

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

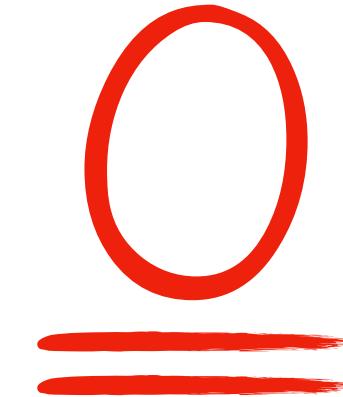
- 아래 문장은 어떤 동화를 한 문장으로 요약한 것이다, 몇점짜리 요약일까?

“The End”

문제

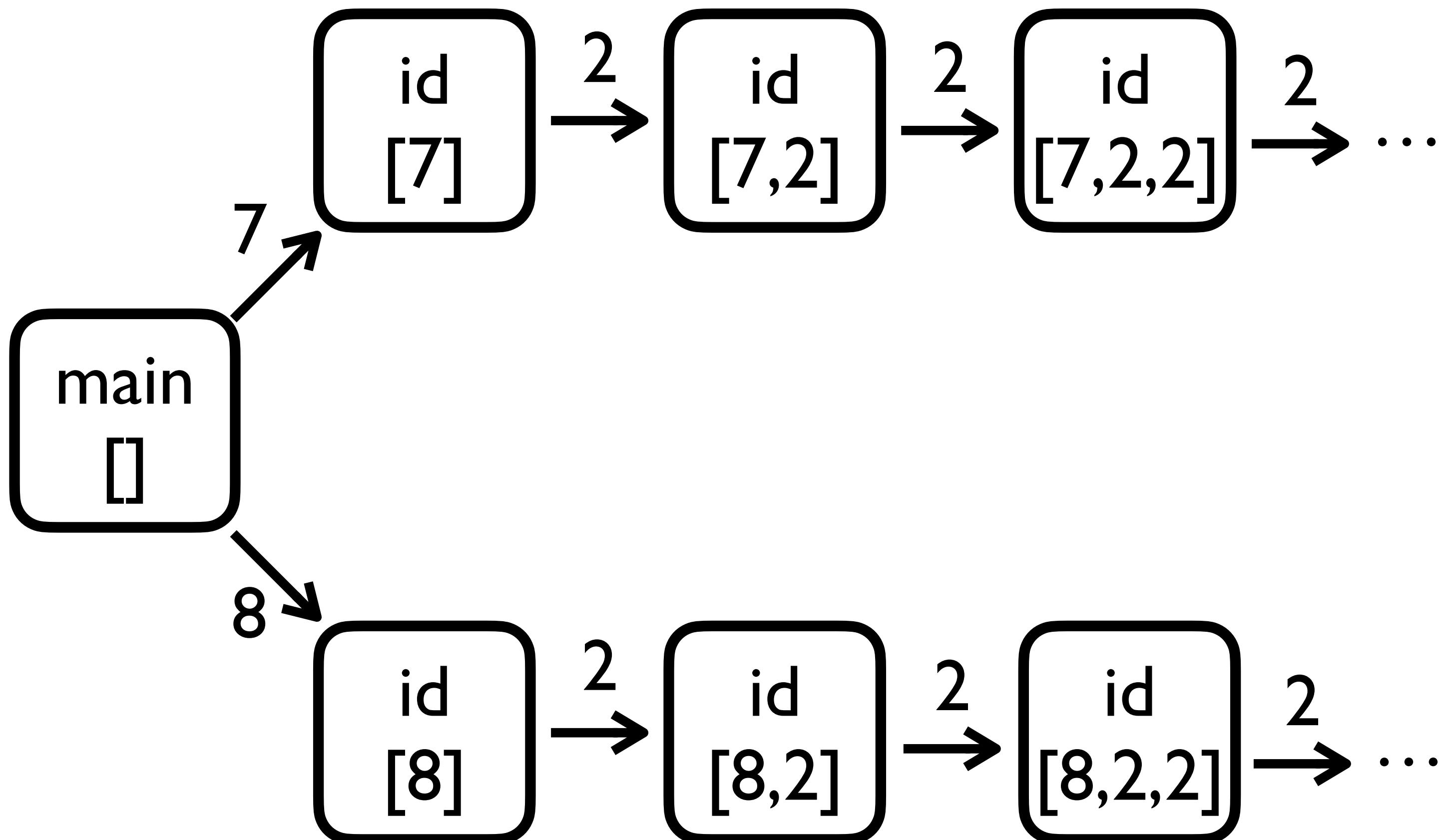
- 아래 문장은 어떤 동화를 한 문장으로 요약한 것이다, 몇점짜리 요약일까?

“The End”



함수 호출 요약의 필요성

```
0: id(v, i){  
1:     if (i > 0){  
2:         return id(v, i-1);}  
3:     return v;}  
4:  
5: main(){  
6:     i = input();  
7:     v1 = id(1, i); //A  
8:     v2 = id(2, i); //B  
9:     assert (v1 != v2); //query  
10: }
```



예제 프로그램

함수 호출 그래프

K개 요소 기반 함수 호출 요약

실제 함수 호출 맥락:

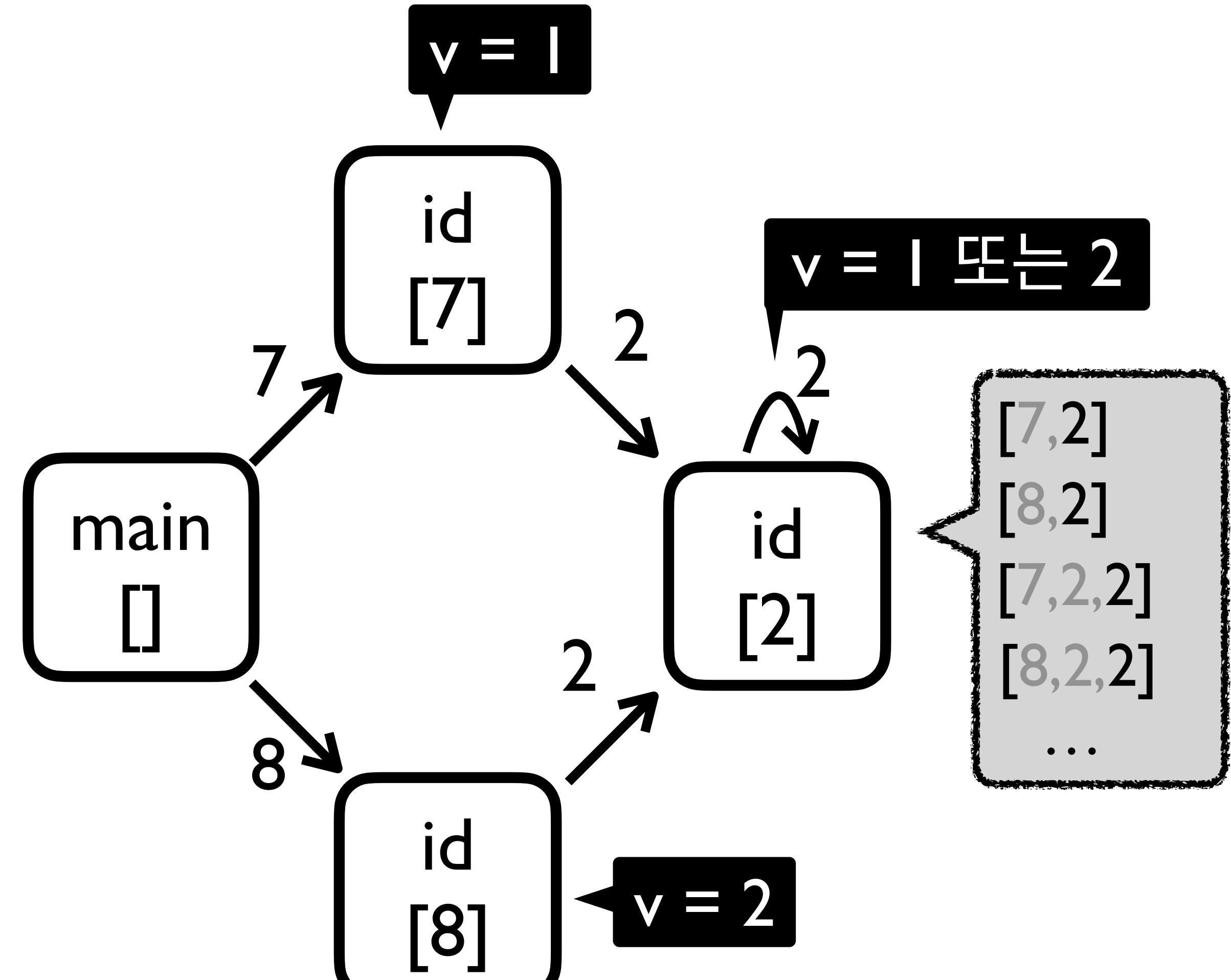


3개 요소 기반 함수 호출 요약
(3-context sensitivity)



K개 요소 기반 함수 호출 요약

```
0: id(v, i){  
1:   if (i > 0){  
2:     return id(v, i-1);}  
3:   return v;}  
4:  
5: main(){  
6:   i = input();  
7:   v1 = id(1, i); //A  
8:   v2 = id(2, i); //B  
9:   assert (v1 != v2); //query  
10: }
```



1개 요소 기반 함수 호출 요약

예제 프로그램

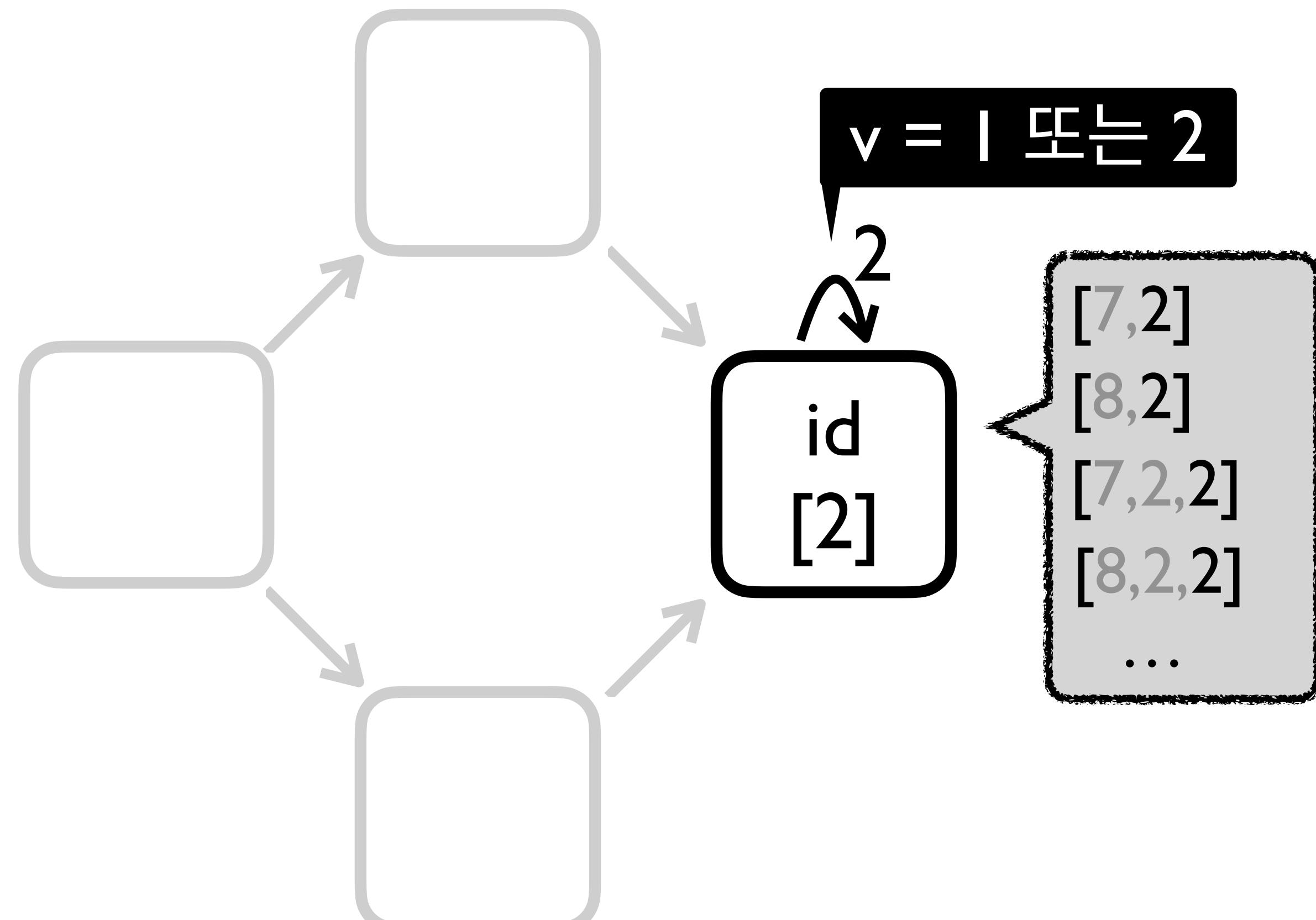
K개 요소 기반 함수 호출 요약

```
0: id(v, i){  
1:   if (i > 0){  
2:     return id(v, i-1);}  
3:   return v; }  
4:  
5: main(){  
6:   i = input();  
7:   v1 = id(1, i);  
8:   v2 = id(2, i);  
9:   assert (v1 != v2); //query  
10: }
```

v = 1 또는 2

v1 = 1 또는 2

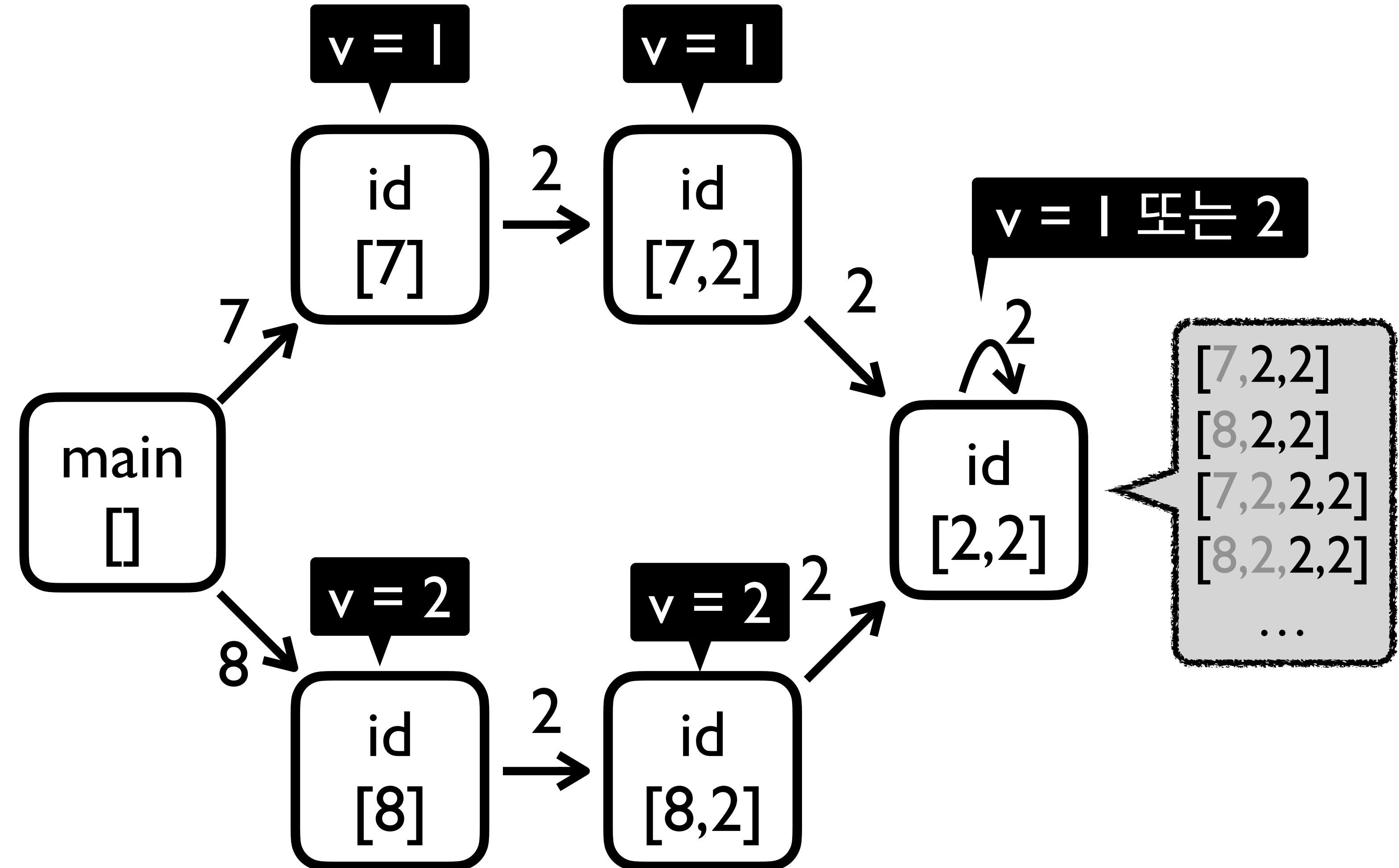
v2 = 1 또는 2



예제 프로그램

K개 요소 기반 함수 호출 요약

```
0: id(v, i){  
1:     if (i > 0){  
2:         return id(v, i-1);}  
3:     return v;}  
4:  
5: main(){  
6:     i = input();  
7:     v1 = id(1, i); //A  
8:     v2 = id(2, i); //B  
9:     assert (v1 != v2); //query  
10: }
```

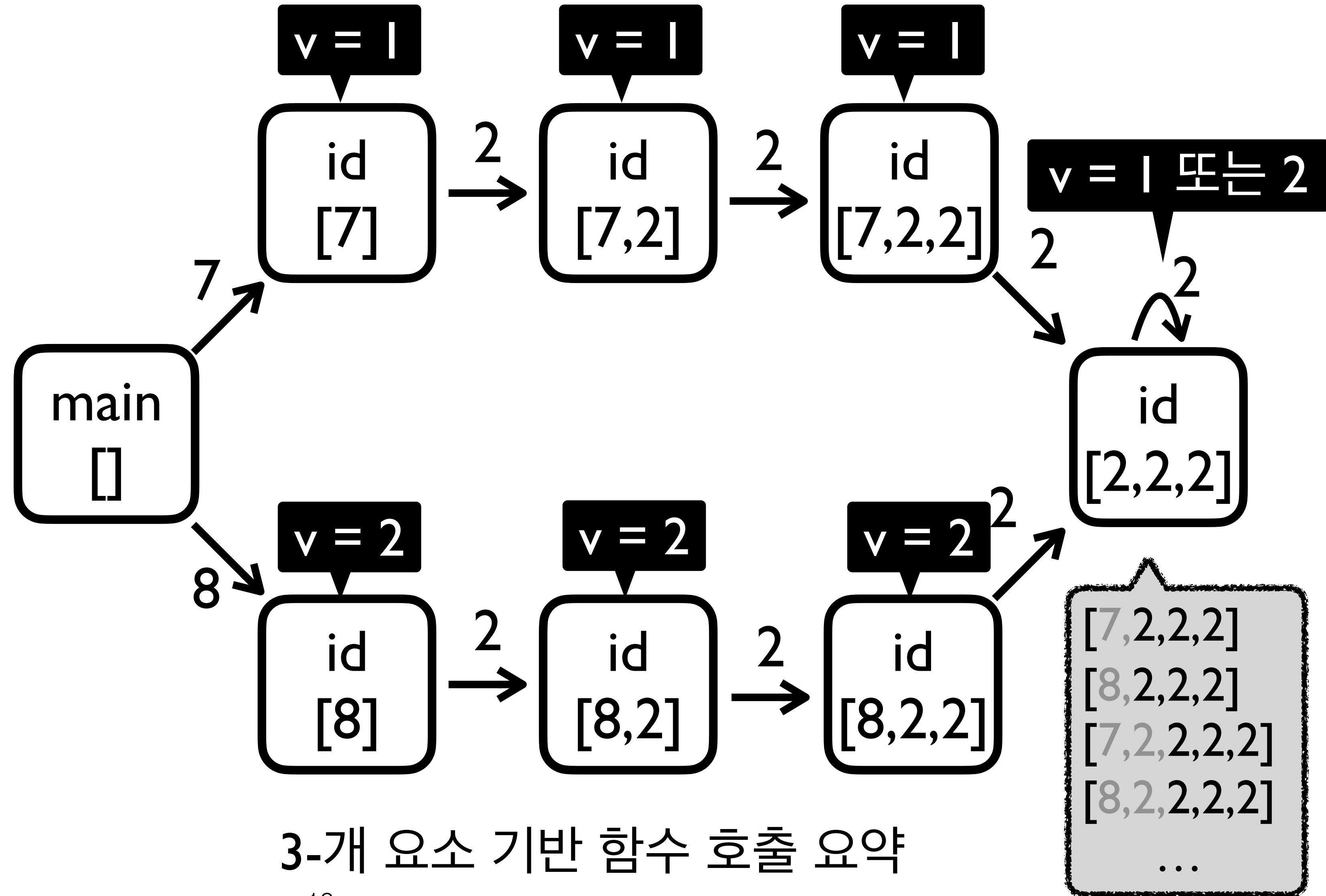


예제 프로그램

2-개 요소 기반 함수 호출 요약

K개 요소 기반 함수 호출 요약

```
0: id(v, i){  
1:   if (i > 0){  
2:     return id(v, i-1);}  
3:   return v;}  
4:  
5: main(){  
6:   i = input();  
7:   v1 = id(1, i); //A  
8:   v2 = id(2, i); //B  
9:   assert (v1 != v2); //query  
10: }
```



예제 프로그램

고정 관념: 마지막 k개 기반 요약

- 마지막 k개 기반 요약 방식이 널리 강의 되는 중

Call-Site Sensitivity



- The best-known flavor of context sensitivity, which uses call-sites 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 call-strings (called k-CFA)

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

- 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



고정 관념: 마지막 k 개 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
- $k = \infty$: keep all contexts, i.e., fully context-sensitive

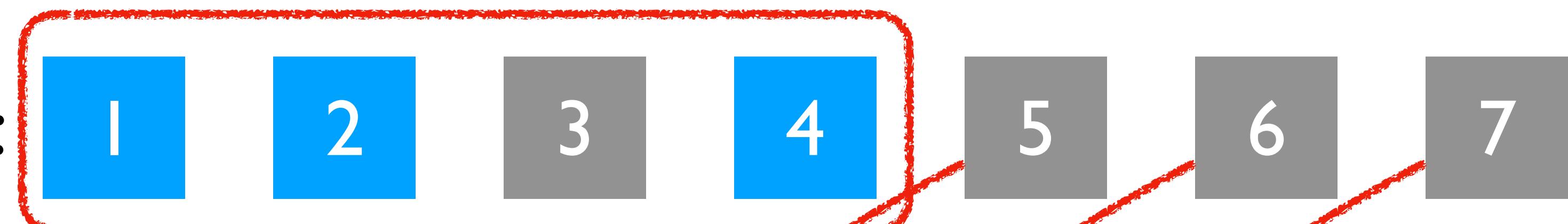


- In practice, k is a small number (usually ≤ 3)
- Method contexts and heap contexts may use different k
 - e.g., $k=2$ for method contexts, $k=1$ for heap contexts

마지막 K개 기반 요약의 문제점

주요 요소들이 지워짐!

실제 함수 호출 맥락:



3개 요소 기반 함수 호출 요약
(3-context sensitivity)

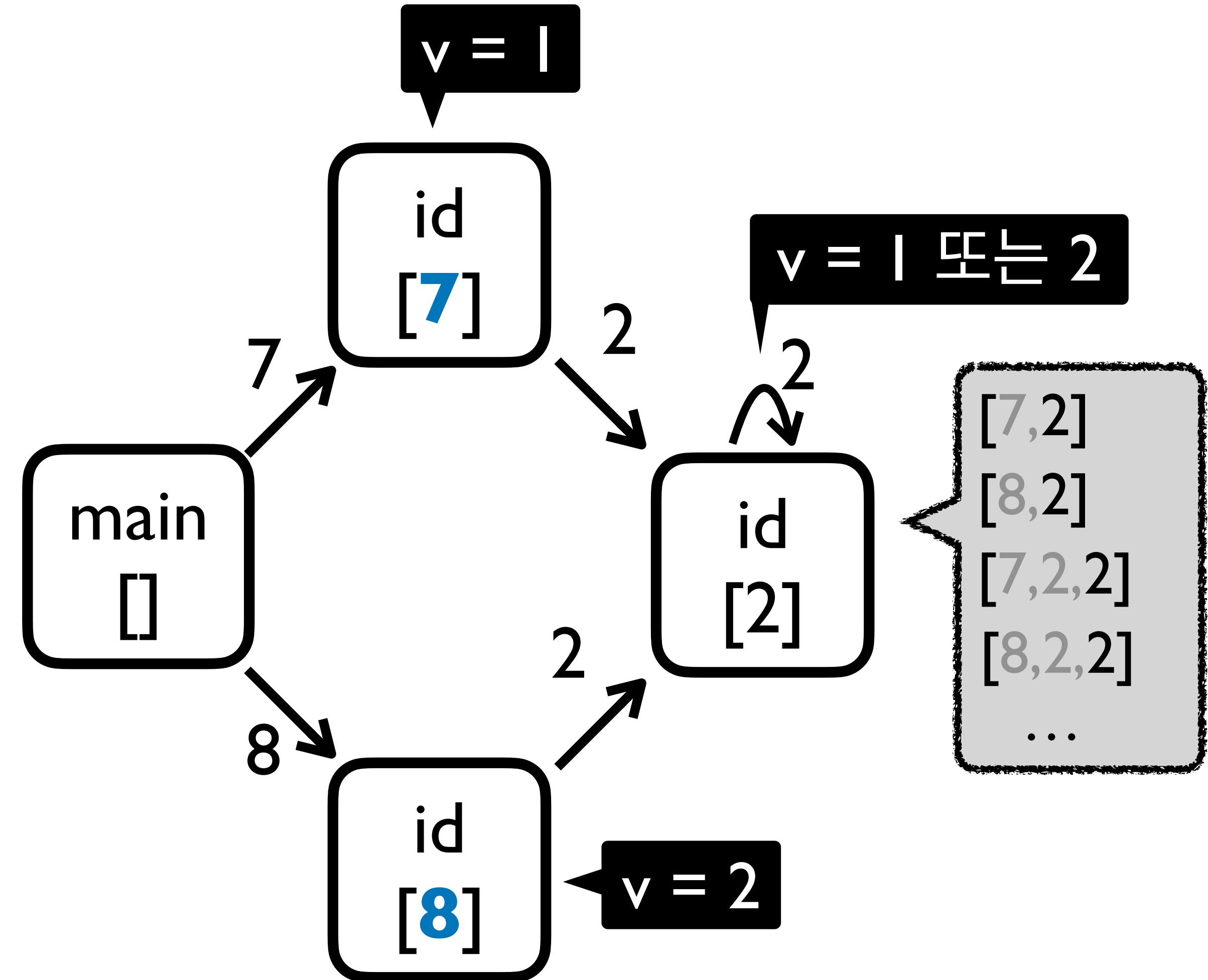


: 주요 요소

: 부차적 요소

마지막 K개 기반 요약의 문제점

```
0: id(v, i){  
1:   if (i > 0){  
2:     return id(v, i-1);}  
3:   return v;}  
4:  
5: main(){  
6:   i = input();  
7:   v1 = id(1, i);  
8:   v2 = id(2, i);  
9:   assert (v1 != v2); //query  
10: }
```



1개 요소 기반 함수 호출 요약

예제 프로그램

발견하게 된 계기

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

발견하게 된 계기

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?

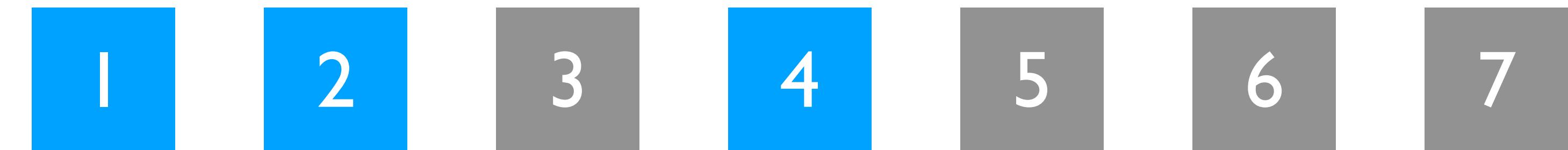
질문:

I-call-site sensitivity로 정확하게 분석할 순 없나?

