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

Dongjie He[1], Yujiang Gui[1], Yaoqing Gao[2], Jingling Xue[1]

[1]School of Computer Science and Engineering
University of New South Wales

[2]Huawei Toronto Research Center

Presenter: Yujiang Gui

32nd 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

Soot

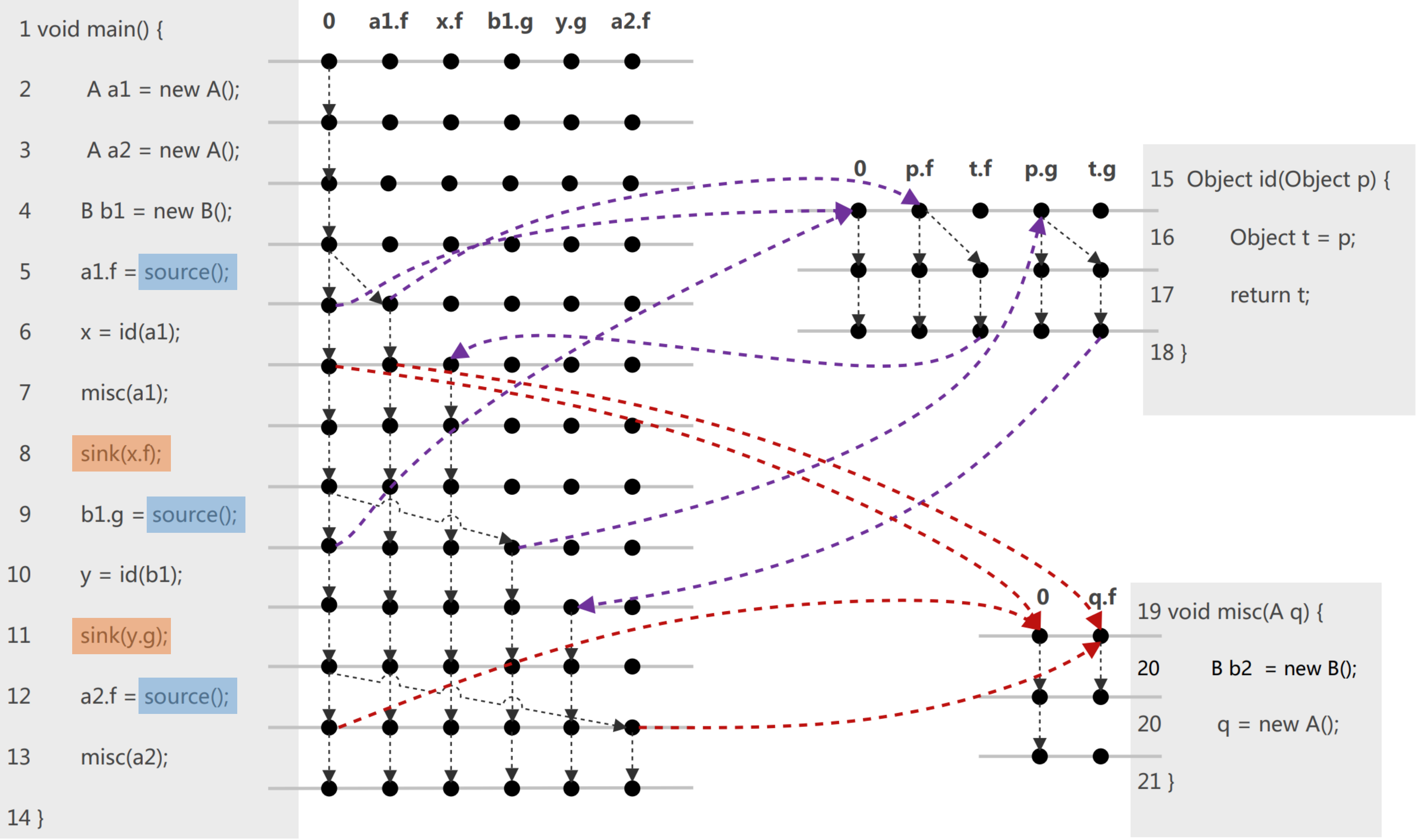
HEROS

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

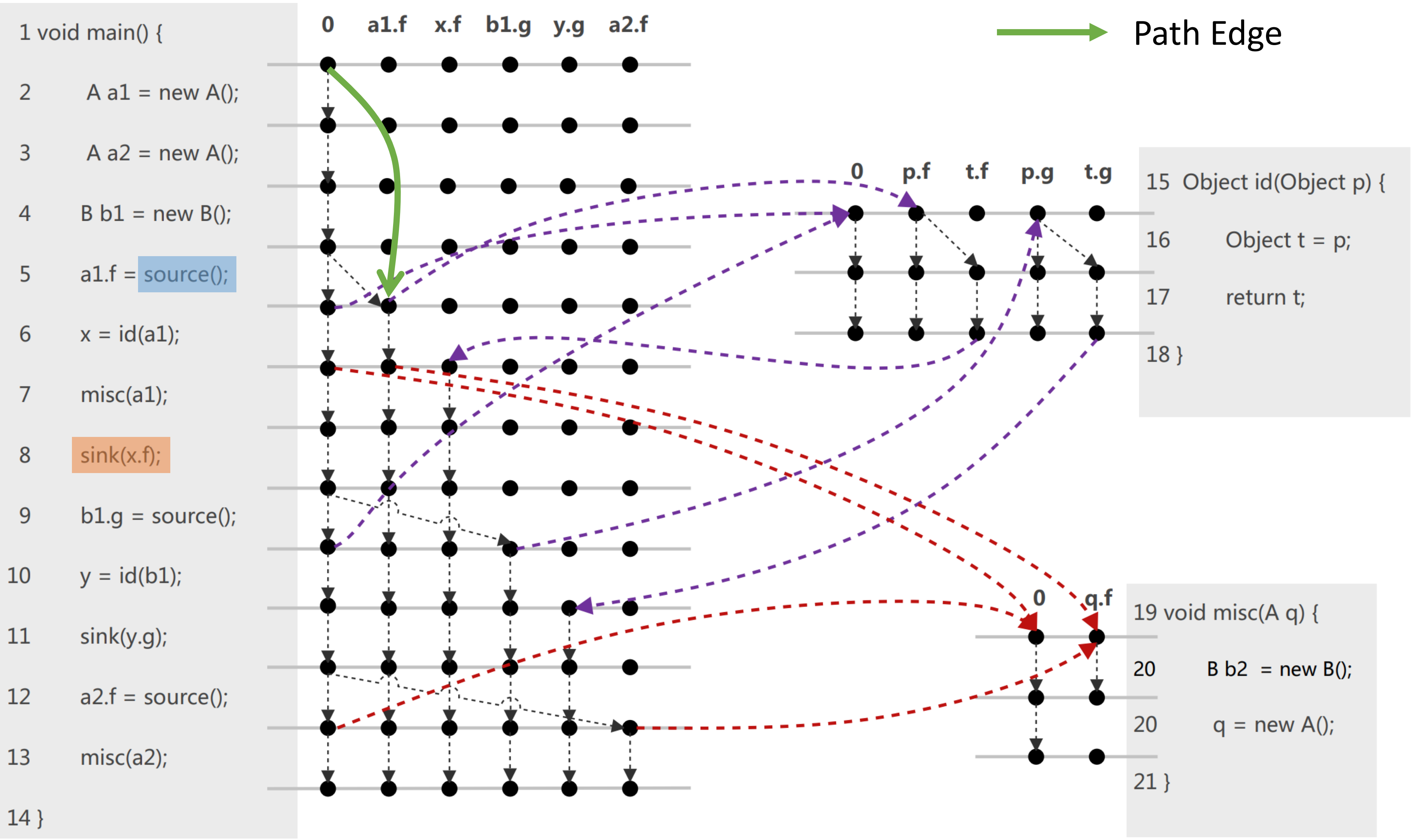


```
void main() {
    A a1 = new A();
    A a2 = new A();
    B b1 = new B();
    a1.f = source();
    x = id(a1);
    misc(a1);
    sink(x.f);
    b1.g = source();
    y = id(b1);
    sink(y.g);
    a2.f = source();
    misc(a2);
}
```

```
Object id(Object p) {
    Object t = p;
    return t;
}
```

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

A Running Example: Taint Analysis



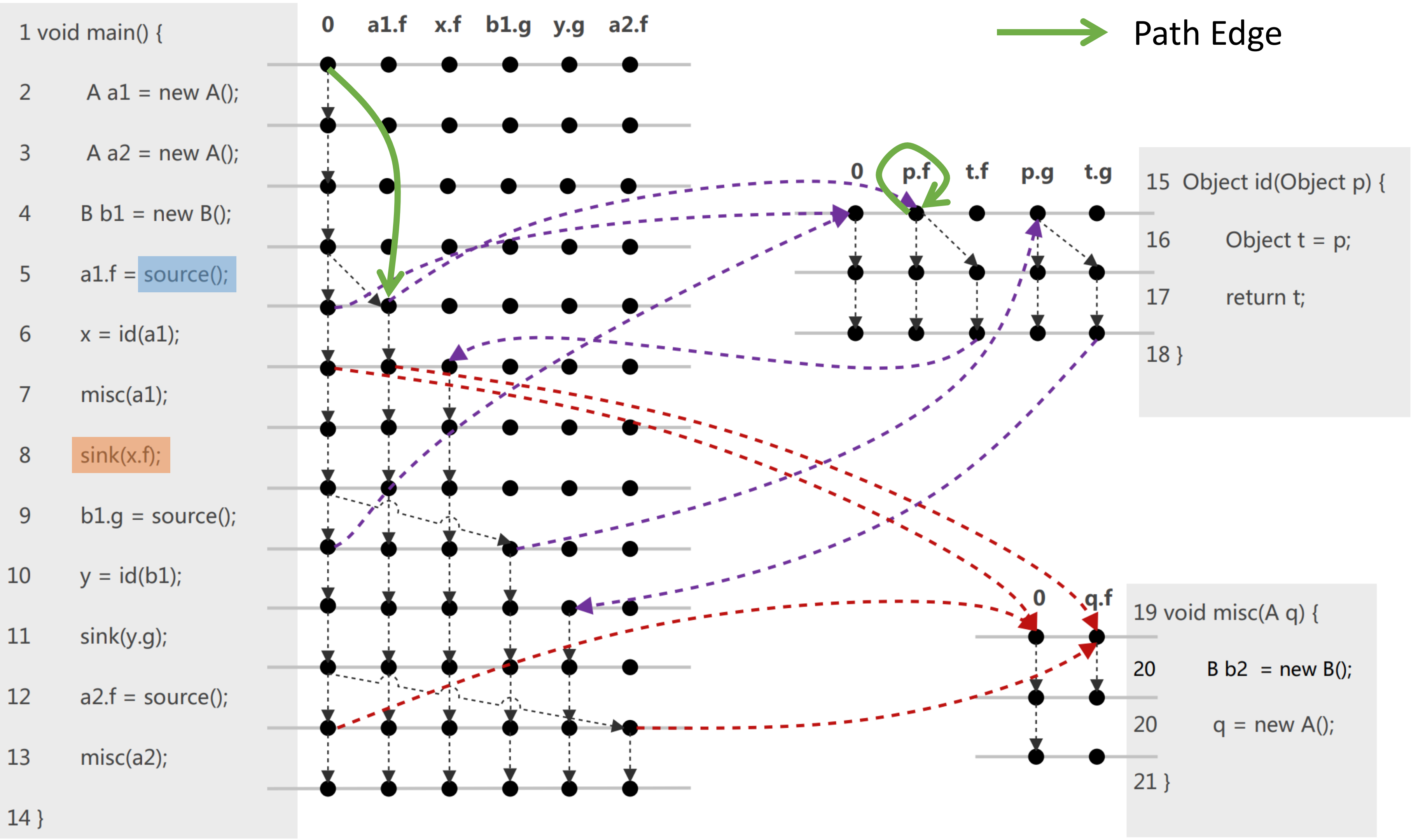
```
1 void main() {
2       A a1 = new A();
3       A a2 = new A();
4       B b1 = new B();
5       a1.f = source();
6       x = id(a1);
7       misc(a1);
8       sink(x.f);
9       b1.g = source();
10      y = id(b1);
11      sink(y.g);
12      a2.f = source();
13      misc(a2);
14 }
```

```
15  Object id(Object p) {
16        Object t = p;
17        return t;
18 }
```

