h-CFA:

A Simplified Approach for Pushdown Control Analysis

Fei Peng

Under the Supervision of Professor Tian Zhao

University of Wisconsin-Milwaukee

July 2016

What is Static Program Analysis?

Answer Runtime Questions without Running the Program.

Answer *Runtime Questions* without Running the Program.

- What is the possible output?
- Will the value of `x` be used in the future?
- Does the variable `x` always have the same value?
- Can the pointer `p` be null ?
- In call site `f(1)`, which functions will be called?

Control Flow Analysis

Answer Runtime Questions ~~*without Running*~~ the Program.

Run it in abstract!

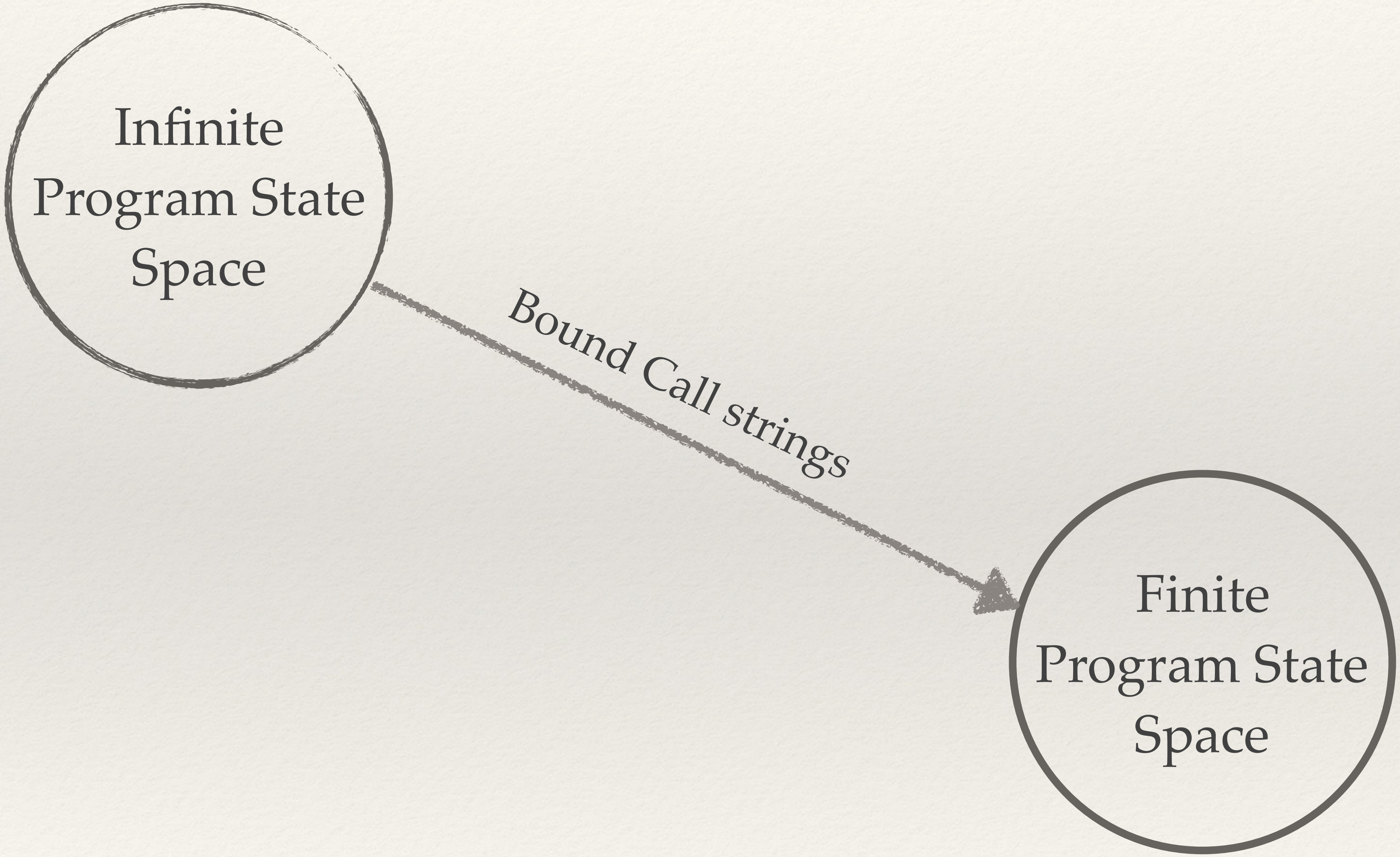
- These questions are difficult in compile-time.

Answer Runtime Questions ~~*without Running*~~ the Program.

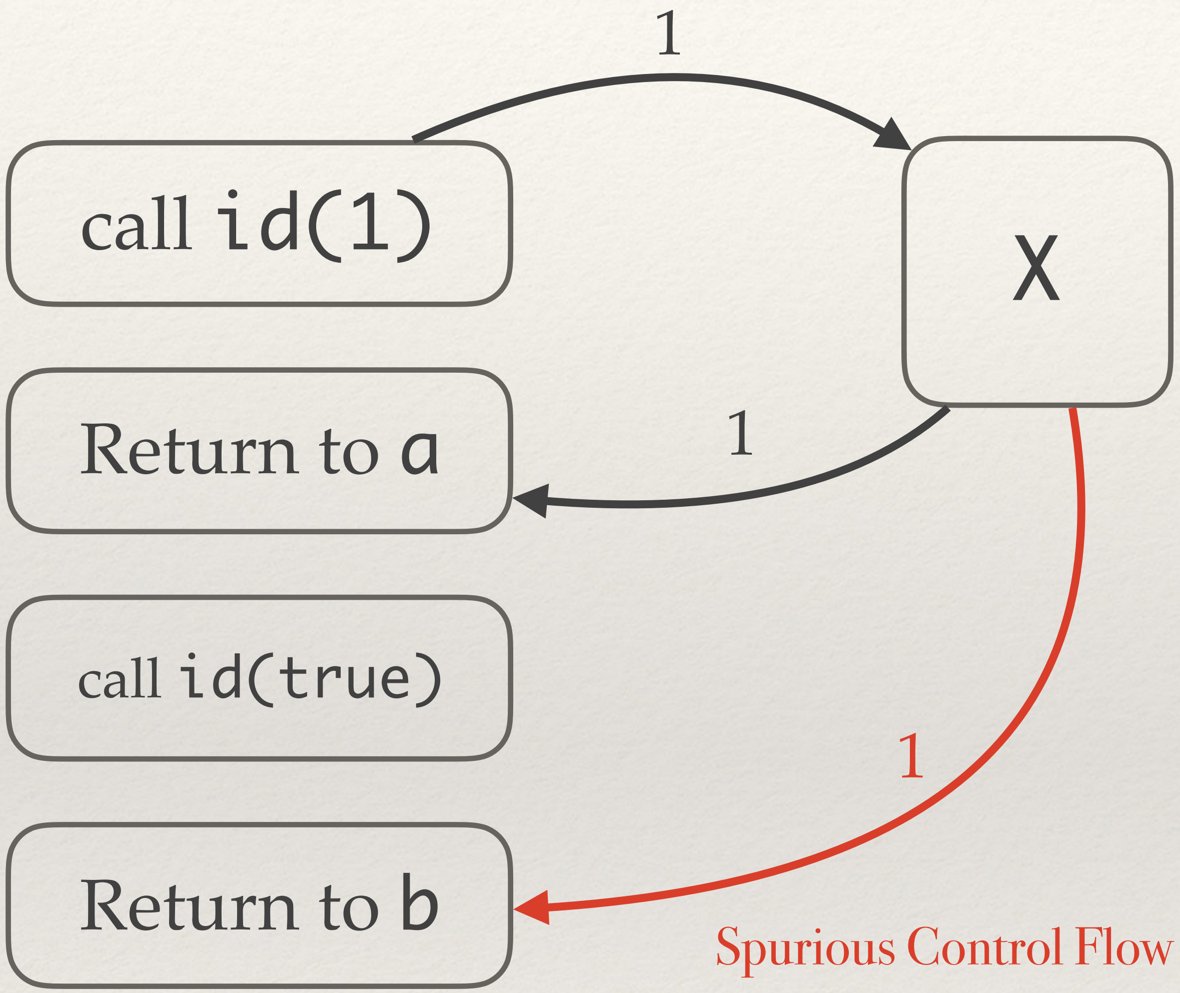
Abstract Interpretation

- These questions are difficult in compile-time.

k-CFA


Infinite Program State Space
Bound Call strings
Finite Program State Space

0-CFA



call id(1)

Return to a

call id(true)

Return to b

x

1

1

1

Spurious Control Flow

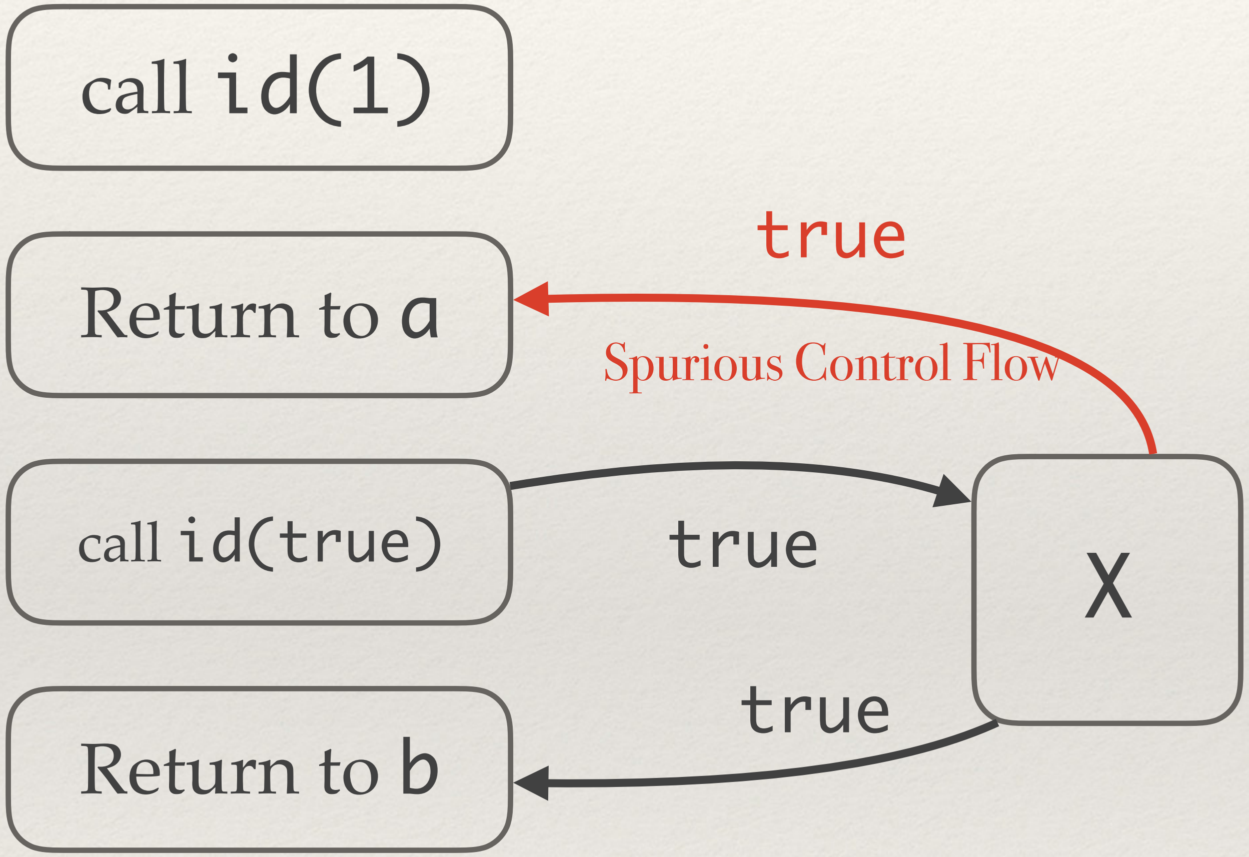
x → {1}

a → {1}

b → {1}

```
function id(x) {
  return x;
}

var a = id(1);
var b = id(true);
```

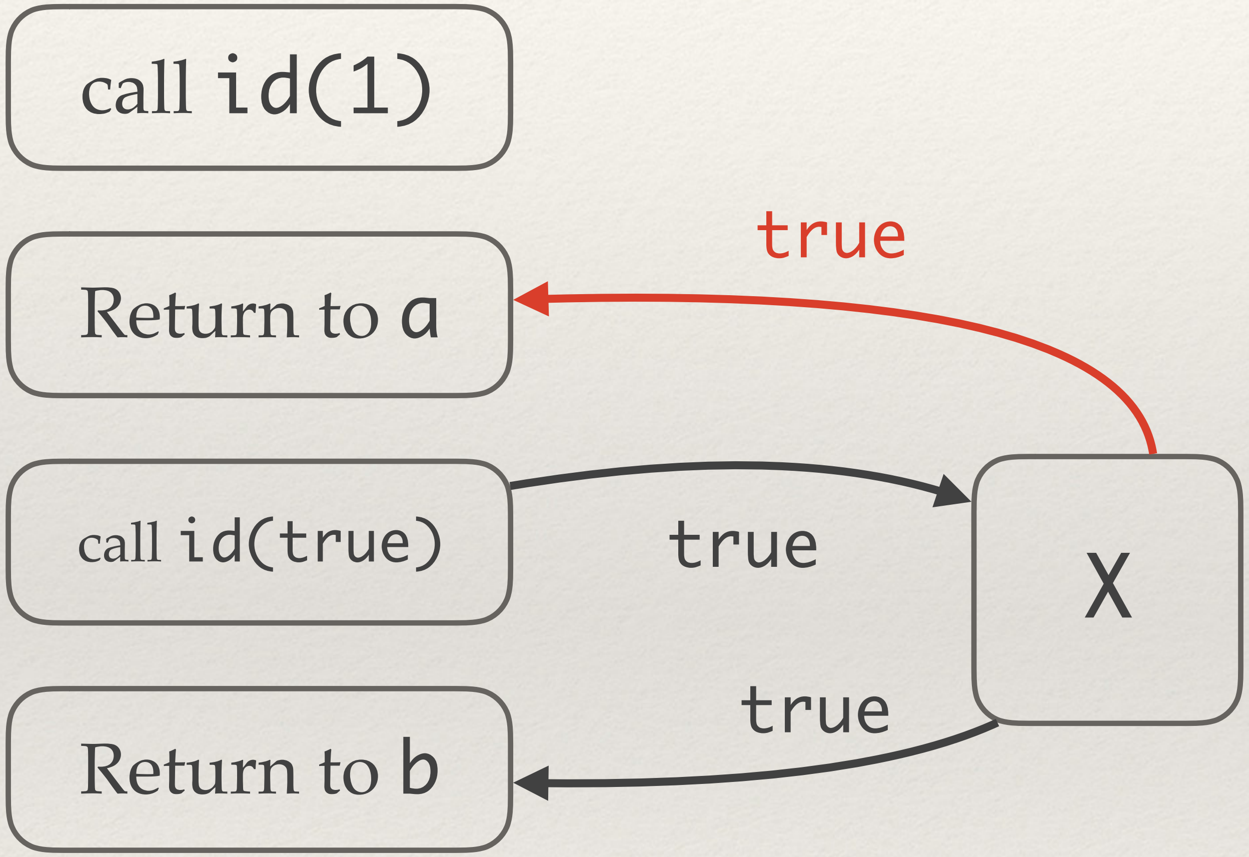
0-CFA



x → {1}
a → {1}
b → {1}

```
function id(x) {
  return x;
}

var a = id(1);
var b = id(true);
```

