Архитектура компиляции: проблемы и решения

Танцы вокруг C++:
двадцать лет упражнений

Евгений Зуев
Университет Иннополис

C++ Russia
Ноябрь 2020

Outline

- Who is this guy?
- Зачем нужен (ещё один) компилятор C++?
- Реально ли это?
- Какова, собственно, цель?
- Компиляция в узком и широком смысле.
- IDE и компилятор: как интегрировать?
- Архитектура компилятора: монолит («черный ящик») или набор сервисов?
- Тестирование: подход Ada, подход gcc или...?
- Импортозамещение и компиляторы

Who is this Guy?

- **Евгений Зуев**
- Работал в МГУ, Swiss Fed Inst of Technology в Zürich (ETHZ) в Lausanne (EPFL), Samsung R&D; Сейчас в университете Иннополис.
- PhD (1999, НИВЦ МГУ).
- Профессиональные интересы: **compiler construction, language design, programming languages' semantics**.
- Несколько книг по программированию.

Email: e.zuev@innopolis.ru
Telegram: @zouev

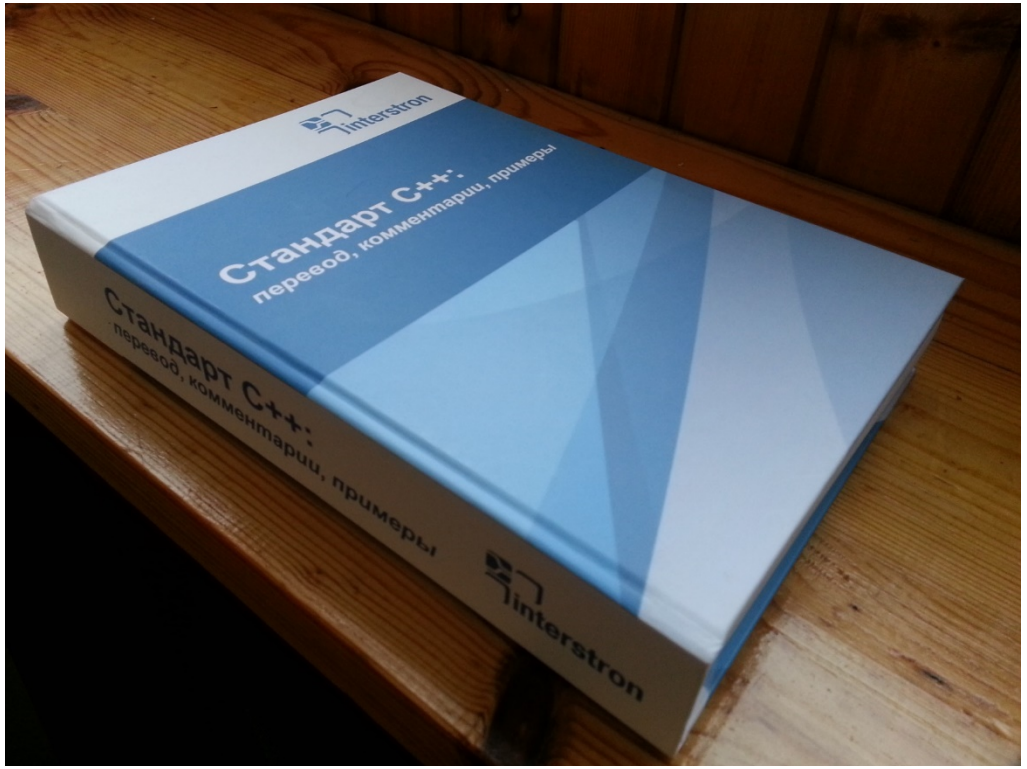
Некоторые проекты

- ISO compliant **C++ front end compiler**
  Интерстрон, Москва, 2000
- **Zonnon language and compiler** for .NET & Visual Studio,
  ETH Zürich, 2005
- **Swift prototype compiler** for OS Tizen
  Samsung Research, Moscow, 2015

Несколько книг по программированию; последние:

- «Редкая профессия», ДМК Пресс, 2014
- Русский перевод предварительного стандарта ISO C++,
  Интерстрон, 2016
- Software Design for Resilient Computer Systems,
  Springer Verlag, 2019

Побочные продукты производства ☺



Е. Зуев, А. Чупринов
Стандарт C++:
перевод, комментарии, примеры
Интерстрон, 2016 г.

Е. Зуев
Редкая профессия
ДМК-Пресс, 2014 г.
ISBN: 978-5-94074-812-0

Зачем нужен (еще один) компилятор C++?

- *А почему разработчик должен давать ответ? –* **Компания** так решила ☺.
- Лицензионные проблемы.
- Существующие компиляторы не подходят для конкретных CPU/ОС.
- Функциональность имеющихся компиляторов слишком ограничена.
- Импортозамещение! ☺

Зачем нужен (еще один) компилятор C++?

- *А почему разработчик* должен давать ответ? – **Компания** так решила ☺.
- Лицензионные проблемы.
- Существующие компиляторы не подходят для конкретных CPU/ОС.
- Функциональность имеющихся компиляторов слишком ограничена.
- Импортозамещение! ☺
- **We can- therefore we must**

J. Gutknecht, ETH Zürich

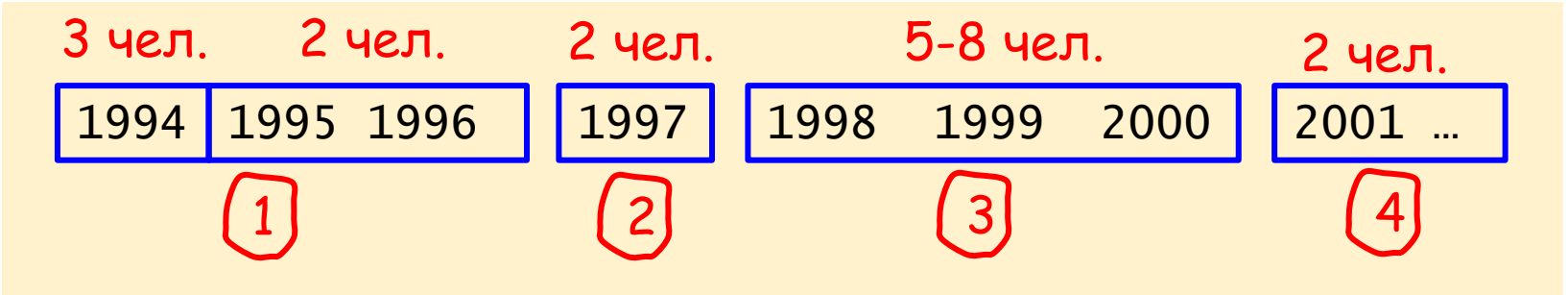
Реально ли это?

Есть весомые резоны для сомнений:

- C++ - исключительно сложный объект: для изучения, для использования, **для реализации**
- Нет необходимой гарантии успеха.
- Помимо сложности, есть и фактор времени: может, быстрее получится, если всё-таки взять gcc/clang и адаптировать/портировать его?

Реально ли это?

Наш опыт

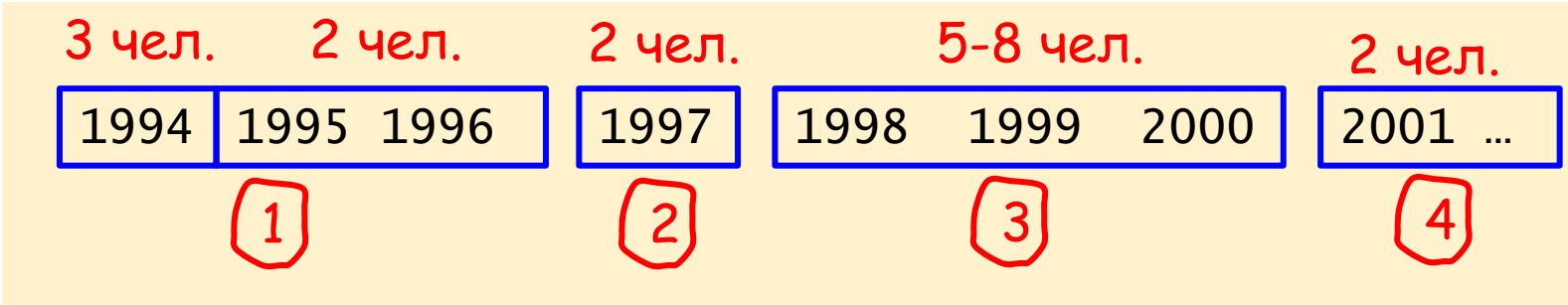


Версия 1:

C++ Front End (ISO 1996) for a multi-language SDK, Unix-based SparcStations, ACE company (the Netherlands)

Реально ли это?

