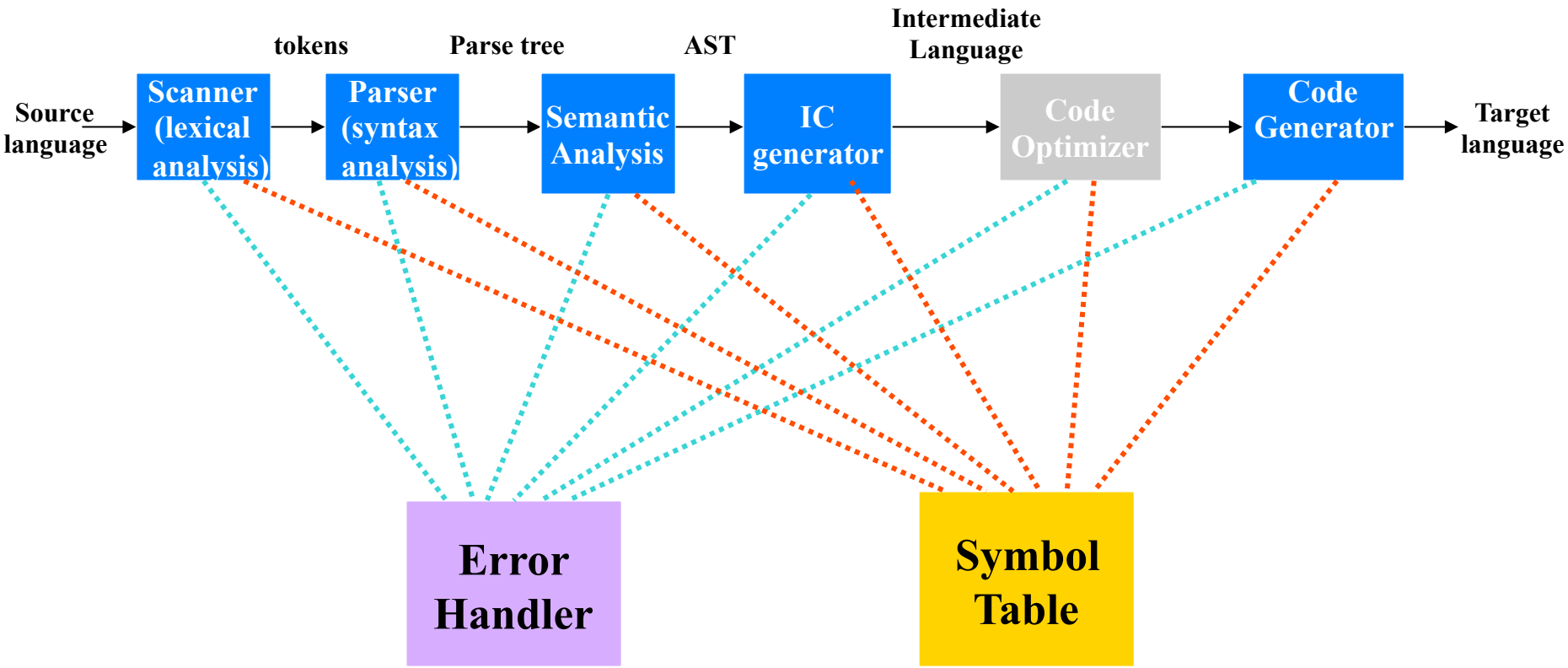
Language Processing Systems

Prof. Mohamed Hamada

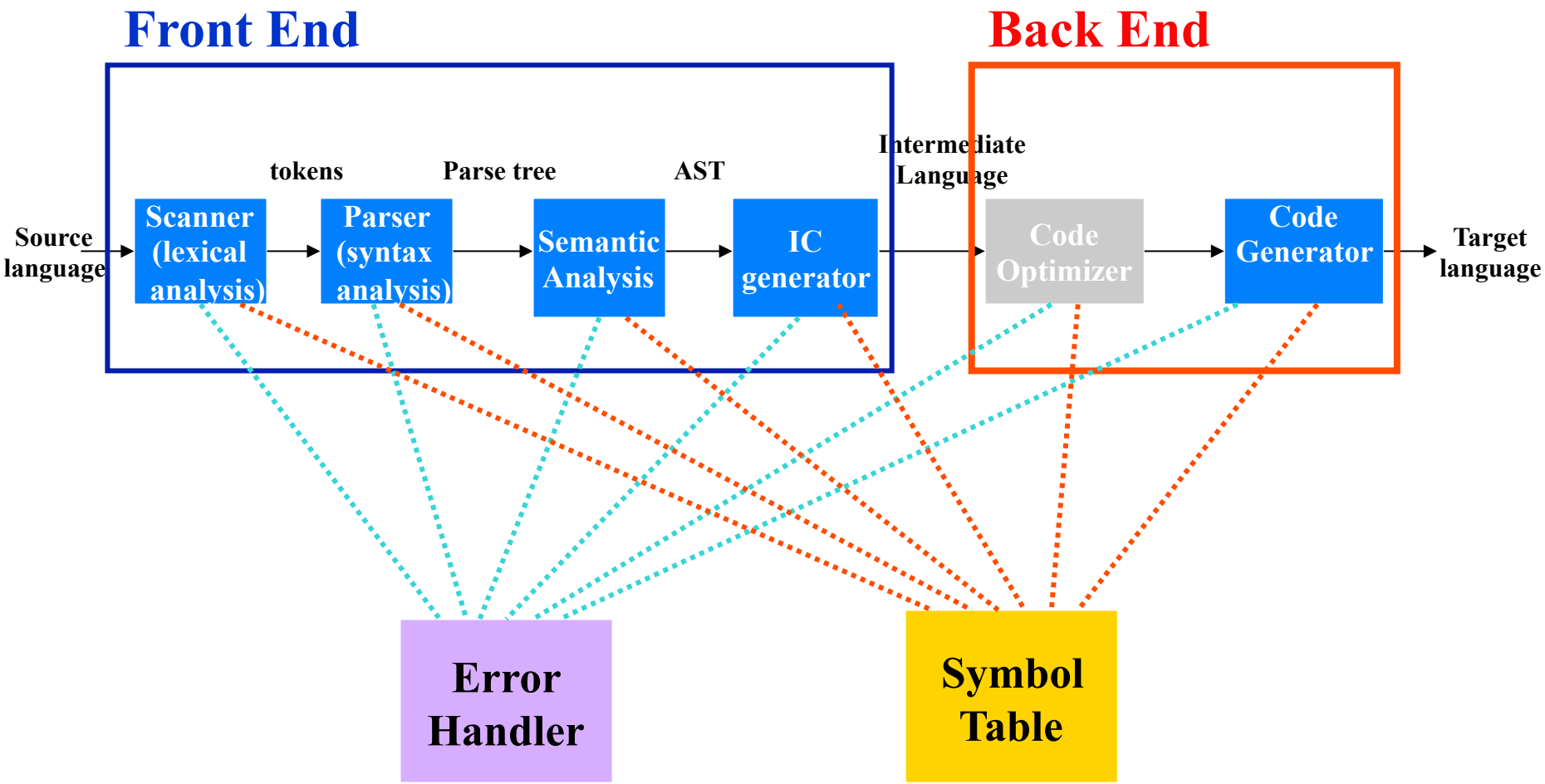
Software Engineering Lab.
The University of Aizu
Japan

