Section 3.2
Inheritance

1. Overview
2. Member access
3. Constructors and destructors
4. Types of inheritance
5. Multiple inheritance

3.2.1 Overview

- Role of inheritance in OO design
  - it’s another way to abstract and encapsulate data and behaviour
- Class can be more detailed specification of another class
  - **is-a** relationship
  - not to be confused with composition (has-a)
- Terminology in C++
  - base class
  - derived class

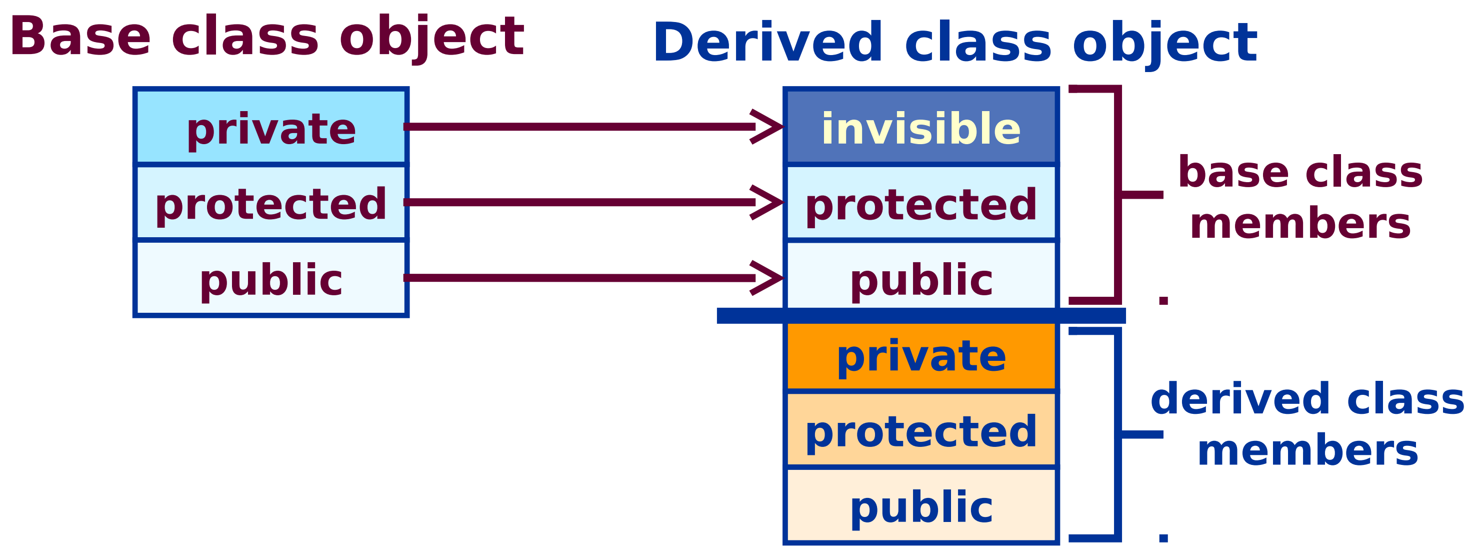
3.2.2 Member Access

- Accessing base class members from derived class
  - **all** base class members are inherited
    - they automatically become part of the derived class object
  - only public and protected base class members are accessible
  - private members are private *to the base class*
    - but they are still part of the derived class
  - **`coding example <p1>`**

