IFDS/IDE-based Inter-procedural Static Analysis of Software Product Lines

Eric Bodden

SECURE SOFTWARE ENGINEERING GROUP

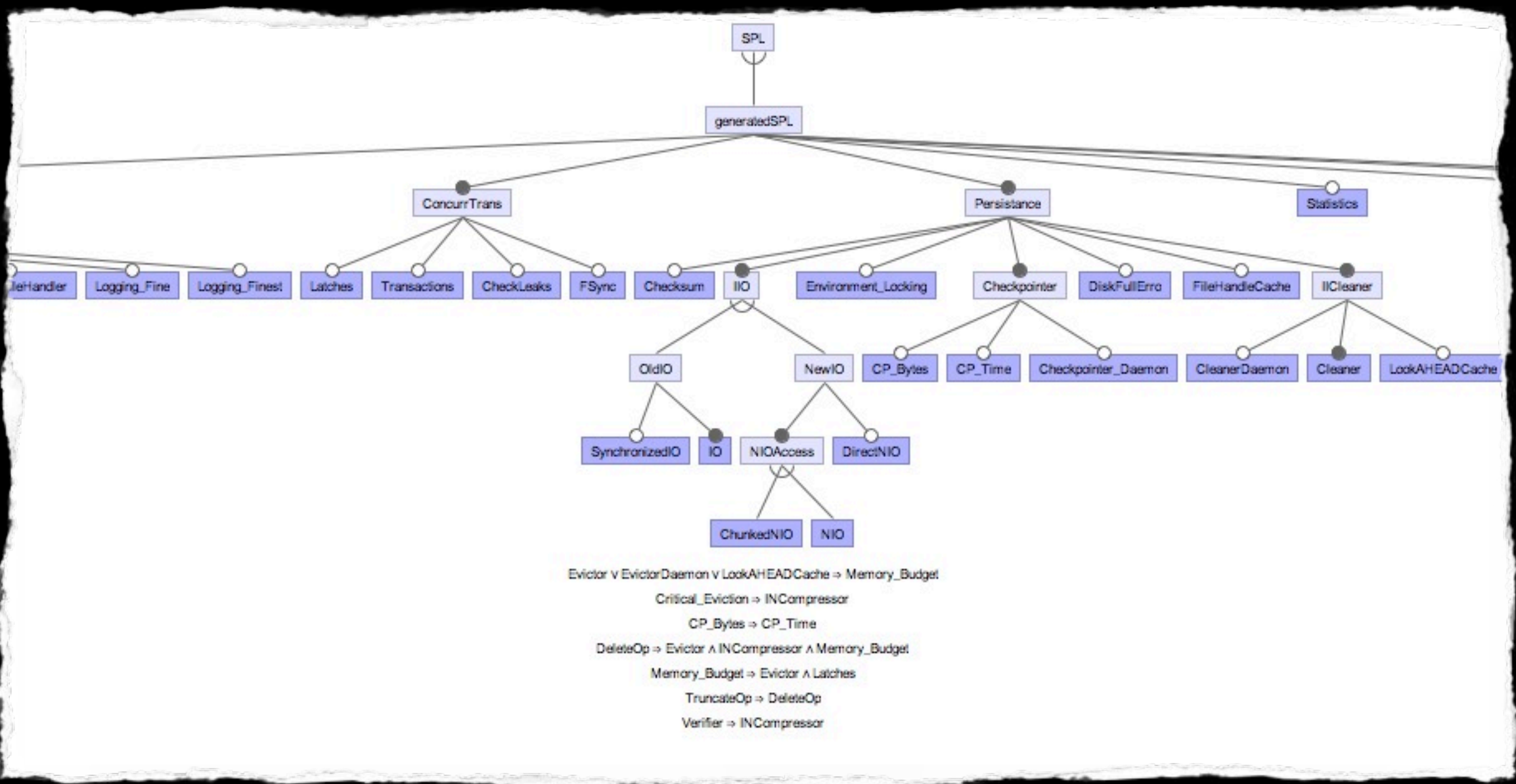
EC SPRIDE
EUROPEAN CENTER FOR SECURITY AND PRIVACY BY DESIGN

TECHNISCHE UNIVERSITÄT DARMSTADT

Product lines model variability

```
/**
 * Javadoc for this public method is generated via
 * the doc templates in the doc_src directory.
 */
public Sequence openSequence(Transaction txn,
                             DatabaseEntry key,
                             SequenceConfig config)
    throws DatabaseException {

    checkEnv();
    DatabaseUtil.checkForNullDbt(key, "key", true);
    checkRequiredDbState(OPEN, "Can't call Database.openSequence:");
    checkWritable("openSequence");
    trace(Level.FINEST, "Database.openSequence", txn, key, null, null);

    return new Sequence(this, txn, key, config);
}
```

Feature model of BerkelyDB



(excerpt)

Analyzing SPLs

- May just re-use existing analyses:
  - Generate all possible products, analyzing one at a time
  - Every product is a plain old Java program
    - Problem: may be millions
- Better: design analysis that is product-line aware!
  - Just analyze entire SPL in one shot!

Analyzing SPLs

- May just re-use existing analyses:
  - Generate all possible products, analyzing one at a time
  - Every product is a plain old Java program
    - Problem: may be millions
- Better: design analysis that is product-line aware!
  - Just analyze entire SPL in one shot!

This Talk!

Traditional info-flow analysis

```
void main() {
    int x = secret();
    int y = 0;
    x = 0;
    y = foo(x);
    print(y);
}

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

Traditional info-flow analysis

```
void main() {
    int x = secret();
    int y = 0;
    x = 0;
    y = foo(x);
    print(y);
}

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

FALSE

Now for SW Product Lines

```
void main() {
    int x = secret();
    int y = 0;
    x = 0;
    y = foo(x);
    print(y);
}

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

Feature F

Feature G

Feature H

Now for SW Product Lines

```
void main() {
    int x = secret();
    int y = 0;
    x = 0;
    y = foo(x);
    print(y);
}

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

Feature F

Feature G

Feature H

¬F ∧ G ∧ ¬ H

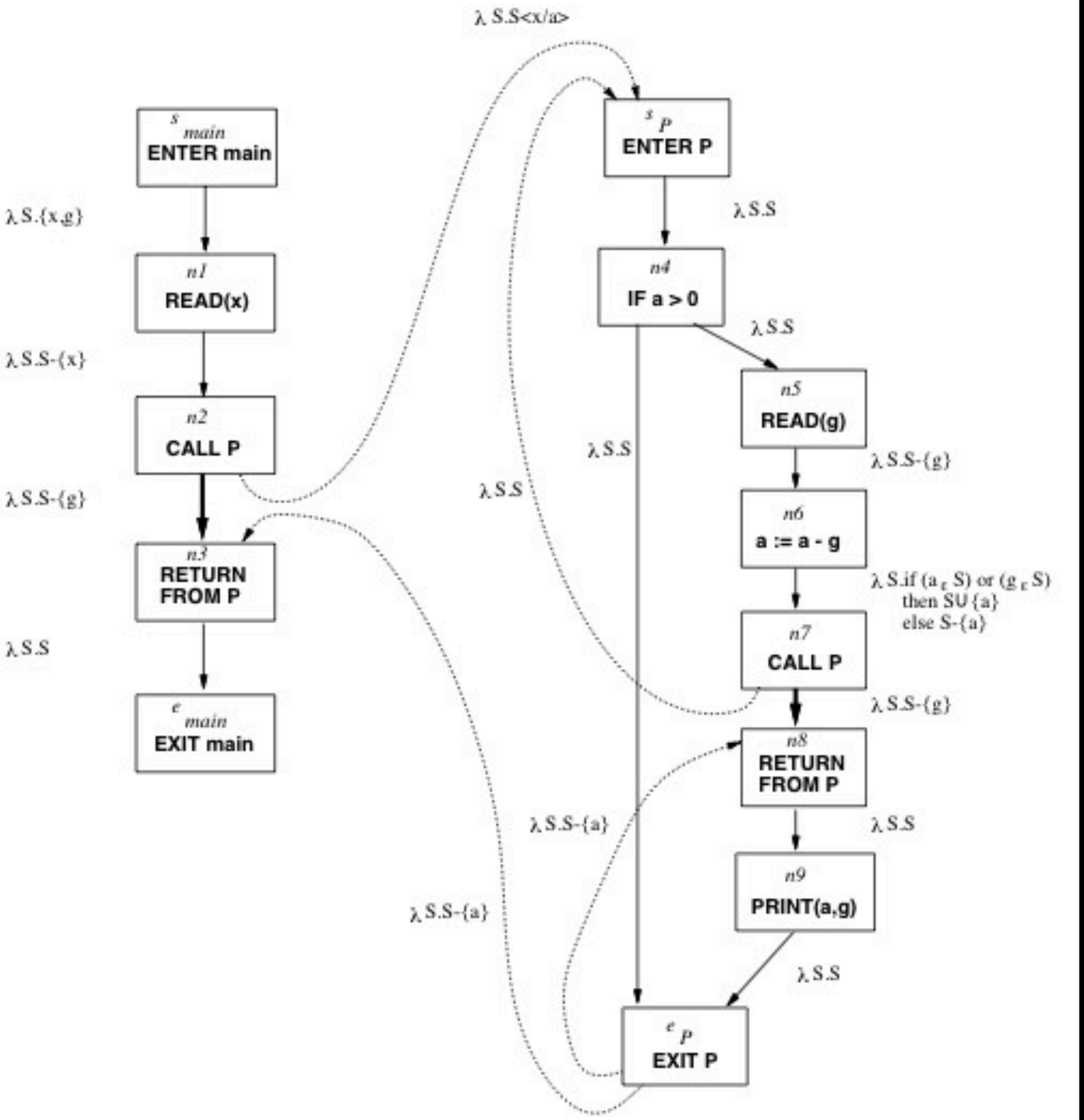
IFDS Framework

- Invented in 1995 by Reps, Horwitz and Sagiv
- Idea: reduce inter-procedural program-analysis problem to graph-reachability
- Works for any flow functions over finite domains that are distributive over the merge operator
- Covers a surprisingly large class of problems

Example Program

```
declare g: integer

program main
begin
  declare x: integer
  read(x)
  call P(x)
end


procedure P(value a: integer)
begin
  if (a > 0) then
    read(g)
    a := a - g
    call P(a)
    print(a, g)
  fi
end
```

Program Super graph



Idea: Specialize graph

Program Super Graph

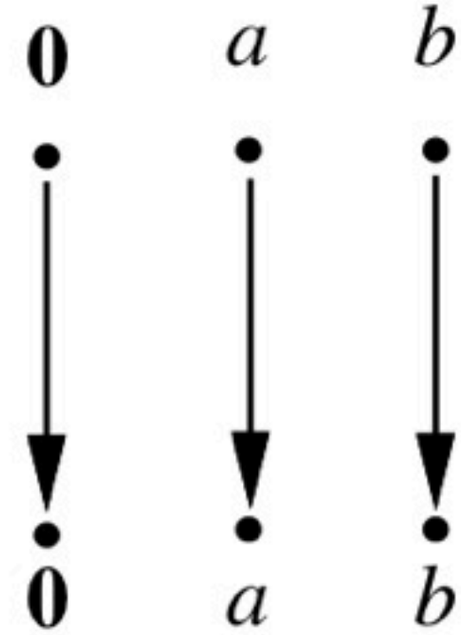
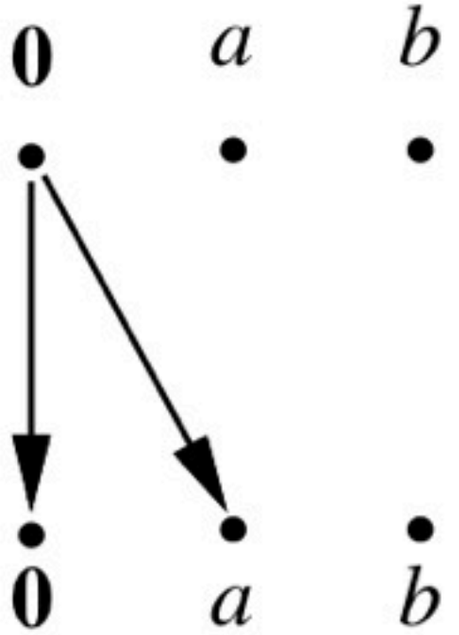
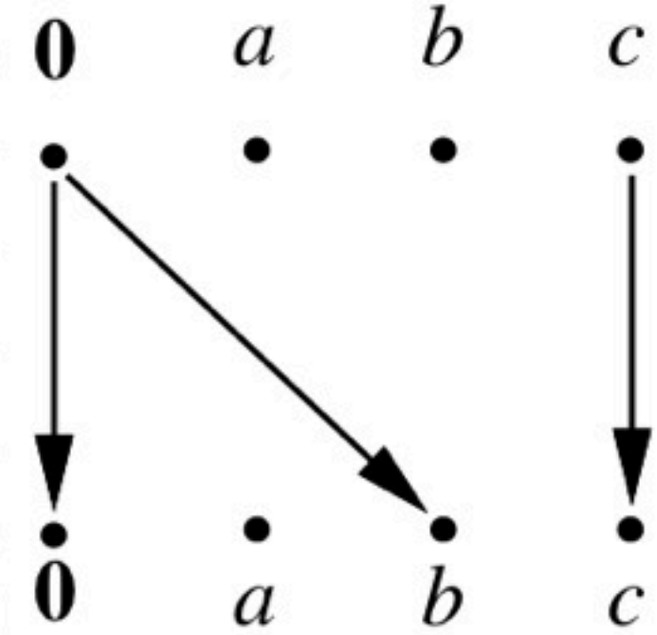
$\otimes$

Flow Functions

$=$

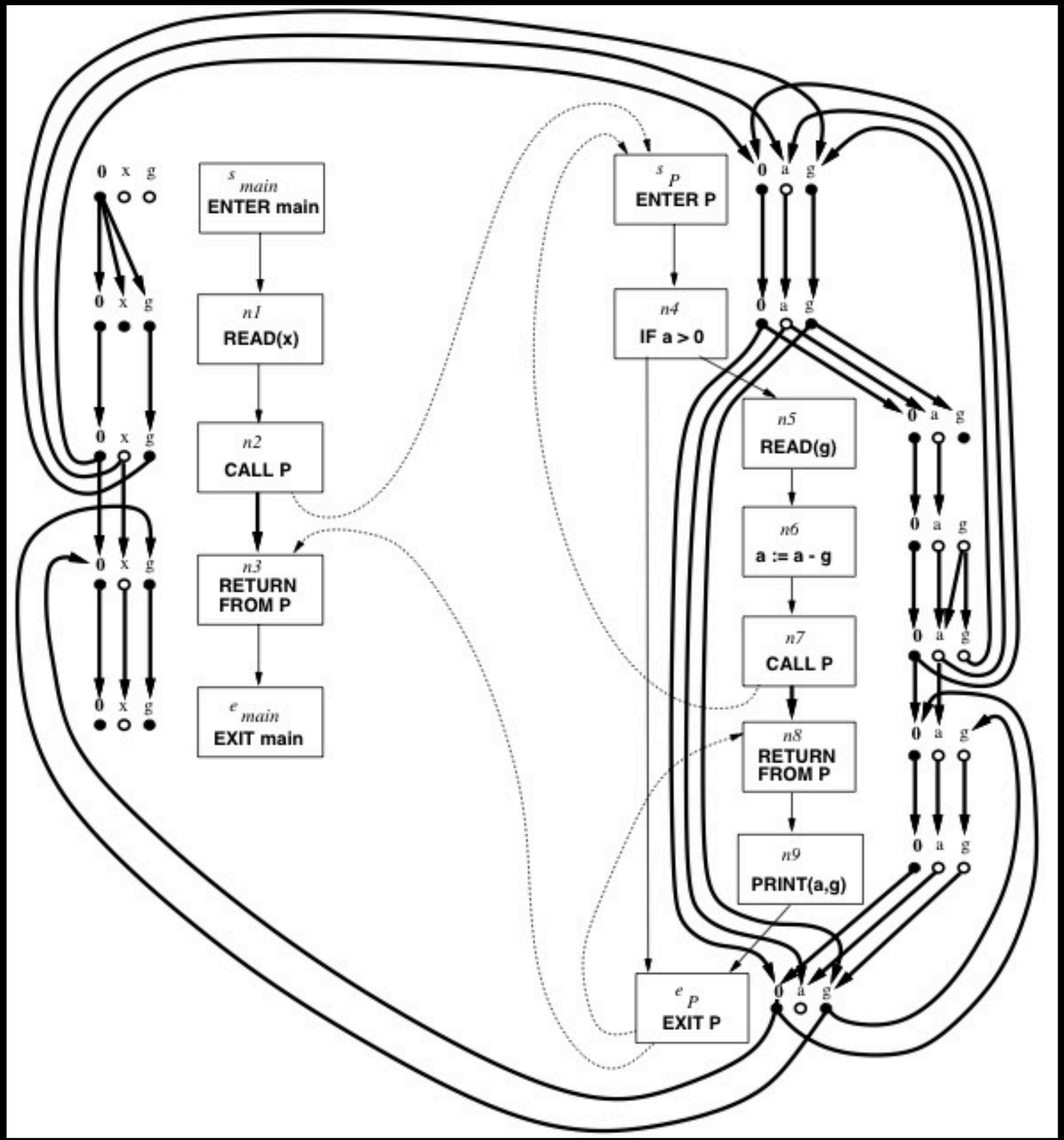
Exploded Super Graph

Flow functions as edges

id: $2^{\{a,b\}} \rightarrow 2^{\{a,b\}}$ **id** = $\lambda S.S$	**a**: $2^{\{a,b\}} \rightarrow 2^{\{a,b\}}$ **a** = $\lambda S.\{a\}$	**f**: $2^{\{a,b,c\}} \rightarrow 2^{\{a,b,c\}}$ **f** = $\lambda S.(S - \{a\}) \cup \{b\}$
		

Requires finite (or at least enumerable) domain

Exploded Super Graph



Previous example

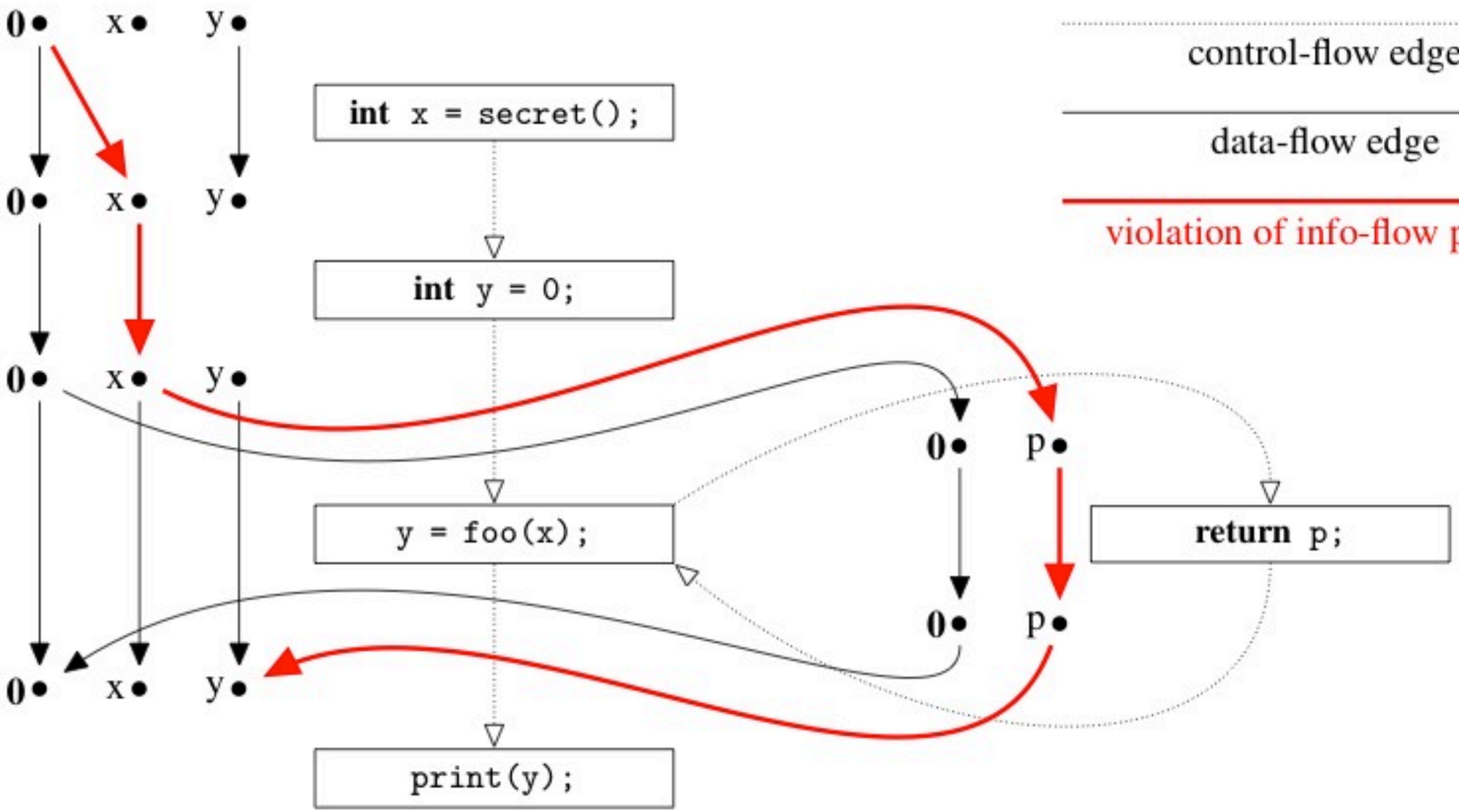
```
void main() {
    int x = secret();
    int y = 0;
    x = 0;          Feature F
    y = foo(x);     Feature G
    print(y);
}

int foo(int p) {
    p = 0;          Feature H
    return p;
}
```