Review

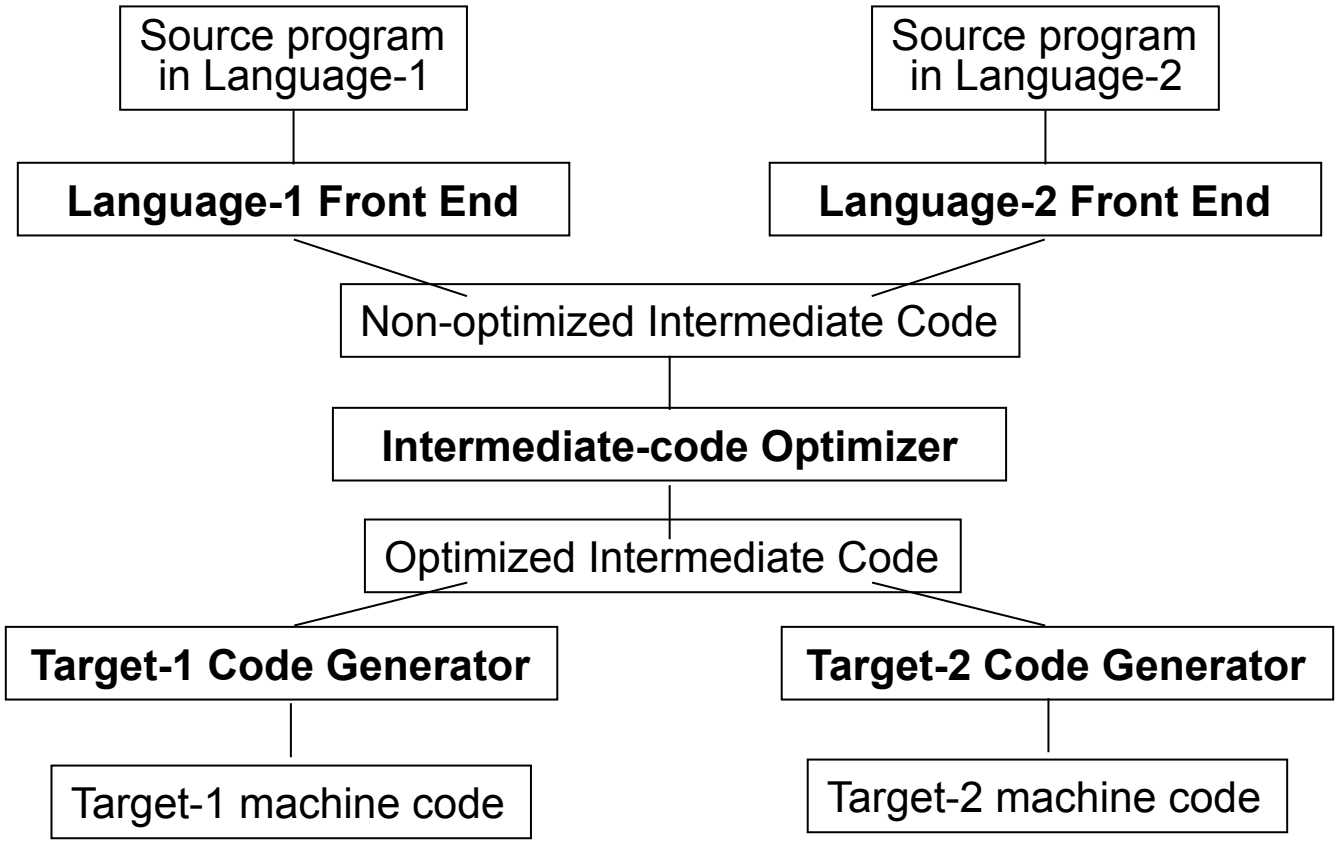
Compiler Architecture



Compiler Architecture

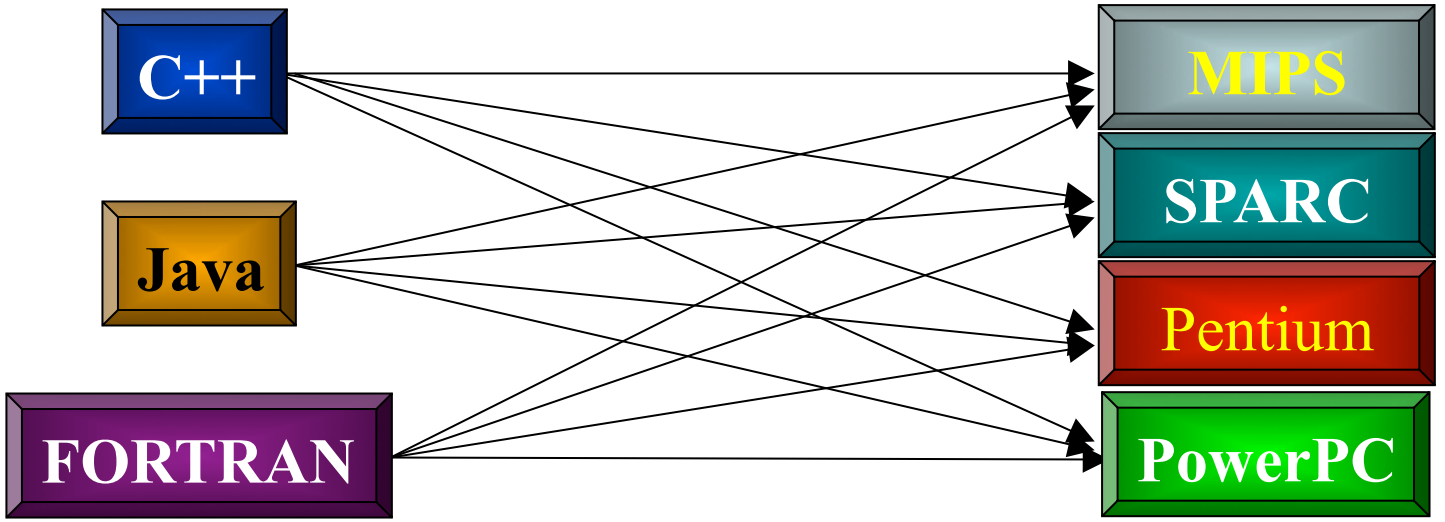


Front-end and Back-end



