12 sordid facts the language manual *won't tell you* about dynamic method dispatch.

#8 will blow your mind!!!

(or: Thinking about run-time class structures to help in thinking about their representation in the AST)

What's a method call?

What does o.m(x,y) actually do?

What if m is an inherited method?

What if m is an overridden method?

What if it's (w.n(x)).m(x,y)?

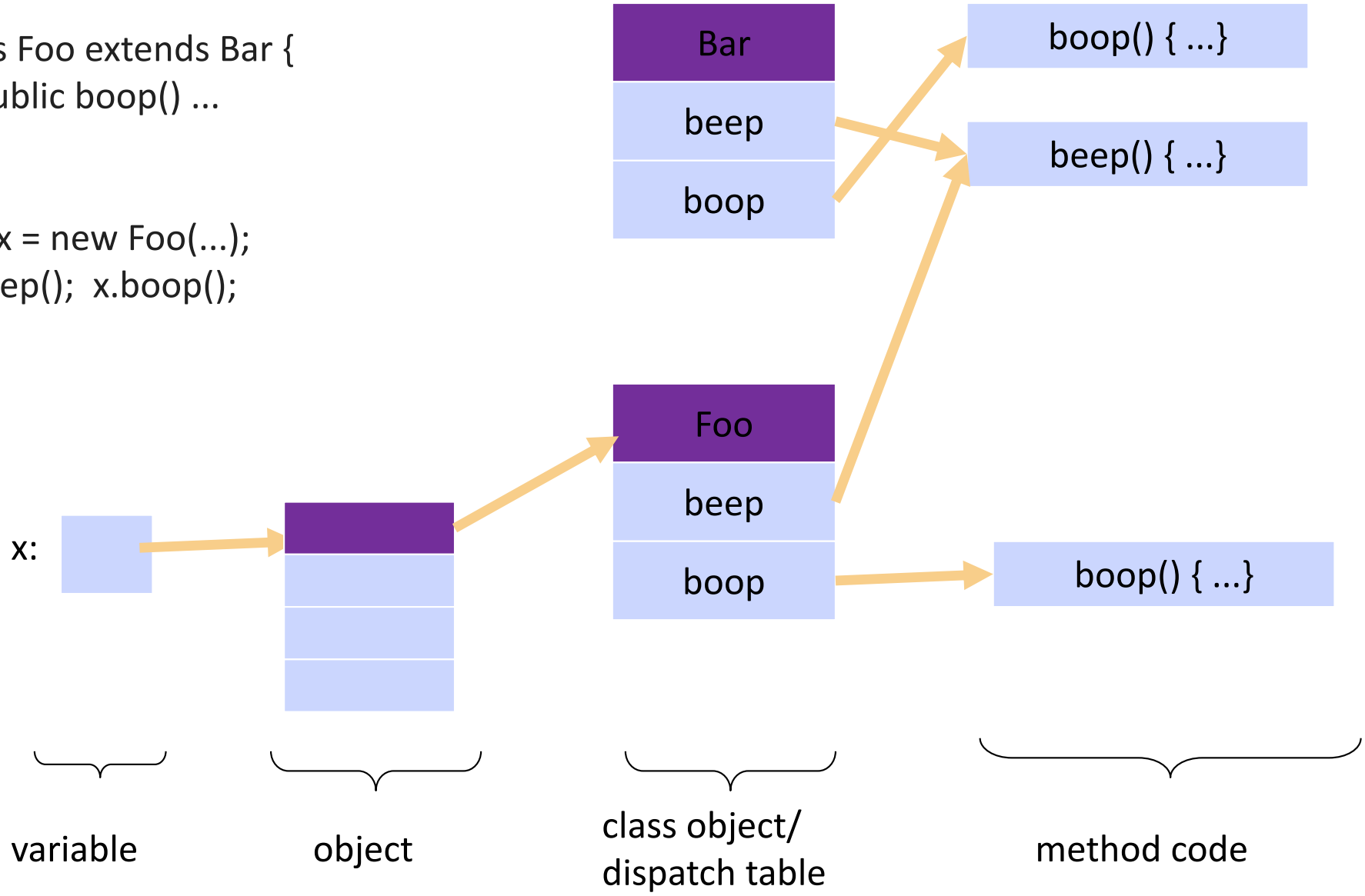
What if (in Java) it's

```
Bar o = new Foo();
o.m(x,y);
```

OO Type Tags & Dispatch

```
class Foo extends Bar {
    public boop() ...
}

Bar x = new Foo(...);
x.beep();  x.boop();
```



Java and Quack Use Static and Dynamic typing

Variables have **static** type

Objects have **dynamic** (tagged) type

- Any subclass of the static type

The tag is (usually) a reference to an object of class Class

- For dynamic dispatch of methods; also for “instanceof” or other type-based dispatch

Static Types in OO

The static type is a promise:

- Field f will appear in the third slot of the object
- The 7th slot of the vtable is a method
  foo(int, int, String) : String
- We can use these slots without checking their type at run-time

In other words: Static type is a *conservative approximation* of the dynamic type

What's in a Class?

- A reference to the superclass
- A list of methods
  - In the same order as the superclass; new methods appended at end
  - Including inherited and overridden methods
    - Inherited methods refer to methods in the parent class; you'll have to clean this up after building the class hierarchy
    - All overridden methods must be compatible
- A sequence of statements
- Instance variables (a.k.a. fields, data members)
  - In Quack: Derive the fields from the list of statements

Compatibility of overridden methods

```
class Super() {
   def foo(x: C) : R { … }
}

class Sub() extends Super() {
    def foo(x: D): S { … }
}
```

What is the required relation of D to C, and S to R?