$$\neg F \wedge G \wedge \neg H$$

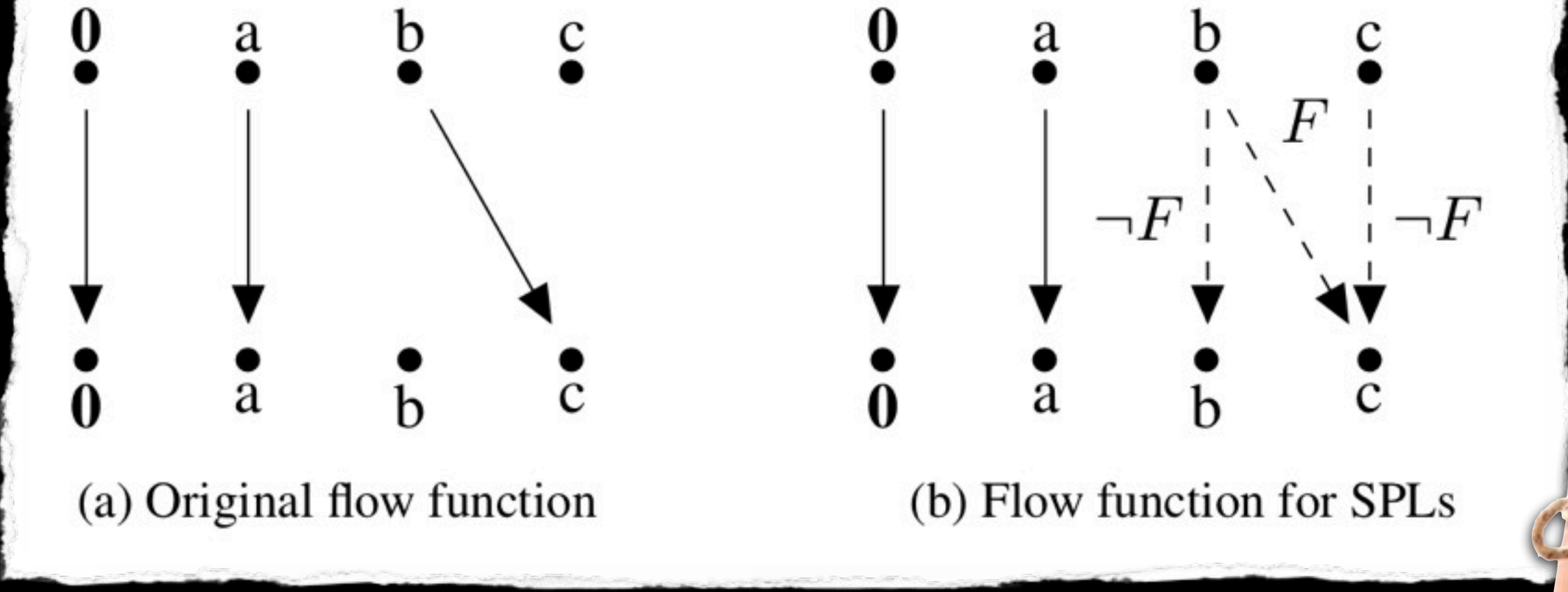
Problematic "Product"

```
void main() {
    int x = secret();
    int y = 0;

    y = foo(x);
    print(y);
}

int foo(int p) {

    return p;
}
```

IFDS Info-flow analysis



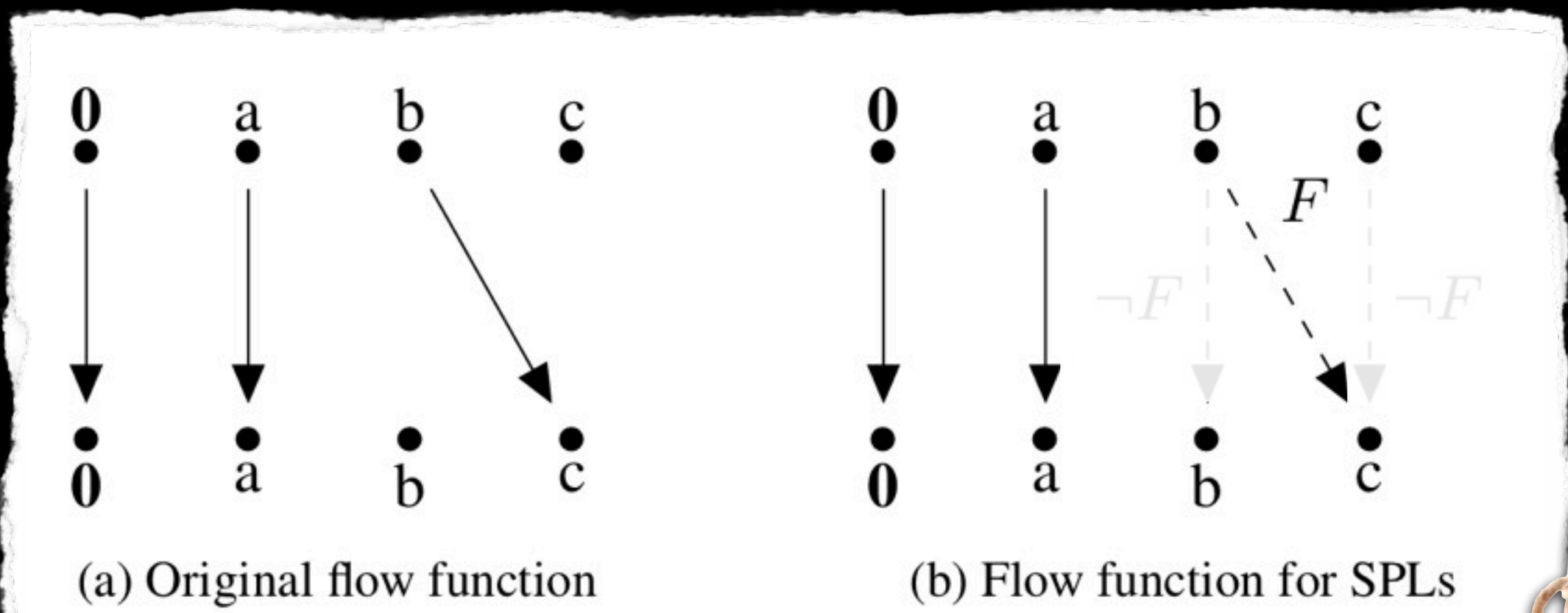
Stmt guarded by F...



(a) Original flow function

(b) Flow function for SPLs

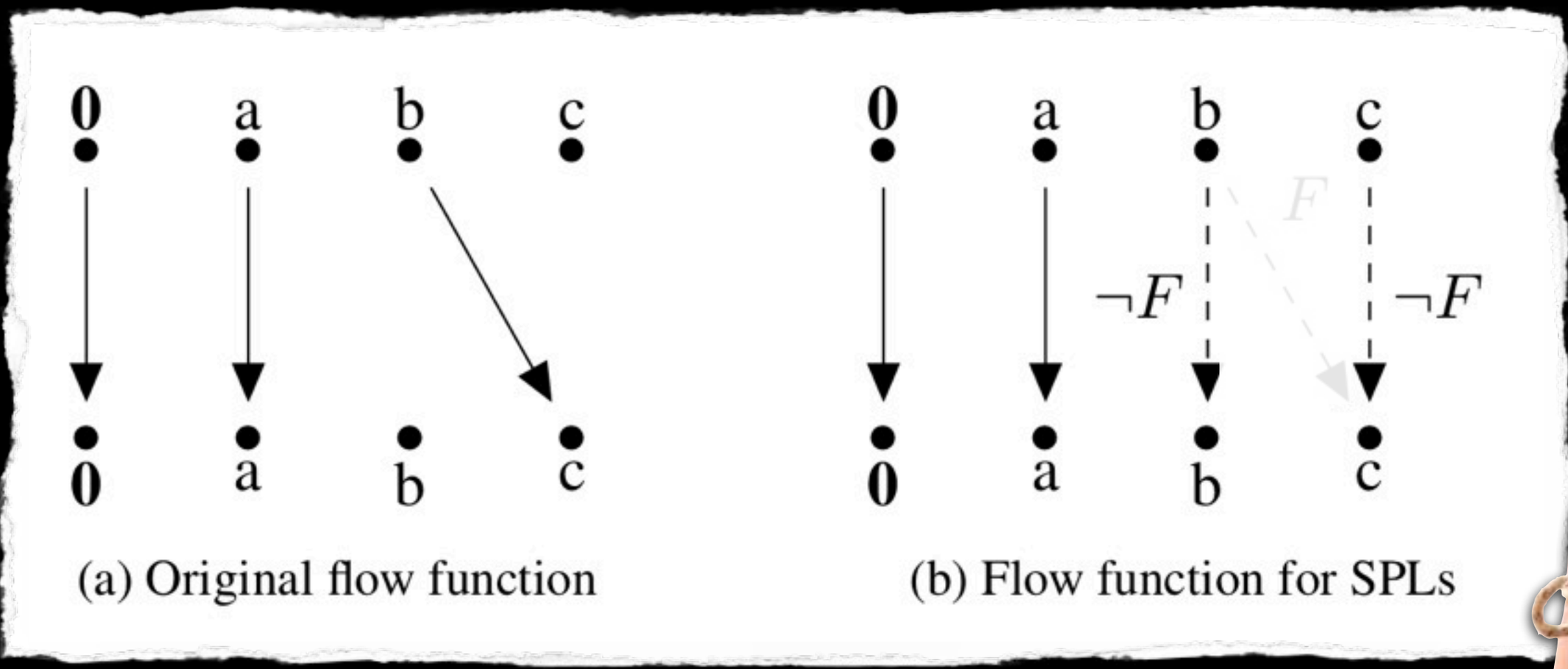
Stmt guarded by F...



(a) Original flow function

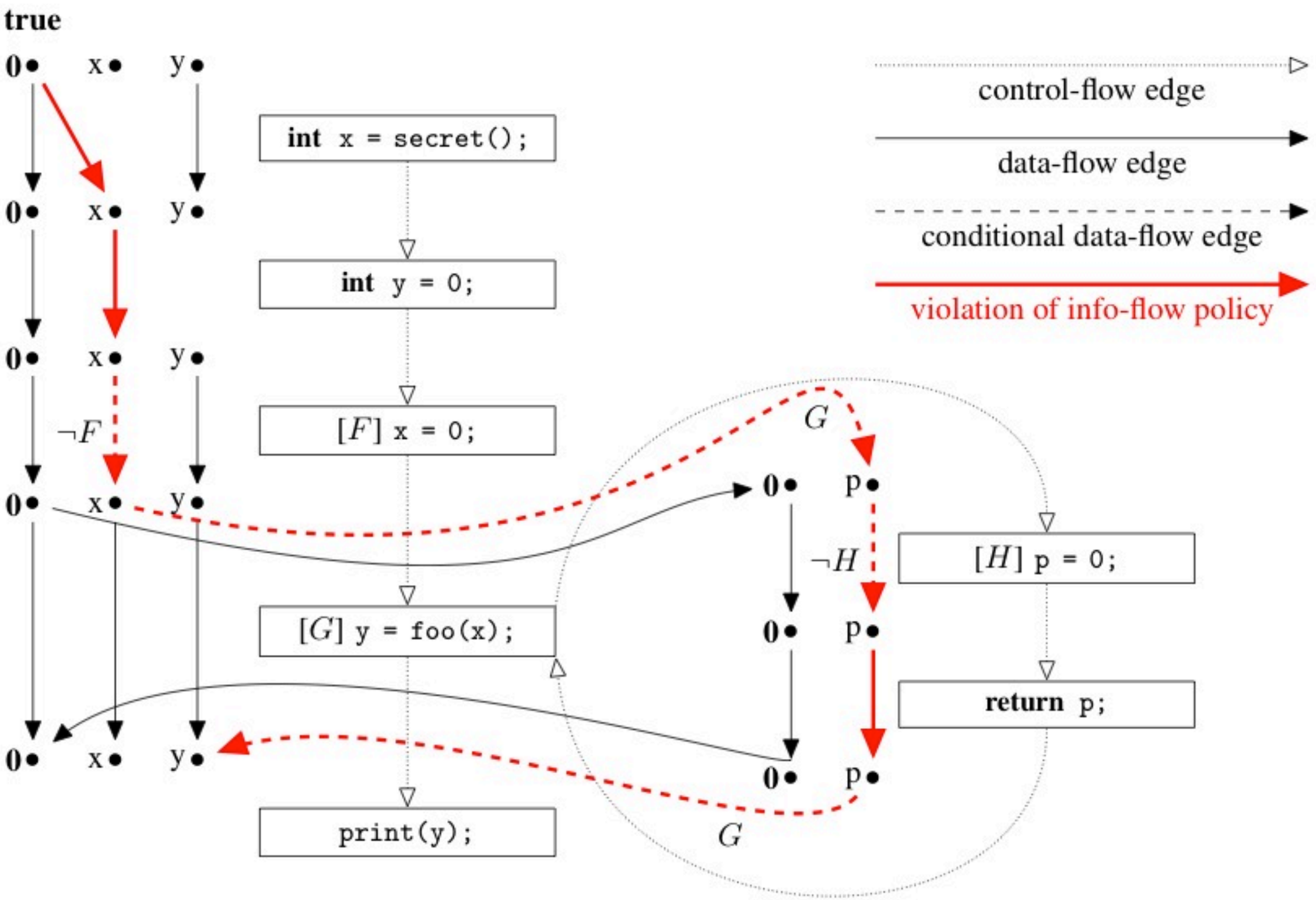
(b) Flow function for SPLs

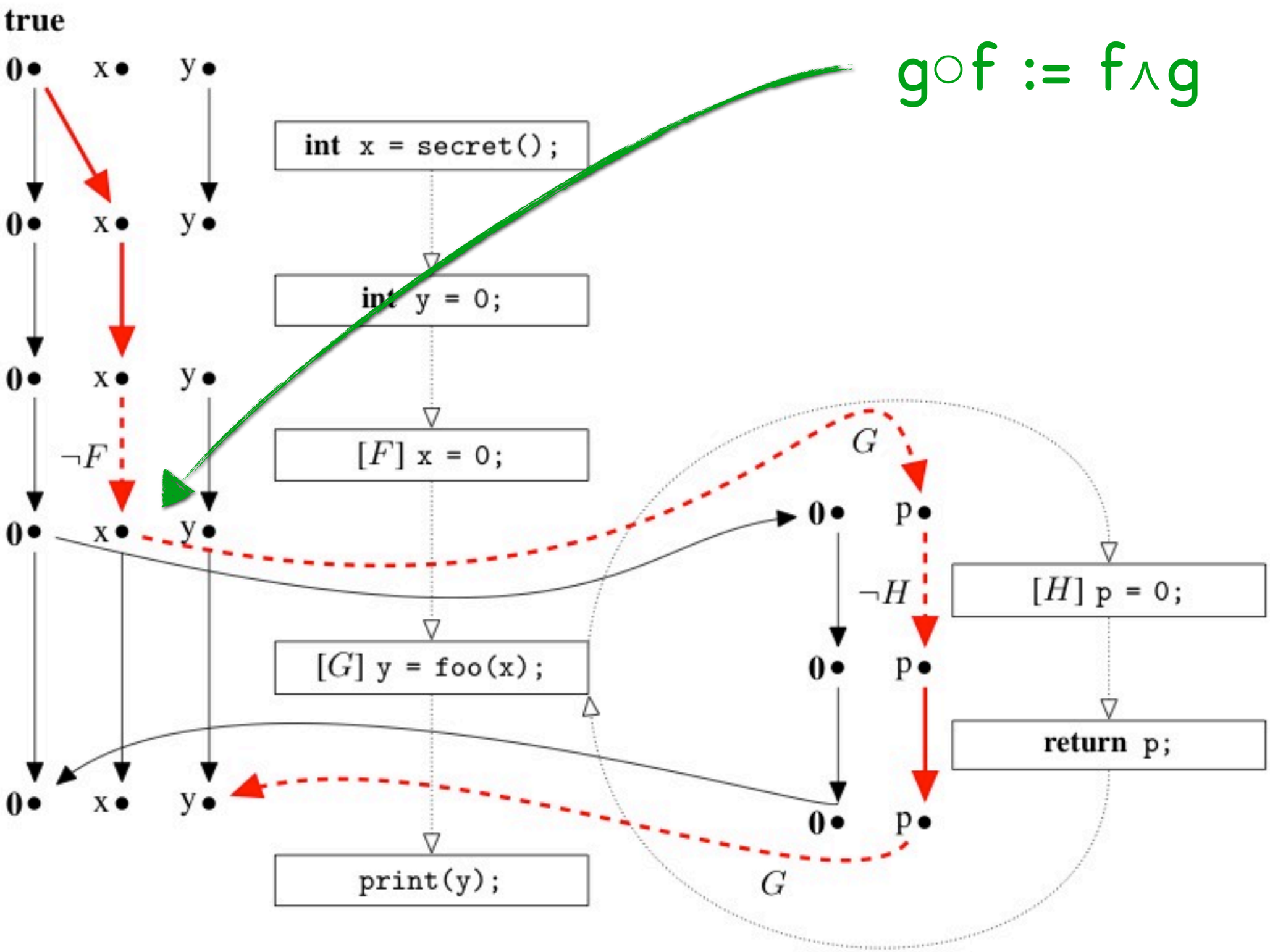
Stmt guarded by F...