Front-end and Back-end

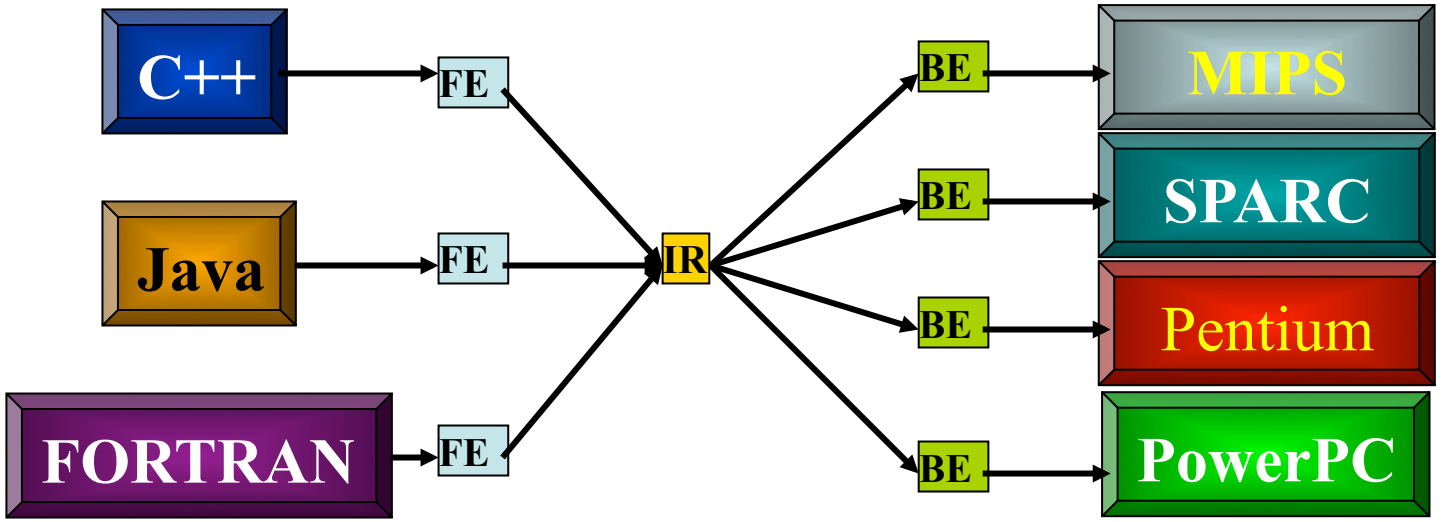
- Suppose you want to write compilers from `C++` to `4` computer platforms:



We need to write 12 programs

Front-end and Back-end

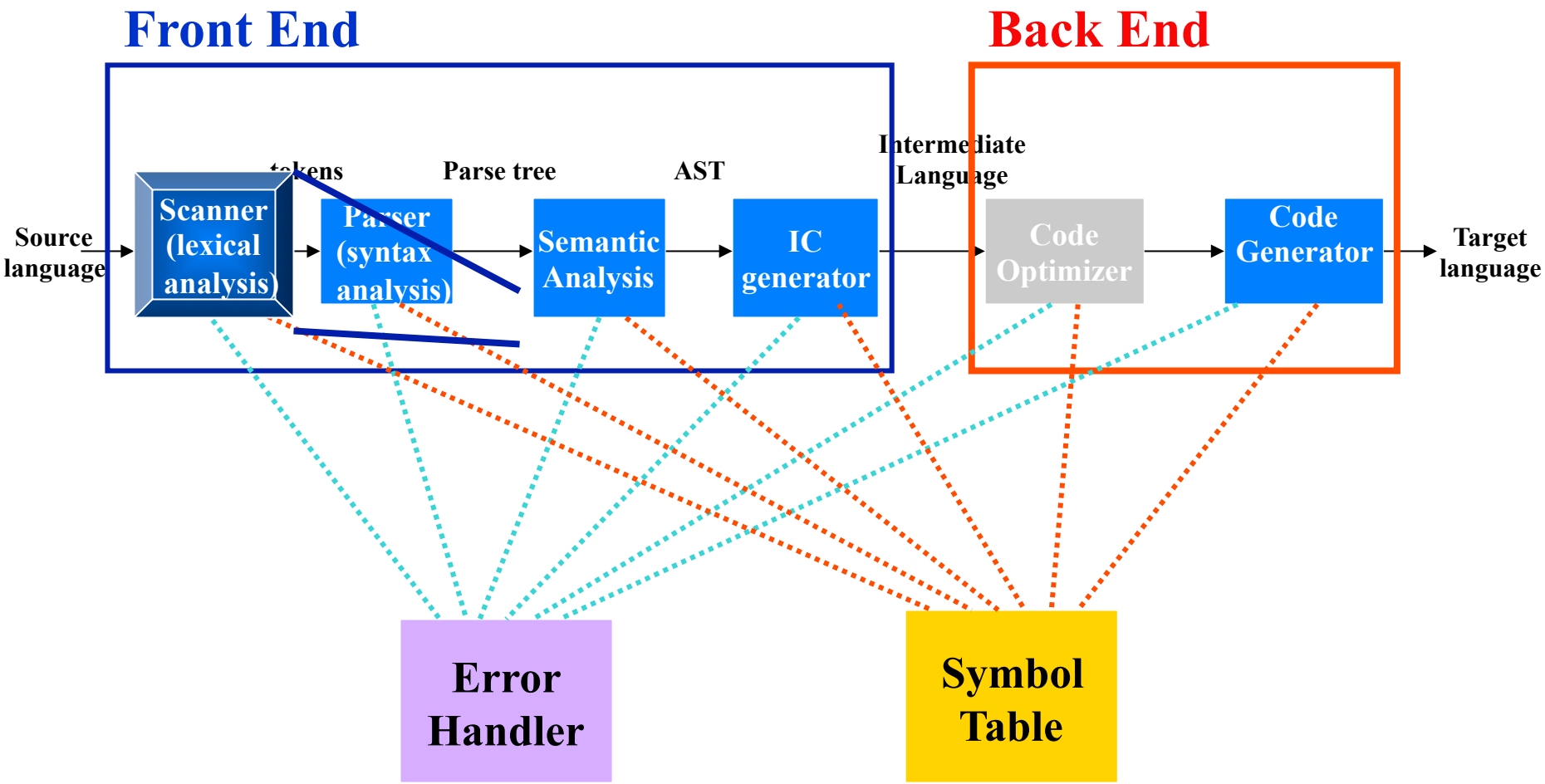
- But we can do it better



We need to write 7 programs only

- **IR: Intermediate Representation**
- **FE: Front-End**
- **BE: Back-End**

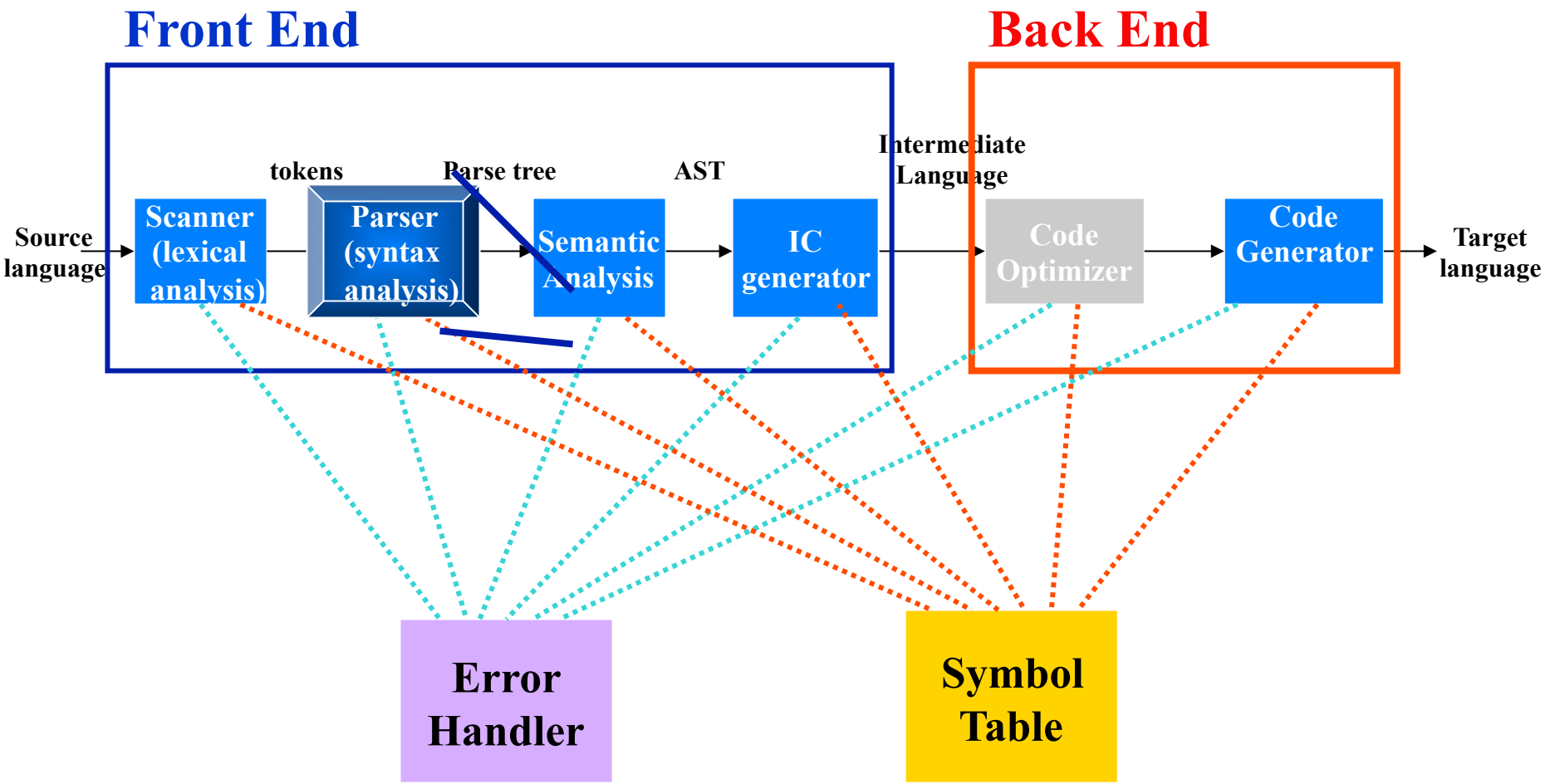
Scanner



How it works? → Use Regular expressions to define tokens
Use Finite Automata to recognize tokens

