CSCI 334:
Principles of Programming Languages

Lecture 19-1: OOP III

Instructor: Dan Barowy

Williams

Topics

Dynamic Dispatch Refresher

C++: Only Pay for What You Use

Virtual dispatch

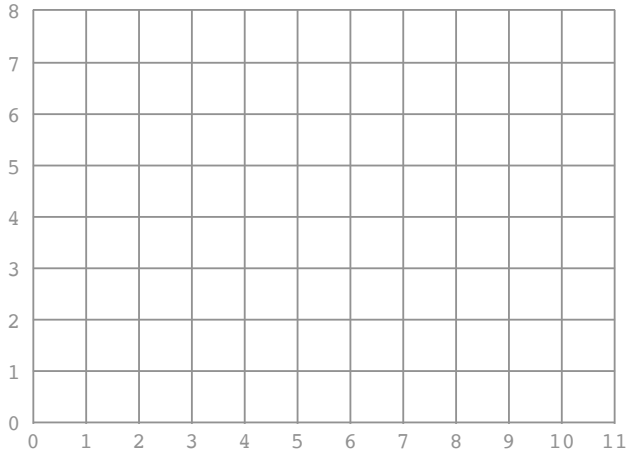
Dynamic Dispatch

How OO polymorphism works

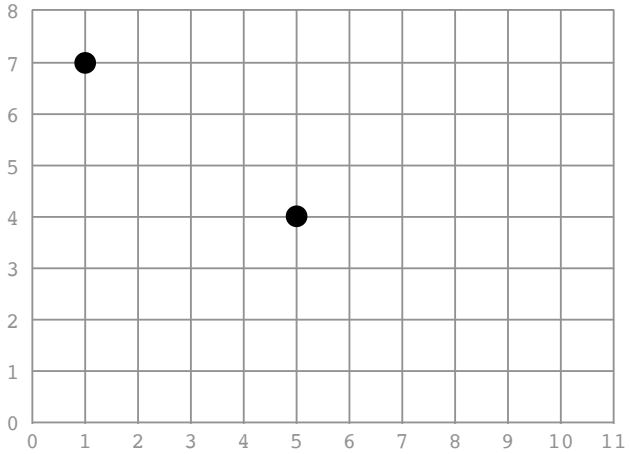
Ingalls Test for Extensibility

- The test is about the ability to extend software *after* it has already been designed and written.
- E.g., suppose you have a class for a `ColoredRectangle`.
- Can you define a new kind of number (e.g., fractions), use your new numbers to redefine (subtype) rectangle, and then ask the system to color the rectangle?
- If so, you have an OO system.