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

🟩→ Path Edge

A Running Example: Taint Analysis

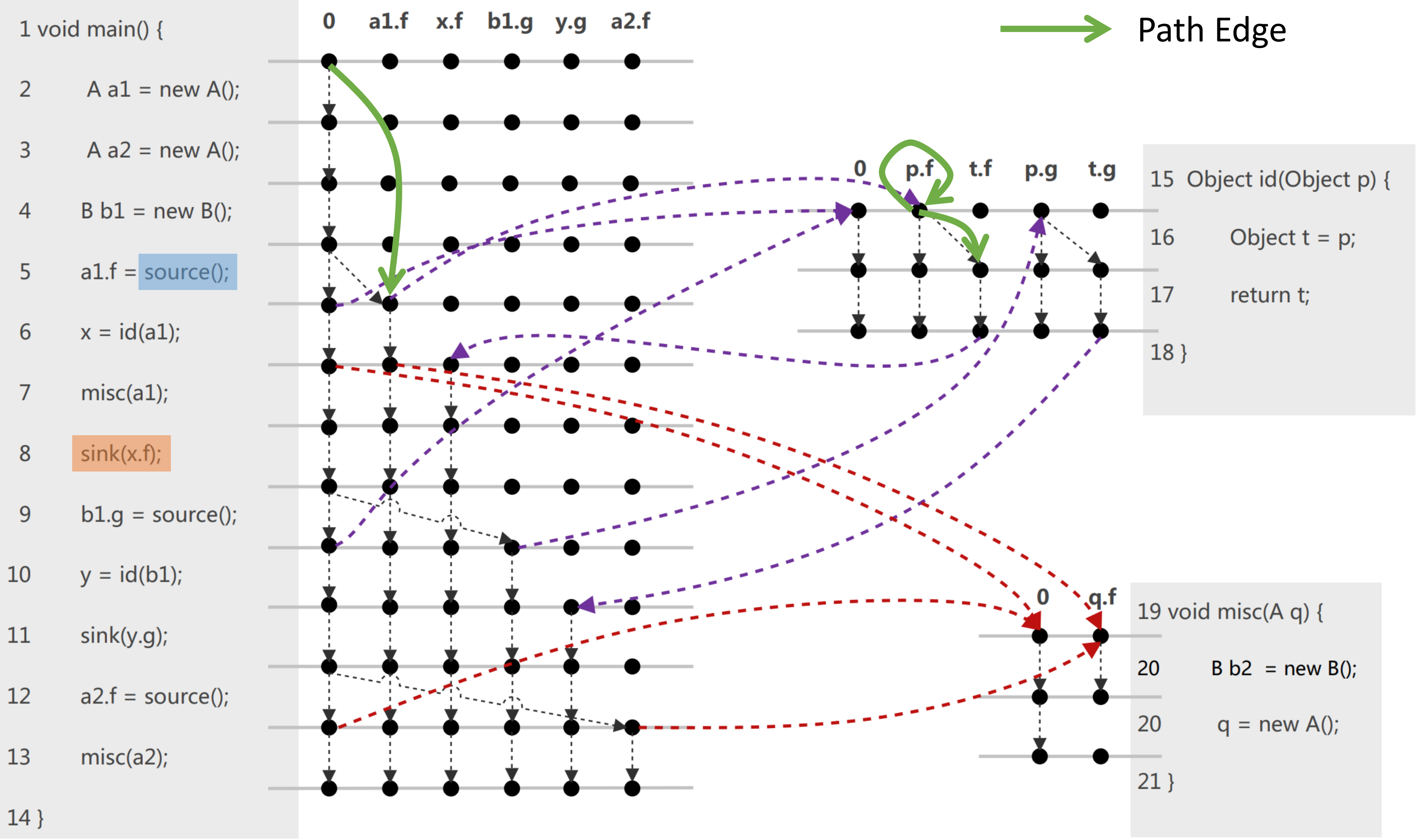


```
1 void main() {
2       A a1 = new A();
3       A a2 = new A();
4       B b1 = new B();
5       a1.f = source();
6       x = id(a1);
7       misc(a1);
8       sink(x.f);
9       b1.g = source();
10      y = id(b1);
11      sink(y.g);
12      a2.f = source();
13      misc(a2);
14 }
```

```
15  Object id(Object p) {
16        Object t = p;
17        return t;
18 }
```

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

A Running Example: Taint Analysis

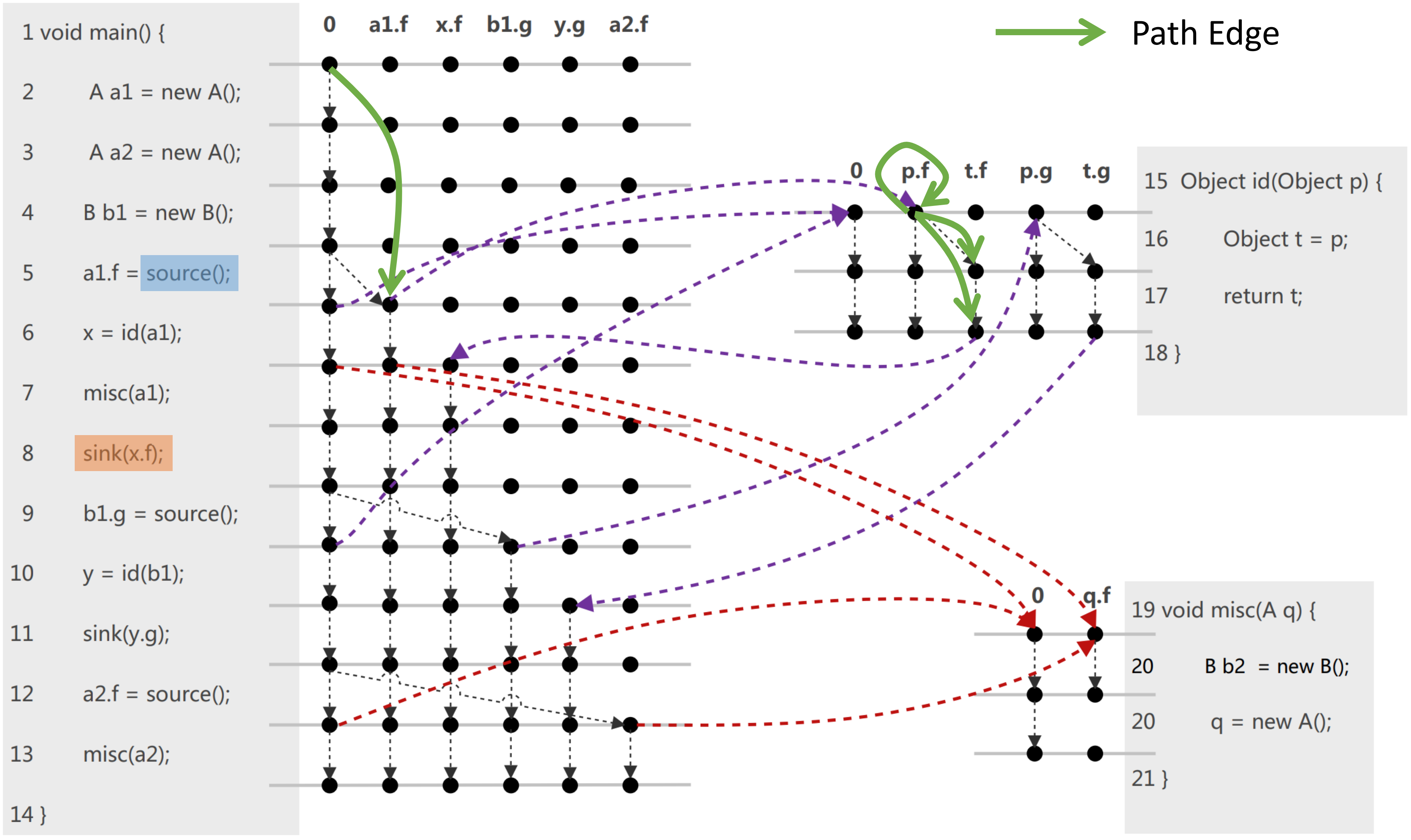


```
1 void main() {
2       A a1 = new A();
3       A a2 = new A();
4       B b1 = new B();
5       a1.f = source();
6       x = id(a1);
7       misc(a1);
8       sink(x.f);
9       b1.g = source();
10      y = id(b1);
11      sink(y.g);
12      a2.f = source();
13      misc(a2);
14 }
```

```
15  Object id(Object p) {
16        Object t = p;
17        return t;
18 }
```

