

# Reducing the Memory Footprint of IFDS-based Data-Flow Analyses Using Fine-Grained Garbage Collection

Dongjie He<sup>1</sup>, Yujiang Gui<sup>1</sup>, Yaoqing Gao<sup>2</sup>, Jingling Xue<sup>1</sup>

<sup>1</sup>School of Computer Science and Engineering  
University of New South Wales

<sup>2</sup>Huawei Toronto Research Center



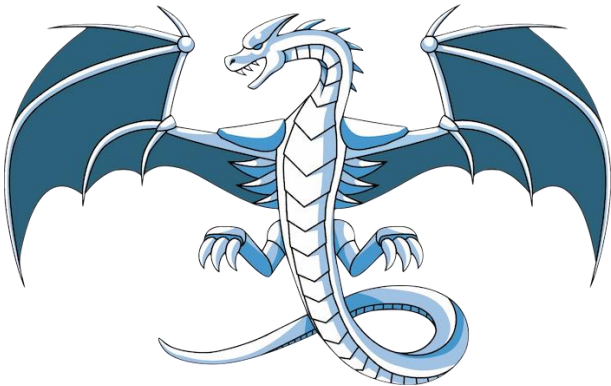
Presenter: Yujiang Gui

32<sup>nd</sup> ISSTA, July 2023



# The IFDS Algorithm

- Transforms an analysis into a graph-reachability problem
- Solves a wide range of data-flow analyses
  - Compiler optimization
  - Bug detection (ASE'18)
  - Taint analysis (PLDI'14, FSE'14)
  - Pointer analysis (ECOOP'16, ASE'21, TSE'23)
  - Typestate-like analysis (OOPSLA'08, PLDI'14)
  - ...
- Has been implemented in many tools

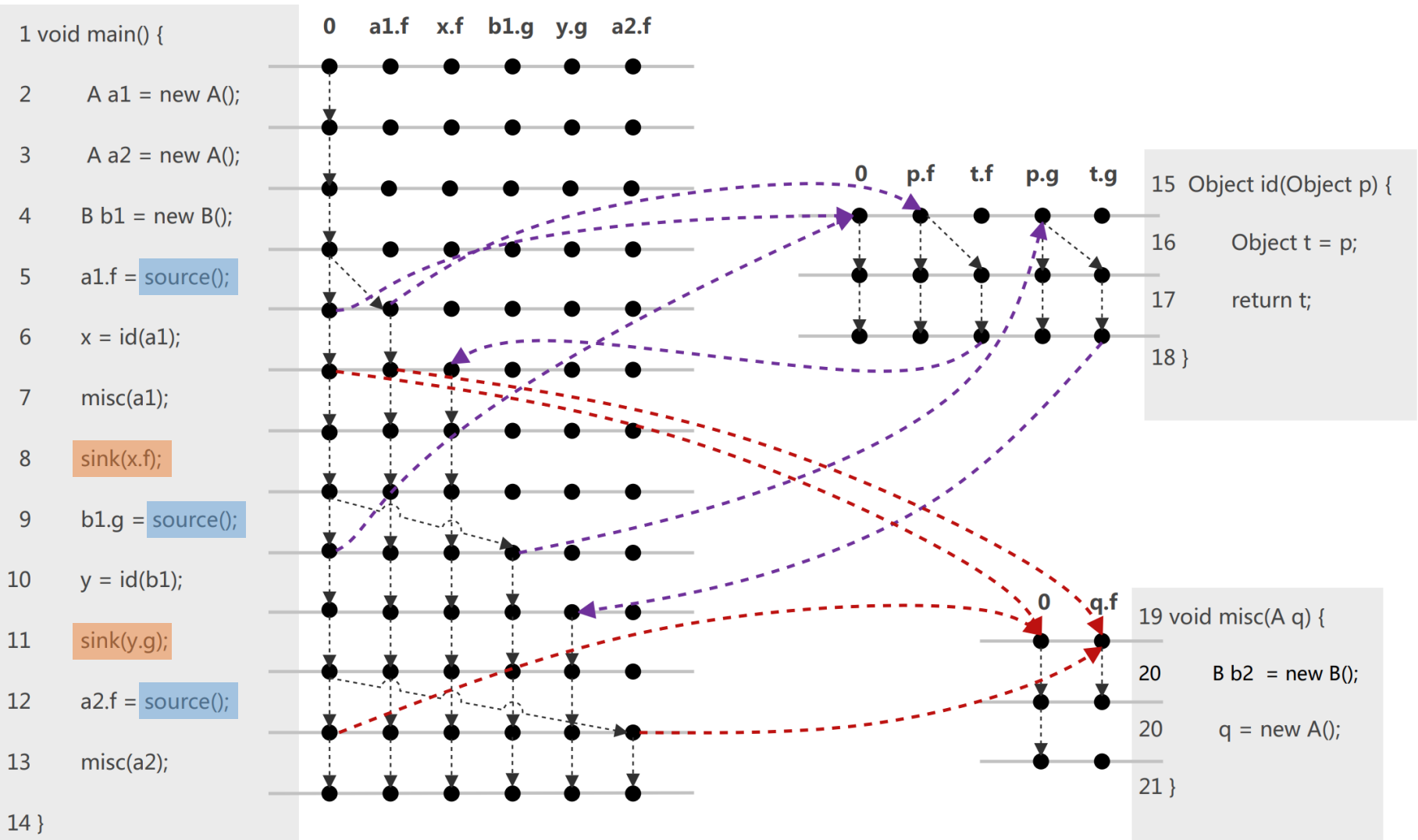


**WALA**

T. J. WATSON LIBRARIES FOR ANALYSIS

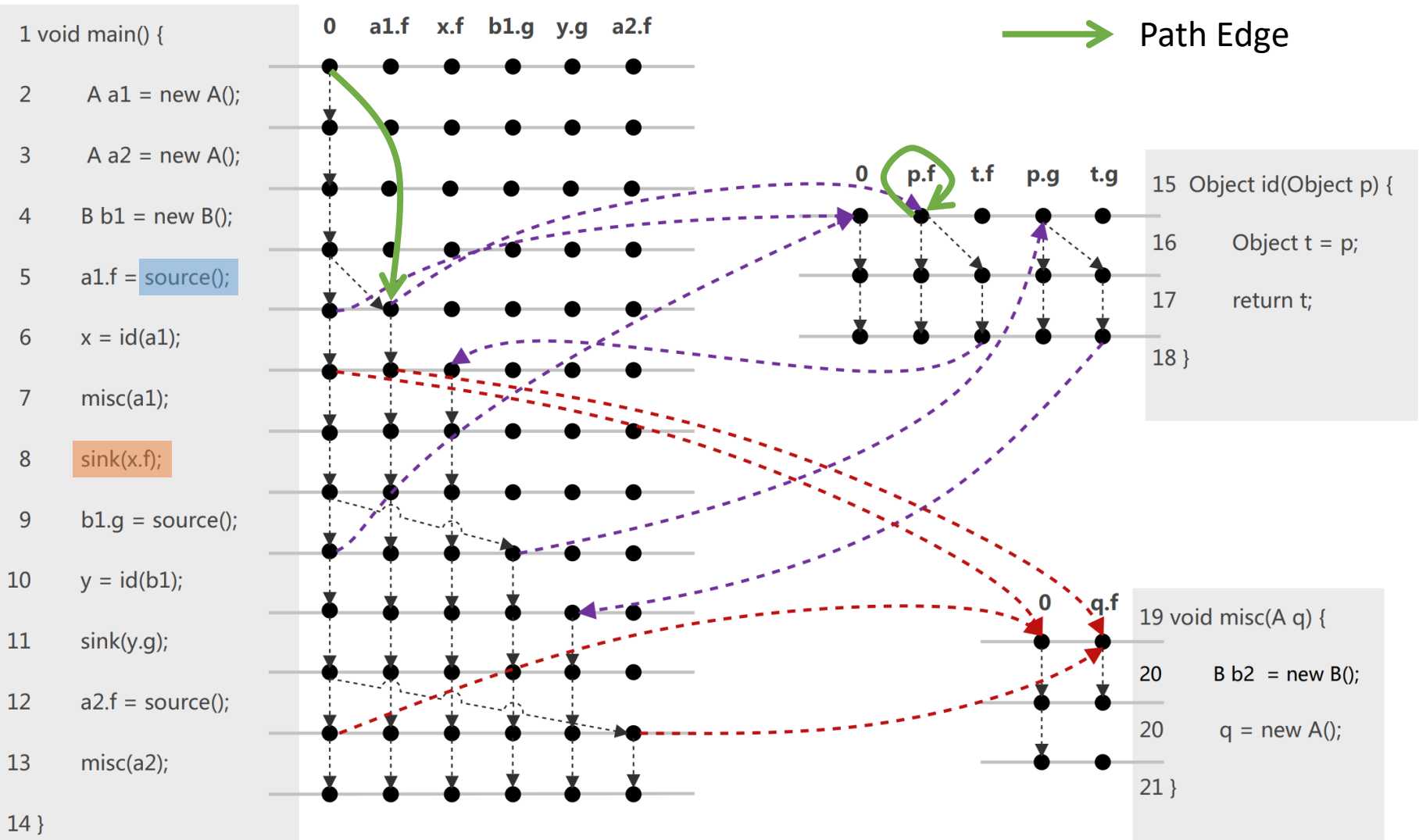
Reps, Thomas, Susan Horwitz, and Mooly Sagiv. "Precise Interprocedural Dataflow Analysis via Graph Reachability.", POPL'95.