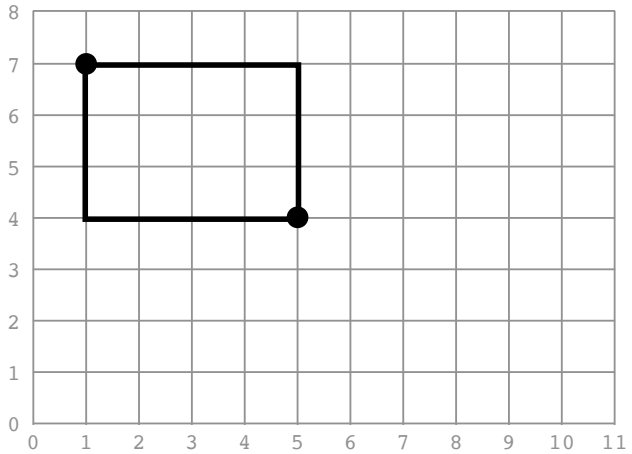
Ingalls Test for Extensibility



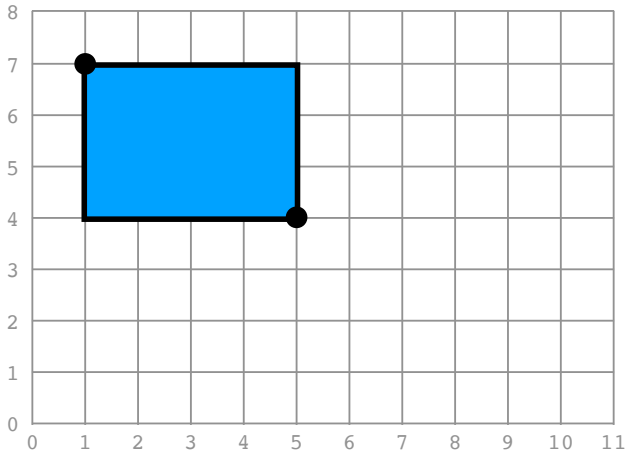
Ingalls Test for Extensibility



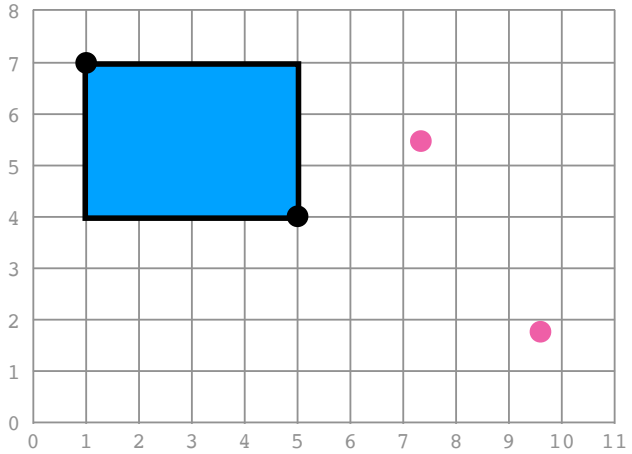
Ingalls Test for Extensibility



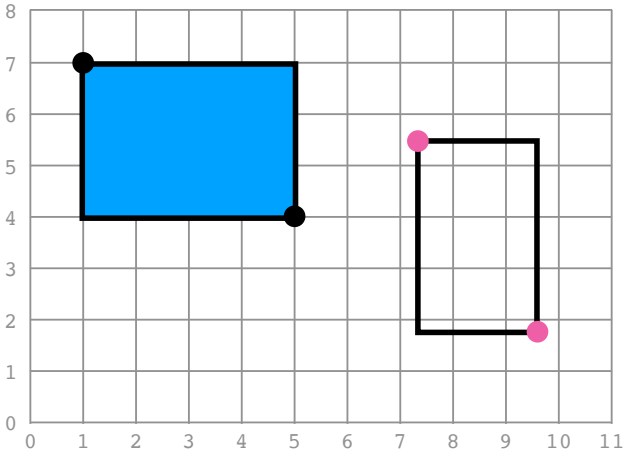
Ingalls Test for Extensibility



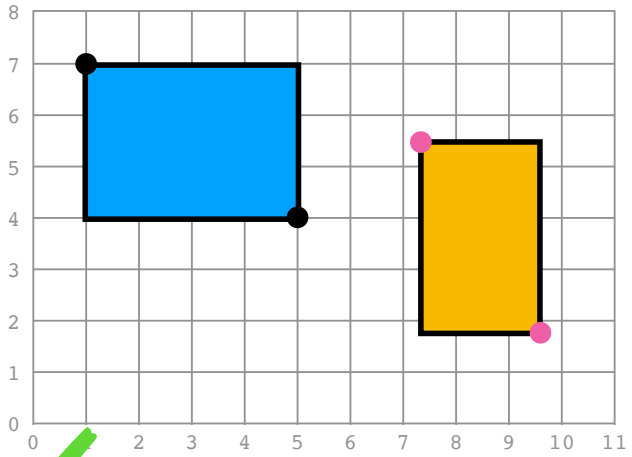
Ingalls Test for Extensibility



Ingalls Test for Extensibility



Ingalls Test for Extensibility

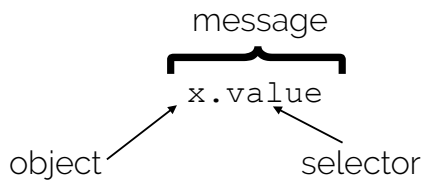


yes, it's object-oriented!

