컨텍스트 터널링

구분할 호출환경 배달하기: 고정관념에 도전하기

전 민 석

DGIST

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?

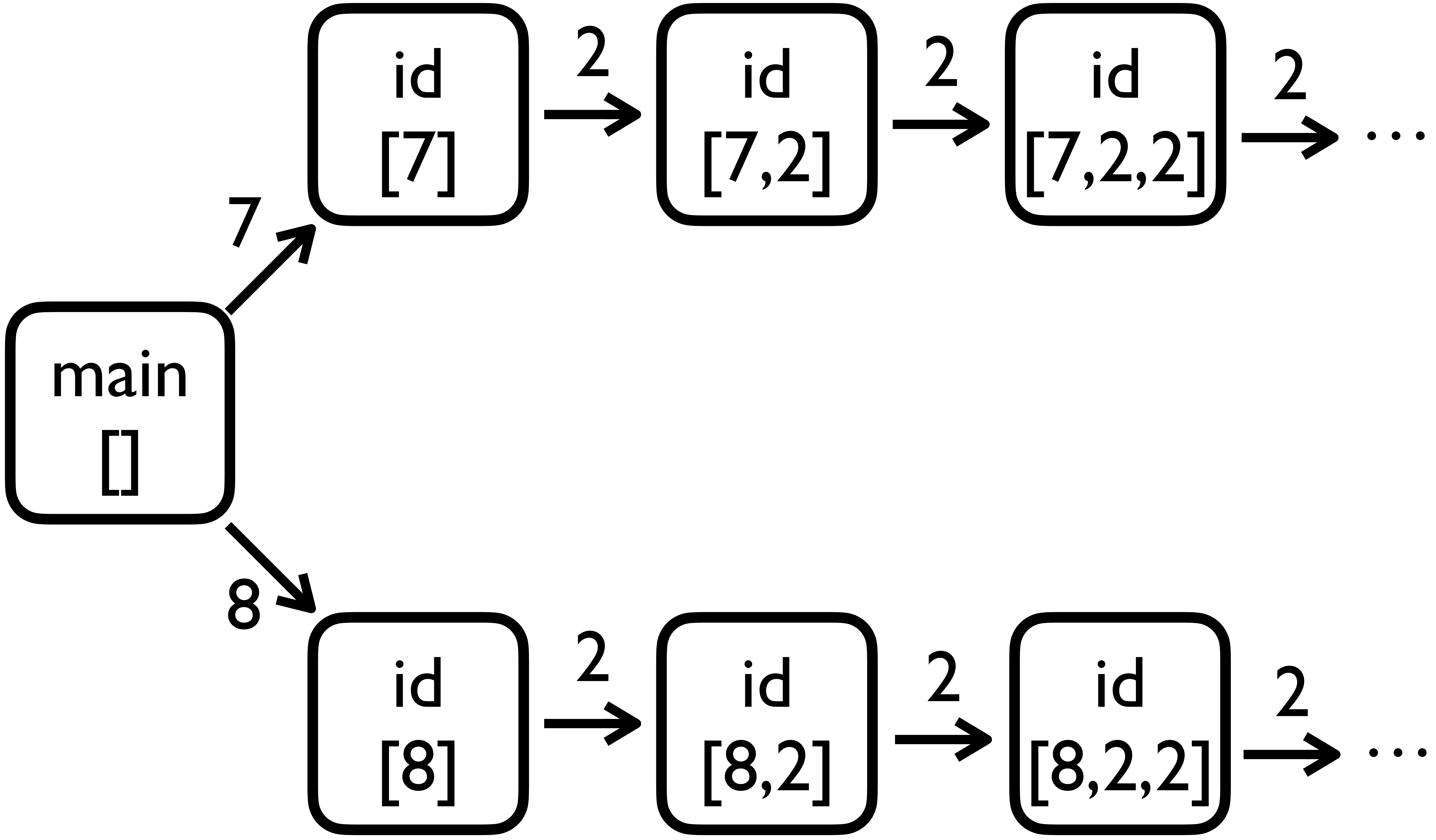
"The End"

0

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

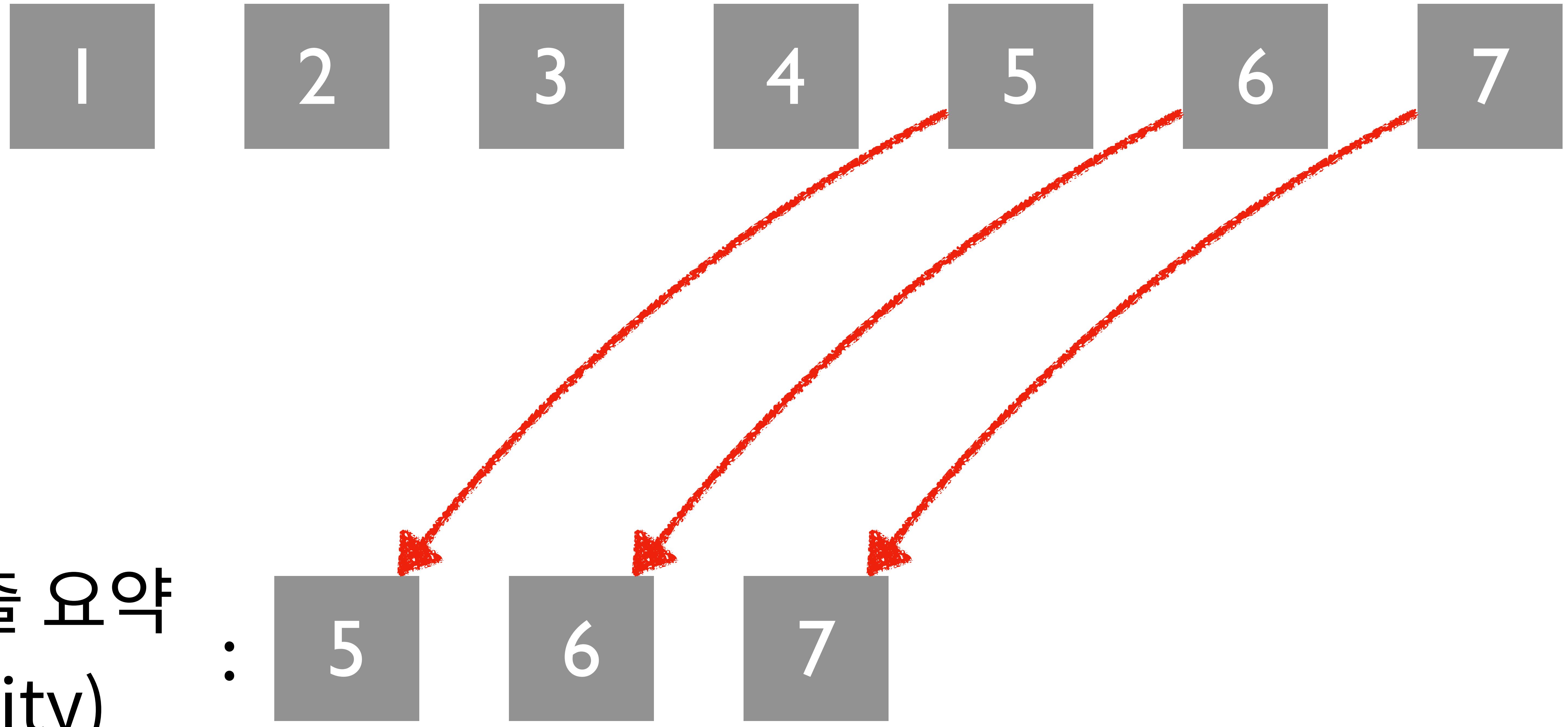


함수 호출 그래프

K개 요소 기반 함수 호출 요약

실제 함수 호출 맥락:

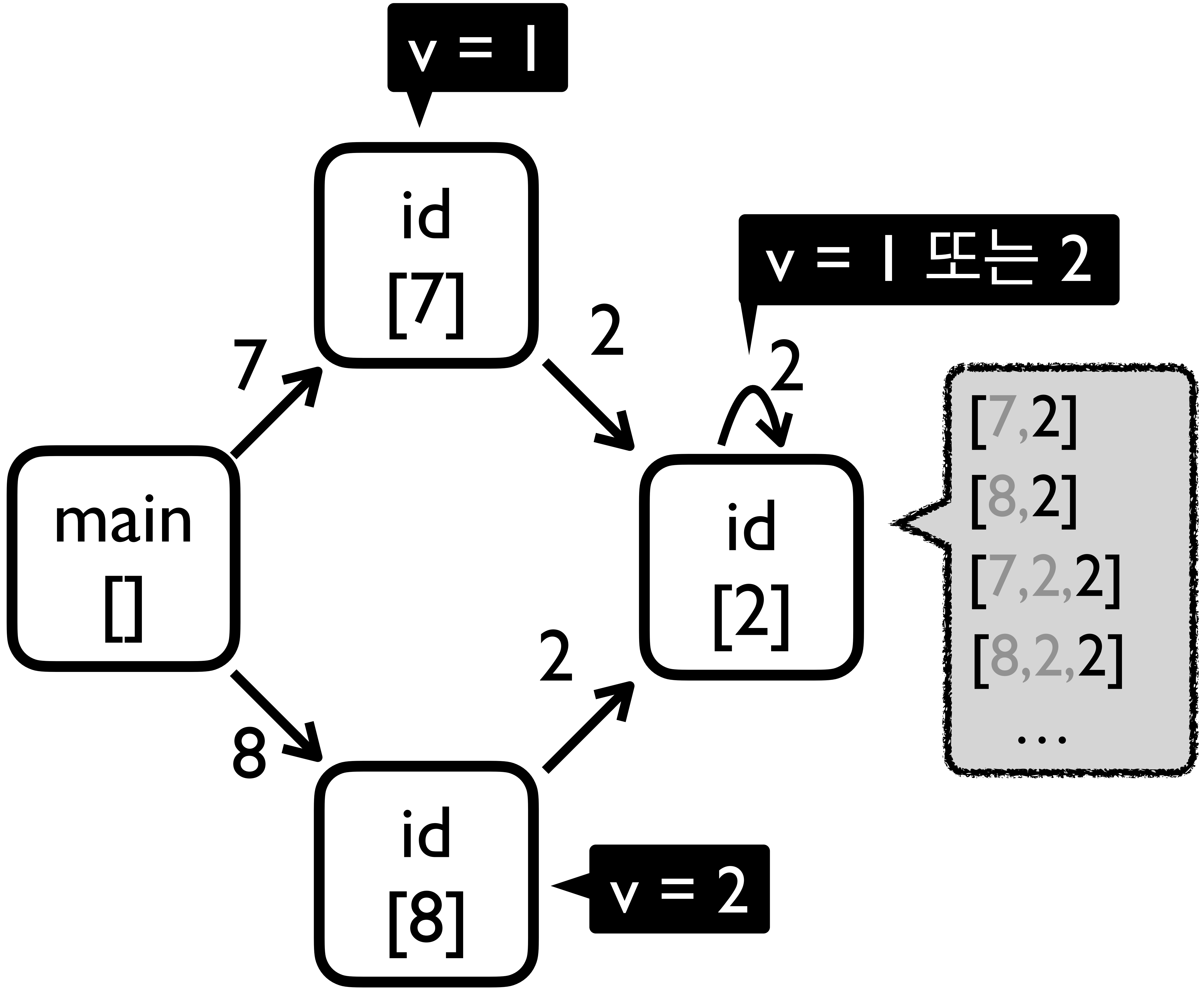
3개 요소 기반 함수 호출 요약
(3-context sensitivity)
:



K개 요소 기반 함수 호출 요약

```
id(v, i){
  if (i > 0){
    return id(v, i-1);}
  return v;}

main(){
  i = input();
  v1 = id(1, i);//A
  v2 = id(2, i);//B
  assert (v1 != v2);//query
}
```

예제 프로그램



1개 요소 기반 함수 호출 요약

K개 요소 기반 함수 호출 요약

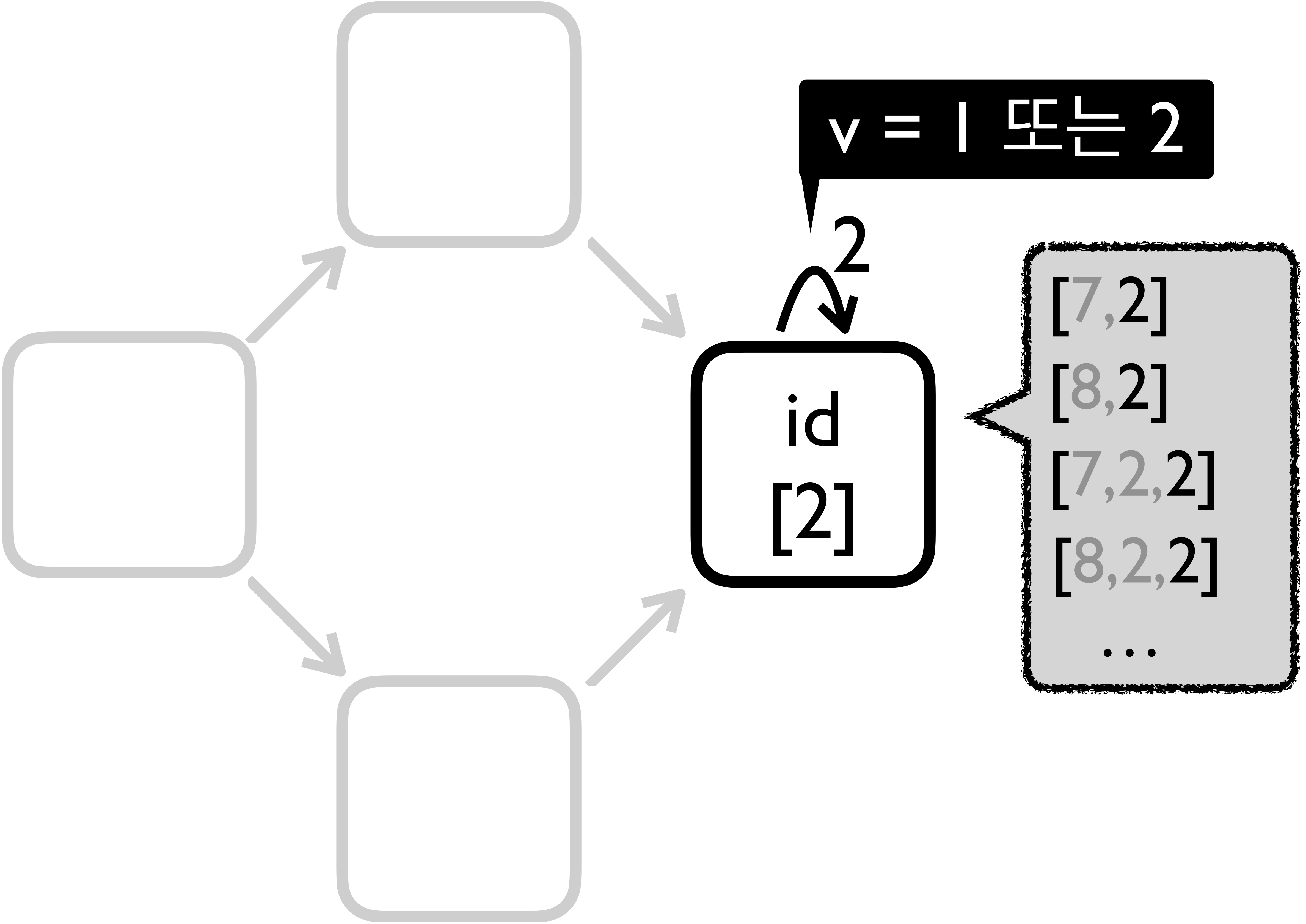
```
id(v, i){
  if (i > 0){
    return id(v, i-1);}
  return v;}

main(){
  i = input();
  v1 = id(1, i);
  v2 = id(2, i);
  assert (v1 != v2);//query
}
```

v = 1 또는 2

1 또는 2

1 또는 2

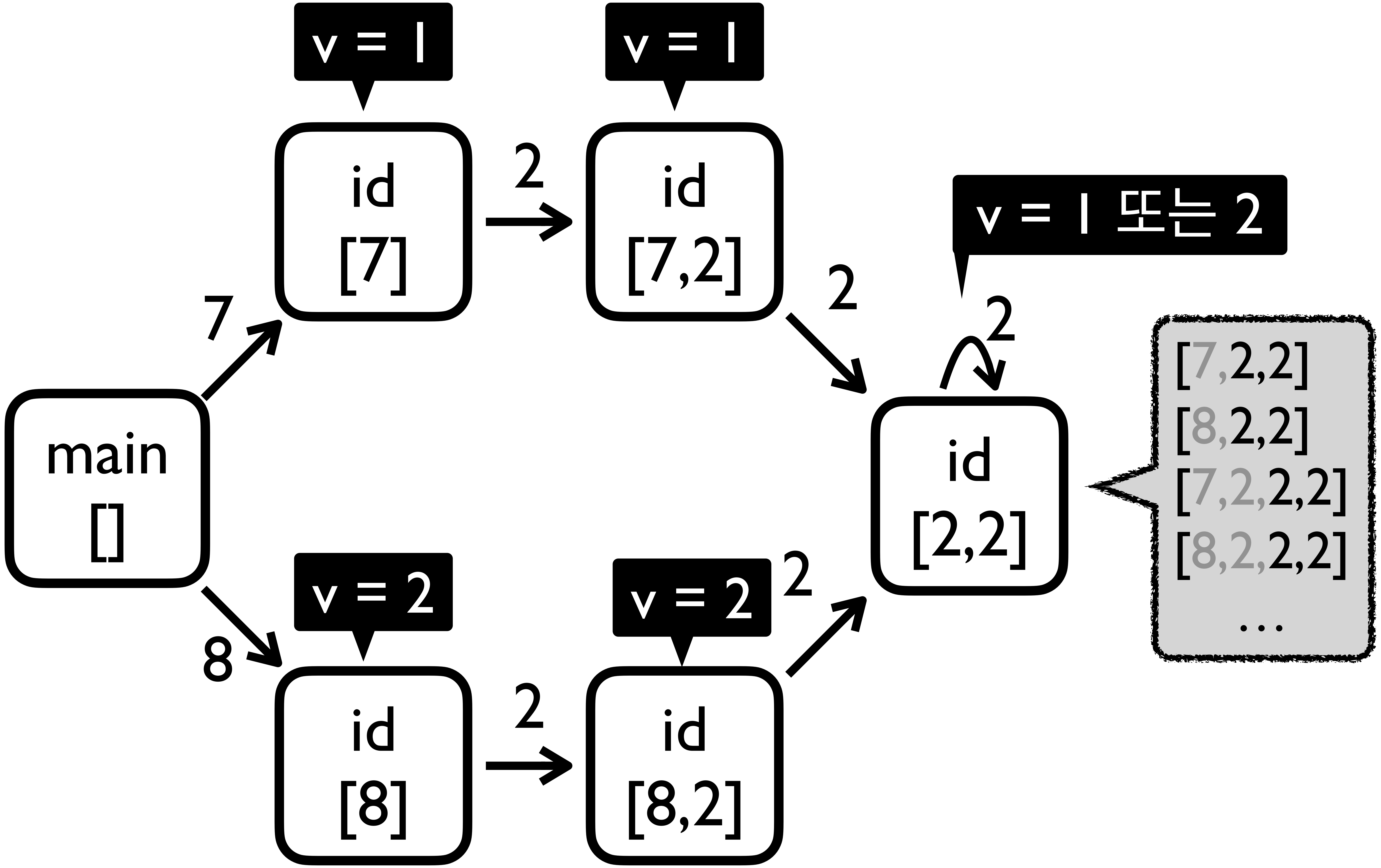
예제 프로그램



K개 요소 기반 함수 호출 요약

```
id(v, i){
  if (i > 0){
    return id(v, i-1);}
  return v;}

main(){
  i = input();
  v1 = id(1, i);//A
  v2 = id(2, i);//B
  assert (v1 != v2);//query
}
```

예제 프로그램

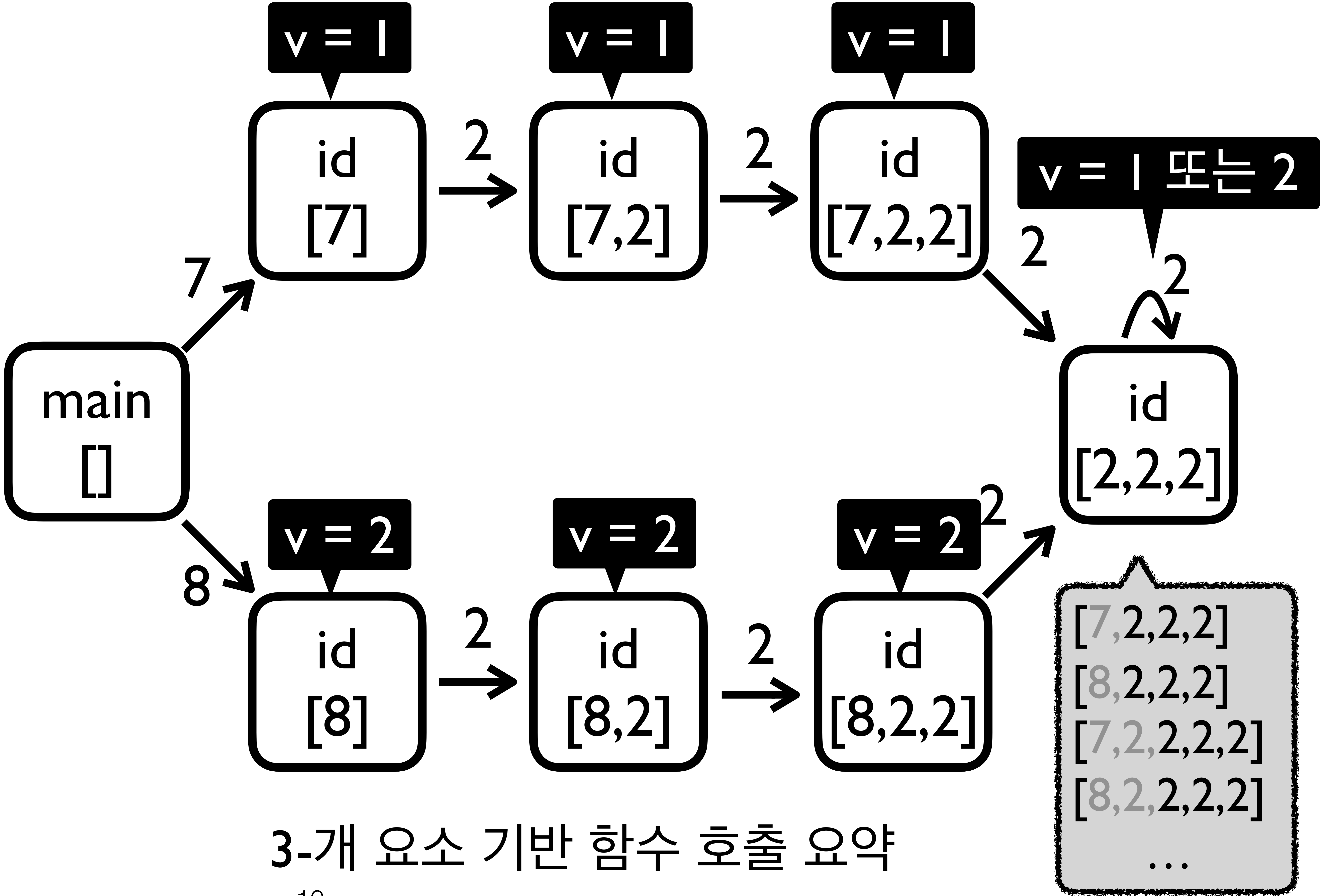


2-개 요소 기반 함수 호출 요약

K개 요소 기반 함수 호출 요약

```
id(v, i){
  if (i > 0){
    return id(v, i-1);}
  return v;}

main(){
  i = input();
  v1 = id(1, i);//A
  v2 = id(2, i);//B
  assert (v1 != v2);//query
}
```

예제 프로그램



3-개 요소 기반 함수 호출 요약

고정 관념: 마지막 k개 기반 요약

- 마지막 k개 기반 요약 방식이 널리 강의 되는 중

Call-Site Sensitivity

KOREA UNIVERSITY

- 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 = ∞: keep all contexts, i.e., fully context-sensitive

KAIST

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

(검색 키워드 : Context Sensitivity Static Analysis Lecture)

고정 관념: 마지막 k개 기반 요약

- 리뷰 코멘트

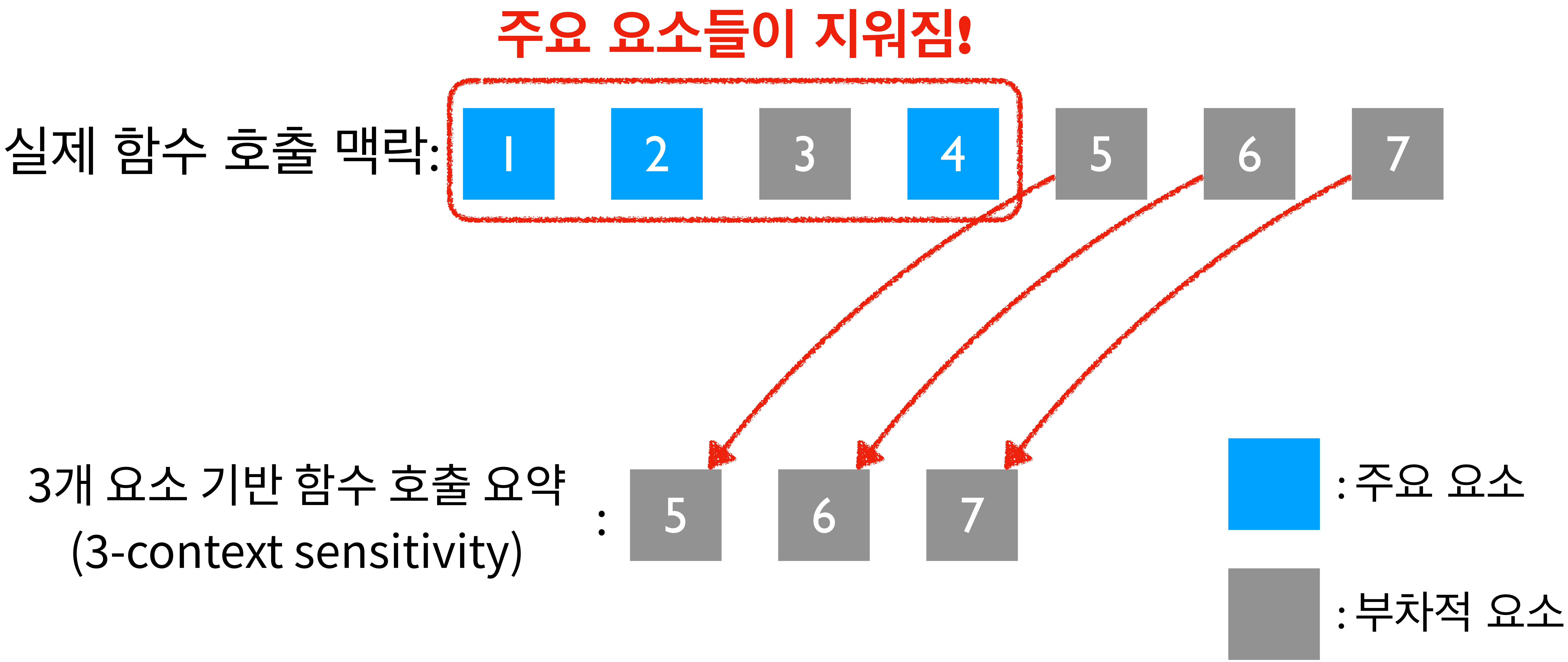
"A key part of the appeal of last k-based context abstraction is its simplicity and universal applicability."