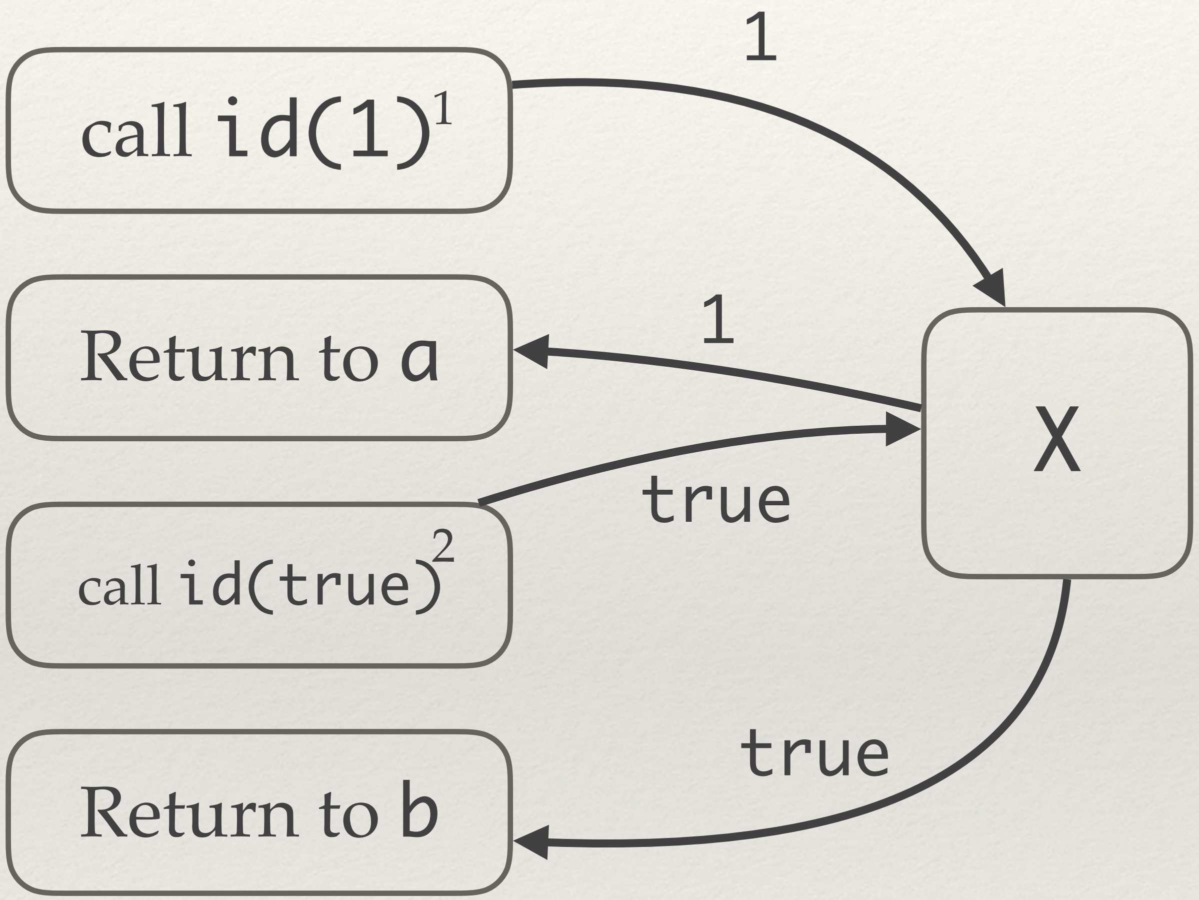
0-CFA



x → {1, true}
a → {1, true}
b → {1, ture}

```
function id(x) {
  return x;
}

var a = id(1);
var b = id(true);
```

1-CFA



(x,[1]) → {1}
(x,[2]) → {true}
(a,[]) → {1}
(b,[]) → {ture}

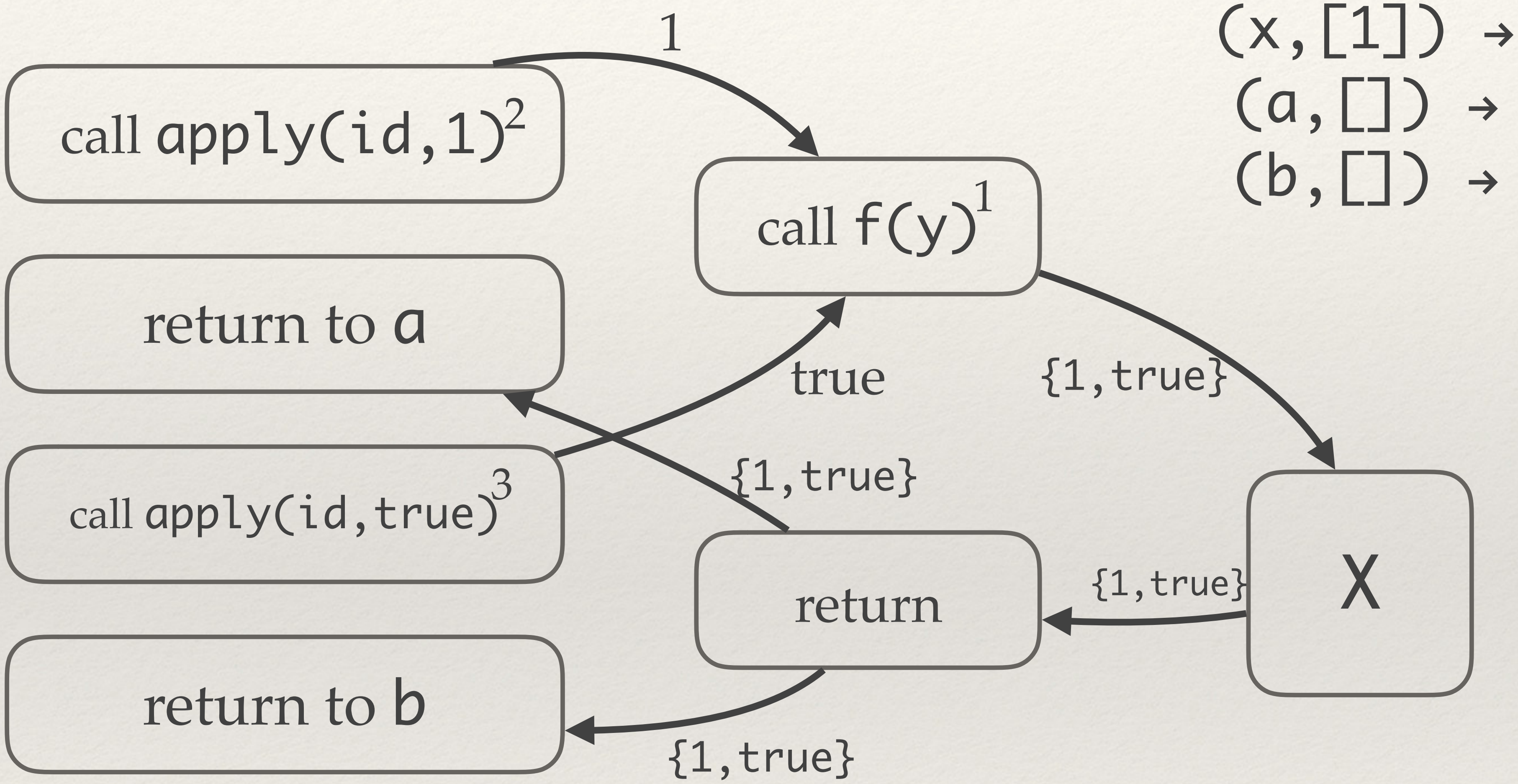
```
function id(x) {
  return x;
}

var a = id(1);
var b = id(true);
```

1-CFA

```
function id(x) {
  return x;
}

function apply(f, y) {
  return f(y);
}

var a = apply(id, 1);
var b = apply(id, true);
```

1-CFA

```
(x,[1]) → {1,true}
 (a,[]) → {1,true}
 (b,[]) → {1,true}
```



1

call apply(id,1)^2

call f(y)^1

return to a

true

{1,true}

call apply(id,true)^3

{1,true}

X

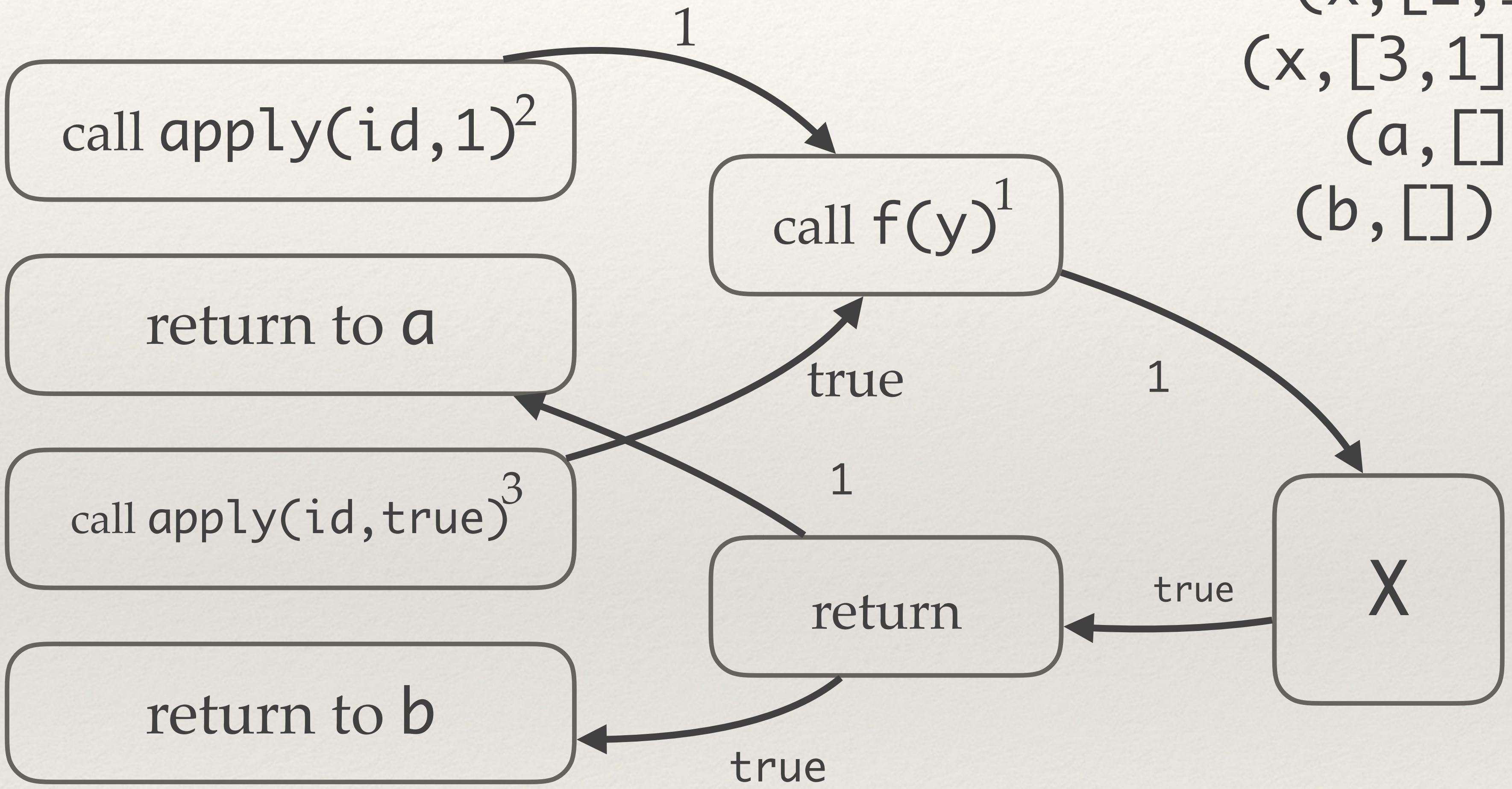
return

{1,true}

return to b

{1,true}

2-CFA



```
(x,[2,1]) → {1}
(x,[3,1]) → {true}
(a,[]) → {1}
(b,[]) → {true}
```

k-CFA

0-CFA : Cubic Time

k-CFA ($k > 0$) : Exponential Time

Only Work with Call Strings

Implementation is Difficult

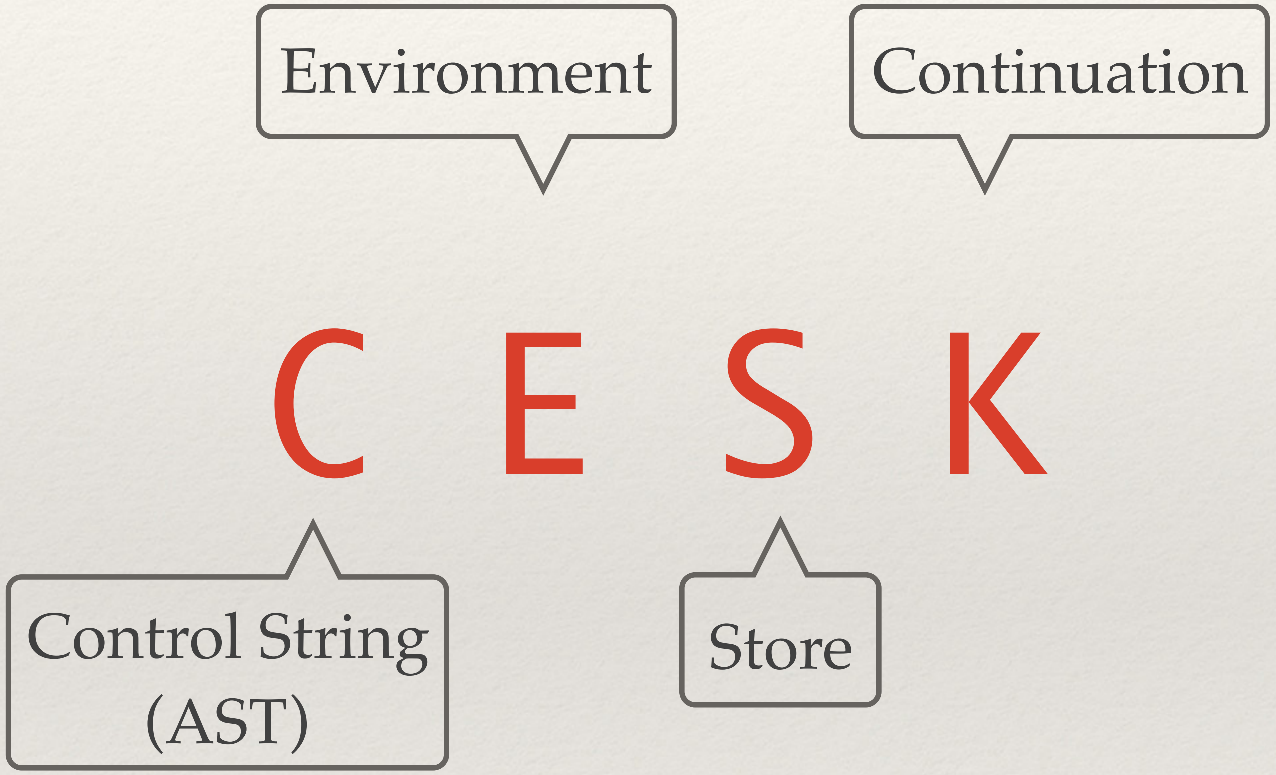
Work on CPS programs

Pushdown Control Flow Analysis

Perfect Call/Return Matching for All

Recursive and Non-Recursive Programs

Abstract Machine



Environment

Continuation

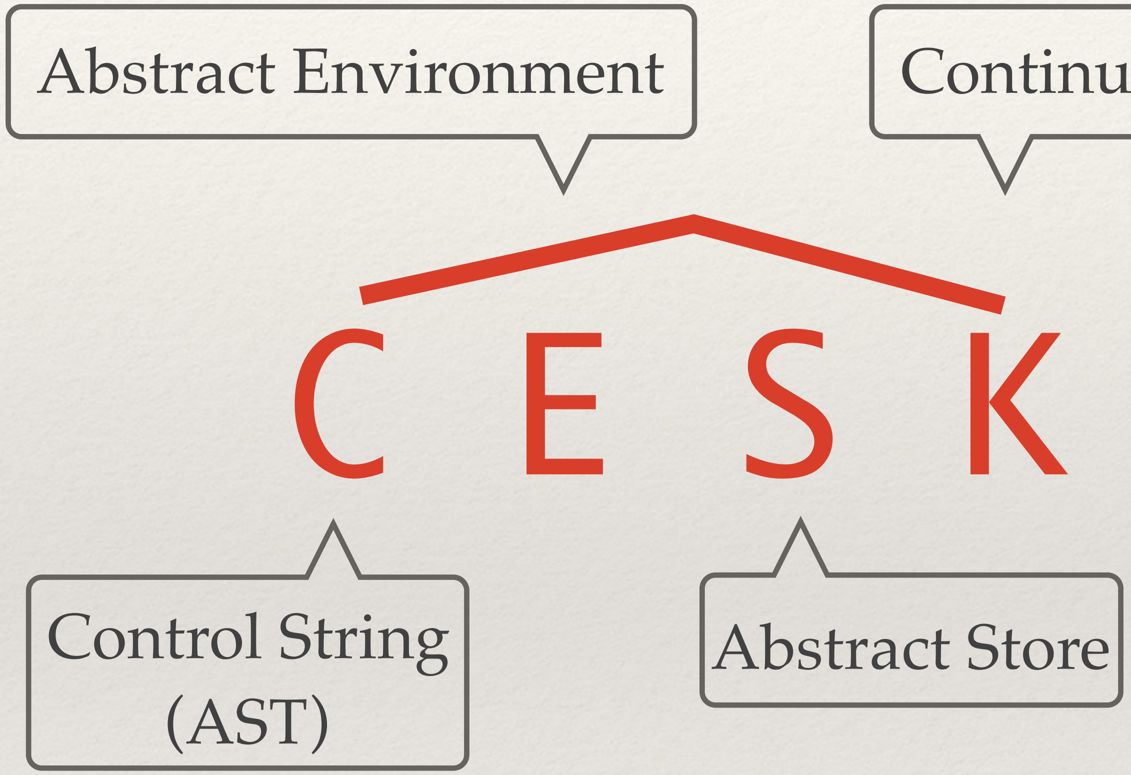
C E S K

Control String
(AST)

Store

Abstracting Abstract Machine

– Van Horn, Might (2010)



Abstract Environment

Continuation Address

C E S K

Control String (AST)

Abstract Store

How to do Abstracting?