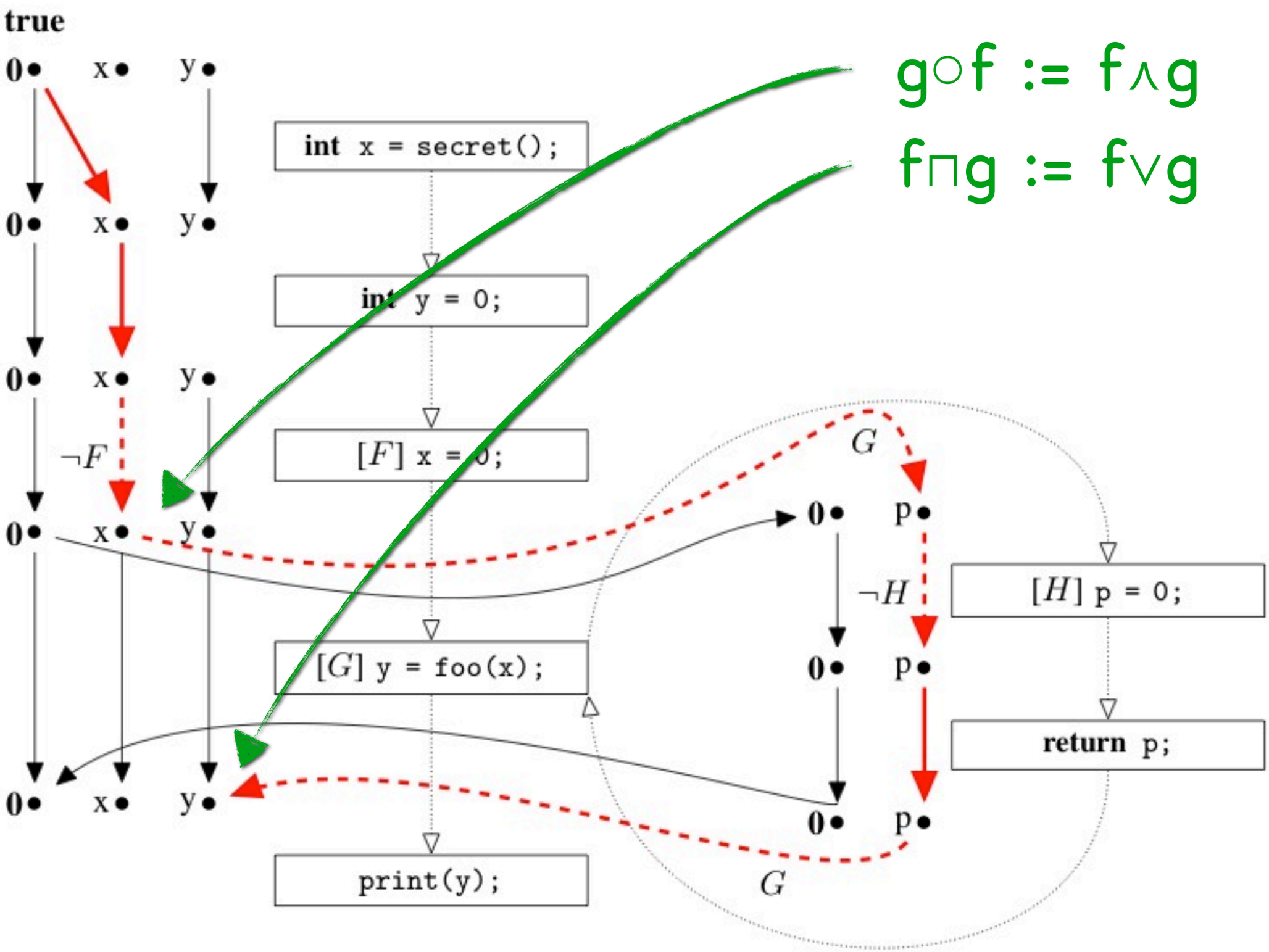


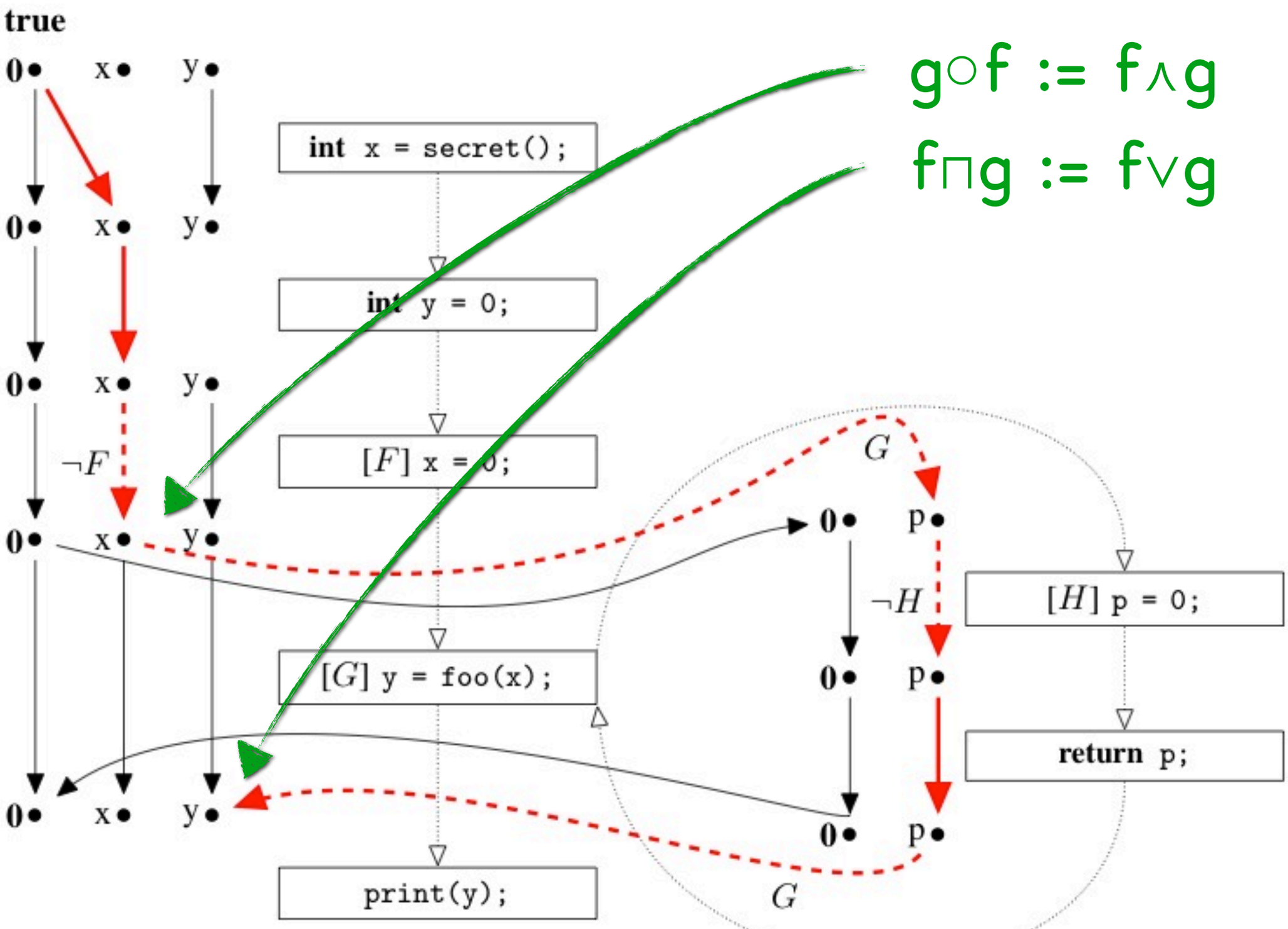
(a) Original flow function

(b) Flow function for SPLs


true
0
x
y
int x = secret();
int y = 0;
[F] x = 0;
¬F
[G] y = foo(x);
print(y);
control-flow edge
data-flow edge
conditional data-flow edge
violation of info-flow policy
G
p
¬H
[H] p = 0;
return p;

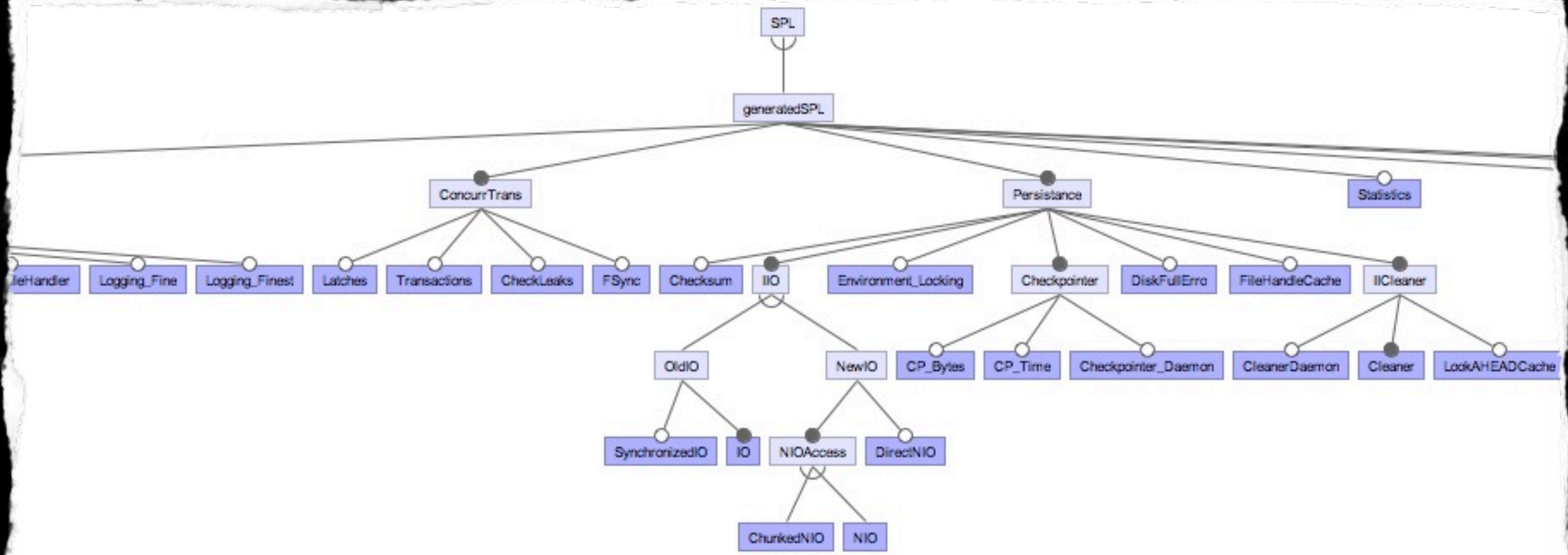

true
0 x y
int x = secret();
int y = 0;
[F] x = 0;
¬F
[G] y = foo(x);
print(y);
G
0 p
¬H
[H] p = 0;
return p;
g∘f := f∧g


true
0
x
y
int x = secret();
int y = 0;
[F] x = 0;
¬F
[G] y = foo(x);
print(y);
G
p
¬H
[H] p = 0;
return p;
g∘f := f∧g
f⊓g := f∨g



$$(\textbf{true} \wedge \neg F \wedge G \wedge \neg H \wedge G) \vee \textbf{false} = \neg F \wedge G \wedge \neg H$$

Exploit Feature Model



Evictor ∨ EvictorDaemon ∨ LookAHEADCache ⇒ Memory_Budget

Critical_Eviction ⇒ INCompressor

CP_Bytes ⇒ CP_Time

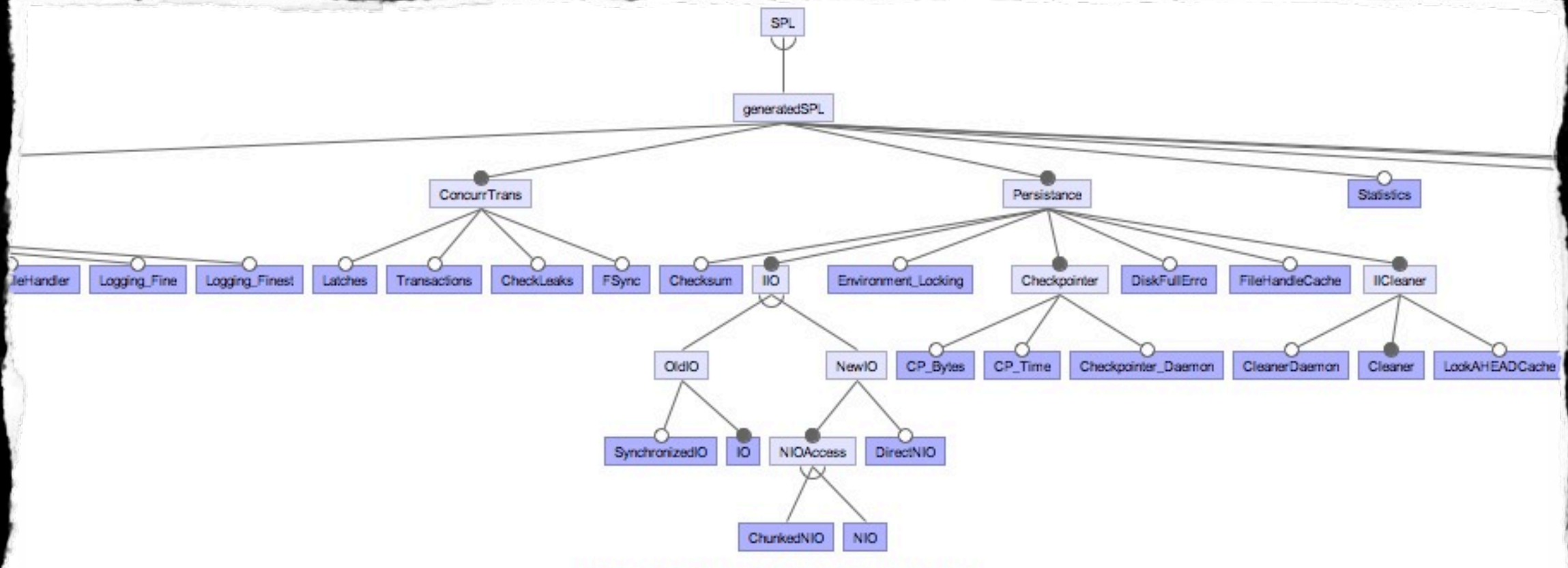
DeleteOp ⇒ Evictor ∧ INCompressor ∧ Memory_Budget

Memory_Budget ⇒ Evictor ∧ Latches

TruncateOp ⇒ DeleteOp

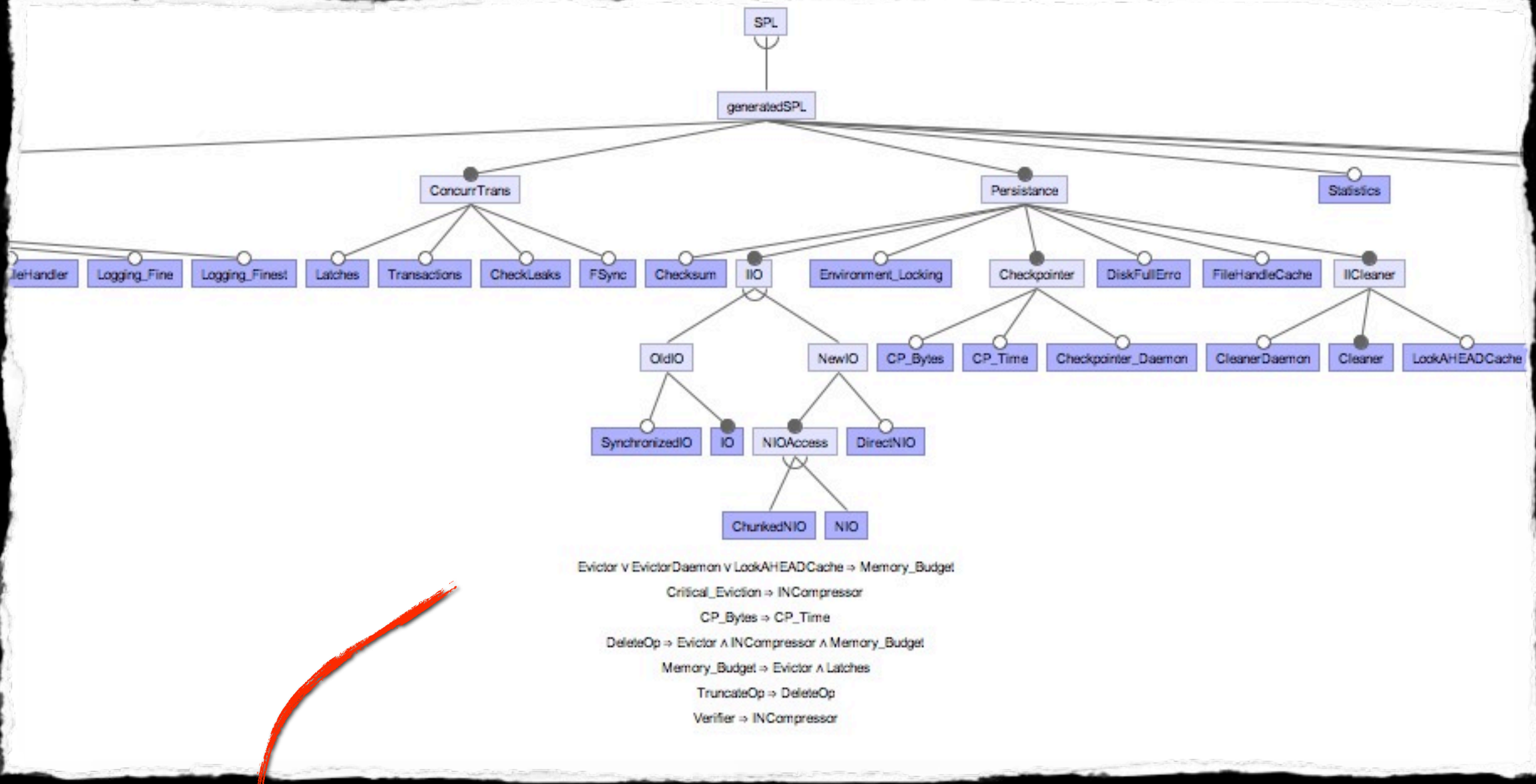
Verifier ⇒ INCompressor

Exploit Feature Model



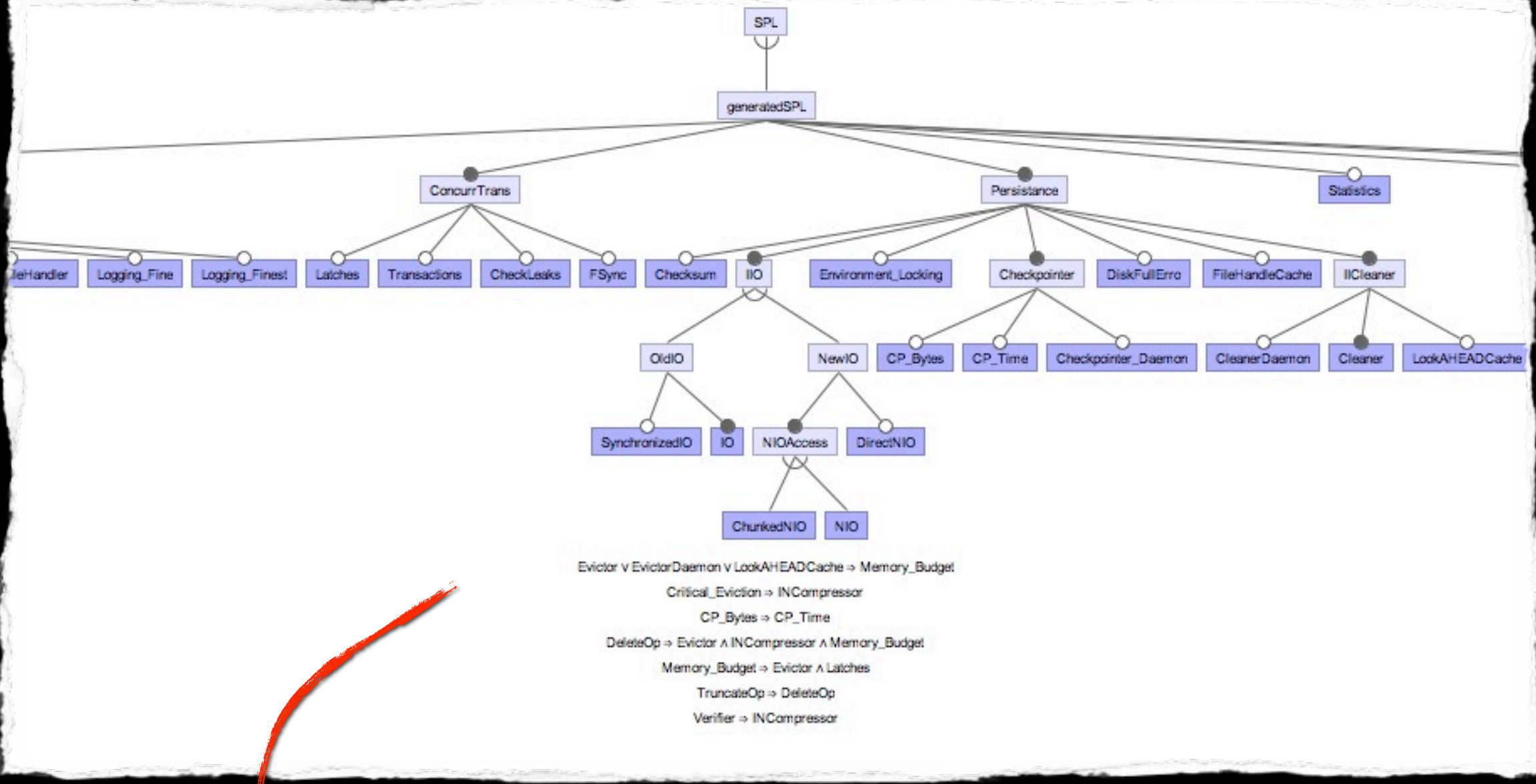
F ⟷ G

Exploit Feature Model

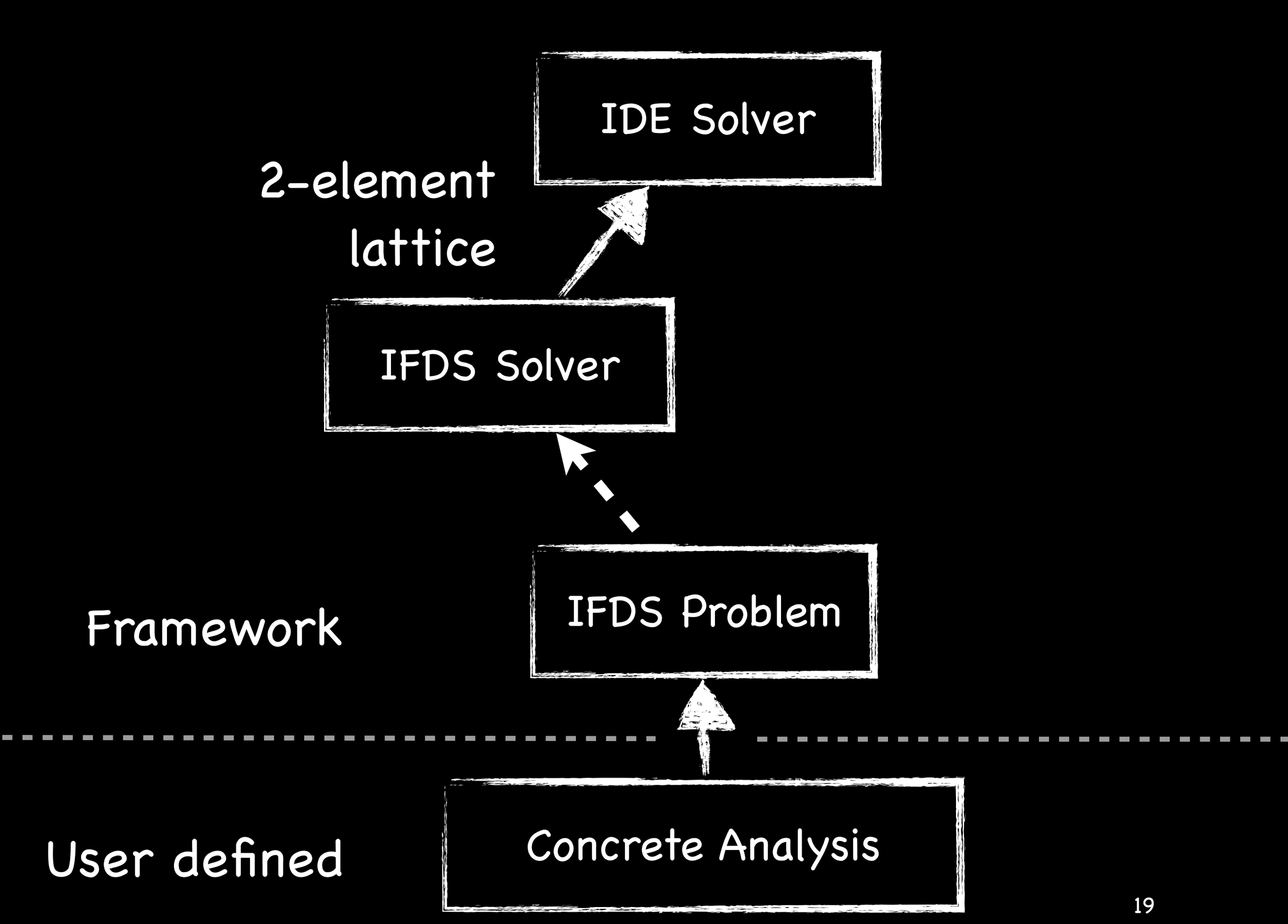


$F \leftrightarrow G$ $\neg F \wedge G \wedge \neg H$

Exploit Feature Model



$$(F \leftrightarrow G) \wedge (\neg F \wedge G \wedge \neg H) == \text{false}$$


IDE Solver
2-element lattice
IFDS Solver
Framework
IFDS Problem
User defined
Concrete Analysis

