Scaling Up the IFDS Algorithm with Efficient Disk-Assisted Computing

Haofeng Li[1,2], Haining Meng[1,2], Hengjie Zheng[1,2], Liqing Cao[1,2], Jie Lu[1,2],

Lian Li[*,1,2] and Lin Gao[3]

[1]State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China

[2]University of Chinese Academy of Sciences, Beijing, China

[3]TianqiSoft Inc, China

Institute of Computing Technology, Chinese academy of sciences

University of Chinese Academy of Sciences

TianqiSoft Inc

IFDS

- The IFDS analysis framework solves inter-procedural, context-sensitive, and flow-sensitive analysis of finite distribute subset problems
- Reps et al. [1] transforming IFDS problems into a special kind of graph-reachability problem

- *[1]Thomas Reps, Susan Horwitz, and Mooly Sagiv. 1995. Precise interprocedural dataflow analysis via graph reachability. In Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '95). Association for Computing Machinery, New York, NY, USA, 49–61. DOI:https://doi.org/10.1145/199448.199462*

Applications

- IFDS can be used to solve a wide range of problems:
  - pointer analysis
  - taint analysis
  - slicing
  - bug detection
  - shape analysis
  - …

WALA

T. J. WATSON LIBRARIES FOR ANALYSIS

Problem

- IFDS algorithm is memory-intensive
- Space complexity: $O(ED^2)$
  - E represents edges in the graph
  - D represents a set of data-flow facts
- 16/2,950 Android apps are unanalyzable on a computer server with 730GB of RAM and 64 Intel Xeon CPU cores[1]

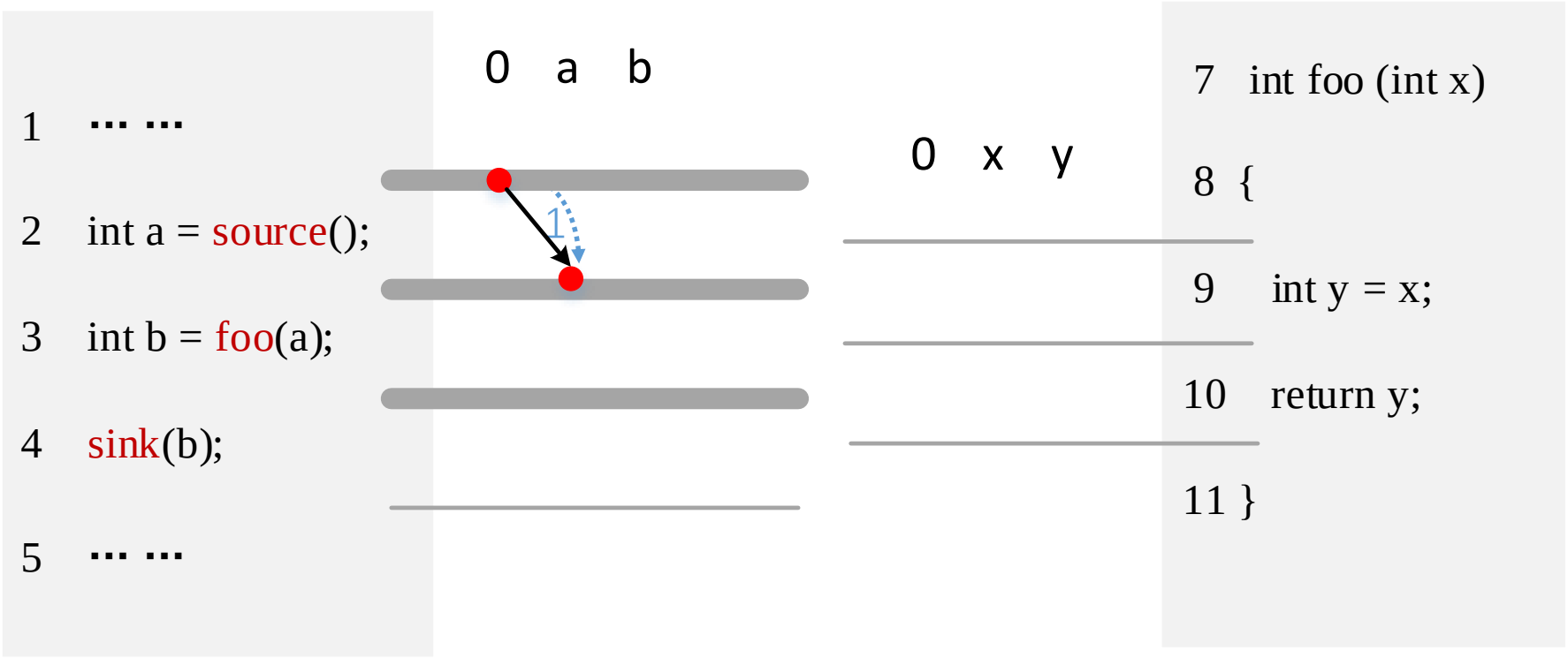
[1]V. Avdiienko, K. Kuznetsov, A. Gorla, A. Zeller, S. Arzt, S. Rasthofer, and E. Bodden, "Mining apps for abnormal usage of sensitive data," in Proceedings of the 37th International Conference on Software Engineering - Volume 1, ser. ICSE '15. IEEE Press, 2015, p. 426–436.

The key data structures of IFDS

- According to the extended algorithm of IFDS designed by Naeem et al. [1]
  - **PathEdge** records the set of *path edges*, representing a subset of the same-level realizable paths.
  - **Incoming** records the set of nodes $< s_p; d >$ reachable from $< s_0; \mathbf{0} >$, and their predecessors.
  - **EndSummary** records the set of path edges from entry node to exit node of a procedure

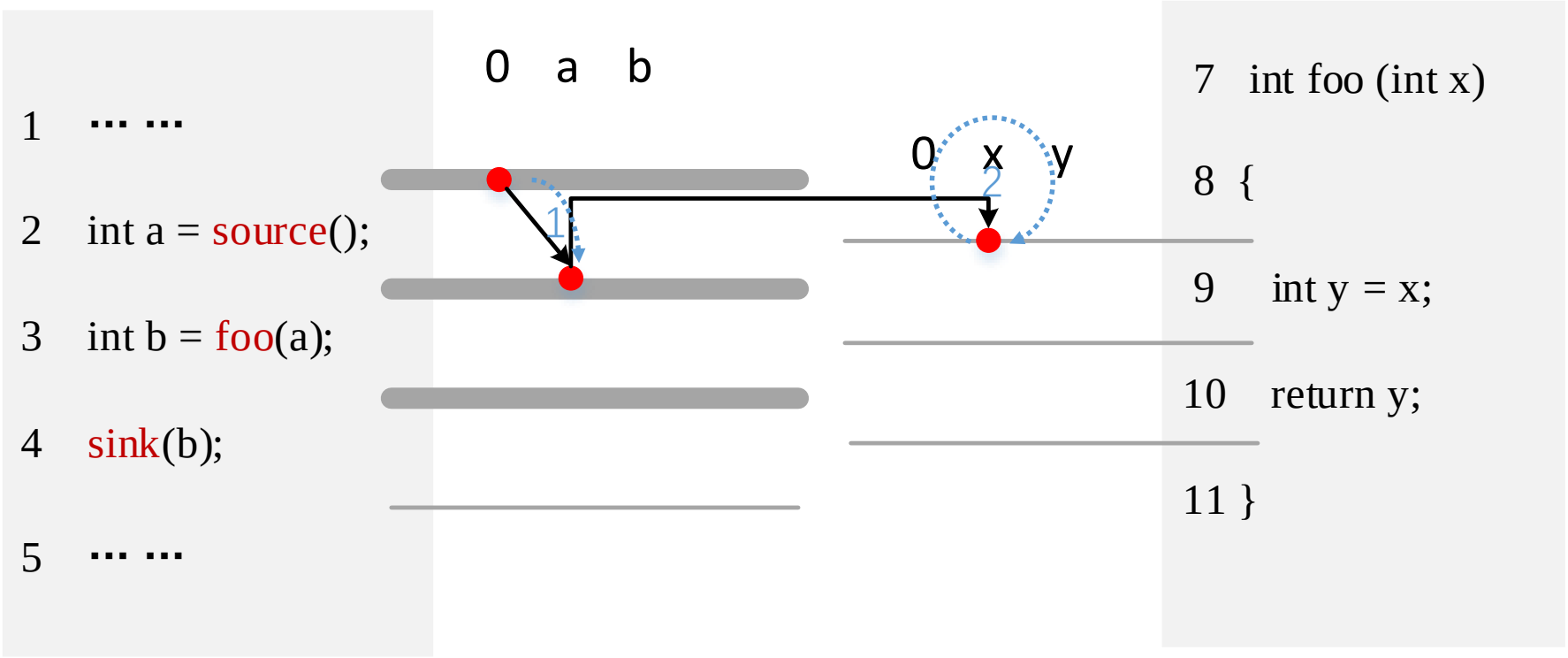
[1]Naeem N A, Lhoták O, Rodriguez J. Practical extensions to the IFDS algorithm[C]//International Conference on Compiler Construction. Springer, Berlin, Heidelberg, 2010: 124-144.

