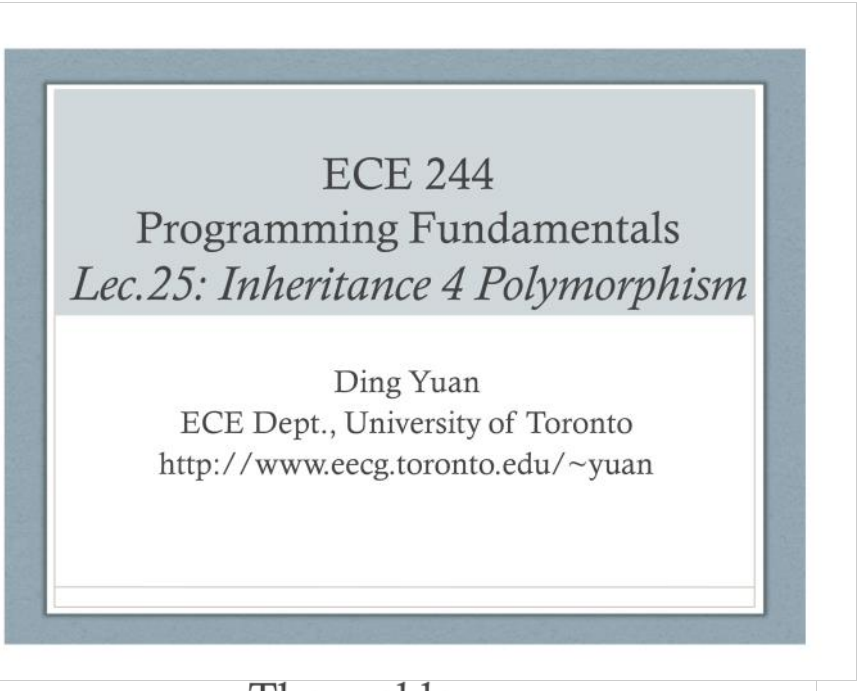
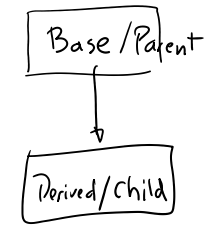
# Inheritance 4

Monday, November 14, 2022 12:55 PM



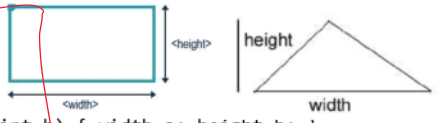




class child : class Parent {

3;

class

## The problem




```
4 class Polygon {
5 protected:
6   int width, height;
7 public:
8   void set_values (int a, int b) { width=a; height=b; }
9 };
10
11 class Rectangle: public Polygon{
12 public:
13   int area() {
       return width*height;
     }
14 };
15
16 class Triangle: public Polygon{
17 public:
18   int area() {
       return width*height/2;
     }
19 };
```

```
21 int main () {
22   Rectangle rect;
23   Triangle trgl;
24   Polygon * ppoly1 = &rect;
25   Polygon * ppoly2 = &trgl;
26   ppoly1->set_values (4,5);
27   ppoly2->set_values (4,5);
28   cout << rect.area() <<'\n';
29   cout << trgl.area() <<'\n';
30   return 0;
31 }
```

But we can't do this: ppoly1->area()

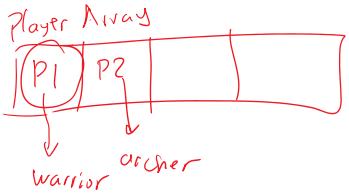
compiler error

# The problem (cont.)

- What if you want to be able to invoke "area()" using a pointer to an object of base class

```
22   Rectangle rect;
23   Triangle trgl;
24   Polygon * ppoly1 = &rect;
25   Polygon * ppoly2 = &trgl;
```

  - E.g., ppoly1->area() will invoke rect.area(), and ppoly2->area() will invoke trgl.area()
- More generally, we want use a pointer to the base class object to invoke a function that is implemented in a derived class
  - You cannot do it using what we have learnt so far



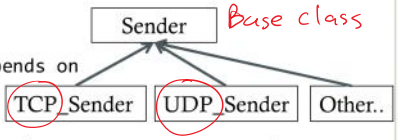

compiler error

A[0]-> attack();

