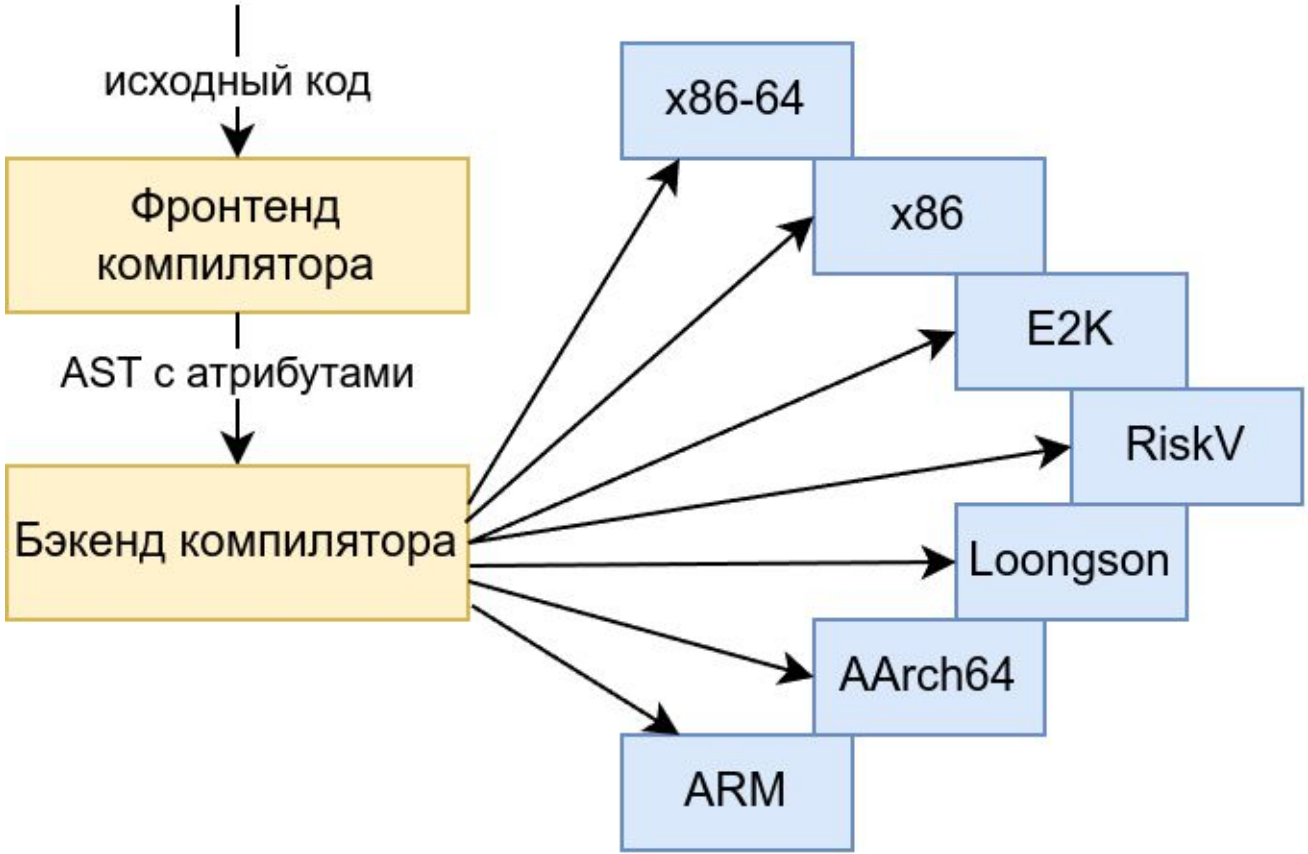
Проект «Компилятор»

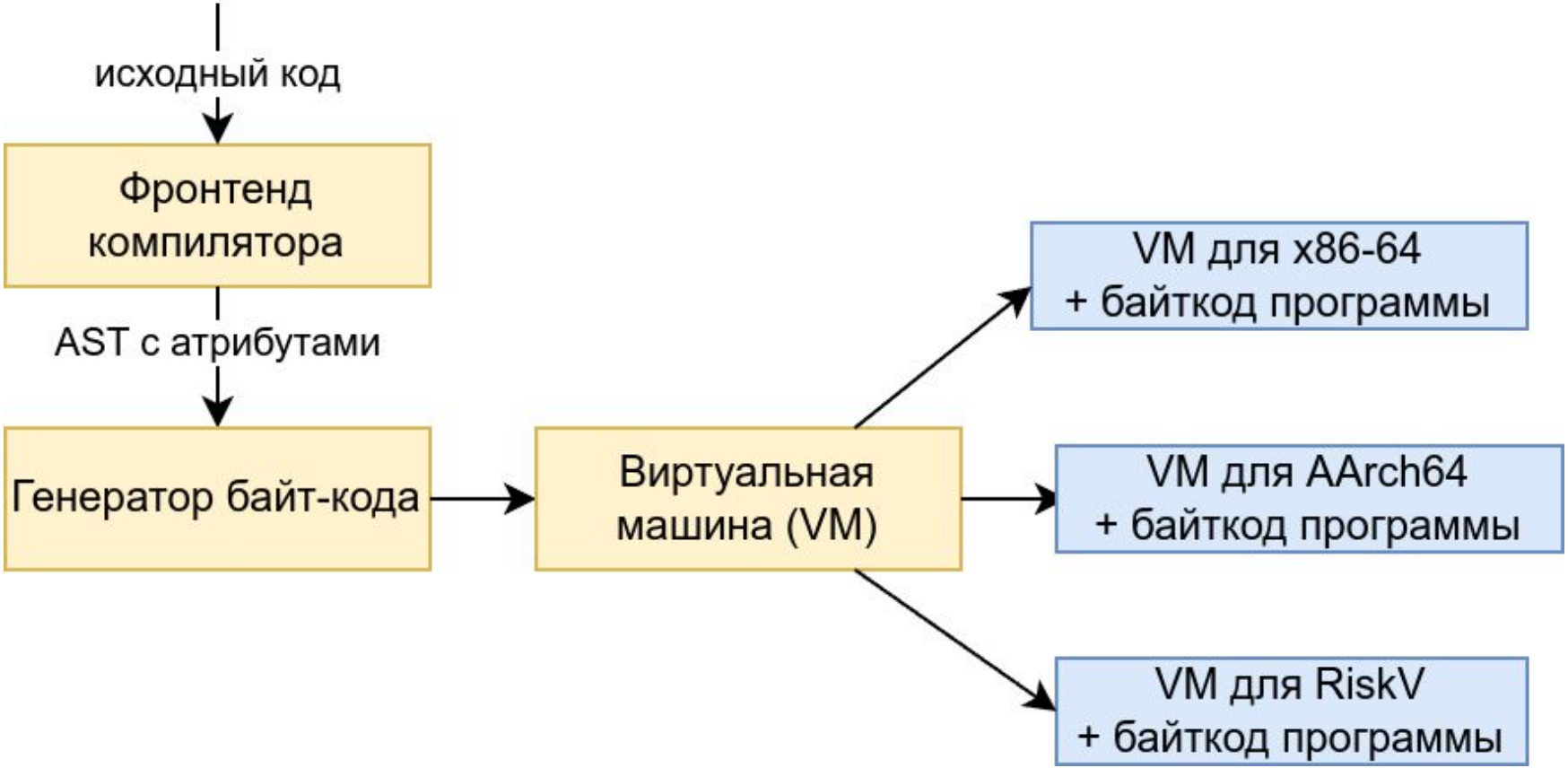
Лекция №1

Проблема: сколько нужно бэкендов компилятору?