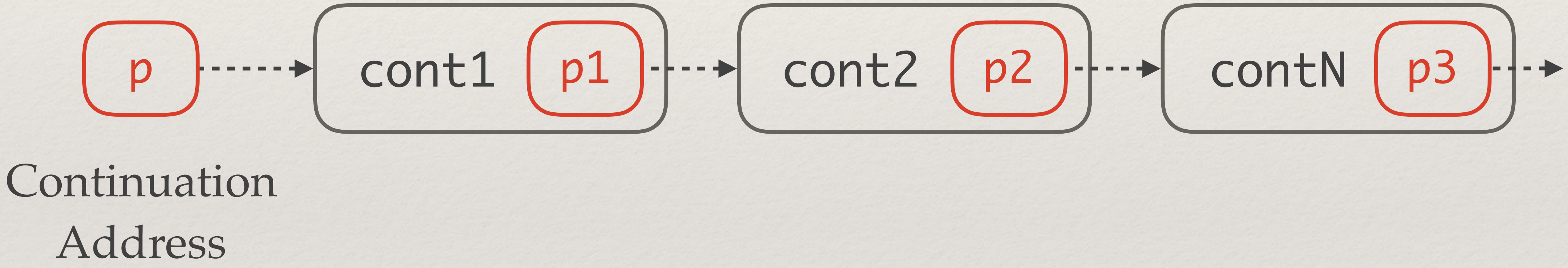
Continuation Stack

```
(cont1, (cont2, (cont3, …(contN, Nil))))
```

Recursive Structures Make Infinite States!

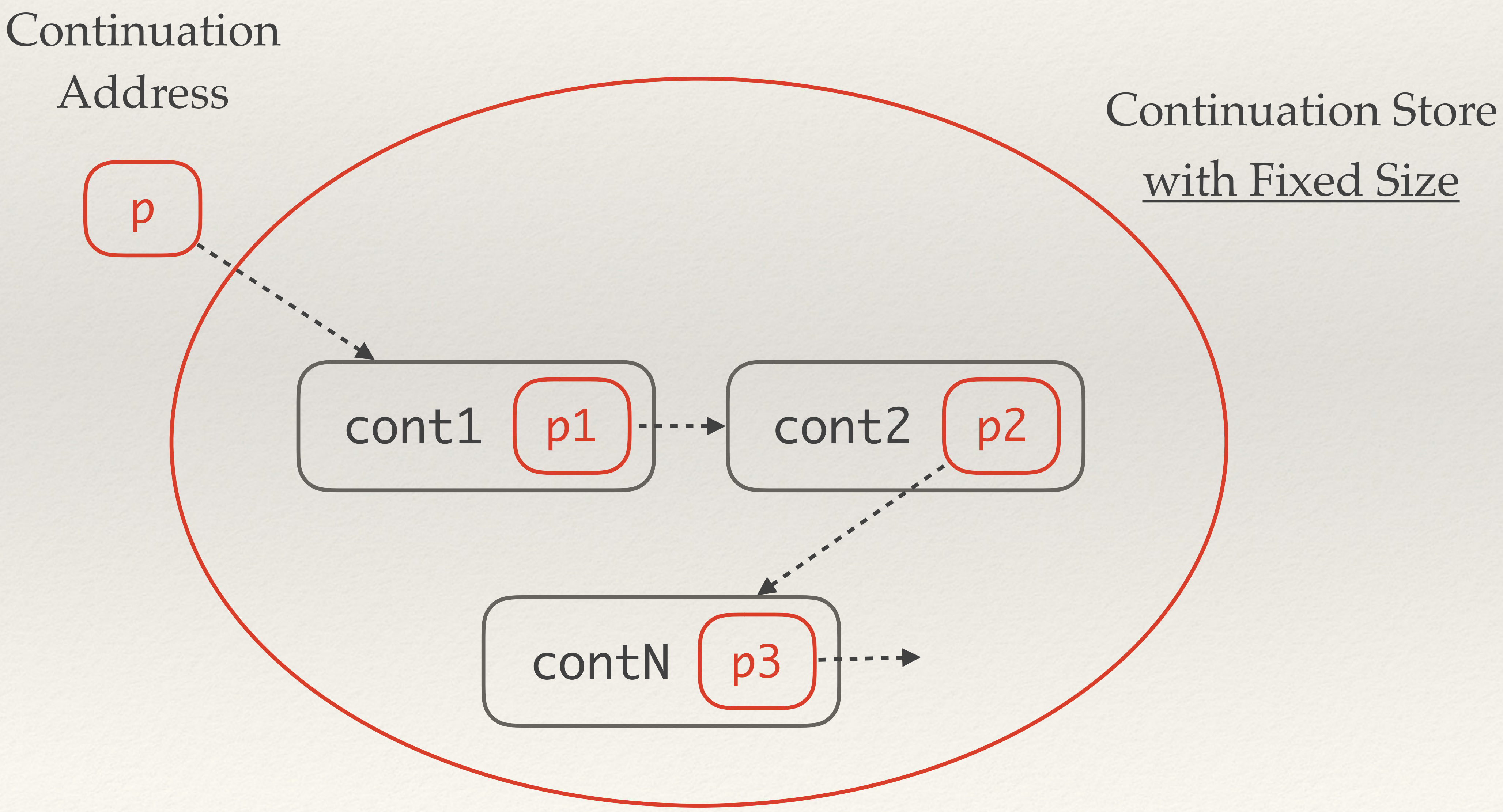
How to do Abstracting?

Continuation Stack (Linked-Structure)



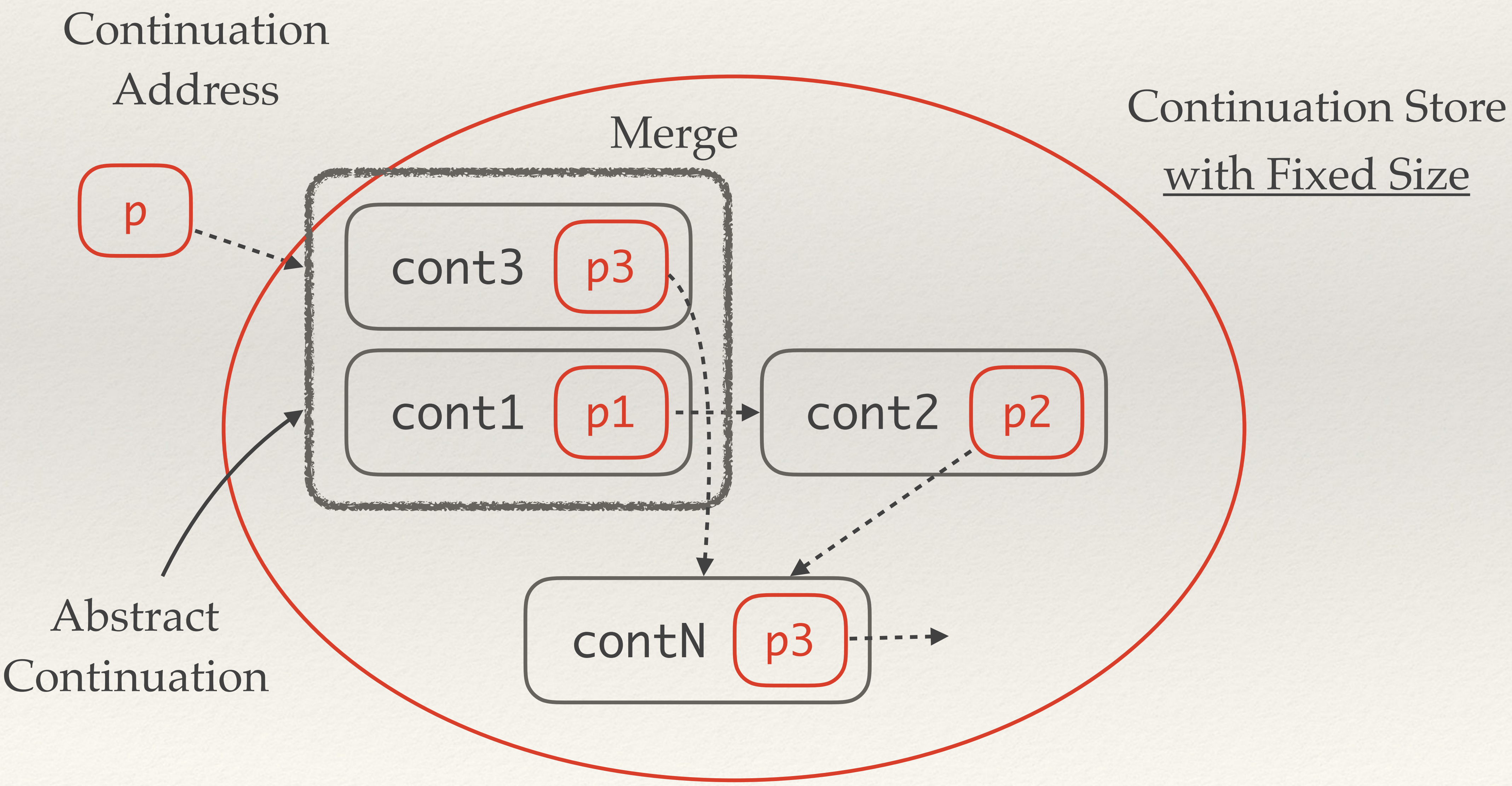
How to do Abstracting?

Continuation Stack (Linked-Structure)



How to do Abstracting?

Continuation Stack (Linked-Structure)



How to do Abstracting?

Fixed Continuation Store

+ = Finite State Space

Fixed Value Store

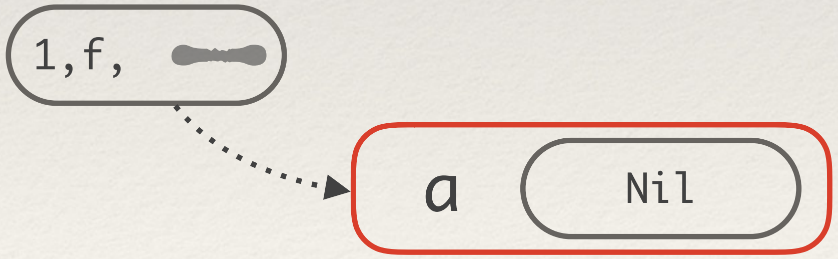
Representation of Continuation Address

P4F	`(callee, target environment)`
AAC	`(source closure, target closure, store)`
h-CFA	`(call site, callee, history)`

h-CFA

```
let fact = function fact(x) {
   let res1 = x < 1;
   if(res1) {
      return 1;
   } else {
      let res2 = x - 1;
      let res3 = fact(res2)[4];
      let res4 = x * res3;
      return res4;
   }
}

var f = function (n) {
   let res0 = fact(n)[3];
   return res0;
}

let a = f(10)[1];
let b = f(20)[2];
```

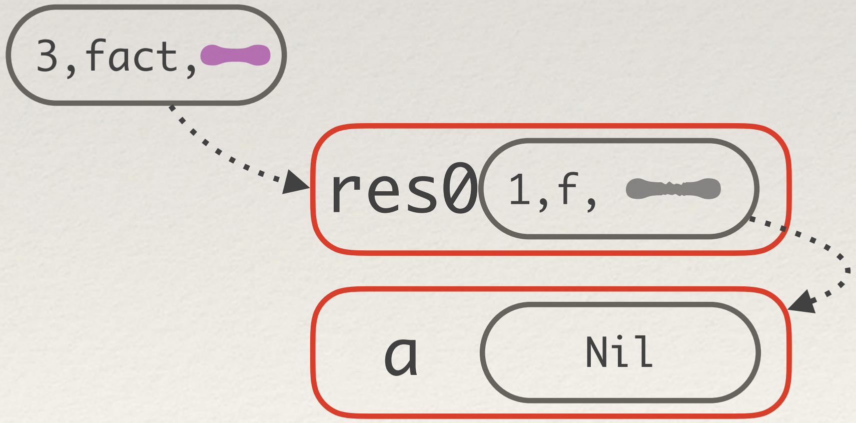
Current History: {fact,f}



h-CFA

```
let fact = function fact(x) {
  let res1 = x < 1;
  if(res1) {
    return 1;
  } else {
    let res2 = x - 1;
    let res3 = fact(res2)4;
    let res4 = x * res3;
    return res4;
  }
}

var f = function (n) {
  let res0 = fact(n)3;
  return res0;
}

let a = f(10)1;
let b = f(20)2;
```

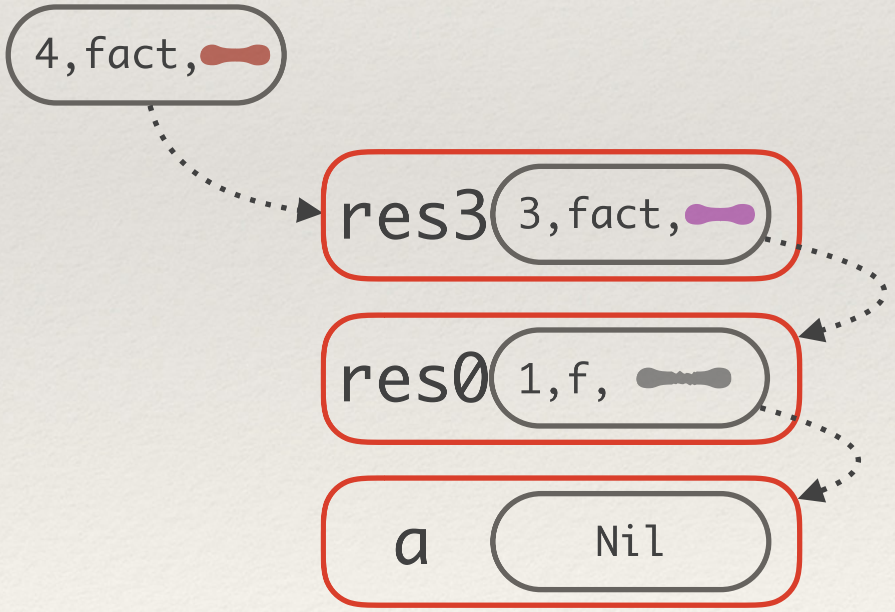
Current History: `{fact,f,n}`



h-CFA

```
let fact = function fact(x) {
  let res1 = x < 1;
  if(res1) {
    return 1;
  } else {
    let res2 = x - 1;
    let res3 = fact(res2)[4];
    let res4 = x * res3;
    return res4;
  }
}

var f = function (n) {
  let res0 = fact(n)[3];
  return res0;
}

let a = f(10)[1];
let b = f(20)[2];
```

