
Fast, Effective Analysis and Optimization of C++ Programs

David F. Bacon and Peter F. Sweeney

IBM T.J. Watson Research Center

IBM C++ Optimization Group

- ◆ Michael Burke
- ◆ Michael Hind
- ◆ G. Ramalingam
- ◆ Harini Srinivasan

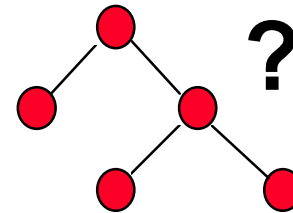
Introduction

- ◆ C++ has powerful features
 - Virtual Function Calls
 - Virtual Base Classes
 - Dynamic Casts
- ◆ But they cost
 - Time
 - Space

Optimization

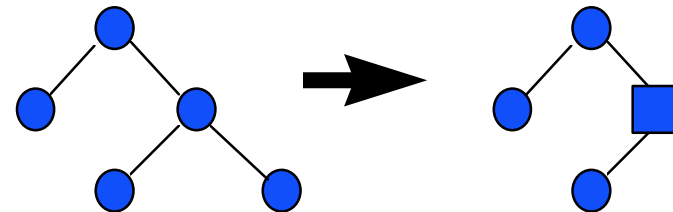
- ◆ **Analysis**

- ask the right questions



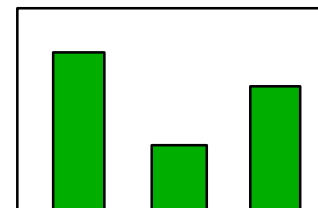
- ◆ **Transformation**

- use the answers

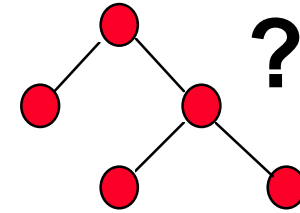


- ◆ **Evaluation**

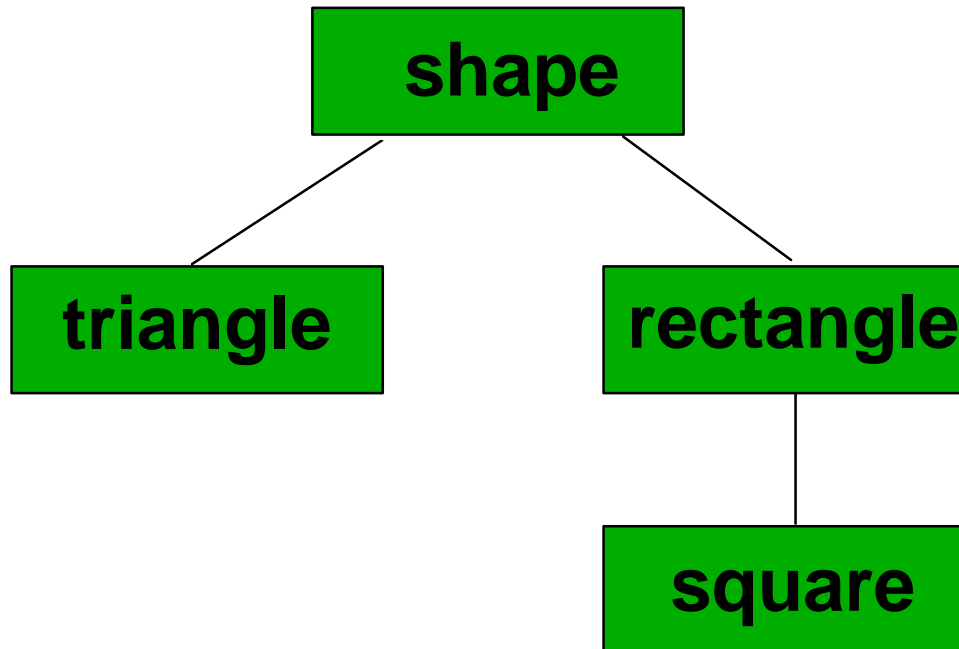
- how well does it work?



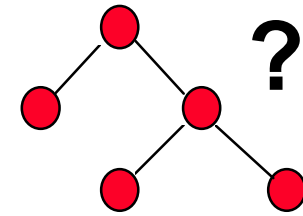
Analysis Framework



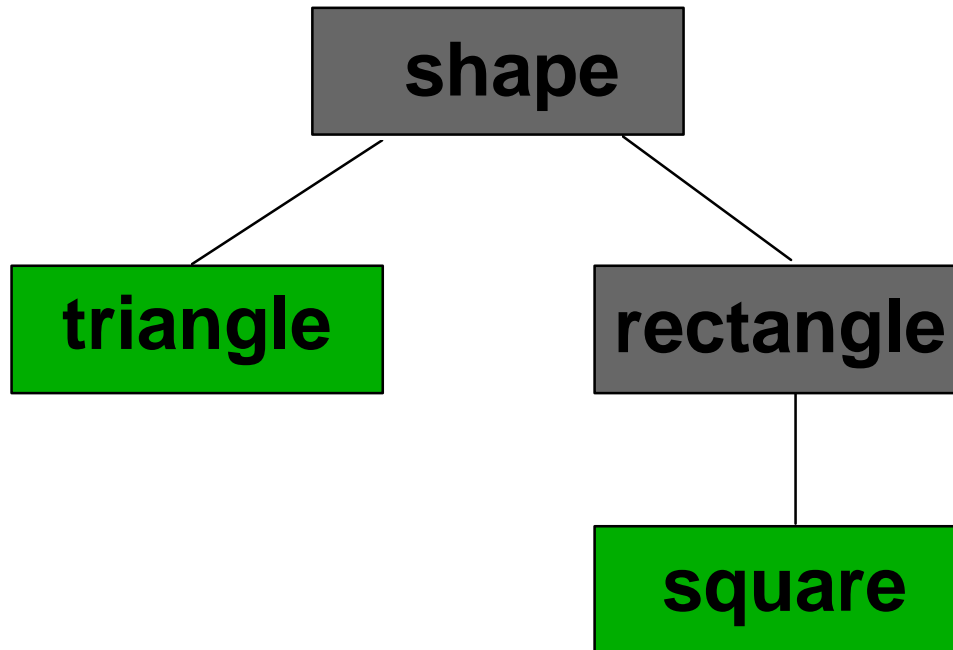
- ◆ What classes are created?



