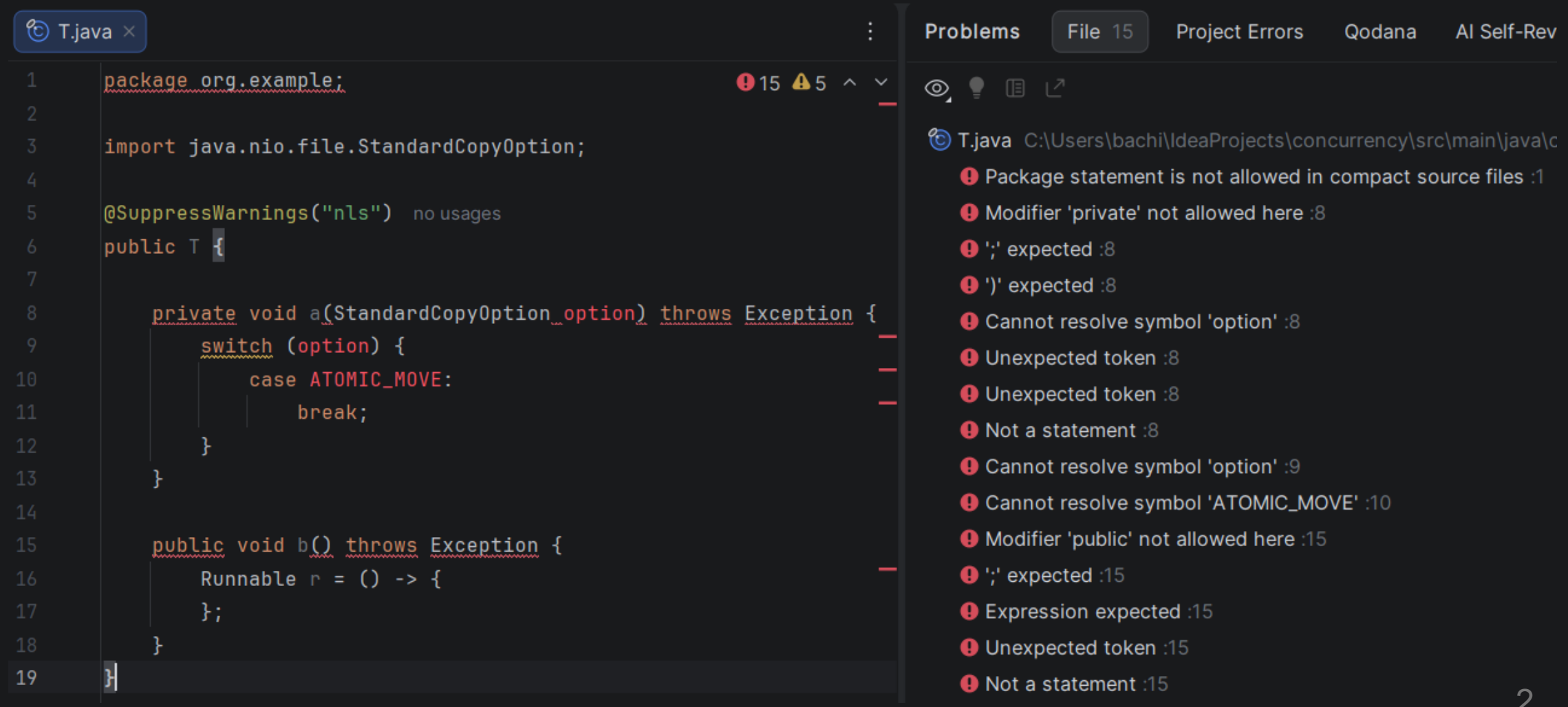
Почему IDE ломается из-за одной ошибки

И как понять, хороший ли у неё error-recovery

Оля Бачище

Одна ошибка – красное окно IDE



План доклада

- Ошибки глазами разработчика

План доклада

- Ошибки глазами разработчика
- Парсер и AST: скрытый фундамент IDE

План доклада

- Ошибки глазами разработчика
- Парсер и AST: скрытый фундамент IDE
- Зачем это AI-агентам

План доклада

- Ошибки глазами разработчика
- Парсер и AST: скрытый фундамент IDE
- Зачем это AI-агентам
- Какие существуют парсеры для Java

План доклада

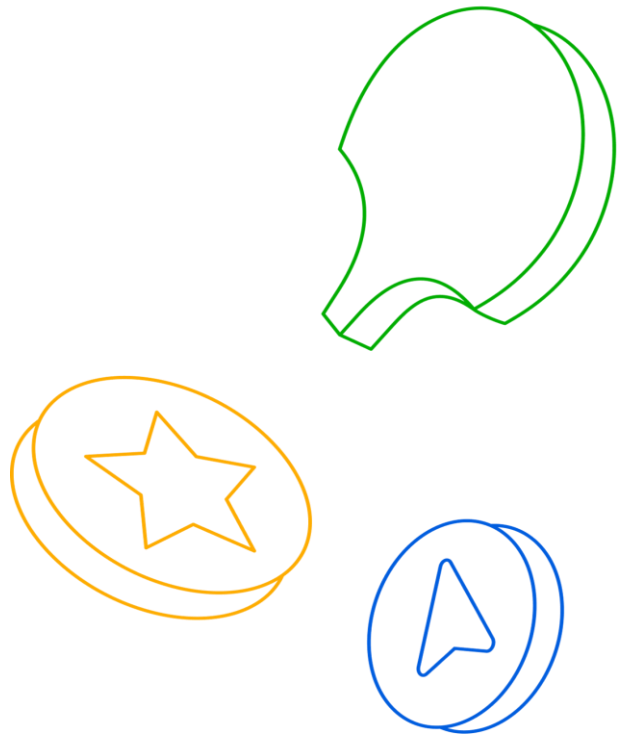
- Ошибки глазами разработчика
- Парсер и AST: скрытый фундамент IDE
- Зачем это AI-агентам
- Какие существуют парсеры для Java
- Error recovery: как понимать сломанный код

План доклада

- Ошибки глазами разработчика
- Парсер и AST: скрытый фундамент IDE
- Зачем это AI-агентам
- Какие существуют парсеры для Java
- Error recovery: как понимать сломанный код
- Можно ли измерить качество error recovery

Об авторе

- LLM-инженер в команде AI-агентов 2ГИС
- Больше 2х лет разрабатывала IDE
- Аспирант НИУ ИТМО
- Преподаю формальные языки



Ошибки глазами разработчика

Чего мы ожидаем от ошибок
при разработке и отладке

Идеальное восстановление после ошибок

```
class T {
    void f()
       System.out.println(“x");
     }
    void g() {}
}
```

1. Найти

Идеальное восстановление после ошибок

```
class T {
    void f()
       System.out.println(“x");
     }
    void g() {}
}
```

❌ Ошибка: вы пропустили открывающую скобку во 2 строке.

1. Найти
2. Сообщить

Идеальное восстановление после ошибок

```
class T {
    void f()
       System.out.println(“x");
     }
    void g() {}
}
```

❌ Ошибка: вы пропустили открывающую скобку во 2 строке.

Я неявно её подставил, весь код до и после в полном порядке.

1. Найти
2. Сообщить
3. Исправить

Обработка ошибок похуже

```
class T {
    void f()
        System.out.println(“x");
    }
    void g() {}
}
```

Обработка ошибок похуже

```
class T {
    void f()
        System.out.println(“x");
    }
    void g() {}
}
```

Ошибка: в коде какая-то ошибка.
Исправьте её как-нибудь.

Ужасная обработка ошибок

```
class T {
    void f()
       System.out.println(“x");
     }
    void g() {}
}
```

Ужасная обработка ошибок

```
class T {
    void f()
       System.out.println(“x");
     }
    void g() {}
}
```

Код в полном порядке! Никаких ошибок нет.
Давай в прод.

Каскадные ошибки

```
class T {
    void f()
       System.out.println(“x");
      }
    void g() {}
}
```

❌ Ошибка: вы что-то пропустили по 2 строке.

1. Найти
2. Сообщить
3. «*Исправить*»

Каскадные ошибки

```
class T {
    void f()
       System.out.println(“x");
      }
    void g() {}
}
```

❌ Ошибка: вы что-то пропустили по 2 строке.

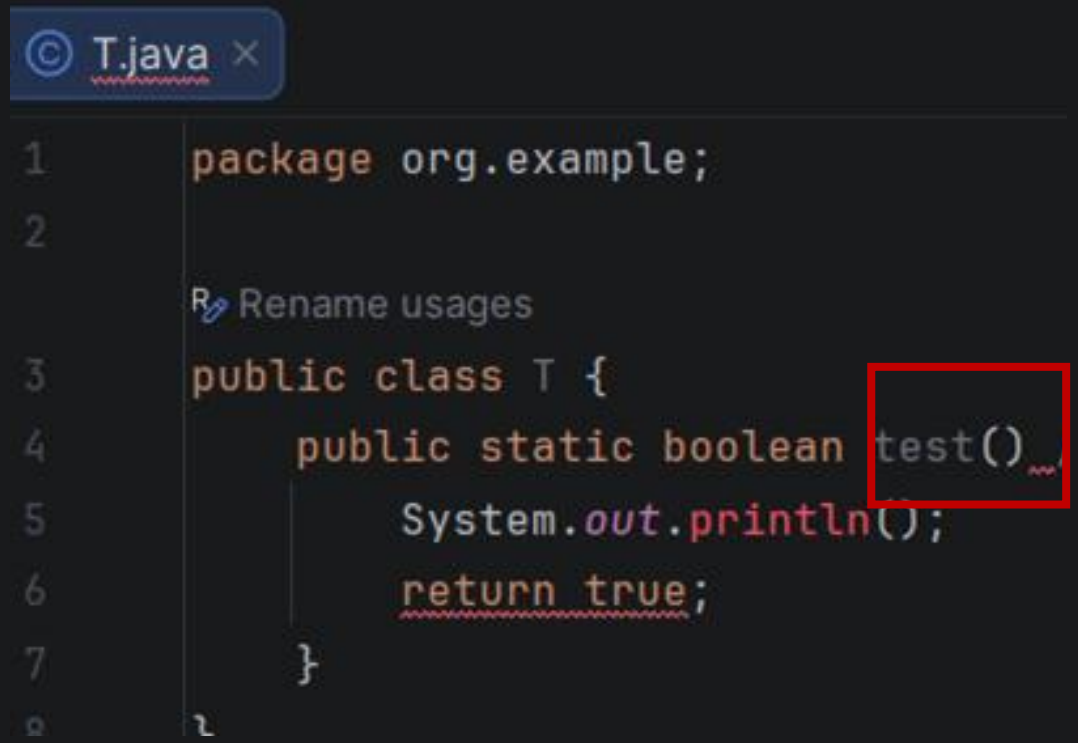
❌ Ошибка: на строке 5 недопустимо использовать “void”

1. Найти
2. Сообщить
3. «*Исправить*»
4. «*Найти*»
5. «*Сообщить*»
6. «*Исправить*»
7. «*Найти*»

…

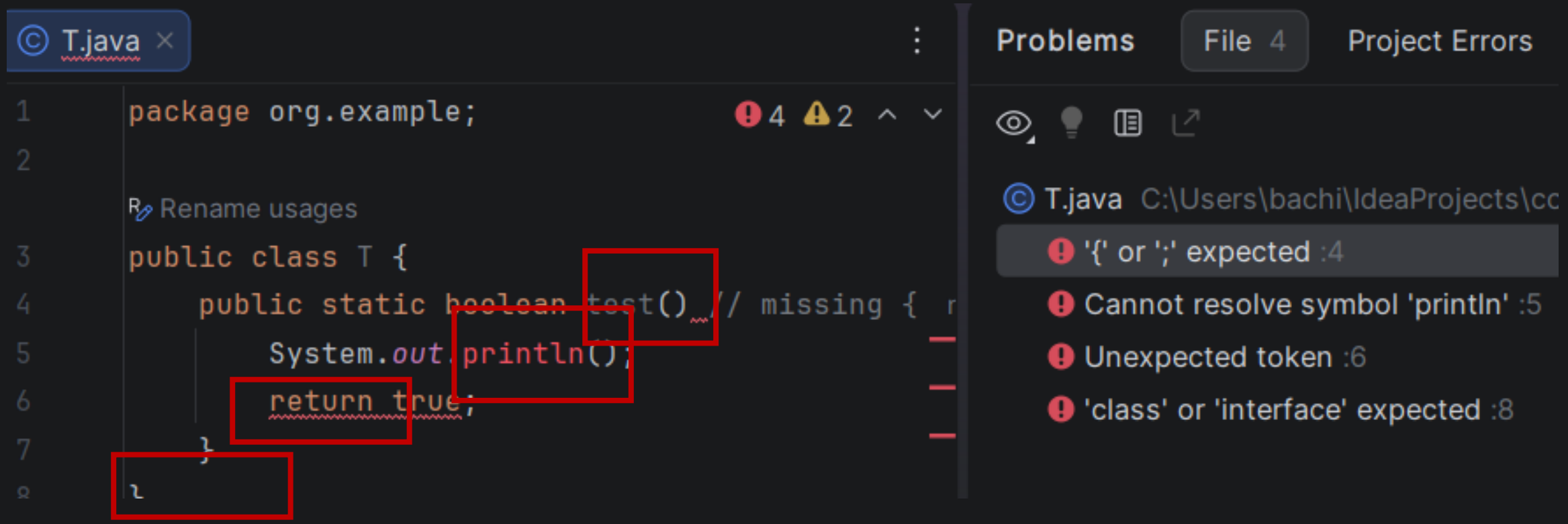
EOF

Пропущена одна скобка



Сколько сообщений от IDE?

Пропущена одна скобка



Сколько сообщений от IDE?

4

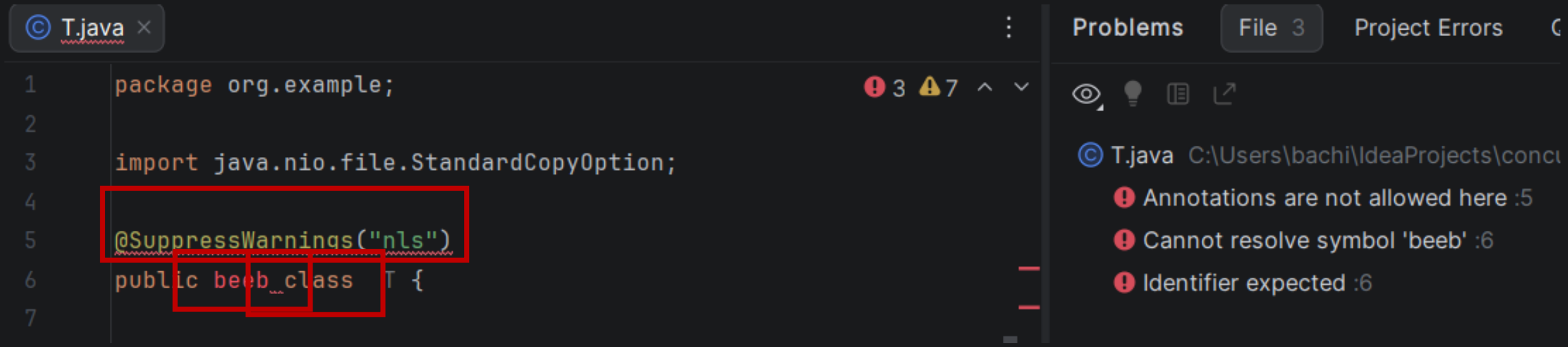
Добавлен лишний id “beeb”

T.java

```
1  package org.example;
2
3  import java.nio.file.StandardCopyOption;
4
5  @SuppressWarnings("nls")
6  public beeb class  T {
7
```

Сколько сообщений от IDE?

Добавлен лишний id “beeb”



Сколько сообщений от IDE?

3

Потеряно ключевое слово class

```
package org.example;
import java.nio.file.StandardCopyOption;
@SuppressWarnings("nls")  no usages
public T {
    private void a(StandardCopyOption option) throws Exception
        switch (option) {
            case ATOMIC_MOVE:
                break;
        }
    }
    public void b() throws Exception {
        Runnable r = () -> {};
    }
}
```

Сколько сообщений от IDE?

Потеряно ключевое слово class

```
package org.example;
import java.nio.file.StandardCopyOption;
@SuppressWarnings("nls")  no usages
public T {
    private void a(StandardCopyOption option) throws Exception {
        switch (option) {
            case ATOMIC_MOVE:
                break;
        }
    }
    public void b() throws Exception {
        Runnable r = () -> {};
    }
}
```

- Package statement is not allowed in compact source files
- Modifier 'private' not allowed here :8
- ';' expected :8
- ')' expected :8
- Cannot resolve symbol 'option' :8
- Unexpected token :8
- Unexpected token :8
- Not a statement :8
- Cannot resolve symbol 'option' :9
- Cannot resolve symbol 'ATOMIC_MOVE' :10
- Modifier 'public' not allowed here :14
- ';' expected :14
- Expression expected :14
- Unexpected token :14
- Not a statement :14

Сколько сообщений от IDE?

15

Вывод по блоку

При ошибке идеальный инструмент для кода

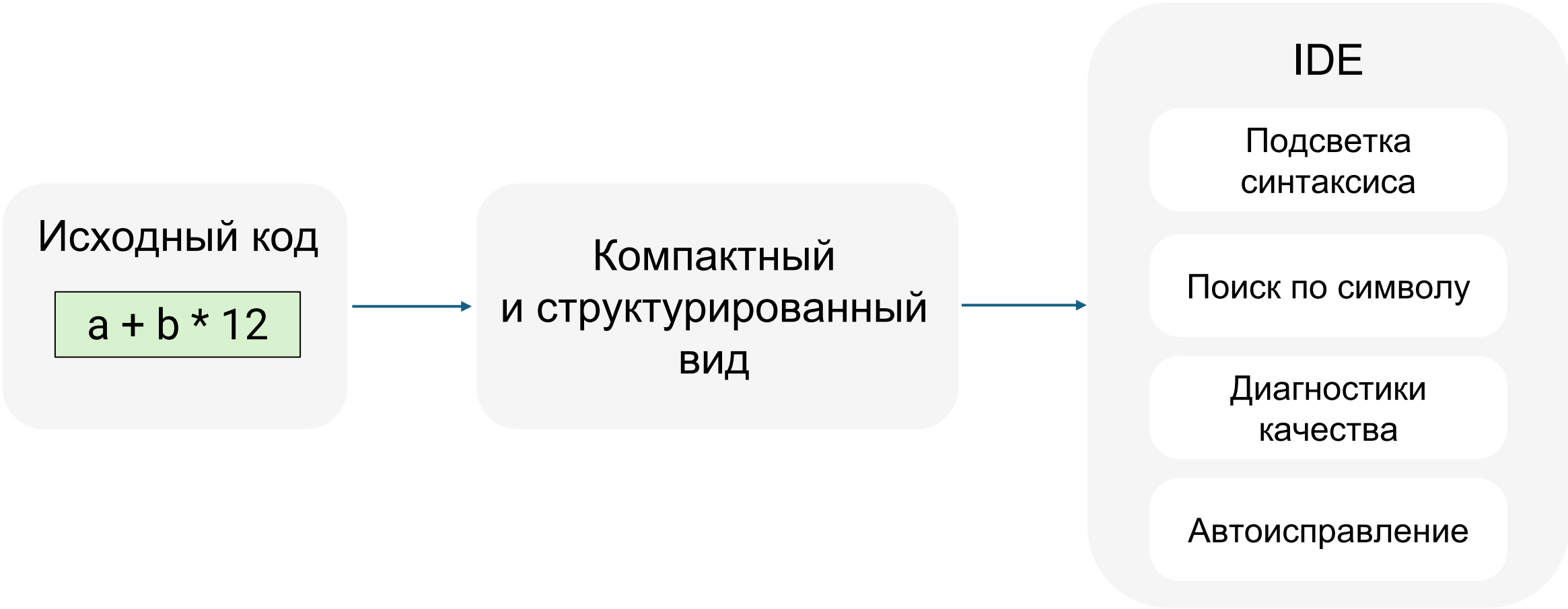
- Находит ее
- Сообщает
- Исправляет

На практике это не всегда так

Поиск ошибок внутри инструментов анализа кода

Что ищет и исправляет ошибки

Основа IDE


Исходный код
a + b * 12
Компактный
и структурированный
вид
IDE
Подсветка
синтаксиса
Поиск по символу
Диагностики
качества
Автоисправление

Парсеры

Исходный код

```
a + b * 12
```

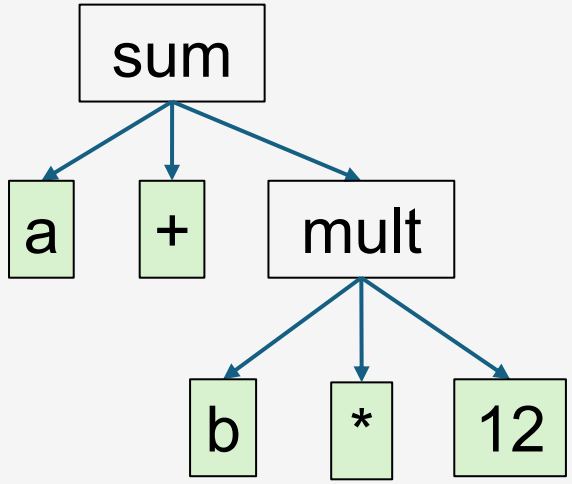
Парсеры

Исходный код

```
a + b * 12
```

парсер

Синтаксическое дерево (AST)



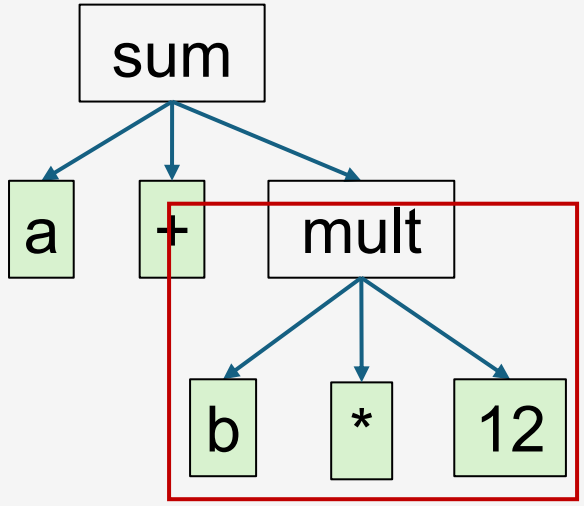
Парсеры

Исходный код

```
a + b * 12
```

парсер

Синтаксическое дерево (AST)



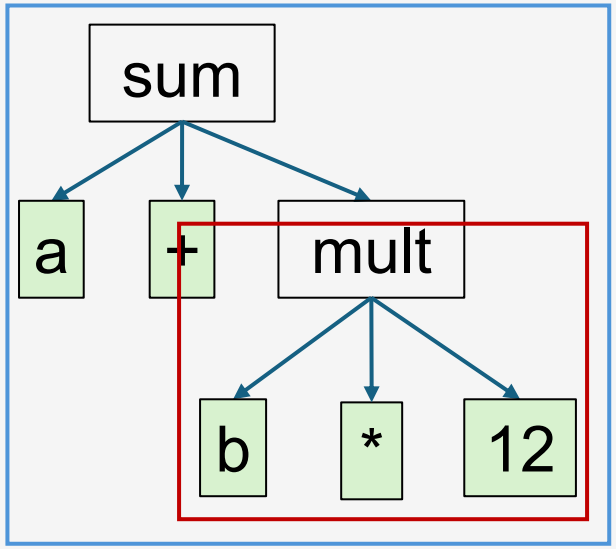
Парсеры

Исходный код

a + b * 12

парсер

Синтаксическое дерево (AST)



Применение AST

- Синтаксический анализ
- Компиляторы и интерпретаторы
- LLM-инструменты для кода
- Запросы к БД

…

Ошибочный код

исходный код
с ошибками

```
a +(((“dsds##b * 12
```

парсер

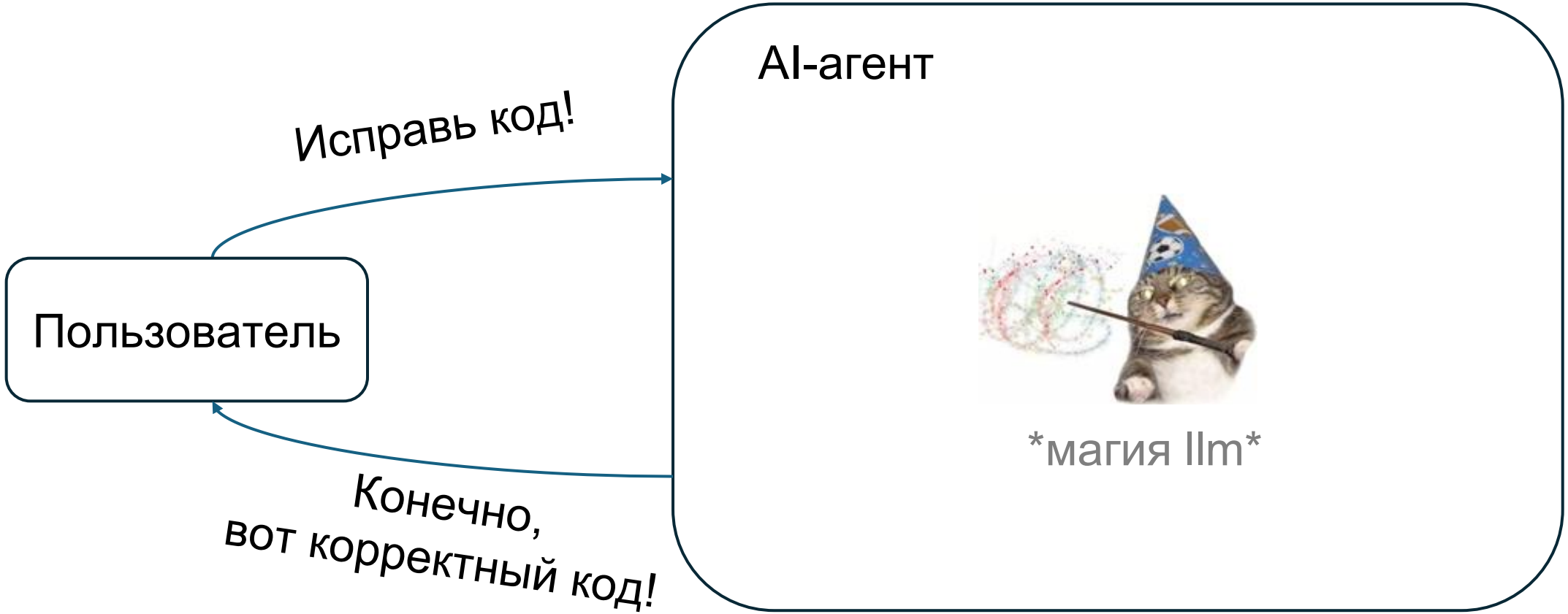
Вывод по блоку

- Парсер превращает исходный код в AST
- AST – это основа IDE и других инструментов анализа

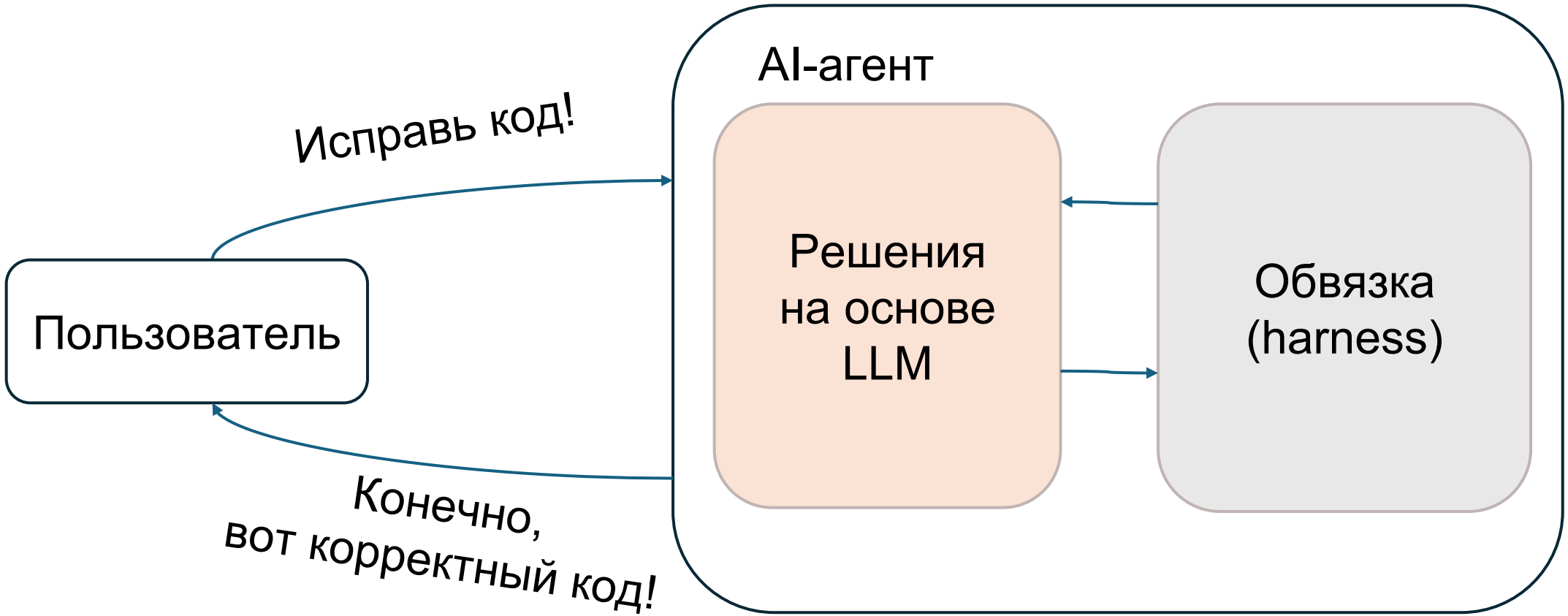
Как парсеры и error-recovery помогают AI-агентам

Зачем нужен парсер и AST,
если весь код можно генерировать и исправлять
с помощью LLM

Кодовый AI-агент



Кодовый AI-агент



Реализация AI агента

Dive into Claude Code: The Design Space of Today's and Future AI Agent Systems (Jiacheng Liu and all, 2026)

Кодовый AI-агент

Решения на основе LLM

- Выбор инструментов
- Генерация кода/текста
- Планирование решения

…

Кодовый AI-агент

Решения на основе LLM

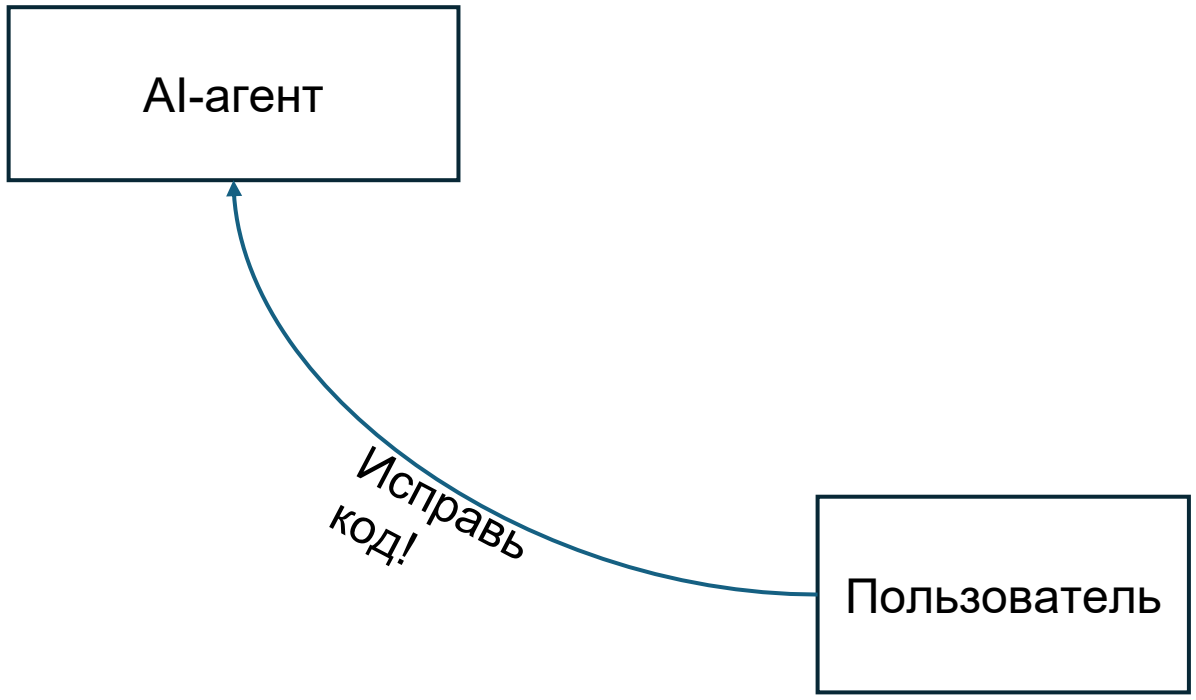
- Выбор инструментов
- Генерация кода/текста
- Планирование решения

