

오픈소스 대상 고성능 정적 분석 기술

오학주

프로그래밍 연구실
고려대학교 정보대학

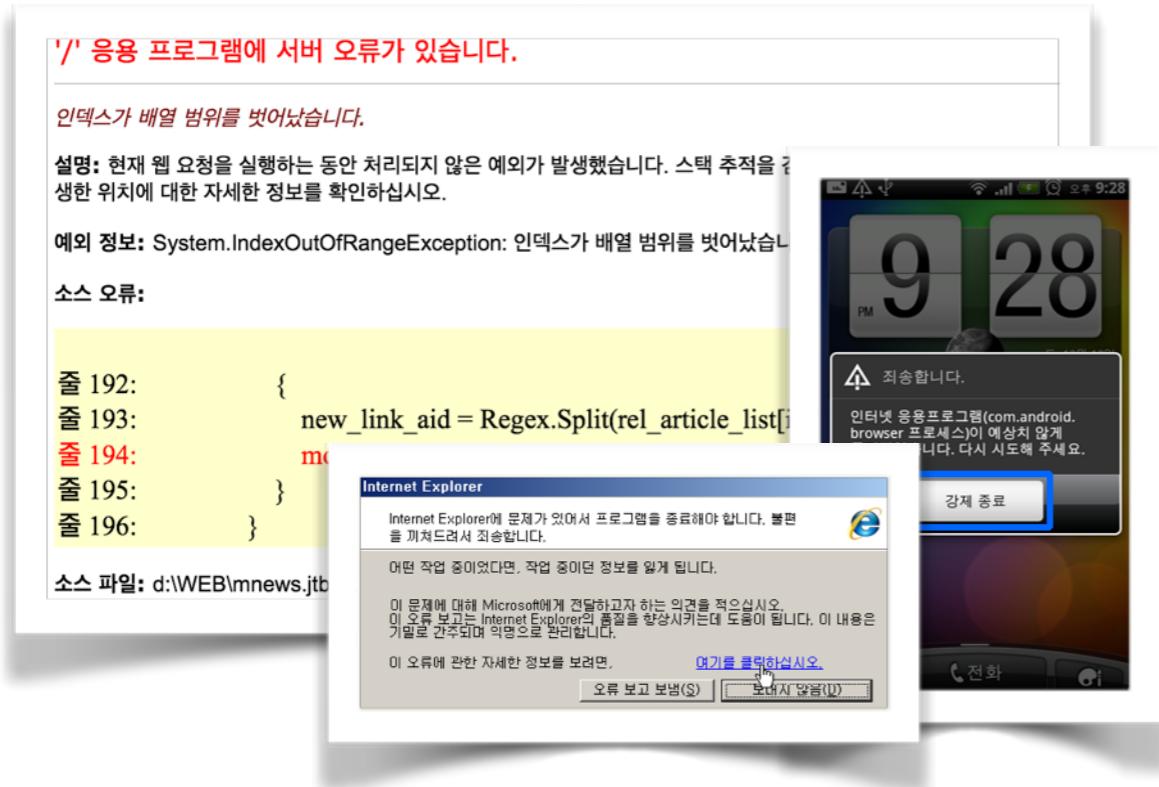


May. 12, 2016

.CSSA

Motivation: Unsafe Software

Motivation: Unsafe Software



Motivation: Unsafe Software



Motivation: Unsafe Software

'/' 응용 프로그램에 서버 오류가 있습니다.

인덱스가 배열 범위를 벗어났습니다.

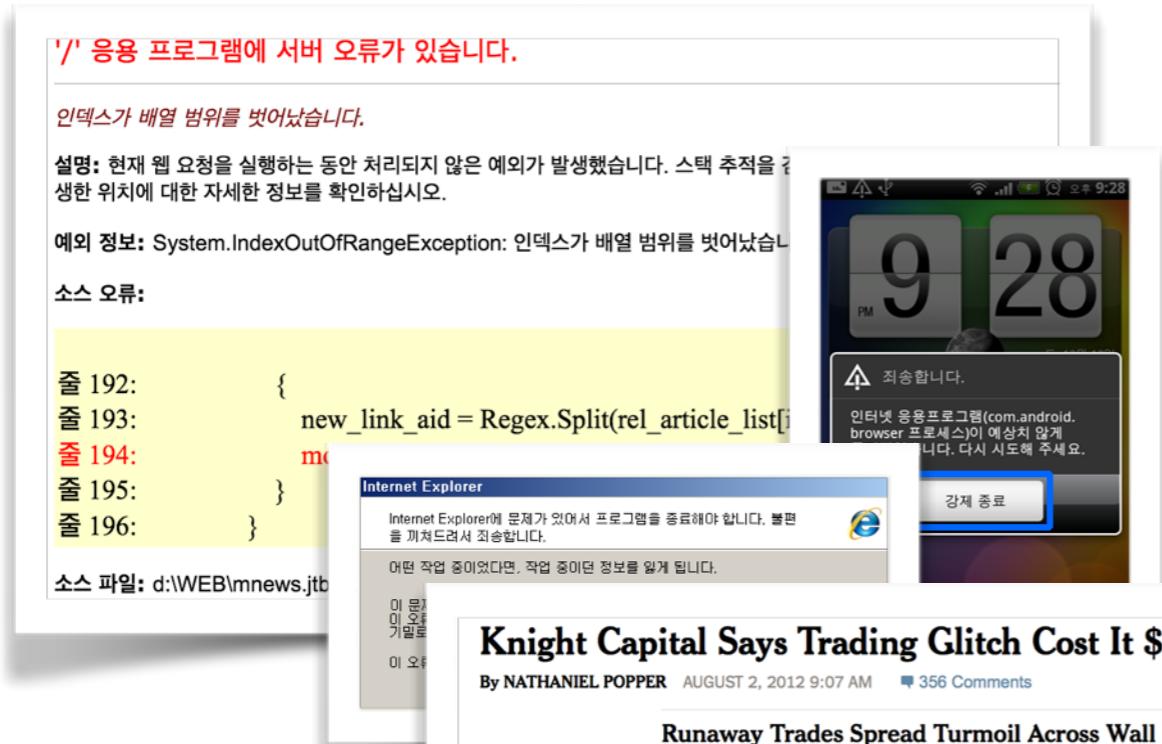
설명: 현재 웹 요청을 실행하는 동안 처리되지 않은 예외가 발생했습니다. 스택 추적을 살펴보면 예외가 발생한 위치에 대한 자세한 정보를 확인하십시오.

예외 정보: System.IndexOutOfRangeException: 인덱스가 배열 범위를 벗어났습니다.

소스 오류:

```
줄 192:         {
줄 193:             new_link_aid = Regex.Split(rel_article_list[i]
줄 194:             m
줄 195:         }
줄 196:     }
```

소스 파일: d:\WEB\mnews.jtb



9 28
죄송합니다.
인터넷 응용프로그램(com.android.browser)이 예상치 않게 종료되었습니다. 다시 시도해 주세요.

Internet Explorer
Internet Explorer에서 문제가 있어서 프로그램을 종료해야 합니다. 불편을 끼쳐드려서 죄송합니다.
어떤 작업 중이었다면, 작업 중이던 정보를 알게 됩니다.

Knight Capital Says Trading Glitch Cost It \$440 Million
By NATHANIEL POPPER AUGUST 2, 2012 9:07 AM 356 Comments
Runaway Trades Spread Turmoil Across Wall St.



사회 "외주업체 지하철



Motivation: Unsafe Software

'/' 응용 프로그램에 서버 오류가 있습니다.

인덱스가 배열 범위를 벗어났습니다.

설명: 현재 웹 요청을 실행하는 동안 처리되지 않은 예외가 발생했습니다. 스택 추적을 강생한 위치에 대한 자세한 정보를 확인하십시오.

예외 정보: System.IndexOutOfRangeException: 인덱스가 배열 범위를 벗어났습니다.

소스 오류:

```
줄 192:         {
줄 193:             new_link_aid = Regex.Split(rel_article_list[i]
줄 194:             m
줄 195:         }
줄 196:     }
```

소스 파일: d:\WEB\mnews.jtb

Knight Capital Says Trading Glitch Cost It \$440 Million

By NATHANIEL POPPER AUGUST 2, 2012 9:07 AM 356 Comments

Runaway Trades Spread Turmoil Across Wall St.

사회 "외주업체 지하철



Current Technology for Safe SW

Manual, ad-hoc, postmortem:

code review, testing, simulation, debugging, etc

Our Mission

Technology for “Software MRI”





- 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

Commit	Description	Date
cilext	remove src/	4 months ago
core	refactoring	7 days ago
domain	fix cmd arg and external input	22 days ago
instance	refactoring	7 days ago
lib	refactoring	7 days ago
pgm	refactoring	7 days ago
pre	refactoring	7 days ago
report	bugfix: alarm inspection when -bb	5 days ago
semantics	Merge branch 'master' of github.com:namriv05/sparrow_public	6 days ago
sparse	refactoring	7 days ago
.gitignore	revise gitignore	5 months ago
LICENSE	add license	3 months ago
README.md	revise readme	3 months ago
_tags	remove src/	4 months ago
build	remove src/	4 months ago
sparrow.odoc	remove src/	4 months ago

```
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             safe → curfinal[keysym] = 1;
33         }
34     }
35     else
36     {
37         buffer-overrun → curfinal[keysym] = 2;
38     }
39     if (keysym < 10)
40     {
41         unparseputc(curfinal[keysym], pty);
42     }
43 }

```

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             safe curfinal[keysym] = 1;
33         }
34     }
35     else
36     {
37         buffer-overrun curfinal[keysym] = 2;
38     }
39     if (keysym < 10)
40     {
41         unparseputc(curfinal[keysym], pty);
42     }
43 }
```

curfinal: buffer of size 10

Sparrow automatically
pinpoints the buffer-overrun bug

```
16 static char *curfinal = "HDACB  FE";
17
18 keysym = read_from_input();
19
20 if (((KeySym)(keysym) >= 0xFF90 & keysym: any integer &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             safe curfinal[keysym] = 1;
33         }
34     else
35     {
36         buffer-overrun curfinal[keysym] = 2;
37     }
38     if (keysym < 10)
39     {
40         unparseputc(curfinal[keysym], pty);
41     }
42 }
43 } safe
```

curfinal: buffer of size 10

Sparrow automatically
pinpoints the buffer-overrun bug

```

16 static char *curfinal = "HDACB  FE";
17
18 keysym = read_from_input();
19
20 if (((KeySym)(keysym) >= 0xFF90 & keysym: any integer &ym) <= 0xFF94))
21 {
22     unparseputc((char)(keysym-0xFF91 + 'P'), pty);
23     key = 1;
24 }
25 else if (keysym >= 0)
26 {
27     if (keysym < 16)           keysym: [0,15]
28     {
29         if (read_from_input())
30         {
31             if (keysym >= 10) return;
32             safe curfinal[keysym] = 1;      Sparrow automatically
33         }
34         else
35         {
36             buffer-overrun curfinal[keysym] = 2;    pinpoints the buffer-overrun bug
37         }
38     }
39     if (keysym < 10)
40     {
41         unparseputc(curfinal[keysym], pty);
42     }
43 }

```

curfinal: buffer of size 10

safe

keysym: [0,15]

safe

```

16 static char *curfinal = "HDACB  FE";
17
18 keysym = read_from_input();
19
20 if (((KeySym)keysym) >= 0xFF90) keysym: any integer [ym] <= 0xFF94))
21 {
22     unparseputc((char)(keysym-0xFF91 + 'P'), pty);
23     key = 1;
24 }
25 else if (keysym >= 0)
26 {
27     if (keysym < 16) keysym: [0,15]
28     {
29         if (read_from_input())
30         {
31             if (keysym >= 10) return;
32             safe curfinal[keysym] = 1;
33         }
34         else
35         {
36             buffer-overrun curfinal[keysym] = 2; curfinal:[10,10]
37             keysym: [10,15]
38         }
39         if (keysym < 10)
40         {
41             unparseputc(curfinal[keysym], pty);
42         }
43     }
44     safe

```

curfinal:buffer of size 10

keysym: [0,15]

Sparrow automatically pinpoints the buffer-overrun bug

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

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



++: Research Internship Positions in Program Analysis @ Google

We have a number of research internship openings in 2015. Internships could be full-time or part-time, and could be done at our Mountain View, CA or New York City, NY office. The choice is up to the candidate's preference. 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++) programming skills.



Google



facebook.

- D
- I
- C
- R
- Ideal ca

Internship Positions in Program Analysis @ Google

research internship openings in 2015. Internships could



Infer

A tool to detect bugs in Android and iOS apps before they
ship

Google

facebook.



- Design
- Identify
- Create
- Review
- Ideal candidate

A tool to do

Internship Positions in Program Analysis @ Google

internship openings in 2015. Internships could

Static Code Analysis

potential bugs—in the source code of a project with the static analyzer built into Xcode. Source code may have subtle errors that the 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`.

Screenshot of Xcode showing the `SKTText.m` file. The code editor displays the following snippet:

```
SKTText.m
357 //textView setAllowsUndo:YES];
358 // This kind of graphic shouldn't appear opaque just be
359 [textView setDrawsBackground:NO];
360 /*
361 * This is has been handy for debugging text editing vi
362 [textView setBackgroundColor:[NSColor greenColor]];
363 textView setDrawsBackground:YES];
364 */
365 // Start off with the all of the text selected.
366 [textView setSelectedRange:NSMakeRange(0, [contents len
367 // Specify that the text view should grow and shrink to
368 removed, but only in the vertical direction. With t
369 be large enough to show an extra line fragment but
370 able to see just-typed text on the screen. Sending
371 view without also sending -setMinSize: or -setMaxSi
372 default minimum and maximum sizes of a text view ar
373 at initialization time.
374 [textView setMinSize:CGSizeMake(bounds.size.width, 0.0)
375 [textView setMaxSize:CGSizeMake(bounds.size.width, supe
376 )];
377 [textView setVerticallyResizable:YES];
```



- D
- I
- C
- R
- Ideal ca

A tool to d

hip Positions in Program Analysis @ Google

internship openings in 2015. Internships could



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.

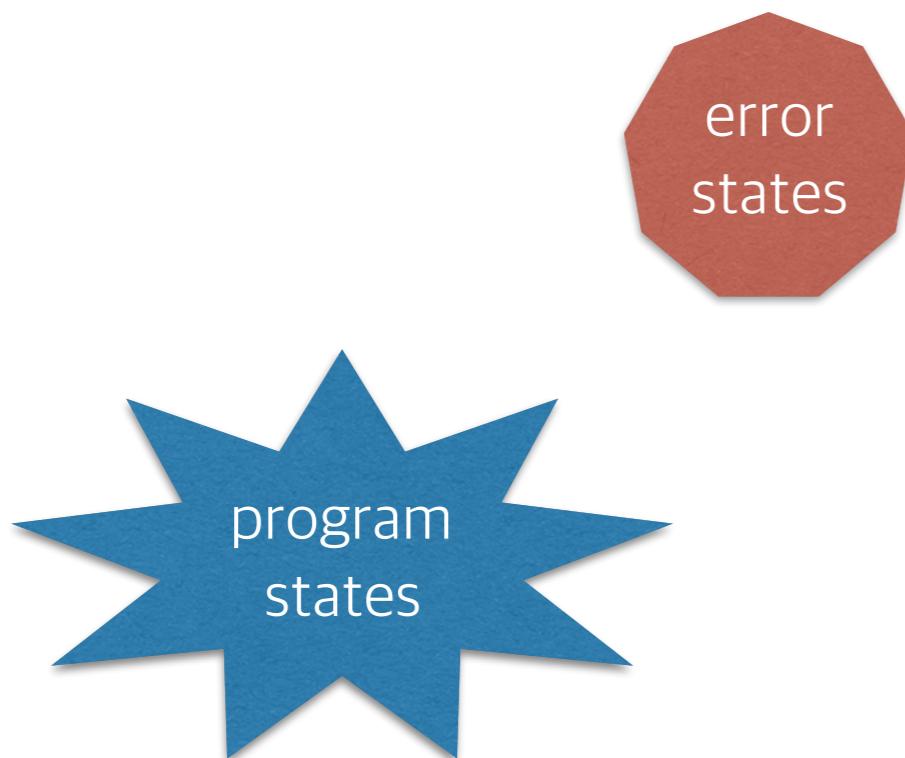
Our Research

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

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

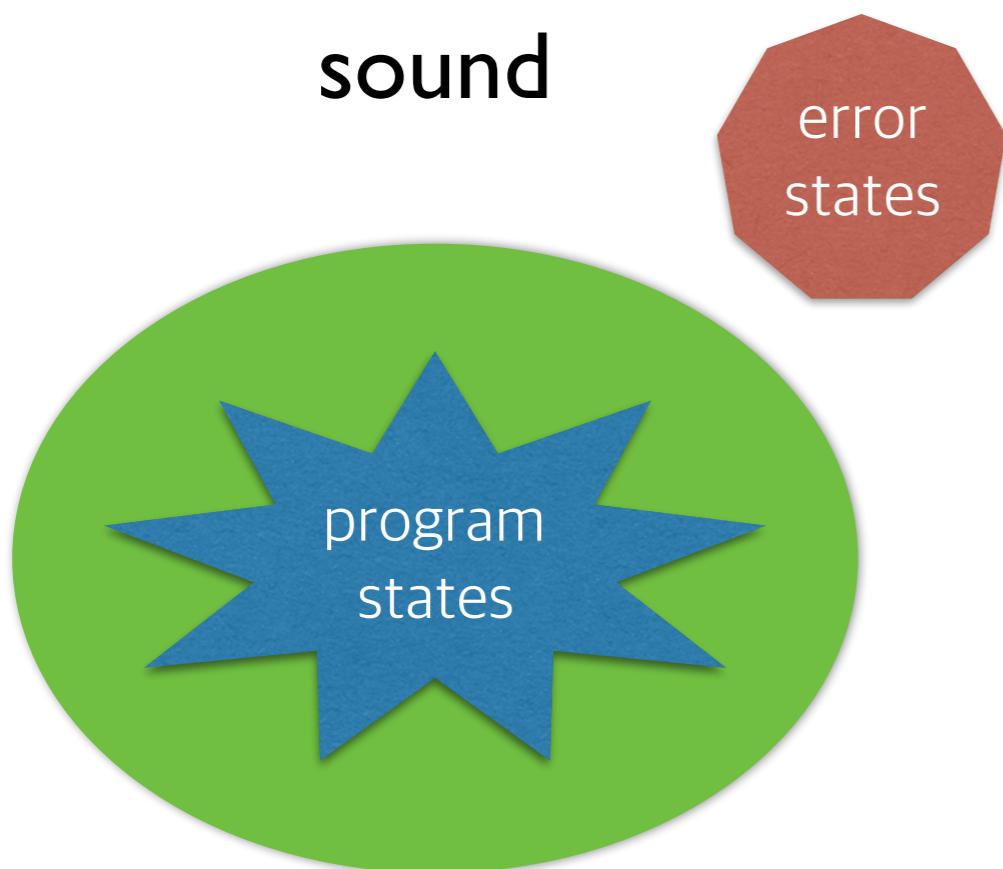
(I) Soundness

Find all bugs / verify absence



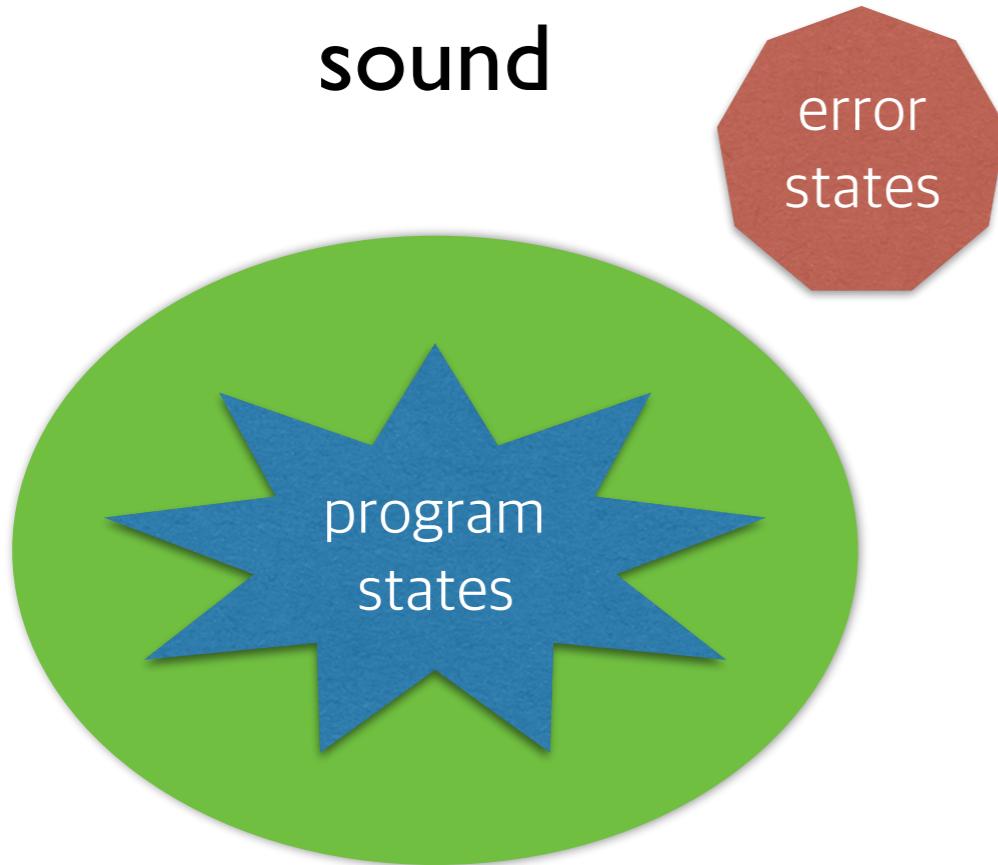
(I) Soundness

Find all bugs / verify absence

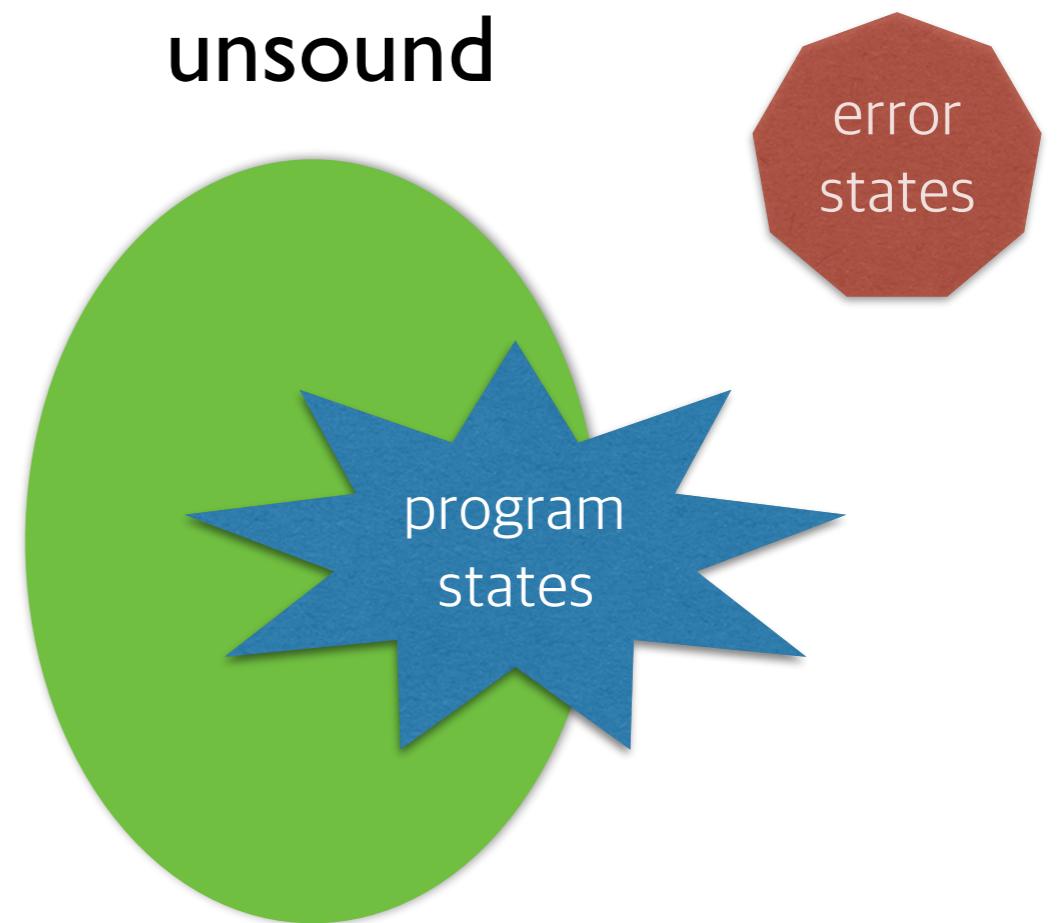


(I) Soundness

Find all bugs / verify absence



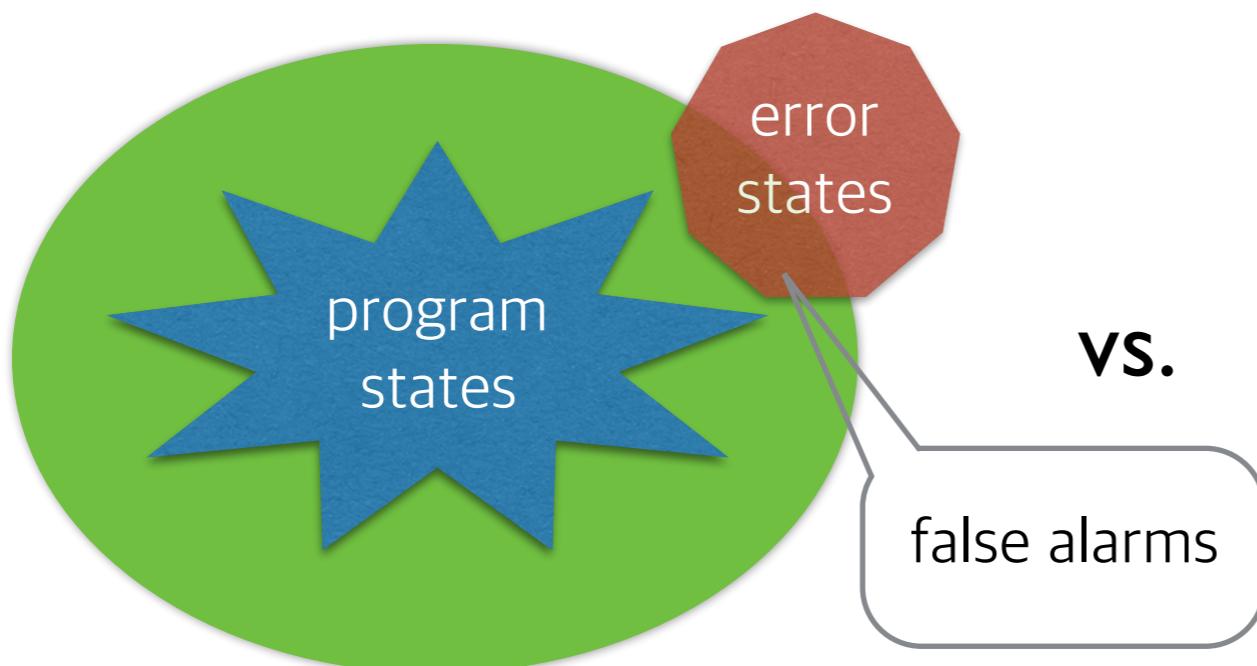
vs.



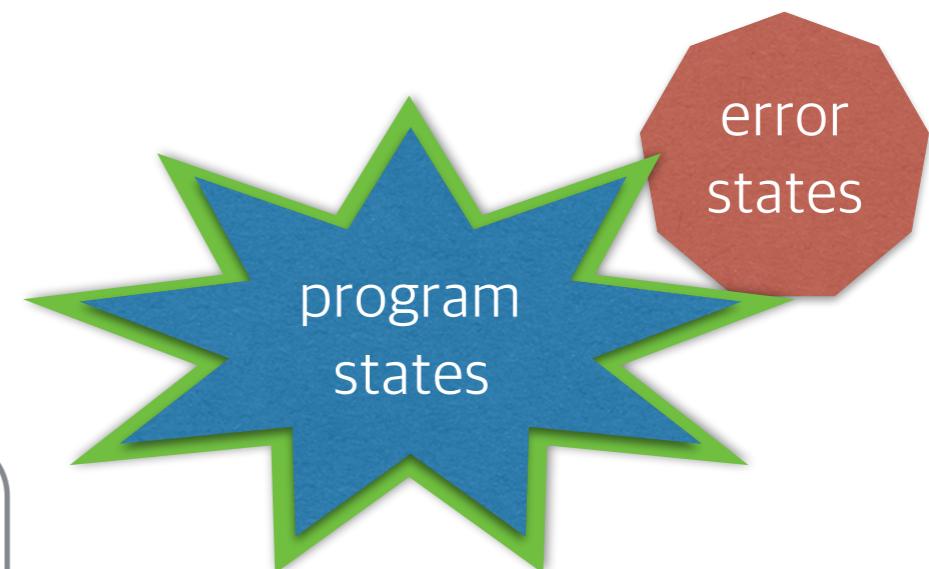
(2) Precision

Few false alarms

imprecise



precise



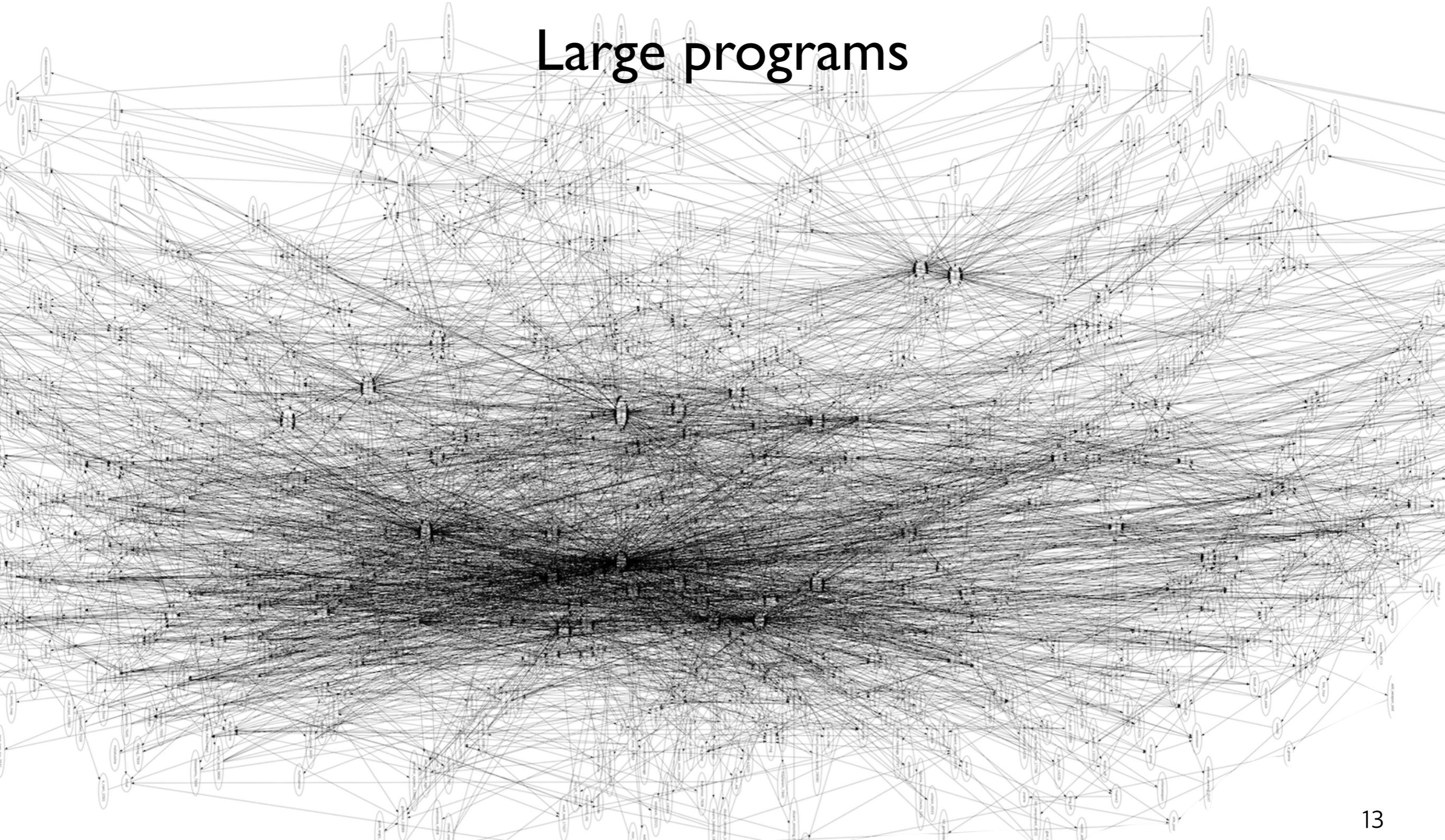
vs.

false alarms

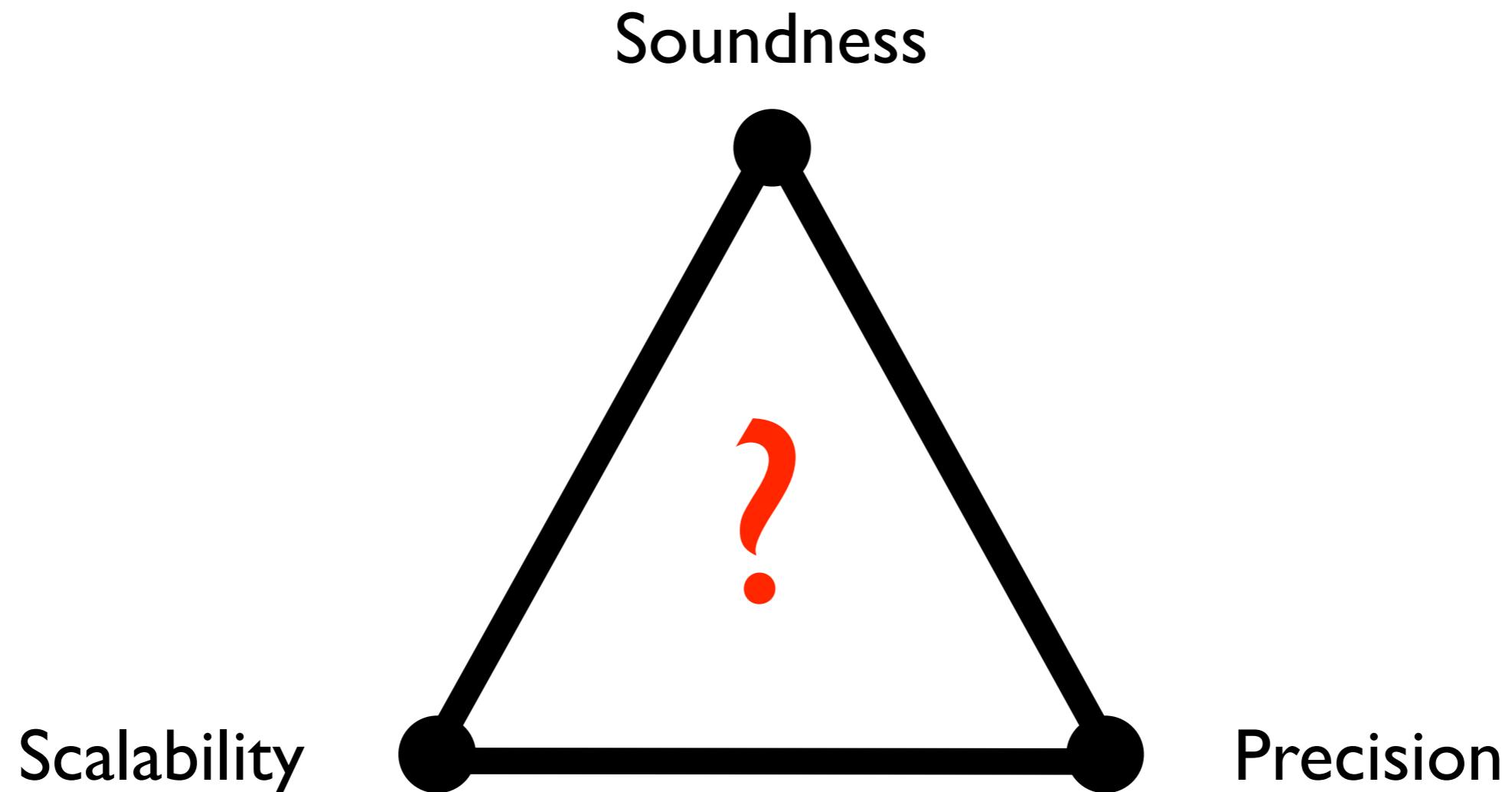
(3) Scalability