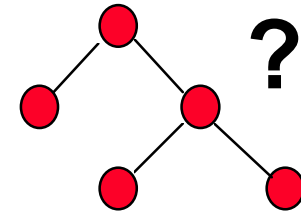
Analysis Framework



- ◆ What classes are created?



Analysis Framework

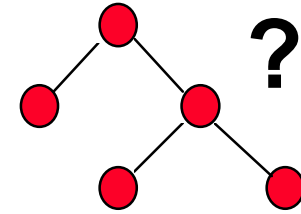


- ◆ What are the classes of each value?

`object->draw() ;` **{ triangle }**

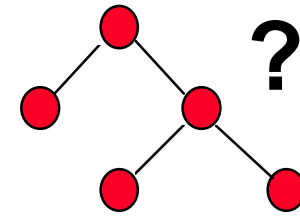
`sign.reshape() ;` **{ rectange, square }**

Class Hierarchy Analysis

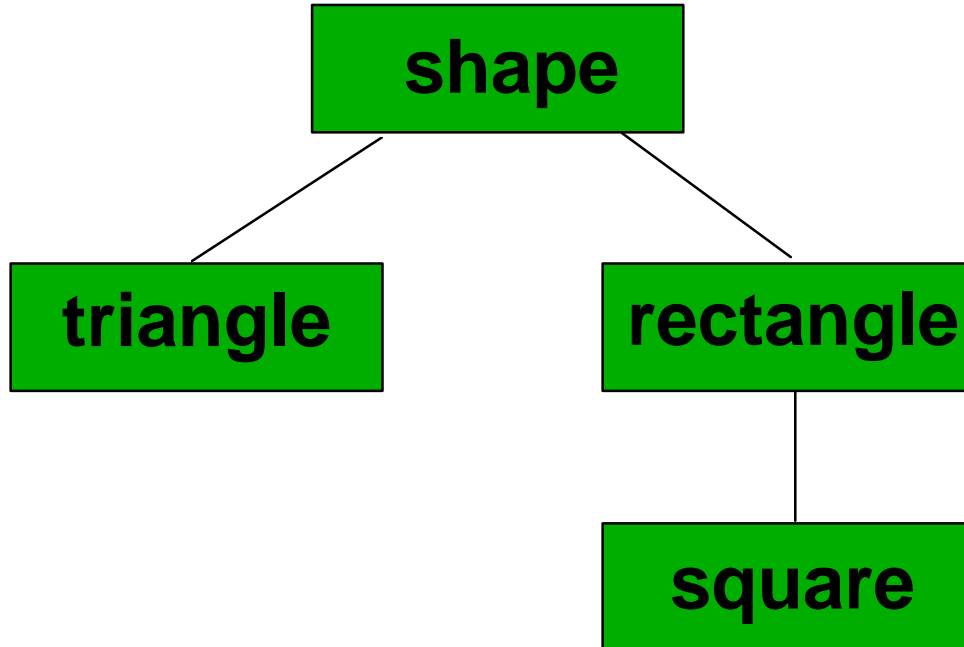


- ◆ What classes are created?
 - all classes
- ◆ What are the classes of each value?
 - the classes derived from the declared type

Class Hierarchy Analysis



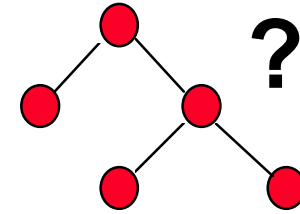
- ◆ the classes derived from the declared type



```
rectangle* r;  
{rectangle,square}
```

