Hybrid Context-Sensitivity for Points-To Analysis

George Kastrinis

Summer Intern @

LOGICBLOX®

Yannis Smaragdakis

University of Athens

Executive Summary

- **Hybrid: Combine** Call-Site & Object sensitivity
- Naively keeping both contexts **not scalable**
- **Favor** each kind in different places
- e.g. Call-Site Sens for **Static** Methods
- The precision from **both**, the cost of **only one**

Hybrid Context-Sensitivity for **Points-To Analysis**

Hybrid Context-Sensitivity for **Points-To Analysis**

What **objects** may a **variable** point to?
(statically, object = allocation site)

Hybrid Context-Sensitivity for **Points-To Analysis**

George Kastrinis, Yannis Smaragdakis ~ Hybrid Context-Sensitivity for Points-To Analysis

Simple Example

```
Obj A::id(Obj i) {return i}

void B::bar(A a1, A a2) {
    Obj o1 = new ■
    Obj c = a1.id(o1)
    Obj o2 = new ▲
    Obj d = a2.id(o2)
}
```

Points-To Sets

```
Obj A::id(Obj i) {return i}

void B::bar(A a1, A a2) {
    Obj o1 = new ■
    Obj c = a1.id(o1)
    Obj o2 = new ▲
    Obj d = a2.id(o2)
}
```

o1 → ■
o2 → ▲

i → ■, ▲

c → ■, ▲
d → ■, ▲

Points-To Sets

```
Obj A::id(Obj i) {return i}

void B::bar(A a1, A a2) {
    Obj o1 = new ■
    Obj c = a1.id(o1)
    Obj o2 = new ▲
    Obj d = a2.id(o2)
}
```

```
o1 → ■
o2 → ▲

i → ■, ▲

c → ■, ▲
d → ■, ▲
```

Can we do better?

Hybrid **Context-Sensitivity** for Points-To Analysis

Hybrid **Context-Sensitivity** for Points-To Analysis

Qualify variables (and objects) with context information

Call-Site Sensitivity

```
Obj A::id(Obj i) {return i}

void B::bar(A a1, A a2) {
    Obj o1 = new ■
inv1  Obj c = a1.id(o1)
    Obj o2 = new ▲
inv2  Obj d = a2.id(o2)
}
```

o1 → ■
o2 → ▲

i(inv1) → ■
i(inv2) → ▲

c → ■
d → ▲

Call-Site Sensitivity

```
Obj A::id(Obj i) {return i}

void B::bar(A a1, A a2) {
    Obj o1 = new ■
inv1  Obj c = a1.id(o1)
    Obj o2 = new ▲
inv2  Obj d = a2.id(o2)
}
```

o1 → ■
o2 → ▲

i(inv1) → ■
i(inv2) → ▲

c → ■
d → ▲

#contexts?
statically known

Object Sensitivity

```
Obj A::id(Obj i) {return i}

void B::bar(A a1, A a2) {
    Obj o1 = new ■
    Obj c = a1.id(o1)
    Obj o2 = new ▲
    Obj d = a2.id(o2)
}
```

a1 → ★
a2 → ♦

Object Sensitivity

```
Obj A::id(Obj i) {return i}

void B::bar(A a1, A a2) {
    Obj o1 = new ■
    Obj c = a1.id(o1)
    Obj o2 = new ▲
    Obj d = a2.id(o2)
}
```

a1 → ★
a2 → ♦

o1 → ■
o2 → ▲

i(★) → ■

i(♦) → ▲

c → ■
d → ▲

Object Sensitivity

```
Obj A::id(Obj i) {return i}

void B::bar(A a1, A a2) {
    Obj o1 = new ■
    Obj c = a1.id(o1)
    Obj o2 = new ▲
    Obj d = a2.id(o2)
}
```

a1 → ★
a2 → ♦

o1 → ■
o2 → ▲

i(★) → ■

i(♦) → ▲

c → ■
d → ▲

#contexts?
depends on analysis

Call-Site Sensitivity

#contexts?
statically known

≠

not comparable

Object Sensitivity

#contexts?
depends on analysis

Natural Idea: Combine The Two

Uniform Combination

(add extra Call-Site information)