…

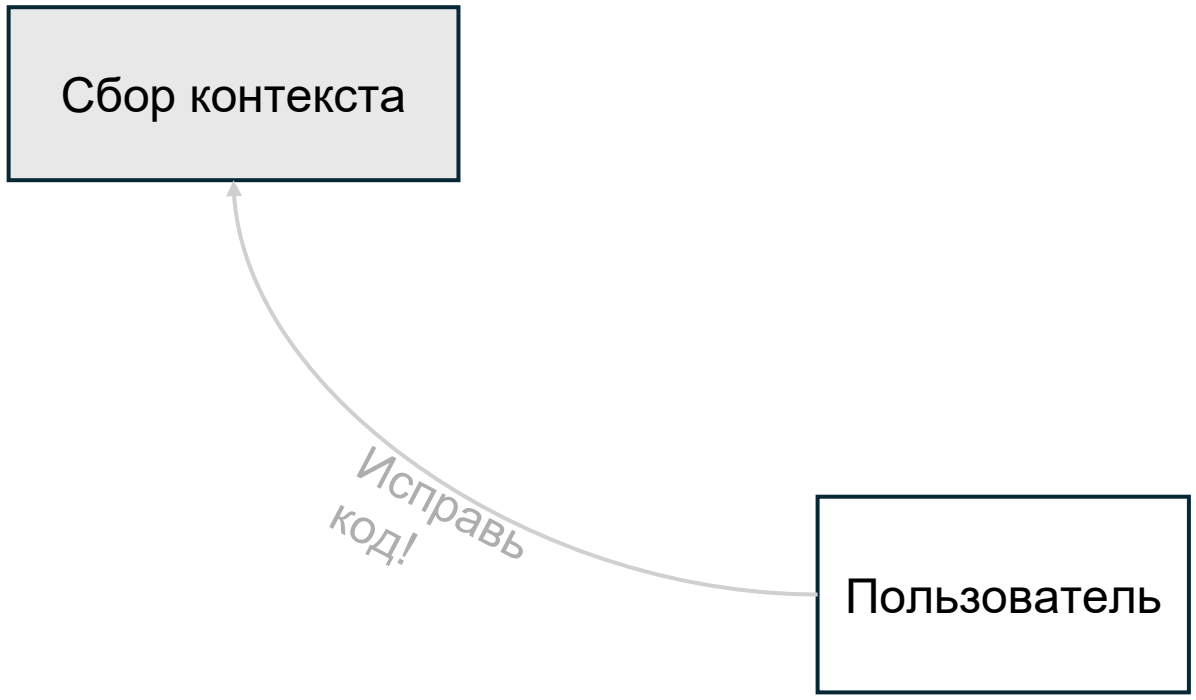
Обязка

- **Валидация кода**

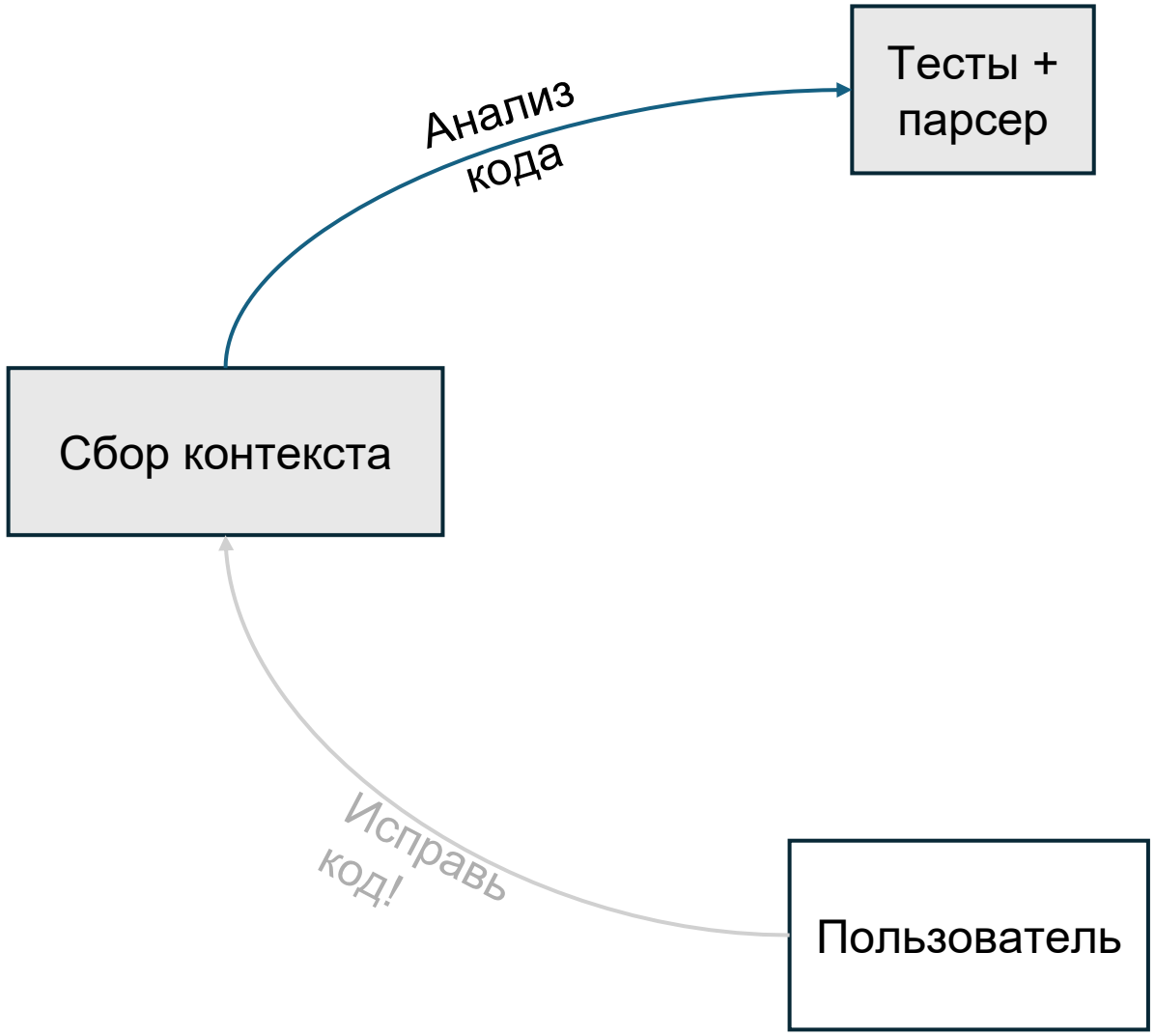
Работа с кодовым агентом



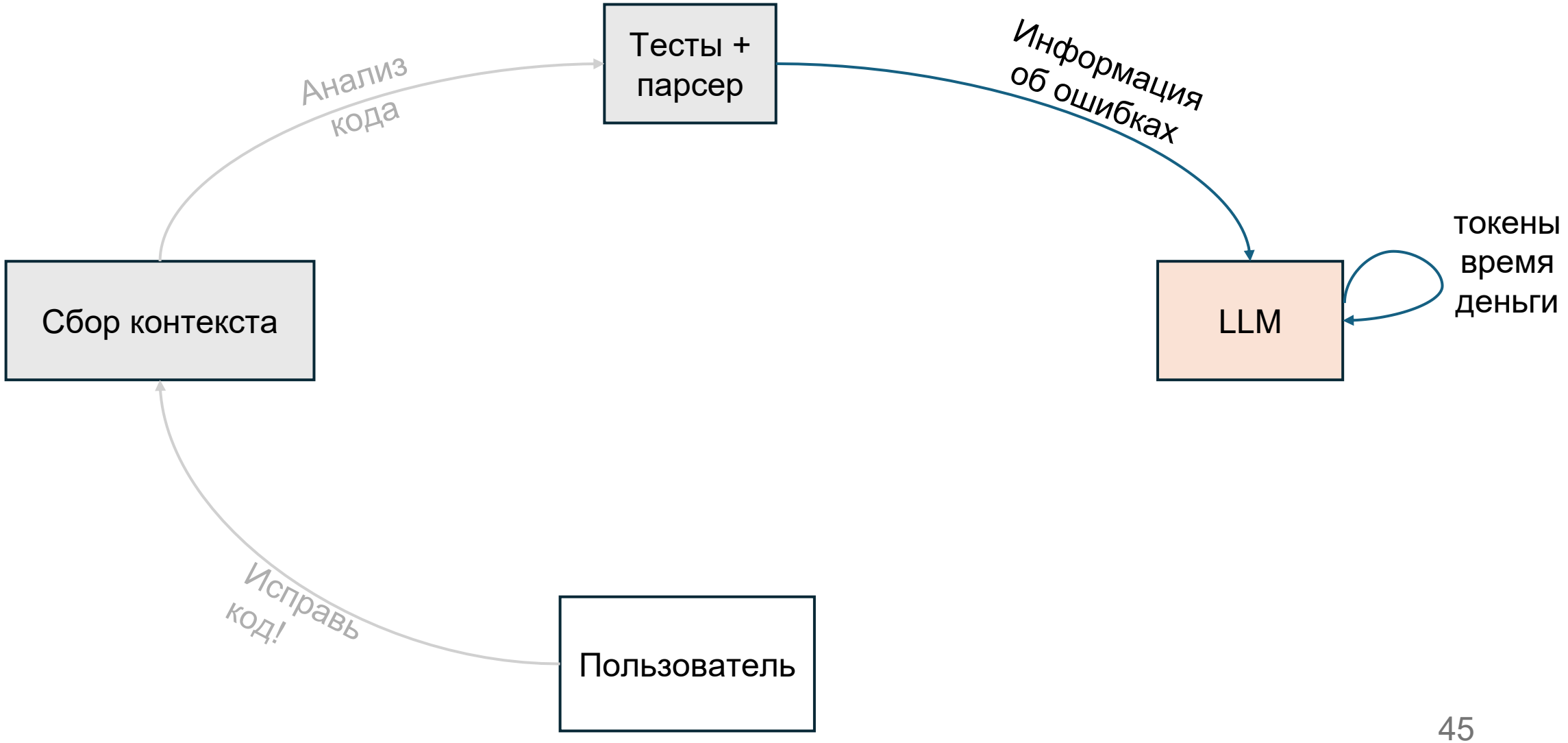
Работа с кодовым агентом



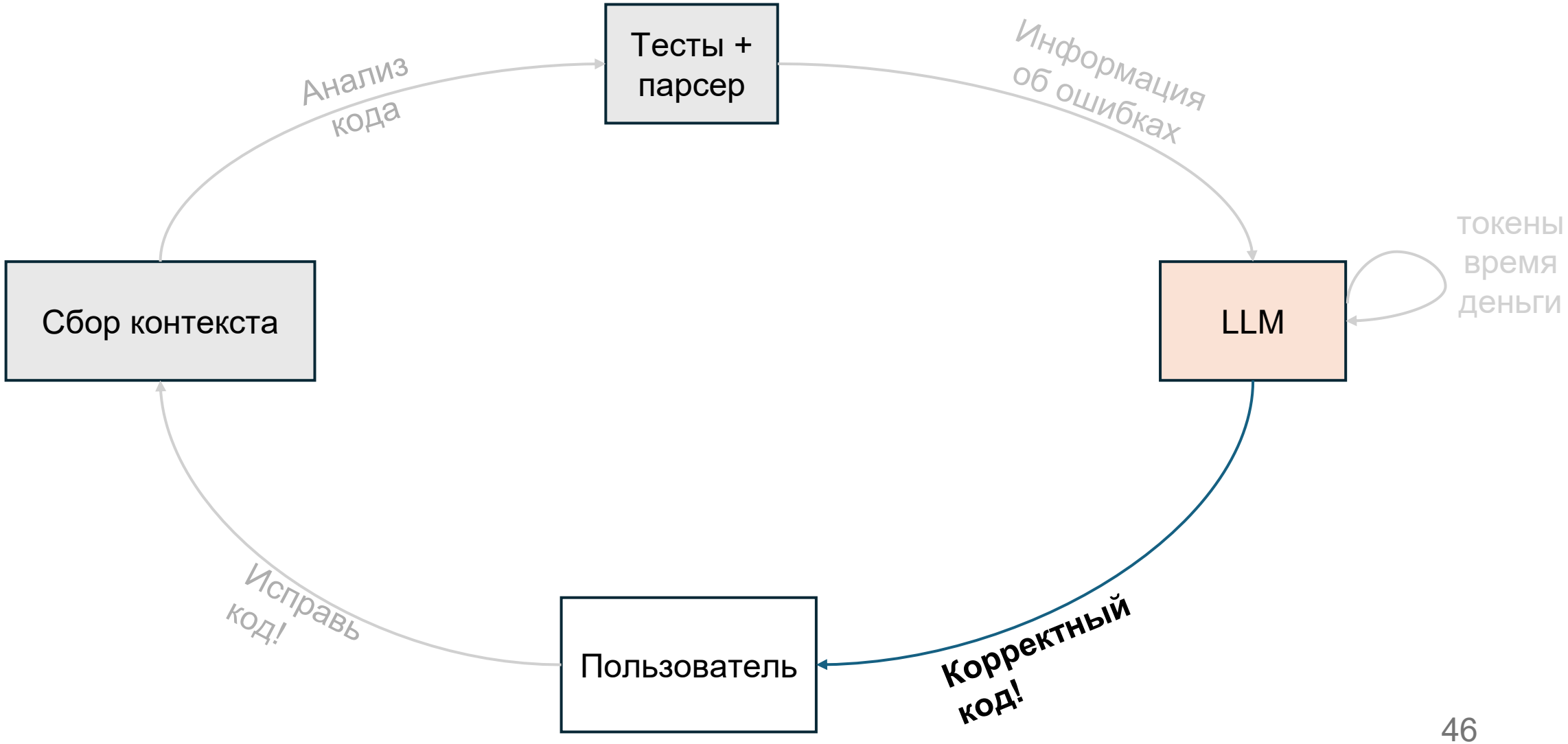
Работа с кодовым агентом



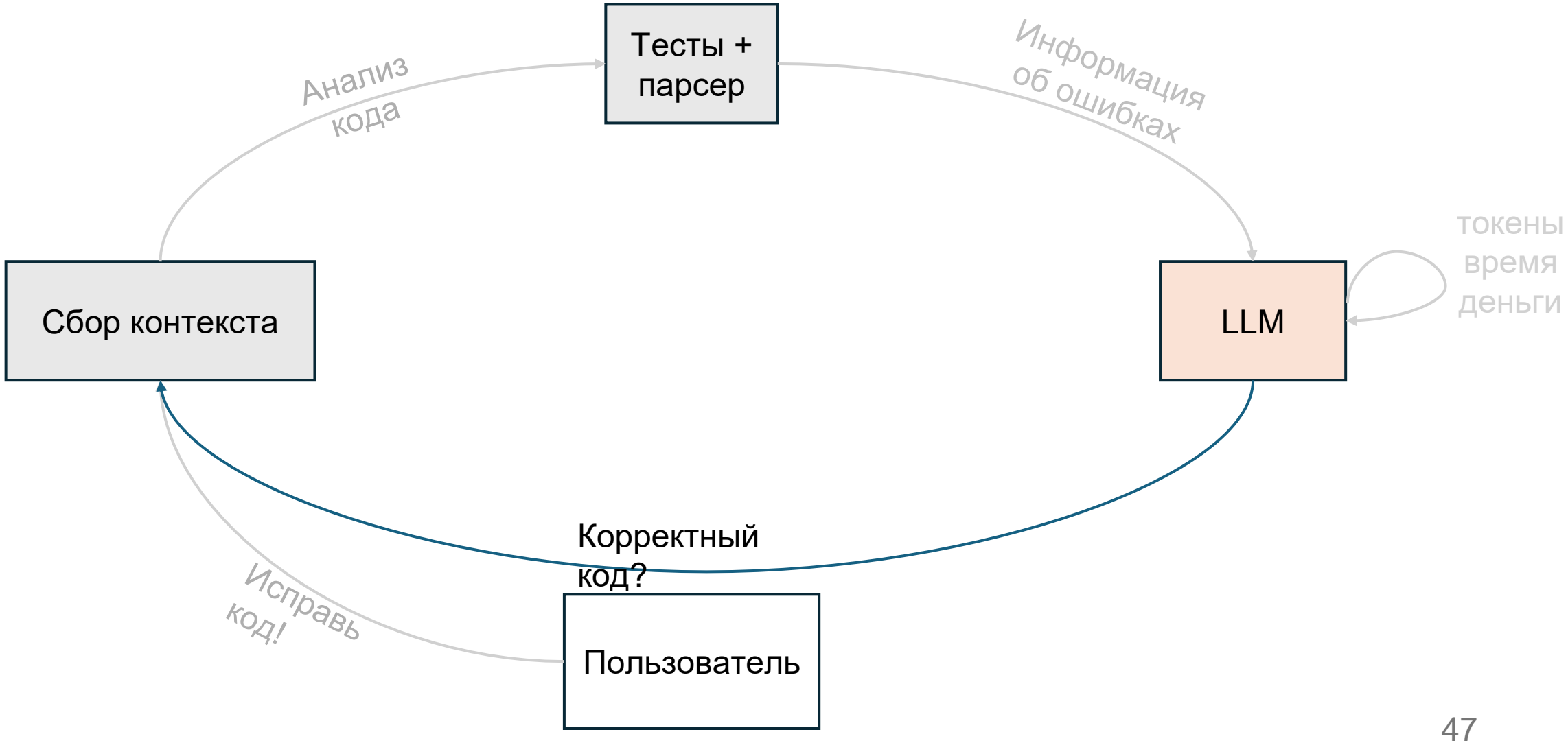
Работа с кодовым агентом



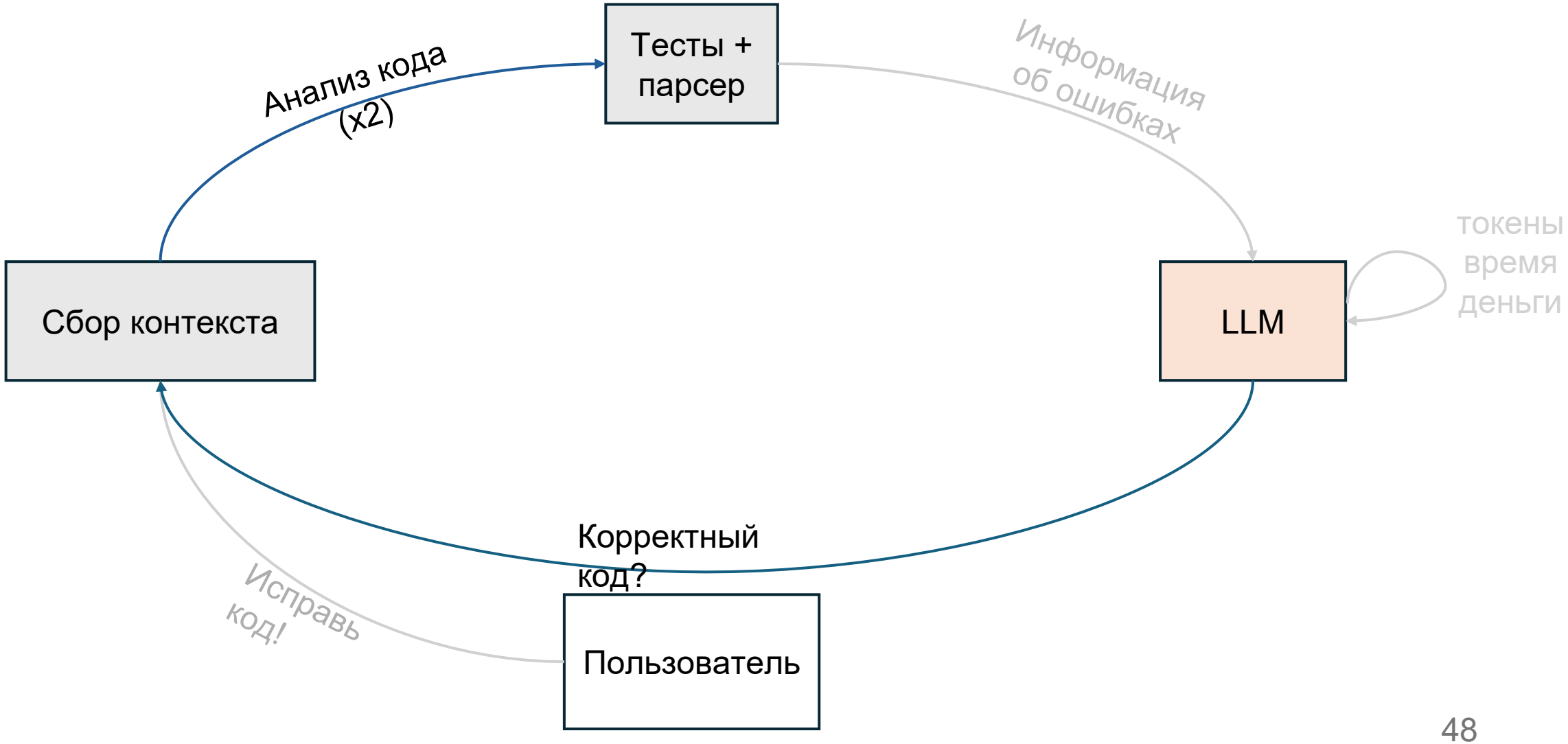
Работа с кодовым агентом



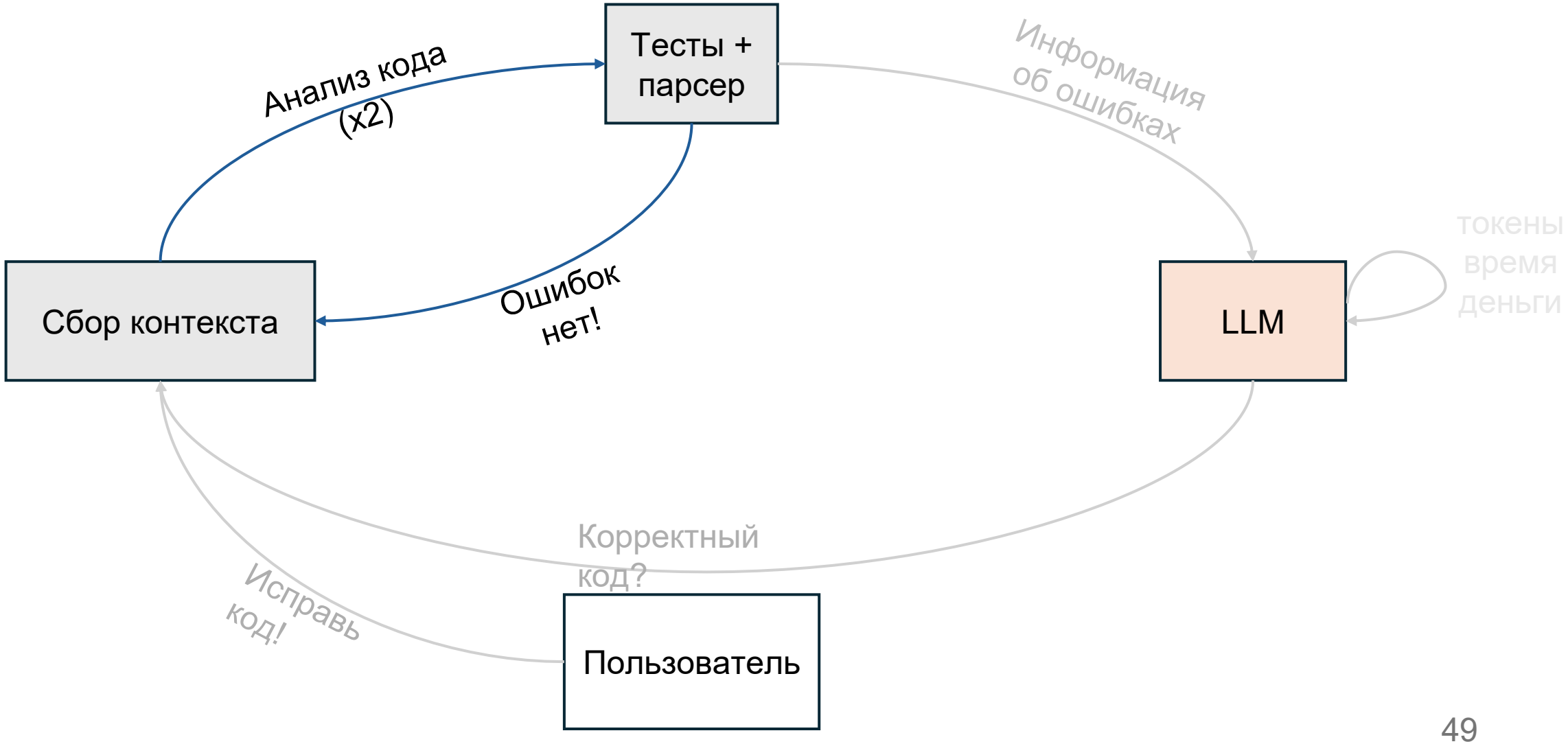
Работа с кодовым агентом



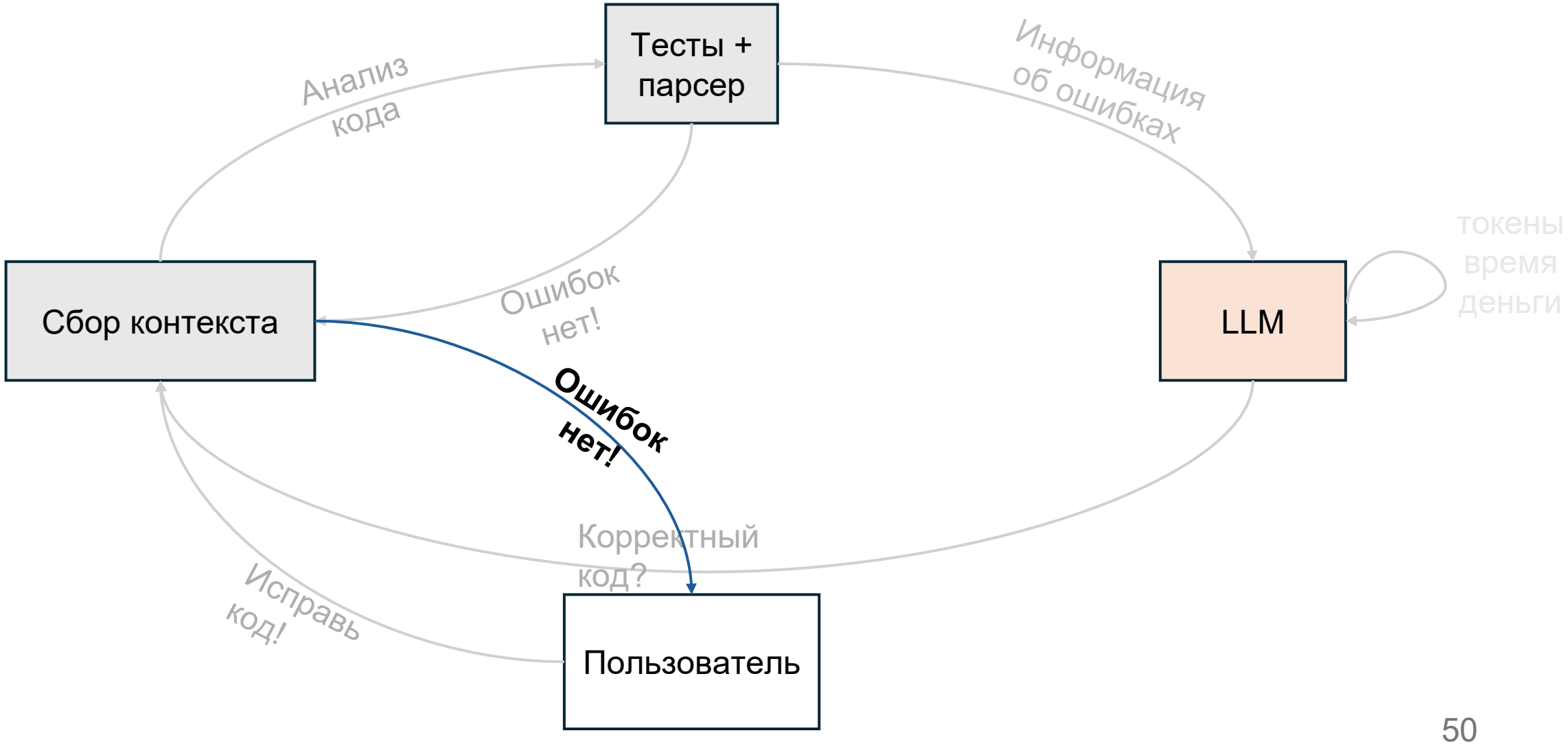
Работа с кодовым агентом



Работа с кодовым агентом

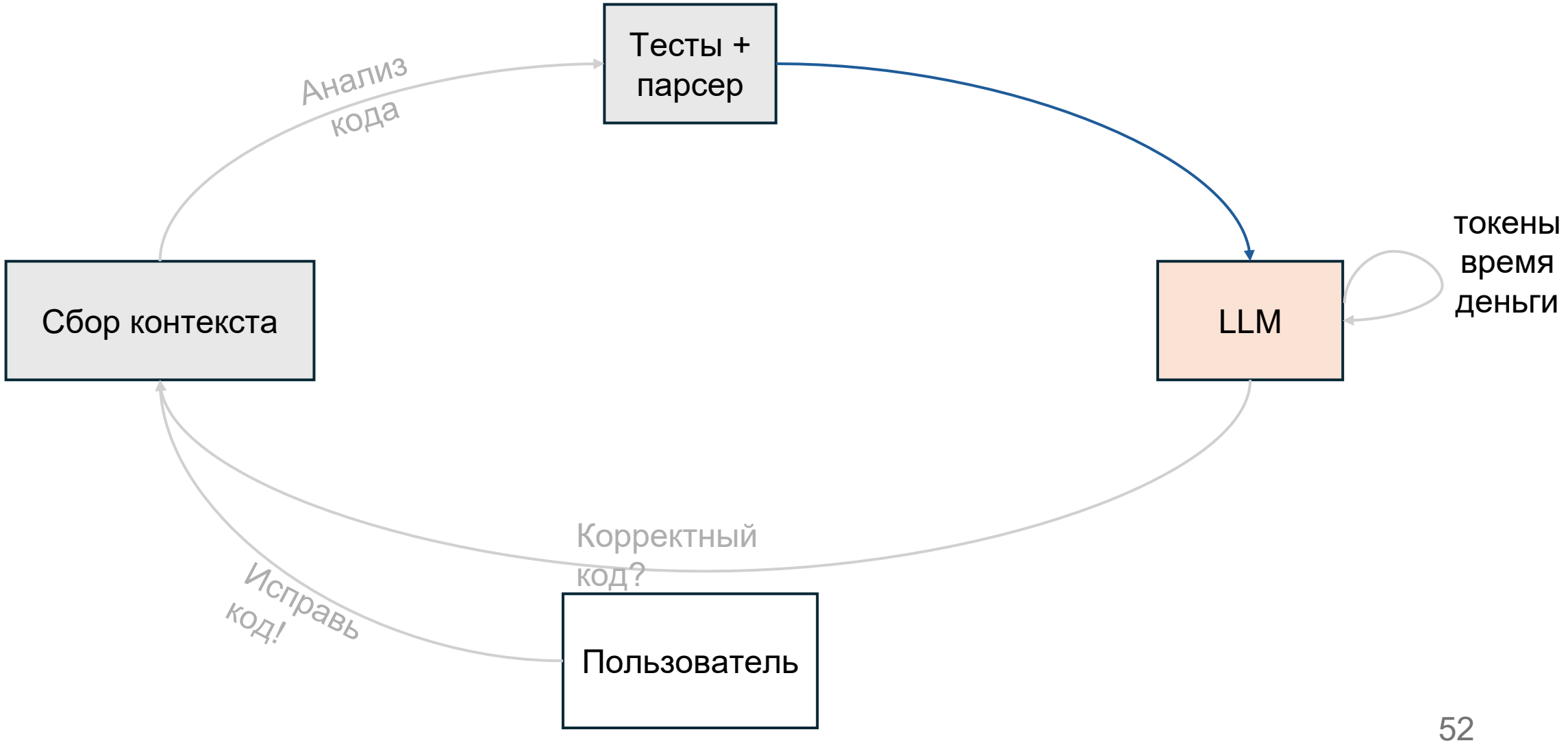


Работа с кодовым агентом

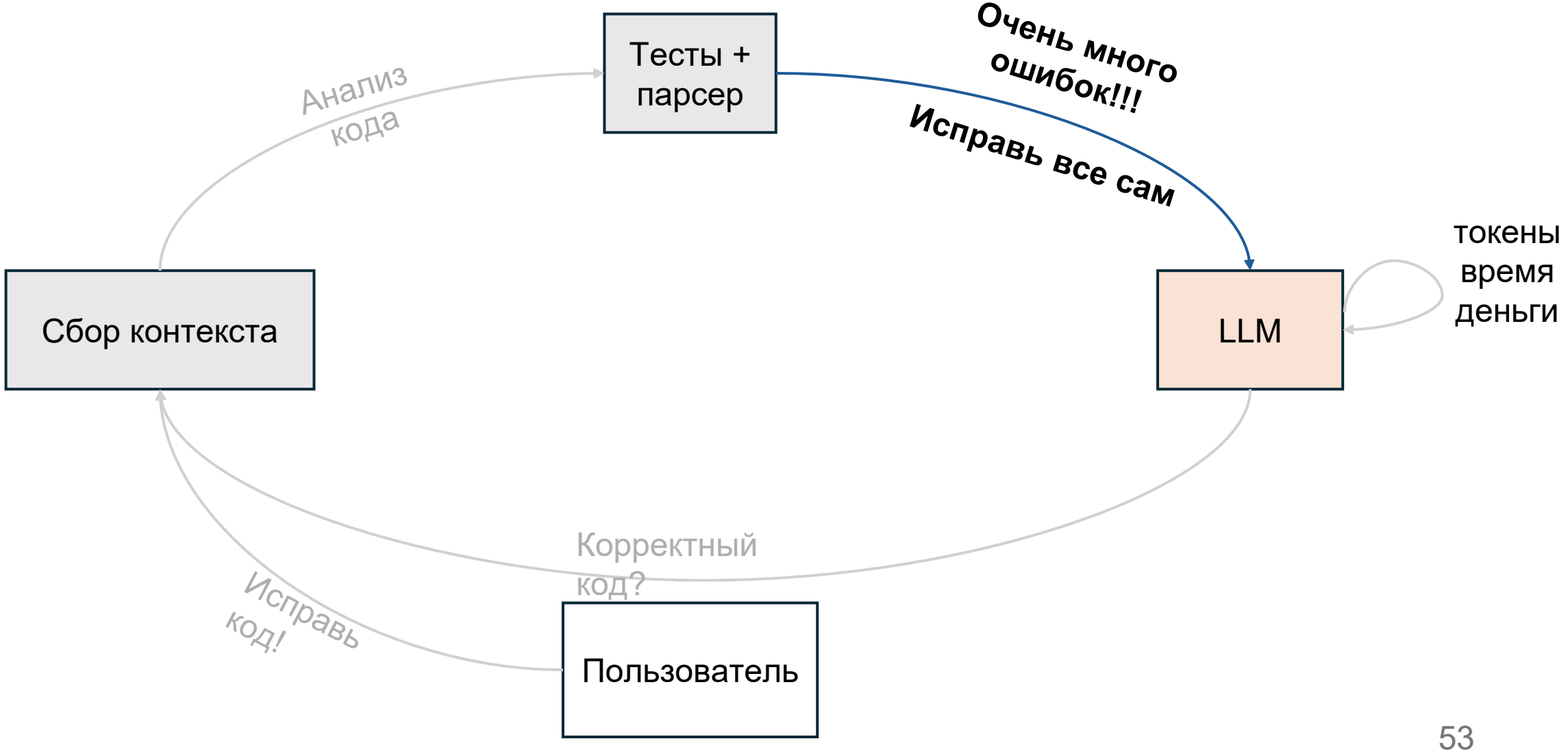


Плохое error-recovery в обвязке

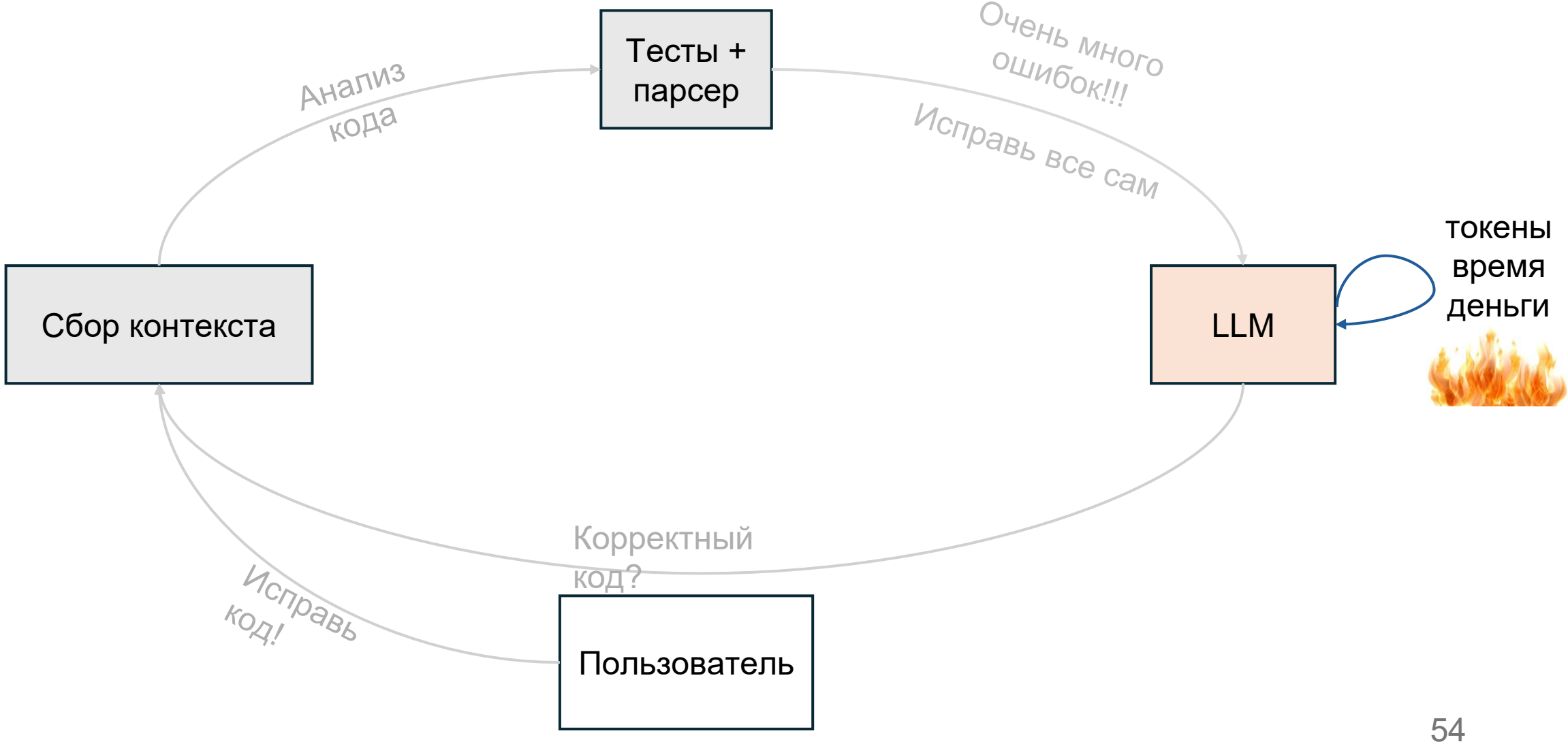
Плохое error-recovery в обвязке



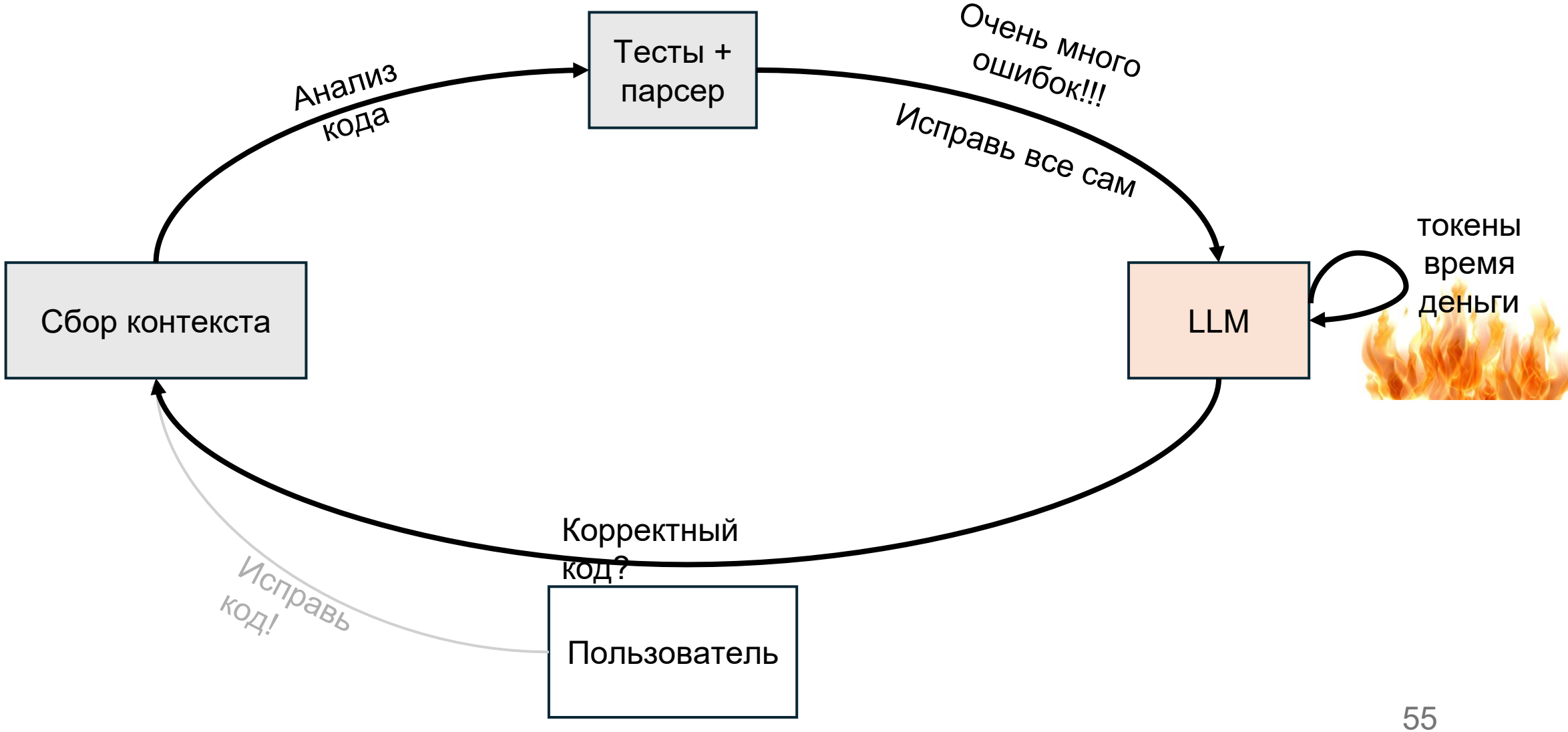
Плохое error-recovery в обвязке



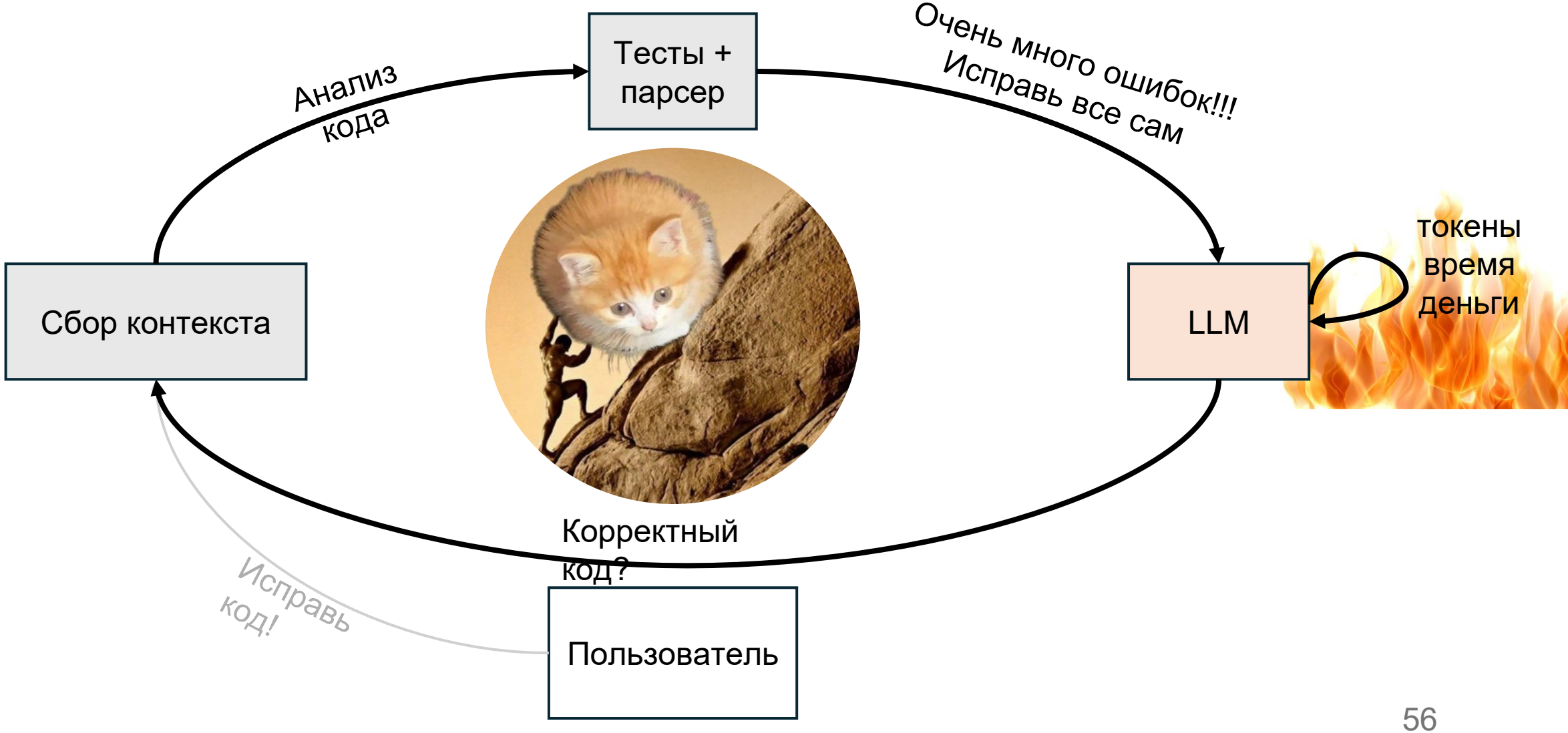
Плохое error-recovery в обвязке



Плохое error-recovery в обвязке

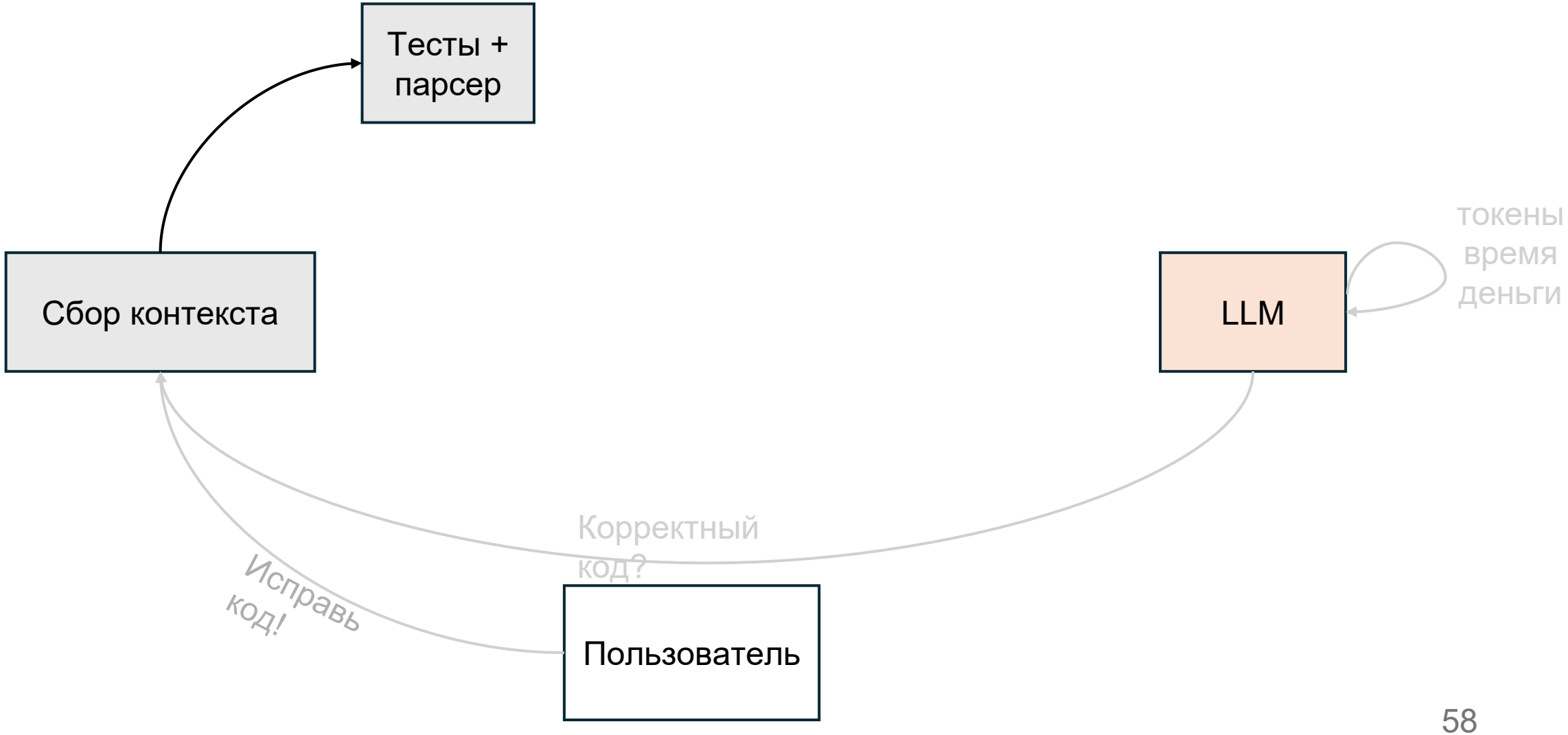


Плохое error-recovery в обвязке

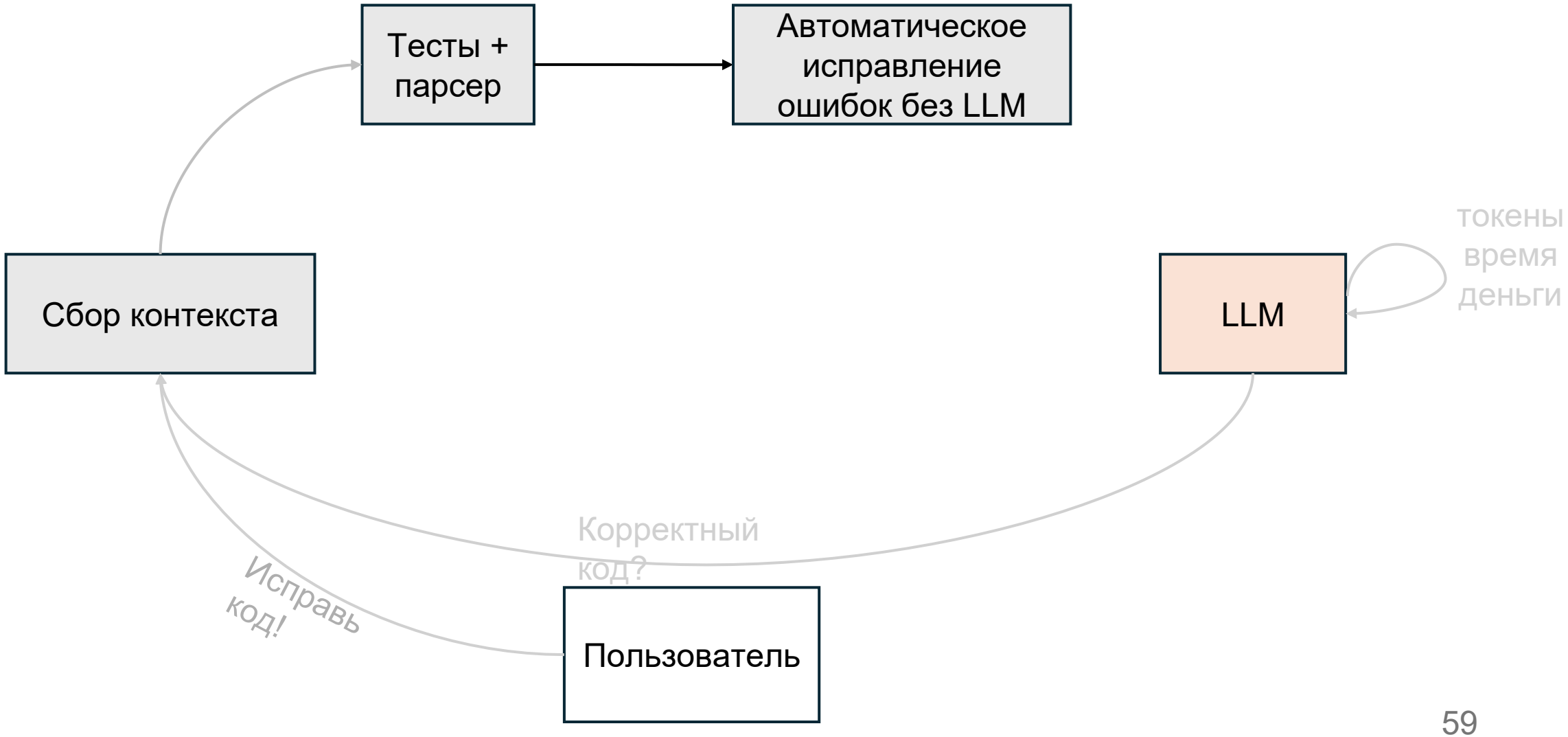


Хорошее error-recovery в обвязке

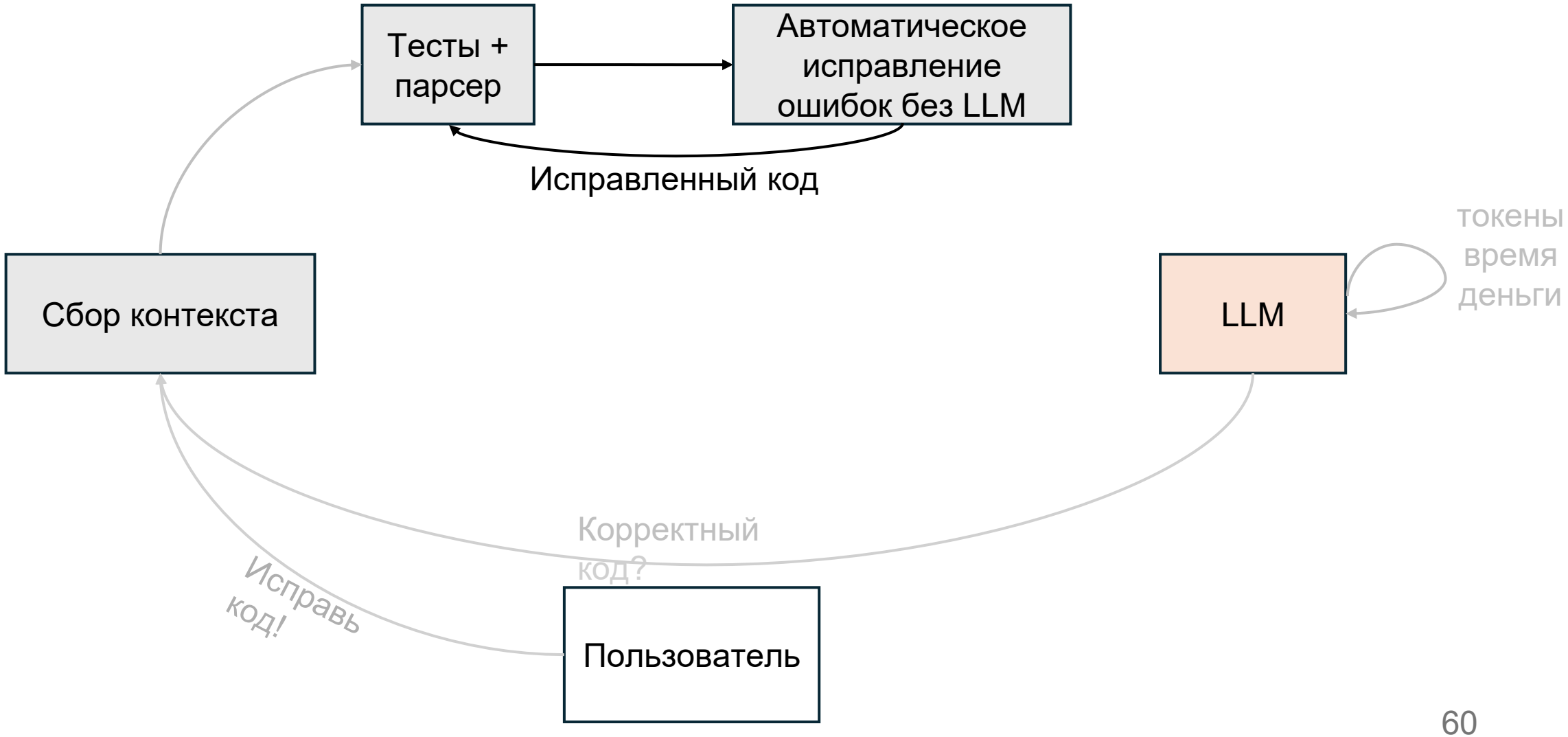
Хорошее error-recovery в обвязке



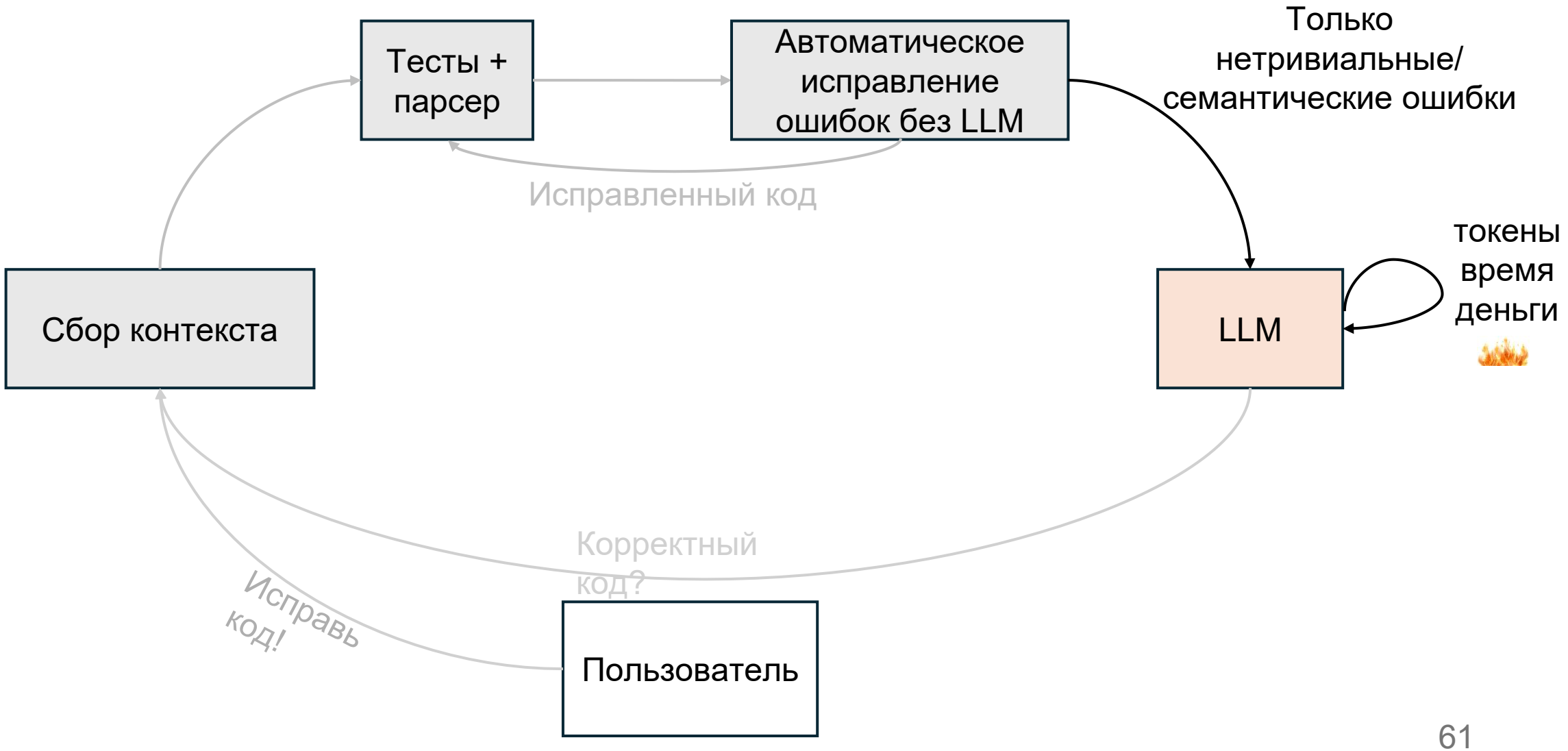
Хорошее error-recovery в обвязке



Хорошее error-recovery в обвязке



Хорошее error-recovery в обвязке



Кодовый AI-агент

Решения на основе LLM

- Выбор инструментов
- Генерация кода/текста
- Планирование решения

…

Обвязка

- Валидация кода
- **Сбор/компакт контекста**

Способ 1: линейный контекст