# A Running Example: Taint Analysis

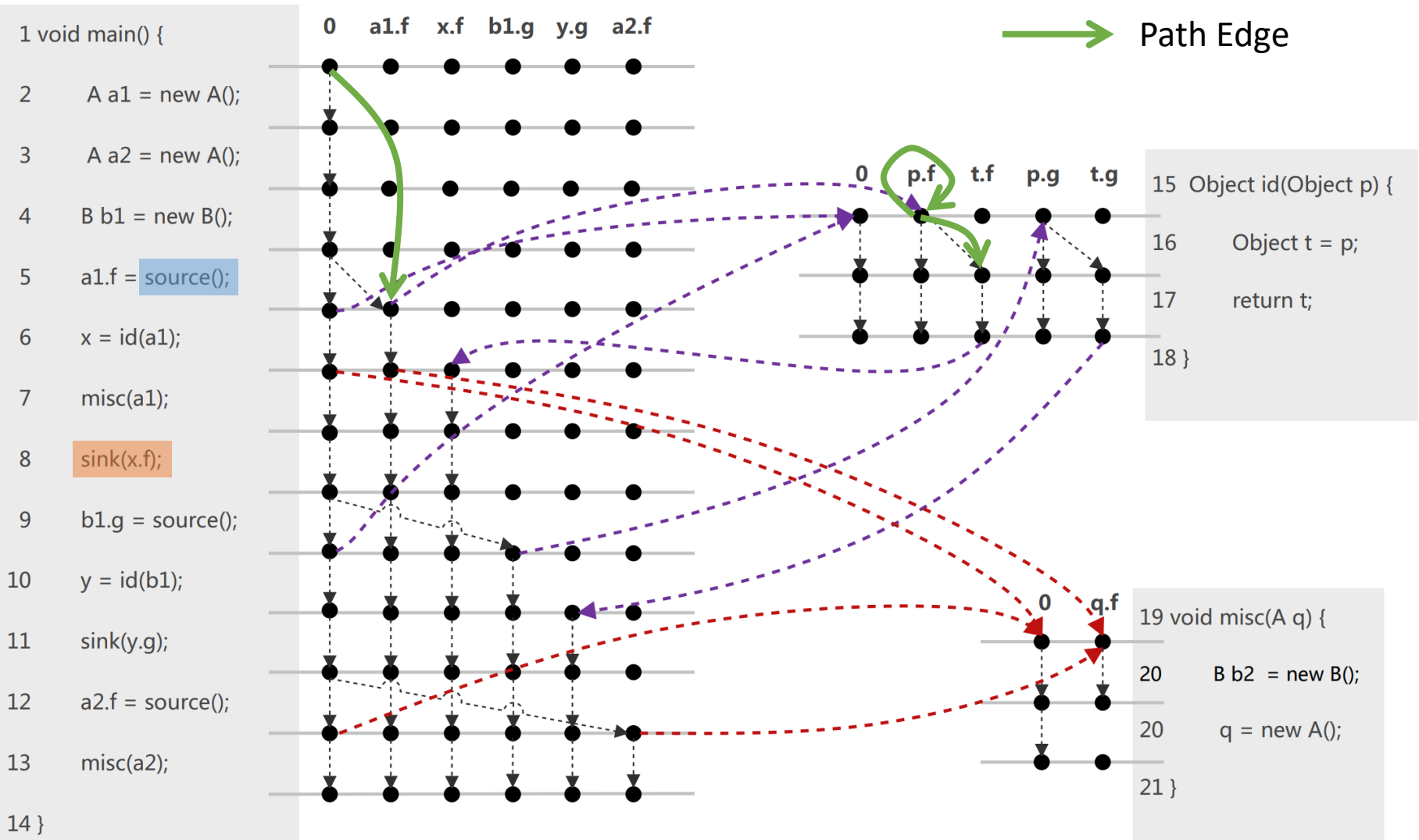




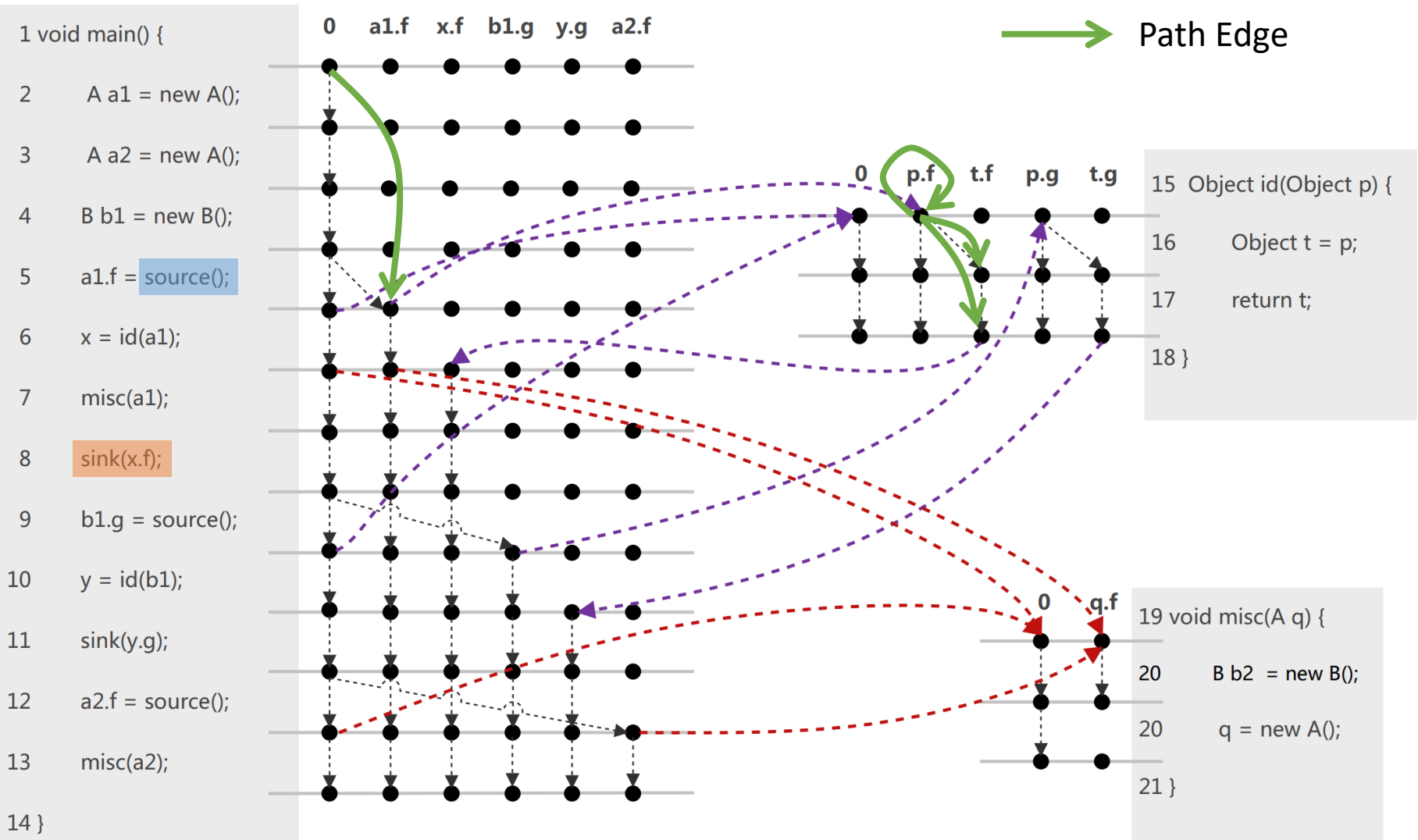
# A Running Example: Taint Analysis



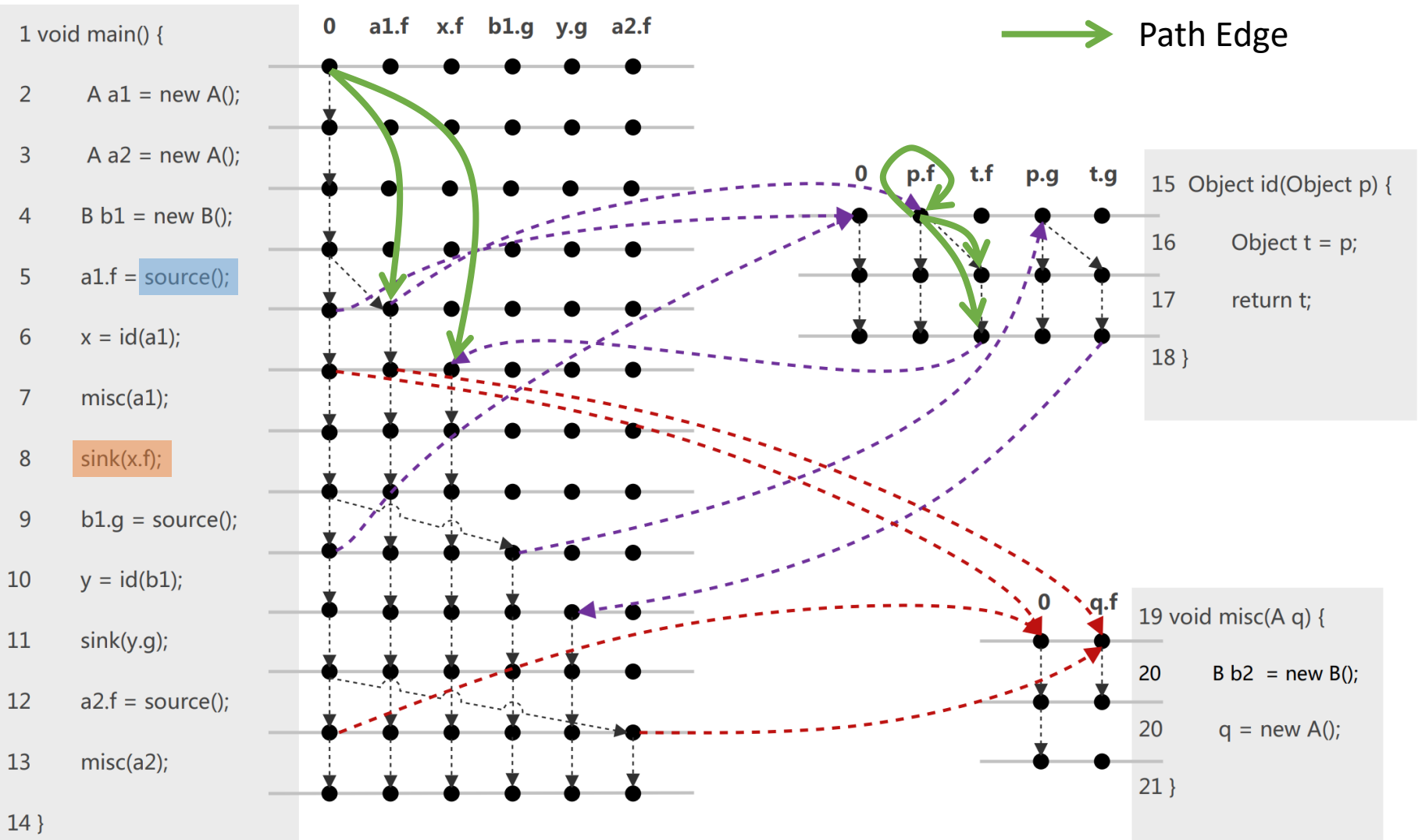
# A Running Example: Taint Analysis



# A Running Example: Taint Analysis

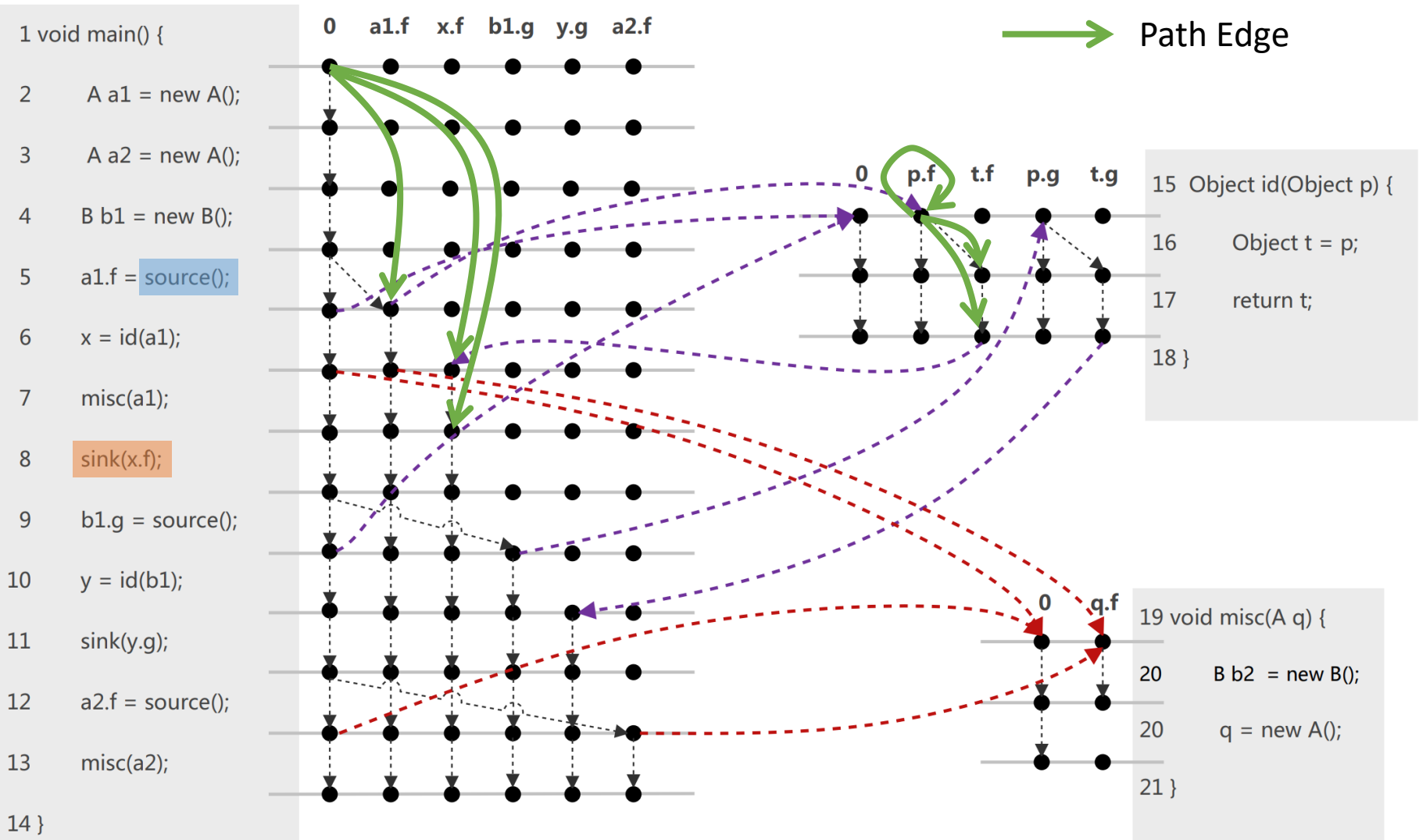


# A Running Example: Taint Analysis

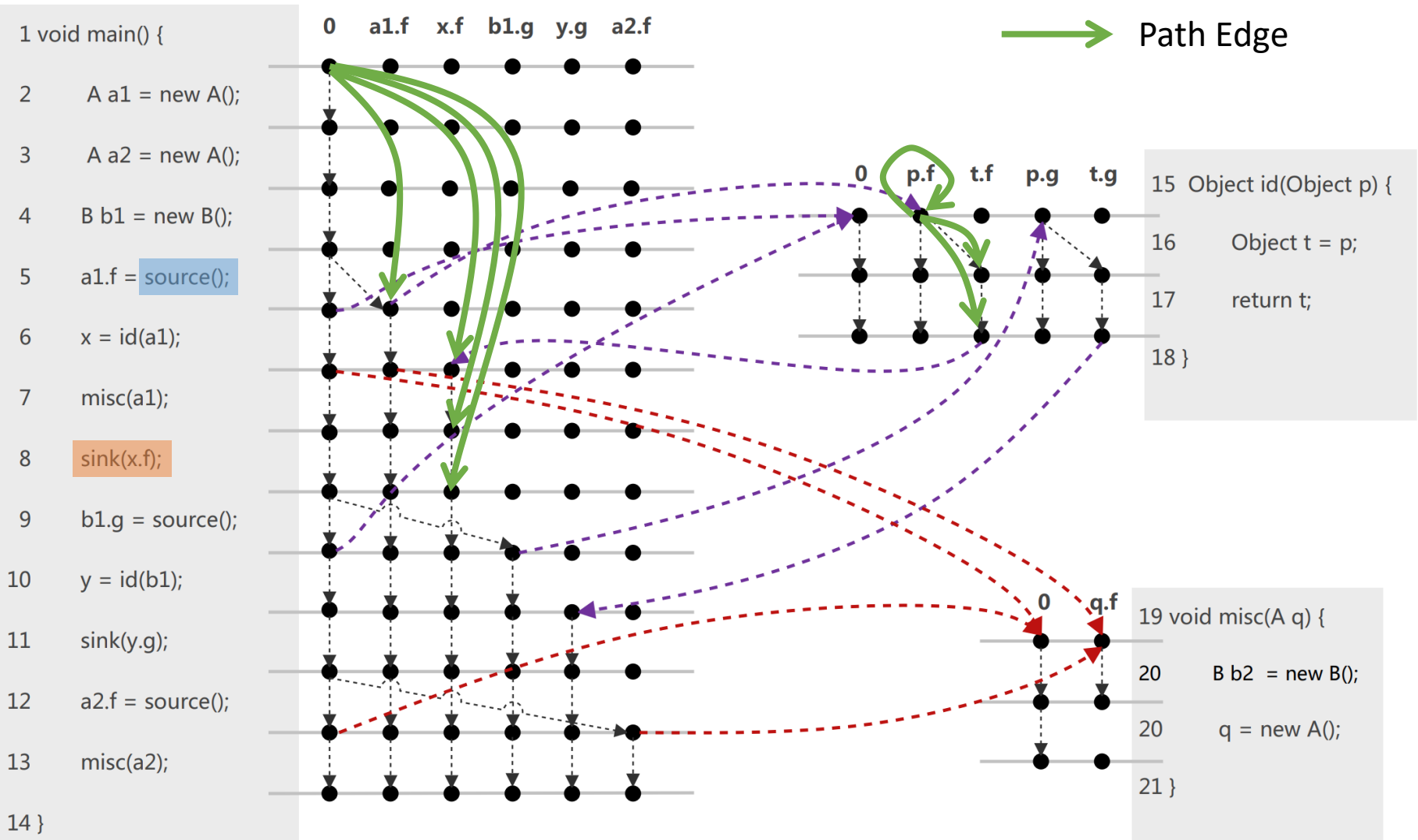




# A Running Example: Taint Analysis

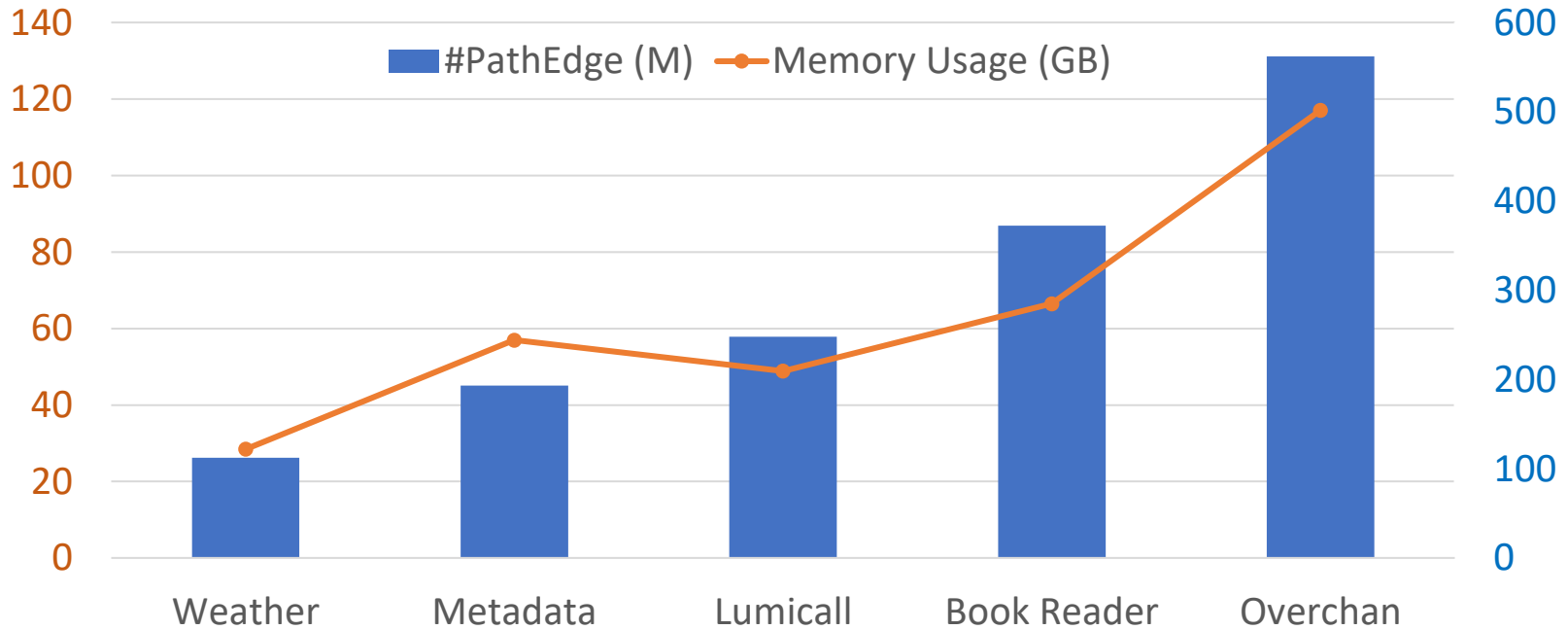


# A Running Example: Taint Analysis



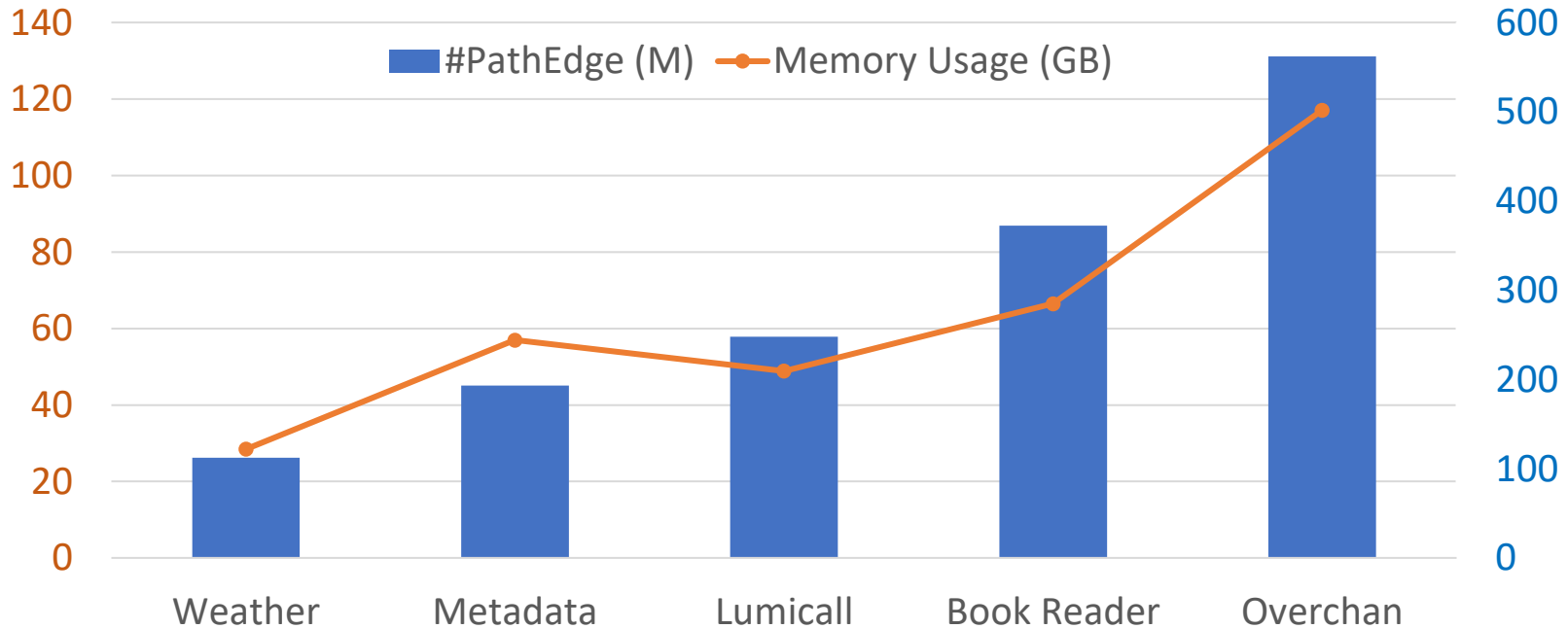
