

컨텍스트 터널링

구분할 호출환경 배달하기: 고정관념에 도전하기

전 민석



Aug.21.2025 @ SIGPL 여름학교

문제

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

문제

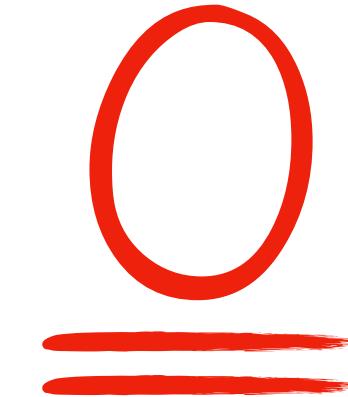
- 아래 문장은 어떤 동화를 한 문장으로 요약한 것이다, 몇점짜리 요약일까?

“The End”

문제

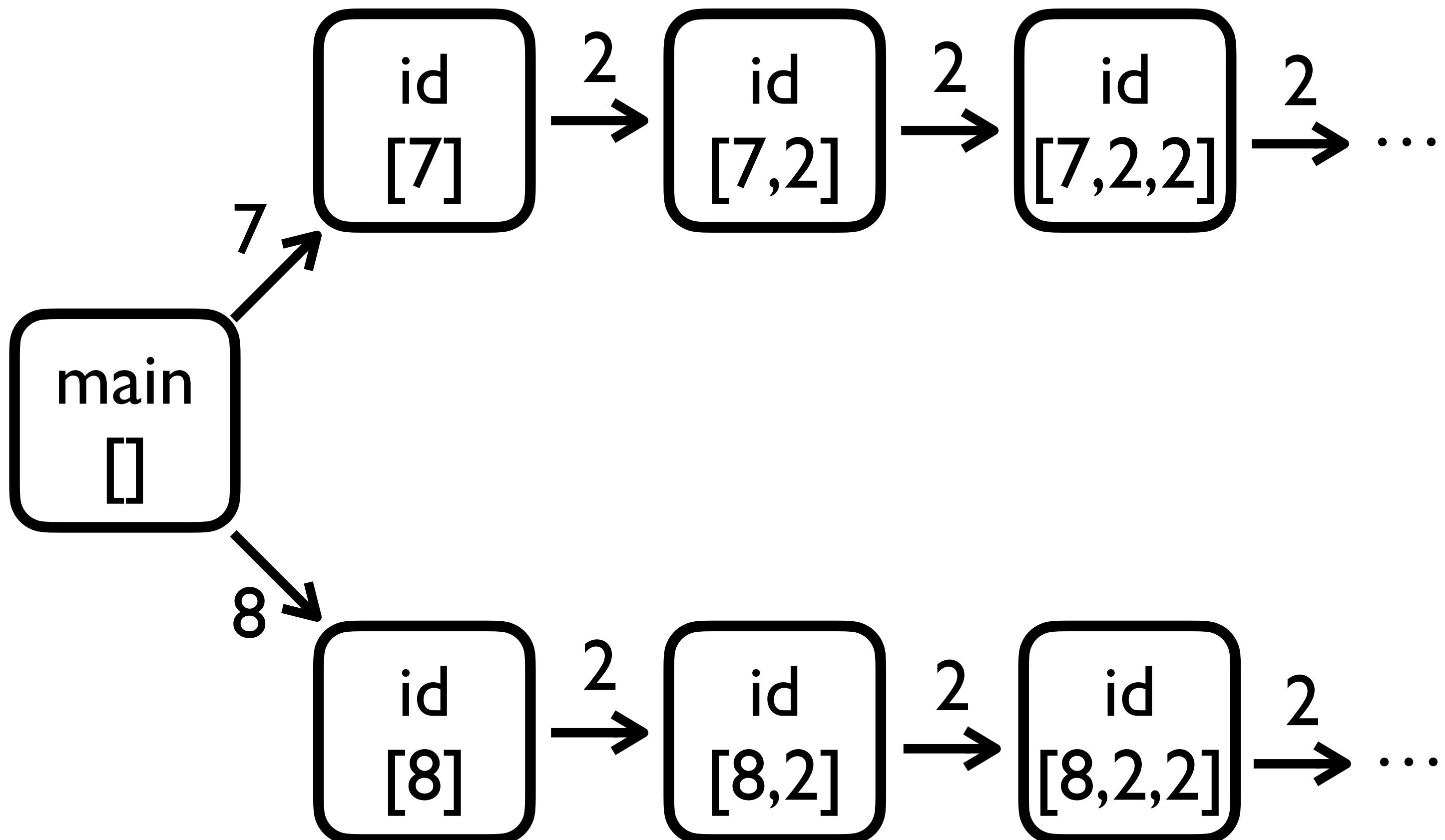
- 아래 문장은 어떤 동화를 한 문장으로 요약한 것이다, 몇점짜리 요약일까?

“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:

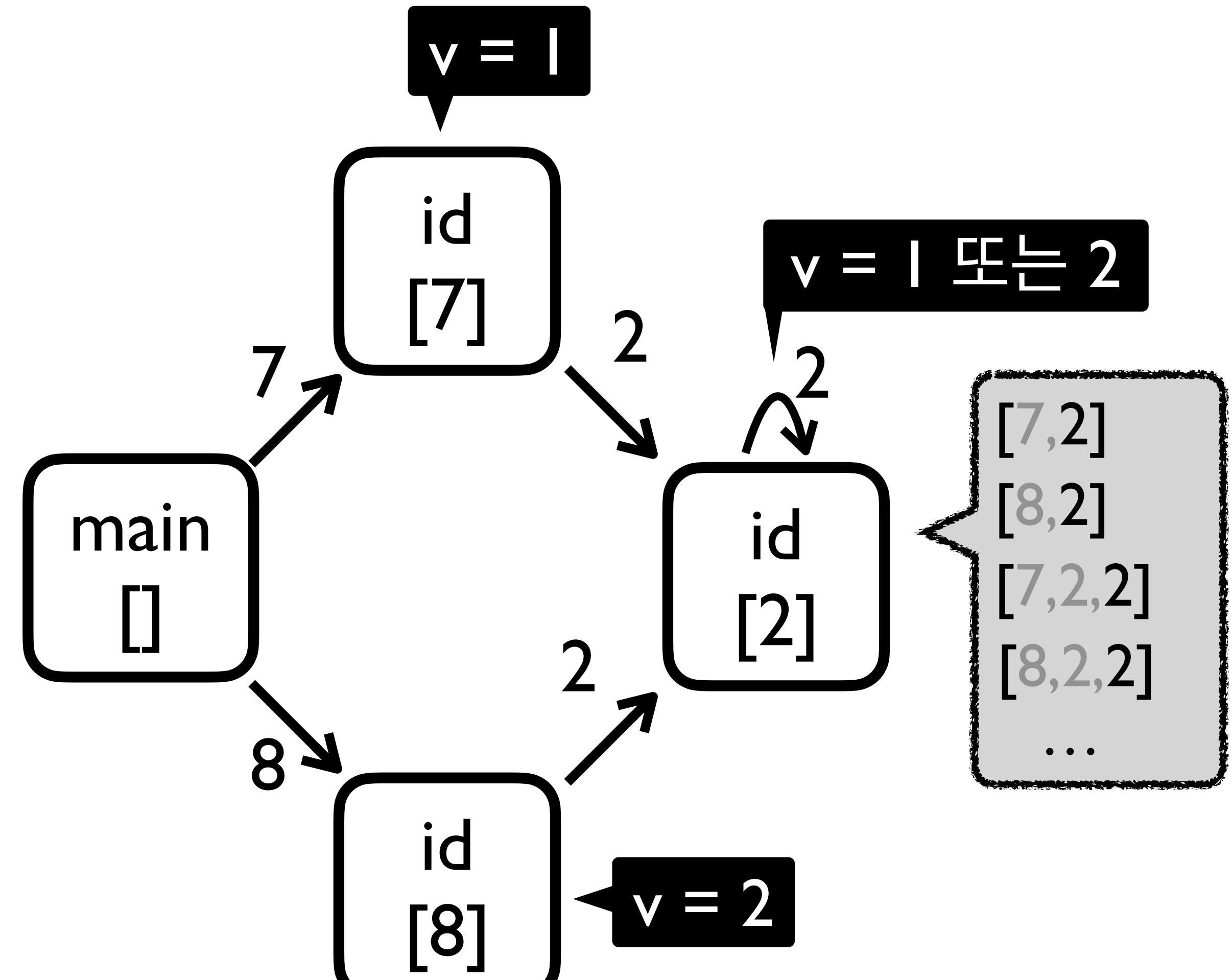


3개 요소 기반 함수 호출 요약
(3-context sensitivity)



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

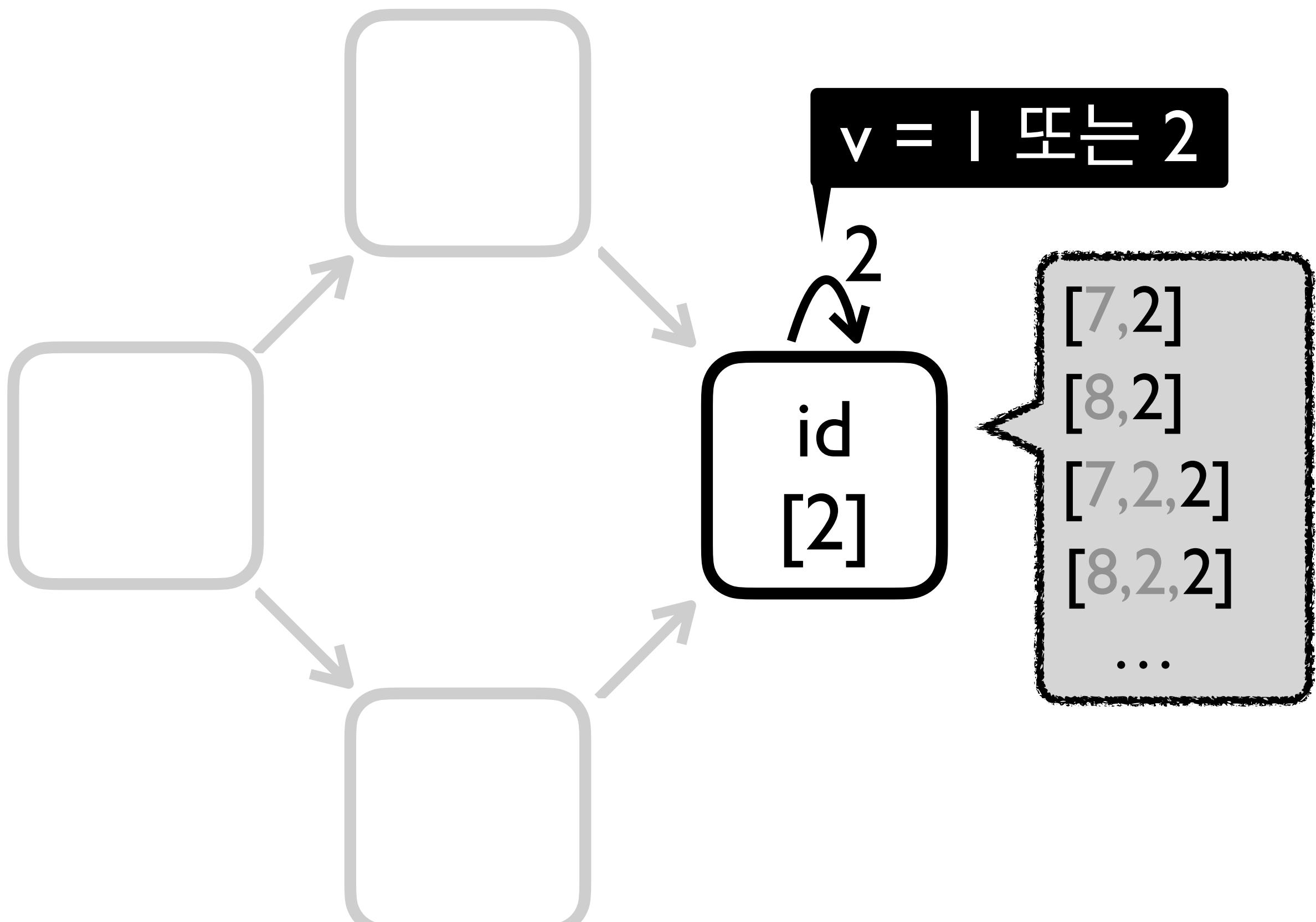
K개 요소 기반 함수 호출 요약

```
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

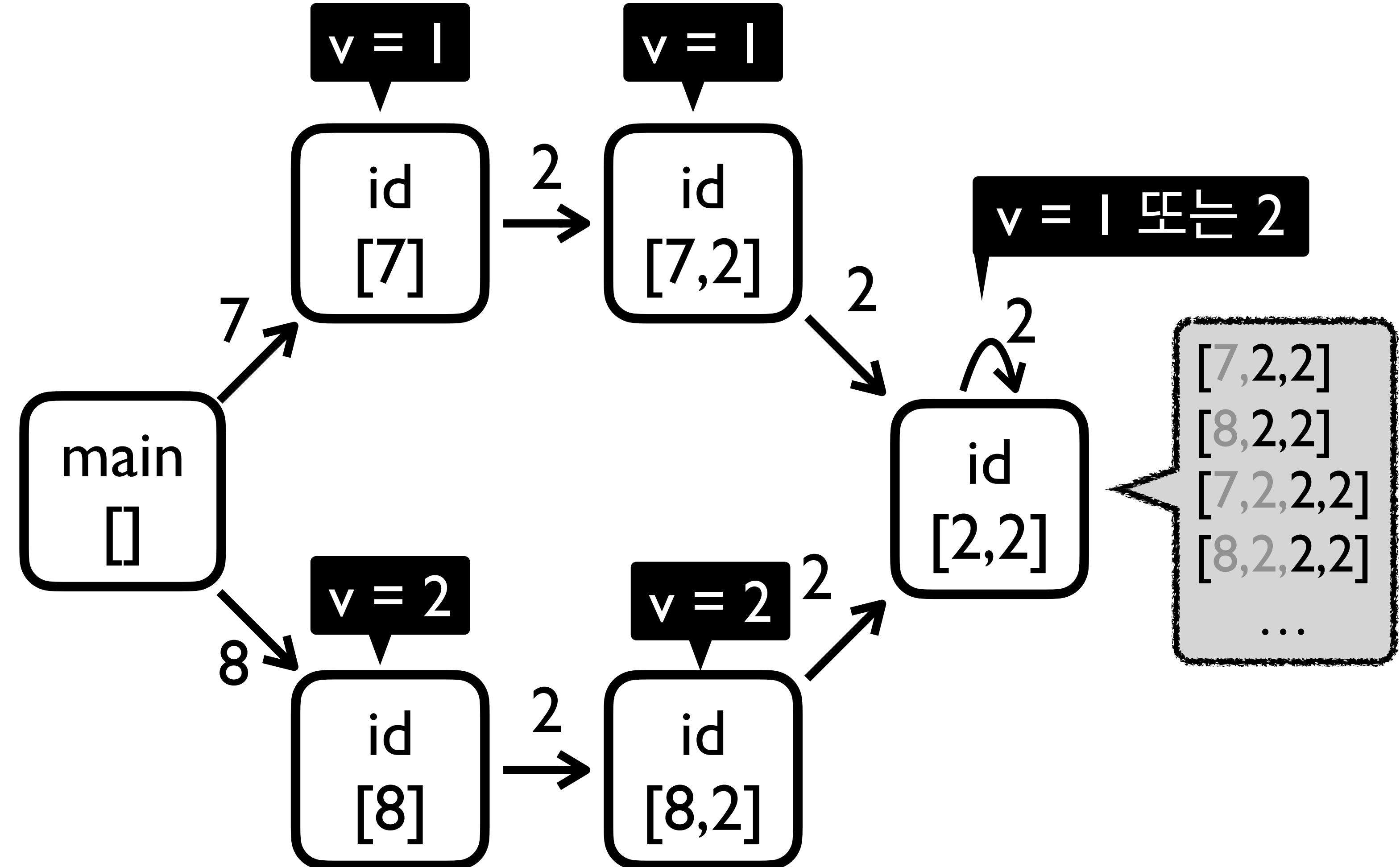
v2 = 1 또는 2



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: }
```

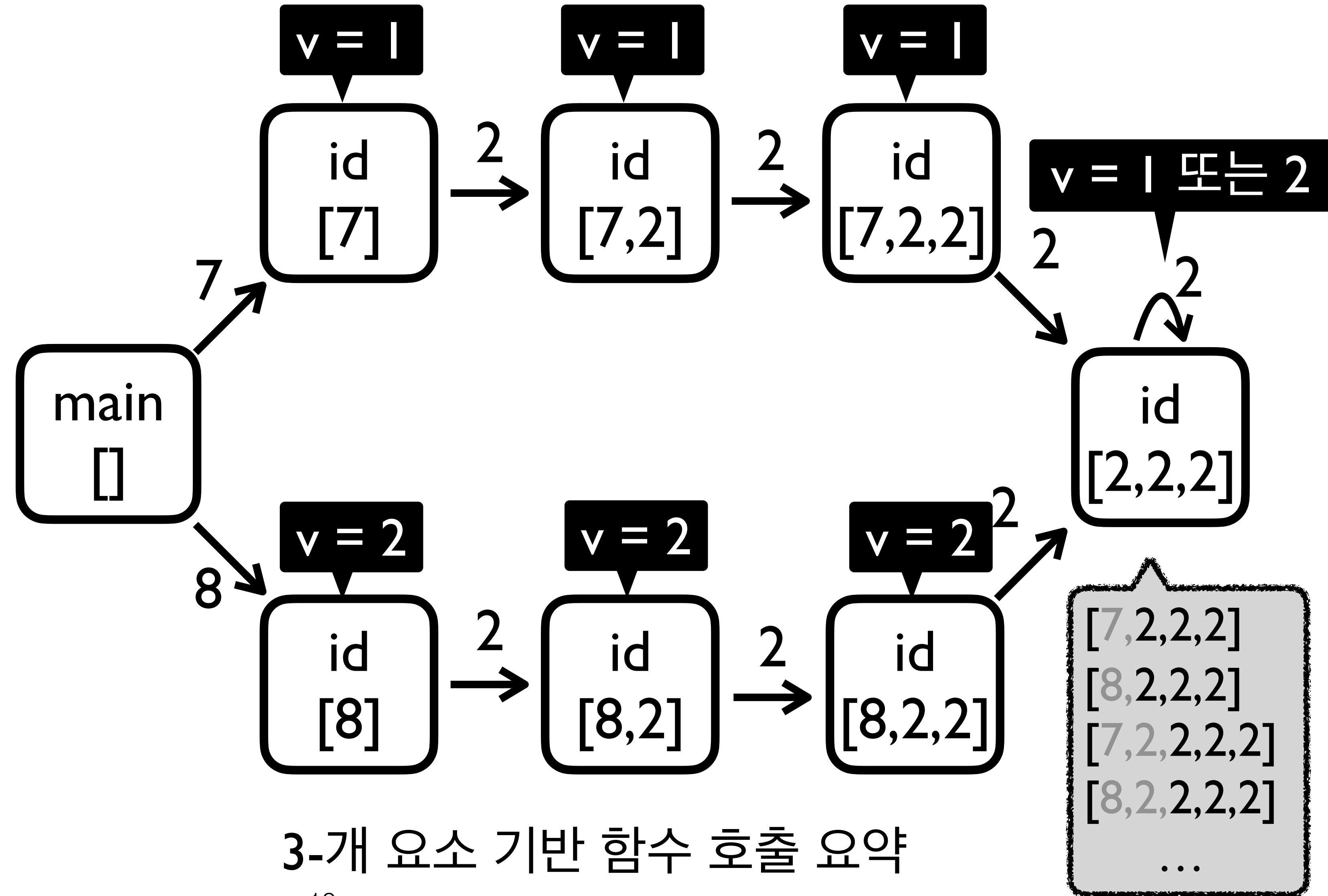


2-개 요소 기반 함수 호출 요약

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); //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

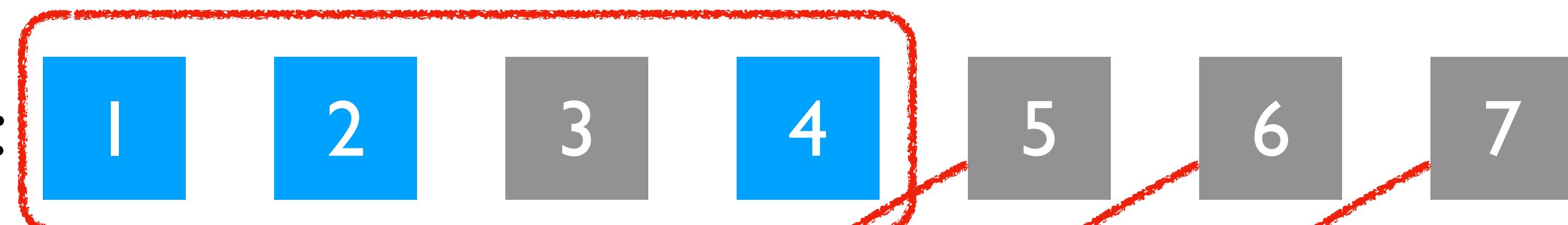


- In practice, k is a small number (usually ≤ 3)
- Method contexts and heap contexts may use different k
 - e.g., $k=2$ for method contexts, $k=1$ for heap contexts

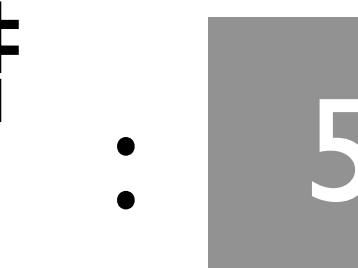
마지막 K개 기반 요약의 문제점

주요 요소들이 지워짐!

실제 함수 호출 맥락:



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

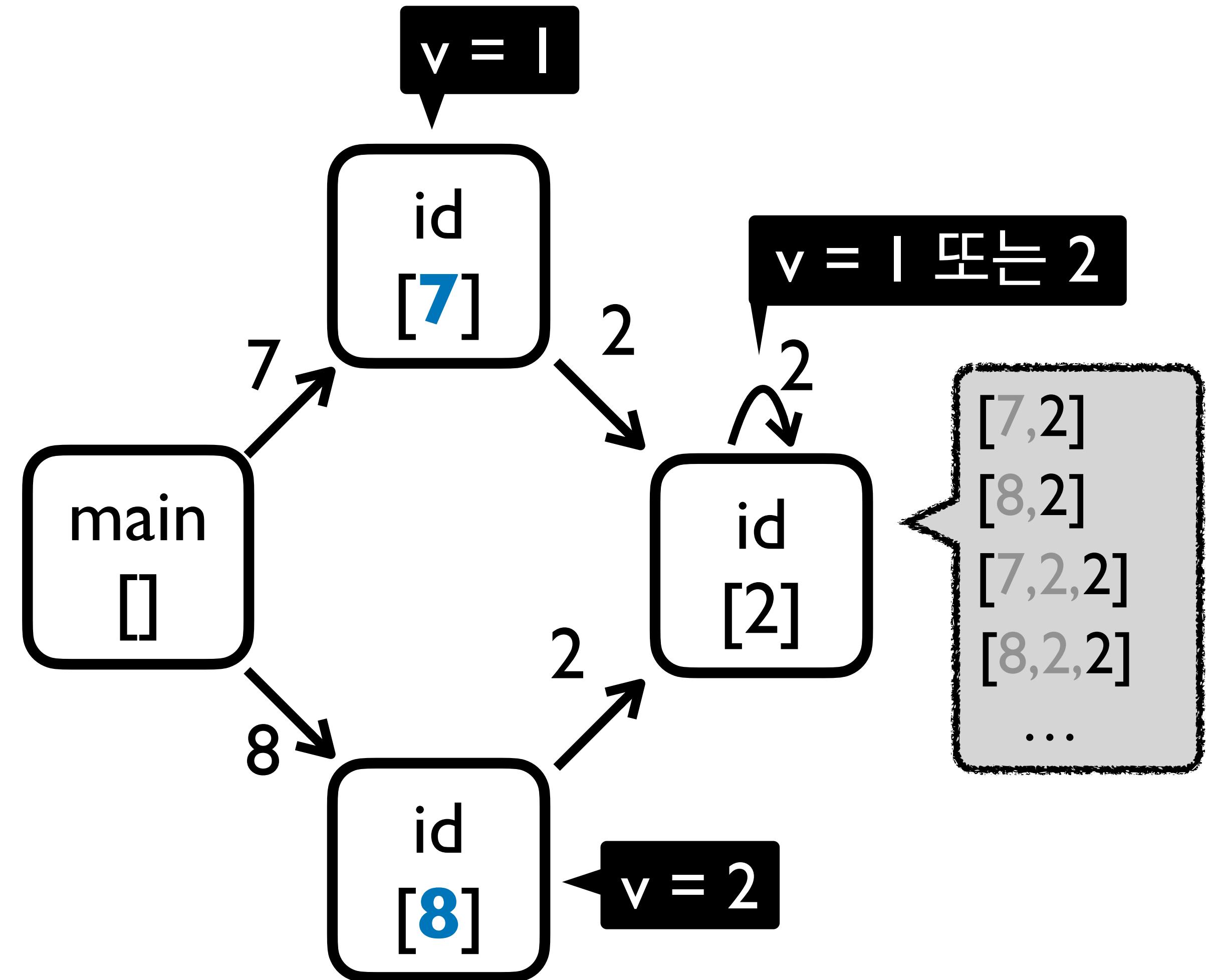


: 주요 요소

: 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?

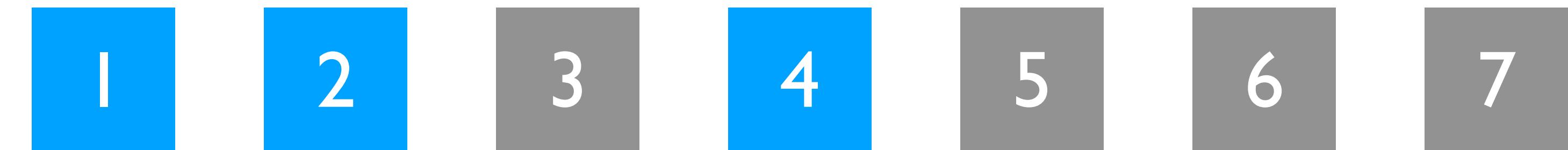
질문:

I-call-site sensitivity로 정확하게 분석할 순 없나?

