

41st IEEE/ACM International Conference on Automated Software Engineering

IncSFS

Incremental Full-Sparse Flow-Sensitive Pointer Analysis for C/C++

Kunlin Liu, Zhenbang Chen, Piyi Zu, Yide Du, Ji Wang



National University of Defense Technology

Flow-Sensitive Pointer Analysis

- **Pointer analysis** statically computes the memory address which pointer variables point to at runtime.

```
1  p = alloca;     \\ A
2  a = alloca;     \\ B
3  b = alloca;     \\ C
4  *p = a;
5  *p = b;
6  t = a;
7  y = *p;
```

What can y point to?

Flow-insensitive

y: {B, C}

Ignore statement order

Flow-sensitive

y: {C}

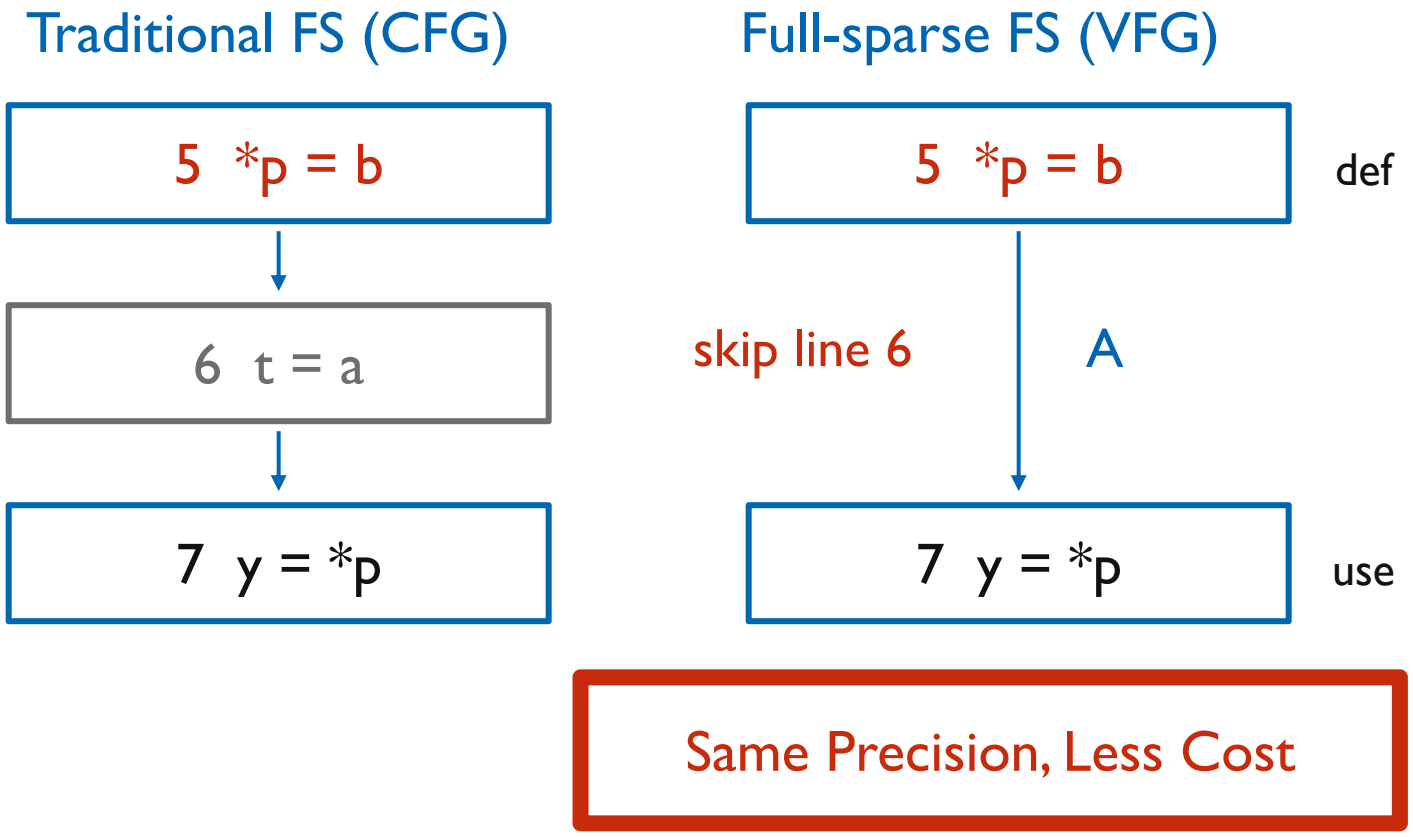
Respect statement order

Higher precision

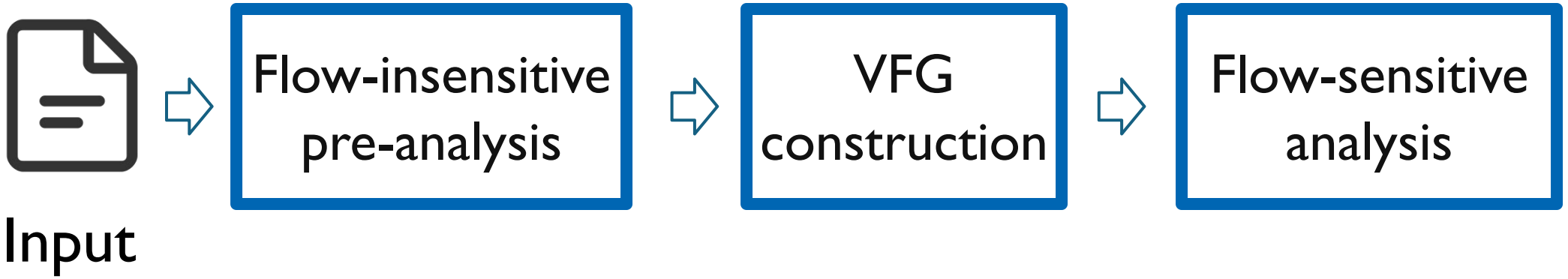
Full-Sparse Flow-Sensitive Analysis

- Full sparse flow sensitivity propagates points-to result along VFG instead of CFG
  - Value flow graph (VFG) describes def use relations among all variables.

```
1   p = alloca;     \\ A
2   a = alloca;     \\ B
3   b = alloca;     \\ C
4   *p = a;
5   *p = b;
6   t = a;
7   y = *p;
```