nethack-3.3.0 (211KLoC)

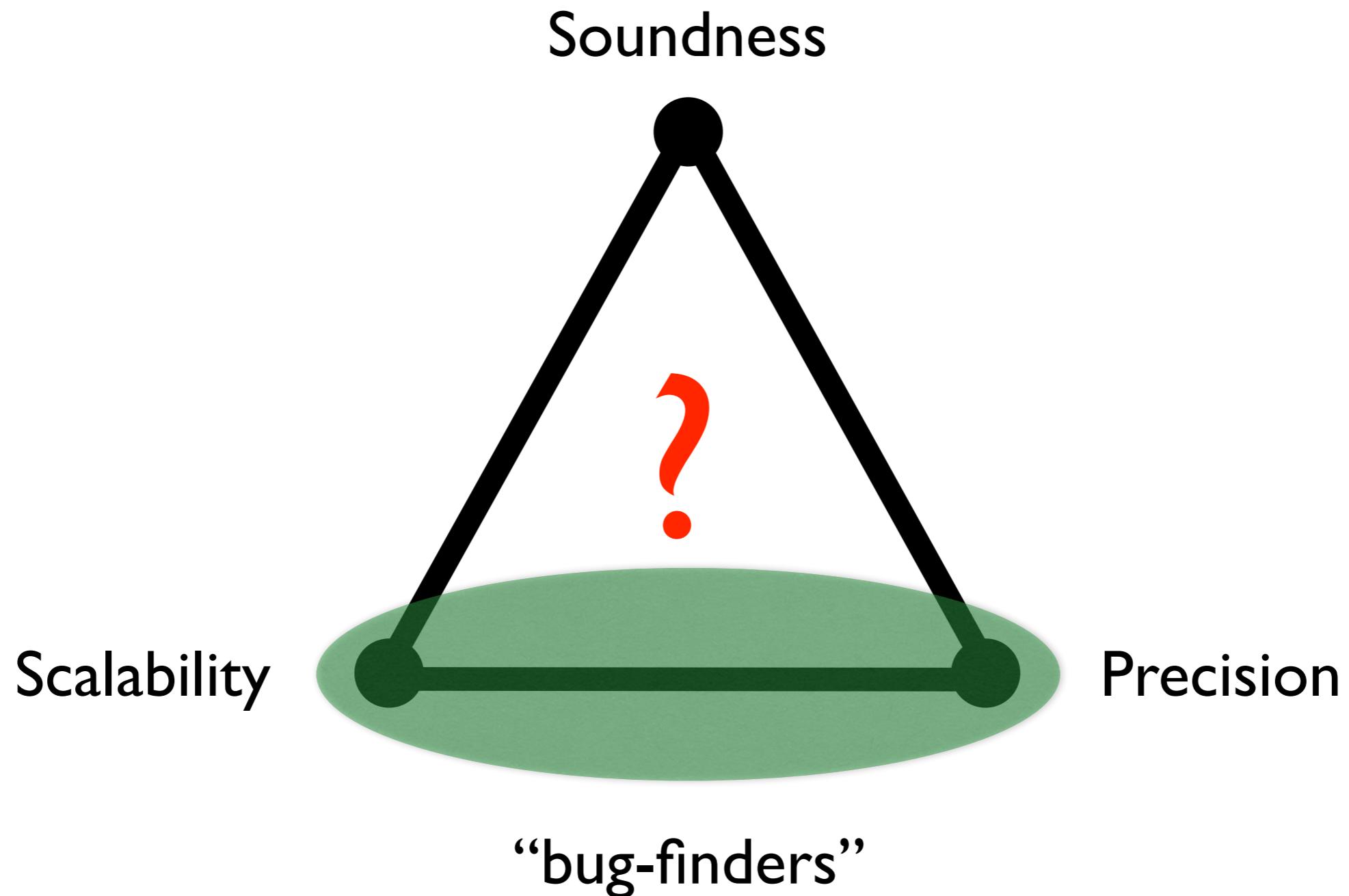
Large programs



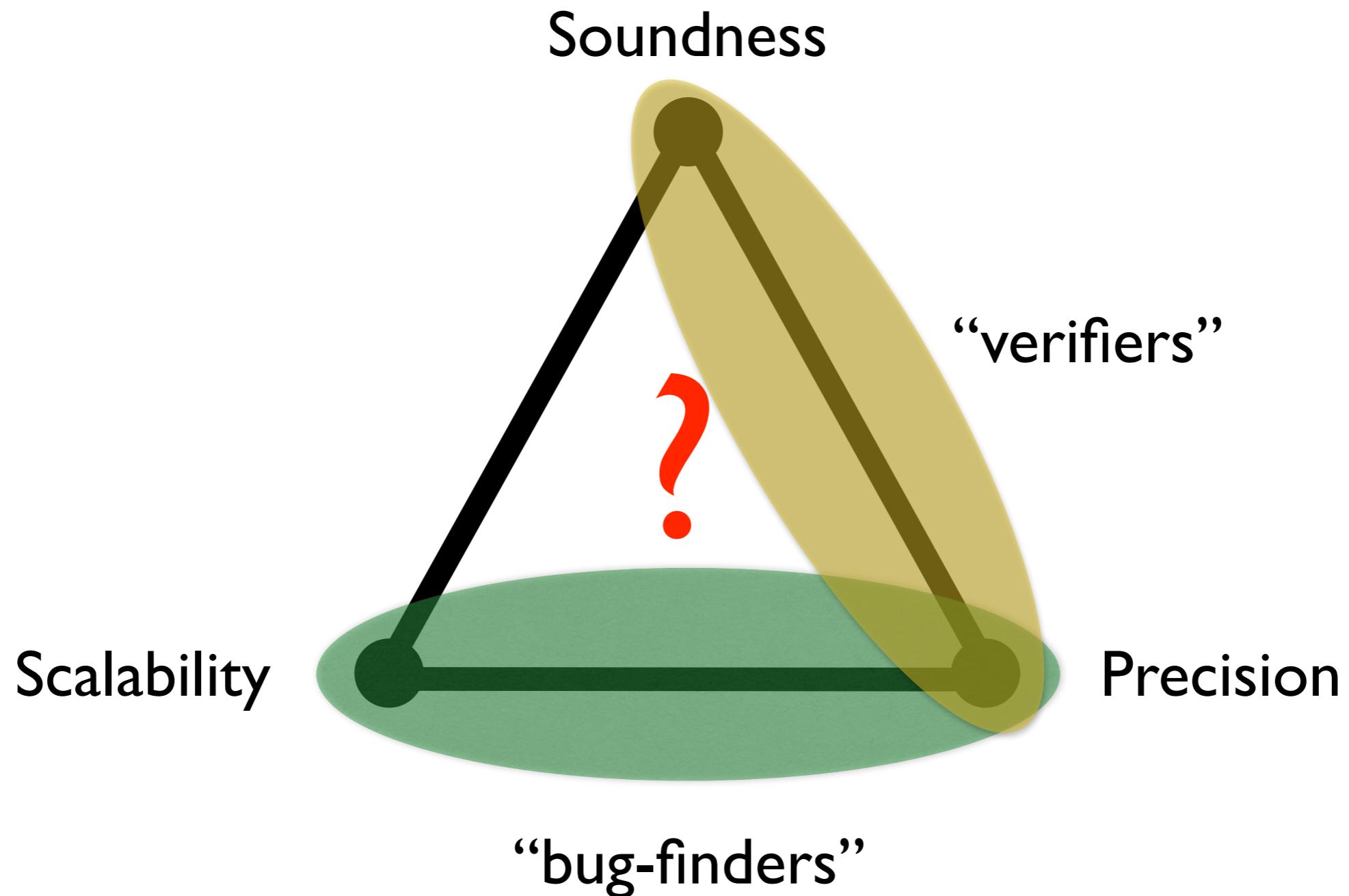
Challenge in Static Analysis



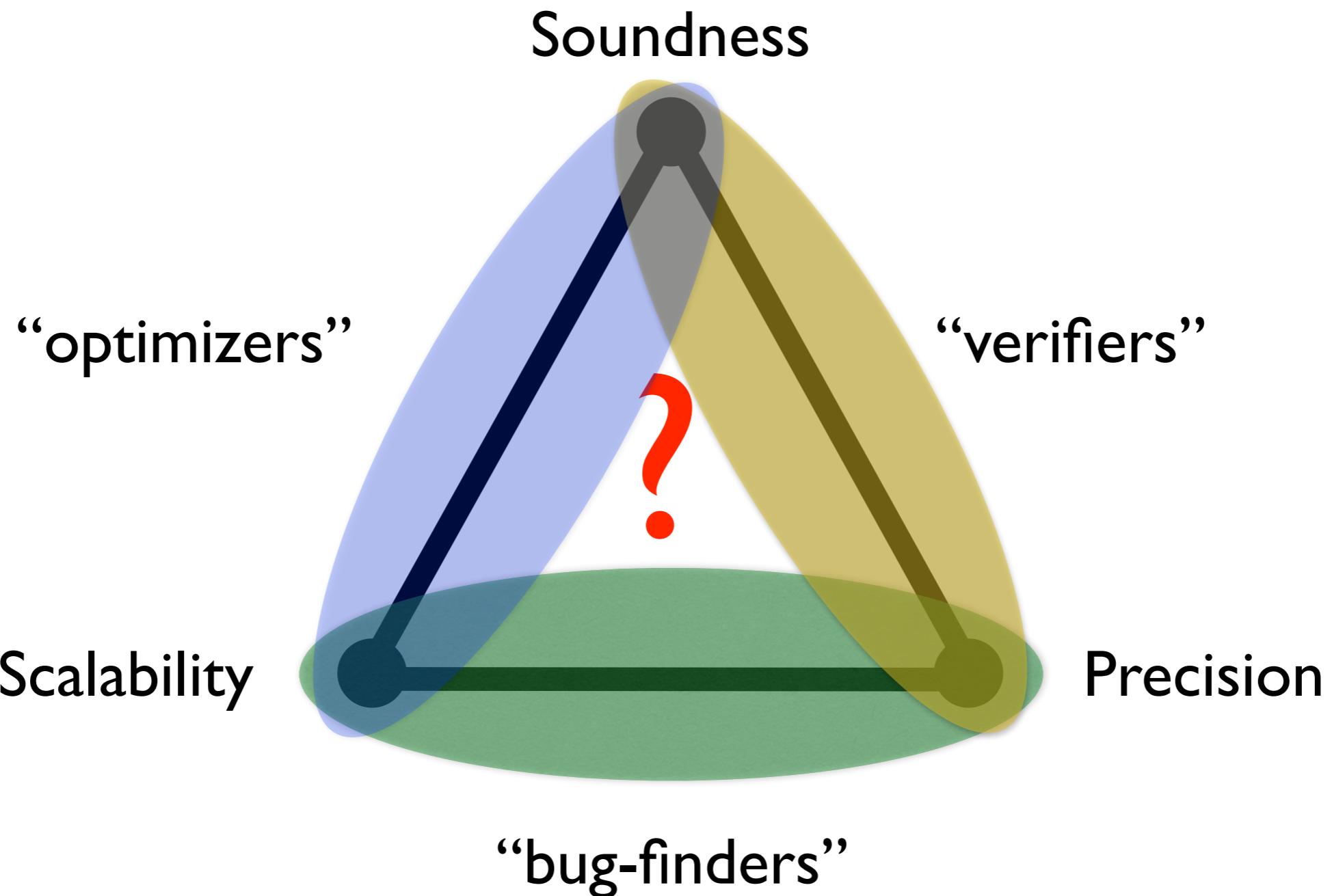
Challenge in Static Analysis



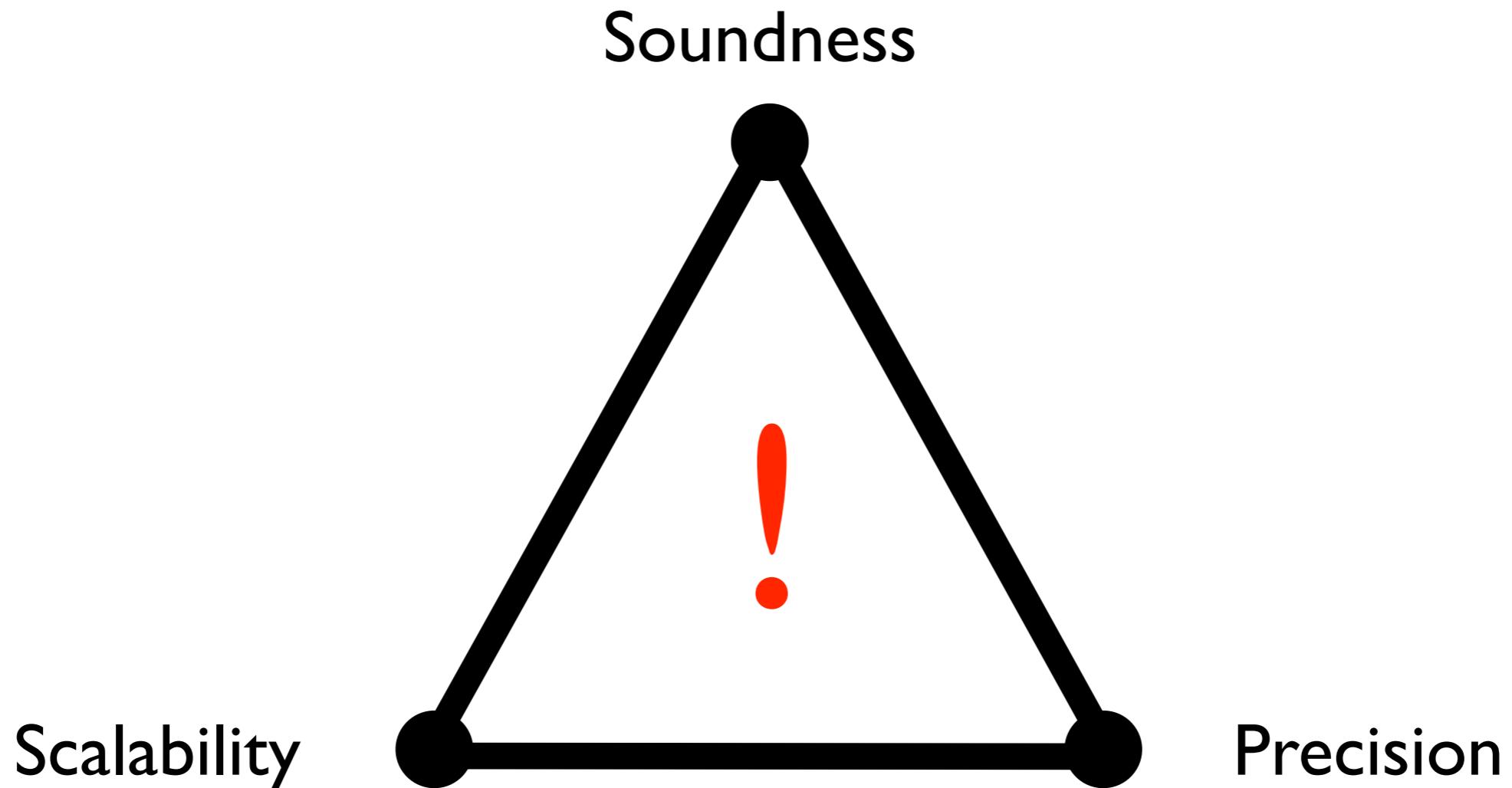
Challenge in Static Analysis



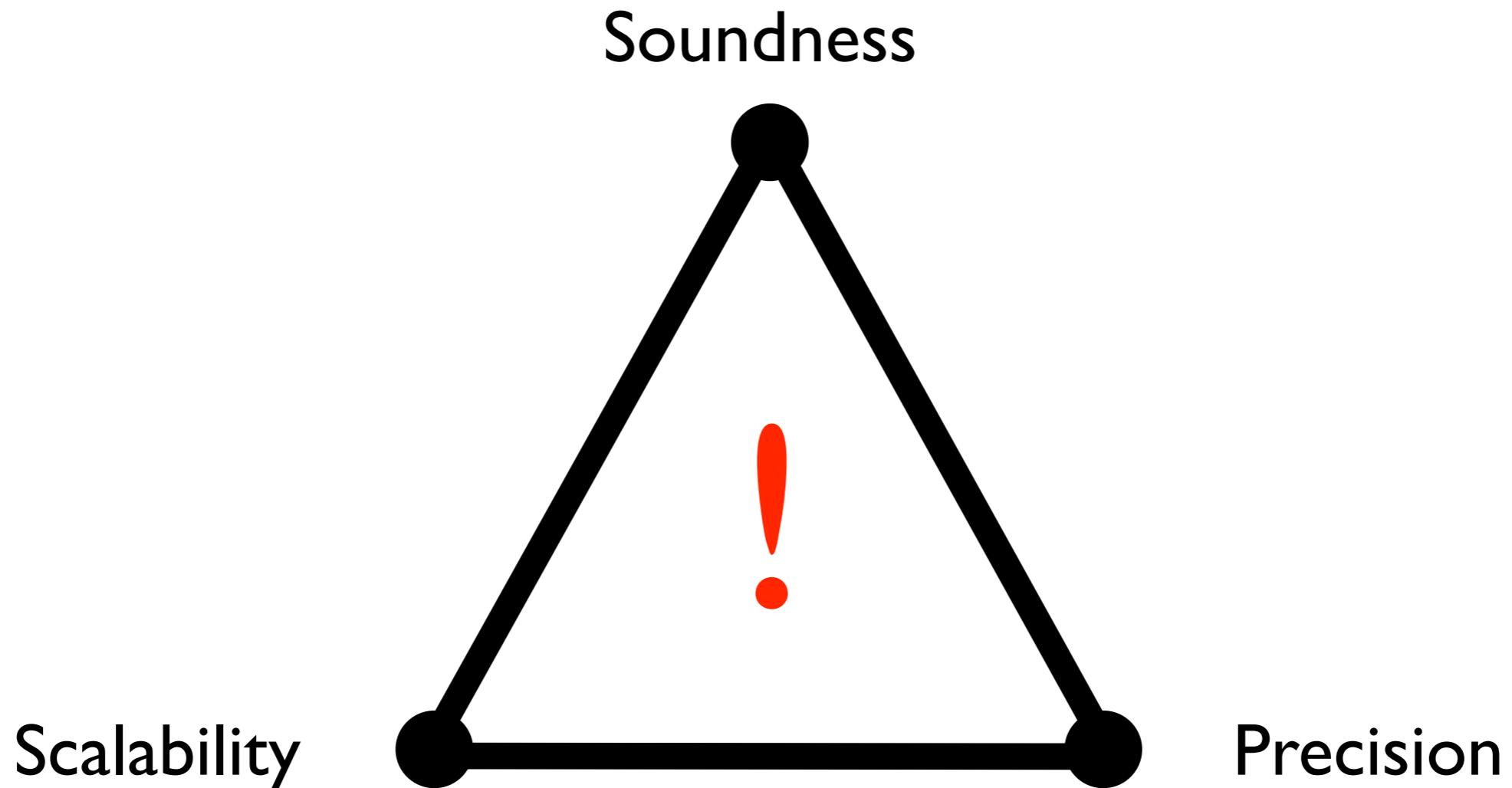
Challenge in Static Analysis



Our Research Goal

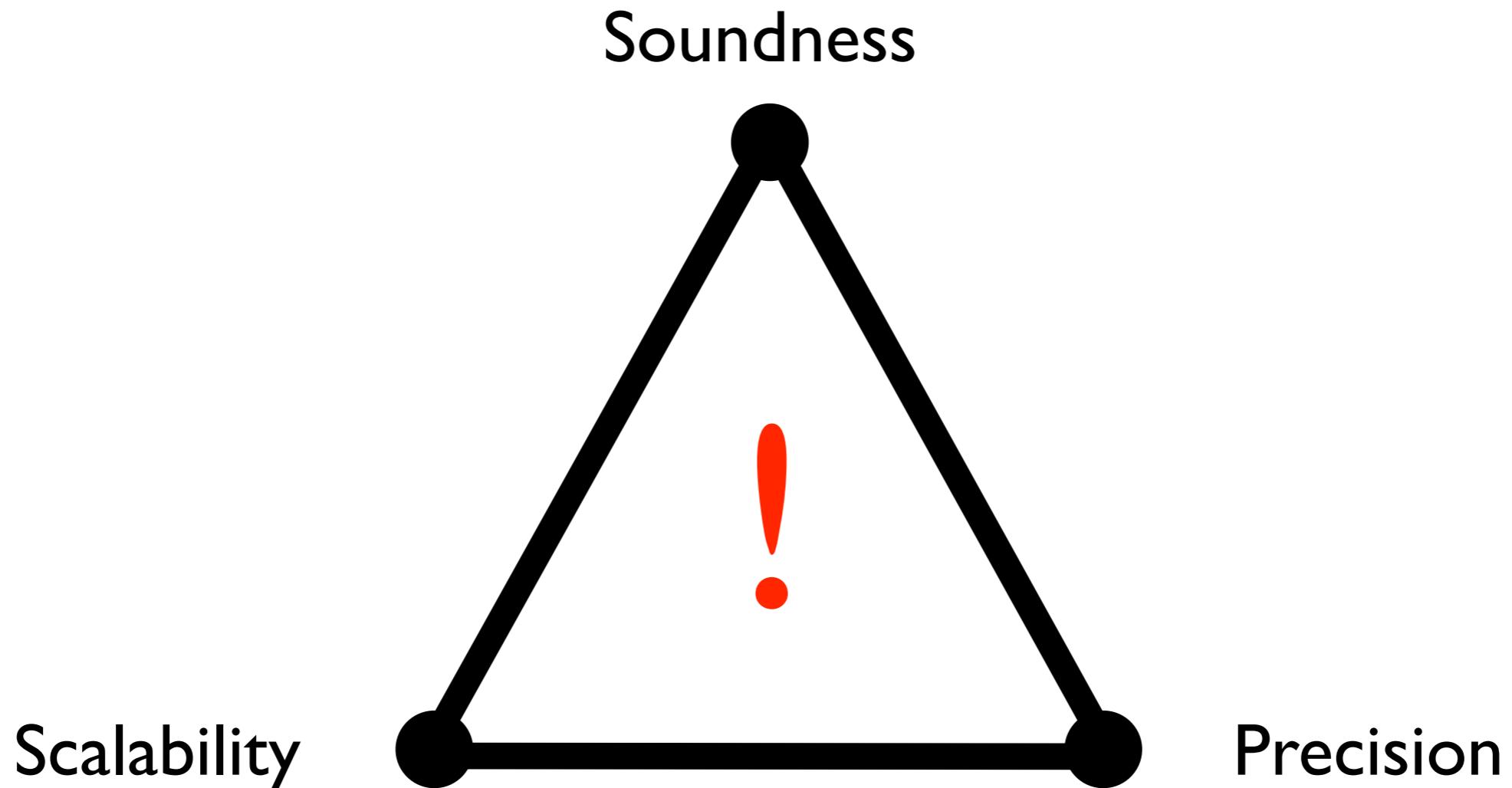


Our Research Goal



**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

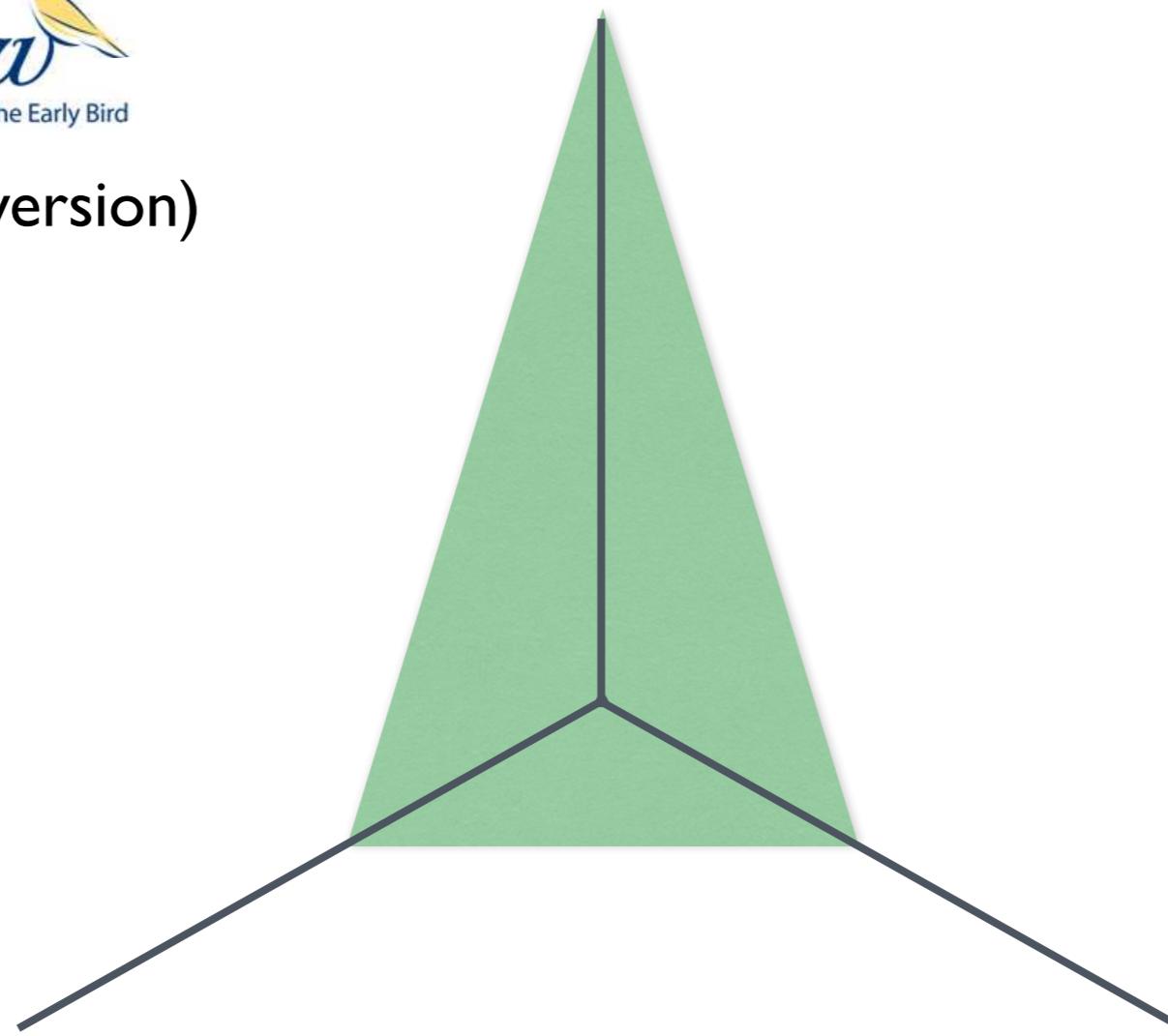


(sound-&-global version)

Soundness

Scalability

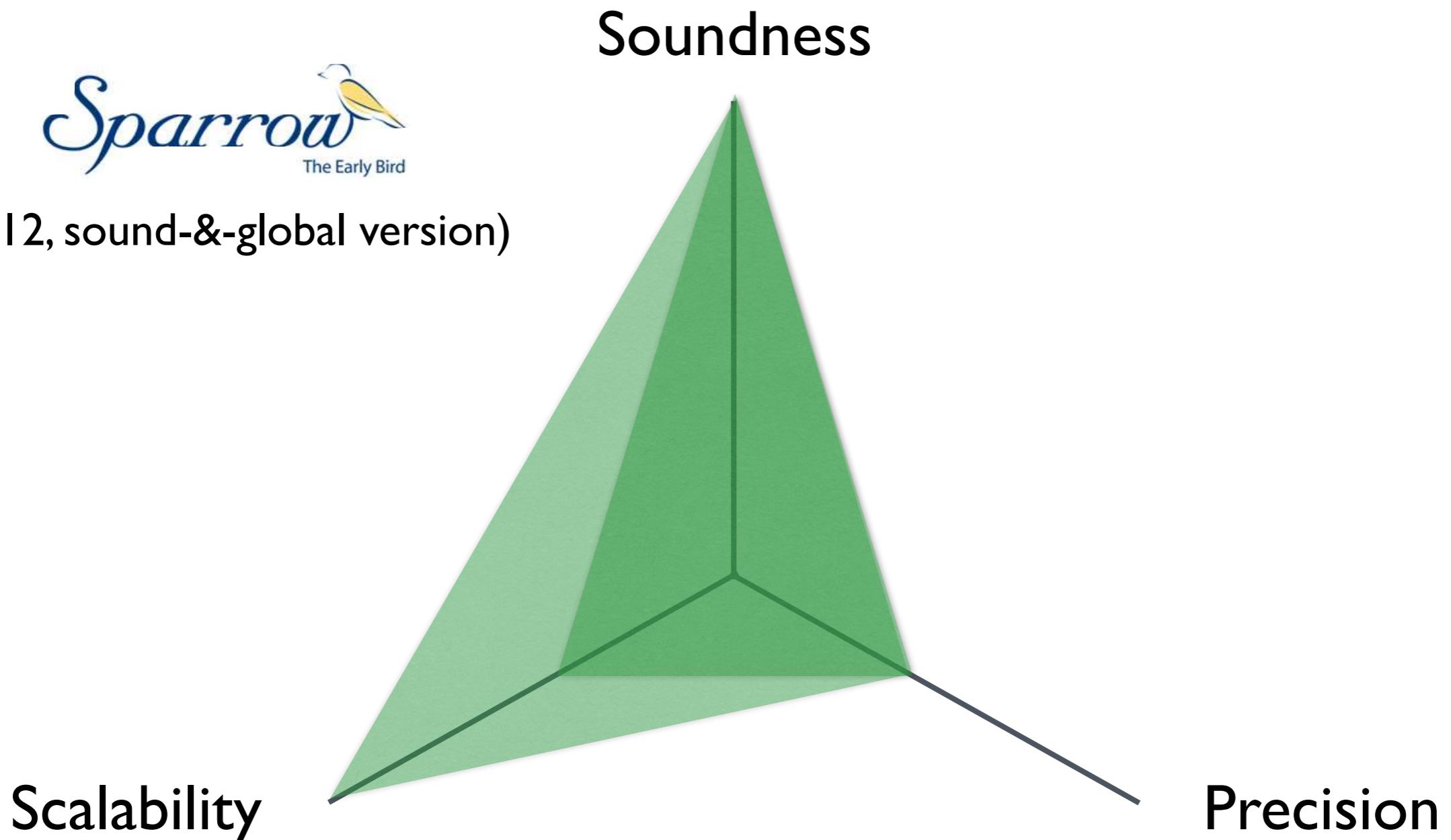
Precision



The First Goal: Scalability



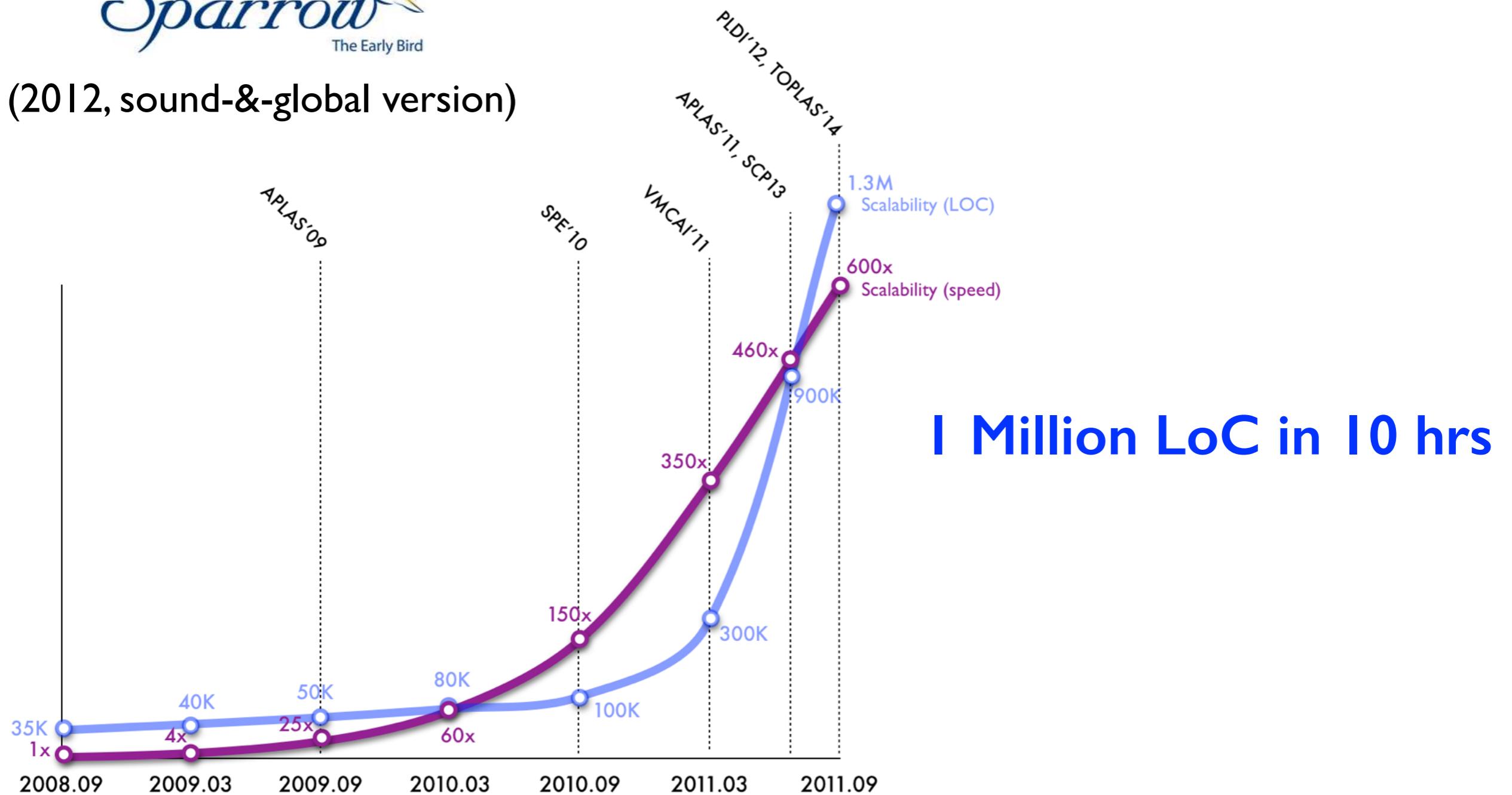
(2012, sound-&-global version)



Scalability Improvement

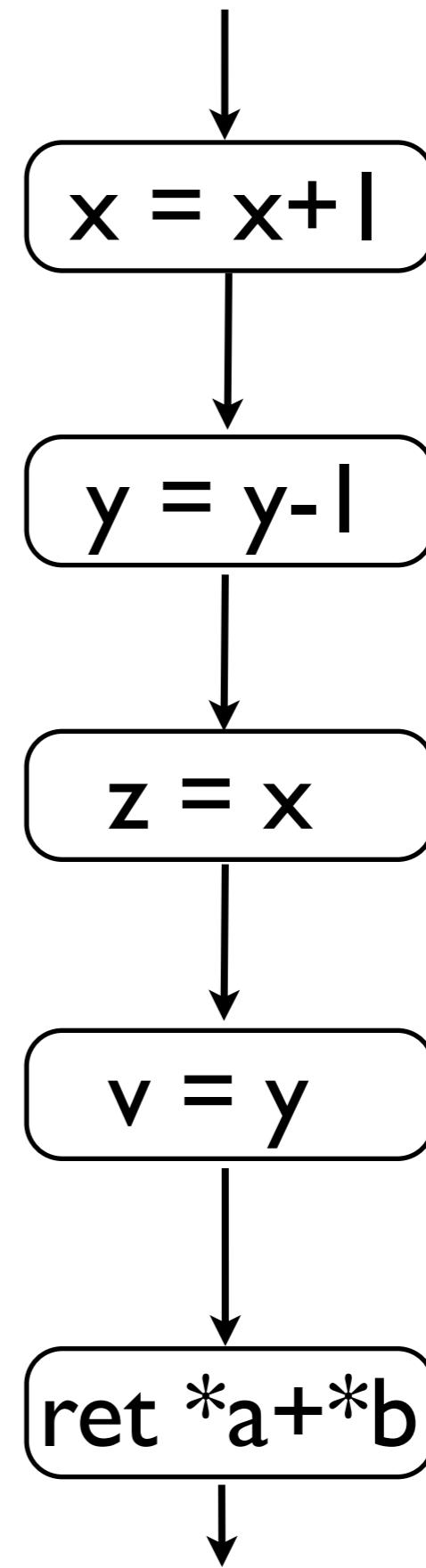


(2012, sound-&-global version)



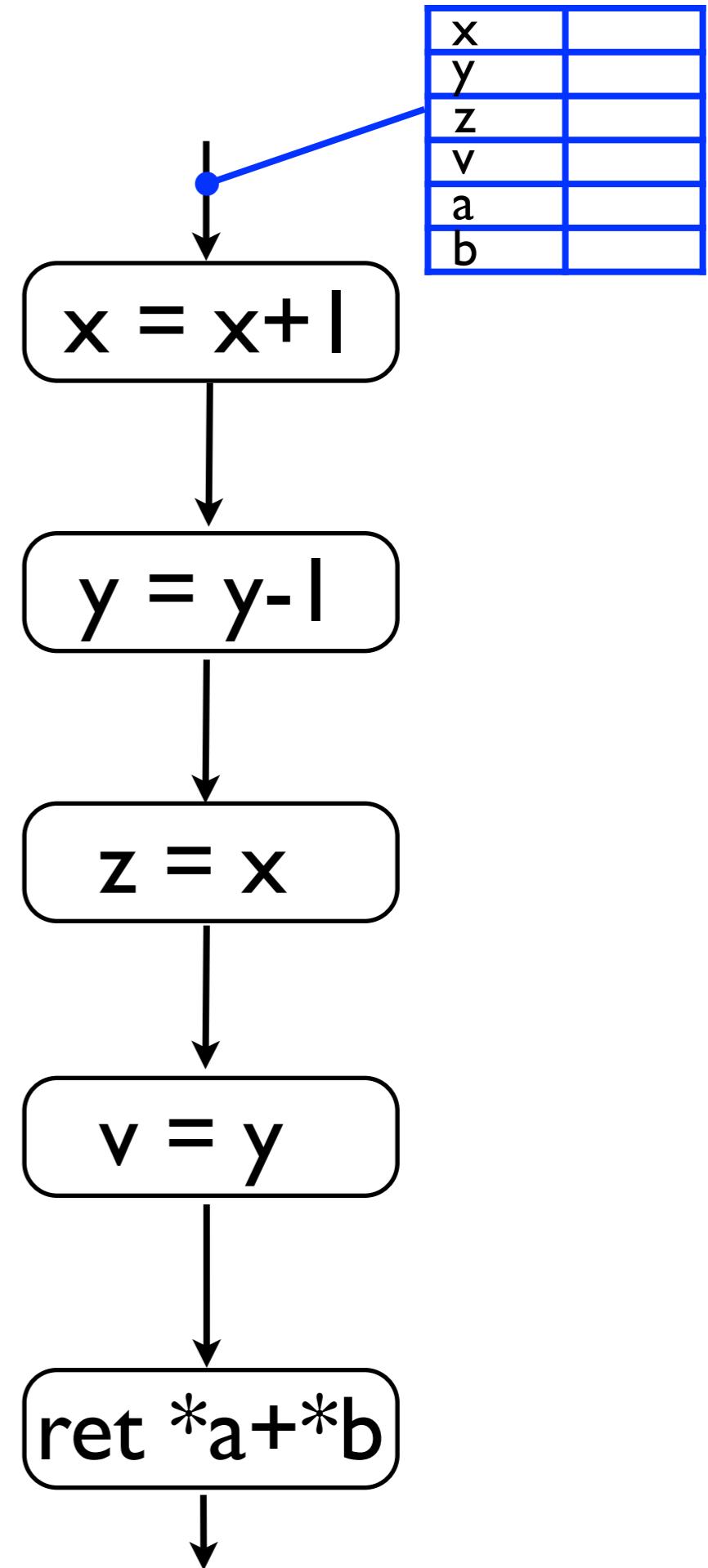
Key: General Sparse Analysis

“Right Part at Right Moment”



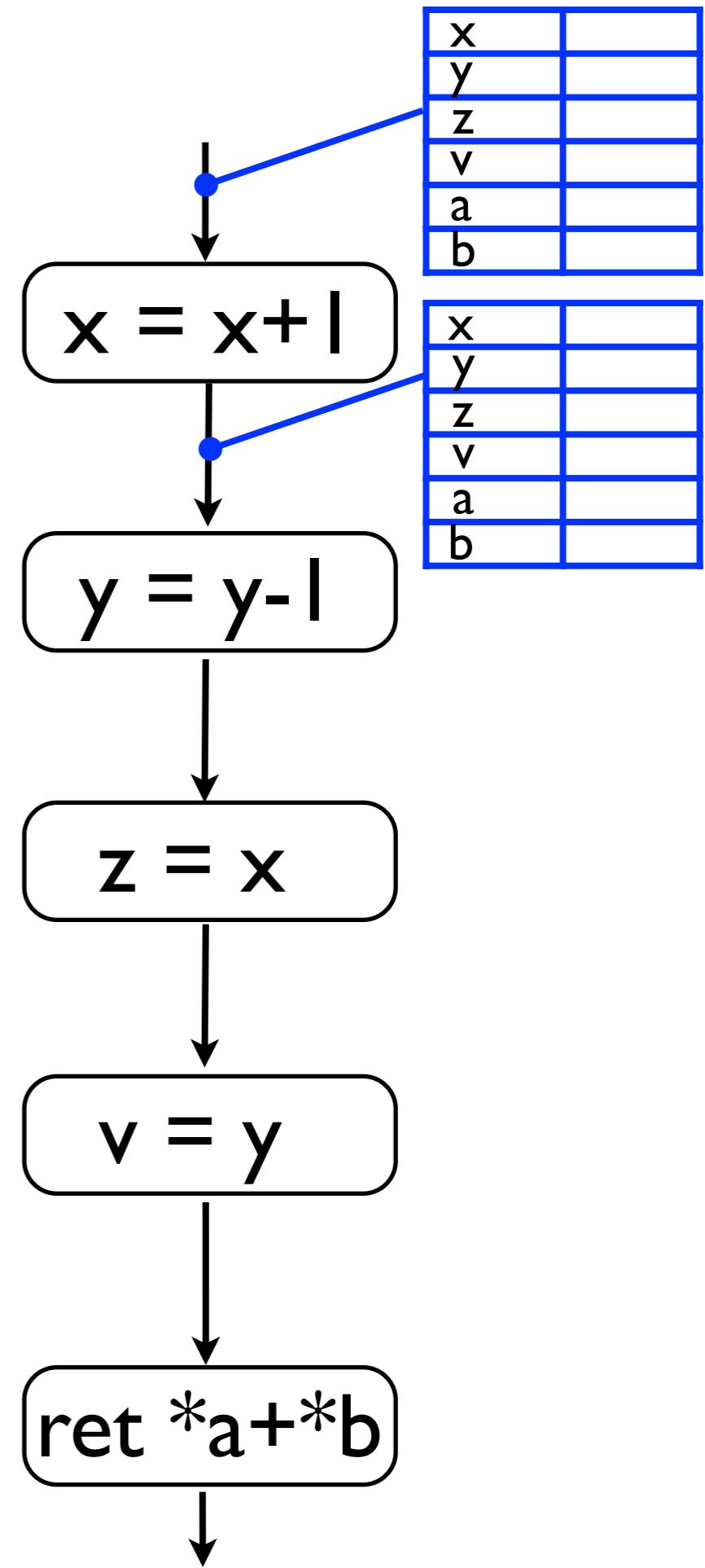
Key: General Sparse Analysis

“Right Part at Right Moment”



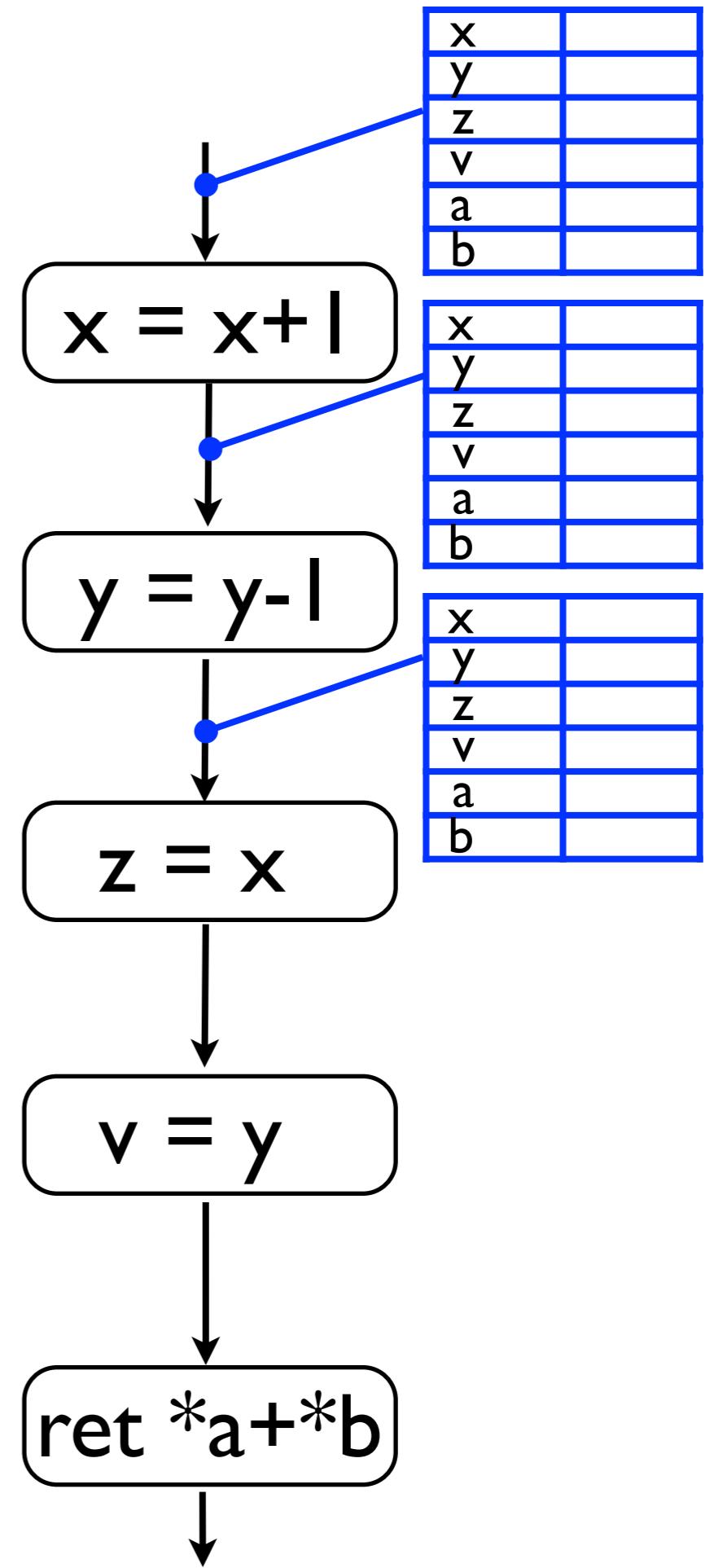
Key: General Sparse Analysis

“Right Part at Right Moment”



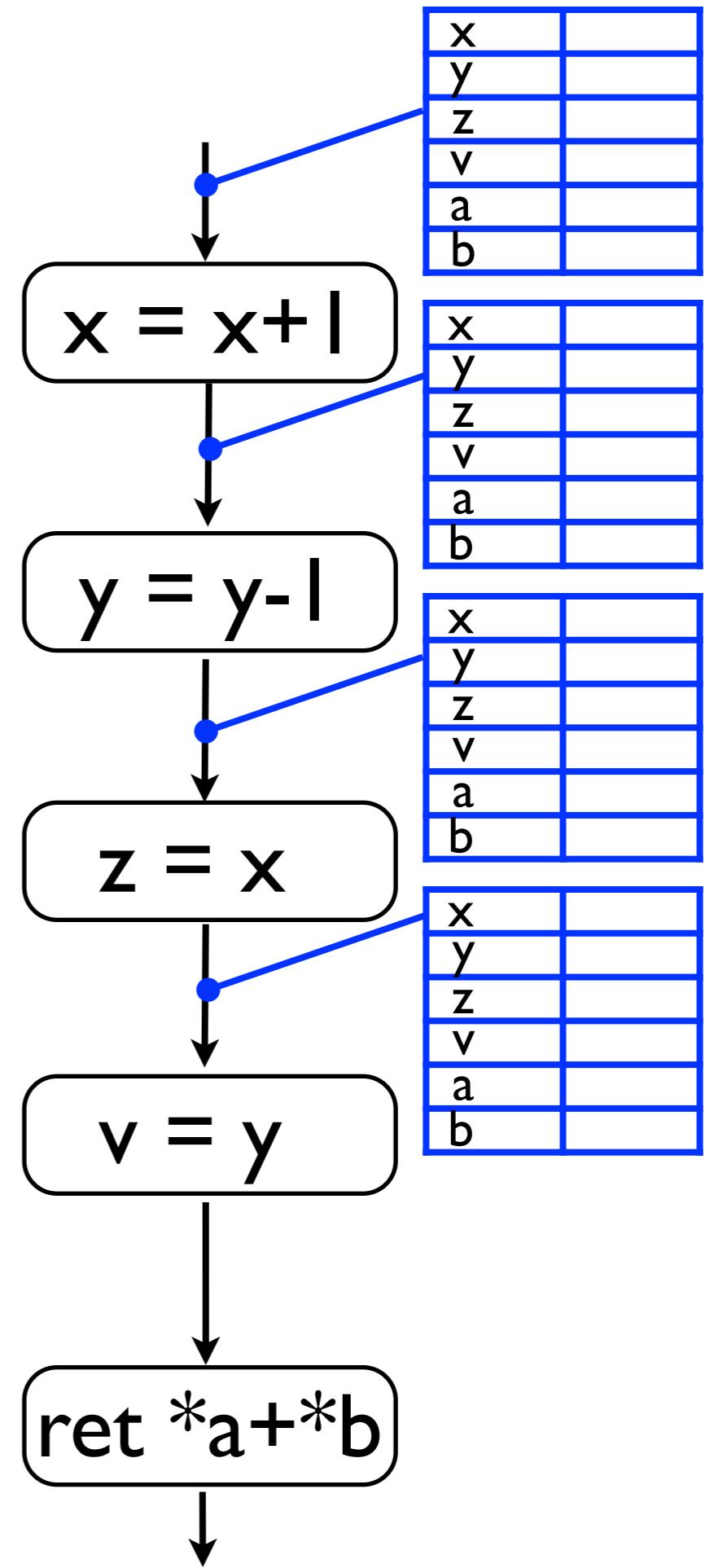
Key: General Sparse Analysis

“Right Part at Right Moment”



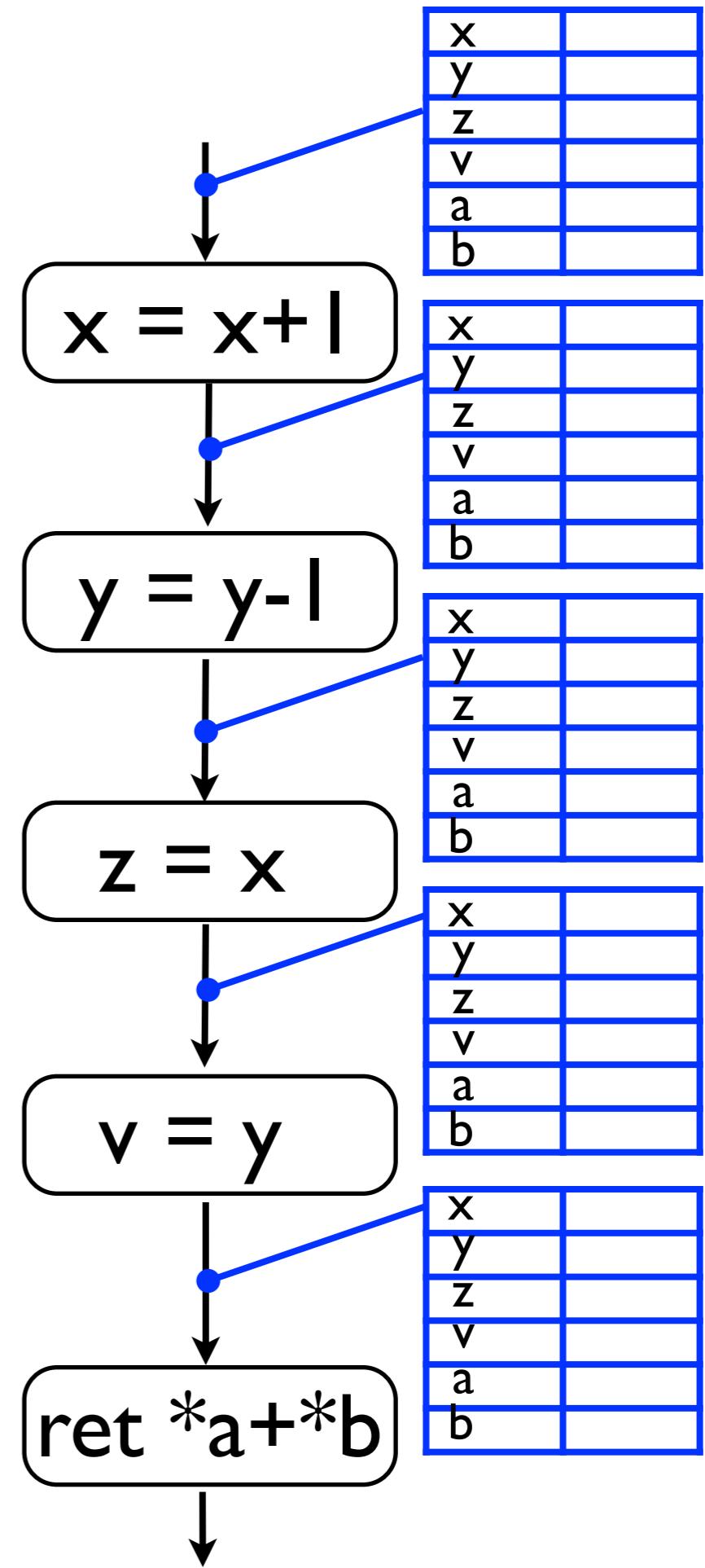
Key: General Sparse Analysis

“Right Part at Right Moment”



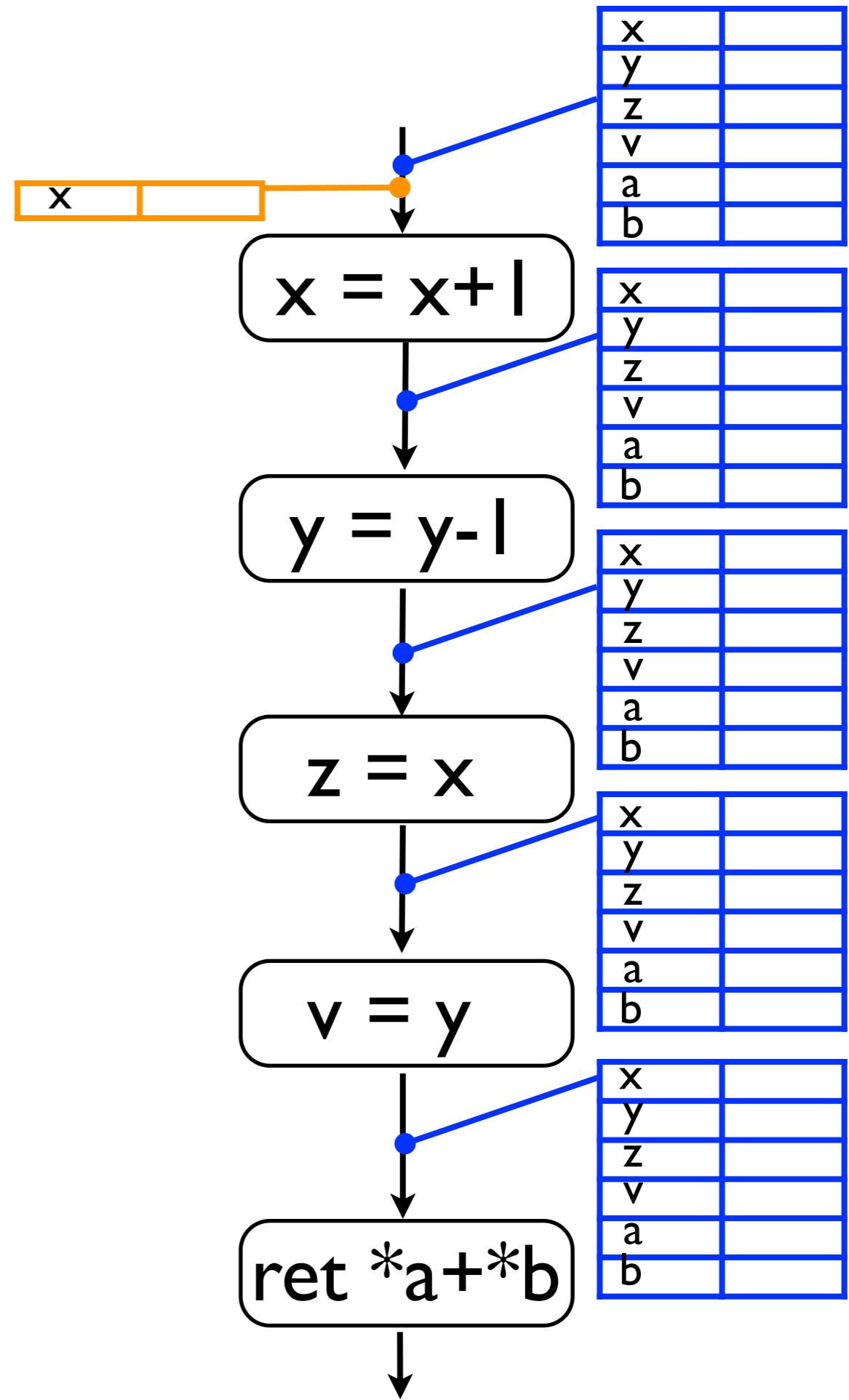
Key: General Sparse Analysis

“Right Part at Right Moment”



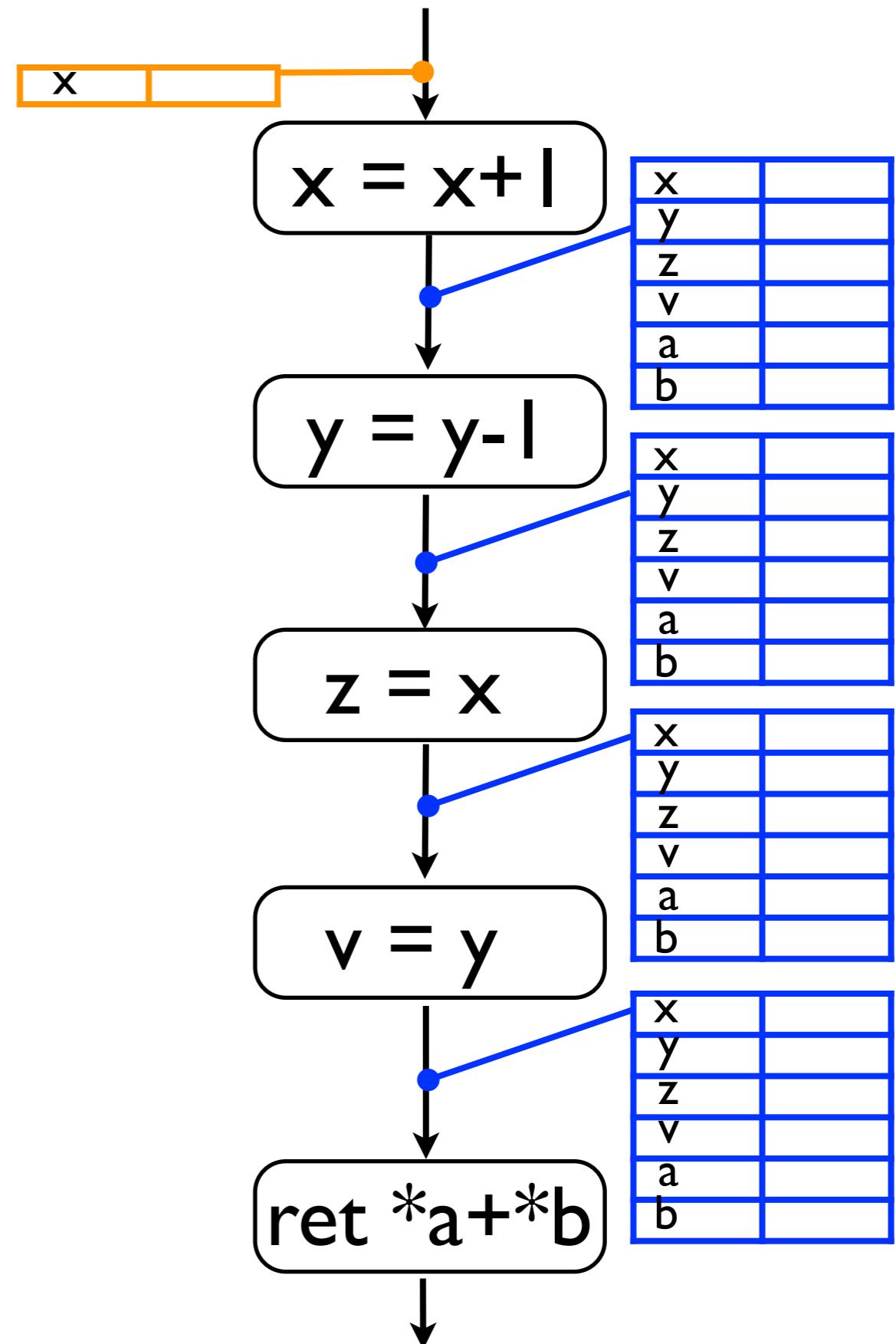
Key: General Sparse Analysis

“Right Part at Right Moment”



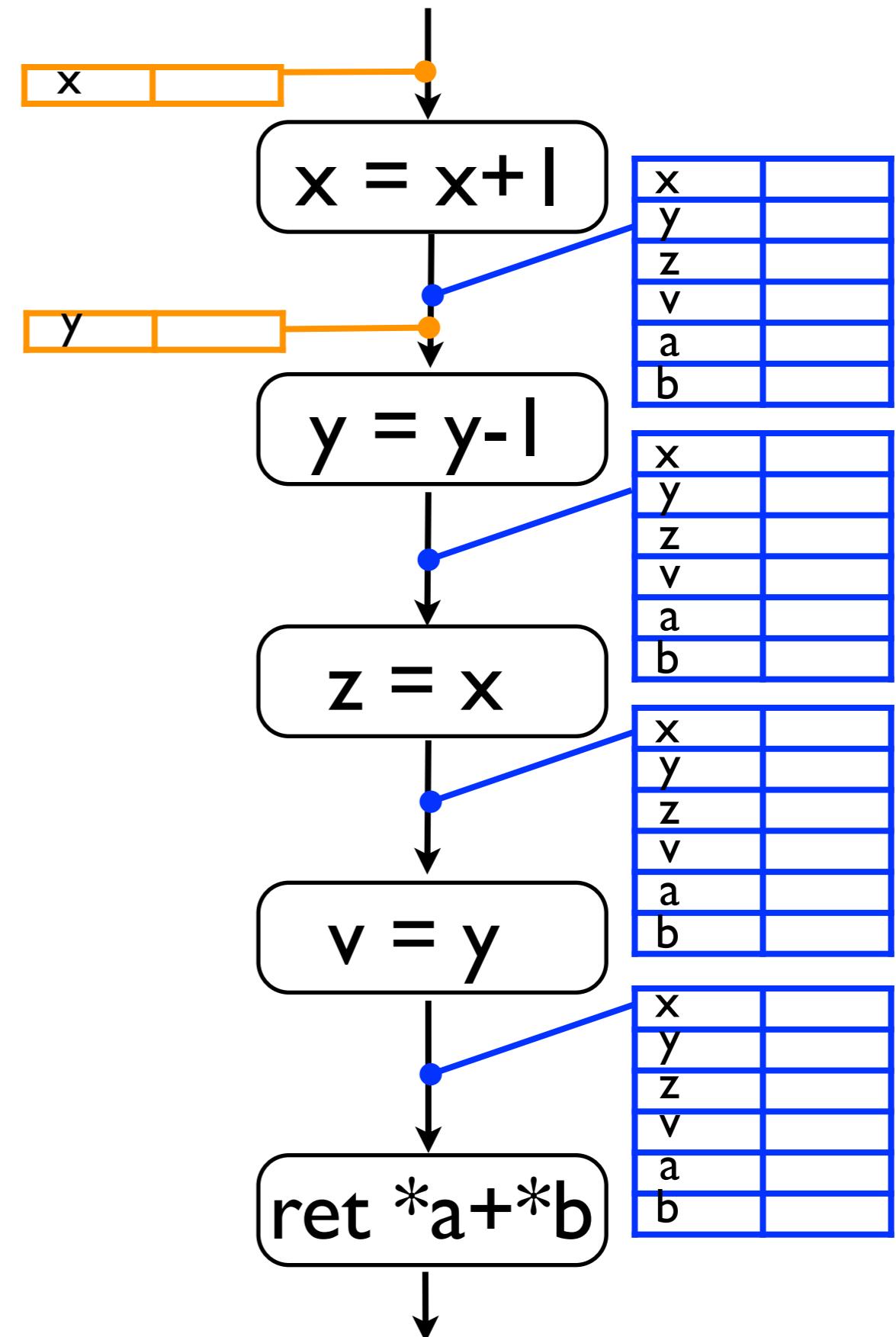
Key: General Sparse Analysis

“Right Part at Right Moment”



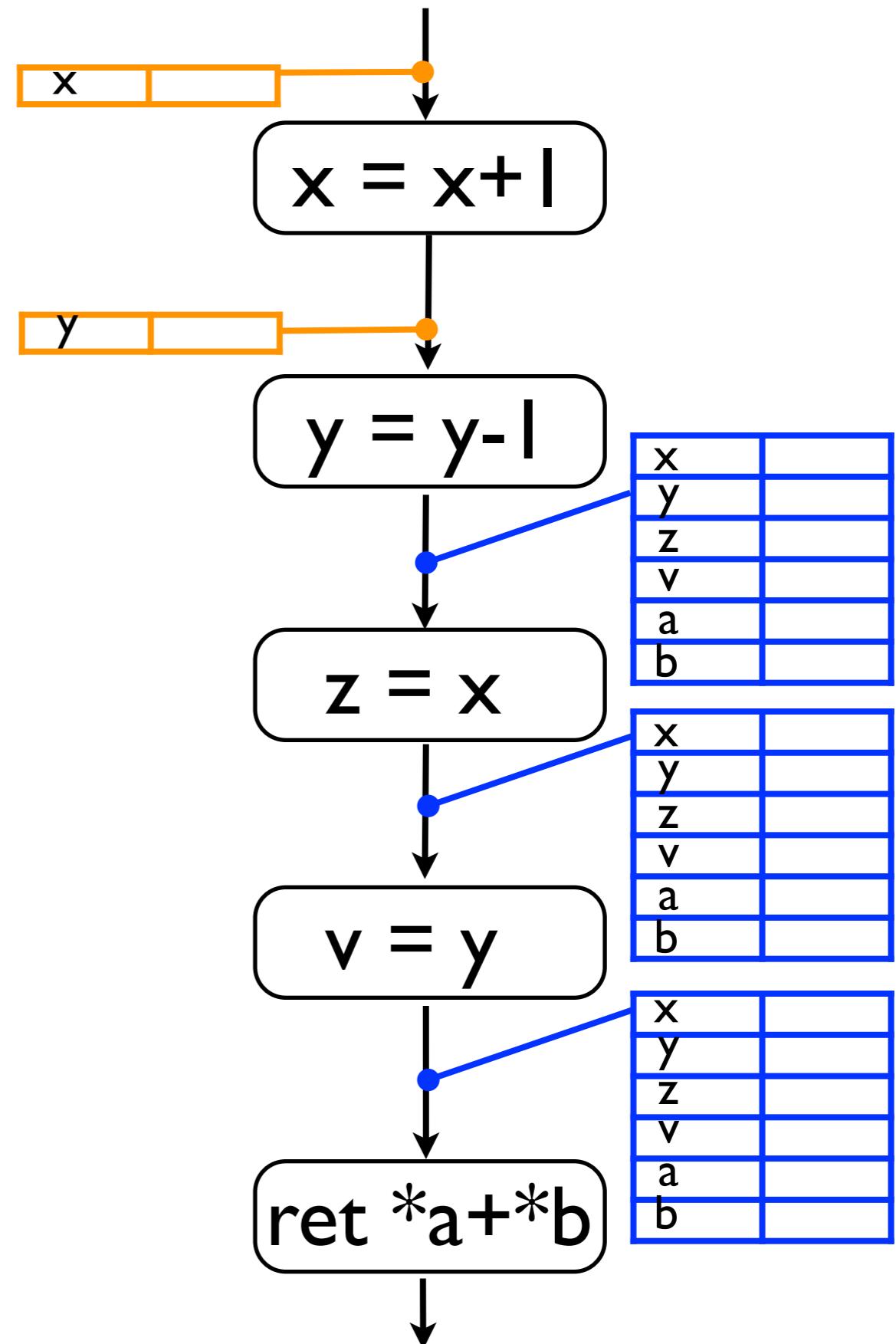
Key: General Sparse Analysis

“Right Part at Right Moment”



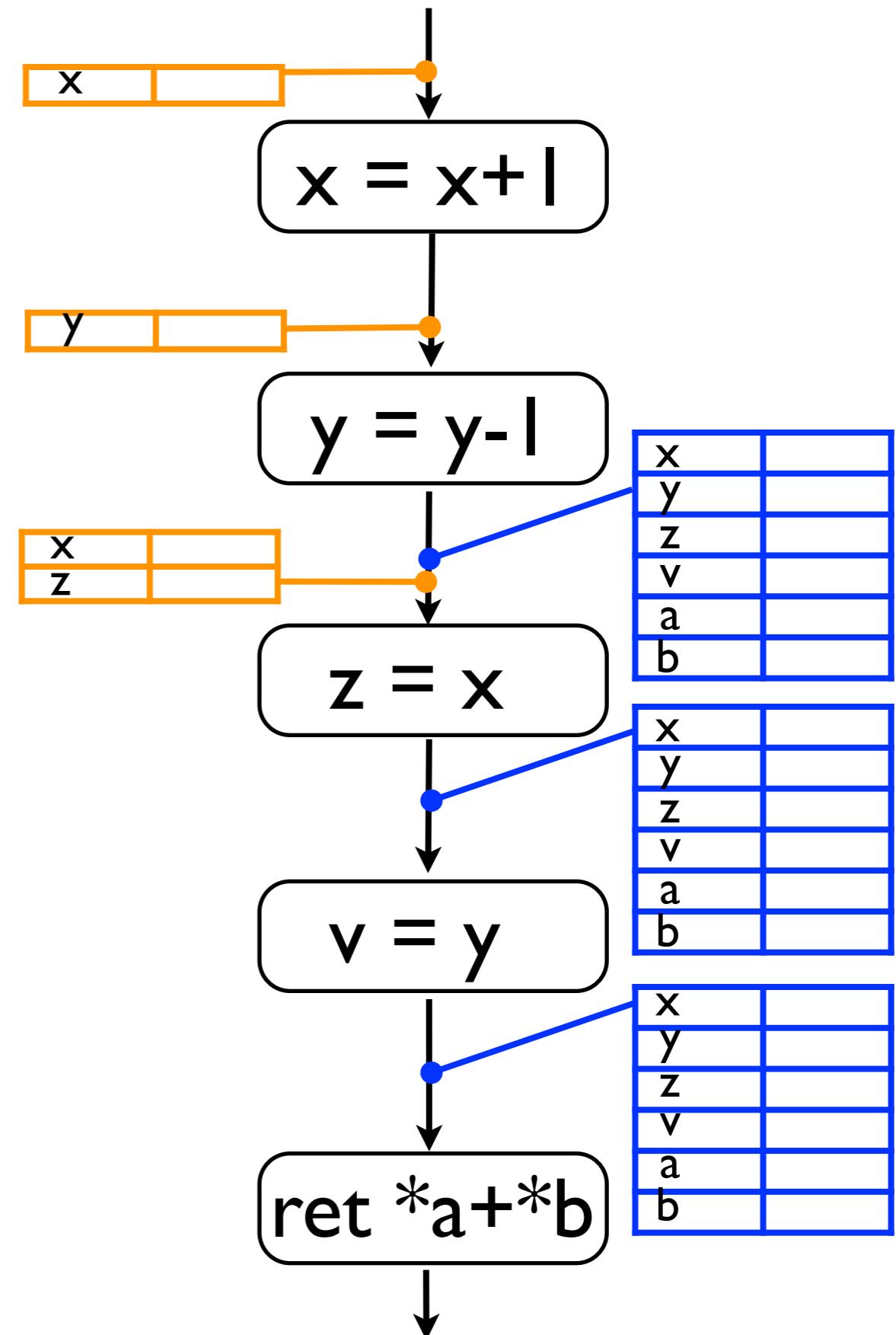
Key: General Sparse Analysis

