오픈소스 대상
고성능 정적 분석 기술

오학주

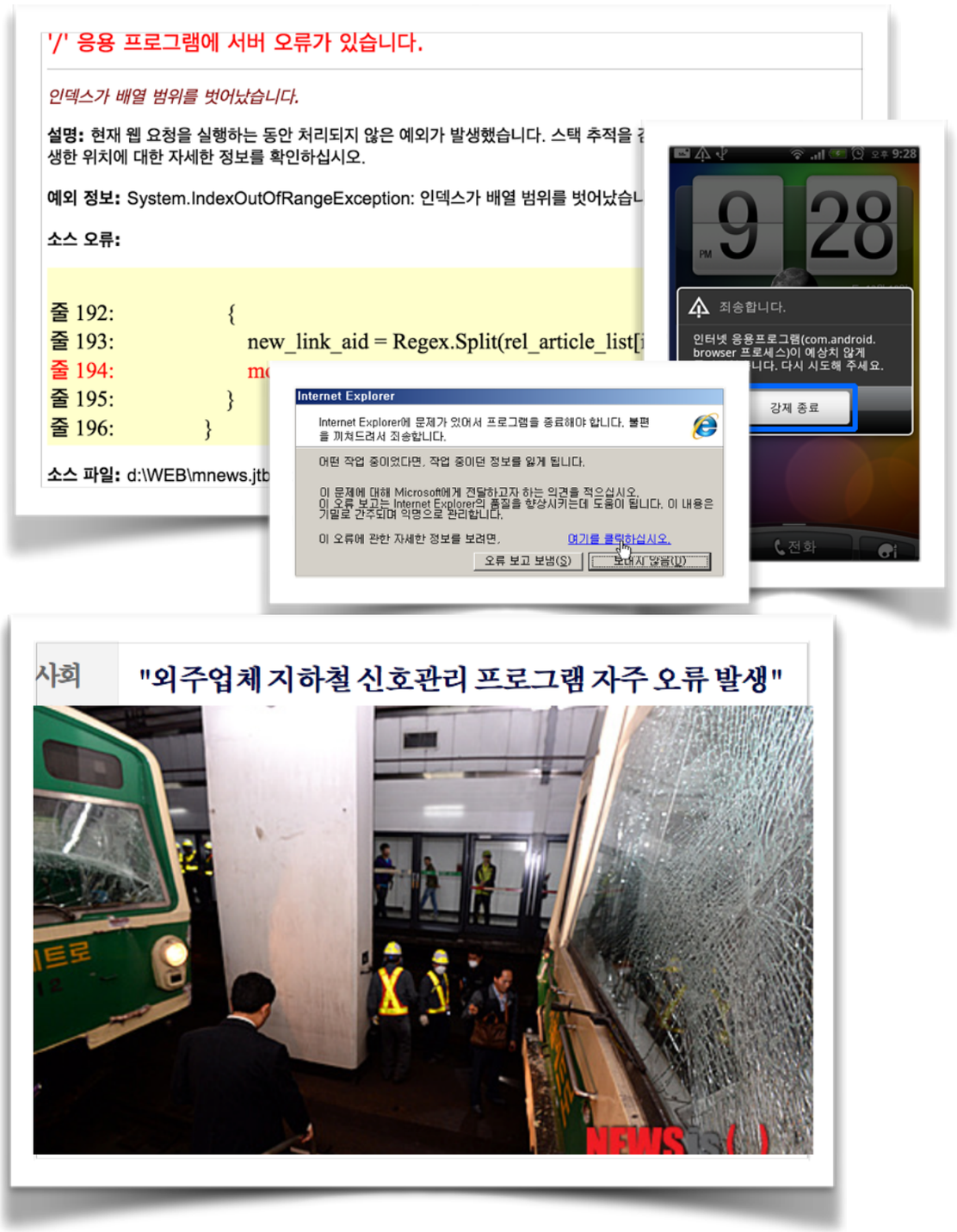
프로그래밍 연구실
고려대학교 정보대학

May. 12, 2016

Motivation: Unsafe Software

Motivation: Unsafe Software

Motivation: Unsafe Software



Motivation: Unsafe Software

Motivation: Unsafe Software

Current Technology for Safe SW

Manual, ad-hoc, postmortem:

code review, testing, simulation, debugging, etc

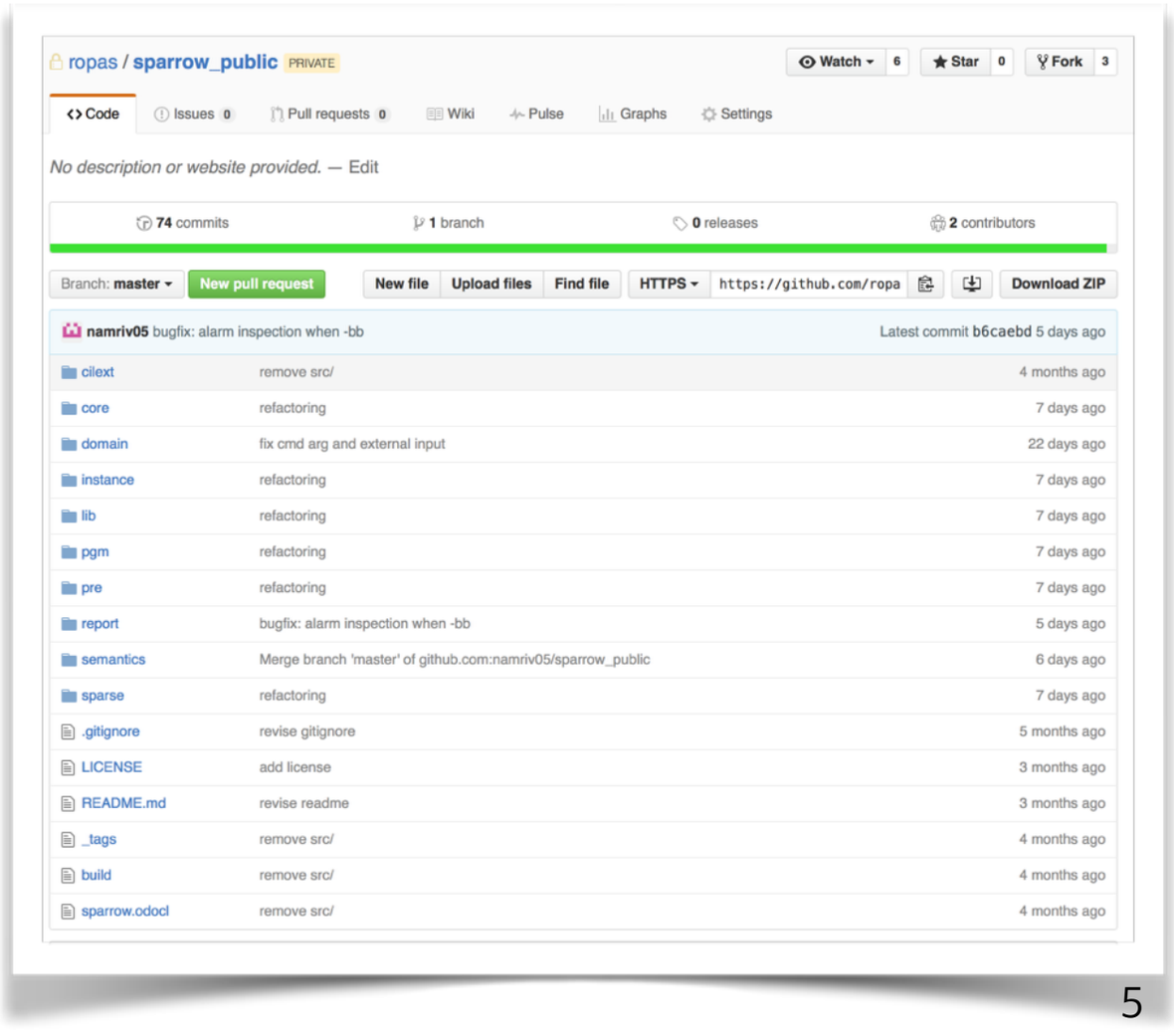
Our Mission

Technology for "Software MRI"

Sparrow

The Early Bird

- Aims to detect memory errors in C programs
  - e.g., buffer-overrun, memory leak, null-dereference, etc
- Features (vs. testing)
  - Full **automation**
  - Find bugs **early**
  - **All bugs** found



```
16      static char *curfinal = "HDACB  FE";
17
18      keysym = read_from_input ();
19
20      if ((((KeySym)(keysym) >= 0xFF91) && ((KeySym)(keysym) <= 0xFF94)))
21      {
22         unparseputc((char)(keysym-0xFF91 +'P'), pty);
23         key = 1;
24      }
25      else if (keysym >= 0)
26      {
27          if (keysym < 16)
28          {
29              if (read_from_input())
30              {
31                 if (keysym >= 10) return;
32                 curfinal[keysym] = 1;
33              }
34              else
35              {
36                  curfinal[keysym] = 2;
37              }
38          }
39          if (keysym < 10)
40          {
41              unparseputc(curfinal[keysym], pty);
42          }
43      }
```

```
16      static char *curfinal = "HDACB  FE";
17
18      keysym = read_from_input ();
19
20      if ((((KeySym)(keysym) >= 0xFF91) && ((KeySym)(keysym) <= 0xFF94)))
21      {
22         unparseputc((char)(keysym-0xFF91 +'P'), pty);
23         key = 1;
24      }
25      else if (keysym >= 0)
26      {
27          if (keysym < 16)
28          {
29              if (read_from_input())
30              {
31                  if (keysym >= 10) return;
32                  curfinal[keysym] = 1;
33              }
34              else
35              {
36                  curfinal[keysym] = 2;
37              }
38          }
39          if (keysym < 10)
40          {
41              unparseputc(curfinal[keysym], pty);
42          }
43      }
```

safe

buffer-overrun

safe

Sparrow automatically pinpoints the buffer-overrun bug

```
16      static char *curfinal = "HDACB  FE";
17
18      keysym = read_from_input ();
19
20      if ((((KeySym)(keysym) >= 0xFF91) && ((KeySym)(keysym) <= 0xFF94)))
21      {
22         unparseputc((char)(keysym-0xFF91 +'P'), pty);
23         key = 1;
24      }
25      else if (keysym >= 0)
26      {
27          if (keysym < 16)
28          {
29              if (read_from_input())
30              {
31                  if (keysym >= 10) return;
32                  curfinal[keysym] = 1;
33              }
34              else
35              {
36                  curfinal[keysym] = 2;
37              }
38          }
39          if (keysym < 10)
40          {
41              unparseputc(curfinal[keysym], pty);
42          }
43      }
```

curfinal: buffer of size 10

safe

buffer-overrun

safe

Sparrow automatically pinpoints the buffer-overrun bug

```
16      static char *curfinal = "HDACB   FE";
17
18      keysym = read_from_input ();
19
20      if ((((KeySym)(keysym) >= 0xFF9                ym) <= 0xFF94)))
21      {
22          unparseputc((char)(keysym-0xFF91 +'P'), pty);
23          key = 1;
24      }
25      else if (keysym >= 0)
26      {
27          if (keysym < 16)
28          {
29              if (read_from_input())
30              {
31                  if (keysym >= 10) return;
32                  curfinal[keysym] = 1;
33              }
34              else
35              {
36                  curfinal[keysym] = 2;
37              }
38          }
39          if (keysym < 10)
40          {
41              unparseputc(curfinal[keysym], pty);
42          }
43      }
```

curfinal: buffer of size 10

keysym: any integer

safe

buffer-overrun

safe

Sparrow automatically pinpoints the buffer-overrun bug

```
16      static char *curfinal = "HDACB   FE";
17
18      keysym = read_from_input ();
19
20      if ((((KeySym)(keysym) >= 0xFF9                ym) <= 0xFF94)))
21      {
22          unparseputc((char)(keysym-0xFF91 +'P'), pty);
23          key = 1;
24      }
25      else if (keysym >= 0)
26      {
27          if (keysym < 16)
28          {
29              if (read_from_input())
30              {
31                  if (keysym >= 10) return;
32                  curfinal[keysym] = 1;
33              }
34              else
35              {
36                  curfinal[keysym] = 2;
37              }
38          }
39          if (keysym < 10)
40          {
41              unparseputc(curfinal[keysym], pty);
42          }
43      }
```

curfinal: buffer of size 10

keysym: any integer

keysym: [0,15]

safe

buffer-overrun

safe

Sparrow automatically pinpoints the buffer-overrun bug

```
16      static char *curfinal = "HDACB   FE";
17
18      keysym = read_from_input ();
19
20      if ((((KeySym)(keysym) >= 0xFF9                ym) <= 0xFF94)))
21      {
22          unparseputc((char)(keysym-0xFF91 +'P'), pty);
23          key = 1;
24      }
25      else if (keysym >= 0)
26      {
27          if (keysym < 16)
28          {
29              if (read_from_input())
30              {
31                  if (keysym >= 10) return;
32                  curfinal[keysym] = 1;
33              }
34              else
35              {
36                  curfinal[keysym] = 2;
37              }
38          }
39          if (keysym < 10)
40          {
41              unparseputc(curfinal[keysym], pty);
42          }
43      }
```

curfinal: buffer of size 10

keysym: any integer

keysym: [0,15]

safe

buffer-overrun

curfinal:[10,10]
keysym: [10,15]

safe

Sparrow automatically
pinpoints the buffer-overrun bug

Performance in Reality

Programs	Size KLOC	Time (sec)	True Alarms	False Alarms
gzip-1.2.4	9.1	8.55	0	17
gnuchess-5.07	17.8	179.58	1	8
tcl8.4.14/unix	17.9	585.99	1	14
hanterm-3.1.6	25.6	52.25	34	1
sed-4.0.8	26.8	49.34	2	11
tar-1.13	28.3	57.98	1	10
grep-2.5.1a	31.5	47.26	0	1
bison-2.3	48.4	281.84	0	18
openssh-4.3p2	77.3	97.69	0	9
fftw-3.1.2	184.0	102.17	9	4
httpd-2.2.2	316.4	265.43	10	33
net-snmp-5.4	358.0	899.73	3	36

Static Program Analysis

- Predict SW behavior statically and automatically
  - **static**: before execution, before sell / embed
  - **automatic**: sw is analyzed by sw (“static analyzers”)
- Applications

Static Program Analysis

- Predict SW behavior statically and automatically
  - **static**: before execution, before sell / embed
  - **automatic**: sw is analyzed by sw ("static analyzers")
- Applications
  - **bug-finding**. e.g., find runtime failures of programs

Static Program Analysis

- Predict SW behavior statically and automatically
  - **static**: before execution, before sell / embed
  - **automatic**: sw is analyzed by sw ("static analyzers")
- Applications
  - **bug-finding**. e.g., find runtime failures of programs
  - **security**. e.g., is this app malicious or benign?

Static Program Analysis

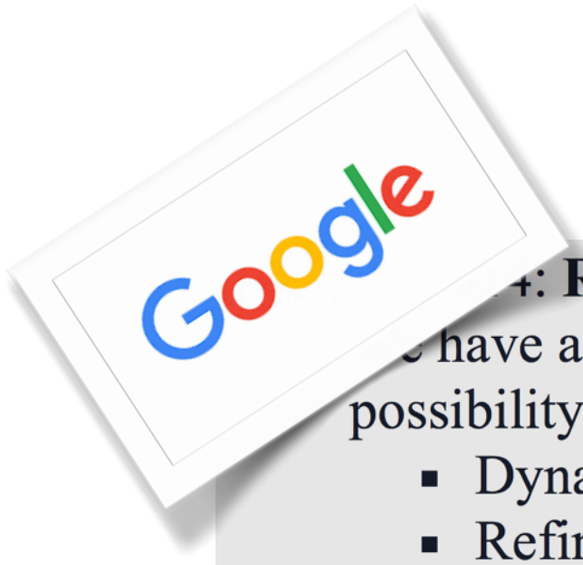
- Predict SW behavior statically and automatically
  - **static**: before execution, before sell / embed
  - **automatic**: sw is analyzed by sw ("static analyzers")

- Applications
  - **bug-finding**. e.g., find runtime failures of programs
  - **security**. e.g., is this app malicious or benign?
  - **verification**. e.g., does the program meet its specification?

Static Program Analysis

- Predict SW behavior statically and automatically
  - **static**: before execution, before sell / embed
  - **automatic**: sw is analyzed by sw ("static analyzers")

- Applications
  - **bug-finding**. e.g., find runtime failures of programs
  - **security**. e.g., is this app malicious or benign?
  - **verification**. e.g., does the program meet its specification?
  - **compiler optimization**, e.g., automatic parallelization



4: Research Internship Positions in Program Analysis @ Google

have a number of research internship openings in 2015. Internships could possibility. Possible topics include:

- Dynamic symbolic execution
- Refinement-based alias analysis
- Distributed static analysis of large applications
- Identification of vulnerabilities in Java through static analysis
- Concurrent data-flow analysis
- Refining flow-insensitive analyses

- Ideal candidates would have strong research background and solid (C++) pro

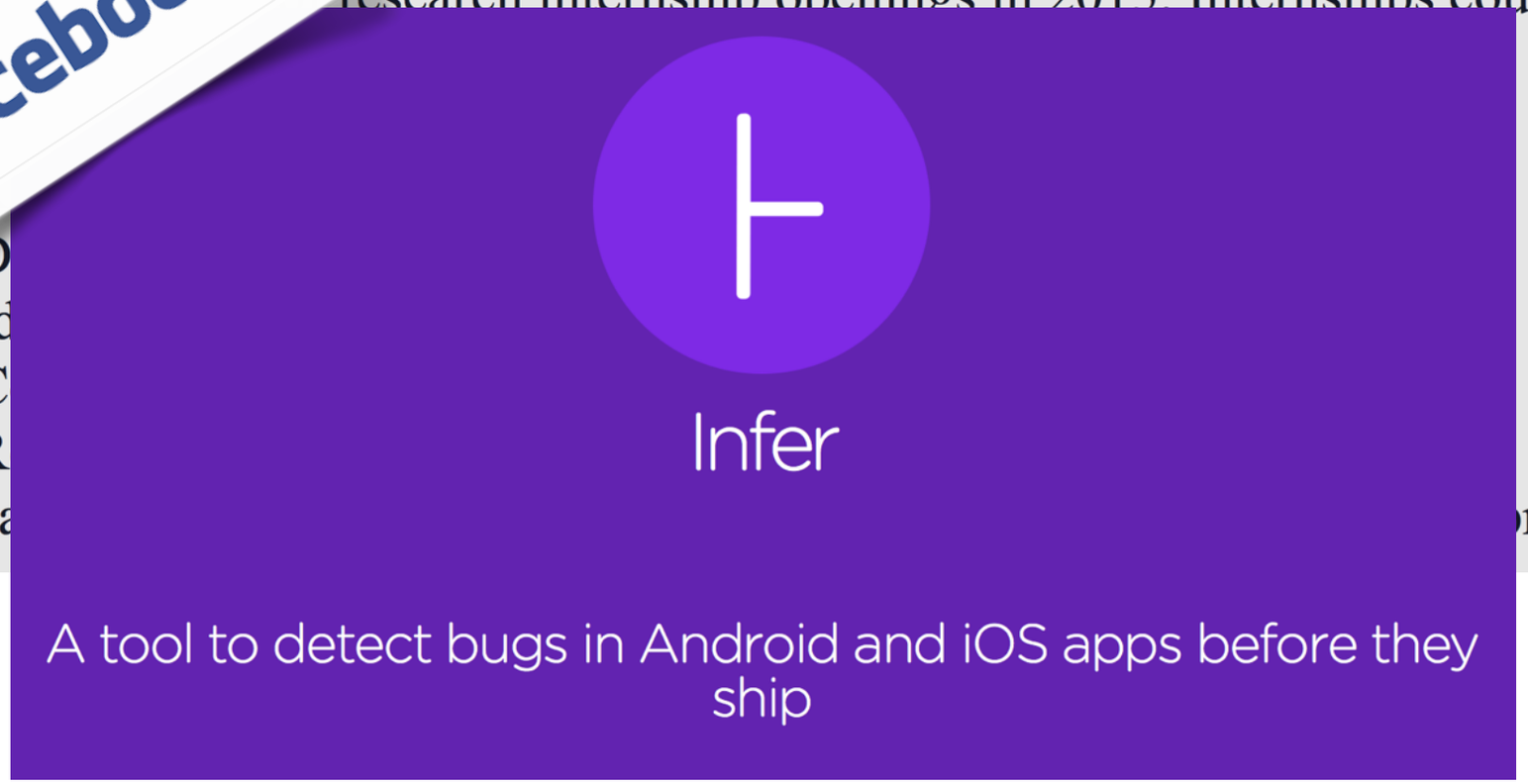
ternship Positions in Program Analysis @ Google

esearch internship openings in 2015. Internships coul

- D
- Id
- C
- R

- Ideal ca

rc



ip Positions in Program Analysis @ Google

internship openings in 2015. Internships coul

- D
- Id
- C
- R

- Ideal ca

orc

A tool to d

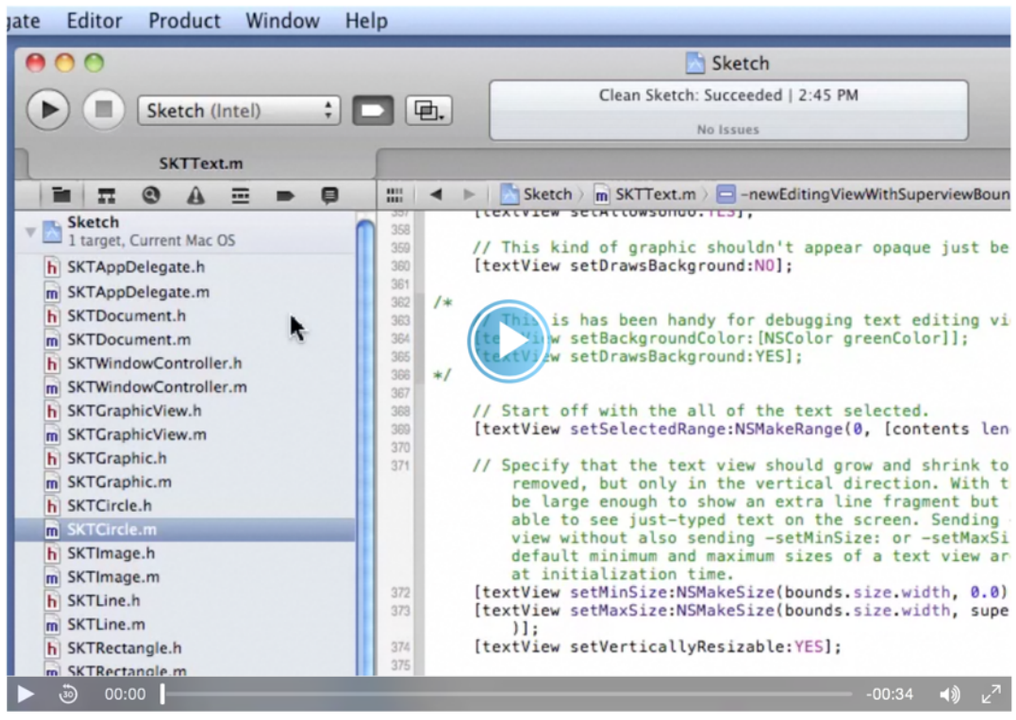
tic Code Analysis

potential bugs—in the source code of a project with the static analyzer built into Xcode. Source code may have subtle e
he compiler and manifest themselves only at runtime, when they could be difficult to identify and fix.

Steps

1. Choose Product > Analyze.
2. In the issue navigator, select an analyzer message.
3. In the source editor, click the corresponding message.
4. Use the pop-up menu in the analysis results bar above the edit area to study the flow path of the flaw.
5. Edit the code to fix the flaw.

   The video shows the process of looking at a flaw in the source file `SKTText.m`.



Google

facebook

FireEye

ip Positions in Program Analysis @ Google

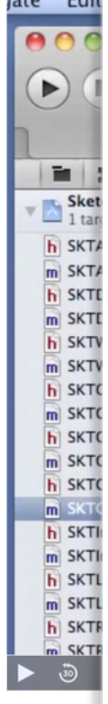
internship openings in 2015. Internships could

- D
- Id
- C
- R
- Ideal ca

A tool to d

2. In the source navig
3. In the source edit
4. Use the pop-up r
5. Edit the code to f

The video shows



FireEye

Products | Solutions | Mandiant

Home > Products > Mobile Security

Mobile Security

Mobile Threat Preve

Detect and prevent cyber attacks that spy on, profile, or use mobile devices

Malicious apps compromise mobile security to access private information, such as contact lists and calendar details. They also use mobile device features, such as cameras and microphones, to spy, profile users, or conduct cyber attacks.

FireEye Mobile Security (Mobile Threat Prevention) detects and prevents these mobile threats and provides visibility into mobile security trends across the enterprise. FireEye Mobile Threat Prevention also integrates with industry leading mobile device management (MDM) providers.

