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

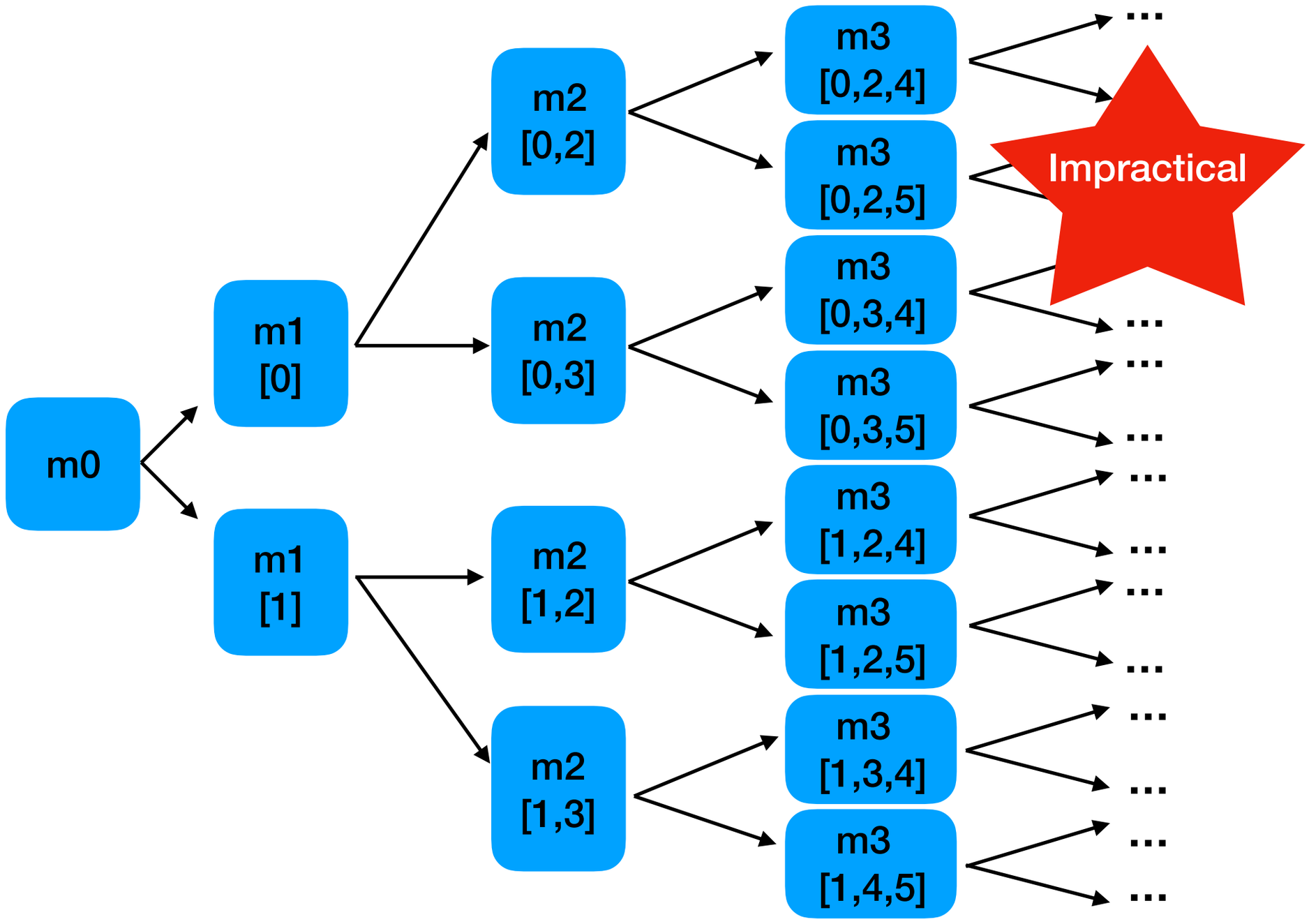
Minseok Jeon, Sehun Jeong, and Hakjoo Oh

Korea University

OOPSLA 2018 @Boston

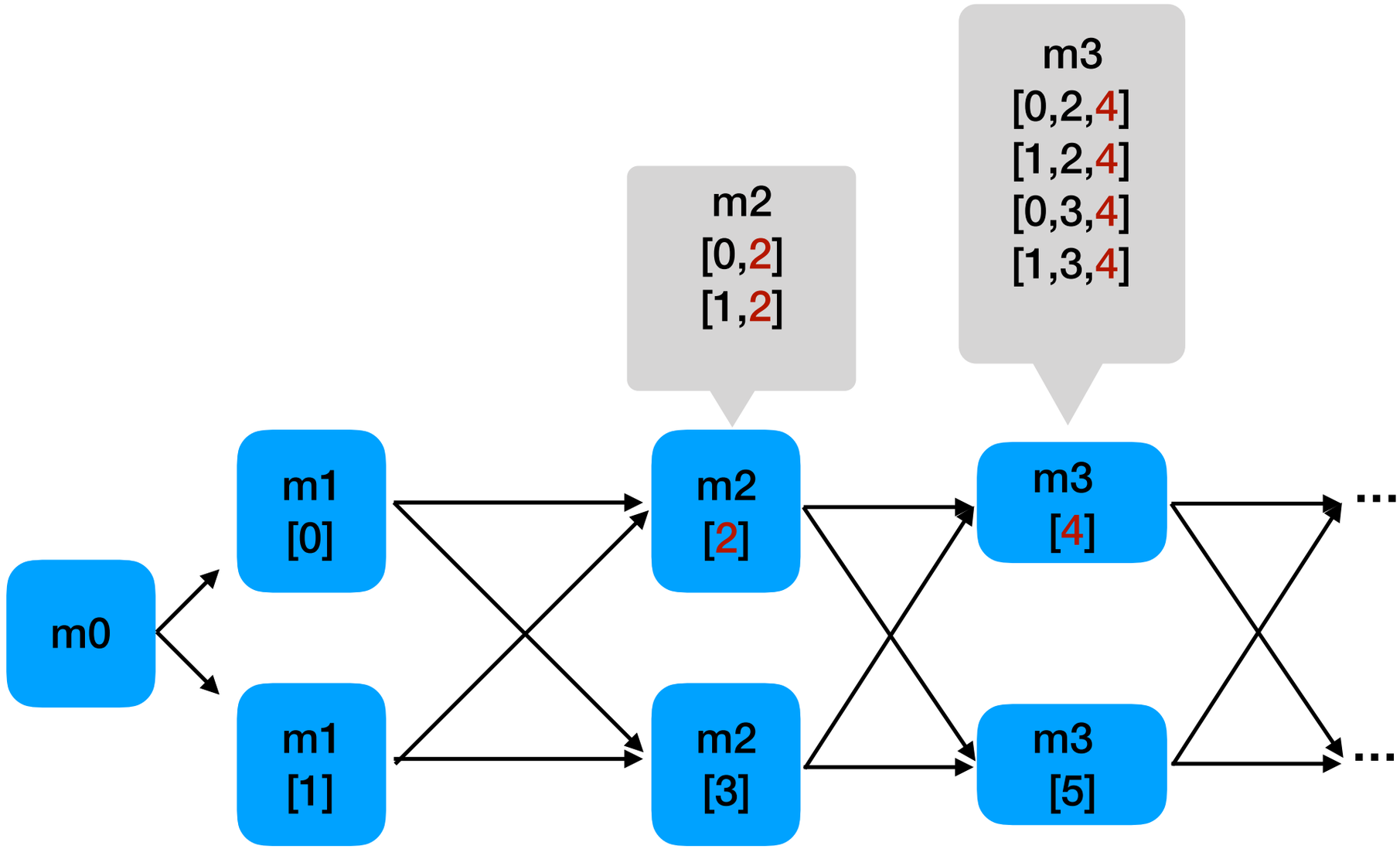
Static Analysis Needs Context Abstraction

```
m0( ) {
    m1( );//0
    m1( );//1
}

m1( ) {
    m2( );//2
    m2( );//3
}

m2( ) {
    m3( );//4
    m3( );//5
}
...
```



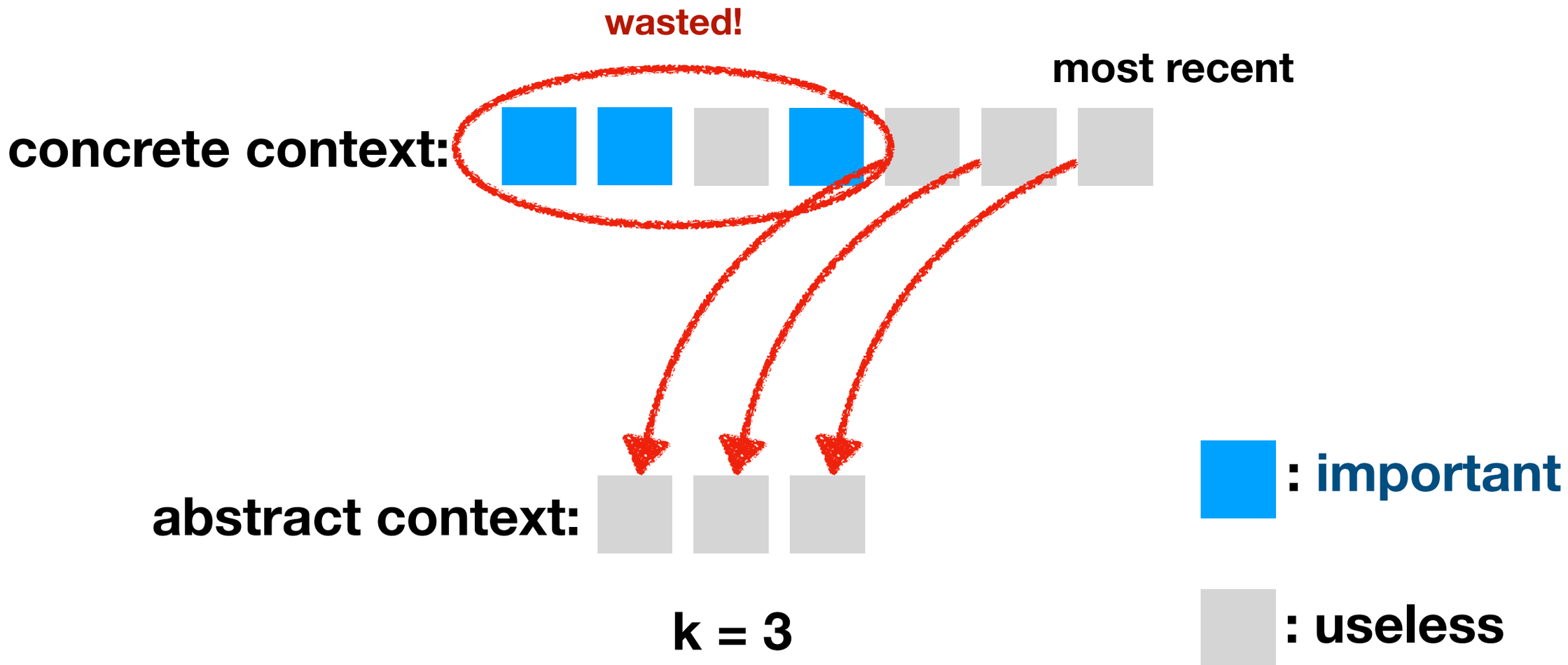
Conventional Abstraction: Keep Most Recent K Contexts

```
m0( ) {
    m1( );//0
    m1( );//1
}

m1( ) {
    m2( );//2
    m2( );//3
}

m2( ) {
    m3( );//4
    m3( );//5
}
...
```



