Implementing Object-Oriented Languages - Part 2

Y.N. Srikant
Computer Science and Automation
Indian Institute of Science
Bangalore 560 012

NPTEL Course on Compiler Design

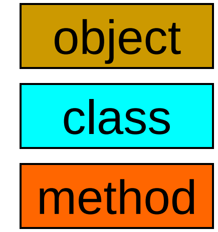
IISc

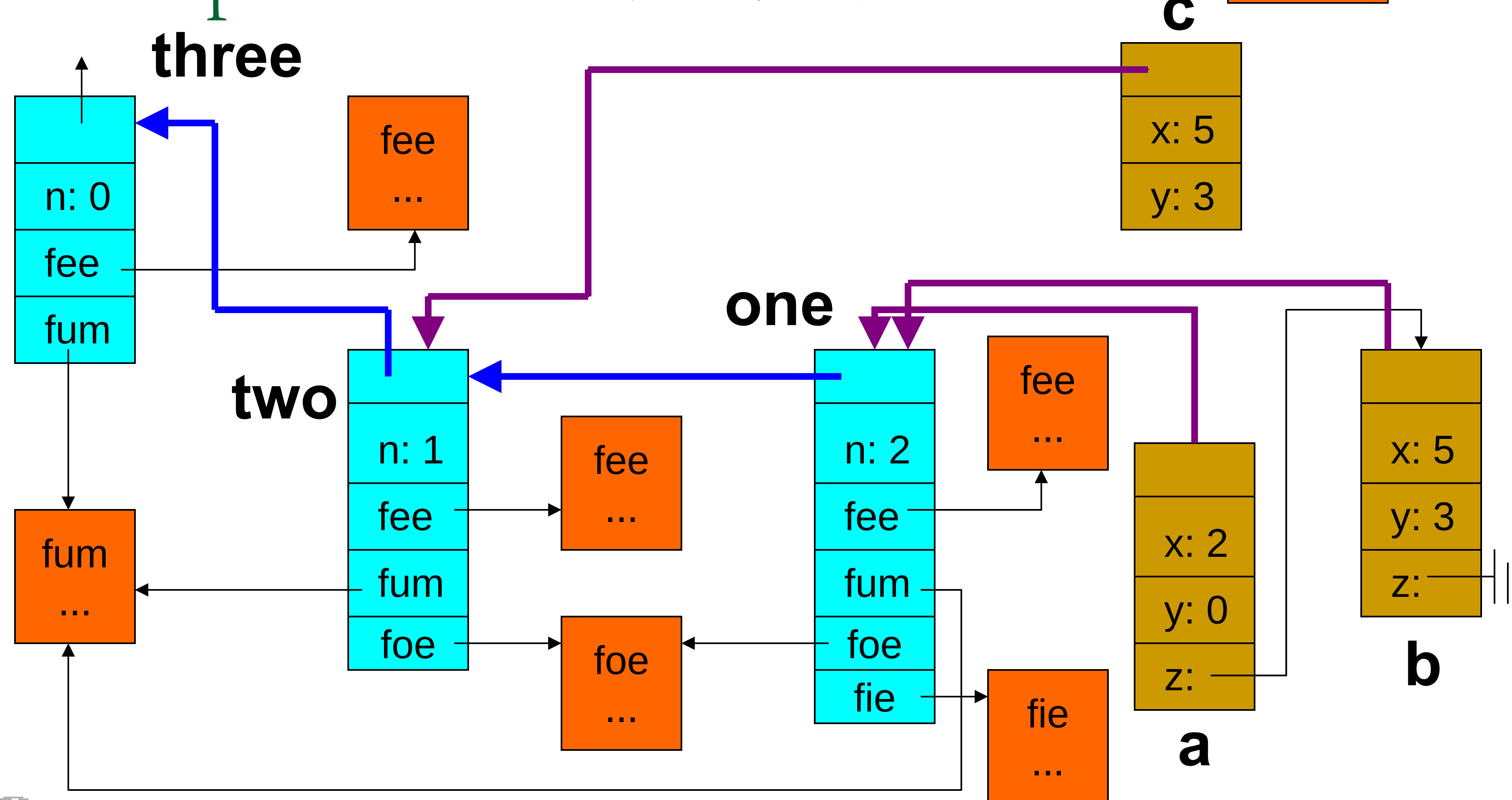
Outline of the Lecture

- Language requirements
- Mapping names to methods
- Variable name visibility
- Code generation for methods
- Simple optimizations
- **Parts of this lecture are based on the book, "Engineering a Compiler", by Keith Cooper and Linda Torczon, Morgan Kaufmann, 2004, sections 6.3.3 and 7.10.**