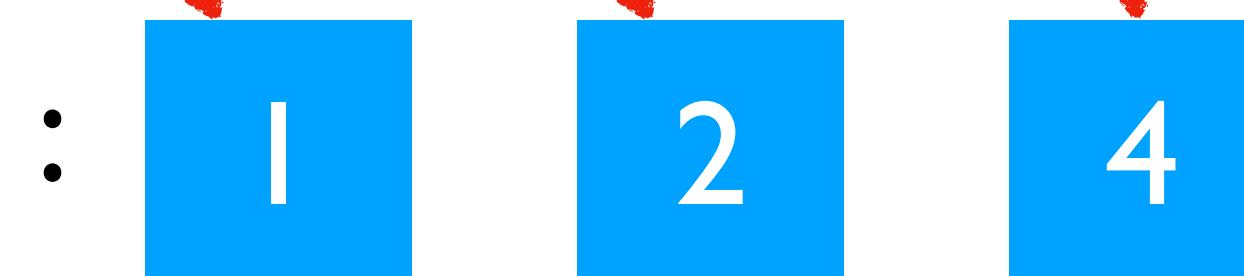
부정확함

호출 환경 배달하기: 주요 K개 기반 요약

실제 함수 호출 맥락:



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



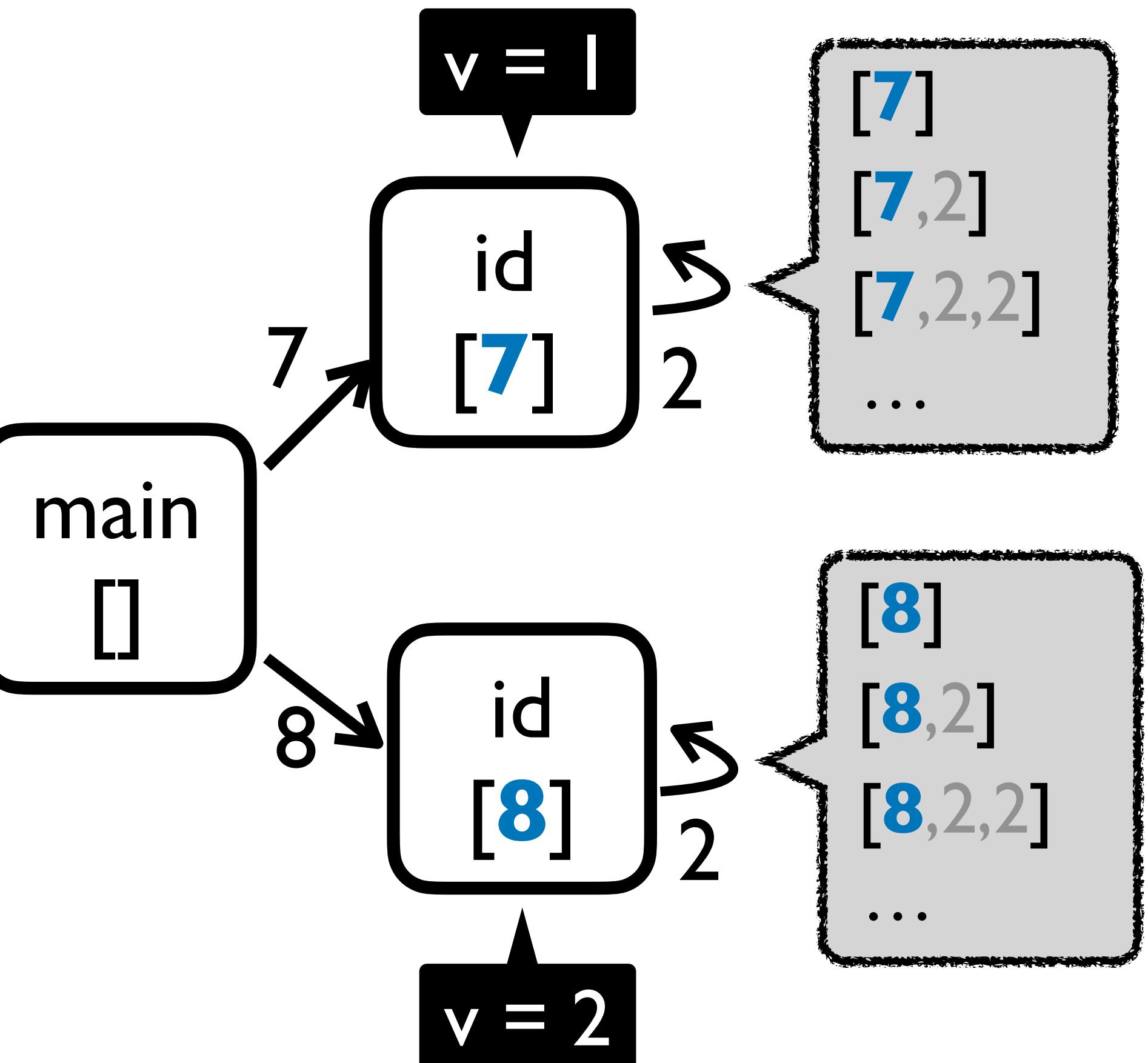
: 주요 요소

: 부차적 요소

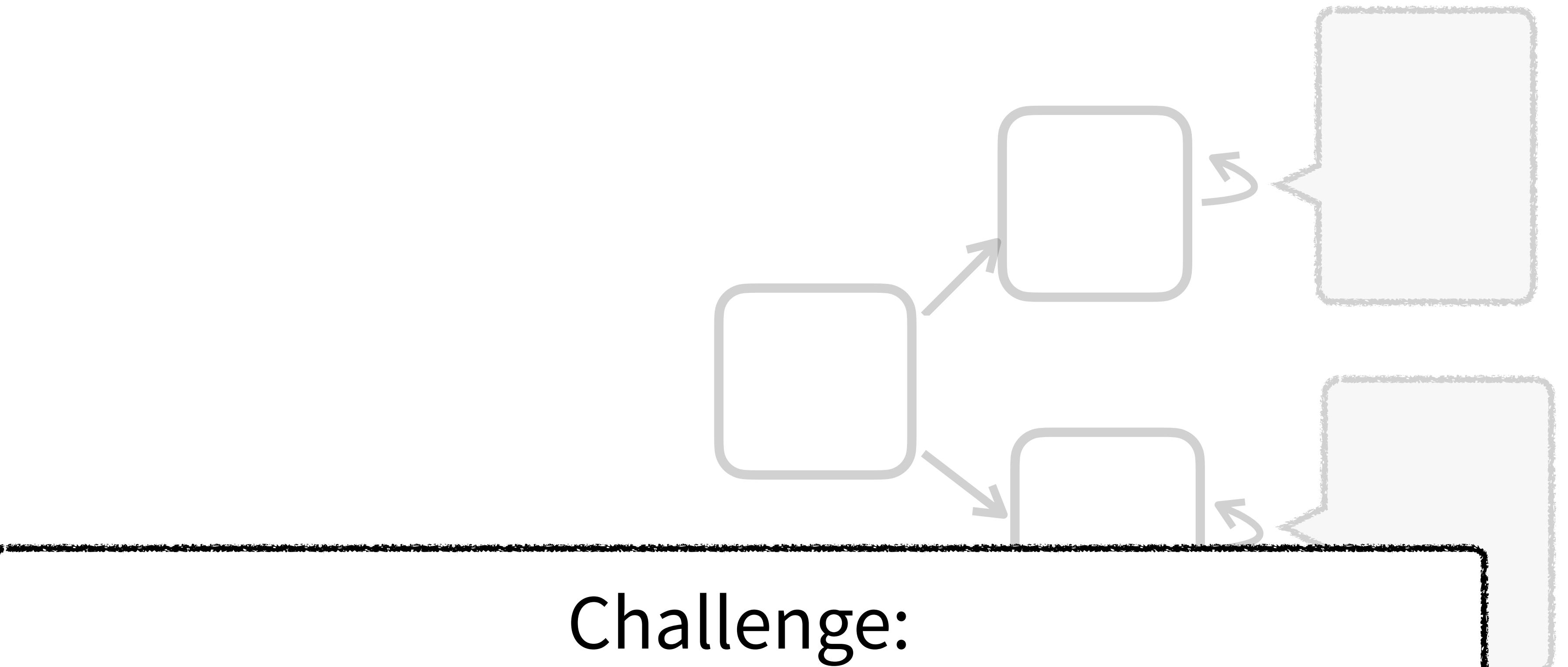
호출 환경 배달하기: 주요 K개 기반 요약

```
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})

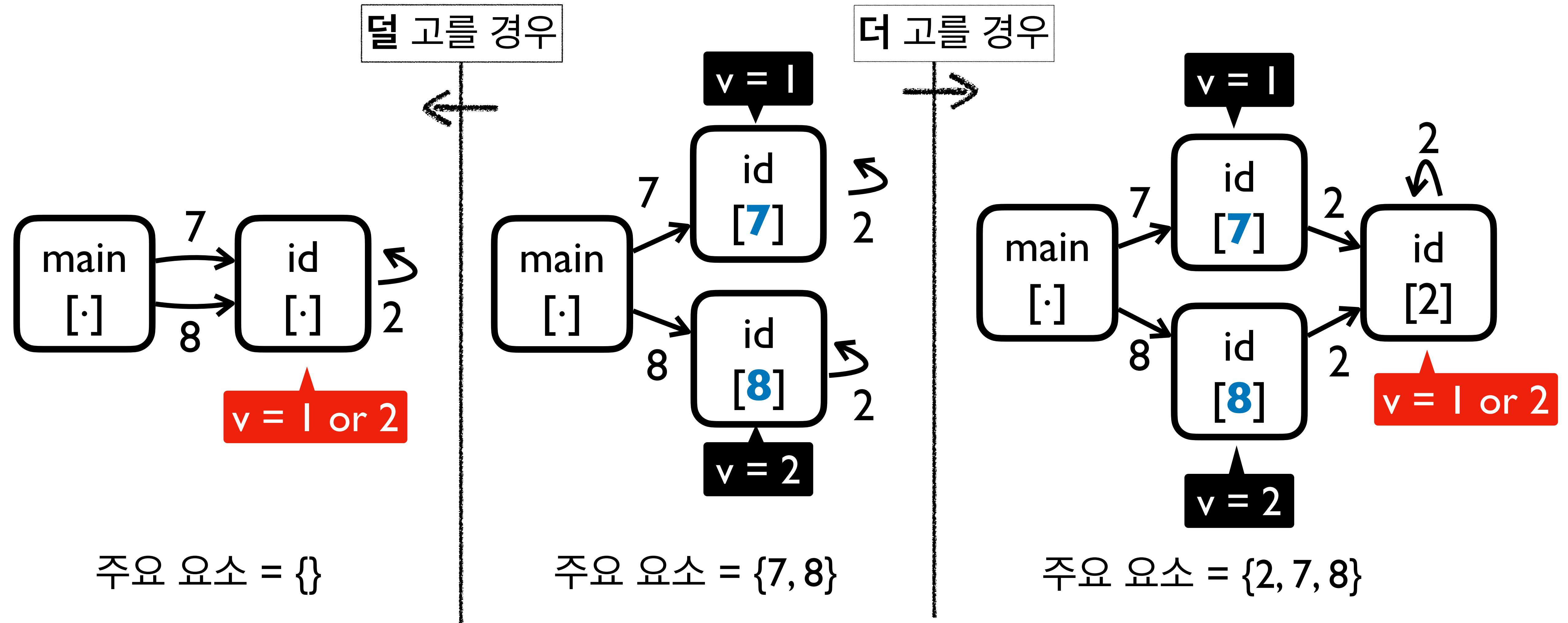


Challenge:
분석 전에 주요 요소를 정확하게 알아내야 함

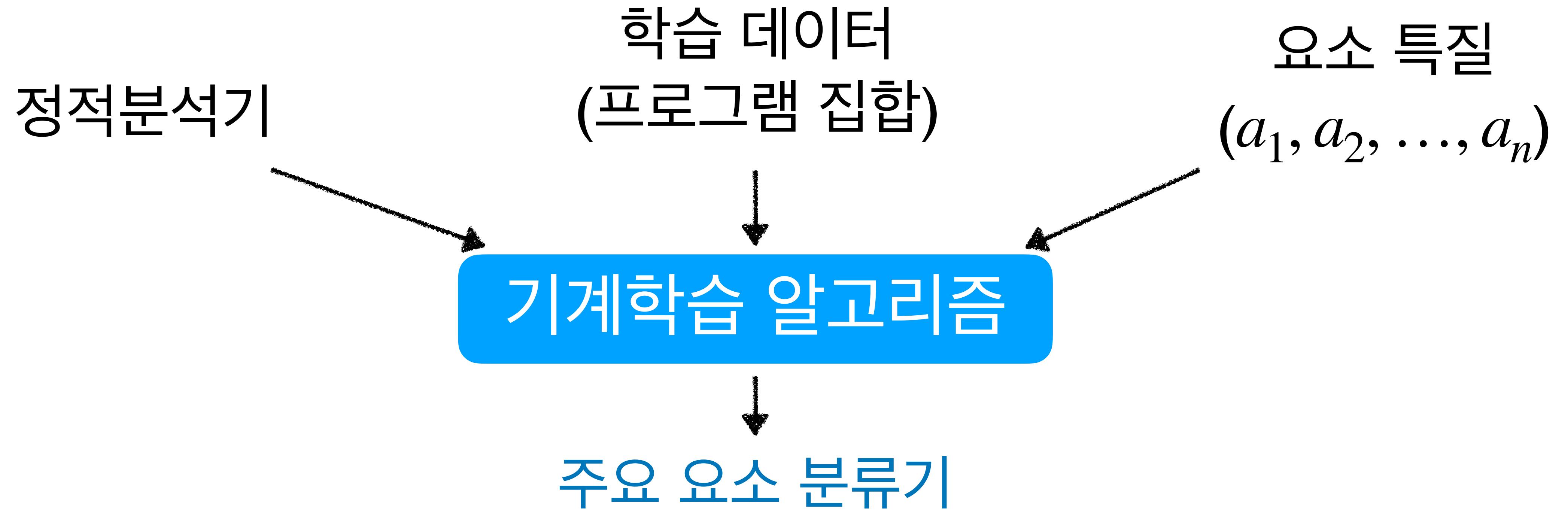
1개 주요 맥락 가변 함수 호출 요약

(주요 요소 = {7, 8})

- 정확하게 분류해야만 분석의 정확도를 향상시킬 수 있음



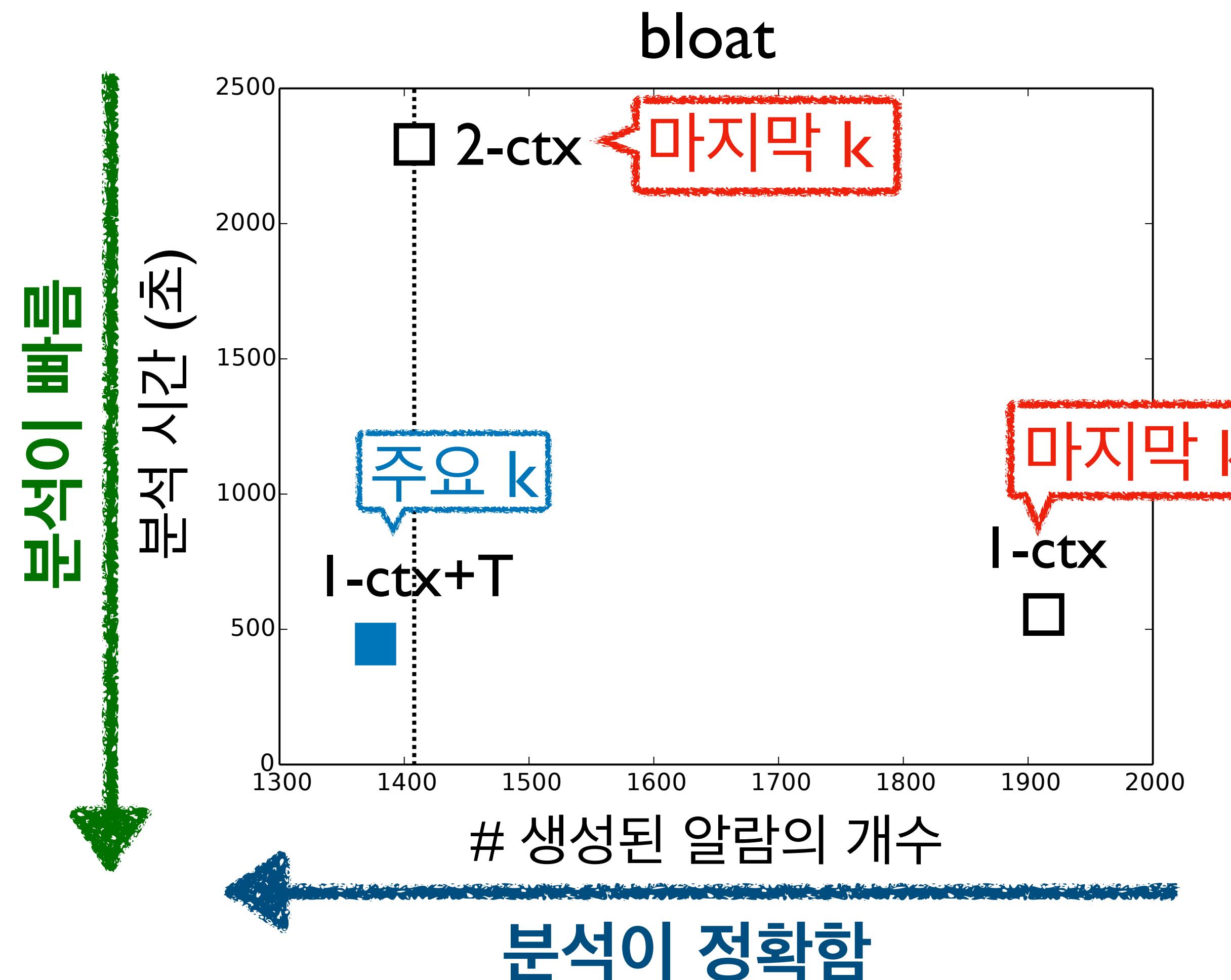
접근 방법: 데이터 기반 컨텍스트 터널링



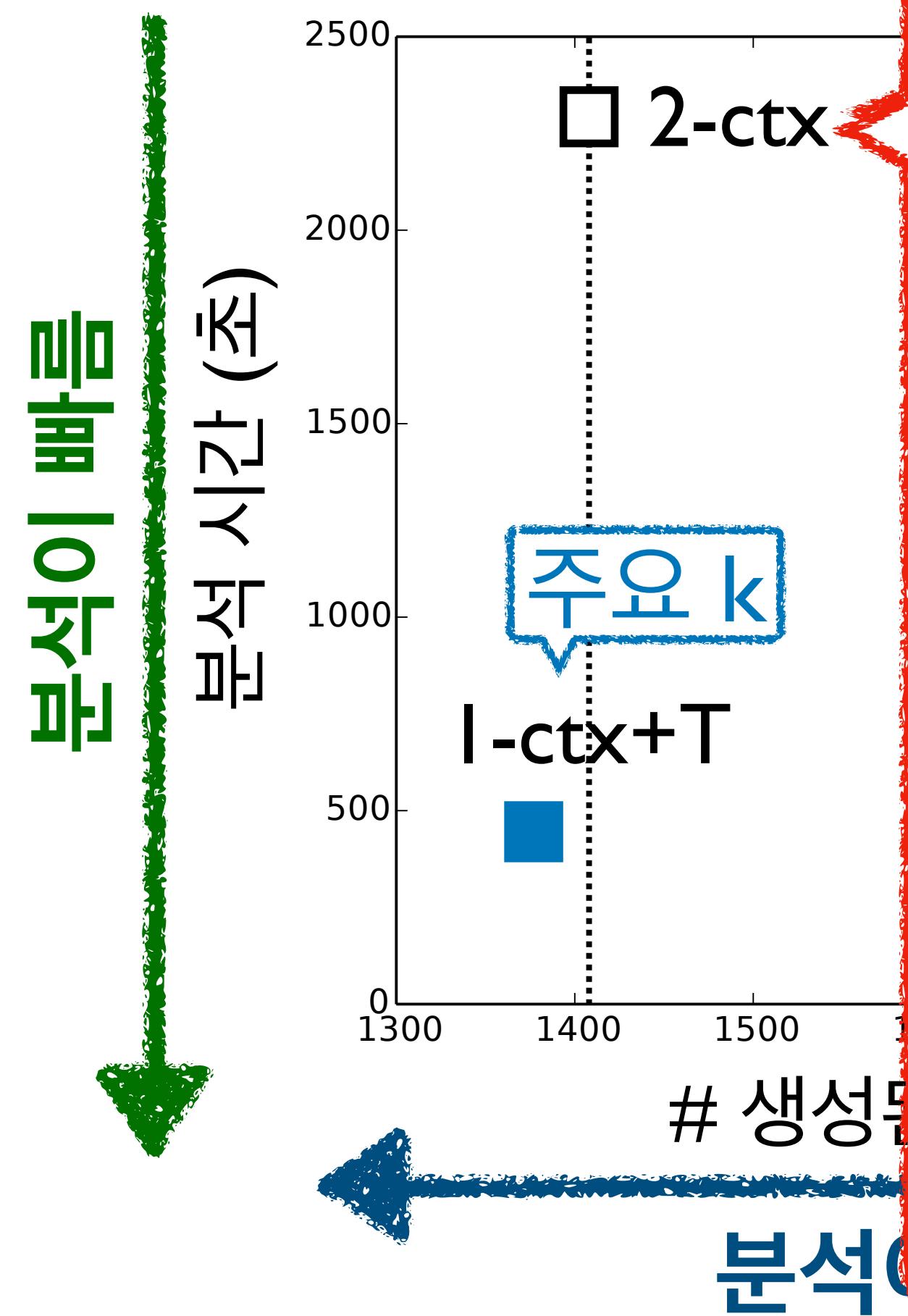
$$f = (a_1 \wedge \neg a_2 \wedge \neg a_3 \wedge \dots) \vee (a_1 \wedge \neg a_3 \wedge a_7 \wedge \dots) \vee \dots$$

마지막 k 기반 요약 vs 주요 k기반 요약

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



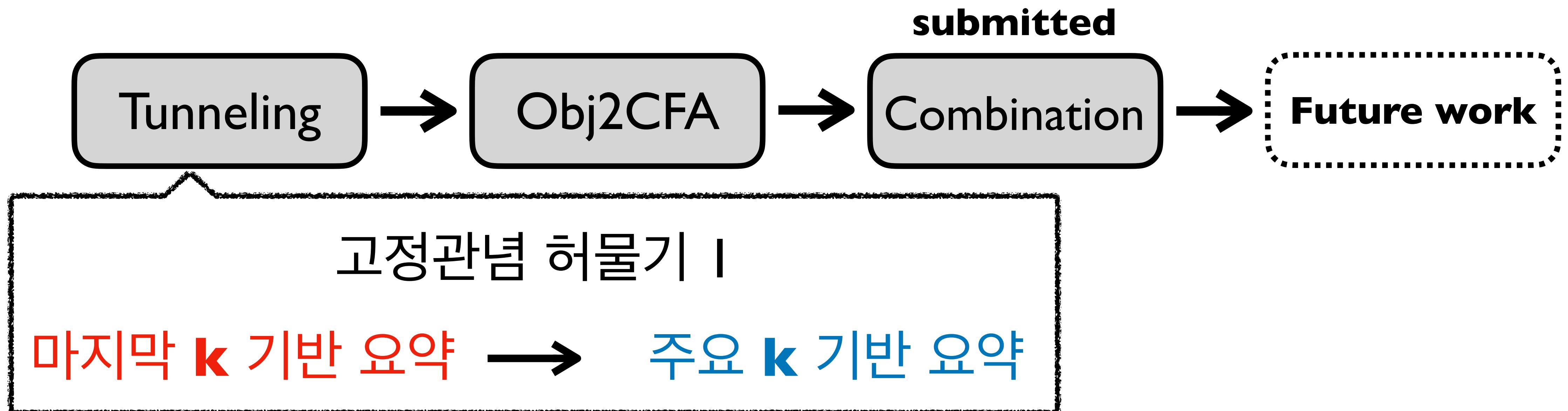
- 2-ctx 는 연구에서 정확도 상한선으로 사용되어 왔음



- “...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]
- ...

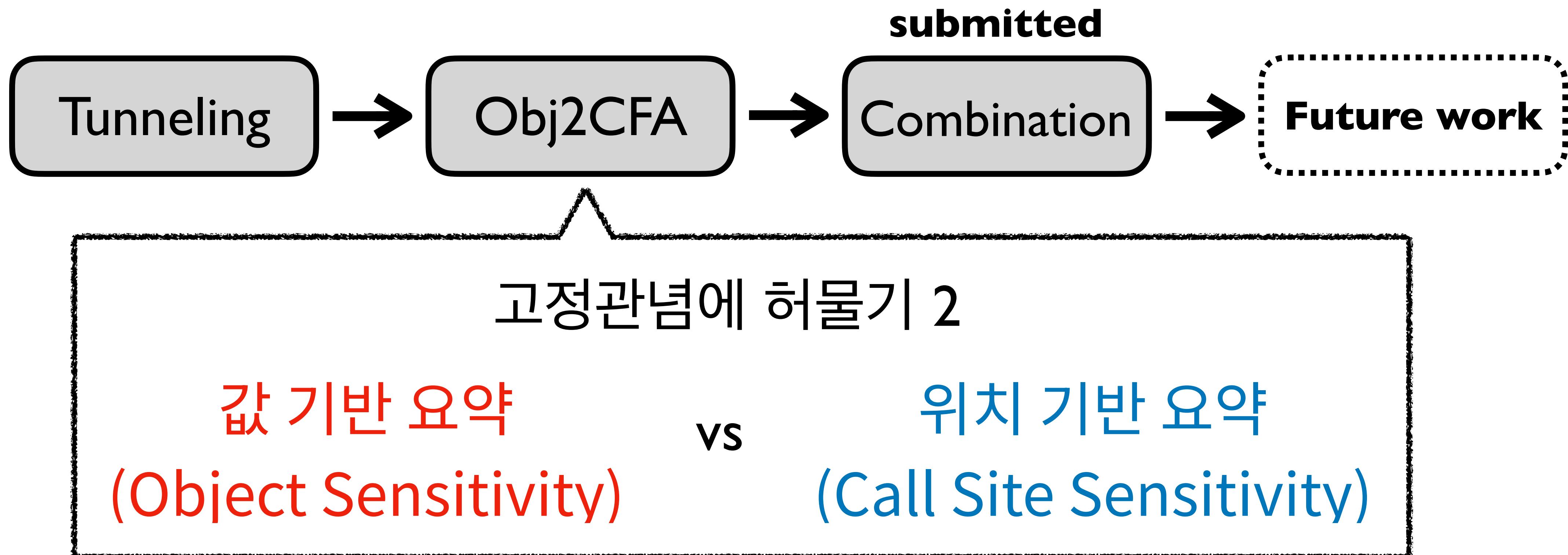
고정관념에 도전하기

- 목표: 주요 k기반 요약 방식을 표준으로 만들기



고정관념에 도전하기

- 목표: 주요 기반 요약 방식을 표준으로 만들기

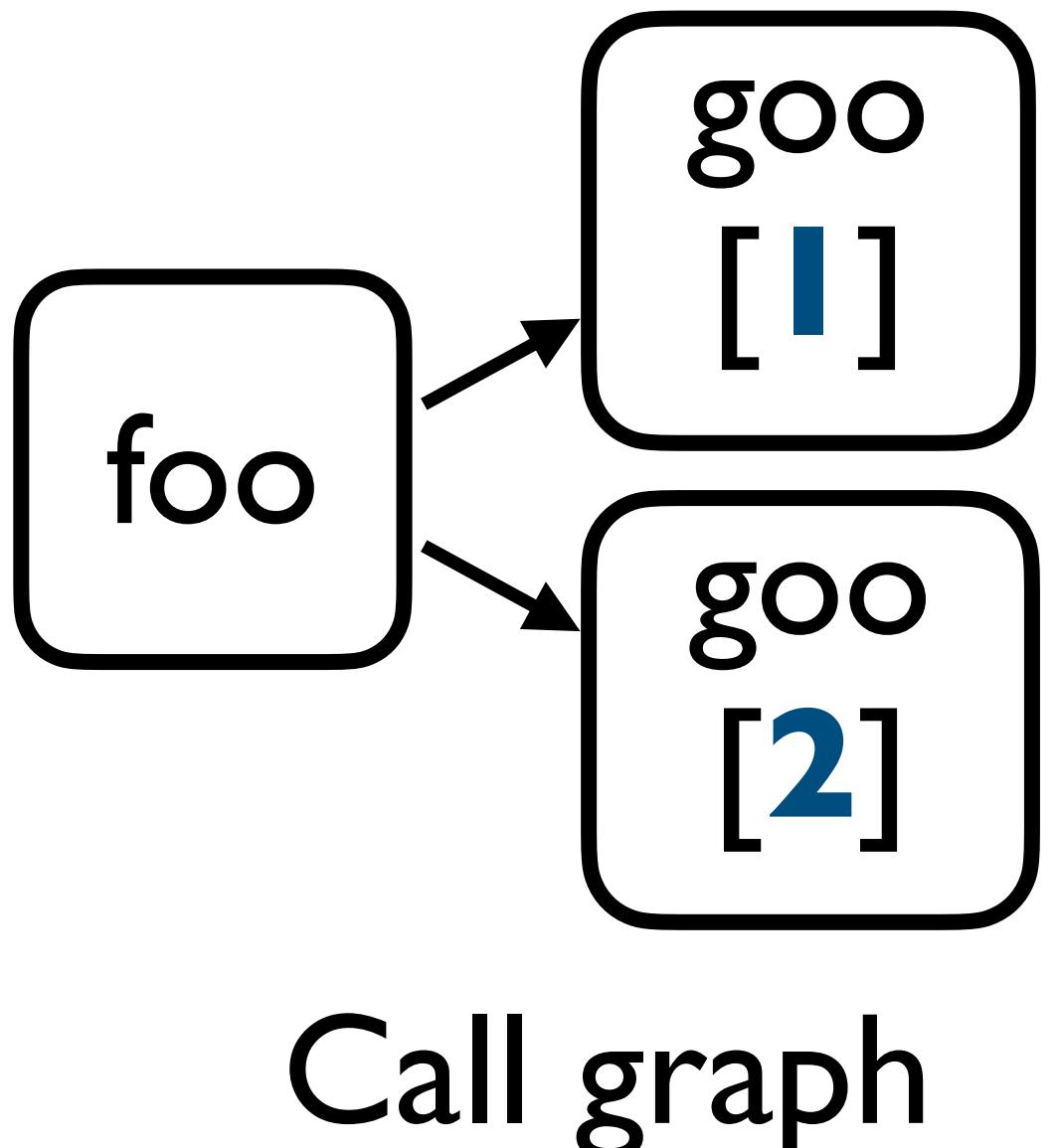


위치 기반 요약 vs 값 기반 요약

위치 기반 요약 (1981)

- “어디”를 기반으로 요약

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

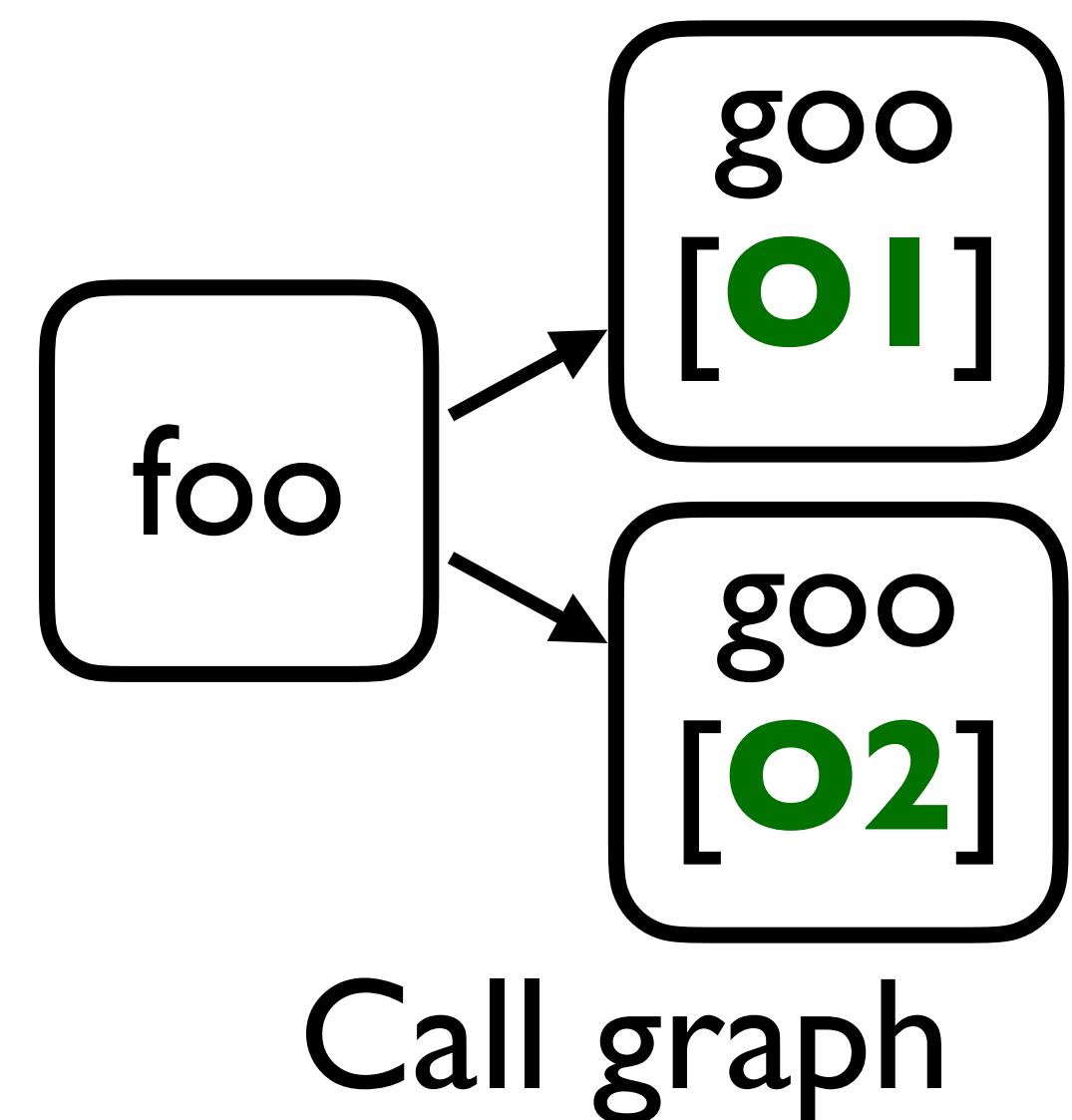


값 기반 요약 (2002)

- “무엇”을 기반으로 요약

O1 or O2

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



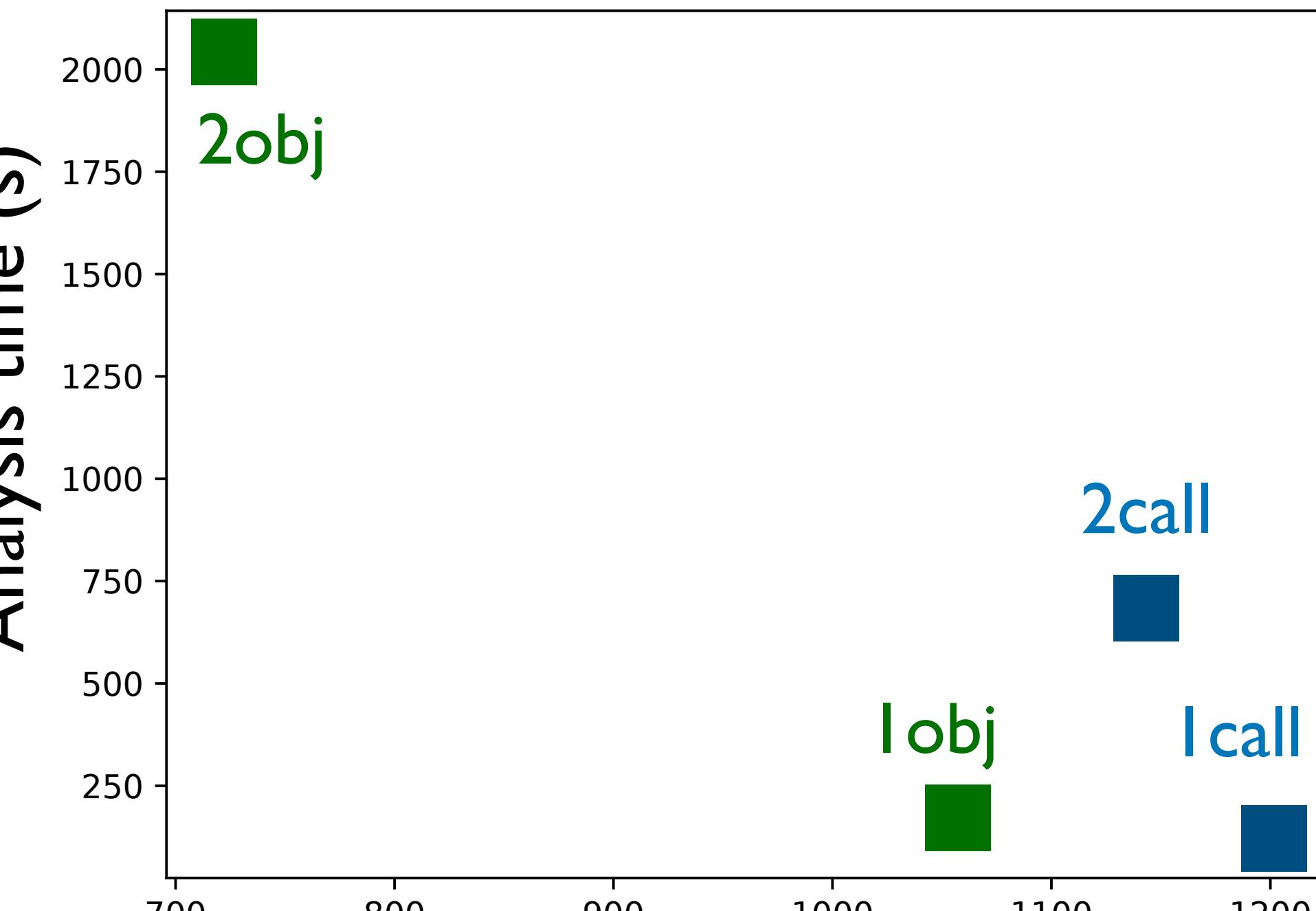
1981

2002

2010

2022

xalan



분석이 정확함

분석이 정확함

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 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 for Java
- Likely to scale better

1-caller
fun1()
id2(id)
id2(id)
fun2()
fun2()

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:

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

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

Of course, it is also possible that we are uncertain to which of the 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

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

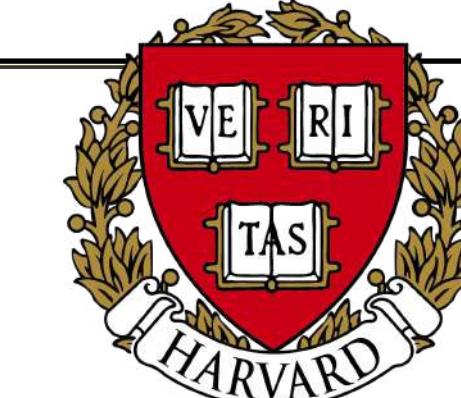
• • •



KOREA
UNIVERSITY



National and Kapodistrian
University of Athens



Carnegie
Mellon
University

now
the essence of knowledge

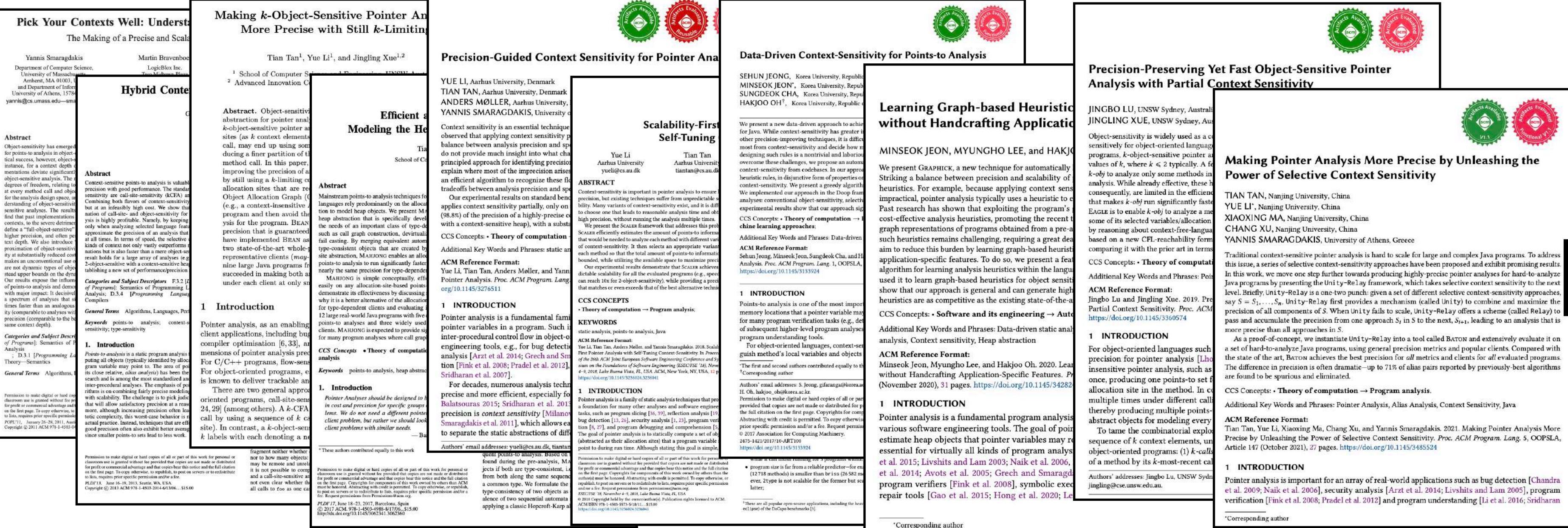
1981

2002

2010

2022

값 기반 요약을 대상으로 하는 연구가 쏟아짐



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]

The image displays a grid of 10 academic conference papers from the ACM SIGART 2018 proceedings. Each paper is represented by a small thumbnail containing the title, authors, and abstract. 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 Points-to Analysis", "Hybrid Context-Sensitivity for Points-to Analysis", "Precision-Guided Context Sensitivity for Points-to Analysis", "Introspective Analysis: Context-Sensitivity, Across the Program", "A Machine-Learning Algorithm with Disjunctive Model for Data-Driven Program Analysis", "MINSEOK JEON, SEHUN JEONG*, SUNGDEOK CHA, and HAKJOO OH†, Korea University, Republic of Korea", and "ACM Reference Format". The thumbnails are arranged in two columns of five.

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]

The figure displays a grid of six academic papers, each representing a significant advancement in static program analysis over three decades:

- Making k -Object-Sensitive Po More Precise with Still k** (Tian Tan, Yue Li, Jingling, 1981)
- Scalability-First Pointer Analysis Self-Tuning Context-Sensitivity** (Yue Li, Tian Tan, Anders Møller, Yannis Smaragdakis, 2002)
- Pick Your Contexts Well: Understanding the Making of a Precise and Scalable Hybrid Context-Sensitivity for Pointer Analysis** (George Kastrinis, Yannis Smaragdakis, Martin Benvenuto, Anders Møller, 2010)
- Precision-Guided Context Sensitivity for Introspective Analysis: Context-Sensitivity, Across the Board** (Yannis Smaragdakis, George Kastrinis, George Balatsouros, 2010)
- A Machine-Learning Algorithm with Disjunctive Model for Data-Driven Program Analysis** (Minseok Jeon, Sehun Jeong, Sungdeok Cha, Hakjoo Oh, 2018)

Each paper includes its title, authors, publication year, and a brief abstract or summary of its main contributions.

1981

2002

2010

2022

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 collage consists of six academic papers arranged in two rows of three:

- Making k -Object-Sensitive Po More Precise with Still k** (1981): Abstract discusses pointer analysis in object-oriented languages.
