CSCI 334:
Principles of Programming Languages

Lecture 25: C++ / wrap up

Instructor: Dan Barowy

Williams

Topics

Why OO matters

Turing tarpits

Why PL matters

SCS

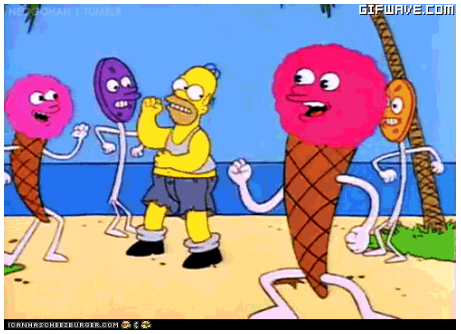
What's up with C++?

Announcements

1. **Senior thesis presentations** in Wege auditorium:
   a. Monday, **May 16, 9:30am-12:10 (2 credits!)**
   b. Monday, **May 16, 1:00-3pm (2 credits!)**
2. **Ward prize presentations** for best class project in Wege auditorium:
   Tuesday, **May 17, 2:30-4pm**

Announcements

1. **No colloquium** this week.
2. Instead: **end of year ice cream social** on Friday.



Your to-dos

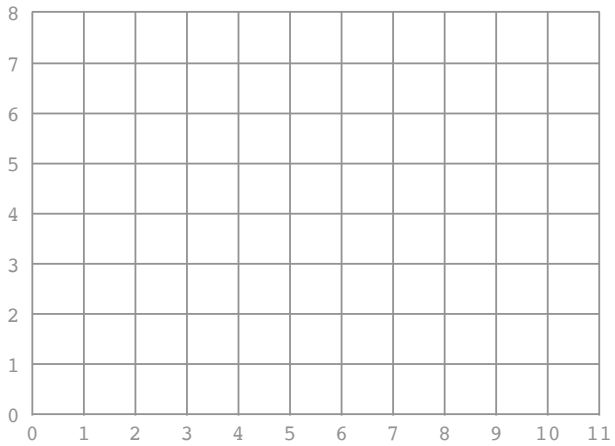
1. Lab 10, “mostly working” checkpoint, **due Sunday 5/15**

Ingalls Test for Extensibility

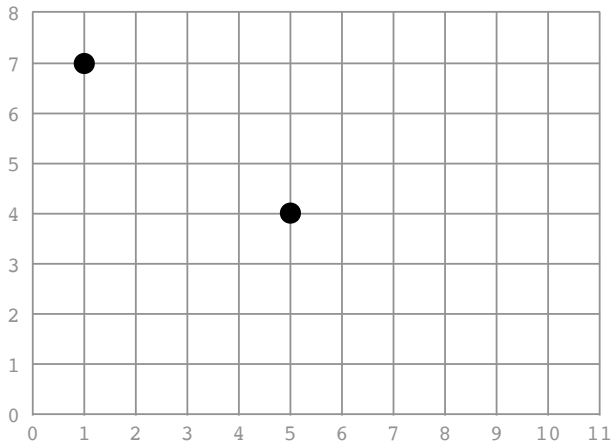
i.e., the “rectangle test”

- The test is about the ability to extend software *after* it has already been designed and written.