Problem of Most Recent K

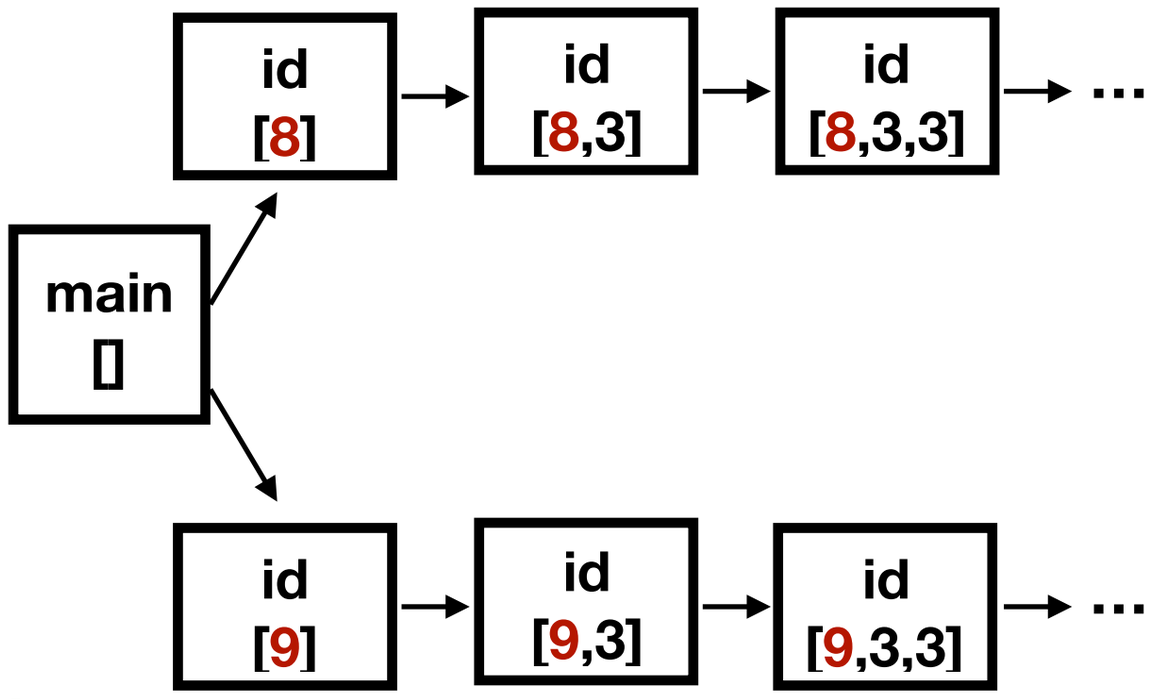
- **Most-recent-k** often abandons important context elements



Example

- **Context abstraction should keep important context 8 or 9**

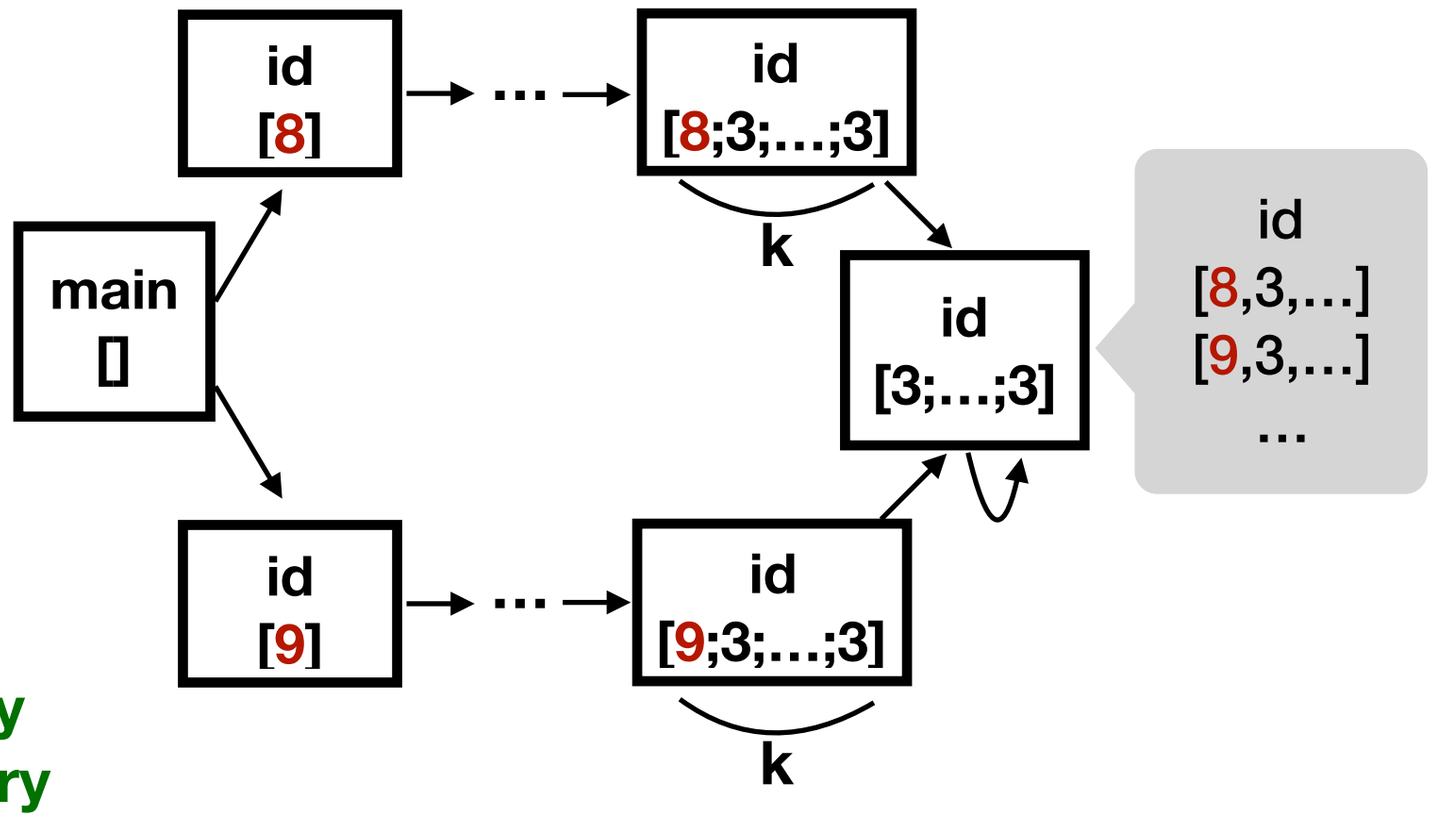
```
1: Object id(Object v, int i){
2:      if (i>0)
3:           return id(v, i-1);
4:      else
5:           return v;}

6: main(){
7:      int i = input();
8:      A a = (A) id (new A( ), i);  //query
9:      B b = (B) id (new B( ), i);}  //query
```



Most Recent K
Can’t Prove the Queries

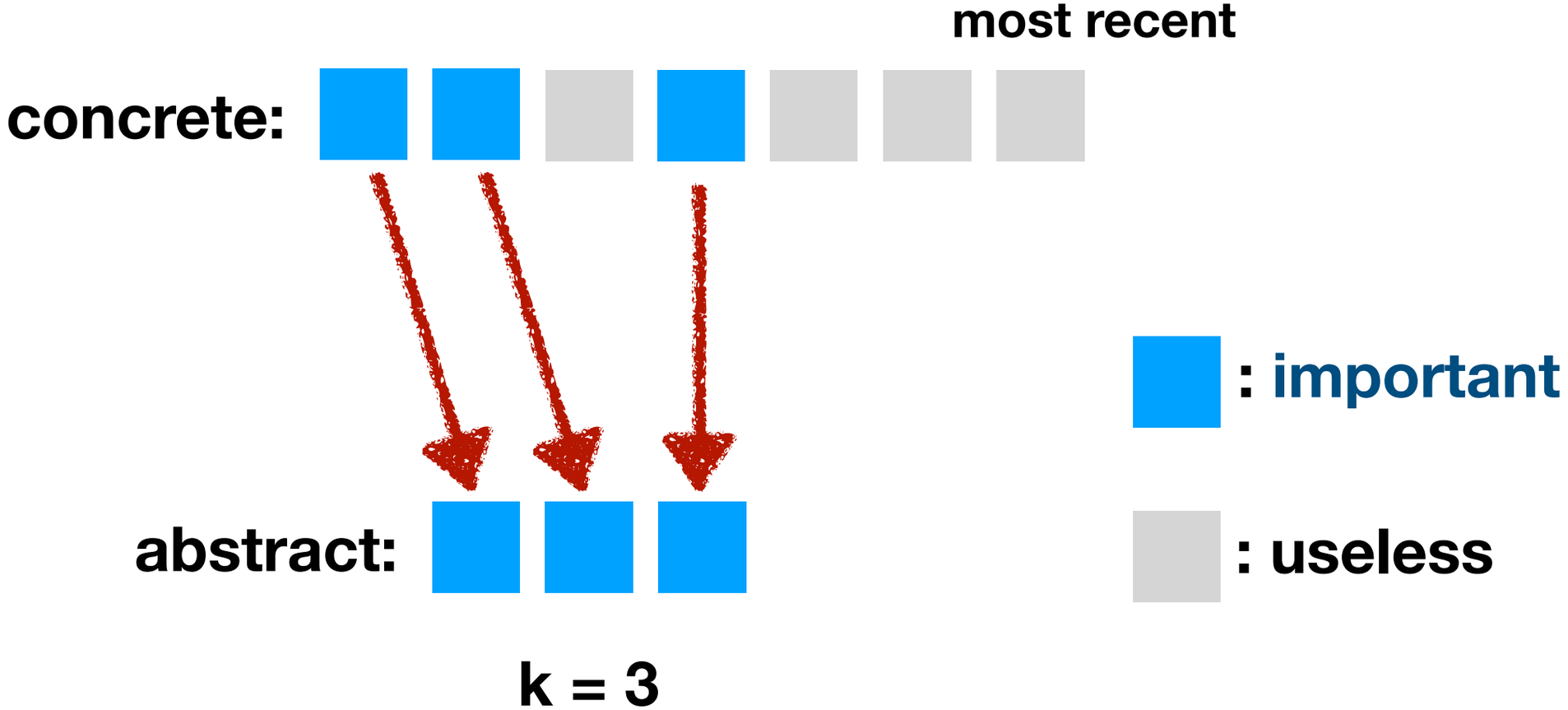
- **Most recent K cannot prove the queries for any K**

```
1: Object id(Object v, int i){
2:      if (i>0)
3:           return id(v, i-1);
4:      else
5:           return v;}

6: main(){
7:      int i = input();
8:      A a = (A) id (new A( ), i);  //query
9:      B b = (B) id (new B( ), i);}  //query
```



Context Tunneling: Keep Most important K

- **Do not keep most recent K**
- **Instead, keep most important K**

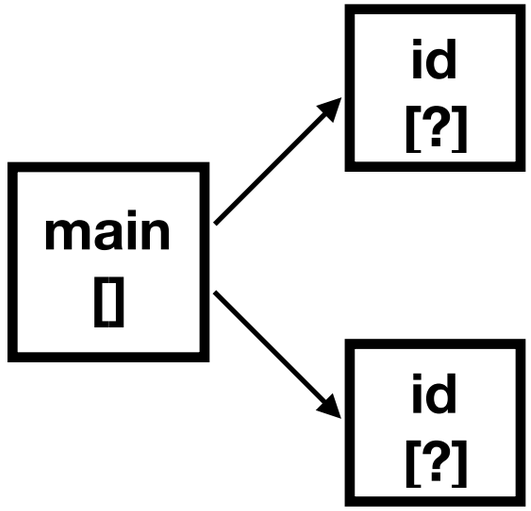


1-CFA with Context Tunneling

- **(main, id): generates important context elements**

main calls id

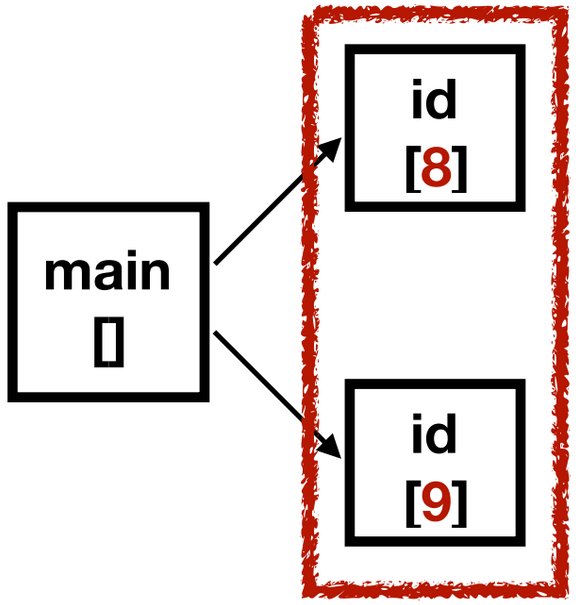
```
1: Object id(Object v, int i){
2:      if (i>0)
3:           return id(v, i-1);
4:      else
5:           return v;}

6: main(){
7:      int i = input();
8:      A a = (A) id (new A( ), i);  //query
9:      B b = (B) id (new B( ), i);} //query
```



