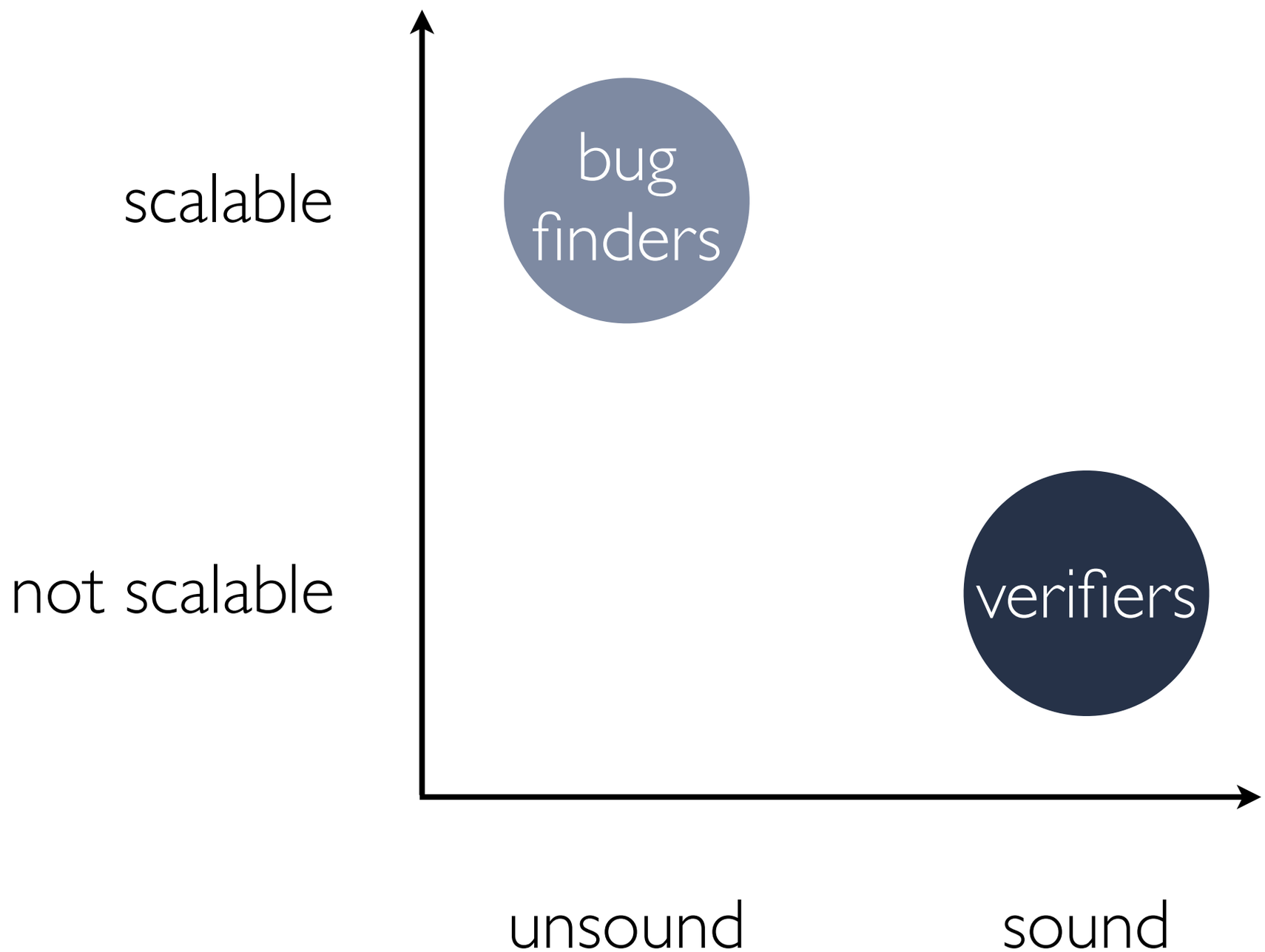
Design and Implementation of Sparse Global Analyses for C-like Languages

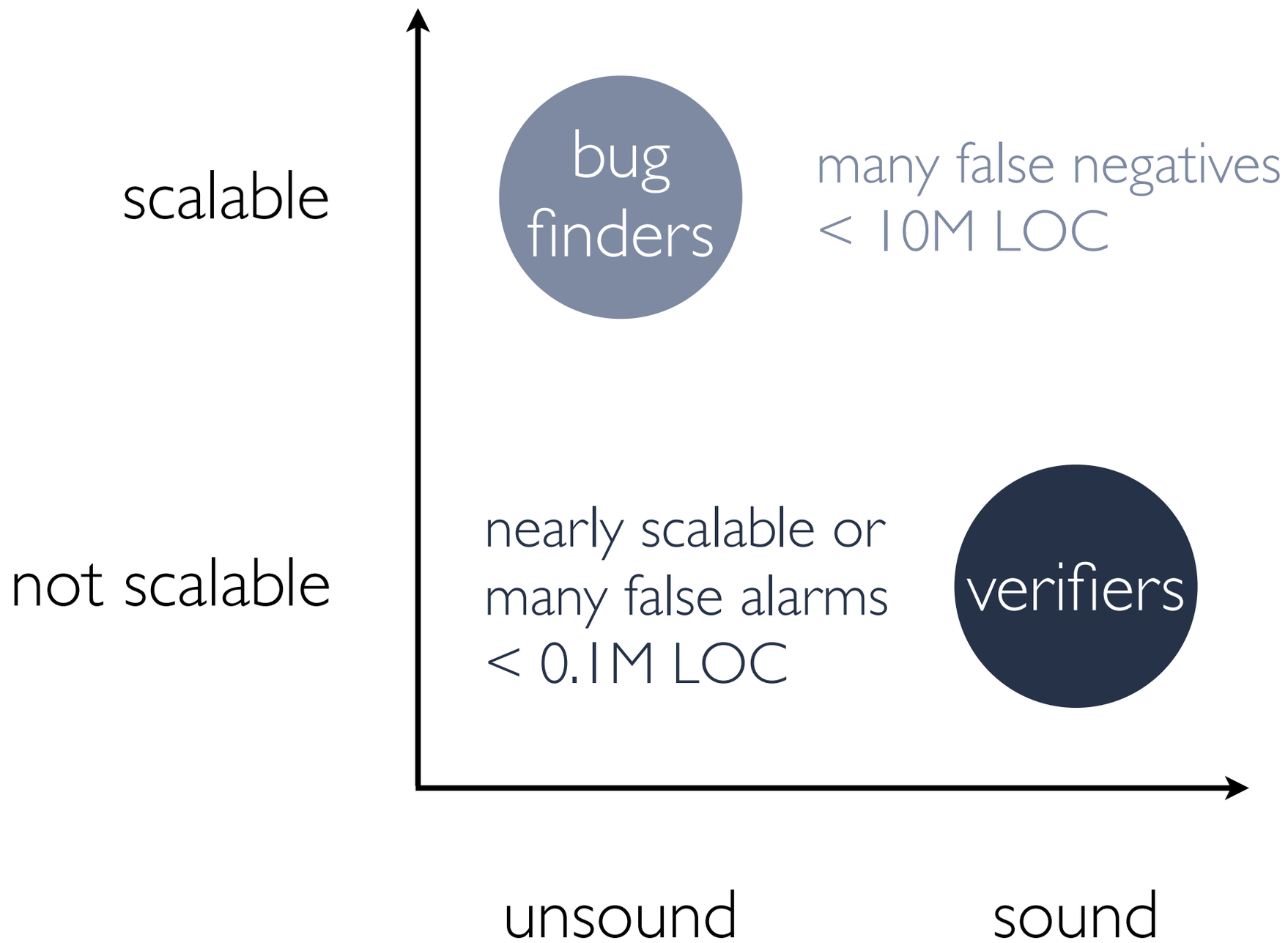
Wonchan Lee
wclee@ropas.snu.ac.kr
(with Hakjoo Oh, Kihong Heo, Woosuk Lee, and Kwangkeun Yi)

April 3rd, 2012
@Sunblaze group meeting

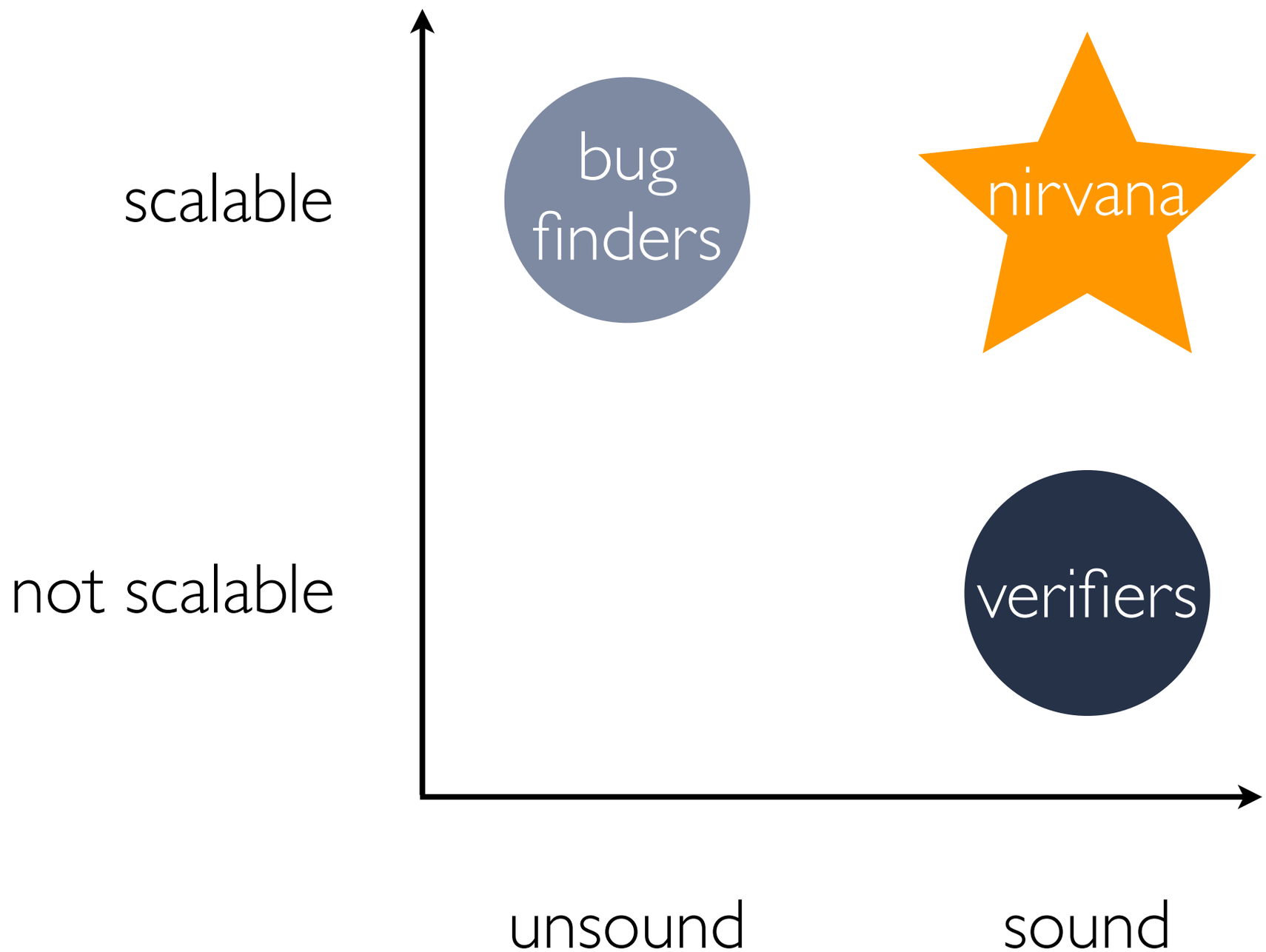
Dichotomy: “Bug-finders” vs. “Verifiers”



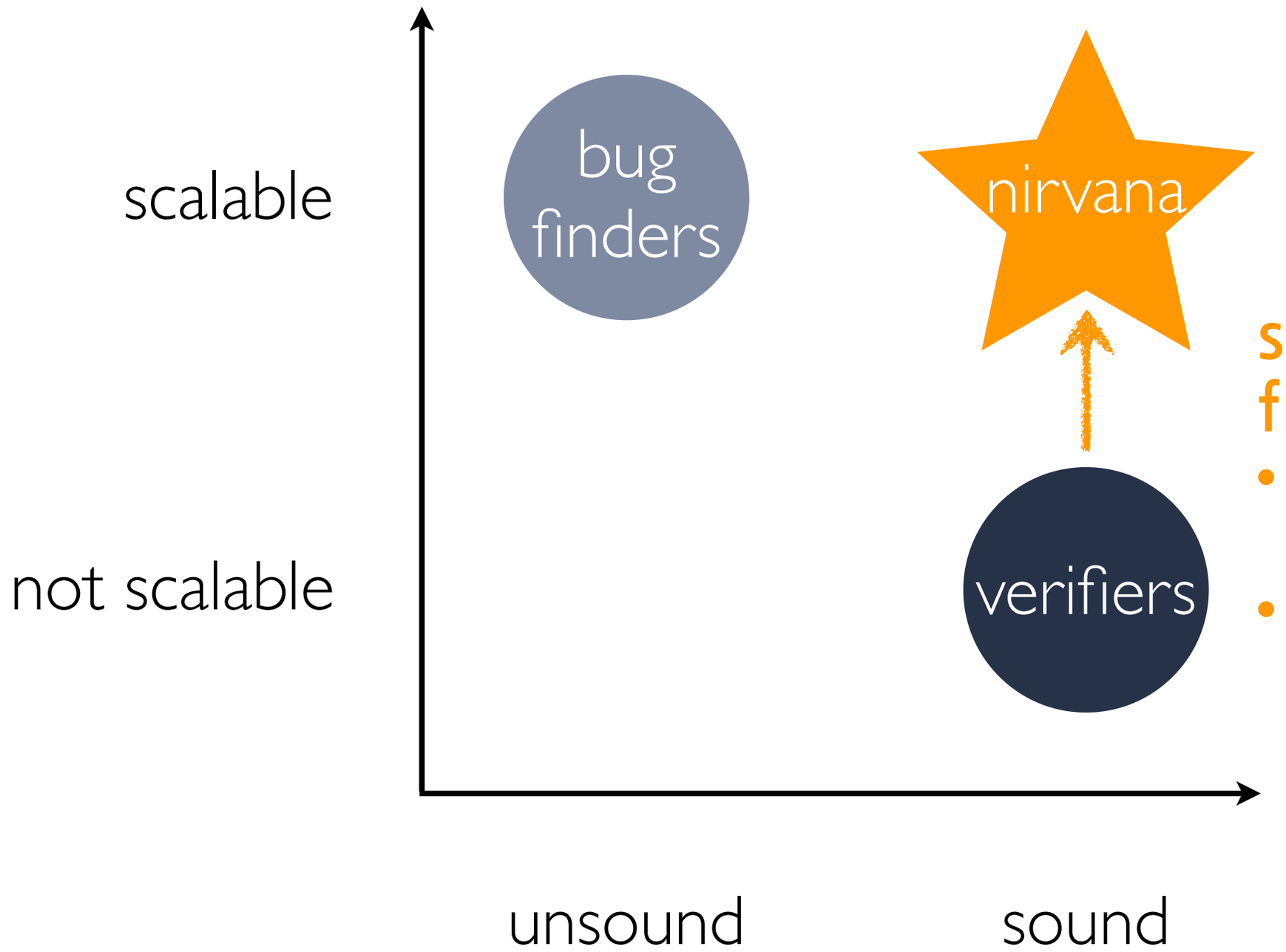
Dichotomy: “Bug-finders” vs. “Verifiers”