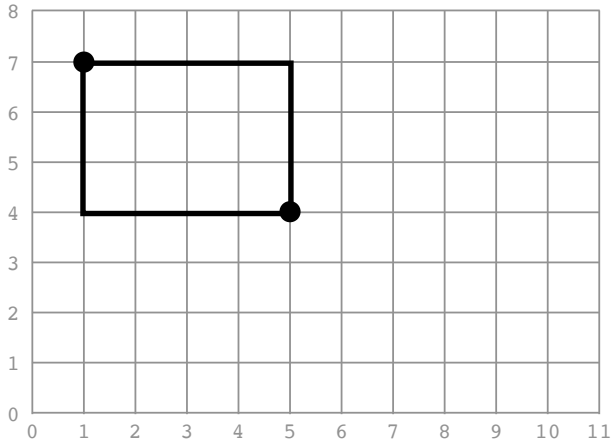
Ingalls Test for Extensibility



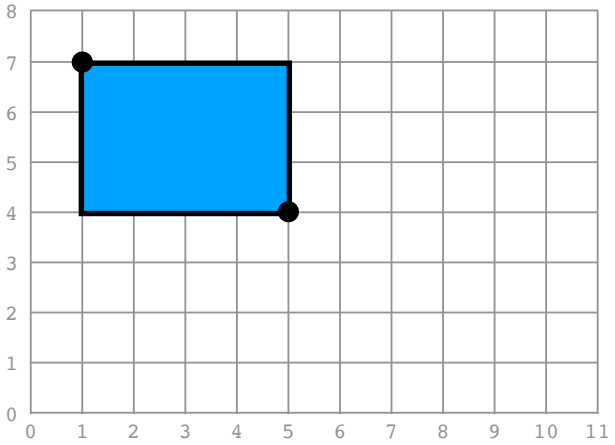
Ingalls Test for Extensibility



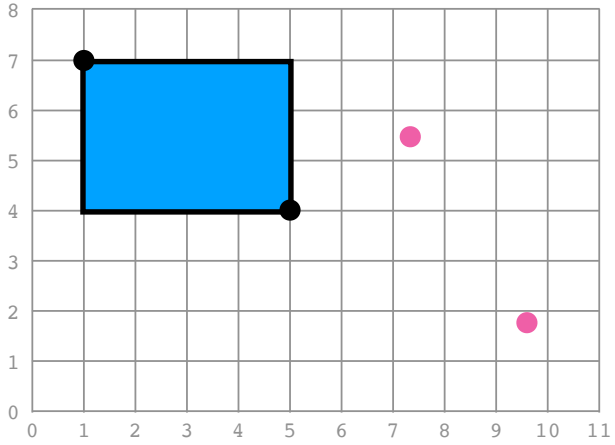
Ingalls Test for Extensibility



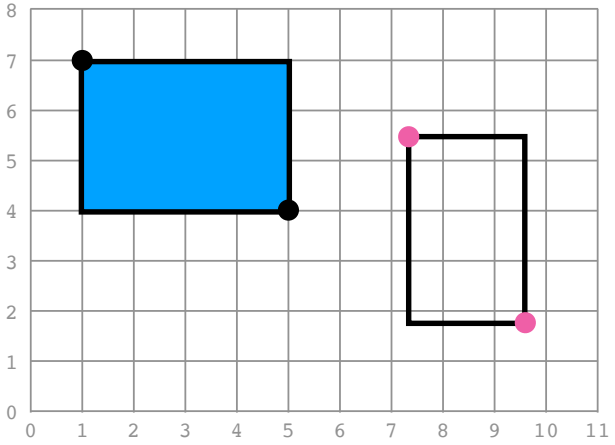
Ingalls Test for Extensibility



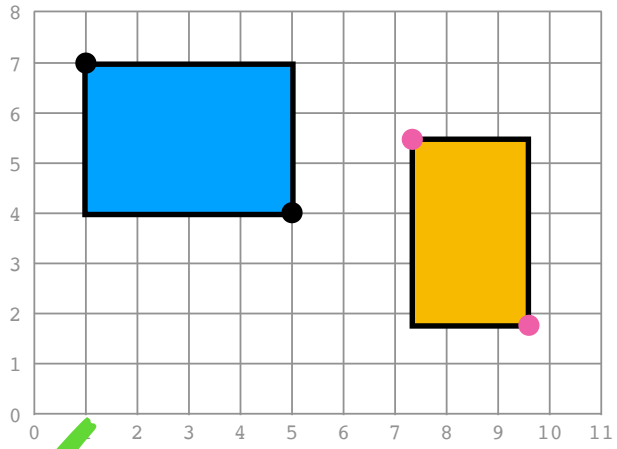
Ingalls Test for Extensibility



Ingalls Test for Extensibility



Ingalls Test for Extensibility



yes, it's object-oriented!