- A reviewer [expert]

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

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

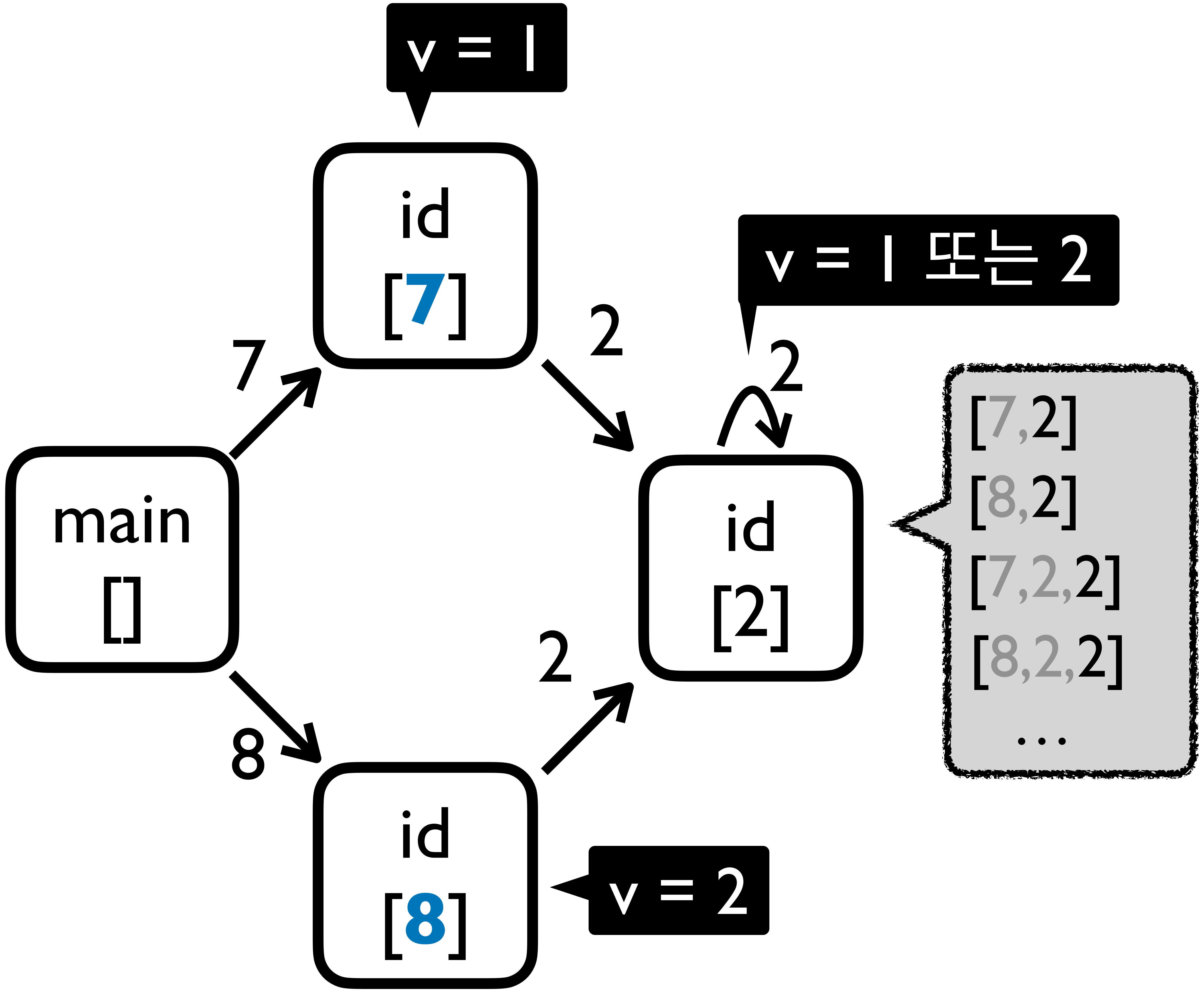
마지막 K개 기반 요약의 문제점



마지막 K개 기반 요약의 문제점

```
id(v, i){
  if (i > 0){
    return id(v, i-1);}
  return v;}

main(){
  i = input();
  v1 = id(1, i);
  v2 = id(2, i);
  assert (v1 != v2);//query
}
```

예제 프로그램



1개 요소 기반 함수 호출 요약

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);
  }}
```

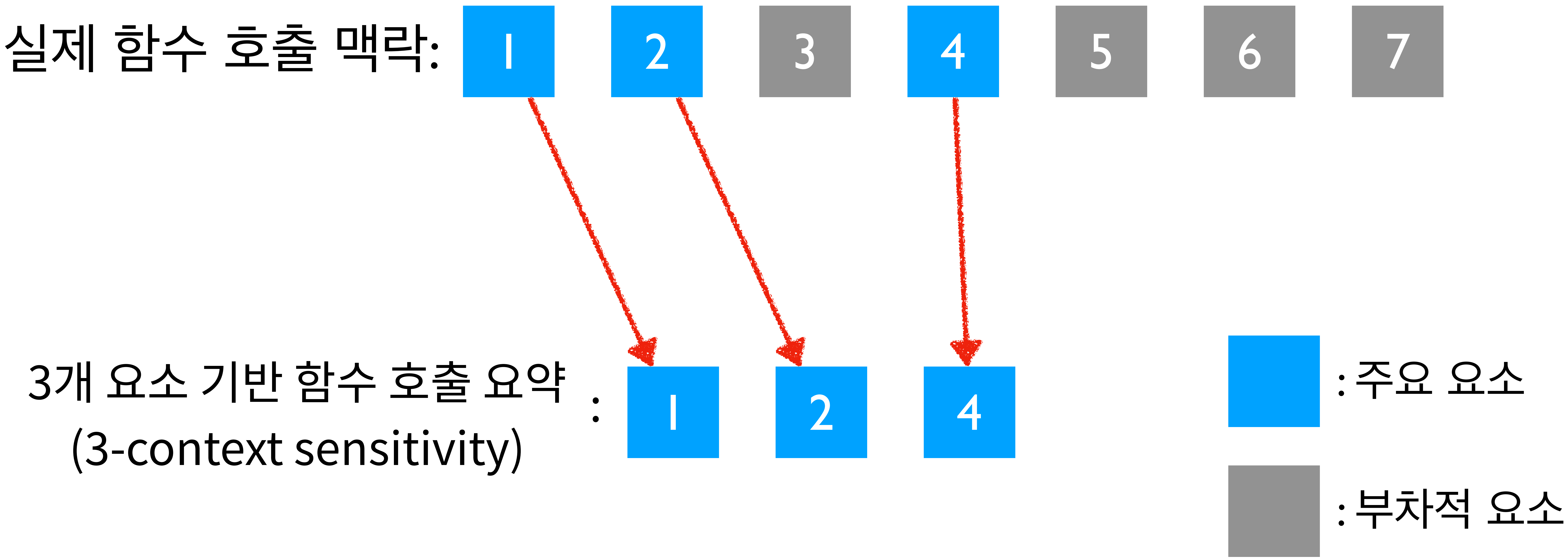
- What is the result of 1-call-site-sensitive analysis?

부정확함

질문:

1-call-site sensitivity로 정확하게 분석할 순 없나?

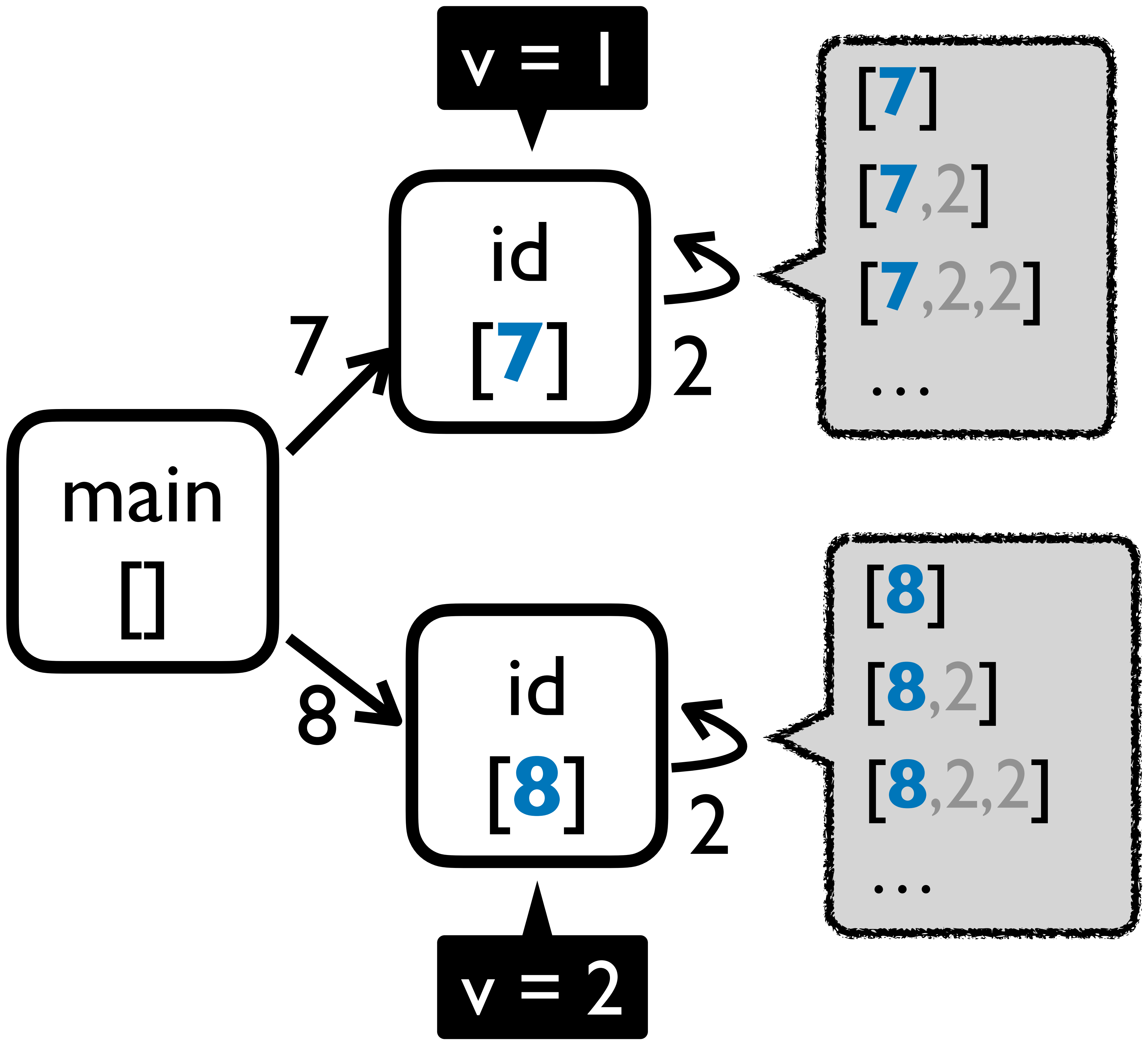
호출 환경 배달하기: 주요 K개 기반 요약



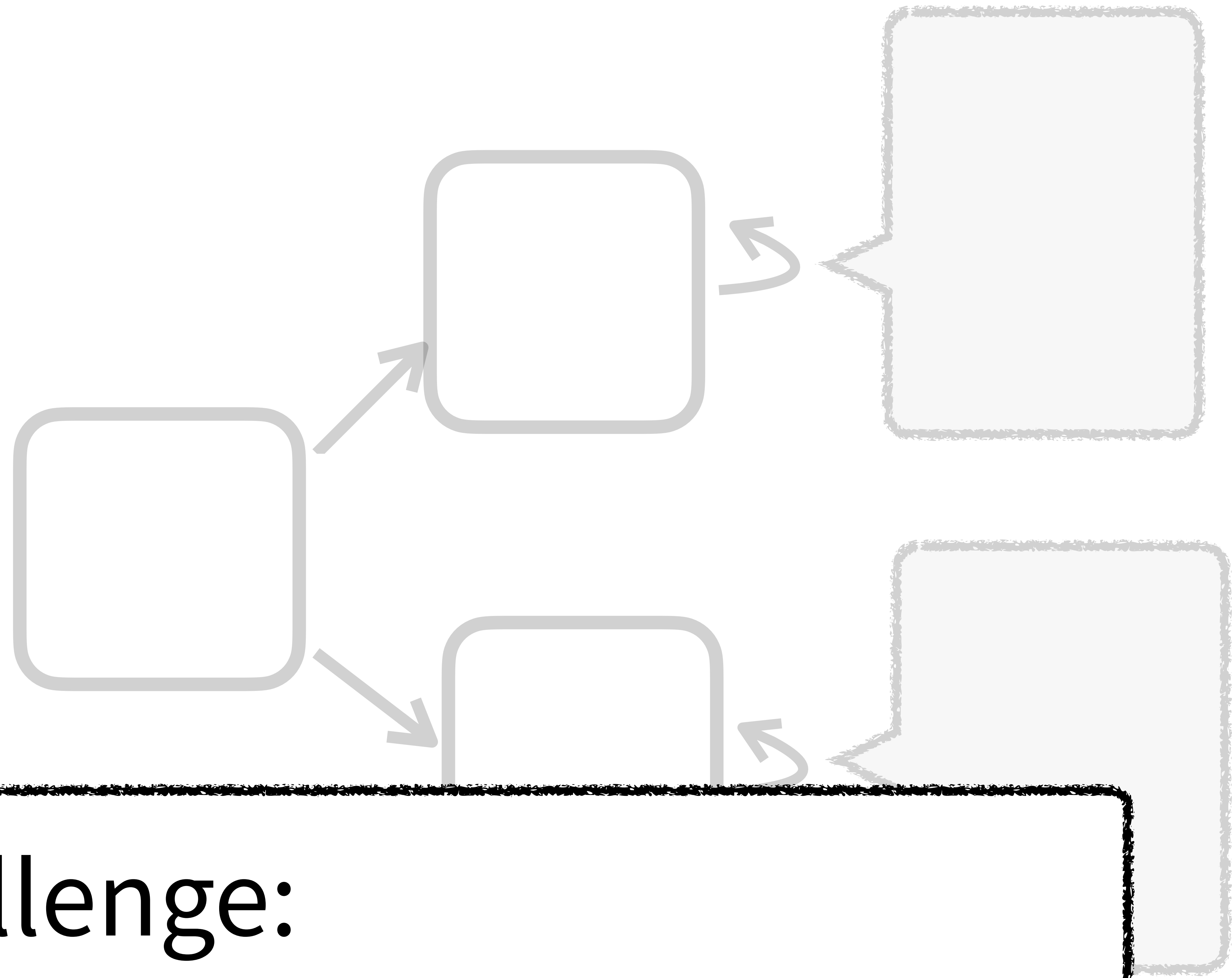
호출 환경 배달하기: 주요 K개 기반 요약

```
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 **정확**하게 알아내야 함

1개 주요 맥락 기반 함수 호출 요약

(주요 요소 = {7, 8})

- **정확하게** 분류해야만 분석의 정확도를 향상시킬 수 있음

주요 요소 = {}

주요 요소 = {7, 8}

주요 요소 = {2, 7, 8}

접근 방법: 데이터 기반 컨텍스트 터널링

학습 데이터
(프로그램 집합)

정적분석기

요소 특질
$(a_1, a_2, \ldots, a_n)$

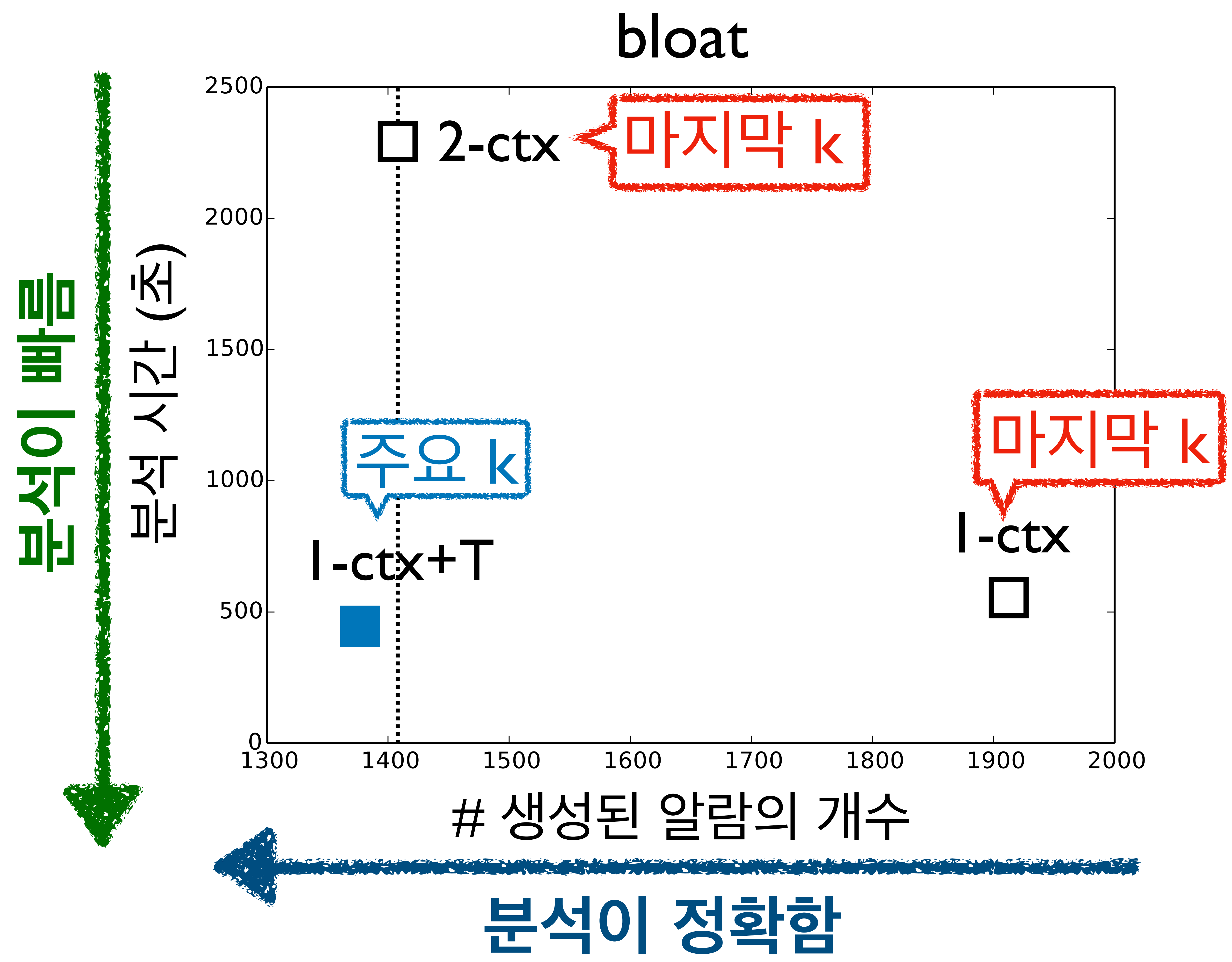
기계학습 알고리즘

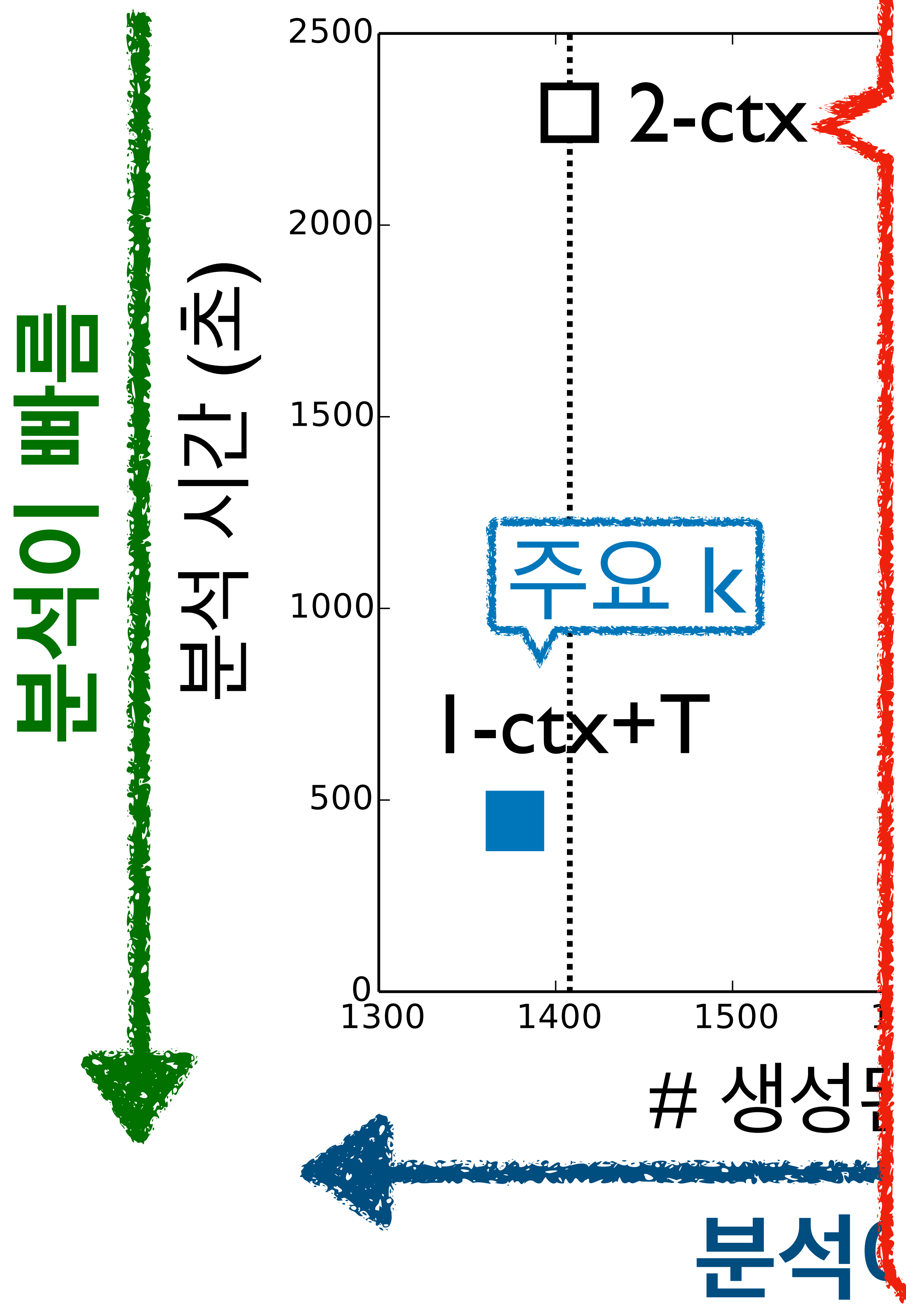
주요 요소 분류기

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

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

submitted

Tunneling → Obj2CFA → Combination → **Future work**

고정관념 허물기 I

마지막 **k** 기반 요약 ⟶ 주요 **k** 기반 요약

고정관념에 도전하기

- 목표: 주요 k기반 요약 방식을 표준으로 만들기

Tunneling → Obj2CFA → Combination (submitted) → **Future work**

고정관념에 허물기 2

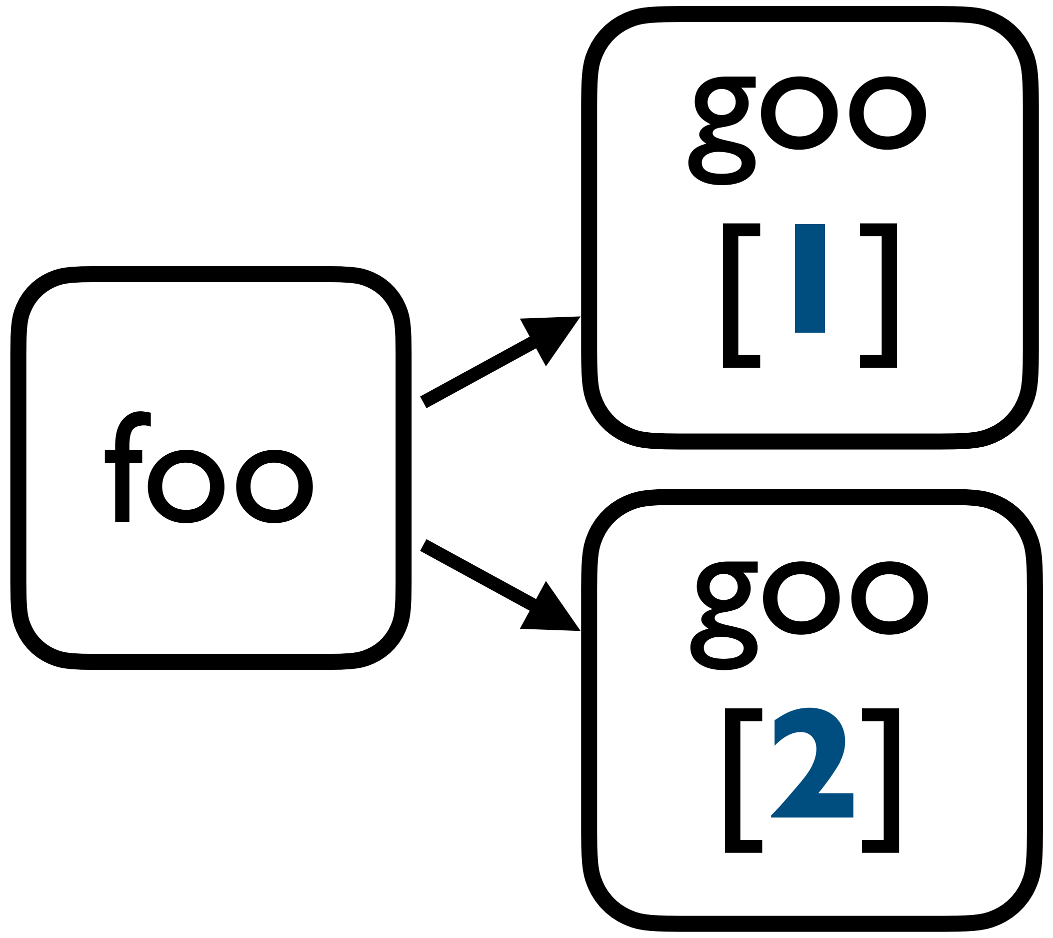
값 기반 요약 (Object Sensitivity) **vs** 위치 기반 요약 (Call Site Sensitivity)

위치 기반 요약 vs 값 기반 요약

위치 기반 요약 (1981)