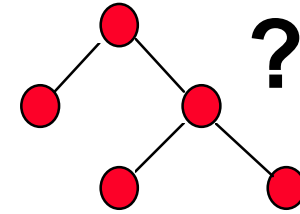
```
square * s;  
{square}
```

Rapid Type Analysis

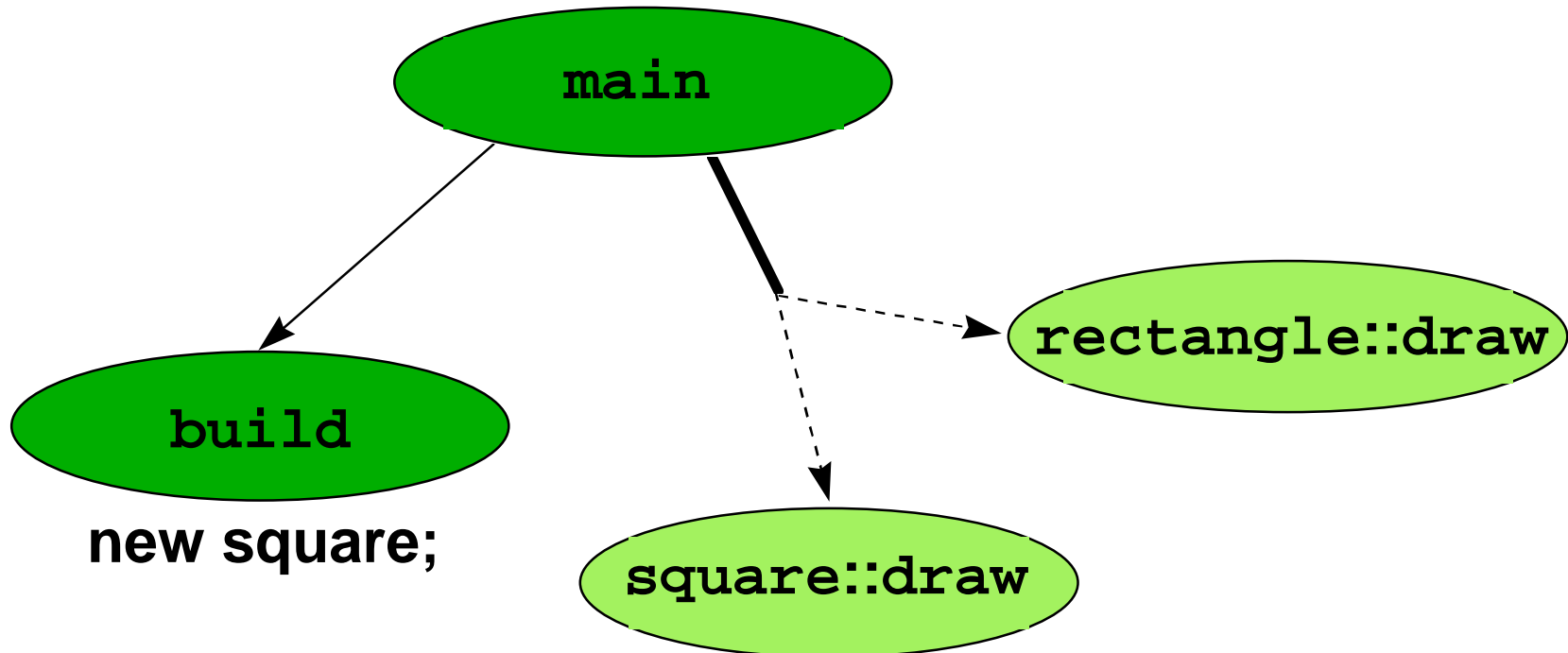


- ◆ What classes are created?
 - the classes in the call graph
- ◆ What are the classes of each value?
 - the classes derived from the declared type
 - that have been created