Goal: Sound, Precise, and Scalable Analysis



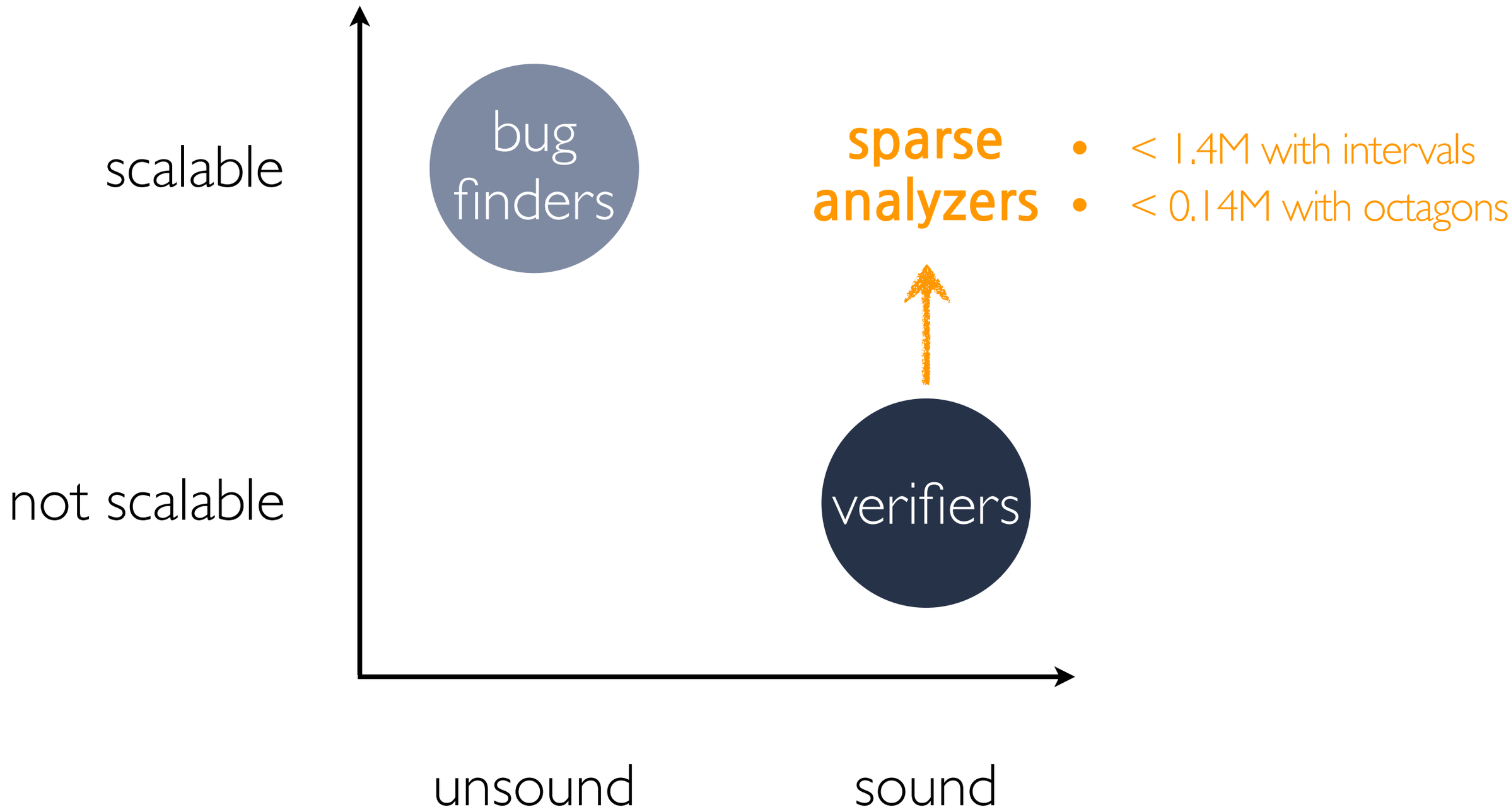
Our Contributions



sparse analysis framework

- general for AI-based analyzers
- precision-preserving

Our Contributions

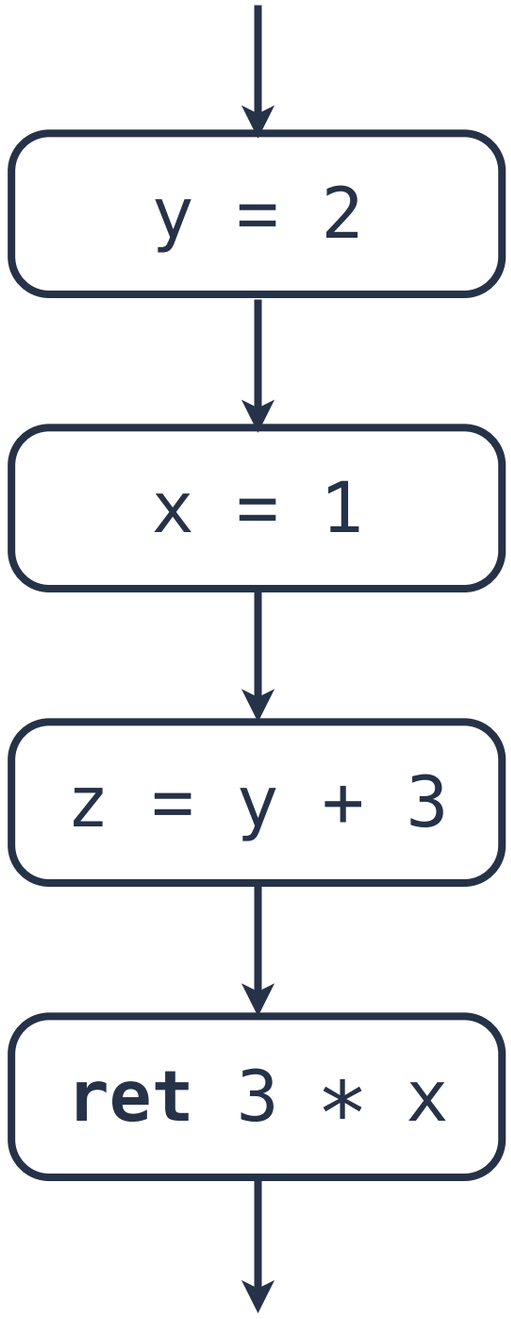


Contents

- Source of inefficiency in static analysis
- Sparse analysis
- Experiment results