“Right Part at Right Moment”



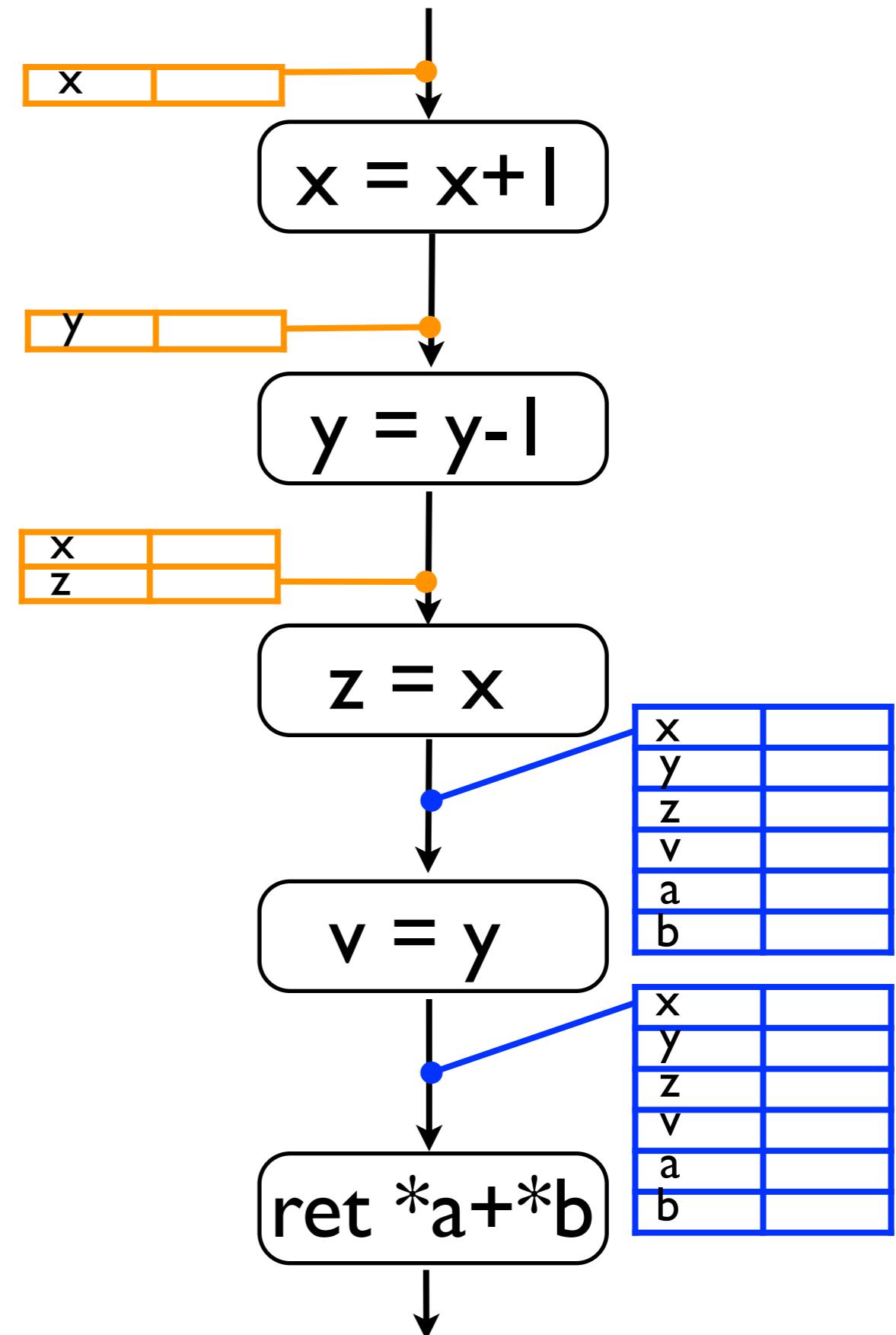
Key: General Sparse Analysis

“Right Part at Right Moment”



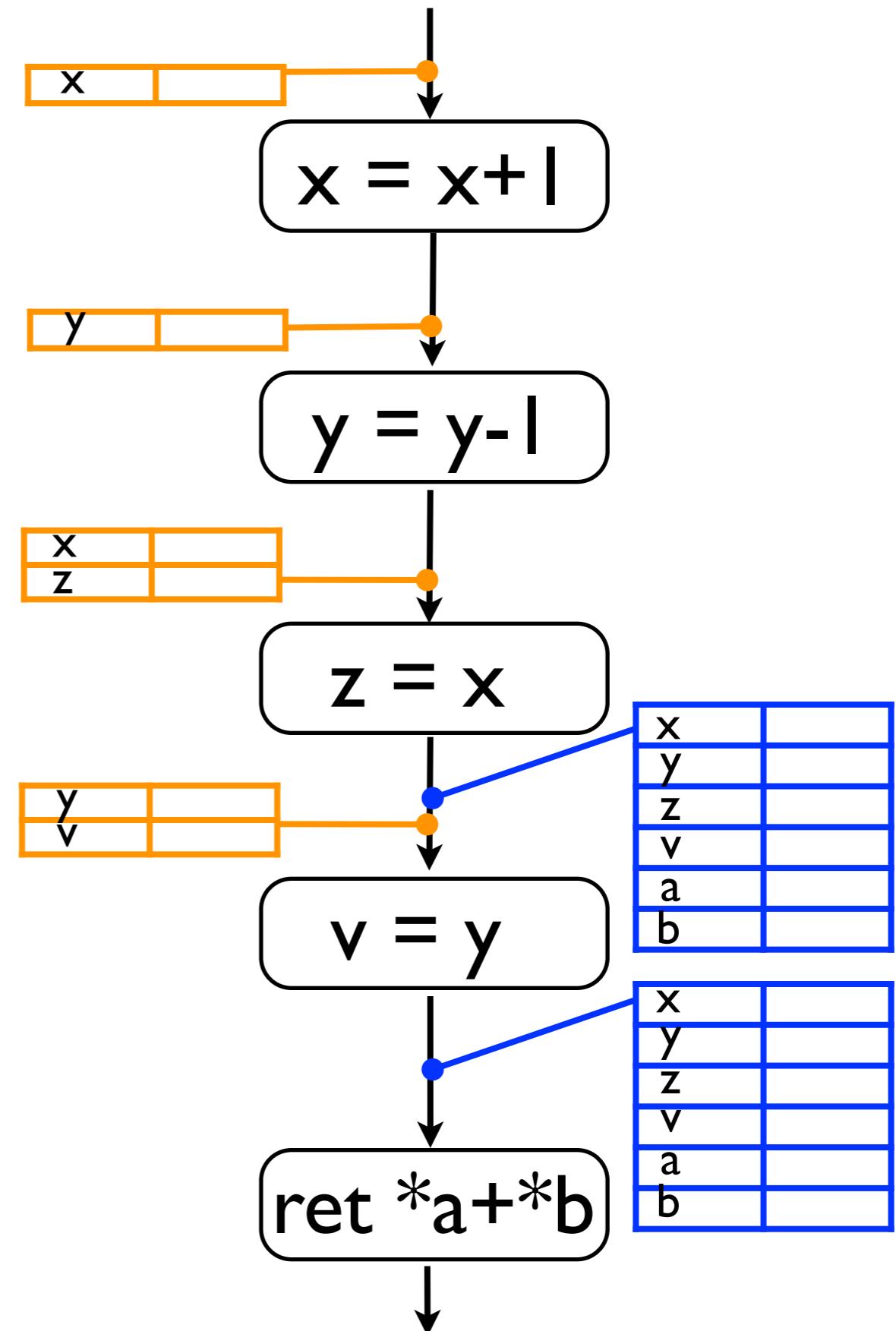
Key: General Sparse Analysis

“Right Part at Right Moment”



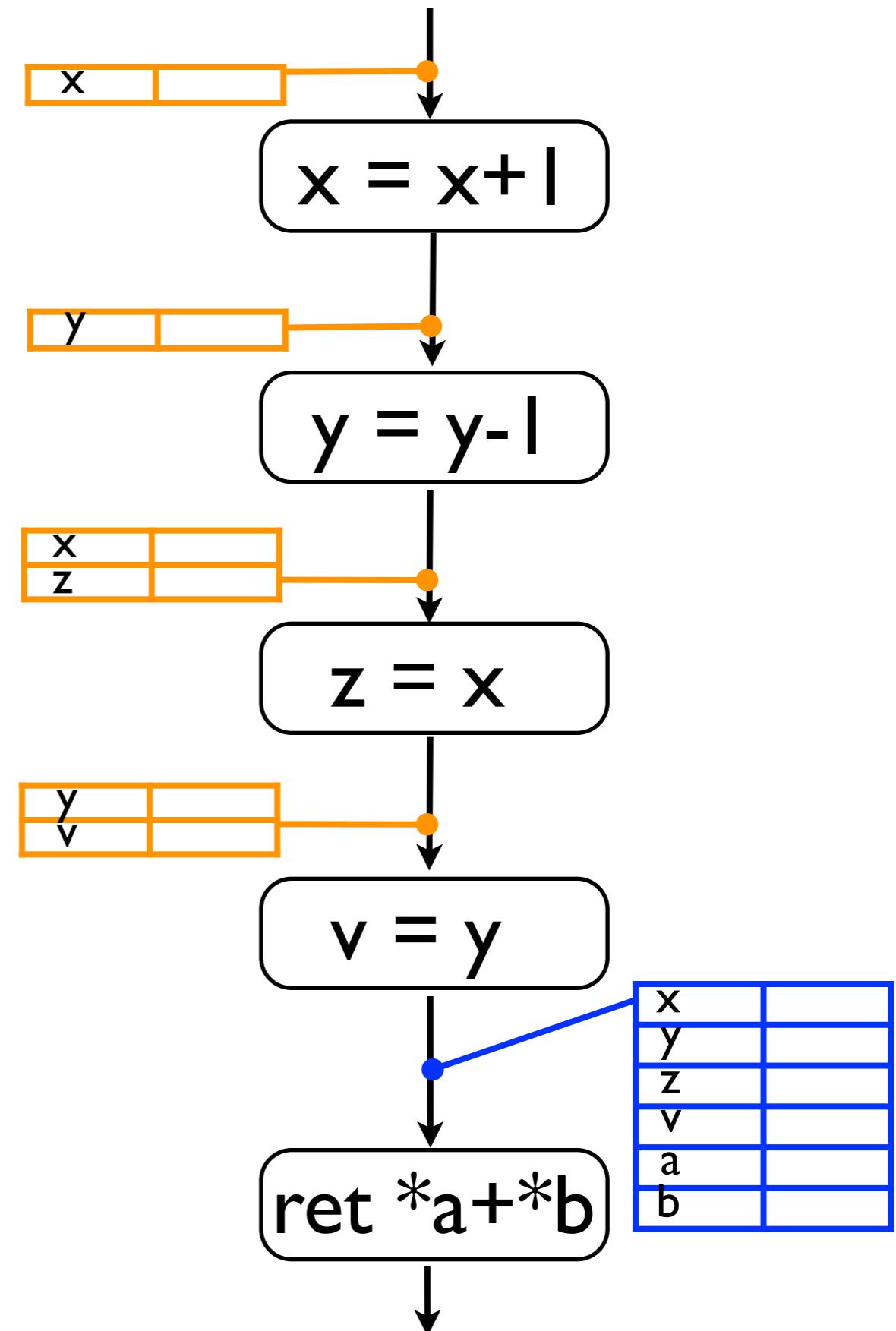
Key: General Sparse Analysis

“Right Part at Right Moment”



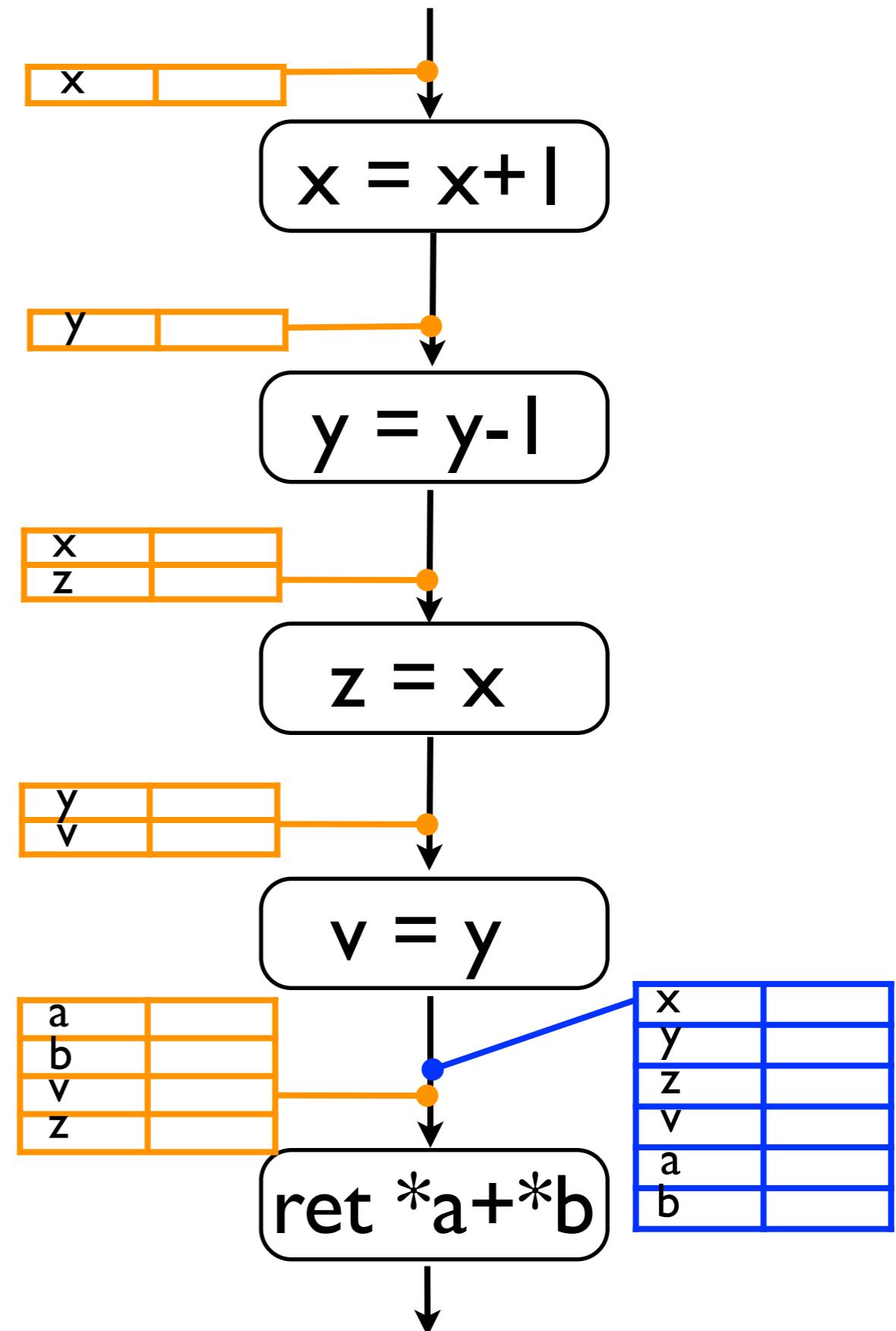
Key: General Sparse Analysis

“Right Part at Right Moment”



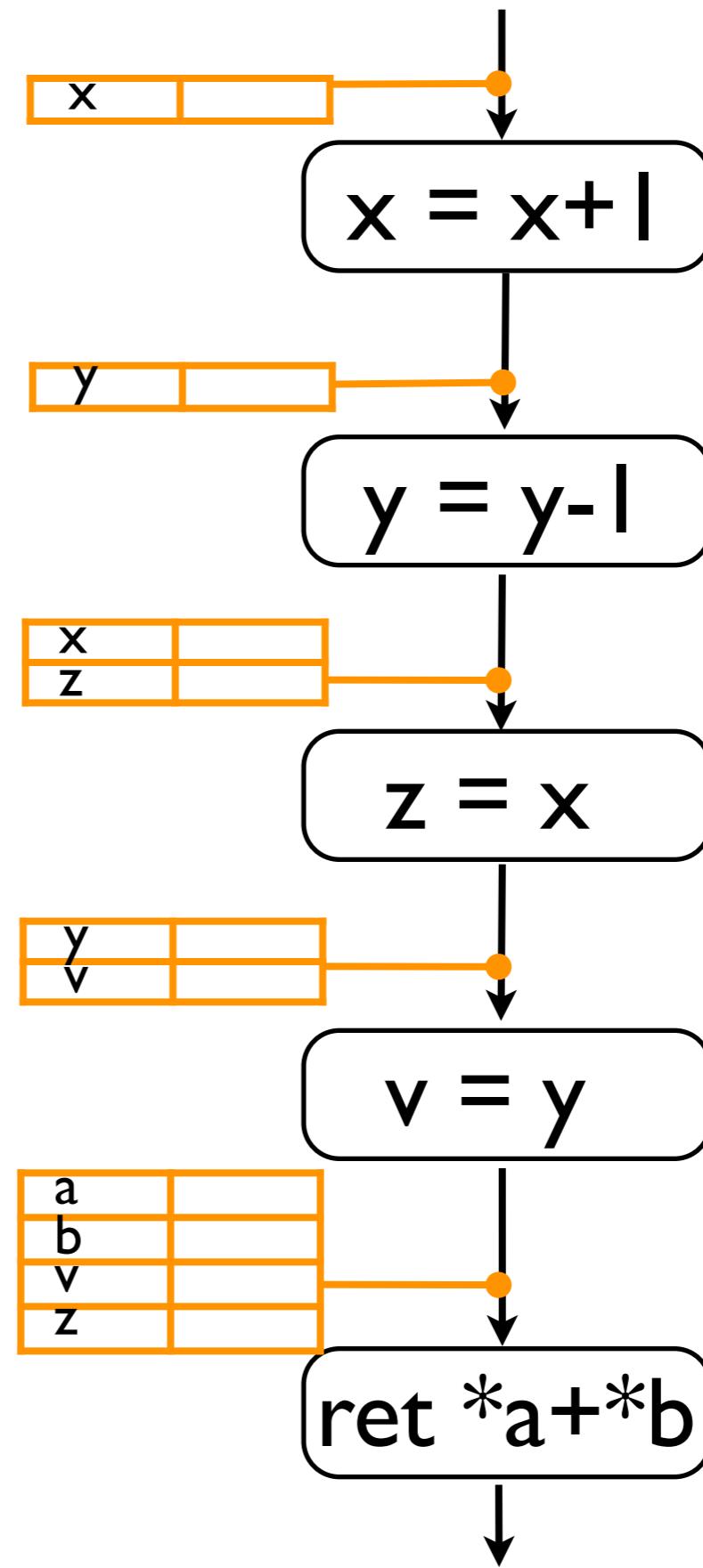
Key: General Sparse Analysis

“Right Part at Right Moment”



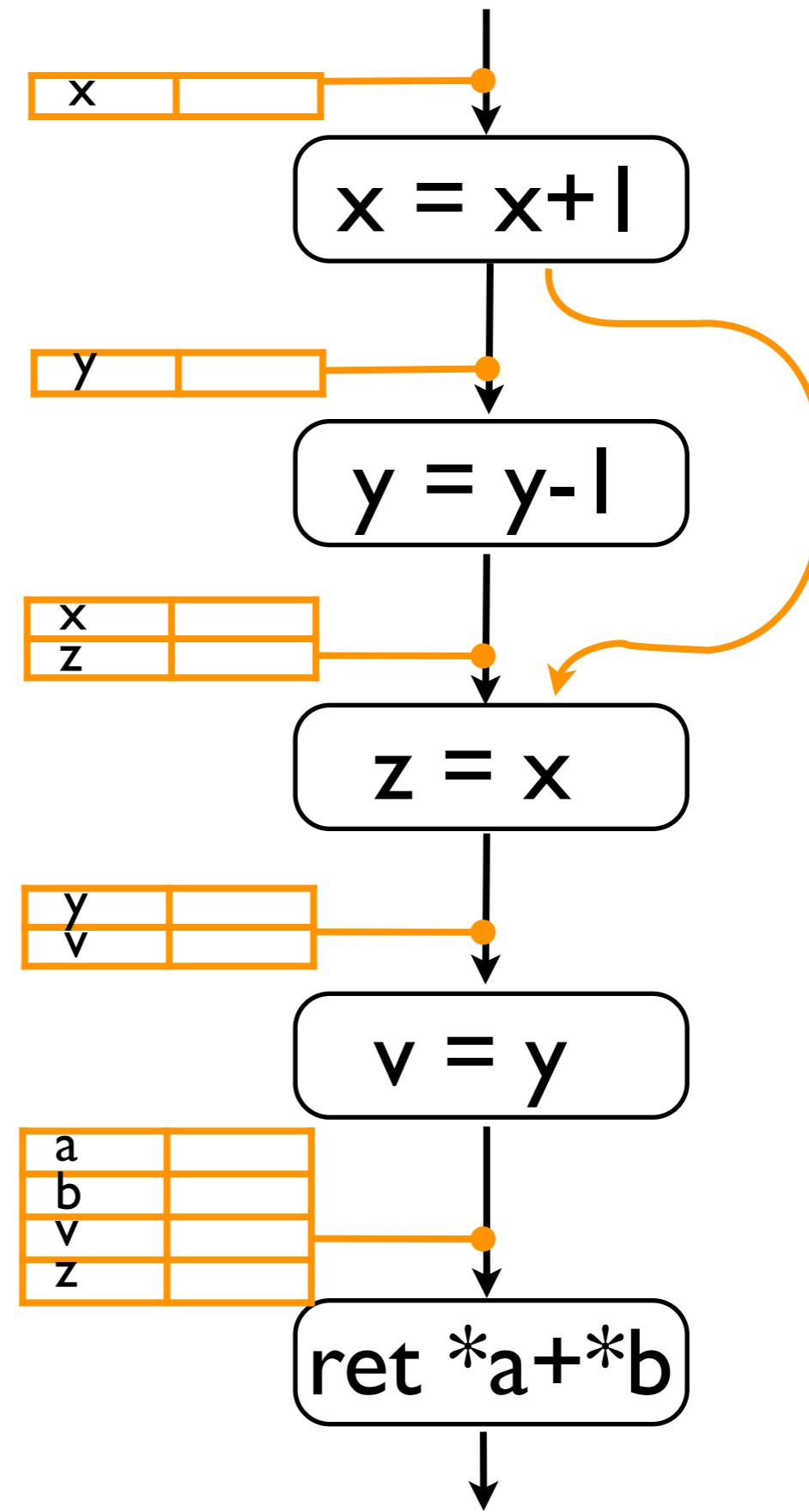
Key: General Sparse Analysis

“Right Part at Right Moment”



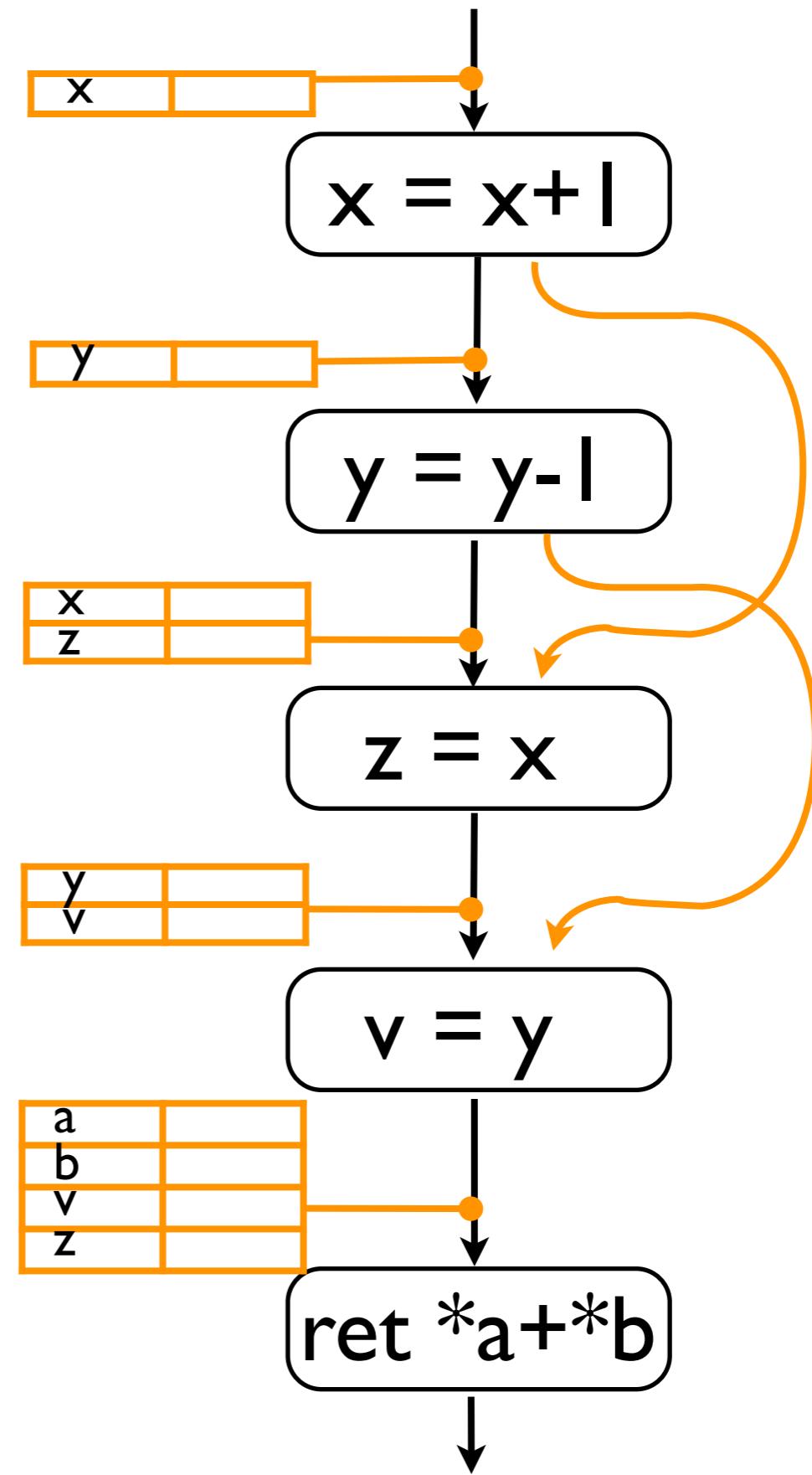
Key: General Sparse Analysis

“Right Part at Right Moment”



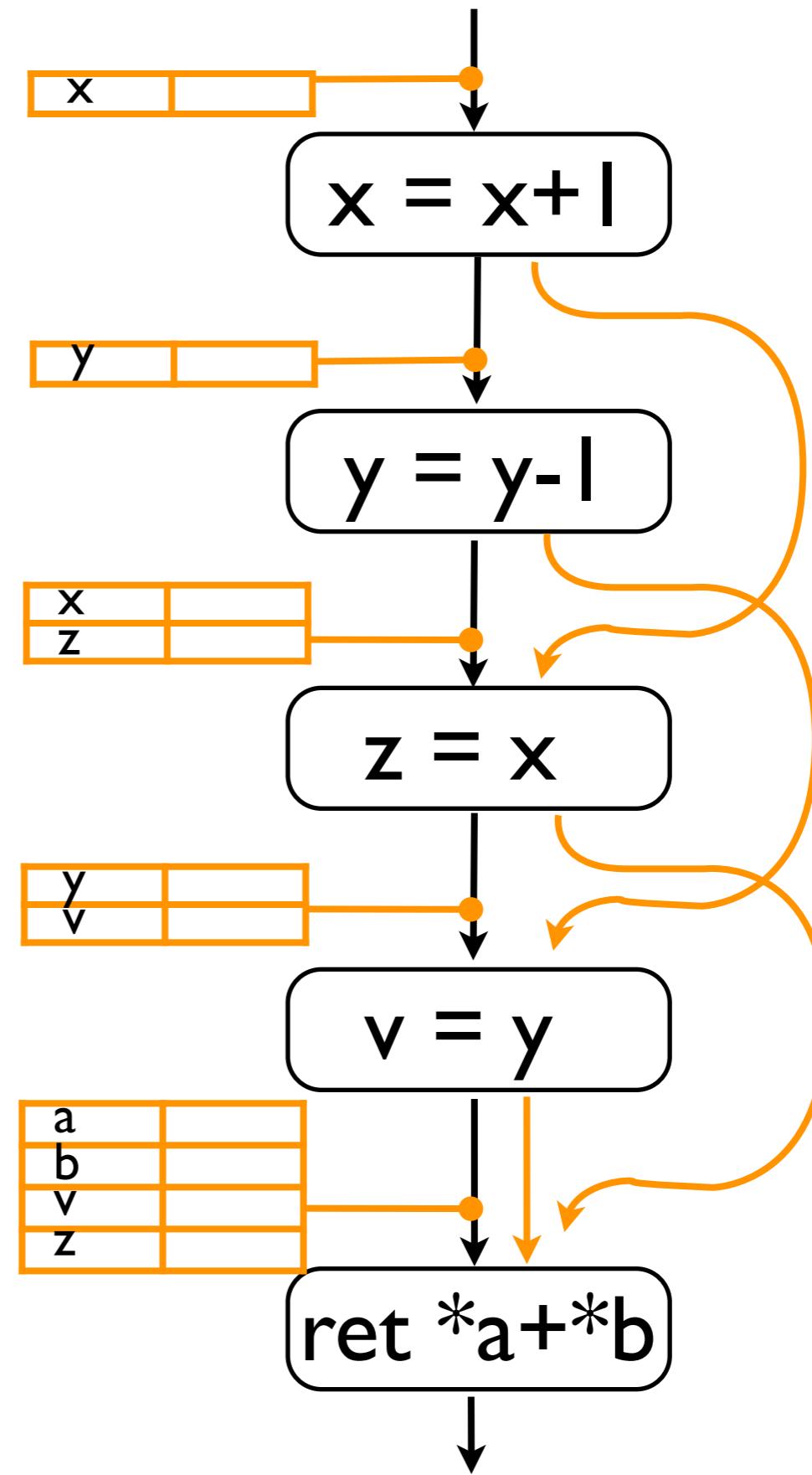
Key: General Sparse Analysis

“Right Part at Right Moment”



Key: General Sparse Analysis

“Right Part at Right Moment”



General Sparse Analysis Framework

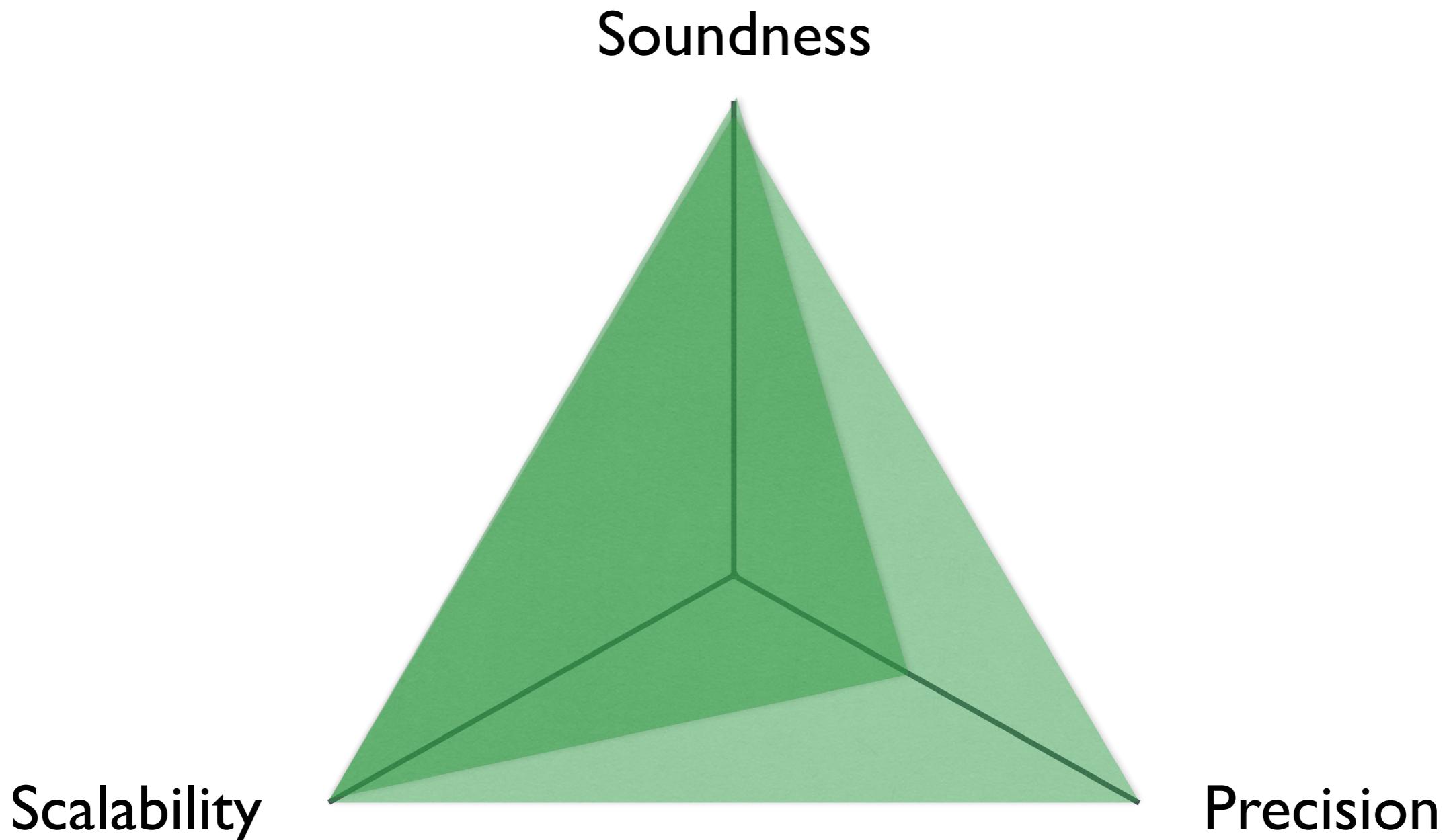
Theorem. (preservation of soundness and precision)

$$\hat{F} : \hat{D} \rightarrow \hat{D} \xrightarrow{\text{sparsify}} \hat{F}_s : \hat{D} \rightarrow \hat{D}$$

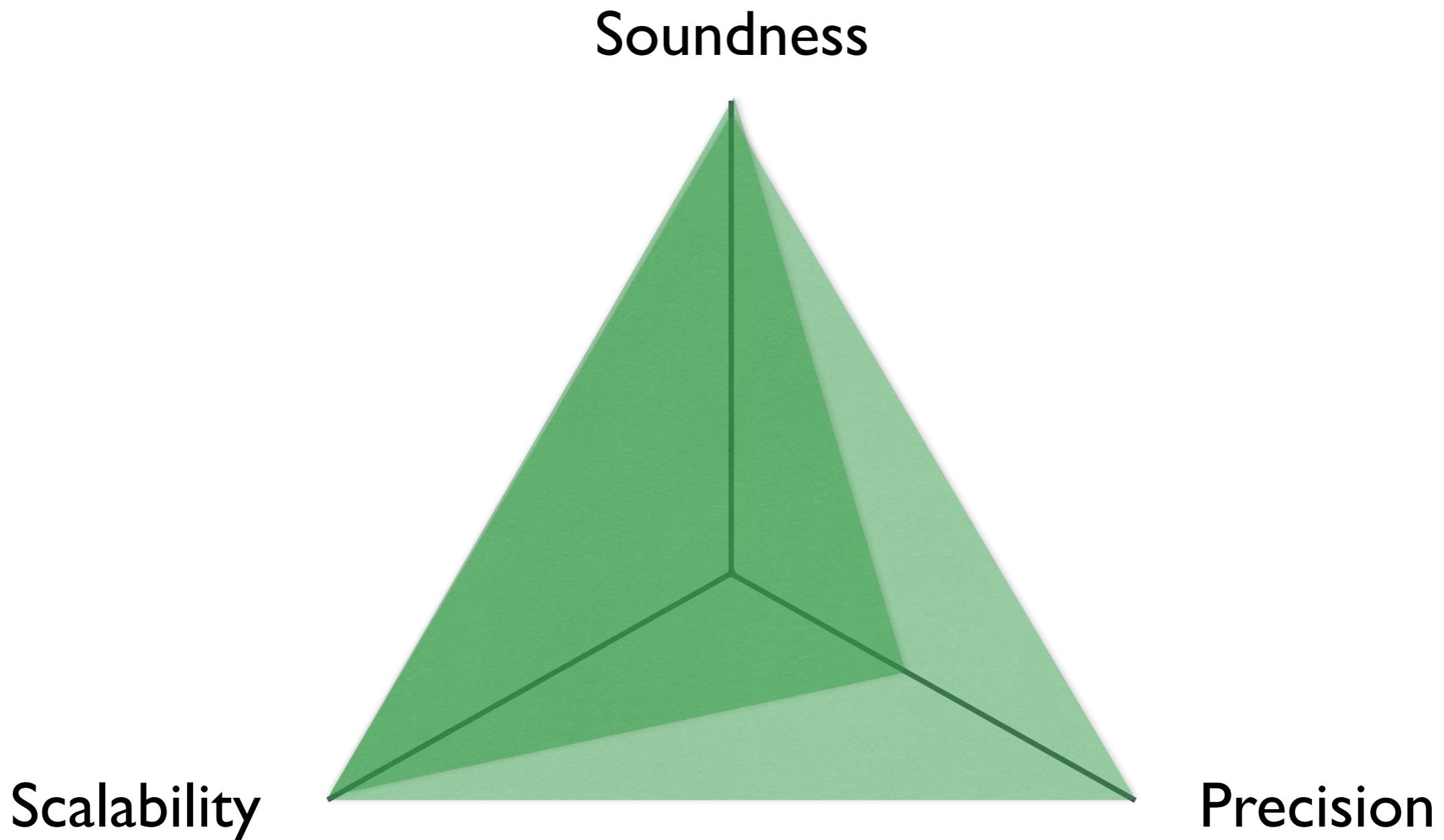
$$\text{fix } \hat{F} = \text{fix } \hat{F}_s$$

*“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

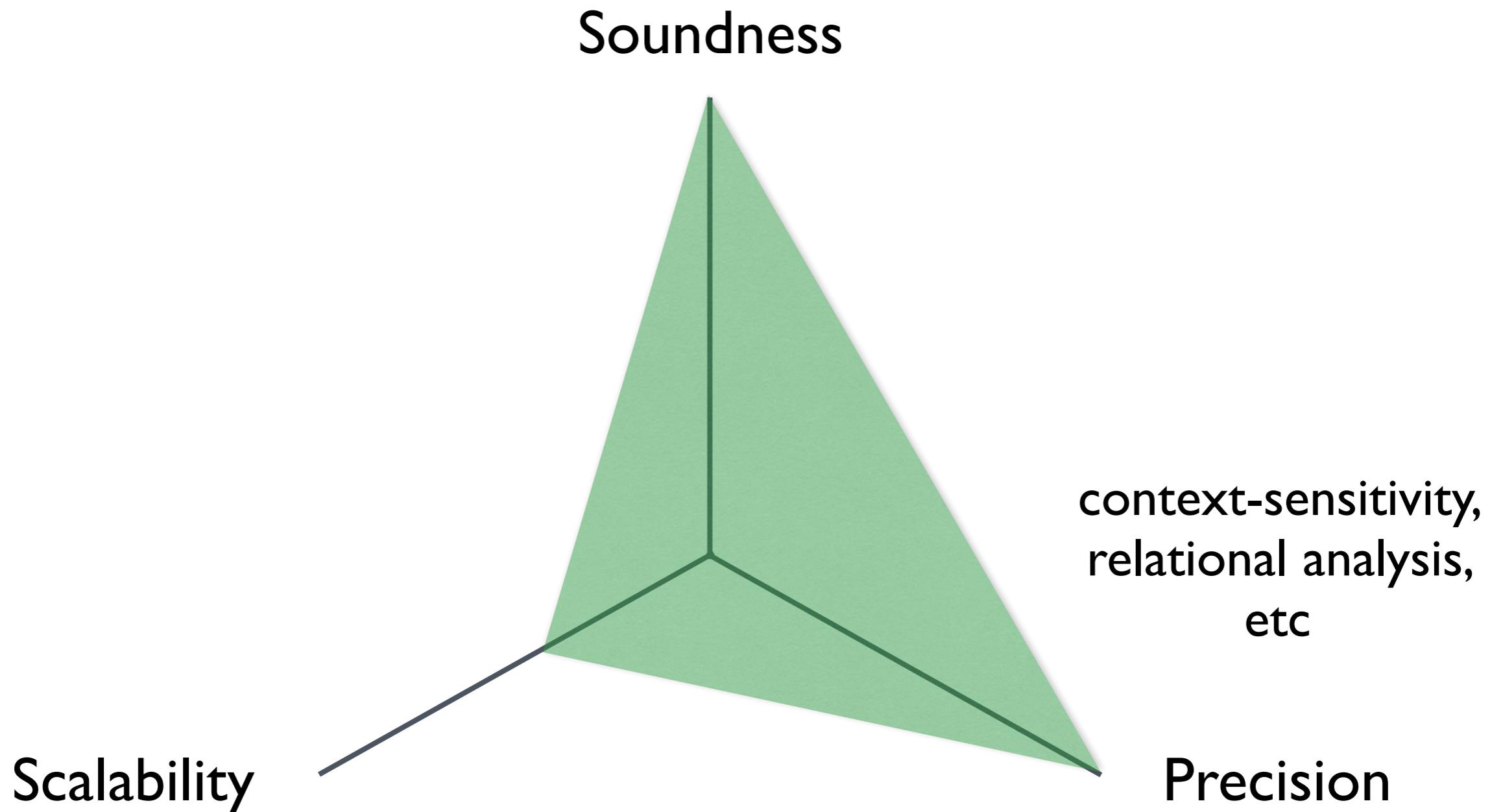


The Second Goal: Precision



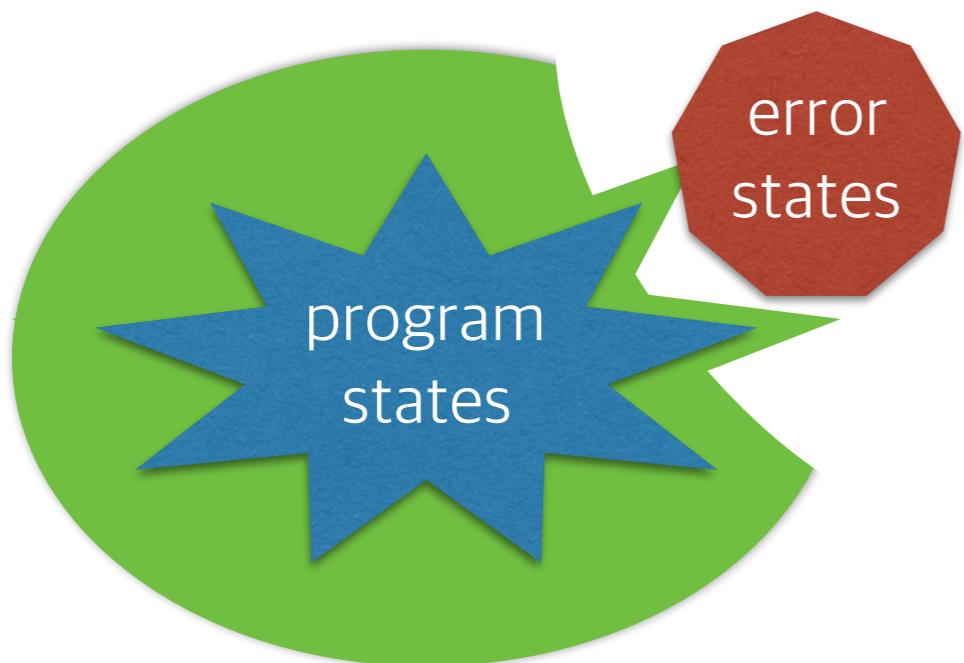
Challenge: Can we achieve it without scalability loss?

cf) Existing Techniques



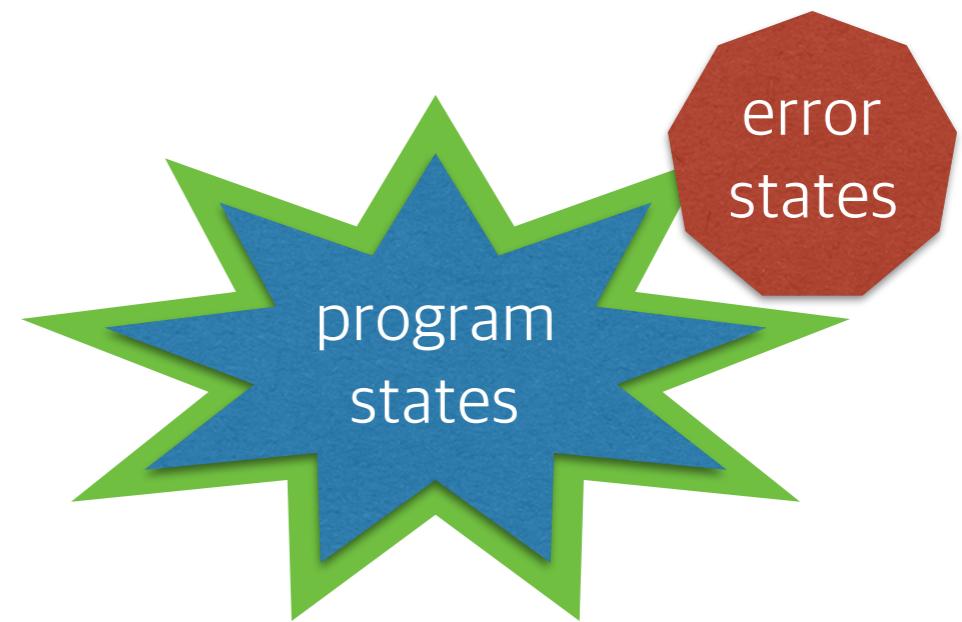
Selective X-Sensitivity Approach

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



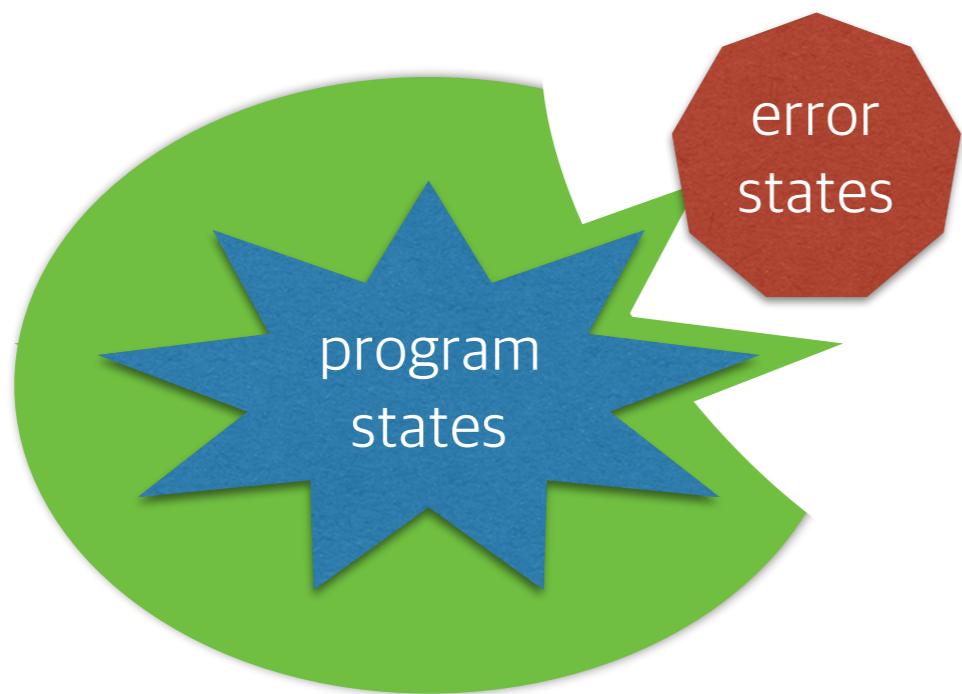
ours

vs.

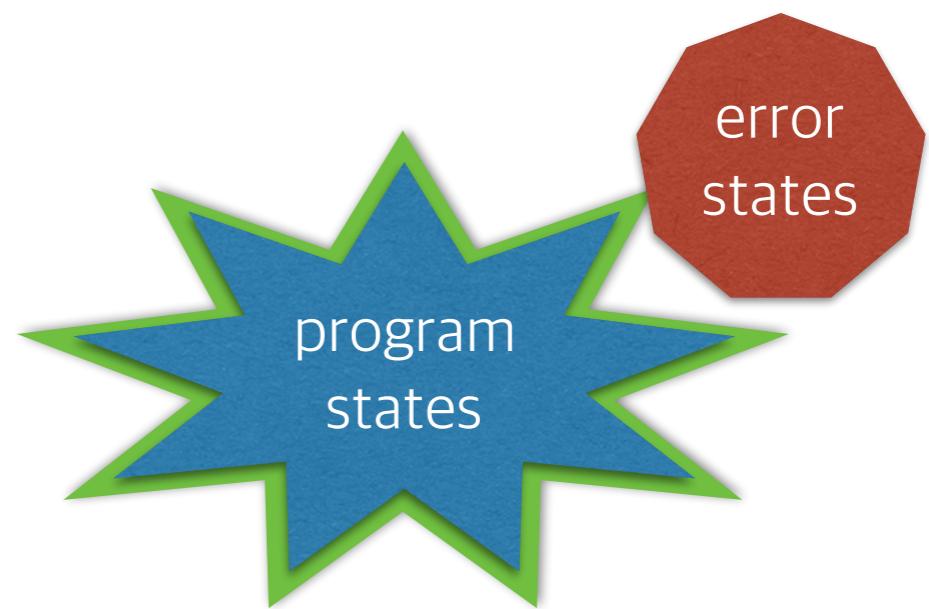


existing techniques

Effectiveness



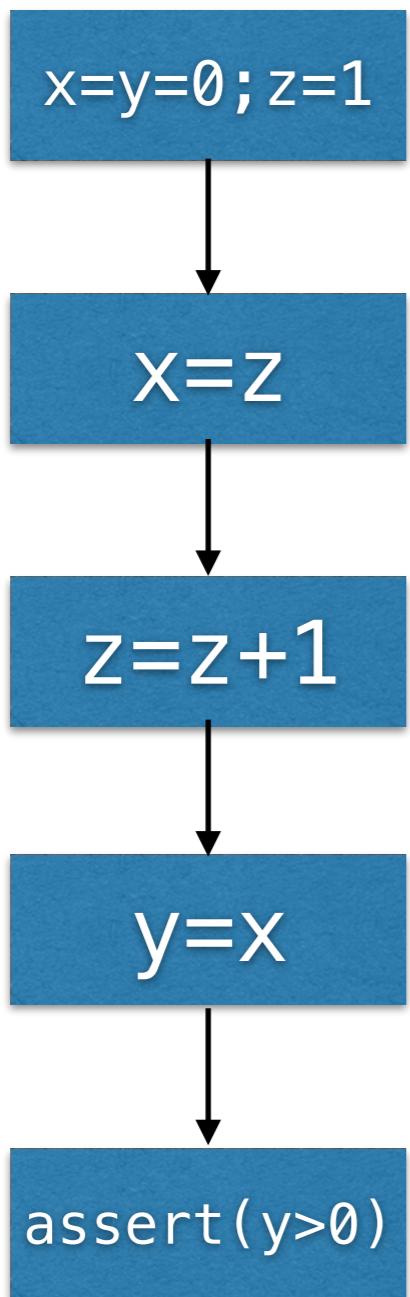
vs.



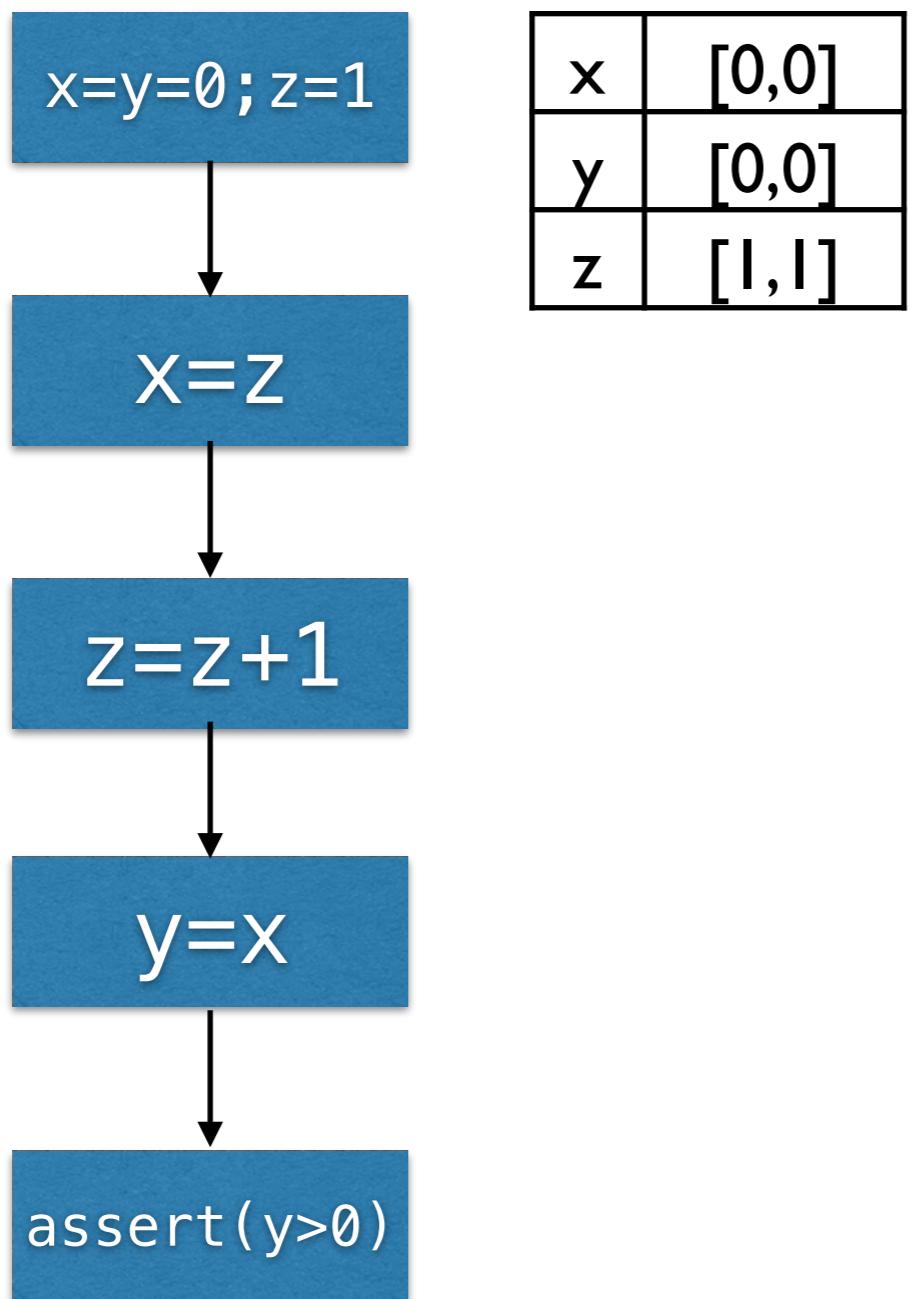
