Making *k*-Object-Sensitive Pointer Analysis More Precise with Still *k*-Limiting

Tian Tan, Yue Li and Jingling Xue

SAS 2016
September, 2016

A New Pointer Analysis for Object-Oriented Programs

Pointer Analysis

- Determine

  "which objects can a variable point to?"

- Foundation of many clients:
  - Bug detection
  - Security analysis
  - Compiler optimization
  - Program understanding
  - …

Object-Oriented Programs

- Java, C#, Object-C, JavaScript, …

Java

- Embedded software:

- Mobile application:

- Web server:

Apache Tomcat®

redhat JBoss Web Server

- Desktop application:

eclipse

Apache OpenOffice™

A Practically Useful Pointer Analysis for Object-Oriented Programs

A Practically Useful Pointer Analysis for Object-Oriented Programs

Good Context Abstraction (Context Sensitivity)

A Practically Useful
Pointer Analysis
for
Object-Oriented Programs

Good Context Abstraction
(Context Sensitivity)

k-CFA (call-site-sensitivity), type-sensitivity, …

Object-Sensitivity

Arguably the best context abstraction

for

pointer analysis

for

object-oriented programs

Object-Sensitivity

- Widely used in diverse real-world clients
  - Property Verification (e.g., API protocol)

    ISSTA'06, TOSEM'08, PLDI'14, FSE'15, …
  - Bug Detection (e.g., data race, deadlock)

    PLDI'06, ICSE'09, ISSTA'13, OOPSLA'15, …
  - Security Analysis (e.g., taint analysis)

    PLDI'09, IEEE S&P'11, FSE'14, NDSS'15, FSE'15, …
  - Other Fundamental Analyses (e.g., slicing)

    PLDI'07, PLDI'14, ICSE'14, ECOOP'16, …

Object-Sensitivity

- Widely implemented in analysis platforms

Doop

Chord

WALA

T. J. WATSON LIBRARIES FOR ANALYSIS

DroidSafe

TAJS

Sources

F

Sinks

FlowDroid

APPOSCOPY

What is Object-Sensitivity?

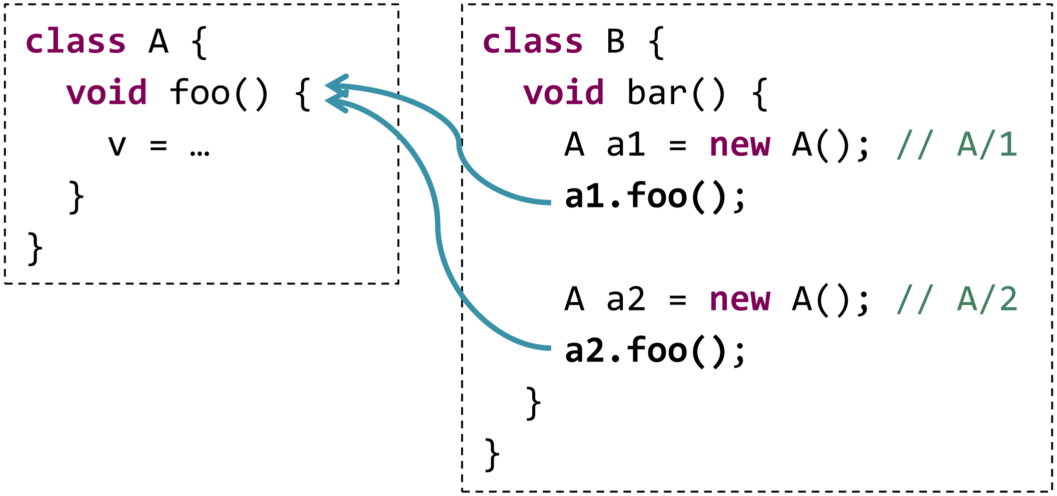
- Objects (allocation sites) as contexts
- *k*-CFA → *k*-obj

A Code Example

```
class A {
  void foo() {
    v = …
  }
}
```

```
class B {
  void bar() {
    A a1 = new A(); // A/1
    a1.foo();

    A a2 = new A(); // A/2
    a2.foo();
  }
}
```

1-CFA
(call-site)

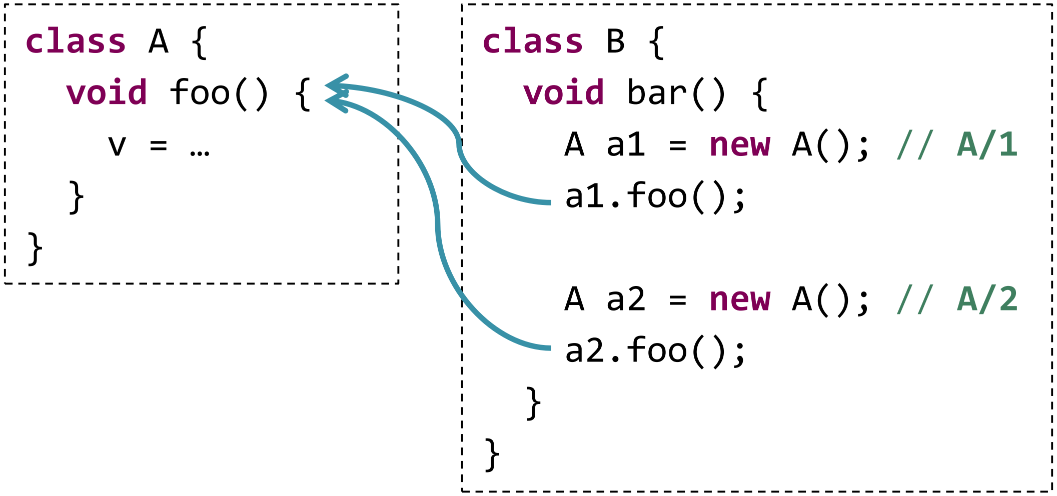


```
class A {
  void foo() {
    v = …
  }
}
```

```
class B {
  void bar() {
    A a1 = new A(); // A/1
    a1.foo();

    A a2 = new A(); // A/2
    a2.foo();
  }
}
```

Context	Variable	Object
[a1.foo()]	v	…
[a2.foo()]	v	…

1-obj
(allocation-site of receiver object)

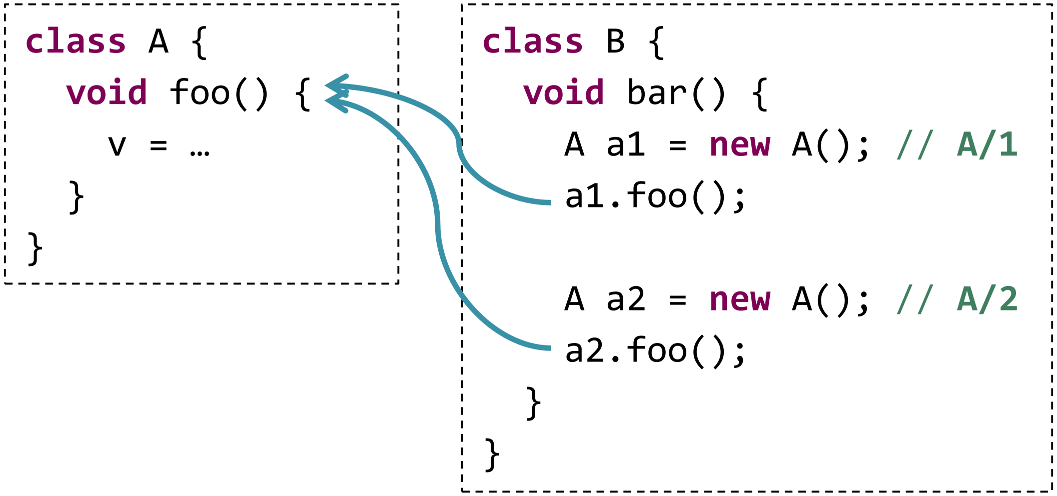


```
class A {
  void foo() {
    v = …
  }
}
```

```
class B {
  void bar() {
    A a1 = new A(); // A/1
    a1.foo();

    A a2 = new A(); // A/2
    a2.foo();
  }
}
```

Context	Variable	Object
[A/1]	v	…
[A/2]	v	…

k-obj when *k* > 1?

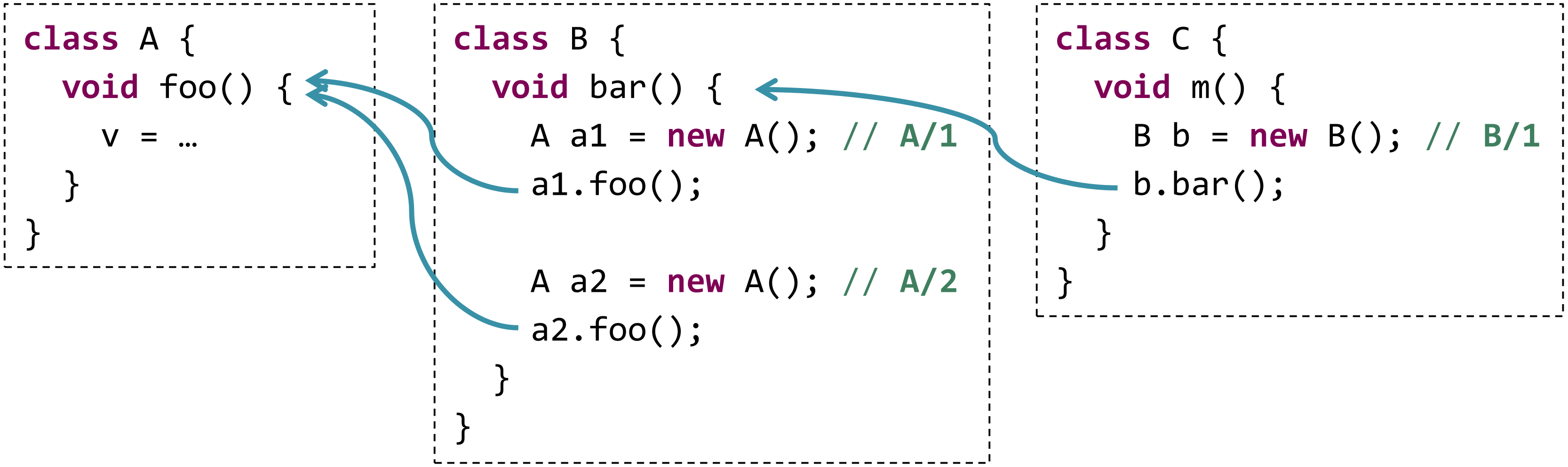


```
class A {
  void foo() {
    v = …
  }
}
```

```
class B {
  void bar() {
    A a1 = new A(); // A/1
    a1.foo();

    A a2 = new A(); // A/2
    a2.foo();
  }
}
```

2-obj

(allocation-sites of 2 “consecutive” receiver objects)



```
class A {
  void foo() {
    v = …
  }
}
```

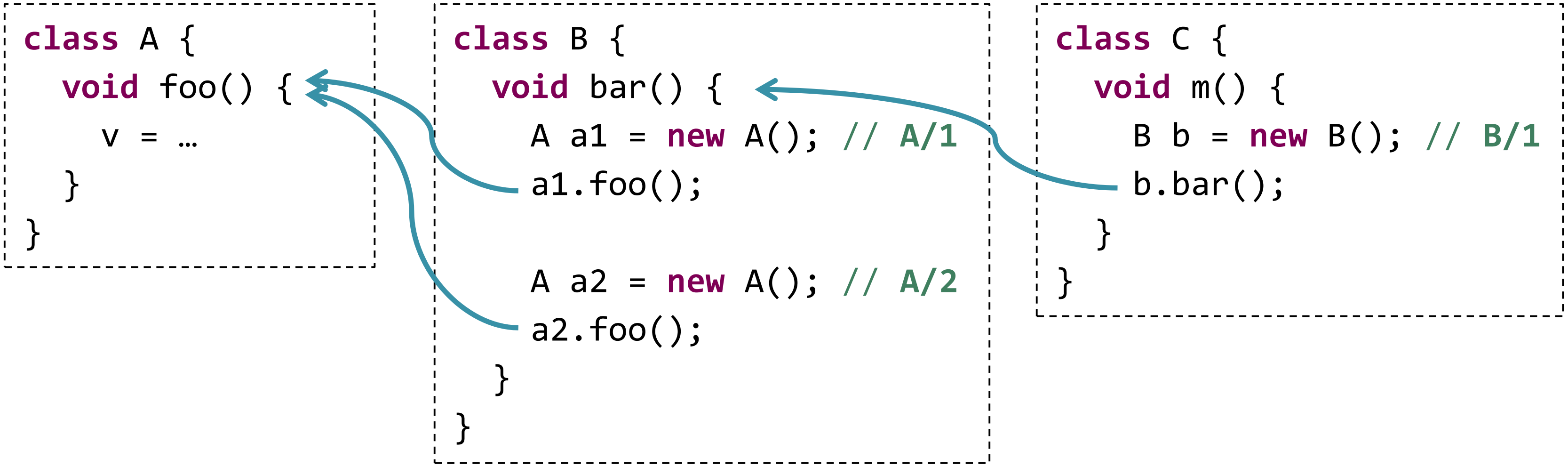
```
class B {
  void bar() {
    A a1 = new A(); // A/1
    a1.foo();

    A a2 = new A(); // A/2
    a2.foo();
  }
}
```

```
class C {
  void m() {
    B b = new B(); // B/1
    b.bar();
  }
}
```

Context	Variable	Object
[B/1,A/1]	v	…
[B/1,A/2]	v	…

2-obj

(allocation-sites of 2 “consecutive” receiver objects)