- "**어디**"를 기반으로 요약

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



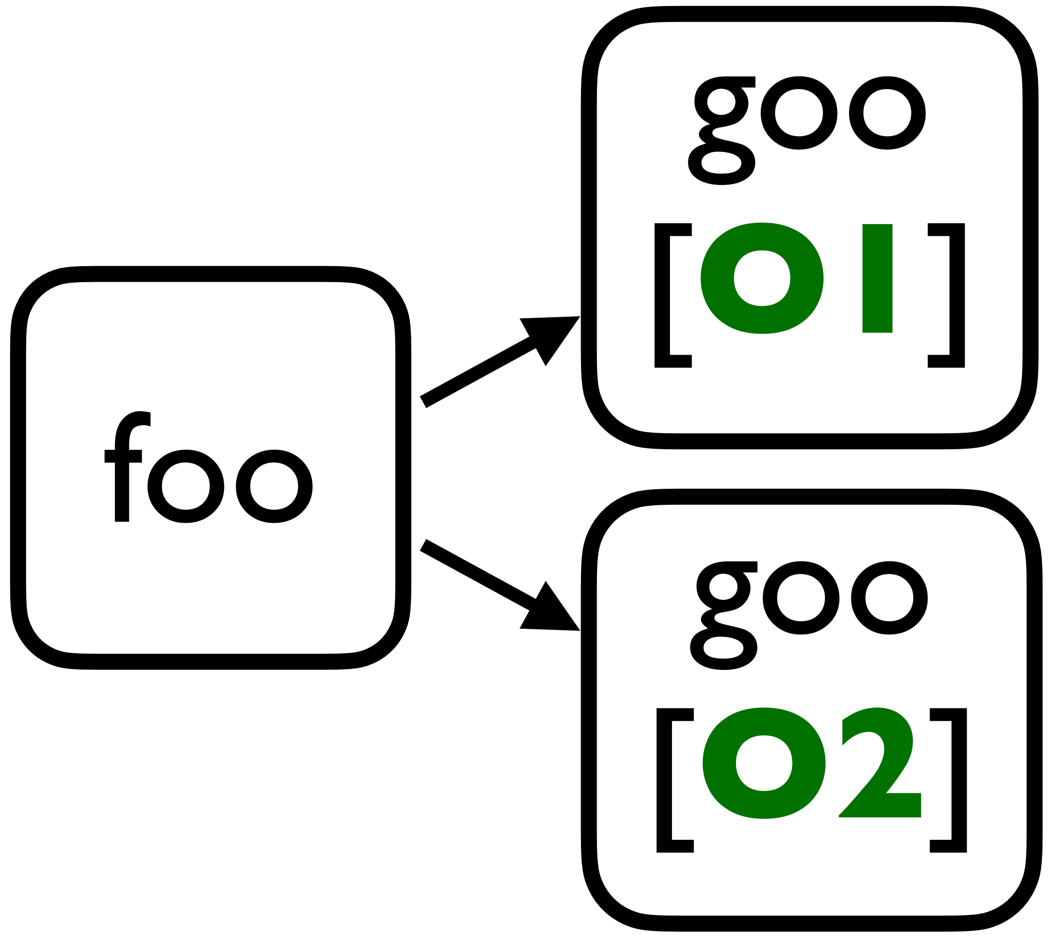
Call graph

값 기반 요약 (2002)

- "**무엇**"을 기반으로 요약

O1 or **O2**

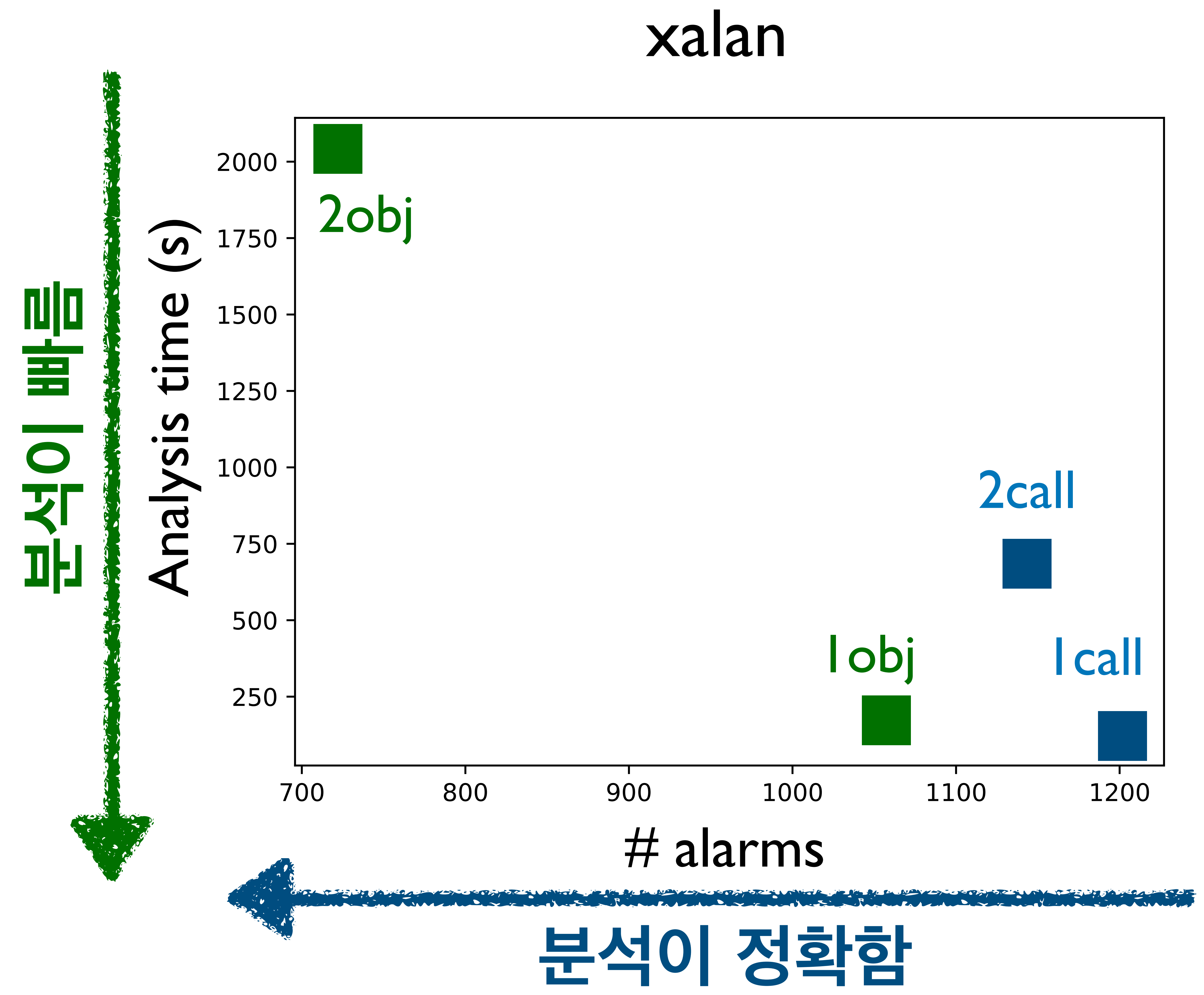
```
0: foo(p){
1:   p.goo();
2: }
```



Call graph

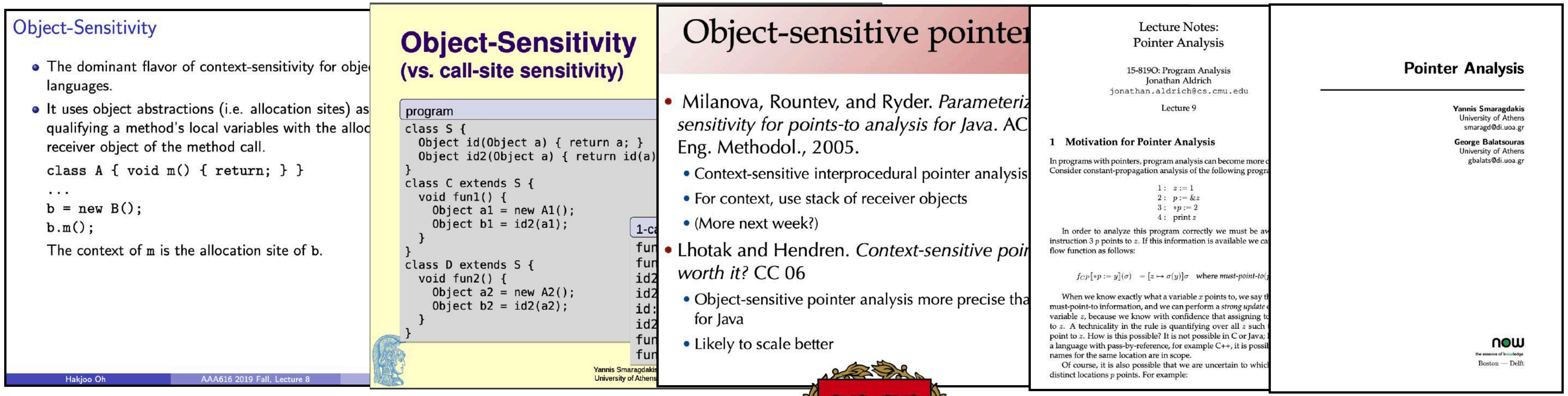
1981 — **2002** — **2010** — **2022**

위치 기반 요약 vs 값 기반 요약



1981 2002 2010 2022

- 값 기반 요약 방식이 우수하다고 강의되는 중



KOREA UNIVERSITY

National and Kapodistrian University of Athens

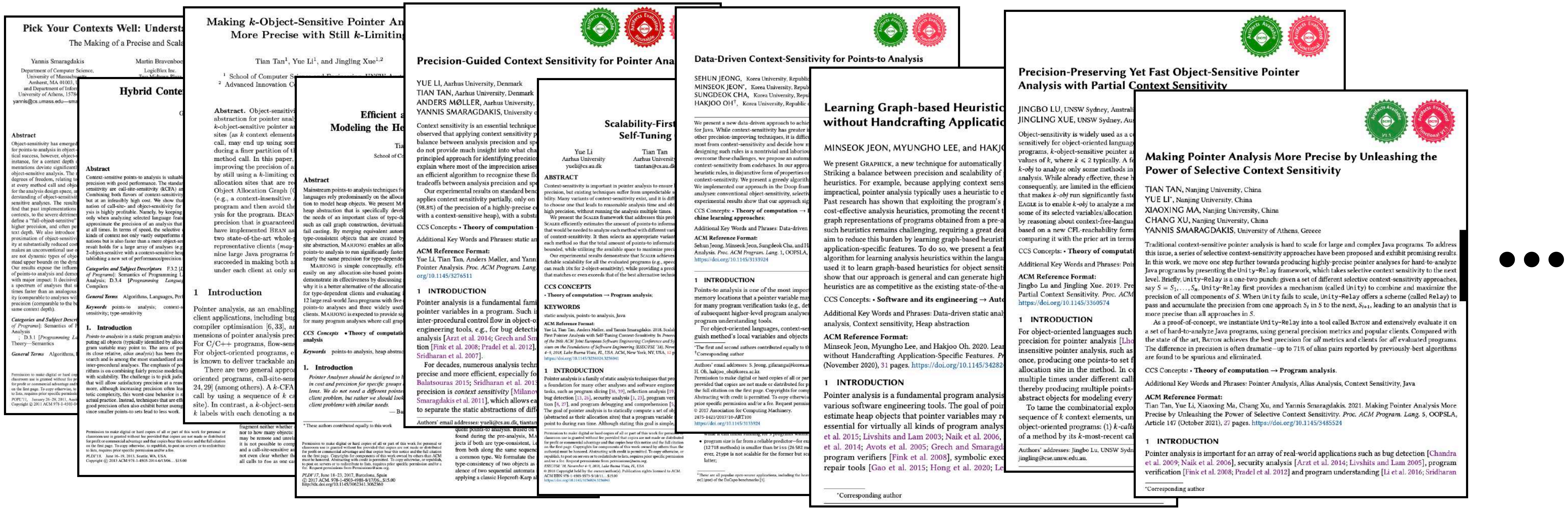
HARVARD

Carnegie Mellon University

now the essence of knowledge

...

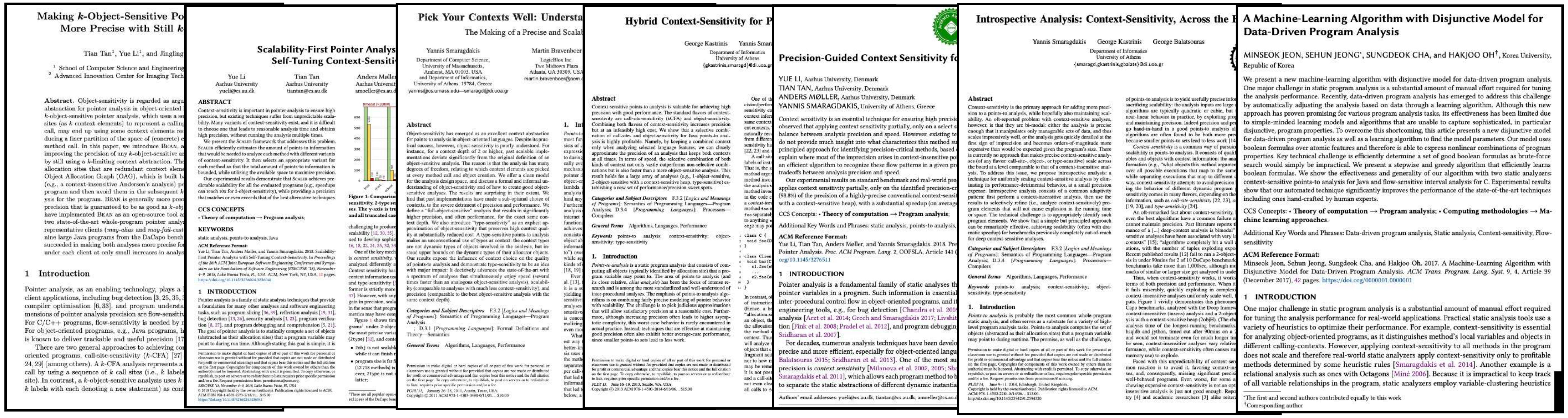
값 기반 요약을 대상으로 하는 연구가 쏟아짐



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

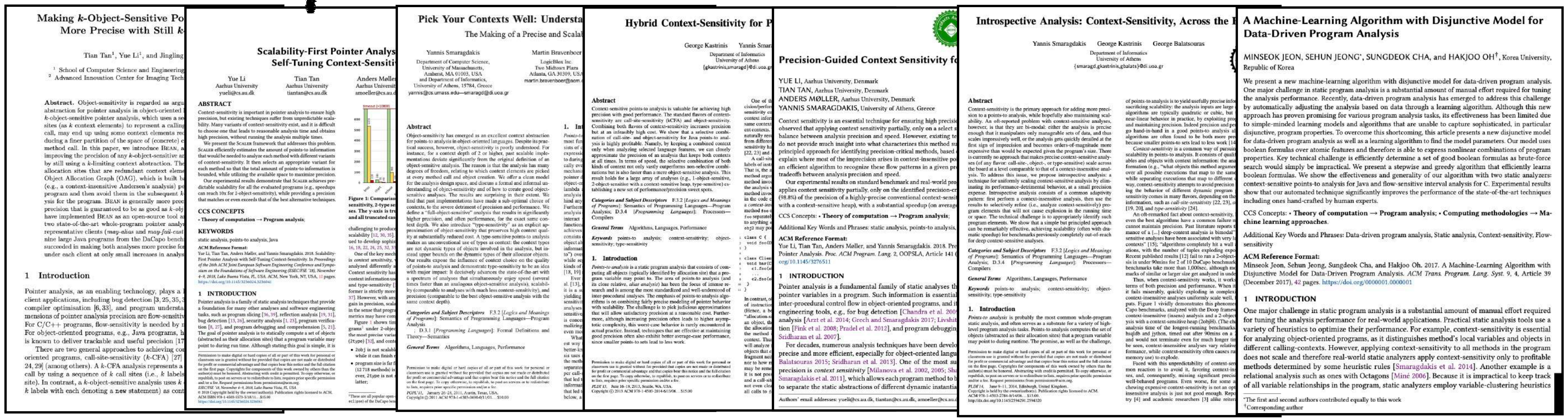
2010

2022

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]

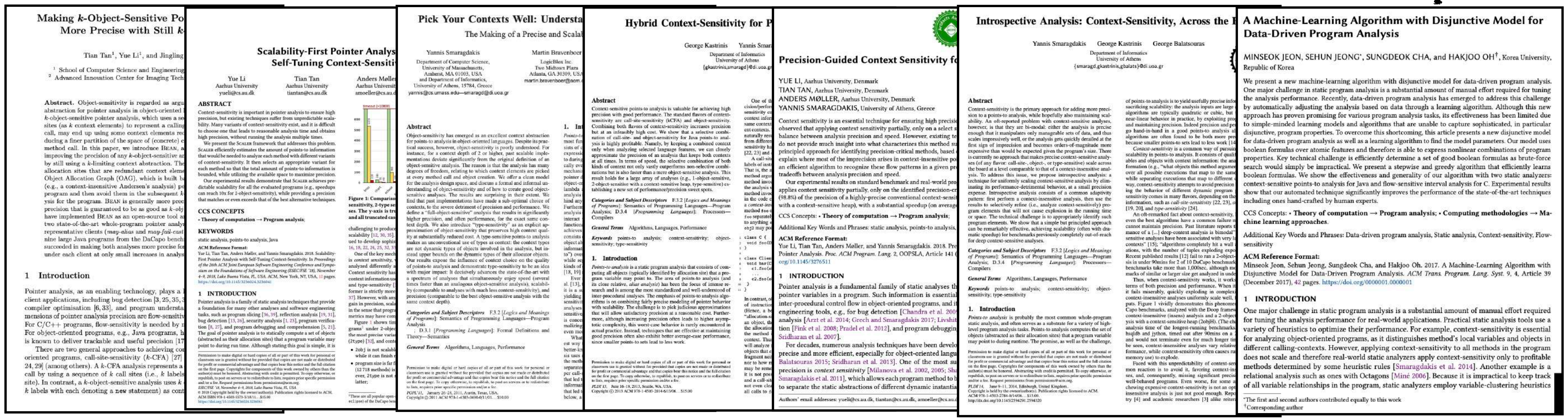


1981 2002 2010 2022

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]

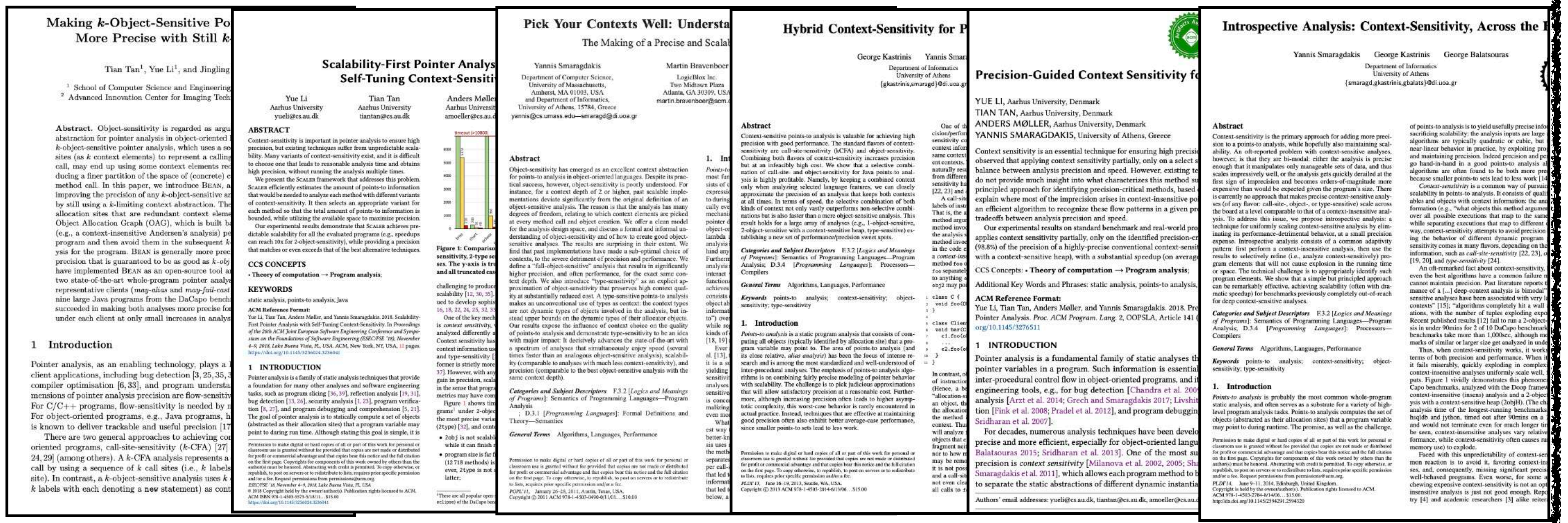


1981 2002 2010 2022

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]



1981 2002 2010 2022

Precise and Scalable Points-to Analysis via Data-Driven Context Tunneling

MINSEOK JEON, Korea University, Republic of Korea
SEHUN JEONG, Korea University, Republic of Korea
HAKJOO OH*, Korea University, Republic of Korea

We present context tunneling, a new approach for making k-limited context-sensitive points-to analysis precise and scalable. As context-sensitivity holds the key to the development of precise and scalable points-to analysis, a variety of techniques for context-sensitivity have been proposed. However, existing approaches such as k-call-site-sensitivity or k-object-sensitivity have a significant weakness that they unconditionally update the context of a method at every call-site, allowing important context elements to be overwritten by more recent, but not necessarily more important, context elements. In this paper, we show that this is a key limiting factor of existing context-sensitive analyses, and demonstrate that remarkable increase in both precision and scalability can be gained by maintaining important context elements only. Our approach, called context tunneling, updates contexts selectively and decides when to propagate the same context without modification.

We attain context tunneling via a data-driven approach. The effectiveness of context tunneling is very sensitive to the choice of important context elements. Even worse, precision is not monotonically increasing with respect to the ordering of the choices. As a result, manually coming up with a good heuristic rule for context tunneling is extremely challenging and likely fails to maximize its potential. We address this challenge by developing a specialized data-driven algorithm, which is able to automatically search for high-quality heuristics over the non-monotonic space of context tunneling.

We implemented our approach in the Doop framework and applied it to four major flavors of context-sensitivity: call-site-sensitivity, object-sensitivity, type-sensitivity, and hybrid context-sensitivity. In all cases, 1-context-sensitive analysis with context tunneling far outperformed deeper context-sensitivity with $k = 2$ in both precision and scalability.

CCS Concepts: • **Theory of computation → Program analysis**; • **Computing methodologies → Machine learning approaches**;

Additional Key Words and Phrases: Points-to analysis, Context-sensitive analysis, Data-driven program analysis

ACM Reference Format:
Minseok Jeon, Sehun Jeong, and Hakjoo Oh. 2018. Precise and Scalable Points-to Analysis via Data-Driven Context Tunneling. *Proc. ACM Program. Lang.* 2, OOPSLA, Article 140 (November 2018), 30 pages. https://doi.org/10.1145/3276510

*Corresponding author

Authors' addresses: Minseok Jeon, minseok_jeon@korea.ac.kr, Department of Computer Science and Engineering, Korea University, 145, Anam-ro, Sungbuk-gu, Seoul, 02841, Republic of Korea; Sehun Jeong, gifaranga@korea.ac.kr, Department of Computer Science and Engineering, Korea University, 145, Anam-ro, Sungbuk-gu, Seoul, 02841, Republic of Korea; Hakjoo Oh, hakjoo_oh@korea.ac.kr, Department of Computer Science and Engineering, Korea University, 145, Anam-ro, Sungbuk-gu, Seoul, 02841, Republic of Korea.

2475-1421/2018/11-ART140
https://doi.org/10.1145/3276510

컨텍스트 터널링

패러다임 변화

마지막 k ⟶ 주요 k

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-

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

Call 승리!

~~마지막 k~~

주요 k

값 기반 요약
(Object sensitivity)

vs

위치 기반 요약
(Call-site Sensitivity)

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? 정확함 ~~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.

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-oriente… means for inc… s and Smara… 2015; Smara…

The… harir and Pr… fer in the ch… nueli 1981] … uses allocat… alysis uses the allocation site of the receiver object … 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!

