Lexical Analysis

Compiler Design

Lexical Analysis

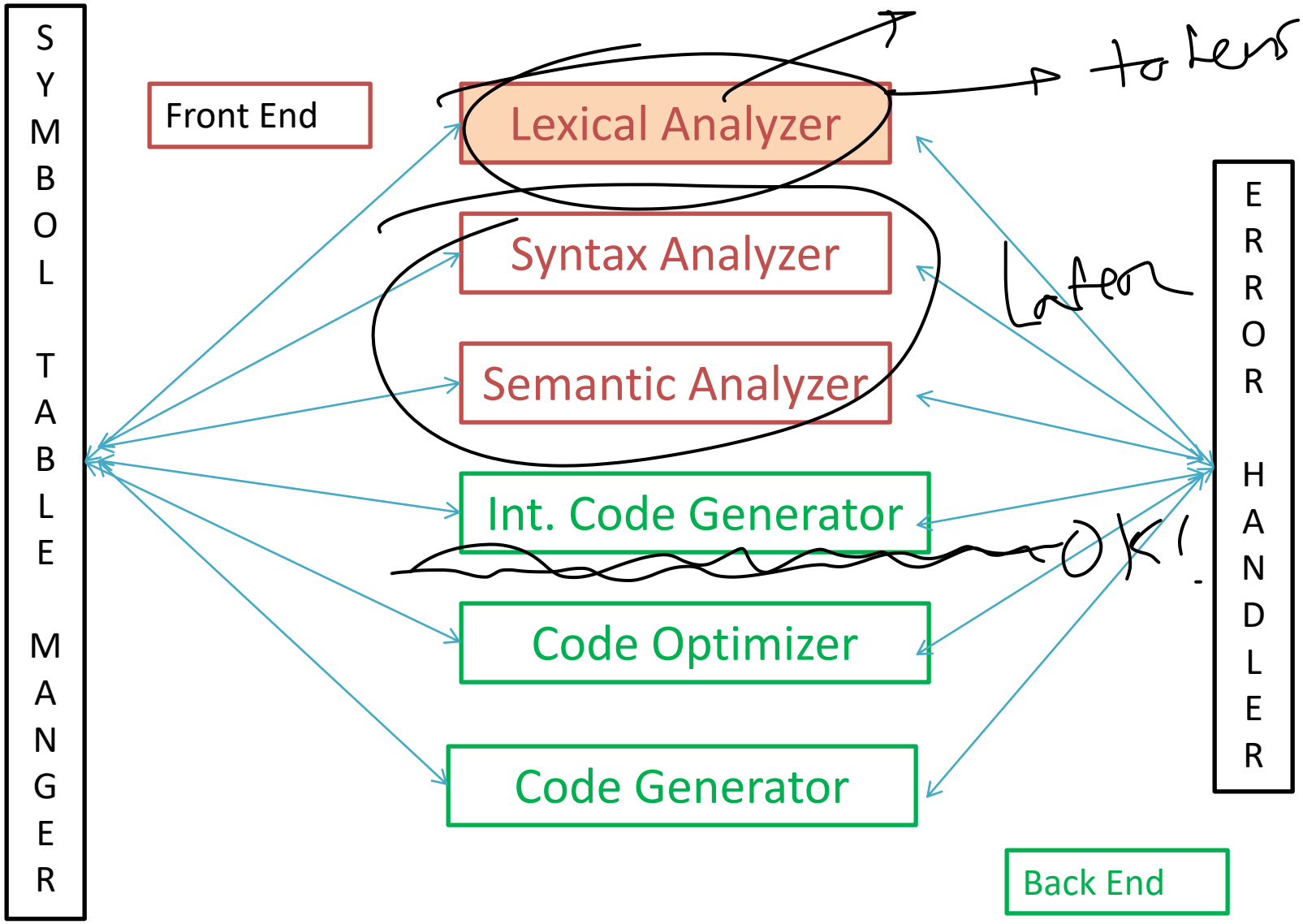
Lexical analysis is a method to convert high level input program into a sequence of Tokens.

The process or program which performs Lexical analysis is known as **Lexical analyzer** or **Scanner**.