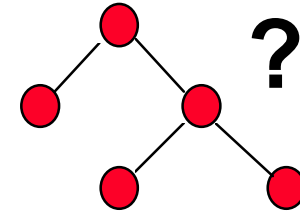
Rapid Type Analysis



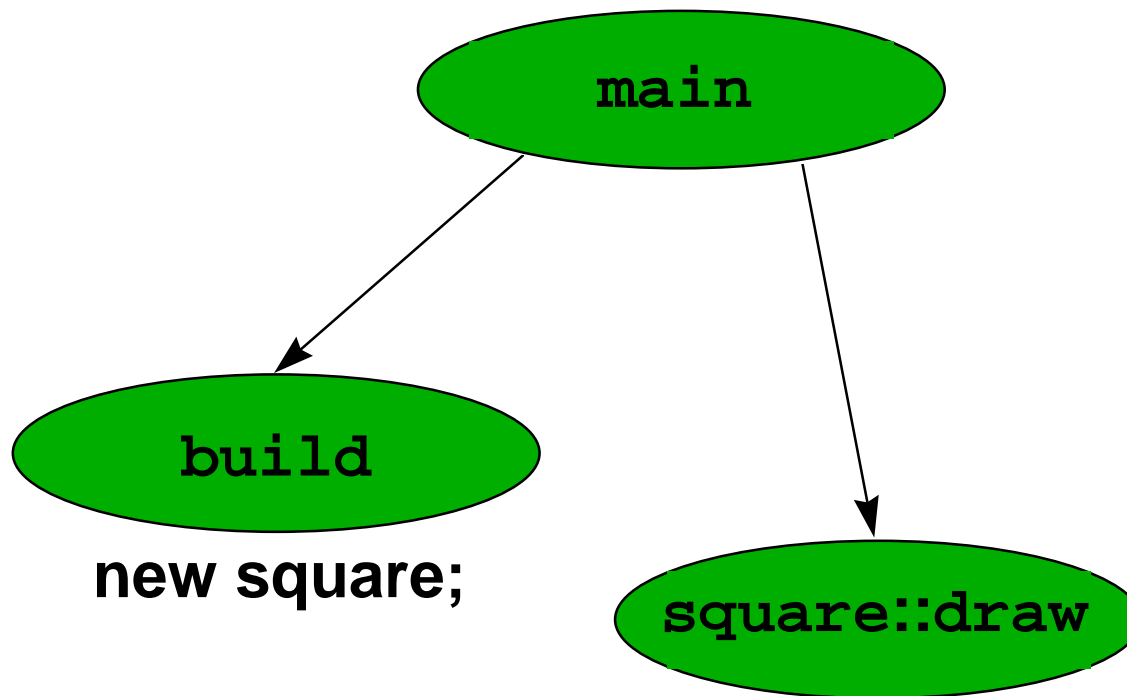
- ◆ the classes in the call graph



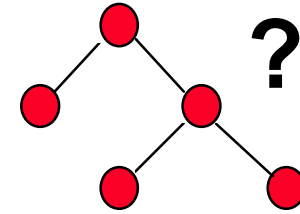
Rapid Type Analysis



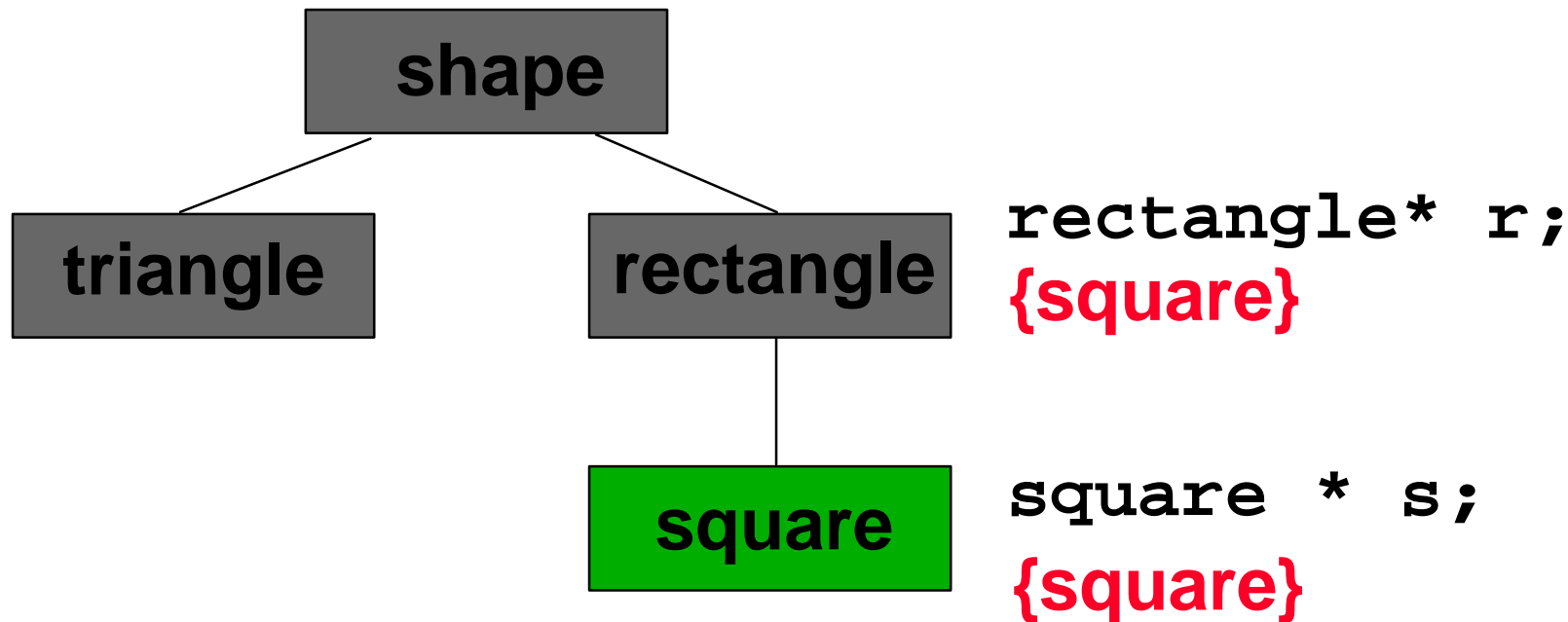
- ◆ the classes in the call graph: **{square}**



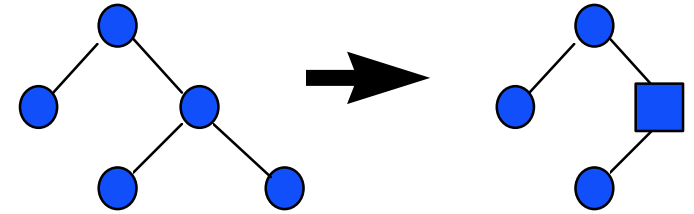
Rapid Type Analysis



- ◆ What are the classes of each value?
 - the classes derived from the declared type
 - that have been created

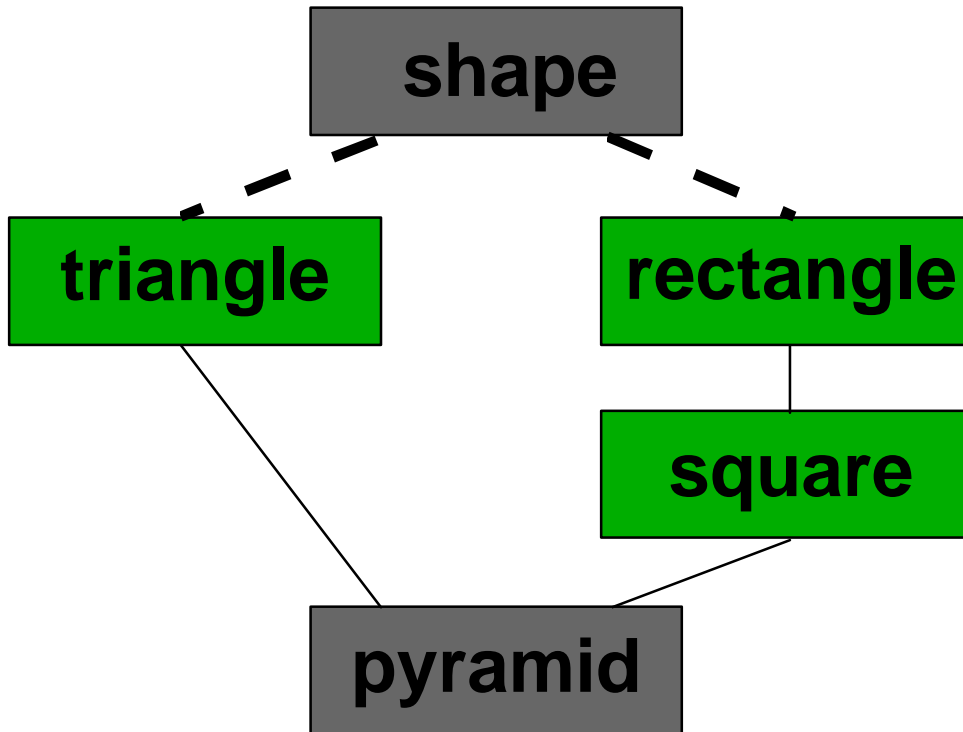
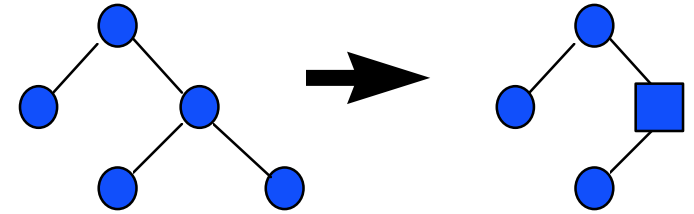


