



Introduction

Professor: Suman Saha

Course Staff



Professor:
Suman Saha



TA: Srujan
Singareddy



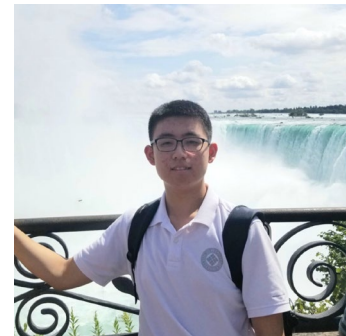
TA: Prudhvi
Gadupudi



TA: Jay Pope



TA: Vishnu Dasu



TA: Bokai
Zhang



LA: Achintya
Lakshmanan



LA: Abhiram
Dodda



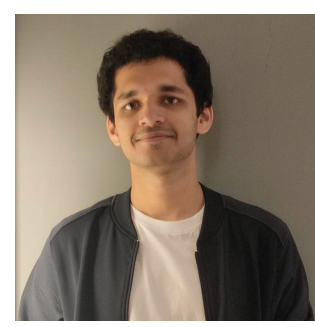
LA: Mehuli
Ghosh



LA: Yash
Parmar



LA: Shasi
Prabhu



LA: Aneesh
Shamraj

Virtual TA

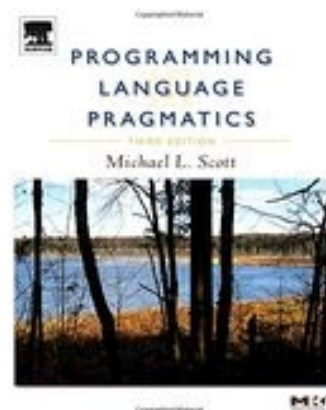


ALAASKA

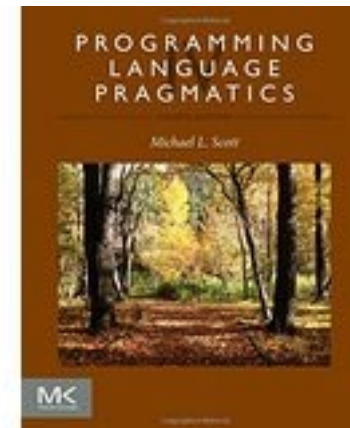
- Campuswire
 - signup: using this [link](#) (passcode: 7137)
 - please join via your .psu email ID.
 - everyone can post, ask questions about lectures/assignments
 - please search past threads before posting
 - do not share your code or your program output or any solutions and it is okay to answer your fellow students' questions
- Canvas
 - first send emails to a TA for any questions.
 - contact the instructors outside their office hours **only when** your TA cannot answer your questions **(remember we have more than 400 students!!!! Instructors will only answer emails that cannot be answered by TAs. In this case, you need to copy your TA in the email)**
- Email: Please use Canvas mailbox

Textbooks

- Required Books:
 - Programming Language Pragmatics (4th edition), Michael Scott
- Recommended Books:
 - Programming Languages Principles and Paradigms (\geq 2nd edition), Allen Tucker and Robert Noonan
 - Concepts of Programming Languages (\geq 11th edition), Robert Sebesta



3rd Edition

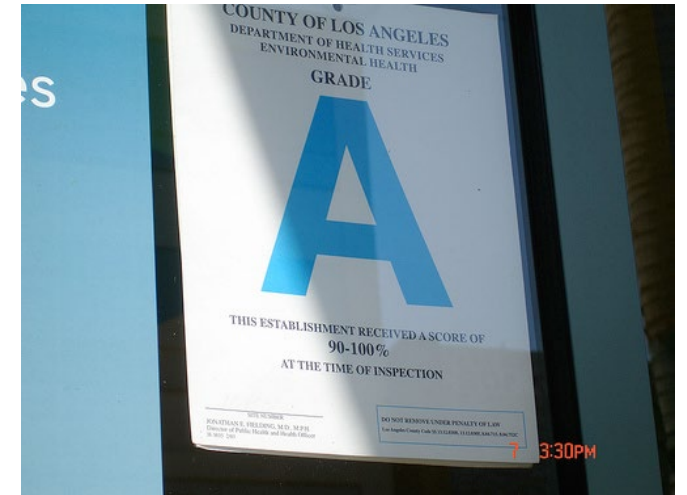


4th Edition

Grading

- The course will be graded on exams and assignments in the following proportions:

Percentage	Activity
20%	7 Assignments
10%	2 Projects
20%	Midterm-1
20%	Midterm-2
30%	Final Exam



- (Optional) : Class Participation (2%) using [Top Hat](#) will be used for curving in the final grading.
- (Optional) : Participate weekly quizzes (2%) which will be published [microlearning](#) study materials.

Projects



- Project - 1: Creating a Parser
- Project - 2: Creating a small system to find type violations

Details will be discussed later.

- For violation of AI, We will follow the [guideline of CSE department](#):
 - 0 for the submission that violates AI, **AND**
 - a reduction of one letter grade for the final course grade(students with prior AI violation will receive an F as the final course grade)

EECS Academic Integrity Quiz



- Completing this quiz is mandatory for enrollment in the course, as it requires you to acknowledge the rules and policies outlined in the syllabus. Please ensure you complete it, as it is a requirement for all students.
- Note that access to other course materials will only be granted once you achieve a 100% score on this quiz.


Honors Option



- Schreyer scholars are welcome to take the honors option of this course
 - Extra reading materials
 - More challenging assignments
- Send me emails to sign up for the honors option
- Non-honors student can do the project by enrolling CMPSC 496

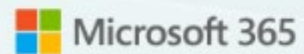
Anonymous Feedback

Anonymous Feedback Form

1. Please provide your feedback to help enhance teaching and learning experiences. 

Enter your answer

Submit



This content is created by the owner of the form. The data you submit will be sent to the form owner. Microsoft is not responsible for the privacy or security practices of its customers, including those of this form owner. Never give out your password.

Questions

?

CMPSC 461 IS NOT



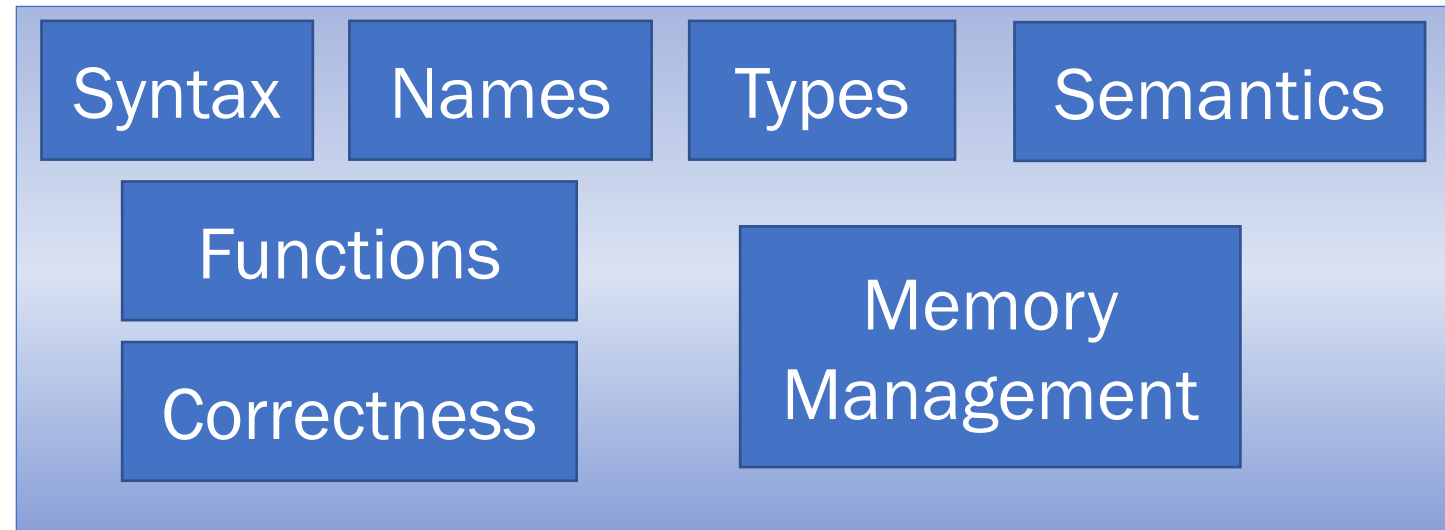
- C/C++/Java/Python/Scheme/... programming
- Compiler construction (471)
- Object-oriented programming (221)
- Data structures

