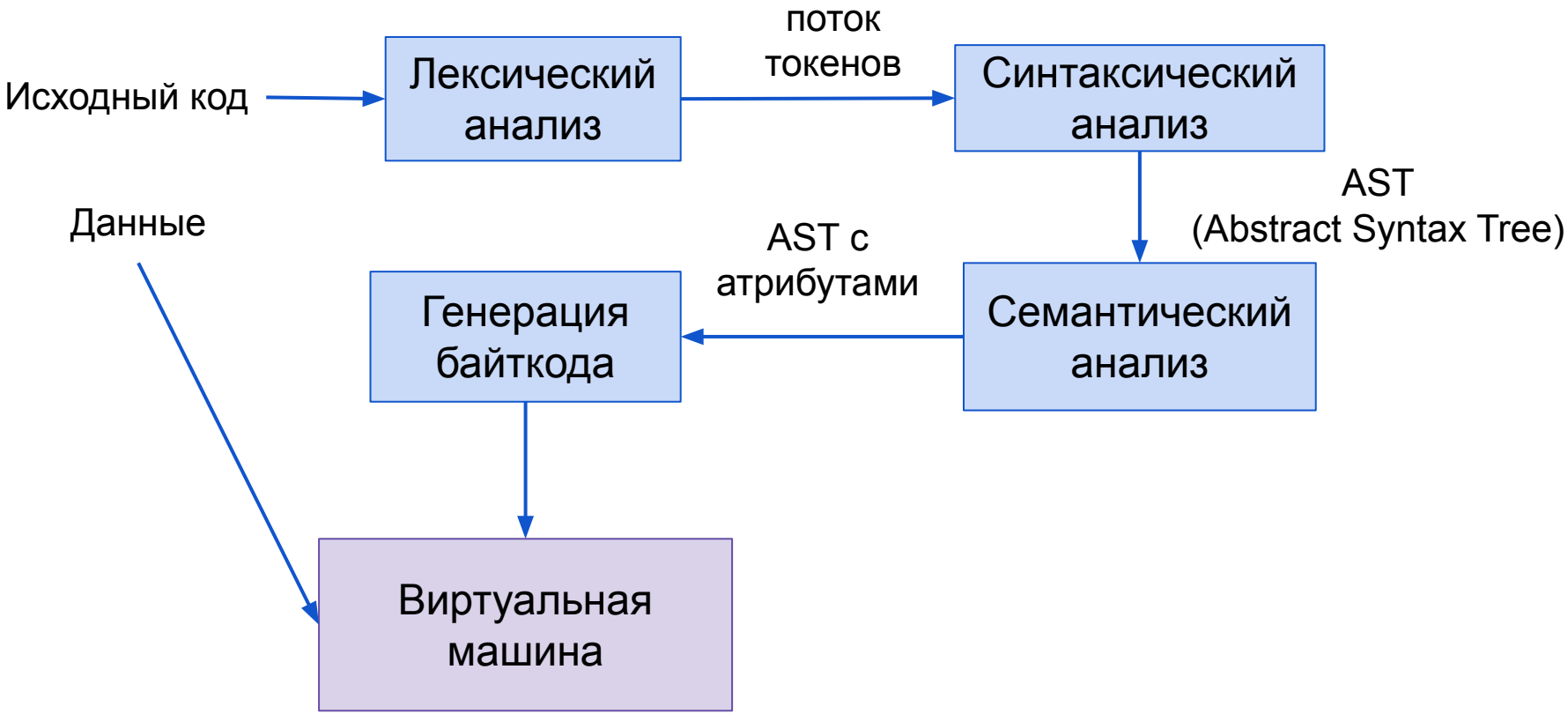
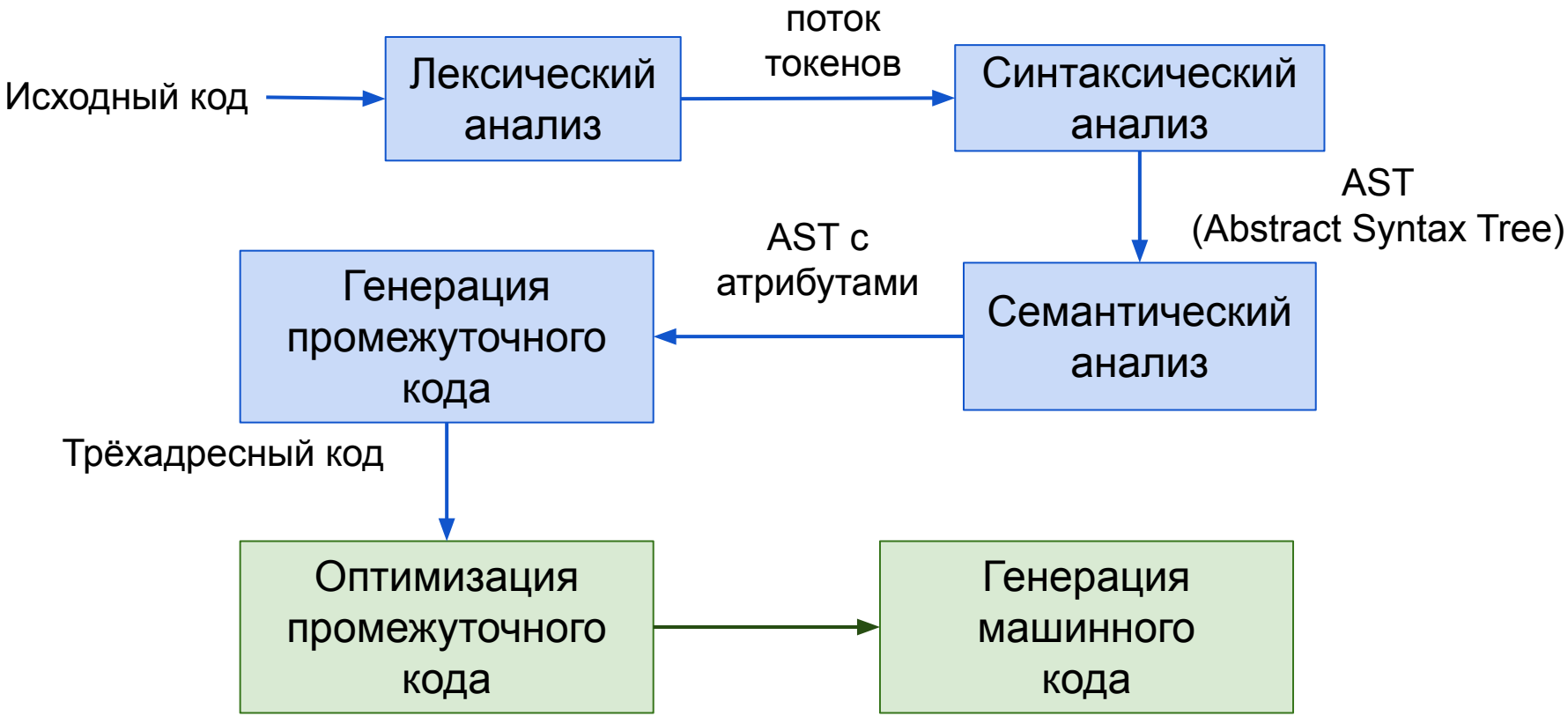
Абстрактные машины