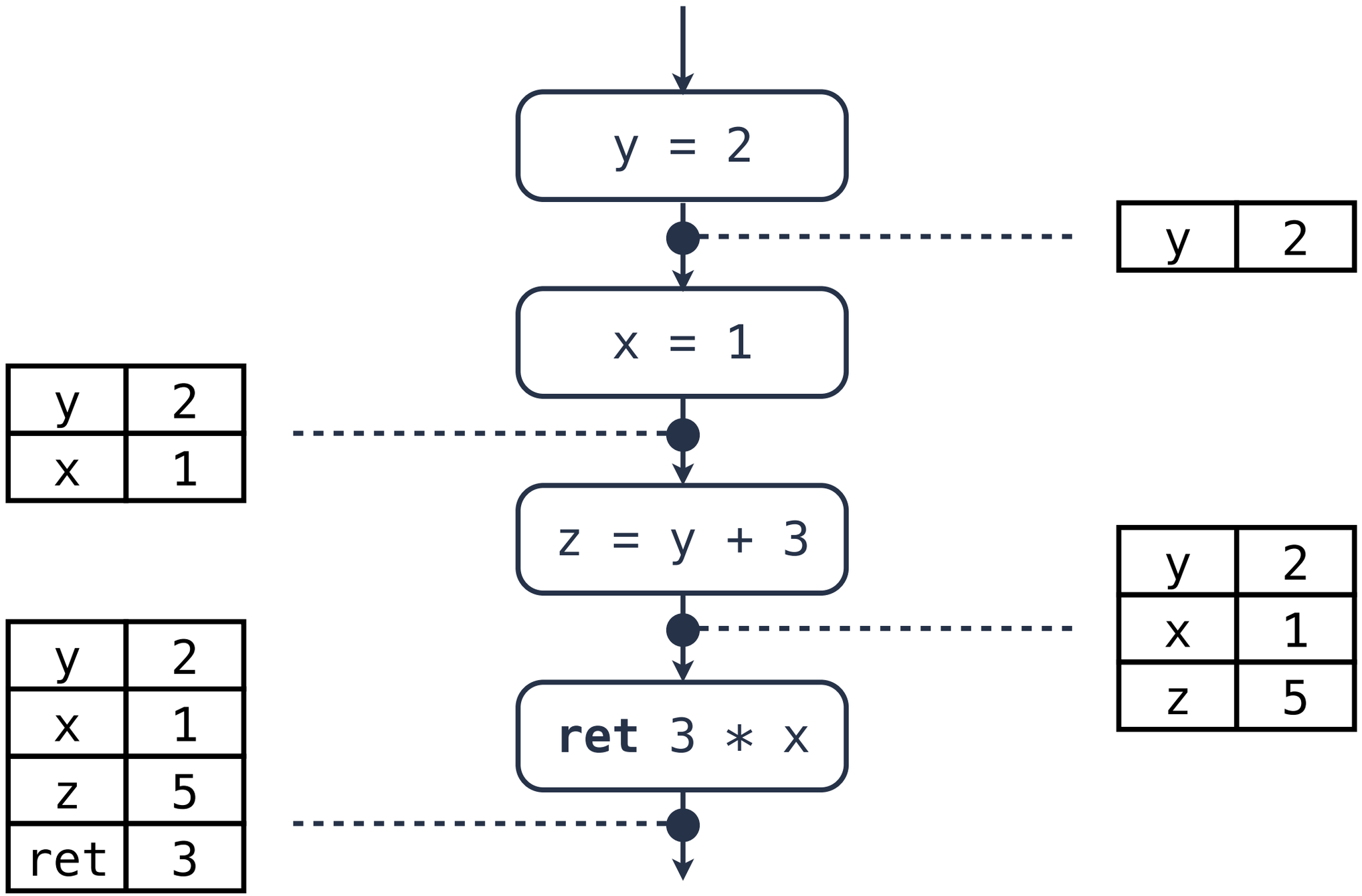
Simple Static Analysis

Reachable states at each control point



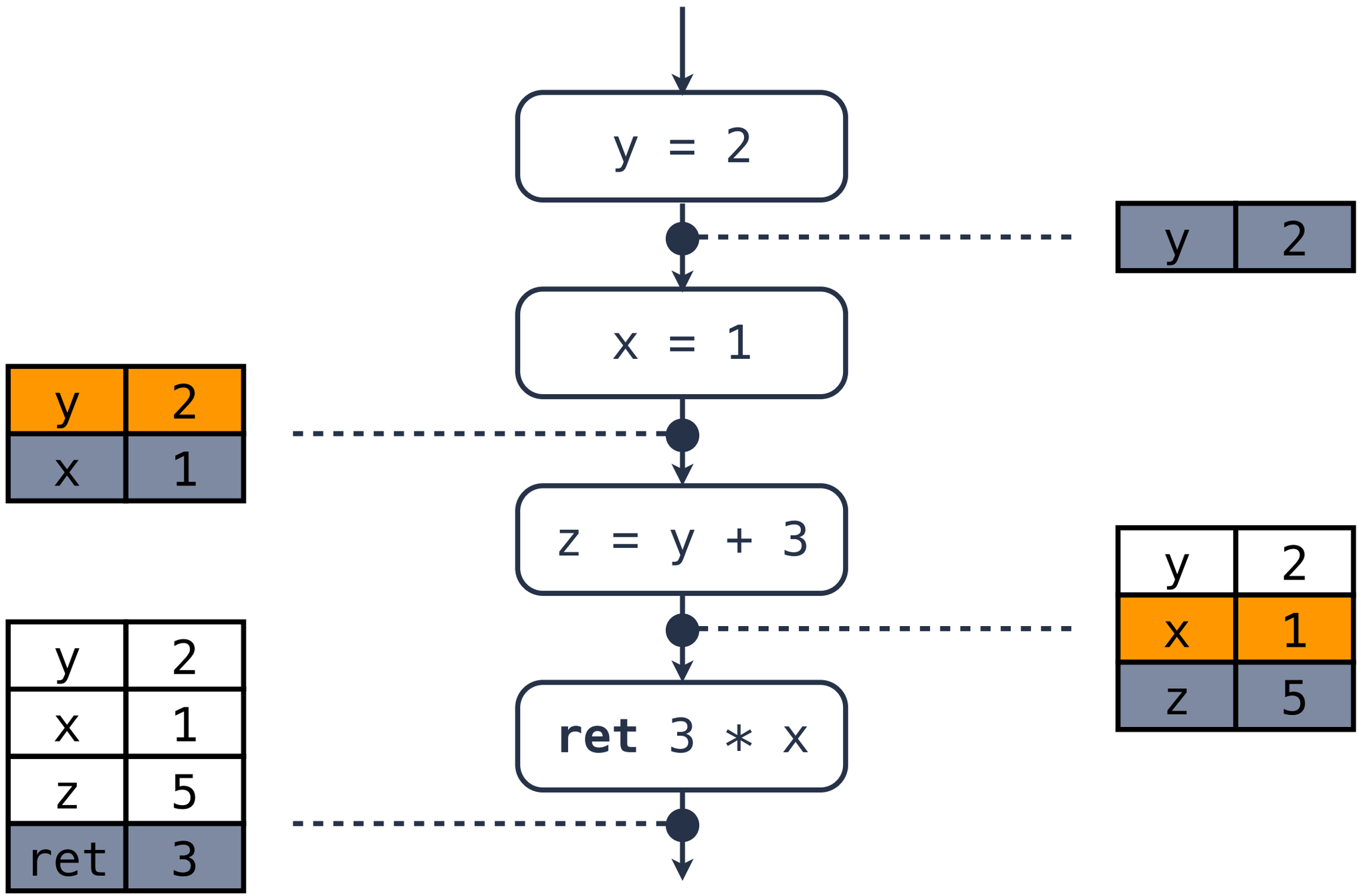
Simple Static Analysis

Reachable states at each control point



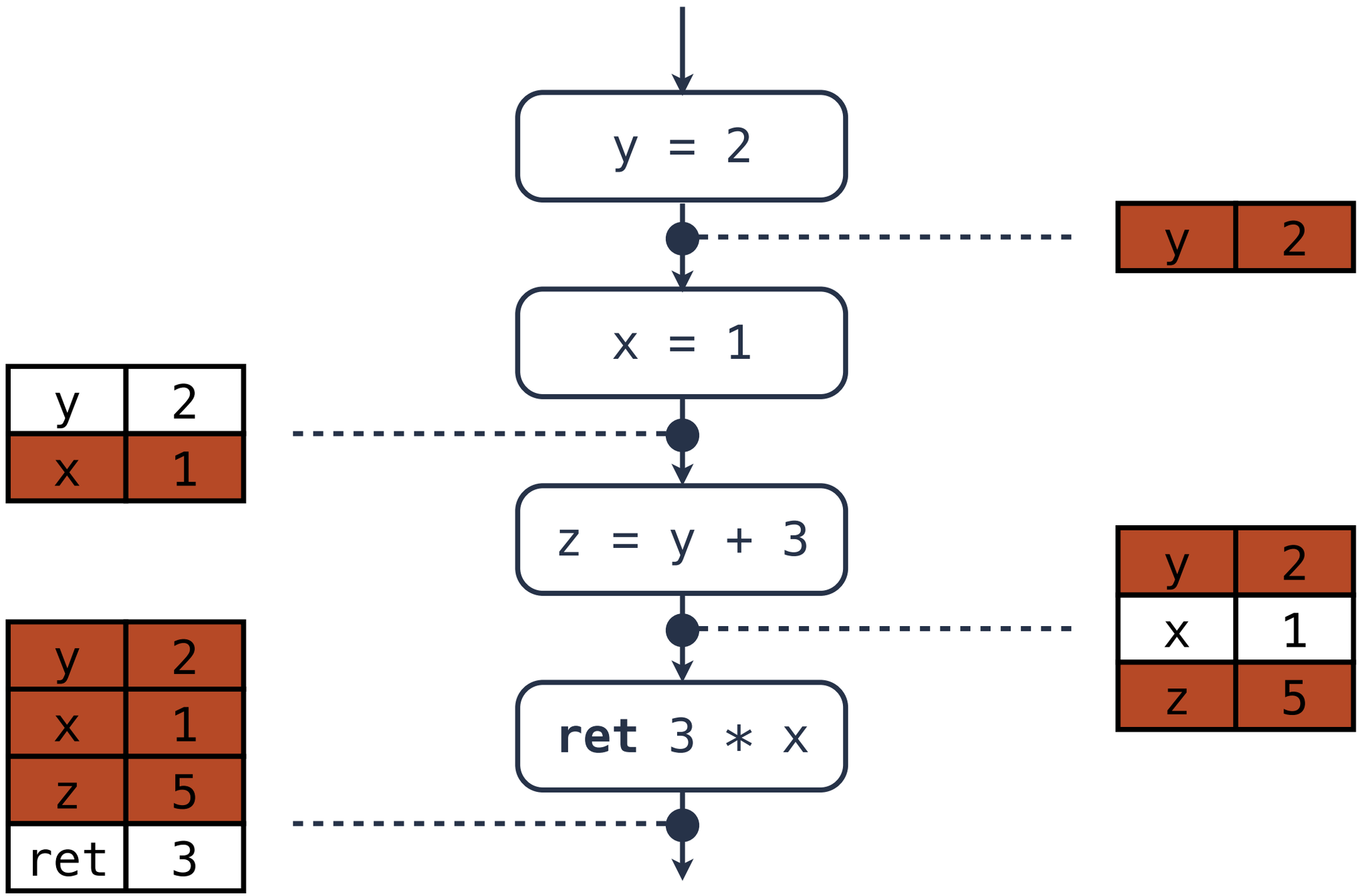
Sparsity in Transfer Function

Values are used and defined in a sparse manner

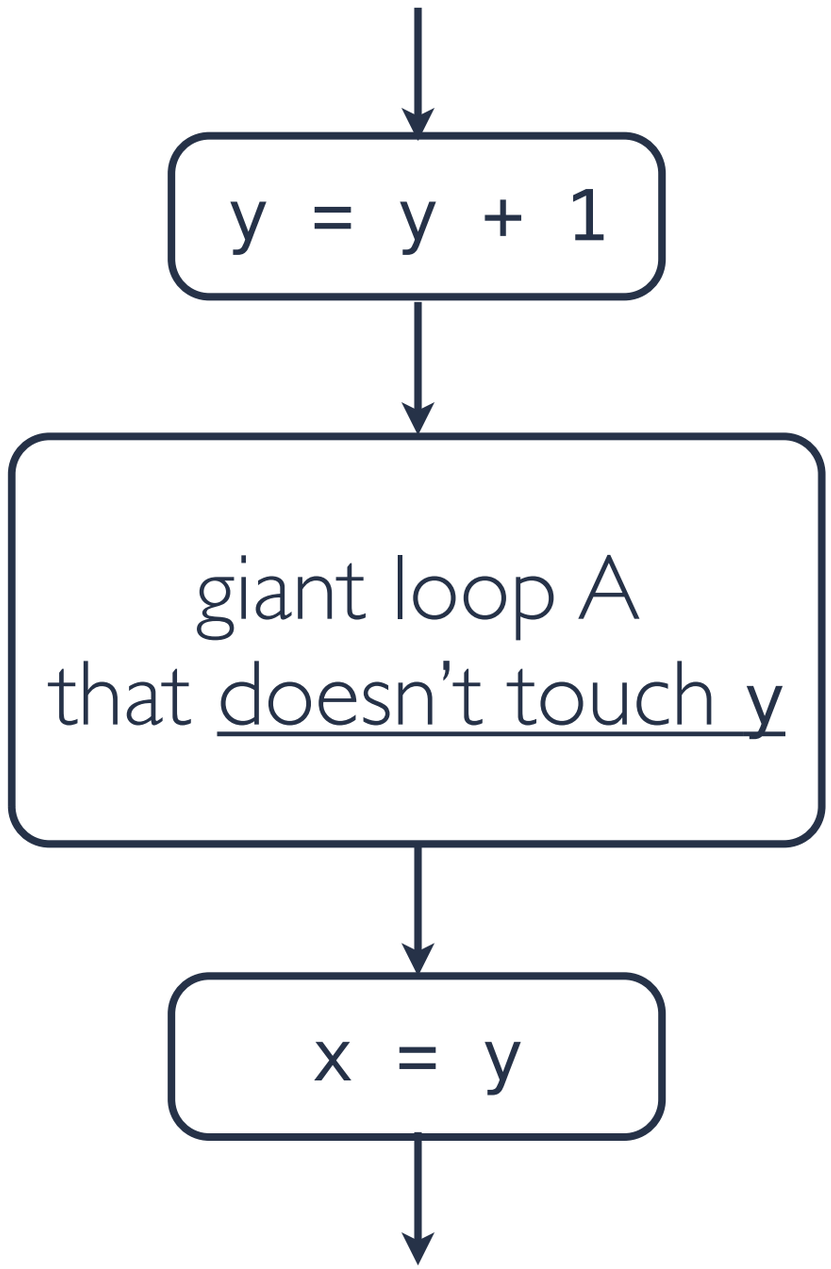


Inefficiency: Unnecessary Propagation

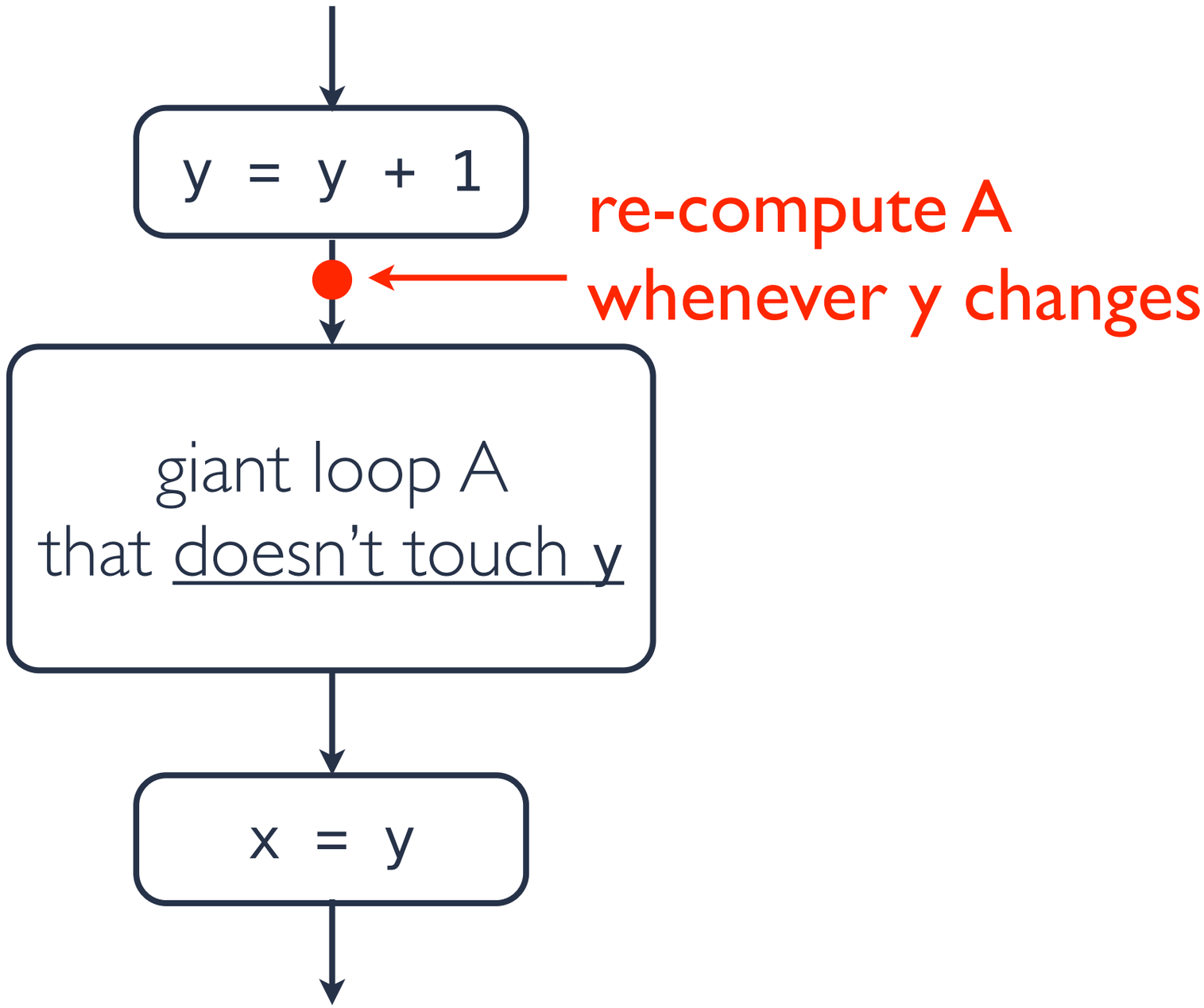
Some values are unnecessarily propagated untill they're actually used



What Really Happens

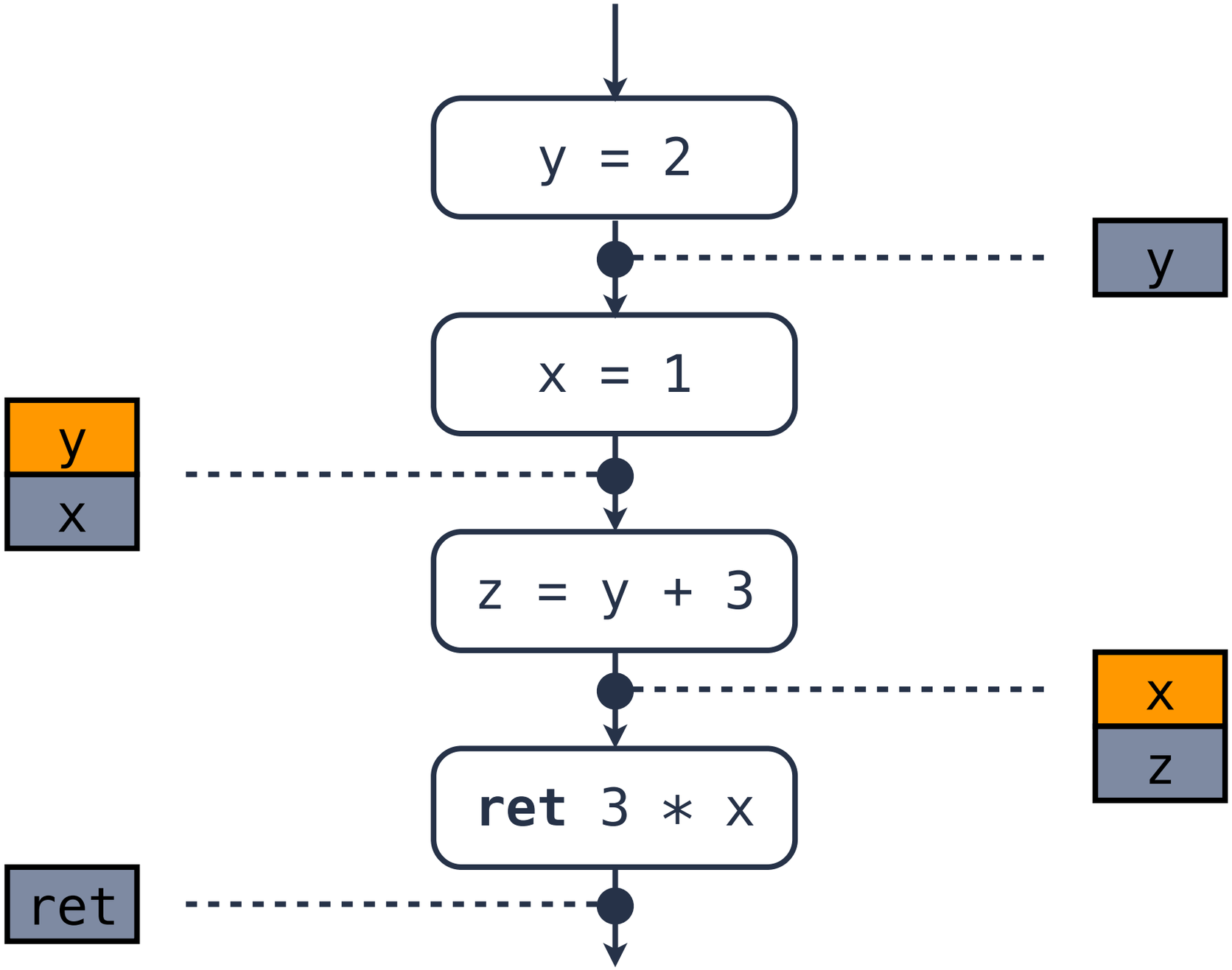


What Really Happens



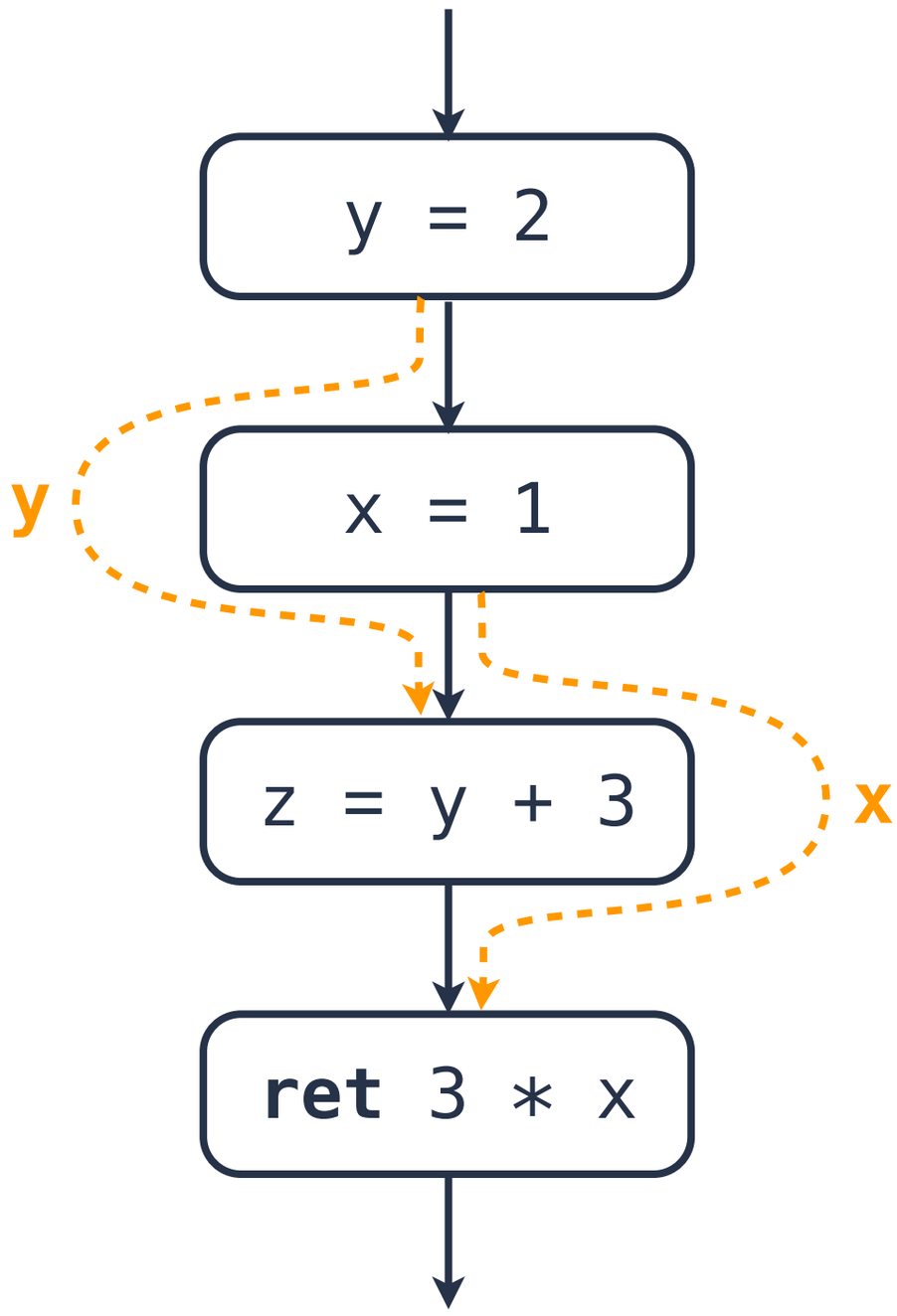
Sparse Analysis: Exploiting Sparsity in TF

