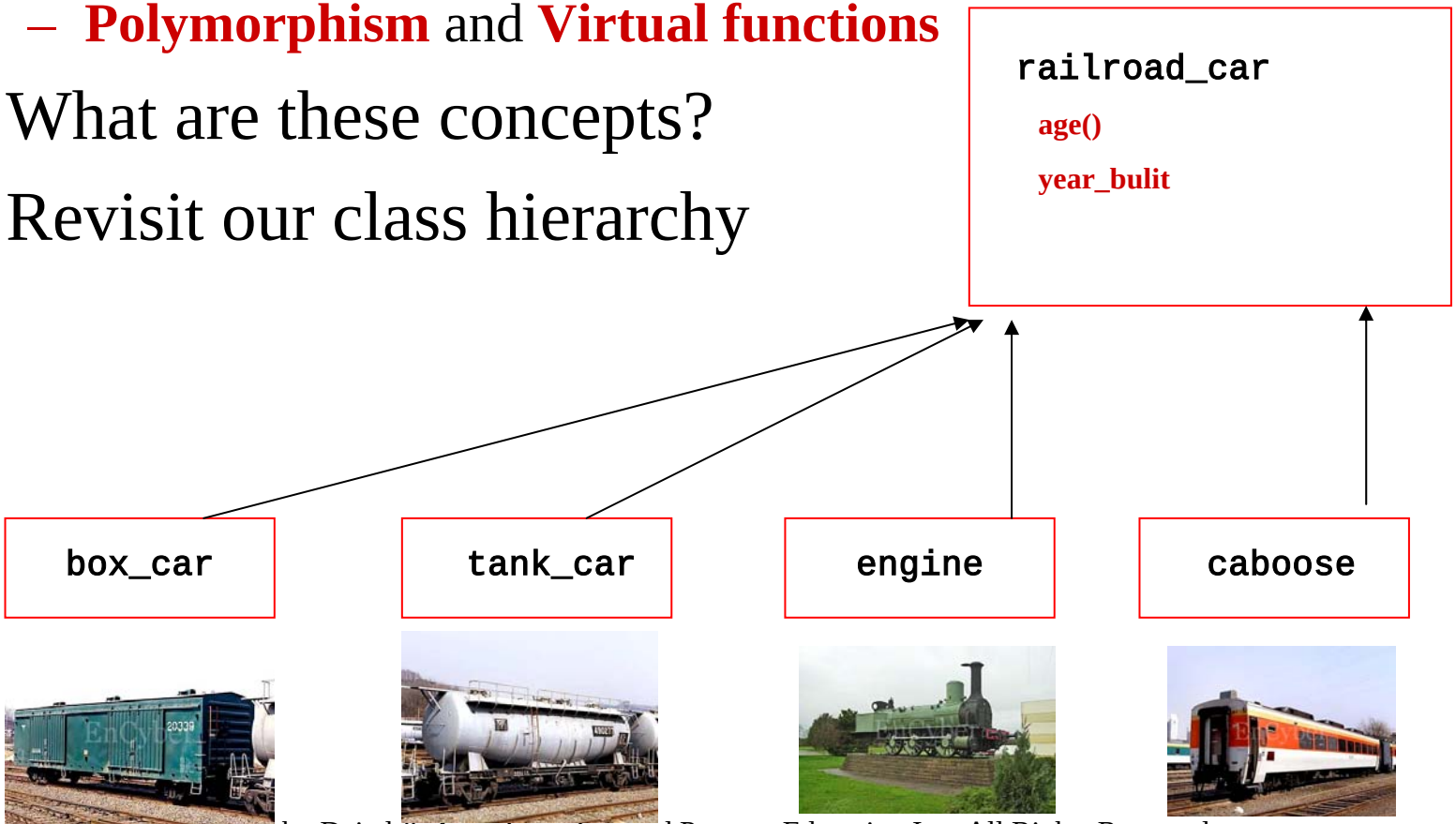
Introduction to Inheritance

Outline

- **How to find member functions at run time**
- **How to make virtual member functions call other member functions**
- **How to make constructors call other constructors in class hierarchy**

