컨텍스트 터널링

구분할 호출환경 배달하기: 고정관념에 도전하기

전민석



Aug.21.2025 @ SIGPL 여름학교

문제

• 아래 문장은 어떤 동화를 한 문장으로 요약한 것이다, 몇점짜리 요약일까?

문제

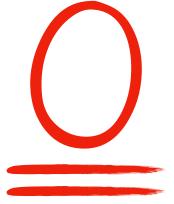
• 아래 문장은 어떤 동화를 한 문장으로 요약한 것이다, 몇점짜리 요약일까?

"The End"

문제

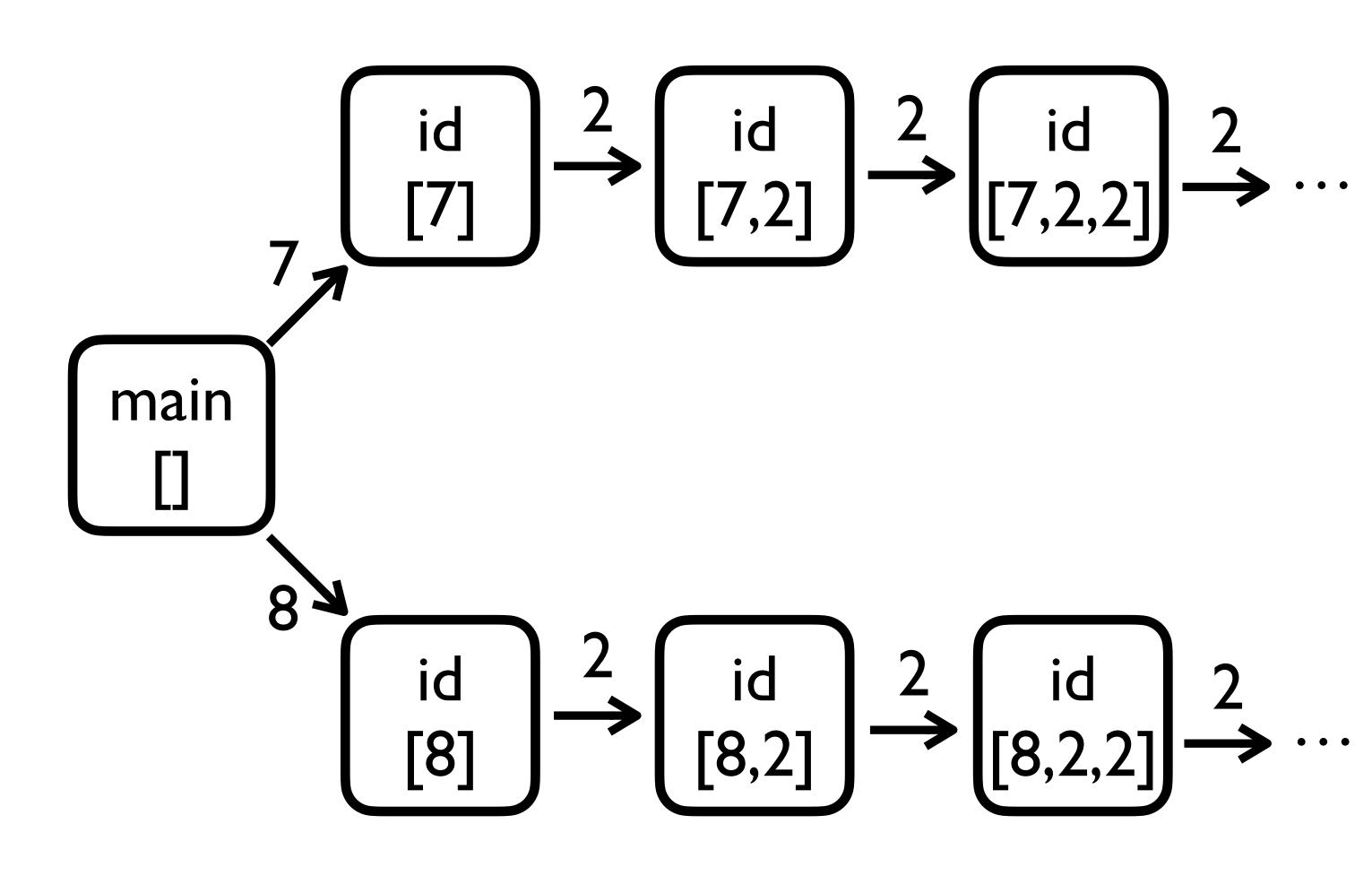
• 아래 문장은 어떤 동화를 한 문장으로 요약한 것이다, 몇점짜리 요약일까?

"The End"

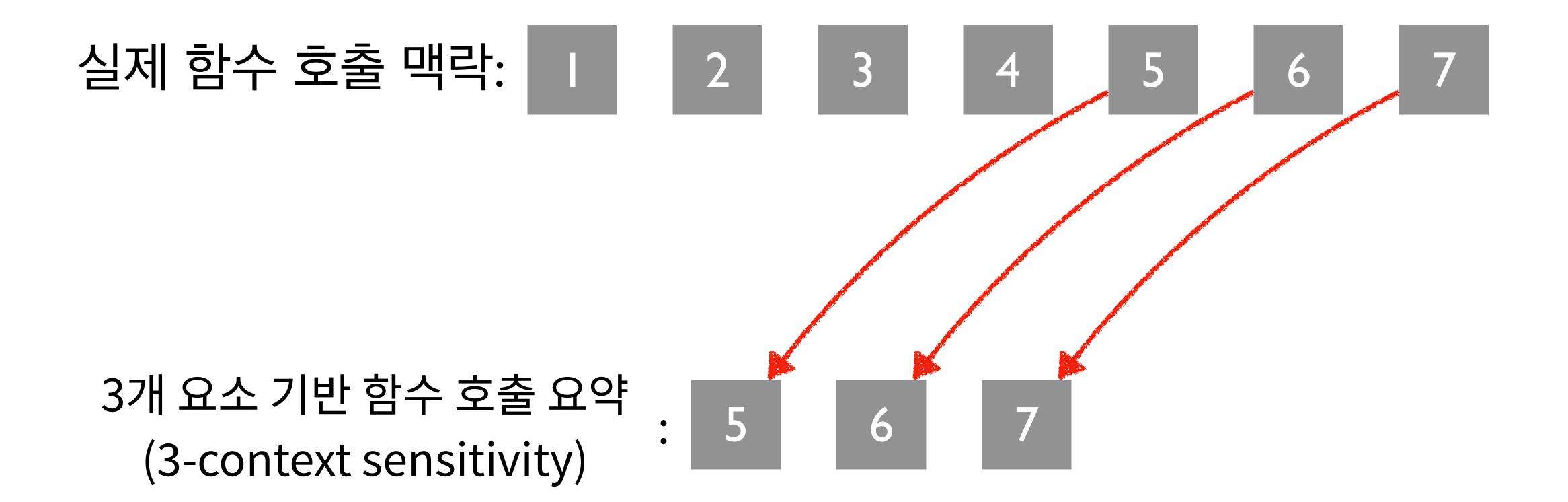


함수호출 요약의 필요성

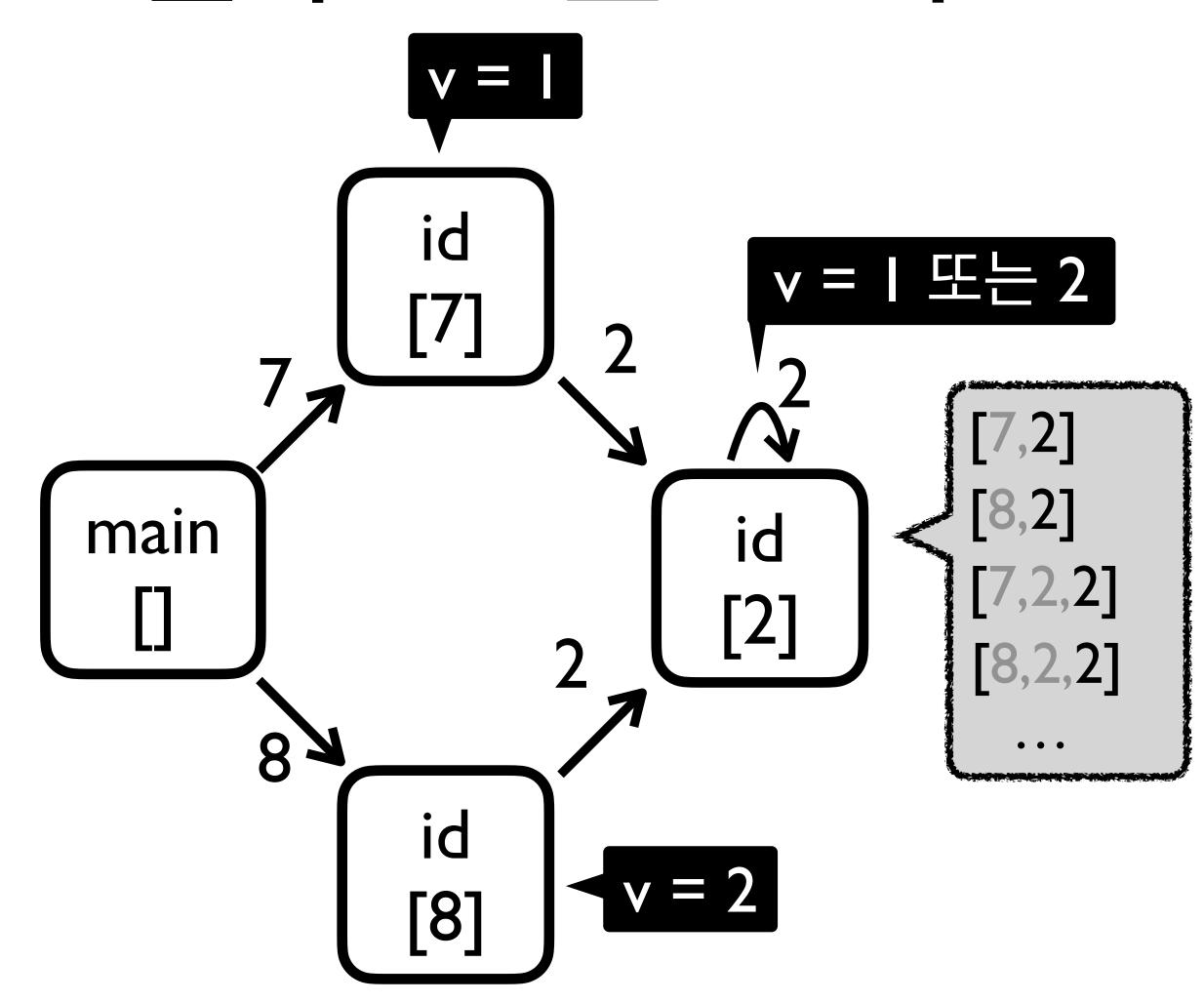
```
0: id(v, i){
   if (i > 0){
    return id(v, i-1);}
    return v;}
4:
5: main(){
   i = input();
   vI = id(I, i);//A
   v2 = id(2, i);//B
8:
    assert (vI != v2);//query
9:
   예제 프로그램
```



함수 호출 그래프

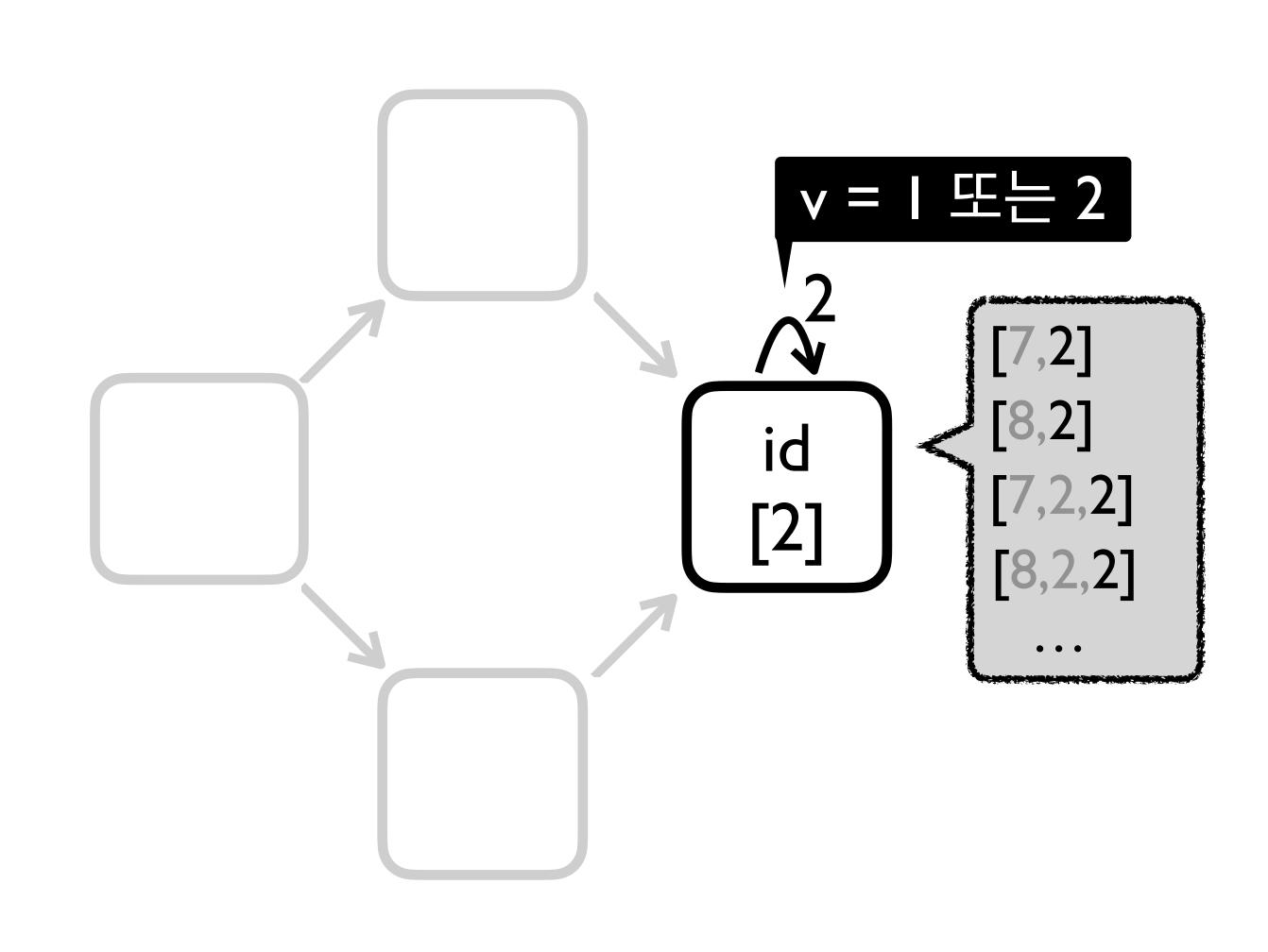


```
0: id(v, i){
  if (i > 0){
    return id(v, i-1);}
    return v;}
4:
5: main(){
   i = input();
   vI = id(I, i);//A
   v2 = id(2, i);//B
8:
    assert (vl != v2);//query
9:
   예제 프로그램
```

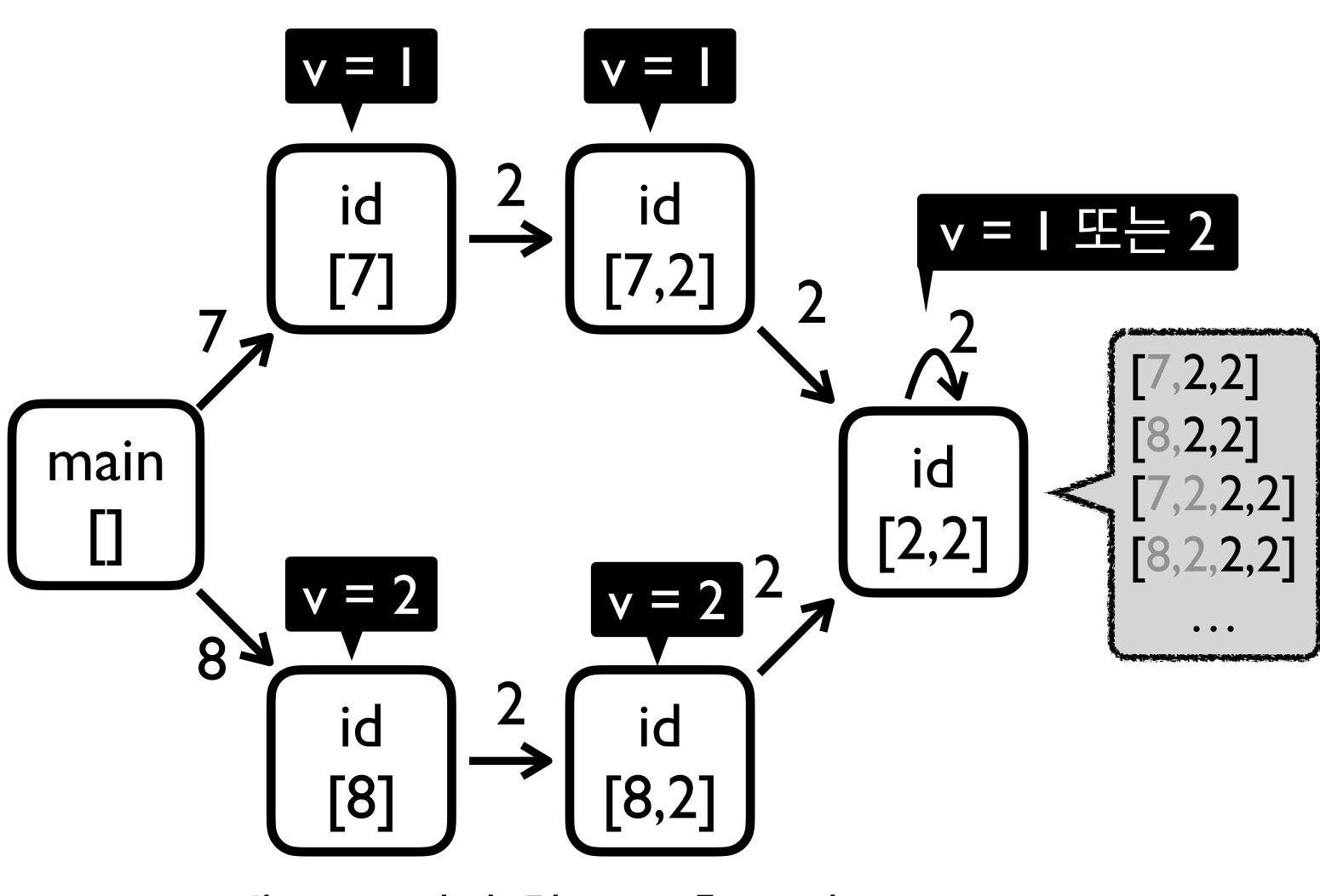


I개 요소 기반 함수 호출 요약

```
0: id(v, i){
1: if (i > 0)
    return id(v, i-1);}
    return_v;} v = [ 年2
    v2 = id(2, i);
8:
    assert (vI != v2);//query
9:
10: }
       1 또는 2
                1 또는 2
   예제 프로그램
```

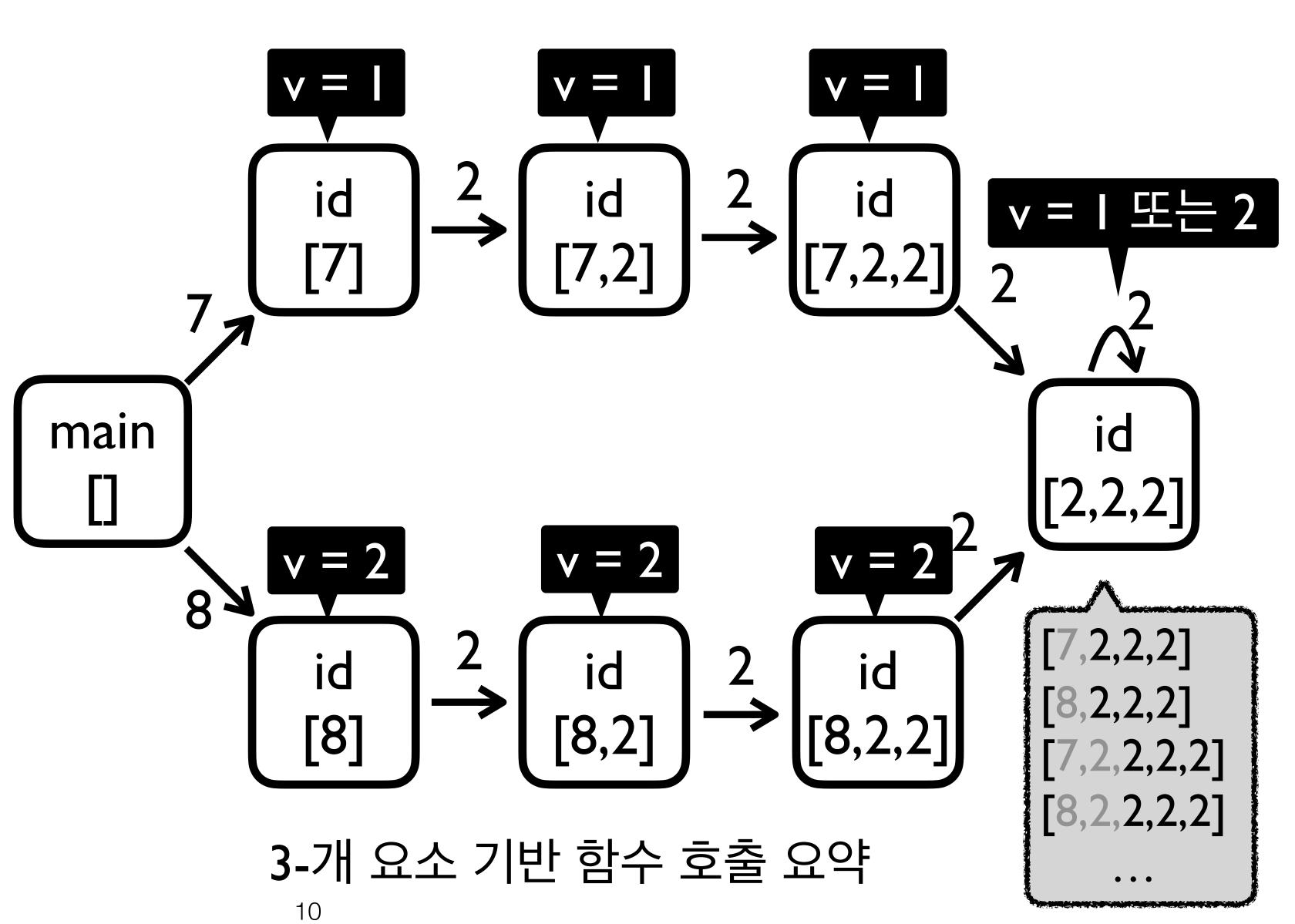


```
0: id(v, i){
   if (i > 0){
     return id(v, i-1);}
    return v;}
4:
5: main(){
   i = input();
   vI = id(I, i);//A
    v2 = id(2, i);//B
8:
    assert (vI != v2);//query
9:
    예제 프로그램
```



2-개 요소 기반 함수 호출 요약

```
0: id(v, i){
    if (i > 0){
     return id(v, i-1);}
     return v;}
4:
5: main(){
    i = input();
    vI = id(I, i);//A
    v2 = id(2, i);//B
8:
     assert (vI != v2);//query
9:
    예제 프로그램
```



고정 관념: 마지막 k개 기반 요약

• 마지막 k개 기반 요약 방식이 널리 강의 되는 중

Call-Site Sensitivity



Partial Context-sensitivity



- The best-known flavor of context sensitivity, which uses callsites as contexts.
- A method is analyzed under the context that is a sequence of the last *k* call-sites

Partial Context-sensitivity

- The most common way: keep only the top-most k continuations (so-called k-CFA)
 - k = 0: ignore all contexts, i.e., context-insensitive
 - $k = \infty$: keep all contexts, i.e., fully context-sensitive



- The most common way: keep only the top-most k callstrings (called k-CFA)
- Approach: set an upper bound for length of contexts, denoted by k



- For call-site sensitivity, each context consists of the last k
 call sites of the call chains
- In practice, *k* is a small number (usually ≤3)
- Method contexts and heap contexts may use different k
 - e.g., k=2 for method contexts, k=1 for heap contexts

(검색 키워드: Context Sensitivity Static Analysis Lecture)

고정 관념: 마지막 k개 기반 요약

• 리뷰 코멘트

"A key part of the appeal of last k-based context abstraction is its simplicity and universal applicability."

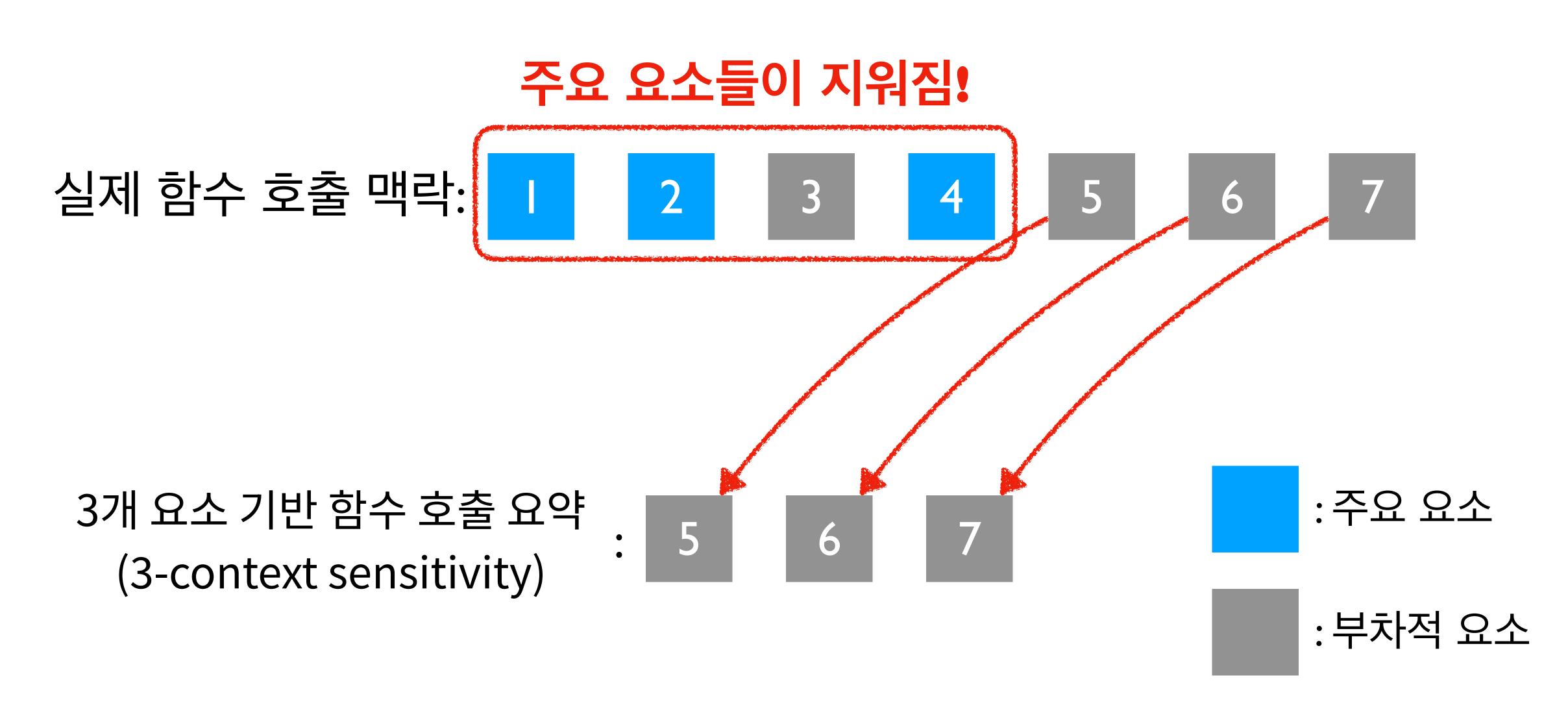
- A reviewer [expert]

- k = 0: ignore all contexts, i.e., context-insensitive
- k = ∞: keep all contexts, i.e., fully context-sensitive