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

A Running Example: Taint Analysis

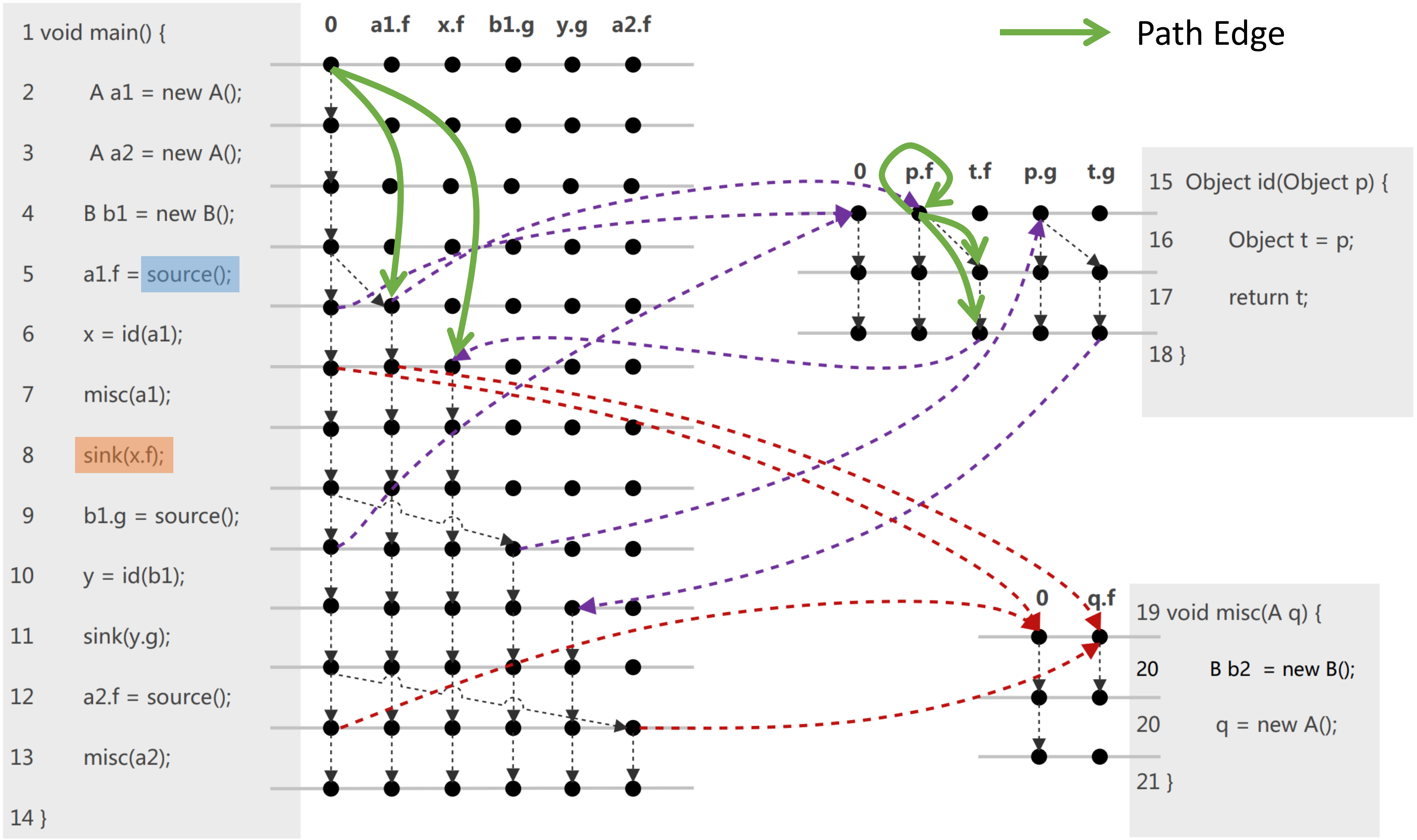


```
void main() {
    A a1 = new A();
    A a2 = new A();
    B b1 = new B();
    a1.f = source();
    x = id(a1);
    misc(a1);
    sink(x.f);
    b1.g = source();
    y = id(b1);
    sink(y.g);
    a2.f = source();
    misc(a2);
}
```

```
Object id(Object p) {
    Object t = p;
    return t;
}
```

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

A Running Example: Taint Analysis

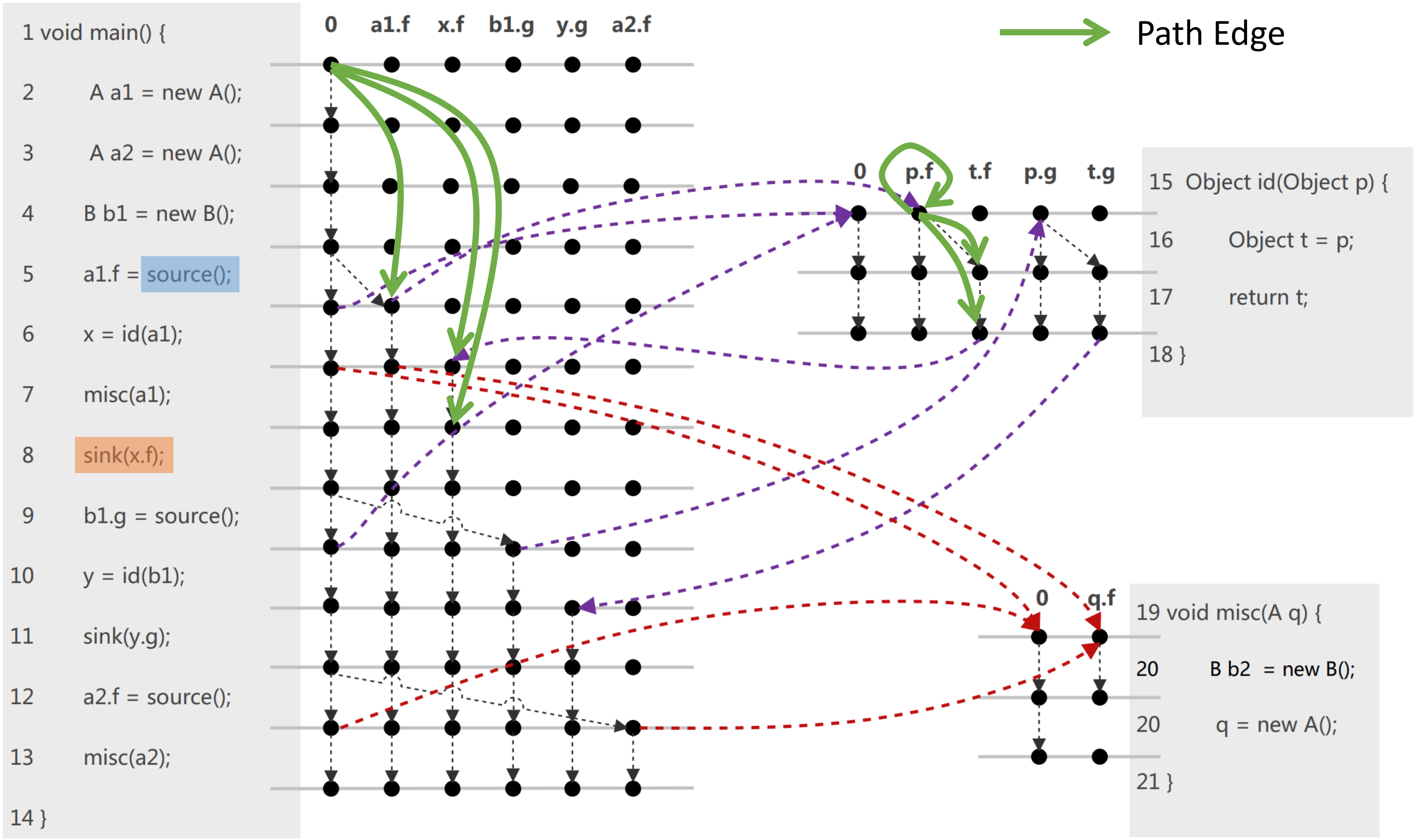


```
void main() {
    A a1 = new A();
    A a2 = new A();
    B b1 = new B();
    a1.f = source();
    x = id(a1);
    misc(a1);
    sink(x.f);
    b1.g = source();
    y = id(b1);
    sink(y.g);
    a2.f = source();
    misc(a2);
}
```

```
Object id(Object p) {
    Object t = p;
    return t;
}
```

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

A Running Example: Taint Analysis

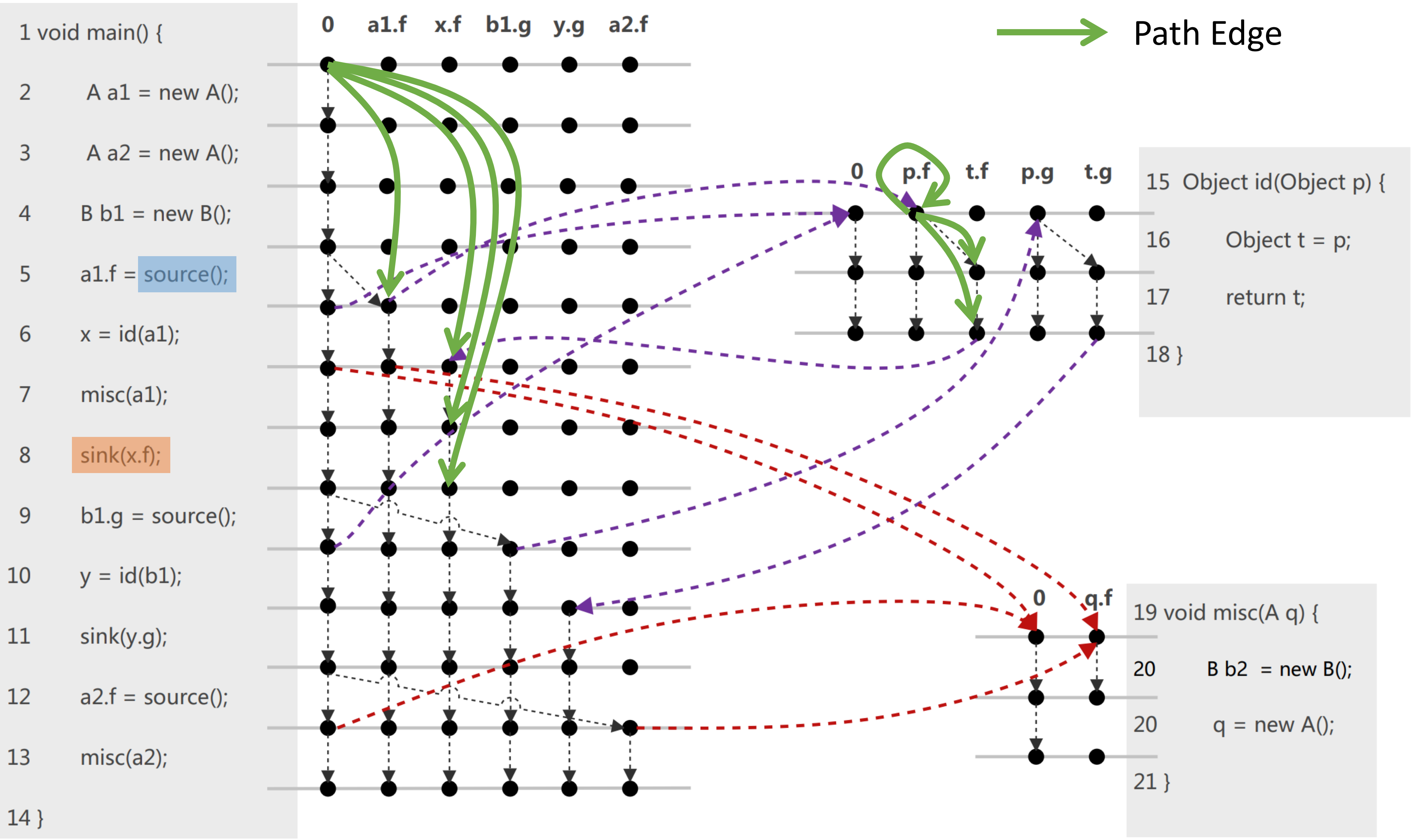


```
void main() {
    A a1 = new A();
    A a2 = new A();
    B b1 = new B();
    a1.f = source();
    x = id(a1);
    misc(a1);
    sink(x.f);
    b1.g = source();
    y = id(b1);
    sink(y.g);
    a2.f = source();
    misc(a2);
}
```

```
Object id(Object p) {
    Object t = p;
    return t;
}
```