- Scalability-First Pointer Analysis Self-Tuning Context-Sensitivity** (1981): Abstract discusses context-sensitivity and its relation to pointer analysis.
- Pick Your Contexts Well: Understanding the Making of a Precise and Scalable** (1981): Abstract discusses context-sensitivity and its relation to pointer analysis.
- Hybrid Context-Sensitivity for Program Analysis** (1981): Abstract discusses hybrid context-sensitivity.
- Introspective Analysis: Context-Sensitivity, Across the Board** (1981): Abstract discusses introspective analysis and context-sensitivity.
- A Machine-Learning Algorithm with Disjunctive Model for Data-Driven Program Analysis** (2019): Abstract discusses a machine-learning algorithm for data-driven program analysis.

Each paper includes a small image of its cover or a snippet of its abstract.

1981

2002

2010

2022

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]

Making k -Object-Sensitive Po More Precise with Still k
Tian Tan¹, Yue Li¹, and Jingling Zhang²
¹ School of Computer Science and Engineering
² Advanced Innovation Center for Imaging Technologies

Abstract: Object-sensitivity is regarded as a good abstraction for pointer analysis in object-oriented languages. It uses a set of objects (as k context elements) to represent a memory location, which may end up using some context elements resulting from a finer partition of the space (of concrete) method call. In this paper, we introduce BEAN, an improved k -object-sensitive pointer analysis technique by using a k -unitizing context abstraction. The allocation sites that are redundant context elements of Object Allocation Graph (OAG), which is built based on a context-insensitive Analysis (BEAN) analysis, are removed to make the OAG more sparse. We present the BEAN framework that addresses this problem. The experimental results demonstrate that BEAN achieves practical scalability for all the evaluated programs (e.g., speedups by 2 for k -object-sensitivity).

CCS CONCEPTS: Theory of computation → Program analysis;

KEYWORDS: pointer analysis, context-sensitivity, k -object-sensitivity, type-sensitivity.

Pick Your Contexts Well: Understanding The Making of a Precise and Scalable Pointer Analysis
Yue Li¹, Tian Tan¹, Anders Møller², and Yannis Smaragdakis³
¹ Aarhus University, ²Aarhus University, ³University of Athens, Greece

Abstract: Context-sensitivity is important in pointer analysis to ensure high-quality analyses, but existing techniques suffer from unpredictable scalability. Many variants of context-sensitivity exist, and it is difficult to choose one that leads to reasonable analysis time and obtains high precision. For a context depth of 2 or higher, most scalable implementations deviate significantly from the original definition of an object-sensitive analysis, which is often less precise and less efficient at every method call and object creation. We show that a selective context-sensitive analysis can be much more efficient than a full object-sensitive analysis at a small cost in terms of precision. We propose a hybrid context-sensitive analysis that combines a selective context analysis with a full object-sensitive analysis. We present the first implementation of this hybrid analysis and evaluate its performance.

Hybrid Context-Sensitivity for Pointer Analysis
Martin Bawdenboer¹, George Kastrinis², Yannis Smaragdakis³, and George Balatsouros⁴
¹ LogicBlox Inc., ²University of Massachusetts, Amherst, MA 01003, USA, ³University of Athens, Greece, ⁴University of Athens, 15784, Greece

Abstract: Context-sensitive pointer analysis is valuable for achieving high precision with good performance. The standard flavors of context-sensitivity are context-insensitive (CICA) and context-sensitive. Context-sensitive flavor of pointer analysis is more precise than CICA, but at an infeasibly high cost. We show that a selective context-sensitive analysis is highly profitable. Namely, by keeping a combined context only when analyzing selected language features, we can closely approximate a full object-sensitive analysis while maintaining high precision at all times. In terms of speed, the selective combination of both kinds of context not only vastly outperforms non-selective context-sensitive analysis, but also matches the speed of a full object-sensitive pointer analysis. We offer a clean model for hybrid context-sensitive pointer analysis, and we prove that this result holds for a large array of analyses (e.g., 1-object-sensitive, 2-object-sensitive, etc.). Our experiments show that this hybrid analysis achieves a new level of performance compared to previous approaches.