Uniform Combination

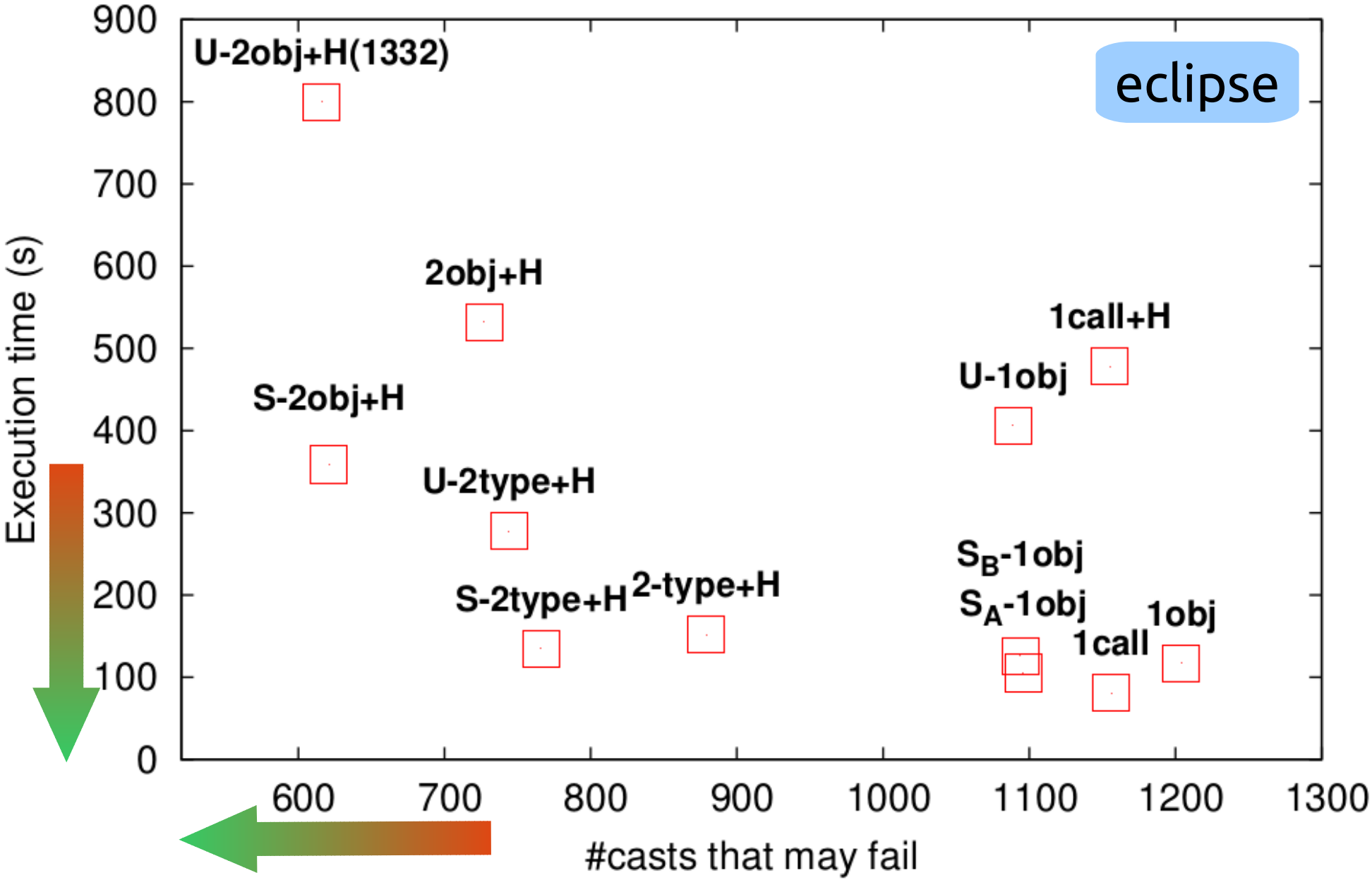
(add <u>extra</u> Call-Site information)