IFDS-based Taint Analysis (FlowDroid)



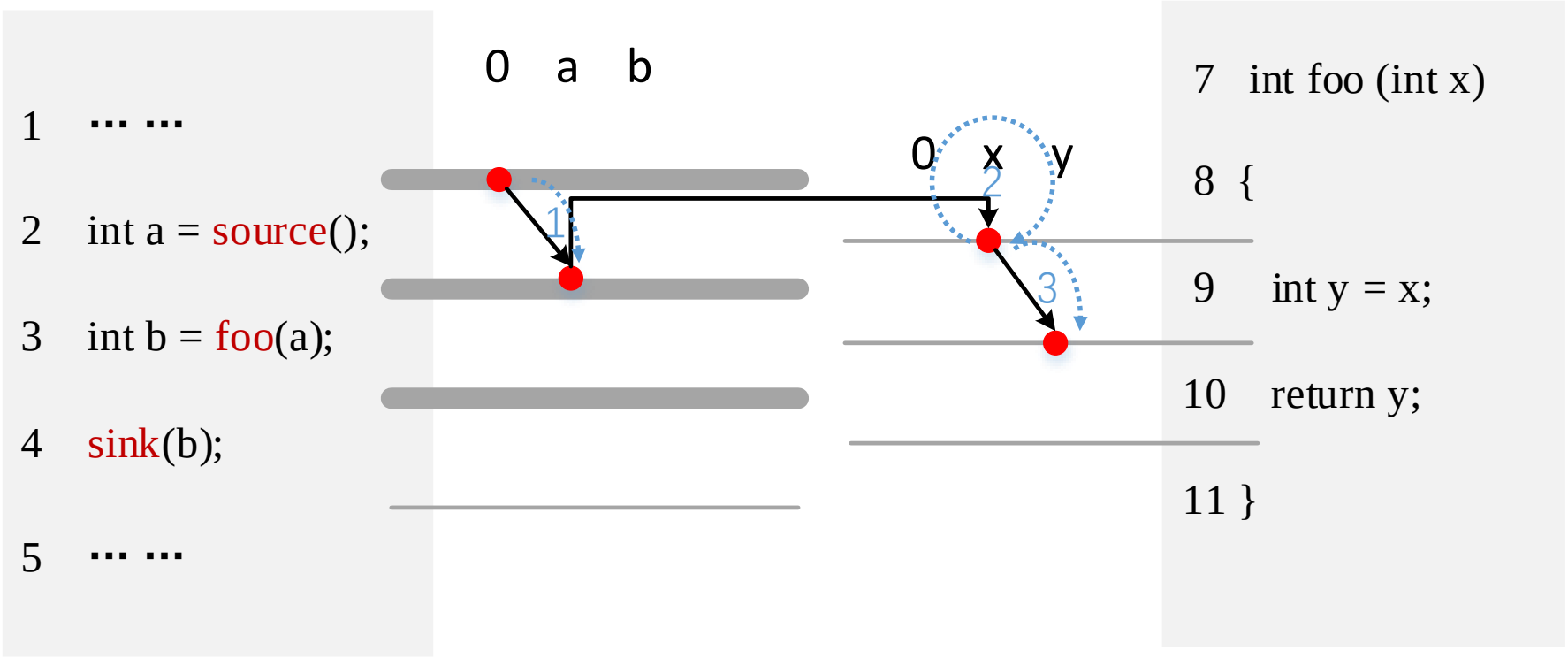
PathEdge: {1}
Incoming: { }
EndSummary: { }

IFDS-based Taint Analysis (FlowDroid)



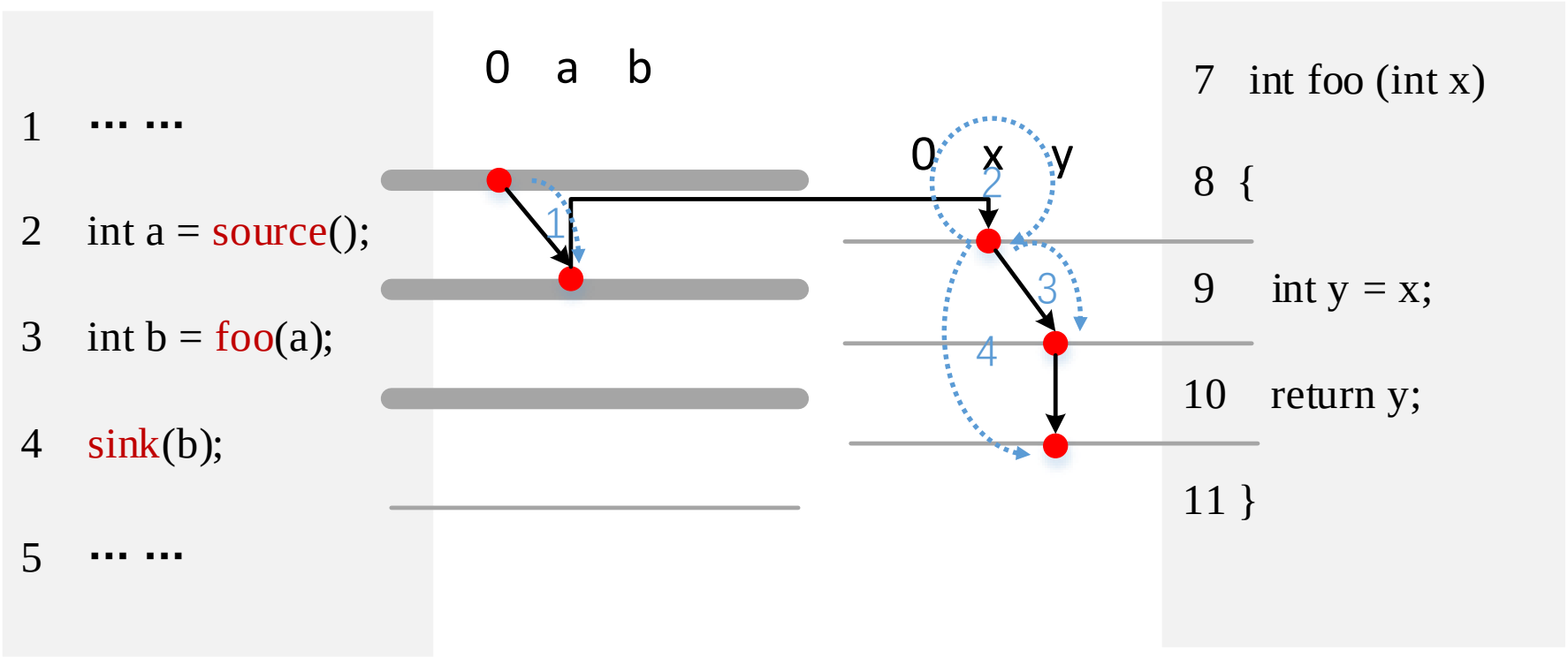
PathEdge: {1, 2}
Incoming: { [x:1]}
EndSummary: { }

