

Dynamic Dispatch

Dynamic dispatch is the process by which different implementation of methods are used in runtime.

This is done in c++ by using the virtual keyword. To show how this works I made a base class named feline and two subclasses named cat and lion. Each of these classes have three functions: sound, purr, and eat. In the feline base class, sound and eat are both declared with the virtual keyword. In the cat and lion subclasses I have different implementations for each. Last, in the main method I create an object and have the object call the sound purr and eat methods. Since the sound and eat methods are both virtual and use dynamic dispatch, they call the version of sound and eat defined in the subclass. This results in the output of “meow, purrpurr, and nibble” for the cat class function calls. The way that these methods are called in assembly is interesting. For the statically dispatched methods, assembly first moves the parameters into the right registers and then calls the function of the feline base class. The way that assembly deals with dynamic dispatch is different however. Instead of directly calling the function, multiple dereferences are made to access the correct function. The first operation is moving a dereferenced base pointer -8 to rax, then dereferencing rax twice, then moving a dereferenced base pointer -8 to rdx then to rdi then calling rax. This works because a pointer to this function is stored at this location.

```
class feline
{
public:
    virtual void sound(){
        cout<<"hello"<<endl;
    }
    void purr(){
        cout<<"purrpurr"<<endl;
    }
    virtual void eat(){
        cout<<"nomnom"<<endl;
    }
};
```

```
class cat:public feline
{
public:
    void sound(){
        cout<<"meow"<<endl;
    }
    void purr(){
        cout<<"purrpurrpurr"<<endl;
    }
    void eat(){
        cout<<"nibble"<<endl;
    }
};
```

```
mov rax, QWORD PTR [rbp-8]
mov rdi, rax
call feline::purr()
```

```
mov rax, QWORD PTR [rbp-8]
mov rax, QWORD PTR [rax]
mov rax, QWORD PTR [rax]
mov rdx, QWORD PTR [rbp-8]
mov rdi, rdx
call rax
```

The dereferences of `rax` allow the assembly code to call `rax` and it is actually calling the function `sound`.

In the assembly code, because the `purr` function of `feline` is not declared as virtual, there is only one implementation of `purr`. Even though `cat` and `lion` have implementations of `purr`, since they cannot be called they are not even included in the assembly code. Since the `sound` and `eat` functions are virtual, there are two different implementations of each function, one for `cat` and one for `lion`, but not one for `feline`.

When the code is created each type of class has a vtable created for virtual functions. The vtable for each class shows which functions are virtual which helps decide whether to use static or dynamic dispatch.

```
vtable for lion:
  .quad 0
  .quad typeinfo for lion
  .quad lion::sound()
  .quad lion::eat()
```