~~마지막 k~~

주요 k

값 기반 요약
(Object sensitivity)

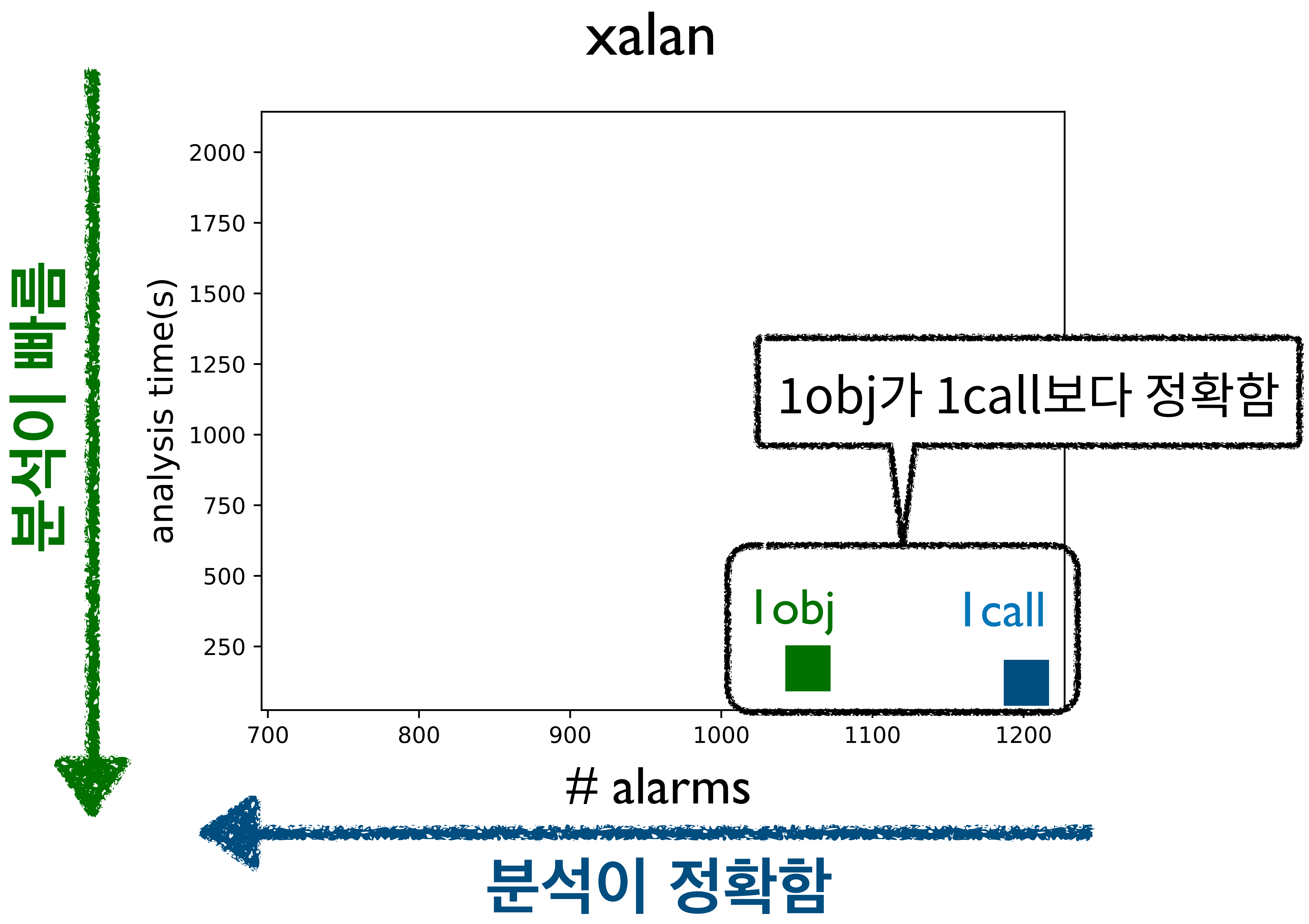
vs

위치 기반 요약
(Call-site Sensitivity)

1981 2002 2010 2018 **2022**

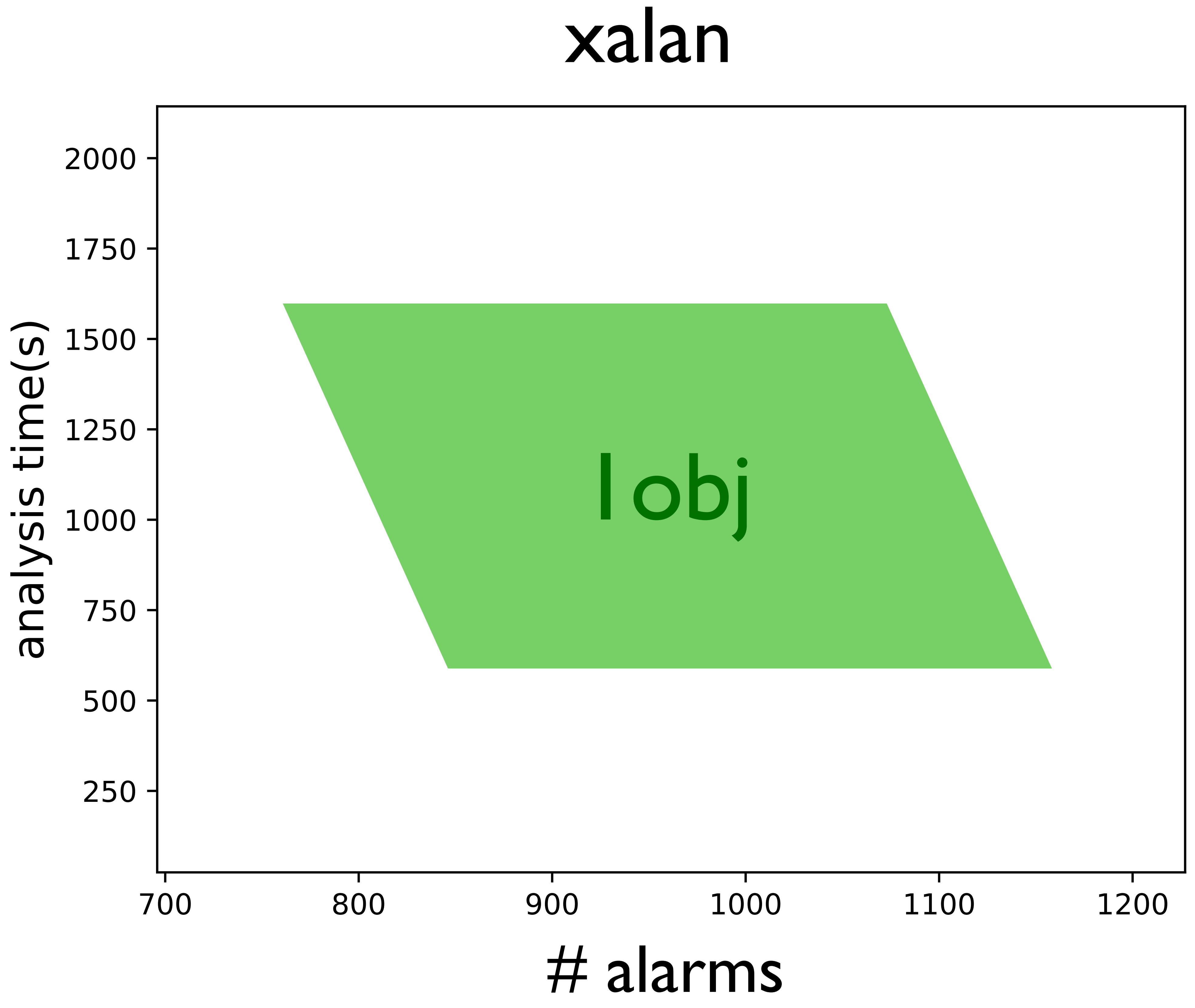
마지막 k 기반 요약에서의 성능 비교

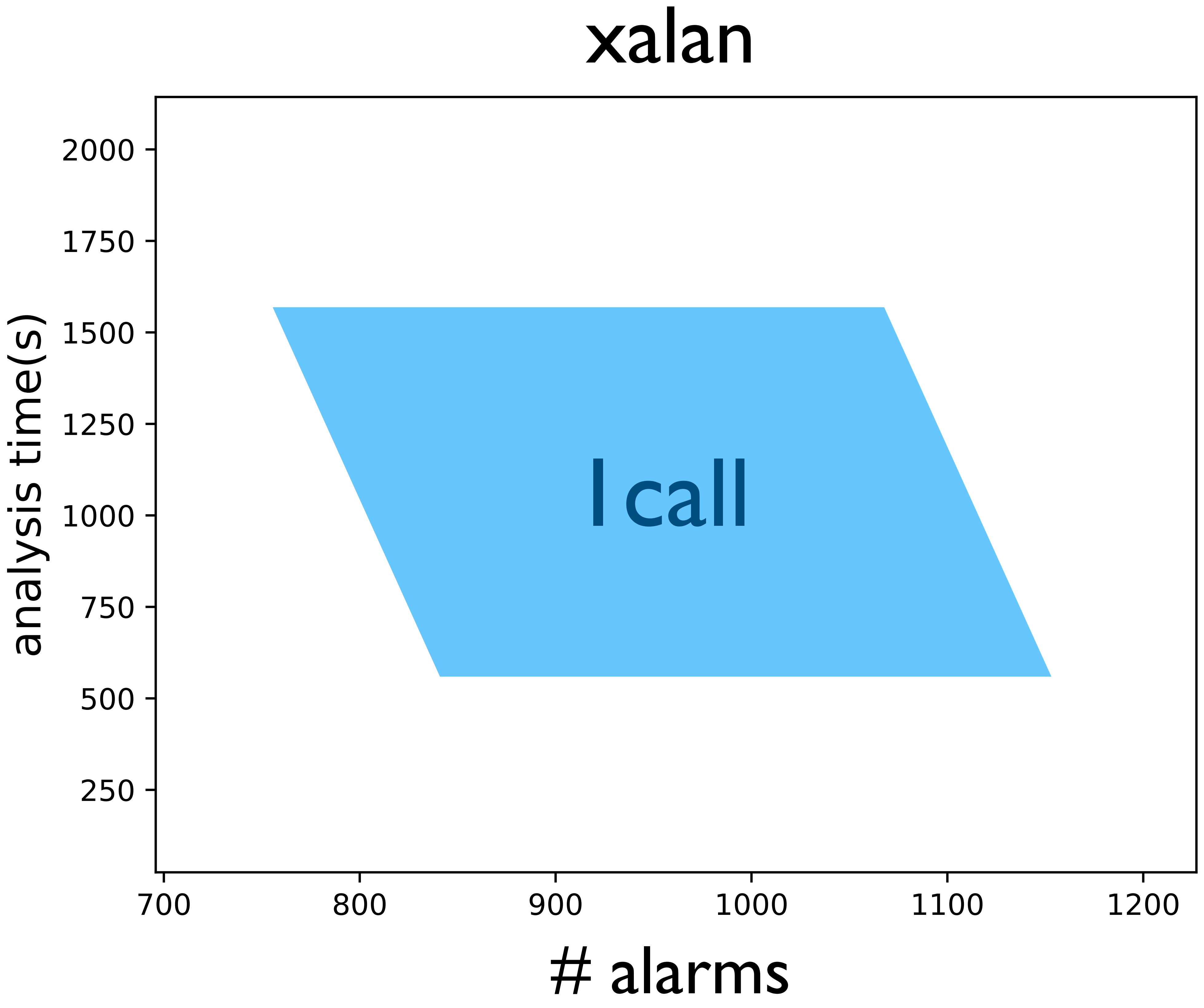
- 분석의 성능을 직접 비교하면 됨



주요 k 기반에서의 성능 비교

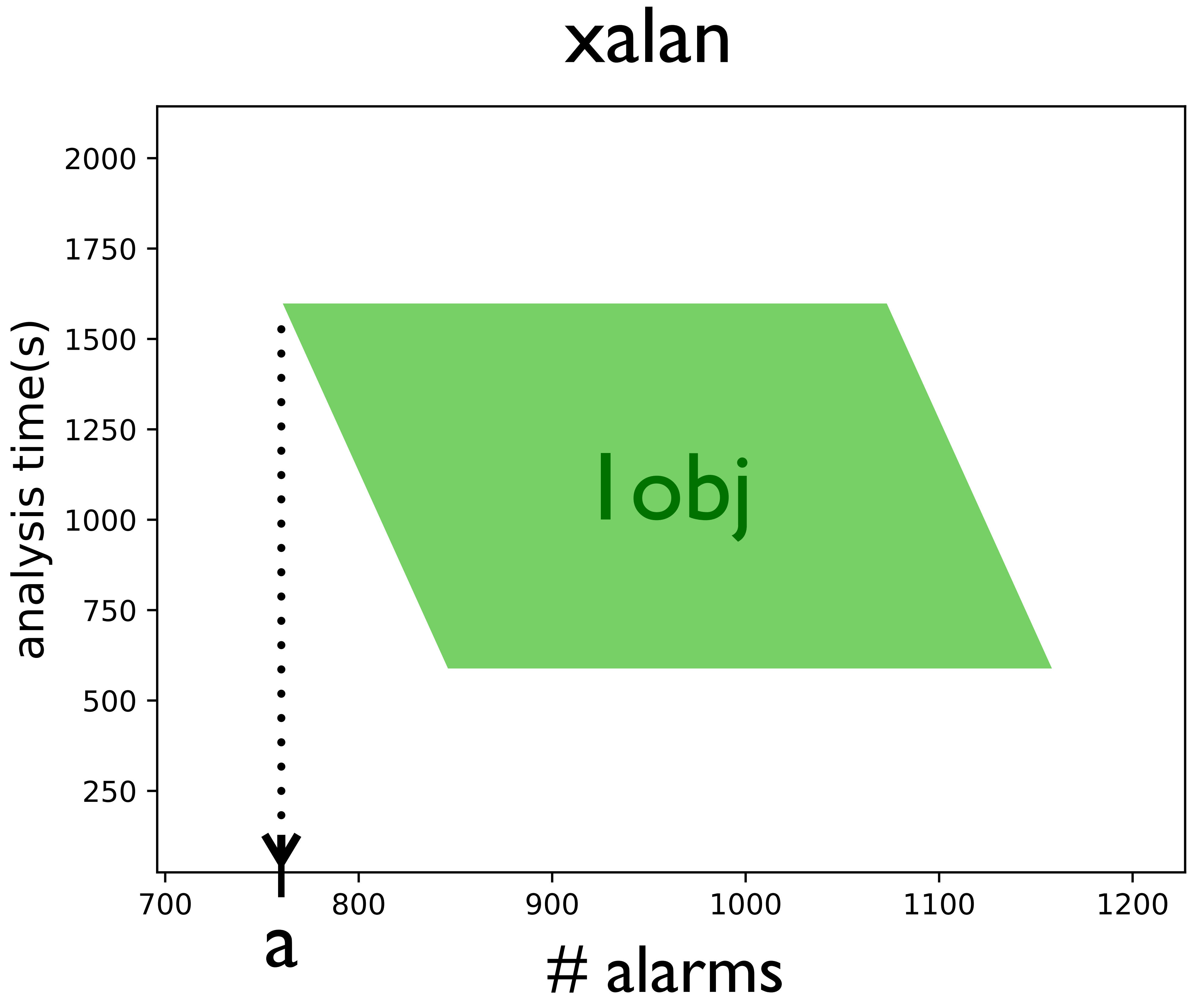
- 주요 요소 분류에 따라 성능이 바뀜

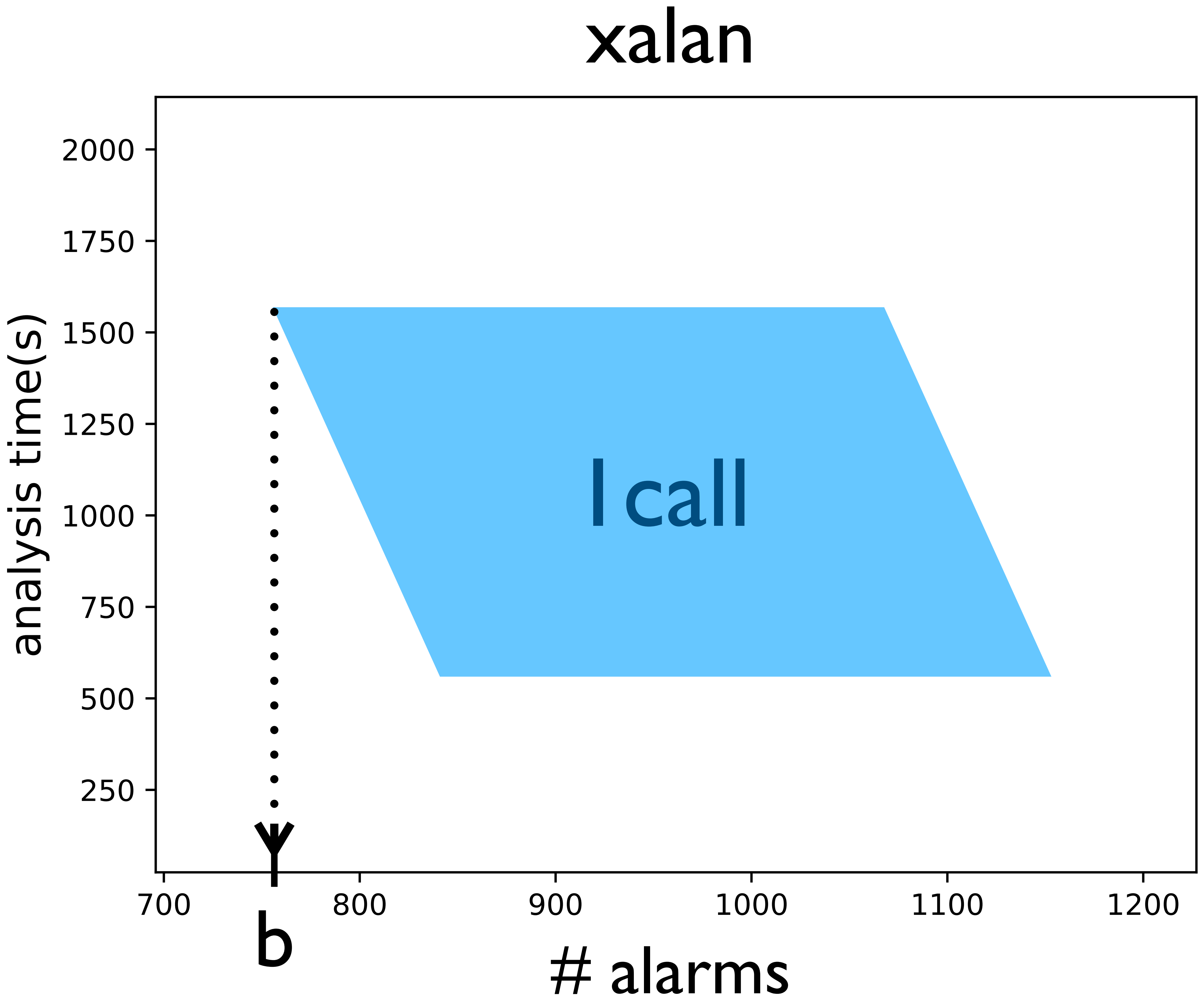




주요 k 기반에서의 성능 비교

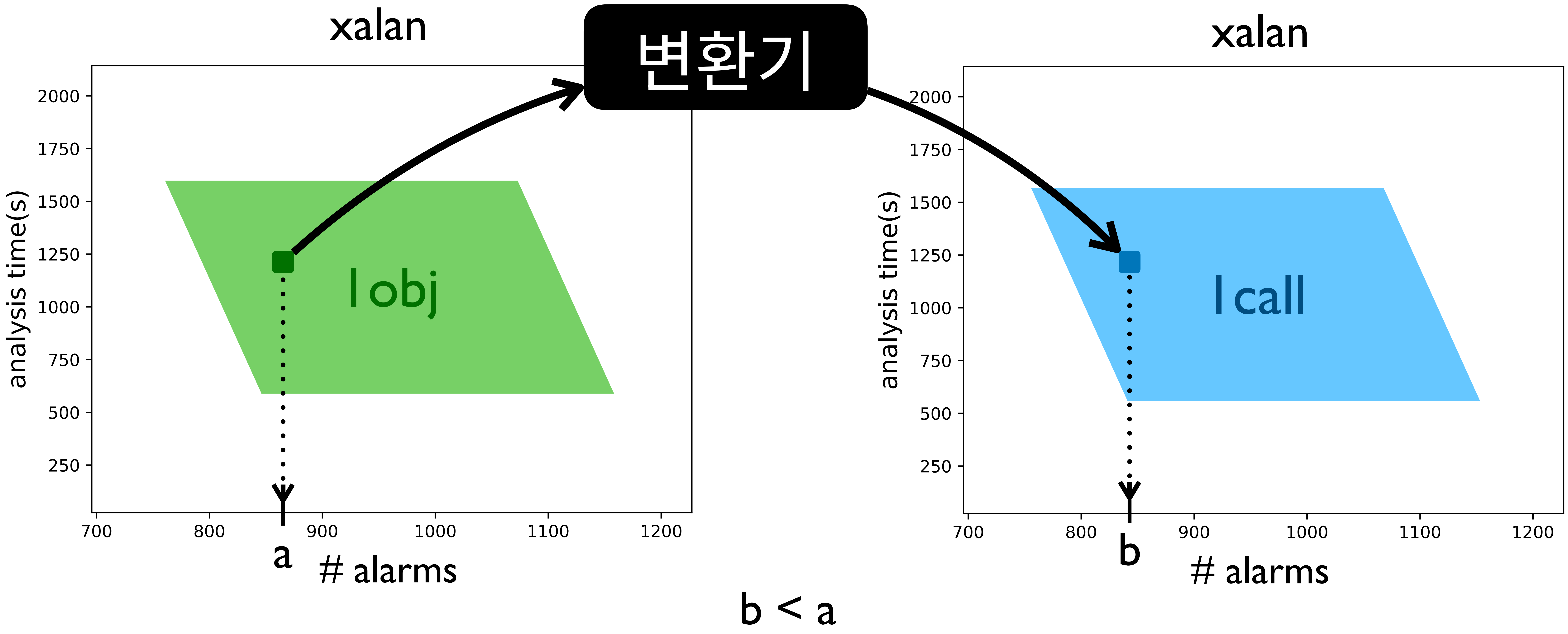
- **방법 1:** 최고 성능을 내는 분류를 알아내서 성능 비교하기 (e.g., b < a)