Сделай рефакторинг класса Program

Способ 1: линейный контекст

Сделай рефакторинг класса Program

Загружаю содержимое файла Program.java в контекст (lines 1-50 000)…

…

Способ 1: линейный контекст

Сделай рефакторинг класса Program

Загружаю содержимое файла Program.java в контекст (lines 1-50 000)…

…

✖ Контекст переполнен, лимиты на неделю закончились. Для продолжения работы купите ещё 🥰 🤗

Способ 2: линейный чанкинг

Загружаю содержимое файла Program.java в контекст (lines 1-500)

Загружаю содержимое файла Program.java в контекст (lines 501-1000)

Способ 2: линейный чанкинг

Загружаю содержимое файла Program.java в контекст (lines 1-500)

Загружаю содержимое файла Program.java в контекст (lines 501-1000)

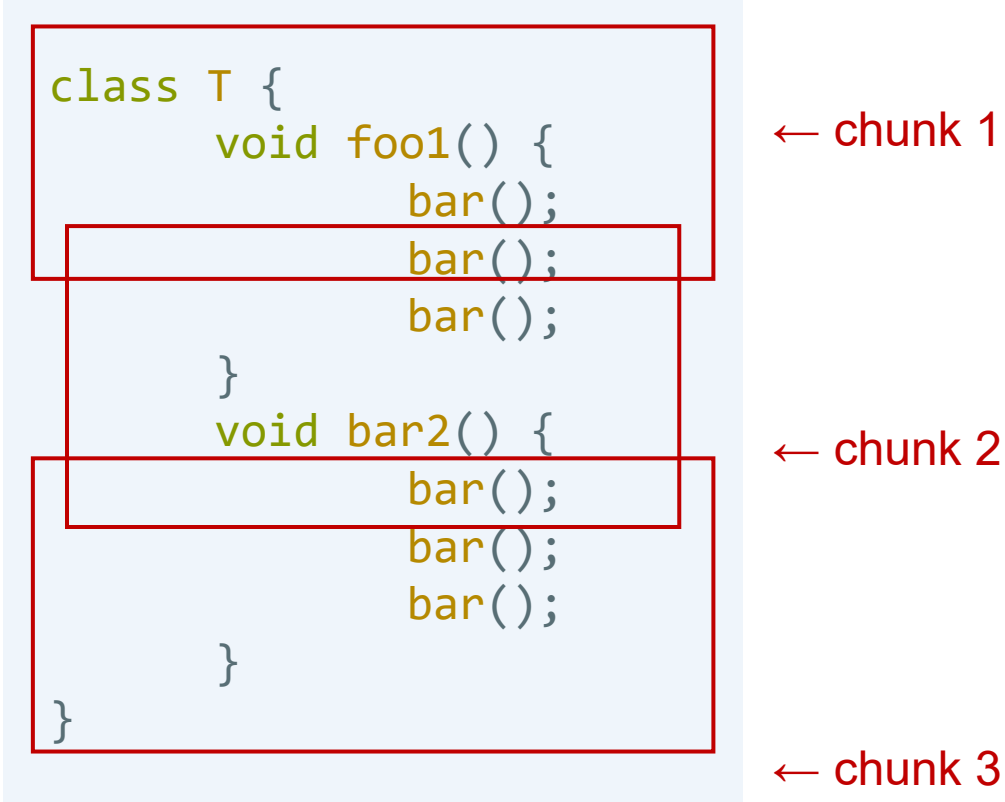
Нужен метод doWork. Поиск по директории…

Загружаю содержимое файла Program.java в контекст (lines 2030-2500)

Способ 2: линейный чанкинг

```
class T {
        void foo1() {
                bar();
                bar();
                bar();
        }
        void bar2() {
                bar();
                bar();
                bar();
        }
}
```

Способ 2: линейный чанкинг

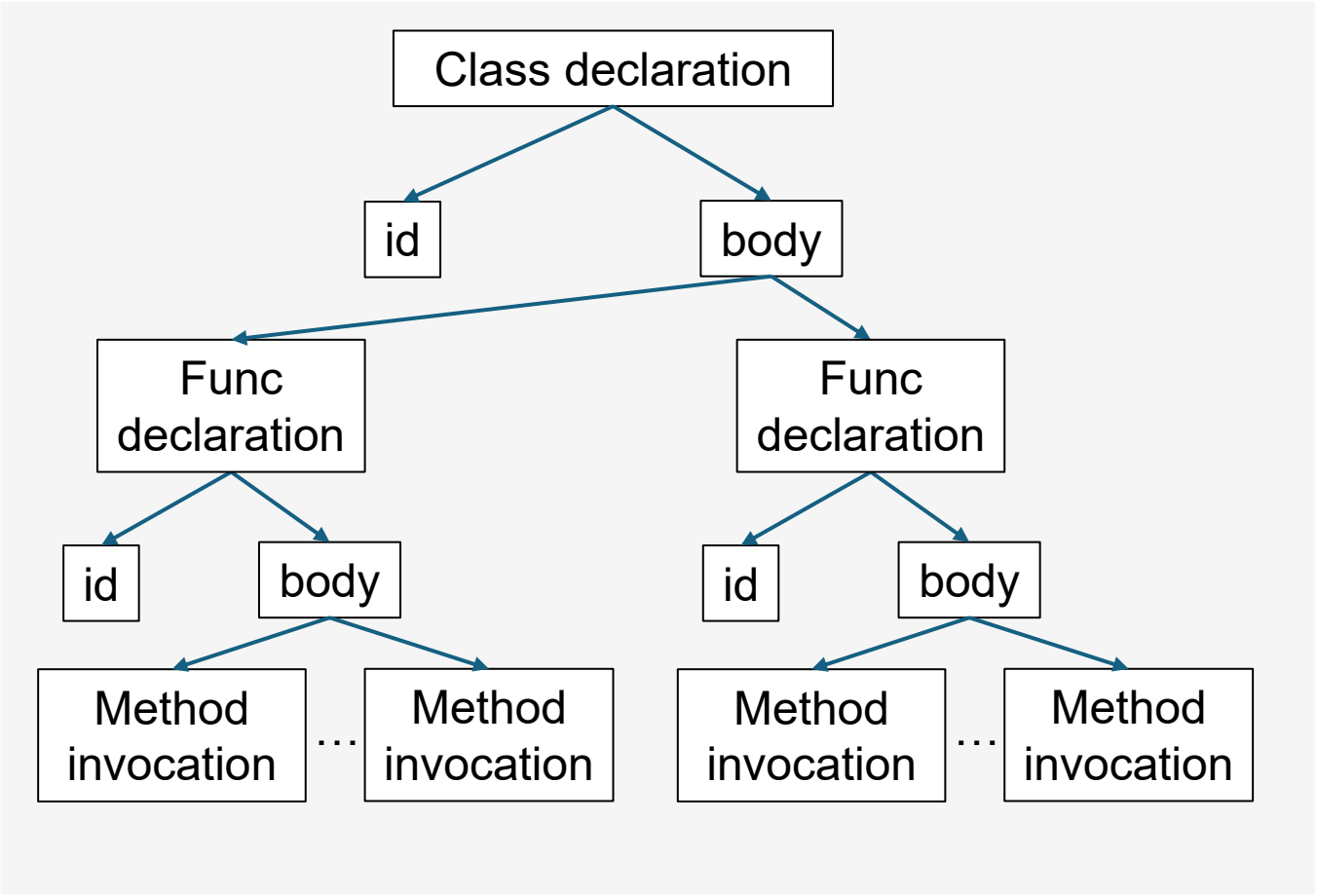


Способ 3: чанкинг по AST

```
class T {
        void foo1() {
                bar();
                bar();
                bar();
        }
        void bar2() {
                bar();
                bar();
                bar();
        }
}
```

CAST: Enhancing Code Retrieval-Augmented Generation with Structural Chunking via Abstract Syntax Tree (Yilin Zhang and all, 2025) +(1-5)%