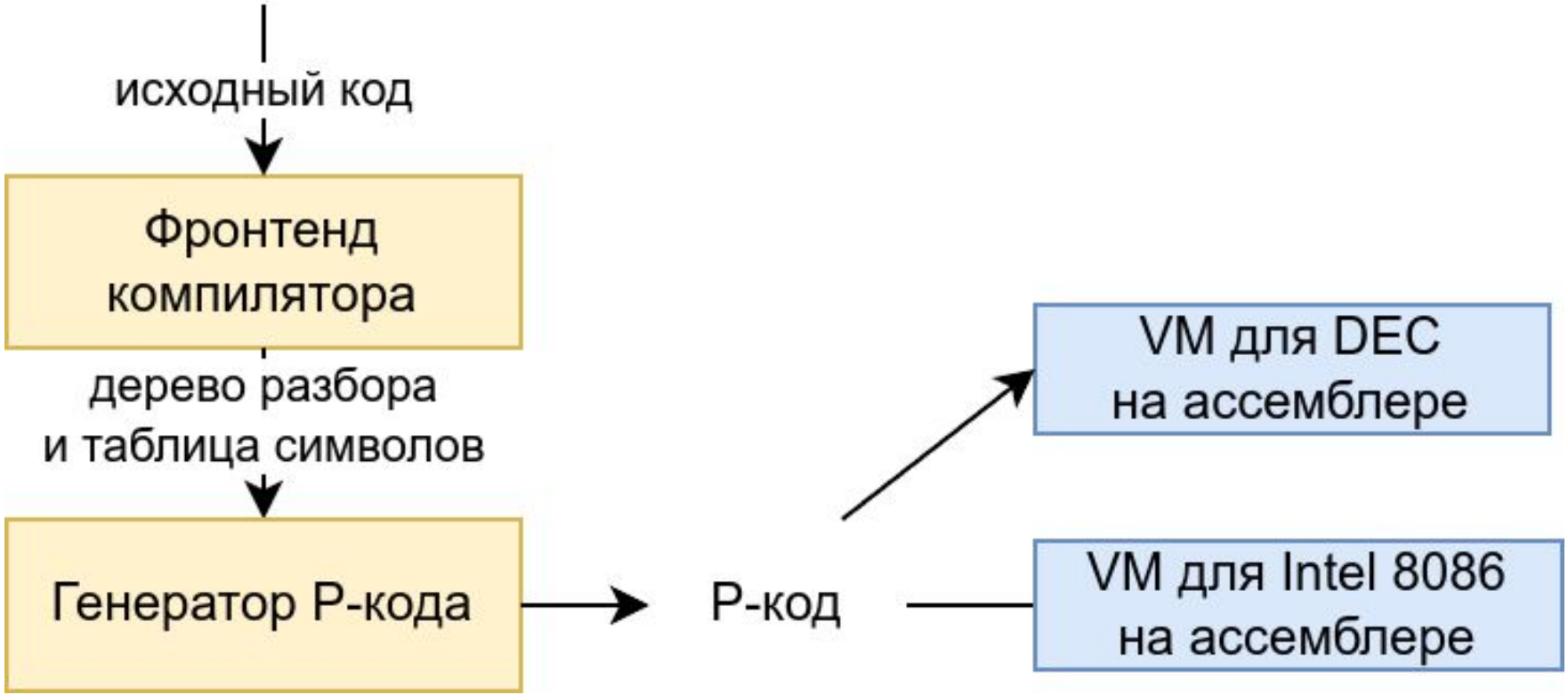


Решение №1:
виртуальная машина

Виртуальная машина



Компиляция Pascal в P-код (1970-е)



Какие проблемы решает виртуальная машина

1. Сокращается код для поддержки разных целевых платформам
    - Один генератор из AST в байт-код виртуальной машины
    - Много реализаций виртуальной машины на ассемблере

Какие проблемы решает виртуальная машина

1. Сокращается код для поддержки разных целевых платформам
    - Один генератор из AST в байт-код виртуальной машины
    - ~~Много реализаций виртуальной машины на ассемблере~~
    - Одна реализация виртуальной машины на C/C++

Какие проблемы решает виртуальная машина

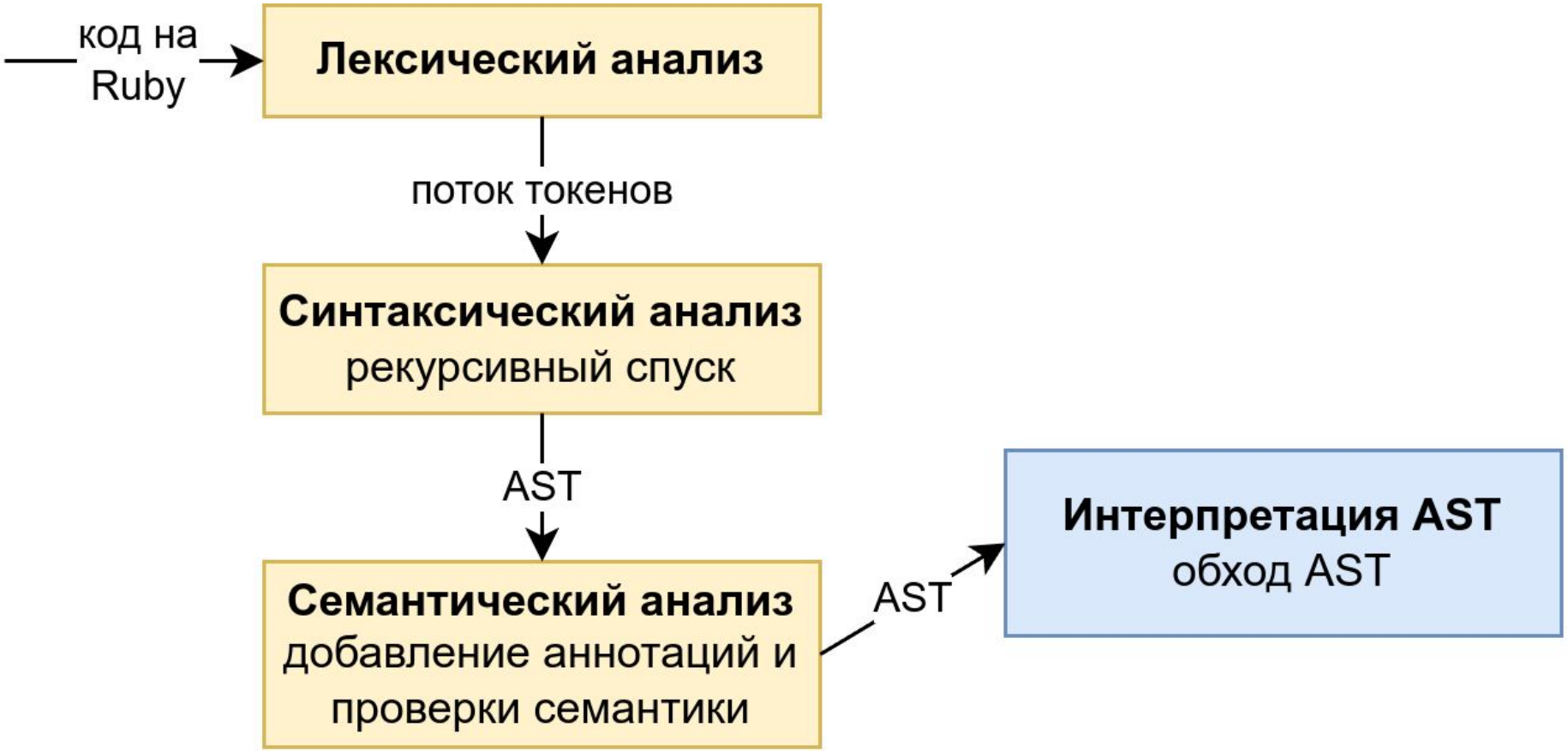
1. Сокращается код для поддержки разных целевых платформам
    - Один генератор из AST в байт-код виртуальной машины
    - Одна реализация виртуальной машины на C/C++
2. Можно доставлять один байт-код на все платформы
    - Один бинарный файл программы вместо файлов под CPU и OS