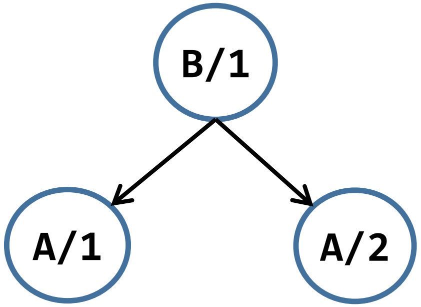


```
class A {
  void foo() {
    v = …
  }
}
```

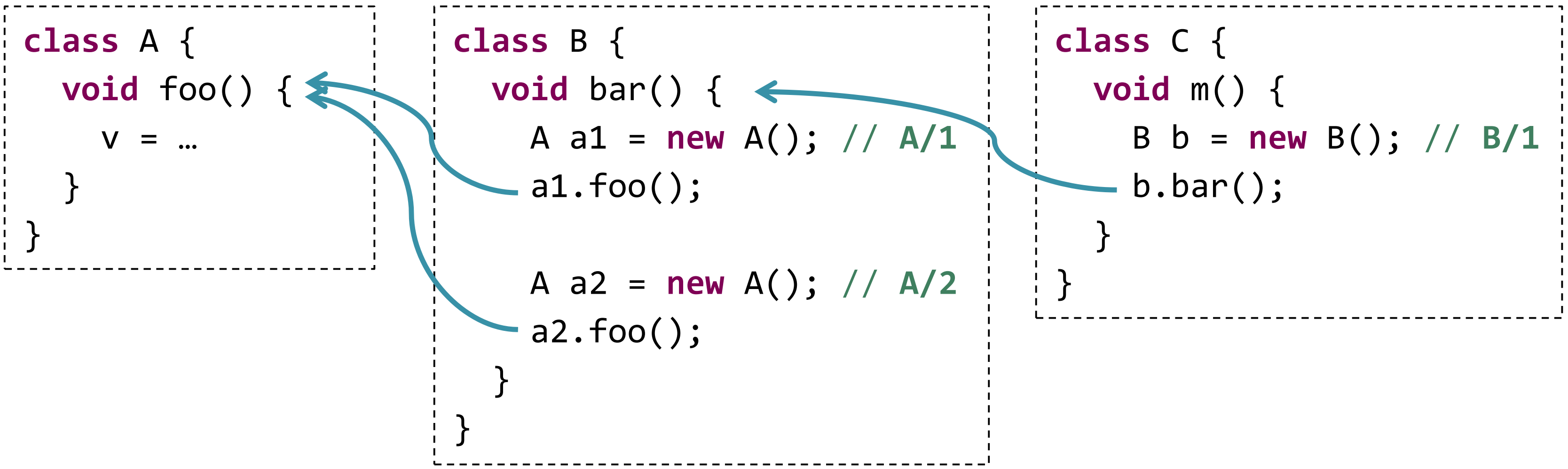
```
class B {
  void bar() {
    A a1 = new A(); // A/1
    a1.foo();

    A a2 = new A(); // A/2
    a2.foo();
  }
}
```

```
class C {
  void m() {
    B b = new B(); // B/1
    b.bar();
  }
}
```

Context	Variable	Object
[B/1,A/1]	v	…
[B/1,A/2]	v	…



2-obj
(allocation-sites of 2 “consecutive” receiver objects)

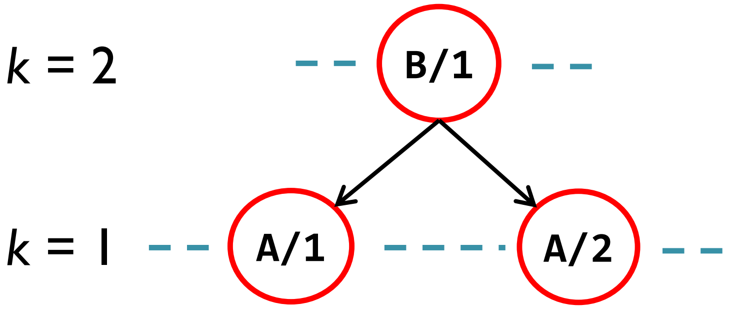


```
class A {
  void foo() {
    v = …
  }
}
```

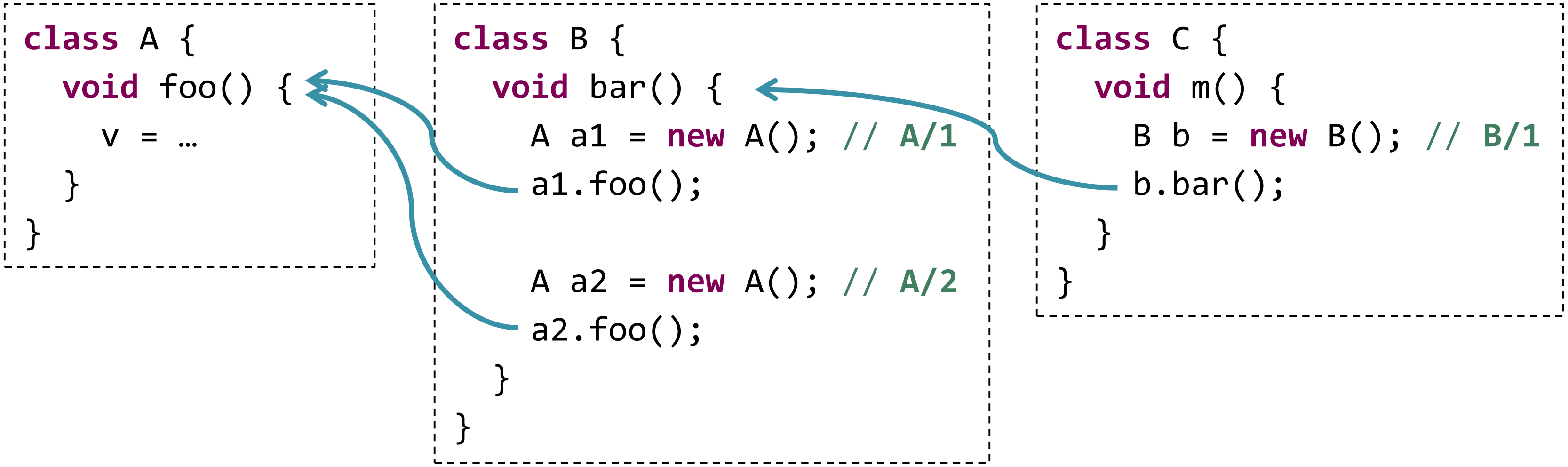
```
class B {
  void bar() {
    A a1 = new A(); // A/1
    a1.foo();

    A a2 = new A(); // A/2
    a2.foo();
  }
}
```

```
class C {
  void m() {
    B b = new B(); // B/1
    b.bar();
  }
}
```

Context	Variable	Object
[B/1,A/1]	v	…
[B/1,A/2]	v	…



2-obj
(allocation-sites of 2 "consecutive" receiver objects)

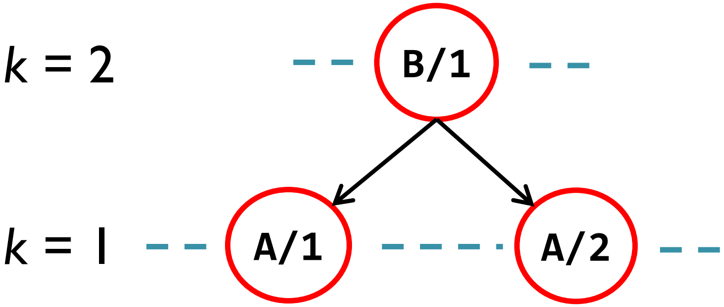


```
class A {
  void foo() {
    v = …
  }
}
```

```
class B {
  void bar() {
    A a1 = new A(); // A/1
    a1.foo();

    A a2 = new A(); // A/2
    a2.foo();
  }
}
```

```
class C {
  void m() {
    B b = new B(); // B/1
    b.bar();
  }
}
```

Context	Variable	Object
[B/1,A/1]	v	…
[B/1,A/2]	v	…



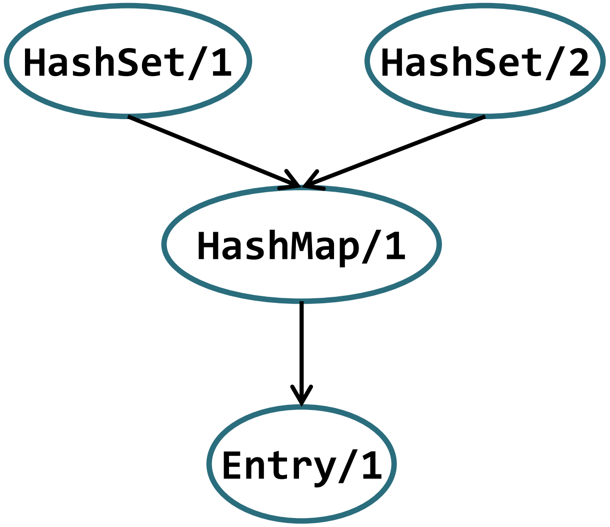
Object Allocation Graph (OAG)

An Observation

- Redundant Context Element

An Observation

- Redundant Context Element

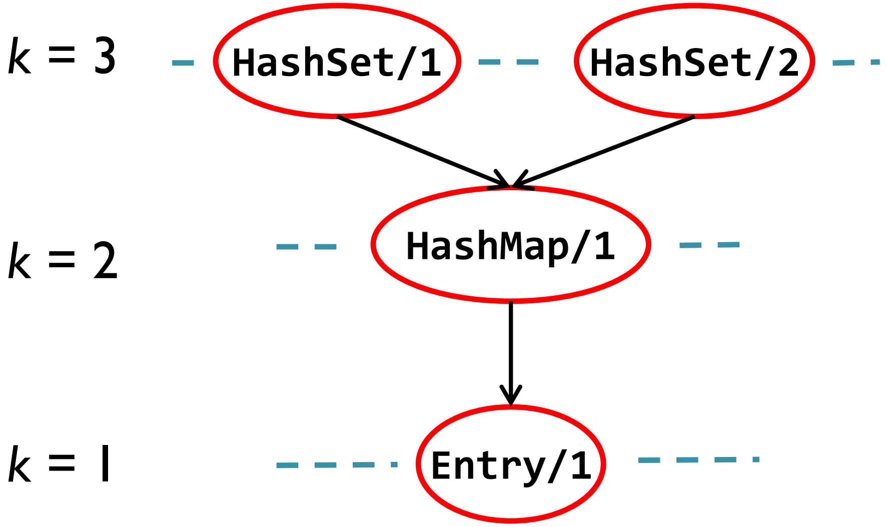


An example from JDK, **java.util.***

3-obj

- Contexts fully separated
- Precise

Two contexts:
[HashSet/1,HashMap/1,Entry/1]
[HashSet/2,HashMap/1,Entry/1]



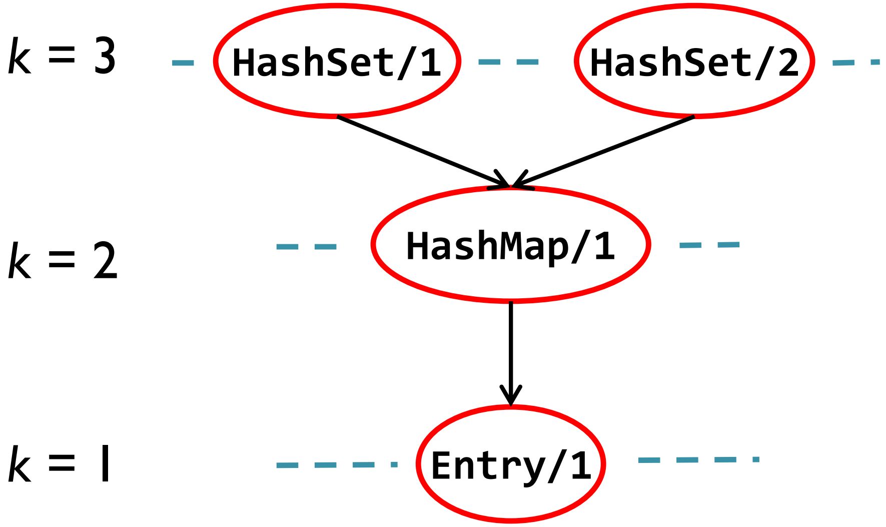
An example from JDK, **java.util.***

3-obj

- Contexts fully separated
- Precise

Two contexts:
[HashSet/1,HashMap/1,Entry/1]
[HashSet/2,HashMap/1,Entry/1]

3-obj is **unscalable**

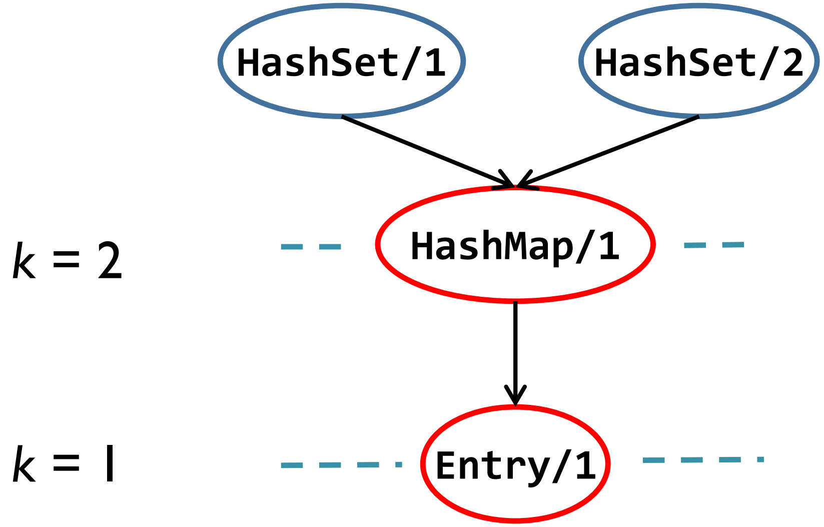


An example from JDK, **java.util.***

2-obj

- Contexts not separated

One context:
[HashMap/1,Entry/1]

