ECOOP 2021 virtual

Accelerating Object-Sensitive Pointer Analysis by Exploiting Object Containment and Reachability

Dongjie He[1], Jingbo Lu[1], Yaoqing Gao[2], and Jingling Xue[1]

A new
Pointer Analysis Technique
for
Object-Oriented Programs

Pointer Analysis

- Statically determines

"possible runtime values of a variable?"

Uses of Pointer Analysis

- Foundation of many clients
  - Call-graph construction
  - Security analysis
  - Bug detection
  - Compiler optimization
  - Program understanding
  - …
- Many tools available

Uses of Pointer Analysis

- Foundation of many clients
  - Call-graph construction
  - Security analysis
  - Bug detection
  - Compiler optimization
  - Program understanding
  - …
- Many tools available

A precise and efficient pointer analysis benefits all above clients & tools.

Context Sensitivity

- One of the most successful techniques in developing highly precise pointer analysis for OO programs

- Distinguish variables/objects in a method by different calling contexts

Context Sensitivity

- Call-site Sensitivity (*k*CFA)
- **Object Sensitivity (*k*OBJ)**
- Type Sensitivity (kType)
- …

Arguably the best context abstraction for OO programs

Object Sensitivity: Example

```
class A {
    Object id(Object p) {
        return p;
    }
}
```

```
static void main() {
    A a1 = new A(); // A1
    v1 = a1.id("s1");
    A a2 = new A(); // A2
    v2 = a2.id("s2");
}
```

Object Sensitivity: Example

```
class A {
    Object id(Object p) {
        return p;
    }
}
```

```
static void main() {
    A a1 = new A(); // A1
    v1 = a1.id("s1");
    A a2 = new A(); // A2
    v2 = a2.id("s2");
}
```

Variable	Object
p	"s1", "s2"
v1	"s1", "s2"
v2	"s1", "s2"

Context-Insensitivity

Object Sensitivity: Example

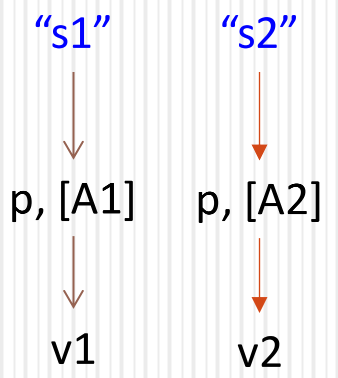
```
class A {
    Object id(Object p) {
        return p;
    }
}
```

```
static void main() {
    A a1 = new A(); // A1
    v1 = a1.id("s1");
    A a2 = new A(); // A2
    v2 = a2.id("s2");
}
```

Variable	Object
p	"s1", "s2"
v1	"s1", "s2"
v2	"s1", "s2"

Context-Insensitivity

Object Sensitivity: Example




```
class A {
    Object id(Object p) {
        return p;
    }
}
```

```
static void main() {
    A a1 = new A(); // A1
    v1 = a1.id(“s1”);
    A a2 = new A(); // A2
    v2 = a2.id(“s2”);
}
```

Context	Variable	Object
[A1]	p	“s1”
[A2]	p	“s2”
[ ]	v1	“s1”
[ ]	v2	“s2”

1-Object-sensitivity

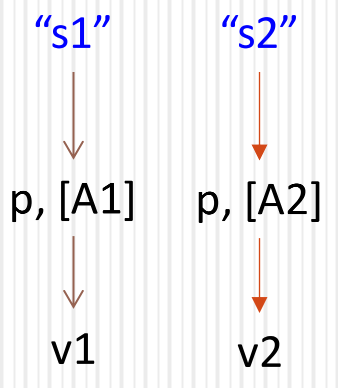
Variable	Object
p	“s1”, “s2”
v1	“s1”, “s2”
v2	“s1”, “s2”

Context-Insensitivity

Object Sensitivity: Example

```
class A {
    Object id(Object p) {
        return p;
    }
}
```

```
static void main() {
    A a1 = new A(); // A1
    v1 = a1.id("s1");
    A a2 = new A(); // A2
    v2 = a2.id("s2");
}
```



Context	Variable	Object
[A1]	p	"s1"
[A2]	p	"s2"
[ ]	v1	"s1"
[ ]	v2	"s2"

1-Object-sensitivity

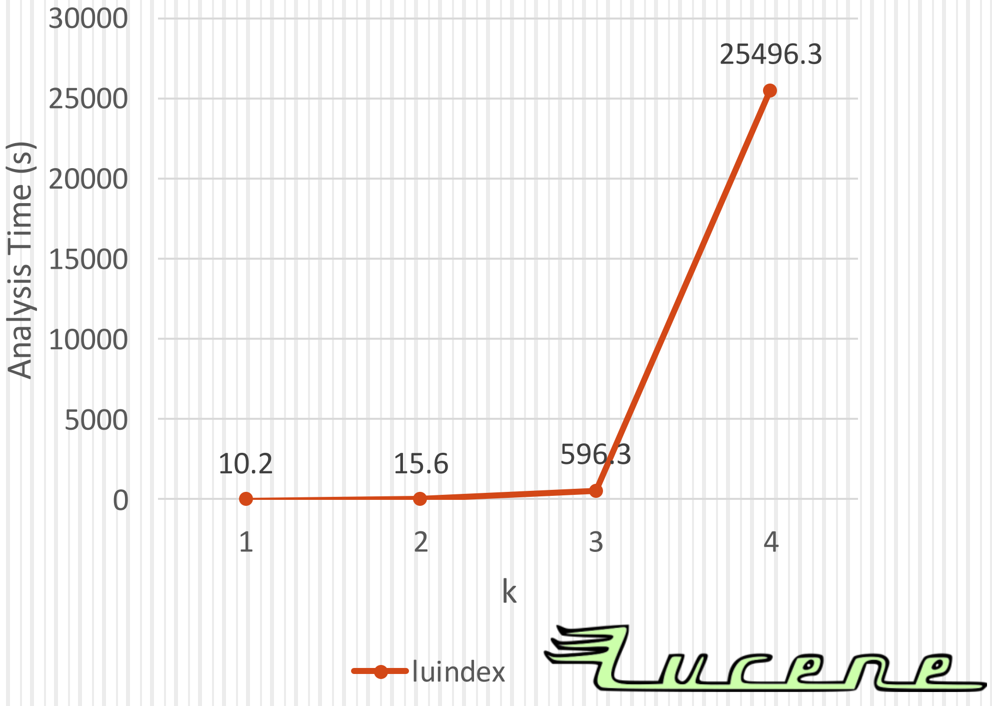
Variable	Object
p	"s1", "s2"
v1	"s1", "s2"
v2	"s1", "s2"

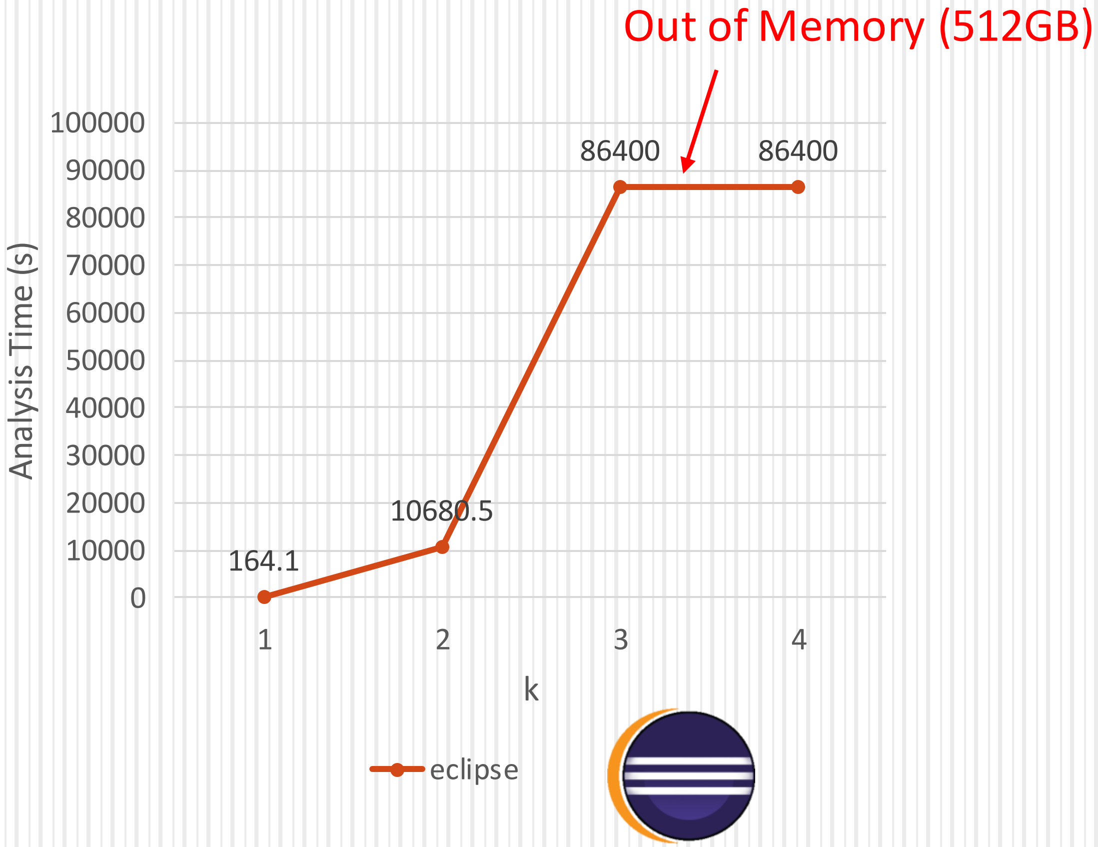
Context-Insensitivity

Precise

Problem with Object Sensitivity (*k*OBJ)

