# Learning Graph-based Heuristics for Pointer Analysis without Handcrafting Application-Specific Features

Minseok Jeon, Myungho Lee, and Hakjoo Oh

KOREA UNIVERSITY

**OOPSLA 2020**

# Importance of Pointer Analysis

- Pointer analysis is a key component of various software engineering tools

## Automatic Program Repair Tools

"we first perform pointer analysis on the whole program"
"Pointer analyses of different sensitivities can be used to increase the precision of the analysis or to improve the analysis speed."

-Gao et al. [ICSE 15]

"Another limitation comes from the pointer analysis…. context-insensitive may information is not precise enough…"

-Lee et al. [FSE 18]

## Symbolic Execution Tool

"In this paper, we propose a novel approach that uses pointer alias analysis to group memory objects …"

-Kapus and Cadar. [FSE 19]

## Bug Finder

"To find memory leaks statically, …, its underlying pointer analysis must also be scalable and accurate"

-Sui et al. [TSE 14]

# Importance of Pointer Analysis

- Pointer analysis is a key component of various software engineering tools

## Automatic Program Repair Tools

"we first perform pointer analysis on the whole program"
"Pointer analyses of different sensitivities can be used to increase the precision of the analysis or to improve the analysis speed."

-Gao et al. [ICSE 15]

"Another limitation comes from the pointer analysis…. context-insensitive may information is not precise enough…"

-Lee et al. [FSE 18]

## Symbolic Execution Tool

"In this paper, we propose a novel approach that uses pointer alias analysis to group memory objects …"

-Kapus and Cadar. [FSE 19]

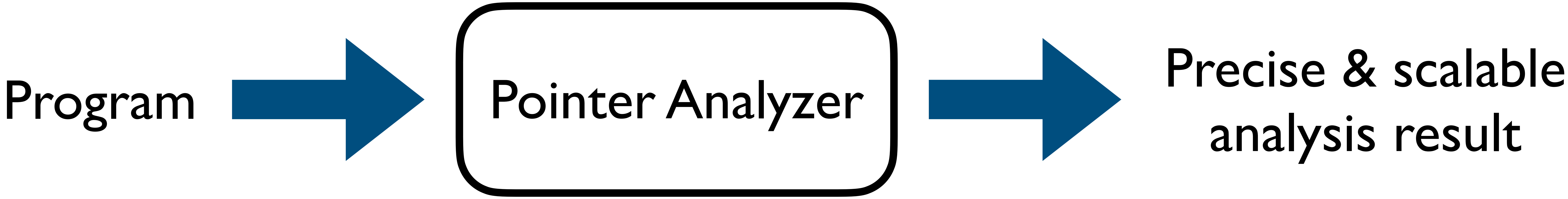
## Bug Finder

"To find memory leaks statically, …, its underlying pointer analysis must also be scalable and accurate"

-Sui et al. [TSE 14]

# Analysis Heuristics in Pointer Analysis

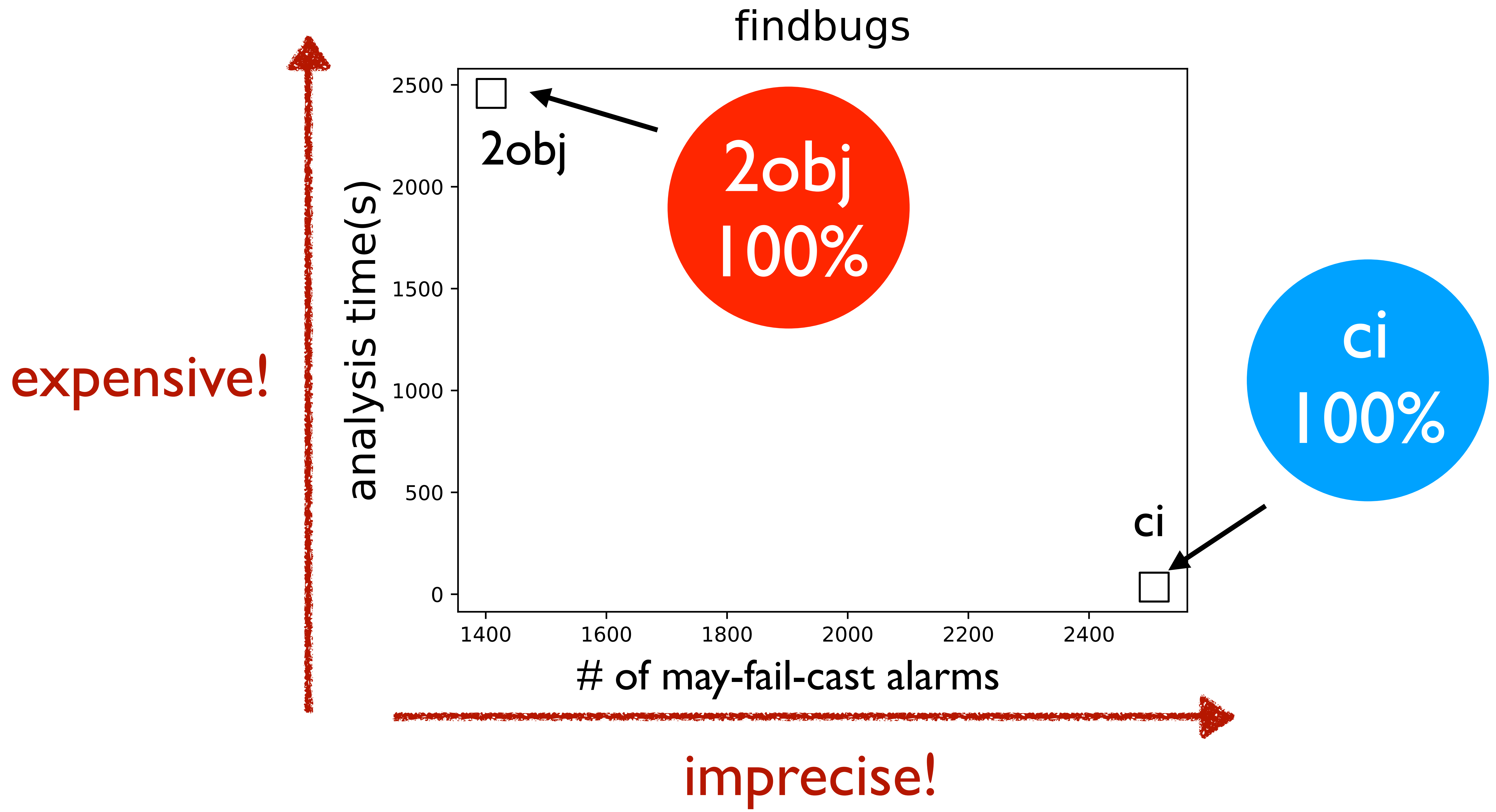
- Cost-effective pointer analyzer contains various analysis heuristics




- Context sensitivity heuristics (which method calls should be analyzed precisely?)
- Heap abstraction heuristics (which objects should be analyzed precisely?)
- Context tunneling heuristics (which context elements should be maintained?)
- …

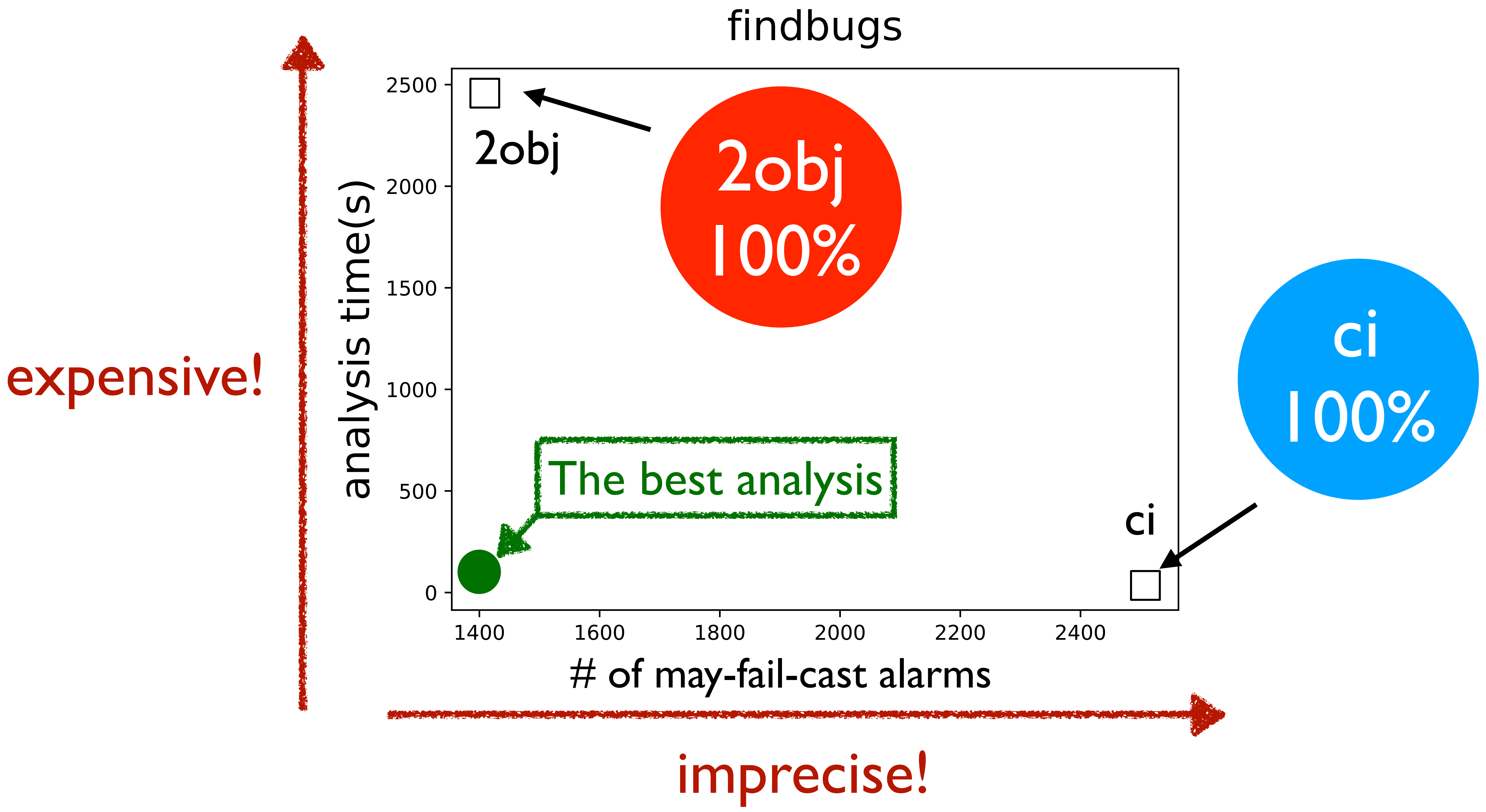
# Necessity of Context Sensitivity Heuristics

- Full 2-object-sensitivity (2obj) is precise but too expensive
- Context insensitive analysis (ci) is cheap but imprecise



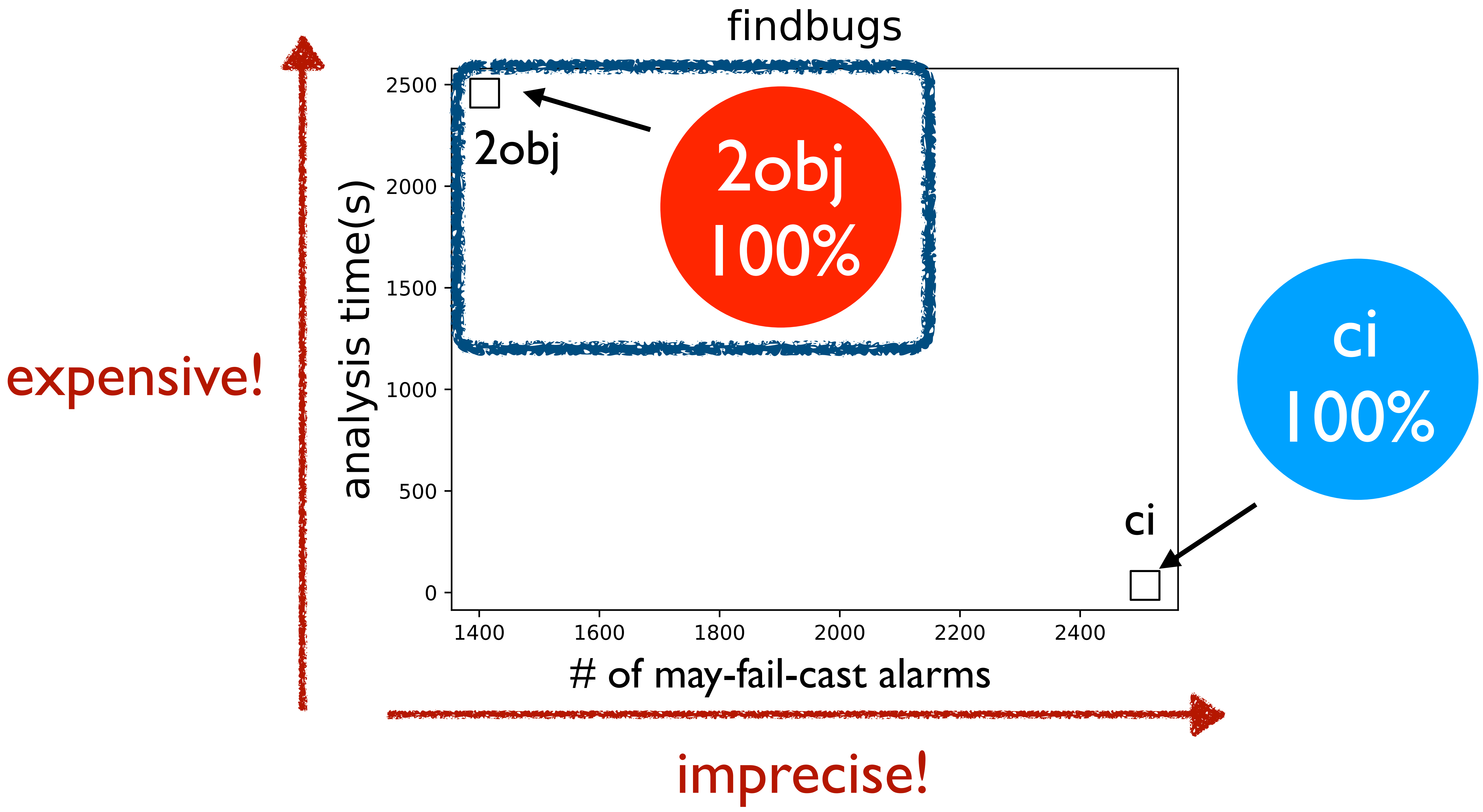
# Necessity of Context Sensitivity Heuristics

- Full 2-object-sensitivity (2obj) is precise but too expensive
- Context insensitive analysis (ci) is cheap but imprecise



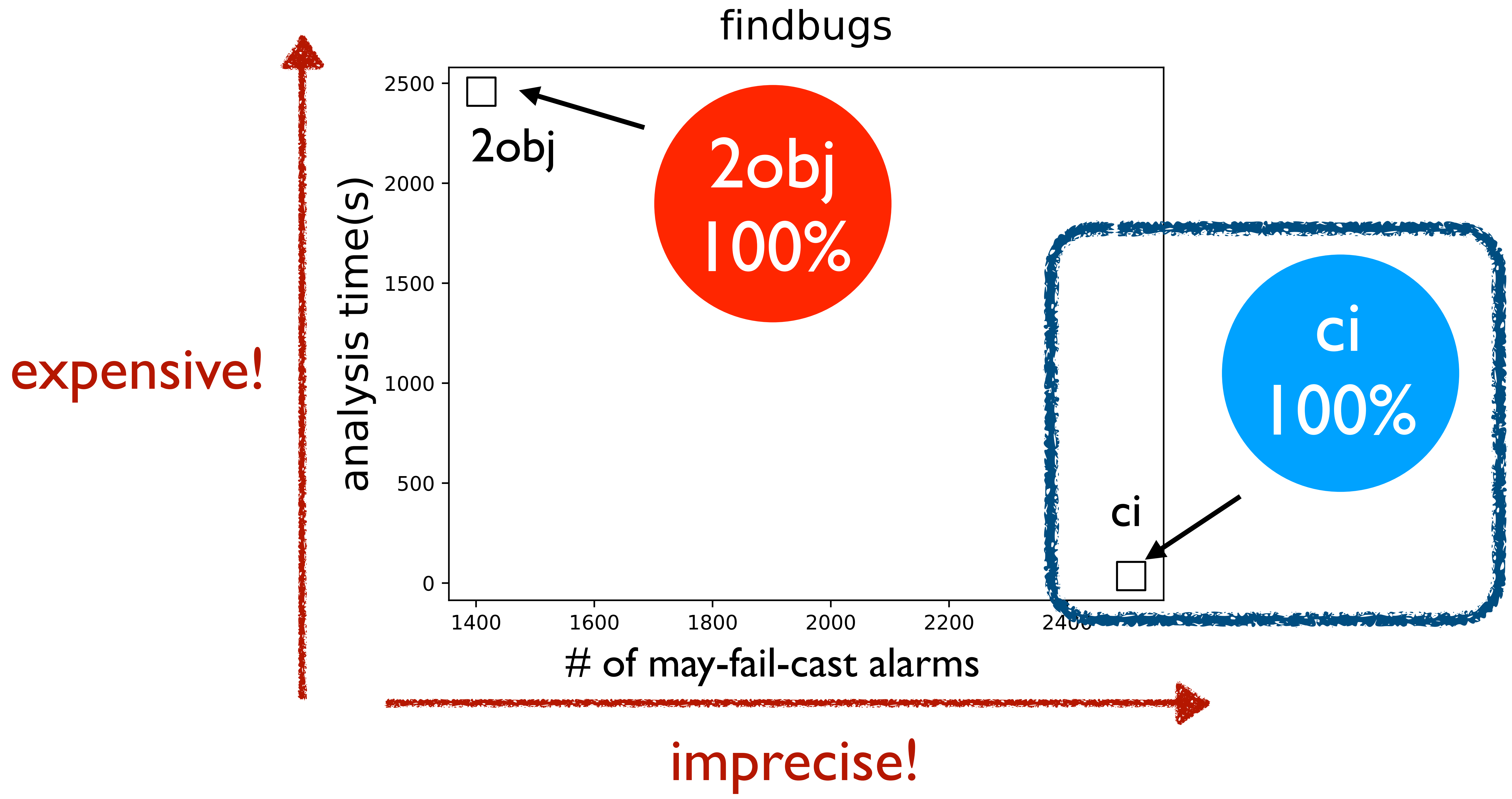
# Necessity of Context Sensitivity Heuristics

- Full 2-object-sensitivity (2obj) is precise but too expensive
- Context insensitive analysis (ci) is cheap but imprecise



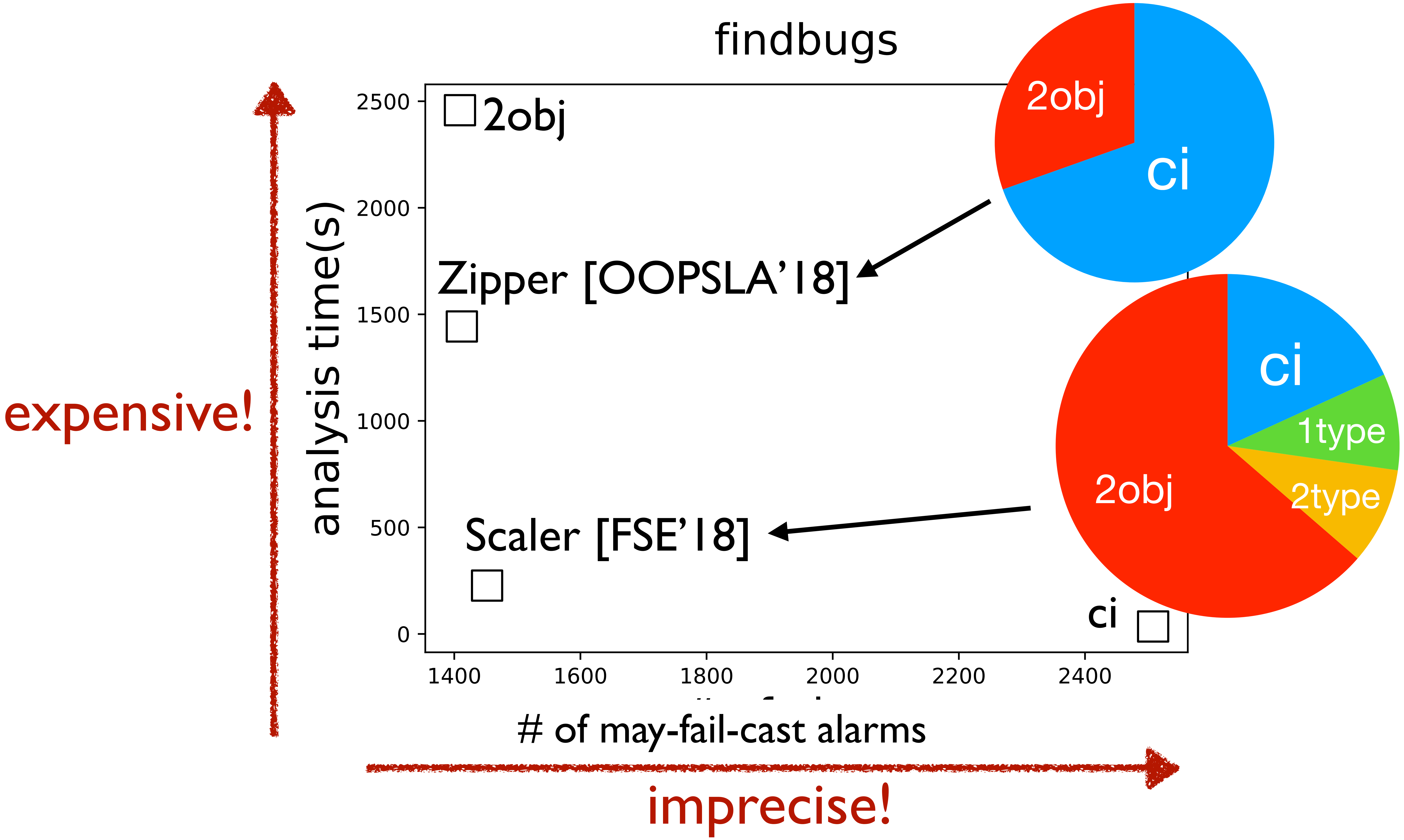
# Necessity of Context Sensitivity Heuristics

- Full 2-object-sensitivity (2obj) is precise but too expensive
- Context insensitive analysis (ci) is cheap but imprecise



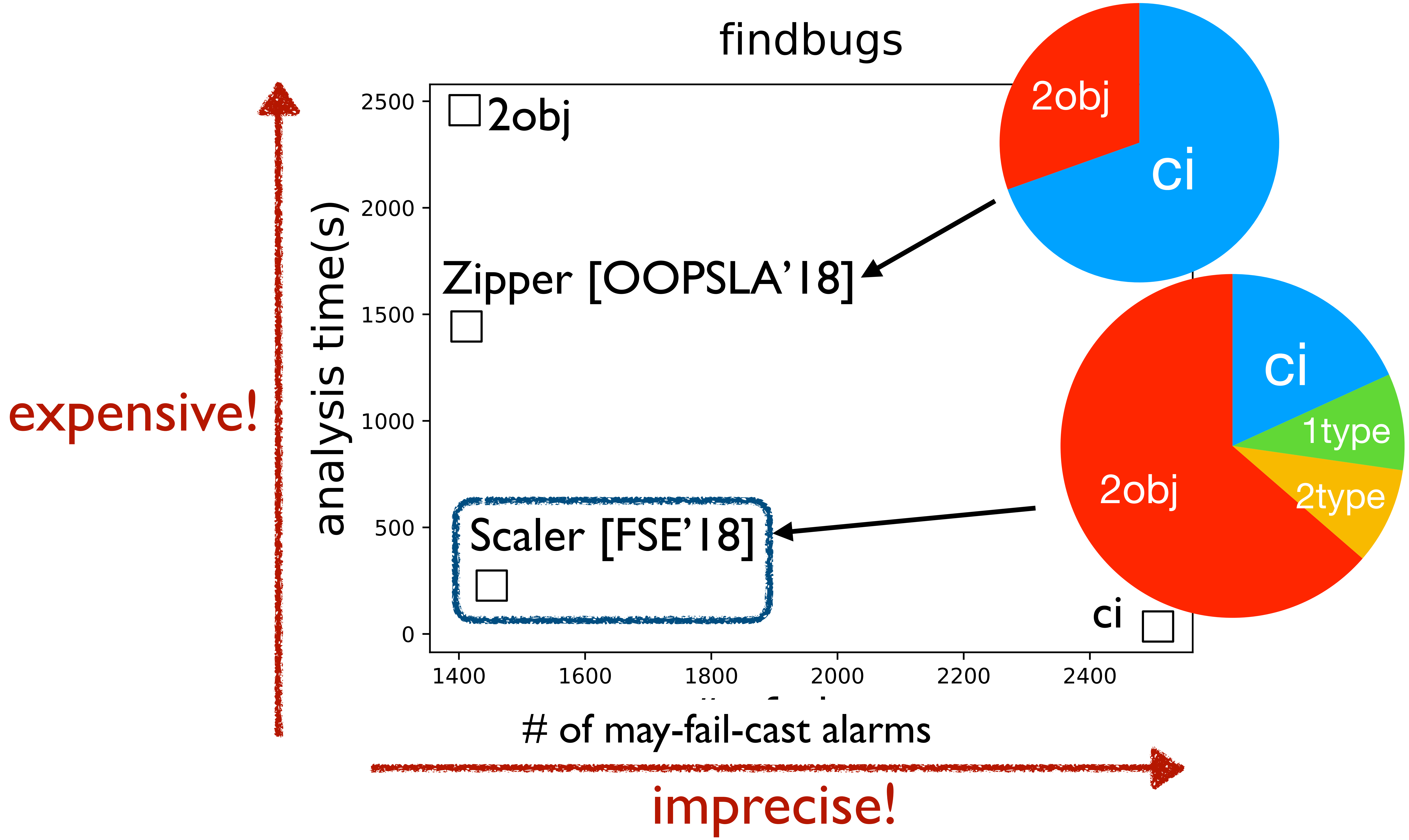
# Necessity of Context Sensitivity Heuristics