Our Approach: Hybrid Combination

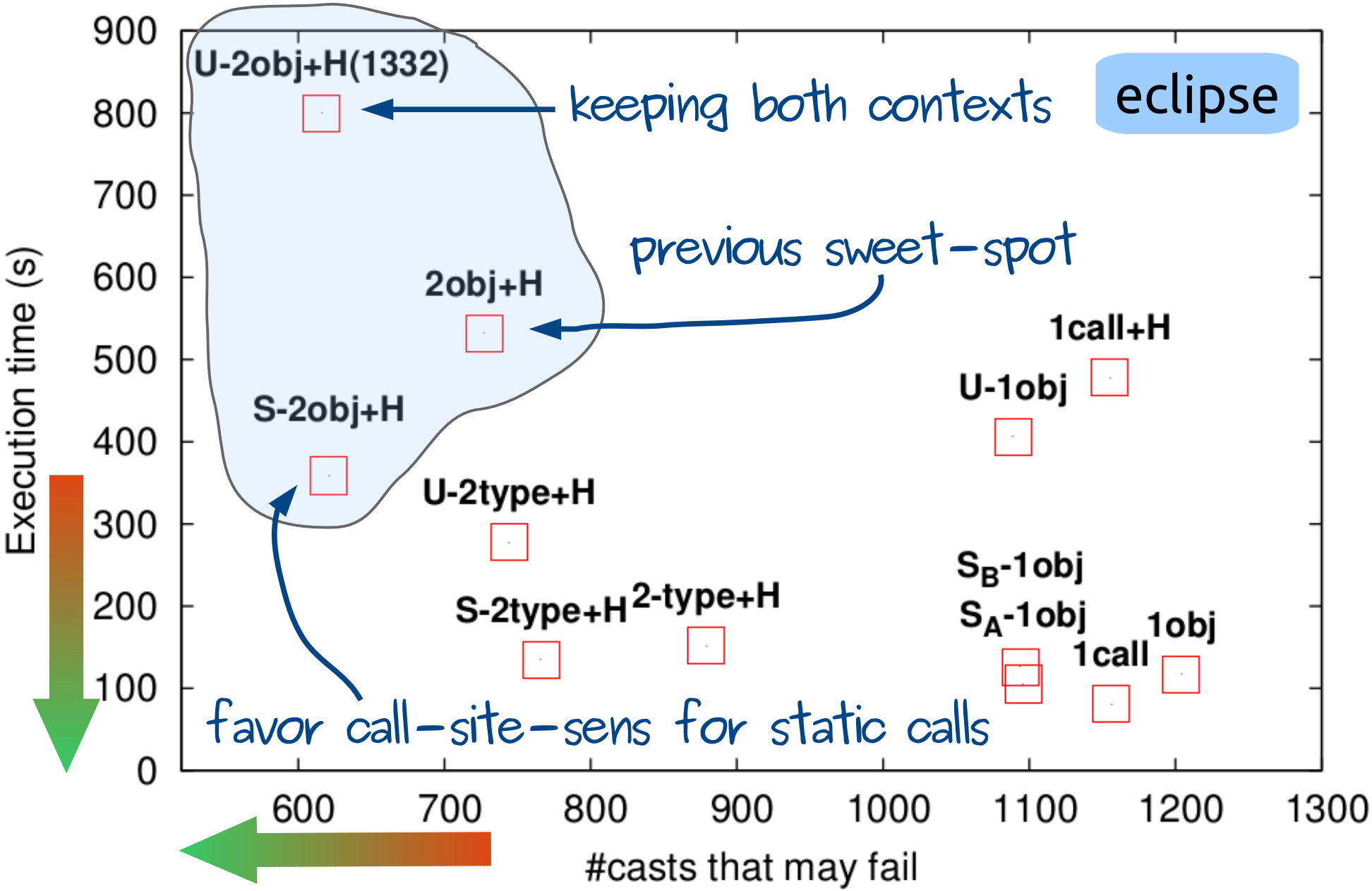
(don't keep both, switch to best)