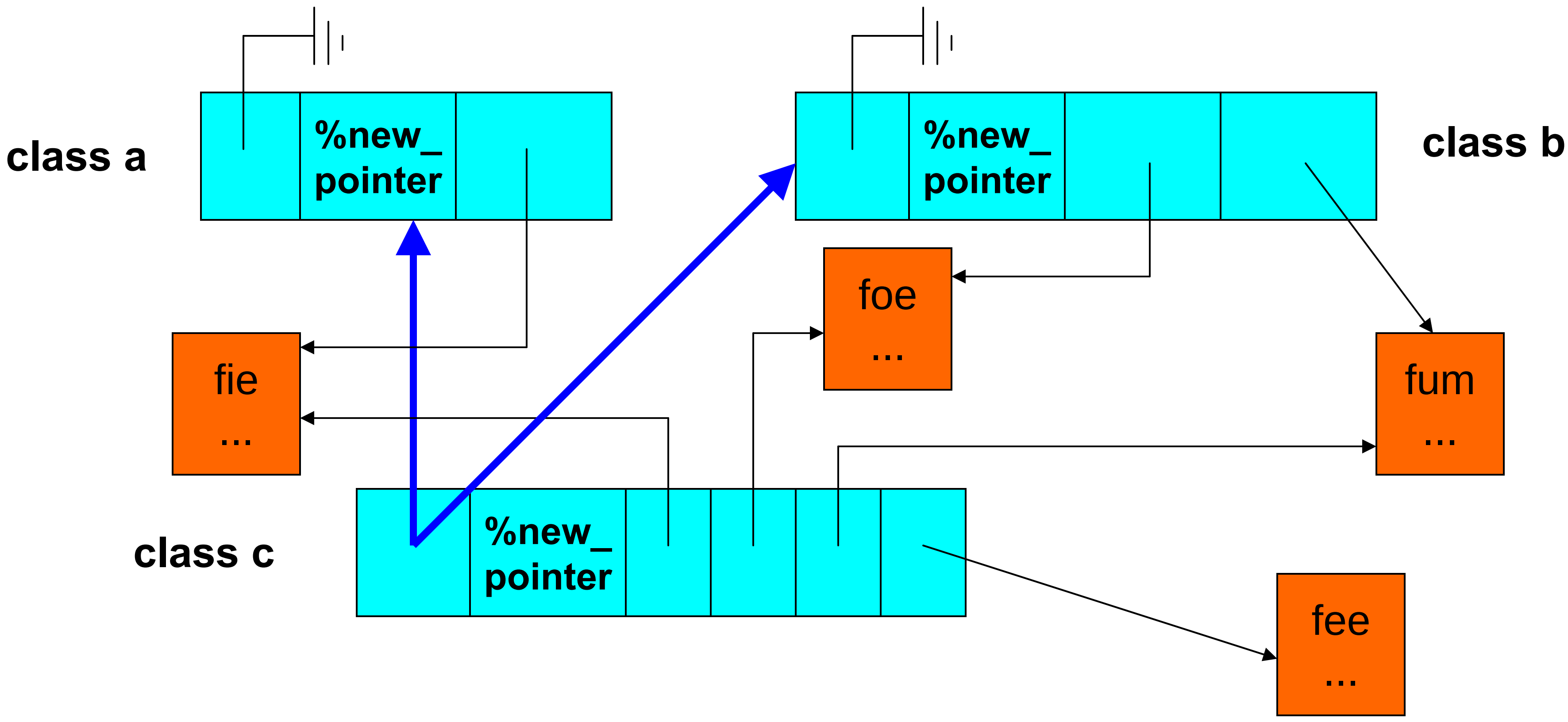
Topics 1,2,3, and, 4 were covered in Part 1 of the lecture.