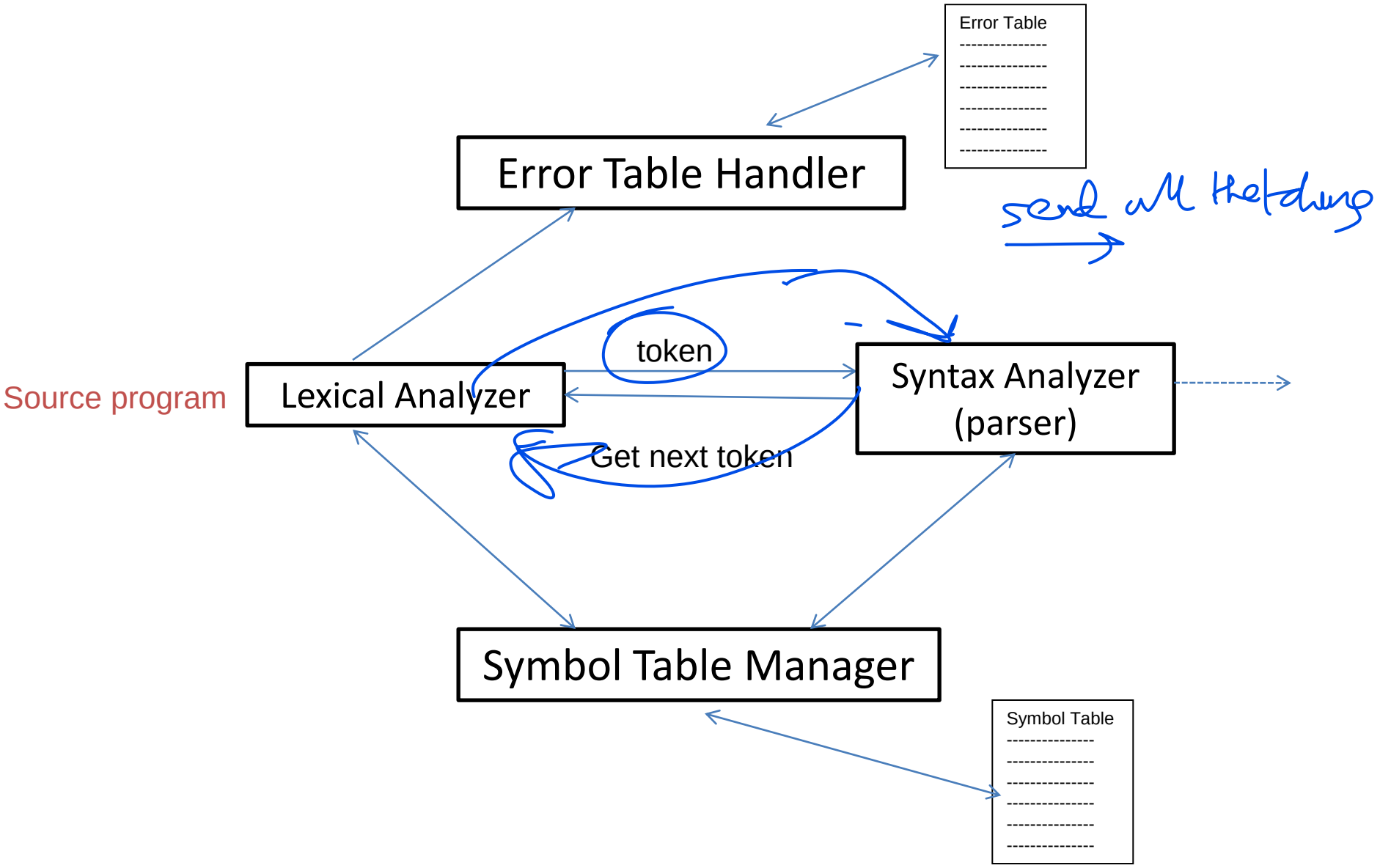
Example:

- *Lex* in Linux platform
- *Flex* in Windows platform
- You can write your own Lexical analyzer

Structure of a Compiler



Role of a Lexical Analyzer



Objective

1. Simplicity
2. Efficiency
3. Portability

Lexical Errors

Types of Errors-

- Lexical: misspelling of an identifier, keyword or operator.
- Syntactic: arithmetic expression with unbalanced parenthesis
- Semantic: operator applied to an incompatible operand
- Logical: infinite recursive call