IDE Solver

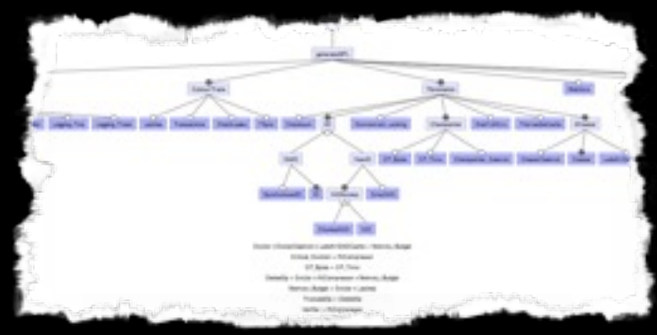
2-element lattice

constraint lattice

IFDS Solver

SPL/IFDS Solver

IFDS Problem

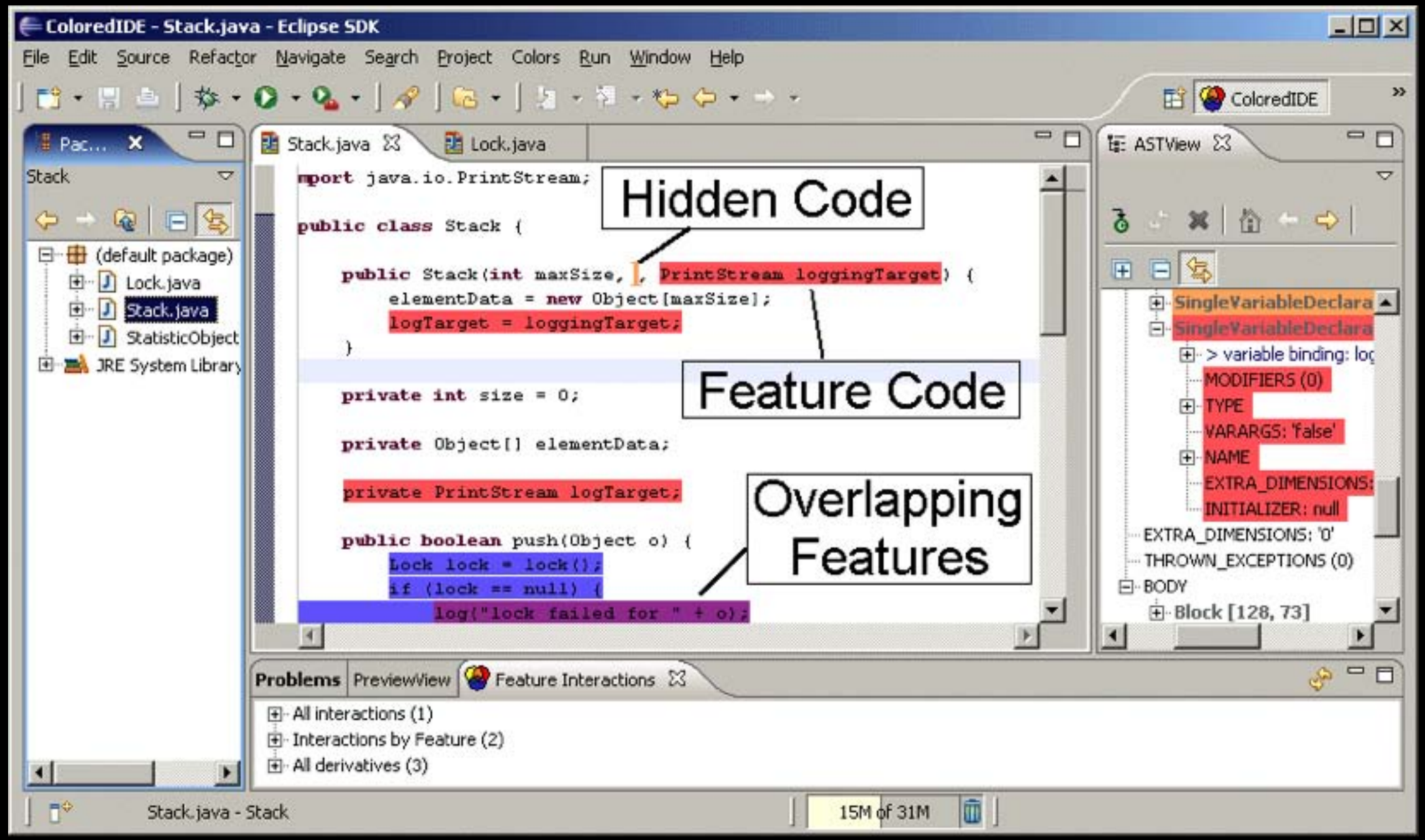


Framework

User defined

Concrete Analysis

We use CIDE


ColoredIDE - Stack.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Colors Run Window Help
ColoredIDE
Pac...
Stack
(default package)
Lock.java
Stack.java
StatisticObject
JRE System Library
Stack.java
Lock.java
import java.io.PrintStream;
public class Stack {
public Stack(int maxSize, PrintStream loggingTarget) {
elementData = new Object[maxSize];
logTarget = loggingTarget;
}
private int size = 0;
private Object[] elementData;
private PrintStream logTarget;
public boolean push(Object o) {
Lock lock = lock();
if (lock == null) {
log("lock failed for " + o);
Hidden Code
Feature Code
Overlapping Features
ASTView
SingleVariableDeclara
SingleVariableDeclara
> variable binding: log
MODIFIERS (0)
TYPE
VARARGS: 'false'
NAME
EXTRA_DIMENSIONS:
INITIALIZER: null
EXTRA_DIMENSIONS: '0'
THROWN_EXCEPTIONS (0)
BODY
Block [128, 73]
Problems
PreviewView
Feature Interactions
All interactions (1)
Interactions by Feature (2)
All derivatives (3)
Stack.java - Stack
15M of 31M

CIDE: pro/con

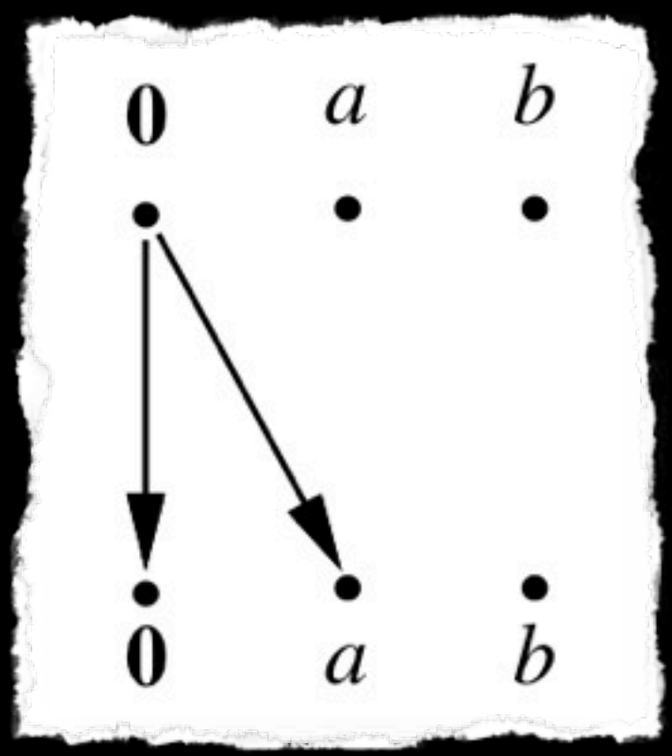
- Advantages:
  - Product line itself is a compileable Java program!
  - Easy to use, decent IDE support
- Limitation:
  - Limited to well-behaved feature annotations (can only color entire statements)
  - In the future want to look at C

Open questions

- How do apply approach to SPLs that are less disciplined (e.g. #ifdef in C)
- How to propagate constraints efficiently?
  - Probably using BDDs
  - Maybe make use of analysis-specific properties...

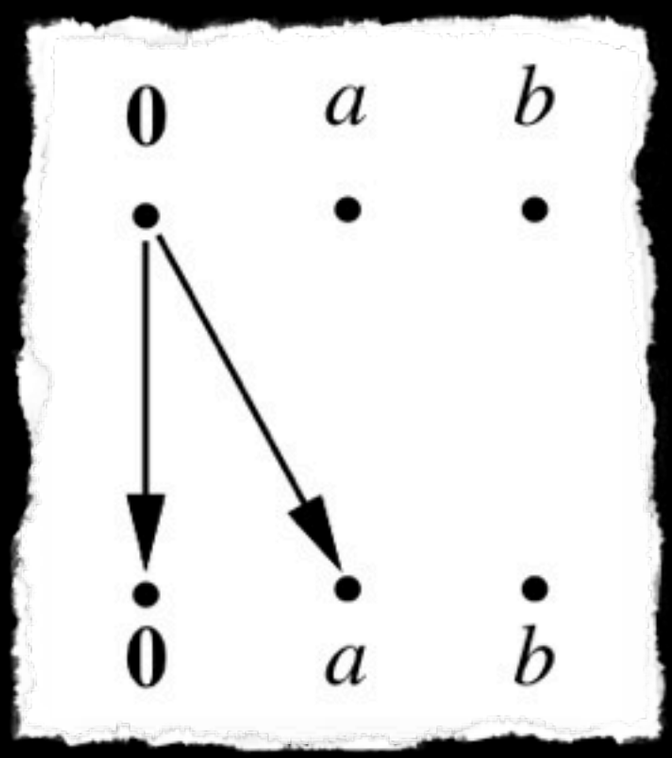
Locally separable functions

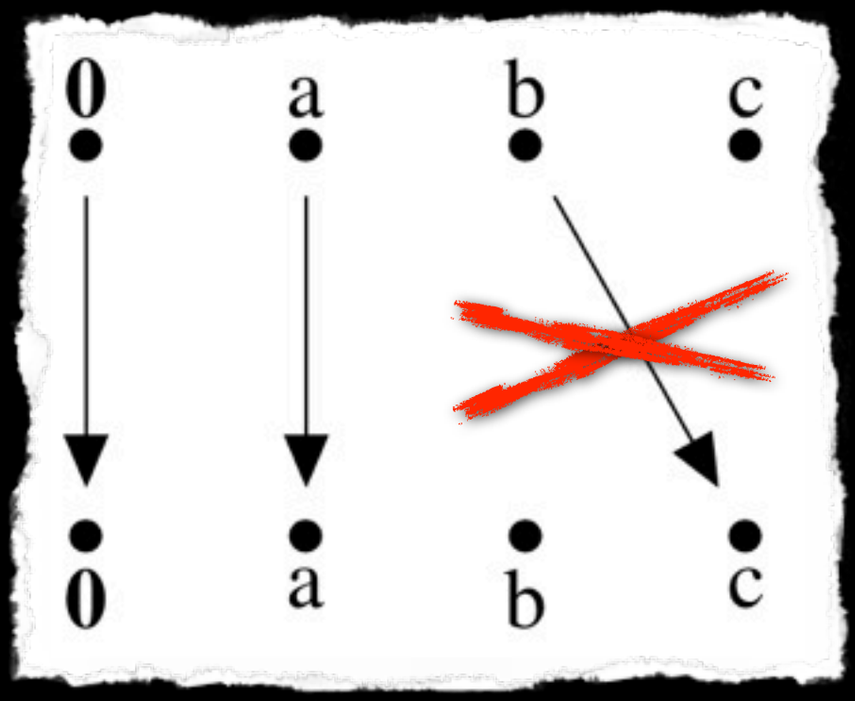
flow function locally separable ≡ function does not depend on input values



Locally separable functions

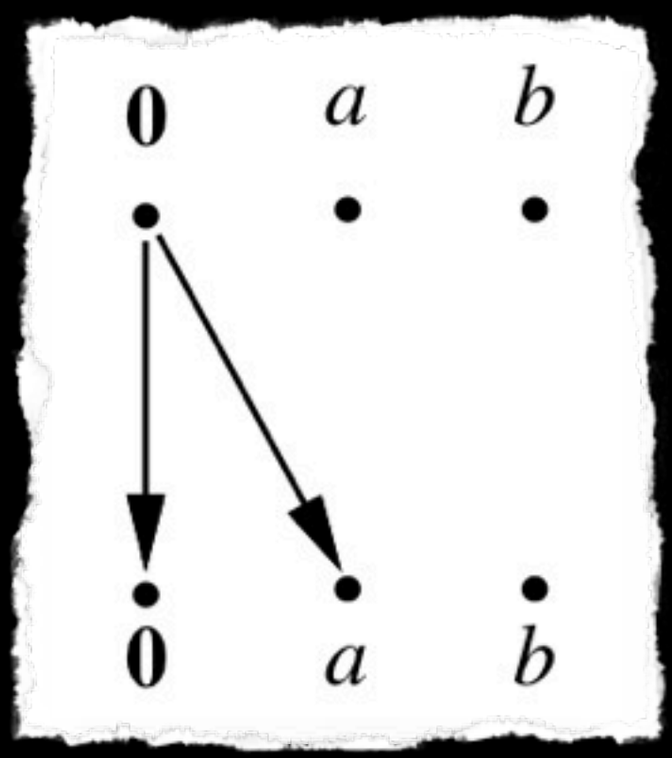
flow function locally separable ≡ function does not depend on input values

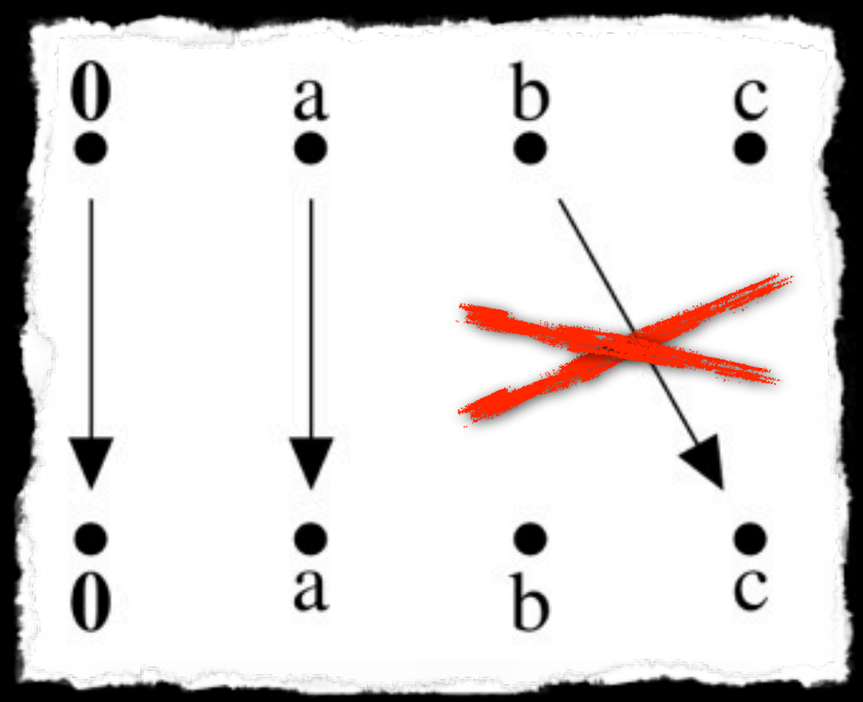




Locally separable functions

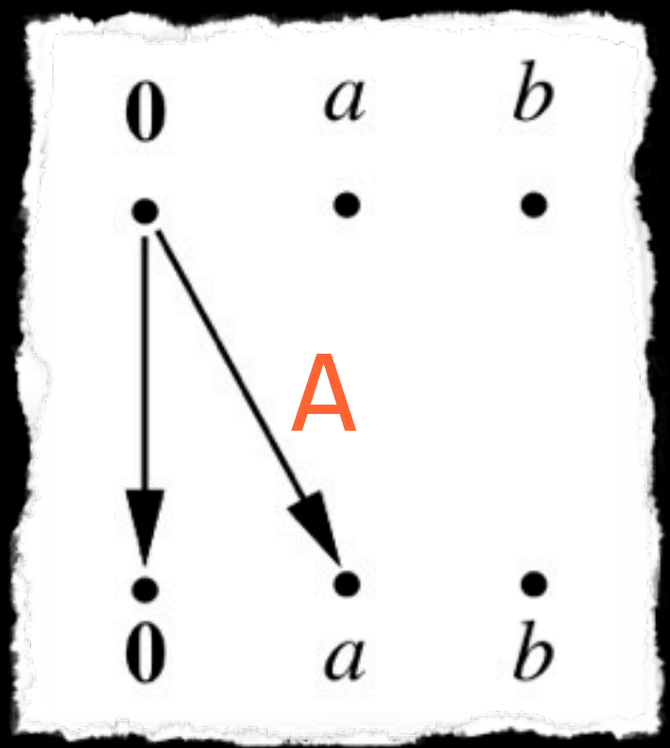
flow function locally separable ≡ function does not depend on input values





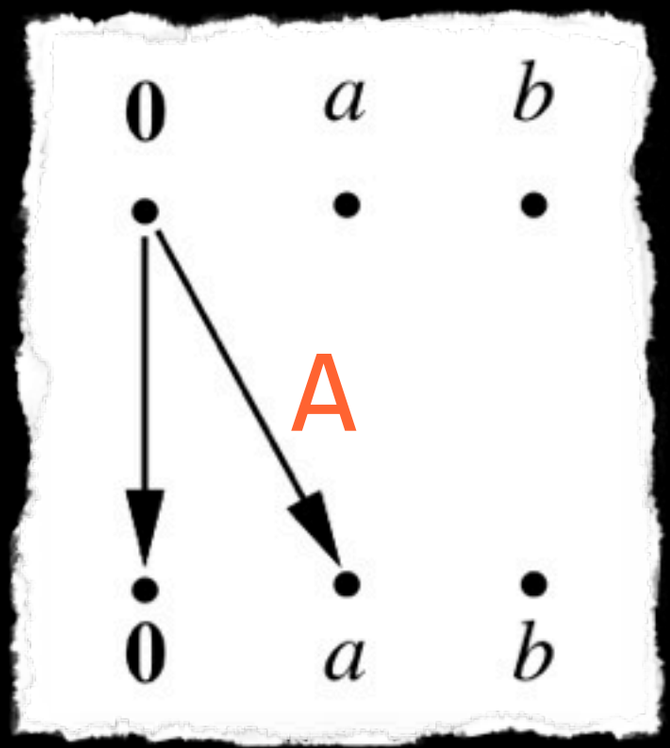
Reaching Definitions, Live Variables …

Yields very special constraints...



A ∧ ¬B ∧ ¬C...

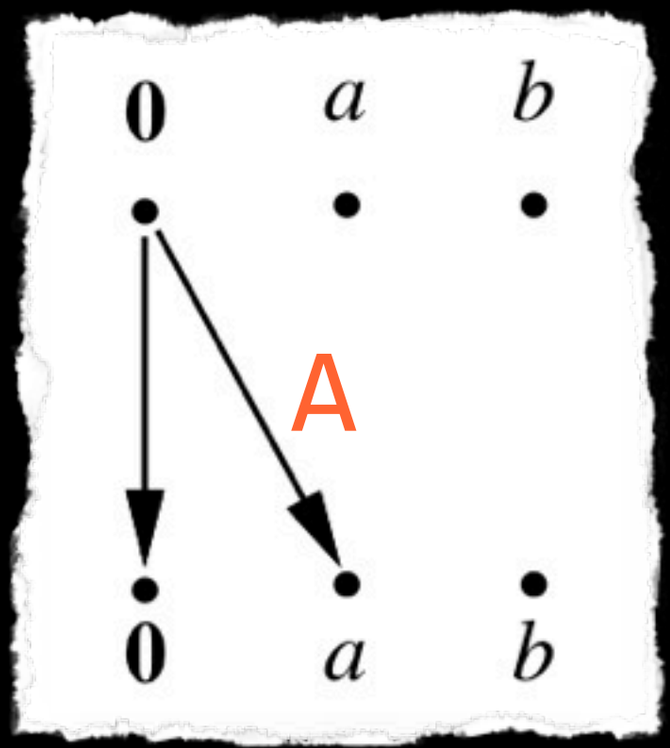
Yields very special constraints...



A ∧ ¬B ∧ ¬C...

Will **not** get: A ∧ B

Yields very special constraints...