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

A Running Example: Taint Analysis



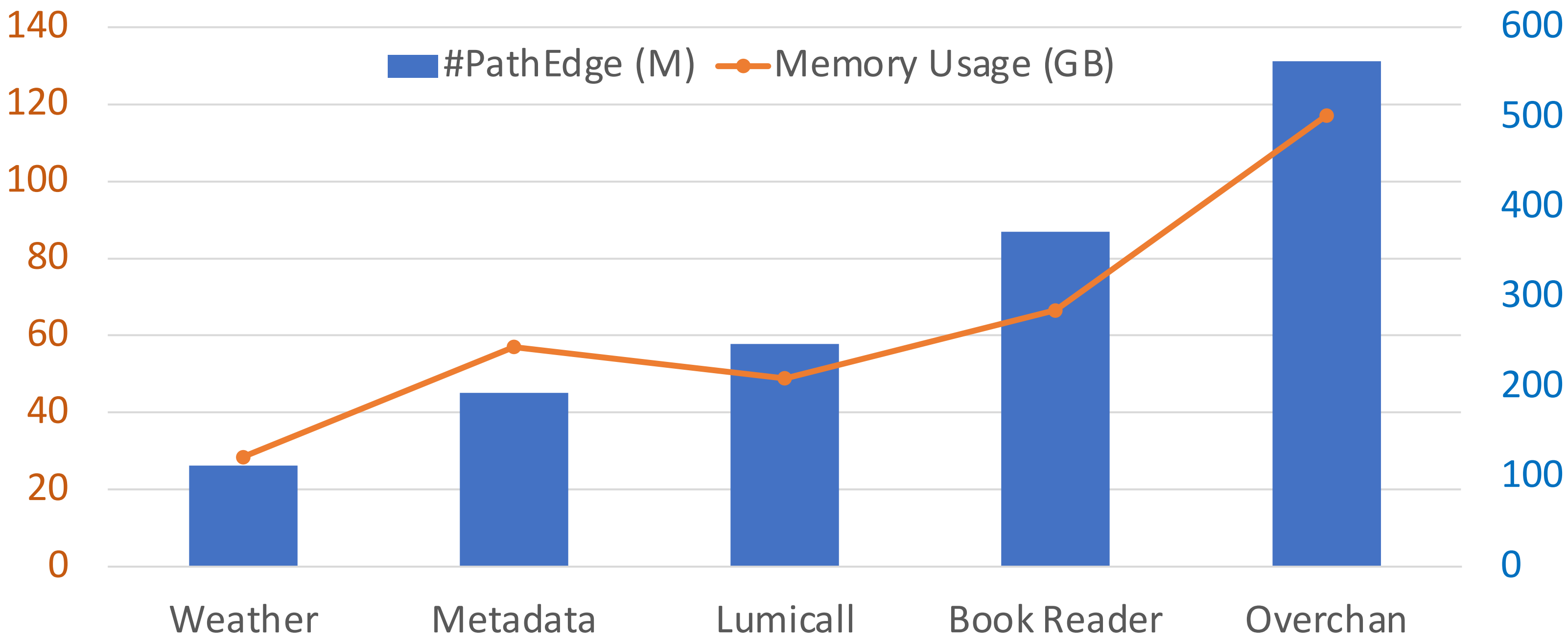
```
void main() {
    A a1 = new A();
    A a2 = new A();
    B b1 = new B();
    a1.f = source();
    x = id(a1);
    misc(a1);
    sink(x.f);
    b1.g = source();
    y = id(b1);
    sink(y.g);
    a2.f = source();
    misc(a2);
}
```

```
Object id(Object p) {
    Object t = p;
    return t;
}
```

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

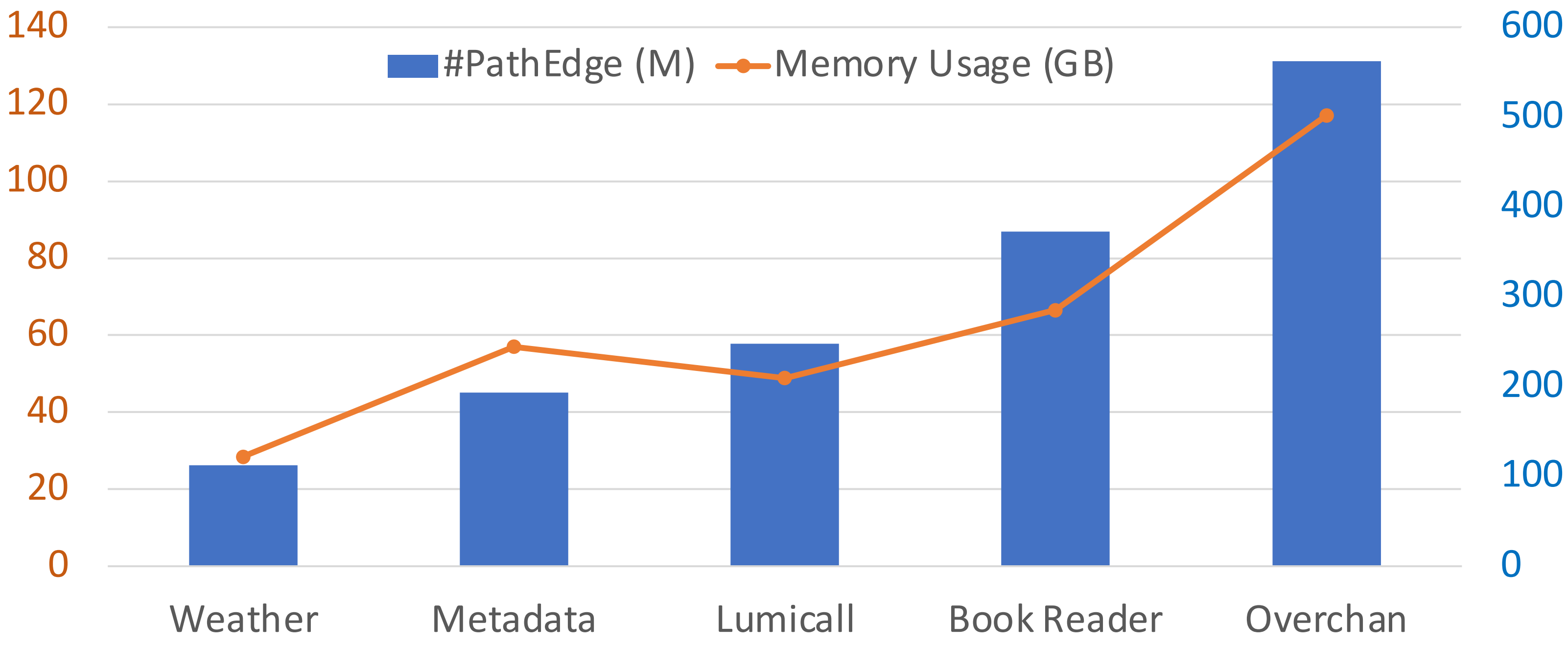
The Drawback: Memory-Intensive

- 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

Objective

Improving the Scalability and Efficiency

- Discovers facts only at some program points
  - Taint analysis

Objective

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

Objective

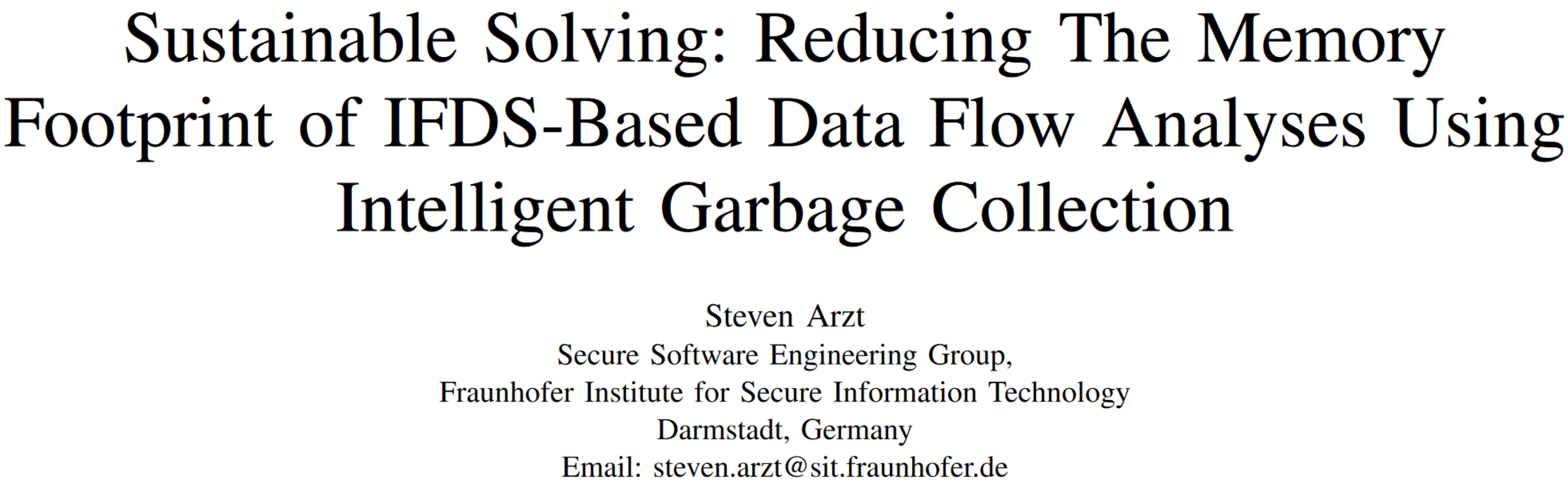
Improving the Scalability and Efficiency