- Context sensitivity heuristics assign different context for each method call



# Necessity of Context Sensitivity Heuristics

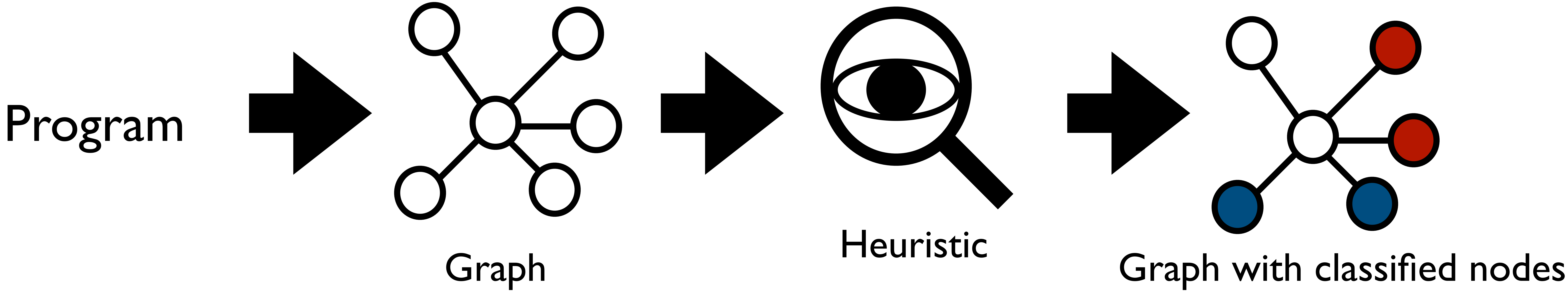
- Context sensitivity heuristics assign different context for each method call



# Current Trend on Designing Heuristics

- A recent trend in pointer analyses is use of graph-based heuristics

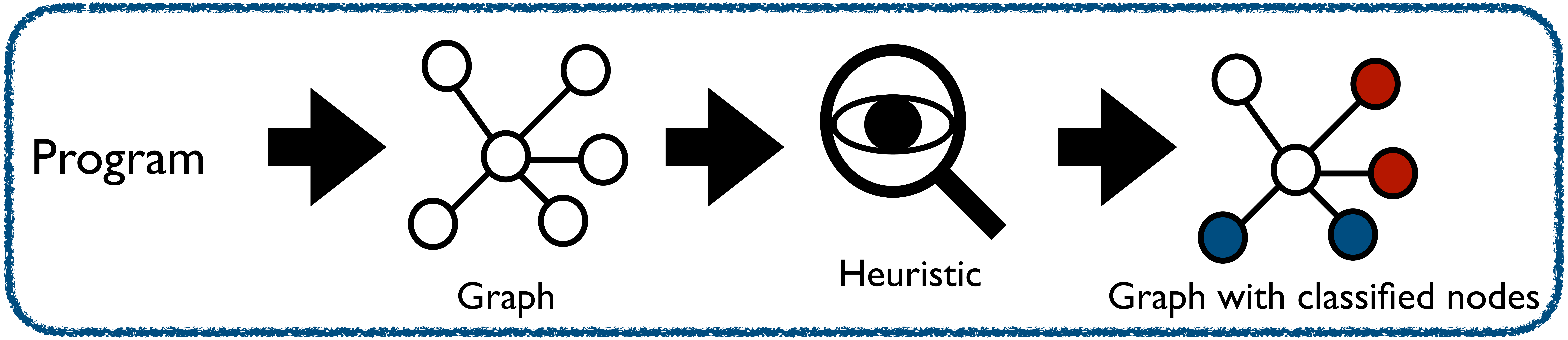
Bean [SAS' 16], Mahjong [PLDI'17], Zipper [OOPSLA '18], Scaler [FSE' 18], Eagle [OOPSLA' 19]



# Current Trend on Designing Heuristics

- A recent trend in pointer analyses is use of graph-based heuristics

Bean [SAS' 16], Mahjong [PLDI'17], Zipper [OOPSLA '18], Scaler [FSE' 18], Eagle [OOPSLA' 19]




**Workflow of graph-based heuristics**

# Current Trend on Designing Heuristics

- A recent trend in pointer analyses is use of graph-based heuristics

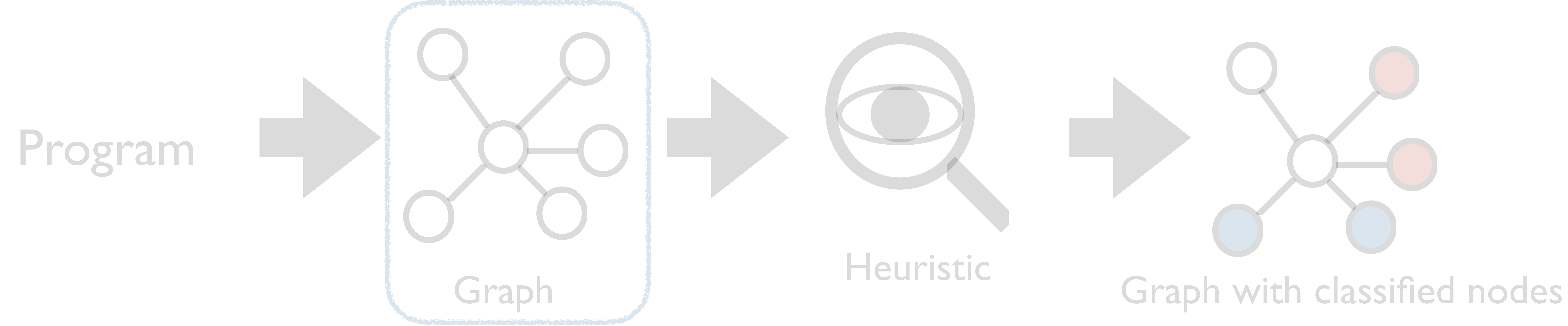
Bean [SAS' 16], Mahjong [PLDI'17], Zipper [OOPSLA '18], Scaler [FSE' 18], Eagle [OOPSLA' 19]

# Current Trend on Designing Heuristics

- A recent trend in pointer analyses is use of graph-based heuristics

Bean [SAS' 16], Mahjong [PLDI'17], Zipper [OOPSLA '18], Scaler [FSE' 18], Eagle [OOPSLA' 19]

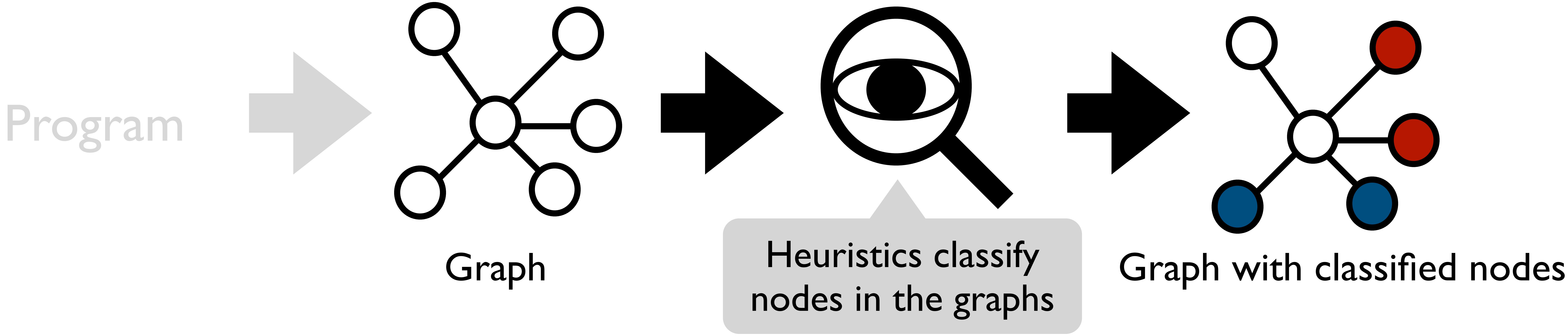
- Bean: Use object allocation graph
- Mahjong: Use field points-to graph
- Zipper: Use precision flow graph
- Scaler: Use object allocation graph
- Eagle: Use CFL-reachability graph



# Current Trend on Designing Heuristics

- A recent trend in pointer analyses is use of graph-based heuristics

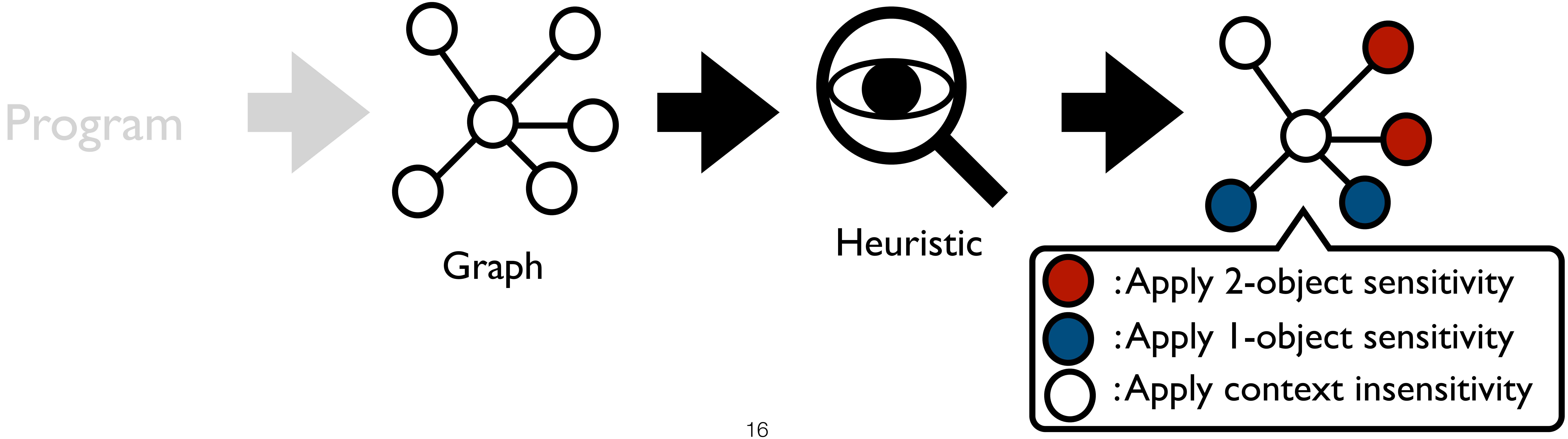
Bean [SAS' 16], Mahjong [PLDI'17], Zipper [OOPSLA '18], Scaler [FSE' 18], Eagle [OOPSLA' 19]



# Current Trend on Designing Heuristics

- A recent trend in pointer analyses is use of graph-based heuristics

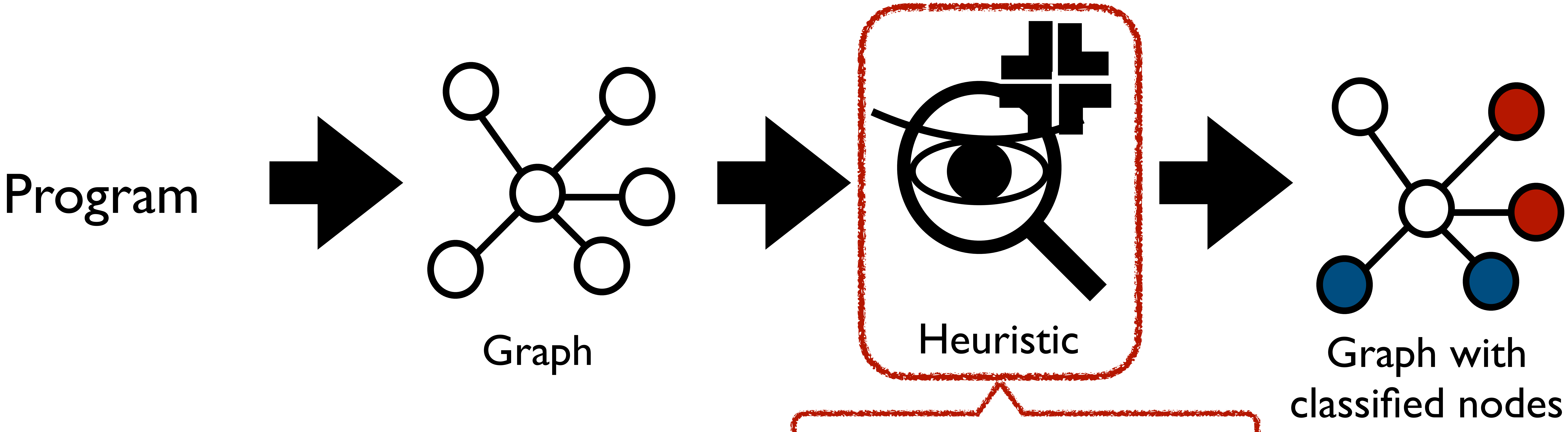
Bean [SAS' 16], Mahjong [PLDI'17], Zipper [OOPSLA '18], Scaler [FSE' 18], Eagle [OOPSLA' 19]



# Problem

- However, it is a difficult task to design graph-based heuristics

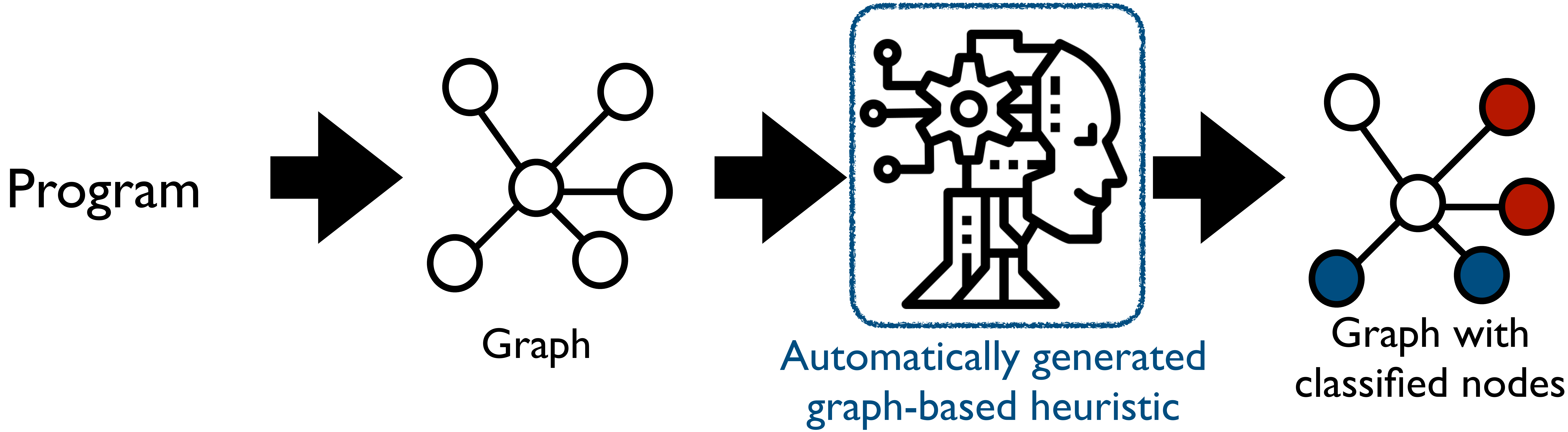
Bean [SAS' 16], Mahjong [PLDI'17], Zipper [OOPSLA '18], Scaler [FSE' 18], Eagle [OOPSLA' 19]




- Need expertise
- Need human effort
- …

# Our Goal

- Our goal is to automatically generate graph based heuristics without human effort



# Previous Data-driven Program Analysis

- Prior data-driven program analyses require application specific features

| Type | # | Features |
|---|---|---|
| A | 1 | local variable |
| | 2 | global variable |
| | 3 | structure field |
| | 4 | location created by dynamic memory allocation |
| | 5 | defined at one program point |
| | 6 | location potentially generated in library code |
| | 7 | assigned a constant expression (e.g., x = c1 + c2) |
| | 8 | compared with a constant expression (e.g., x < c) |
| | 9 | compared with an other variable (e.g., x < y) |
| | 10 | negated in a conditional expression (e.g., if (!x)) |
| | 11 | directly used in malloc (e.g., malloc(x)) |
| | 12 | indirectly used in malloc (e.g., y = x; malloc(y)) |
| | 13 | directly used in realloc (e.g., realloc(x)) |
| | 14 | indirectly used in realloc (e.g., y = x; realloc(y)) |
| | 15 | directly returned from malloc (e.g., x = malloc(e)) |
| | 16 | indirectly returned from malloc |
| | 17 | directly returned from realloc (e.g., x = realloc(e)) |
| | 18 | indirectly returned from realloc |
| | 19 | incremented by one (e.g., x = x + 1) |
| | 20 | incremented by a constant expr. (e.g., x = x + (1+2)) |
| | 21 | incremented by a variable (e.g., x = x + y) |
| | 22 | decremented by one (e.g., x = x - 1) |
| | 23 | decremented by a constant expr (e.g., x = x - (1+2)) |
| | 24 | decremented by a variable (e.g., x = x - y) |
| | 25 | multiplied by a constant (e.g., x = x * 2) |
| | 26 | multiplied by a variable (e.g., x = x * y) |
| | 27 | incremented pointer (e.g., p++) |
| | 28 | used as an array index (e.g., a[x]) |
| | 29 | used in an array expr. (e.g., x[e]) |
| | 30 | returned from an unknown library function |
| | 31 | modified inside a recursive function |
| | 32 | modified inside a local loop |
| | 33 | read inside a local loop |
| B | 34 | $1 \wedge 8 \wedge (11 \vee 12)$ |
| | 35 | $2 \wedge 8 \wedge (11 \vee 12)$ |
| | 36 | $1 \wedge (11 \vee 12) \wedge (19 \vee 20)$ |
| | 37 | $2 \wedge (11 \vee 12) \wedge (19 \vee 20)$ |
| | 38 | $1 \wedge (11 \vee 12) \wedge (15 \vee 16)$ |

Features for flow sensitivity heuristics

| Type | # | Features |
|---|---|---|
| A | 1 | leaf function |
| | 2 | function containing malloc |
| | 3 | function containing realloc |
| | 4 | function containing a loop |
| | 5 | function containing an if statement |
| | 6 | function containing a switch statement |
| | 7 | function using a string-related library function |
| | 8 | write to a global variable |
| | 9 | read a global variable |
| | 10 | write to a structure field |
| | 11 | read from a structure field |
| | 12 | directly return |
| | 13 | indirectly retur |
| | 14 | directly return |
| | 15 | indirectly retur |
| | 16 | directly return |
| | 17 | indirectly return a reallocated memory |
| | 18 | return expression involves field access |
| | 19 | return value depends on a structure field |
| | 20 | return void |
| | 21 | directly invoked with a constant |
| | 22 | constant is passed to an argument |
| | 23 | invoked with an unknown value |
| | 24 | functions having no arguments |
| | 25 | functions having one argument |
| | 26 | functions having more than one argument |
| | 27 | functions having an integer argument |
| | 28 | functions having a pointer argument |
| | 29 | functions having a structure as an argument |
| B | 30 | $2 \wedge (21 \vee 22) \wedge (14 \vee 15)$ |
| | 31 | $2 \wedge (21 \vee 22) \wedge \neg(14 \vee 15)$ |
| | 32 | $2 \wedge 23 \wedge (14 \vee 15)$ |

Features for context sensitivity heuristics

| # | Description | Calculation |
|---|---|---|
| 1 | Number of variables in the block $\mathcal{X}^c_\sqcup$ | $\lvert\mathcal{X}^c_\sqcup\rvert$ |
| 2 | Number of constraints in the factor $\mathcal{X}^c_\sqcup$ | $\lvert\mathcal{I}_\sqcup(\mathcal{X}^c_\sqcup)\rvert$ |
| 3 | Number of generators in the factor $\mathcal{X}^c_\sqcup$ | $\lvert\mathcal{G}_\sqcup(\mathcal{X}^c_\sqcup)\rvert$ |
| 4 | Number of loop head variables in $c$ | $\lvert\mathcal{X}^c \cap \mathcal{X}_H\rvert$ |
| 5 | Boolean, true if $\mathcal{X}^c$ is a subset of $\mathcal{X}_H$ | $\mathcal{X}^c \subseteq \mathcal{X}_H$ |
| 6 | Boolean, true if $c$ is in $\mathcal{I}_H$ | $c \in \mathcal{I}_H$ |
| 7 | Number of variables in constraint $c$ | $\lvert\mathcal{X}^c\rvert$. |
| 11 | Boolean, true if $c$ coarsens partition | COARS |
| 12 | Boolean, true if $c$ is an equality | ∘ is = |

Features for approximating Polyhedra join

| # | Description | Calculation |
|---|---|---|
| 1 | Number of variables in the block $\mathcal{X}^c_\sqcup$ | $\lvert\mathcal{X}^c_\sqcup\rvert$ |
| 2 | Number of constraints in the factor for $\mathcal{X}^c_\sqcup$ | $\lvert\mathcal{I}_\sqcup(\mathcal{X}^c_\sqcup)\rvert$ |
| 3 | Number of loop head variables in $c$ | $\lvert\mathcal{X}^c \cap \mathcal{X}_H\rvert$ |
| 4 | Boolean, true if $c$ is in $\mathcal{I}_H$ | $c \in \mathcal{I}_H$ |
| 5 | Score of variable $x_i$ | SCORE($x_i$) |
| 6 | Score of variable $x_j$ | SCORE($x_j$) |
| 7 | Number of finite bounds for variable $x_i$ | See text |
| 8 | Number of finite bounds for variable $x_j$ | See text |
| 9 | Absolute value of constraint upper bound $b$ | $\lvert b\rvert$ |
| 10 | Boolean, true if upper bound $b'$ is $\infty$ | See text |
| 11 | Number of constraints coarsening partition | See text |

Features for approximating Octagon join

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Class A (Signature features) | | | | | | | | | |
| A1 | "java" | A2 | "lang" | A3 | "sun" | A4 | "()" | A5 | "void" |
| A6 | "security" | A7 | "int" | A8 | "util" | A9 | "String" | A10 | "init" |

| | | | |
|---|---|---|---|
| Class B (Additional features) | | | |
| B1 | Methods contained in nested class | B7 | Methods containing static method invocation |
| B2 | Methods taking multiple arguments | B8 | Methods containing virtual method invocation |
| B3 | Methods containing array load | B9 | Static method |
| B4 | Methods containing local assignments | B10 | Methods containing a single heap allocation |

Features for context tunneling heuristics

...