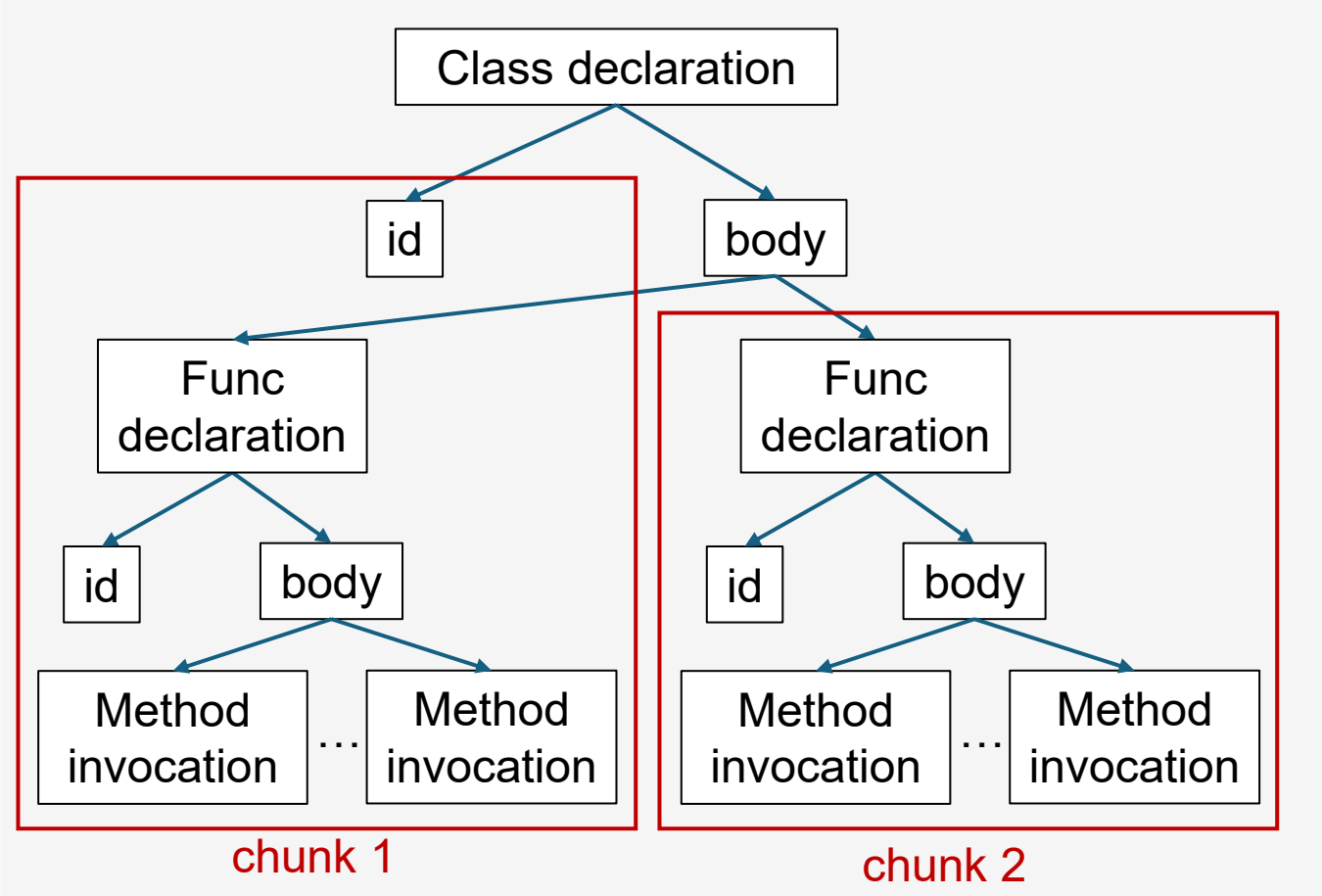
Способ 3: чанкинг по AST




```
class T {
        void foo1() {
                bar();
                bar();
                bar();
        }
        void bar2() {
                bar();
                bar();
                bar();
        }
}
```

CAST: Enhancing Code Retrieval-Augmented Generation with Structural Chunking via Abstract Syntax Tree (Yilin Zhang and all, 2025) +(1-5)%

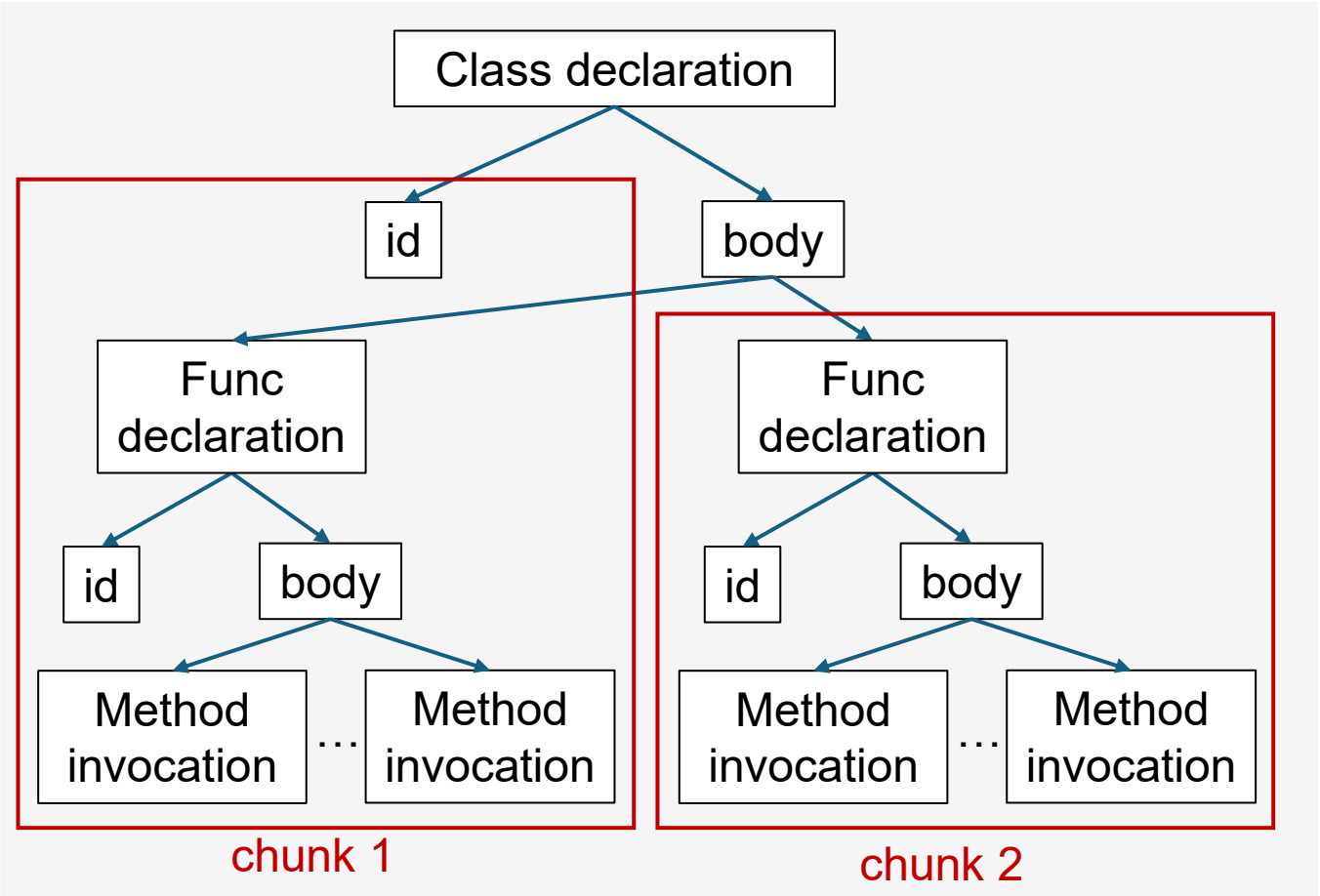
Способ 3: чанкинг по AST




```
class T {
        void foo1() {
                bar();
                bar();
                bar();
        }
        void bar2() {
                bar();
                bar();
                bar();
        }
}
```

CAST: Enhancing Code Retrieval-Augmented Generation with Structural Chunking via Abstract Syntax Tree (Yilin Zhang and all, 2025) +(1-5)%

Способ 3: чанкинг по AST




```
class T {
        void foo1() {
                bar();
                bar();
                bar();
        }
        void bar2() {
                bar();
                bar();
                bar();
        }
}
```

← chunk 1

← chunk 2

CAST: Enhancing Code Retrieval-Augmented Generation with Structural Chunking via Abstract Syntax Tree (Yilin Zhang and all, 2025) +(1-5)%

Пример – поиск методов класса

```
#LICENSE HEADER …
#a lot of imports…
class T {
        /**JAVADOCS**/
        void foo1() {
                if (x == y){
                        bar();
                }
                else {
                        bar();
                }
        }
        void bar2() {
        bar();
        bar(); }
 }
```

Surgical precision with AST-based code editing in Kiro (Feb 2026). Код иллюстративный!

Пример – поиск методов класса

Текстовый подход (1,309 tokens)

Весь файл – контекст →

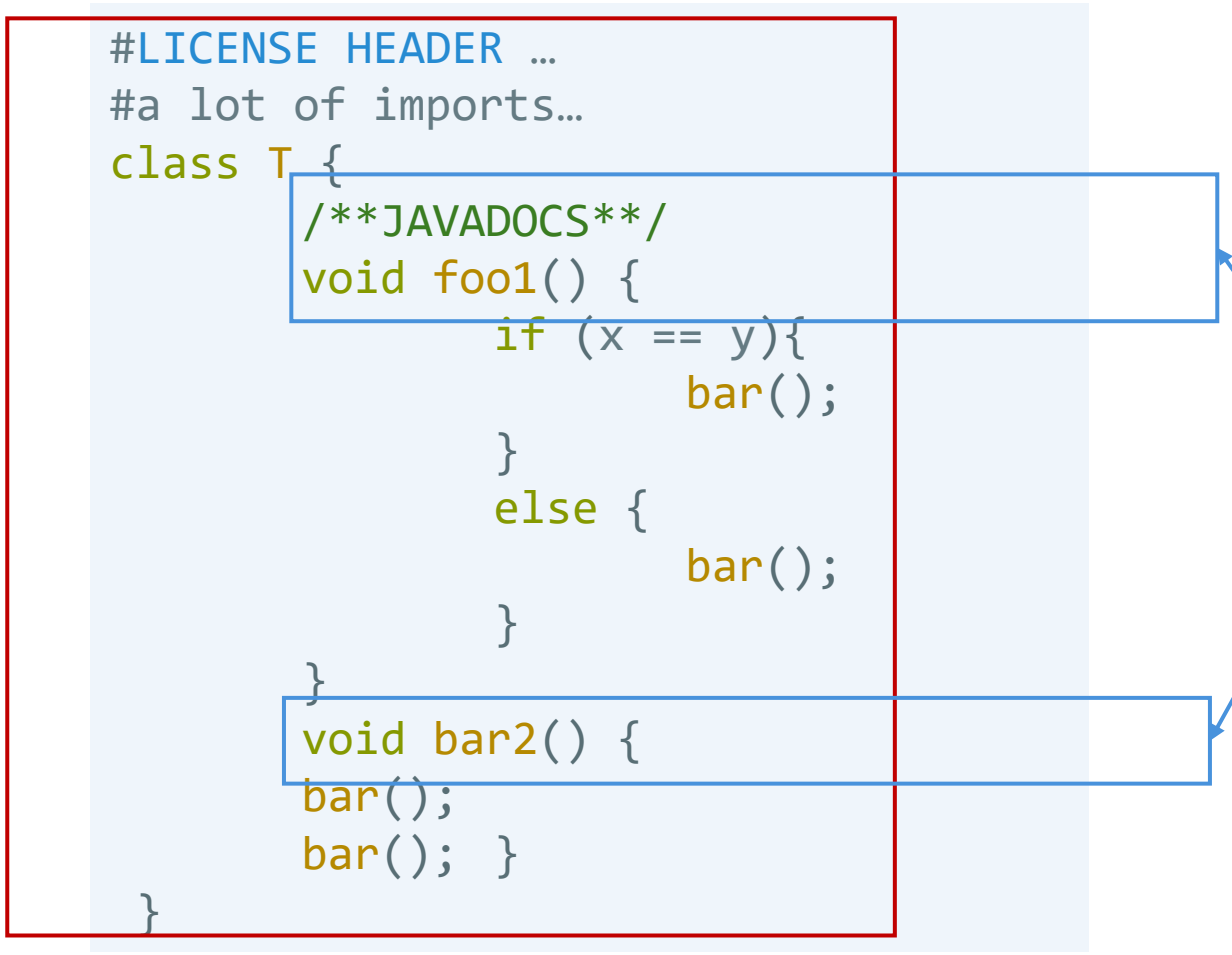
```
#LICENSE HEADER …
#a lot of imports…
class T {
        /**JAVADOCS**/
        void foo1() {
                if (x == y){
                        bar();
                }
                else {
                        bar();
                }
        }
        void bar2() {
        bar();
        bar(); }
 }
```

Surgical precision with AST-based code editing in Kiro (Feb 2026). Код иллюстративный!

Пример – поиск методов класса

Текстовый подход (1,309 tokens)

Весь файл – контекст →



AST-based подход (545 tokens)

Контекст == узлы нужного типа из AST

Surgical precision with AST-based code editing in Kiro (Feb 2026). Код иллюстративный!

Кодовый AI-агент

Решения на основе LLM

- Выбор инструментов
- Генерация кода/текста
- Планирование решения

…

Обвязке

- Валидация кода
- Сбор/компакт контекста
- **Изменение кода в проекте**

…

Обновление кода в проекте

Текстовый подход
(361 tokens)

replace (oldCode, newCode)

Surgical precision with AST-based code editing in Kiro (Feb 2026)

Обновление кода в проекте

Текстовый подход (361 tokens)

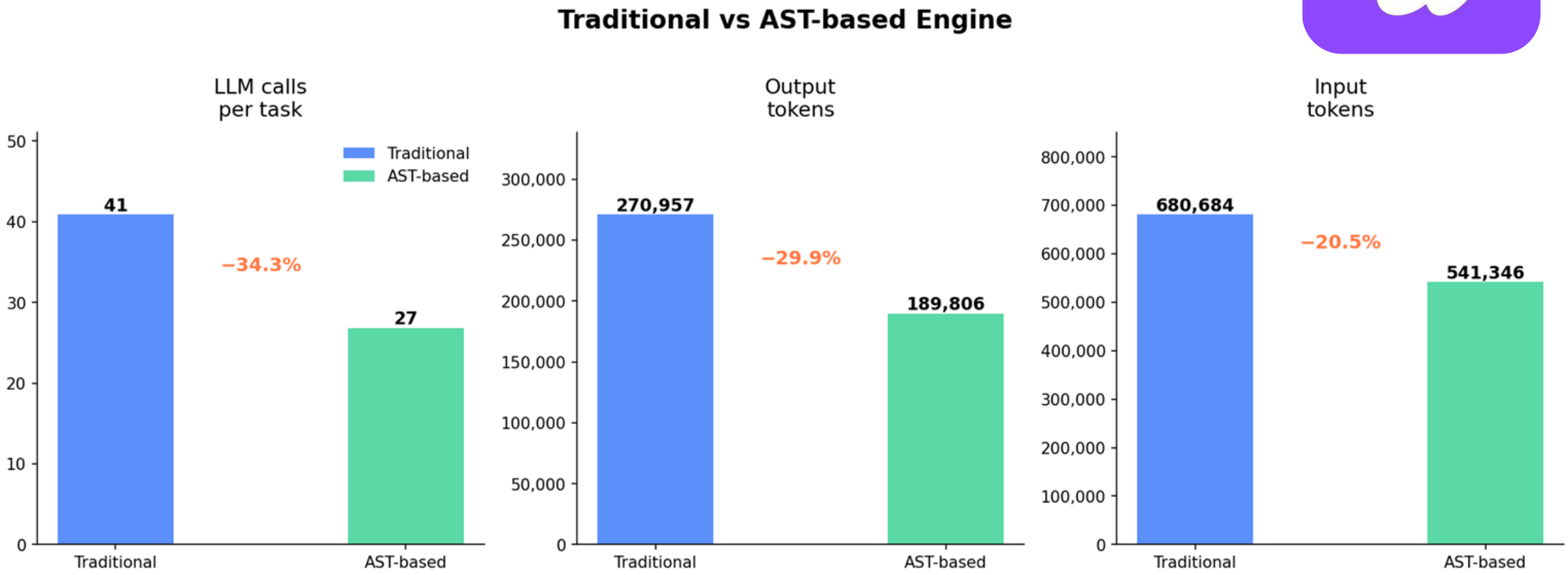
replace (oldCode, newCode)

AST-based подход (96 tokens)

replace(node, newCode)

Surgical precision with AST-based code editing in Kiro (Feb 2026)

Изменение кода в проекте



Surgical precision with AST-based code editing in Kiro (Feb 2026)

Вывод по блоку

AST экономит ресурсы AI агента:

- Сбор контекста (чанкинг)
- Поиск по проекту
- Проверка корректности
- Вставка кода в проект

Парсеры для java

Где и как искать парсер, если он понадобился для вашего проекта

Способы реализации парсеров

Написать вручную

Сгенерировать автоматически

Рукописные парсеры

```
if (code[i] == CLASS) {
  i++;
  return parseClass();
}
```

Рукописные парсеры

```
if (code[i] == CLASS) {
  i++;
  return parseClass();
}
if (code[i] == INTERFACE) {
  i++;
  return parseInterface();
```

Рукописные парсеры

```
if (code[i] == CLASS) {
  i++;
  return parseClass();
}
if (code[i] == INTERFACE) {
  i++;
  return parseInterface();
…
// ЯВНО ПОВТОРИТЬ ДЛЯ ВСЕХ
  КОНСТРУКЦИЙ В ЯЗЫКЕ
```

Рукописные парсеры

```
if (code[i] == CLASS) {
  i++;
  return parseClass();
}
if (code[i] == INTERFACE) {
  i++;
  return parseInterface();
…
// ЯВНО ПОВТОРИТЬ ДЛЯ ВСЕХ
  КОНСТРУКЦИЙ В ЯЗЫКЕ
```

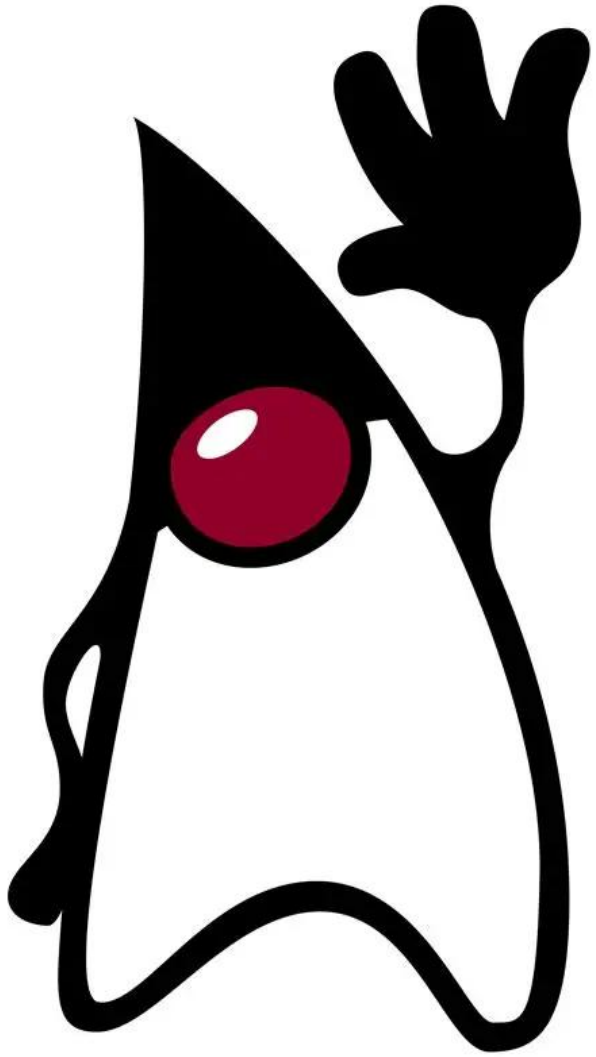
Существующие инструменты

- IntelliJ IDEA Parser
- Javac Parser
- Janino compiler Parser

JavacParser

- Часть OpenJDK (де-факто стандарт)
- Тесты
- Спецификации

OpenJDK



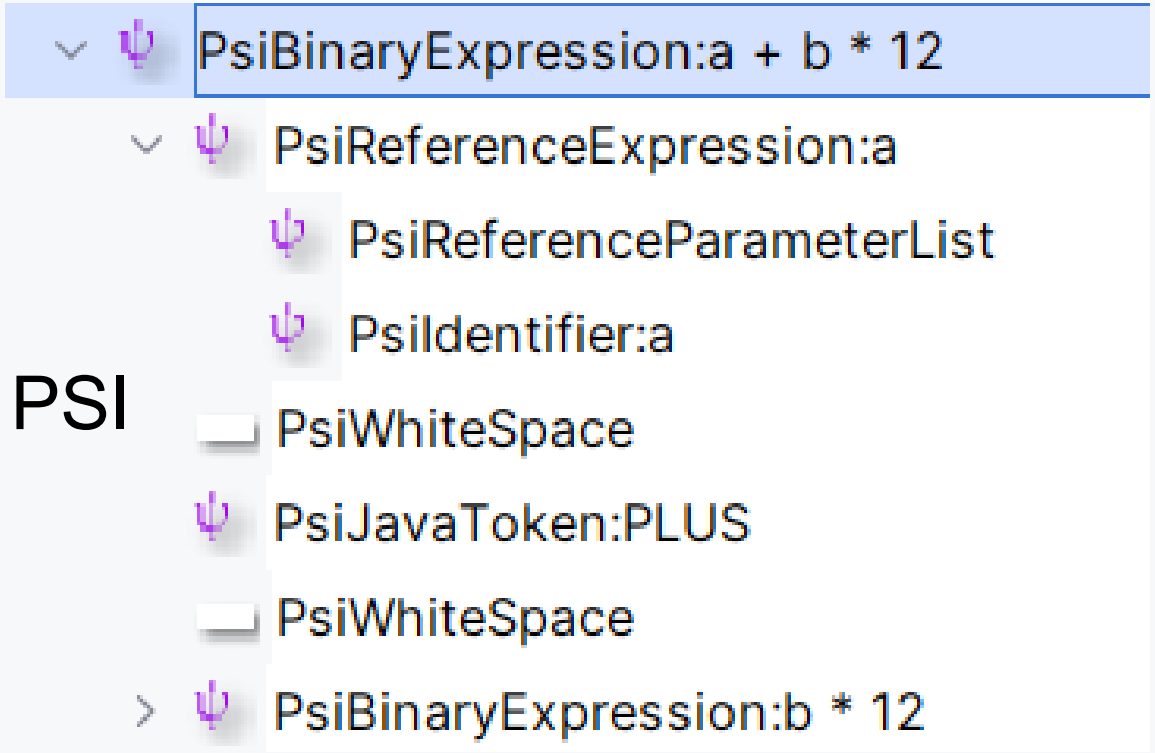
IntelliJIdea Java Parser

Тесно связан с IDEA

IntelliJIdea Java Parser

Тесно связан с IDEA

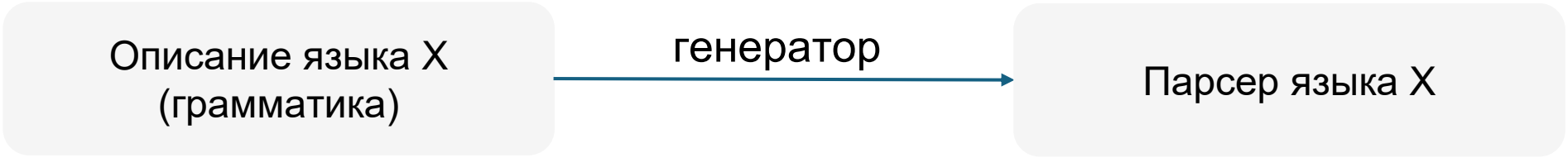
Богатое представление кода – PSI (Program Structure Interface)



Парсер-генераторы

Описание языка X
(грамматика)

Парсер-генераторы



Парсер-генераторы

- Tree-sitter
- ANTLR
- GrammarKit (from JB)
- Bison
- Jikespg (JDT)
- JavaCC (JavaParser)
- …

Tree-sitter: генератор

- Ядро на C
- «полное восстановление»

Tree-sitter: применение

- GitHub code navigation
- AI coding assistants
  - **Cursor**, **Roo Code**, **Aider**
- Code search / индексы для агентов

ANTLR

- Apache DSL's (Spark SQL, Hive)
- Elasticsearch Painless language
- Cypher language tooling

Jikespg (JDT)

- Eclipse IDE
- LS в VSCode Java extension (RedHat)
- Spring Tools (VSCode, Eclipse)
- LS в emax
- javalens-mcp

