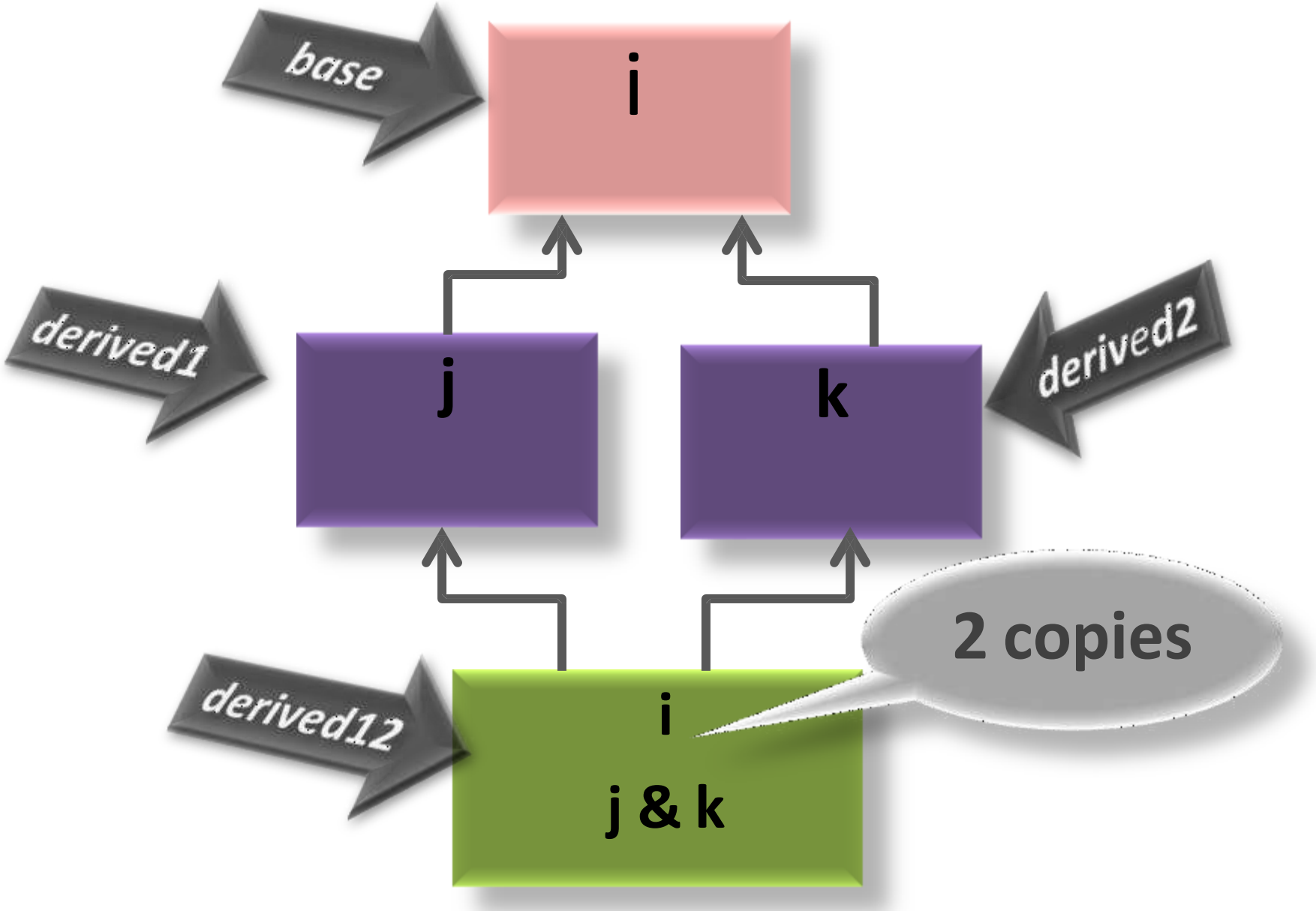
CSC127 Virtual Functions and Abstract Classes

Hierarchy of Classes



Remedy……?

Hierarchy of Classes

Virtual Base Classes

- Used to **prevent** multiple copies of the base class from being present in an object derived from those objects by declaring the base class as **virtual when it is inherited.**

- **Syntax:**

```
class derived : virtual public base
{   .  .  .   };
```

Virtual Functions

- “A virtual function is a member function that is declared **within** a **base class** and **redefined** by a **derived class**.”