Наш опыт

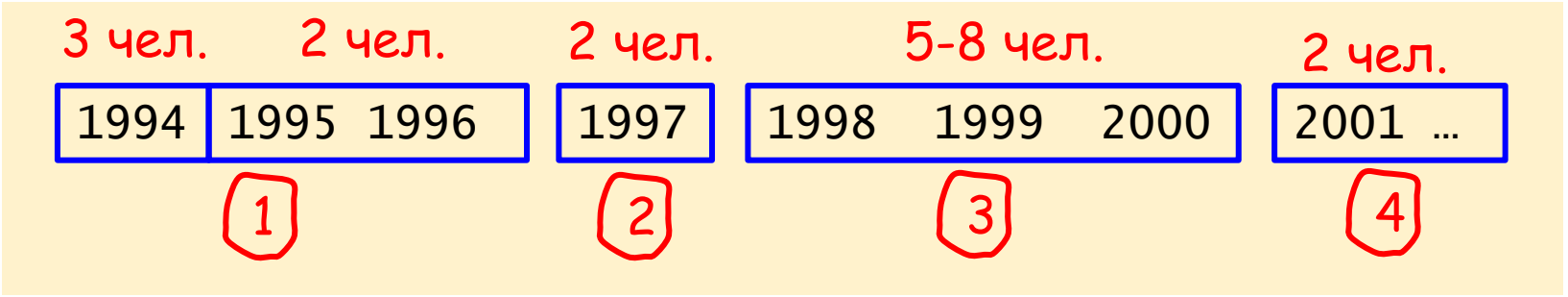


Версия 1:
C++ Front End (ISO 1996) for a multi-language SDK, Unix-based SparcStations, ACE company (the Netherlands)

Версия 2:
Portable C++ front end for Solaris & Windows
(*версия 1 была практически полностью переписана*)

Реально ли это?

Наш опыт



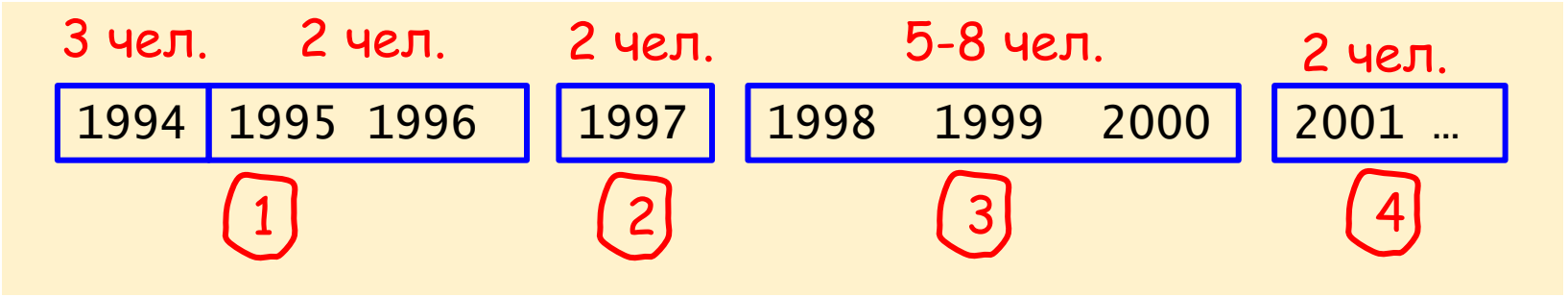
Версия 1:
C++ Front End (ISO 1996) for a multi-language SDK, Unix-based SparcStations, ACE company (the Netherlands)

Версия 2:
Portable C++ front end for Solaris & Windows
(*версия 1 была практически полностью переписана*)

Версия 3:
C++ Front End with API & C++ Virtual Machine (ISO 1998)

Реально ли это?

Наш опыт



Версия 1:
C++ Front End (ISO 1996) for a multi-language SDK, Unix-based SparcStations, ACE company (the Netherlands)

Версия 2:
Portable C++ front end for Solaris & Windows
(*версия 1 была практически полностью переписана*)

Версия 3:
C++ Front End with API & C++ Virtual Machine (ISO 1998)

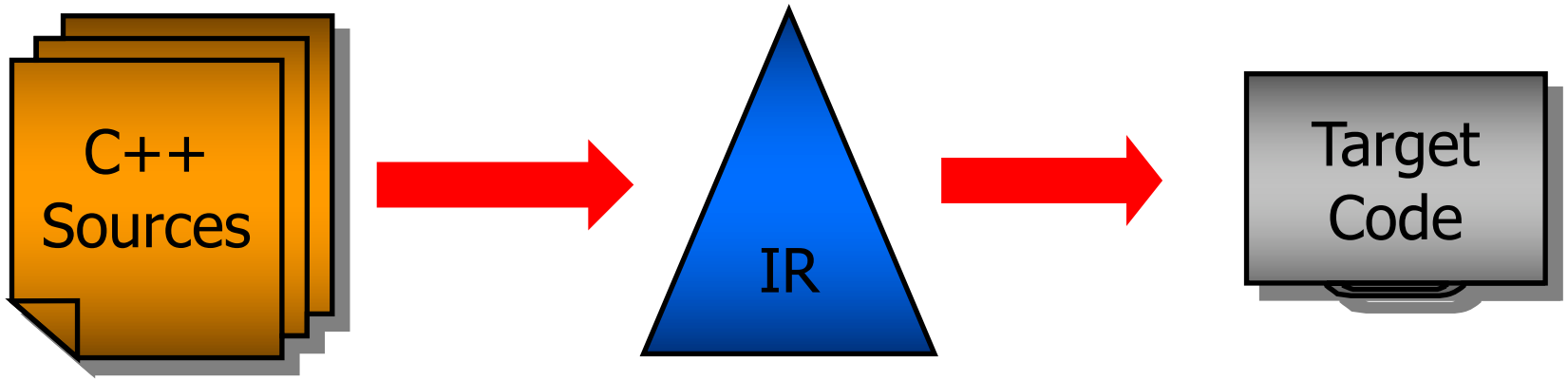
Версия 4:
Interstron C++ front end (ISO 2011) для нескольких моделей росс. микропроцессоров

Какова цель?

Какие задачи должен выполнять компилятор?

- **Генерация эффективного кода**

  Анализ программы в целях генерации семантически эквивалентного машинного кода для непосредственного выполнения



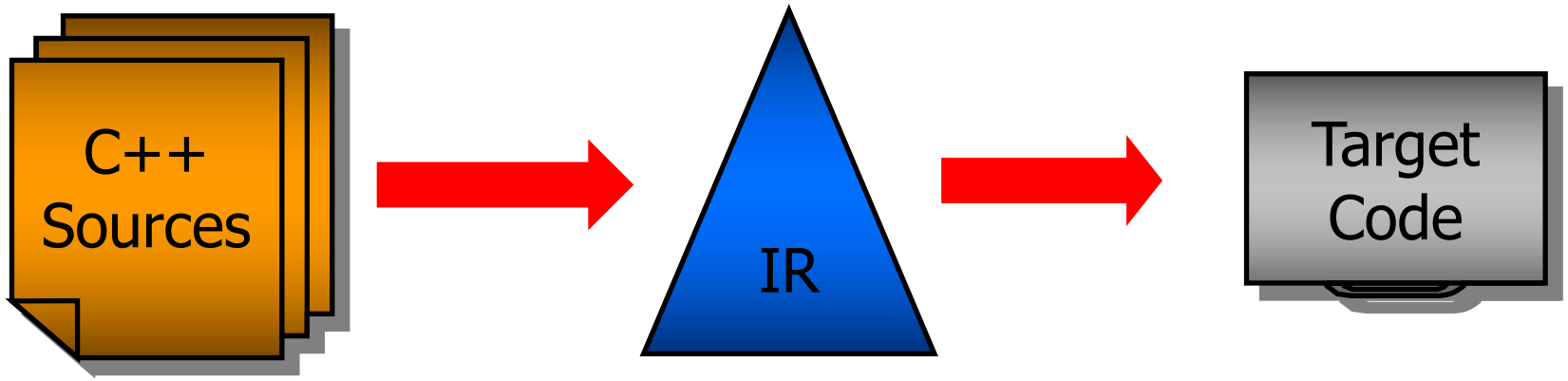
- Компиляция в узком смысле

Какова цель?

Какие задачи должен выполнять компилятор?

- **Генерация эффективного кода**

  Анализ программы в целях генерации семантически эквивалентного машинного кода для непосредственного выполнения



- Компиляция в узком смысле

Однако, генерация кода не является единственной целью – зачастую она даже не самая главная!

