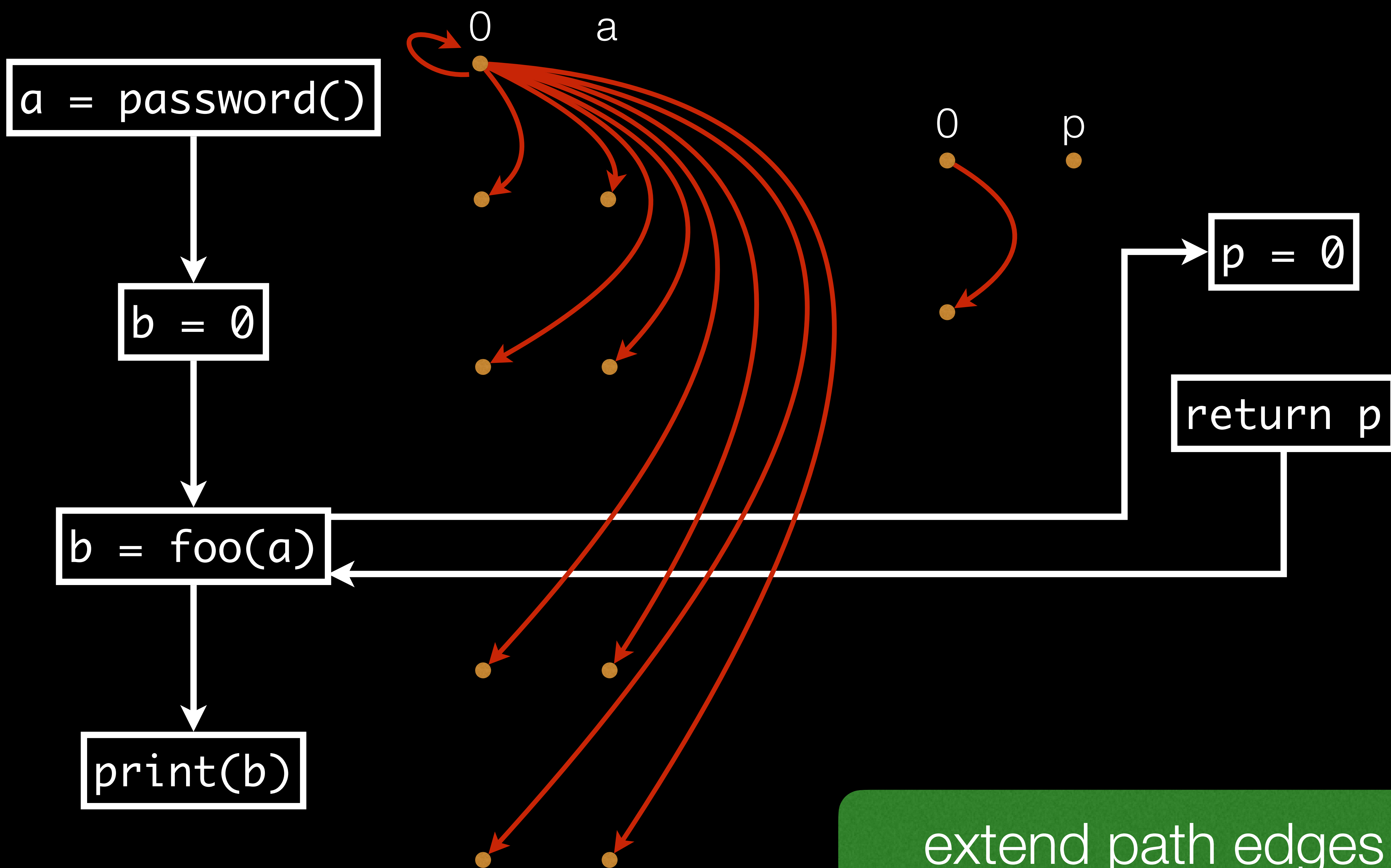
extend path edges

On-the-fly ESG



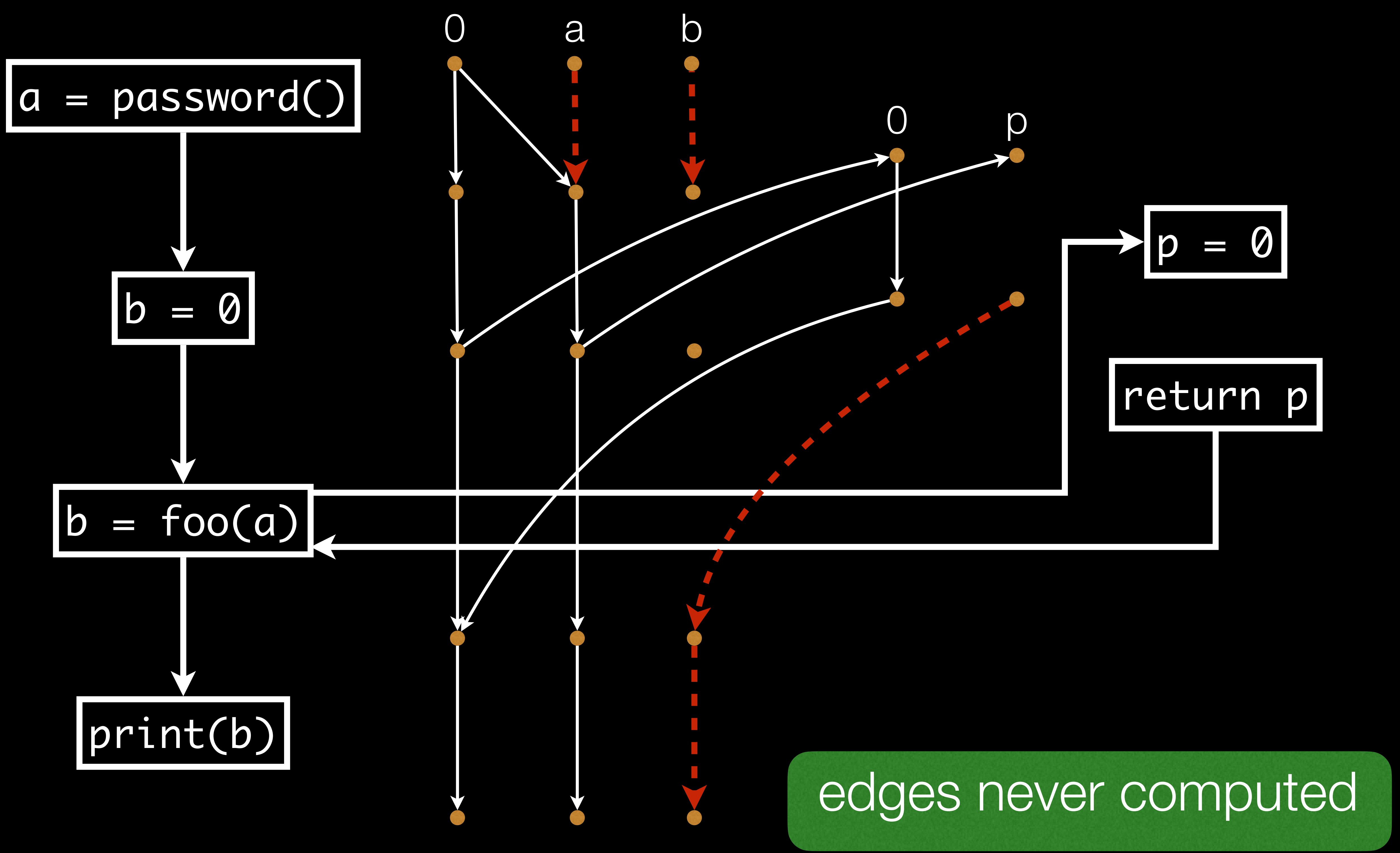
Normal-Flow

On-the-fly ESG



extend path edges

On-the-fly ESG



Tidbits!

Field-Sensitivity

a.f

Field-Sensitive

a.*

Field-Insensitive

A.f

Field-Based

Field-Based

A.f

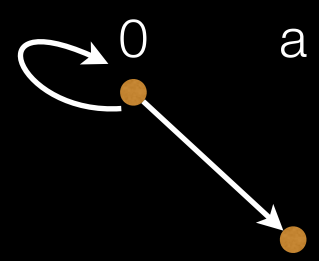
0

```
a = source();

b = new DS();

c = new DS();

b.f = a;

d = c.f;

sink(d);
```

D = Locals ∪ Fields

Field-Based

A.f

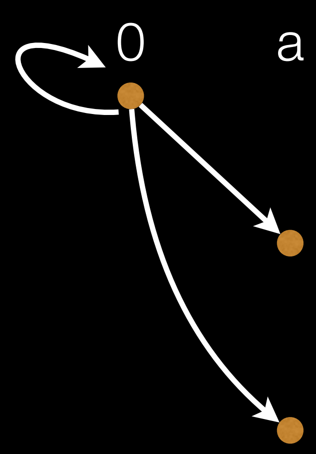



```
a = source();

b = new DS();

c = new DS();

b.f = a;

d = c.f;

sink(d);
```

D = Locals ∪ Fields

Field-Based

A.f

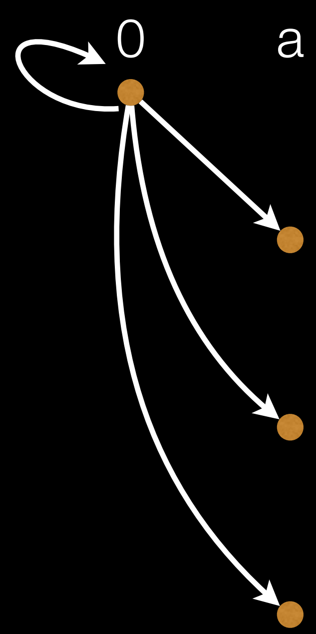



```
a = source();

b = new DS();

c = new DS();

b.f = a;

d = c.f;

sink(d);
```

D = Locals ∪ Fields

Field-Based

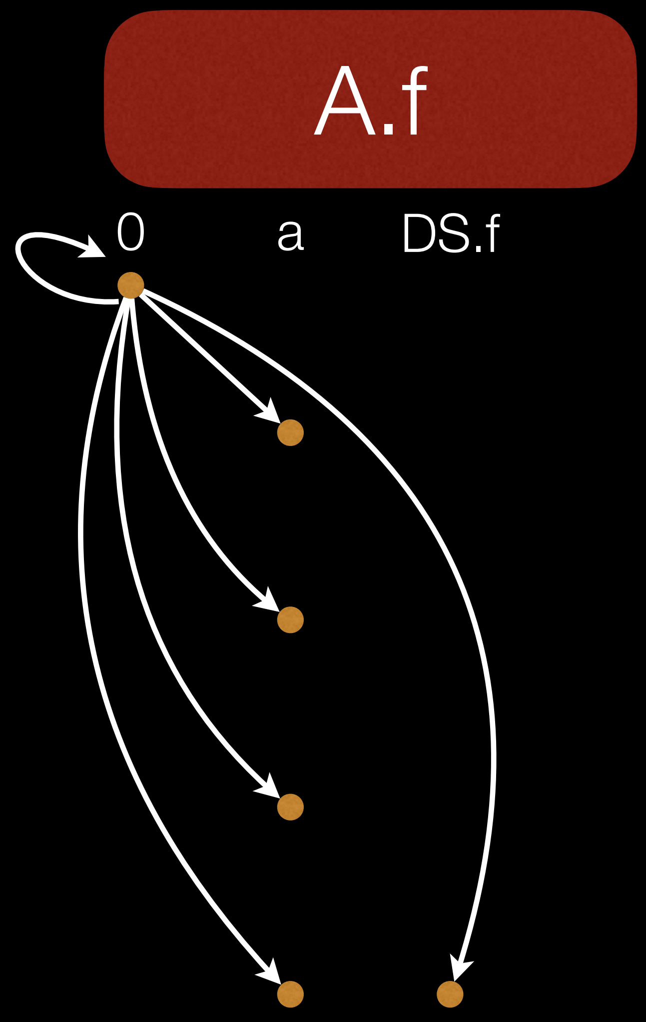
A.f




```
a = source();

b = new DS();

c = new DS();

b.f = a;

d = c.f;

sink(d);
```