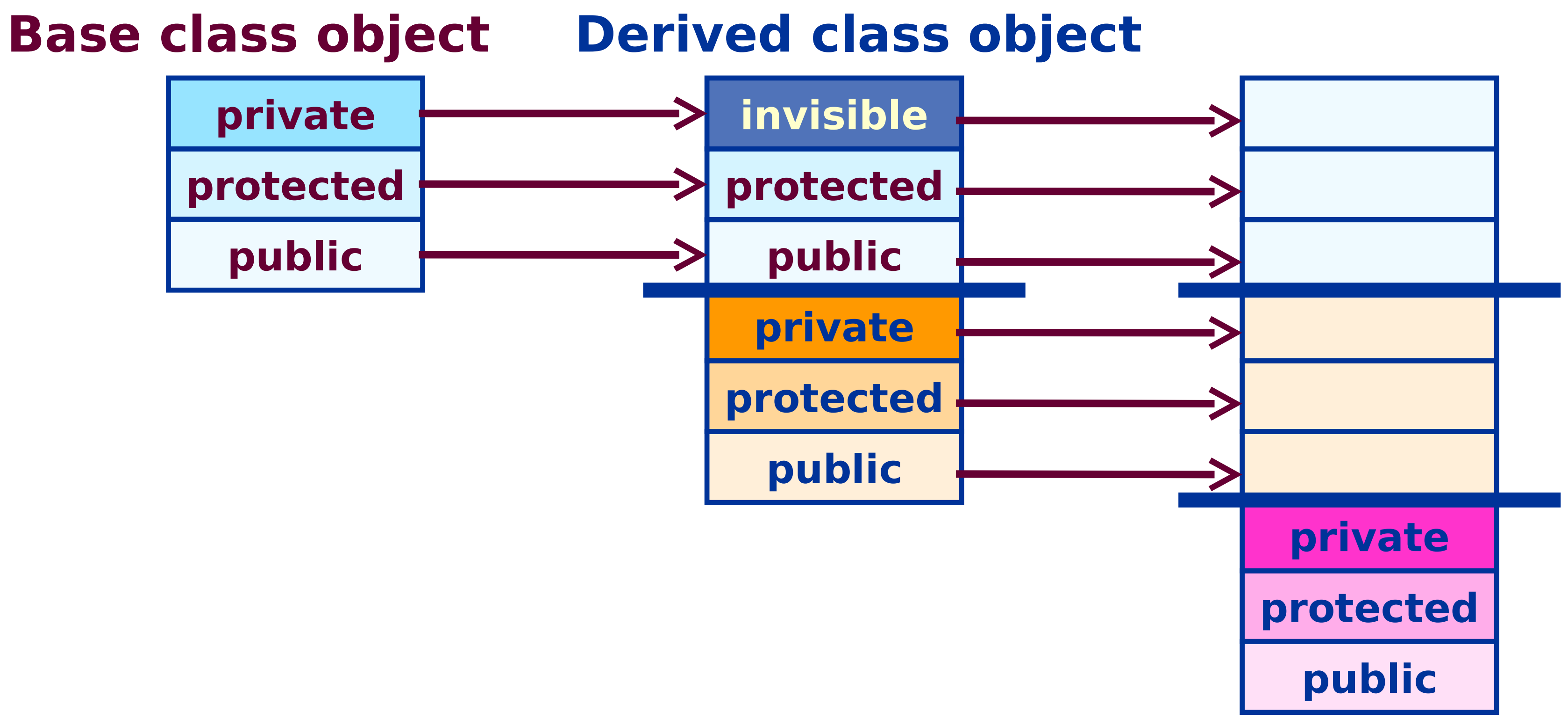
Member Access (cont.)

- Accessing private base class members:
  - **all** members of the base class are inherited
    - private members are inherited too
  - private members of base class are **not** visible to derived class
    - they can be accessed using:
      - the base class’s public or protected member functions
      - the base class’s friend classes and friend functions
    - *private* means private to the class where the member is defined

Member Access (cont.)



Member Access (cont.)



Member Usage

- Using inherited members
  - public and protected members
    - can be accessed directly by name
  - private members
    - can be accessed using base class public or protected member functions
    - can be accessed with base class friend classes and friend function

Member Usage (cont.)

- Overriding class members
  - use binary scope resolution operator to access base class member
  - overriding member redefines and **hides** the inherited member
- Friendship
  - base class’s friend functions and classes are **not** inherited
- **`coding example <p2>`**

3.2.3 Constructors and Destructors

- Initializing derived class objects
  - the base class constructor is always called, either:
    - *explicitly* by the derived class constructor, or
    - *implicitly* -- this executes the default base class constructor
  - use **base class initializer syntax** to prevent temporary objects
  - a constructor is responsible for initializing **its own members only**
    - **not** members from another class, including base class members
      - remember encapsulation
- Constructors and destructors are **not** inherited
  - but the base class constructor can still be *called*

Constructors and Destructors (cont.)