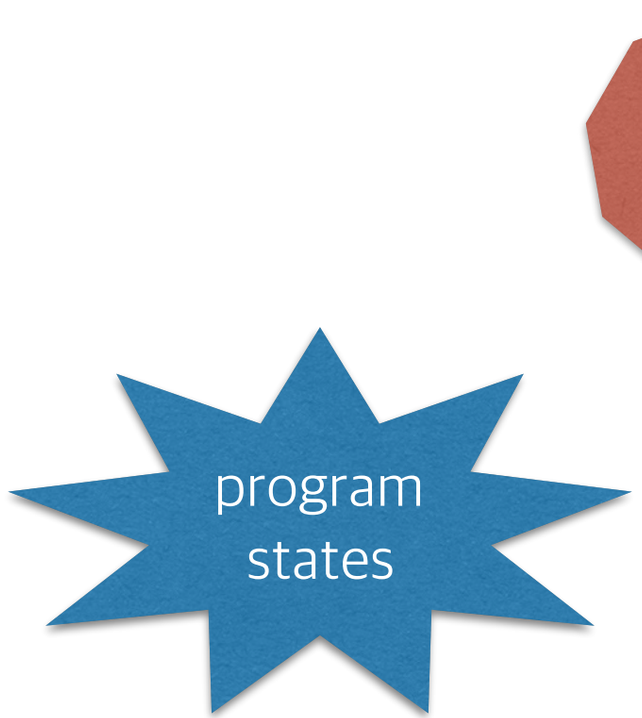
Our Research

Towards **sound**, **precise**, and **scalable** static analysis

- Sparse analysis framework
- Selective X-Sensitivity
  - by pre-analysis
  - by machine learning

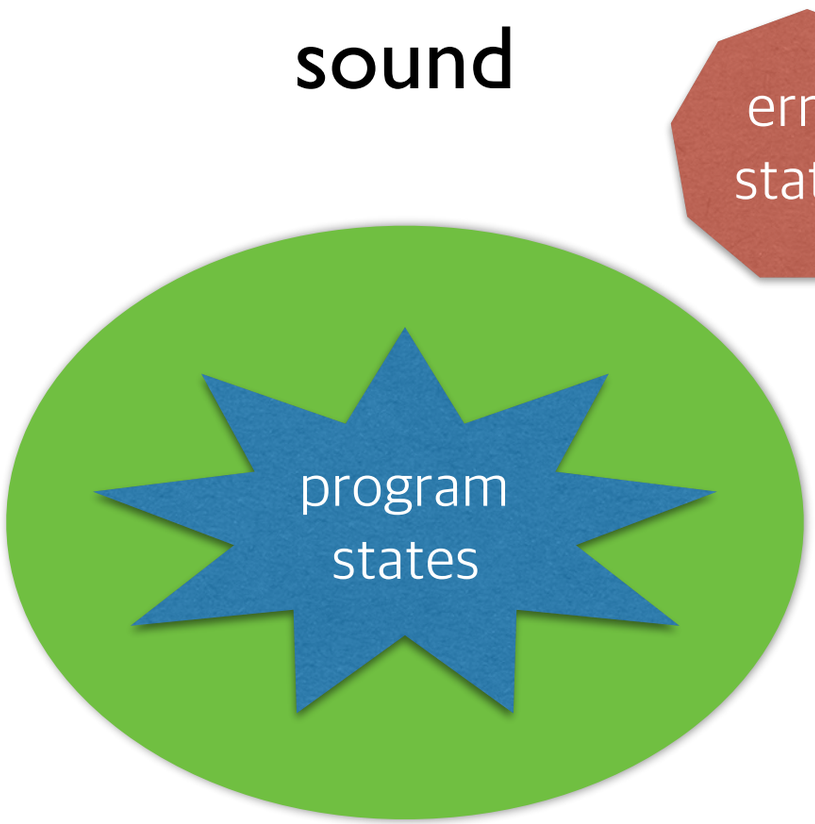
(1) Soundness

Find all bugs / verify absence



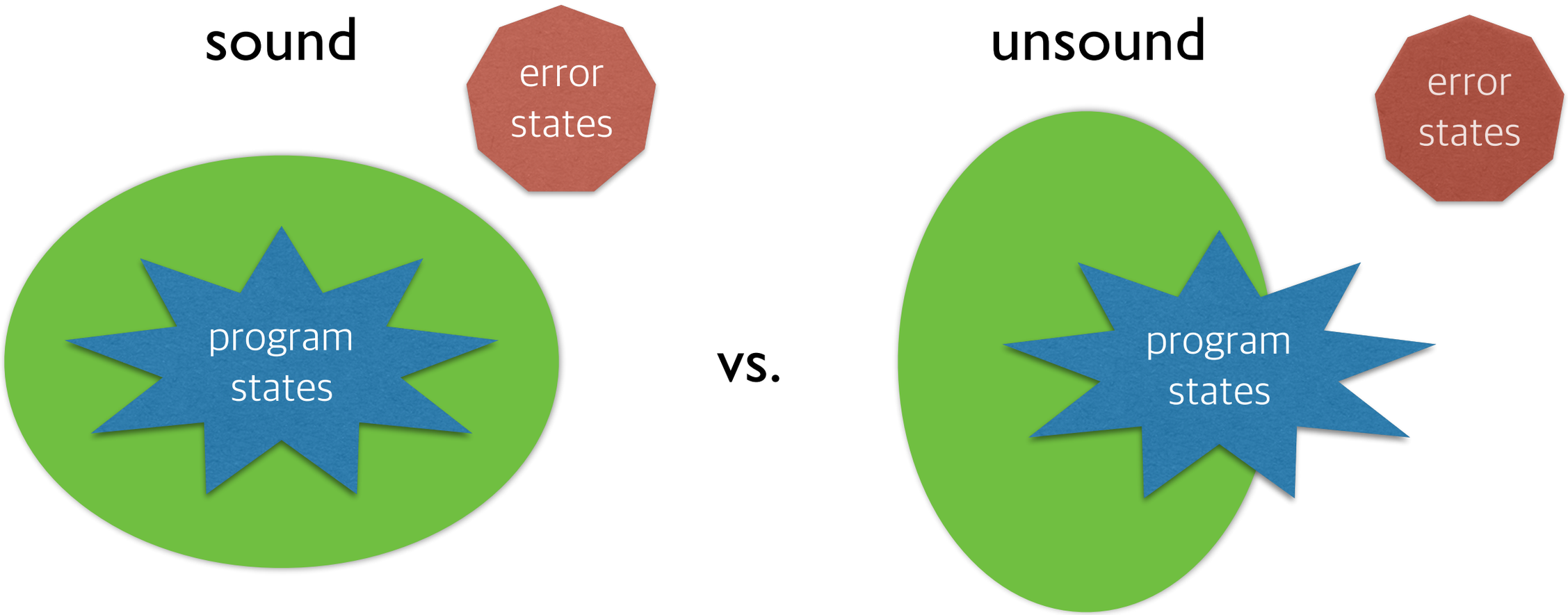
(1) Soundness

Find all bugs / verify absence



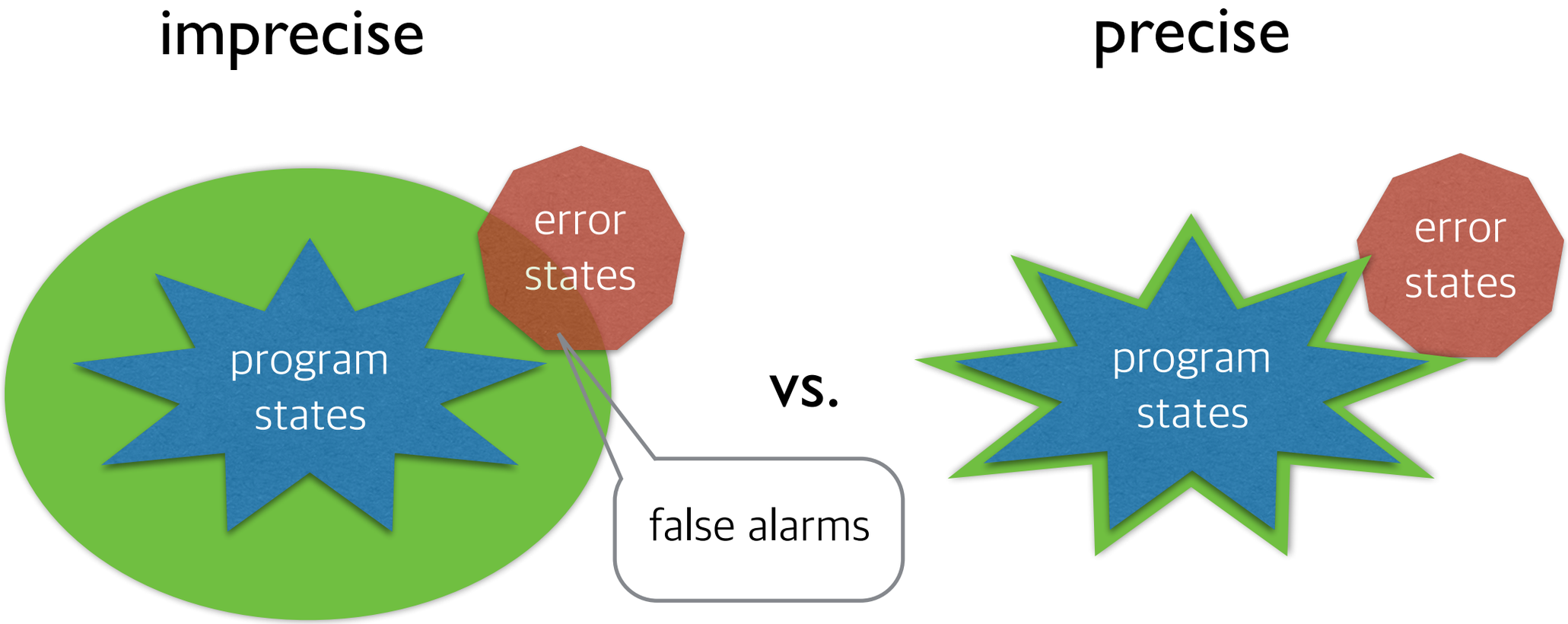
(1) Soundness

Find all bugs / verify absence



(2) Precision

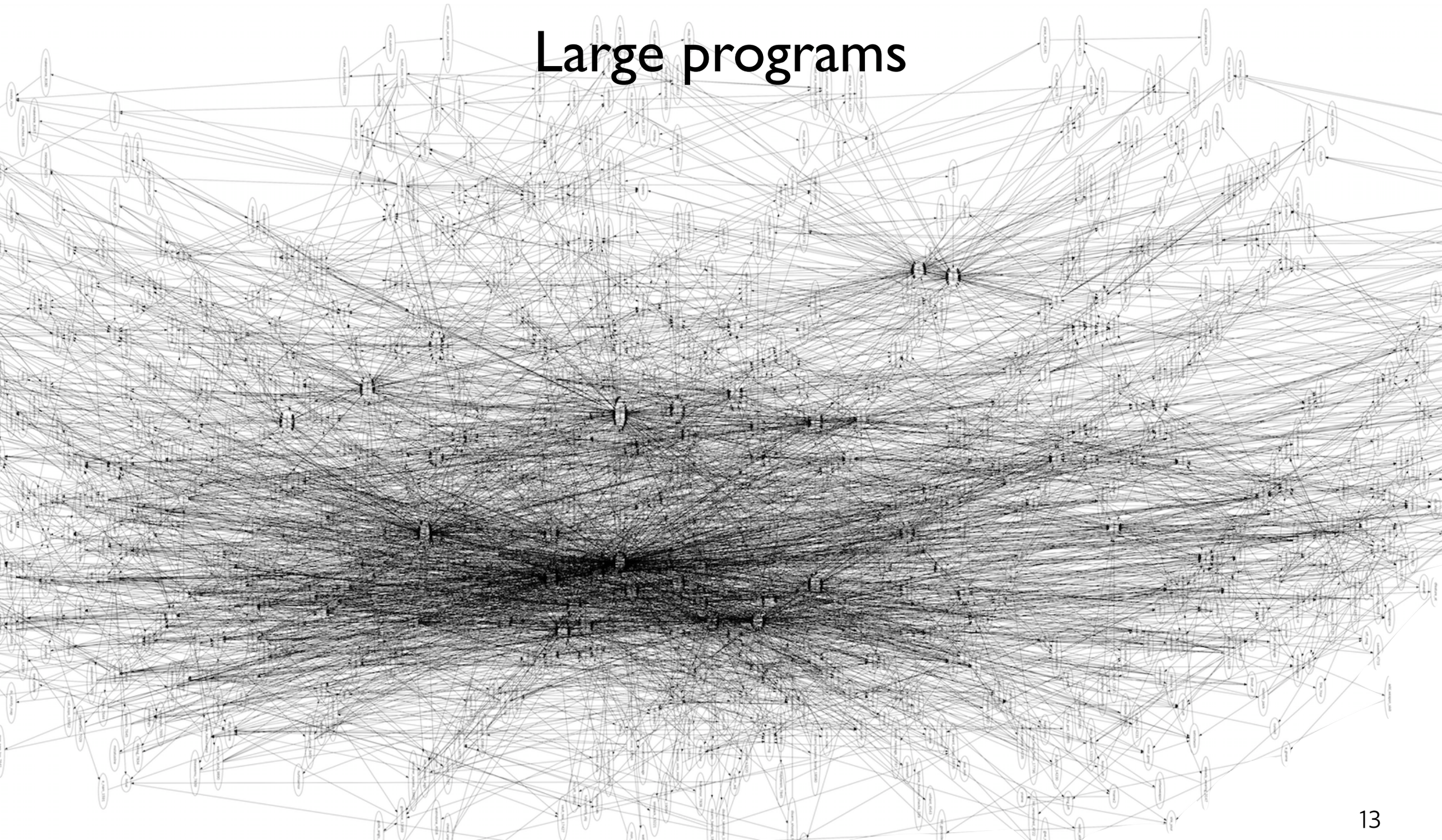
Few false alarms



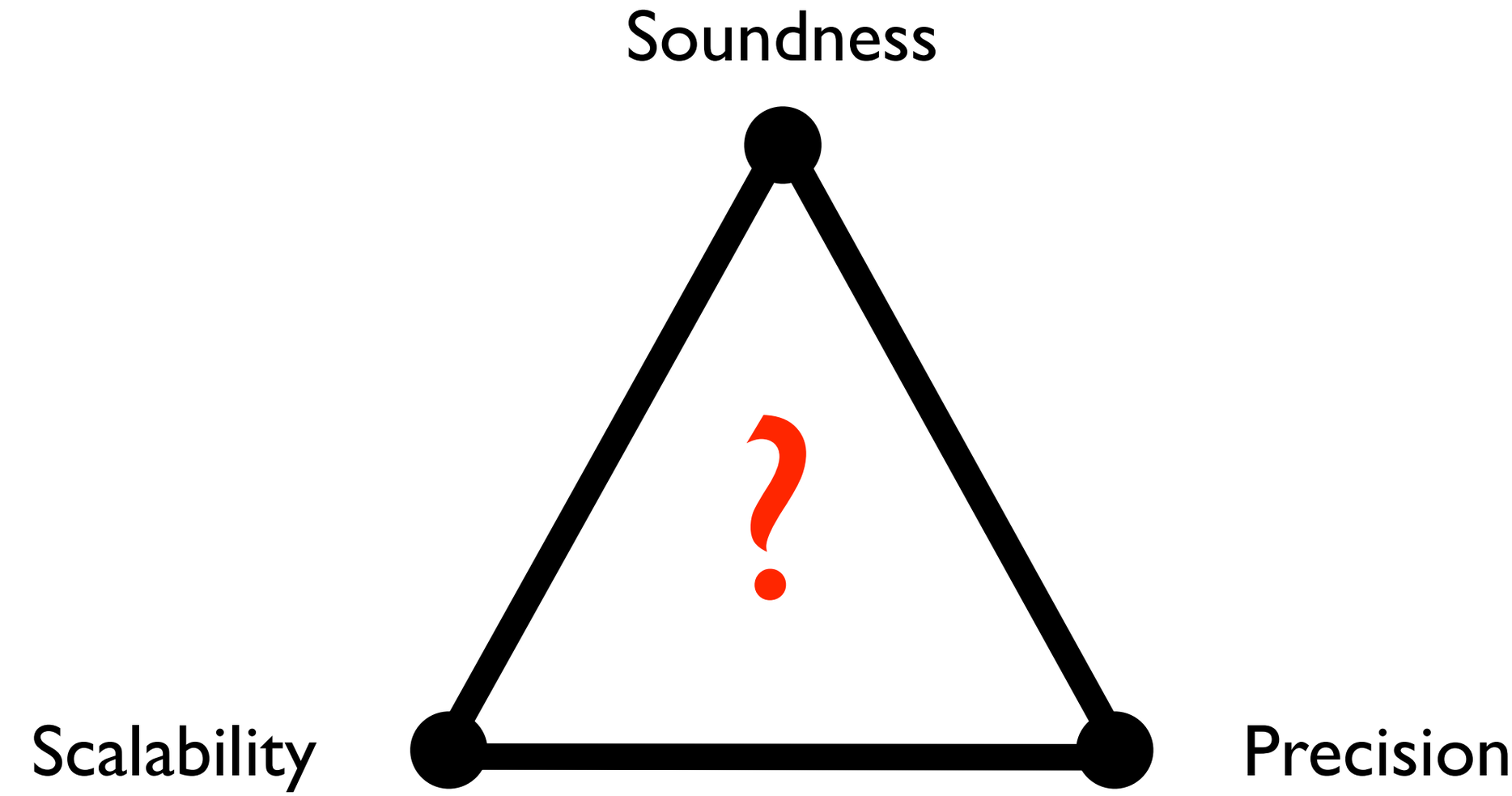
(3) Scalability

Large programs

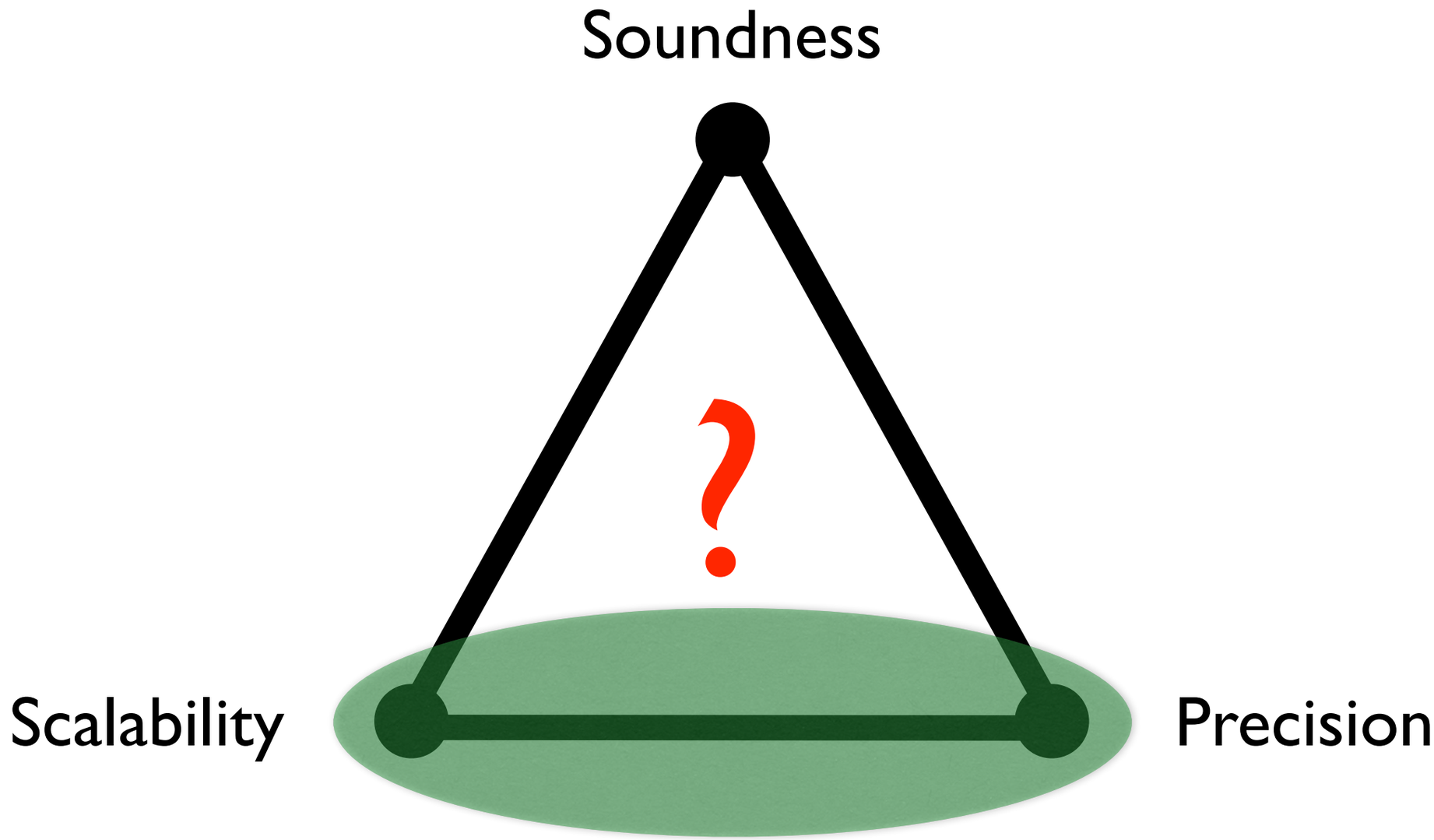
nethack-3.3.0 (211KLoC)



Challenge in Static Analysis

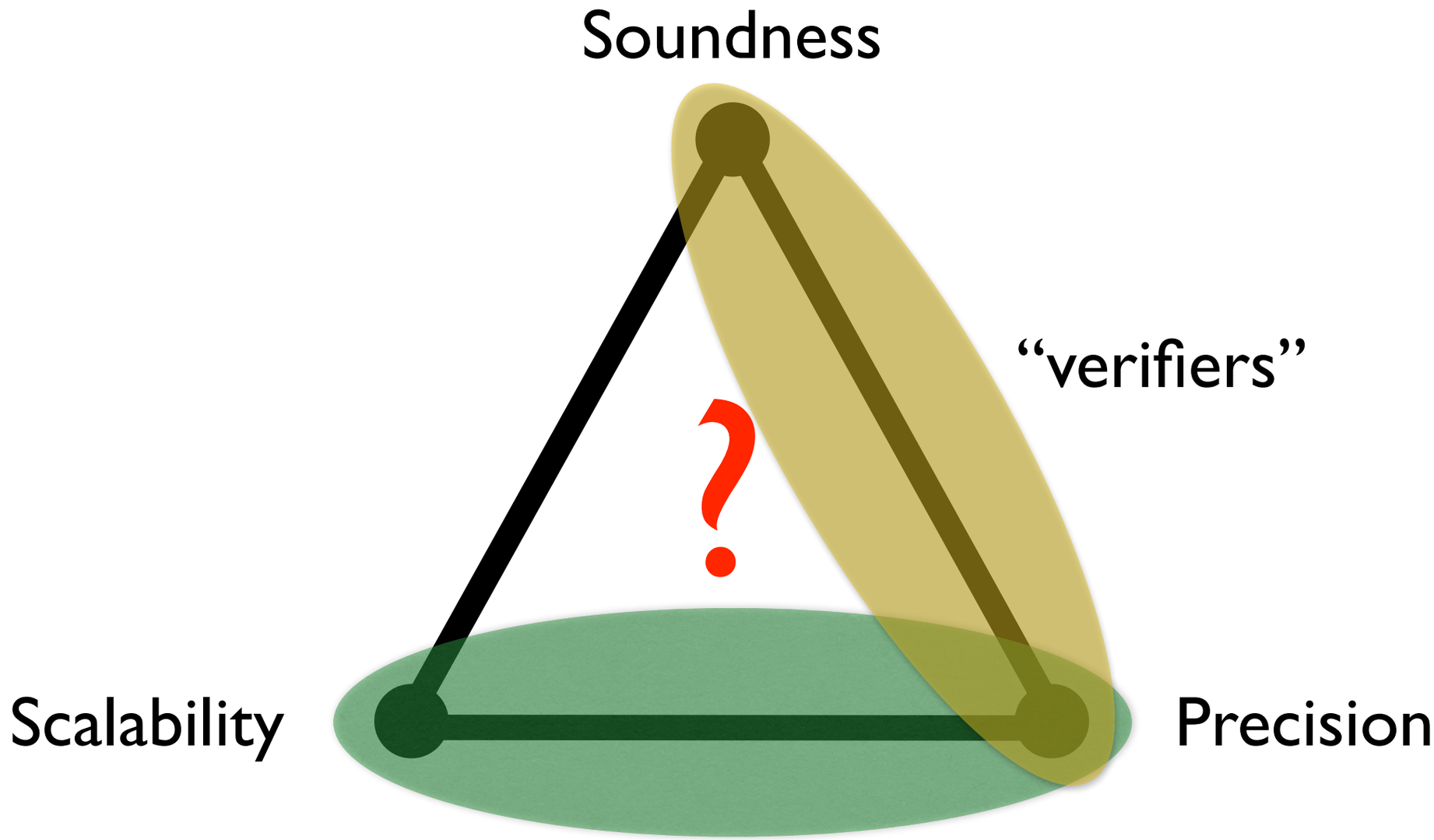


Challenge in Static Analysis

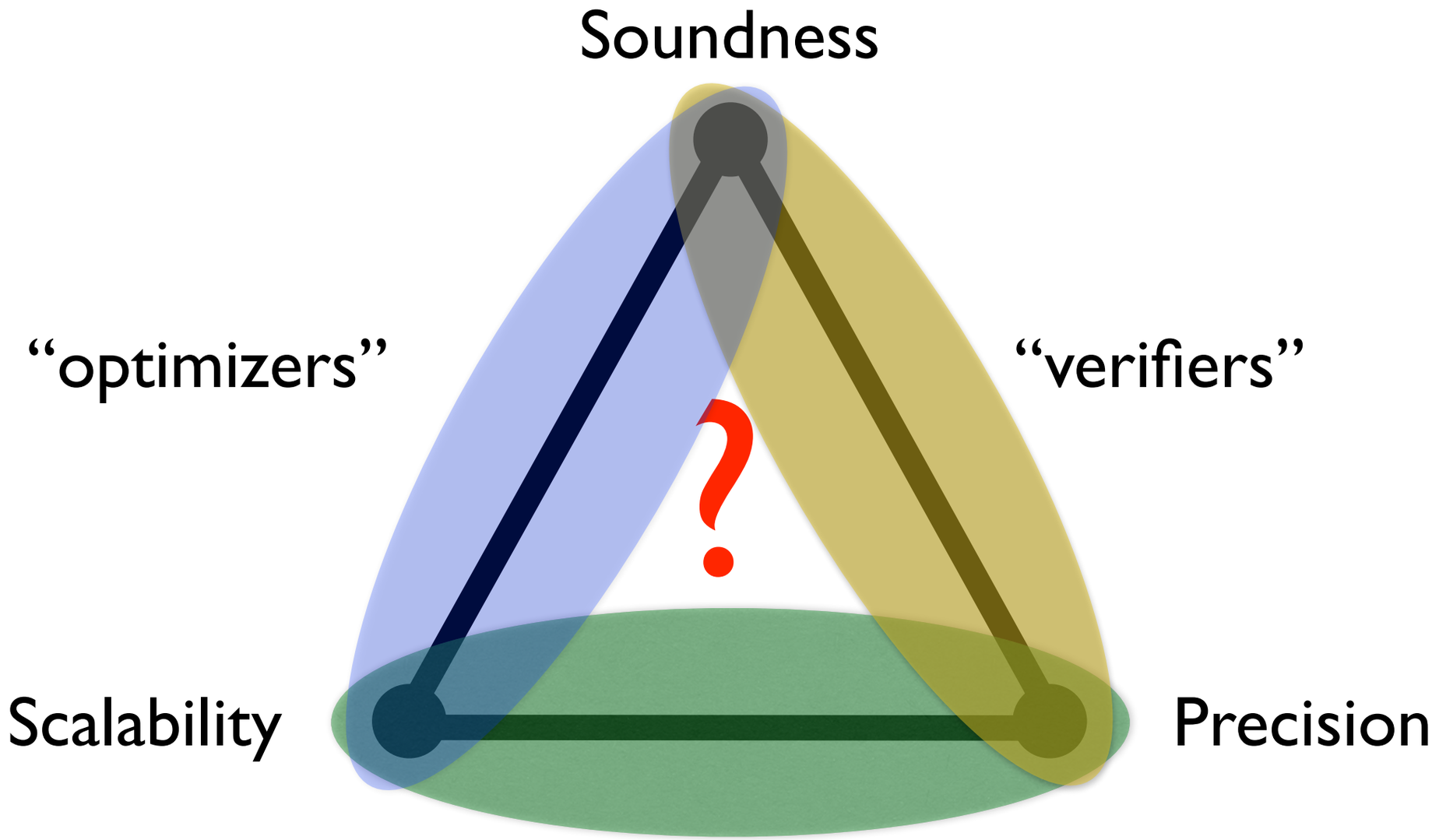


"bug-finders"