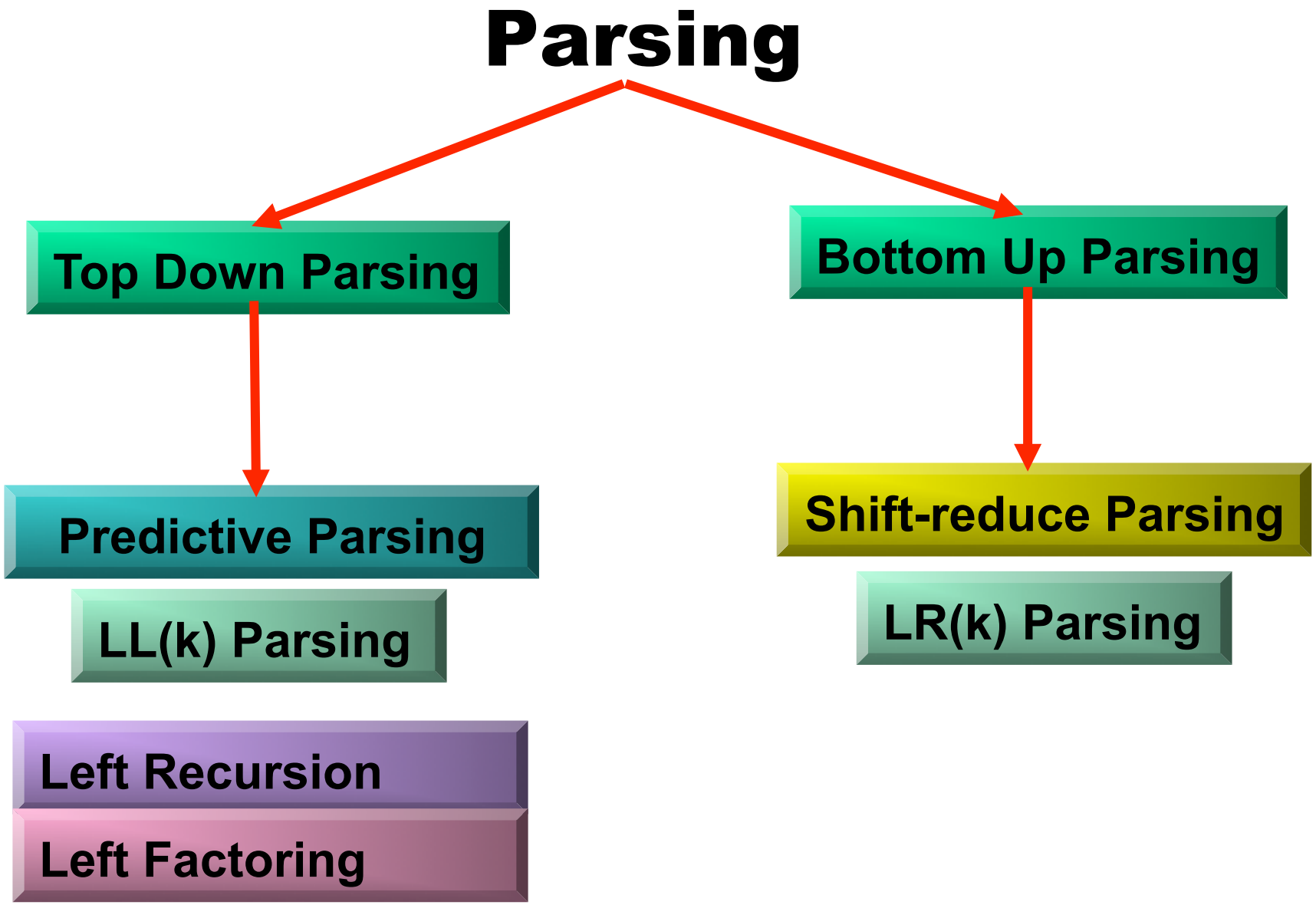
How to write it? → Use Unix command LEX

Parser

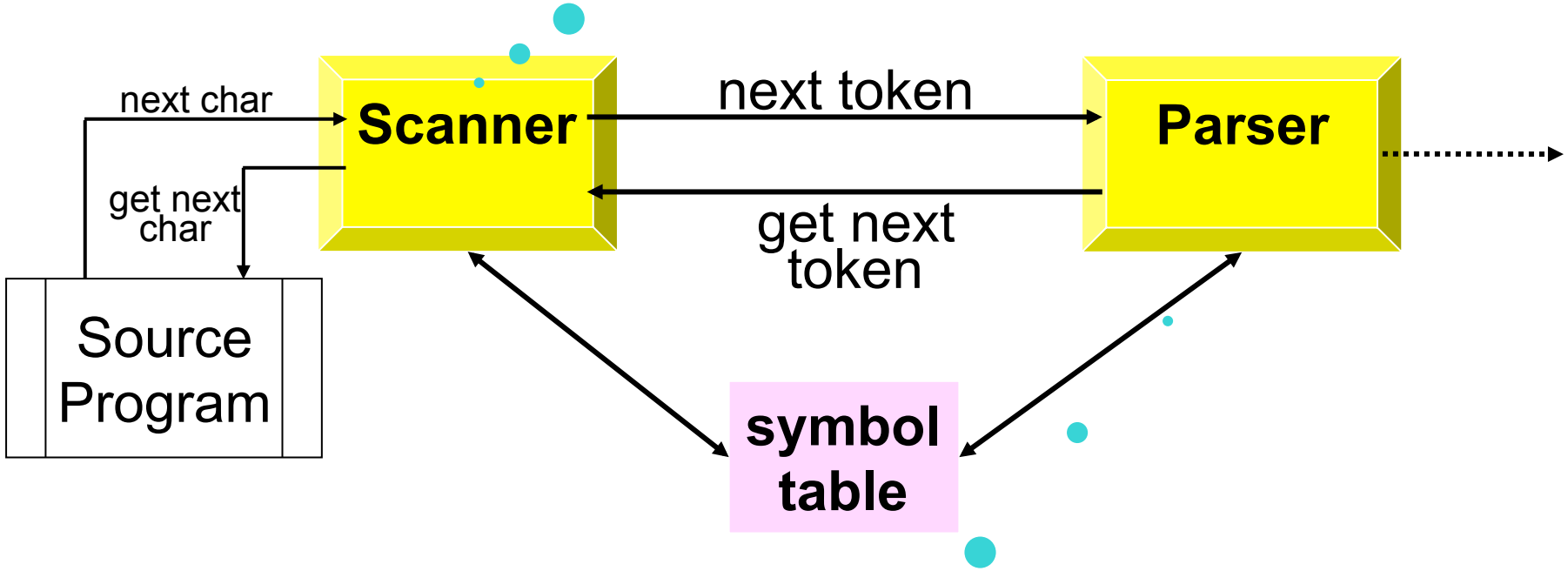


How it works? → Use Top-down (LL(k)) or Bottom-up (LR(k)) parsing to make the parse tree

How to write it? → Use Unix command Yacc


Parsing