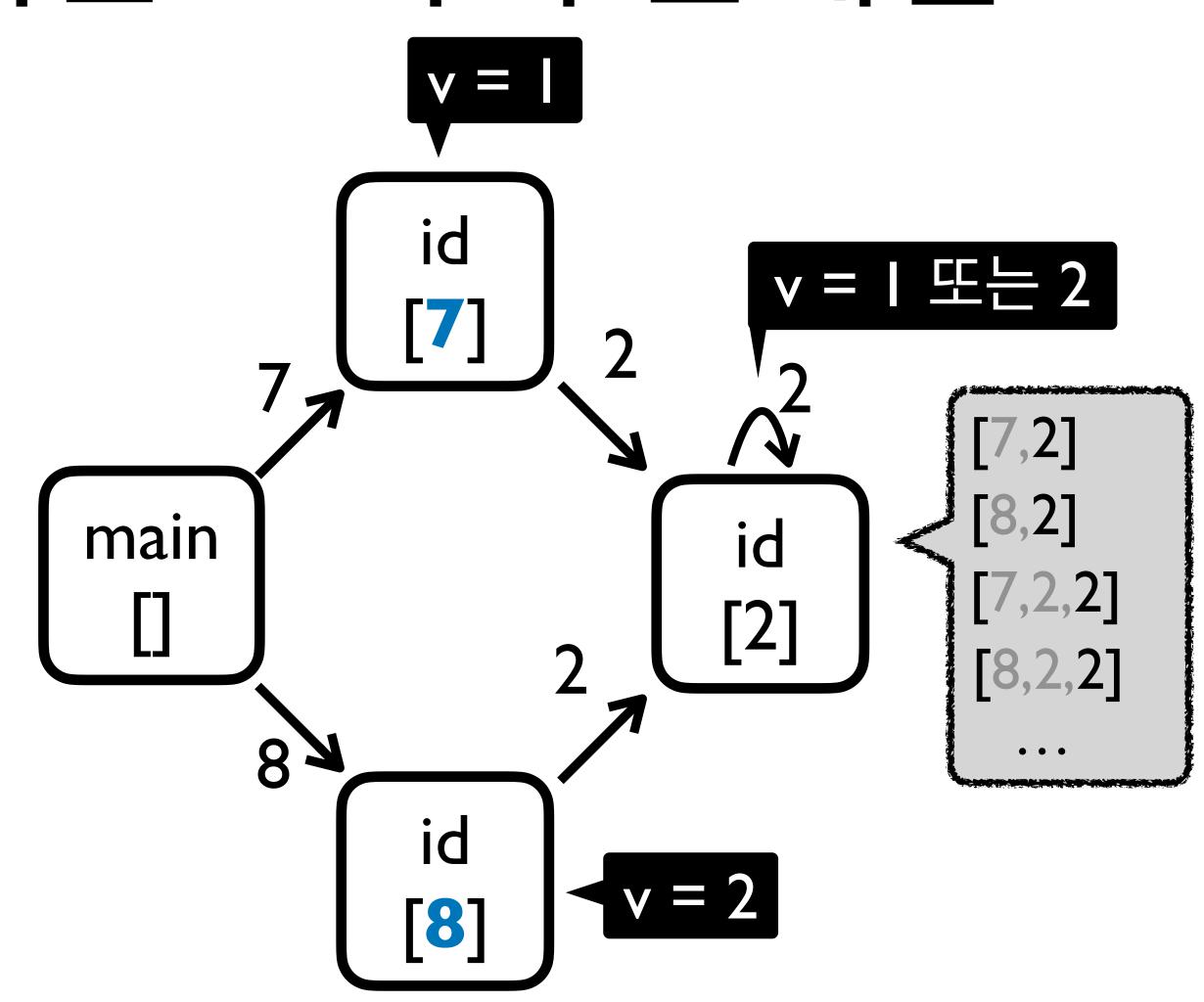
- In practice, *k* is a small number (usually ≤3)
- Method contexts and heap contexts may use different k
 - e.g., k=2 for method contexts, k=1 for heap contexts

마지막 K개 기반 요약의 문제점



마지막 K개 기반 요약의 문제점

```
0: id(v, i){
  if (i > 0){
    return id(v, i-1);}
    return v;}
4:
5: main(){
   i = input();
   vI = id(I, i);
   v2 = id(2, i);
8:
    assert (vI != v2);//query
9:
   예제 프로그램
```



I개 요소 기반 함수 호출 요약

발견하게된계기

Exercise

```
class S {
 Object id(Object a) { return a; }
 Object id2(Object a) { return id(); }
class C extends S {
 void fun1() {
   Object a1 = new A1();
   Object b1 = id2(a1);
  }}
class D extends S {
 void fun2() {
   Object a2 = new A2();
   Object b2 = id2(a2);
 }}
```

● What is the result of 1-call-site-sensitive analysis? < 부정확함

발견하게된계기

Exercise

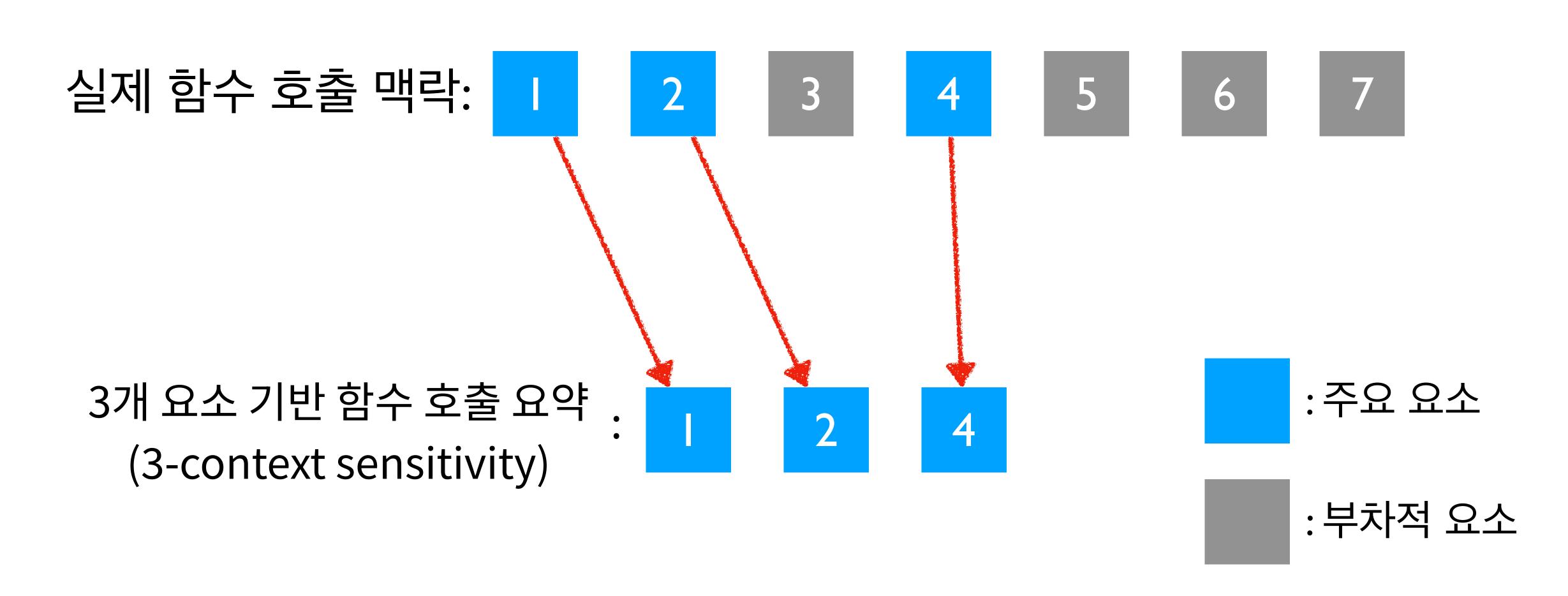
```
class S {
 Object id(Object a) { return a; }
 Object id2(Object a) { return id(); }
class C extends S {
 void fun1() {
   Object a1 = new A1();
   Object b1 = id2(a1);
  }}
class D extends S {
 void fun2() {
   Object a2 = new A2();
   Object b2 = id2(a2);
  }}
```

질문:

I-call-site sensitivity로 정확하게 분석할 순 없나?

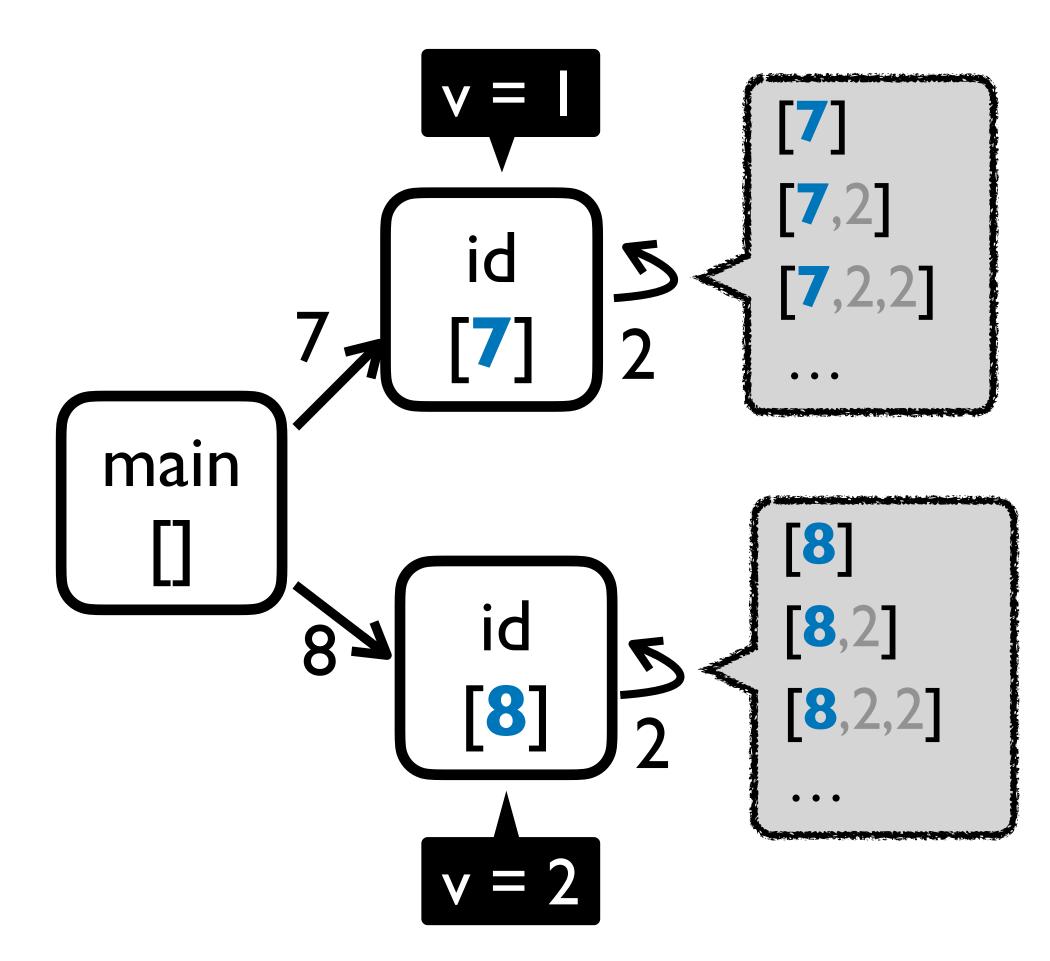
● What is the result of 1-call-site-sensitive analysis? ✔ 부정확함

호출 환경 배달하기: 주요 K개 기반 요약

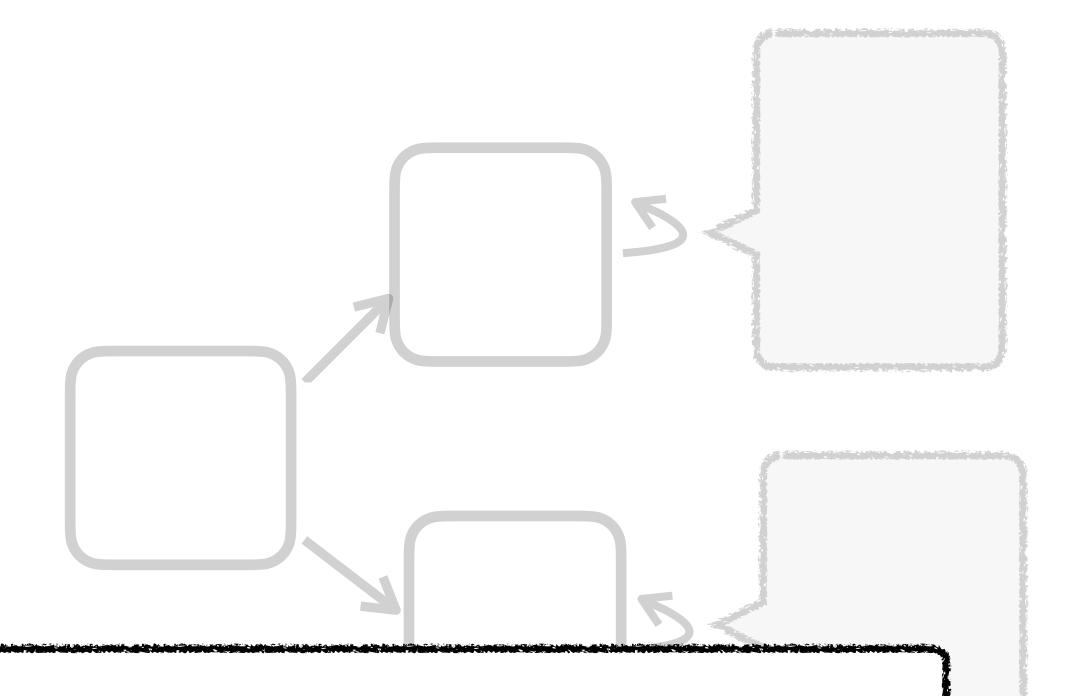


호출 환경 배달하기: 주요 K개 기반 요약

```
0: id(v, i){
  if (i > 0){
    return id(v, i-1);}
    return v;}
4:
5: main(){
   i = input();
   vI = id(I, i);
   v2 = id(2, i);
8:
9:
    assert (vI != v2);//query
   예제 프로그램
```



I개 주요 요소 기반 함수 호출 요약 (주요 요소 = {7,8})



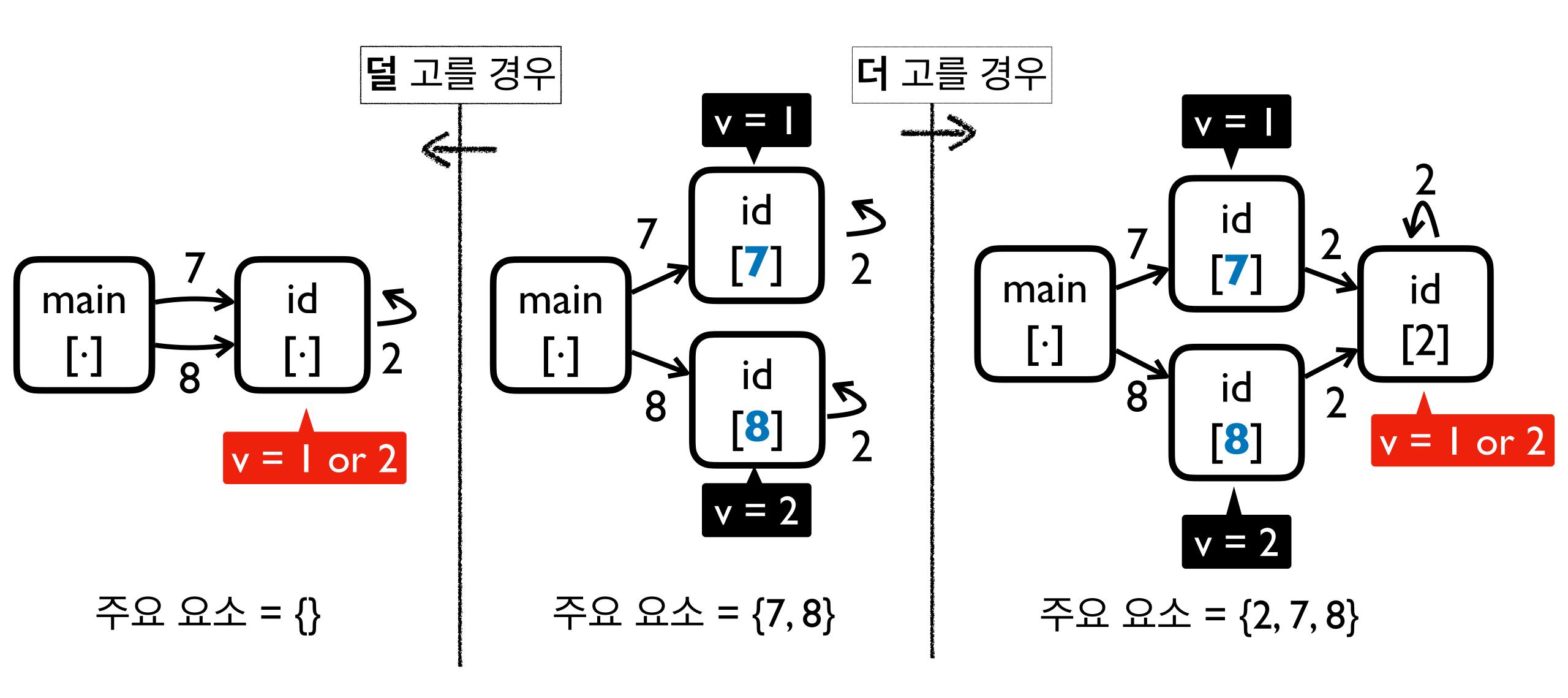
Challenge:

분석 전에 주요 요소를 **정확**하게 알아내야 함

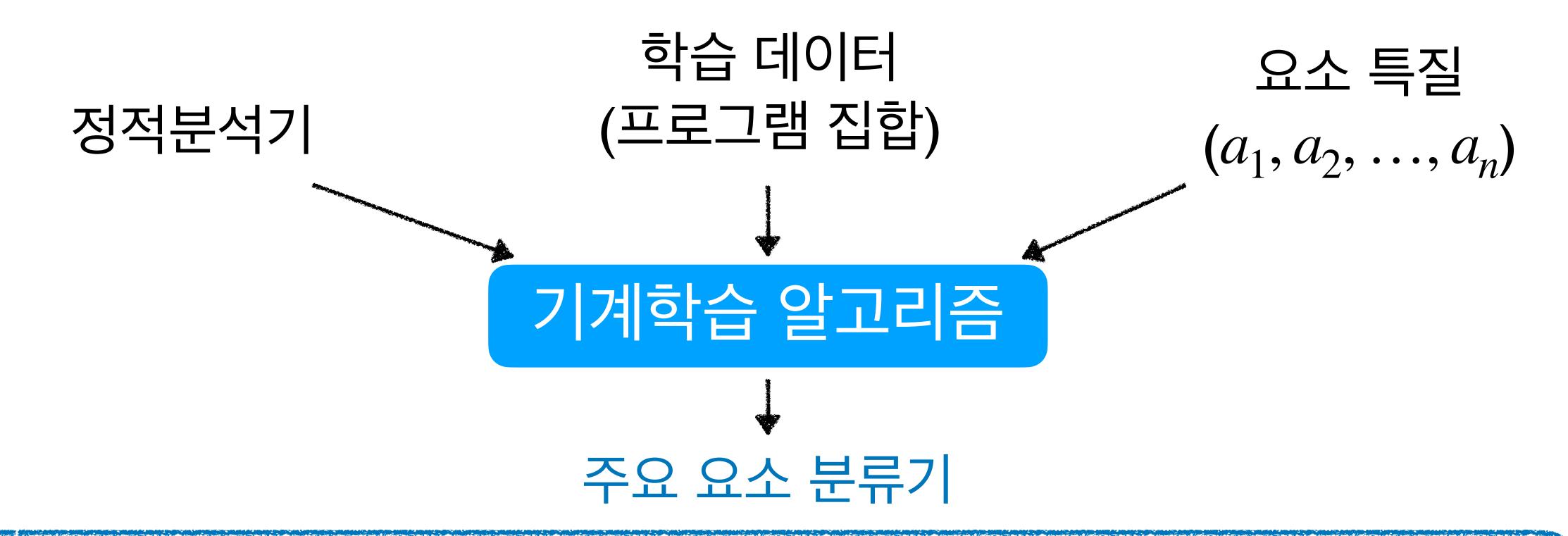
1개주요 맥락기산 함수 호출 요약

(주요 요소 = {7,8})

• 정확하게 분류해야만 분석의 정확도를 향상시킬 수 있음



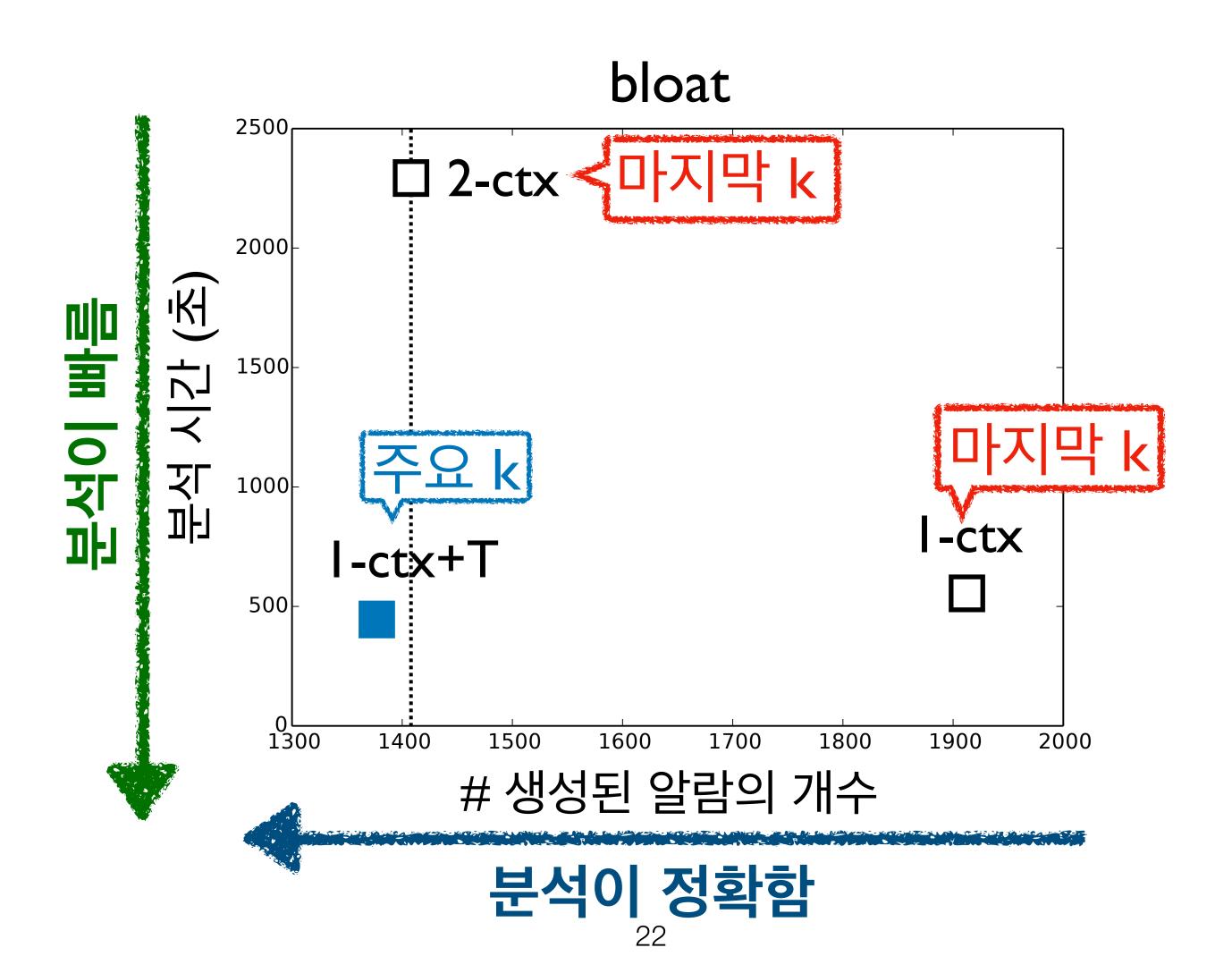
접근 방법: 데이터 기반 컨텍스트 터널링



$$f = (a_1 \land \neg a_2 \land \neg a_3 \land \ldots) \lor (a_1 \land \neg a_3 \land a_7 \land \ldots) \lor \ldots$$

마지막 k 기반 요약 vs 주요 k기반 요약

• 주요 1개 기반 요약이 마지막 2개 기반 요약보다 더 높은 정확도를 보임



2-ctx **4** 2000 주요 k 1000 -ctx+T1500

• 2-ctx 는 연구에서 정확도 상한선으로 사용되어 왔음

" ...it covers more than two-thirds of the precision advantage of 20bjH" -Smaragdakis et al. [PLDI' 14]

"... 98.8% of the precision of 2obj can be preserved..."

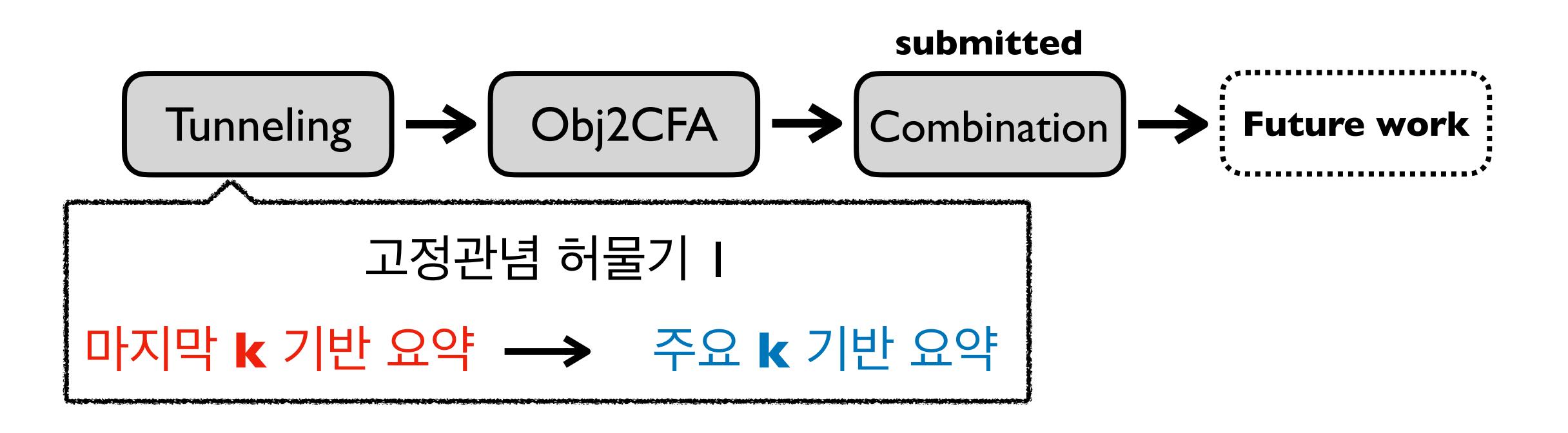
-Li et al. [OOPSLA' 18]

"Scaler still attains most of the precision gains of 20bj ..."
-Li et al. [FSE' 18]

• • •

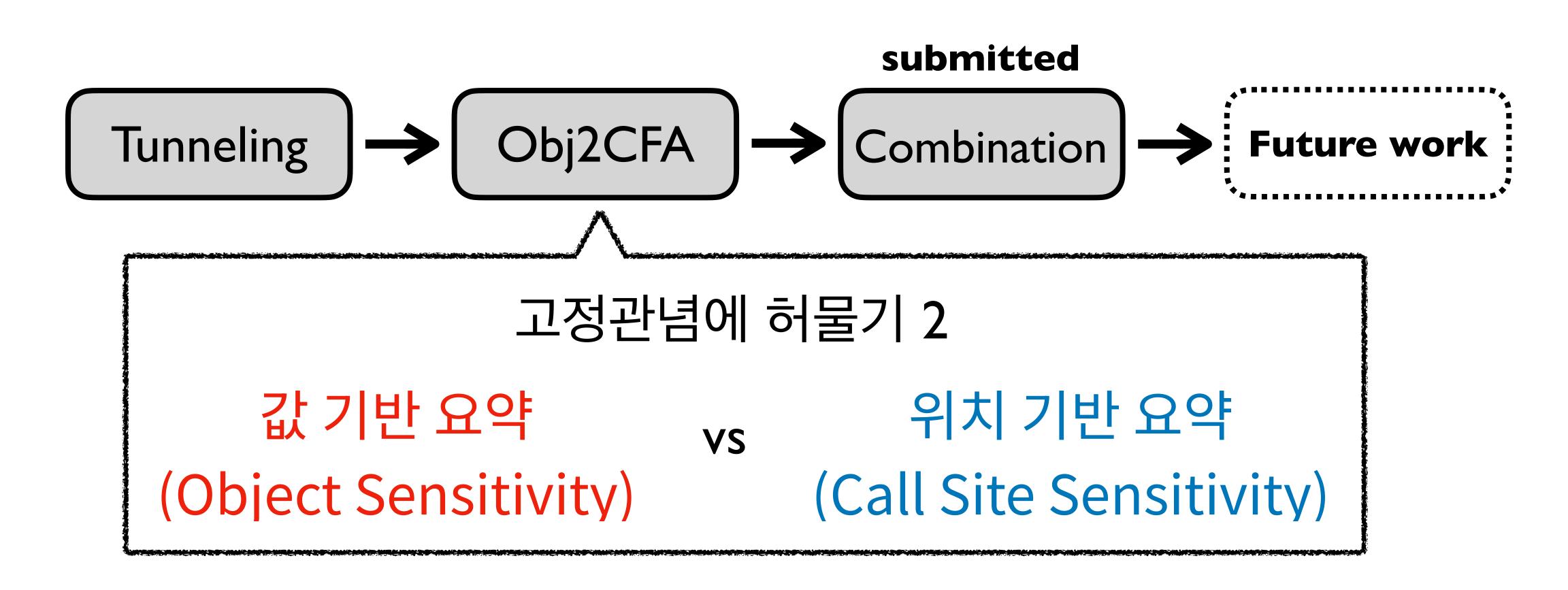
고정관념에 도전하기

• 목표: 주요 k기반 요약 방식을 표준으로 만들기



고정관념에 도전하기

• 목표: 주요 k기반 요약 방식을 표준으로 만들기



위치 기반요약 vs 값기반요약

위치 기반 요약 (1981) 값 기반 요약 (2002) "무엇"을 기반으로 요약 • "어디"를 기반으로 요약 goo goo **OI** or **O2** 0: foo(){ 0: foo(: goo(); foo foo goo goo 2: goo(); p.goo(); Call graph Call graph

2022

1981

위치기반요약vs값기반요약

Parameterized Object Sensitivity for Points-to Analysis for Java

NA MILANOVA

ATANAS ROUNTEV Ohio State University

BARBARA G. RYDER Rutgers University

he goal of *points-to analysis* for Java is to determine the set of objects pointed to by a referer the goal of points-to analysis for Java is to determine the set of objects pointed to by a reference object arriable or a reference object field. We present object sensitivity, a new form of context sensitivity for flow-insensitive points-to analysis for Java. The key idea of our approach is to analyze a method separately for each of the object anness that represent run-time objects on which this method may be invoked. To ensure flexibility and practicality, we propose a parameterization framework that allows analysis designers to control the tradeoffs between cost and precision in the object-sensitive analysis.

analysis.