Вывод по блоку: популярные парсеры для Java

Рукописные

IntelliJ IDEA Parser

Javac Parser

На основе генераторов

ANTLR

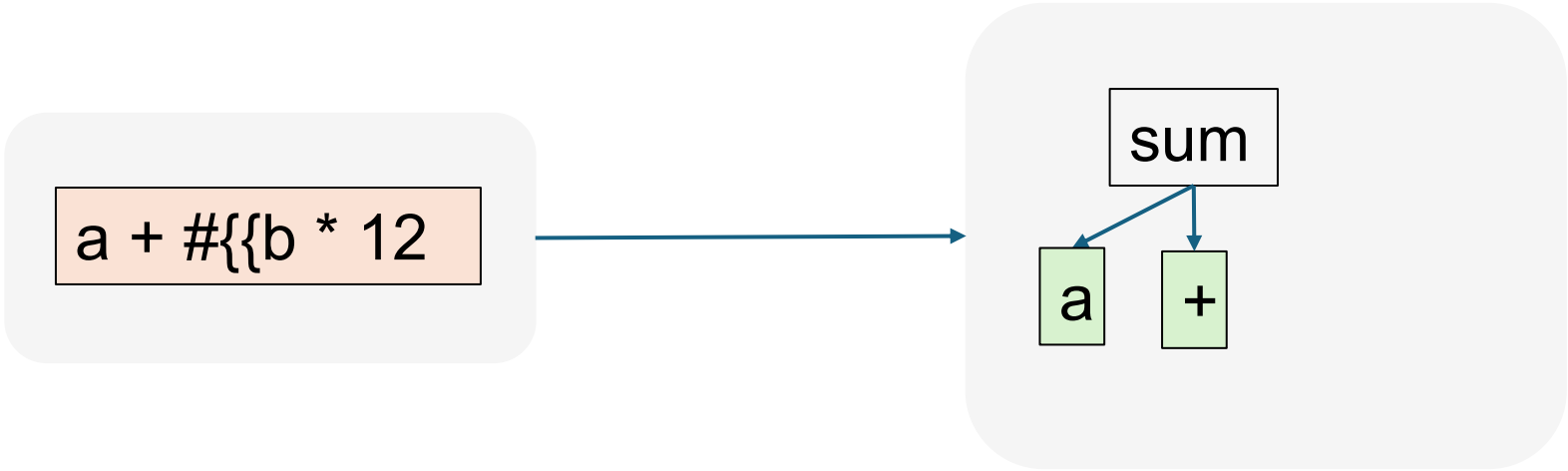
Tree-Sitter

Jikespg

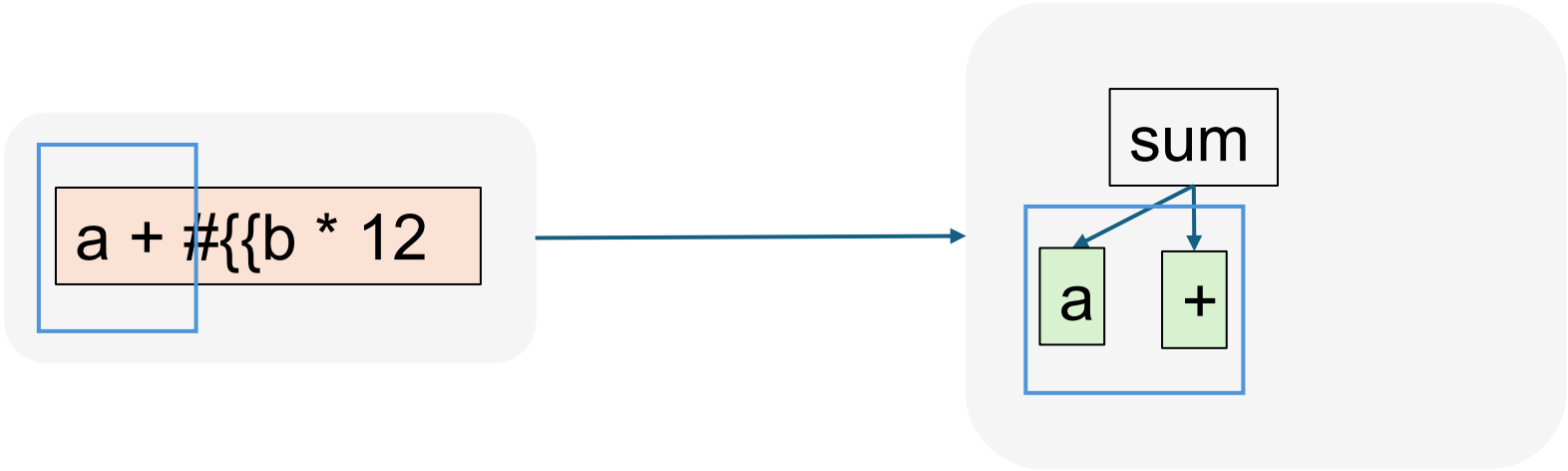
Виды восстановления

Как парсер строит дерево для сломанного кода

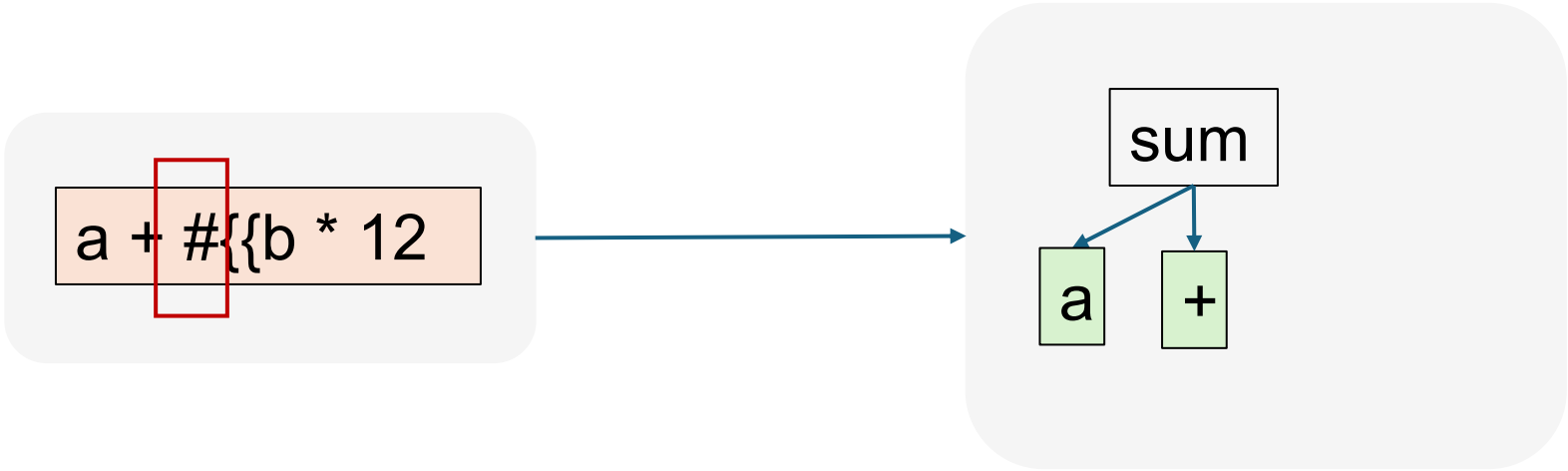
Способ 1: до первой ошибки



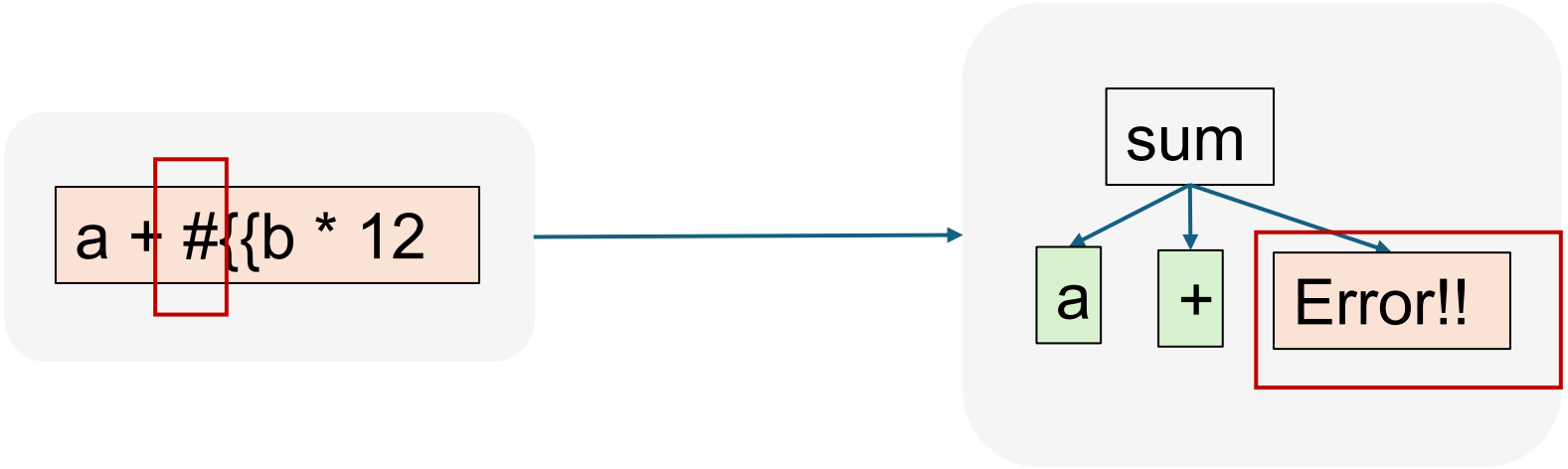
Способ 1: до первой ошибки



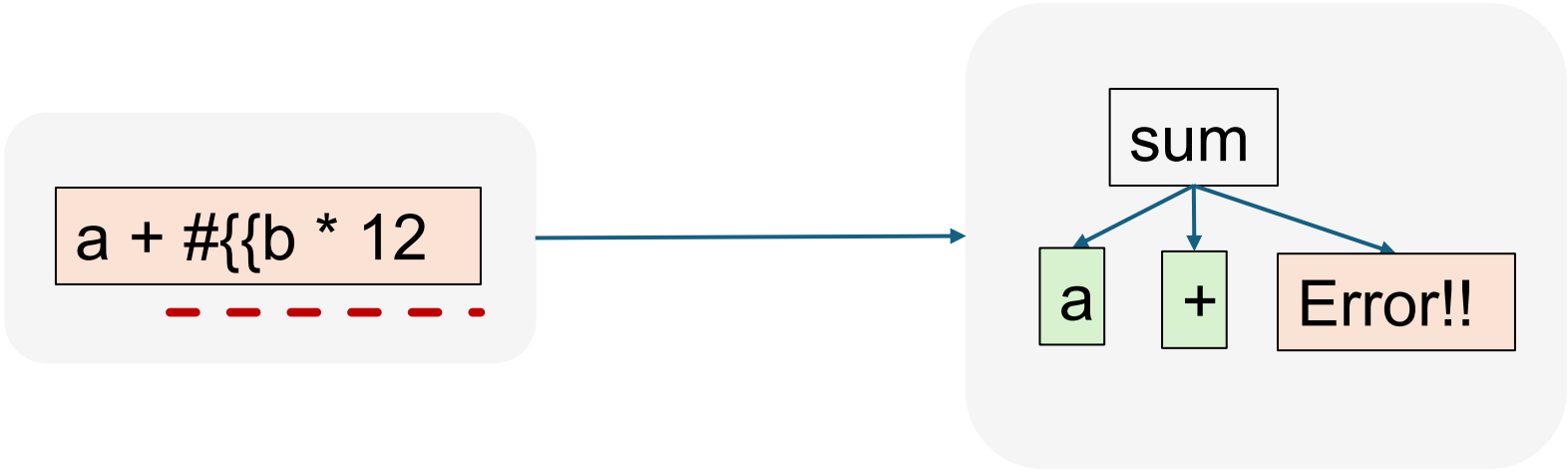
Способ 1: до первой ошибки



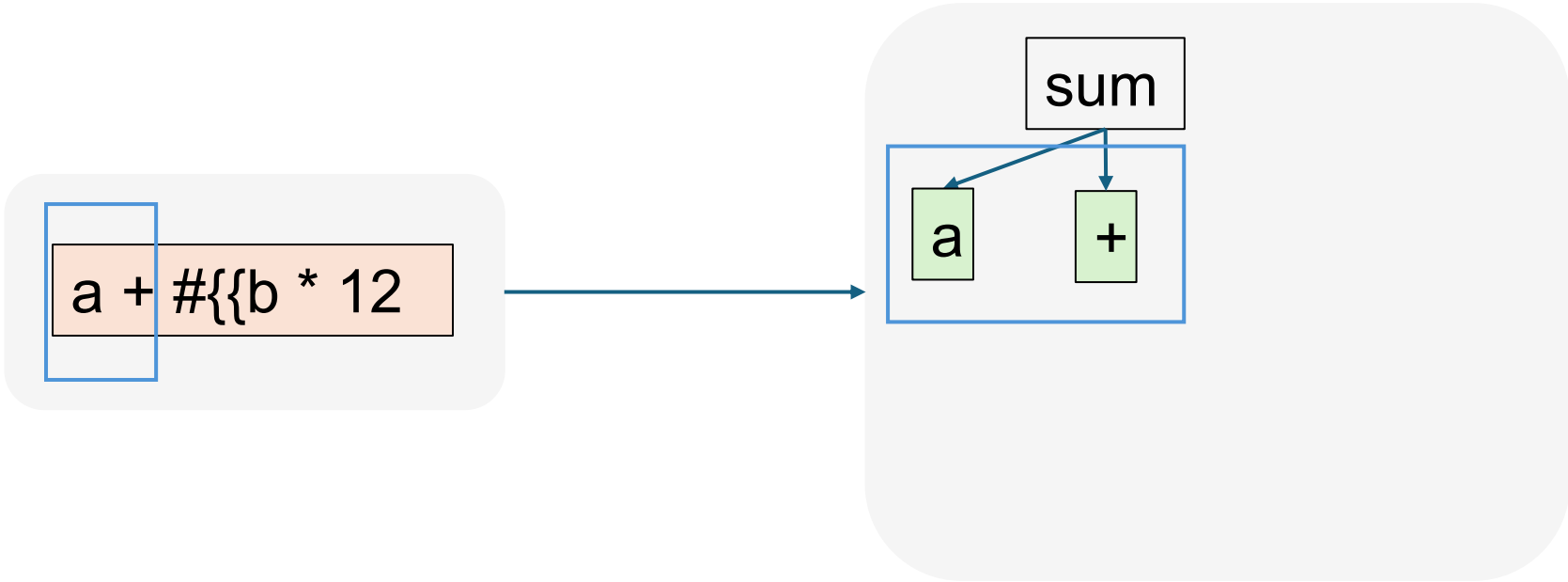
Способ 1: до первой ошибки



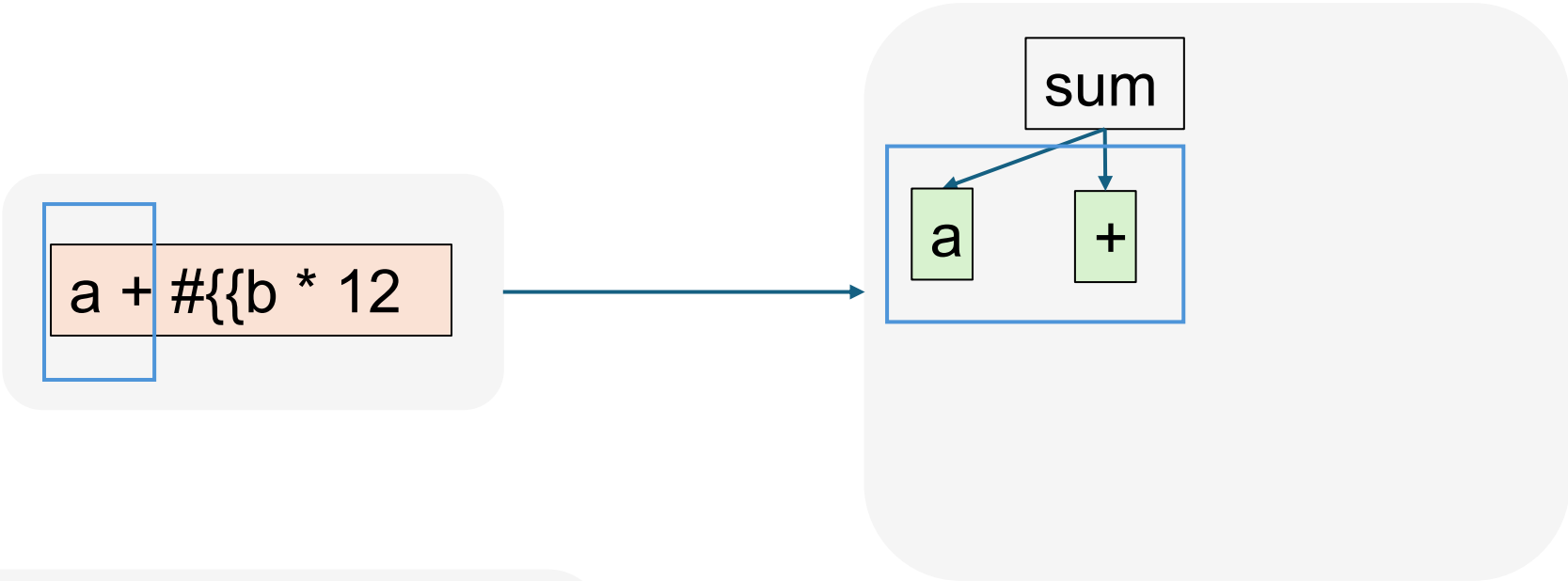
Способ 1: до первой ошибки



Способ 2: на эвристиках

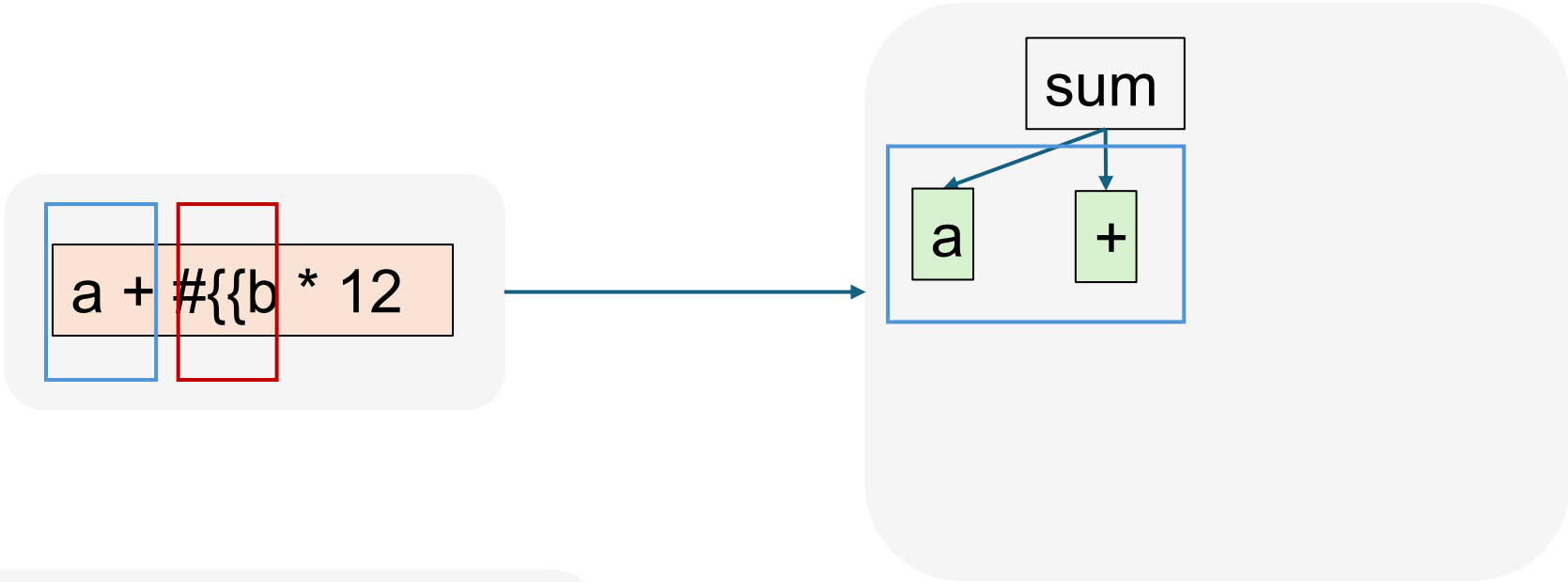


Способ 2: на эвристиках



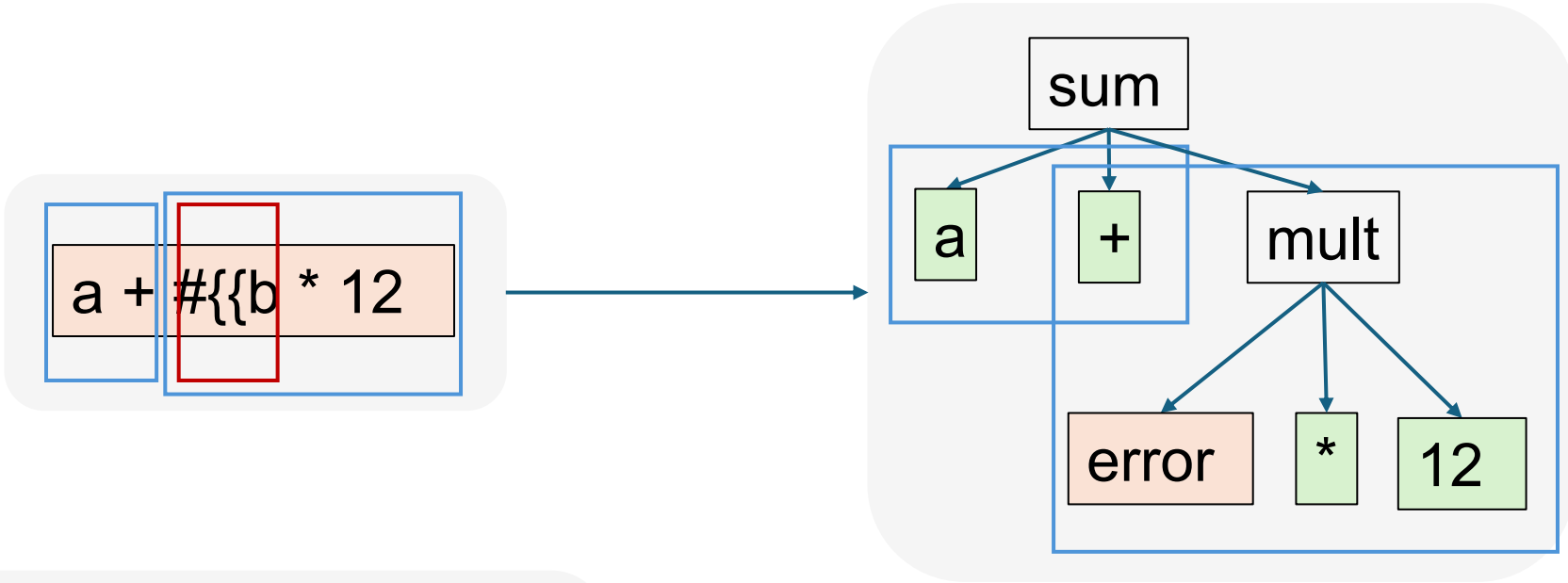

```
while(error) {
        skipAnyExcep(*, +, /, \n)
}
```

Способ 2: на эвристиках



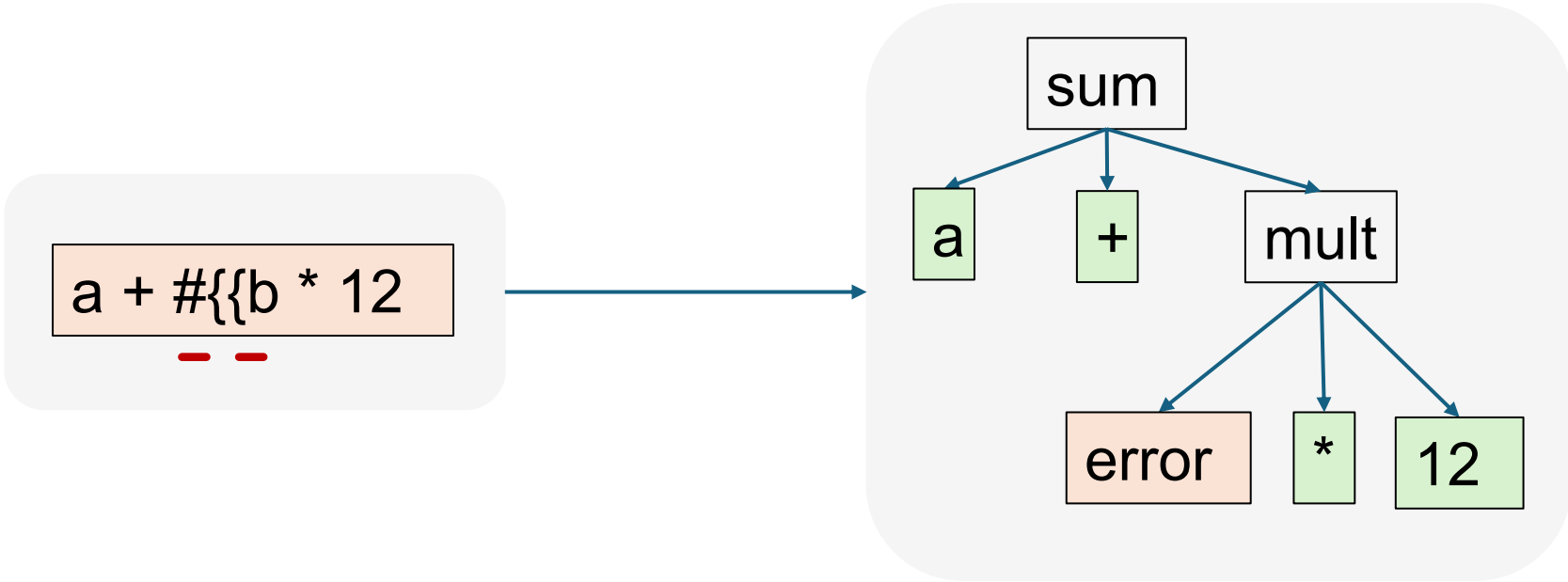

```
while(error) {
        skipAnyExcep(*, +, /, \n)
}
```

Способ 2: на эвристиках

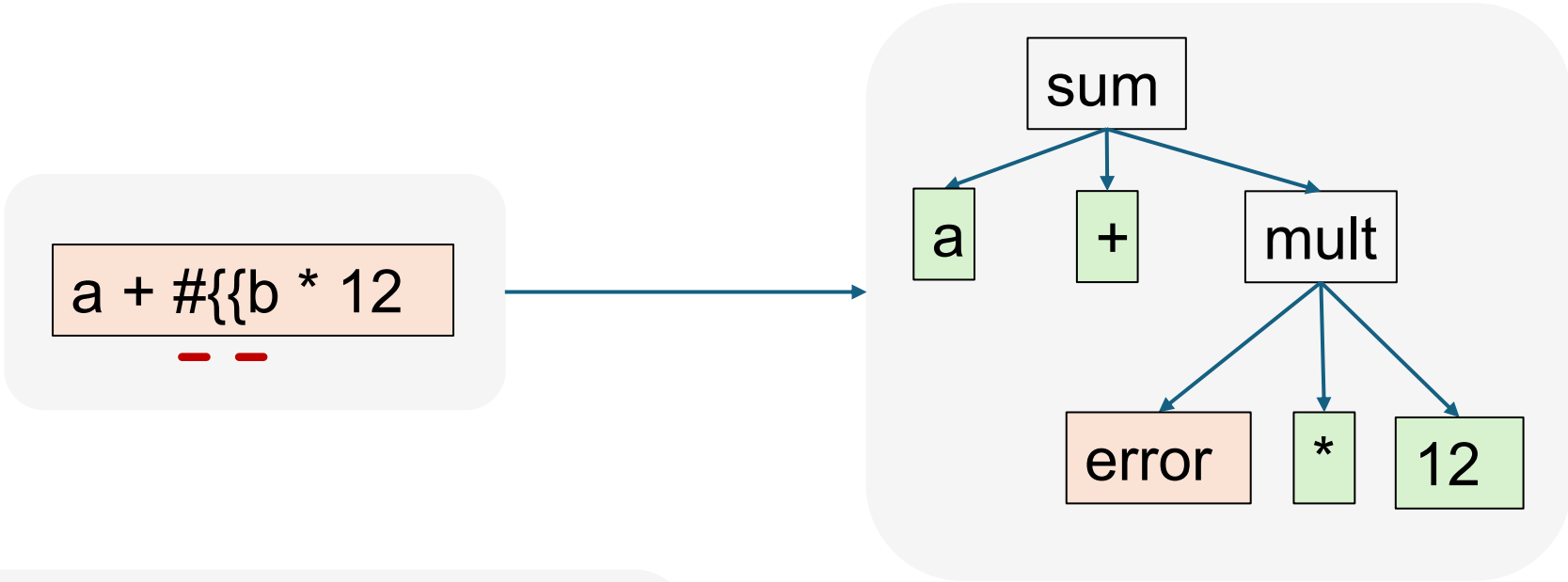



```
while(error) {
        skipAnyExcep(*, +, /, \n)
}
```

Способ 2: на эвристиках

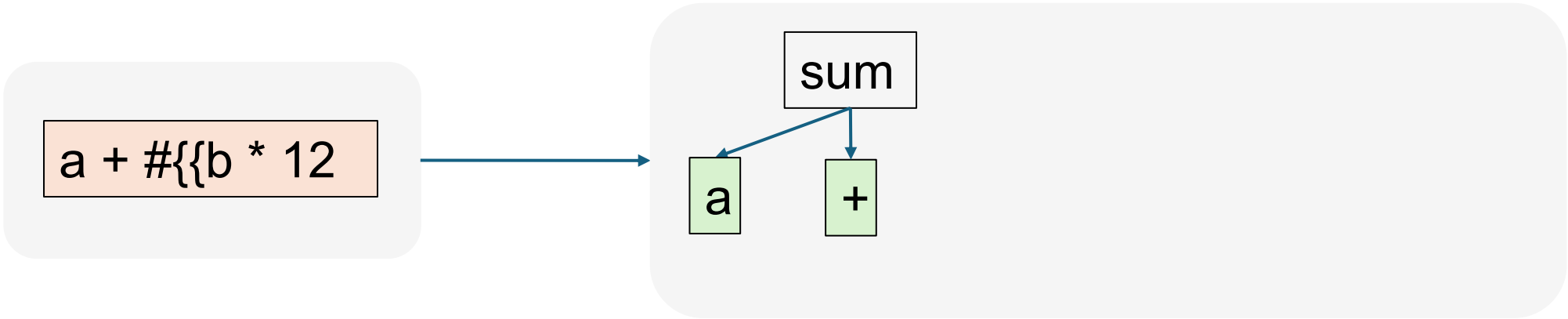


Способ 2: на эвристиках

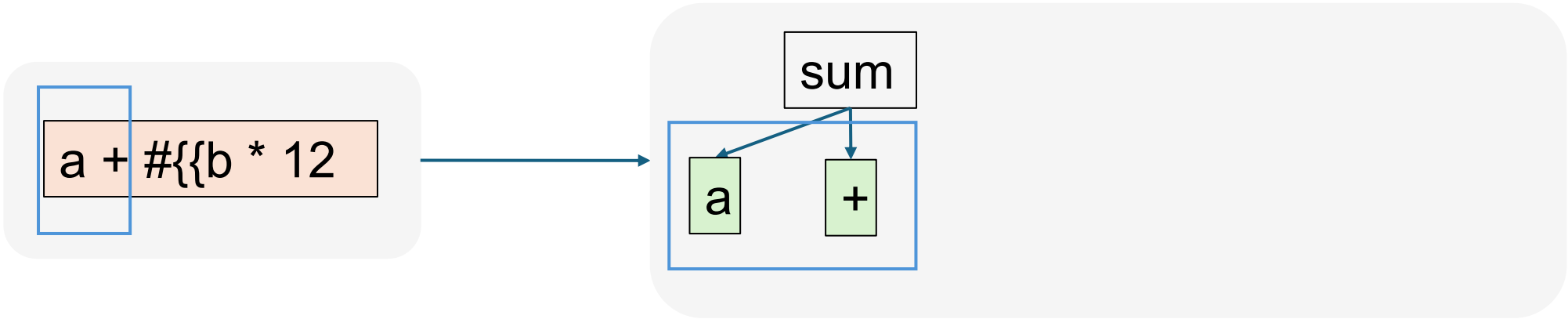


Рукописные парсеры
GrammarKit
ANTLR (опционально)

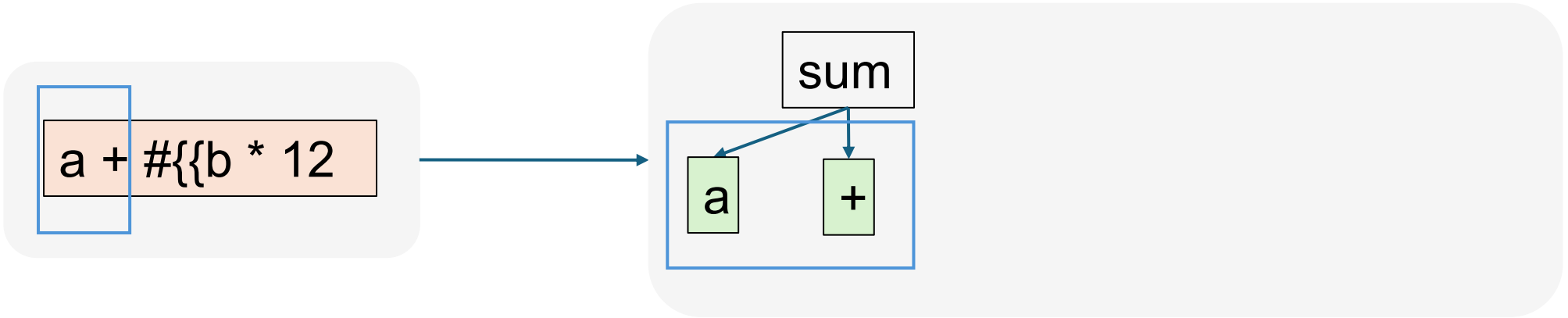
Способ 3: локальное восстановление



Способ 3: локальное восстановление

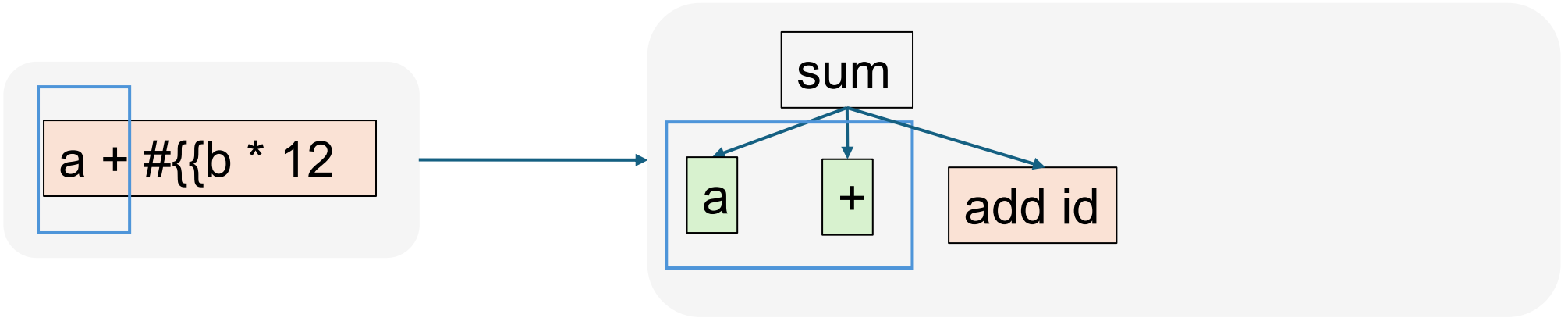


Способ 3: локальное восстановление



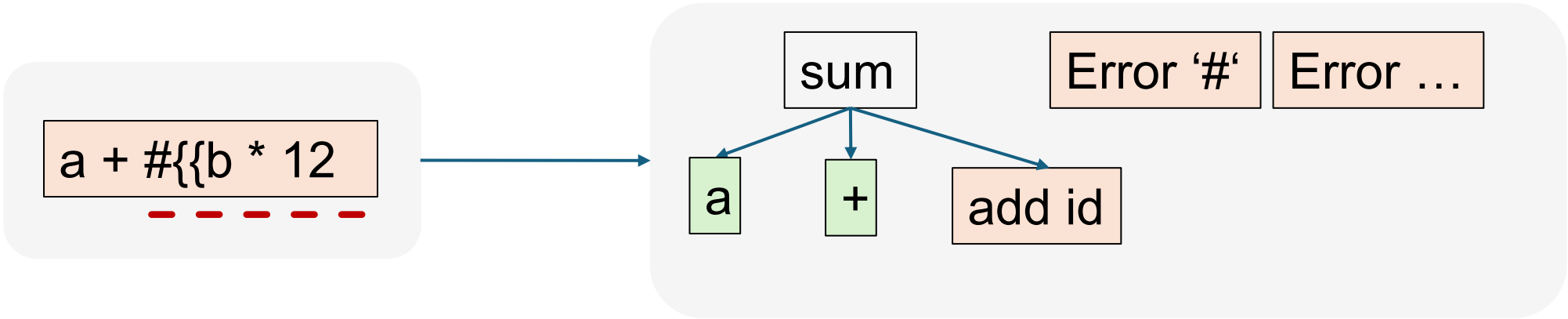

```
if (error) {
        delete() or add() or replace()
}
```

Способ 3: локальное восстановление

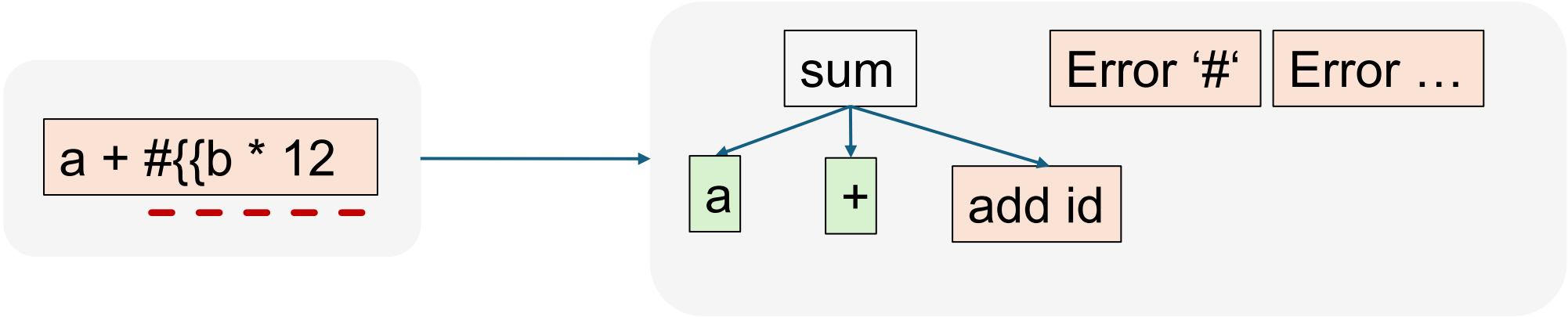



```
if (error) {
        delete() or add() or replace()
}
```

Способ 3: локальное восстановление

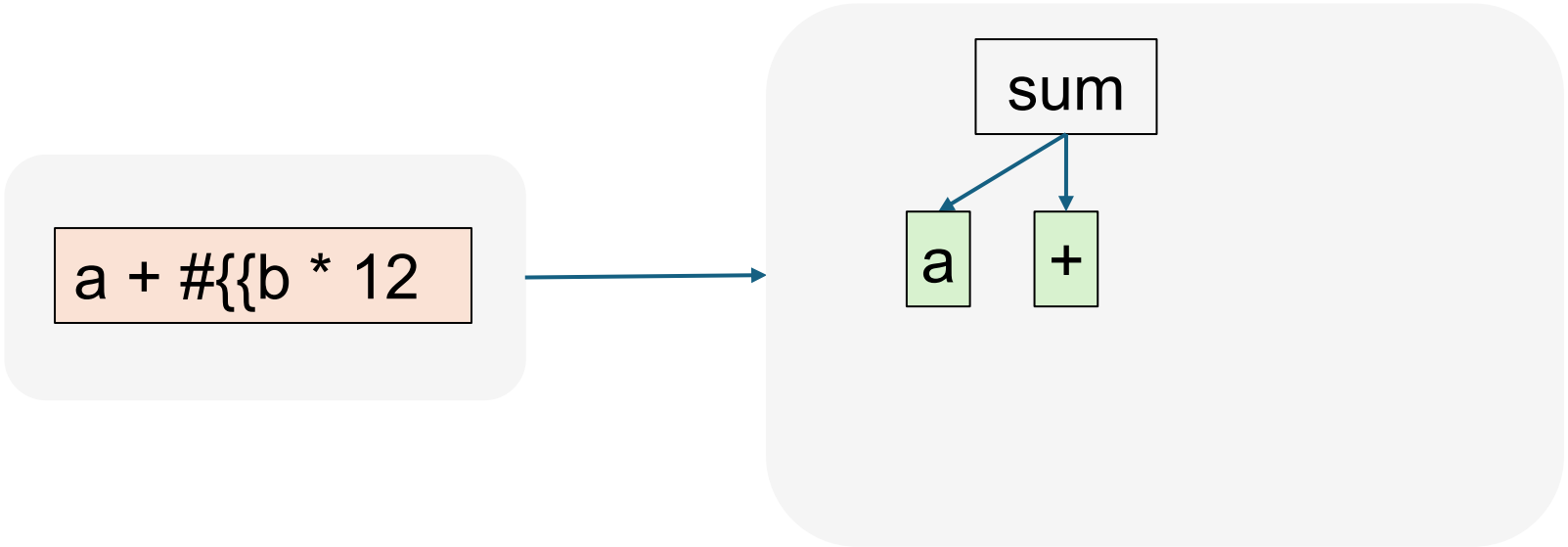


Способ 3: локальное восстановление

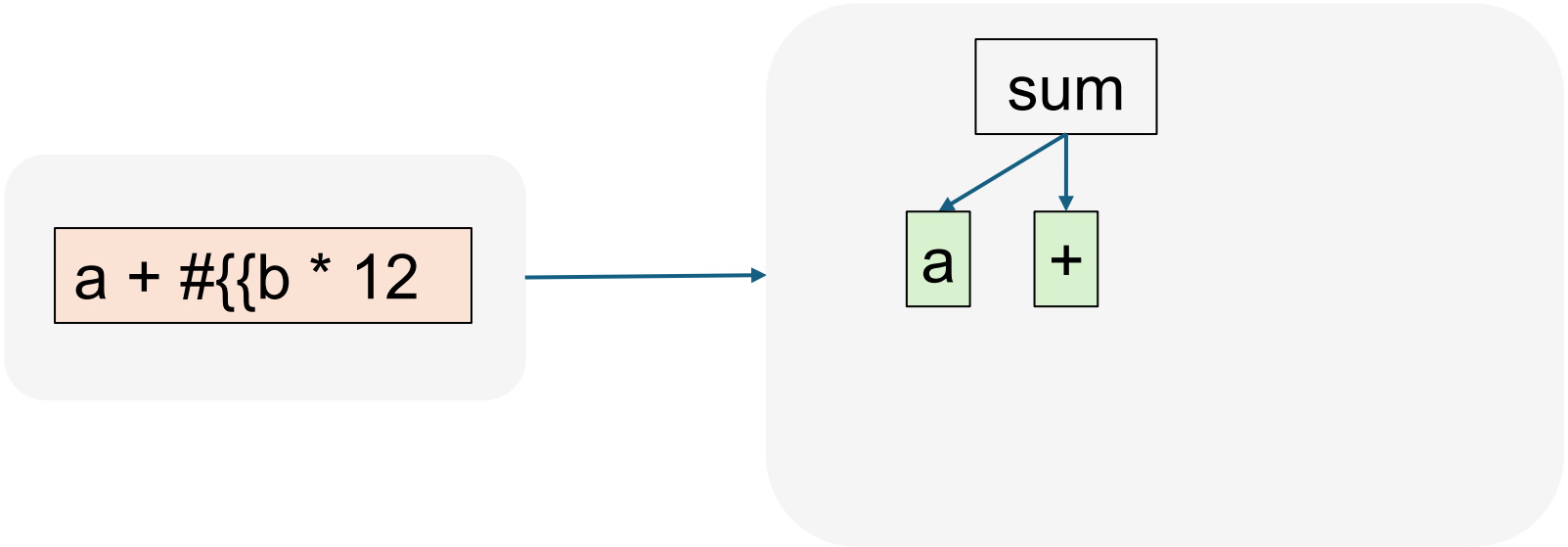


ANTLR
JDT

Способ 4: полное восстановление

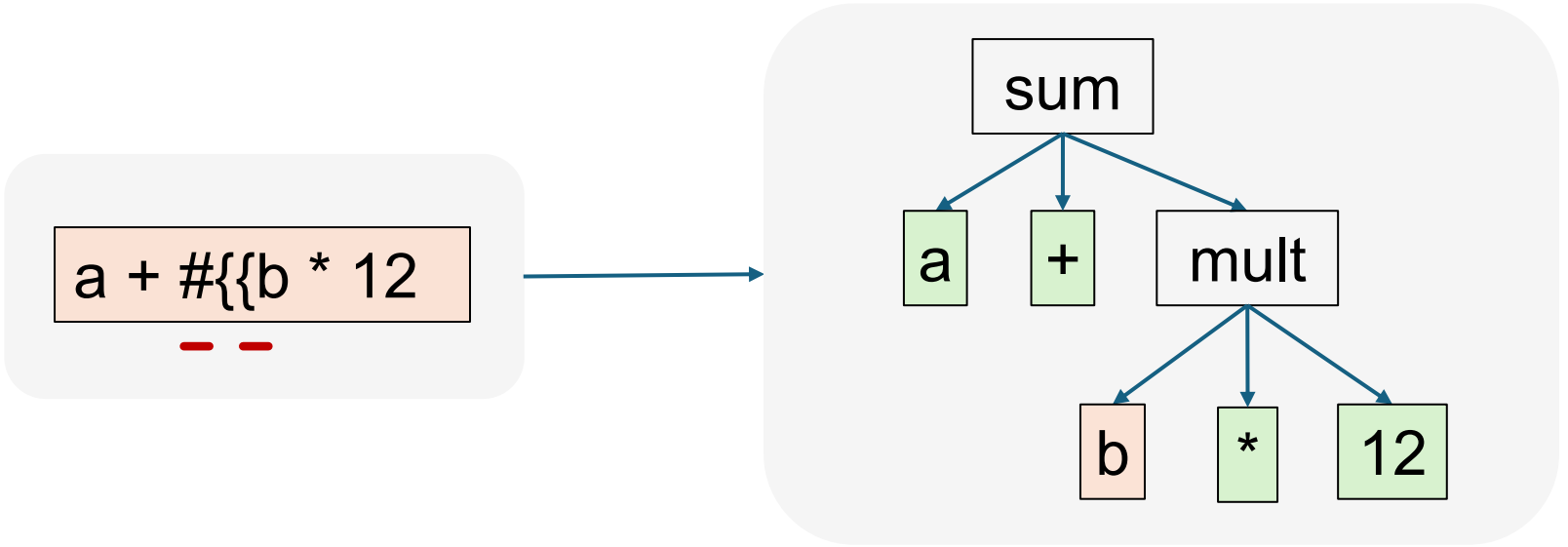


Способ 4: полное восстановление

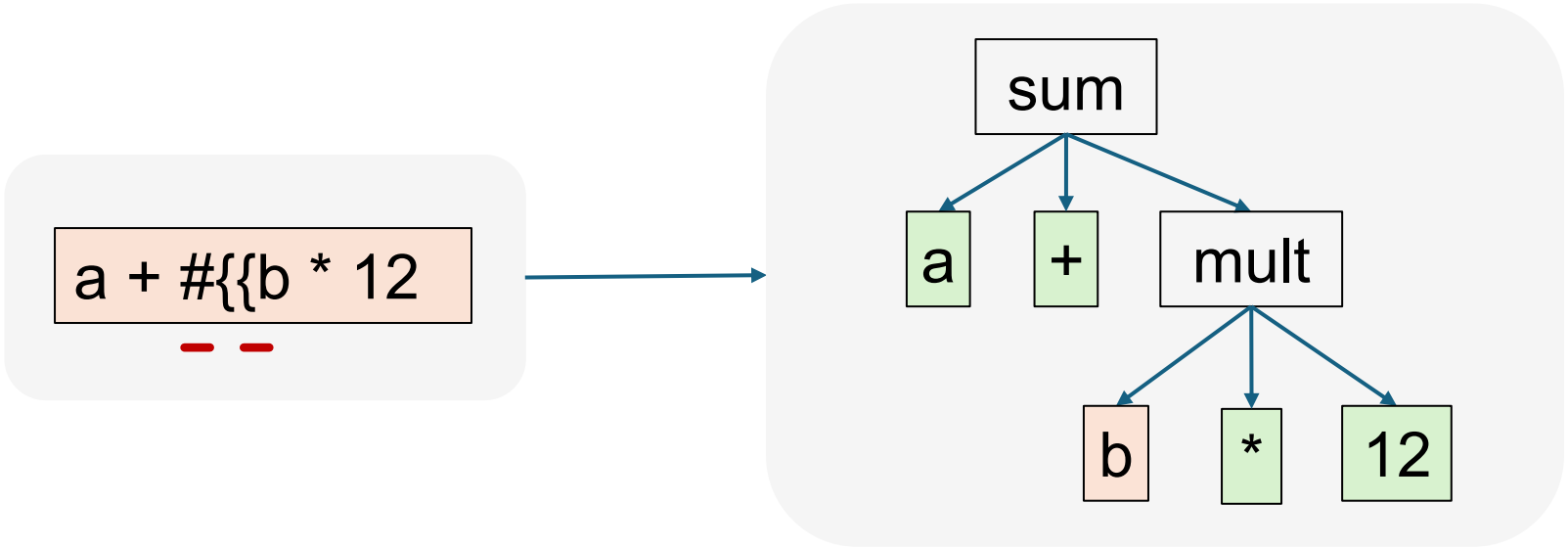



```
if (error) {
        findMin(delete() and add() and replace())
}
```

Способ 4: полное восстановление



Способ 4: полное восстановление



TreeSitter (??)