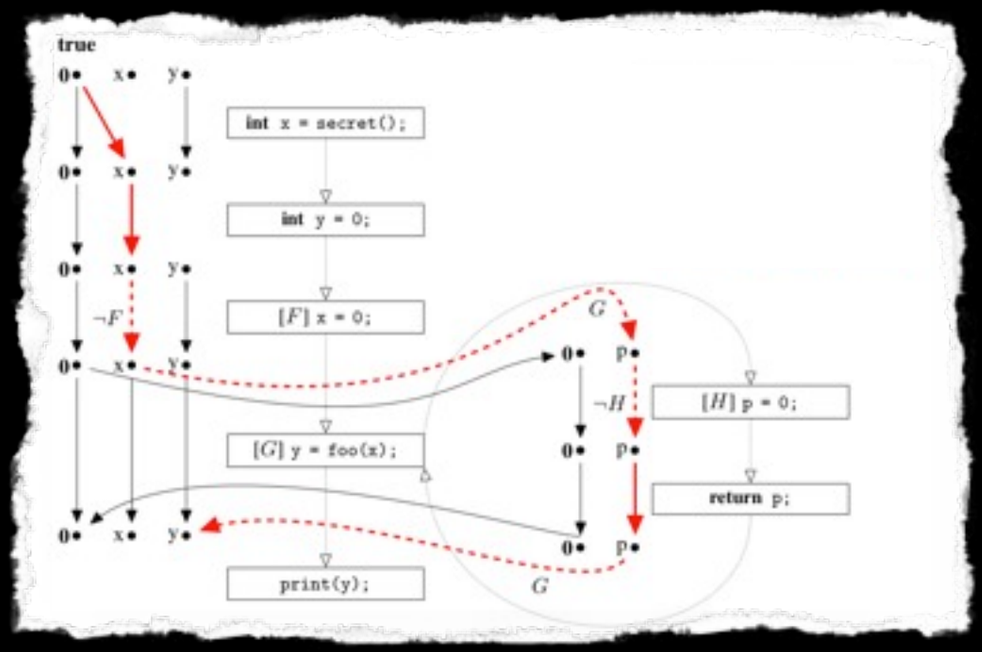
A ∧ ¬B ∧ ¬C...

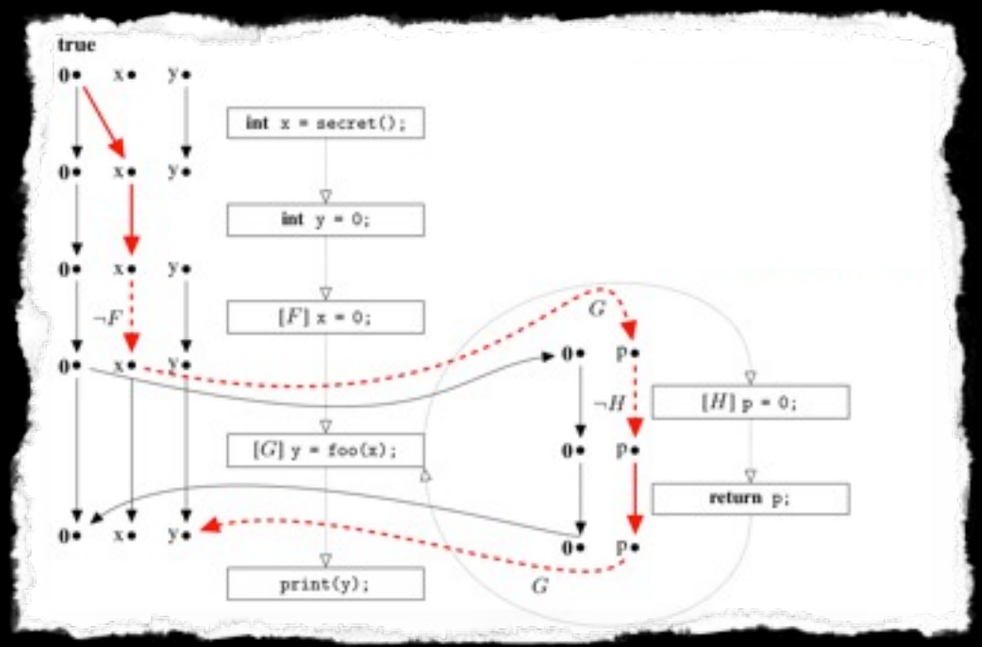
Will **not** get: A ∧ B

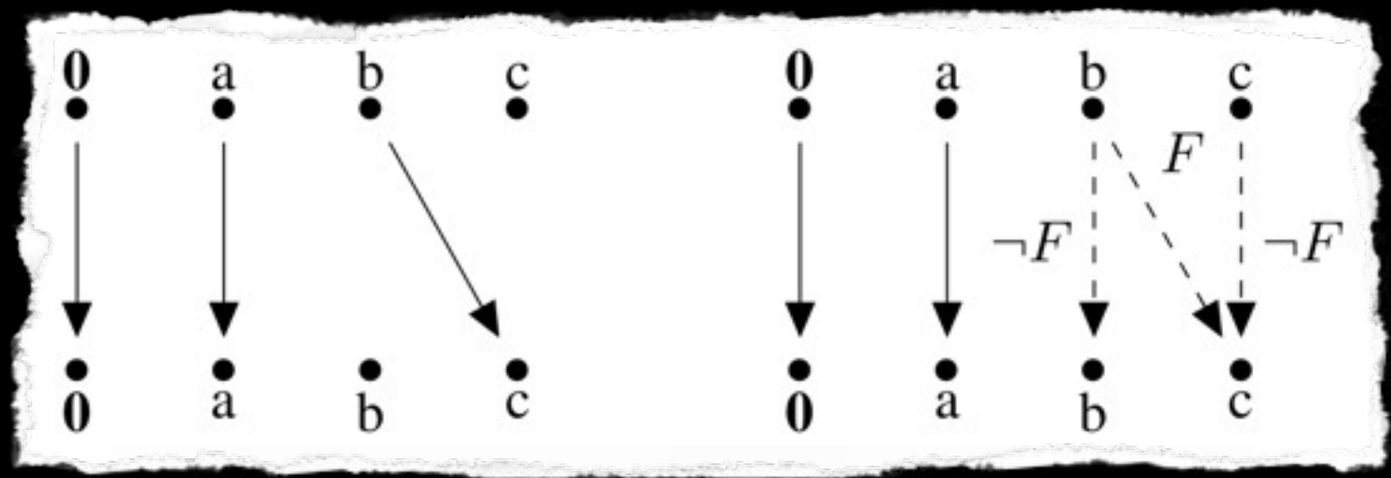
But those a feature model could help eliminate best!

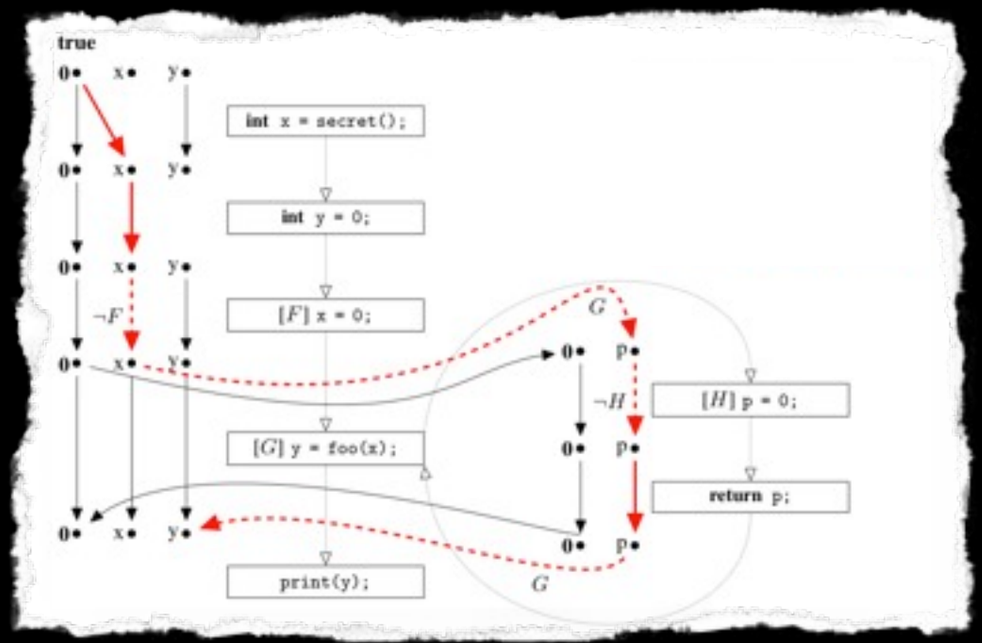
Thanks to...

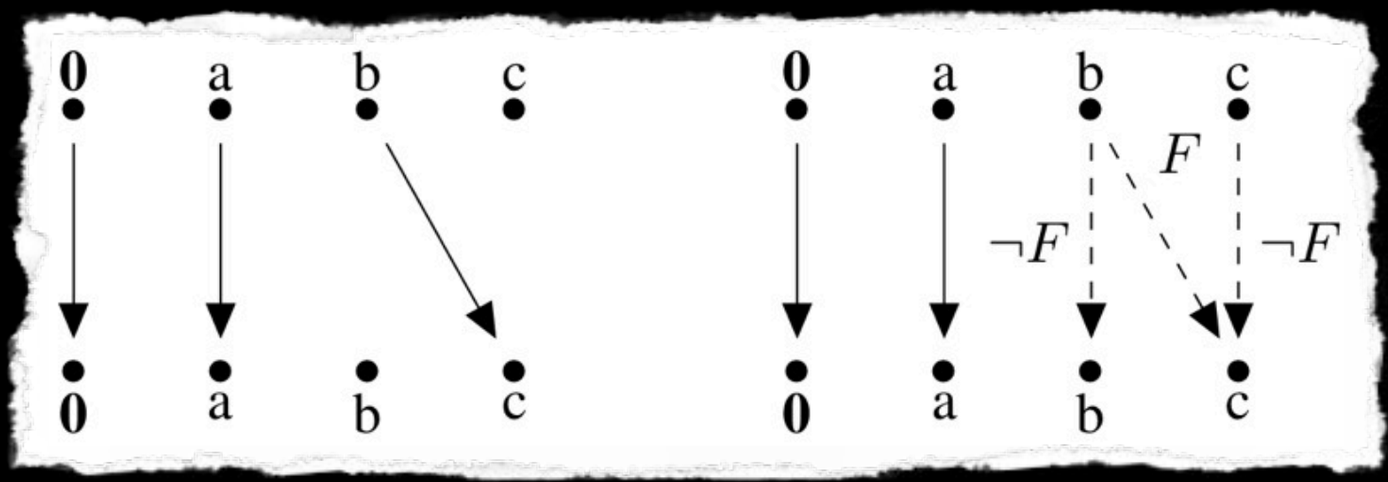
- Claus Braband (ITU Copenhagen)
- Marcio Ribeiro (UFPE Brazil)
- Tarsis Toledo (UFPE Brazil)
- Paulo Borba (UFPE Brazil)
- CIDE developers

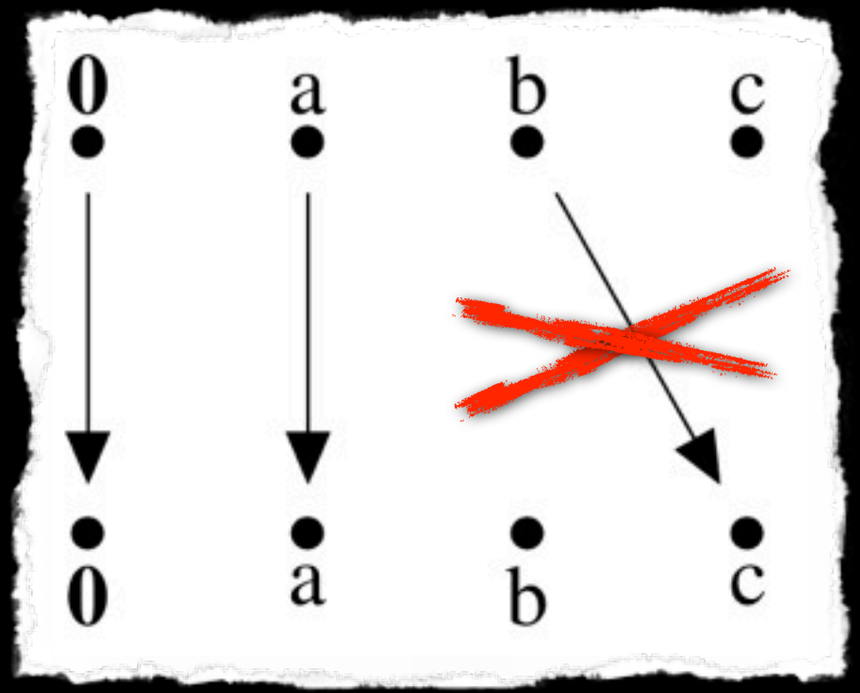

true
0• x• y•
int x = secret();
int y = 0;
[F] x = 0;
¬F
G
[G] y = foo(x);
print(y);
p•
¬H
[H] p = 0;
return p;

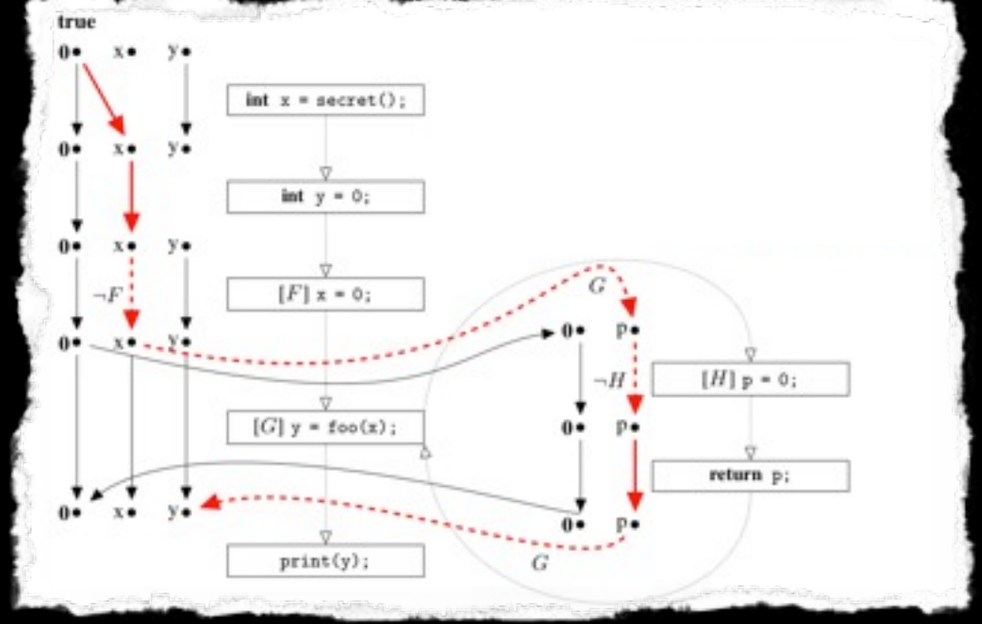

true
int x = secret();
int y = 0;
[F] x = 0;
[G] y = foo(x);
print(y);
[H] p = 0;
return p;
¬F
G
¬H

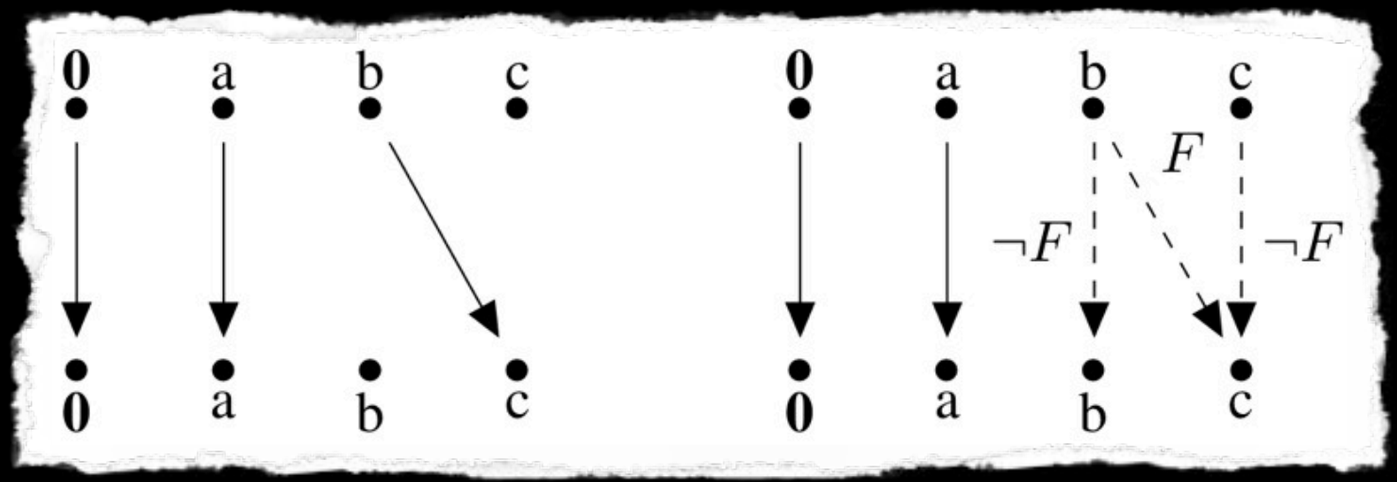

0
a
b
c
F
¬F
¬F

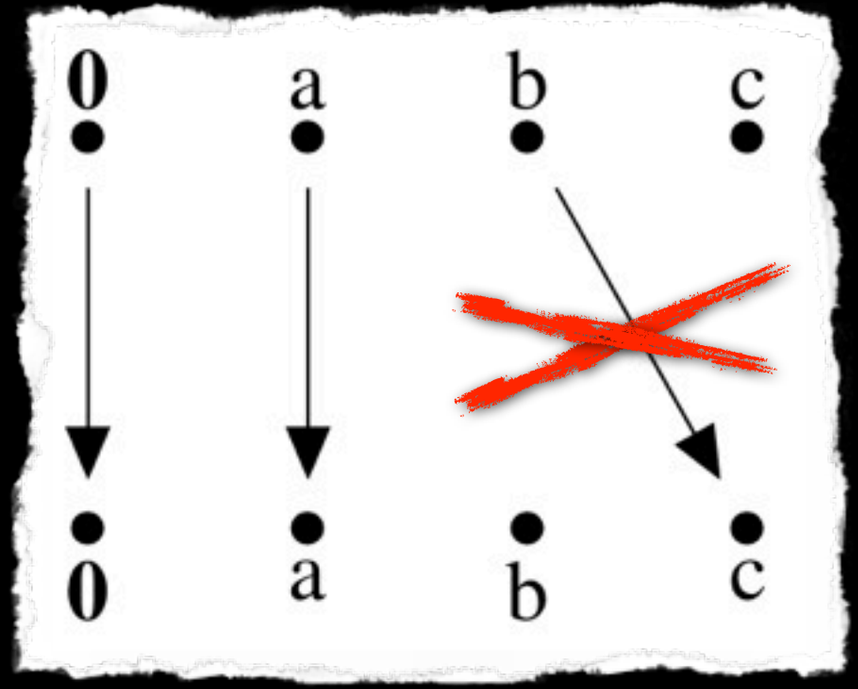

true
0 x y
int x = secret();
int y = 0;
[F] x = 0;
¬F
G
p
¬H
[H] p = 0;
[G] y = foo(x);
return p;
print(y);