Example of Class Hierarchy with Complete Method Tables





Implementing Multiple Inheritance



Implementing Multiple Inheritance

object layout for objects of class a

class pointer	*a* data members

object layout for objects of class b

class pointer	*b* data members

object layout for objects of class c

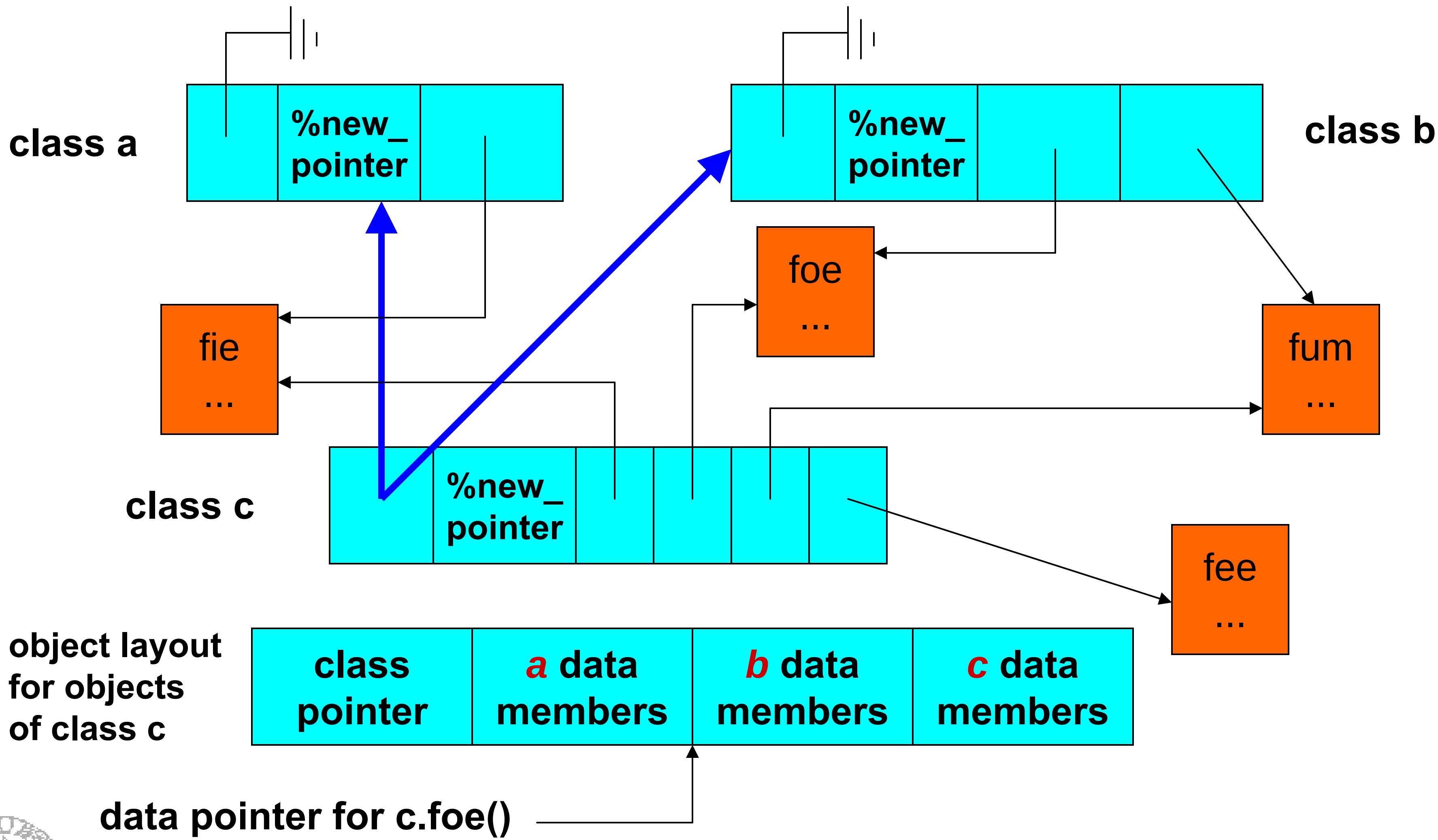
class pointer	*a* data members	*b* data members	*c* data members

Implementing Multiple Inheritance - Fixed Offset Method

object layout for objects of class c	class pointer	*a* data members	*b* data members	*c* data members

- Record the constant offset in the method table along with the methods
  - Offsets for this example are as follows:
    - (c) fee : 0, (a) fie: 0, (b) foe : 8, (b) fum : 8, assuming that instance variables of class a take 8 bytes
  - Generated code adds this offset to the receiver's pointer address before invoking the method