IFDS-based Taint Analysis (FlowDroid)



PathEdge: {1, 2, 3}
Incoming: { [x:1]}
EndSummary: { }

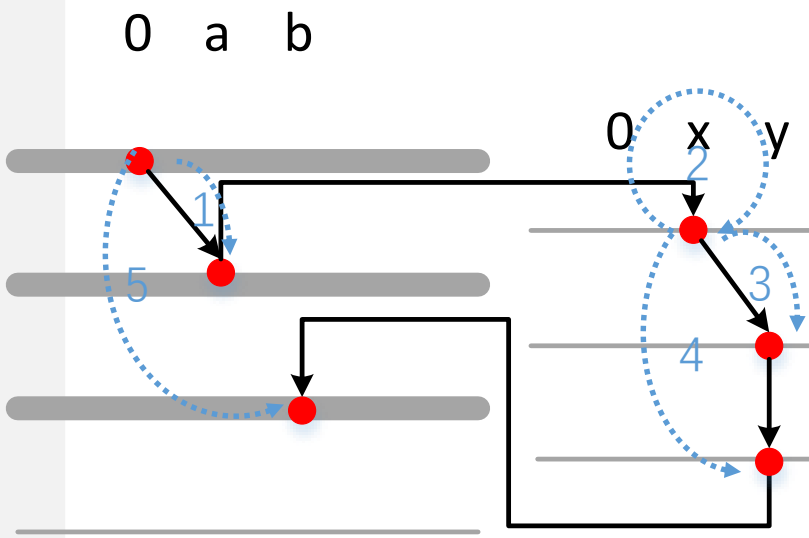
IFDS-based Taint Analysis (FlowDroid)



PathEdge: {1, 2, 3, 4}
Incoming: { [x:1] }
EndSummary: { [4] }

IFDS-based Taint Analysis (FlowDroid)

```
1   … …
2   int a = source();
3   int b = foo(a);
4   sink(b);
5   … …
```

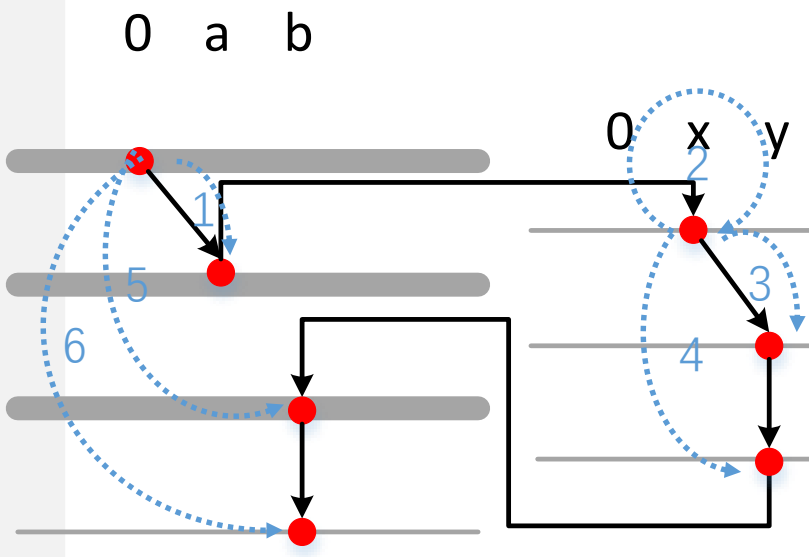



```
7   int foo (int x)
8   {
9     int y = x;
10    return y;
11  }
```

PathEdge: {1, 2, 3, 4, 5}
Incoming: { [x:1] }
EndSummary: { [4] }

IFDS-based Taint Analysis (FlowDroid)

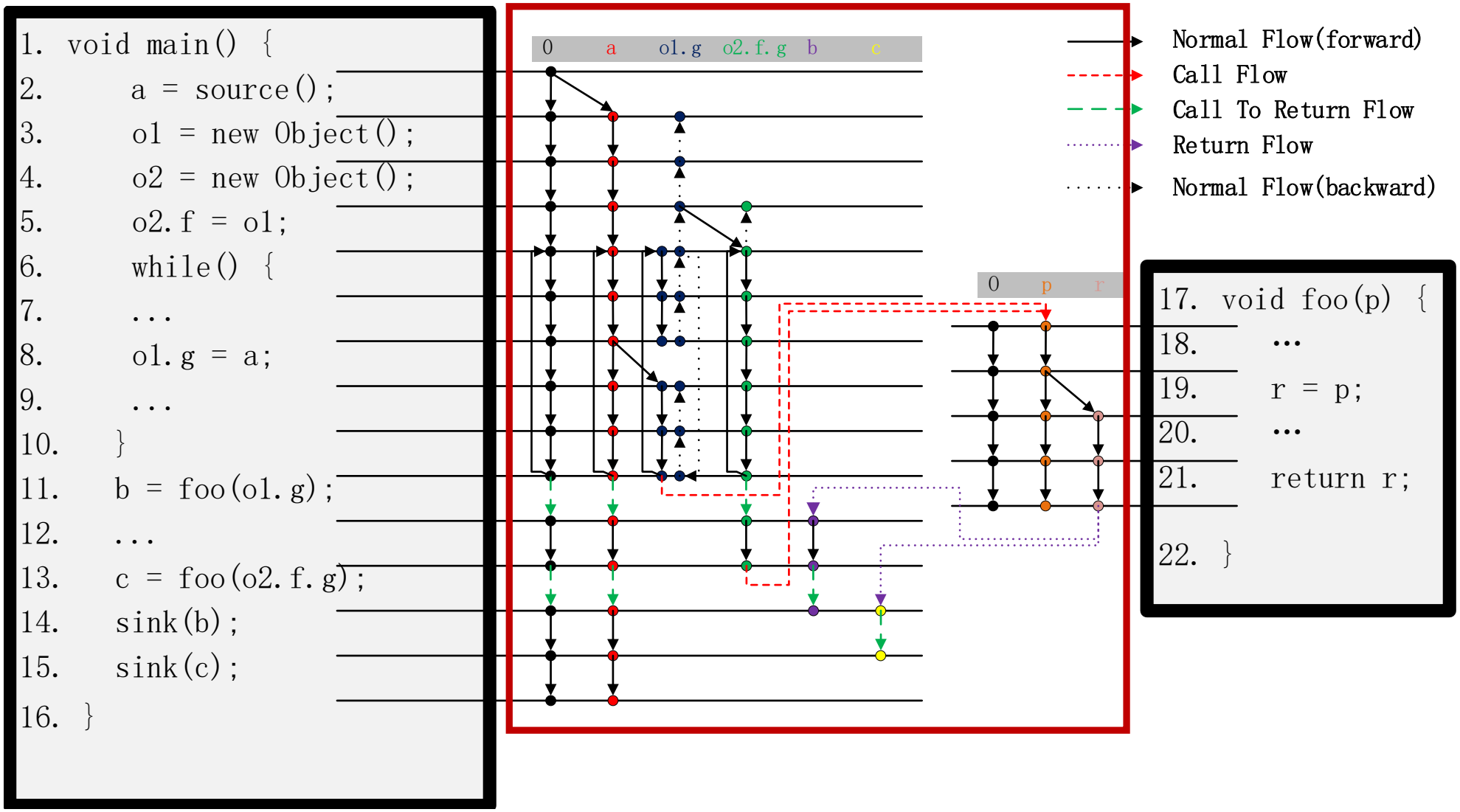
```
1   … …
2   int a = source();
3   int b = foo(a);
4   sink(b);
5   … …
```