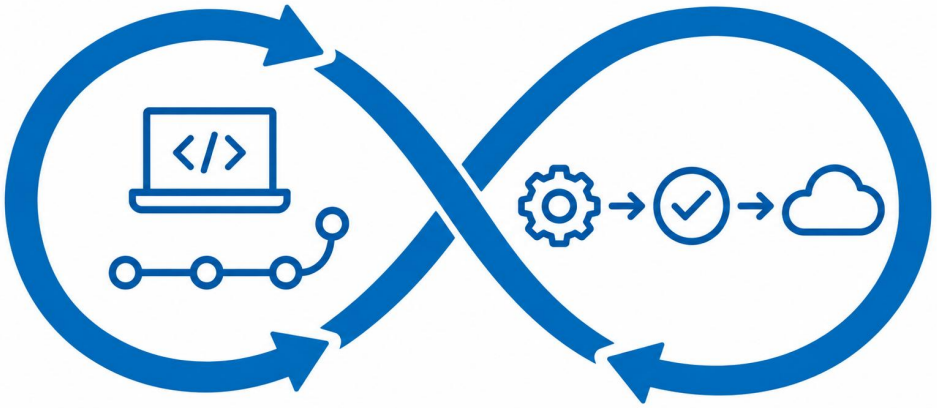


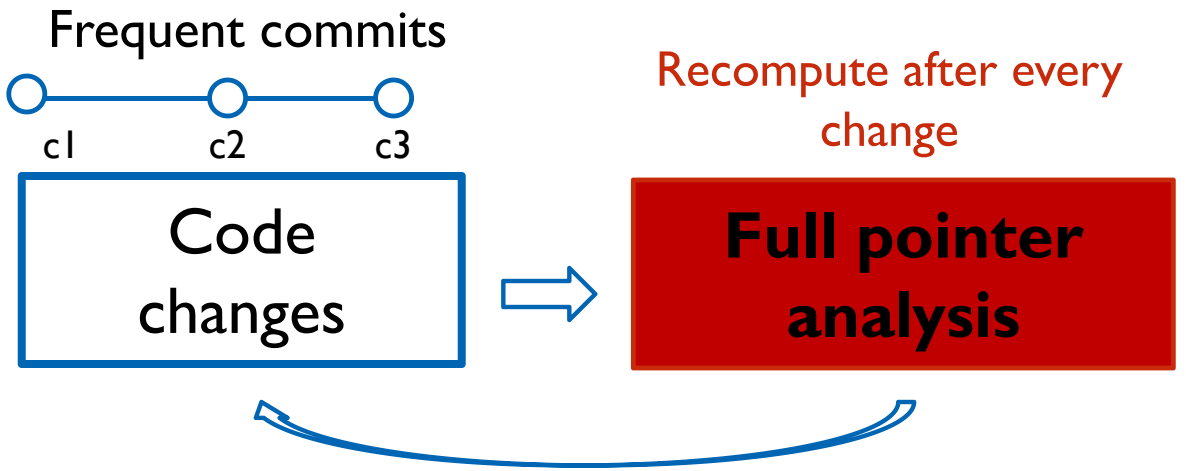
Full-Sparse Flow-Sensitive Analysis



Problem

CI/CD





Frequent code changes make repeated full analysis a major scalability bottleneck in CI/CD, delaying feedback and slowing development.

Existing Work

- Reset–Recompute [ICSE'99]

  Reset affected results and recompute

- Restart–Iteration [VMCAI'23]

  Old facts in cycles can reduce precision

- IPA [TOPLAS'19, TOSEM'25]

  Propagate only changes; higher efficiency

Low Efficiency

Existing Work

- Reset–Recompute [ICSE'99]

  Reset affected results and recompute

- Restart–Iteration [VMCAI'23]

  Old facts in cycles can reduce precision

- IPA [TOPLAS'19, TOSEM'25]

  Propagate only changes; higher efficiency

Low Efficiency

Precision Loss

Existing Work

- Reset–Recompute [ICSE'99]

  Reset affected results and recompute

- Restart–Iteration [VMCAI'23]

  Old facts in cycles can reduce precision

- IPA [TOPLAS'19,TOSEM'25]

  Propagate only changes; higher efficiency

Low Efficiency

Precision Loss

Limited to Flow-insensitive

Existing Work

- Reset–Recompute [1]
  Reset affected results and recompute
- Restart–Iteration [2]
  Reuse analysis results and restart propagation
- IPA [3, 4]
  Propagate only changes, higher efficiency

Low Efficiency

Precision Loss

Limited to Flow-insensitive

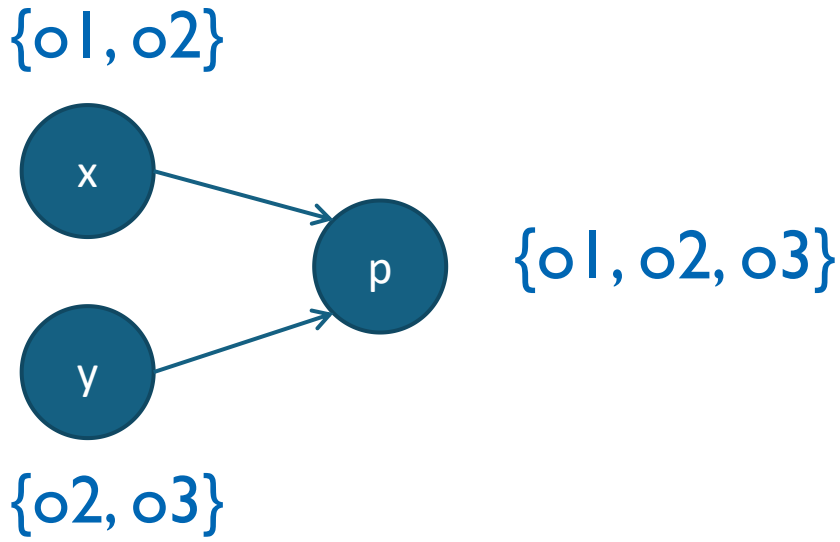
No prior work on incremental full-sparse flow-sensitive pointer analysis.

References

[1] Yur et al. Incremental flow- and context-sensitive pointer aliasing analysis. ICSE 1999.
[2] Van der Plas et al. Result Invalidation for Incremental Modular Analyses. VMCAI 2023.
[3] Liu et al. Rethinking Incremental and Parallel Pointer Analysis. TOPLAS 2019.
[4] Wang et al. SILVA: A Scalable Incremental Layered Sparse Value-Flow Analysis. TOSEM 2025.

Incremental Pointer Analysis (IPA)

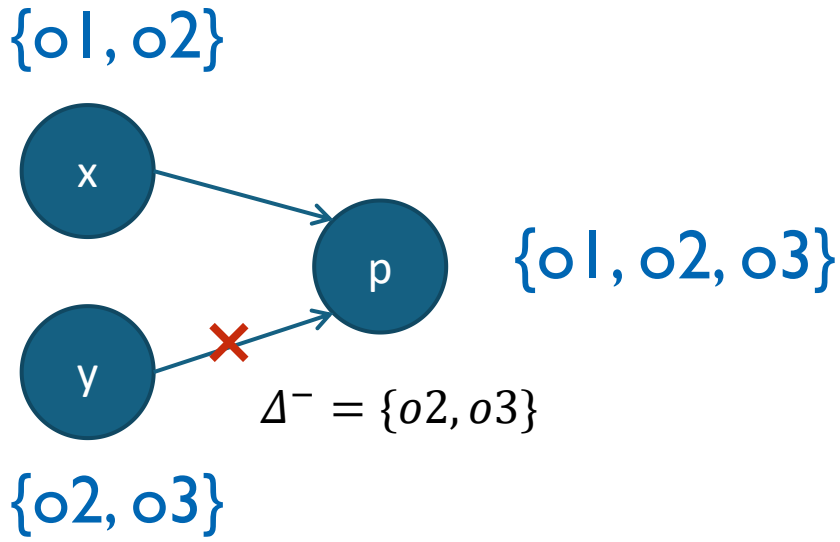
- Propagate changes only target to pointer analysis , higher efficiency



$$q \rightarrow p \Rightarrow pts(q) \subseteq pts(p)$$

Incremental Pointer Analysis (IPA)