Implementing Multiple Inheritance



Implementing Multiple Inheritance - Trampoline Functions

- Create trampoline functions for each method of class b
  - A function that increments this (pointer to receiver) by the required offset and then invokes the actual method from b.
  - On return, it decrements the receiver pointer, if it was passed by reference

Implementing Multiple Inheritance

- Trampolines appear to be more expensive than the fixed offset method, but not really so
  - They are used only for calls to methods inherited from b
    - In the other method, offset (possibly 0) was added for all calls
  - Method inlining will make it better than option 1, since the offset is a constant
- Finally, a duplicate class pointer (pointing to class c) may need to be inserted just before instance variables of b (for convenience)

Fast Type Inclusion Tests – The need

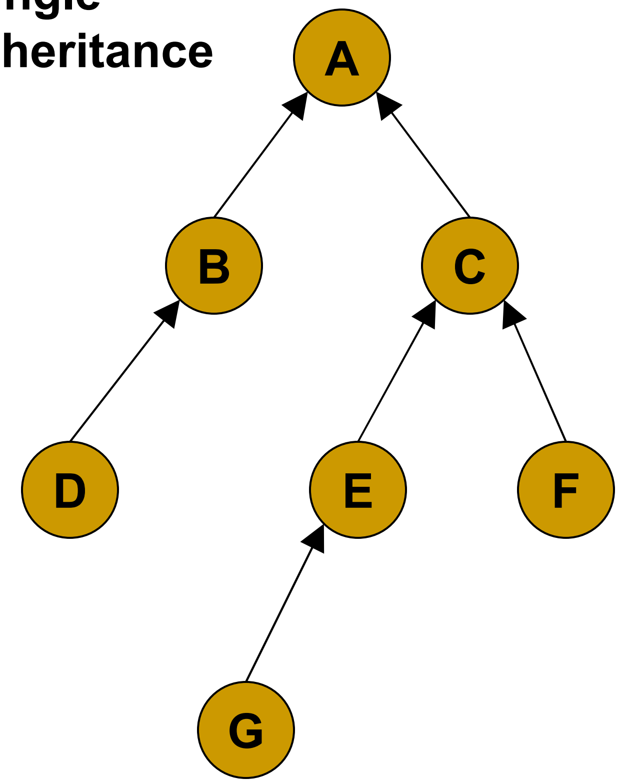
- If class Y is a subclass of class X
  - X a = new Y(); //a is of type base class of Y, okay
    // other code omitted
    Y b = a; // a holds a value of type Y
  - The above assignment is valid, but the following is not
  - X a = new X();
    // other code omitted
    Y b = a; // a holds a value of type X
- Runtime type checking to verify the above is needed
- Java has an explicit instanceof test that requires a runtime type checking

Fast Type Inclusion Tests – Searching the Class Hierarchy Graph

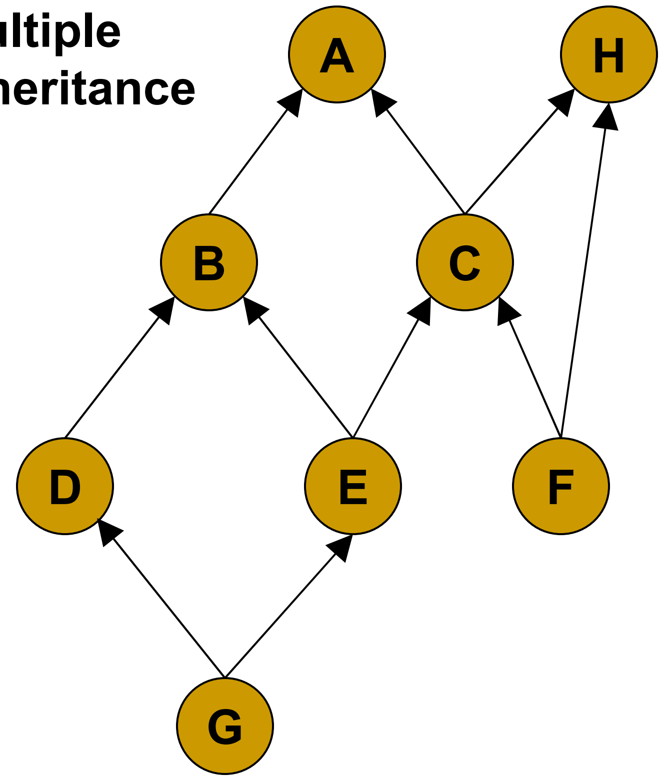
- Store the class hierarchy graph in memory
- Search and check if one node is an ancestor of another
- Traversal is straight forward to implement only for single inheritance
- Cumbersome and slow for multiple inheritance
- Execution time increases with depth of class hierarchy