1-CFA with Context Tunneling

- **(main, id): generates important context elements**
- **update context!**

```
1: Object id(Object v, int i){
2:      if (i>0)
3:           return id(v, i-1);
4:      else
5:           return v;}

6: main(){
7:      int i = input();
8:      A a = (A) id (new A( ), i);  //query
9:      B b = (B) id (new B( ), i);} //query
```

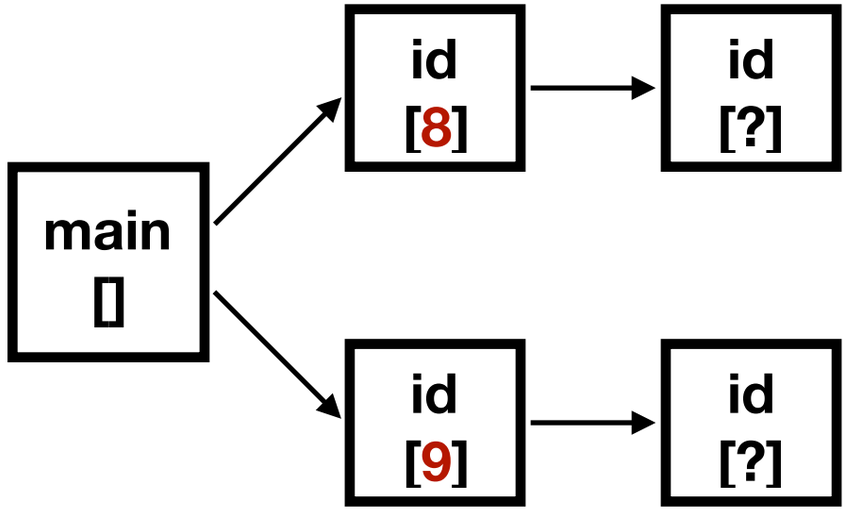


1-CFA with Context Tunneling

- **(id, id): generates useless context element**

id calls id

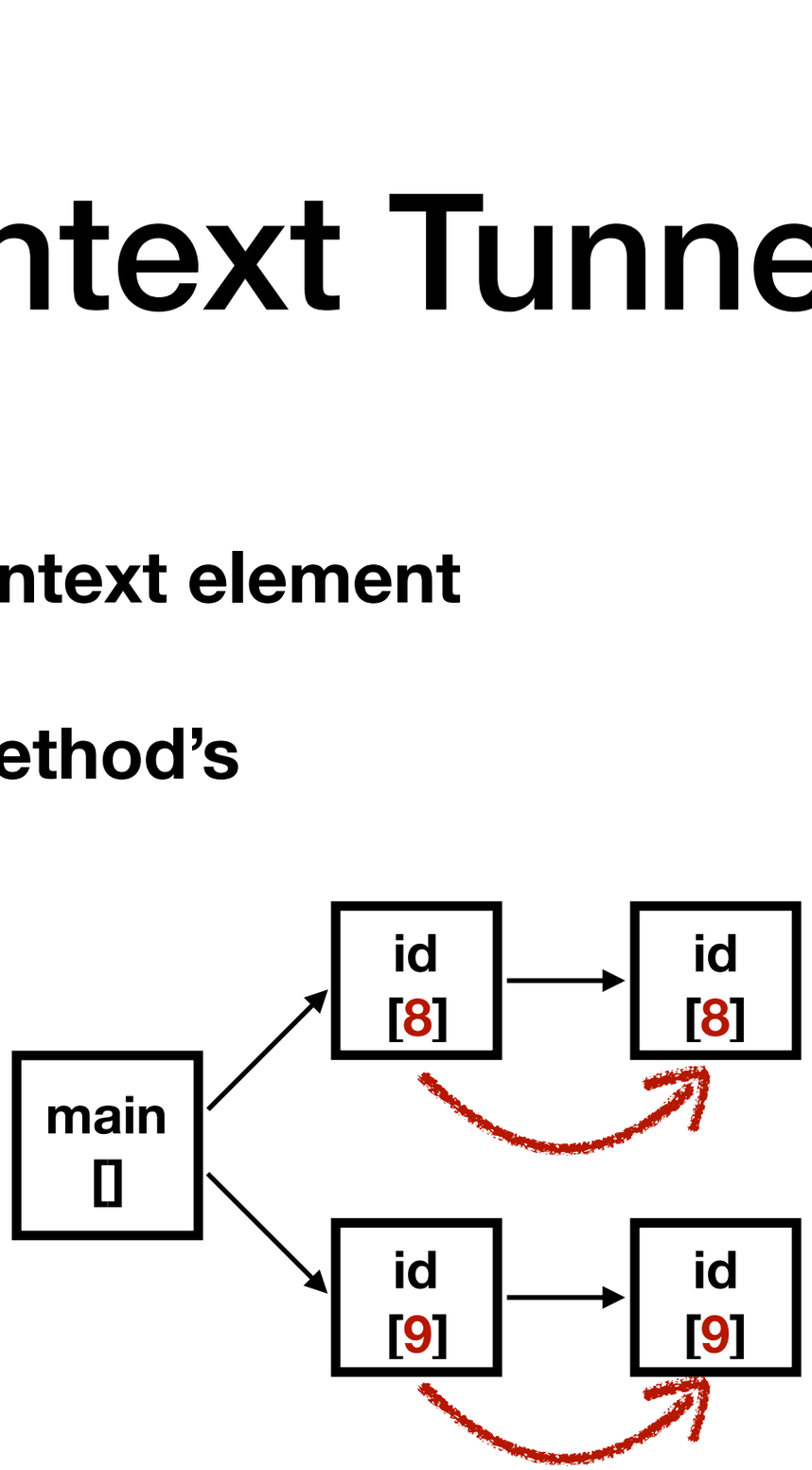
```
1: Object id(Object v, int i){
2:      if (i>0)
3:          return id(v, i-1);
4:      else
5:          return v;}

6: main(){
7:      int i = input();
8:      A a = (A) id (new A( ), i);  //query
9:      B b = (B) id (new B( ), i);}  //query
```



1-CFA with Context Tunneling

- **(id, id): generates useless context element**
- **Inherit context from caller method's**

```
1: Object id(Object v, int i){
2:     if (i>0)
3:         return id(v, i-1);
4:     else
5:         return v;}

6: main(){
7:     int i = input();
8:     A a = (A) id (new A( ), i);  //query
9:     B b = (B) id (new B( ), i);}  //query
```

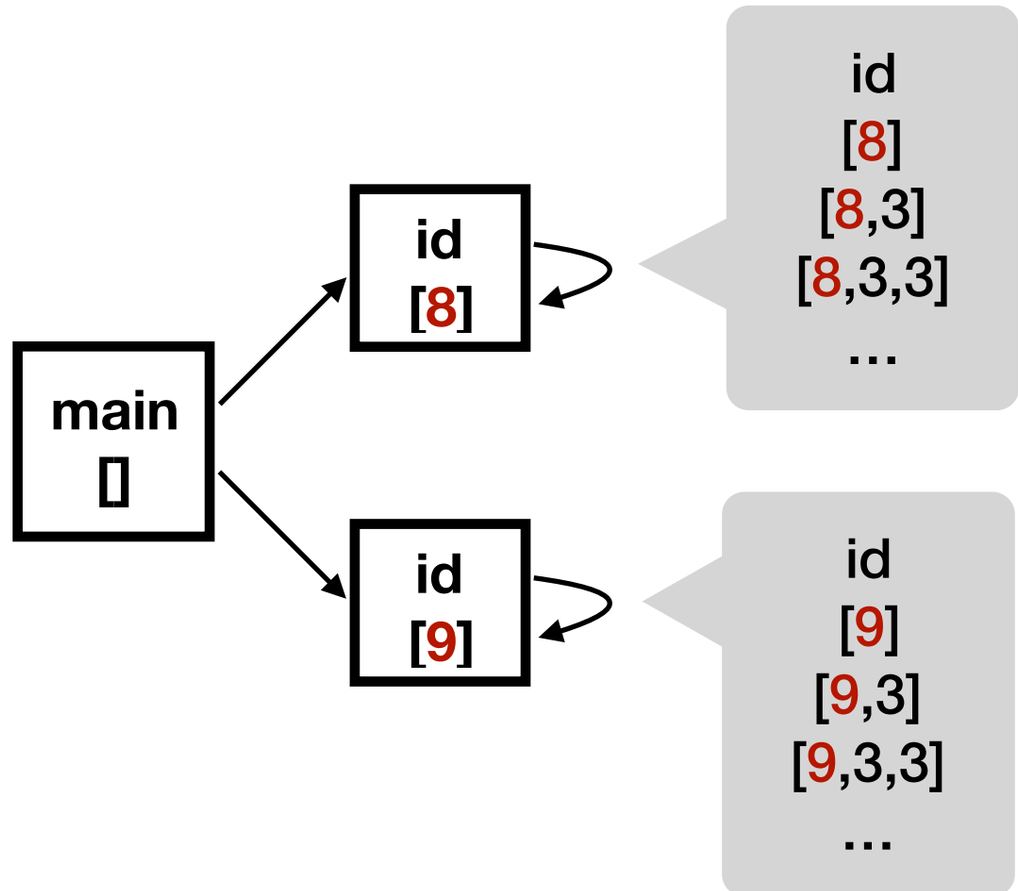


apply tunneling

1-CFA with Context Tunneling

- **Analysis ends**
- **Analyzer proves the queries!**

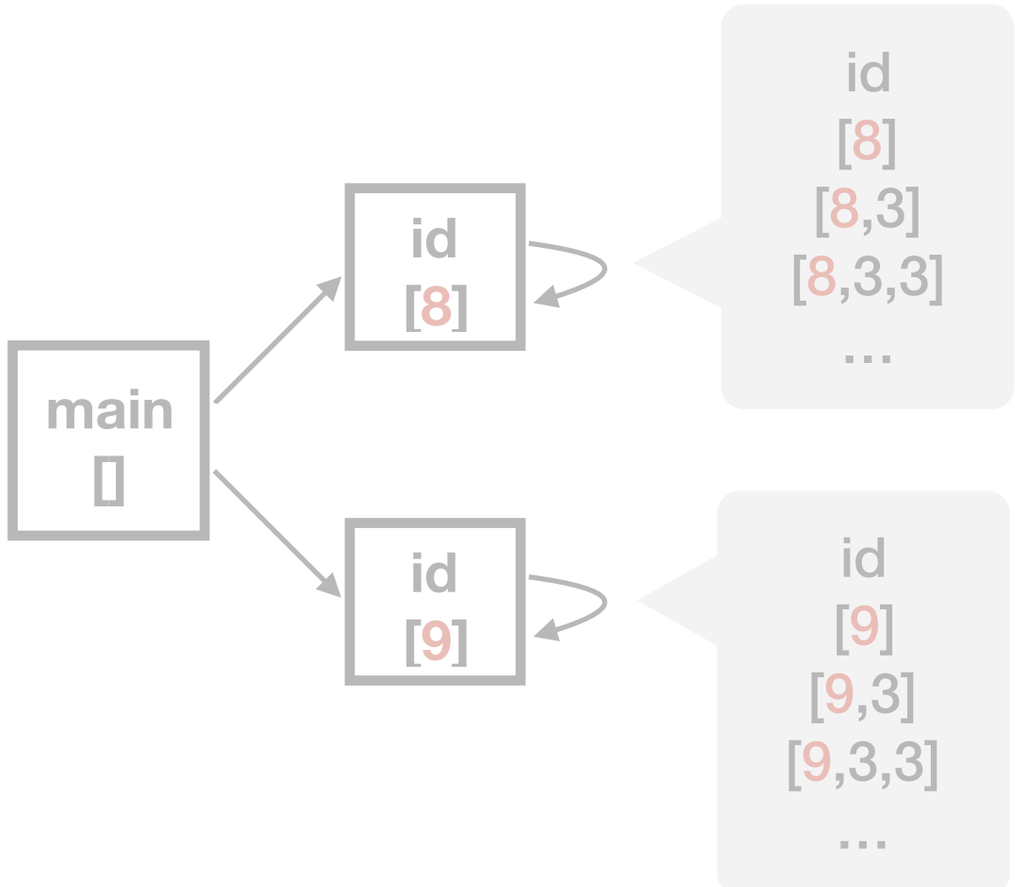
```
1: Object id(Object v, int i){
2:      if (i>0)
3:           return id(v, i-1);
4:      else
5:           return v;}

6: main(){
7:      int i = input();
8:      A a = (A) id (new A( ), i);  //query
9:      B b = (B) id (new B( ), i);}  //query
```



Challenge: When to apply Context Tunneling?