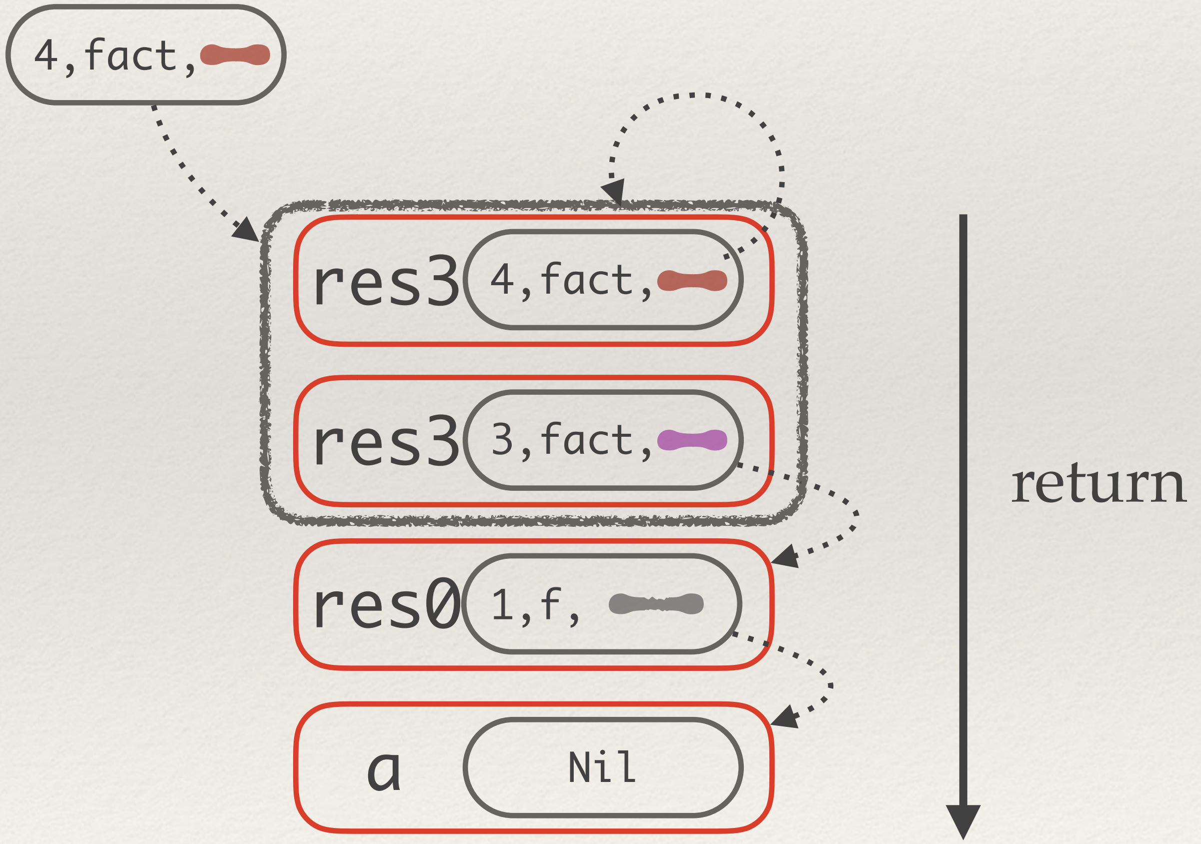
Current History:
{fact,f,n,x,res1,res2}



h-CFA

```
let fact = function fact(x) {
  let res1 = x < 1;
  if(res1) {
    return 1;
  } else {
    let res2 = x - 1;
    let res3 = fact(res2)4;
    let res4 = x * res3;
    return res4;
  }
}


var f = function (n) {
  let res0 = fact(n)3;
  return res0;
}


let a = f(10)1;
let b = f(20)2;
```

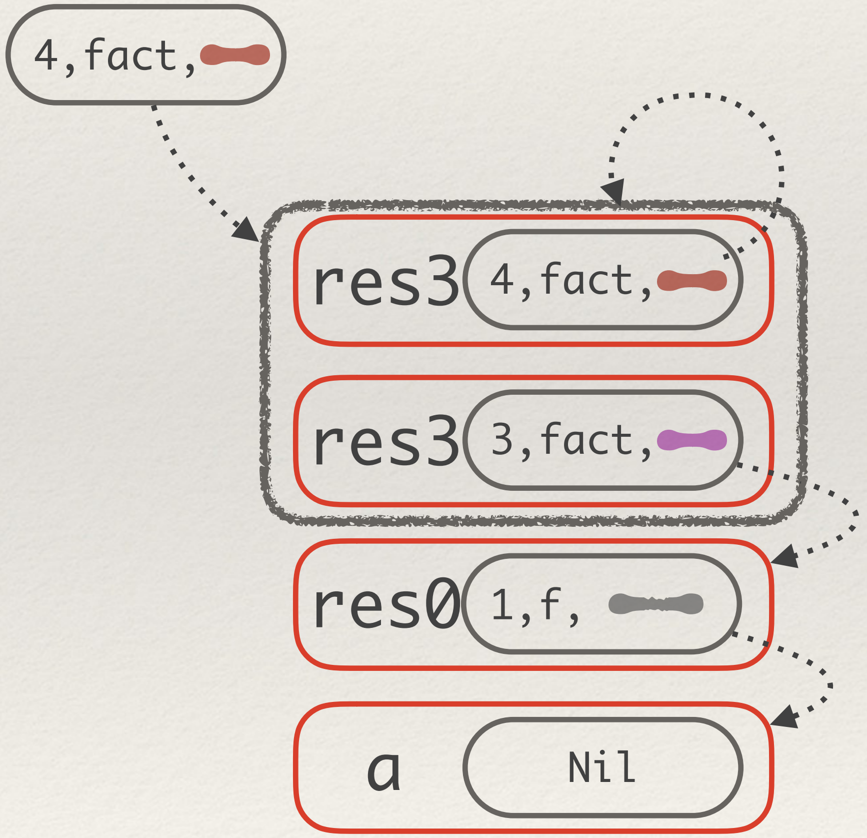
Current History:
{fact,f,n,x,res1,res2}



h-CFA

```
let fact = function fact(x) {
  let res1 = x < 1;
  if(res1) {
    return 1;
  } else {
    let res2 = x - 1;
    let res3 = fact(res2)[4];
    let res4 = x * res3;
    return res4;
  }
}

var f = function (n) {
  let res0 = fact(n)[3];
  return res0;
}

let a = f(10)[1];
let b = f(20)[2];
```

Current History:
{fact,f,n,x,res1,res2}



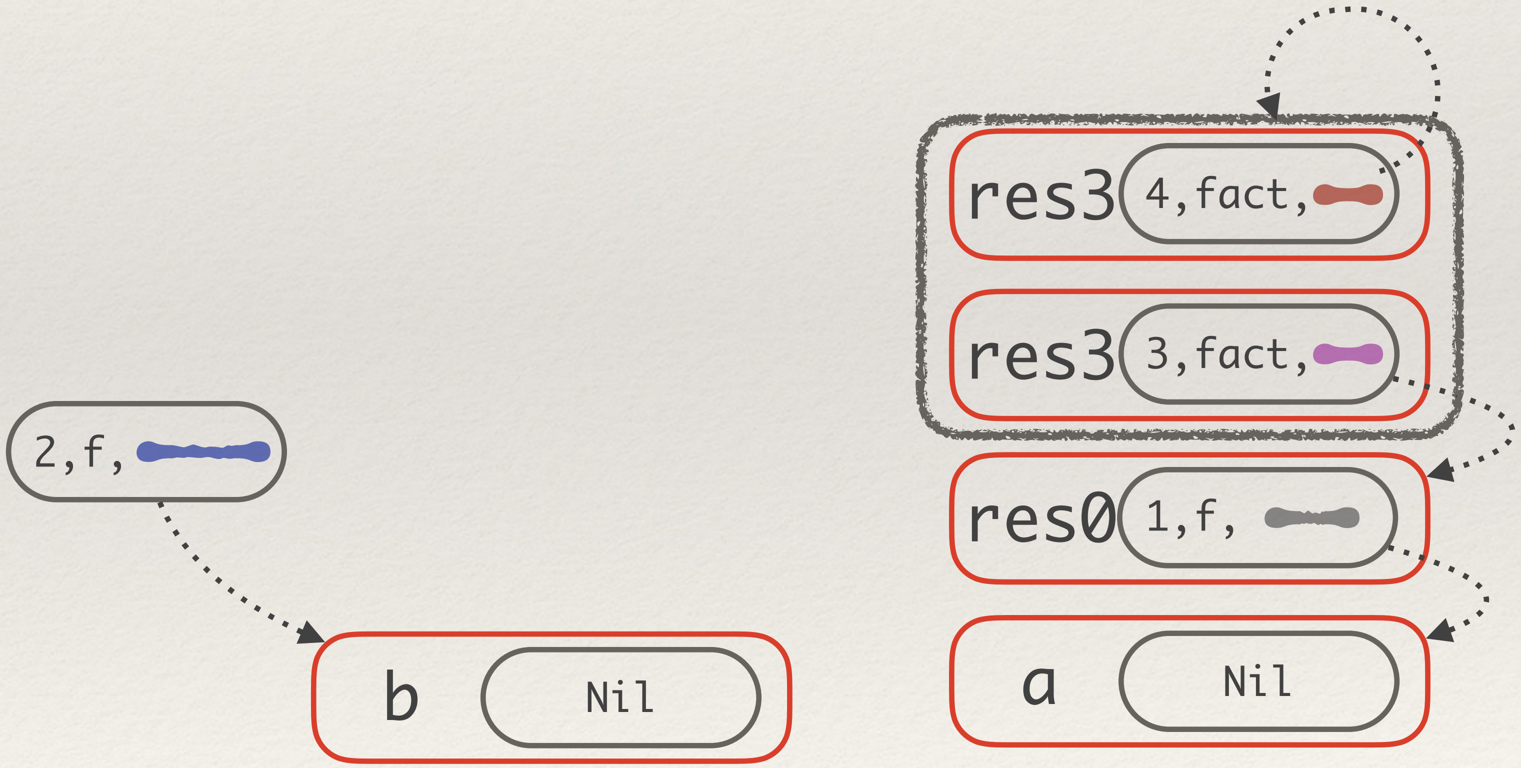
2-CFA
Call-String

4
4
3
1

h-CFA

```
let fact = function fact(x) {
   let res1 = x < 1;
   if(res1) {
      return 1;
   } else {
      let res2 = x - 1;
      let res3 = fact(res2)[4];
      let res4 = x * res3;
      return res4;
   }
}

var f = function (n) {
   let res0 = fact(n)[3];
   return res0;
}

let a = f(10)[1];
let b = f(20)[2];
```

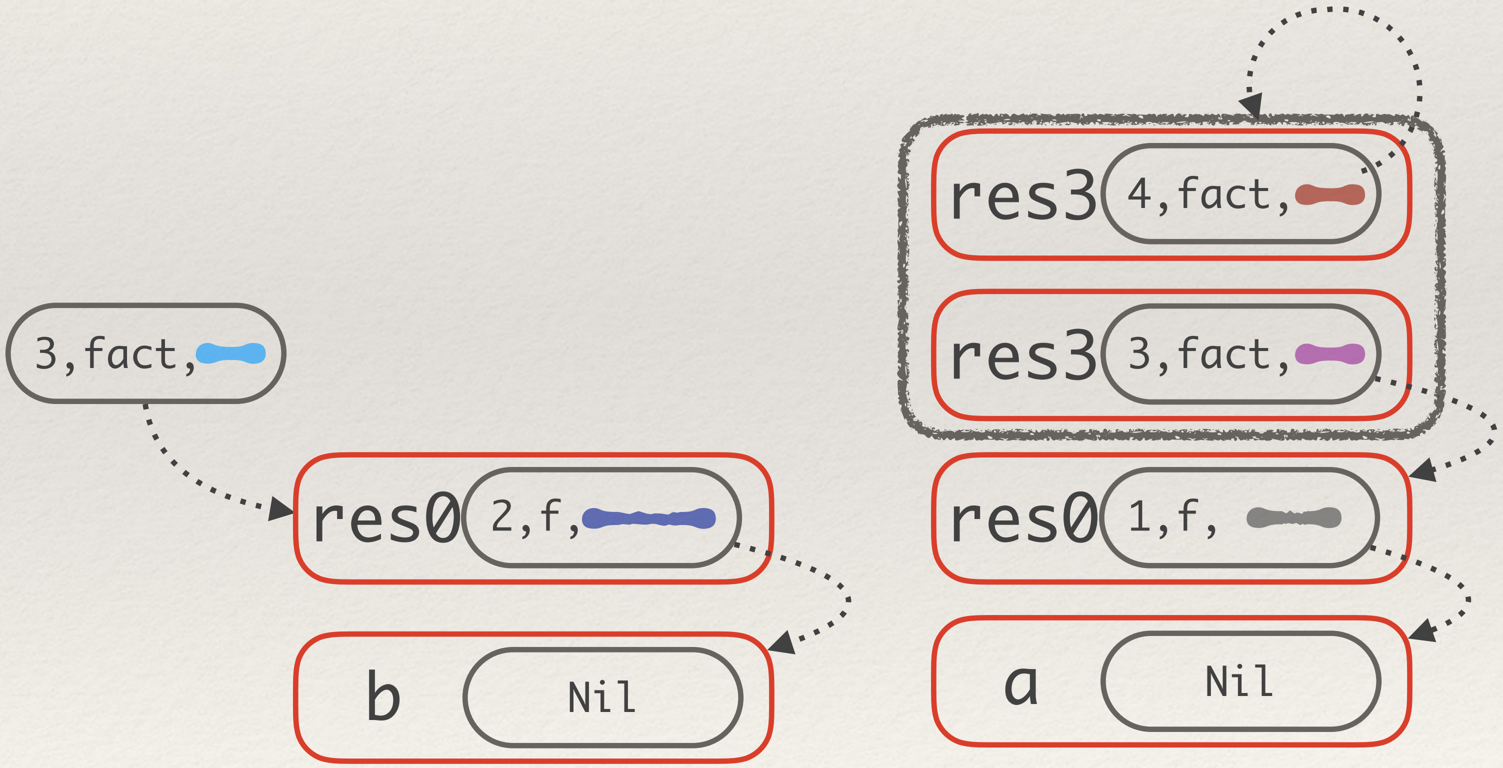
Current History: {fact,f,a}



h-CFA

```
let fact = function fact(x) {
  let res1 = x < 1;
  if(res1) {
    return 1;
  } else {
    let res2 = x - 1;
    let res3 = fact(res2)[4];
    let res4 = x * res3;
    return res4;
  }
}

var f = function (n) {
  let res0 = fact(n)[3];
  return res0;
}

let a = f(10)[1];
let b = f(20)[2];
```