Почему компиляции в узком смысле недостаточно?

Имеется много актуальных задач, связанных с анализом программ, **не связанных с генерацией кода**:

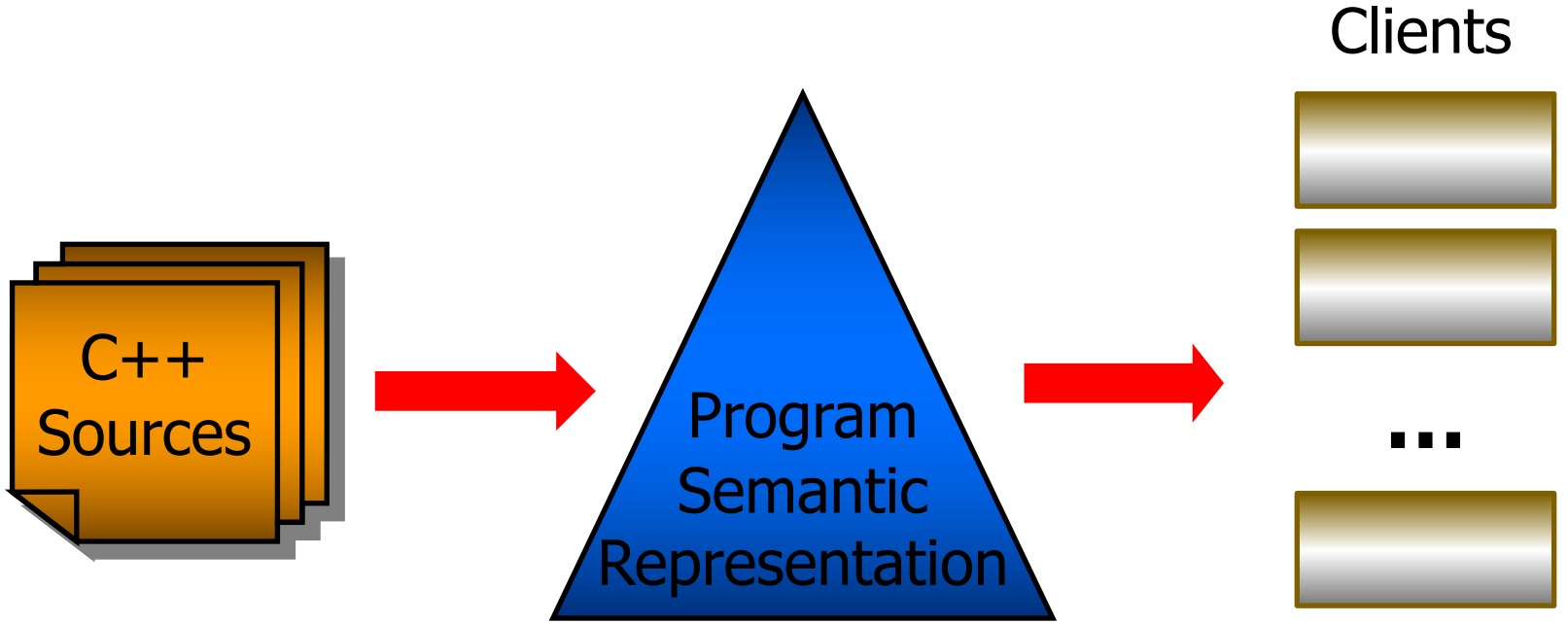
- **Реинжиниринг унаследованного кода**; автоматическая генерация программ.
- Сопровождение и развитие существующих программ.
- **Статический анализ программ**: детальная диагностика без выполнения кода; обнаружение уязвимостей и проблемного кода; оптимизации на уровне исходного текста; **рефакторинг**.
- «Понимание» programs; визуализация: генерация UML-диаграмм (reverse engineering), XREF-диаграмм; вычисление метрик.
- **Тестирование**; генерация тестовых покрытий.
- **Верификация** программ: формальное доказательство правильности; абстрактная интерпретация; частичные вычисления.
- **Интеграция компилятора и IDE**
- ...

Для решения этих задач необходима **полная информация о семантике программы**

Какова цель?

Какие задачи должен выполнять компилятор?

- **Анализ программ** с целью получения полной информации обо всех её свойствах



- Статические анализаторы
- Визуализаторы исх. кода
- Генераторы объектного кода
- Инструменты реинжиниринга
- Форматировщики
- ...

- Компиляция в широком смысле

Как эти цели достигаются?

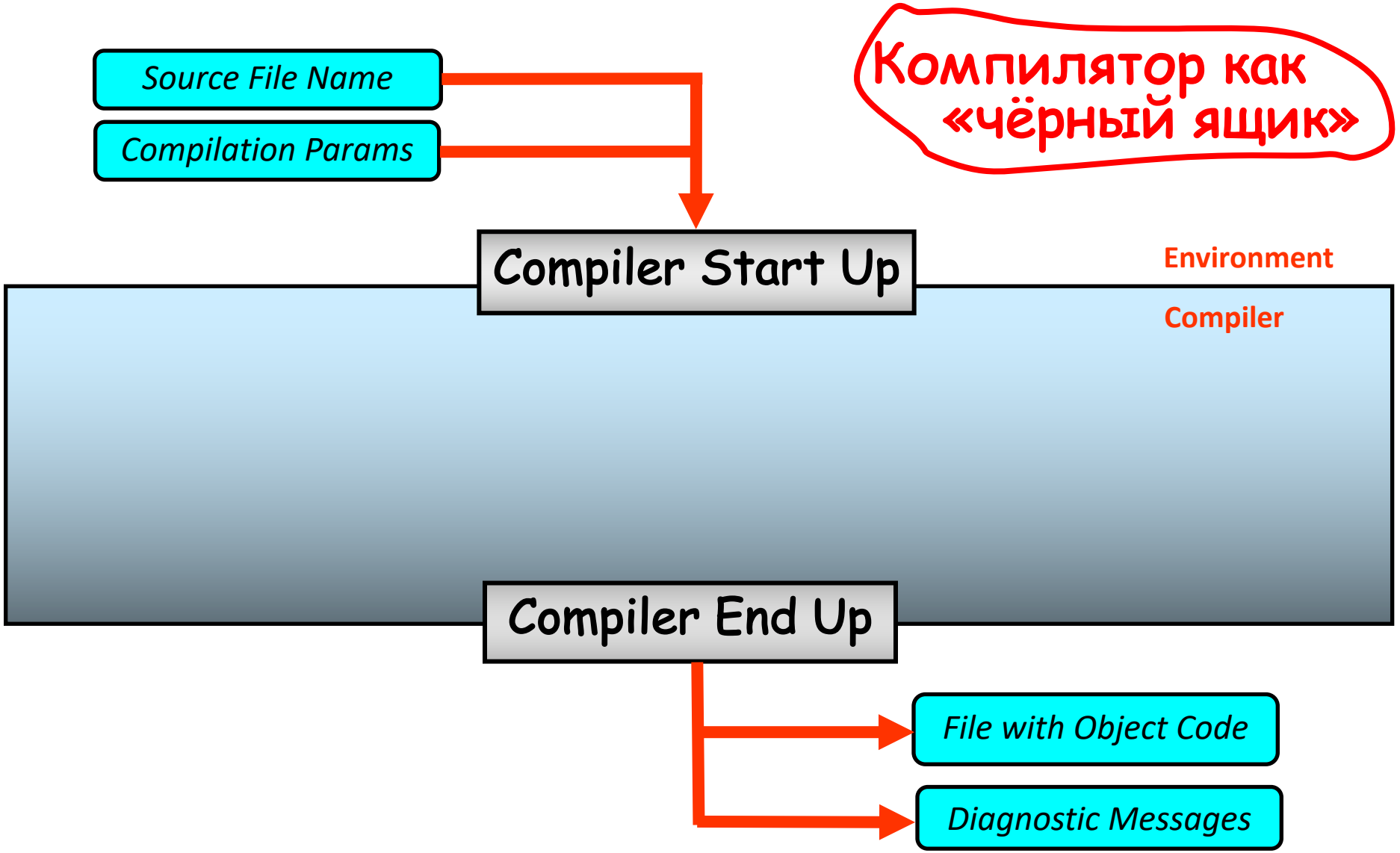
- Созданием **специализированных инструментов и систем**: от простых утилит вроде **lint** для C (или **lint++** для C++) до мощных и дорогих систем: **Klocwork**, **Fortify**, **Coverity**.

  Как правило, это либо утилиты командной строки, либо «герметичные» системы без возможности наращивания функциональности.