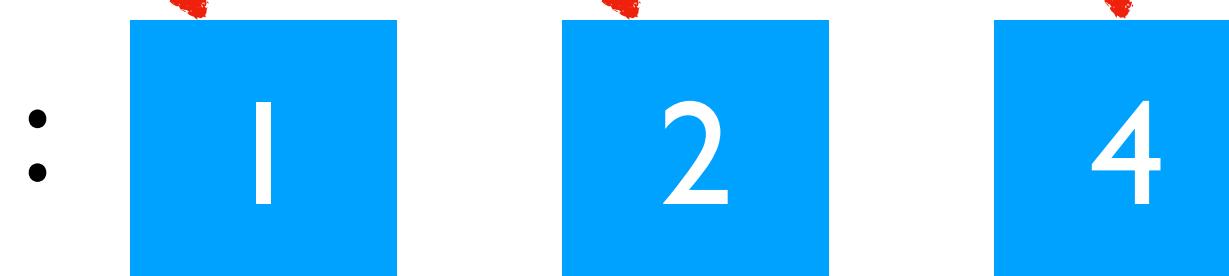
부정확함

호출 환경 배달하기: 주요 K개 기반 요약

실제 함수 호출 맥락:



3개 요소 기반 함수 호출 요약
(3-context sensitivity)



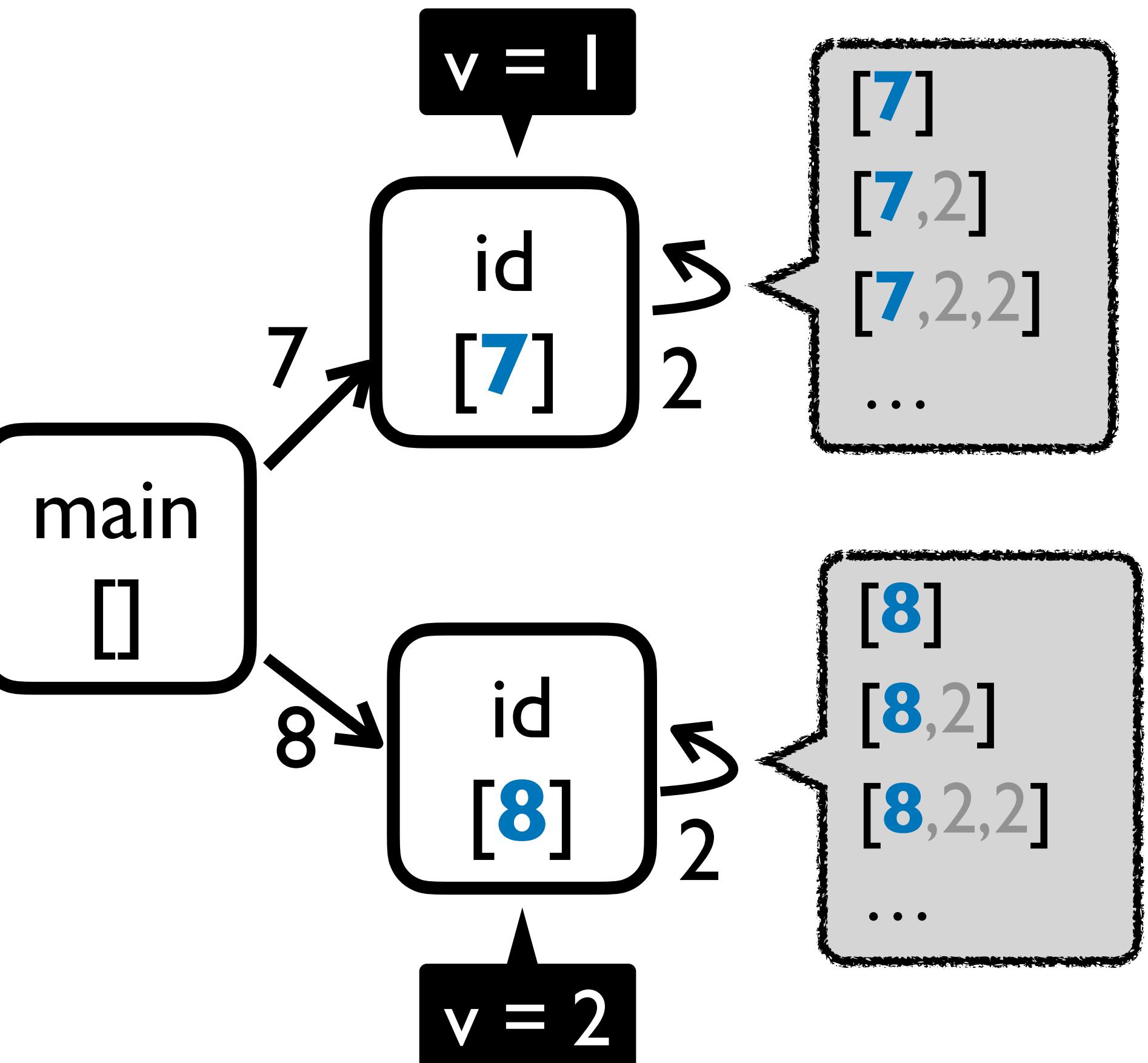
: 주요 요소

: 부차적 요소

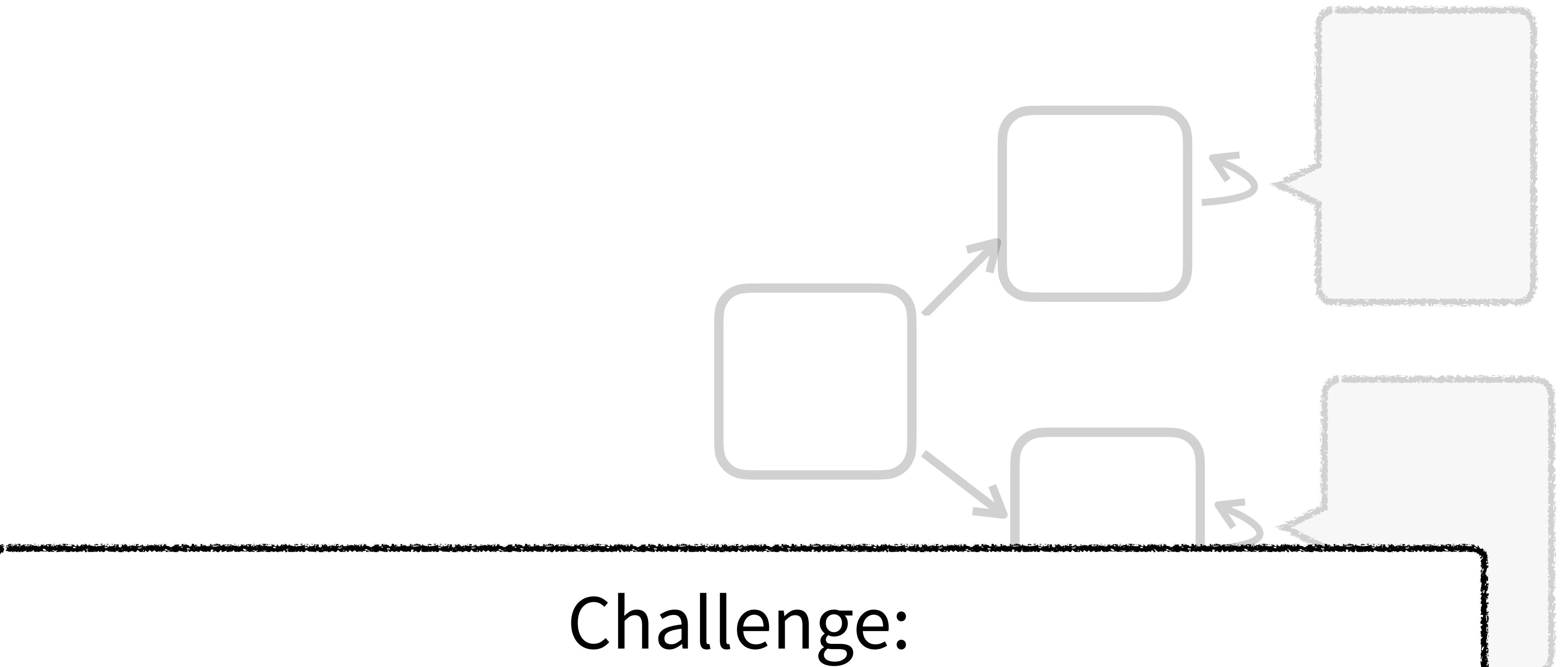
호출 환경 배달하기: 주요 K개 기반 요약

```
0: id(v, i){  
1:   if (i > 0){  
2:     return id(v, i-1);}  
3:   return v;}  
4:  
5: main(){  
6:   i = input();  
7:   v1 = id(1, i);  
8:   v2 = id(2, i);  
9:   assert (v1 != v2); //query  
10: }
```

예제 프로그램



1개 주요 요소 기반 함수 호출 요약
(주요 요소 = {7, 8})

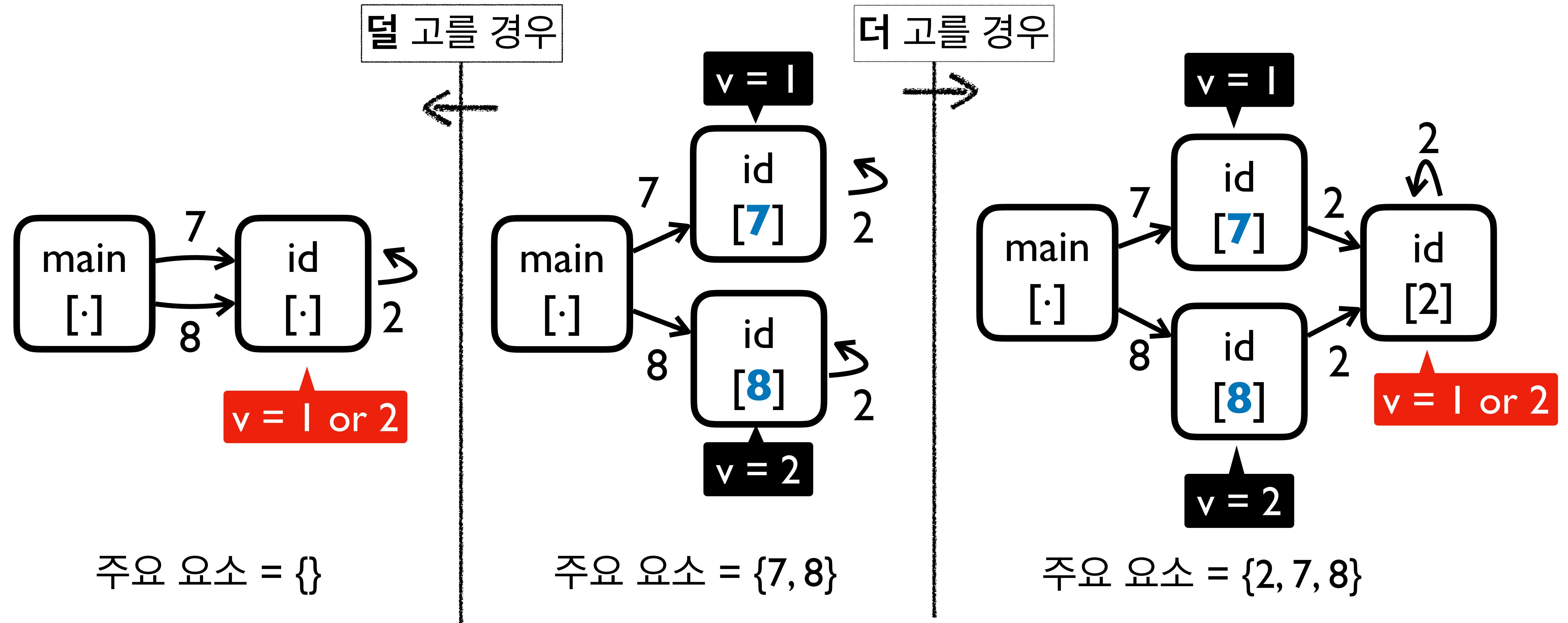


Challenge:
분석 전에 주요 요소를 정확하게 알아내야 함

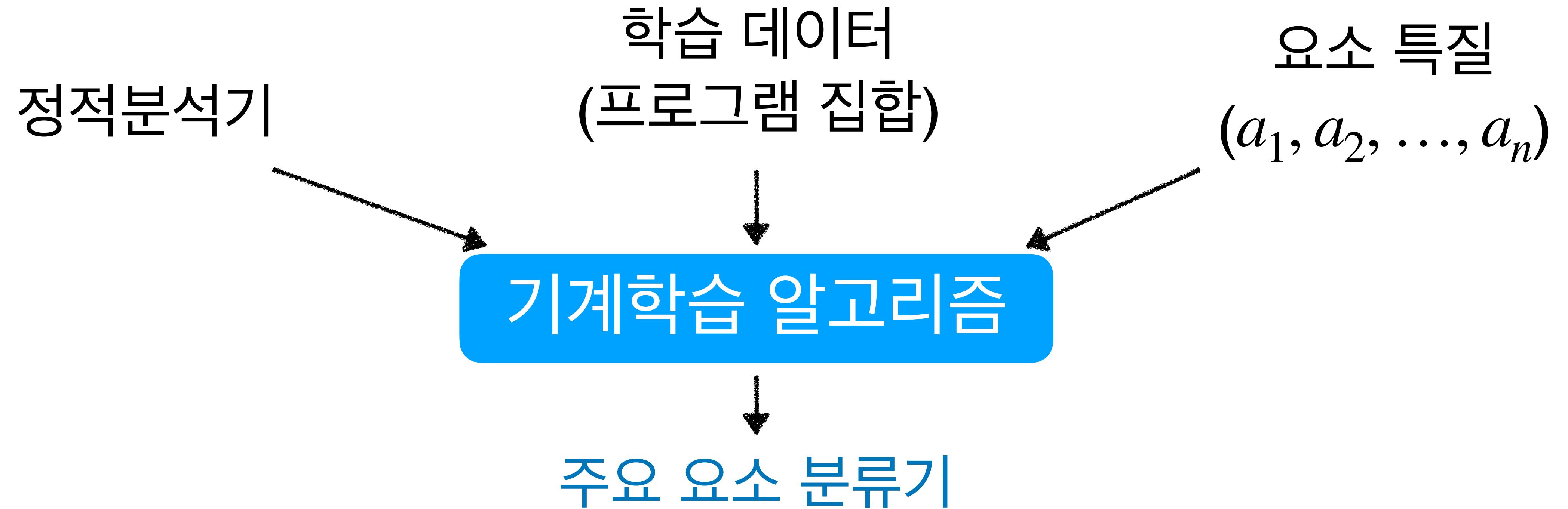
1개 주요 맥락 가변 함수 호출 요약

(주요 요소 = {7, 8})

- 정확하게 분류해야만 분석의 정확도를 향상시킬 수 있음



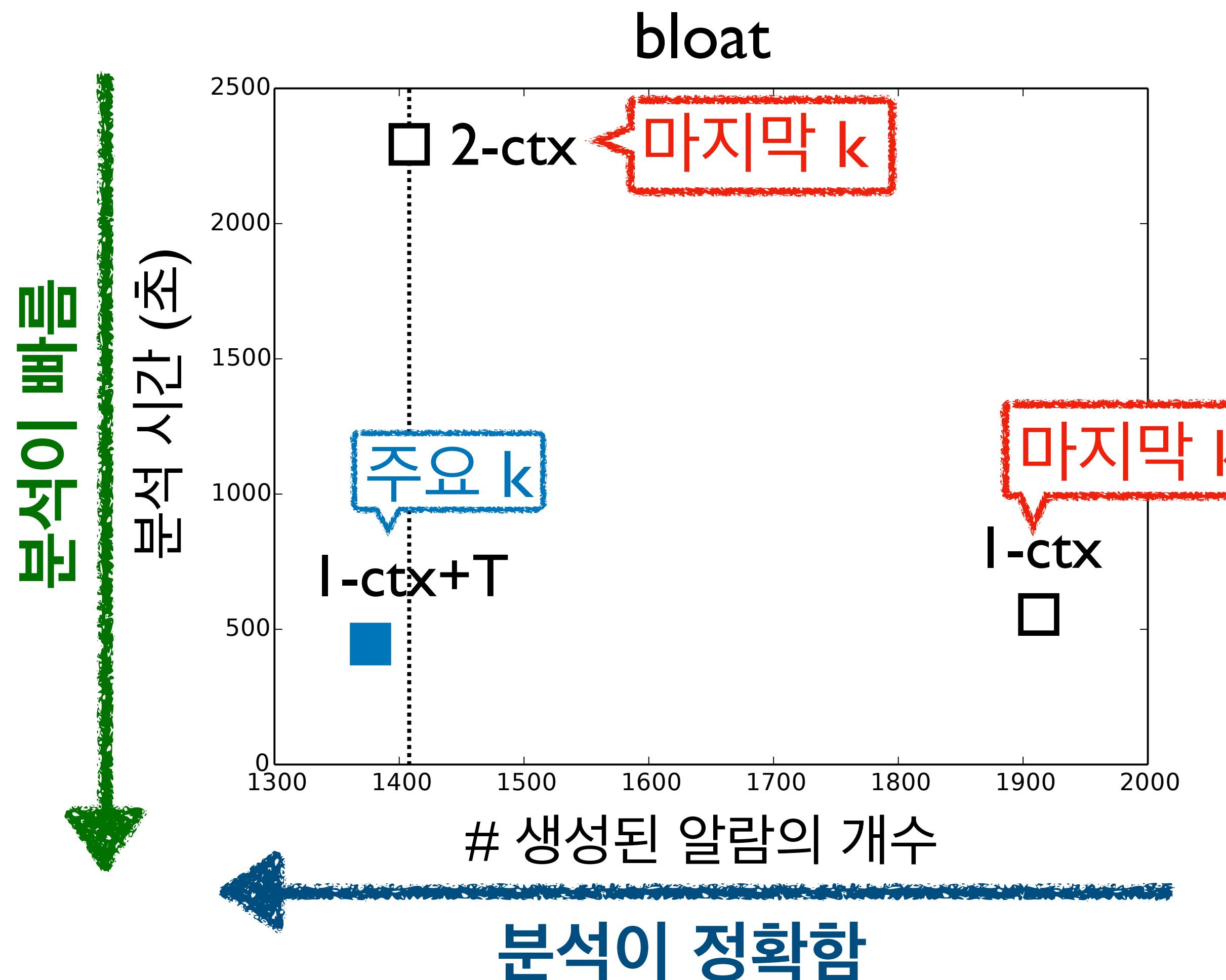
접근 방법: 데이터 기반 컨텍스트 터널링



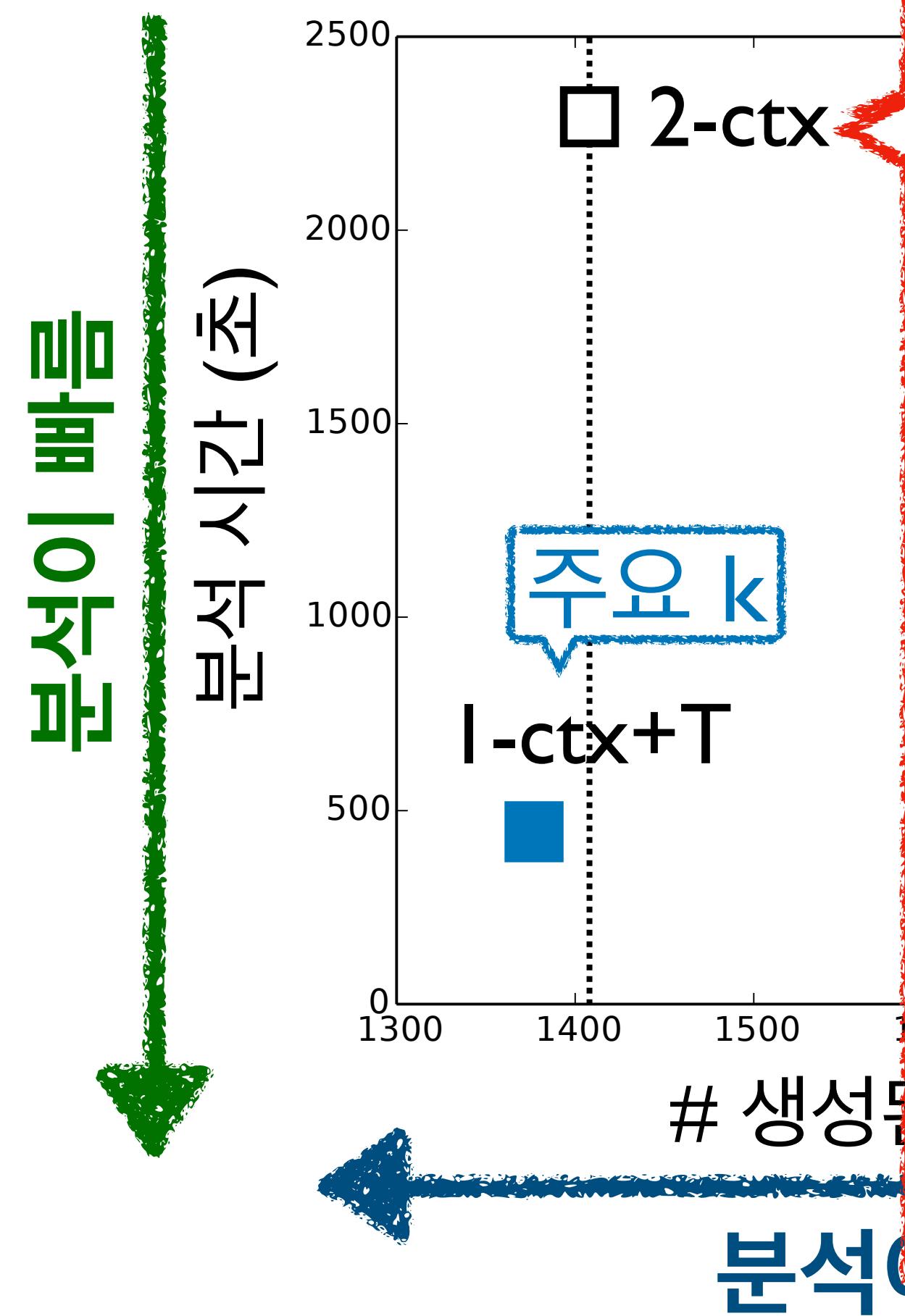
$$f = (a_1 \wedge \neg a_2 \wedge \neg a_3 \wedge \dots) \vee (a_1 \wedge \neg a_3 \wedge a_7 \wedge \dots) \vee \dots$$

마지막 k 기반 요약 vs 주요 k기반 요약

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



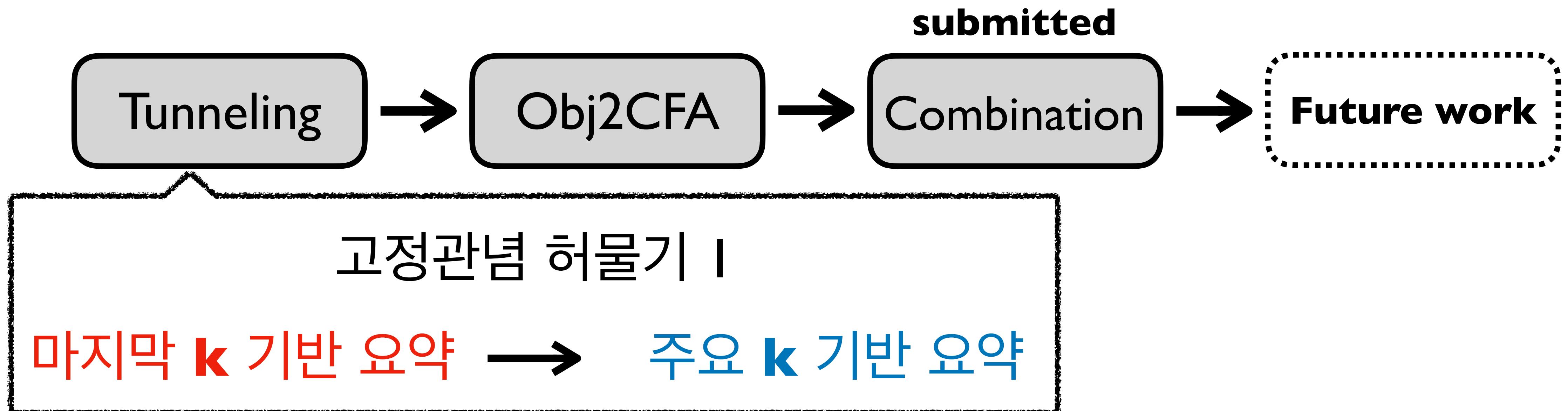
- 2-ctx 는 연구에서 정확도 상한선으로 사용되어 왔음



- “...it covers more than two-thirds of the precision advantage of 2objH”
-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 2obj ...”
-Li et al. [FSE’ 18]
- ...

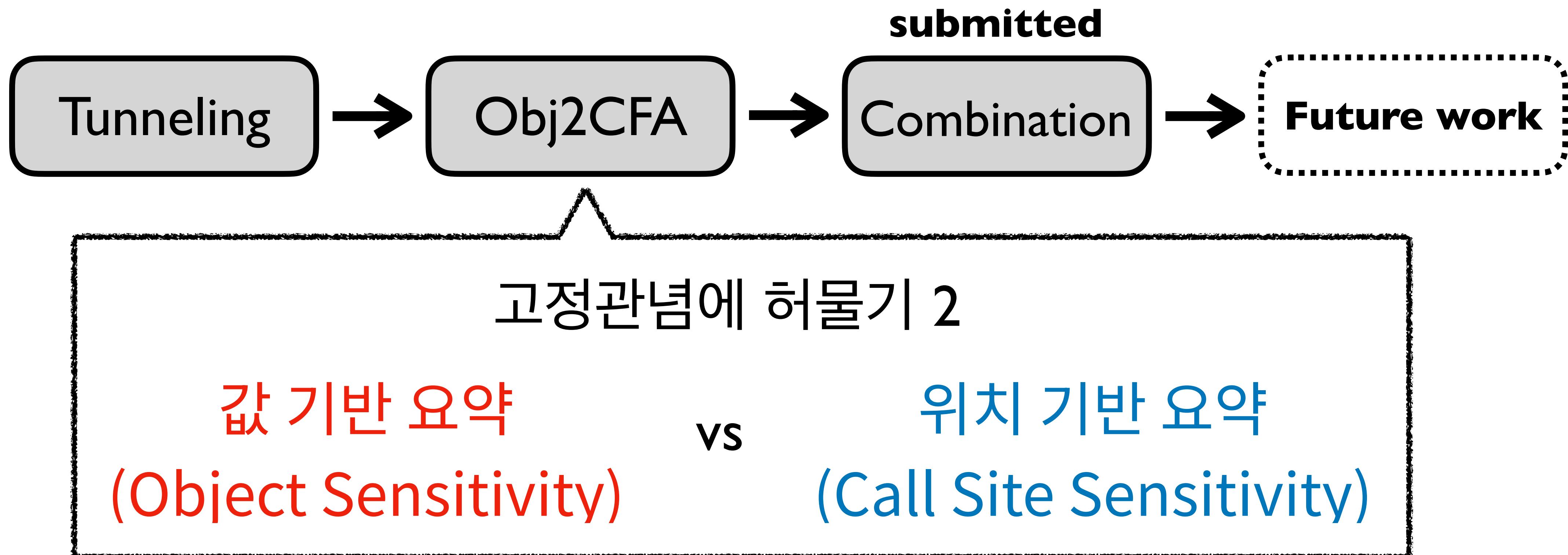
고정관념에 도전하기

- 목표: 주요 k기반 요약 방식을 표준으로 만들기



고정관념에 도전하기

- 목표: 주요 기반 요약 방식을 표준으로 만들기

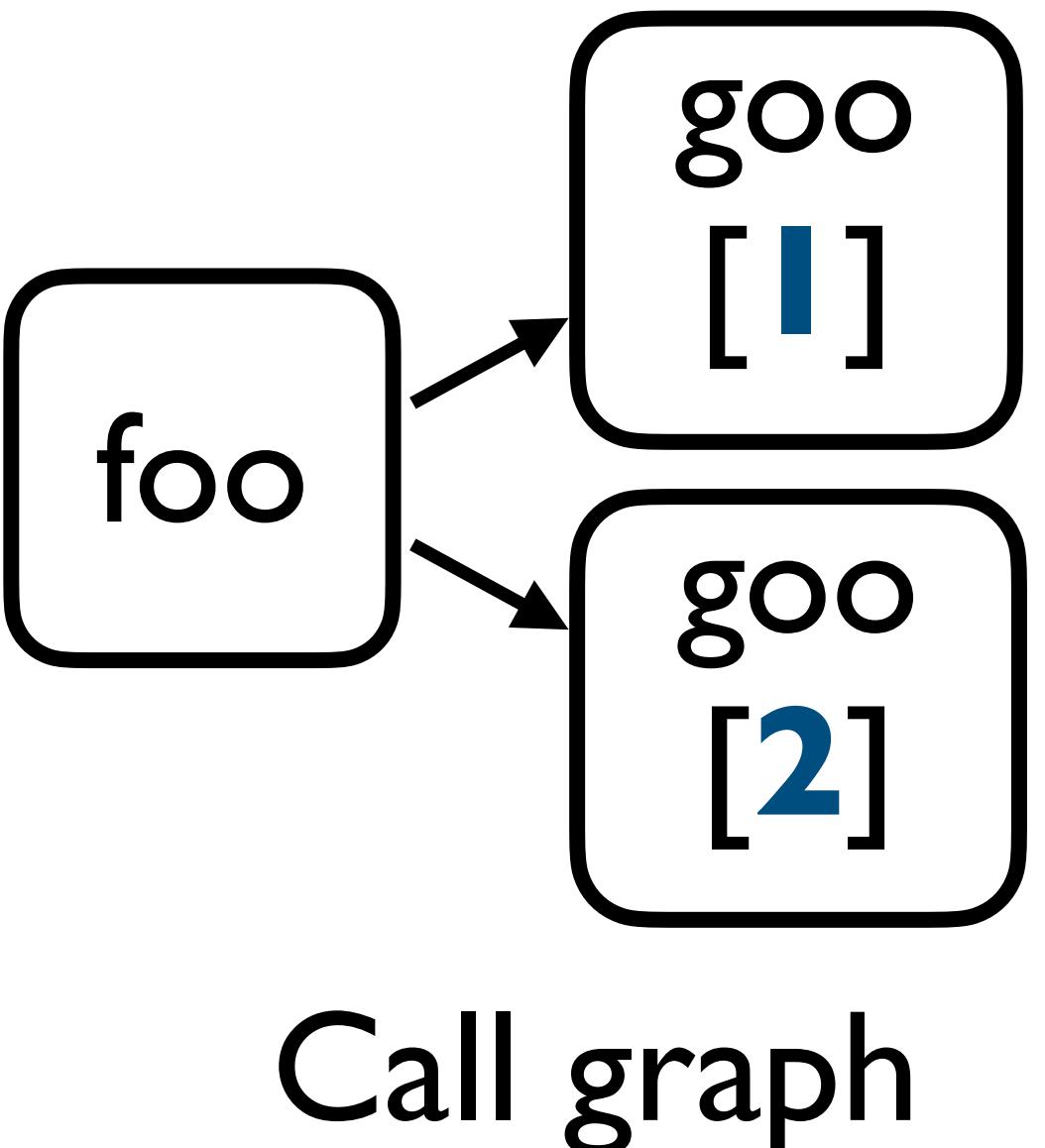


위치 기반 요약 vs 값 기반 요약

위치 기반 요약 (1981)

- “어디”를 기반으로 요약

```
0: foo(){  
1:   goo();  
2:   goo();  
3: }
```



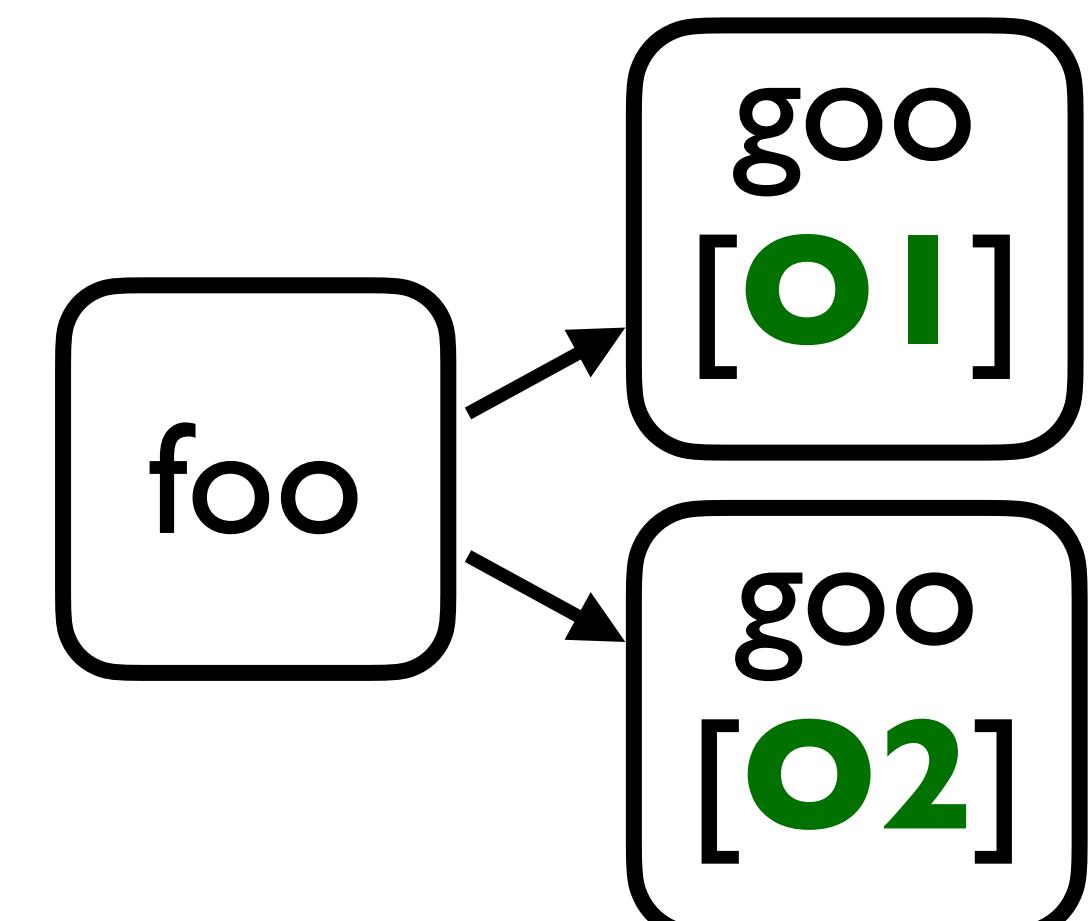
Call graph

값 기반 요약 (2002)

- “무엇”을 기반으로 요약

O1 or O2

```
0: foo(p){  
1:   p.goo();  
2: }
```



Call graph

1981

2002

2010

2022

위치 기반 요약 vs 값 기반 요약

Parameterized Object Sensitivity for Points-to Analysis for Java

ANA MILANOVA
Rensselaer Polytechnic Institute
ATANAS ROUNTEV
Ohio State University
and
BARBARA G. RYDER
Rutgers University

The goal of *points-to analysis* for Java is to determine the set of objects pointed to by a reference variable in the 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 names that represent run-time objects on which this method may be invoked. To ensure flexibility and practicality, we propose a parameterization framework that allows designers to control the tradeoffs between cost and precision in the object-sensitive analysis.