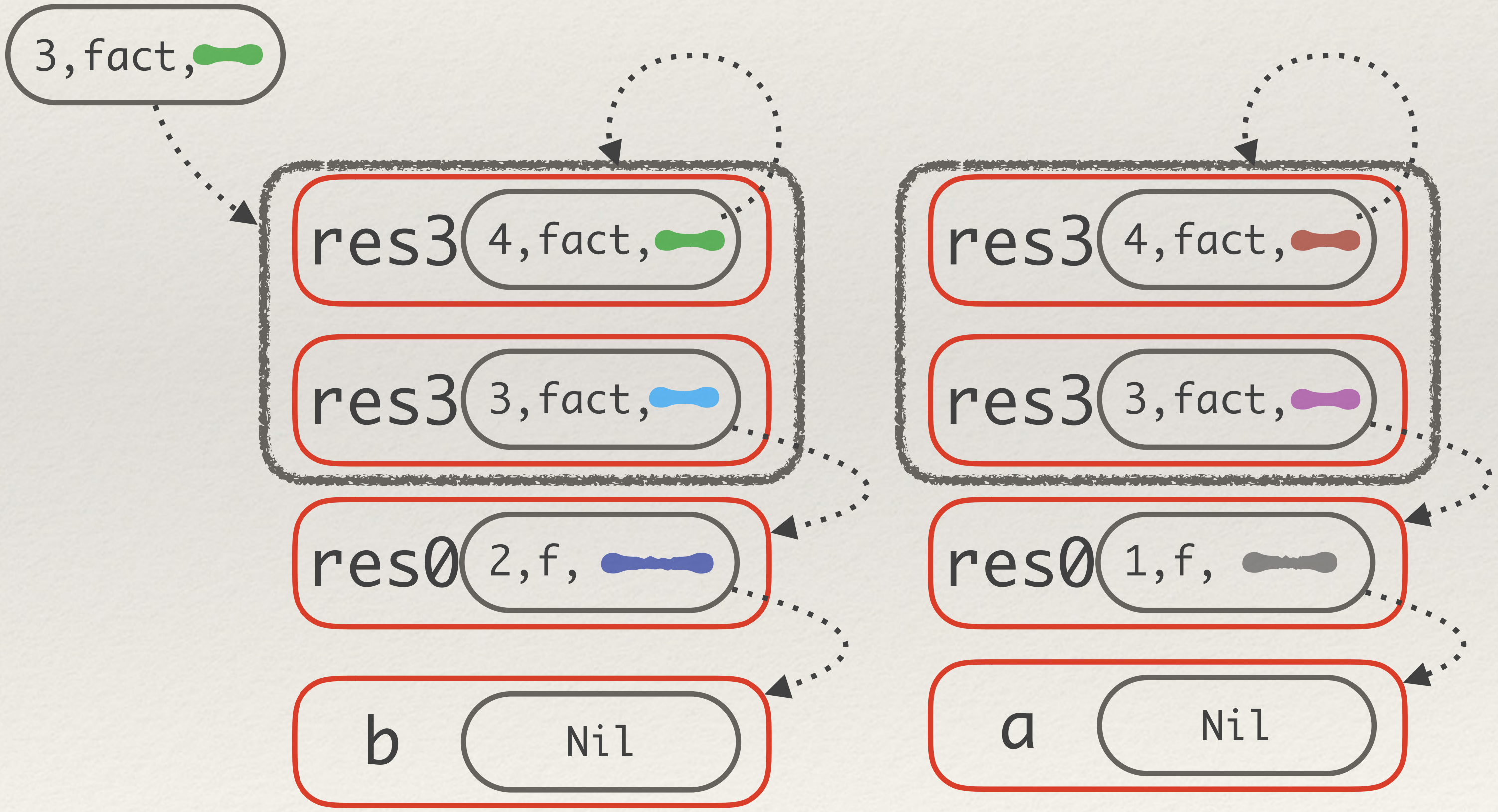
Current History: {fact,f,a,n}



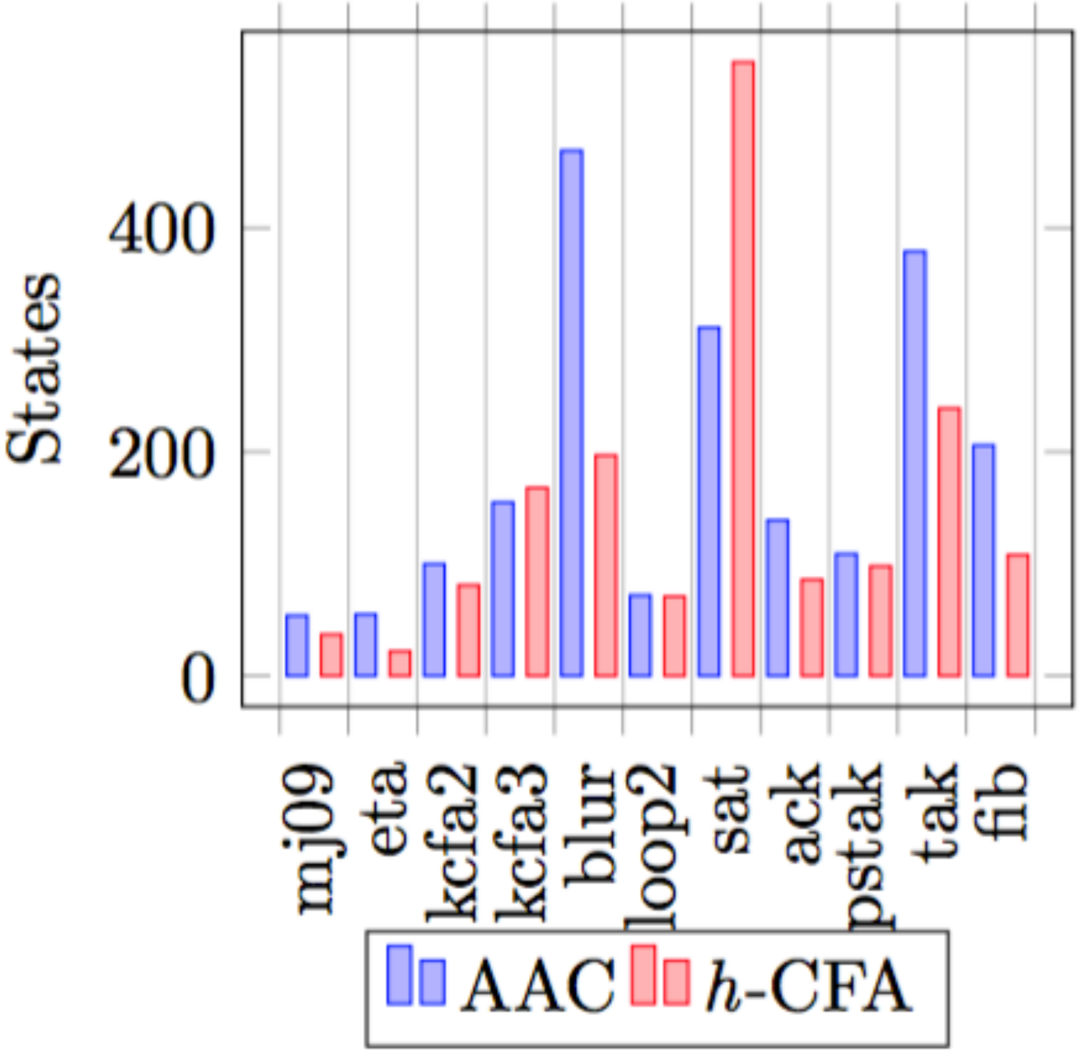
h-CFA

```
let fact = function fact(x) {
   let res1 = x < 1;
   if(res1) {
      return 1;
   } else {
      let res2 = x - 1;
      let res3 = fact(res2)4;
      let res4 = x * res3;
      return res4;
   }
}

var f = function (n) {
   let res0 = fact(n)3;
   return res0;
}

let a = f(10)1;
let b = f(20)2;
```

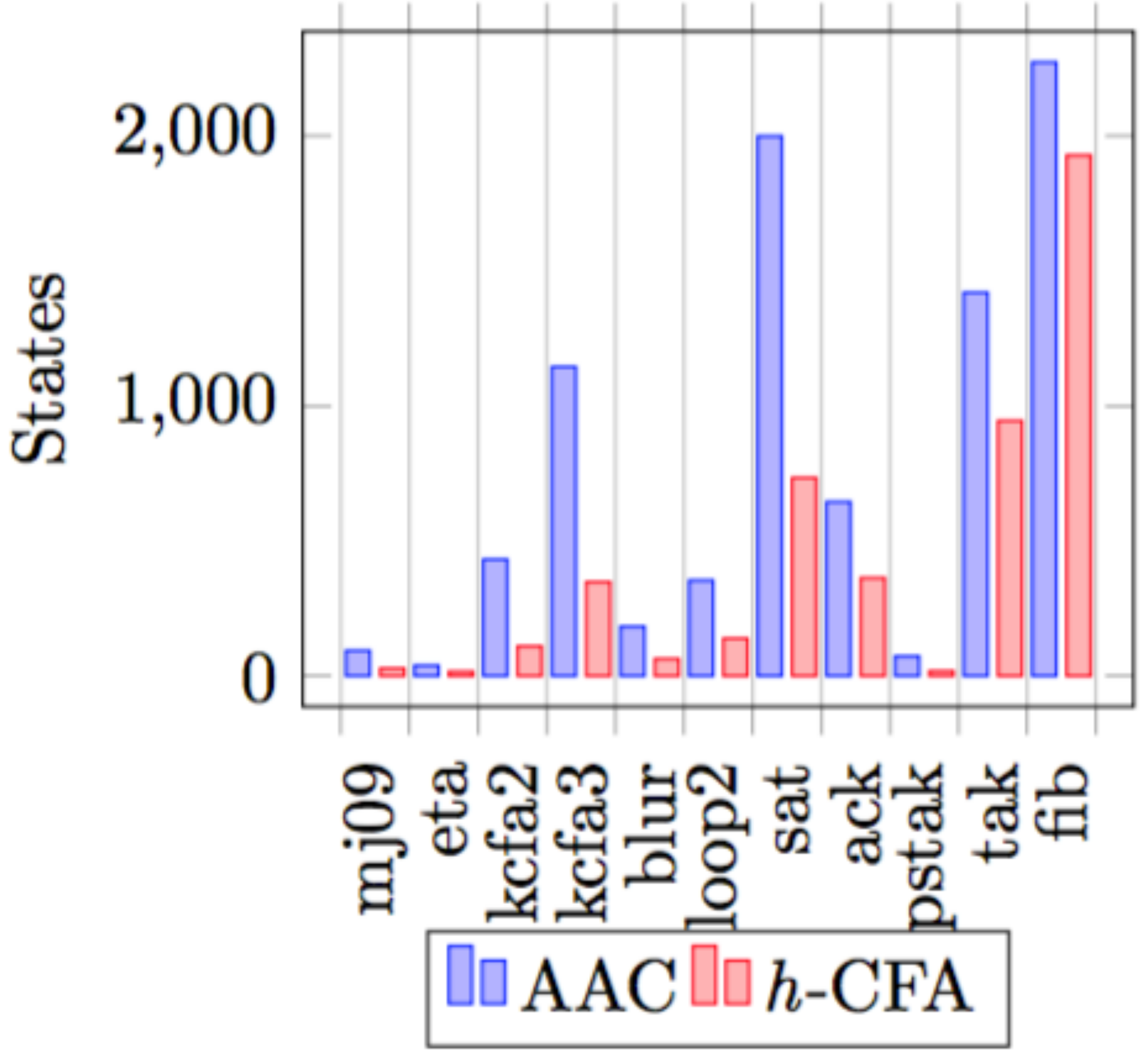
Current History:
{fact,f,a,n,x,res1,res2}