Какие проблемы решает виртуальная машина

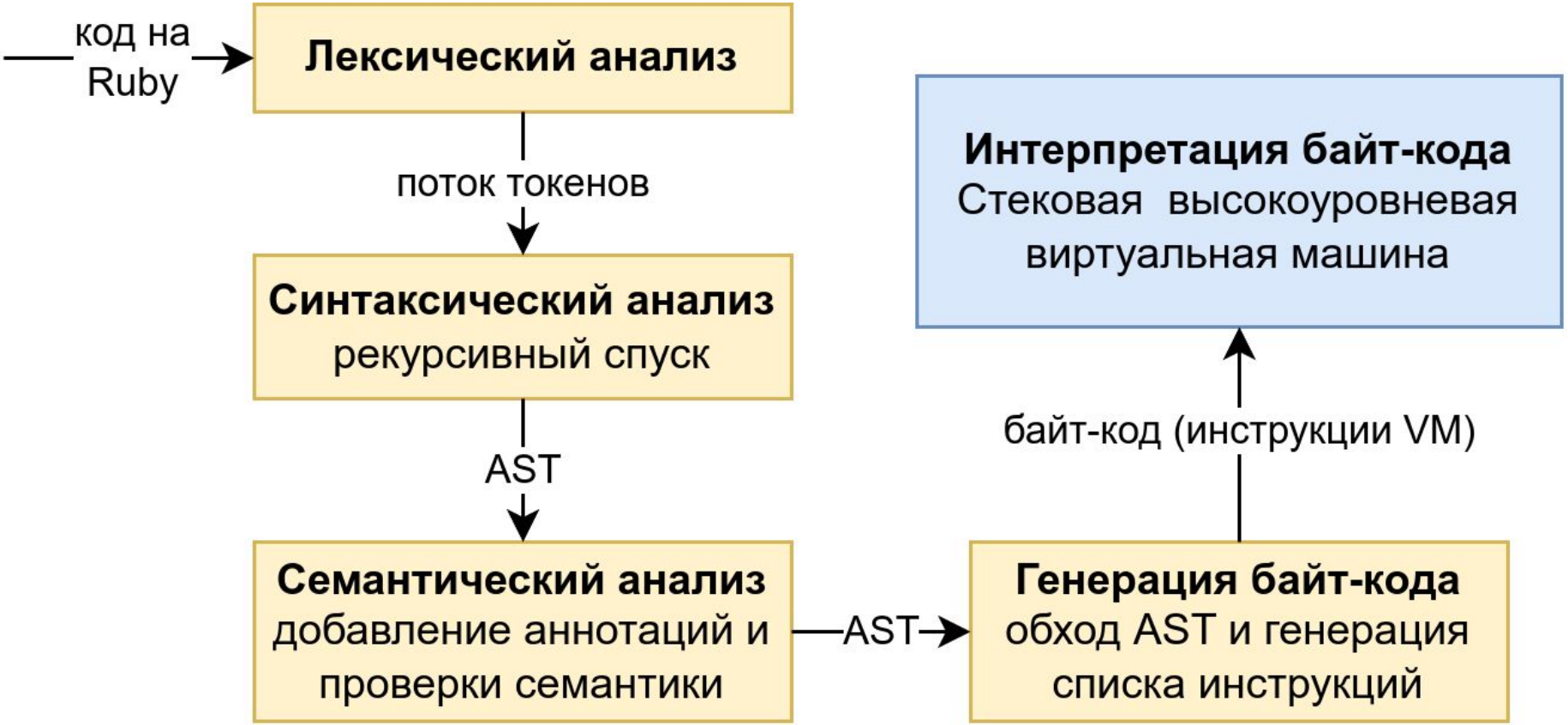
1. Сокращается код для поддержки разных целевых платформам
    - Один генератор из AST в байт-код виртуальной машины
    - Одна реализация виртуальной машины на C/C++
2. Можно доставлять один байт-код на все платформы
    - ~~Один бинарный файл программы вместо файлов под CPU и OS~~
    - Простым пользователям нужны бинарные файлы под конкретные CPU и OS с виртуальной машиной внутри (self-contained)

Интерпретаторы
с виртуальной машиной

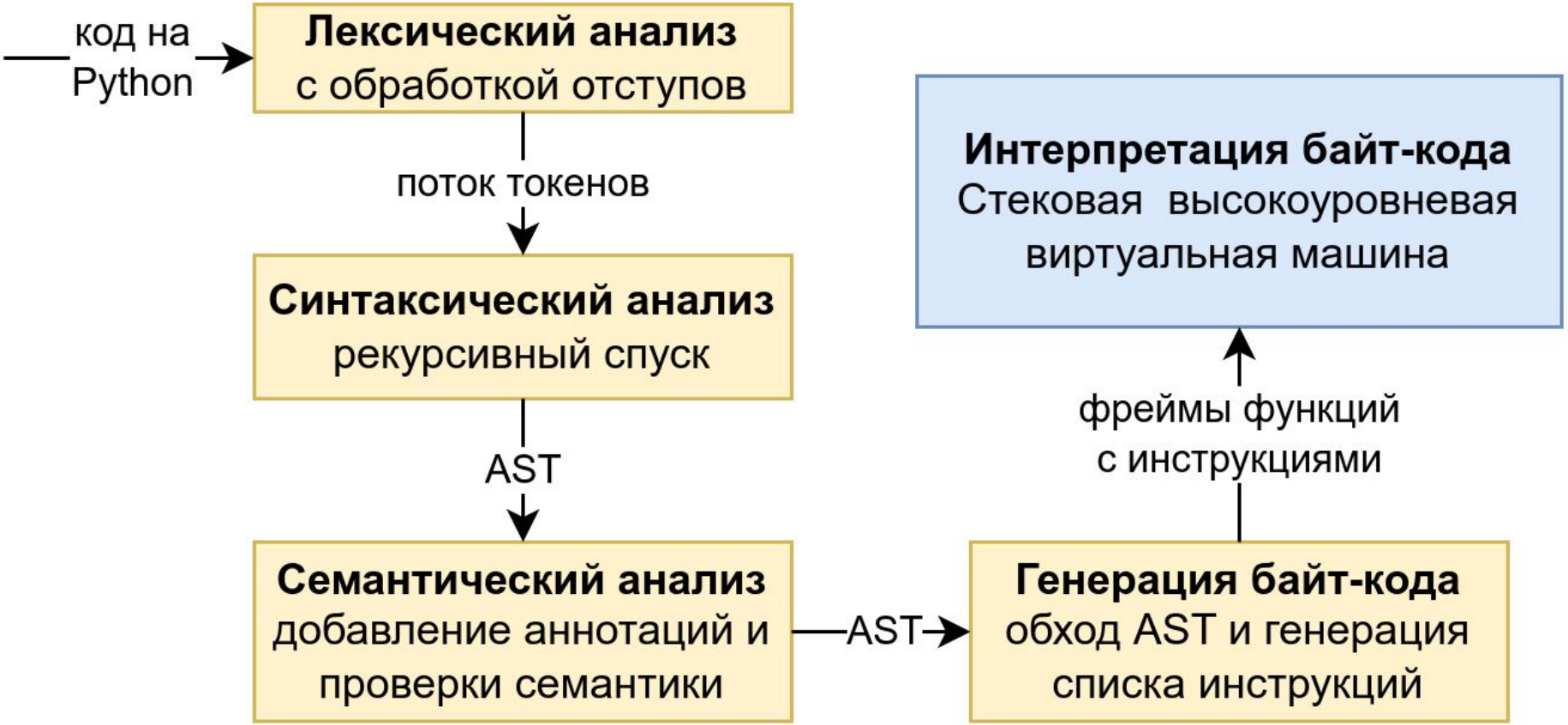
Старый интерпретатор Ruby (версии до 1.9)