- Inefficient & Unscalable



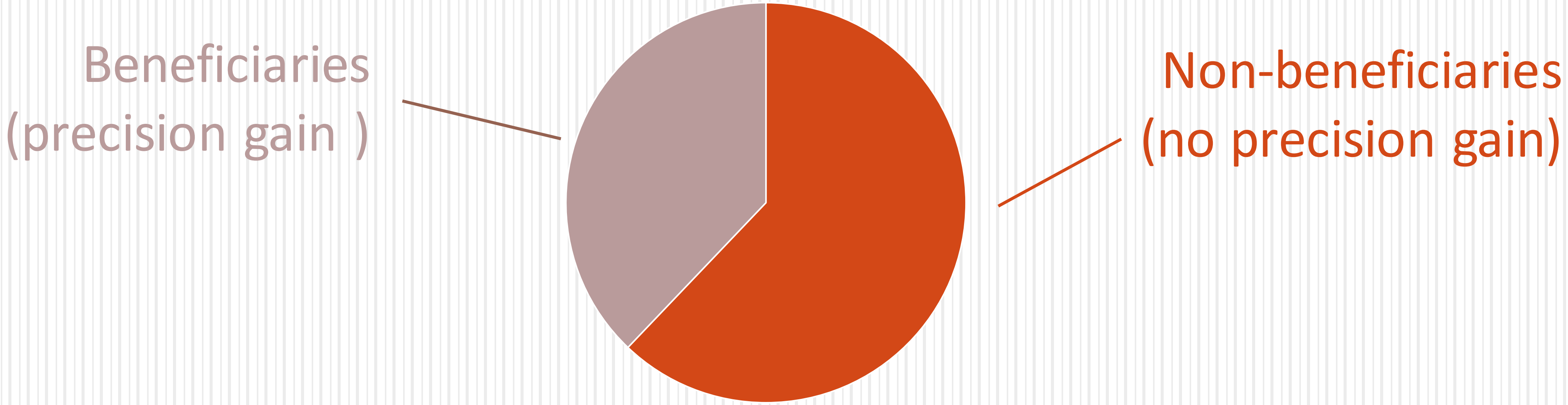


Problem with Object Sensitivity (kOBJ)

Traditional: apply contexts to all variables/objects

Problem with Object Sensitivity (kOBJ)

Tranditional: apply contexts to all variables/objects



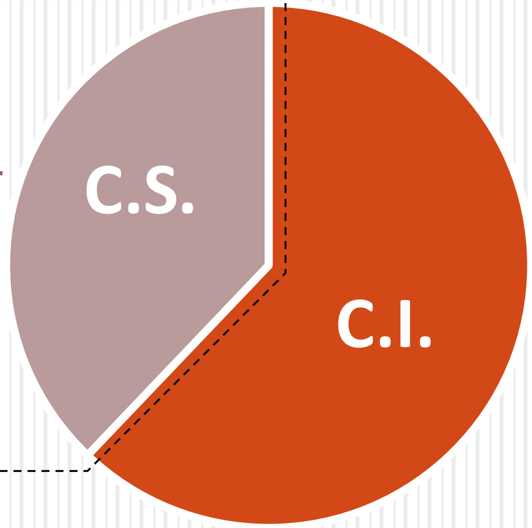
Our Goal

Identify precision-critical variables/objects

Traditional: apply contexts to
all variables/objects

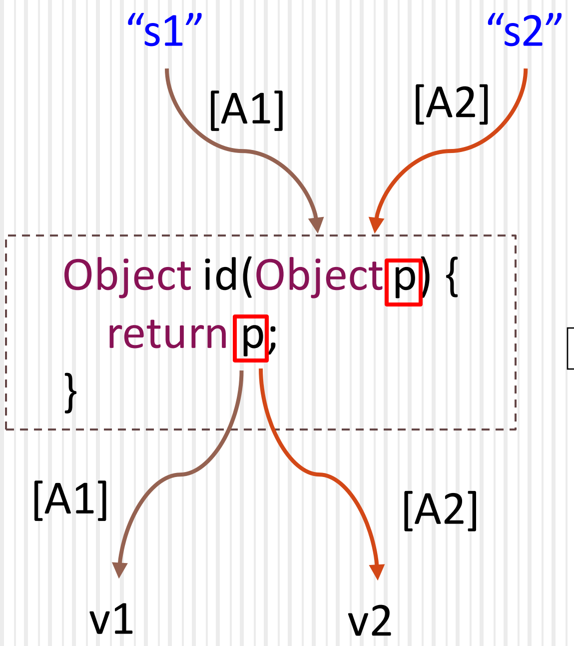
Beneficiaries
(precision gain)

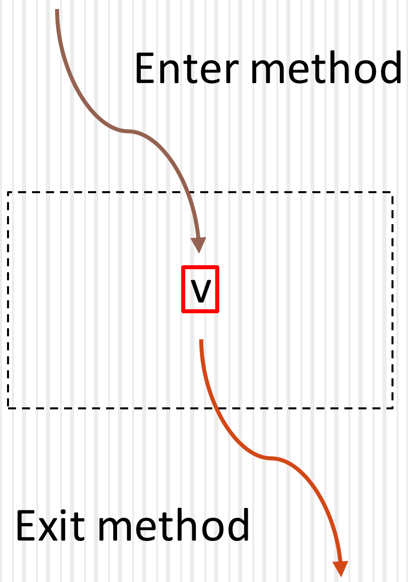
Precision-critical variables/objects



Non-beneficiaries
(no precision gain)

Challenge





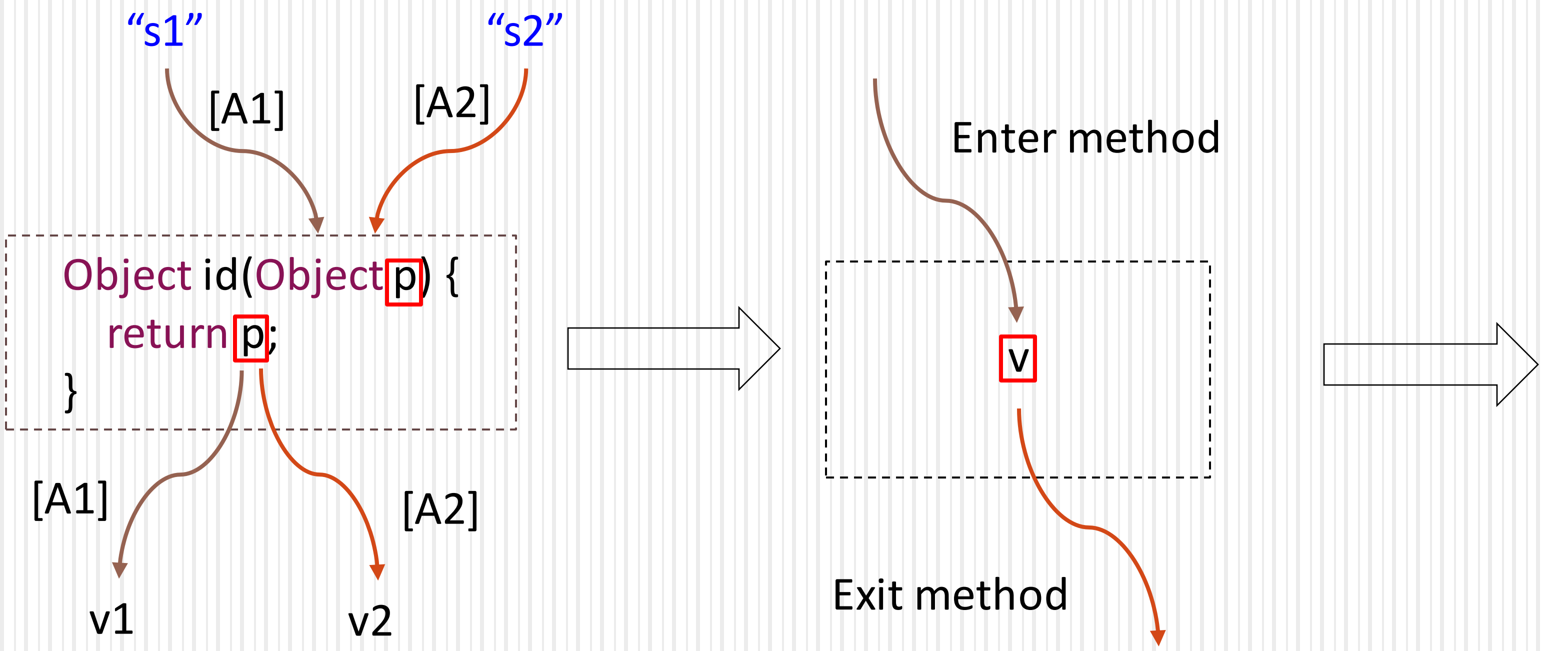
$$L_{FC} = L_F \bigcap L_C$$ [1][2][3]

L_F: field sensitivity

L_C: context sensitivity

[1] Manu Sridharan and Rastislav Bodík. Refinement-based context-sensitive points-to analysis for Java. In PLDI 2006.
[2] Jingbo Lu and Jingling Xue. Precision-Preserving Yet Fast Object-Sensitive Pointer Analysis with Partial Context Sensitivity. In OOPSLA 2019.
[3] Jingbo Lu, Dongjie He and Jingling Xue. Eagle: CFL-Reachability-based Precision-Preserving Acceleration of Object-Sensitive Pointer Analysis with Partial Context Sensitivity. In TOSEM 2021.