D = Locals ∪ Fields

Field-Based

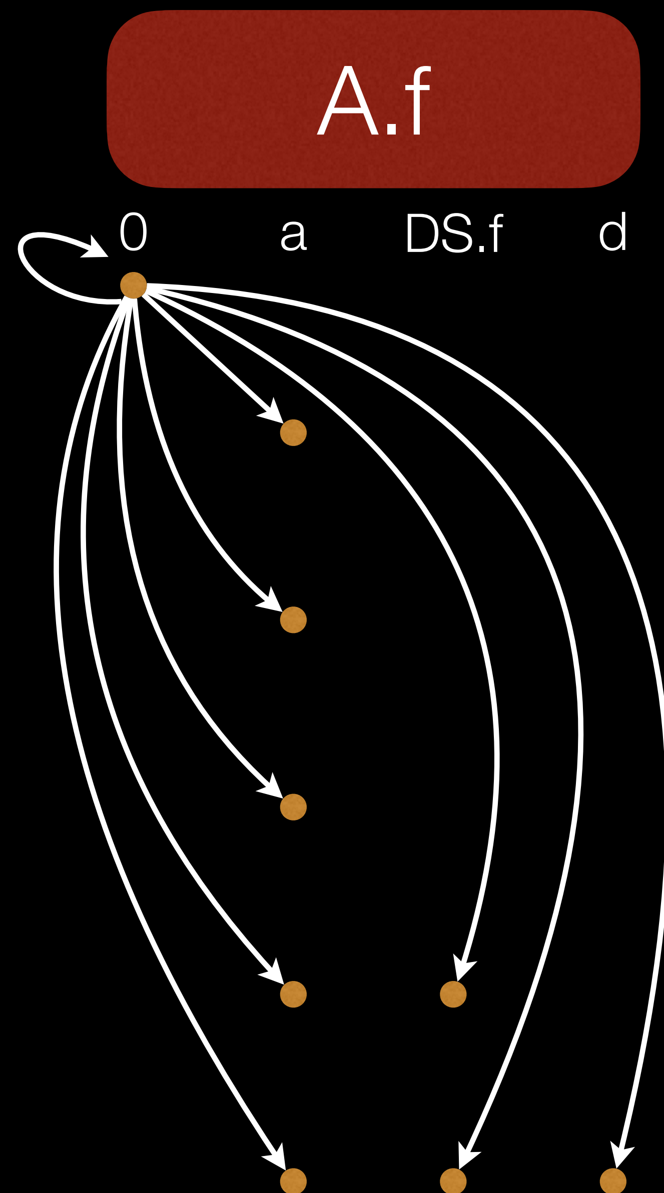



```
a = source();

b = new DS();

c = new DS();

b.f = a;

d = c.f;

sink(d);
```

D = Locals ∪ Fields

Field-Based

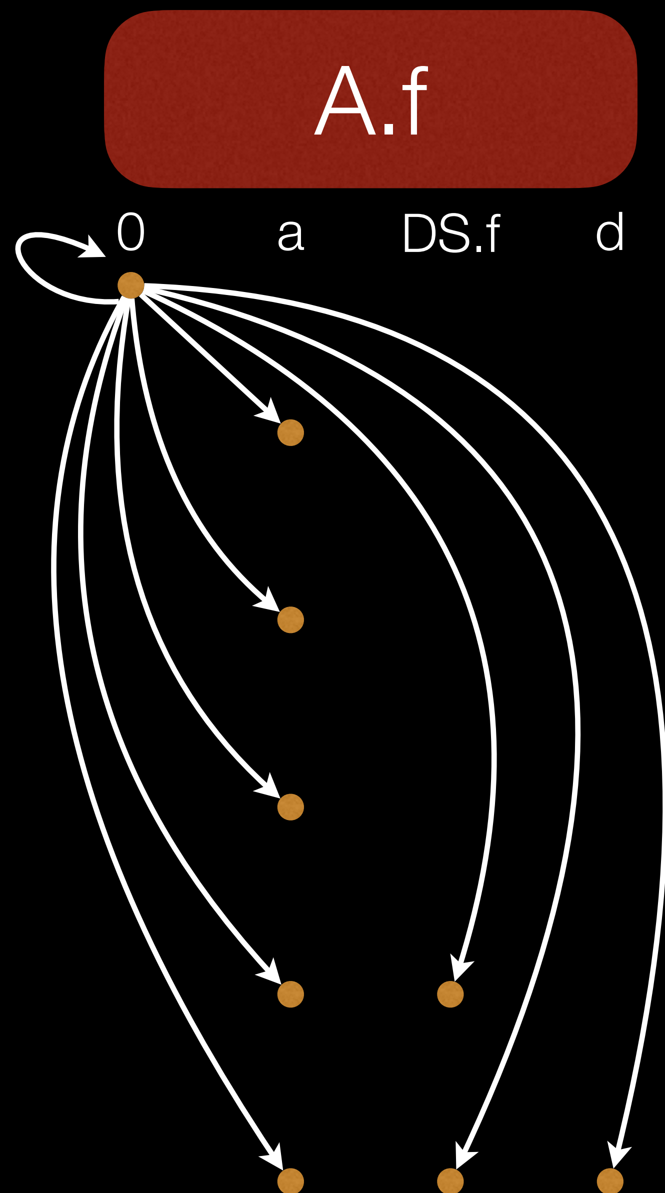
```
a = source();

b = new DS();

c = new DS();

b.f = a;

d = c.f;

sink(d);
```

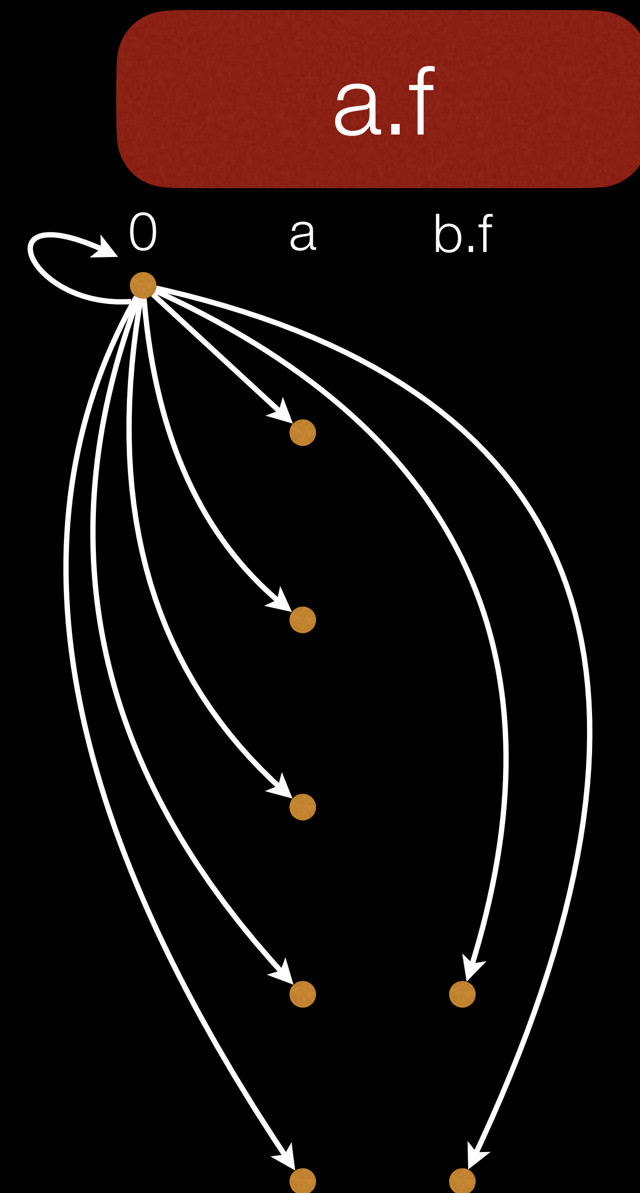


D = Locals ∪ Fields

Field-Sensitive

```
a = source();

b = new DS();

c = new DS();

b.f = a;

d = c.f;

sink(d);
```