Java, Python, etc. pass the rectangle test

Turing Tarpit

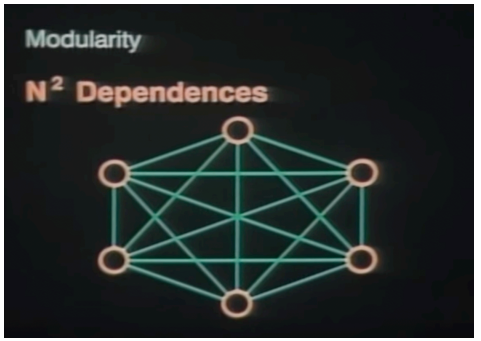
A **Turing tarpit** is a programming language flexible enough to do anything (i.e., it is **Turing equivalent**) while also being **difficult to learn and use** for everyday tasks.

"Beware of the Turing tar-pit in which everything is possible but nothing of interest is easy." —Alan Perlis

Examples:

- Turing machines
- The Lambda Calculus
- Breph
- C?

Why I like OO



OO is fundamentally based on **the idea that people matter** in the design of a programming language.

How do we **minimize human effort** while designing large pieces of software?

The right choice depends on the problem

- OO offers a **different kind of extensibility** than functional (or function-oriented) languages.
- Suppose you're **modeling a hospital**.

Operation	Doctor	Nurse	Orderly
`Print`	`Print Doctor`	`Print Nurse`	`Print Orderly`
`Pay`	`Pay Doctor`	`Pay Nurse`	`Pay Orderly`

- FP makes it **easy to add operations** (**rows** above).
- OOP makes it **easy to add data** (**columns** above).

Programming of the People, by the People, and for the People

Daniel Barowy

UMass Amherst

Williams College, January 9, 2017

In the future: There are no programmers.

A Bicycle for the Mind

This work is not done yet.

Evaluation Forms

(all of these are anonymous)

We **listen carefully** to what you say in these forms. Please take your time and write thoughtful responses.

Your feedback is **very valuable** to us!

Purpose of SCS Forms

"[T]he SCS provides instructors with feedback regarding their courses and teaching. The faculty legislation governing the SCS provides that SCS results are made available to the appropriate department chair, the Dean of the Faculty, and at appropriate times, to members of the Committee on Appointments and Promotions (CAP). The results are considered in matters of faculty reappointment, tenure, and promotion."

—Office of the Provost, Williams College

Purpose of "Blue Sheets"

Student comments on the blue sheets […] are solely for your benefit. They are not made available to department or program chairs, the Dean of the Faculty, or the CAP for evaluation purposes.

—Office of the Provost, Williams College

Blue sheet prompts:

* What **course topic** did you **enjoy the most**?

* What **course topic** did you **least enjoy**? Do you think that it was valuable to learn anyway?

* Are there **other aspects** of the course that you **liked** or **disliked**? (E.g., *office hours*, *TAs*, *assignments*, *course structure*, *meeting times*, etc.) Feel free to suggest alternative approaches!

* Did you **look forward to coming to class**?

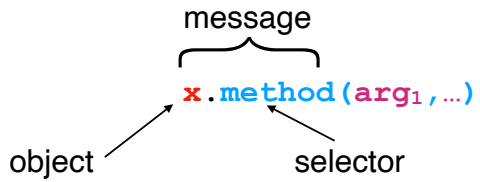
Dispatch

- **Dispatch** is how a function call works.
- We've seen many examples this semester.
- Ordinary functions can be dispatched **statically**, meaning that deciding what to do can be **determined at compile time**.

```
method(arg1,…)
```

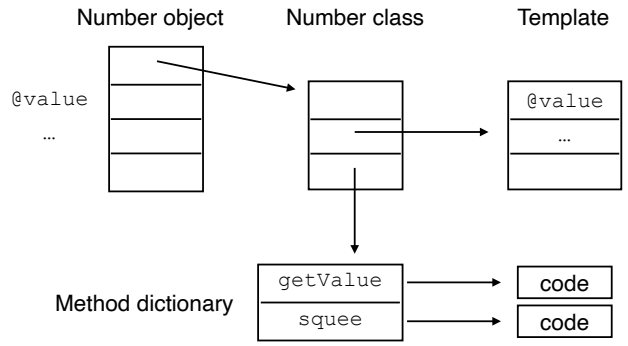
Dynamic Dispatch

- **Dynamic dispatch** is the OO mechanism for **polymorphism**.
- OO functions are dispatched **dynamically**, because they **depend on data**.
- This means that what they do must be **determined at runtime**.
- A method is called ("**dispatched**") by sending a "**message**" to the "**selector**" of an object.