# The Drawback: Memory-Intensive

- Maintains a huge number of path edges



# The Drawback: Memory-Intensive

- Maintains a huge number of path edges



- Fails to solve taint analysis on some Android apps
  - Even on a server with **730GB RAM**
- Slows down the analysis
  - Frequent re-hashing operations

# Objective

Improving the Scalability and Efficiency

## Improving the Scalability and Efficiency

- Discovers facts only at some program points
  - Taint analysis

## Improving the Scalability and Efficiency

- Discovers facts only at some program points
  - Taint analysis
- Non-live path edges
  - Visited only once (**86.97%**)
  - Waste memory resources

## Improving the Scalability and Efficiency

- Discovers facts only at some program points
  - Taint analysis
- Non-live path edges
  - Visited only once (**86.97%**)
  - Waste memory resources
- **Garbage collection!** But how to...
  - Preserve important properties
    - Correctness
    - Precision
    - Termination
  - Avoid redundant computations



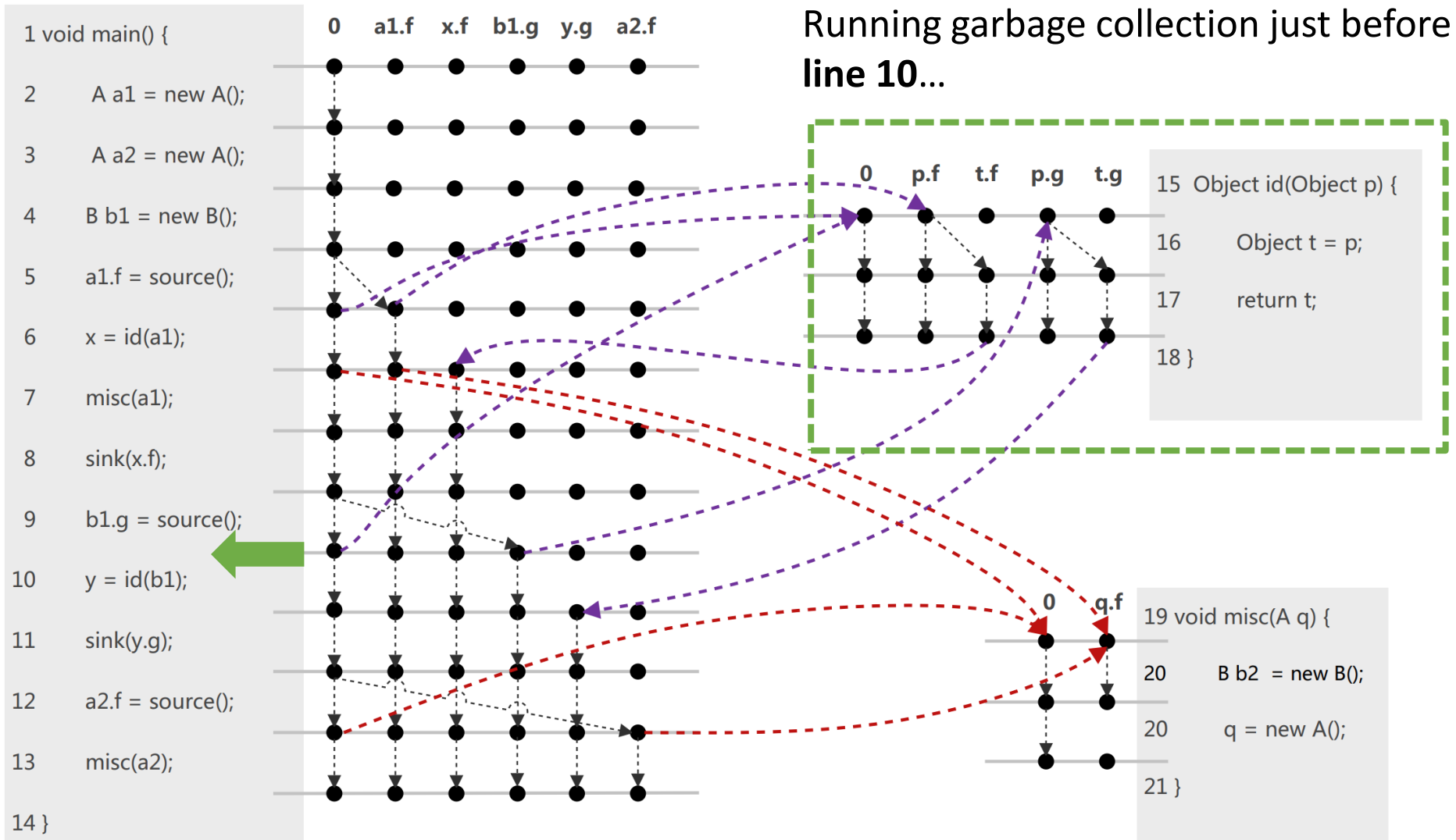
## Sustainable Solving: Reducing The Memory Footprint of IFDS-Based Data Flow Analyses Using Intelligent Garbage Collection