Side-effect analysis determines the memory locations that may be modified by the execution of program statement. Def-use analysis identifies pairs of statements that set the value of a memory ocation and subsequently use that value. The information computed by such analyses has a wid variety of uses in compilers and software tools. This work proposes new versions of these analyse that are based on object-sensitive points-to analysis.

We have implemented the unitaryitation of our parameterized object-sensitive points-to analysis.

is. On a set of 23 Java programs, our experiments show that these analyses have comparable cost o a context-insensitive points-to analysis for Java which is based on Andersen's analysis for C. Our sults also show that object sensitivity significantly improves the precision of side-effect analys nd call graph construction, compared to (1) context-insensitive analysis, and (2) context-sensiti ints-to analysis that models context using the invoking call site. These experiments dem hat object-sensitive analyses can achieve substantial precision improvement, while at the sam me remaining efficient and practical.

preliminary version of this article appeared in Proceedings of the International Symposium

his research was supported in part by National Science Foundation (NSF) grant CCR-990098 uthor's addresses: A. Milanova, Department of Computer Science, Rensselaer Polytechnic Inst ute, 110 8th Street, Troy, NY 12180; email: milanova@cs.rpi.edu; A. Rountey, Department of Co uter Science and Engineering, Ohio State University, 2015 Neil Avenue, Columbus, OH 43210 mail: rountev@cse.ohio-state.edu; B. G. Ryder, Department of Computer Science, Rutgers Univer ty, 100 Frelinghuysen Road, Piscataway, NJ 08854; email: ryder@cs.rutgers.edu sion to make digital or hard copies of part or all of this work for personal or classroom use

Context-sensitive points-to analysis: is it worth it?*

olhotak@uwaterloo.ca hendren@sable.mcgill.ca

School of Computer Science, University of Waterloo, Waterloo, ON, Canada ² School of Computer Science, McGill University, Montreal, QC, Canada

Abstract. We present the results of an empirical study evaluating the precision of subset-based points-to analysis with several variations of context sensitivity on Java benchmarks of significant size. We compare the use of call site strings as the context abstraction, object sensitivity, and the BDD-based context-sensitive algorithm proposed by Zhu and Calman, and by Whaley and Lam, Our study includes analyses that context-sensitively specialize only pointer variables, as well as ones that also specialize the heap abstraction. We measure both characteristics of the points-to sets themselves, as well as effects on the precision of client analyses. To guide development of efficient analysis implement sets that arise with each context sensitivity variation. To evaluate precision, we measure the size of the call graph in terms of methods and edges, the number of The results of our study indicate that object-sensitive analysis implementations are likely to scale better and more predictably than the other approaches; that objectsensitive analyses are more precise than comparable variations of the other approaches; that specializing the heap abstraction improves precision more than extending the length of context strings; and that the profusion of cycles in Java call graphs severely reduces precision of analyses that forsake context sensitivity in cyclic regions.

Does context sensitivity significantly improve precision of interprocedural analysis o object-oriented programs? It is often suggested that it could, but lack of scalable implementations has hindered thorough empirical verification of this intuition.

Of the many context sensitive points-to analyses that have been proposed (e.g. [1, 4, 8,11,17-19,25,28-31]), which improve precision the most? Which are most effective for specific client analyses, and for specific code patterns? For which variations are we likely to find scalable implementations? Before devoting resources to finding efficient imple mentations of specific analyses, we should have empirical answers to these questions

This study aims to provide these answers. Recent advances in the use of Binary De cision Diagrams (BDDs) in program analysis [3, 12, 29, 31] have made context sensitive analysis efficient enough to perform an empirical study on benchmarks of significant size

Evaluating the Benefits of Context-Sensitive Points-to Analysis Using a BDD-Based Implementation

ONDŘEJ LHOTÁK

University of Waterloo

LAURIE HENDREN McGill University

We present Paddle, a framework of BDD-based context-sensitive points-to and call graph analysis for Java, as well as client analyses that use their results. Paddle supports several variations of context-sensitive analyses, including call site strings and object sensitivity, and context-sensitively. context-sensitive analyses, including can site strings and object sensitivity, and context-sensitive specializes both pointer variables and the heap abstraction. We empirically evaluate the prec-sion of these context-sensitive analyses on significant Java programs. We find that that object sensitive analyses are more precise than comparable variations of the other approaches, and the

General Terms: Languages, Design, Experimentation, Measurement

Additional Key Words and Phrases: Interprocedural program analysis, context sensitivity, binar decision diagrams, Java, points-to analysis, call graph construction, cast safety analysis

Lhoták, O. and Hendren, L. 2008. Evaluating the benefits of context-sensitive points-to analy $using\ a\ BDD-based\ implementation.\ ACM\ Trans.\ Softw.\ Engin.\ Method.\ 18,\ 1,\ Article\ 3\ (Septemb2008),\ 53\ pages.\ DOI=10.1145/1391984.1391987\ \ http://doi.acm.org/10.1145/1391984.1391987$

This is a revised and extended version of an article which appeared in Proceedings of the 15s

Springer, 47-64. Springer, 47-64. Authors' addresses: O. Lhoták, D. R. Cheriton School of Computer Science, University of Waterlo 200 University Avenue West, Waterloo, ON, NZL 3GI, Canada; L. Hendren, School of Cor uter Science, McGill University, 3480 University Street, Room 318, Montreal, QC, H3A 2A

Strictly Declarative Specification of Sophisticated Points-to Analyses

Department of Computer Science

Amherst, MA 01003, USA martin.bravenboer@acm.org yannis@cs.umass.edu

We present the Doop framework for points-to analysis of Java programs. Door builds on the idea of specifying pointer analysis algorithms declaratively, using Datalog: a logicbased language for defining (recursive) relations. We carry the declarative approach further than past work by describing the full end-to-end analysis in Datalog and optimizing aggressively using a novel technique specifically targeting

aggressively using a novel technique specifically targeting highly recursive Datalog programs.

As a result, Door achieves several benefits, including full order-of-magnitude improvements in runtime. We compare Door with Lhoták and Hendren's Paddle, which defines the state of the art for context-sensitive analyses. For the exact same logical points-to definitions (and, consequently, identical precision) Door is more than 15x faster than Paddle of the sensitive analysis of the DoCano benchmarks. a 1-call-site sensitive analysis of the DaCapo benchmarks, compared to the prior state of the art, Doop often achieves speedups of an order-of-magnitude for several important analyses. Additionally, Door scales to very precise analyses that are impossible with PADLE and Whaley et al.'s bddbddb, directly addressing open problems in past literature. Finally, our implementation is modular and can be easily configured to analyses with a wide range of characteristics, largely due to its declarativeness.

Compared to the prior state of the art, Doop often achieves speedups of an order-of-magnitude for several important analyses. The main elements of our approach are the use of the Datalog under the prior state of the art, Doop often achieves speedups of an order-of-magnitude for several important analyses. The main elements of our approach are the use of the Datalog under the prior state of the art, Doop often achieves speedups of an order-of-magnitude for several important analyses. The main elements of our approach are the use of the Datalog under the prior state of the art, Doop often achieves speedups of an order-of-magnitude for several important analyses. The main elements of our approach are the use of the Datalog under the prior state of the art, Doop often achieves speedups of an order-of-magnitude for several important analyses. The main elements of our approach are the use of the Datalog under the prior state of the prior state of

Languages—Program Analysis; D.1.6 [Programming Techniques]: Logic Programming General Terms Algorithms, Languages, Performance

Points-to (or pointer) analysis intends to answer the question "what objects can a program variable point to?" This question forms the basis for practically all higher-level program

has been devoted to efficient and precise pointer analysis techniques. Context-sensitive analyses are the most commo class of precise points-to analyses. Context sensitive analys approaches qualify the analysis facts with a context abstration, which captures a static notion of the dynamic content

points-to analysis framework that makes feasible the mos precise context-sensitive analyses reported in the literature Door implements a range of algorithms, including content insensitive, call-site sensitive, and object-sensitive analyse insensitive, call-site sensitive, and object-sensitive analyses, all specified modularly as variations on a common code base. Compared to the prior state of the art, Door often achieves speedups of an order-of-magnitude for several important analyses.

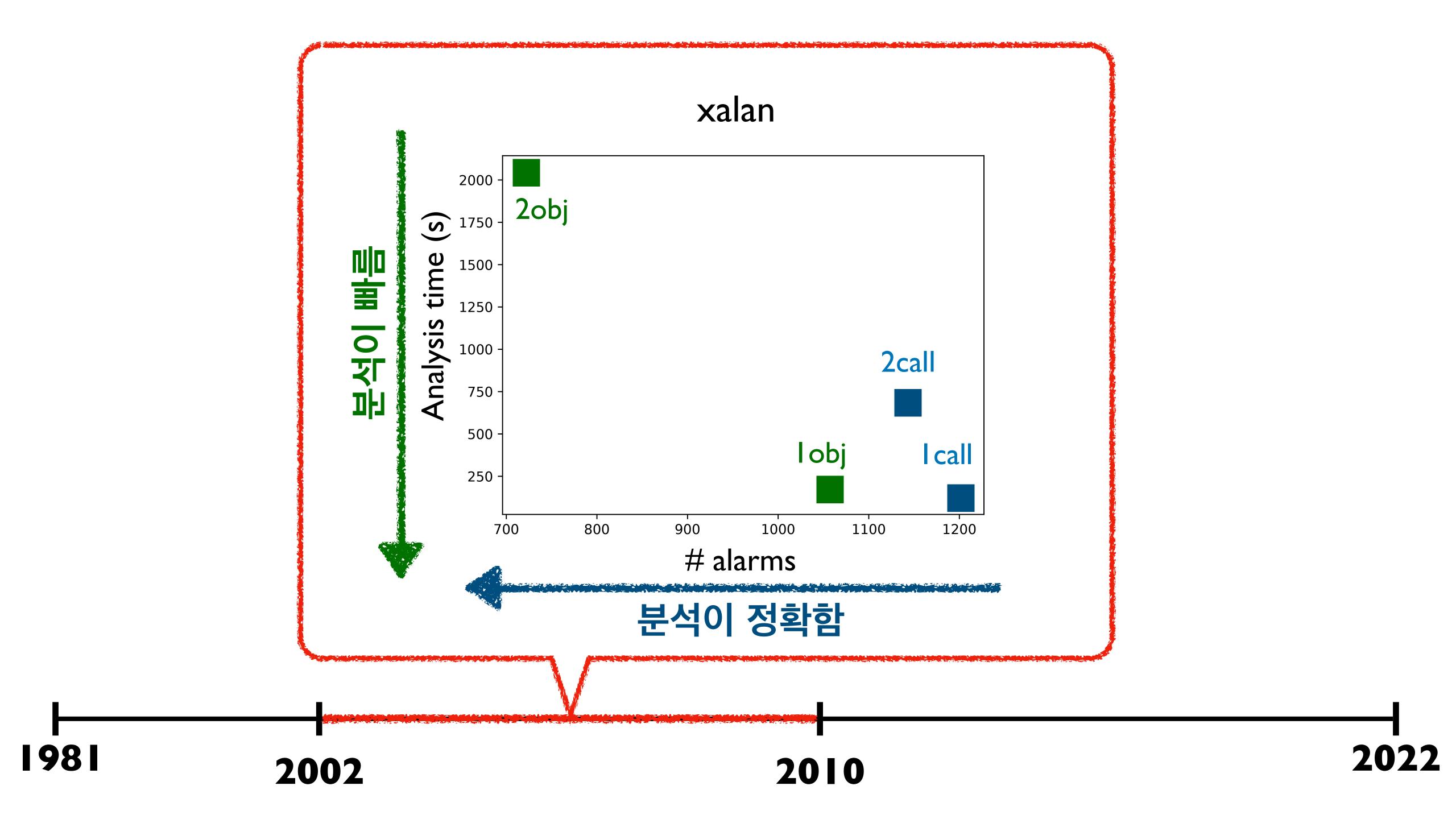
The main elements of our approach are the use of the Datalog learning for expections the processor maybe up and the

high-level [6,9]) is far from new. Our novel optimization a database, by specifically targeting the indexing scheme an thermore, our approach is entirely Datalog based, encoding declaratively the logic required both for call graph constrution as well as for handling the full semantic complex

2022 2010

1981

2002



• 값 기반 요약 방식이 우수하다고 강의되는 중

Object-Sensitivity

- The dominant flavor of context-sensitivity for objecting languages.
- It uses object abstractions (i.e. allocation sites) as qualifying a method's local variables with the alloc receiver object of the method call.

```
class A { void m() { return; } }
...
b = new B();
b.m();
The context of m is the allocation site of b.
```

Object-Sensitivity (vs. call-site sensitivity)

```
program

class S {
    Object id(Object a) { return a; }
    Object id2(Object a) { return id(a) }
}
class C extends S {
    void fun1() {
        Object a1 = new A1();
        Object b1 = id2(a1);
    }
}
class D extends S {
    void fun2() {
        Object a2 = new A2();
        Object b2 = id2(a2);
    }
}

Yannis Smaragdakis
Lighterity of Athers
```

Object-sensitive pointer

- Milanova, Rountev, and Ryder. Parameteriz sensitivity for points-to analysis for Java. AC Eng. Methodol., 2005.
- Context-sensitive interprocedural pointer analysis
- For context, use stack of receiver objects
- (More next week?)
- Lhotak and Hendren. Context-sensitive poir worth it? CC 06
- Object-sensitive pointer analysis more precise that for Java
- Likely to scale better

Lecture Notes: Pointer Analysis

15-819O: Program Analysis Jonathan Aldrich jonathan.aldrich@cs.cmu.edu

Lecture 9

1 Motivation for Pointer Analysis

In programs with pointers, program analysis can become more Consider constant-propagation analysis of the following progr

 $egin{array}{lll} 1: & z:=1 \ 2: & p:=\&z \ 3: & *p:=2 \ 4: & {
m print}\,z \end{array}$

In order to analyze this program correctly we must be aw instruction 3 p points to z. If this information is available we car flow function as follows:

 $f_{CP}[\![*p:=y]\!](\sigma)=[z\mapsto\sigma(y)]\sigma$ where must-point-to

When we know exactly what a variable x points to, we say t must-point-to information, and we can perform a *strong update* variable z, because we know with confidence that assigning to z. A technicality in the rule is quantifying over all z such point to z. How is this possible? It is not possible in C or Java; a language with pass-by-reference, for example C++, it is possinames for the same location are in scope.

Of course, it is also possible that we are uncertain to which distinct locations p points. For example:

Pointer Analysis

Yannis Smaragdakis University of Athens smaragd@di.uoa.gr

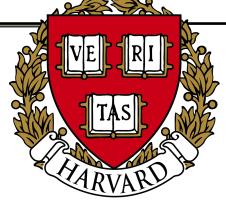
> George Balatsouras University of Athens gbalats@di.uoa.gr

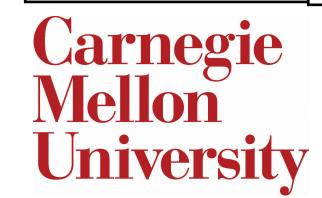
the essence of knowledge Boston — Deli





National and Kapodistrian University of Athens

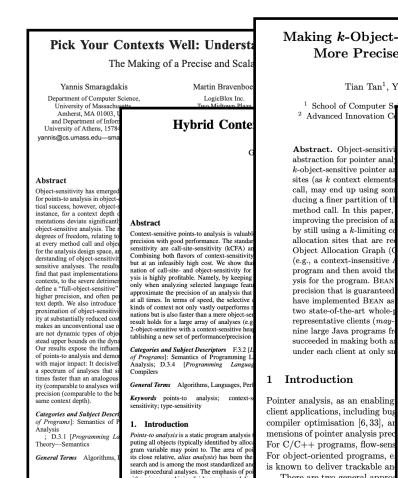






1981 2002 2010

값 기반 요약을 대상으로 하는 연구가 쏟아짐



Making k-Object-Sensitive Pointer A More Precise with Still k-Limiti

Tian Tan¹, Yue Li¹, and Jingling Xue^{1,2}

School of Comput

Abstract. Object-sens abstraction for pointer ar k-object-sensitive pointe sites (as k context eleme call, may end up using s ducing a finer partition of method call. In this pap

mproving the precision o by still using a k-limiting of

allocation sites that are re Object Allocation Graph (C Mainstream points-to analysis tech Object Allocation Graph (
(e.g., a context-insensitive A program and then avoid the ysis for the program. BEAN precision that is guaranteed such as call graph construction, devirtualis. have implemented Bean as two state-of-the-art whole-prepresentative clients (may-site abstraction, Mahjong enables an allo points-to analysis to run significantly faster nine large Java programs fr nearly the same precision for type-depend succeeded in making both as

Introduction

client applications, including bug compiler optimisation [6, 33] mensions of pointer analysis pred For C/C++ programs, flow-sens For object-oriented programs, is known to deliver trackable

There are two general approx oriented programs, call-site-ser [24,29] (among others). A k-CFA call by using a sequence of ksite). In contrast, a k-object-se k labels with each denoting a:

lemonstrate its effectiveness by discussi why it is a better alternative of the alloca

or type-dependent clients and evaluat

for many program analyses where call gra

in cost and precision for specific grou

lems. We do not need a different point

CCS Concepts • Theory of compu

Modeling the H

Context sensitivity is an essential techni observed that applying context sensitivi balance between analysis precision and do not provide much insight into what ch principled approach for identifying precis explain where most of the imprecision as an efficient algorithm to recognize these

tradeoffs between analysis precision and sp Our experimental results on standard b applies context sensitivity partially, only on (98.8%) of the precision of a highly-precise of with a context-sensitive heap), with a sub CCS Concepts: • Theory of computation Additional Key Words and Phrases: static a

ACM Reference Format: Yue Li, Tian Tan, Anders Møller, and Ya Pointer Analysis. Proc. ACM Program. Lang.

1 INTRODUCTION Pointer analysis is a fundamental i

pointer variables in a program. Such i inter-procedural control flow in object

engineering tools, e.g., for bug deter analysis [Arzt et al. 2014; Grech and S tion [Fink et al. 2008; Pradel et al. 20 Sridharan et al. 2007]. For decades, numerous analysis ted

precise and more efficient, especially fo Balatsouras 2015; Sridharan et al. 20 precision is context sensitivity [Mila Smaragdakis et al. 2011], which allows to separate the static abstractions of diff

found during the pre-analysis, M a common type. We formulate the type-consistency of two objects a

Scalability-Firs

Self-Tuning

Tian Tar

TIAN TAN, Aarhus University, Denma ANDERS MØLLER, Aarhus Univers YANNIS SMARAGDAKIS, University

Aarhus Universit yueli@cs.au.dk

Precision-Guided Context Sensitivity for Pointer Ana

ABSTRACT gh precision, without running the analysis multiple to

choose one that leads to reasonable analysis time and LER efficiently estimates the amount of points-to context-sensitivity. It then selects an appropriate v ch method so that the total amount of points-to infe ounded, while utilizing the available space to maximize Our experimental results demonstrate that SCALER ac dictable scalability for all the evaluated programs (e.g., s an reach 10x for 2-object-

CCS CONCEPTS

KEYWORDS

atic analysis, points-to analysis, Java he 26th ACM Joint European Software En

inter analysis is a family of static analysis techn oundation for many other analyses and software eng sks, such as program slicing [36, 39], reflection analys g detection [13, 26], security analysis [1, 23], program [8, 27], and program debugging and compre 2475-1421/2017/10-ART100



Learning Graph-based Heuristic

without Handcrafting Applicati

MINSEOK JEON, MYUNGHO LEE, and HAK

We present GRAPHICK, a new technique for automatical

Striking a balance between precision and scalability o

heuristics. For example, because applying context ser

impractical, pointer analysis typically uses a heuristic to

Past research has shown that exploiting the program'

cost-effective analysis heuristics, promoting the recent

graph representations of programs obtained from a pre

such heuristics remains challenging, requiring a great de

aim to reduce this burden by learning graph-based heuri-

application-specific features. To do so, we present a features.

algorithm for learning analysis heuristics within the lang

used it to learn graph-based heuristics for object sens

show that our approach is general and can generate hi

repair tools [Gao et al. 2015; Hong et al. 2020; I

ACM Reference Format:

1 INTRODUCTION

*Corresponding author

Data-Driven Context-Sensitivity for Points-to Analysis

SEHUN JEONG, Korea University, Repul SUNGDEOK CHA. Korea University Re HAKJOO OH[†], Korea University, Repu

We present a new data-driven approach to ac for Java. While context-sensitivity has great other precision-improving techniques, it is d most from context-sensitivity and decide how designing such rules is a nontrivial and labor overcome these challenges, we propose an aut context-sensitivity from codebases. In our app heuristic rules, in disjunctive form of propertie context-sensitivity. We present a greedy alg We implemented our approach in the Doop is experimental results show that our approach

CCS Concepts: • Theory of computation Additional Key Words and Phrases: Da

ACM Reference Format: Sehun Jeong, Minseok Jeon, Sungdeok Cha, an Analysis. Proc. ACM Program. Lang. 1, OOPSL

heuristics are as competitive as the existing state-of-the-Points-to analysis is one of the most in CCS Concepts: • Software and its engineering → Au for many program verification tasks (e.g., of subsequent higher-level program anal analysis, Context sensitivity, Heap abstraction

program understanding tools. For object-oriented languages, conte

Authors' email addresses: S. Jeong, gifaranga@ko H. Oh, hakjoo_oh@korea.ac.kr. Permission to make digital or hard copies of all o rovided that copies are not made or distribut he full citation on the first page. Copyrights f Abstracting with credit is permitted. To copy of prior specific permission and/or a fee. Request p © 2017 Association for Computing Machiner

• program size is far from a reliable predictor ever, 2type is not scalable for the former b





Precision-Preserving Yet Fast Object-Sensitive Pointer

sensitively for object-oriented langu programs, k-object-sensitive pointe k-obj to analyze only some methods analysis. While already effective, the that makes k-obj run significantly fa some of its selected variables/allocati by reasoning about context-free-lang

comparing it with the prior art in terr Additional Key Words and Phrases: I

ACM Reference Format: Jingbo Lu and Jingling Xue. 2019. Partial Context Sensitivity. Proc. ACM https://doi.org/10.1145/3360574

Additional Key Words and Phrases: Data-driven static an

Minseok Jeon, Myungho Lee, and Hakjoo Oh. 2020. Le without Handcrafting Application-Specific Features. (November 2020), 31 pages. https://doi.org/10.1145/342

Pointer analysis is a fundamental program analysis various software engineering tools. The goal of po estimate heap objects that pointer variables may essential for virtually all kinds of program analy et al. 2015; Livshits and Lam 2003; Naik et al. 2006

et al. 2014; Avots et al. 2005; Grech and Smarag program verifiers [Fink et al. 2008], symbolic exe jingling@cse.unsw.edu.au.

Analysis with Partial Context Sensitivity

JINGBO LU, UNSW Sydney, Austra JINGLING XUE, UNSW Sydney, As

Object-sensitivity is widely used as a values of k, where $k \leq 2$ typically. A consequently, are limited in the efficien Eagle is to enable k-obj to analyze a m based on a new CFL-reachability for

CCS Concepts: • Theory of comput

INTRODUCTION

For object-oriented languages suc precision for pointer analysis [Ll insensitive pointer analysis, such as once, producing one points-to set allocation site in the method. In multiple times under different ca thereby producing multiple point abstract objects for modeling ever

To tame the combinatorial expl sequence of k context elements. object-oriented programs: (1) k-cal of a method by its k-most-recent Authors' addresses: Jingbo Lu, UNSW Sy

Making Pointer Analysis More Precise by Unleashing the **Power of Selective Context Sensitivity**

TIAN TAN, Nanjing University, China

YUE LI*, Nanjing University, China XIAOXING MA, Nanjing University, China CHANG XU, Nanjing University, China

YANNIS SMARAGDAKIS, University of Athens, Greece

Traditional context-sensitive pointer analysis is hard to scale for large and complex Java programs. To addre this issue, a series of selective context-sensitivity approaches have been proposed and exhibit promising resul In this work, we move one step further towards producing highly-precise pointer analyses for hard-to-analyz Java programs by presenting the Unity-Relay framework, which takes selective context sensitivity to the nex level. Briefly, Unity-Relay is a one-two punch: given a set of different selective context-sensitivity approaches say $S = S_1, \ldots, S_n$, Unity-Relay first provides a mechanism (called Unity) to combine and maximize th precision of all components of S. When Unity fails to scale, Unity-Relay offers a scheme (called Relay) to pass and accumulate the precision from one approach S_i in S to the next, S_{i+1} , leading to an analysis that is

nore precise than all approaches in S. As a proof-of-concept, we instantiate Unity-Relay into a tool called BATON and extensively evaluate it or a set of hard-to-analyze Java programs, using general precision metrics and popular clients. Compared with the state of the art, BATON achieves the best precision for all metrics and clients for all evaluated programs The difference in precision is often dramatic—up to 71% of alias pairs reported by previously-best algorithms are found to be spurious and eliminated.

CCS Concepts: • Theory of computation \rightarrow Program analysis.

Additional Key Words and Phrases: Pointer Analysis, Alias Analysis, Context Sensitivity, Java Tian Tan, Yue Li, Xiaoxing Ma, Chang Xu, and Yannis Smaragdakis. 2021. Making Pointer Analysis More

Precise by Unleashing the Power of Selective Context Sensitivity. Proc. ACM Program. Lang. 5, OOPSLA, Article 147 (October 2021), 27 pages. https://doi.org/10.1145/3485524

Pointer analysis is important for an array of real-world applications such as bug detection [Chandra et al. 2009; Naik et al. 2006], security analysis [Arzt et al. 2014; Livshits and Lam 2005], program verification [Fink et al. 2008; Pradel et al. 2012] and program understanding [Li et al. 2016; Sridharar

1981 2022 2010 2002

"For comparison, we have included 2call to demonstrate the superiority of object sensitivity over call-site sensitivity"

ointer analysis is a fundamental family of static analyses

ointer variables in a program. Such information is essent ter-procedural control flow in object-oriented programs, an agineering tools, e.g., for bug detection [Chandra et al. 2

recise and more efficient, especially for object-oriented lange

alatsouras 2015; Sridharan et al. 2013]. One of the most s ecision is *context sensitivity* [Milanova et al. 2002, 2005; Sl

aragdakis et al. 2011], which allows each program method to

- Tan et al. [2016]

Minseok Jeon, Sehun Jeong, Sungdeok Cha, and Hakjoo Oh. 2017. A Machine-Learning Algorithm with Disjunctive Model for Data-Driven Program Analysis. *ACM Trans. Program. Lang. Syst.* 9, 4, Article 39

for tuning the analysis performance for real-world applications. Practical static analysis tools use a

variety of heuristics to optimize their performance. For example, context-sensitivity is essential

or analyzing object-oriented programs, as it distinguishes method's local variables and objects in

different calling-contexts. However, applying context-sensitivity to all methods in the progran

does not scale and therefore real-world static analyzers apply context-sensitivity only to profitable

nethods determined by some heuristic rules [Smaragdakis et al. 2014]. Another example is a

relational analysis such as ones with Octagons [Miné 2006]. Because it is impractical to keep track

of all variable relationships in the program, static analyzers employ variable-clustering heuris

December 2017), 42 pages. https://doi.org/0000001.0000001

Making k-Object-Sensitive P Pick Your Contexts Well: Underst A Machine-Learning Algorithm with Disjunctive Model for Introspective Analysis: Context-Sensitivity, Across the **Hybrid Context-Sensitivity for** More Precise with Still k The Making of a Precise and Sca Data-Driven Program Analysis Yannis Smaragdakis George Kastrinis George Balatsoura **Scalability-First Pointer Analy** Tian Tan¹, Yue Li¹, and Jinglin MINSEOK JEON, SEHUN JEONG st , SUNGDEOK CHA, and HAKJOO OH † , Korea University **Precision-Guided Context Sensitivity Self-Tuning Context-Sensit** YUE LI, Aarhus University, Denmark We present a new machine-learning algorithm with disjunctive model for data-driven program analys AN TAN, Aarhus University, Denmark ne major challenge in static program analysis is a substantial amount of manual effort required for tuning ANDERS MØLLER, Aarhus University, Denmark the analysis performance. Recently, data-driven program analysis has emerged to address this challenge abstraction for pointer analysis in object-orient k-object-sensitive pointer analysis, which uses a approach has proven promising for various program analysis tasks, its effectiveness has been limited du call, may end up using some context eleme lisjunctive, program properties. To overcome this shortcoming, this article presents a new disjunctive mode or data-driven program analysis as well as a learning algorithm to find the model parameters. Our model use poolean formulas over atomic features and therefore is able to express nonlinear combinations of program properties. Key technical challenge is efficiently determine a set of good boolean formulas as brute-force by still using a k-limiting context abstraction. earch would simply be impractical. We present a stepwise and greedy algorithm that efficiently learn n efficient algorithm to recognize these flow patterns in a given poolean formulas. We show the effectiveness and generality of our algorithm with two static analyzer Object Allocation Graph (OAG), which is built (e.g., a context-insensitive Andersen's analysis) program and then avoid them in the subsequent ntext-sensitive points-to analysis for Java and flow-sensitive interval analysis for C. Experimental result show that our automated technique significantly improves the performance of the state-of-the-art technique ysis for the program. BEAN is generally more precision that is guaranteed to be as good as k-c have implemented BEAN as an open-source tool CCS Concepts: • Theory of computation \rightarrow Program analysis; • Computing r we implemented BEAN as an open-source state-of-the-art whole-program point Theory of computation → Program analysis chine learning approaches static analysis, points-to analysis, Java under each client at only small increases in inter Analysis. Proc. ACM Program. Lang. 2, OOPSLA, Article 14

2022

Pointer analysis, as an enabling technology, plays

nensions of pointer analysis precision are flow-sen

For C/C++ programs, flow-sensitivity is needed

For object-oriented programs, e.g., Java programs

s known to deliver trackable and useful precisi

There are two general approaches to achie

call by using a sequence of k call sites (i.e., k labe

24, 29] (among others). A k-CFA analysis

1981

"We do not consider call-site sensitive analyses as they are typically both less precise and scalable..."

