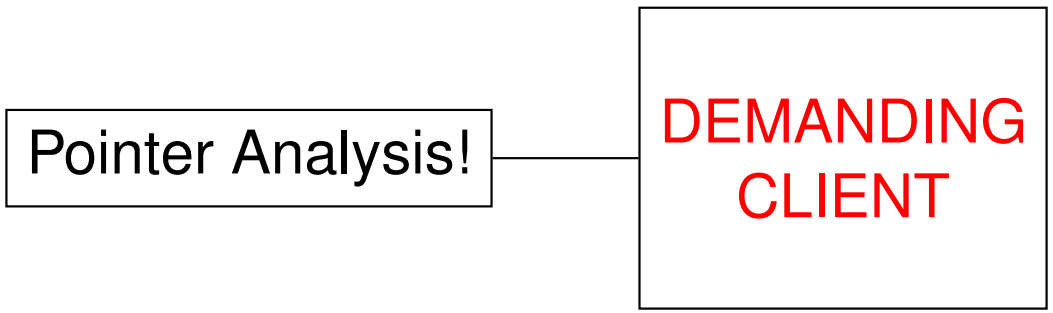
Object representatives: a uniform abstraction for pointer information

Eric Bodden, Patrick Lam and Laurie Hendren

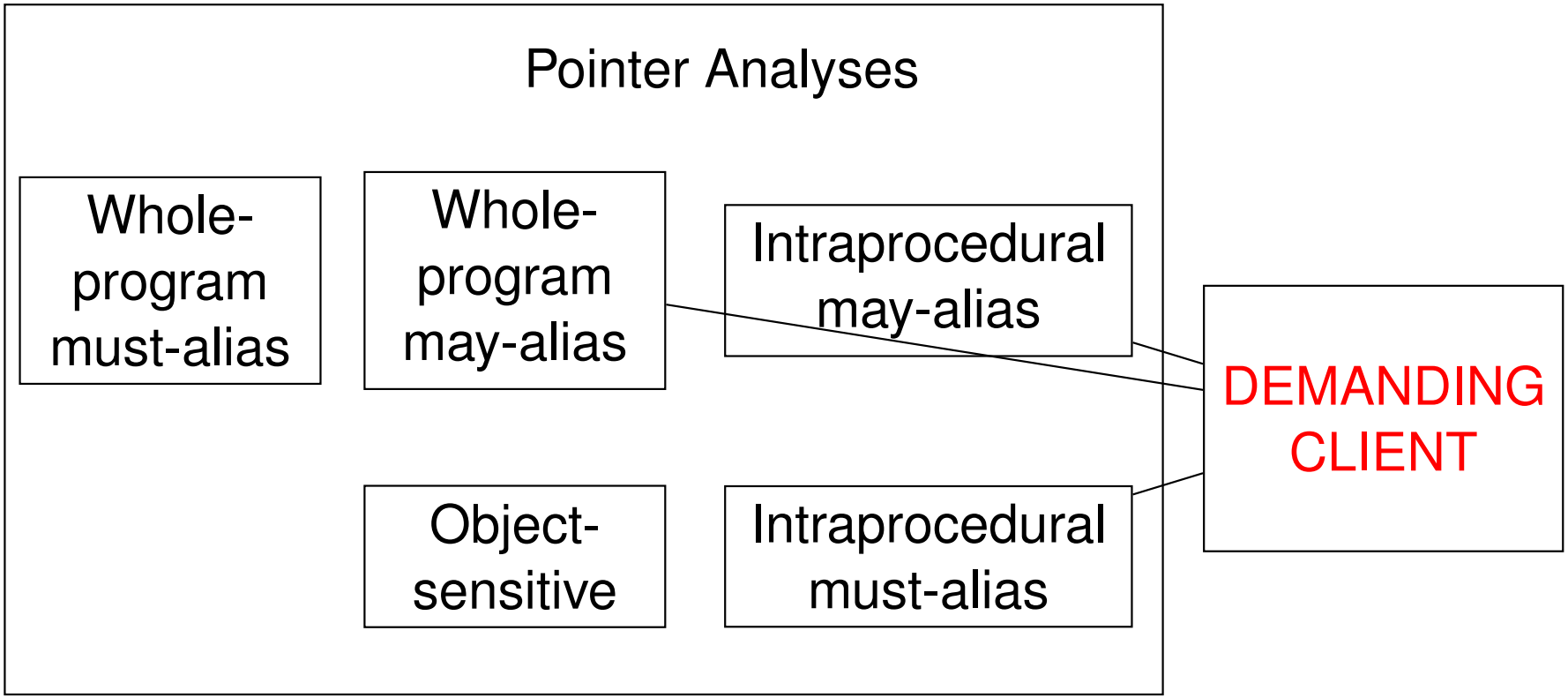
September 2008

Situation

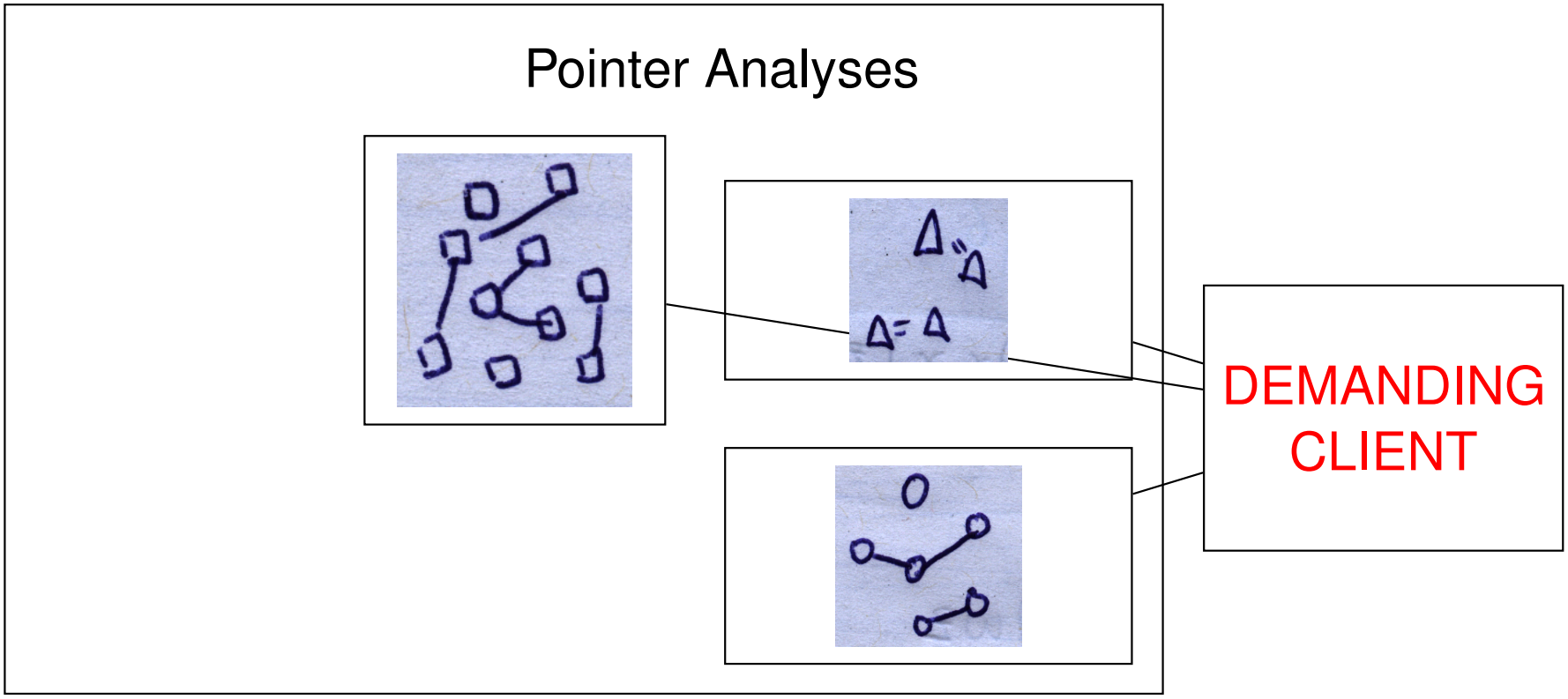
Situation



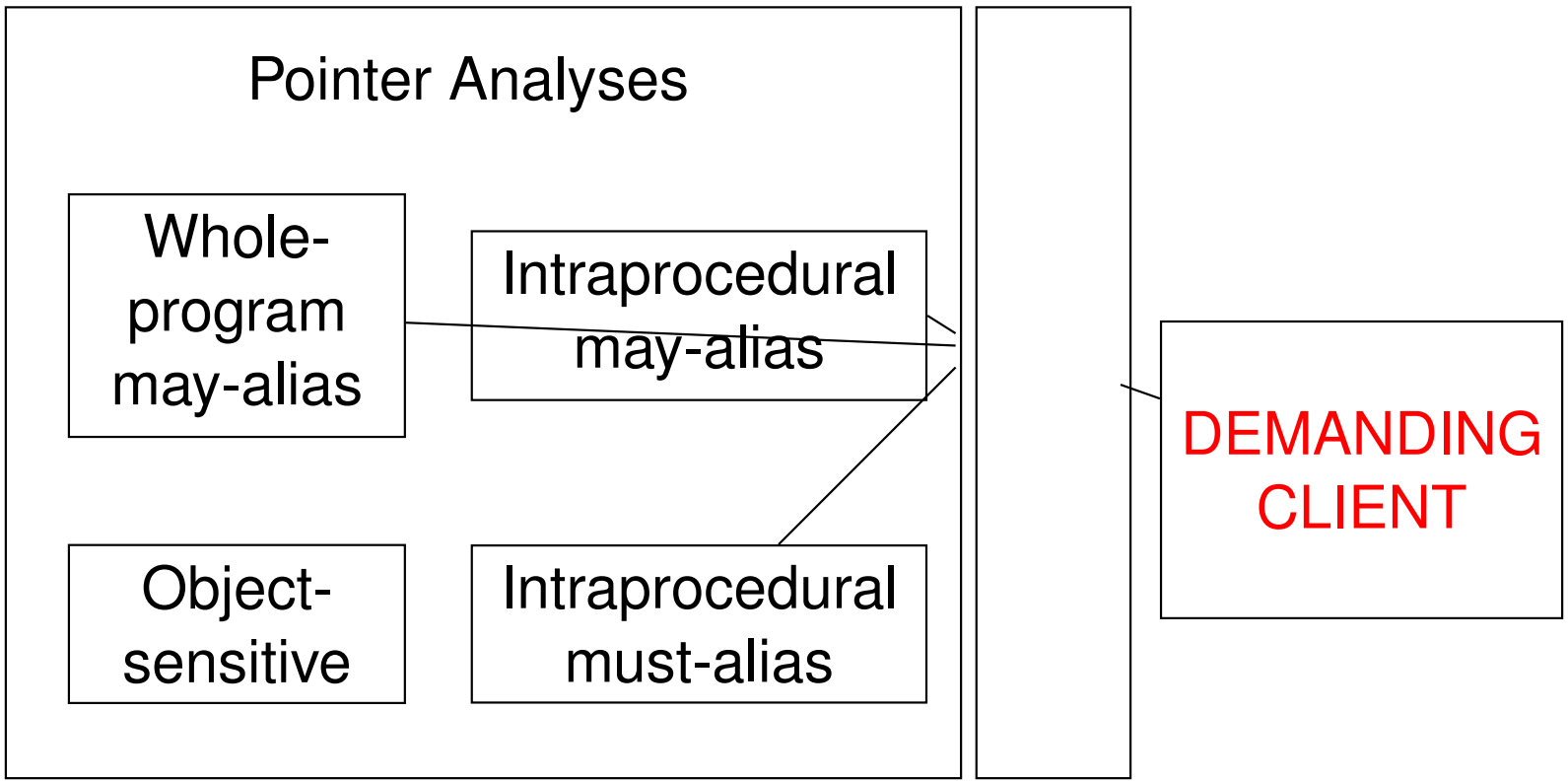
Situation


Pointer Analyses
Whole-program must-alias
Whole-program may-alias
Intraprocedural may-alias
Object-sensitive
Intraprocedural must-alias
DEMANDING CLIENT

Situation



What We Really Want



On the Demanding Client

- property we want to check
- what it looks like at runtime
- how we can mirror this at compile time
- (perhaps bad attempt with variable names?)
- Each pointer analysis gives a different abstraction and interface
- We want a common abstraction

(next few slides also contain this)

Property to Check

Runtime View

Static Analysis Approach

Verify properties at compile time (tracematch workings figure)

How to analyze the property

Automaton

Runtime view

Moving to the compile-time view

Actually the abstraction is none of your business. But we interpose a “object representatives” box between the client and the analyses. We will answer these questions:

1. does object r must-alias object r'?
2. does object r not-must-alias object r'?

