


# **CPEG 421/621**

## **- Fall 2010**



### **Topics I**

### **Fundamentals**

# Topic I: Outline



- Part I: Compiler Fundamentals
  - An Overview on Compiler Design
  - Compiler Front-End and IR
  - Middle-End: Analysis and Optimizations
  - Back-End: Code Generation and Optimization

# Foundations for Compiler Design



1. Processor architecture design flow
2. Compiler structure and design flow
3. Code generation design flow

# Why Study Compilers?

- Influences on programming language design
- Influences on computer design
- Compiling techniques are useful for software development
  - Parsing techniques are often used
  - Learn practical data structures and algorithms
  - Basis for many tools such as text formatters, structure editors, silicon compilers, design verification tools,...

**Writing a compiler requires an understanding of almost all important CS subfields**

# Architecture Models

Vector architectures, SIMD

Instructional Level Parallelism (ILP)

superscalar

VLIW

Multithreaded Architectures

Chip multiprocessing (CMP, multi-core, many-core, etc.)

GPGPUs

Reconfigurable Architectures

**What is the impact of these ideas on compilers?**

**Arch./Compiler  
and System  
Software  
Design Toolset**

ISA  
Simulator

System Level  
Simulator

**Processor Architecture  
Design Flow  
Diagram**

Instruction Set  
Architecture  
Design  
(Microarchitecture  
Design-I)

System-Level  
Design

Compiler  
Design

Code  
Optimizer

Code  
Generator

Toolchain

- Intel VTune™
- IBM Performance Evaluator

Debugger

RTL Level Design  
(Microarchitecture  
Design II)

Hardware  
Design

Switch  
Level  
Design

Circuit  
Level  
design

RTL  
Level  
Simulator

Switch  
Level  
Simulator

Circuit  
Level  
Simulator

HDL (VHDL or Verilog)

# What does a Compiler do Anyway?

The image shows a screenshot of Microsoft Visual Studio 2010. The main window displays the source code for a C++ program named `Hello.cpp`. The code is as follows:

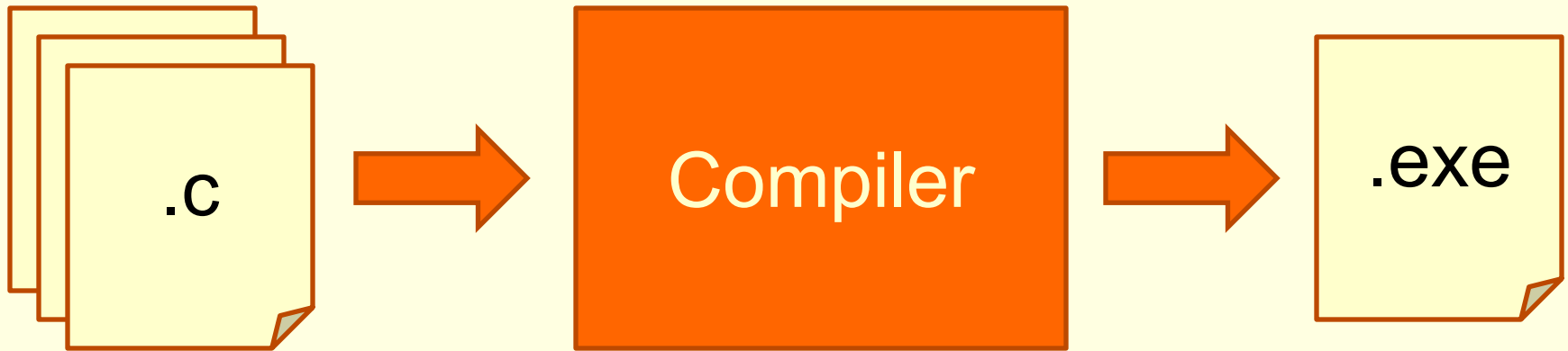
```
1 // Hello.cpp : Defines the entry point for the console application.
2 //
3
4 #include "stdafx.h"
5 #include <iostream>
6
7 int main(int argc, char* argv[])
8 {
9     std::cout << "Hello, world!" << std::endl;
10    std::cout << "Press any key then enter to finish the program" << std::endl;
11    char c;
12    std::cin >> c;
13    return 0;
14 }
15
16
```