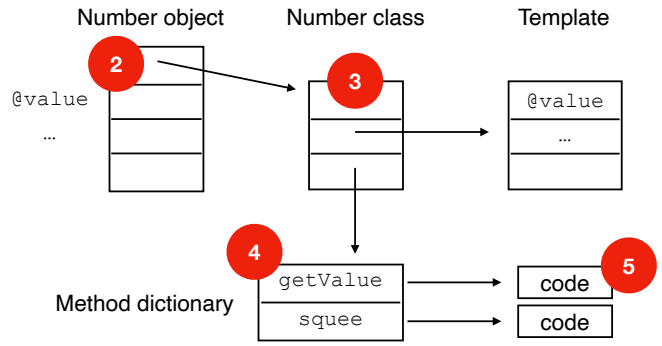


Dynamic Dispatch

- Dynamic dispatch is an **algorithm** for finding a the **implementation** for a given **selector** (i.e., method).



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



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;
 }
}
```

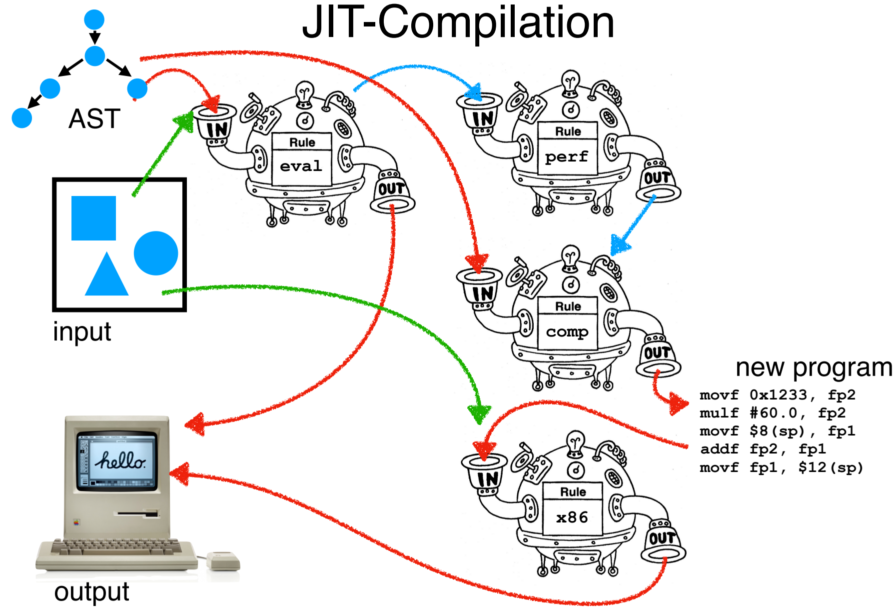
It uses **no dynamic dispatch**.

In fact, it barely uses any objects at all.

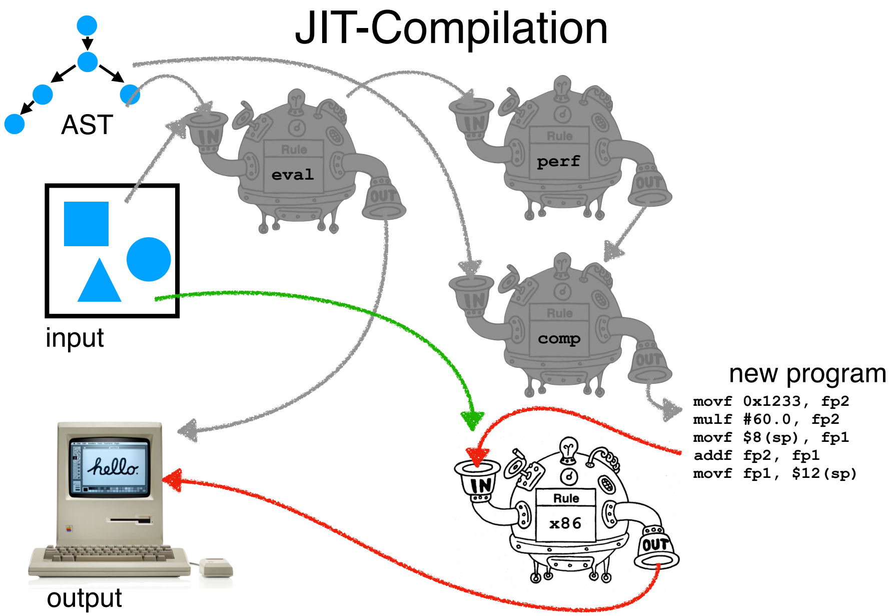
But Java still does **a lot of work** anyway…

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

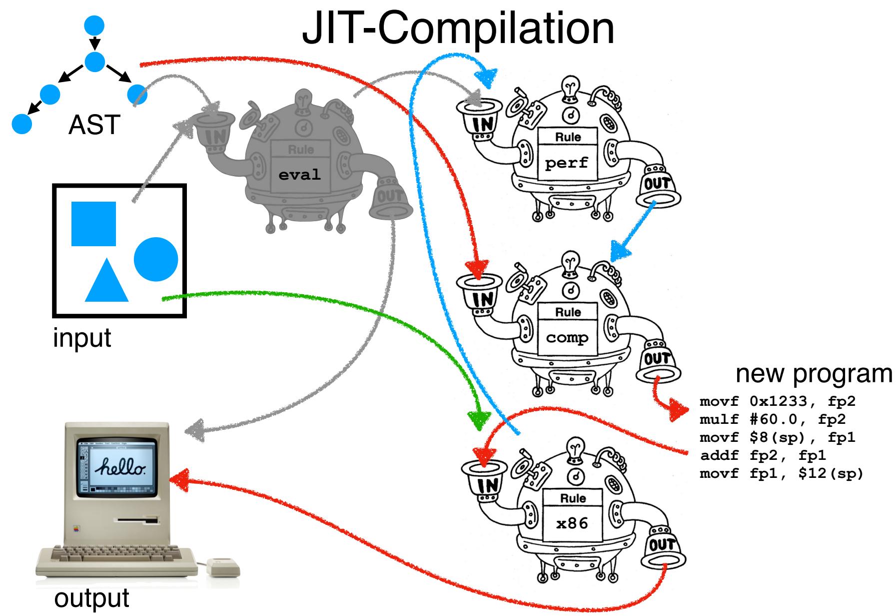
JIT-Compilation