- Li et al. [2018]

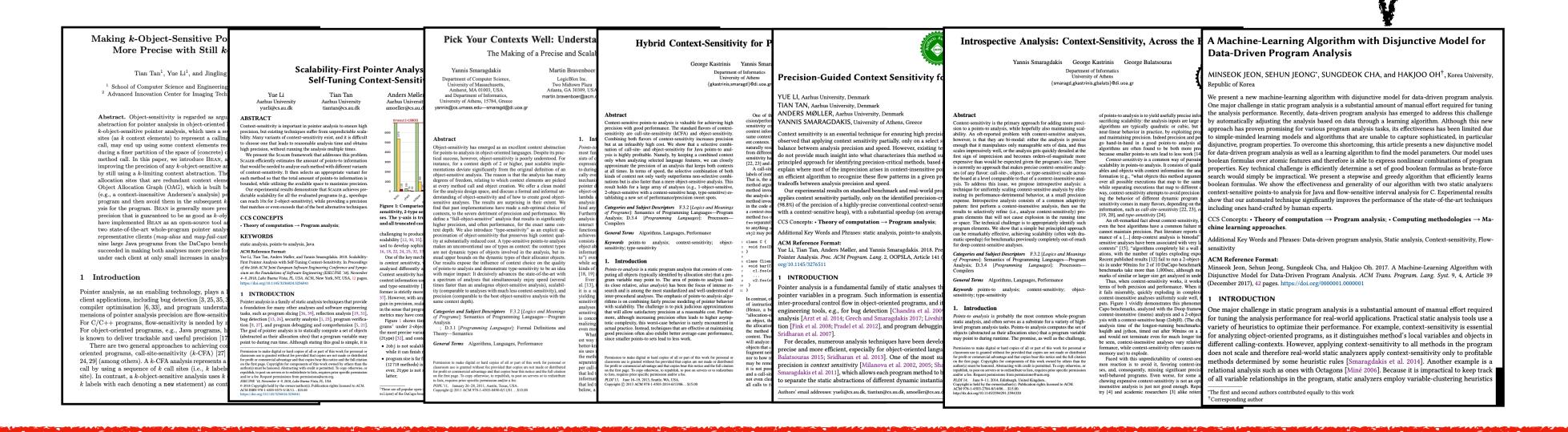
Making k-Object-Sensitive P Pick Your Contexts Well: Underst Introspective Analysis: Context-Sensitivity, Across the A Machine-Learning Algorithm with Disjunctive Model for **Hybrid Context-Sensitivity for** More Precise with Still k The Making of a Precise and Scal Data-Driven Program Analysis Yannis Smaragdakis George Kastrinis George Balatsoura **Scalability-First Pointer Analy** Tian Tan¹, Yue Li¹, and Jinglin MINSEOK JEON, SEHUN JEONG*, SUNGDEOK CHA, and HAKJOO OH † , Korea Universit **Precision-Guided Context Sensitivity Self-Tuning Context-Sensit** YUE LI. Aarhus University. Denmark We present a new machine-learning algorithm with disjunctive model for data-driven program analysi AN TAN, Aarhus University, Denmark ne major challenge in static program analysis is a substantial amount of manual effort required for tuning ANDERS MØLLER, Aarhus University, Denmark the analysis performance. Recently, data-driven program analysis has emerged to address this challenge abstraction for pointer analysis in object-orient k-object-sensitive pointer analysis, which uses a approach has proven promising for various program analysis tasks, its effectiveness has been limited du sites (as k context elements) to represent a call call, may end up using some context eleme lisjunctive, program properties. To overcome this shortcoming, this article presents a new disjunctive mode or data-driven program analysis as well as a learning algorithm to find the model parameters. Our model use poolean formulas over atomic features and therefore is able to express nonlinear combinations of program properties. Key technical challenge is efficiently determine a set of good boolean formulas as brute-force by still using a k-limiting context abstraction. n efficient algorithm to recognize these flow patterns in a given earch would simply be impractical. We present a stepwise and greedy algorithm that efficiently learn poolean formulas. We show the effectiveness and generality of our algorithm with two static analyzer Object Allocation Graph (OAG), which is built left. (e.g., a context-insensitive Andersen's analysis) program and then avoid them in the subsequent ntext-sensitive points-to analysis for Java and flow-sensitive interval analysis for C. Experimental result show that our automated technique significantly improves the performance of the state-of-the-art technique ysis for the program. BEAN is generally more precision that is guaranteed to be as good as k-c have implemented BEAN as an open-source tool ${\tt CCS\ Concepts: \bullet Theory\ of\ computation} \rightarrow {\tt Program\ analysis; \bullet Computing\ methodological}$ we implemented BEAN as an open-source state-of-the-art whole-program point Theory of computation → Program analysis chine learning approaches nine large Java programs from the DaCapo static analysis, points-to analysis, Java under each client at only small increases in inter Analysis. Proc. ACM Program. Lang. 2, OOPSLA, Article 14 Minseok Jeon, Sehun Jeong, Sungdeok Cha, and Hakjoo Oh. 2017. A Machine-Learning Algorithm with Disjunctive Model for Data-Driven Program Analysis. *ACM Trans. Program. Lang. Syst.* 9, 4, Article 39 December 2017), 42 pages. https://doi.org/0000001.0000001 ointer analysis is a fundamental family of static analyses Pointer analysis, as an enabling technology, plays ointer variables in a program. Such information is essent compiler optimisation [6, 33], and program under igineering tools, e.g., for bug detection [Chandra et al. 20 nensions of pointer analysis precision are flow-sen for tuning the analysis performance for real-world applications. Practical static analysis tools use a For C/C++ programs, flow-sensitivity is needed by variety of heuristics to optimize their performance. For example, context-sensitivity is essential For object-oriented programs, e.g., Java programs or analyzing object-oriented programs, as it distinguishes method's local variables and objects in s known to deliver trackable and useful precisi different calling-contexts. However, applying context-sensitivity to all methods in the progran There are two general approaches to achie recise and more efficient, especially for object-oriented langu does not scale and therefore real-world static analyzers apply context-sensitivity only to profitable oriented programs, call-site-sensitivity (k-CFA) [2 alatsouras 2015; Sridharan et al. 2013]. One of the most strecision is *context sensitivity* [Milanova et al. 2002, 2005; Sh [24, 29] (among others). A k-CFA analysis nethods determined by some heuristic rules [Smaragdakis et al. 2014]. Another example is a relational analysis such as ones with Octagons [Miné 2006]. Because it is impractical to keep track call by using a sequence of k call sites (i.e., k labe aragdakis et al. 2011], which allows each program method to of all variable relationships in the program, static analyzers employ variable-clustering heuris

1981 2002 2010

"We do not discuss the performance of our approach for call-sitesensitivity since call-site-sensitivity is less important than others

. . .

- Jeon et al. [2019]



1981 2002 2010

"We do not discuss the performance of our approach for call-sitesensitivity since call-site-sensitivity is less important than others

- Jeon et al. [2019]

Making k-Object-Sensitive P More Precise with Still A Tian Tan¹, Yue Li¹, and Jingli abstraction for pointer analysis in object-orient k-object-sensitive pointer analysis, which uses by still using a k-limiting context abstraction ysis for the program. Bean is generally mo precision that is guaranteed to be as good as k-Pointer analysis, as an enabling technology, play

For object-oriented programs, e.g., Java program

call by using a sequence of k call sites (i.e., k lab

Scalability-First Pointer Analy **Self-Tuning Context-Sensi**

The Making of a Precise and Sca

Pick Your Contexts Well: Underst **Hybrid Context-Sensitivity for**

Precision-Guided Context Sensitivity

UE LI, Aarhus University, Denmark AN TAN, Aarhus University, Denmark ANDERS MØLLER, Aarhus University, Denmar

inter Analysis. Proc. ACM Program. Lang. 2, OOPSLA, Article 14

inter variables in a program. Such information is esse

alatsouras 2015; Sridharan et al. 2013]. One of the most recision is *context sensitivity* [Milanova et al. 2002, 2005; S

ntrospective Analysis: Context-Sensitivity, Across

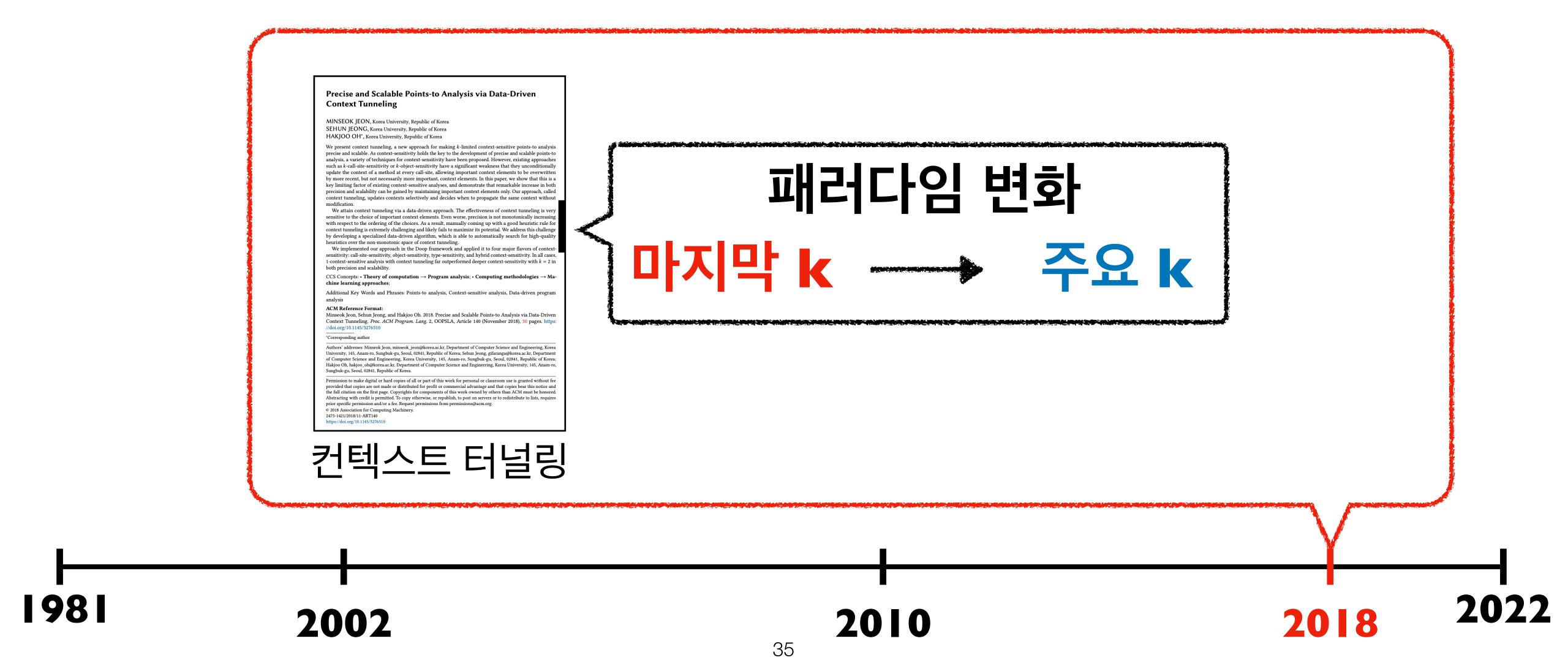
Yannis Smaragdakis George Kastrinis George Balatsour



1981 2002

2010

2022



Return of CFA: Call-Site Sensitivity Can Be Superior to Object Sensitivity Even for Object-Oriented Programs

MINSEOK JEON and HAKJOO OH*, Korea University, Republic of Korea

In this paper, we challenge the commonly-accepted wisdom in static analysis that object sensitivity is superior to call-site sensitivity for object-oriented programs. In static analysis of object-oriented programs, object sensitivity has been established as the dominant flavor of context sensitivity thanks to its outstanding precision. On the other hand, call-site sensitivity has been regarded as unsuitable and its use in practice has been constantly discouraged for object-oriented programs. In this paper, however, we claim that call-site sensitivity is generally a superior context abstraction because it is practically possible to transform object sensitivity into more precise call-site sensitivity. Our key insight is that the previously known superiority of object sensitivity holds only in the traditional *k*-limited setting, where the analysis is enforced to keep the most recent *k* context elements. However, it no longer holds in a recently-proposed, more general setting with context tunneling. With context tunneling, where the analysis is free to choose an arbitrary *k*-length subsequence of context strings, we show that call-site sensitivity can simulate object sensitivity almost completely, but not vice versa. To support the claim, we present a technique, called OBJ2CFA, for transforming arbitrary context-tunneled object sensitivity into more precise, context-tunneled call-site-sensitivity. We implemented OBJ2CFA in Doop and used it to derive a new call-site-sensitive analysis from a state-of-the-art object-sensitive pointer analysis. Experimental results confirm that the resulting call-site sensitivity outperforms object sensitivity in precision and scalability for real-world Java programs. Remarkably, our results show that even 1-call-site sensitivity can be more precise than the conventional 3-object-sensitive analysis.

1 INTRODUCTION

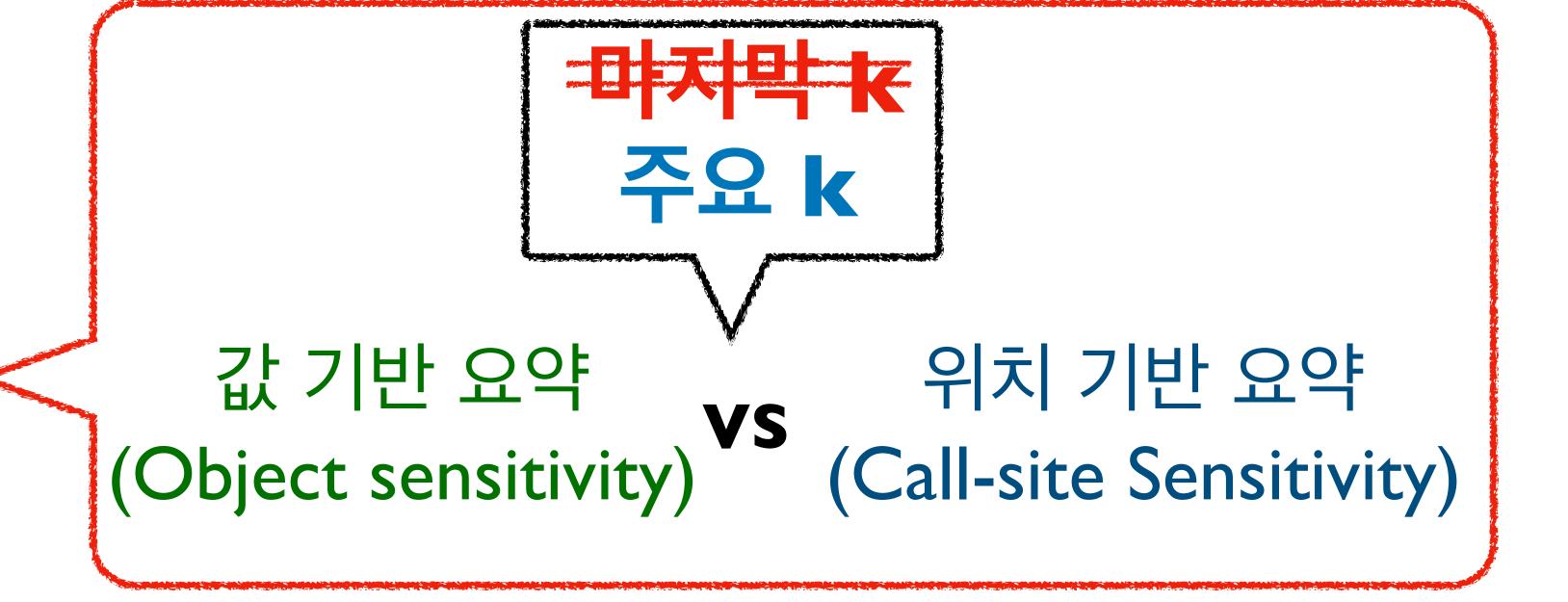
"Since its introduction, object sensitivity has emerged as the dominant flavor of context sensitivity for object-oriented languages."

-Smaragdakis and Balatsouras [2015]

Context sensitivity is critically important for static program analysis of object-oriented programs. A context-sensitive analysis associates local variables and heap objects with context information of method calls, computing analysis results separately for different contexts. This way, context sensitivity prevents analysis information from being merged along different call chains. For object-oriente

for inc Smara Smara The and Pri the chi 1981] it allocat

sensitive analysis [Milanova et al. 2002, 2005; Smaragdakis et al. 2011] maintains a sequence of



1981 2002

2010

2018 2022

Exercise

```
class S {
 Object id(Object a) { return a; }
 Object id2(Object a) { return id(); }
class C extends S {
 void fun1() {
   Object a1 = new A1();
   Object b1 = id2(a1);
 }}
class D extends S {
 void fun2() {
   Object a2 = new A2();
   Object b2 = id2(a2);
 }}
 ● What is the result of 1-object-sensitive analysis? 		 정확함

    Explain the strength of object-sensitivity over call-site-sensitivity.
```

Hakjoo Oh

```
Exercise
class S {
  Object id(Object a) { return a; }
  Object id2(Object a) { return id(); }
class C extends S {
  void fun1() {
    Object a1 = new A1();
    Object b1 = id2(a1);
  }}
class D extends S {
  void fun2() {
    Object a2 = new A2();
    Object b2 = id2(a2);
  }}
  • What is the result of 1-call-site-sensitive analysis?
                                                  정확함
  What is the result of 1-object-sensitive analysis?
  • Explain the strength of object-sensitivity over call-site-sensitivity. Lobi > call
```

Exercise

```
class S {
   Object id(Object a) { return a; }
   Object id2(Object a) { return id(); }
}
class C extends S {
   void fun1() {
      Object a1 = new A1();
      Object b1 = id2(a1);
   }}
class D extends S {
   void fun2() {
      Object a2 = new A2();
      Object b2 = id2(a2);
}
```

- What is the result of 1-call-site-sensitive analysis?
- What is the result of 1-object-sensitive analysis?
- Explain the strength of object-sensitivity over call-site-sensitivity.

질문:

이를 보일 수 있는 예제를 만들어 보자

Hakjoo Oh AAA616 2022 Fall, Lecture 8

October 18, 2022

28 / 31

Exercise

```
class S {
   Object id(Object a) { return a; }
   Object id2(Object a) { return id(); }
}
class C extends S {
   void fun1() {
      Object a1 = new A1();
      Object b1 = id2(a1);
   }}
class D extends S {
   void fun2() {
      Object a2 = new A2();
      Object b2 = id2(a2);
   }
}
```

- What is the result of 1-call-site-sensitive analysis?
- What is the result of 1-object-sensitive analysis?
- Explain the strength of object-sensitivity over call-site-sensitivity.

질문:

이를 보일 수 있는 예제를 만들어 보자

(6개월 후) 도저히 안만들어짐

가설: Call > Obj

Return of CFA: Call-Site Sensitivity Can Be Superior to Object Sensitivity Even for Object-Oriented Programs

MINSEOK JEON and HAKJOO OH*, Korea University, Republic of Korea

In this paper, we challenge the commonly-accepted wisdom in static analysis that object sensitivity is superior to call-site sensitivity for object-oriented programs. In static analysis of object-oriented programs, object sensitivity has been established as the dominant flavor of context sensitivity thanks to its outstanding precision. On the other hand, call-site sensitivity has been regarded as unsuitable and its use in practice has been constantly discouraged for object-oriented programs. In this paper, however, we claim that call-site sensitivity is generally a superior context abstraction because it is practically possible to transform object sensitivity into more precise call-site sensitivity. Our key insight is that the previously known superiority of object sensitivity holds only in the traditional *k*-limited setting, where the analysis is enforced to keep the most recent *k* context elements. However, it no longer holds in a recently-proposed, more general setting with context tunneling. With context tunneling, where the analysis is free to choose an arbitrary *k*-length subsequence of context strings, we show that call-site sensitivity can simulate object sensitivity almost completely, but not vice versa. To support the claim, we present a technique, called OBJ2CFA, for transforming arbitrary context-tunneled object sensitivity into more precise, context-tunneled call-site-sensitivity. We implemented OBJ2CFA in Doop and used it to derive a new call-site-sensitive analysis from a state-of-the-art object-sensitive pointer analysis. Experimental results confirm that the resulting call-site sensitivity outperforms object sensitivity in precision and scalability for real-world Java programs. Remarkably, our results show that even 1-call-site sensitivity can be more precise than the conventional 3-object-sensitive analysis.

1 INTRODUCTION

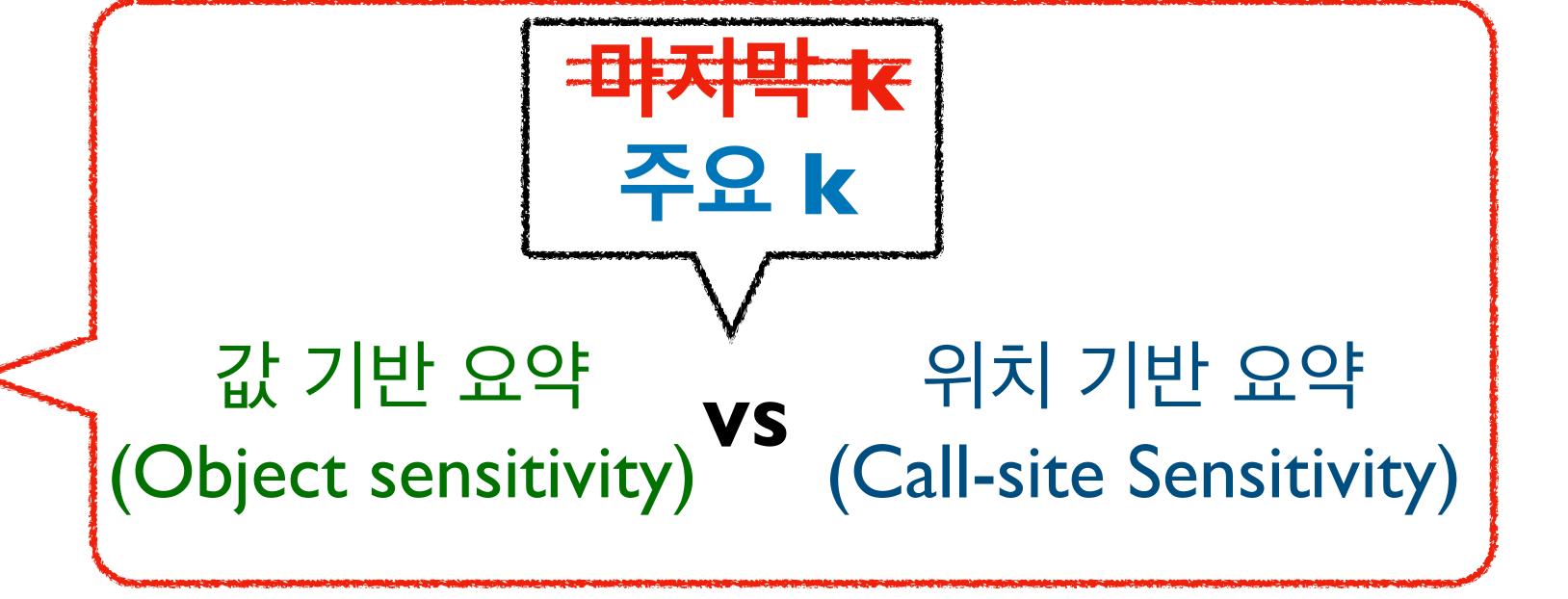
"Since its introduction, object sensitivity has emerged as the dominant flavor of context sensitivity for object-oriented languages."

-Smaragdakis and Balatsouras [2015]

Context sensitivity is critically important for static program analysis of object-oriented programs. A context-sensitive analysis associates local variables and heap objects with context information of method calls, computing analysis results separately for different contexts. This way, context sensitivity prevents analysis information from being merged along different call chains. For object-oriented

for inc Smara Smara The and Pri the chi 1981] i allocat

sensitive analysis [Milanova et al. 2002, 2005; Smaragdakis et al. 2011] maintains a sequence of