Precision-Guided Context Sensitivity for Pointer Analysis
George Kastrinis¹, Yannis Smaragdakis², and George Balatsouros³
¹ Department of Informatics, University of Athens, ²{gkastriis,ysmaragd}@di.uoa.gr, ³balats@hua.gr

Abstract: Context-sensitive pointer analysis is a primary approach for adding more precision to a pointer-to analysis, by hopefully also maintaining scalability. An off-the-shelf pointer-to analysis is typically quadratic or worse in the number of pointers, and thus it is not feasible to maintain precision. Indeed, precision and maintaining precision, indeed, are in conflict. We propose a hybrid approach that maintains precision only on a subset of pointers, and maintains precision only on a subset of pointers. This is done by identifying points-to sets that lead to less work [1].

Introspective Analysis: Context-Sensitivity, Across the Board
YUE LI¹, Tian Tan¹, Anders Møller², and Yannis Smaragdakis³
¹Aarhus University, Denmark, ²Aarhus University, Denmark, ³University of Athens, Greece

Abstract: Context sensitivity is a common way of parallelizing pointer analysis. However, it is well-known that context sensitivity is not always the best choice for achieving high precision. We observed that applying context sensitivity partially only on a select balance between analysis precision and speed. However, existing tools do not provide much insight into what characterizes this method. We propose an introspective approach for identifying precision-critical methods, based on an analysis of the precision patterns in the code. This is because smaller points-to sets lead to less work [1].

Categories and Subject Descriptors: F.3.2 [Logics and Meanings of Programs]: Semantics of Programming Languages—Program Analysis; F.3.3 [Programming Languages]: Formal Definitions and Theories—Semantics

General Terms: Algorithms, Languages, Performance.

Keywords: pointer-to analysis, context-sensitivity, object-sensitivity, type-sensitivity.



1981

2002

2010

2022

1981

2002

2010

2018

2022

컨텍스트 터널링



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.

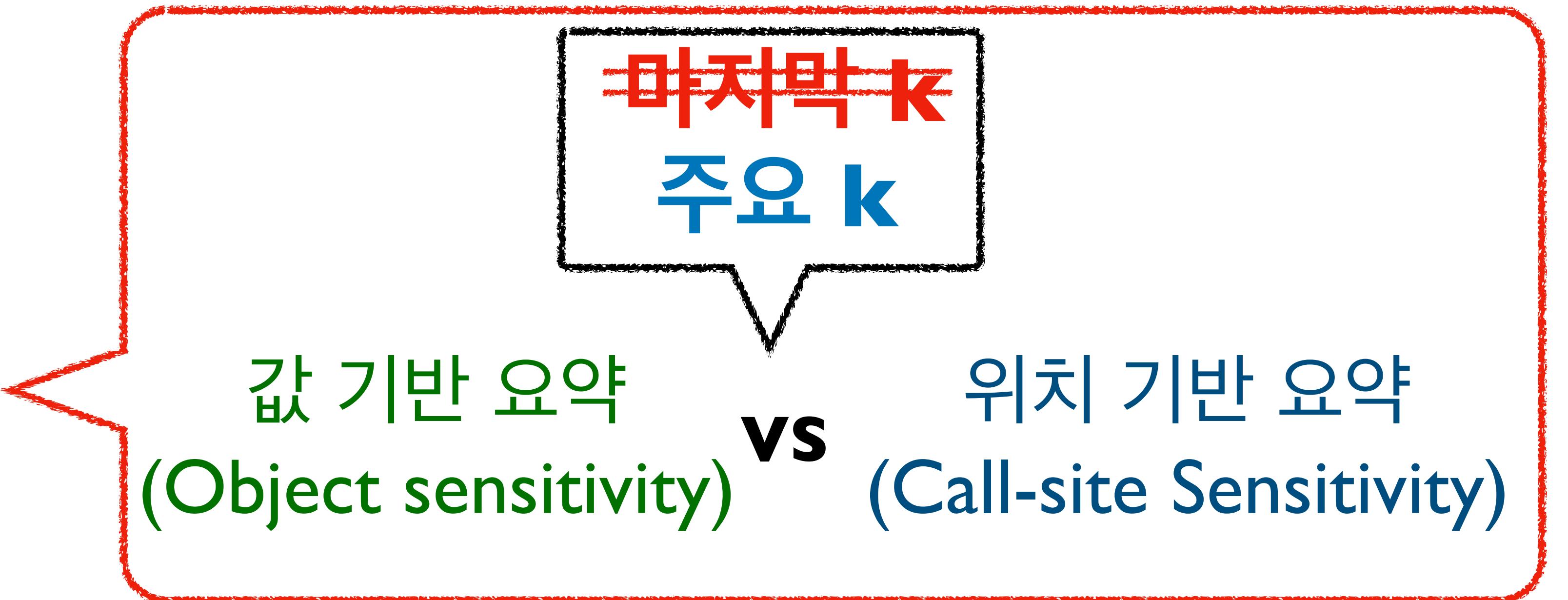
1 INTRODUCTION

“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 “object sensitivity is the dominant flavor of context sensitivity for object-oriented programs.” Sharir et al. [1981] propose a context-sensitive pointer analysis for C programs. The analysis 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



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.

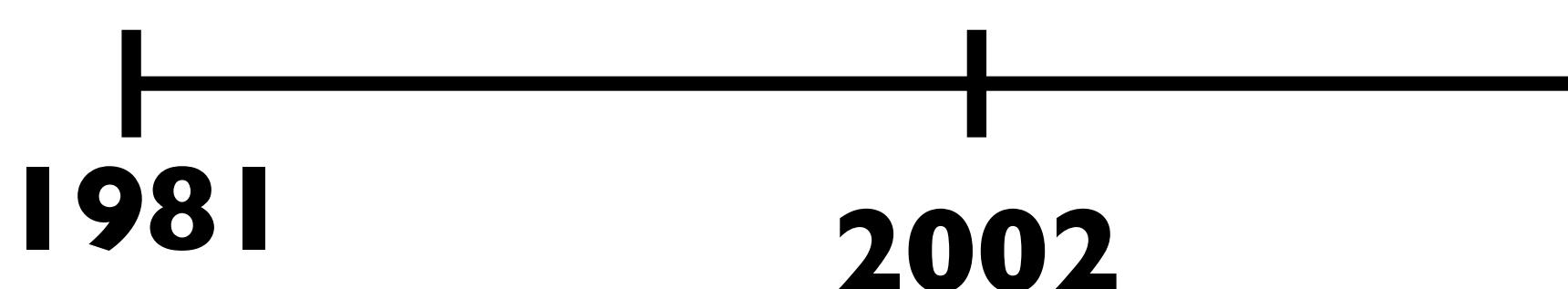
1 INTRODUCTION

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

—Smaragdakis and Balatsouras [2015]

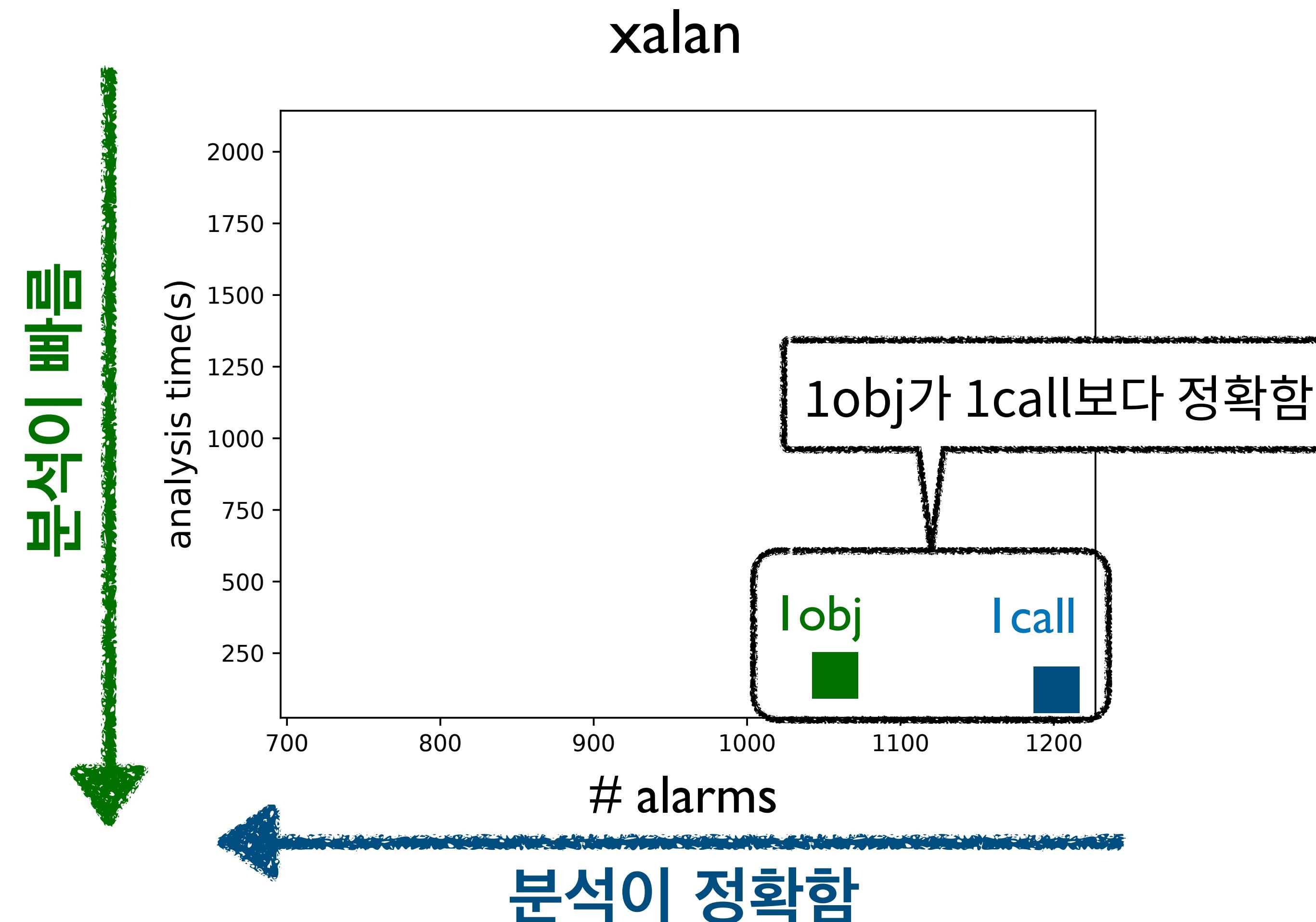
Context sensitivity is critically important for static program analysis of object-oriented programs. A context-sensitive analysis associates local variables and heap objects with context information of method calls, computing analysis results separately for different contexts. This way, context sensitivity 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 for C programs. The analysis 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



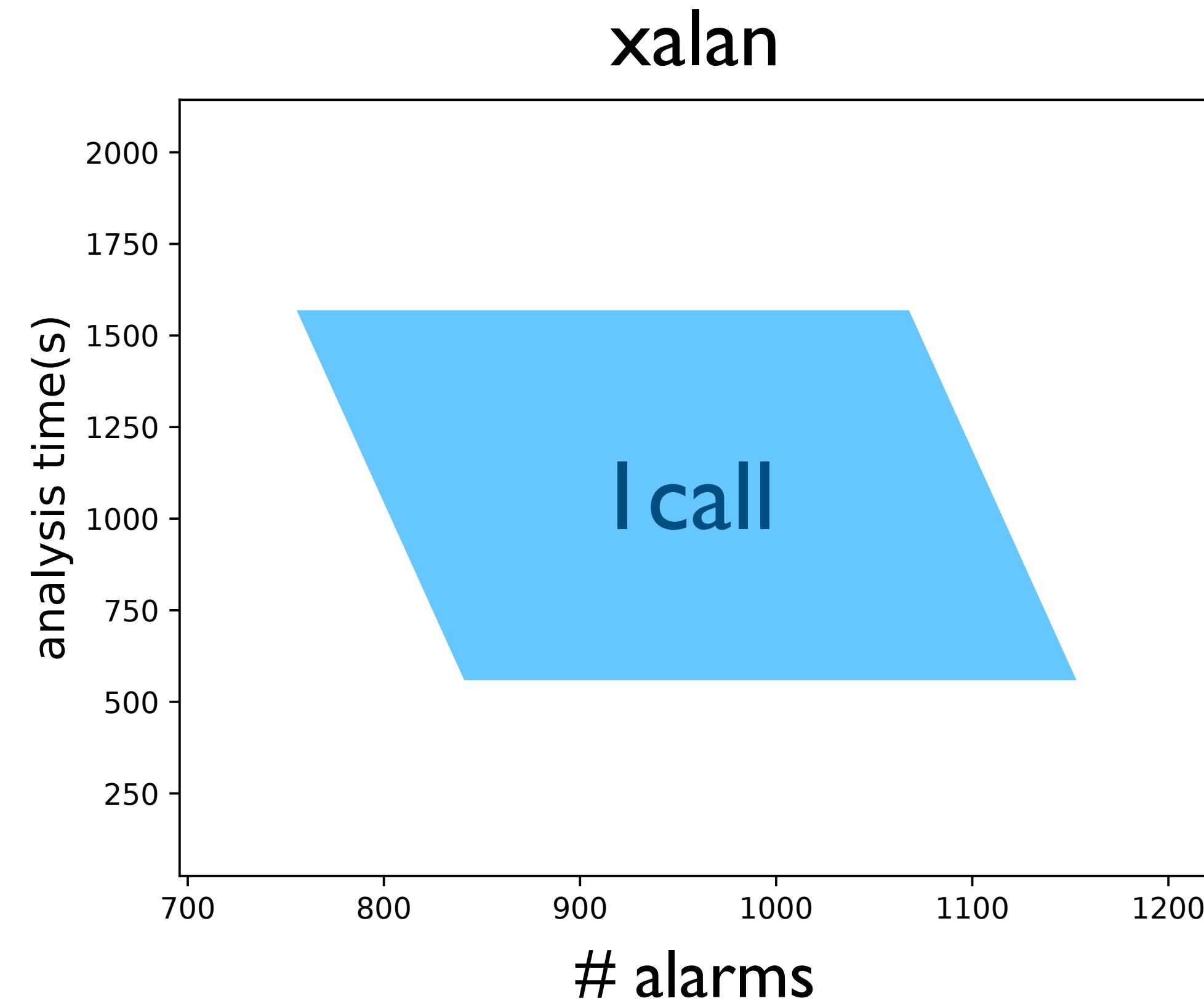
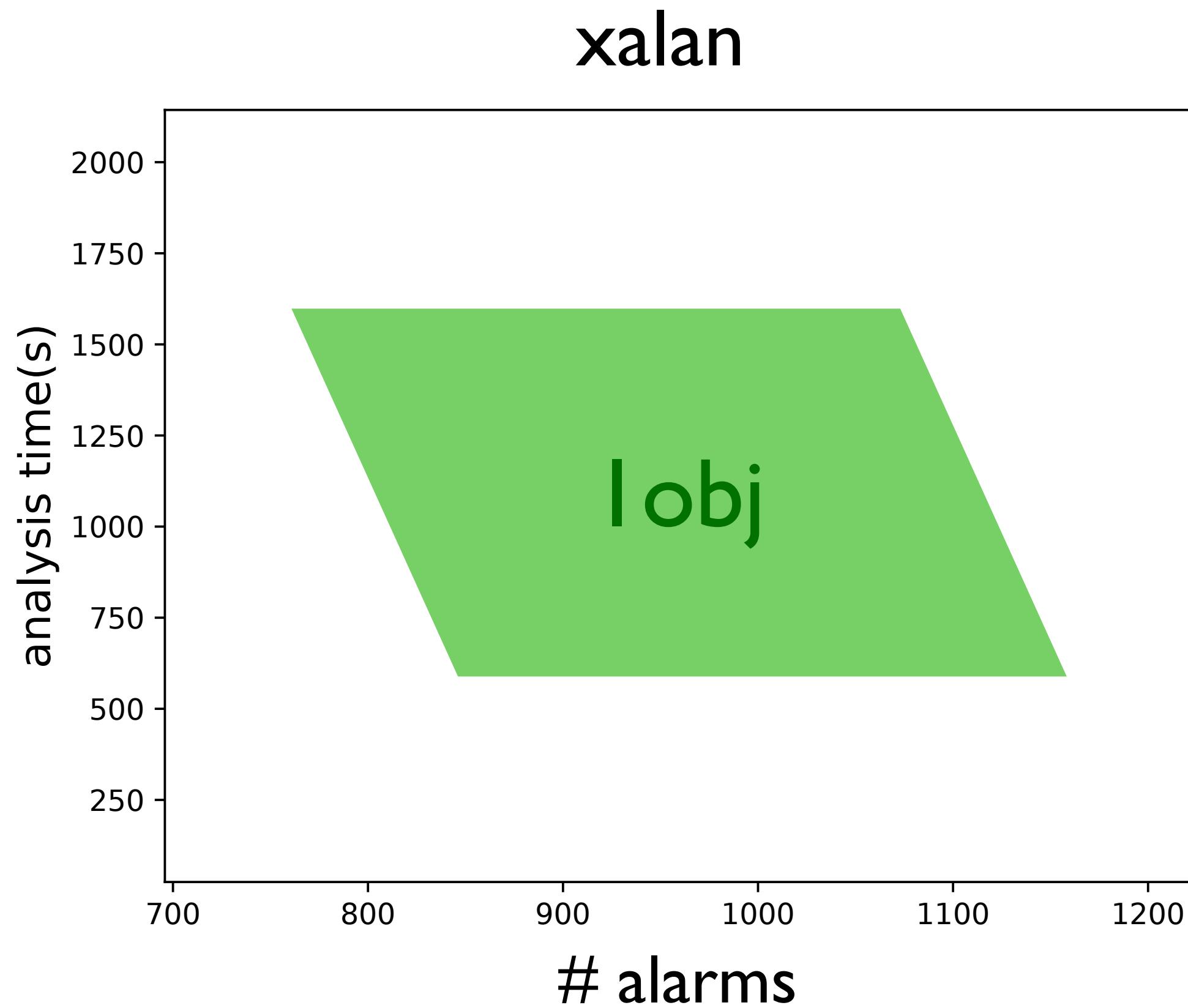
마지막 k 기반 요약에서의 성능 비교

- 분석의 성능을 직접 비교하면 됨



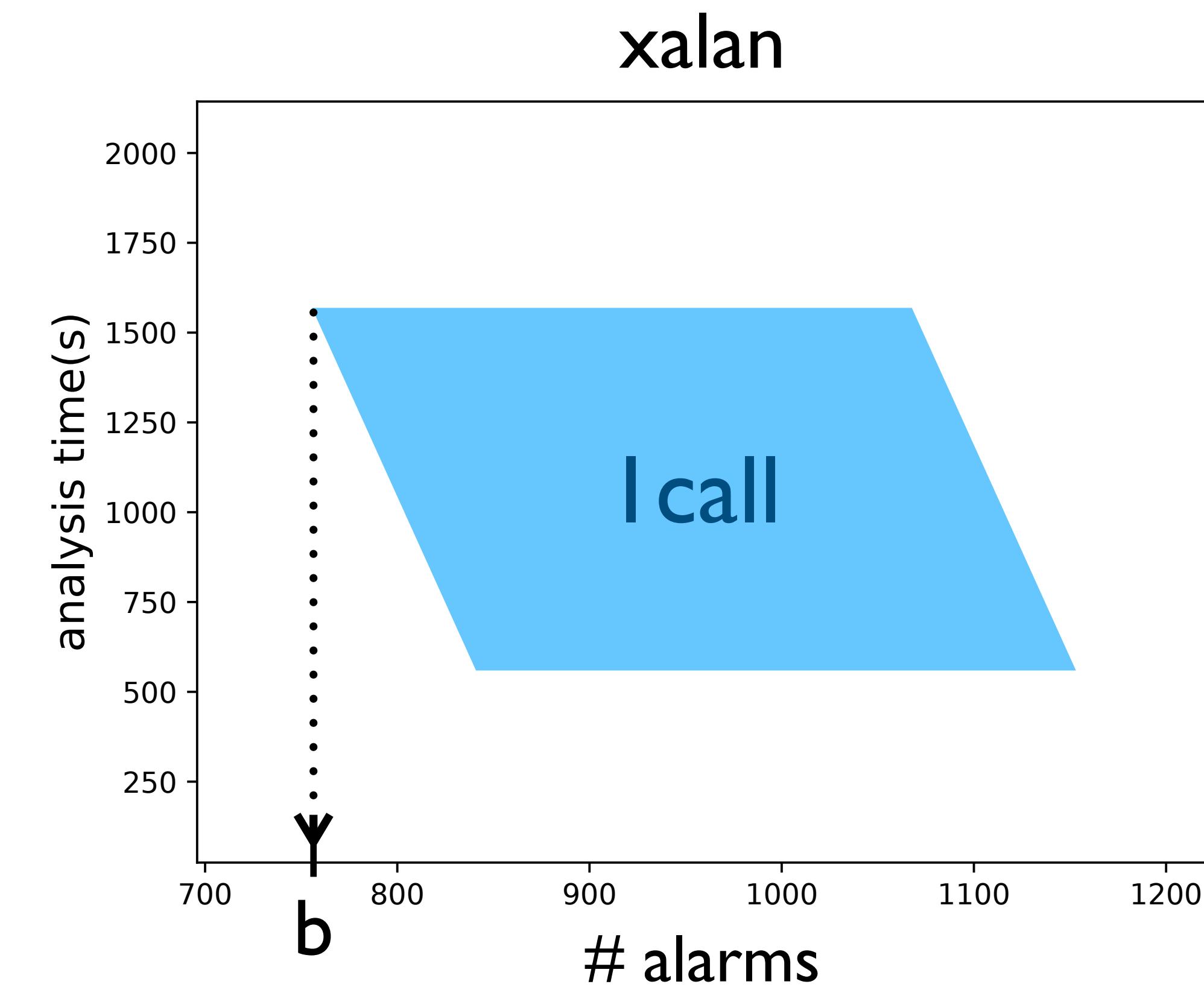
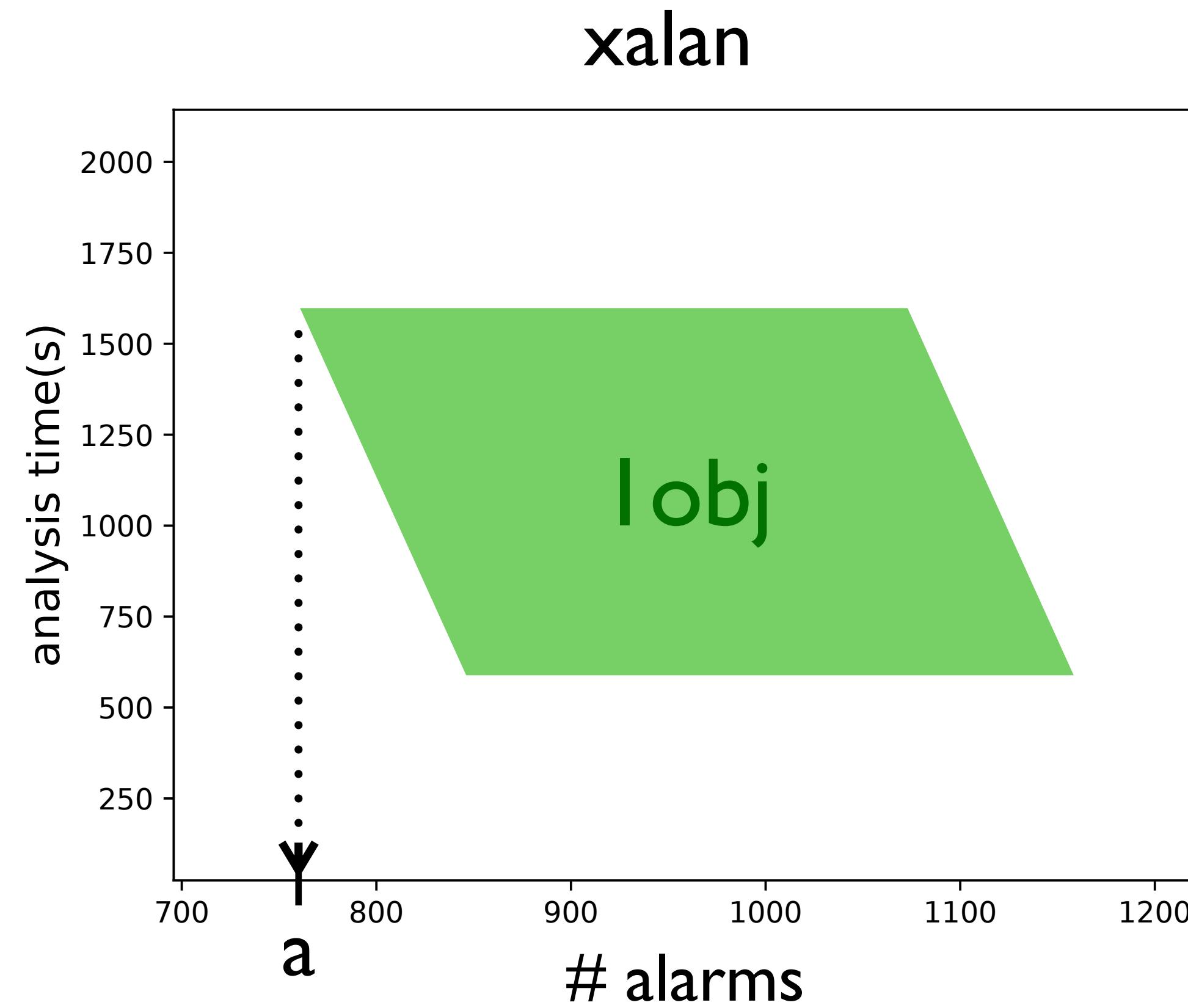
주요 k 기반에서의 성능 비교

- 주요 요소 분류에 따라 성능이 바뀜



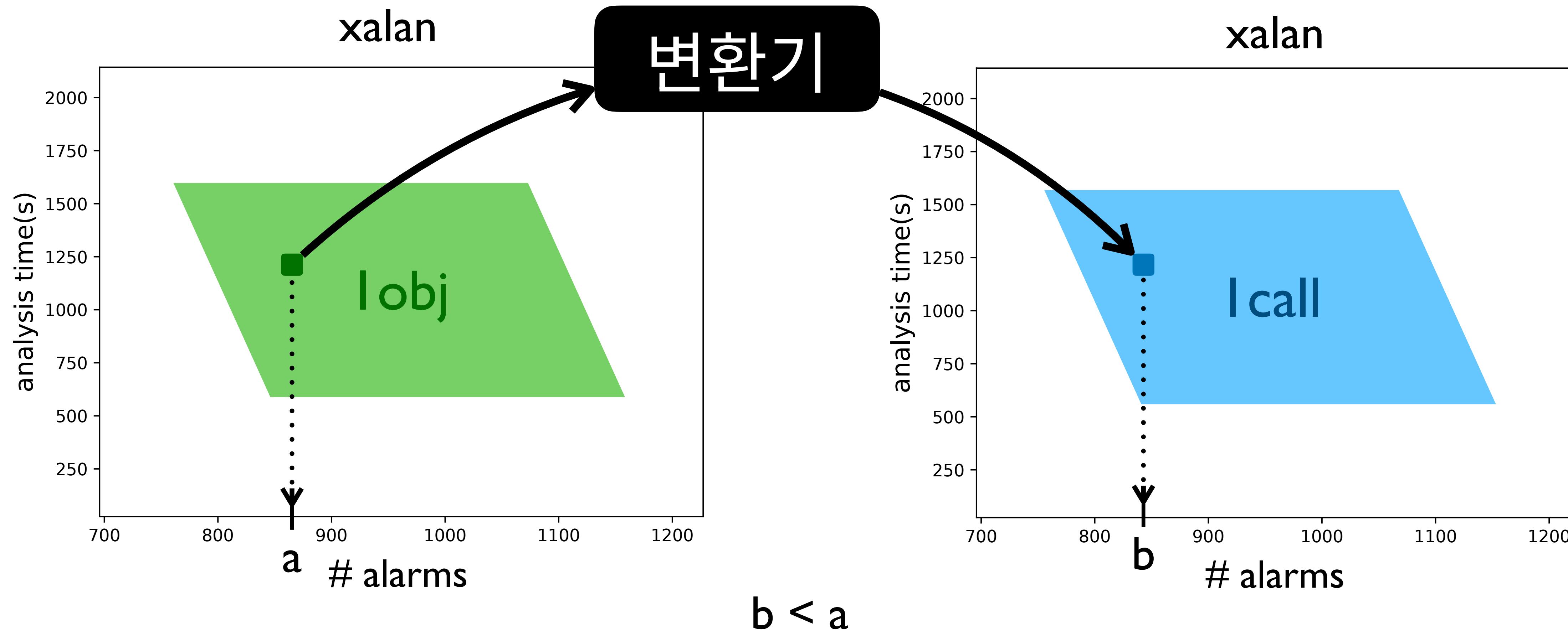
주요 k 기반에서의 성능 비교

- 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



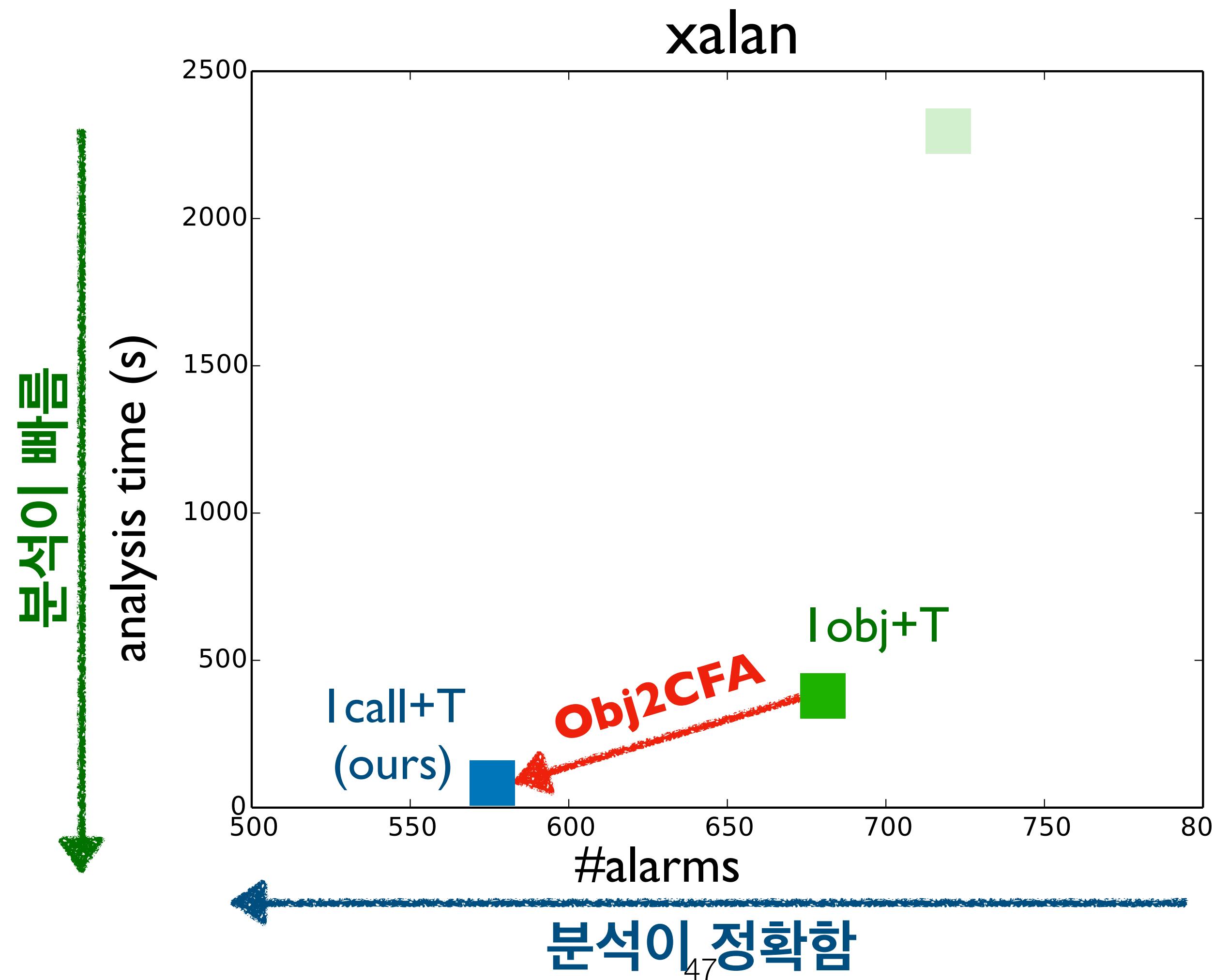
변환기 : Obj2CFA

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



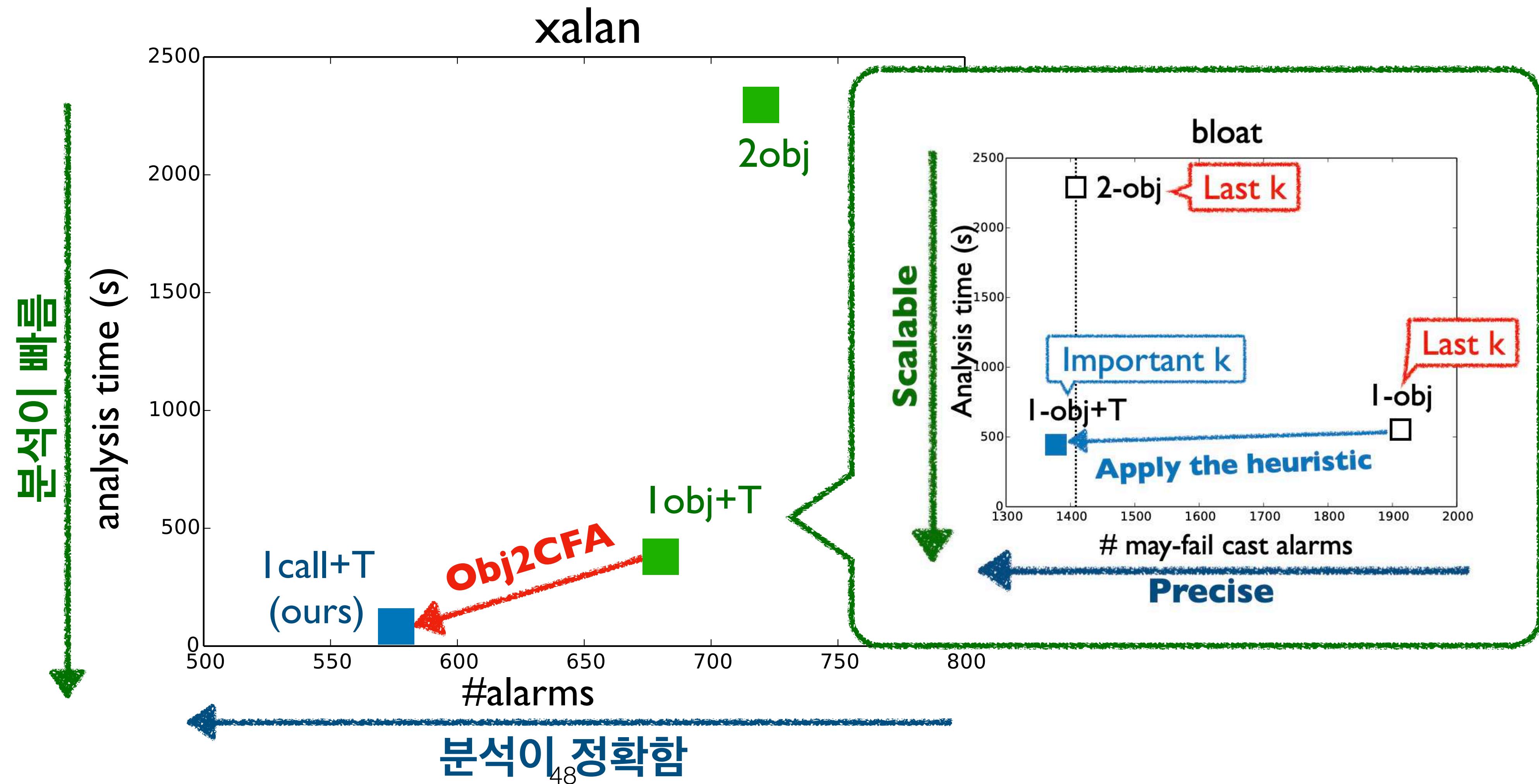
변환기 : Obj2CFA

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



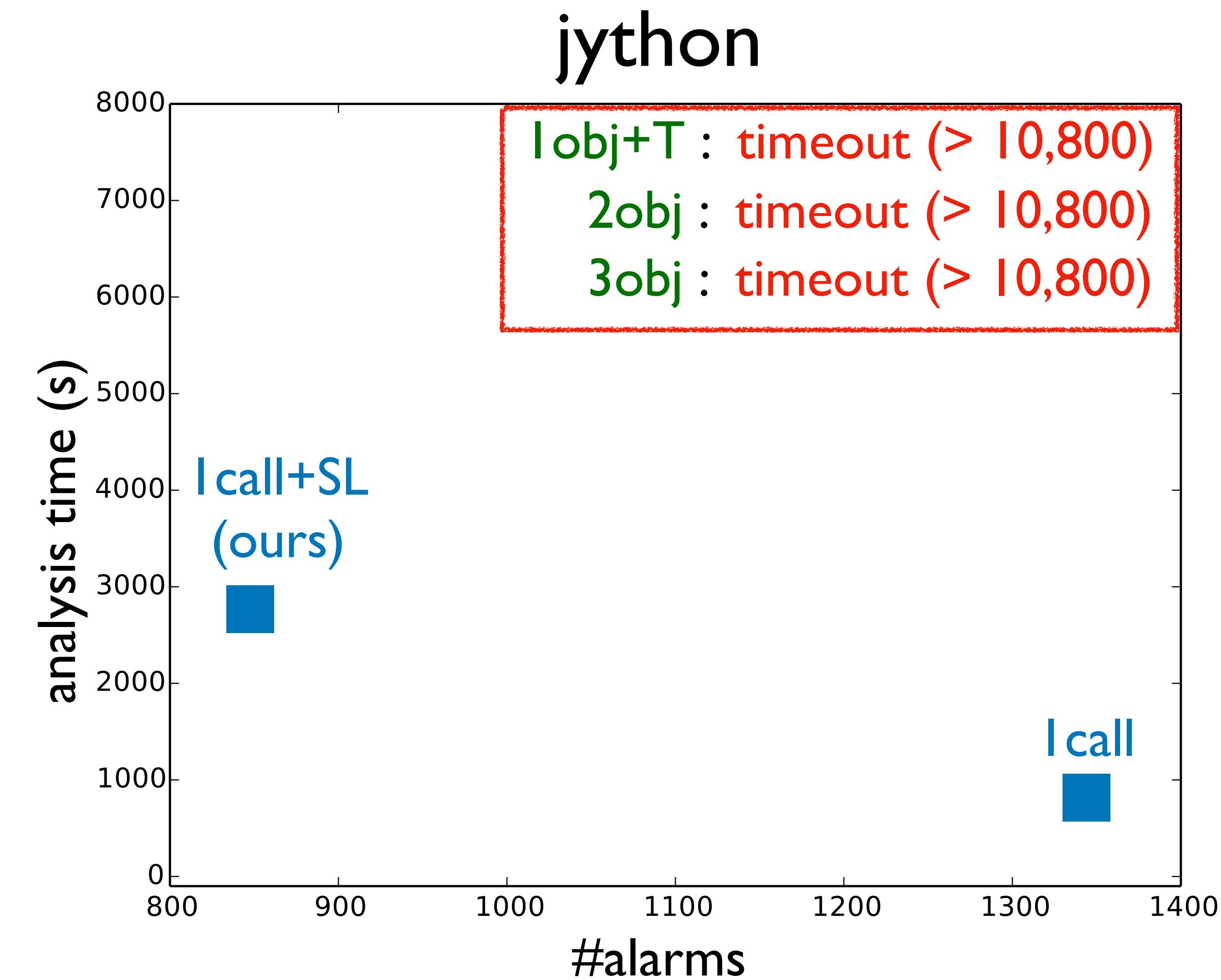
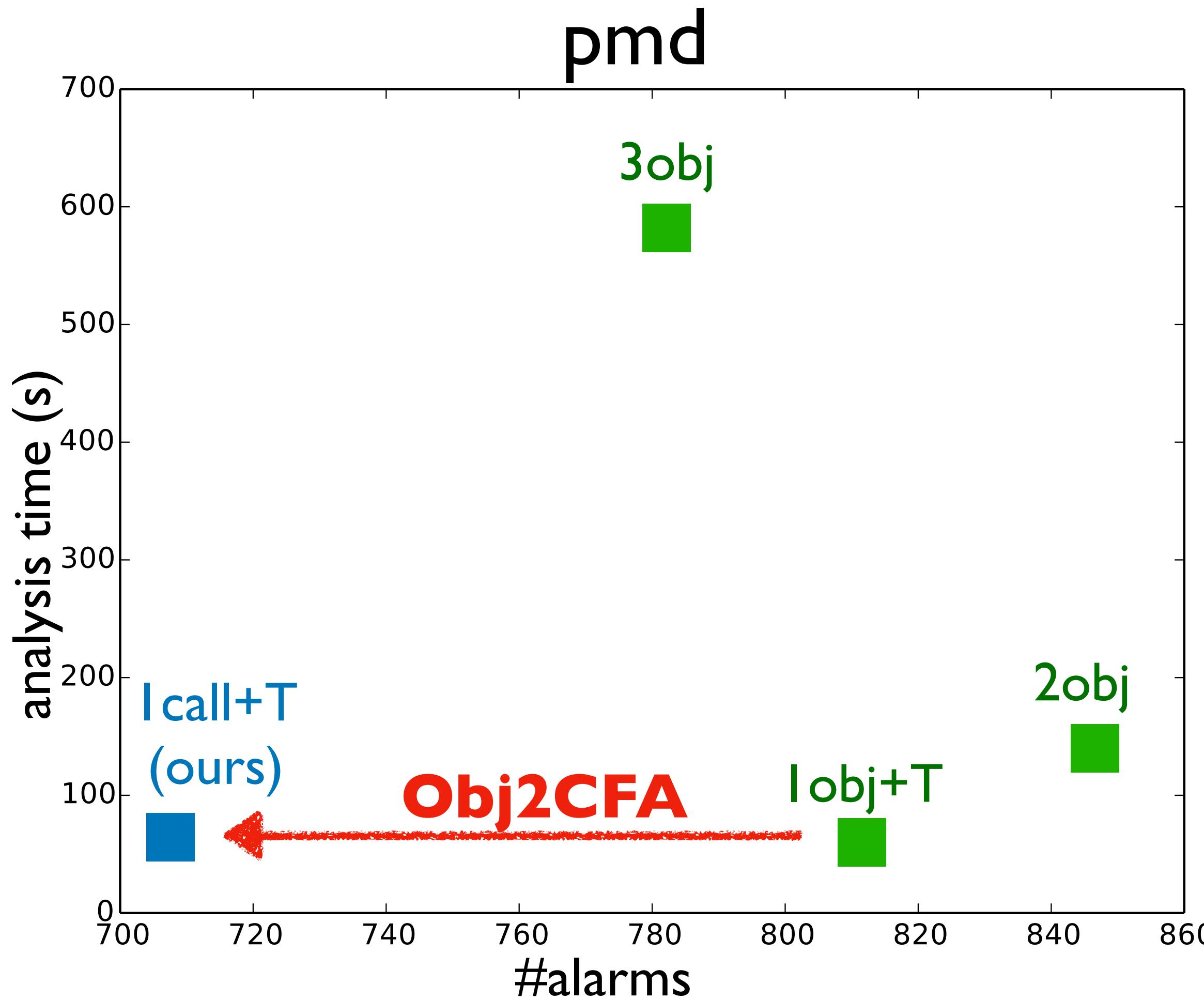
변환기 : Obj2CFA

- **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

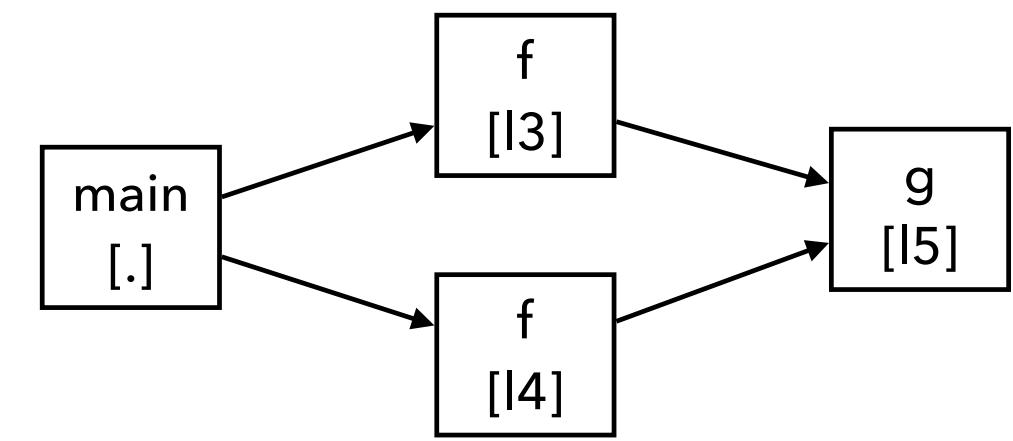
- In theory, their precision is incomparable
- In practice, object sensitivity generally outperforms call-site sensitivity for OO languages (like Java)

Call-site vs. Object Sensitivity

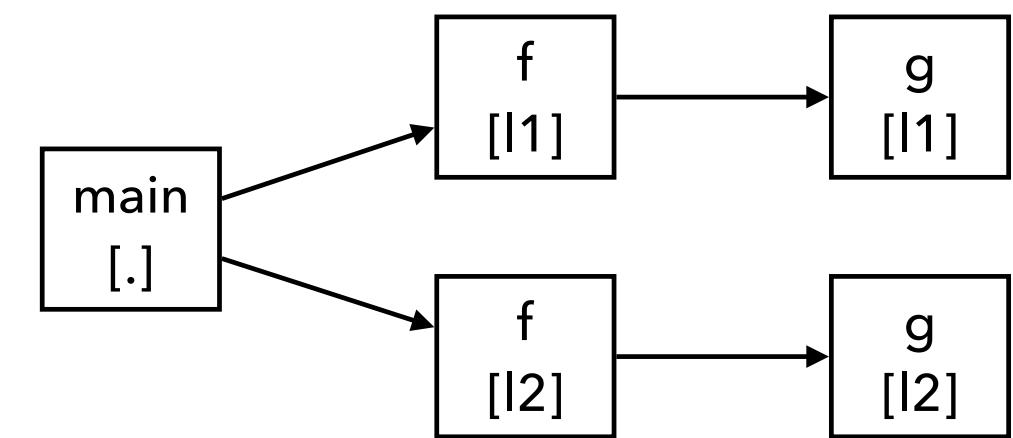
- Typical example that benefits from object sensitivity:

```
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
```



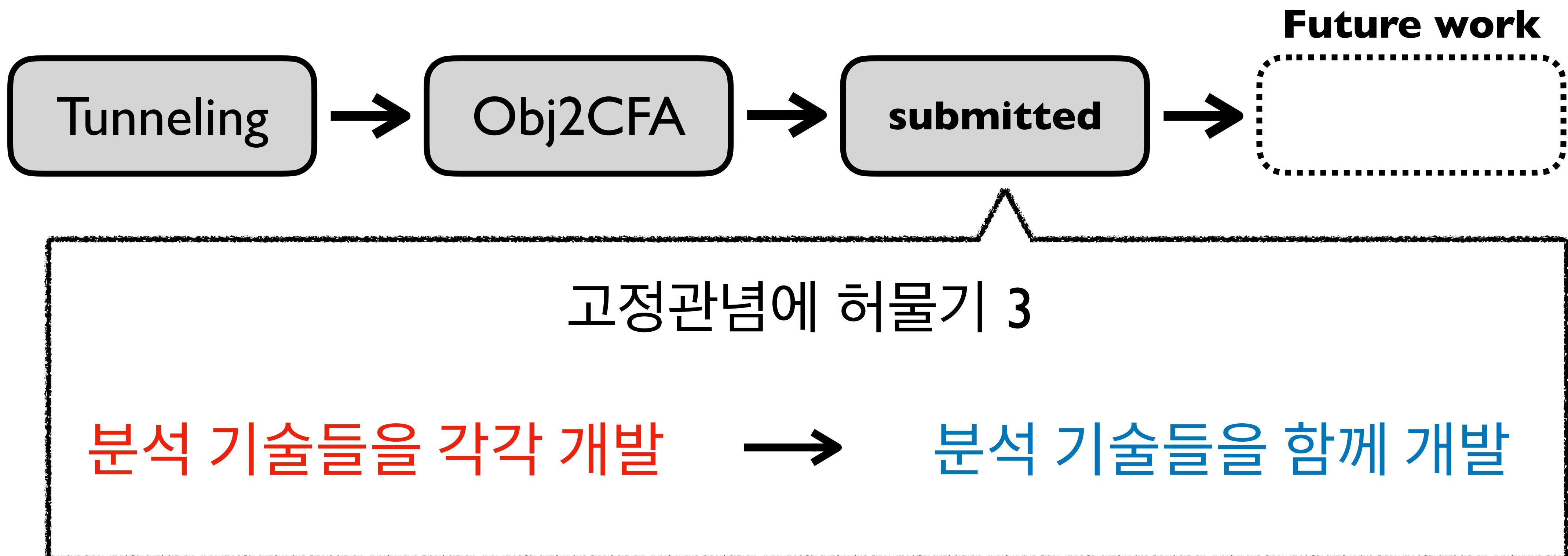
1-call-site sensitivity



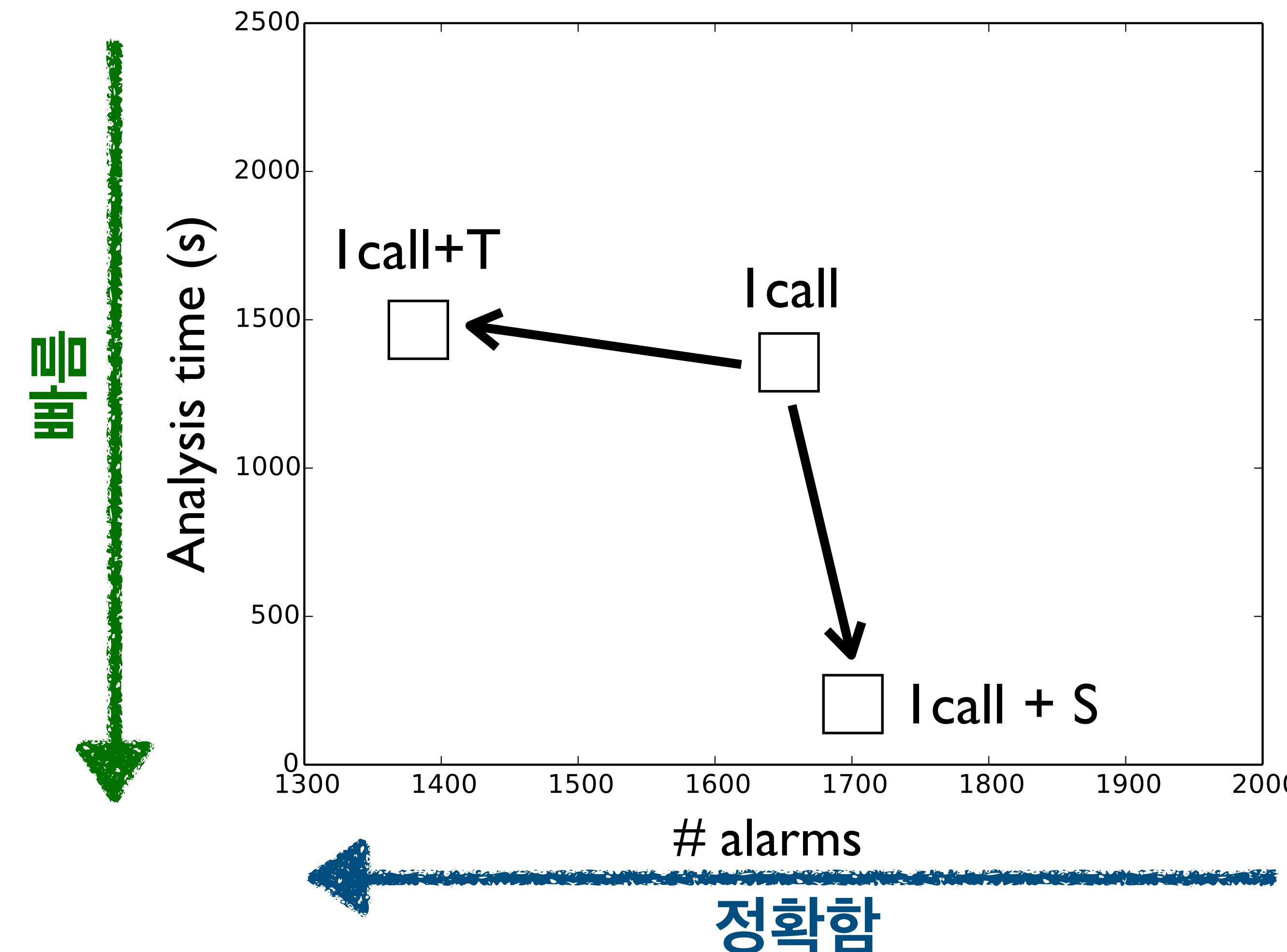
1-object sensitivity

고정관념에 도전하기

- 목표: 주요 k기반 요약 방식을 표준으로 만들기

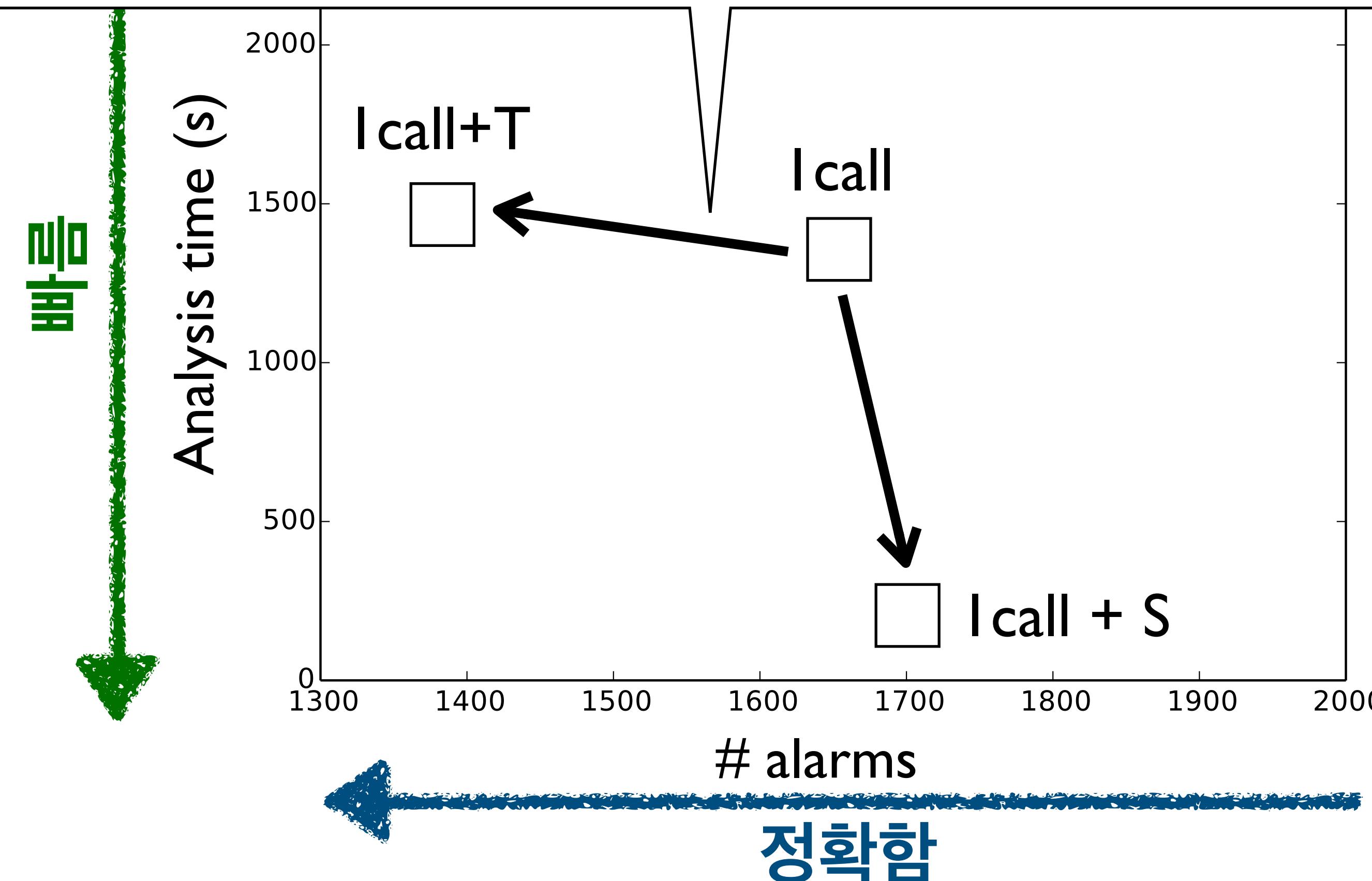


- 호출 환경 배달하기: 속도를 유지한채 정확도를 올려주는 기술
- 적당히 함수 호출 분석 (selective ctx sensitivity): 정확도를 유지한채 속도를 올려주는 기술

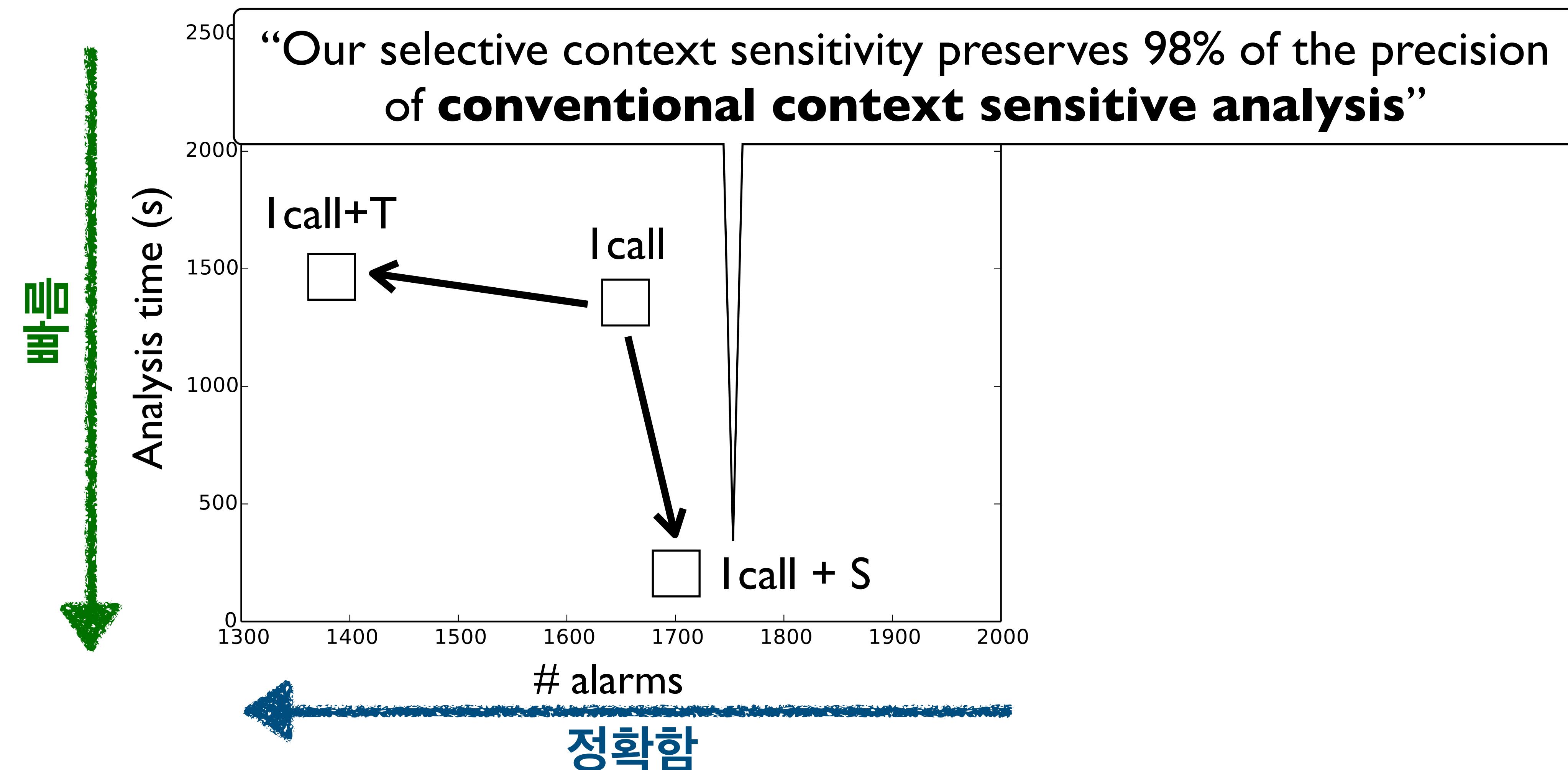


- 호출 환경 배달하기: 속도를 유지한채 정확도를 올려주는 기술

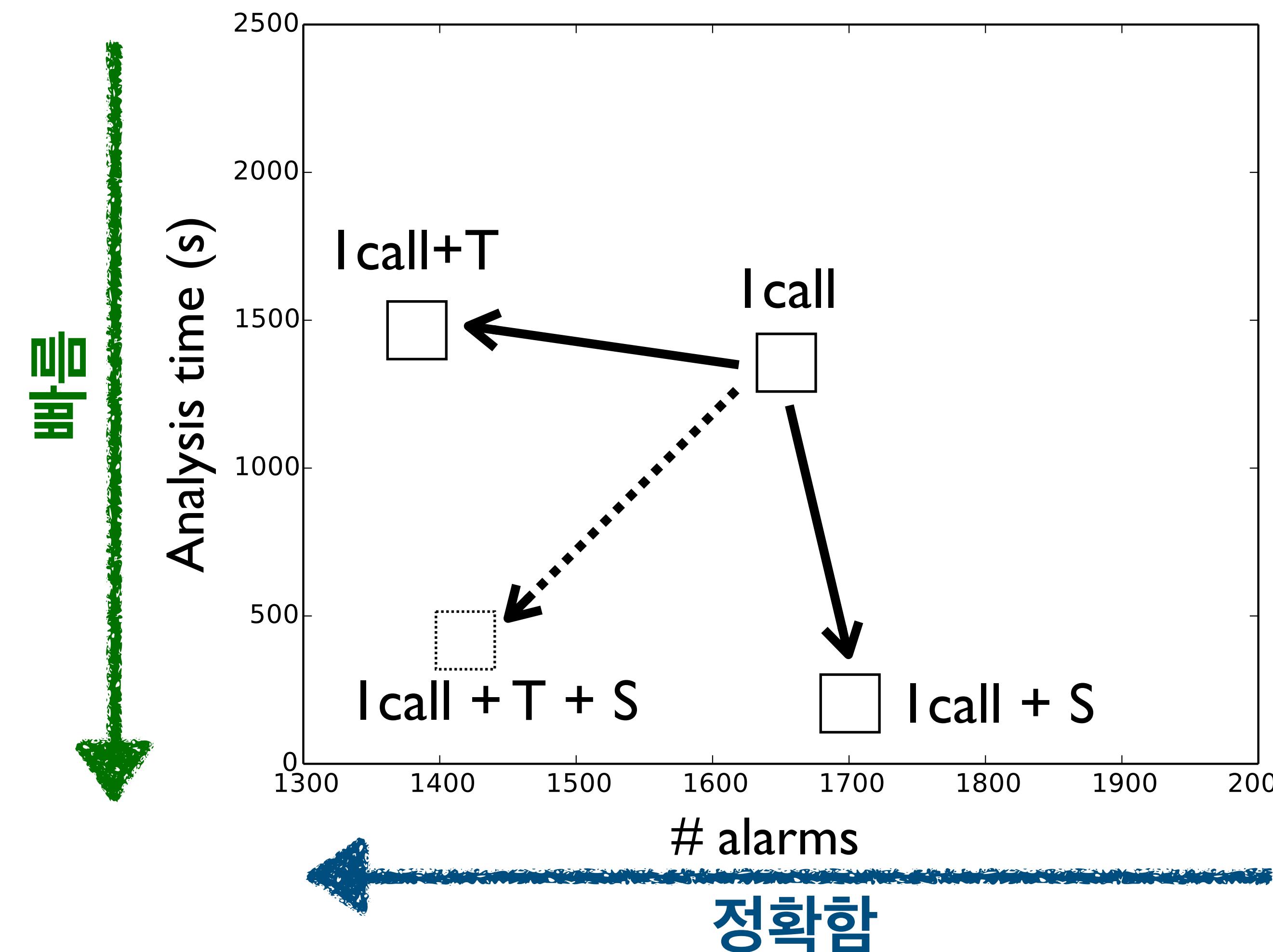
“We generated **Icall+T** by applying context tunneling to **Icall...**”



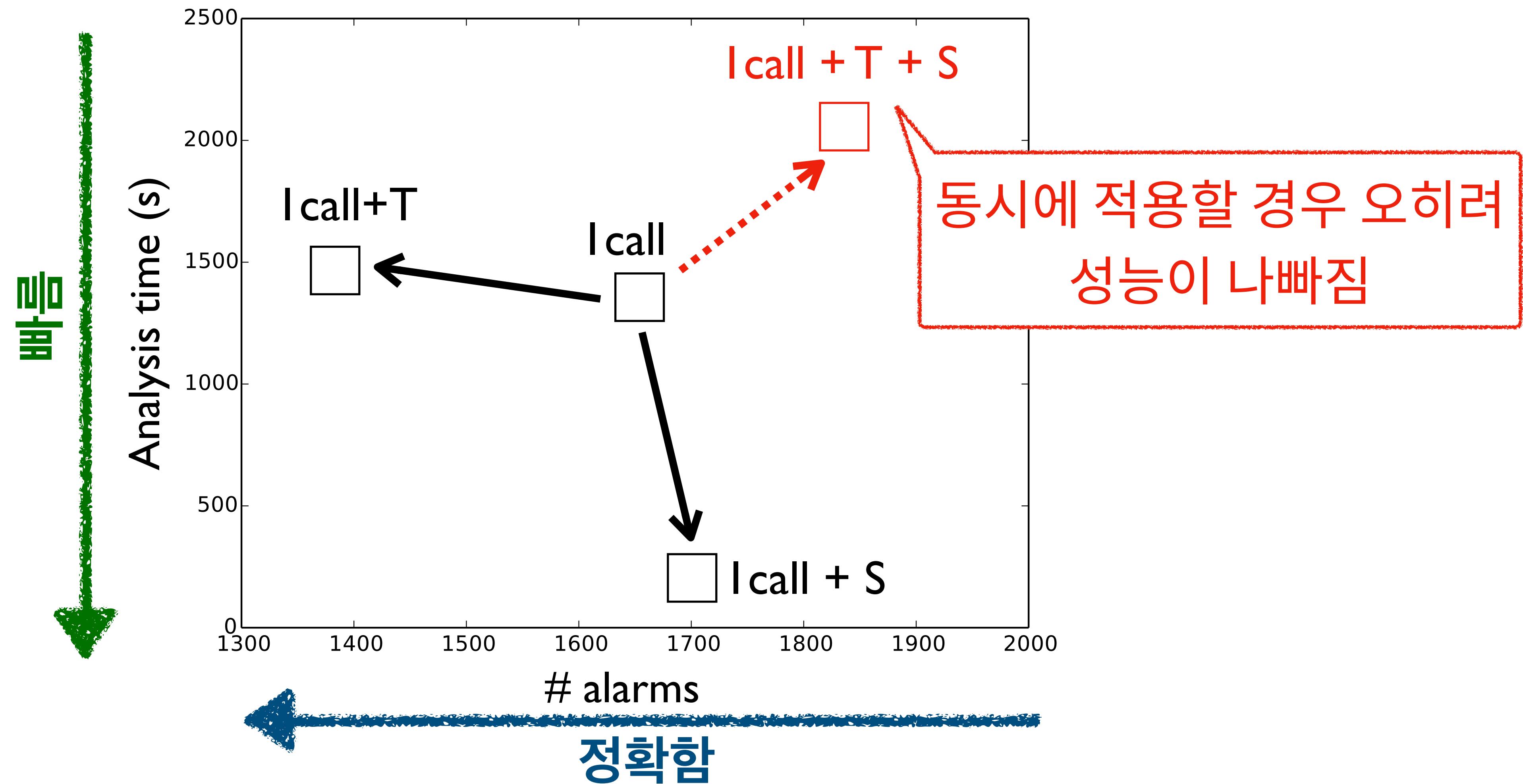
- 적당히 함수 호출 분석: 정확도를 유지한채 속도를 올려주는 기술



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



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

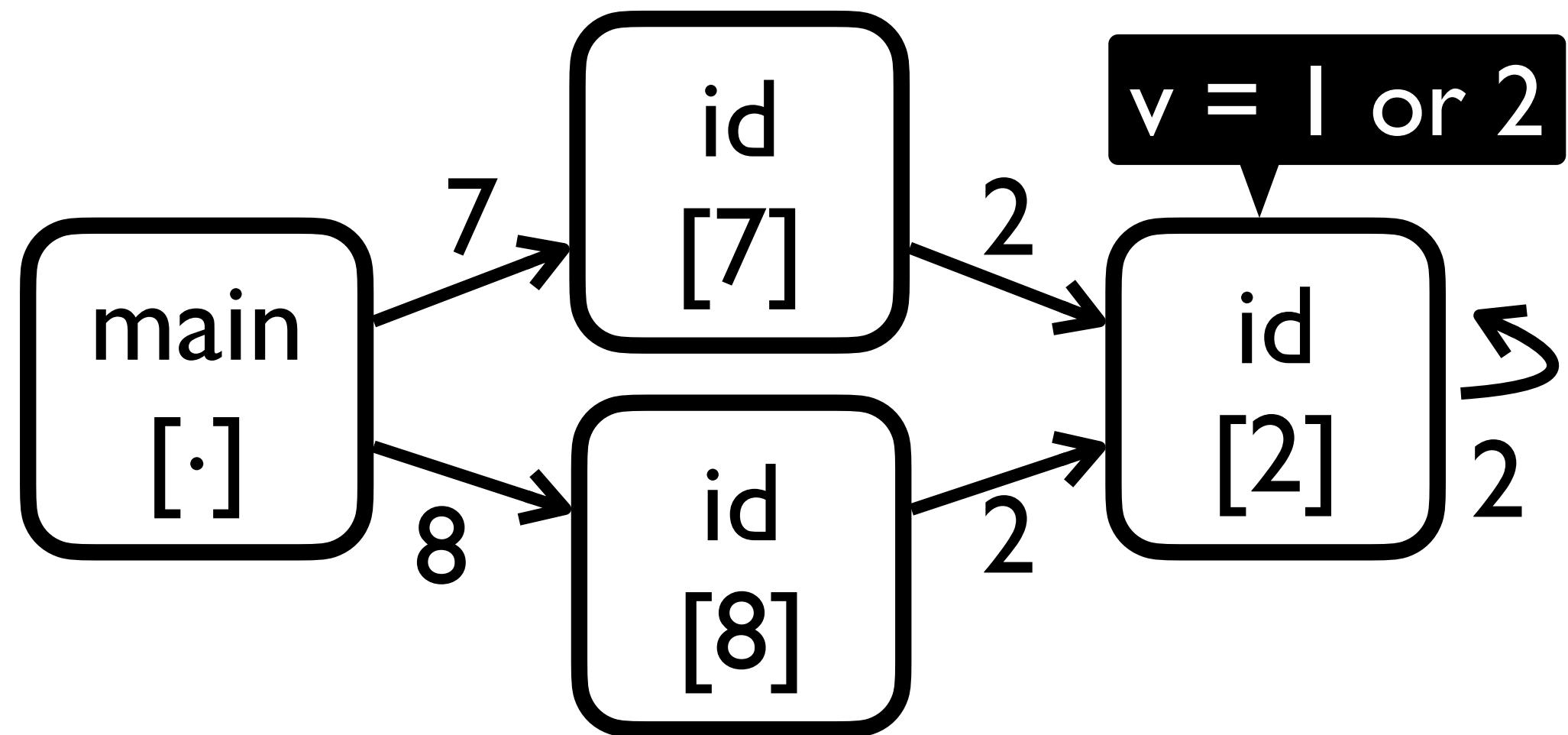