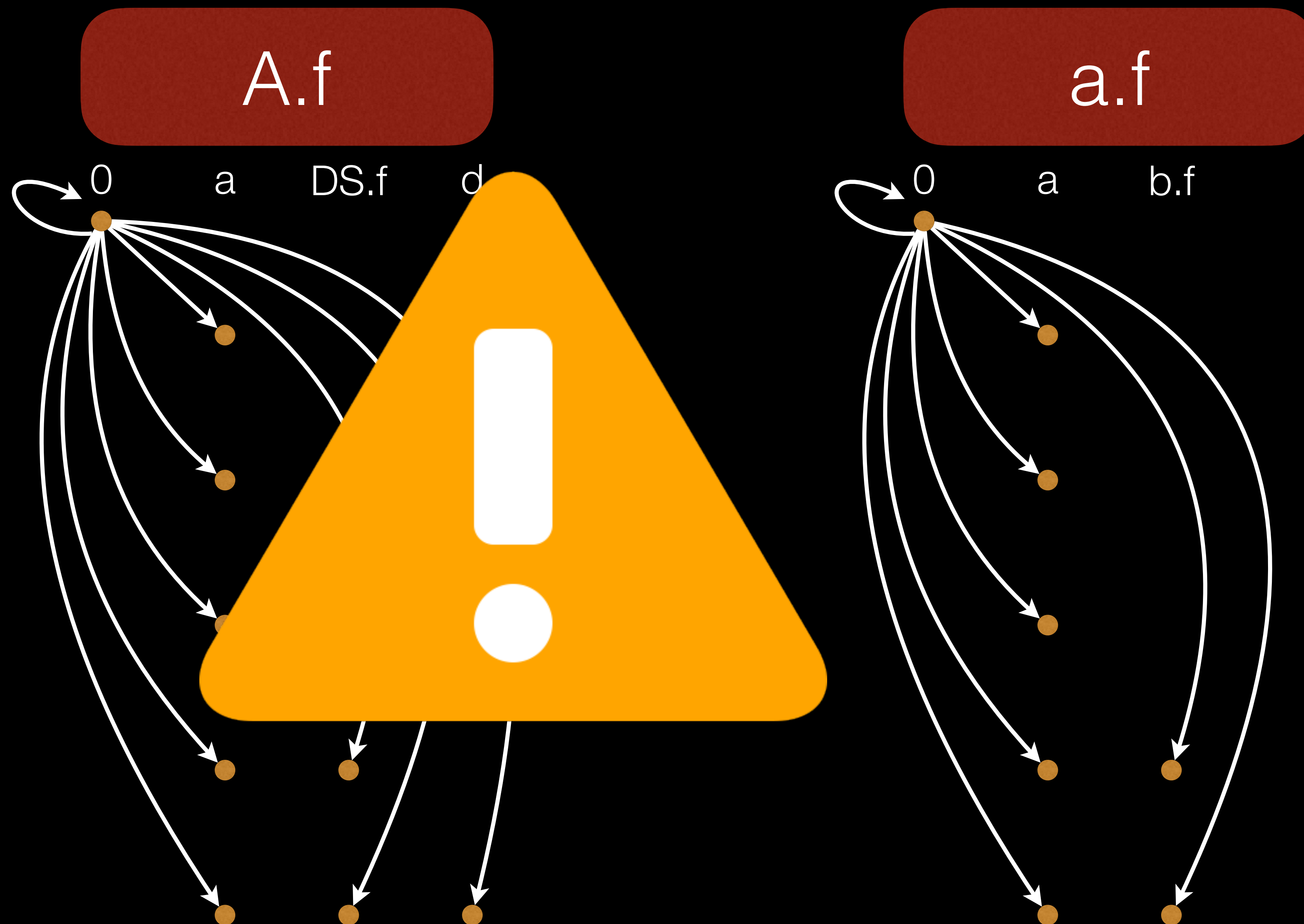




$D = \{l.f_1.f_2.f_3. \dots .f_n\}$
$l \in$ Locals, $f_i \in$ fields
$n \geq 0$

Field-Sensitive

```
a = source();

b = new DS();

c = new DS();

b.f = a;

d = c.f;

sink(d);
```

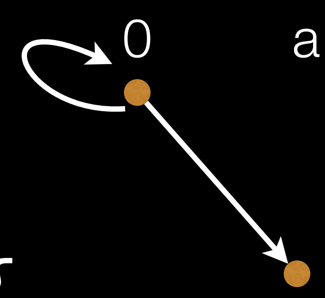


Infinite domain!!

$D = \{l.f_1.f_2.f_3. \ldots .f_n\}$
$l \in \text{Locals}, f_i \in \text{fields}$
$n \geq 0$

Field-Sensitive

a.f

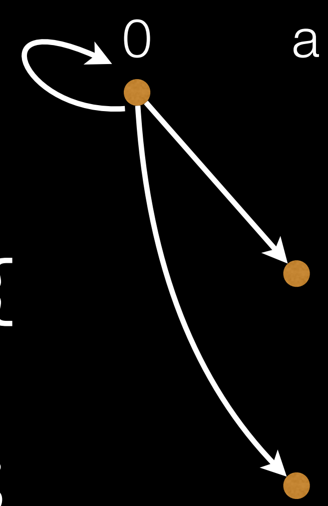



```
a = source();

while(rand()) {

  b = new DS();

  b.f = a;

  a = b;
}

sink(d);
```

$D = \{l.f_1.f_2.f_3. \dots .f_n\}$
$l \in Locals$, $f_i \in fields$
$n \geq 0$

Field-Sensitive

a.f

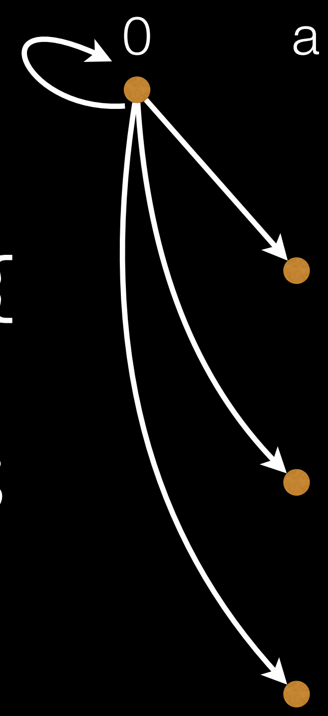



```
a = source();

while(rand()) {

  b = new DS();

  b.f = a;

  a = b;
}

sink(d);
```

$D = \{l.f_1.f_2.f_3. \ \dots \ .f_n\}$
$l \in \text{Locals}, f_i \in \text{fields}$
$n \geq 0$

Field-Sensitive

a.f

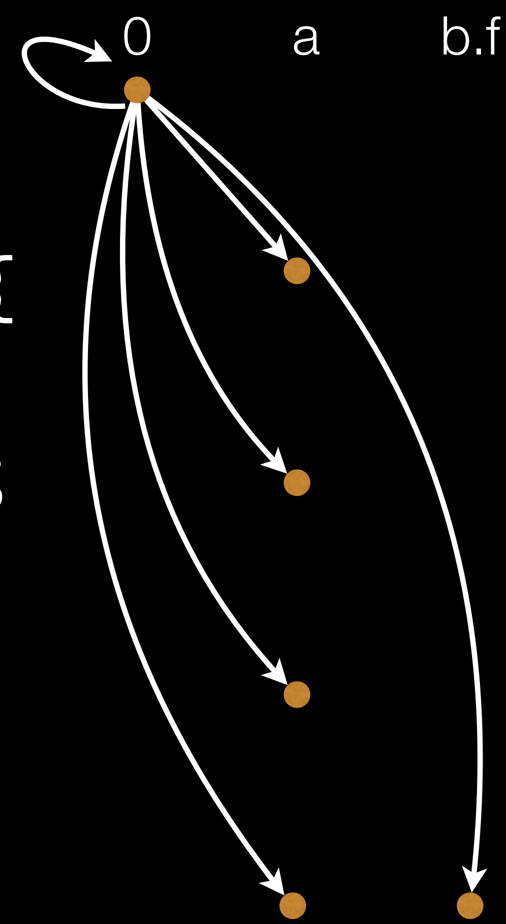



```
a = source();

while(rand()) {

  b = new DS();

  b.f = a;

  a = b;
}

sink(d);
```

$D = \{l.f_1.f_2.f_3.\ \ldots\ .f_n\}$
$l \in$ Locals, $f_i \in$ fields
$n \geq 0$

Field-Sensitive

a.f



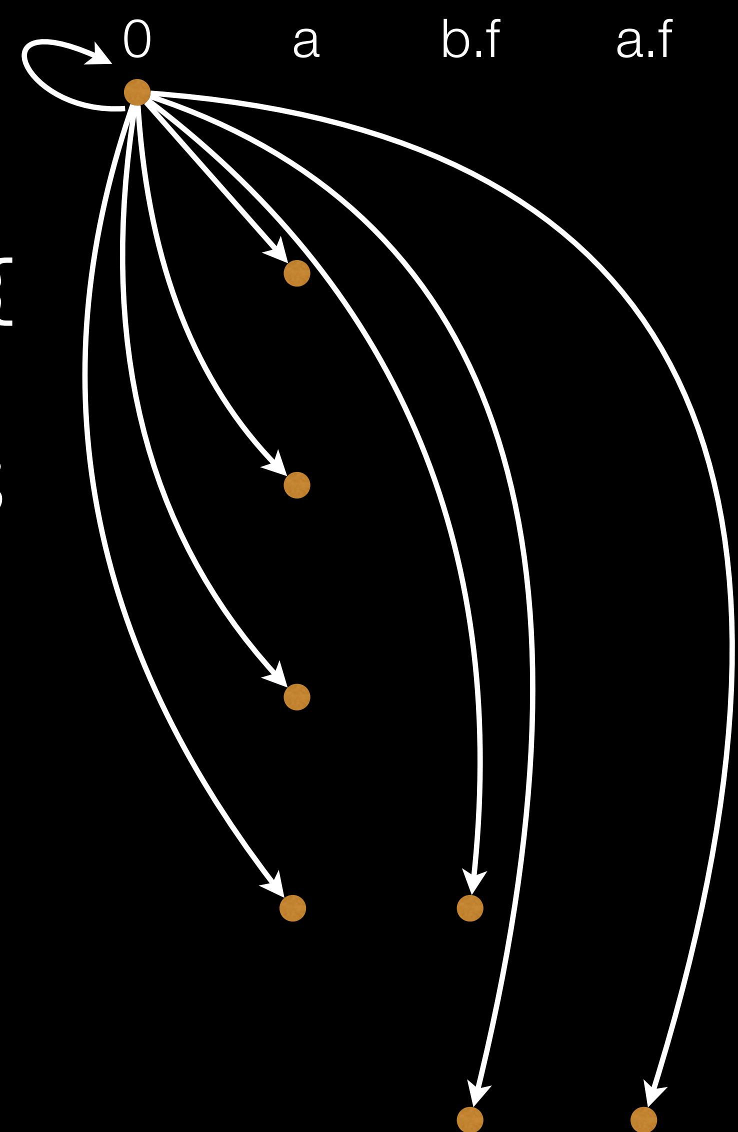

```
a = source();

while(rand()) {

  b = new DS();

  b.f = a;

  a = b;
}

sink(d);
```

$D = \{l.f_1.f_2.f_3. \ \dots \ .f_n\}$
$l \in$ Locals, $f_i \in$ fields
$n \geq 0$

Field-Sensitive

a.f

```
a = source();

while(rand()) {

  b = new DS();

  b.f = a;

  a = b;
}

sink(d);
```

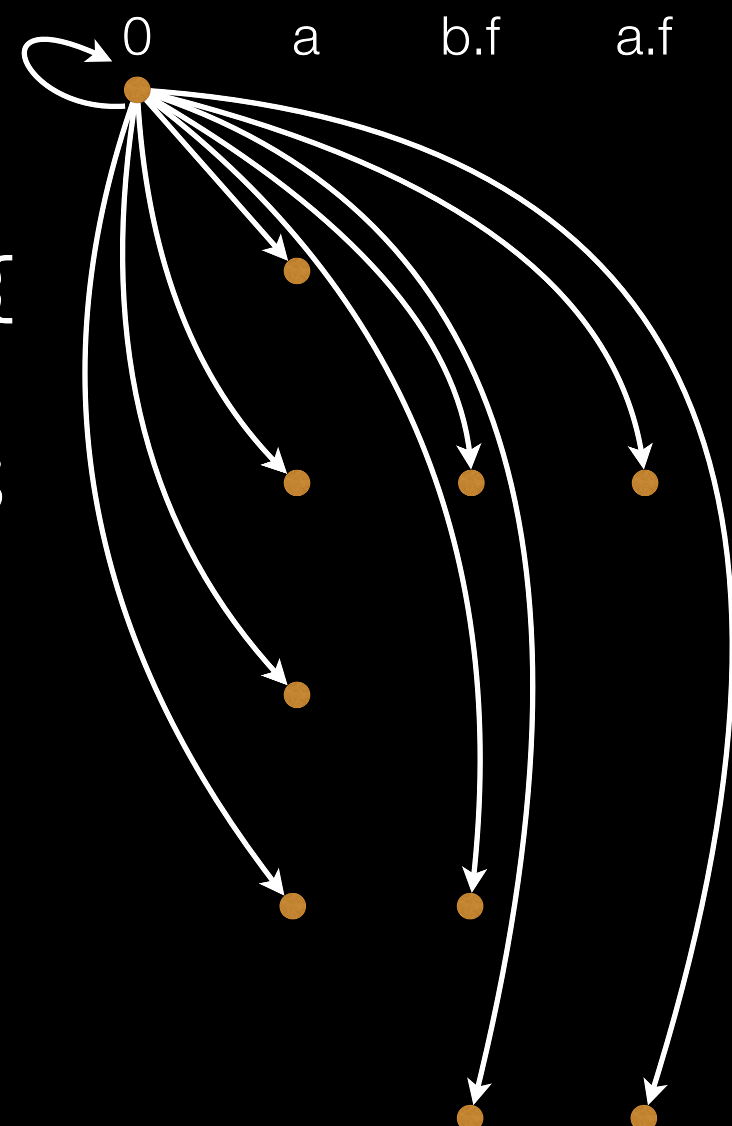


$D = \{l.f_1.f_2.f_3. \ldots .f_n\}$
$l \in$ Locals, $f_i \in$ fields
$n \geq 0$

Field-Sensitive

a.f

```
a = source();

while(rand()) {

  b = new DS();

  b.f = a;

  a = b;
}

sink(d);
```