주요 k 기반에서의 성능 비교

- **방법 2:** 주어진 obj를 더 정확한 call로 변환해주는 함수가 **존재한다**는 것을 보이기



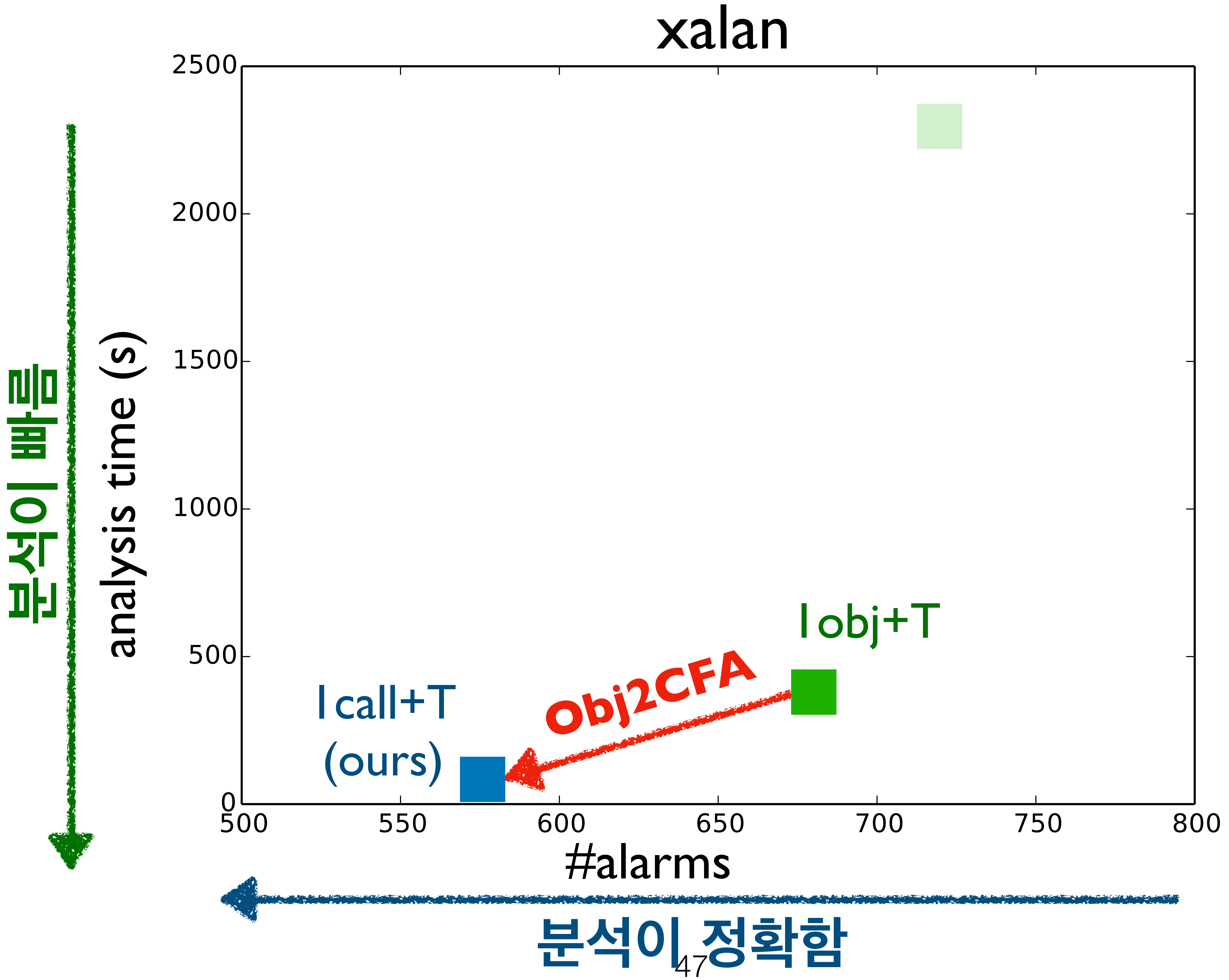
변환기 : Obj2CFA

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

Obj 분석 결과 → **Obj2CFA** → CFA를 위한 **주요 요소**

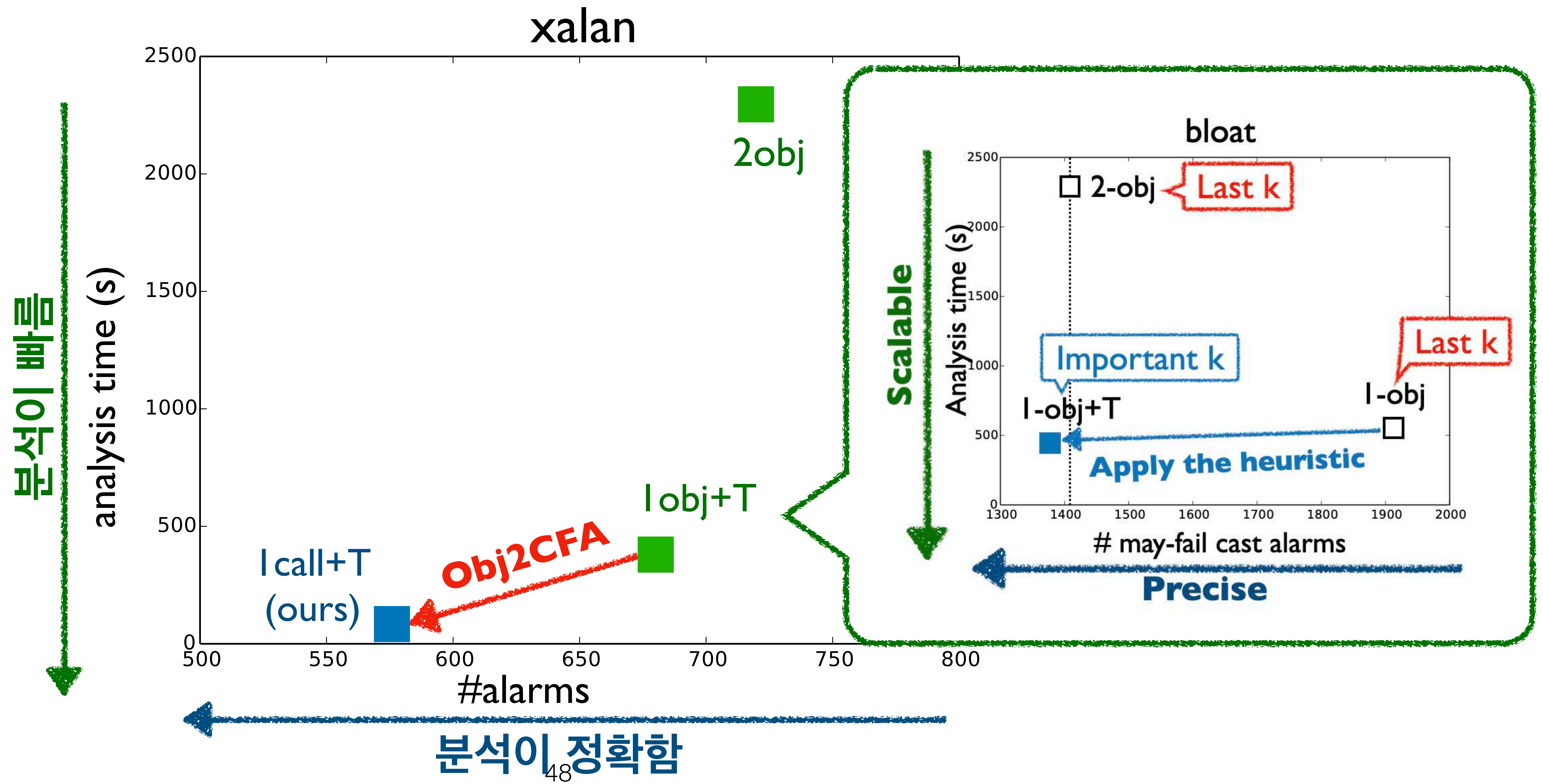
변환기 : Obj2CFA

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



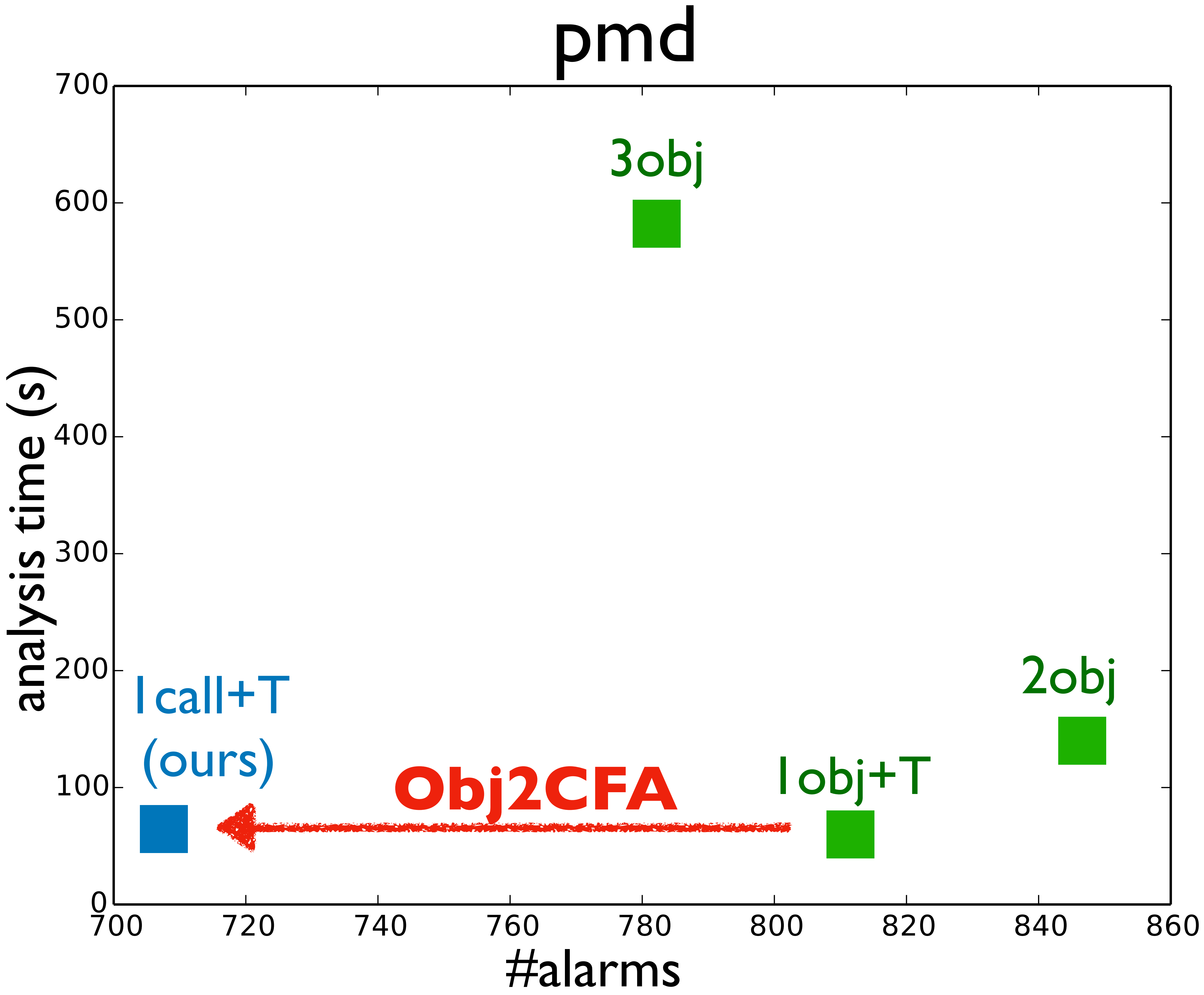
변환기 : Obj2CFA

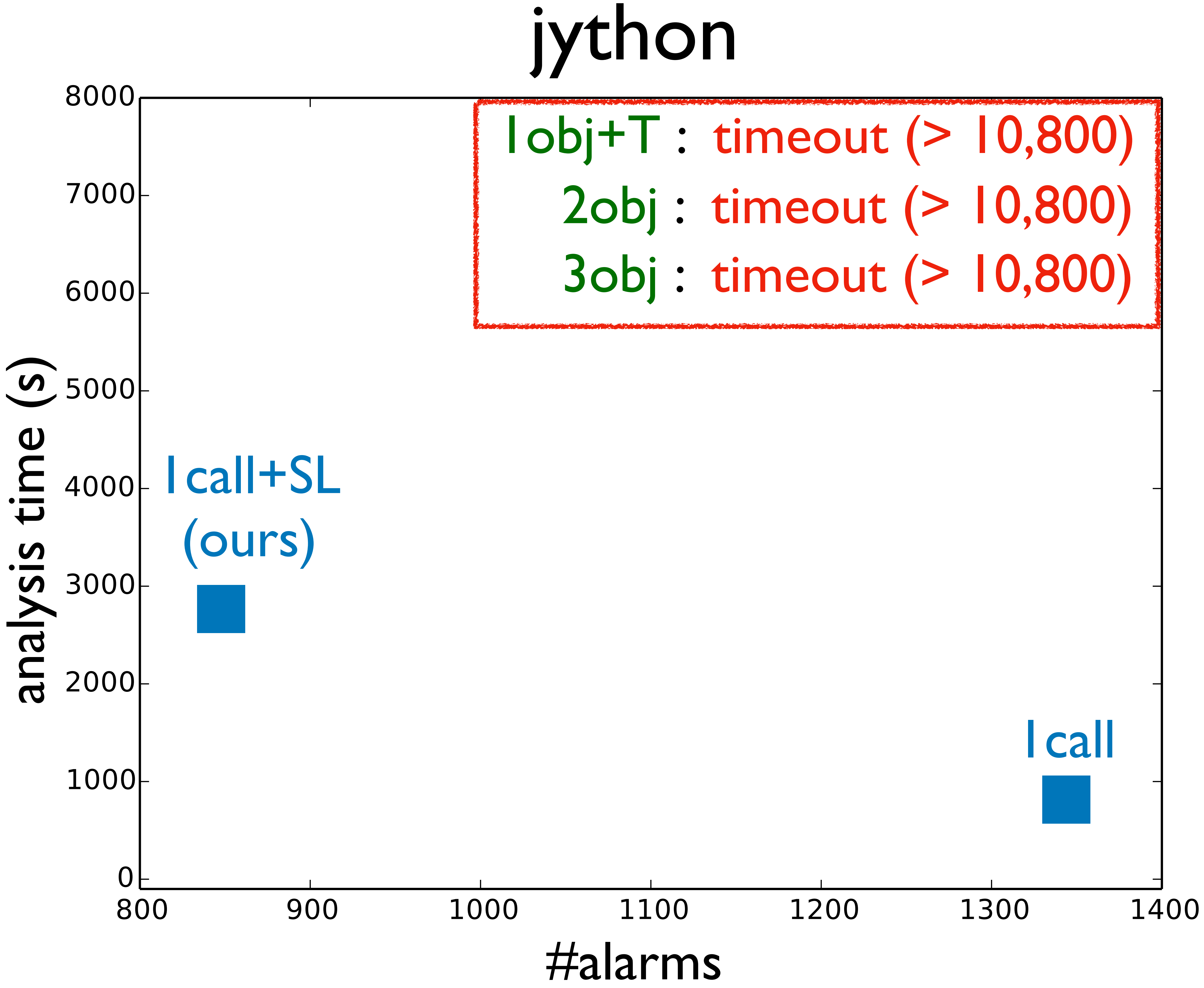
- **Obj2CFA** 는 주어진 값 기반 요약을 더 정확한 위치 기반 요약으로 바꾸어줌



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

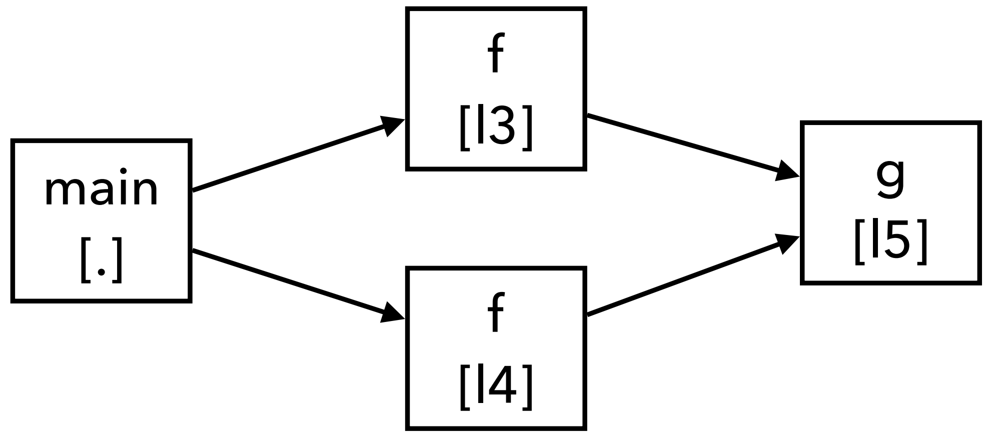
Call-Site vs. Object Sensitivity

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

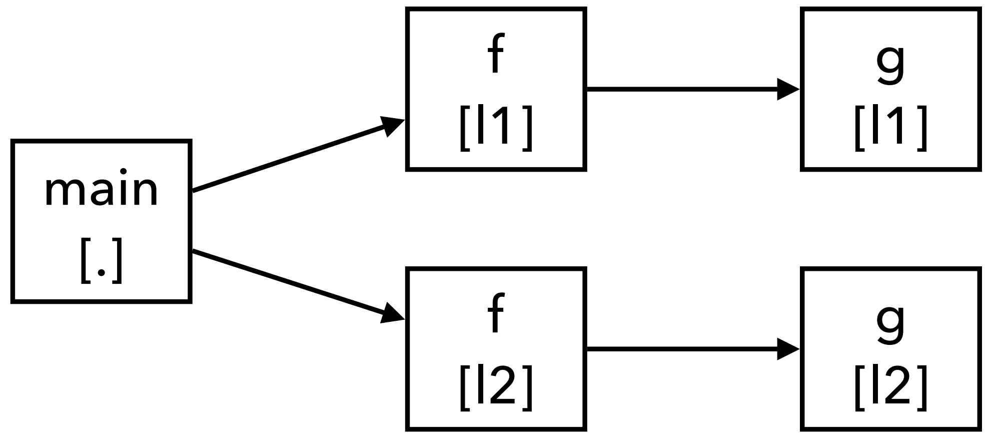
Call-site vs. Object Sensitivity

- Typical example that benefits from object sensitivity:

```
class A:
  def g(self):
     return
  def f(self):
     return self.g() // l5

def main():
   a = A()   // l1
   b = A()   // l2
   a.f()     // l3
   b.f()     // l4
```



1-call-site sensitivity



1-object sensitivity

고정관념에 도전하기

- 목표: 주요 k기반 요약 방식을 표준으로 만들기

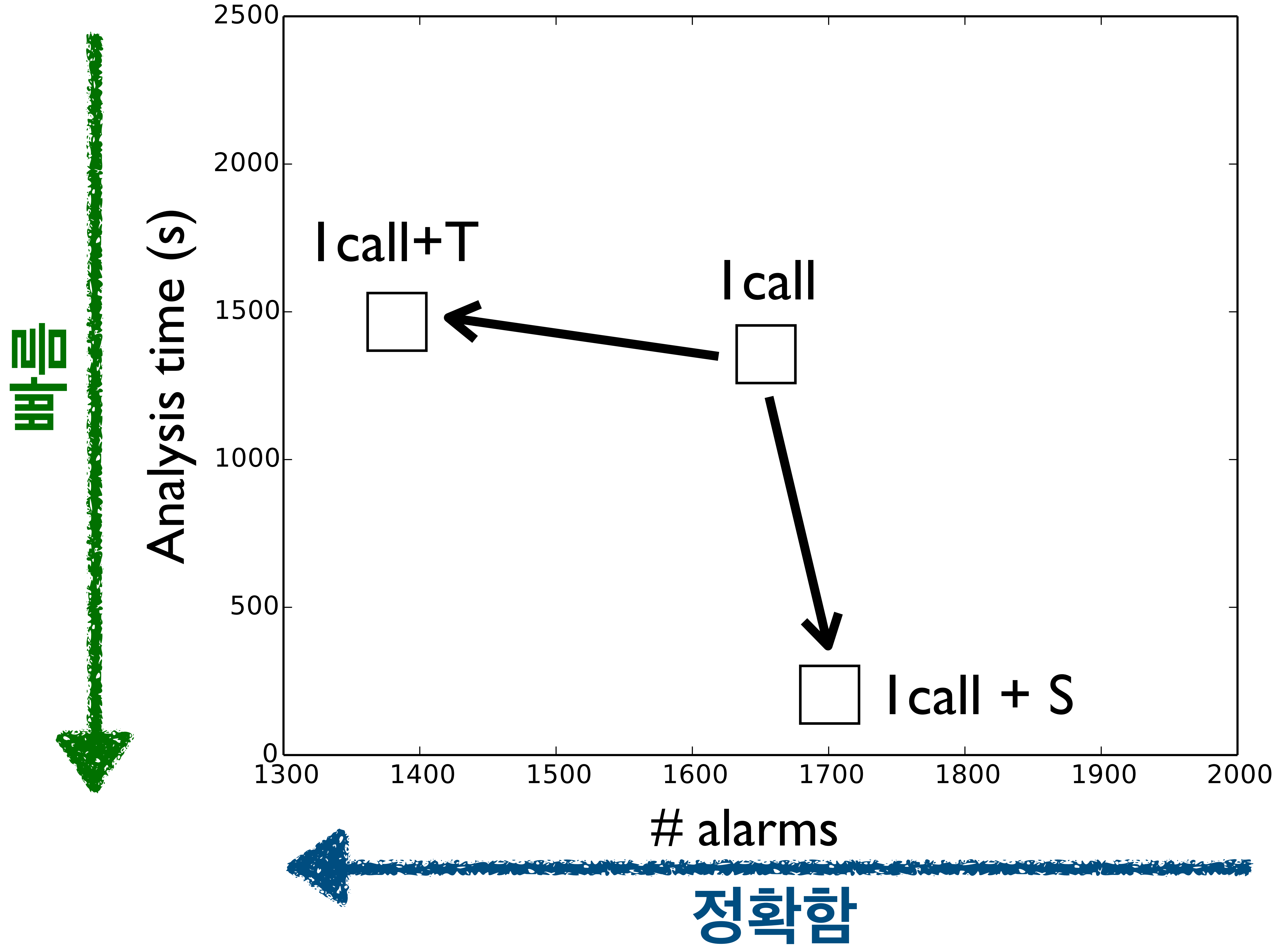
Future work

Tunneling → Obj2CFA → **submitted** →

고정관념에 허물기 3

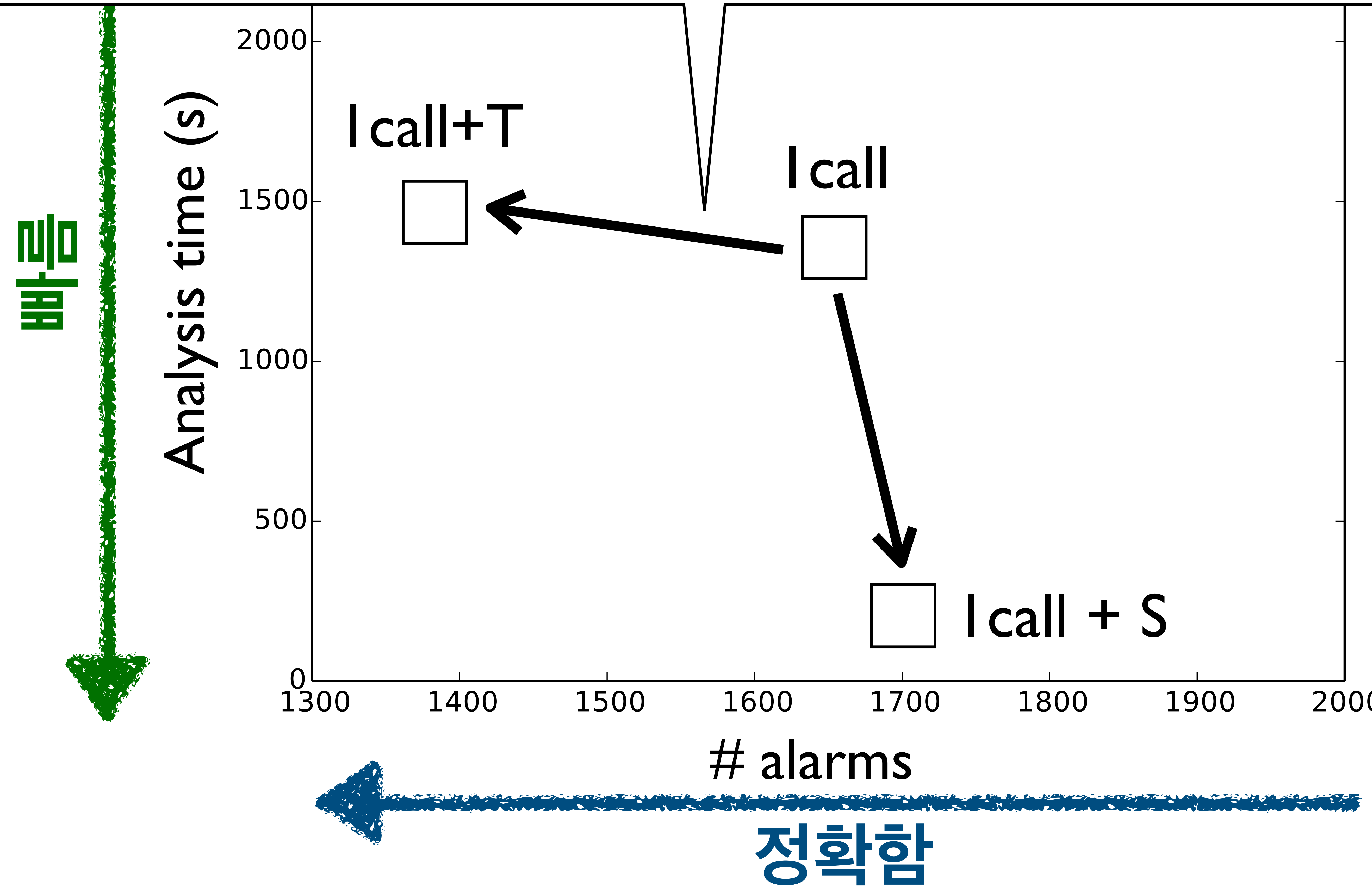
분석 기술들을 각각 개발 → 분석 기술들을 함께 개발

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

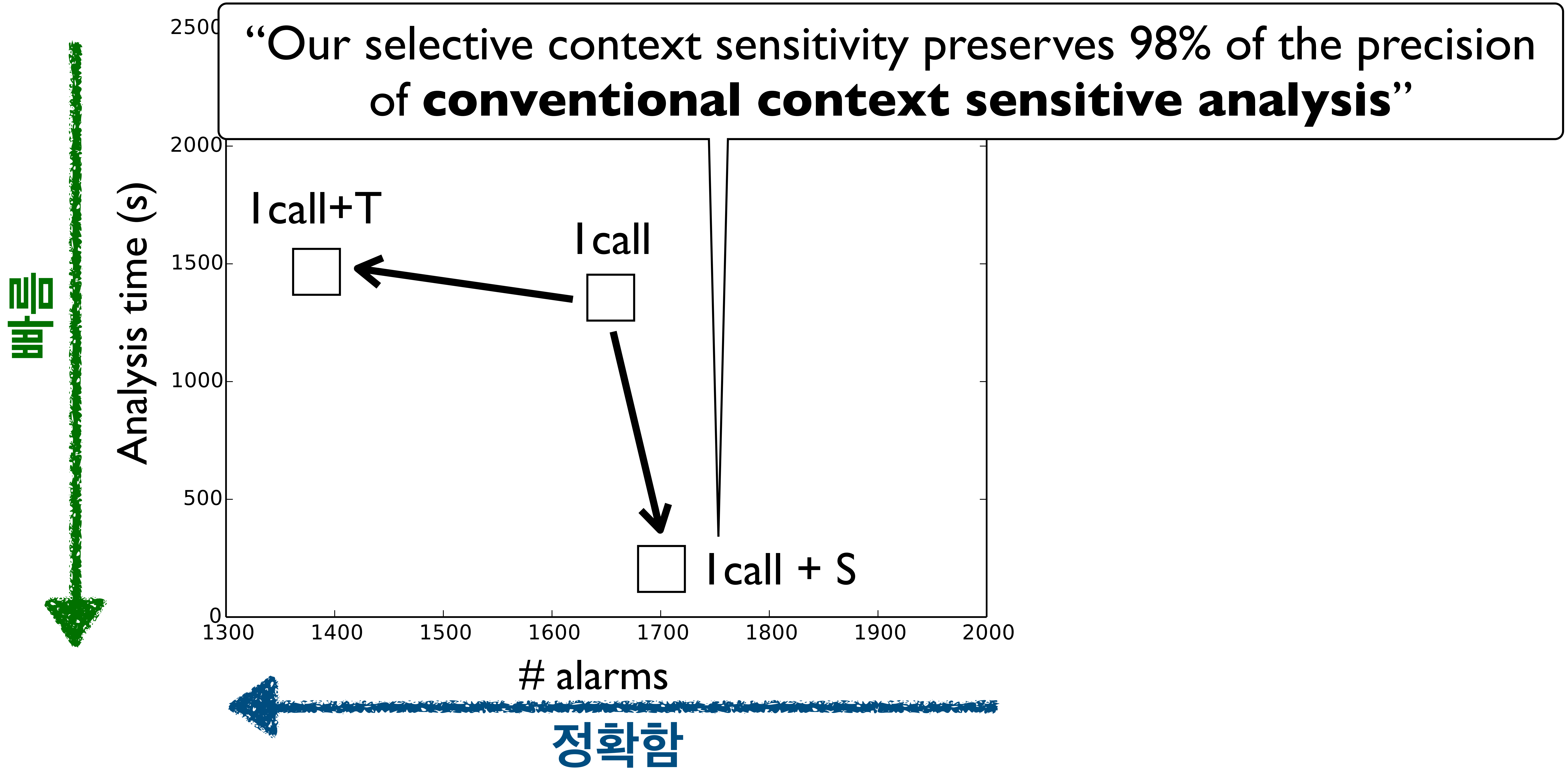


- 호출 환경 배달하기: 속도를 유지한채 정확도를 올려주는 기술

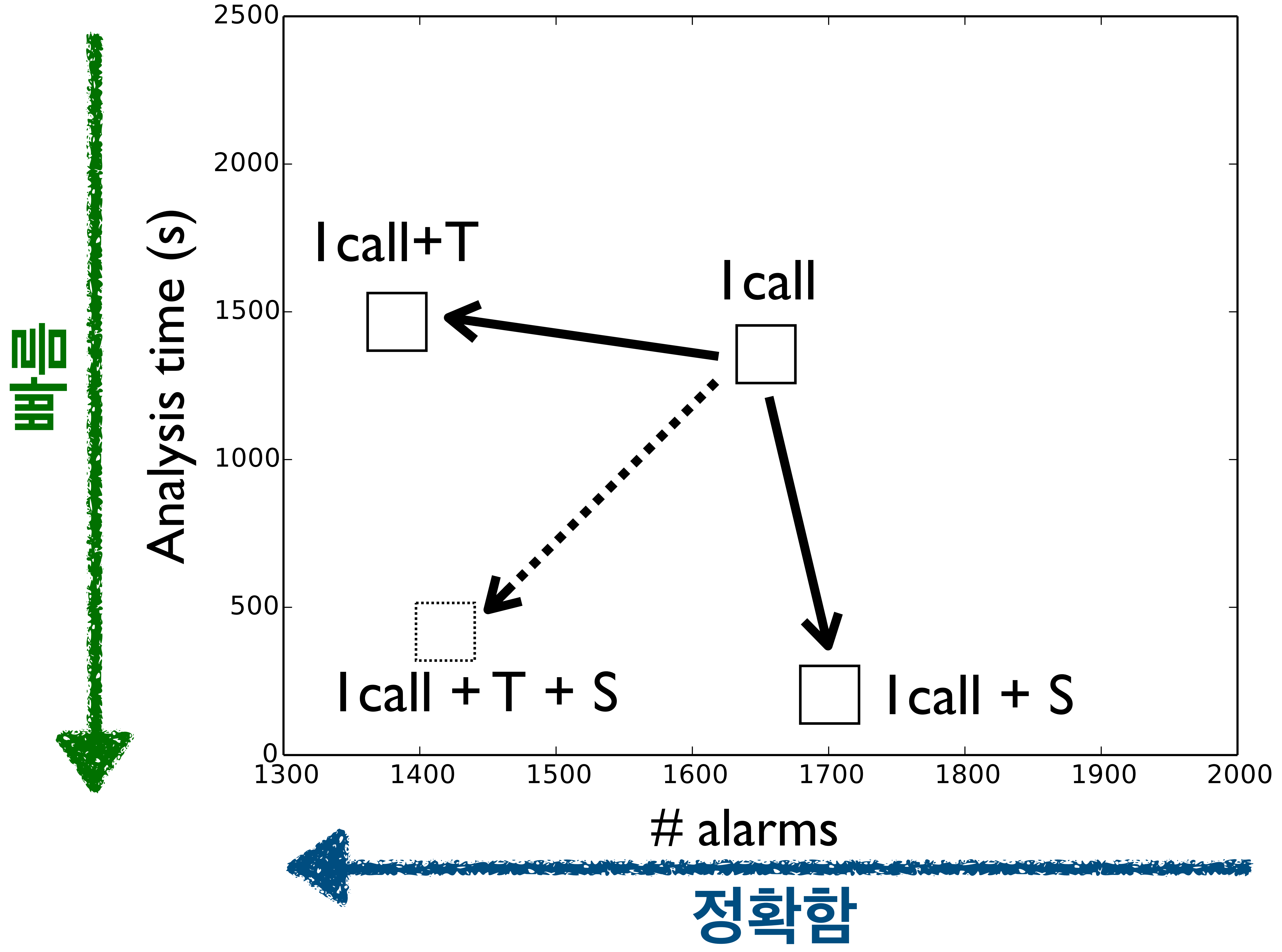
"We generated 1call+T by applying context tunneling to **1call…**"