Idea: propagating definitions directly to their uses



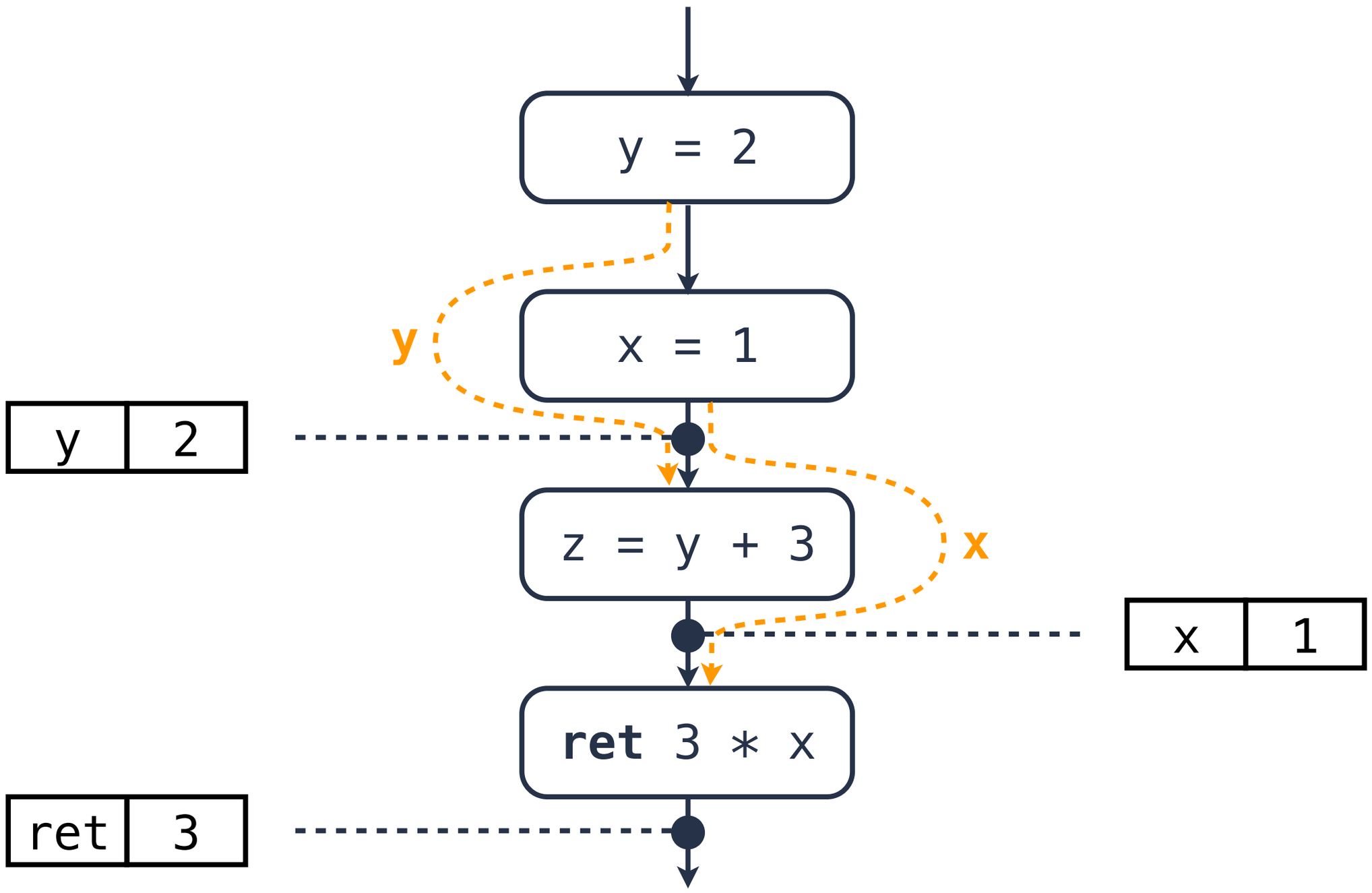
Exploiting Sparsity with DFG

Idea: propagating definitions directly to their uses with DFG



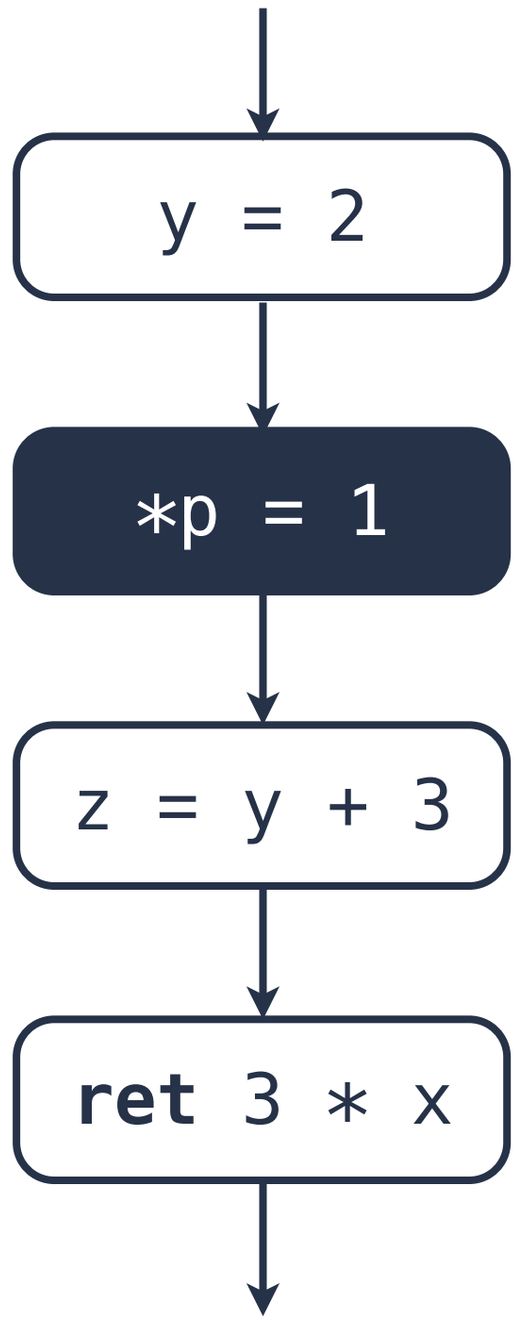
Exploiting Sparsity with DFG

Idea: propagating definitions directly to their uses with DFG



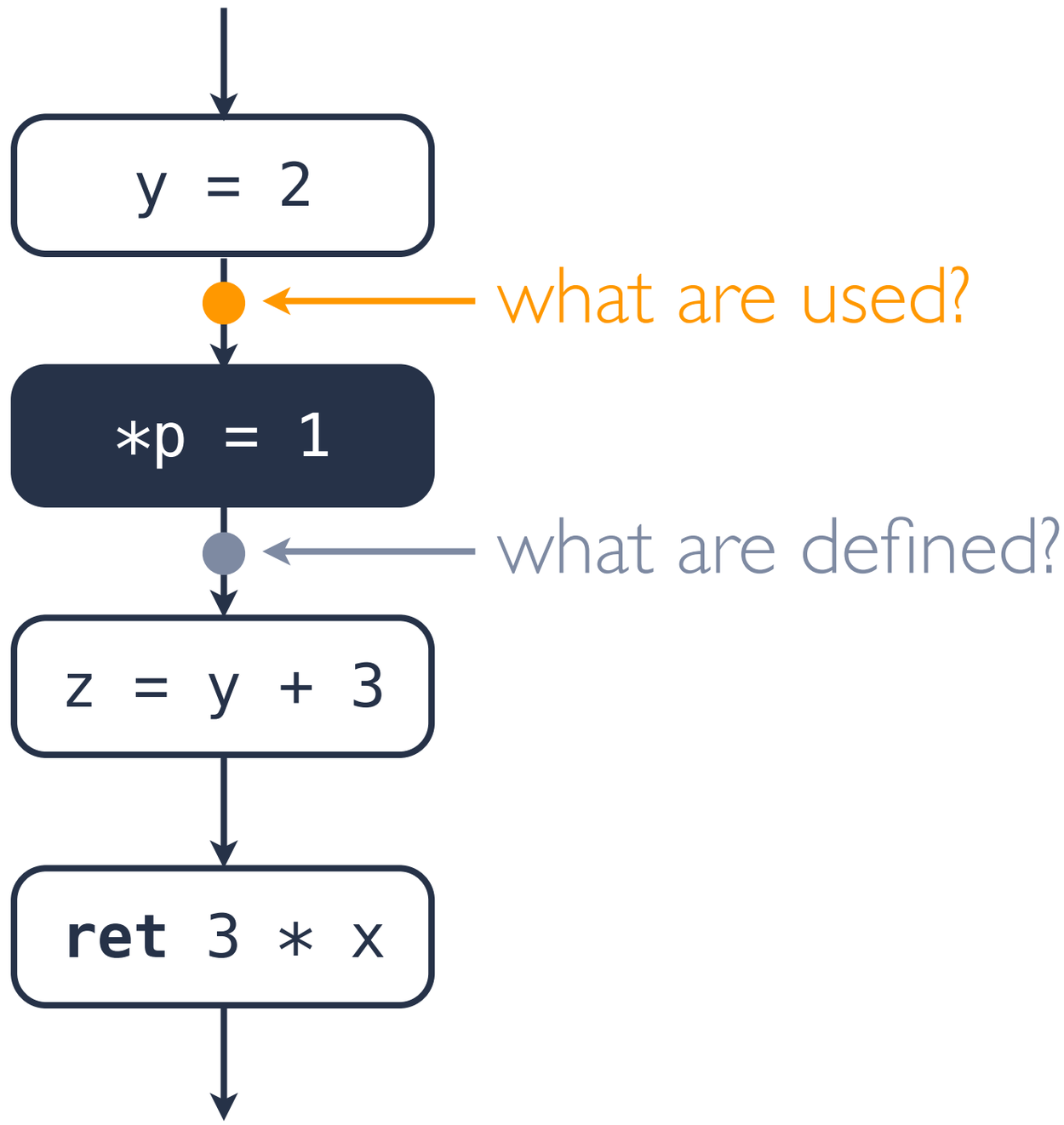
Technical Hurdle

Definitions and uses are sometimes determined at runtime
- Mostly because of pointers in C-like languages



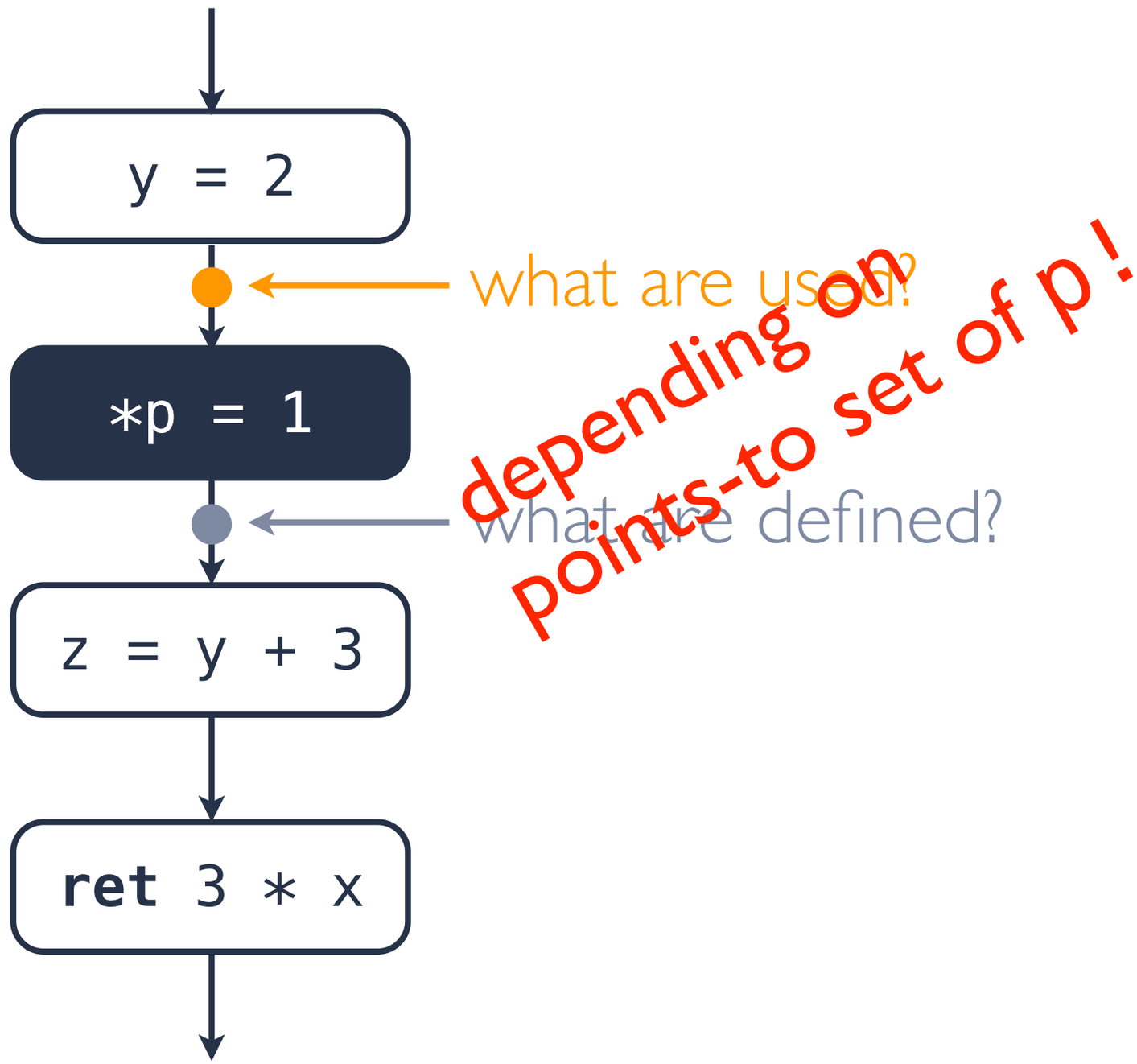
Technical Hurdle

Definitions and uses are sometimes determined at runtime
- Mostly because of pointers in C-like languages



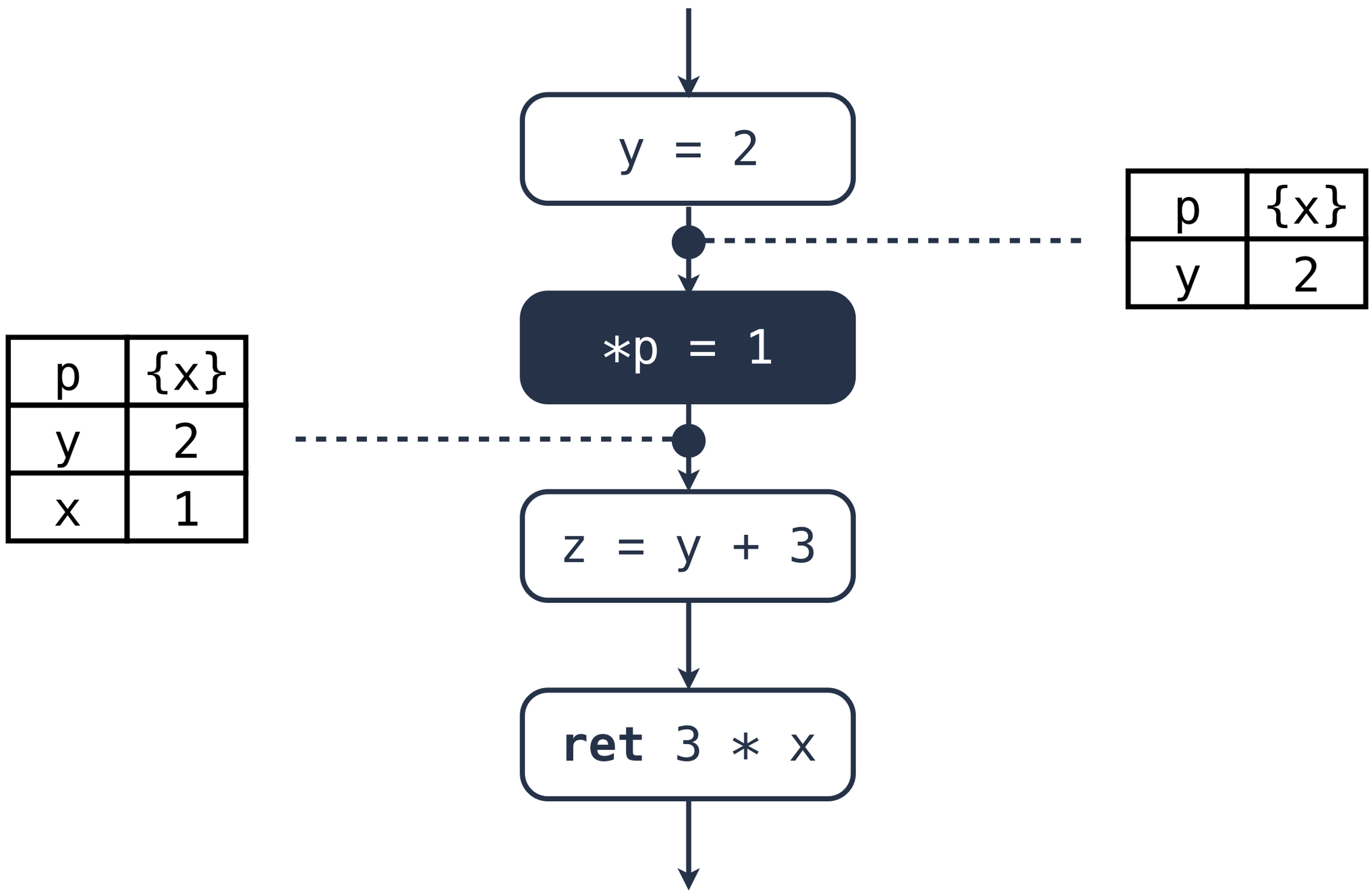
Technical Hurdle

Definitions and uses are sometimes determined at runtime
- Mostly because of pointers in C-like languages