Side-effect analysis determines the memory locations that may be modified by the execution of a program statement. *Def-use analysis* identifies pairs of statements that share the value of a memory location. These analyses are often used to prove program properties. The information computed by such analyses has a wide variety of uses in compilers and software tools. Our work proposes new versions of these analyses that are based on object-sensitive points-to analysis.

We have implemented two instantiations of our parameterized object-sensitive points-to analysis. On a set of 23 Java programs, our experiments show that the cost of our analysis is comparable to a more expensive points-to analysis for Java, which is based on Hendren's analysis for C. Our results also show that object sensitivity significantly improves the precision of side-effect analysis and call graph construction, compared to (1) context-insensitive analysis, and (2) context-sensitive points-to graph analysis that models contexts using the invoking call site. These experiments demonstrate that object-sensitive analyses can achieve substantial precision improvement, while at the same time remaining efficient and practical.

A preliminary version of this article appeared in *Proceedings of the International Symposium on Software Testing and Analysis* (July), 2002, pp. 1–11.

This research was supported in part by National Science Foundation (NSF) grant CCR-9900988.

Authors' addresses: A. Milanova, Department of Computer Science, Rensselaer Polytechnic Institute, 198 8th Street, Troy, NY 12180; e-mail: amilanova@rpi.edu; A. Rountev, Department of Computer Science, The Ohio State University, 196 Neil Avenue, Columbus, OH 43210; e-mail: rountev@cs.ohio-state.edu; B. G. Ryder, Department of Computer Science, Rutgers University, 100 Frelinghuysen Road, Piscataway, NJ 08854; e-mail: ryder@cs.rutgers.edu.

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

Obj 승리

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

Ondřej Lhoták^{1,2} and Laurie Hendren²
olhotak@uwaterloo.ca hendren@sable.mcgill.ca

¹ 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 Whaley, and by Whaley and Lhoták. Our study includes many points-to sets, including specifically specialized 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 implementations, we measure the number of contexts, the number of distinct contexts, and the number of distinct points-to 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 distinct pointer variables, and the number of distinct heap regions.

The results of our study indicate that object-sensitive analysis implementations are likely to scale better and more predictably than the other approaches; that object-sensitive 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.

1 Introduction

Does context sensitivity significantly improve precision of interprocedural analysis of object-oriented programs? It is often suggested that it could, but lack of implementational details 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 implementations of specific analyses, we should have answers to these questions.

This paper aims to provide these answers. Recent advances in the theory of Binary Decision Diagrams (BDDs) in points-to analysis [3, 12, 29, 31] have made context sensitive analysis efficient enough to perform an empirical study on benchmarks of significant size. Using the JEDD system [14], we have implemented three different families of context

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

ONDŘEJ LHOTÁK
University of Waterloo
and
LAURIE HENDREN
McGill University

We present Points-to, a framework for BDD-based context-sensitive points-to and call graph analysis for Java, as well as client analyses that use their results. Points-to supports several variations of context-sensitive analyses, including call site strings and object sensitivity, and context-sensitivity specializes both pointer variables and the heap abstraction. We empirically evaluate the precision 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 that specializing the heap abstraction improves precision more than extending the length of context strings.

Categories and Subject Descriptors: D.4 [Programming Languages]: Processors; D.3.3 [Programming Languages]: Language Constructs and Features

General Terms: Languages, Design, Experimentation, Measurement

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

ACM Reference Format:
Lhoták, O. and Hendren, L. 2008. Evaluating the benefits of context-sensitive points-to analysis using a BDD-based implementation. ACM Trans. Softw. Engin. Methodol. 18, 1, Article 3 (September 2008), 55 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 15th International Conference on Compiler Construction*, Lecture Notes in Computer Science, vol. 3923, Springer, 47–64.

Authors' addresses: O. Lhoták, D. R. Cheriton School of Computer Science, University of Waterloo, 200 University Avenue West, Waterloo, ON, N2L 3G1, Canada; L. Hendren, School of Computer Science, McGill University, 3480 University Street, Room 318, Montreal, QC, H3A 2A7, Canada.

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

Copyright © 2008, ACM, Inc. All rights reserved. Permission to copy and distribute material in whole or in part for internal or personal use is granted without fee if no profit is made and copyright is acknowledged on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Contact permissions@acm.org.

DOI: 10.1145/1391984.1391987. http://doi.acm.org/10.1145/1391984.1391987

ISSN: 0360-0300/08/0103-0047 \$15.00 © 2008 ACM, Inc.

1. Introduction

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

analyses. It is, thus, not surprising that a wealth of research has been devoted to efficient and precise pointer analysis techniques. *Context-sensitive* analyses are the most common class of precise points-to analyses. Context-sensitive analysis algorithms declaratively use Datalog: a logic-based language for defining (recursive) relations. We carry the declarative approach through to our work by describing the full end-to-end analysis in Datalog by specifying aggressively using a novel technique specifically targeting highly recursive Datalog programs.

As a result, Door achieves several benefits, including full declarativity, modularity, and composability.

In this work we present Door, a general and versatile

points-to analysis framework that makes feasible the most precise context-sensitive analyses reported in the literature.

Door implements a range of analyses, including context 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 speeds of an order-of-magnitude faster.

The main elements of our approach are the use of the Datalog language for specifying the program analyses, and the aggressive optimization of the Datalog programs. The use of Datalog for pointer analysis is well-known [13, 23, 29] and high-level [6, 9] is in from use. Our novel pointer analysis approach, however, accounts for several orders of magnitude of performance improvement: unoptimized analyses typically run over 1000 times more slowly. Generally our optimization fits well with the approach of treating programs as a database, by directly targeting the query language and the incremental evaluation of Datalog implementations. Furthermore, our approach is entirely Datalog based, encoding declarations, logical rules, and call graph construction, as well as the handling the full scope and complexity of the Java language (e.g., static initialization, finalizers, reference objects, threads, exceptions, reflection, etc.). This makes our pointer analysis specifications elegant, modular, but also efficient and easy to tune. Generally, our work is a

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

Copyright © 2008, ACM, Inc. All rights reserved. Permission to copy and distribute material in whole or in part for internal or personal use is granted without fee if no profit is made and copyright is acknowledged on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Contact permissions@acm.org.

DOI: 10.1145/1391984.1391987. http://doi.acm.org/10.1145/1391984.1391987

ISSN: 0360-0300/08/0103-0047 \$15.00 © 2008 ACM, Inc.

Obj 승리

Strictly Declarative Specification of Sophisticated Points-to Analyses

Martin Bravenboer Yannis Smaragdakis
Department of Computer Science
University of Massachusetts, Amherst
Amherst, MA 01003, USA
martin.bravenboer@acm.org yannis@cs.umass.edu

Abstract
We present the Door framework for points-to analysis of Java programs. Door builds on the idea of specifying pointer analysis algorithms declaratively, using Datalog: a logic-based language for defining (recursive) relations. We carry the declarative approach through to our work by describing the full end-to-end analysis in Datalog by specifying aggressively using a novel technique specifically targeting highly recursive Datalog programs.

As a result, Door achieves several benefits, including full declarativity, modularity, and composability.

In this work we present Door, a general and versatile

points-to analysis framework that makes feasible the most

precise context-sensitive analyses reported in the literature.

Door implements a range of analyses, including context

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

speeds of an order-of-magnitude faster than previous approaches.

The main elements of our approach are the use of the

Datalog language for specifying the program analyses, and the

aggressive optimization of the Datalog programs. The use of

Datalog for pointer analysis is well-known [13, 23, 29]

and high-level [6, 9] is in from use. Our novel pointer

analysis approach, however, accounts for several orders of

magnitude of performance improvement: unoptimized analyses

typically run over 1000 times more slowly. Generally our

optimization fits well with the approach of treating programs as a

database, by directly targeting the query language and the

incremental evaluation of Datalog implementations. Fur-

thermore, our approach is entirely Datalog based, encoding

declarations, logical rules, and call graph construction,

as well as the handling the full scope and complexity

of the Java language (e.g., static initialization, finaliz-

ers, reference objects, threads, exceptions, reflection, etc.).

This makes our pointer analysis specifications elegant,

modular, but also efficient and easy to tune. Generally, our

work is a

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial

advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Contact permissions@acm.org.

Copyright © 2008, ACM, Inc. All rights reserved. Permission to copy and distribute material in whole or in part for internal or personal use is granted without fee if no profit is made and copyright is acknowledged on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Contact permissions@acm.org.

DOI: 10.1145/1391984.1391987. http://doi.acm.org/10.1145/1391984.1391987

ISSN: 0360-0300/08/0103-0047 \$15.00 © 2008 ACM, Inc.

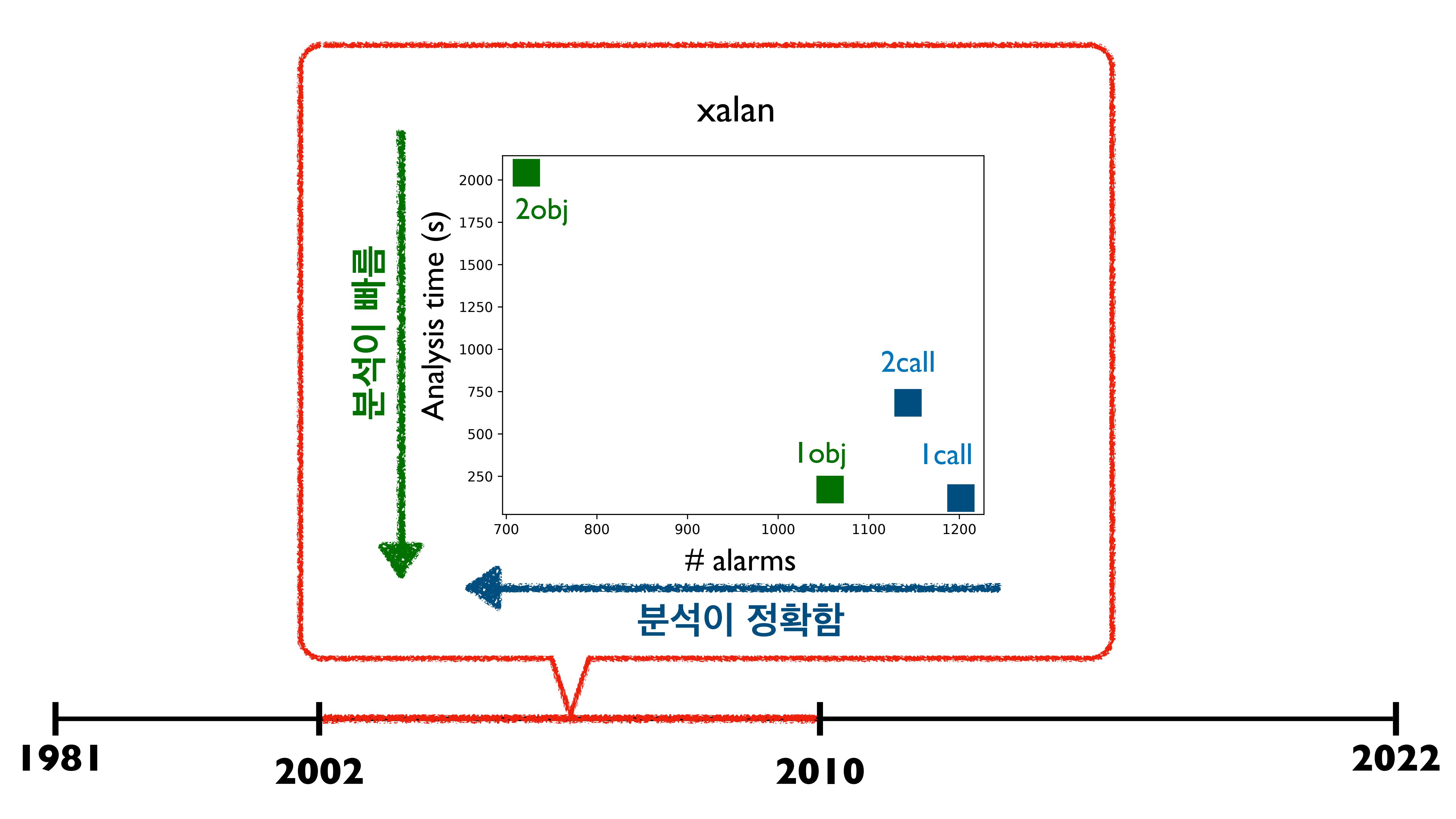
Obj 승리

1981

2002

2010

2022



• 값 기반 요약 방식이 우수하다고 강의되는 중

Object-Sensitivity

- The dominant flavor of context-sensitivity for object-oriented languages.
- It uses object abstractions (i.e. allocation sites) as contexts, qualifying a method's local variables with the allocation site of the 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`.

Hakjoo Oh AAA616 2019 Fall, Lecture 8

**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
University of Athens

Object-sensitive pointer analysis

- Milanova, Rountev, and Ryder. *Parameterizing sensitivity for points-to analysis for Java*. ACM SIGART Eng. Methodol., 2005.
- Lhotak and Hendren. *Context-sensitive pointer analysis: Is it worth it?* CC 06

- Context-sensitive interprocedural pointer analysis
- For context, use stack of receiver objects
- (More next week?)

- Object-sensitive pointer analysis more precise than call-site sensitivity for Java
- Likely to scale better

VE RI
TAS
HARVARD

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 complex. Consider constant-propagation analysis of the following program:

```
1: z := 1
2: p := &z
3: *p := 2
4: print z
```

In order to analyze this program correctly we must be aware that instruction 3 `p` points to `z`. If this information is available we can analyze the function as follows:

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

When we know exactly what a variable `x` points to, we say that `x` has a *must-point-to* information, and we can perform a *strong update* on variable `x`, because we know with confidence that assigning to `x` will result in `x` pointing to the same value. A technicality in the rule is quantifying over all `z` such that `p` points to `z`. How is this possible? It is not possible in C or Java; they are languages with pass-by-value. In a language with pass-by-reference, for example C++, it is possible to have two distinct locations `p` point to the same names for the same location are in scope.

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

Yannis Smaragdakis
University of Athens
smaragd@di.uoa.gr
George Balatsouras
University of Athens
gbalats@di.uoa.gr

now
the essence of knowledge
Boston — Delft

Pointer Analysis

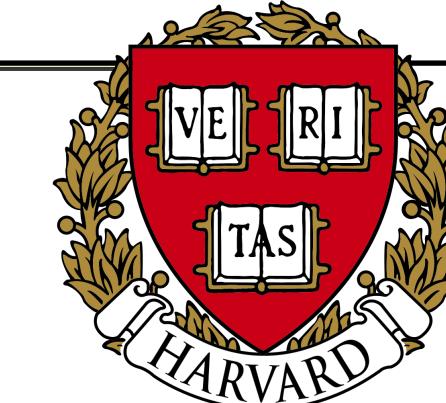
• • •



KOREA
UNIVERSITY



National and Kapodistrian
University of Athens



Carnegie
Mellon
University

now
the essence of knowledge

1981

2002

2010

2022

값 기반 요약을 대상으로 하는 연구가 쏟아짐

Pick Your Contexts Well: Understanding the Making of a Precise and Scalable Hybrid Context Sensitivity

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

Precision-Guided Context Sensitivity for Pointer Analysis

Data-Driven Context-Sensitivity for Points-to Analysis

Precision-Preserving Yet Fast Object-Sensitive Pointer Analysis with Partial Context Sensitivity

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

Efficient and Scalability-First Self-Tuning

Learning Graph-based Heuristic without Handcrafting Application

JINGBO LU, UNSW Sydney, Australia

JINGLING XUE, UNSW Sydney, Australia

Traditional context-sensitive pointer analysis is hard to scale for large and complex Java programs. To address this issue, a set of selective context-sensitivity approaches have been proposed and exhibit promising results. In this work, we move one step further towards producing highly-precise pointer analyses for hard-to-analyze Java programs by presenting the Unity-Relay framework, which takes selective context sensitivity to the next level. Briefly, Unity-Relay is a one-to-many fusion of a set of different selective context-sensitivity approaches, say $S = S_1, \dots, S_n$. Unity-Relay first provides a mechanism to combine and maximize the precision of all components of S . When Unity fails to scale, Unity-Relay offers a scheme called Unity₂ 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 more precise than all approaches in S .

As a proof-of-concept, we instantiate Unity-Relay into a tool called BARON and extensively evaluate it on various software engineering tools. The goal of pointer analysis is to precisely estimate heap objects that pointer variables may reference. For object-oriented languages such as Java, pointer analysis is typically performed by abstracting objects for modeling every client problem with different clients. To tame the combinatorial explosion of client problems, we propose a method to abstract objects for modeling every client problem with different clients. The goal of pointer analysis is to precisely estimate heap objects that pointer variables may reference. For object-oriented languages such as Java, pointer analysis is typically performed by abstracting objects for modeling every client problem with different clients. To tame the combinatorial explosion of client problems, we propose a method to abstract objects for modeling every client problem with different clients.

1981

2002

2010

2022

위치 기반 요약은 사장됨

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

Tan et al. [2016]

1981

2002

2010

위치 기반 요약은 사장됨

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

- Li et al. [2018]

1981

2002

2010

위치 기반 요약은 사장됨

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

”
... .

- Jeon et al. [2019]

The image displays a grid of 10 academic conference papers from the ACM SIGART 2018 proceedings. Each paper is represented by a vertical column containing the title, authors, abstract, and other details. The titles include "Making k-Object-Sensitive Pointer Analysis More Precise with Still k", "Scalability-First Pointer Analysis Self-Tuning Context-Sensitivity", "Pick Your Contexts Well: Understanding the Making of a Precise and Scalable Pointer Analysis", "Hybrid Context-Sensitivity for Points-to Analysis", "Precision-Guided Context Sensitivity for Points-to Analysis", "Introspective Analysis: Context-Sensitivity, Across the Board", and "A Machine-Learning Algorithm with Disjunctive Model for Data-Driven Program Analysis". Each column also includes the names of the authors, their institutions, and a brief abstract or description of the research.

1981

2002

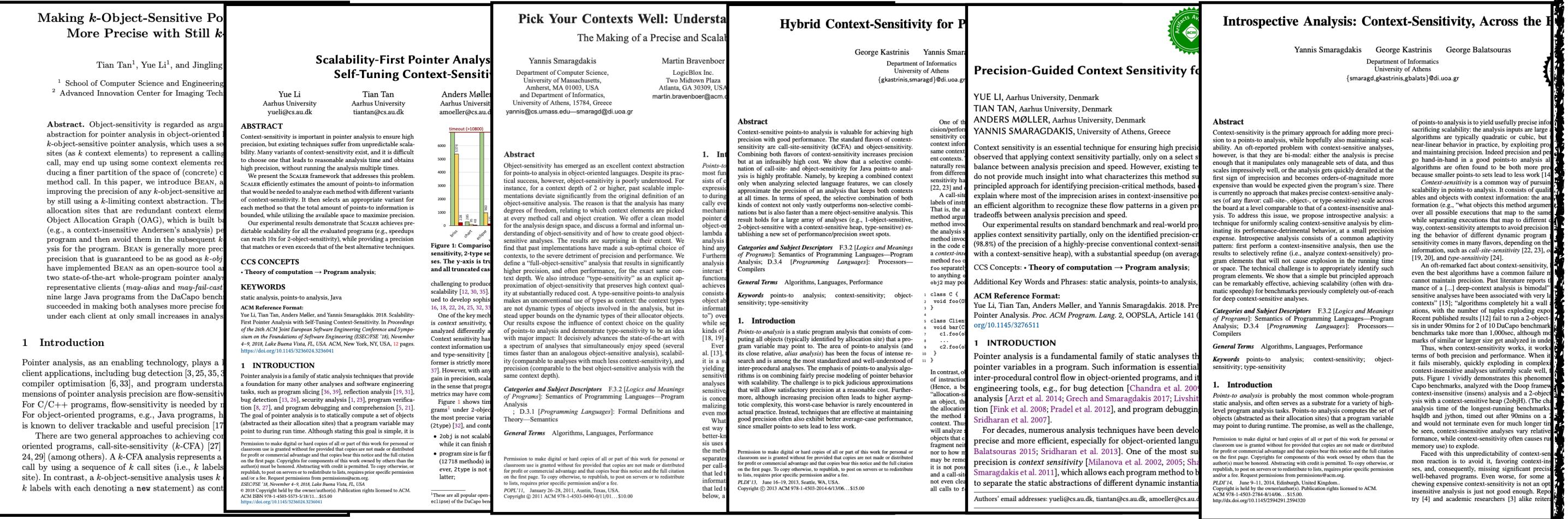
2010

위치 기반 요약은 사장됨

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

‘ ’
... .

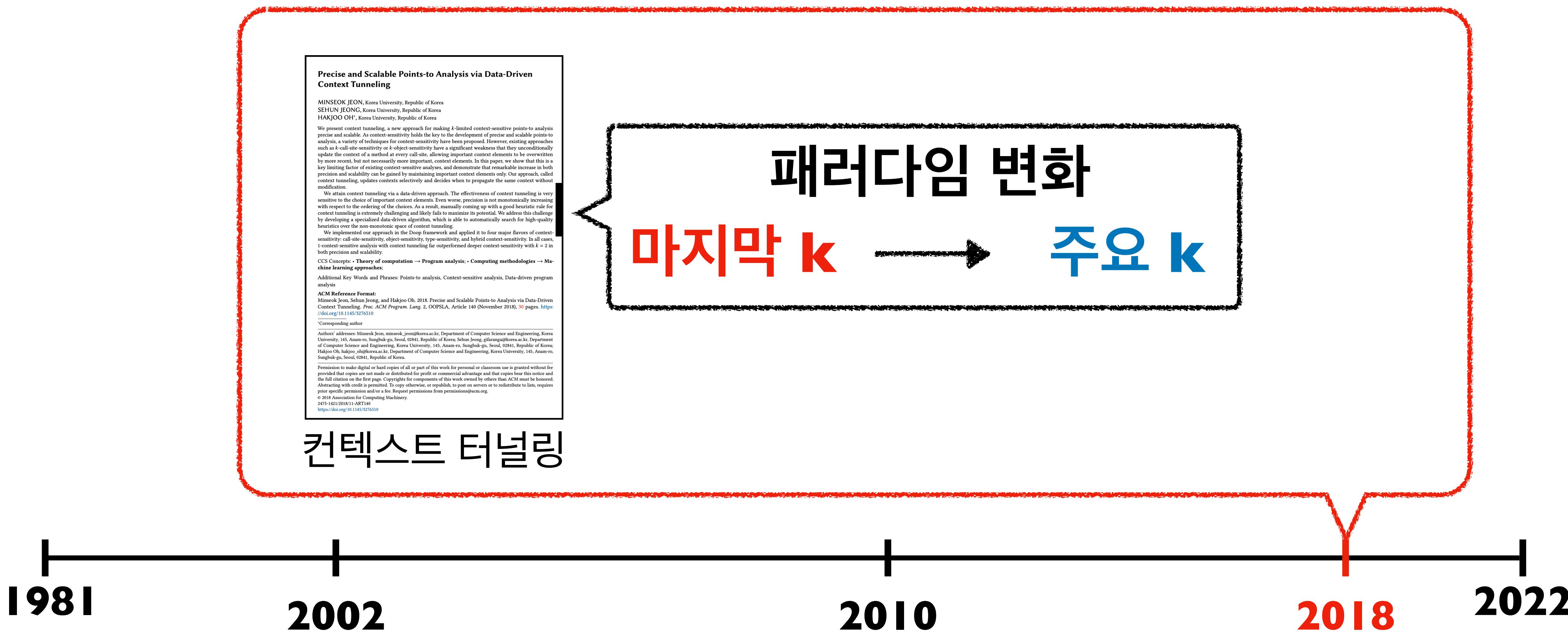
- Jeon et al. [2019]



1981

2002

2010



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

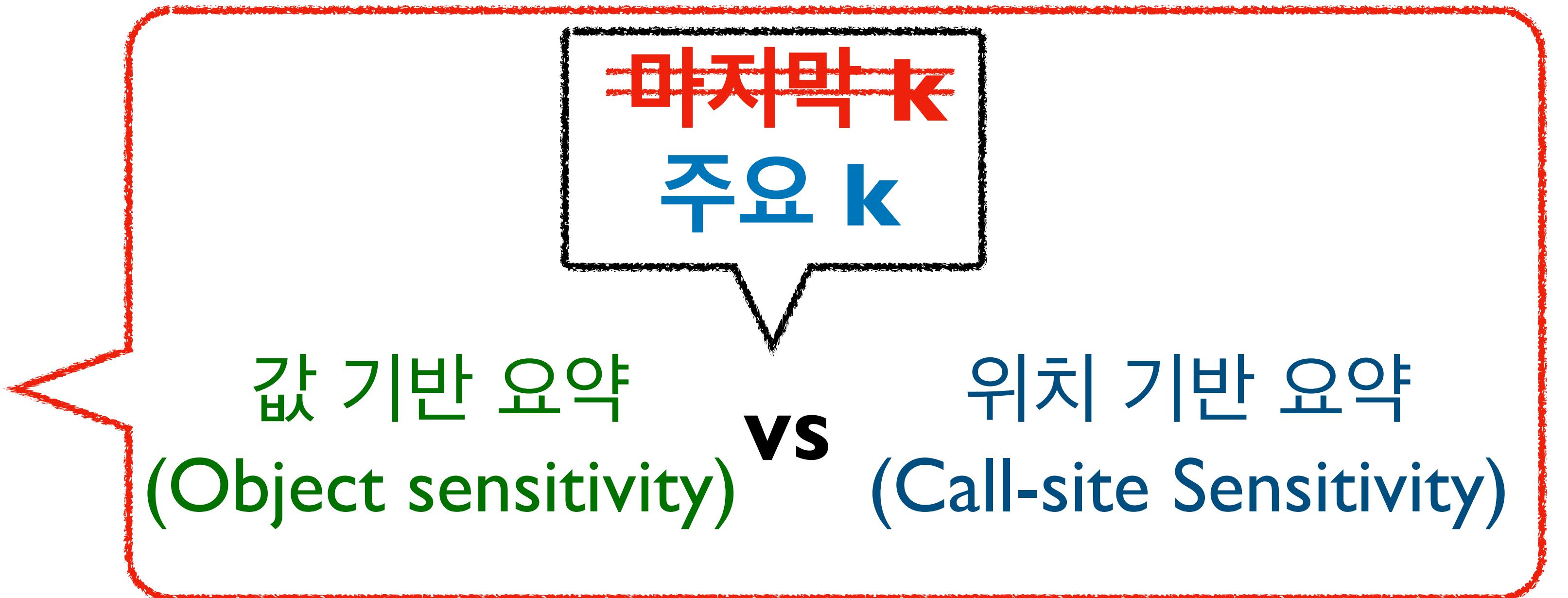
“Since its introduction, object sensitivity has emerged as the dominant flavor of context sensitivity for object-oriented languages.”