Вывод по блоку: способы error-recovery

- До первой ошибки
- На эвристиках
- Локальное
- Глобальное

Неудачные метрики качества

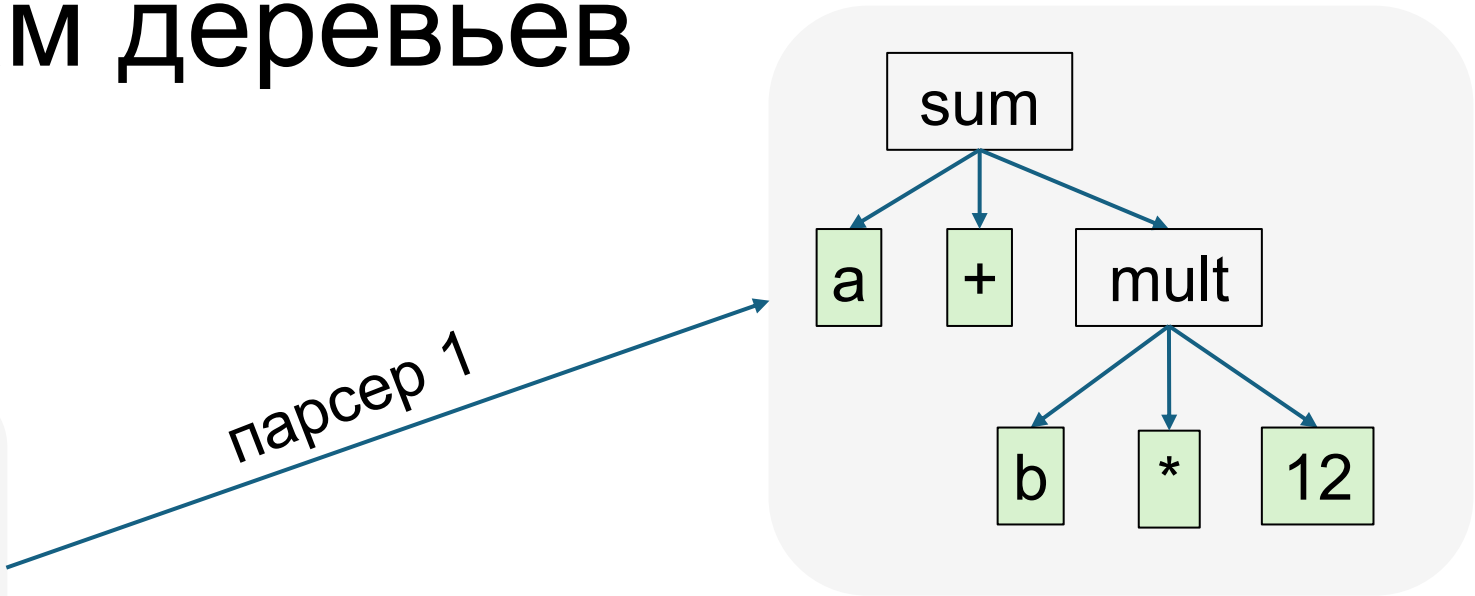
Как пытались измерять восстановление раньше
и почему ничего не получилось

1. Вручную

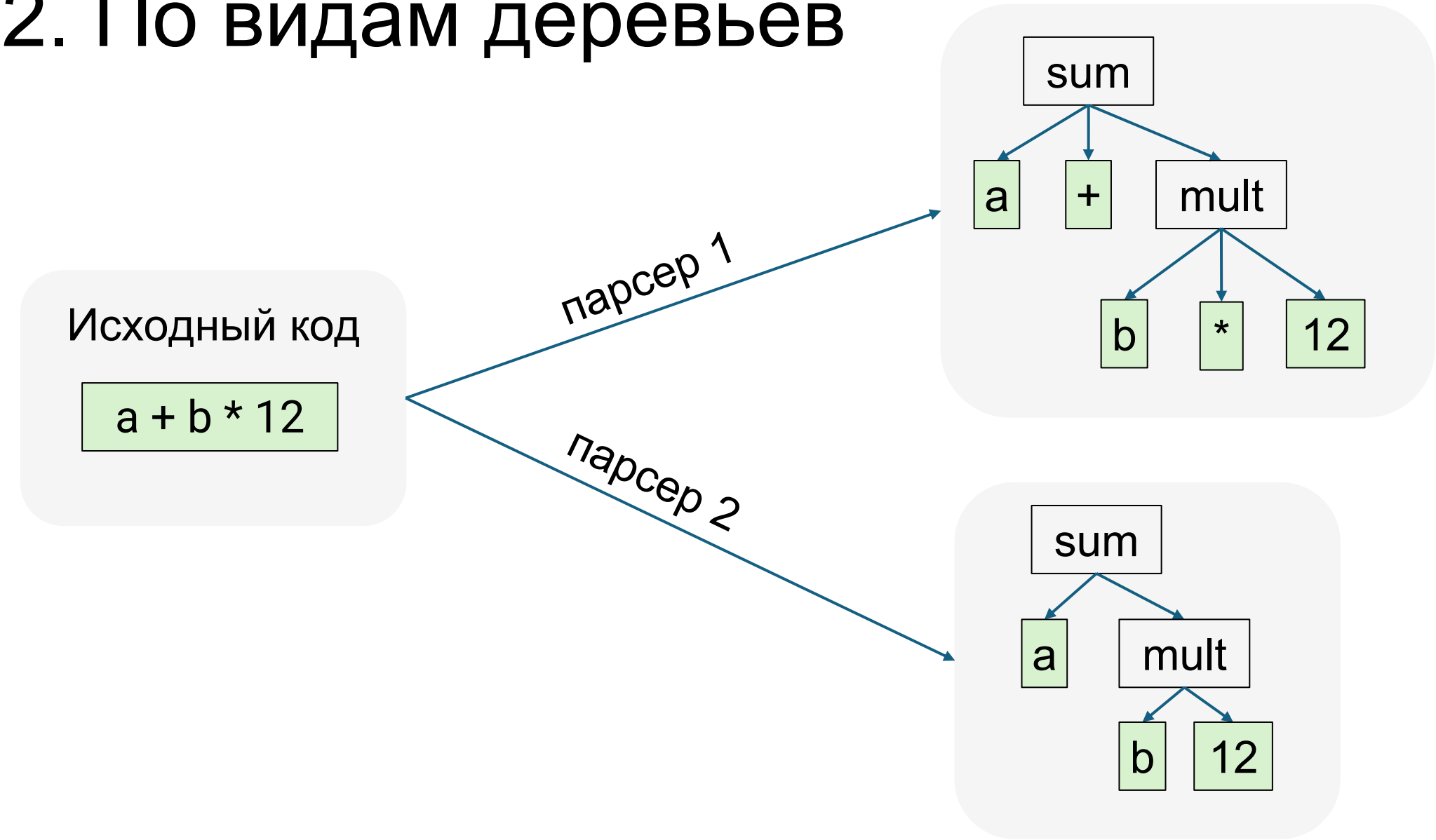
2. По видам деревьев

Исходный код

a + b * 12



2. По видам деревьев



3. Инструментирование алгоритма

4. Количество каскадных ошибок

```
class T {
    void f()
       System.out.println(“x");
      }
    void g() {}
}
```

❌ Ошибка: вы пропустили открывающую скобку во 2 строке.

❌ Ошибка: на строке 5 недопустимо использовать “void”

4. Количество каскадных ошибок

```
class T {
    void f()
       System.out.println(“x");
      }
    void g() {}
}
```

❌ Ошибка: вы пропустили открывающую скобку во 2 строке.

❌ Ошибка: на строке 5 недопустимо использовать “void”

```
class T {
    void f()
       System.out.println(“x");
      }
    void g() {}
}
```

Не вижу здесь ошибок ✨ ✨ ✨

Вывод по блоку: как НЕ стоит измерять качество error-recovery

- Вручную
- По деревьям
- Инструментированием алгоритма
- По количеству каскадных ошибок

Общие метрики качества

Что измерили и какие результаты получили

Данные для анализа

bluej.org

IDE для студентов

Данные для анализа

bluej.org

IDE для студентов

Логи для преподавателей

Данные для анализа

bluej.org

IDE для студентов

Логи для преподавателей

Выборка на java8

Парсеры для анализа (тоже для Java8)

Сгенерированные с помощью:

- Tree-Sitter

Парсеры для анализа (тоже для Java8)

Сгенерированные с помощью:

- Tree-Sitter
- ANTLR по спецификации Java8
- ANTLR по оптимизированной грамматике Java

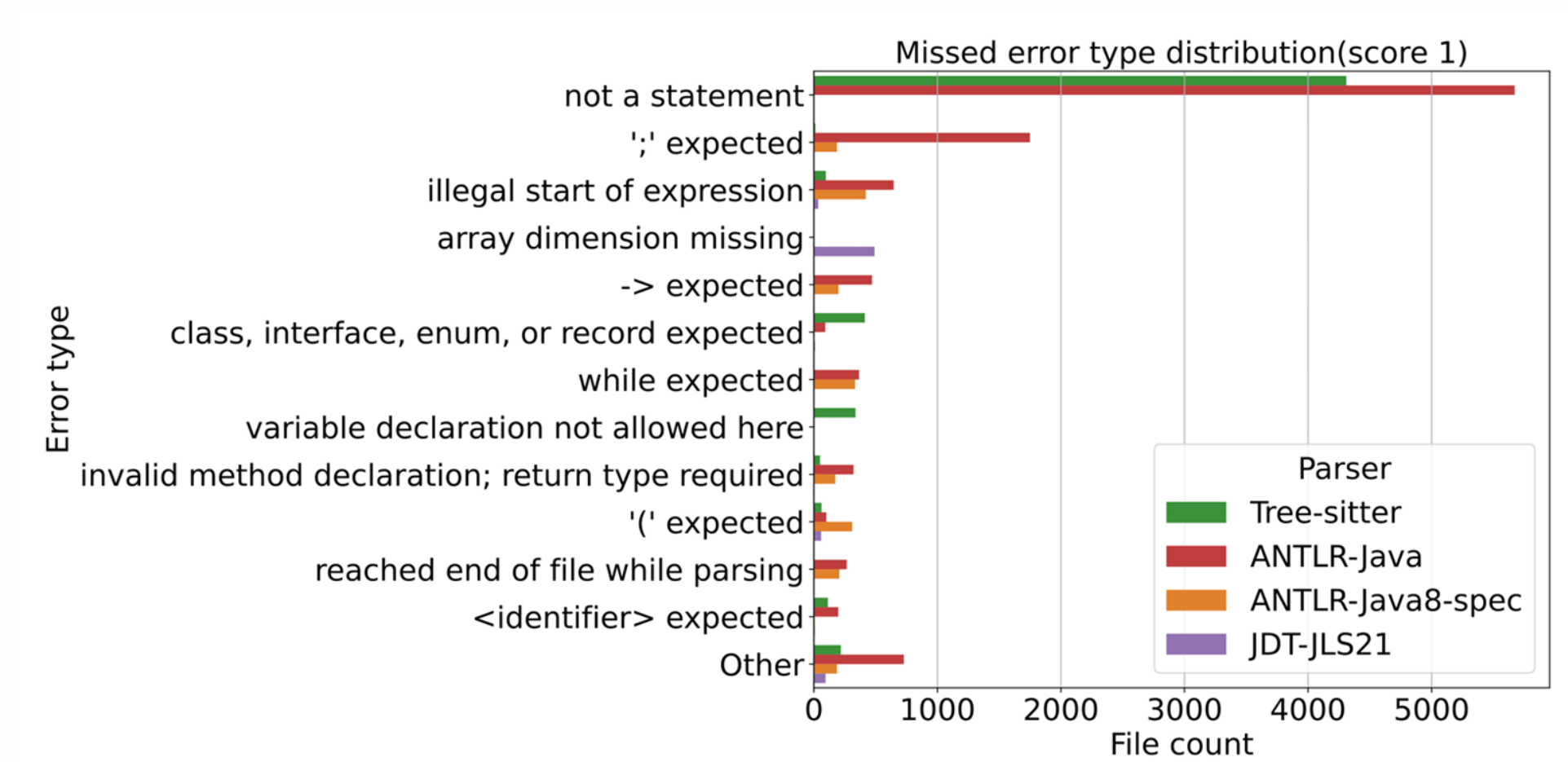
Парсеры для анализа (тоже для Java8)

Сгенерированные с помощью:

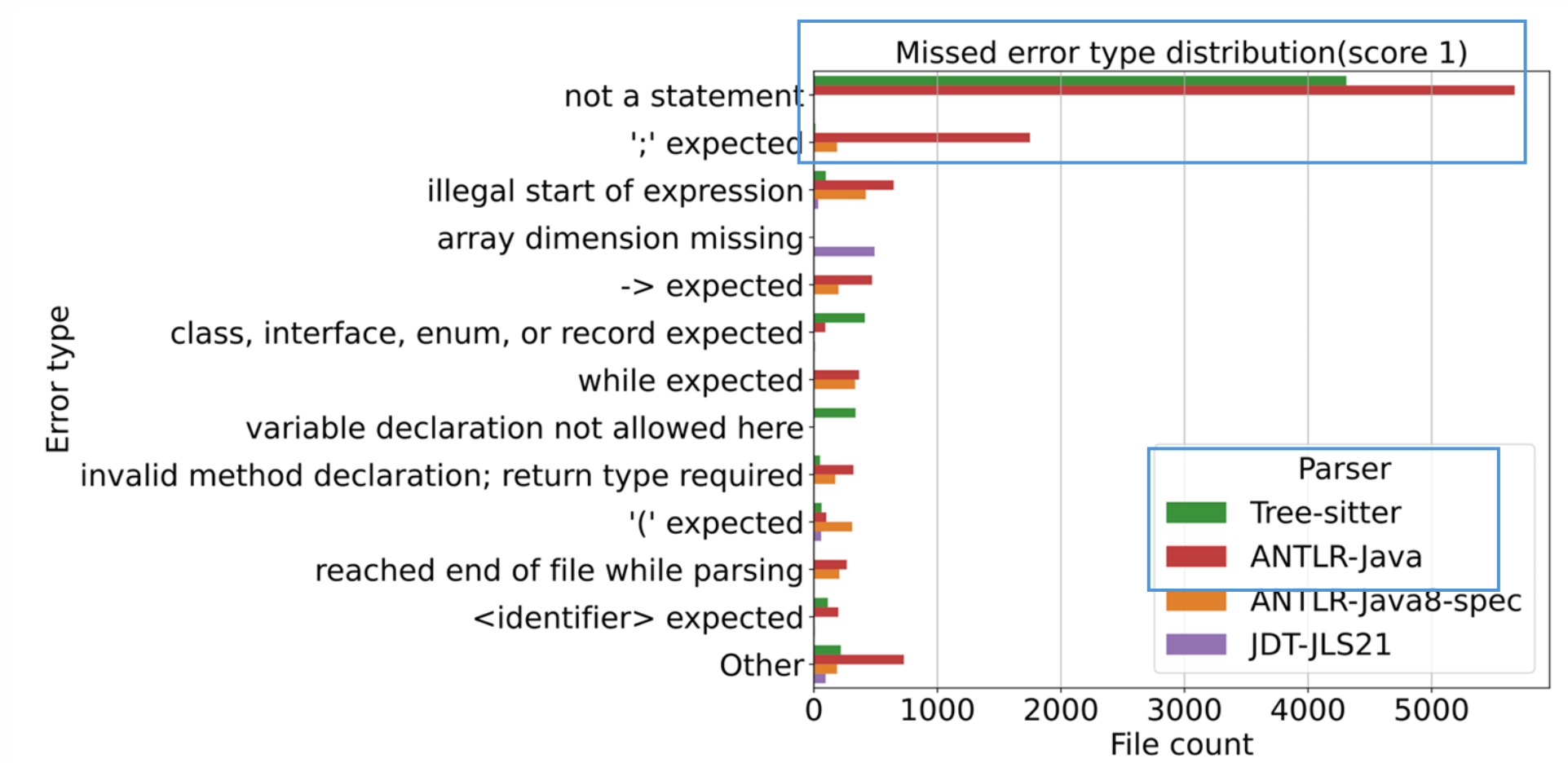
- Tree-Sitter
- ANTLR по спецификации Java8
- ANTLR по оптимизированной грамматике Java
- Jikespg (JDT)

Подход 1. Количество пропущенных ошибок

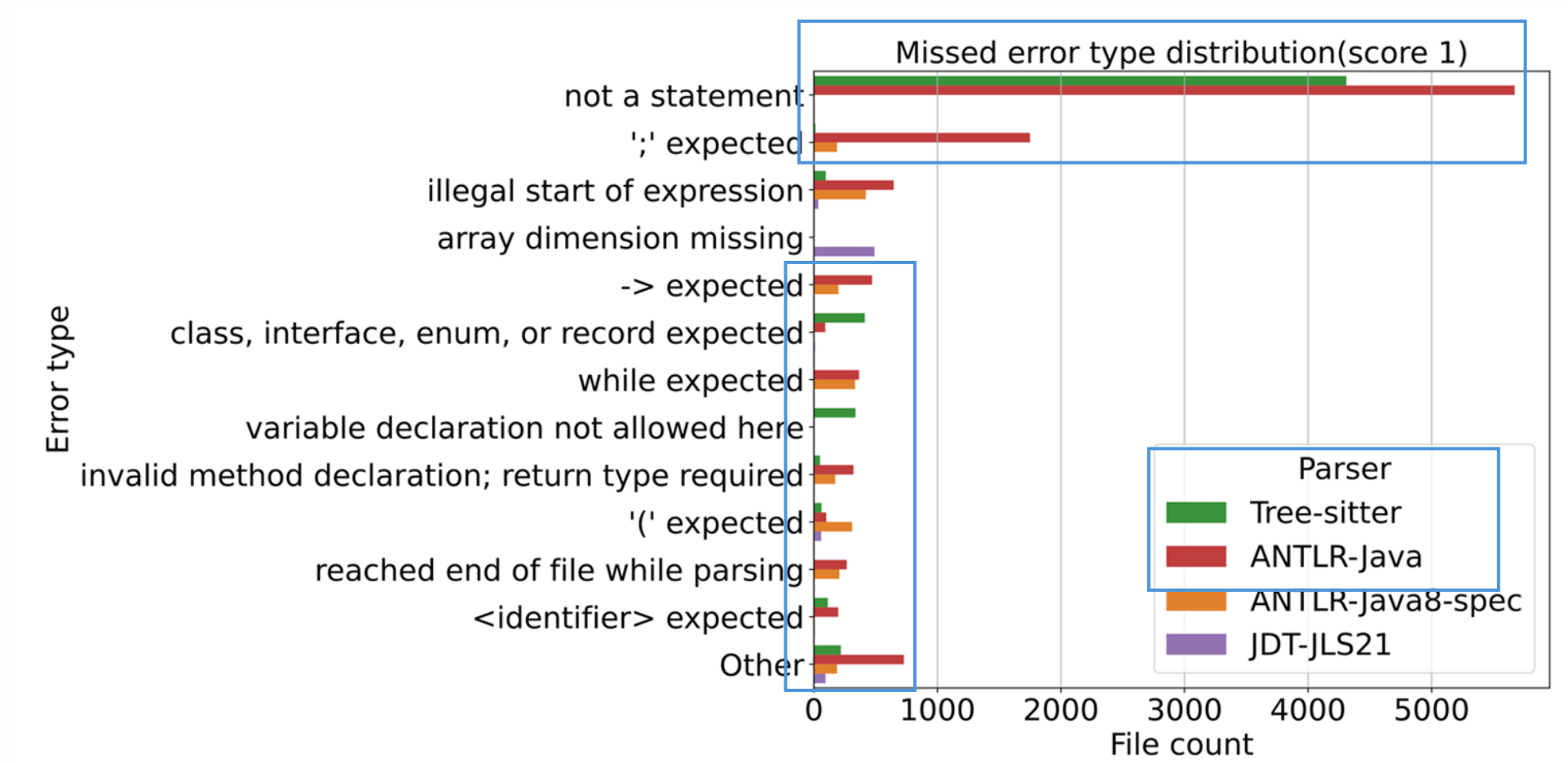
Подход 1. Количество пропущенных ошибок



Подход 1. Количество пропущенных ошибок

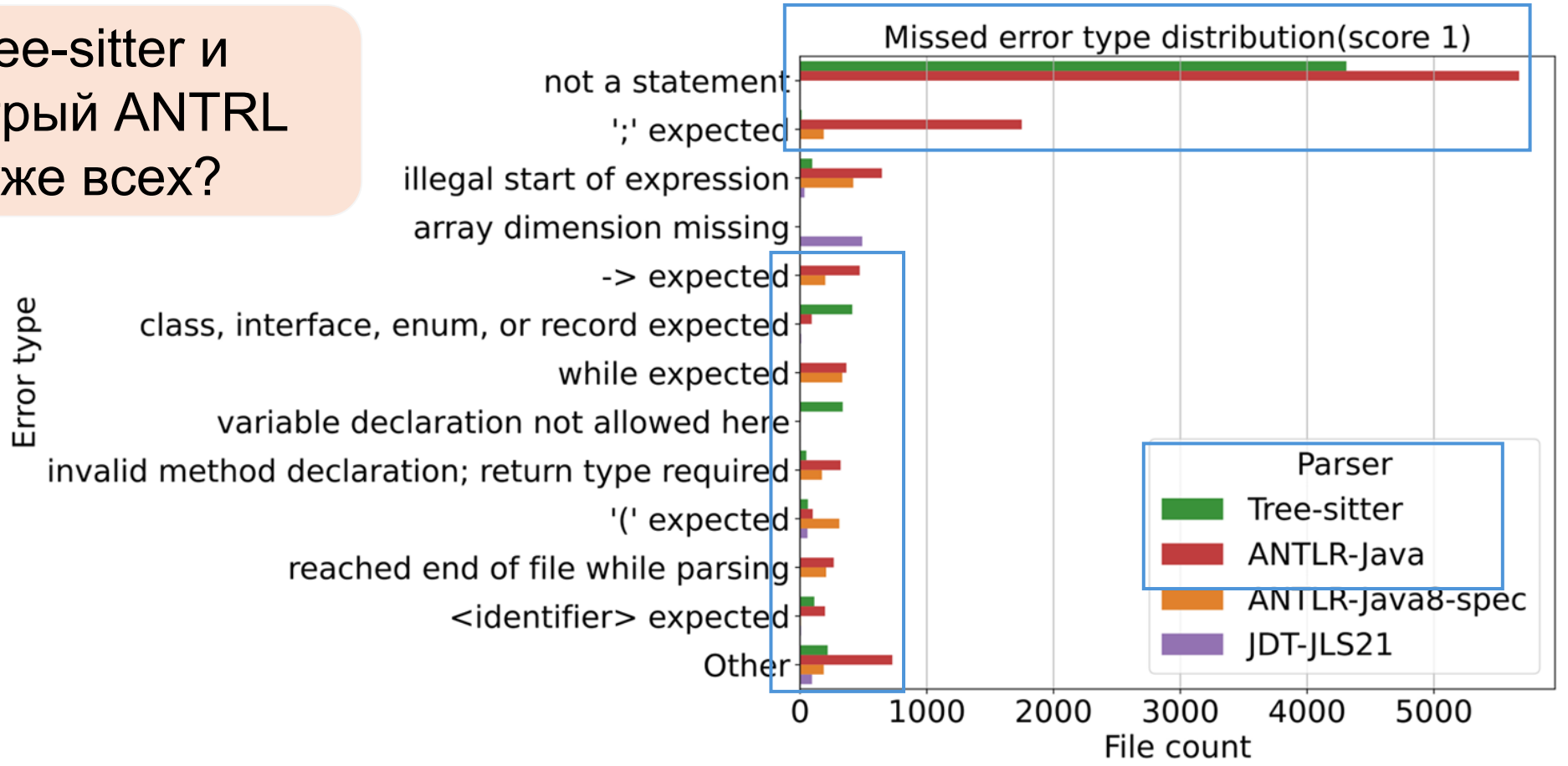


Подход 1. Количество пропущенных ошибок

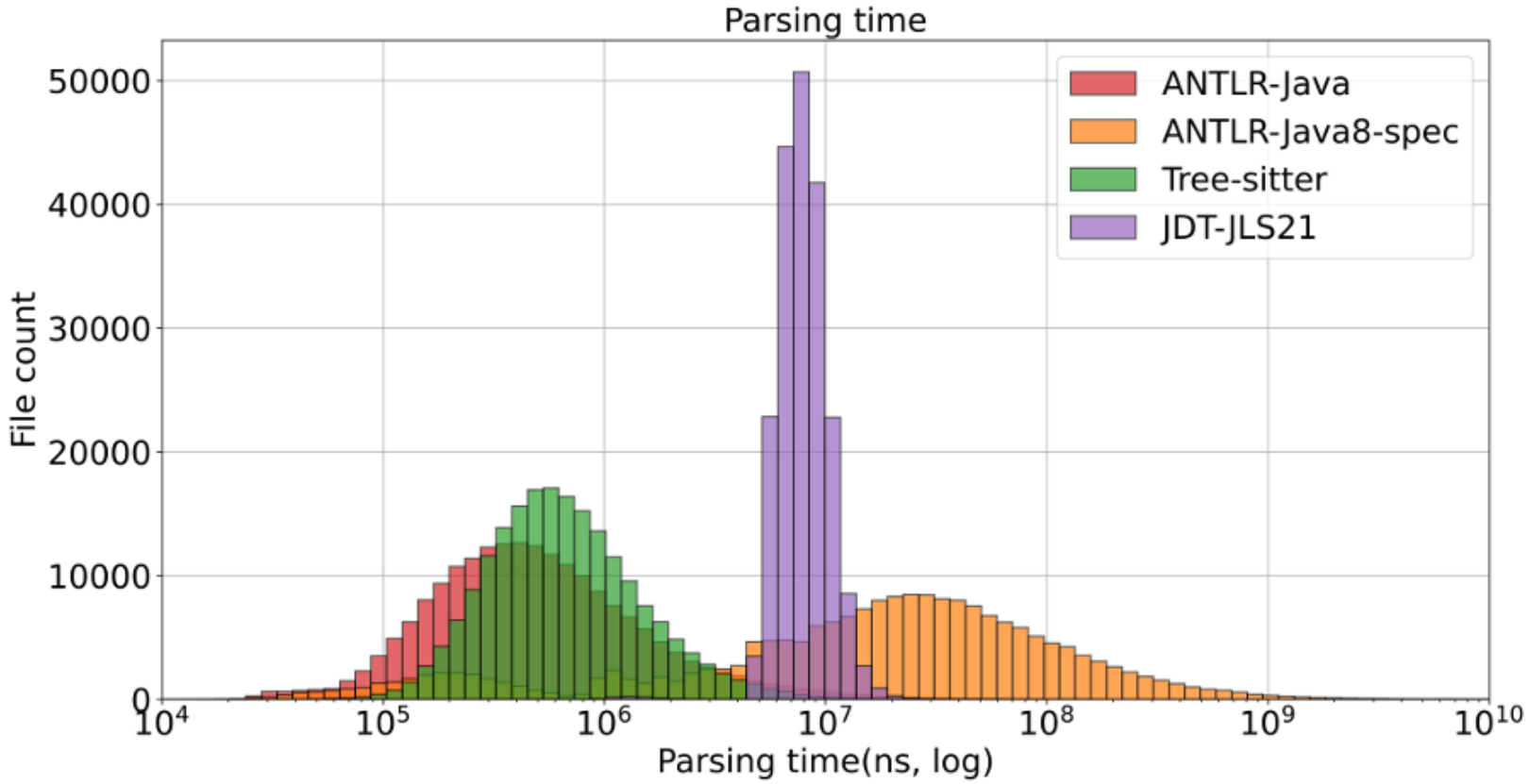


Подход 1. Количество пропущенных ошибок

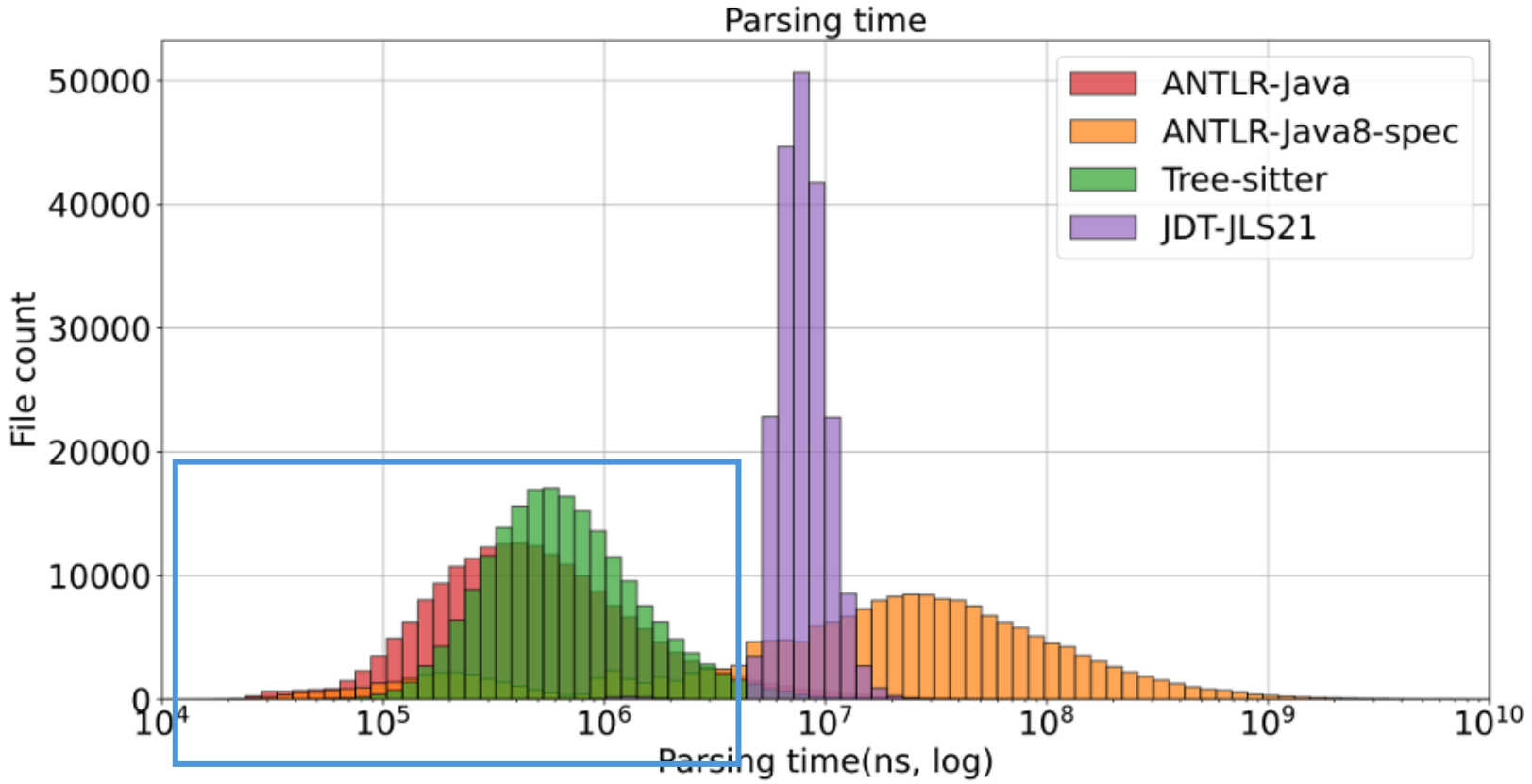
Tree-sitter и быстрый ANTRL хуже всех?