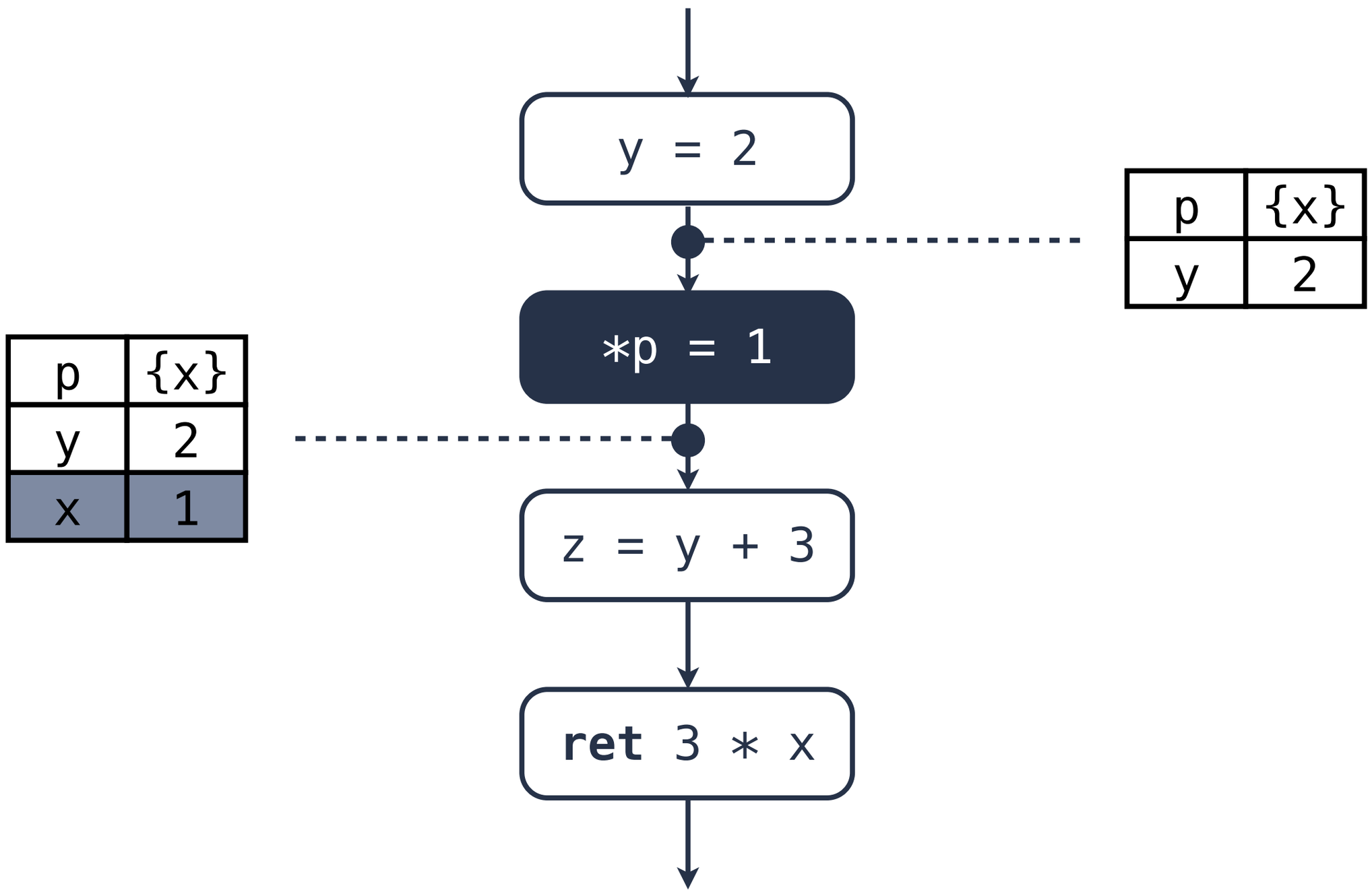
Technical Hurdle

Case 1) strong update



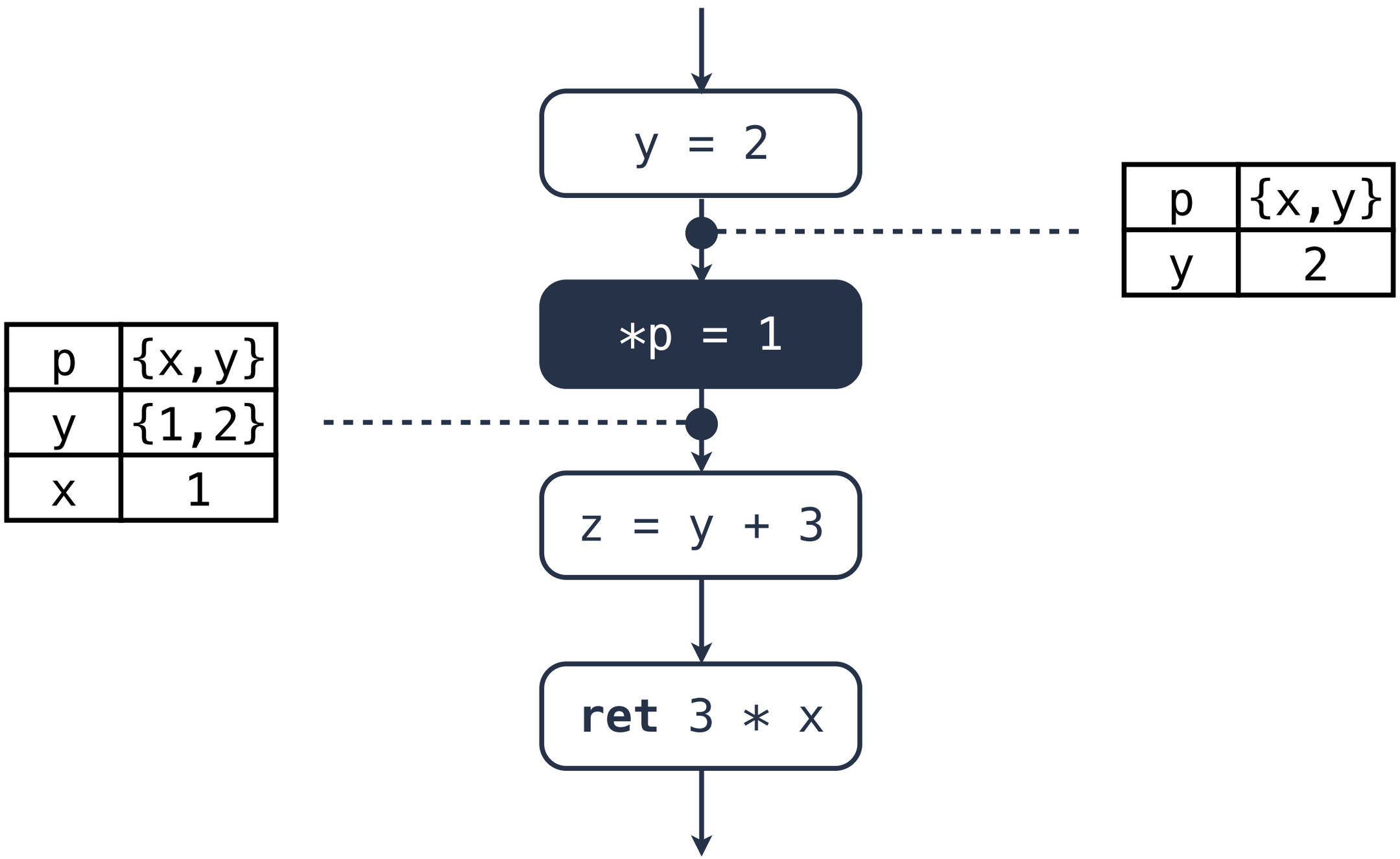
Technical Hurdle

Case 1) strong update



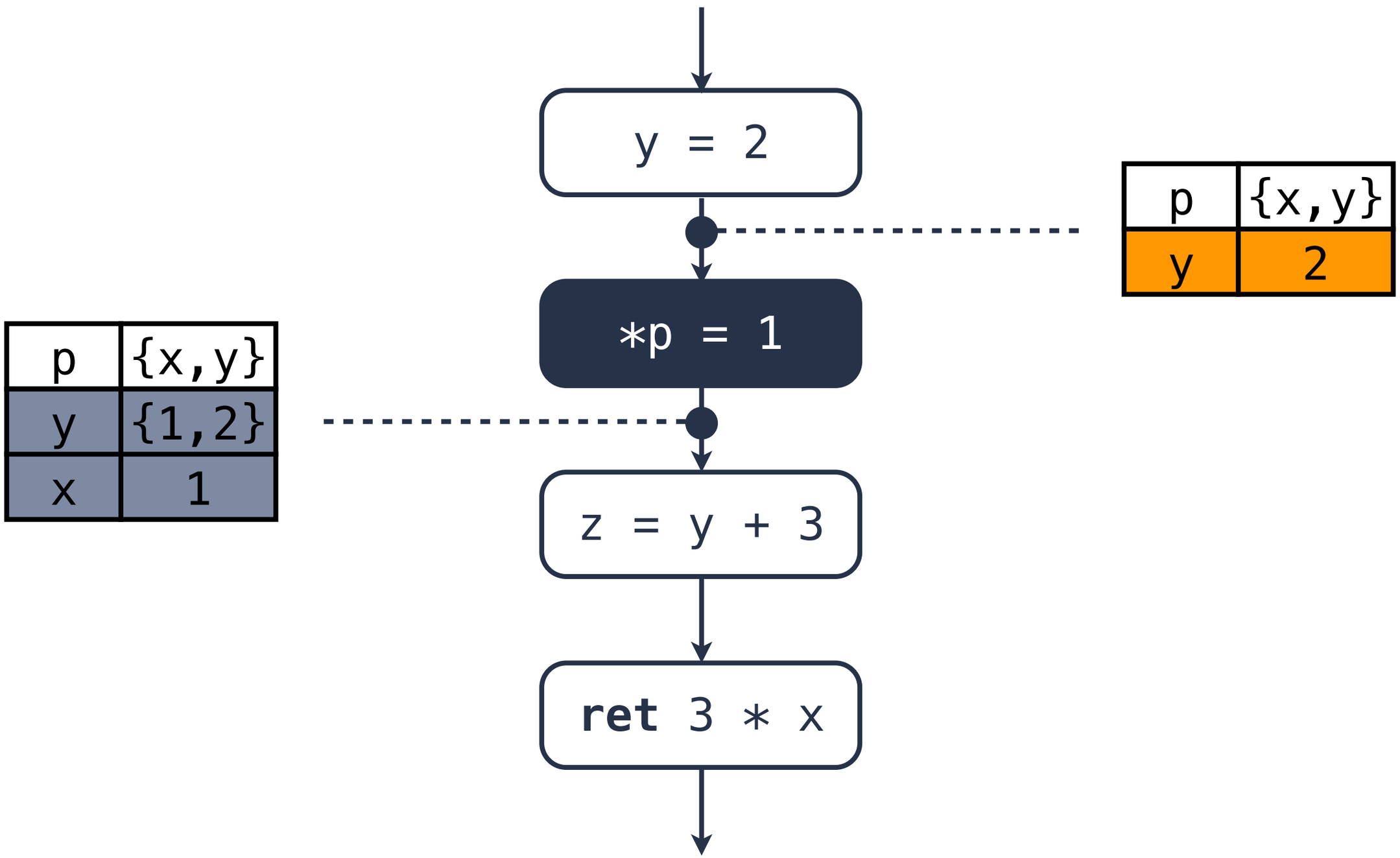
Technical Hurdle

Case 2) weak update



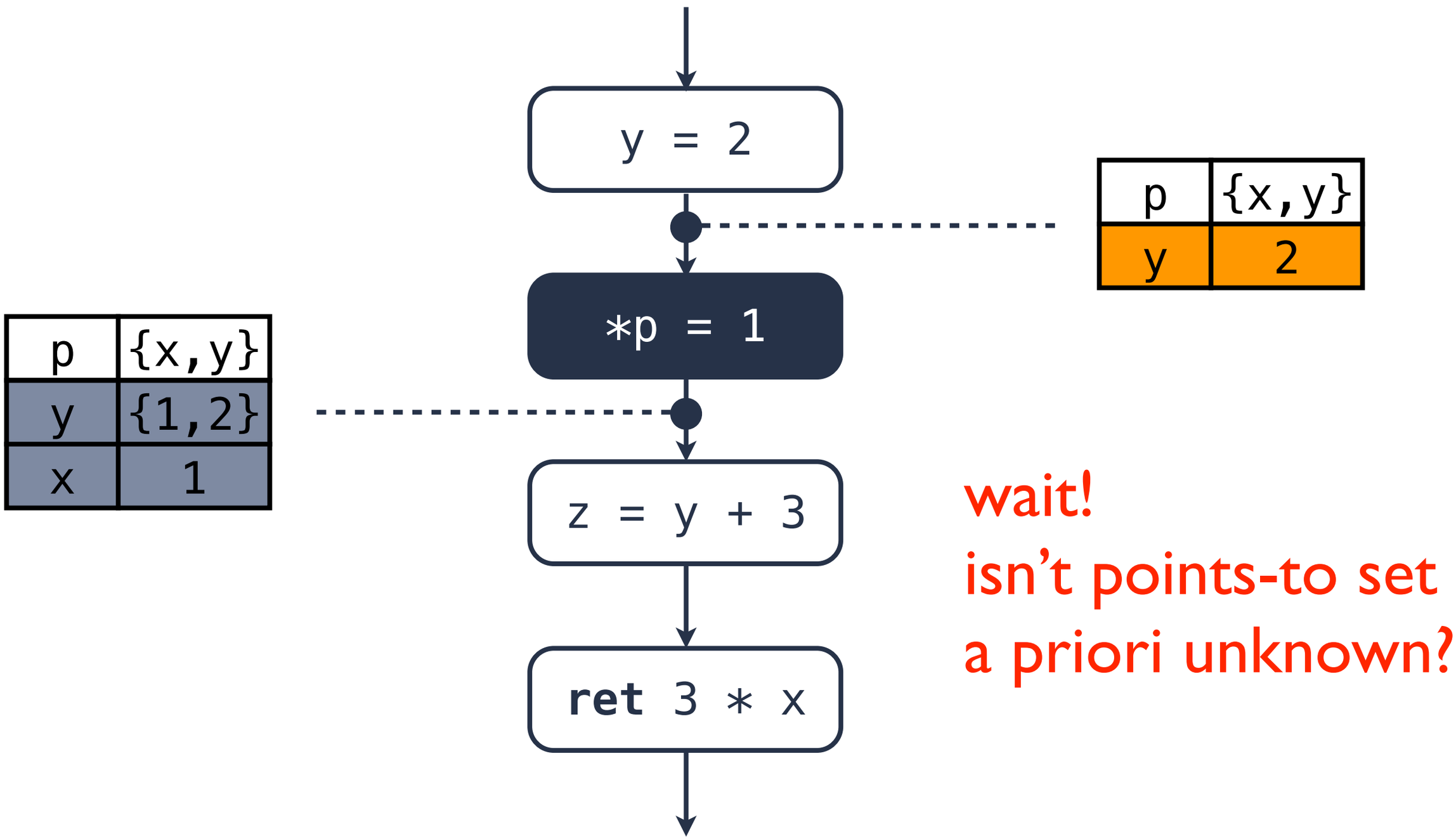
Technical Hurdle

Case 2) weak update



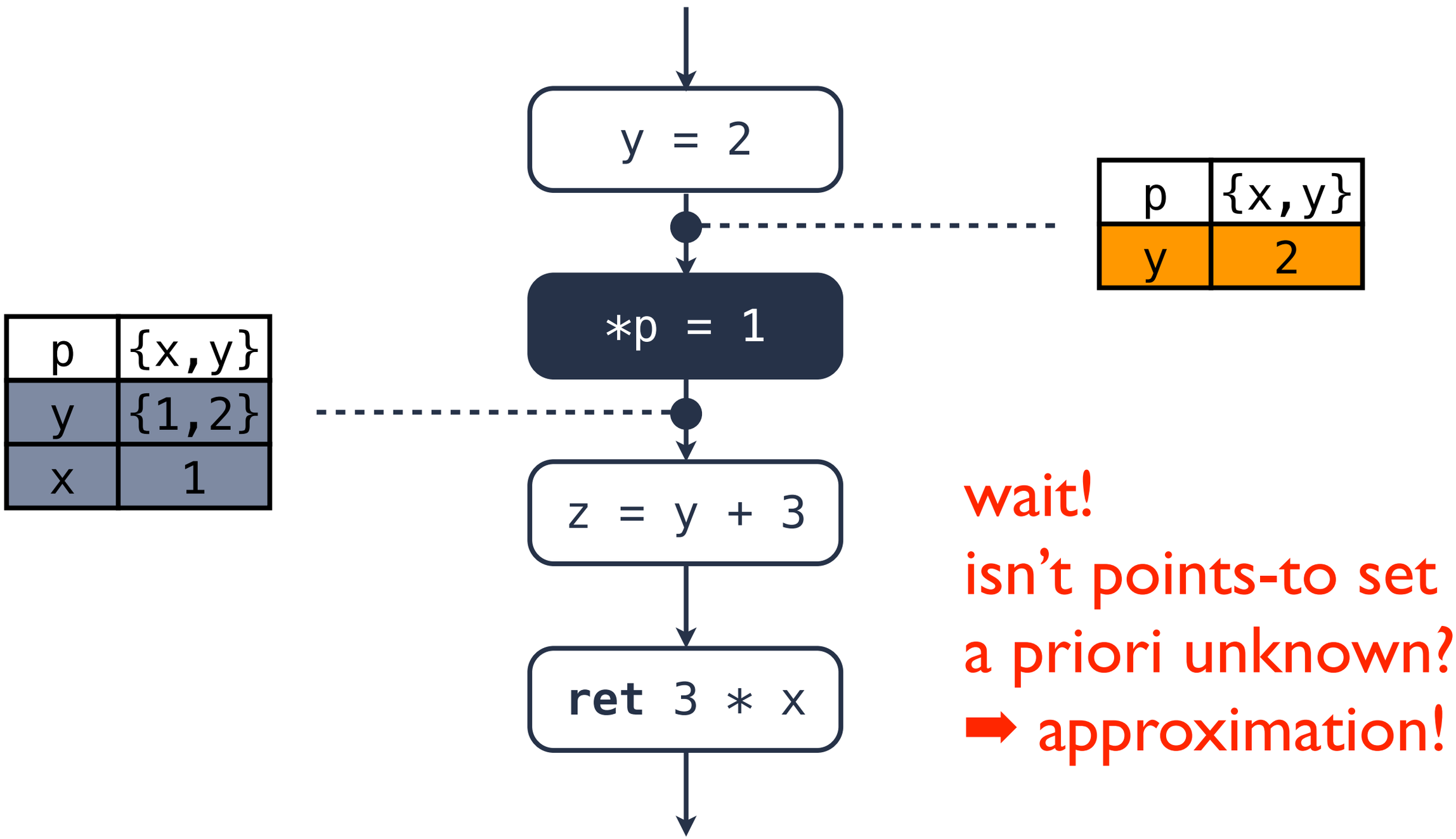
Technical Hurdle

Case 2) weak update

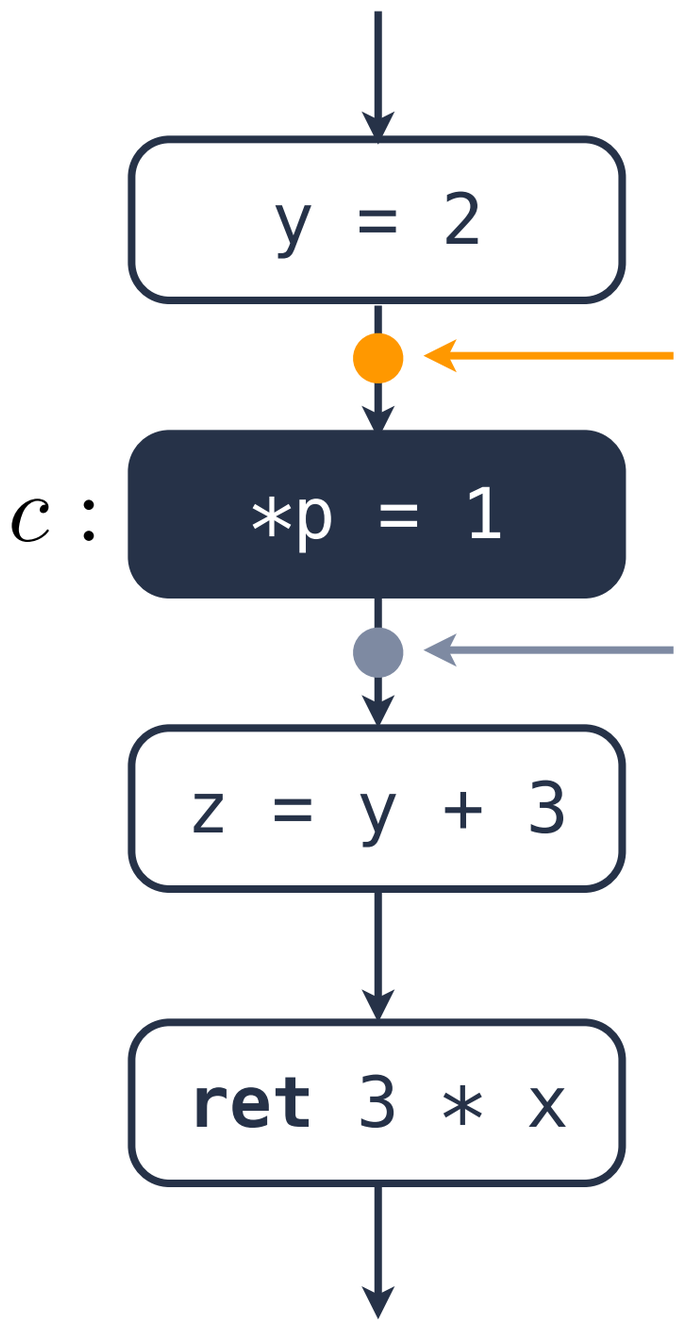


Technical Hurdle

Case 2) weak update