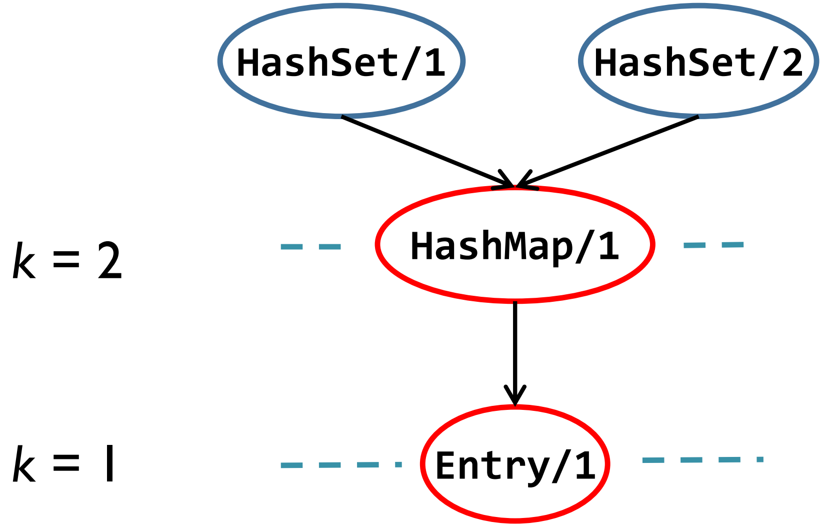


An example from JDK, **java.util.***

2-obj

- Contexts not separated
- Imprecise

One context:
[HashMap/1,Entry/1]



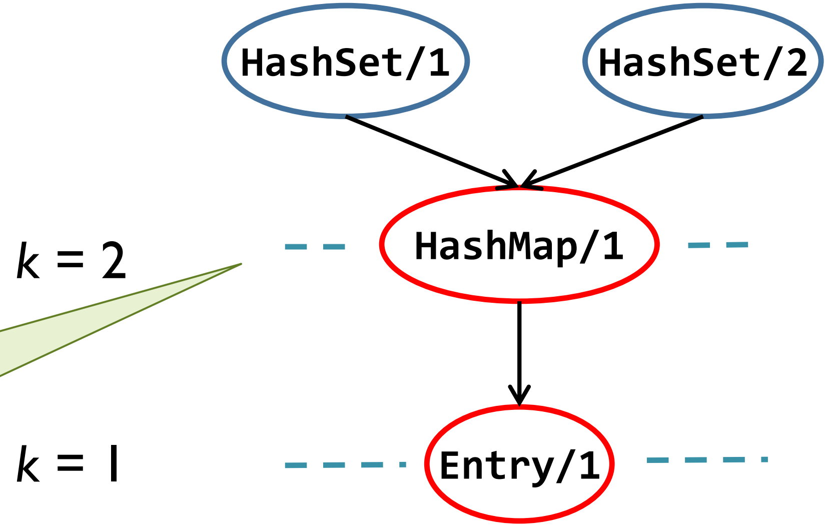
An example from JDK, **java.util.***

2-obj

- Contexts not separated
- Imprecise
- Redundant context elements used

One context:
[**HashMap/1**,Entry/1]

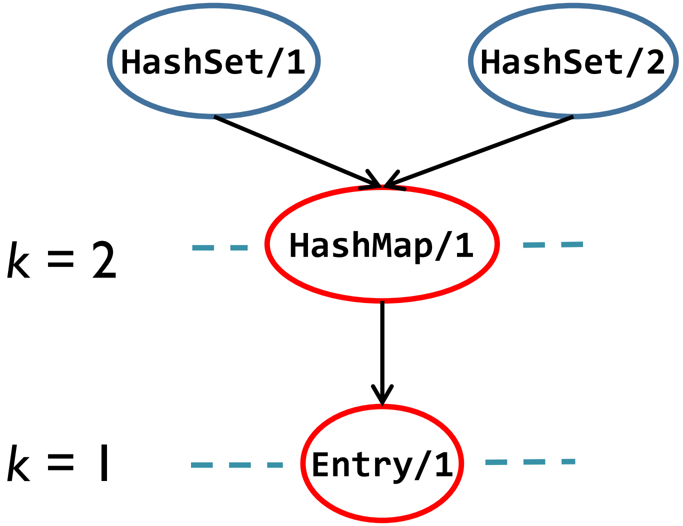
HashMap/1 as context element is <span>redundant</span>



An example from JDK, **java.util.***

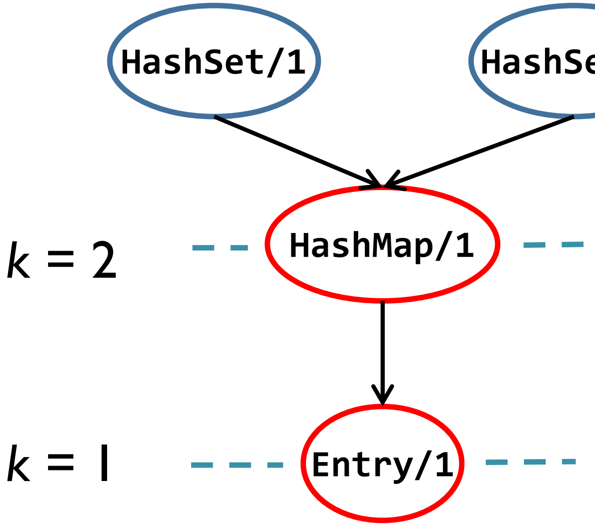
This Paper:
Avoid
Redundant Context Element

2-obj



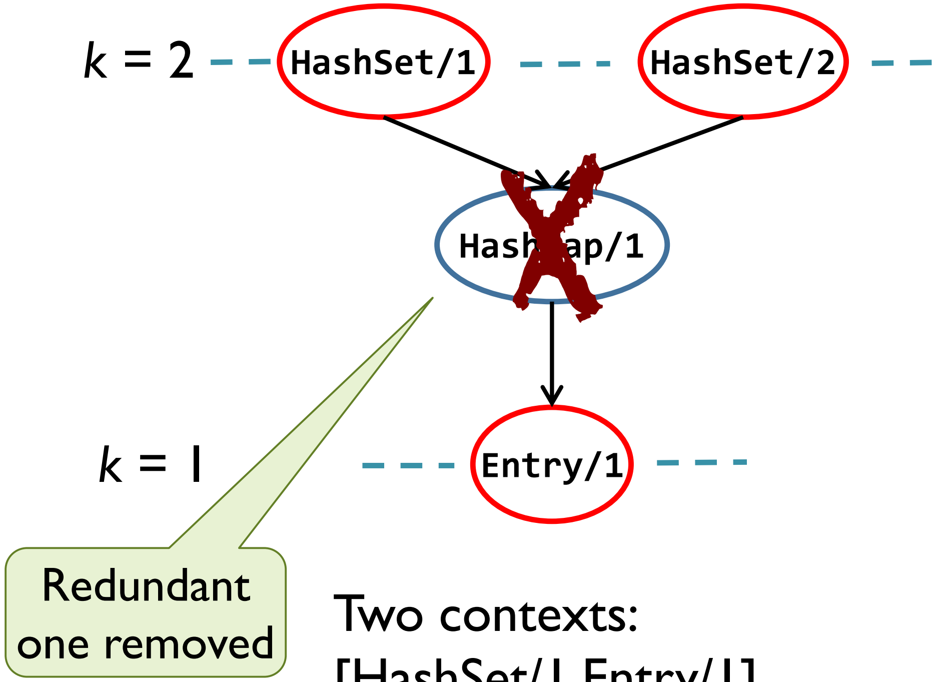
One context:
[HashMap/1,Entry/1]

2-obj



One context:
[HashMap/1,Entry/1]

Our approach

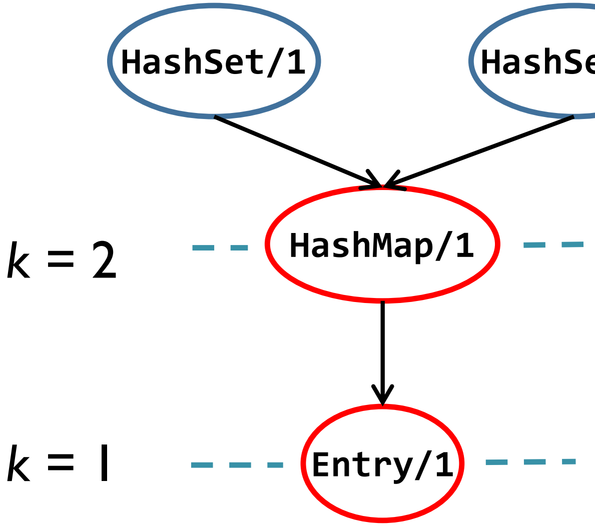


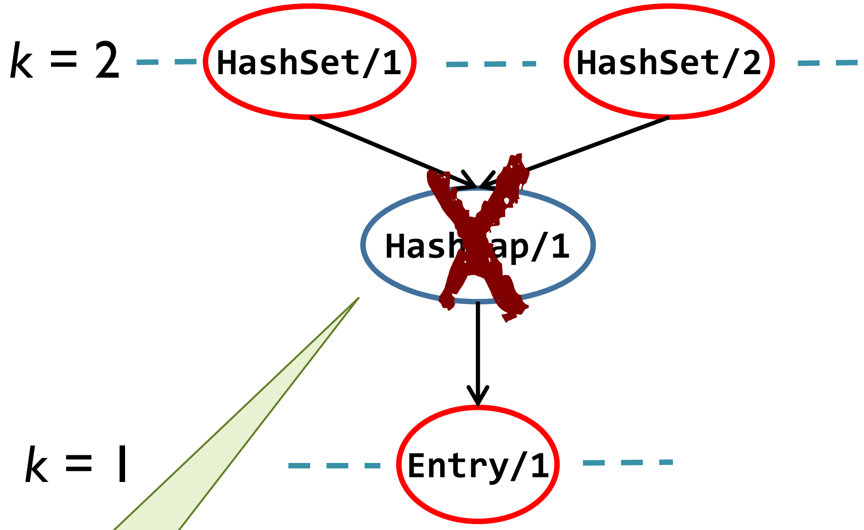
Redundant one removed

Two contexts:
[HashSet/1,Entry/1]
[HashSet/2,Entry/1]

2-obj

Our approach





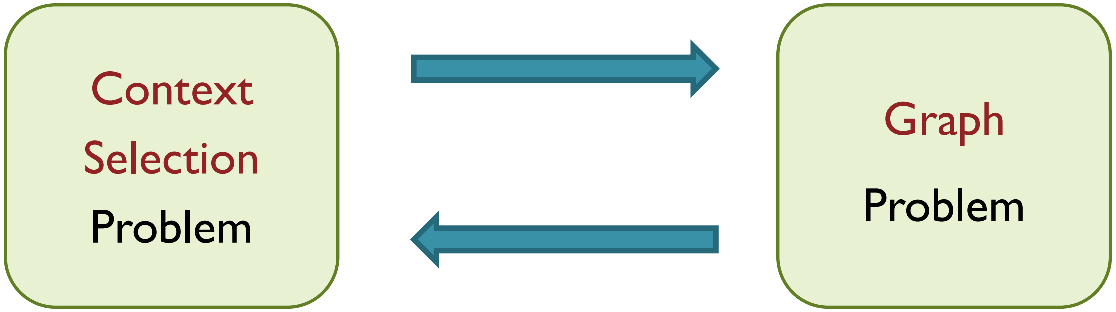
One context:
[HashMap/1,Entry/1]

Redundant
one removed

Two contexts:
[HashSet/1,Entry/1]
[HashSet/2,Entry/1]

Benefit: improve precision
with still *k*-limiting

Methodology (BEAN)



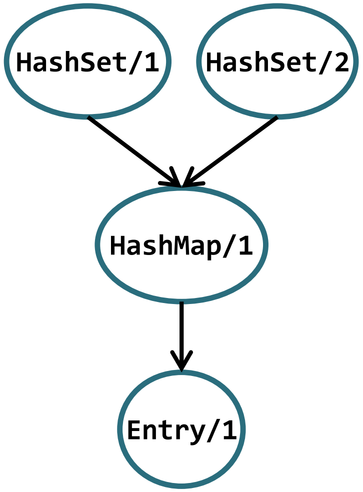
Context Selection Problem

Graph Problem

Context Selection Problem

Graph Problem

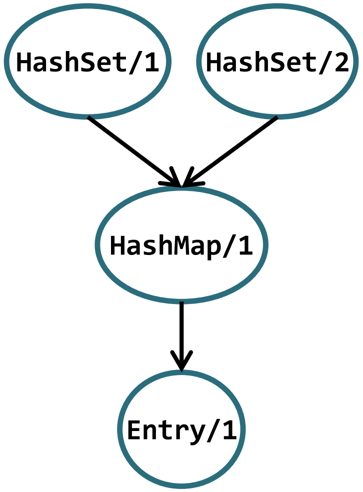
Context Relation ⟺ Object Allocation Graph (OAG)