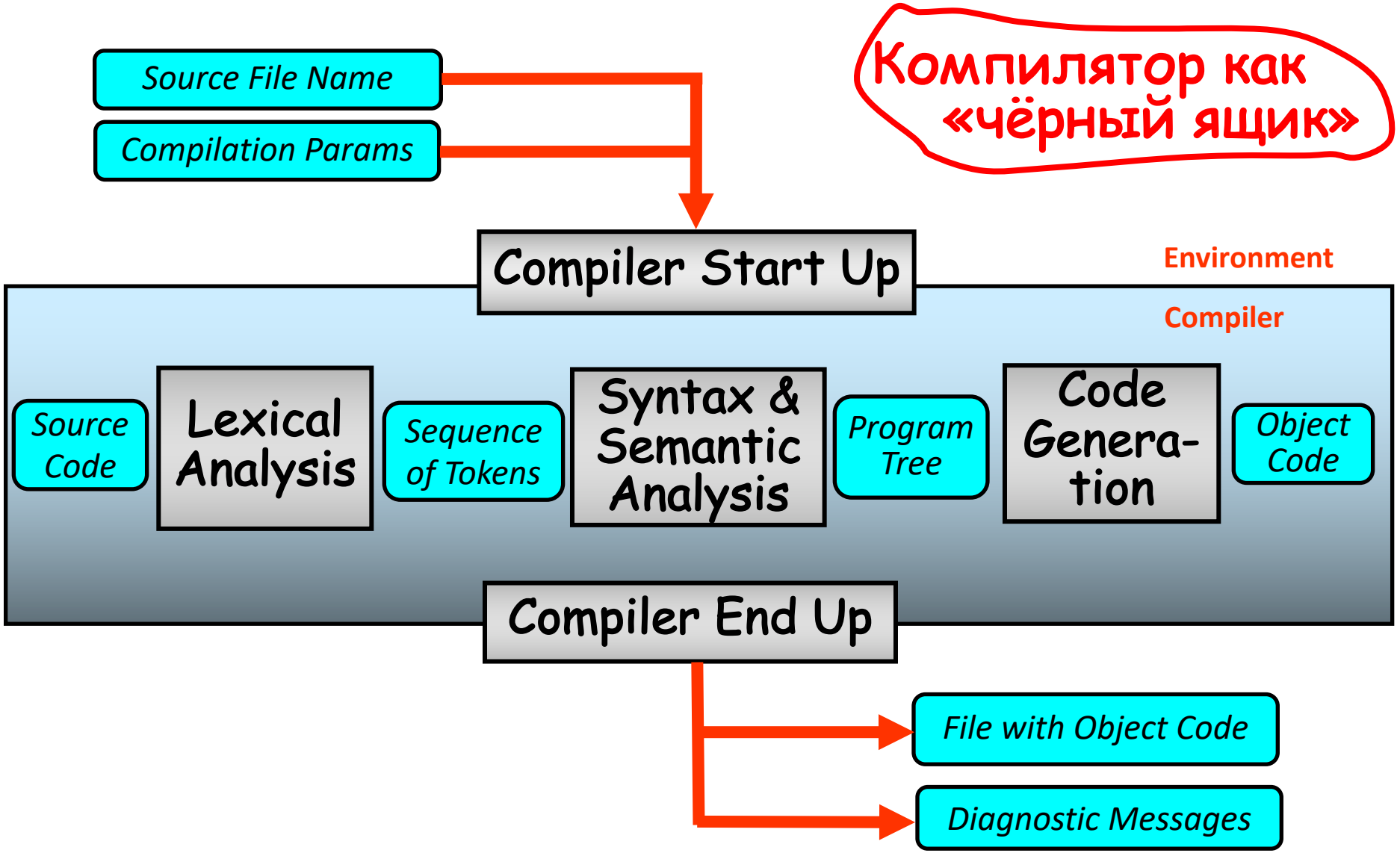
- Созданием **открытых инфраструктур** (API), предоставляющих доступ к семантическим свойствам программ.

  Примеры: Roslyn for C#, Clang for C++, инструменты JetBrains

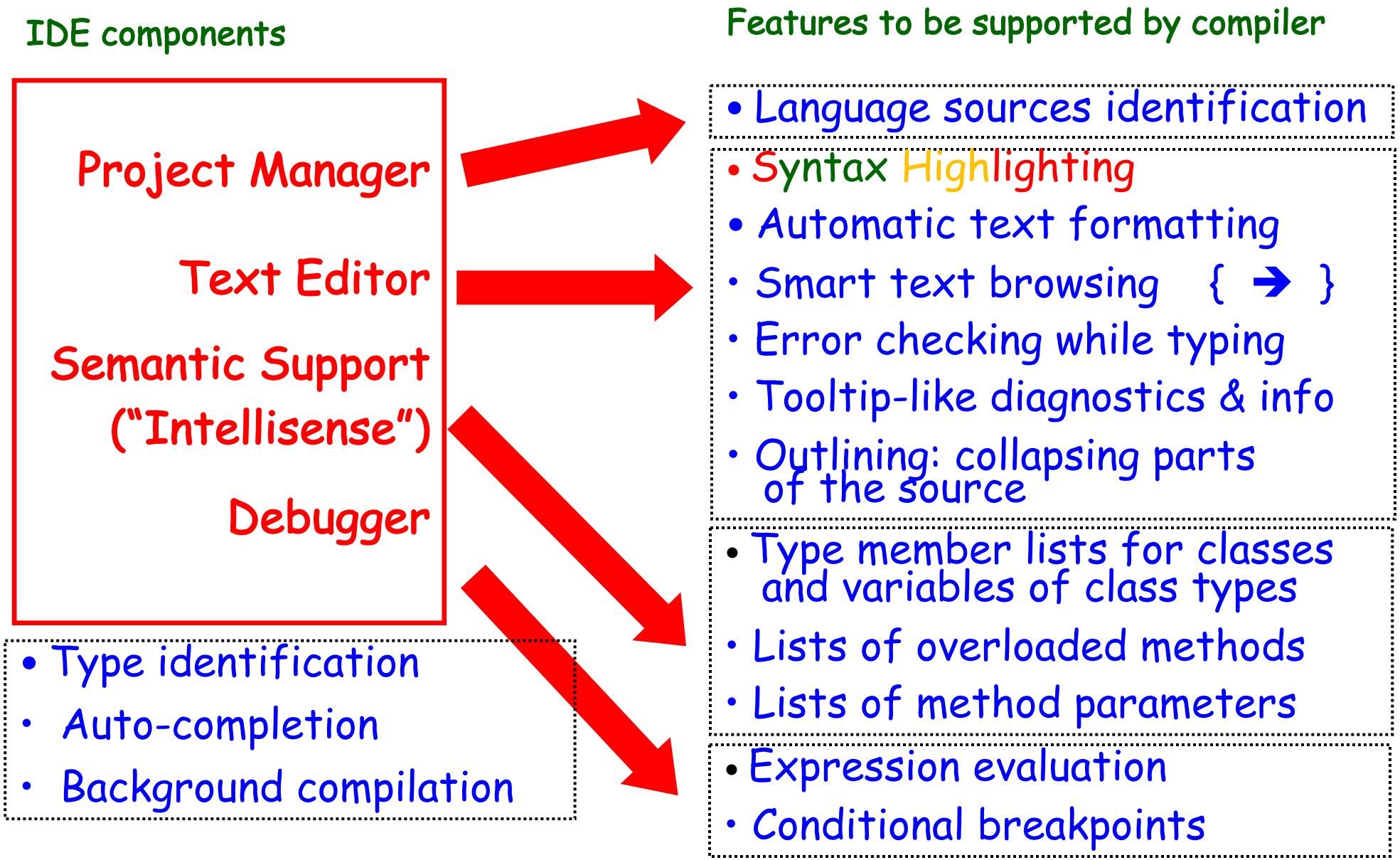
Компиляция: традиционный подход



Компиляция: традиционный подход



Зачем интегрировать компилятор в IDE?