Лекция №12

Интерпретация через виртуальную машину



Архитектура компилятора



Вопросы этой лекции

1. Почему интерпретатор и компилятор генерируют код для **машины** (физической или виртуальной)?

Вопросы этой лекции

1. Почему интерпретатор и компилятор генерируют код для **машины** (физической или виртуальной)?
2. Как программист понимает, правильно ли работает программа?

Вычислимость

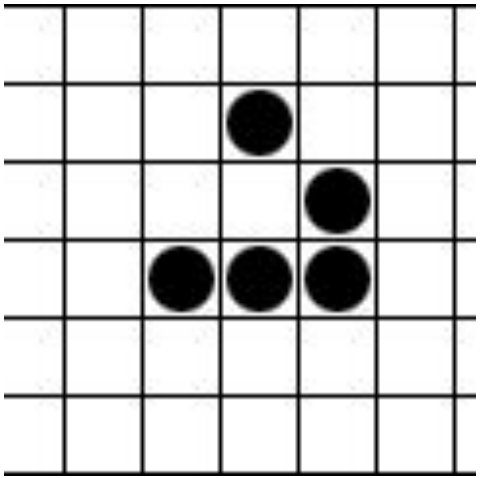
Вычислимая функция

- Если для функции существует алгоритм вычисления, то это **вычислимая функция**.
- **Алгоритм** — конечная последовательность шагов вычисления

Игра «Жизнь»

На каждой итерации есть два правила:

1. В пустой клетке, с которой соседствуют 3 живые клетки, зарождается жизнь
2. Живая клетка продолжает жить, если есть ровно 2 или 3 живых соседних клетки

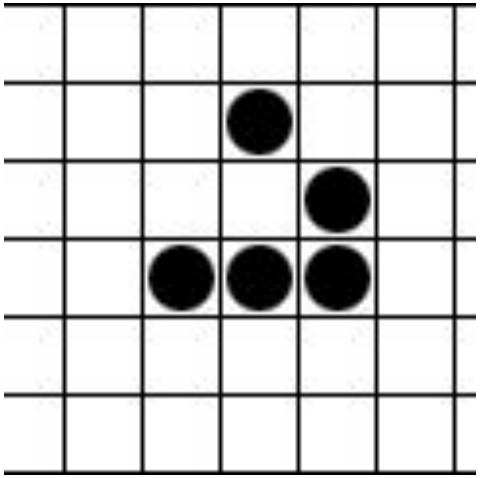


Игра «Жизнь»

На каждой итерации есть два правила:

1. В пустой клетке, с которой соседствуют 3 живые клетки, зарождается жизнь
2. Живая клетка продолжает жить, если есть ровно 2 или 3 живых соседних клетки

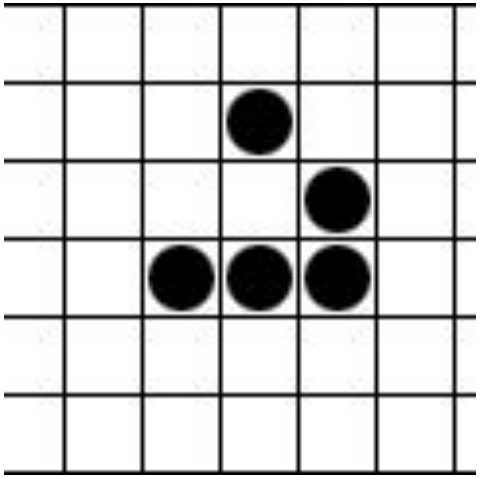
N-й шаг игры — это **вычислимая функци**я



Игра «Жизнь»

На каждой итерации есть два правила:

1. В пустой клетке, с которой соседствуют 3 живые клетки, зарождается жизнь
2. Живая клетка продолжает жить, если есть ровно 2 или 3 живых соседних клетки



N-й шаг игры — это **вычислимая функци**я

Нельзя написать алгоритм, который для любых двух паттернов A и B вычислит, может ли A превратиться в B — это **невычислимая функция**

Проблема останова

- Каждый шаг программы — **вычислимая функция**
- Ответ на вопрос «остановится ли когда-нибудь вычисление этой программы» — в общем случае **невычислимая функция**

Ограничение оптимизирующих трансляторов

Для любого алгоритма преобразования всегда найдутся две программы A и B, которые имеют одинаковую семантику, но не могут быть преобразованы этим алгоритмом.

Оптимизация программ — **невычислимая функция**.