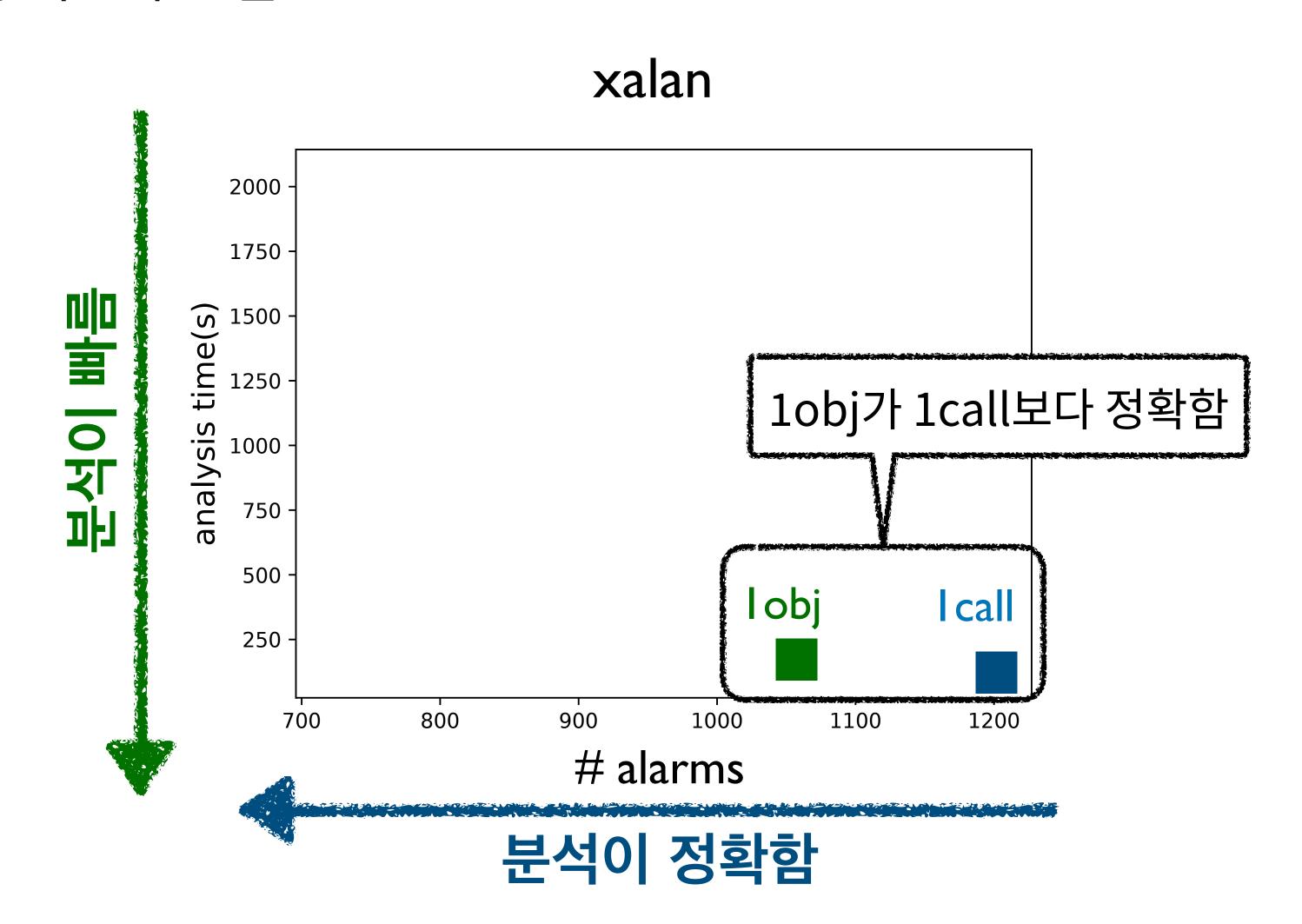
1981 2002

2010

2018 2022

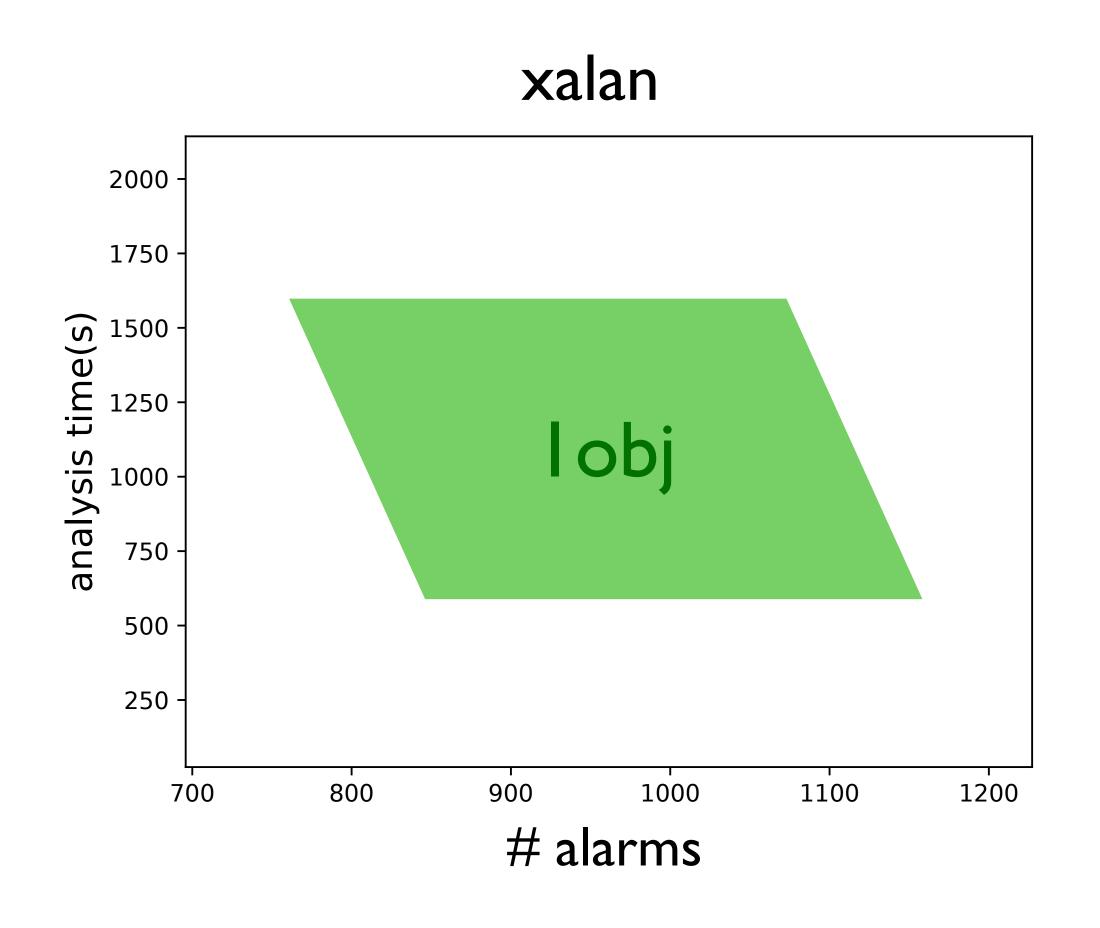
마지막 k 기반 요약에서의 성능 비교

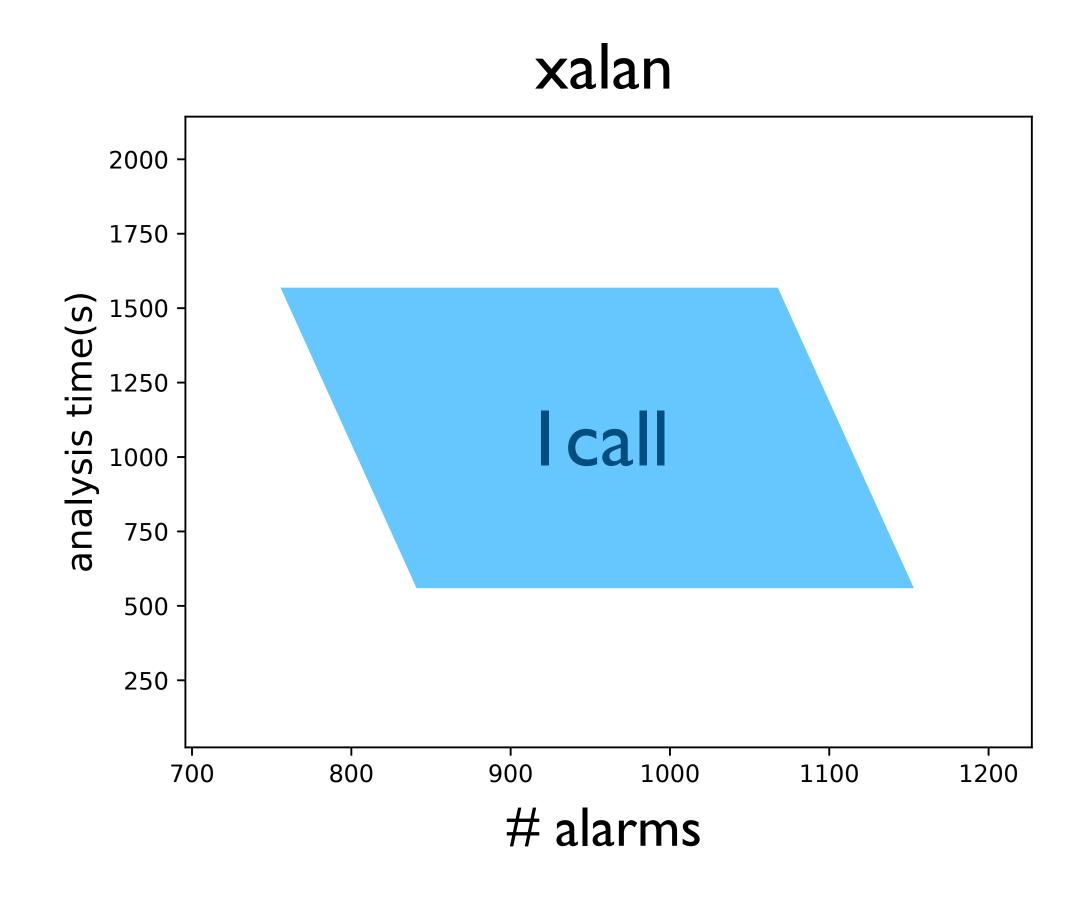
• 분석의 성능을 직접 비교하면 됨



주요 k 기반에서의 성능 비교

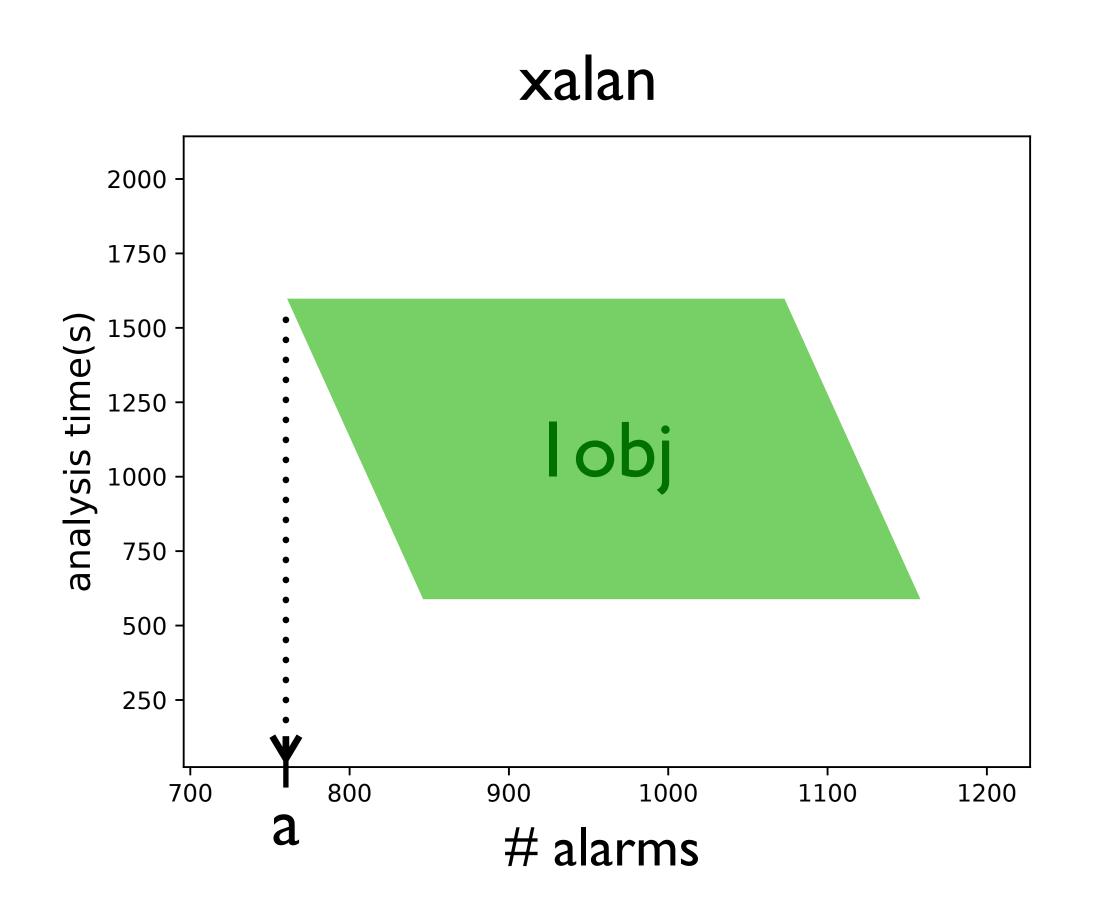
• 주요 요소 분류에 따라 성능이 바뀜

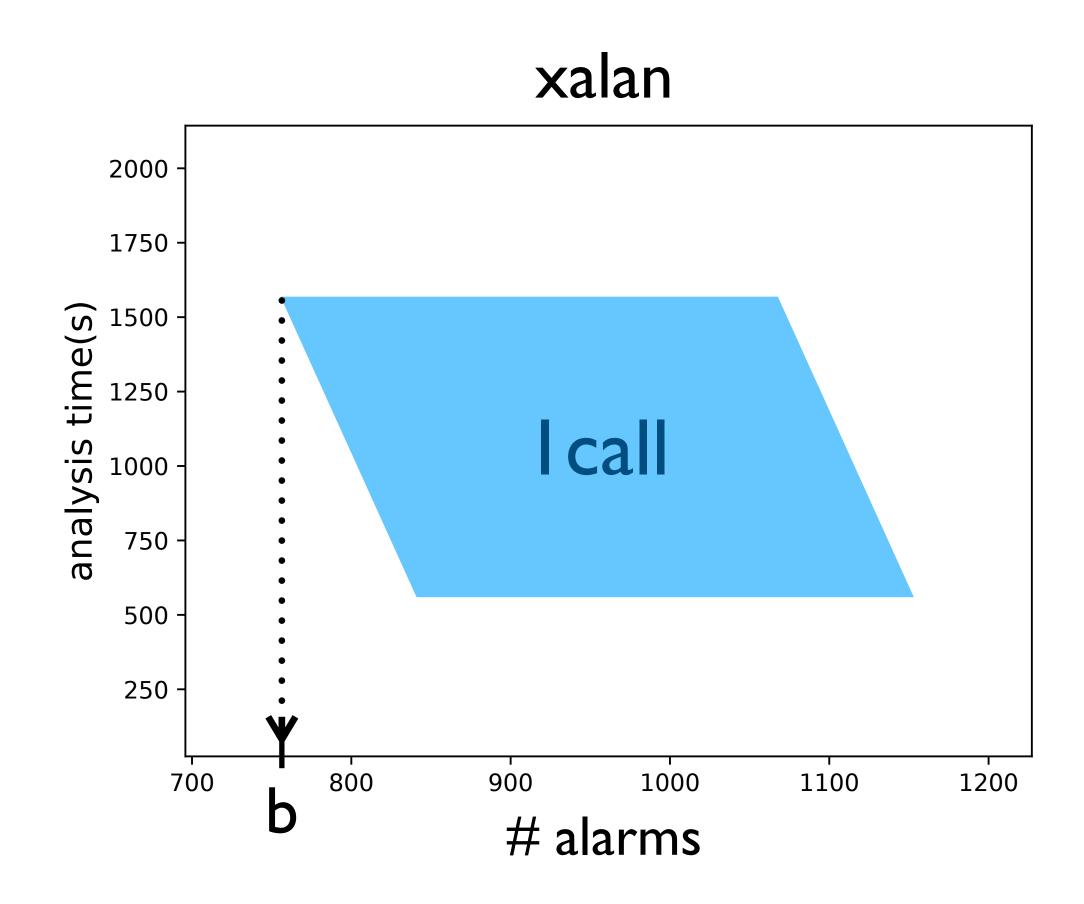




주요 k 기반에서의 성능 비교

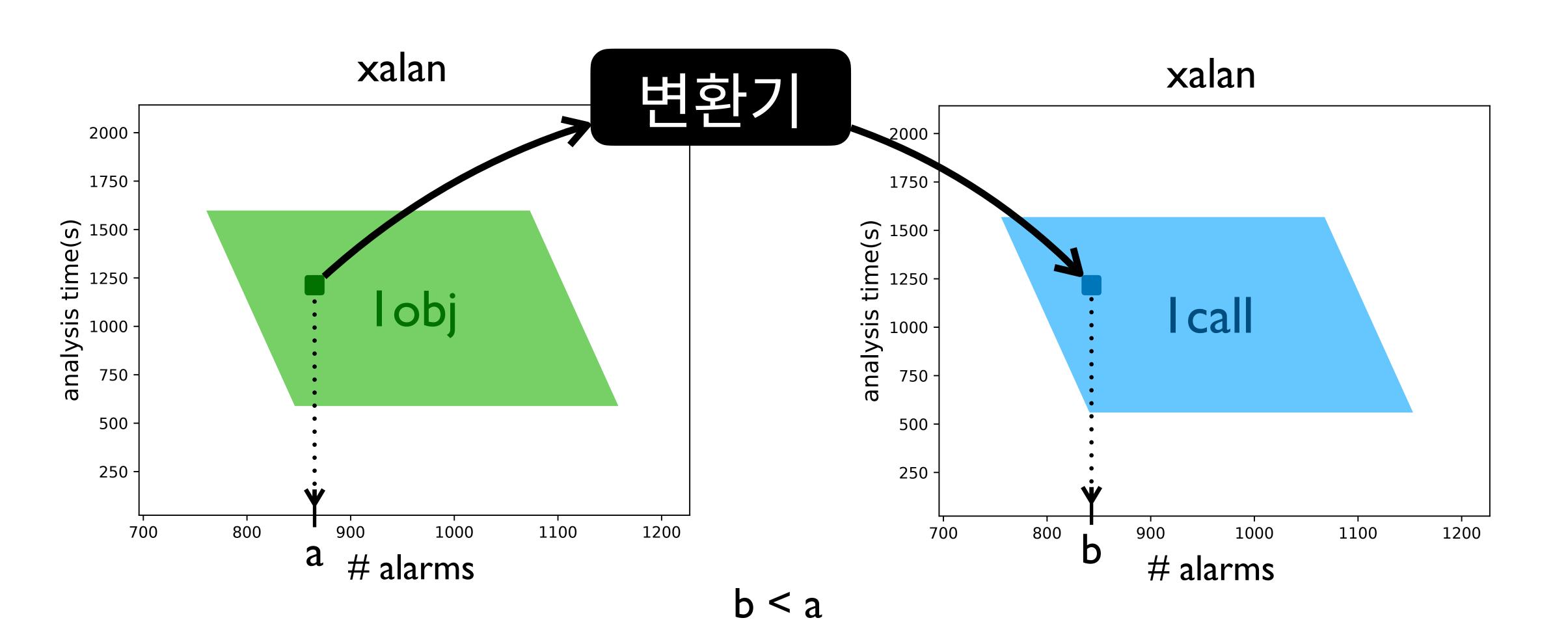
• **방법 1:** 최고 성능을 내는 분류를 알아내서 성능 비교하기 (e.g., b < a)





주요 k 기반에서의 성능 비교

• 방법 2: 주어진 obj를 더 정확한 call로 변환해주는 함수가 존재한다는 것을 보이기



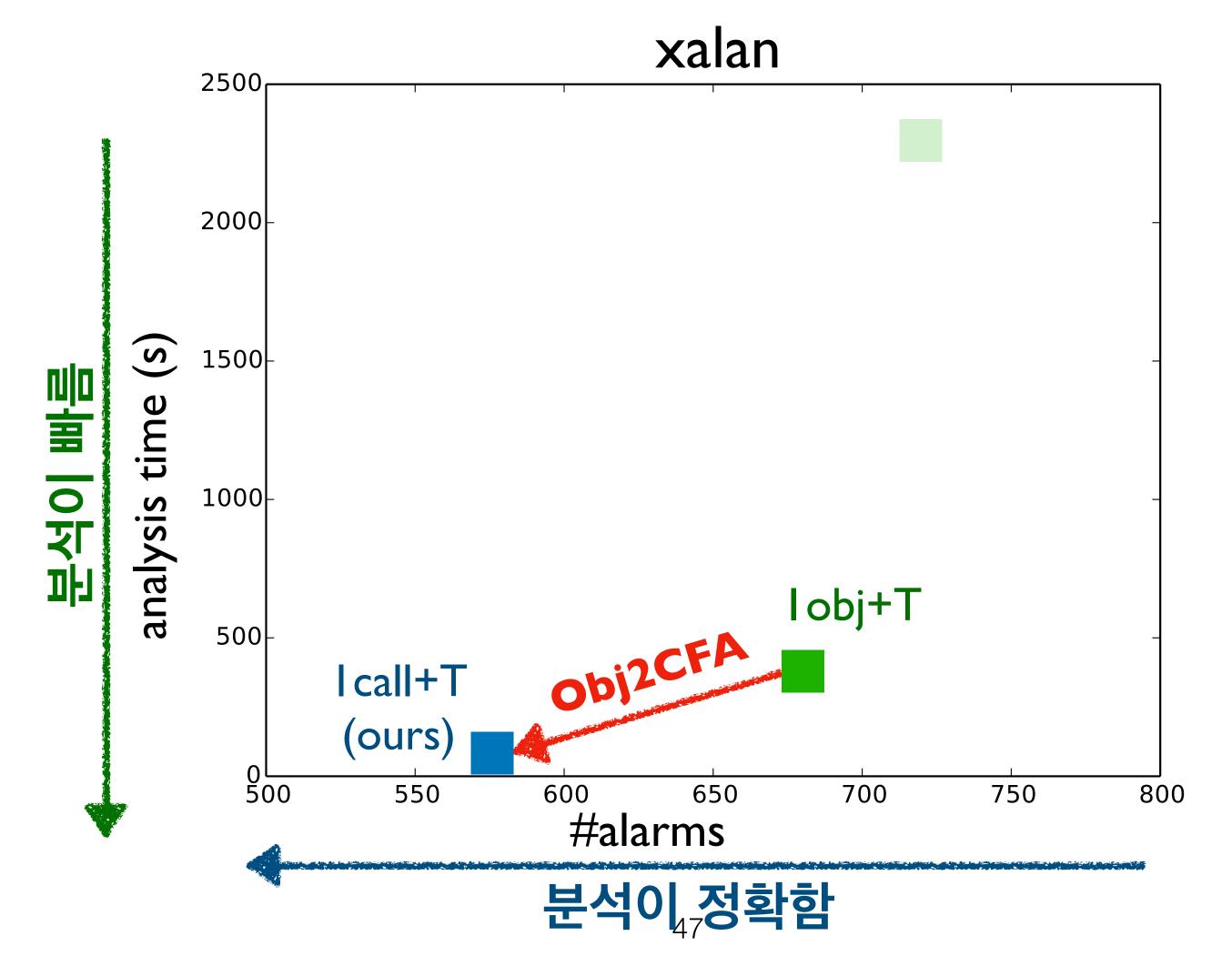
변환기: Obj2CFA

• Obj2CFA 는 주어진 값 기반 요약을 더 정확한 위치 기반 요약으로 바꾸어주는 함수



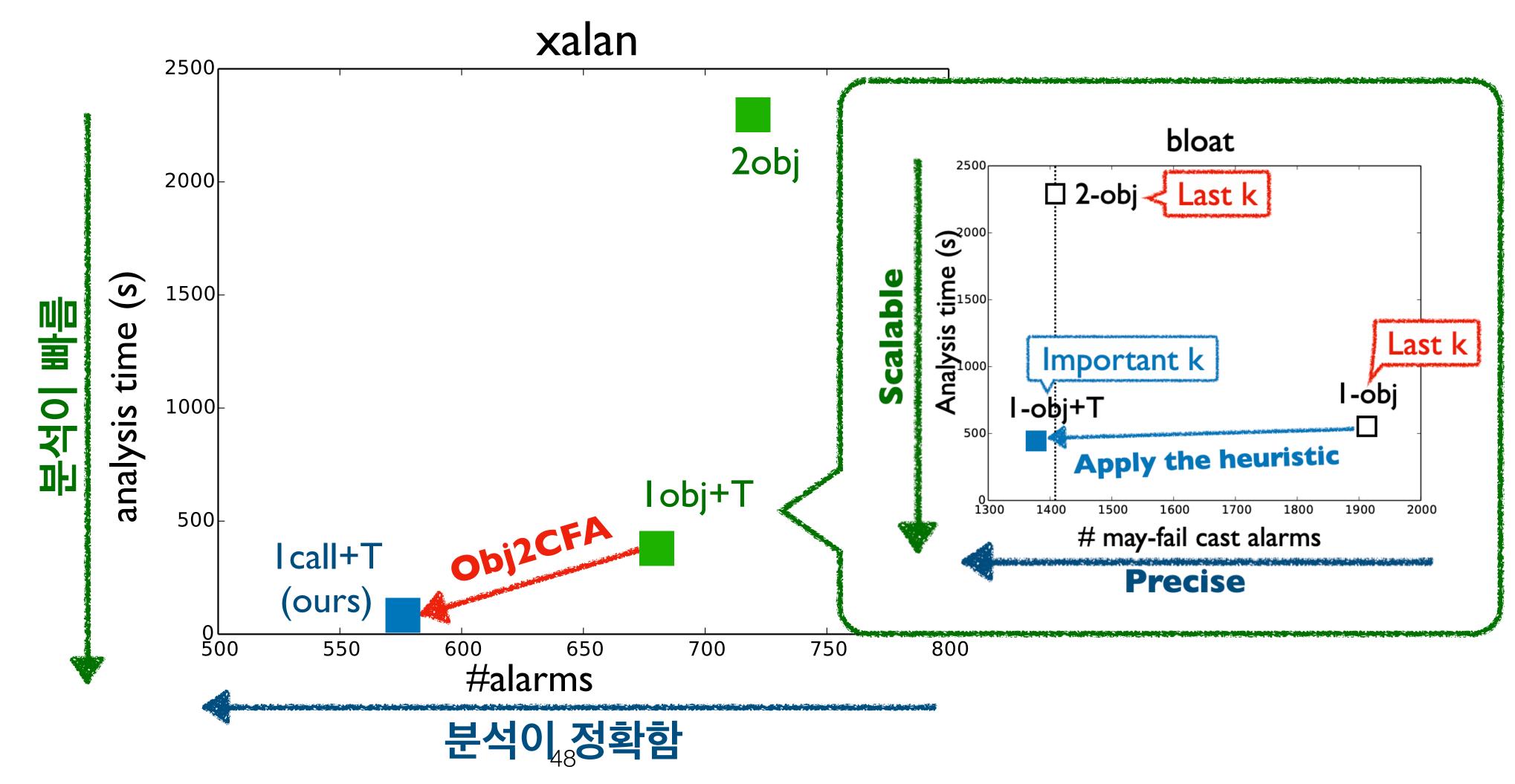
변환기: Obj2CFA

• Obj2CFA 는 주어진 값 기반 요약을 더 정확한 위치 기반 요약으로 바꾸어줌



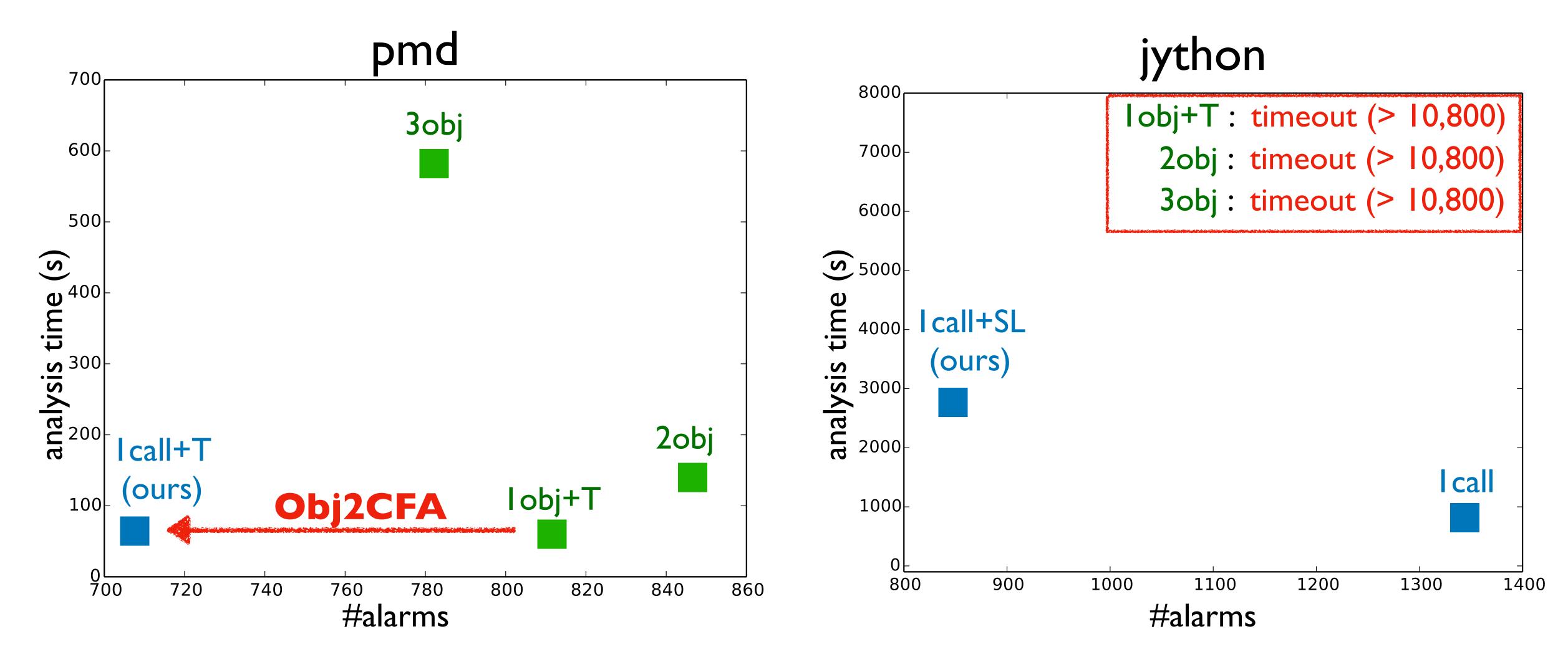
변환기: Obj2CFA

• Obj2CFA 는 주어진 값 기반 요약을 더 정확한 위치 기반 요약으로 바꾸어줌



위치 기반 요약 vs 값 기반 요약

● 변환된 위치 기반 요약은 현존하는 값 기반 요약들보다 높은 정확도와 속도를 보임



Some parts of the paper is too strong; this paper should be rejected. - A reviewer [Expert] POPL should accept this paper to encourage discussions. - A reviewer [Expert] **ICSE 2020** OOPSLA2019 **PLDI 2020** OOPSLA 2021 **POPL 2022** (Rejected) (Rejected) (Rejected) (Rejected) (Accepted)

논문 출판후

Call-Site vs. Object Sensitivity

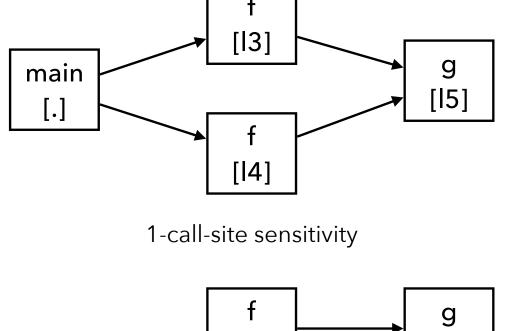
- In theory, their precision is incomparable
- In practice, object sensitivity generally outperforms call-site sensitivity for OO languages (like Java)

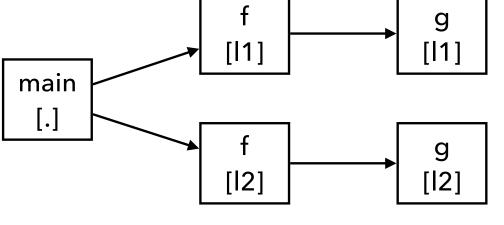
Call-site vs. Object Sensitivity

Typical example that benefits from object sensitivity:

```
class A:
    def g(self):
        return
    def f(self):
        return self.g() // 15

def main():
    a = A() // 11
    b = A() // 12
    a.f() // 13
    b.f() // 14
```