Интеграция компилятора и IDE

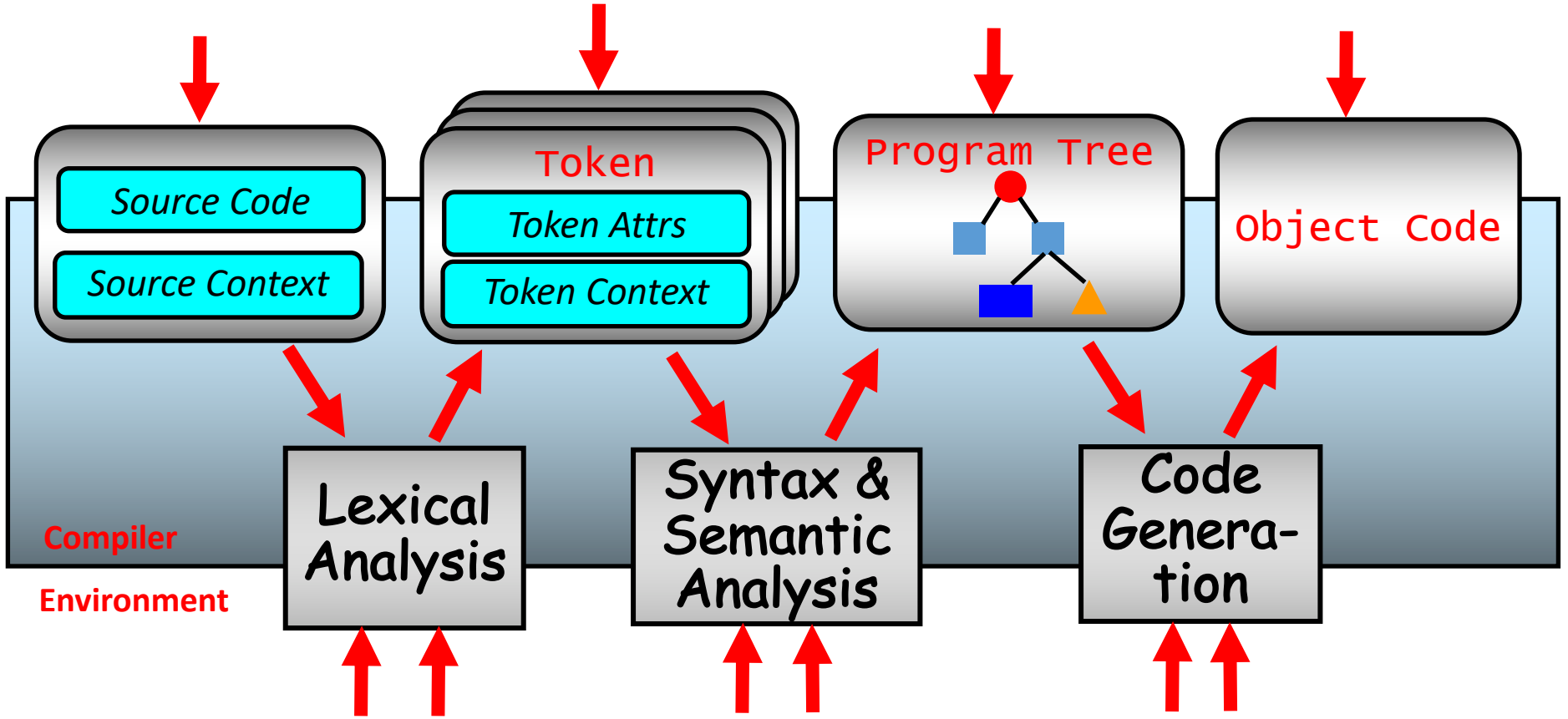
Пример: семантическая информация от компилятора C# используется в Visual Studio

```
// For that, taking tokens from the buffer until STRINGIZE_END appears.

Token previous = null;
Span begin = null, end = null;
string literal = "";
while ( true )
{
    token = this.get(); // Recursion 'cause we need macro processing
    forget();

    if ( begin == null )
        begin = token.sp
    if ( token.code ==                                    D ) break;

    if ( previous != n
        literal += " ";

    literal += token.image;
    previous = token;
    end = token.span;
}
if ( literal == "" )
{
    // There was an empty sequence STRINGIZE, STRINGIZE_END.
```

commentSpan
modifySpan
span

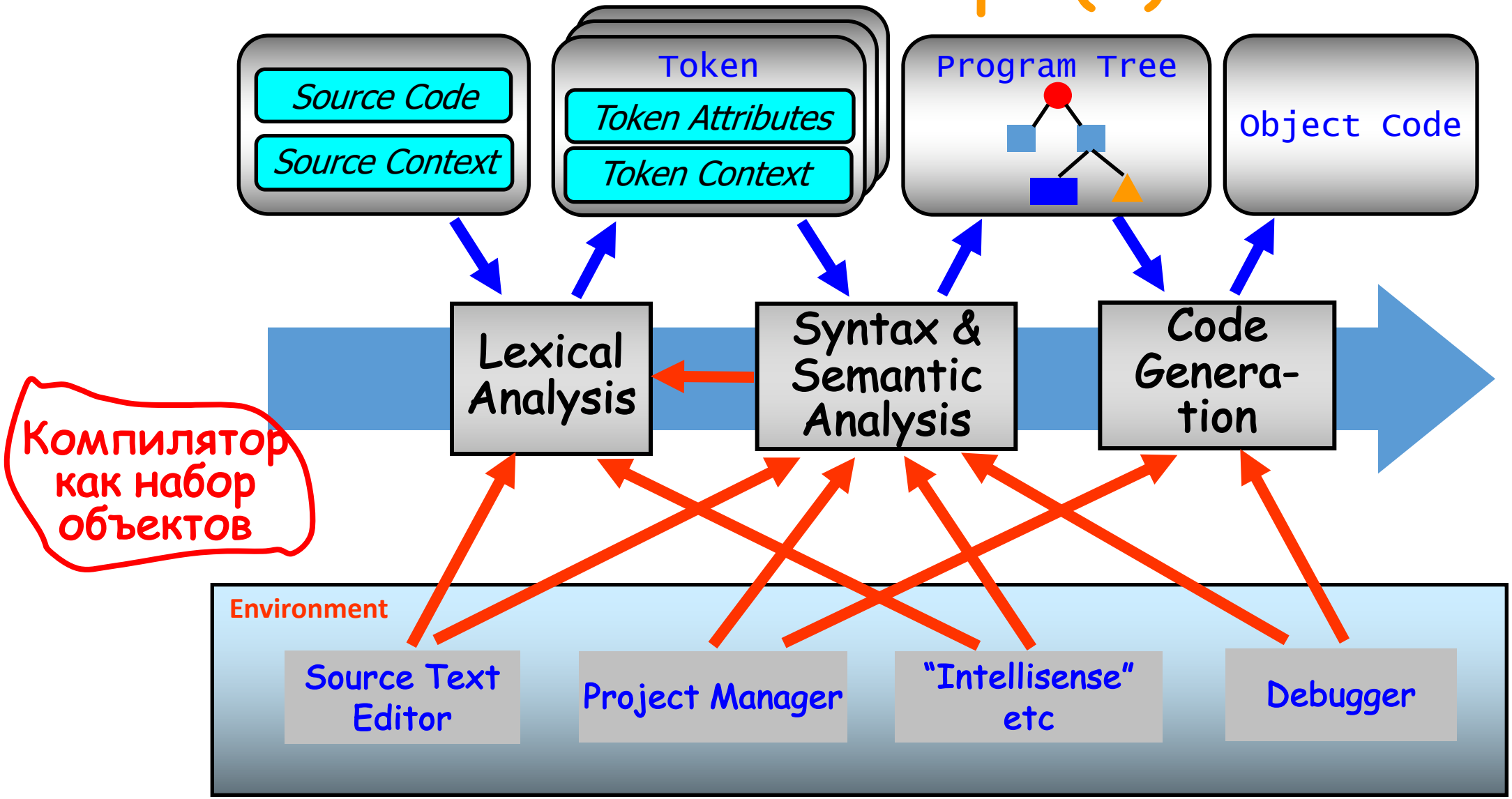
Span TokenBase.span
Textual coordinates of the original token (line/position of the beginning and the end of the token).

Задача интеграция и архитектура компилятора (1)



Компилятор как коллекция ресурсов

Задача интеграция и архитектура компилятора (2)



Тестирование компилятора

- **Подход Ada**
  Стандартный набор тестов для сертификации Ada-компиляторов:
  Ada Compiler Validation Capability (ACVC)
  Ada Conformity Assessment Test Suite (ACATS)
- **Подход gcc**
  Случайный набор тестов, сформированный из «баг-репортов» пользователей.
- **Наш подход**
  Собственный тестовый набор **Conformance Test Suite** для проверки соответствия компилятора стандарту ISO.
  ~6.500 test cases and growing

Импортозамещение !?

- Зачем импортозамещать?
- Что импортозамещать?
- Как импортозамещать, или free software: благо или …?

…И причём здесь компиляторы?

Зачем импортозамещать?

- «Закладки»?? Вредоносный код??
- Недокументированные возможности??

Зачем импортозамещать?