Challenge



Undecidable problem[4]

$$L_{FC} = L_F \bigcap L_C$$ [1][2][3]

L_F: field sensitivity

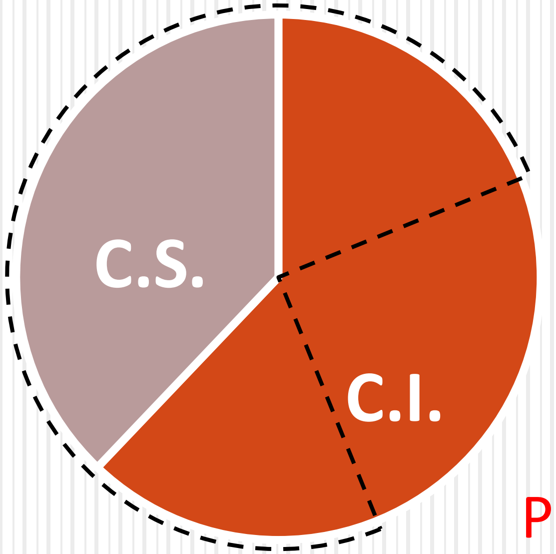
L_C: context sensitivity

[1] Manu Sridharan and Rastislav Bodík. Refinement-based context-sensitive points-to analysis for Java. In PLDI 2006.
[2] Jingbo Lu and Jingling Xue. Precision-Preserving Yet Fast Object-Sensitive Pointer Analysis with Partial Context Sensitivity. In OOPSLA 2019.
[3] Jingbo Lu, Dongjie He and Jingling Xue. Eagle: CFL-Reachability-based Precision-Preserving Acceleration of Object-Sensitive Pointer Analysis with Partial Context Sensitivity. In TOSEM 2021.
[4] Thomas Reps. Undecidability of context-sensitive data-dependence analysis. In TOPLAS 2000.

Existing Solutions

- Eagle[2][3]： over-approximation (based on CFL reachability)
  - Inter-procedural
  - Precision-preserving

$$L_{FC} = L_F \bigcap L_C \Rightarrow L_{RC} = L_R \bigcap L_C$$

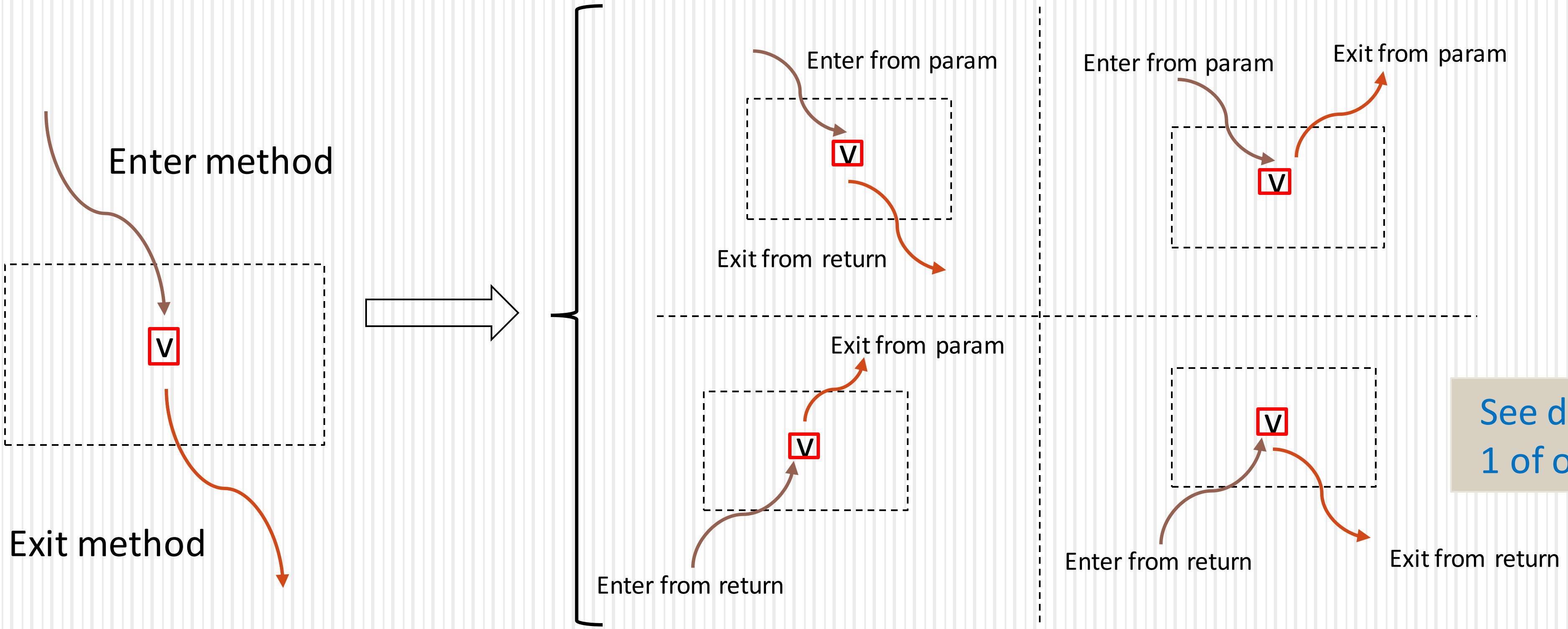


Precise but less efficient

[2] Jingbo Lu and Jingling Xue. Precision-Preserving Yet Fast Object-Sensitive Pointer Analysis with Partial Context Sensitivity. In OOPSLA 2019.
[3] Jingbo Lu, Dongjie He and Jingling Xue. Eagle: CFL-Reachability-based Precision-Preserving Acceleration of Object-Sensitive Pointer Analysis with Partial Context Sensitivity. In TOSEM 2021.

Existing Solutions

- Zipper[5][6]: patterns-based

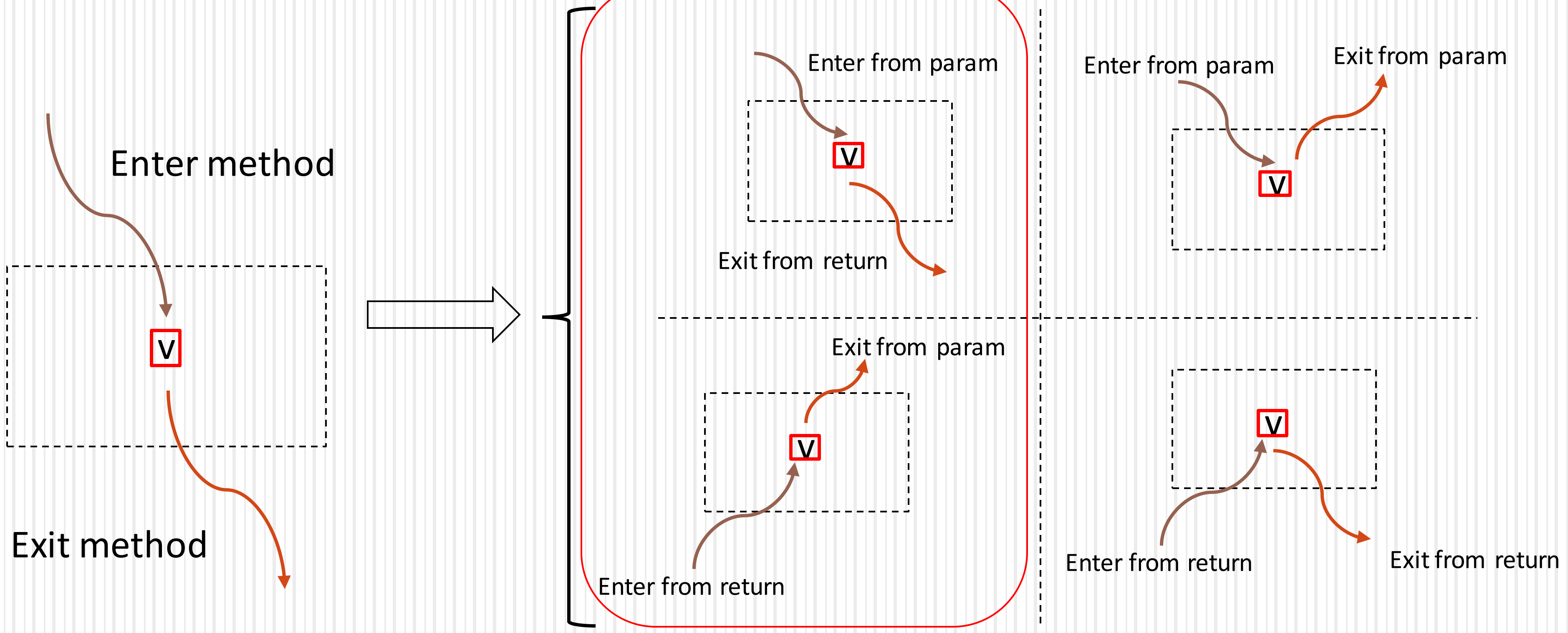


See details in Figure 1 of our Paper

[5] Yue Li, Tian Tan, Anders Møller, and Yannis Smaragdakis. Precision-Guided Context Sensitivity for Pointer Analysis. In OOPSLA 2018.
[6] Yue Li, Tian Tan, Anders Møller, and Yannis Smaragdakis. A Principled Approach to Selective Context Sensitivity for Pointer Analysis. In TOPLAS 2020.

