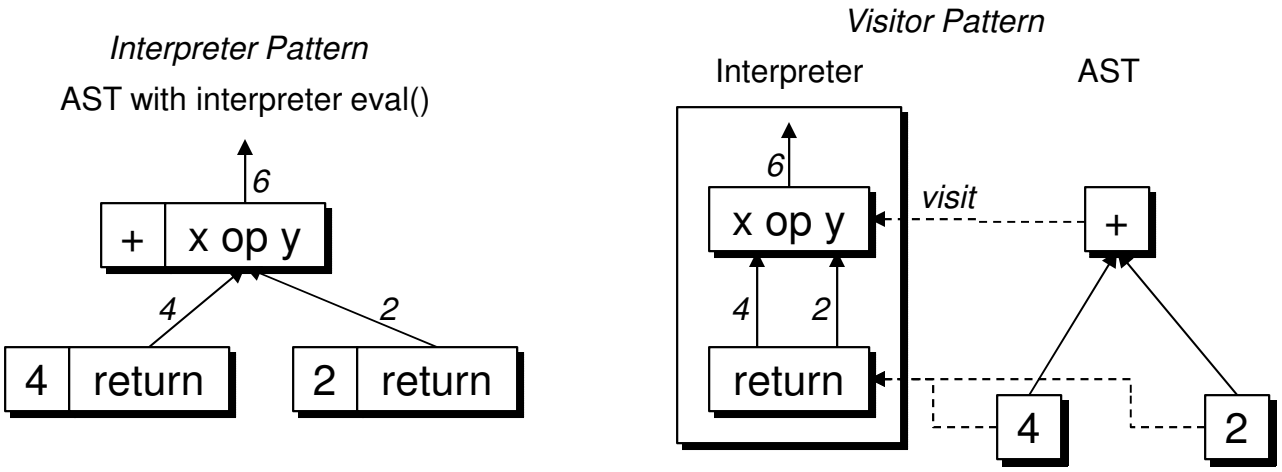
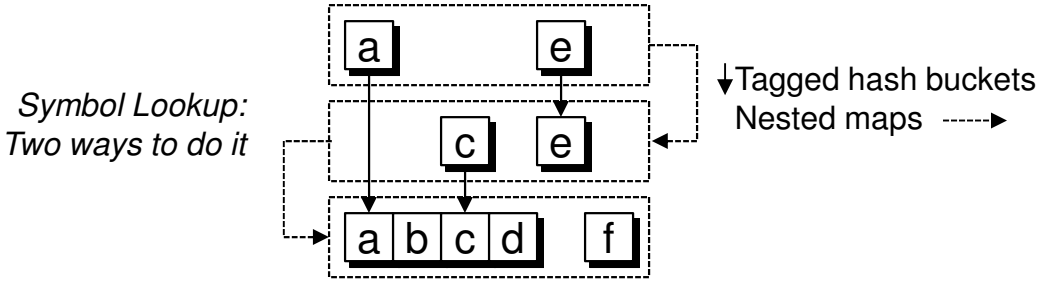
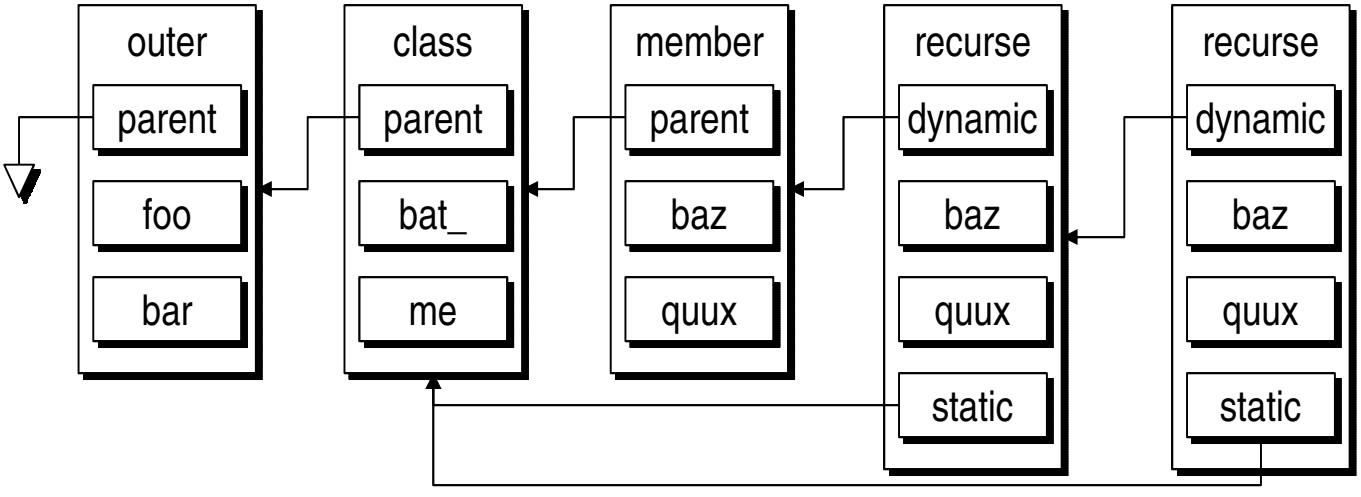
- Parser examples (pyparsing, Python 3):
  - (Word(alphas) + "/" + Word(alphas)).parseString("42/1337") => `['42', '/', '1337']`
  - infixNotation(Word(nums),[('*',2,opAssoc.LEFT),('+',2,opAssoc.LEFT)]).parseString("1+2*3")
    =>`['1', '+', ['2', '*', '3'] ]`
- So what do you do with that array once you have it?
- In a compiler, the Abstract Syntax Tree is just a step on the way. In an interpreter, the AST is the whole deal.





Nested Environments, a.k.a. Stack Frames



Decisions to make

- Paradigm(s): Imperative, functional, object-oriented, … ← *Don't skimp on this one!*
- Semantics: value, reference, … ← *Nor this one!*
- Pass count: parse, compile, link, fixup, …, execute
  - in any combo. E.g., 2-pass parse/exec vs. 1-pass parse+exec.
- AST: active (interpreter) vs. passive (visitor) (over, top)
- Environments/scopes/stack frames: hash tables vs. nested maps (over, bottom)
  - Parse-time and run-time same structure or different?
- How much do you want your interpreter to behave like Real HW?

Parse Time vs. Run Time

- The nested symbol structure you build at parse time is similar to a runtime stack.
- But any given nesting level at parse time can repeat many times at runtime – recursion.

Keeping Everything Straight

- Duck typing vs. your language's typing
- The implementation language's stack vs. your program's stack
  - E.g., do you use the implementation language's stack or provide your own stack?
- The implementation language's scope vs. your program's scope
  - Likewise implementation's vs. your classes, instances

What makes it harder?

- Recursion – you have to declare methods first, then inject the bodies later
  - Trampolines, rewrites, lookups
- Multithreading – the static stack and the dynamic stack may be unrelated
- Multiple inheritance – enough said :)

Multiline REPL – how do you know when you're done?

- My cheap way: Add `\n`. If the error is on the line after, you're not done!

Keeping going after syntax errors – unwinding partial commands

- Keep track of declarations and state changes, then remove them if the whole parse doesn't succeed.

Error handling and reporting

- Recovering gracefully from errors in your implementation
- Reporting errors in the program being interpreted
- How to totally confuse your users (or preferably not)