Context Selection Problem

Graph Problem

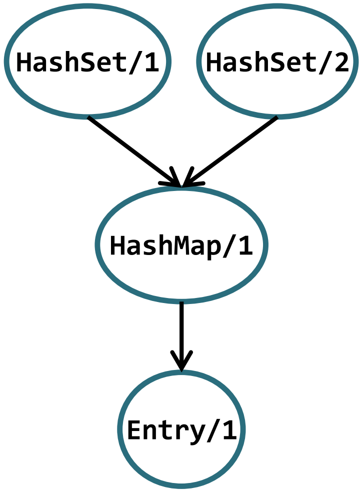
Context Relation ⟺ Object Allocation Graph (OAG)



Contexts in *k*-obj ⟺ Paths in OAG

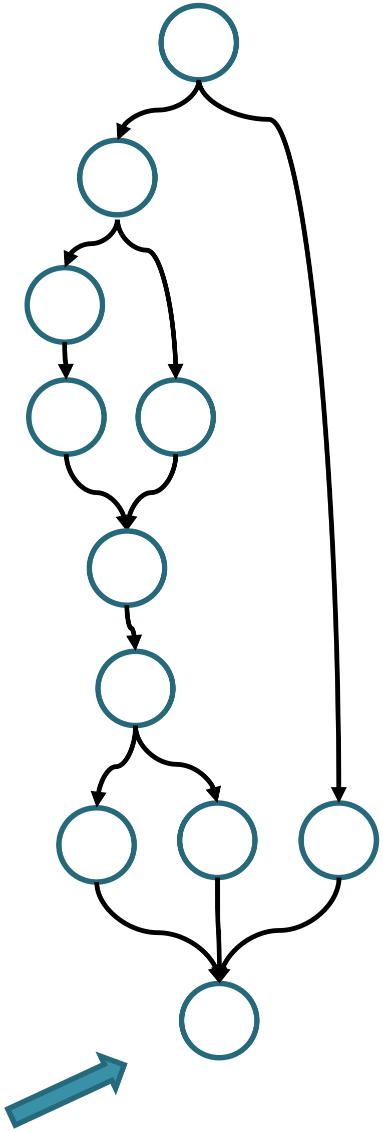
Context Selection Problem ⟺ Graph Problem

Context Relation ⟺ Object Allocation Graph (OAG)



Contexts in *k*-obj ⟺ Paths in OAG

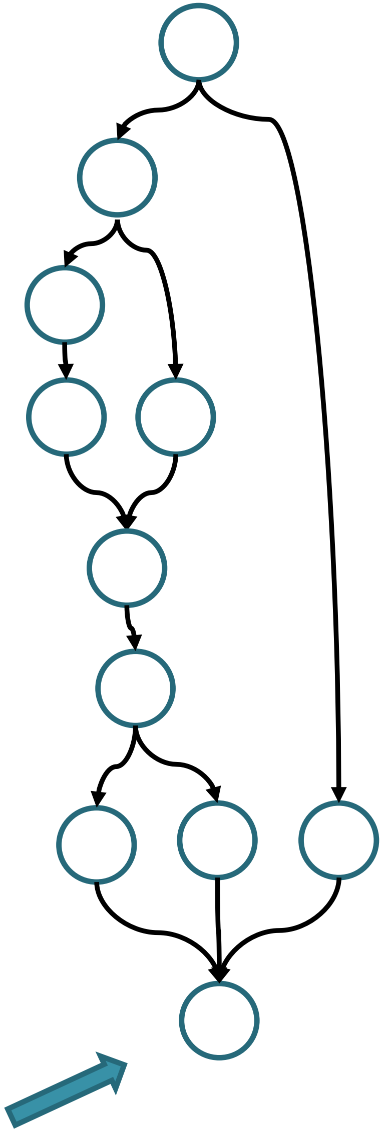
Avoid Redundant Context Elements ⟺ Select Representative Nodes to Distinguish Paths



An OAG

5 contexts in *k*-obj

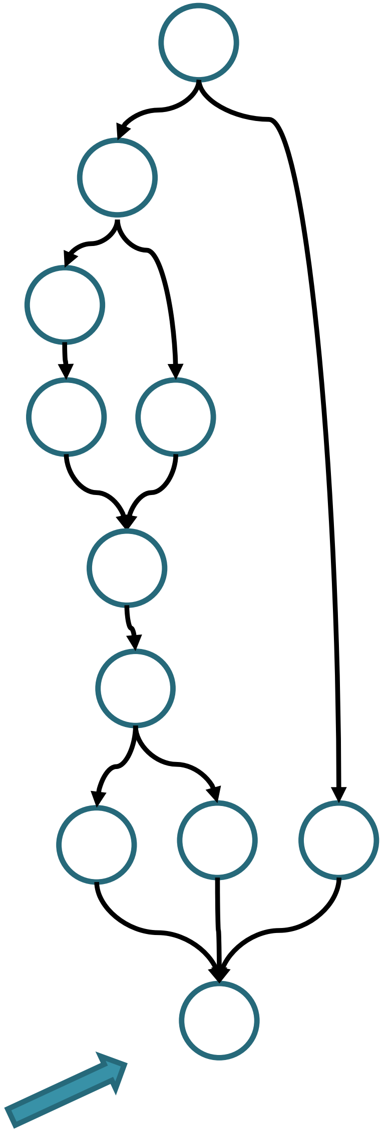
5 paths in OAG



An OAG

Select 5 contexts in *k*-obj

Distinguish 5 paths in OAG



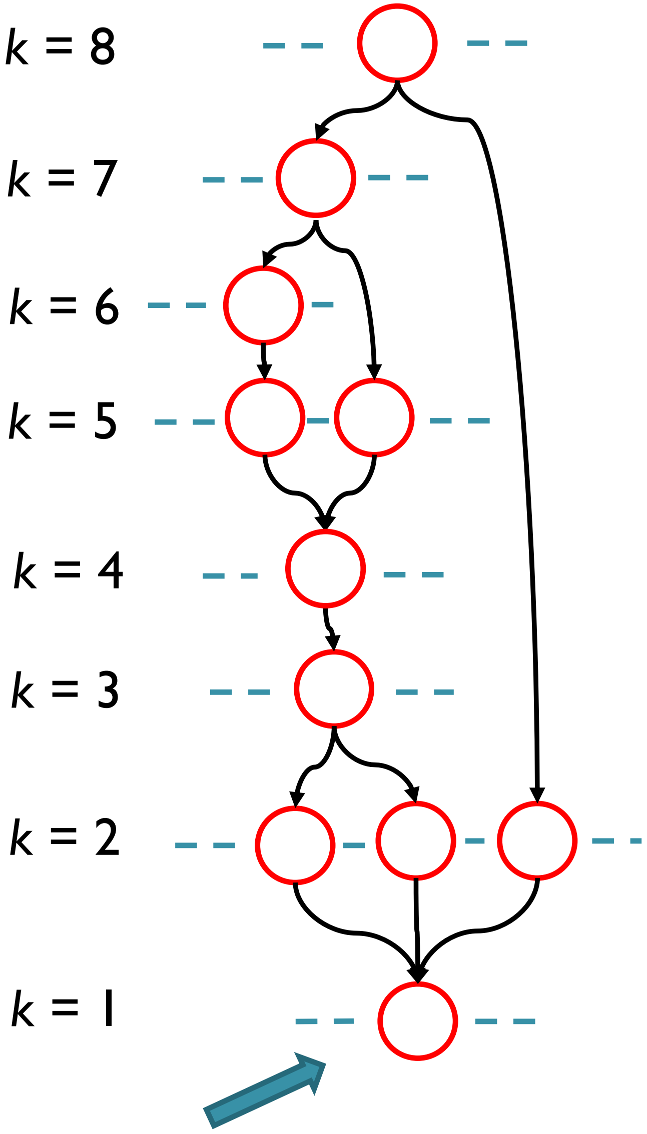
An OAG

Select 5 contexts in *k*-obj

Distinguish 5 paths in OAG

k-obj: *k* = 8

(all nodes selected)



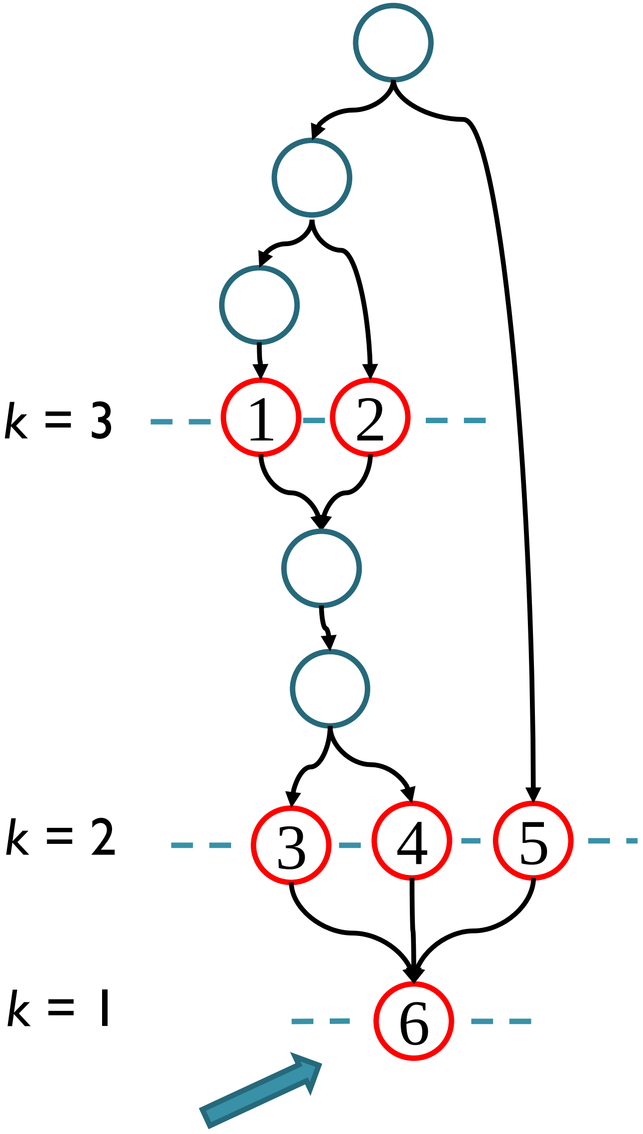
An OAG

Select 5 contexts in *k*-obj

Distinguish 5 paths in OAG

k-obj: *k* = 8
(all nodes selected)

BEAN: *k* = 3
(representative nodes selected)



An OAG

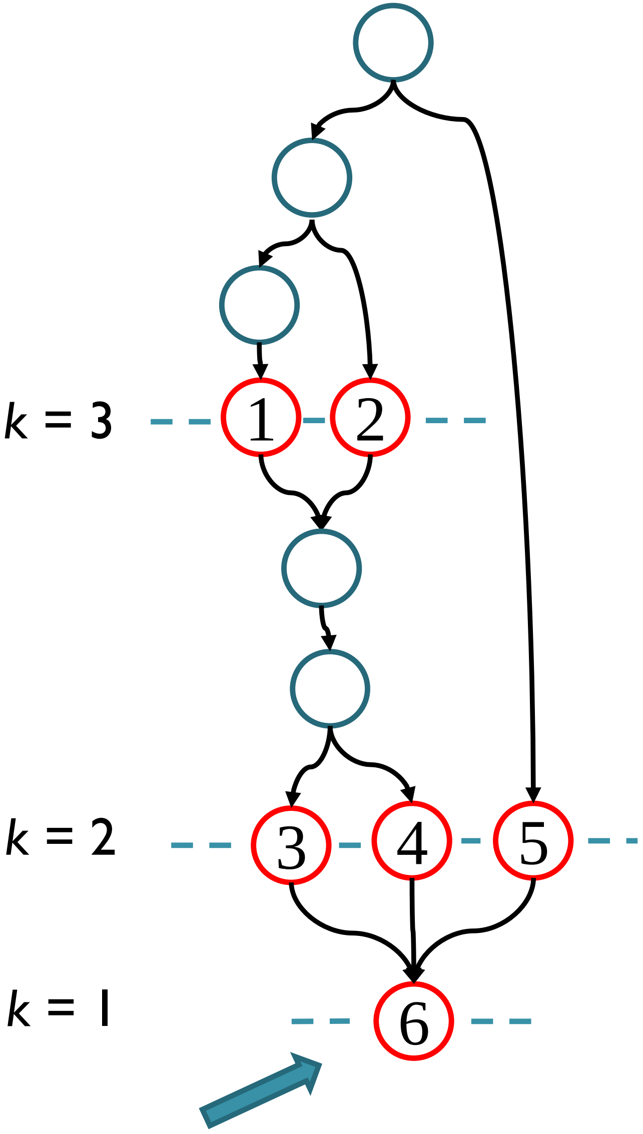
Select 5 contexts in *k*-obj

Distinguish 5 paths in OAG

k-obj: *k* = 8
(all nodes selected)

BEAN: *k* = 3
(representative nodes selected)

5 contexts selected by BEAN:
[1,3,6], [2,3,6],
[1,4,6], [2,4,6], [5,6]



An OAG

Select 5 contexts in *k*-obj

Distinguish 5 paths in OAG

k-obj: *k* = 8

(all nodes selected)