Error Recovery Strategies

- Panic mode error recovery
- Deleting an extraneous character
- Inserting a missing character
- Replacing an incorrect character by another
- Transposing two adjacent character

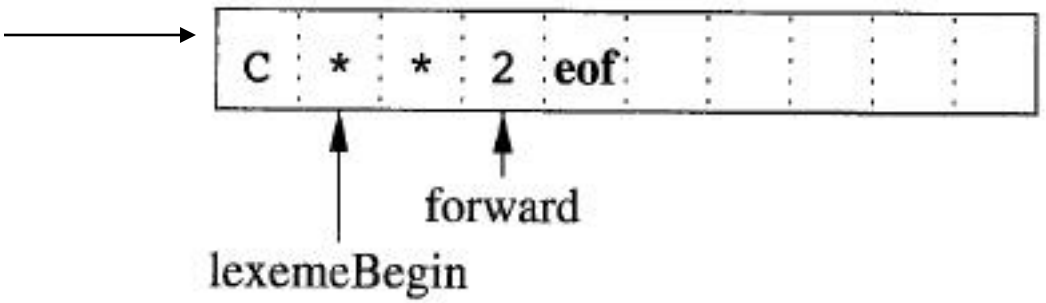
Input Buffering

Instead of reading single character from a file a block of data is retrieved into a variable by system call.

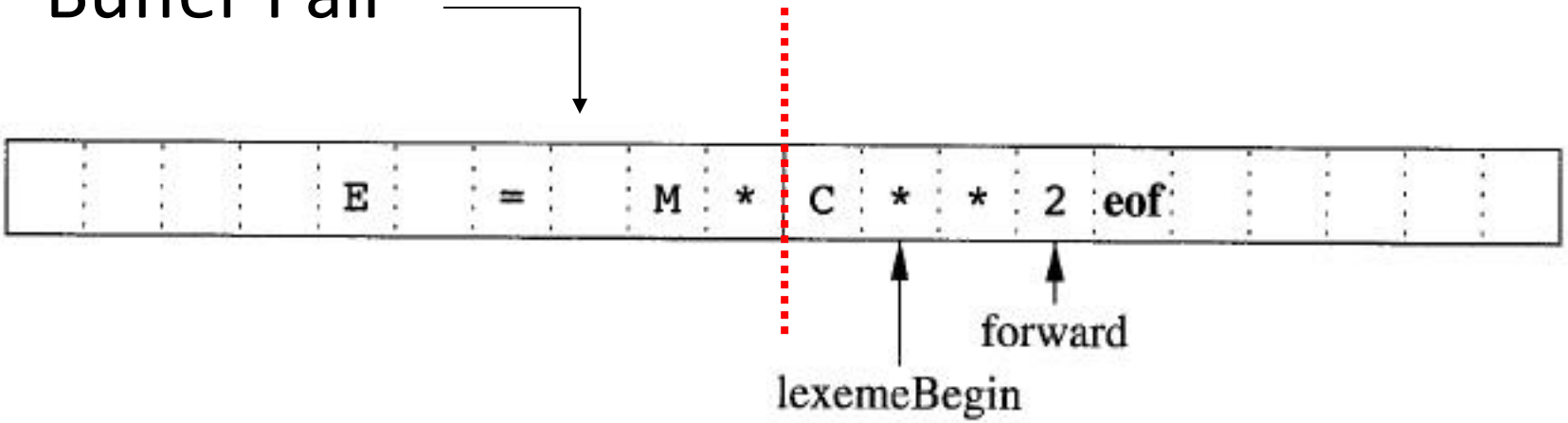
Usually Buffer size is taken same as disk block size.

Input Buffering types

- Single Buffer



- Buffer Pair



Tokens, Patterns, Lexemes

Tokens	Sample Lexemes
const	**const**
if	**if**
relation	**<, <=, =, !=, >=**
id	**pi, student_name**
num	**3.14156, 084499**
literal	**“Abdur Rahman”**

Parts of string

- Prefix
- Suffix
- Substring
- Proper prefix, suffix, and substring
- Subsequence

Operation on Languages

OPERATION	DEFINITION AND NOTATION
Union of L and M	$L \cup M = \{s \mid s \text{ is in } L \text{ or } s \text{ is in } M\}$
Concatenation of L and M	$LM = \{st \mid s \text{ is in } L \text{ and } t \text{ is in } M\}$
Kleene closure of L	$L^* = \cup_{i=0}^{\infty} L^i$
Positive closure of L	$L^+ = \cup_{i=1}^{\infty} L^i$

Language

A language is considered as a finite set of strings over some finite set of alphabets. Computer languages are considered as finite sets, and mathematically set operations can be performed on them. Finite languages can be described by means of regular expressions.

