Virtual function → pre-requisite → inheritan

Inheritance → is-a relationship

Product

↓

Electronics

What additional functionality virtual keyword adds??

↳ Dynamic Dispatch and Static dispatch.

```
class A {
  public:
      void foo();   // declaration
}
void A::foo() {
     cout << "Hello foo\n";
}
```

Compiler knows during → compile time what function to call.

Static Dispatch / Early binding

whenever compiler finds a call to foo() on an instance of A, it will print "Hello foo" → routine

```
class B {
  public:
    virtual void bar() { cout << "bar from B \n"; }

    virtual void car() { cout << "car from B \n"; }
};

class C : public B {
  public:
    void bar() override {
        cout << "bar from C \n";
    }
};
```

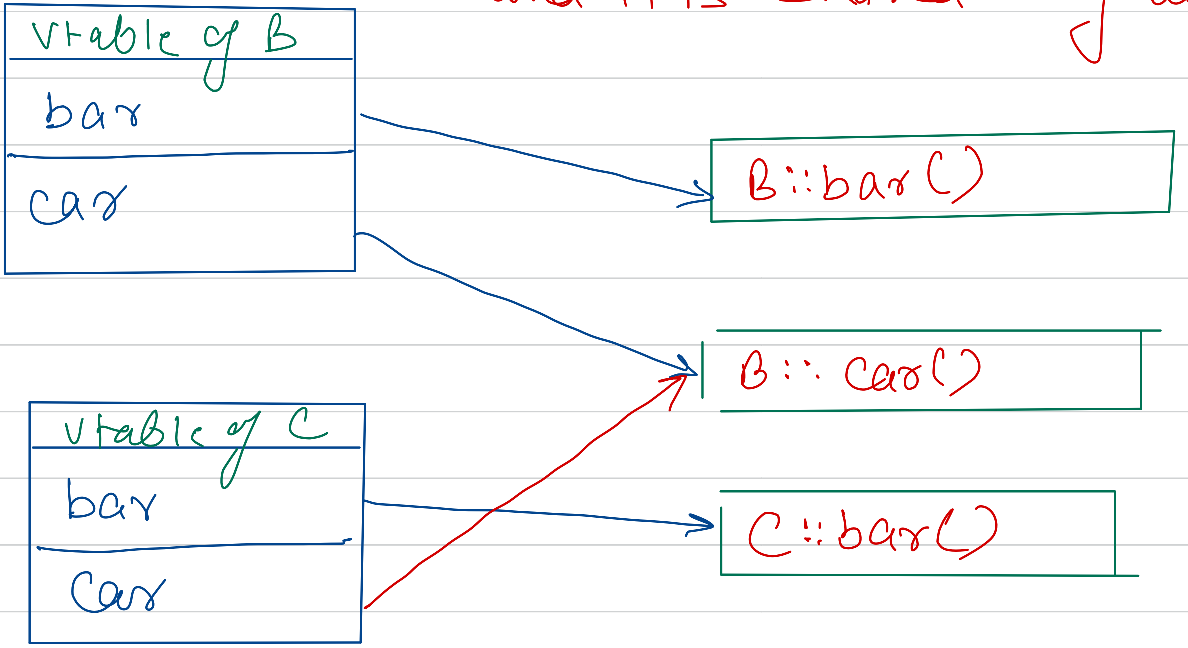
Dynamic Dispatch / Late Binding

```
B* b = new B();    b → bar(); → bar from B
B* c = new C();    c → bar(); → bar from C
```

Because virtual function shows dynamic dispatch using vtable(virtual table)

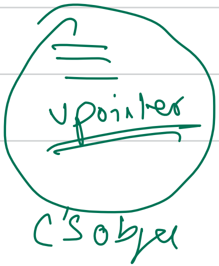
The vtable contains an entry for each virtual fun^n accessible by the class & stores a pointer to the actual definition of func^n.