Новый интерпретатор Ruby (с версии 1.9)



Интерпретатор CPython

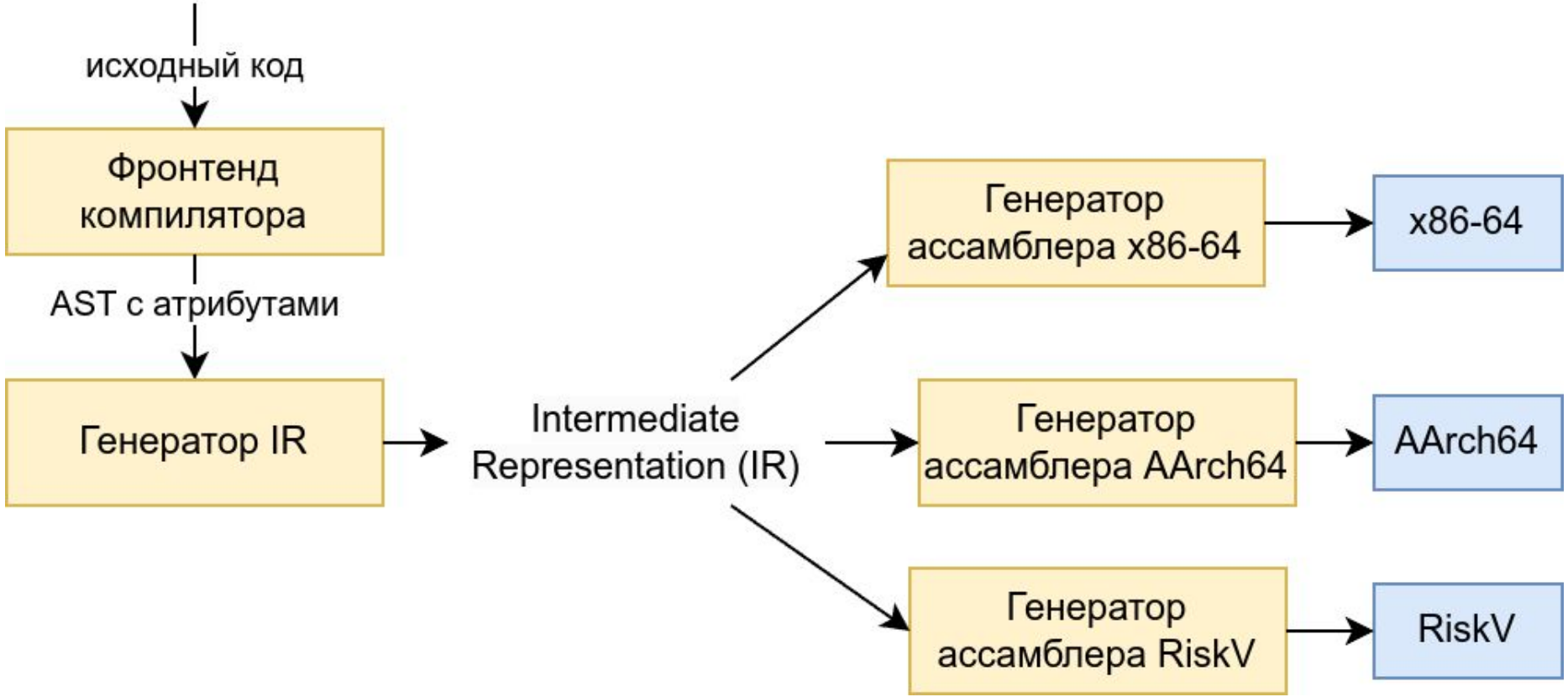


Зачем интерпретатору нужна VM?

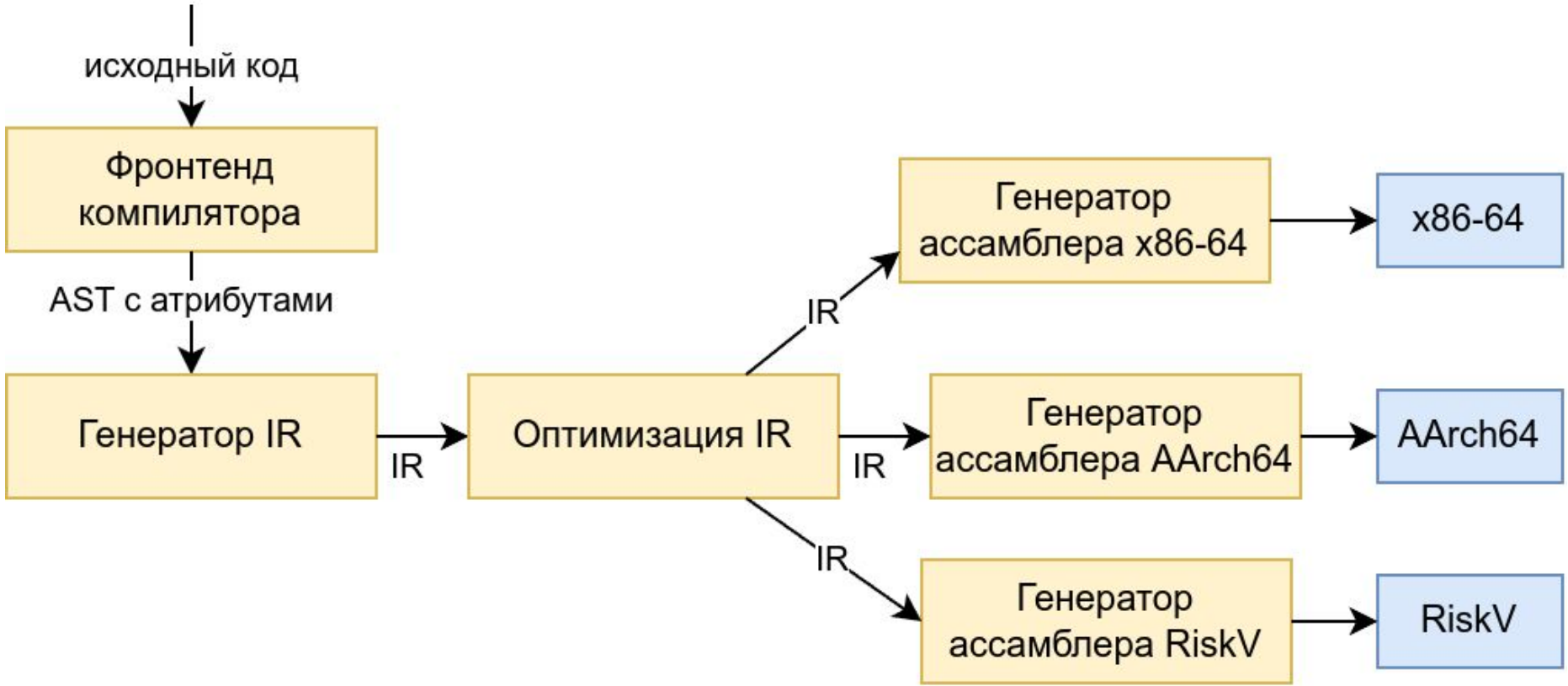
1. Интерпретация байт-кода эффективнее, чем интерпретация AST
    - Лучше локальность данных
    - Меньше вызовов функций
    - Меньше проверок
2. Ряд конструкций языка станет проще
    - break / continue / return / exit
    - присваивания переменных
    - *… и другие*