—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 programs, the dominant flavor of context sensitivity is object sensitivity. This means that for incoming objects, the analysis maintains a sequence of contexts for each object. Smaragdakis and Balatsouras [2015] argue that “the dominant flavor of context sensitivity for object-oriented programs is object sensitivity.” Sharir et al. [1981] propose a context-sensitive pointer analysis for C programs that uses the allocation site of the receiver object (`a`) as the context of `free(a)`. The standard k -object-sensitive analysis [Milanova et al. 2002, 2005; Smaragdakis et al. 2011] maintains a sequence of

Call 승리!



1981

2002

2010

2018

2022

발견하게 된 계기

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

발견하게 된 계기

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.

정확함
~~부정확함~~

정확함

??

obj > call

발견하게 된 계기

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



발견하게 된 계기

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 programs, the dominant flavor of context sensitivity is object sensitivity. This means that for incoming objects, the analysis maintains a sequence of contexts for each object. Smaragdakis and Balatsouras [2015] argue that “the dominant flavor of context sensitivity for object-oriented programs is object sensitivity.” Sharir et al. [1981] propose a context-sensitive pointer analysis that uses the allocation site of the receiver object (`a`) as the context of `foo`. The standard k -object-sensitive analysis [Milanova et al. 2002, 2005; Smaragdakis et al. 2011] maintains a sequence of

Call 승리!

1981

2002

2010

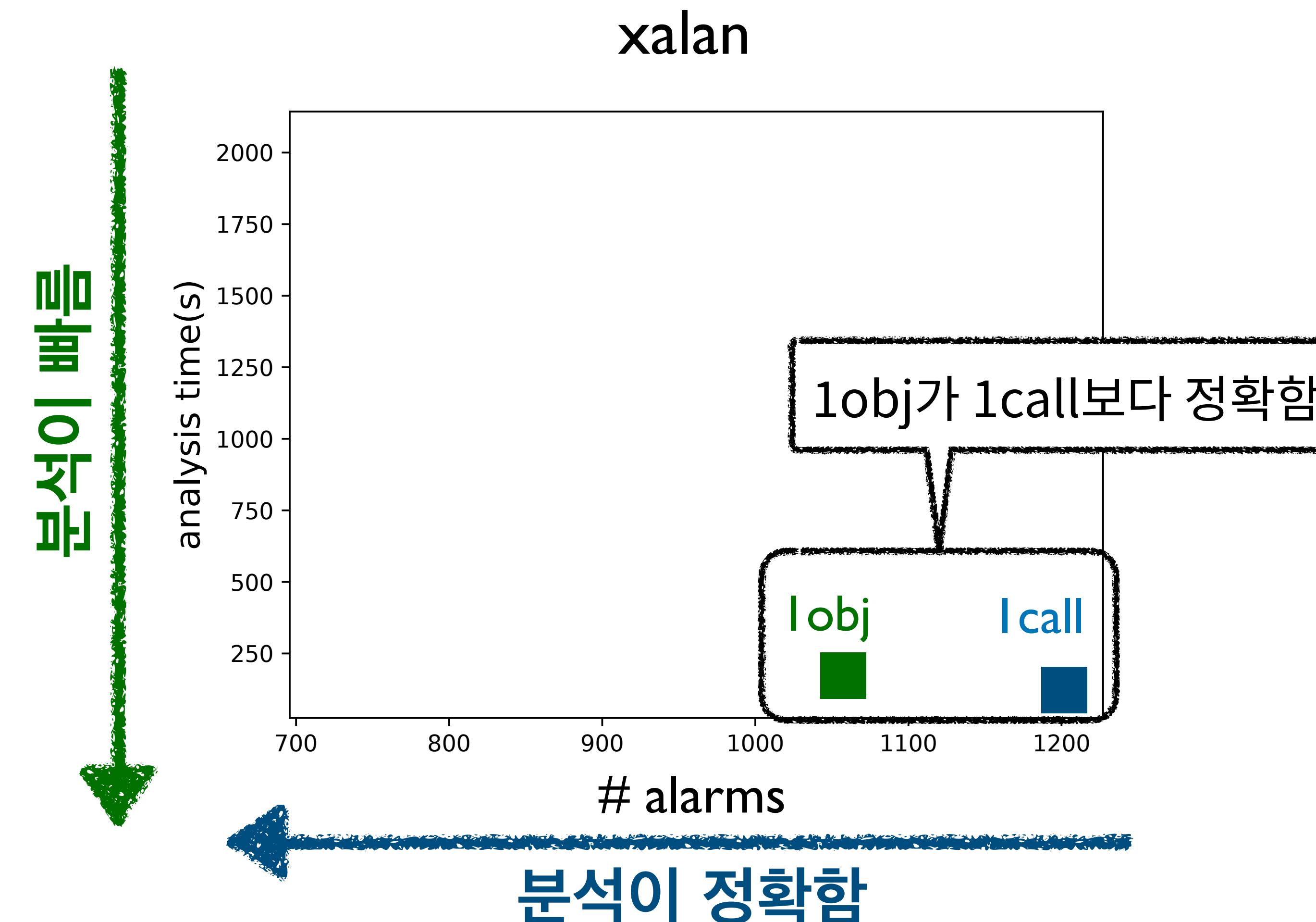
2018

2022



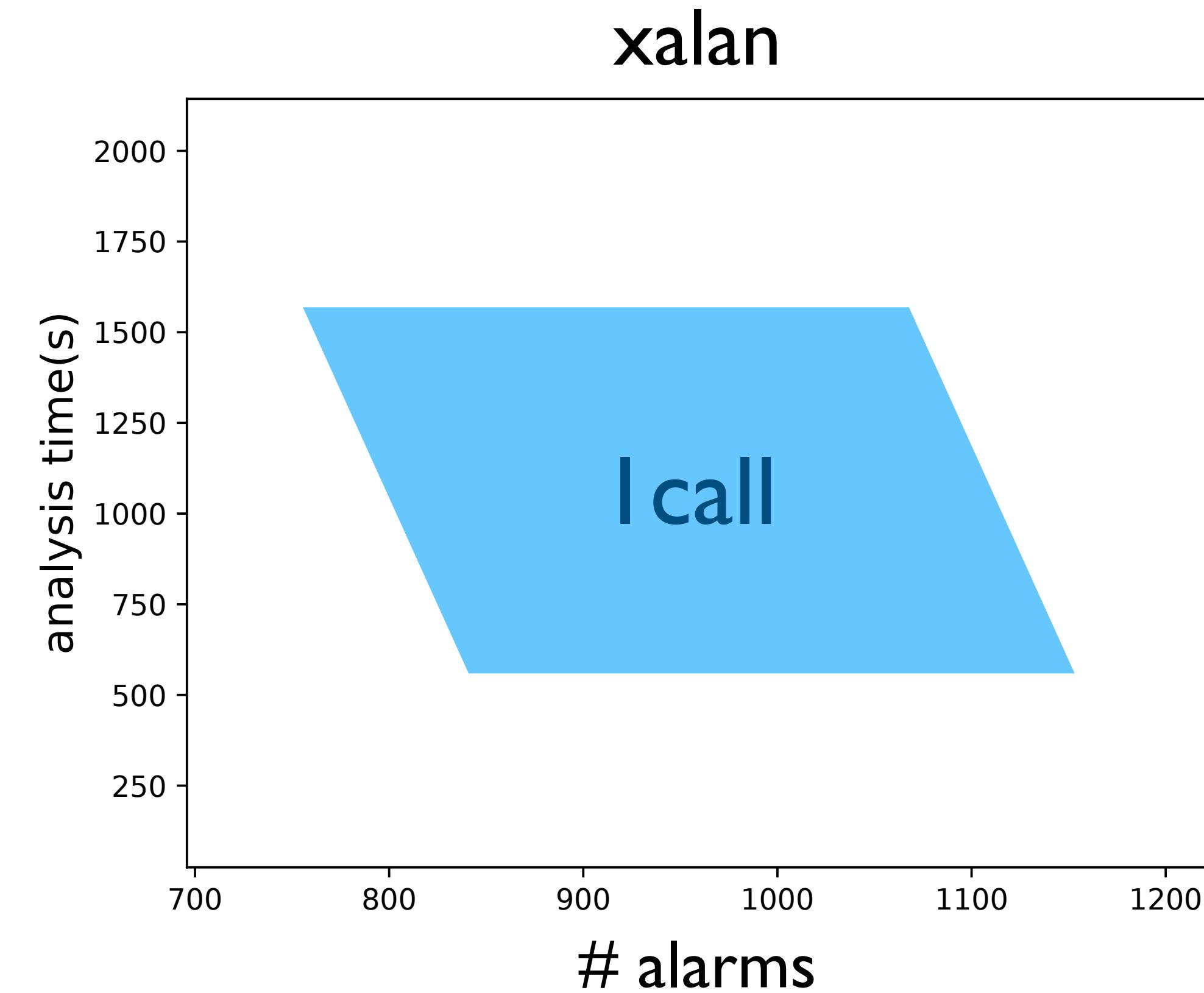
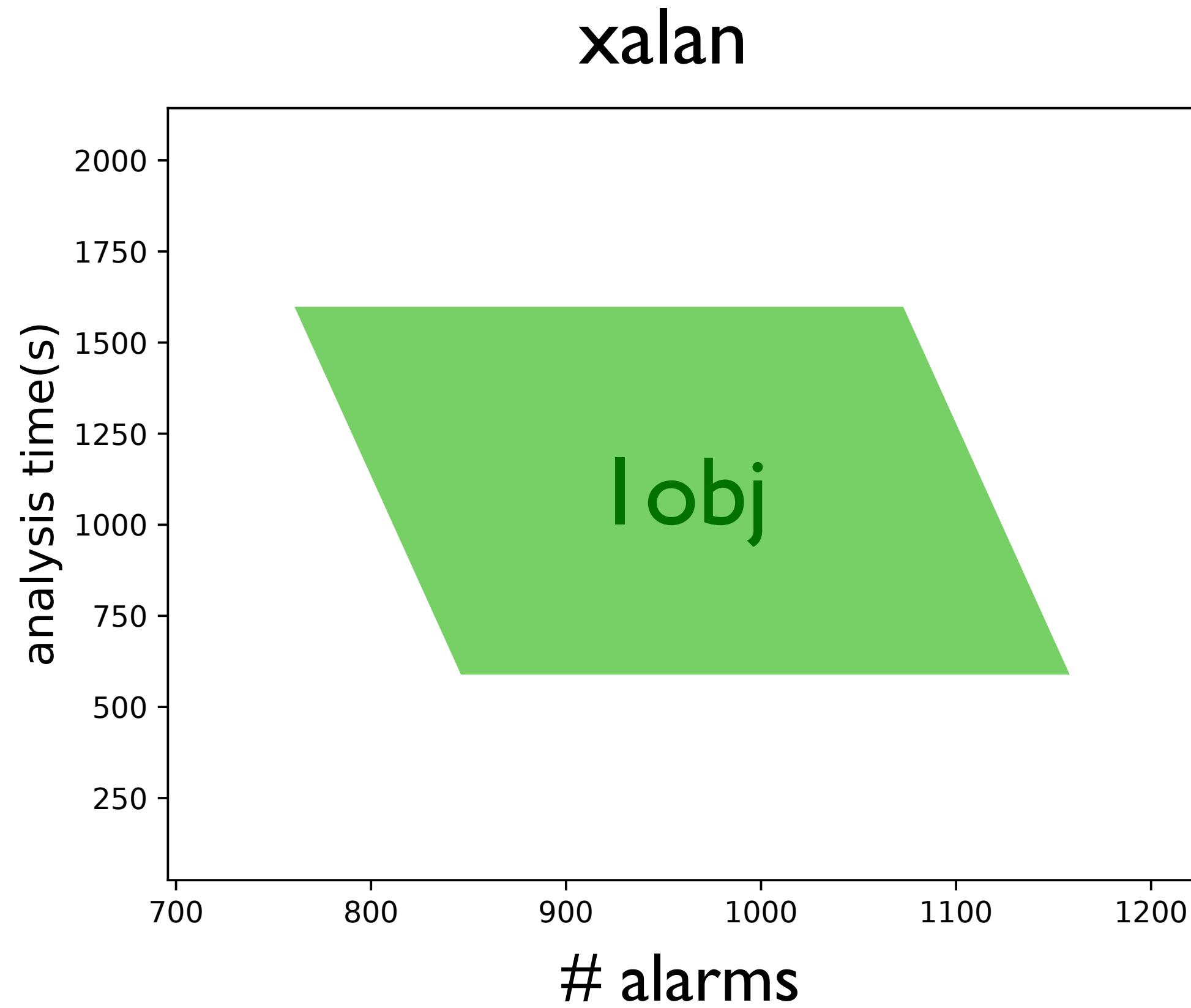
마지막 k 기반 요약에서의 성능 비교

- 분석의 성능을 직접 비교하면 됨



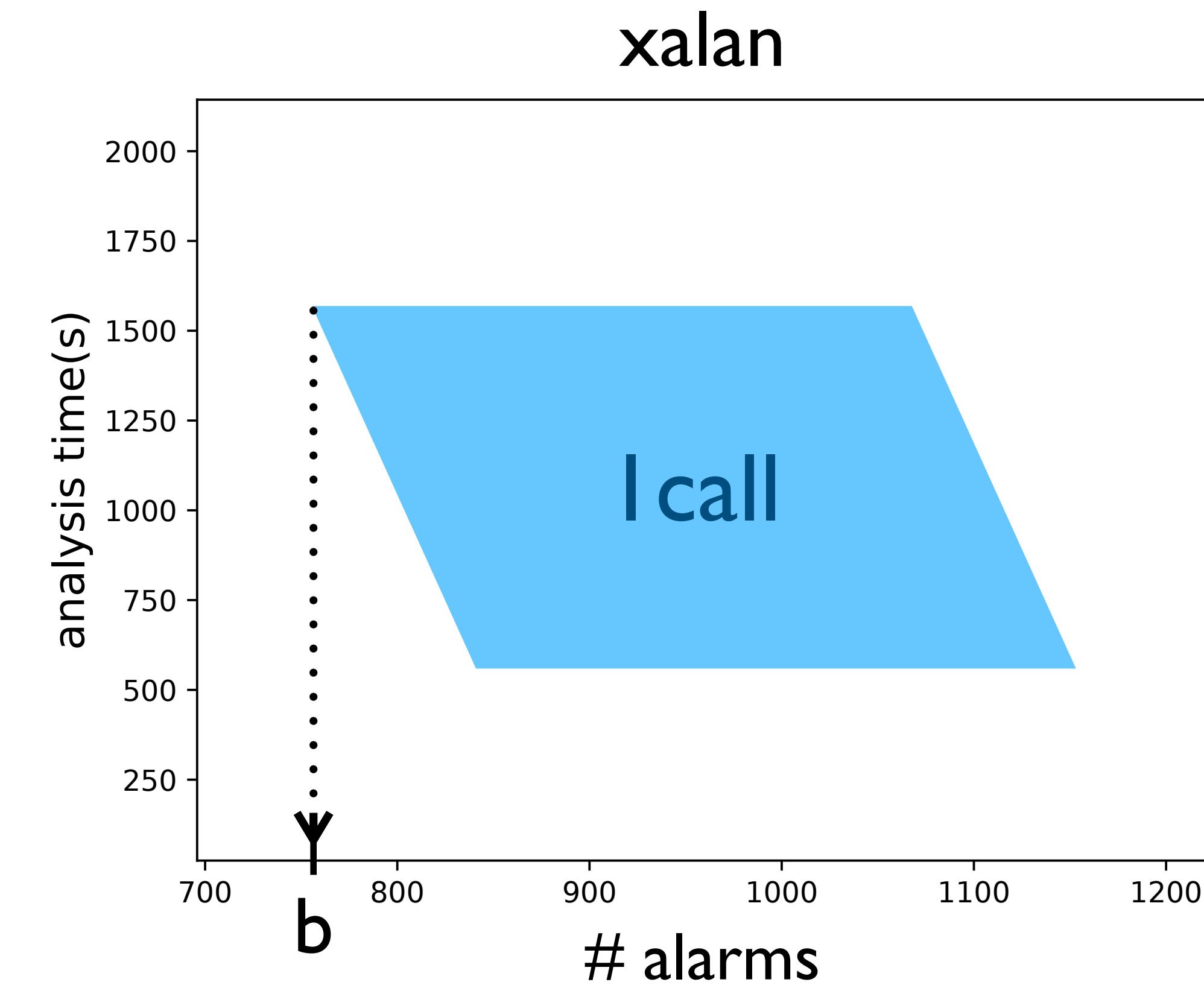
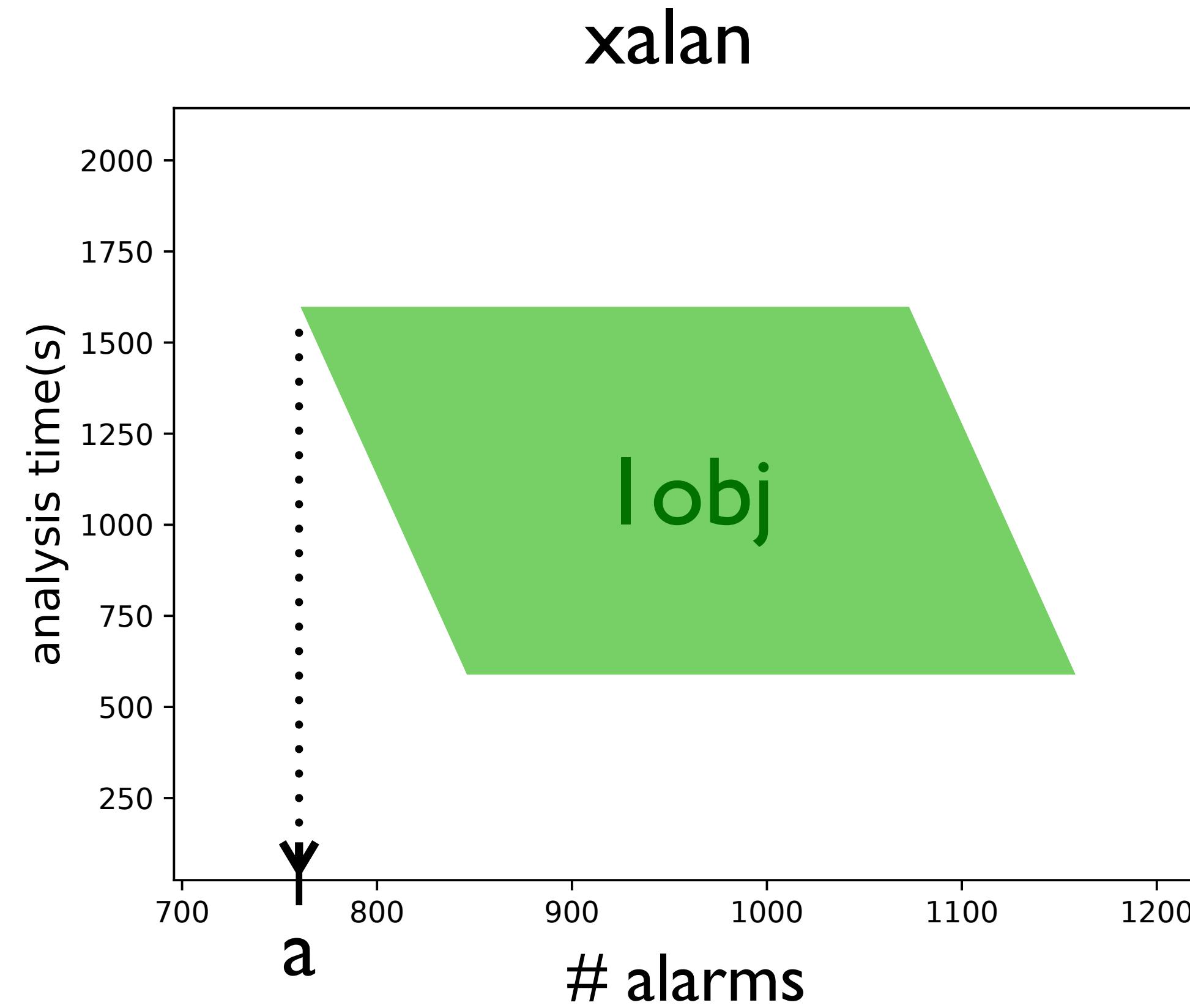
주요 k 기반에서의 성능 비교