$D = \{l.f_1.f_2.f_3. \ldots .f_n\}$

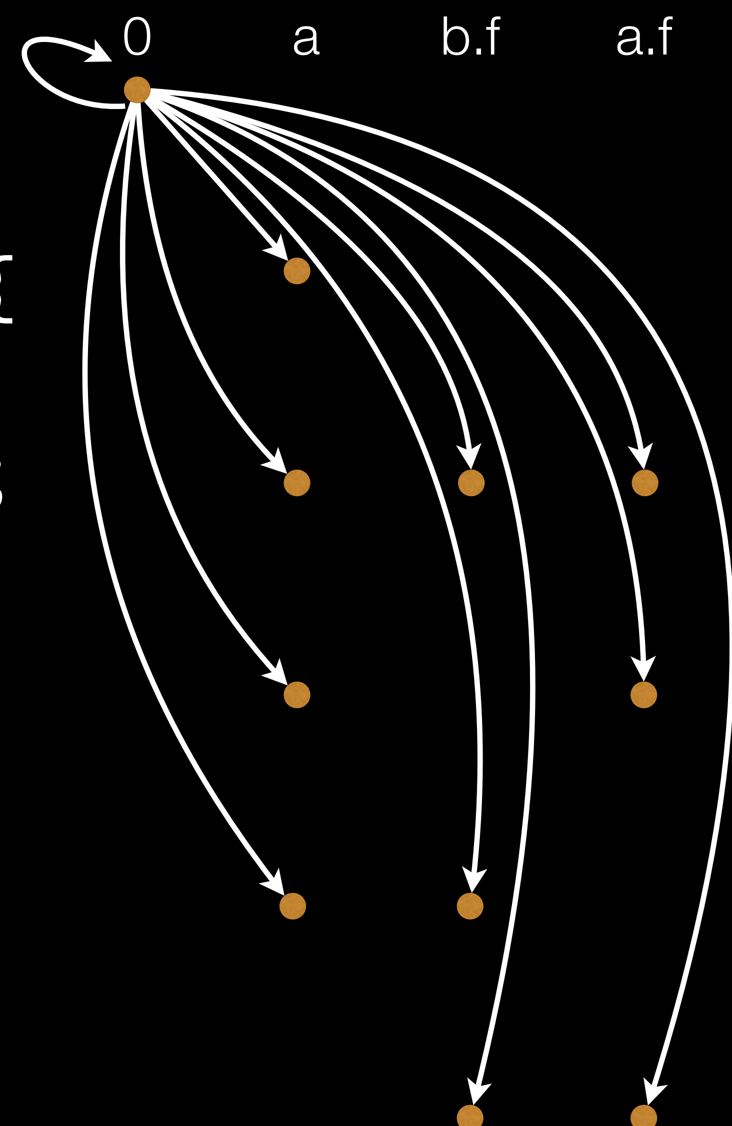
$l \in$ Locals, $f_i \in$ fields

$n \geq 0$

Field-Sensitive

a.f

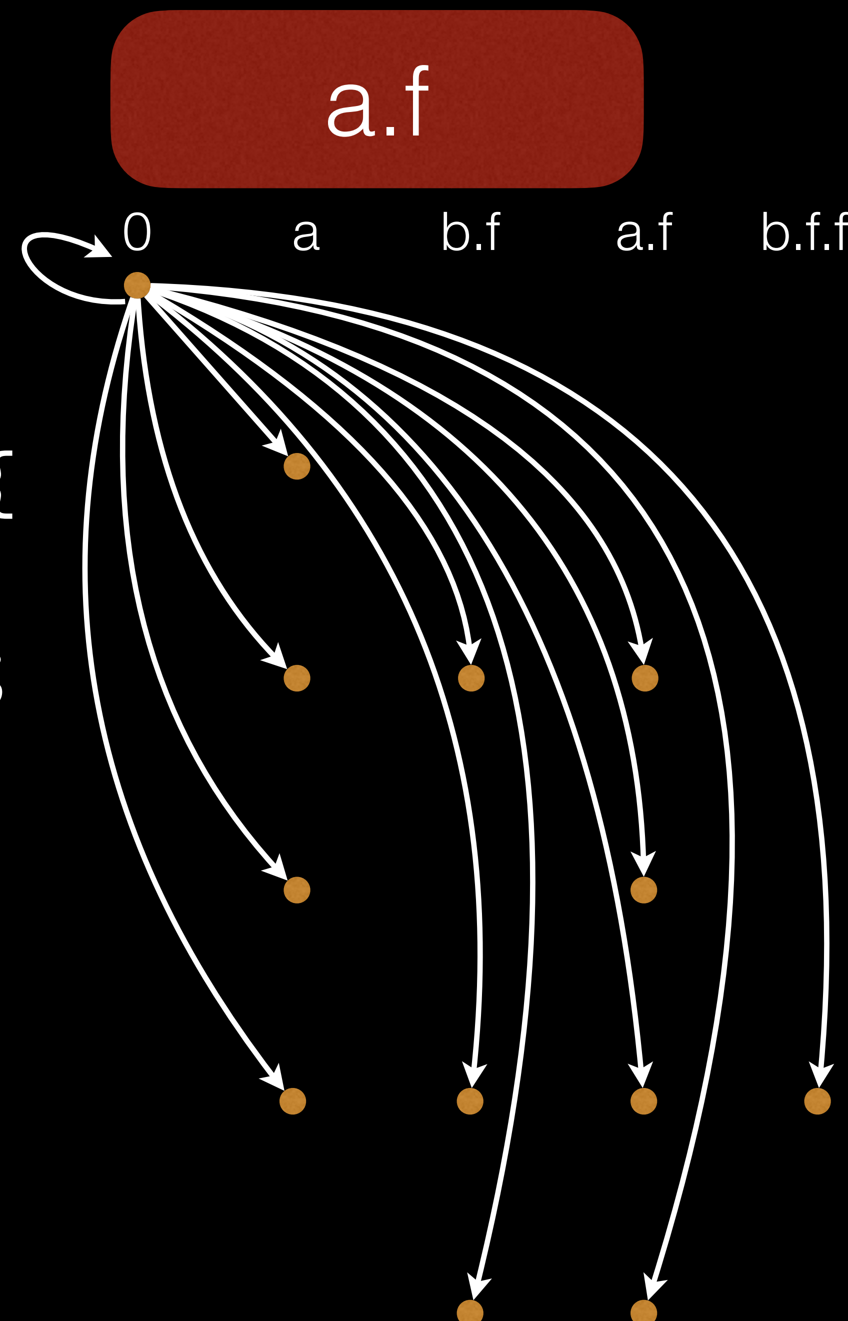
```
a = source();

while(rand()) {

  b = new DS();

  b.f = a;

  a = b;
}

sink(d);
```



$D = \{l.f_1.f_2.f_3. \ldots .f_n\}$
$l \in \text{Locals}, f_i \in \text{fields}$
$n \geq 0$

Field-Sensitive

```
a = source();

while(rand()) {

  b = new DS();

  b.f = a;

  a = b;
}

sink(d);
```