The Solution Explorer on the left shows the project structure for 'Hello'. The Locals window on the right is empty. A console window in the foreground shows the output of the program:

```
c:\users\lasher\documents\visual studio 2010\Projects\Hello\Release\Hello.exe
Hello, world!
Press any key then enter to finish the program
-
```

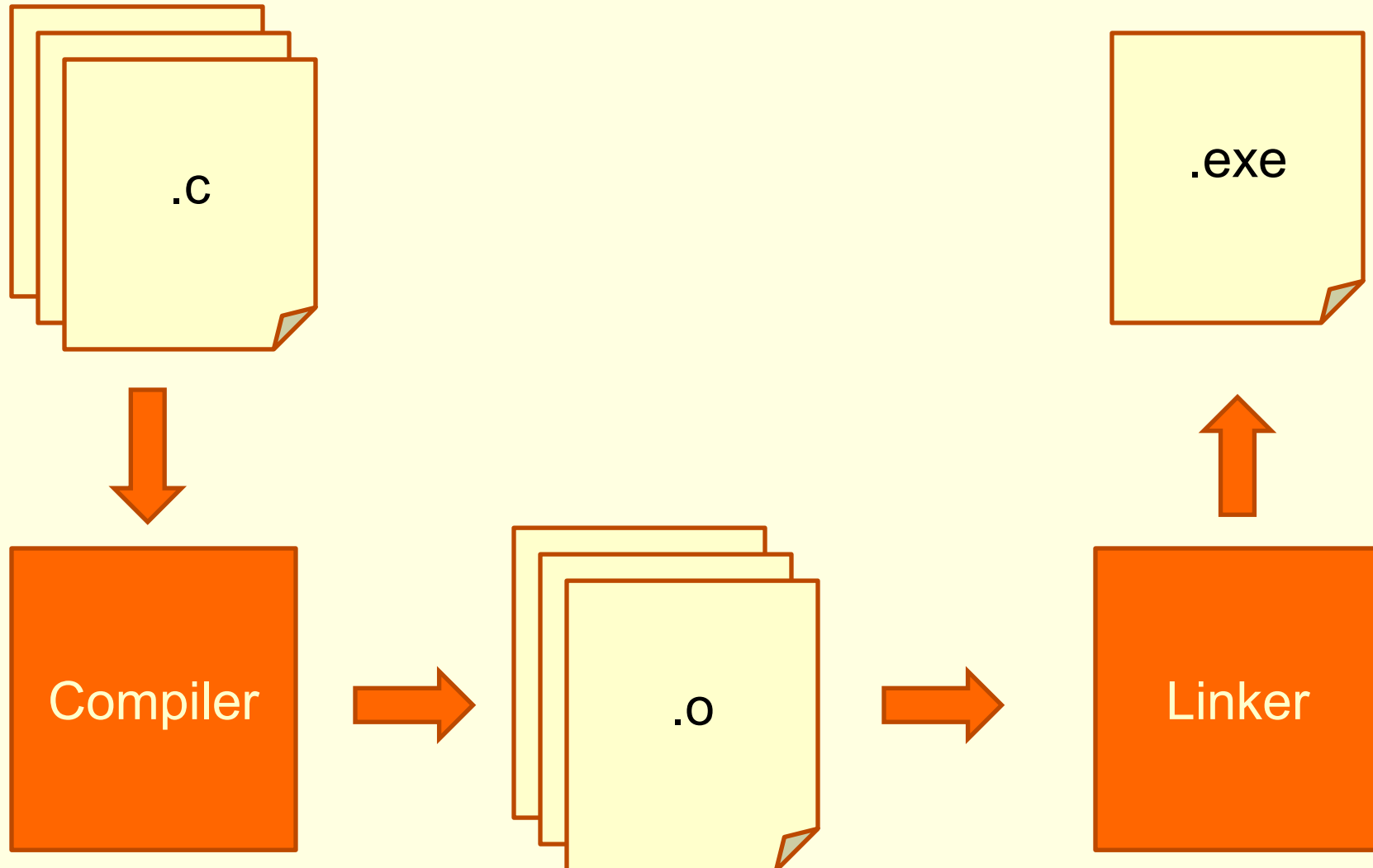
The Output window at the bottom right shows the debug output, including system messages about loaded DLLs. The Windows taskbar at the bottom shows the system tray with the date and time: 11:33 AM 2/7/2012.