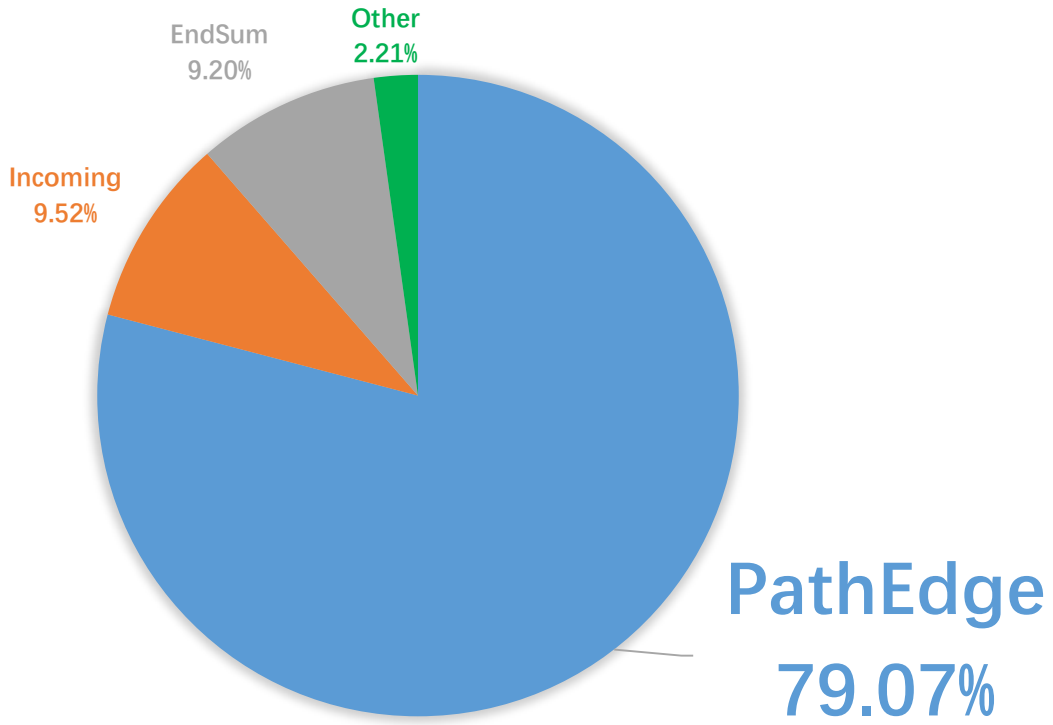

```
7   int foo (int x)
8   {
9       int y = x;
10      return y;
11  }
```

PathEdge: {1, 2, 3, 4, 5, 6}
Incoming: { [x:1] }
EndSummary: { [4] }

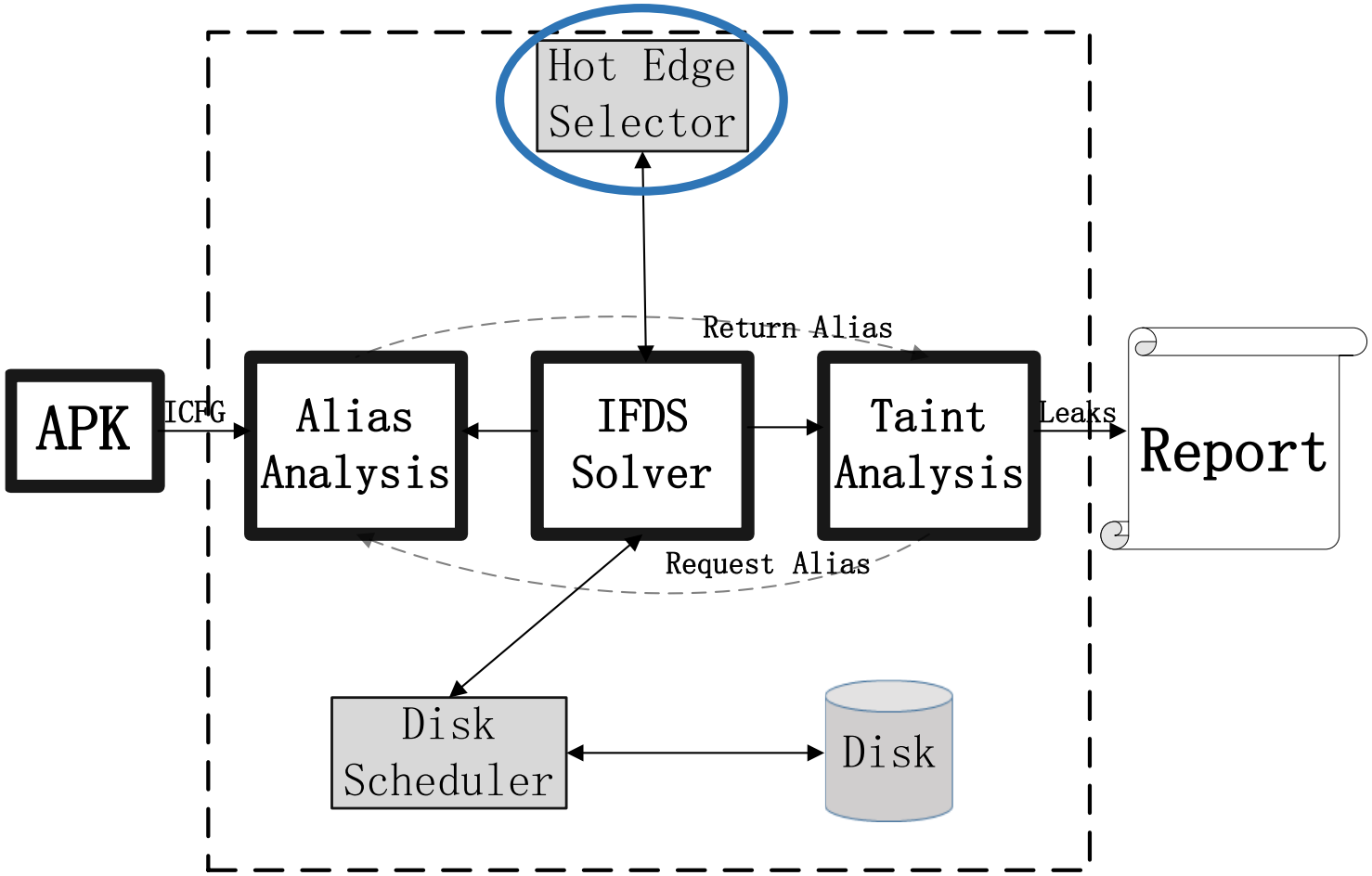
IFDS-based Taint Analysis (FlowDroid)