- Order of invocation for objects with inheritance:
  - constructors
    - objects are built top-down
    - base class part is constructed first
    - derived class part is last
  - destructors
    - objects are destroyed bottom-up
    - derived class part destroyed first, then base class part
    - destructors are invoked in reverse order of constructors

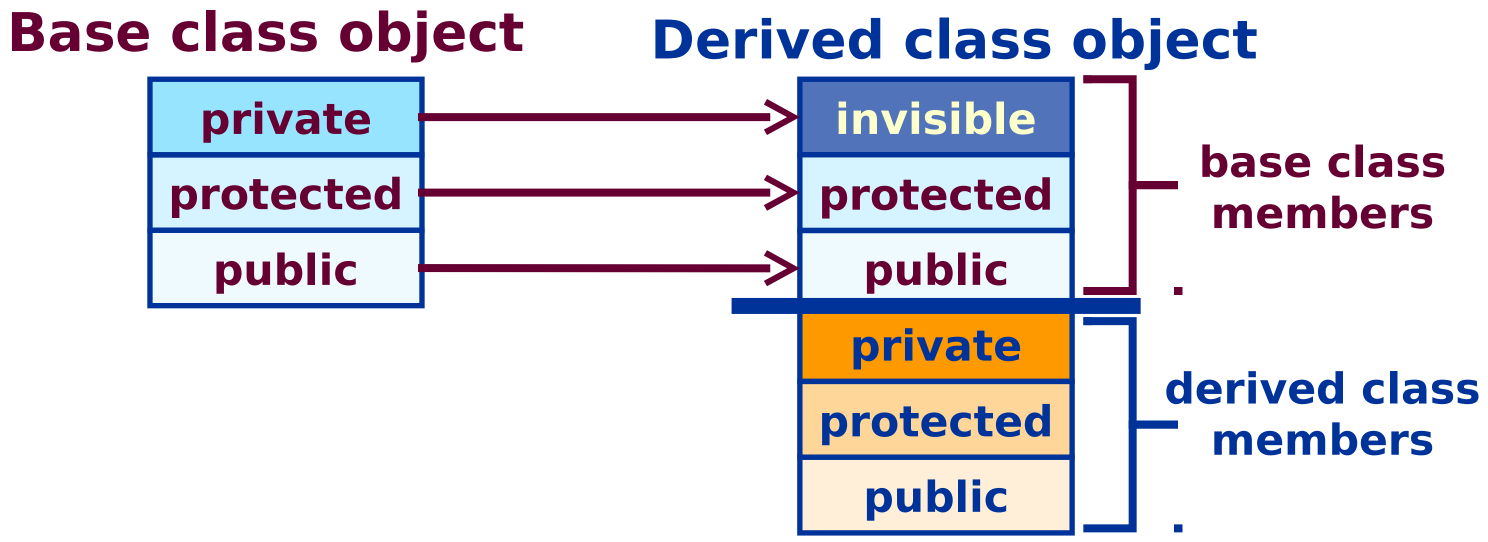
Constructors and Destructors (cont.)

- Order of invocation for objects with inheritance and composition:
  - constructors
    - objects are built top-down, inside-out
    - base class part is constructed first
      - containee objects within base class built before container class
    - derived class part is constructed last
      - containee objects within derived class built before container class
  - destructors
    - destructors are invoked in reverse order of constructors
  - **`coding example <p3>`**