- 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

Existing Approach



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

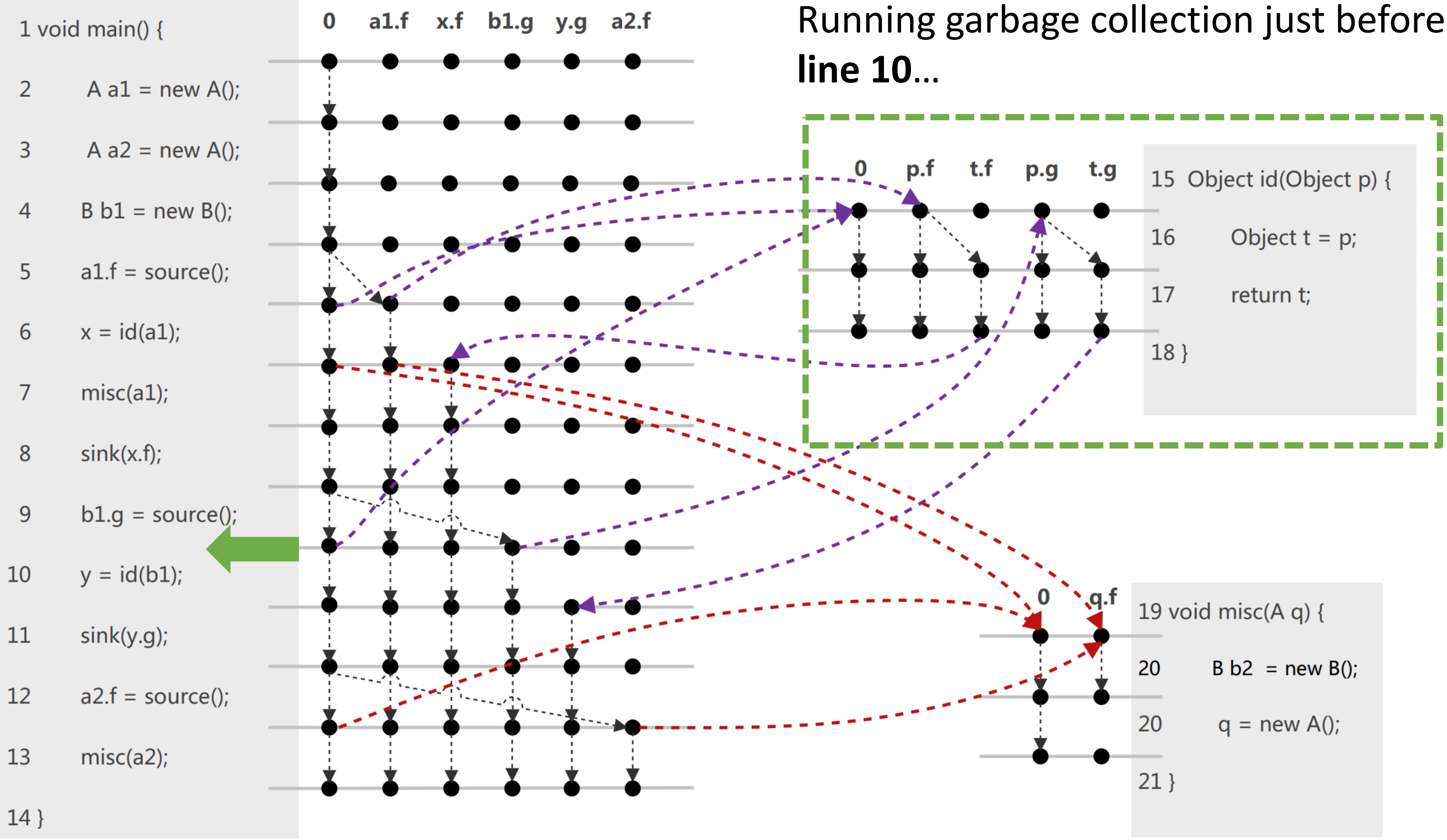
Steven Arzt
Secure Software Engineering Group,
Fraunhofer Institute for Secure Information Technology
Darmstadt, Germany
Email: steven.arzt@sit.fraunhofer.de

CleanDroid (ICSE’21)

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

Limitation 1: Coarse Granularity

Running garbage collection just before **line 10…**



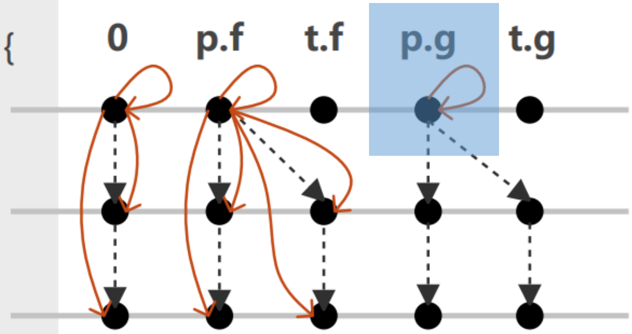
```
1 void main() {
2     A a1 = new A();
3     A a2 = new A();
4     B b1 = new B();
5     a1.f = source();
6     x = id(a1);
7     misc(a1);
8     sink(x.f);
9     b1.g = source();
10    y = id(b1);
11    sink(y.g);
12    a2.f = source();
13    misc(a2);
14 }
```

```
15 Object id(Object p) {
16     Object t = p;
17     return t;
18 }
```

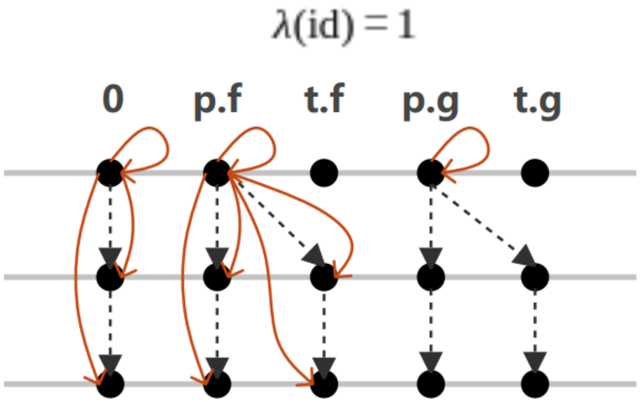
```
19 void misc(A q) {
20     B b2  = new B();
20     q = new A();
21 }
```

Limitation 1: Coarse Granularity

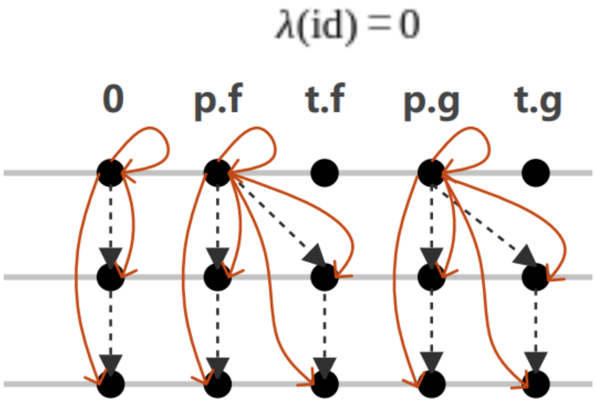
```
Object id(Object p) {
        Object t = p;
        return t;
}
```



(a) before GC



(b) after GC

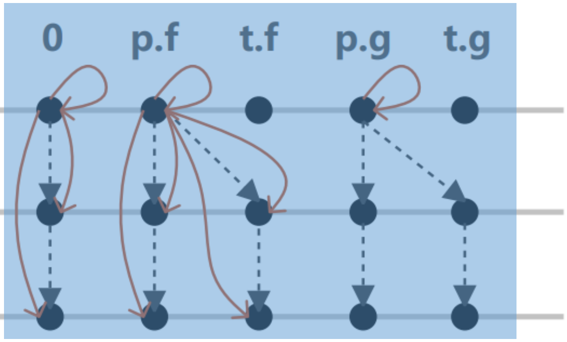


(c) after the analysis

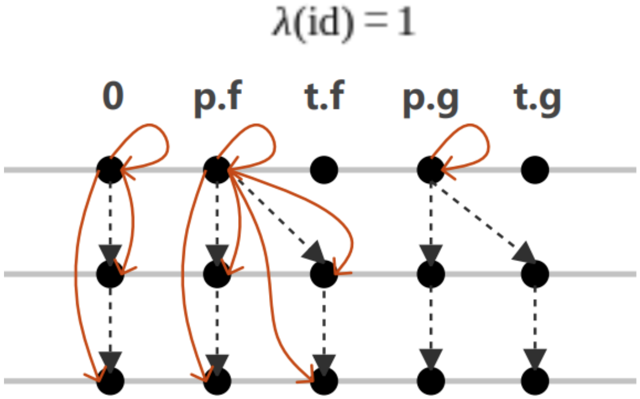
Maximum #PathEdge maintained: 13.