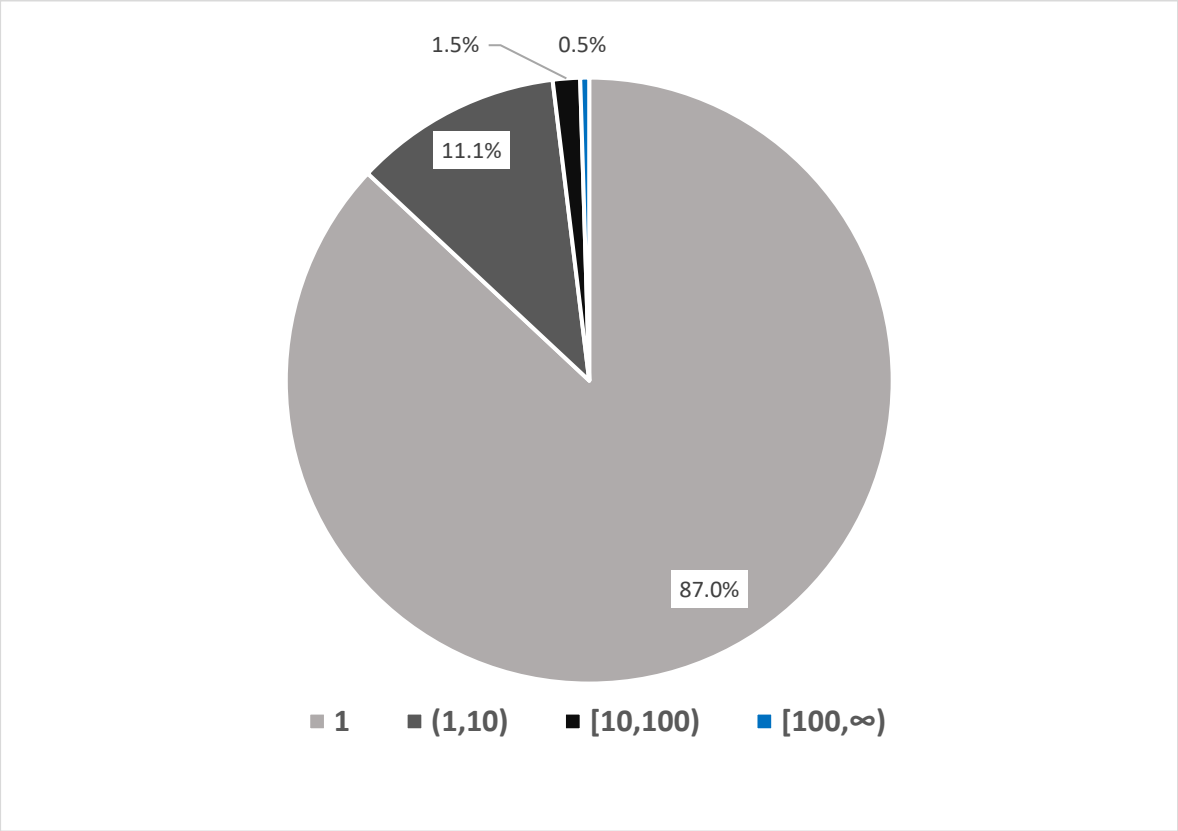
Memory Distribution



DiskDroid – Framework



Hot Edge Selector – Motivation



Most of the path edges (87%) are visited only once

Hot Edge Selector – Algorithm

Procedure Prop(e):
    if e *is not a hot edge* :
        Insert e into $WorkList$;
    elif $e \notin PathEdge$:
        Insert e into $WorkList$
        Insert e into $PathEdge$

Hot Edge Selector

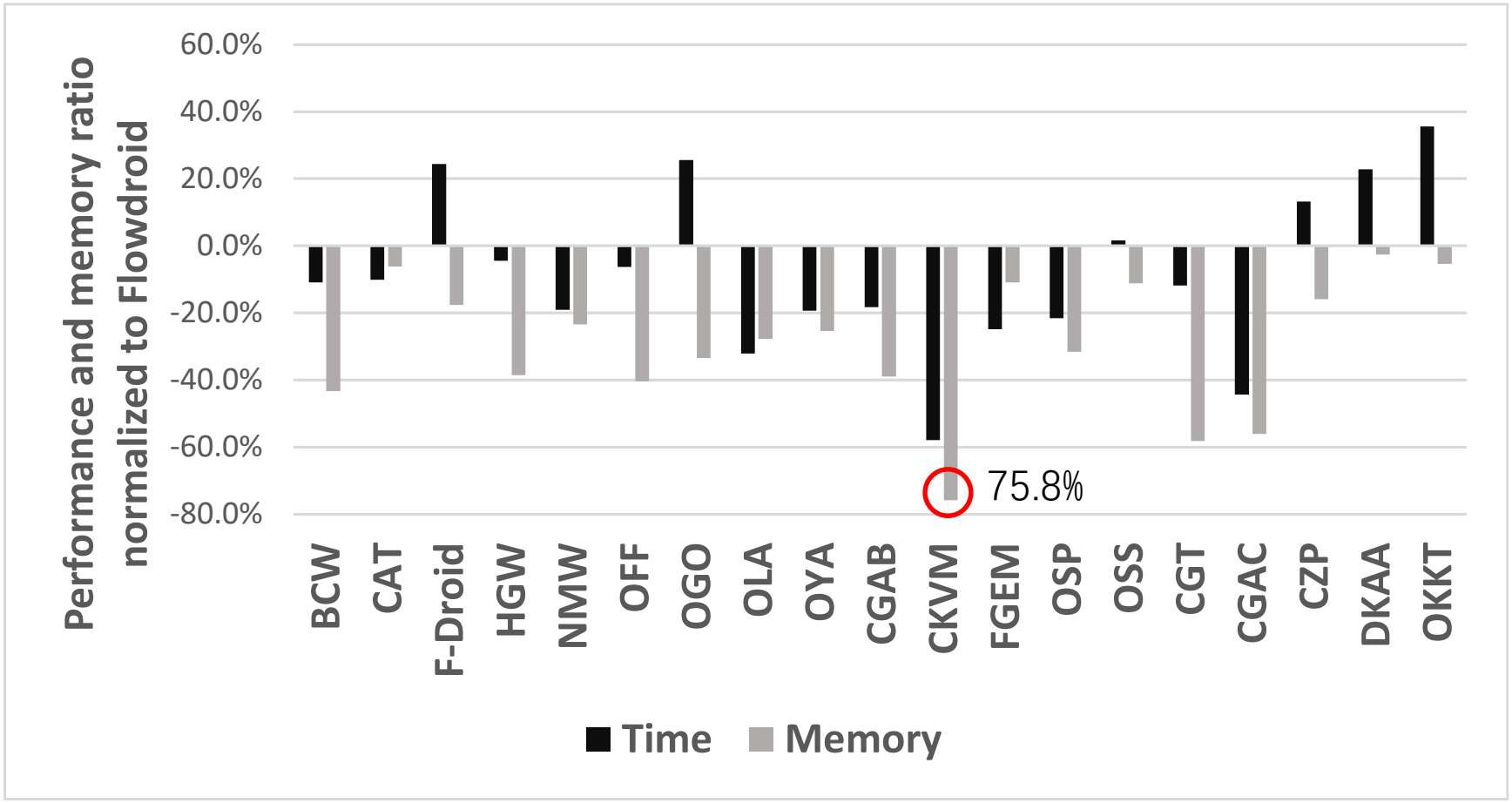
Is PathEdge $< *, * > \rightarrow < n, d >$ hot or not?

It is difficult to distinguish

Hot Edge Selector

- Three heuristics to determine whether a path edge $p = <*,* > \rightarrow <n, d>$ is hot or not:
  - *n* is a loop header
  - *n* is a function entry, or *n* is an exit node with *d* related to the formal parameters of procedure of *n*, or *n* is a return site with *d* related to the actual parameters at the callsite
  - p is derived from a backward IFDS pass in FlowDroid

Is hot edge optimization effective?