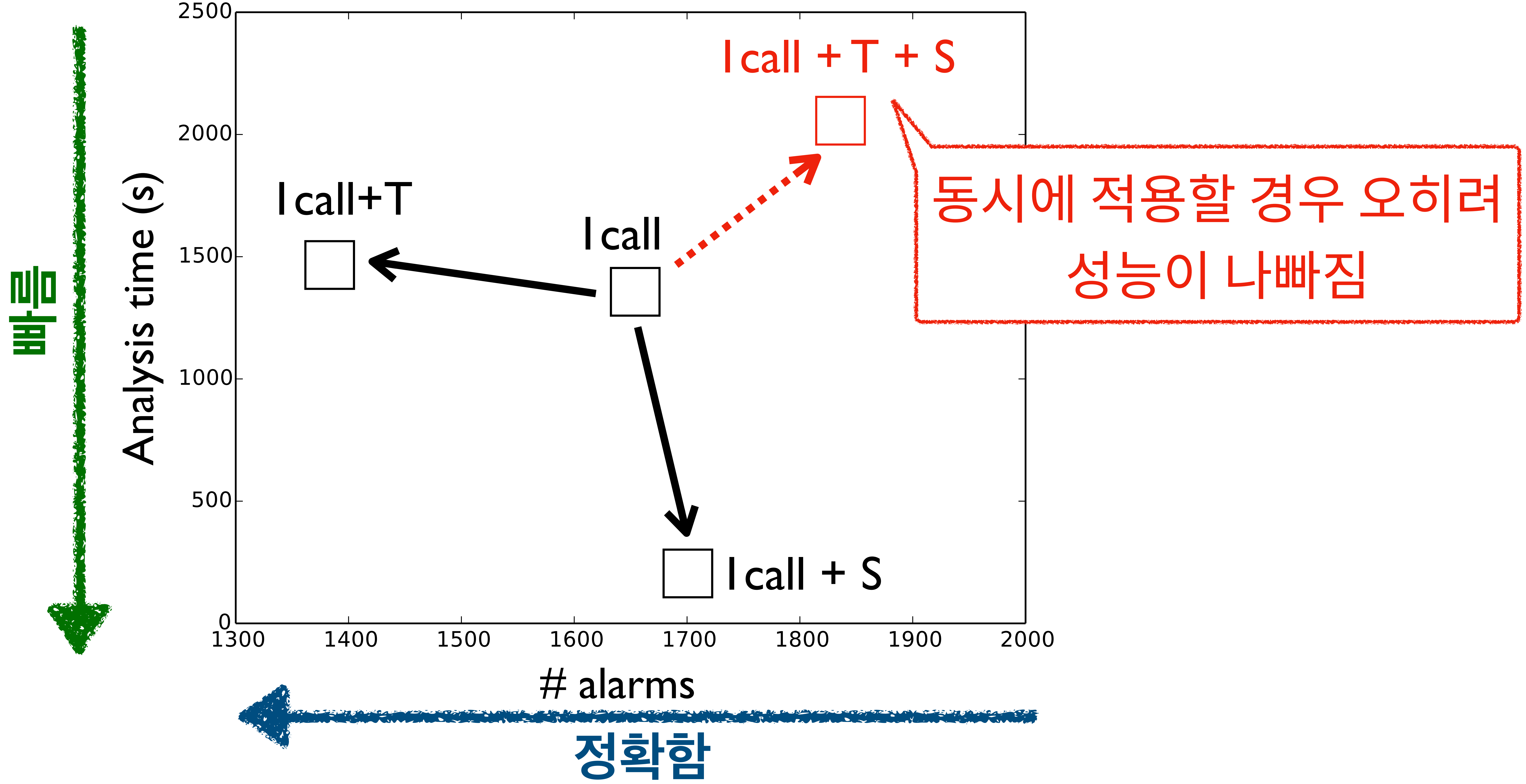
- 적당히 함수 호출 분석: 정확도를 유지한채 속도를 올려주는 기술



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

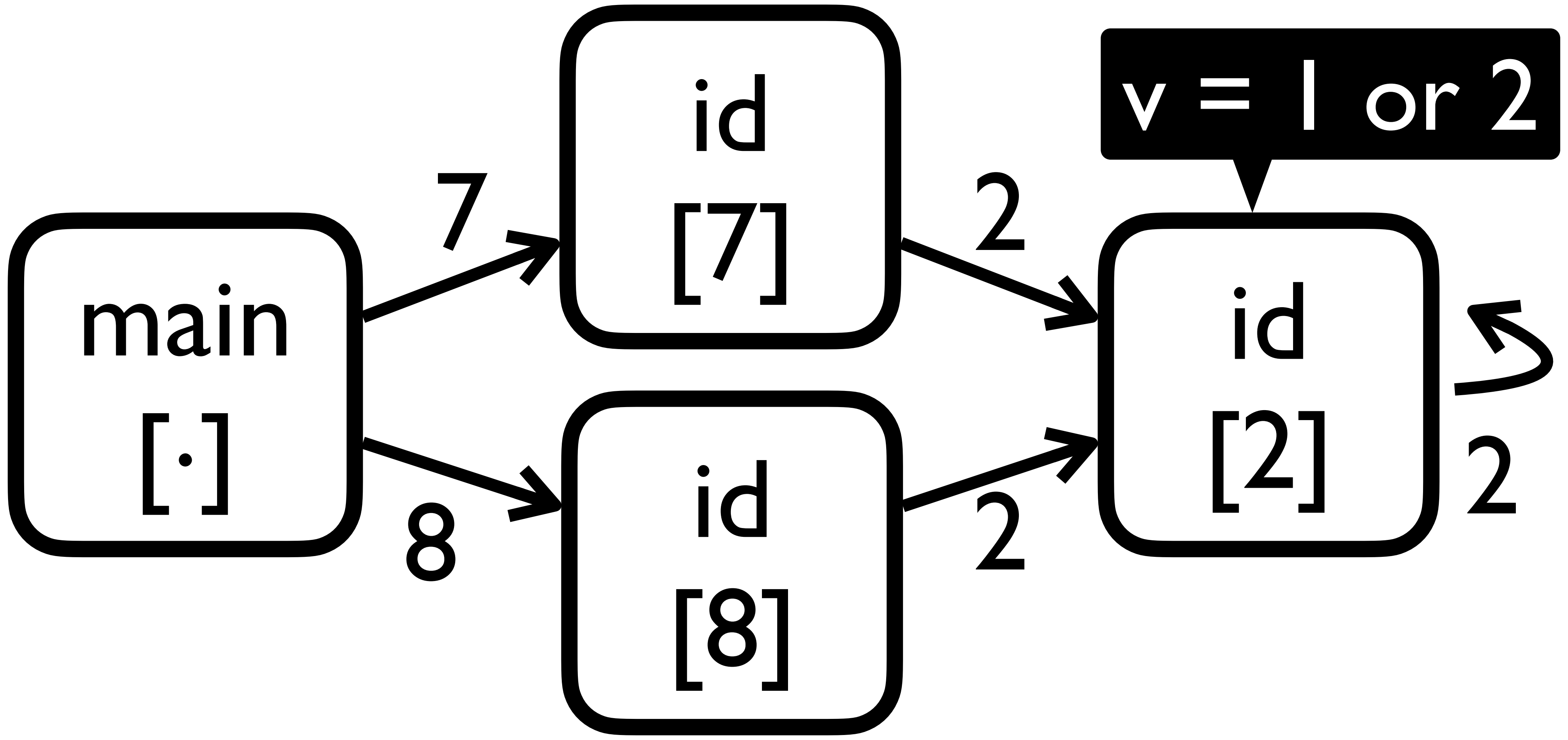


- 질문: 독립적으로 각각 개발한 기술들을 동시에 사용하면 궁극의 성능이 나올까?

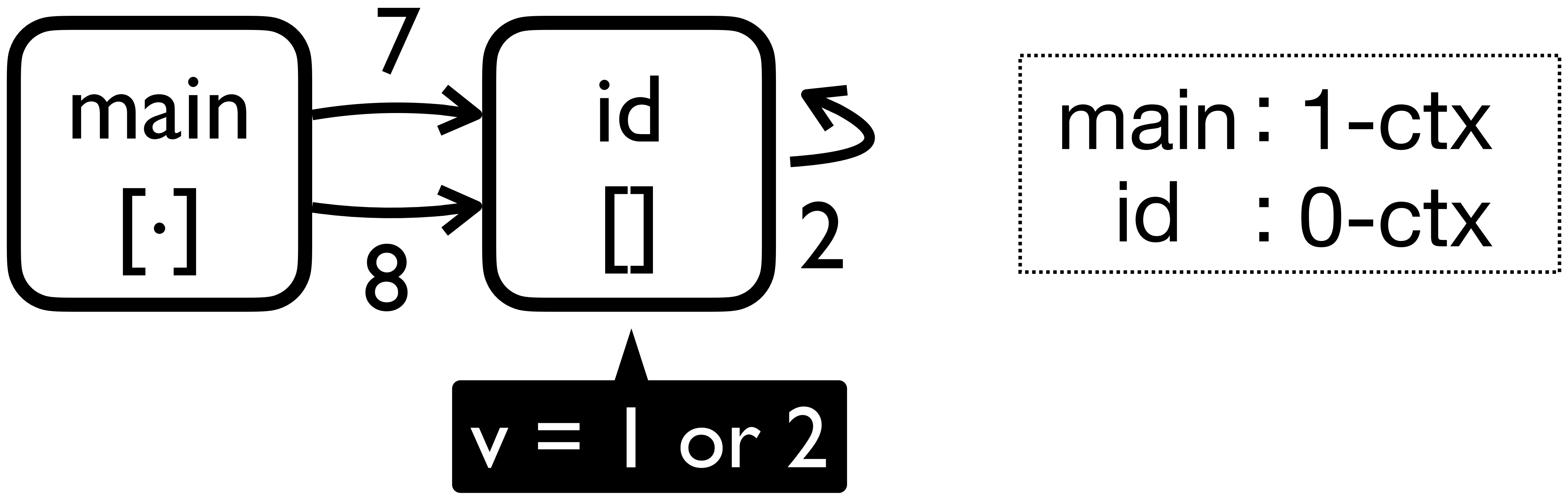


```
0:  id(v, i){
1:    if (i > 0){
2:      return id(v, i-1);}
3:    return v;}
4:
5:  main(){
6:    i = input();
7:    v1 = id(1, i);//A
8:    v2 = id(2, i);//B
9:    assert (v1 != v2);//query
10: }
```

예제 프로그램



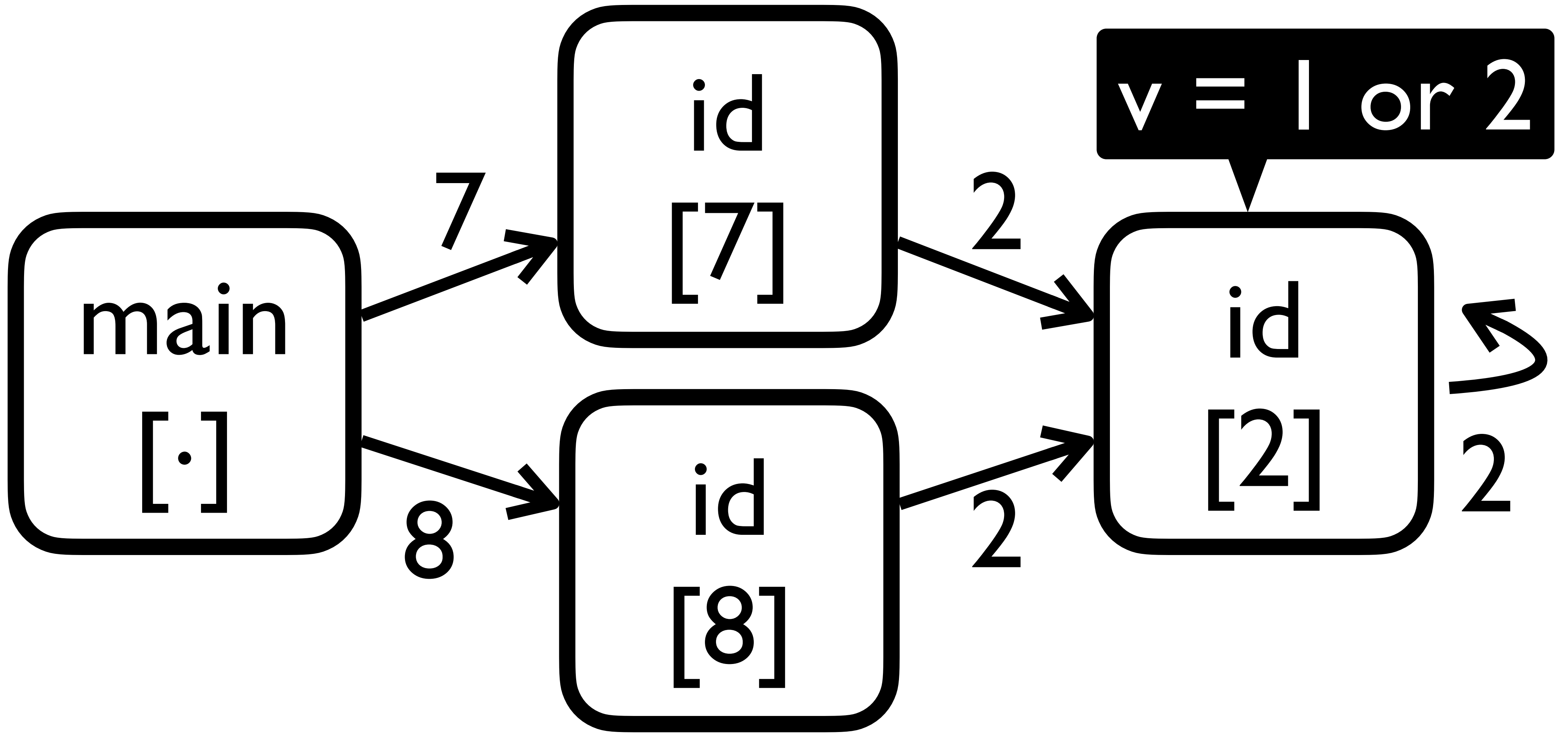
1-요소 기반 함수 호출 요약



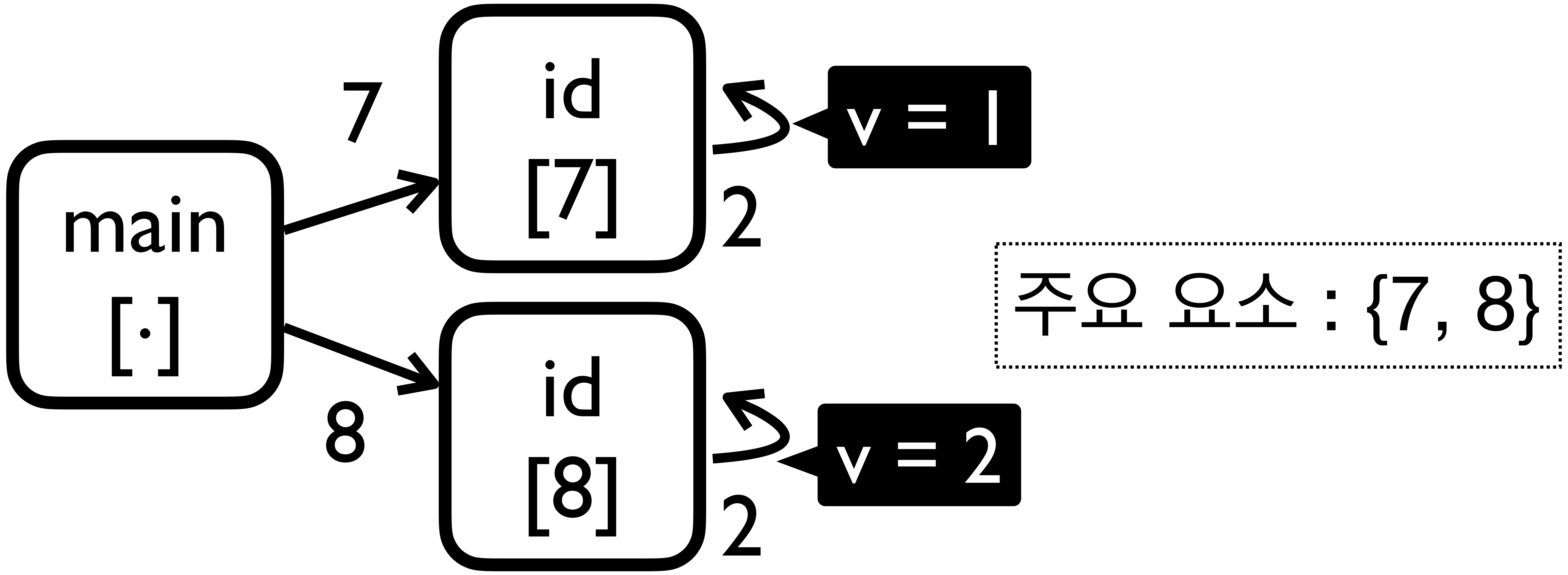
적당히 1-요소 기반 함수 호출 요약

```
id(v, i){
  if (i > 0){
    return id(v, i-1);}
  return v;}

main(){
  i = input();
  v1 = id(1, i);//A
  v2 = id(2, i);//B
  assert (v1 != v2);//query
}
```