Top Down Parsing
Bottom Up Parsing
Predictive Parsing
Shift-reduce Parsing
LL(k) Parsing
LR(k) Parsing
Left Recursion
Left Factoring

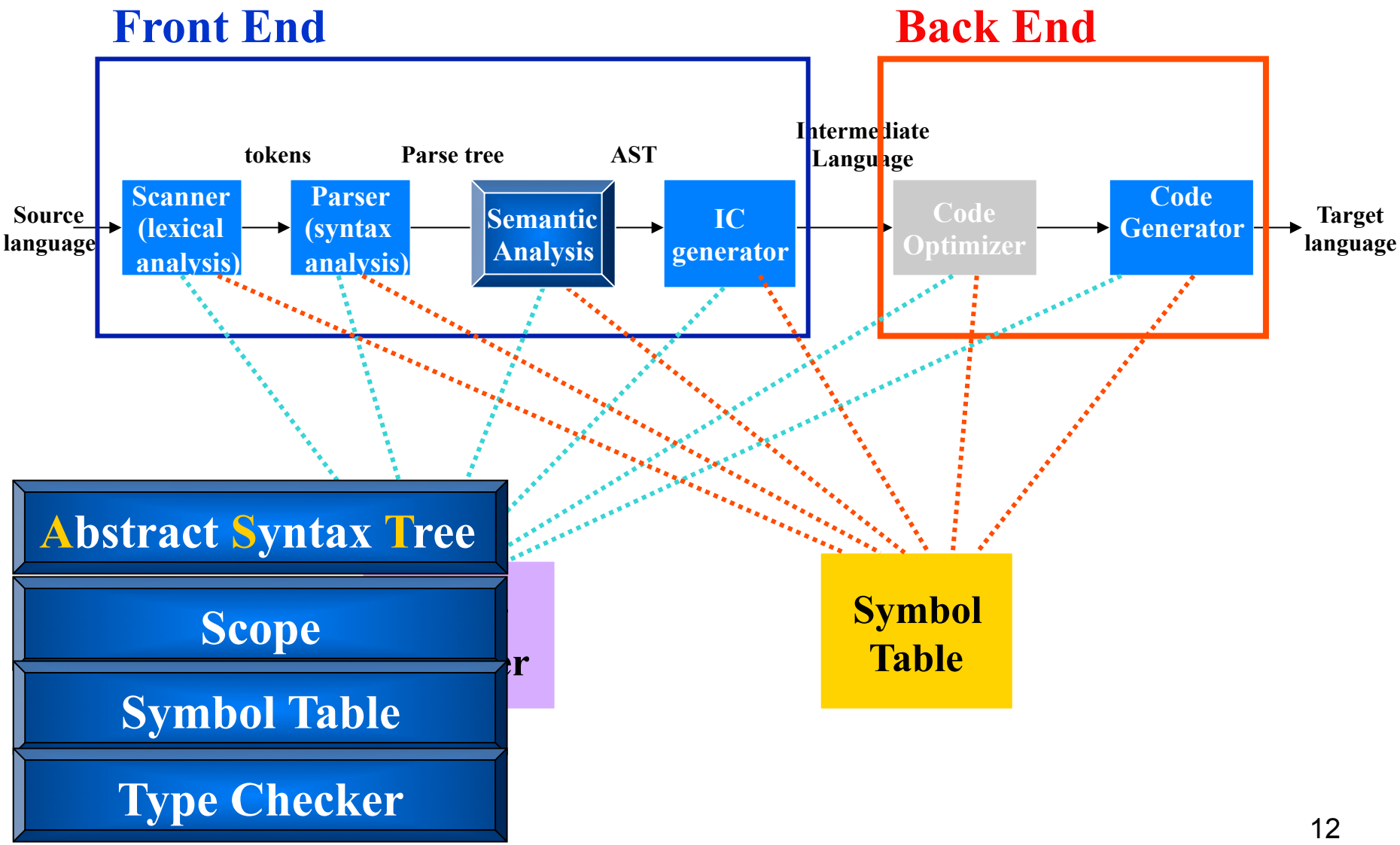
1. Uses Regular Expressions to define tokens
2. Uses Finite Automata to recognize tokens