Natural Language

- The syntax of a **natural language**, that is, a spoken language, such as English, French, German, or Spanish, is extremely complicated.

- In fact, it does not seem possible to specify all the rules of syntax for a natural language.

Formal Language

- Formal language, which, unlike a natural language, is specified by a *well-defined set of rules of syntax*. Rules of syntax are important not only in linguistics, the study of natural languages, but also in the study of programming languages.
- This set of **rules of syntax** is called a grammar.

Phrase-structure grammar

A phrase-structure grammar **G = {V,T,S,P},** where-

V: vocabulary

T: a subset of V consisting of terminal elements

S: start symbol

P: a set of productions or rules

N: V – T, is known as non-terminal symbols, every production must contain at least one nonterminal on its left side.

Types of Phrase-structure grammar

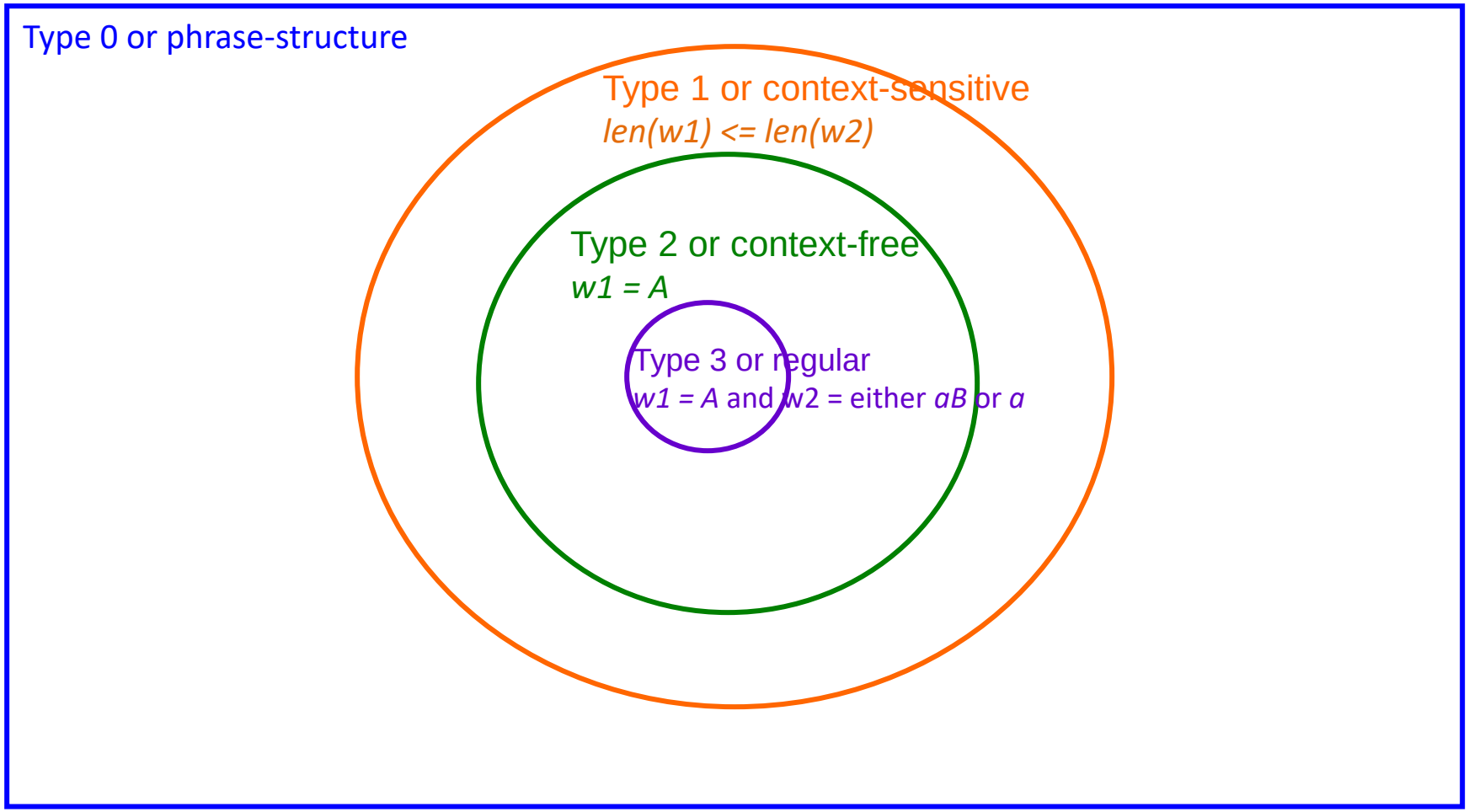
It can be classified according to the types of productions that are allowed.