Limitation 1: Coarse Granularity

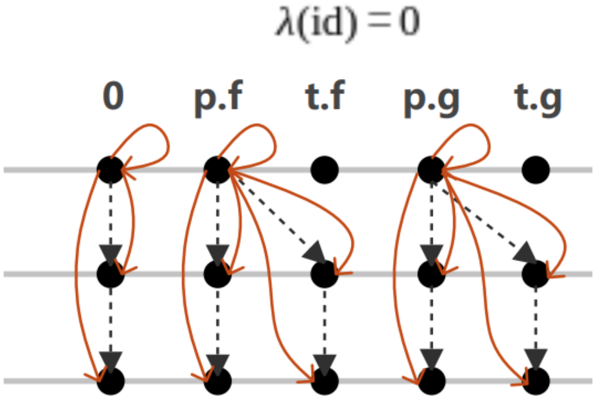
```
Object id(Object p) {
        Object t = p;
        return t;
}
```



(a) before GC



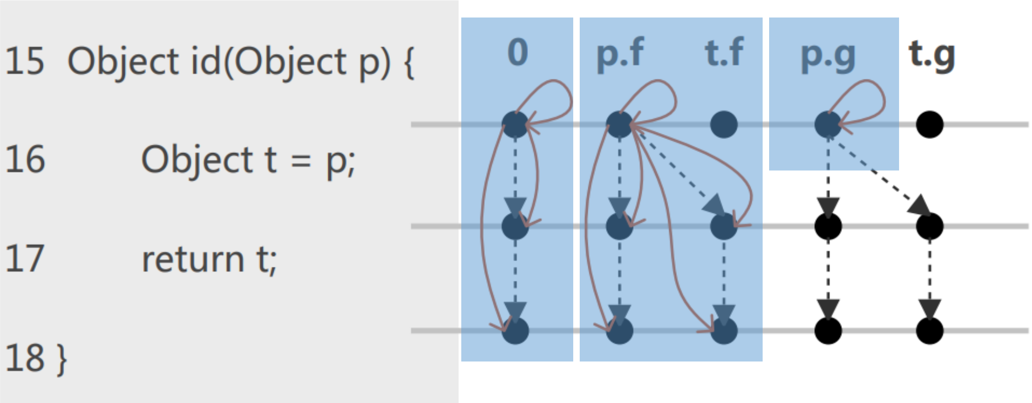
(b) after GC



(c) after the analysis

Method-Level: Coarse-Grained

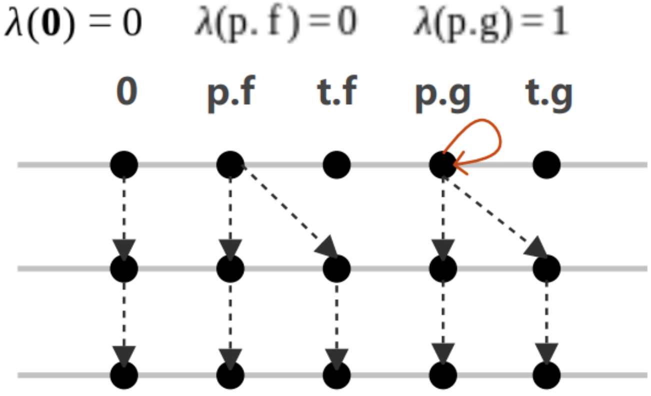
Fine-Grained Data-Fact-Level GC



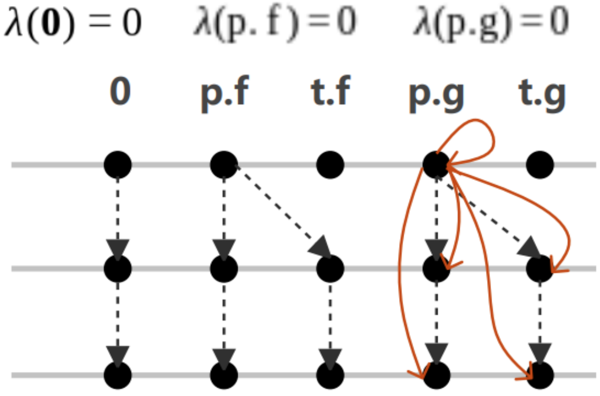
```
Object id(Object p) {
        Object t = p;
        return t;
}
```

(a) before GC

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



(b) after GC

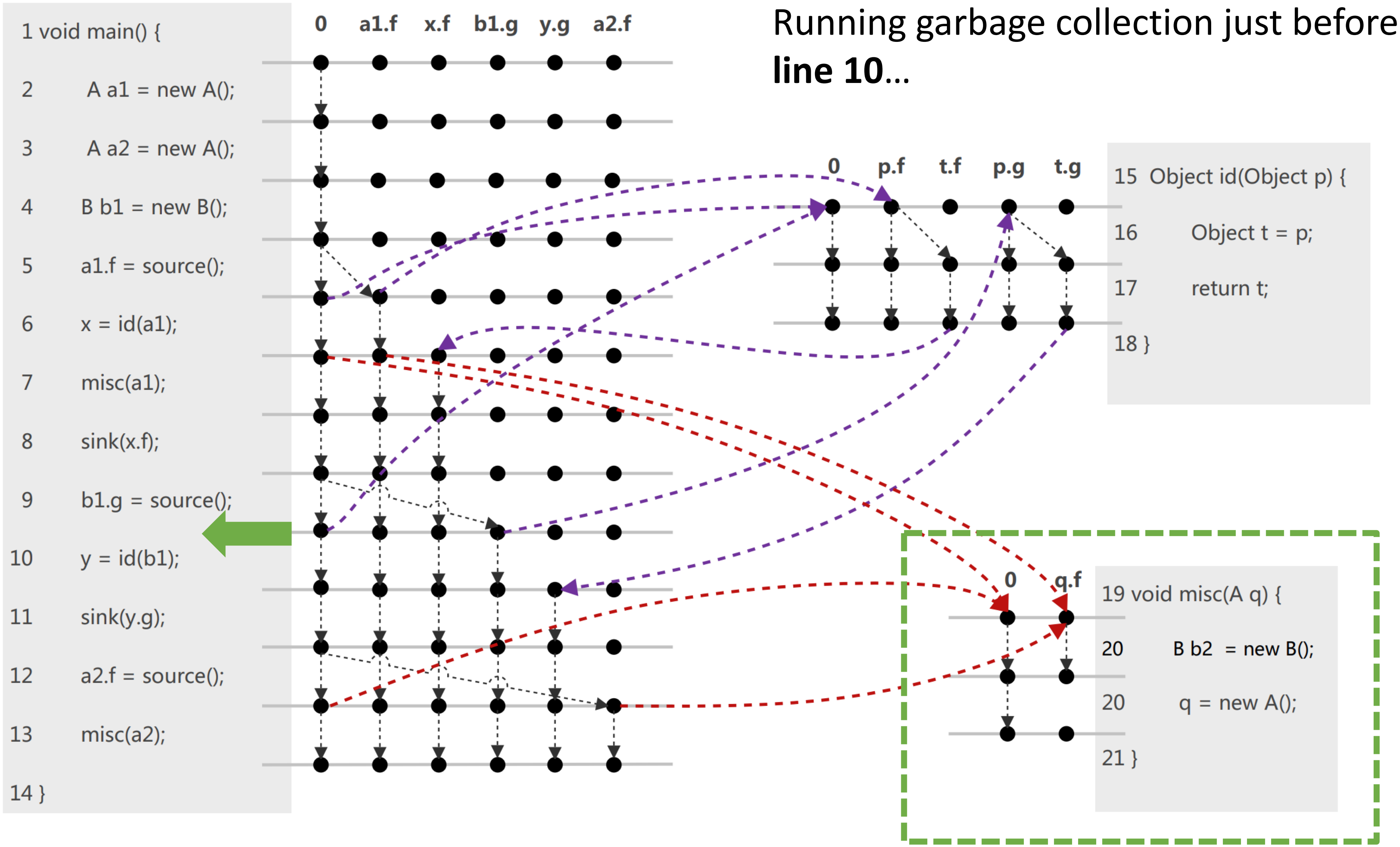


(c) after the analysis

Maximum #PathEdge maintained: 9.

Limitation 2: Redundant Computations

Running garbage collection just before **line 10…**



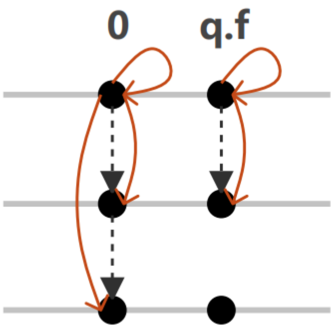
```
1 void main() {
2     A a1 = new A();
3     A a2 = new A();
4     B b1 = new B();
5     a1.f = source();
6     x = id(a1);
7     misc(a1);
8     sink(x.f);
9     b1.g = source();
10    y = id(b1);
11    sink(y.g);
12    a2.f = source();
13    misc(a2);
14 }
```

```
15 Object id(Object p) {
16     Object t = p;
17     return t;
18 }
```

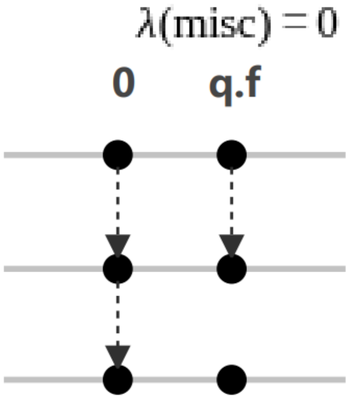
```
19 void misc(A q) {
20     B b2 = new B();
20     q = new A();
21 }
```

Limitation 2: Redundant Computations

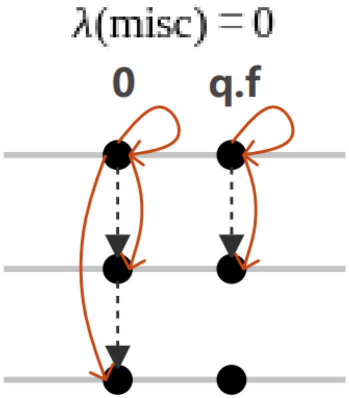
```
void misc(A q) {
      B b2  = new B();
       q = new A();
}
```



(a) before GC



(b) after GC

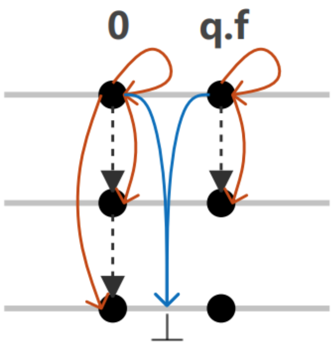


(c) after the analysis

The method `misc()` is analyzed redundantly.