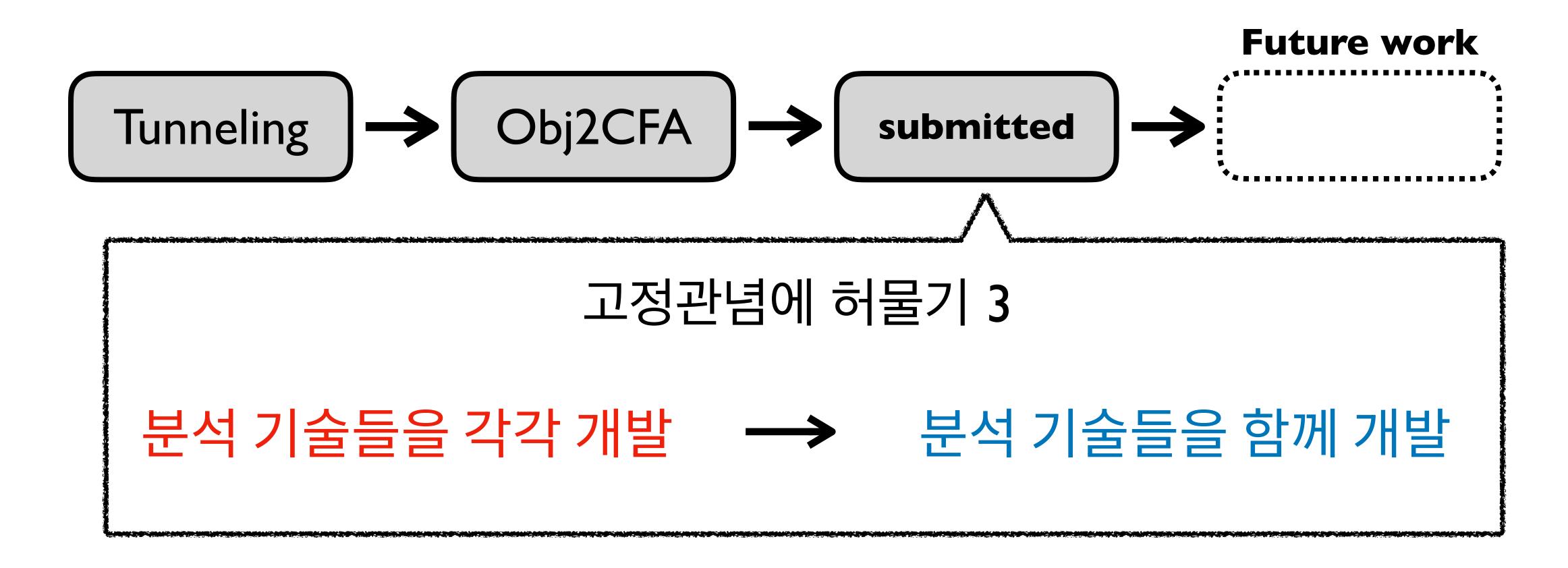


1-object sensitivity

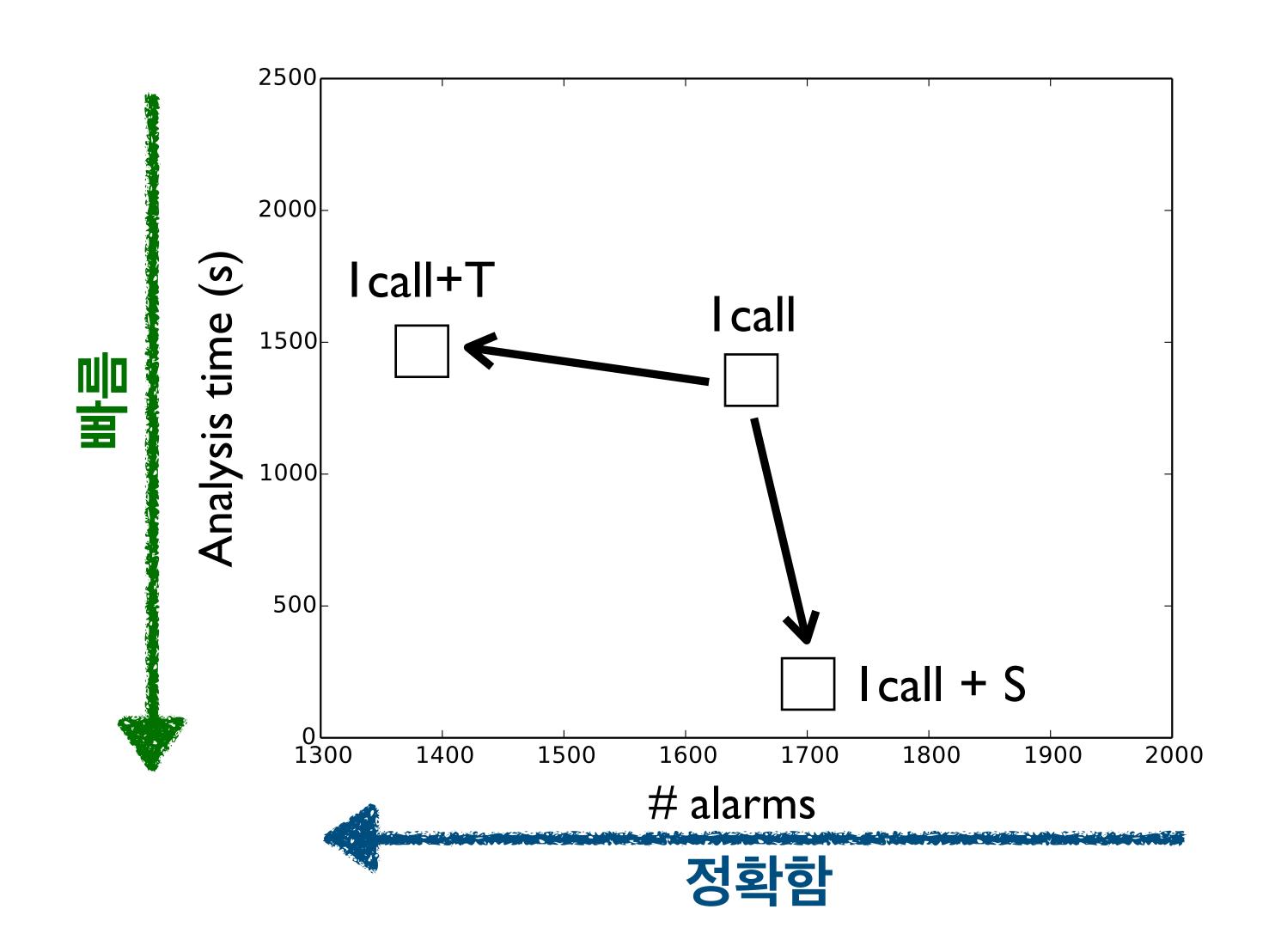
Tian Tan @ Nanjing University

고정관념에 도전하기

• 목표: 주요 k기반 요약 방식을 표준으로 만들기

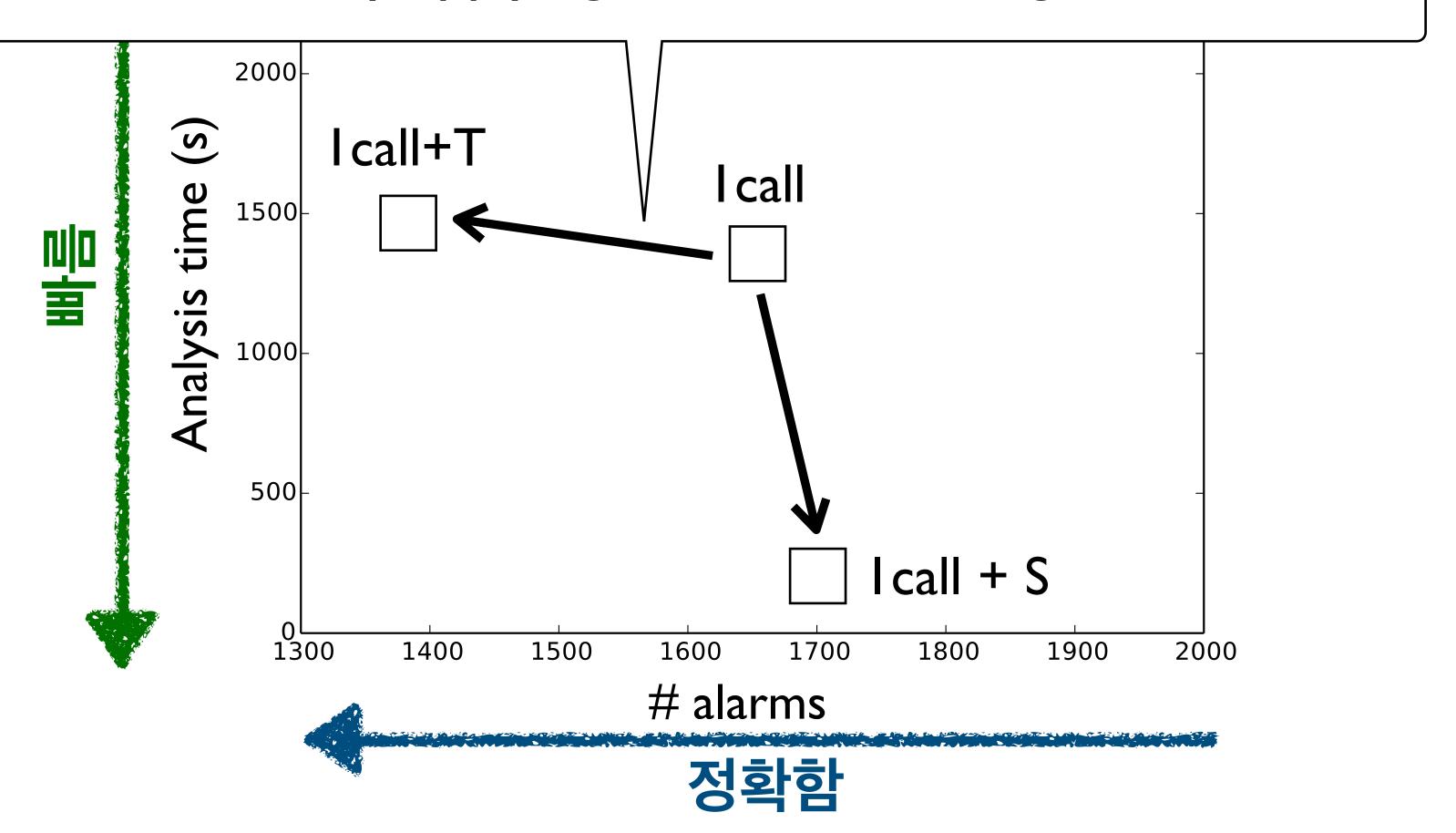


- 호출 환경 배달하기: 속도를 유지한채 정확도를 올려주는 기술
- 적당히 함수 호출 분석 (selective ctx sensitivity): 정확도를 유지한채 속도를 올려주는 기술

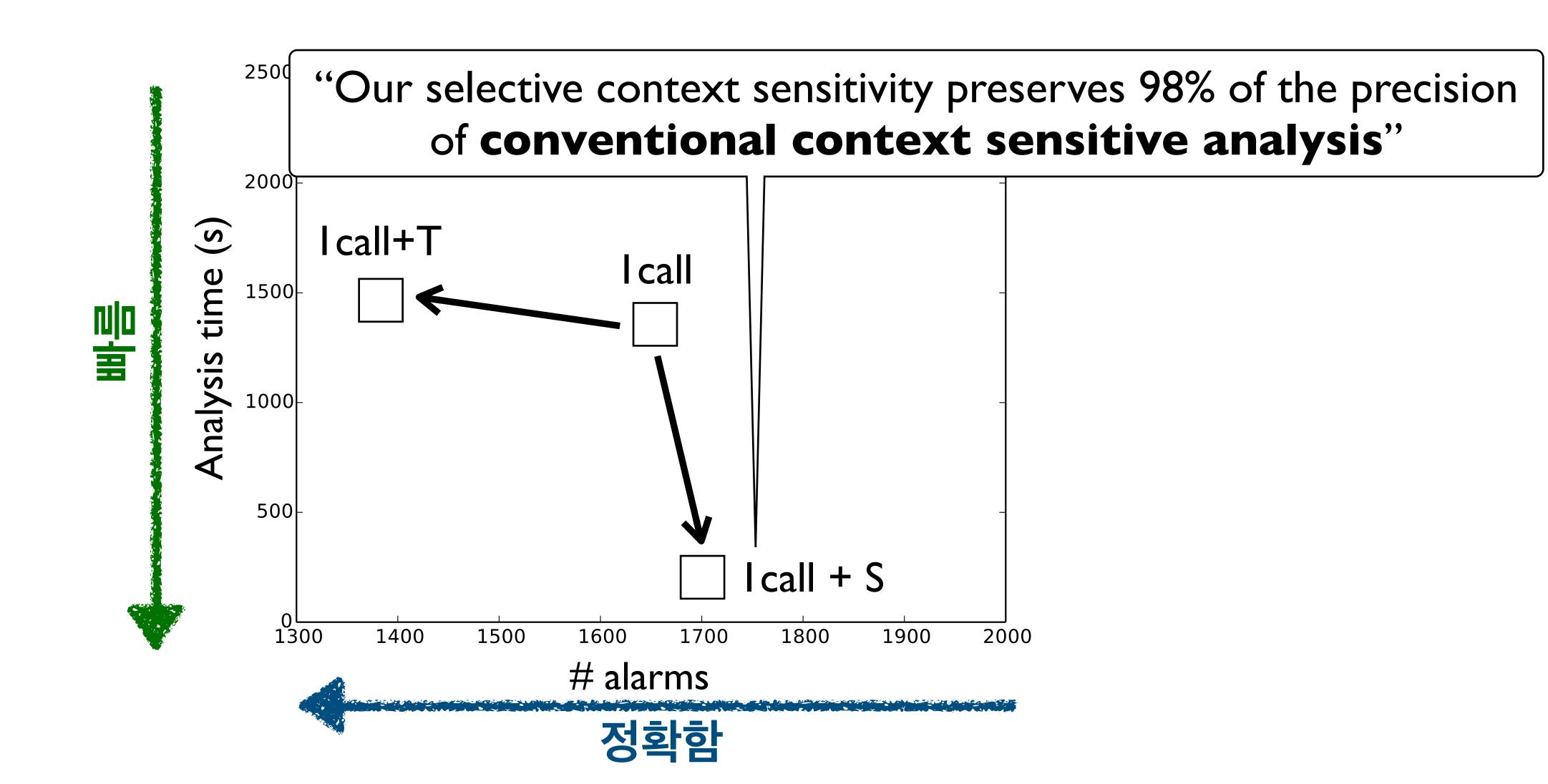


• 호출 환경 배달하기: 속도를 유지한채 정확도를 올려주는 기술

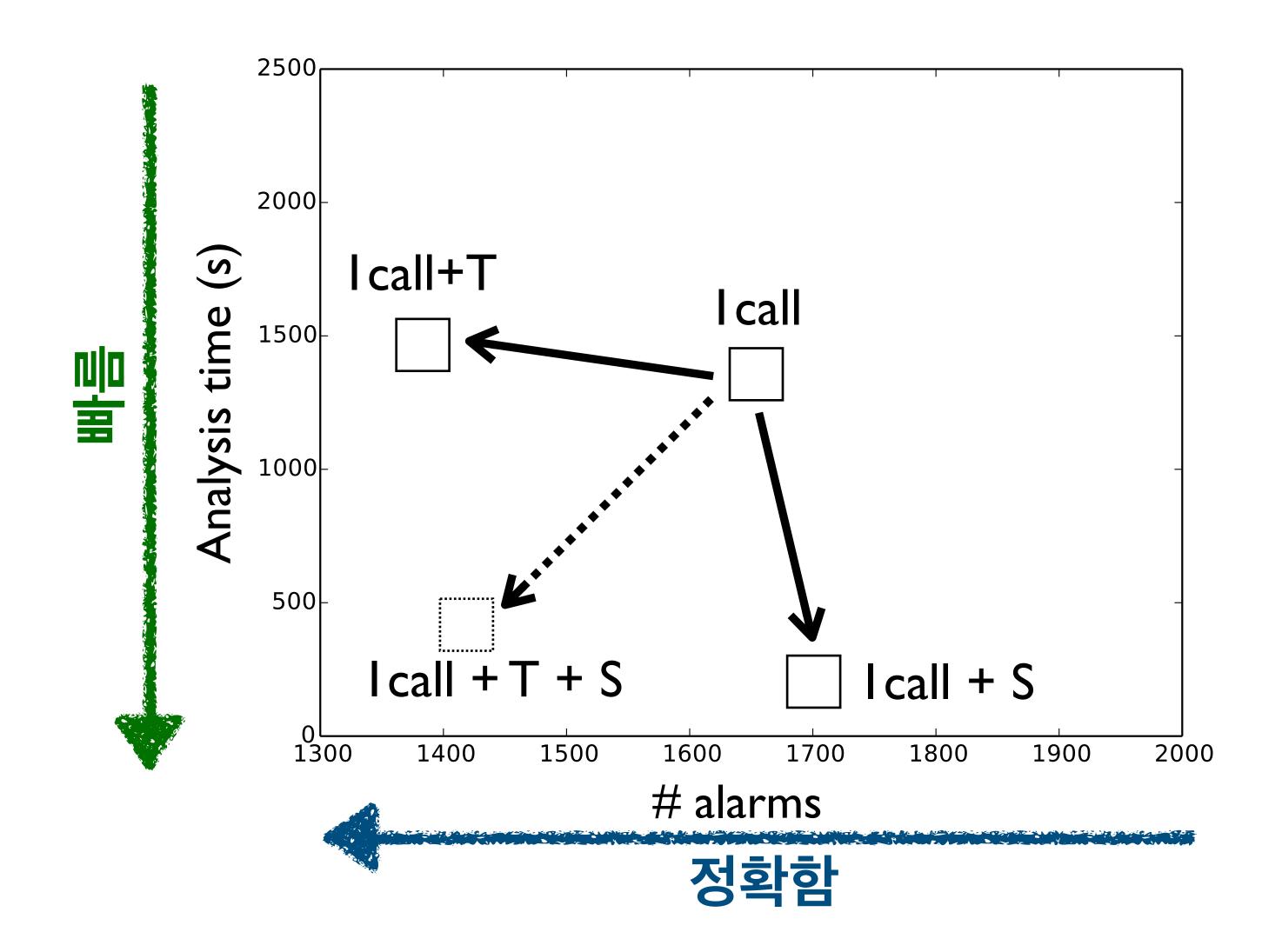
"We generated I call+T by applying context tunneling to I call..."



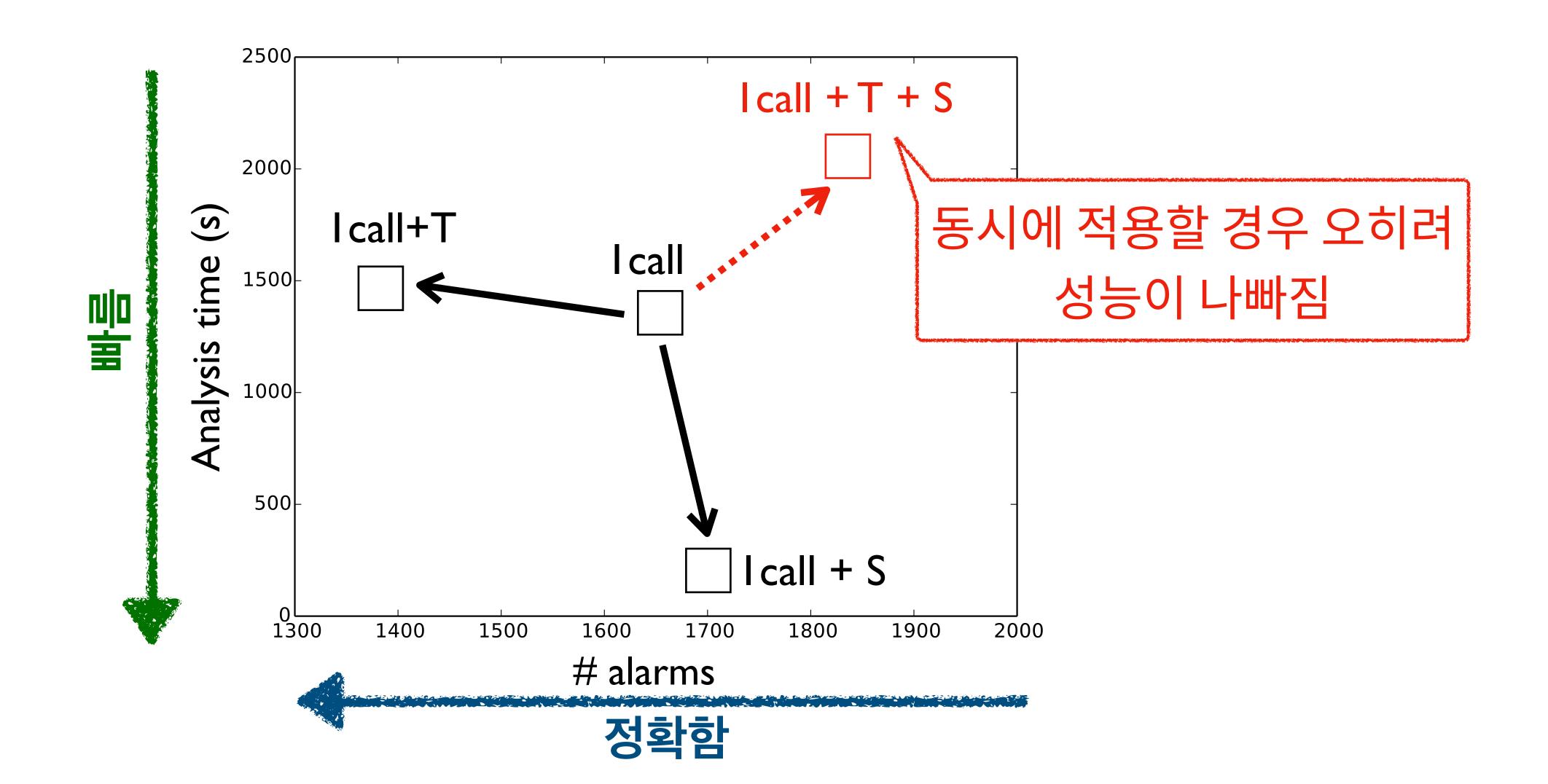
• 적당히 함수 호출 분석: 정확도를 유지한채 속도를 올려주는 기술



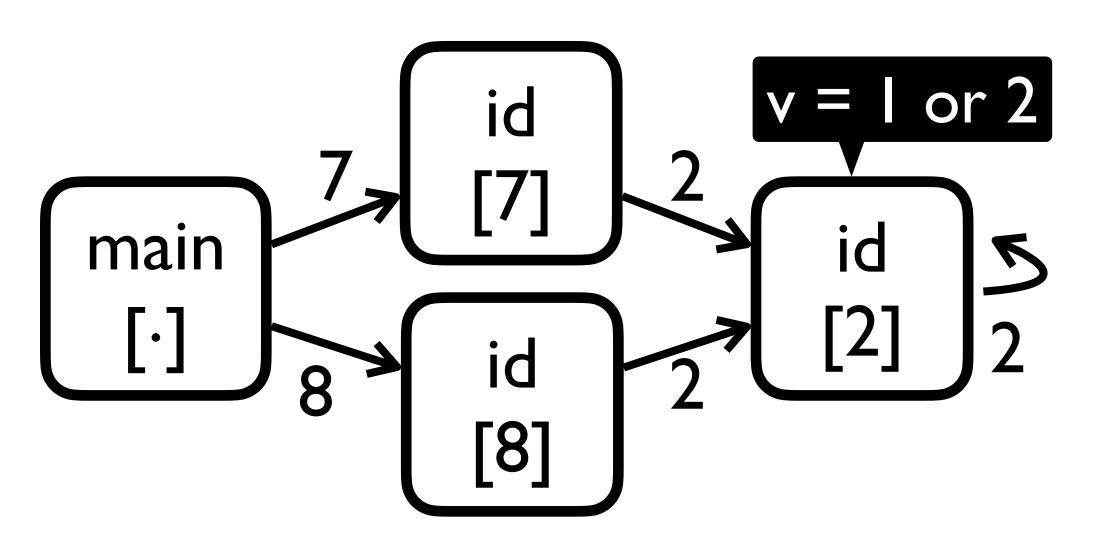
● 질문: 독립적으로 각각 개발한 기술들을 동시에 사용하면 궁극의 성능이 나올까?



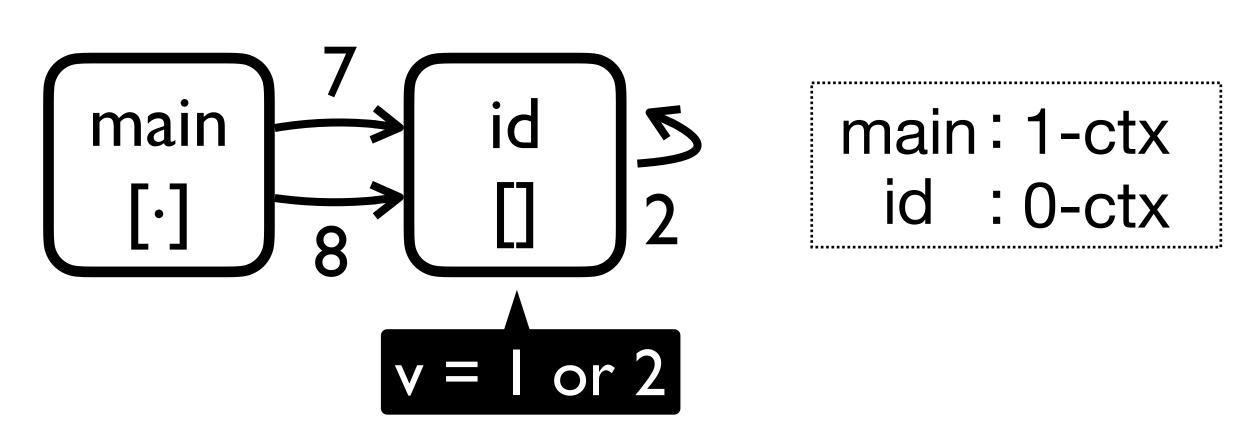
● 질문: 독립적으로 각각 개발한 기술들을 동시에 사용하면 궁극의 성능이 나올까?



```
0: id(v, i){
   if (i > 0){
    return id(v, i-1);}
    return v;}
4:
5: main(){
6: i = input();
7: vI = id(I, i); //A
   v2 = id(2, i);//B
    assert (vI != v2);//query
10: }
    예제 프로그램
```

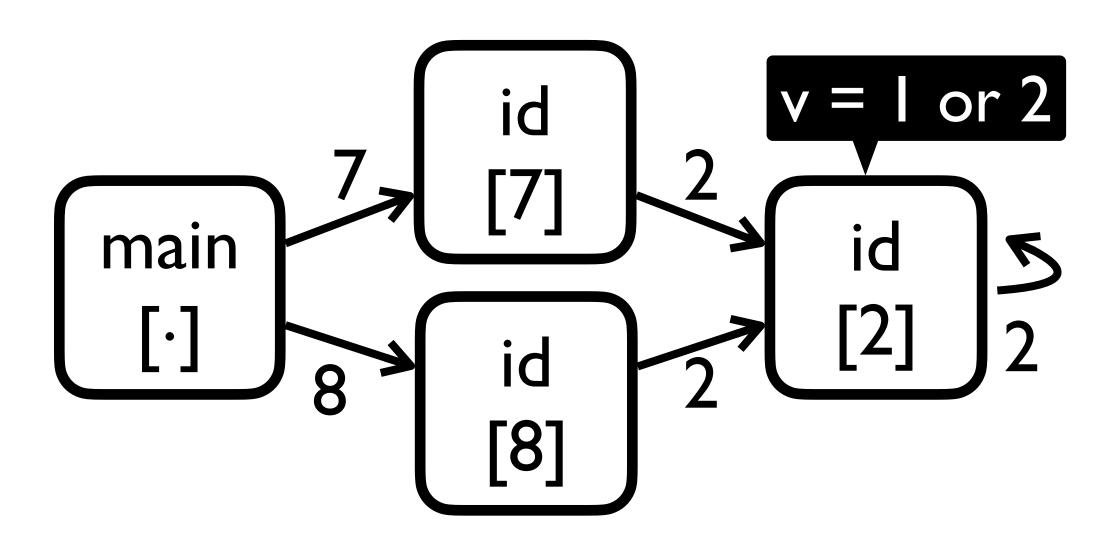


I-요소 기반 함수 호출 요약

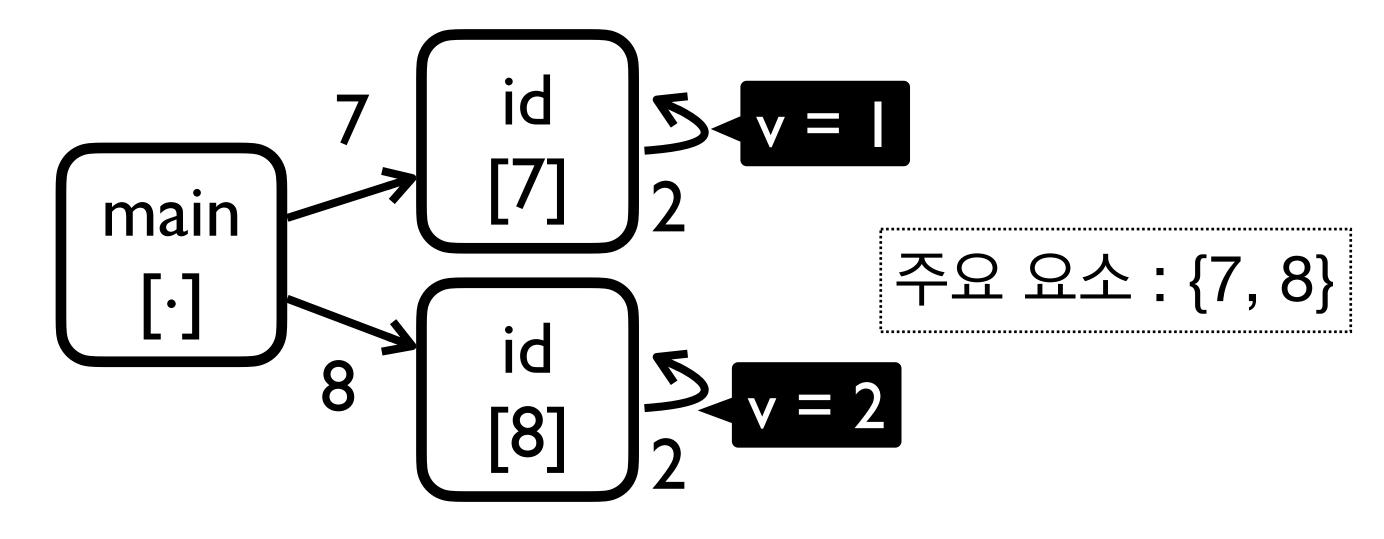


적당히 I-요소 기반 함수 호출 요약

```
0: id(v, i){
  if (i > 0){
    return id(v, i-1);}
    return v;}
4:
5: main(){
6: i = input();
7: vI = id(I, i);//A
   v2 = id(2, i);//B
    assert (vI != v2);//query
10: }
    예제 프로그램
```