Модели вычислимости

Любой алгоритм можно представить в нескольких эквивалентных формах:

- Нормальный Алгорифм Маркова
- Машина Тьюринга
- Частично рекурсивные функции
- Лямбда-исчисление
- *… и другие*

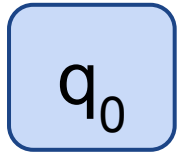
Машина Тьюринга
(Turing Machine)

Машина Тьюринга

Машина Тьюринга описывает один алгоритм

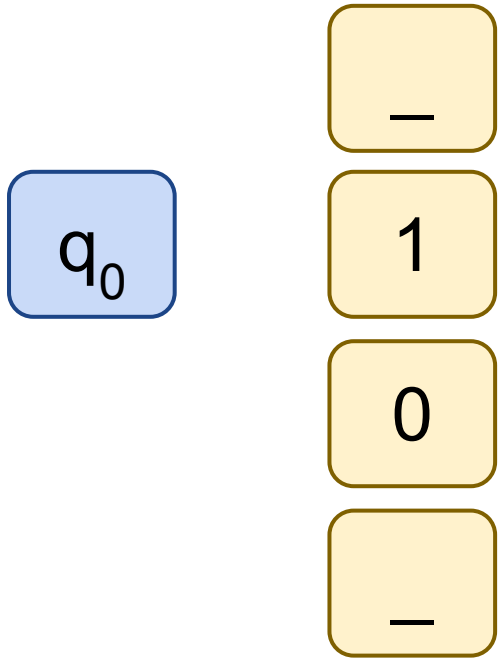
Элементы машины Тьюринга

1. Внутренняя память — конечный набор состояний Q
   - Есть начальное состояние
   - Есть состояние останова



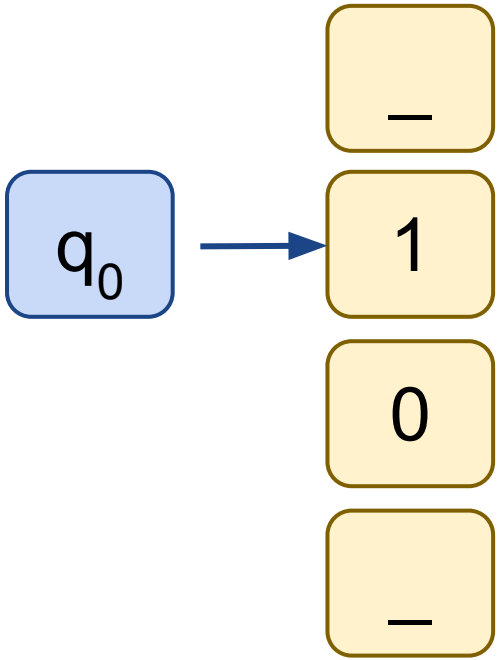
Элементы машины Тьюринга

1. Внутренняя память — конечный набор состояний Q
   - Есть начальное состояние
   - Есть состояние останова
2. Внешняя память — лента, в каждой ячейке которой хранится символ $П \in Σ$



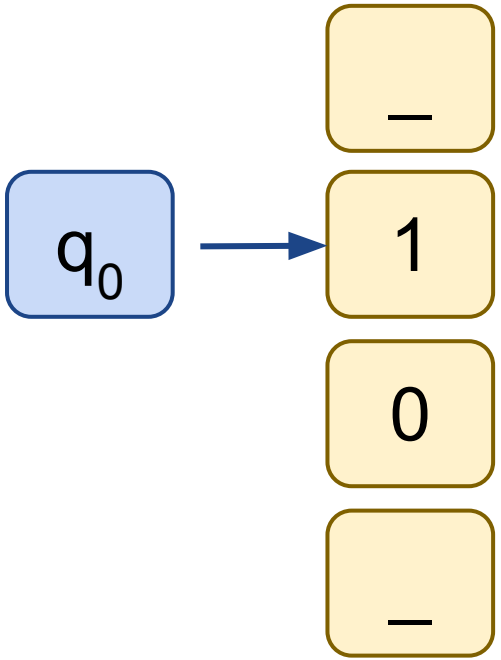
Элементы машины Тьюринга

1. Внутренняя память — конечный набор состояний Q
   - Есть начальное состояние
   - Есть состояние останова
2. Внешняя память — лента, в каждой ячейке которой хранится символ $П \in Σ$
3. Головка чтения-записи — указывает на ленту



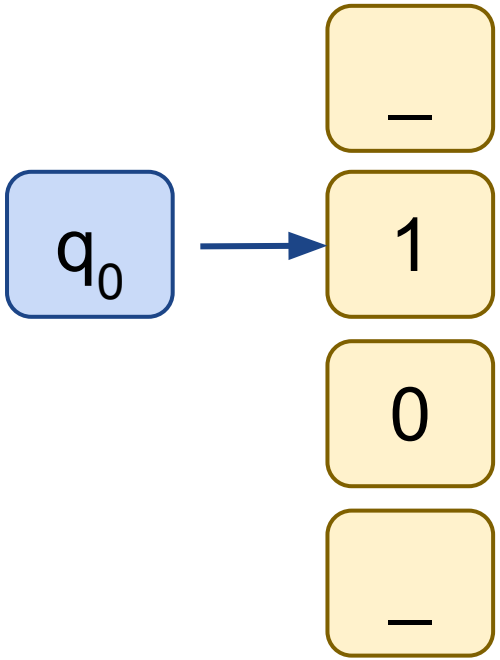
Элементы машины Тьюринга

1. Внутренняя память — конечный набор состояний Q
   - Есть начальное состояние
   - Есть состояние останова
2. Внешняя память — лента, в каждой ячейке которой хранится символ $П \in Σ$
3. Головка чтения-записи — указывает на ленту
4. Таблица перехода $Q \times П \to Q \times П \times \{\leftarrow, \rightarrow, \downarrow\}$