Existing Solutions

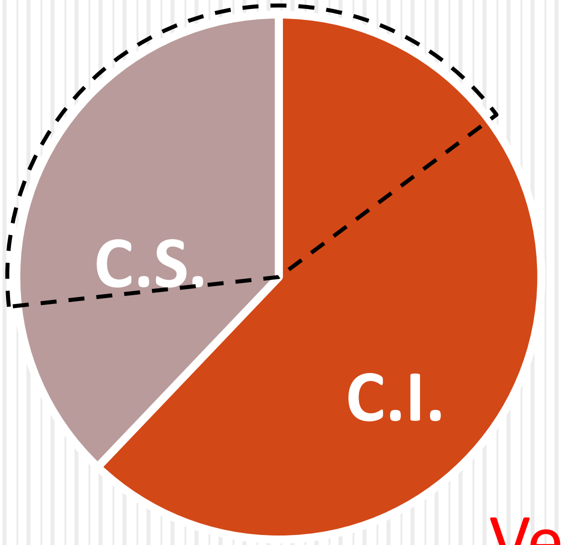
- Zipper[5][6]: patterns-based



[5] Yue Li, Tian Tan, Anders Møller, and Yannis Smaragdakis. Precision-Guided Context Sensitivity for Pointer Analysis. In OOPSLA 2018.
[6] Yue Li, Tian Tan, Anders Møller, and Yannis Smaragdakis. A Principled Approach to Selective Context Sensitivity for Pointer Analysis. In TOPLAS 2020.

Existing Solutions

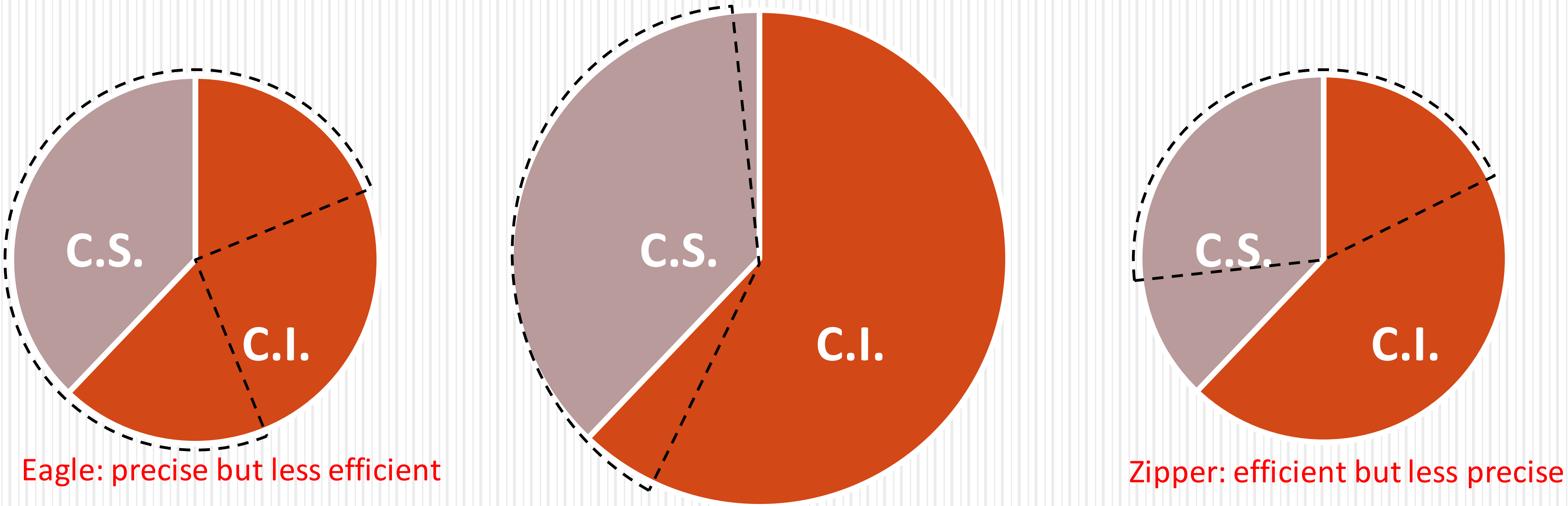
- Zipper[5][6]: patterns-based



Very efficient but less precise

[5] Yue Li, Tian Tan, Anders Møller, and Yannis Smaragdakis. Precision-Guided Context Sensitivity for Pointer Analysis. In OOPSLA 2018.
[6] Yue Li, Tian Tan, Anders Møller, and Yannis Smaragdakis. A Principled Approach to Selective Context Sensitivity for Pointer Analysis. In TOPLAS 2020.

Our Solution: Turner



Turner: precise and efficient

Our Solution: Turner

- Stage 1: Object Containment Analysis
- Stage 2: Object Reachability Analysis

Our Solution: Turner

A motivating example:

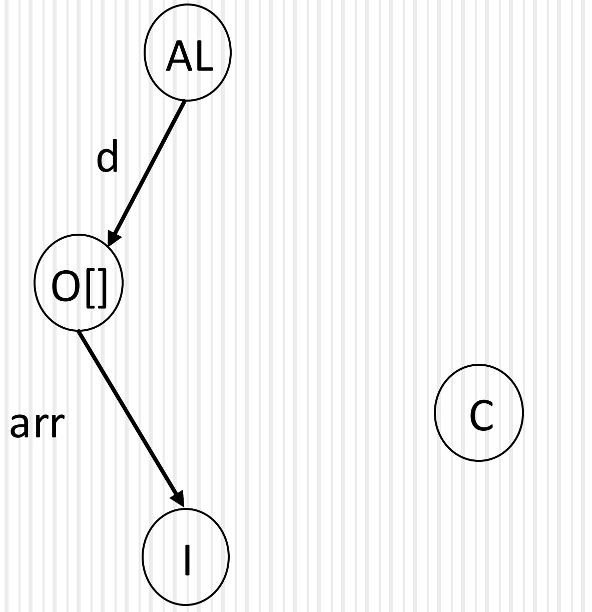
```
class Integer { int v; }
class ArrayList {
    Object[] d;
    ArrayList() {
        Object[] t = new Object[10]; // O[]
        this.d = t;
}}
```

```
class Client {
    void foo(Integer p) {
        ArrayList a = new ArrayList(); // AL
        a.d[0] = p;
        …
        print(a.d[0]);
    }
    static void main() {
        Client c = new Client(); // C
        Integer t = new Integer(1000); // I
        c.foo(t);
}}
```

Turner: Object Containment Analysis

```
class Integer { int v; }
class ArrayList {
   Object[] d;
   ArrayList() {
      Object[] t = new Object[10]; // O[]
      this.d = t;
}}
```

```
class Client {
   void foo(Integer p) {
        ArrayList a = new ArrayList(); // AL
        a.d[0] = p;
        ...
        print(a.d[0]);
   }
   static void main() {
      Client c = new Client(); // C
      Integer t = new Integer(1000); // I
      c.foo(t);
}}
```

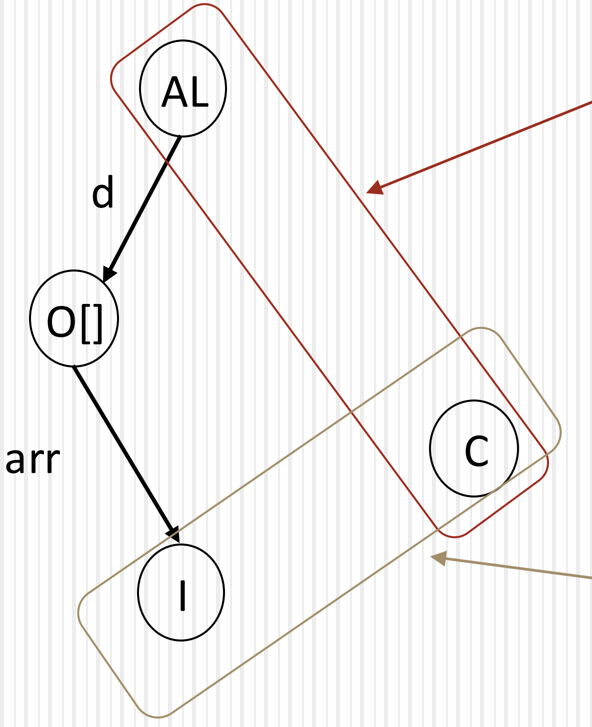


Object Containment Graph

Turner: Object Containment Analysis

```
class Integer { int v; }
class ArrayList {
   Object[] d;
   ArrayList() {
      Object[] t = new Object[10]; // O[]
      this.d = t;
}}
```

```
class Client {
   void foo(Integer p) {
       ArrayList a = new ArrayList(); // AL
       a.d[0] = p;
       …
       print(a.d[0]);
   }
   static void main() {
       Client c = new Client(); // C
       Integer t = new Integer(1000); // I
       c.foo(t);
}}
```