- References: [Why CMPSC 461?](#)

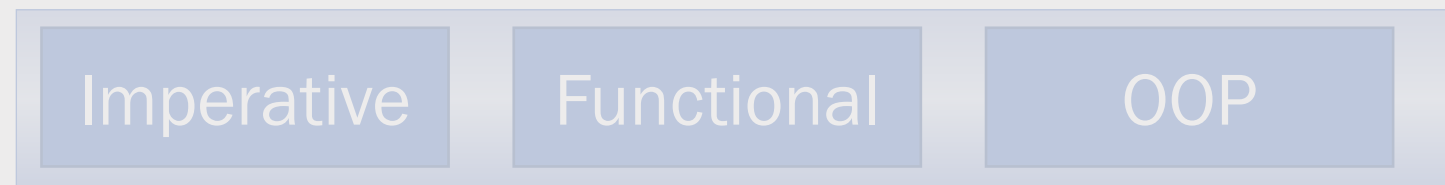
- Explores *fundamental principles and paradigms* of programming languages.
- Studies features found in many different languages and examine how they work and how they interact with each other.
- Programming languages is a powerful tool once you master the principles
- Example: use **type systems** to build secure software

Course Coverage

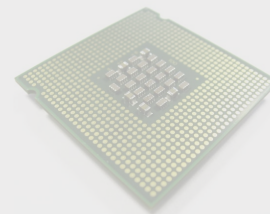
Programming Principals



Programming Paradigms



Language Implementation



Language Definition

- Language = Syntax + Semantics + Design Philosophy
- Syntax: specifies what valid programs are

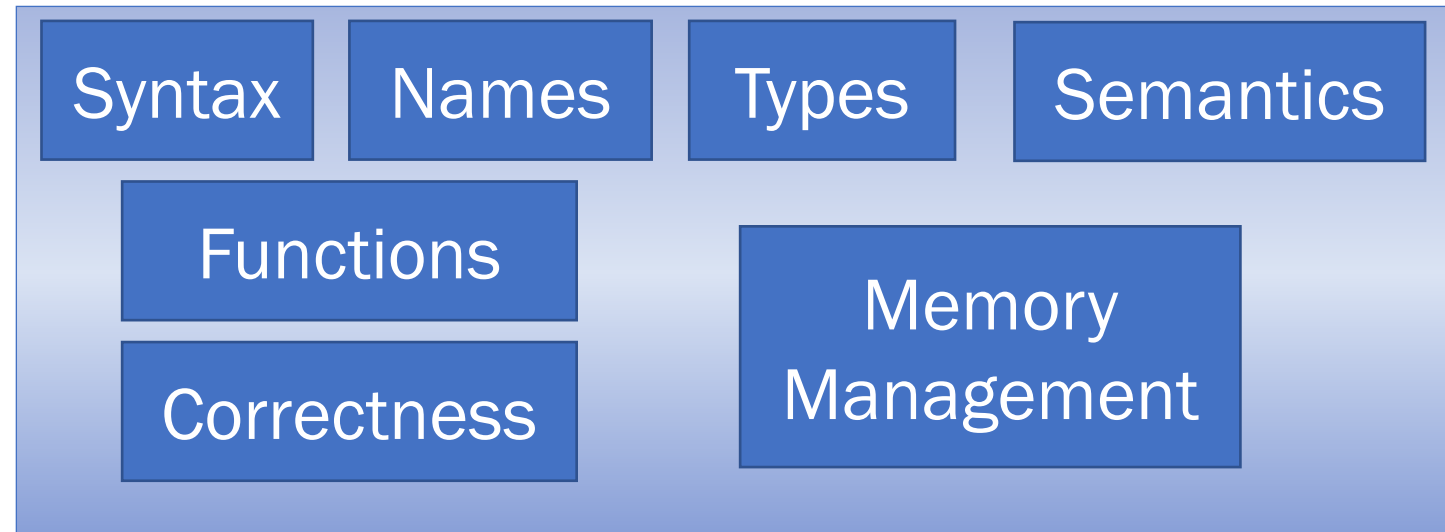
```
class MyFirstJavaProg {  
    public static void main(String args[]) {  
        int x = 3 + 4;  
        System.out.println("x= " + x);  
    }  
}
```