Пример Машины Тьюринга

Машина, заменяющая 1 на 0

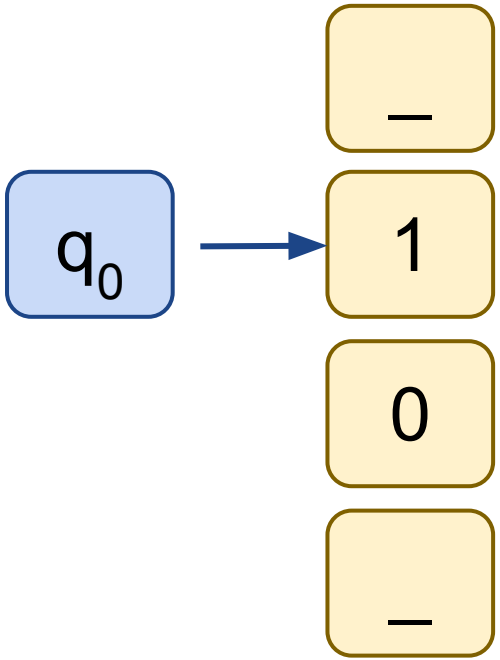


Пример Машины Тьюринга

Машина, заменяющая 1 на 0

Состояния:

- q0 — останов
- q1 — начальное состояние
- q2 — единственное промежуточное



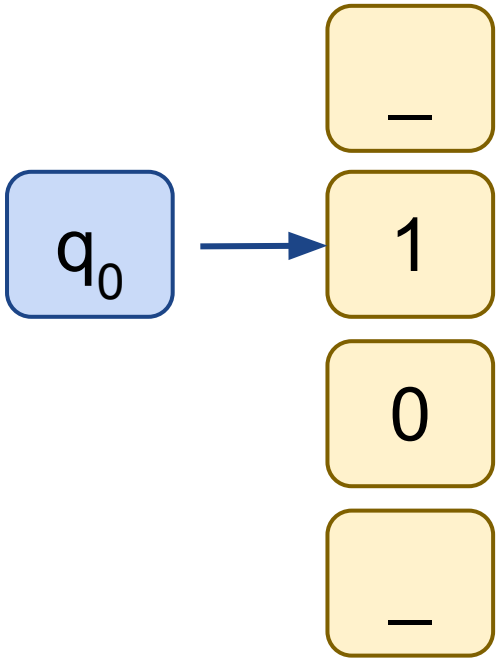
Пример Машины Тьюринга

Машина, заменяющая 1 на 0

Состояния:

- q0 — останов
- q1 — начальное состояние
- q2 — единственное промежуточное

	1	0	_
q1	,←,	,←,	,→, q2
q2	0,→,	0,→,	,↓,q0



Архитектура Фон-Неймана

Универсальная машина Тьюринга

Это Машина Тьюринга, способная выполнять любую другую Машину Тьюринга.

Универсальная машина Тьюринга

Это Машина Тьюринга, способная выполнять любую другую Машину Тьюринга.

Ключевая идея:

1. Закодировать на ленте таблицу переходов
2. Закодировать на ленте данные

Универсальная машина Тьюринга

Это Машина Тьюринга, способная выполнять любую другую Машину Тьюринга.

Ключевая идея:

1. Закодировать на ленте таблицу переходов
2. Закодировать на ленте данные

} Архитектура Фон-Неймана

Две абстрактные архитектуры процессоров

Архитектура Фон-Неймана

- Программа и данные хранятся в одной памяти
- Процессор читает инструкции из памяти

Гарвардская архитектура

- Программа и данные хранятся в разных областях памяти
- Программа не может читать/изменять собственные инструкции

Две абстрактные архитектуры процессоров

Архитектура Фон-Неймана

- Программа и данные хранятся в одной памяти
- Процессор читает инструкции из памяти

Гарвардская архитектура

- Программа и данные хранятся в разных областях памяти
- Программа не может читать/изменять собственные инструкции

1. Машина Тьюринга — какая из архитектур?

Две абстрактные архитектуры процессоров

Архитектура Фон-Неймана

- Программа и данные хранятся в одной памяти
- Процессор читает инструкции из памяти

Гарвардская архитектура

- Программа и данные хранятся в разных областях памяти
- Программа не может читать/изменять собственные инструкции

1. Машина Тьюринга — какая из архитектур?
2. Универсальная Машина Тьюринга?

Две абстрактные архитектуры процессоров

Архитектура Фон-Неймана

- Программа и данные хранятся в одной памяти
- Процессор читает инструкции из памяти

Гарвардская архитектура

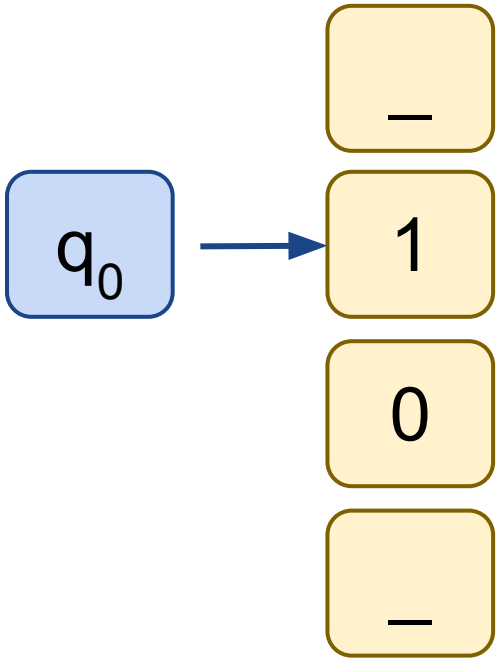
- Программа и данные хранятся в разных областях памяти
- Программа не может читать/изменять собственные инструкции

1. Машина Тьюринга — какая из архитектур?
2. Универсальная Машина Тьюринга?
3. Программа на языке высокого уровня?