Why You Care: Serious Practical Benefit

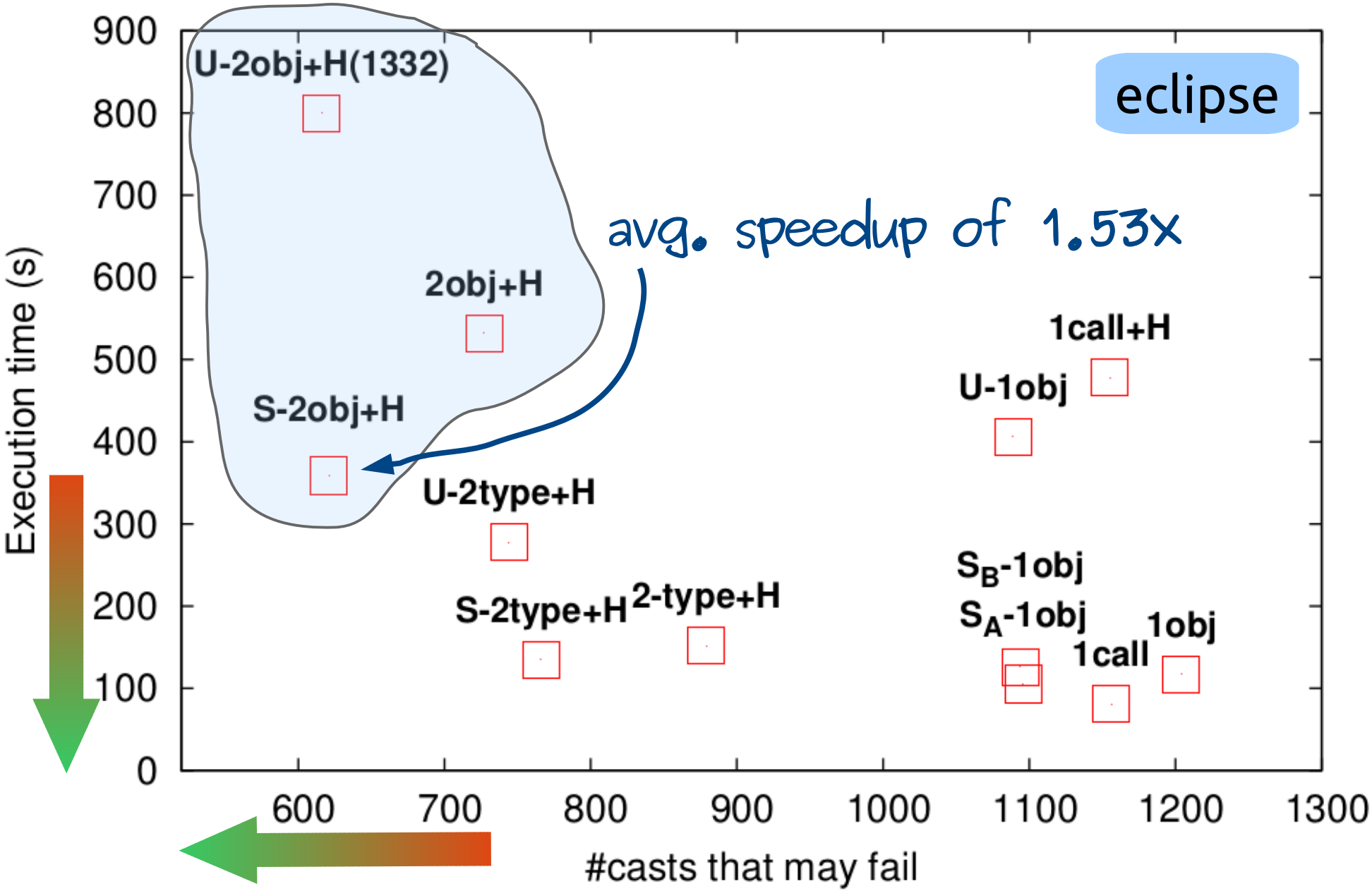
Performance vs Precision



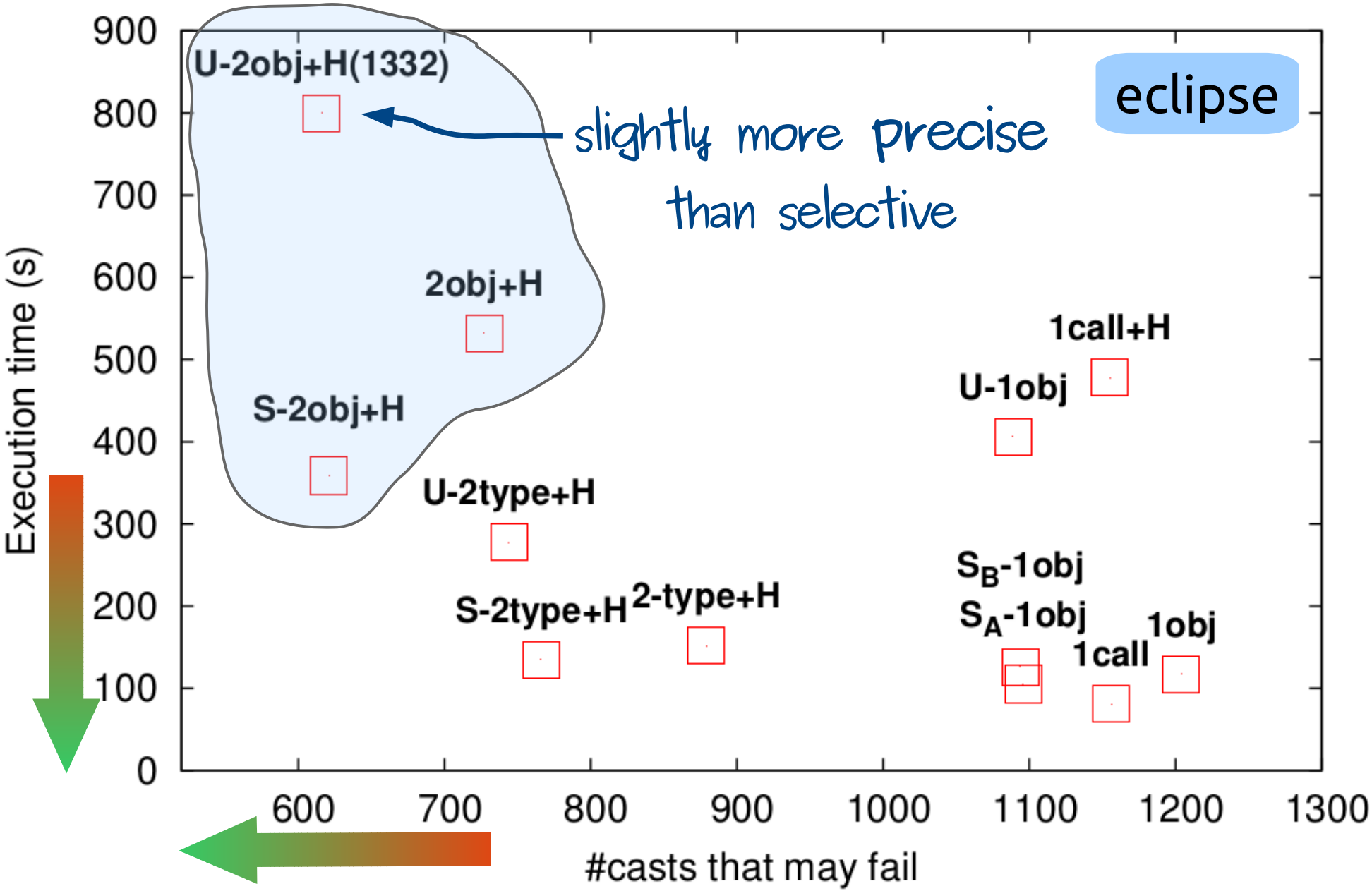
Performance vs Precision



Performance vs Precision



Performance vs Precision



OK, How???

(the technical part)

Our Setting:
Analysis as Datalog rules

Example Datalog Rule

$$\mathbf{P}\ (x),$$
$$\mathbf{Q}\ (x,\ z) \leftarrow \mathbf{R}\ (x,\ y,\ w),$$
$$\mathbf{S}\ (y,\ z).$$

Example Datalog Rule

then...

```
P (x),
Q (x, z) ← R (x, y, w),
           S (y, z).
```

if...

Analyzing Java with Datalog

Get from this

```
v = new A()

to = from

to = base.fld
base.fld = from

base.sig(...)

A.sig(...)
```

Get from this	To this
`v = new A()`	ALLOC (*var, obj, meth*) OBJTYPE (*obj, type*)
`to = from`	MOVE (*to, from*)
`to = base.fld` `base.fld = from`	LOAD (*to, base, fld*) STORE (*base, fld, from*)
`base.sig(...)`	VCALL (*base, sig, invo*)
`A.sig(...)`	SCALL (*sig, invo*)

Get from this To this

`v = new A()`	ALLOC (*var, obj, meth*) OBJTYPE (*obj, type*)
`to = from`	MOVE (*to, from*)
`to = base.fld`	LOAD (*to, base, fld*)
`base.fld = from`	STORE (*base, fld, from*)
`base.sig(...)`	VCALL (*base, sig, invo*)
`A.sig(...)`	SCALL (*sig, invo*)

Rules in our Analysis

Static Method Calls

```
A.sfoo(...);
```

REACHABLE (*inMeth*, *callerCtx*),
SCALL (*toMeth*, *invo*, *inMeth*).