- If we write + 3 4, then that's not a valid Java program, or not syntactically correct

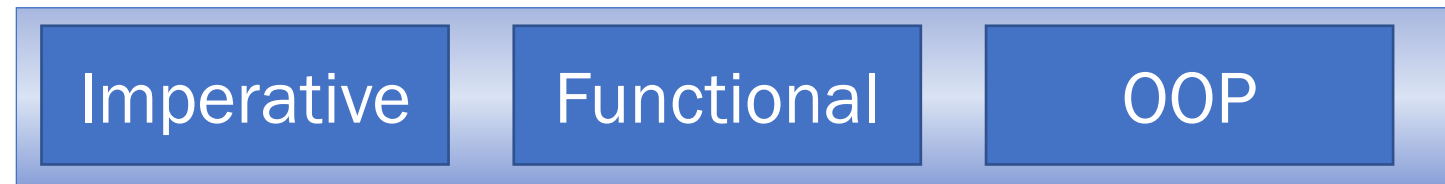
- Semantics: dictates what a program does
 - The meaning of a program
 - Informal description: English description, by examples
 - E.g., the “Java language spec” book

Course Coverage

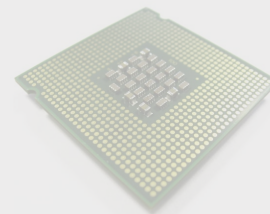
Programming
Principals



Programming
Paradigms



Language
Implementation



Language Design Philosophy



- A **programming paradigm** is a style of programming
 - A single computing task can be accomplished in many ways
 - Different philosophies of how programs should accomplish a task leads to many programming paradigms

Major Programming Paradigms



- Imperative programming
 - Computation as a sequence of commands that change a program's state
 - Example languages: C, Pascal
- Object-oriented programming (OOP)
 - Computation as objects and their interaction
 - interaction: message-passing between objects for changing their states
 - Example languages: Java, C++, Smalltalk

Major Programming Paradigms



- Functional Programming (FP)
 - Computation as mathematical functions: input and output
 - Pure FP: no notion of states and always returns same output for the same input
 - Example languages: Lisp, ML, Haskell, Scheme
- Logic Programming
 - Computation using mathematical logical rules
 - Rule-based programming
 - Example language: Prolog