Эмуляция

Элементы машины Тьюринга

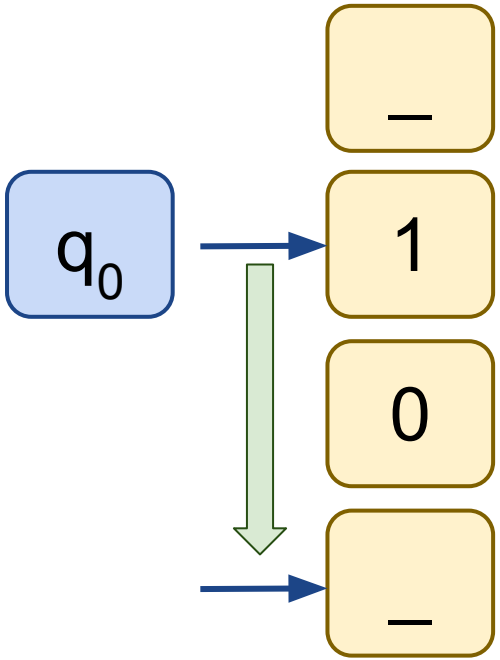
1. Внутренняя память — конечный набор состояний Q
2. Внешняя память — лента, в каждой ячейке которой хранится символ $П \in Σ$
3. Головка чтения-записи — указывает на ленту
4. Таблица перехода $Q \times П \to Q \times П \times \{\leftarrow, \rightarrow, \downarrow\}$



Элементы машины Тьюринга

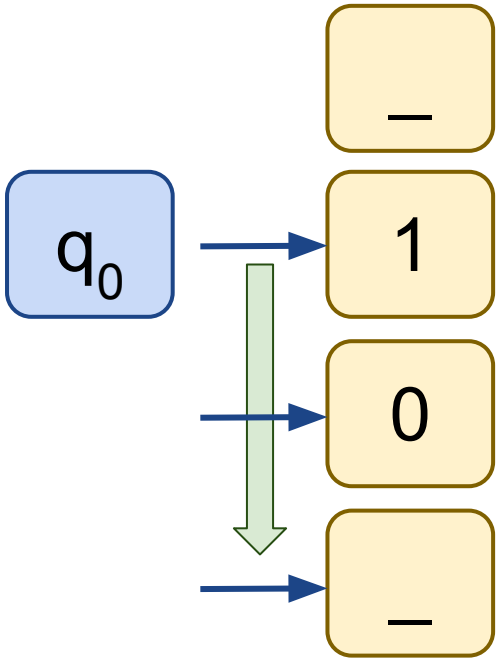
1. Внутренняя память — конечный набор состояний Q
2. Внешняя память — лента, в каждой ячейке которой хранится символ $П \in Σ$
3. Головка чтения-записи — указывает на ленту
4. Таблица перехода $Q \times П \to Q \times П \times \{\leftarrow, \rightarrow, \downarrow\}$

Вопрос: как передвинуть головку на N позиций?



Элементы машины Тьюринга

1. Внутренняя память — конечный набор состояний Q
2. Внешняя память — лента, в каждой ячейке которой хранится символ $П \in Σ$
3. Головка чтения-записи — указывает на ленту
4. Таблица перехода $Q \times П \to Q \times П \times \{\leftarrow, \rightarrow, \downarrow\}$



Вопрос: как передвинуть головку на N позиций?

Ответ: добавить N-1 промежуточных состояний

RAM и RASP

Назначение RAM-машины (Random Access Machine)

Абстрактные машины применяются в доказательствах

Назначение RAM-машины (Random Access Machine)

Абстрактные машины применяются в доказательствах:

1. Машины Тьюринга — доказательство вычислимости
    - Эффективность не имеет значения

Назначение RAM-машины (Random Access Machine)

Абстрактные машины применяются в доказательствах:

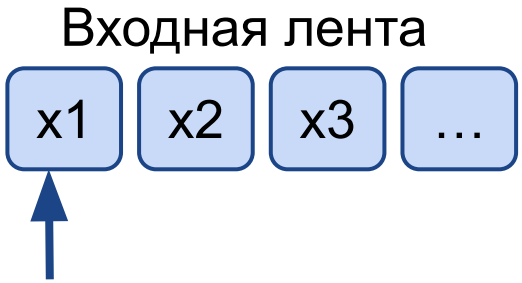
1. Машины Тьюринга — доказательство вычислимости
    - Эффективность не имеет значения
2. RAM-машины — доказательство алгоритмической сложности
    - RAM-машины близки к реальным процессорам
    - Алгоритмическая сложность для RAM-машины и для реального CPU обычно одинакова

Элементы RAM-машины (Random Access Machine)

1. Внутренняя память — конечный набор регистров с целыми числами
   - Первый регистр — аккумулятор
   - Можно обратиться к любому регистру за 1 шаг

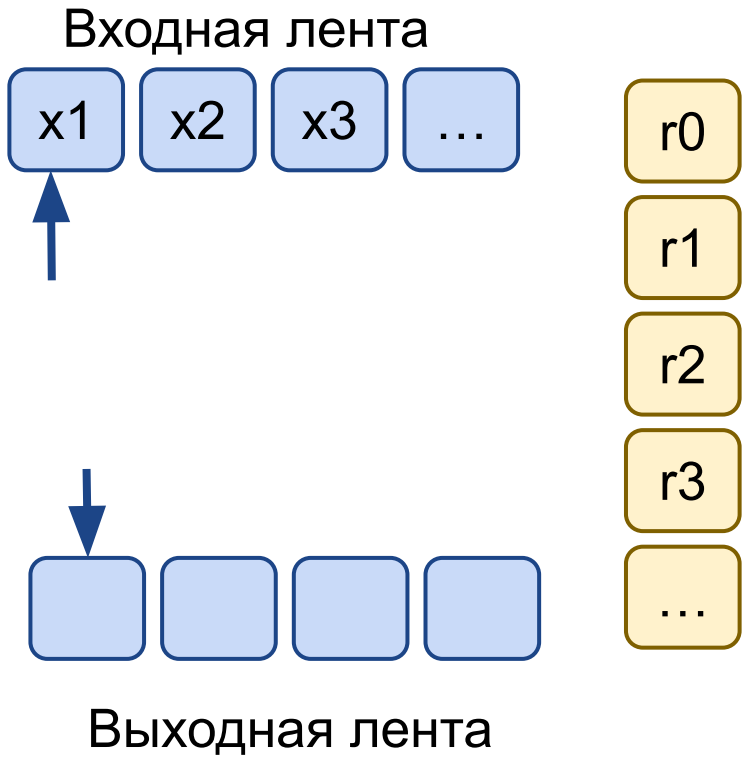
Элементы RAM-машины (Random Access Machine)