Benefit of Inheritance

- We know inheritance avoids needless duplication,
- But, that's all? No, its real value comes from
  - **Polymorphism** and **Virtual functions**
- What are these concepts?
- Revisit our class hierarchy



A Scenario: analyze_train Program

- Suppose there are many railroad cars that you want to save in a program. Where to save them?
  - Saving in some sort of an array would be a natural solution
  - Input: a stream of type code (in a file or from user input)
    - 0: engine, 1: box_car, 2: tank_car, 3: caboose
    - Input example: 0 1 2 1 3,….
  - Create a railroad car for each type code
    - e.g., `new engine` for 0, `new box_car` for 1, etc.
  - Save it on an array
    - `train[i] = new engine;`
    - `train[i] = new box_car;`
    - …
  - What would be the type of the array `train[]` ?

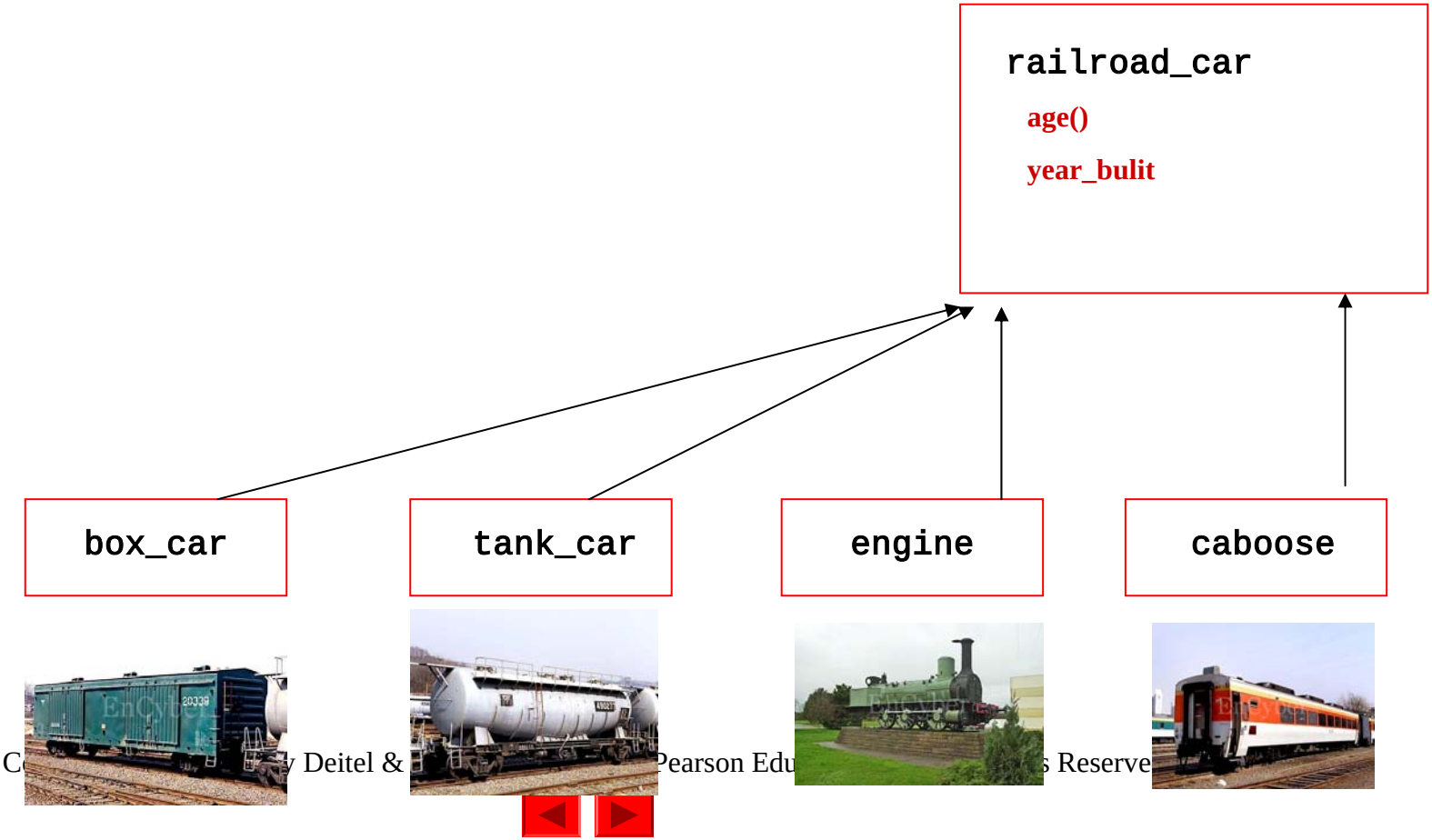
Array of Pointers to Objects

- C++ requires objects in an array to be of the same type
  - Why? Each object should occupy the same amount of memory
  - Even if it is an array of pointers, they should be of same pointer type
- However, if you define an array of pointers to a certain class, the actual pointers can point to
  - Not only any object of that class, but also
  - Any object in its **subclass**
  - e.g., the following is allowed

```
railroad_car *train[100];
train[0] = new engine;
train[1] = new caboose;
train[2] = new tank_car;
```


Polymorphism

- A class pointer variable can point an object of its subclass
  - e.g., railroad_car class pointer variable can point a tank_car object
  - As in a variable, an array, a formal argument, ..



```
class railroad_car {
 public: railroad_car(){}
};

class box_car : public railroad_car {
 public: box_car(){}
};
class tank_car : public railroad_car {
 public: tank_car(){}
};
class engine : public railroad_car {
  public: engine(){ }
};
class caboose : public railroad_car {
  public: caboose(){ }
};
```


```
// Define railroad car array:
railroad_car *train[100];
main () {
  // Declare various integer variables:
  int car_count, type_code;
  // Read type number and create corresponding objects:
  for (car_count = 0; cin >> type_code; ++car_count)
    if (type_code == 0)        train[car_count] = new engine;
    else if (type_code == 1)   train[car_count] = new box_car;
    else if (type_code == 2)   train[car_count] = new tank_car;