예제 프로그램



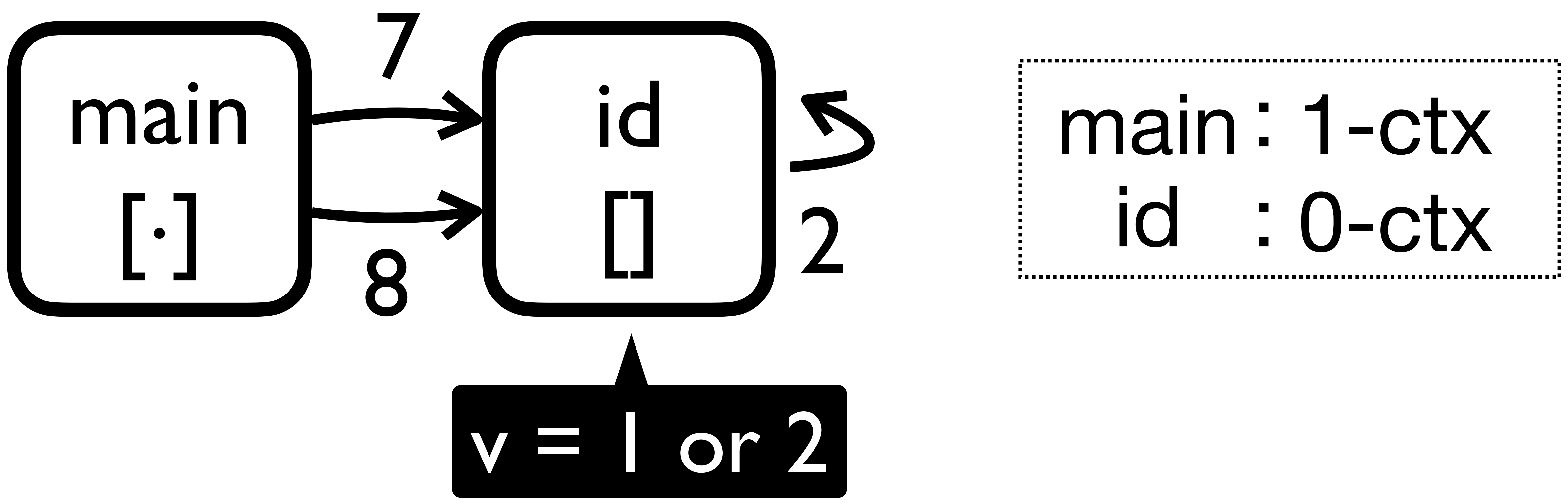
1-요소 기반 함수 호출 요약



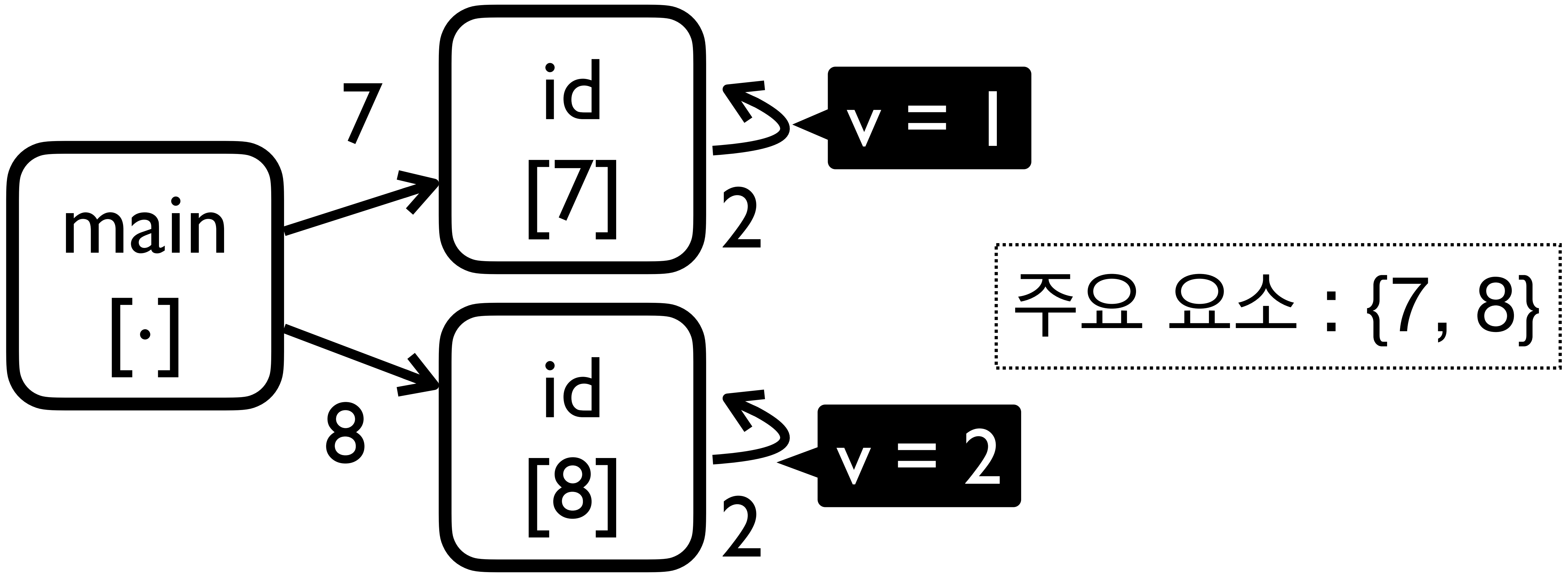
주요 1-요소 기반 함수 호출 요약

```
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



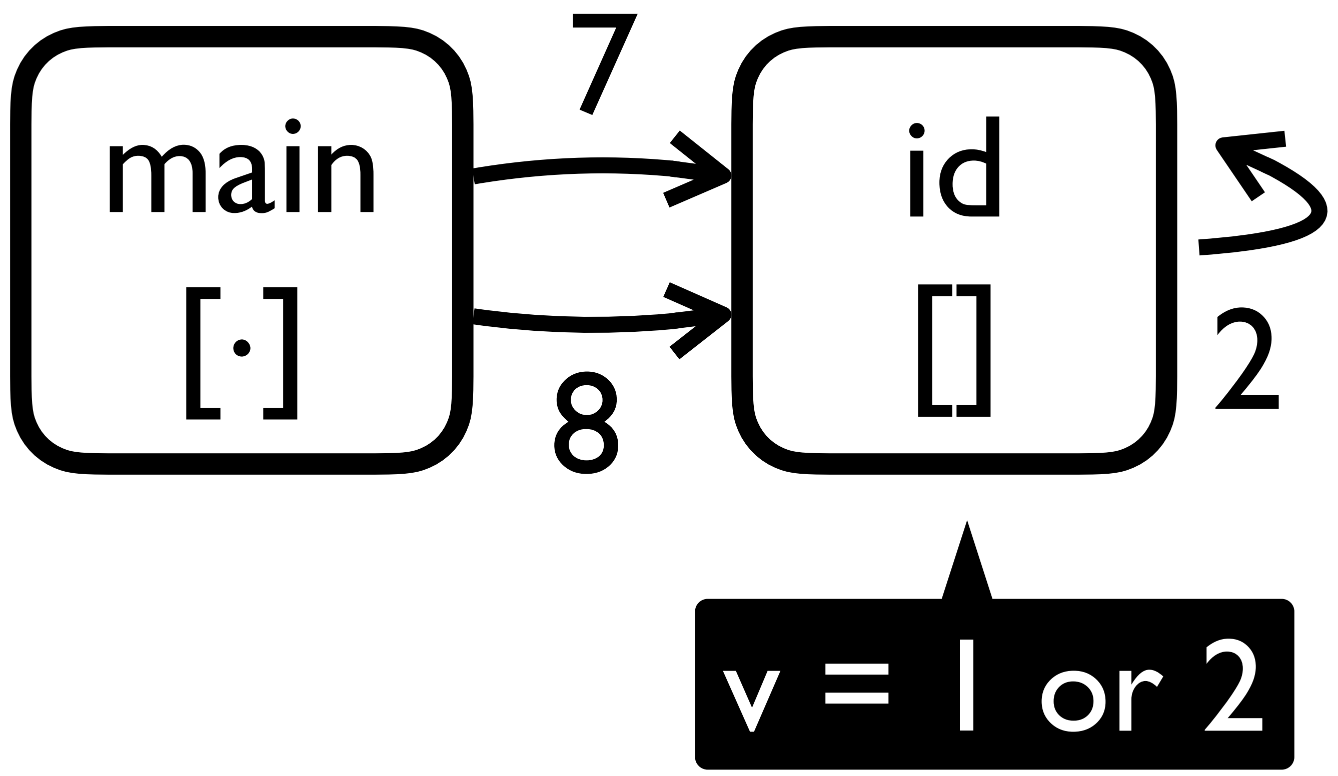
적당히 1-요소 기반 함수 호출 요약



주요 1-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



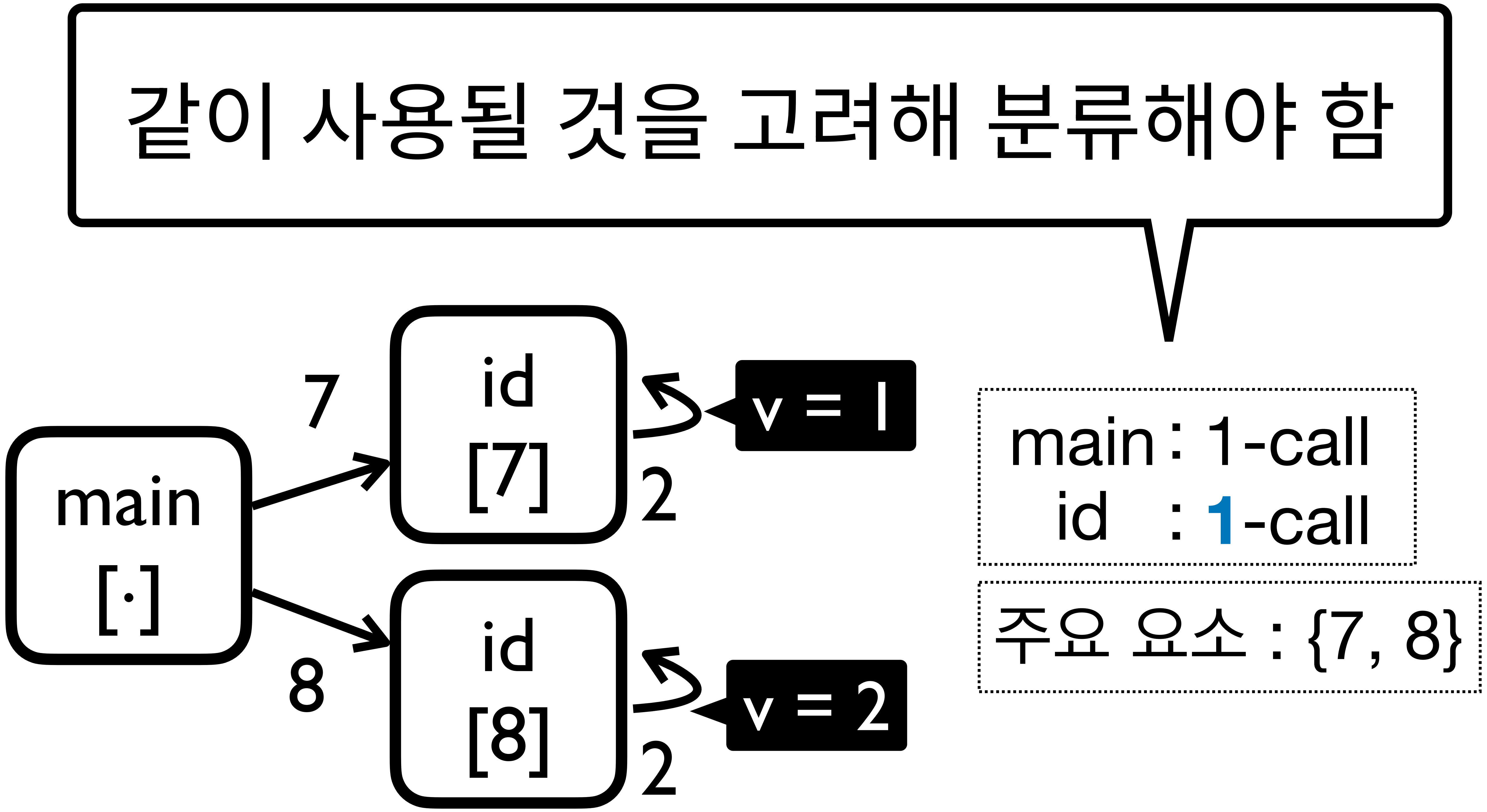
main : 1-call
id : 0-call

주요 요소 : {7, 8}

적당히 주요 1-요소 기반 함수 호출 요약

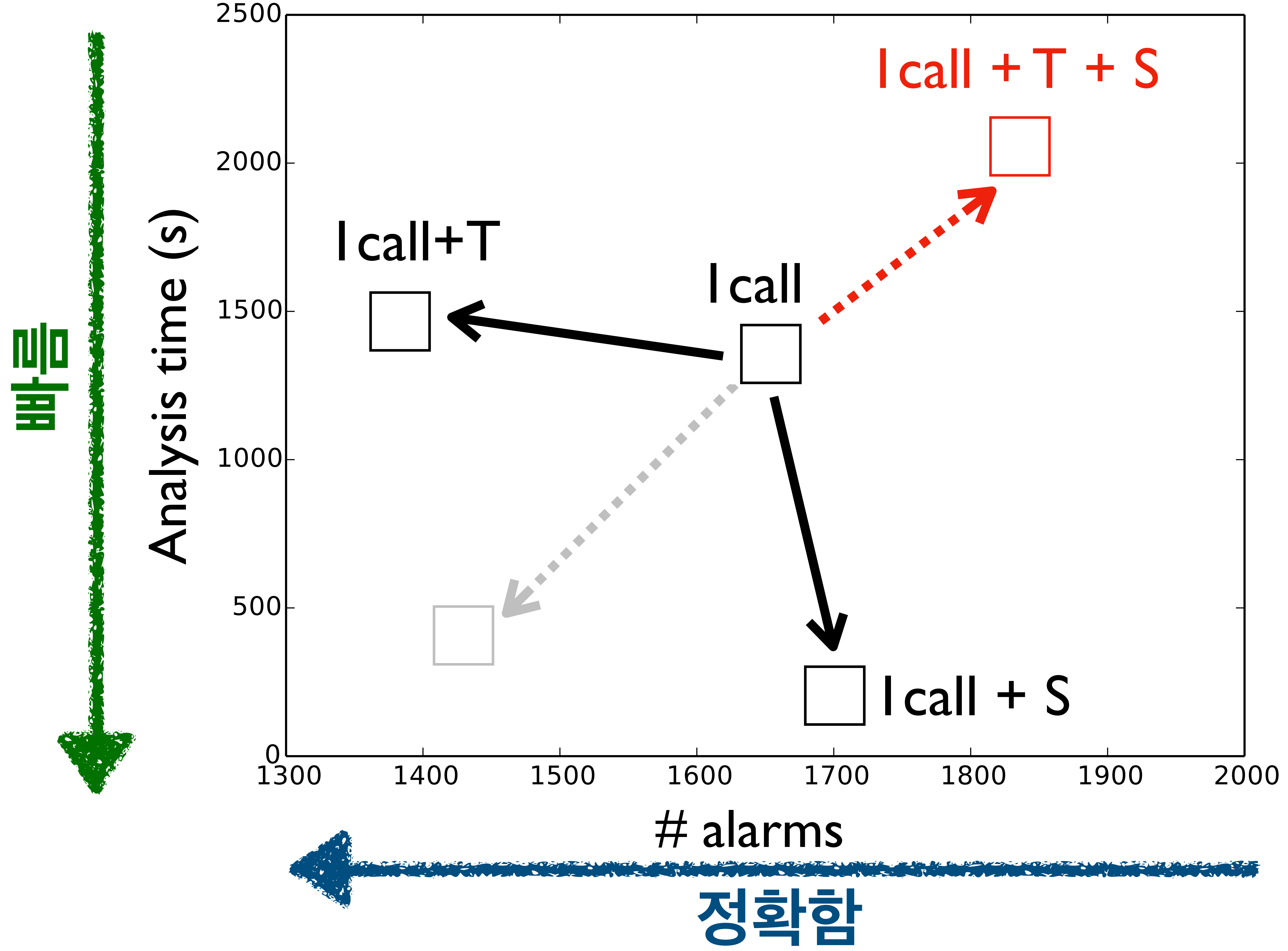
```
0:  id(v, i){
1:    if (i > 0){
2:      return id(v, i-1);}
3:    return v;}
4:
5:  main(){
6:    i = input();
7:    v1 = id(1, i);//A
8:    v2 = id(2, i);//B
9:    assert (v1 != v2);//query
10: }
```

예제 프로그램

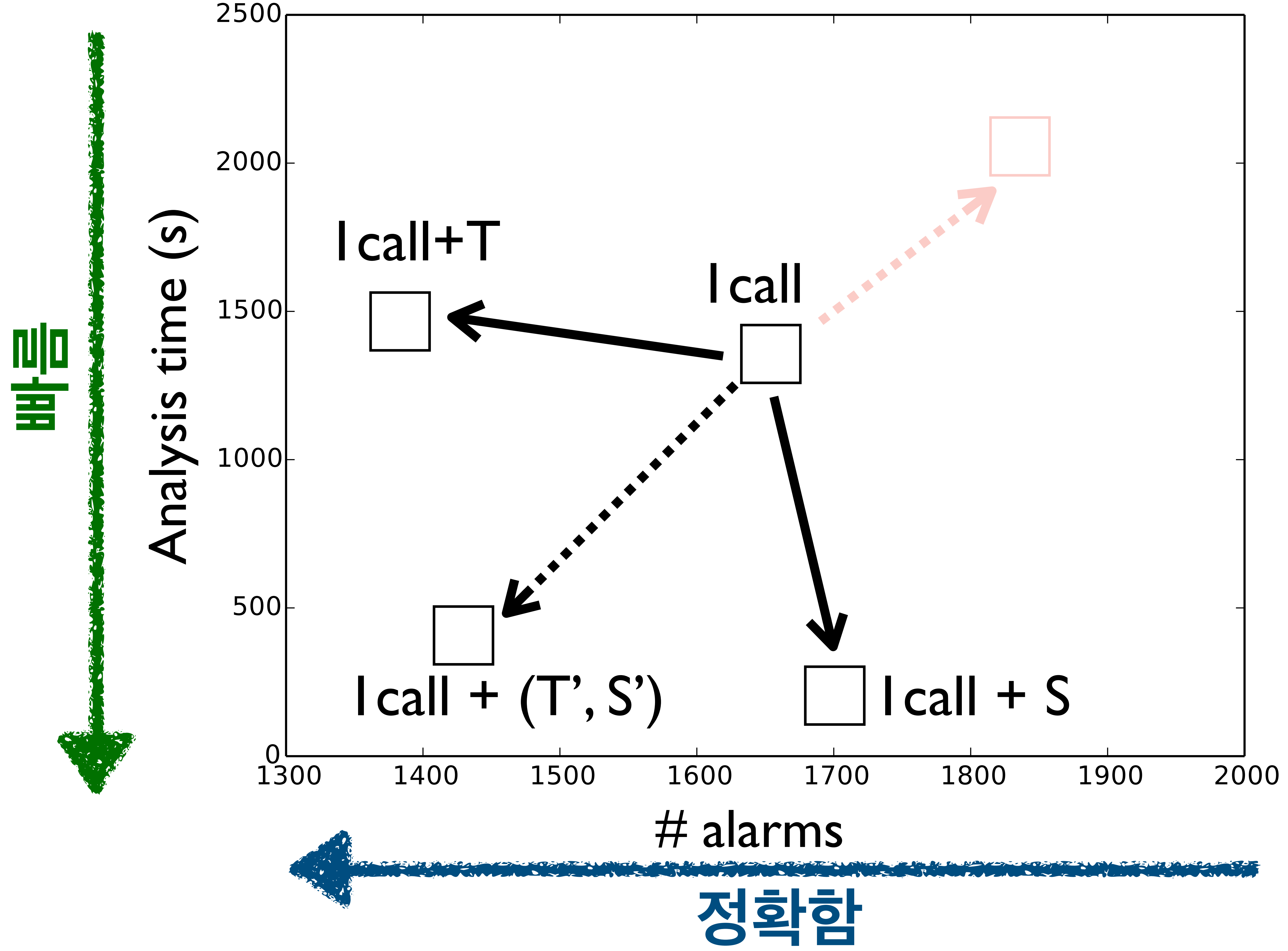


적당히 주요 1-요소 기반 함수 호출 요약

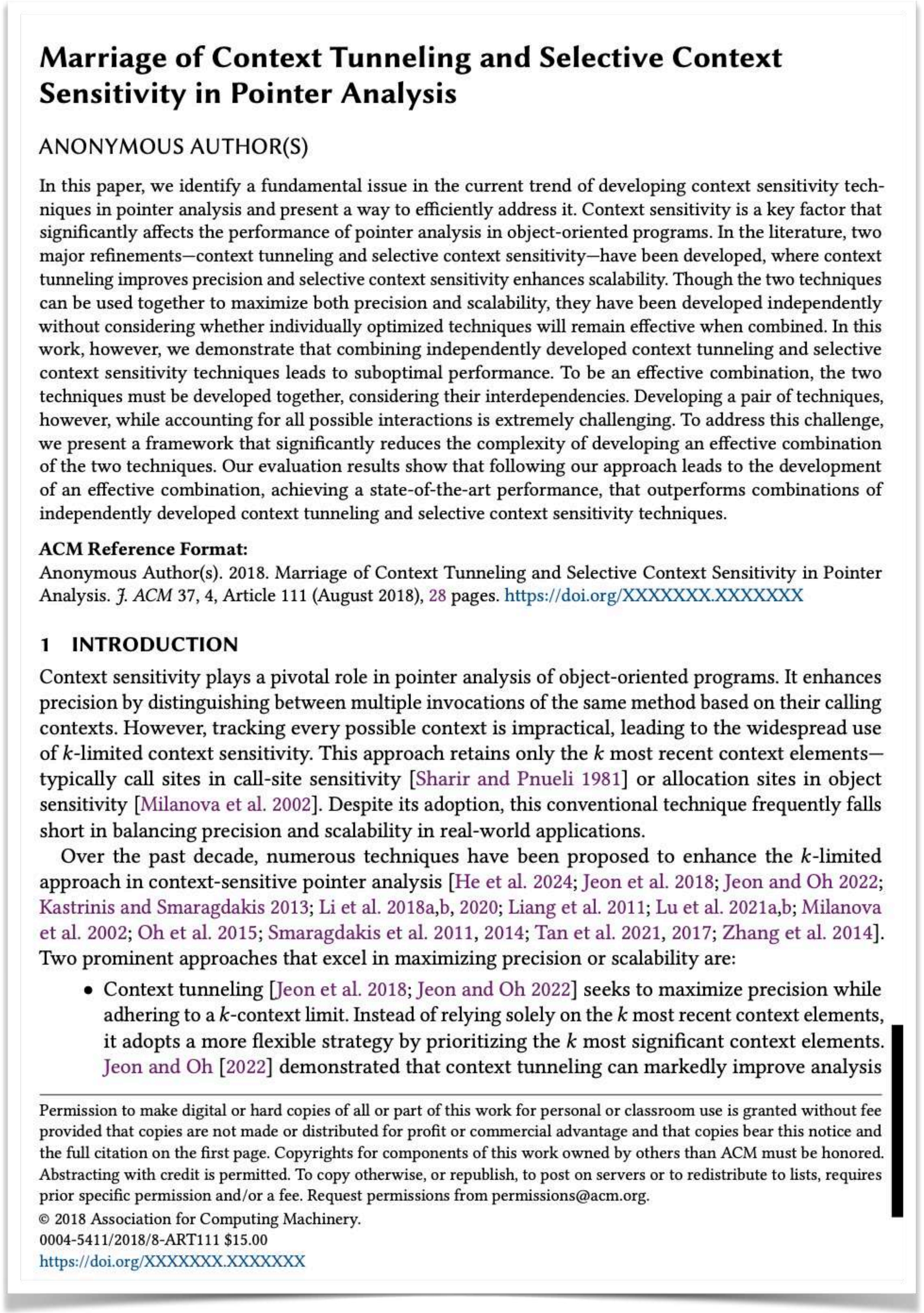
- 연구 트렌드를 따라 독립적으로 각각 개발한 기술들을 동시에 사용하면 성능이 오히려 나빠짐



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



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



Marriage of Context Tunneling and Selective Context Sensitivity in Pointer Analysis

ANONYMOUS AUTHOR(S)

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

ACM Reference Format:
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/XXXXXXX.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

0004-5411/2018/8-ART111 $15.00
https://doi.org/XXXXXXX.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

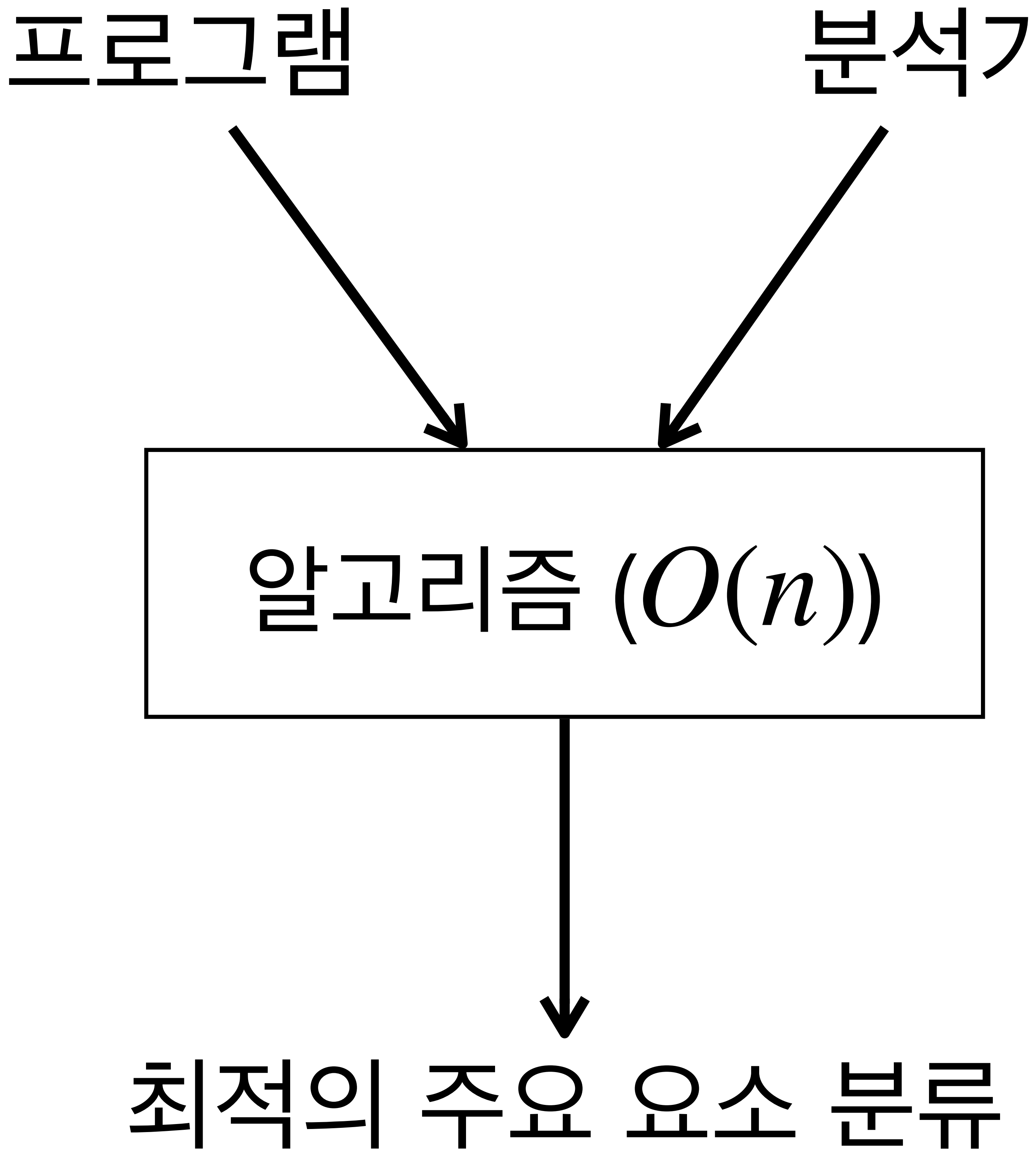
- 목표: 주요 k기반 요약 방식을 표준으로 만들기

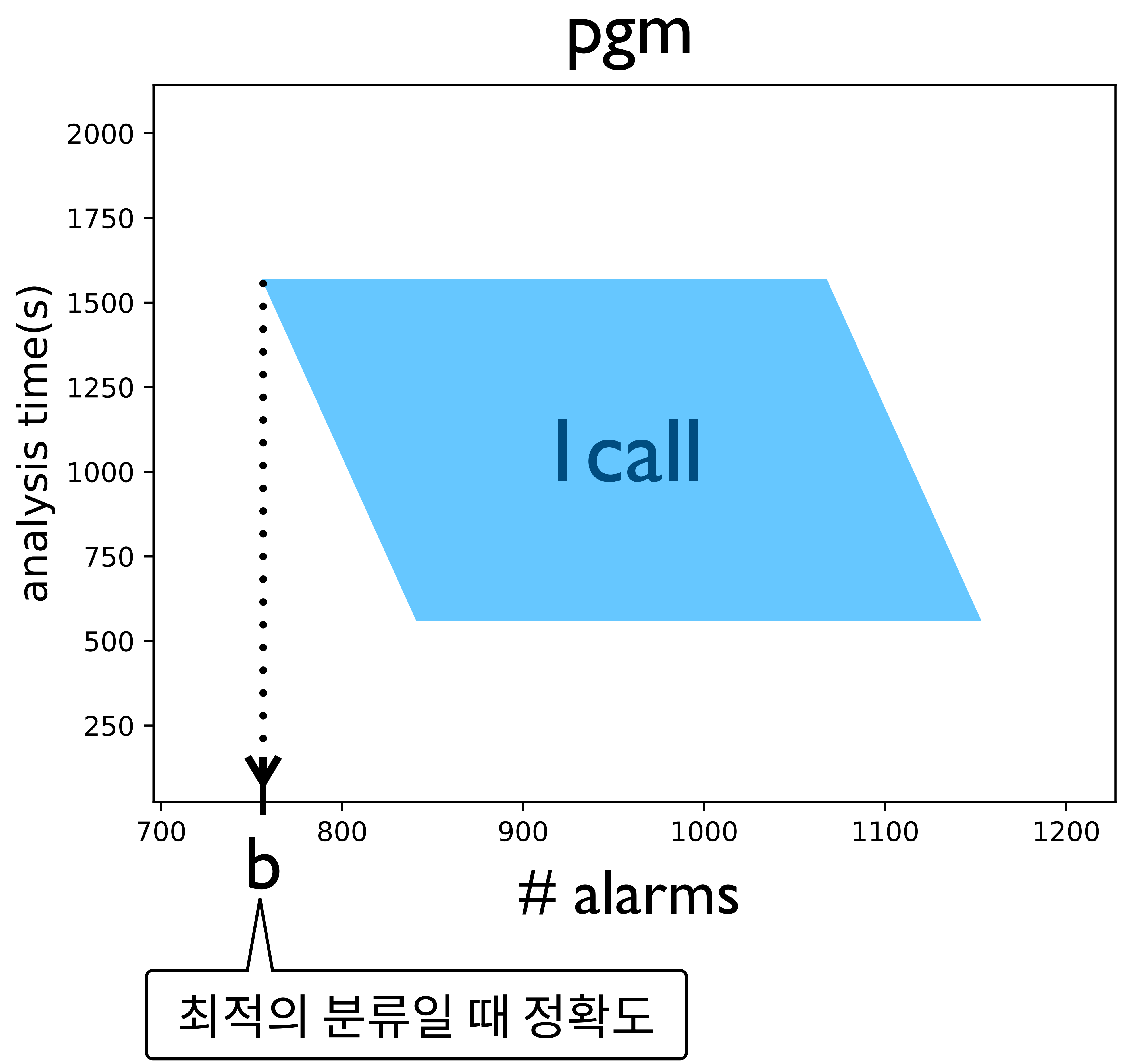
submitted

Tunneling → Obj2CFA → Combination → **Future work**

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

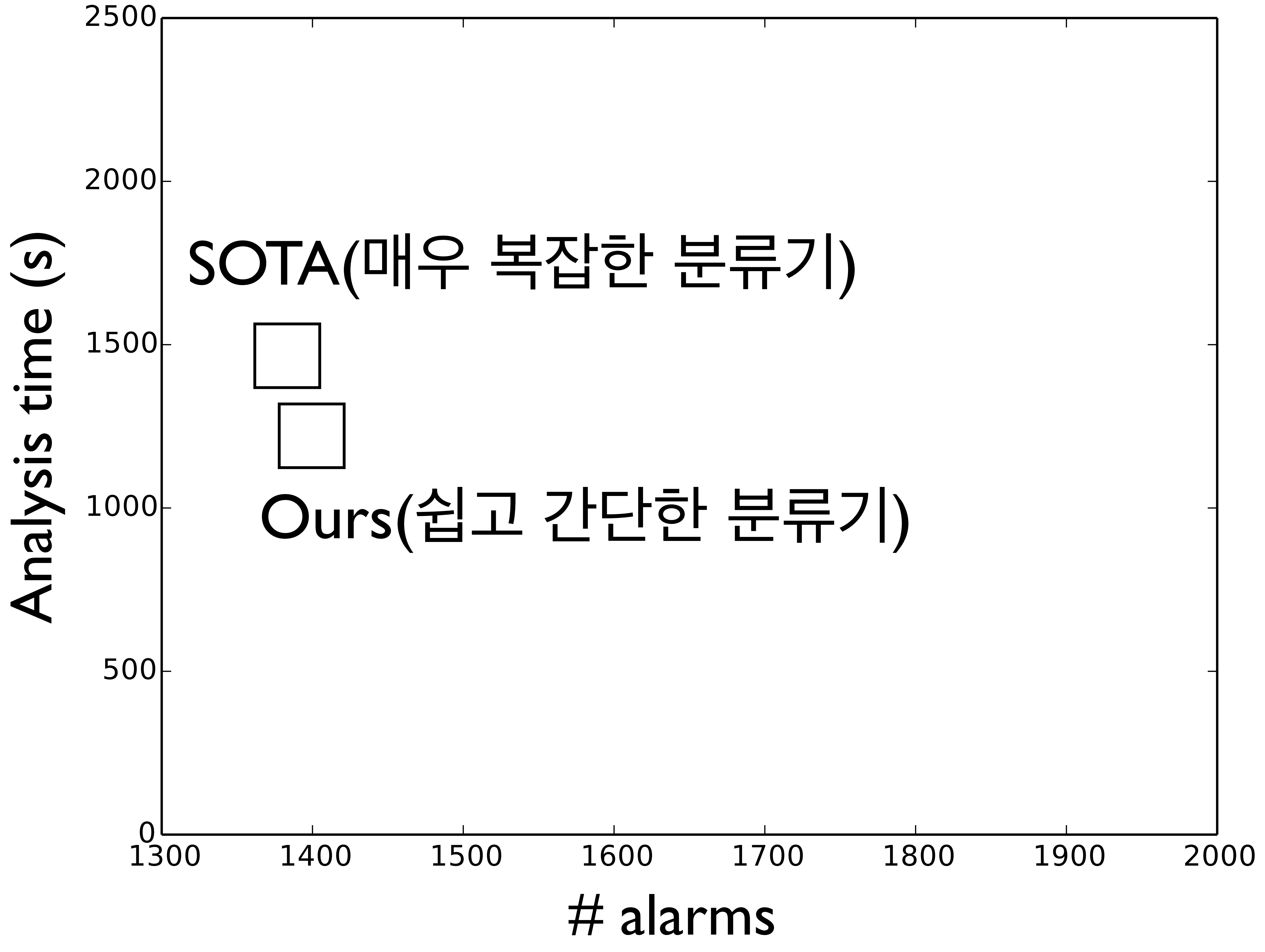
후속 연구 1: 최적의 주요 요소 분류 찾기 알고리즘





후속 연구 2: 쉽고 간단한 주요 요소 분류 방법

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



후속 연구 3: CFA의 Obj 대비 우수성을 이론적으로 보이기

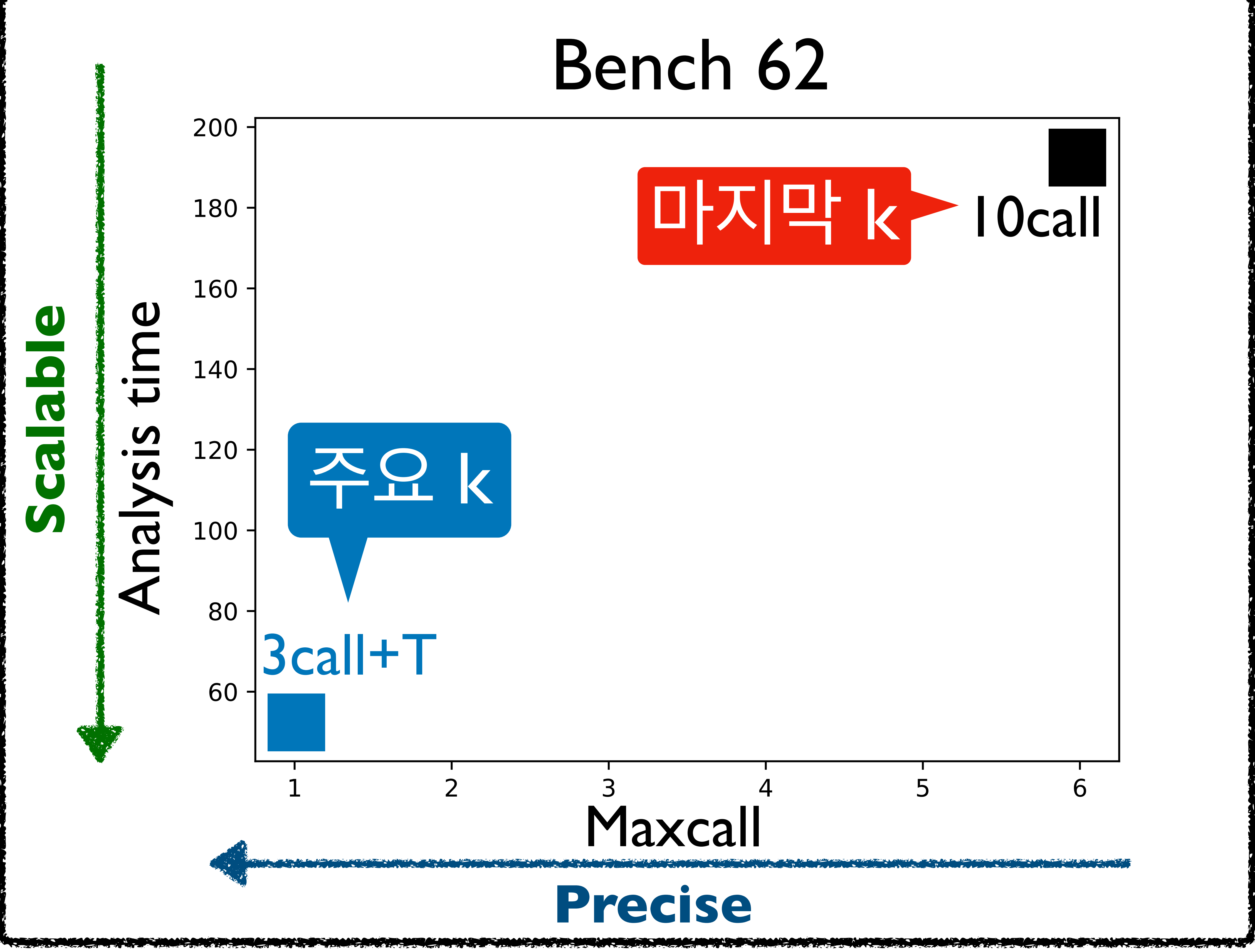
임의의 프로그램과 k-obj가 내놓은 분석 결과에 대하여, 더 정확한 분석 결과를 내놓는 k-CFA가 항상 존재하는가?

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

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

후속 연구 4: 타 언어에 적용하기

Java	JavaScript	Python	C	…
OOPSLA'18 **POPL' 22**	**ToDo**	**ToDo**	**ToDo**	



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:

1-call-site sensitivity로 정확하게 할 순 없나?

Tunneling → Obj2CFA → Combination → Future work

submitted

마지막 k ⟶ 주요 k

Obj ⟶ CFA

따로 개발 ⟶ 동시에 개발

Back up

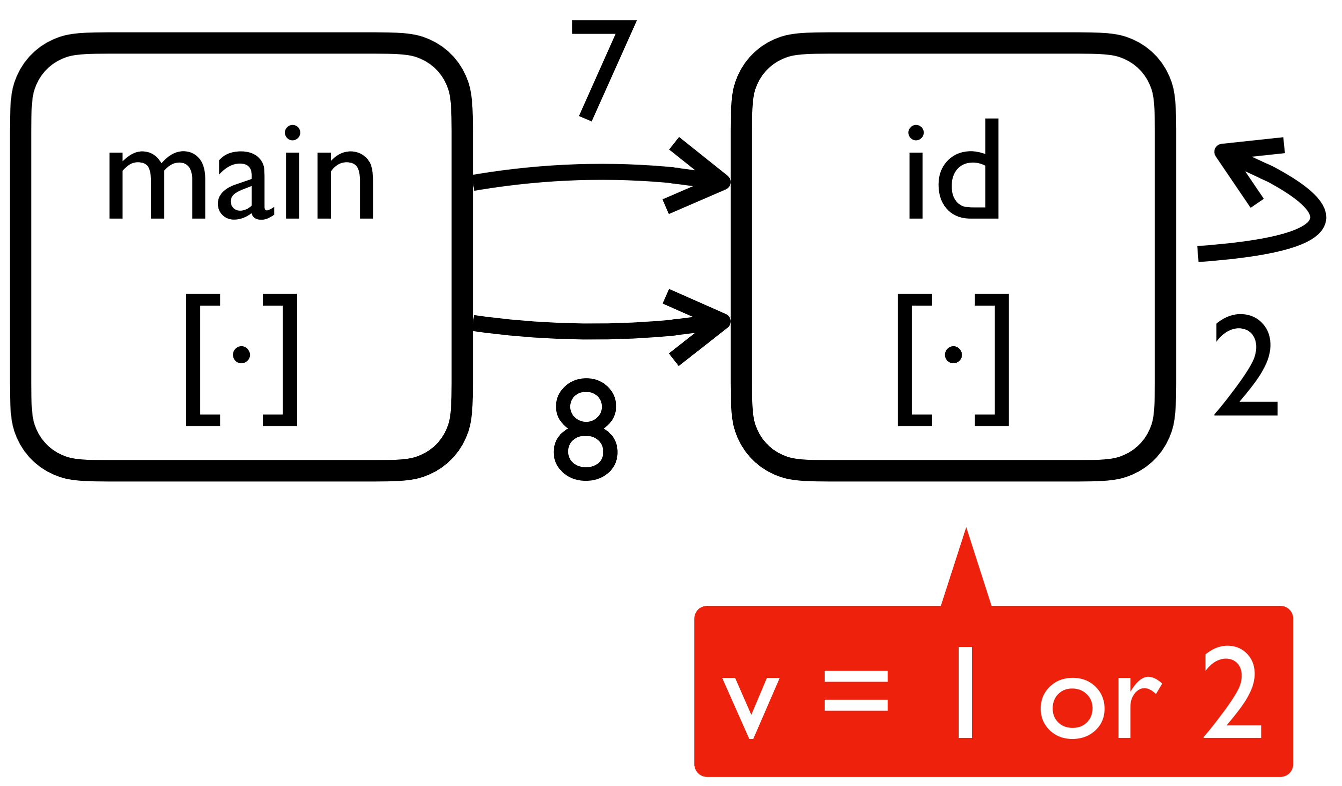
- 주요 1-요소 기반 함수 호출 요약

주요 요소 = {}

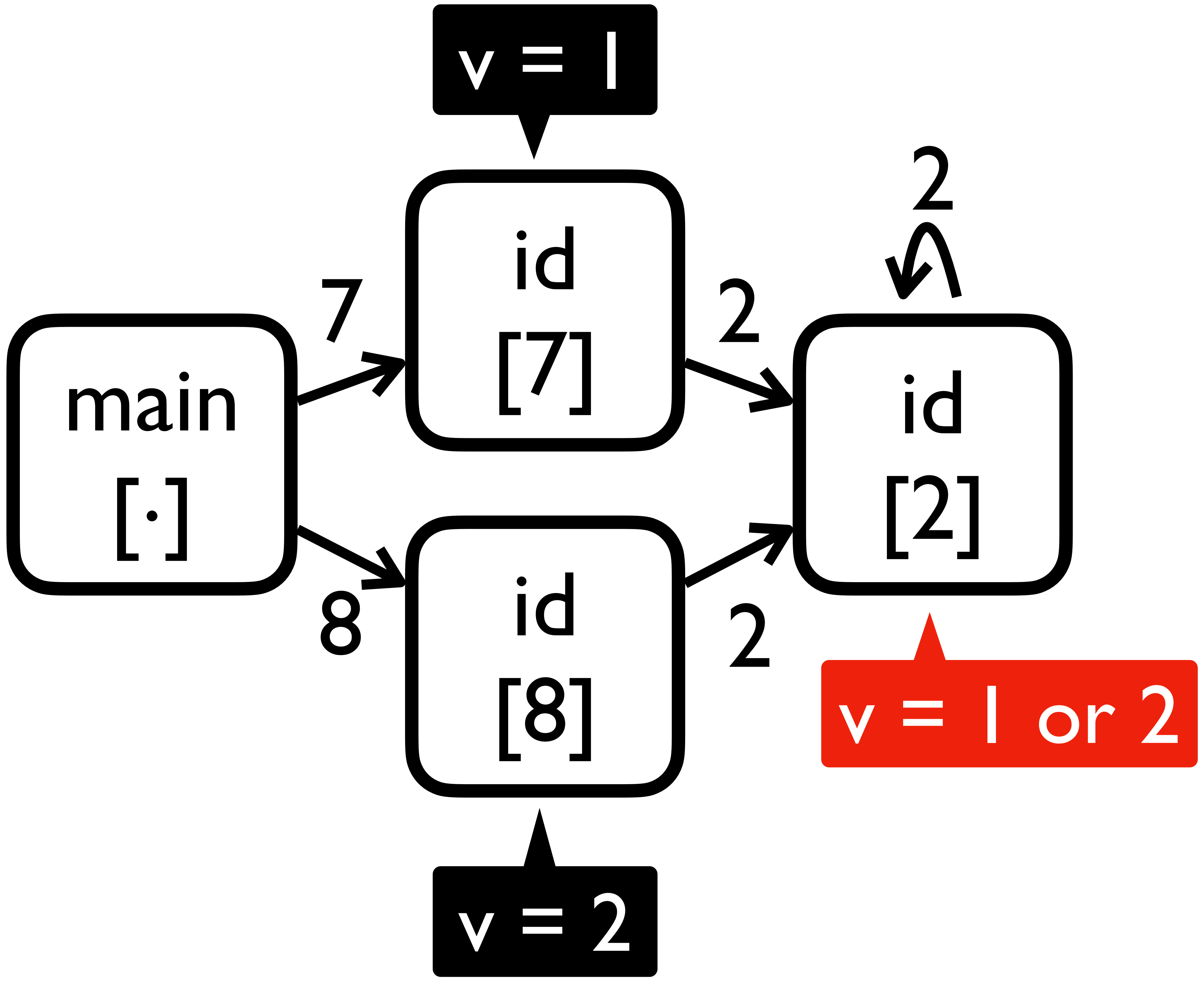
주요 요소 = {7, 8}

주요 요소 = {2, 7, 8}

- 선택적 1-요소 기반 함수 호출 요약



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

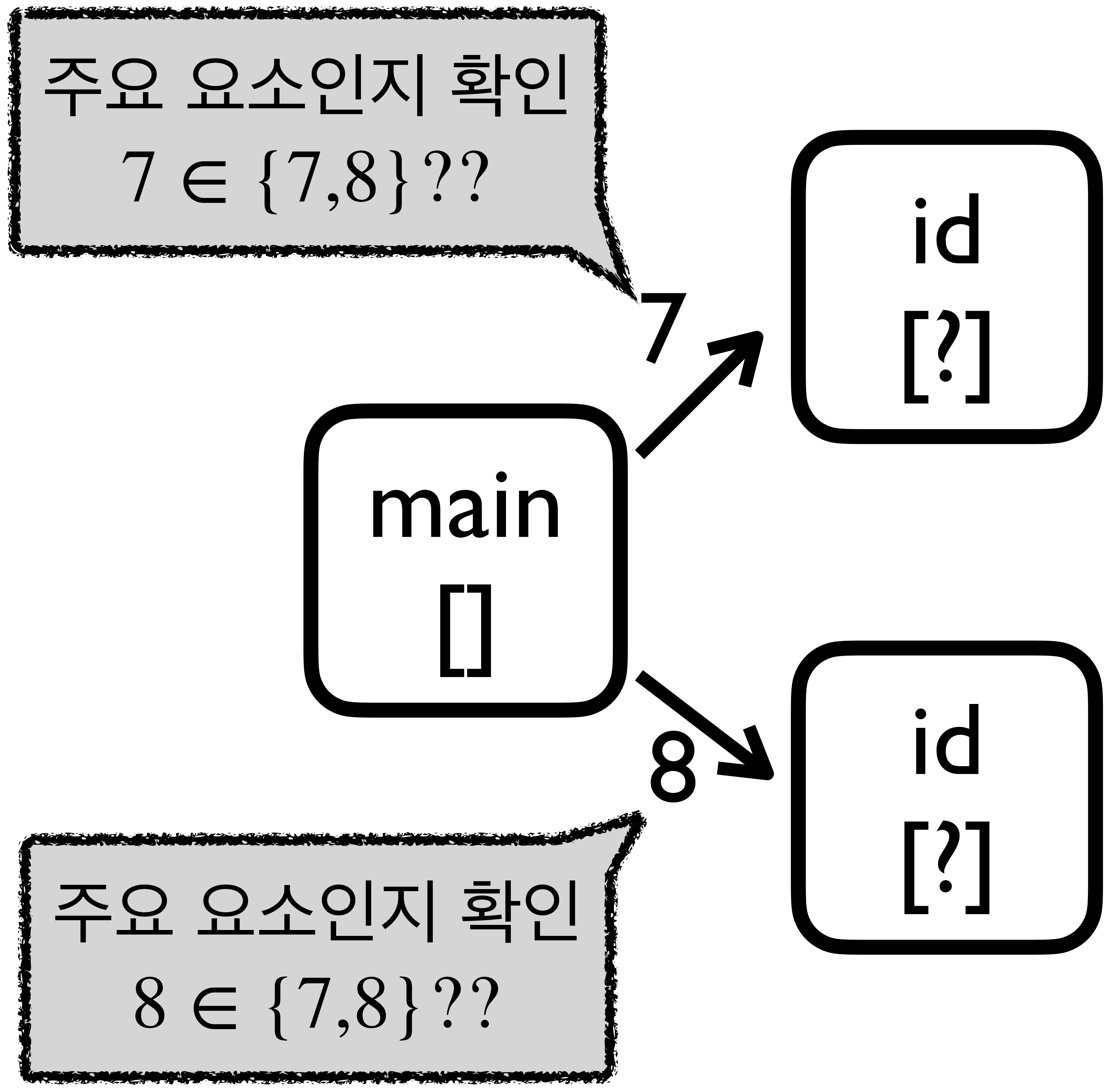


1-call : {id}
0-call : {main}

컨텍스트 터널링: 주요 K개 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: }
```