Is hot edge optimization effective?

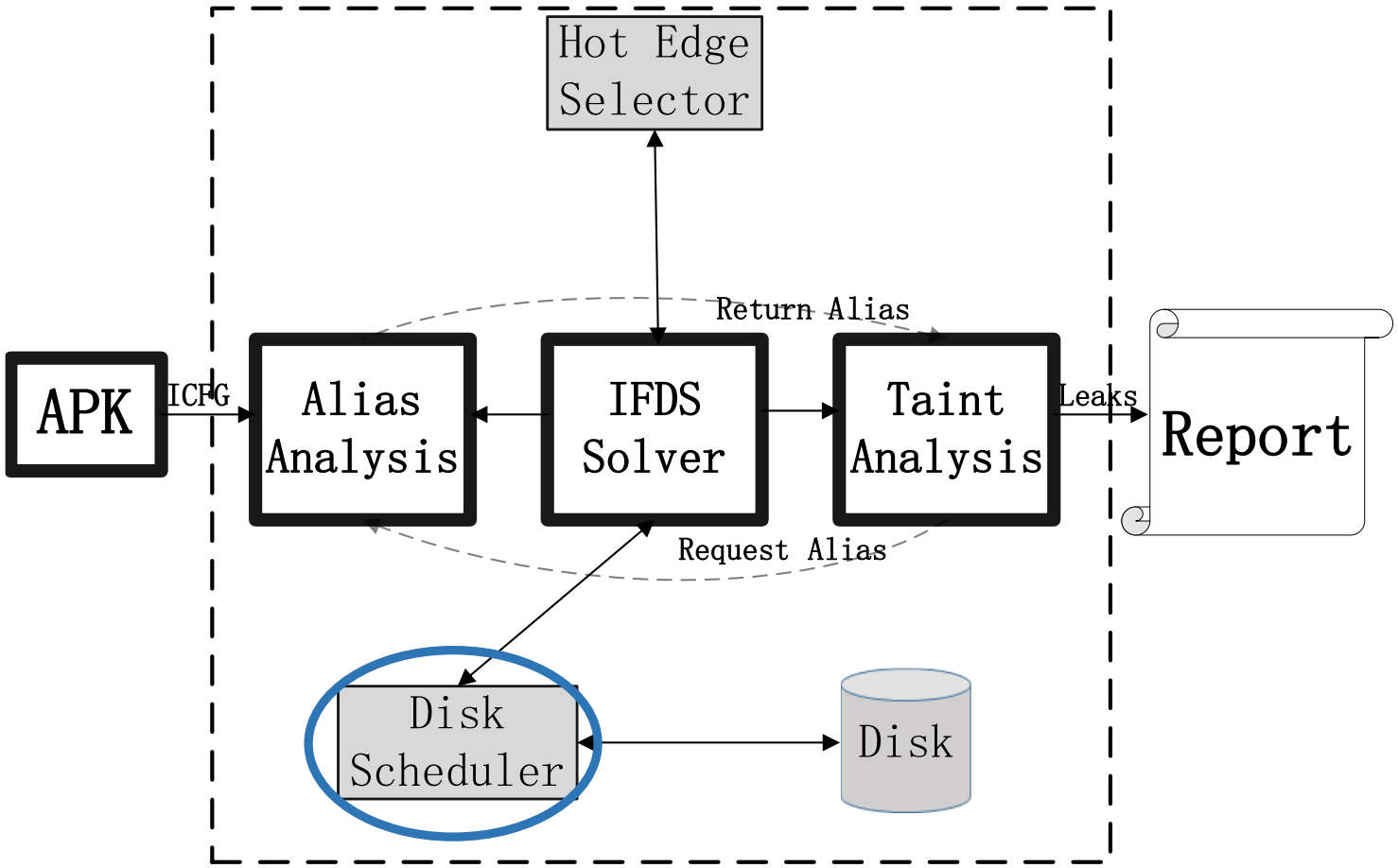
	#FlowDroid	#Optimized	Ratio
BCW	32,447,505	44,222,211	1.36
CAT	44,069,465	77,675,474	1.76
F-Droid	29,206,743	38,638,259	1.32
HGW	25,926,773	83,714,388	3.23
NMW	30,813,008	40,804,585	1.32
OFF	25,710,812	34,552,561	1.34
OGO	39,295,629	80,583,394	2.05
OLA	41,666,549	57,461,639	1.38
OYA	29,122,085	32,275,022	1.11
CGAB	132,176,249	275,527,399	2.08
CKVM	38,541,452	41,518,262	1.08
FGEM	37,480,947	85,214,757	2.27
OSP	52,378,247	60,983,905	1.16
OSS	67,487,451	158,045,173	2.34
CGT	160,534,182	517,692,586	3.22
CGAC	103315965	177215471	1.72
CZP	140323650	466792064	3.33
DKAA	93279912	173273865	1.86
OKKT	64216399	131378047	2.05

MonitorStmt
IfStmt
SwitchStmt
NopStmt
…

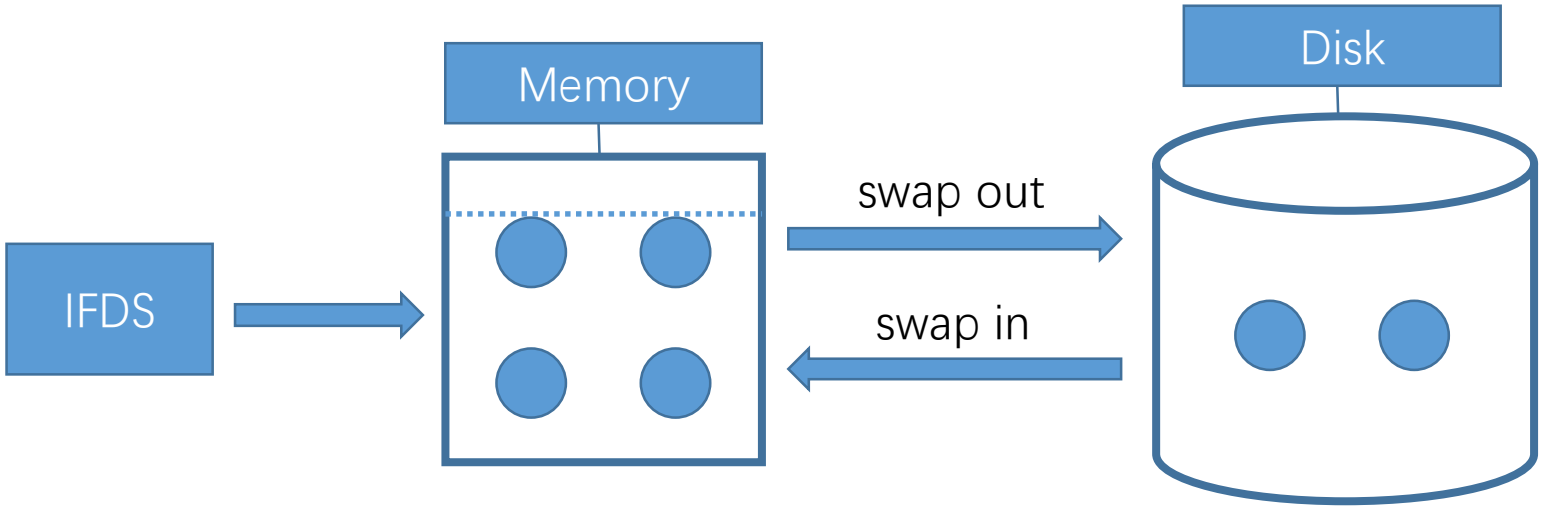
Hot Edge Selector

Some APPs cannot be analyzed under 10GB memory budget, even though we apply Hot Edge Selector