    else if (type_code == 3)   train[car_count] = new caboose;
  // Display car count:
  cout << "There are " << car_count << " cars in the array."
   << endl;
}
```


An Improved `analyze_train` Program

```
railroad_car *train[100];
//Declare enumeration constants, needed in switch statement:
enum {eng_code, box_code, tnk_code, cab_code};
main () {
  // Declare various integer variables:
  int n, car_count, type_code;
  // Read car-type number and create car class objects:
  for (car_count = 0; cin >> type_code; ++car_count)
    switch (type_code) {
      case eng_code: train[car_count] = new engine;   break;
      case box_code: train[car_count] = new box_car;  break;
      case tnk_code: train[car_count] = new tank_car; break;
      case cab_code: train[car_count] = new caboose;  break;
      default: cerr << "Car code " << type_code
                    << " is unknown!" << endl;
               exit (0);
    }
}
```


Adding a Member Function to Subclasses

- Now we want to define an member function in each subclass of railroad_car that displays its car name
  - For example,

```
class box_car : public railroad_car {
  public: box_car () { }
          void display_short_name () {cout << "box";}
};
class tank_car : public railroad_car {
  public: tank_car () { }
          void display_short_name () {cout << "tnk";}
};

...
```


Walk Thru the Array and Print

- Then, we walk thru the array and print name of each object

```
for (n = 0; n < car_count; ++n) {
   train[n]->display_short_name();
   cout << endl;
}
```

- This is a very elegant way of handling the print job because
  - Otherwise, we need a member variable identifying each object, and
  - We would need to check the type of each object, something like

```
for (n = 0; n < car_count; n++)
    switch (train[n]->type_code) {
      case eng_code: cout << “eng” << endl;  break;
      case box_code: cout << “box” << endl;   break;
      case tnk_code: cout << “tnk” << endl; break;
      case cab_code: cout << “cab” << endl;   break;
    }
```


Compiler Rejects It, Though

- Unfortunately, C++ compiler cannot compile this code

```
for (n = 0; n < car_count; ++n) {
  train[n]->display_short_name();
  cout << endl;
}
```

  - Why not? No definition of `display_short_name()` in `railroad_car` class
  - After all, train[] is a pointer array to `railroad_car` class objects