- Virtual functions implements the **"one interface, multiple methods"** philosophy under polymorphism.

Virtual Functions

- The virtual function within the base class defines the form of the **interface** to that
- function.

- Each redefinition of the virtual function by a derived class implements its operation as it relates specifically to the derived class. That is, the redefinition creates a specific method.

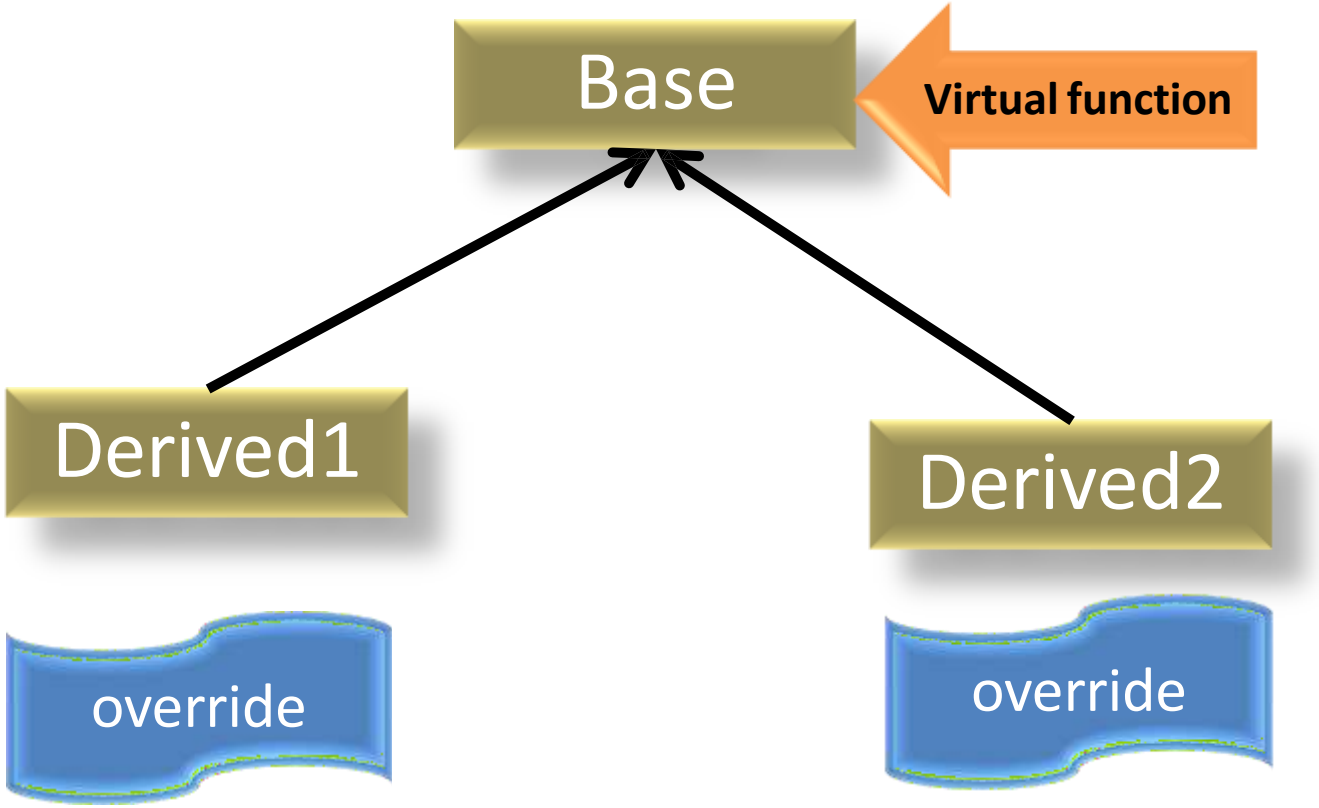
Virtual Functions

- To create a virtual function, precede the function‘s declaration in the base class with the keyword **virtual.**

- **Example:**

```
class base {
    public:
        virtual void member_func(){   }
};
```

Virtual Functions



Virtual Functions

- When accessed **"normally"** virtual functions behave just like any other type of class member function.

- But virtual functions' importance and capacity lies in supporting the **run-time polymorphism** when they accessed via a pointer.

Virtual Functions

- **How to implement run-time polymorphism?**
  - create base-class pointer can be used to point to an object of any class derived from that base
  - initialize derived object(s) to base class object.