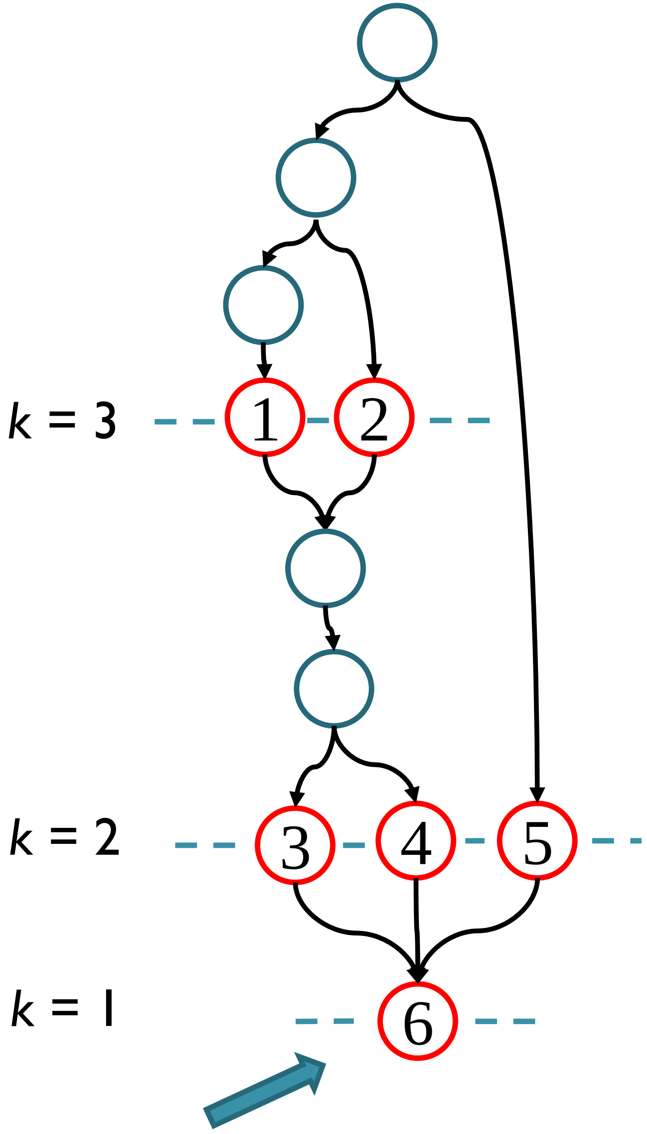
|| precision

BEAN: *k* = 3

(representative nodes selected)

5 contexts selected by BEAN:

[1,3,6], [2,3,6],
[1,4,6], [2,4,6], [5,6]



An OAG

How to Select Representative Nodes to Distinguish Paths?

How to Select Representative Nodes to Distinguish Paths?

- Our intuition:

Multiple paths

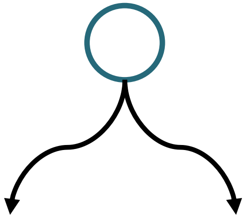
How to Select Representative Nodes to Distinguish Paths?

- Our intuition:

Multiple paths

=

Divergence



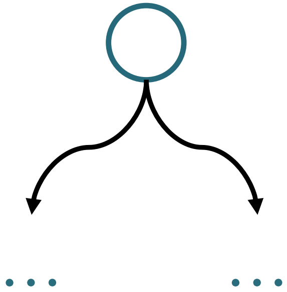
How to Select Representative Nodes to Distinguish Paths?

- Our intuition:

Multiple paths

||

Divergence



+

How to Select Representative Nodes to Distinguish Paths?

- Our intuition:

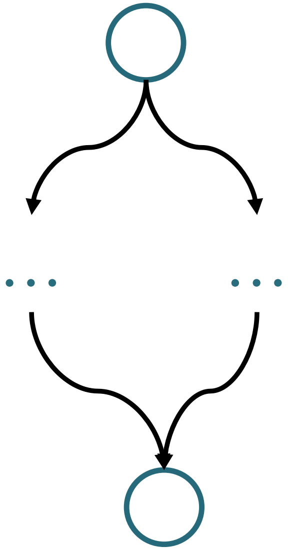
Multiple paths

||

Divergence

+

Confluence



How to Select Representative Nodes to Distinguish Paths?

- Our intuition:

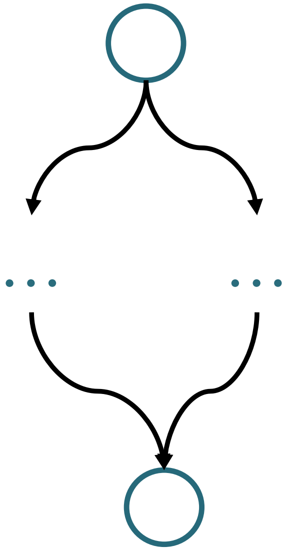
Multiple paths

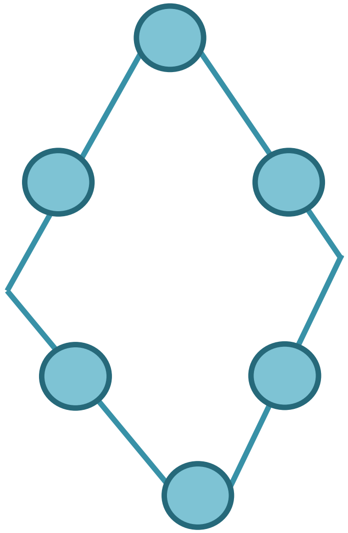
||

Divergence

+

Confluence





How to Select Representative Nodes to Distinguish Paths?

- Our intuition:

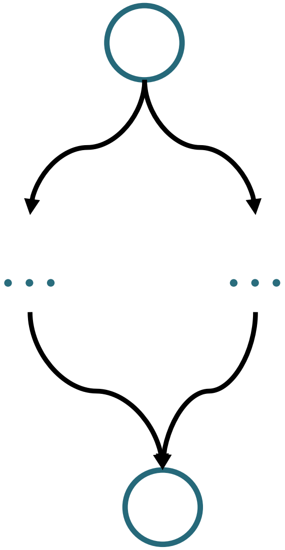
Multiple paths

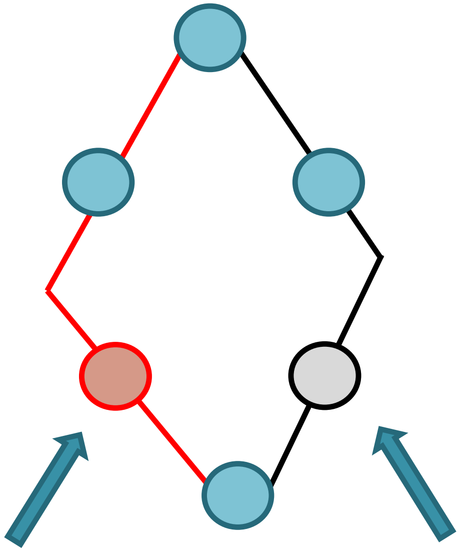
II

Divergence

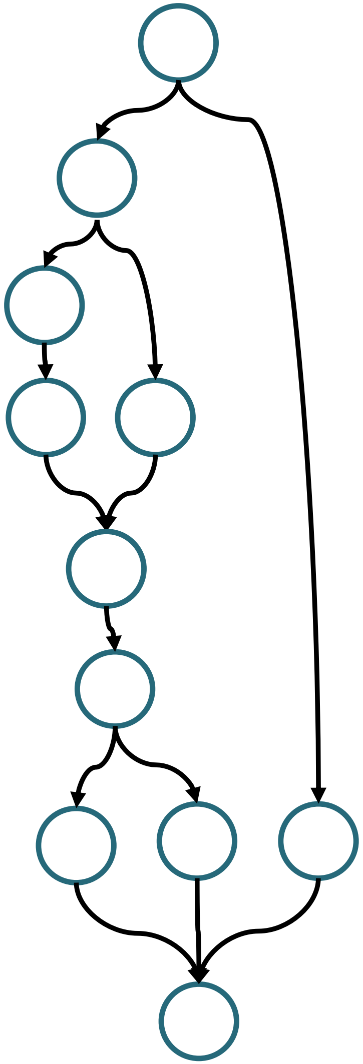
+

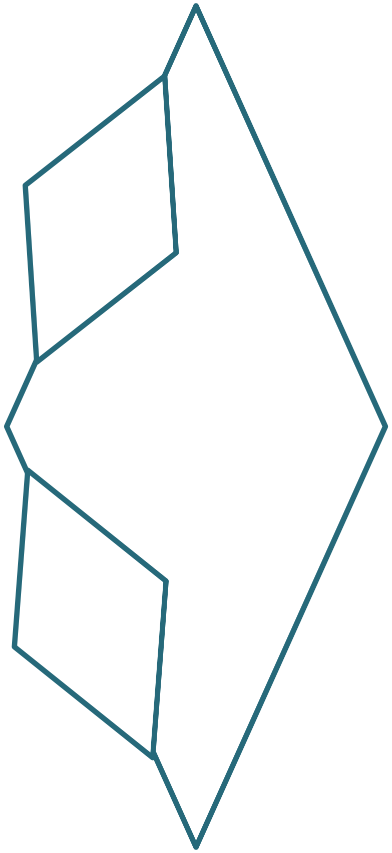
Confluence

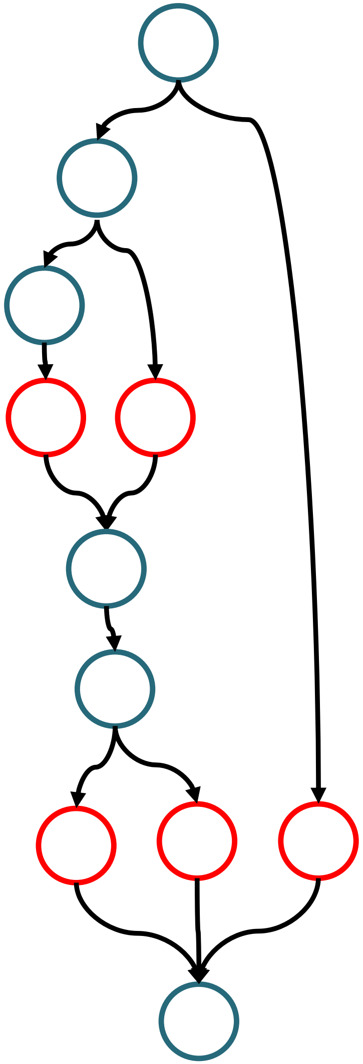


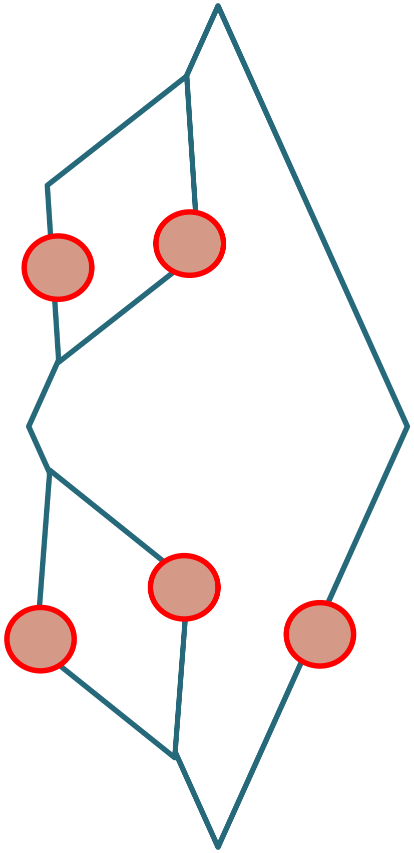


Representative
nodes









Representative nodes

Theorem 1

- Under *full*-object-sensitivity (when *k* = ∞)