```

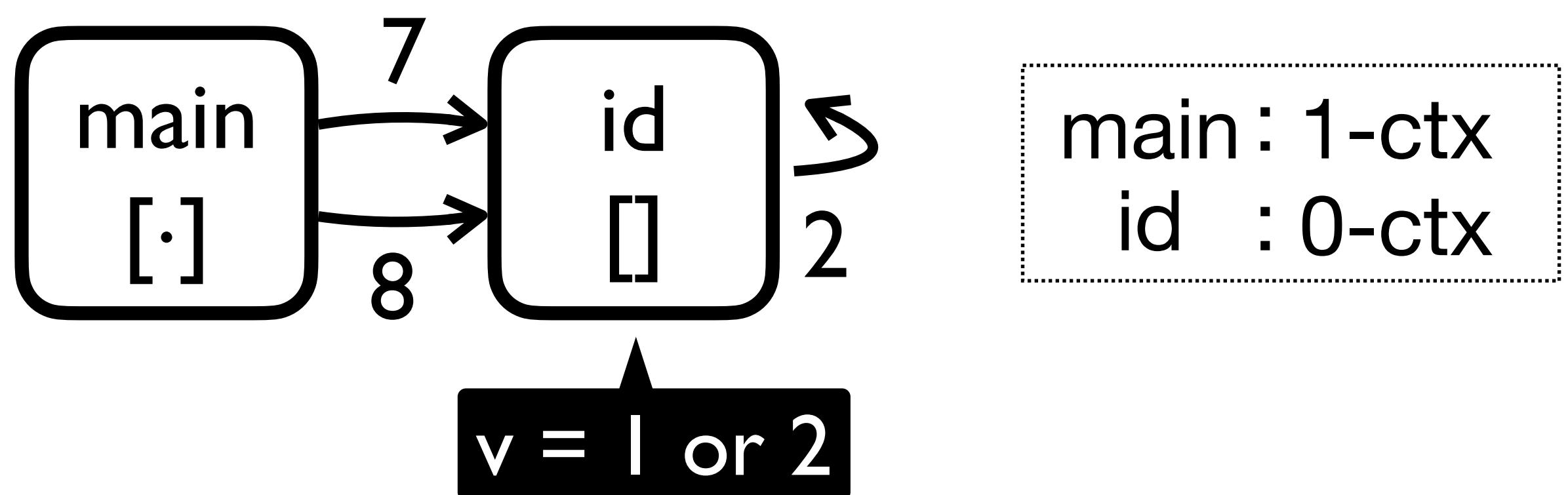
0: id(v, i){
1:   if (i > 0){
2:     return id(v, i-1);}
3:   return v;}
4:
5: main(){
6:   i = input();
7:   v1 = id(1, i); //A
8:   v2 = id(2, i); //B
9:   assert (v1 != v2); //query
10: }

```

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;

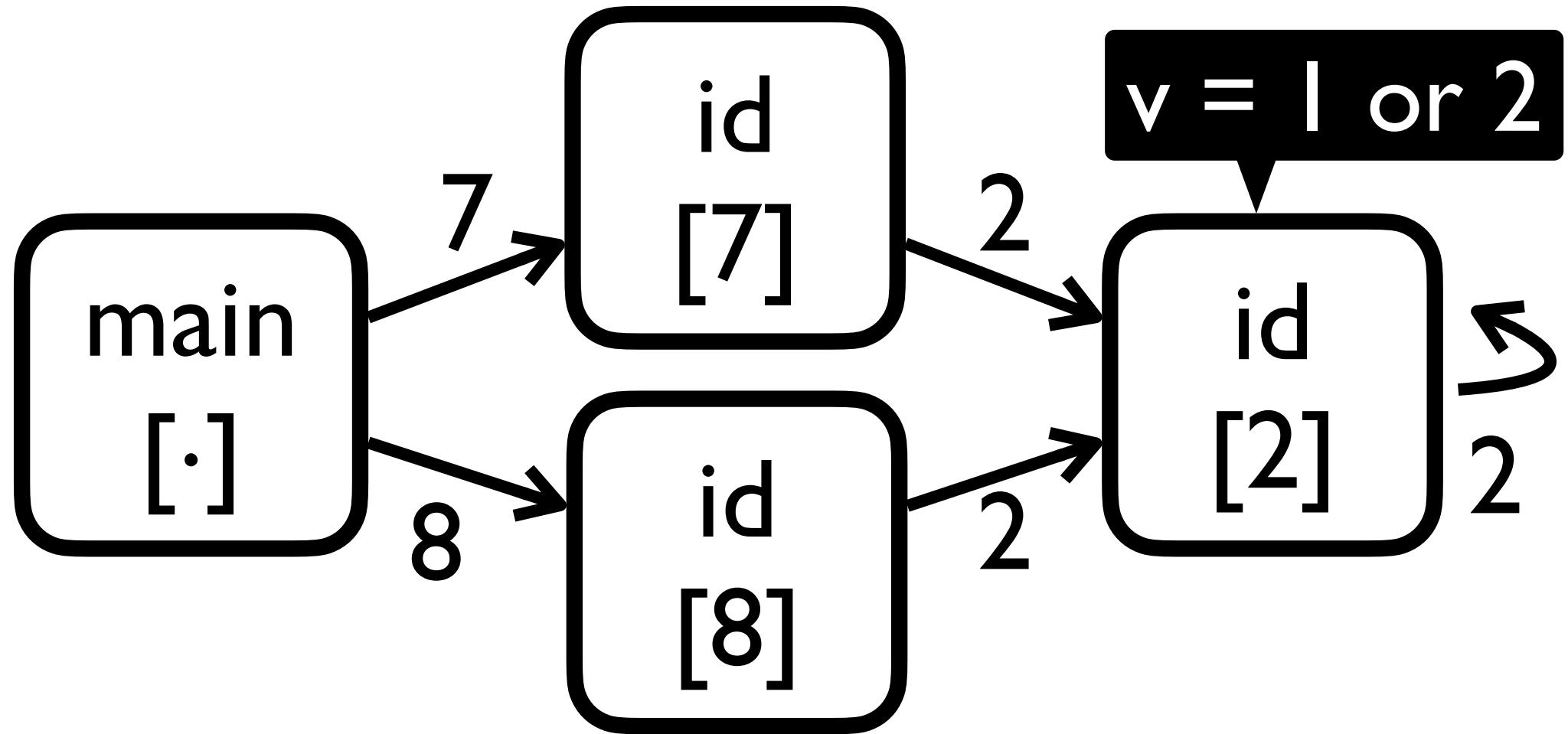
```

```

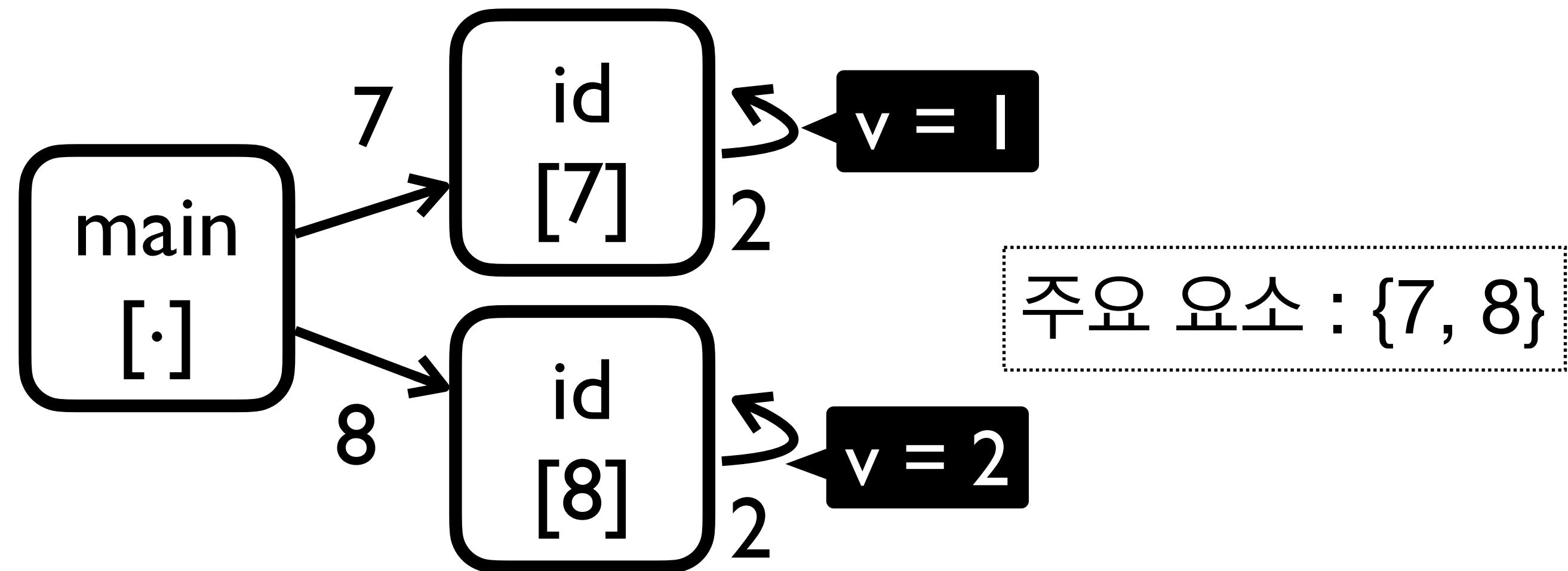
4:
5: main(){
6:   i = input();
7:   v1 = id(1, i); //A
8:   v2 = id(2, i); //B
9:   assert (v1 != v2); //query
10: }

```

예제 프로그램



I-요소 기반 함수 호출 요약



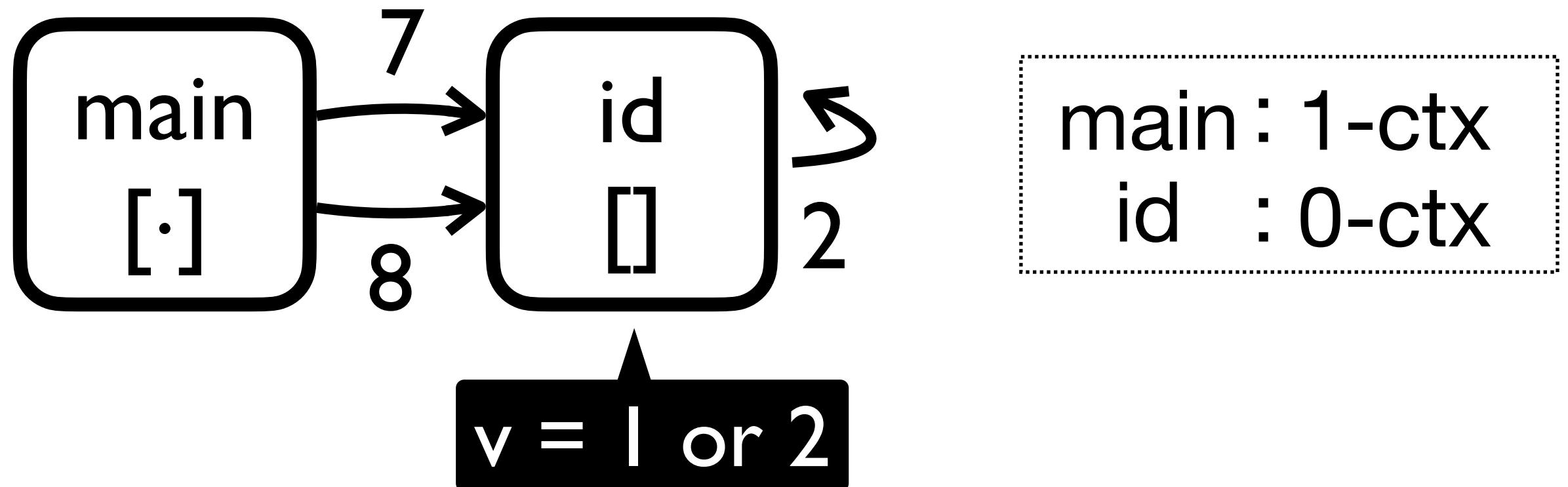
주요 I-요소 기반 함수 호출 요약

```

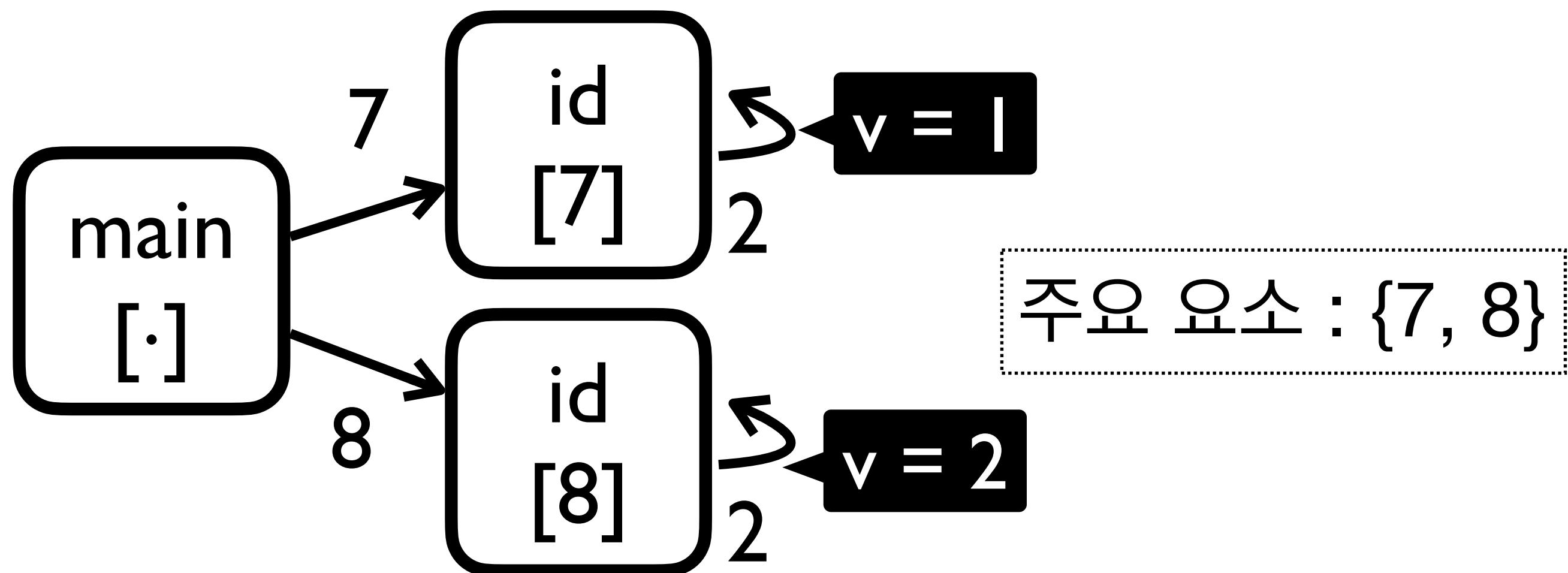
0: id(v, i){
1:   if (i > 0){
2:     return id(v, i-1);
3:   return v;
4:
5: main(){
6:   i = input();
7:   v1 = id(1, i); //A
8:   v2 = id(2, i); //B
9:   assert (v1 != v2); //query
10: }

```

예제 프로그램



적당히 I-요소 기반 함수 호출 요약



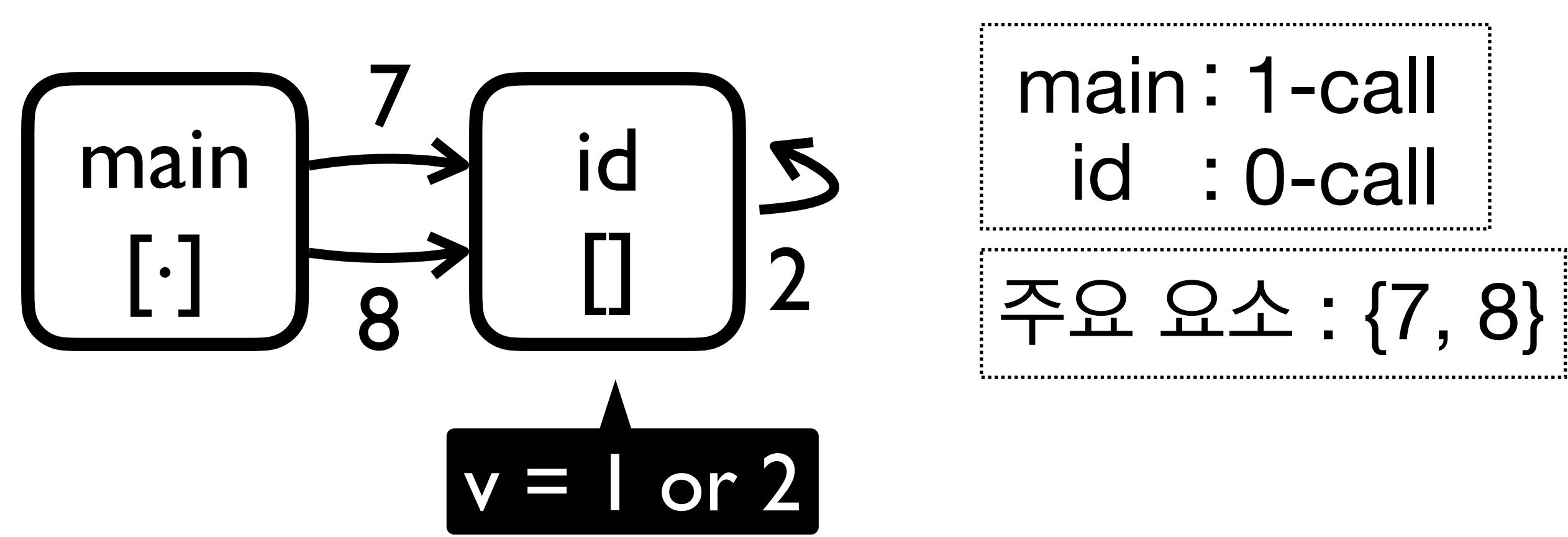
주요 I-요소 기반 함수 호출 요약

```

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: }

```

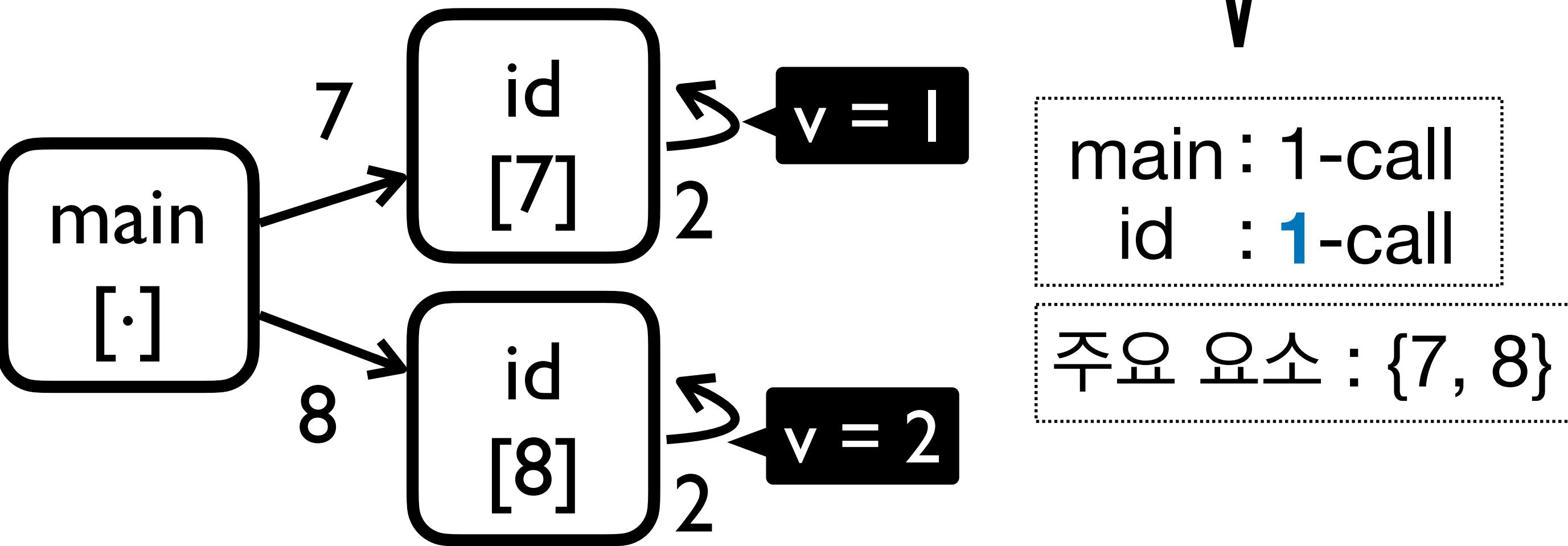
예제 프로그램



적당히 주요 I-요소 기반 함수 호출 요약

같이 사용될 것을 고려해 분류해야 함

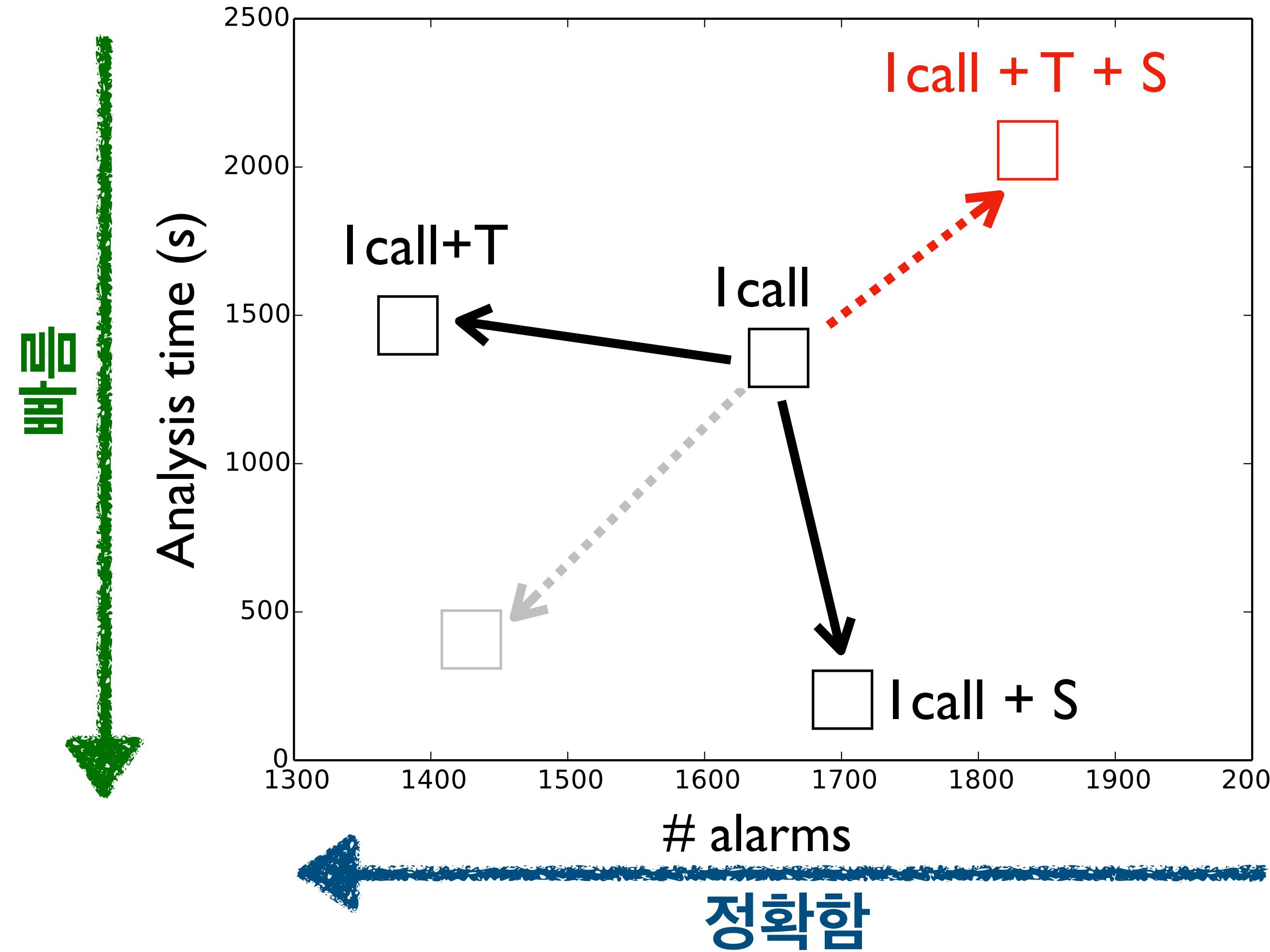
```
0: id(v, i){  
1:   if (i > 0){  
2:     return id(v, i-1);}  
3:   return v;}  
4:  
5: main(){  
6:   i = input();  
7:   v1 = id(1, i); //A  
8:   v2 = id(2, i); //B  
9:   assert (v1 != v2); //query  
10: }
```