For productions: w1 ⟶ w2

- Type 0: no restrictions
- Type 1: *len(w1) <= len(w2)*
- Type 2: *w1 = A*
- Type 3: *w1 = A* and w2 = either *aB* or *a*

A, B are nonterminal and *a* is a terminal symbol.

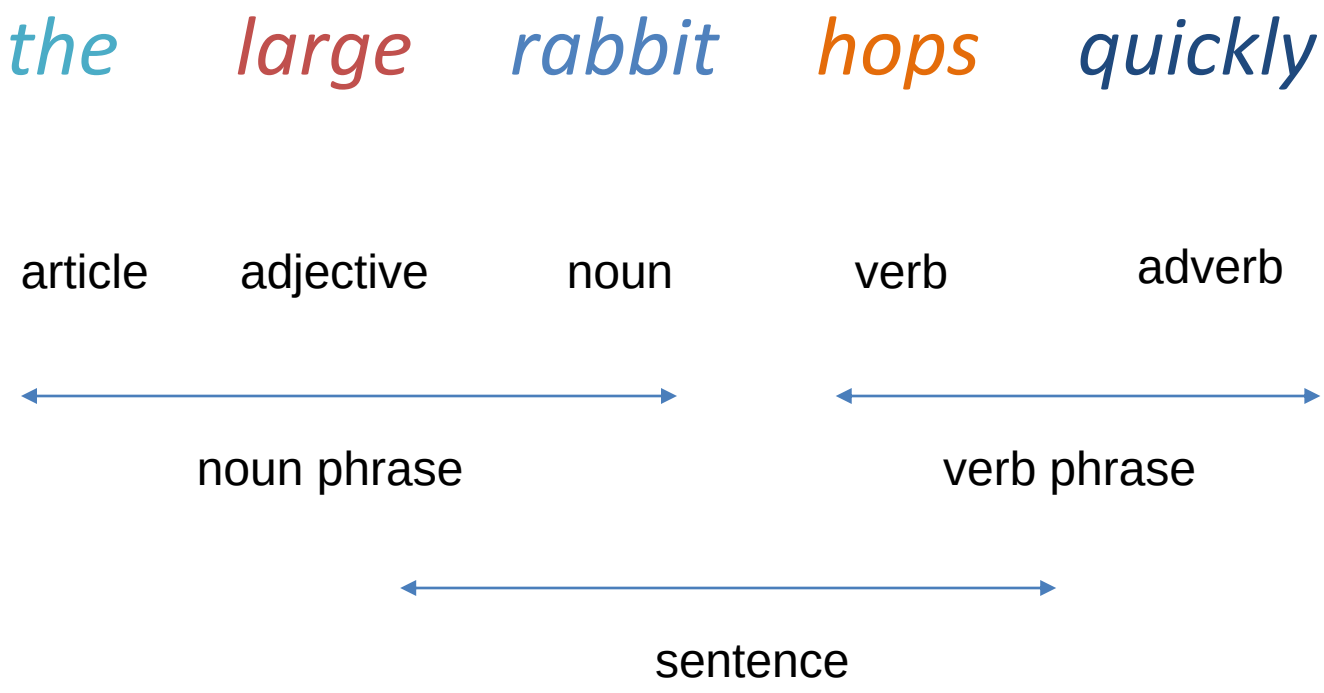
Types of Phrase-structure grammar

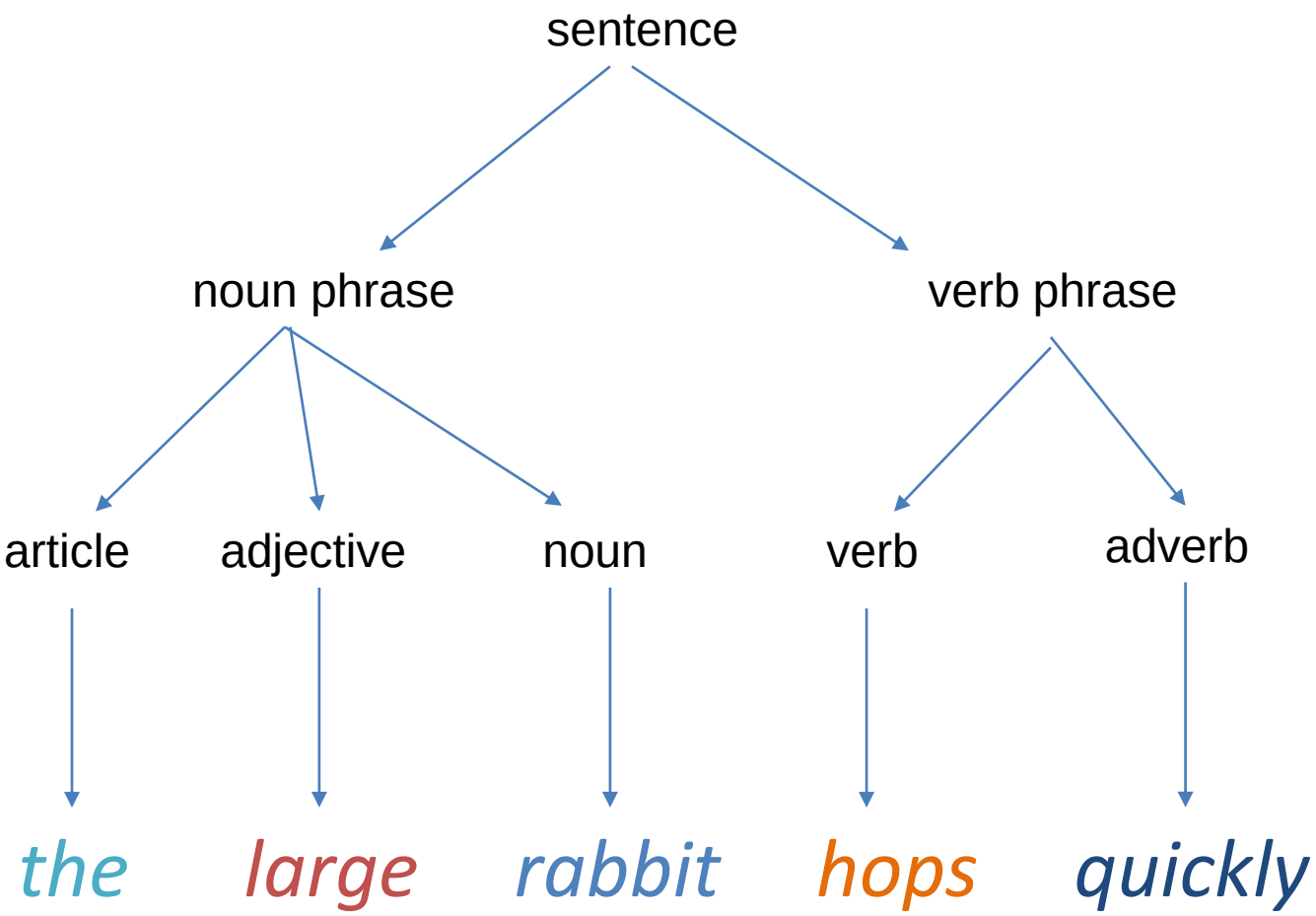


A Sample Grammar (context free)

1. *sentence* => *noun_phrase verb_phrase*
2. *noun_phrase* => *article adjective noun* |
3. *article noun*
4. *verb_phrase* => *verb adverb* |
5. *verb*

the large rabbit hops quickly




sentence
noun phrase
verb phrase
article
adjective
noun
verb
adverb
the
large
rabbit
hops
quickly