Precision of BEAN	=	Precision of *k*-obj

Theorem 2

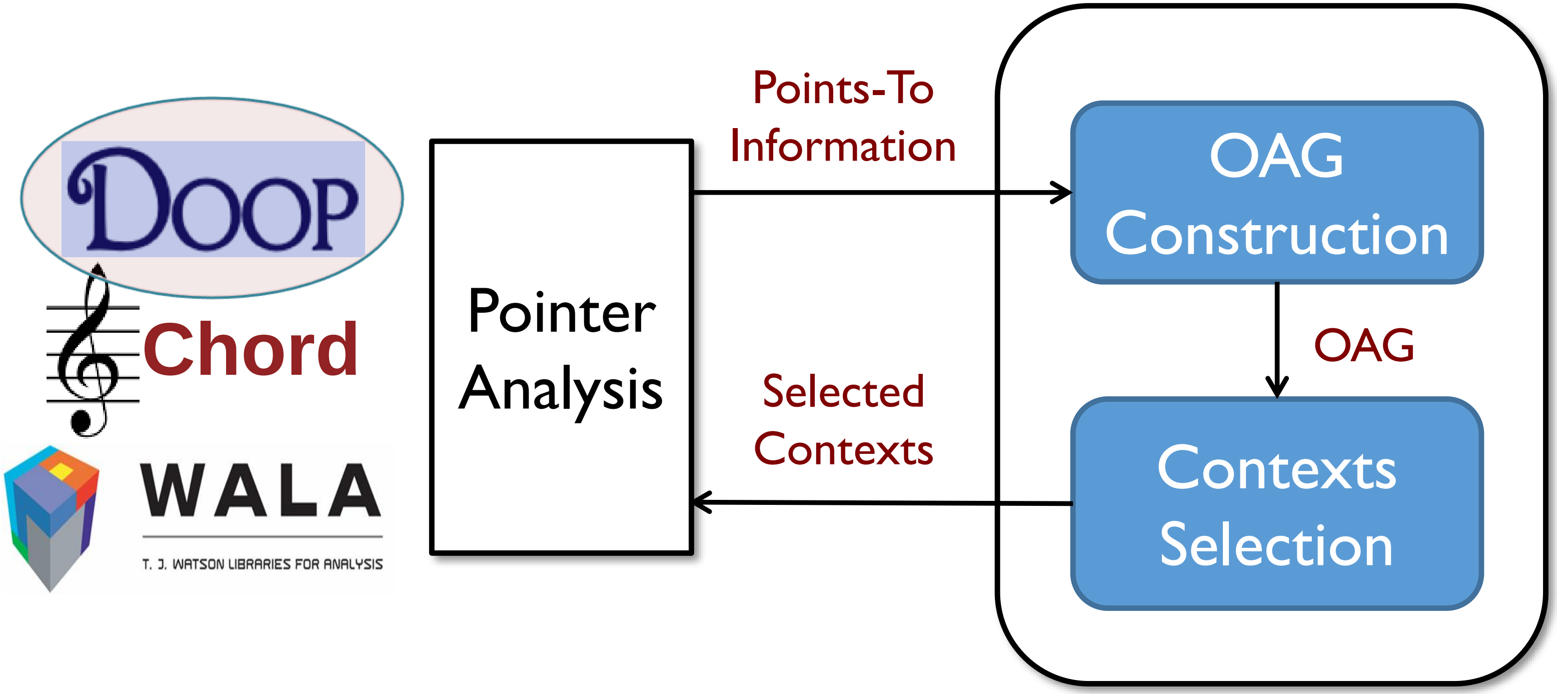
- Under the same *k*-limiting

Precision
of
BEAN

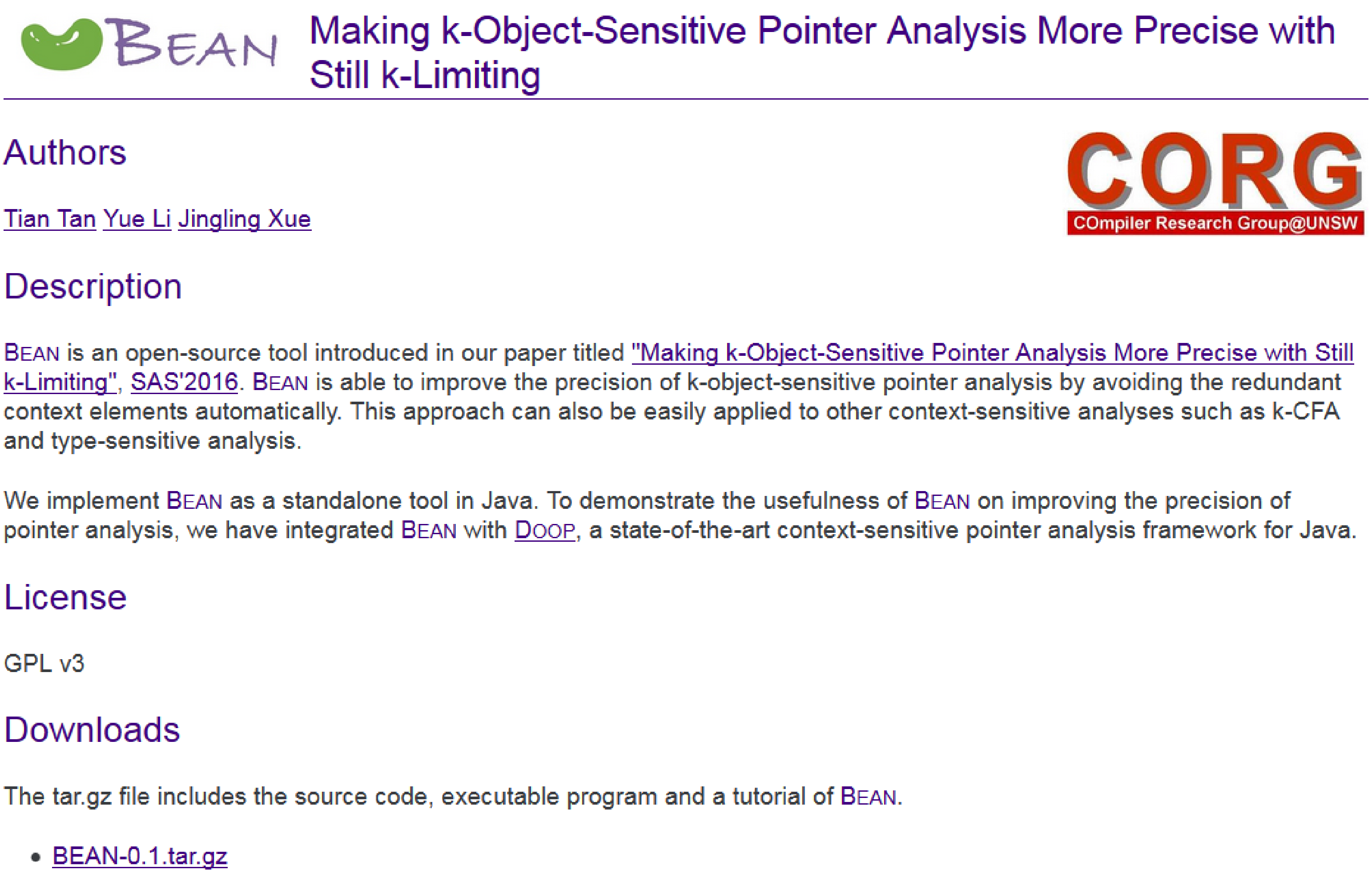
≥

Precision
of
k-obj

BEAN: Framework



Open-Source Implementation



BEAN Making k-Object-Sensitive Pointer Analysis More Precise with Still k-Limiting

Authors

Tian Tan Yue Li Jingling Xue

Description

BEAN is an open-source tool introduced in our paper titled "Making k-Object-Sensitive Pointer Analysis More Precise with Still k-Limiting", SAS'2016. BEAN is able to improve the precision of k-object-sensitive pointer analysis by avoiding the redundant context elements automatically. This approach can also be easily applied to other context-sensitive analyses such as k-CFA and type-sensitive analysis.

We implement BEAN as a standalone tool in Java. To demonstrate the usefulness of BEAN on improving the precision of pointer analysis, we have integrated BEAN with DOOP, a state-of-the-art context-sensitive pointer analysis framework for Java.

License

GPL v3

Downloads

The tar.gz file includes the source code, executable program and a tutorial of BEAN.

- BEAN-0.1.tar.gz

www.cse.unsw.edu.au/~corg/bean

Evaluation - Clients

- May-Alias
- May-Fail-Cast

Typical clients to evaluate pointer analysis's effectiveness
e.g., APLAS'15, PLDI'14, PLDI'13, POPL'11, OOPSLA'09, …

Evaluation - Analyzed Targets

- Standard DaCapo Java benchmarks
- Large Java library: JDK 1.6

Widely used programs and library in pointer analysis
e.g., PLDI'14, ECOOP'14, PLDI'13, OOPSLA'13, POPL'11, …

Evaluation - Compared Analyses

1. 2-CFA: 2-call-site-sensitive analysis
2. 2-obj: 2-object-sensitive analysis
3. B-2-obj: BEAN-directed 2-obj
4. S-2-obj: Selective hybrids of 2-obj*
5. B-S-2-obj: BEAN-directed S-2-obj

* *Kastrinis et al., Hybrid Context-Sensitivity for Points-To Analysis, PLDI'13*

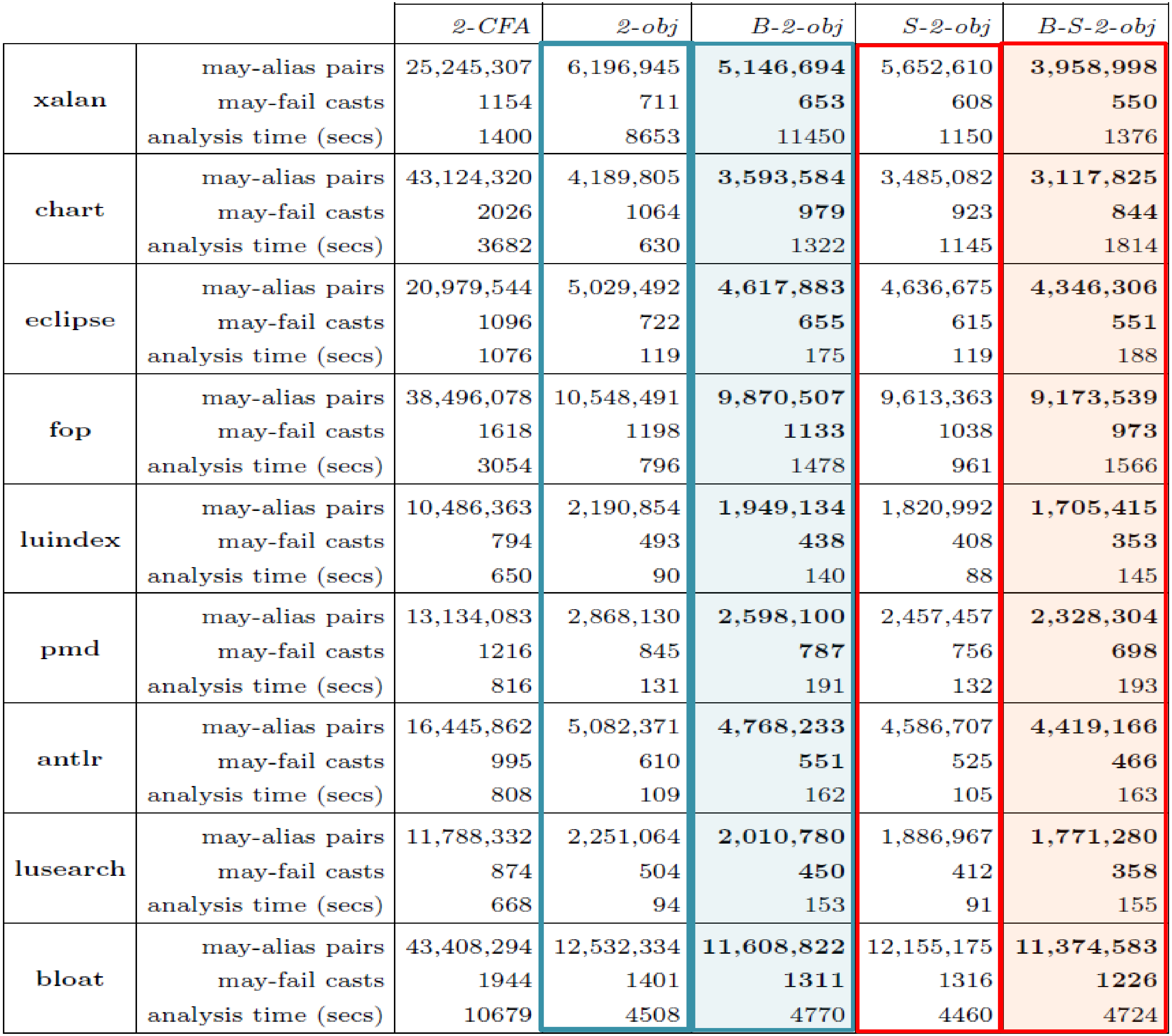
Evaluation - Metrics

- Precision
- Performance

Precision

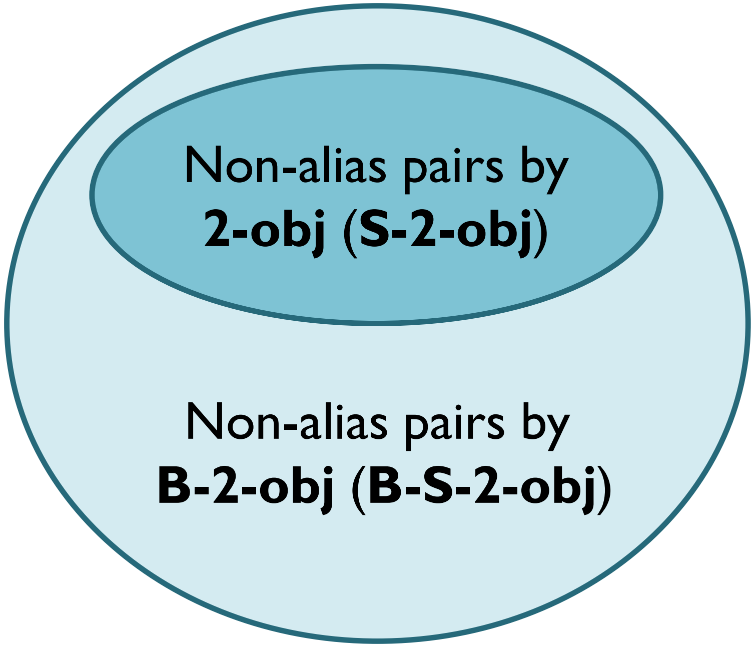
- 2 clients
- 5 pointer analyses (2 state-of-the-art)
- 9 evaluated Java programs

BEAN improves the precision
of both state-of-the-art analyses,
under each client,
for each program!