0
a
b
c
F
¬F


0
a
b
c







http://bodden.de/spl/

http://sse.ec-spride.de/

SECURE SOFTWARE ENGINEERING GROUP

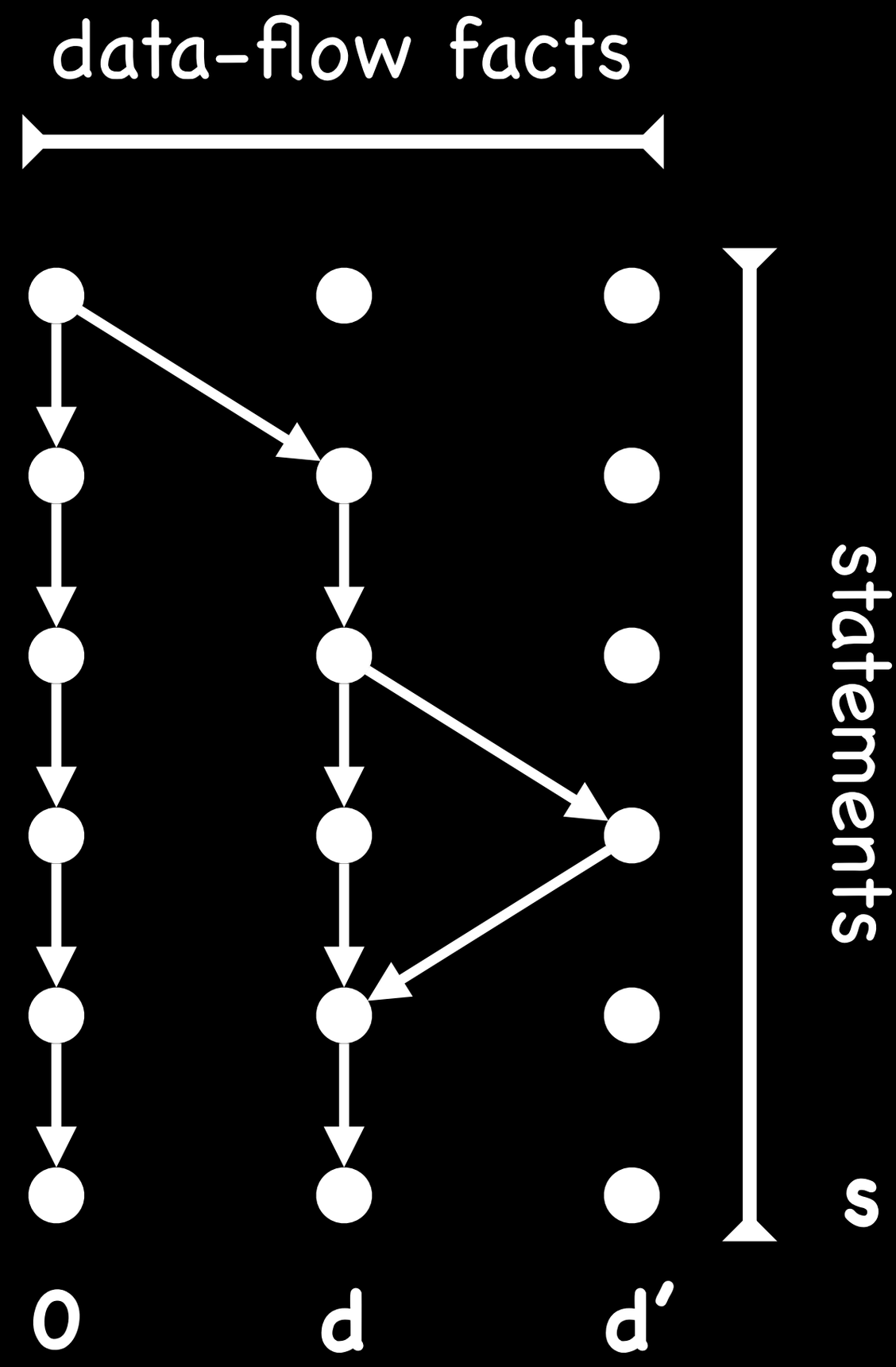
EC SPRIDE
EUROPEAN CENTER FOR SECURITY AND PRIVACY BY DESIGN

TECHNISCHE UNIVERSITÄT DARMSTADT

RHS 95

Static
Super-graph edge

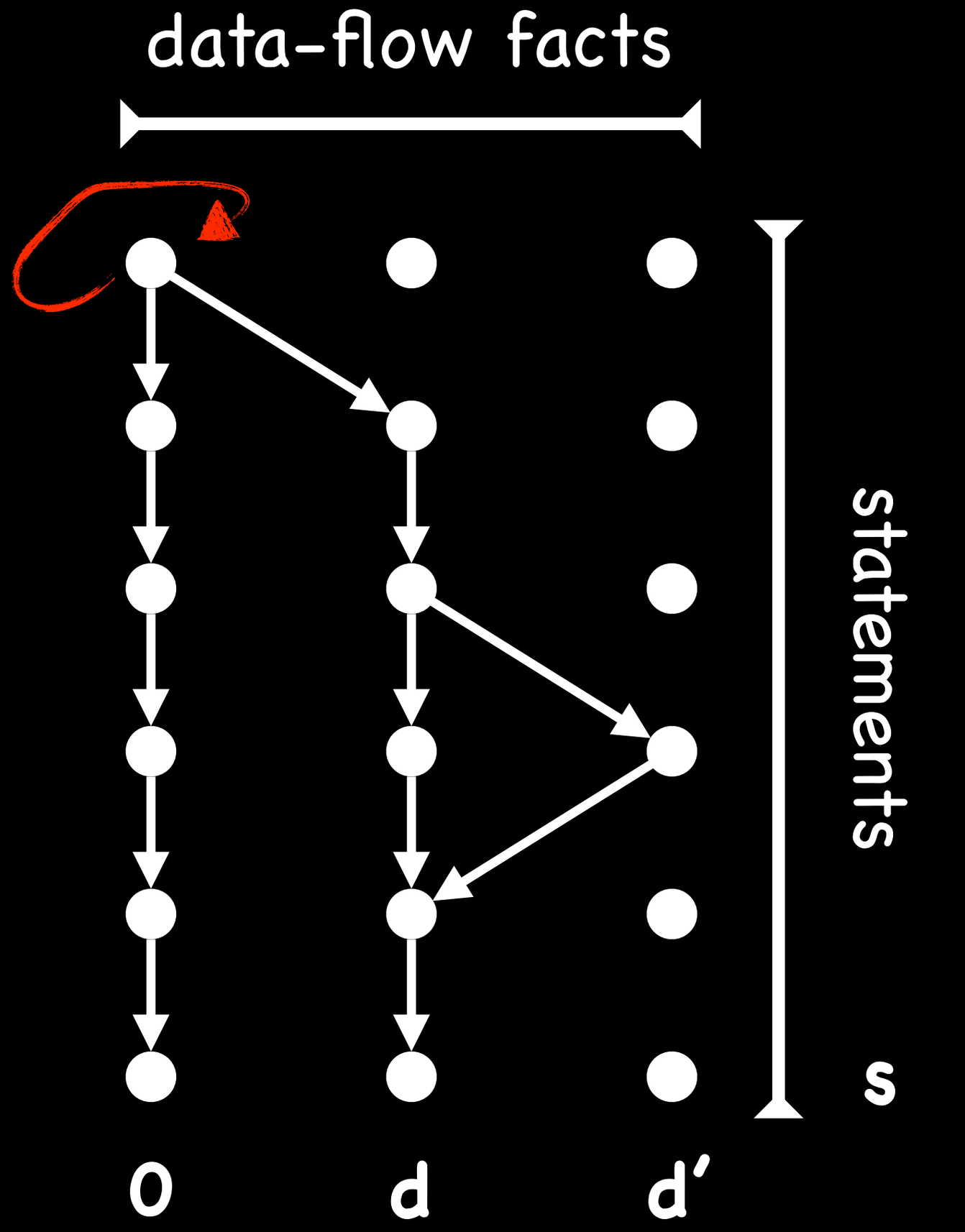
Computed
Summary Edge



RHS 95

Static
Super-graph edge

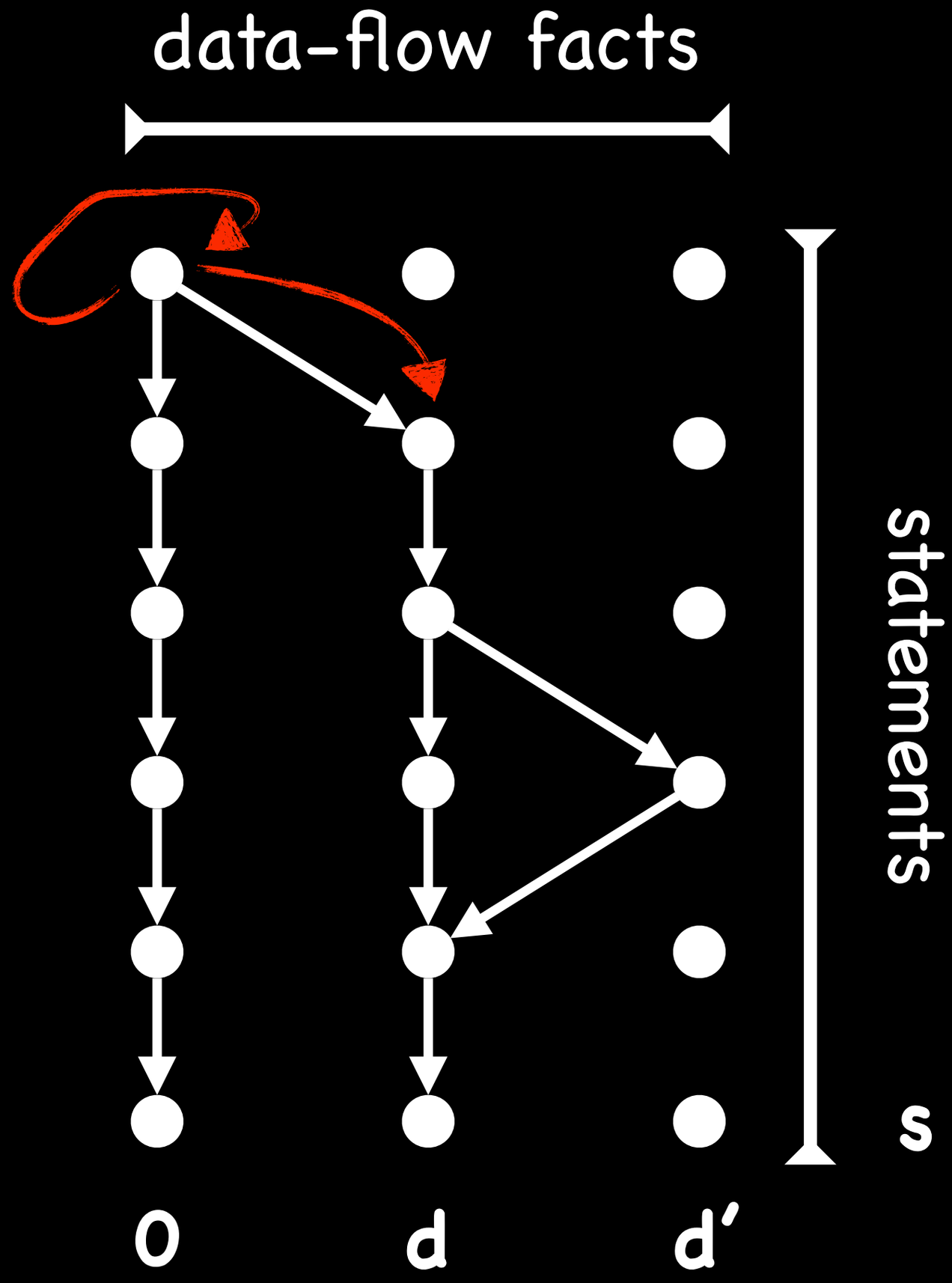
Computed
Summary Edge



RHS 95

Static
Super-graph edge

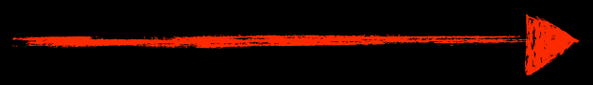
Computed
Summary Edge

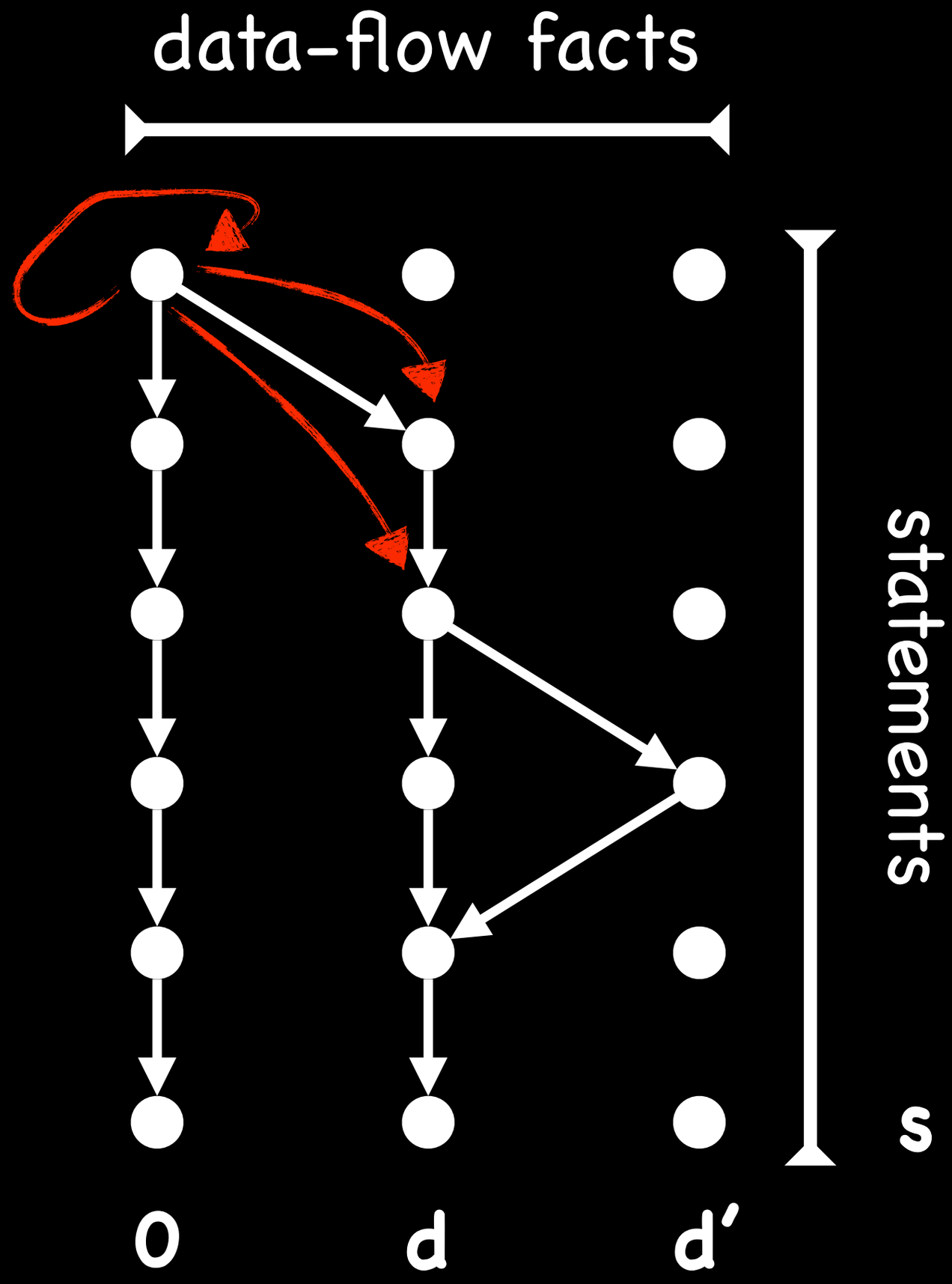


RHS 95

Static
Super-graph edge

Computed
Summary Edge

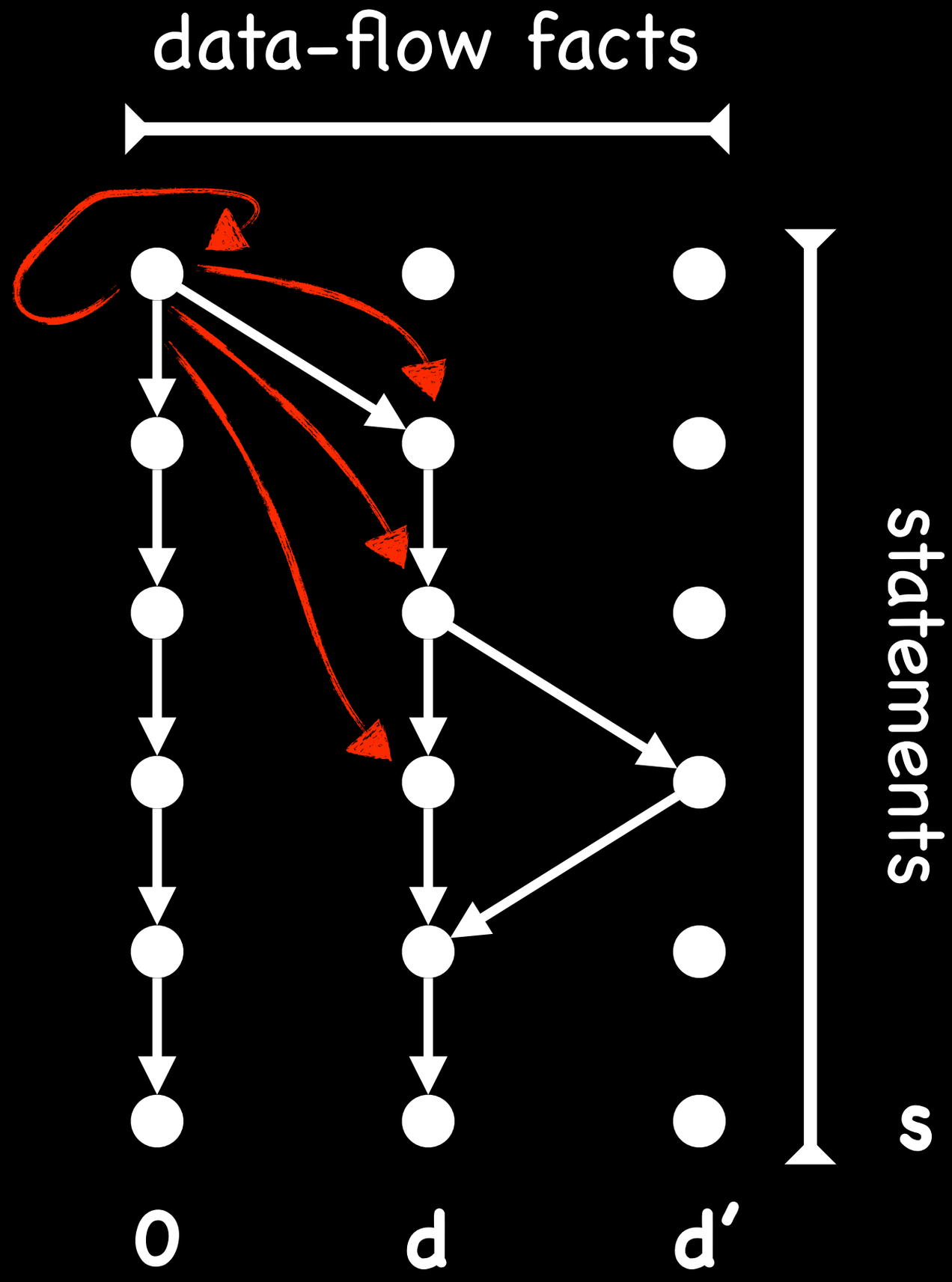