Object Containment Graph

Top Container

often used locally

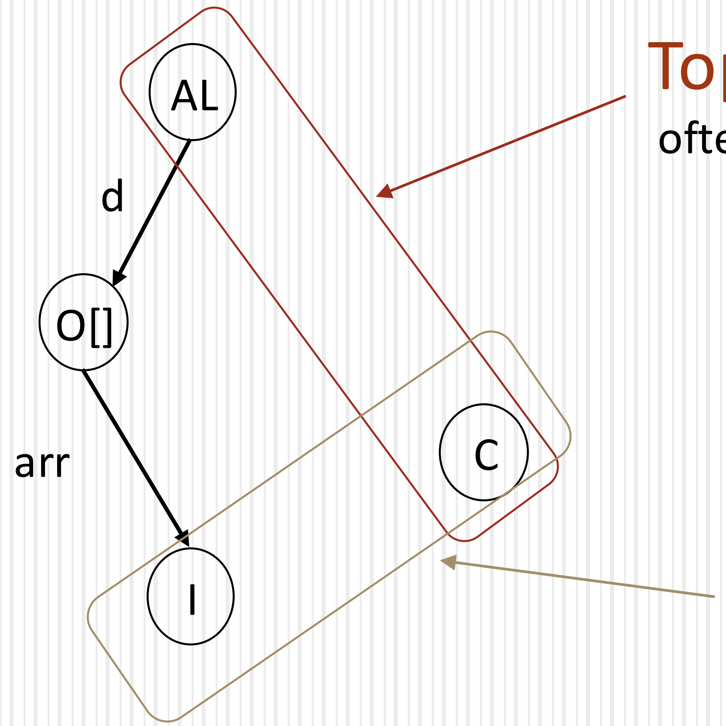
Bottom Container

Often encapsulating primitive data, e.g., Long, Integer, String, etc.

Turner: Object Containment Analysis

```
class Integer { int v; }
class ArrayList {
   Object[] d;
   ArrayList() {
      Object[] t = new Object[10]; // O[]
      this.d = t;
}}
```

```
class Client {
   void foo(Integer p) {
       ArrayList a = new ArrayList(); // AL
       a.d[0] = p;
       …
       print(a.d[0]);
   }
   static void main() {
      Client c = new Client(); // C
      Integer t = new Integer(1000); // I
      c.foo(t);
}}
```



Top Container
often used locally

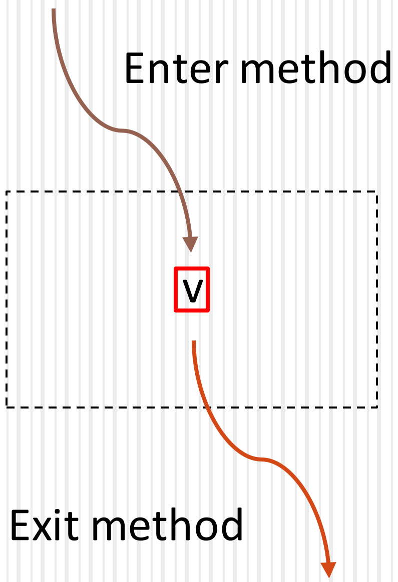
Bottom Container
Often encapsulating primitive data, e.g., Long, Integer, String, etc.

Object Containment Graph

Observation: objects in Top/Bottom Containers are unlikely to be context-sensitive.

Turner: Object Reachability Analysis

- **Basic Idea**
  - Define a language *L*
    - Cheaper than $L_{FC} = L_F \bigcap L_C$
    - Effective than L_{RC}
    - Consider all 4 sub-patterns.
  - Eliminate context-insensitive value flows.
    - Exploit Object Containment Analysis
  - Over-approximate (context-sensitive) value flows.
  - Select precision-critical variables/objects via *L*



Turner: Object Reachability Analysis

- from L_0[1][7] to L_1: eliminate context-insensitive value flows
  - remove global edges (e.g., assignglobal)
  - cs-likely edges are added only for objects not in Top/Bottom Containers (e.g. O[]).

$$
L_0 : \begin{cases}
\mathit{flowsto} \longrightarrow \mathsf{new}\ \mathit{flows}^* \\
\mathit{flows} \longrightarrow \mathsf{assign} \mid \mathsf{assignglobal} \mid \mathsf{store[f]}\ \mathit{alias}\ \mathsf{load[f]} \\
\overline{\mathit{flowsto}} \longrightarrow \overline{\mathit{flows}}^*\ \overline{\mathsf{new}} \\
\overline{\mathit{flows}} \longrightarrow \overline{\mathsf{assign}} \mid \overline{\mathsf{assignglobal}} \mid \overline{\mathsf{load[f]}}\ \mathit{alias}\ \overline{\mathsf{store[f]}} \\
\mathit{alias} \longrightarrow \overline{\mathit{flowsto}}\ \mathit{flowsto}
\end{cases}
$$

$$
L_1 : \begin{cases}
\mathit{flowsto} \longrightarrow \mathsf{new}\ \mathit{flows}^* \\
\mathit{flows} \longrightarrow \mathsf{assign} \mid \mathsf{store[f]}\ \mathit{alias}\ \mathsf{load[f]} \\
\overline{\mathit{flowsto}} \longrightarrow \overline{\mathit{flows}}^*\ \overline{\mathsf{new}} \\
\overline{\mathit{flows}} \longrightarrow \overline{\mathsf{assign}} \mid \overline{\mathsf{load[f]}}\ \mathit{alias}\ \overline{\mathsf{store[f]}} \\
\mathit{alias} \longrightarrow \overline{\mathit{flowsto}}\ \mathsf{cs\text{-}likely}\ \mathit{flowsto}
\end{cases}
$$

introduce negligible precision loss

[1] Manu Sridharan and Rastislav Bodík. Refinement-based context-sensitive points-to analysis for Java. In PLDI 2006.
[7] Manu Sridharan, Denis Gopan, Lexin Shan, and Rastislav Bodík. Demand-driven points-to analysis for Java. In OOPSLA 2005.

Turner: Object Reachability Analysis

- from L_1 to L_2: enable to reason about reachability from variables
  - do not distinguish flowsto and flows

$$L_1 : \begin{cases} \mathit{flowsto} \longrightarrow \mathsf{new}\ \mathit{flows}^{*} \\ \mathit{flows} \longrightarrow \mathsf{assign} \mid \mathsf{store[f]}\ \mathit{alias}\ \mathsf{load[f]} \\ \overline{\mathit{flowsto}} \longrightarrow \overline{\mathit{flows}}^{*}\ \overline{\mathsf{new}} \\ \overline{\mathit{flows}} \longrightarrow \overline{\mathsf{assign}} \mid \overline{\mathsf{load[f]}}\ \mathit{alias}\ \overline{\mathsf{store[f]}} \\ \mathit{alias} \longrightarrow \overline{\mathit{flowsto}}\ \mathsf{cs\text{-}likely}\ \mathit{flowsto} \end{cases}$$

$$L_2 : \begin{cases} \mathit{flows} \longrightarrow (\mathsf{new} \mid \mathsf{assign} \mid \mathsf{store[f]}\ \mathit{alias}\ \mathsf{load[f]})^{*} \\ \overline{\mathit{flows}} \longrightarrow (\overline{\mathsf{new}} \mid \overline{\mathsf{assign}} \mid \overline{\mathsf{load[f]}}\ \mathit{alias}\ \overline{\mathsf{store[f]}})^{*} \\ \mathit{alias} \longrightarrow \overline{\mathit{flows}}\ \mathsf{cs\text{-}likely}\ \mathit{flows} \end{cases}$$

Turner: Object Reachability Analysis

- from L_2 to L_3: enable intra-procedural analysis.
  - 1-limited access path.

$$L_2 : \begin{cases} \mathit{flows} \longrightarrow (\mathsf{new} \mid \mathsf{assign} \mid \mathsf{store[f]}\ \mathit{alias}\ \mathsf{load[f]})^* \\ \overline{\mathit{flows}} \longrightarrow (\overline{\mathsf{new}} \mid \overline{\mathsf{assign}} \mid \overline{\mathsf{load[f]}}\ \mathit{alias}\ \overline{\mathsf{store[f]}})^* \\ \mathit{alias} \longrightarrow \overline{\mathit{flows}}\ \mathsf{cs\text{-}likely}\ \mathit{flows} \end{cases}$$

$$L_3 : \begin{cases} \mathit{flows} \longrightarrow (\mathsf{new} \mid \mathsf{assign} \mid \mathsf{load} \mid \mathsf{store}\ \mathit{alias})^* \\ \overline{\mathit{flows}} \longrightarrow (\overline{\mathsf{new}} \mid \overline{\mathsf{assign}} \mid \overline{\mathsf{load}} \mid \mathit{alias}\ \overline{\mathsf{store}})^* \\ \mathit{alias} \longrightarrow \overline{\mathit{flows}}\ \mathsf{cs\text{-}likely}\ \mathit{flows} \end{cases}$$

  - a_i could reach a_j as long as a_0 points to a cs-likely object. (making inter-procedural analysis unnecessary)

$$\frac{b \ = \ a_0.m(a_1, ..., a_r)}{\forall\ i : a_i \xrightarrow{\mathsf{store}[p_i^{m'}]} a_0 \quad \forall\ i : a_0 \xrightarrow{\overline{\mathsf{store}[p_i^{m'}]}} a_i \quad a_0 \xrightarrow{\mathsf{load}[ret^{m'}]} b \quad b \xrightarrow{\overline{\mathsf{load}[ret^{m'}]}} a_0}$$

Turner: Object Reachability Analysis

- from L_3 to L_4: Regularization.