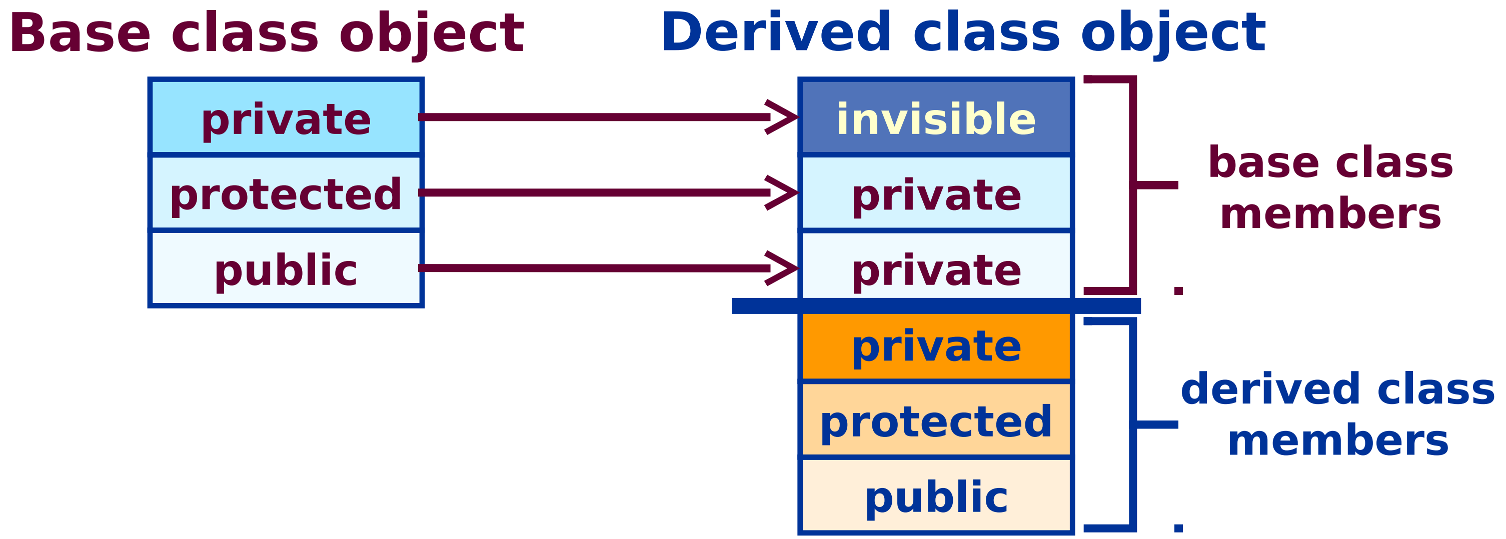
3.2.4 Types of Inheritance

- Public inheritance
  - this is an **is-a** relationship between base and derived classes
- Private or protected inheritance
  - this is **not** an is-a relationship
  - it is an advanced OO programming technique
  - it is an alternative to composition relationship and *delegation*
    - delegation: when one object passes on the responsibility for an operation to another object
  - it is used to *restrict access* to the inherited members