- Based upon which derived class objects' assignment to the base class pointer, c++ determines which version of the virtual function to be called. And this determination is made at run time.

Virtual Functions

- The **redefinition of a virtual function** by a derived class appears similar to **function overloading**?
- No

- The prototype for a redefined virtual function must match exactly the prototype specified in the base class.

Virtual Functions

Restrictions:

- All aspects of its prototype must be the **same** as base class virtual function.
- Virtual functions are of **non-static** members.
- Virtual functions can not be **friends**.
- **Constructor** functions can**not** be **virtual**.

•But **destructor** functions **can** be **virtual**.

NOTE:

Function overriding is used to describe virtual function redefinition by a derived class.

Destructor functions can be virtual?

- Yes.
- In large projects, the destructor of the derived class was not called at all.
- This is where the virtual mechanism comes into our rescue. By making the Base class Destructor virtual, both the destructors will be called in order.

Function overriding

- “A function overriding is a process in which a member function that is declared within a base class and redefined by a derived class to implement the "one interface, multiple methods" philosophy under polymorphism.”

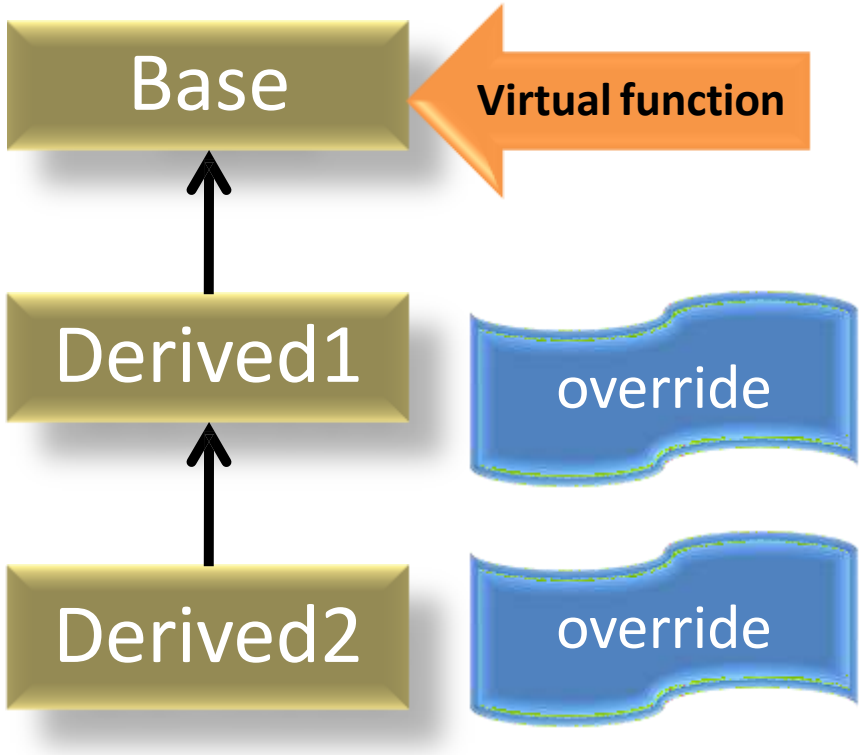
Calling a Virtual Function Through a Base Class Reference

- Since **reference is an implicit pointer**, it can be used to access virtual function.

- When a virtual function is called through a **base-class reference**, the version of the function executed is determined by the **object** being **referred** to at the time of the call.

The Virtual Attribute Is Inherited

- When a virtual function is inherited, its virtual nature is also inherited.