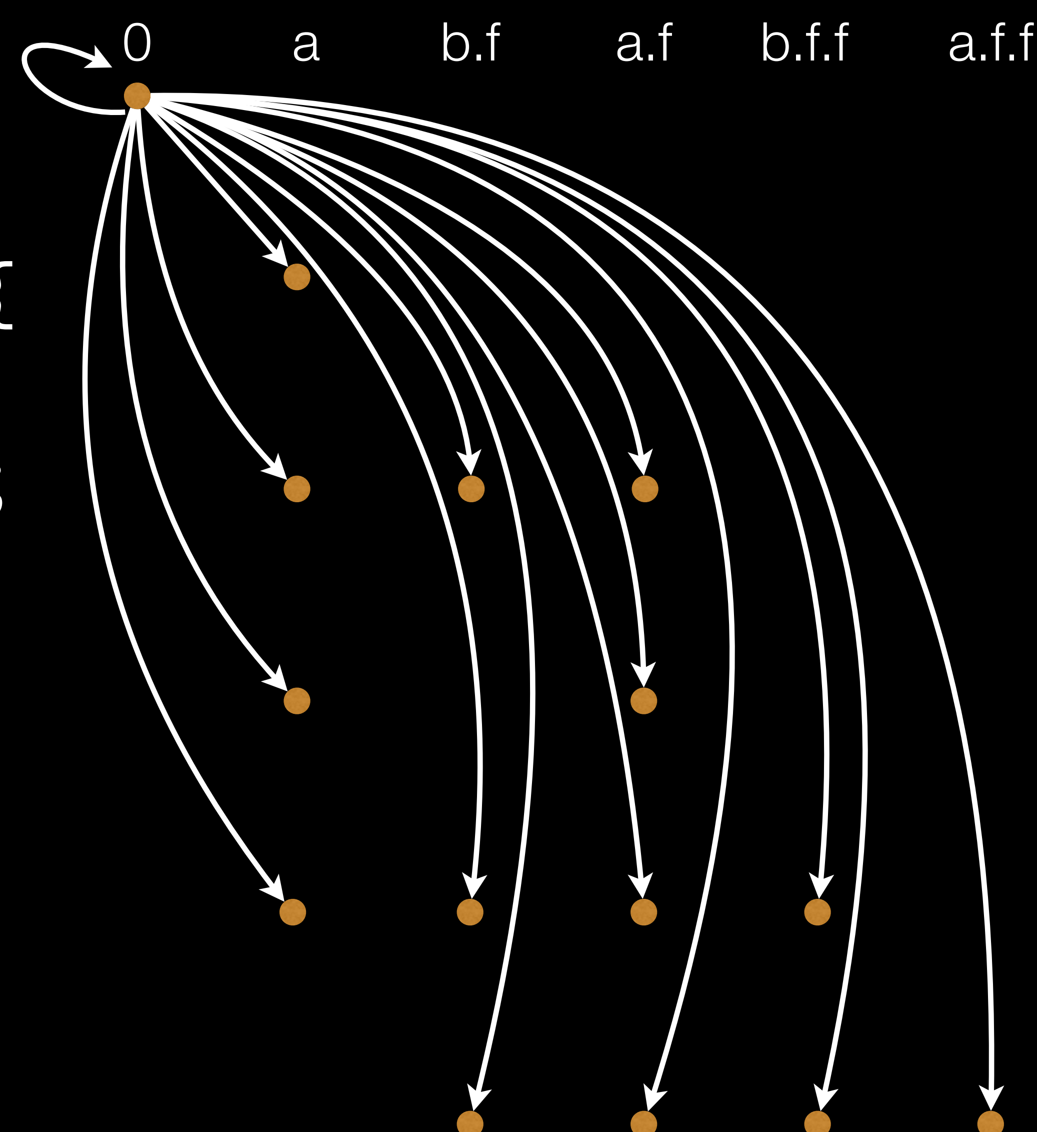


$D = \{l.f_1.f_2.f_3. \ldots .f_n\}$
$l \in Locals$, $f_i \in fields$
$n \geq 0$

Field-Sensitive

a.f

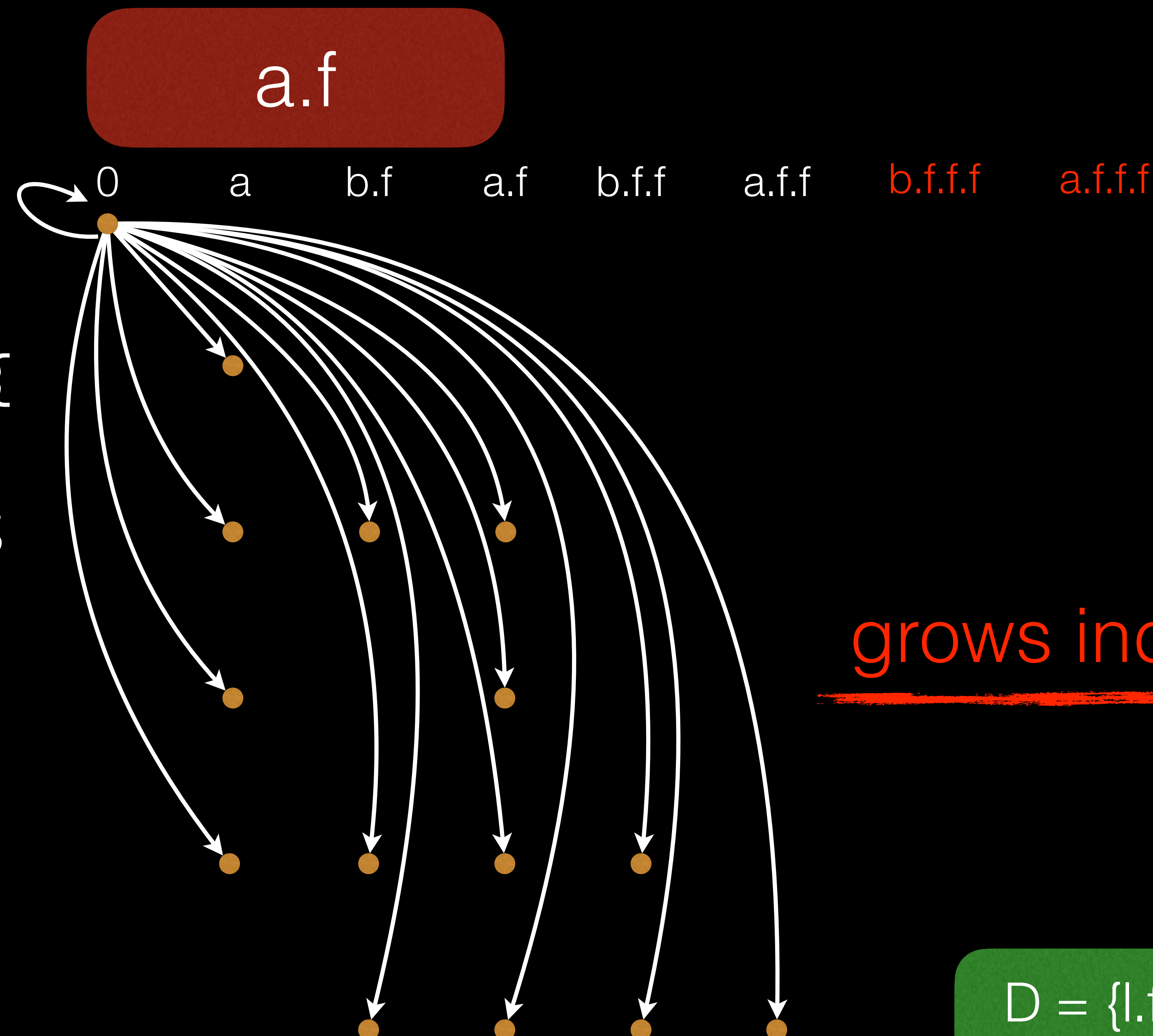
```
a = source();

while(rand()) {

  b = new DS();

  b.f = a;

  a = b;
}

sink(d);
```



$D = \{l.f_1.f_2.f_3. \ \ldots \ .f_n\}$
$l \in Locals,\ f_i \in fields$
$n \geq 0$

Field-Sensitive

```
a = source();

while(rand()) {

  b = new DS();

  b.f = a;

  a = b;
}

sink(d);
```



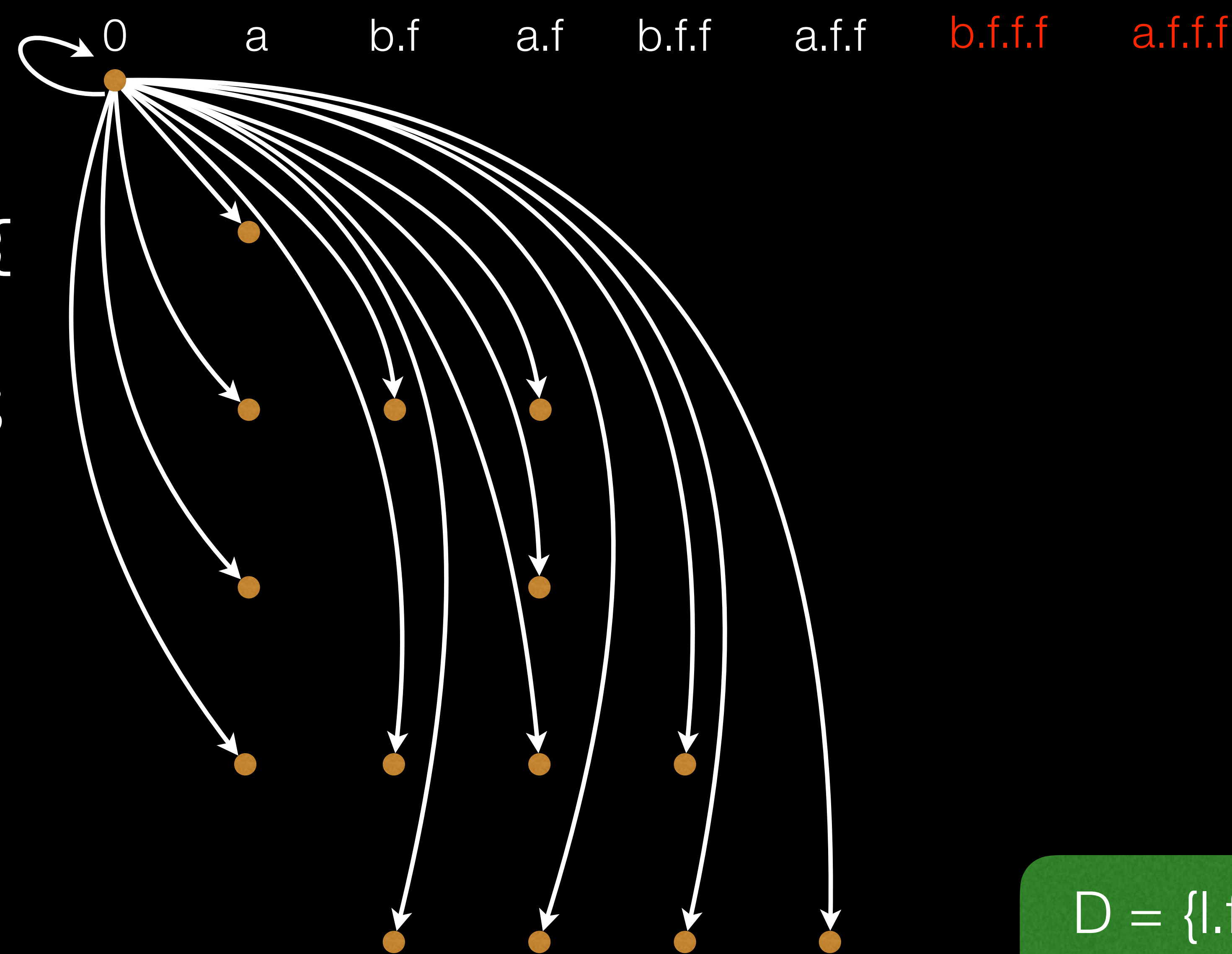
grows indefinitely

$D = \{l.f_1.f_2.f_3. \ldots .f_n\}$
$l \in \text{Locals}, f_i \in \text{fields}$
$n \geq 0$