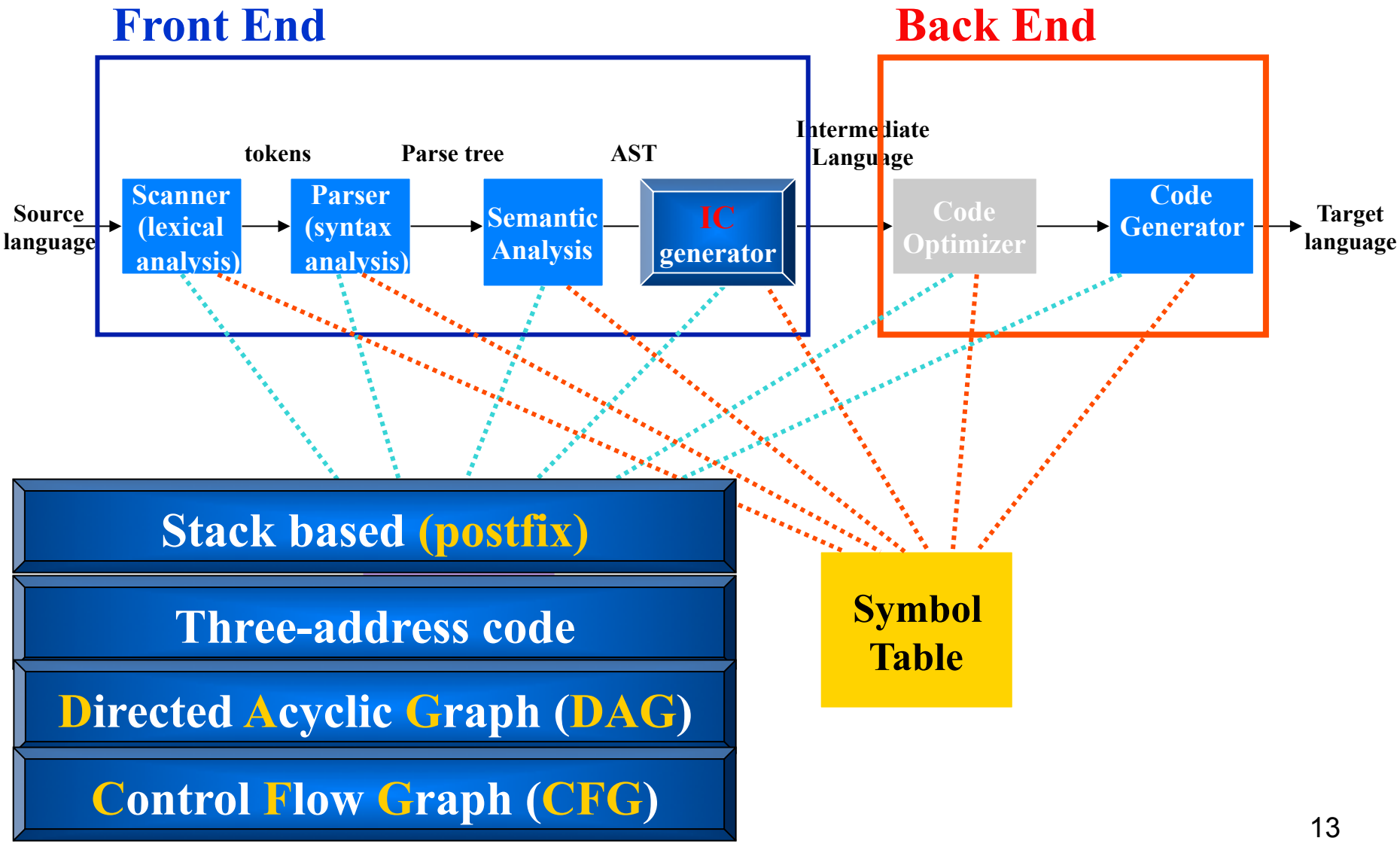
Uses Top-down parsing or Bottom-up parsing