- **{(id, id)}: apply tunneling!**

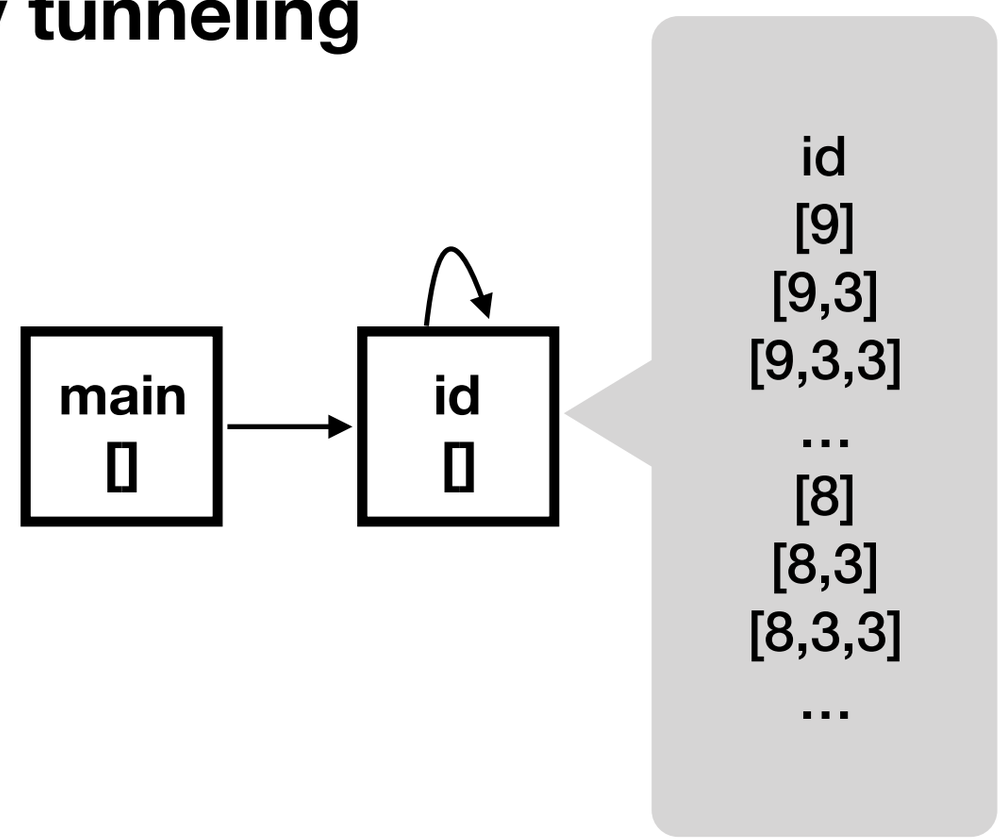
```
1: Object id(Object v, int i){
2:     if (i>0)
3:         return id(v, i-1);
4:     else
5:         return v;}

6: main(){
7:     int i = input();
8:     A a = (A) id (new A( ), i);  //query
9:     B b = (B) id (new B( ), i);}  //query
```



Challenge: When to apply Context Tunneling?

- **Incorrectly applying tunneling results in imprecision**
- **e.g) {(main, id), (id, id)}: apply tunneling**

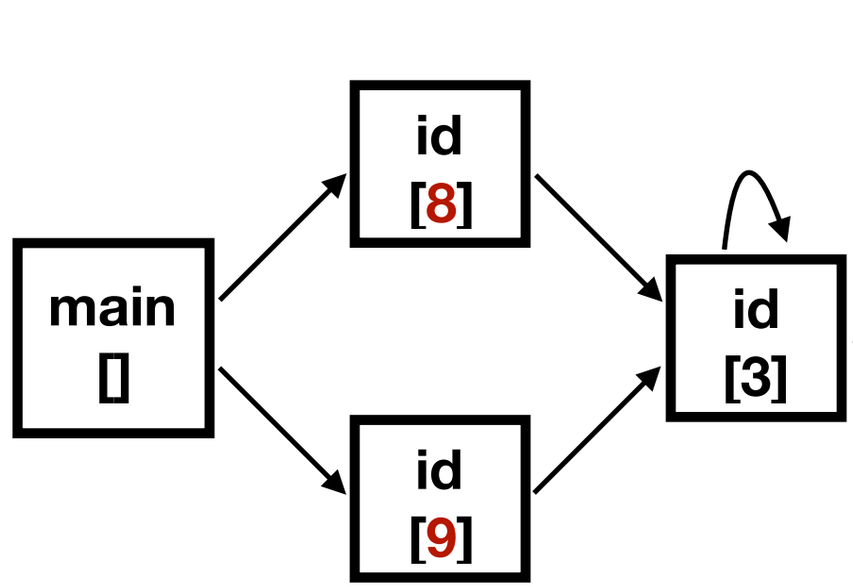
```
1: Object id(Object v, int i){
2:      if (i>0)
3:           return id(v, i-1);
4:      else
5:           return v;}

6: main(){
7:      int i = input();
8:      A a = (A) id (new A( ), i);  //query
9:      B b = (B) id (new B( ), i);}  //query
```



Challenge: When to apply Context Tunneling?

- **Missing tunneling opportunity also results in imprecision**
- **e.g) {}: apply tunneling**

```
1: Object id(Object v, int i){
2:      if (i>0)
3:           return id(v, i-1);
4:      else
5:           return v;}

6: main(){
7:      int i = input();
8:      A a = (A) id (new A( ), i);  //query
9:      B b = (B) id (new B( ), i);}  //query
```



Challenge: When to apply Context Tunneling?

- **Tunneling abstraction is a set of relations between methods**
- **Manually finding tunneling abstraction is difficult**
  - $2^{Method \times Method}$ **cases to consider**
  - **Only a few tunneling abstractions actually work**

{ ~~{}~~, ~~{(main,id)}~~, {(id,id)}, ~~{(main,id), (id,id)}~~ }

Challenge: When to apply Context Tunneling?

- Tunneling abstraction is a set of relation between methods
- Manually fi tion is difficult

Our Solution: Data-Driven Approach

- $2^{Method \times Method}$ cases to consider
- Only a few tunneling abstraction works

{ ∅, {(main,id)}, {(id,id)}, {(main,id), (id,id)} }

Data-Driven Context Tunneling

Parametric static analyzer

Training data (programs)

Atomic features (a_1 , a_2 , ... ,a_{23})

Predicates over methods

Data-Driven Framework

- $\mathbf{f_{caller}}$ **: Property of caller methods**

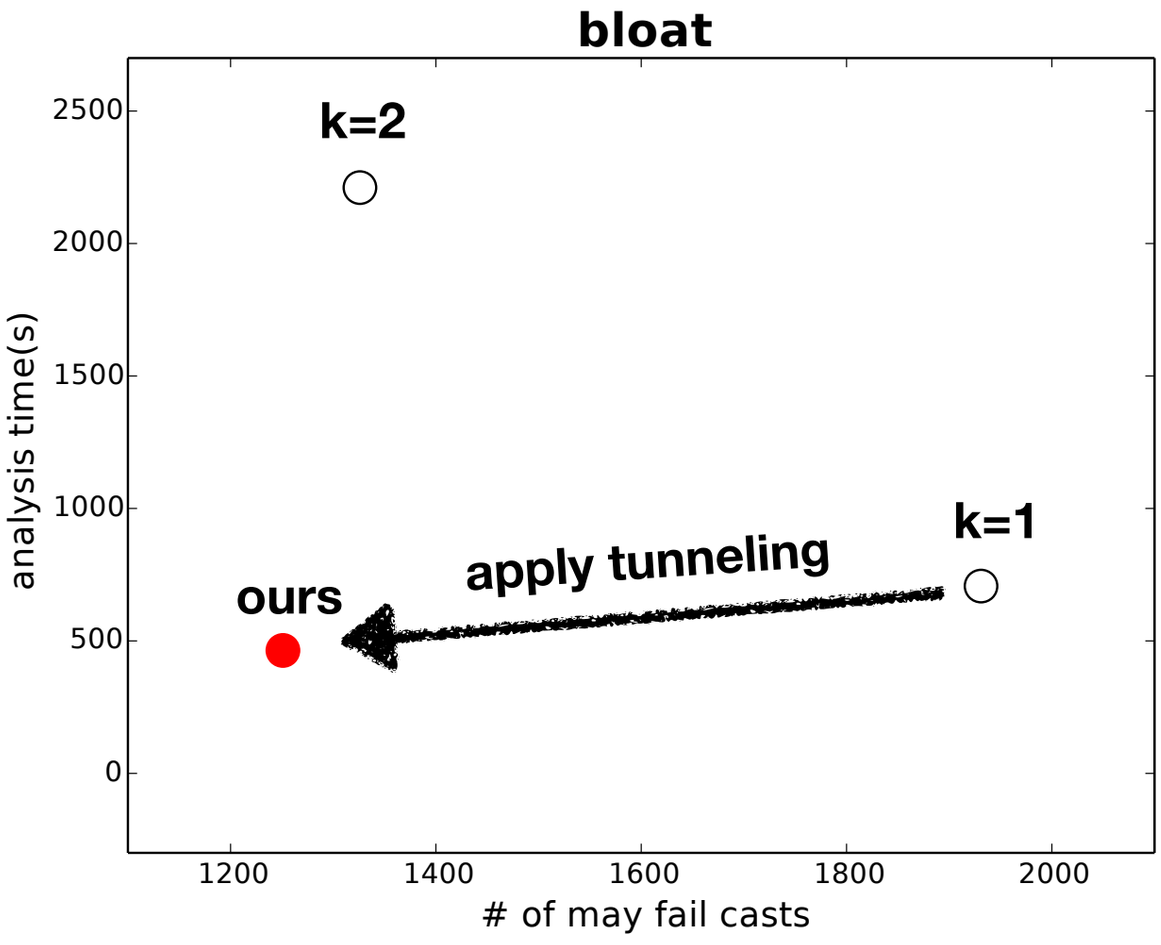
$$(\neg a_6 \wedge a_8 \wedge \neg a_{10} \wedge \neg a_{11} \wedge a_{14} \wedge a_{15} \wedge \neg a_{16} \wedge \neg a_{17} \wedge \neg a_{18} \wedge \neg a_{19} \wedge \neg a_{20} \wedge \neg a_{22}) \vee$$
$$(a_1 \wedge a_2 \wedge \neg a_3 \wedge \neg a_4 \wedge \neg a_6 \wedge a_8 \wedge \neg a_9 \wedge \neg a_{10} \wedge \neg a_{11} \wedge a_{12} \wedge a_{14} \wedge a_{15} \wedge \dots) \vee$$
$$(a_1 \wedge \neg a_2 \wedge \neg a_3 \wedge a_4 \wedge \neg a_6 \wedge \neg a_7 \wedge \wedge a_8 \wedge \neg a_9 \wedge \neg a_{10} \wedge \neg a_{11} \wedge \neg a_{12} \wedge \dots)$$