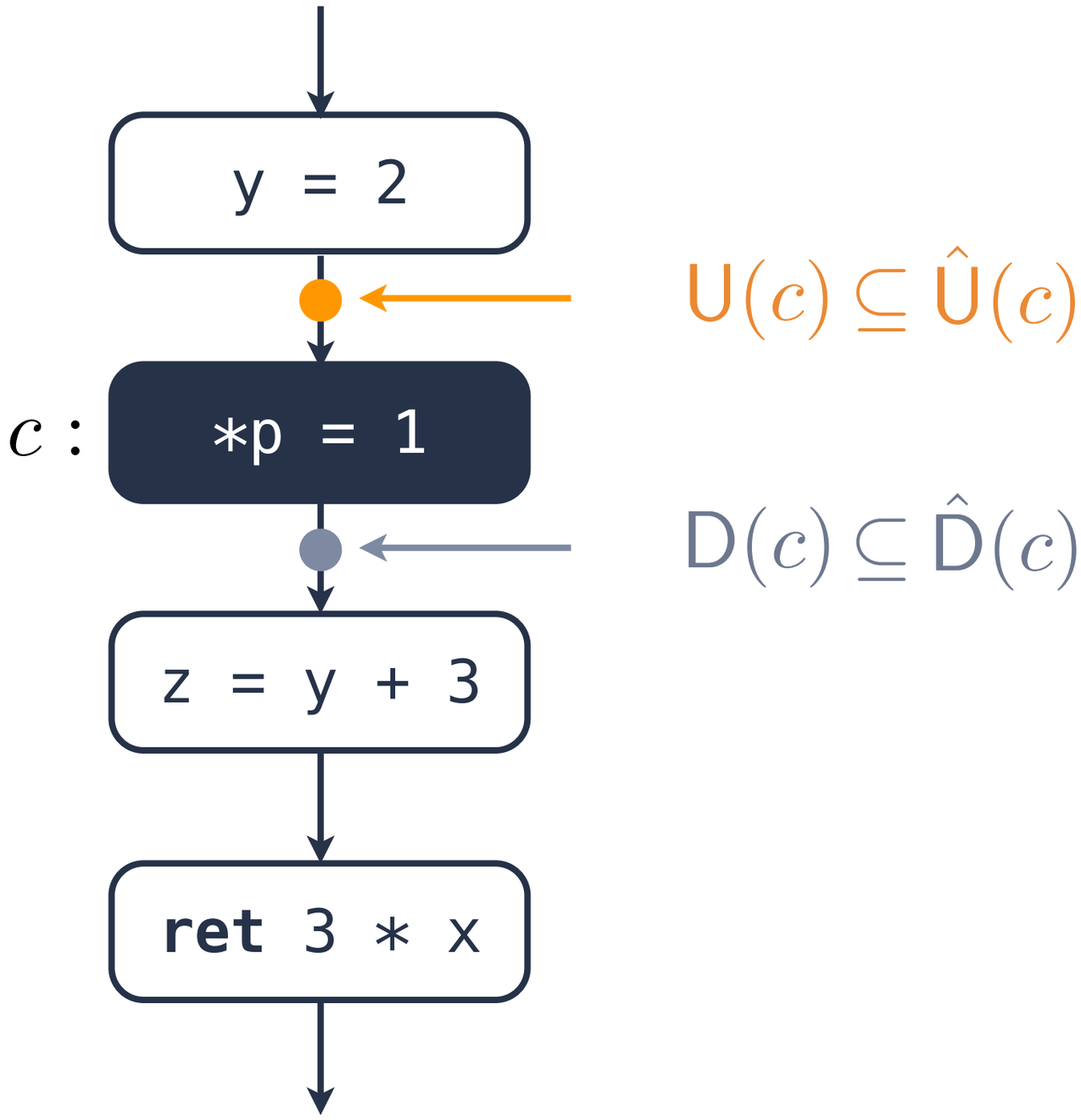
Approximation of Definitions and Uses



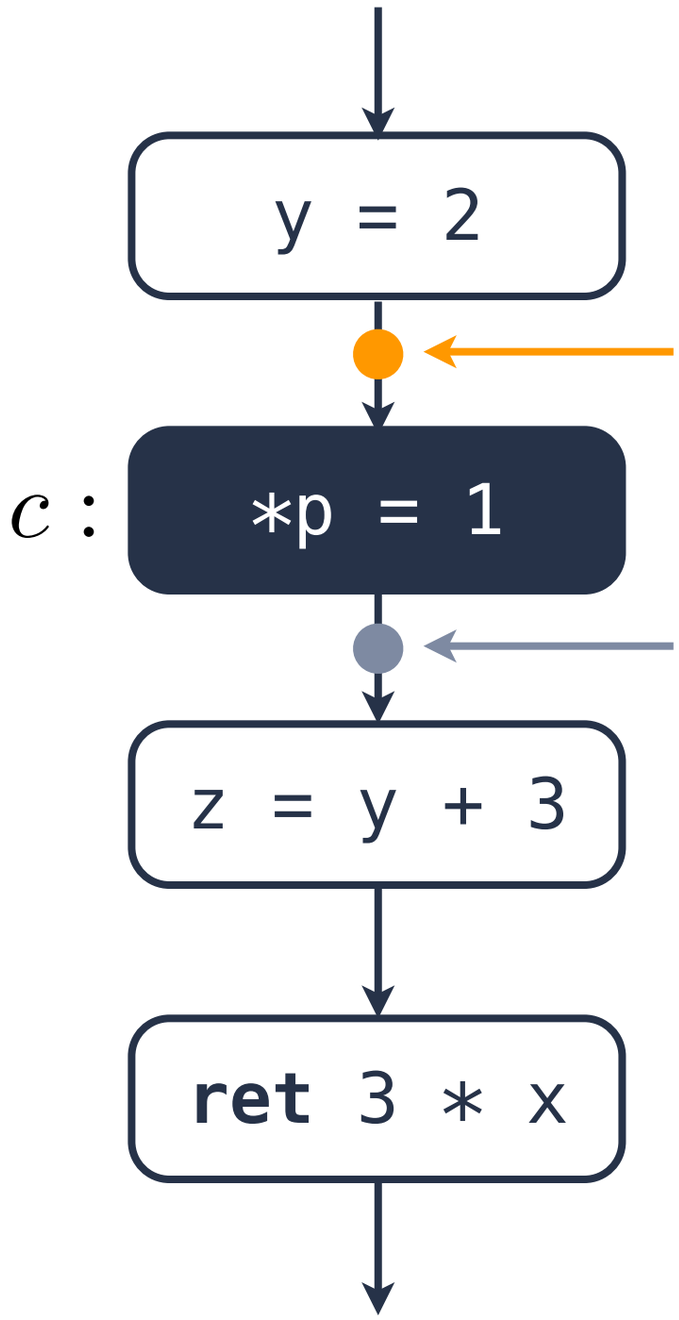
$\mathrm{U}(c)$ uses of primary analysis

$\mathrm{D}(c)$ definitions of primary analysis

Approximation of Definitions and Uses



Approximation of Definitions and Uses



$\mathsf{U}(c) \subseteq \hat{\mathsf{U}}(c)$

$\mathsf{D}(c) \subseteq \hat{\mathsf{D}}(c)$

<u>safety condition</u>

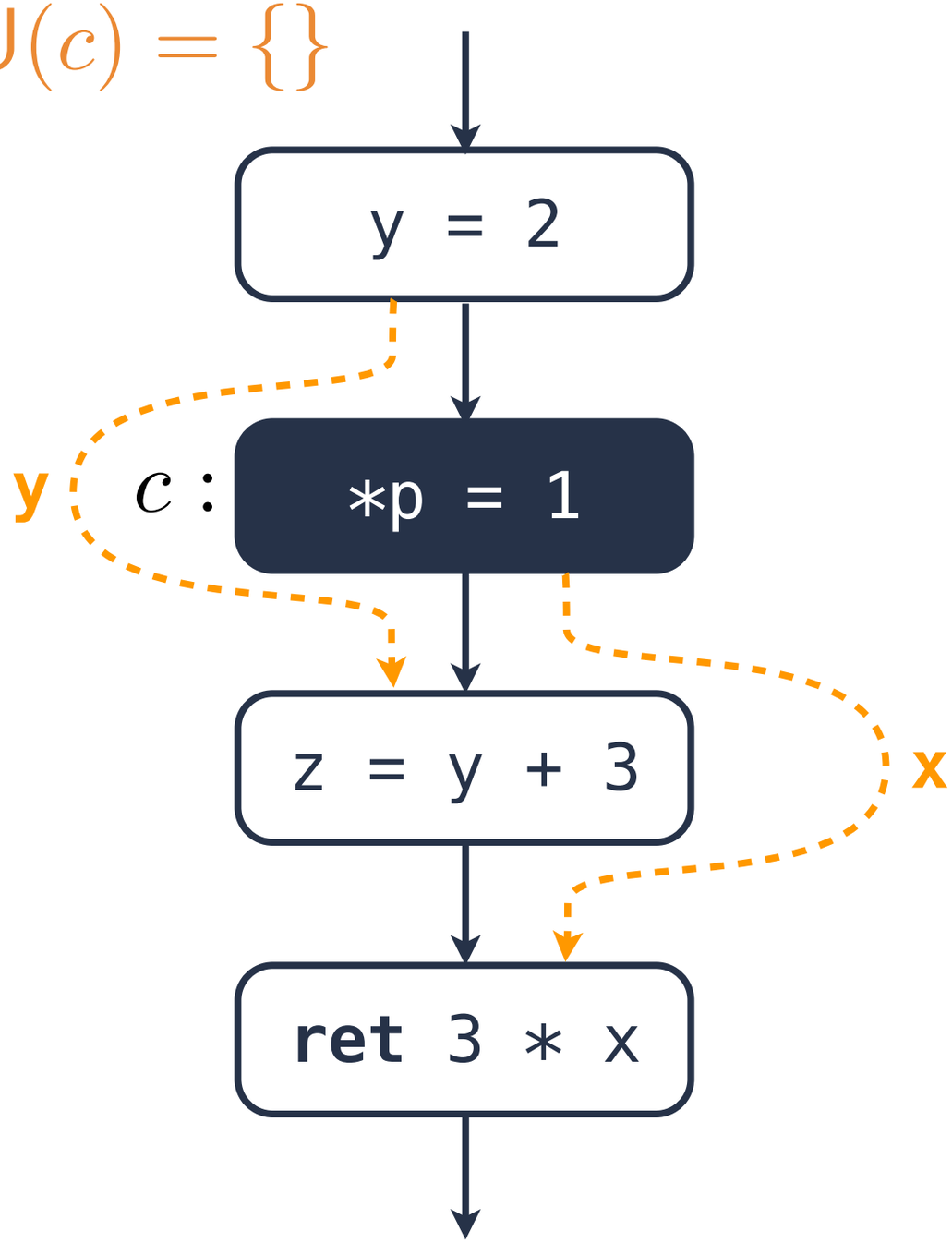
$\hat{\mathsf{D}}(c) - \mathsf{D}(c) \subseteq \hat{\mathsf{U}}(c)$