# Previous Data-driven Program Analysis

- Prior data-driven program analyses require application specific features

| Type | # | Features |
|---|---|---|
| A | 1 | local variable |
| | 2 | global variable |
| | 3 | structure field |
| | 4 | location created by dynamic memory allocation |
| | 5 | defined at one program point |
| | 6 | location potentially generated in library code |
| | 7 | assigned a constant expression (e.g., x = c1 + c2) |
| | 8 | compared with a constant expression (e.g., x < c) |
| | 9 | compared with an other variable (e.g., x < y) |
| | 10 | negated in a conditional expression (e.g., if (!x)) |
| | 11 | directly used in malloc (e.g., malloc(x)) |
| | 12 | indirectly used in malloc (e.g., y = x; malloc(y)) |
| | 13 | directly used in realloc (e.g., realloc(x)) |
| | 14 | indirectly used in realloc (e.g., y = x; realloc(y)) |
| | 15 | directly returned from malloc (e.g., x = malloc(e)) |
| | 16 | indirectly returned from malloc |
| | 17 | directly returned from realloc (e.g., x = realloc(e)) |
| | 18 | indirectly returned from realloc |
| | 19 | incremented by one (e.g., x = x + 1) |
| | 20 | incremented by a constant expr. (e.g., x = x + (1+2)) |
| | 21 | incremented by a variable (e.g., x = x + y) |
| | 22 | decremented by one (e.g., x = x - 1) |
| | 23 | decremented by a constant expr (e.g., x = x - (1+2)) |
| | 24 | decremented by a variable (e.g., x = x - y) |
| | 25 | multiplied by a constant (e.g., x = x * 2) |
| | 26 | multiplied by a variable (e.g., x = x * y) |
| | 27 | incremented pointer (e.g., p++) |
| | 28 | used as an array index (e.g., a[x]) |
| | 29 | used in an array expr. (e.g., x[e]) |
| | 30 | returned from an unknown library function |
| | 31 | modified inside a recursive function |
| | 32 | modified inside a local loop |
| | 33 | read inside a local loop |
| B | 34 | $1 \wedge 8 \wedge (11 \vee 12)$ |
| | 35 | $2 \wedge 8 \wedge (11 \vee 12)$ |
| | 36 | $1 \wedge (11 \vee 12) \wedge (19 \vee 20)$ |
| | 37 | $2 \wedge (11 \vee 12) \wedge (19 \vee 20)$ |
| | 38 | $1 \wedge (11 \vee 12) \wedge (15 \vee 16)$ |

Features for flow sensitivity heuristics

| Type | # | Features |
|---|---|---|
| A | 1 | leaf function |
| | 2 | function containing malloc |
| | 3 | function containing realloc |
| | 4 | function containing a loop |
| | 5 | function containing an if statement |
| | 6 | function containing a switch statement |
| | 7 | function using a string-related library function |
| | 8 | write to a global variable |
| | 9 | read a global variable |
| | 10 | write to a structure field |
| | 11 | read from a structure field |
| | 12 | directly return a constant expression |
| | 13 | indirectly return a constant expression |
| | 14 | directly return an allocated memory |
| | 15 | indirectly return an allocated memory |
| | 16 | directly return a reallocated memory |
| | 17 | indirectly return a reallocated memory |
| | 18 | return expression involves field access |
| | 19 | return value depends on a structure field |
| | 20 | return void |
| | 21 | directly invoked with a constant |
| | 22 | constant is passed to an argument |
| | 23 | invoked with an unknown value |
| | 24 | functions having no arguments |
| | 25 | functions having one argument |
| | 26 | functions having more than one argument |
| | 27 | functions having an integer argument |
| | 28 | functions having a pointer argument |
| | 29 | functions having a structure as an argument |
| B | 30 | $2 \wedge (21 \vee 22) \wedge (14 \vee 15)$ |
| | 31 | $2 \wedge (21 \vee 22) \wedge \neg(14 \vee 15)$ |
| | 32 | $2 \wedge 23 \wedge (14 \vee 15)$ |
| | 33 | $2 \wedge 23 \wedge \neg(14 \vee 15)$ |
| | 34 | $2 \wedge (21 \vee 22) \wedge (16 \vee 17)$ |
| | 35 | $2 \wedge (21 \vee 22) \wedge \neg(16 \vee 17)$ |
| | 36 | $2 \wedge 23 \wedge (16 \vee 17)$ |
| | 37 | $2 \wedge 23 \wedge \neg(16 \vee 17)$ |
| | 38 | $(21 \vee 22) \wedge \neg 23$ |

Features for context sensitivity heuristics

| # | Description | Calculation |
|---|---|---|
| 1 | Number of variables in the block $\mathcal{X}^c_\sqcup$ | $\lvert\mathcal{X}^c_\sqcup\rvert$ |
| 2 | Number of constraints in the factor $\mathcal{X}^c_\sqcup$ | $\lvert\mathcal{I}_\sqcup(\mathcal{X}^c_\sqcup)\rvert$ |
| 3 | Number of generators in the factor $\mathcal{X}^c_\sqcup$ | $\lvert\mathcal{G}_\sqcup(\mathcal{X}^c_\sqcup)\rvert$ |
| 4 | Number of loop head variables in $c$ | $\lvert\mathcal{X}^c \cap \mathcal{X}_H\rvert$ |
| 5 | Boolean, true if $\mathcal{X}^c$ is a subset of $\mathcal{X}_H$ | $\mathcal{X}^c \subseteq \mathcal{X}_H$ |
| 6 | Boolean, true if $c$ is in $\mathcal{I}_H$ | $c \in \mathcal{I}_H$ |
| 7 | Number of variables in constraint $c$ | $\lvert\mathcal{X}^c\rvert$. |
| 8 | Number of large coefficients in $c$ | $\sum_i(\lvert a_i\rvert >= 100)$ |
| 9 | Sum of scores for variables in $c$ | $\sum_i \textsc{Score}(x_i)$ |
| 10 | Boolean, true if $c$ is in join inputs | $c \in \mathcal{I}_P \wedge c \in \mathcal{I}_Q$ |
| 11 | Boolean, true if $c$ coarsens partition | $\textsc{Coarse}(\mathcal{X}^c_\sqcup, \overline{\pi}_P, \overline{\pi}_Q)$ |
| 12 | Boolean, true if $c$ is an equality | $\circ$ is $=$ |

Features for approximating Polyhedra join

| # | Description | Calculation |
|---|---|---|
| 1 | Number of variables in the block $\mathcal{X}^c_\sqcup$ | $\lvert\mathcal{X}^c_\sqcup\rvert$ |
| 2 | Number of constraints in the factor for $\mathcal{X}^c_\sqcup$ | $\lvert\mathcal{I}_\sqcup(\mathcal{X}^c_\sqcup)\rvert$ |
| 3 | Number of loop head variables in $c$ | $\lvert\mathcal{X}^c \cap \mathcal{X}_H\rvert$ |
| 4 | Boolean, true if $c$ is in $\mathcal{I}_H$ | $c \in \mathcal{I}_H$ |
| 5 | Score of variable $x_i$ | $\textsc{Score}(x_i)$ |
| 6 | Score of variable $x_j$ | $\textsc{Score}(x_j)$ |
| 7 | Number of finite bounds for variable $x_i$ | See text |
| 8 | Number of finite bounds for variable $x_j$ | See text |
| 9 | Absolute value of constraint upper bound $b$ | $\lvert b\rvert$ |
| 10 | Boolean, true if upper bound $b'$ is $\infty$ | See text |
| 11 | Number of constraints coarsening partition | See text |
| 12 | Loop iteration number | $iter$ |

Features for approximating Octagon join

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Class A (Signature features) | | | | | | | | | |
| A1 | "java" | A2 | "lang" | A3 | "sun" | A4 | "()" | A5 | "void" |
| A6 | "security" | A7 | "int" | A8 | "util" | A9 | "String" | A10 | "init" |

| | | | |
|---|---|---|---|
| Class B (Additional features) | | | |
| B1 | Methods contained in nested class | B7 | Methods containing static method invocation |
| B2 | Methods taking multiple arguments | B8 | Methods containing virtual method invocation |
| B3 | Methods containing array load | B9 | Static method |
| B4 | Methods containing local assignments | B10 | Methods containing a single heap allocation |
| B5 | Methods containing local variables | B11 | Methods taking an argument of Object type |
| B6 | Methods containing field store | B12 | Methods containing multiple heap allocations |
| | | B13 | Methods contained in a large class |

Features for context tunneling heuristics

...

# Previous Data-driven Program Analysis

- Prior data-driven program analyses require application specific features

| Type | # | Features |
|---|---|---|
| A | 1 | local variable |
| | 2 | global variable |
| | 3 | structure field |
| | 4 | location created by dynamic memory allocation |
| | 5 | defined at one program point |
| | 6 | location potentially generated in library code |
| | 7 | assigned a constant expression (e.g., x = c1 + c2) |
| | 8 | compared with a constant expression (e.g., x < c) |
| | 9 | compared with an other variable (e.g., x < y) |
| | 10 | negated in a conditional expression (e.g., if (!x)) |
| | 11 | directly used in malloc (e.g., malloc(x)) |
| | 12 | indirectly used in malloc (e.g., y = x; malloc(y)) |
| | 13 | directly used in realloc (e.g., realloc(x)) |
| | 14 | indirectly used in realloc (e.g., y = x; realloc(y)) |
| | 15 | directly returned from malloc (e.g., x = malloc(e)) |
| | 16 | indirectly returned from malloc |
| | 17 | directly returned from realloc (e.g., x = realloc(e)) |
| | 18 | indirectly returned from realloc |
| | 19 | incremented by one (e.g., x = x + 1) |
| | 20 | incremented by a constant expr. (e.g., x = x + (1+2)) |
| | 21 | incremented by a variable (e.g., x = x + y) |
| | 22 | decremented by one (e.g., x = x - 1) |
| | 23 | decremented by a constant expr (e.g., x = x - (1+2)) |
| | 24 | decremented by a variable (e.g., x = x - y) |
| | 25 | multiplied by a constant (e.g., x = x * 2) |
| | 26 | multiplied by a variable (e.g., x = x * y) |
| | 27 | incremented pointer (e.g., p++) |
| | 28 | used as an array index (e.g., a[x]) |
| | 29 | used in an array expr. (e.g., x[e]) |
| | 30 | returned from an unknown library function |
| | 31 | modified inside a recursive function |
| | 32 | modified inside a local loop |
| | 33 | read inside a local loop |
| B | 34 | $1 \wedge 8 \wedge (11 \vee 12)$ |
| | 35 | $2 \wedge 8 \wedge (11 \vee 12)$ |
| | 36 | $1 \wedge (11 \vee 12) \wedge (19 \vee 20)$ |
| | 37 | $2 \wedge (11 \vee 12) \wedge (19 \vee 20)$ |
| | 38 | $1 \wedge (11 \vee 12) \wedge (15 \vee 16)$ |
| | 39 | $2 \wedge (11 \vee 12) \wedge (15 \vee 16)$ |
| | 40 | $(11 \vee 12) \wedge 29$ |
| | 41 | $(15 \vee 16) \wedge 29$ |
| | 42 | $1 \wedge (19 \vee 20) \wedge 33$ |
| | 43 | $2 \wedge (19 \vee 20) \wedge 33$ |
| | 44 | $1 \wedge (19 \vee 20) \wedge \neg 33$ |
| | 45 | $2 \wedge (19 \vee 20) \wedge \neg 33$ |

Features for flow sensitivity heuristics

| Type | # | Features |
|---|---|---|
| A | 1 | leaf function |
| | 2 | function containing malloc |
| | 3 | function containing realloc |
| | 4 | function containing a loop |
| | 5 | function containing an if statement |
| | 6 | function containing a switch statement |
| | 7 | function using a string-related library function |
| | 8 | write to a global variable |
| | 9 | read a global variable |
| | 10 | write to a structure field |
| | 11 | read from a structure field |
| | 12 | directly return a constant expression |
| | 13 | indirectly return a constant expression |
| | 14 | directly return an allocated memory |
| | 15 | indirectly return an allocated memory |
| | 16 | directly return a reallocated memory |
| | 17 | indirectly return a reallocated memory |
| | 18 | return expression involves field access |
| | 19 | return value depends on a structure field |
| | 20 | return void |
| | 21 | directly invoked with a constant |
| | 22 | constant is passed to an argument |
| | 23 | invoked with an unknown value |
| | 24 | functions having no arguments |
| | 25 | functions having one argument |
| | 26 | functions having more than one argument |
| | 27 | functions having an integer argument |
| | 28 | functions having a pointer argument |
| | 29 | functions having a structure as an argument |
| B | 30 | $2 \wedge (21 \vee 22) \wedge (14 \vee 15)$ |
| | 31 | $2 \wedge (21 \vee 22) \wedge \neg(14 \vee 15)$ |
| | 32 | $2 \wedge 23 \wedge (14 \vee 15)$ |

Features for context sensitivity heuristics

| # | Description | Calculation |
|---|---|---|
| 1 | Number of variables in the block $\mathcal{X}^c_\sqcup$ | $\lvert\mathcal{X}^c_\sqcup\rvert$ |
| 2 | Number of constraints in the factor $\mathcal{X}^c_\sqcup$ | $\lvert\mathcal{I}_\sqcup(\mathcal{X}^c_\sqcup)\rvert$ |
| 3 | Number of generators in the factor $\mathcal{X}^c_\sqcup$ | $\lvert\mathcal{G}_\sqcup(\mathcal{X}^c_\sqcup)\rvert$ |
| 4 | Number of loop head variables in $c$ | $\lvert\mathcal{X}^c \cap \mathcal{X}_H\rvert$ |
| 5 | Boolean, true if $\mathcal{X}^c$ is a subset of $\mathcal{X}_H$ | $\mathcal{X}^c \subseteq \mathcal{X}_H$ |
| 6 | Boolean, true if $c$ is in $\mathcal{I}_H$ | $c \in \mathcal{I}_H$ |
| 7 | Number of variables in constraint $c$ | $\lvert\mathcal{X}^c\rvert$. |
| 8 | Number of large coefficients in $c$ | $\sum_i(\lvert a_i\rvert >= 100)$ |
| 9 | Sum of scores for variables in $c$ | $\sum_i \text{SCORE}(x_i)$. |
| 10 | Boolean, true if $c$ is in join inputs | $c \in \mathcal{I}_P \wedge c \in \mathcal{I}_Q$ |
| 11 | Boolean, true if $c$ coarsens partition | $\text{COARSE}(\mathcal{X}^c_\sqcup, \overline{\pi}_P, \overline{\pi}_Q)$ |
| 12 | Boolean, true if $c$ is an equality | $\circ$ is $=$ |

Features for approximating Polyhedra join

| # | Description | Calculation |
|---|---|---|
| 1 | Number of variables in the block $\mathcal{X}^c_\sqcup$ | $\lvert\mathcal{X}^c_\sqcup\rvert$ |
| 2 | Number of constraints in the factor for $\mathcal{X}^c_\sqcup$ | $\lvert\mathcal{I}_\sqcup(\mathcal{X}^c_\sqcup)\rvert$ |
| 3 | Number of loop head variables in $c$ | $\lvert\mathcal{X}^c \cap \mathcal{X}_H\rvert$ |
| 4 | Boolean, true if $c$ is in $\mathcal{I}_H$ | $c \in \mathcal{I}_H$ |
| 5 | Score of variable $x_i$ | $\text{SCORE}(x_i)$ |
| 6 | Score of variable $x_j$ | $\text{SCORE}(x_j)$ |
| 7 | Number of finite bounds for variable $x_i$ | See text |
| 8 | Number of finite bounds for variable $x_j$ | See text |
| 9 | Absolute value of constraint upper bound $b$ | $\lvert b\rvert$ |
| 10 | Boolean, true if upper bound $b'$ is $\infty$ | See text |
| 11 | Number of constraints coarsening partition | See text |
| 12 | Loop iteration number | $iter$ |

Features for approximating Octagon join

| Class A (Signature features) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| A1 | "java" | A2 | "lang" | A3 | "sun" | A4 | "()" | A5 | "void" |
| A6 | "security" | A7 | "int" | A8 | "util" | A9 | "String" | A10 | "init" |

| Class B (Additional features) | | | |
|---|---|---|---|
| B1 | Methods contained in nested class | B7 | Methods containing static method invocation |
| B2 | Methods taking multiple arguments | B8 | Methods containing virtual method invocation |
| B3 | Methods containing array load | B9 | Static method |
| B4 | Methods containing local assignments | B10 | Methods containing a single heap allocation |
| B5 | Methods containing local variables | B11 | Methods taking an argument of Object type |
| B6 | Methods containing field store | B12 | Methods containing multiple heap allocations |
| | | B13 | Methods contained in a large class |

Features for context tunneling heuristics

...

# Previous Data-driven Program Analysis

- Prior data-driven program analyses require application specific features

Features for flow sensitivity heuristics

Features for context sensitivity heuristics

| # | Description | Calculation |
|---|---|---|
| 1 | Number of variables in the block $\mathcal{X}^c_\sqcup$ | $\lvert\mathcal{X}^c_\sqcup\rvert$ |
| 2 | Number of constraints in the factor $\mathcal{X}^c_\sqcup$ | $\lvert\mathcal{I}_\sqcup(\mathcal{X}^c_\sqcup)\rvert$ |
| 3 | Number of generators in the factor $\mathcal{X}^c_\sqcup$ | $\lvert\mathcal{G}_\sqcup(\mathcal{X}^c_\sqcup)\rvert$ |
| 4 | Number of loop head variables in $c$ | $\lvert\mathcal{X}^c \cap \mathcal{X}_H\rvert$ |
| 5 | Boolean, true if $\mathcal{X}^c$ is a subset of $\mathcal{X}_H$ | $\mathcal{X}^c \subseteq \mathcal{X}_H$ |
| 6 | Boolean, true if $c$ is in $\mathcal{I}_H$ | $c \in \mathcal{I}_H$ |
| 7 | Number of variables in constraint $c$ | $\lvert\mathcal{X}^c\rvert$ |
| 11 | Boolean, true if $c$ coarsens partition | COARSE |
| 12 | Boolean, true if $c$ is an equality | $\circ$ is $=$ |

Features for approximating Polyhedra join

| # | Description | Calculation |
|---|---|---|
| 1 | Number of variables in the block $\mathcal{X}^c_\sqcup$ | $\lvert\mathcal{X}^c_\sqcup\rvert$ |
| 2 | Number of constraints in the factor for $\mathcal{X}^c_\sqcup$ | $\lvert\mathcal{I}_\sqcup(\mathcal{X}^c_\sqcup)\rvert$ |
| 3 | Number of loop head variables in $c$ | $\lvert\mathcal{X}^c \cap \mathcal{X}_H\rvert$ |
| 4 | Boolean, true if $c$ is in $\mathcal{I}_H$ | $c \in \mathcal{I}_H$ |
| 5 | Score of variable $x_i$ | $\textsc{Score}(x_i)$ |
| 6 | Score of variable $x_j$ | $\textsc{Score}(x_j)$ |
| 7 | Number of finite bounds for variable $x_i$ | See text |
| 8 | Number of finite bounds for variable $x_j$ | See text |
| 9 | Absolute value of constraint upper bound $b$ | $\lvert b\rvert$ |
| 10 | Boolean, true if upper bound $b'$ is $\infty$ | See text |
| 11 | Number of constraints coarsening partition | See text |

Features for approximating Octagon join

| Class A (Signature features) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| A1 | "java" | A2 | "lang" | A3 | "sun" | A4 | "()" | A5 | "void" |
| A6 | "security" | A7 | "int" | A8 | "util" | A9 | "String" | A10 | "init" |

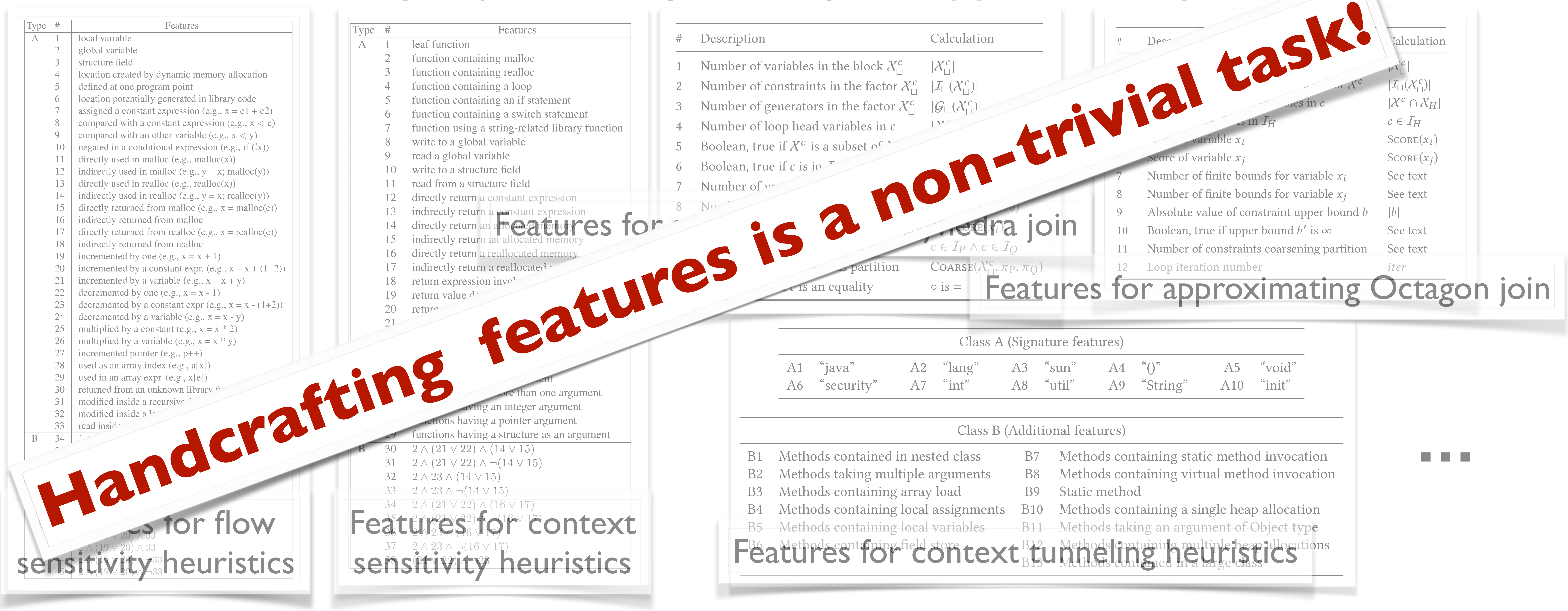
| Class B (Additional features) | | | |
|---|---|---|---|
| B1 | Methods contained in nested class | B7 | Methods containing static method invocation |
| B2 | Methods taking multiple arguments | B8 | Methods containing virtual method invocation |
| B3 | Methods containing array load | B9 | Static method |
| B4 | Methods containing local assignments | B10 | Methods containing a single heap allocation |

Features for context tunneling heuristics

...

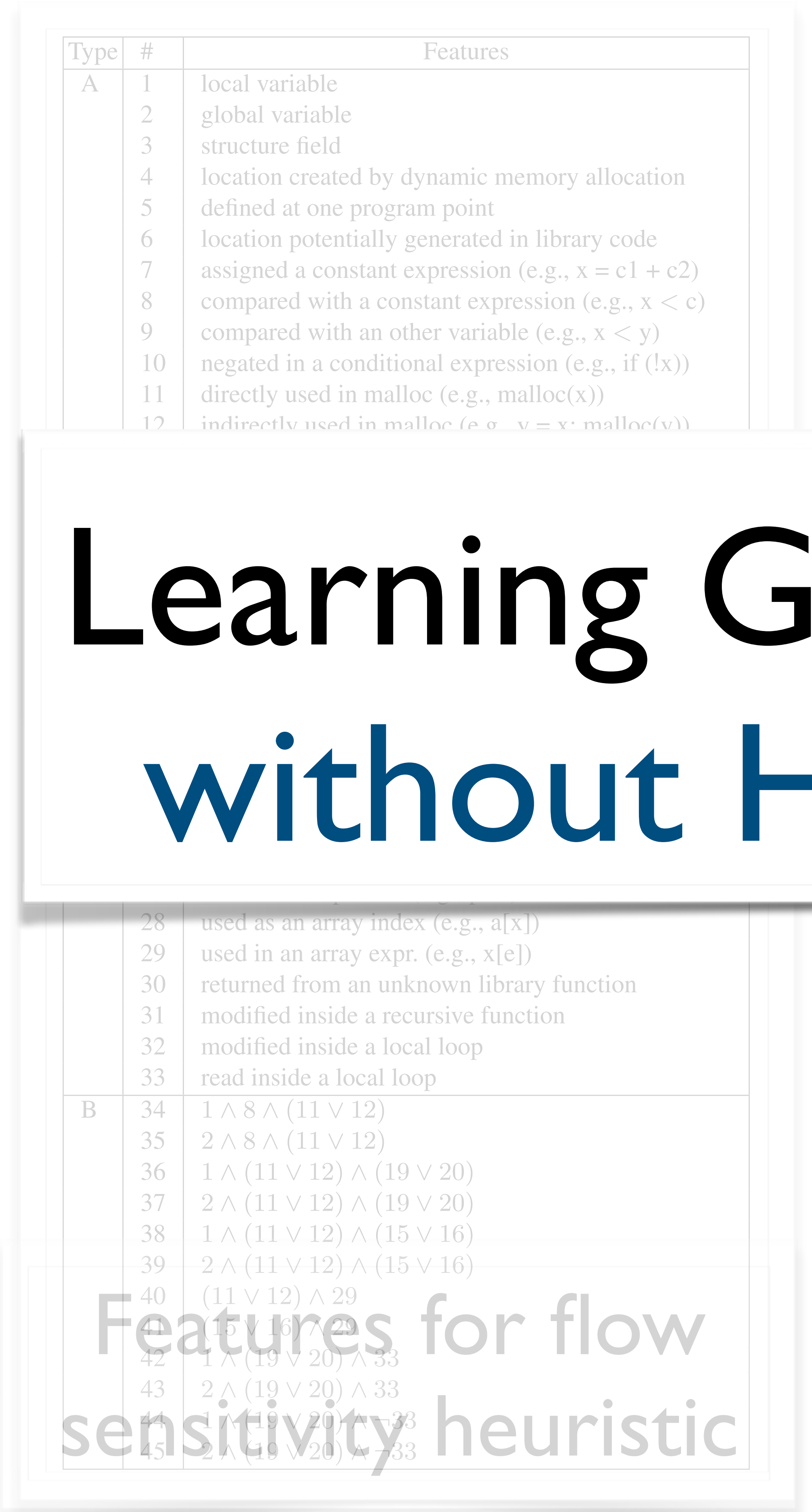
# Previous Data-driven Program Analysis

- Prior data-driven program analyses require application specific features



# Previous Data-driven Program Analysis

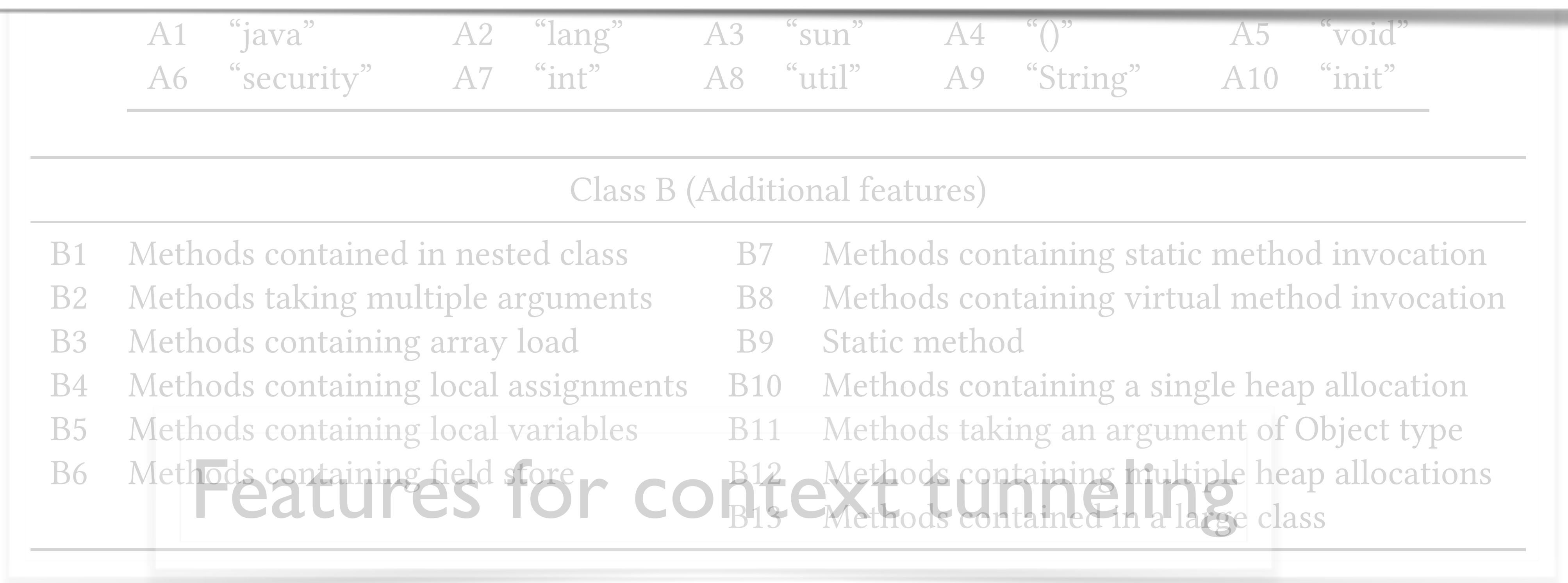
- Our technique does not require such application specific features












**Learning Graph-based Heuristics for Pointer Analysis without Handcrafting Application-Specific Features**

...