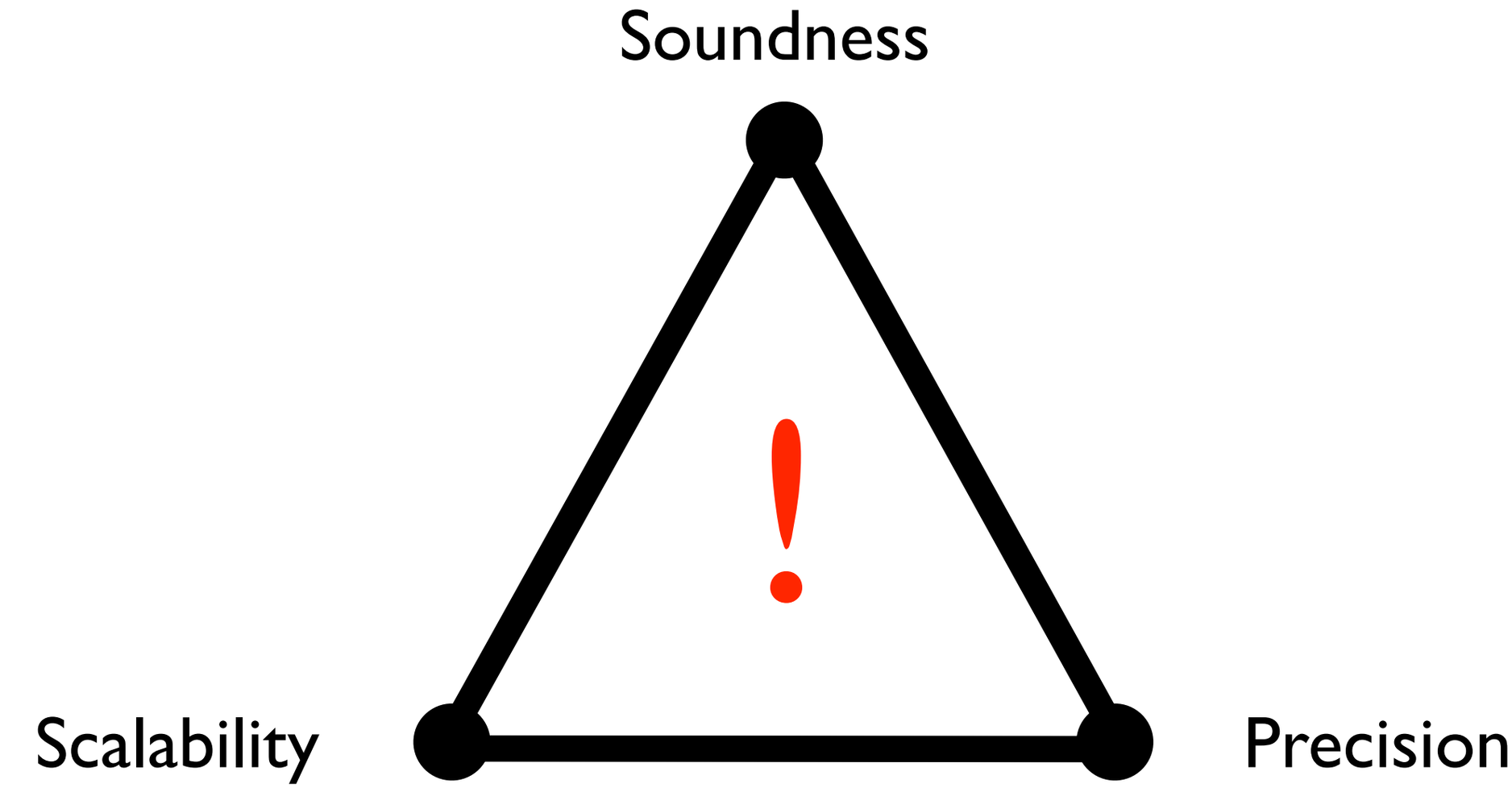
Challenge in Static Analysis



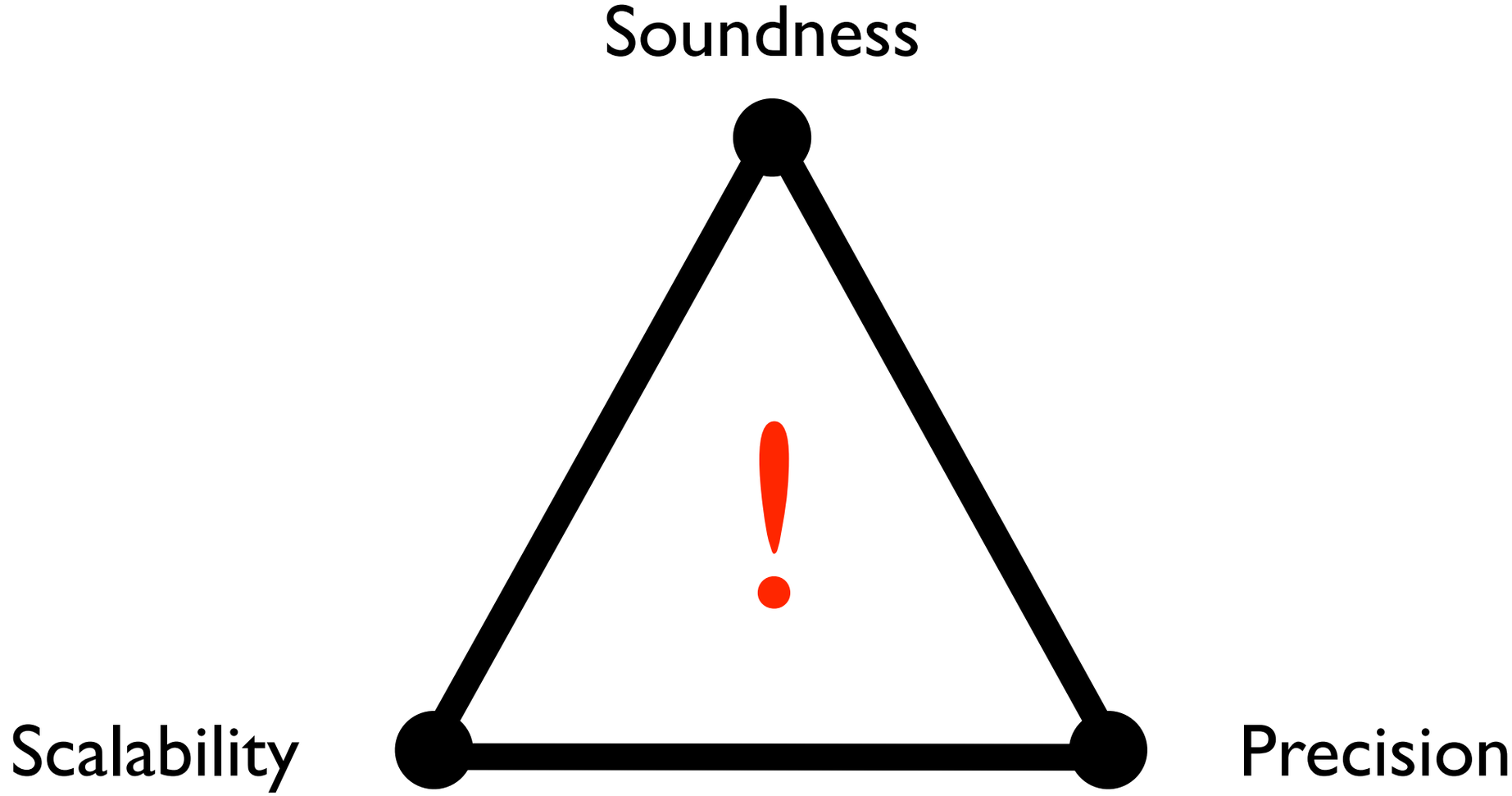
Challenge in Static Analysis



Our Research Goal

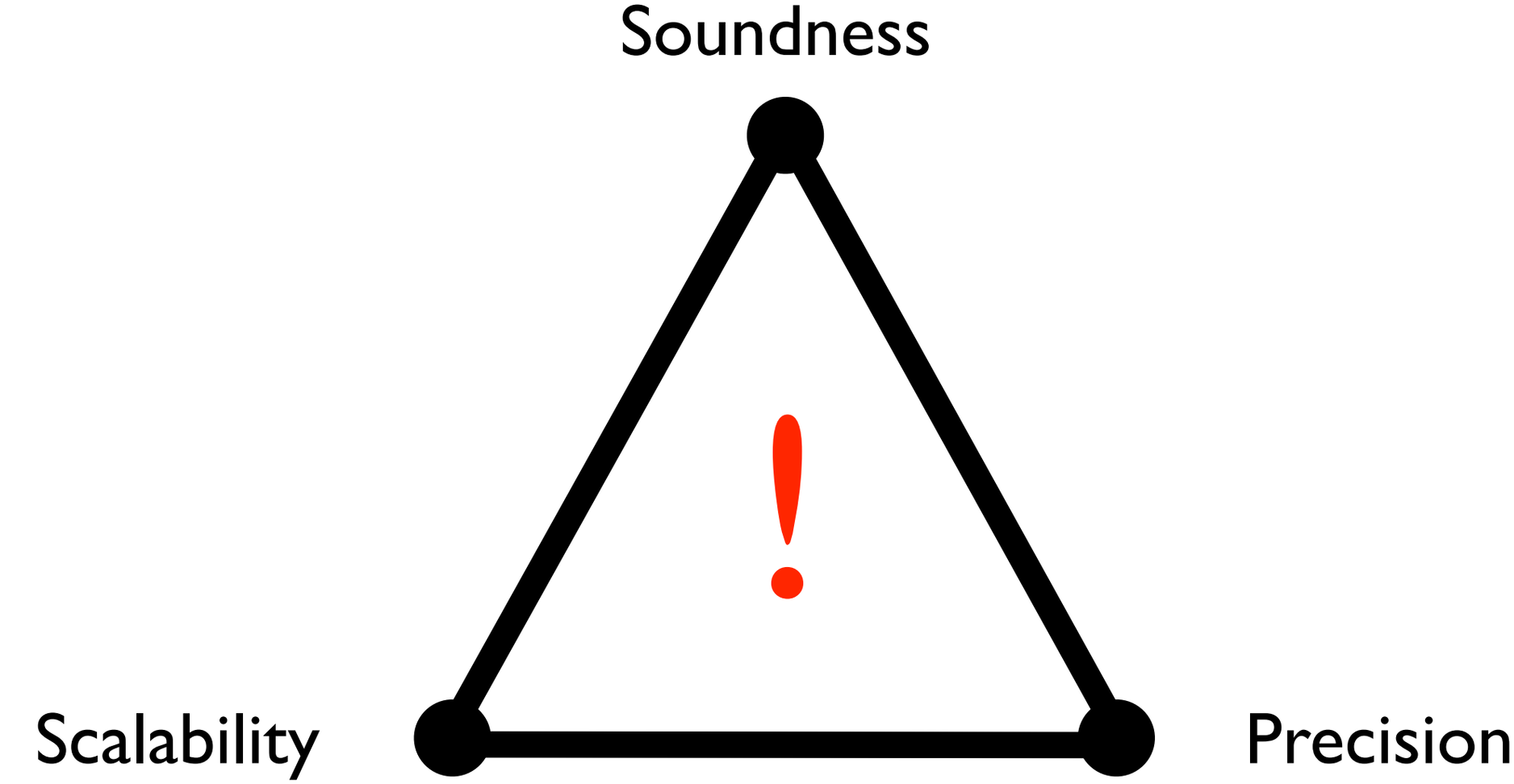


Our Research Goal



General Sparse Analysis Framework

Our Research Goal



General Sparse Analysis Framework

Selective X-Sensitivity Approach

Significance

- Cracked down the common sense that sound, precise, and scalable static analysis is infeasible
- Publication:
  - General Sparse Analysis Framework
    - **ACM PLDI 2012** (top conference in programming languages)
    - **ACM TOPLAS 2014** (top journal in programming languages)
  - Selective X-Sensitivity Approach
    - **ACM PLDI 2014** (top conference in programming languages)
    - **ACM OOPSLA 2015** (top conference in programming languages)
    - **ACM TOPLAS 2015** (top journal in programming languages)

Motivation

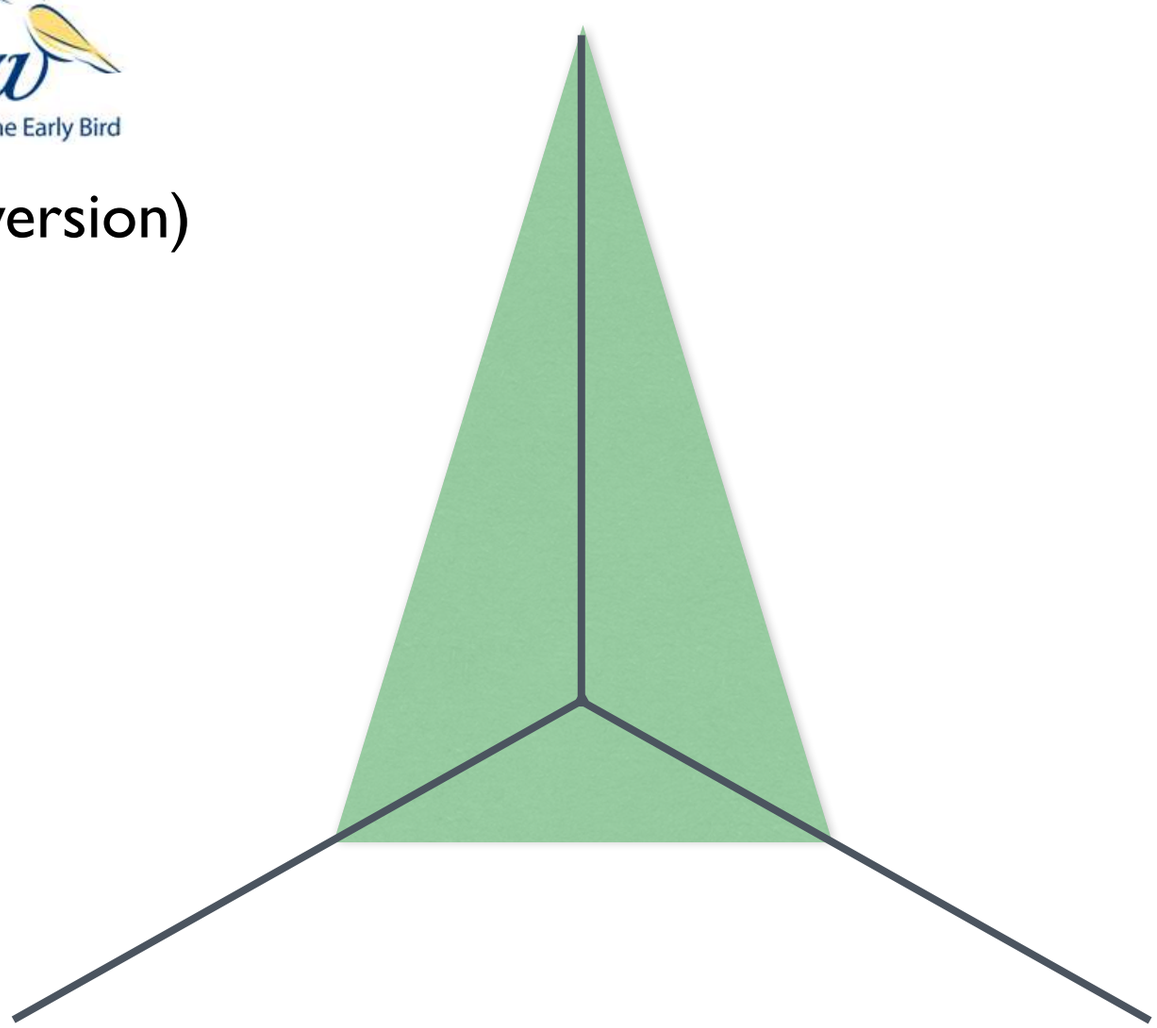
- Commercialized version of Sparrow The Early Bird
  - memory-bug-finding tool for full C
  - sound in design, unsound yet scalable in reality
- Realistic workbench available
  - “let’s try to achieve sound, precise, yet scalable version”

The Challenge in Reality

Soundness

Sparrow
The Early Bird

(sound-&-global version)



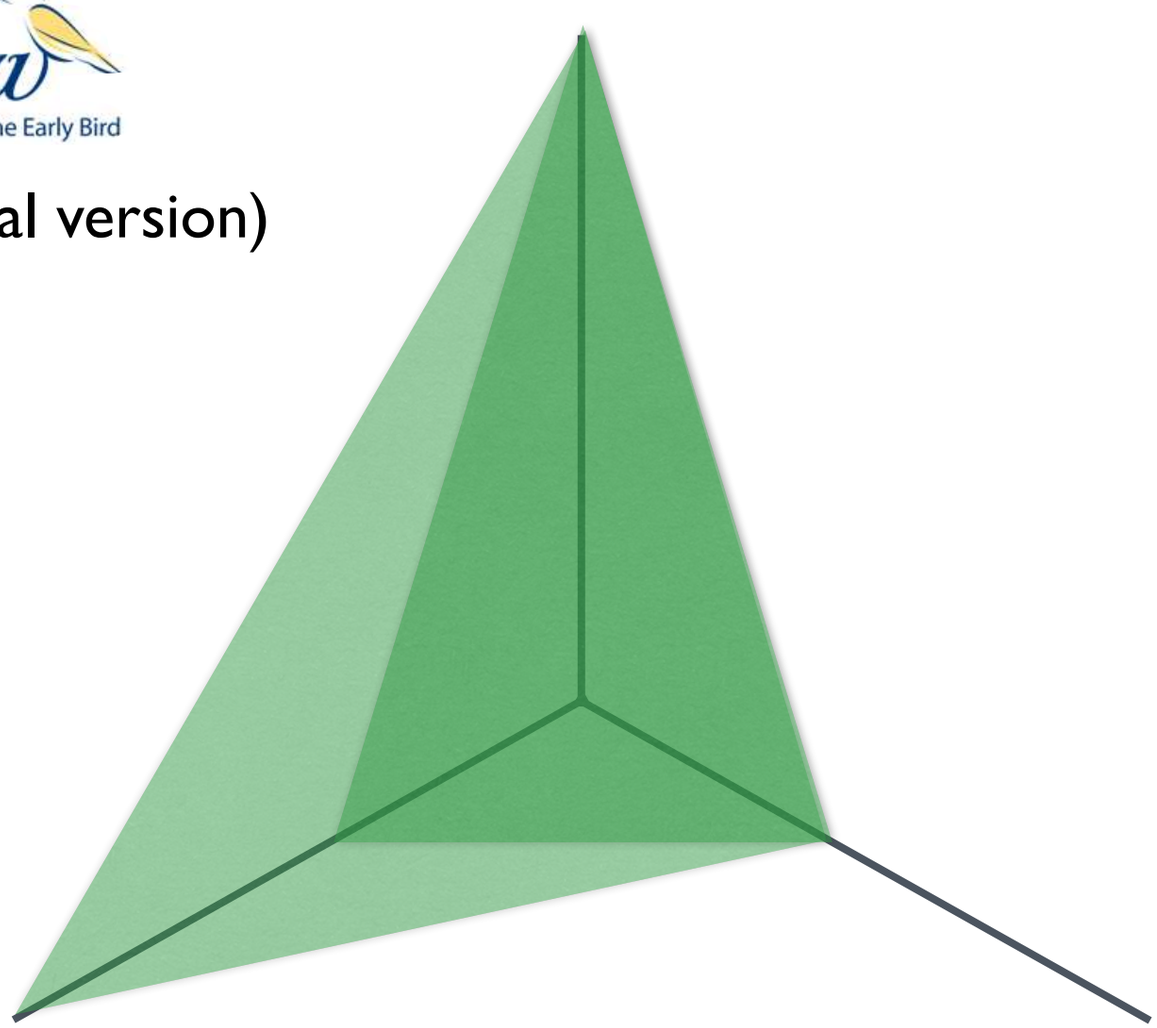
Scalability

Precision

The First Goal: Scalability

Sparrow
The Early Bird

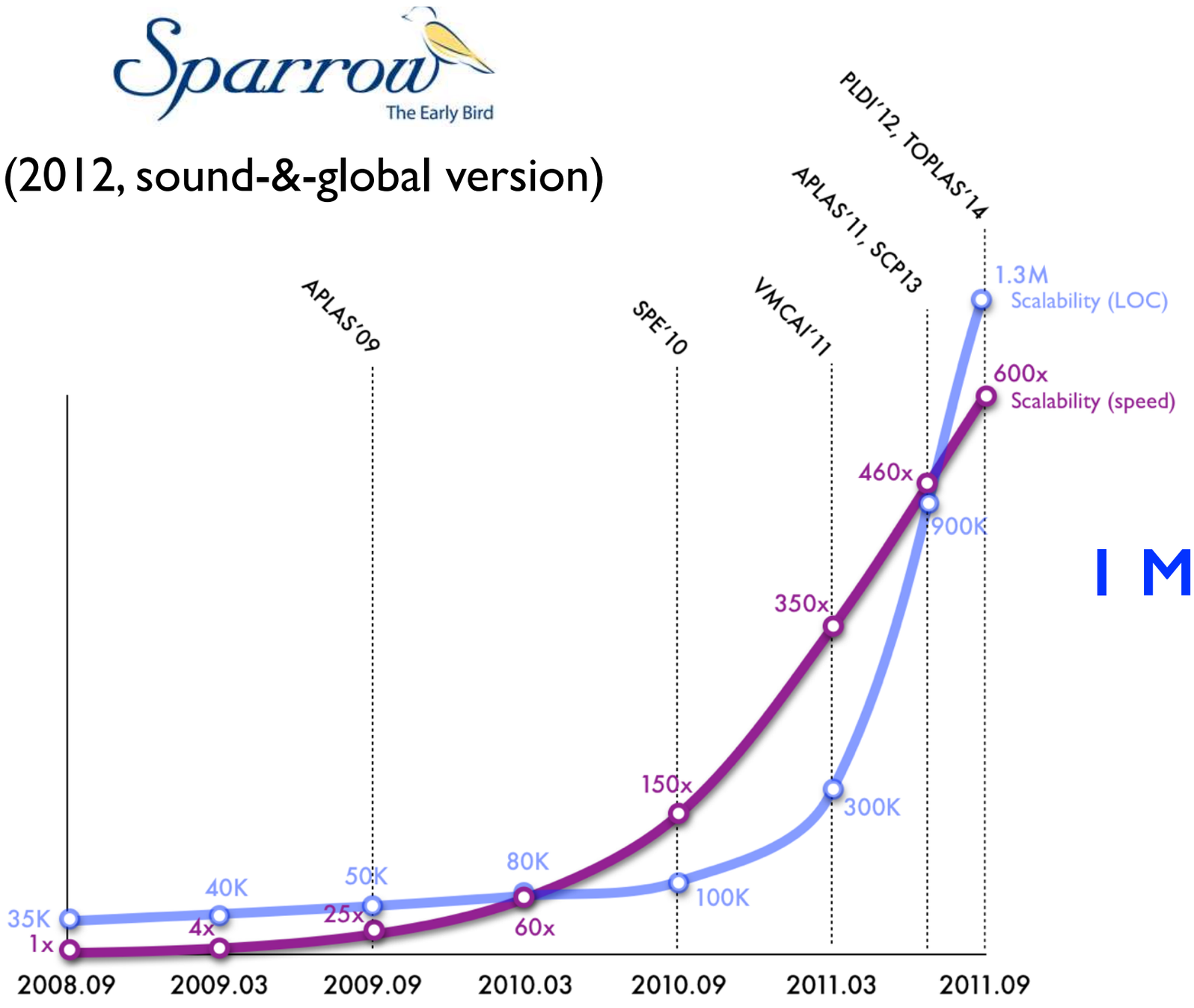
(2012, sound-&-global version)



Scalability Improvement

Sparrow
The Early Bird

(2012, sound-&-global version)

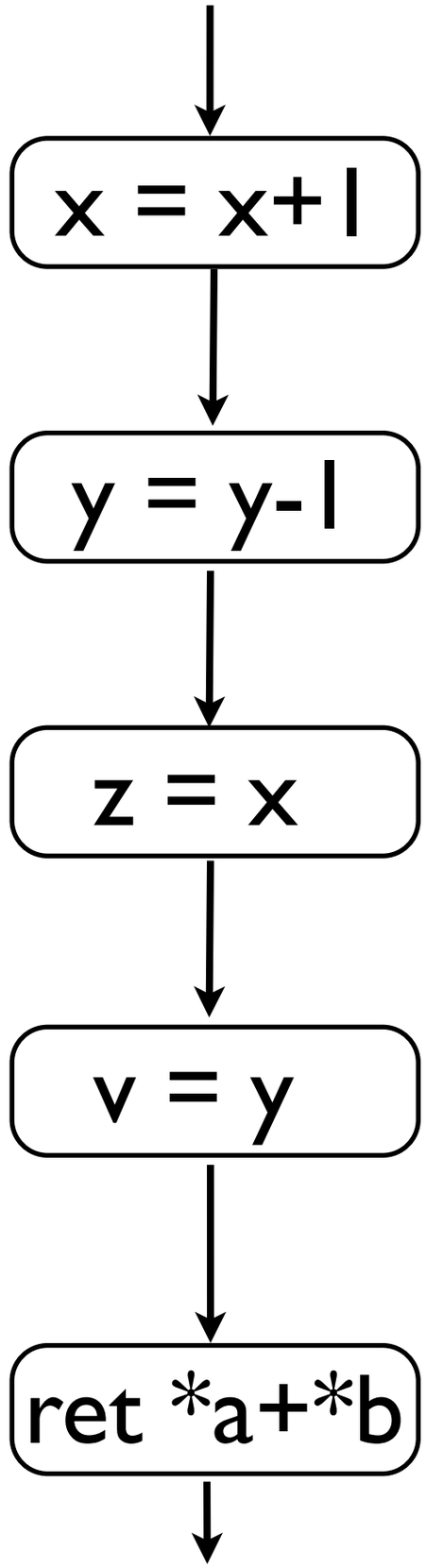


1 Million LoC in 10 hrs

PLDI'12

Key: General Sparse Analysis

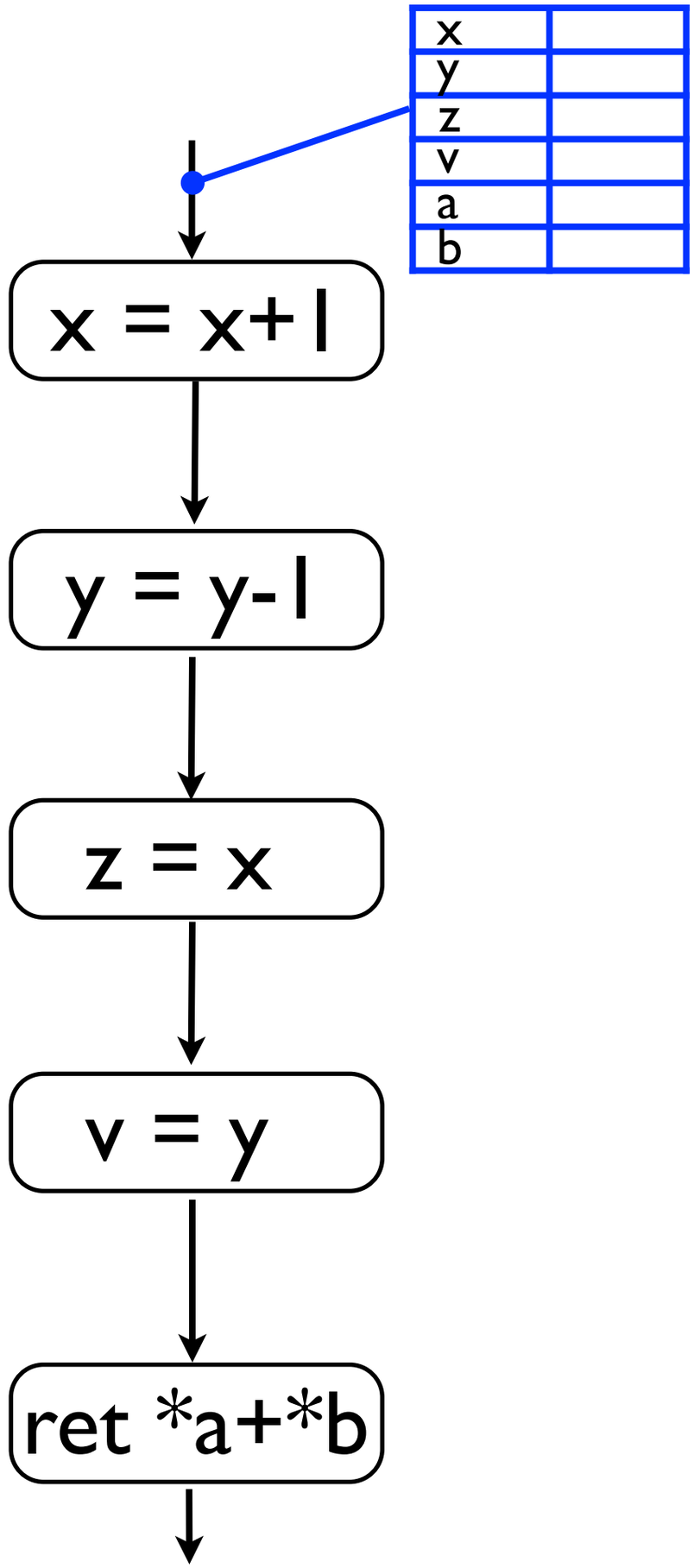
"Right Part at Right Moment"



PLDI'12

Key: General Sparse Analysis

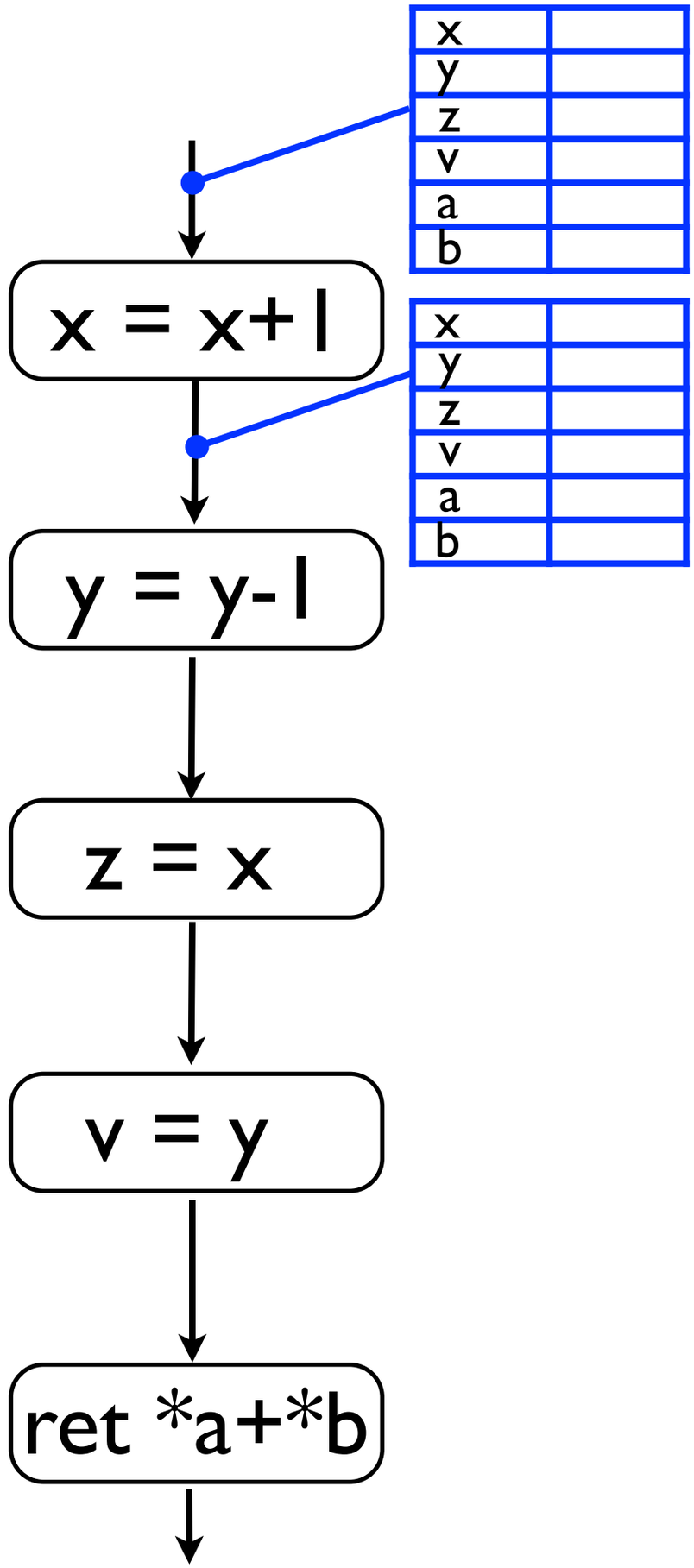
"Right Part at Right Moment"