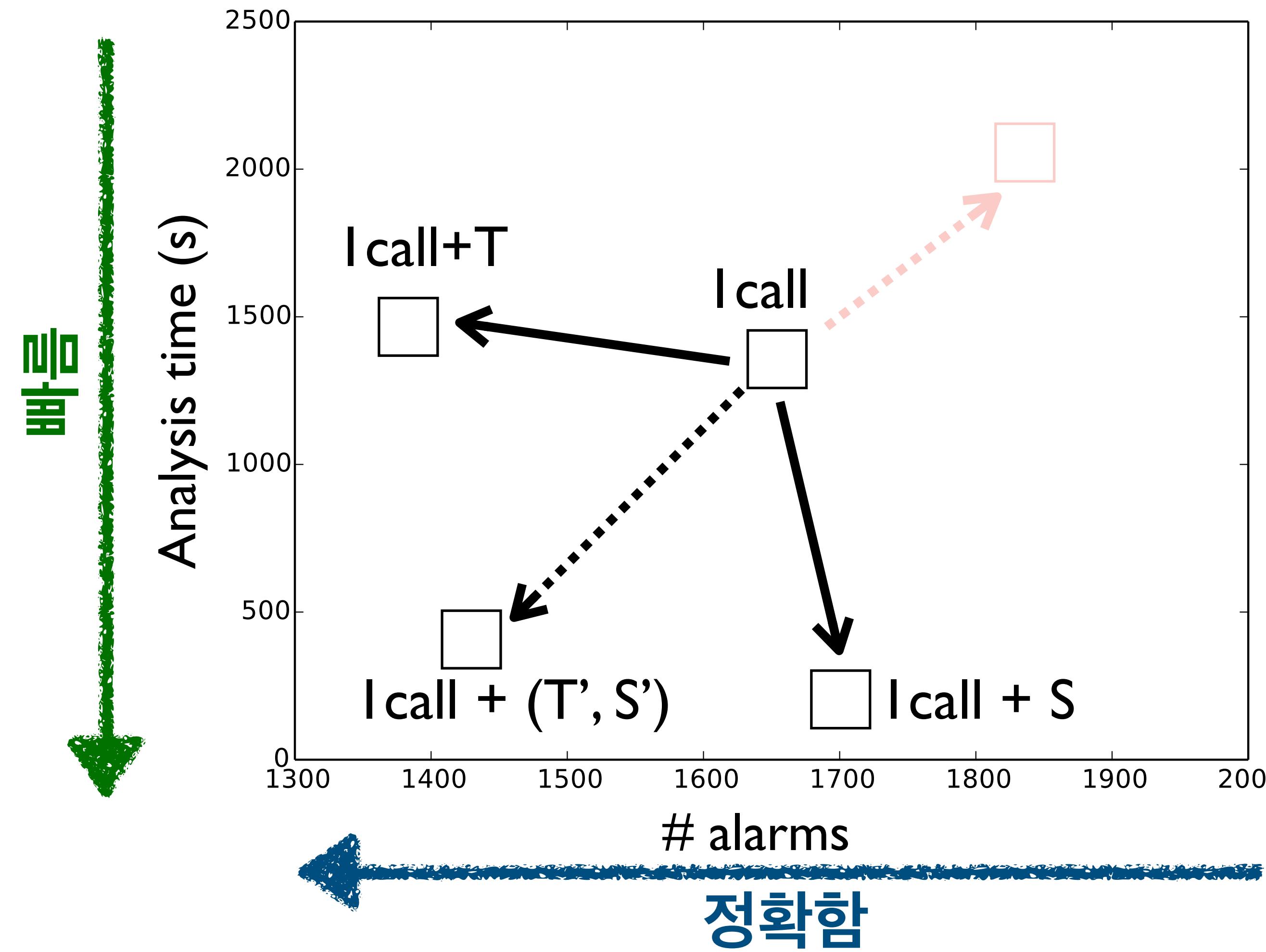
적당히 주요 I-요소 기반 함수 호출 요약

예제 프로그램

- 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

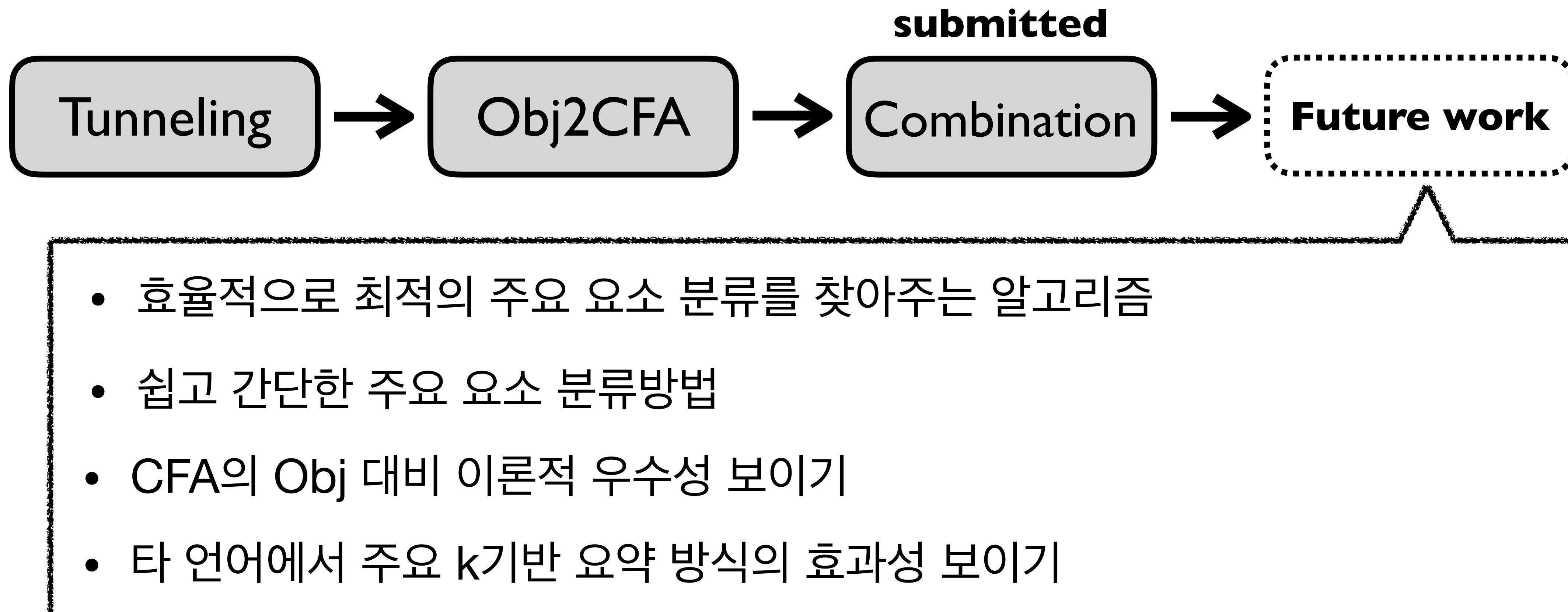
- We identify a fundamental issue in the current trend of developing context sensitivity techniques in pointer analysis and present a way to efficiently address it.

분석 기술들을 동시에 개발하는 법

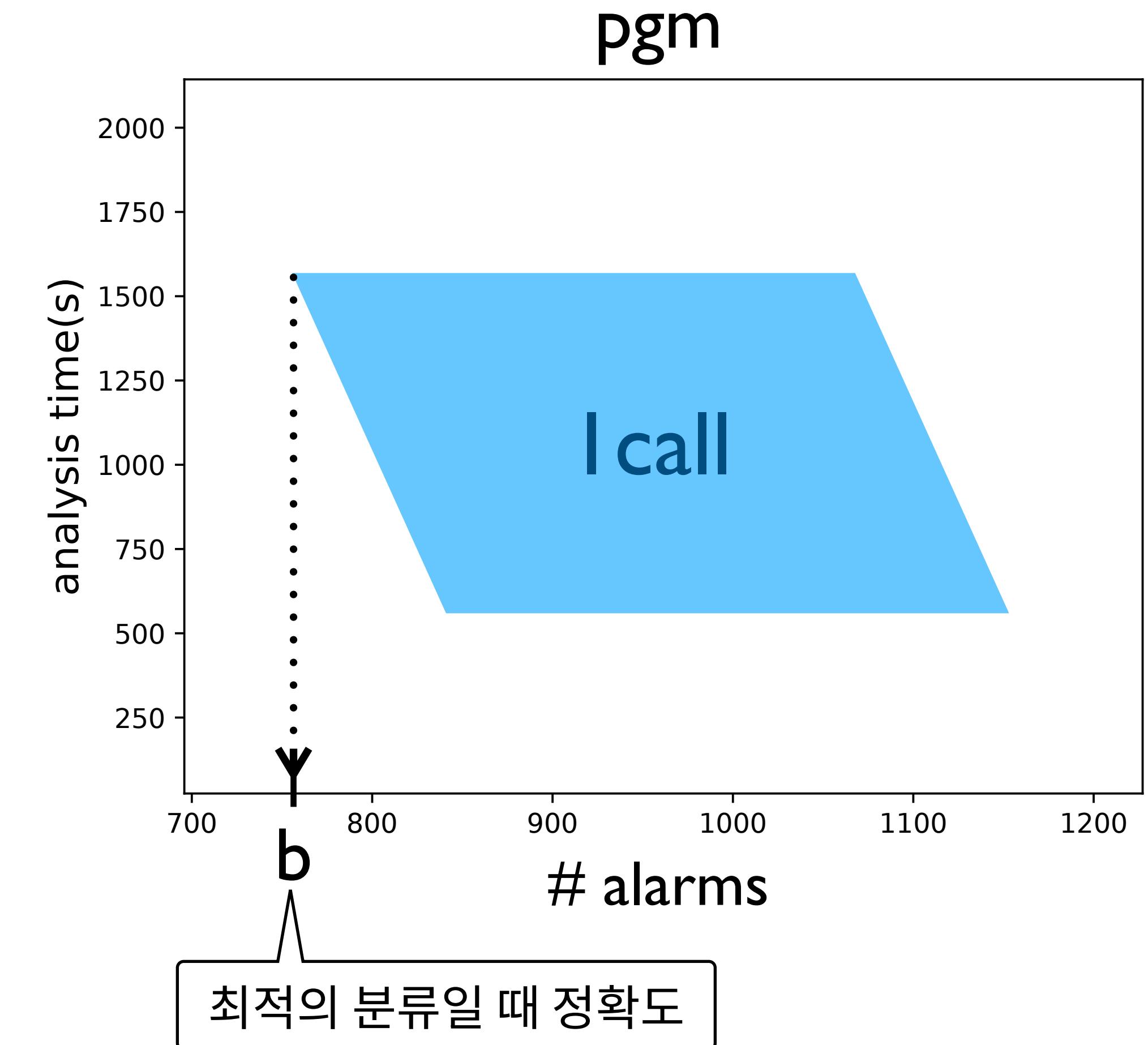
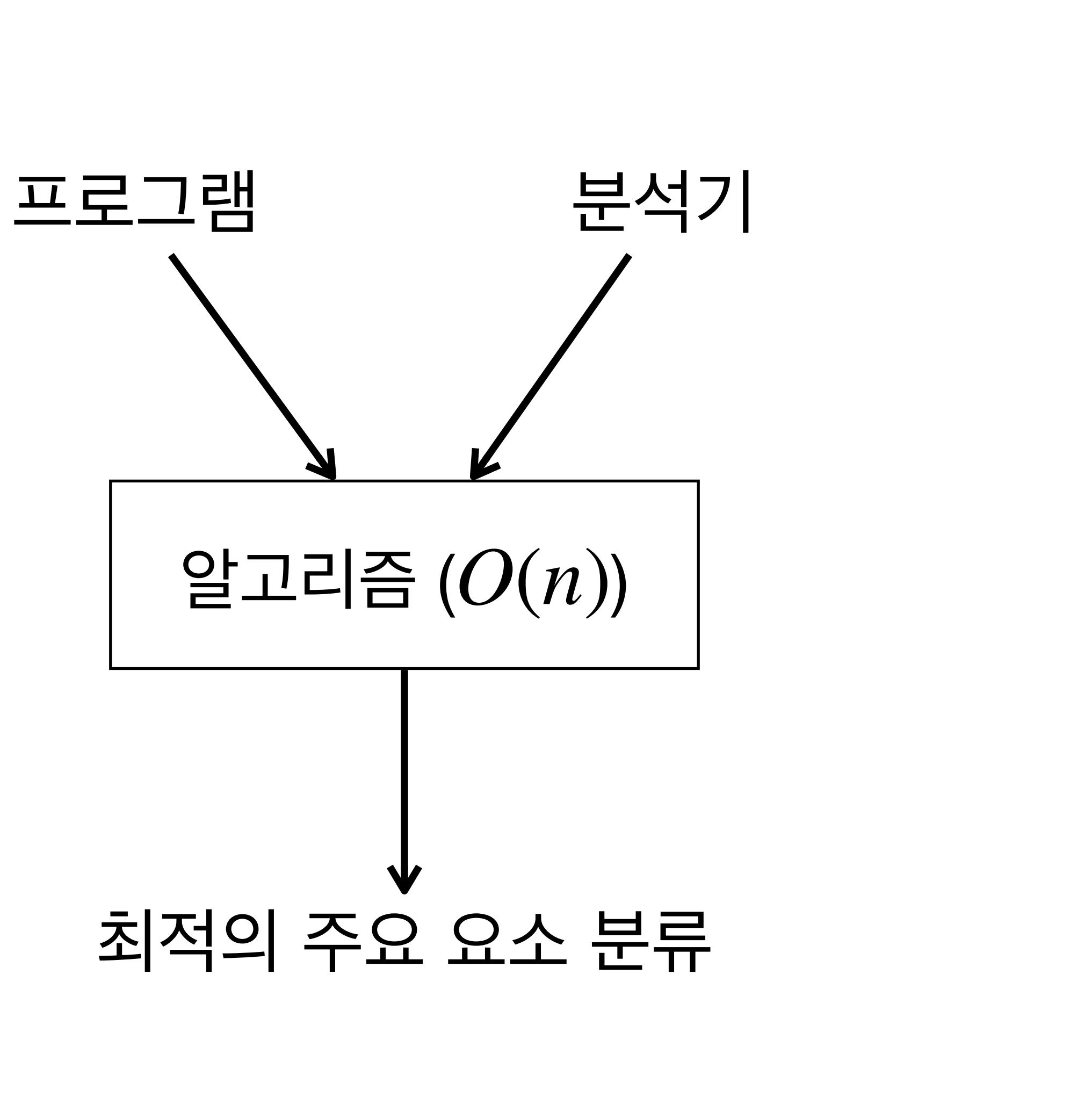
- We present a framework that significantly reduces the complexity of developing an effective combination of the two techniques