Class Hierarchy Graph - Example

Single inheritance



Multiple inheritance



Fast Type Inclusion Tests – Binary Matrix

Class types (columns) / Class types (rows)

	C1	C2	C3	C4	C5
C1	**0**	**1**	**0**	**0**	**1**
C2	**0**	**0**	**1**	**0**	**1**
C3	**1**	**0**	**0**	**1**	**0**
C4	**1**	**0**	**0**	**0**	**1**
C5	**0**	**0**	**1**	**0**	**0**

Tests are efficient, but Matrix will be large in practice. The matrix can be compacted, but this increases access time. This can handle multiple inheritance also.

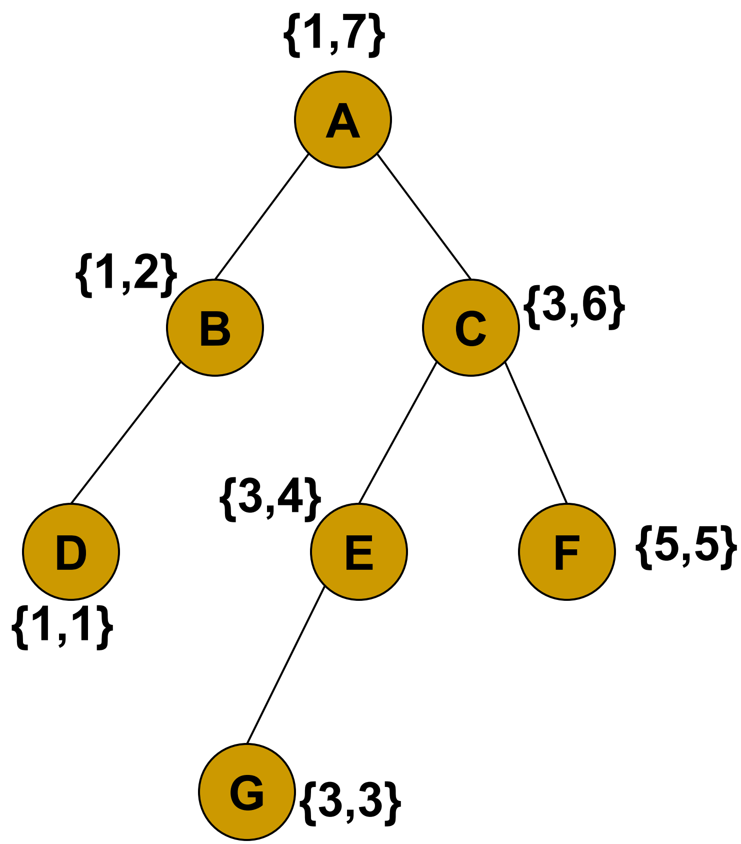
BM [Ci, Cj] = 1, iff Ci is a subclass of Cj

Relative (Schubert's) Numbering

{ l_a, r_a } for a node **a** :
r_a is the ordinal number of the node **a** in a **postorder traversal** of the tree. Let $\trianglelefteq$ denote **"subtype of"** relation. All descendants of a node are subtypes of that node.
$\trianglelefteq$ is reflexive and transitive.
$l_a = \min \{ r_p \mid p$ is a descendant of $a \}$.

Now, $a \trianglelefteq b$, iff $l_b \leq r_a \leq r_b$.

This test is very fast and is O(1). Works only for single inheritance.
Extensions to handle multiple inheritance are complex.



Devirtualization – Class Hierarchy Analysis

- Reduces the overhead of virtual method invocation
- Statically determines which virtual method calls resolve to a single method
- Such calls are either inlined or replaced by static calls
- Builds a class hierarchy and a call graph

Class Hierarchy Analysis

```
class X extends object {
        void f1() {. . .}
        void f2() {. . .}
}
class Y extends X {
        void f1() {. . .}
}
class Z extends X {
        void f1() {. . .}
        public static void main(...) {
                X a = new X(); Y b = new Y();
                Z c = new Z();
                 if (...) a = c;
                // other code
                a.f1();   b.f1();   b.f2();
        }
}
```