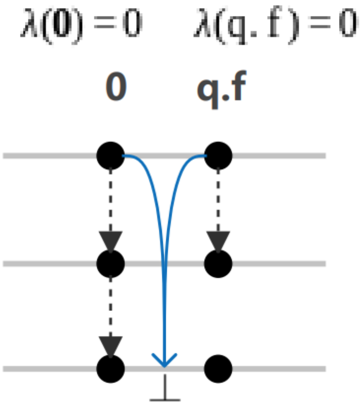
Avoiding Redundant Computations

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

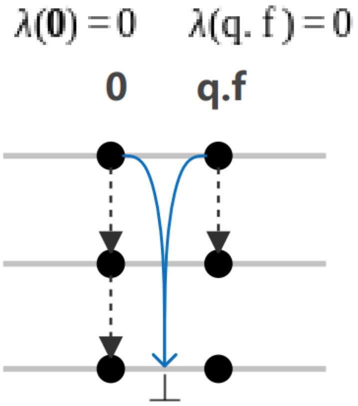


(a) before GC

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



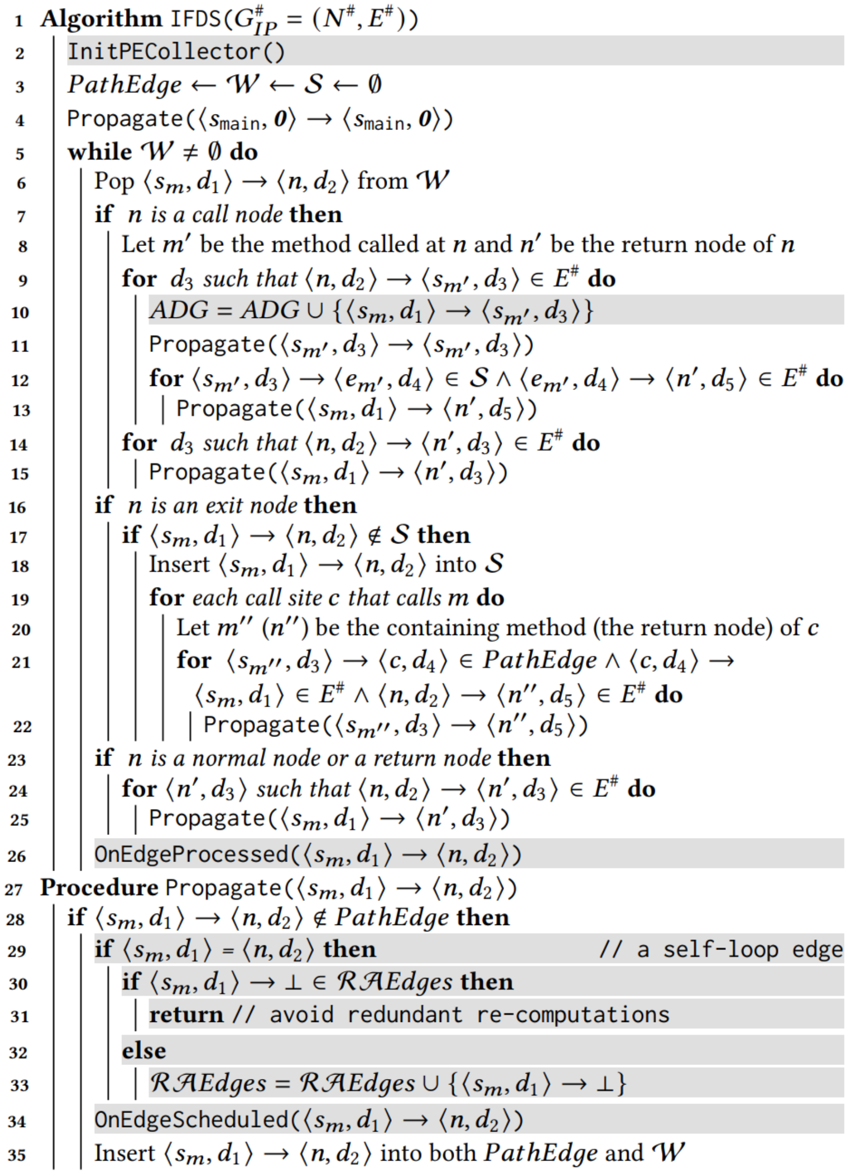
(b) after GC



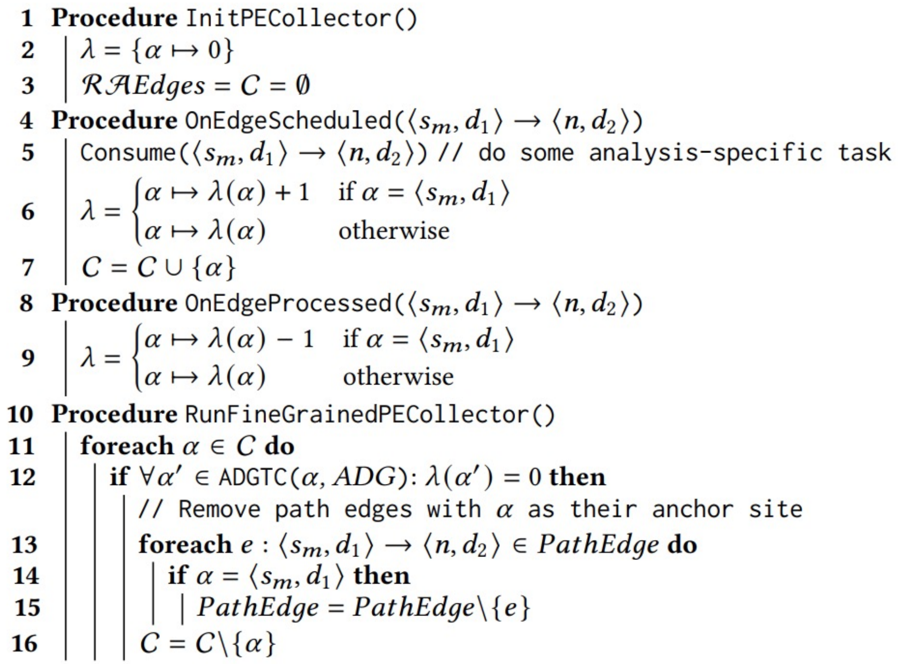
(c) after the analysis

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

Data-Fact-Level Path-Edge GC Algorithm



```
Algorithm IFDS($G^{\#}_{IP} = (N^{\#}, E^{\#})$)
    InitPECollector()
    $PathEdge \leftarrow \mathcal{W} \leftarrow \mathcal{S} \leftarrow \emptyset$
    Propagate($\langle s_{\text{main}}, \mathbf{0} \rangle \rightarrow \langle s_{\text{main}}, \mathbf{0} \rangle$)
    while $\mathcal{W} \neq \emptyset$ do
        Pop $\langle s_m, d_1 \rangle \rightarrow \langle n, d_2 \rangle$ from $\mathcal{W}$
        if $n$ is a call node then
            Let $m'$ be the method called at $n$ and $n'$ be the return node of $n$
            for $d_3$ such that $\langle n, d_2 \rangle \rightarrow \langle s_{m'}, d_3 \rangle \in E^{\#}$ do
                $ADG = ADG \cup \{\langle s_m, d_1 \rangle \rightarrow \langle s_{m'}, d_3 \rangle\}$
                Propagate($\langle s_{m'}, d_3 \rangle \rightarrow \langle s_{m'}, d_3 \rangle$)
                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
                    Propagate($\langle s_m, d_1 \rangle \rightarrow \langle n', d_5 \rangle$)
            for $d_3$ such that $\langle n, d_2 \rangle \rightarrow \langle n', d_3 \rangle \in E^{\#}$ do
                Propagate($\langle s_m, d_1 \rangle \rightarrow \langle n', d_3 \rangle$)
        if $n$ is an exit node then
            if $\langle s_m, d_1 \rangle \rightarrow \langle n, d_2 \rangle \notin \mathcal{S}$ then
                Insert $\langle s_m, d_1 \rangle \rightarrow \langle n, d_2 \rangle$ into $\mathcal{S}$
                for each call site $c$ that calls $m$ do
                    Let $m''$ ($n''$) be the containing method (the return node) of $c$
                    for $\langle s_{m''}, d_3 \rangle \rightarrow \langle c, d_4 \rangle \in PathEdge \wedge \langle c, d_4 \rangle \rightarrow \langle s_m, d_1 \rangle \in E^{\#} \wedge \langle n, d_2 \rangle \rightarrow \langle n'', d_5 \rangle \in E^{\#}$ do
                        Propagate($\langle s_{m''}, d_3 \rangle \rightarrow \langle n'', d_5 \rangle$)
        if $n$ is a normal node or a return node then
            for $\langle n', d_3 \rangle$ such that $\langle n, d_2 \rangle \rightarrow \langle n', d_3 \rangle \in E^{\#}$ do
                Propagate($\langle s_m, d_1 \rangle \rightarrow \langle n', d_3 \rangle$)
        OnEdgeProcessed($\langle s_m, d_1 \rangle \rightarrow \langle n, d_2 \rangle$)
Procedure Propagate($\langle s_m, d_1 \rangle \rightarrow \langle n, d_2 \rangle$)
    if $\langle s_m, d_1 \rangle \rightarrow \langle n, d_2 \rangle \notin PathEdge$ then
        if $\langle s_m, d_1 \rangle = \langle n, d_2 \rangle$ then                // a self-loop edge
            if $\langle s_m, d_1 \rangle \rightarrow \bot \in \mathcal{RAEdges}$ then
                return // avoid redundant re-computations
            else
                $\mathcal{RAEdges} = \mathcal{RAEdges} \cup \{\langle s_m, d_1 \rangle \rightarrow \bot\}$
        OnEdgeScheduled($\langle s_m, d_1 \rangle \rightarrow \langle n, d_2 \rangle$)
        Insert $\langle s_m, d_1 \rangle \rightarrow \langle n, d_2 \rangle$ into both $PathEdge$ and $\mathcal{W}$
```



```
Procedure InitPECollector()
    $\lambda = \{\alpha \mapsto 0\}$
    $\mathcal{RAEdges} = C = \emptyset$
Procedure OnEdgeScheduled($\langle s_m, d_1 \rangle \rightarrow \langle n, d_2 \rangle$)
    Consume($\langle s_m, d_1 \rangle \rightarrow \langle n, d_2 \rangle$) // do some analysis-specific task
    $\lambda = \begin{cases} \alpha \mapsto \lambda(\alpha) + 1 & \text{if } \alpha = \langle s_m, d_1 \rangle \\ \alpha \mapsto \lambda(\alpha) & \text{otherwise} \end{cases}$
    $C = C \cup \{\alpha\}$
Procedure OnEdgeProcessed($\langle s_m, d_1 \rangle \rightarrow \langle n, d_2 \rangle$)
    $\lambda = \begin{cases} \alpha \mapsto \lambda(\alpha) - 1 & \text{if } \alpha = \langle s_m, d_1 \rangle \\ \alpha \mapsto \lambda(\alpha) & \text{otherwise} \end{cases}$
Procedure RunFineGrainedPECollector()
    foreach $\alpha \in C$ do
        if $\forall \alpha' \in$ ADGTC($\alpha, ADG$): $\lambda(\alpha') = 0$ then
            // Remove path edges with $\alpha$ as their anchor site
            foreach $e : \langle s_m, d_1 \rangle \rightarrow \langle n, d_2 \rangle \in PathEdge$ do
                if $\alpha = \langle s_m, d_1 \rangle$ then
                    $PathEdge = PathEdge \backslash \{e\}$
            $C = C \backslash \{\alpha\}$
```