To construct a Parse tree

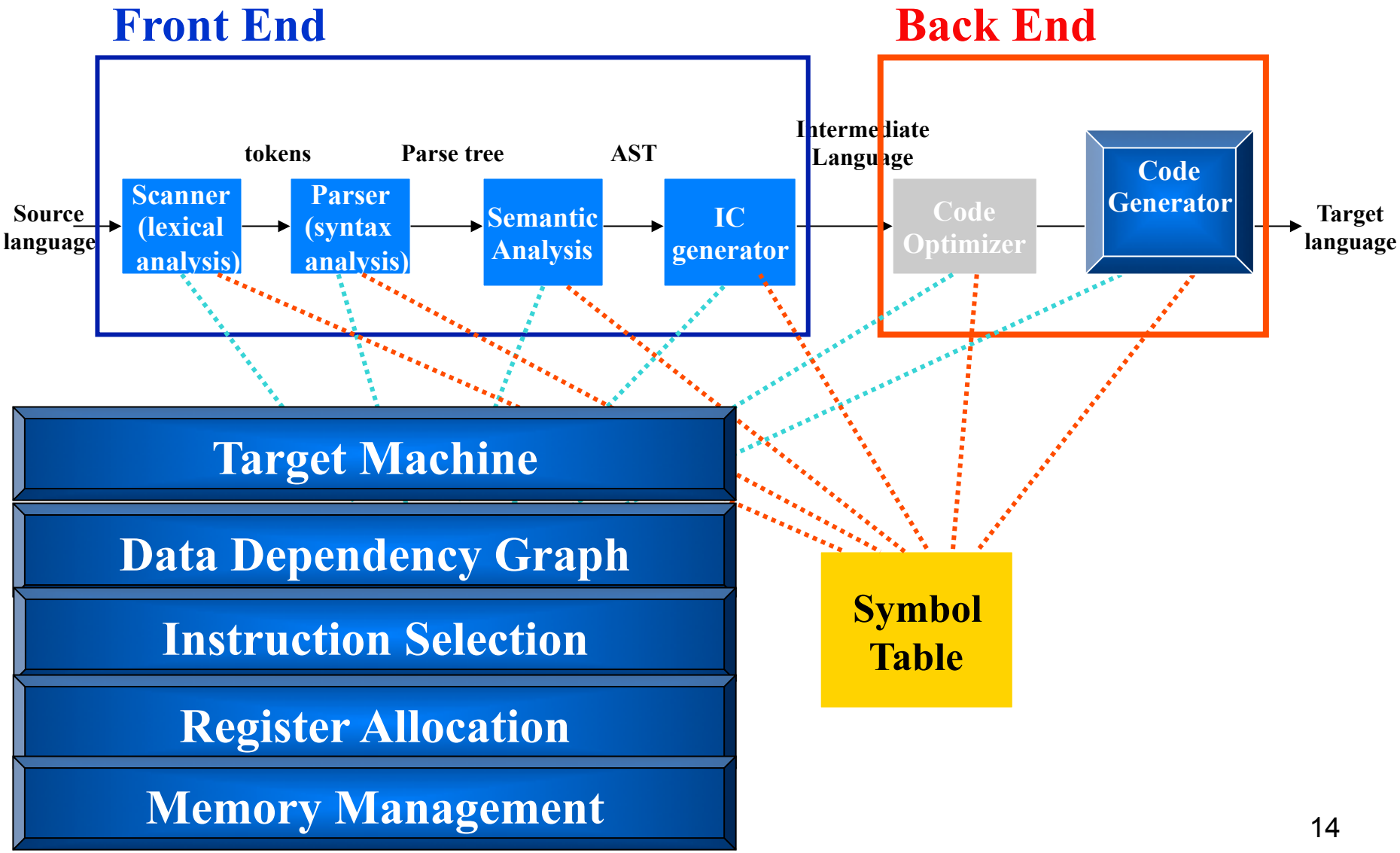
Semantics analysis



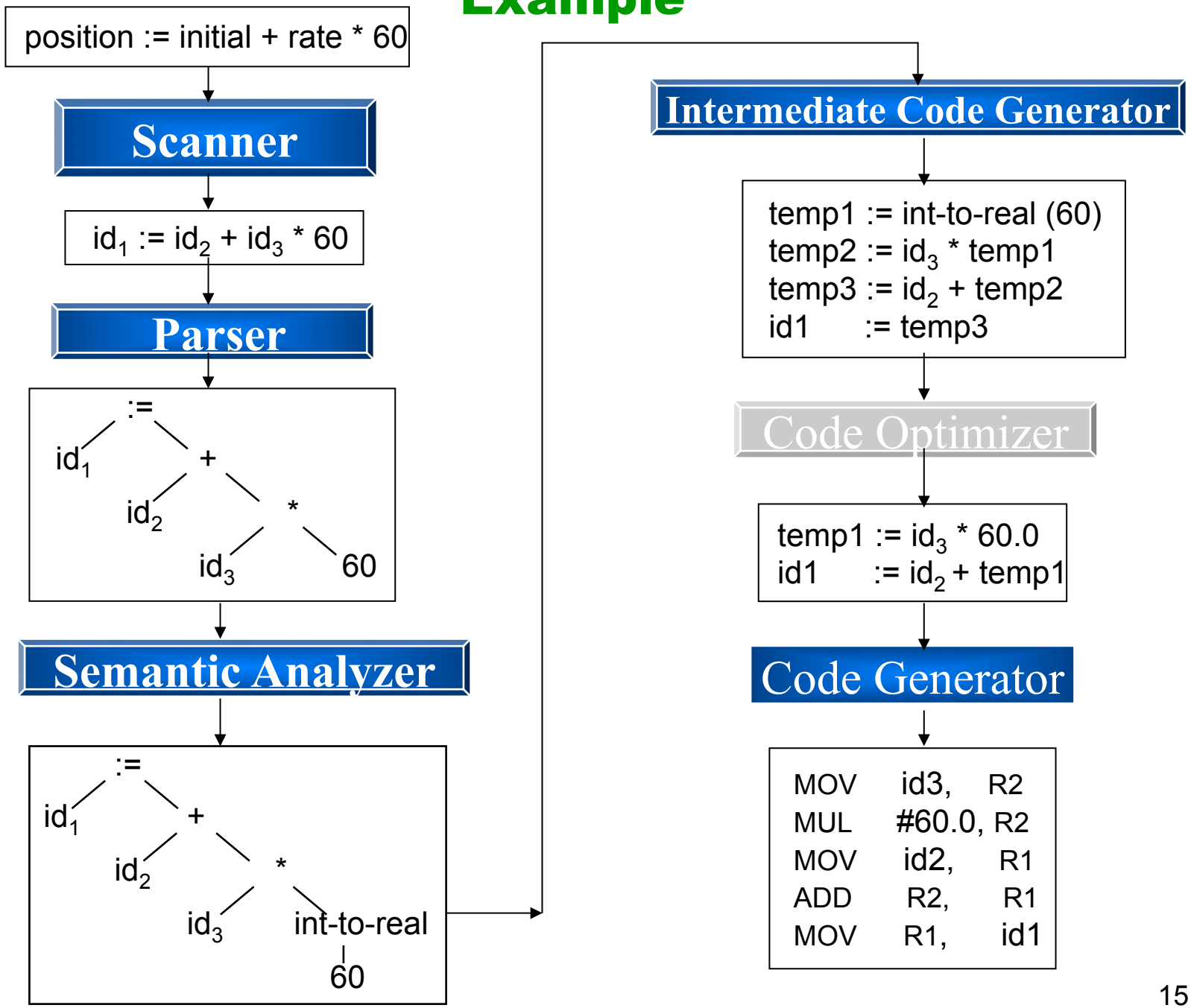
Intermediate Code (IC) Generator



Code Generator



Example


position := initial + rate * 60
Scanner
id1 := id2 + id3 * 60
Parser
:=
id1
+
id2
*
id3
60
Semantic Analyzer
:=
id1
+
id2
*
id3
int-to-real
60
Intermediate Code Generator
temp1 := int-to-real (60)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1 := temp3
Code Optimizer
temp1 := id3 * 60.0
id1 := id2 + temp1
Code Generator
MOV id3, R2
MUL #60.0, R2
MOV id2, R1
ADD R2, R1
MOV R1, id1