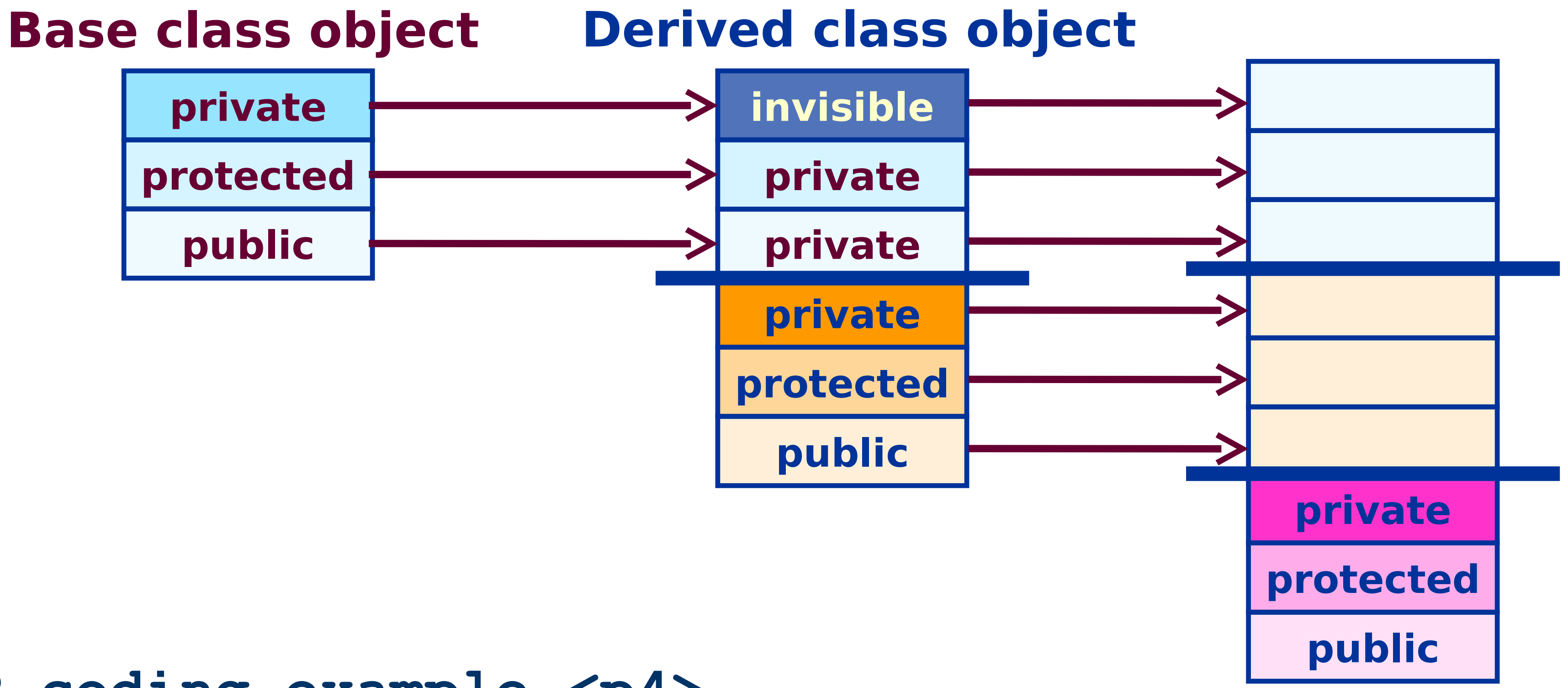
Public Inheritance



Private Inheritance

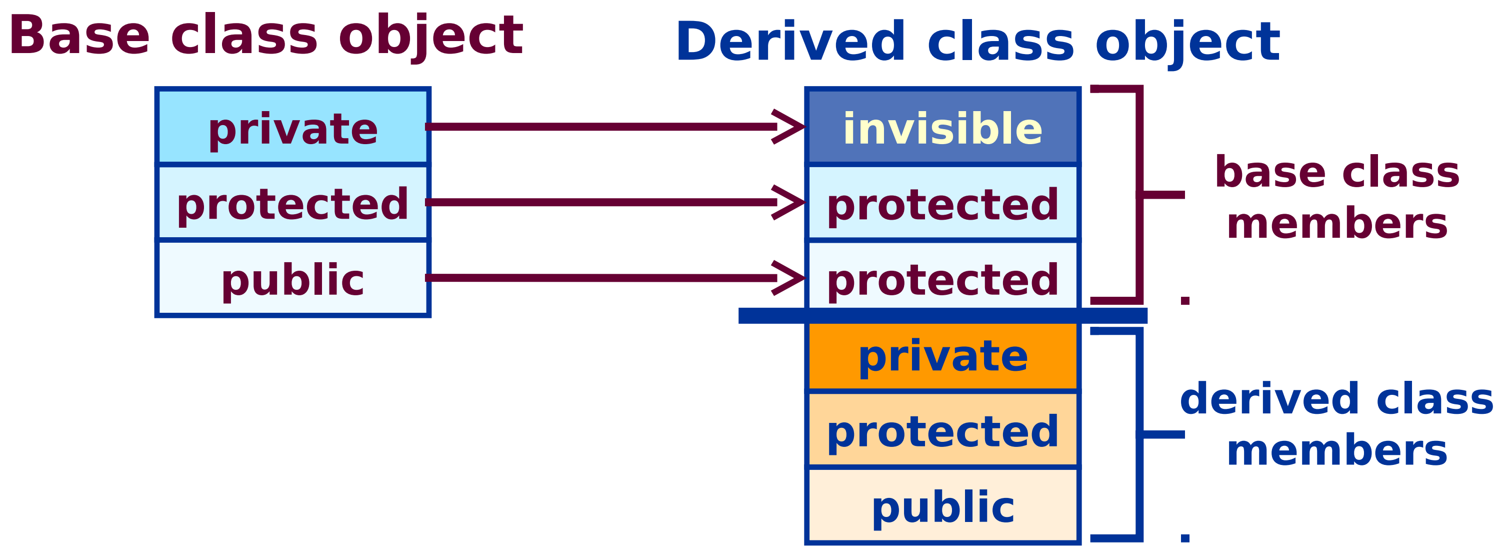


Private Inheritance (cont.)



- `coding example <p4>`

Protected Inheritance



3.2.5 Multiple Inheritance

- What is multiple inheritance?
  - when a class inherits from more than one base class
  - it is a technique not supported in many OO languages
  - ambiguity is resolved using the binary scope resolution operator
  - **`coding example <p5>`**

Multiple Inheritance (cont.)

- Types of multiple inheritance
  - distinct base class
  - multiple inclusion base class
  - virtual base class
- Problem: diamond hierarchy