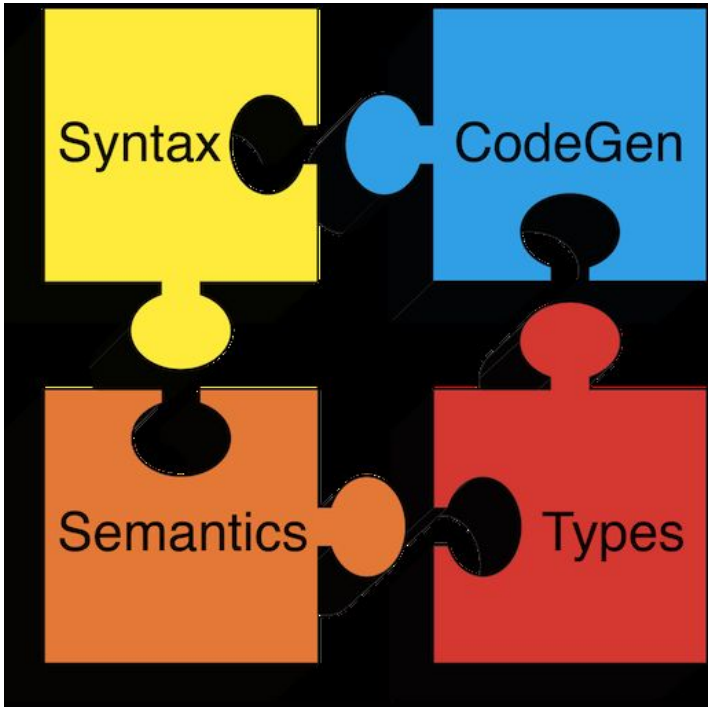
Языки программирования. Часть 2. Компиляторы



Компиляторы

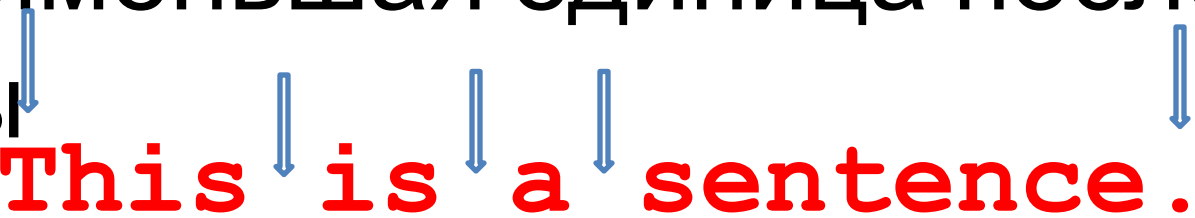
Структура компилятора

Структура компилятора

1. Лексический анализ
2. Разбор текста
3. Семантический анализ
4. Оптимизация
5. Генерация кода

Структура компилятора

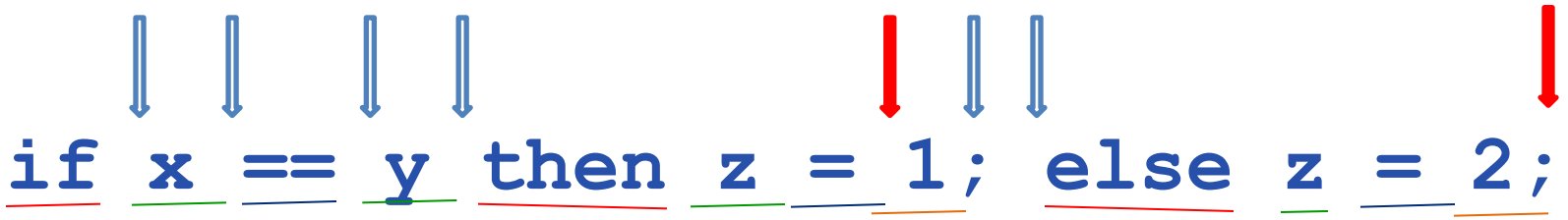
- Первый шаг: распознавание слов.
  – наименьшая единица после буквы



ist his ase nte nce

Лексический анализ

- Лексический анализатор делит текст программы на “слова(words)” или “токены (tokens)”