Steven Arzt  
Secure Software Engineering Group,  
Fraunhofer Institute for Secure Information Technology  
Darmstadt, Germany  
Email: [steven.arzt@sit.fraunhofer.de](mailto:steven.arzt@sit.fraunhofer.de)

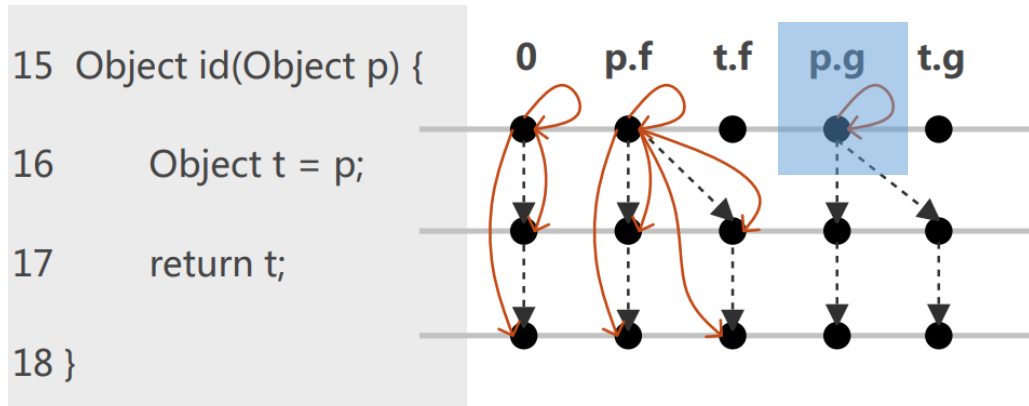
### CleanDroid (ICSE'21)

- **With 2 major limitations**
  - Coarse-grained
  - Allows redundant computations

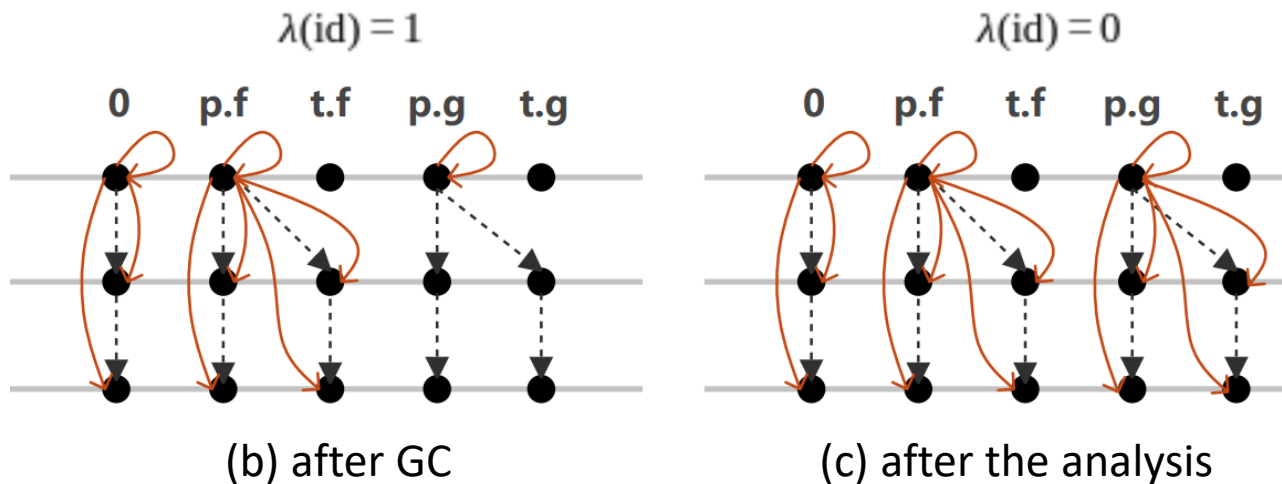
# Limitation 1: Coarse Granularity



# Limitation 1: Coarse Granularity



(a) before GC

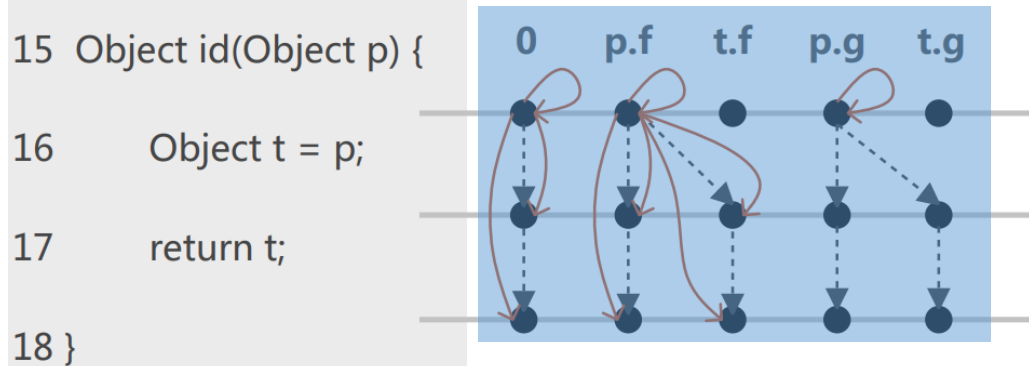


(b) after GC

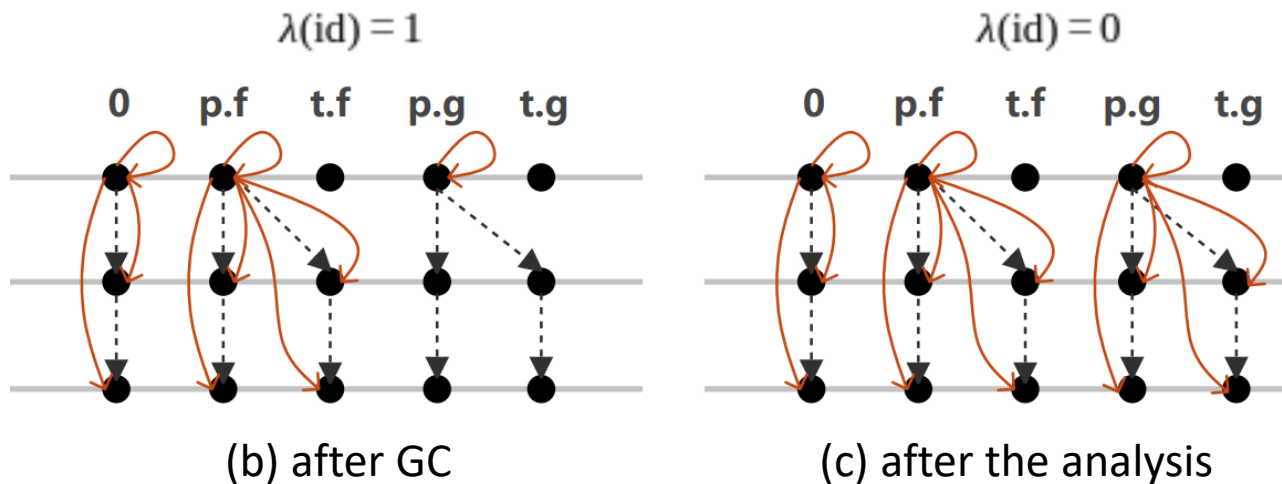
(c) after the analysis

Maximum #PathEdge maintained: 13.