# What does a Compiler do Anyway? (Cont'd)

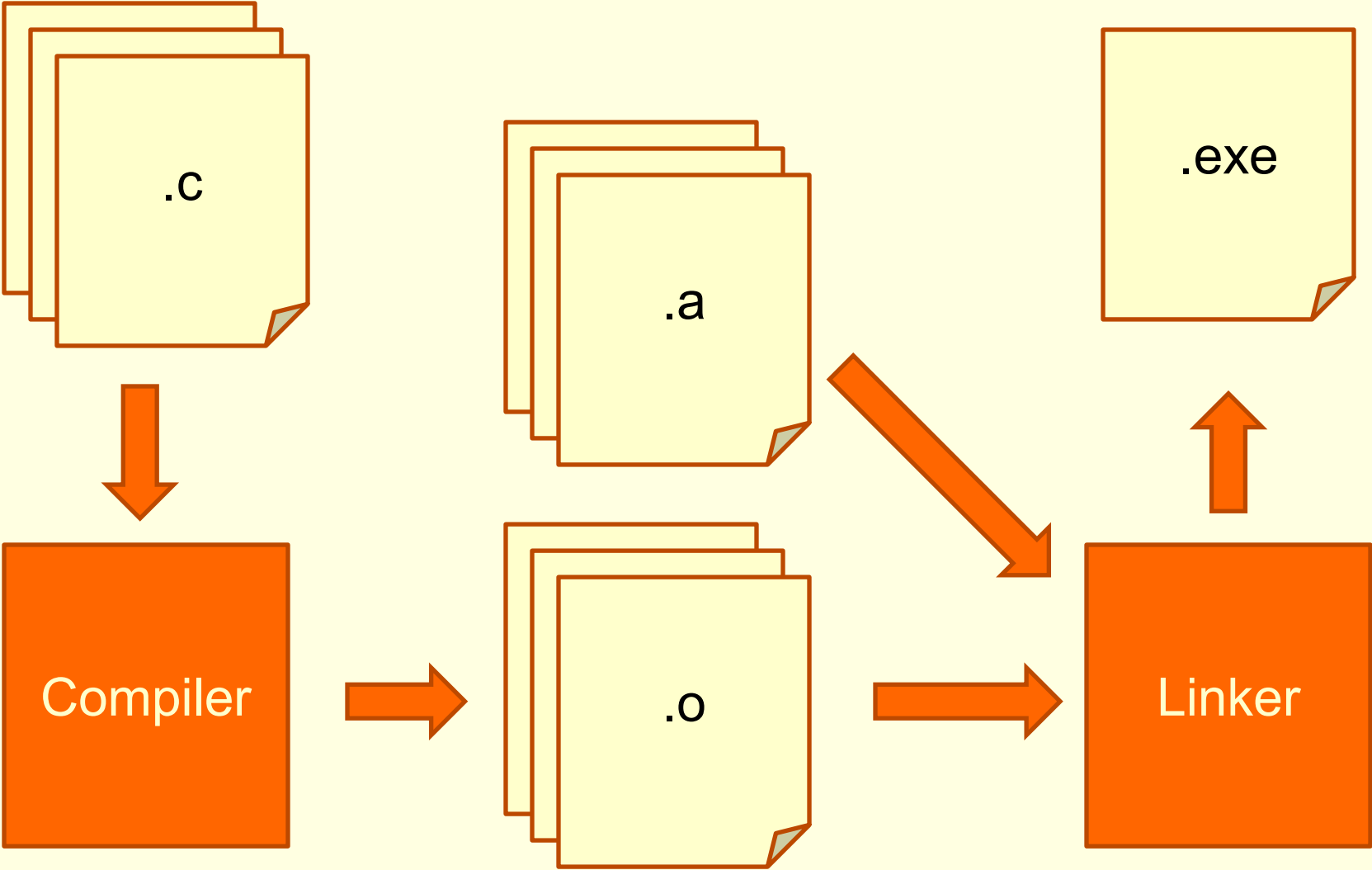