Ruby, Java, etc. pass the rectangle test

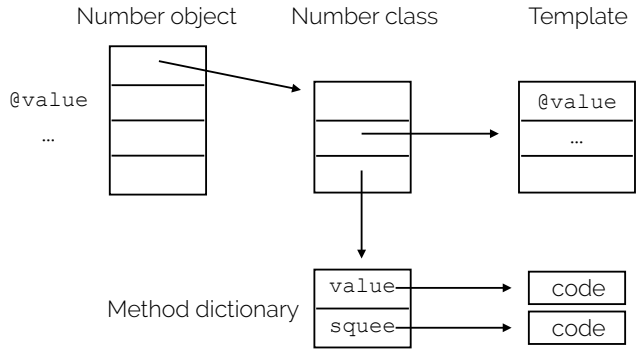
Dynamic Dispatch

- Dynamic dispatch is the OO mechanism for polymorphism.
- Functions ("methods") are always bound to an object (or class).
- A method is called ("dispatched") by sending a "message" to the "selector" of an object.

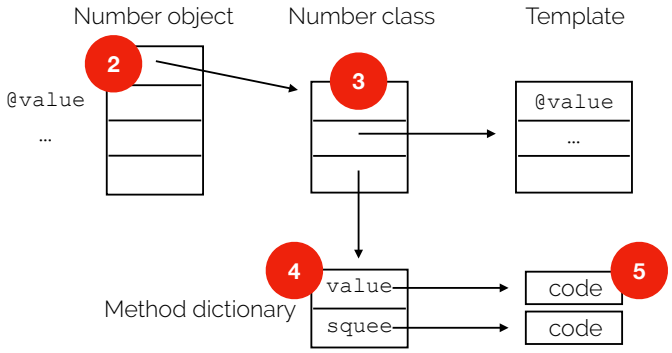


Dynamic Dispatch

- Dynamic dispatch is an algorithm for finding an object's method corresponding to a given selector name.

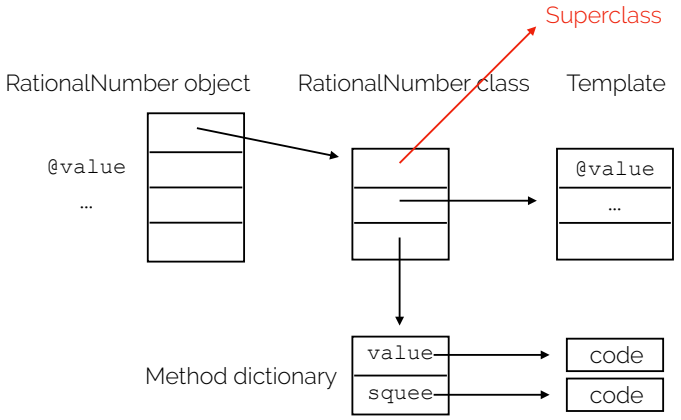


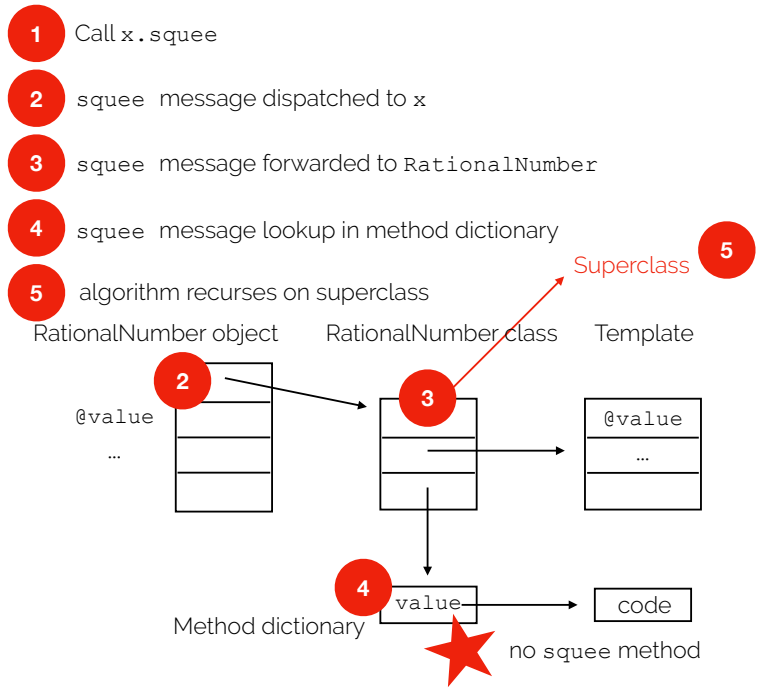
1. Call `x.value`
2. `value` message dispatched to `x`
3. `value` message forwarded to `Number`
4. `value` message lookup in method dictionary
5. `value` executed.