$$L_3 : \left\{ \begin{array}{rcl} \mathit{flows} & \longrightarrow & (\mathsf{new} \mid \mathsf{assign} \mid \mathsf{load} \mid \mathsf{store}\ \mathit{alias})^* \\ \overline{\mathit{flows}} & \longrightarrow & (\overline{\mathsf{new}} \mid \overline{\mathsf{assign}} \mid \overline{\mathsf{load}} \mid \mathit{alias}\ \overline{\mathsf{store}})^* \\ \mathit{alias} & \longrightarrow & \overline{\mathit{flows}}\ \textsf{cs-likely}\ \mathit{flows} \end{array} \right.$$

$$L_4 : \left\{ \begin{array}{rcl} \mathit{flows} & \longrightarrow & (\mathsf{new} \mid \mathsf{assign} \mid \mathsf{load})^* ((\mathsf{store} \mid \overline{\mathsf{store}})\ \overline{\mathit{flows}})? \\ \overline{\mathit{flows}} & \longrightarrow & (\overline{\mathsf{new}} \mid \overline{\mathsf{assign}} \mid \overline{\mathsf{load}})^* (\textsf{cs-likely}\ \mathit{flows})? \end{array} \right.$$

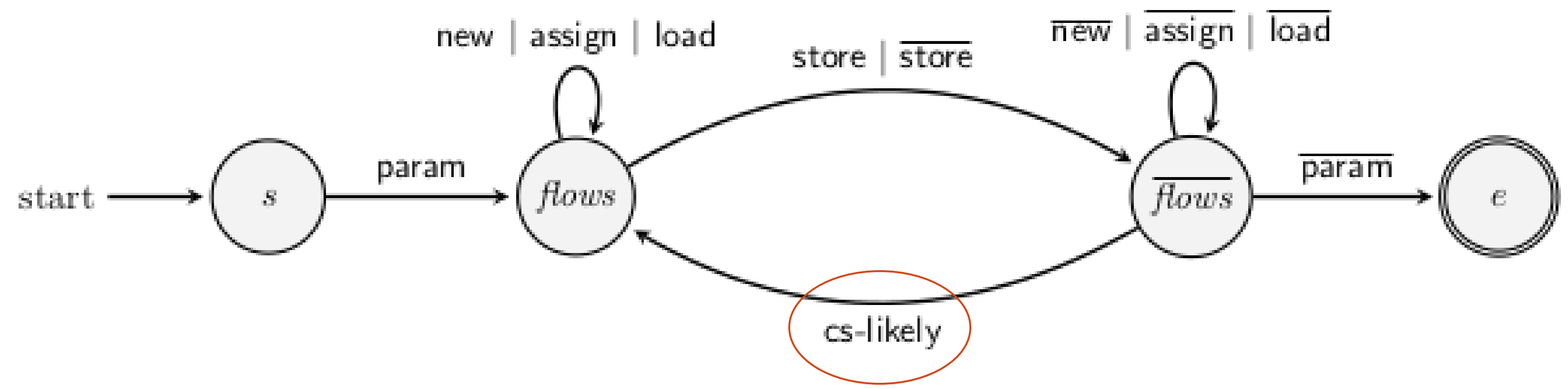
- From L_4 to L_5: enforce reasoning from parameters/return variables

$$L_4 : \left\{ \begin{array}{rcl} \mathit{flows} & \longrightarrow & (\mathsf{new} \mid \mathsf{assign} \mid \mathsf{load})^* ((\mathsf{store} \mid \overline{\mathsf{store}})\ \overline{\mathit{flows}})? \\ \overline{\mathit{flows}} & \longrightarrow & (\overline{\mathsf{new}} \mid \overline{\mathsf{assign}} \mid \overline{\mathsf{load}})^* (\textsf{cs-likely}\ \mathit{flows})? \end{array} \right.$$

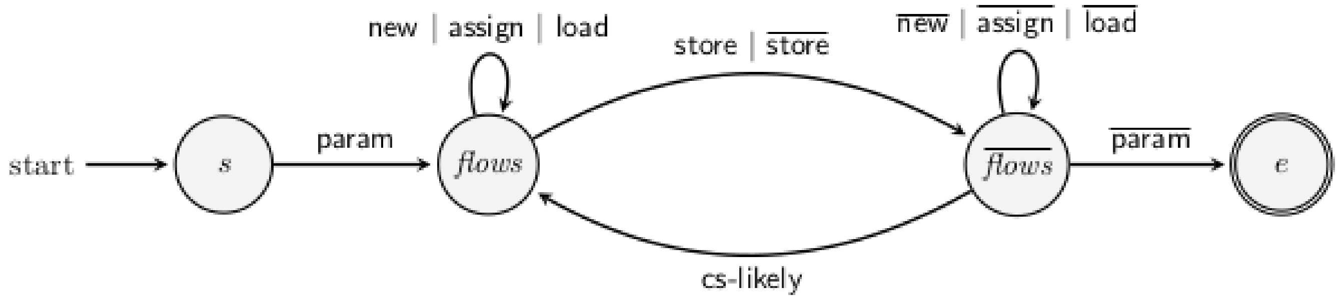
$$L_5 : \left\{ \begin{array}{rcl} \boxed{s \longrightarrow \mathsf{param}\ \mathit{flows}} & & \\ \mathit{flows} & \longrightarrow & (\mathsf{new} \mid \mathsf{assign} \mid \mathsf{load})^* ((\mathsf{store} \mid \overline{\mathsf{store}})\ \overline{\mathit{flows}})? \\ \overline{\mathit{flows}} & \longrightarrow & (\overline{\mathsf{new}} \mid \overline{\mathsf{assign}} \mid \overline{\mathsf{load}})^* (\textsf{cs-likely}\ \mathit{flows})? \\ \overline{\mathit{flows}} & \longrightarrow & \overline{\mathsf{param}}\ e \\ e & \longrightarrow & \epsilon \end{array} \right.$$

Turner: Object Reachability Analysis

- DFA (equivalent to L_5): over-approximate intra-procedural value flow
  - **v is on a path from s to e $\longrightarrow$ v should be C.S.**
  - Precision-preserving if objects selected in stage 1 are indeed context-insensitive.



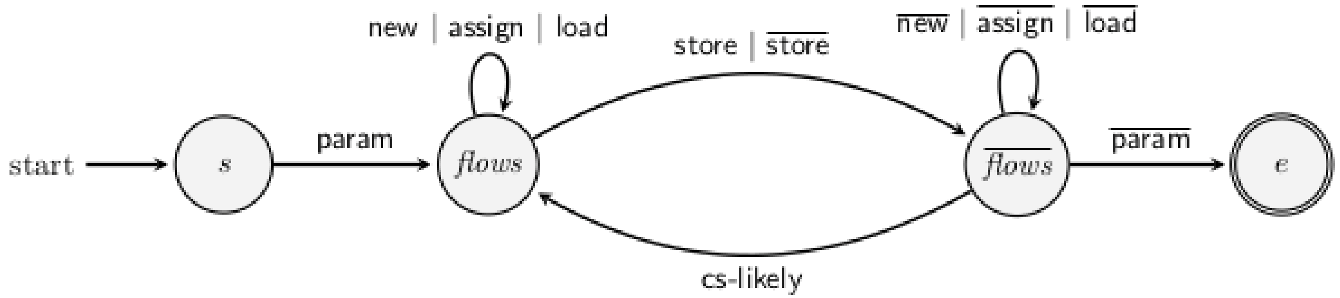
Turner: Object Reachability Analysis




```
class Integer { int v; }
class ArrayList {
    Object[] d;
    ArrayList() {
        Object[] t = new Object[10]; // O[]
        this.d = t;
}}
```

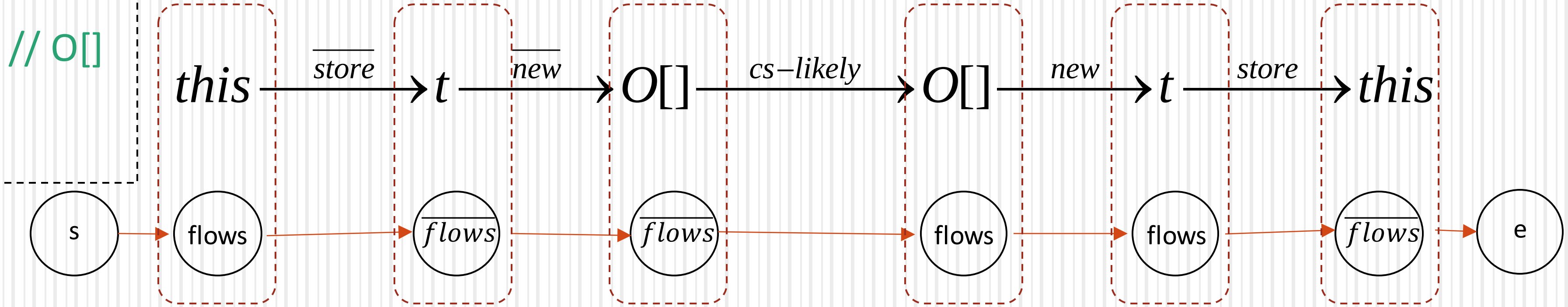
- 1. this, t, and O[] are precision-critical