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

Artifacts Available acm V1.1

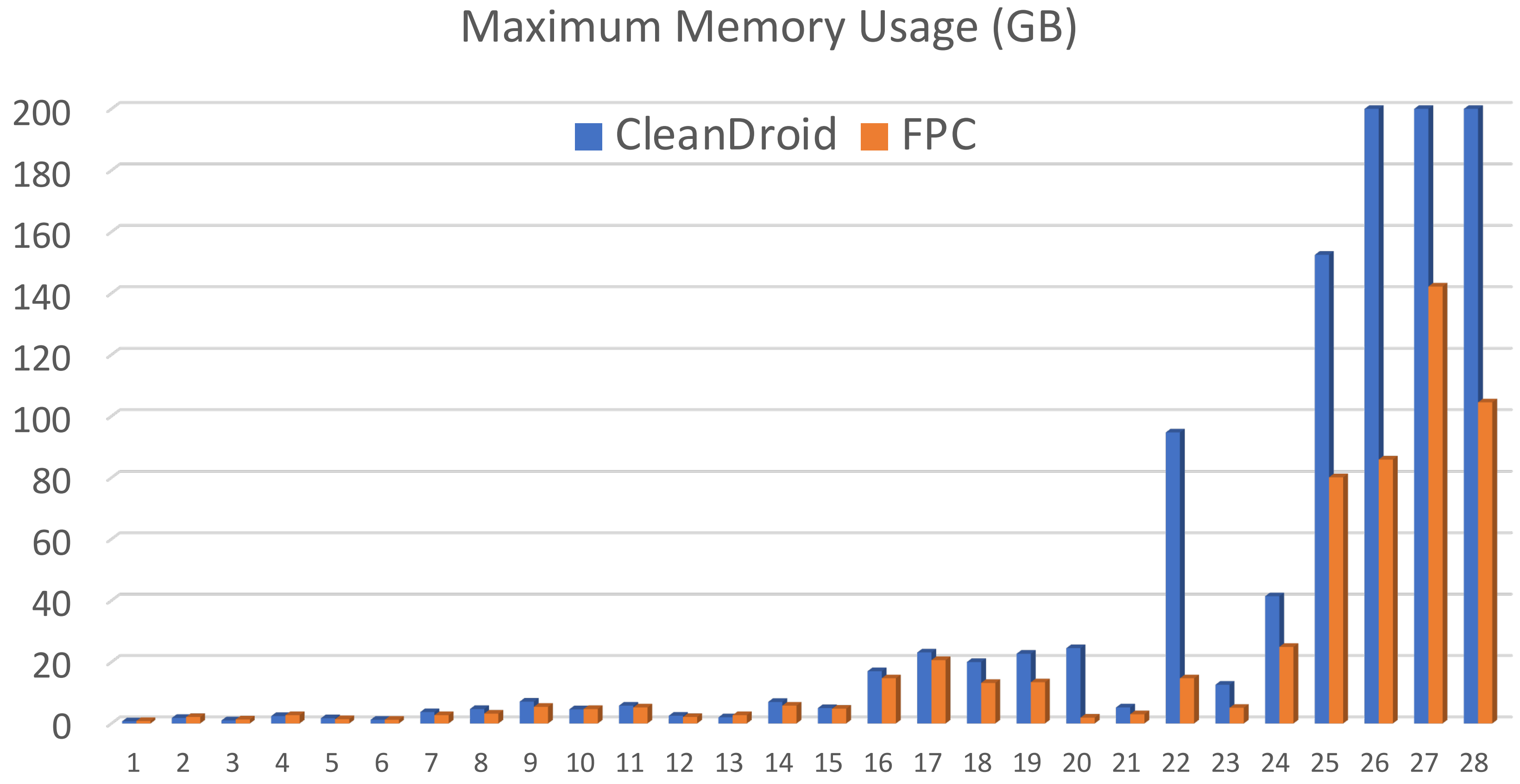
Artifacts Evaluated acm Reusable V1.1

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



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

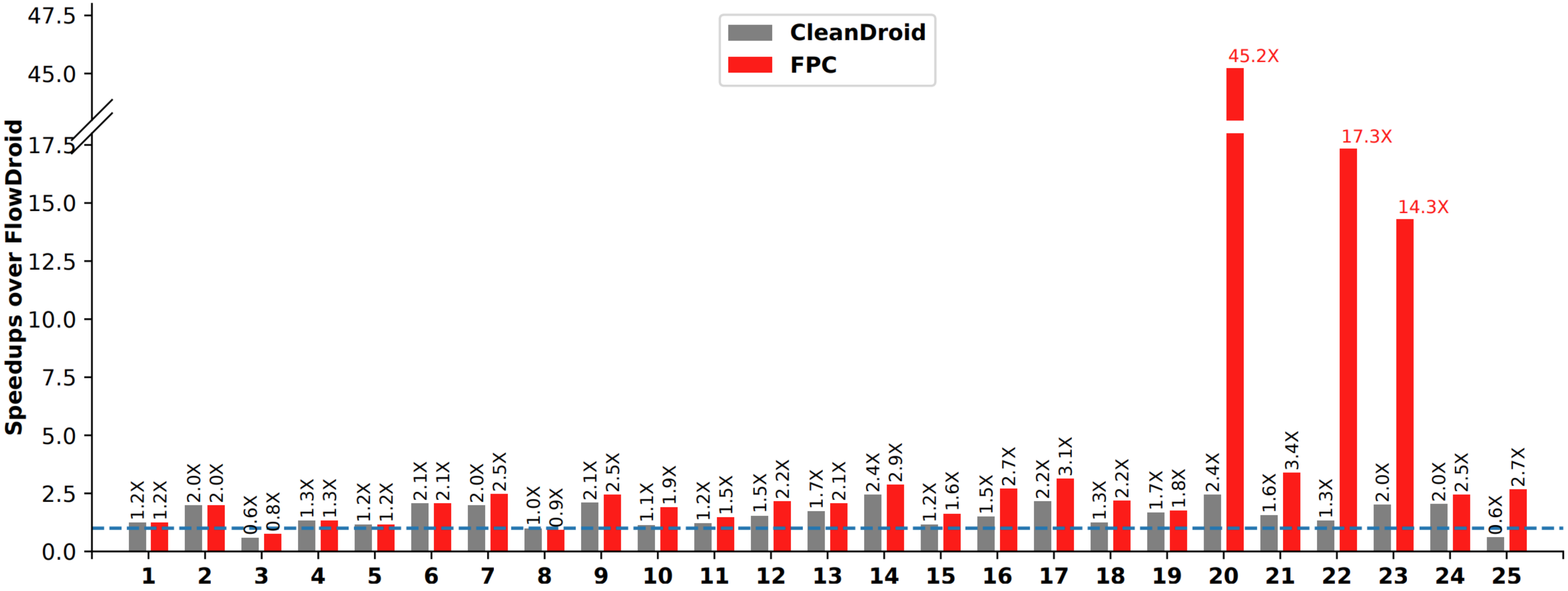
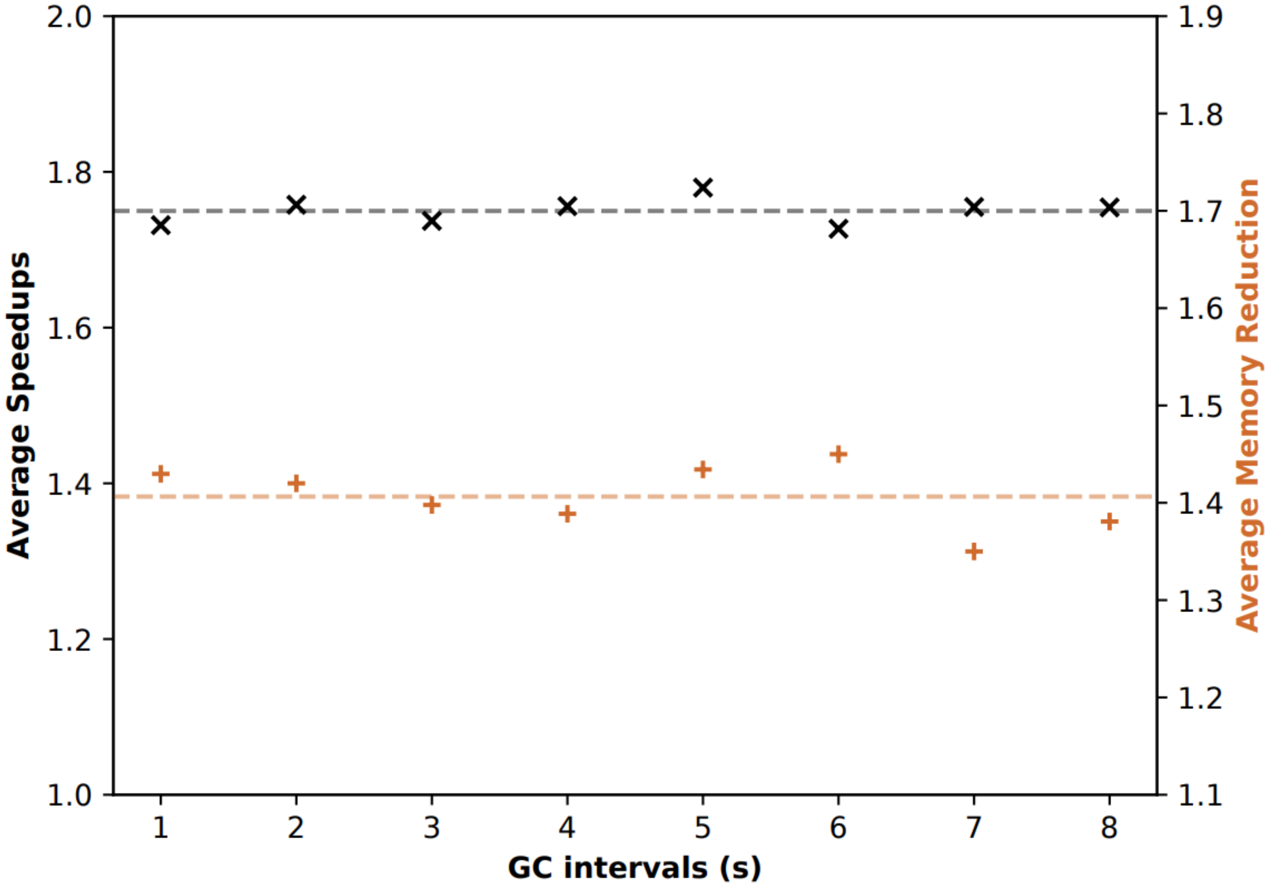


Figure 9: Comparing the speedups of FPC and CleanDroid with FlowDroid as the baseline.

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

RQ3: Varying GC Intervals



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

Q & A

Thank you!

Dongjie He: dongjieh@cse.unsw.edu.au
Yujiang Gui: yujiang.gui@unsw.edu.au