Performance

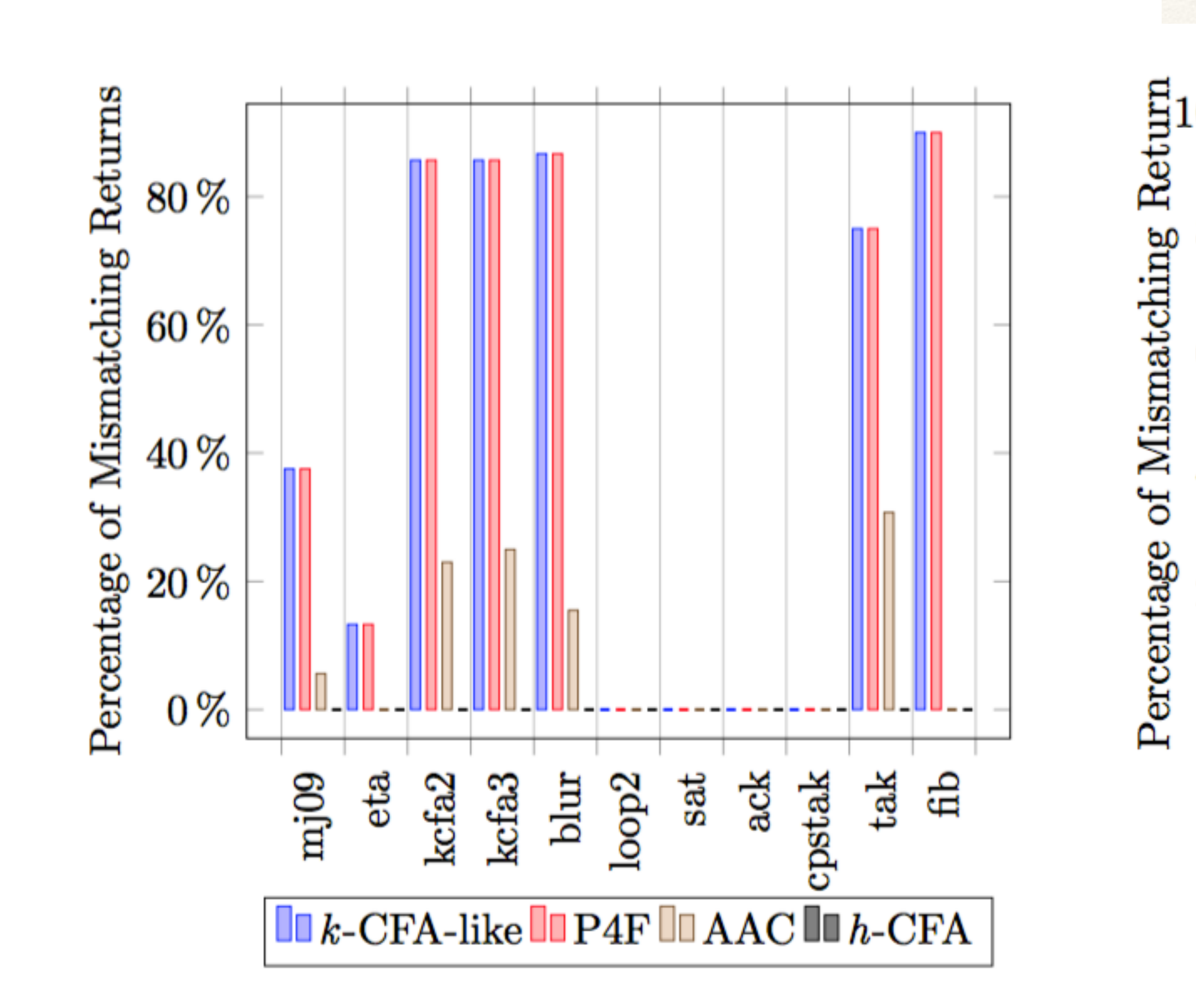


Monovariant Analysis

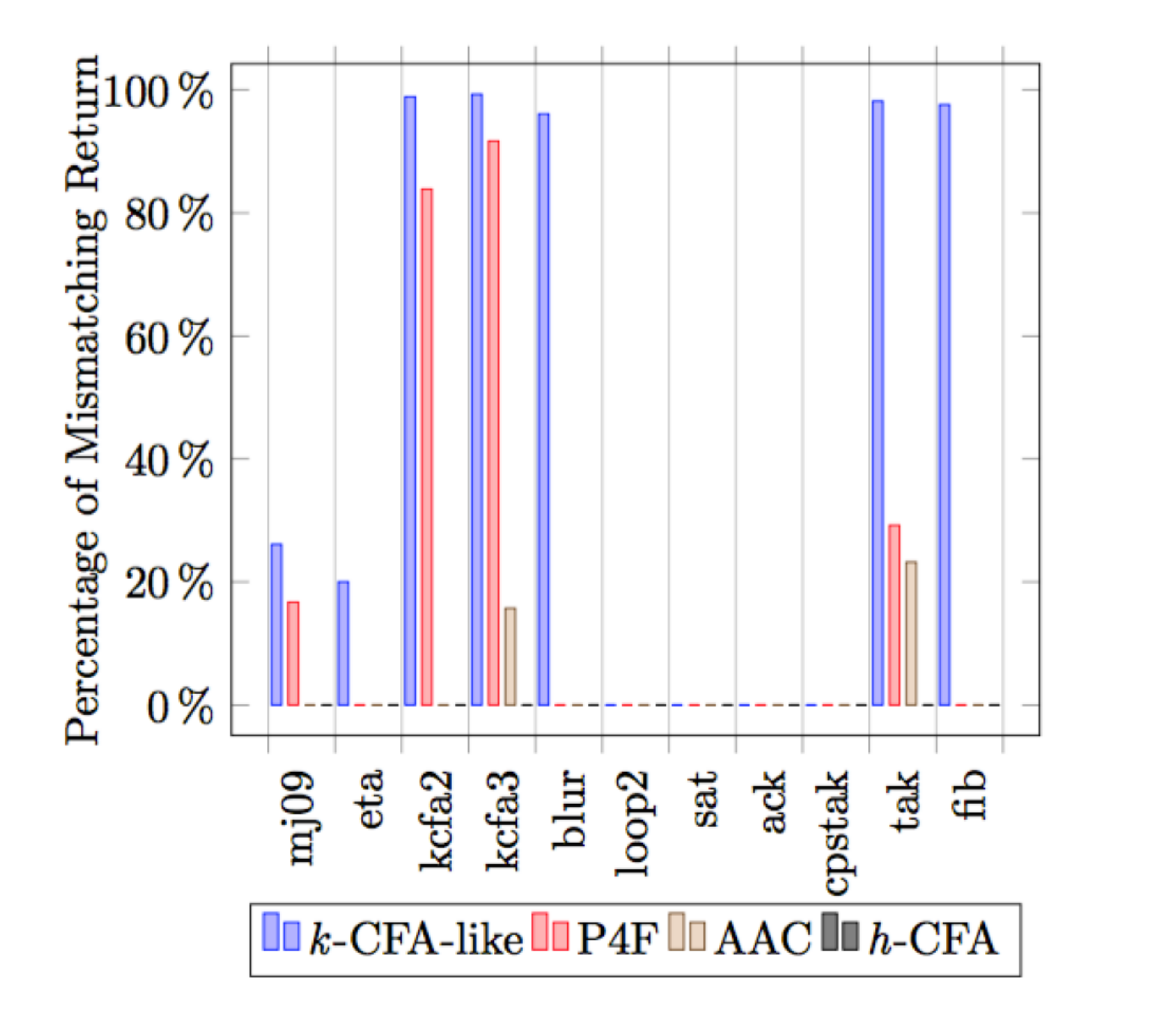


1-Call-Site Sensitive Analysis

Precision



Monovariant Analysis



1-Call-Site Sensitive Analysis

JsCFA

A Static Analyzer for JavaScript

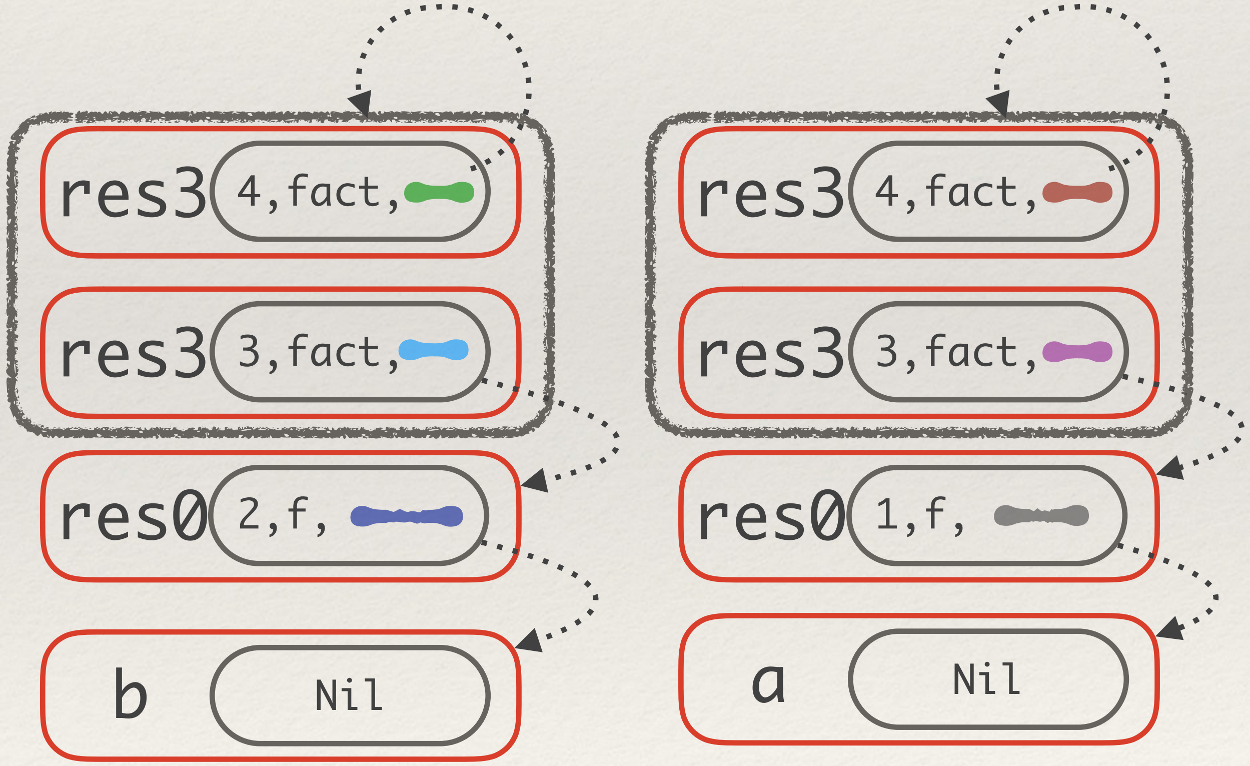
Pushdown Control Flow Analysis on Direct AST

Abstract Garbage Collection for Local Variables

Configureable Context-Sensitivity for Heap Variables

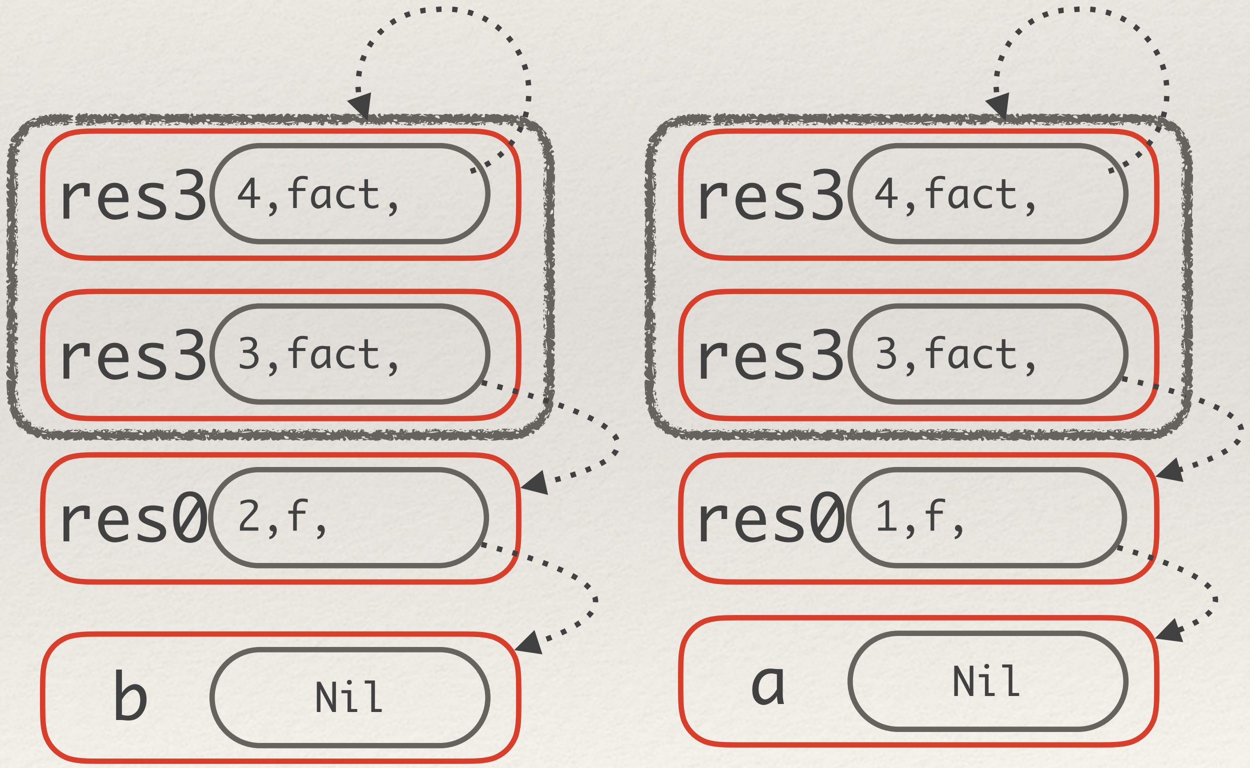
h-CFA without History

```
let fact = function fact(x) {
  let res1 = x < 1;
  if(res1) {
    return 1;
  } else {
    let res2 = x - 1;
    let res3 = fact(res2)[4];
    let res4 = x * res3;
    return res4;
  }
}

var f = function (n) {
  let res0 = fact(n)[3];
  return res0;
}

let a = f(10)[1];
let b = f(20)[2];
```



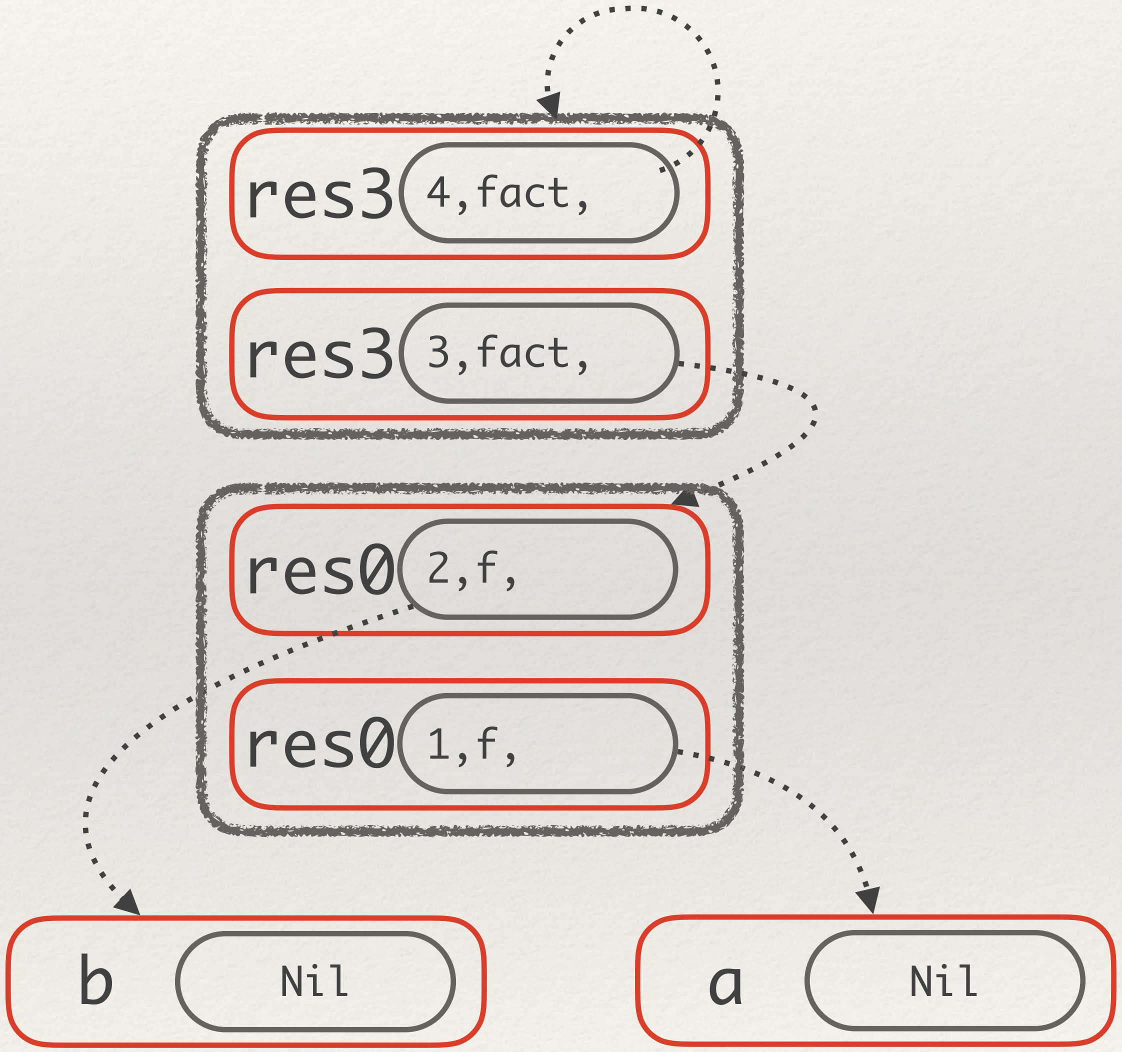
h-CFA without History

```
let fact = function fact(x) {
  let res1 = x < 1;
  if(res1) {
    return 1;
  } else {
    let res2 = x - 1;
    let res3 = fact(res2)[4];
    let res4 = x * res3;
    return res4;
  }
}

var f = function (n) {
  let res0 = fact(n)[3];
  return res0;
}

let a = f(10)[1];
let b = f(20)[2];
```



h-CFA without History

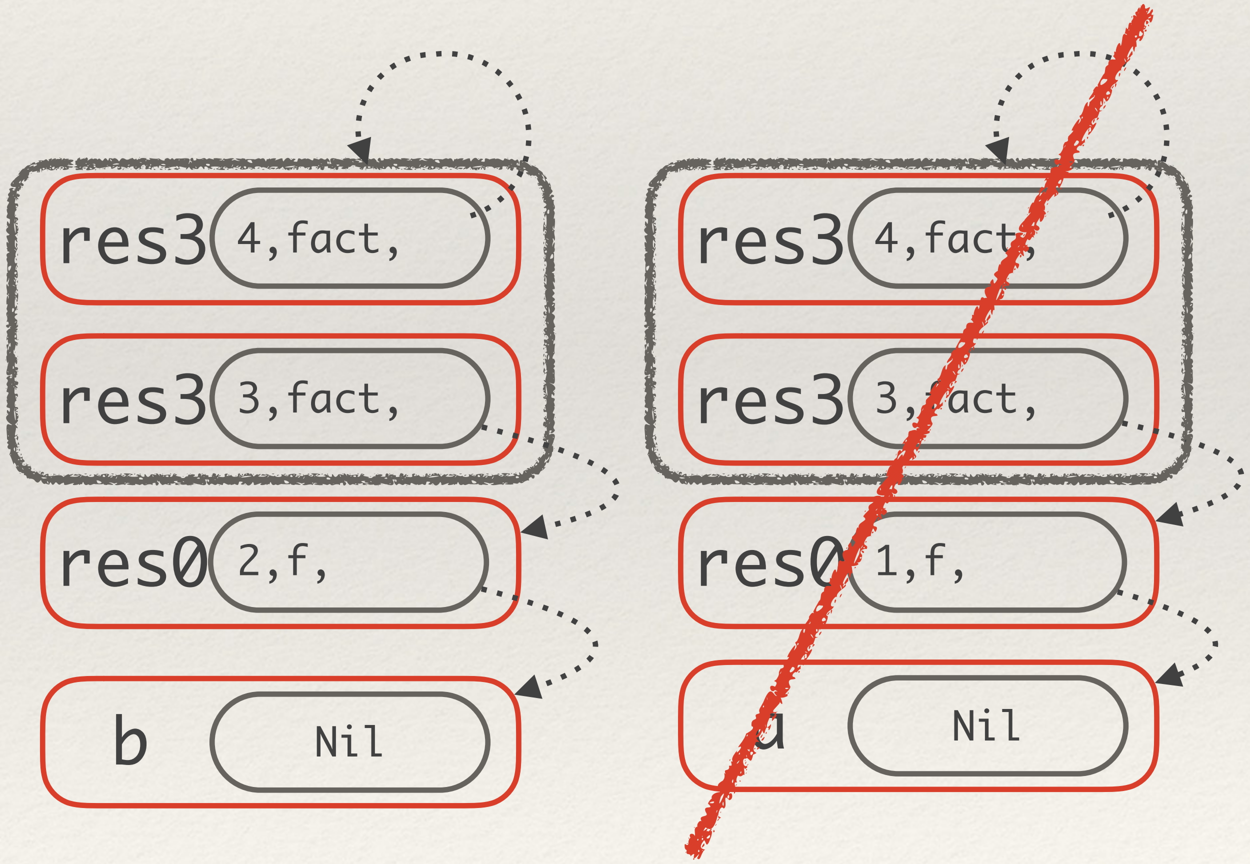
```
let fact = function fact(x) {
  let res1 = x < 1;
  if(res1) {
    return 1;
  } else {
    let res2 = x - 1;
    let res3 = fact(res2)[4];
    let res4 = x * res3;
    return res4;
  }
}

var f = function (n) {
  let res0 = fact(n)[3];
  return res0;
}

let a = f(10)[1];
let b = f(20)[2];
```



h-CFA without History

```
let fact = function fact(x) {
  let res1 = x < 1;
  if(res1) {
    return 1;
  } else {
    let res2 = x - 1;
    let res3 = fact(res2)[4];
    let res4 = x * res3;
    return res4;
  }
}

var f = function (n) {
  let res0 = fact(n)[3];
  return res0;
}

let a = f(10)[1];
let b = f(20)[2];
```