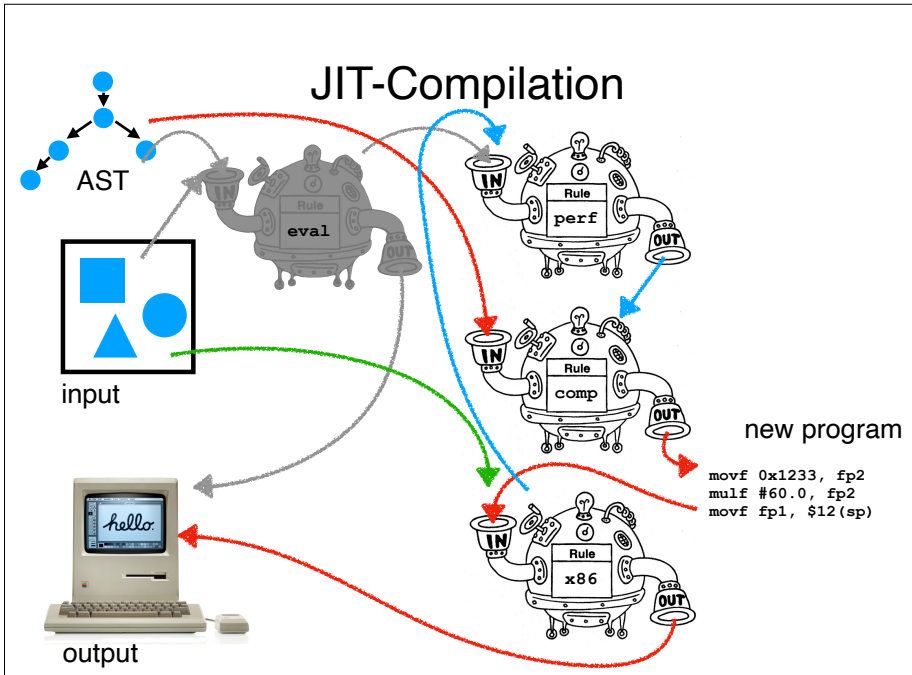


JIT-Compilation



JIT-Compilation





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)

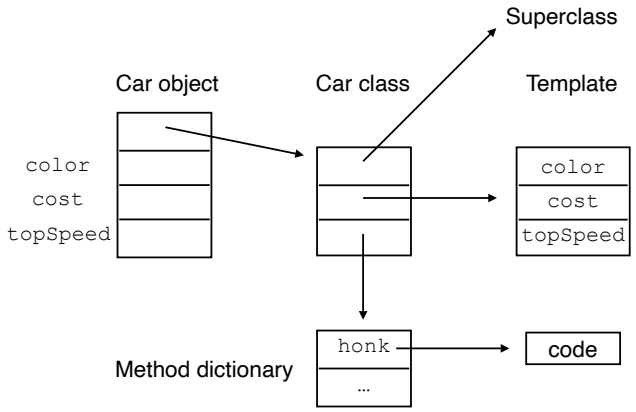
The version we came up with still **doesn't pay for OO** because it wasn't polymorphic!

C++ does OO efficiently

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

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

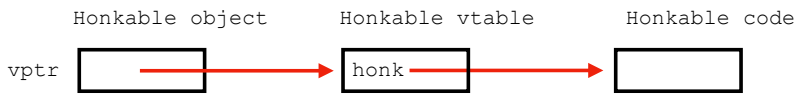
C++ copies any needed superclass method pointers into class



C++: Only Pay for What You Use

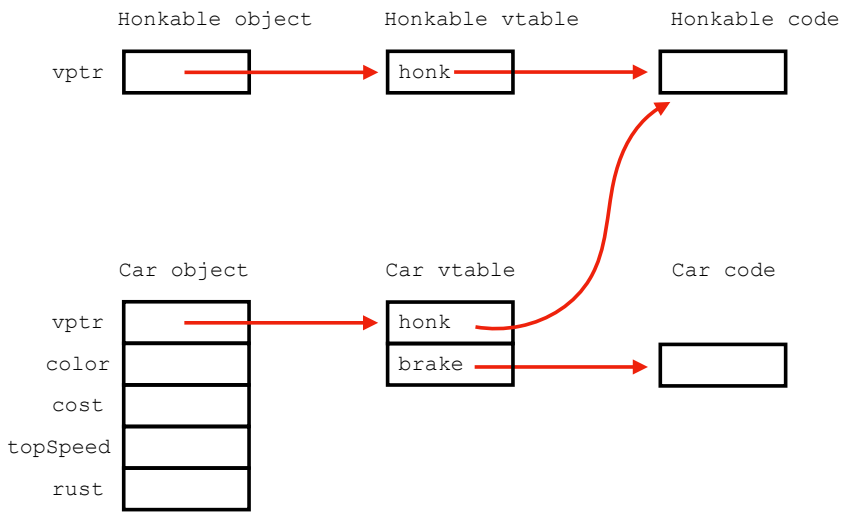
(demo polymorphic C++)

Virtual Dispatch

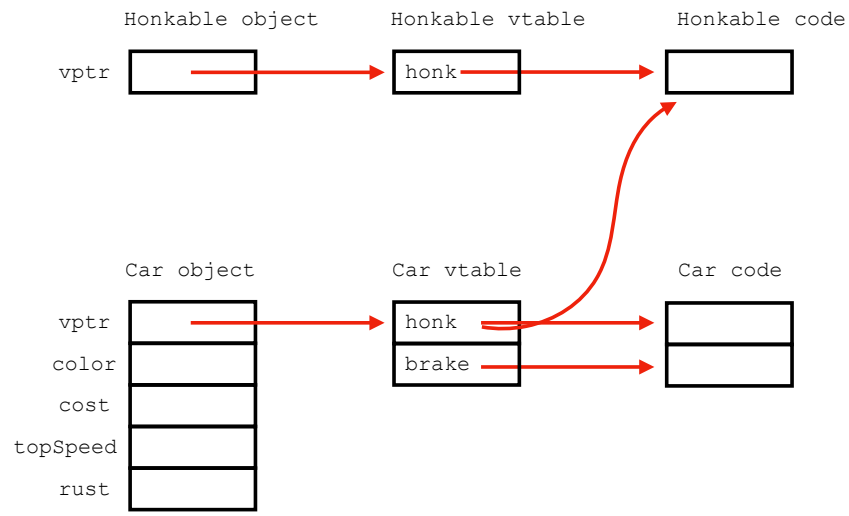