1. Внутренняя память — конечный набор регистров с целыми числами
   - Первый регистр — аккумулятор
   - Можно обратиться к любому регистру за 1 шаг
2. Входная лента целых чисел



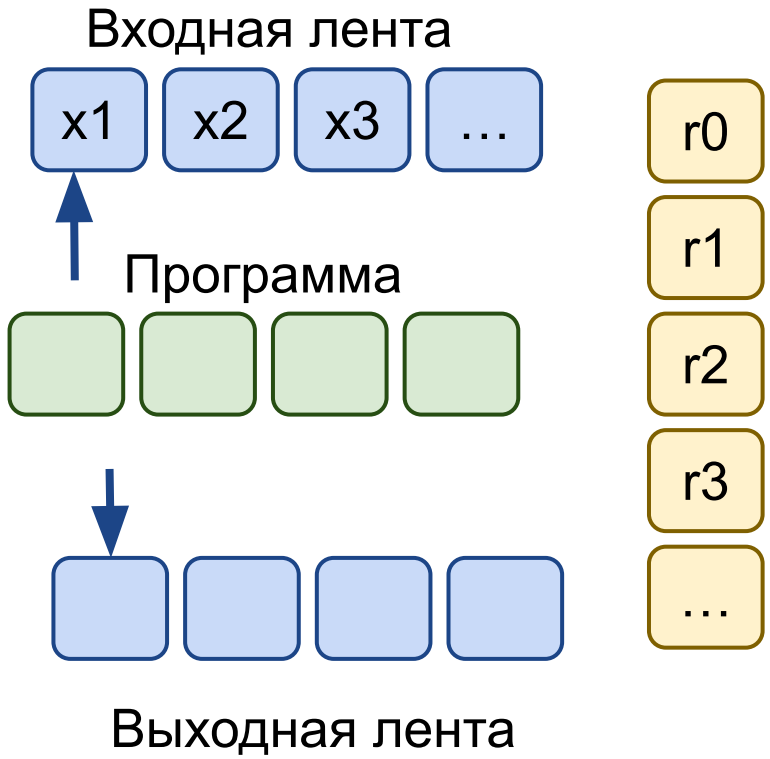
Элементы RAM-машины (Random Access Machine)

1. Внутренняя память — конечный набор регистров с целыми числами
   - Первый регистр — аккумулятор
   - Можно обратиться к любому регистру за 1 шаг
2. Входная лента целых чисел
3. Выходная лента целых чисел



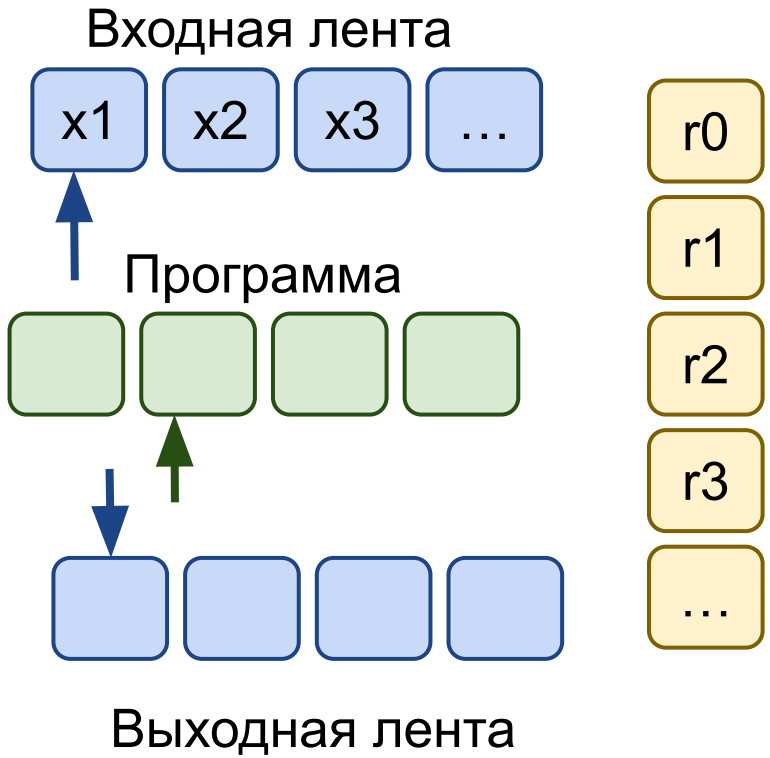
Элементы RAM-машины (Random Access Machine)

1. Внутренняя память — конечный набор регистров с целыми числами
   - Первый регистр — аккумулятор
   - Можно обратиться к любому регистру за 1 шаг
2. Входная лента целых чисел
3. Выходная лента целых чисел
4. Программа — список инструкций



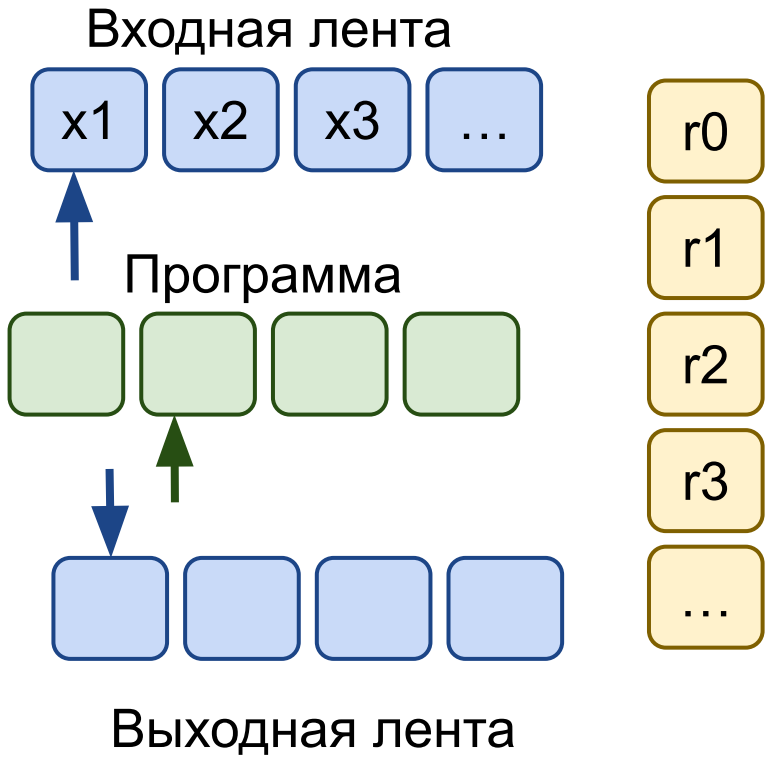
Элементы RAM-машины (Random Access Machine)