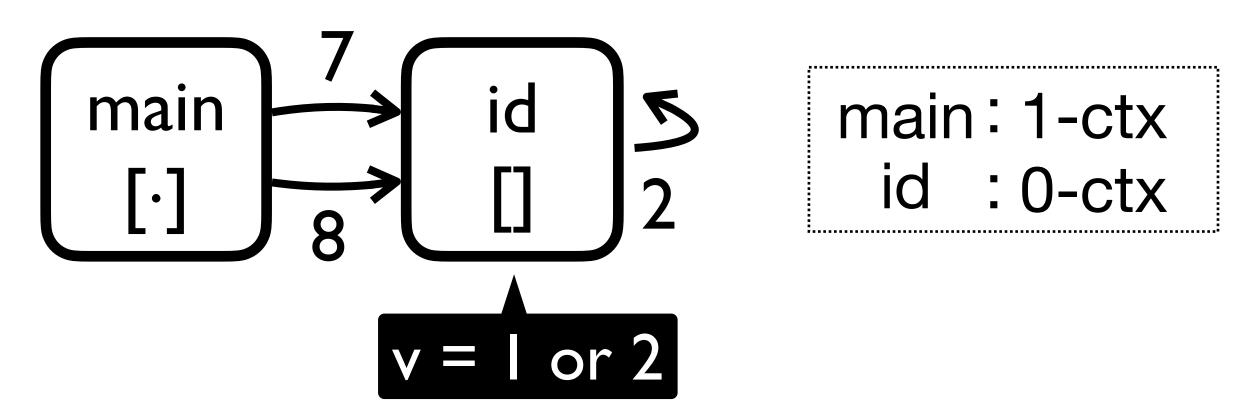


1-요소 기반 함수 호출 요약

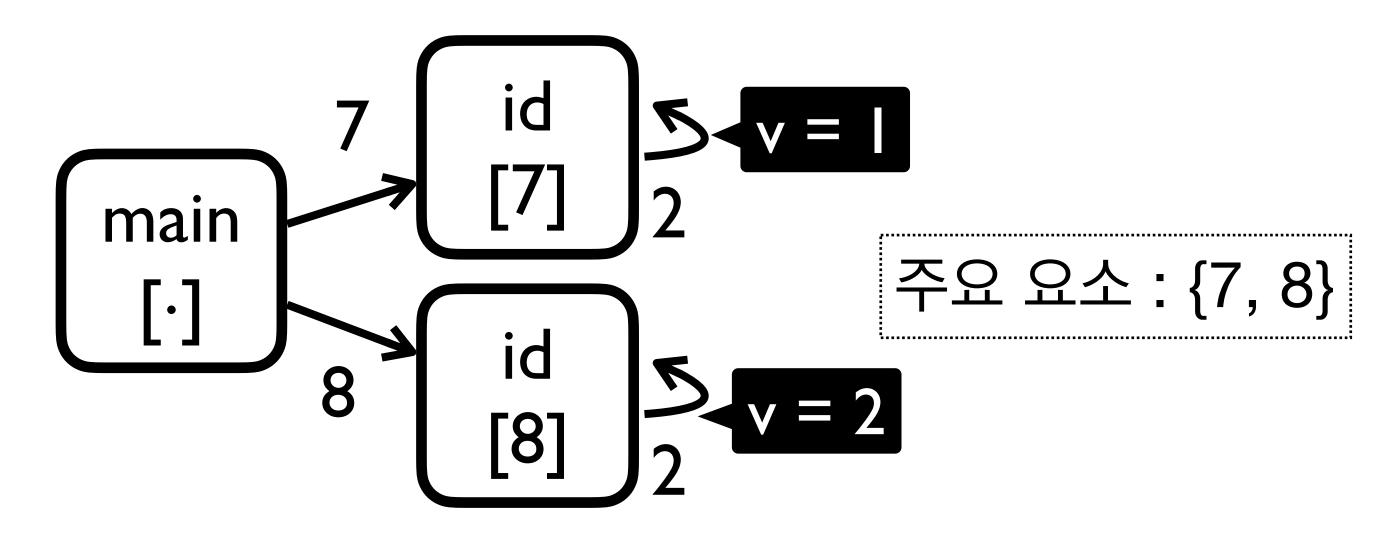


주요 I-요소 기반 함수 호출 요약

```
0: id(v, i){
   if (i > 0){
    return id(v, i-1);}
    return v;}
4:
5: main(){
6: i = input();
   vI = id(I, i);//A
   v2 = id(2, i);//B
    assert (vI != v2);//query
10: }
    예제 프로그램
```

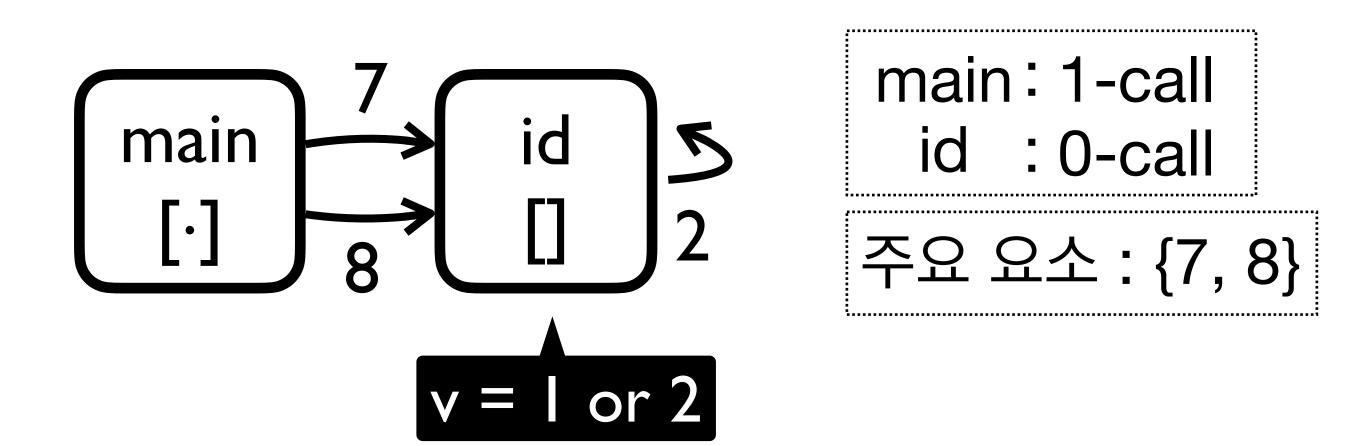


적당히 I-요소 기반 함수 호출 요약



주요 1-요소 기반 함수 호출 요약

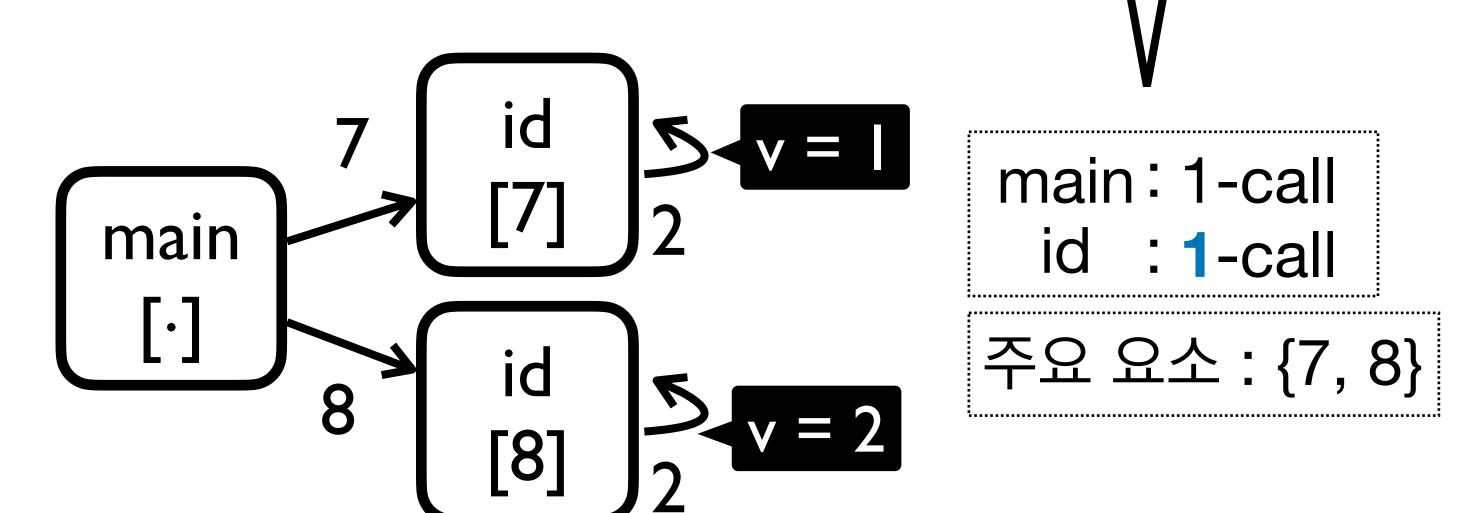
```
0: id(v, i){
   if (i > 0){
    return id(v, i-1);}
    return v;}
4:
5: main(){
6: i = input();
7: vI = id(I, i);//A
   v2 = id(2, i);//B
   assert (vI != v2);//query
9:
10: }
    예제 프로그램
```



적당히 주요 I-요소 기반 함수 호출 요약

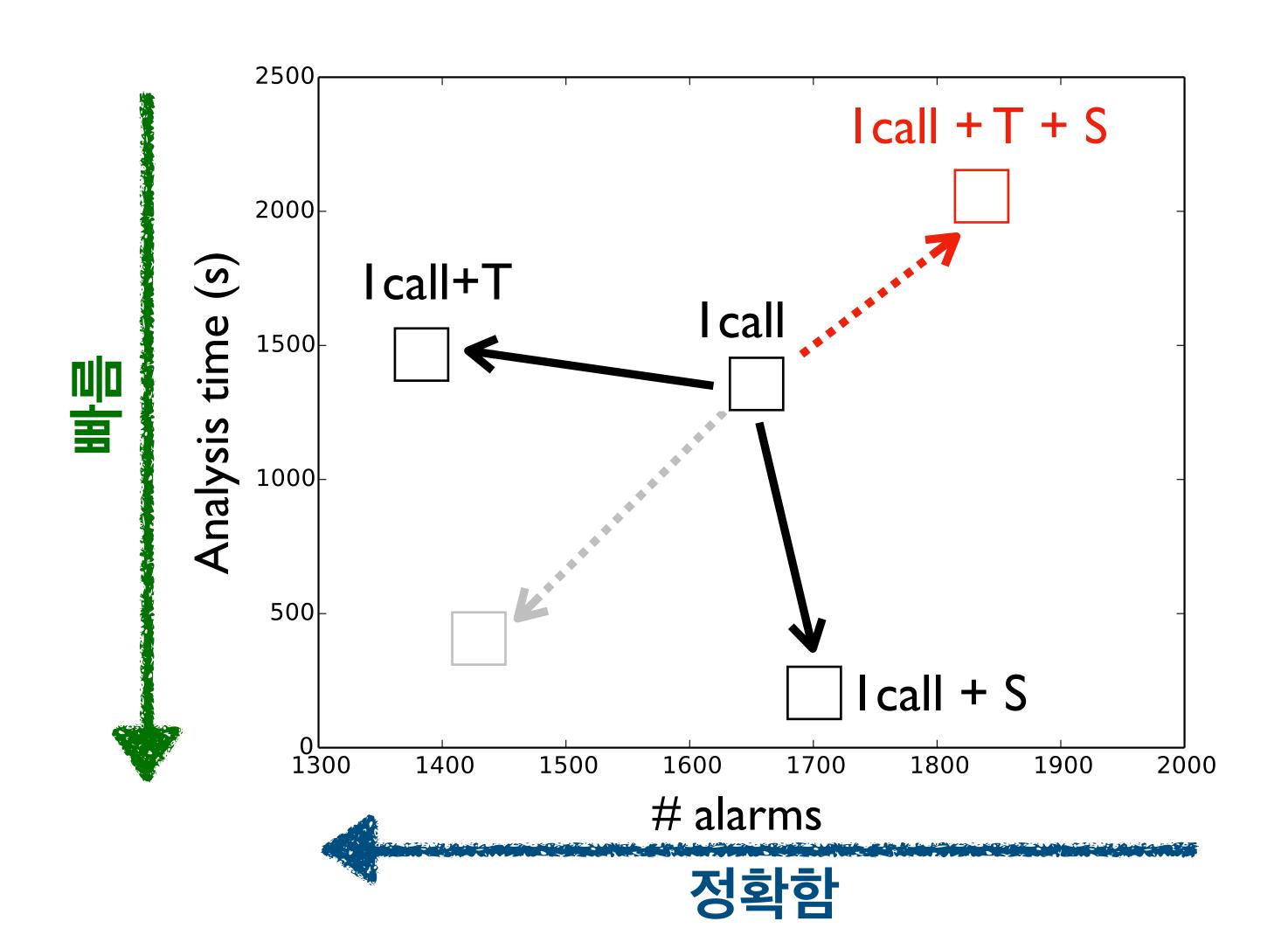
```
0: id(v, i){
   if (i > 0){
     return id(v, i-1);}
    return v;}
4:
5: main(){
6: i = input();
   vI = id(I, i);//A
   v2 = id(2, i);//B
    assert (vl != v2);//query
10: }
    예제 프로그램
```

같이 사용될 것을 고려해 분류해야 함

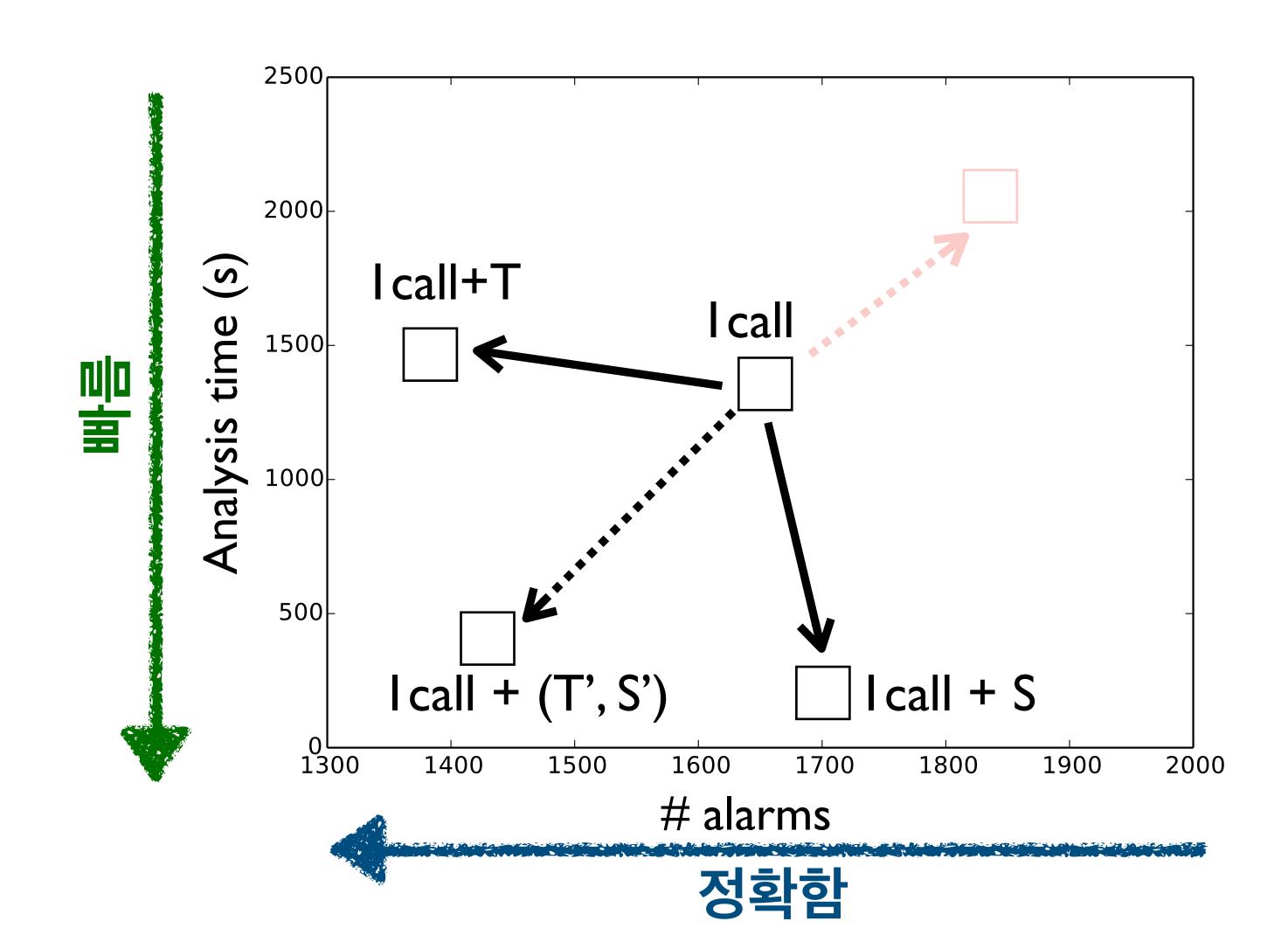


적당히 주요 I-요소 기반 함수 호출 요약

● 연구 트렌드를 따라 독립적으로 각각 개발한 기술들을 동시에 사용하면 성능이 오히려 나빠짐



• 궁극의 성능을 내기 위해선 서로가 미칠 영향을 고려해 동시에 만들어야 함



• 궁극의 성능을 내기 위해선 서로가 미칠 영향을 고려해 동시에 만들어야 함

Marriage of Context Tunneling and Selective Context Sensitivity in Pointer Analysis

ANONYMOUS AUTHOR(S)

In this paper, we identify a fundamental issue in the current trend of developing context sensitivity techniques in pointer analysis and present a way to efficiently address it. Context sensitivity is a key factor that significantly affects the performance of pointer analysis in object-oriented programs. In the literature, two major refinements—context tunneling and selective context sensitivity—have been developed, where context tunneling improves precision and selective context sensitivity enhances scalability. Though the two techniques can be used together to maximize both precision and scalability, they have been developed independently without considering whether individually optimized techniques will remain effective when combined. In this work, however, we demonstrate that combining independently developed context tunneling and selective context sensitivity techniques leads to suboptimal performance. To be an effective combination, the two techniques must be developed together, considering their interdependencies. Developing a pair of techniques, however, while accounting for all possible interactions is extremely challenging. To address this challenge, we present a framework that significantly reduces the complexity of developing an effective combination of the two techniques. Our evaluation results show that following our approach leads to the development of an effective combination, achieving a state-of-the-art performance, that outperforms combinations of independently developed context tunneling and selective context sensitivity techniques.

ACM Reference Format:

1 INTRODUCTION

Context sensitivity plays a pivotal role in pointer analysis of object-oriented programs. It enhances precision by distinguishing between multiple invocations of the same method based on their calling contexts. However, tracking every possible context is impractical, leading to the widespread use of k-limited context sensitivity. This approach retains only the k most recent context elements—typically call sites in call-site sensitivity [Sharir and Pnueli 1981] or allocation sites in object sensitivity [Milanova et al. 2002]. Despite its adoption, this conventional technique frequently falls short in balancing precision and scalability in real-world applications.

Over the past decade, numerous techniques have been proposed to enhance the *k*-limited approach in context-sensitive pointer analysis [He et al. 2024; Jeon et al. 2018; Jeon and Oh 2022; Kastrinis and Smaragdakis 2013; Li et al. 2018a,b, 2020; Liang et al. 2011; Lu et al. 2021a,b; Milanova et al. 2002; Oh et al. 2015; Smaragdakis et al. 2011, 2014; Tan et al. 2021, 2017; Zhang et al. 2014]. Two prominent approaches that excel in maximizing precision or scalability are:

• Context tunneling [Jeon et al. 2018; Jeon and Oh 2022] seeks to maximize precision while adhering to a k-context limit. Instead of relying solely on the k most recent context elements, it adopts a more flexible strategy by prioritizing the k most significant context elements. Jeon and Oh [2022] demonstrated that context tunneling can markedly improve analysis

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2018 Association for Computing Machinery.

0004-5411/2018/8-ART111 \$15.00

https://doi.org/XXXXXXXXXXXXXXX

분석 기술들을 각각 개발

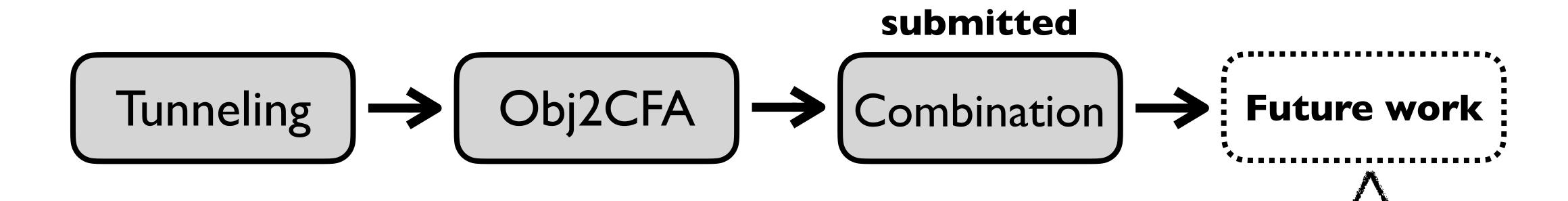
• We identify a fundamental issue in the current trend of developing context sensitivity techniques in pointer analysis and present a way to efficiently address it.

분석 기술들을 동시에 개발하는 법

• We present a framework that significantly reduces the complexity of developing an effective combination of the two techniques

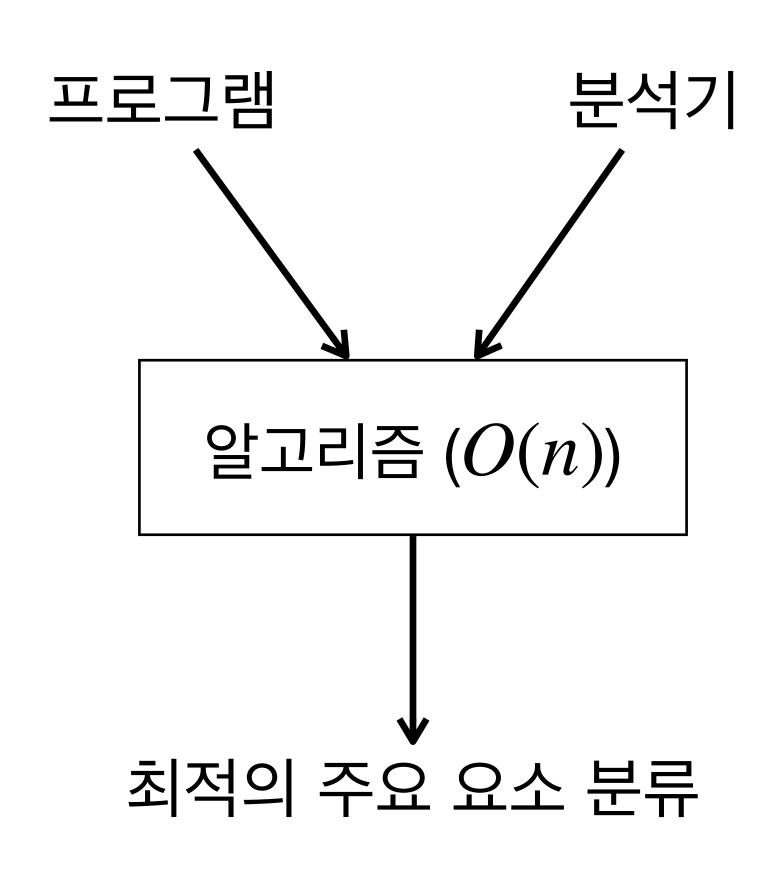
컨텍스트 터널링: 고정관념에 도전하기

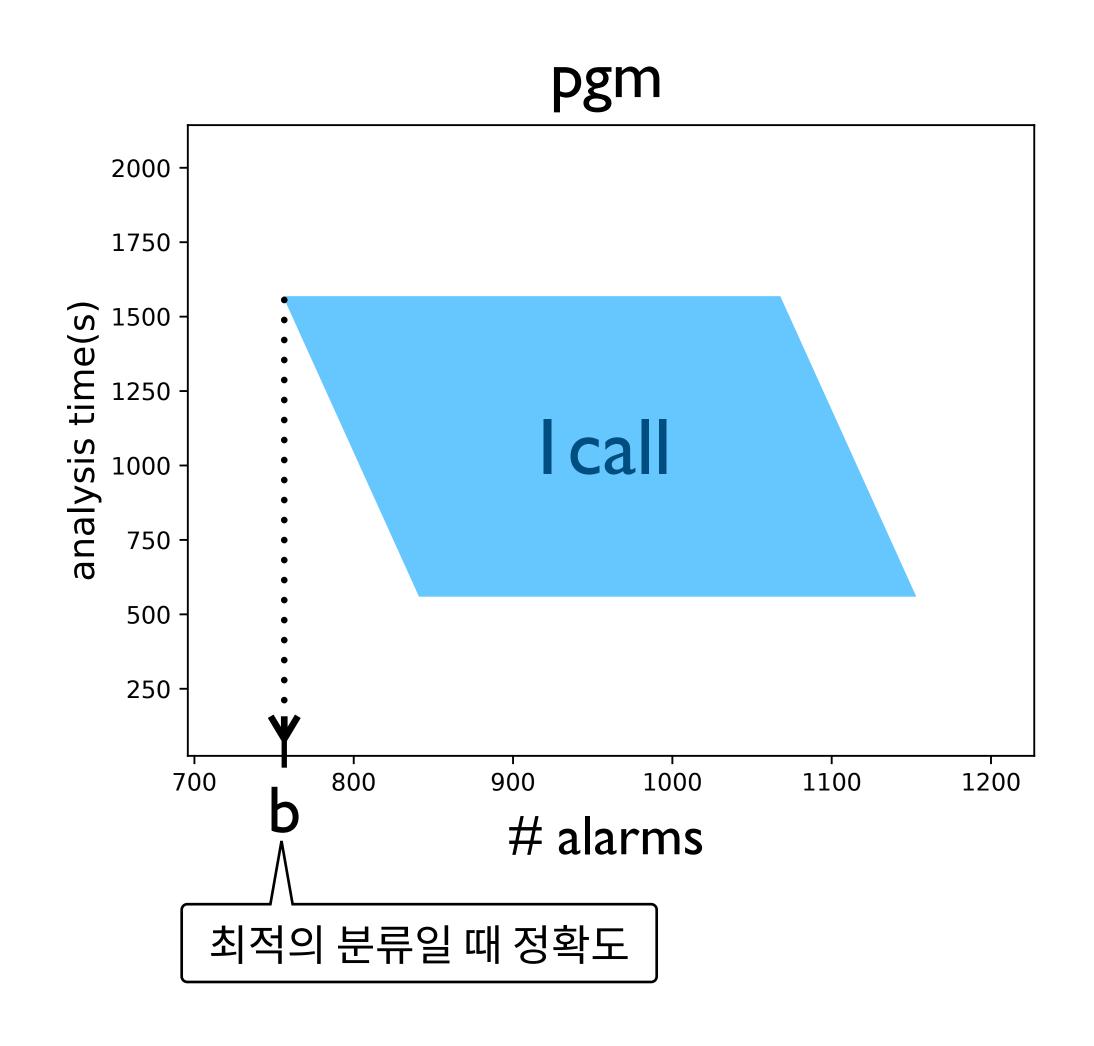
• 목표: 주요 k기반 요약 방식을 표준으로 만들기



- 효율적으로 최적의 주요 요소 분류를 찾아주는 알고리즘
- 쉽고 간단한 주요 요소 분류방법
- CFA의 Obj 대비 이론적 우수성 보이기
- 타 언어에서 주요 k기반 요약 방식의 효과성 보이기