- «Закладки»?? Вредоносный код??
- Недокументированные возможности??
- Зависимость от производителя..
- Трудности сопровождения..

Зачем импортозамещать?

- «Закладки»?? Вредоносный код??
- Недокументированные возможности??
- Зависимость от производителя..
- Трудности сопровождения..
- **Ошибки!!**
- **Устаревающая архитектура!!**

Зачем импортозамещать?

Немного пафоса

Развитая сфера ИТ в 21 веке принадлежит к **фундаментальным**, **системообразующим** аспектам любого государства, претендующего на лидерство в мировом масштабе.

Наша страна, по всему комплексу исторических, географических, материальных и культурных причин, безусловно, должна принадлежать к категории мировых лидеров.

Без развитой **собственной** ИТ-индустрии мировое лидерство вряд ли возможно.

Что импортозамещать?

Прикладные

- Специализированные библиотеки
- Веб-браузеры, поисковики
- Словари/переводчики
- OCR
- Социальные сети
- Почта, мессенджеры
- …

? - да, конечно

Что импортозамещать?

Прикладные
- Специализированные библиотеки
- Веб-браузеры, поисковики
- Словари/переводчики
- OCR
- Социальные сети
- Почта, мессенджеры
- …

? - да, конечно

В первую очередь !

Системные
- Операционные системы
- Средства разработки: **языки, компиляторы, IDE**
- Базы данных
- …

Free Software, или Как импортозамещать?

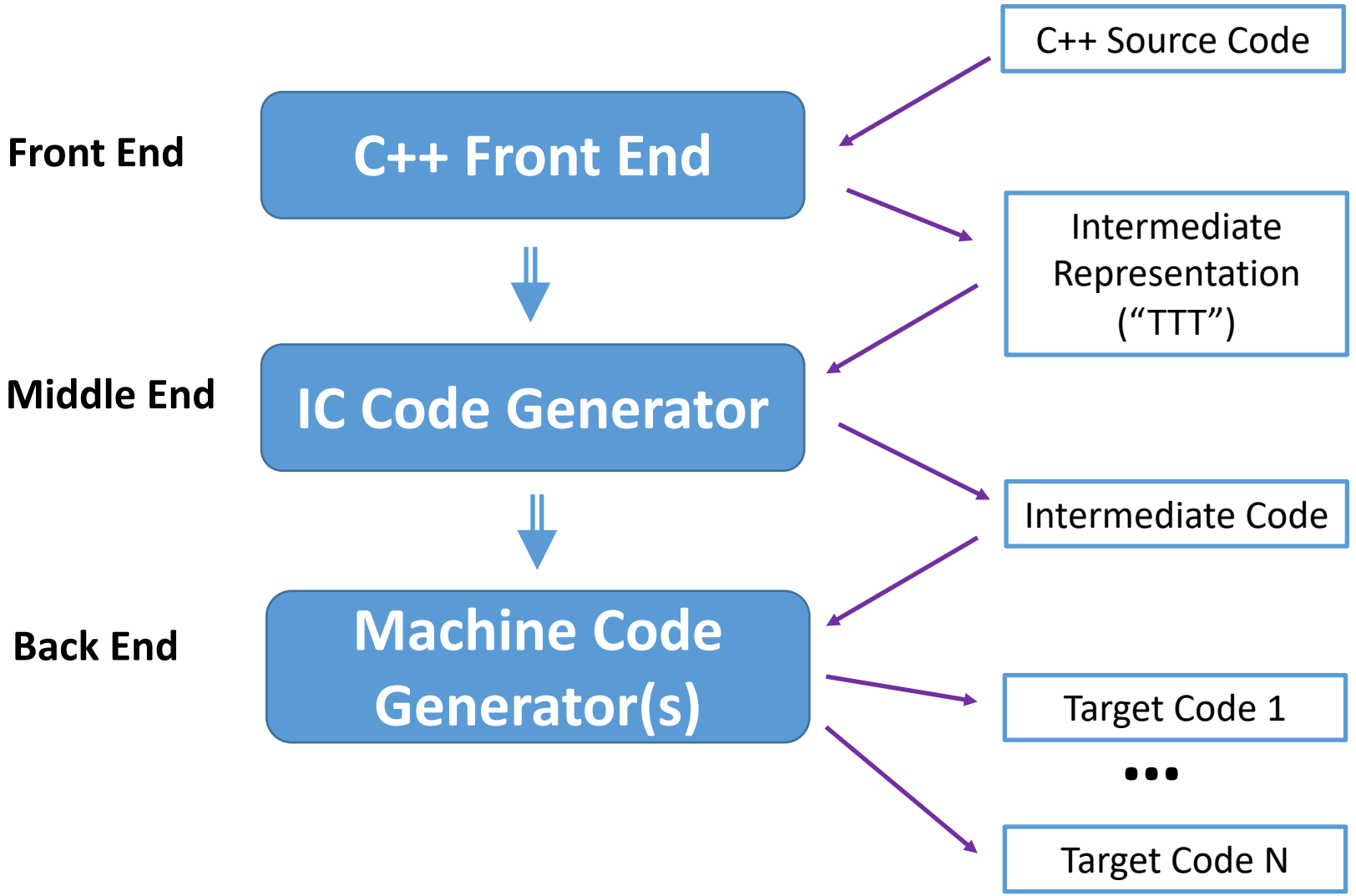
- Копировать «свободный» исходный код ИТ-систем – обрекать себя на пожизненную **зависимость** от архитектурных и технических решений, принятых другими, и на необходимость внесения сиюминутных правок, предлагаемых другими: «**permanent followers**».
- Не столь важно, является ли исходный код открытым – должны быть открытыми **интерфейсы** (Н.Вирт).
- Не заимствовать исходный код, а **перенимать удачные архитектурные и технические решения** – и воплощать их в собственных разработках.
- Важный фактор востребованности ИТ-решения (в том числе, экспортных перспектив) – следование **международным стандартам** (которые открыты).

Дополнения

Компиляция C++: основные проектные решения

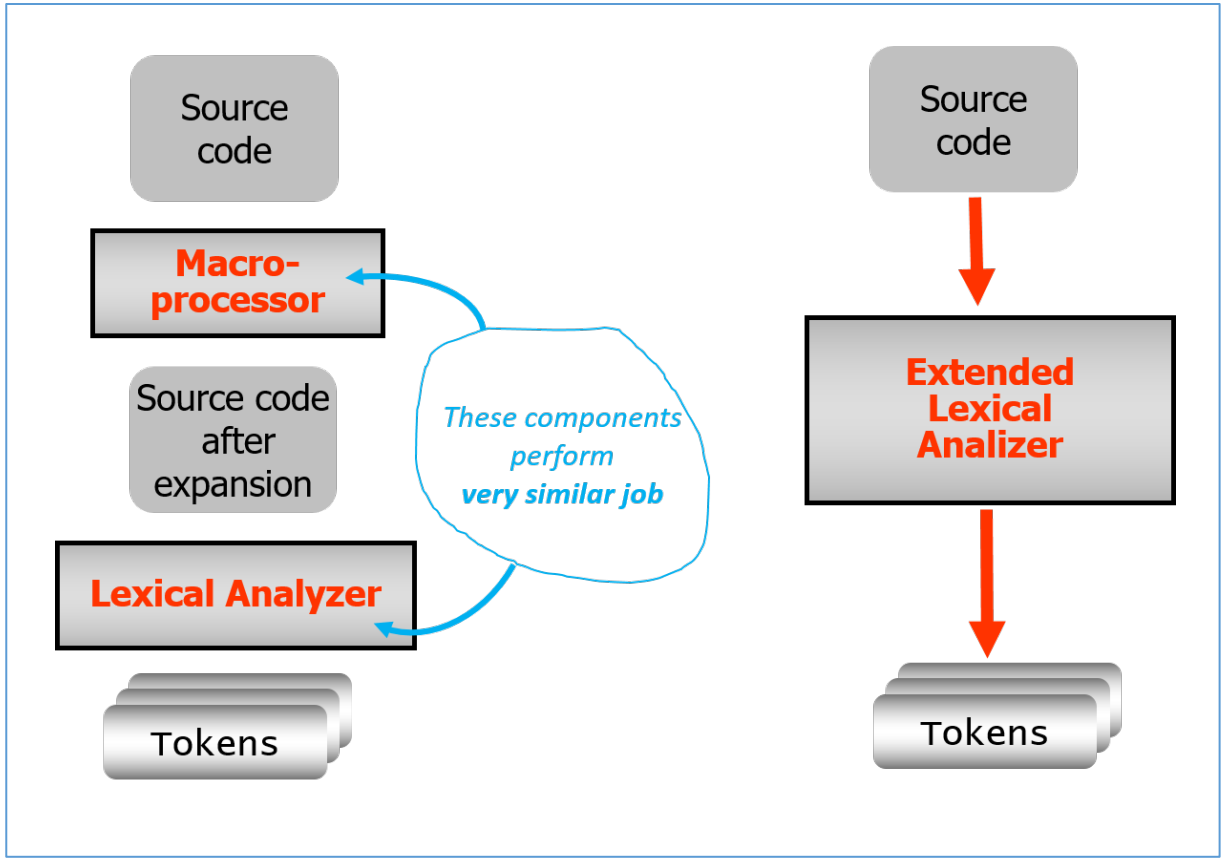
- **Только собственный код**, без использования «free software» или какого-либо проприетарного кода: компилятор написан **from scratch.**
- Один проход по исходному тексту.
- Лексическая структура C++ переработана («суперлексемы» вместо обычных лексем).
- Решение проблемы неоднозначности: разрешение имён **на ранних фазах** компиляции.
- **Высокий уровень внутреннего представления**, с сохранением всей семантической информации.
- **Smart linking**: связывание модулей на основе семантической информации.
- **Тестовый пакет** создавался одновременно с компилятором.