PLDI'12

Key: General Sparse Analysis

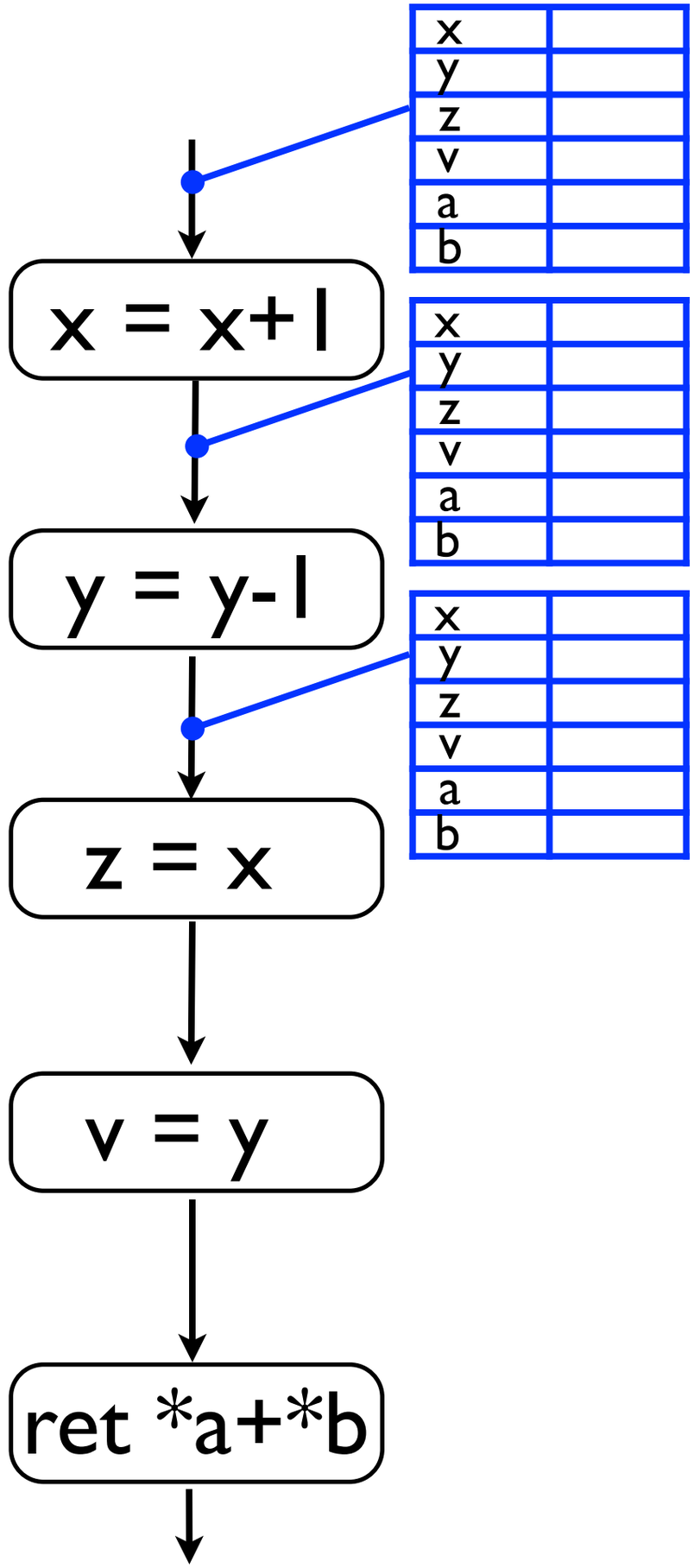
"Right Part at Right Moment"



PLDI'12

Key: General Sparse Analysis

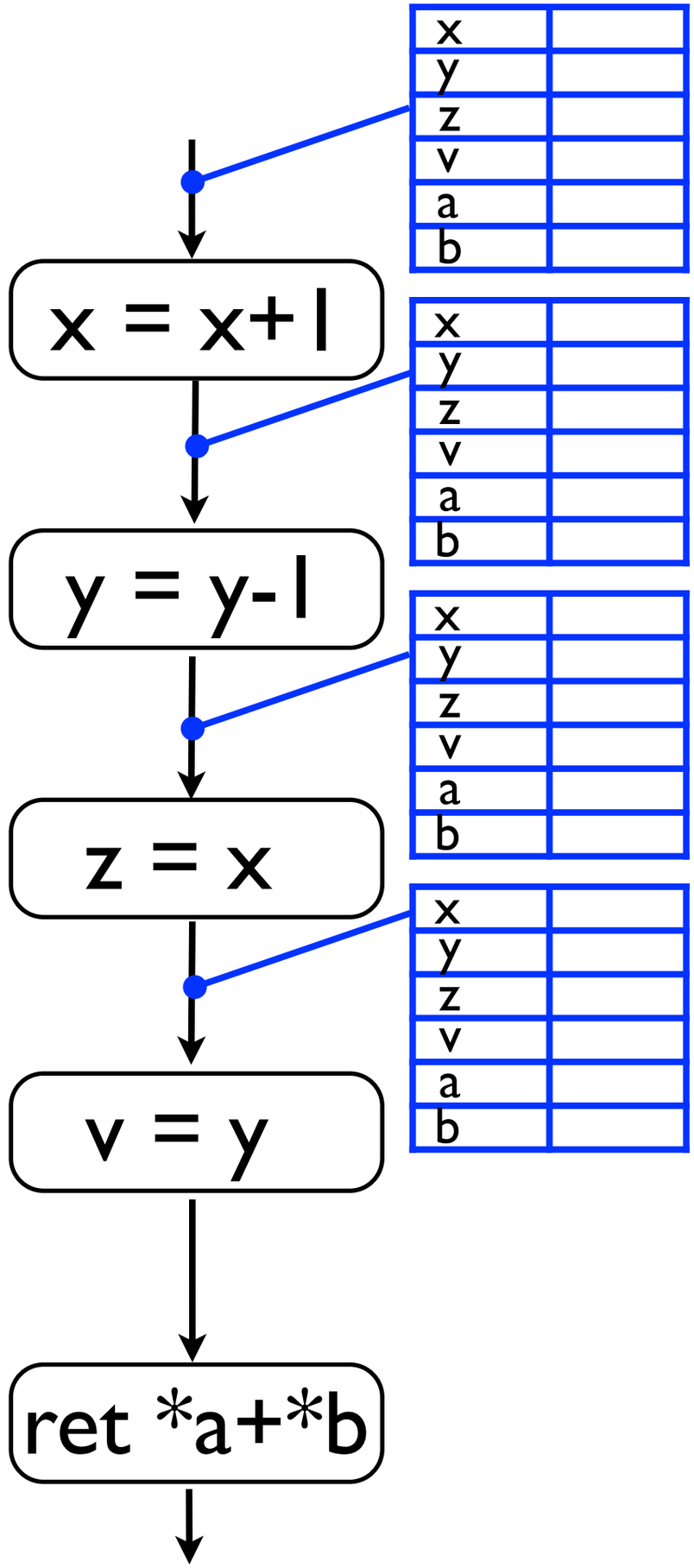
"Right Part at Right Moment"



PLDI'12

Key: General Sparse Analysis

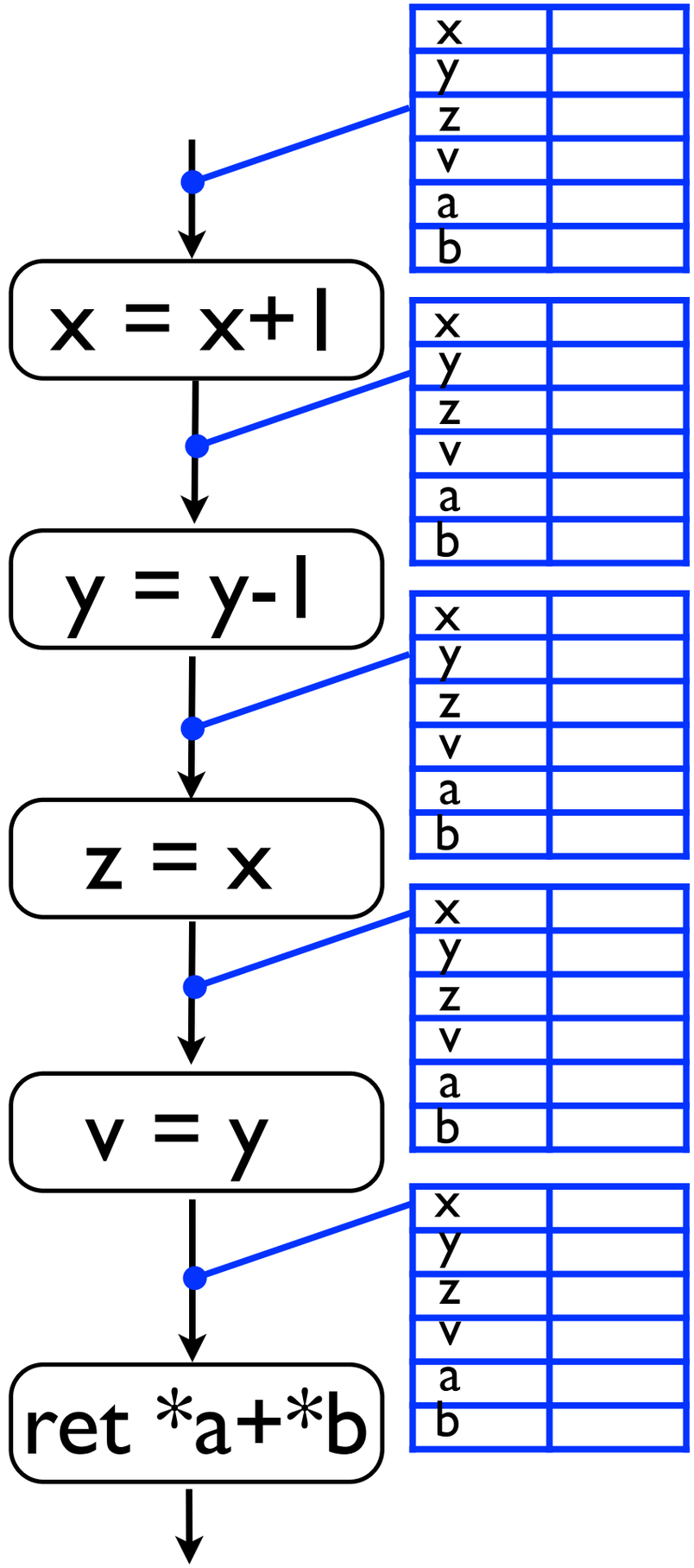
"Right Part at Right Moment"



PLDI'12

Key: General Sparse Analysis

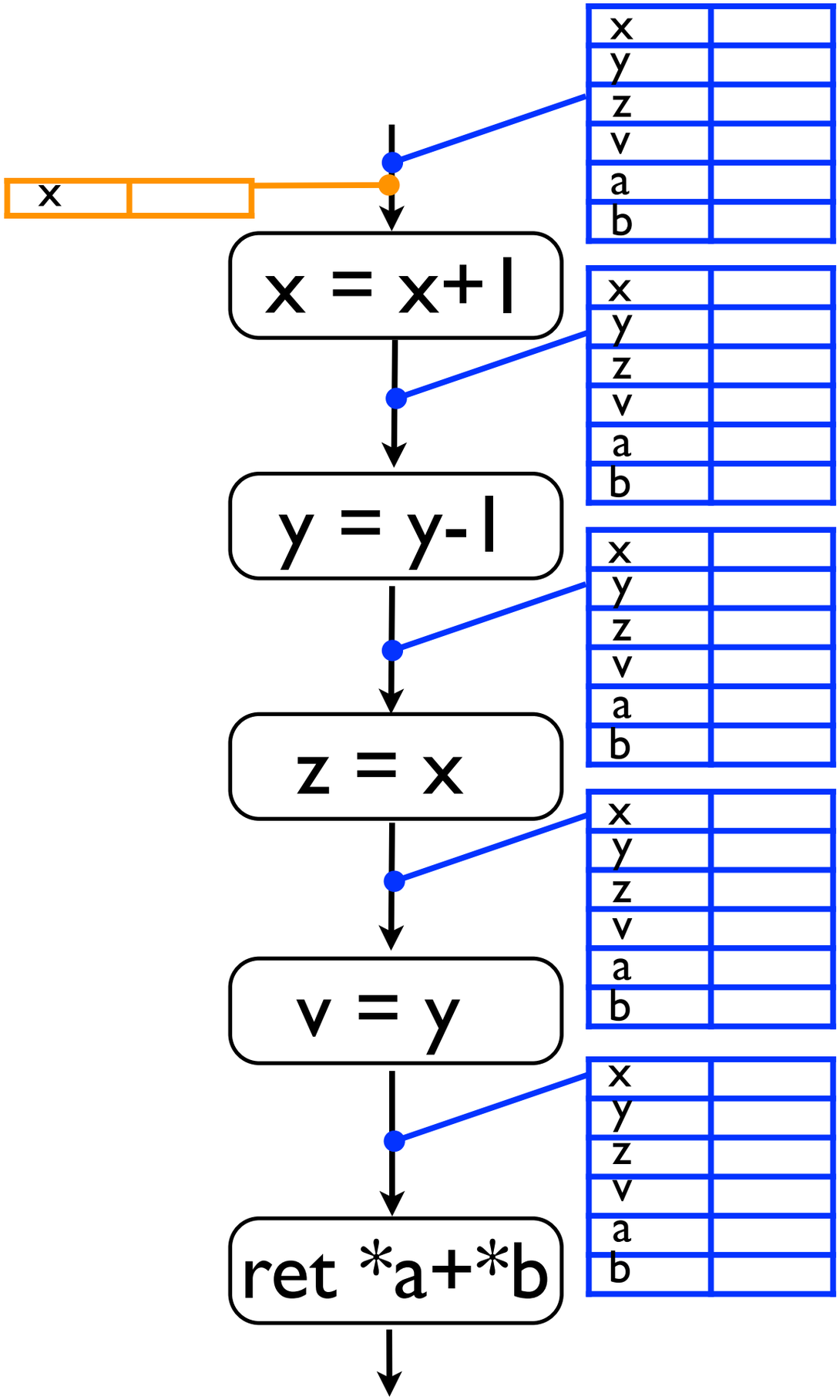
"Right Part at Right Moment"



PLDI'12

Key: General Sparse Analysis

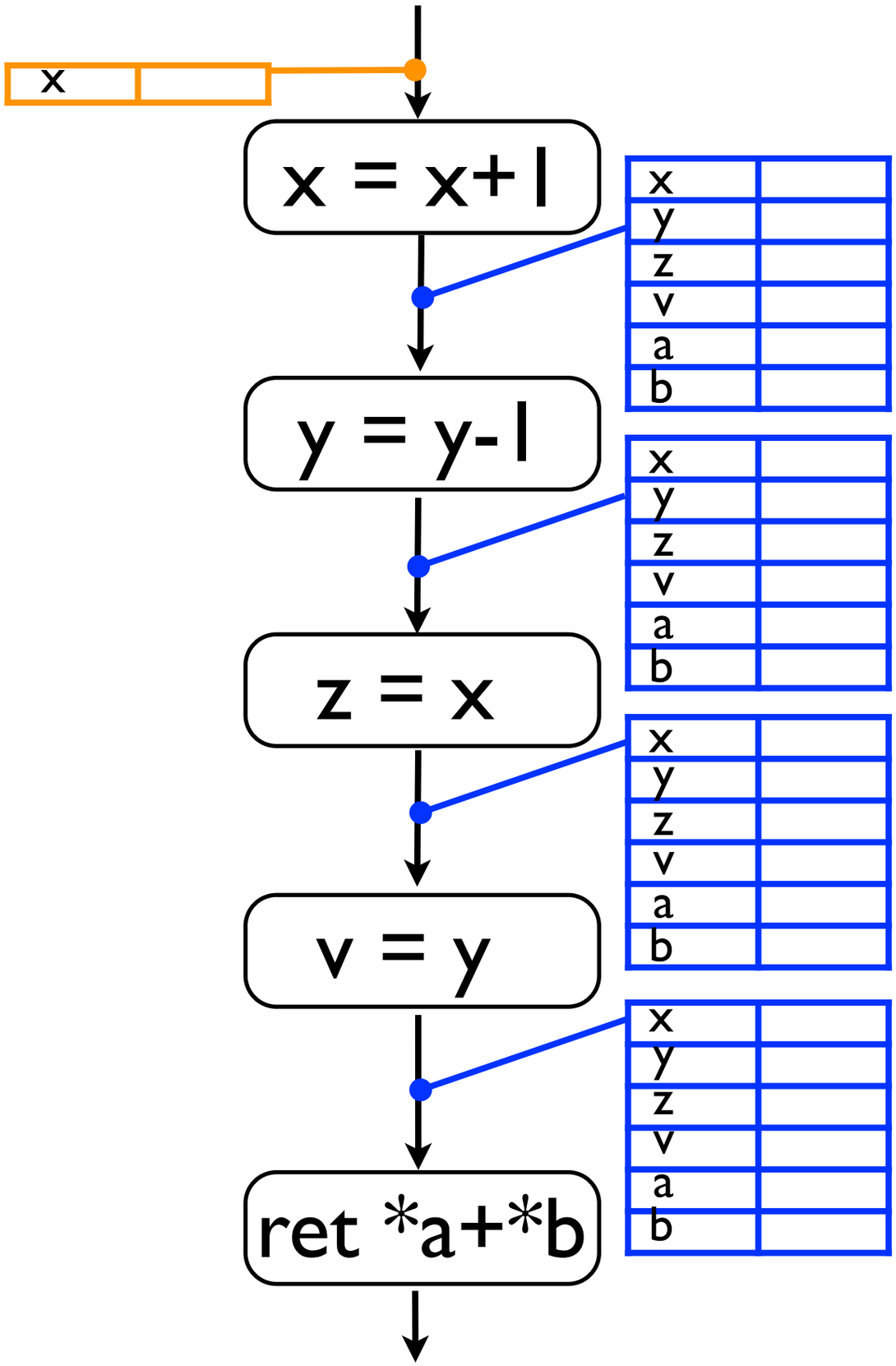
"Right Part at Right Moment"



PLDI'12

Key: General Sparse Analysis

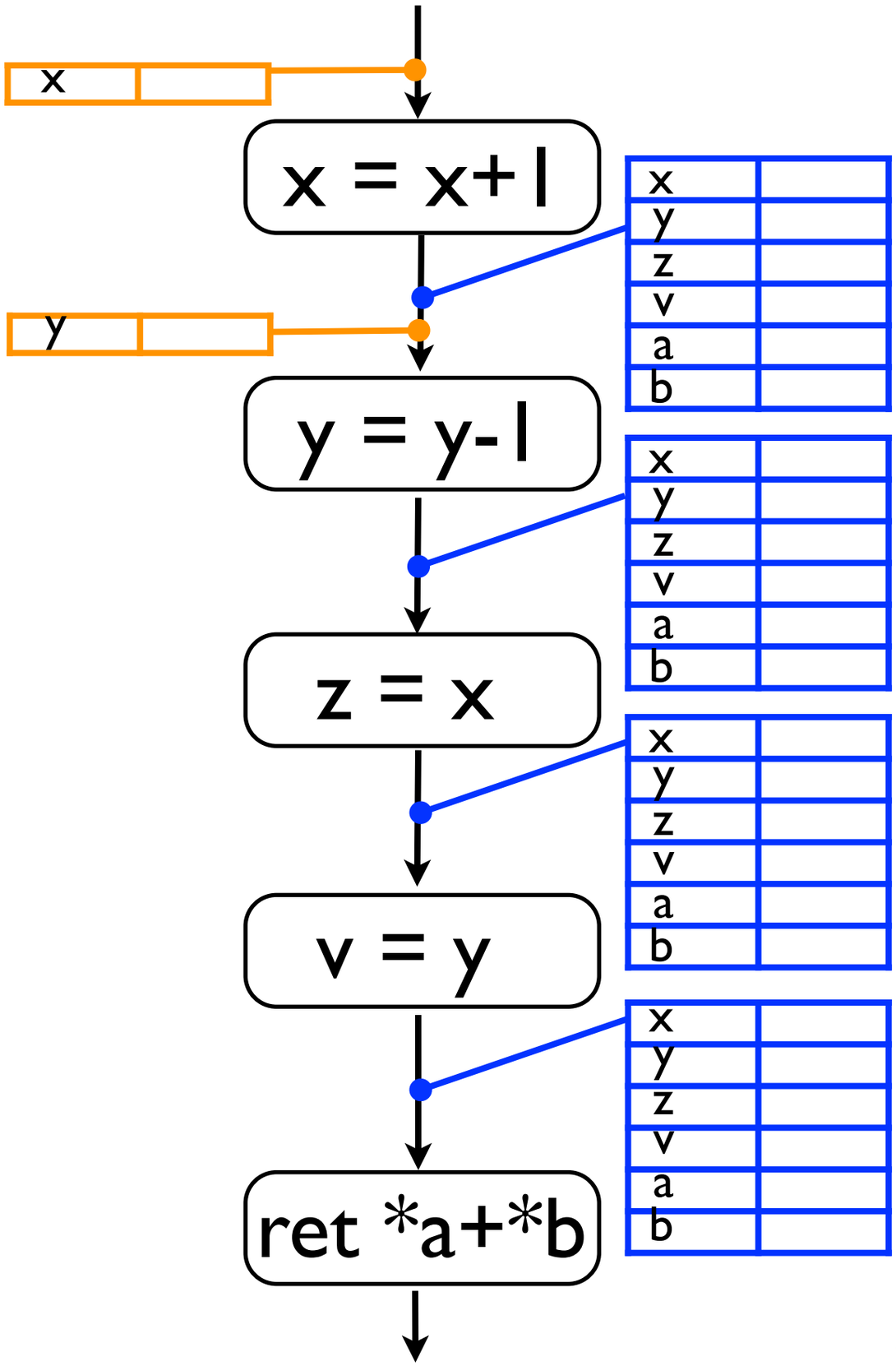
"Right Part at Right Moment"



PLDI'12

Key: General Sparse Analysis

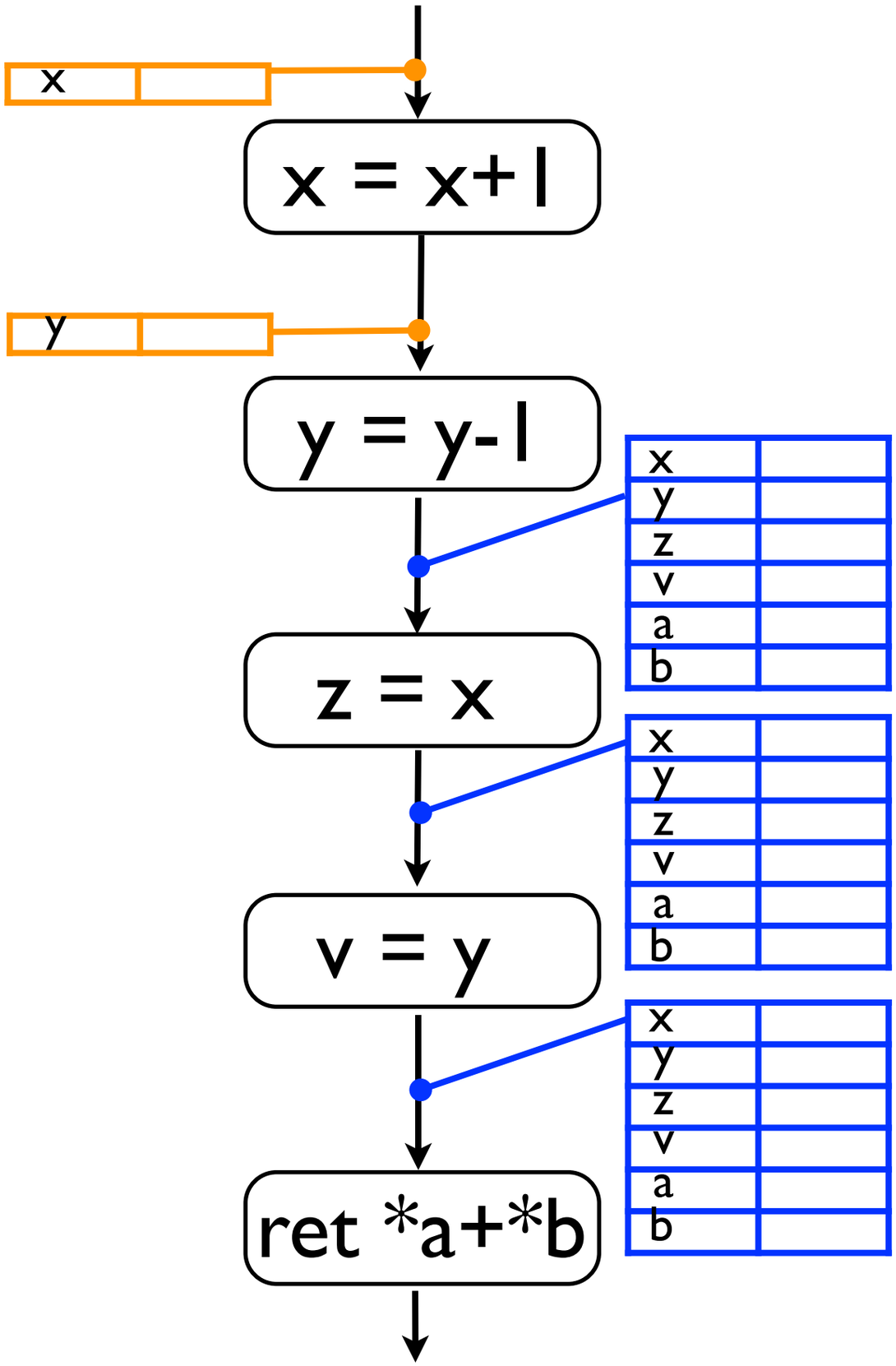
"Right Part at Right Moment"



PLDI'12

Key: General Sparse Analysis

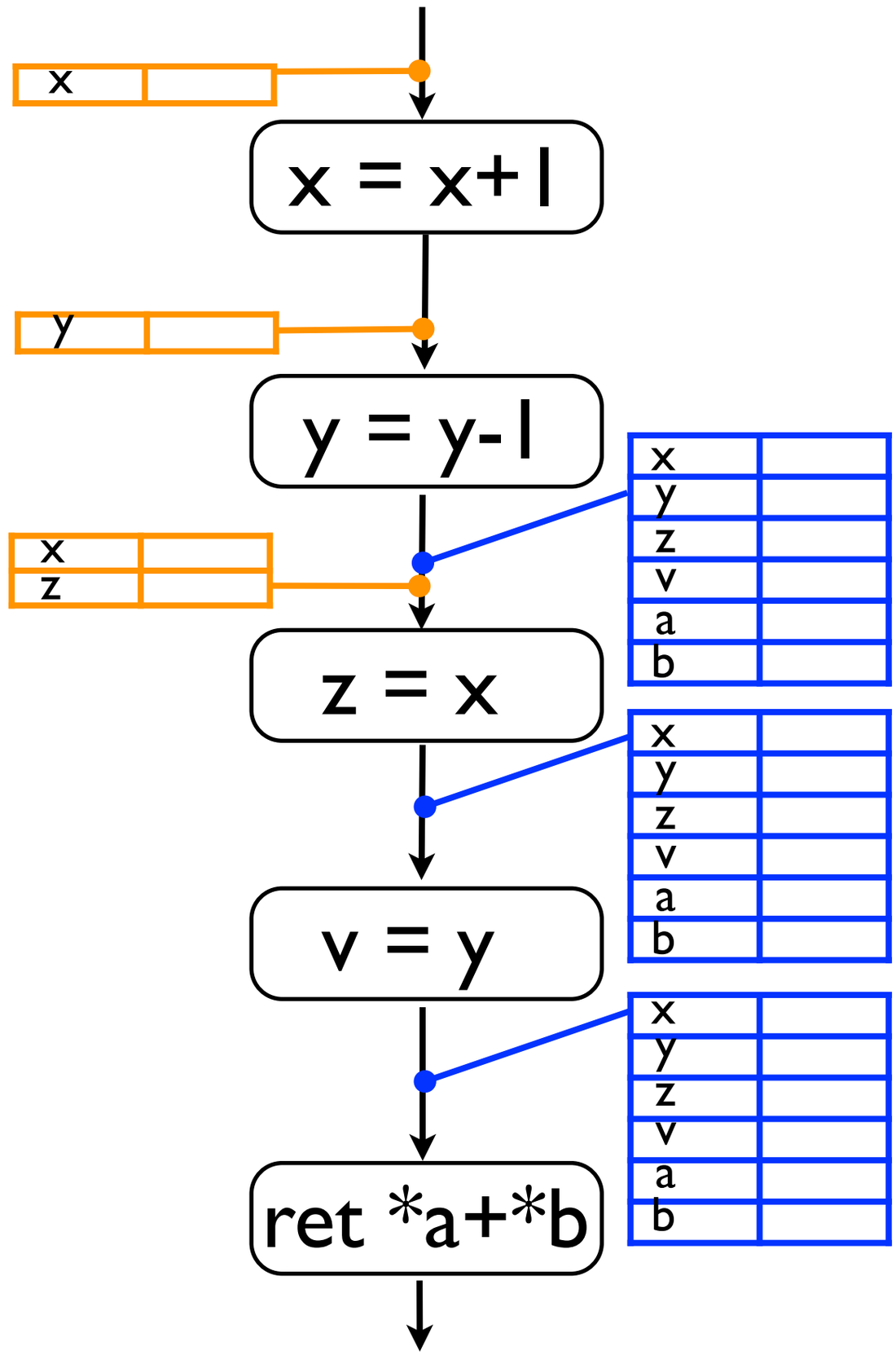
"Right Part at Right Moment"