Inheritance

- One small change enables inheritance.





C++

Efficient object oriented programming.

"Only pay for what you use"

Consider the following Java program.

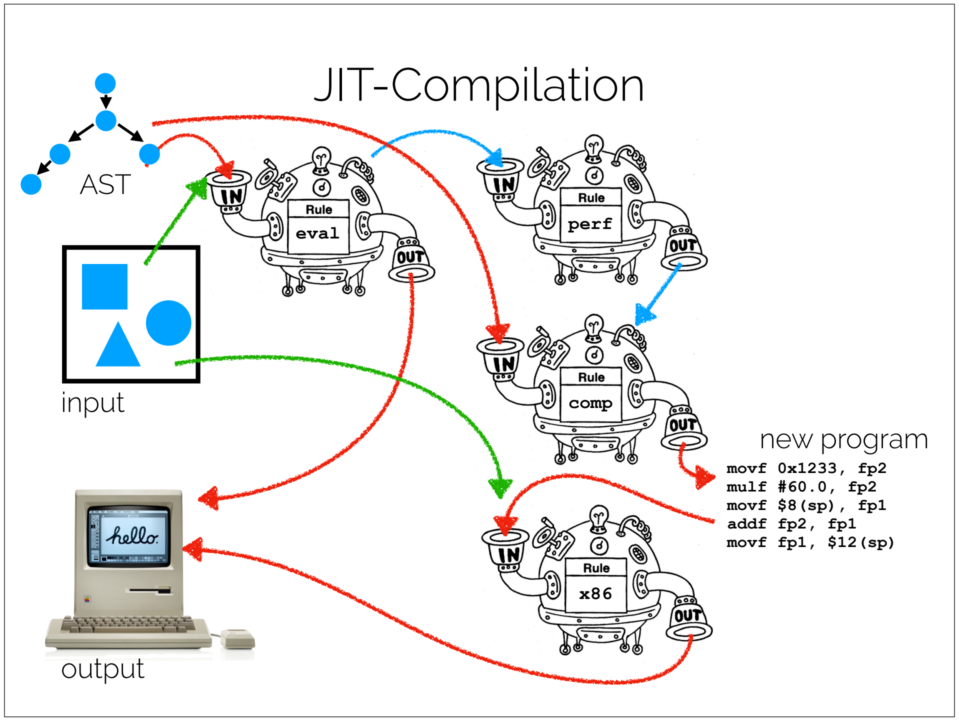
```
class Math {
 public static double mean(int[] nums, int len){
   int sum = 0;
   for (int i = 0; i < len; i++) {
     sum += nums[i];
   }
   return (double) sum / len;
 }
}
```

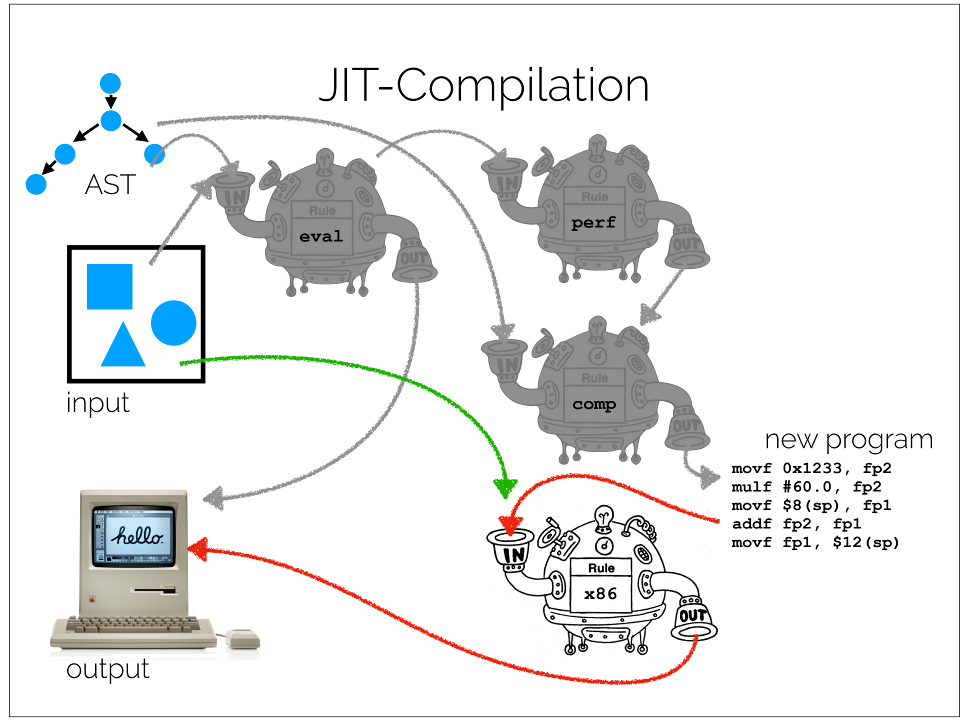
It uses **no dynamic dispatch**.