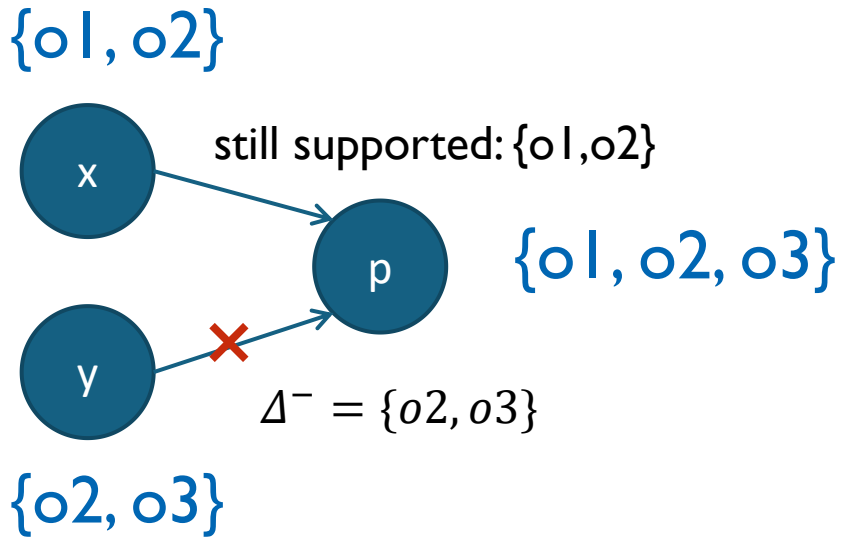
- Propagate changes only target to pointer analysis , higher efficiency



$$q \rightarrow p \Rightarrow pts(q) \subseteq pts(p)$$

Incremental Pointer Analysis (IPA)

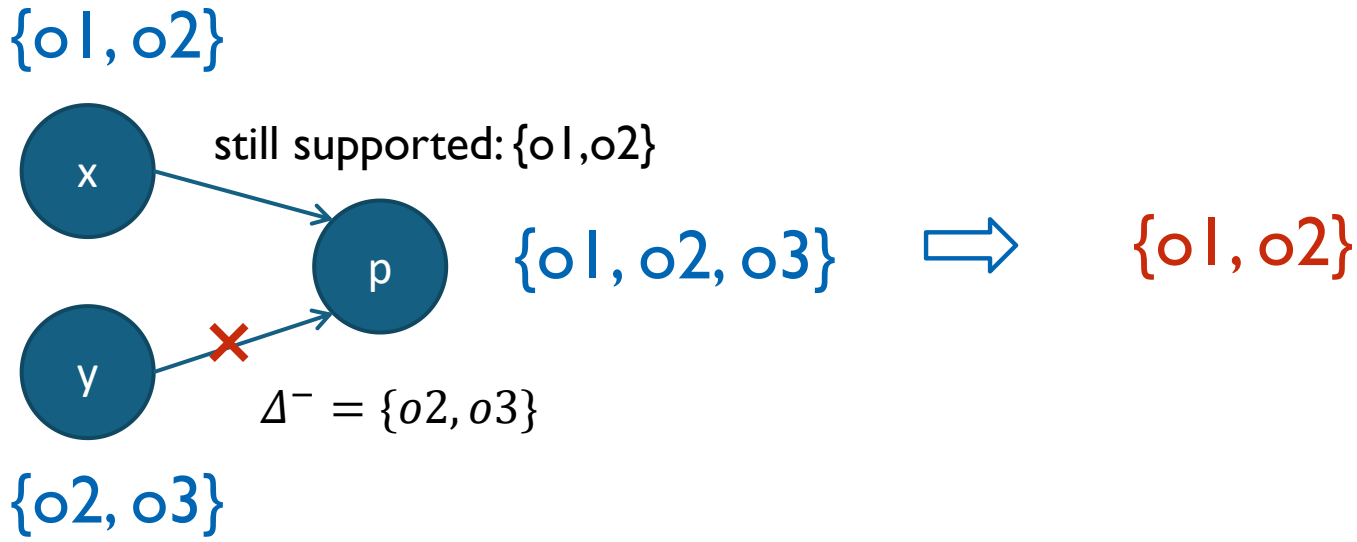
- Propagate changes only target to pointer analysis , higher efficiency



$$q \rightarrow p \Rightarrow pts(q) \subseteq pts(p)$$

Incremental Pointer Analysis (IPA)

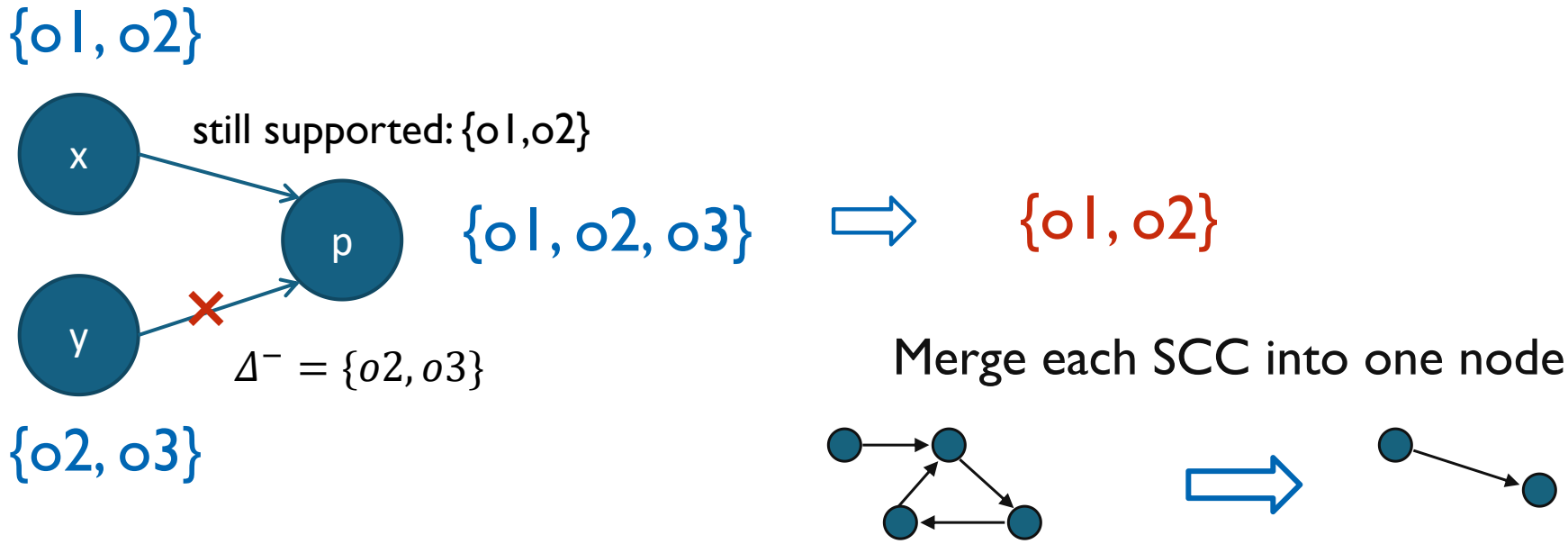
- Propagate changes only target to pointer analysis , higher efficiency



$$q \rightarrow p \Rightarrow pts(q) \subseteq pts(p)$$

Incremental Pointer Analysis (IPA)