- 주요 요소 분류에 따라 성능이 바뀜



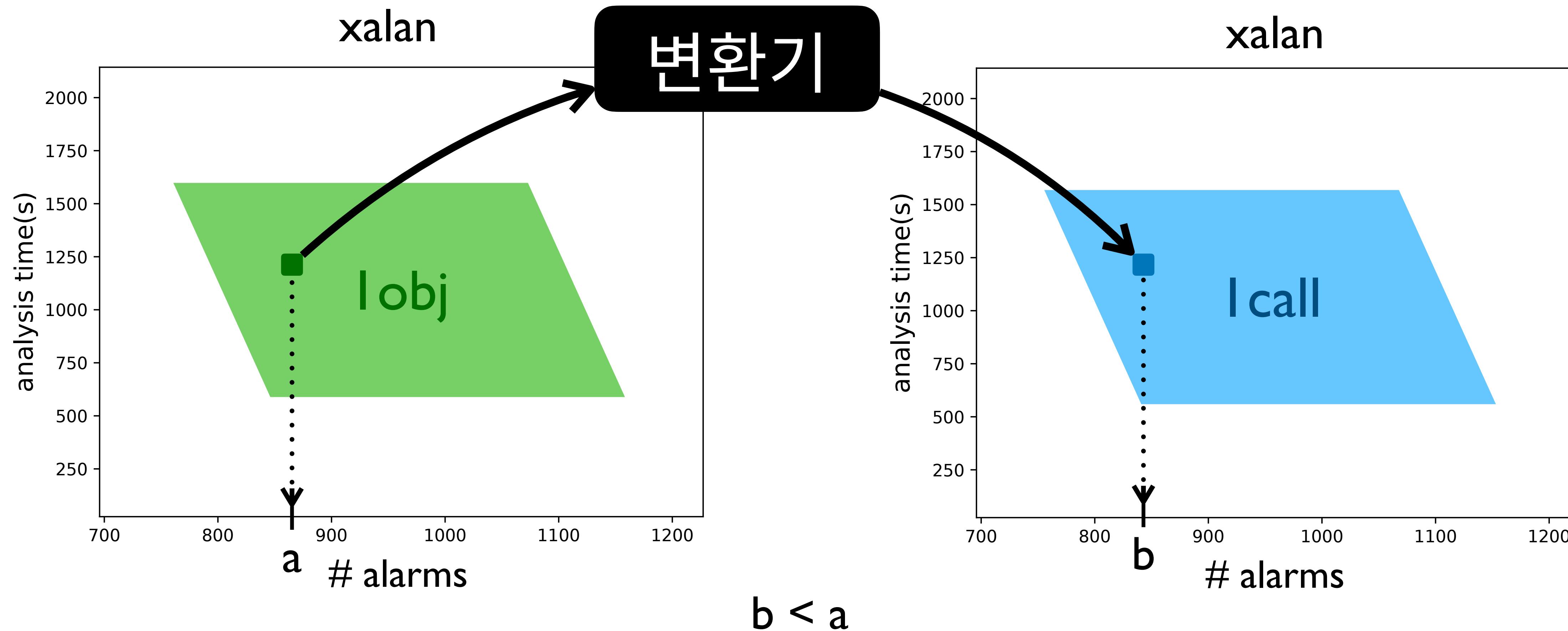
주요 k 기반에서의 성능 비교

- 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



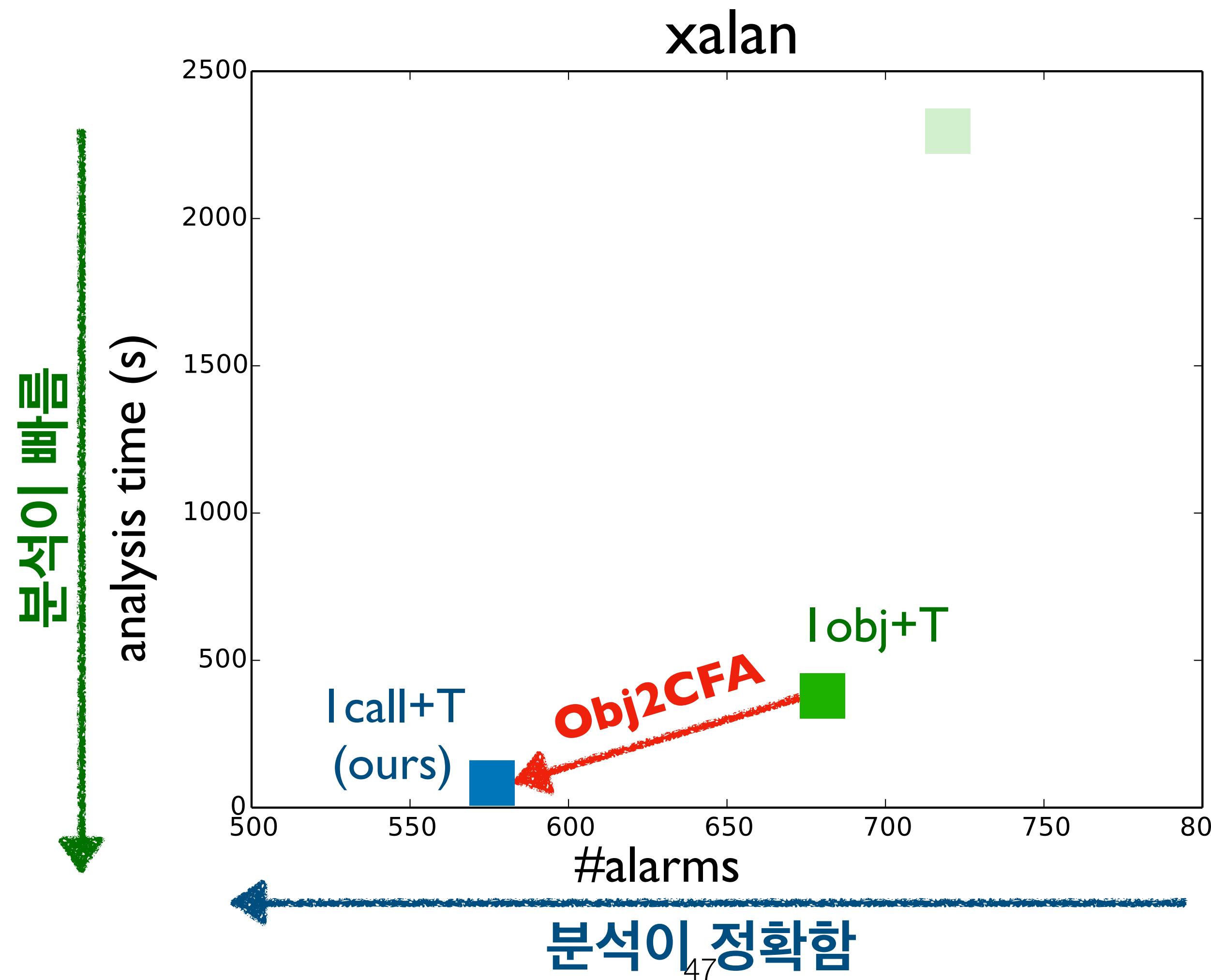
변환기 : Obj2CFA

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



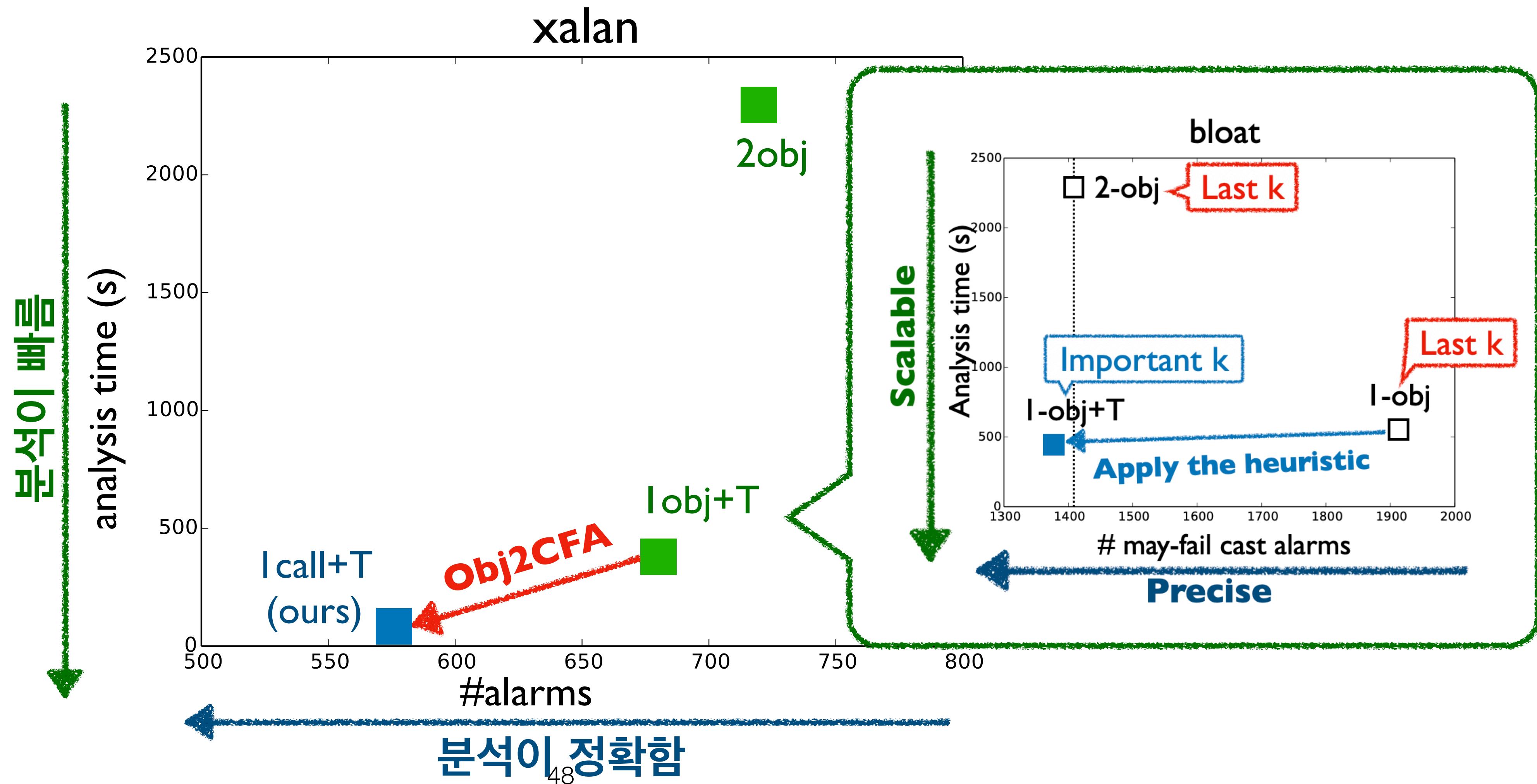
변환기 : Obj2CFA

- **Obj2CFA** 는 주어진 값 기반 요약을 더 정확한 위치 기반 요약으로 바꾸어줌



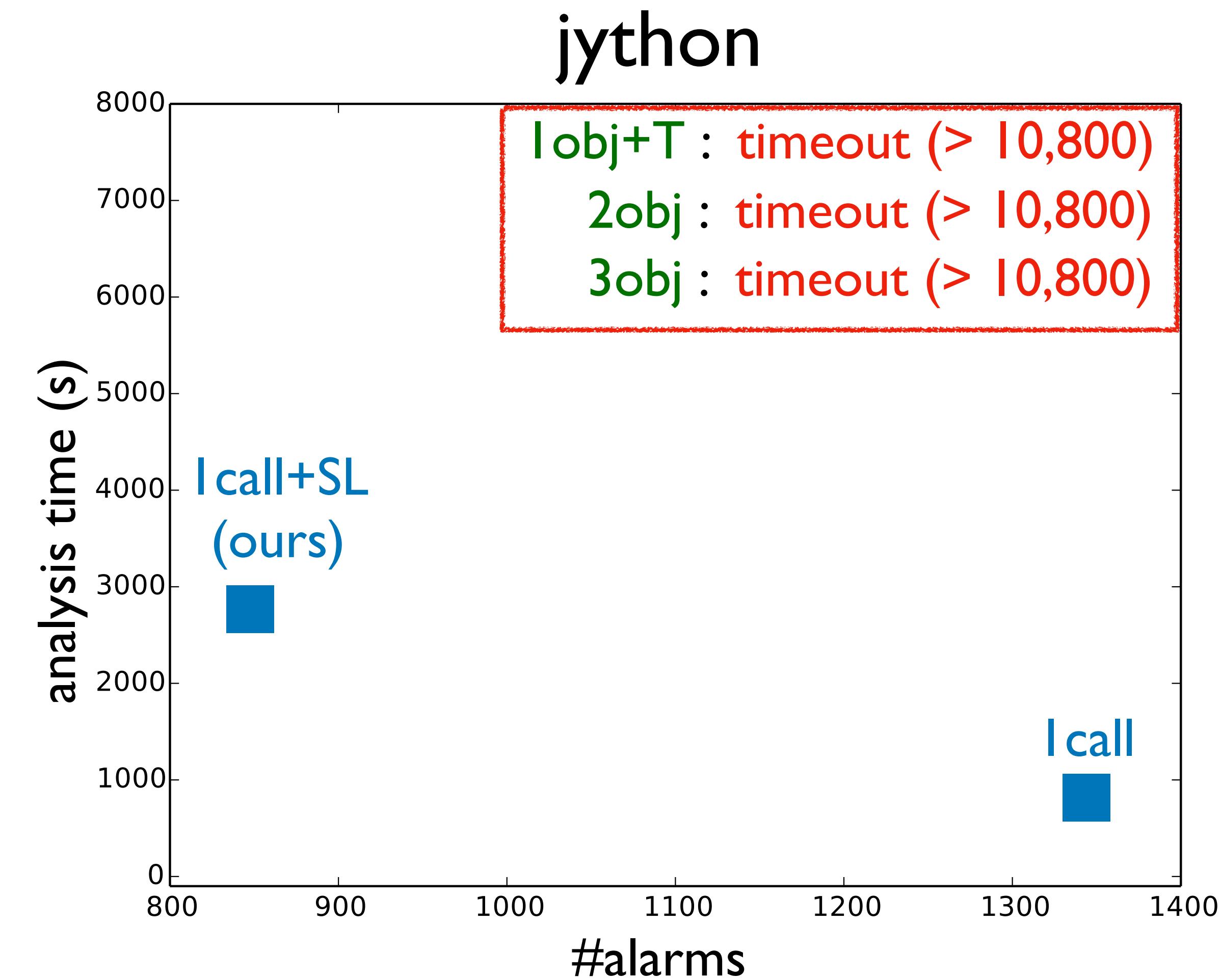
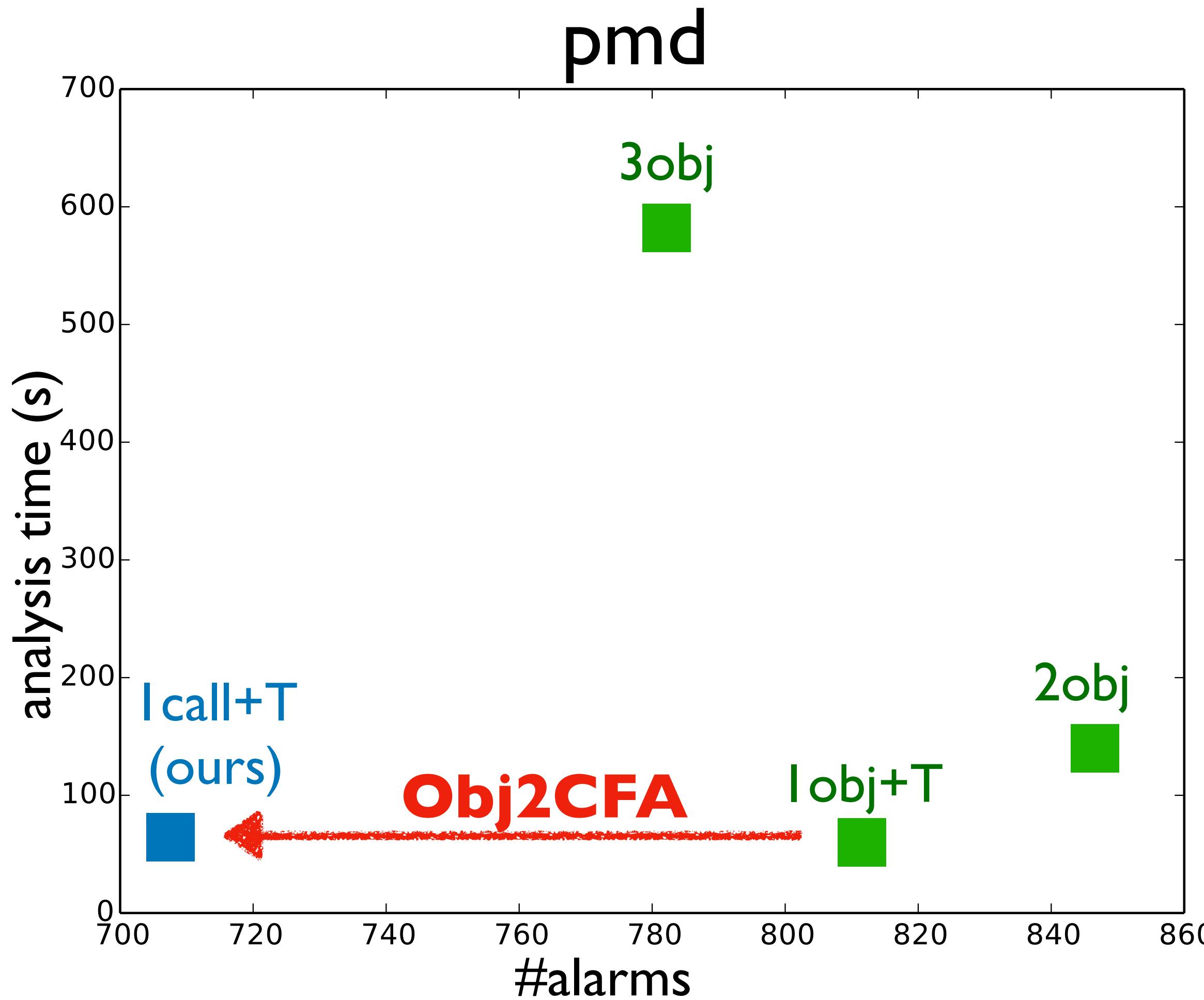
변환기 : Obj2CFA

- **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]

OOPSLA2019
(Rejected)

PLDI 2020
(Rejected)

ICSE 2020
(Rejected)

OOPSLA 2021
(Rejected)

POPL 2022
(Accepted)

논문 출판 후

Call-Site vs. Object Sensitivity

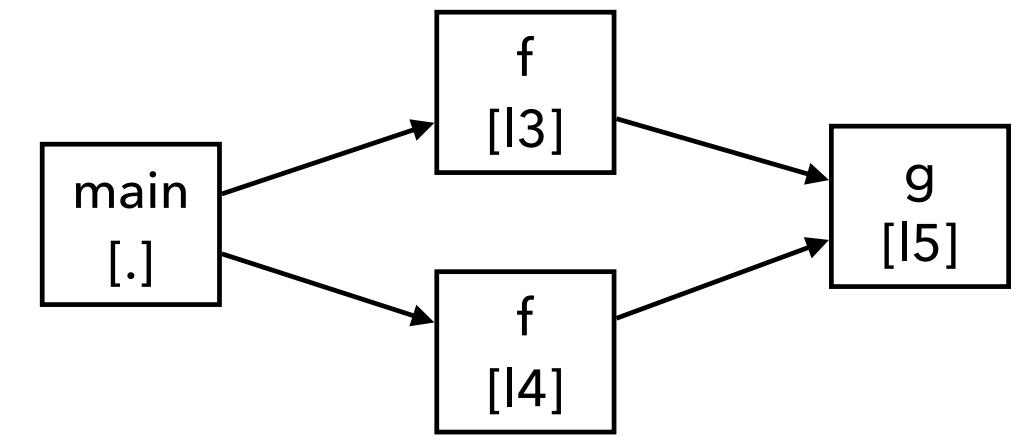
- In theory, their precision is incomparable
- In practice, object sensitivity generally outperforms call-site sensitivity for OO languages (like Java)

Call-site vs. Object Sensitivity

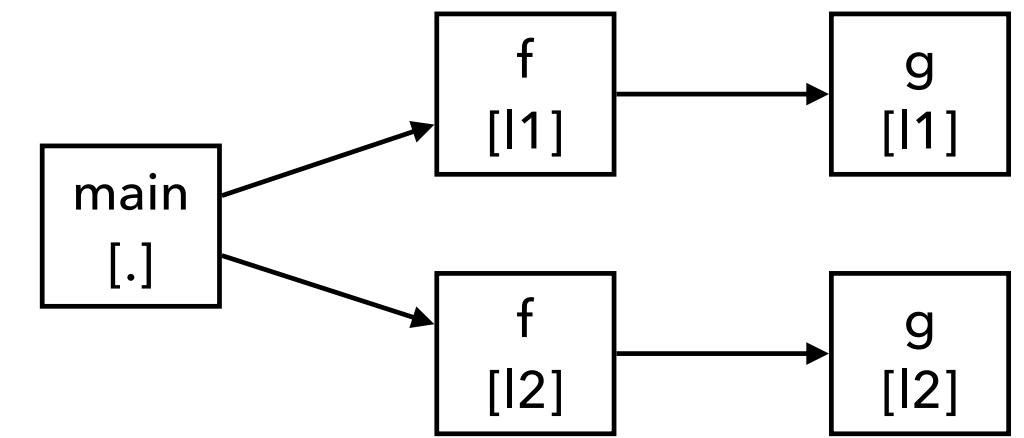
- Typical example that benefits from object sensitivity:

```
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
```



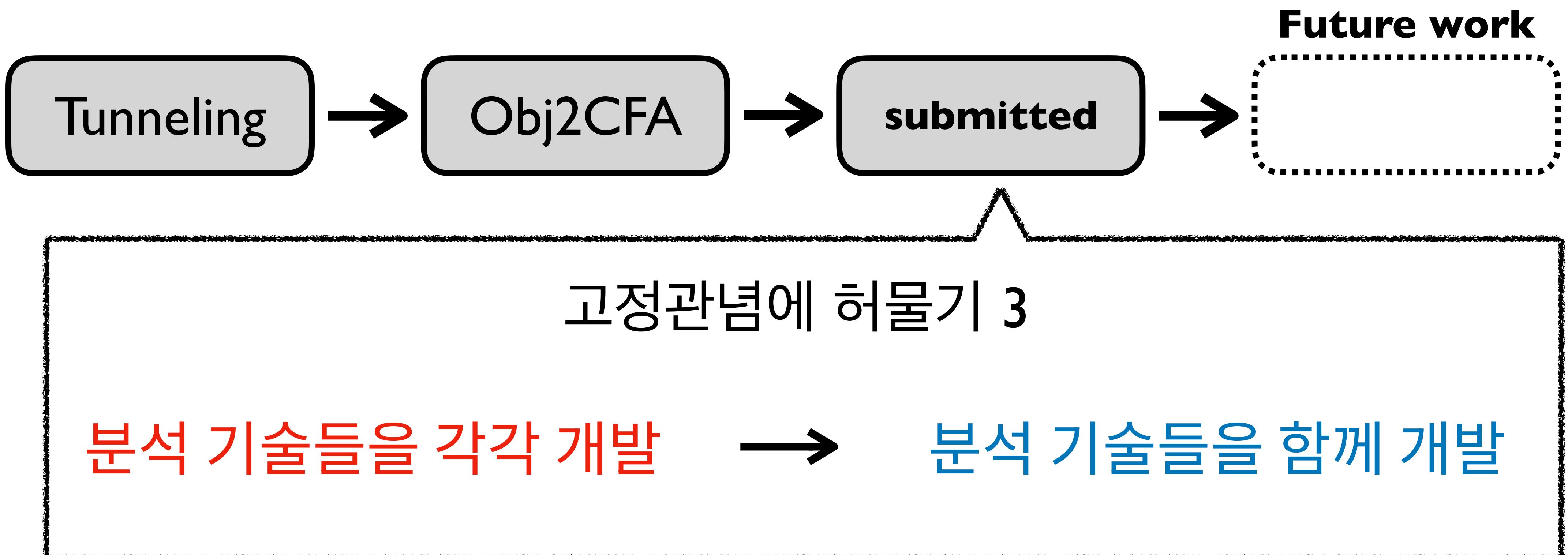
1-call-site sensitivity



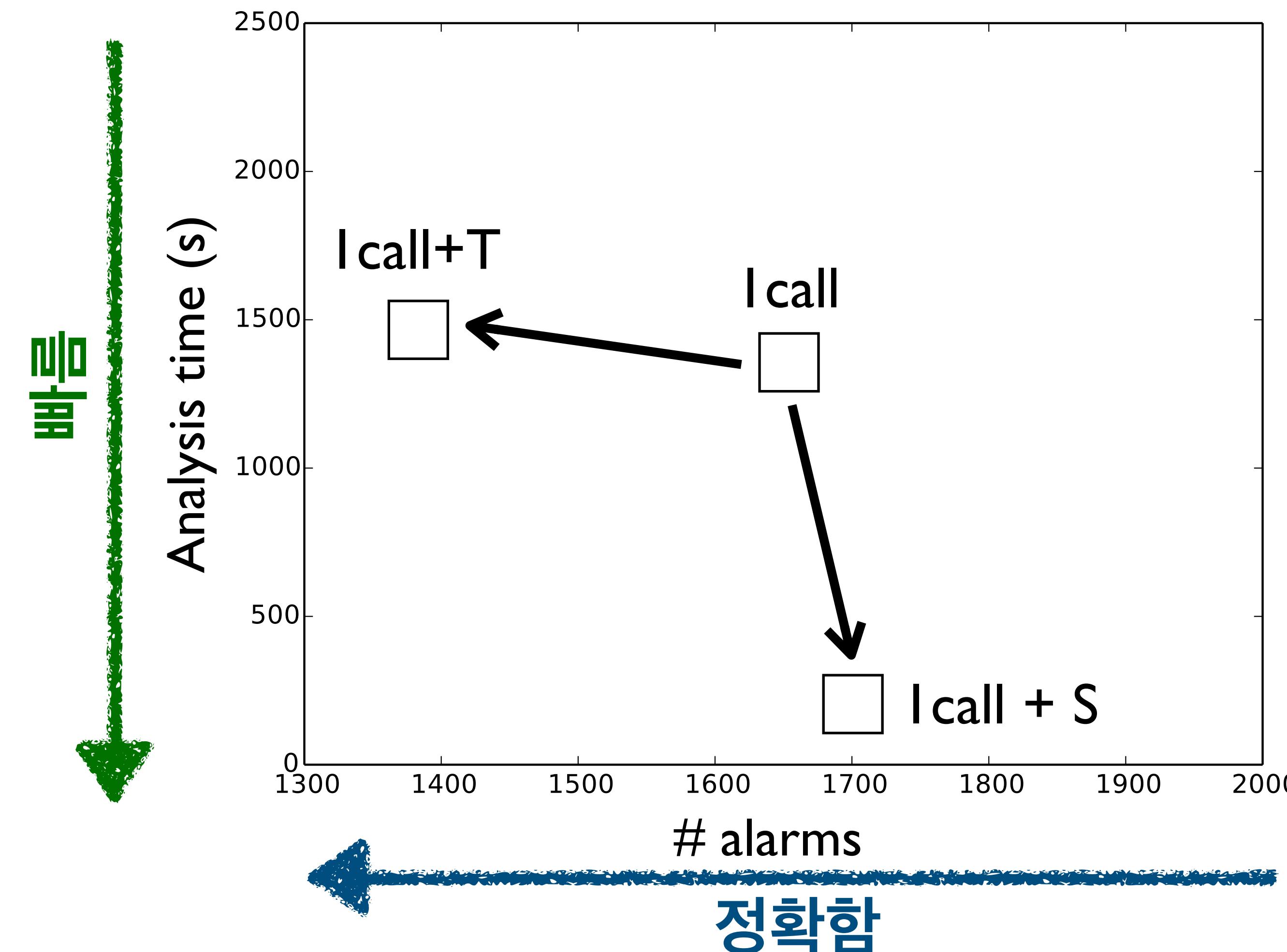
1-object sensitivity

고정관념에 도전하기

- 목표: 주요 k기반 요약 방식을 표준으로 만들기

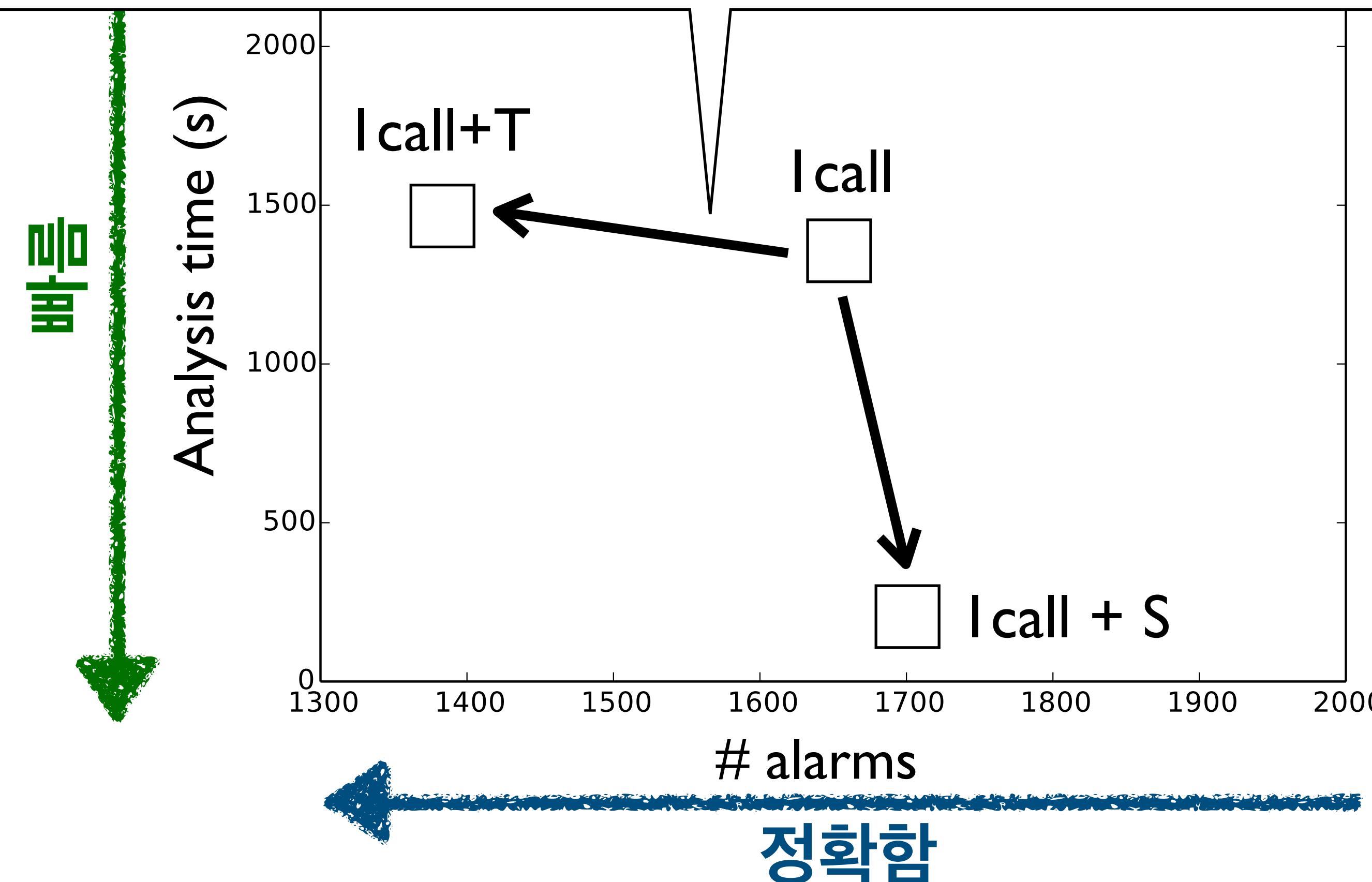


- 호출 환경 배달하기: 속도를 유지한채 정확도를 올려주는 기술
- 적당히 함수 호출 분석 (selective ctx sensitivity): 정확도를 유지한채 속도를 올려주는 기술

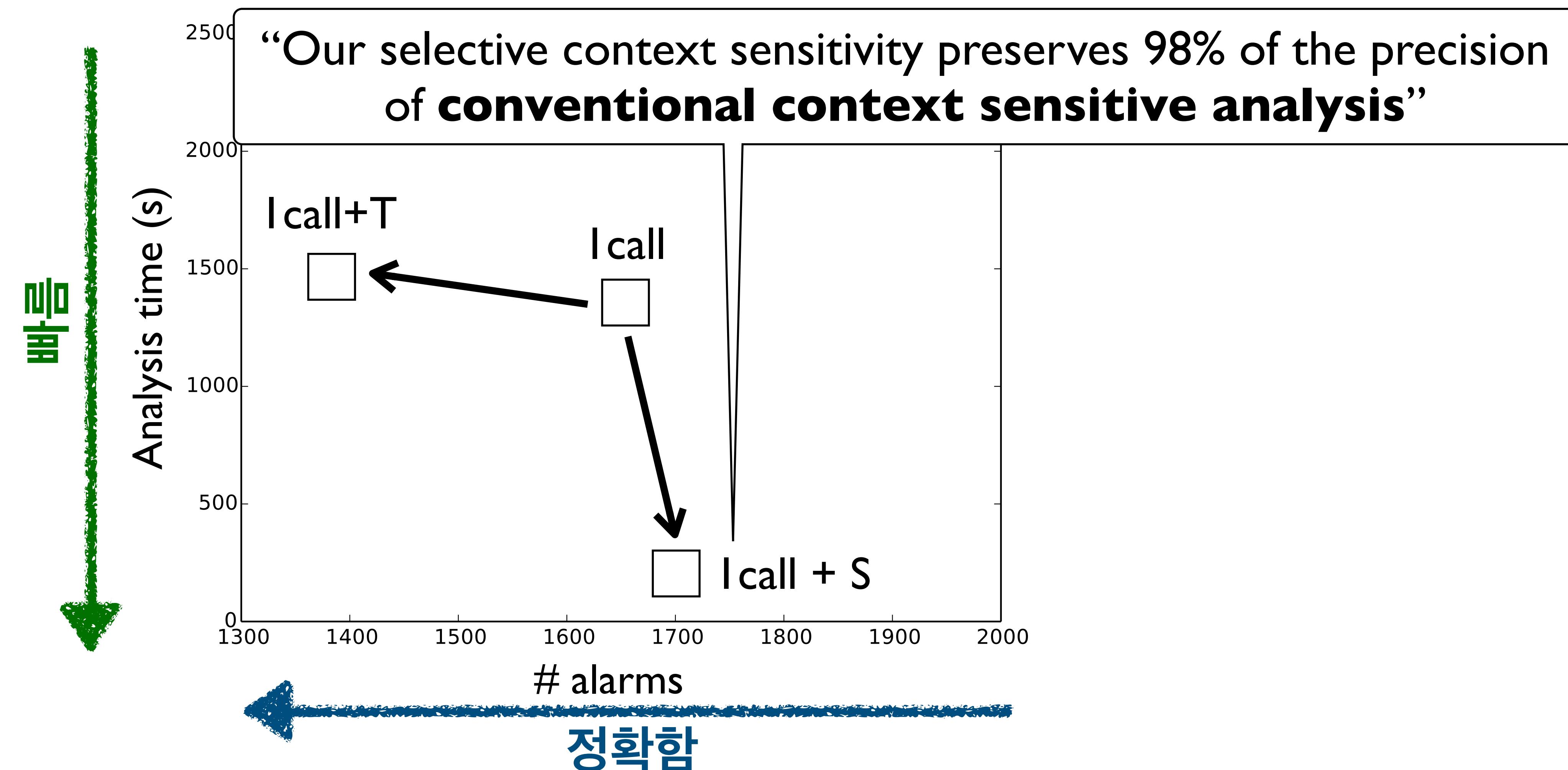


- 호출 환경 배달하기: 속도를 유지한채 정확도를 올려주는 기술

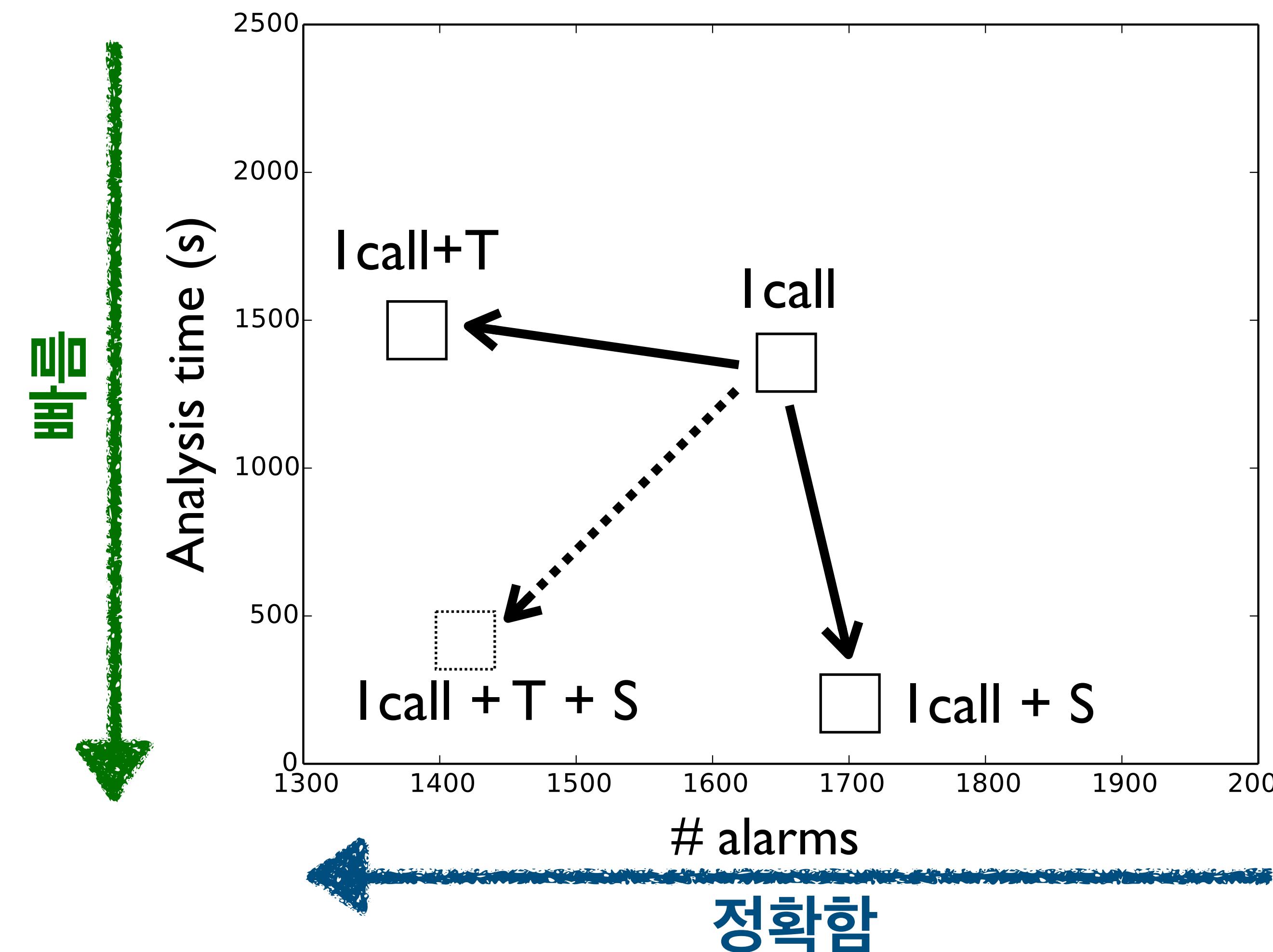
“We generated **Icall+T** by applying context tunneling to **Icall...**”