k-Limiting

a.f

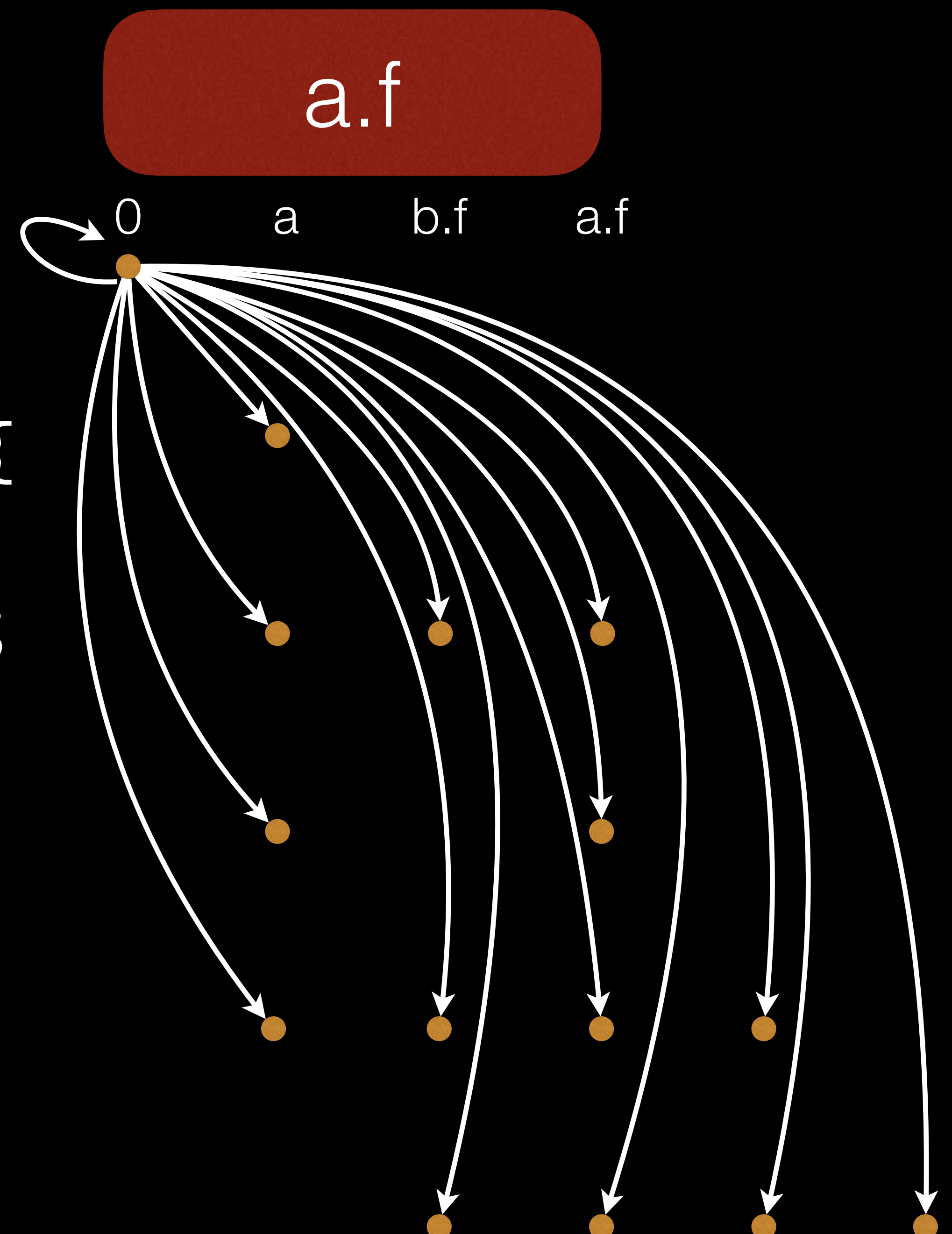
```
a = source();

while(rand()) {

  b = new DS();

  b.f = a;

  a = b;
}

sink(d);
```



$D = \{l.f_1.f_2.f_3. \ldots .f_n\}$
$l \in \text{Locals}, f_i \in \text{fields}$
$0 \leq n \leq k$

k-Limiting

```
a = source();

while(rand()) {

  b = new DS();

  b.f = a;

  a = b;
}

sink(d);
```



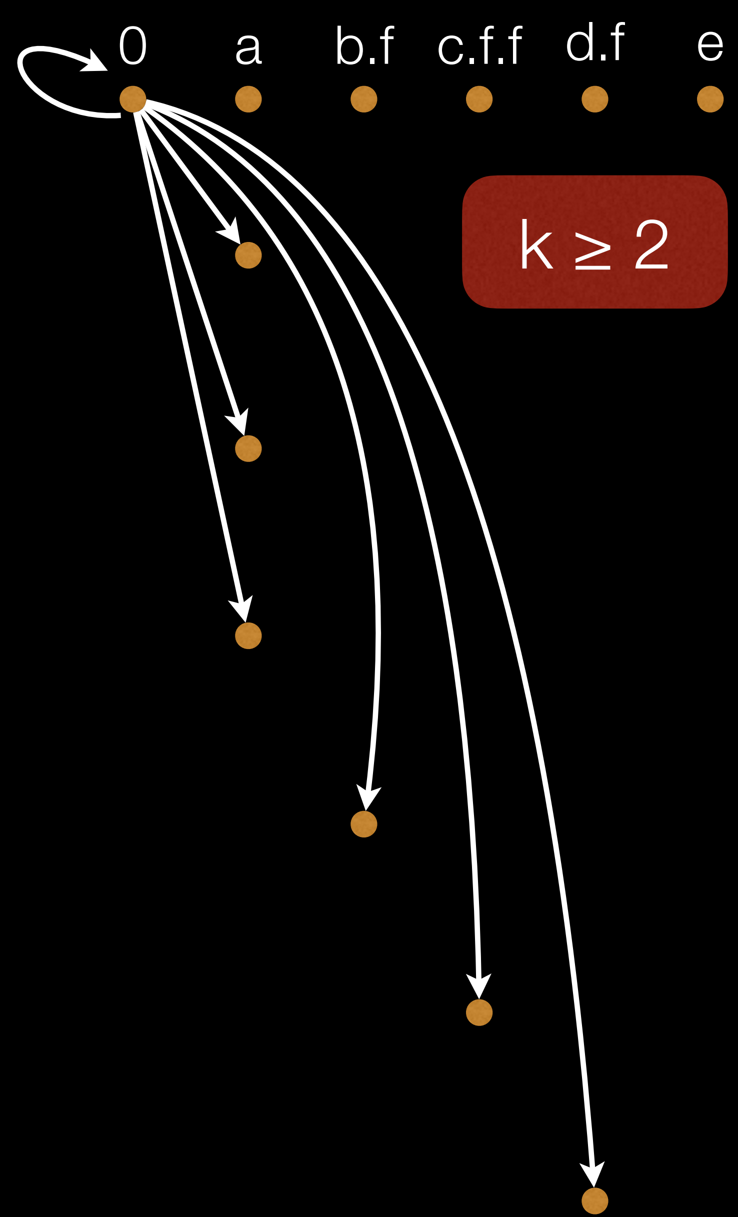
unsound domain
beyond k => imprecise

k = 2

$D = \{l.f_1.f_2.f_3. \dots .f_n\}$
$l \in$ Locals, $f_i \in$ fields
$0 \leq n \leq k$

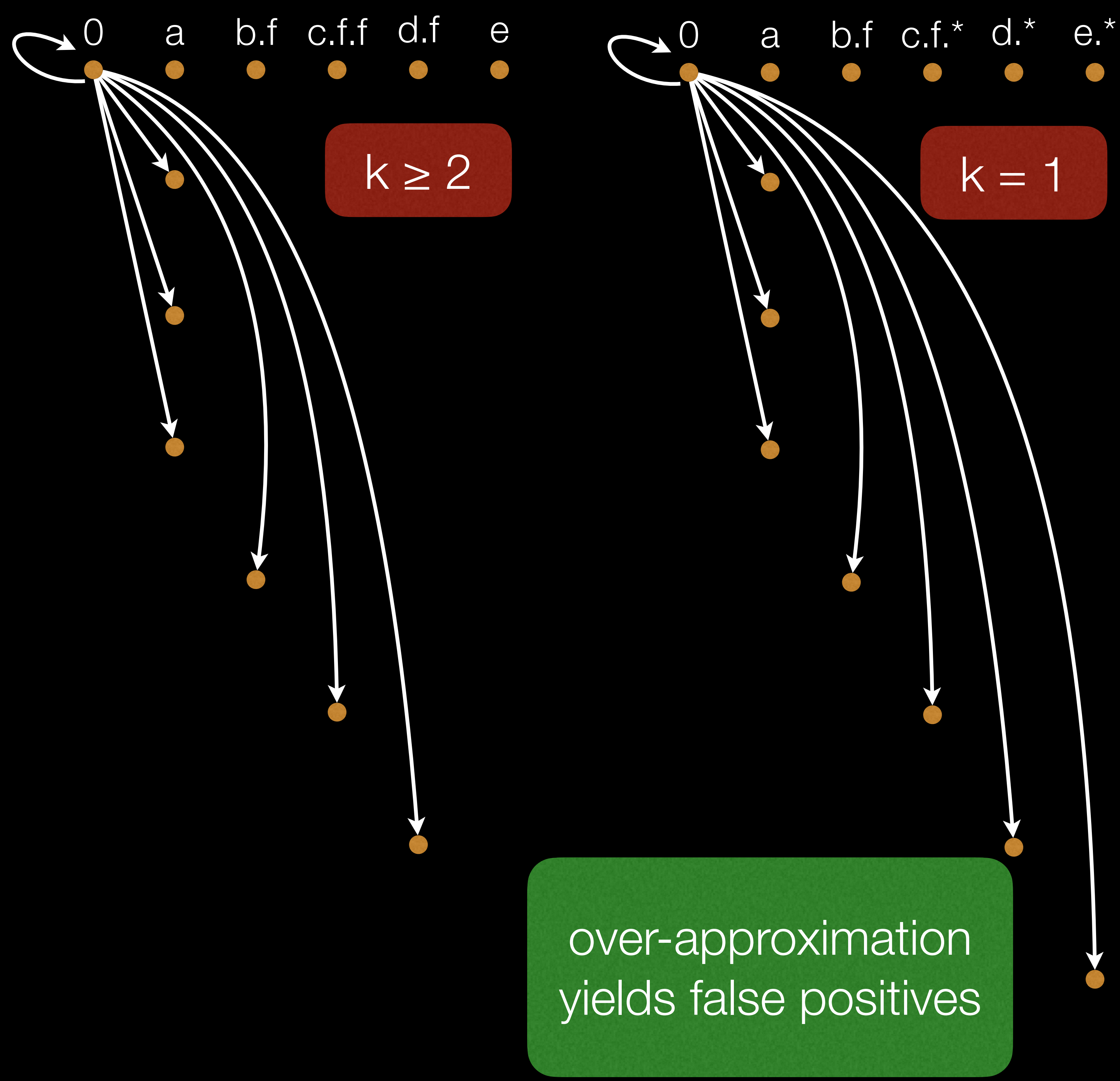
k-Limiting

```
a = source();

b = new DS();

c = new DS();

b.f = a;

c.f = b;

d = c.f;

e = d.g;

sink(e);
```



k-Limiting

```
a = source();

b = new DS();

c = new DS();

b.f = a;

c.f = b;

d = c.f;

e = d.g;

sink(e);
```