- Propagate changes only target to pointer analysis , higher efficiency

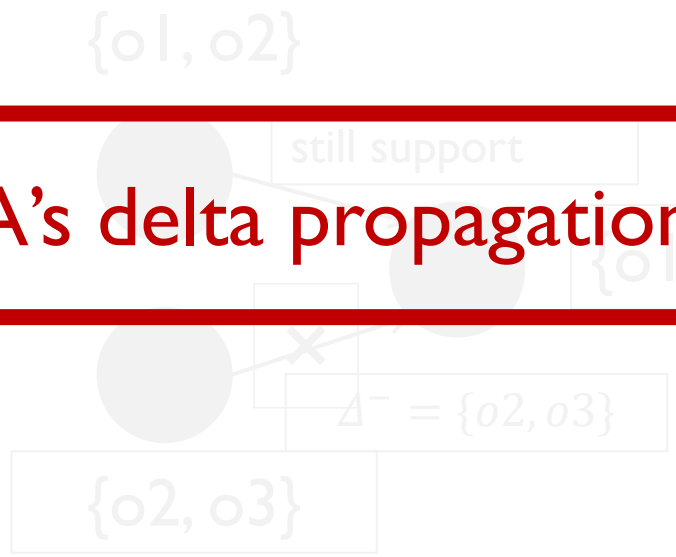


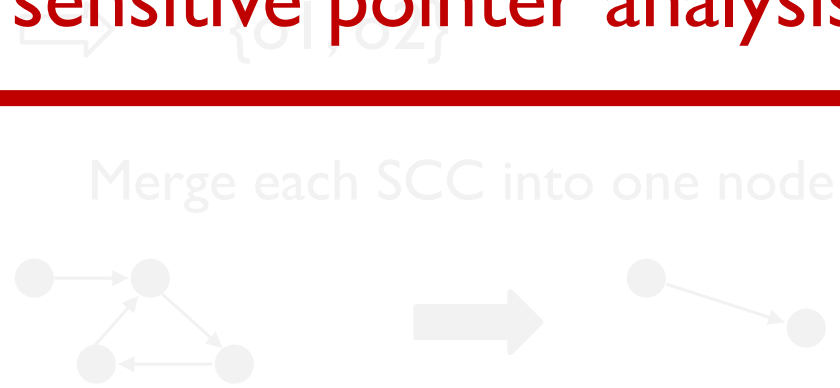
$$q \rightarrow p \Rightarrow pts(q) \subseteq pts(p)$$

Key Idea

- Propagate changes only target to pointer analysis , higher efficiency

Apply IPA's delta propagation to flow sensitive pointer analysis

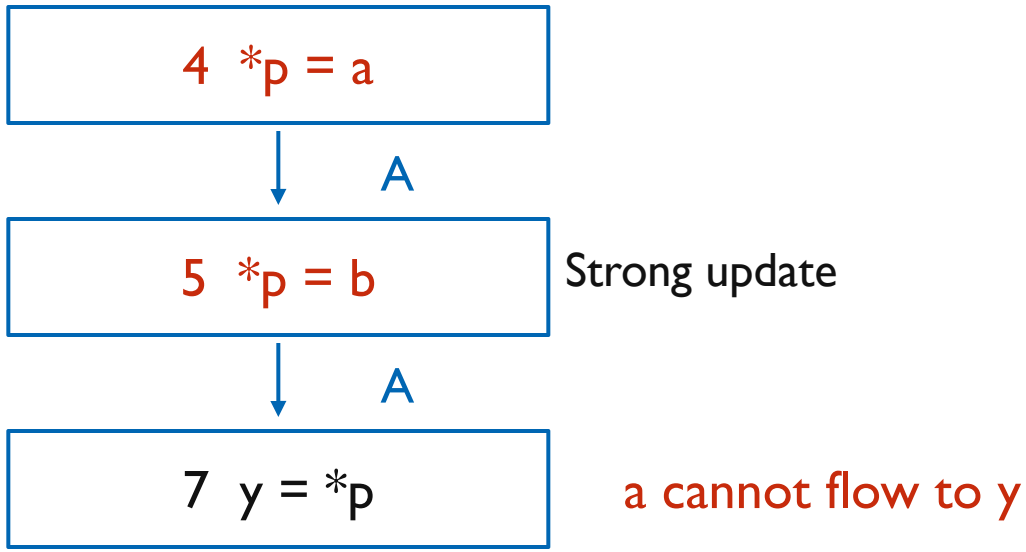




$$q \rightarrow p \Rightarrow pts(q) \subseteq pts(p)$$

Challenge 1: Encoding Flow Sensitivity

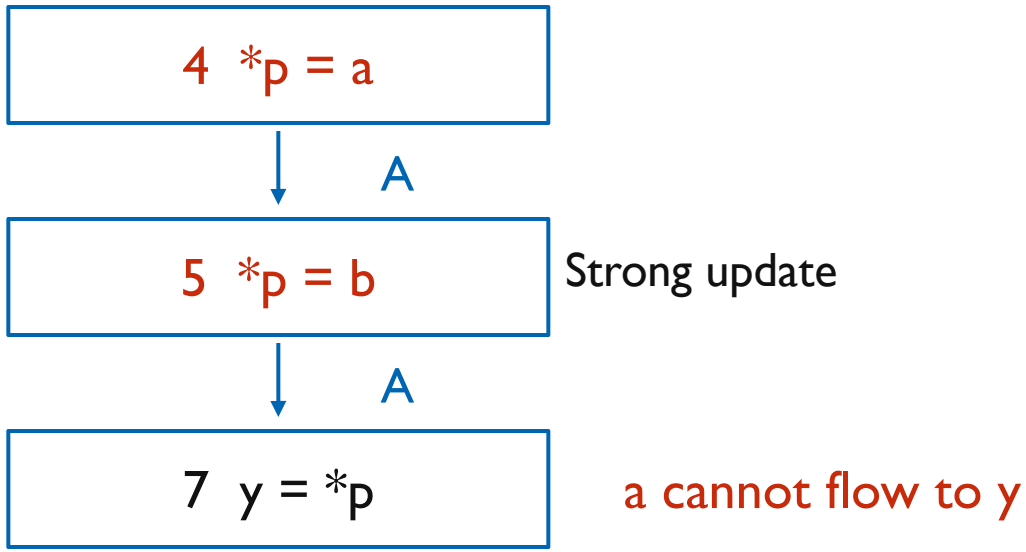
- Strong updates kill incoming propagation.