Abstract
Garbage
Collection

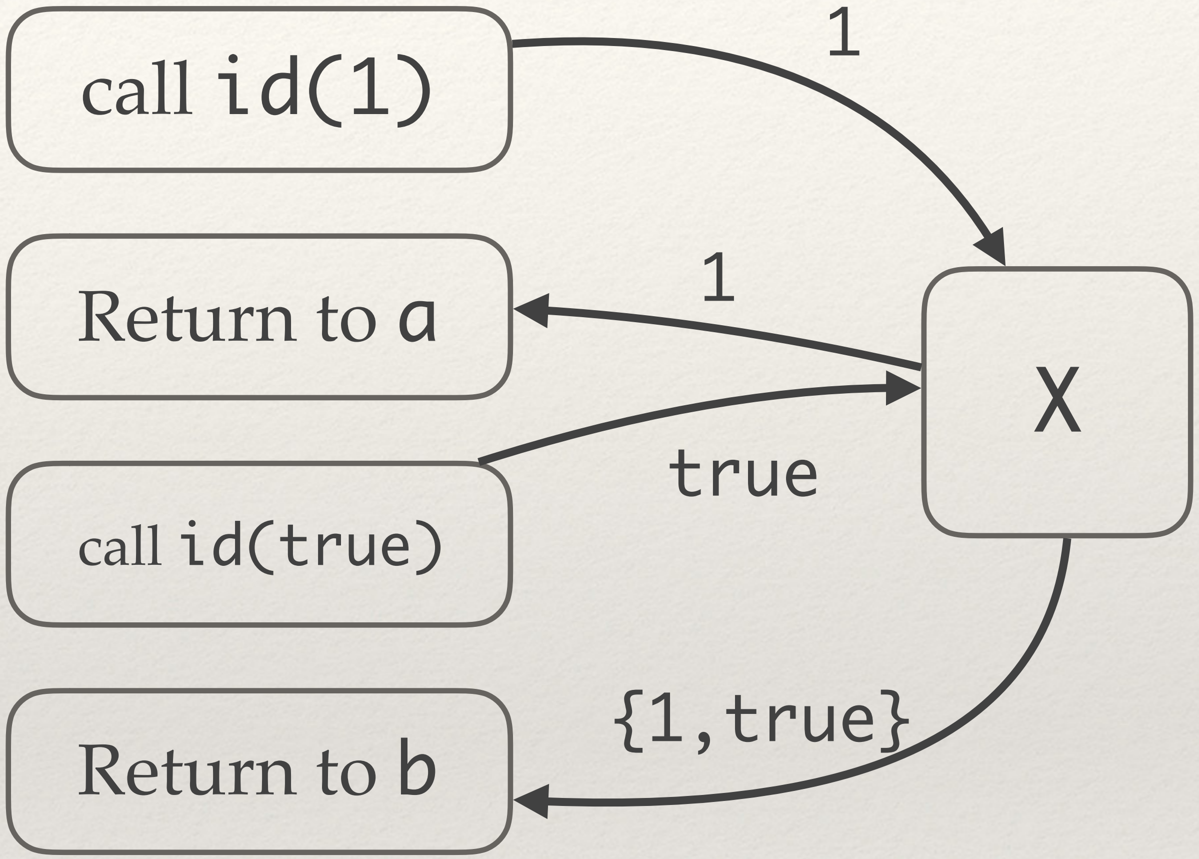


res3 4,fact,
res3 3,fact,
res0 2,f,
b Nil

~~res3 4,fact,~~
~~res3 3,fact,~~
~~res0 1,f,~~
~~a Nil~~

Spurious Data Flows

```
function id(x) {
  return x;
}

var a = id(1);
var b = id(true);
```

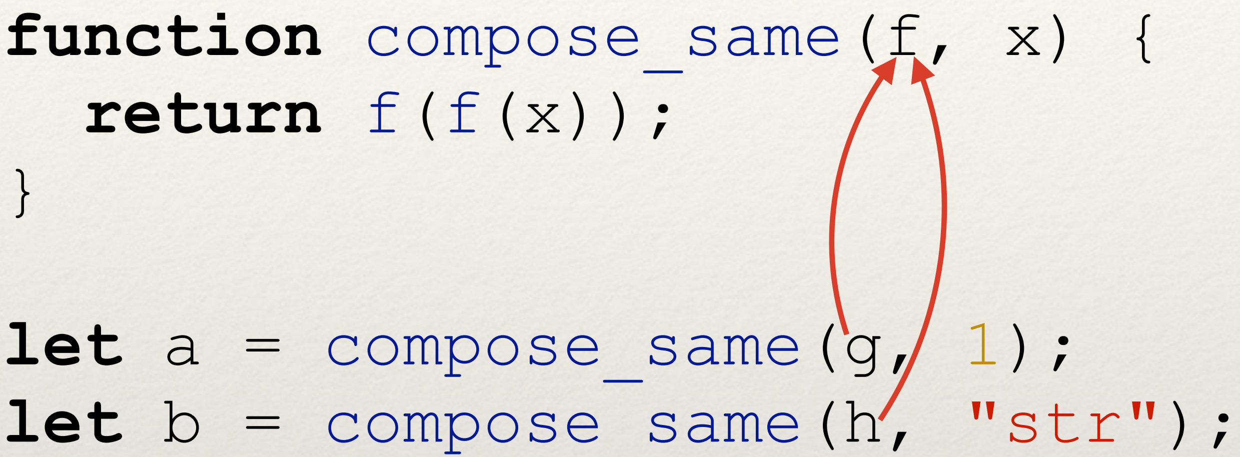


x → {1, true}

a → {1}

b → {1, ture}

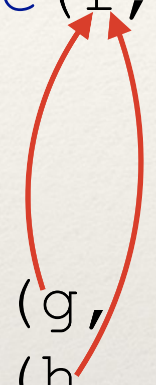
Fake Rebinding



```
function compose_same(f, x) {
  return f(f(x));
}

let a = compose_same(g, 1);
let b = compose_same(h, "str");
```

During Runtime, `f` is always bound to the same function at the two call sites.

Fake Rebinding

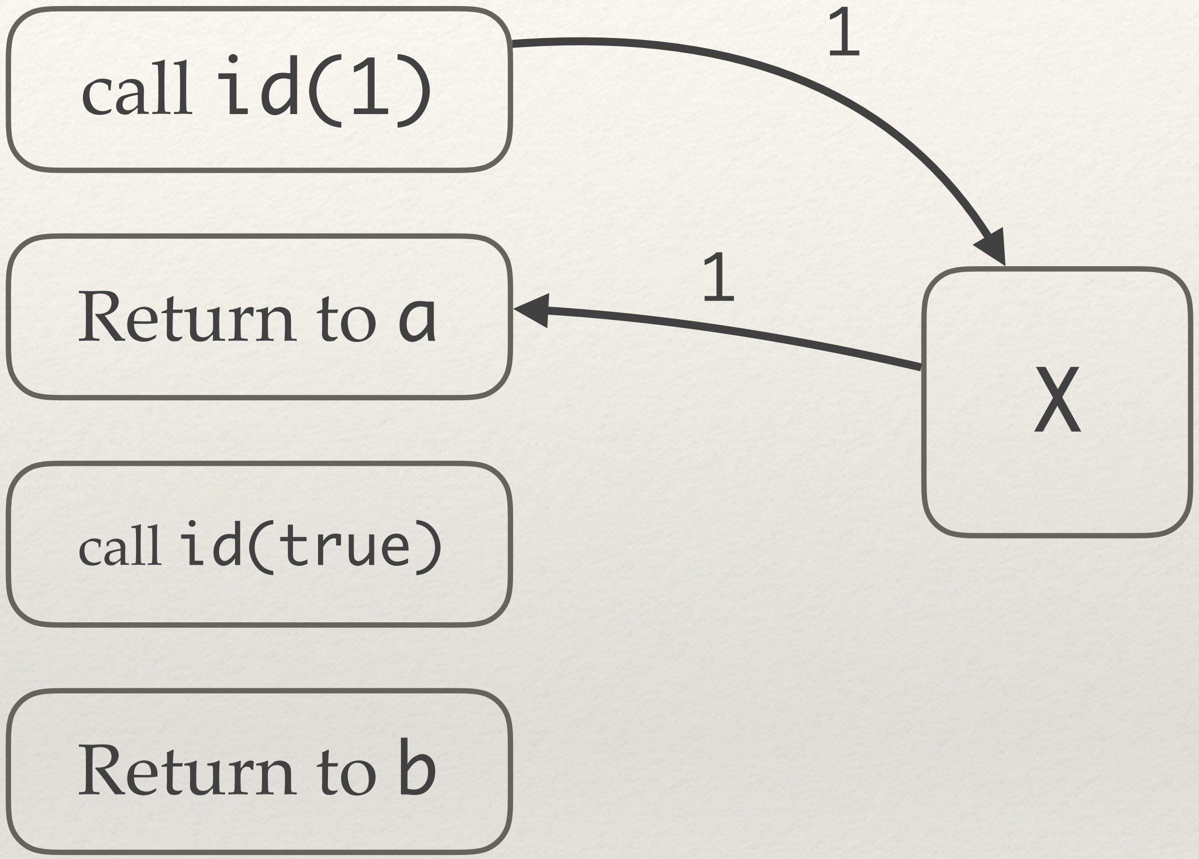


```
function compose_same(f, x) {
  return f(f(x));
}

let a = compose_same(g, 1);
let b = compose_same(h, "str");
```

```
g (g (x)) √
h (h (x)) √
g (h (x)) ×
h (g (x)) ×
```

Abstract GC for Values

```
function id(x) {
  return x;
}

var a = id(1);
var b = id(true);
```



call id(1)

1

Return to a

1

x

call id(true)

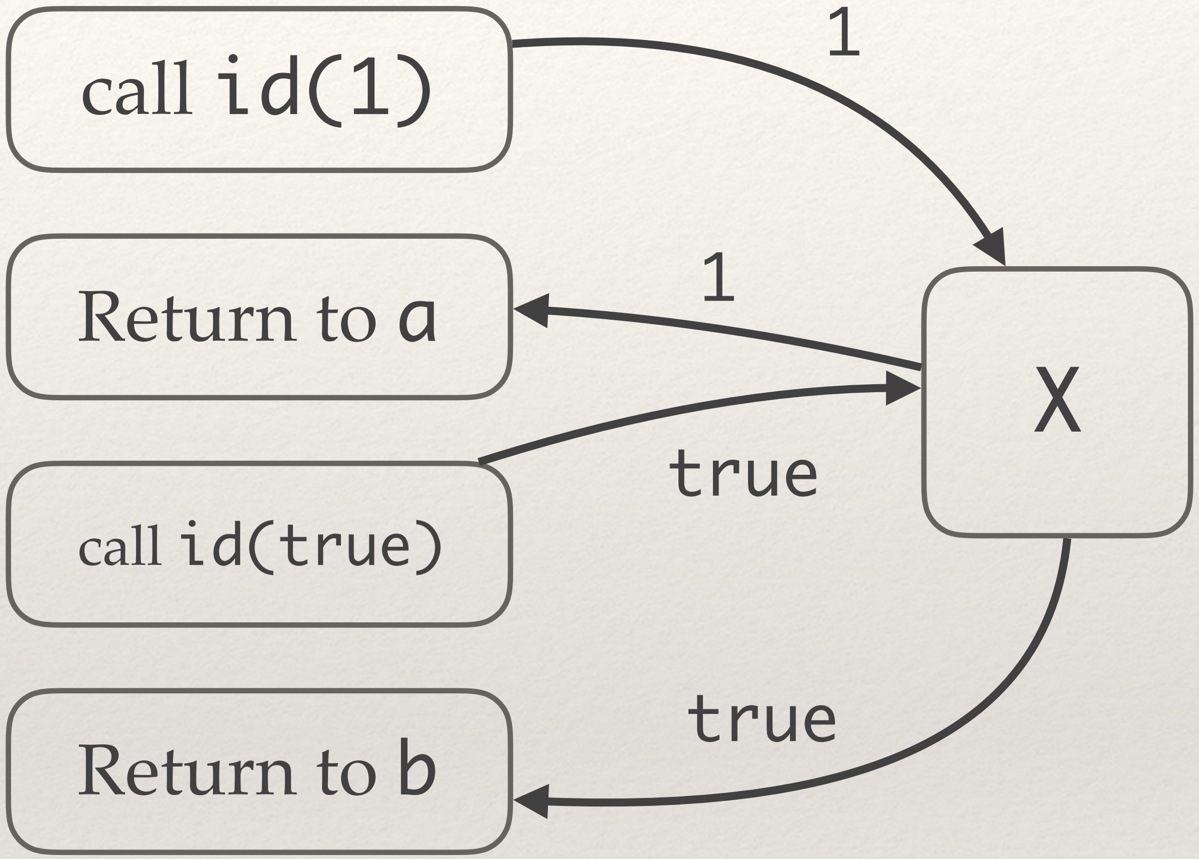
Return to b

x → ~~{1}~~ Garbage Collect

a → {1}

Abstract GC for Values

```
function id(x) {
  return x;
}

var a = id(1);
var b = id(true);
```



call id(1)

Return to a

call id(true)