1. Внутренняя память — конечный набор регистров с целыми числами
   - Первый регистр — аккумулятор
   - Можно обратиться к любому регистру за 1 шаг
2. Входная лента целых чисел
3. Выходная лента целых чисел
4. Программа — список инструкций
5. Указатель на текущую инструкцию



Элементы RAM-машины (Random Access Machine)

1. Внутренняя память — конечный набор регистров с целыми числами
   - Первый регистр — аккумулятор
   - Можно обратиться к любому регистру за 1 шаг
2. Входная лента целых чисел
3. Выходная лента целых чисел
4. Программа — список инструкций
5. Указатель на текущую инструкцию
6. Набор инструкций



Набор инструкций RAM-машины (пример)

1. **LOAD** const — запись числа в аккумулятор
2. **STORE** reg — запись из аккумулятора в регистр

Набор инструкций RAM-машины (пример)

1. **LOAD** const — запись числа в аккумулятор
2. **STORE** reg — запись из аккумулятора в регистр
3. **ADD**, **SUM**, **MULT**, **DIV** — арифметические операции

Набор инструкций RAM-машины (пример)

1. **LOAD** const — запись числа в аккумулятор
2. **STORE** reg — запись из аккумулятора в регистр
3. **ADD**, **SUM**, **MULT**, **DIV** — арифметические операции
4. **READ** reg — чтение из входной ленты в регистр
5. **WRITE** reg — запись из регистра в выходную ленту

Набор инструкций RAM-машины (пример)

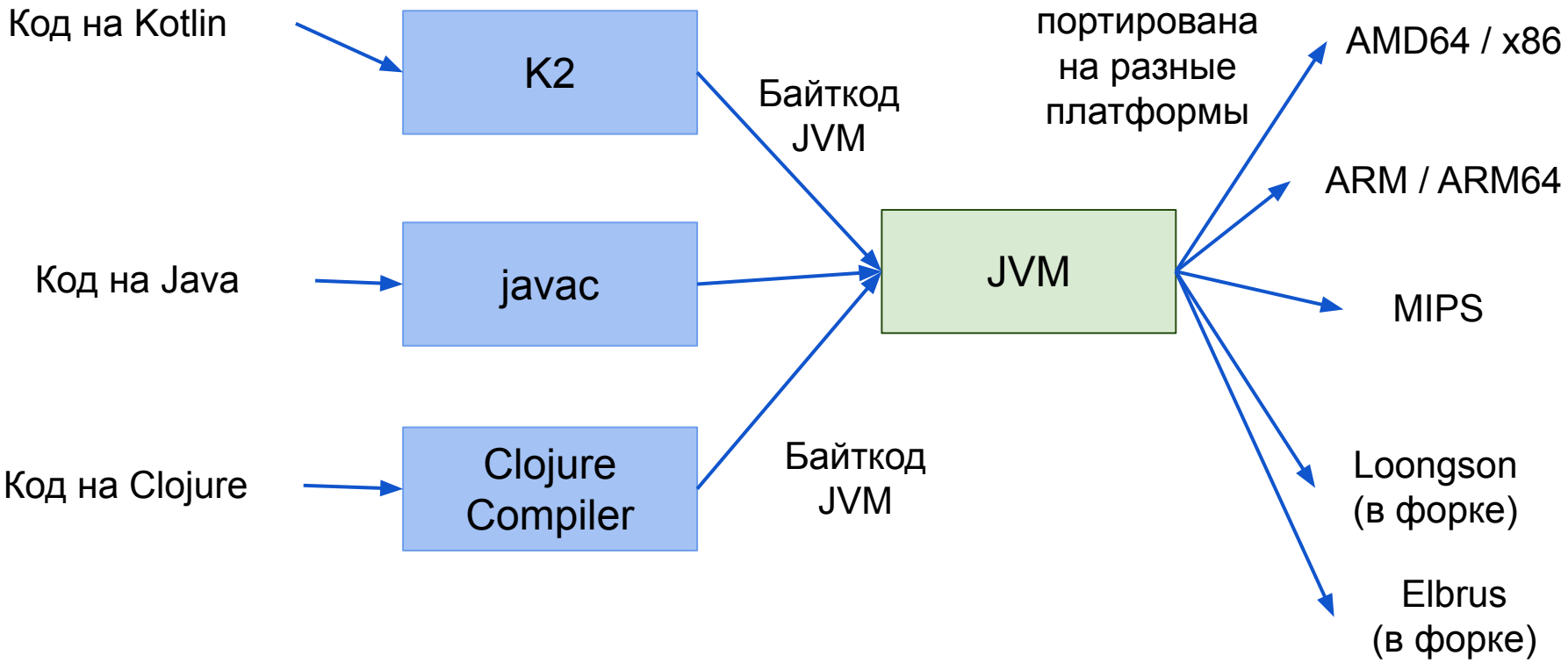
1. **LOAD** const — запись числа в аккумулятор
2. **STORE** reg — запись из аккумулятора в регистр
3. **ADD**, **SUM**, **MULT**, **DIV** — арифметические операции
4. **READ** reg — чтение из входной ленты в регистр
5. **WRITE** reg — запись из регистра в выходную ленту
6. **JUMP** — безусловный переход
7. **JGTZ**, **JZERO** — условные переходы