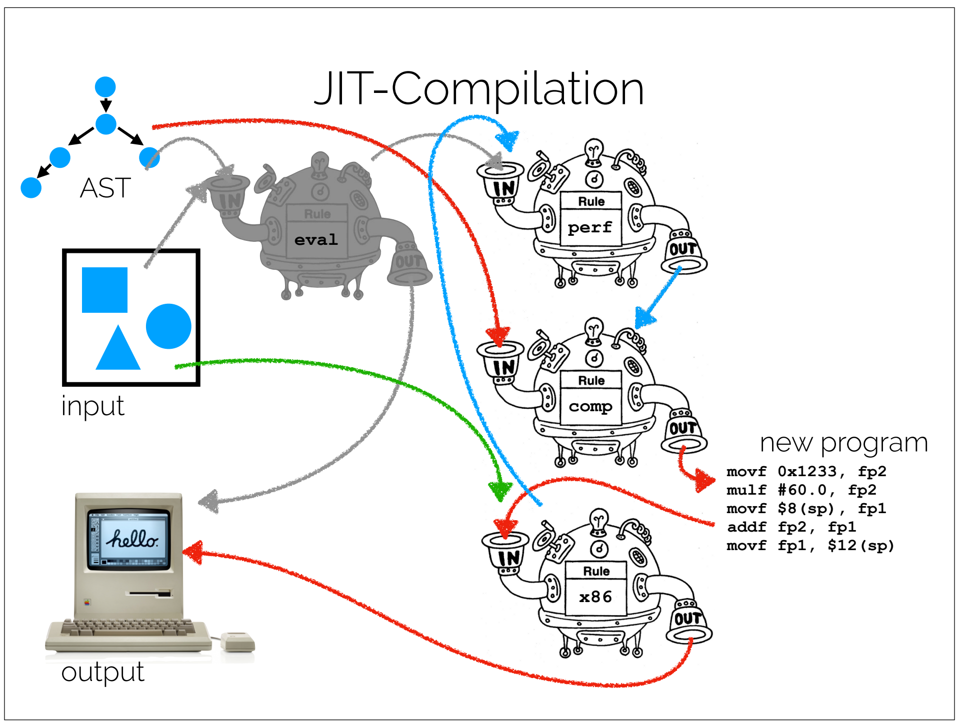
In fact, it barely uses any objects at all.

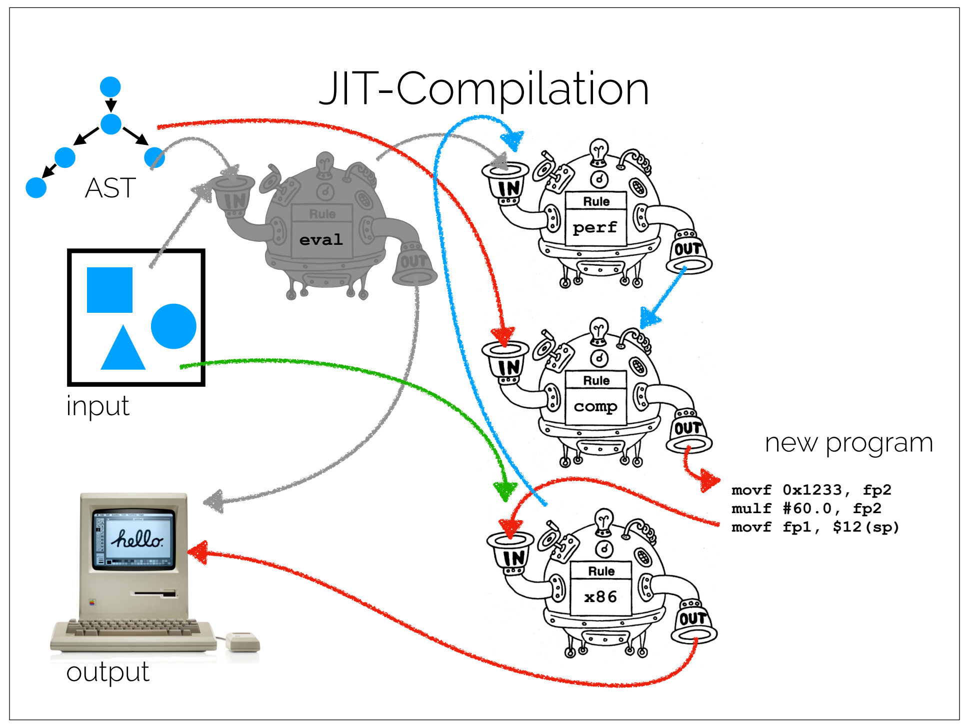
But Java still does **a lot of work** anyway…