Transformation



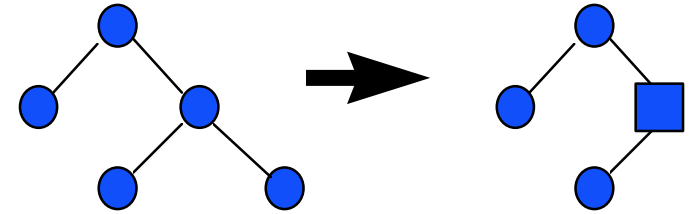
- ◆ virtual base classes to non-virtual bases
- ◆ dynamic casts to static casts
- ◆ virtual calls to direct calls

Virtual Bases



- ◆ Is **shape** multiply inherited?
- ◆ Is that class live?
- ◆ If not, make **shape** a non-virtual base

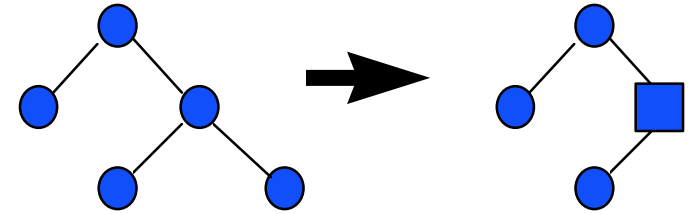
Dynamic Casts



```
foo(shape* p) {dynamic_cast<square> p;}
```

- ◆ What object types could **p** be?
- ◆ Is **square** the only possible type?
- ◆ If so, do a compile-time cast

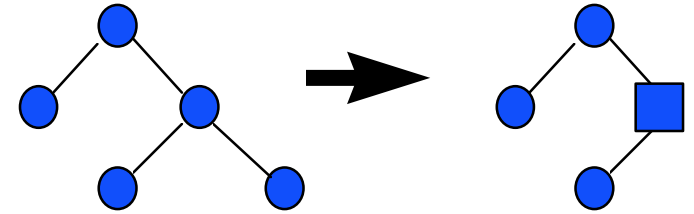
Virtual Calls



`p->draw() ;`

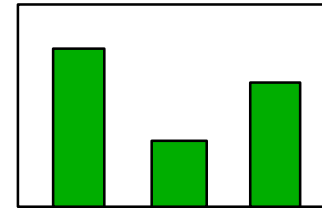
- ◆ What object types could **p** be?
- ◆ What **draw()** functions are defined?
- ◆ If only 1 **draw()** function
 - resolve the call

Benefits of VFR



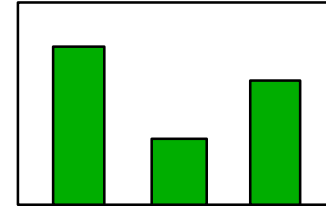
- ◆ Time
 - direct calls are faster
 - in-lining is possible
- ◆ Space
 - unused functions can be eliminated
- ◆ Understandability
 - programmer doesn't see unused code

Evaluation



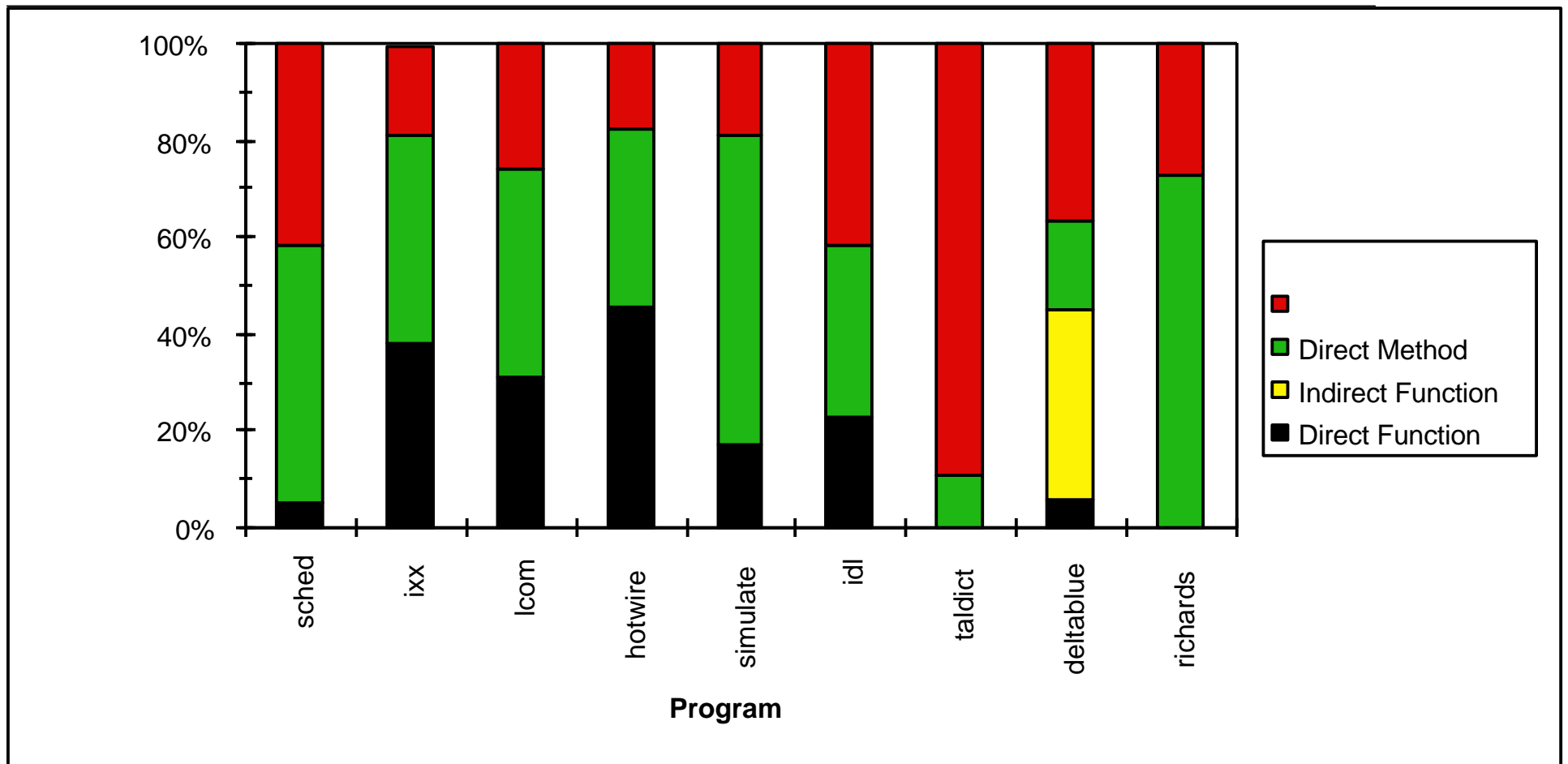
- ◆ Compare 3 analysis algorithms
 - Rapid Type Analysis
 - Class Hierarchy Analysis
 - Unique Name
- ◆ On virtual function resolution