Tidbits

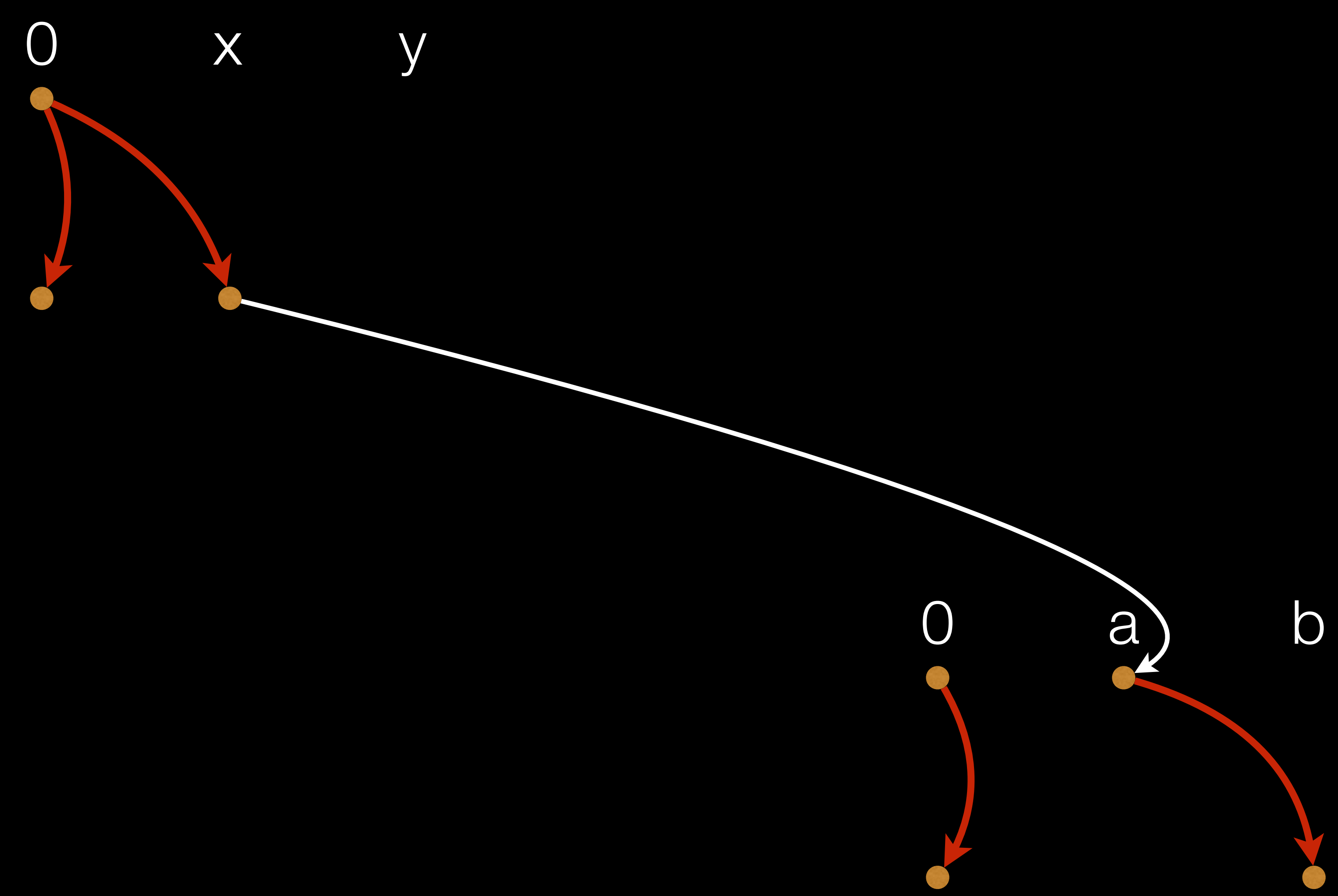
- Flow Sensitivity?
- Context Sensitivity?

Tidbits

- Flow Sensitivity?
- Context Sensitivity?