Static Method Calls

```
A.sfoo(...);
```

```
REACHABLE (toMeth              ),
CALLGRAPH (invo, callerCtx, toMeth              ) ←
   REACHABLE (inMeth, callerCtx),
   SCALL (toMeth, invo, inMeth).
```

Static Method Calls

```
A.sfoo(...);
```

Construct a new calling context

```
MergeStatic (invo, ctx) =   calleeCtx,
Reachable (toMeth, calleeCtx),
CallGraph (invo, callerCtx, toMeth, calleeCtx) ←
   Reachable (inMeth, callerCtx),
   SCall (toMeth, invo, inMeth).
```

Virtual Method Calls

```
a.foo(...);
```

MERGE (*obj*, *objCtx*, *invo*, *ctx*) = ***calleeCtx***,
REACHABLE (*toMeth*, ***calleeCtx***),
CALLGRAPH (*invo*, *callerCtx*, *toMeth*, ***calleeCtx***) ←
  REACHABLE (*inMeth*, *callerCtx*),
  VCALL (*base*, *sig*, *invo*, *inMeth*),
  VARPOINTSTO (*base*, *callerCtx*, *obj*, *objCtx*),
  OBJTYPE (*obj*, *objT*), LOOKUP (*objT*, *sig*, *toMeth*).

Same Logic

3 functions to rule all contexts

RECORD (...) = *newObjCtx* — Object Allocation

MERGE (...) = *newCtx* — Virtual Methods

MERGESTATIC (...) = *newCtx* — Static Methods

Use the information available at each point to create a new Context

1 Object Sensitive + 1 Heap

Allocation Site
of Receiver

```
a.foo(...);
        MERGE (obj, objCtx, invo, ctx) = obj

A.sfoo(...);
        MERGESTATIC (invo, ctx) = ?
```

Need an **Allocation Site**
We are in a **Static** Method!

1 Object Sensitive + 1 Heap

Allocation Site of Receiver

```
a.foo(...);
        MERGE (obj, objCtx, invo, ctx) = obj

A.sfoo(...);
        MERGESTATIC (invo, ctx) = ctx
```

Copy that of the Caller...

Hybrid Context-Sensitivity for Points-To Analysis

Hybrid Context-Sensitivity for Points-To Analysis

Combine different context kinds

Hybrid Context-Sensitivity for Points-To Analysis

Combine different context kinds

WHERE? **HOW?**

Hybrid Context-Sensitivity for Points-To Analysis

Combine different context kinds

WHERE? **HOW?**

look at one example; many more in the paper

Selective Combination

(<u>favor</u> call-site sens. e.g. in static methods)

e.g. of Selective Combination

```
A a = new A();
```

```
RECORD (obj, ctx) = first(ctx)
```

Current Context of allocating method

e.g. of Selective Combination

```
a.foo(...);
```

```
RECORD (obj, ctx) = first(ctx)

MERGE (obj, objCtx, invo, ctx) = [obj, objCtx]
```

Allocation Site of Receiver

Allocation Site of Receiver's Receiver

e.g. of Selective Combination

```
A.sfoo(...);
```

```
RECORD (obj, ctx) = first(ctx)

MERGE (obj, objCtx, invo, ctx) = [obj, objCtx]

MERGESTATIC (invo, ctx) = [first(ctx), invo, second(ctx)]
```

Allocation Site
of Caller's Receiver

Invocation Site

?

e.g. of Selective Combination

```
A.sfoo(...);
```

```
RECORD (obj, ctx) = first(ctx)

MERGE (obj, objCtx, invo, ctx) = [obj, objCtx]

MERGESTATIC (invo, ctx) = [first(ctx), invo, second(ctx)]
```

Allocation Site
of Caller's Receiver

Invocation Site

?

Example

Selective Combination Approach

```
RECORD (obj, ctx) = first(ctx)
MERGE (obj, objCtx, invo, ctx) = [obj, objCtx]
MERGESTATIC (invo, ctx) = [first(ctx), invo, second(ctx)]
```

```
void foo (...) {
   A a = new ▲
   a.bar (...)
   A.sbar (...)
}
```

Example

Selective Combination Approach

```
RECORD (obj, ctx) = first(ctx)
MERGE (obj, objCtx, invo, ctx) = [obj, objCtx]
MERGESTATIC (invo, ctx) = [first(ctx), invo, second(ctx)]
```

ctx1 = [■, ★]

```
void foo (...) {
    A a = new ▲
    a.bar (...)
    A.sbar (...)
}
```