Methodology

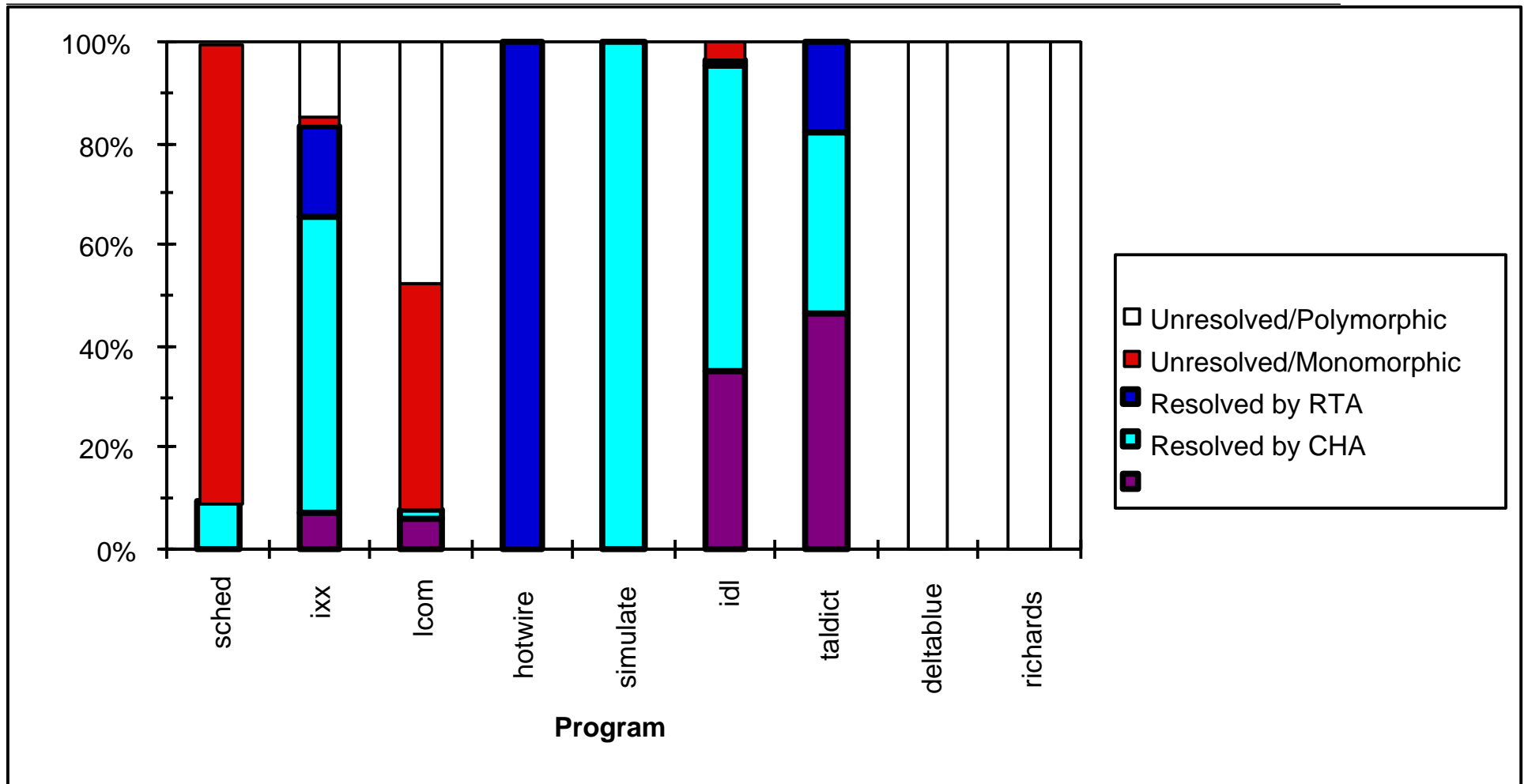


- ◆ Use real programs as benchmarks
 - 7 large programs 5000-20000 lines each
- ◆ Determine causes of
 - success
 - failure
- ◆ Evaluate against best possible algorithm

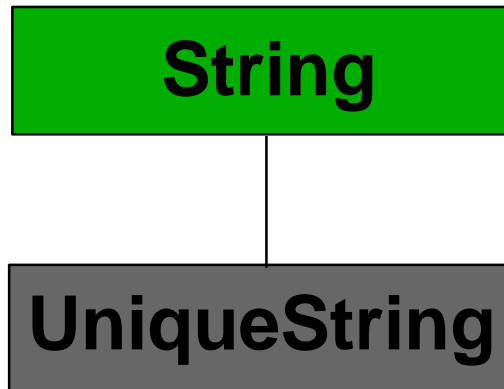
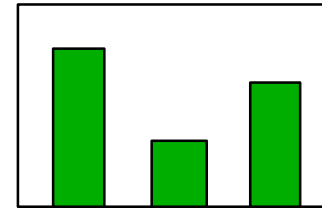
Dynamic Call Types