Every class maintains a vtable and it's shared by all instance



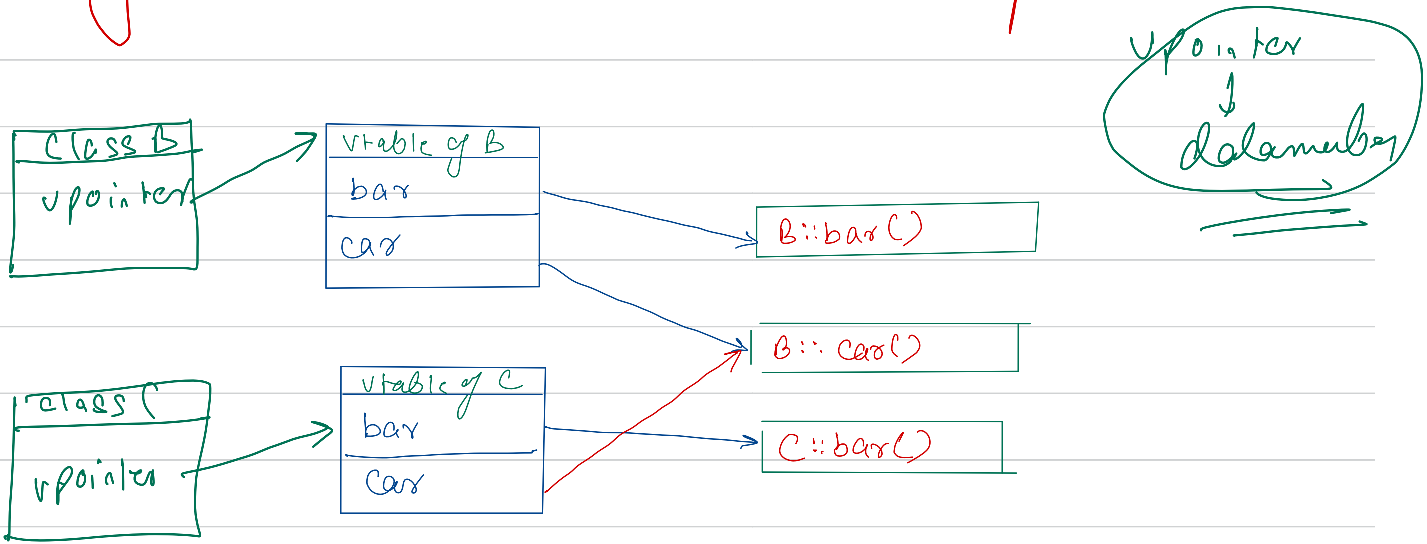
b->bar()

C->bar()



virtual pointer
(vpointer)

Every time compiler creates a vtable for a class it adds an extra argument to it, a pointer corresponding to the vtable called vpointer



Product → display()