PLDI'12

Key: General Sparse Analysis

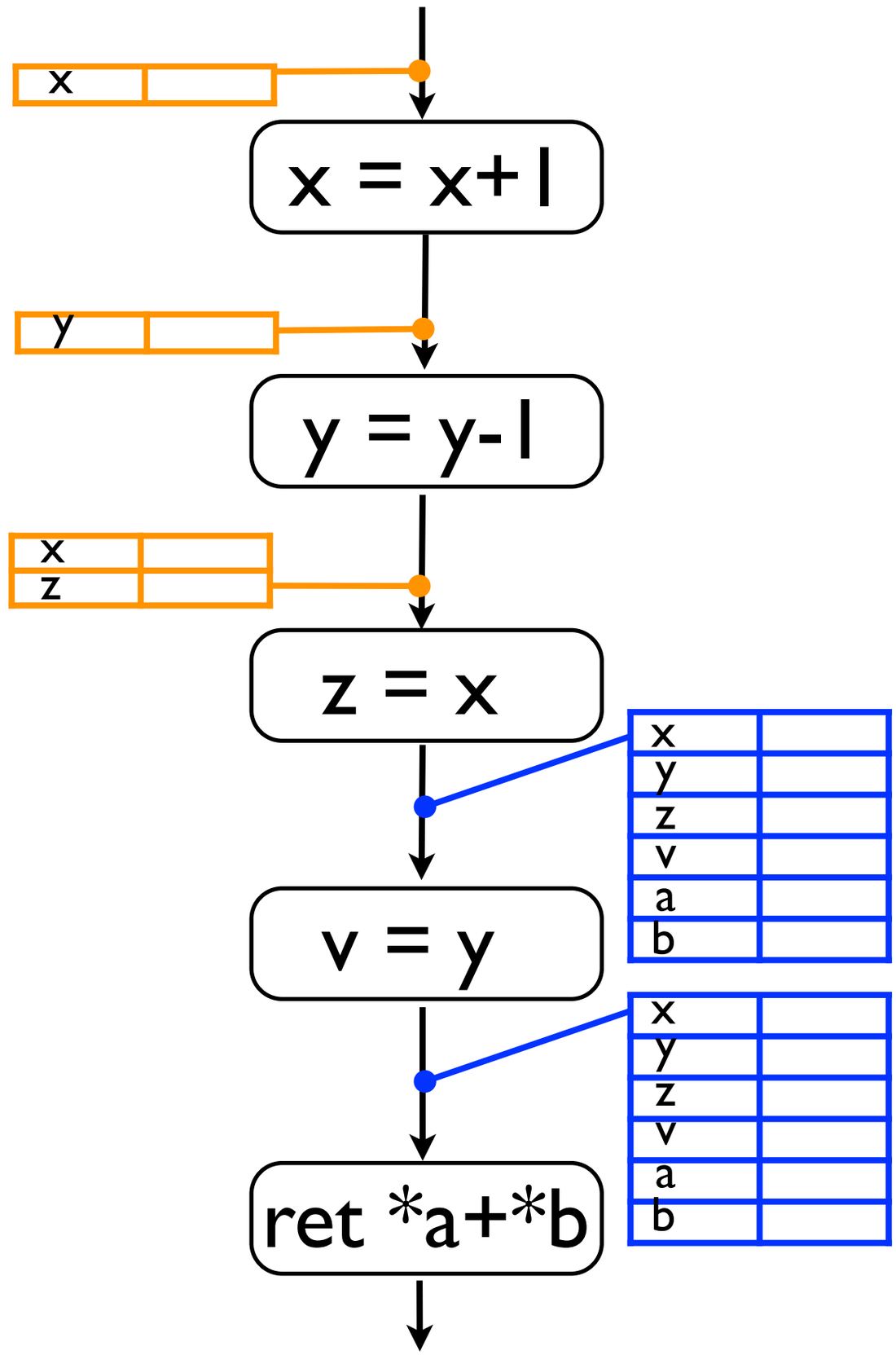
"Right Part at Right Moment"



PLDI'12

Key: General Sparse Analysis

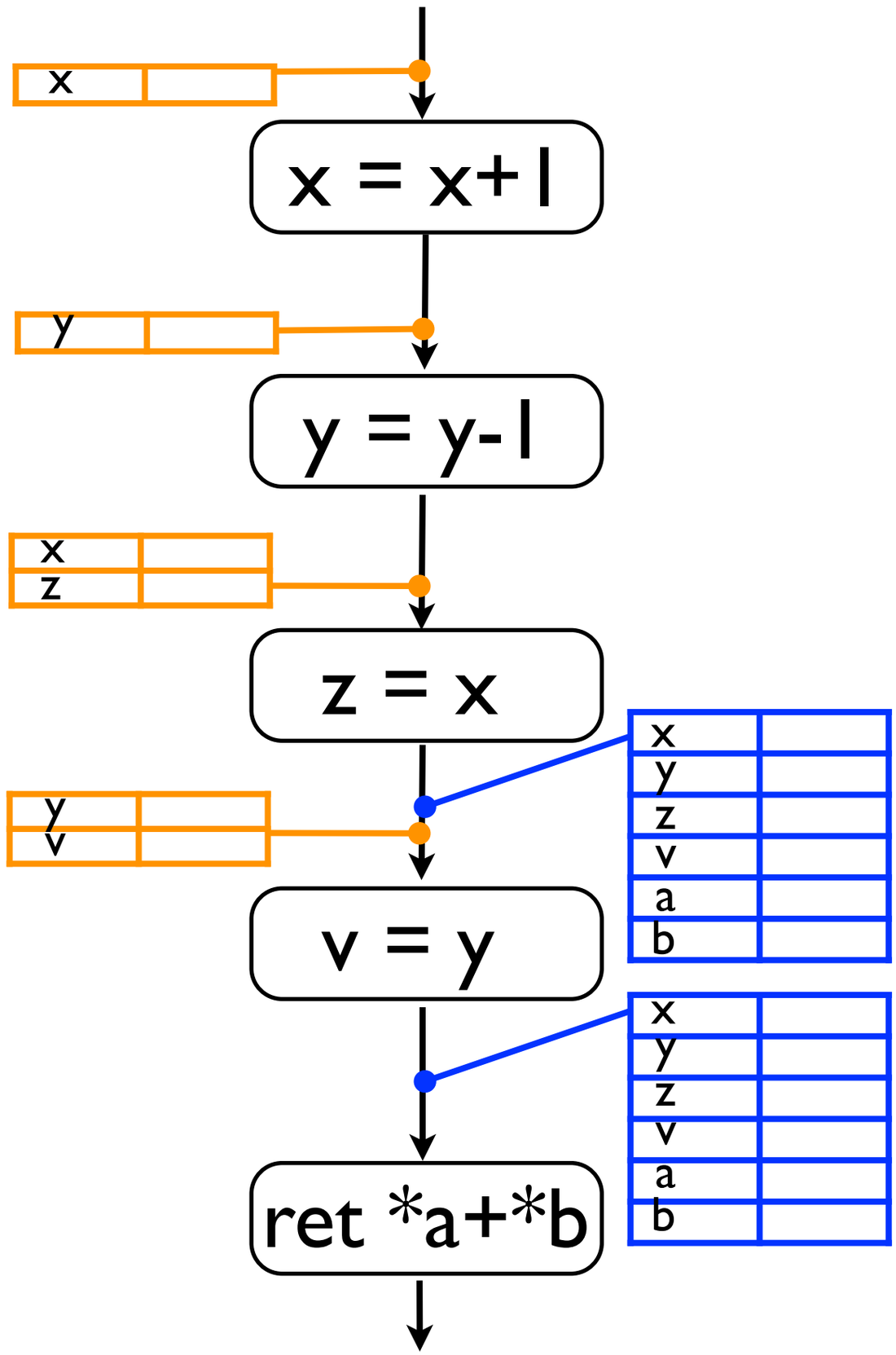
"Right Part at Right Moment"



PLDI'12

Key: General Sparse Analysis

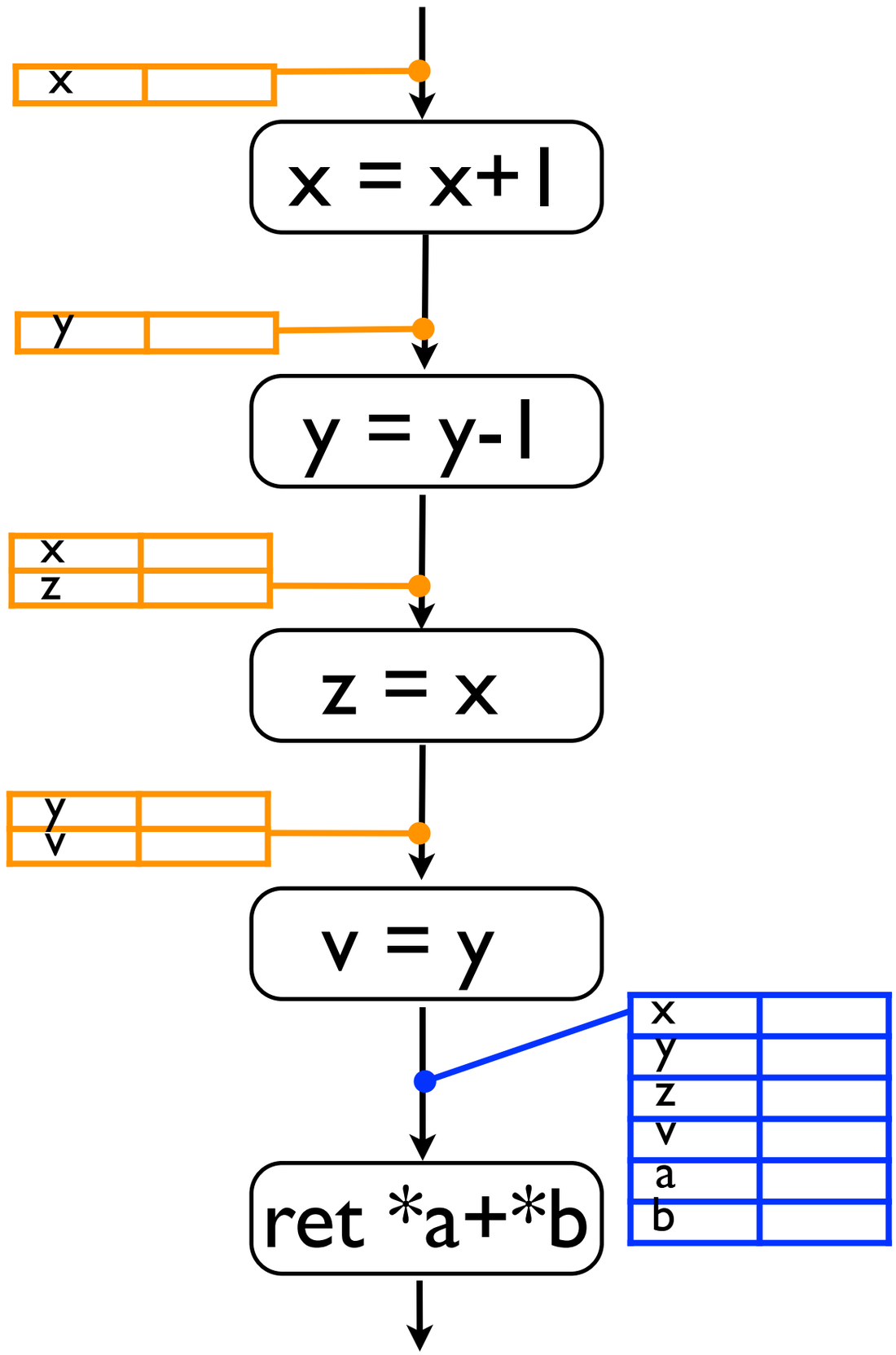
"Right Part at Right Moment"



PLDI'12

Key: General Sparse Analysis

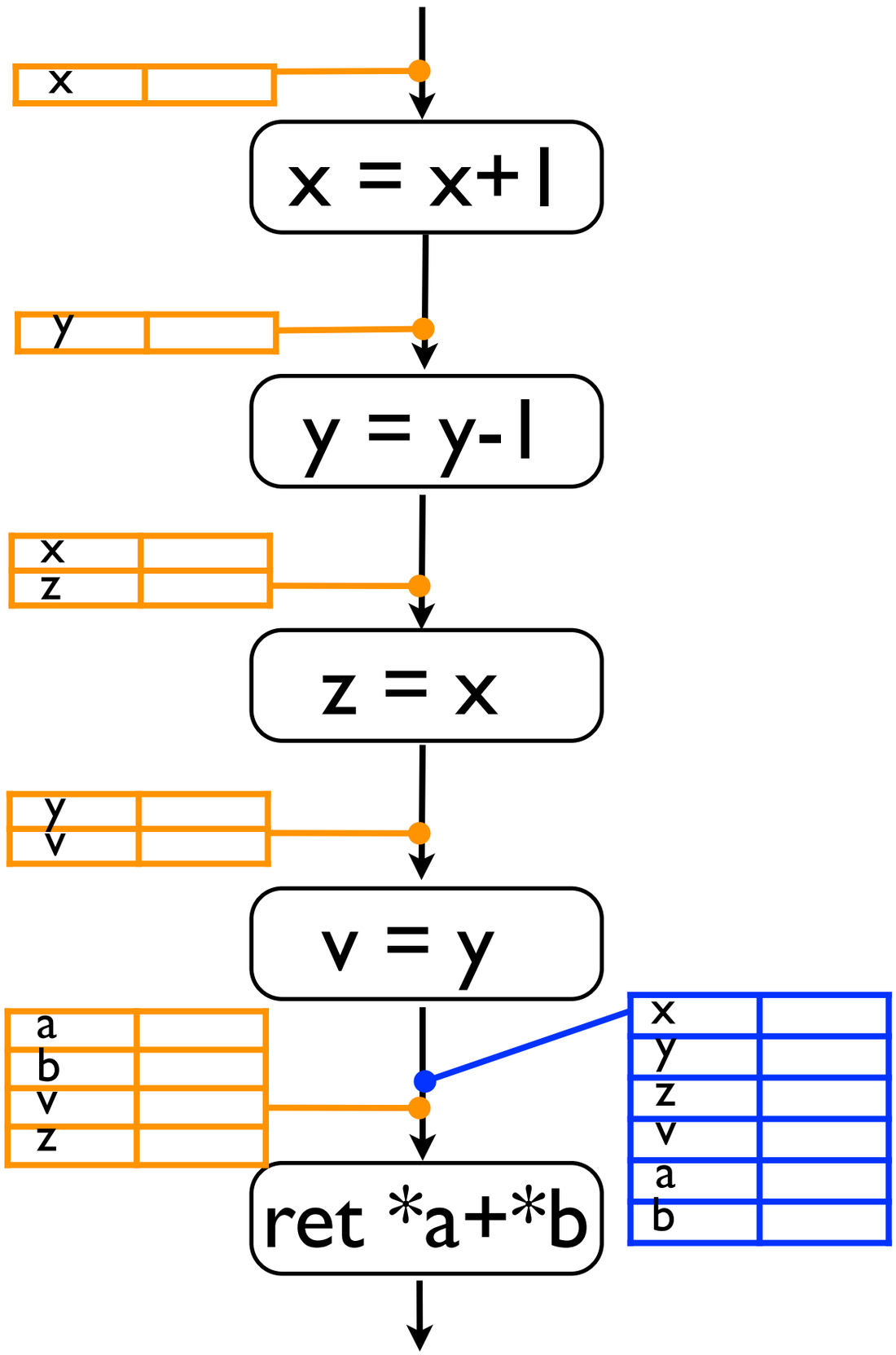
"Right Part at Right Moment"



PLDI'12

Key: General Sparse Analysis

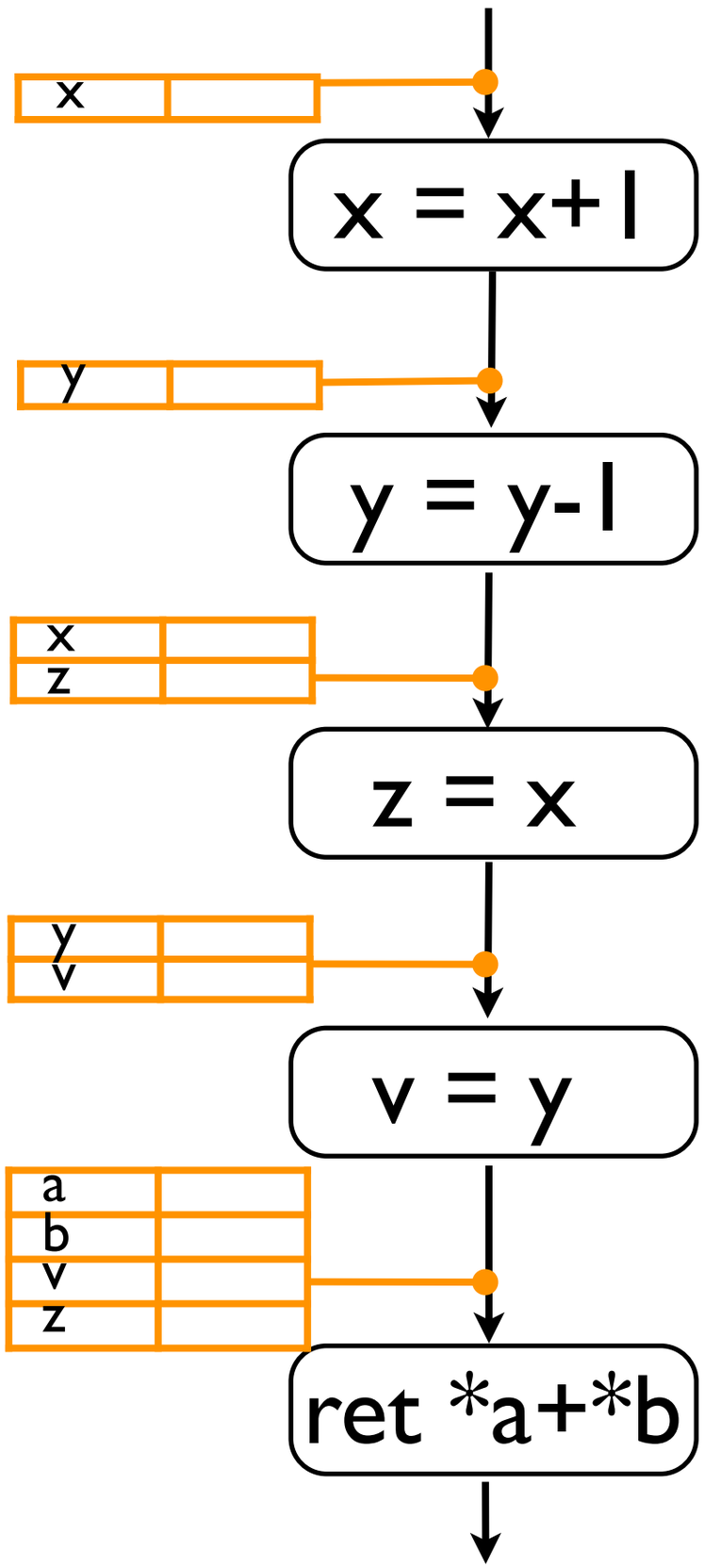
"Right Part at Right Moment"



PLDI'12

Key: General Sparse Analysis

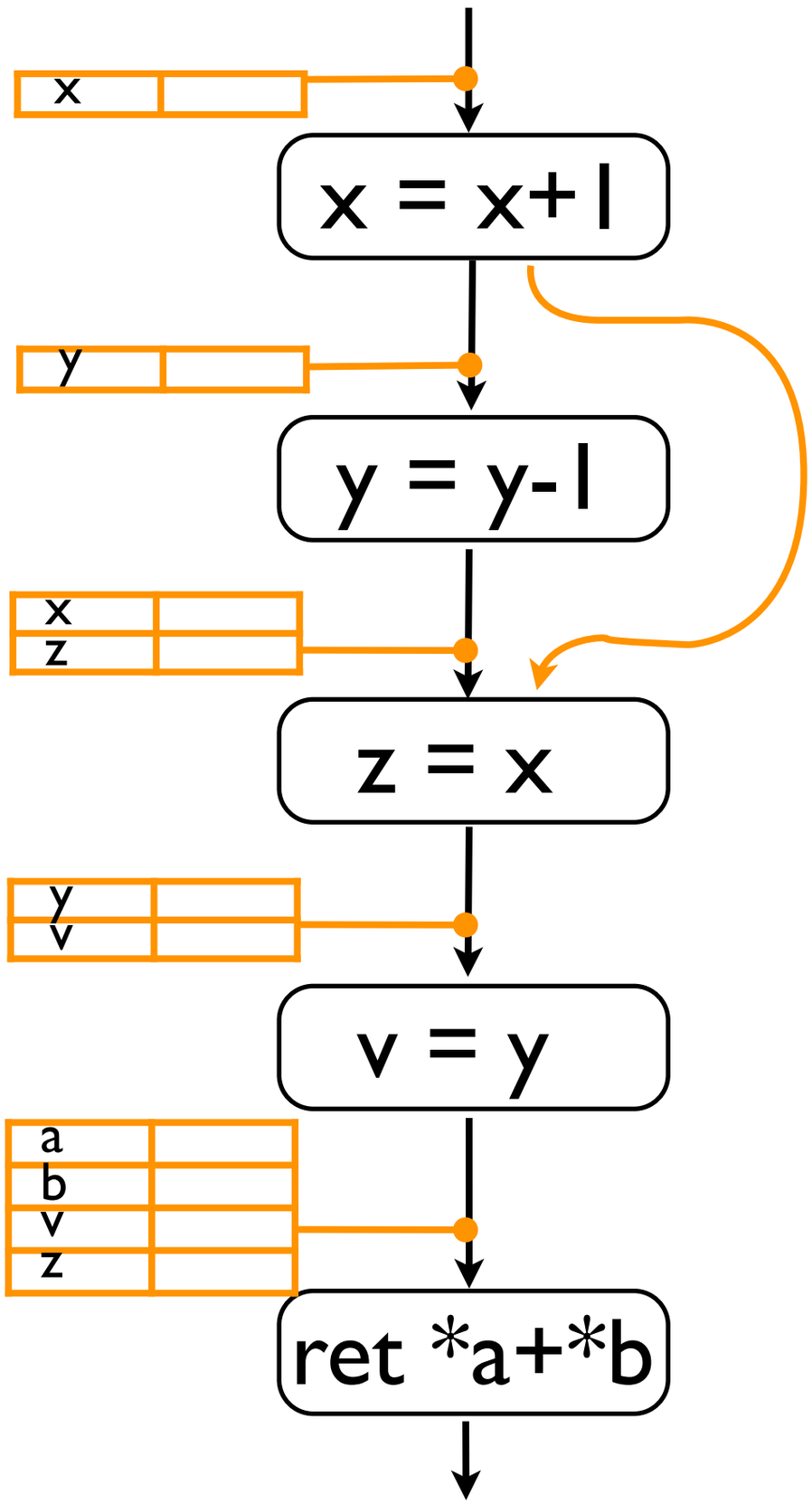
"Right Part at Right Moment"



PLDI'12

Key: General Sparse Analysis

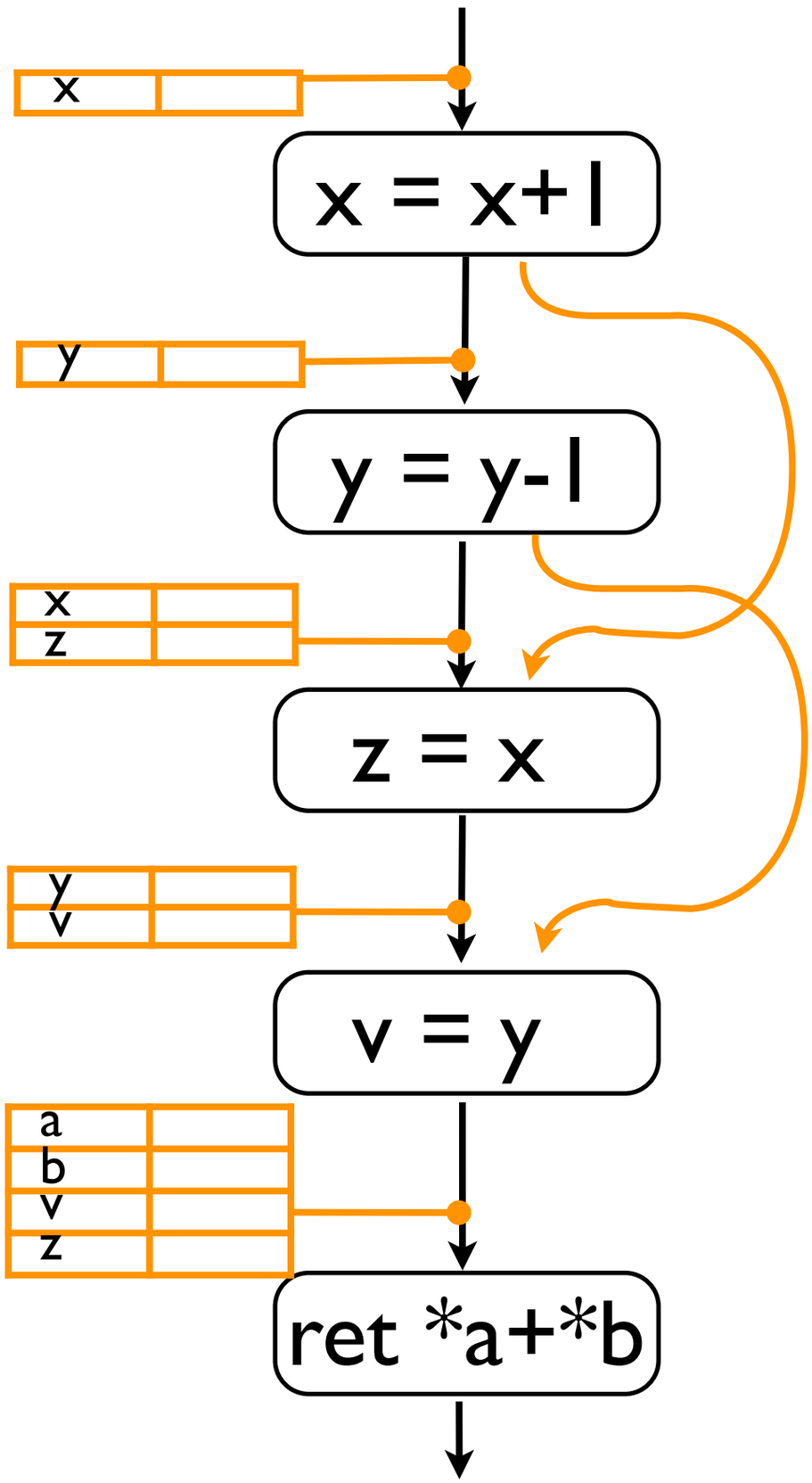
"Right Part at Right Moment"