Resolved Dynamic Calls

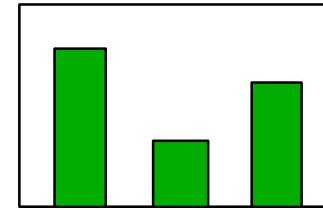


Why RTA Wins: **ixx**

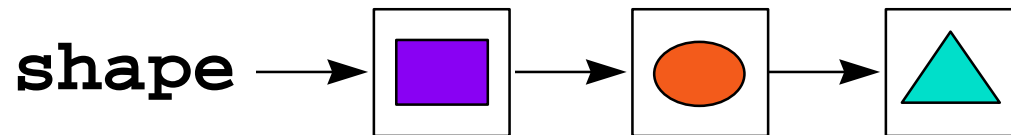


- ◆ RTA finds unused classes
- ◆ String ops are in inner loops
- ◆ Similar win for
 - `taldict`
 - `hotwire`

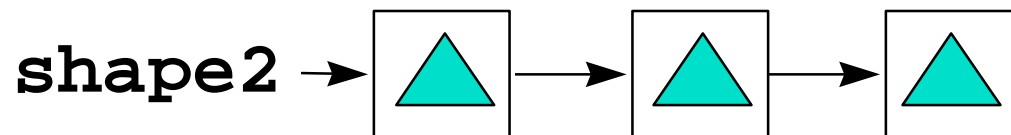
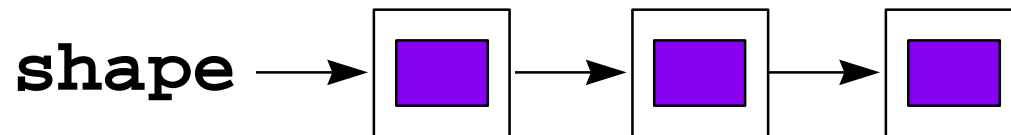
Why RTA Loses: **sched**