Properties of object representatives

1. represent one or more runtime objects;
2. supports r1.mustNotAlias(r2);
3. supports r1.equals(r2): must-alias
4. belongs to a must-aliasing scope

Options

You can tweak these settings:

1. weak vs. strong
2. must-aliasing scope

Object Representatives as Java objects

Computing object representatives

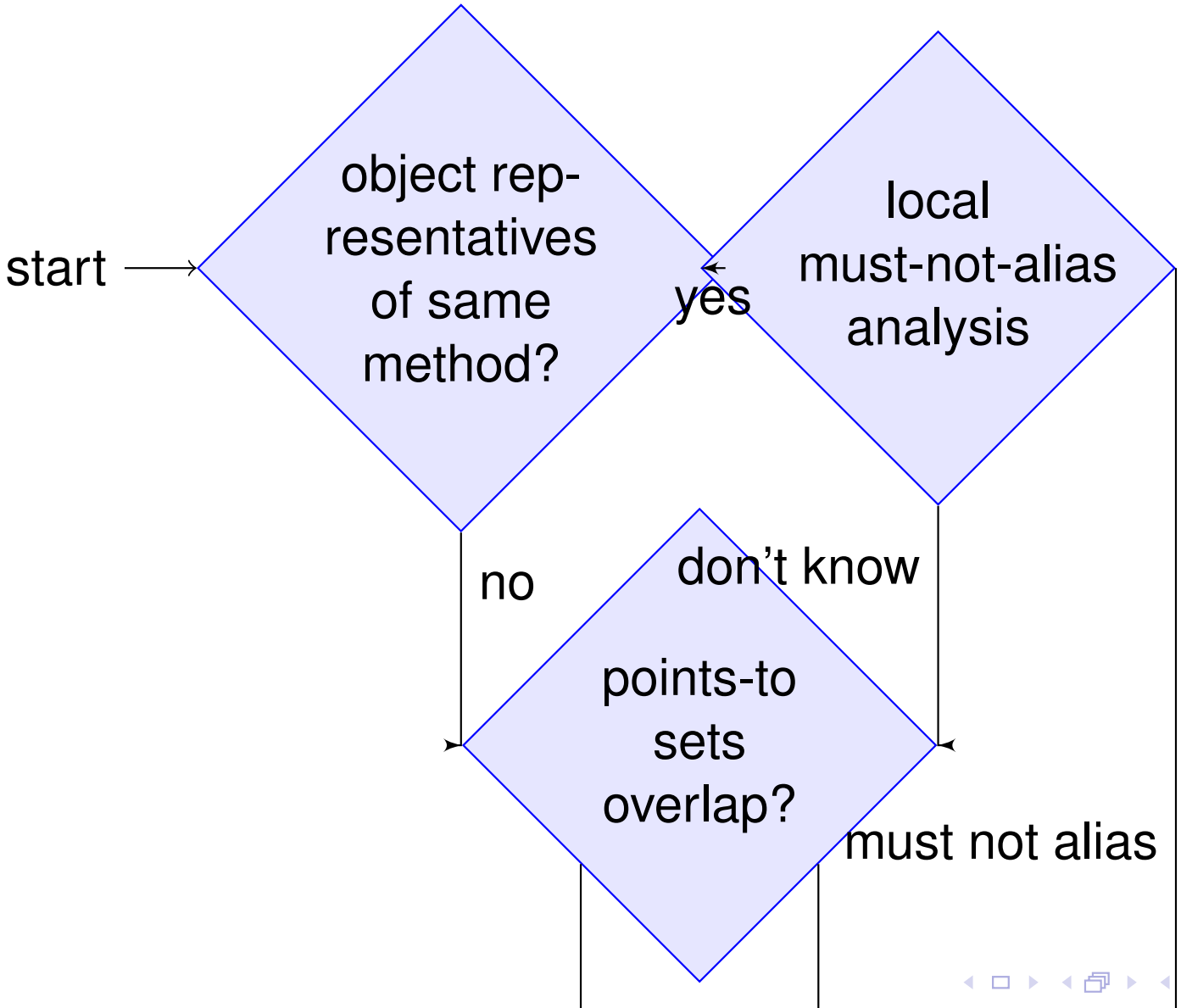
Three analyses, plus combining them.
Go back to the starting figure. Put in the intermediate box for obj representatives.