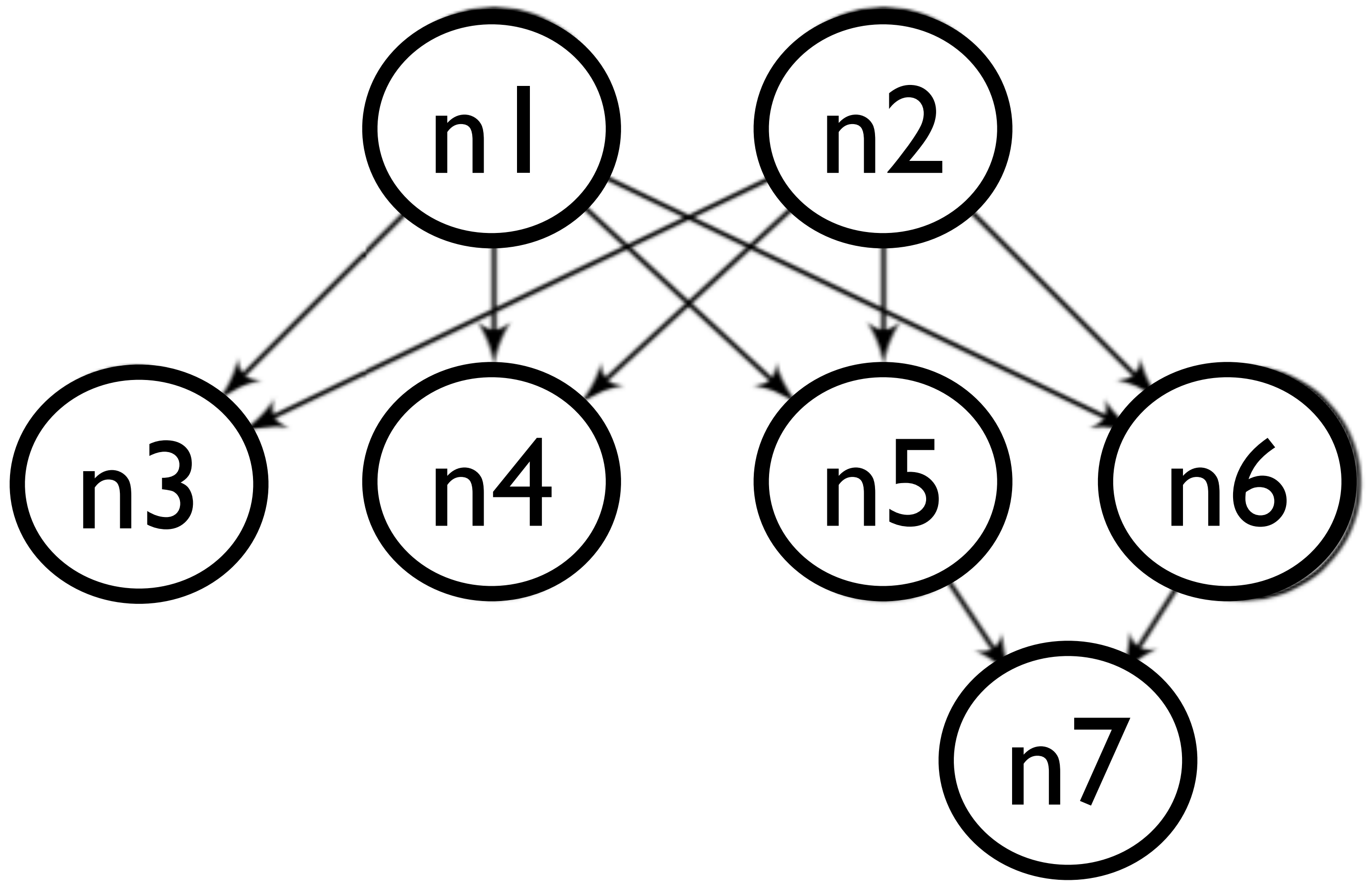
# Our Technique: Graphick


Automatically generated graph-based context sensitivity heuristic

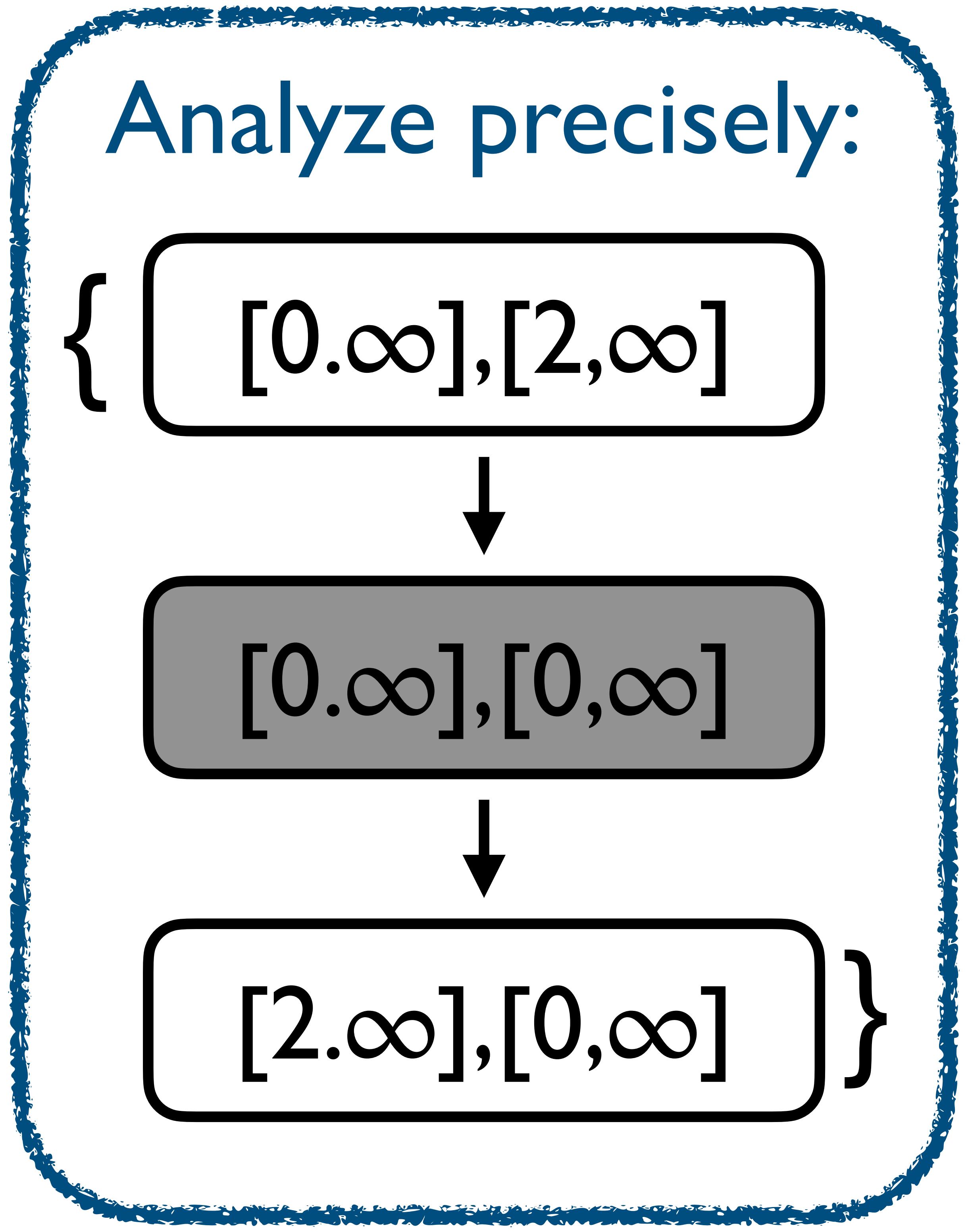
# How a Learned Heuristic Works

- A learned heuristics classifies nodes with features it contains

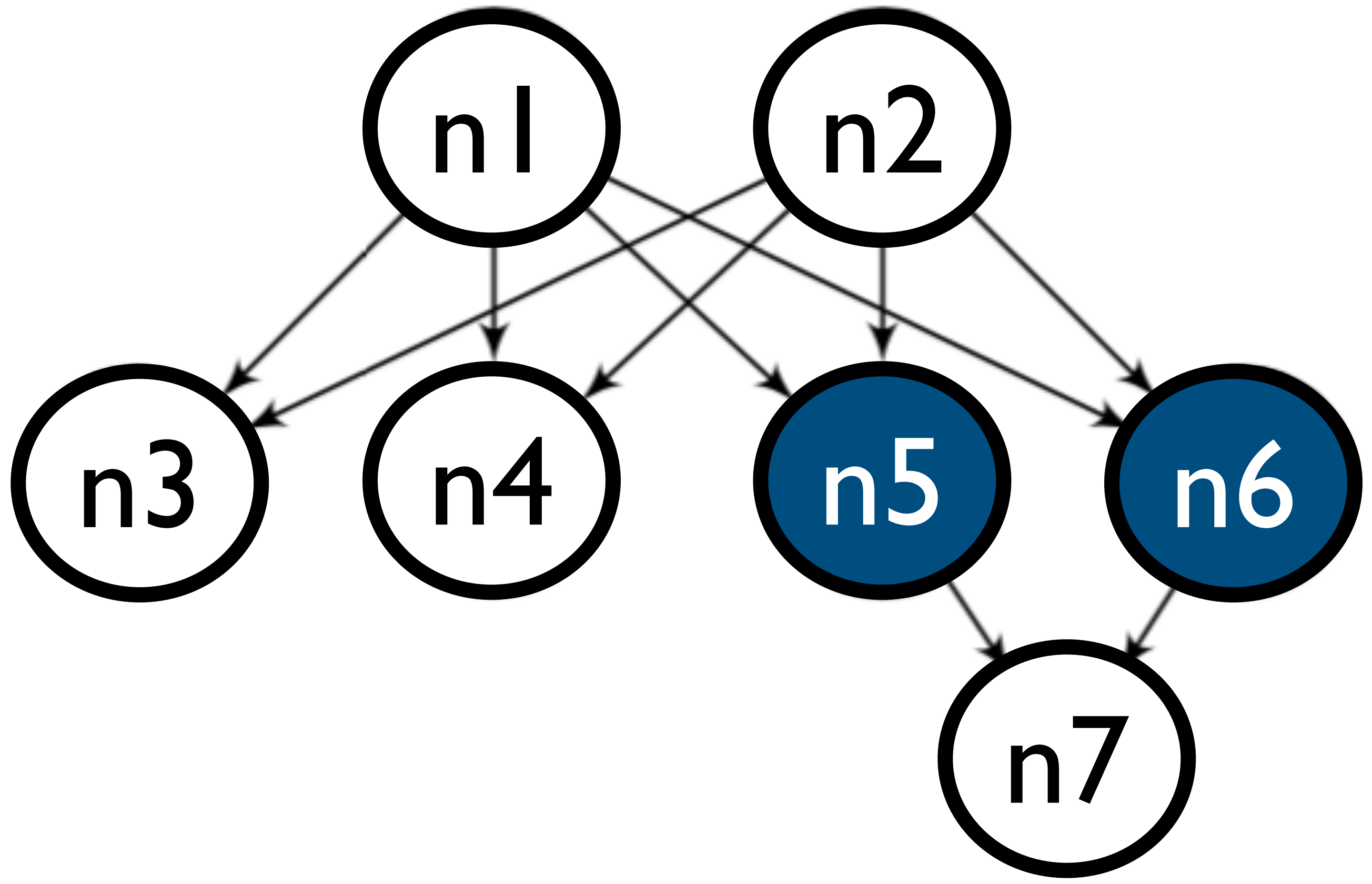
describe topological properties




Given graph




Heuristic example




Classified nodes

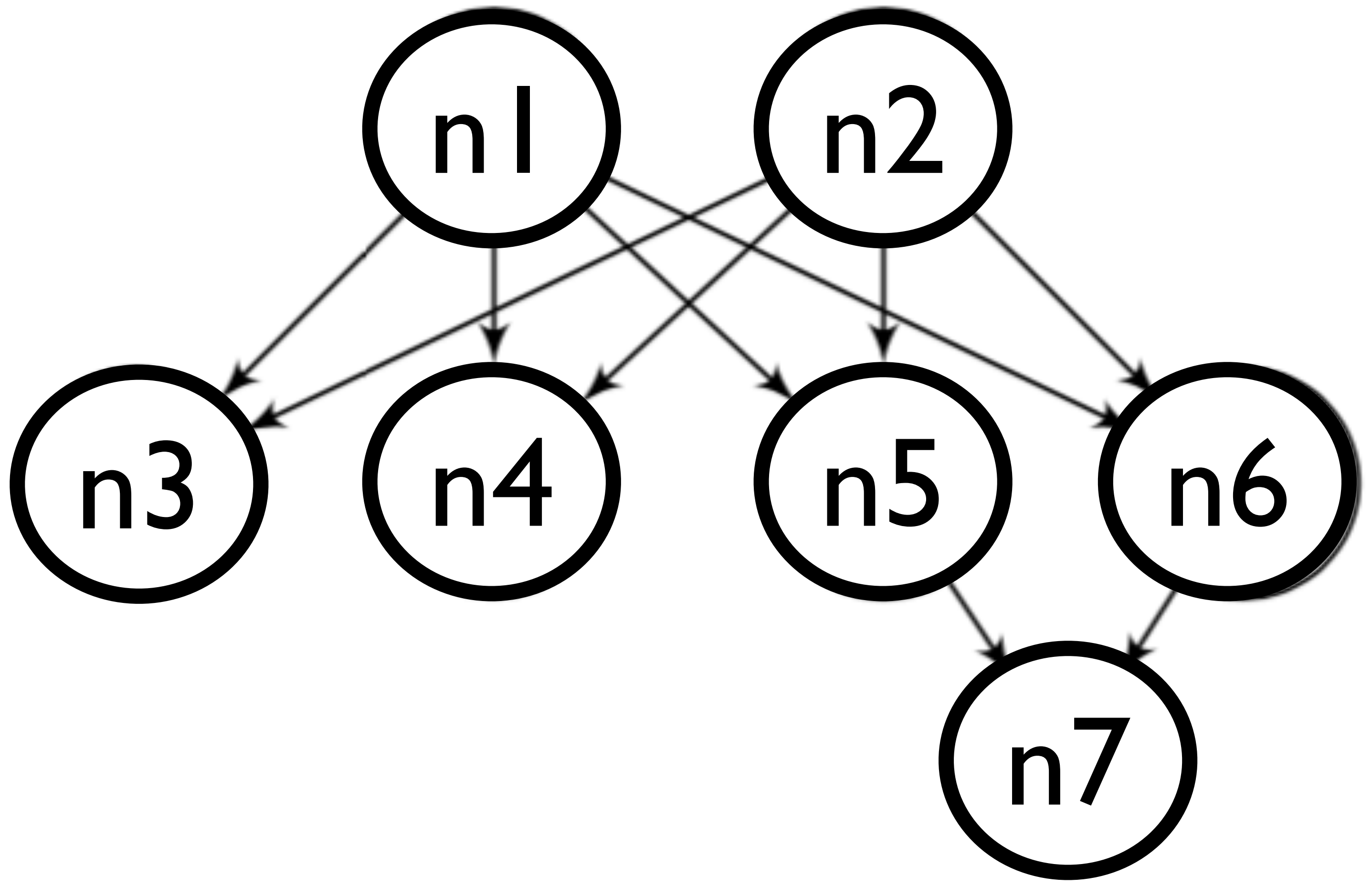
analyze precisely: n5, n6

analyze coarsely: others

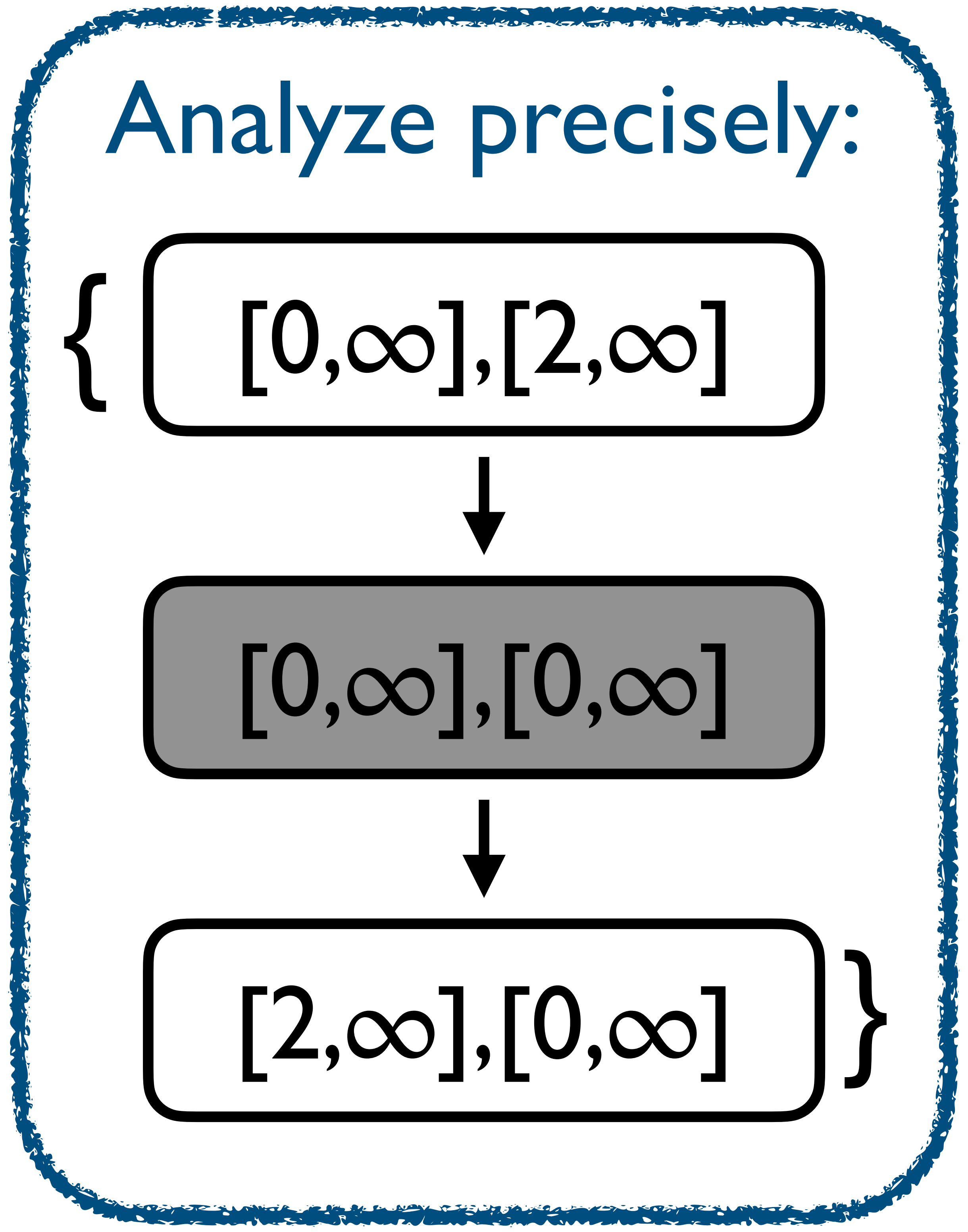
# How a Learned Heuristic Works

- A learned heuristics classifies nodes with features it contains

describe topological properties




Given graph




Heuristic example



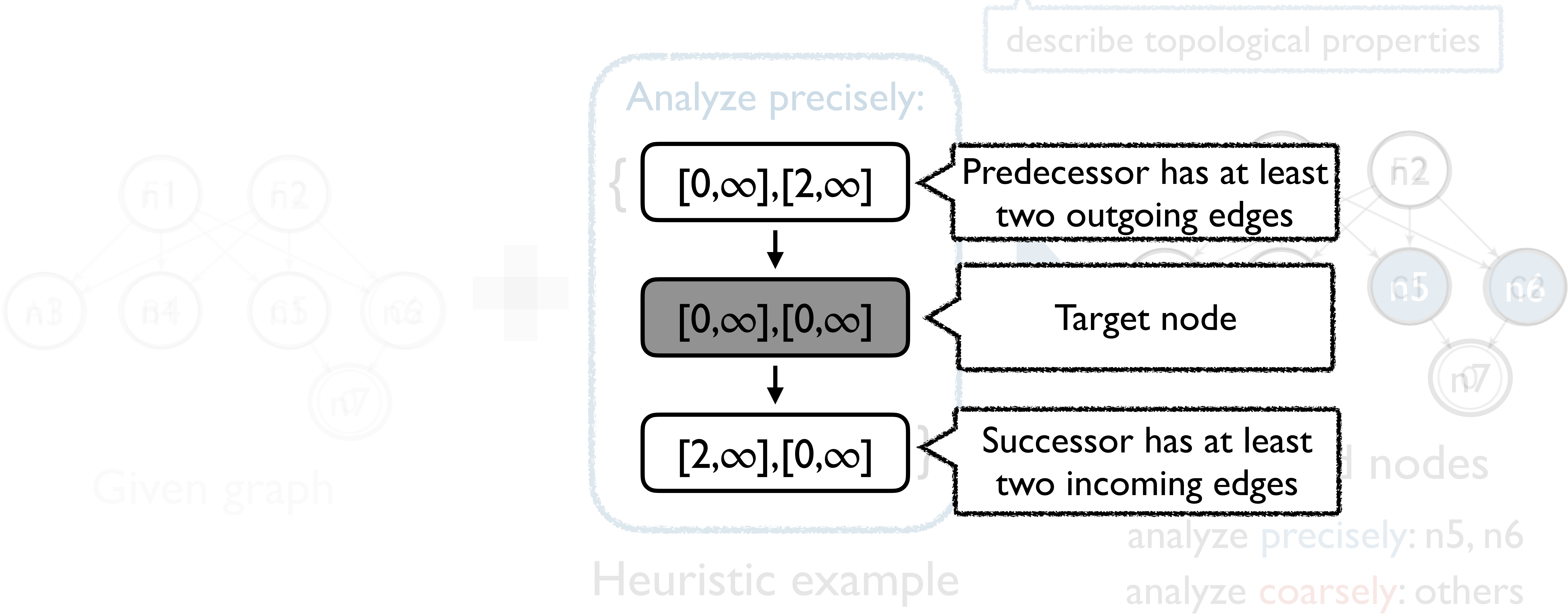

Classified nodes

analyze precisely: n5, n6

analyze coarsely: others

# How a Learned Heuristic Works

- A learned heuristics classifies nodes with features it contains



# How a Learned Heuristic Works

- A learned heuristics classifies nodes with features it contains