- A path in the VFG does not necessarily imply that points-to information can propagate along it
- Different objects have different propagation paths.

Challenge 1: Encoding Flow Sensitivity

- Strong updates kill incoming propagation.

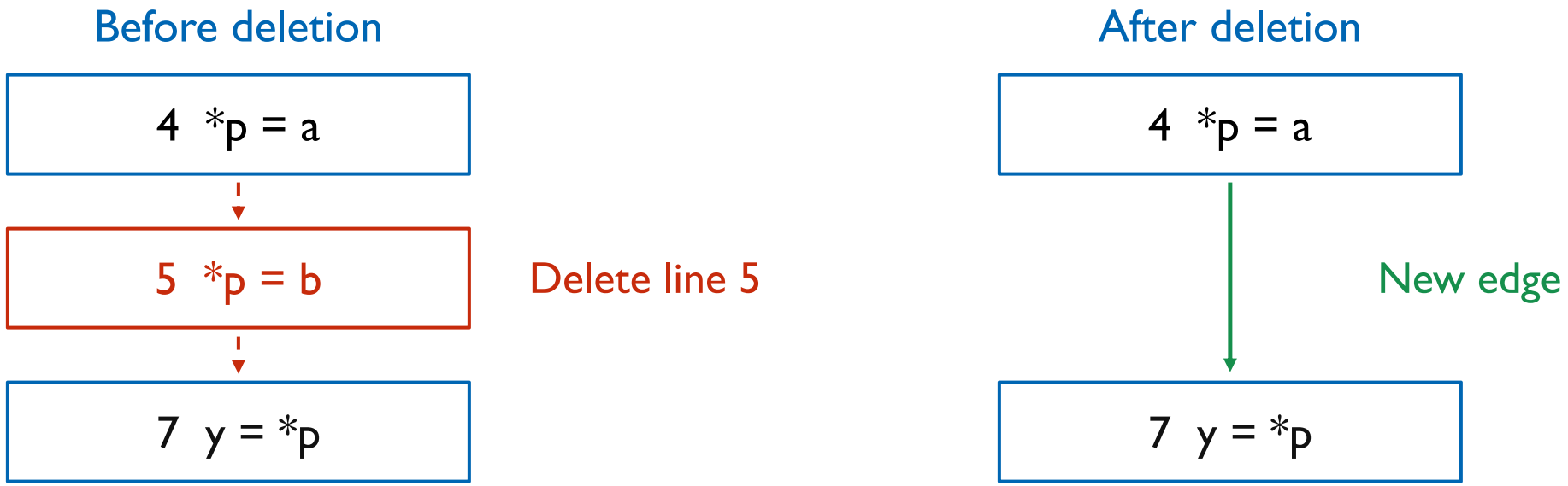


- A path in the VFG does not necessarily imply that points-to information can propagate along it
- Different objects have different propagation paths.

Need a graph that explicitly encodes flow-sensitive propagation

Challenge 2: Mixed Additions and Deletions

- Existing IPA processes code deletions and insertions separately.
  - Flow-insensitive analysis treats statements as an unordered set.
  - Code deletion only removes elements from this set, while insertion only adds elements.
- Flow-sensitive deletion can both remove and add propagation edges

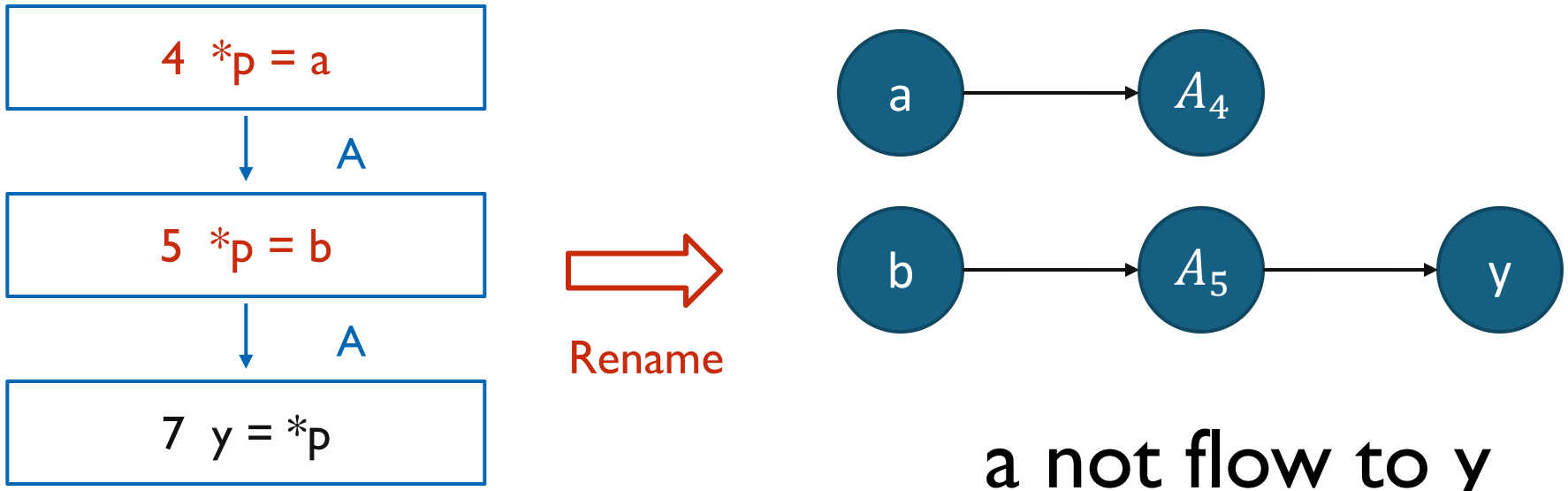


Solution: Variable Renaming

- Rename variables to transform VFG into a constraint graph.
- Update propagation relations over the renamed variables.