+25% / -25%

+25% / -1300%

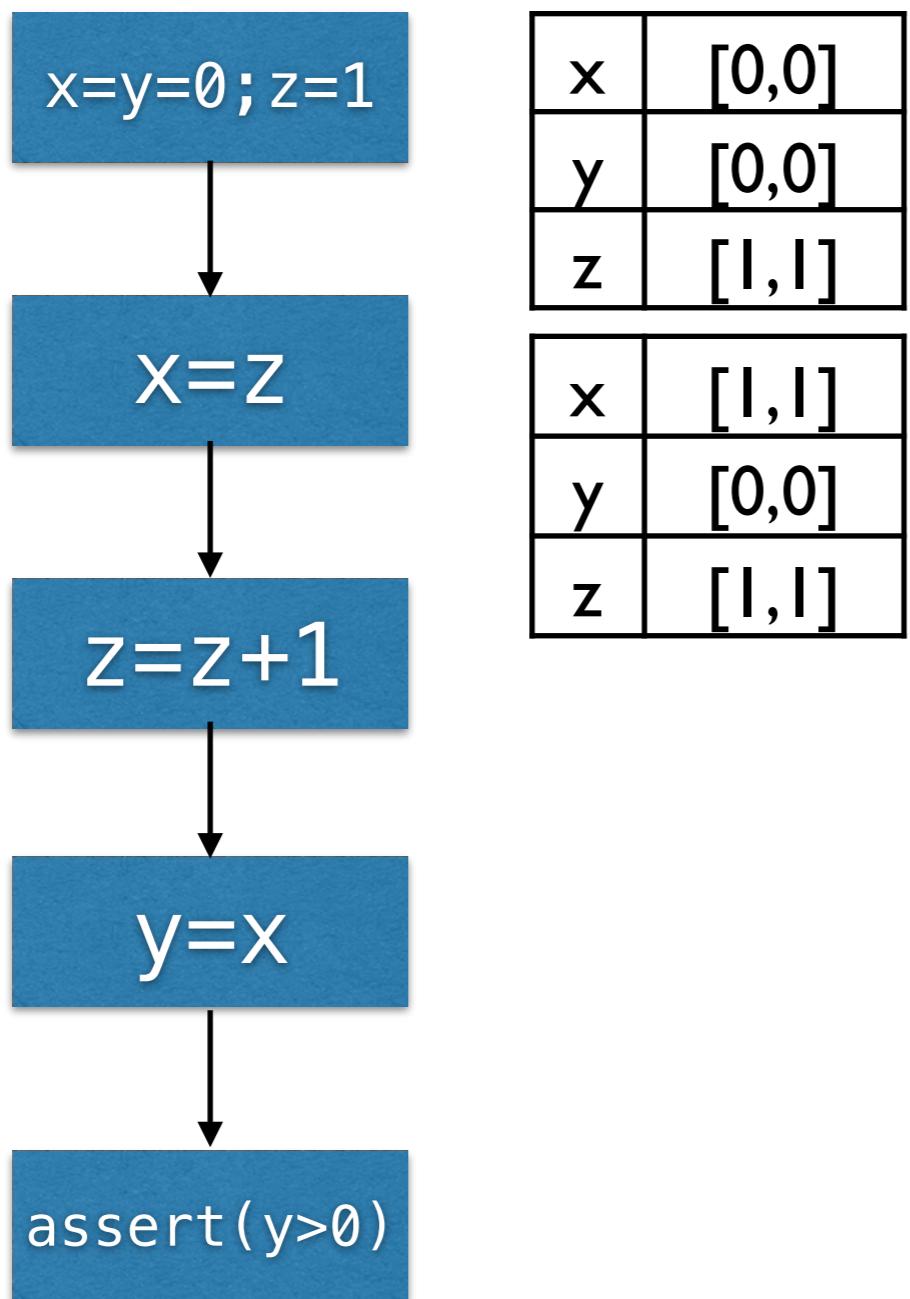
Flow-Sensitivity



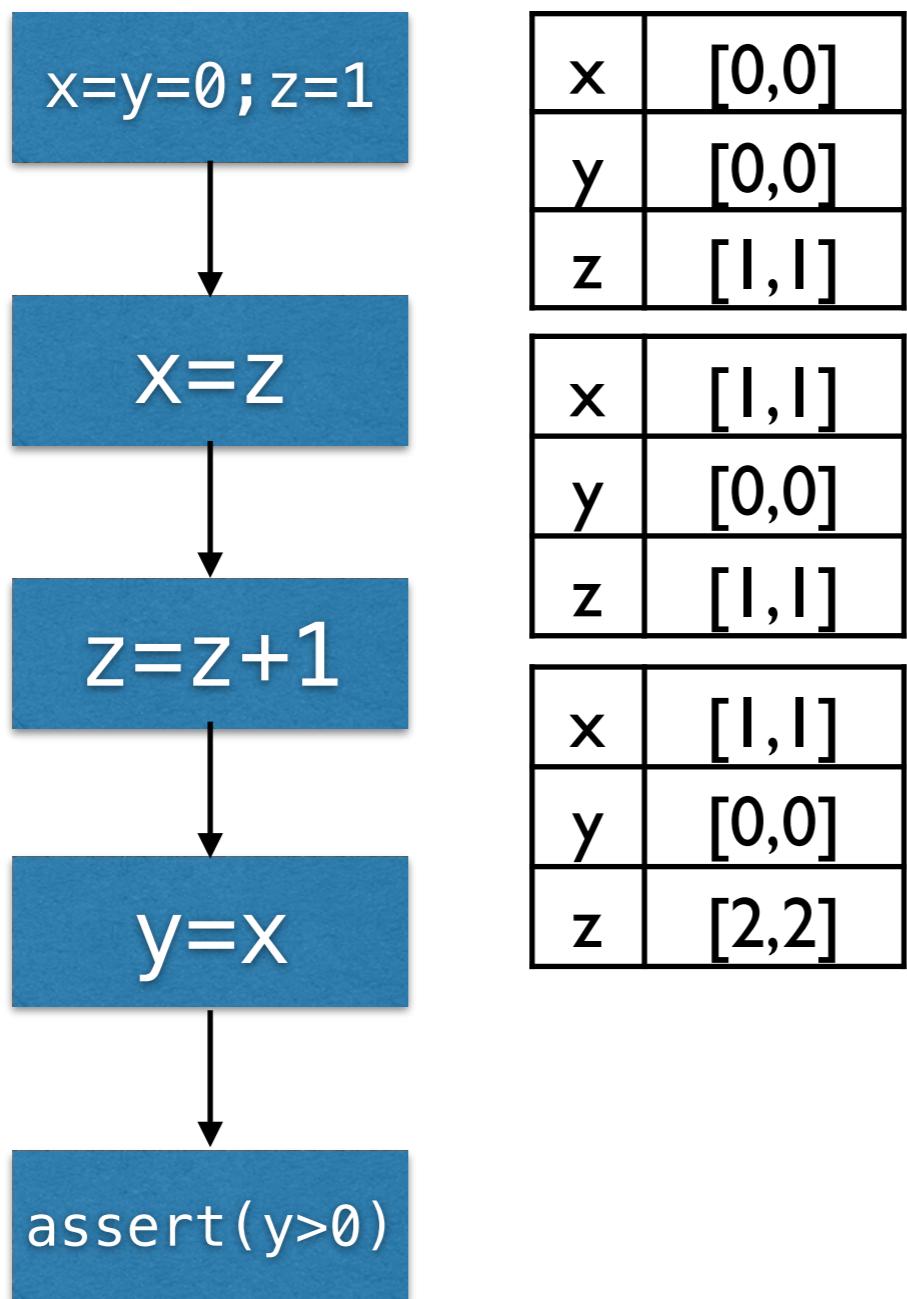
Flow-Sensitivity



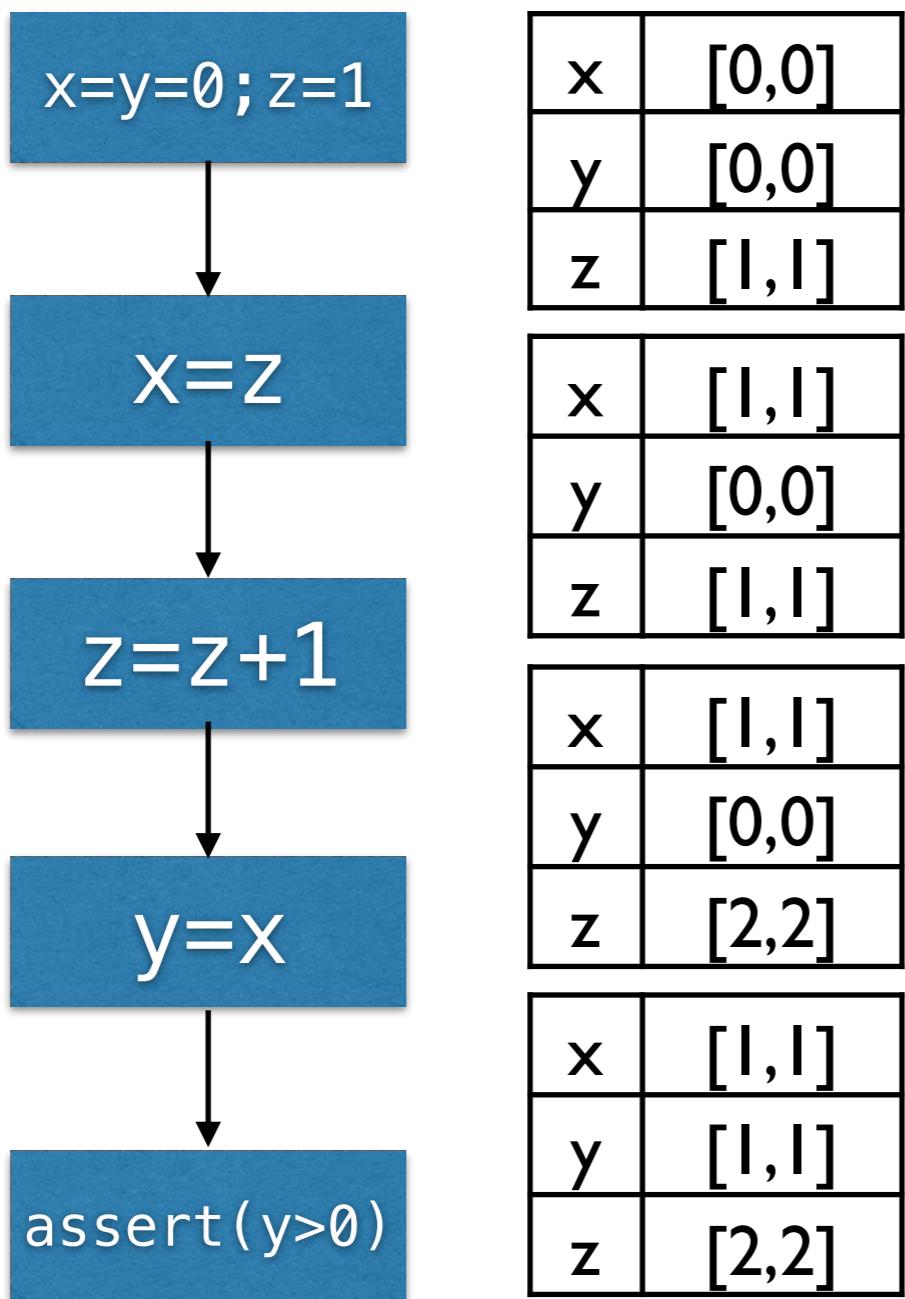
Flow-Sensitivity



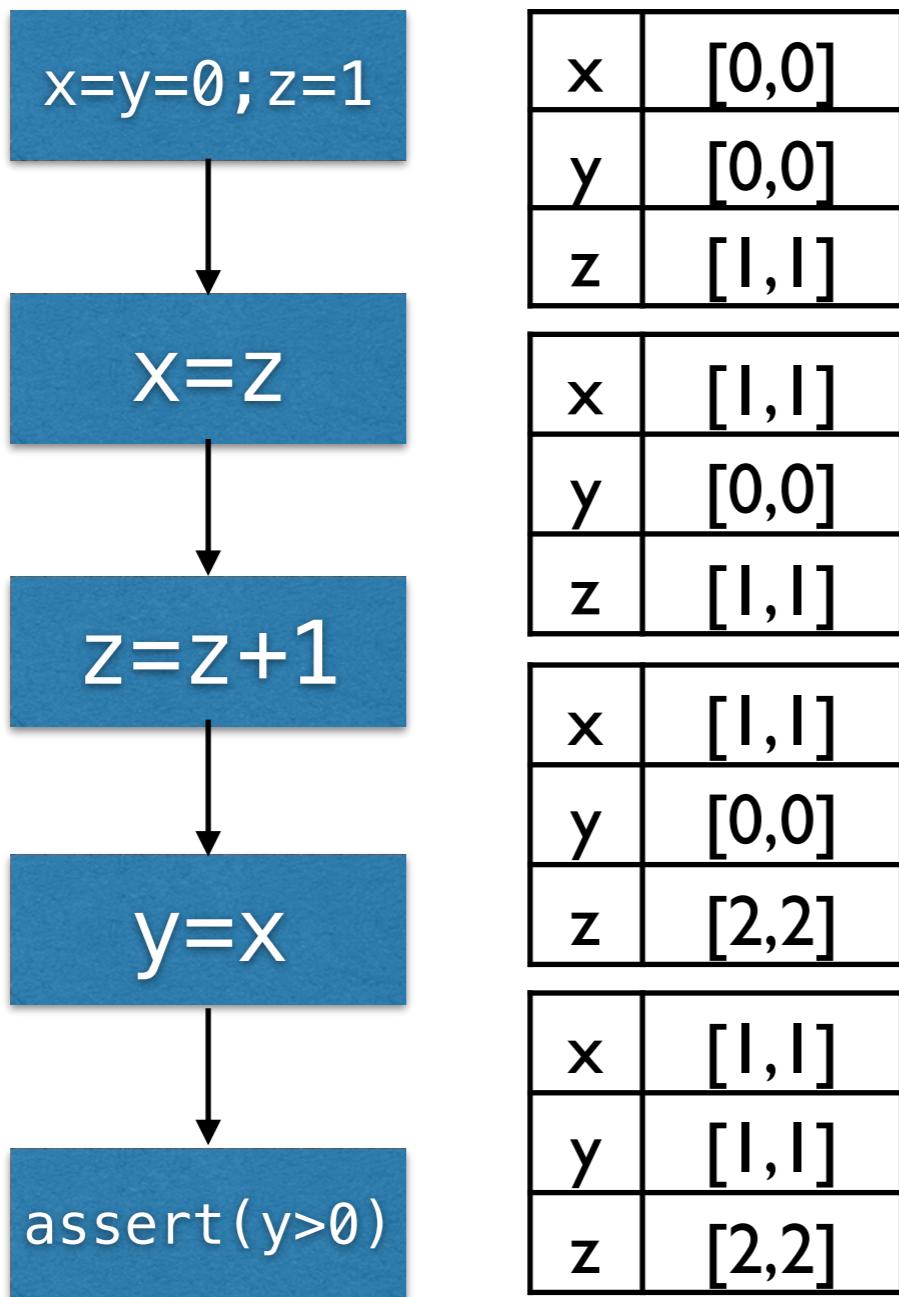
Flow-Sensitivity



Flow-Sensitivity

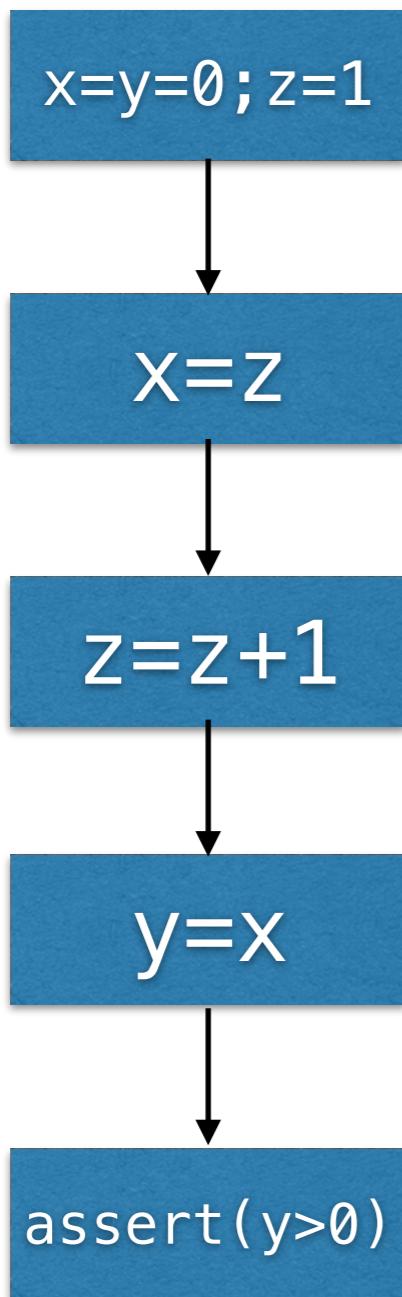


Flow-Sensitivity



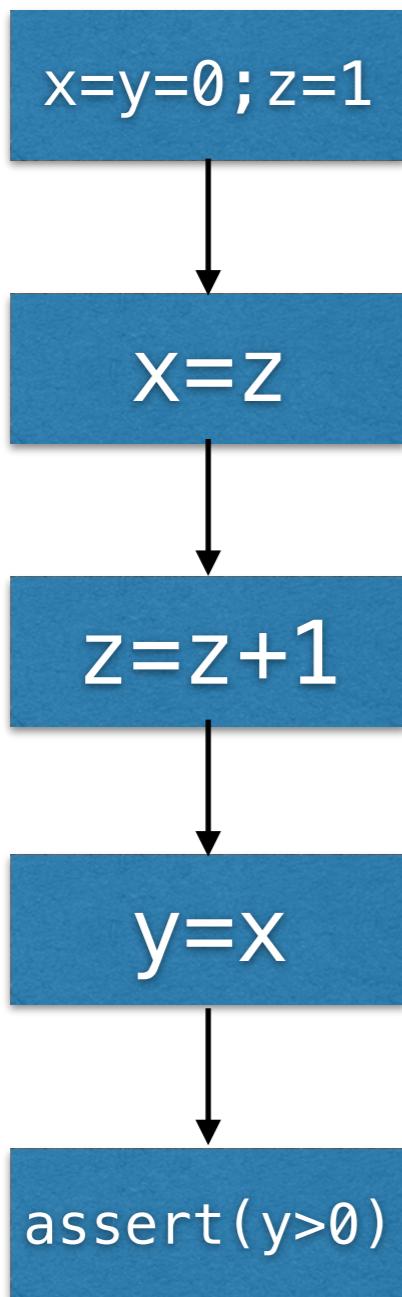
precise but costly

Flow-Insensitivity



x	[0, +∞]
y	[0, +∞]
z	[1, +∞]

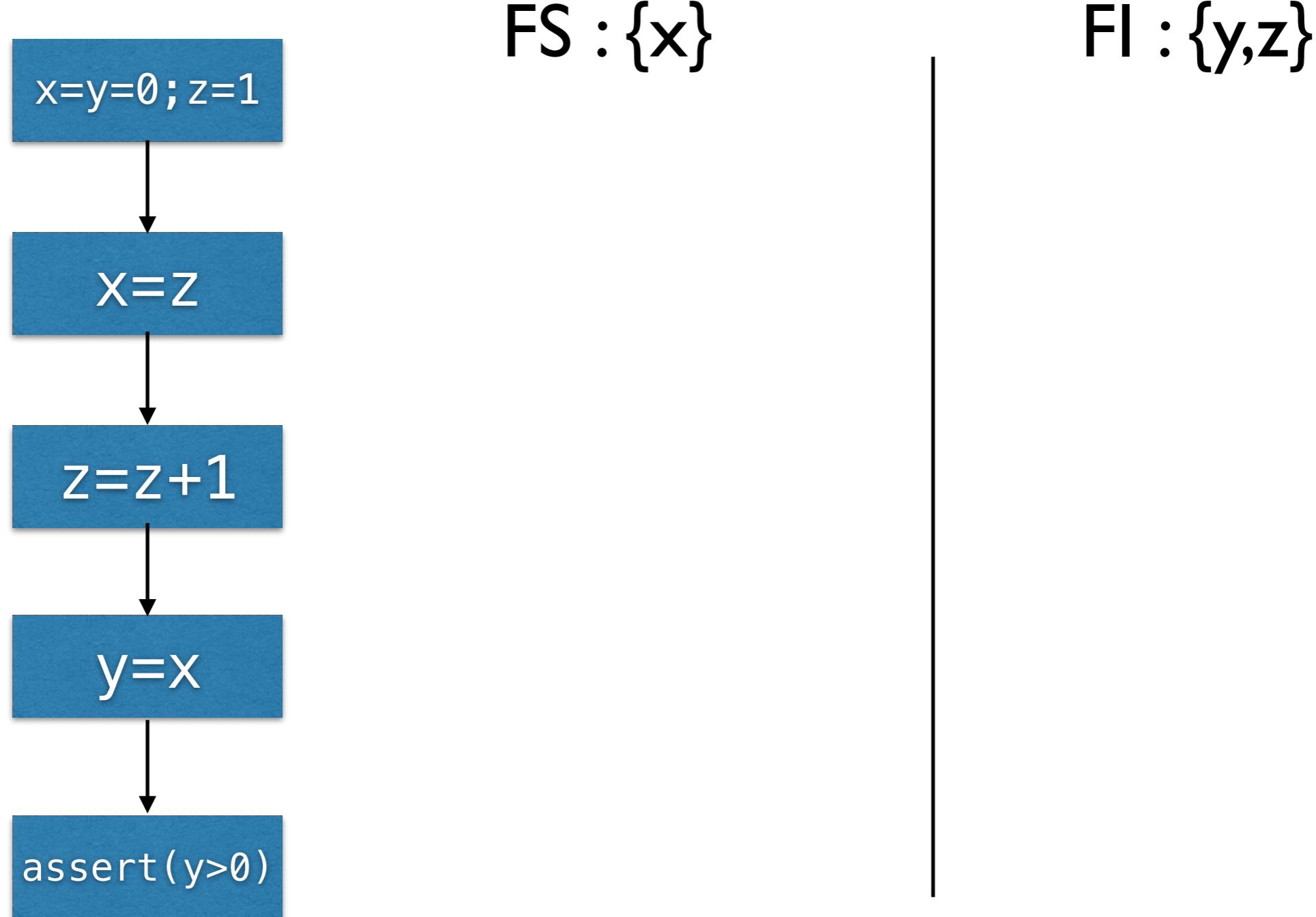
Flow-Insensitivity



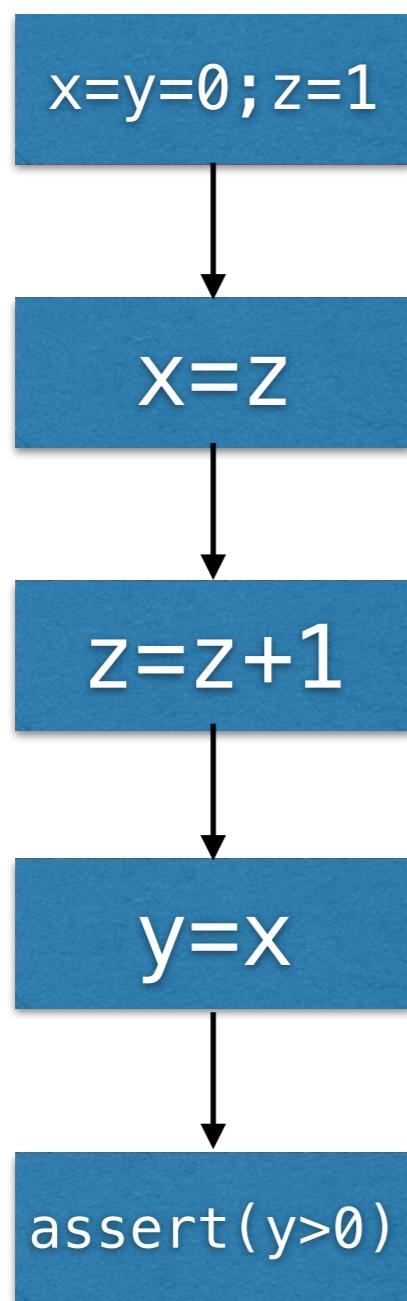
x	$[0, +\infty]$
y	$[0, +\infty]$
z	$[1, +\infty]$

cheap but imprecise

Selective Flow-Sensitivity



Selective Flow-Sensitivity



$\text{FS} : \{x\}$

x	[0,0]
---	-------

x	[l, +∞]
---	---------

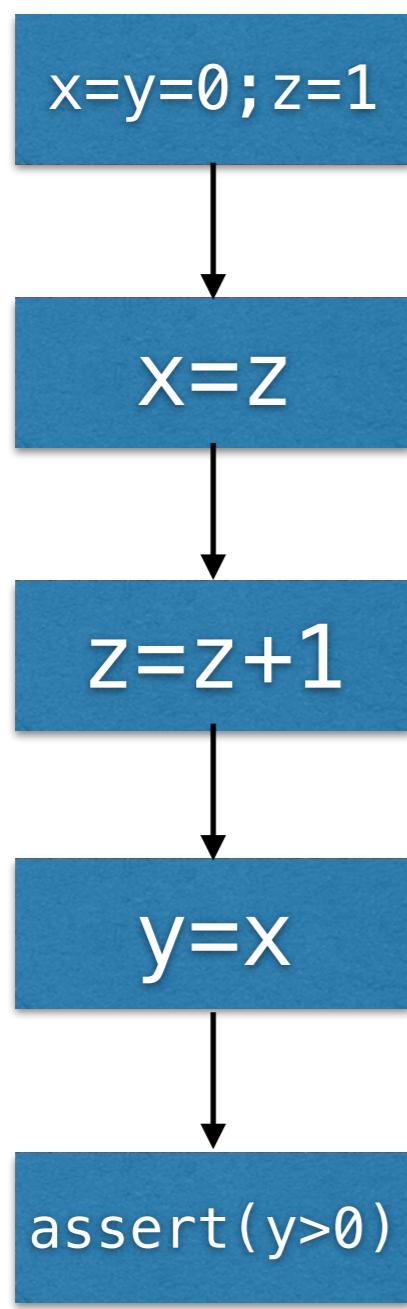
x	[l, +∞]
---	---------

x	[l, +∞]
---	---------

$\text{FI} : \{y, z\}$

y	[0, +∞]
z	[l, +∞]

Selective Flow-Sensitivity



$\text{FS} : \{x\}$

x	[0,0]
---	-------

x	[l, +∞]
---	---------

x	[l, +∞]
---	---------

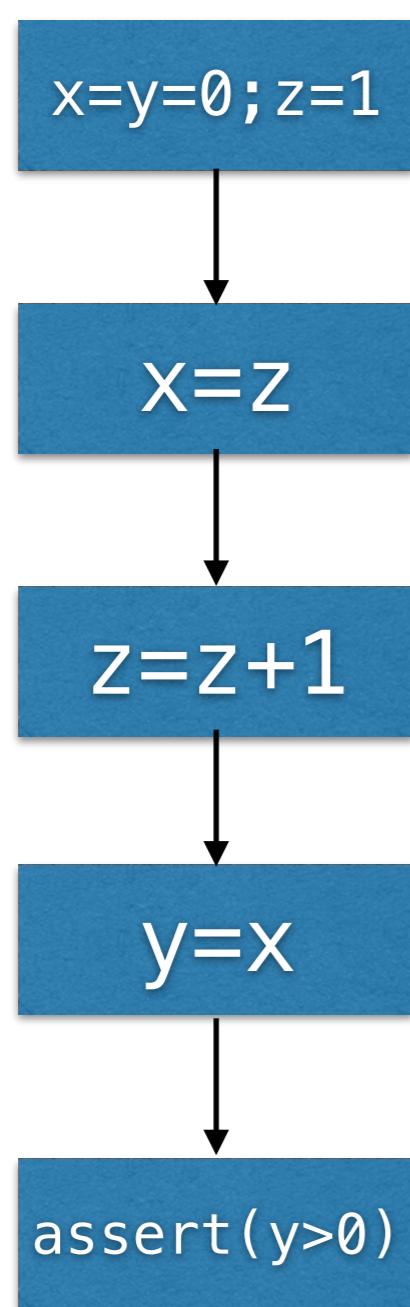
x	[l, +∞]
---	---------

fail to prove

$\text{FI} : \{y, z\}$

y	[0, +∞]
z	[l, +∞]

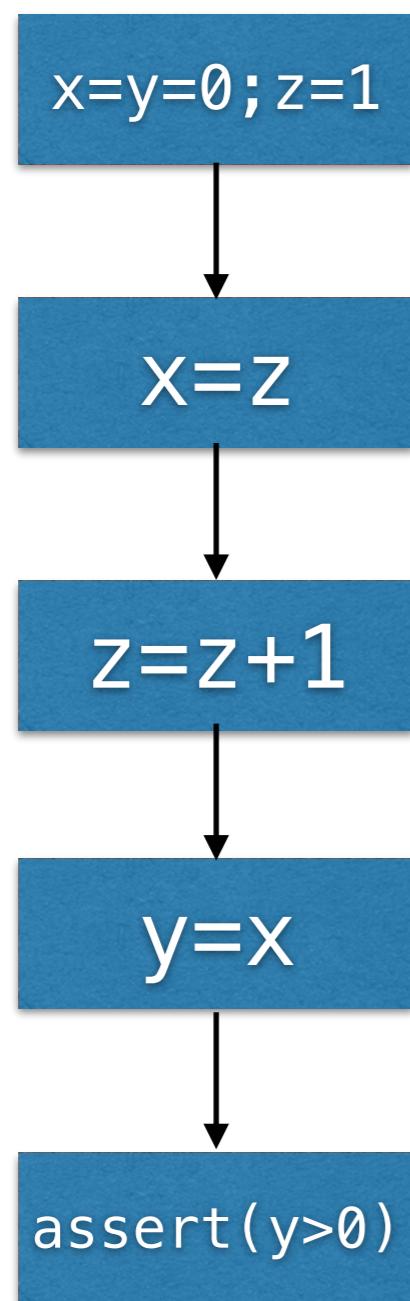
Selective Flow-Sensitivity



$\text{FS} : \{y\}$

$\text{FI} : \{x,z\}$

Selective Flow-Sensitivity



$\text{FS} : \{y\}$

y	[0,0]
---	-------

y	[0,0]
---	-------

y	[0,0]
---	-------

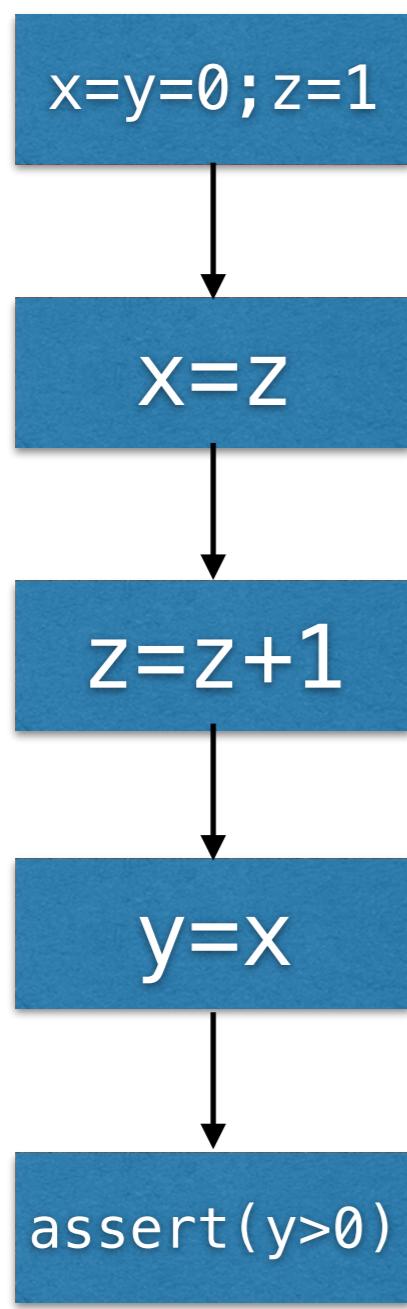
y	[0, +∞]
---	---------

$\text{FI} : \{x,z\}$

x	[0, +∞]
z	[1, +∞]

fail to prove

Selective Flow-Sensitivity



$\text{FS} : \{z\}$

z	$[l, l]$
-----	----------

z	$[l, l]$
-----	----------

z	$[2, 2]$
-----	----------

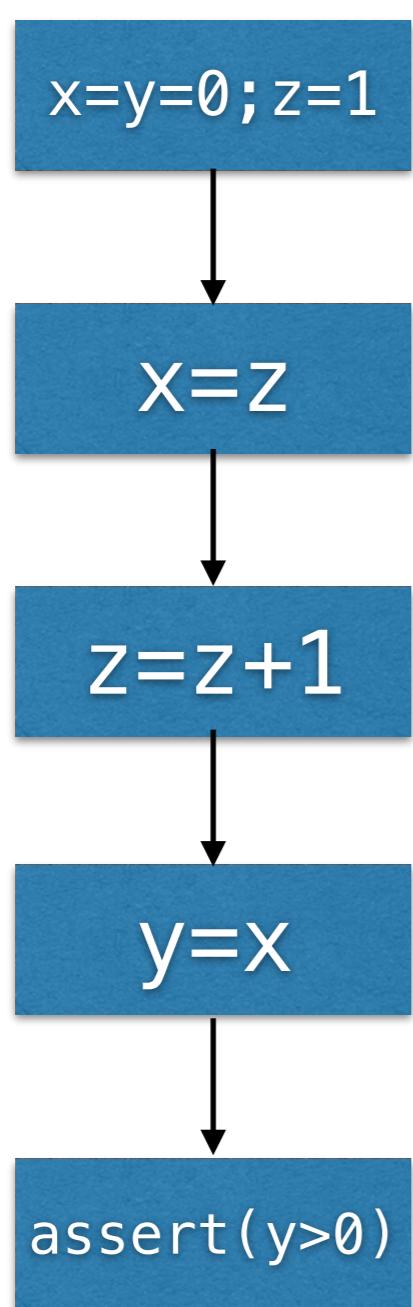
z	$[2, 2]$
-----	----------

fail to prove

$\text{FI} : \{x, y\}$

x	$[0, +\infty]$
y	$[0, +\infty]$

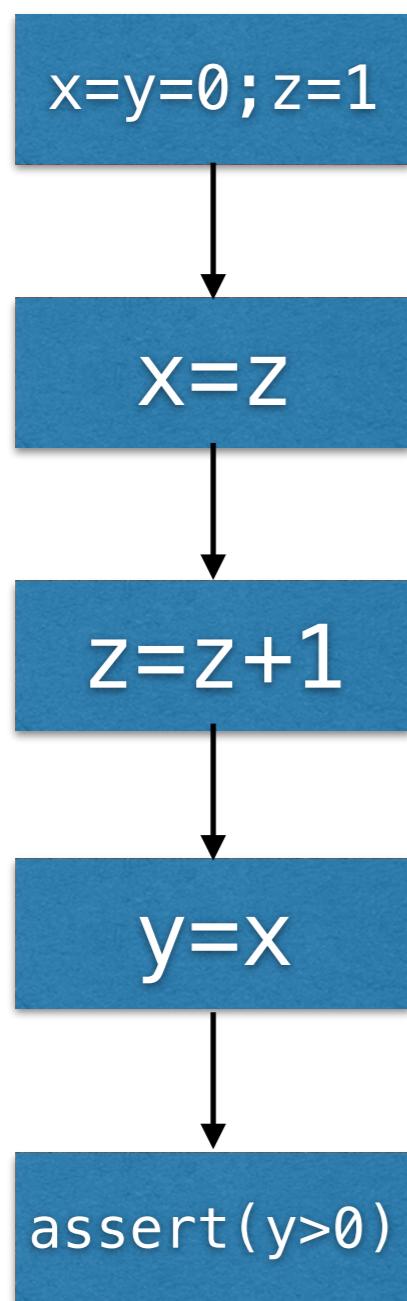