- So we want to add `display_short_name()` to `railroad_car` class

```
class railroad_car {
  public: railroad_car () { }
          void display_short_name () {cout << "rrc";}
};
```

- Still, it does not work
  - The for loop will repetitively print `rrc` only

Virtual Functions

- We want appropriate function is chosen at run time, not decided by compiler statically, while being compiled OK
- We can convey the idea to C++ with a keyword "`virtual`"

```
class railroad_car {
  public: railroad_car () { }
          virtual void display_short_name () {cout << "rrc";}
};
```

- Add `virtual` to functions in the subclasses as well

```
class box_car : public railroad_car {
  public: box_car () { }
          virtual void display_short_name () {cout << "box";}
};
```

- Why virtual? Because which function to use is not available at compile-time

```
class railroad_car {
  public: railroad_car () { }
          virtual void display_short_name () {cout << "rrc";}
};
class box_car : public railroad_car {
  public: box_car () { }
          virtual void display_short_name () {cout << "box";}
};
class tank_car : public railroad_car {
  public: tank_car () { }
          virtual void display_short_name () {cout << "tnk";}
};
class engine : public railroad_car {
  public: engine () { }
          virtual void display_short_name () {cout << "eng";}
};
class caboose : public railroad_car {
  public: caboose () { }
          virtual void display_short_name () {cout << "cab";}
};
```


When We Use Virtual Functions?

- We have a pointer defined to point some class A
- You assign the pointer to an object, introduced at runtime, which belongs to a subclass of the class A
- You want C++ to pick a member function foo(), on the basis of the object's class

Then you must define a version of foo() in A and mark it with `virtual`

  - foo() will automatically be virtual in all subclasses
  - However, it would be clearer to mark them all `virtual` explicitly
  - foo() in subclasses will shadow foo() in A

Pure Virtual Function

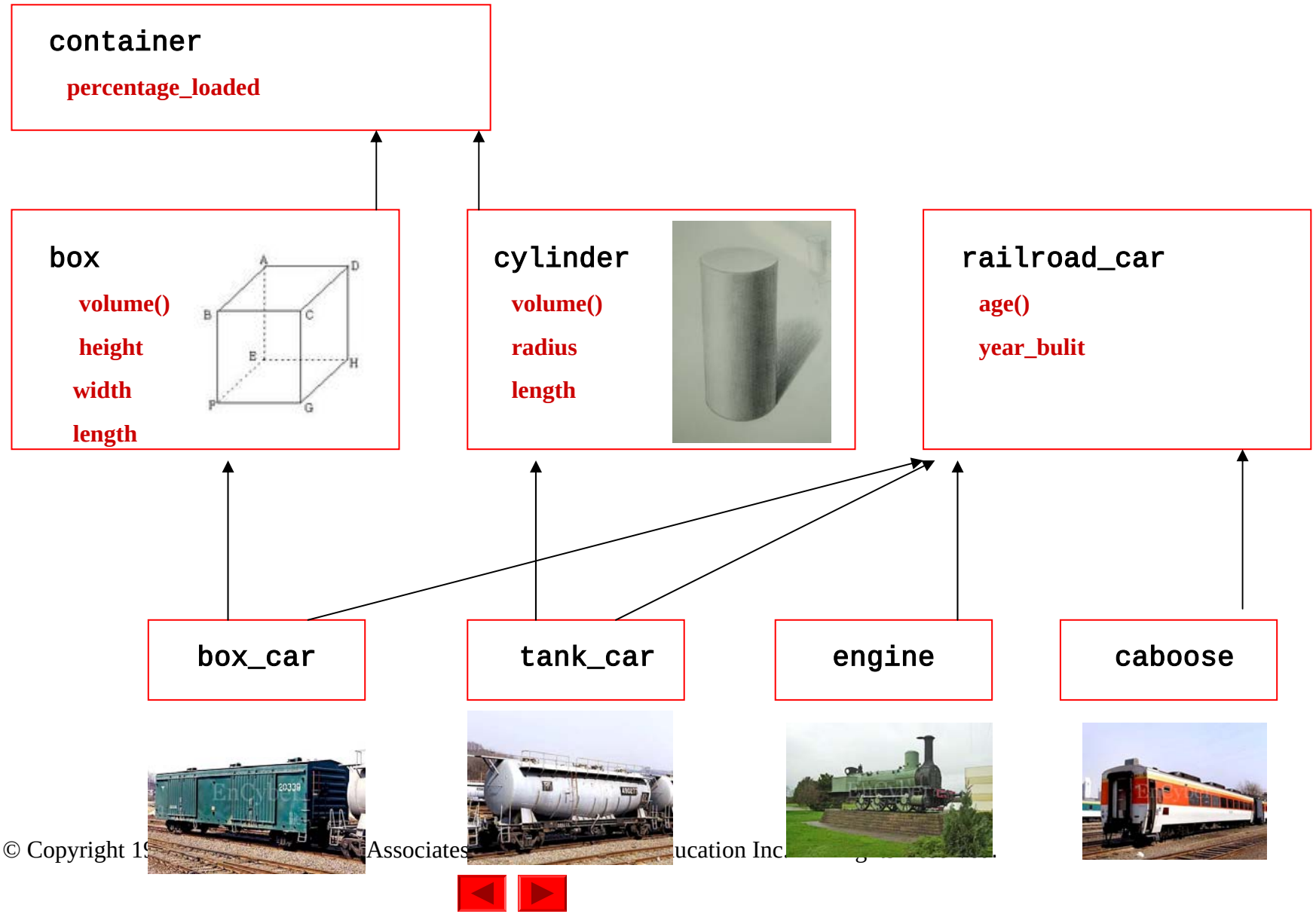
- If display_short_name() in railroad_car class is only for correct compilation (i.e., shadowed in every situation), you can make a pure virtual function

```
class railroad_car{
  public: railroad_car () { }
          virtual void display_short_name () = 0;

};
```