		2-CFA	*2-obj*	*B-2-obj*	*S-2-obj*	*B-S-2-obj*
xalan	may-alias pairs	25,245,307	6,196,945	**5,146,694**	5,652,610	**3,958,998**
	may-fail casts	1154	711	**653**	608	**550**
	analysis time (secs)	1400	8653	11450	1150	1376
chart	may-alias pairs	43,124,320	4,189,805	**3,593,584**	3,485,082	**3,117,825**
	may-fail casts	2026	1064	**979**	923	**844**
	analysis time (secs)	3682	630	1322	1145	1814
eclipse	may-alias pairs	20,979,544	5,029,492	**4,617,883**	4,636,675	**4,346,306**
	may-fail casts	1096	722	**655**	615	**551**
	analysis time (secs)	1076	119	175	119	188
fop	may-alias pairs	38,496,078	10,548,491	**9,870,507**	9,613,363	**9,173,539**
	may-fail casts	1618	1198	**1133**	1038	**973**
	analysis time (secs)	3054	796	1478	961	1566
luindex	may-alias pairs	10,486,363	2,190,854	**1,949,134**	1,820,992	**1,705,415**
	may-fail casts	794	493	**438**	408	**353**
	analysis time (secs)	650	90	140	88	145
pmd	may-alias pairs	13,134,083	2,868,130	**2,598,100**	2,457,457	**2,328,304**
	may-fail casts	1216	845	**787**	756	**698**
	analysis time (secs)	816	131	191	132	193
antlr	may-alias pairs	16,445,862	5,082,371	**4,768,233**	4,586,707	**4,419,166**
	may-fail casts	995	610	**551**	525	**466**
	analysis time (secs)	808	109	162	105	163
lusearch	may-alias pairs	11,788,332	2,251,064	**2,010,780**	1,886,967	**1,771,280**
	may-fail casts	874	504	**450**	412	**358**
	analysis time (secs)	668	94	153	91	155
bloat	may-alias pairs	43,408,294	12,532,334	**11,608,822**	12,155,175	**11,374,583**
	may-fail casts	1944	1401	**1311**	1316	**1226**
	analysis time (secs)	10679	4508	4770	4460	4724

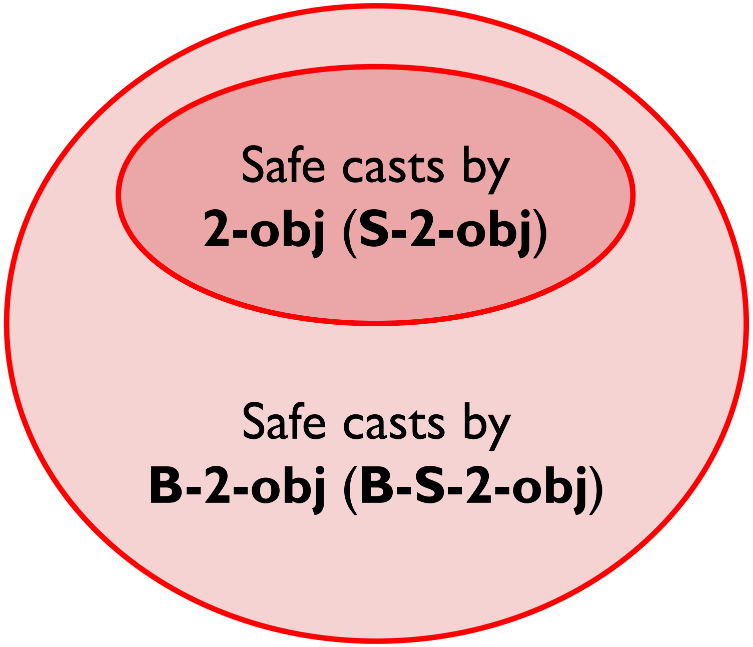
May-Alias



Non-alias pairs by **2-obj (S-2-obj)**

Non-alias pairs by **B-2-obj (B-S-2-obj)**

May-Fail-Cast

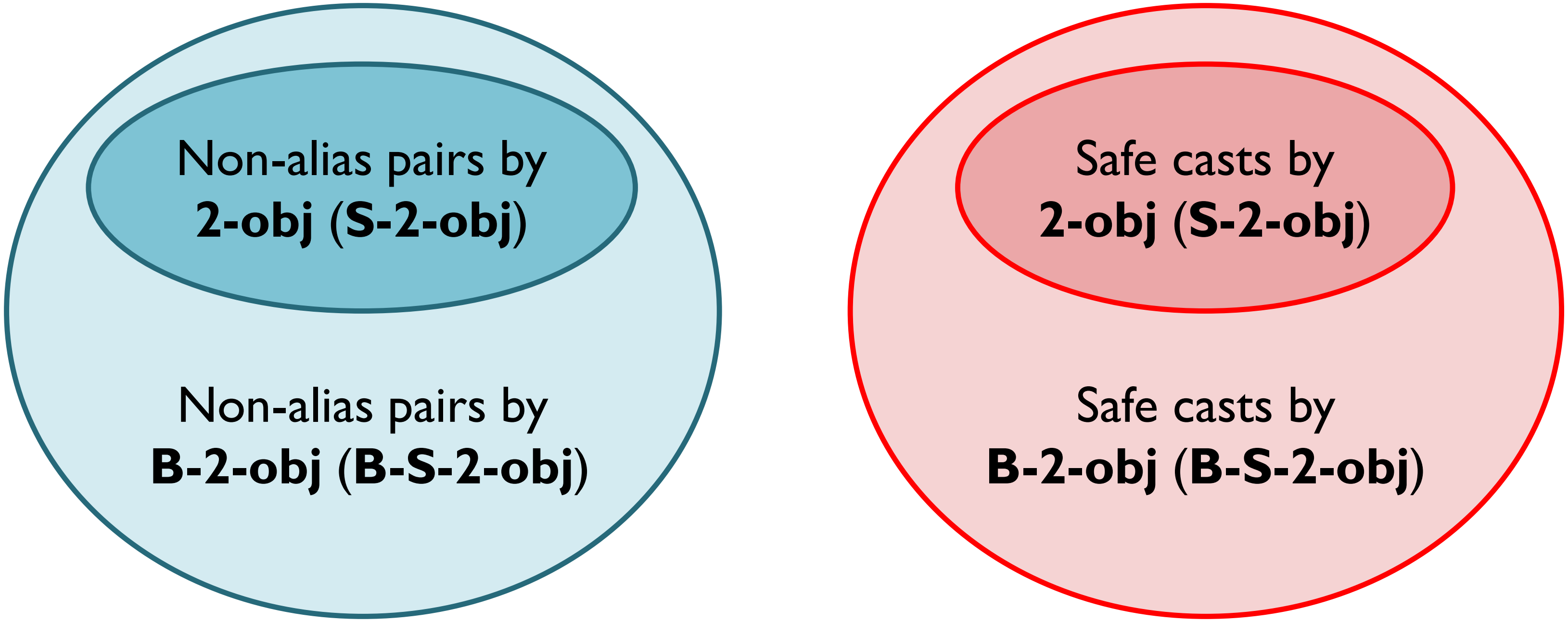


Safe casts by **2-obj (S-2-obj)**

Safe casts by **B-2-obj (B-S-2-obj)**

May-Alias

May-Fail-Cast



Non-alias pairs by **2-obj (S-2-obj)**

Non-alias pairs by **B-2-obj (B-S-2-obj)**

Safe casts by **2-obj (S-2-obj)**

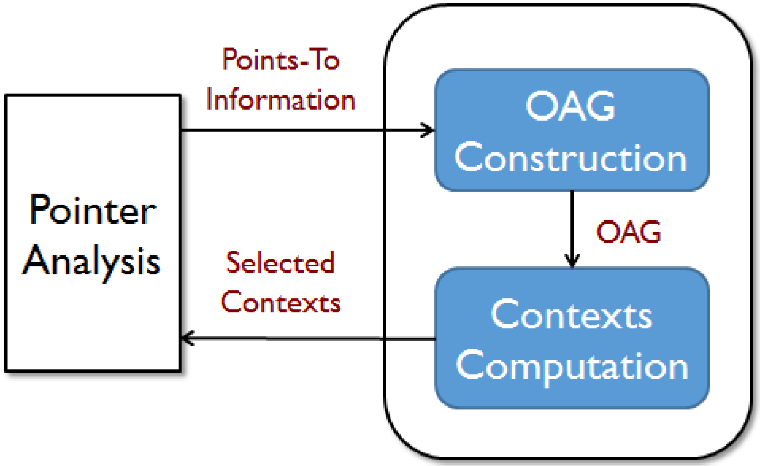
Safe casts by **B-2-obj (B-S-2-obj)**

Verify Theorem 2 practically

Under the same *k*-limiting

Precision of BEAN ≥ Precision of *k*-obj

Performance of BEAN


Pointer Analysis

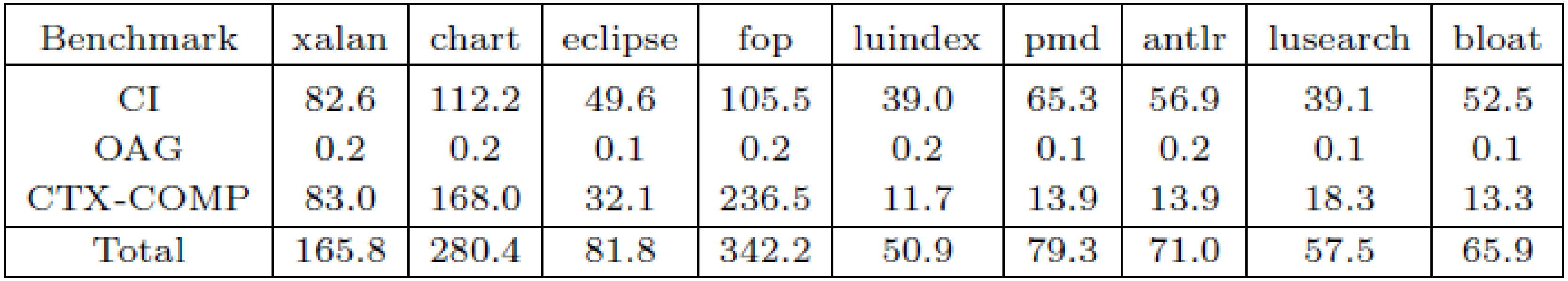
Points-To Information

OAG Construction

OAG

Contexts Computation

Selected Contexts

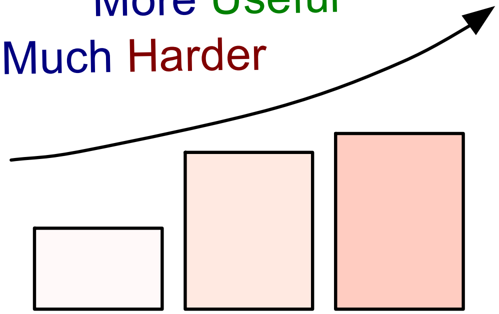


Benchmark	xalan	chart	eclipse	fop	luindex	pmd	antlr	lusearch	bloat
CI	82.6	112.2	49.6	105.5	39.0	65.3	56.9	39.1	52.5
OAG	0.2	0.2	0.1	0.2	0.2	0.1	0.2	0.1	0.1
CTX-COMP	83.0	168.0	32.1	236.5	11.7	13.9	13.9	18.3	13.3
Total	165.8	280.4	81.8	342.2	50.9	79.3	71.0	57.5	65.9

- CI: Context-Insensitive pointer analysis
- OAG: OAG construction
- CTX-COMP: Context Computation