Архитектура компилятора



Лексический анализ

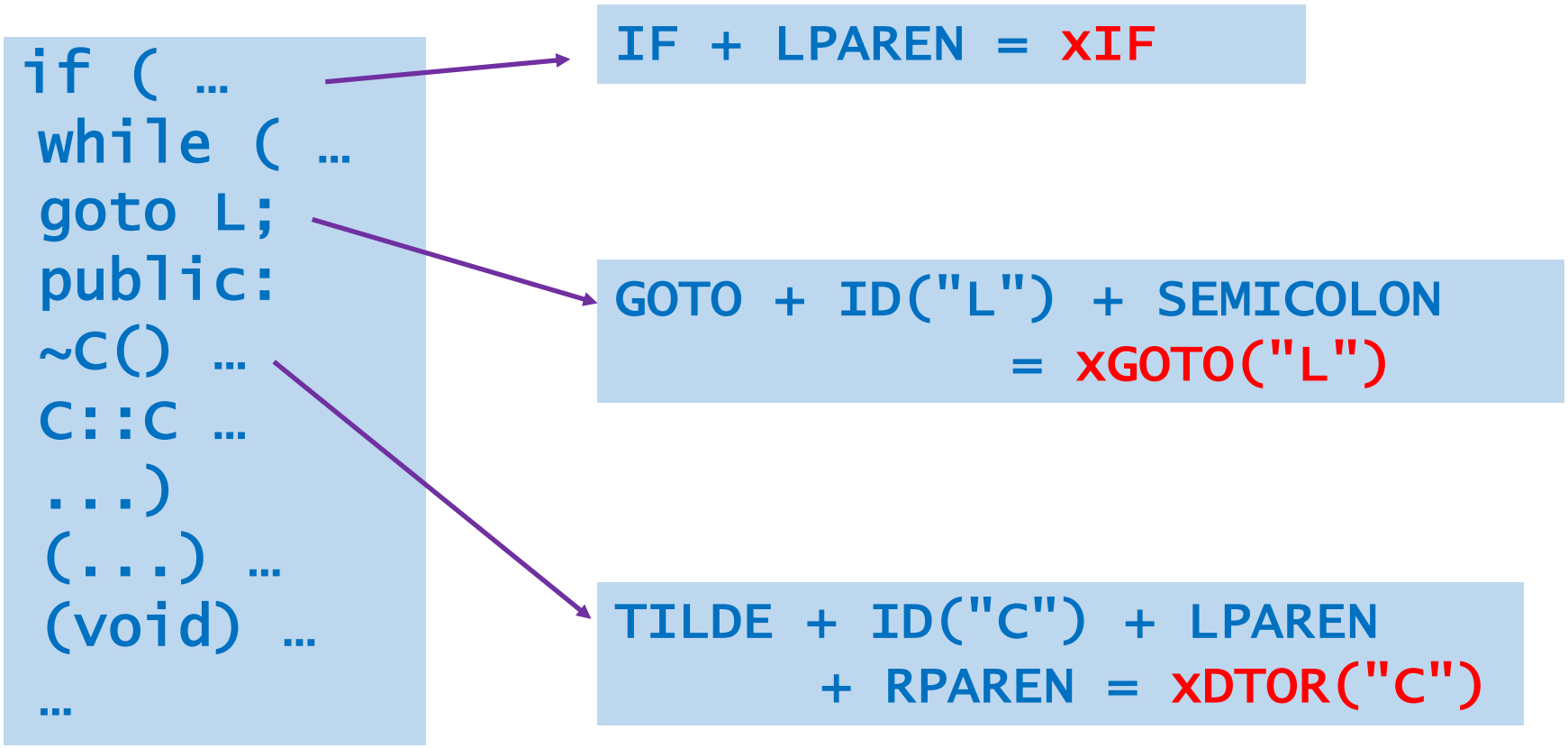
- Объединение фаз препроцессирования и собственно лексического анализа: **Extended Lexan**.
- **Разработан вручную**: без использования *lex* или какого-либо средства автом. генерации.
- Лексическая структура C++ была переработана («суперлексемы» вместо обычных лексем).
- Разрешение имён **на этапе лексического анализа**.



ДЗ

Лексический анализ

Лексемы vs «Суперлексемы»



Синтаксический анализ

- Реализован на основе формального описания грамматики C++; генерация распознавателя инструментом ***bison***.
- В определении грамматики используется **много трюков** с целью преодоления синтаксических проблем (неоднозначности).
- Некоторые грамматические неоднозначности разрешаются **на ранних фазах**: на этапе лексического анализа.
- Рассматриваются перспективы переработки парсера: использовать рекурсивный спуск вместо генерации парсера из формальной грамматики.

Синтаксические трюки

```
int a = 0;
class C {
public:
   void f() { a = 7; }
   int a;
};
int main() {
    C c;
    c.f();
    cout << a; // 0 or 7?
}
```

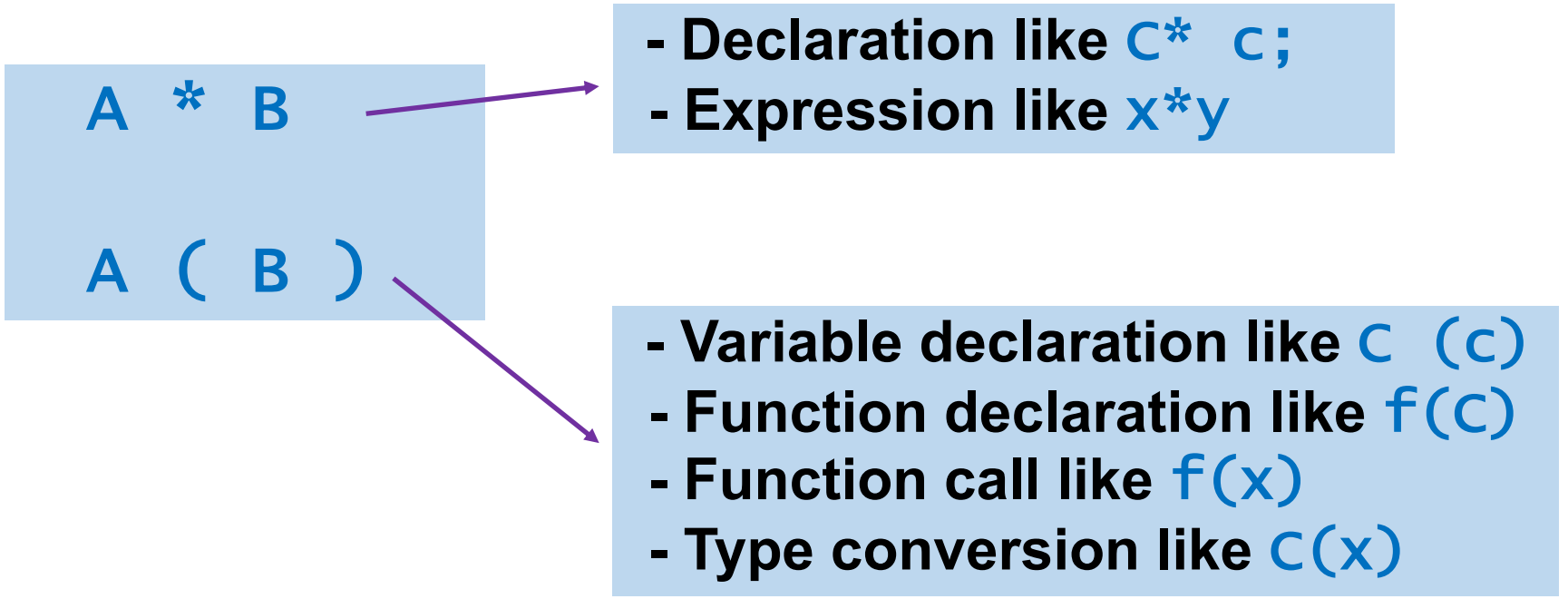
Стандарт:

- **Тела функций-членов транслируются в контексте всего класса**
- **Объявление класса считается полным, когда достигнута завершающая лексема его тела: "}"**

Синтаксический анализ (4)

Синтаксические неоднозначности: что это?