# Limitation 1: Coarse Granularity

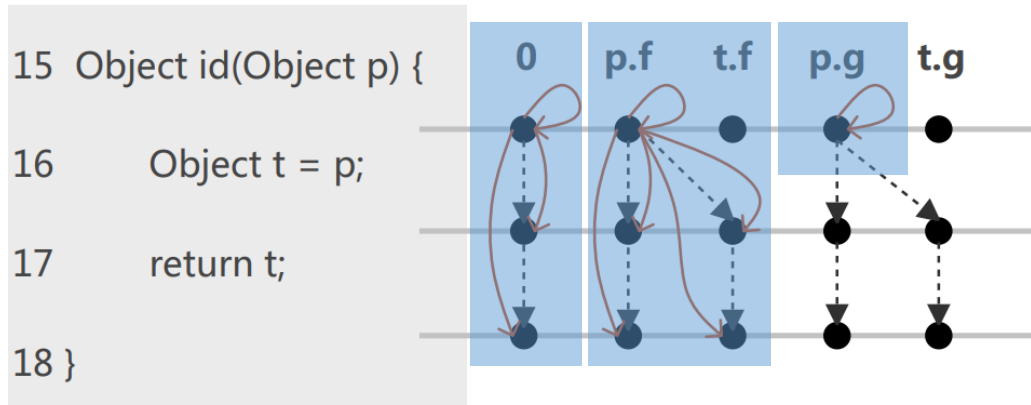


(a) before GC



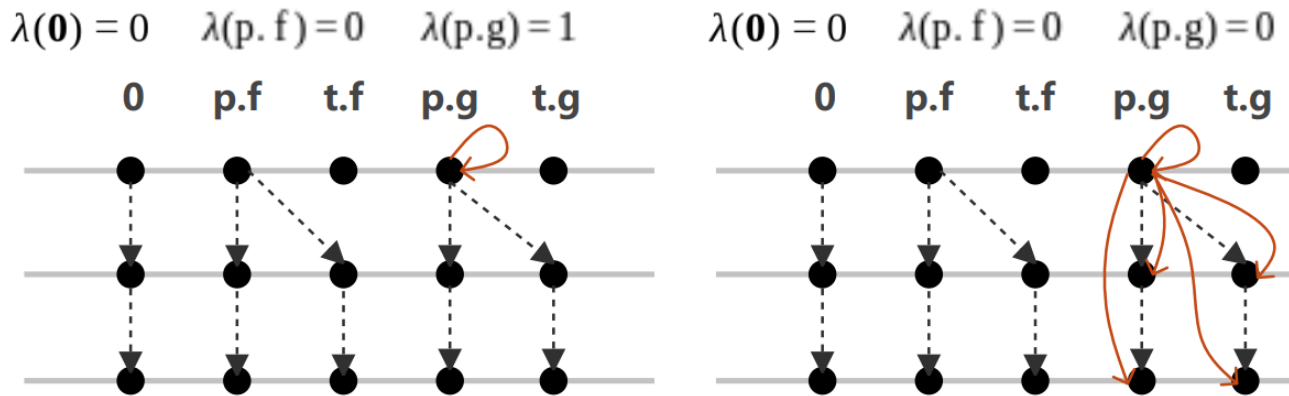
Method-Level: Coarse-Grained

# Fine-Grained Data-Fact-Level GC



Observation 1: The path edges with different anchor sites are handled **independently**.

(a) before GC

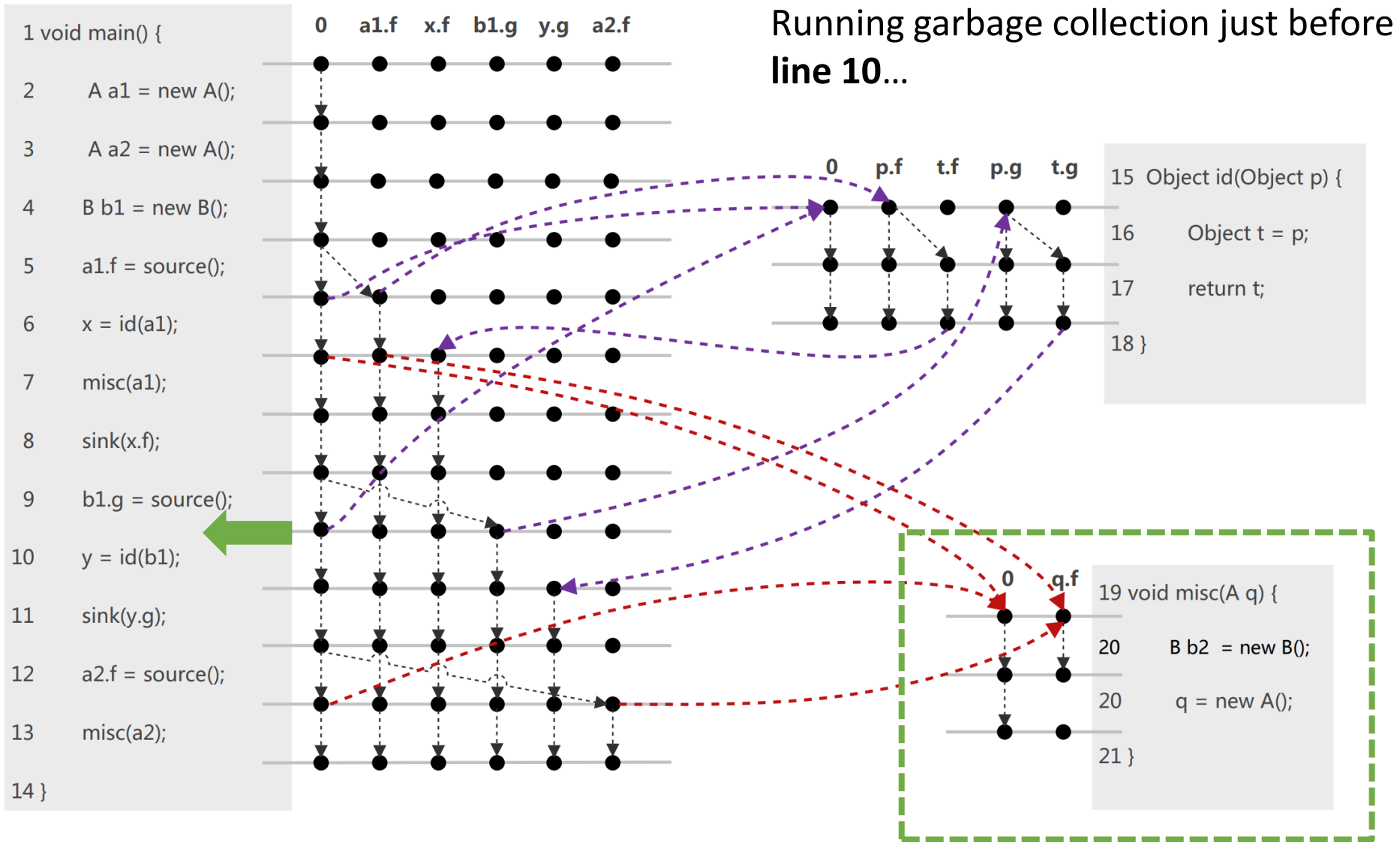


(b) after GC

(c) after the analysis

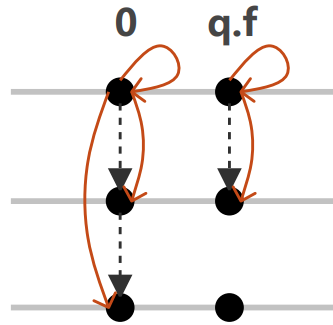
Maximum #PathEdge maintained: 9.

# Limitation 2: Redundant Computations

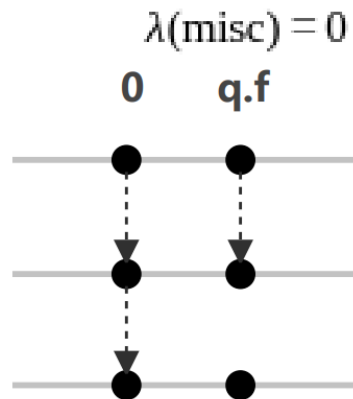


# Limitation 2: Redundant Computations

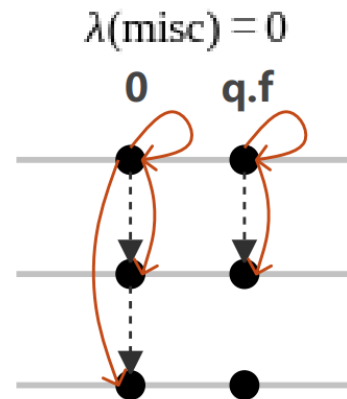
```
19 void misc(A q) {  
20     B b2 = new B();  
20     q = new A();  
21 }
```



(a) before GC



(b) after GC

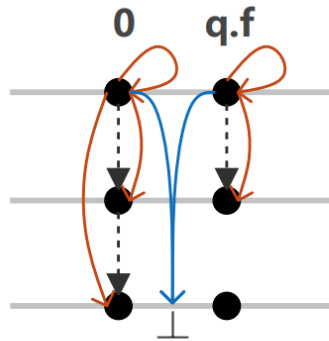


(c) after the analysis

The method `misc()` is analyzed redundantly.