Intraprocedural must-alias analysis

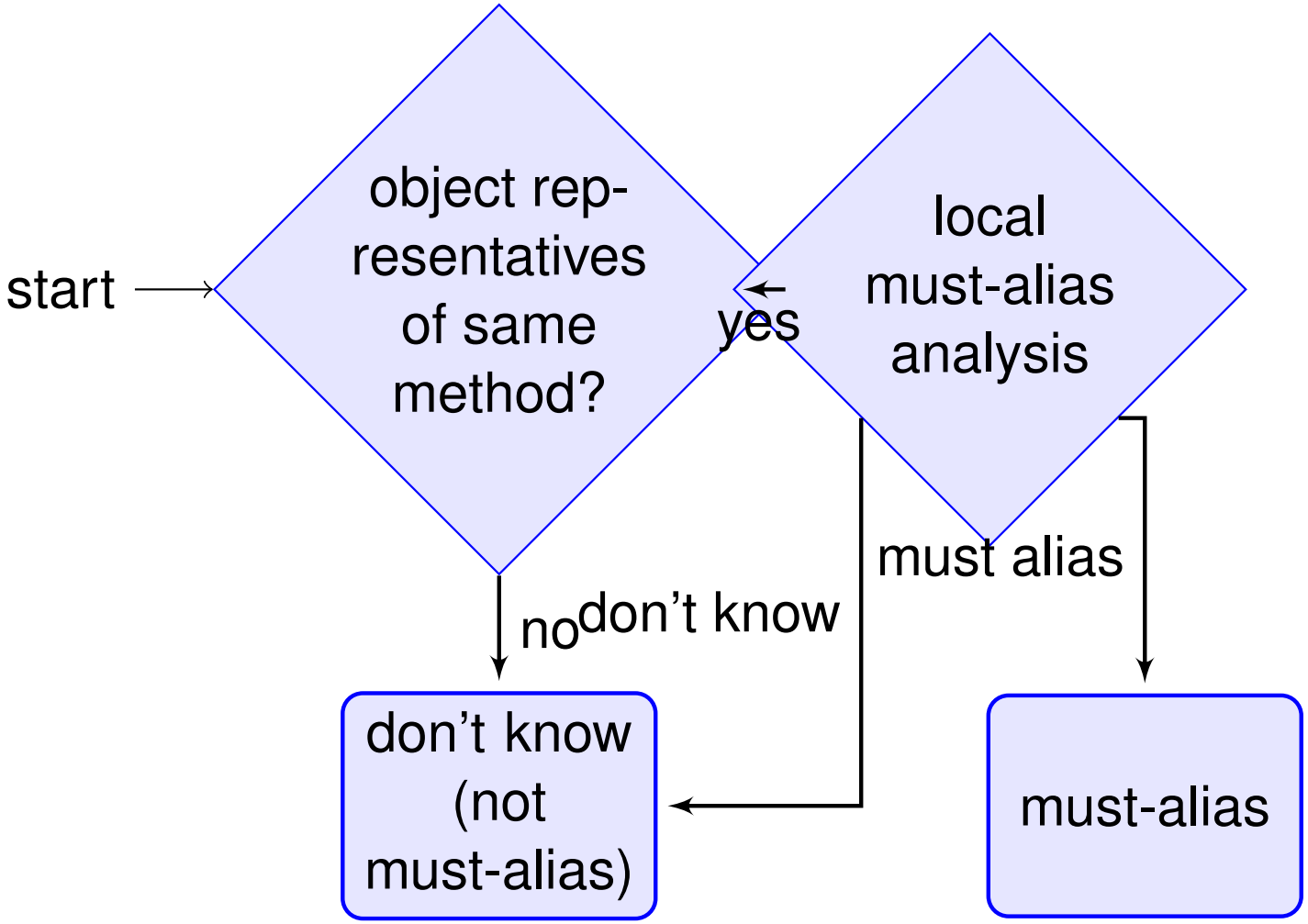
Intraprocedural may-alias analysis

Interprocedural may-alias analysis

Combining analysis results: not-may-alias queries



Combining analysis results: must-alias queries



Related Work

Conclusion