: all spurious definitions should be spurious uses

Example of Unsafe Approximation

$\mathsf{D}(c) = \{x\}$

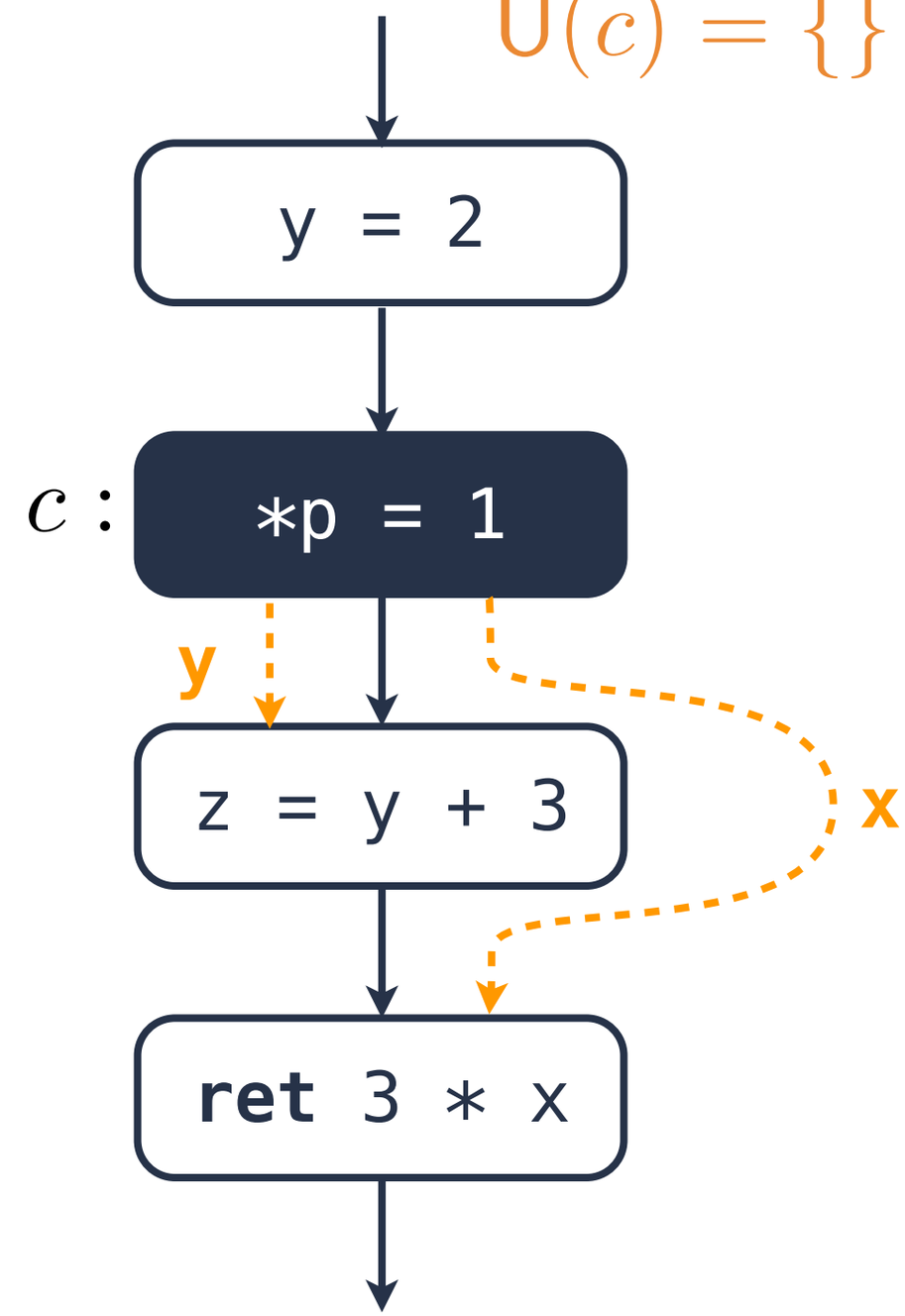
$\mathsf{U}(c) = \{\}$



most precise DFG
when p = {x}

$\hat{\mathsf{D}}(c) = \{x, y\}$

$\hat{\mathsf{U}}(c) = \{\}$

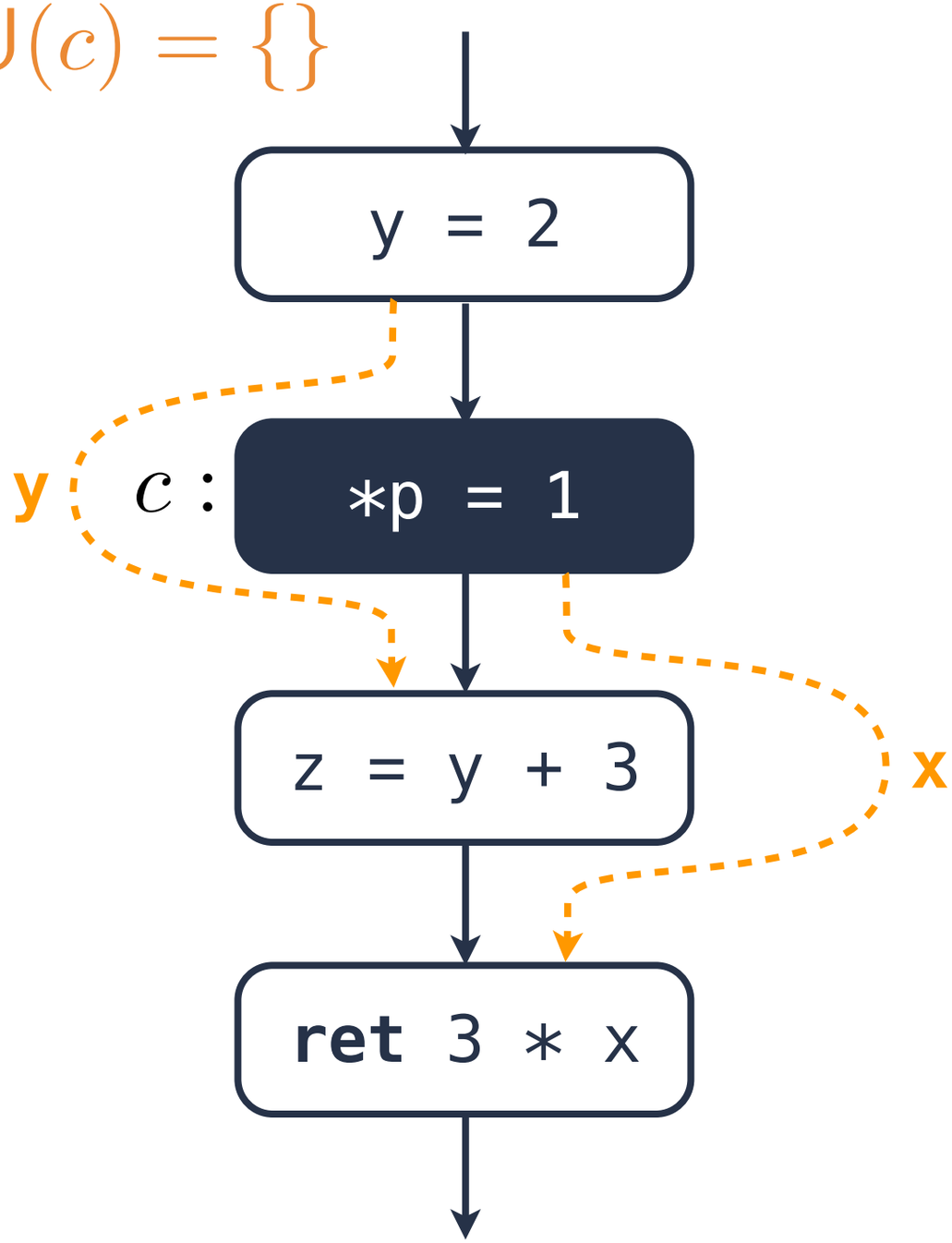


DFG
with approximated def/use

Example of Unsafe Approximation

$\mathsf{D}(c) = \{x\}$

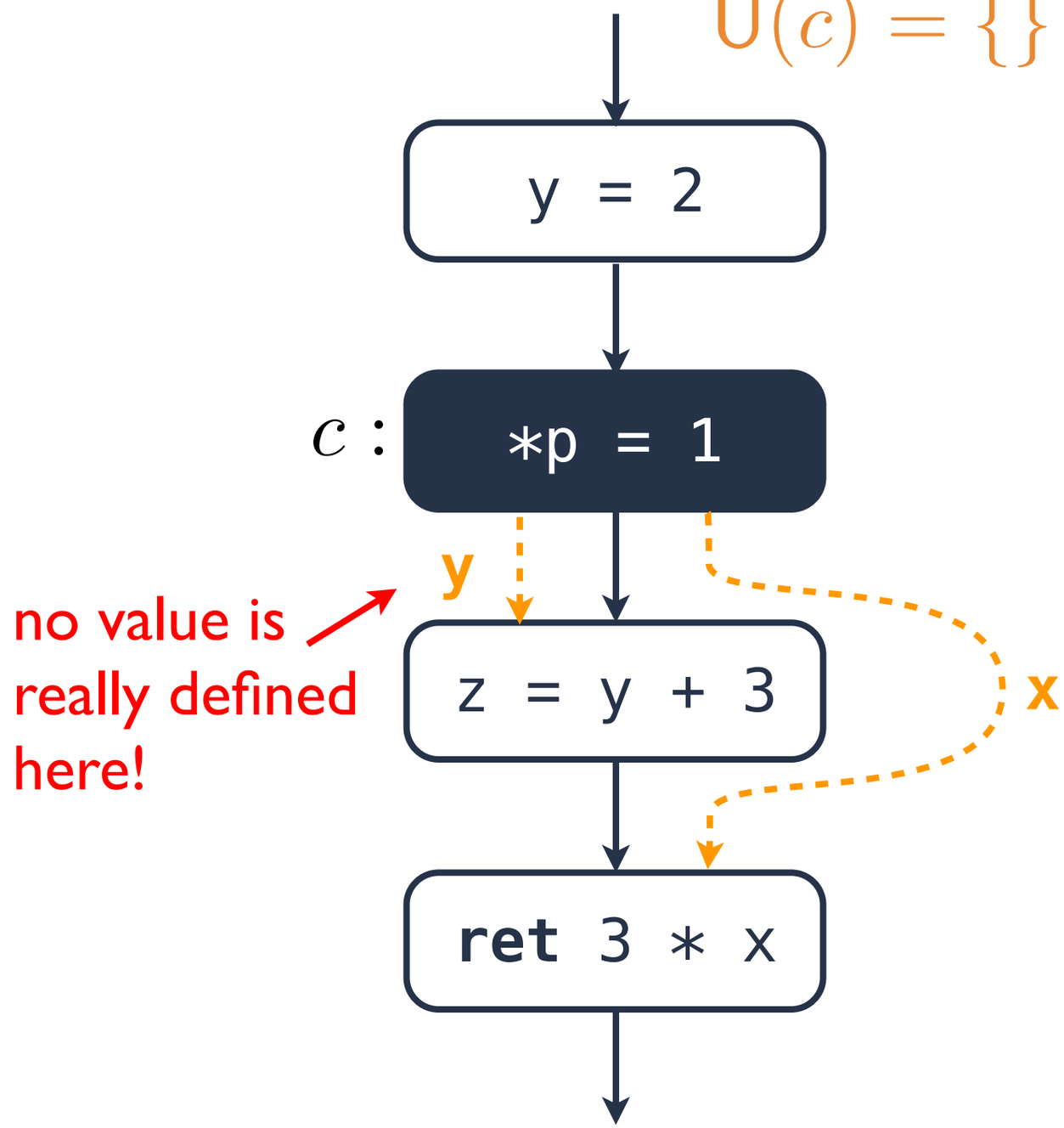
$\mathsf{U}(c) = \{\}$



most precise DFG
when p = {x}

$\hat{\mathsf{D}}(c) = \{x, y\}$

$\hat{\mathsf{U}}(c) = \{\}$

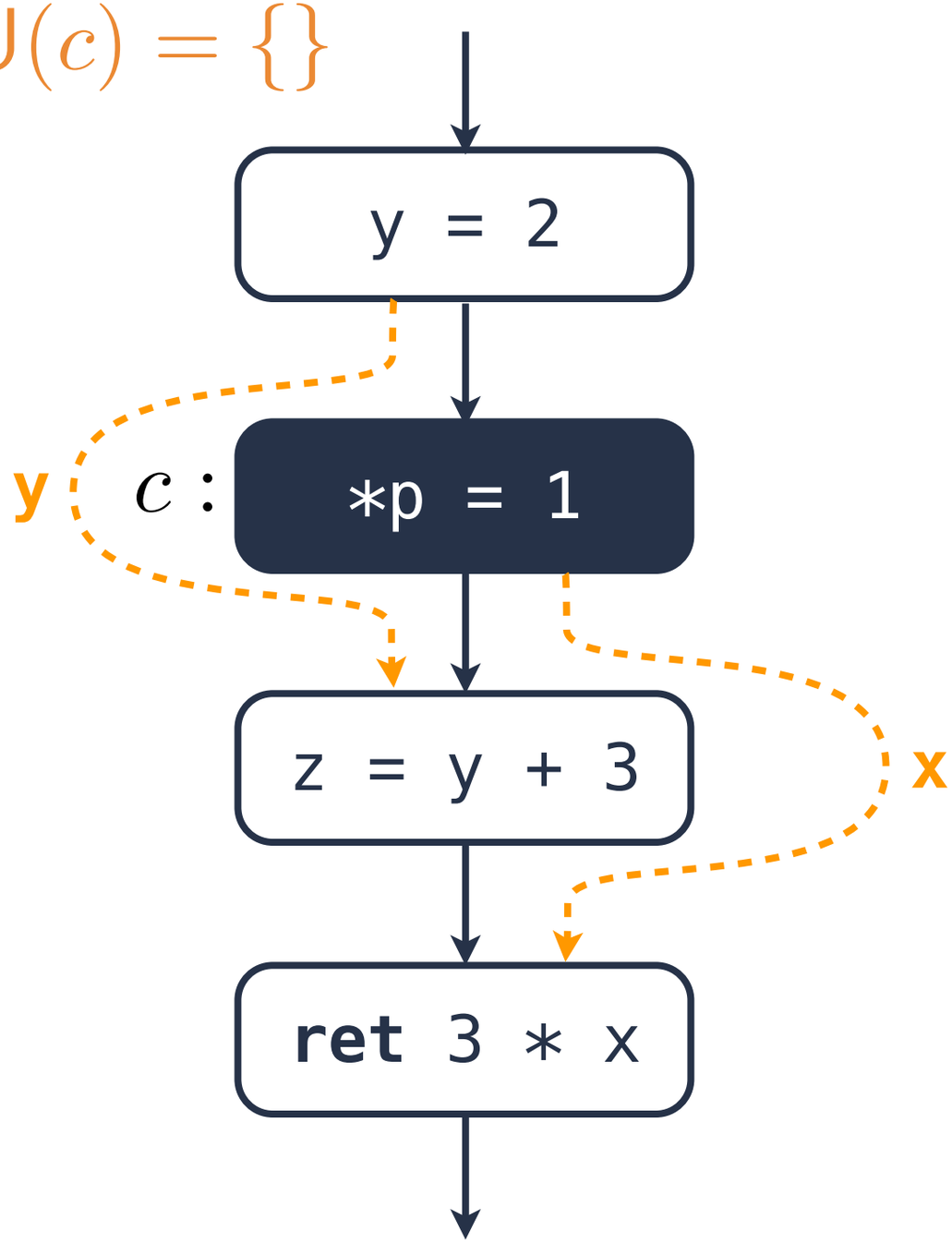


DFG
with approximated def/use

Example of Safe Approximation

$\mathsf{D}(c) = \{x\}$

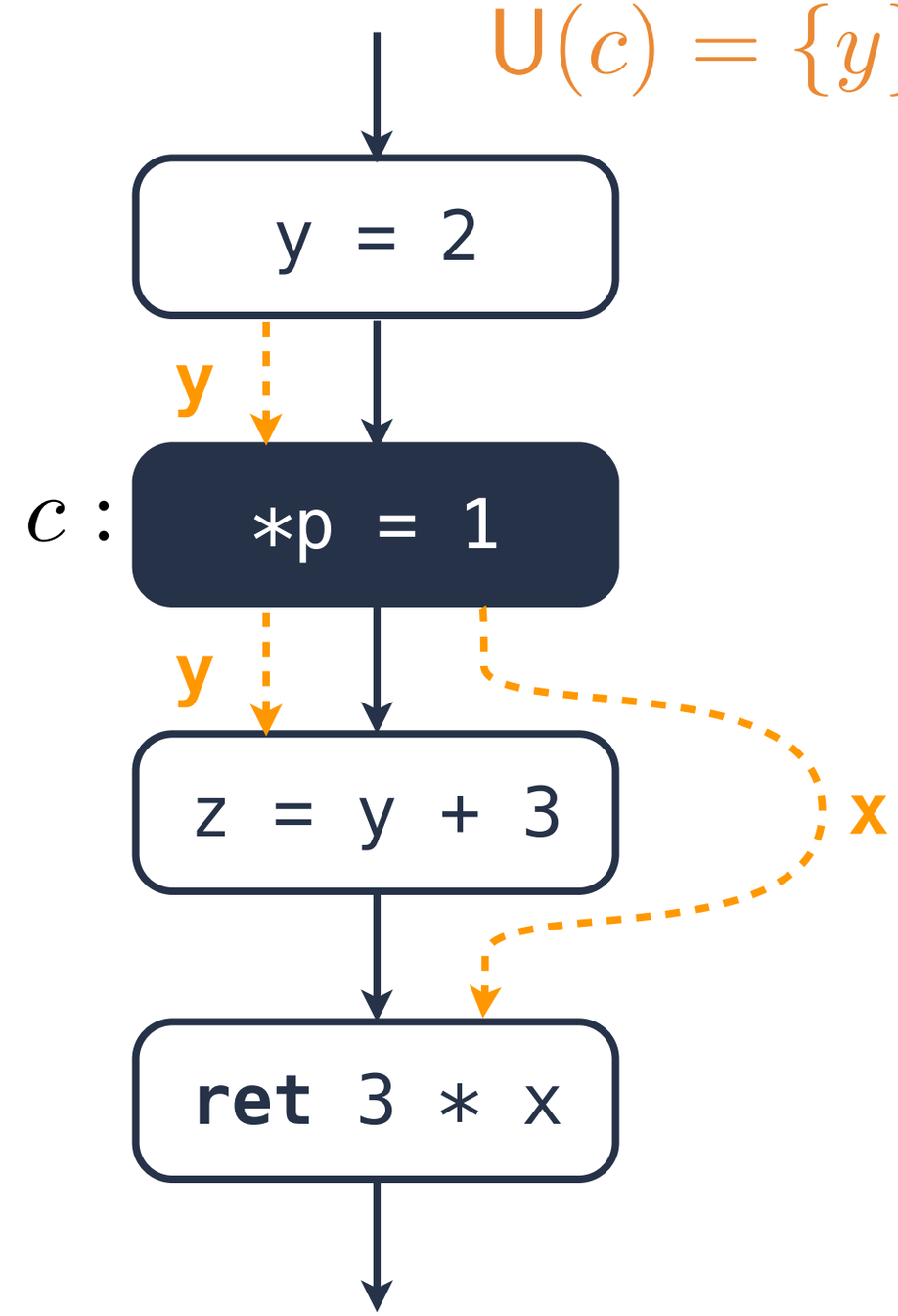
$\mathsf{U}(c) = \{\}$



most precise DFG
when p = {x}

$\hat{\mathsf{D}}(c) = \{x, y\}$

$\hat{\mathsf{U}}(c) = \{y\}$

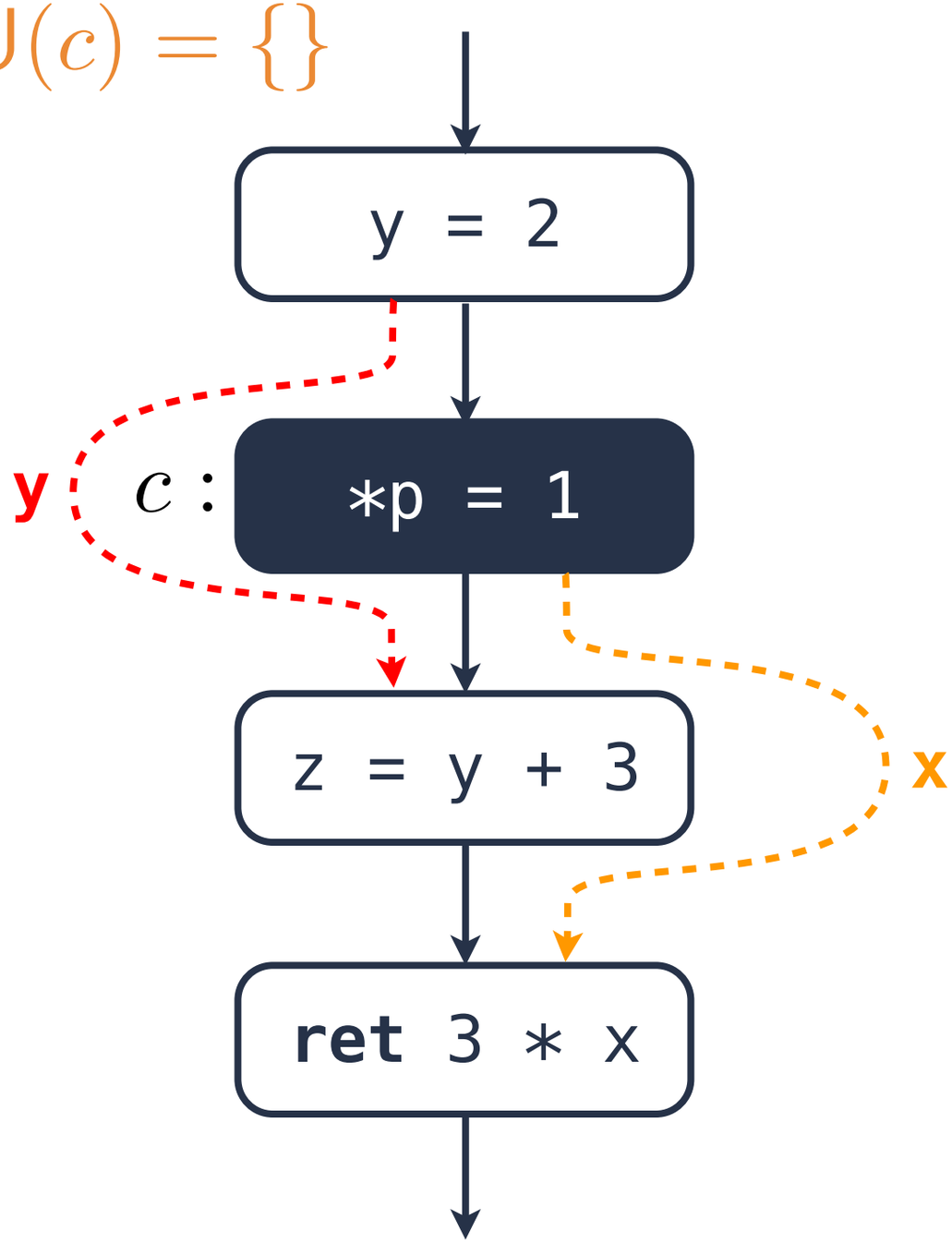


DFG
with approximated def/use

Example of Safe Approximation

$\mathsf{D}(c) = \{x\}$

$\mathsf{U}(c) = \{\}$

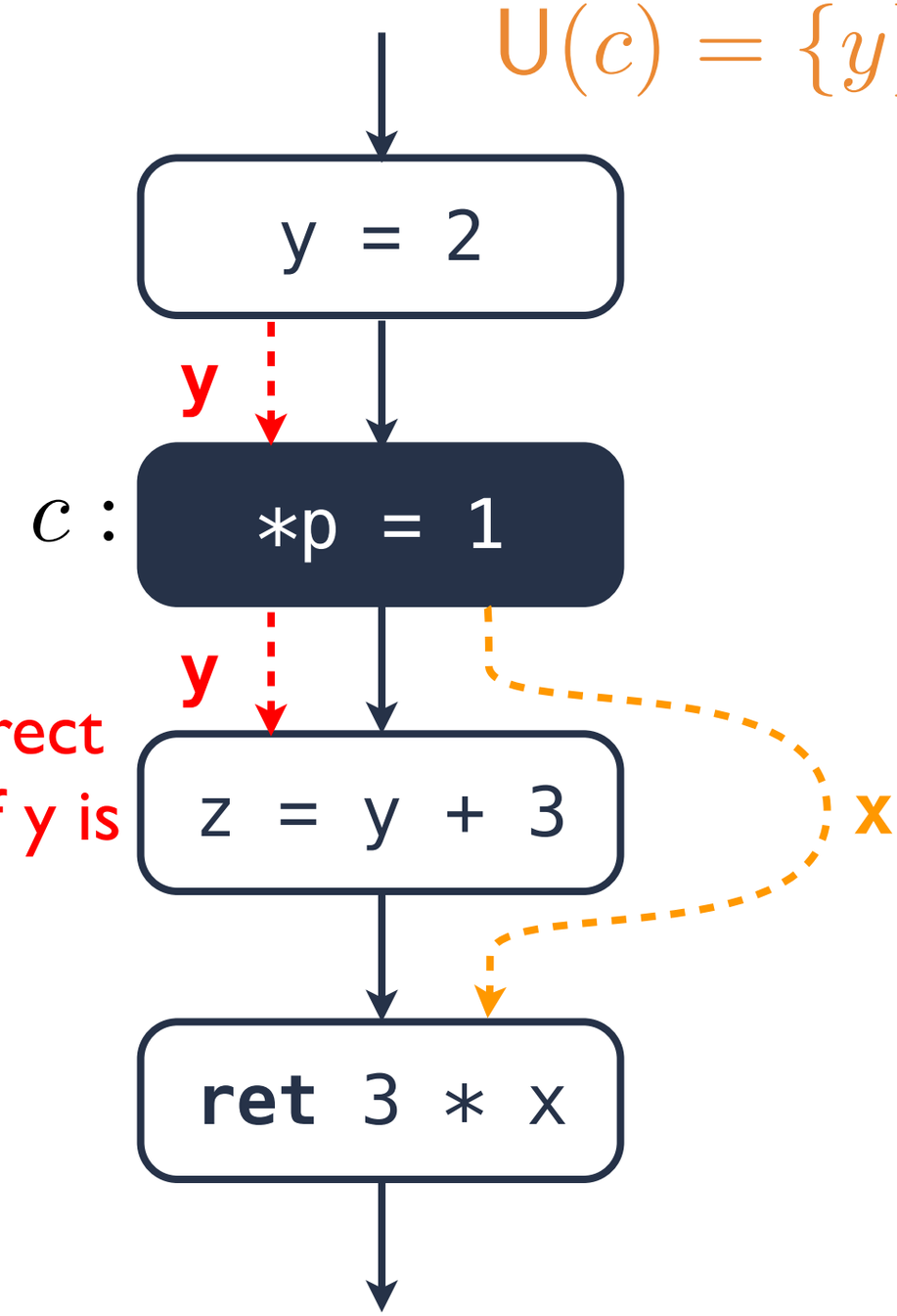


most precise DFG
when p = {x}

still the correct definition of y is propagated!

$\hat{\mathsf{D}}(c) = \{x, y\}$

$\hat{\mathsf{U}}(c) = \{y\}$



DFG
with approximated def/use

Sparse Analysis Preserves Precision

Original Static Analysis

$$\hat{F}(\hat{X}) = \lambda c.\hat{f}_c(\bigsqcup_{c' \hookrightarrow c} \hat{X}(c'))$$

Sparse Static Analysis

$$\hat{F}_s(\hat{X}) = \lambda c.\hat{f}_c(\bigsqcup_{c' \overset{x}{\rightsquigarrow} c} \hat{X}(c')|_x)$$

Sparse Analysis Preserves Precision

Original Static Analysis

$$\hat{F}(\hat{X}) = \lambda c.\hat{f}_c(\bigsqcup_{c' \hookrightarrow c} \hat{X}(c'))$$

CFG

Sparse Static Analysis

$$\hat{F}_s(\hat{X}) = \lambda c.\hat{f}_c(\bigsqcup_{c' \overset{x}{\rightsquigarrow} c} \hat{X}(c')|_x)$$

DFG

Sparse Analysis Preserves Precision

Original Static Analysis

$$\hat{F}(\hat{X}) = \lambda c.\hat{f}_c(\bigsqcup_{c' \hookrightarrow c} \hat{X}(c'))$$

CFG

Sparse Static Analysis

$$\hat{F}_s(\hat{X}) = \lambda c.\hat{f}_c(\bigsqcup_{c' \overset{x}{\rightsquigarrow} c} \hat{X}(c')|_x)$$

DFG

Theorem 1. $\mathbf{lfp}(\hat{F}) \simeq \mathbf{lfp}(\hat{F}_s)$

Sparse Analysis Preserves Precision

Original Static Analysis

$$\hat{F}(\hat{X}) = \lambda c.\hat{f}_c(\bigsqcup_{c' \hookrightarrow c} \hat{X}(c'))$$

Sparse Static Analysis

$$\hat{F}_s(\hat{X}) = \lambda c.\hat{f}_c(\bigsqcup_{c' \overset{x}{\rightsquigarrow} c} \hat{X}(c')|_x)$$

Theorem 1. $\mathbf{lfp}(\hat{F}) \simeq \mathbf{lfp}(\hat{F}_s)$

same precision up to actually used locations

Experiment Results

Benchmarks

Program	LOC	Functions	Statements	Blocks	maxSCC	AbsLocs
gzip-1.2.4a	7K	132	6,446	4,152	2	1,784
bc-1.06	13K	132	10,368	4,731	1	1,619
tar-1.13	20K	221	12,199	8,586	13	3,245
less-382	23K	382	23,367	9,207	46	3,658
make-3.76.1	27K	190	14,010	9,094	57	4,527
wget-1.9	35K	433	28,958	14,537	13	6,675
screen-4.0.2	45K	588	39,693	29,498	65	12,566
a2ps-4.14	64K	980	86,867	27,565	6	17,684
sendmail-8.13.6	130K	756	76,630	52,505	60	19,135
nethack-3.3.0	211K	2,207	237,427	157,645	997	54,989