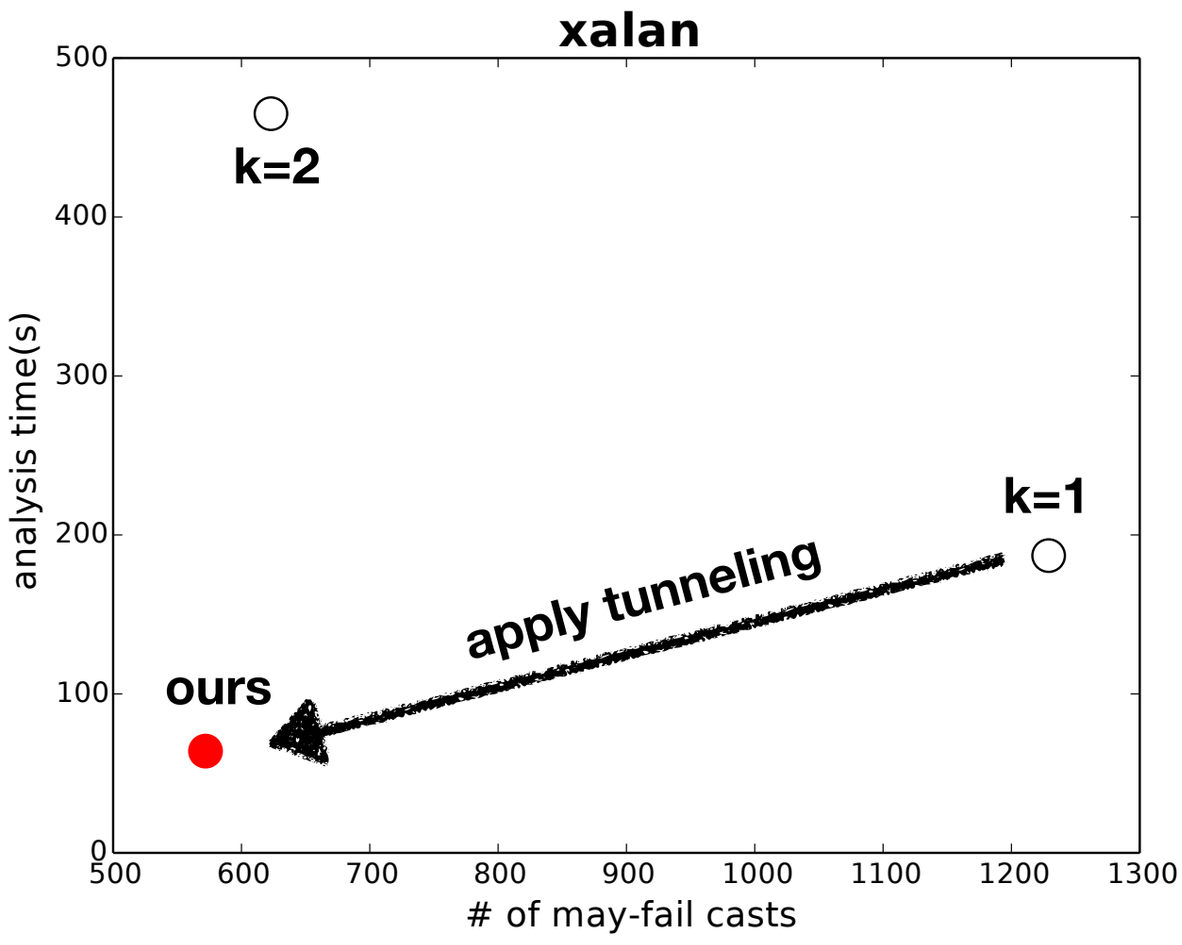
- $\mathbf{f_{callee}}$ **: Property of callee methods**

$$(a_1 \wedge \neg a_2 \wedge \neg a_3 \wedge \neg a_6 \wedge \neg a_9 \wedge a_{11} \wedge \wedge \neg a_{13} \wedge a_{14} \wedge a_{15} \wedge \neg a_{16} \wedge \neg a_{17} \wedge \dots) \vee$$
$$(a_1 \wedge \neg a_3 \wedge \neg a_4 \wedge a_7 \wedge \neg a_9 \wedge a_{12} \wedge a_{14} \wedge a_{15} \wedge \neg a_{16} \wedge \neg a_{19} \wedge \neg a_{21}) \vee$$
$$(\neg a_3 \wedge a_6 \wedge \neg a_9 \wedge a_{14} \wedge a_{15} \wedge \neg a_{18} \wedge \neg a_{19} \wedge \neg a_{23})$$

Performance Highlight

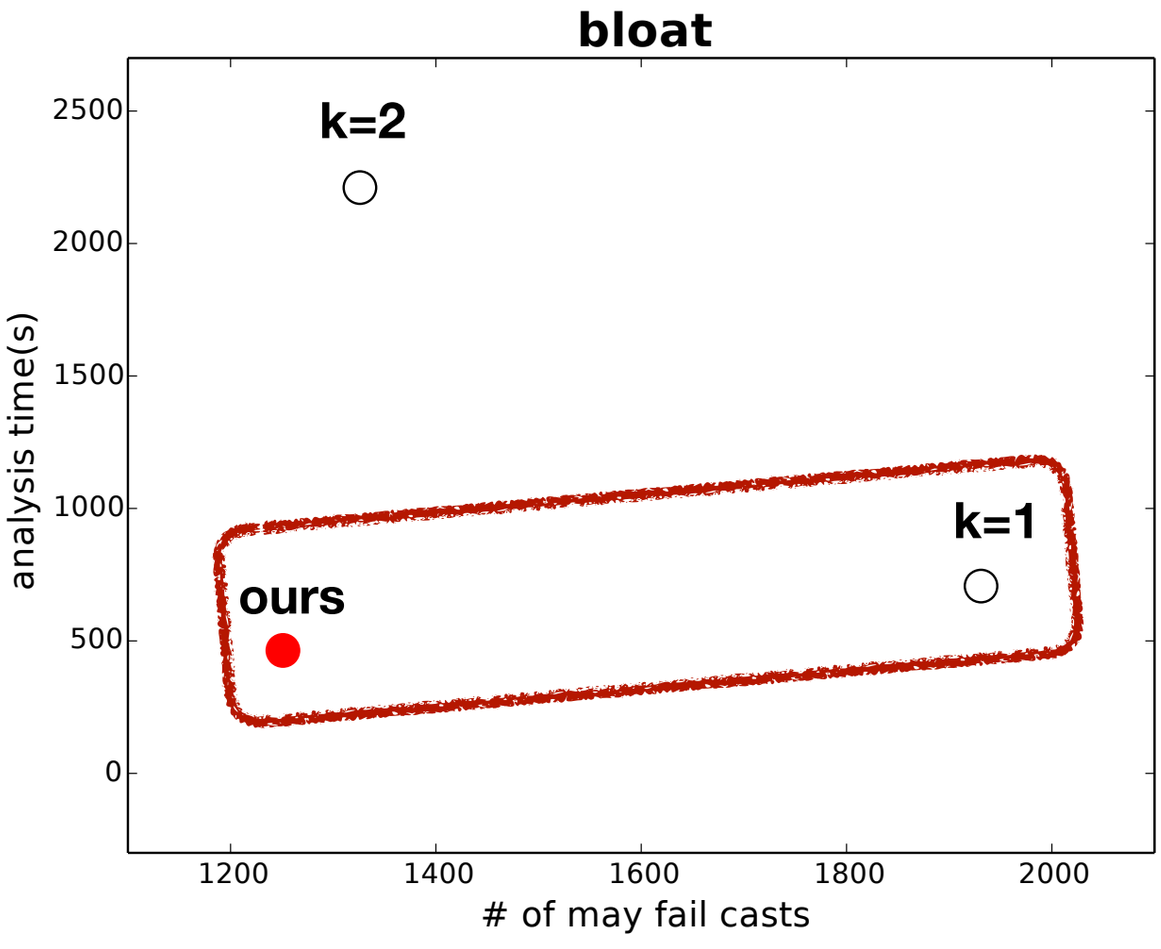
- **Hybrid 1-object sensitivity with context tunneling is**
  - **faster than k=1, and**
  - **more precise than k=2**