DiskDroid – Framework



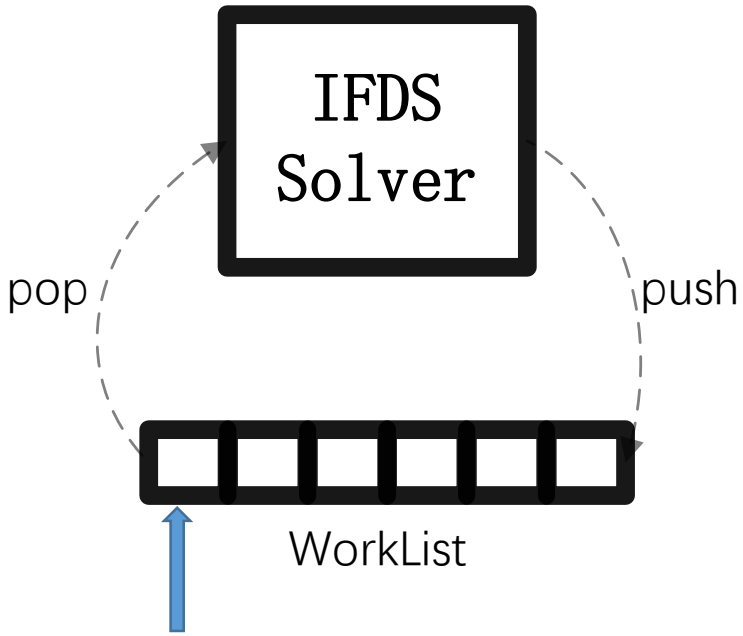
Disk Scheduler



Disk Scheduler

Which path edge needs to be swapped out?

Disk Scheduler – Swapping Policies



Disk Scheduler

How to swap out?

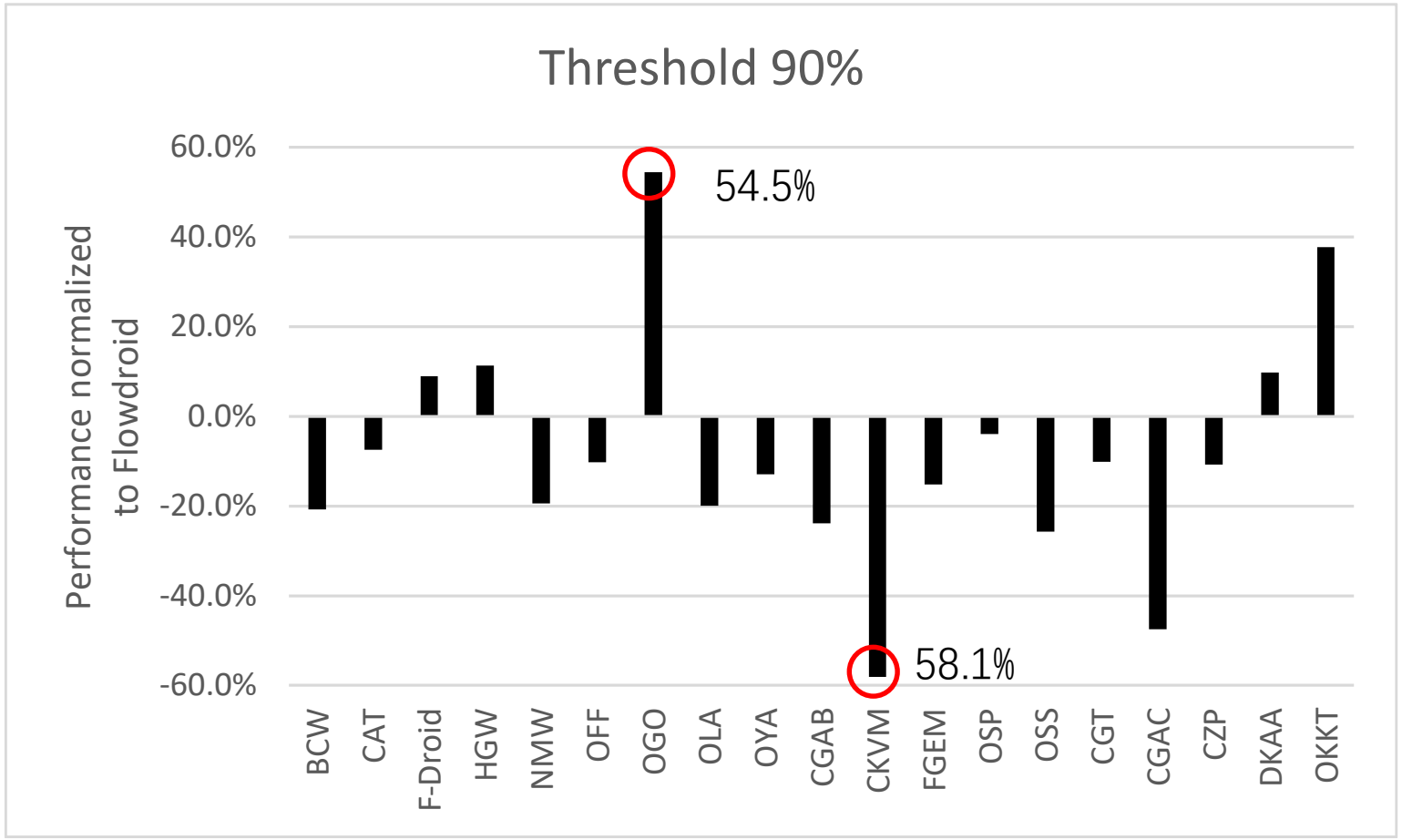
Disk Scheduler – Grouping Policies

- Five Grouping Policies:
  - Method: $< s_m, * > \rightarrow < *, * >$
  - Method&Source: $< s_m, d > \rightarrow < *, * >$
  - Method&Target: $< s_m, * > \rightarrow < *, d >$
  - Source: $< *, d > \rightarrow < *, * >$
  - Target: $< *, * > \rightarrow < *, d >$

Evaluation

- Benchmarks:
  - 19/2,053 apps (F-Droid) require 10GB to 128 GB of RAM
  - 162/2,053 apps (F-Droid) require more than 128GB of RAM
- Platform:
  - Intel Xeon E7-4809v3 (2.0GHz) server with 128GB RAM
  - IFDS solver time budget: 3 hours
  - FlowDroid (**128GB** RAM)
  - DiskDroid (**10GB** RAM)
- DiskDroid produces the same results as FlowDroid
  - validated by DroidBench and apps

How is the runtime performance of DiskDroid compared to FlowDroid?