More Programming Paradigms

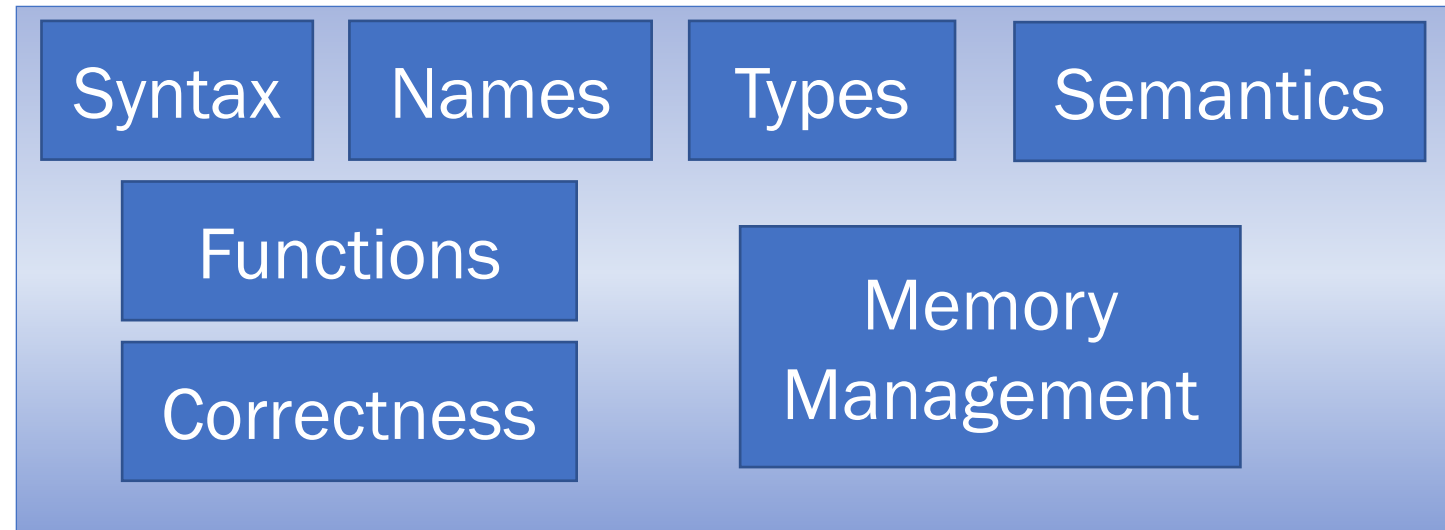


- Structure Query Language (SQL)
- Dataflow languages
- Scripting languages
- ...

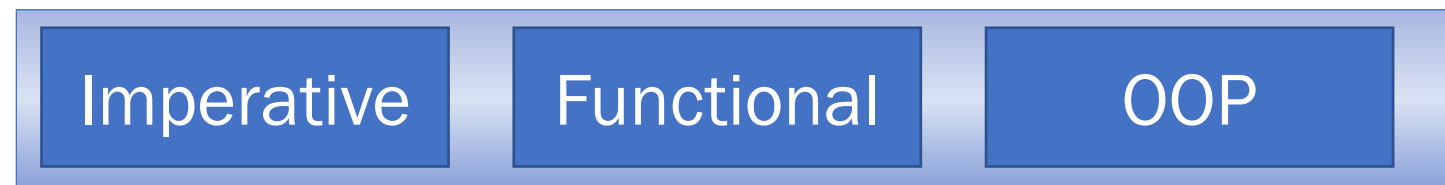
- A language usually uses a mix of those paradigms
 - C++: mix of imperative and OO programming
 - Scala: OO and functional programming