PLDI'12

Key: General Sparse Analysis

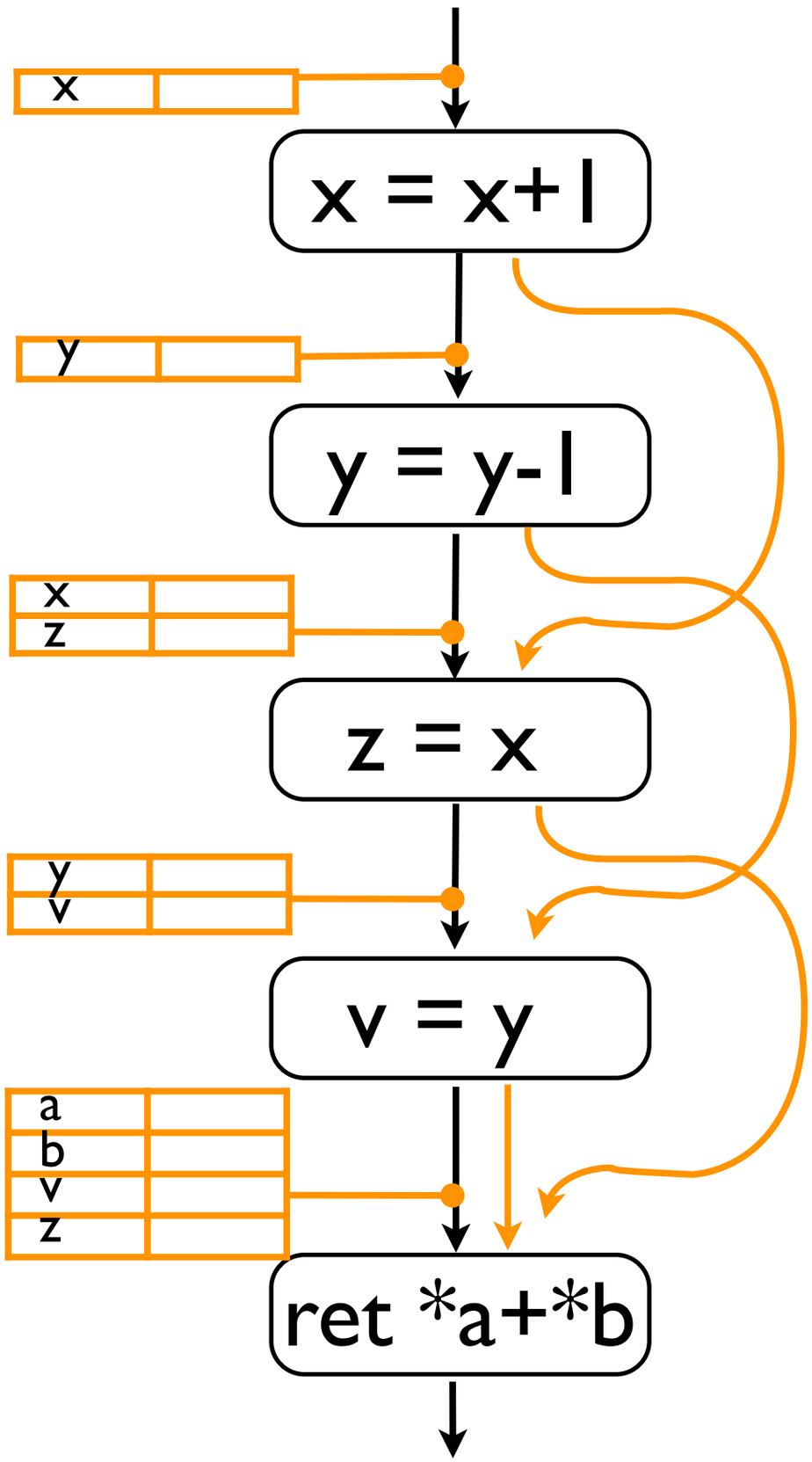
"Right Part at Right Moment"



PLDI'12

Key: General Sparse Analysis

"Right Part at Right Moment"



PLDI'12

General Sparse Analysis Framework

Theorem. (preservation of soundness and precision)

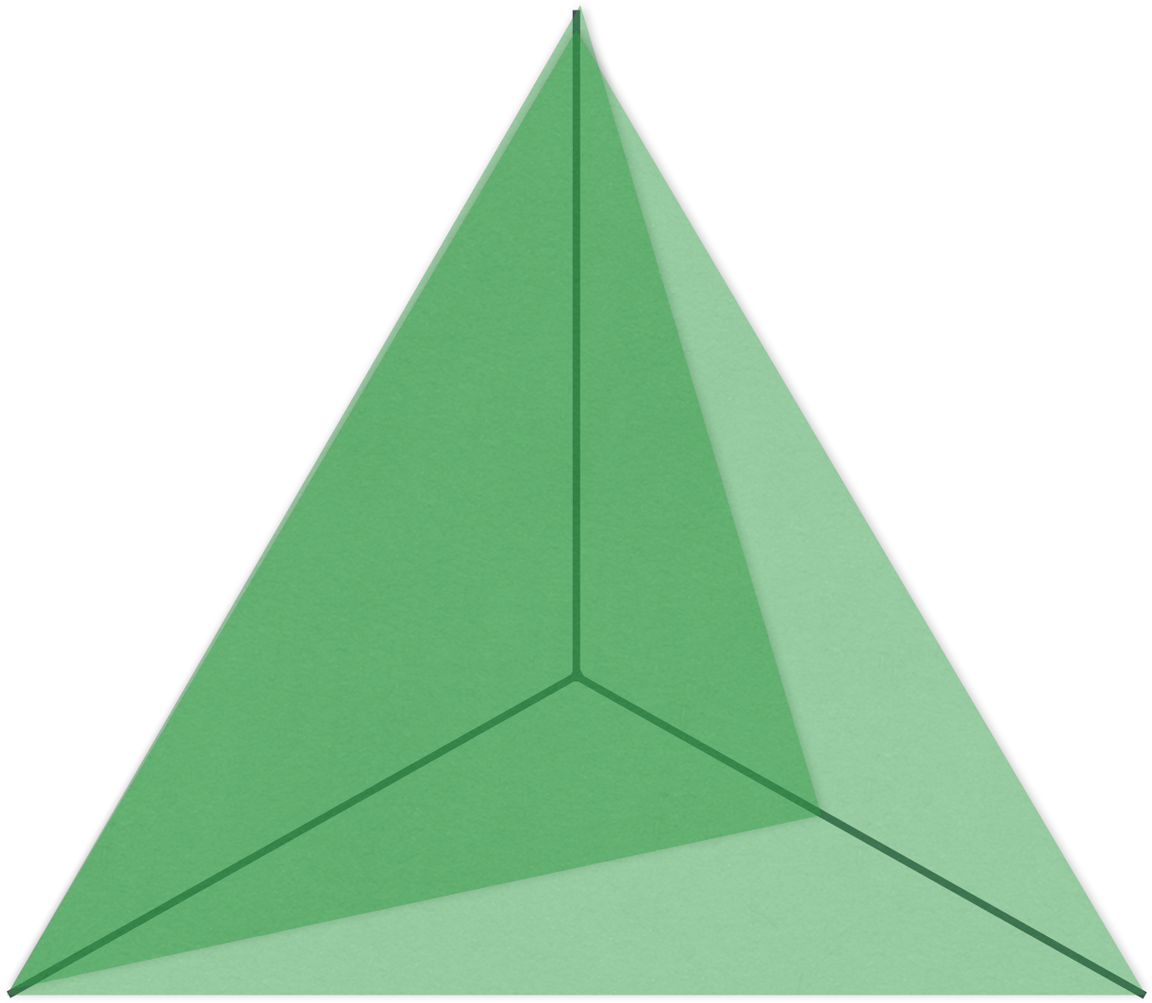
$$\hat{F} : \hat{D} \to \hat{D} \quad \xRightarrow{\text{sparsify}} \quad \hat{F}_s : \hat{D} \to \hat{D}$$

$$fix\,\hat{F} \quad = \quad fix\,\hat{F}_s$$

*"An important strength is that the **theoretical result** is **very general** ... The result should be **highly influential** on future work in sparse analysis."* *(from PLDI reviews)*

The Second Goal: Precision

Soundness

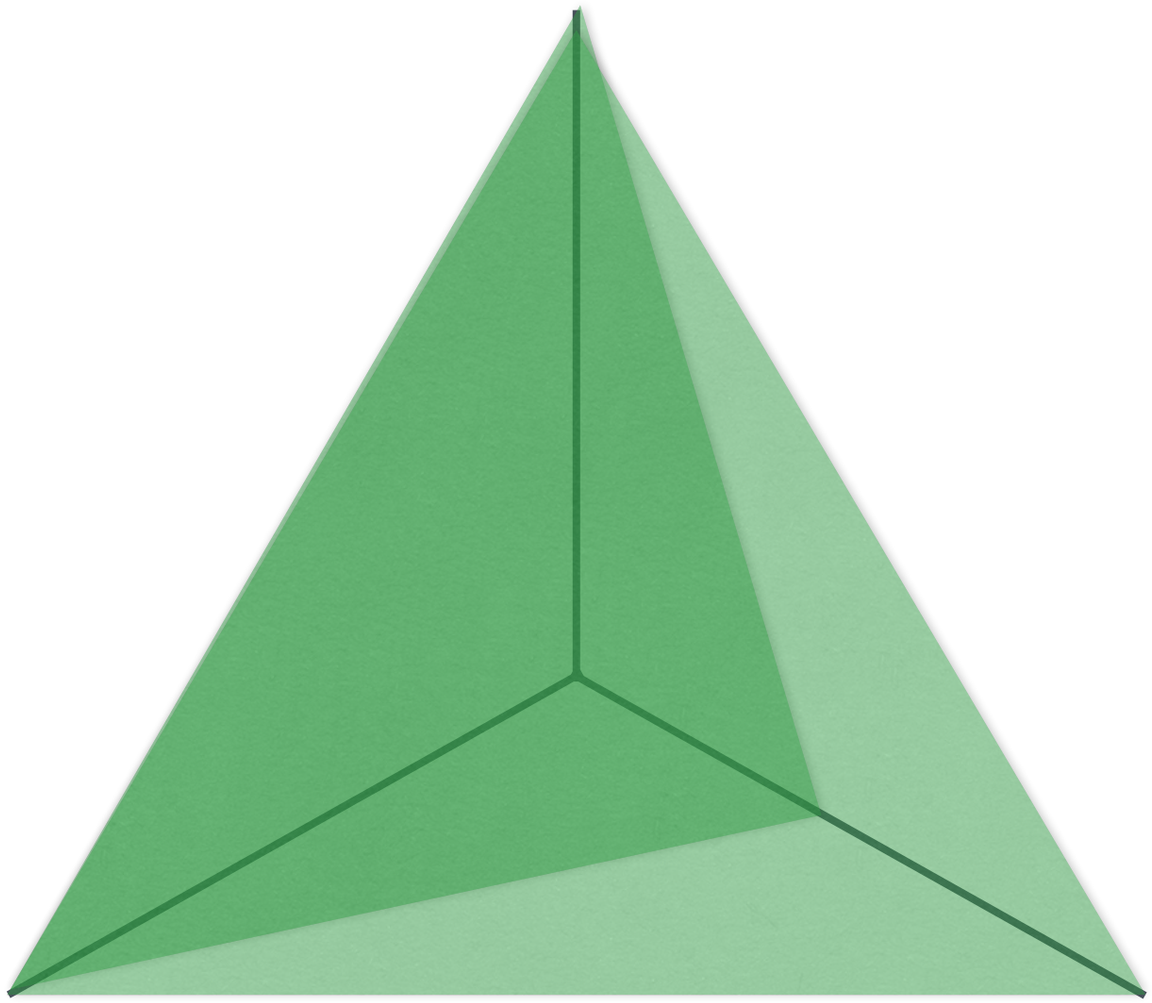


Scalability

Precision

The Second Goal: Precision

Soundness



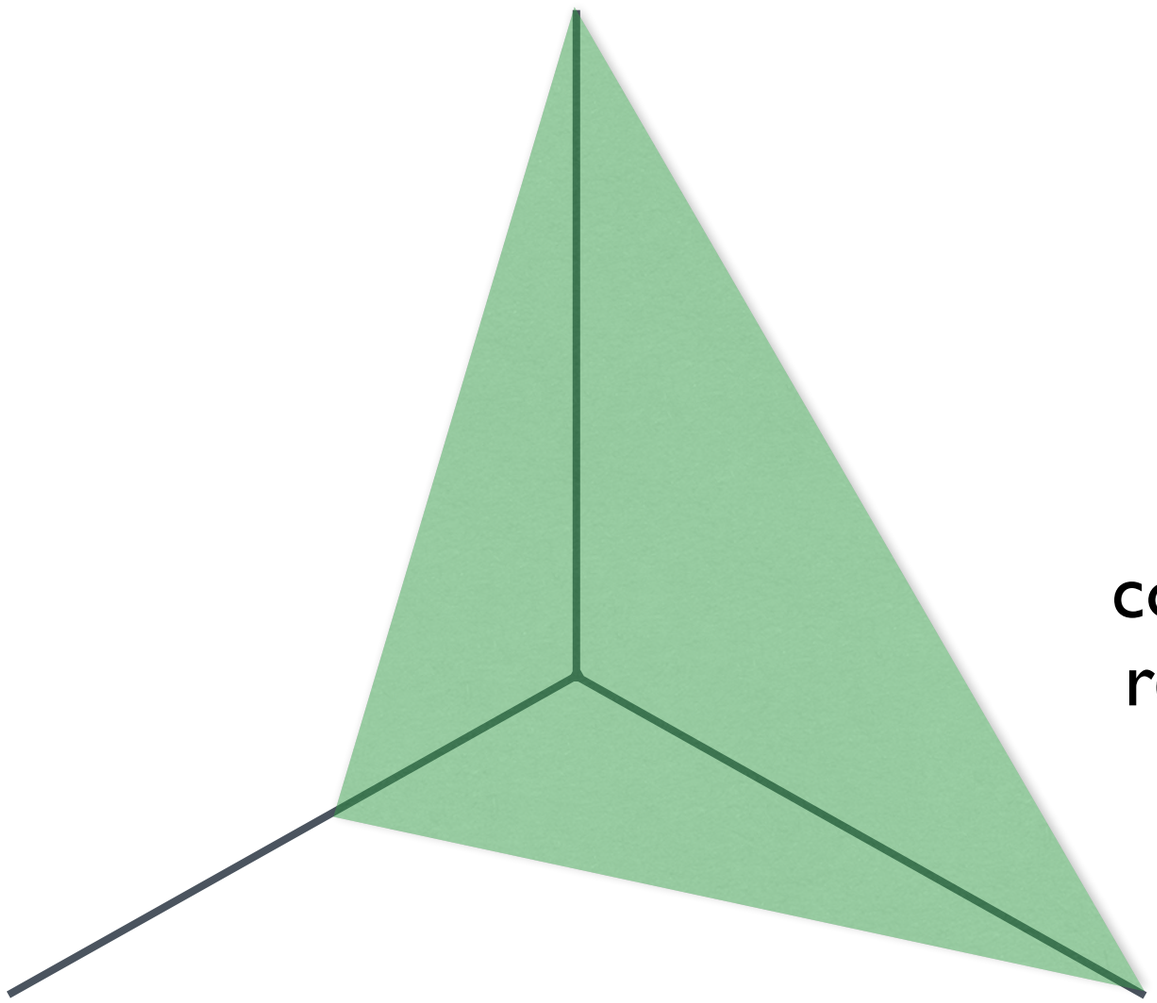
Scalability

Precision

Challenge: Can we achieve it without scalability loss?

cf) Existing Techniques

Soundness



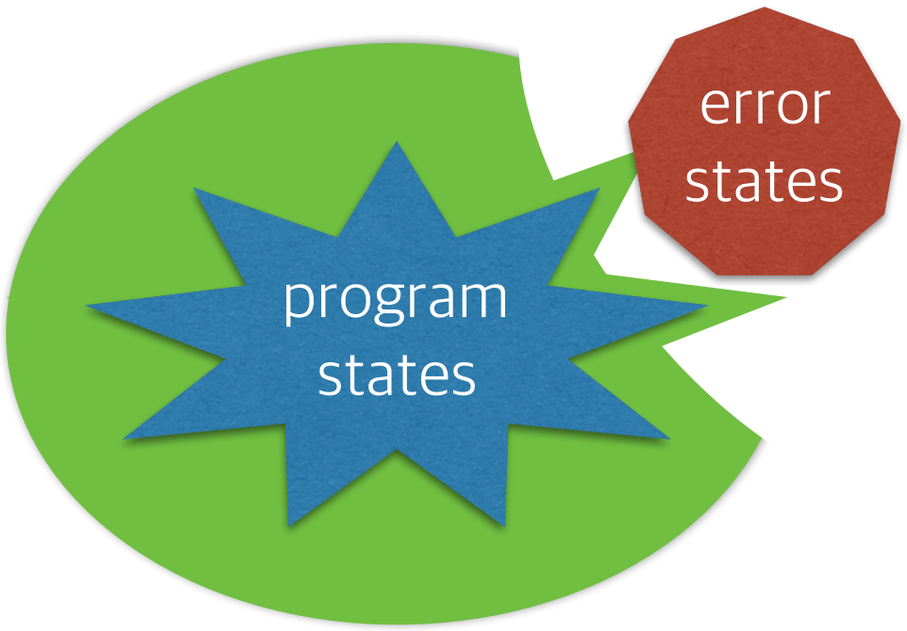
context-sensitivity,
relational analysis,
etc

Scalability

Precision

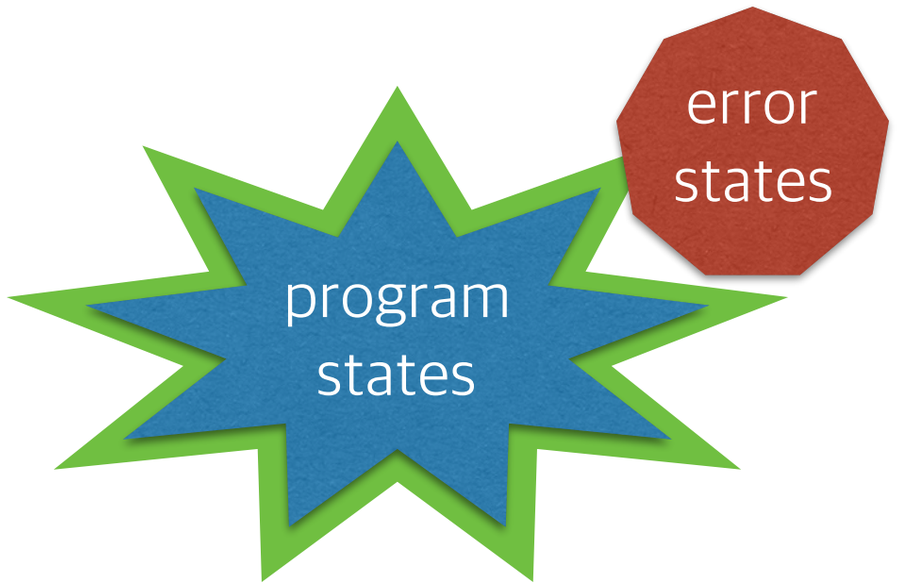
Selective X-Sensitivity Approach

- **Key Idea**: Improve precision only when it matters



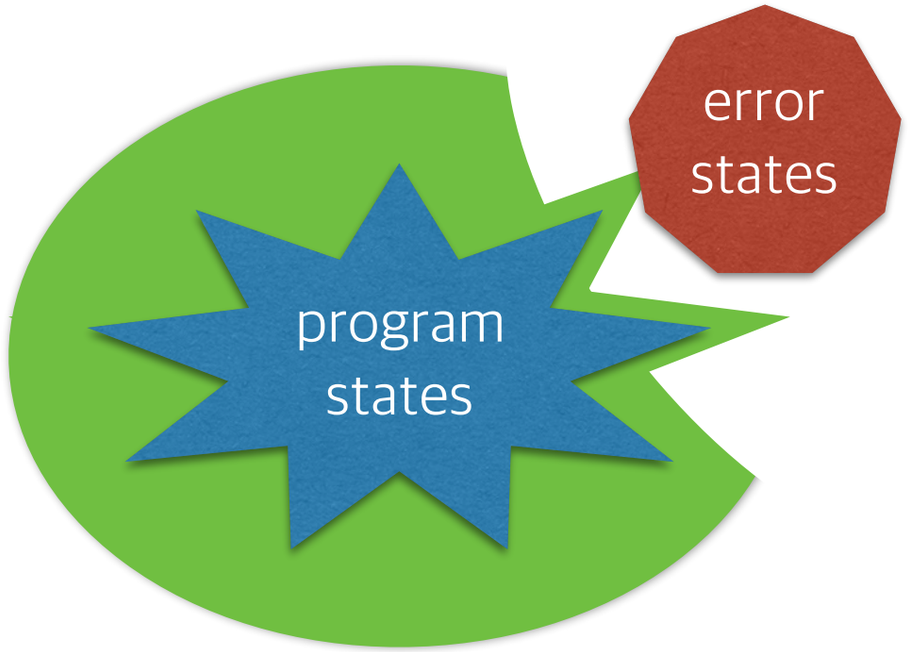
ours

vs.

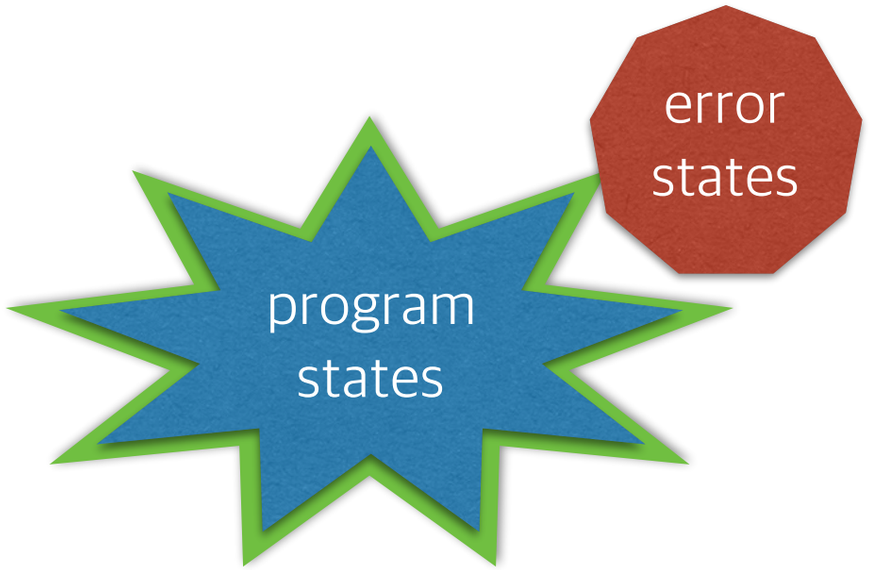


existing techniques

Effectiveness



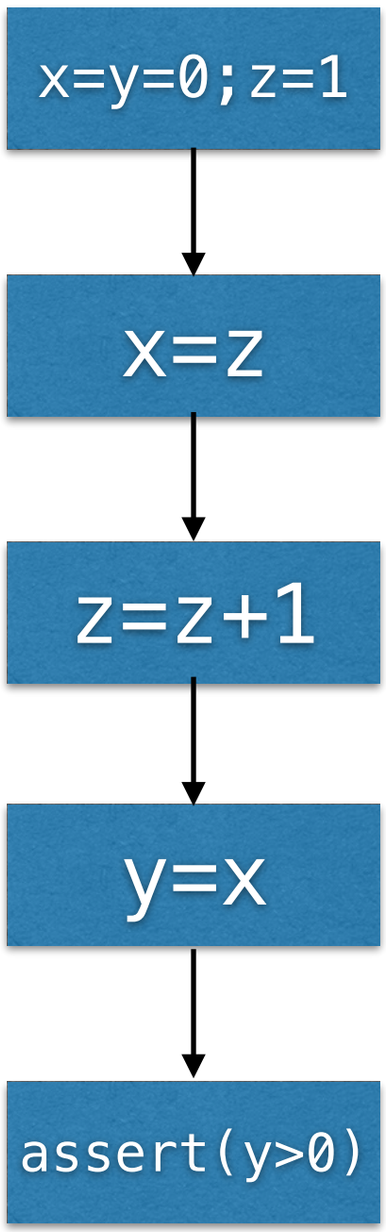
vs.



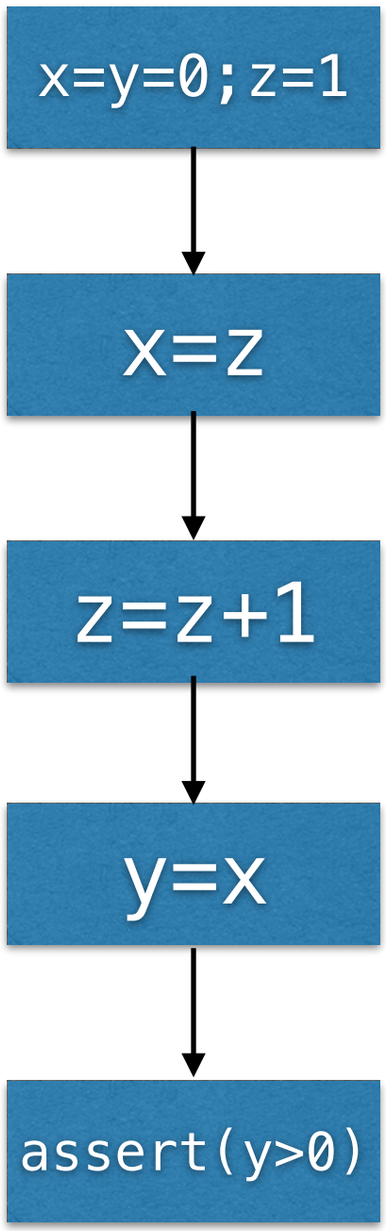
+25% / -25%

+25% / -1300%

Flow-Sensitivity

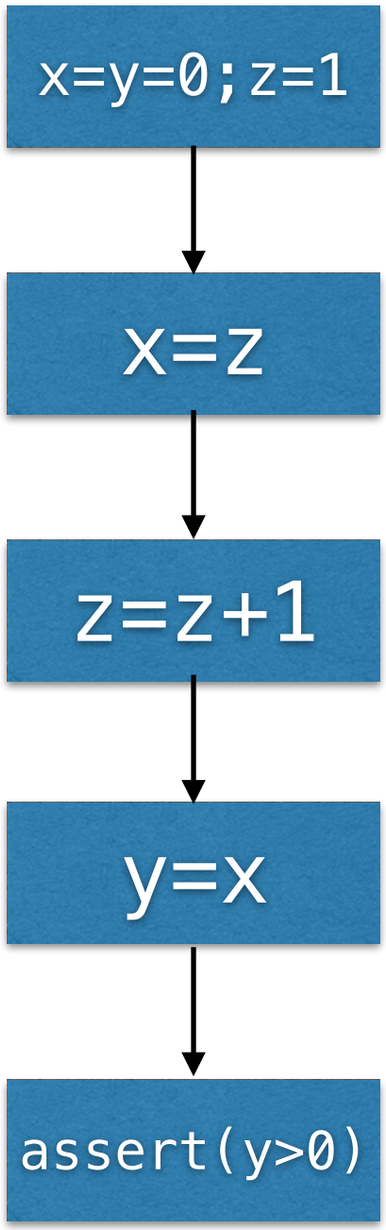


Flow-Sensitivity



x	[0,0]
y	[0,0]
z	[1,1]

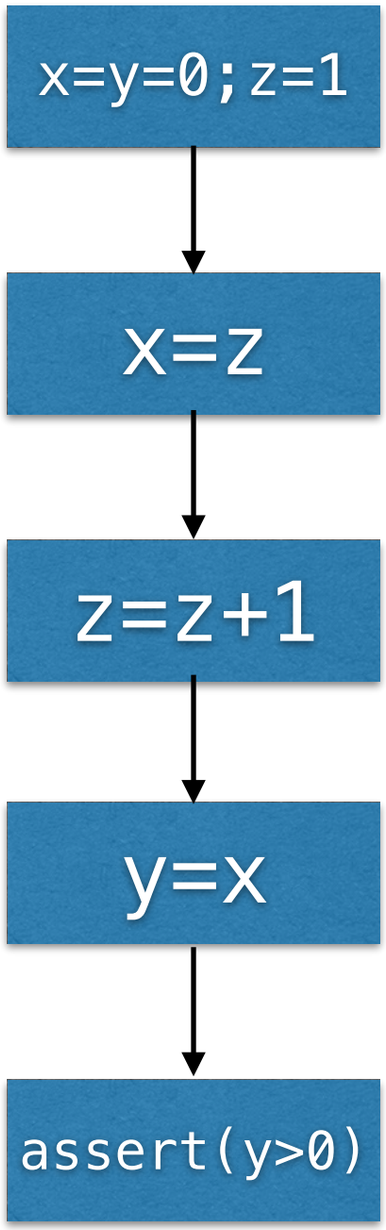
Flow-Sensitivity



x	[0,0]
y	[0,0]
z	[1,1]

x	[1,1]
y	[0,0]
z	[1,1]

Flow-Sensitivity

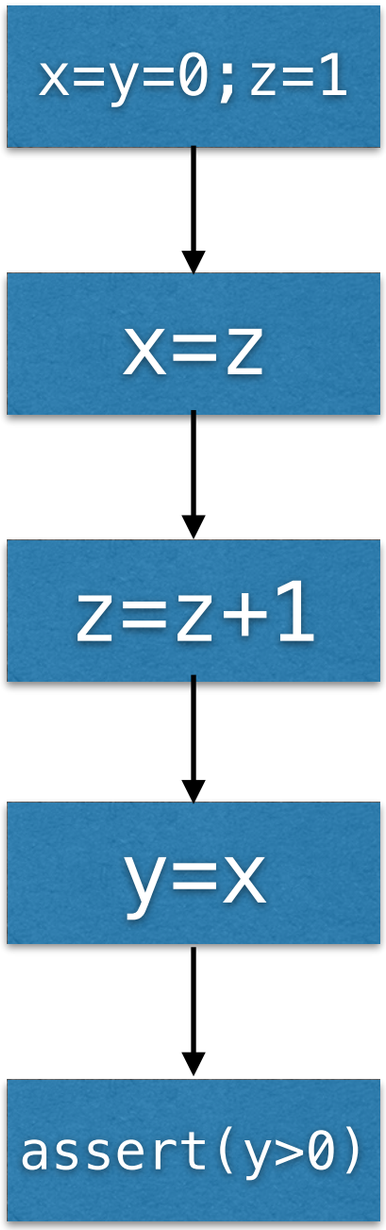


x	[0,0]
y	[0,0]
z	[1,1]

x	[1,1]
y	[0,0]
z	[1,1]

x	[1,1]
y	[0,0]
z	[2,2]