keywords

variable names

constants

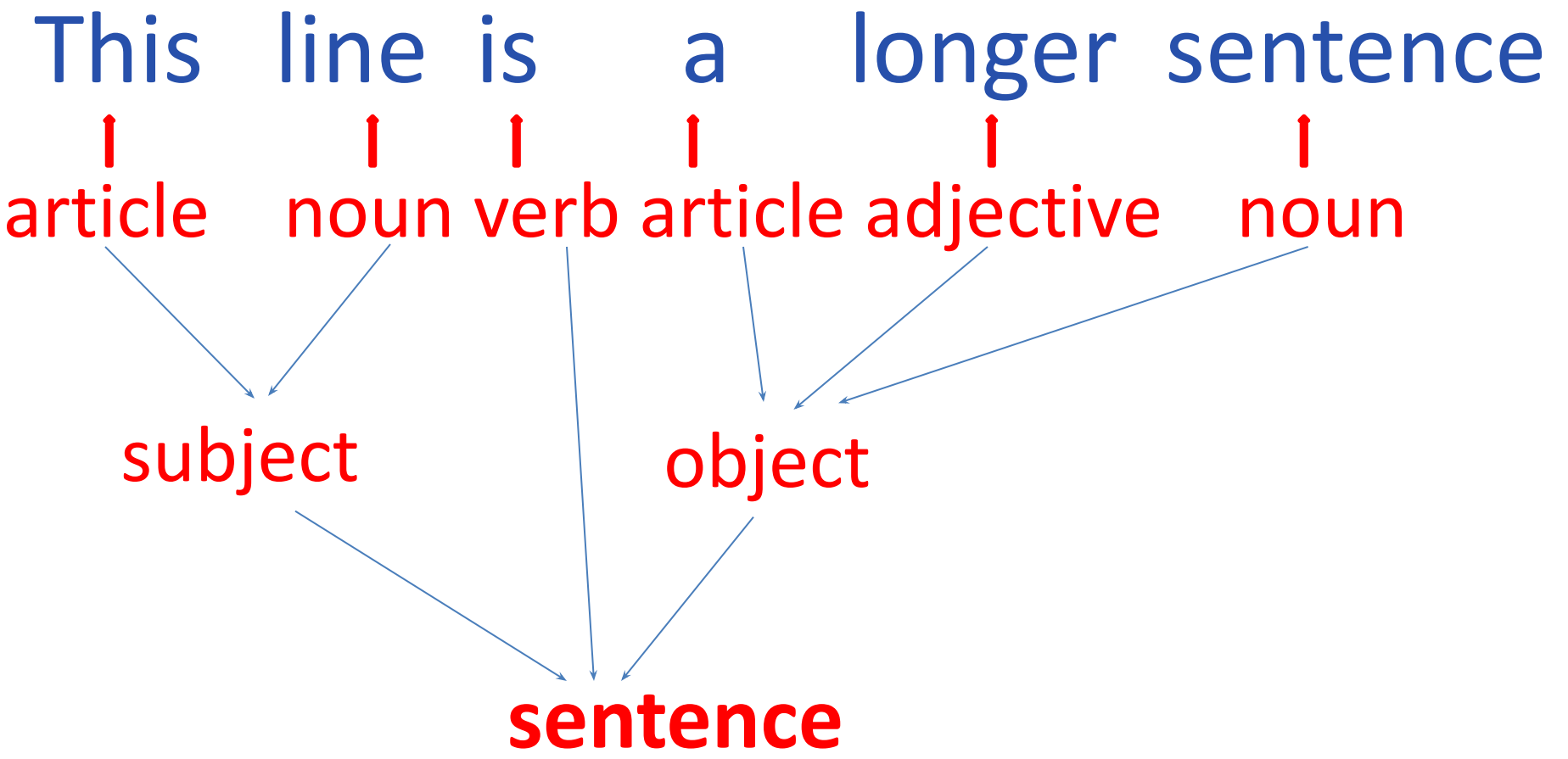
operators

punctuations, separators

Разбор текста

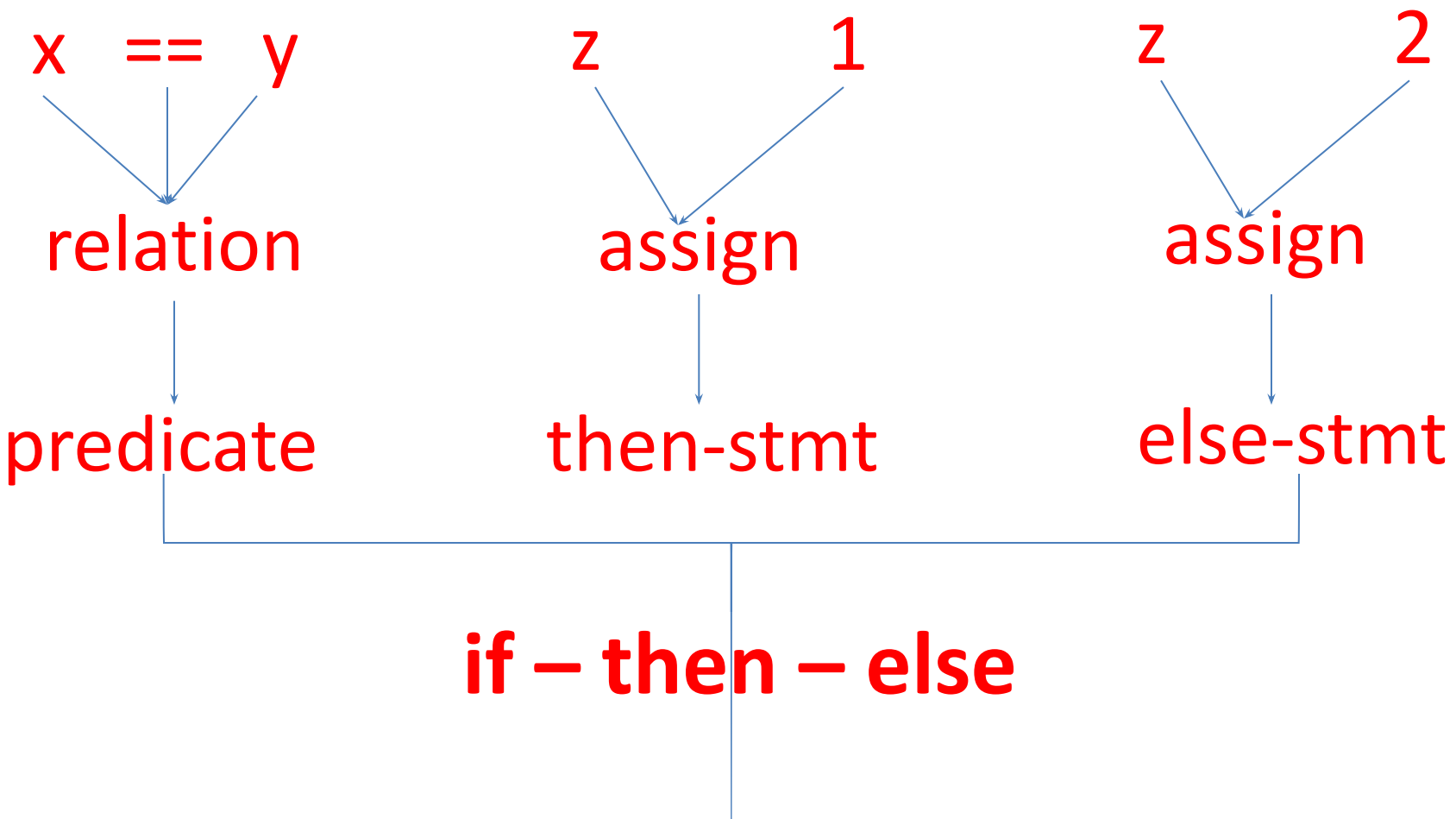
- Как только слова определены, следующим шагом является понимание структуры предложения
- Разбор текста = Построение диаграммы предложения
  – Диаграмма представляется в виде дерева

Разбор текста



Разбор текста

if x == y then z = 1; else z = 2;



Семантический анализ

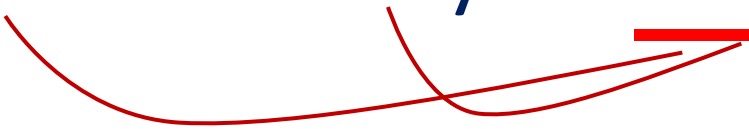
- Как только структура предложения определена, необходимо попытаться понять «смысл»
  - Это сложно!

- Компиляторы выполняют ограниченные виды семантического анализа, чтобы обнаружить несоответствия

Семантический анализ

- **Например:**

  Jack said Jerry left his assignment at home.



1. Jack said Jerry left Jerry's assignment at home.
2. Jack said Jerry left Jack's assignment at home.

- **Ещё хуже:**

  Jack said Jack left his assignment at home?

Семантический анализ

- Языки программирования определяют строгие правила, чтобы избежать таких разногласий

```
{
    int Jack = 3;
    {
        int Jack = 4;
        cout << Jack;
    }