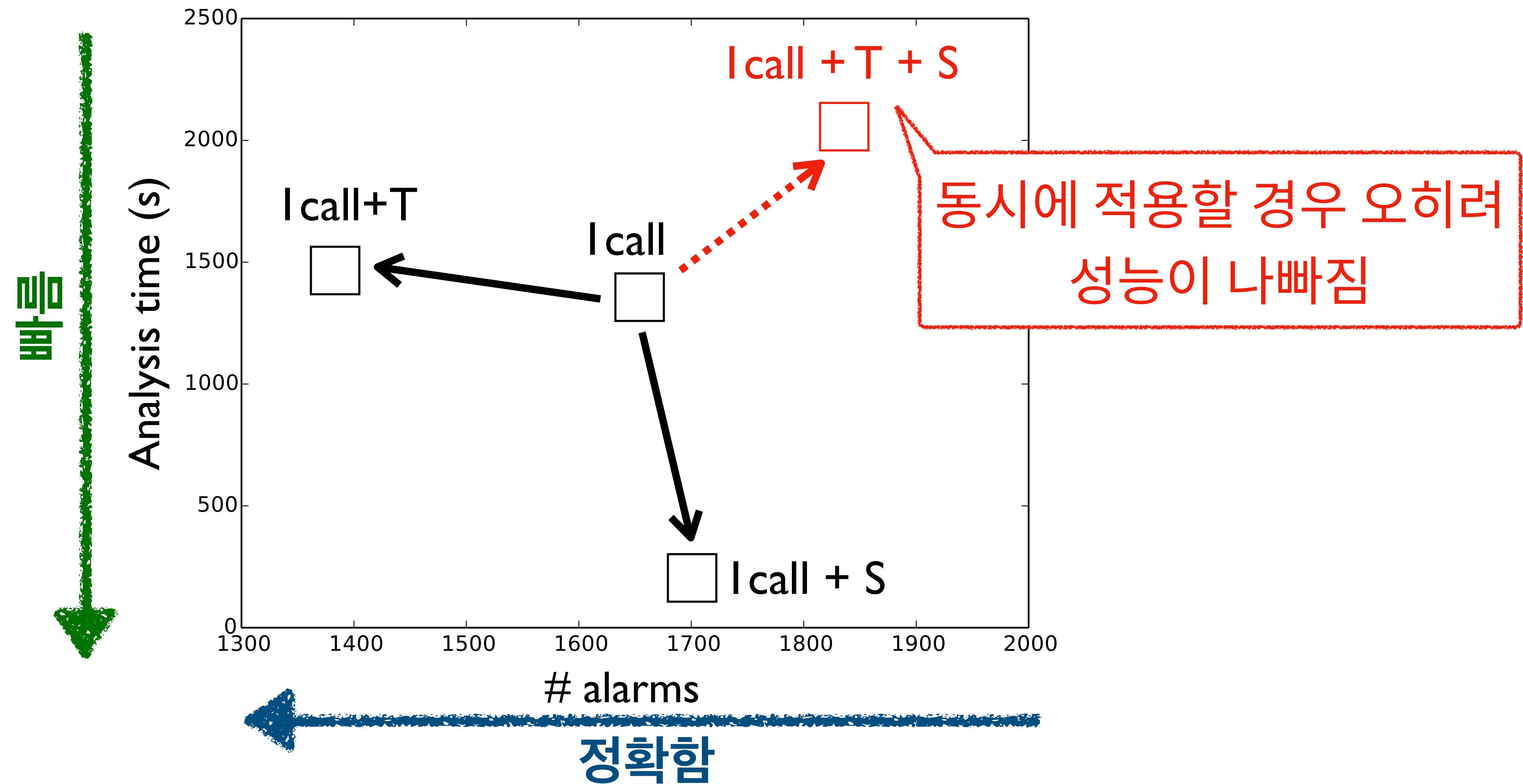
- 적당히 함수 호출 분석: 정확도를 유지한채 속도를 올려주는 기술



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



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

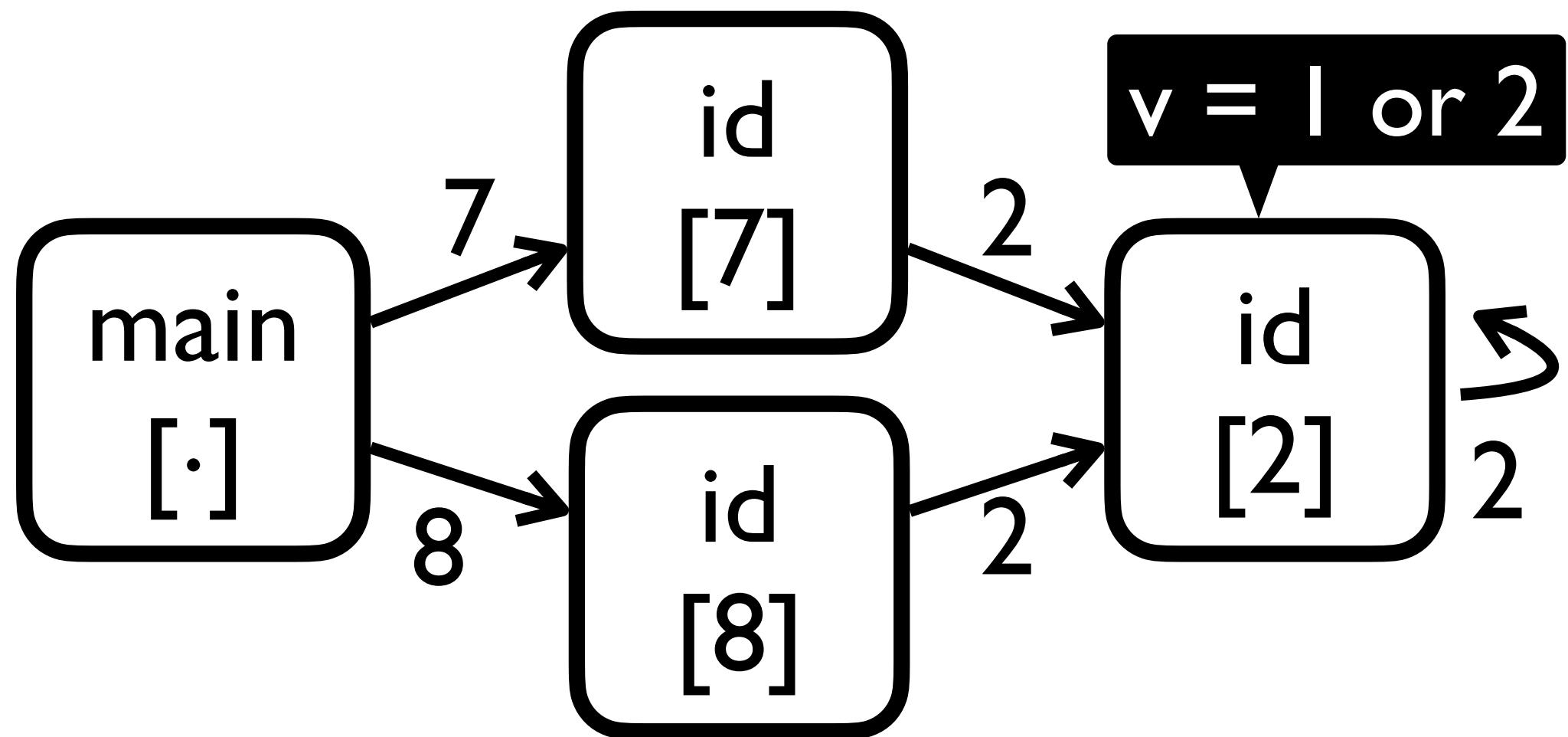


```

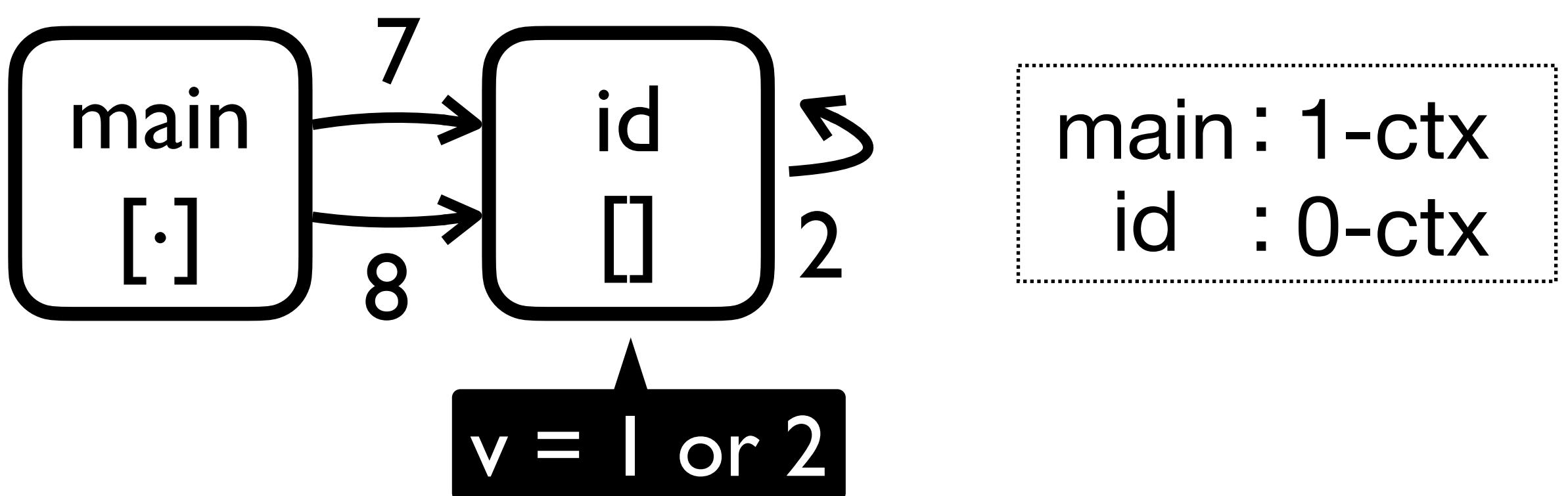
0: id(v, i){
1:   if (i > 0){
2:     return id(v, i-1);}
3:   return v;}
4:
5: main(){
6:   i = input();
7:   v1 = id(1, i); //A
8:   v2 = id(2, i); //B
9:   assert (v1 != v2); //query
10: }

```

예제 프로그램



I-요소 기반 함수 호출 요약



적당히 I-요소 기반 함수 호출 요약

```

0: id(v, i){
1:   if (i > 0){
2:     return id(v, i-1);}
3:   return v;

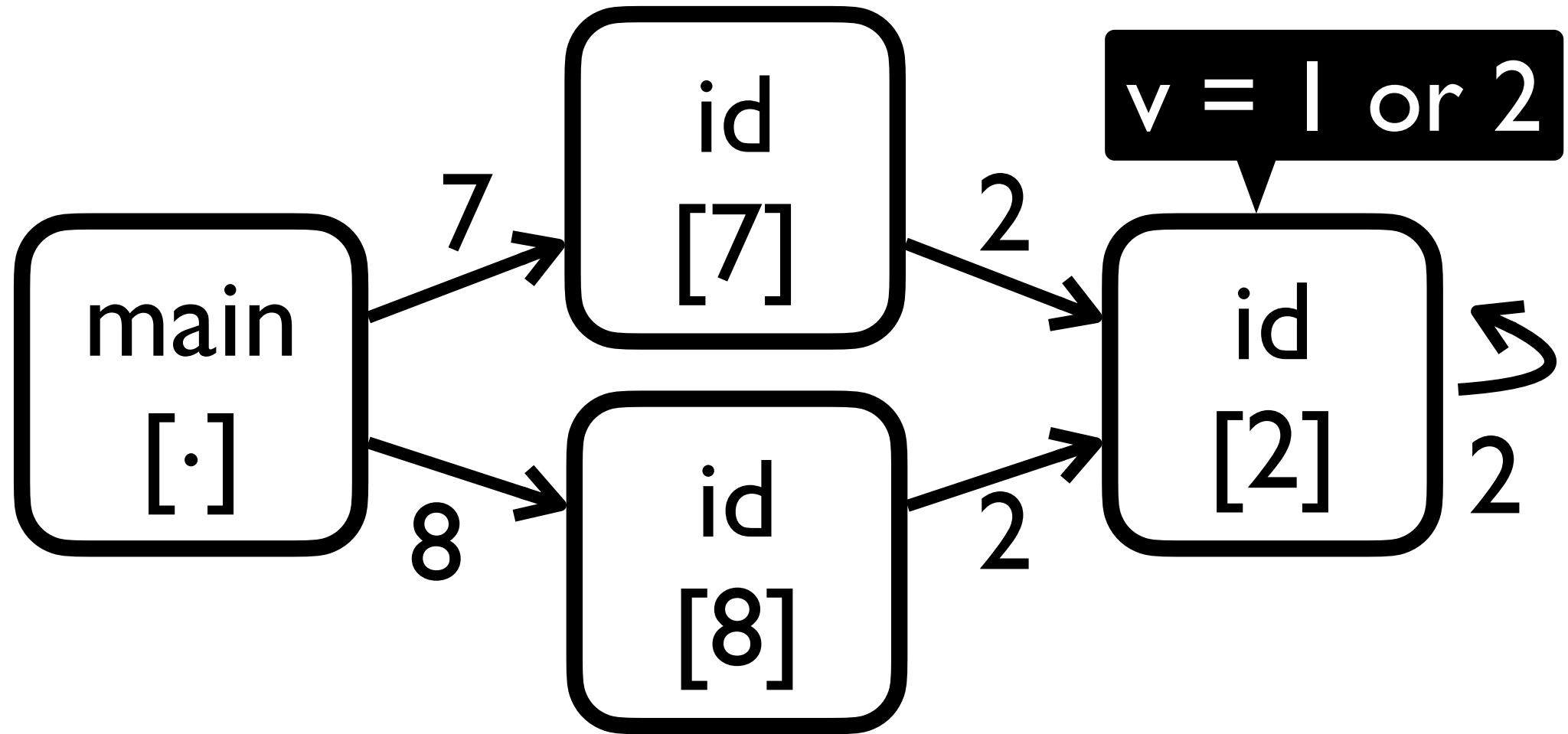
```

```

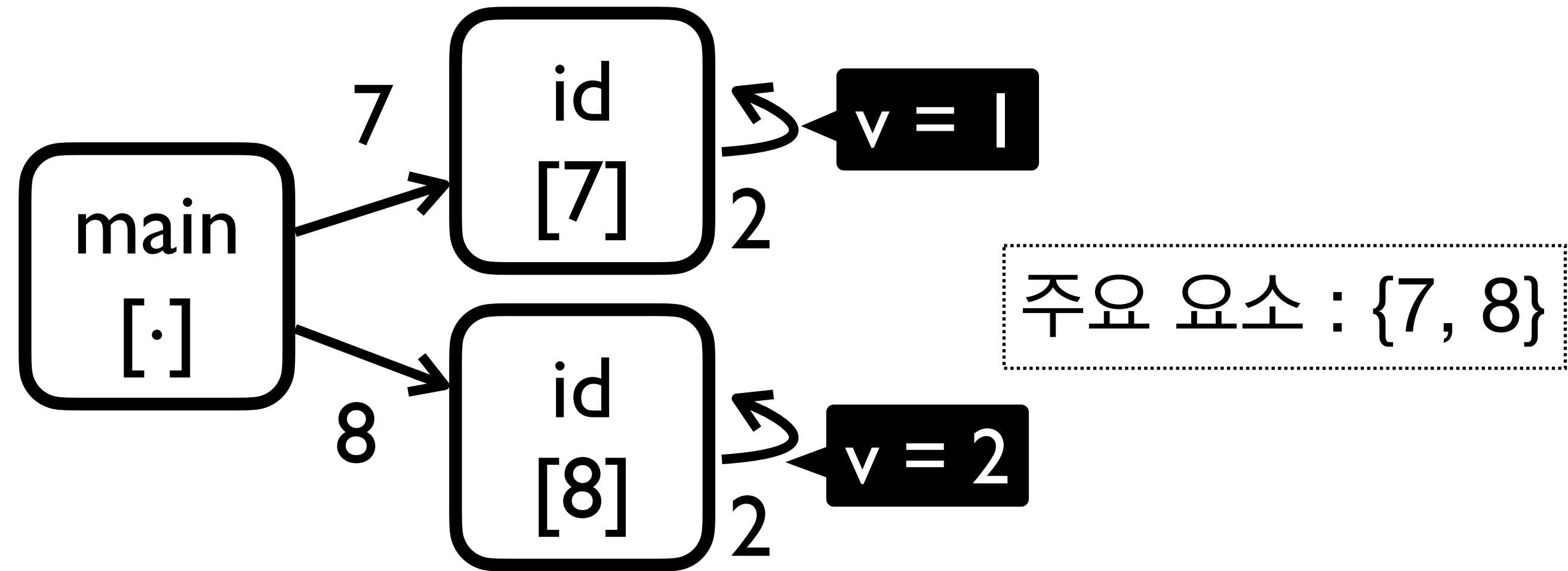
4:
5: main(){
6:   i = input();
7:   v1 = id(1, i); //A
8:   v2 = id(2, i); //B
9:   assert (v1 != v2); //query
10: }

```

예제 프로그램



I-요소 기반 함수 호출 요약



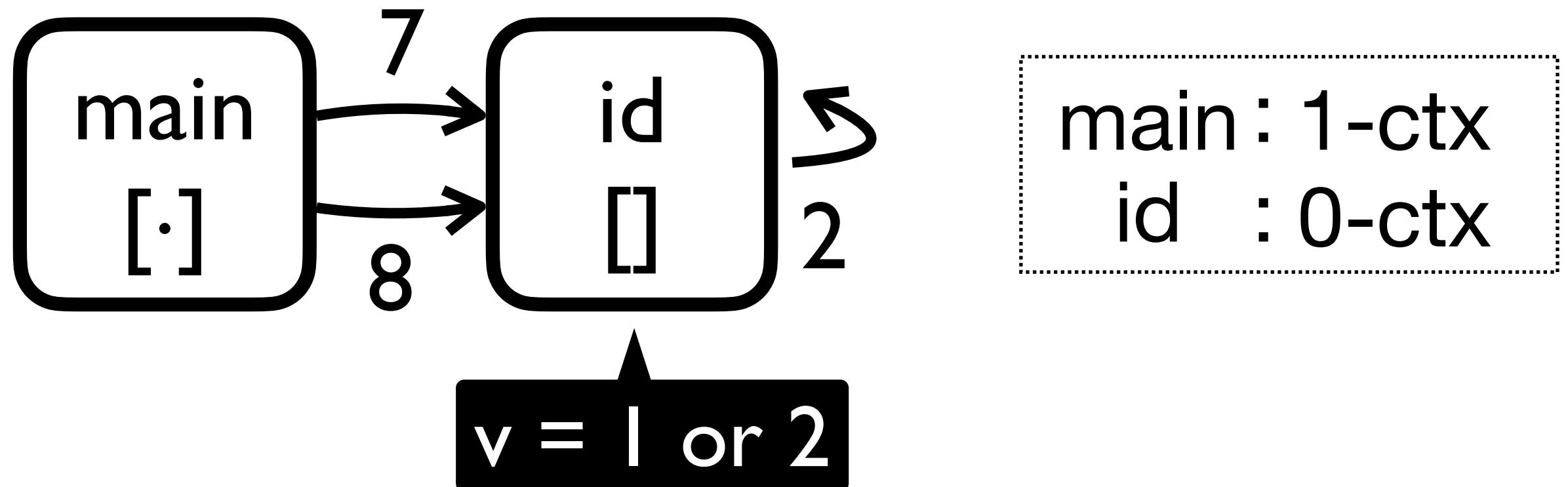
주요 I-요소 기반 함수 호출 요약

```

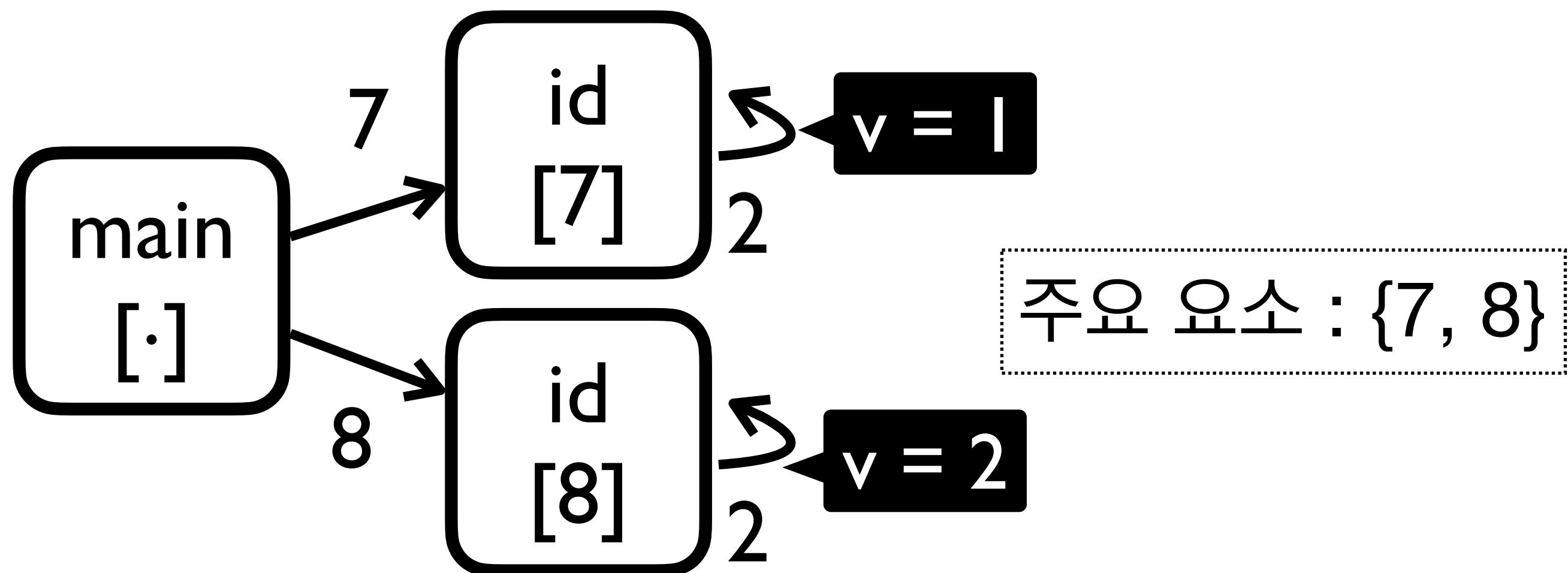
0: id(v, i){
1:   if (i > 0){
2:     return id(v, i-1);
3:   return v;
4:
5: main(){
6:   i = input();
7:   v1 = id(1, i); //A
8:   v2 = id(2, i); //B
9:   assert (v1 != v2); //query
10: }

```

예제 프로그램



적당히 I-요소 기반 함수 호출 요약



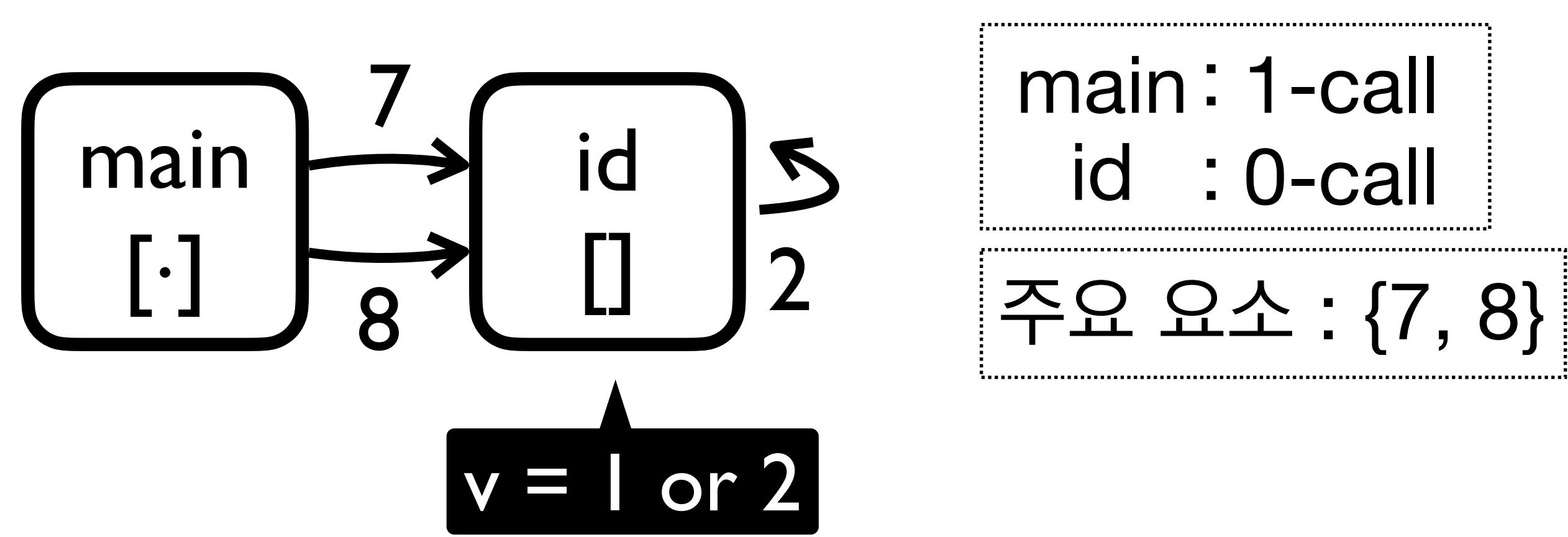
주요 I-요소 기반 함수 호출 요약

```

0: id(v, i){
1:   if (i > 0){
2:     return id(v, i-1);}
3:   return v;}
4:
5: main(){
6:   i = input();
7:   v1 = id(1, i); //A
8:   v2 = id(2, i); //B
9:   assert (v1 != v2); //query
10: }

```

예제 프로그램

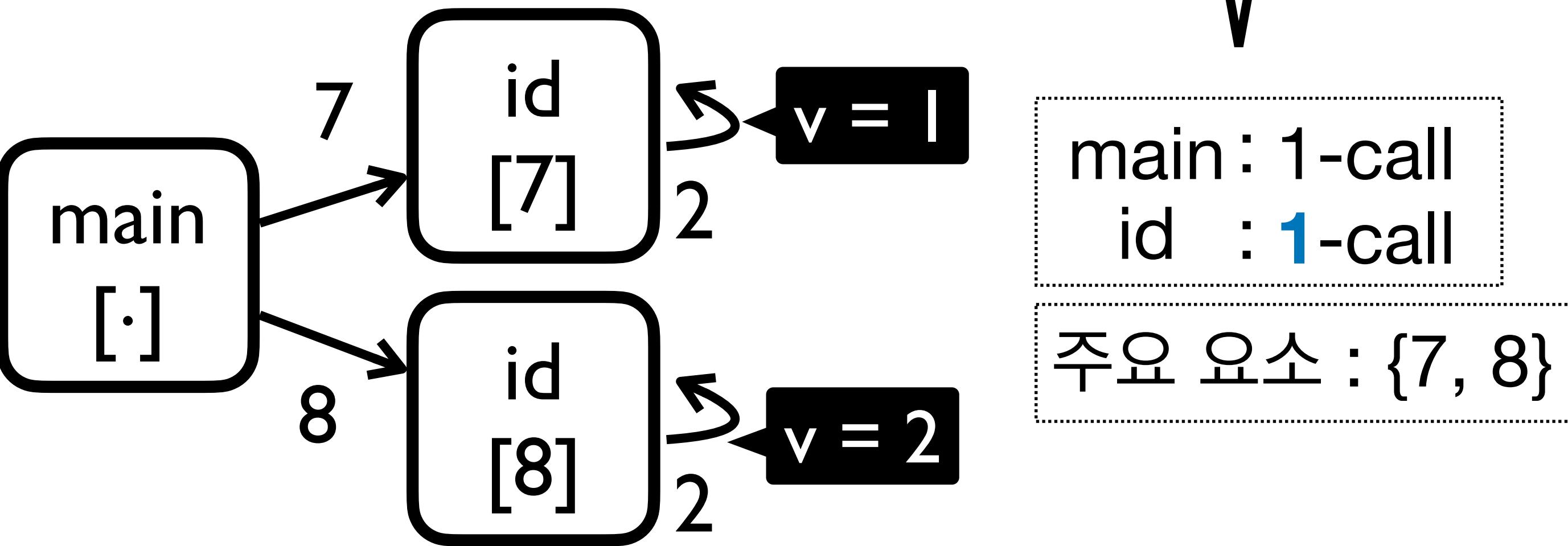


적당히 주요 I-요소 기반 함수 호출 요약

같이 사용될 것을 고려해 분류해야 함

```
0: id(v, i){  
1:   if (i > 0){  
2:     return id(v, i-1);}  
3:   return v;}  
4:  
5: main(){  
6:   i = input();  
7:   v1 = id(1, i); //A  
8:   v2 = id(2, i); //B  
9:   assert (v1 != v2); //query  
10: }
```