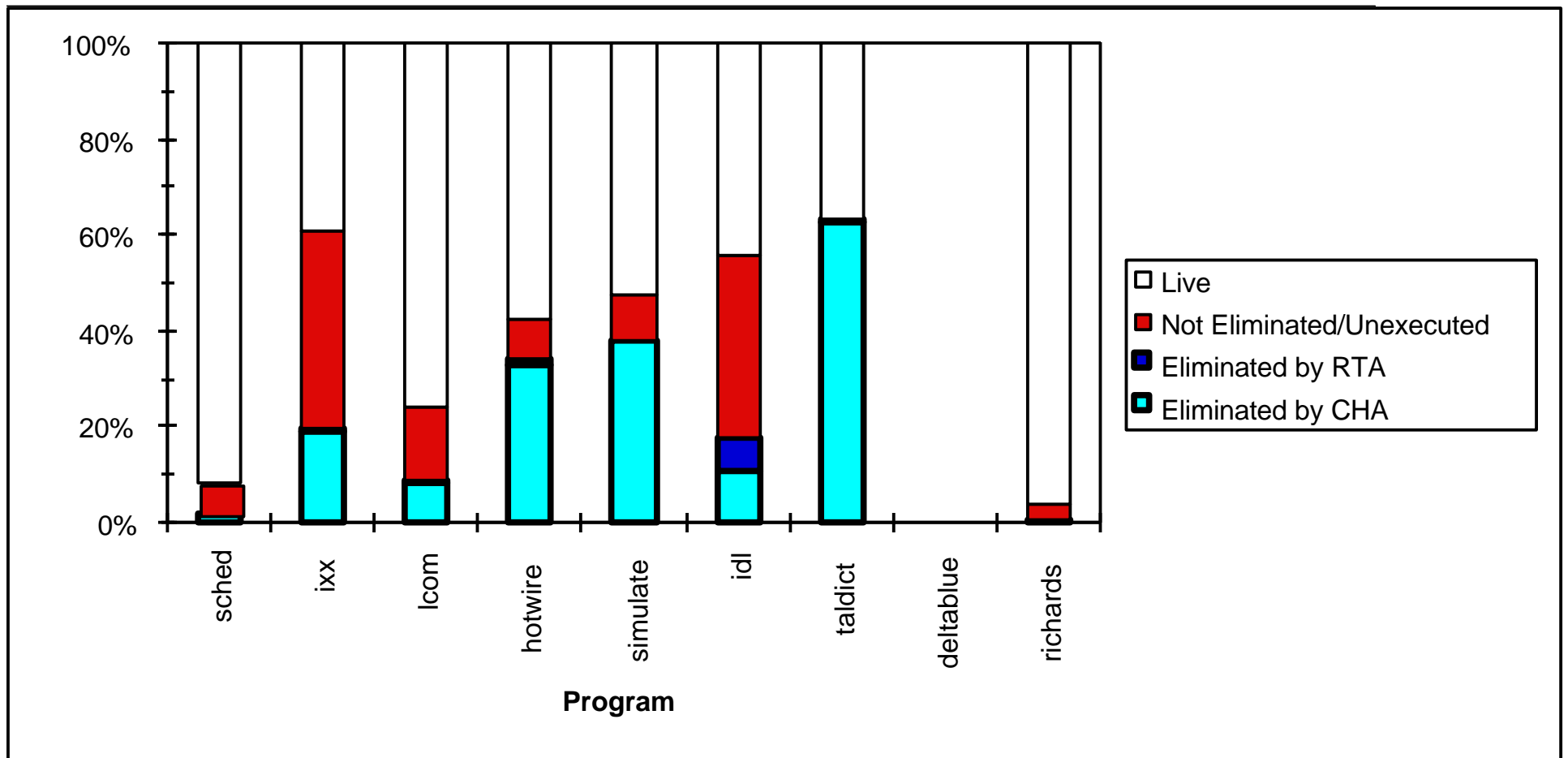
◆ True Polymorphism



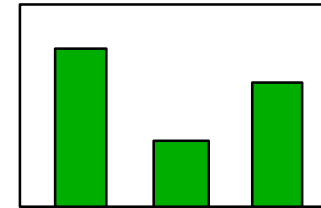
◆ Disjointed Polymorphism



Reduction in Code Size

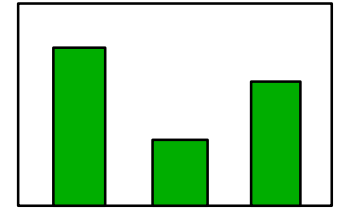


Speed of Analysis



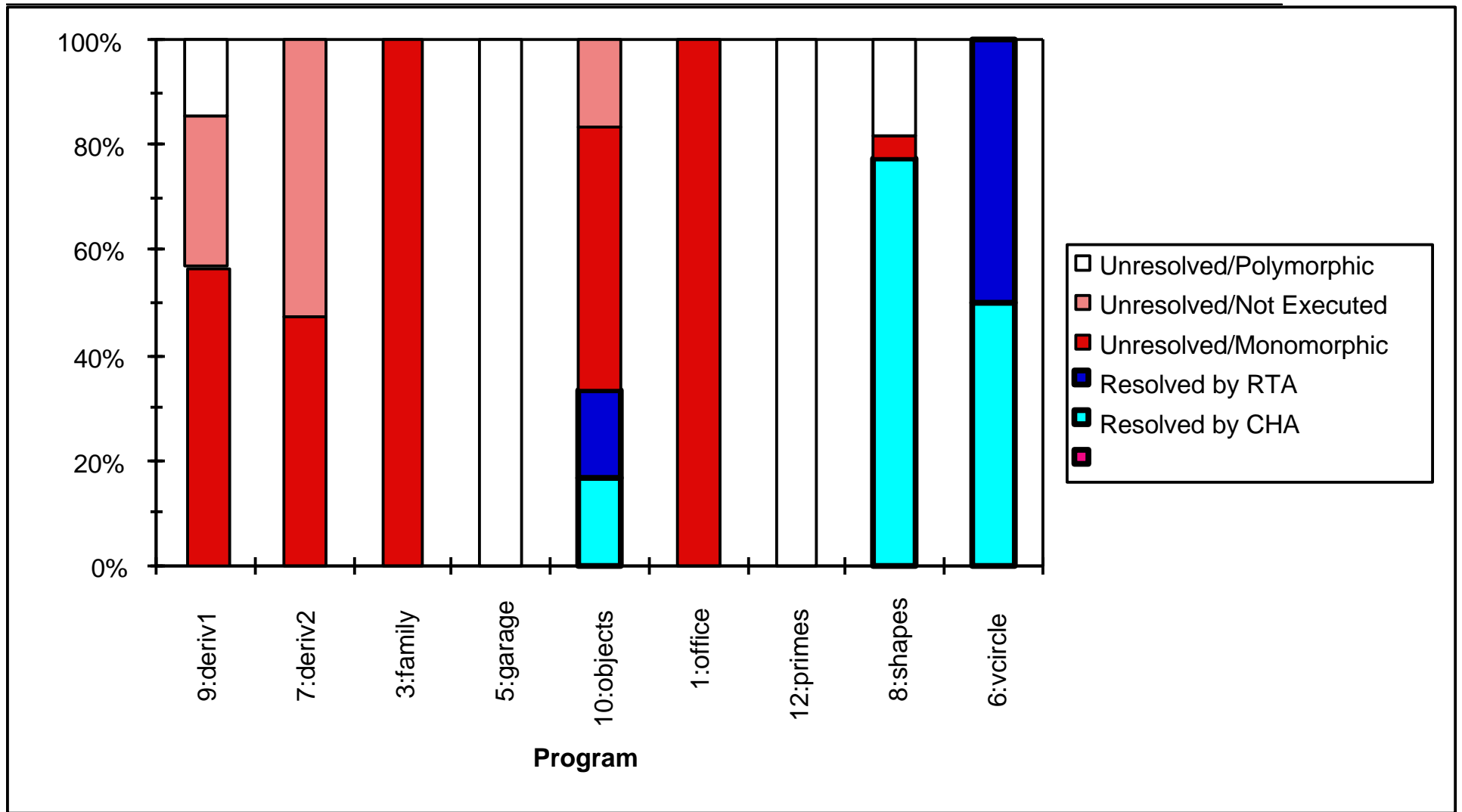
Benchmark	Size (lines)	Analysis Time (s)	Overhead (%)
sched	5,712	1.94	0.1%
ixx	11,157	5.22	1.4%
lcom	17,278	6.50	3.0%
hotwire	5,335	2.06	1.3%
simulate	6,672	2.75	5.6%
idl	30,288	6.42	1.4%
taldict	11,854	1.78	4.0%
deltablue	1,250	0.44	2.4%
richards	606	0.32	3.6%

Comparison: Alias Analysis



- ◆ Best precision from a static analysis
- ◆ Complex algorithm--expensive to implement
- ◆ Slow
 - between 0.4 and 55 source lines analyzed/second
 - RTA is 45 to 8250 times faster

Comparison: Alias Analysis

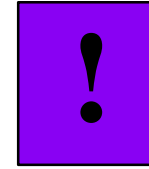


Contributions



- ◆ Analysis Framework
- ◆ Rapid Type Analysis (RTA) algorithm
- ◆ Evaluation showing power of RTA

Rapid Type Analysis



- ◆ RTA is *effective*:
 - resolves 71% of virtual function calls
 - reduces code size by 25%
- ◆ RTA is *fast*:
 - analyzes 3300 lines per second
- ◆ RTA is often as good as alias analysis