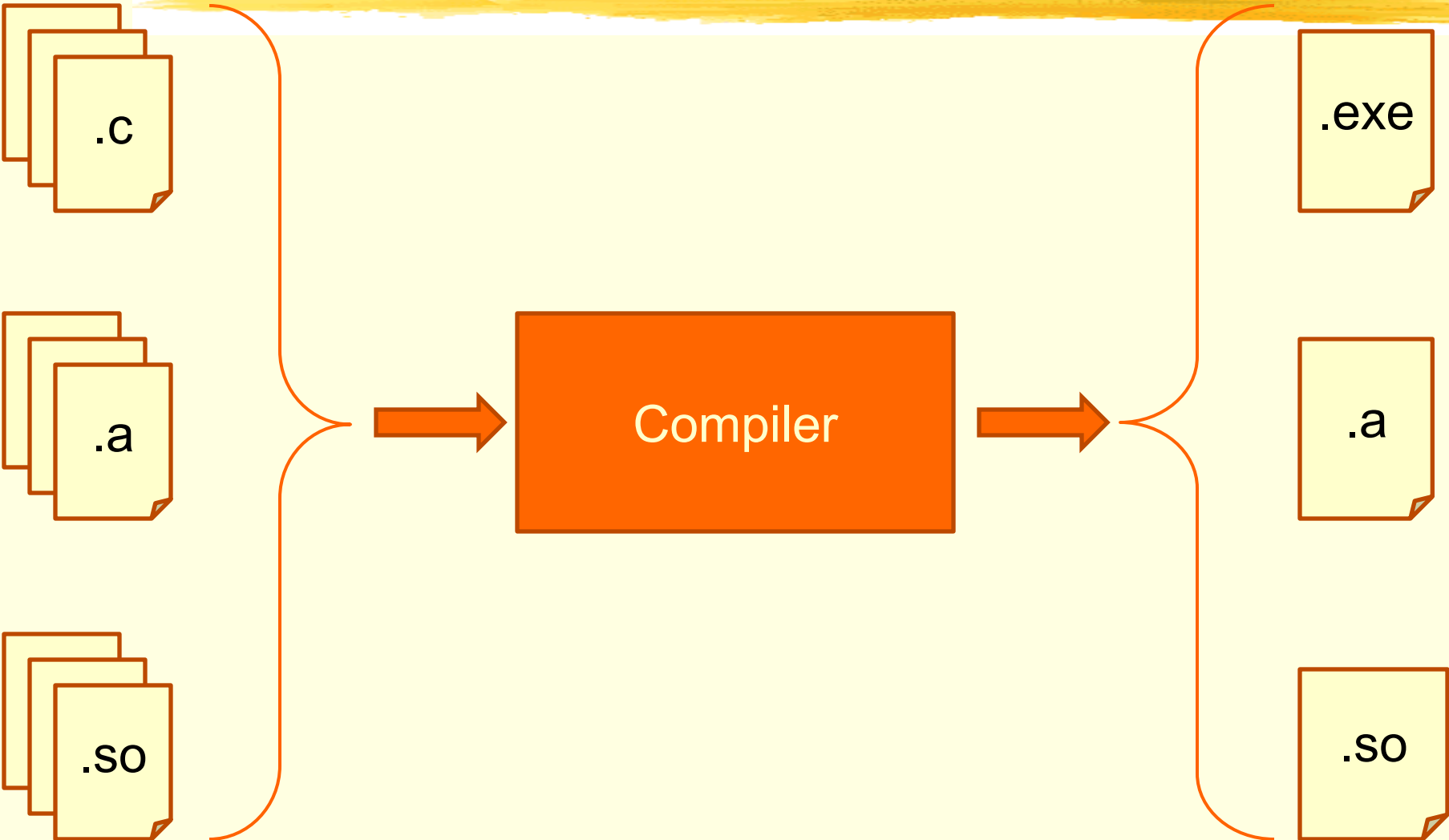
# What does a Compiler do Anyway? (Cont'd)