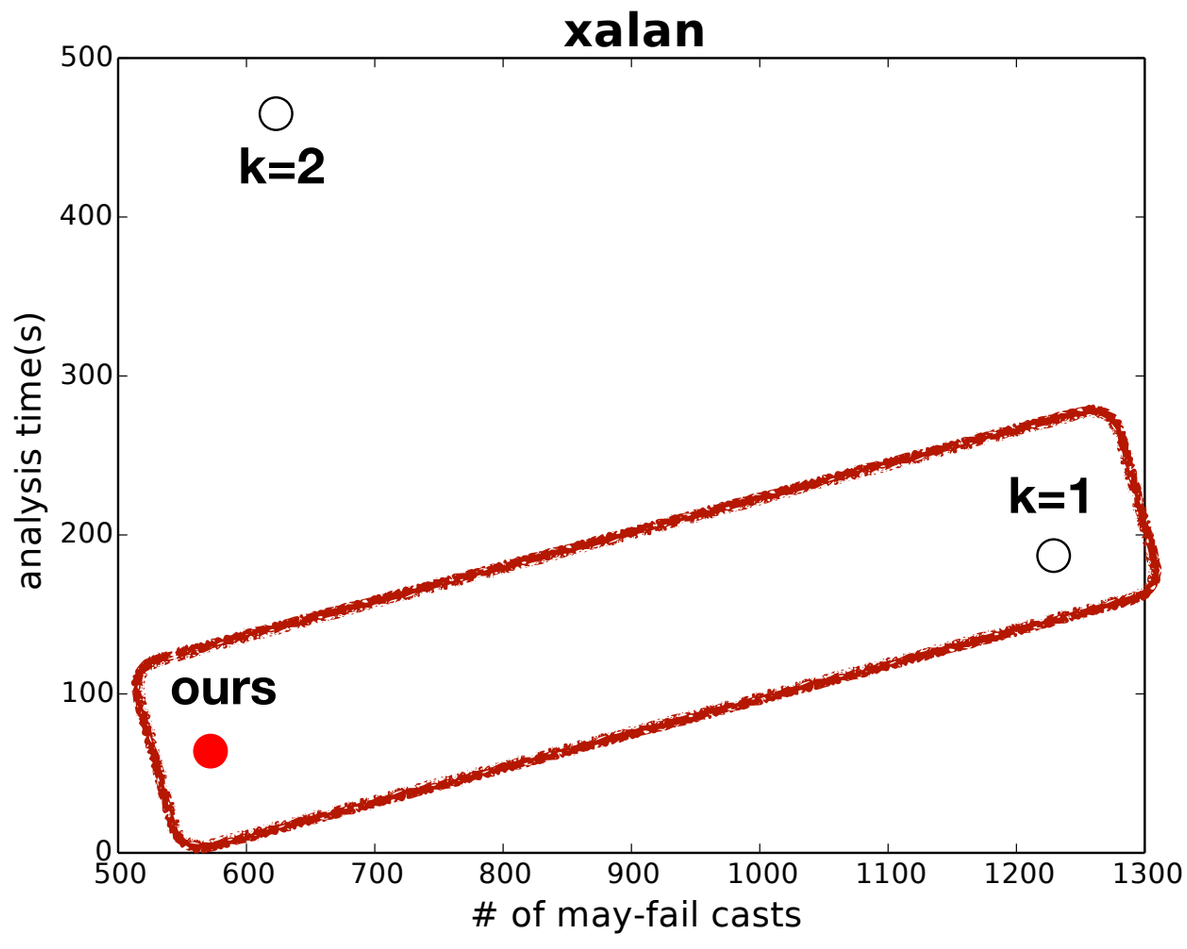




Performance Highlight

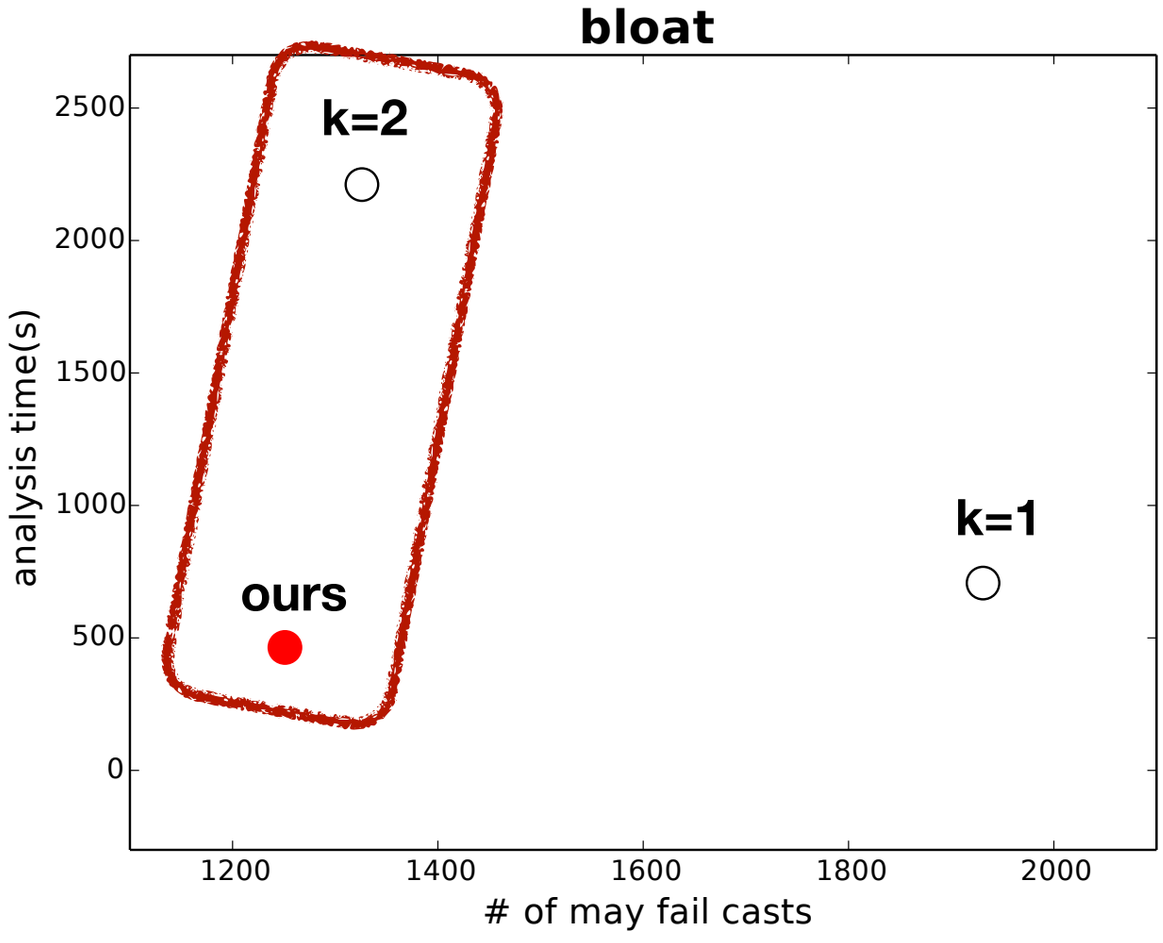
- **Hybrid 1-object sensitivity with context tunneling is**
  - **faster than k=1, and**
  - **more precise than k=2**

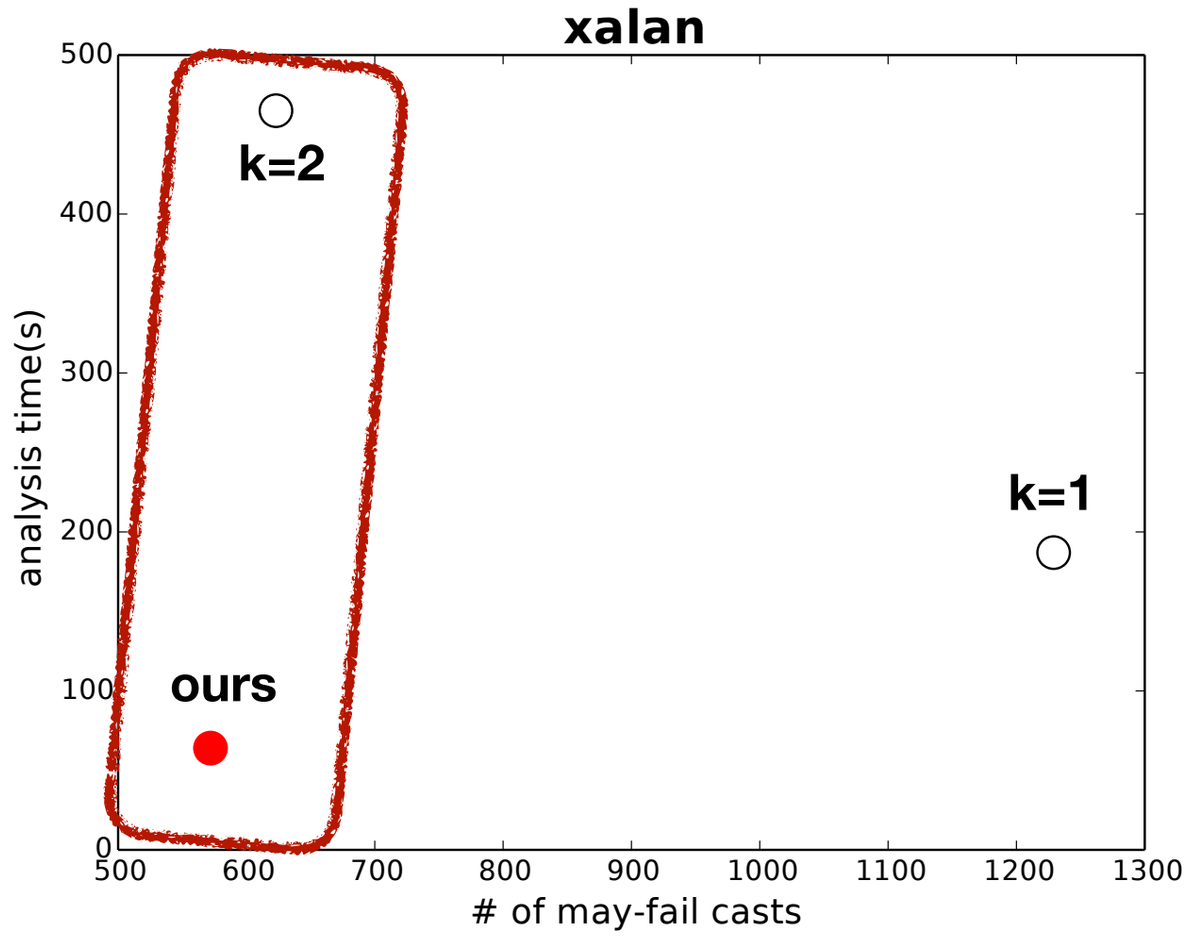




Performance Highlight

- **Hybrid 1-object sensitivity with context tunneling is**
  - **faster than k=1, and**
  - **more precise than k=2**





Details

- **Static analysis with context tunneling**
- **Learning context tunneling heuristics**

Parametric Static Analyzer

Proven queries

$$F_P : A_P \rightarrow Q \times cost$$

Tunneling abstraction
(Set of relations between methods)

Parametric Static Analyzer

$$F_P(\varnothing)$$

- **Most recent K context abstraction**

Parametric Static Analyzer

$$F_P(\mathbb{M}_P \times \mathbb{M}_P)$$

- **Equals to context insensitive analysis**
  - **All the methods will have the same context**

Parametric Static Analyzer

$$F_P(H_\Pi)$$

- **Ours use machine-tuned tunneling heuristic H.**

Parametric Tunneling Heuristic

$$H_{\prod} : P \rightarrow \mathbb{M}_p \times \mathbb{M}_p$$

- **Takes two Steps:**

    1. **Represents methods as feature set**

    2. **Generates a tunneling abstraction**

Features

- **Predicates over method**

$$\mathbb{A} = \{a_1, a_2, \ldots, a_{23}\}$$

$$a_i : M_P \rightarrow \{true, false\}$$

- **23 syntactic features**
    - **e.g) is inner class' method, has store instruction, takes void input, …**

(1) Represent Method as Features

- **We have two atomic features**
- **Let P has four methods**

$$\Pi = \langle f_1 = a_1 \wedge a_2, \\ f_2 = \neg a_1 \wedge \neg a_2 \rangle$$

$$M_1 = \{a_1, a_2\}$$
$$M_2 = \{a_1\}$$
$$M_3 = \{a_2\}$$
$$M_4 = \{\}$$

(1) Represent Method as Features

- **We have two atomic features**
- **Let P has four methods**

$$\Pi = \langle f_1 = a_1 \wedge a_2, \quad f_2 = \neg a_1 \wedge \neg a_2 \rangle$$

$$M_1 = \{a_1, a_2\}$$
$$M_2 = \{a_1\}$$
$$M_3 = \{a_2\}$$
$$M_4 = \{\}$$

$\{M_1\}$

$\{M_4\}$

(2) Generate Tunneling Abstraction

- **We have two atomic features**
- **Let P has four methods**

$$M_1 = \{a_1, a_2\}$$
$$M_2 = \{a_1\}$$
$$M_3 = \{a_2\}$$
$$M_4 = \{\}$$

$\{M_1\}$

$$\Pi = \langle f_1 = a_1 \wedge a_2,\ f_2 = \neg a_1 \wedge \neg a_2 \rangle$$

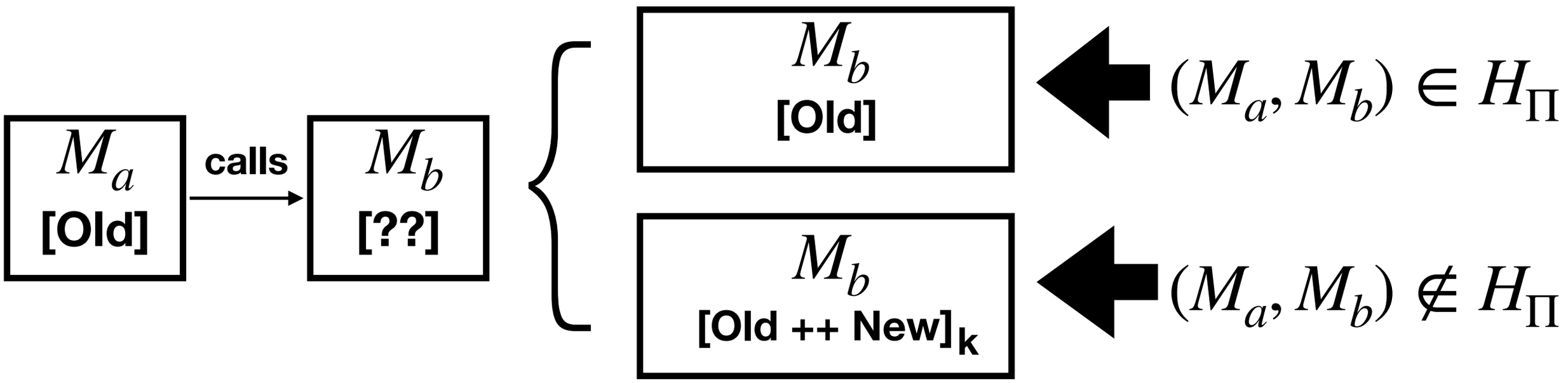
$\{M_4\}$

$$H_\Pi : \{(M_1, M_1), (M_1, M_2), (M_1, M_3), (M_1, M_4), (M_2, M_4), (M_3, M_4), (M_4, M_4)\}$$

Caller is M_1 or Callee is M_4

Context Tunneling: Selectively Update Contexts

$$H_{\Pi} : \{(M_1, M_1), (M_1, M_2), \dots\}$$



Learning Problem

$$P_1, P_2, \ldots, P_n \Rightarrow \langle f_1, f_2 \rangle$$

Code base

Find <f1, f2> which maximizes precision while more scalable than <false, false> over code base