Подход 2. Скорость

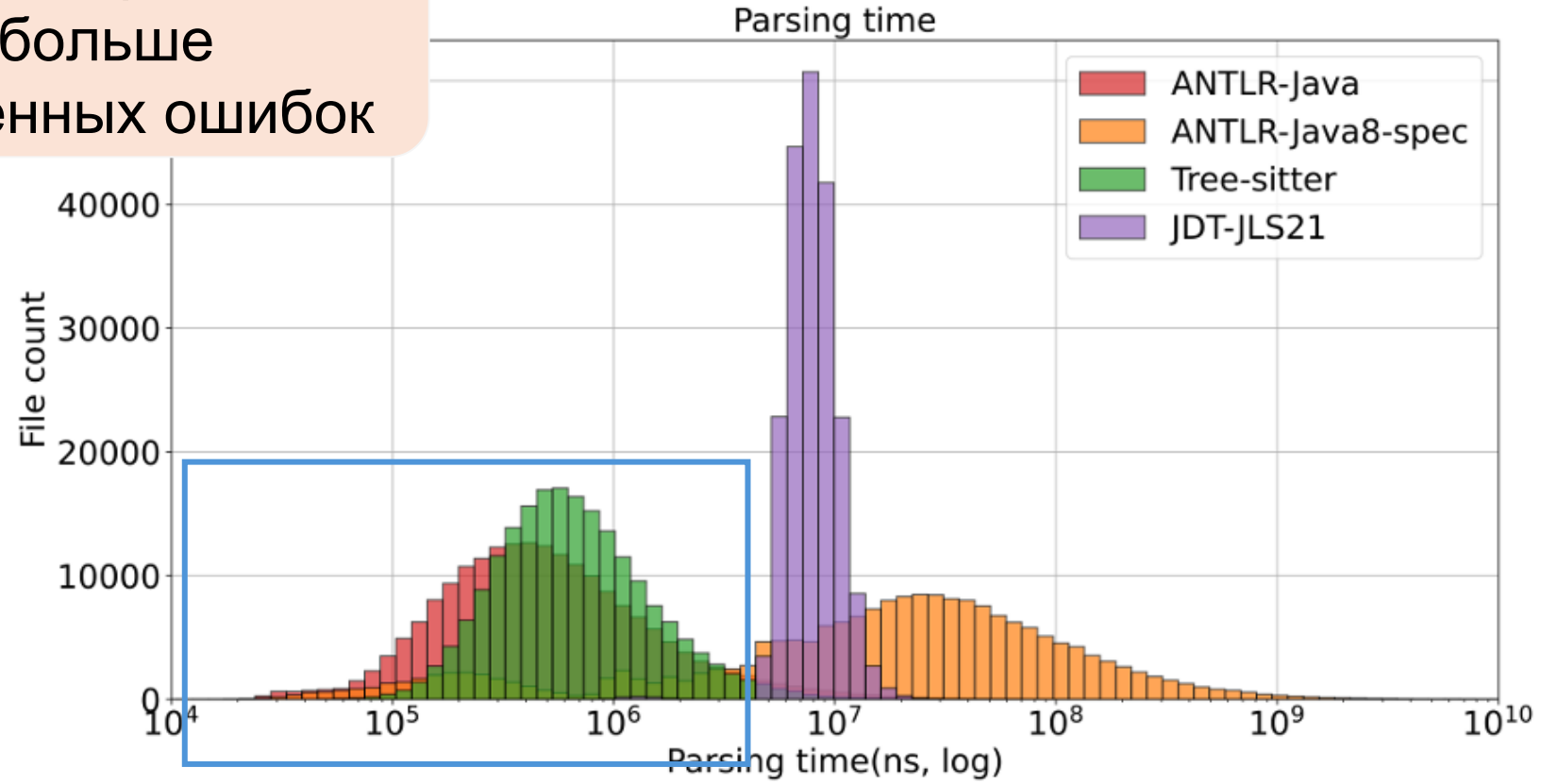


Подход 2. Скорость



Подход 2. Скорость

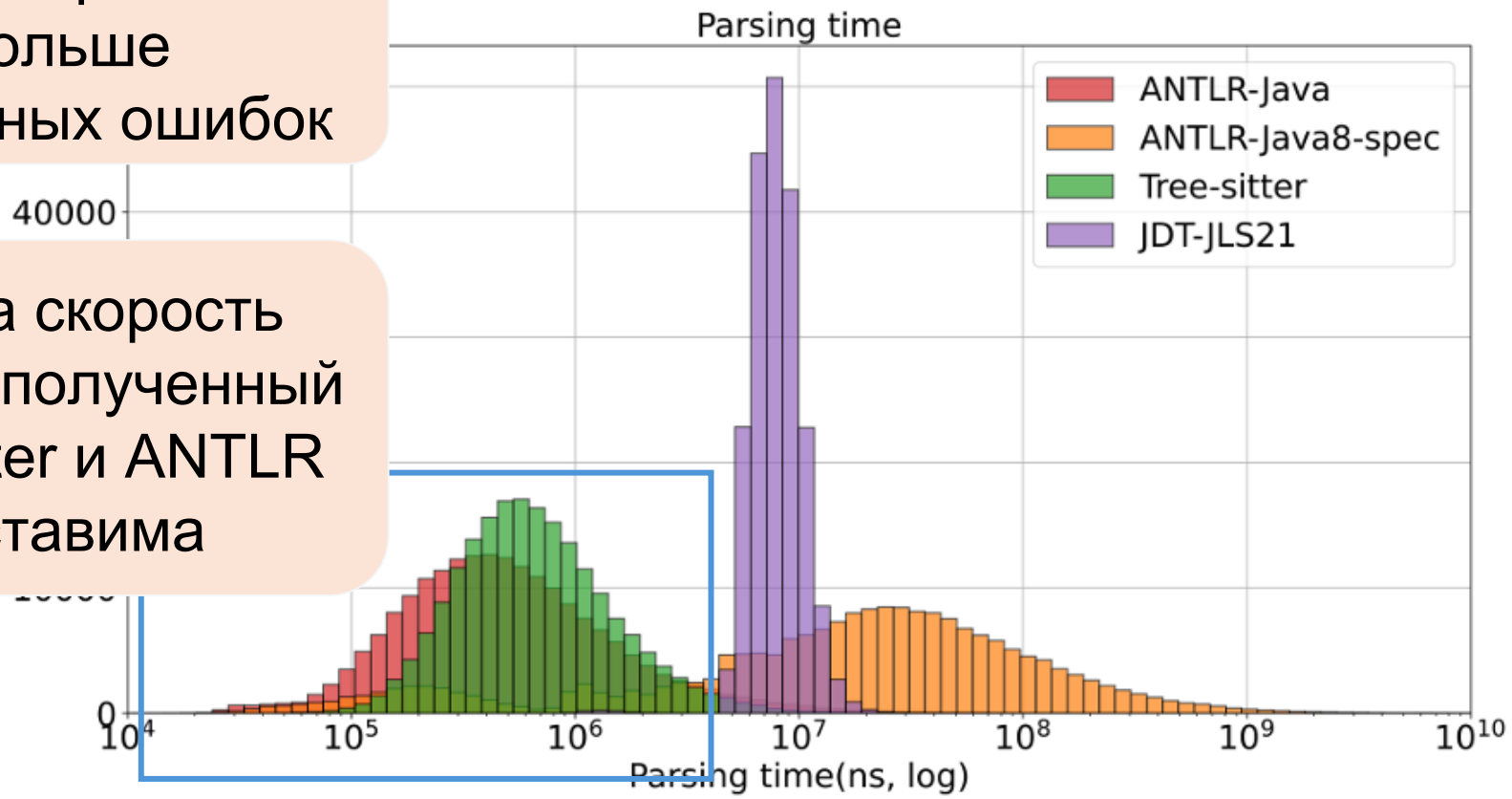
Быстрее обработка
→ больше
пропущенных ошибок



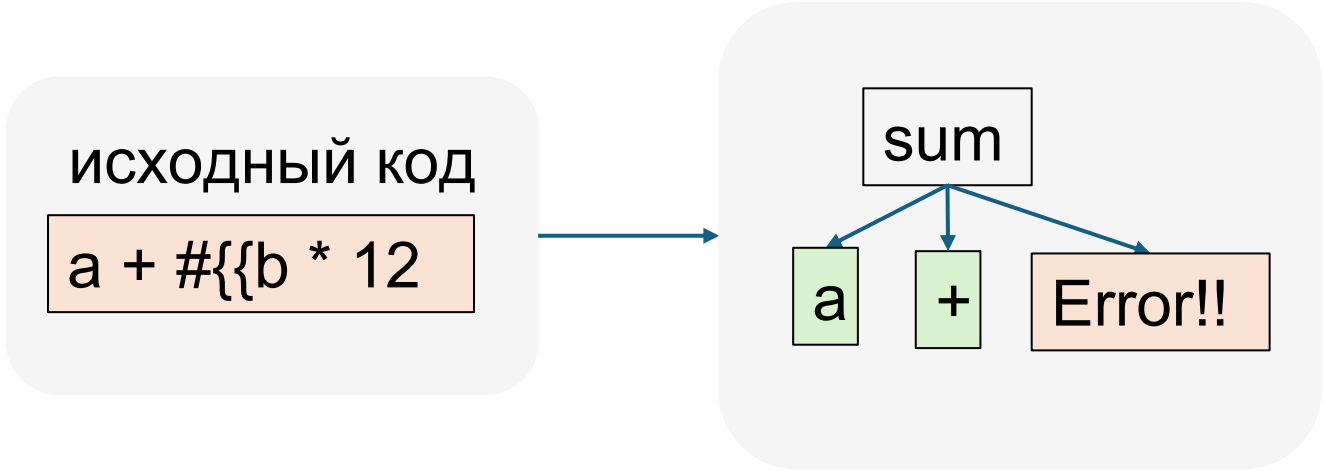
Подход 2. Скорость

Быстрее обработка → больше пропущенных ошибок

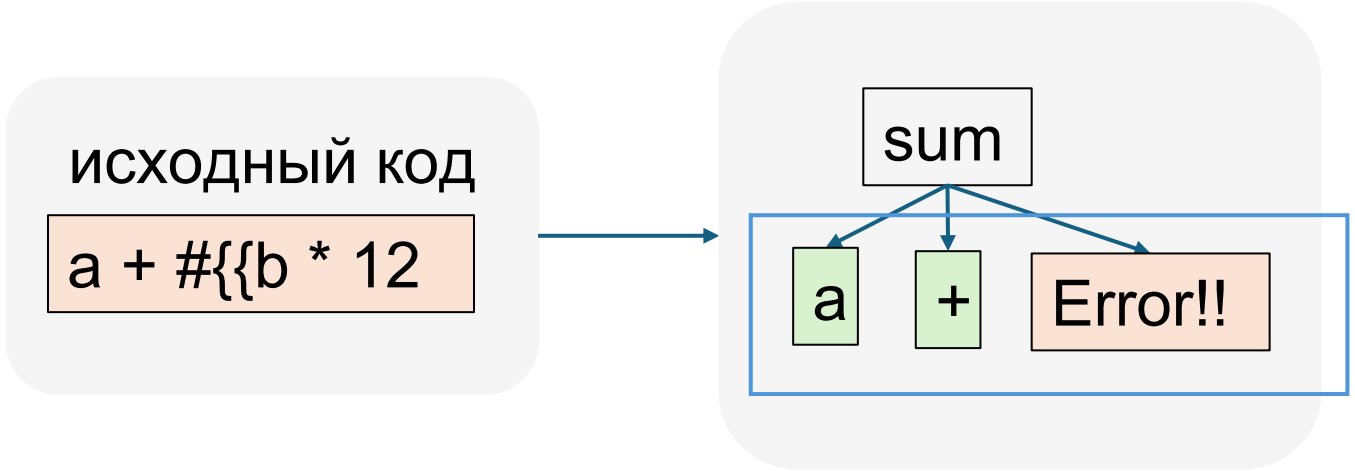
Для Java скорость парсеров, полученный с Tree-Sitter и ANTLR сопоставима