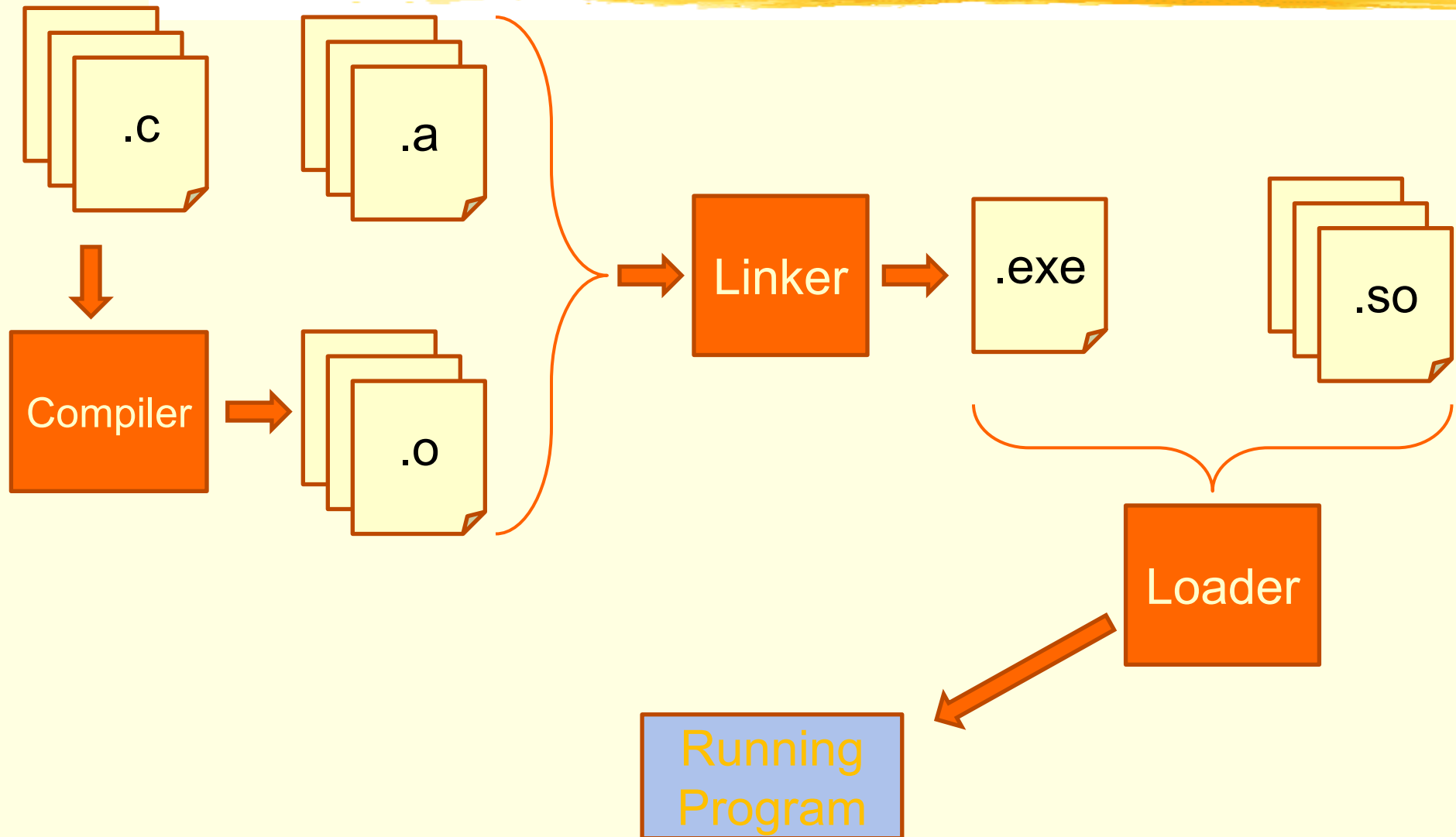
# What does a Compiler do Anyway? (Cont'd)



# What does a Compiler do Anyway? (Cont'd)



# The Whole Compilation Chain



# Inside a Compiler

## Front-End

Scanning

Parsing

Semantic Analysis

## Middle-End

Optimization 1

⋮

Optimization N

## Back-End

Select

Schedule

Allocate

# A Quick Look at the Front-End: The Lexer



- Makes sure that every single “word” in the programming language is well-formed
- Outputs tokens which describe to what category each word in a given program belongs to
- Think of it as a “spell checker”

# A Quick Look at the Front-End: The Parser



- Takes tokens as input
- Makes sure the tokens are inserted in a valid sequence
- Think of it as a “grammatical checker”

**The lexer and the parser make sure there the input program is correct with respect to the formal semantics of the language used by the programmer**