Analysis without context tunneling

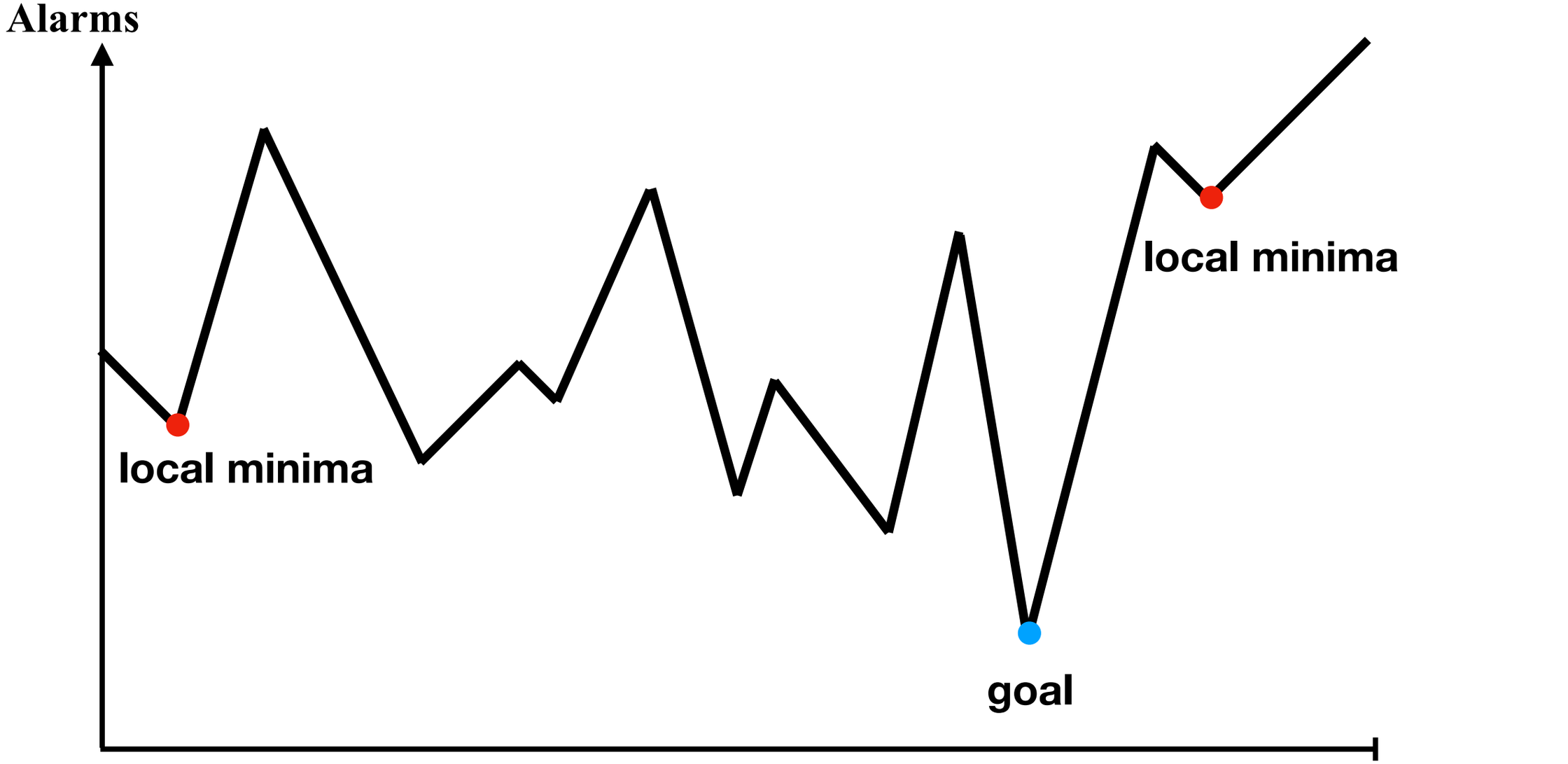
Challenge: Non-monotonicity

- **Many parametric static analyses assume monotonicity but context tunneling is not**

Applying tunneling to more method calls $\not\Rightarrow$ **Increase of precision**

Context Tunneling: Not Monotone

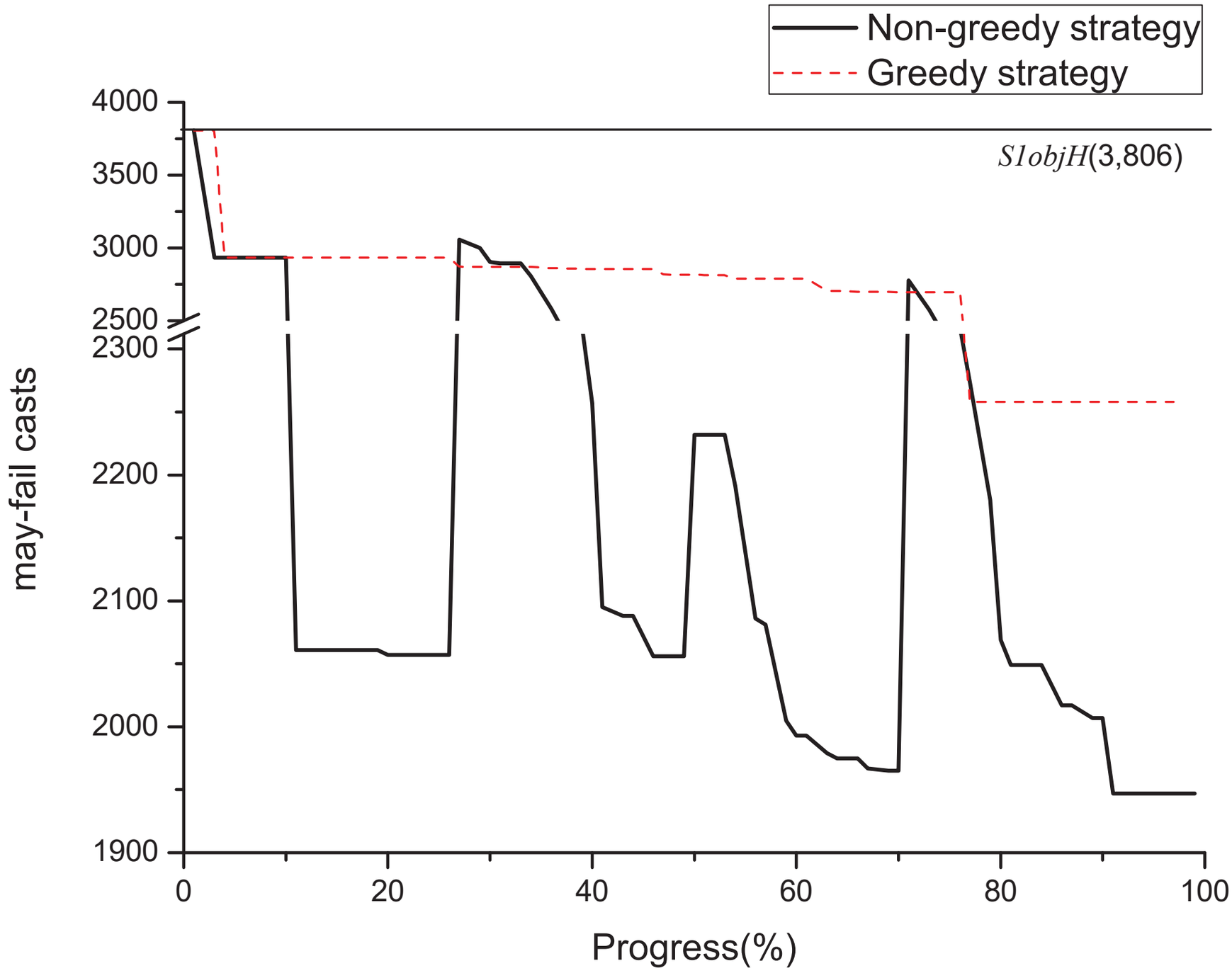
- **Greedy algorithms do not work in this space**