Selective Flow-Sensitivity



FS : {y,z}

FI : {x}

Selective Flow-Sensitivity



FS : {y,z}

y	[0,0]
z	[l,l]

y	[0,0]
z	[l,l]

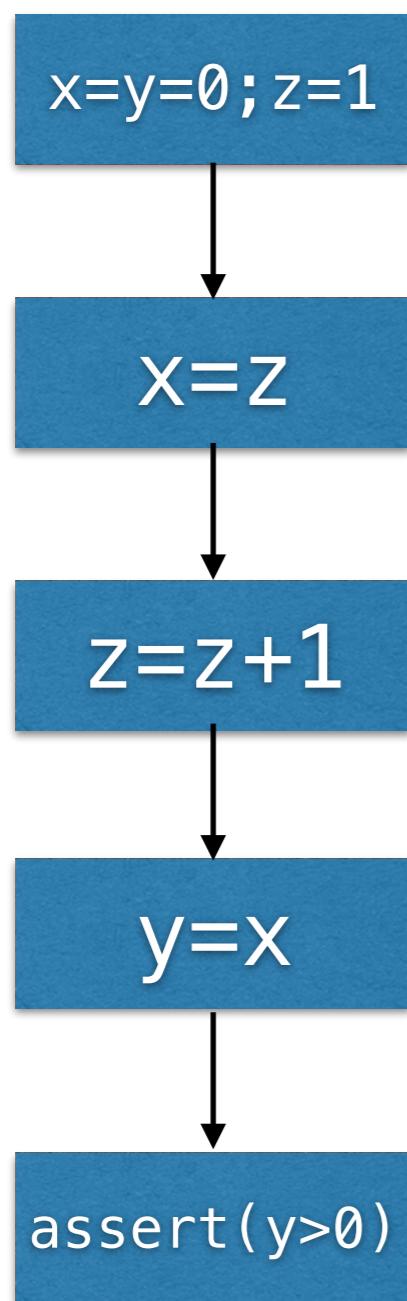
y	[0,0]
z	[2,2]

y	[0, +∞]
z	[2,2]

FI : {x}

x	[0, +∞]
---	---------

Selective Flow-Sensitivity



FS : {y,z}

y	[0,0]
z	[l,l]

y	[0,0]
z	[l,l]

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

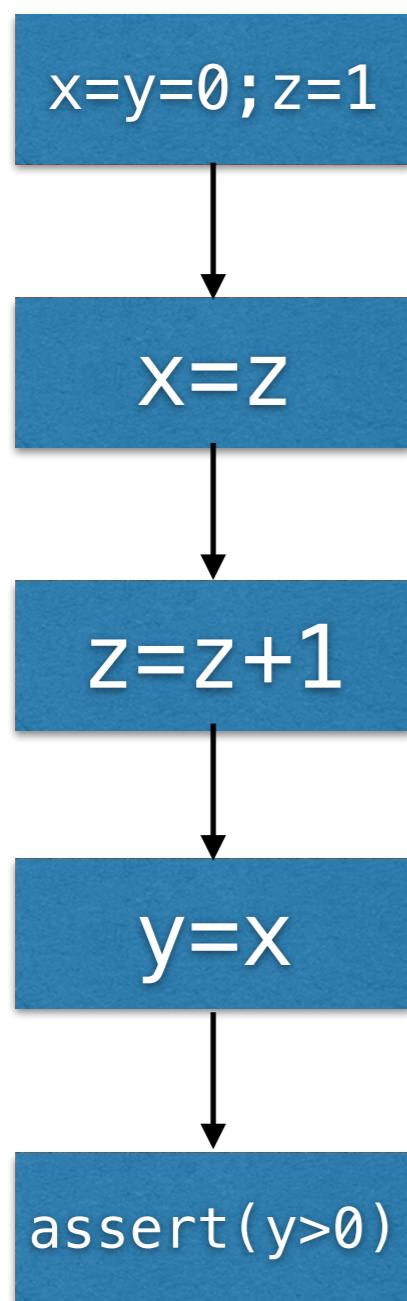
y	[0, +∞]
z	[2,2]

FI : {x}

x	[0, +∞]
---	---------

fail to prove

Selective Flow-Sensitivity



FS : { x, y }

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

x	[l, +∞]
y	[0,0]

x	[l, +∞]
y	[0,0]

x	[l, +∞]
y	[l, +∞]

FI : { z }

z	[l, +∞]
---	---------

Succeed

Hard Search Problem

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

Our Solutions

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



Learning-based Approach

Learning-based Approach

- Parameterized adaptation strategy

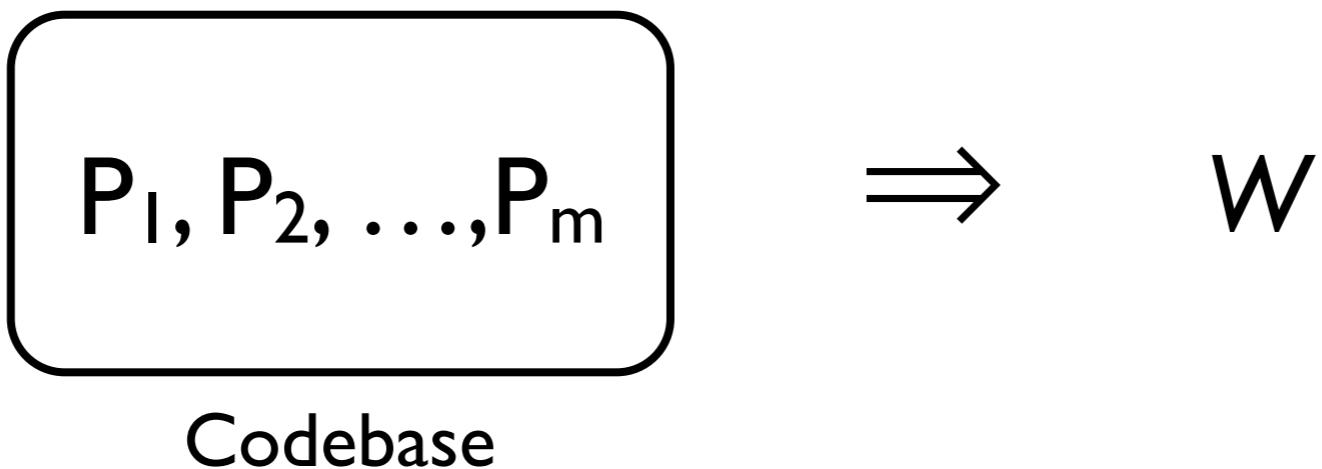
$$S_w : \text{pgm} \rightarrow 2^{\text{Var}}$$

Learning-based Approach

- Parameterized adaptation strategy

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

- Learn a good parameter W from existing codebase

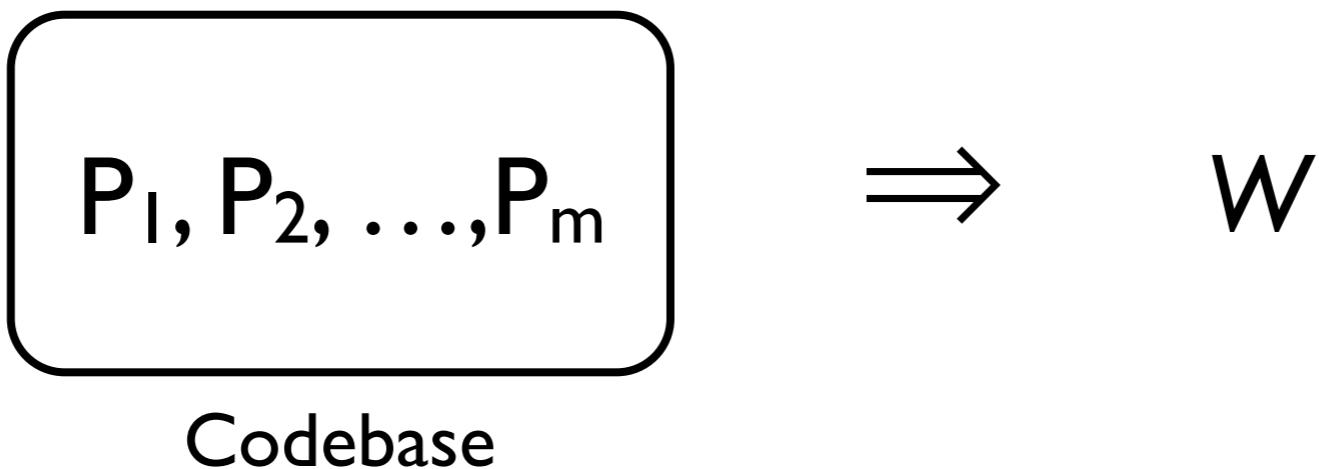


Learning-based Approach

- 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 $S_w(P)$

I. 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.

(I) Features

- Predicates over variables:

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

- 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.,

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

- Compute scores of variables:

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

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

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

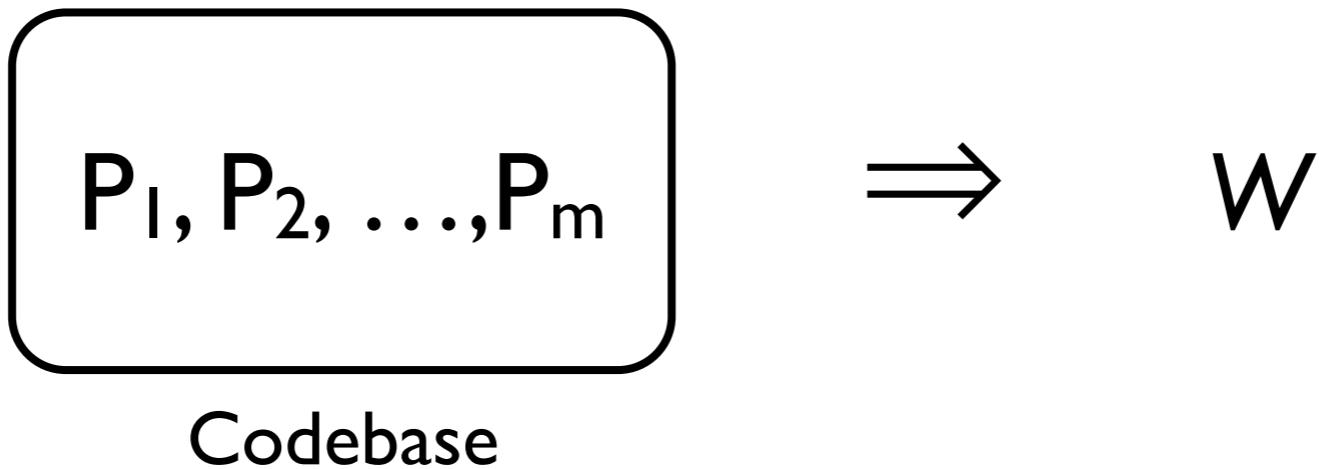
(3) Choose Top-k Variables