Course Coverage

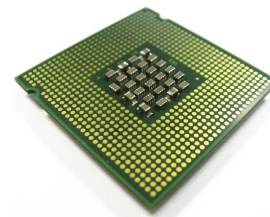
Programming
Principals



Programming
Paradigms



Language
Implementation

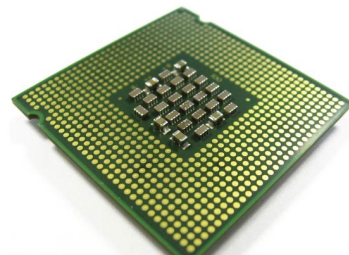
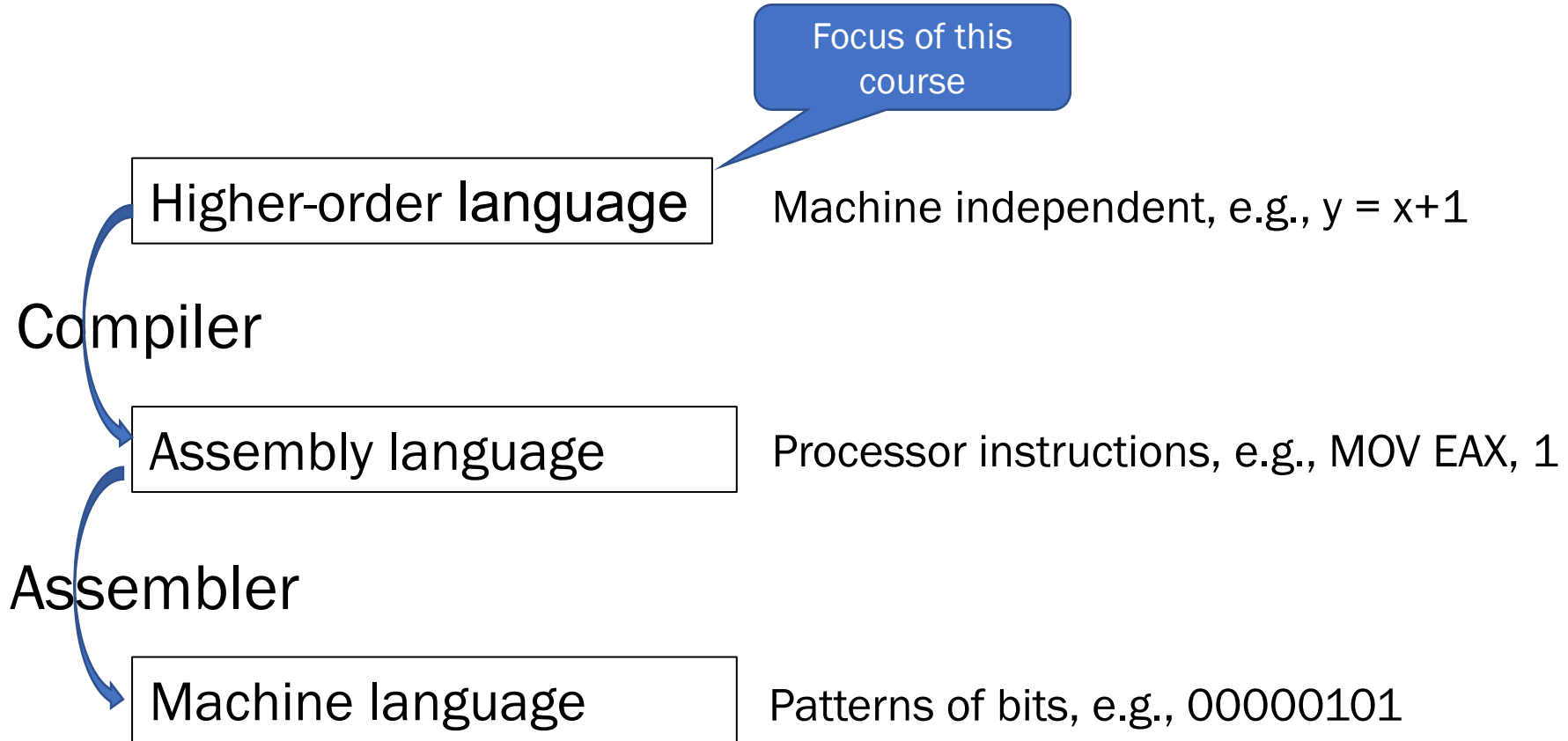