Решение №2: промежуточный код

Промежуточный код в бэкенде компилятора



Оптимизации промежуточного кода



Какие проблемы решает промежуточный код

1. Сокращается код для поддержки разных целевых платформам
    - Один генератор из AST в IR
    - Один оптимизатор IR
    - Много генераторов из IR в машинный код
2. IR удобнее для оптимизаций
    - Лучше всего IR в форме SSA (Static Single-Assignment form)
    - Лучше всего два IR — High-Level IR и Machine-Level IR

Архитектура .NET

Сравним VM или IR

Виртуальная машина (VM)

- Компилятор создаёт байт-код
- Виртуальная машина исполняет байт-код (интерпретация)

Промежуточный код (IR)

- Компилятор создаёт промежуточный код
- Бэкенд компилирует IR в машинный код (компиляция)

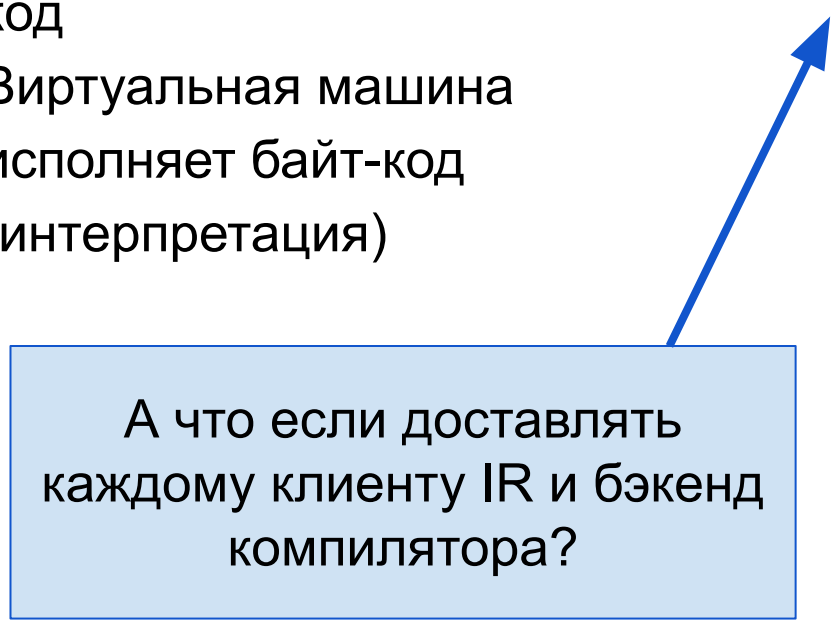
Сравним VM или IR

Виртуальная машина (VM)

- Компилятор создаёт байт-код
- Виртуальная машина исполняет байт-код (интерпретация)

Промежуточный код (IR)

- Компилятор создаёт промежуточный код
- Бэкенд компилирует IR в машинный код (компиляция)



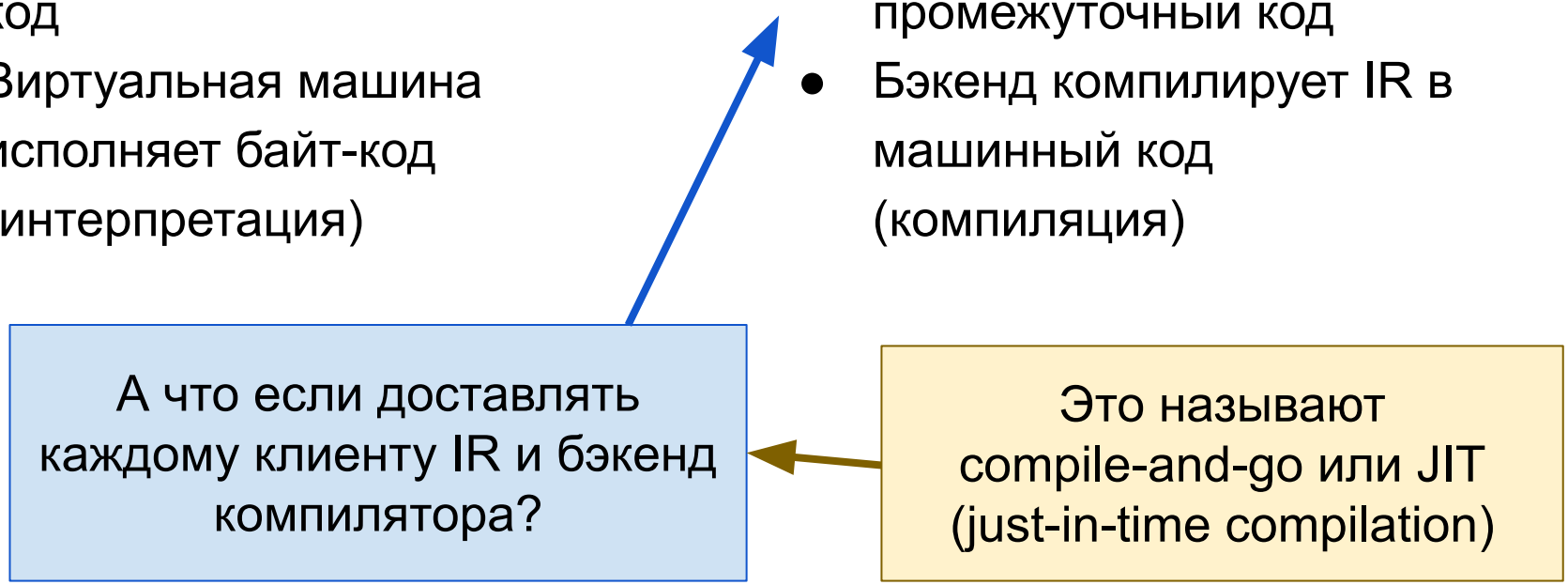
Сравним VM или IR

Виртуальная машина (VM)