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

$$\begin{array}{l} \text{score}(x) = 0.3 \\ \text{score}(y) = 0.6 \\ \text{score}(z) = 0.1 \end{array} \quad \rightarrow \quad \{x, y\}$$

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

2. Learn a Good Parameter



- Solve the optimization problem:

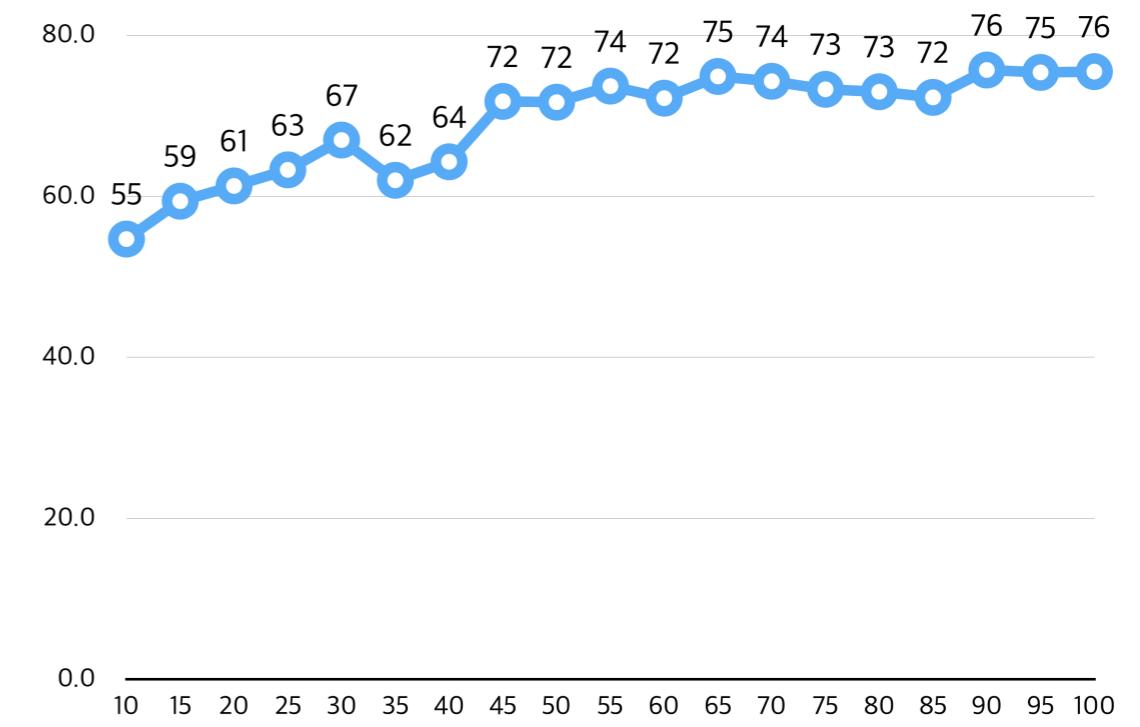
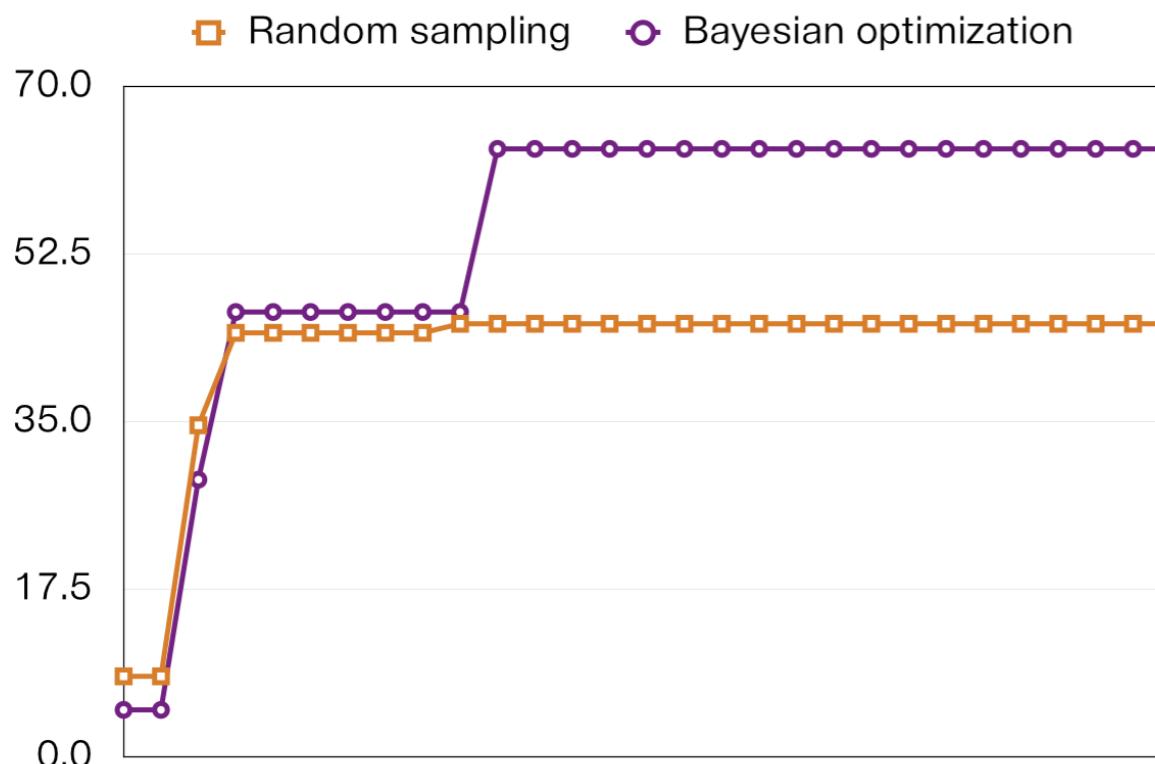
Find w that maximizes $\sum_{P_i} F(P_i, S_w(P_i))$

Solving the Opt. Problem

- How to solve the optimization problem efficiently?

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

- Using ideas of Bayesian optimization

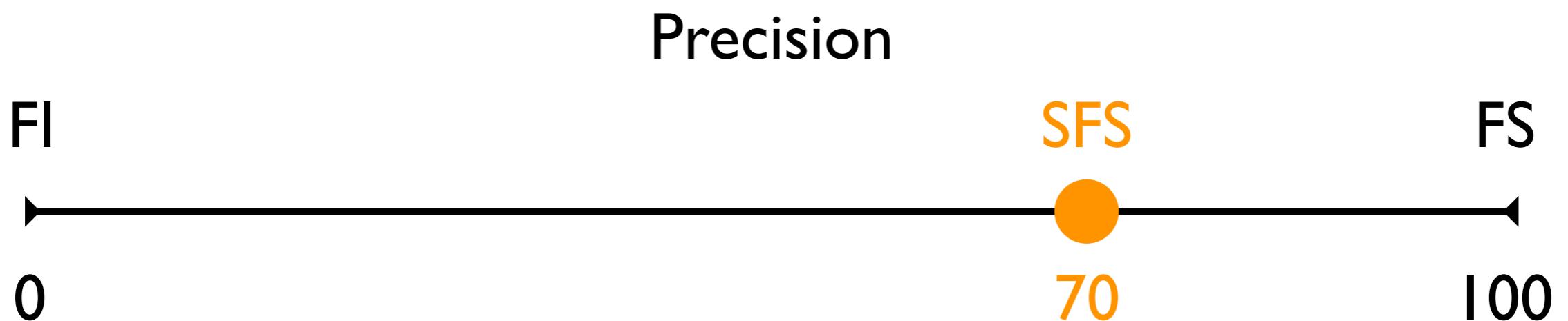


Effectiveness

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

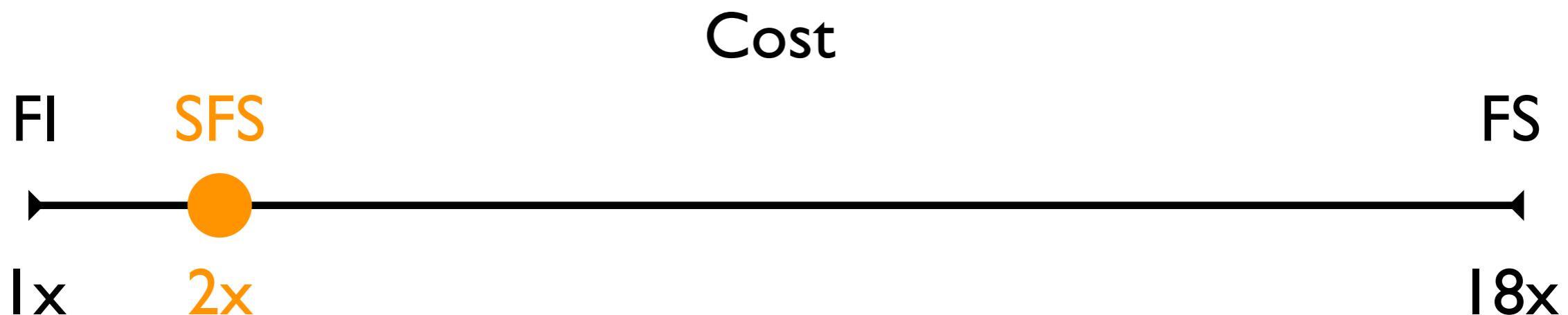
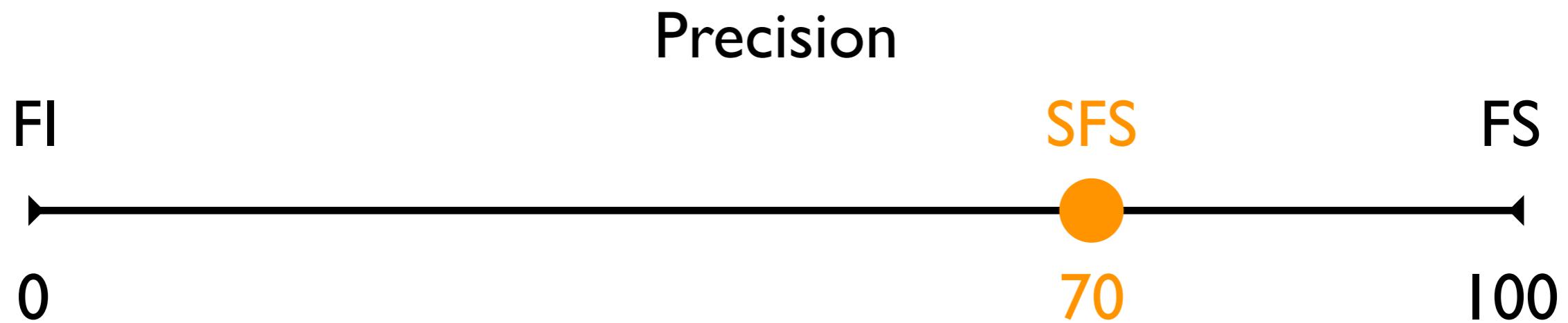
Effectiveness

- 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



Hurdle

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

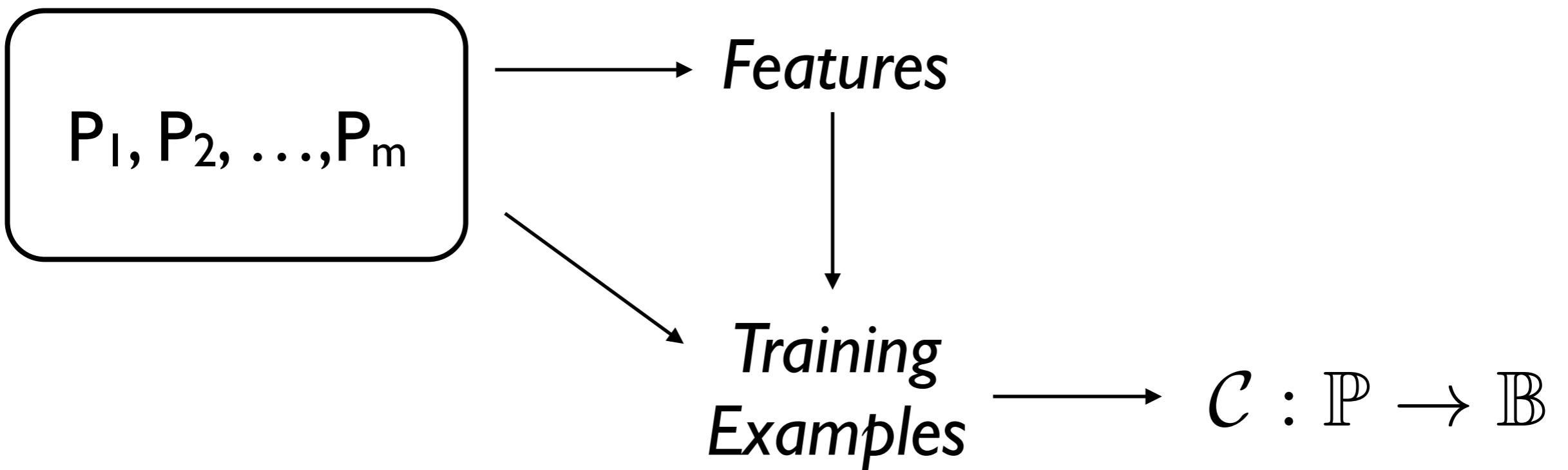
Type	#	Features
A	1	local variable
	2	global variable
	3	structure field
	4	location created by dynamic memory allocation
	5	defined at one program point
	6	location potentially generated in library code
	7	assigned a constant expression (e.g., $x = c1 + c2$)
	8	compared with a constant expression (e.g., $x < c$)
	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	$(21 \vee 22) \wedge \neg 23$

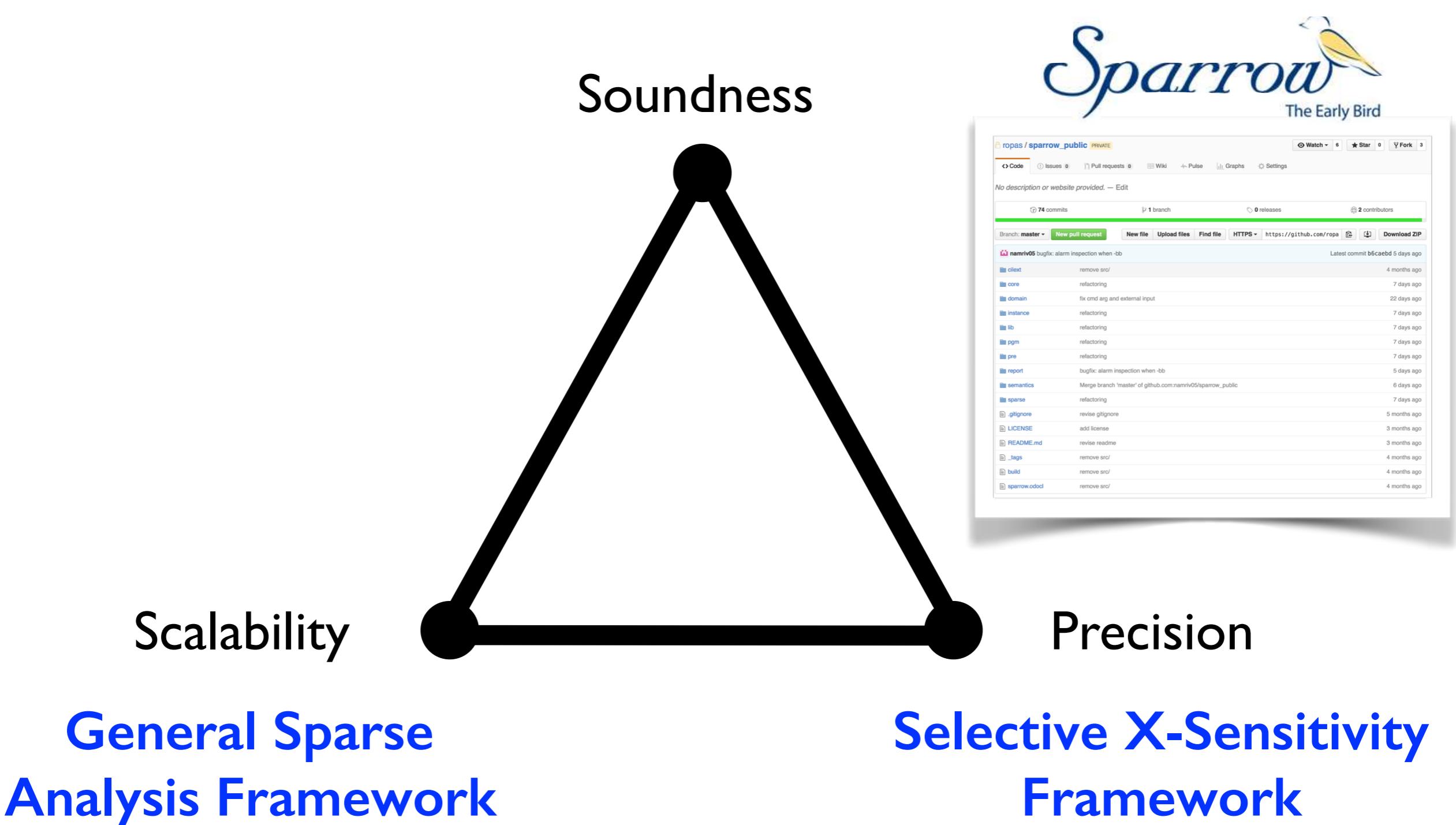
Type	#	Features
A	1	used in array declarations (e.g., a[c])
	2	used in memory allocation (e.g., malloc(c))
	3	used in the righthand-side of an assignment (e.g., $x = c$)
	4	used with the less-than operator (e.g., $x < c$)
	5	used with the greater-than operator (e.g., $x > c$)
	6	used with \leq (e.g., $x \leq c$)
	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	$13 \wedge 12$

Learning with Automatic Feature Construction