Solution: Variable Renaming

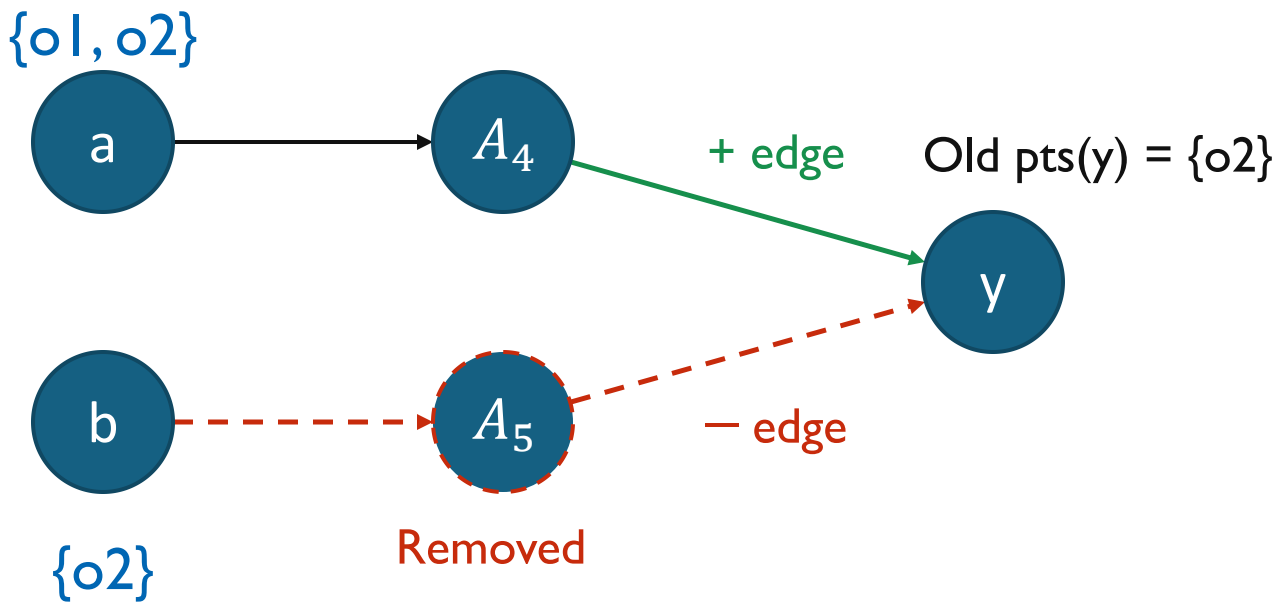
- Rename variables to transform VFG into a constraint graph.
- Update propagation relations over the renamed variables.



a not flow to y

Joint Delta Propagation: Example

- Use existing and new support to preserve unchanged facts.

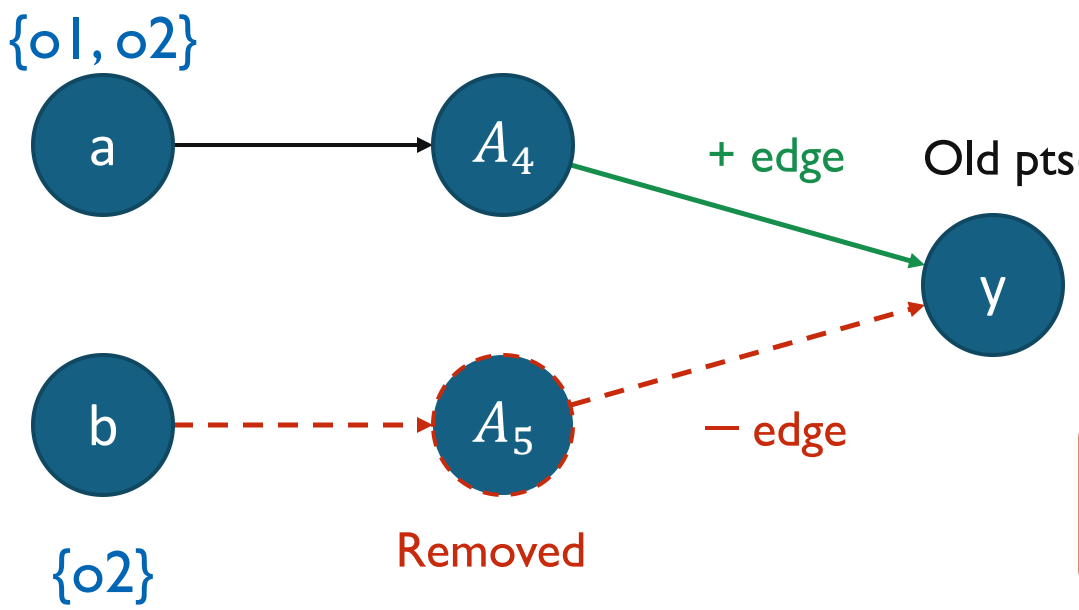


- Seperate delta propagation at y

$$1.\ \Delta_y^- = \{o2\}$$
$$2.\ \Delta_y^+ = \{o1, o2\}$$

Joint Delta Propagation: Example

- Use existing and new support to preserve unchanged facts.



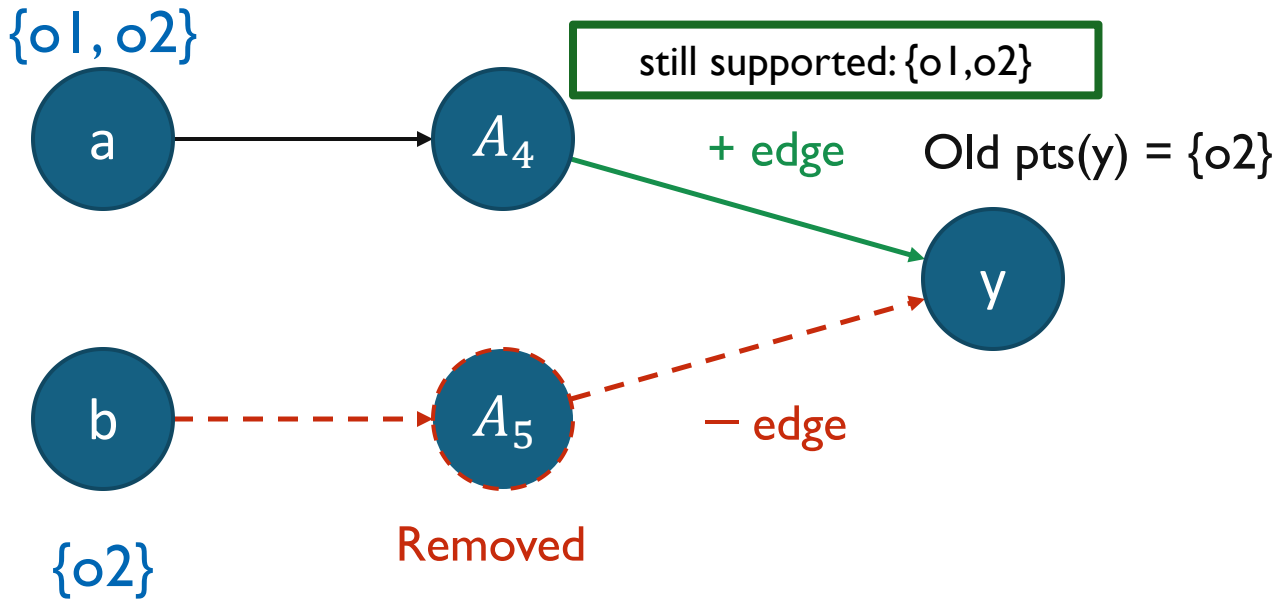
- Seperate delta propagation at y

$$1.\ \Delta_y^- = \{o2\}$$
$$2.\ \Delta_y^+ = \{o1, o2\}$$

Two-phase IPA removes {o2} first,
then adds {o1, o2}.

Joint Delta Propagation: Example

- Use existing and new support to preserve unchanged facts.