vim60	227K	2,770	150,950	107,629	1,668	40,979
emacs-22.1	399K	3,388	204,865	161,118	1,554	66,413
python-2.5.1	435K	2,996	241,511	99,014	723	51,859
linux-3.0	710K	13,856	345,407	300,203	493	139,667
gimp-2.6	959K	11,728	1,482,230	286,588	2	190,806
ghostscript-9.00	1,363K	12,993	2,891,500	342,293	39	201,161

maxSCC: size of the largest recursive call cycle

Experiment Results

Buffer Overrun Detection with Intervals

Programs	$\text{Interval}_{\text{vanilla}}$	$\text{Interval}_{\text{base}}$	$\text{Spd}\uparrow_1$	$\text{Interval}_{\text{sparse}}$	$\text{Spd}\uparrow_2$
gzip-1.2.4a	772	14	55 x	3	5 x
bc-1.06	1,270	96	13 x	7	14 x
tar-1.13	12,947	338	38 x	8	42 x
less-382	9,561	1,211	8 x	33	37 x
make-3.76.1	24,240	1,893	13 x	21	90 x
wget-1.9	44,092	1,214	36 x	11	110 x
screen-4.0.2	∞	31,324	N/A	767	41 x
a2ps-4.14	∞	3,200	N/A	40	80 x
sendmail-8.13.6	∞	∞	N/A	744	N/A
nethack-3.3.0	∞	∞	N/A	16,373	N/A
vim60	∞	∞	N/A	23,798	N/A
emacs-22.1	∞	∞	N/A	37,830	N/A
python-2.5.1	∞	∞	N/A	11,039	N/A
linux-3.0	∞	∞	N/A	33,618	N/A
gimp-2.6	∞	∞	N/A	3,874	N/A
ghostscript-9.00	∞	∞	N/A	14,814	N/A

$\text{Interval}_{\text{vanilla}}$: worklist-based analyzer with no optimization
$\text{Interval}_{\text{base}}$: $\text{Interval}_{\text{vanilla}}$ + localization optimization (VMCAI'11)
$\text{Interval}_{\text{sparse}}$: sparse version of $\text{Interval}_{\text{vanilla}}$

Experiment Results

Buffer Overrun Detection with Intervals

Programs	$\text{Interval}_{\text{vanilla}}$	$\text{Interval}_{\text{base}}$	$\text{Spd}\uparrow_1$	$\text{Interval}_{\text{sparse}}$	$\text{Spd}\uparrow_2$
gzip-1.2.4a	772	14	55 x	3	5 x
bc-1.06	1,270	96	13 x	7	14 x
tar-1.13	12,947	338	38 x	8	42 x
less-382	9,561	1,211	8 x	33	37 x
make-3.76.1	24,240	1,893	13 x	21	90 x
wget-1.9	44,092	1,214	36 x	11	110 x
screen-4.0.2	∞	31,324	N/A	767	41 x
a2ps-4.14	∞	3,200	N/A	40	80 x
sendmail-8.13.6	∞	∞	N/A	744	N/A
nethack-3.3.0	∞	∞	N/A	16,373	N/A
vim60	∞	∞	N/A	23,798	N/A
emacs-22.1	∞	∞	N/A	37,830	N/A
python-2.5.1	∞	∞	N/A	11,039	N/A
linux-3.0	∞	∞	N/A	33,618	N/A
gimp-2.6	∞	∞	N/A	3,874	N/A
ghostscript-9.00	∞	∞	N/A	14,814	N/A

code size is not the only factor for the running time

Experiment Results

Buffer Overrun Detection with Octagons

Programs	$\text{Octagon}_{\text{vanilla}}$	$\text{Octagon}_{\text{base}}$	$\text{Spd}\uparrow_1$	$\text{Octagon}_{\text{sparse}}$	$\text{Spd}\uparrow_2$
gzip-1.2.4a	2,078	273	8 x	21	13 x
bc-1.06	9,536	1,065	9 x	55	19 x
tar-1.13	∞	9,566	N/A	188	51 x
less-382	∞	16,121	N/A	432	37 x
make-3.76.1	∞	17,724	N/A	331	53 x
wget-1.9	∞	15,998	N/A	288	56 x
screen-4.0.2	∞	∞	N/A	16,433	N/A
a2ps-4.14	∞	∞	N/A	8,566	N/A
sendmail-8.13.6	∞	∞	N/A	64,808	N/A

$\text{Octagon}_{\text{vanilla}}$: worklist-based analyzer with no optimization
$\text{Octagon}_{\text{base}}$: $\text{Octagon}_{\text{vanilla}}$ + localization optimization
$\text{Octagon}_{\text{sparse}}$: sparse version of $\text{Octagon}_{\text{vanilla}}$

Conclusion

Sparse analysis makes sound static analysis scalable

Questions?

for technical details:
http://ropas.snu.ac.kr/~wclee/papers/pldi12b.pdf