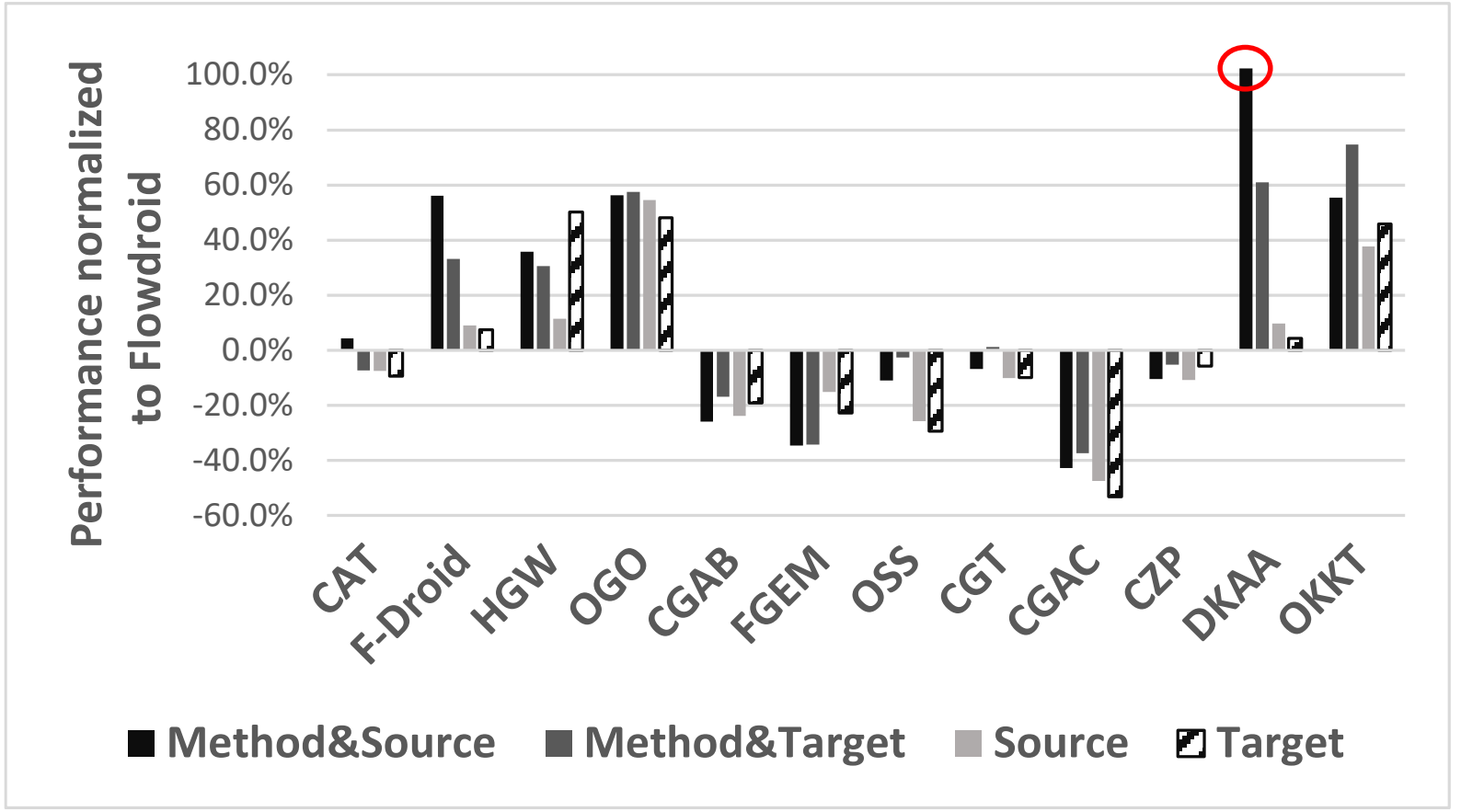


Speedup 8.6% averagely

How is the runtime performance of DiskDroid compared to FlowDroid?

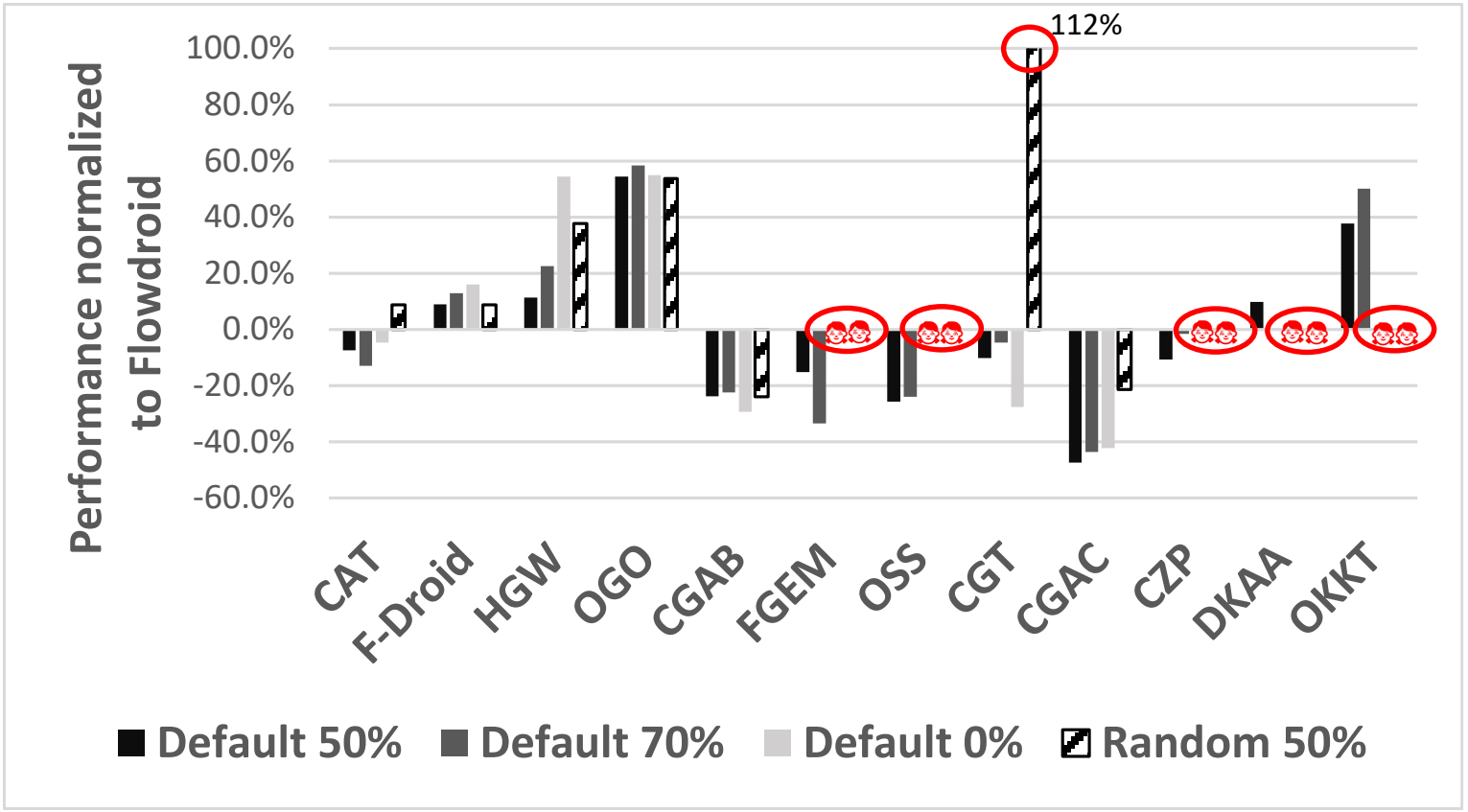
21/162 apps requiring more than 128GB of RAM by FlowDroid, DiskDroid can analyze each app in 3 hours

Are the grouping strategies effective?



Source scheme exhibits the best run time performance and it is used as the default grouping scheme.

Are the scheduling strategies effective?



The ***random*** policy performs poorly

The run time differences are insignificant when a different swapping ratio (50% vs 70%) is enforced

Conclusion

- **DiskDroid**: a new disk-assisted approach to scale up IFDS algorithms
- DiskDroid can analyze apps under the memory budget of 10GB while FlowDroid requires up to 128GB
- DiskDroid achieve slight performance improvement of 8.6%
- The tool is publicly available at https://github.com/HaofLi/DiskDroid

Q & A

Thanks!