(в предположении, что A и B - идентификаторы)



Как разрешить неоднозначности:

- На основе семантической идентификации имен в процессе **лексического анализа.**

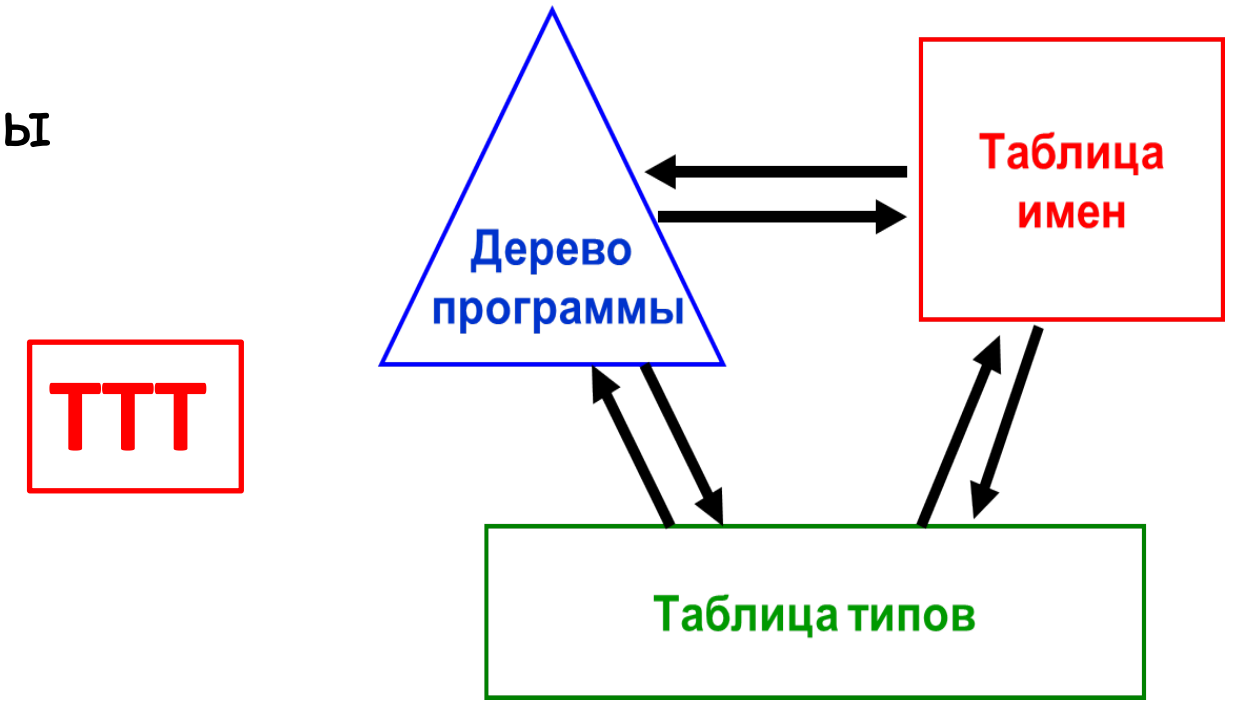
Промежуточное представление

Для чего:

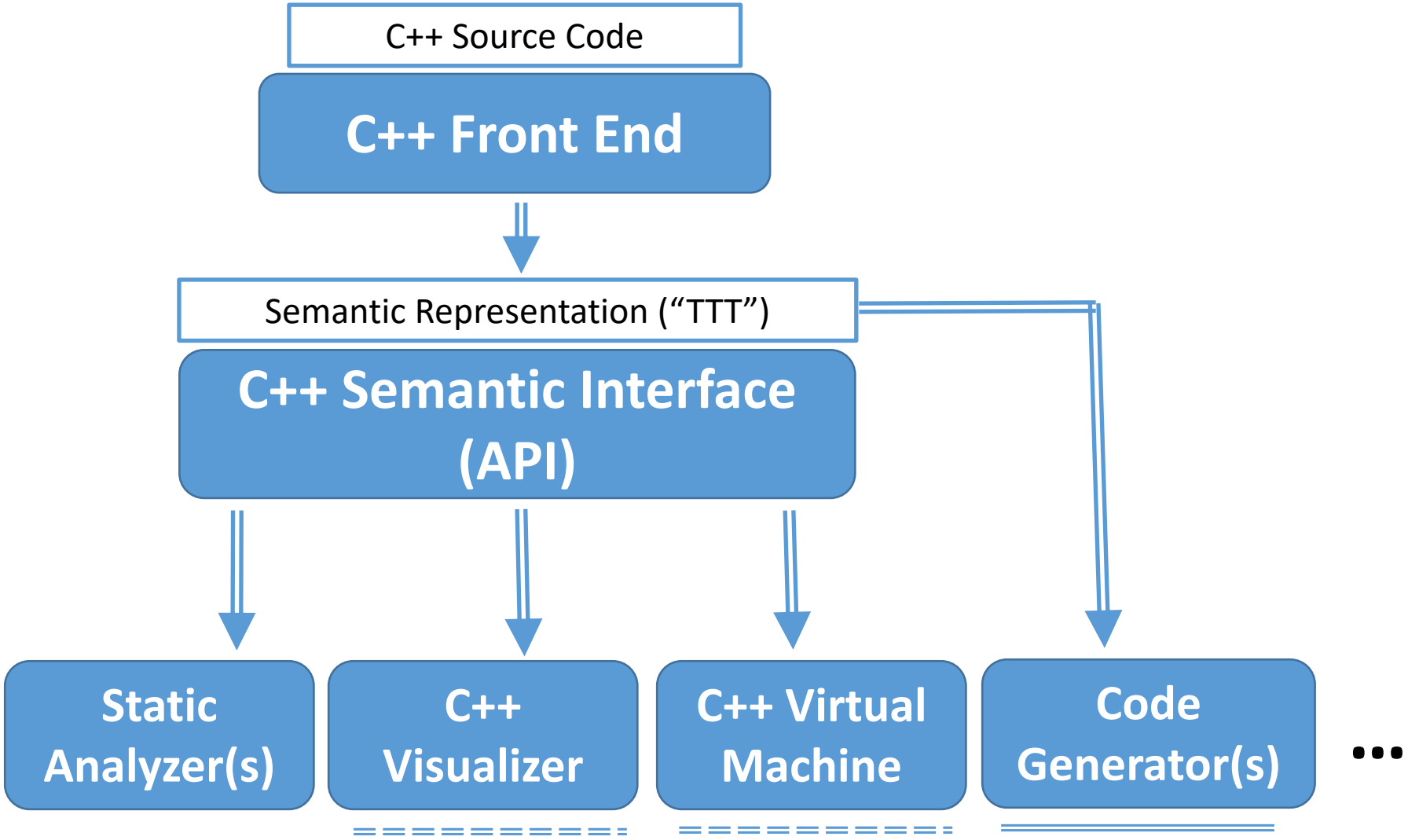
Представить всю семантическую информацию, извлеченную из исходного текста программы, в виде, удобном для анализа и обработки.

Три категории языковых сущностей:

- Объявления объектов
- Выражения & операторы
- Типы



Компилятор: продвинутая архитектура



Компилятор: Summary

- The size:
  - 58 MB, > 5400 files, > 460000 SLOC
- Backend approximately half
- Supported Standards:
  - ANSI C89 , C99 (partial)
  - C++ 2011
  - GNU extensions (partial)
  - Microsoft C++ (COM support)
  - Borland C++ (VCL support)
- Hosted on Windows (2000, XP, 8/10), Linux
- Built with Visual Studio, GCC 4.x

Vendor	Model	Notes
Unicore	UNC 320	32 bit / RISC
Unicore	UNC 80xx	8 bit
Progress	UNICON	16 bit / CISC
Elvees	MultiCore	32 bit MIPS + DSP
Module	NeuroMatrix	32 bit VLIW + Matrix
Progress	KVARC	32 bit / RISC