예제 프로그램

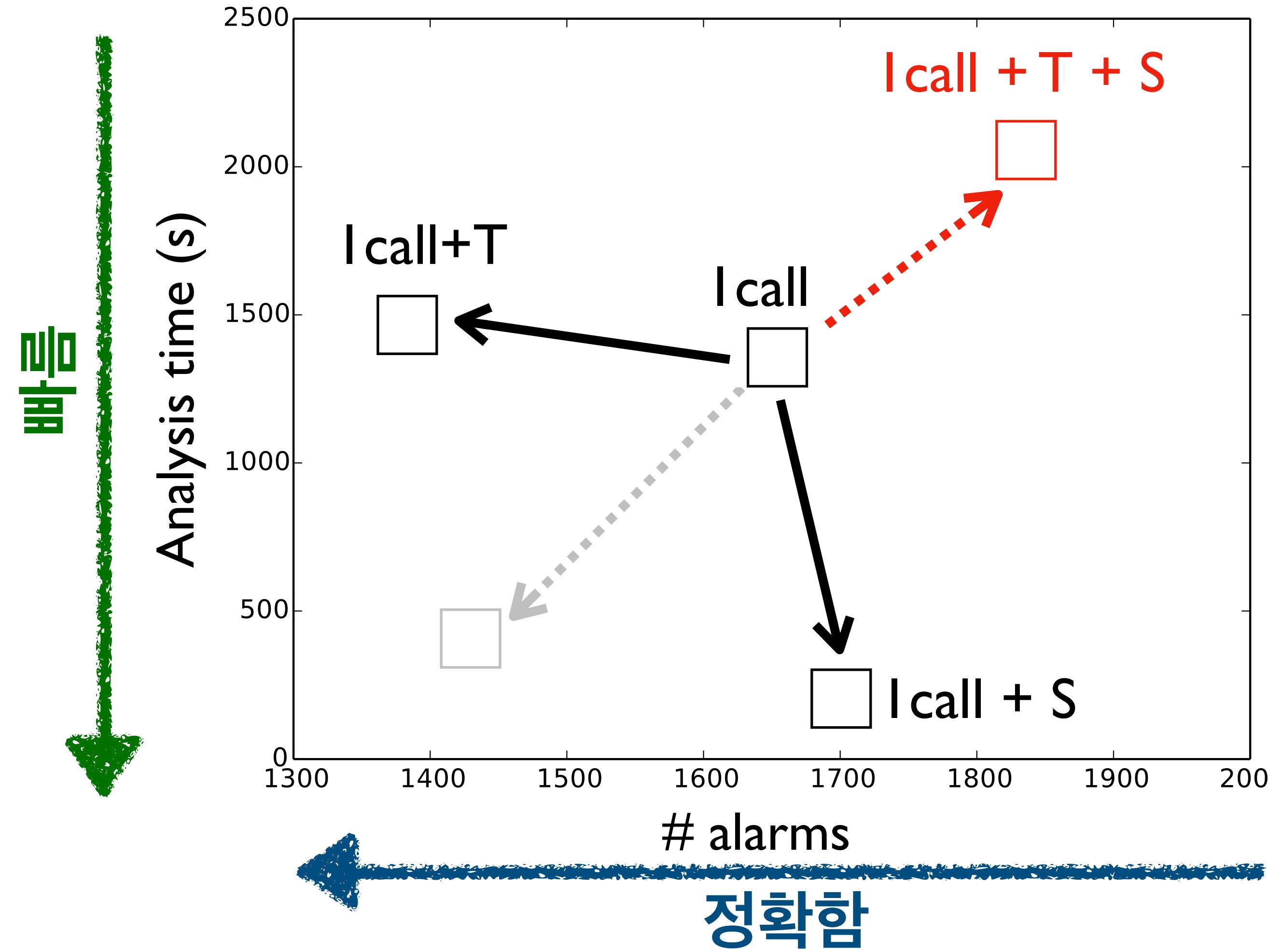


적당히 주요 I-요소 기반 함수 호출 요약

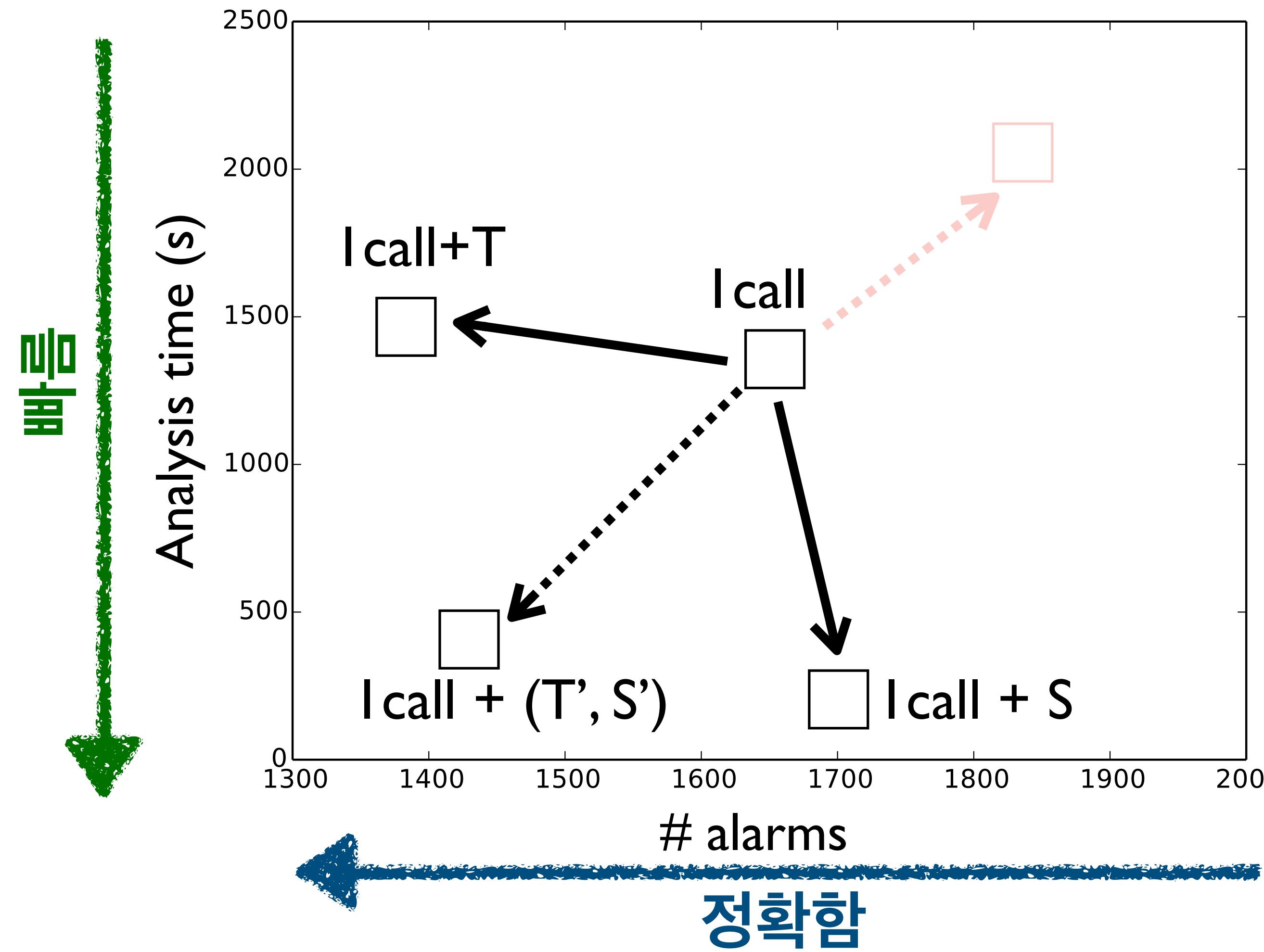
main: 1-call
id : 1-call

주요 요소 : {7, 8}

- 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:

Anonymous Author(s). 2018. Marriage of Context Tunneling and Selective Context Sensitivity in Pointer Analysis. *J. ACM* 37, 4, Article 111 (August 2018), 28 pages. <https://doi.org/XXXXXX.XXXXXXX>

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/XXXXXX.XXXXXXX>

분석 기술들을 각각 개발

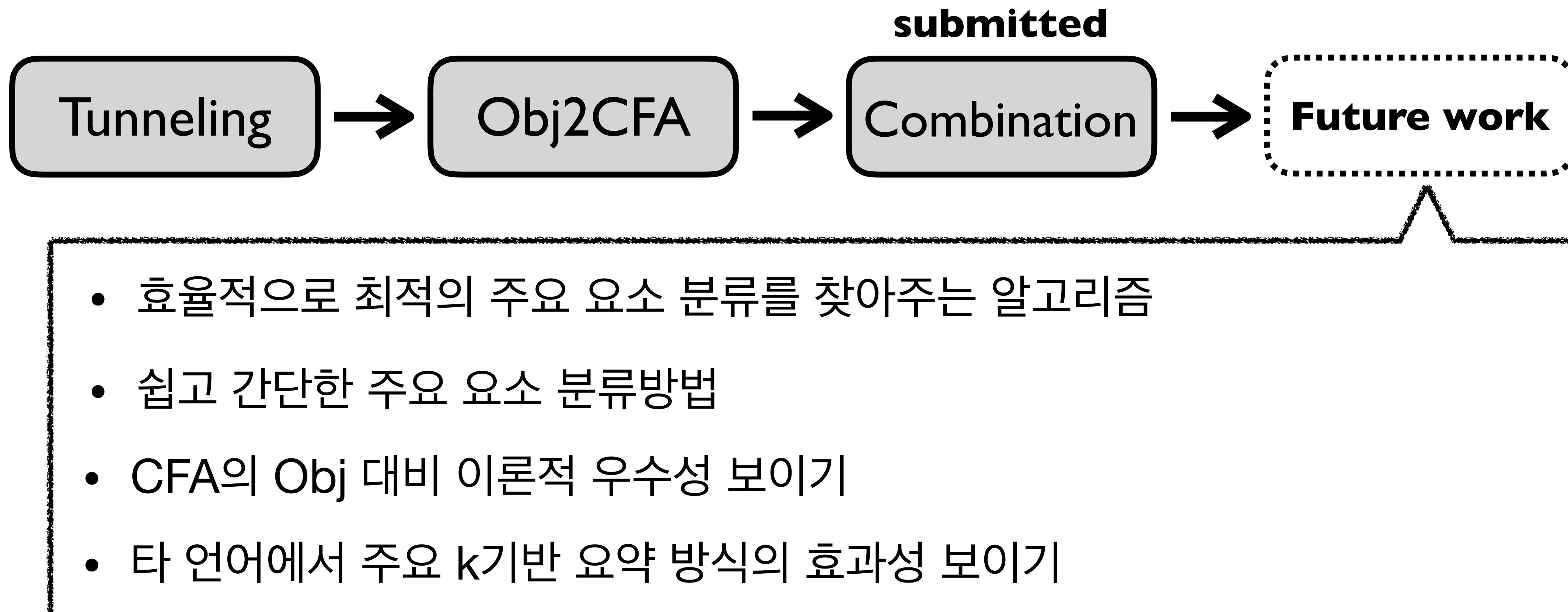
- We identify a fundamental issue in the current trend of developing context sensitivity techniques in pointer analysis and present a way to efficiently address it.

분석 기술들을 동시에 개발하는 법

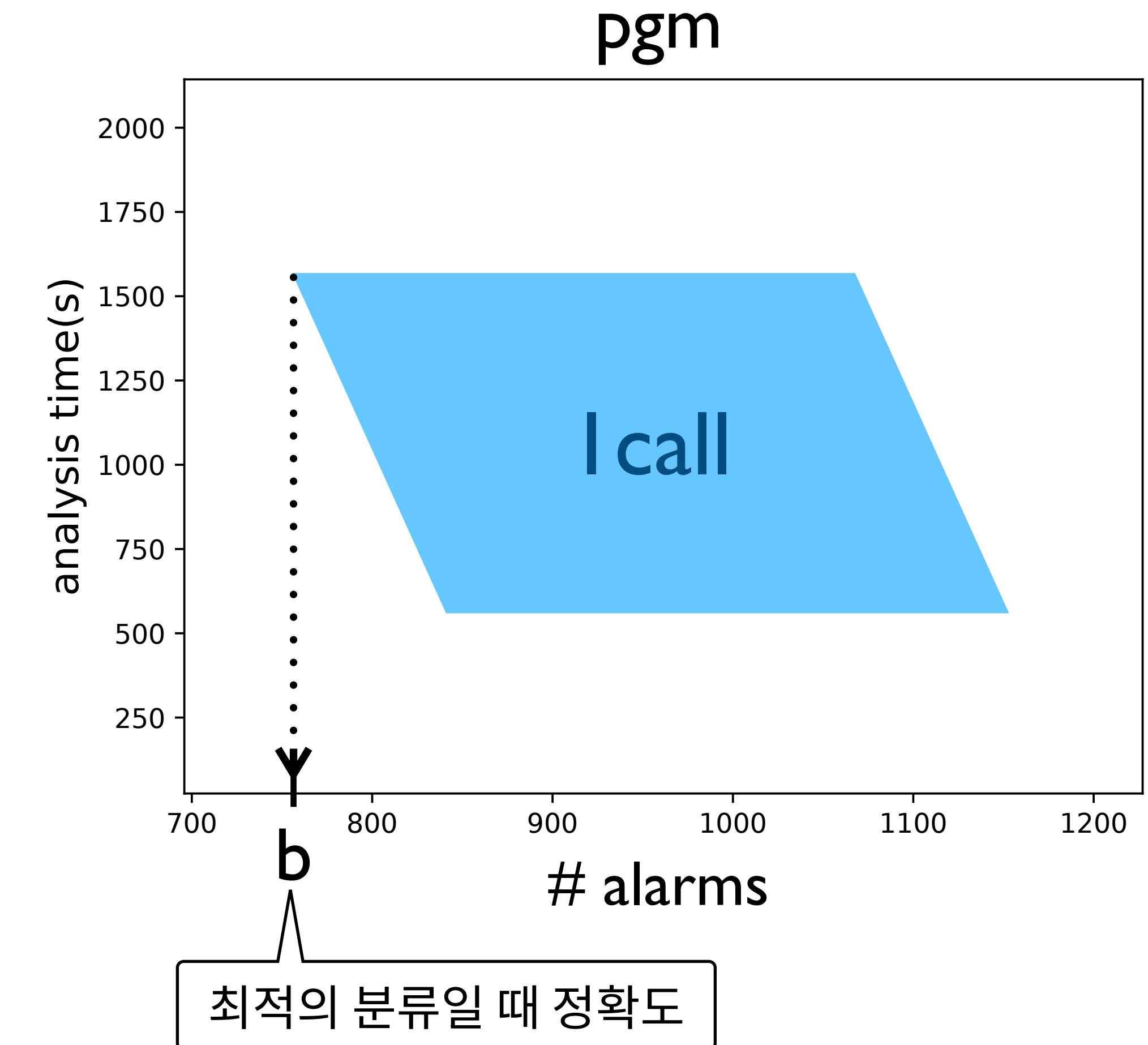
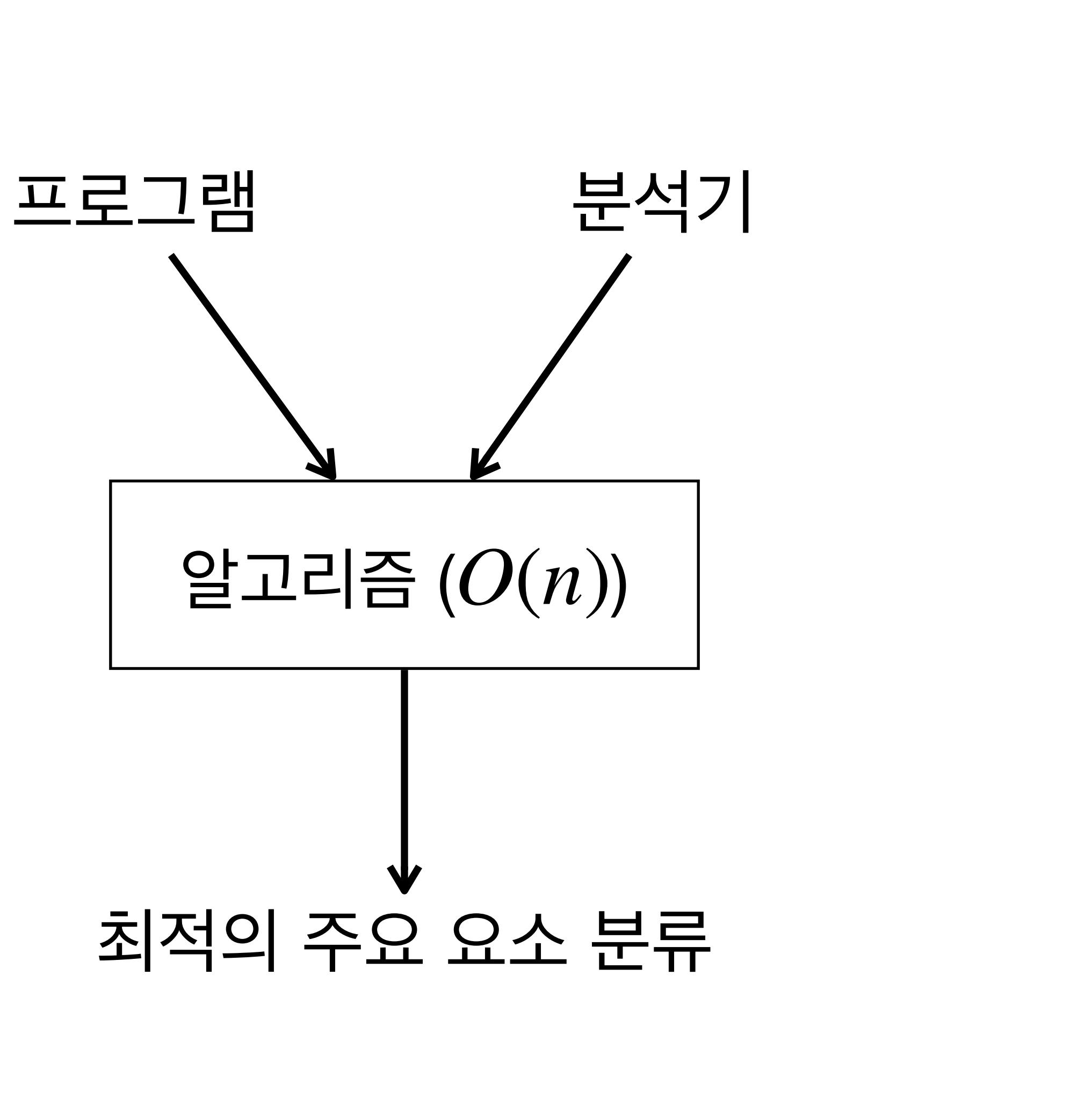
- We present a framework that significantly reduces the complexity of developing an effective combination of the two techniques

컨텍스트 터널링: 고정관념에 도전하기

- 목표: 주요 k기반 요약 방식을 표준으로 만들기

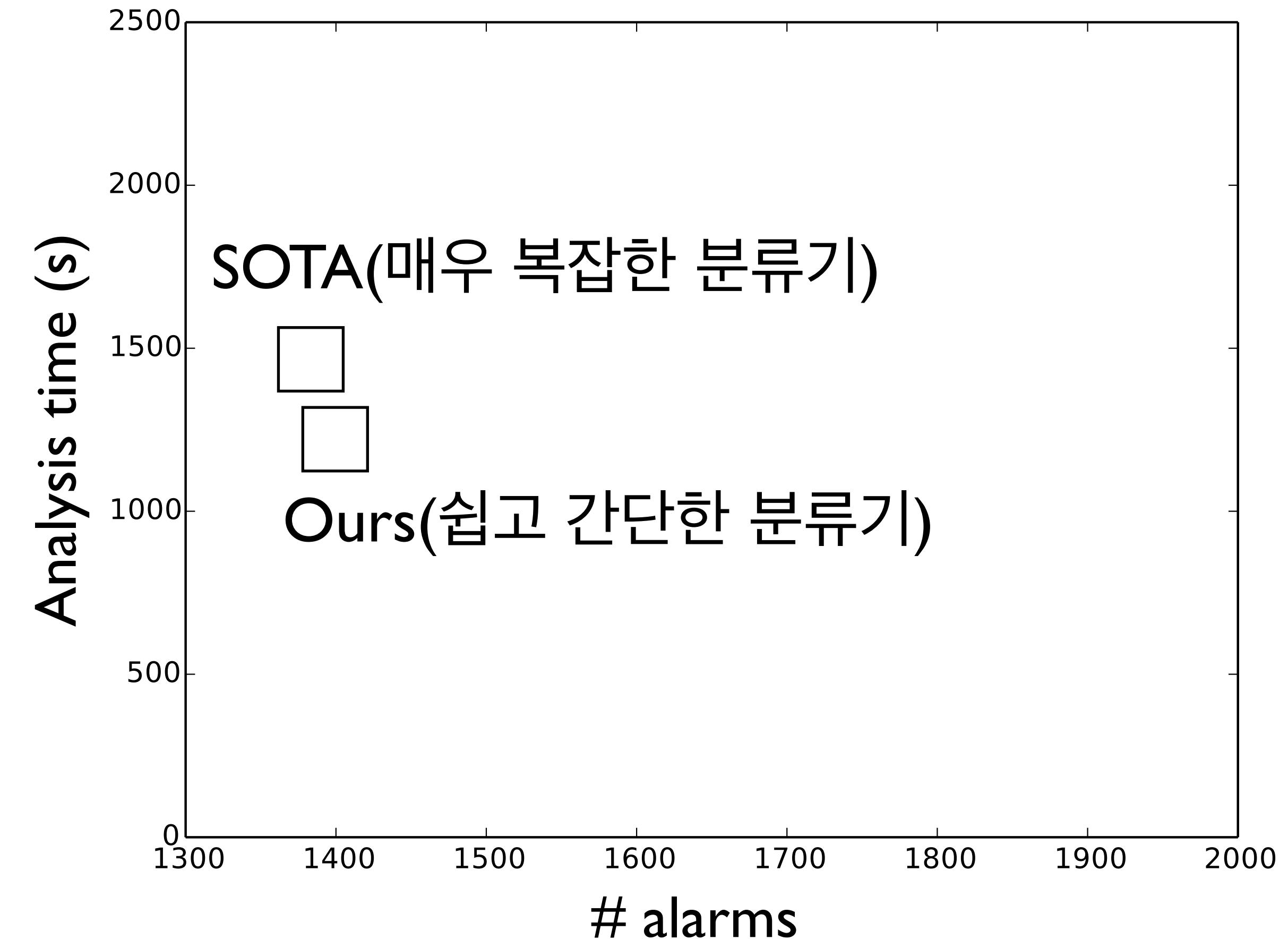


후속 연구 1: 최적의 주요 요소 분류 찾기 알고리즘



후속 연구 2: 쉽고 간단한 주요 요소 분류 방법

```
classifier(e):  
if ( ?? (쉽고 간단한 조건) ):  
    return true //주요 요소임  
  
else:  
    return false //부수적 요소임
```



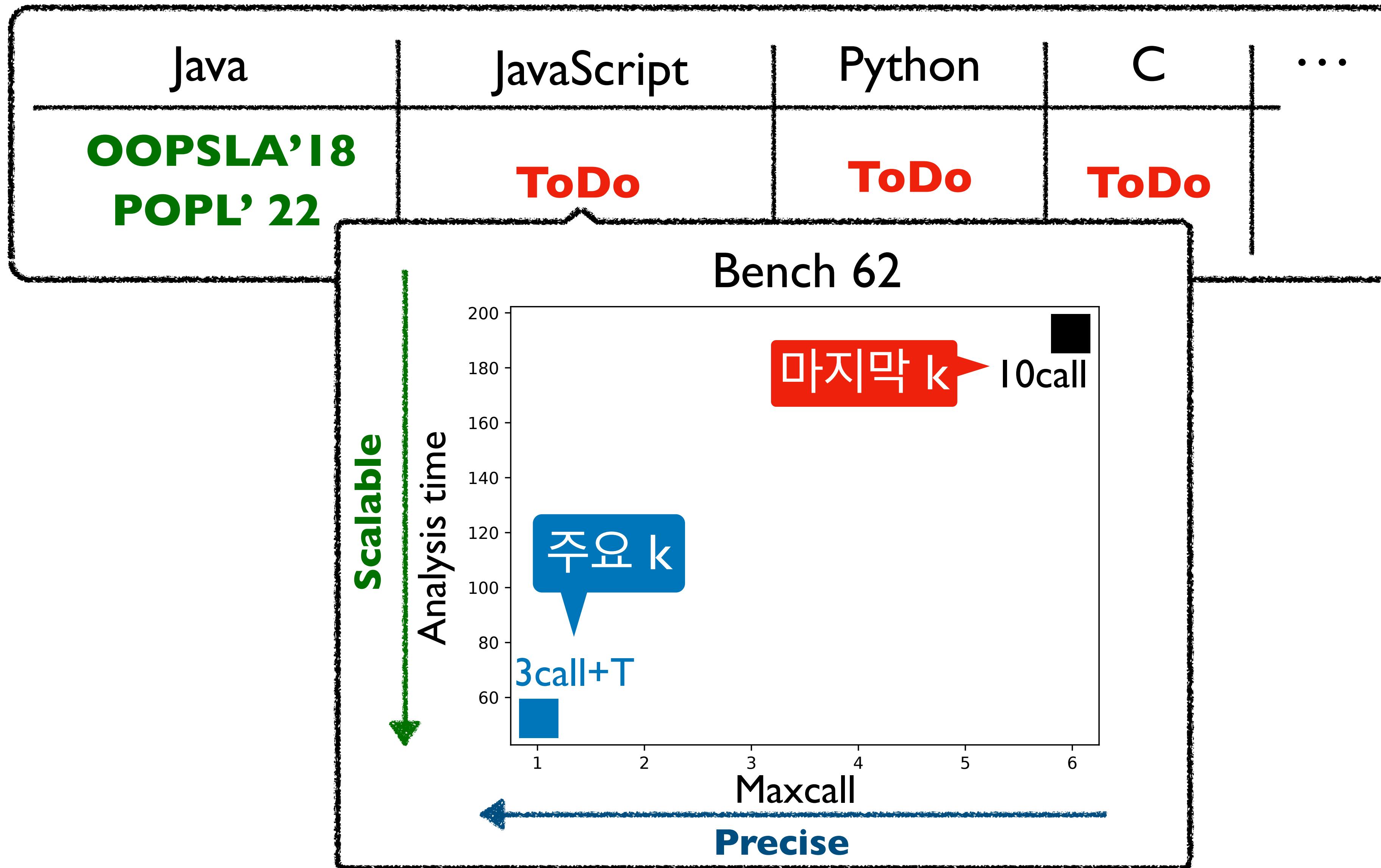
후속 연구 3: CFA의 Obj 대비 우수성을 이론적으로 보이기

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 it is always possible to simulate object sensitivity via call-site sensitivity:

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

후속 연구 4: 타 언어에 적용하기



마무리: 고정관념에 도전하기

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>
```

질문:

I-call-site sensitivity로 정확하게 할 순 없나?



Tunneling

Obj2CFA

submitted
Combination

Future work

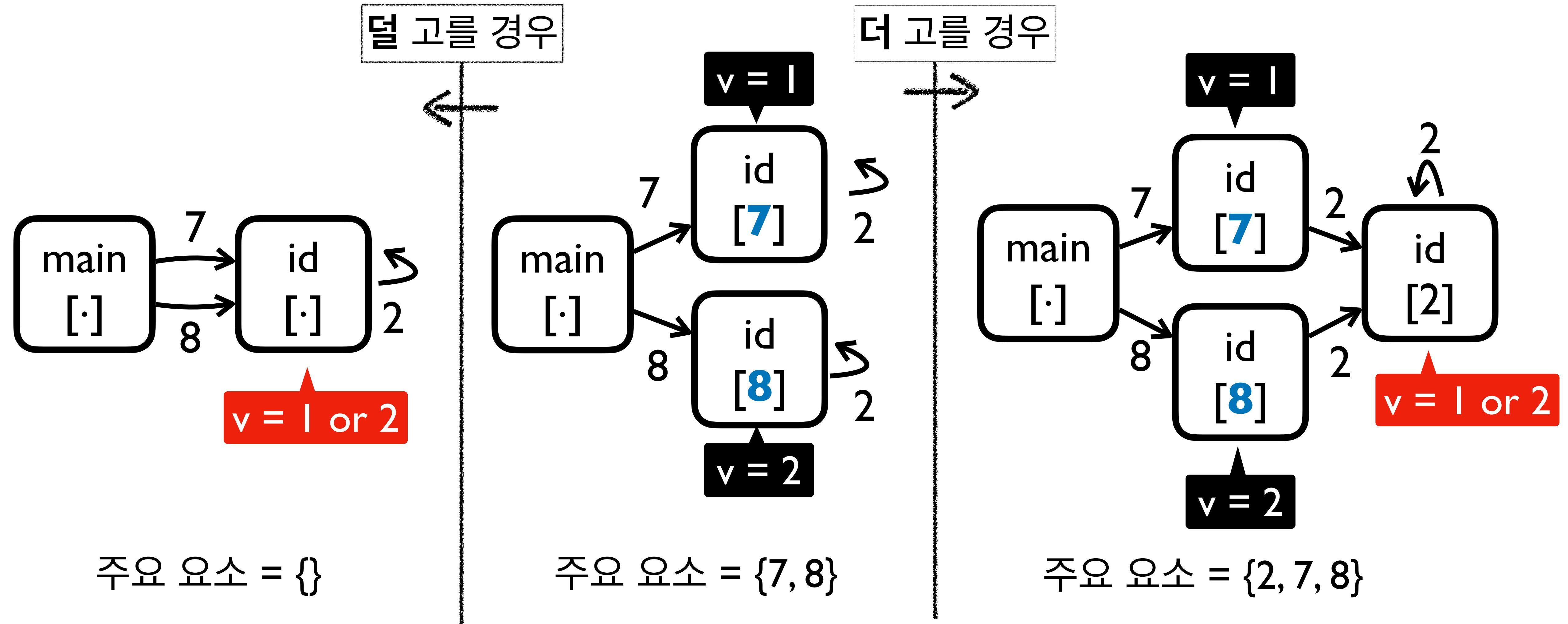
마지막 k → 주요 k

Obj → CFA

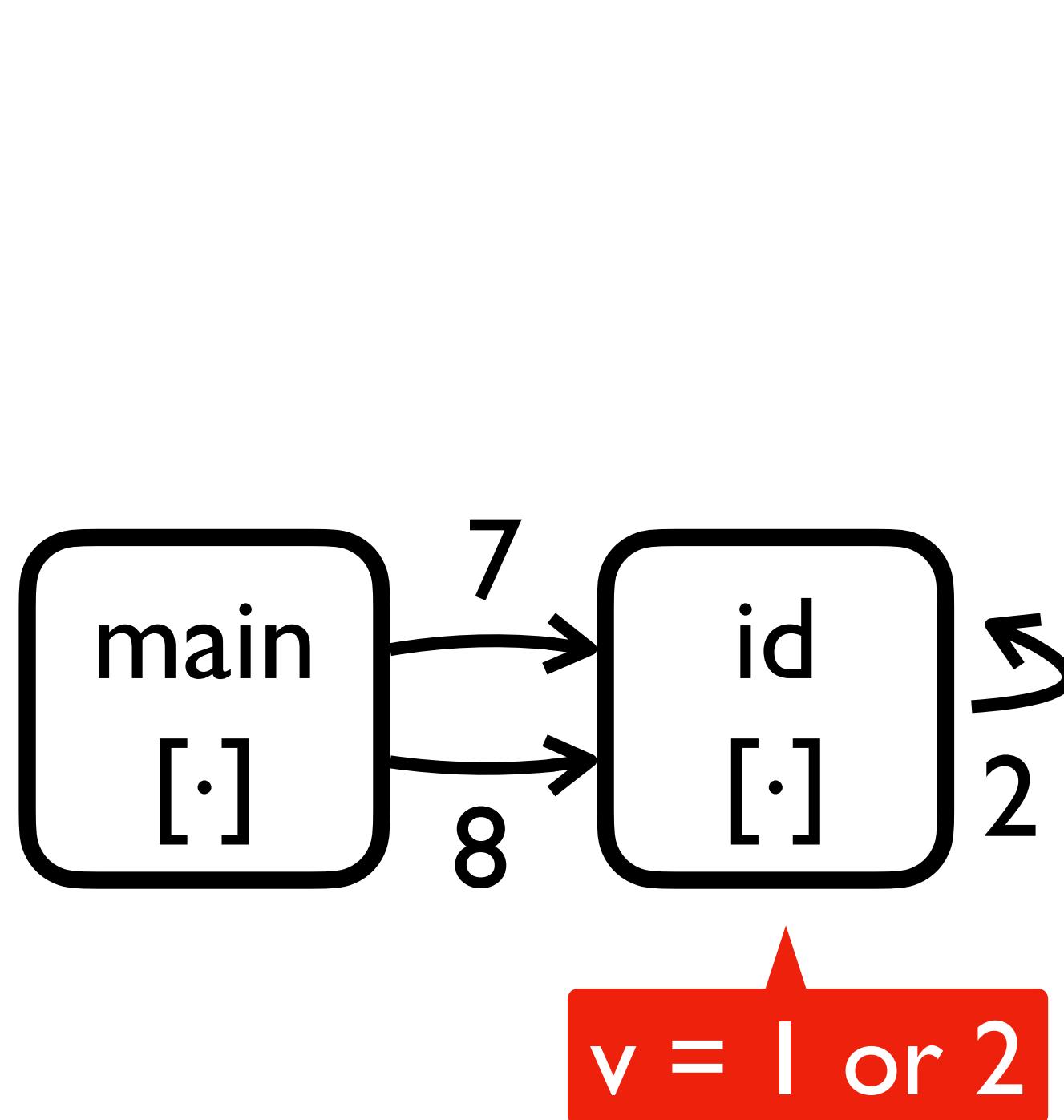
따로 개발 → 동시에 개발

Back up

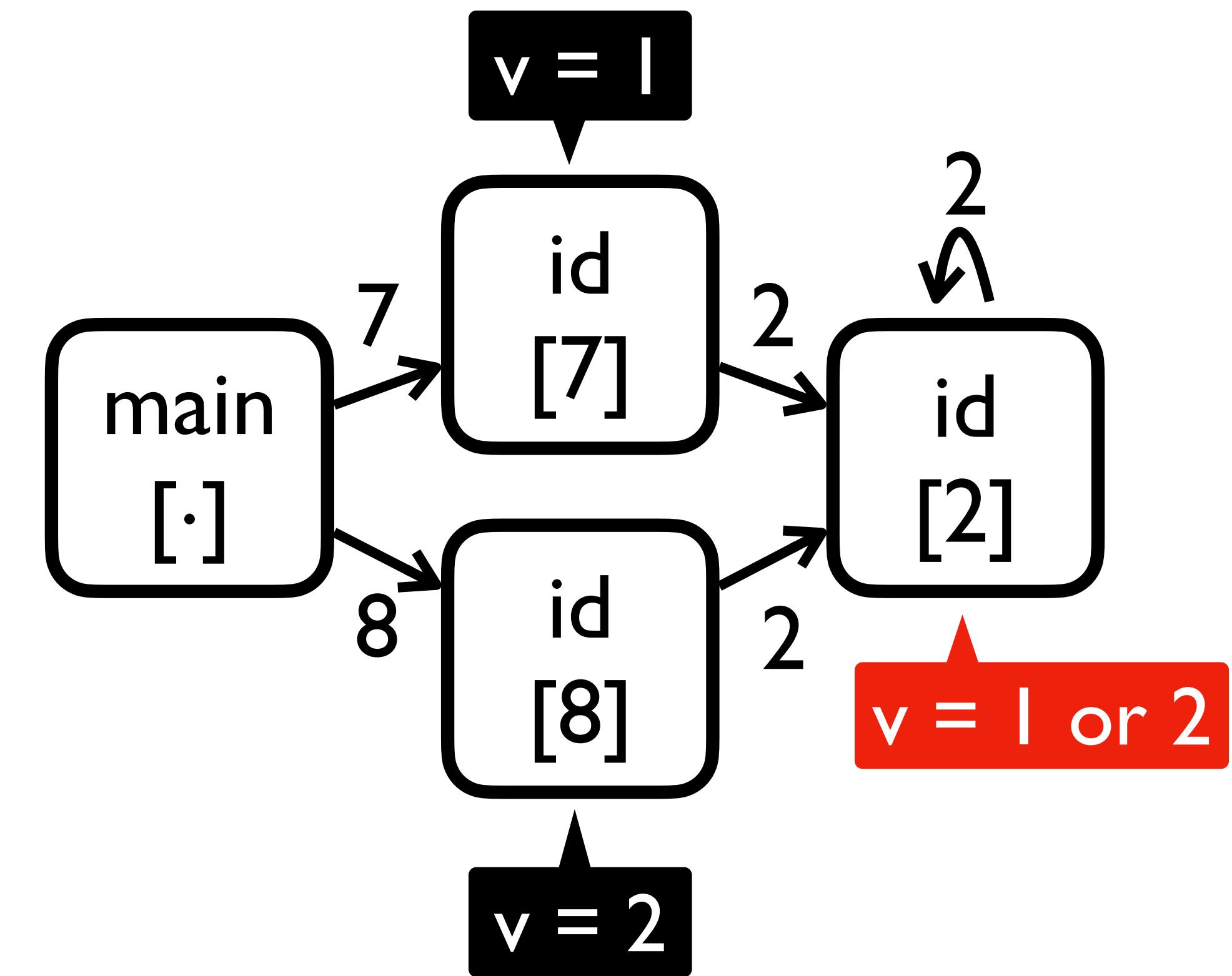
- 주요 |-요소 기반 함수 호출 요약



- 선택적 1-요소 기반 함수 호출 요약



```
1-call : {}
0-call : {main, id}
```

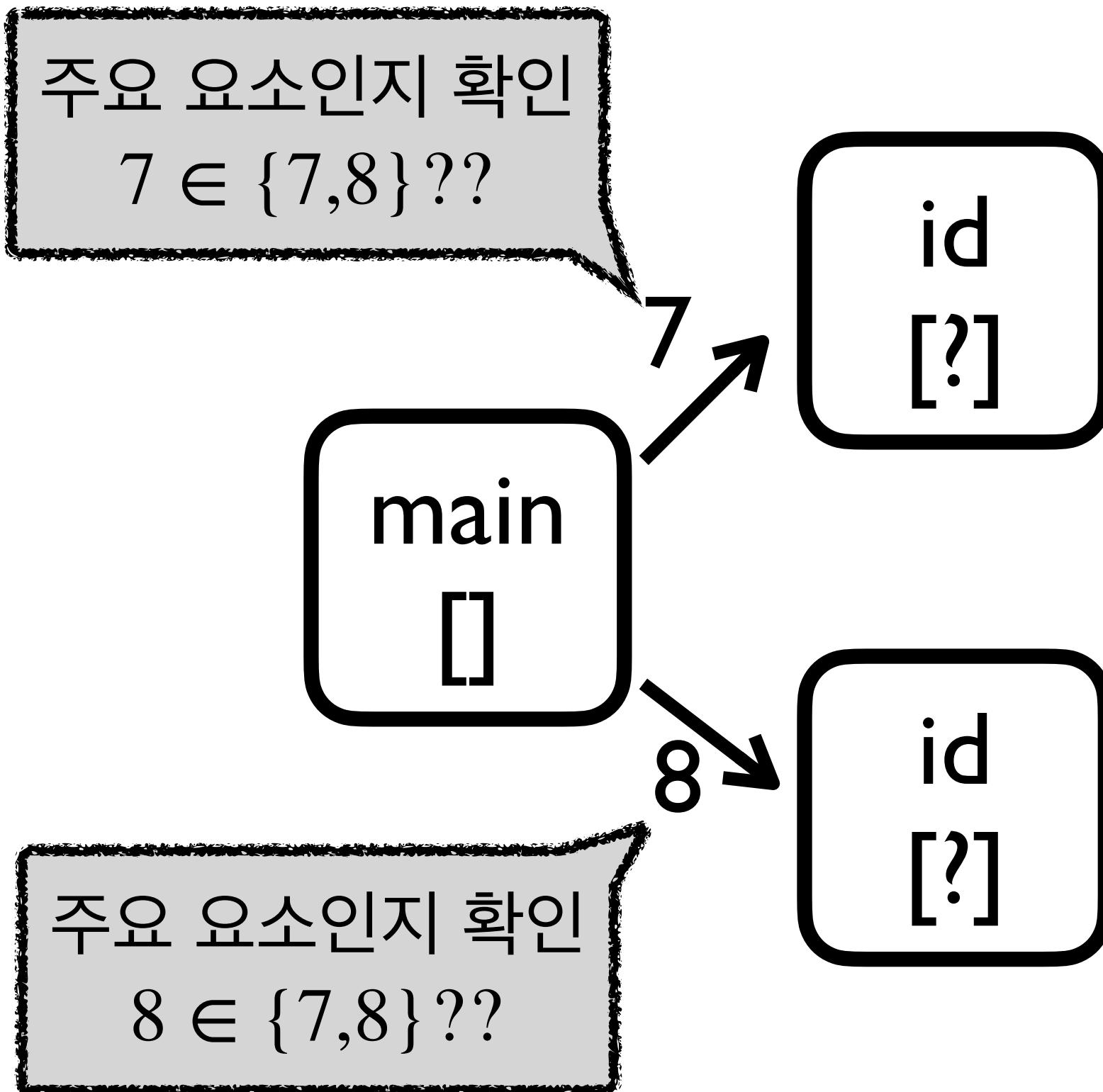


```
1-call : {id}
0-call : {main}
```

컨텍스트 터널링: 주요 K가 기반 요약

```
0: id(v, i){  
1:   if (i > 0){  
2:     return id(v, i-1);}  
3:   return v;}  
4:  
5: main(){  
6:   i = input();  
7:   v1 = id(1, i);  
8:   v2 = id(2, i);  
9:   assert (v1 != v2); //query  
10: }
```

주요 요소 = {7, 8}

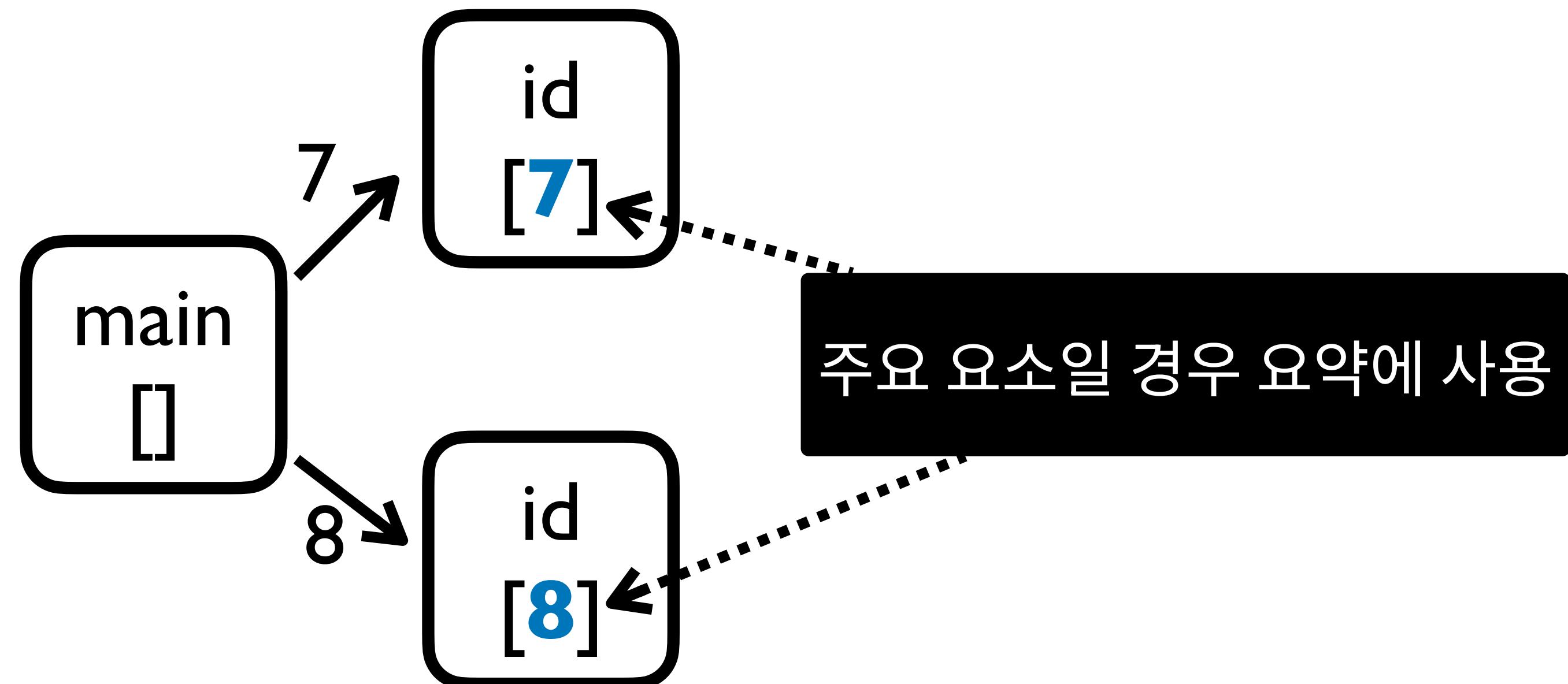


예제 프로그램

컨텍스트 터널링: 주요 K가 기반 요약

주요 요소 = {7, 8}

```
0: id(v, i){  
1:   if (i > 0){  
2:     return id(v, i-1);}  
3:   return v;}  
4:  
5: main(){  
6:   i = input();  
7:   v1 = id(1, i);  
8:   v2 = id(2, i);  
9:   assert (v1 != v2);//query  
10: }
```

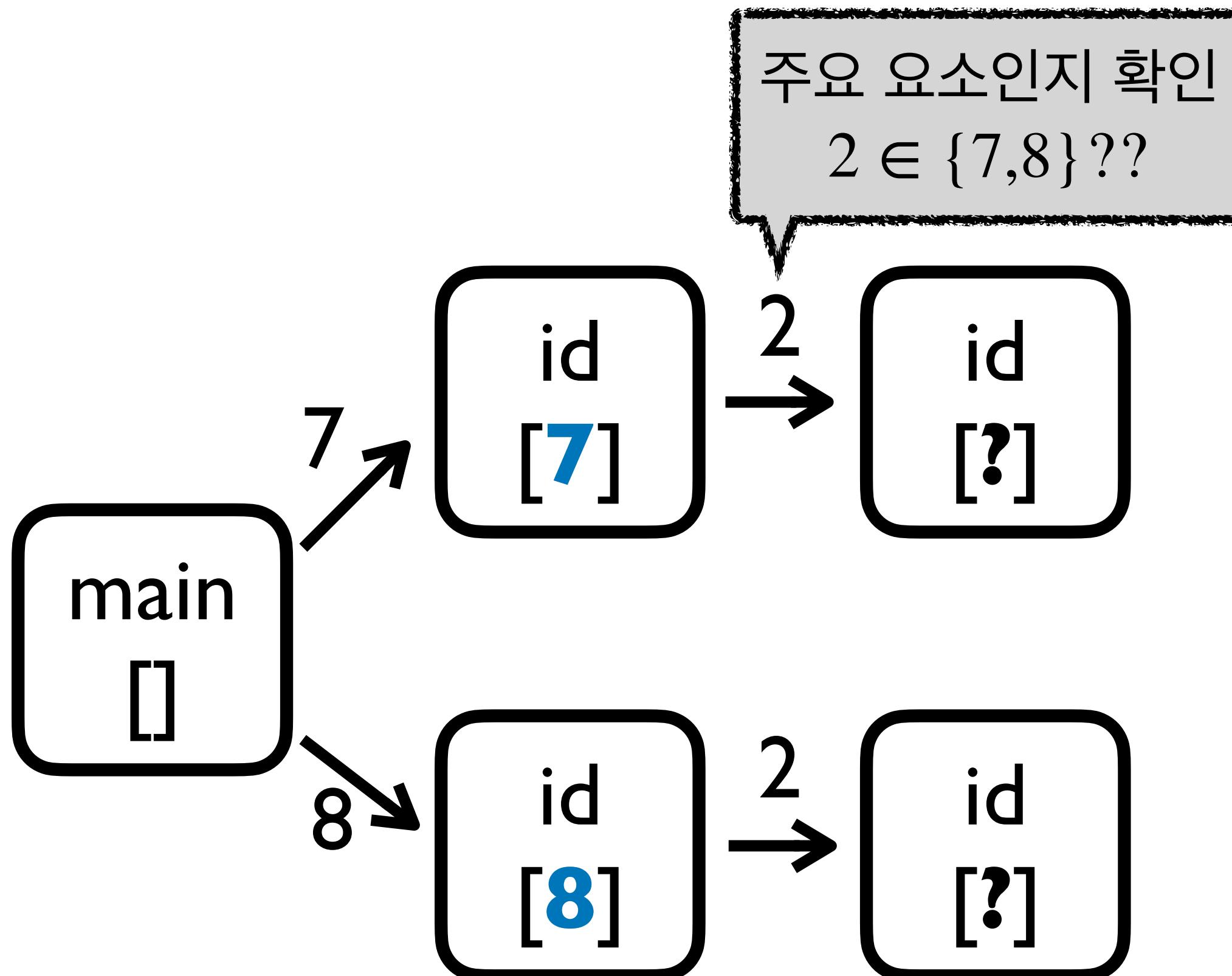


예제 프로그램

컨텍스트 터널링: 주요 K가 기반 요약

```
0: id(v, i){  
1:   if (i > 0){  
2:     return id(v, i-1);}  
3:   return v;}  
4:  
5: main(){  
6:   i = input();  
7:   v1 = id(1, i);  
8:   v2 = id(2, i);  
9:   assert (v1 != v2); //query  
10: }
```

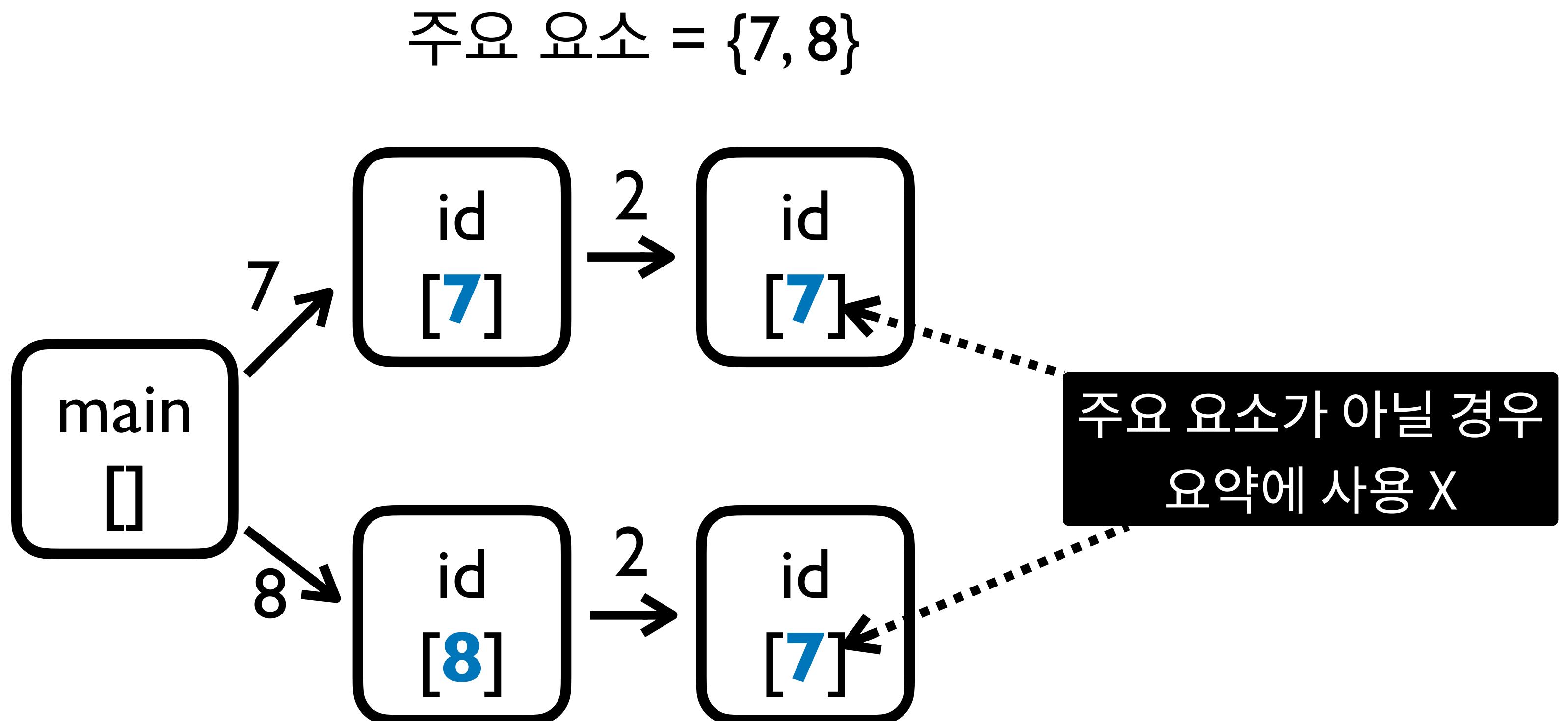
주요 요소 = {7, 8}



예제 프로그램

컨텍스트 터널링: 주요 K가 기반 요약

```
0: id(v, i){  
1:   if (i > 0){  
2:     return id(v, i-1);}  
3:   return v;}  
4:  
5: main(){  
6:   i = input();  
7:   v1 = id(1, i);  
8:   v2 = id(2, i);  
9:   assert (v1 != v2); //query  
10: }
```

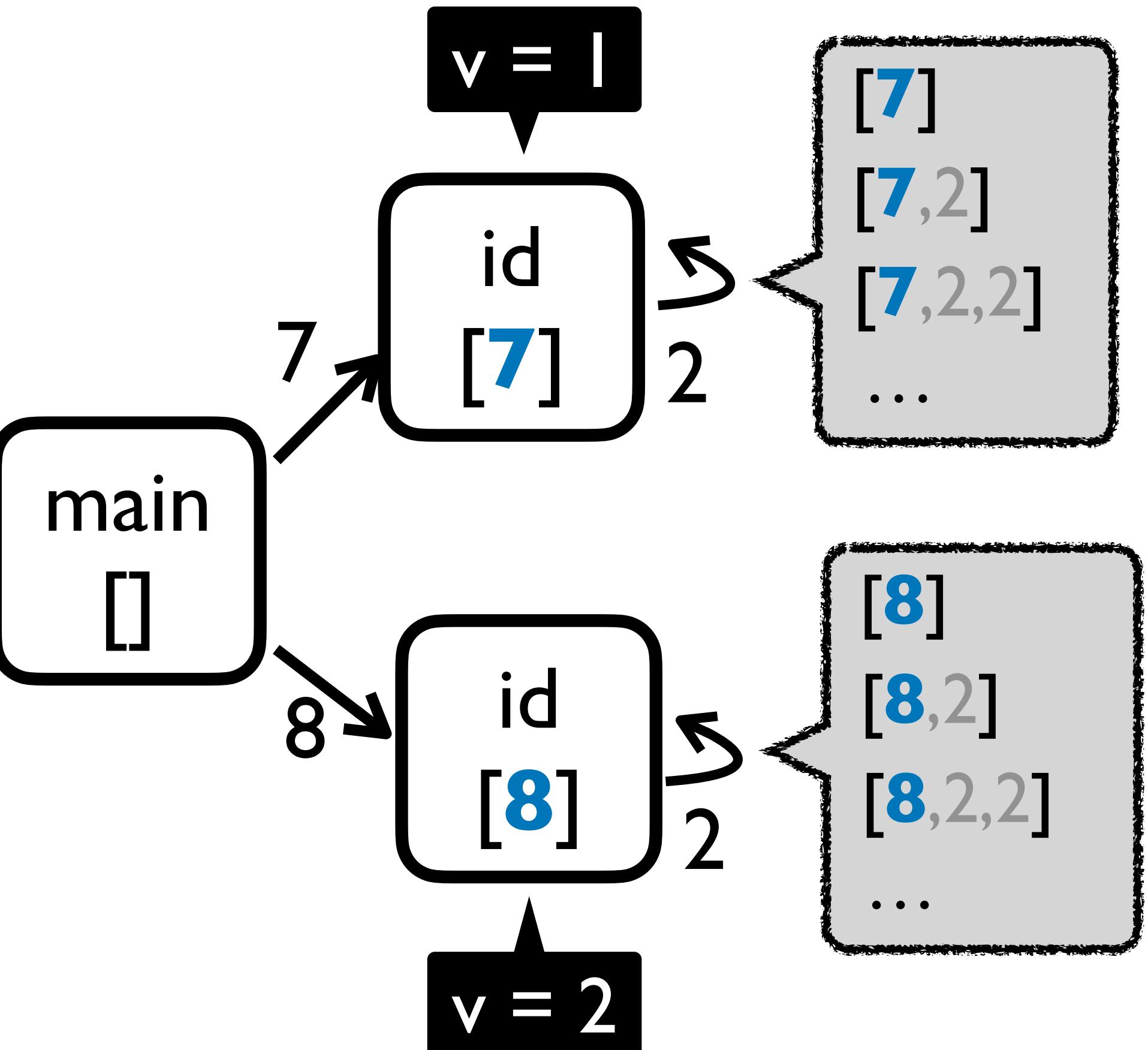


예제 프로그램

컨텍스트 터널링: 주요 K개 기반 요약

```
0: id(v, i){  
1:   if (i > 0){  
2:     return id(v, i-1);}  
3:   return v;}  
4:  
5: main(){  
6:   i = input();  
7:   v1 = id(1, i);  
8:   v2 = id(2, i);  
9:   assert (v1 != v2); //query  
10: }
```

예제 프로그램



1개 주요 요소 기반 함수 호출 요약
(주요 요소 = {7, 8})