On Average: about 2 minutes

Evaluation Summary



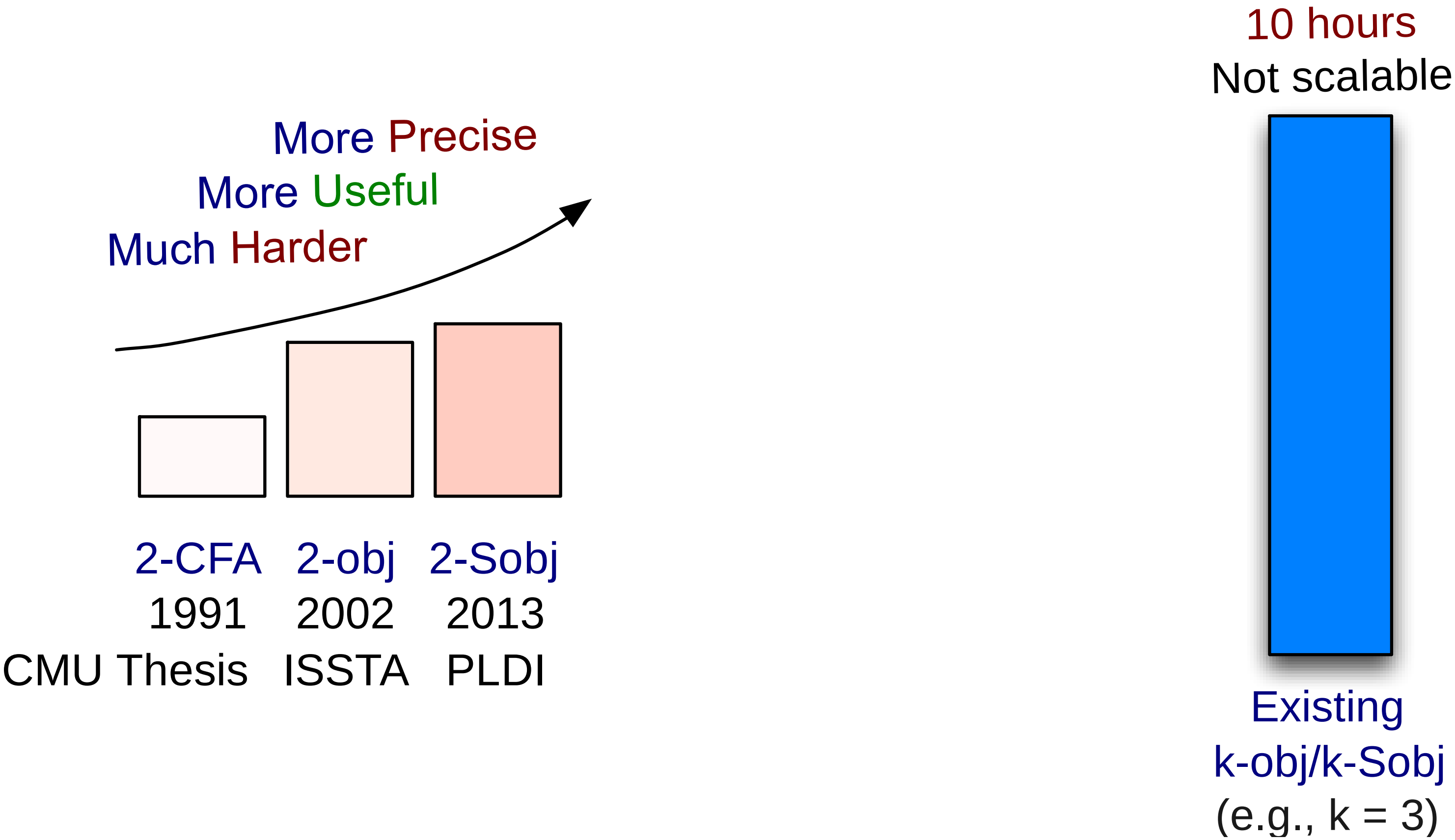
More Precise

More Useful

Much Harder

2-CFA	2-obj	2-Sobj
1991	2002	2013
CMU Thesis	ISSTA	PLDI

Evaluation Summary



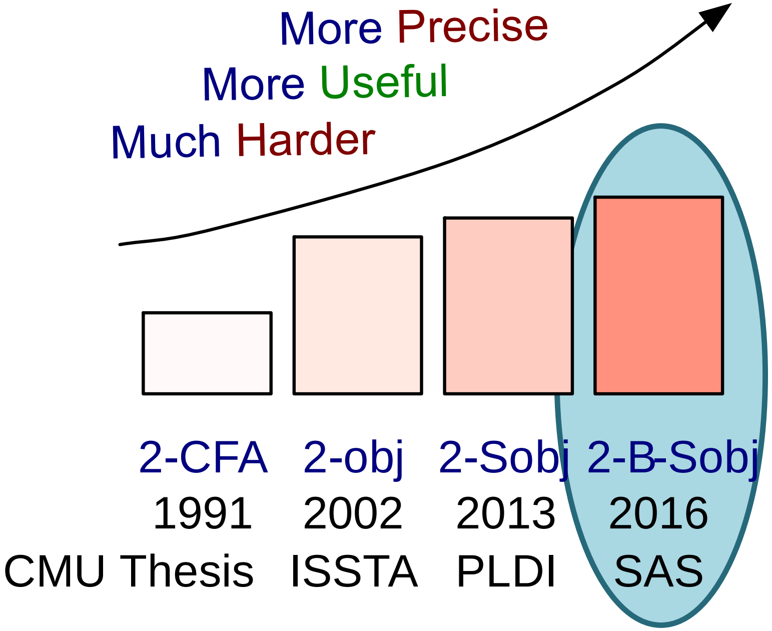
More Precise
More Useful
Much Harder

2-CFA	2-obj	2-Sobj
1991	2002	2013
CMU Thesis	ISSTA	PLDI

10 hours
Not scalable

Existing
k-obj/k-Sobj
(e.g., k = 3)

Evaluation Summary



More Precise

More Useful

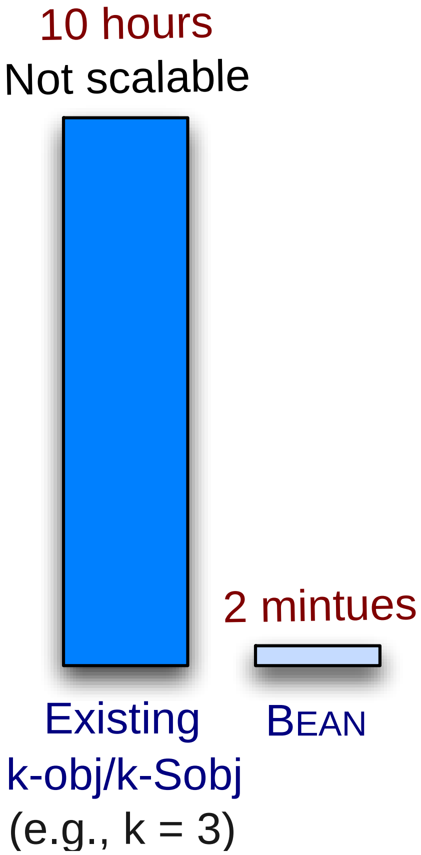
Much Harder

2-CFA 1991 CMU Thesis

2-obj 2002 ISSTA

2-Sobj 2013 PLDI

2-B-Sobj 2016 SAS



10 hours

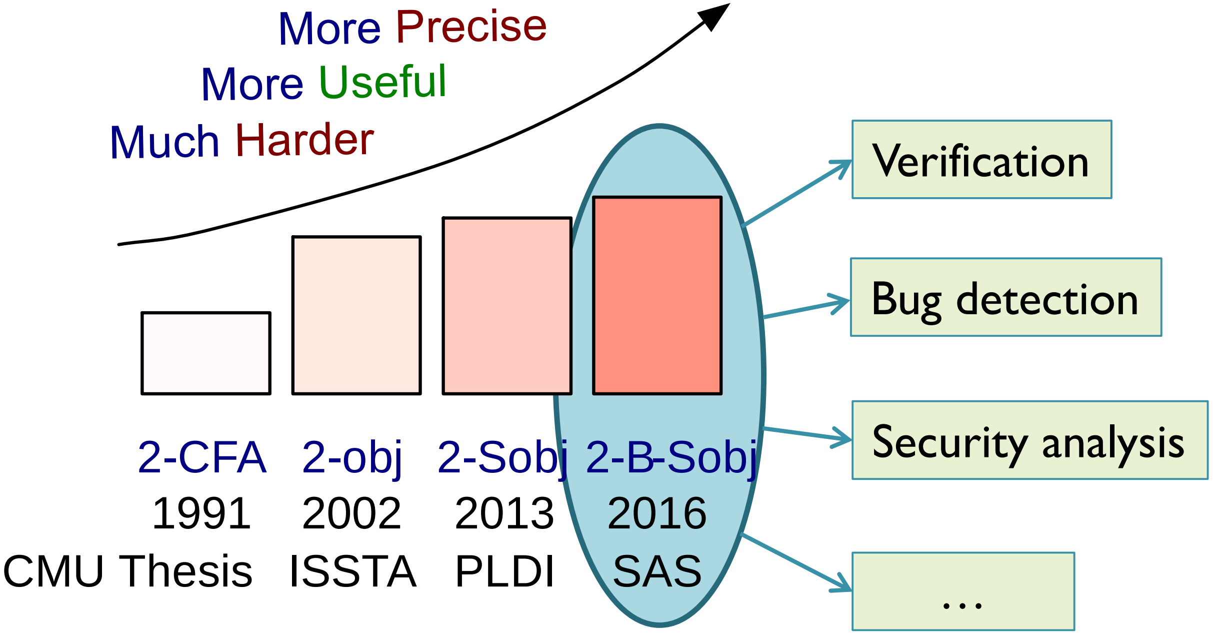
Not scalable

2 mintues

Existing k-obj/k-Sobj (e.g., k = 3)

Bean

Evaluation Summary



More Precise

More Useful

Much Harder

2-CFA
1991
CMU Thesis

2-obj
2002
ISSTA

2-Sobj
2013
PLDI

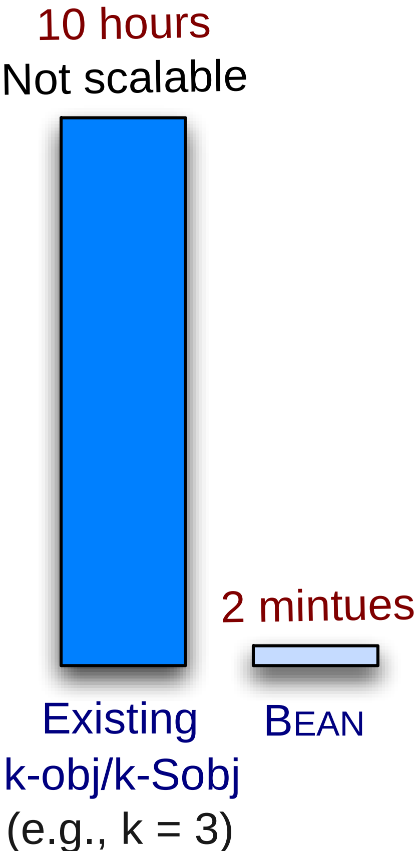
2-B-Sobj
2016
SAS

Verification

Bug detection

Security analysis

…



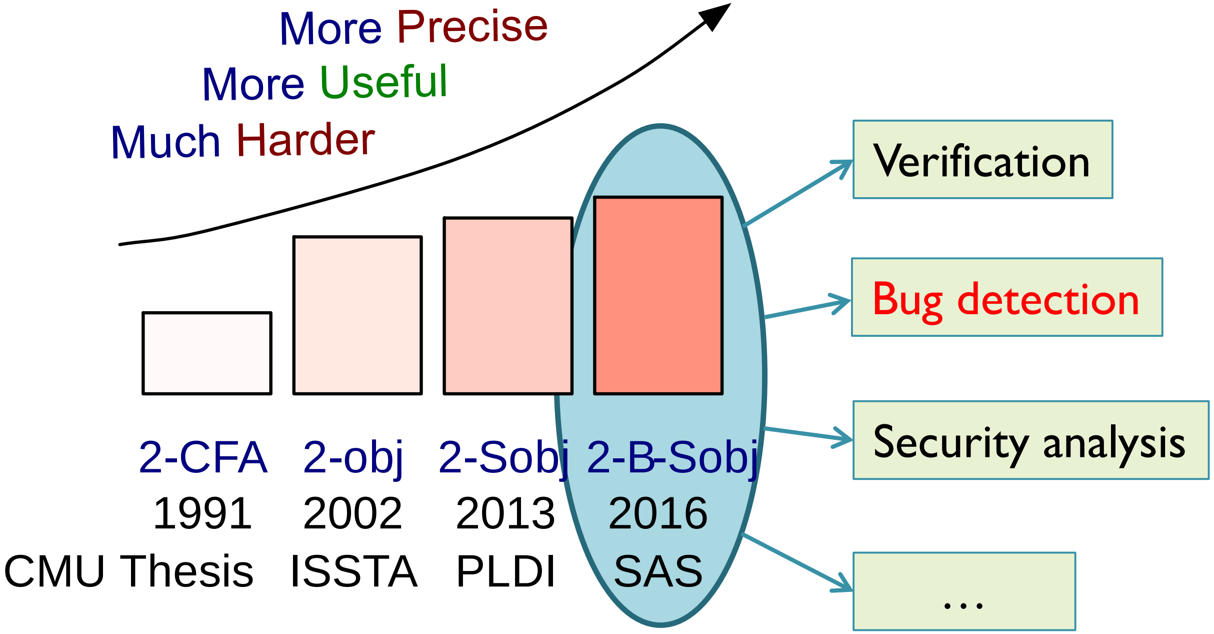
10 hours
Not scalable

2 mintues

Existing
k-obj/k-Sobj
(e.g., k = 3)

BEAN

Evaluation Summary



More Precise

More Useful

Much Harder

2-CFA 1991 CMU Thesis

2-obj 2002 ISSTA

2-Sobj 2013 PLDI

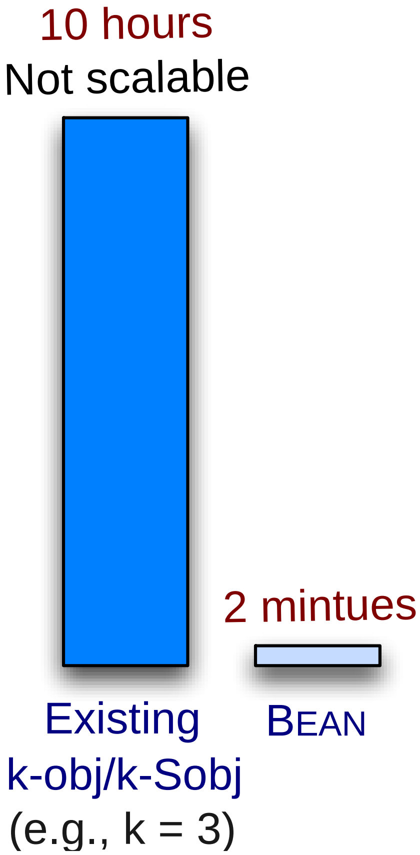
2-B-Sobj 2016 SAS

Verification

Bug detection

Security analysis

…



10 hours

Not scalable

2 mintues

Existing k-obj/k-Sobj (e.g., k = 3)

BEAN

"Using static data race detection will likely show even more dramatic improvement in precision **using your approach**."

Conclusion

Making *k*-Object-Sensitive Pointer Analysis
More Precise with Still *k*-Limiting

- Improve the precision of object-sensitivity by avoiding redundant context elements
  - *k*-limiting, *k*+ precision
  - Scalable

- Easily applied to other context-sensitive analyses
  - *k*-CFA
  - Type-sensitive analysis

Thank you!