- Calling a pure virtual function causes an error
- If not completely shadowed, use a do-nothing function

```
class railroad_car{
  public: railroad_car () { }
          virtual void display_short_name () { };

};
```


Revisit Our Full Class Hierarchy



Constructors Call Other Constructors

- Previously box had only a default constructor

```
class box : public container {
  public: double height, width, length;
          box () { }
          double volume () {return height * width * length;}
};
class box_car : public railroad_car, public box {
  public: box_car () {
            height = 10.5; width = 9.2; length = 40.0;}
};
```

- Now we want to add argument-bearing constructor for box

```
box (double h, double w, double l) {
          height = h; width = w; length = l;
}
```


Constructors Call Other Constructors

- How to make `box_car()` call `box(parameters)` explicitly?

```
class box : public container {
  public: double height, width, length;
          box () { }
          box (double h, double w, double l) {
               height = h; width = w; length = l;
          }
          double volume () {return height * width * length;}
};
class box_car : public railroad_car, public box {
  public: box_car (): box(10.5, 9.5, 40.0) { };
```


Virtual Member Function Calls Other Function

- We also want to print capacity of each train

```
for (n = 0; n < car_count; ++n) {
  train[n] -> display_short_name ();
  cout << "     ";
  train[n] -> display_capacity ();
  cout << endl;
}
```

- Define `display_capacity()` as a virtual function in `box_car`

```
virtual void display_capacity () {cout << height * width * length;}
```

  - By inheriting `height, width, length` from the `box` class
  - Similarly we define `display_capacity()` for the `tank_car` class

- However, it would be better to use volume() in `box`

```
virtual void display_capacity () {cout << volume ();}
```
, or
```
virtual void display_capacity () {cout << this -> volume ();}
```


Updated Class Definitions

```
class railroad_car {
  public: railroad_car () { }
          virtual void display_short_name () { }
          virtual void display_capacity () { }
};
class box_car : public railroad_car, public box {
  public: // Default constructor:
          box_car () : box (10.5, 9.2, 40.0) { }
          // Displayers:
          virtual void display_short_name () {cout << "box";}
          virtual void display_capacity () {cout << volume ();}
};
class tank_car : public railroad_car, public cylinder {
  public: // Default constructor:
          tank_car () : cylinder (3.5, 40.0) { }
          // Displayers:
          virtual void display_short_name () {cout << "tnk";}
          virtual void display_capacity () {cout << volume ();}
};
```