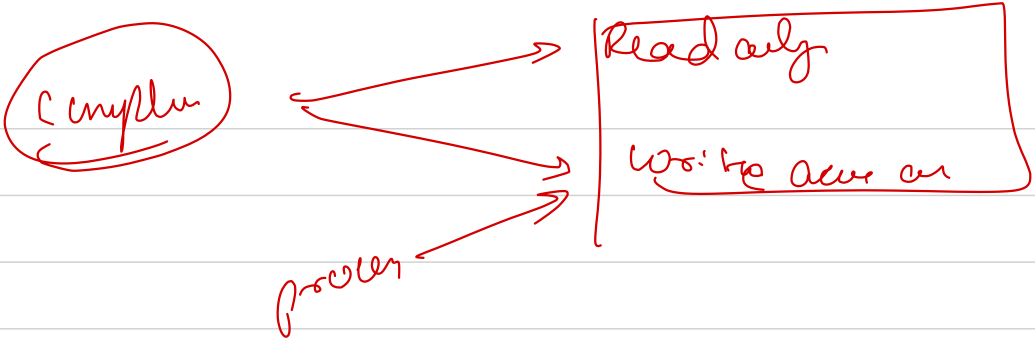
↓

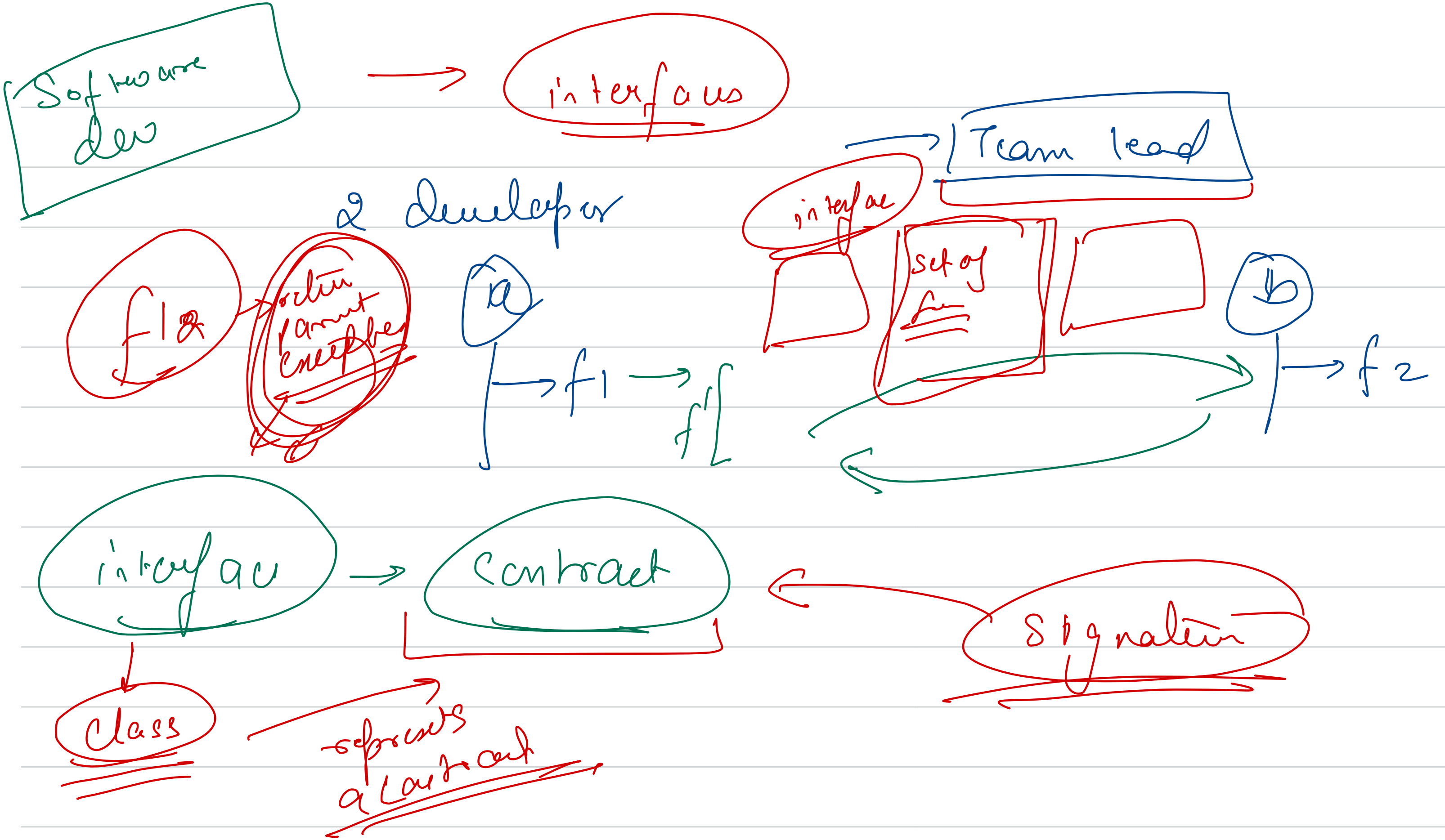
Electronic → display()

↓

mobil → display()

```
void fun(Product *p) {
    p → display();
}
```


Complex
Read only
Write access on
process

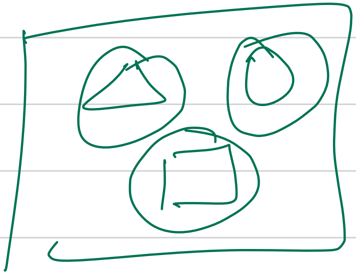

Software dev
interfaces
2 developer
Team lead
interface
set of fn
f1 &
solution
format
[illegible]
a
f1
f1
b
f2
interface
contract
signature
class
represents a contract

interface / abstract class → forces the class to implement the functions acting as a contract.