describe topological properties


Given graph

Heuristic example

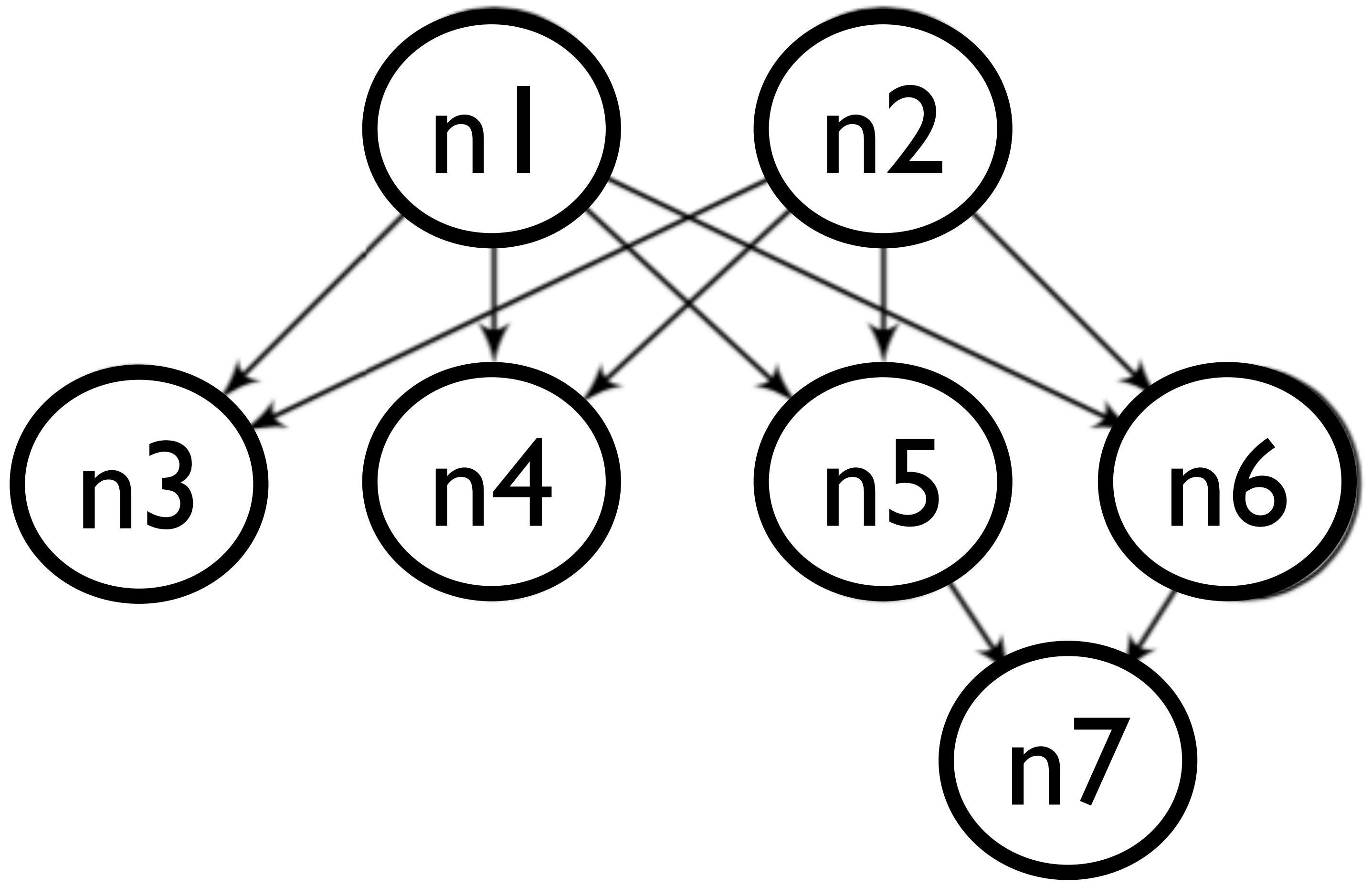
Classified nodes

analyze precisely: n5, n6

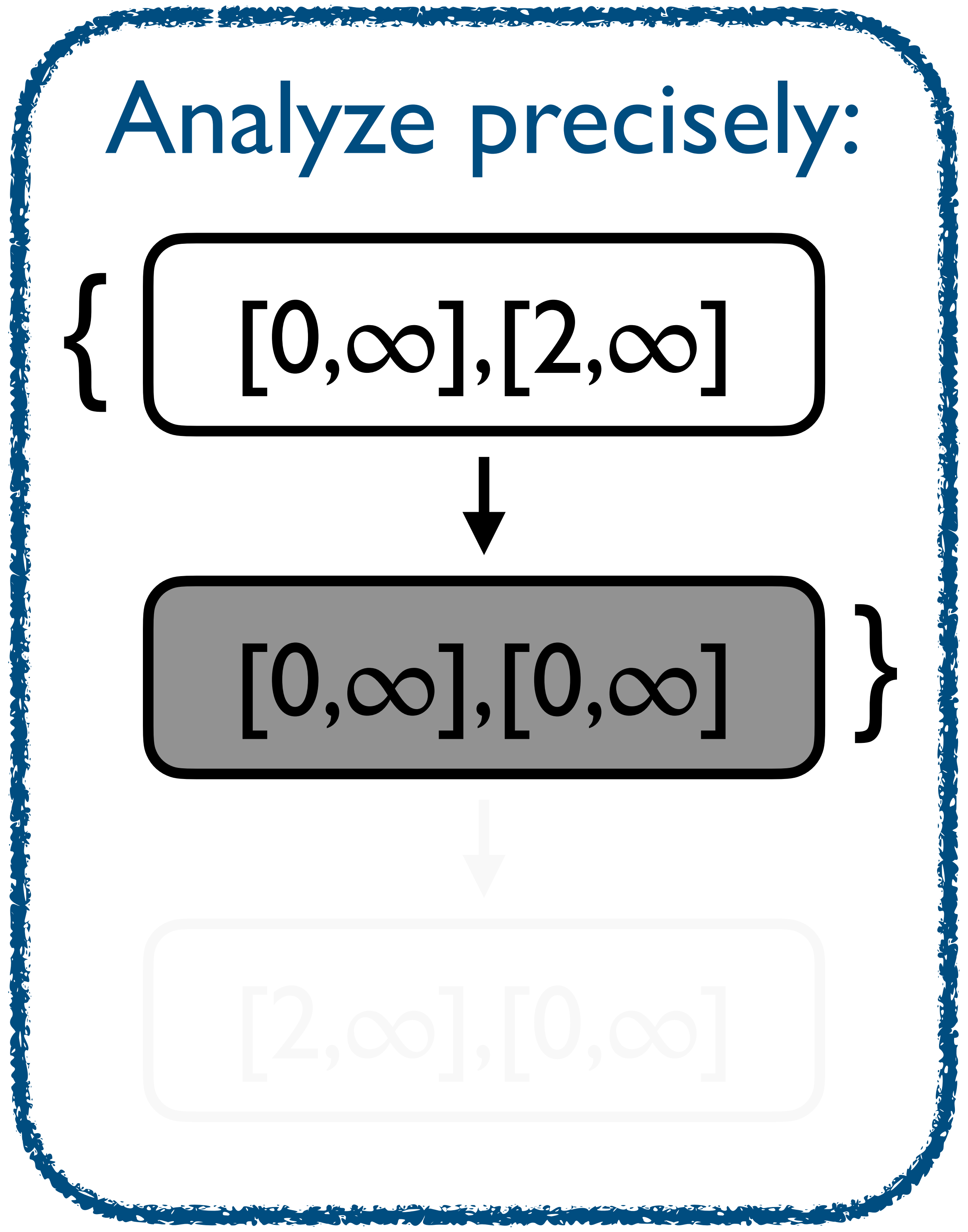
analyze coarsely: others

# How a Learned Heuristic Works

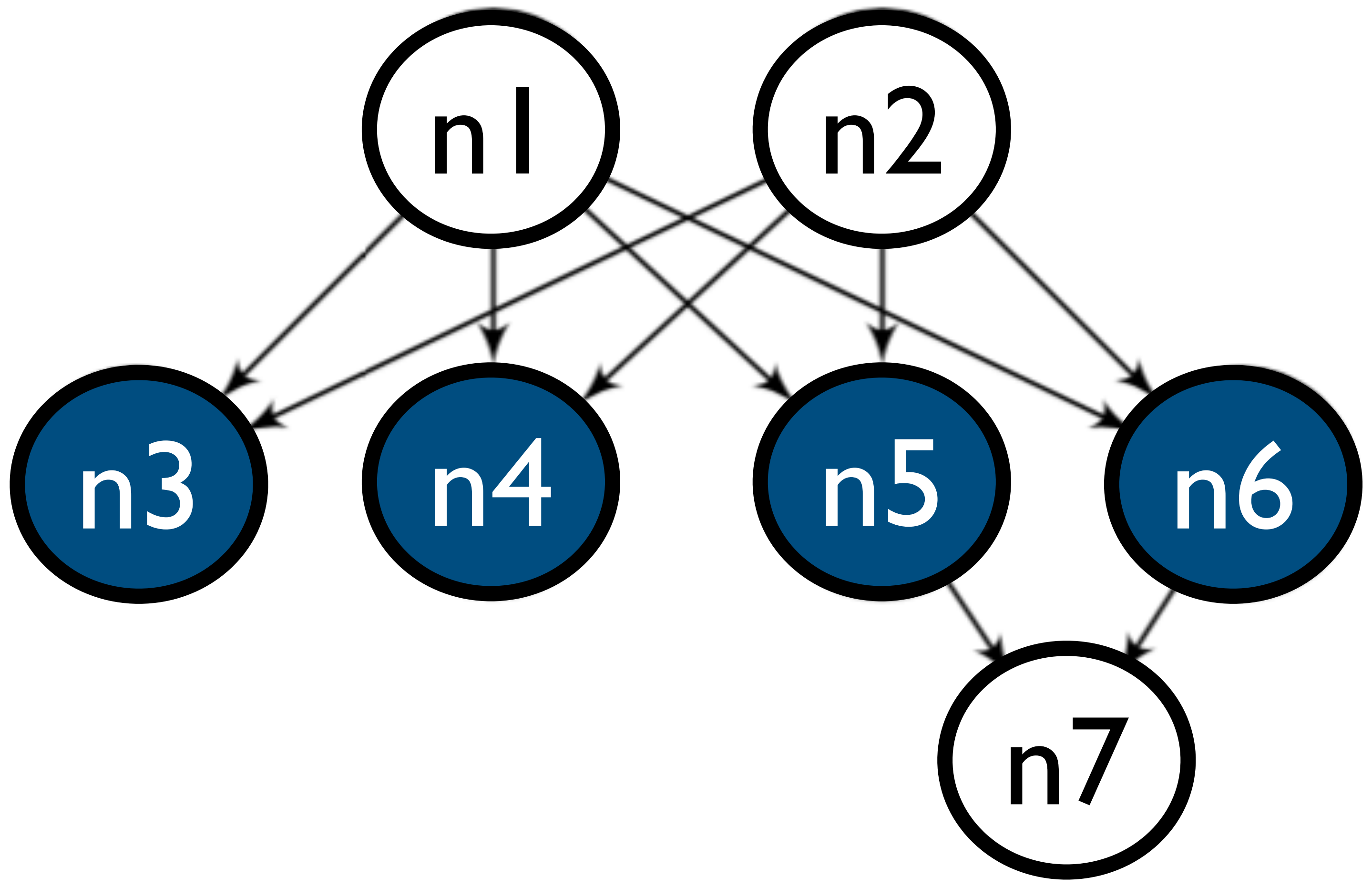
- Features in heuristic determine analysis performance




Given graph




Heuristic example



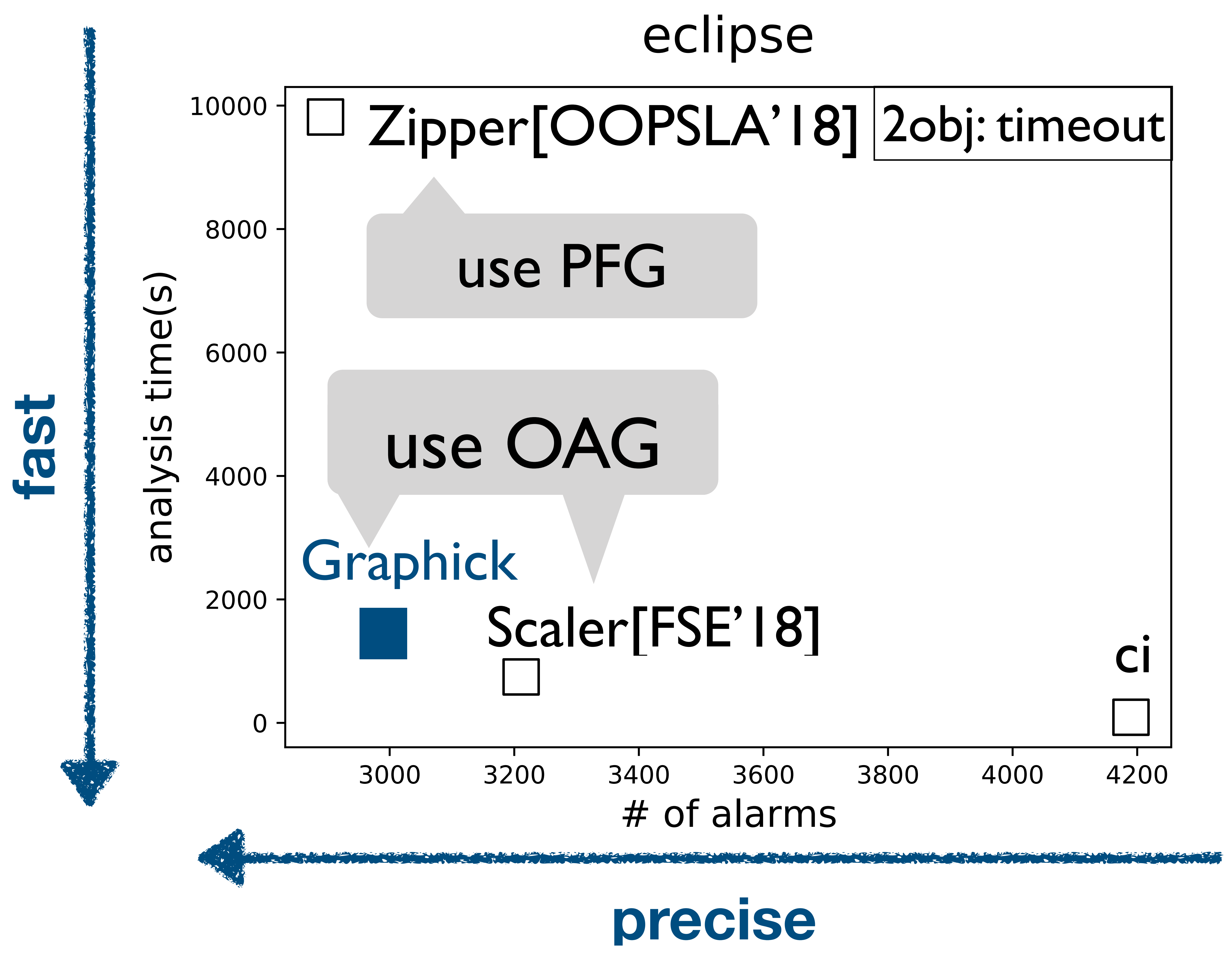

Classified nodes

analyze precisely: n3, n4, n5, n6

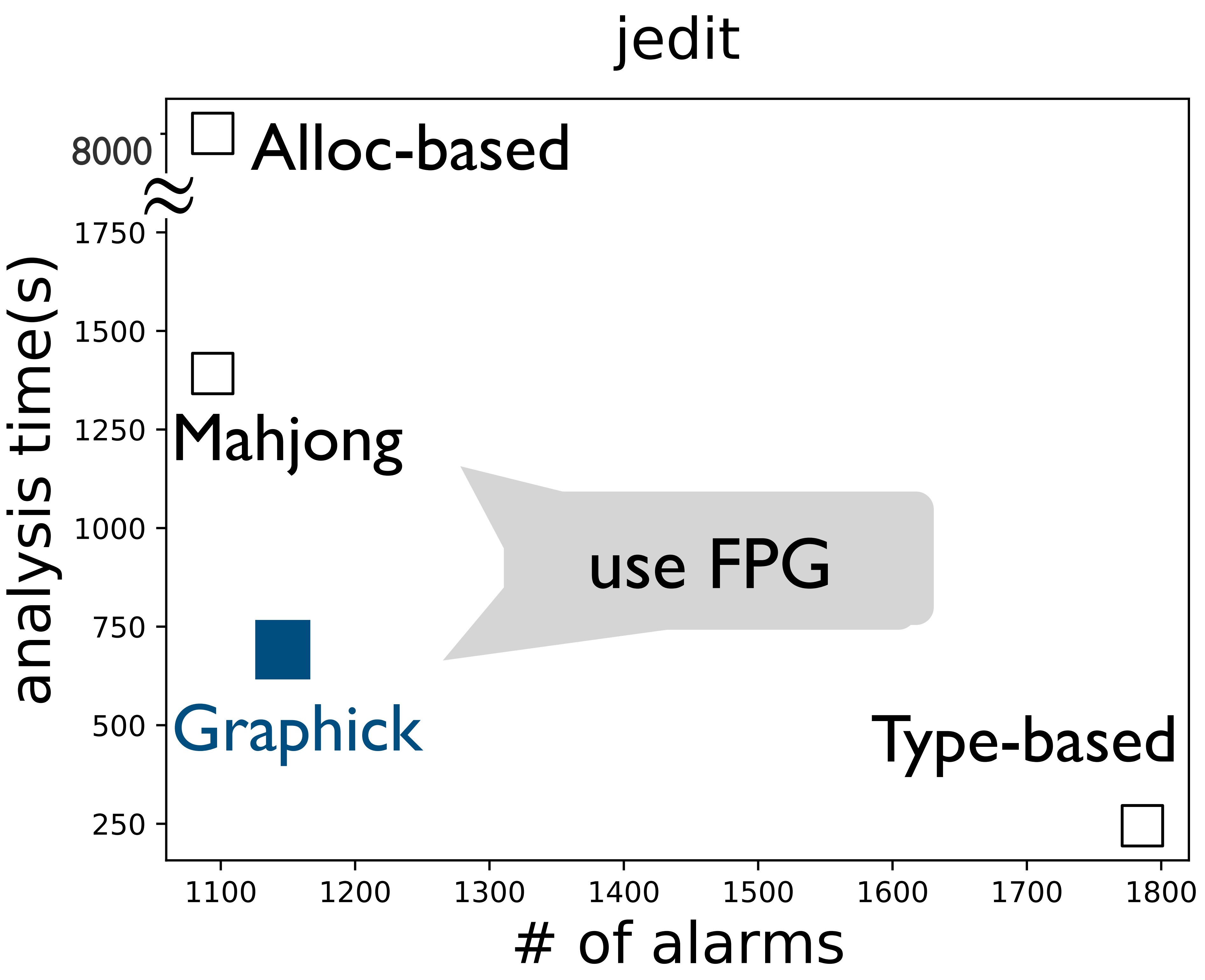
analyze coarsely: others

# Performance Highlight

- Graphick successfully produces context-sensitivity and heap abstraction heuristics



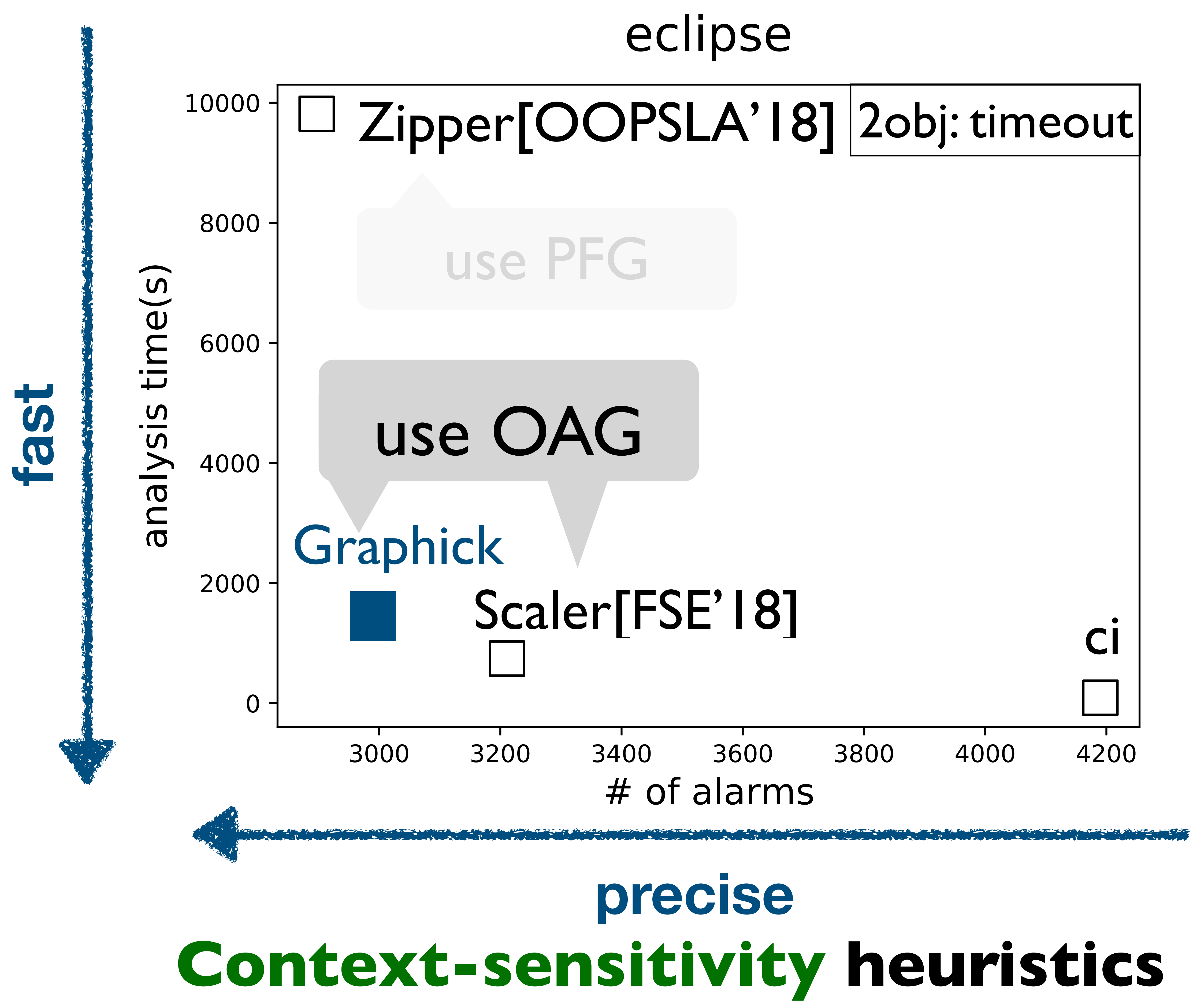

**Context-sensitivity heuristics**




**Heap abstraction heuristics**

# Performance Highlight

- Graphick successfully produces context-sensitivity and heap abstraction heuristics

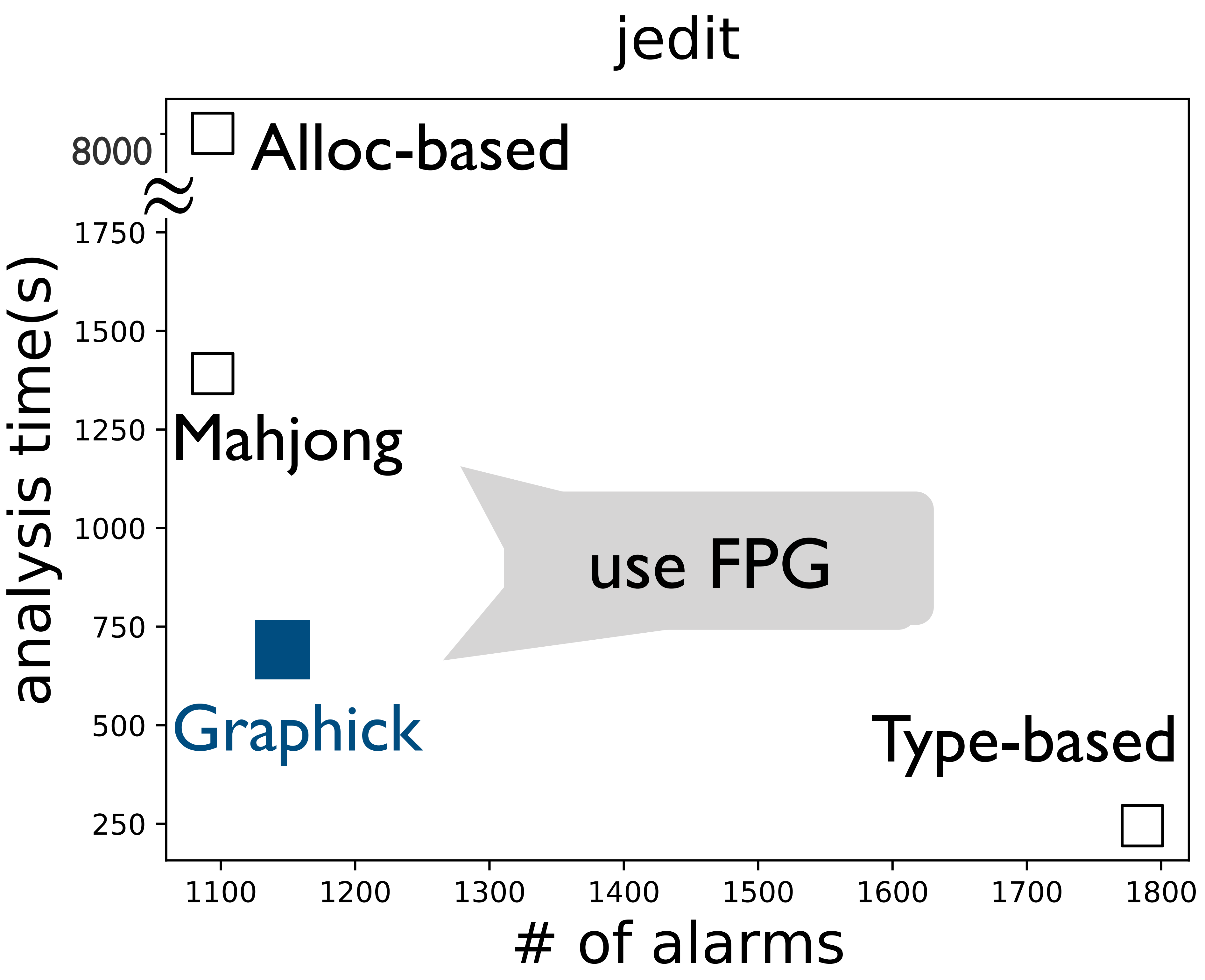


# Performance Highlight

- Graphick successfully produces context-sensitivity and heap abstraction heuristics



