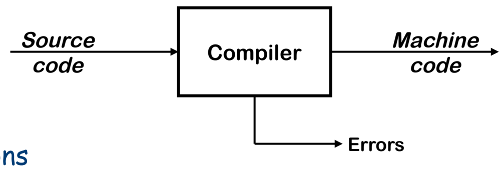
COMP 412
FALL 2010

The View from 35,000 Feet
Comp 412

Note by Baris Aktemur:
Our slides are adapted from Cooper and Torczon's slides that they prepared for COMP 412 at Rice.

High-level View of a Compiler

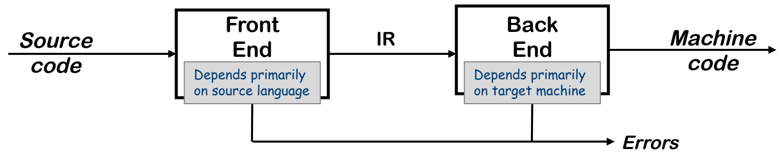


Implications

- Must recognize legal (and illegal) programs
- Must generate correct code
- Must manage storage of all variables (and code)
- Must agree with OS & linker on format for object code

Big step up from assembly language—use higher level notations

Traditional Two-pass Compiler



Implications

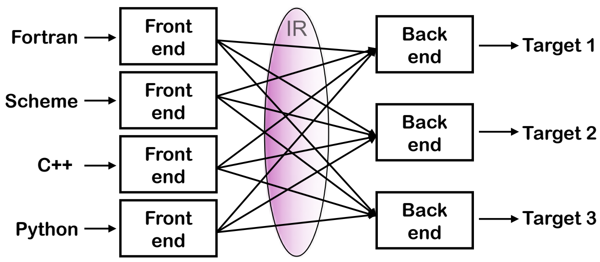
- Use an intermediate representation (IR)
- Front end maps legal source code into IR
- Back end maps IR into target machine code
- Admits multiple front ends & multiple passes (*better code*)

Classic principle from software engineering: Separation of concerns

Typically, front end is O(n) or O(n log n), while back end is NPC

Comp 412, Fall 2010 2

A Common Fallacy



Can we build *n x m* compilers with *n+m* components?

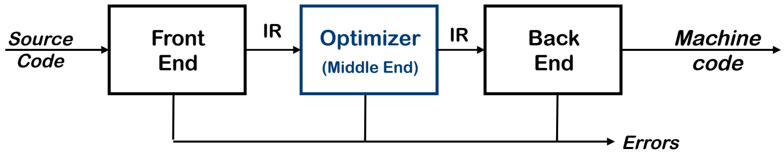
- Must encode all language specific knowledge in each front end
- Must encode all features in a single IR
- Must encode all target specific knowledge in each back end

Successful in systems with assembly level (or lower) IRs

e.g., gcc's rtl or llvm ir

Comp 412, Fall 2010 3

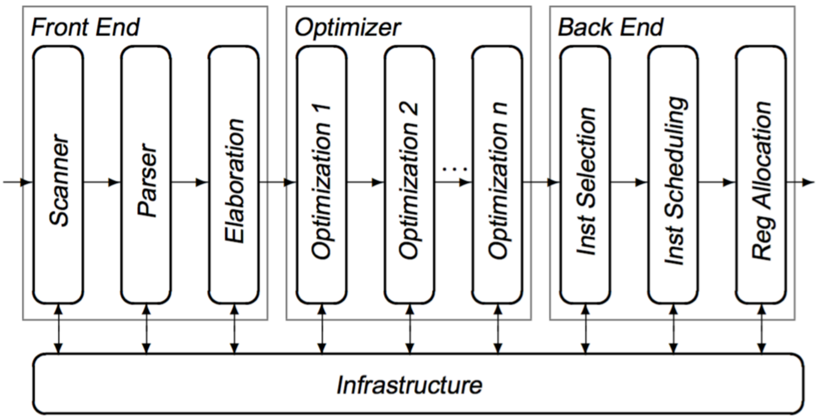
Traditional Three-part Compiler



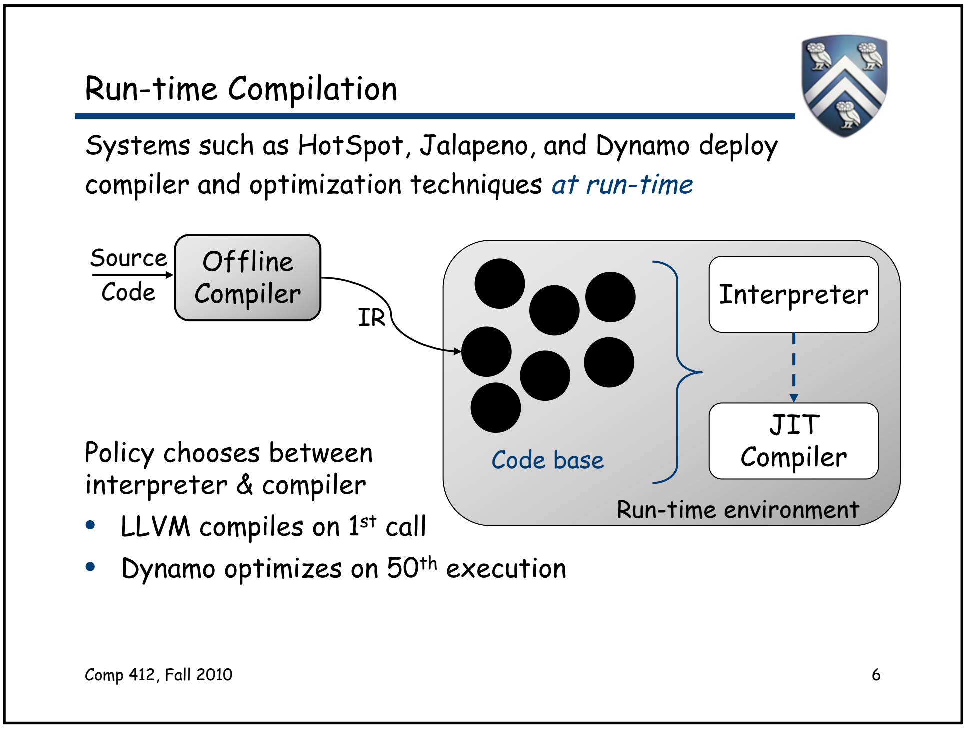
Code Improvement (or *Optimization*)

- Analyzes IR and rewrites (or *transforms*) IR
- Primary goal is to reduce running time of the compiled code
  - May also improve space, power consumption, …
- Must preserve "meaning" of the code
  - Measured by values of named variables

Traditional Three-part Compiler



■ **FIGURE 1.1** Structure of a Typical Compiler.


Run-time Compilation
Systems such as HotSpot, Jalapeno, and Dynamo deploy compiler and optimization techniques at run-time
Source Code
Offline Compiler
IR
Code base
Interpreter
JIT Compiler
Run-time environment
Policy chooses between interpreter & compiler
• LLVM compiles on 1st call
• Dynamo optimizes on 50th execution
Comp 412, Fall 2010
6