Turner: Object Reachability Analysis

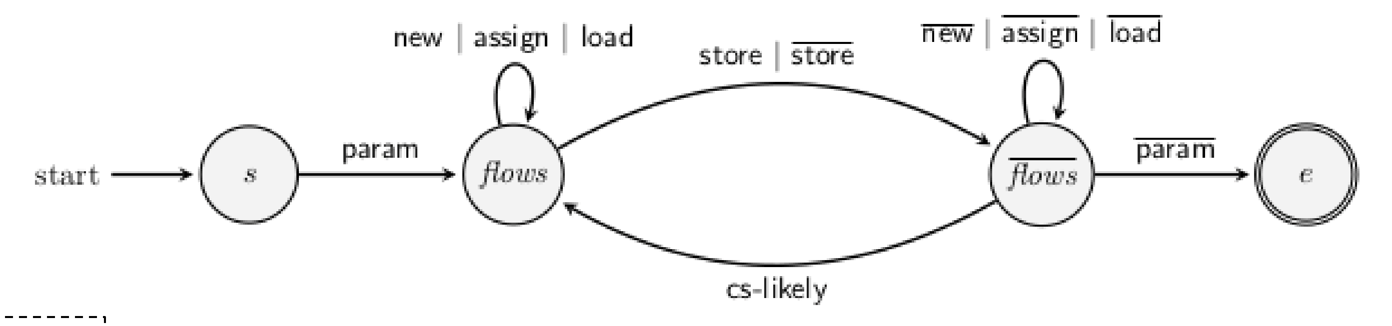



```
class Integer { int v; }
class ArrayList {
    Object[] d;
    ArrayList() {
        Object[] t = new Object[10]; // O[]
        this.d = t;
}}
```

❑ 1. this, t, and O[] are precision-critical

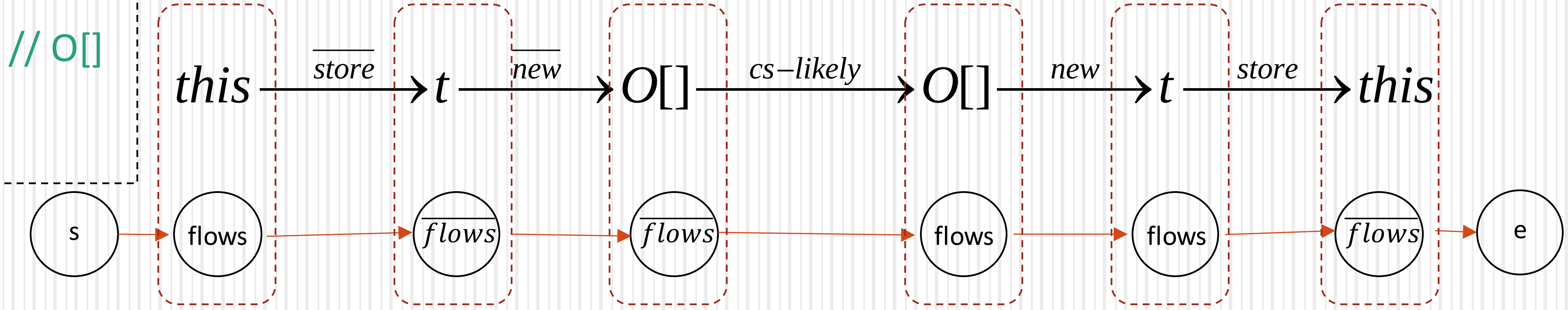


Turner: Object Reachability Analysis




```
class Integer { int v; }
class ArrayList {
    Object[] d;
    ArrayList() {
        Object[] t = new Object[10]; // O[]
        this.d = t;
}}
```

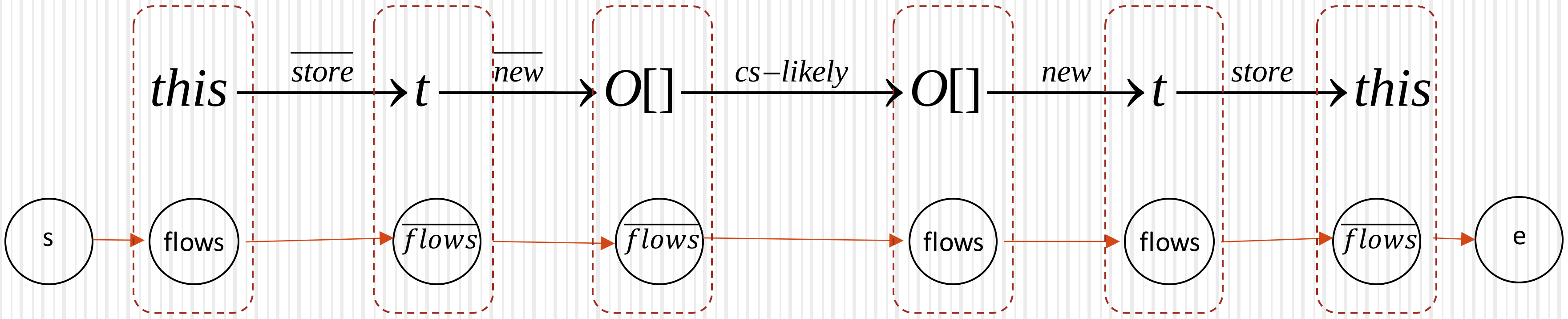
- 1. this, t, and O[] are precision-critical



Other variables/objects could be verified similarly!

Turner: Algorithm

- Exploit antisymmetric property of DFA
  - Criterion: $n \in CS \Leftrightarrow n \in R(flows) \cap R(\overline{flows})$
  - Time complexity: linear to the number of statements



$$\frac{n \in N_M}{n \in R_M(s) \quad s \in R_M^{-1}(n)} \quad \text{[A-I]}$$

$$\frac{n_1 \xrightarrow{\sigma} n_2 \in E_M \quad q_1 \in R_M^{-1}(n_1) \quad \delta(q_1, \sigma) = q_2 \quad q_2 \notin R_M^{-1}(n_2)}{n_2 \in R_M(q_2) \quad q_2 \in R_M^{-1}(n_2)} \quad \text{[A-II]}$$

Implementation

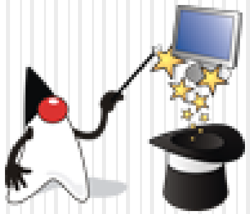
- ❑ Written in Java (~ 1000 LOC)

- ❑ Artifact is deployed on Docker Hub: https://hub.docker.com/r/hdjay2013/turner_artifact

- ❑ Open source: http://www.cse.unsw.edu.au/~corg/turner/

Evaluation

- 12 large Java Programs
  - 9 DaCapo benchmarks
  - 3 popular real-world applications



JPC

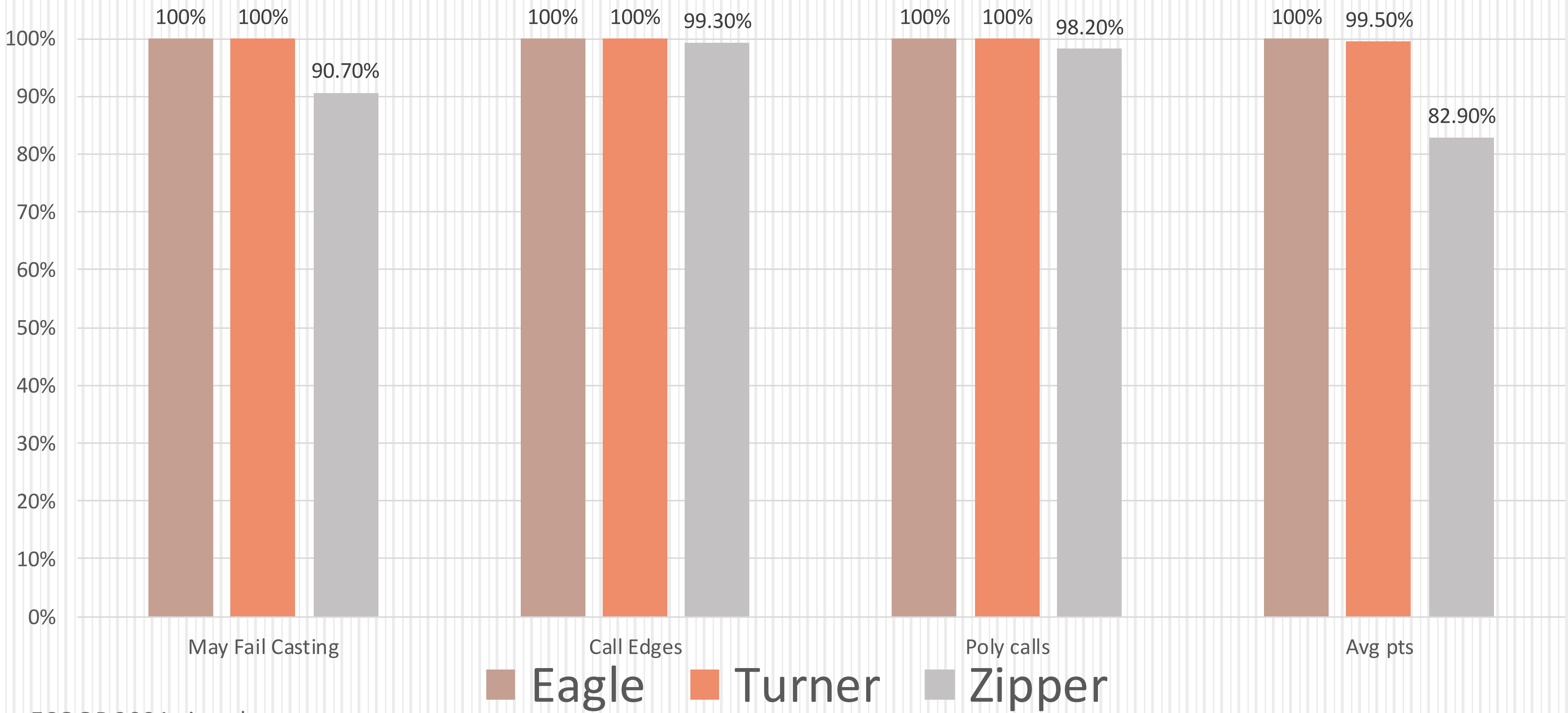
Evaluation

- 4 metrics
  - May-fail casting
  - De-virtualization
  - Call graph construction
  - Average points-to size