- Компилятор создаёт байт-код
- Виртуальная машина исполняет байт-код (интерпретация)

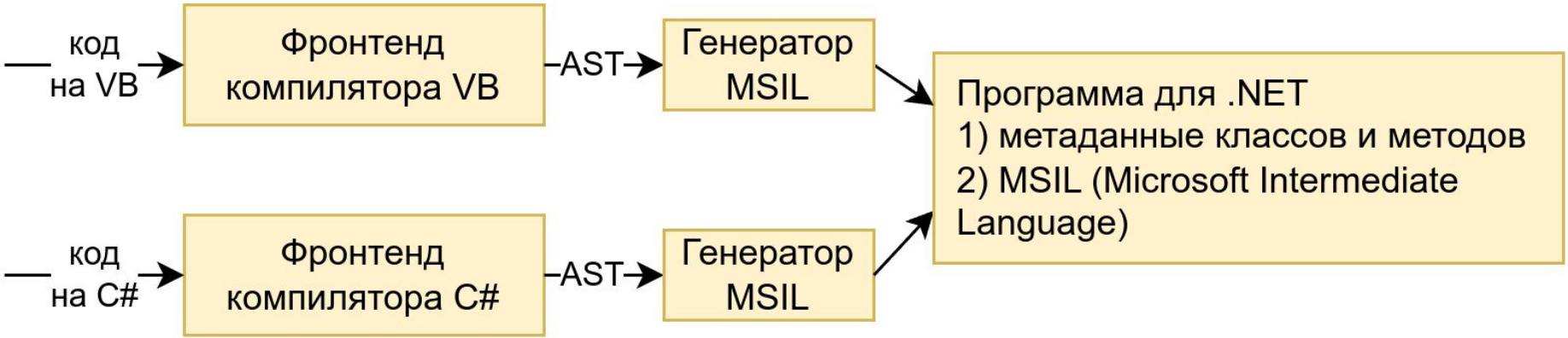
Промежуточный код (IR)

- Компилятор создаёт промежуточный код
- Бэкенд компилирует IR в машинный код (компиляция)