- Fully automatic learning approach

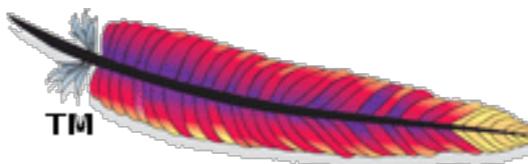


Powerful Static Analysis Enabled



Applications (I)

Security-critical softwares



Massive Security Bug In OpenSSL Could Affect A Huge Chunk Of The Internet

Posted Apr 7, 2014 by Greg Kumparak (@grg)

122 Like 14k Share 1,152 Tweet 1,490



I saw a t-shirt one time. "I'm a bomb disposal technician," it read. "If you see me running, to keep up."

The same sort of idea can be applied to net security: when all the net security people you know are freaking out, it's probably an okay time to worry.

This afternoon, many of the net security people I know are freaking out. A very serious bug in OpenSSL — a cryptographic library that

Sendmail disasters

These are the most serious sendmail security and reliability problems through sendmail 8.8.7 in 1997. Unattributed quotes here are from All

- Security verifiers for OpenSSL, Apache, Sendmail, etc

Applications (2)

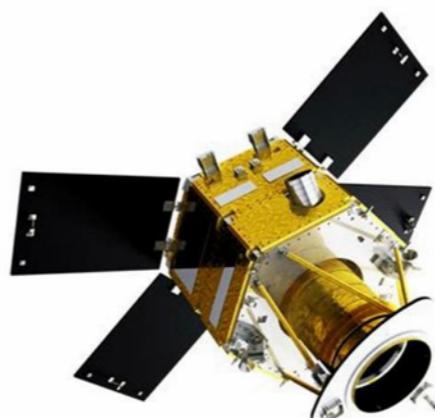
Safety-critical softwares



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)

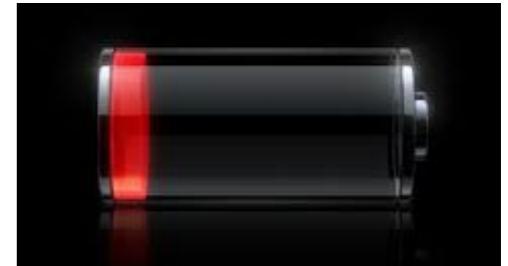
Efficient uses of modern computing platforms

- Mobile
- Cloud
- Parallel
- Wearable
- ...

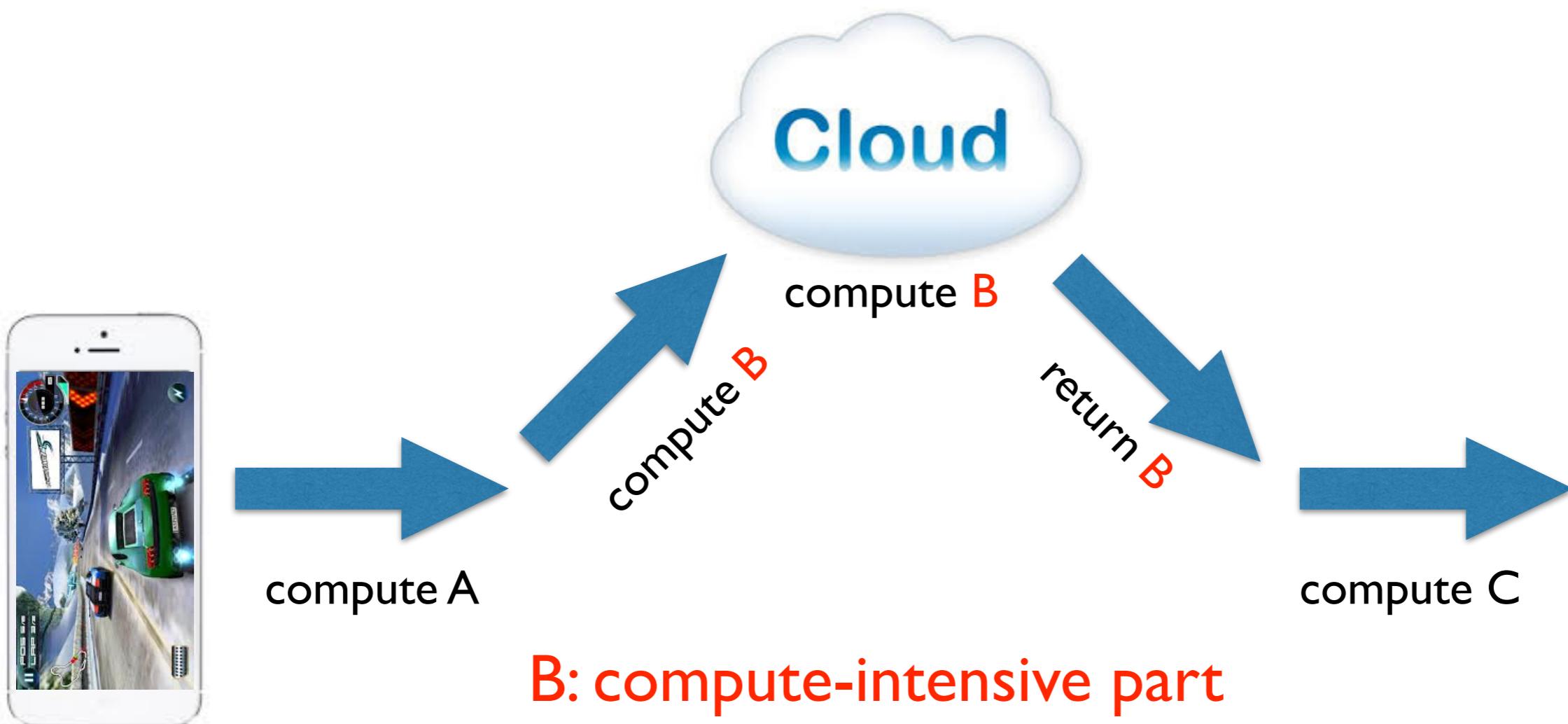
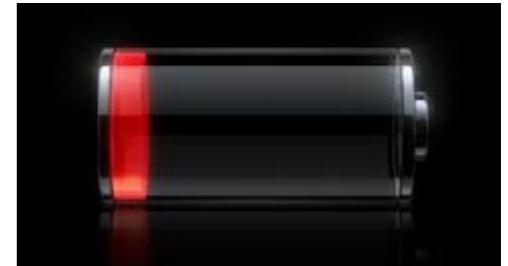


- New Software challenges:
e.g., reliability, energy-efficiency, security, ...

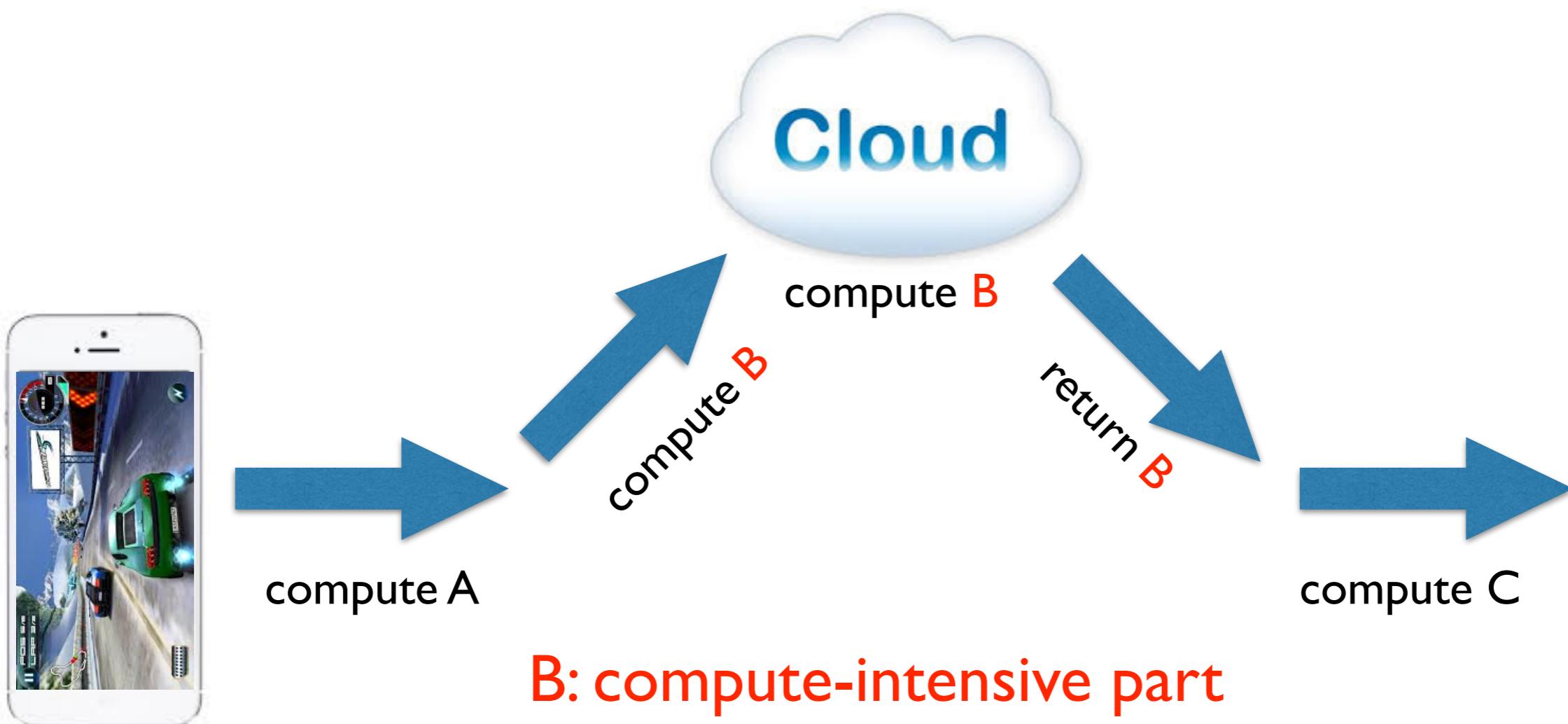
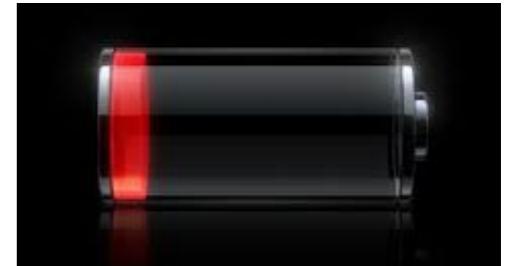
Static Analysis for Mobile Computing



Static Analysis for Mobile Computing



Static Analysis for Mobile Computing



Plan: Static analysis to estimate power consumption

Summary

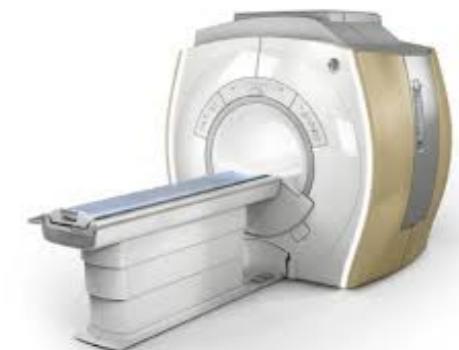
Mobile / Cloud / Parallel Computing



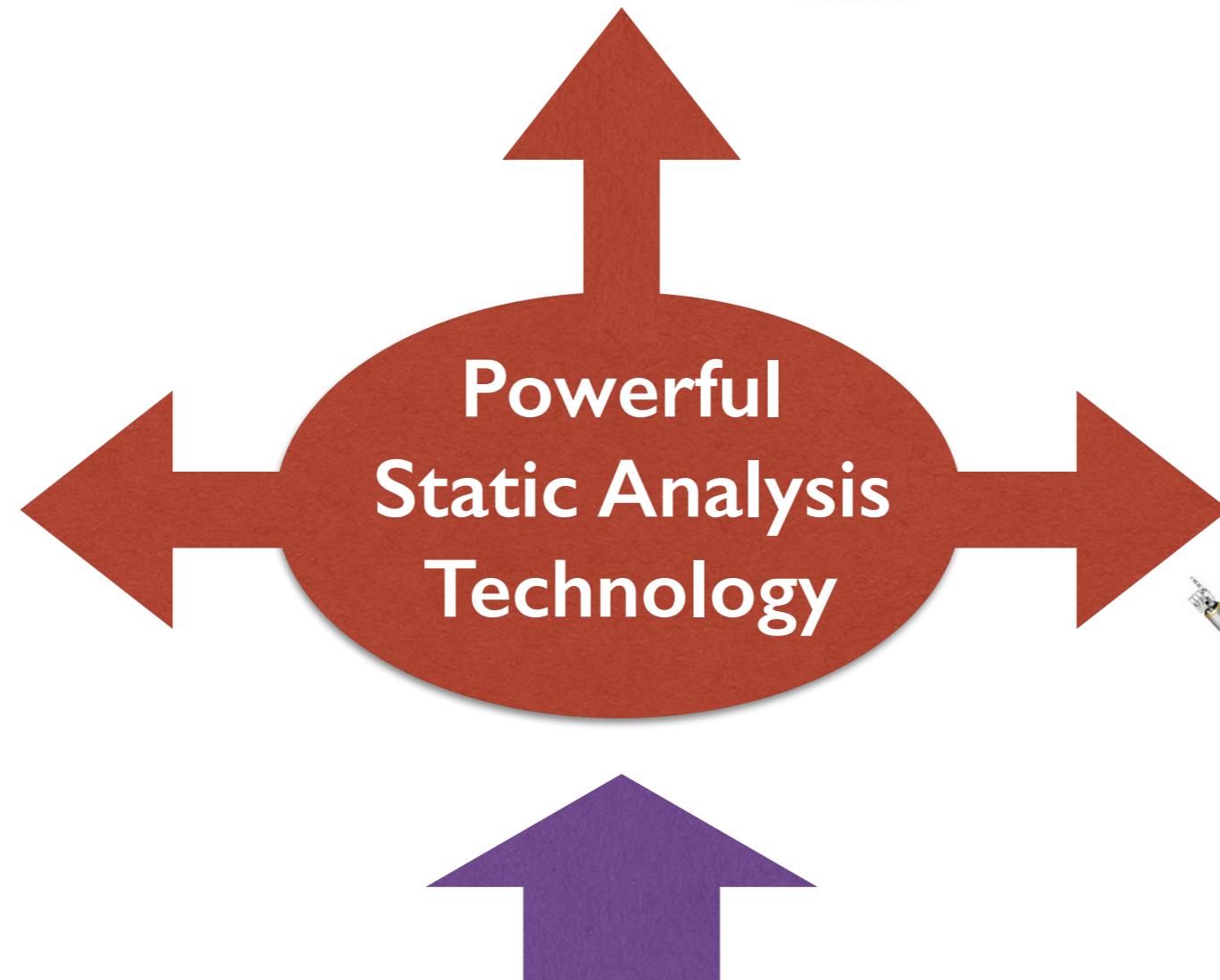
SW Security



SW Verification



**Powerful
Static Analysis
Technology**



Programming Languages Theories