Flow-Sensitivity



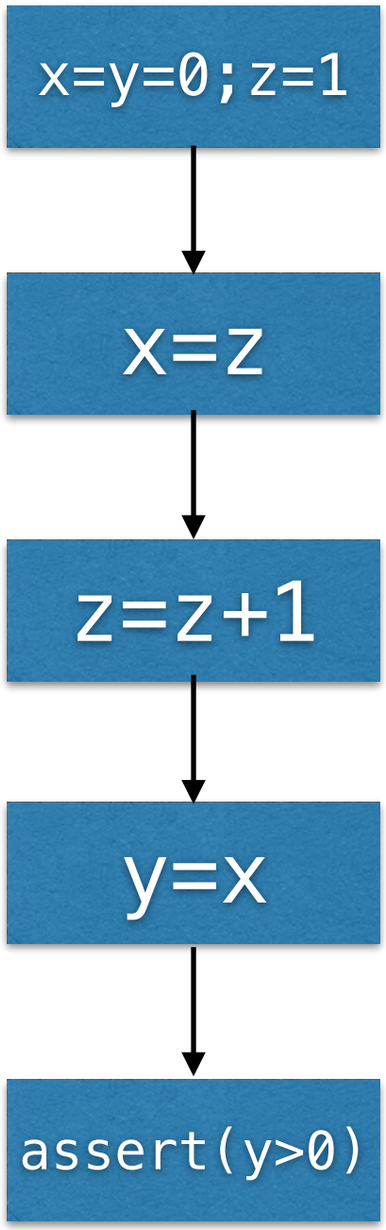
x	[0,0]
y	[0,0]
z	[1,1]

x	[1,1]
y	[0,0]
z	[1,1]

x	[1,1]
y	[0,0]
z	[2,2]

x	[1,1]
y	[1,1]
z	[2,2]

Flow-Sensitivity



x	[0,0]
y	[0,0]
z	[1,1]

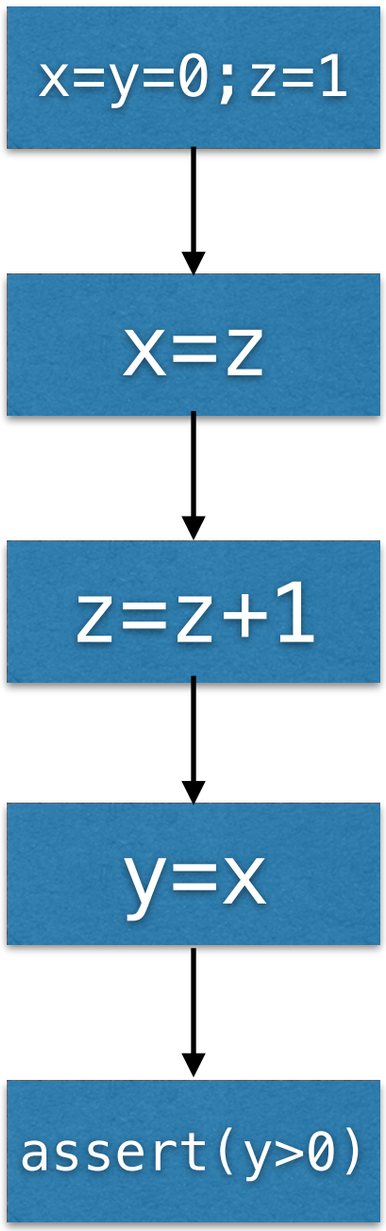
x	[1,1]
y	[0,0]
z	[1,1]

x	[1,1]
y	[0,0]
z	[2,2]

x	[1,1]
y	[1,1]
z	[2,2]

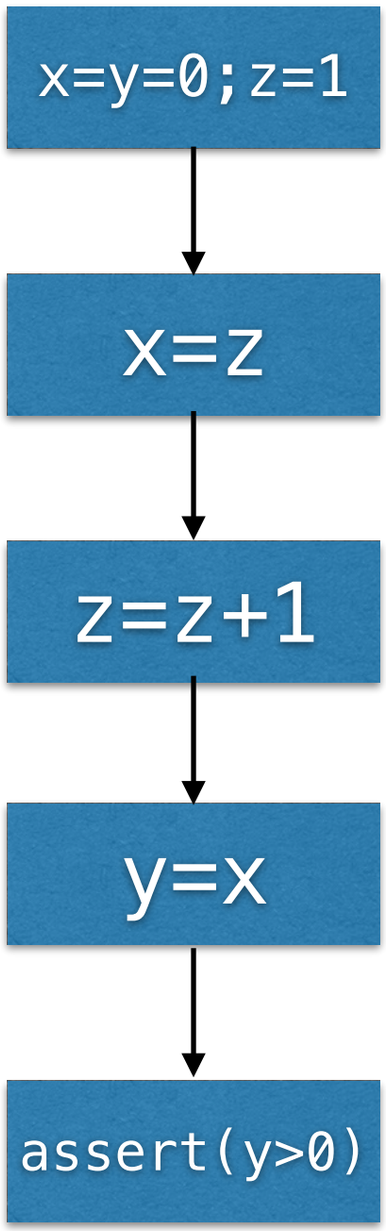
precise but costly

Flow-Insensitivity



x	[0,+∞]
y	[0,+∞]
z	[1,+∞]

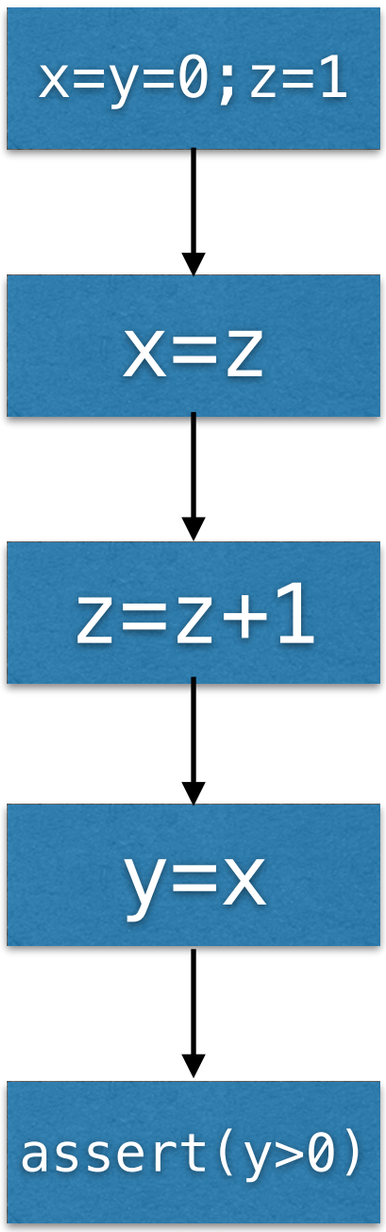
Flow-Insensitivity



x	[0,+∞]
y	[0,+∞]
z	[1,+∞]

cheap but imprecise

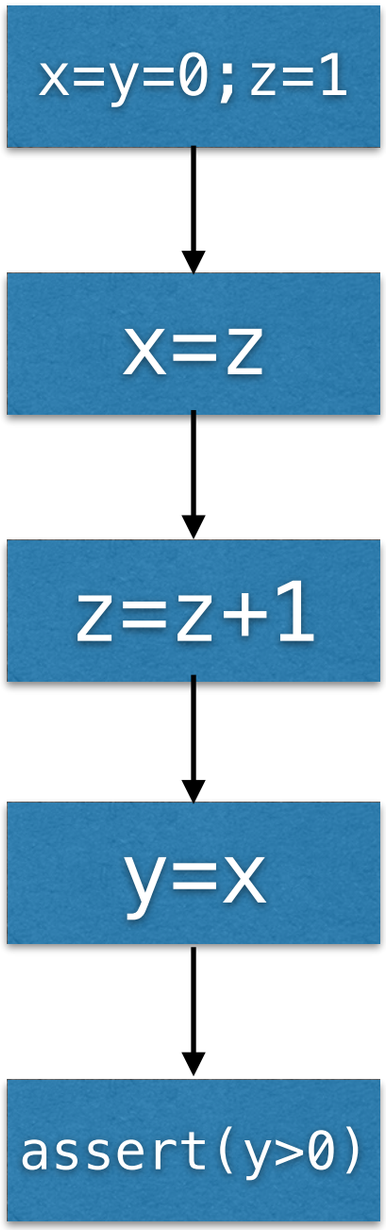
Selective Flow-Sensitivity



FS : {x}

FI : {y,z}

Selective Flow-Sensitivity



FS : {x}

x	[0,0]

x	[1,+∞]

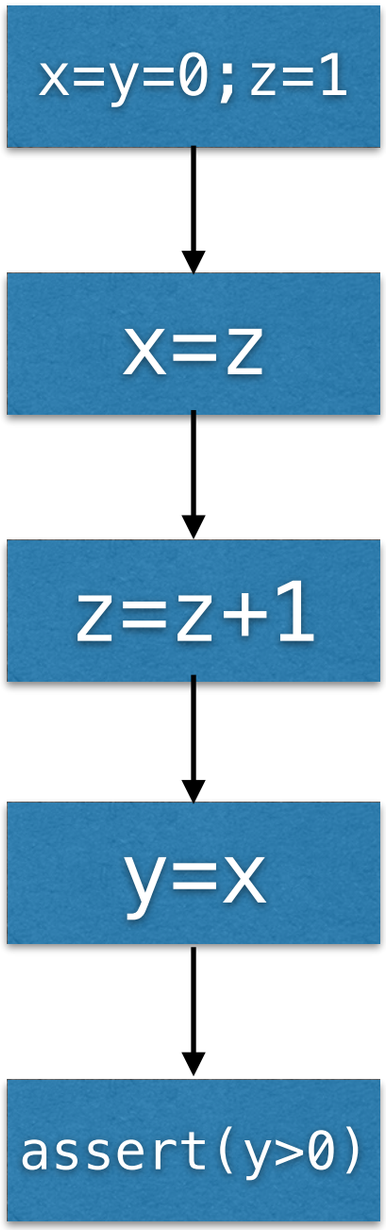
x	[1,+∞]

x	[1,+∞]

FI : {y,z}

y	[0,+∞]
z	[1,+∞]

Selective Flow-Sensitivity



FS : {x}

x	[0,0]

x	[1,+∞]

x	[1,+∞]

x	[1,+∞]

fail to prove

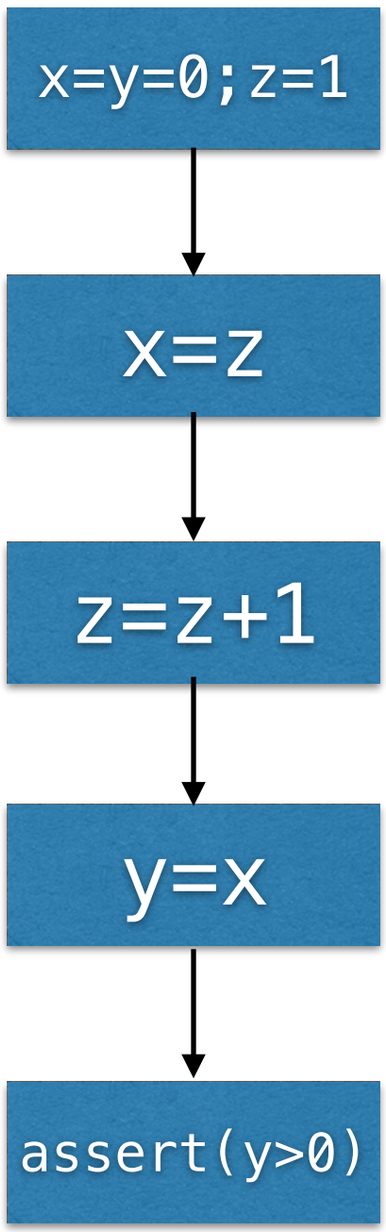
FI : {y,z}

y	[0,+∞]
z	[1,+∞]

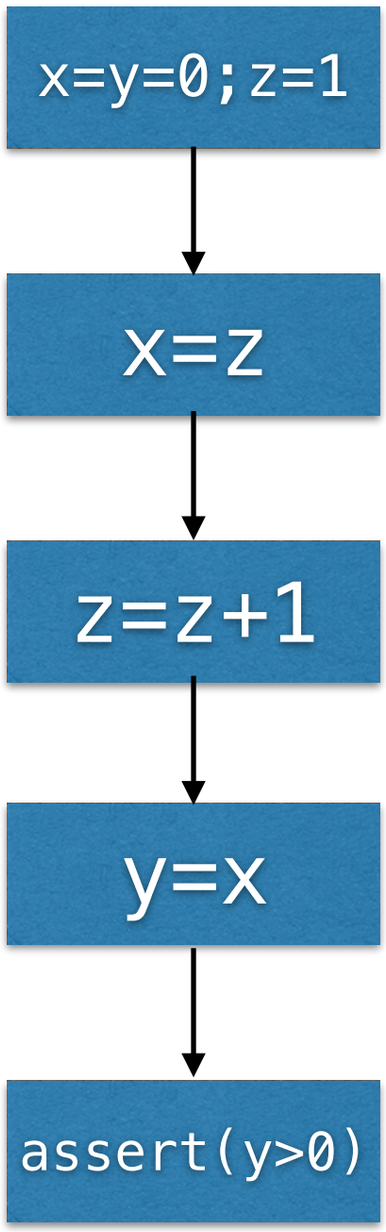
Selective Flow-Sensitivity

FS : {y}

FI : {x,z}



Selective Flow-Sensitivity



FS : {y}

y	[0,0]

y	[0,0]

y	[0,0]

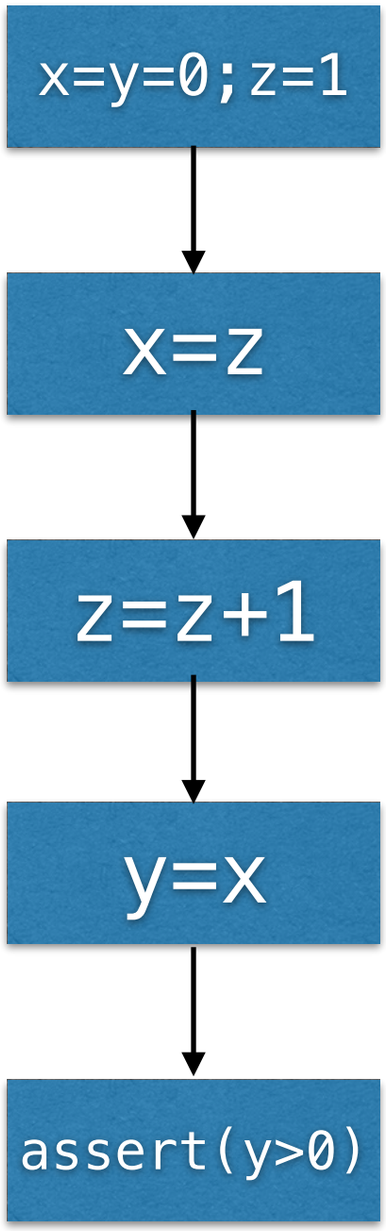
y	[0,+∞]

fail to prove

FI : {x,z}

x	[0,+∞]
z	[1,+∞]

Selective Flow-Sensitivity



FS : {z}

z	[1,1]

z	[1,1]

z	[2,2]

z	[2,2]

fail to prove

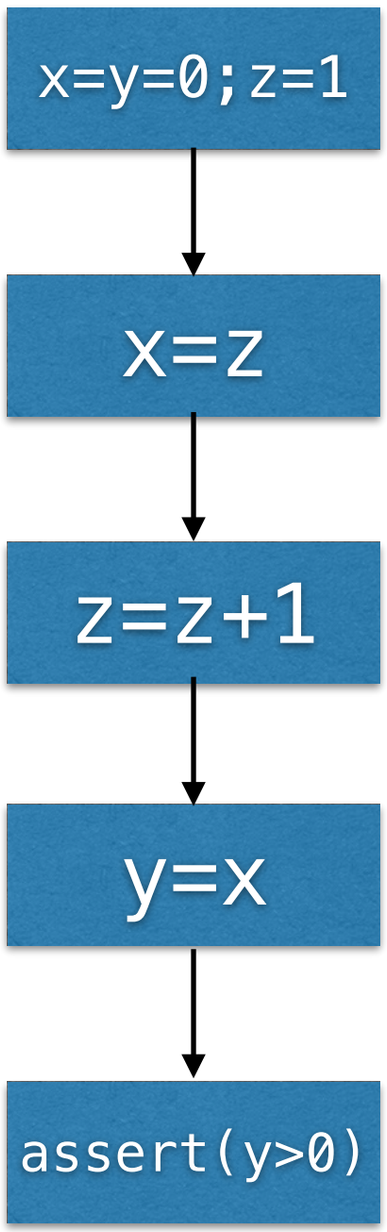
FI : {x,y}

x	[0,+∞]
y	[0,+∞]

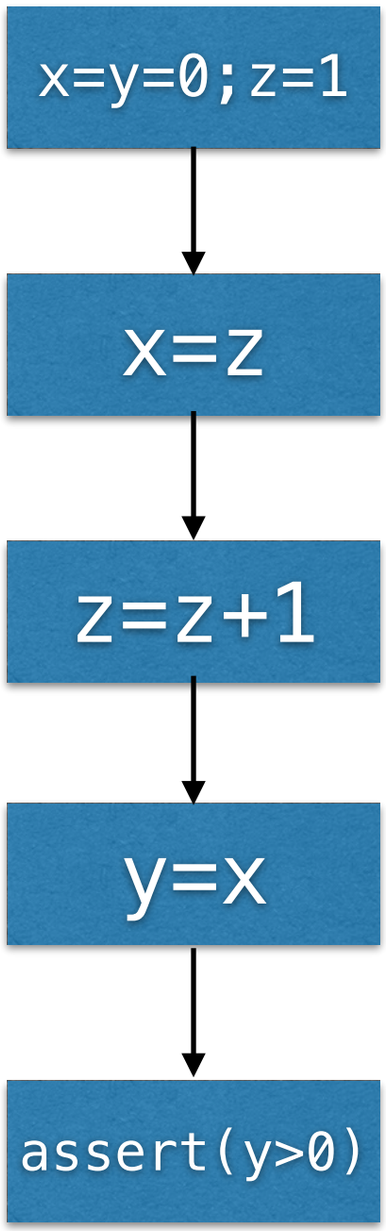
Selective Flow-Sensitivity

FS : {y,z}

FI : {x}



Selective Flow-Sensitivity



FS : {y,z}

y	[0,0]
z	[1,1]

y	[0,0]
z	[1,1]

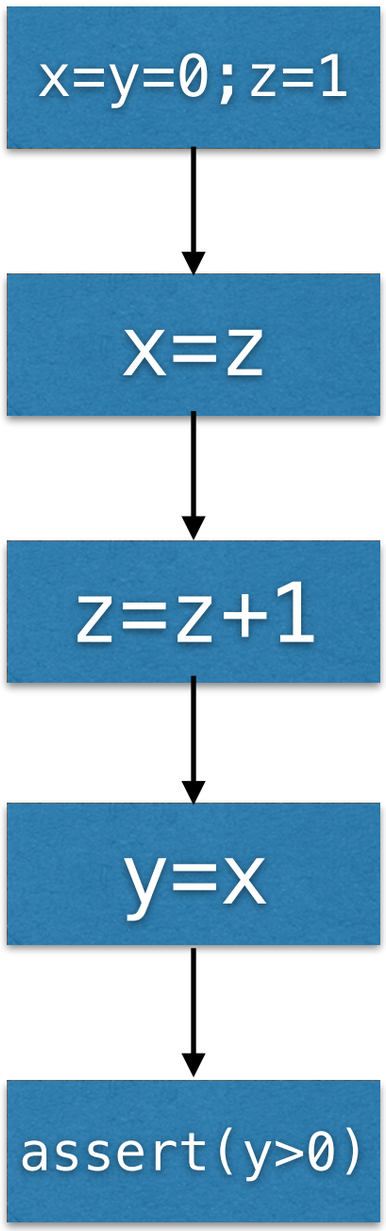
y	[0,0]
z	[2,2]

y	[0,+∞]
z	[2,2]

FI : {x}

x	[0,+∞]

Selective Flow-Sensitivity



FS : {y,z}

y	[0,0]
z	[1,1]

y	[0,0]
z	[1,1]

y	[0,0]
z	[2,2]

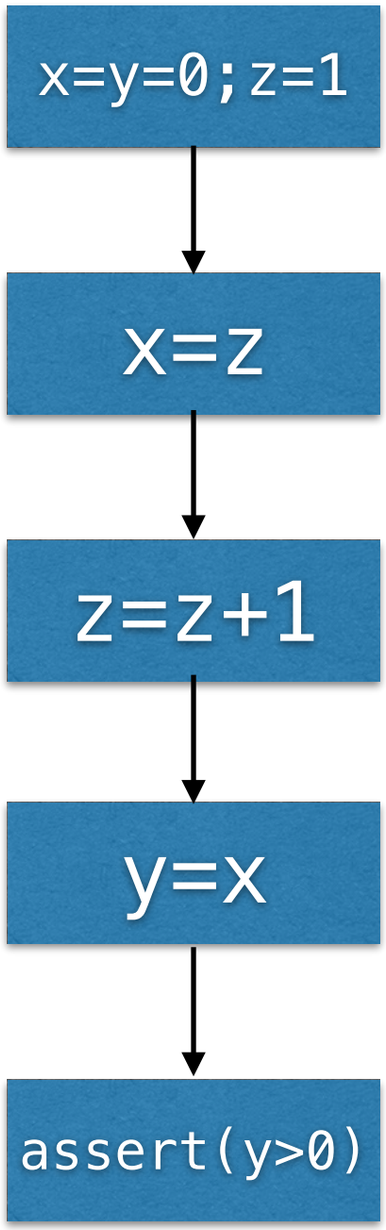
y	[0,+∞]
z	[2,2]

fail to prove

FI : {x}

x	[0,+∞]

Selective Flow-Sensitivity



FS : {x,y}

x	[0,0]
y	[0,0]

x	[1,+∞]
y	[0,0]

x	[1,+∞]
y	[1,+∞]

Succeed

FI : {z}

z	[1,+∞]

Hard Search Problem

- Intractably large space, if not infinite
  - 2^{Var} different abstractions for FS
- Most of them are too imprecise or costly
  - P({x,y,z}) = {~~∅~~,~~{x}~~,~~{y}~~,~~{z}~~,{x,y},~~{y,z}~~,~~{x,z}~~,{x,y,z}}

Our Solutions

- Two approaches:
  - PL approaches [PLDI'14, TOPLAS'16]
  - ML approaches [OOPSLA'15, on-going work]

OOPSLA'15

Learning-based Approach

OOPSLA'15

Learning-based Approach

- Parameterized adaptation strategy

$$S_w : \text{pgm} \rightarrow 2^{\text{Var}}$$

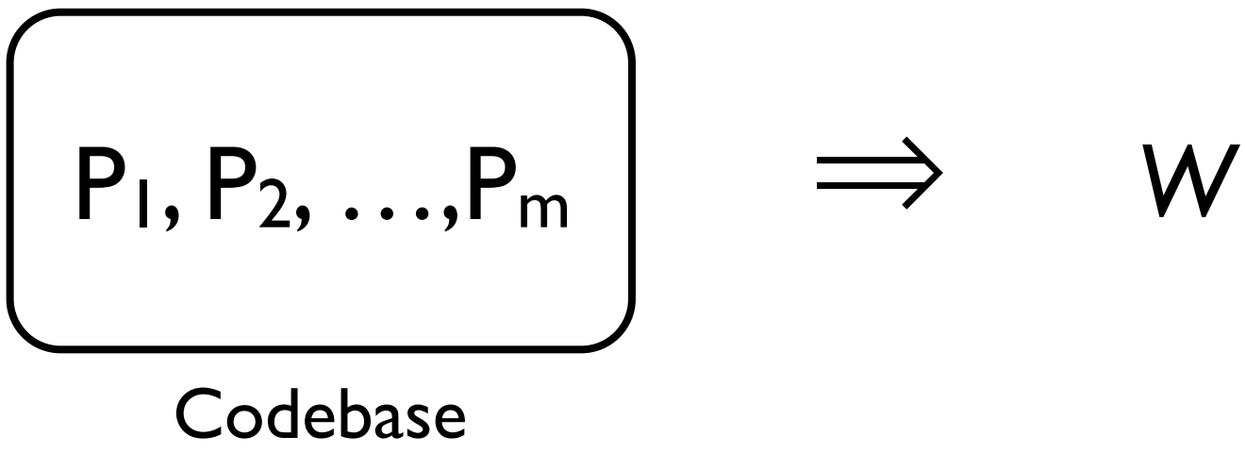
OOPSLA'15

Learning-based Approach

- Parameterized adaptation strategy

$$S_w : \text{pgm} \rightarrow 2^{\text{Var}}$$

- Learn a good parameter *W* from existing codebase



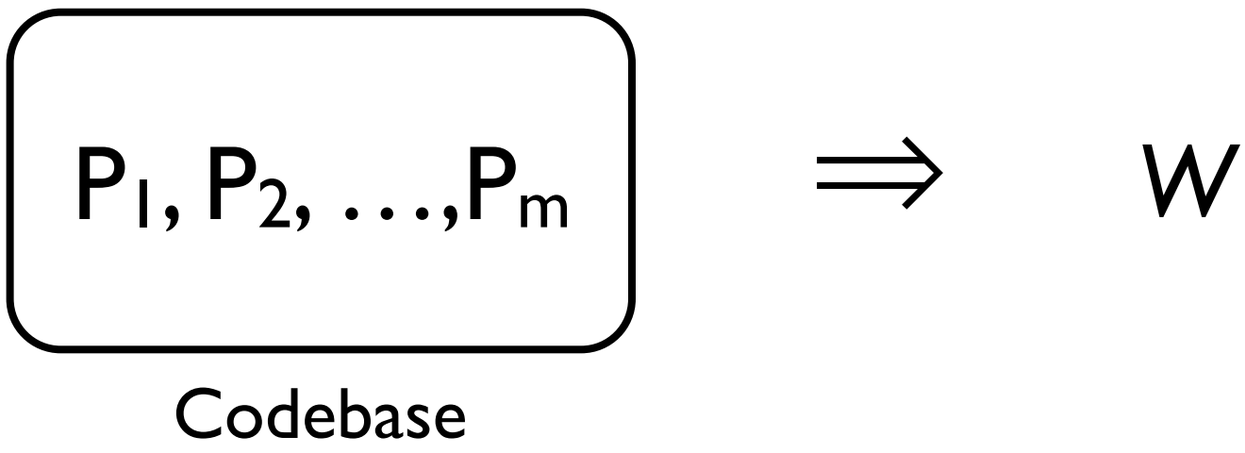
OOPSLA'15

Learning-based Approach

- Parameterized adaptation strategy

$$S_W : \text{pgm} \rightarrow 2^{\text{Var}}$$

- Learn a good parameter *W* from existing codebase



- For new program P, run static analysis with *Sw*(P)

1. Parameterized Strategy

$$S_w : \text{pgm} \rightarrow 2^{\text{Var}}$$

(1) Represent program variables as feature vectors.

(2) Compute the score of each variable.

(3) Choose the top-k variables based on the score.

(1) Features

- Predicates over variables:

$$f = \{f_1, f_2, \ldots, f_5\} \qquad (f_i : \text{Var} \rightarrow \{0,1\})$$

- 45 simple syntactic features for variables: e.g,
  - local / global variable, passed to / returned from malloc, incremented by constants, etc

- Represent each variable as a feature vector:

$$f(x) = \langle f_1(x), f_2(x), f_3(x), f_4(x), f_5(x) \rangle$$

(2) Scoring

- The parameter **w** is a real-valued vector: e.g.,

$$\mathbf{w} = \langle 0.9, 0.5, -0.6, 0.7, 0.3 \rangle$$

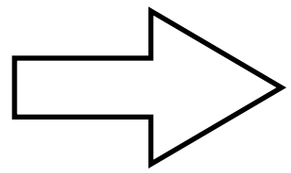
- Compute scores of variables:

$$\text{score}(x) = \langle 1,0,1,0,0 \rangle \cdot \langle 0.9, 0.5, -0.6, 0.7, 0.3 \rangle = 0.3$$
$$\text{score}(y) = \langle 1,0,1,0,1 \rangle \cdot \langle 0.9, 0.5, -0.6, 0.7, 0.3 \rangle = 0.6$$
$$\text{score}(z) = \langle 0,0,1,1,0 \rangle \cdot \langle 0.9, 0.5, -0.6, 0.7, 0.3 \rangle = 0.1$$

(3) Choose Top-k Variables

- Choose the top-k variables based on their scores:
  e.g., when k=2,

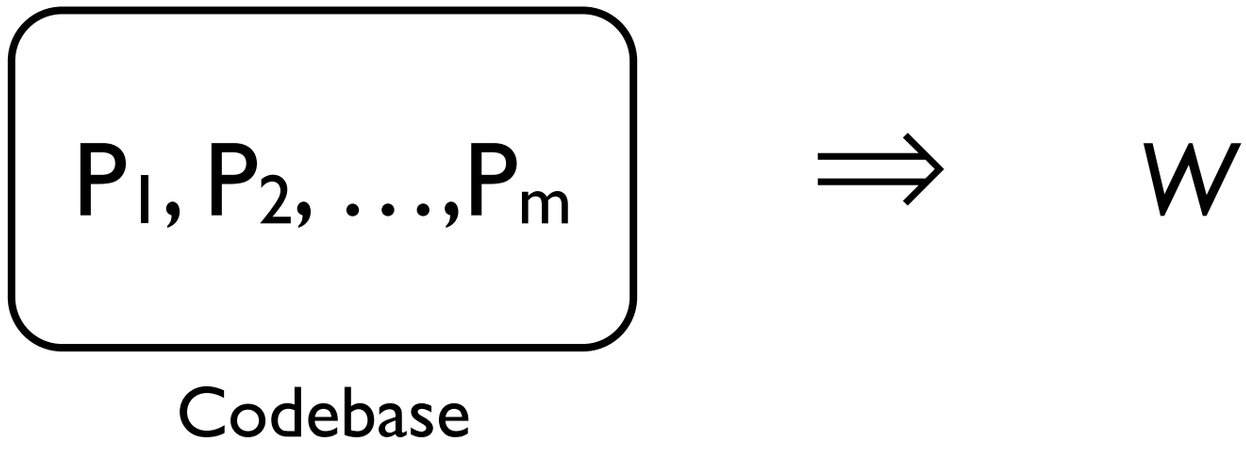
score(x) = 0.3
score(y) = 0.6
score(z) = 0.1



{x,y}

- In experiments, we chosen 10% of variables with highest scores.