Return to b

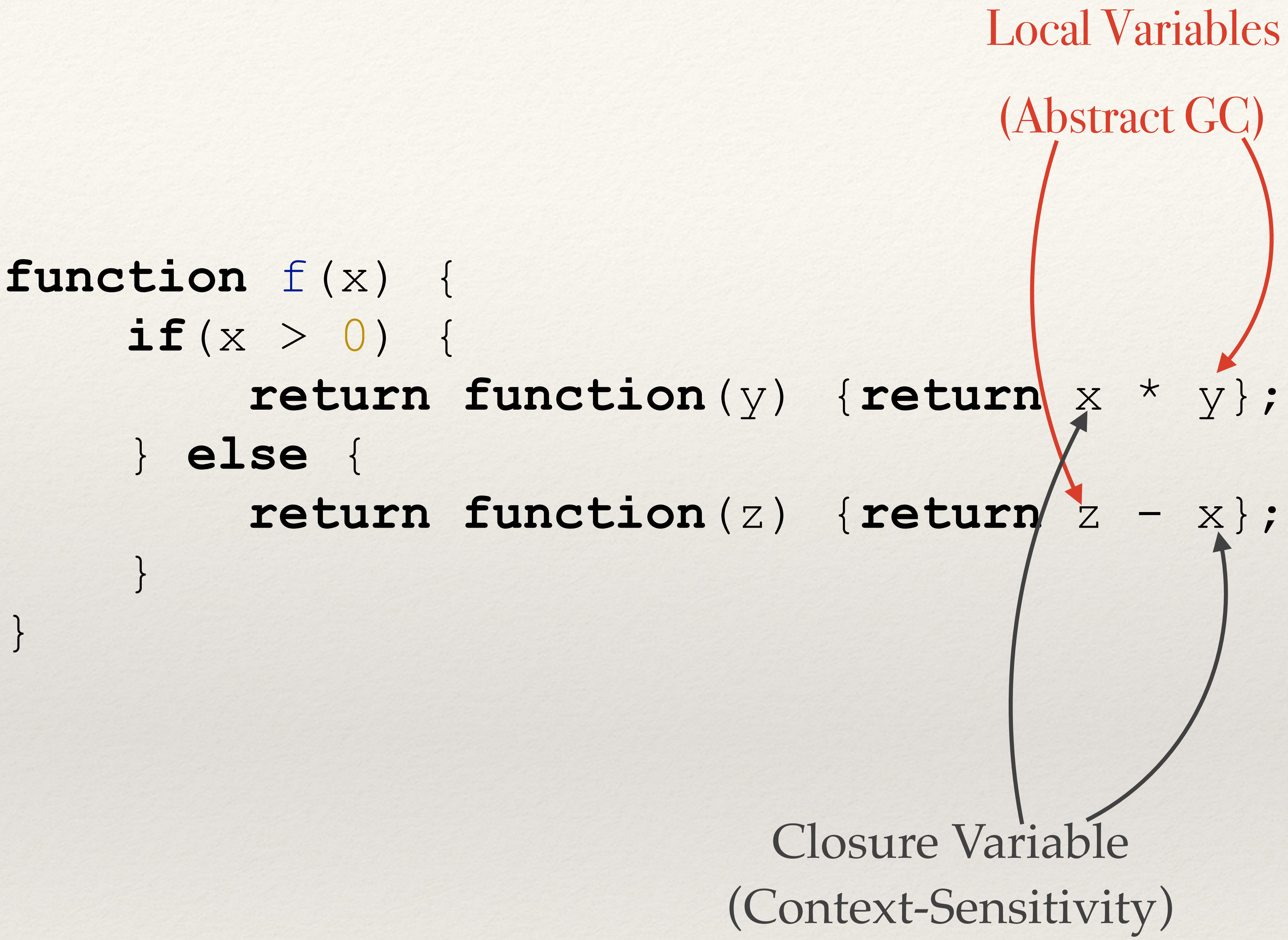
X

Global Table

x → {1,true}
a → {1}
b → {ture}

x → {true}
a → {1}
b → {ture}

Store



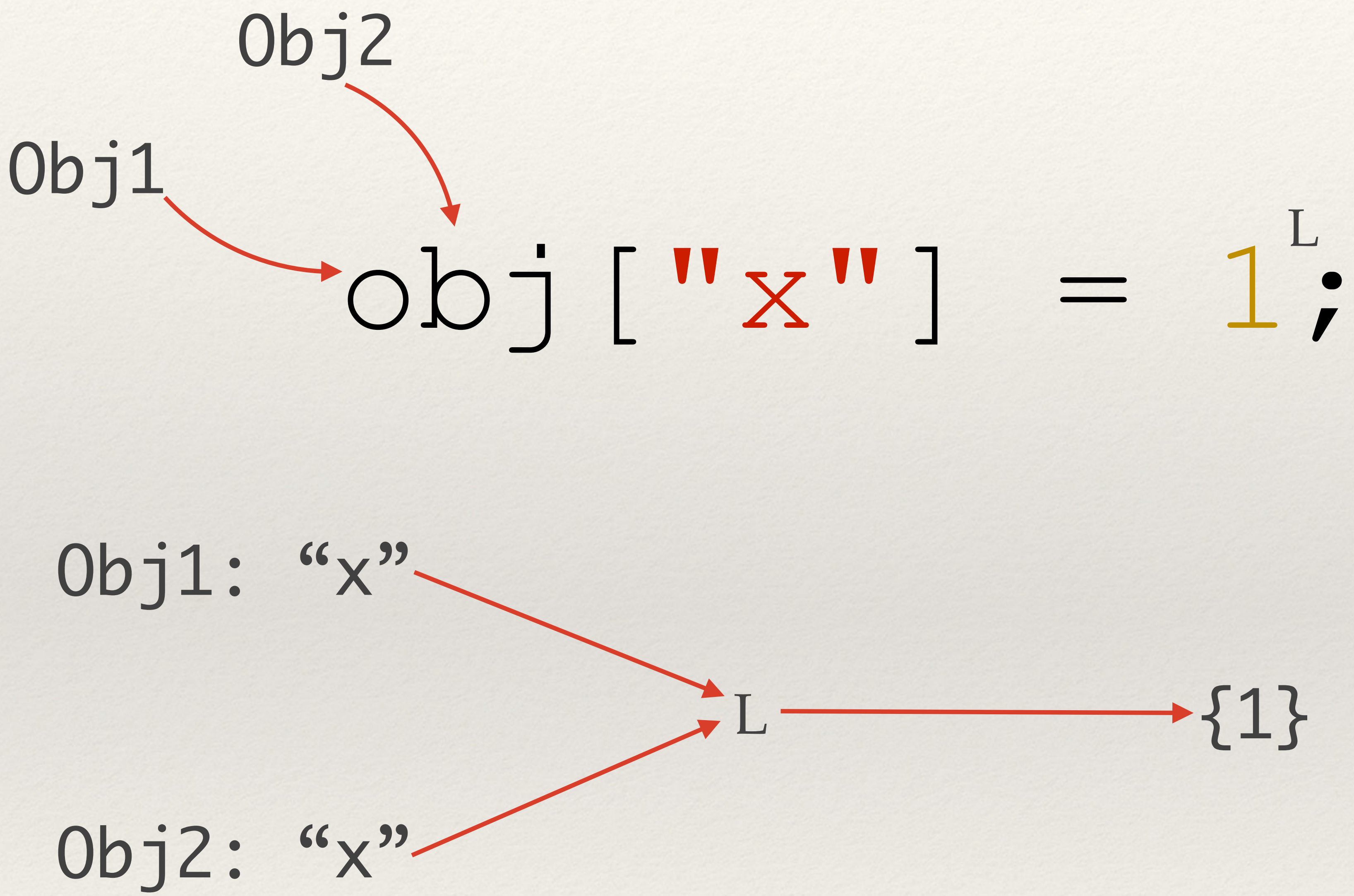
Local Variables

(Abstract GC)

```
function f(x) {
    if(x > 0) {
        return function(y) {return x * y};
    } else {
        return function(z) {return z - x};
    }
}
```

Closure Variable
(Context-Sensitivity)

Configureable Context Sensitivity



Obj2

Obj1

```
obj["x"] = 1;
```

L

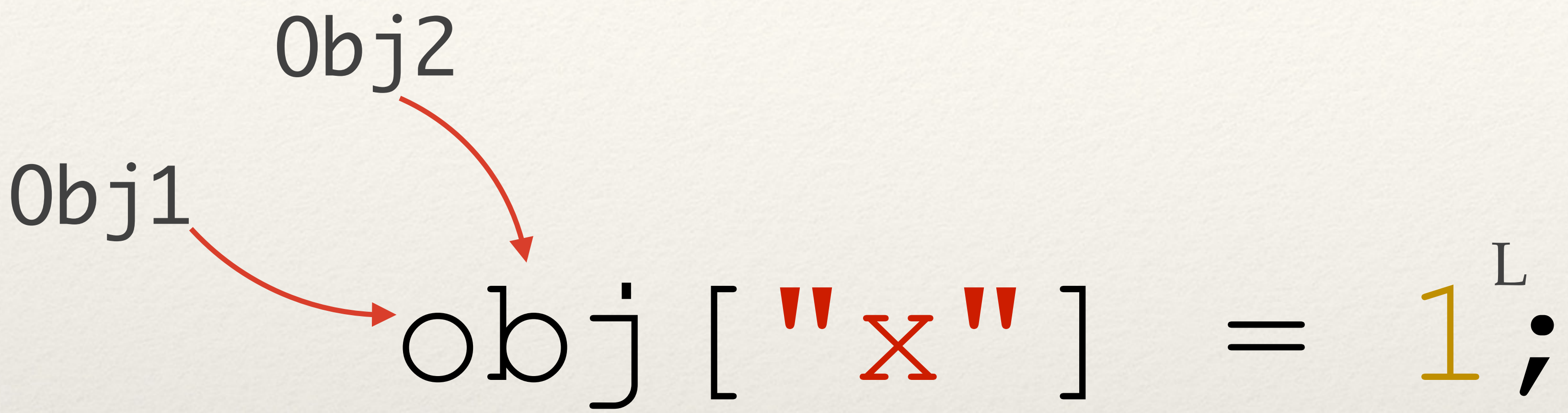
Obj1: “x”

L

{1}

Obj2: “x”

Configureable Context Sensitivity

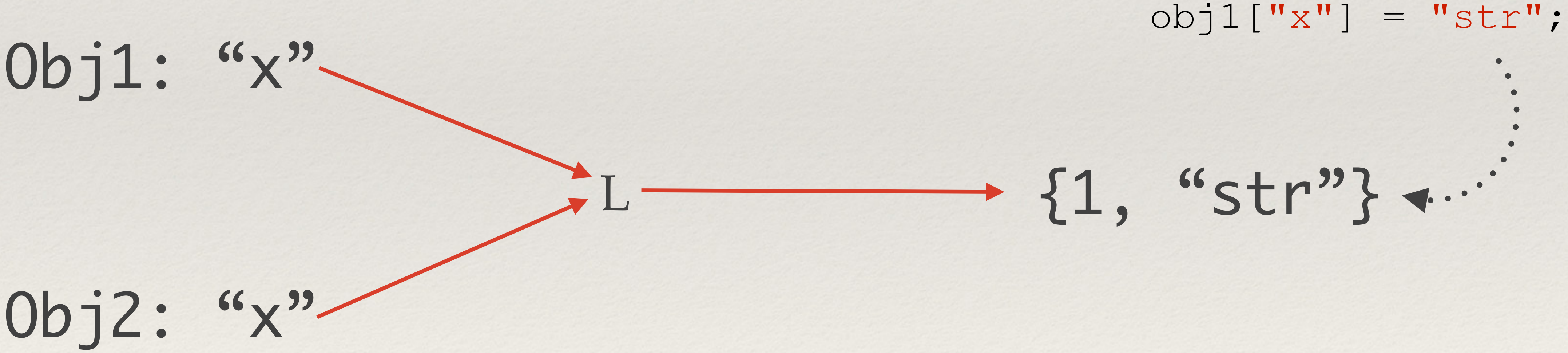


Obj2

Obj1

L

```
obj["x"] = 1;
```



```
obj1["x"] = "str";
```

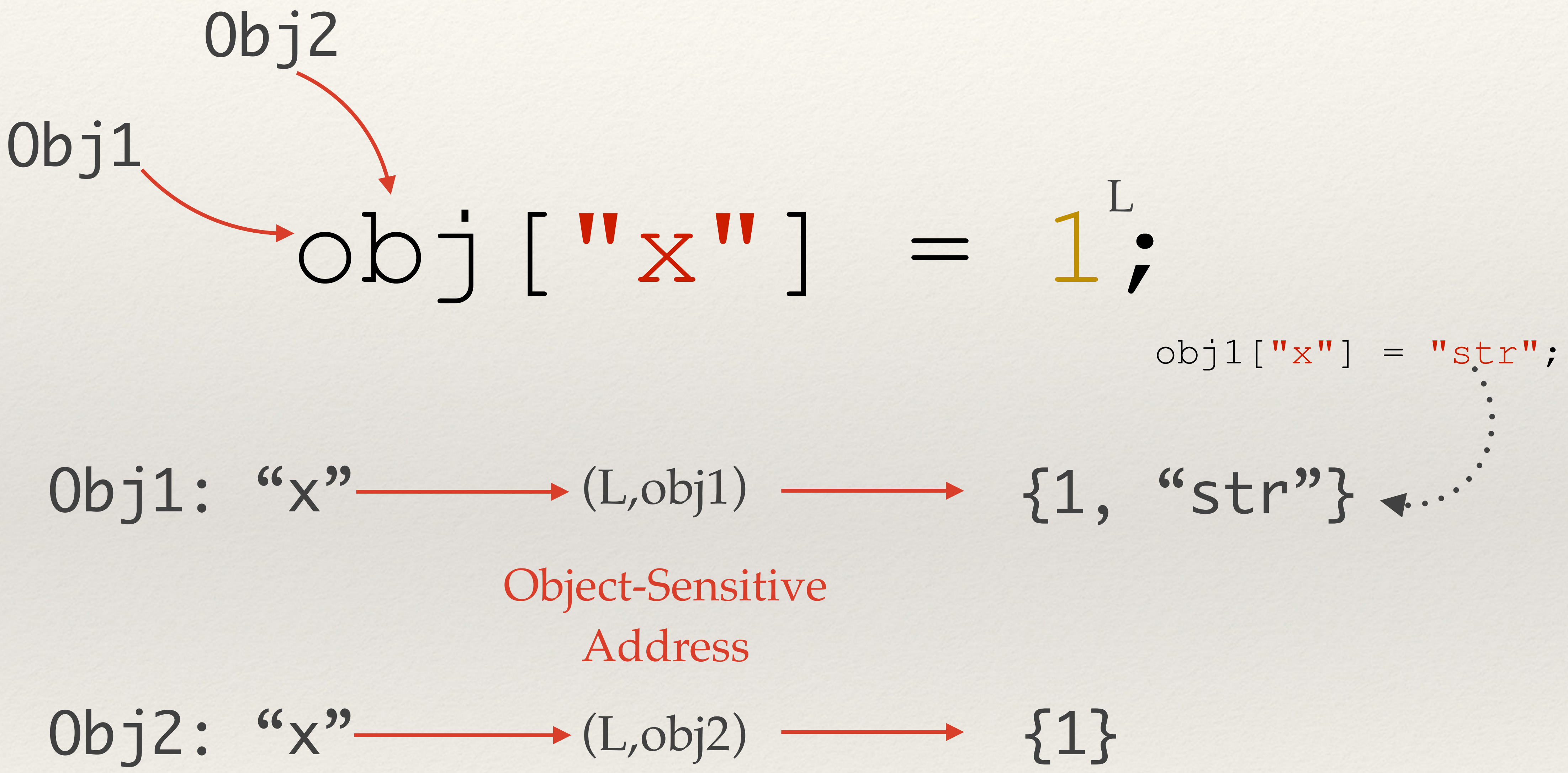
Obj1: “x”

L

{1, “str”}

Obj2: “x”

Configureable Context Sensitivity



Obj2

Obj1

```
obj["x"] = 1;
```

L

```
obj1["x"] = "str";
```

Obj1: “x” → (L,obj1) → {1, “str”}

Object-Sensitive
Address

Obj2: “x” → (L,obj2) → {1}

Conclusion

- *h*-CFA: perfect call/return matching for monovariance and polyvariance;
- JsCFA: pushdown control flow analysis cooperating with abstract GC (local) and context-sensitivity (heap).

Thank You