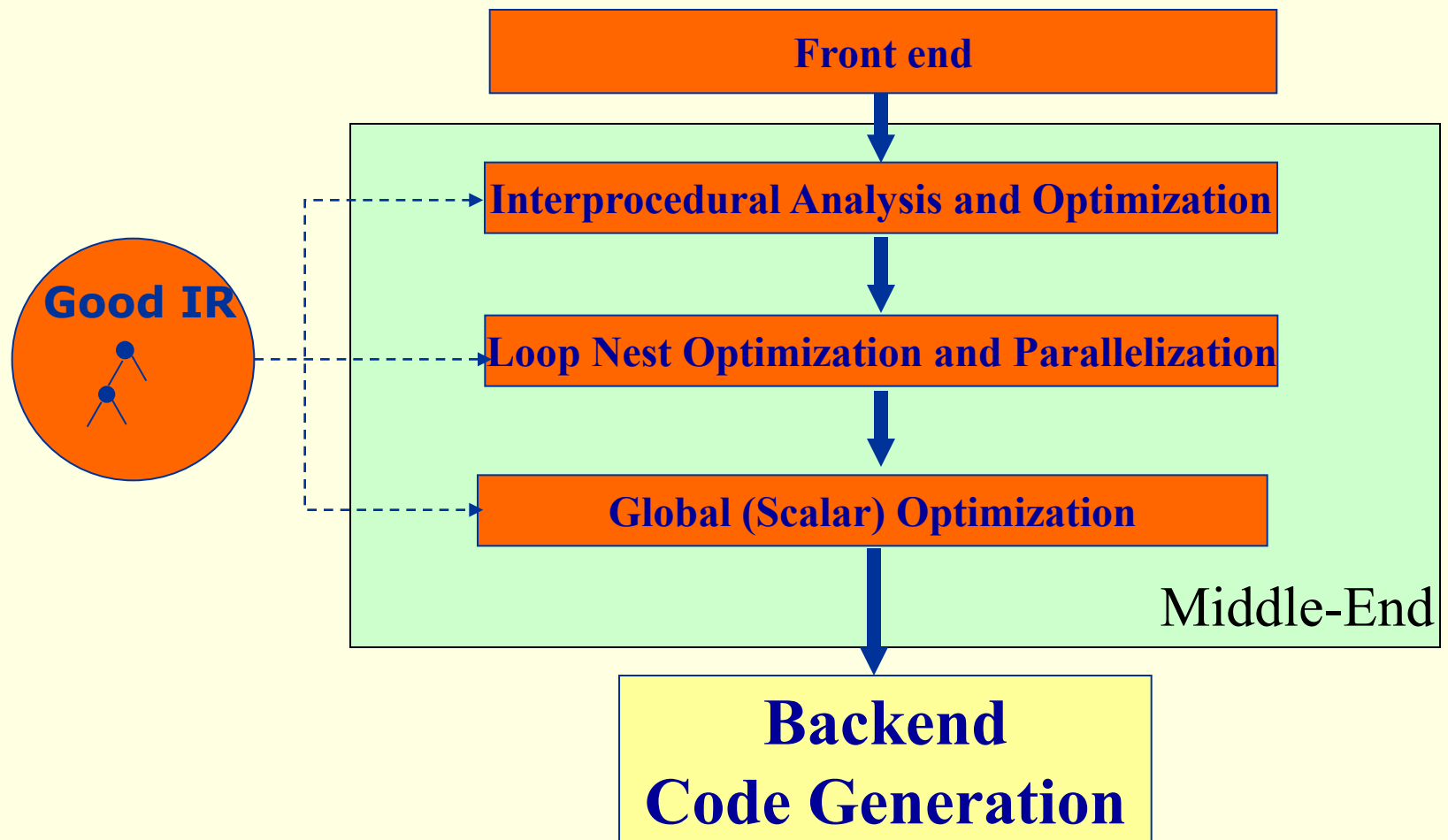
# A Quick Look at the Front-End: The Semantic Analyser



- Context-sensitive analysis (or semantic analysis) checks that the output of the lexer and parser has *meaning*.
  - E.g. “This house is very clever” vs “This student is very clever” → both are grammatically correct, only one has meaning
- It is useful at several levels:
  - Correctness can be further ensured
  - Can ensure safety through type-checking
  - Can provide the middle-end and back-end with useful information w.r.t. certain expressions



# A Good Compiler Infrastructure Needed – A modern View



# Middle-End Optimization



- Flow Analysis
  - Control flow analysis
  - Dataflow analysis
- Global scalar optimization
- Loop nest optimization
- Advanced topics:
  - Static Single Assignment form (SSA)
  - Application of SSA to scalar optimization

# Backend Optimization (I)



- Instruction selection
- Instruction scheduling
- Register allocation
- Others

# Backend Optimization (II)



- Loop optimization and scheduling
- Software pipelining