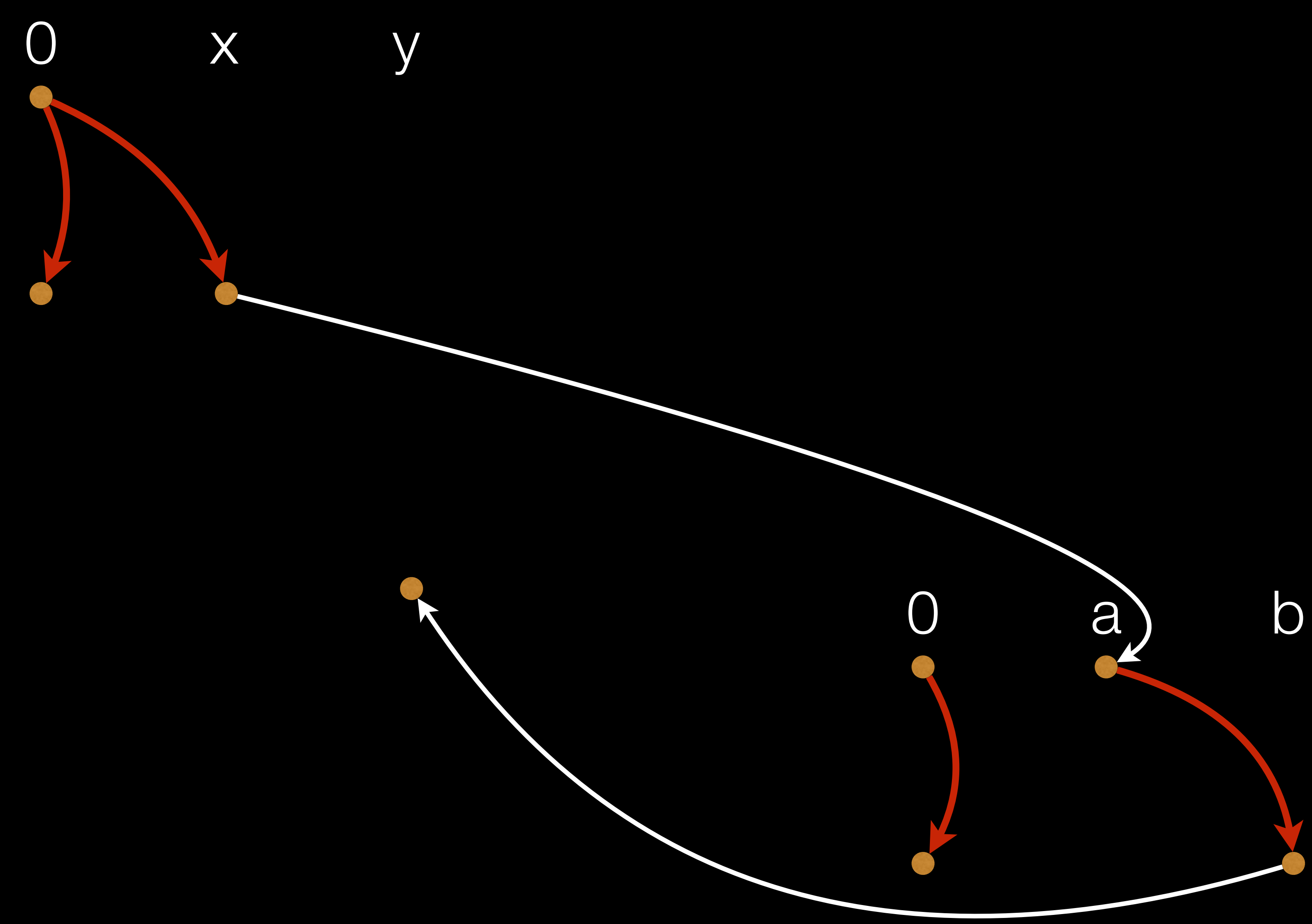
Method Summaries

Context Sensitivity




```
...

call1();

...

call2();
```

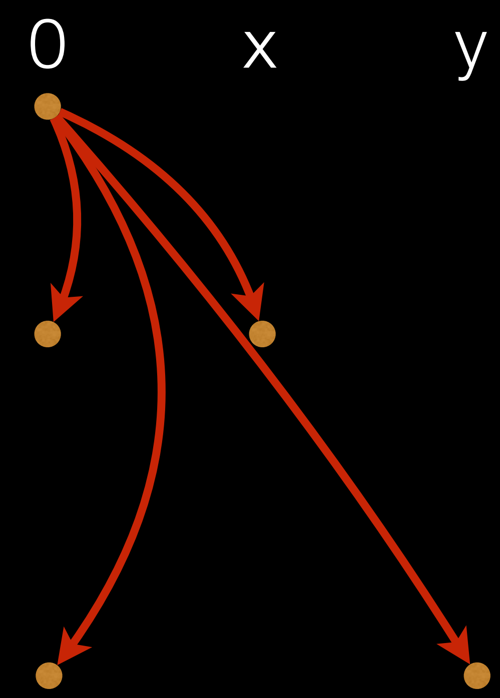
Context Sensitivity

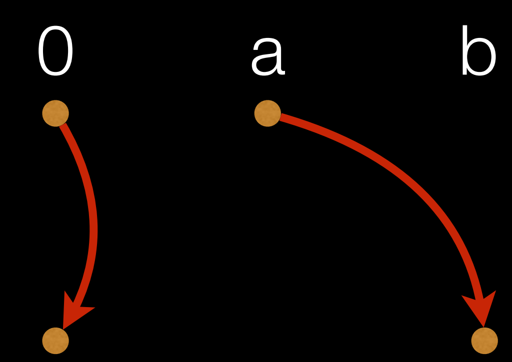



```
...

call1();

...

call2();
```

Context Sensitivity

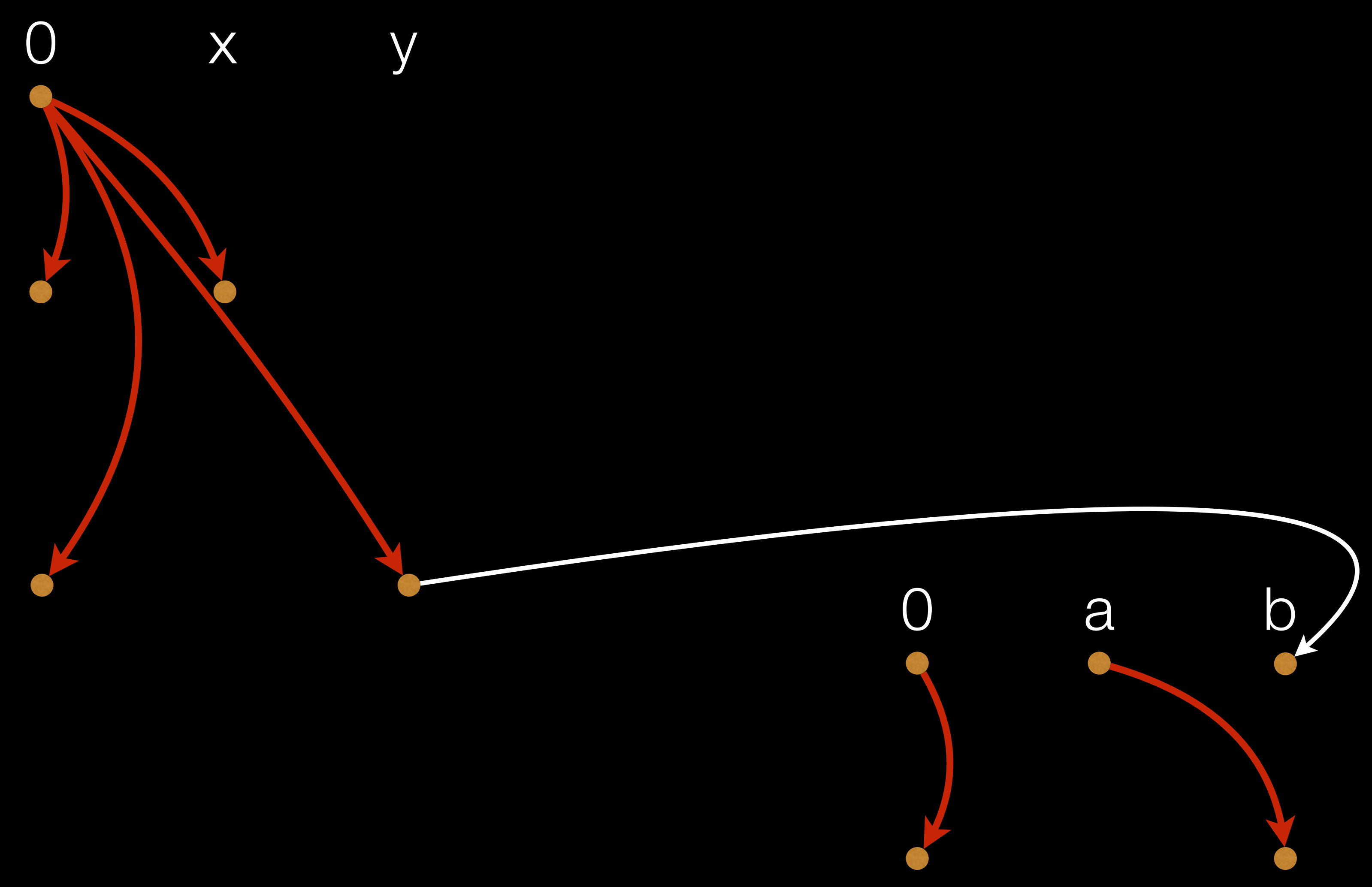
```
...

call1();

...

call2();
```



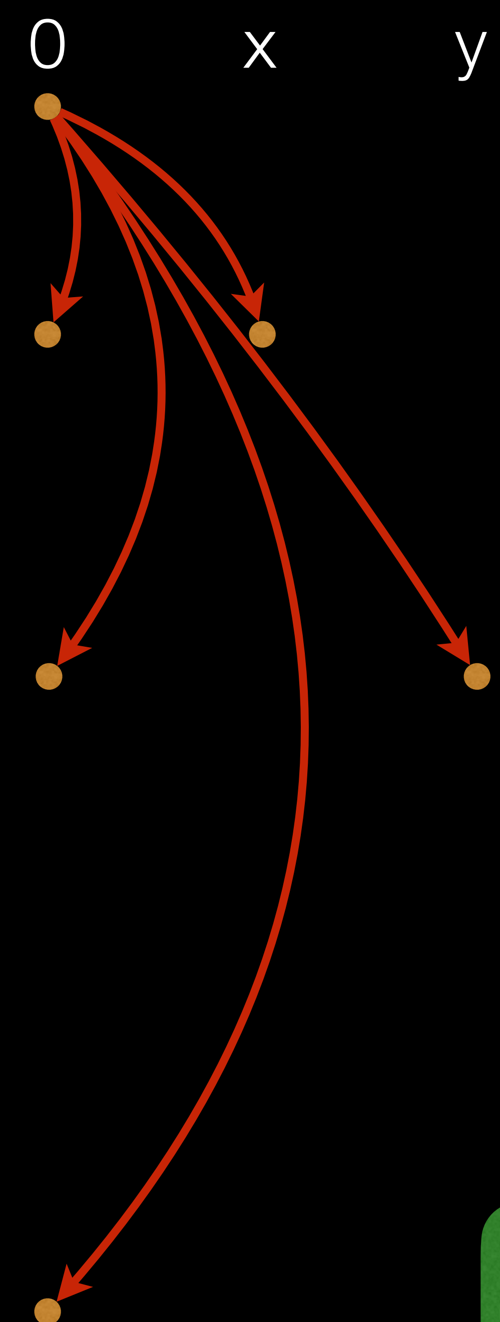


Context Sensitivity

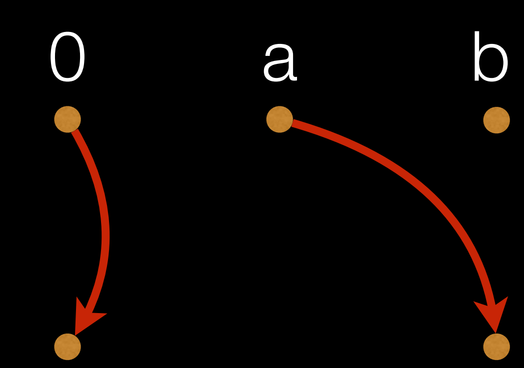
```
...

call1();

...

call2();
```



Context Sensitivity




```
...

call1();

...

call2();
```



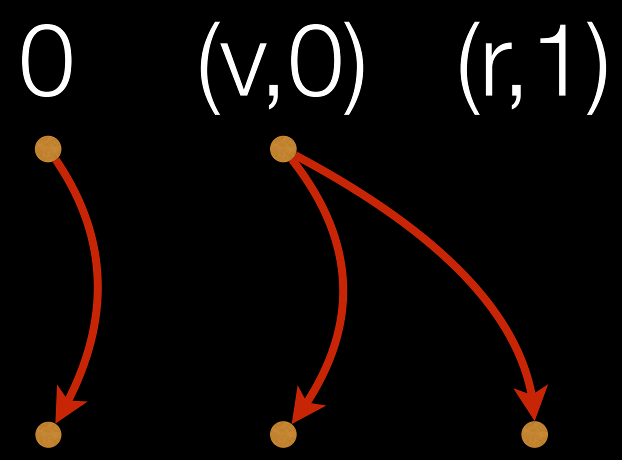
same summary
different contexts => different results

Tidbits

- Flow Sensitivity
- Context Sensitivity
- Large Domains => performance problems

Large Domains

`r = v + 1`

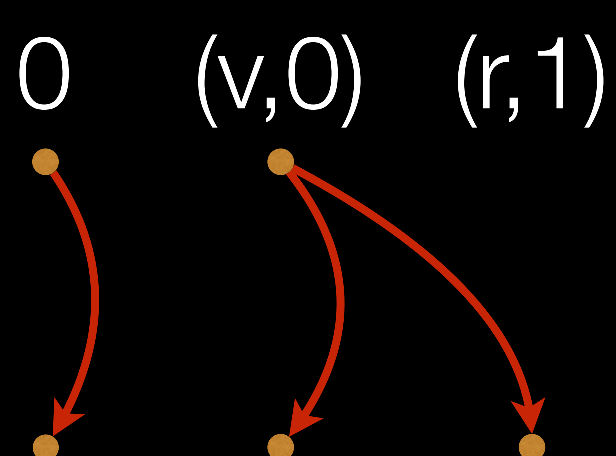


In any context, if *v* is 0 before the call, then it is true that *r* is 1 after the call

Large Domains

```
main() {
 x = inc(1);
 y = inc(x);
 z = inc(y);
 print(z);
}
```

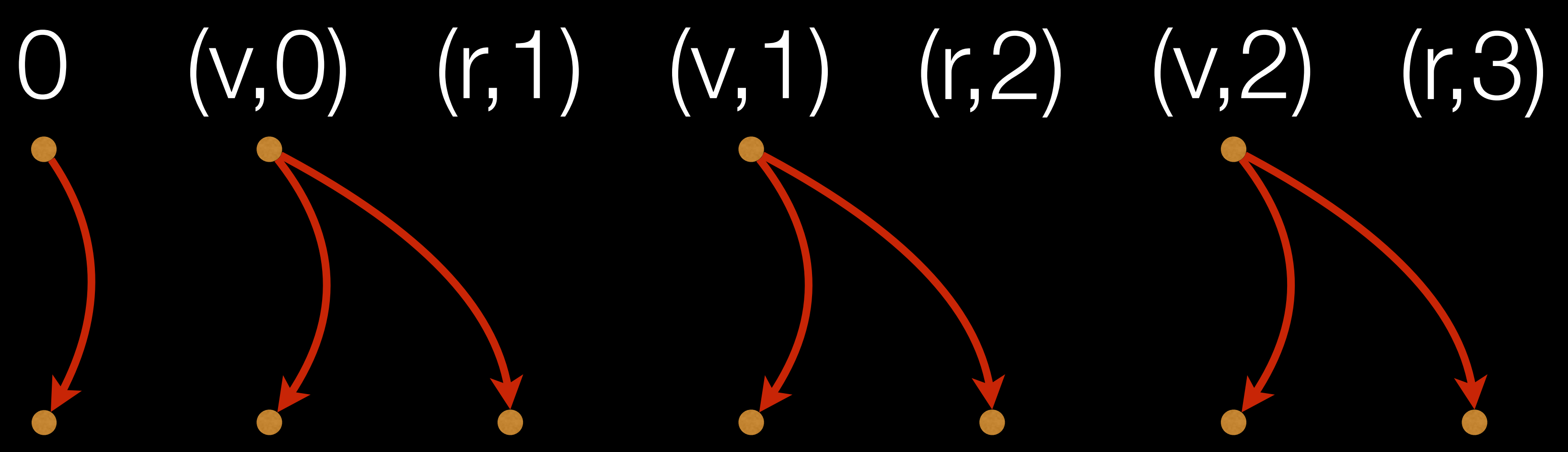
```
inc(v) {
 r = v + 1;
 return r;
}
```



Large Domains

```
main() {
 x = inc(1);
 y = inc(x);
 z = inc(y);
 print(z);
}
```

```
inc(v) {
 r = v + 1;
 return r;
}
```



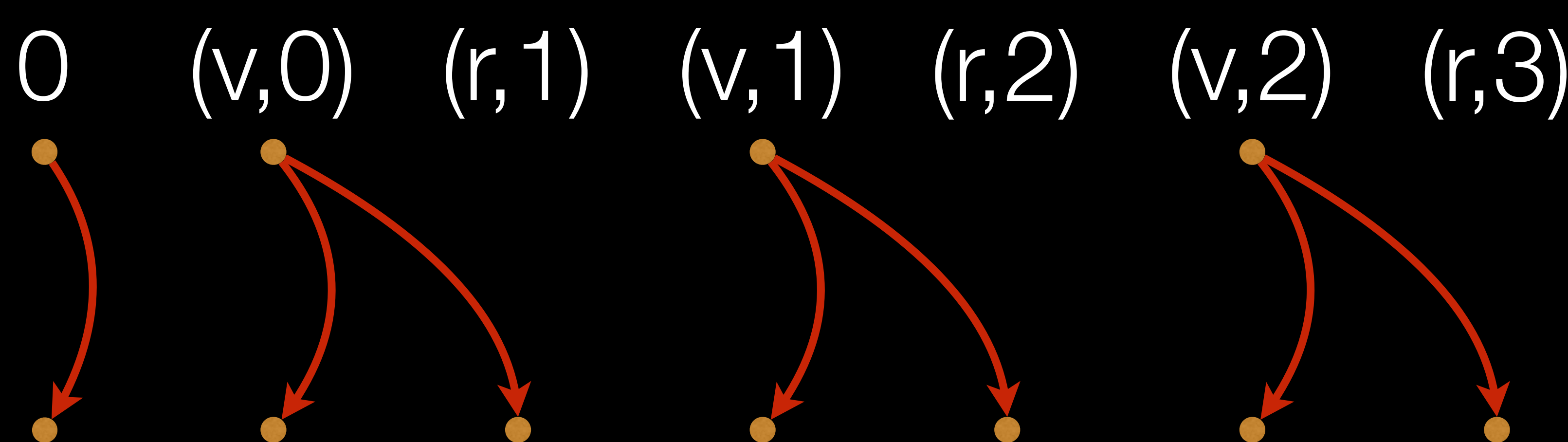
Large Domains

```
main() {
 x = inc(1);
 y = inc(x);
```

```
inc(v) {
 r = v + 1;
```

Solution: IDE instead of IFDS

```
}
```



Recap

- Interprocedural Finite Distributive Subset
- Flow Functions
- Taint Analysis Example
- On-the-fly ESG
- Tidbits: field-sensitivity, context-sensitivity, large domains

Next

- Distributive Environments (IDE)