컨텍스트 터널링: 고정관념에 도전하기

- 목표: 주요 k기반 요약 방식을 표준으로 만들기

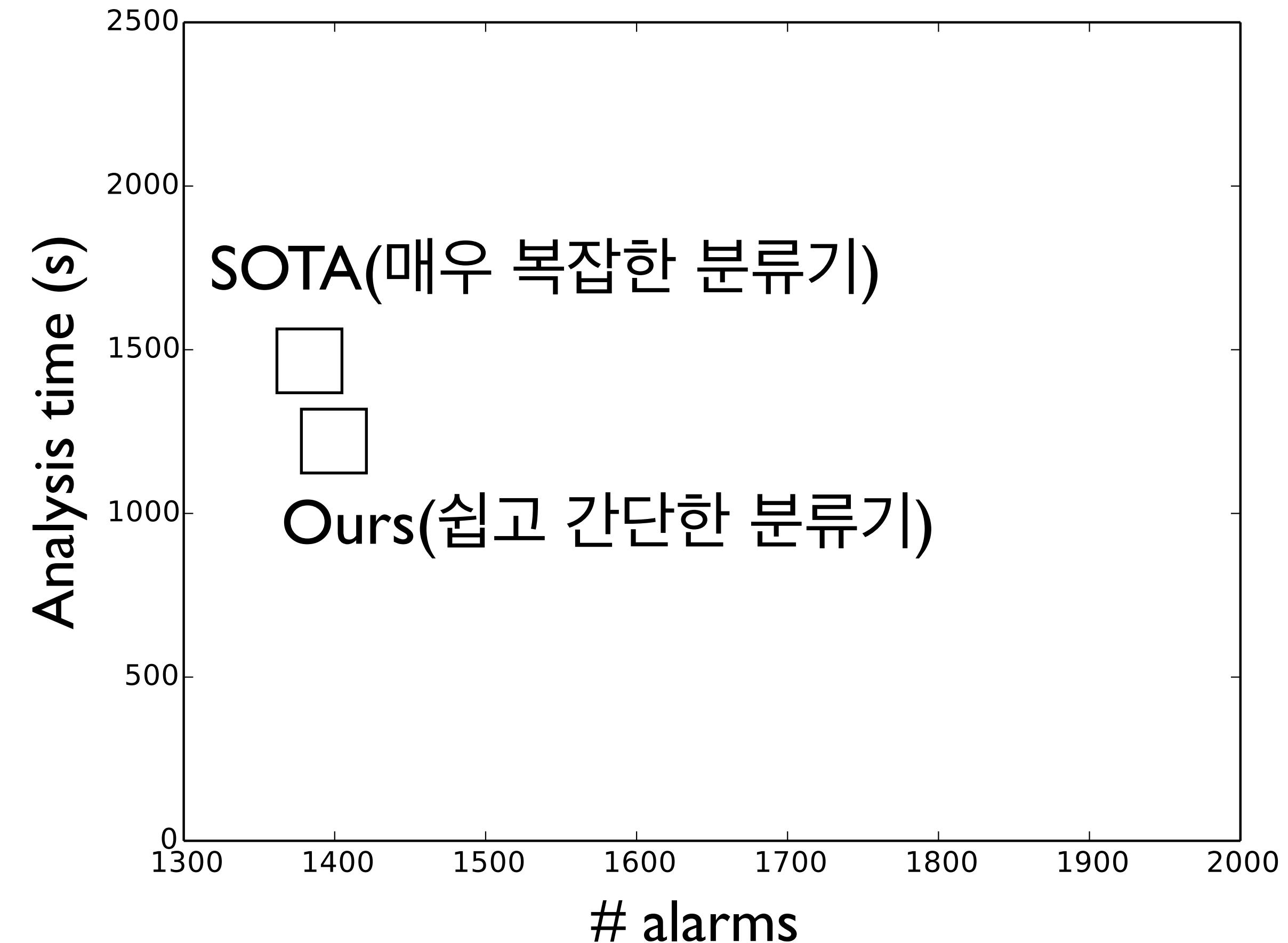


후속 연구 1: 최적의 주요 요소 분류 찾기 알고리즘



후속 연구 2: 쉽고 간단한 주요 요소 분류 방법

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



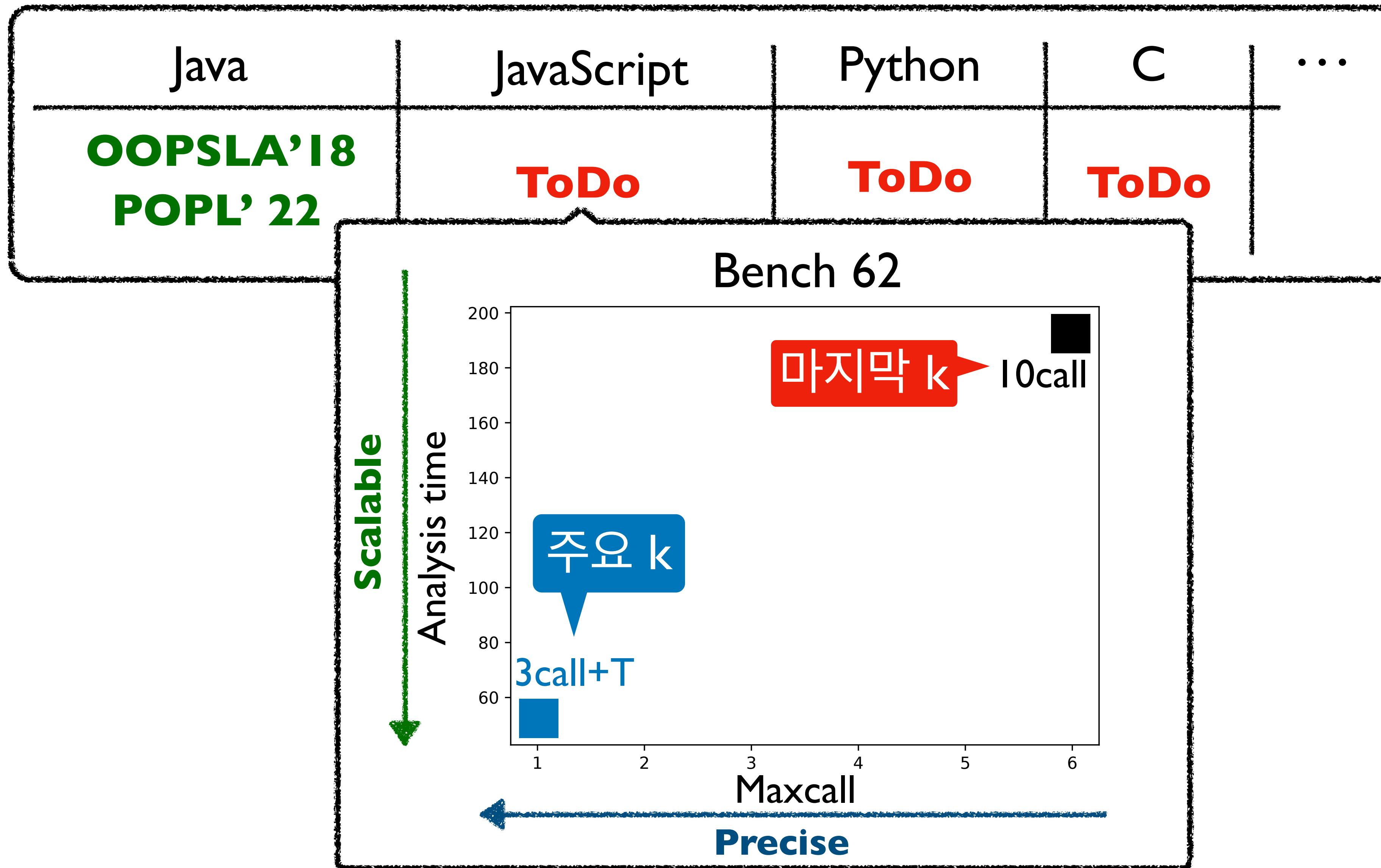
후속 연구 3: CFA의 Obj 대비 우수성을 이론적으로 보이기

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



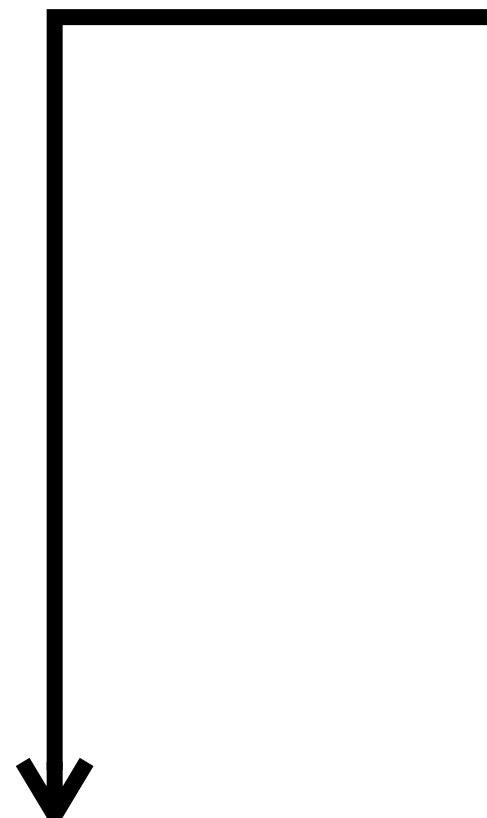
마무리: 고정관념에 도전하기

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

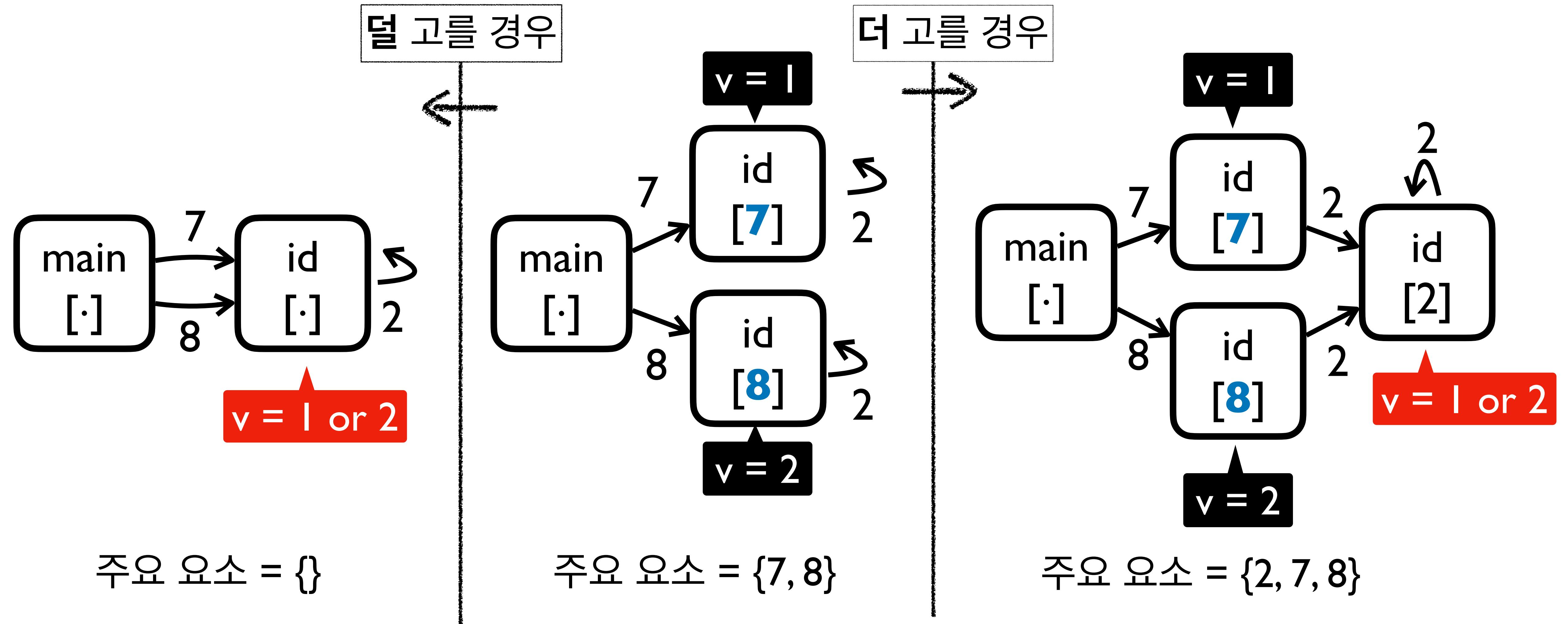
마지막 k → 주요 k

Obj → CFA

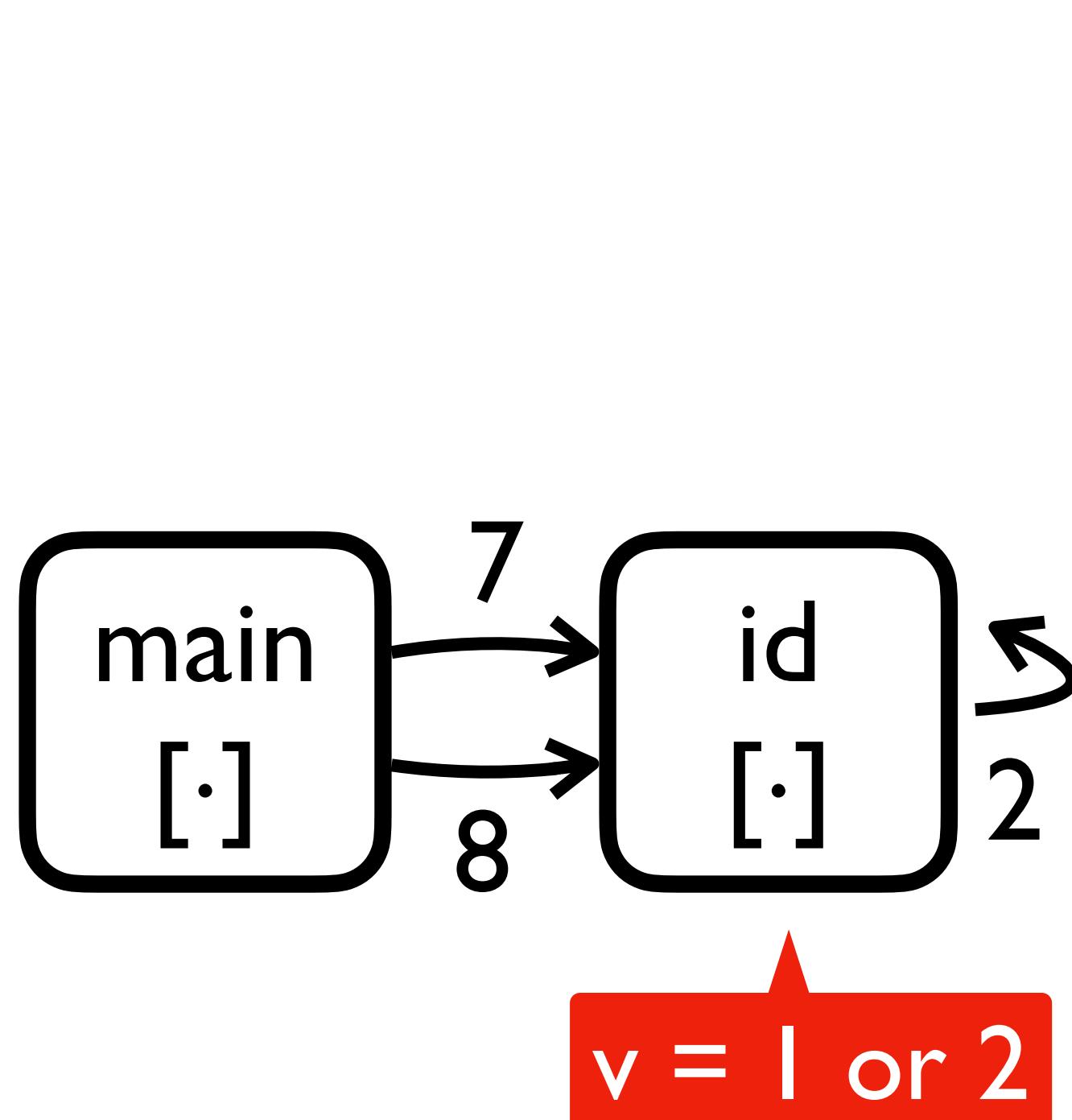
따로 개발 → 동시에 개발

Back up

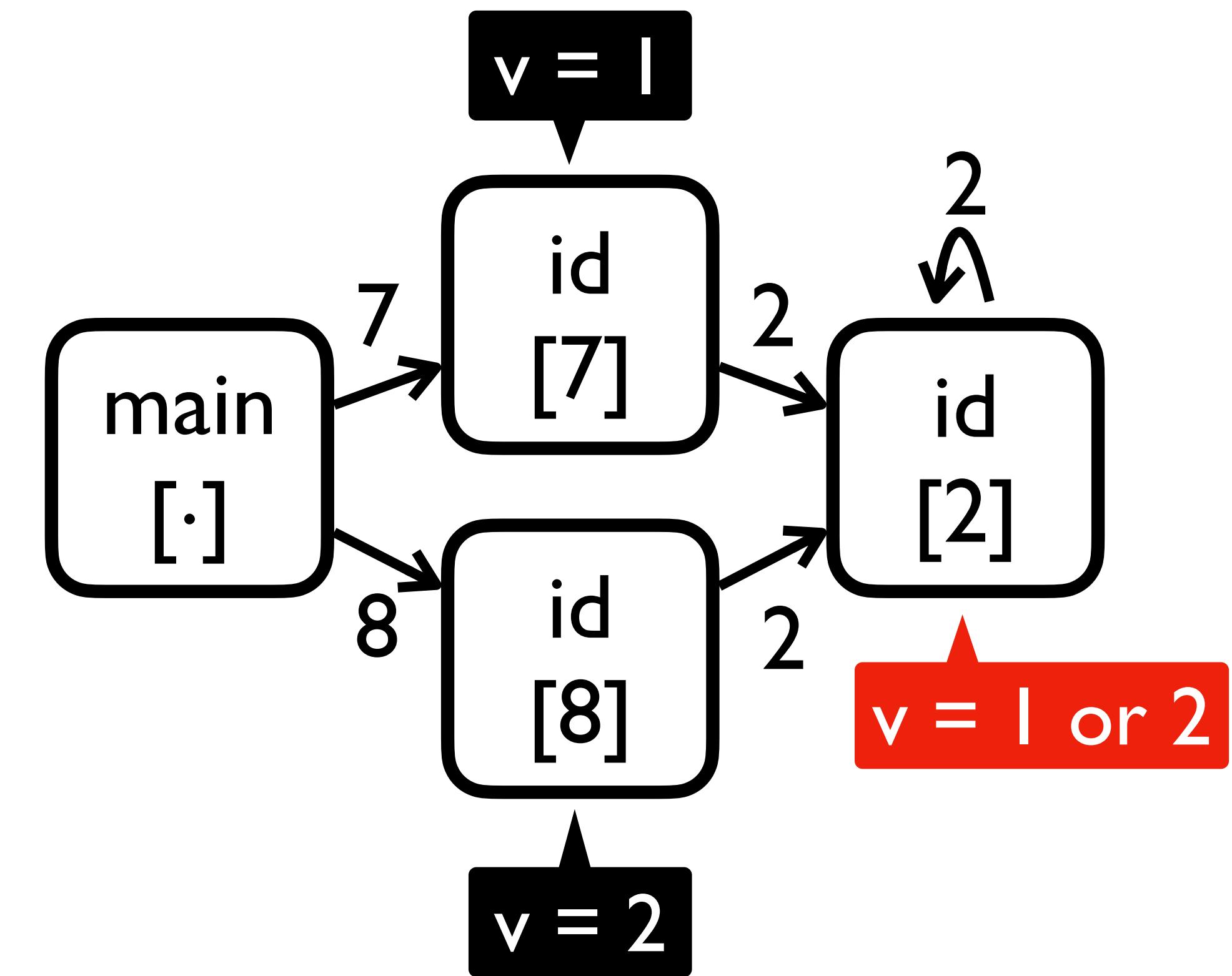
- 주요 |-요소 기반 함수 호출 요약



- 선택적 1-요소 기반 함수 호출 요약



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

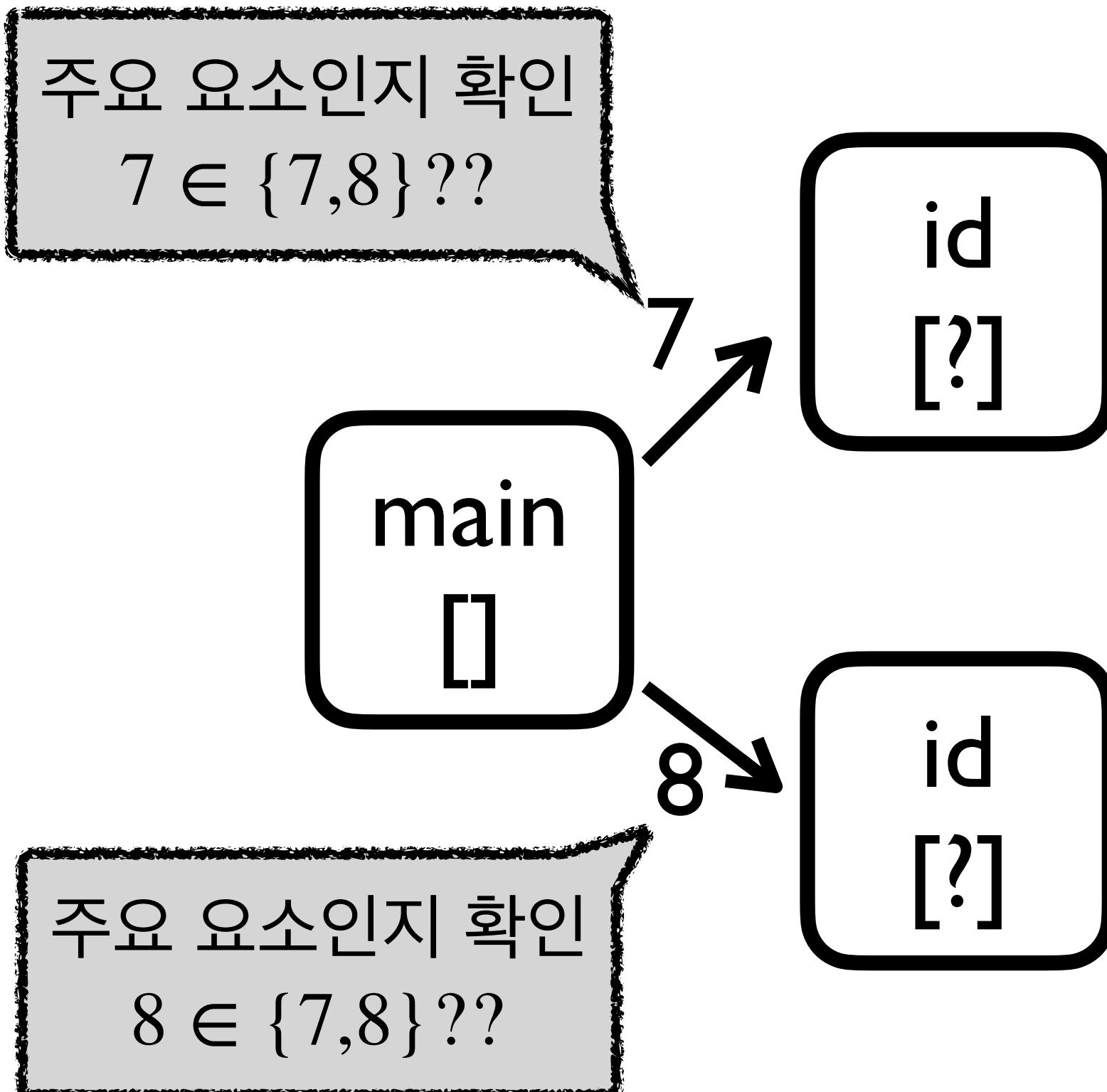


1-call : {id}
0-call : {main}

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}

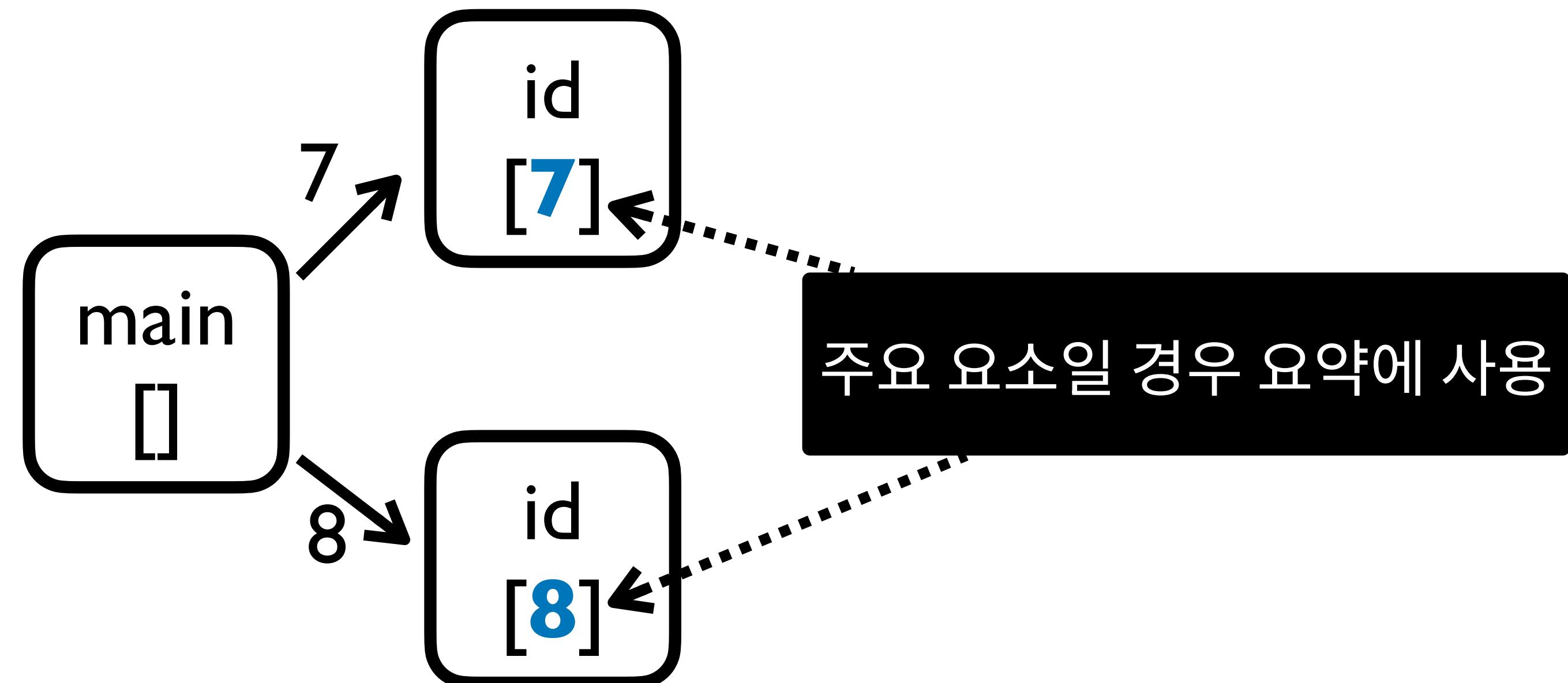


예제 프로그램

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: }
```

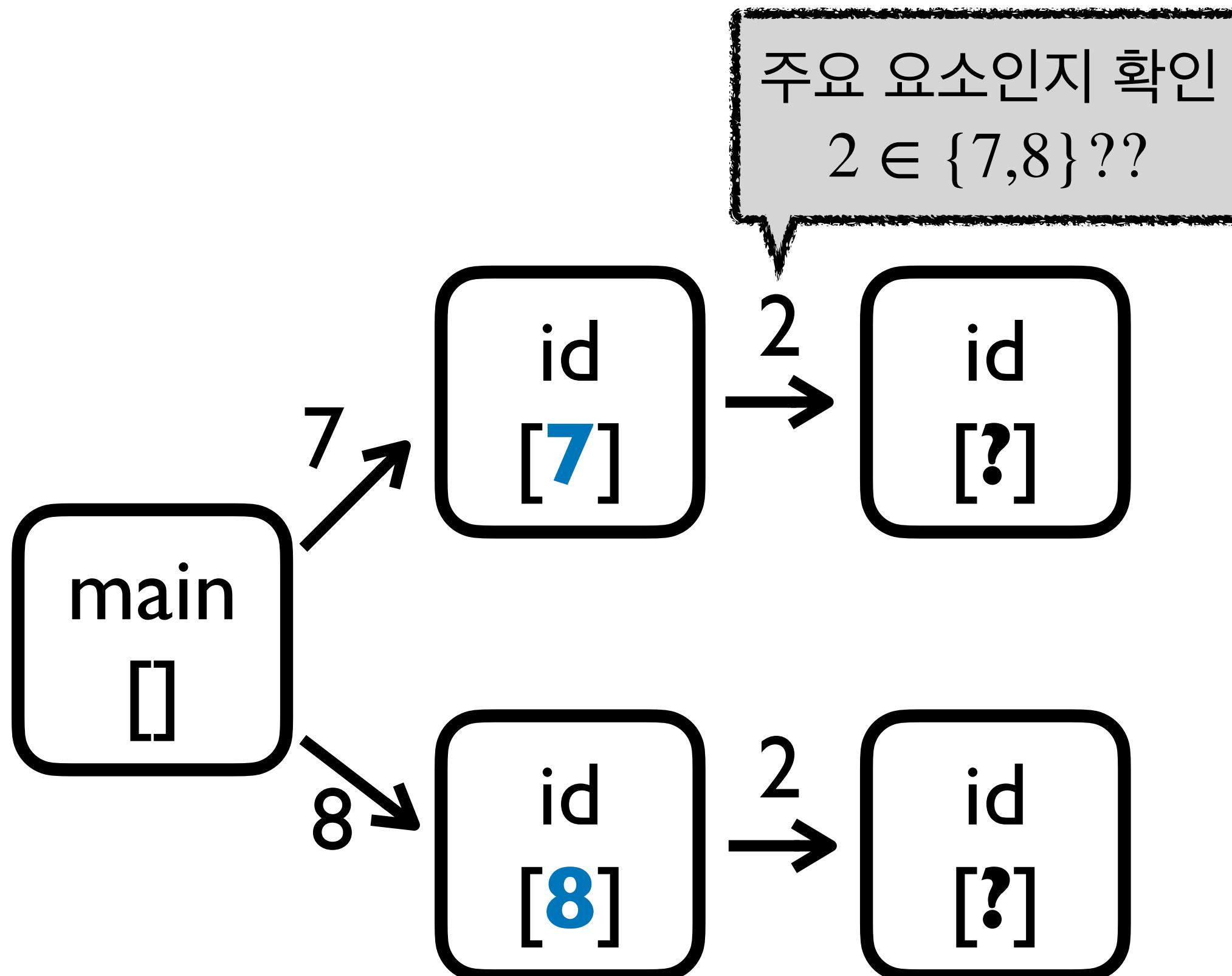


예제 프로그램

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: }
```

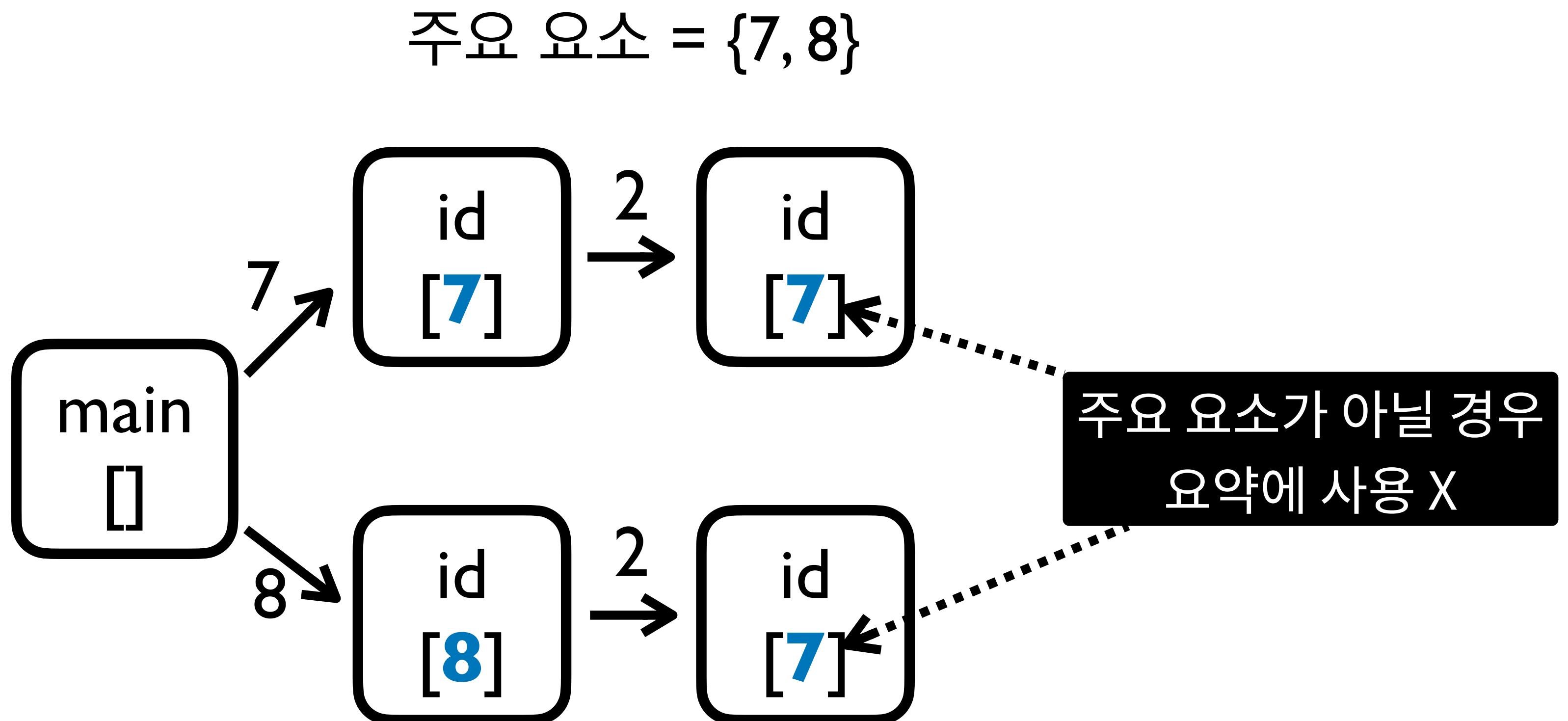
주요 요소 = {7, 8}



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: }
```

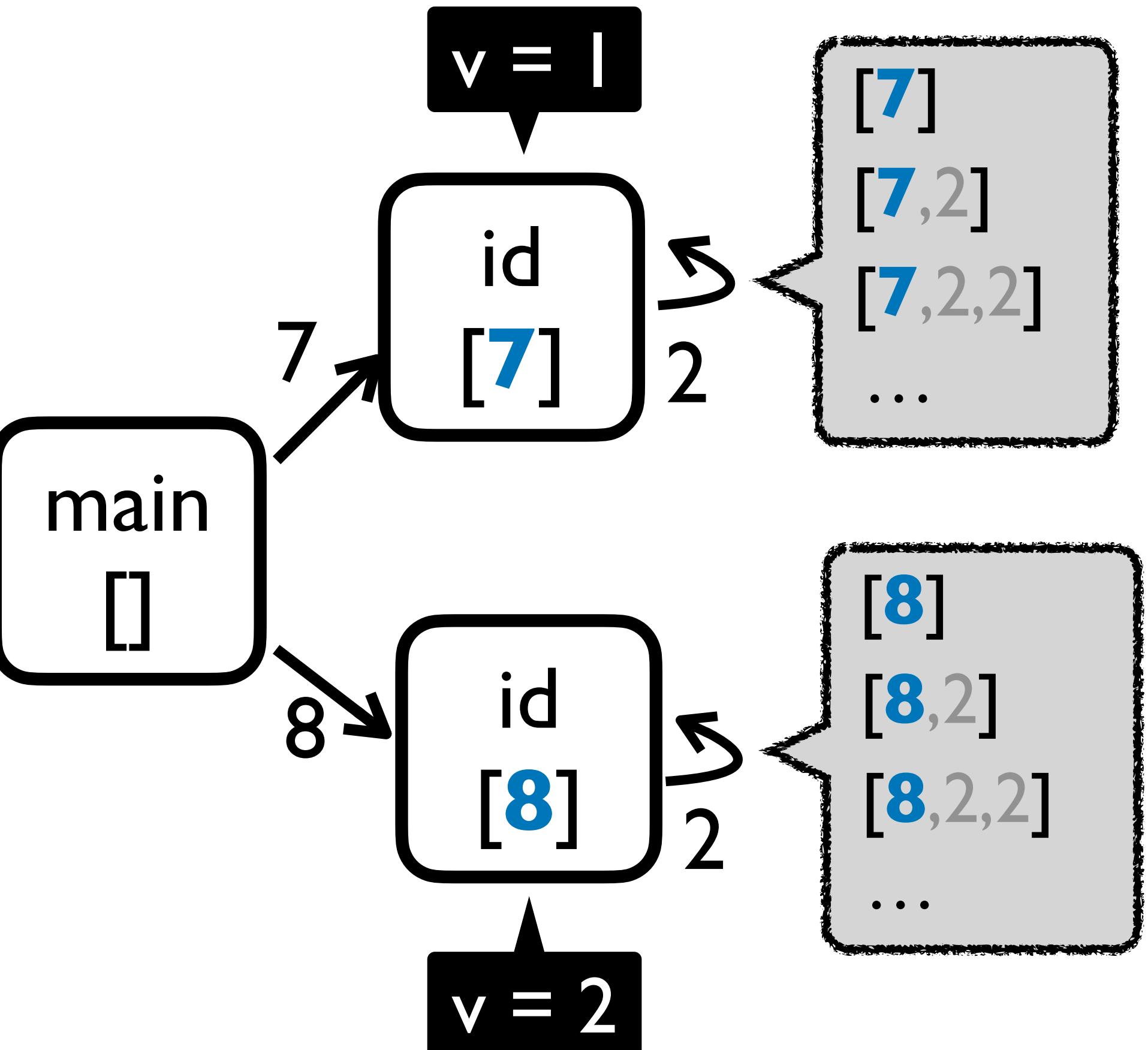


예제 프로그램

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})