RHS 95

Static
Super-graph edge

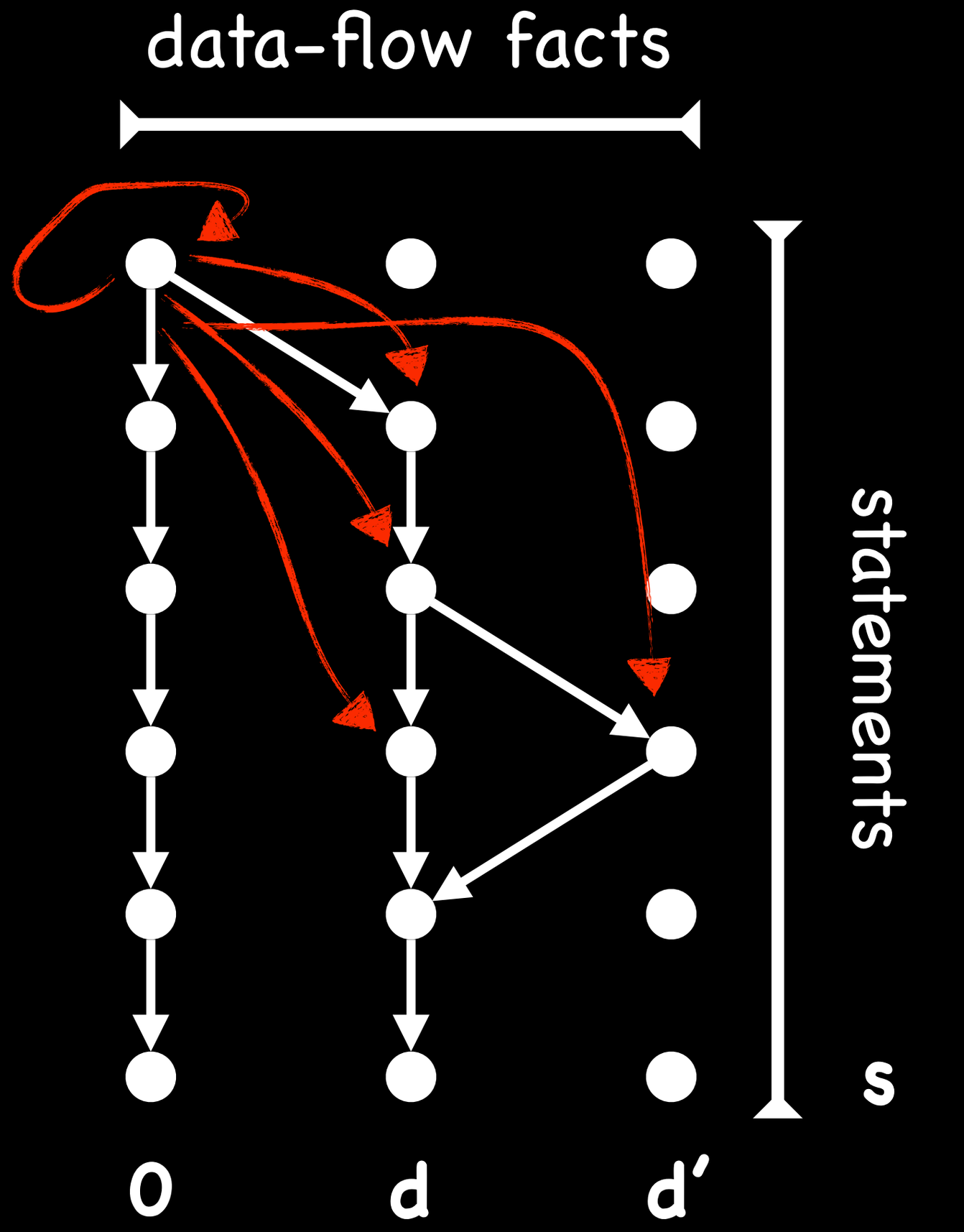
Computed
Summary Edge



RHS 95

Static
Super-graph edge

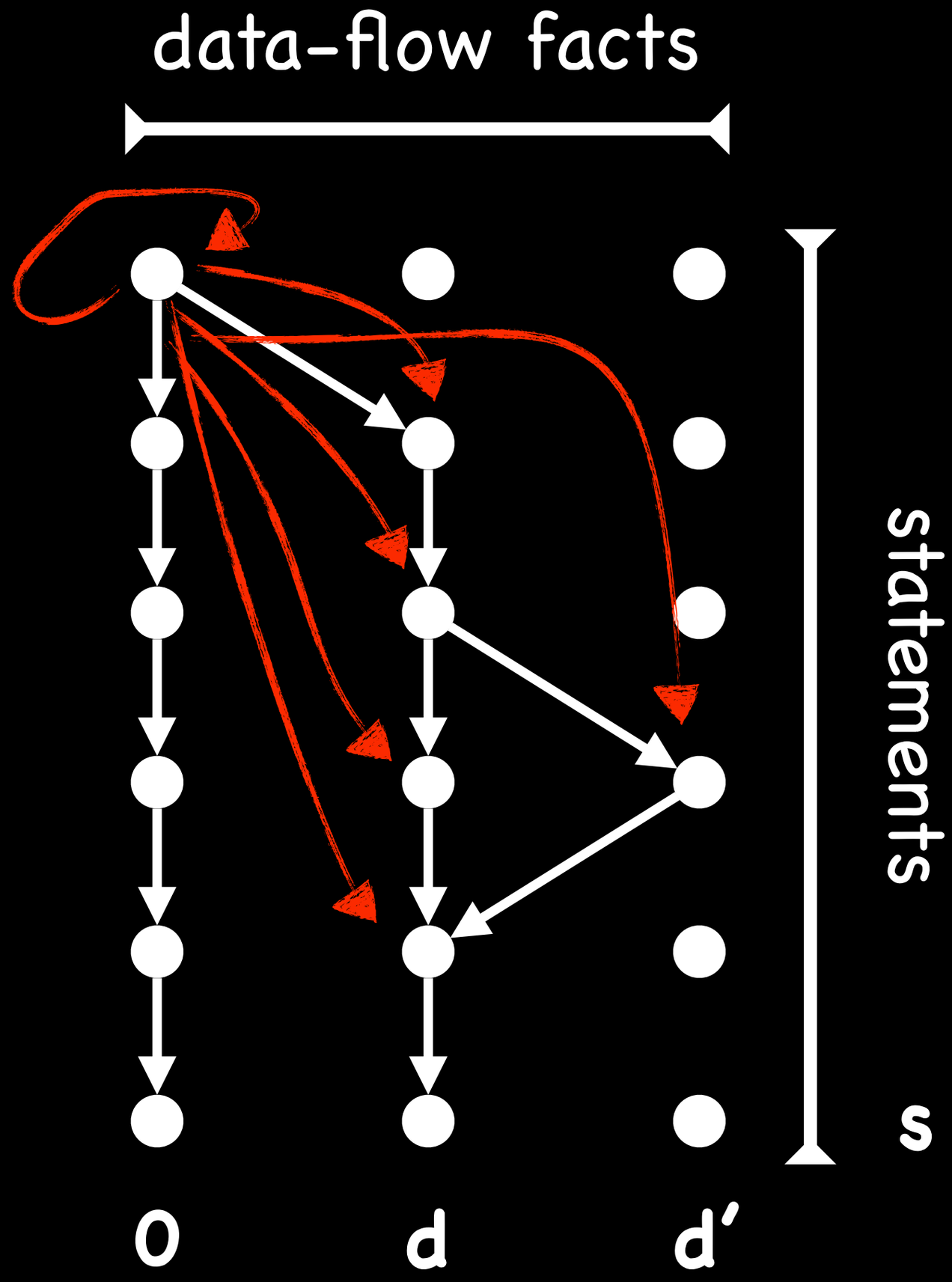
Computed
Summary Edge



RHS 95

Static
Super-graph edge

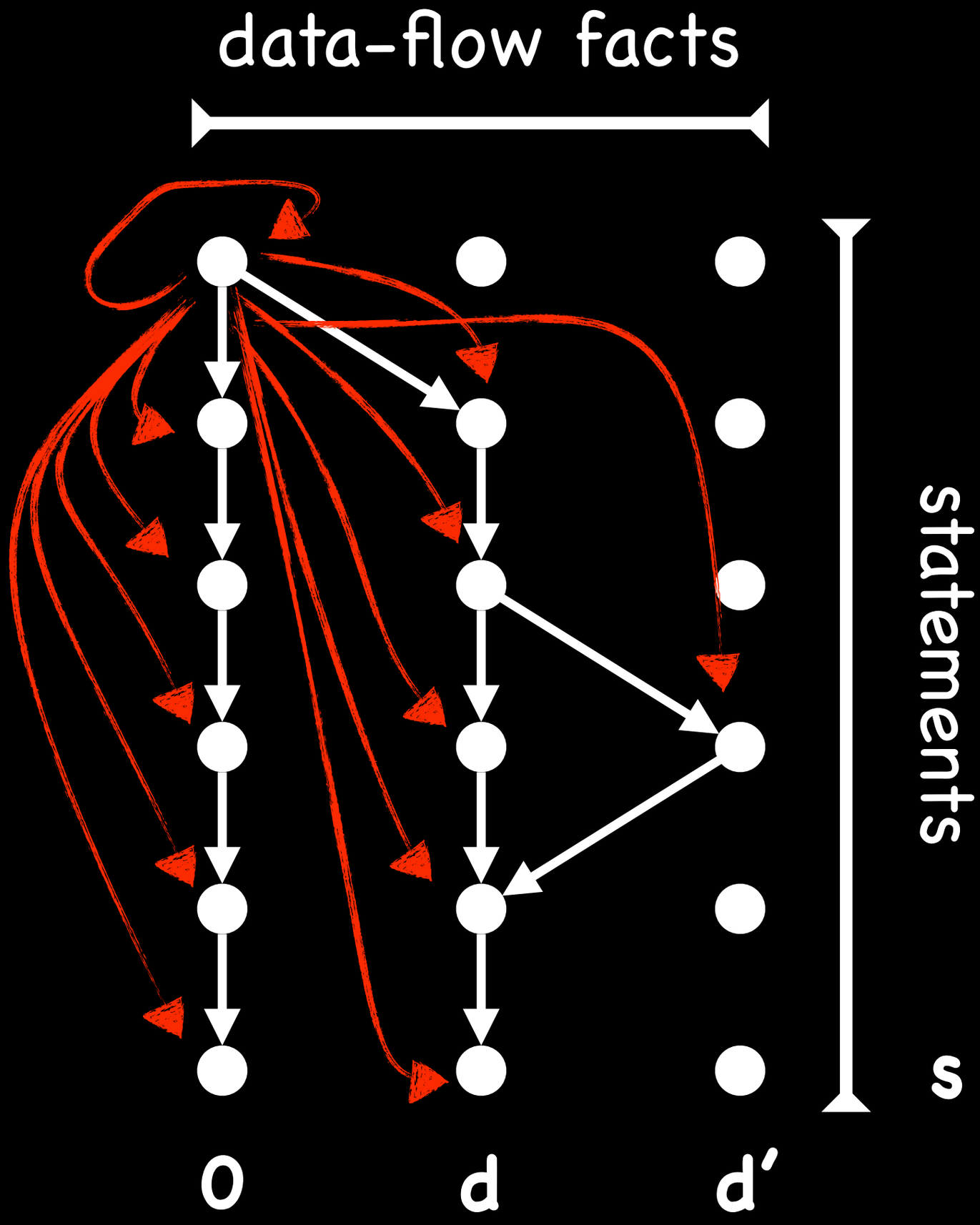
Computed
Summary Edge



RHS 95

Static
Super-graph edge

Computed
Summary Edge

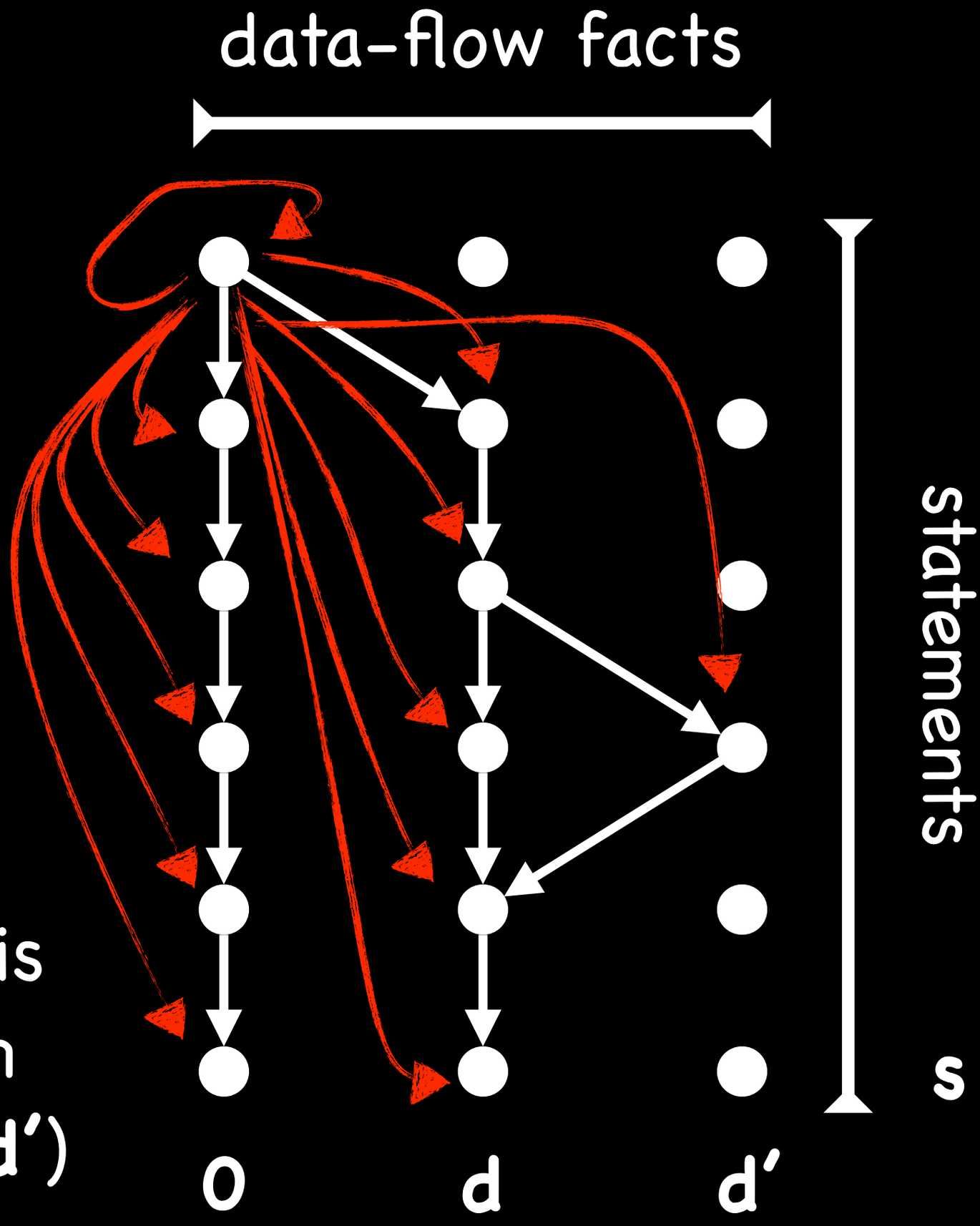


RHS 95

Static
Super-graph edge

Computed
Summary Edge

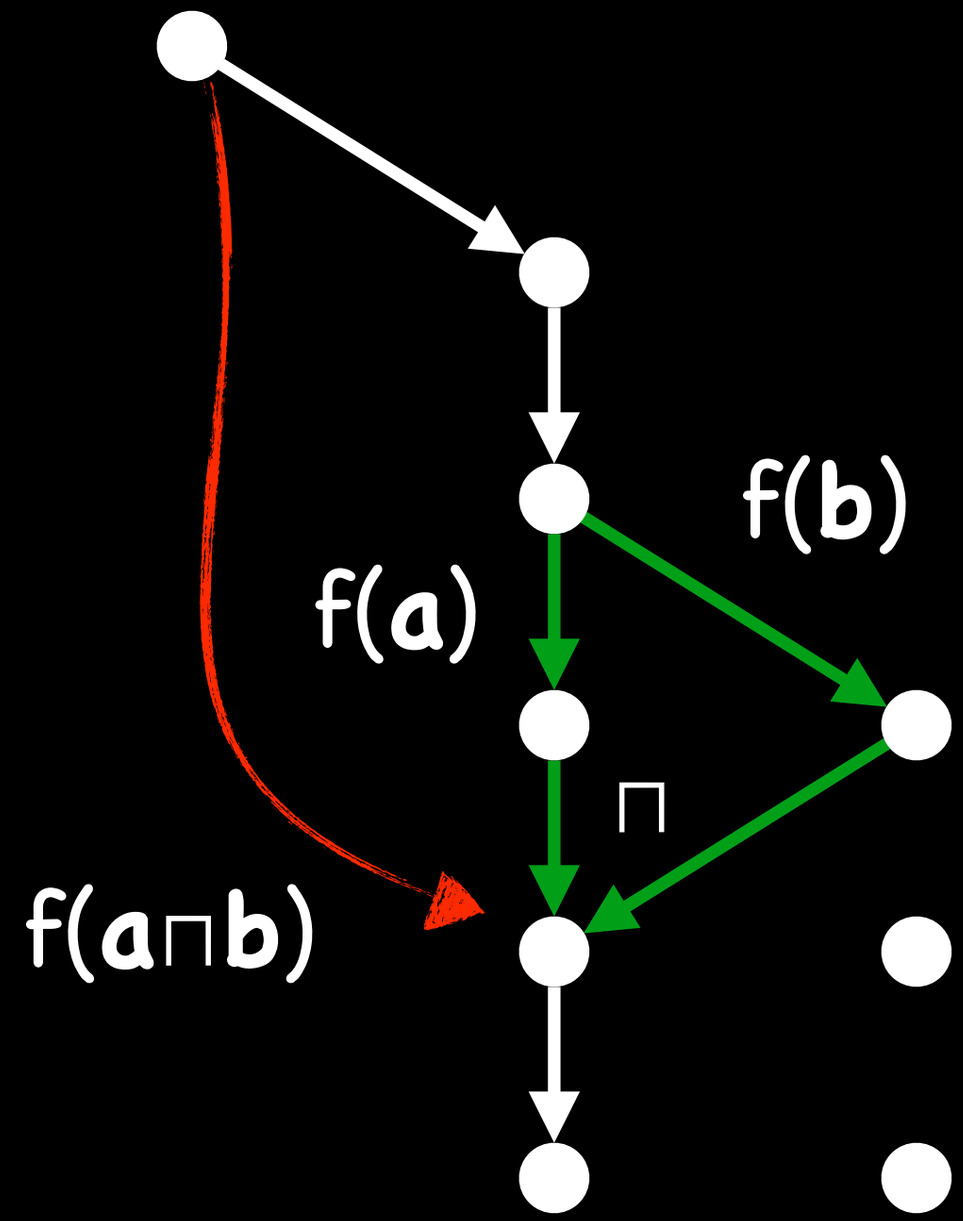
d holds at **s** if there is a summary edge from startOf(methodOf(**s**),**d′**) to (**s**,**d**) for some **d′**



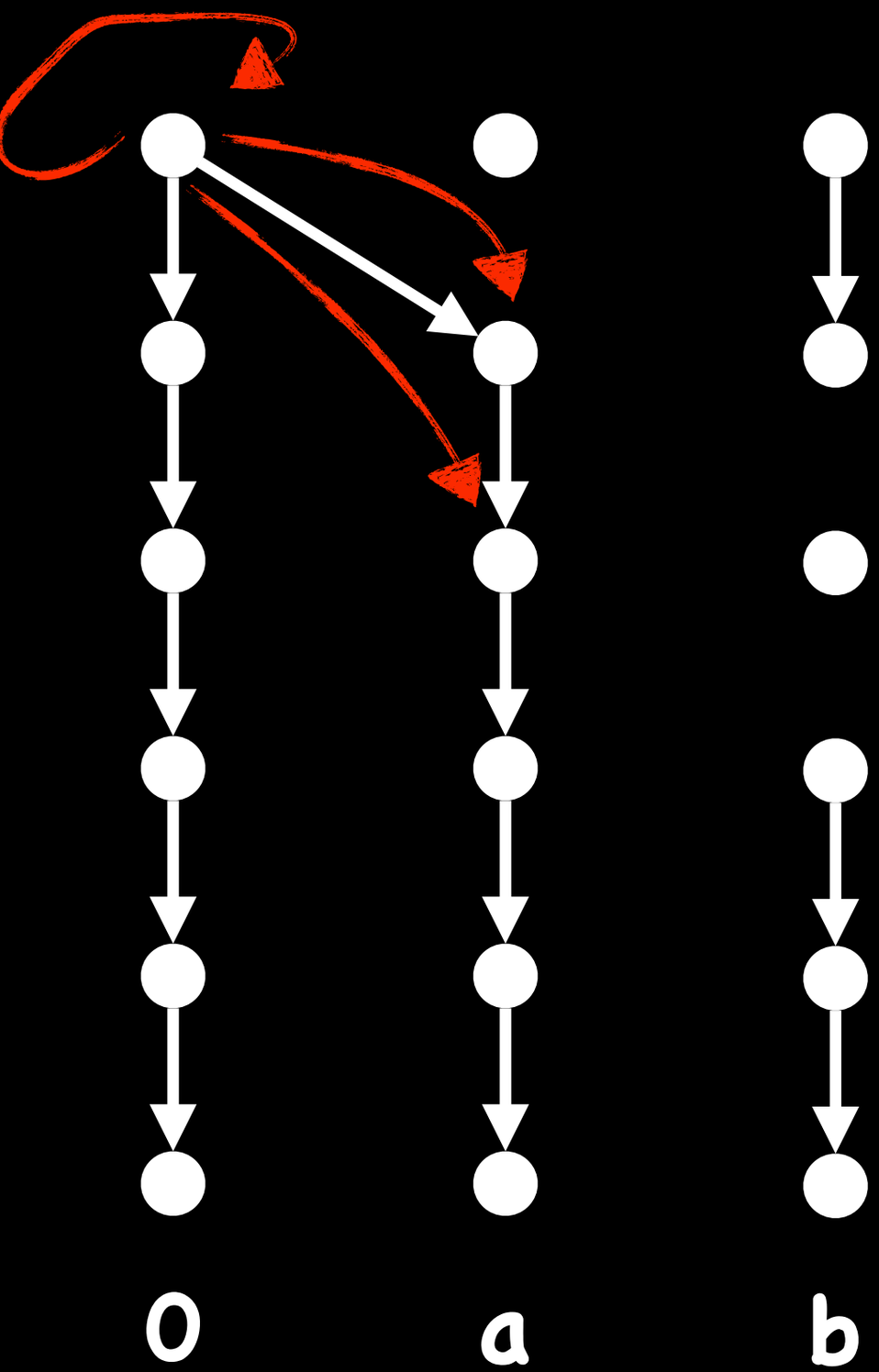
RHS 95

flow functions must be distributive:

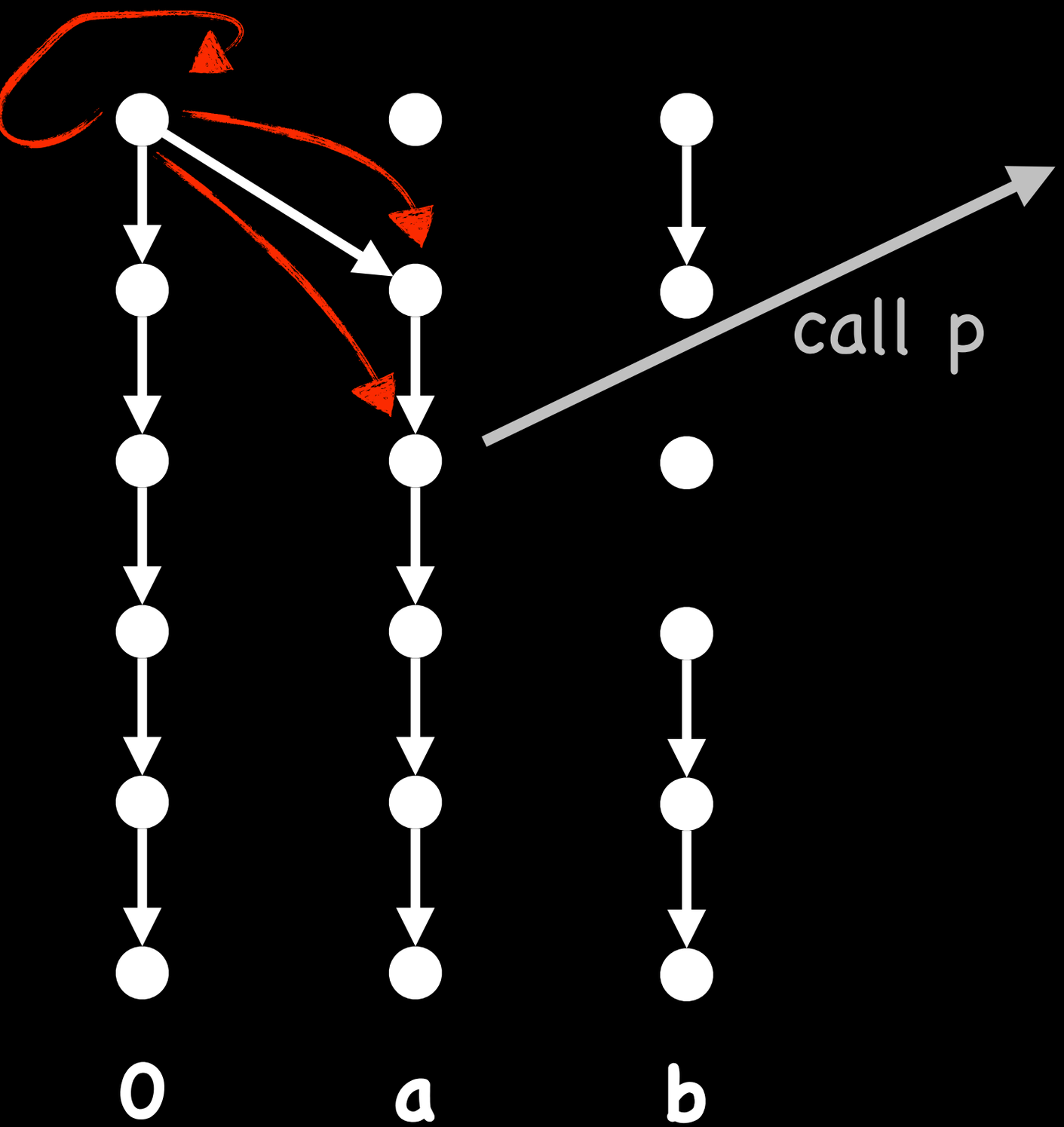
$f(\mathbf{a}) \sqcap f(\mathbf{b}) = f(\mathbf{a} \sqcap \mathbf{b})$



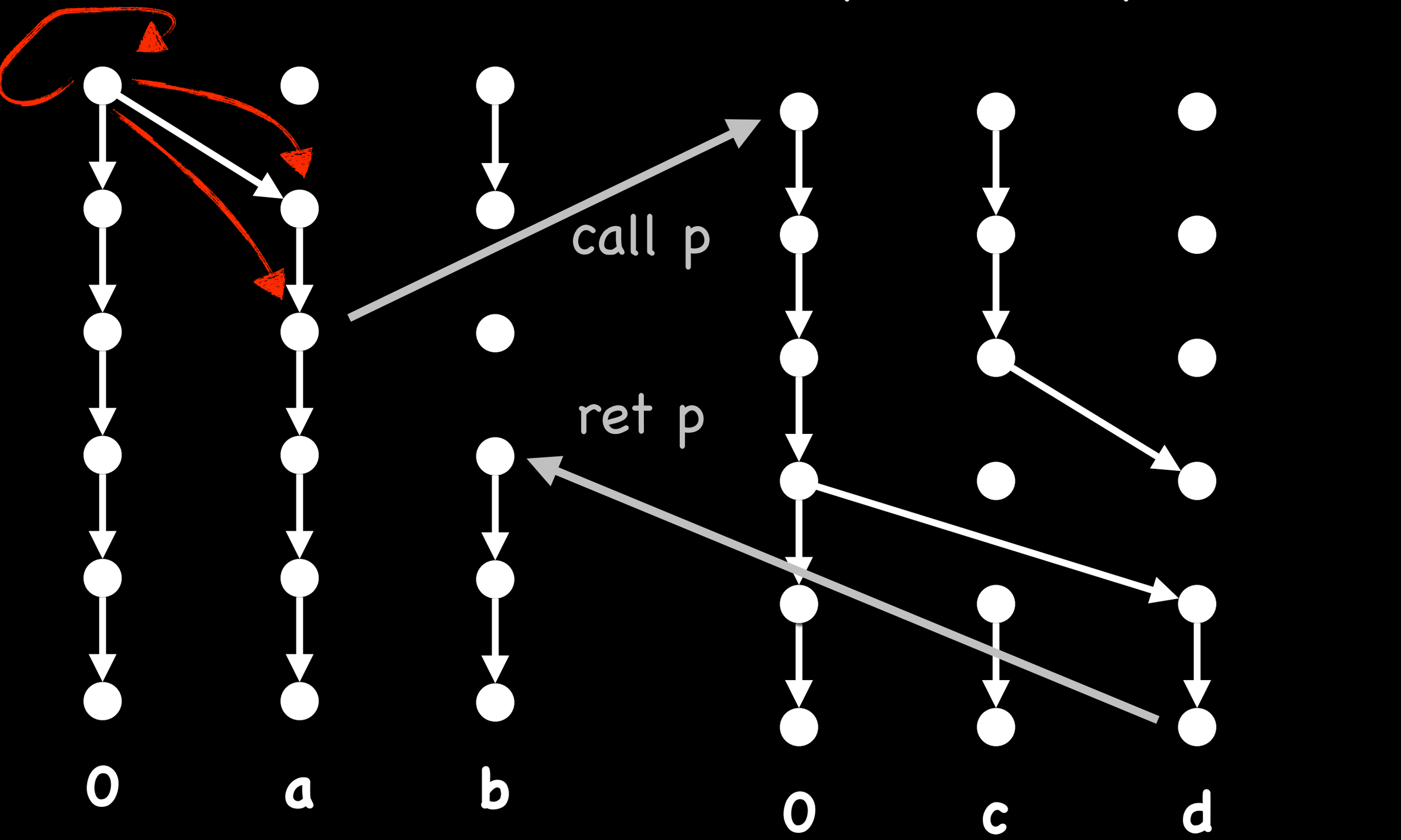
method calls



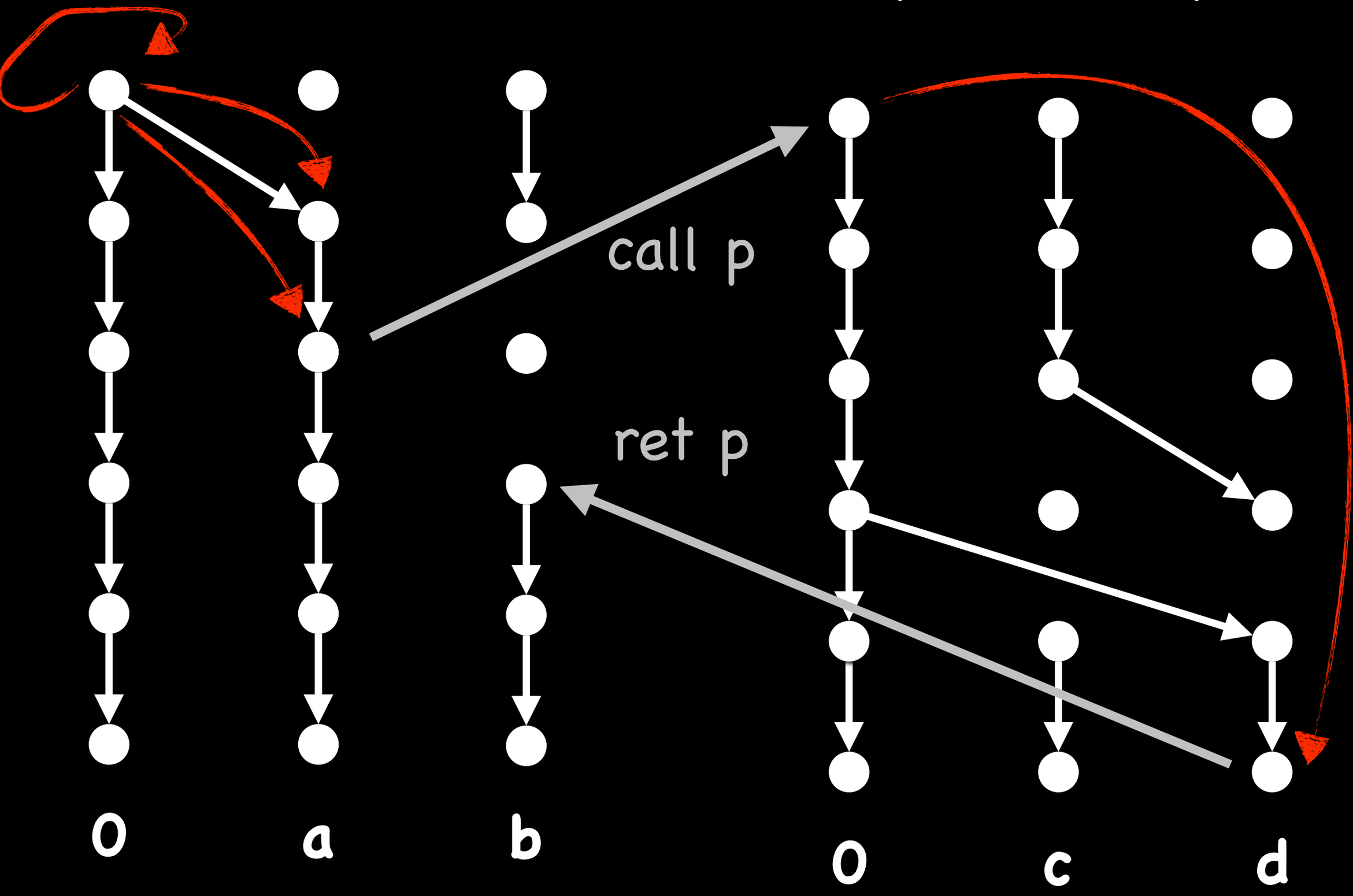
method calls



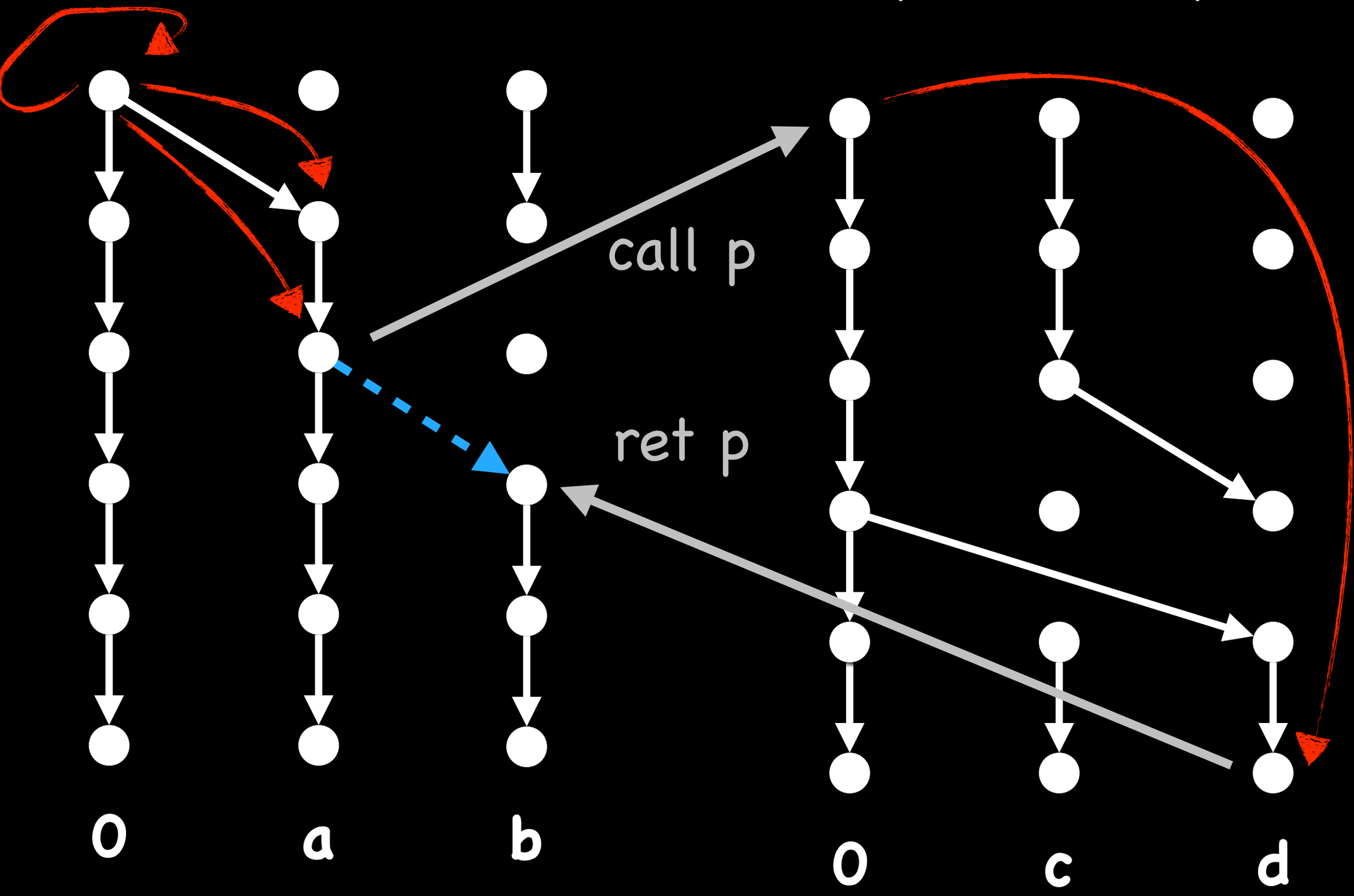
method calls



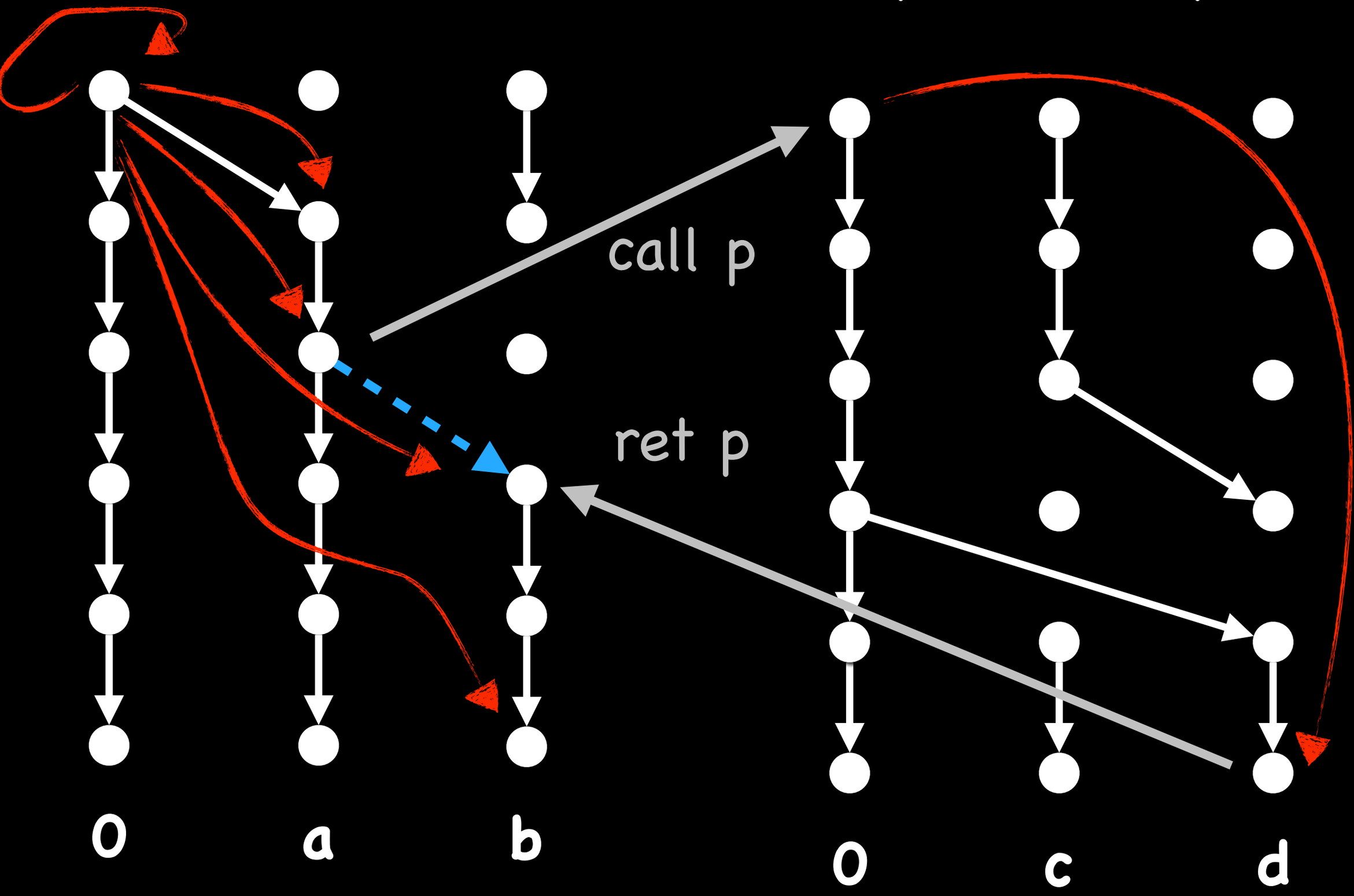
method calls



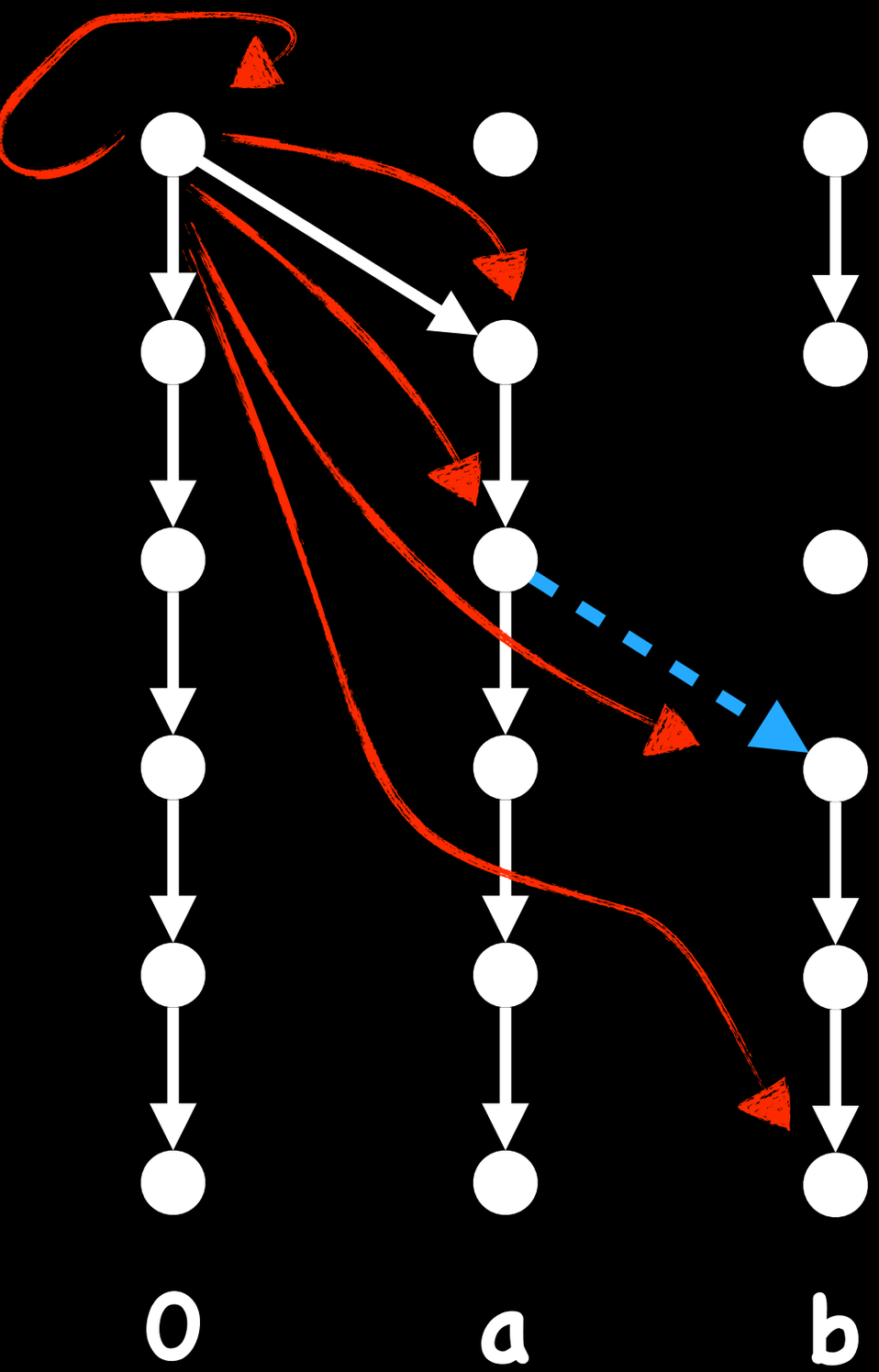
method calls



method calls



method calls



afterwards method-local decision:

b holds at **s** if there is a summary edge from startOf(methodOf(**s**),**d'**) to (**s**,**b**) for some **d'**