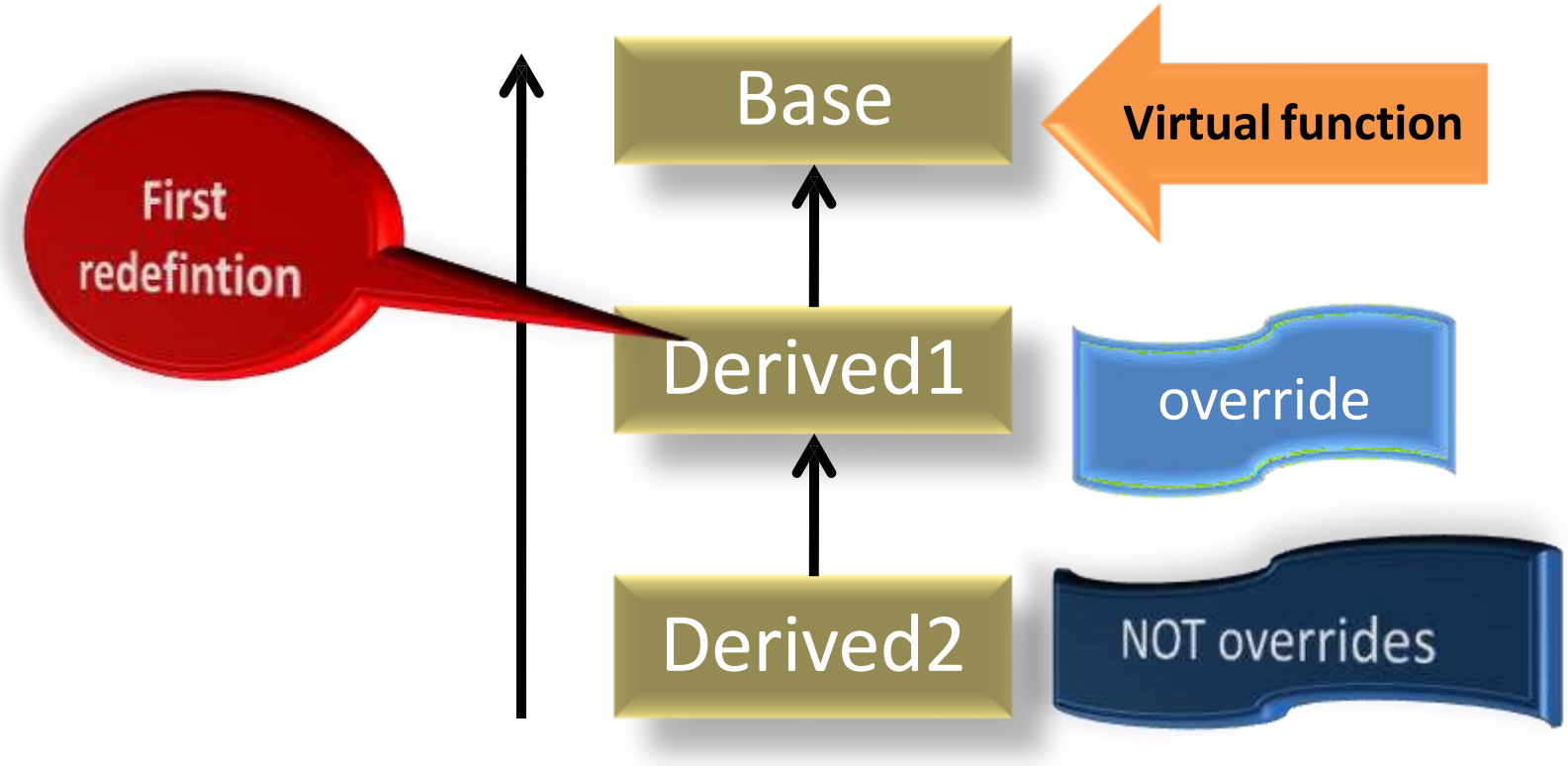


Virtual Functions Are Hierarchical

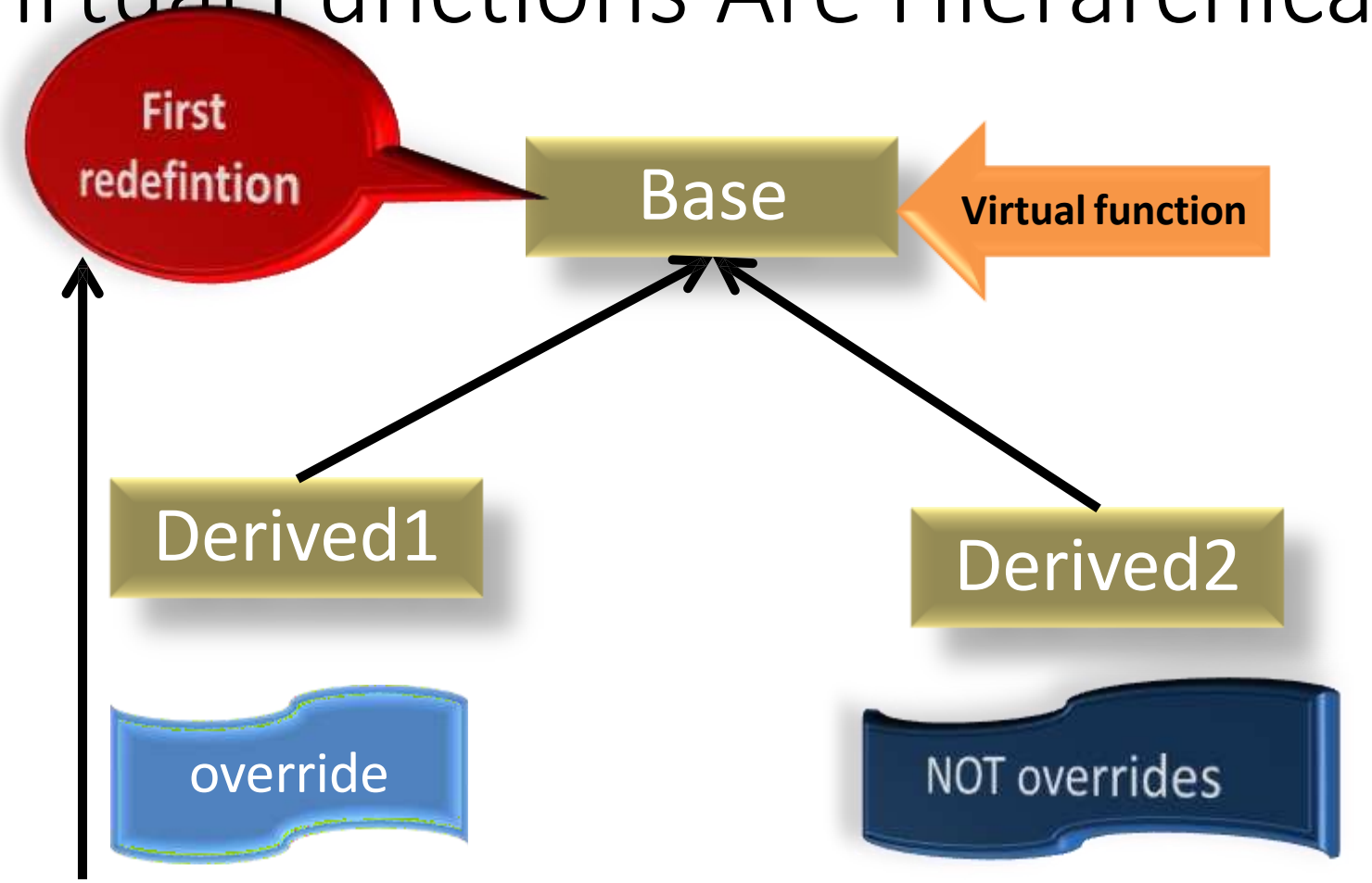
- Virtual functions are also **hierarchical in nature**.

- This means that when a **derived class fails** to override a virtual function, then **first redefinition** found in **reverse** order of derivation is used.

Virtual Functions Are Hierarchical



Virtual Functions Are Hierarchical



Pure Virtual Functions

- “A pure virtual function is a virtual function that has **no definition** within the base class.”

- To declare a pure virtual function:

  Syntax:

  virtual rtype func-name(parameter-list) = 0;

Pure Virtual Functions

- When a **virtual function** is made **pure**, any **derived** class must **provide** its **definition**.

- If the **derived** class **fails** to override the pure virtual function, a **compile-time error** will result.

NOTE:
When a virtual function is declared as pure, then all derived classes must override it.

Abstract Classes

- “A class that contains **at least** one **pure virtual** function then it is said to be abstract class.”

- **No objects** of an abstract class be **created**.

- Abstract class **constitutes** an **incomplete type** that is **used** as a **foundation** for **derived**. **classes**.

Using Virtual Functions

- We can achieve the most powerful and flexible ways to implement the **"one interface, multiple methods“**.

- We can create a class hierarchy that moves from **general to specific** (base to derived).

Using Virtual Functions

- We can define all **common** features and interfaces in a base class.
- **Specific** actions can be implemented only by the derived class.
- We can **add new** case easily.

Early vs. Late Binding

- “Early binding refers to events that occur at **compile time**.”
- Early binding occurs when all information needed to call a function is known at compile time.
- **Examples :**

  function calls ,overloaded function calls, and overloaded operators.

Early vs. Late Binding

- “Late binding refers to function calls that are **not resolved until run time**.”
- Late binding can make for somewhat **slower** execution times.
- **Example:**

  virtual functions