}
```

Семантический анализ

- Компиляторы выполняют множество семантических проверок помимо привязок переменных (variable bindings)
- Например:

  Jack left her homework at home.

- “несоответствие типов” между her и Jack; обозначает, что это разные люди (аналог проверки типов компилятором)

Оптимизация

- Оптимизация не имеет сильного соответствия в естест
  - But a little bit like editing

akin to

- Автоматическое преобразование программы нацелено на то, чтобы
  - Работать быстрее
  - Использовать меньше памяти
  - Мощность
  - Сеть
  - База данных

Оптимизация

X = Y * 0 is the same as X = 0

No !

valid for integers

Invalid for floating point

NaN – not a number

NaN * 0 = NaN

Генерация кода

Code Gen

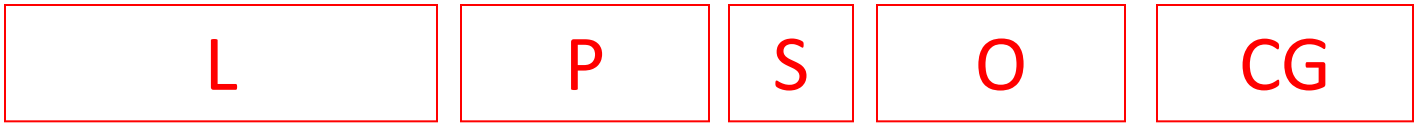
- Создает код сборки (обычно)

- Перевод на другой язык
  – Аналогично переводу в ЕЯ-х

Прим. ЕЯ – естественный язык

Генерация кода

- Общая структура большинства компиляторов соответствует схеме
- Пропорции изменились со времен FORTRAN 1



Прим. L- лексический анализ, P(arse)- разбор, S-семантический анализ, O - оптимизация, CG – генерация кода

Спасибо за внимание