# Some real world examples

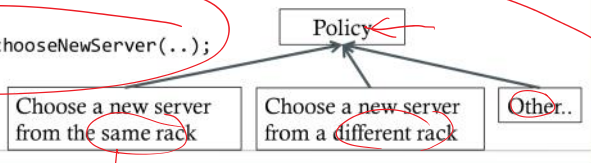
- A network system can use different protocols

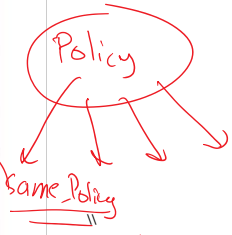
```
void connect (Sender *sender) {
  sender->send (..);
  // Which send() is invoked depends on
  // the actual type of sender
}
```

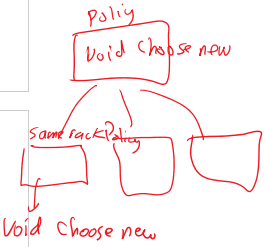



- Server fails frequently in data centers; you may have different policies to handle the failure

```
nv = policy->chooseNewServer(..);
```




# Data center

declaration

# Solution: virtual functions

If a function is declared as virtual in base class A, and redefined in derived class B, then a call made via a pointer, which is declared as A* type but actually points to an obj. of type B, will cause the function defined in B being invoked

```
 4 class Polygon {
 5 protected:
 6   int width, height;
 7 public:
 8   void set_values (int a, int b) { width=a; height=b; }
 9   virtual int area() { return 0; }
10 };
11
12 class Rectangle: public Polygon {
13 public:
14   int area() { return width*height; }
15 };
16
17 class Triangle: public Polygon {
18 public:
19   int area() { return width*height/2; }
20 };
```

```
Rectangle rect;
Triangle trgl;
Polygon * ppoly1 = &rect;
Polygon * ppoly2 = &trgl;
ppoly1->set_values(4,5);
ppoly2->set_values(4,5);
cout<<ppoly1->area(); // 20
cout<<ppoly2->area(); // 10
```

*What if we remove the "virtual" keyword from line 9?*

Polygon poly;
Polygon* pp3 = &poly;
pp3 → area(); return 0;

ppoly → Polygon::area();
Parent function

function foo(Polygon&) {

# Polymorphism

- A class that declares or inherits a virtual function is called a polymorphic class
  - Both Rectangle and Triangle are polymorphic classes of Polygon
  - Both TCP_Sender and UDP_Sender are polymorphic classes of Sender

# Virtual functions: more details

- The decision as to which inherited function to call is made at runtime when the function is invoked (rather than at compile time) depending on the type of the target object
  - This feature is also called late binding
- The virtual function in Base class is like a "dummy" or "placeholder" declaration of the function
- You cannot declare a member variable as virtual
  - Only functions

jump foo

runtime

virtual → operator =
↳ destructor

# Pure virtual functions

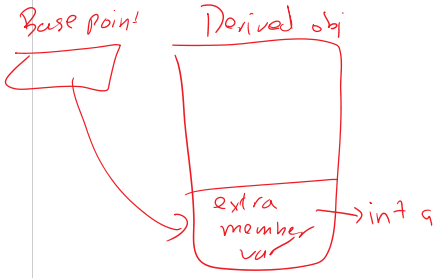
abstract class

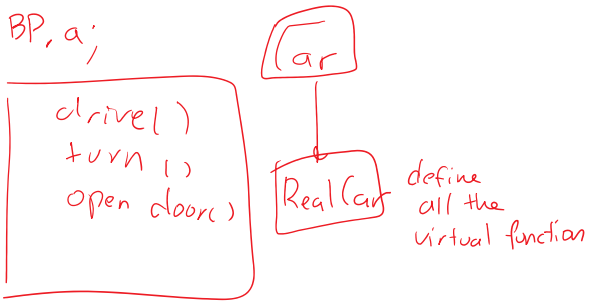
- If we never intended to use "Polygon" to create objects (i.e., we use the base only to derive classes), then we do not fill dummy function definition

```
4 class Polygon {
5 protected:
6   int width, height;
7 public:
8   void set_values (int a, int b) { width=a; height=b; }
9   virtual int area() = 0; // Pure virtual function
10 };
```