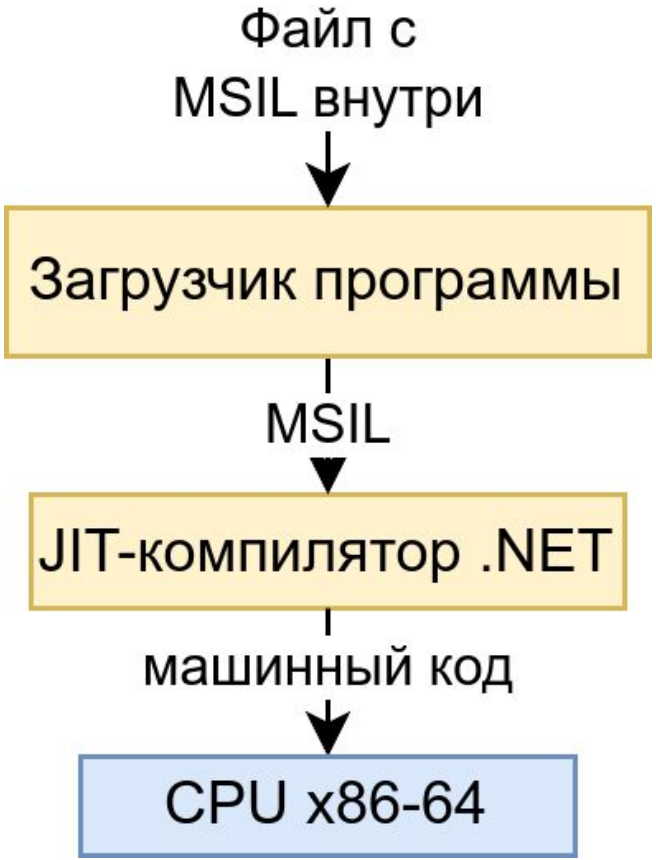


Архитектура .NET

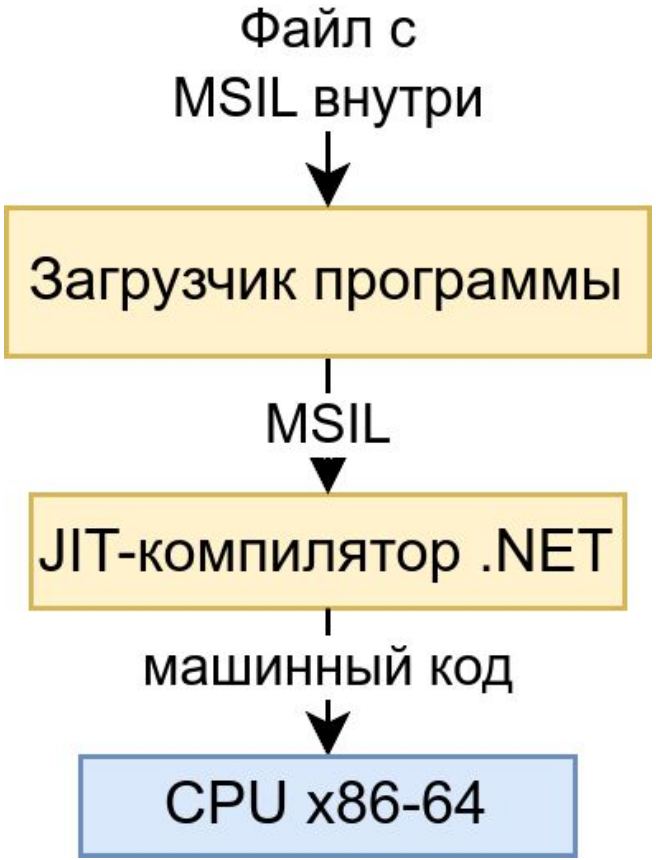
Компиляторы для .NET



Виртуальная машина .NET



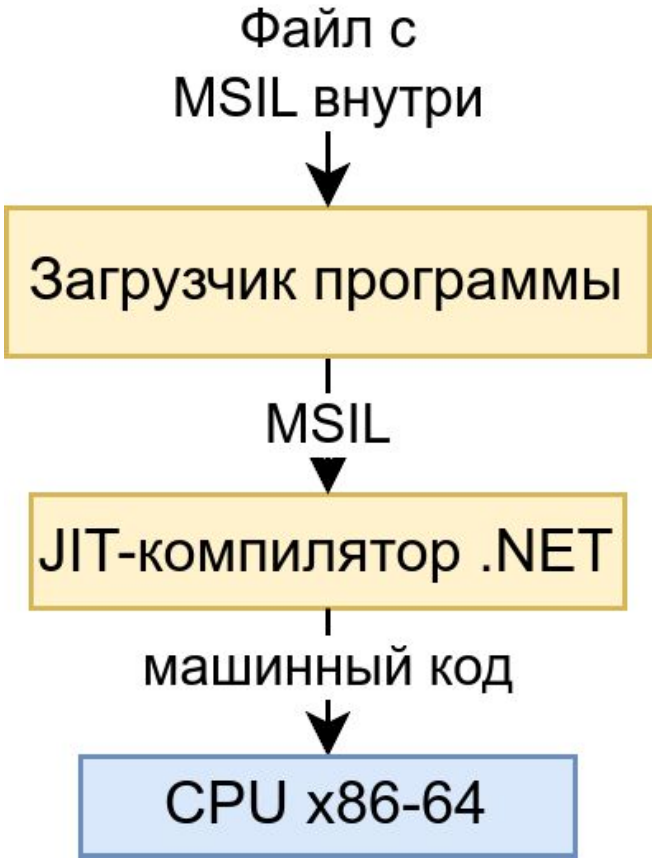
Виртуальная машина .NET



При запуске программы на .NET

1. Загрузчик помещает MSIL в память — для каждого метода создаётся заглушка

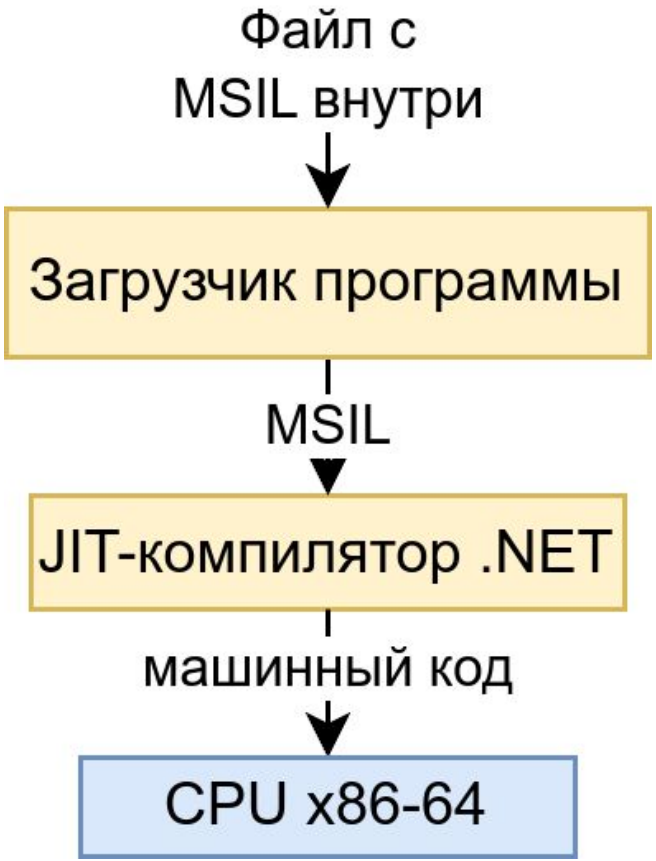
Виртуальная машина .NET



При запуске программы на .NET

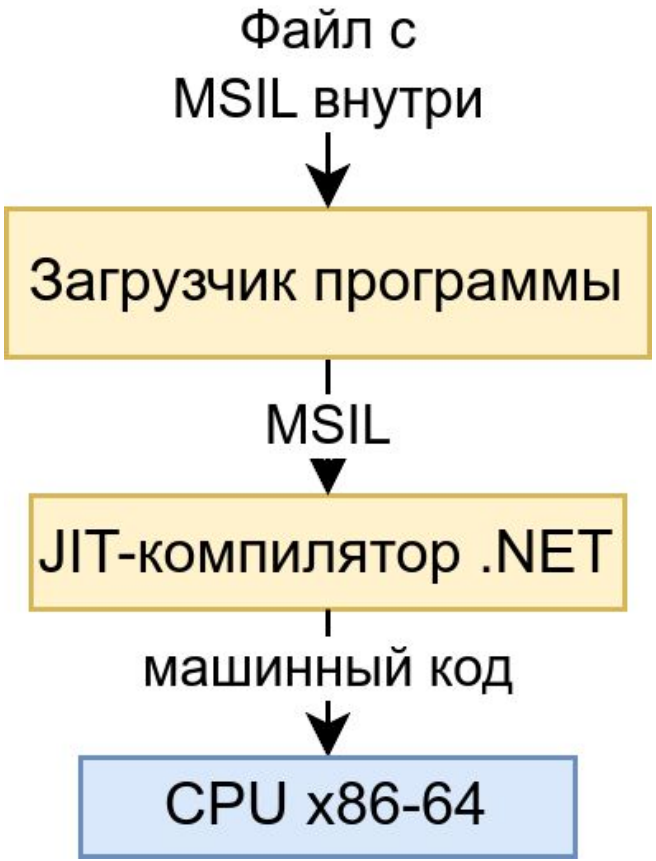
1. Загрузчик помещает MSIL в память — для каждого метода создаётся заглушка
2. Когда выполнение достигает заглушки:
   - Происходит компиляция MSIL в машинный код
   - Заглушка заменяется машинным кодом

Загрузчики для .NET



В Windows поддержка .NET встроена в загрузчик ОС

Загрузчики для .NET



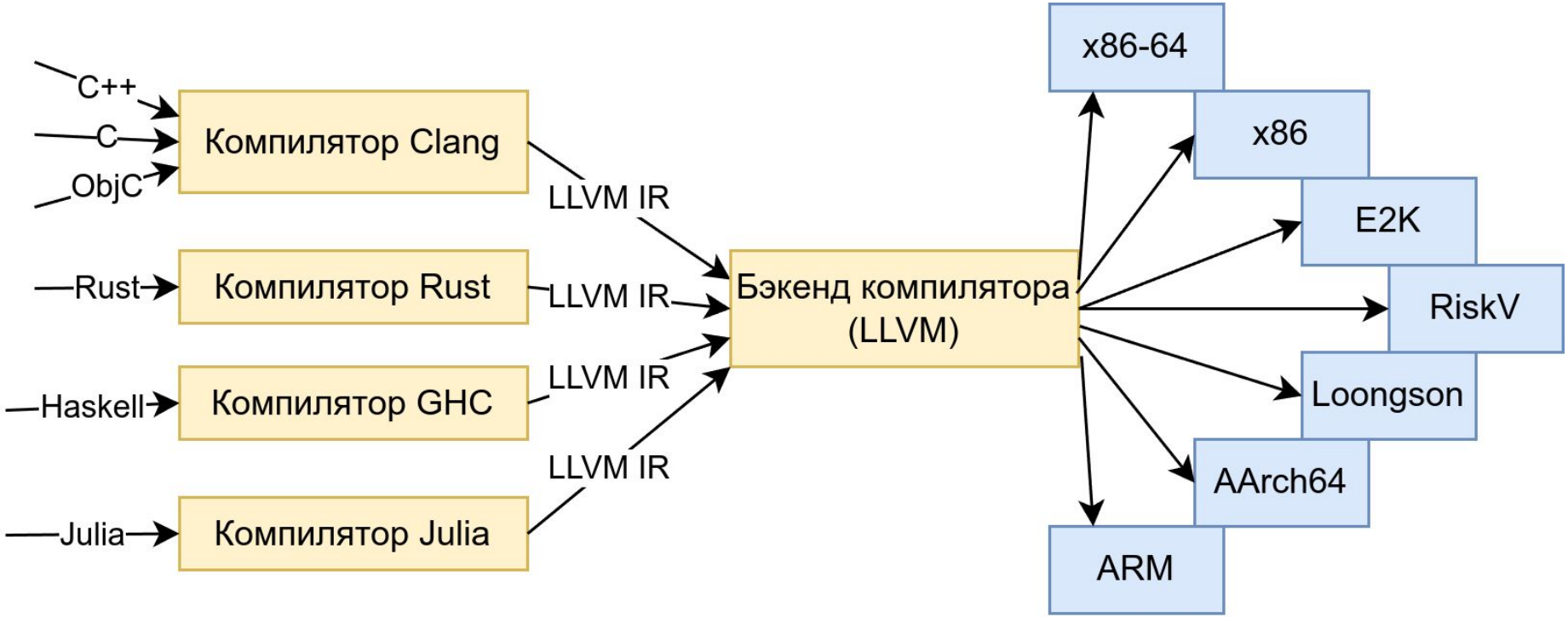
В Windows поддержка .NET встроена в загрузчик ОС