Example

Selective Combination Approach

```
RECORD (obj, ctx) = first(ctx)
MERGE (obj, objCtx, invo, ctx) = [obj, objCtx]
MERGESTATIC (invo, ctx) = [first(ctx), invo, second(ctx)]
```

```
void foo (...) {
  A a = new ▲
  a.bar (...)
  A.sbar (...)
}
```

ctx1 = [■, ★]

objCtx1 = ■

Example

Selective Combination Approach

```
RECORD (obj, ctx) = first(ctx)
MERGE (obj, objCtx, invo, ctx) = [obj, objCtx]
MERGESTATIC (invo, ctx) = [first(ctx), invo, second(ctx)]
```

```
void foo (...) {
  A a = new ▲
  a.bar (...)
  A.sbar (...)
}
```

ctx1 = [■, ★]

objCtx1 = ■

ctx2 = [▲, ■]

Example

Selective Combination Approach

```
RECORD (obj, ctx) = first(ctx)
MERGE (obj, objCtx, invo, ctx) = [obj, objCtx]
MERGESTATIC (invo, ctx) = [first(ctx), invo, second(ctx)]
```

```
void foo (...) {
   A a = new ▲
   a.bar (...)
invo2  A.sbar (...)
}
```

ctx1 = [■, ★]

objCtx1 = ■

ctx2 = [▲, ■]

ctx3 = [■, invo2, ★]

first part of ctx always allocation site → precision in RECORD

Example (cont'd)

Selective Combination Approach

```
RECORD (obj, ctx) = first(ctx)
MERGE (obj, objCtx, invo, ctx) = [obj, objCtx]
MERGESTATIC (invo, ctx) = [first(ctx), invo, second(ctx)]
```

ctx3 = [■, invo2, ★]

```
    static void A::sbar (...) {
invo3  B.sfoo (...)
       C c = new ♦
       c.baz (...)
    }
```

ctx4 = [■, invo3, invo2]

objCtx2 = ■

ctx5 = [♦, ■]

static calls inside static → simulate call-site sens

virtual calls inside static → revert back to object sens

RECAP

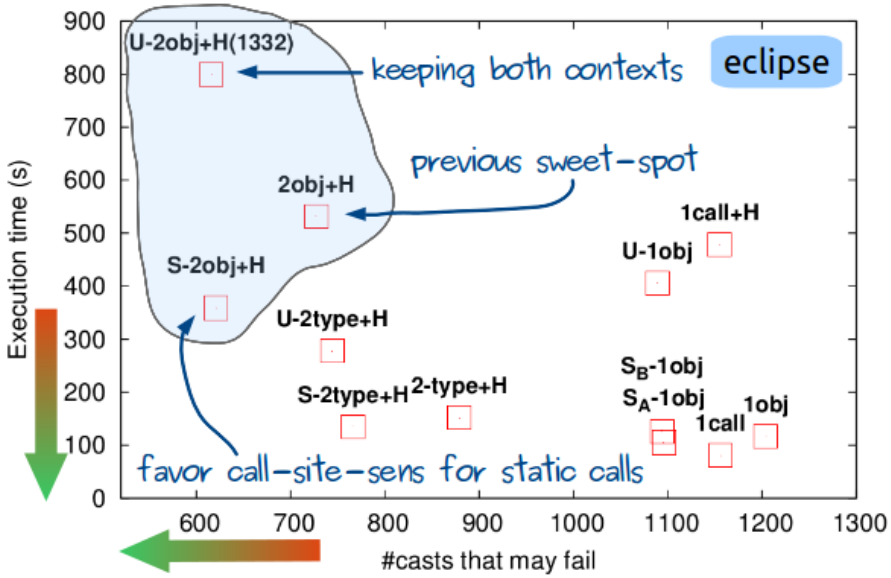
Uniform Combination

(add extra Call-Site information)

Selective Combination

(favor call-site sens. e.g. in static methods)

Performance vs Precision



3 functions to rule all contexts

RECORD (...) = *newObjCtx* — Object Allocation

MERGE (...) = *newCtx* — Virtual Methods

MERGESTATIC (...) = *newCtx* — Static Methods

Use the information available at each point to create a new Context

Hope you enjoyed!

George Kastrinis • http://gkastrinis.info

University of Athens

Summer Intern @ LOGICBLOX®