1. boot up the Java Virtual Machine (JVM)
   a. allocate Java heap, stack, and global var areas
   b. start up garbage collector
   c. start up Just-in-Time performance monitor & compiler (JIT)
2. load first class definition (the one with `main`)
   a. verify bytecode for runtime safety
3. load all class defs for linked code (e.g., stdlib)
   a. verify, if necessary
4. allocate space for static variables
5. initialize static variables
6. execute main
   a. repeat loading, linking, verifying, allocation, and initialization steps as needed.
   b. periodically run the garbage collector
   c. run the JIT constantly, in a separate thread


JIT-Compilation
AST
input
output
Rule eval
Rule perf
Rule comp
Rule x86
new program
movf 0x1233, fp2
mulf #60.0, fp2
movf $8(sp), fp1
addf fp2, fp1
movf fp1, $12(sp)


JIT-Compilation
AST
input
output
eval
perf
comp
Rule x86
new program
movf 0x1233, fp2
mulf #60.0, fp2
movf $8(sp), fp1
addf fp2, fp1
movf fp1, $12(sp)


JIT-Compilation
AST
input
output
eval
Rule perf
Rule comp
Rule x86
new program
movf 0x1233, fp2
mulf #60.0, fp2
movf $8(sp), fp1
addf fp2, fp1
movf fp1, $12(sp)


JIT-Compilation
AST
input
output
eval
Rule perf
Rule comp
Rule x86
new program
movf 0x1233, fp2
mulf #60.0, fp2
movf fp1, $12(sp)

C++: “Only pay for what you use”

What does this mean?

```
class Math {
 public static double mean(int[] nums, int len){
   int sum = 0;
   for (int i = 0; i < len; i++) {
     sum += nums[i];
   }
   return (double) sum / len;
 }
}
```

C++: “Only pay for what you use”

What does this mean?

```
double mean(int nums[], int len) {
  int sum = 0;
  for (int i = 0; i < len; i++) {
    sum += nums[i];
  }
  return (double) sum / len;
}
```

In C++, the “no class” program is as fast as C

Without classes, C++ is essentially C

C++: Only Pay for What You Use

(demo OOP version)

C++: Only Pay for What You Use

(demo OOP version)

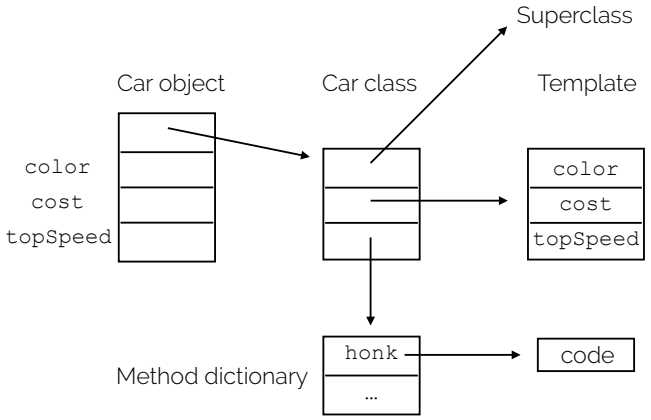
The version we came up with still **doesn’t pay for OO** because it wasn’t polymorphic!