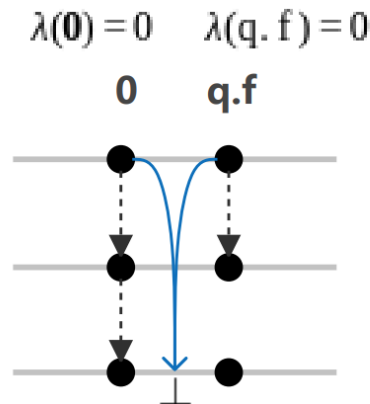
# Avoiding Redundant Computations

```
19 void misc(A q) {  
20     B b2 = new B();  
20     q = new A();  
21 }
```

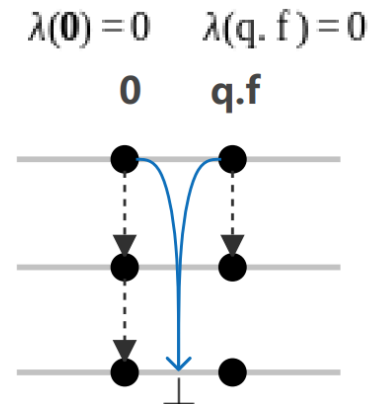


Observation 2: All path edges sharing the same anchor site are generated from the same self-loop path edge.

(a) before GC



(b) after GC



(c) after the analysis

With redundancy-avoiding edges, the method `misc()` is analyzed only once.



# Data-Fact-Level Path-Edge GC Algorithm

```

1 Algorithm IFDS( $G_{IP}^\# = (N^\#, E^\#)$ )
2   InitPECollector()
3   PathEdge  $\leftarrow \mathcal{W} \leftarrow \mathcal{S} \leftarrow \emptyset$ 
4   Propagate( $\langle s_{main}, \emptyset \rangle \rightarrow \langle s_{main}, \emptyset \rangle$ )
5   while  $\mathcal{W} \neq \emptyset$  do
6     Pop  $\langle s_m, d_1 \rangle \rightarrow \langle n, d_2 \rangle$  from  $\mathcal{W}$ 
7     if  $n$  is a call node then
8       Let  $m'$  be the method called at  $n$  and  $n'$  be the return node of  $n$ 
9       for  $d_3$  such that  $\langle n, d_2 \rangle \rightarrow \langle s_{m'}, d_3 \rangle \in E^\#$  do
10        ADG = ADG  $\cup \{ \langle s_m, d_1 \rangle \rightarrow \langle s_{m'}, d_3 \rangle \}$ 
11        Propagate( $\langle s_{m'}, d_3 \rangle \rightarrow \langle s_{m'}, d_3 \rangle$ )
12        for  $\langle s_{m'}, d_3 \rangle \rightarrow \langle e_{m'}, d_4 \rangle \in \mathcal{S} \wedge \langle e_{m'}, d_4 \rangle \rightarrow \langle n', d_5 \rangle \in E^\#$  do
13          Propagate( $\langle s_m, d_1 \rangle \rightarrow \langle n', d_5 \rangle$ )
14        for  $d_3$  such that  $\langle n, d_2 \rangle \rightarrow \langle n', d_3 \rangle \in E^\#$  do
15          Propagate( $\langle s_m, d_1 \rangle \rightarrow \langle n', d_3 \rangle$ )
16      if  $n$  is an exit node then
17        if  $\langle s_m, d_1 \rangle \rightarrow \langle n, d_2 \rangle \notin \mathcal{S}$  then
18          Insert  $\langle s_m, d_1 \rangle \rightarrow \langle n, d_2 \rangle$  into  $\mathcal{S}$ 
19          for each call site  $c$  that calls  $m$  do
20            Let  $m''$  ( $n''$ ) be the containing method (the return node) of  $c$ 
21            for  $\langle s_{m''}, d_3 \rangle \rightarrow \langle c, d_4 \rangle \in PathEdge \wedge \langle c, d_4 \rangle \rightarrow$ 
22               $\langle s_m, d_1 \rangle \in E^\# \wedge \langle n, d_2 \rangle \rightarrow \langle n'', d_5 \rangle \in E^\#$  do
23                Propagate( $\langle s_{m''}, d_3 \rangle \rightarrow \langle n'', d_5 \rangle$ )
24            if  $n$  is a normal node or a return node then
25              for  $\langle n', d_3 \rangle$  such that  $\langle n, d_2 \rangle \rightarrow \langle n', d_3 \rangle \in E^\#$  do
26                Propagate( $\langle s_m, d_1 \rangle \rightarrow \langle n', d_3 \rangle$ )
27            OnEdgeProcessed( $\langle s_m, d_1 \rangle \rightarrow \langle n, d_2 \rangle$ )
28      Procedure Propagate( $\langle s_m, d_1 \rangle \rightarrow \langle n, d_2 \rangle$ )
29        if  $\langle s_m, d_1 \rangle \rightarrow \langle n, d_2 \rangle \notin PathEdge$  then
30          if  $\langle s_m, d_1 \rangle = \langle n, d_2 \rangle$  then // a self-loop edge
31            if  $\langle s_m, d_1 \rangle \rightarrow \perp \in \mathcal{RAEdges}$  then
32              return // avoid redundant re-computations
33            else
34               $\mathcal{RAEdges} = \mathcal{RAEdges} \cup \{ \langle s_m, d_1 \rangle \rightarrow \perp \}$ 
35            OnEdgeScheduled( $\langle s_m, d_1 \rangle \rightarrow \langle n, d_2 \rangle$ )
36          Insert  $\langle s_m, d_1 \rangle \rightarrow \langle n, d_2 \rangle$  into both  $PathEdge$  and  $\mathcal{W}$ 

```

```

1 Procedure InitPECollector()
2    $\lambda = \{ \alpha \mapsto 0 \}$ 
3    $\mathcal{RAEdges} = \mathcal{C} = \emptyset$ 
4 Procedure OnEdgeScheduled( $\langle s_m, d_1 \rangle \rightarrow \langle n, d_2 \rangle$ )
5   Consume( $\langle s_m, d_1 \rangle \rightarrow \langle n, d_2 \rangle$ ) // do some analysis-specific task
6    $\lambda = \begin{cases} \alpha \mapsto \lambda(\alpha) + 1 & \text{if } \alpha = \langle s_m, d_1 \rangle \\ \alpha \mapsto \lambda(\alpha) & \text{otherwise} \end{cases}$ 
7    $\mathcal{C} = \mathcal{C} \cup \{ \alpha \}$ 
8 Procedure OnEdgeProcessed( $\langle s_m, d_1 \rangle \rightarrow \langle n, d_2 \rangle$ )
9    $\lambda = \begin{cases} \alpha \mapsto \lambda(\alpha) - 1 & \text{if } \alpha = \langle s_m, d_1 \rangle \\ \alpha \mapsto \lambda(\alpha) & \text{otherwise} \end{cases}$ 
10 Procedure RunFineGrainedPECollector()
11   foreach  $\alpha \in \mathcal{C}$  do
12     if  $\forall \alpha' \in ADGTC(\alpha, ADG): \lambda(\alpha') = 0$  then
13       // Remove path edges with  $\alpha$  as their anchor site
14       foreach  $e : \langle s_m, d_1 \rangle \rightarrow \langle n, d_2 \rangle \in PathEdge$  do
15         if  $\alpha = \langle s_m, d_1 \rangle$  then
16            $PathEdge = PathEdge \setminus \{ e \}$ 
17          $\mathcal{C} = \mathcal{C} \setminus \{ \alpha \}$ 

```

# Our Approach: FPC

Using a fine-grained path edge garbage collector

- Data-fact-level
- Light-weight
- Efficient
- No redundancy

# Implementation

- Implemented in **FlowDroid (PLDI'14)** to compare with **CleanDroid (ICSE'21)**
- In about **600 lines** of Java code
- Has been **merged** into FlowDroid
- Available at <https://github.com/DongjieHe/FPC>