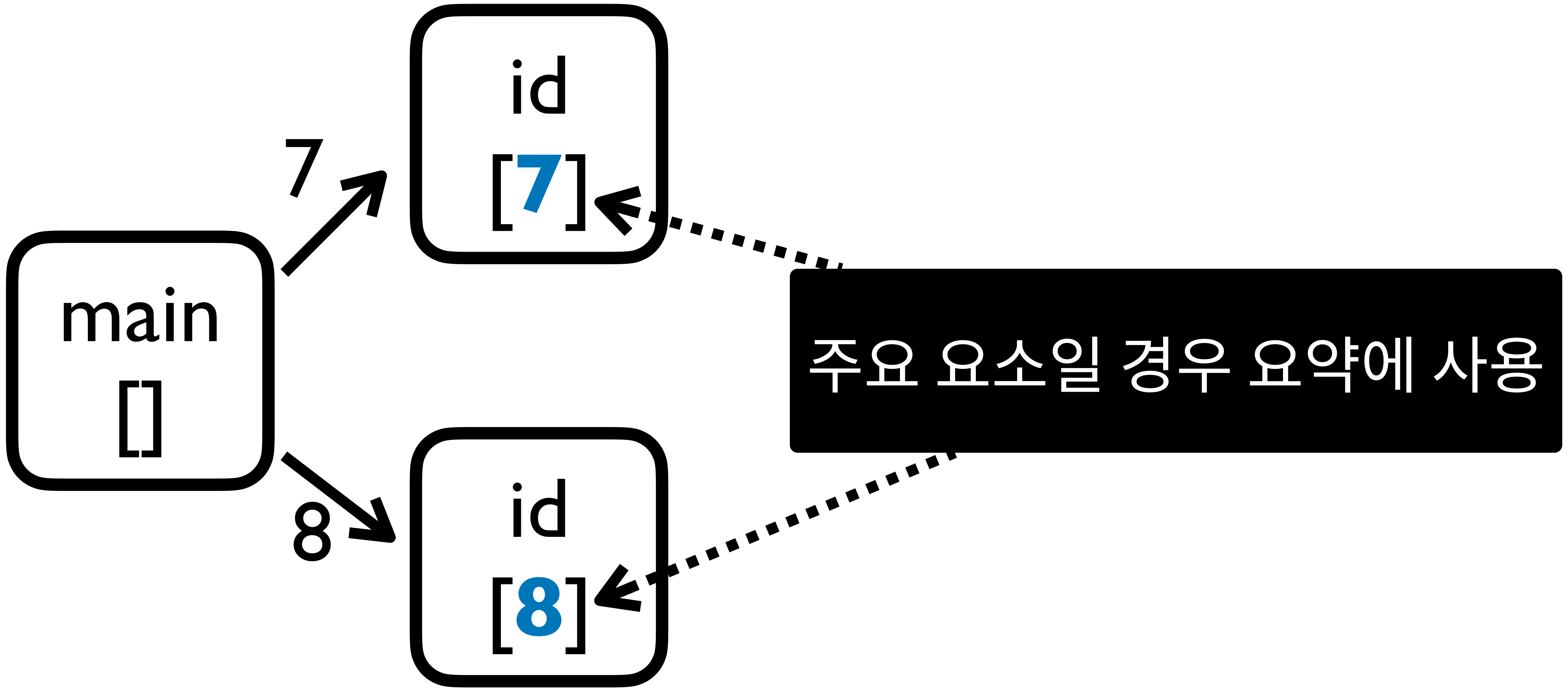


예제 프로그램

컨텍스트 터널링: 주요 K개 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: }
```



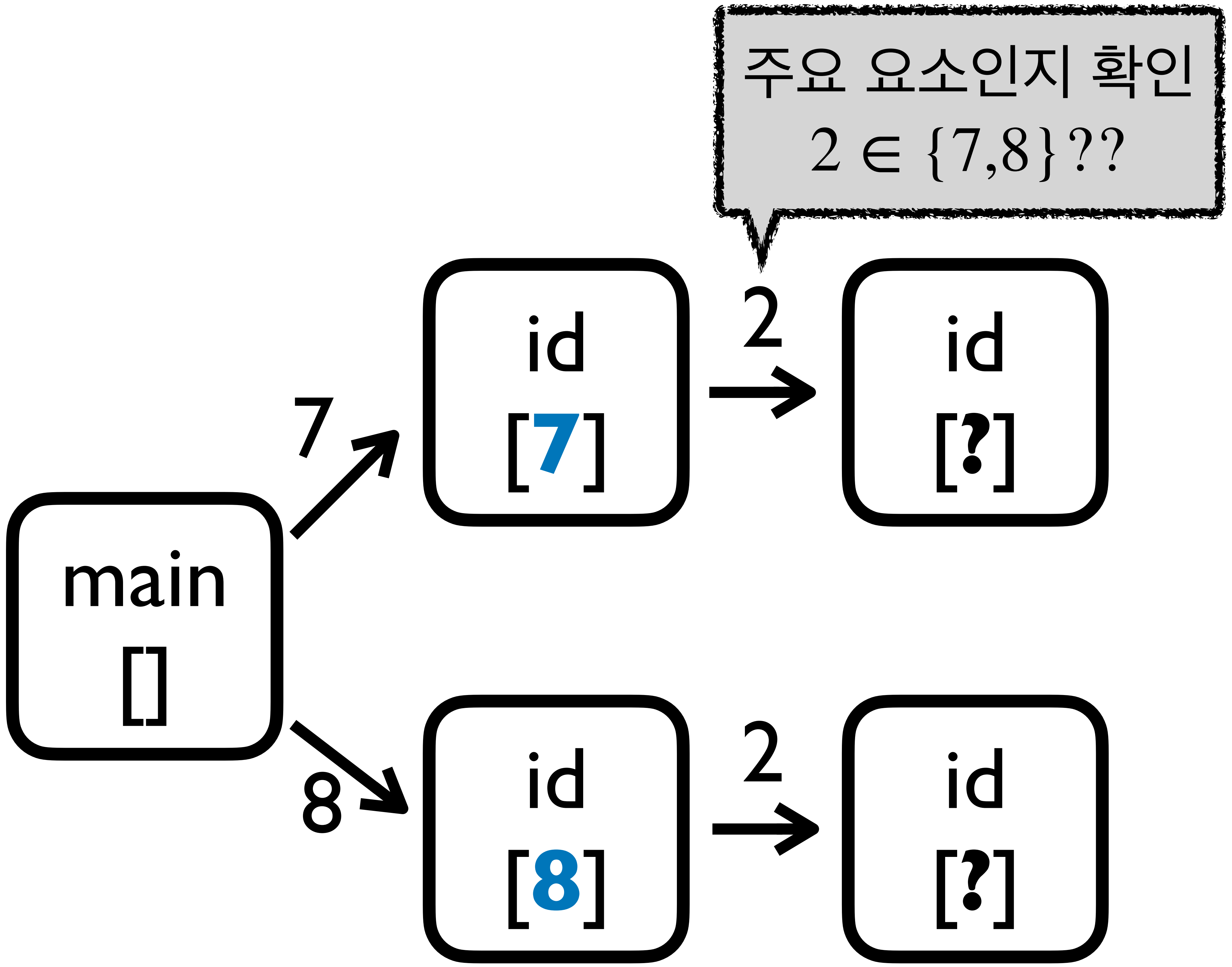
예제 프로그램

컨텍스트 터널링: 주요 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

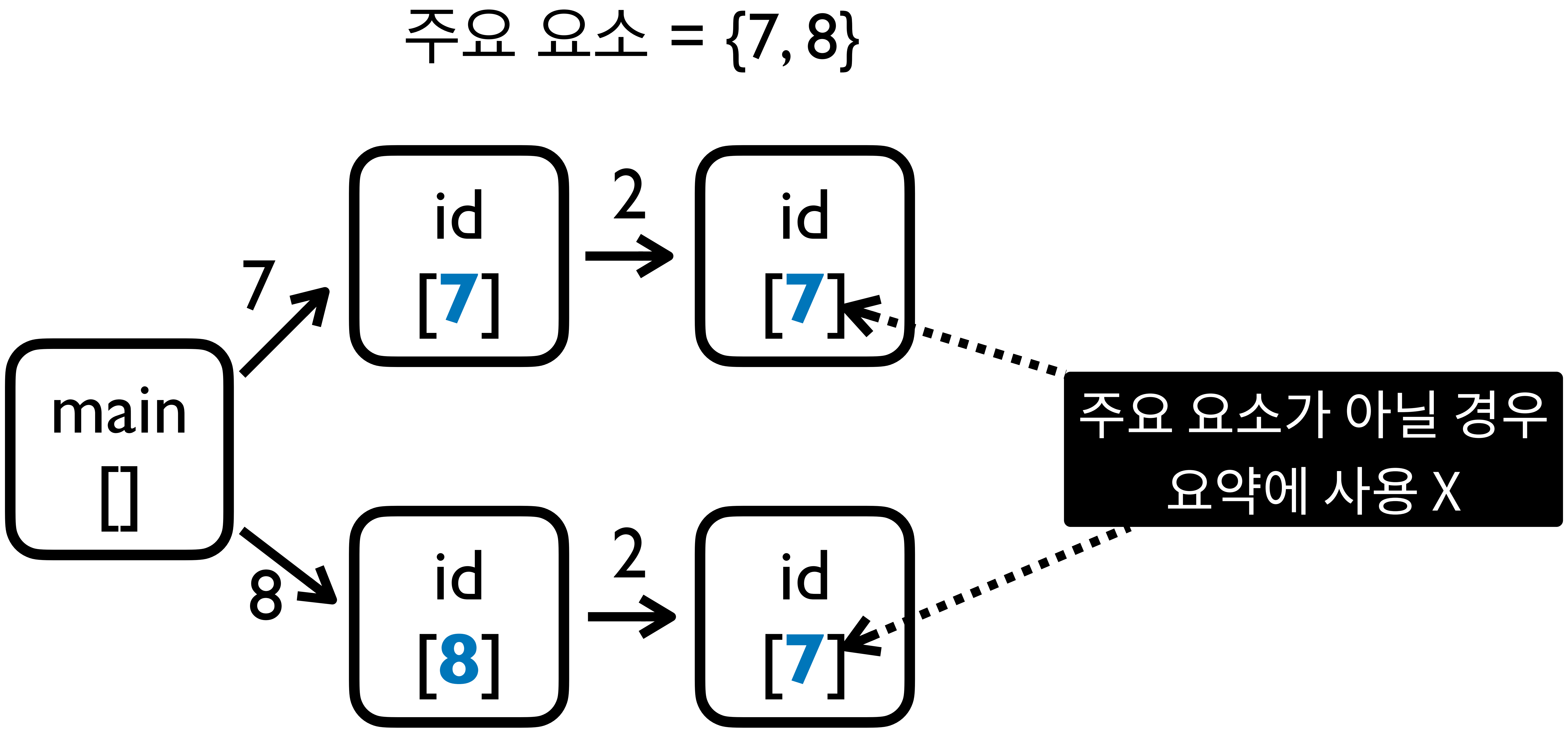
주요 요소 = {7, 8}



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: }
```

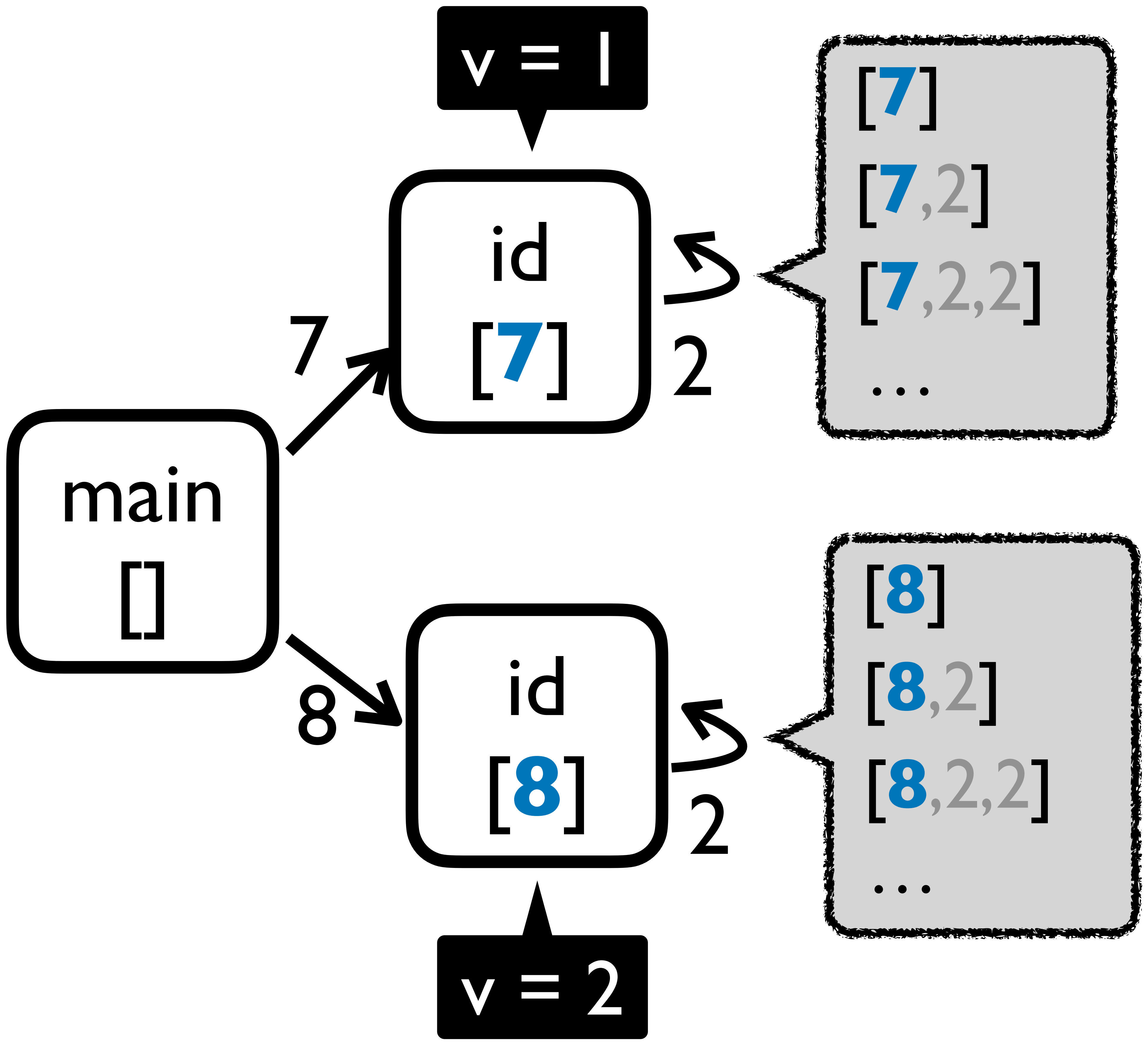
예제 프로그램



컨텍스트 터널링: 주요 K개 기반 요약

```
id(v, i){
  if (i > 0){
    return id(v, i-1);}
  return v;}

main(){
  i = input();
  v1 = id(1, i);
  v2 = id(2, i);
  assert (v1 != v2);//query
}
```

예제 프로그램



1개 주요 요소 기반 함수 호출 요약
(주요 요소 = {7, 8})