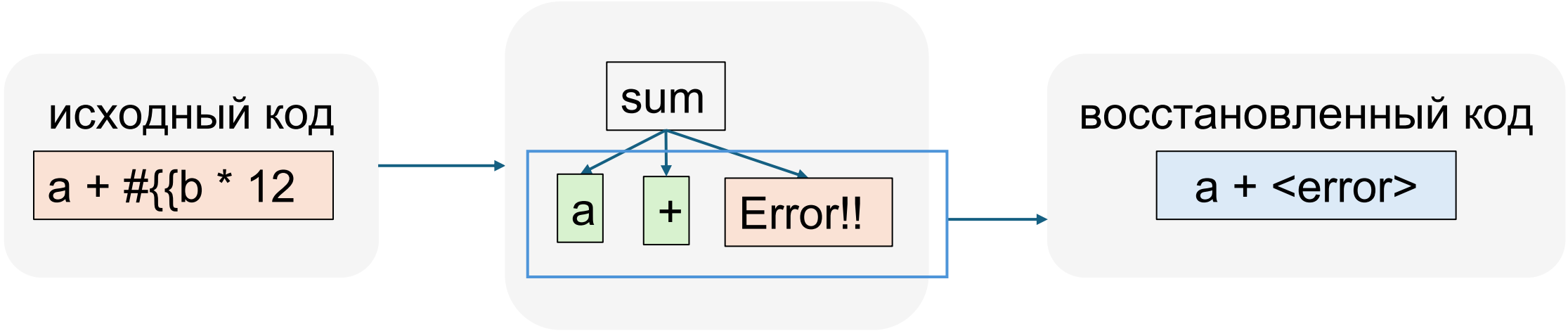
Подход 3. Близость текста



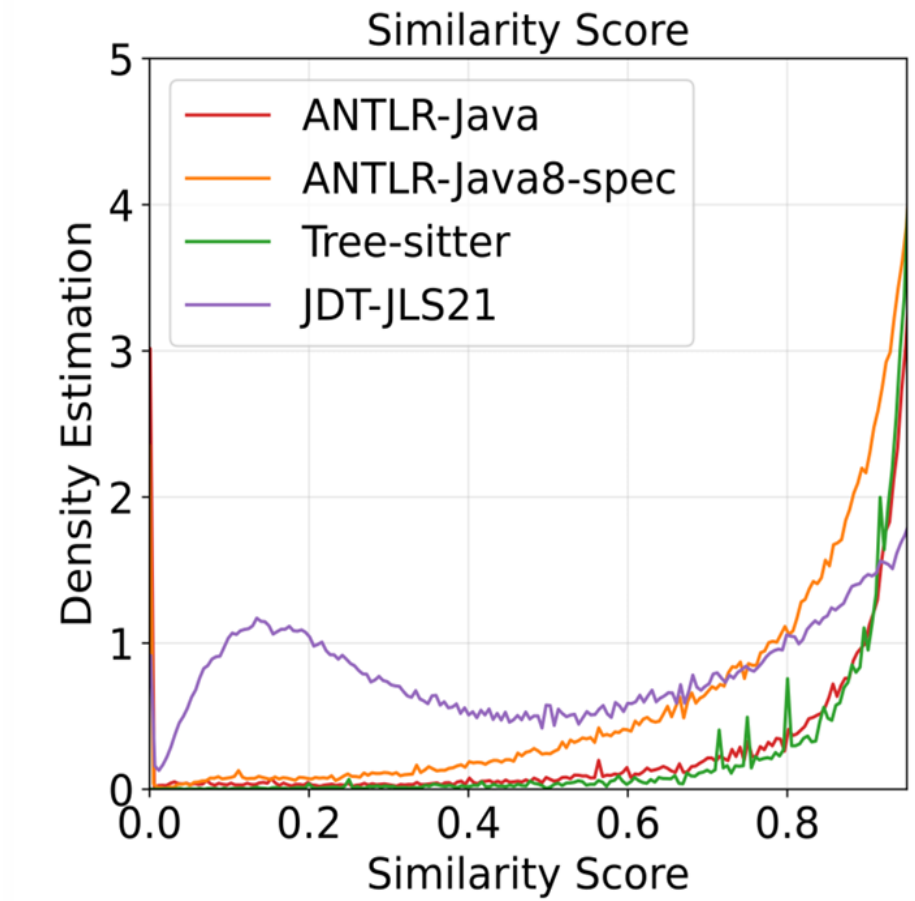
Подход 3. Близость текста

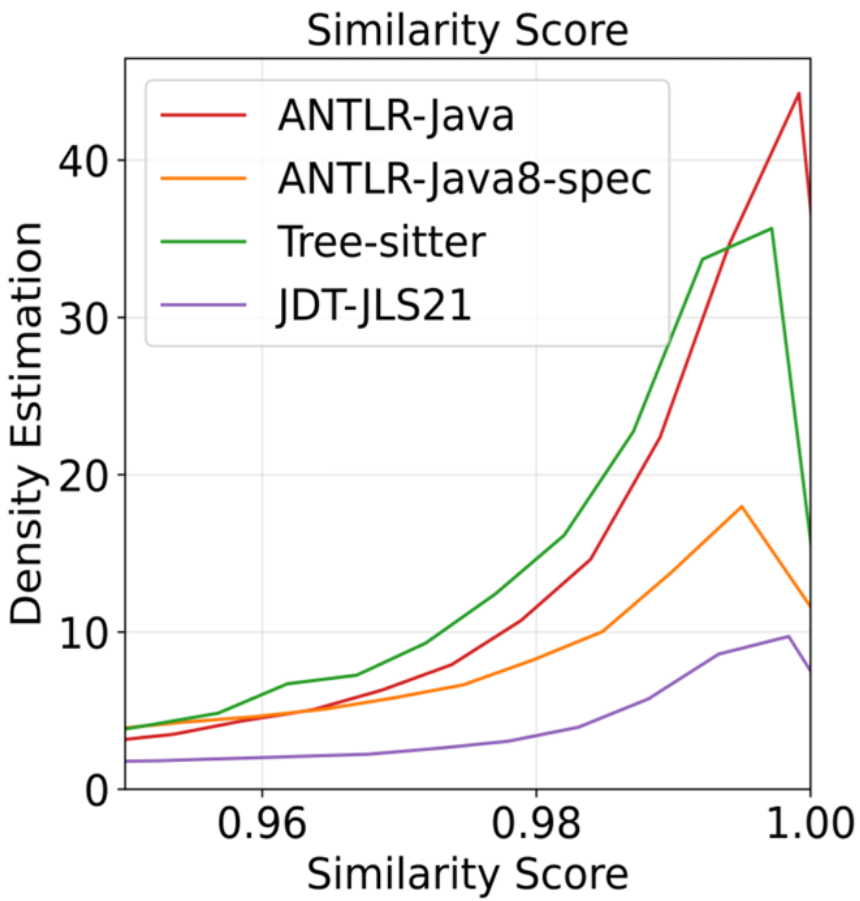


Подход 3. Близость текста



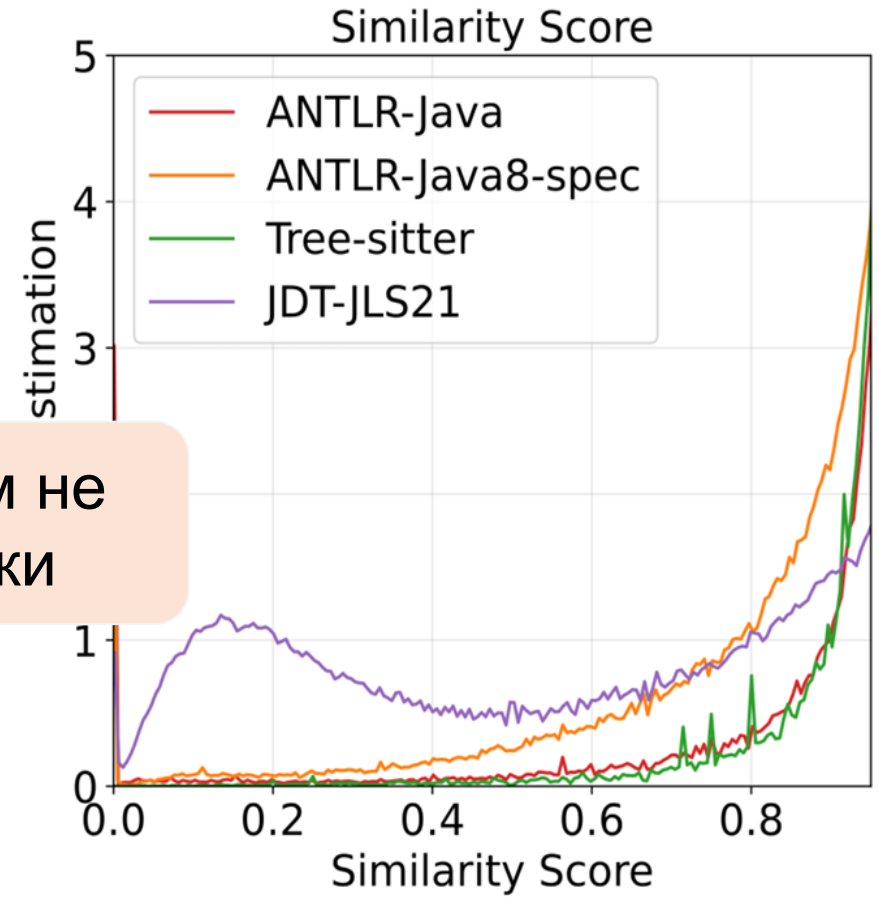
Подход 3. Близость текста

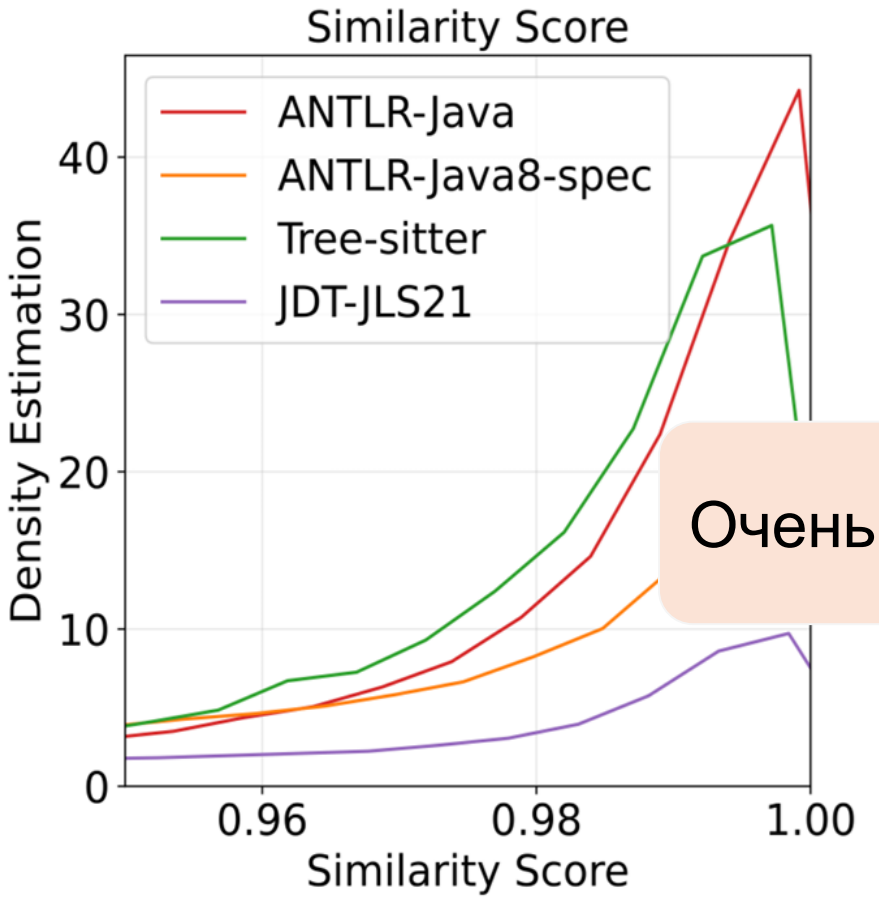




Подход 3. Близость текста

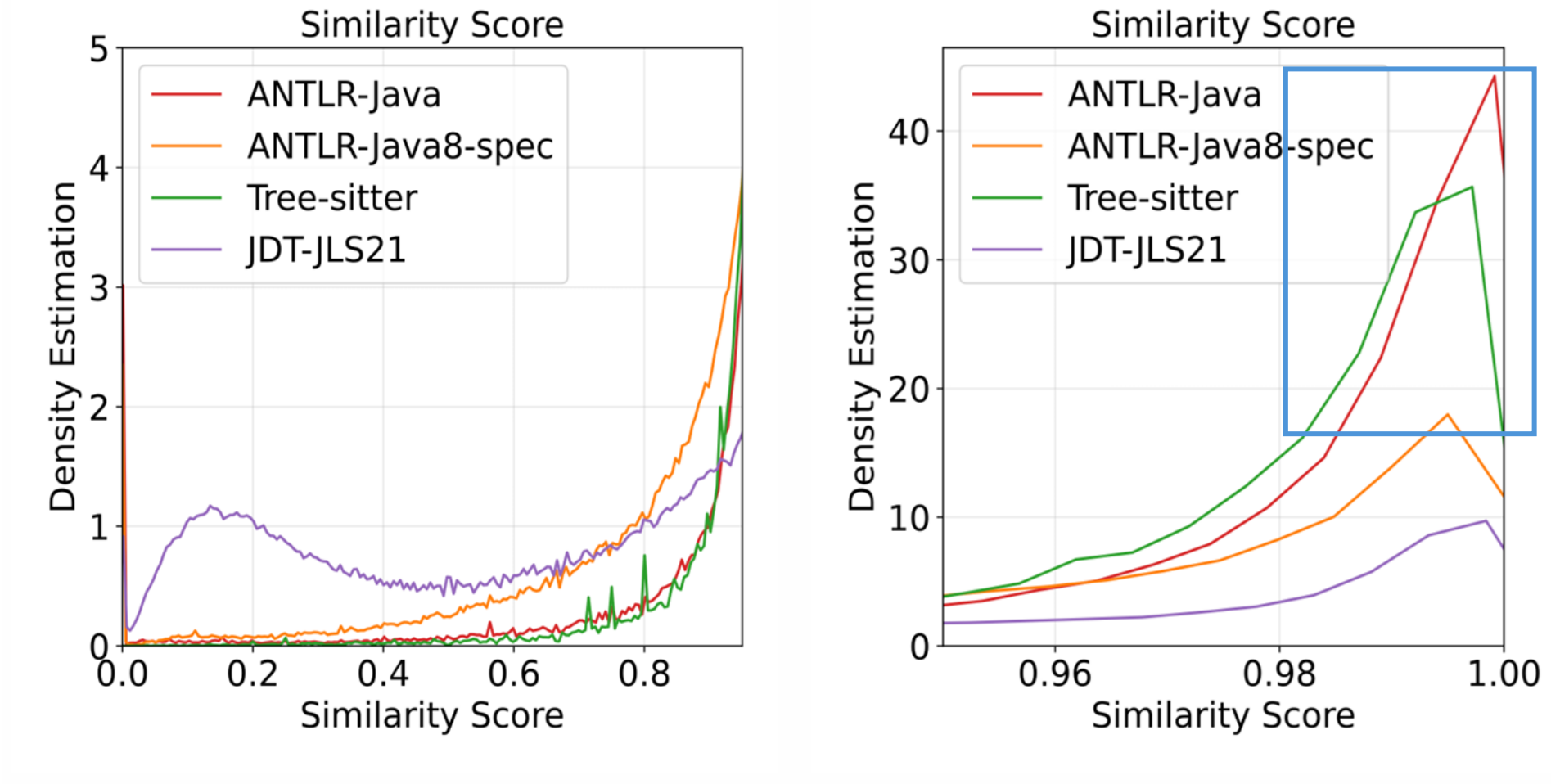
Совсем не похожи



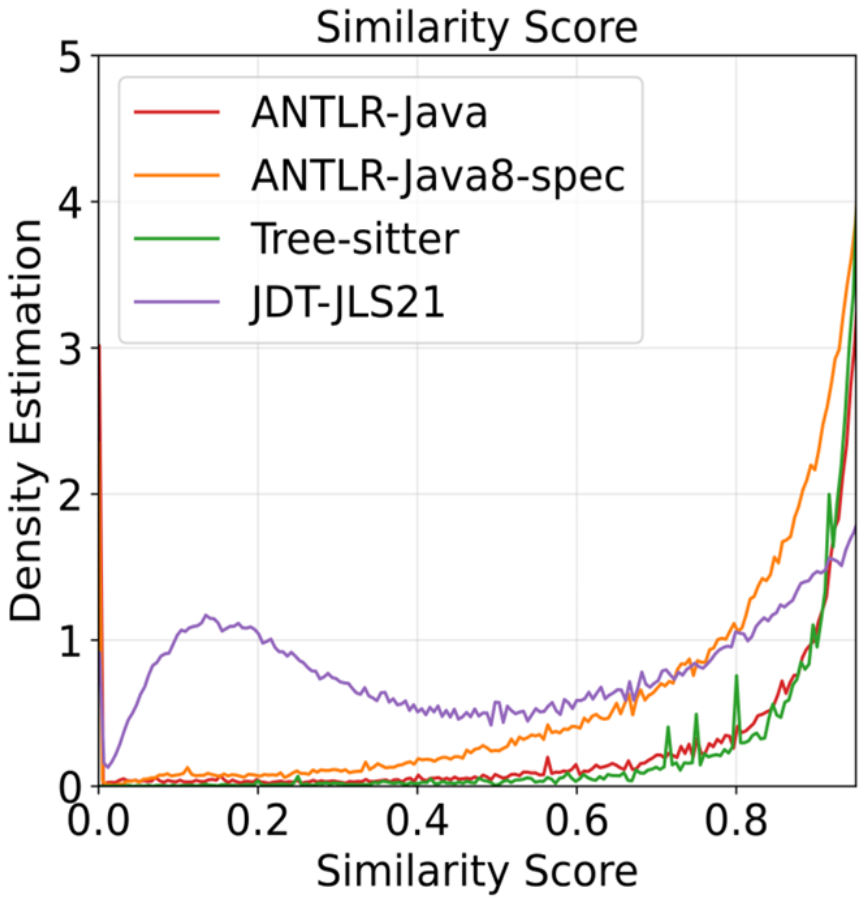


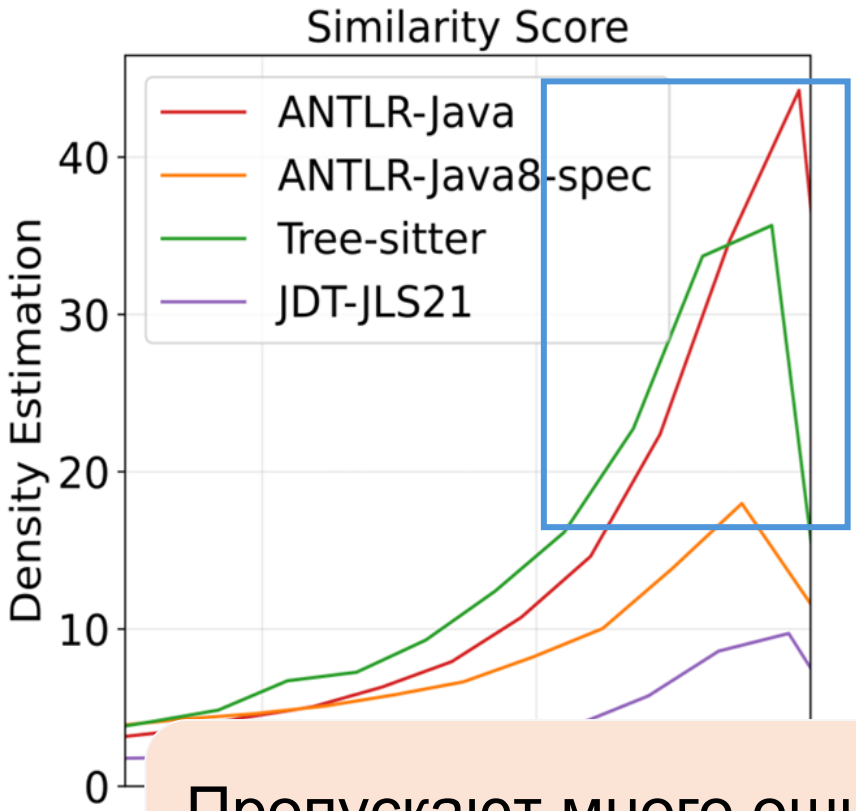
Очень похожи

Подход 3. Близость текста



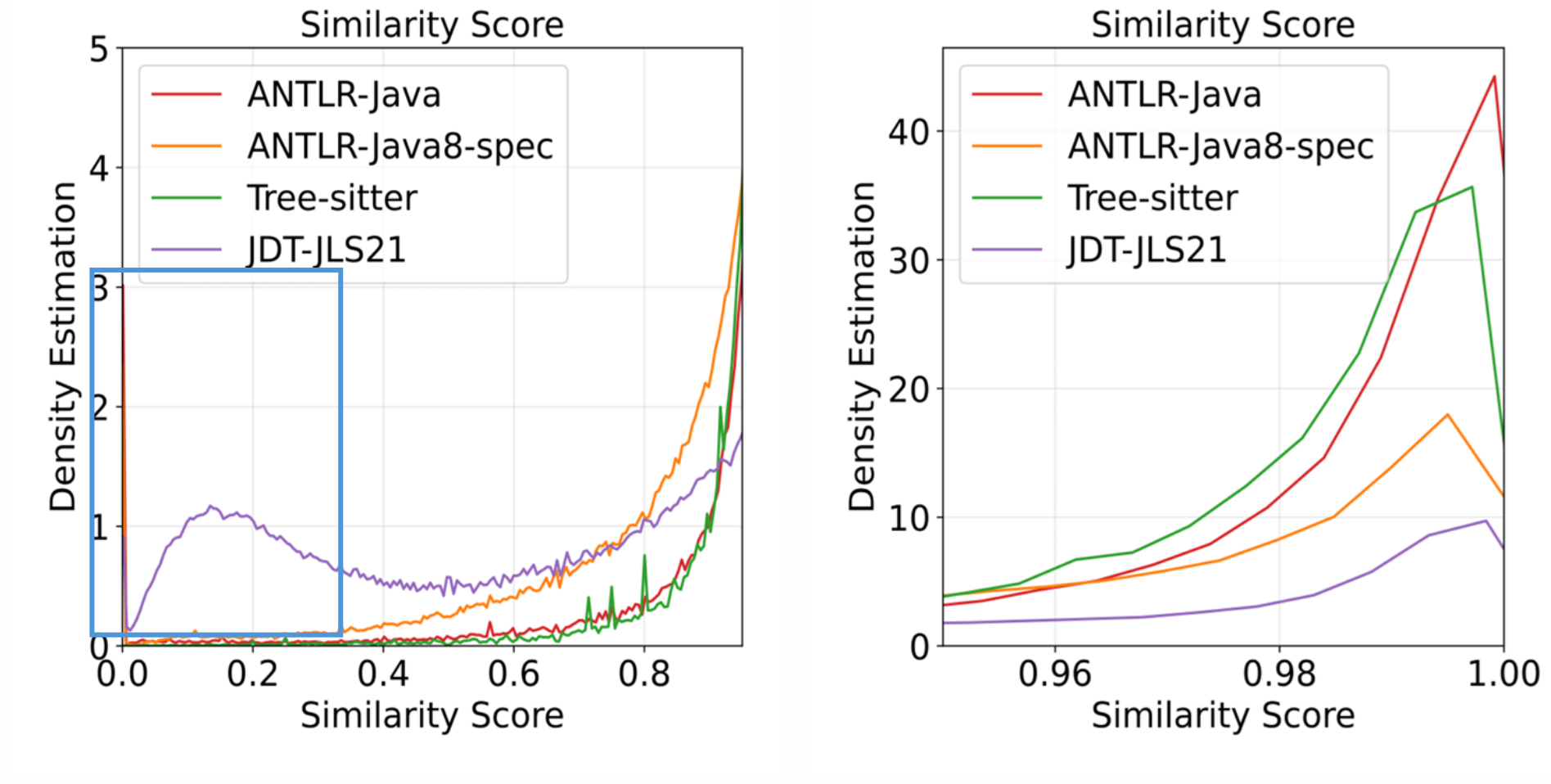
Подход 3. Близость текста



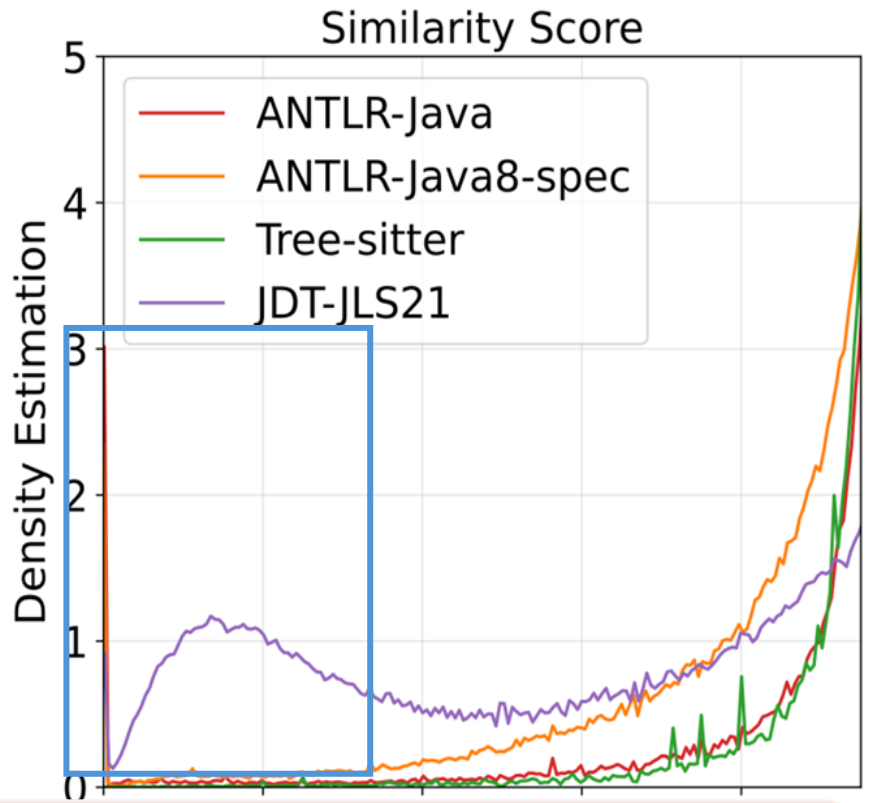


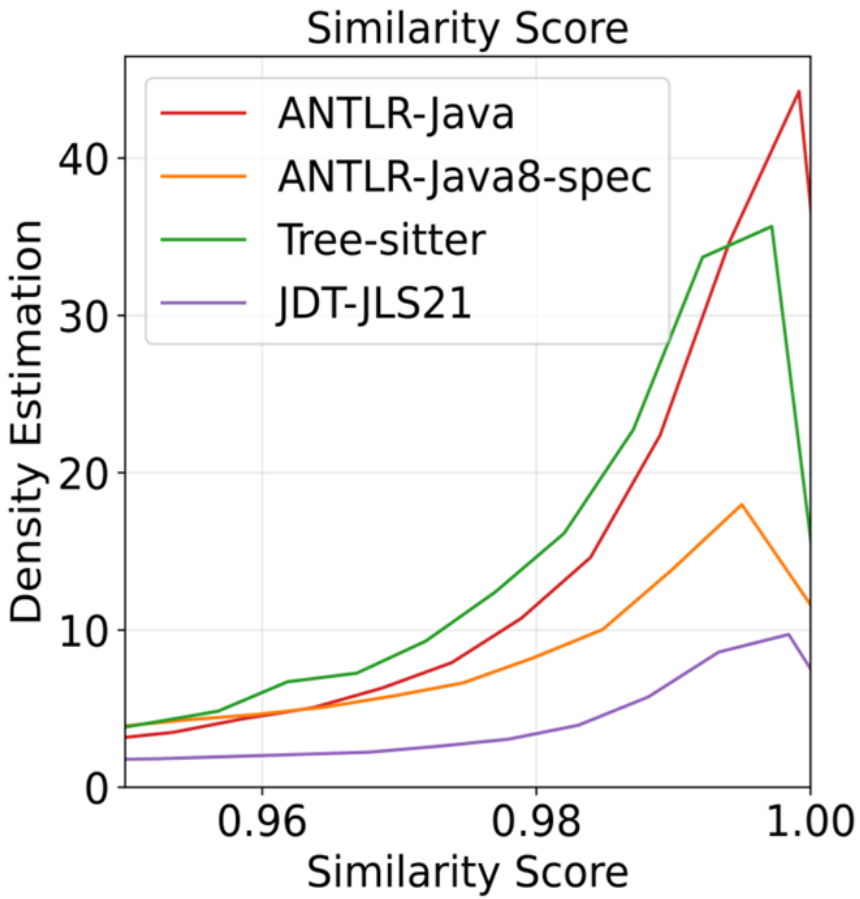
Пропускают много ошибок → текст ближе к исходному

Подход 3. Близость текста



Подход 3. Близость текста





Строги к грамматике →
сильно искажают текст

Выводы

- Парсер — фундамент IDE и AI coding tools

Выводы

- Парсер — фундамент IDE и AI coding tools
- Хороший error recovery экономит время разработчика, а в AI-системах ещё и токены, время и деньги

Выводы

- Парсер — фундамент IDE и AI coding tools
- Хороший error recovery экономит время разработчика, а в AI-системах ещё и токены, время и деньги
- Хороший генератор != хороший парсер

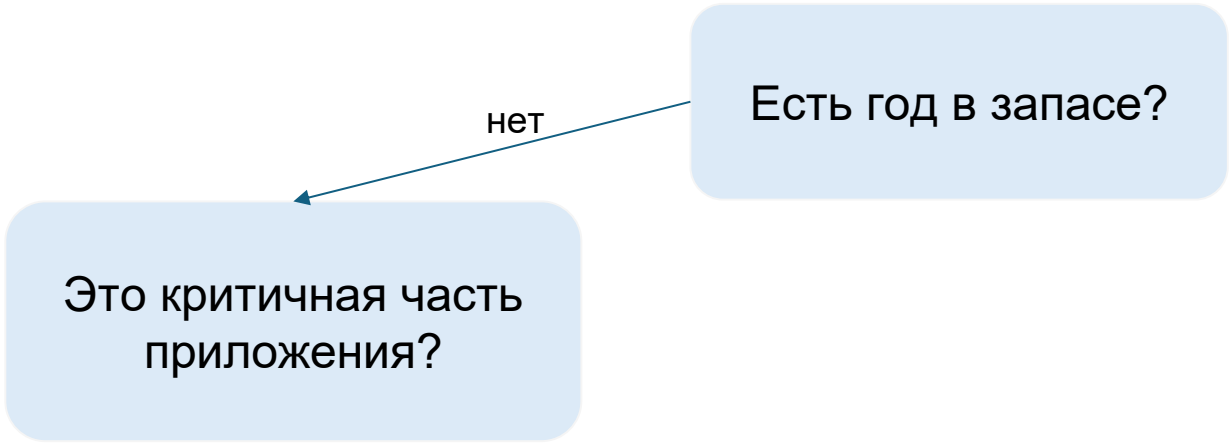
Выводы

- Парсер — фундамент IDE и AI coding tools
- Хороший error recovery экономит время разработчика, а в AI-системах ещё и токены, время и деньги
- Хороший генератор != хороший парсер
- Идеальной метрики нет

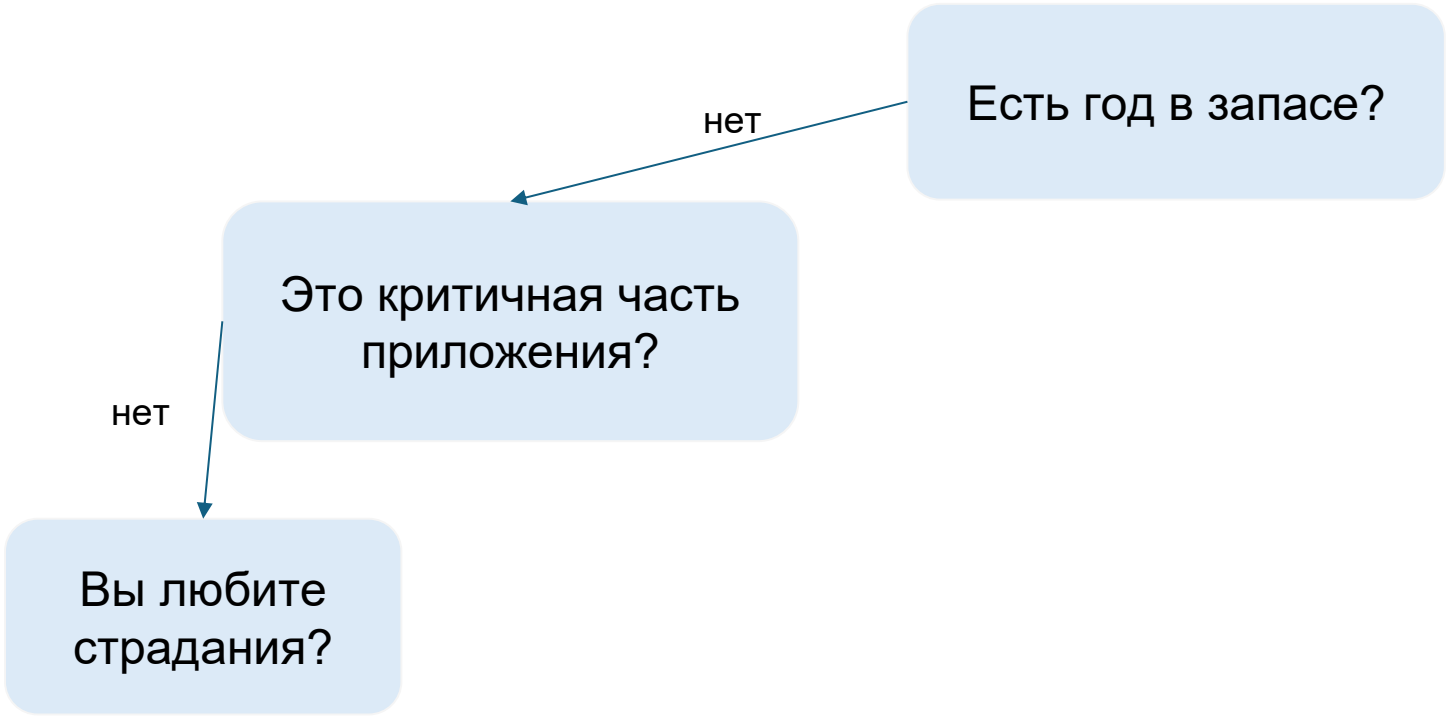
Если нужен парсер в проект

Есть год в запасе?

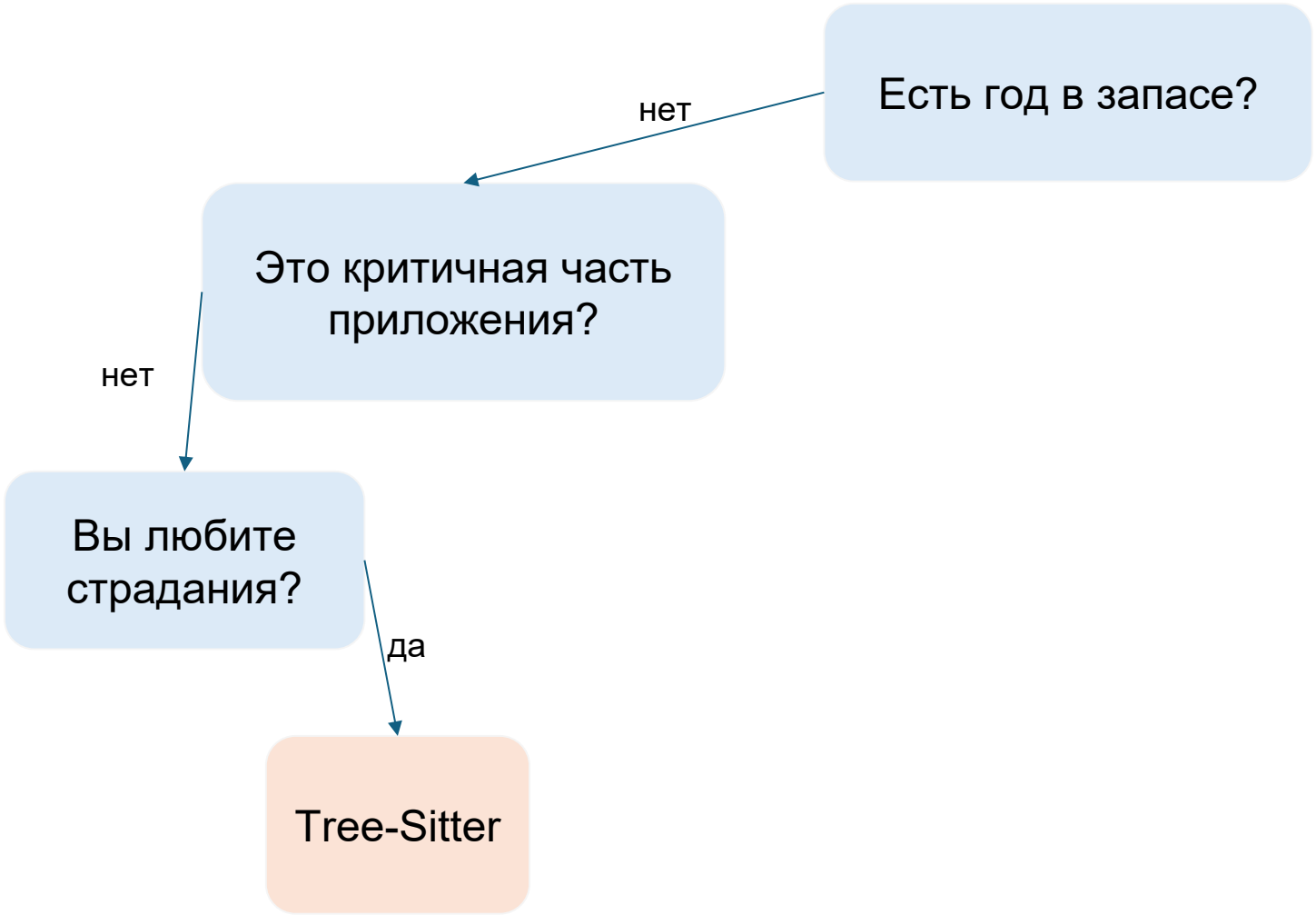
Если нужен парсер в проект



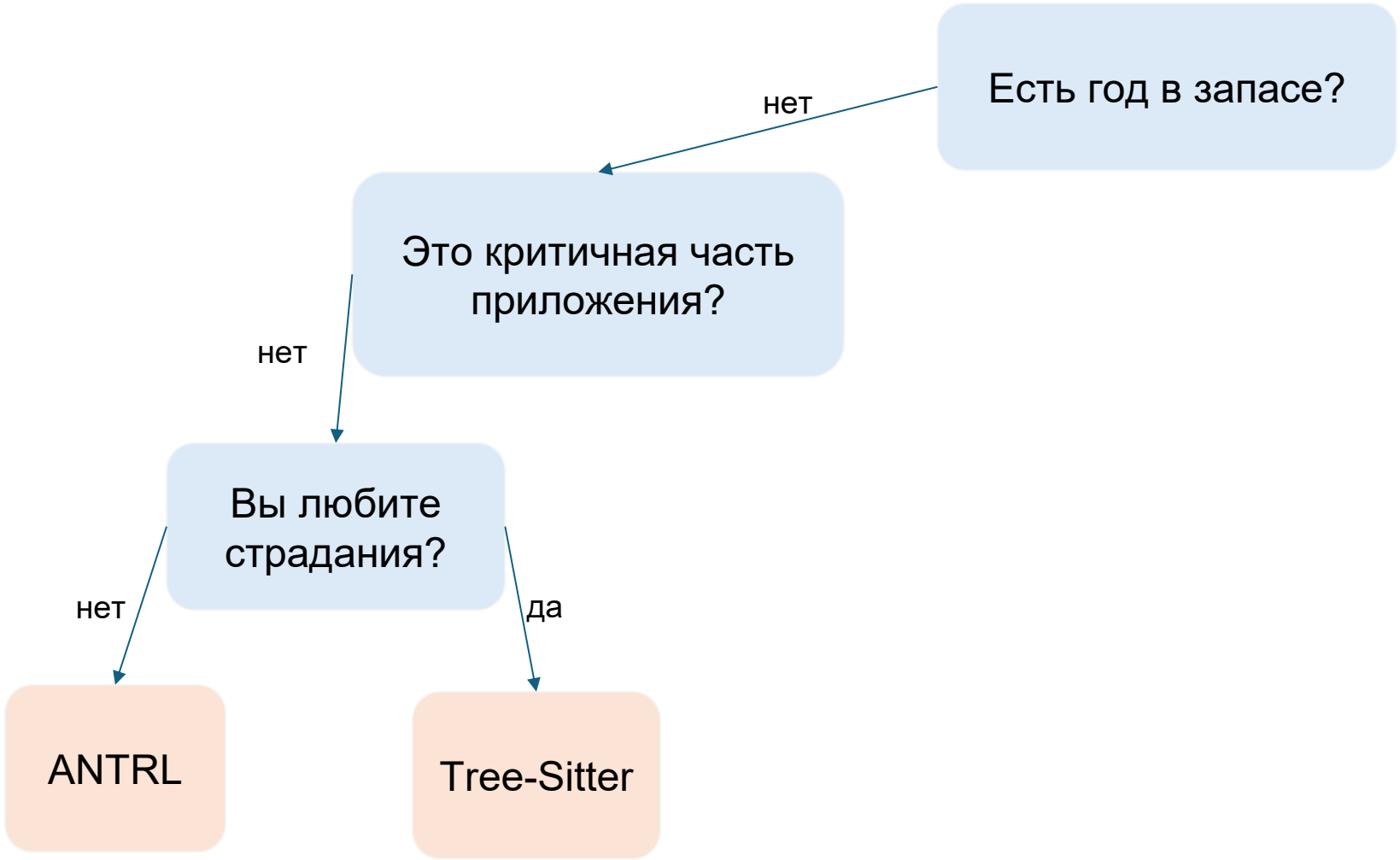
Если нужен парсер в проект



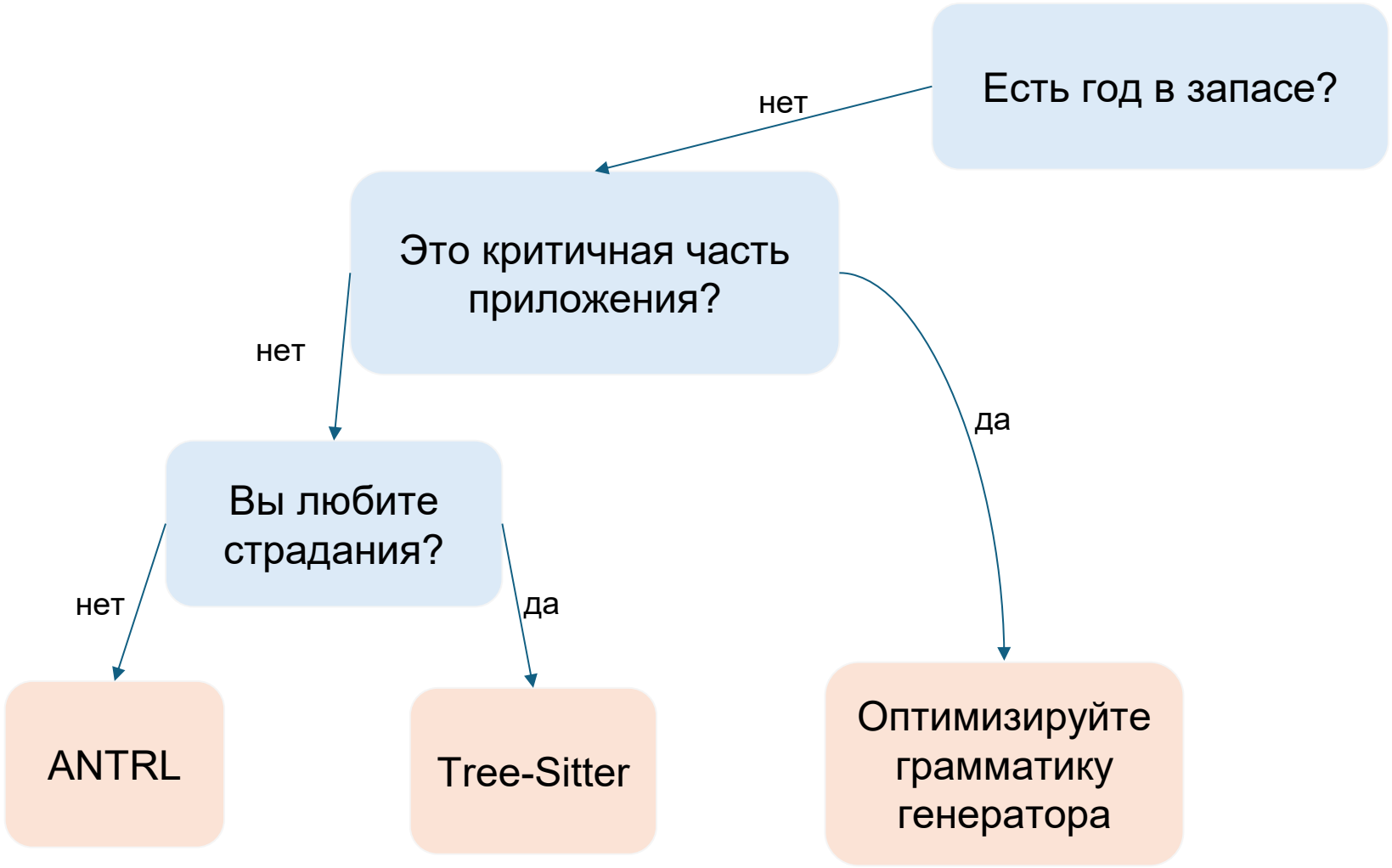
Если нужен парсер в проект



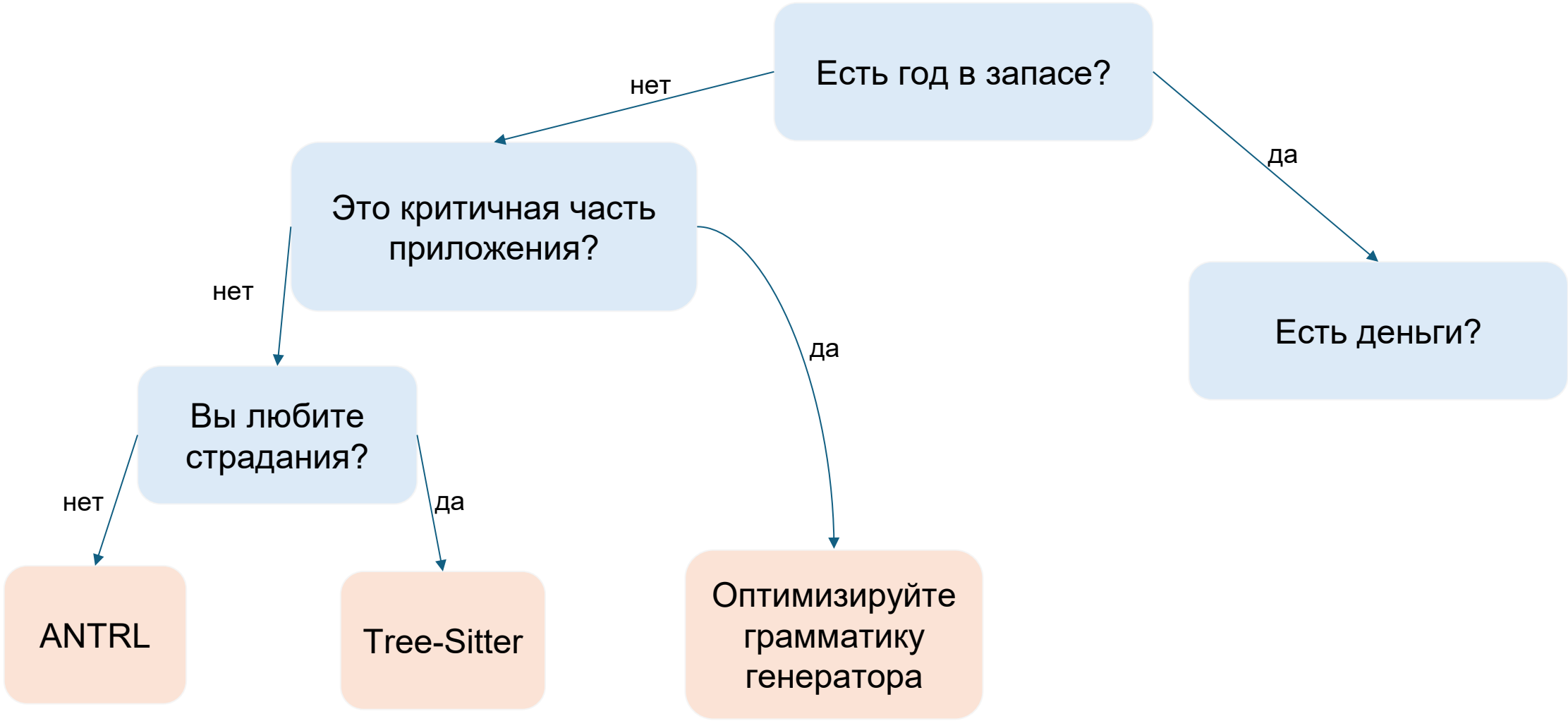
Если нужен парсер в проект



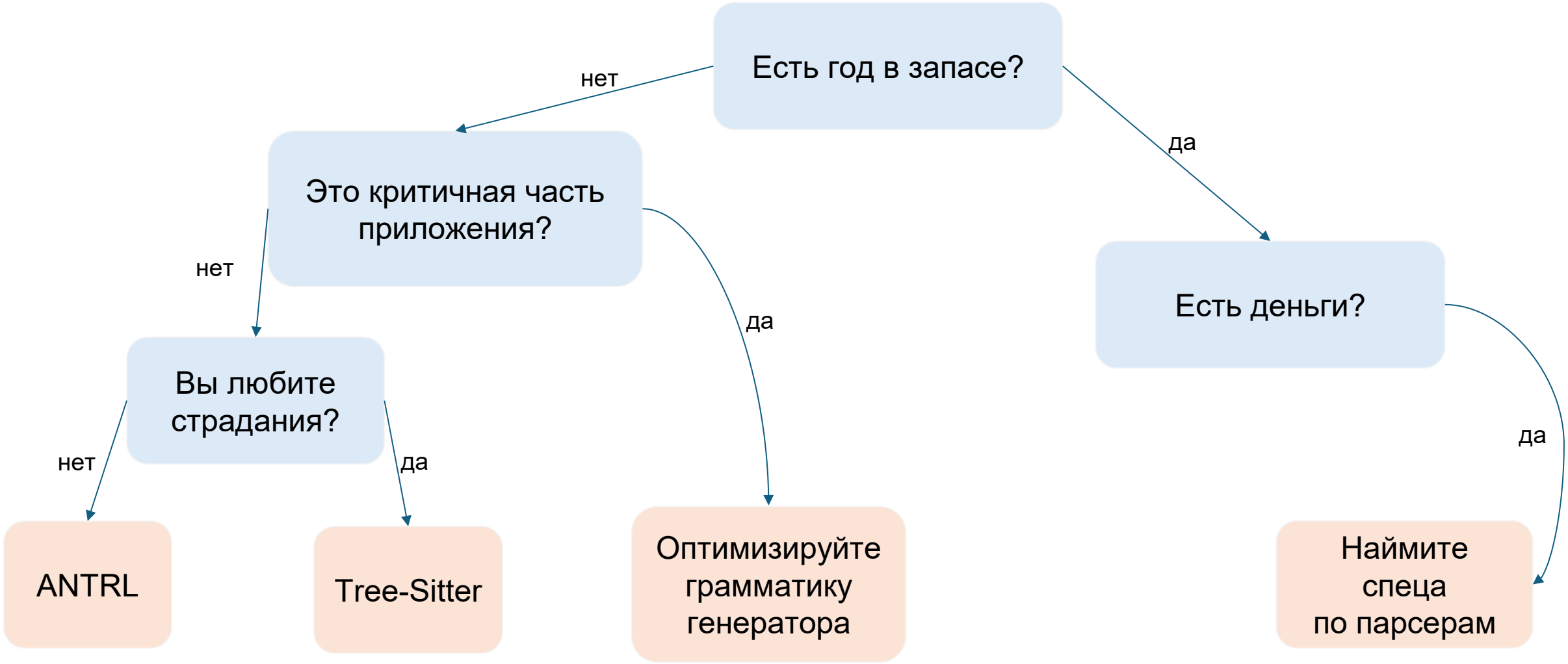
Если нужен парсер в проект



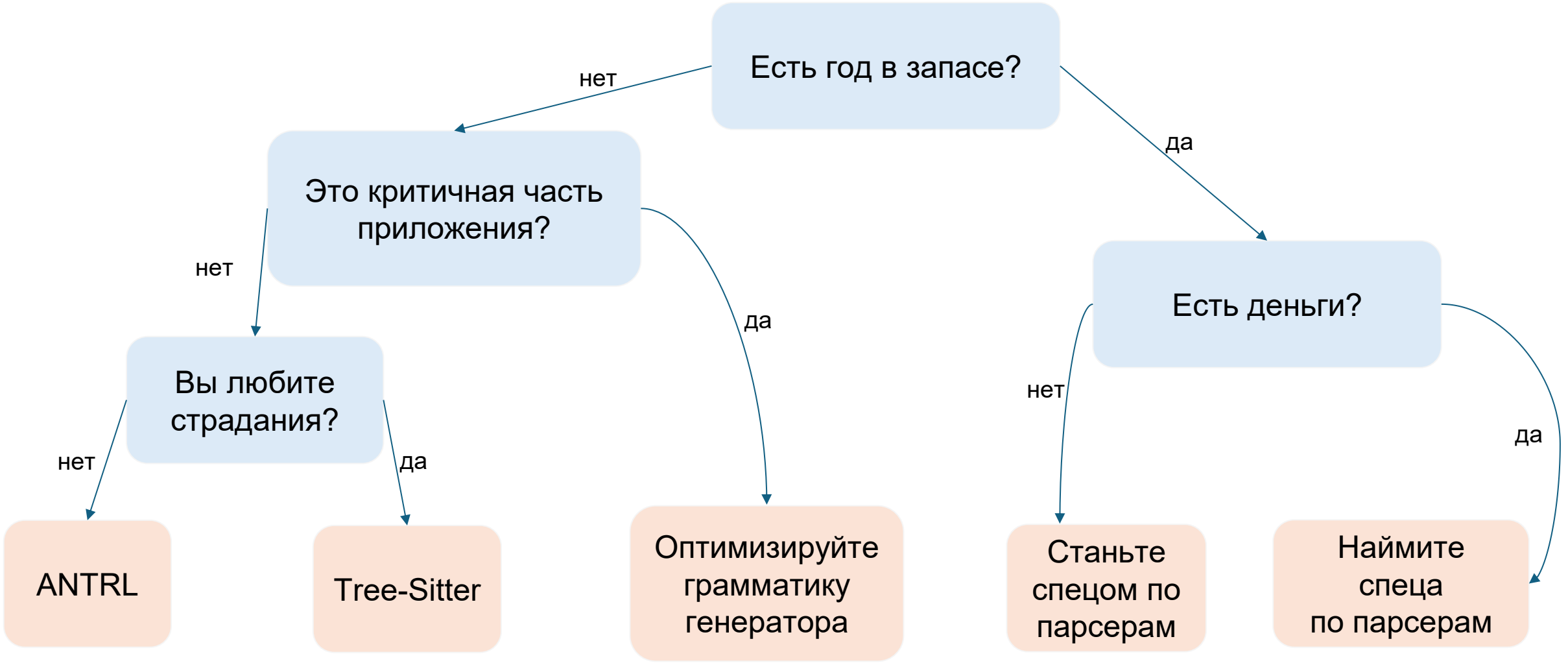
Если нужен парсер в проект



Если нужен парсер в проект



Если нужен парсер в проект



Время для вопросов

Почему IDE ломается из-за одной ошибки
и как понять, хороший ли у нее error-recovery?

Оля Бачище
bachisheo@yandex.ru
tg: @bachisheo
tgc: @olya_bach