В Linux / OSX / Android / iOS иначе:

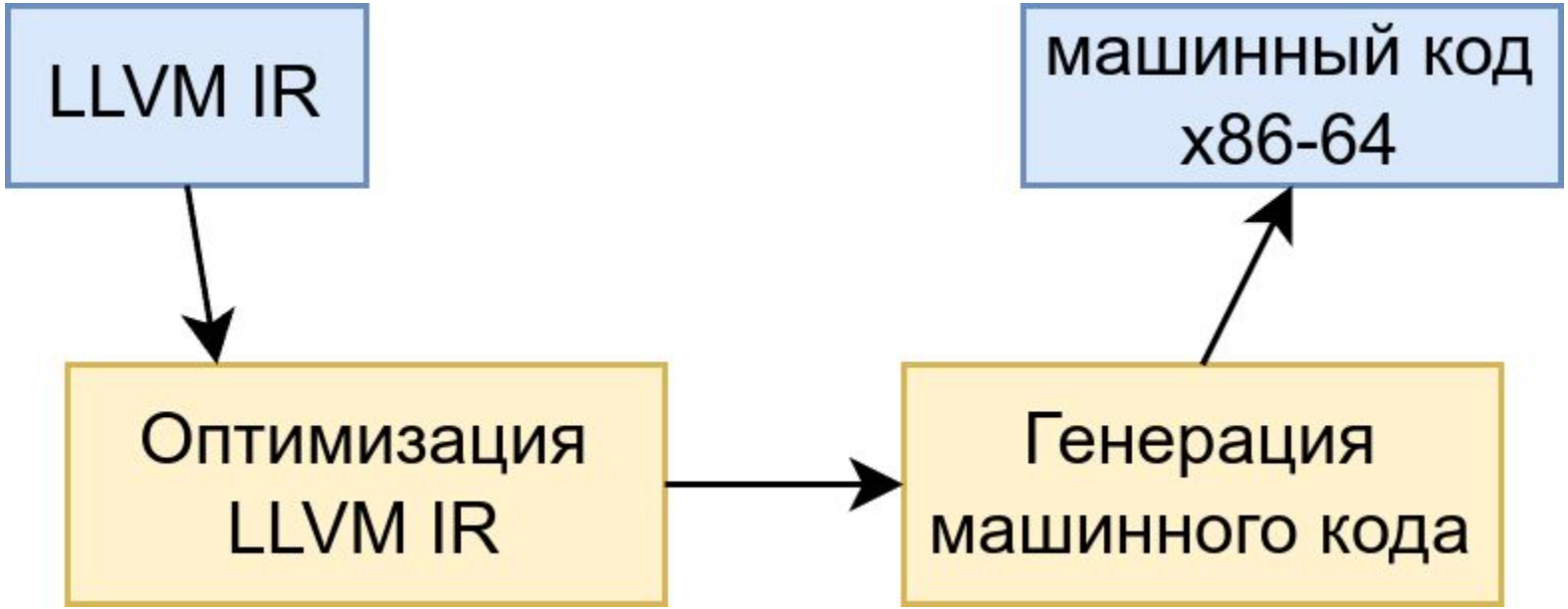
1. **Framework-dependent** program — загрузчик как отдельная программа (`dotnet exec`)
2. **Self-contained** program с виртуальной машиной .NET внутри

Архитектура LLVM

Роль LLVM в экосистеме компиляторов



Что делает LLVM в целом?



Проект нашего курса

Три дорожки на выбор команды

- **Зелёная** — интерпретатор с виртуальной машиной на C#
  - Переменные и выражения — оценка 3
  - Функции, ветвления, циклы — оценка 4
  - Массивы, классы и методы — оценка 5

Три дорожки на выбор команды

- **Зелёная** — интерпретатор с виртуальной машиной на C#
  - Переменные и выражения — оценка 3
  - Функции, ветвления, циклы — оценка 4
  - Массивы, классы и методы — оценка 5
- **Жёлтая** — компилятор для .NET (MSIL) на C#
  - Функции, ветвления, циклы — оценка 4
  - Массивы и структуры — оценка 5

Три дорожки на выбор команды

- **Зелёная** — интерпретатор с виртуальной машиной на C#
  - Переменные и выражения — оценка 3
  - Функции, ветвления, циклы — оценка 4
  - Массивы, классы и методы — оценка 5
- **Жёлтая** — компилятор для .NET (MSIL) на C#
  - Функции, ветвления, циклы — оценка 4
  - Массивы и структуры — оценка 5
- **Красная** — компилятор для LLVM на C++
  - Функции, ветвления и циклы — оценка 5

Команды по 3-4 человека

Роли:

1. Аналитик (1)
2. Разработчики (1-2)
3. Тестировщик (1)

Команды по 3-4 человека

Роли:

1. Аналитик (1)
2. Разработчики (1-2)
3. Тестировщик (1)

Важен успех команды:

- Каждая итерация — это закрытый этап проекта
- За итерацию вся команда получает баллы

Семь итераций

Каждая итерация — это целое число недель, от 1 до 4.

Семь итераций

Каждая итерация — это целое число недель, от 1 до 4.

- первая итерация — спецификация, примеры, архитектура, сроки
- далее 3 эпика по 2 итерации на каждый

Семь итераций

Каждая итерация — это целое число недель, от 1 до 4.

- первая итерация — спецификация, примеры, архитектура, сроки
- далее 3 эпика по 2 итерации на каждый

Пример деления на эпики

1. Ввод/вывод, выражения и переменные
2. Функции, ветвления и циклы
3. Массивы, классы и методы

Начисление баллов

Баллы за итерацию:

`score = base_score × deadline_factor × quality_factor`

- **`base_score`** — всегда 12 баллов
- **`deadline_factor`** — коэффициент за соблюдение сроков
  - Итерация длится до 4 недель
  - Не успели в 4 недели — коэффициент 0.3
- **`quality_factor`** — коэффициент за качество от 0.5 до 1.0
  - у каждой роли свои критерии качества

Критерии качества

Аналитик	1. полнота спецификации 2. преемственность спецификаций 3. понятный план итераций
Разработчик	1. соответствие спецификации 2. чистота кода 3. качество тестов
Тестировщик	1. соответствие спецификации 2. полнота приёмочных тестов 3. качество примеров программ

Конец