- Functions without the `virtual` keyword are just regular C functions (that also have access to class instance data).
- C++ virtual dispatch does *never searches* as in SmallTalk; vtable/instance variable offsets known at compile-time.

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



Virtual Dispatch (if I do override `honk`)



Cost

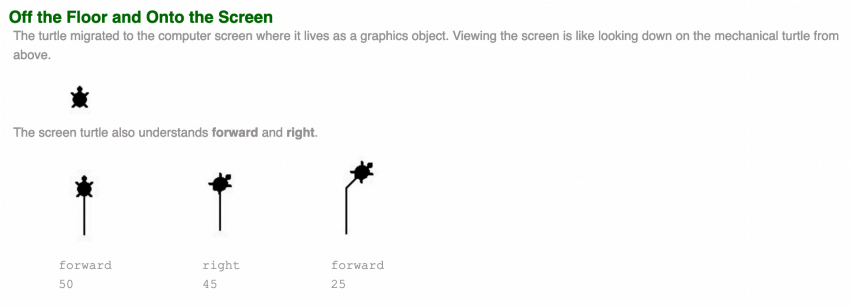
1. dereference object
2. defererence class
3. ~~dereference method dictionary~~ } ~~for each class or superclass~~
4. dereference method

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

O(1) method lookup

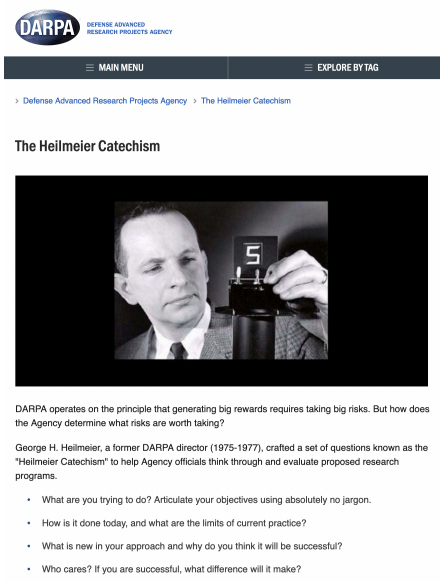
What to talk about in your presentation / tutorial

Tutorial



https://el.media.mit.edu/logo-foundation/what_is_logo/logo_primer.html

Video presentation



Next steps
(aka, some things to do over the summer)

- **Teach yourself** another programming language.
- Dig in to **a problem that bugs you**.
  (me: I've always wanted to write a computer algebra solver)
- **Keep playing** with your project! It's yours! (and you should **show it off** to interviewers)
- Most of all, **do something that excites you**.

Recap & Next Class

This lecture:

More OOP

Why PL

C++

Next lecture:

No next lecture! Have a great summer!