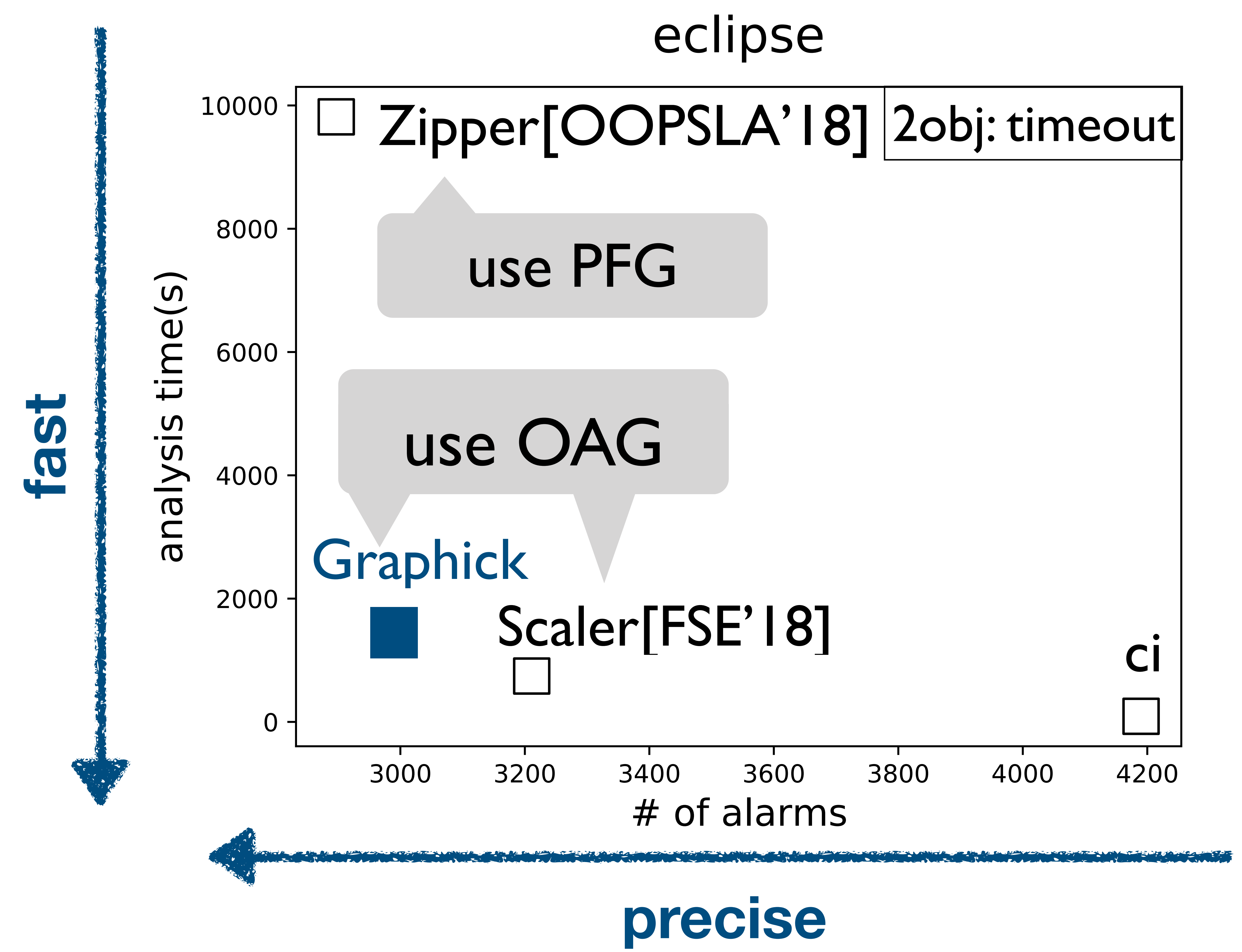

**Context-sensitivity heuristics**



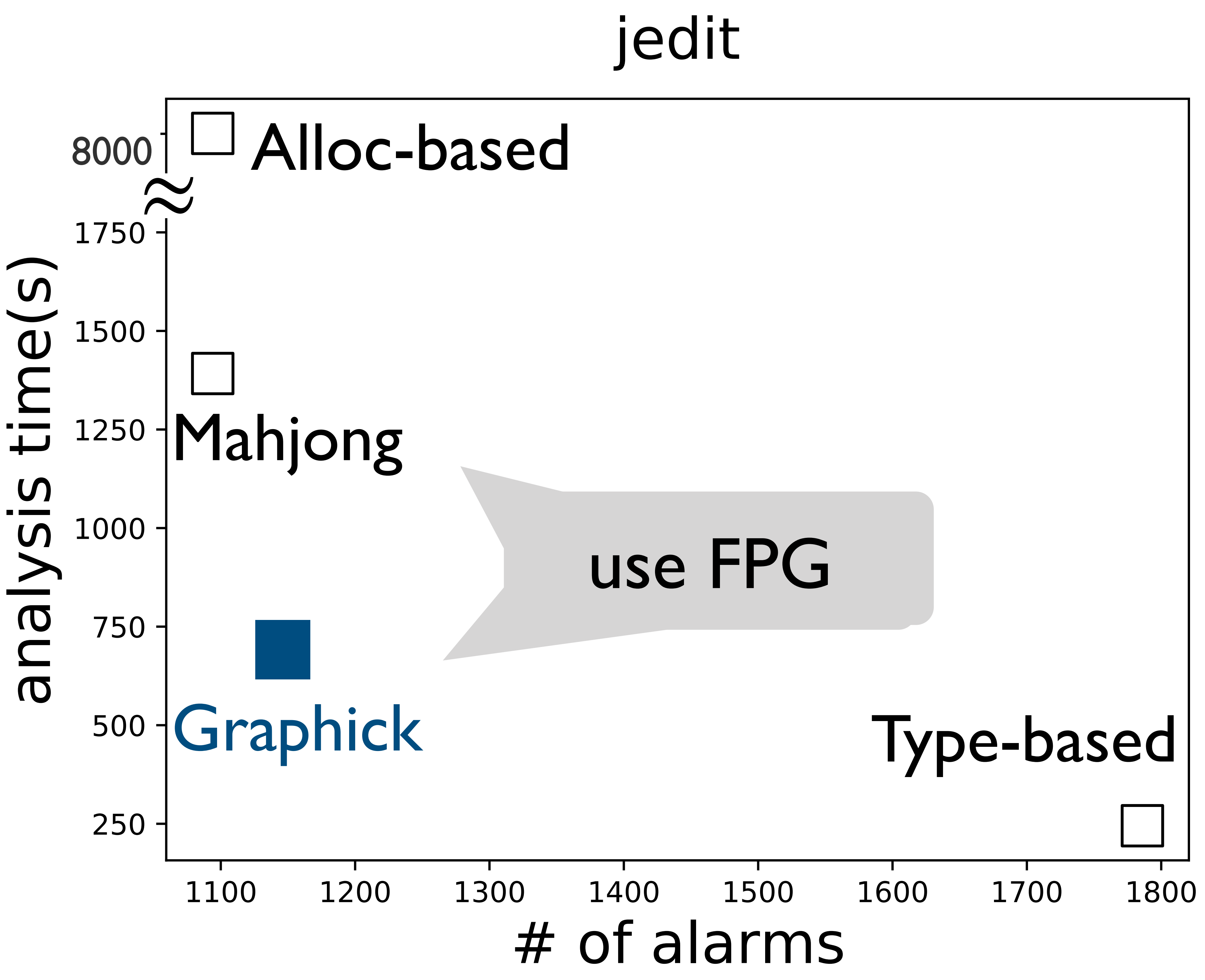

**Heap abstraction heuristics**

# Performance Highlight

- Graphick successfully produces context-sensitivity and heap abstraction heuristics




**Context-sensitivity heuristics**



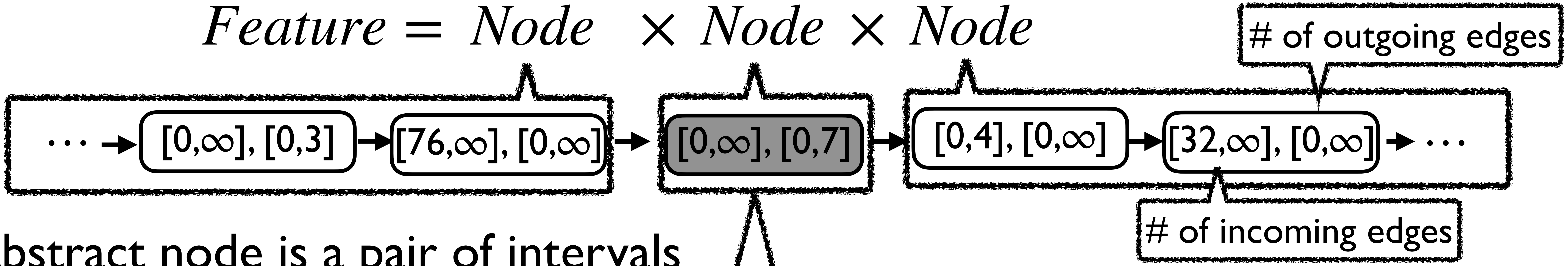

**Heap abstraction heuristics**

# Details

# Feature Description Language

- A feature is a list of abstract nodes

$$Feature = \widehat{Node}^{*} \times \widehat{Node} \times \widehat{Node}^{*}$$




- An abstract node is a pair of intervals

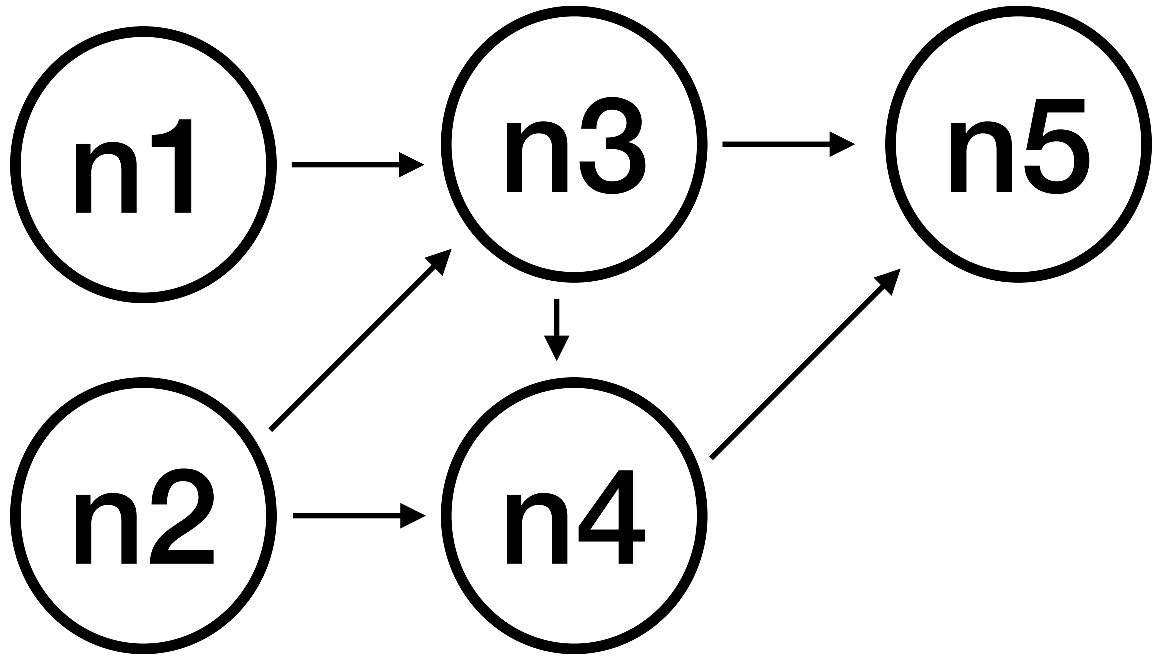
$$\widehat{Node} = Itv \times Itv$$

$$Itv = \{[a, b] \mid a \in \mathbb{N}, b \in \mathbb{N} \cup \infty,\}$$

# How a Feature Work

- A feature describes a topological property of nodes in graphs

[0, ∞], [2,∞] → [0,∞], [2, 2] → [2, 2], [0,∞] ✚




Nodes which have 2 incoming edges, and has a predecessor with 2 outgoing edges where the predecessor has a predecessor with at least 2 outgoing edges

=

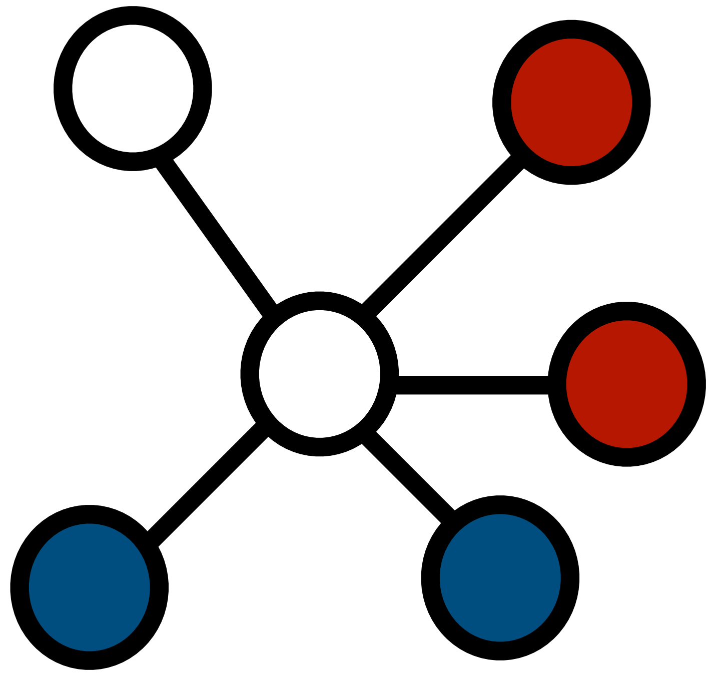
{n5}

# Our Technique: Graphick

# Learning Algorithm

- First, we find suitable labels for nodes in the graphs of training programs

**Graphs of training programs**



graphs with labeled nodes

Cost effective for the training programs

**Learning Minimal Abstractions**

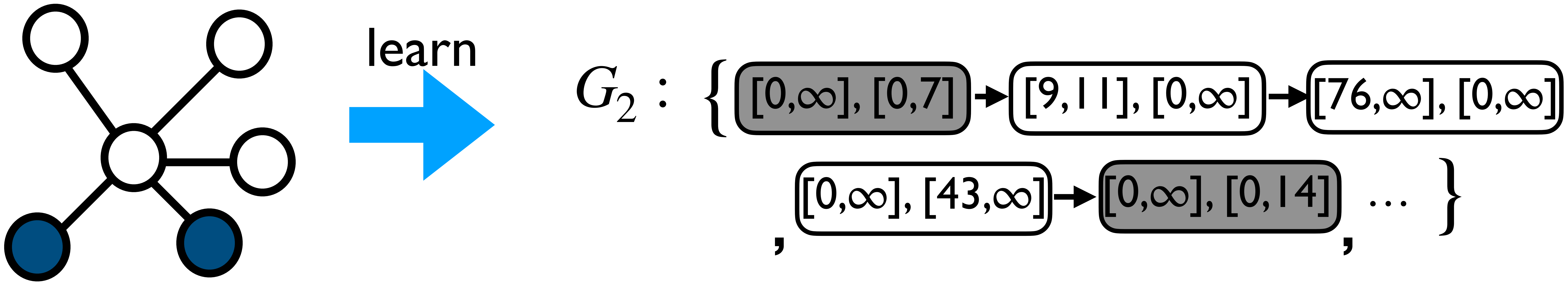
-Liang et al. [POPL 11]

# Learning Algorithm

- Nodes with each label are transformed into a set of features

# Learning a Set of Features

- Our algorithm transforms all the labeled node into a set of features




- To produce qualified features, we use the following score function:

$$\text{Score of a feature f} = \frac{\text{The number of labeled nodes chosen by f}}{\text{The number of nodes chosen by f}}$$

# Learning a Set of Features

- Our algorithm transforms all the labeled node into a set of features

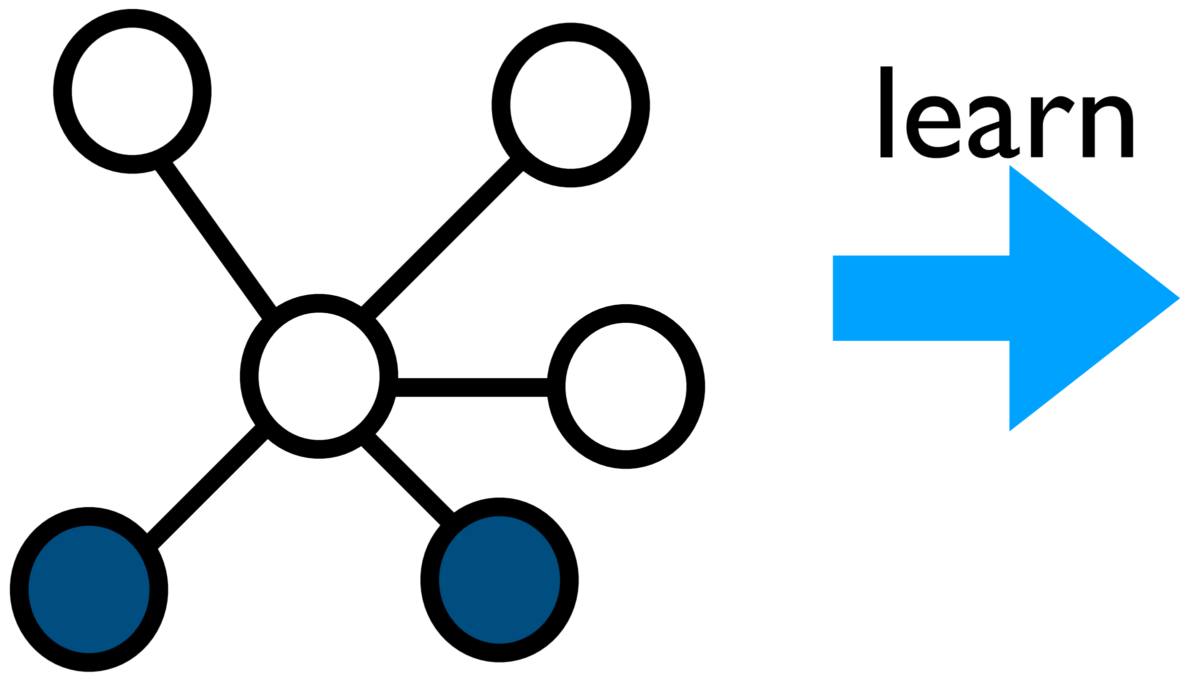



- To produce qualified features, we use the following score function:

$$\text{Score of a feature f} = \frac{\text{The number of labeled nodes chosen by f}}{\text{The number of nodes chosen by f}}$$

# Learning a Set of Features

- Our algorithm transforms all the labeled node into a set of features




$G_2 : \{ [0,\infty], [0,7] \rightarrow [9,11], [0,\infty] \rightarrow [76,\infty], [0,\infty], \; [0,\infty], [43,\infty] \rightarrow [0,\infty], [0,14], \; \ldots \}$

- To produce qualified features, we use the following score function:

$$\text{Score of a feature f} = \frac{\text{The number of labeled nodes chosen by f}}{\text{The number of nodes chosen by f}}$$

# Learning a Feature

(1) Starts from the most general feature f:

[0,∞], [0,∞]

# Learning a Feature

(1) Starts from the most general feature f:

[0,∞], [0,∞]

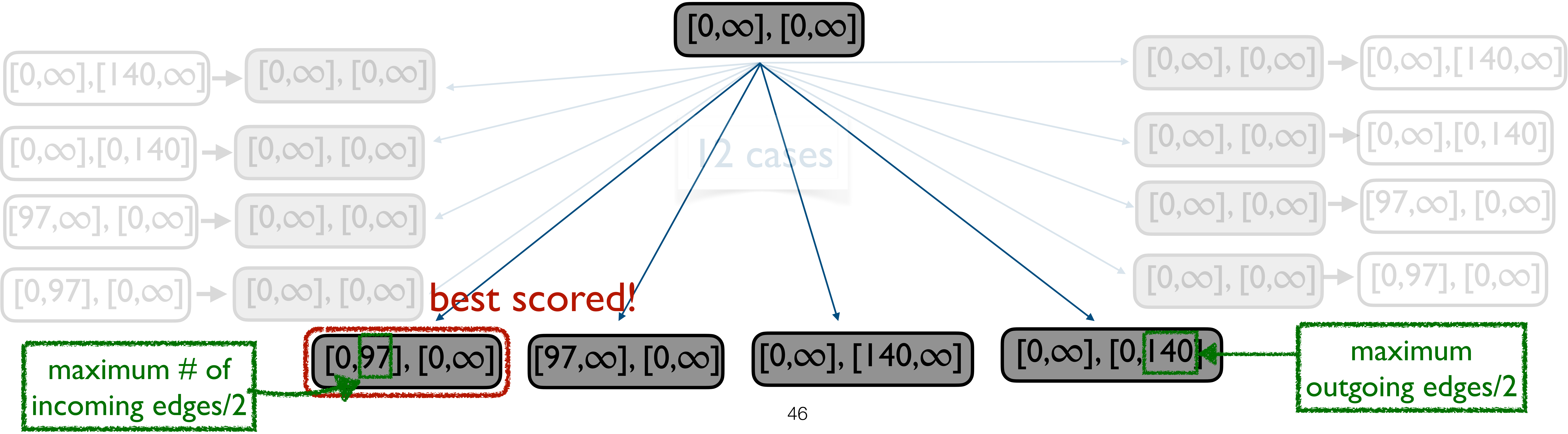
(2) Enumerate possible specified features from f and choose the best scored one:

# Learning a Feature

(1) Starts from the most general feature f:

[0,∞], [0,∞]

(2) Enumerate possible specified features from f and choose the best scored one:



# Learning a Feature

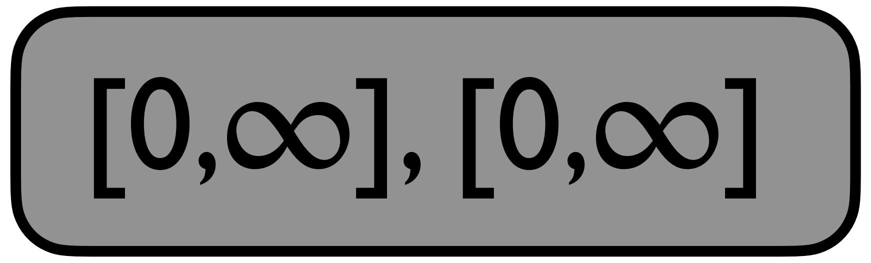
(1) Starts from the most general feature f:

[0,∞], [0,∞]

(2) Enumerate possible specified features from f and choose the best scored one:

# Learning a Feature

(1) Starts from the most general feature f:




(2) Enumerate possible specified features from f and choose the best scored one:

# Learning a Feature

(1) Starts from the most general feature f:

[0,∞], [0,∞]

(2) Enumerate possible specified features from f and choose the best scored one:

# Learning a Feature

(1) Starts from the most general feature f:

[0,∞], [0,∞]

(2) Enumerate possible specified features from f and choose the best scored one:

[0,97], [0,∞] score : 0.06

(3) Repeat (2) until the specified feature has a better score than a hyper-parameter $\gamma$:

[0,48], [0,∞] → [97,∞], [140,∞] $> \gamma$ (= 0.5)

score : 0.55

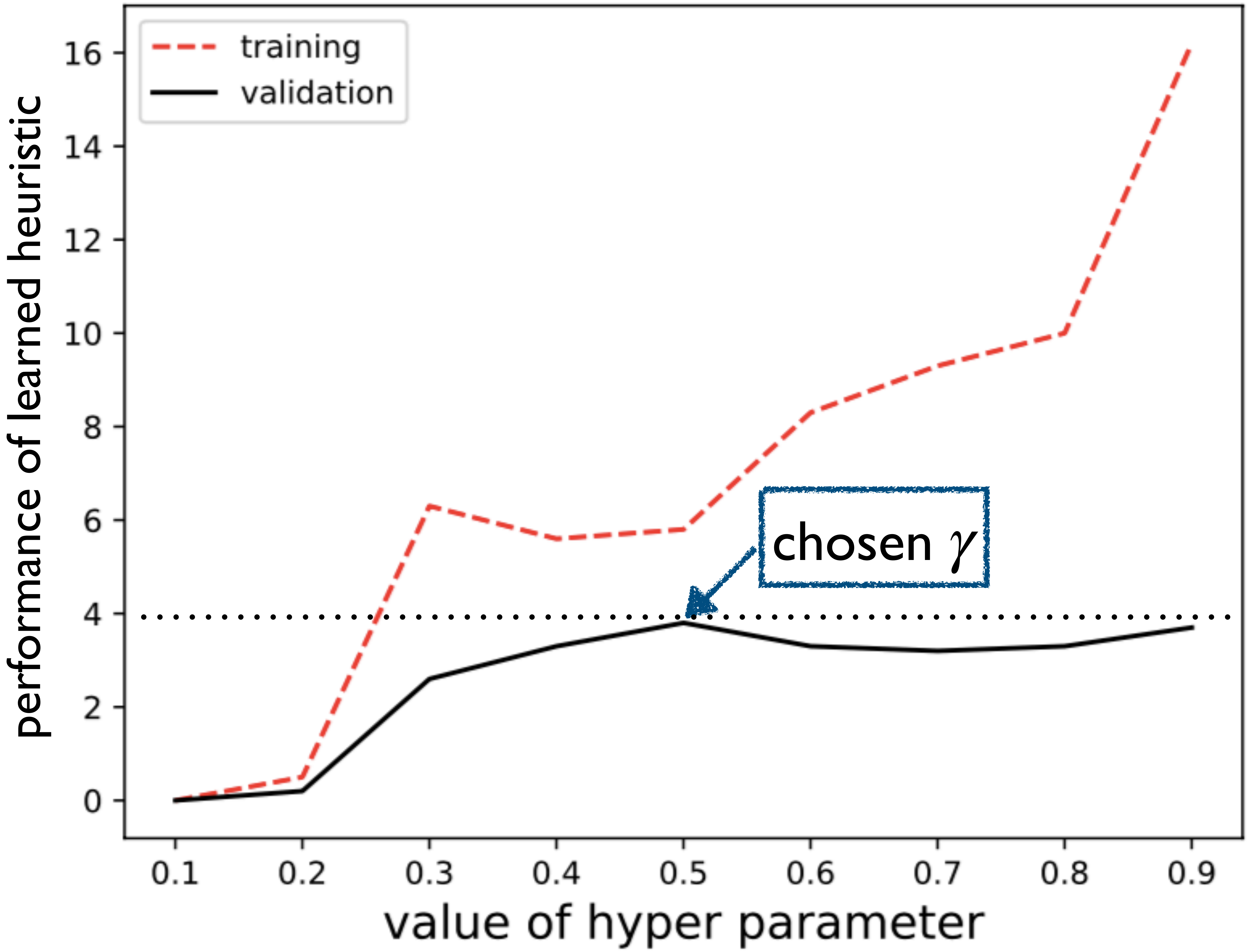
# Learning a Feature

(1) Starts from th

(2) Enumerate pc … scored one:

(3) Repeat (2) un … ·perparameter $\gamma$:

- Too high value of $\gamma$ results overfitting



# Learning a Feature

(1) Starts from the most general feature f:

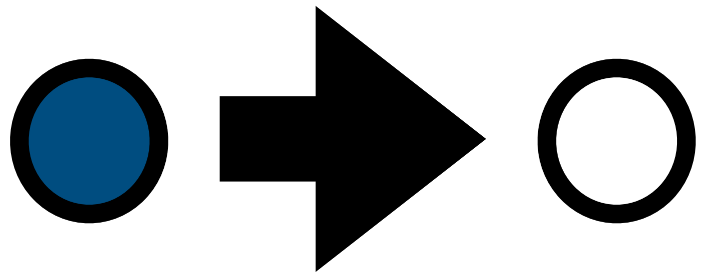
[0,∞], [0,∞]

(2) Enumerate possible specified features from f and chose the best scored one:

[0,97], [0,∞]

(3) Repeat (2) until the specified feature has a better score than a hyper-parameter $\gamma$:

[0,48], [0,∞] → [97,∞], [140,∞] $> \gamma$ (0.5)

(4) Relabel the nodes chosen from the feature (e.g., ).

# Learning a Feature

(1) Starts from the most general feature f:

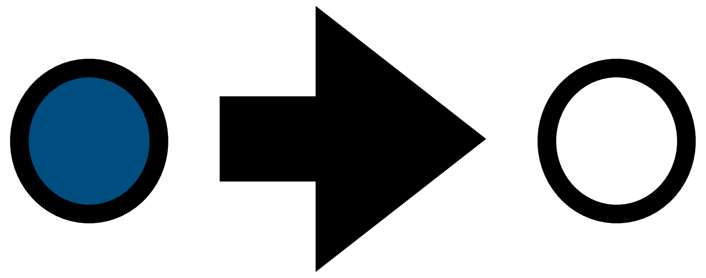
[0,∞], [0,∞]

(2) Enumerate possible specified features from f and chose the best scored one:

[0,97], [0,∞]

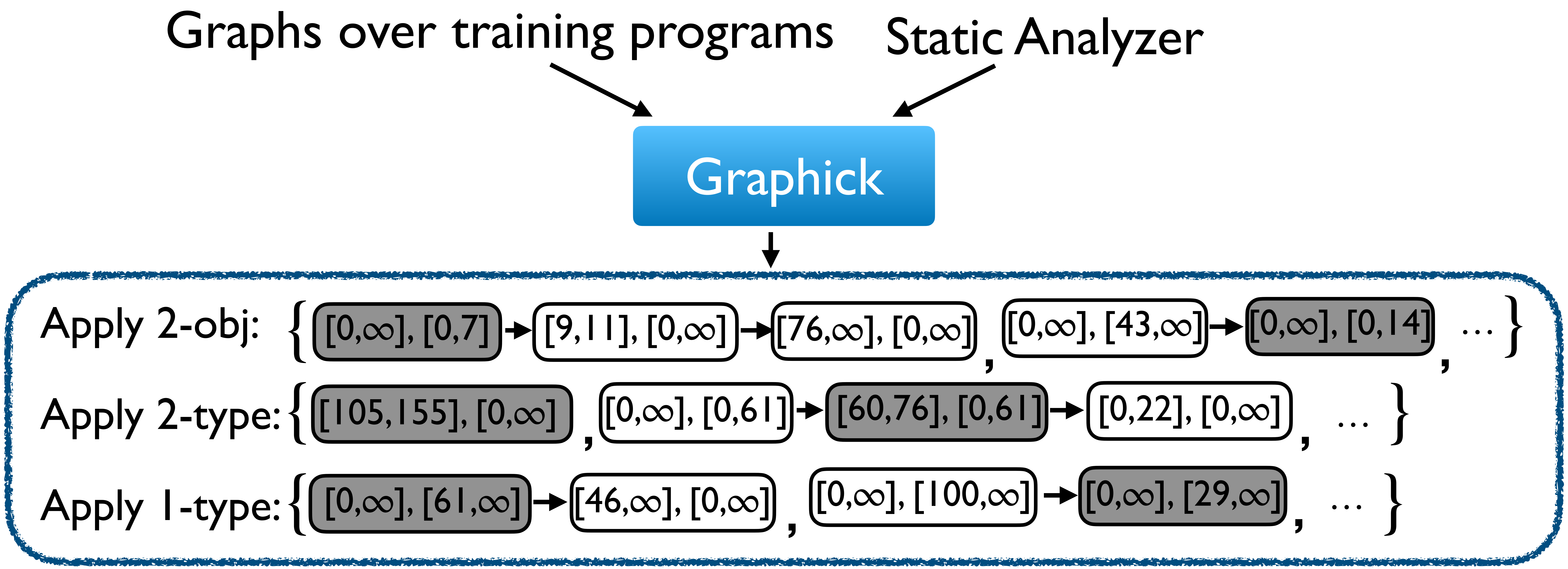
(3) Repeat (2) until the specified feature has a better score than a hyper-parameter $\gamma$:

[0,48], [0,∞] → [97,∞], [140,∞] $> \gamma$ (0.5)

(4) Relabel the nodes chosen from the feature (e.g., ).

(5) Repeat (1)~(4) until all the labeled nodes are covered.

# Our Framework: Graphick




Automatically generated graph-based heuristic

# Experiments

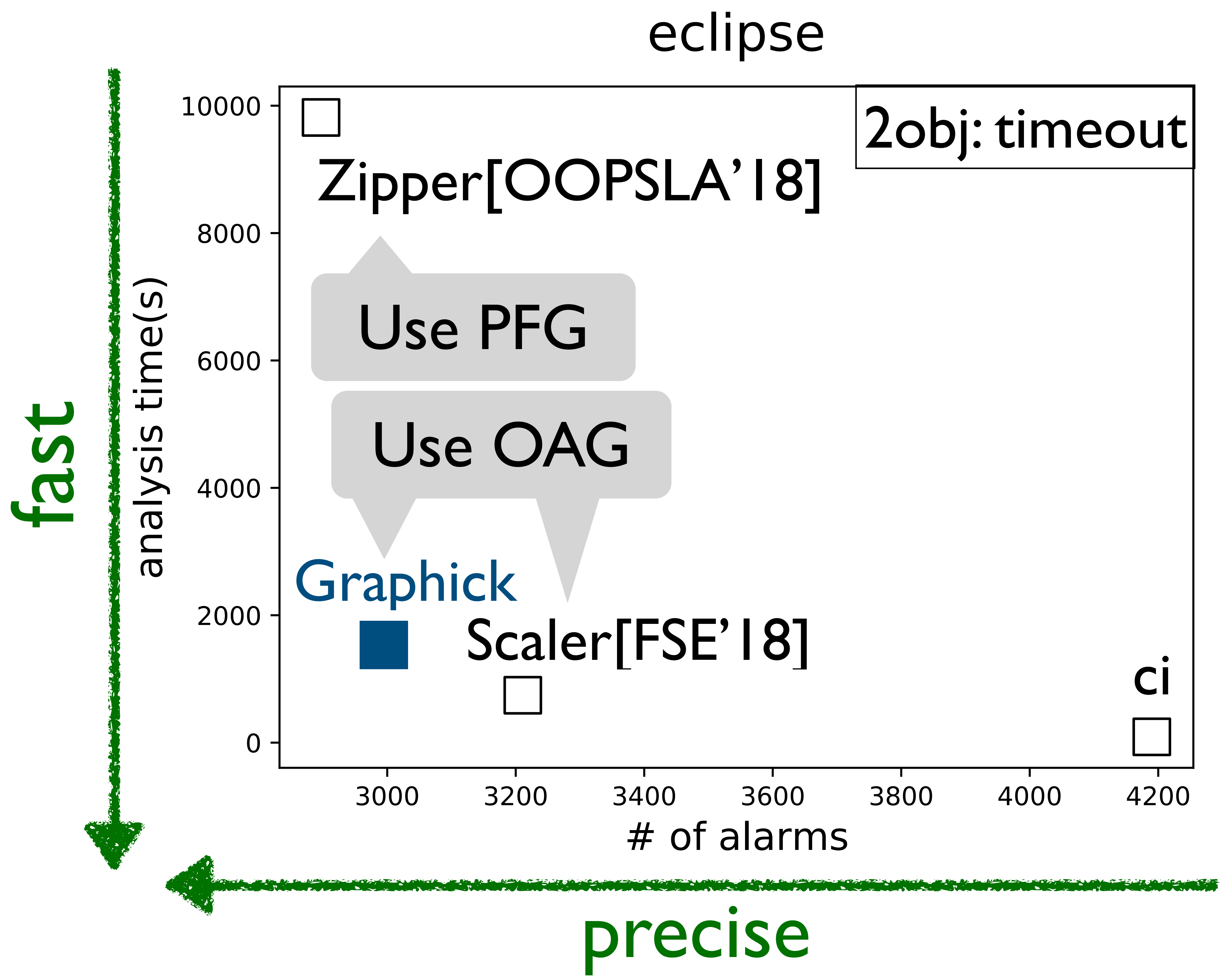
# Settings

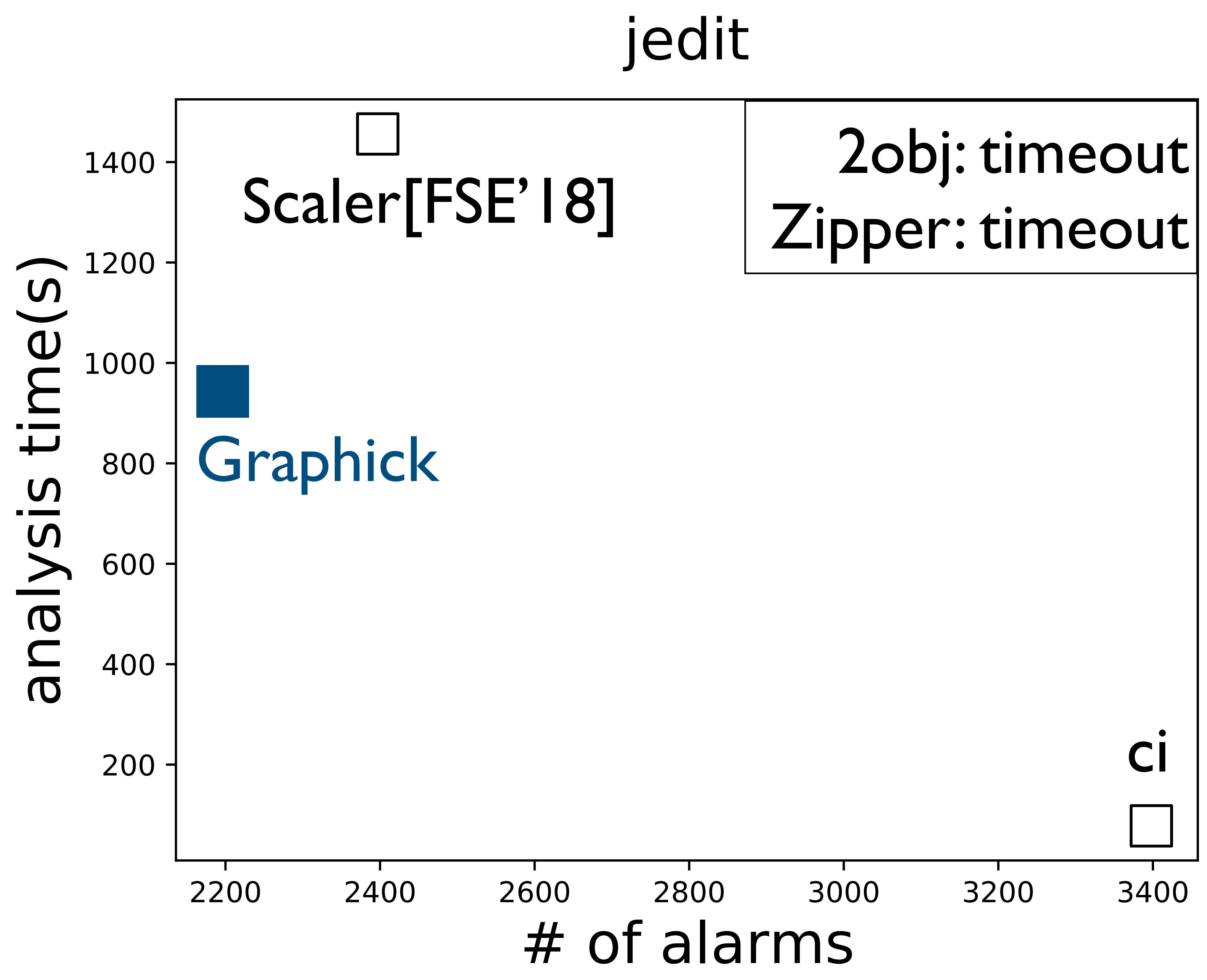
- Doop:
  - State-of-the-art Java pointer analyzer

- Heuristic instances:
  - Context sensitivity heuristic (we trained heuristic with 3 small programs)
  - Heap abstraction heuristic (we trained heuristic with 4 small programs)

# Comparison to Graph-based Context Sensitivity Heuristics

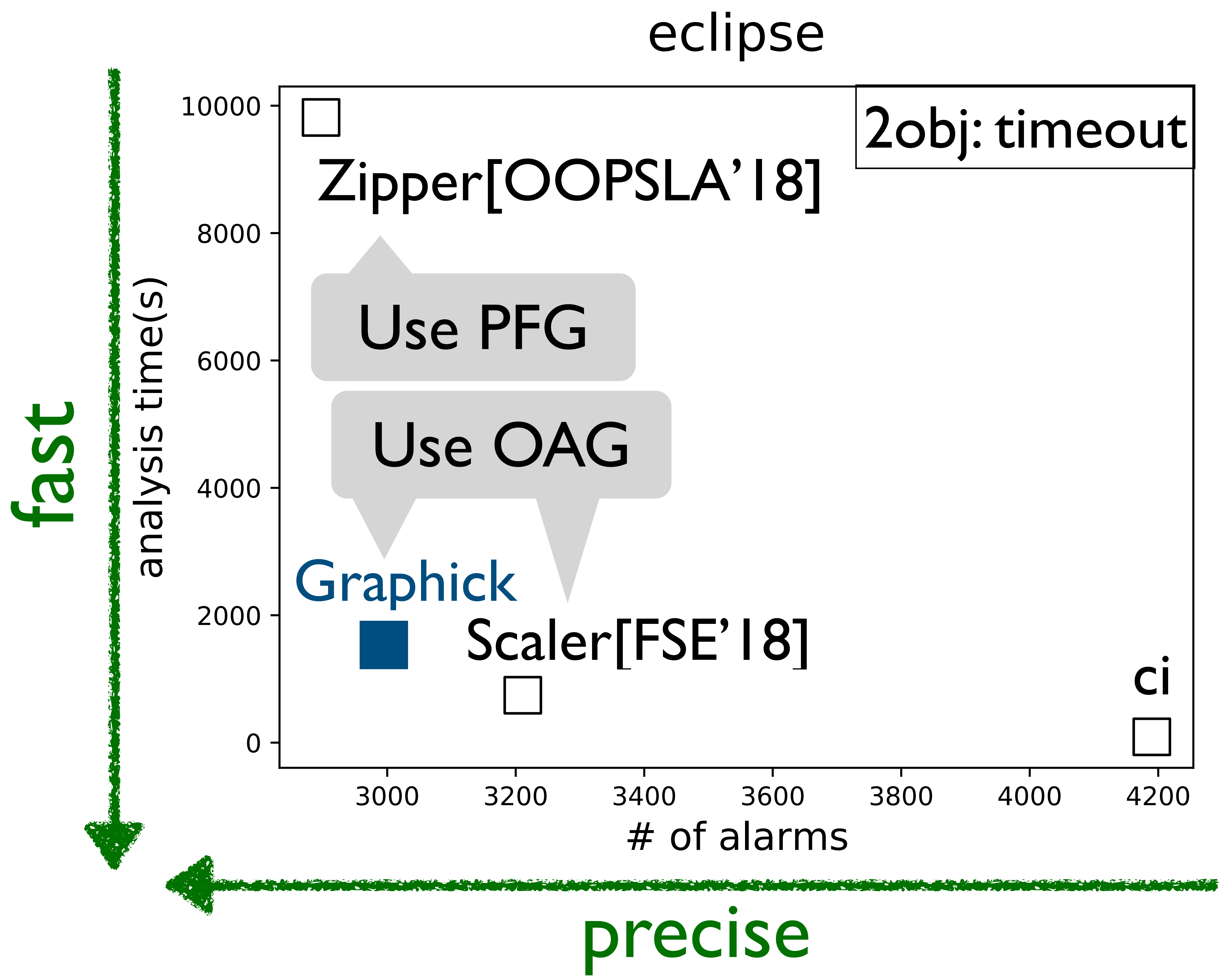
- We use OAG, used in Scaler, to produce a context sensitivity heuristic
- From OAG, Graphick produces a competitive context-sensitivity heuristic

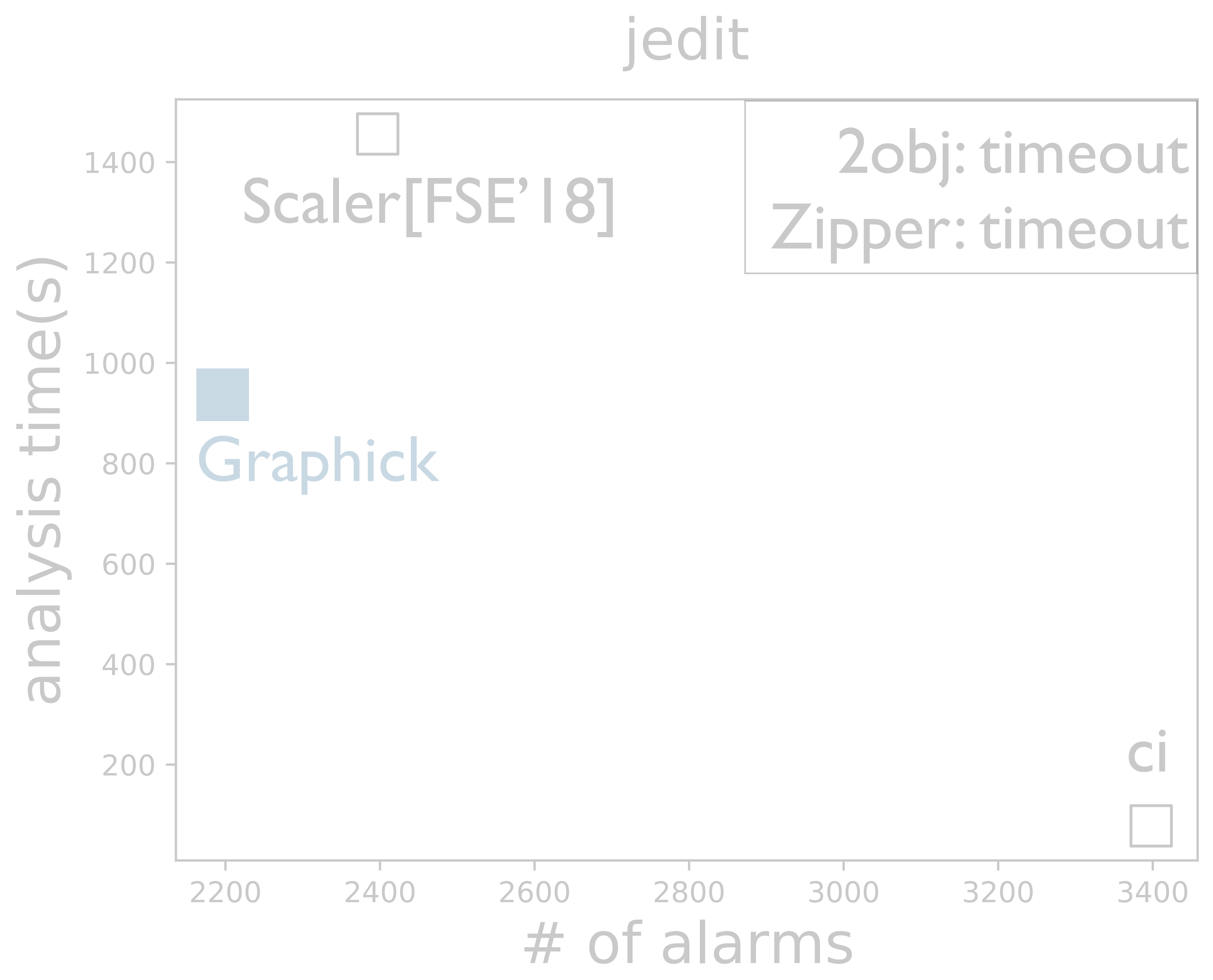


# Comparison to Graph-based Context Sensitivity Heuristics

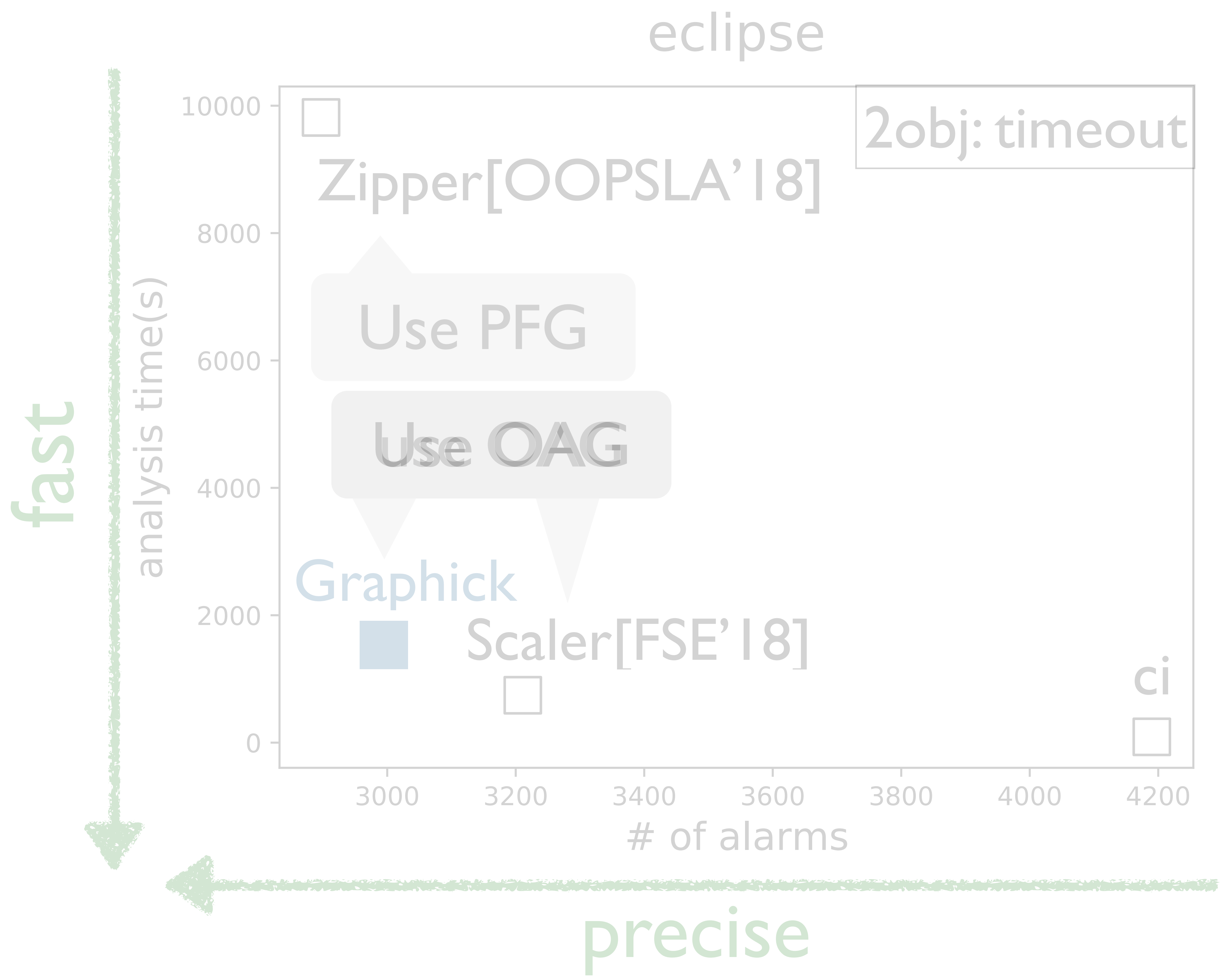
- We use OAG, used in Scaler, to produce a context sensitivity heuristic
- From OAG, Graphick produces a competitive context-sensitivity heuristic