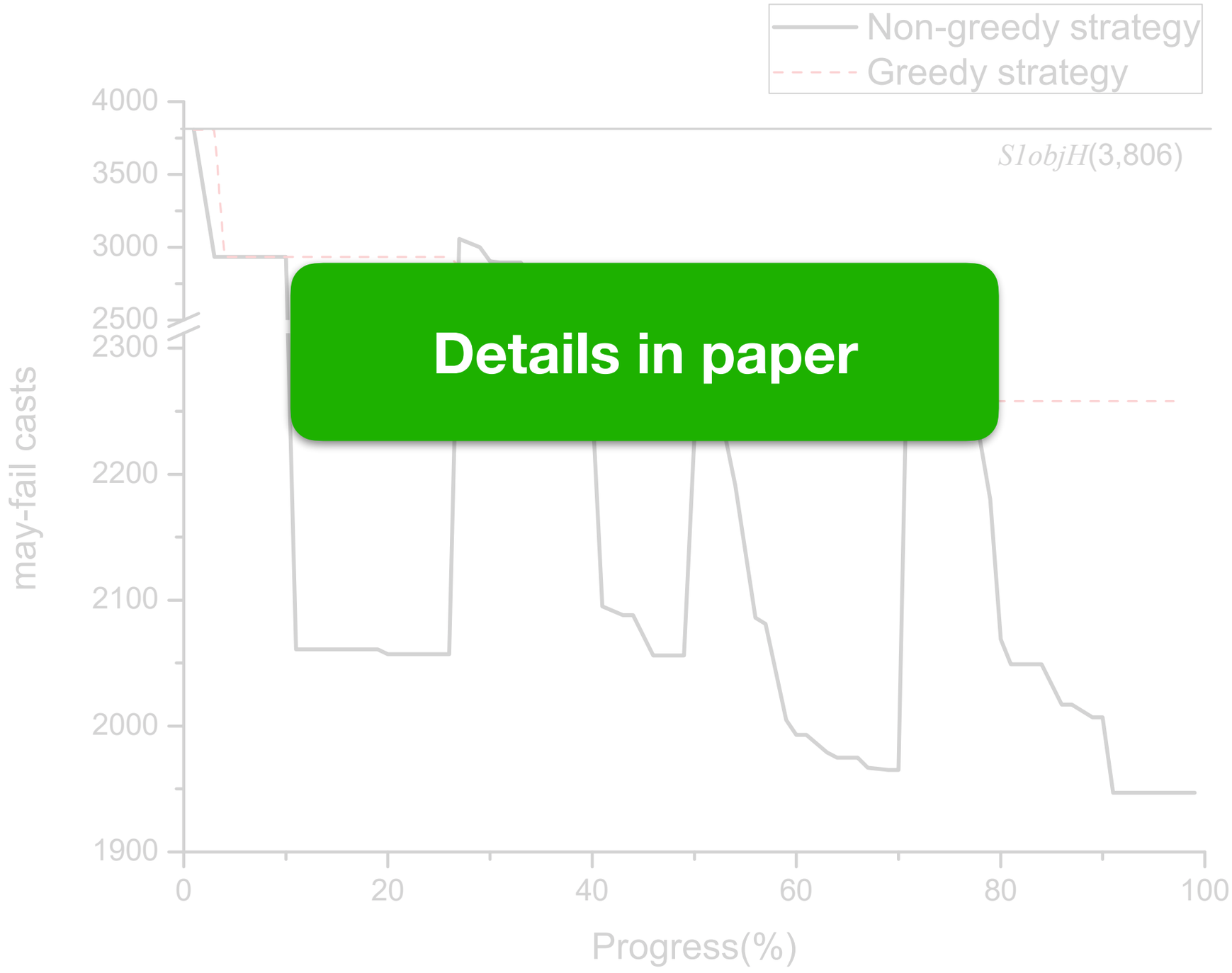
$A_P = \emptyset$

$A_P = \mathbb{M}_P \times \mathbb{M}_P$

Specialized Learning Algorithm for Context Tunneling



Specialized Learning Algorithm for Context Tunneling



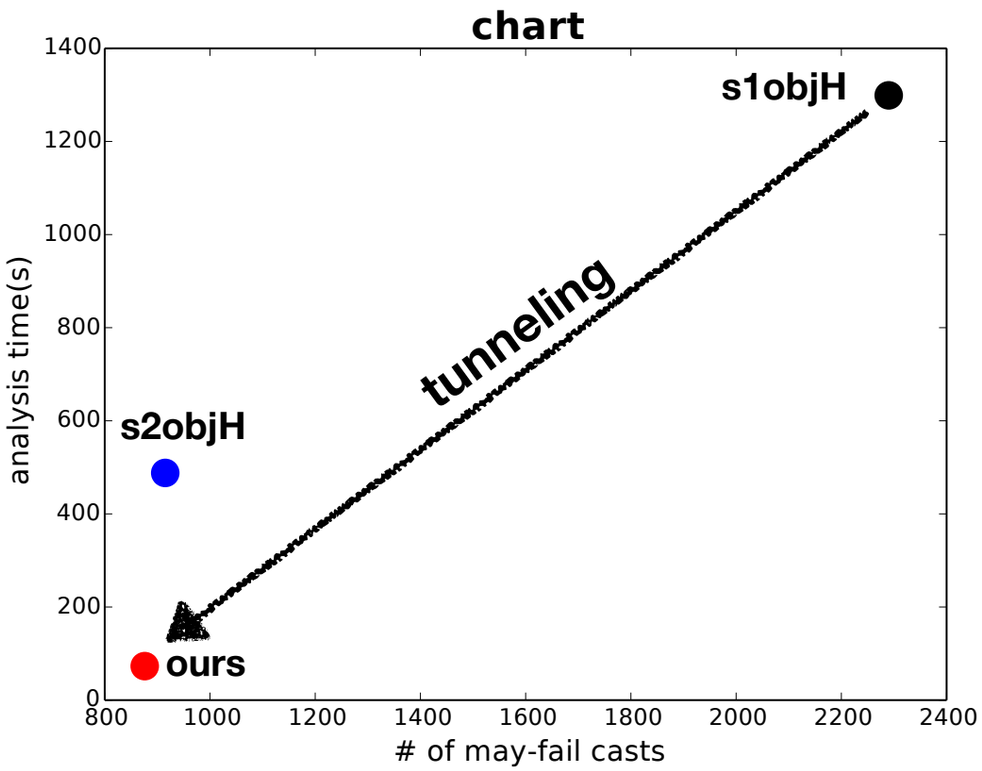
Evaluation

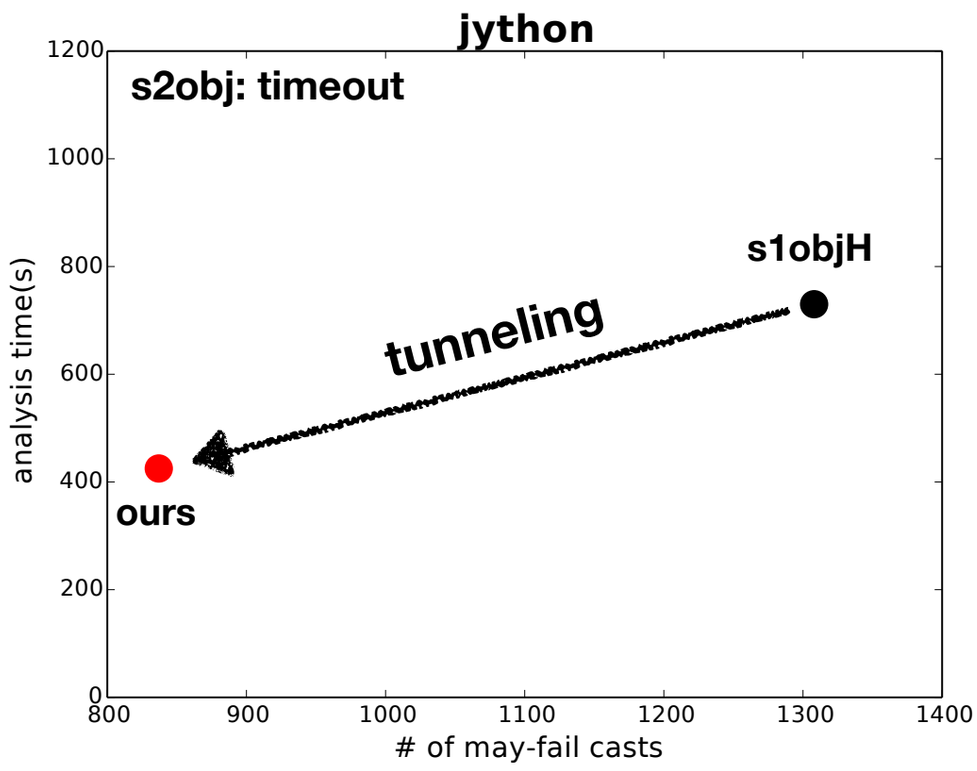
Setting

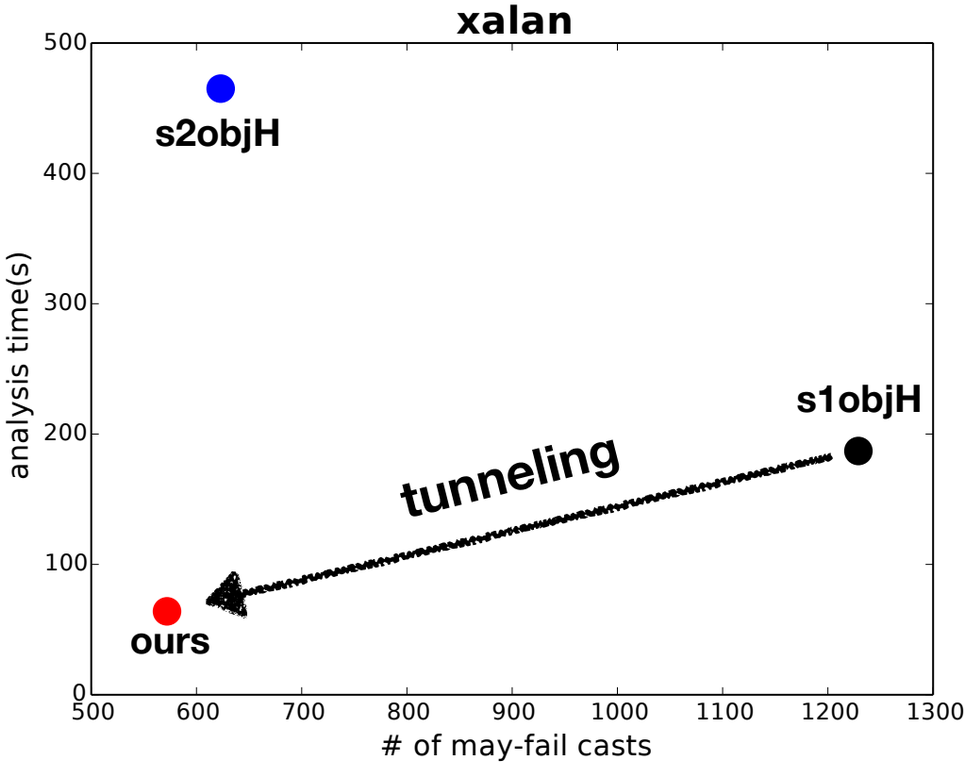
- Tunneled Doop:
  - **Points-to analysis framework**

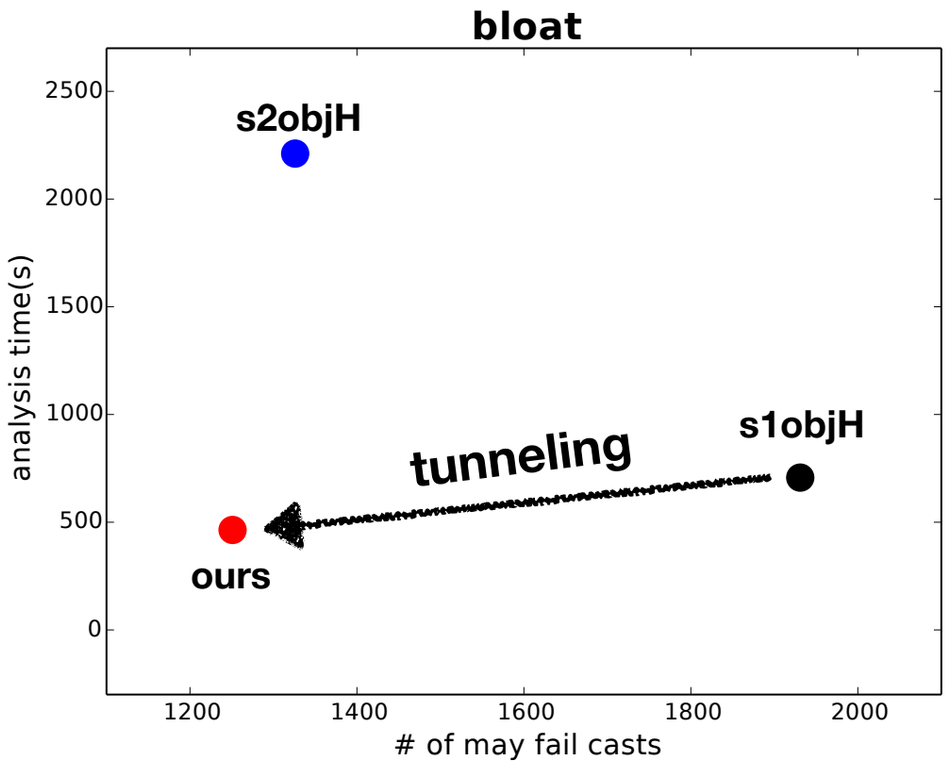
- Four flavors of context sensitivity
  - **Selective hybrid object sensitivity**
  - **Object sensitivity**
  - **Call-site sensitivity**
  - **Type sensitivity**
- DaCapo Benchmark suite
  - **Four small programs for training**
  - **Others for testing**

Effectiveness









Learned Heuristic

Deeper object-sensitivity may be useful for methods that belong to sub-objects. [Milanova2005]

Learned Heuristic

Context Tunneling may be useful for methods that belong to sub-objects. [Milanova2005]

Learned Heuristic

$$\Pi_{obj} = \langle f_1, f_2 \rangle$$

$$f_1 = \cdots \vee (A8 \wedge B5 \wedge \neg B3 \wedge A1 \wedge \neg A6 \wedge \neg A3 \wedge \neg B9 \wedge B4 \wedge \neg A9 \wedge \neg B11 \wedge \neg A2 \wedge \neg B7 \wedge \neg B1 \wedge \neg B8 \wedge \underline{B12} \wedge \underline{B10} \wedge \neg B13 \wedge \underline{B6} \wedge A5 \wedge \underline{A10} \wedge A7 \wedge \neg A4)$$

f_1 :When callee method's base object is allocated in these methods

A10: Method is constructor method
B10: Method has a heap allocation
B12: Method has at least one heap allocation
B6: Method contains store instruction

Conclusion

- **Do not keep most recent K**
- **Instead, keep most important K**