Interpreter vs. Compiler

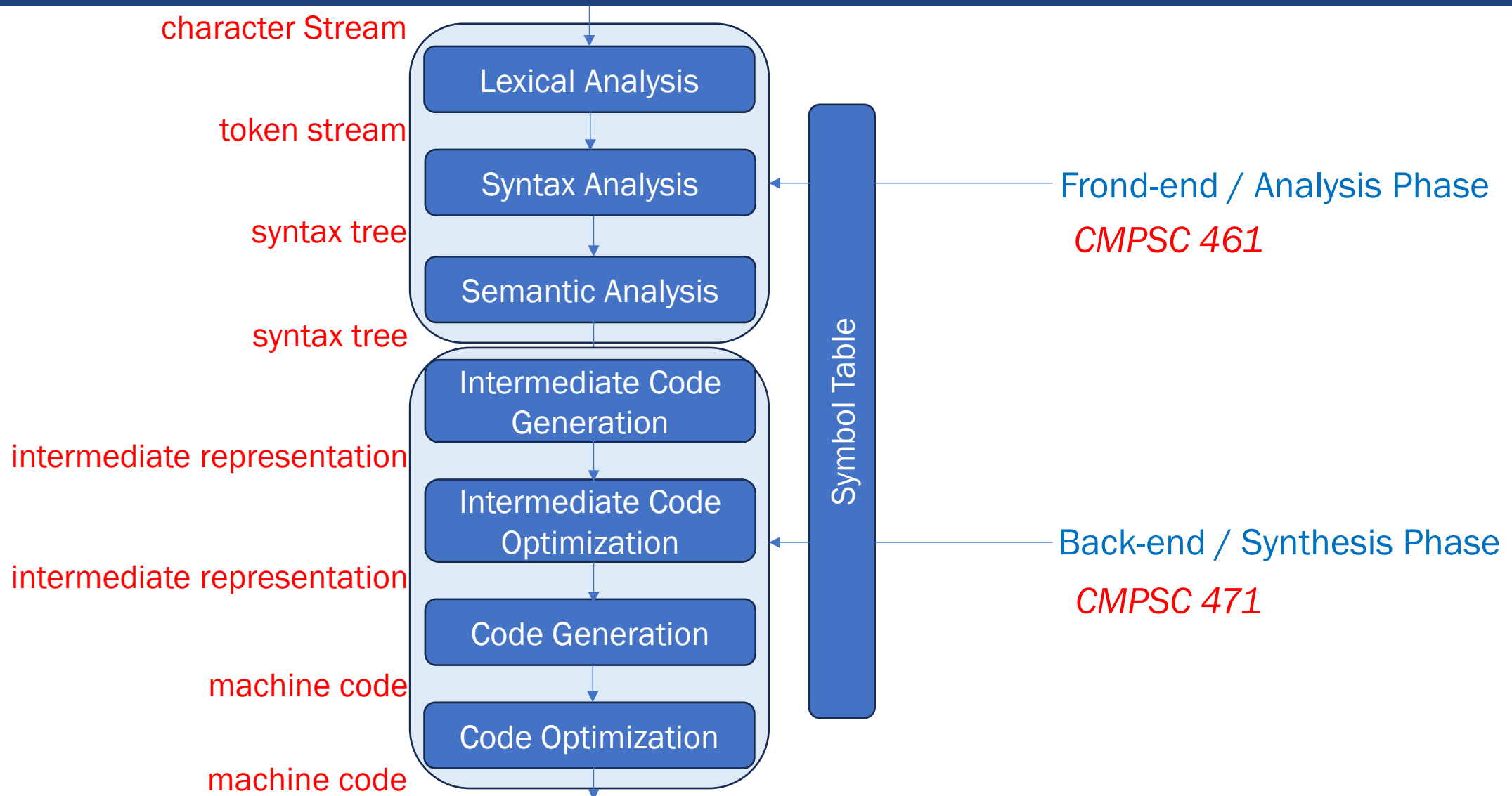
- Interpreter: easy to implement, but slow
 - Mix the translation and execution; translation performed multiple times on the same function if it is executed multiple times
 - Can be 10 times slower than the compiler
- Compiler: harder to implement, but more efficient
- Many language starts with an interpreter, then a compiler

- Java: a mixed mode
 - Java compiler produces instructions for an architecture-independent machine (Java bytecode)
 - JVM interprets these instructions to machine code

Levels



Structure of a Compiler



Language Design Goals



- Readability
 - Most of the time is spent on software debugging and maintenance
 - Bad readability
 - `i++` vs. `++i` in C
 - “`i++`” post increment (use `i` first, then increment)
 - “`++i`” is pre increment (first increment, then use its value)
- Simplicity
 - A list of orthogonal features (abstractions)
 - Bad example: Perl (motto: there is more than one way to do it)
- Efficiency
 - Pitfall: don't take today's implementation to fault a language (e.g., recursive functions)
- Safety and reliability
 - C vs. Java

Positive impact of Programming Language Concepts

- Knowledge of this field could be applied to do
 - Designing new programming language
 - Constructing a compiler for any programming language
 - Building tools to verify and validate any software
 - Finding security holes and vulnerabilities in the system source code
 - Performance analysis of a system