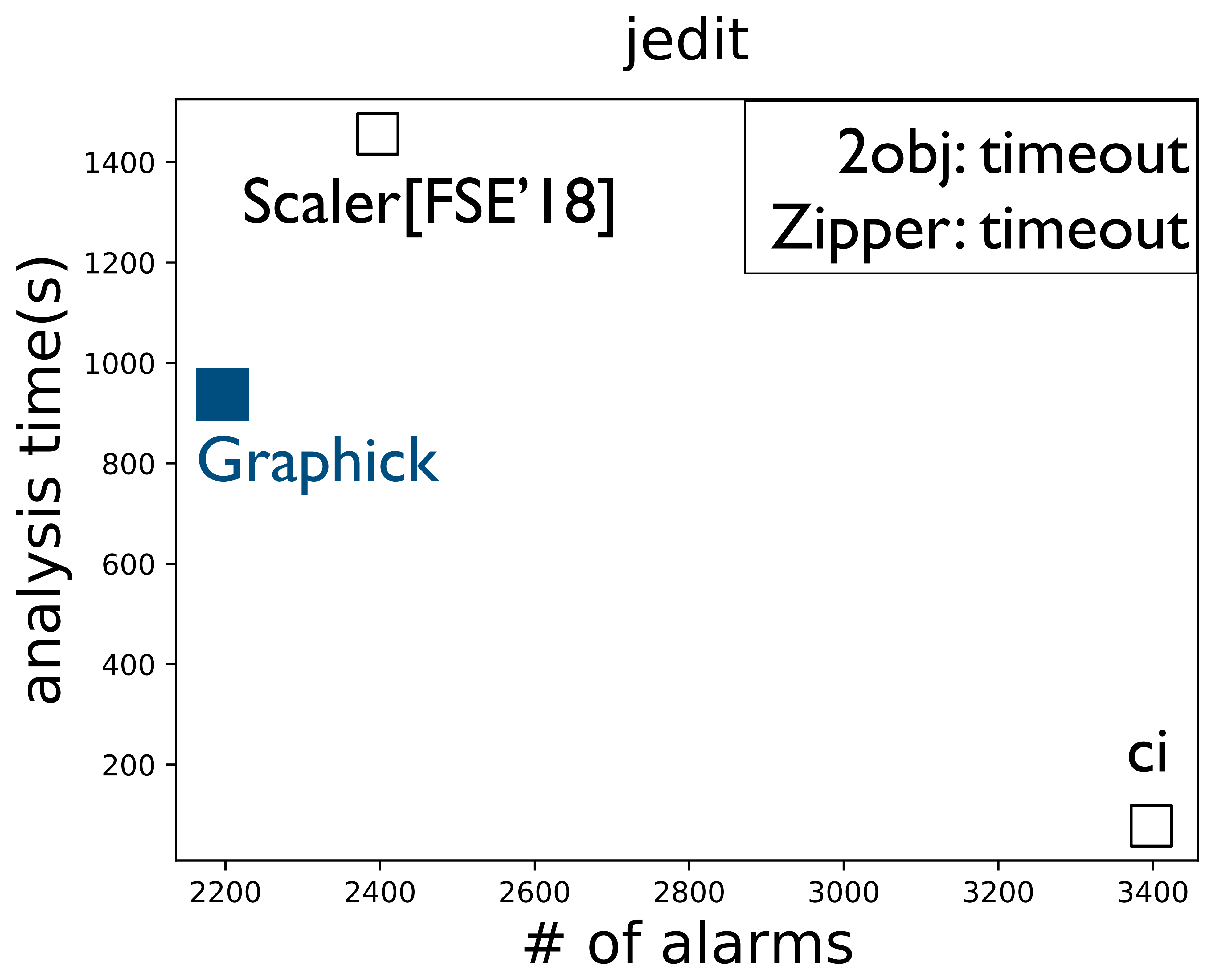


# Comparison to Graph-based Context Sensitivity Heuristics

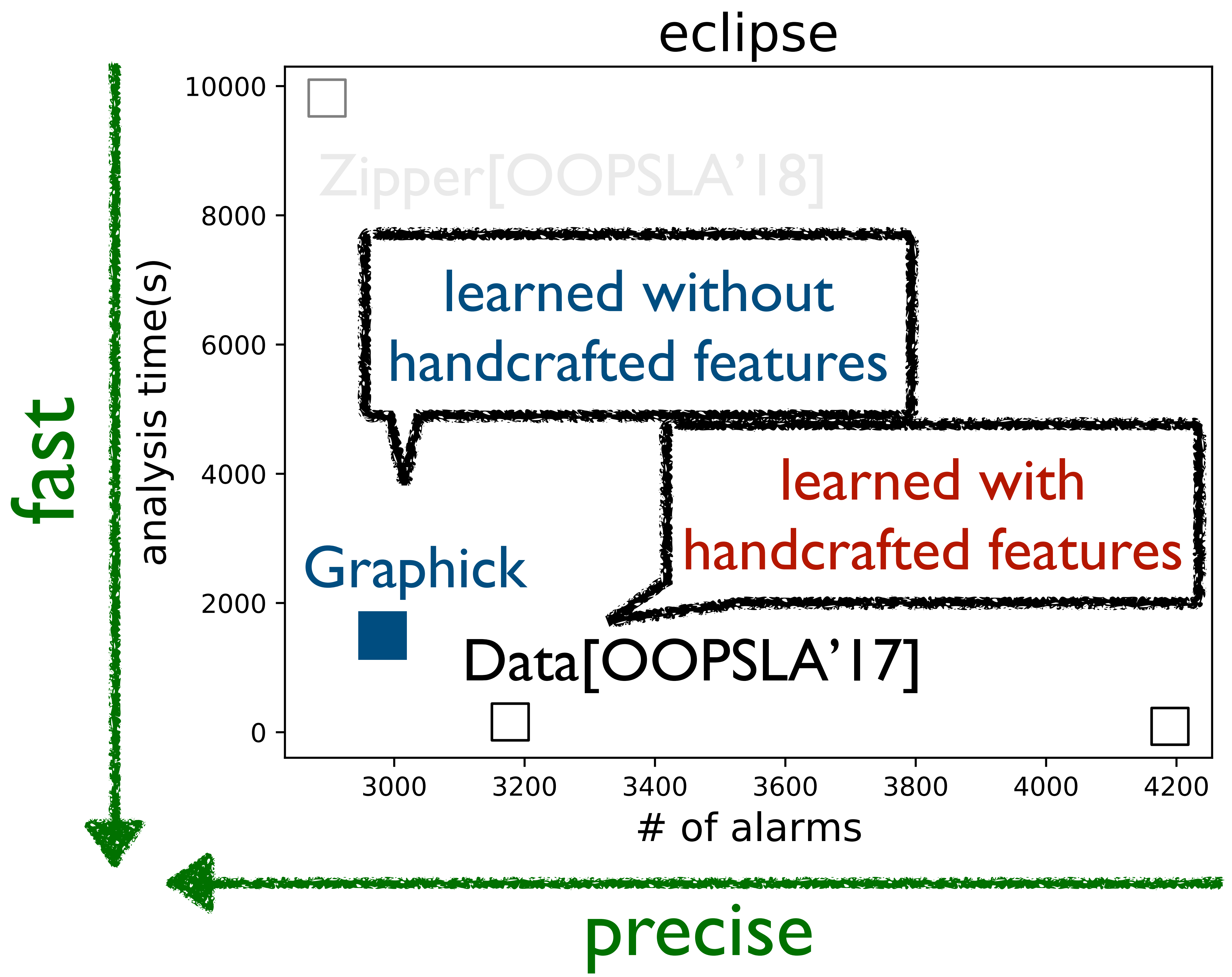
- We use OAG, used in Scaler, to produce a context sensitivity heuristic
- From OAG, Graphick produces a competitive context-sensitivity heuristic

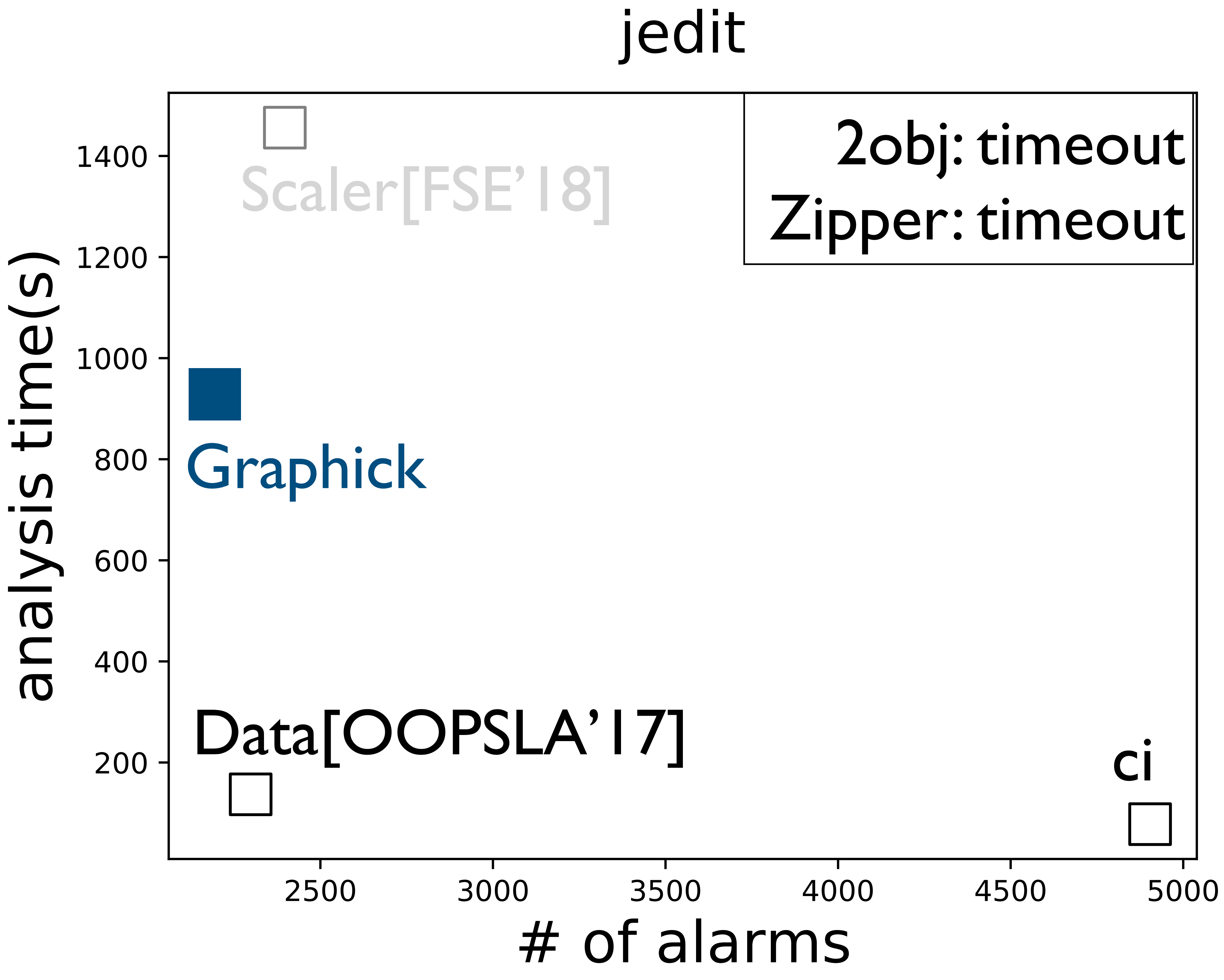


# Comparison to a Previous Data-driven Context Sensitivity Heuristic

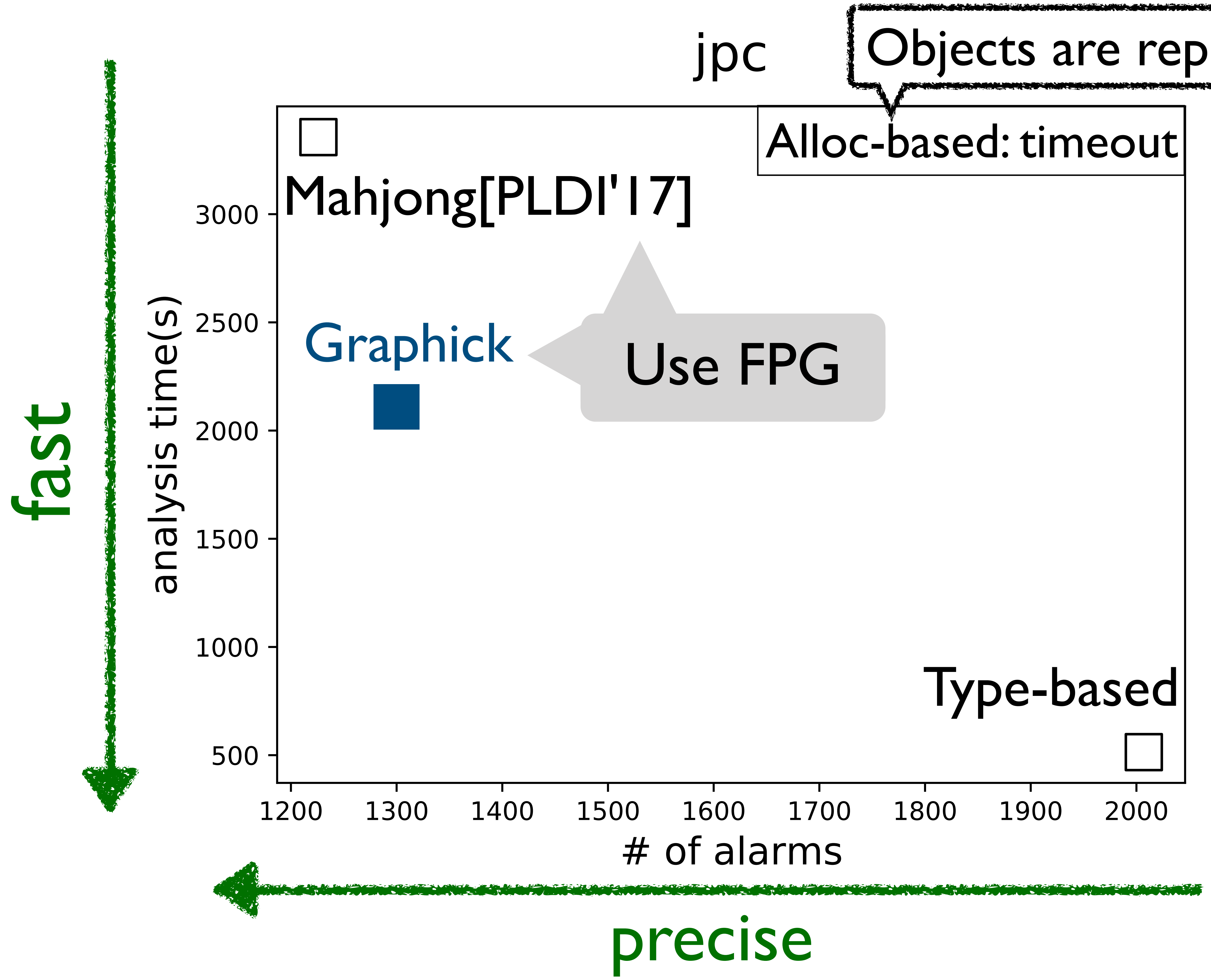
- Comparison with a data-driven context-sensitivity heuristic learned with handcrafted features
- Without handcrafted features, Graphick produces a competitive context-sensitivity heuristic

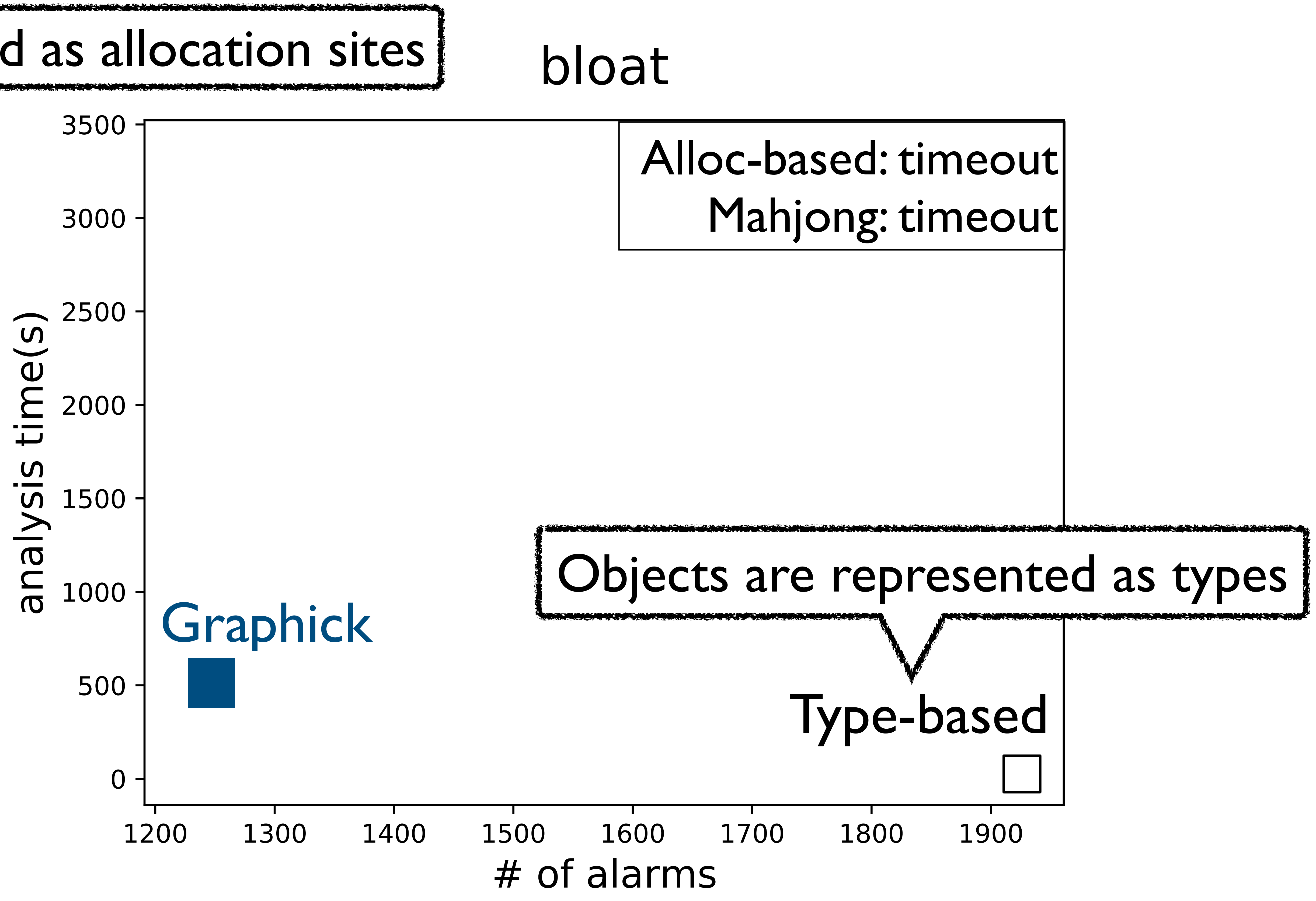


# Comparison to Heap Abstraction Heuristics

- We use field-points-to graph (FPG), used in Mahjong, to produce heap abstraction heuristic
- From FPG, Graphick produces cost-effective heap-abstraction heuristic



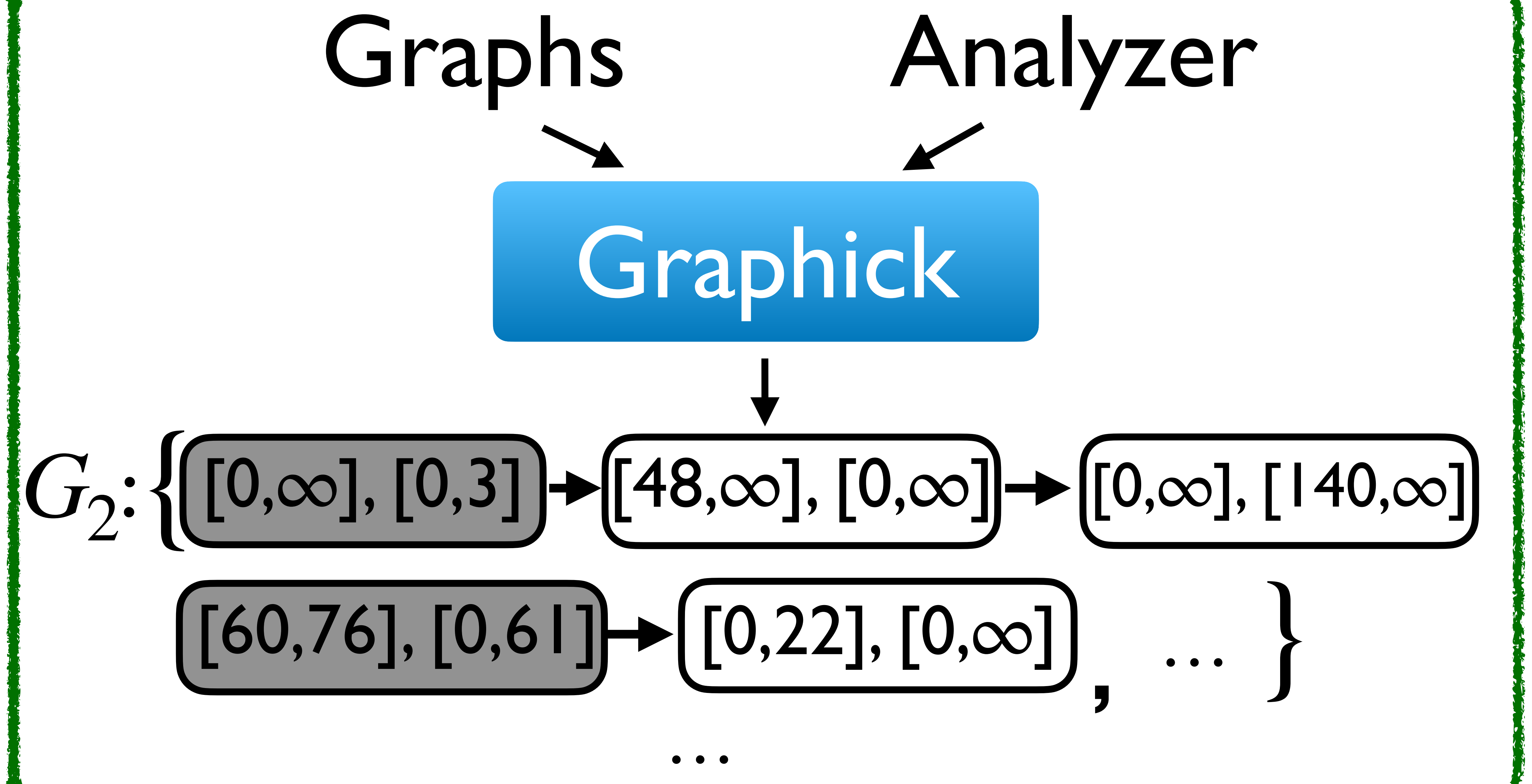
# Summary

- We made Graphick to automatically generate graph-based analysis heuristics
  - Two key ideas are our feature description language and learning algorithm

$$Feature = \widehat{Node}^* \times \widehat{Node} \times \widehat{Node}^*$$

$$\widehat{Node} = Itv \times Itv$$

$$Itv = \{[a, b] \mid a \in \mathbb{N}, b \in \mathbb{N} \cup \infty, \}$$




Thank you