2. Learn a Good Parameter



- Solve the optimization problem:

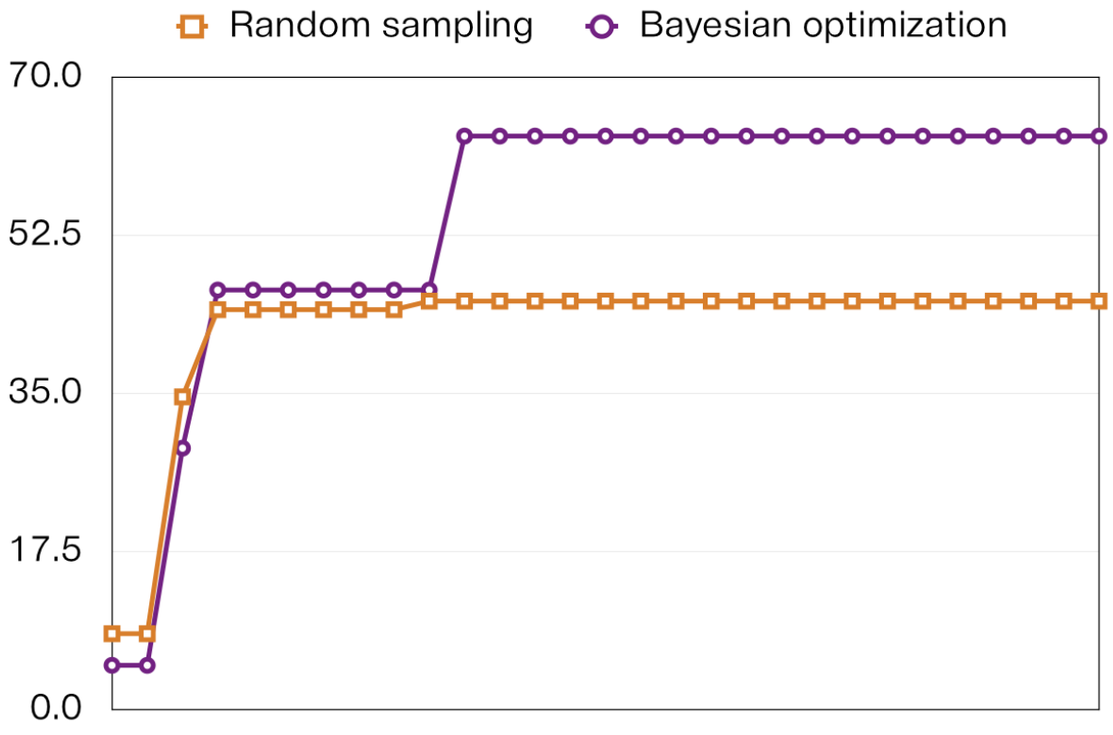
Find **w** that maximizes $\sum_{P_i} F(P_i, S_{\mathbf{w}}(P_i))$

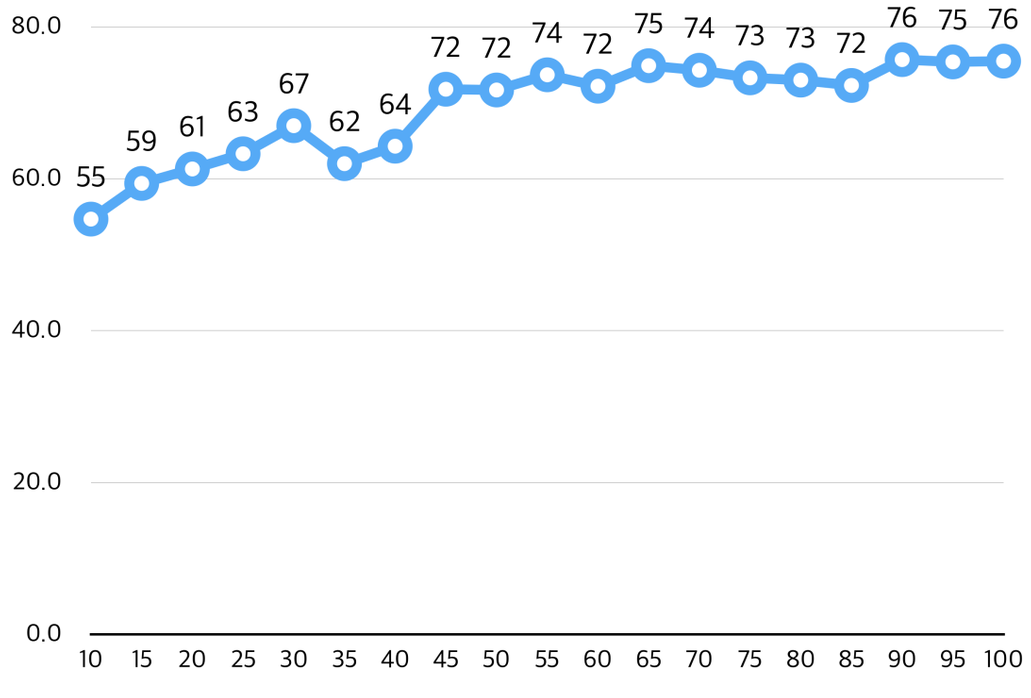
Solving the Opt. Problem

- How to solve the optimization problem efficiently?

Find **w** that maximizes $\sum_{P_i} F(P_i, S_{\mathbf{w}}(P_i))$

- Using ideas of Bayesian optimization



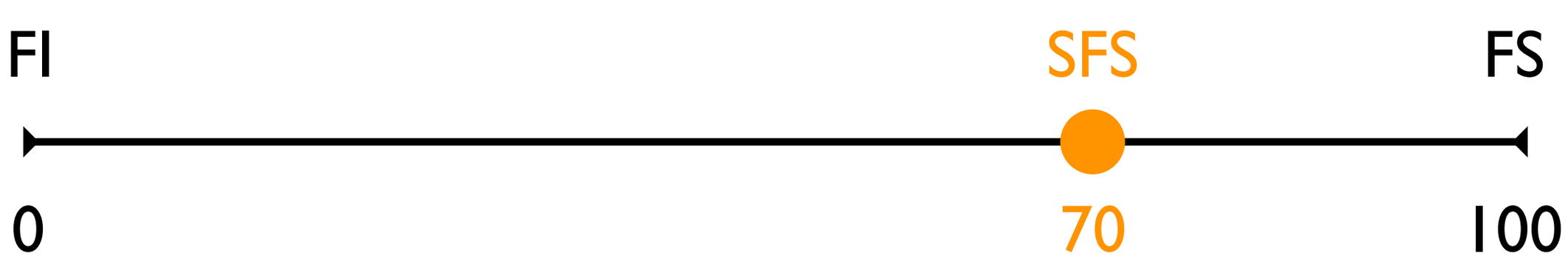


Effectiveness

- Implemented in Sparrow, an interval analyzer for C
- Evaluated on 30 open-source benchmarks

Effectiveness

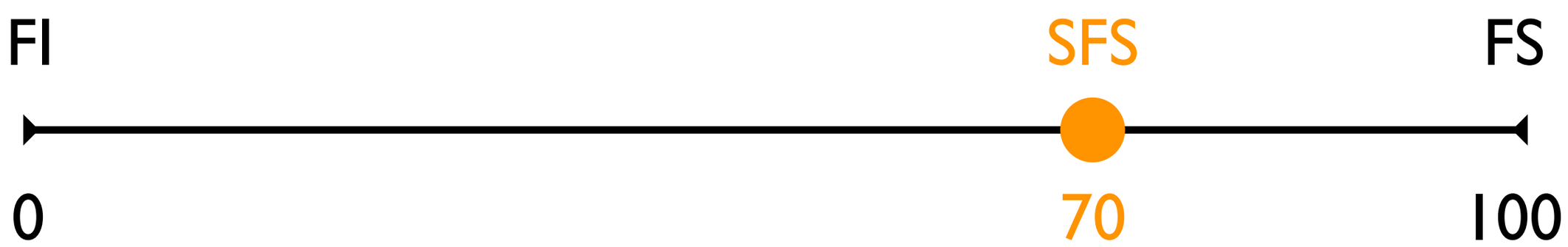
- Implemented in Sparrow, an interval analyzer for C
- Evaluated on 30 open-source benchmarks



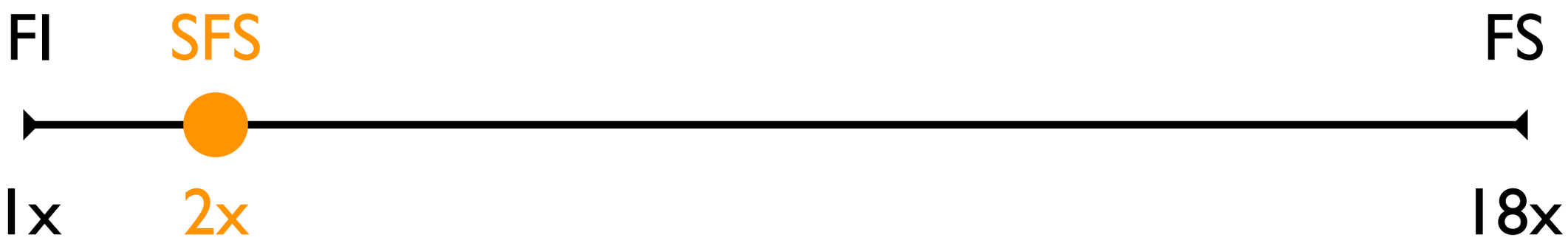
Effectiveness

- Implemented in Sparrow, an interval analyzer for C
- Evaluated on 30 open-source benchmarks

Precision



Cost



Hurdle

- The success crucially depends on the choice of features
- Feature construction is nontrivial and tedious
- |analyzers| x |parameter types| x |query types|

Type	#	Features
A	1	local variable
	2	global variable
	3	structure field
	4	location created by dynamic memory allocation
	5	defined at one program point
	6	location potentially generated in library code
	7	assigned a constant expression (e.g., x = c1 + c2)
	8	compared with a constant expression (e.g., $x < c$)
	9	compared with an other variable (e.g., $x < y$)
	10	negated in a conditional expression (e.g., if (!x))
	11	directly used in malloc (e.g., malloc(x))
	12	indirectly used in malloc (e.g., y = x; malloc(y))
	13	directly used in realloc (e.g., realloc(x))
	14	indirectly used in realloc (e.g., y = x; realloc(y))
	15	directly returned from malloc (e.g., x = malloc(e))
	16	indirectly returned from malloc
	17	directly returned from realloc (e.g., x = realloc(e))
	18	indirectly returned from realloc
	19	incremented by one (e.g., x = x + 1)
	20	incremented by a constant expr. (e.g., x = x + (1+2))
	21	incremented by a variable (e.g., x = x + y)
	22	decremented by one (e.g., x = x - 1)
	23	decremented by a constant expr (e.g., x = x - (1+2))
	24	decremented by a variable (e.g., x = x - y)
	25	multiplied by a constant (e.g., x = x * 2)
	26	multiplied by a variable (e.g., x = x * y)
	27	incremented pointer (e.g., p++)
	28	used as an array index (e.g., a[x])
	29	used in an array expr. (e.g., x[e])
	30	returned from an unknown library function
	31	modified inside a recursive function
	32	modified inside a local loop
	33	read inside a local loop
B	34	$1 \wedge 8 \wedge (11 \vee 12)$
	35	$2 \wedge 8 \wedge (11 \vee 12)$
	36	$1 \wedge (11 \vee 12) \wedge (19 \vee 20)$
	37	$2 \wedge (11 \vee 12) \wedge (19 \vee 20)$
	38	$1 \wedge (11 \vee 12) \wedge (15 \vee 16)$
	39	$2 \wedge (11 \vee 12) \wedge (15 \vee 16)$
	40	$(11 \vee 12) \wedge 29$
	41	$(15 \vee 16) \wedge 29$
	42	$1 \wedge (19 \vee 20) \wedge 33$
	43	$2 \wedge (19 \vee 20) \wedge 33$
	44	$1 \wedge (19 \vee 20) \wedge \neg 33$
	45	$2 \wedge (19 \vee 20) \wedge \neg 33$

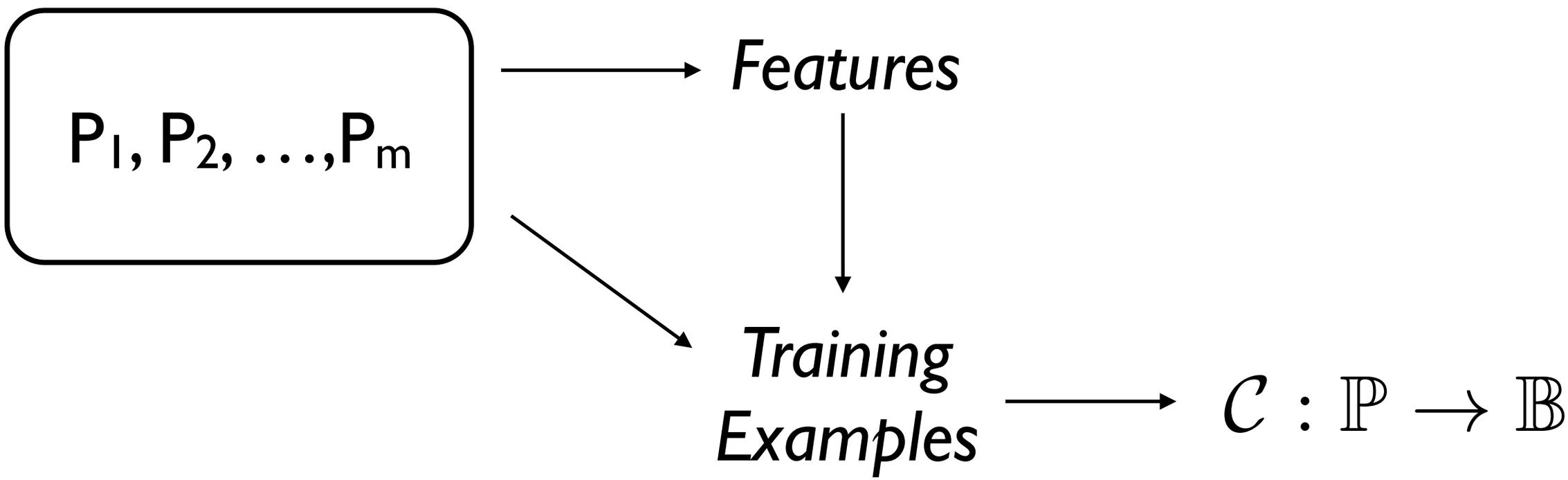
Type	#	Features
A	1	leaf function
	2	function containing malloc
	3	function containing realloc
	4	function containing a loop
	5	function containing an if statement
	6	function containing a switch statement
	7	function using a string-related library function
	8	write to a global variable
	9	read a global variable
	10	write to a structure field
	11	read from a structure field
	12	directly return a constant expression
	13	indirectly return a constant expression
	14	directly return an allocated memory
	15	indirectly return an allocated memory
	16	directly return a reallocated memory
	17	indirectly return a reallocated memory
	18	return expression involves field access
	19	return value depends on a structure field
	20	return void
	21	directly invoked with a constant
	22	constant is passed to an argument
	23	invoked with an unknown value
	24	functions having no arguments
	25	functions having one argument
	26	functions having more than one argument
	27	functions having an integer argument
	28	functions having a pointer argument
	29	functions having a structure as an argument
B	30	$2 \wedge (21 \vee 22) \wedge (14 \vee 15)$
	31	$2 \wedge (21 \vee 22) \wedge \neg(14 \vee 15)$
	32	$2 \wedge 23 \wedge (14 \vee 15)$
	33	$2 \wedge 23 \wedge \neg(14 \vee 15)$
	34	$2 \wedge (21 \vee 22) \wedge (16 \vee 17)$
	35	$2 \wedge (21 \vee 22) \wedge \neg(16 \vee 17)$
	36	$2 \wedge 23 \wedge (16 \vee 17)$
	37	$2 \wedge 23 \wedge \neg(16 \vee 17)$
	38	$(21 \vee 22) \wedge \neg 23$

Type	#	Features
A	1	used in array declarations (e.g., a[c])
	2	used in memory allocation (e.g., malloc(c))
	3	used in the righthand-side of an assignment (e.g., x = c)
	4	used with the less-than operator (e.g, $x < c$)
	5	used with the greater-than operator (e.g., $x > c$)
	6	used with $\leq$ (e.g., $x \leq c$)
	7	used with $\geq$ (e.g., $x \geq c$)
	8	used with the equality operator (e.g., x == c)
	9	used with the not-equality operator (e.g., $x != c$)
	10	used within other conditional expressions (e.g., $x < c+y$)
	11	used inside loops
	12	used in return statements (e.g., return c)
	13	constant zero
B	14	$(1 \vee 2) \wedge 3$
	15	$(1 \vee 2) \wedge (4 \vee 5 \vee 6 \vee 7)$
	16	$(1 \vee 2) \wedge (8 \vee 9)$
	17	$(1 \vee 2) \wedge 11$
	18	$(1 \vee 2) \wedge 12$
	19	$13 \wedge 3$
	20	$13 \wedge (4 \vee 5 \vee 6 \vee 7)$
	21	$13 \wedge (8 \vee 9)$
	22	$13 \wedge 11$
	23	$13 \wedge 12$

On-going

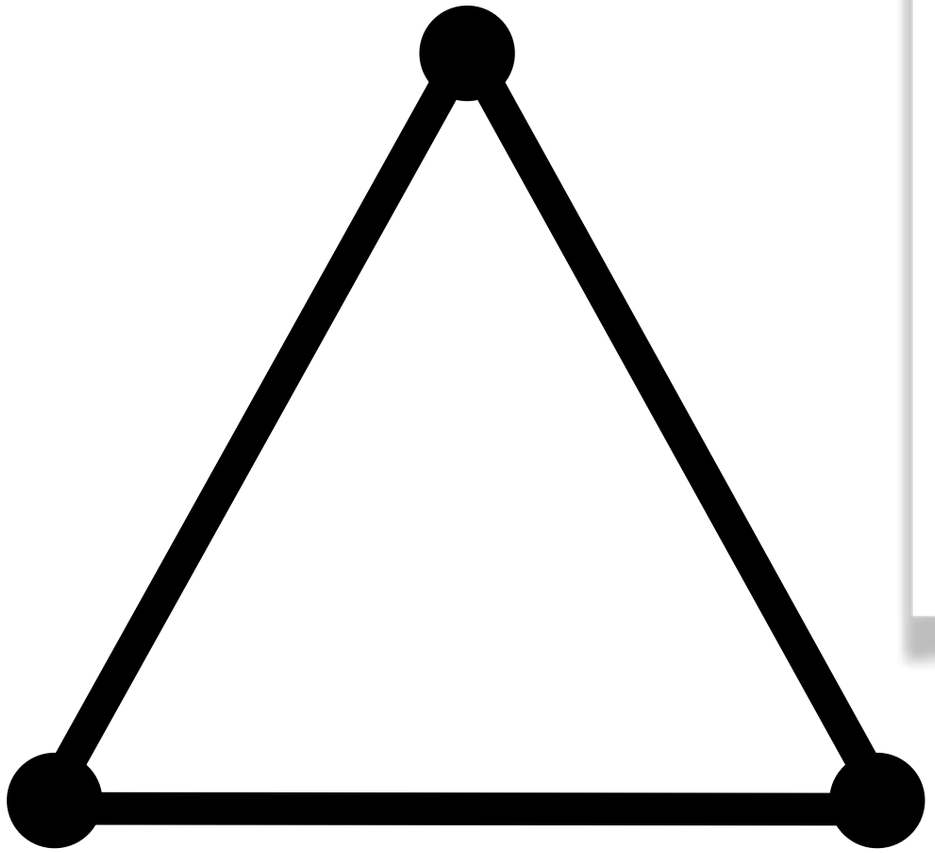
Learning with Automatic Feature Construction

- Fully automatic learning approach



Powerful Static Analysis Enabled

Soundness



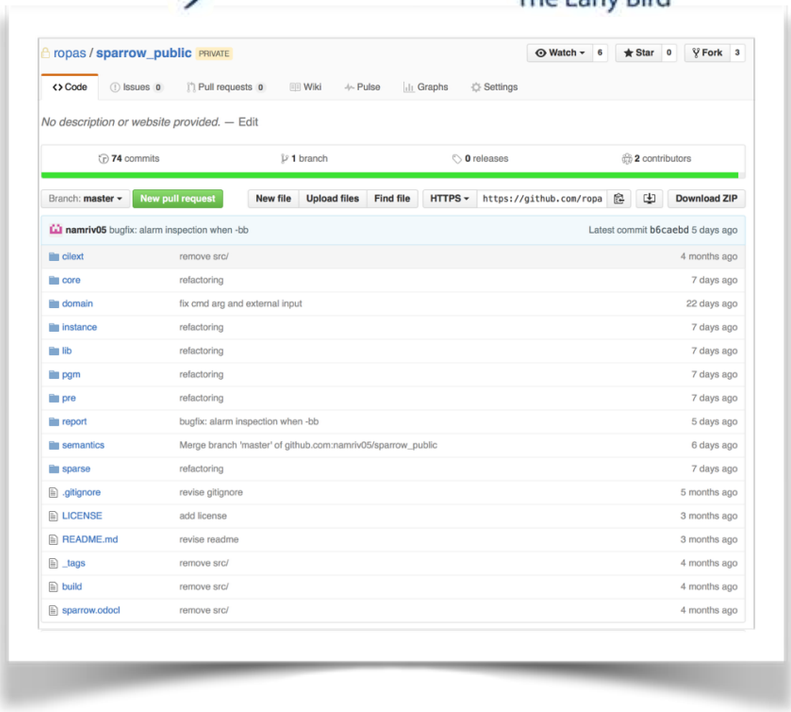
Scalability

Precision

General Sparse Analysis Framework

Selective X-Sensitivity Framework

Sparrow
The Early Bird



Applications (1)

Security-critical softwares

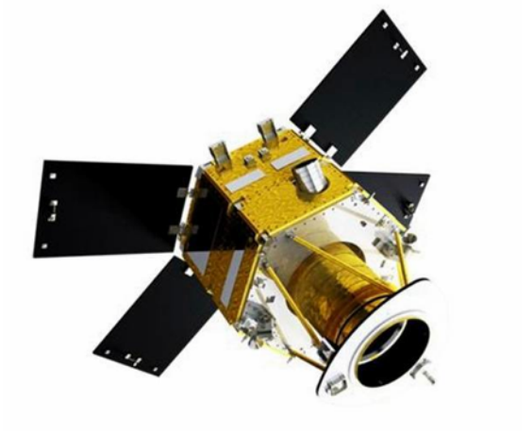
- Security verifiers for OpenSSL, Apache, Sendmail, etc

Applications (2)

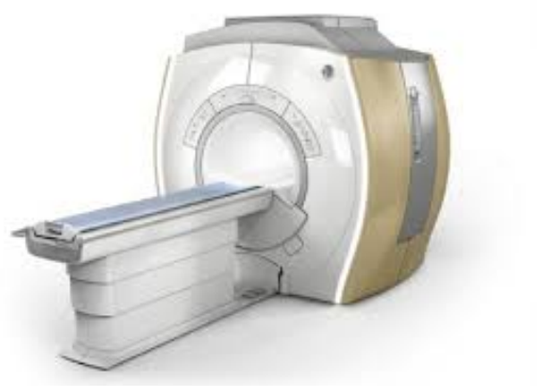
Safety-critical softwares

Red Hawk

Hubo



Deimos-2



KAIST PET-MRI

- Static verifier for flight SW
- Static verifier for robot SW
- Static verifier for satellite SW, etc

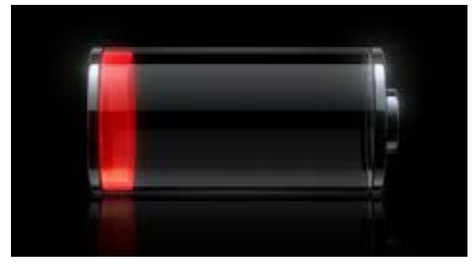
Applications (3)

Efficient uses of modern computing platforms

- Mobile
- Cloud
- Parallel
- Wearable
- …

- New Software challenges:
  e.g., reliability, energy-efficiency, security, …

Static Analysis for Mobile Computing



Static Analysis for Mobile Computing

compute A

compute B

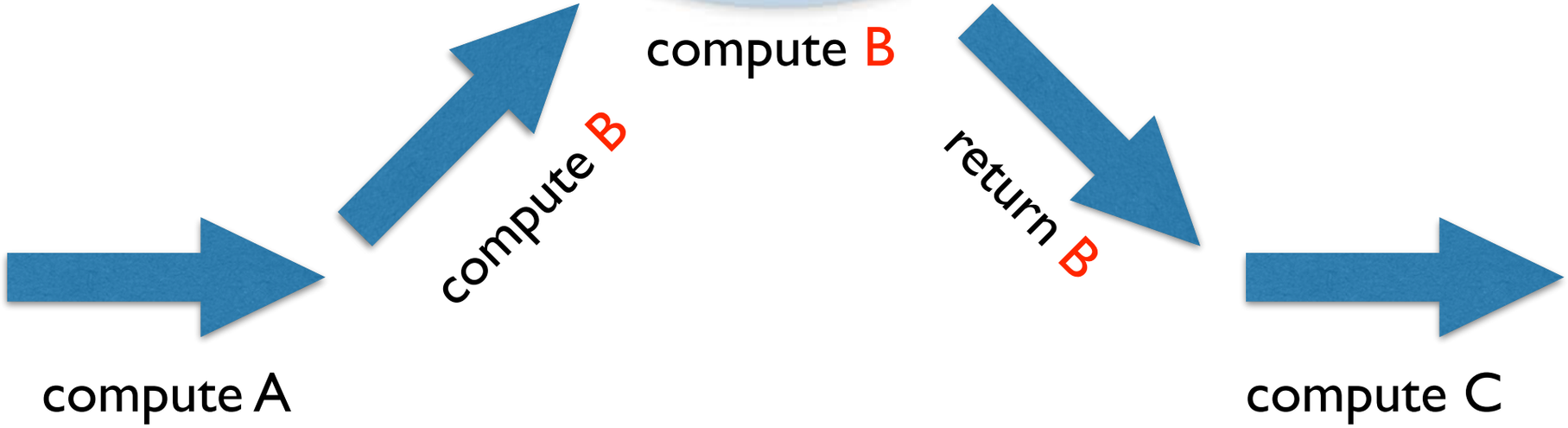
compute B

return B

compute C

B: compute-intensive part

Static Analysis for Mobile Computing



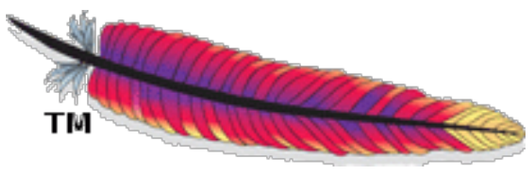
B: compute-intensive part

Plan: Static analysis to estimate power consumption

Summary

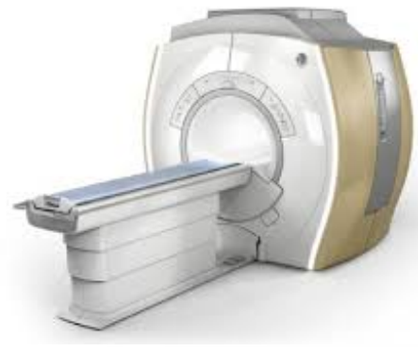
Mobile / Cloud / Parallel Computing

SW Security



Powerful Static Analysis Technology

SW Verification



Programming Languages Theories