Polygon p;
compile time

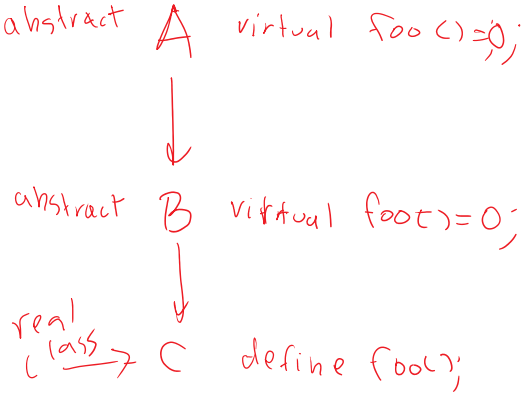
- A class that contains at least one pure virtual function is called an abstract class
  - Also called an interface
- You cannot instantiate an object of an abstract class



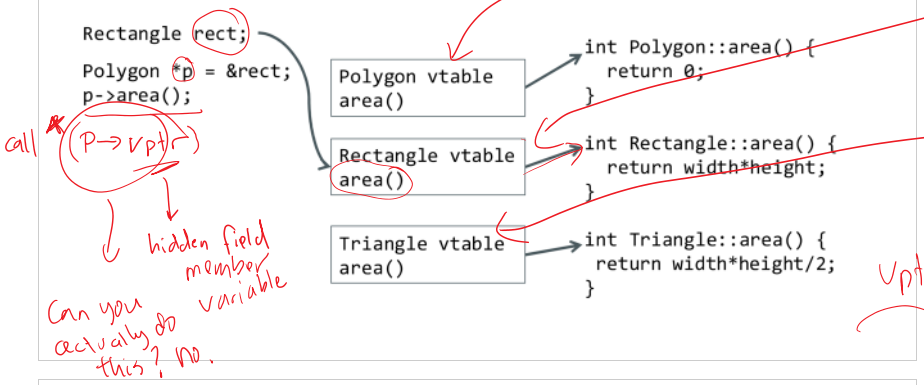

# But there is no free lunch

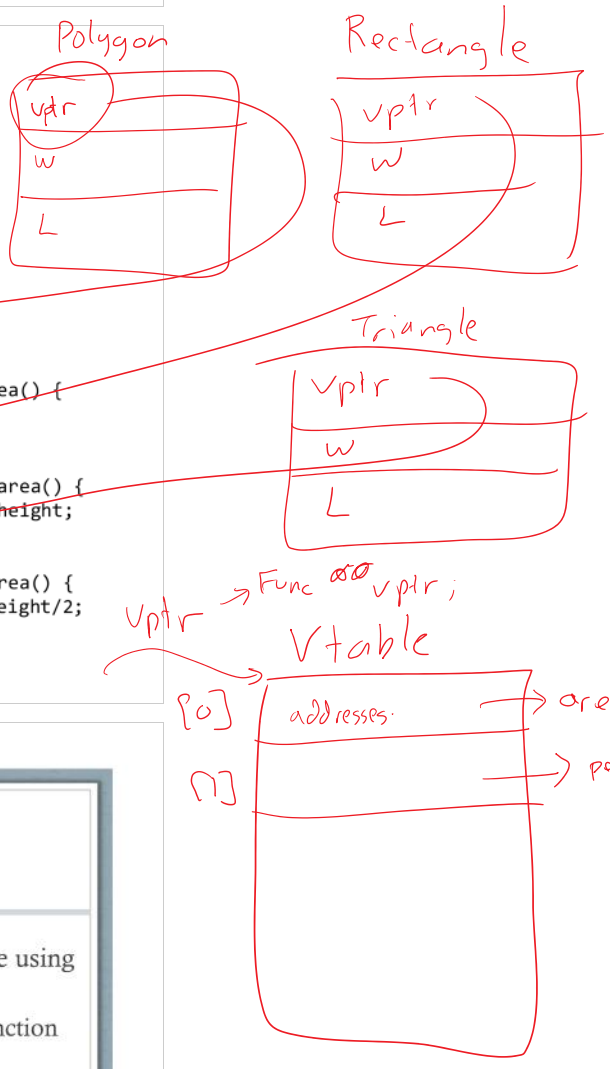
- At runtime, a call to a virtual function is no longer a simple jump
- Needs to figure out the call target
  - Performance penalty: more expensive than a normal call
- Implementation: vtable (virtual method table)



# vtable

- For every class, there is a table that maps each virtual function to the most-derived function target
- Each object has a pointer to the vtable of its type




# Overhead

- On C++, this overhead is 6-13% compared w/software using only non-virtual function calls
  - Also prevent compilers from other optimizations, e.g., function inlining
- Modern systems use frameworks, tend to make things worse
- Just-In-Time compiler can use inline caching to reduce this overhead