- Seperate delta propagation at y

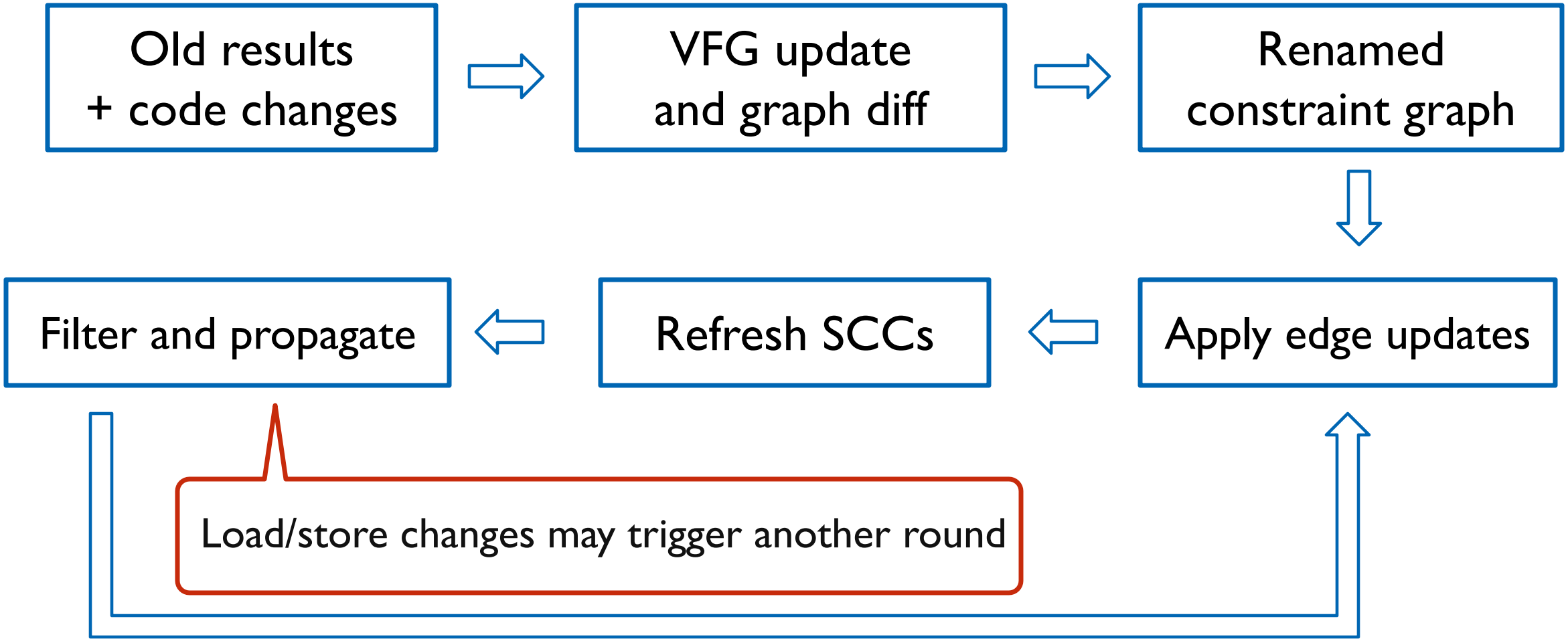
$$1.\ \Delta_y^- = \{o2\}$$
$$2.\ \Delta_y^+ = \{o1, o2\}$$

- Joint delta propagation at y

$\Delta_y^+ = \{o1\}$ $\Delta_y^- = \emptyset$

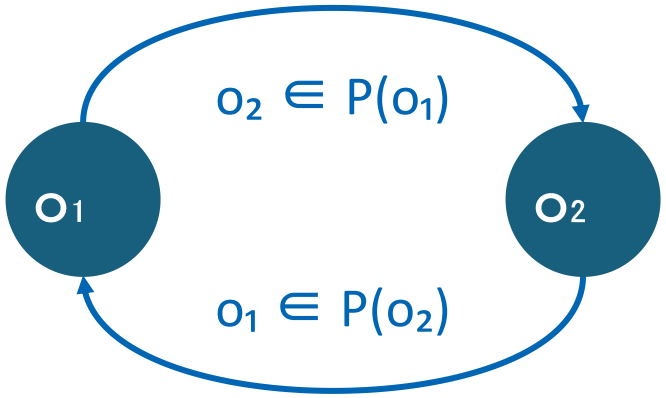
Our joint delta propagation

IncSFS Framework



Theoretical Guarantees

- **Object points-to cycle**



Under no object points-to cycle condition, Termination and Consistency are guaranteed

Evaluation

- **Research Questions**

  - RQ1: How much faster is IncSFS than full SFS and Reset–Recompute?
  - RQ2: How much faster is joint propagation than two-phase IPA?
  - RQ3: How does runtime grow with the commit interval?
  - RQ4: Do incremental points-to sets match full SFS?

Evaluation

■ Implementation and Experimental Setup

1. **Implementation:** SVF 2.7 + LLVM 14.0.1

2. **Benchmarks:** 6 programs

janet	zstd	tmux	astyle	nginx	sqlite
85K LOC	96K LOC	74K LOC	100K LOC	171K LOC	525K LOC

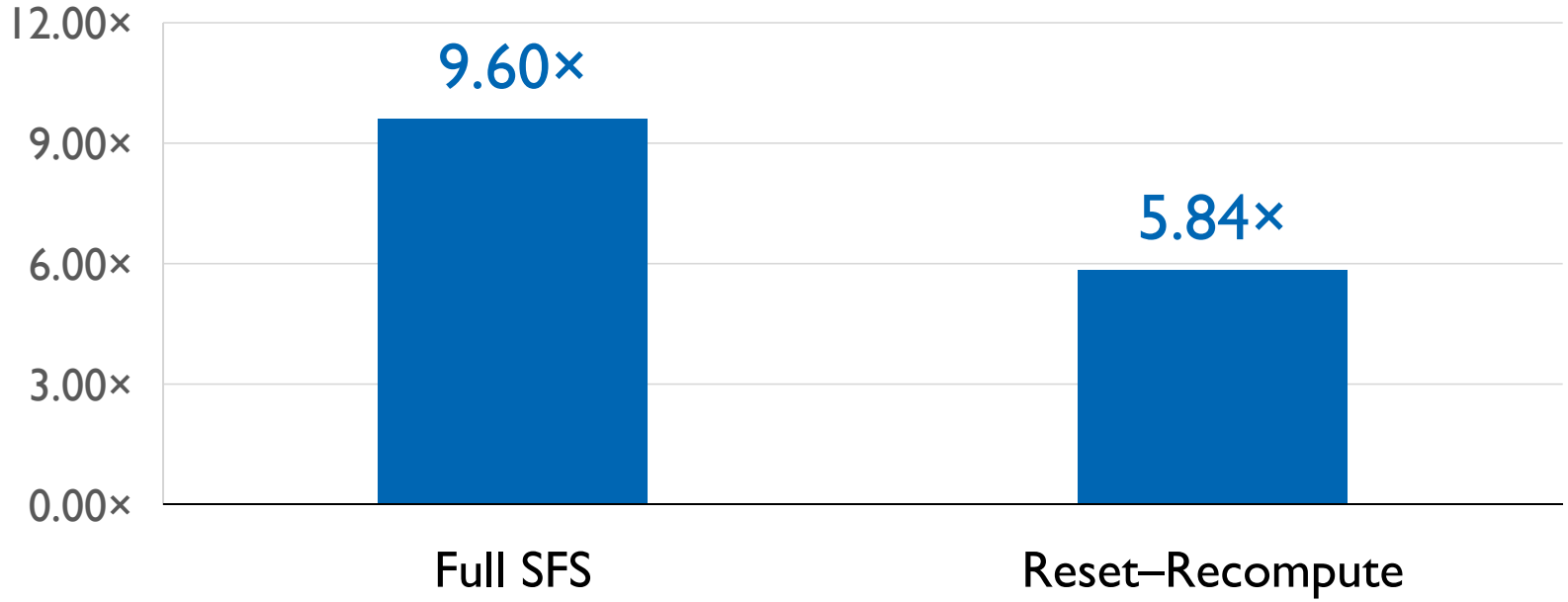
3. **Version pairs:** 4 commit intervals (1, 10, 20, 30)
   6 programs × 4 intervals × 10 pairs = 240 pairs