C++ does OO efficiently

C++ static methods are just C procedures. No classes needed.

C++ eliminates lookups by computing locations at compile-time.

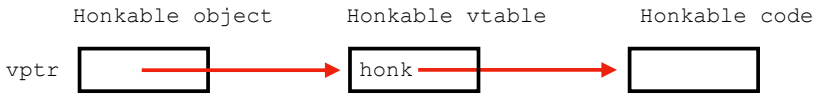
C++ also copies any needed superclass method pointers into class



C++: Only Pay for What You Use

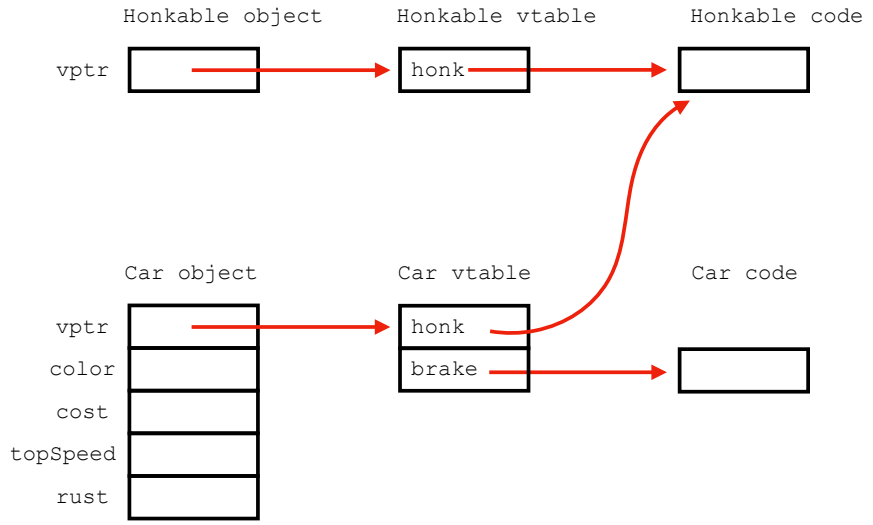
(demo polymorphic C++)

Virtual Dispatch

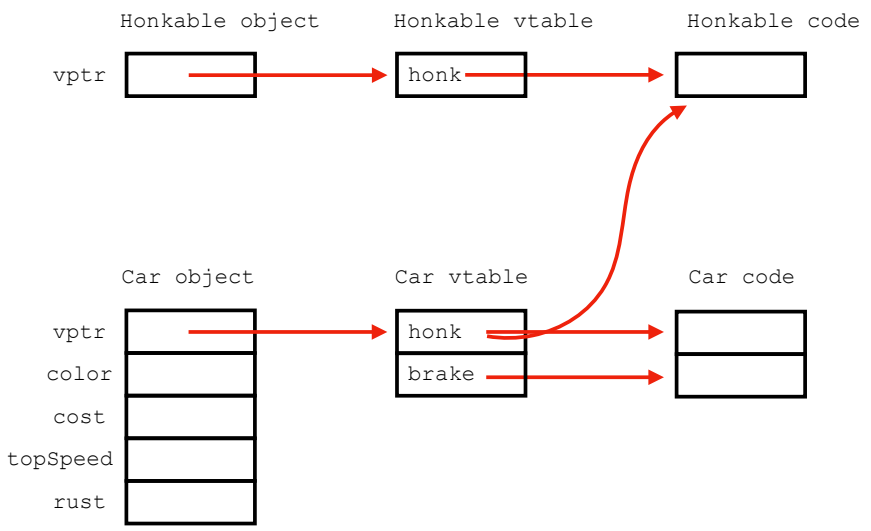


- Functions without the `virtual` keyword are just regular C functions (that also have access to class instance data).
- C++ virtual dispatch does *never searches* as in SmallTalk; vtable/instance variable offsets known at compile-time.

Virtual Dispatch (if I don't override **honk**)



Virtual Dispatch (if I do override **honk**)



Cost

1.dereference object
2.defererence class
~~3.dereference method dictionary~~ ~~for each class or superclass~~
4.dereference method

~~**O(n)** method lookup, where **n** is the number of superclasses.~~

O(1) method lookup

Recap & Next Class

This lecture:

C++: Only Pay for What You Use

Virtual Dispatch

Next lecture:

How to give a good talk