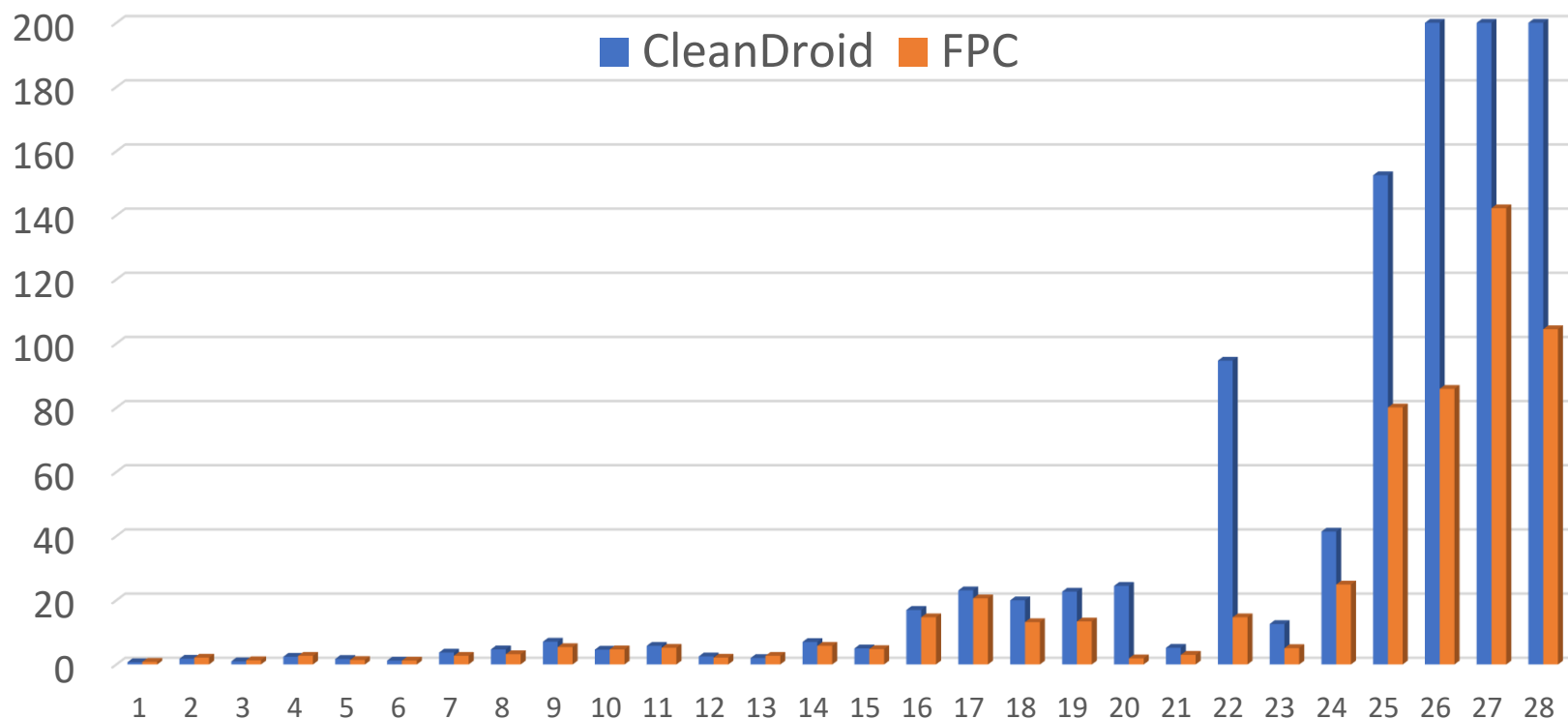
# Evaluation

- Benchmark
  - 58 apps from 2 previous papers: **ASE'19** and **CGO'21**
- Evaluate on 2 metrics
  - Memory usage
  - Analysis time
- Evaluate under varying GC intervals
  - Default: 1-second

# RQ1: Memory Usage

Memory budget: 200 GB per app

Maximum Memory Usage (GB)



FPC can reduce the peak memory usage by 1.4× on average and can scale **3 more apps** than baseline.

# RQ2: Analysis Time

Time budget: 3 hours per app

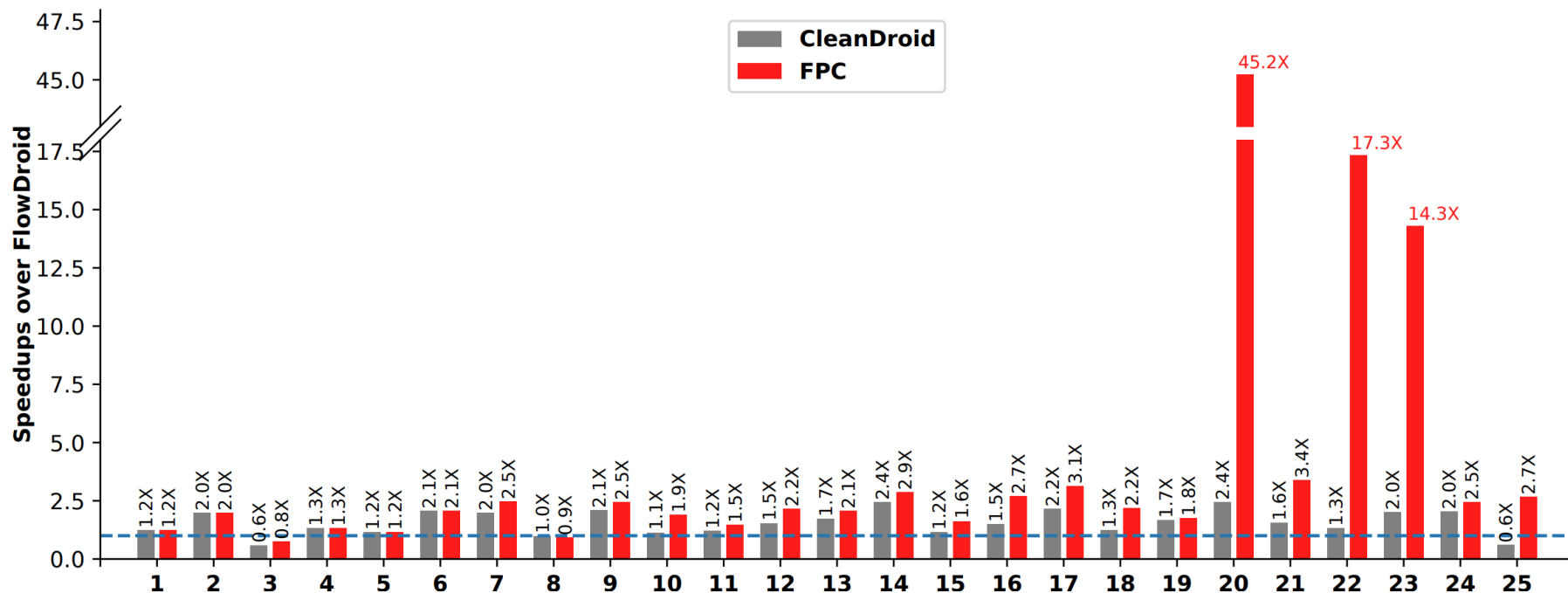
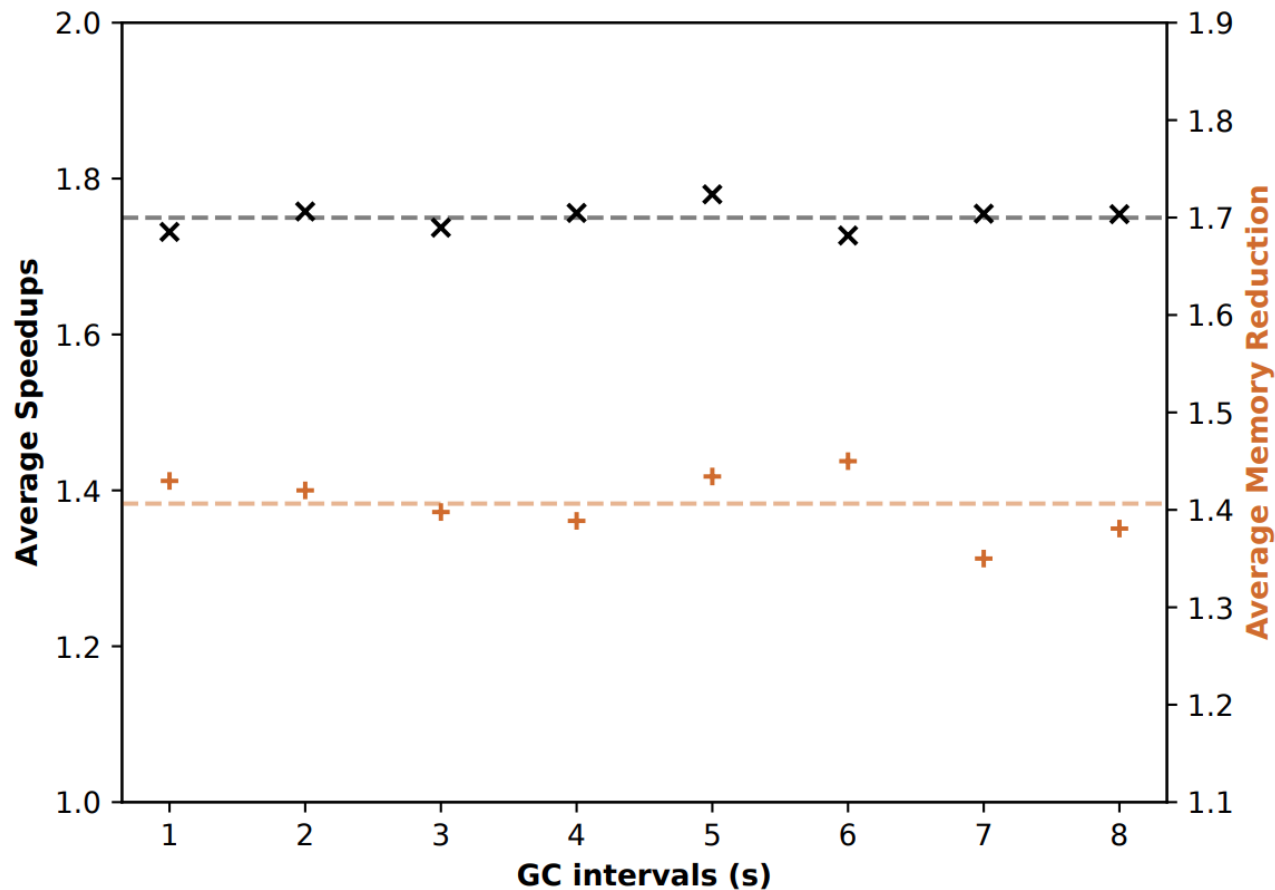


Figure 9: Comparing the speedups of FPC and CLEANROID with FLOWDROID as the baseline.

FPC can improve both the scalability and efficiency of CleanDroid, the speedups range from 0.9x to 18.5x with an average of 1.7x.

# RQ3: Varying GC Intervals



FPC has improved CleanDroid by an average of  $1.40\times \pm 0.03$  for the memory usage, and  $1.74\times \pm 0.02$  for the analysis time. The result obtained using 1-second GC interval is **reliable**.

# Thank you!

**Dongjie He:** [dongjieh@cse.unsw.edu.au](mailto:dongjieh@cse.unsw.edu.au)

**Yujiang Gui:** [yujiang.gui@unsw.edu.au](mailto:yujiang.gui@unsw.edu.au)