4. **Baselines:** Full SFS, Reset–Recompute

Evaluation

■ **RQ1: Performance Gain**

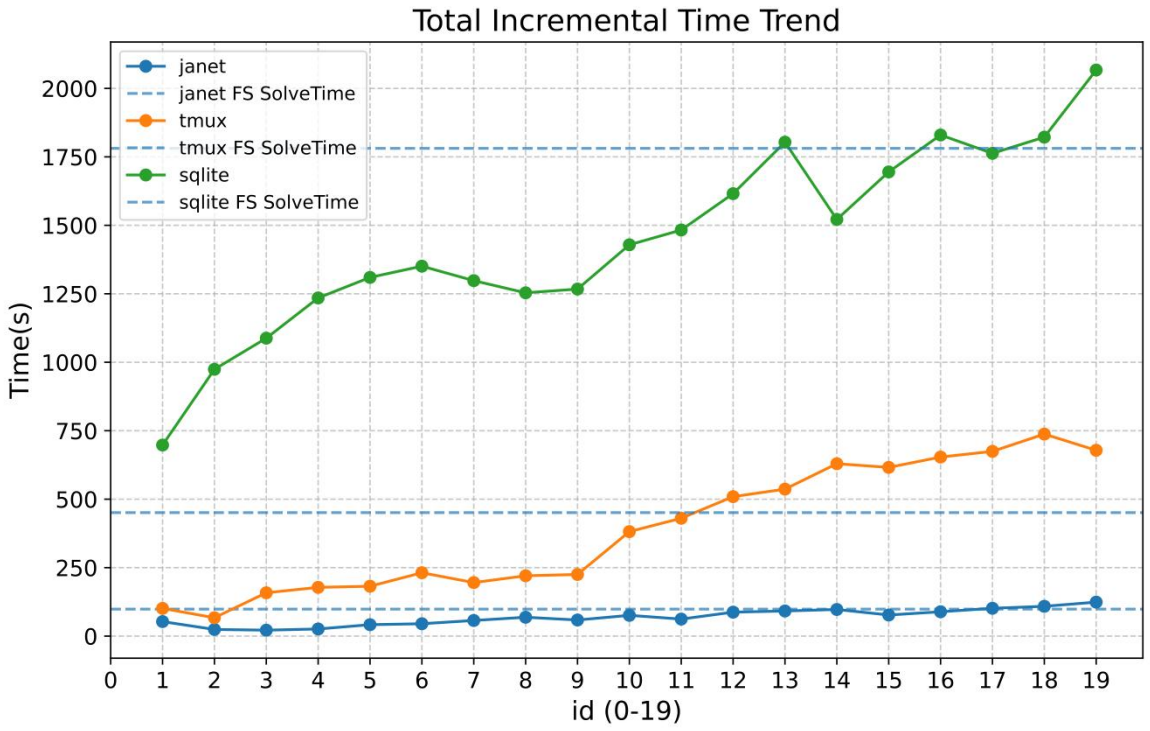
Average speedup



9.60× over full analysis 5.84× over reset-recompute

Evaluation

- **RQ3: Scalability with Code Changes**



Change ratios at the crossover with full analysis

Program	Deleted code	Inserted code
janet	67.11%	150.92%
tmux	33.82%	68.67%
sqlite	6.95%	5.66%

IncSFS stays faster across most change scales.

Evaluation

■ RQ4: Correctness

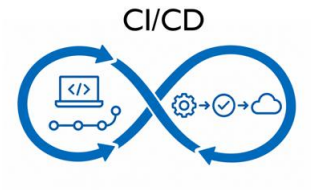
Exact points-to set comparison for maintained top-level variables

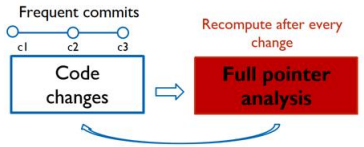
Program	Version pairs	Variable comparisons	Mismatches
janet	40	1,766,214	0
nginx	40	2,653,743	0
tmux	40	2,947,271	0
astyle	40	2,531,033	0
sqlite	40	5,582,440	0
zstd	40	8,142,750	0
Total	**240**	**23,623,451**	**0**

Zero mismatches in 23,623,451 comparisons across 240 version pairs.

Conclusion

Problem

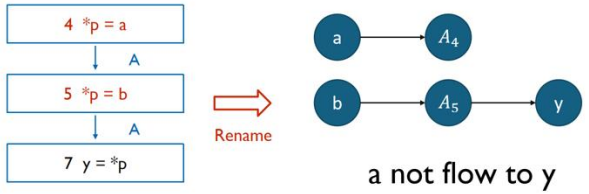




Frequent code changes make repeated full analysis a major scalability bottleneck in CI/CD, delaying feedback and slowing development.

Solution 1: Variable Renaming

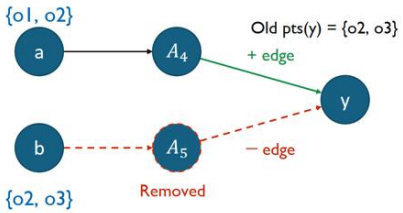
- Rename VFG variables to build a constraint graph.
- Update propagation relations over the renamed variables.



a not flow to y

Joint Delta Propagation: Example

- Use existing and new support to preserve unchanged facts.



- Candidate deltas at y

$\Delta_y^+ = \{o1, o2\}$

$\Delta_y^- = \{o2, o3\}$

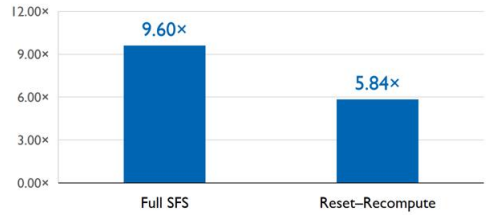
- Actual deltas at y

+ {o1} − {o3}

o2 is still supported by a

Results: Incremental Analysis Is Faster

Average speedup



9.60× over full analysis 5.84× over reset-recompute

Thank you!

Questions?

Artifact: https://doi.org/10.5281/zenodo.21771594