후속 연구 1: 최적의 주요 요소 분류 찾기 알고리즘





후속 연구 2: 쉽고 간단한 주요 요소 분류 방법

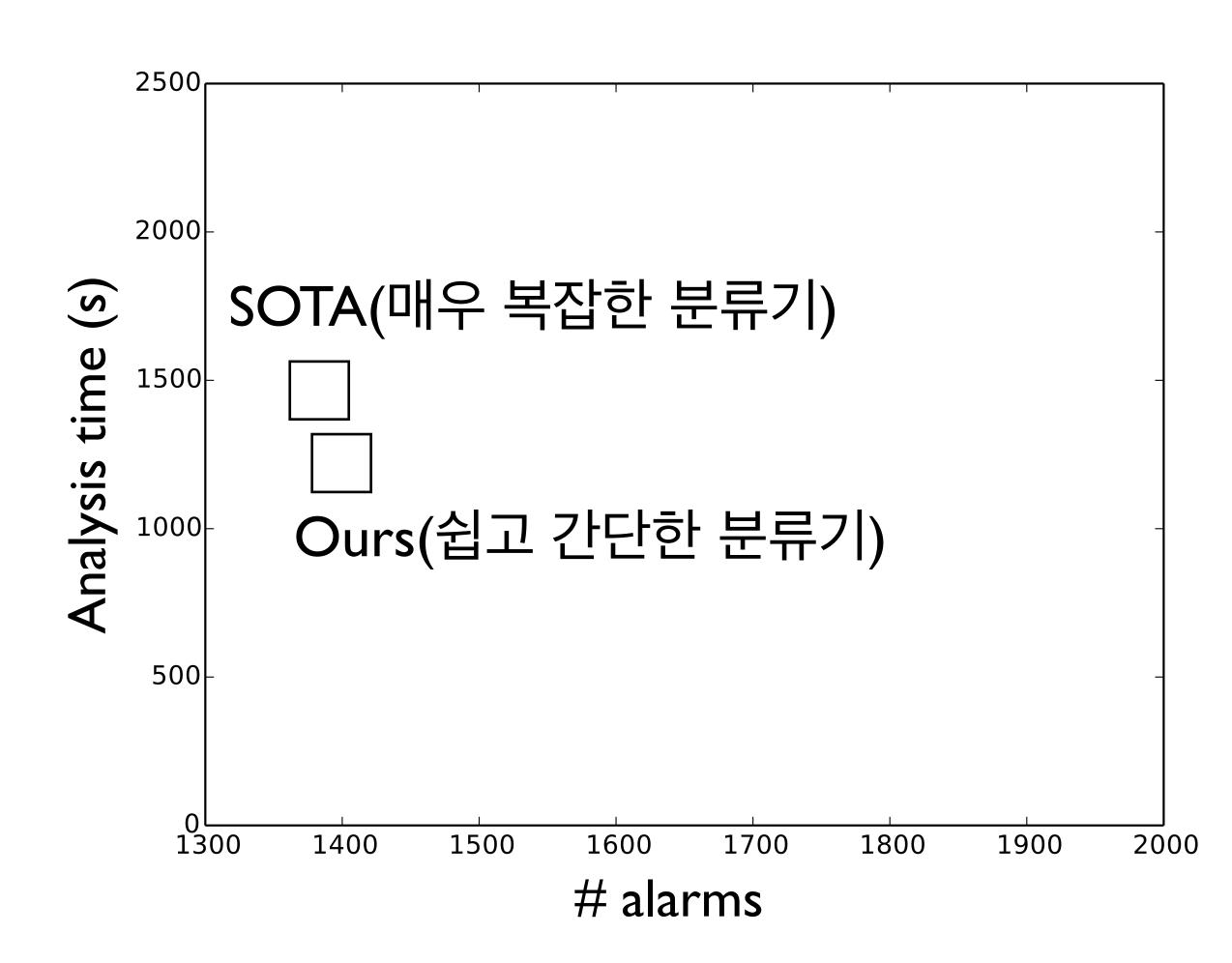
classifier(e):

if (<u>??</u> (쉽고 간단한 조건)):

return true //주요 요소임

else:

return false //부수적 요소임



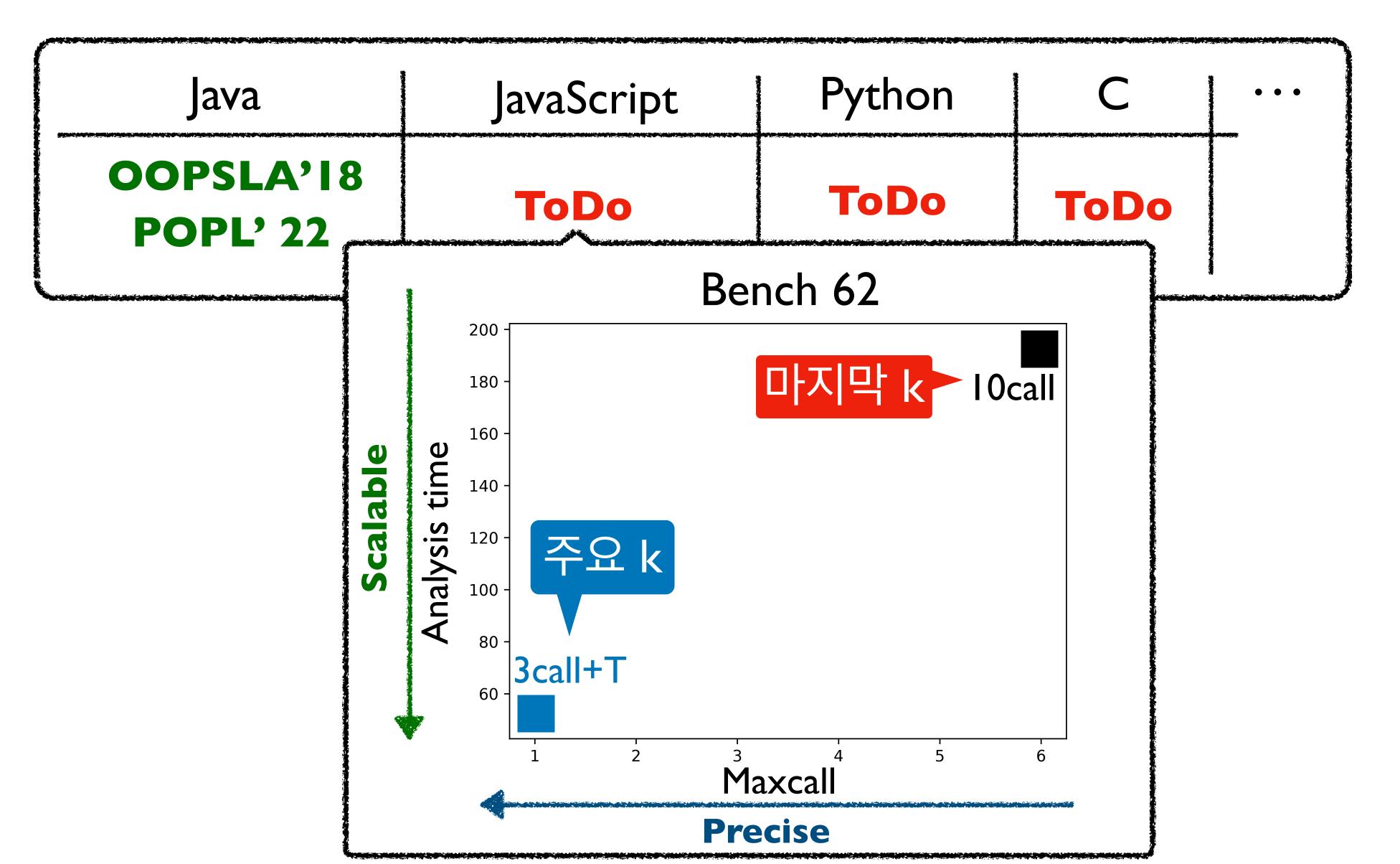
후속 연구 3: CFA의 Obj 대비 우수성을 이론적으로 보이기

임의의 프로그램과 k-obj가 내놓은 분석 결과에 대하여, 더 정확한 분석 결과를 내놓는 k-CFA가 항상 존재하는가?

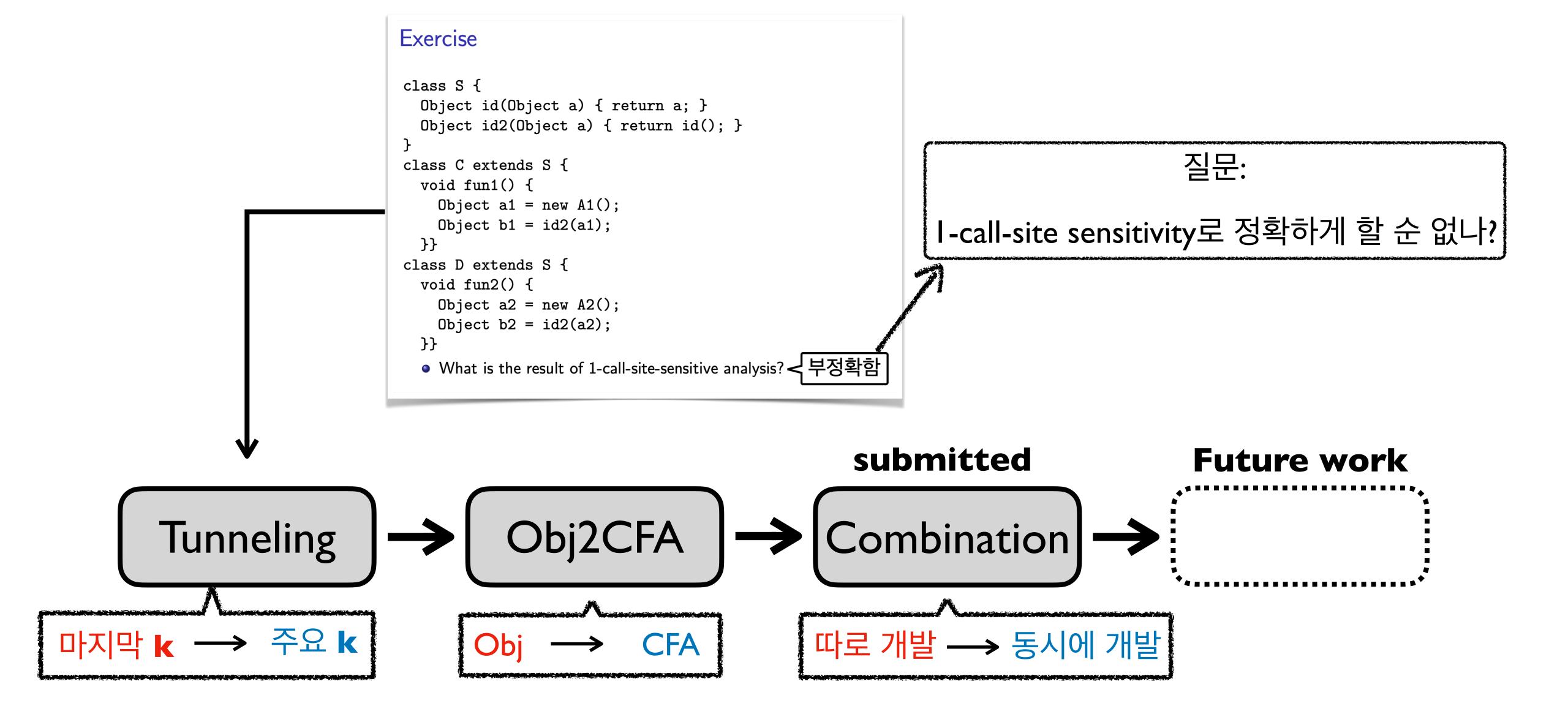
Definition 7.1 (Superiority of Call-Site Sensitivity). Let \mathbb{P} be a set of target programs. Let \mathbb{S} be a context-tunneling space for the target programs. We say call-site sensitivity is superior to object sensitivity with respect to \mathbb{S} if is always possible to simulate object sensitivity via call-site sensitivity:

 $\forall P \in \mathbb{P}. \forall T_{obj} \in \mathbb{S}. \exists T_{call} \in \mathbb{S}. \forall k \in [0, \infty]. \ fixF_{P,k}^{T_{call}, U_{call}} \geq \text{ (more precise than)} \ fixF_{P,k}^{T_{obj}, U_{obj}}$ (5)

후속 연구 4: 타 언어에 적용하기

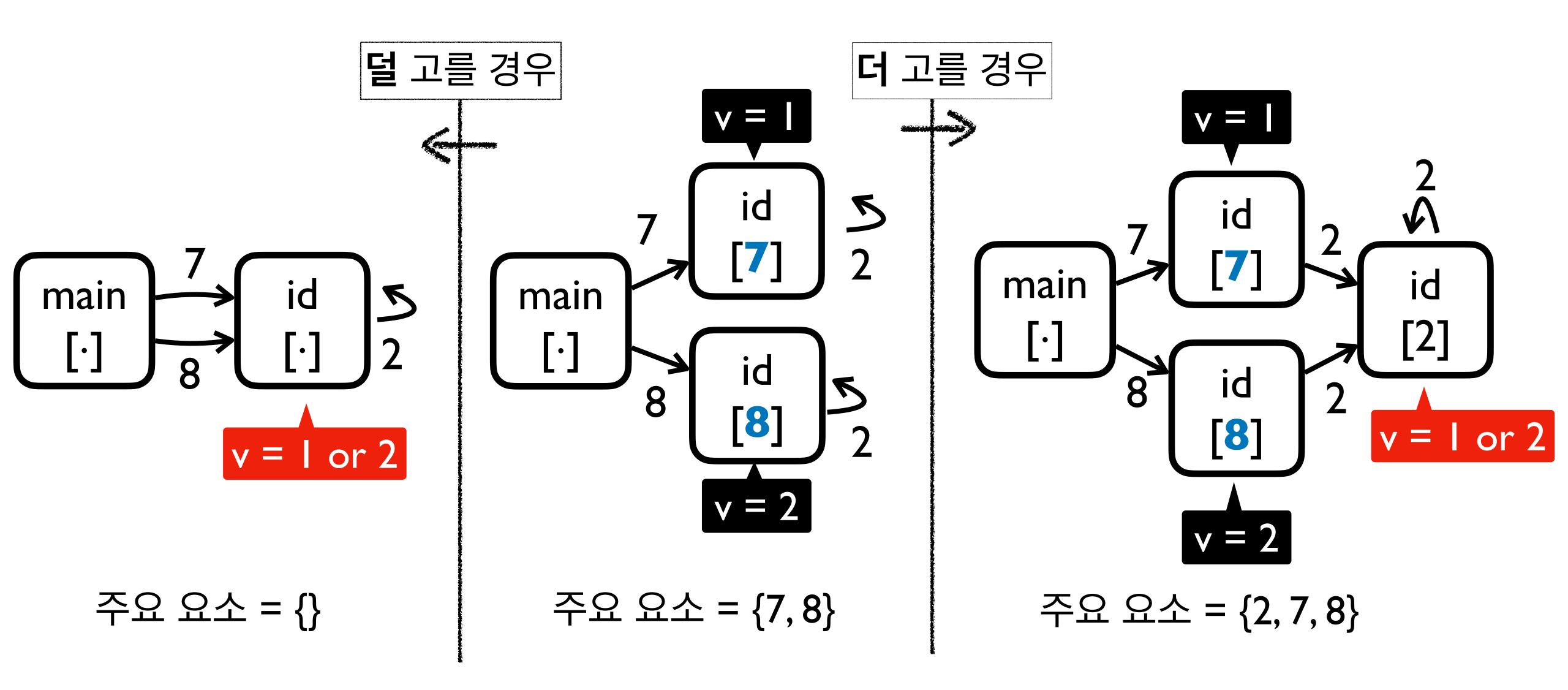


마무리: 고정관념에 도전하기

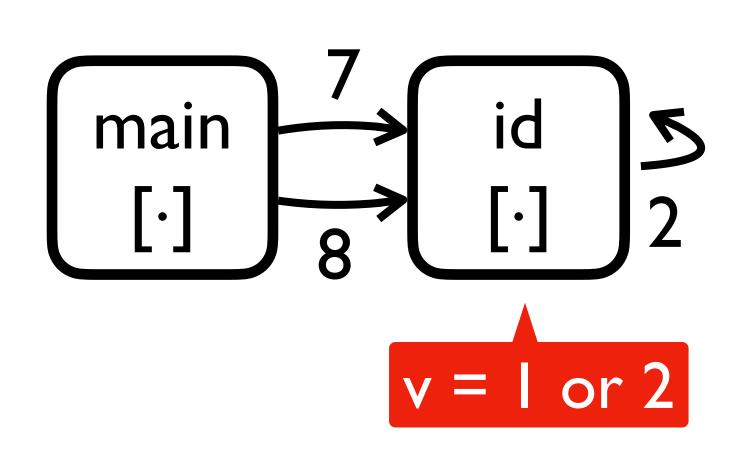


Back up

• 주요 I-요소 기반 함수 호출 요약

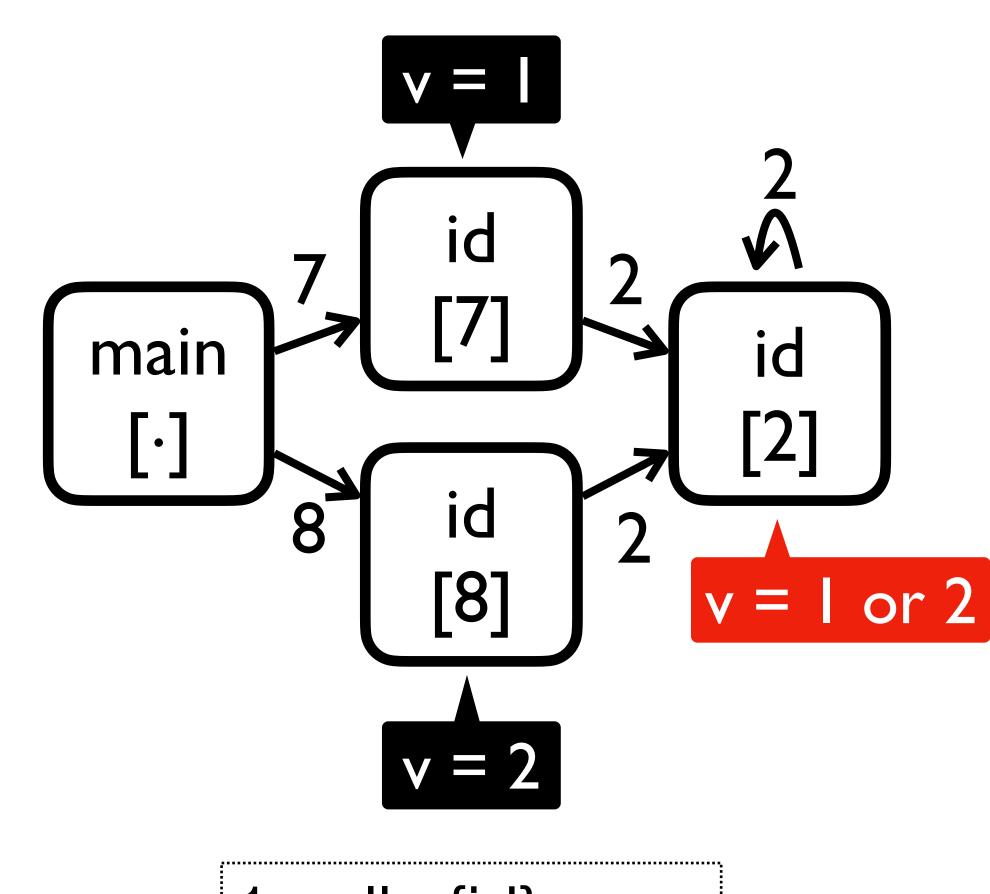


• 선택적 1-요소 기반 함수 호출 요약



1-call : {}

0-call: {main, id}

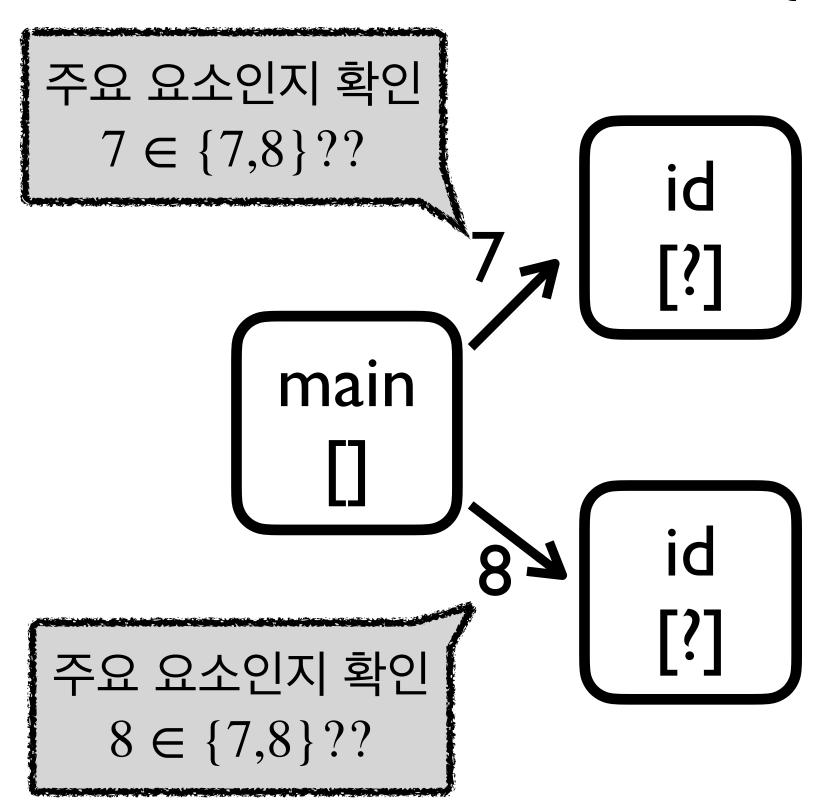


1-call : {id}

0-call: {main}

주요 요소 = {7,8}

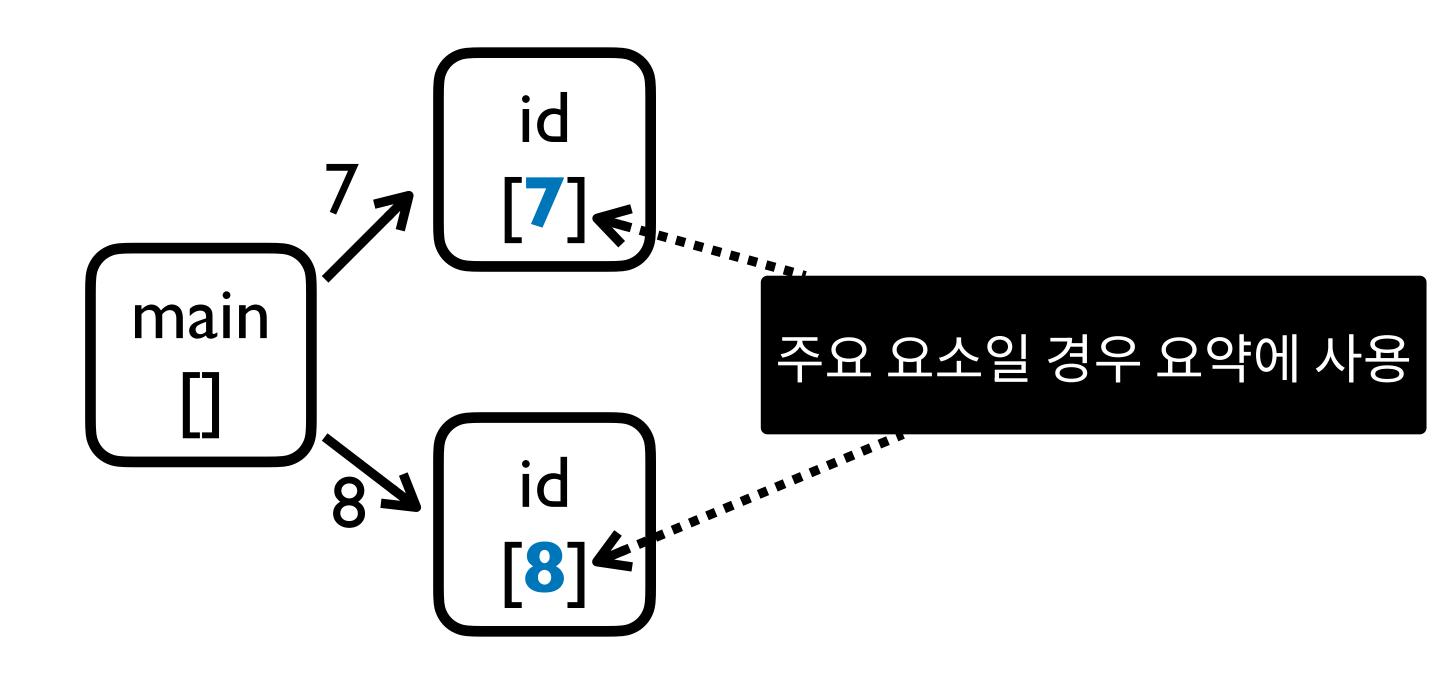
```
0: id(v, i){
I: if (i > 0)
   return id(v, i-1);}
    return v;}
4:
5: main(){
6: i = input();
7: vI = id(I, i);
   v2 = id(2, i);
8:
9: assert (vI != v2);//query
```



예제 프로그램

주요 요소 = {7,8}

```
0: id(v, i){
I: if (i > 0)
    return id(v, i-1);}
   return v;}
5: main(){
6: i = input();
7: vI = id(I, i);
   v2 = id(2, i);
8:
9: assert (vl != v2);//query
```



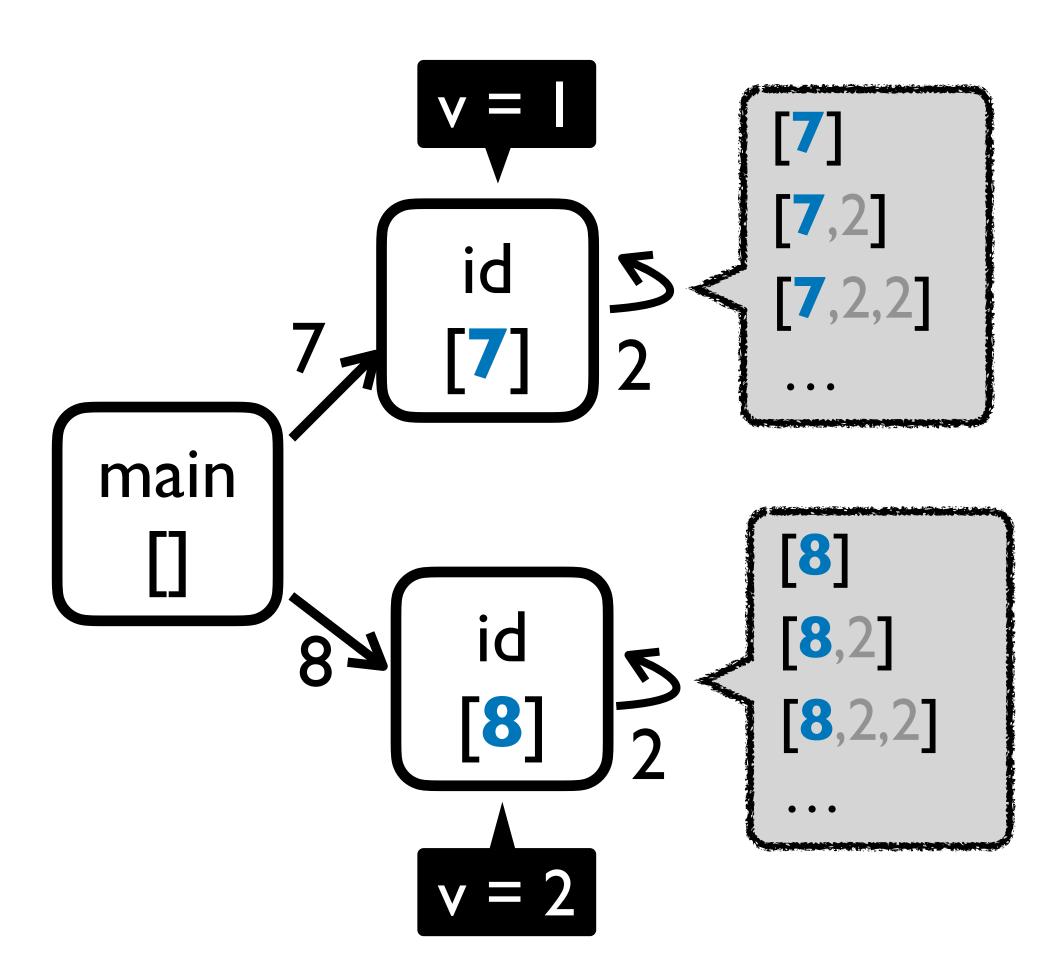
```
0: id(v, i){
I: if (i > 0)
    return id(v, i-1);}
   return v;}
5: main(){
6: i = input();
7: vI = id(I, i);
8: v2 = id(2, i);
9: assert (vI != v2);//query
```

```
주요 요소 = {7,8}
                    주요 요소인지 확인
                      2 \in \{7,8\}??
main
```

```
0: id(v, i){
                                             주요 요소 = {7,8}
1: if (i > 0)
    return id(v, i-1);}
   return v;}
5: main(){
                                 main
                                                                     주요 요소가 아닐 경우
6: i = input();
                                                                         요약에 사용 X
7: vI = id(I, i);
8: v2 = id(2, i);
9: assert (vI != v2);//query
```

예제 프로그램

```
0: id(v, i){
  if (i > 0){
    return id(v, i-1);}
    return v;}
4:
5: main(){
  i = input();
   vI = id(I, i);
   v2 = id(2, i);
8:
    assert (vI != v2);//query
9:
   예제 프로그램
```



I개 주요 요소 기반 함수 호출 요약 (주요 요소 = {7,8})