In order to make an interface we produce pure virtual function

Coming in 2 mins

Software → print shapes   H/w



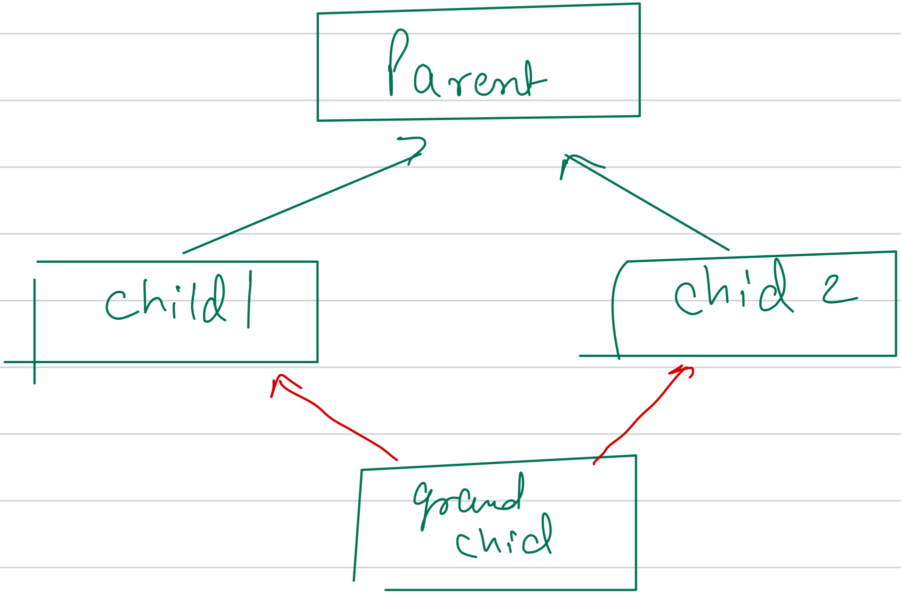
print shapes
- → square → display ()
- → tri → display ()
- → circle → display ()

kuttrapram (Shape* s)
s → display
}

Shape
↳ pure virtual → display () = 0;

square   trice   circle
↳   ↦   ↦

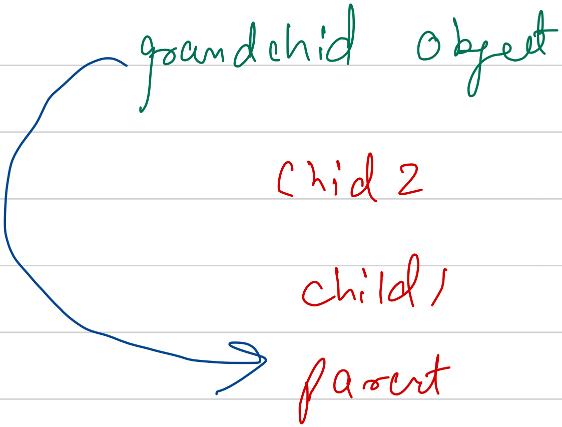
Virtual Base Class



This diamond problem is solved by virtual <u>inheritance</u>

① There will be new virtual base pointers which will be inserted by compiler when we use virtual inheritance.

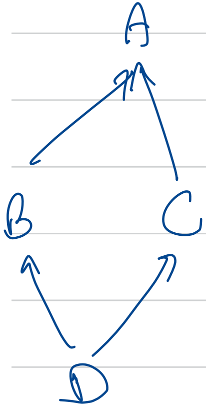
② This virtual base pointer will provide indirect access to class A's object.

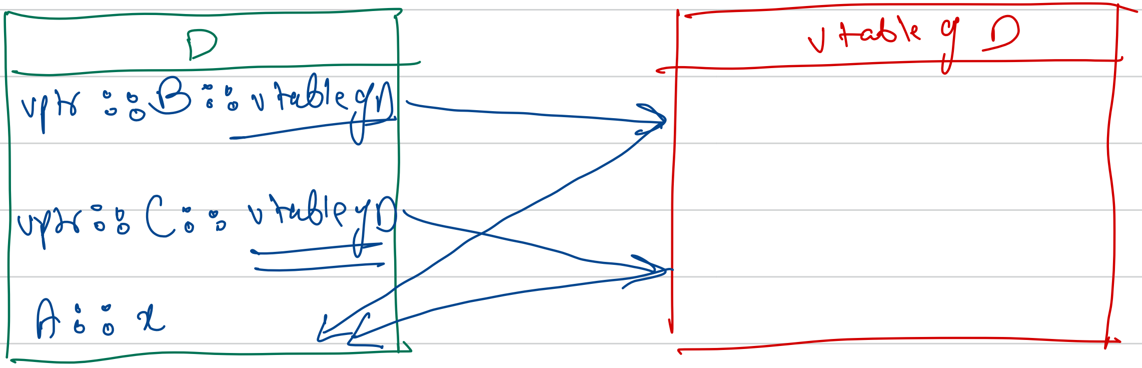


doing virtual inheritance affects size ??

- Yes ✓
- No

v pointer





```
class A {
  public:
    int x;

}
```

vptrs index a virtual table & there is a vptr per every virtual base class

Runtime Polymorphism