Widely-used metrics to evaluate pointer analysis's precision

e.g., OOPSLA'19, OOPSLA'18, PLDI'17, OOPSLA'17, PLDI'14, PLDI'13, POPL'11

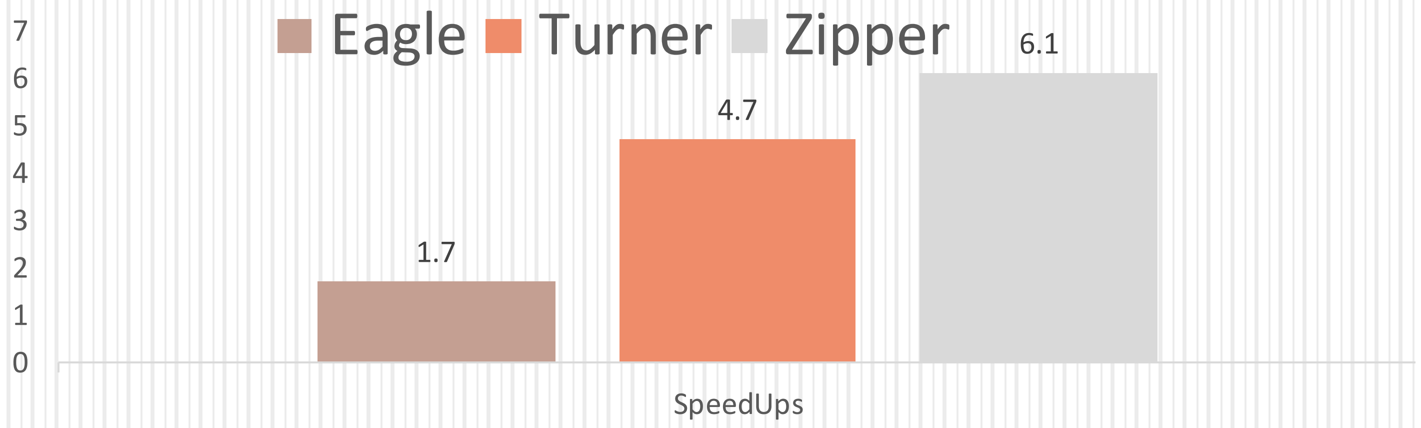
RQ1: precise



100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
100%
100%
90.70%
100%
100%
99.30%
100%
100%
98.20%
100%
99.50%
82.90%
May Fail Casting
Call Edges
Poly calls
Avg pts
Eagle
Turner
Zipper

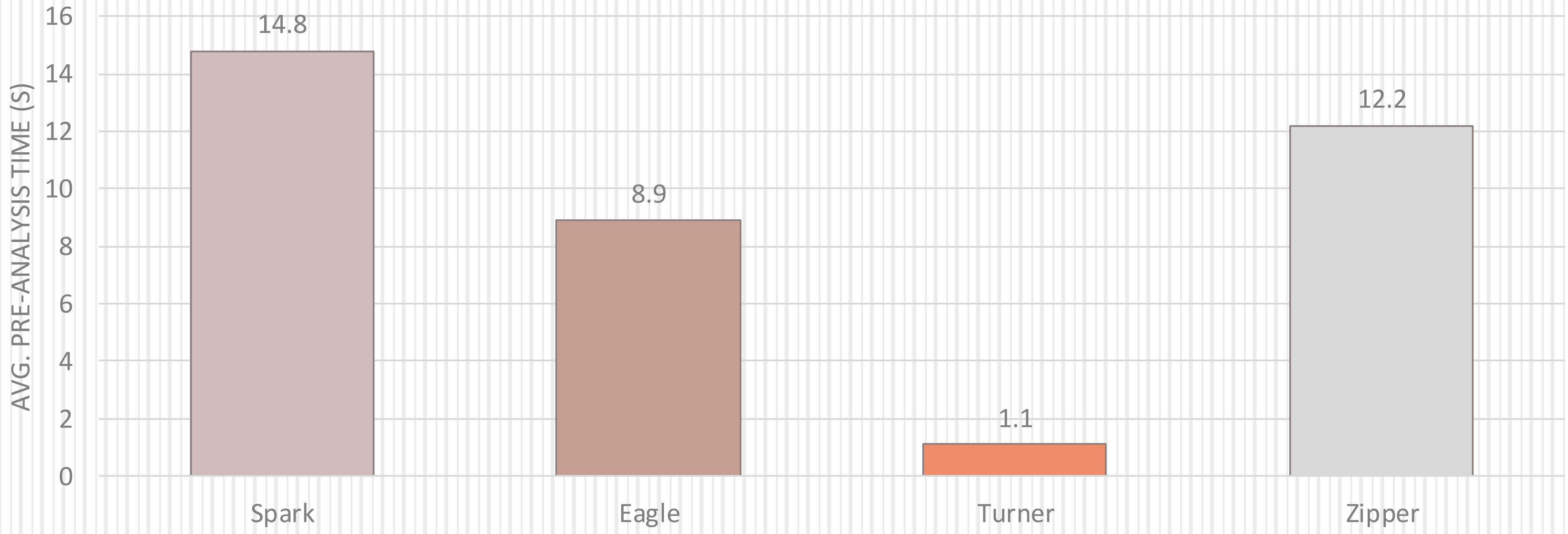
RQ2: efficient

- Average Speedups



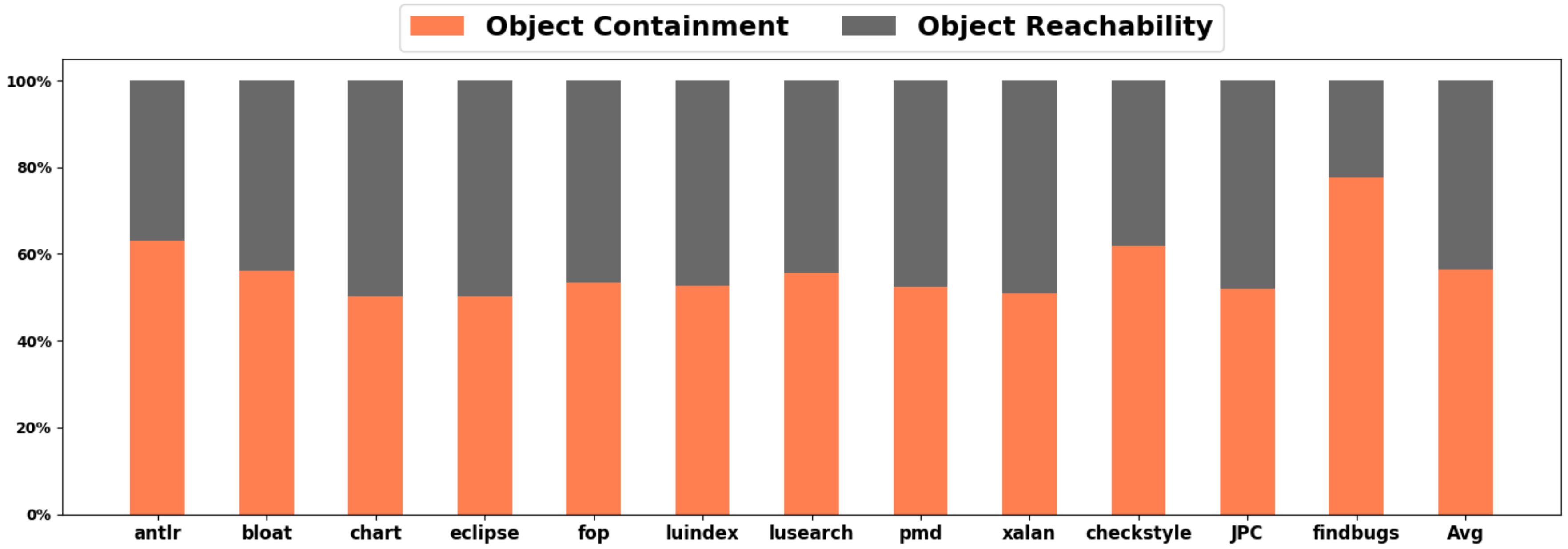
- Scalability
  - Zipper and Turner have the same scalability (10 benchmarks when k = 3).
  - Scale 1 more benchmarks than Eagle, 2 more than baseline.

RQ2: efficient



Very efficient as a pre-analysis.

RQ3: effective



Both object containment and object reachability are effective.

Conclusion

- Turner: Object Containment + Object Reachability
  - first intra-procedural technique.
  - linear to number of statements.
- Implementation
  - open source: http://www.cse.unsw.edu.au/~corg/turner/
  - artifact: https://hub.docker.com/r/hdjay2013/turner_artifact
- Evaluation
  - very precise (preserves almost all the precision)
  - very efficient (not only in guiding pointer analysis but also as a pre-analysis)

Q & A

- Please refer to our paper for technical details!
- Contact: @hdjay20131