Набор инструкций RAM-машины (пример)

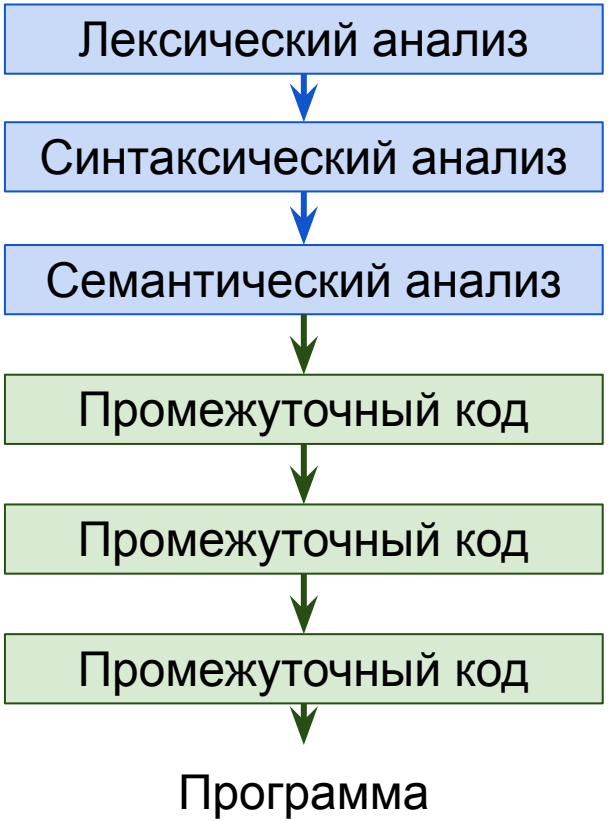
1. **LOAD** const — запись числа в аккумулятор
2. **STORE** reg — запись из аккумулятора в регистр
3. **ADD**, **SUM**, **MULT**, **DIV** — арифметические операции
4. **READ** reg — чтение из входной ленты в регистр
5. **WRITE** reg — запись из регистра в выходную ленту
6. **JUMP** — безусловный переход
7. **JGTZ**, **JZERO** — условные переходы
8. **HALT** — останов

Абстрактная машина как модель мышления

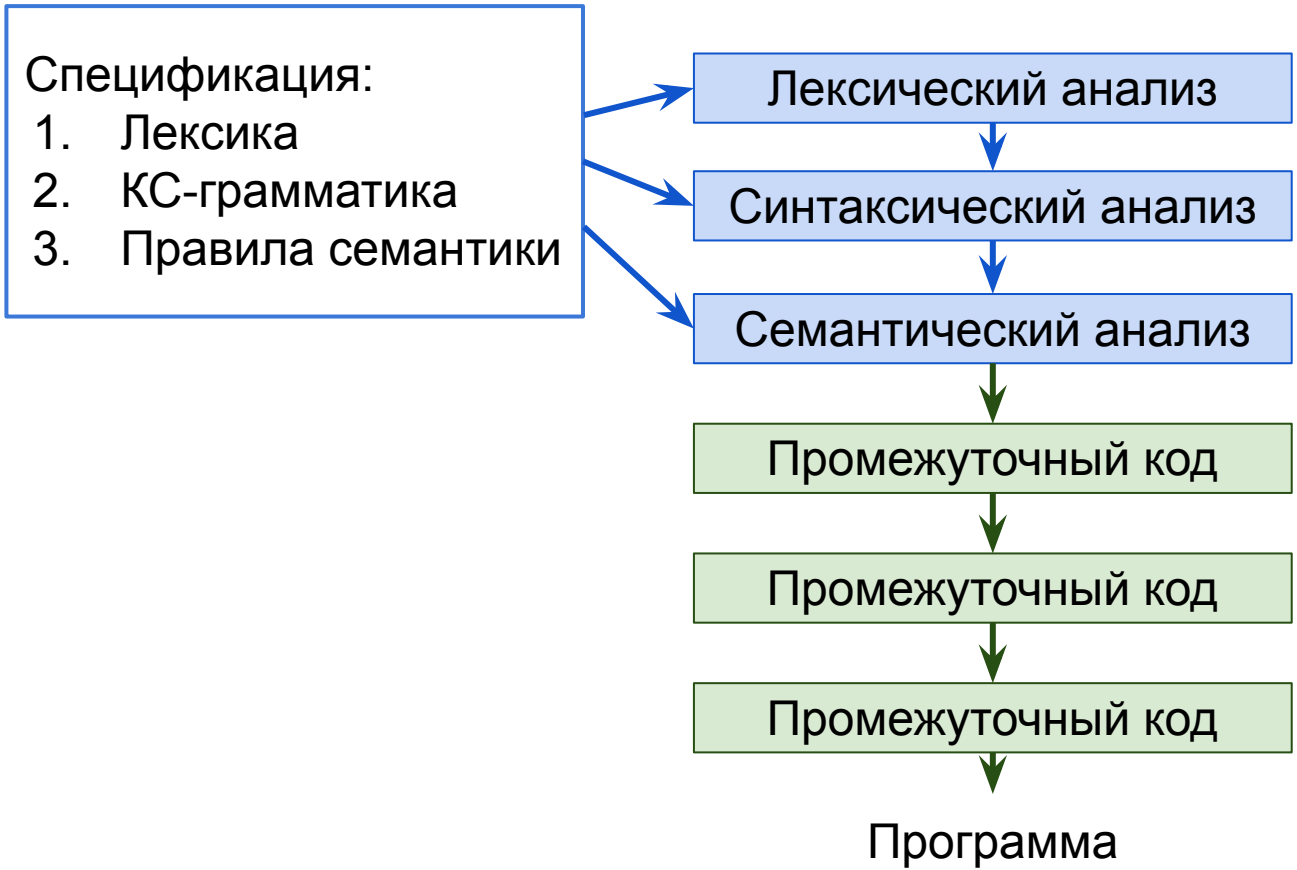
Java Virtual Machine



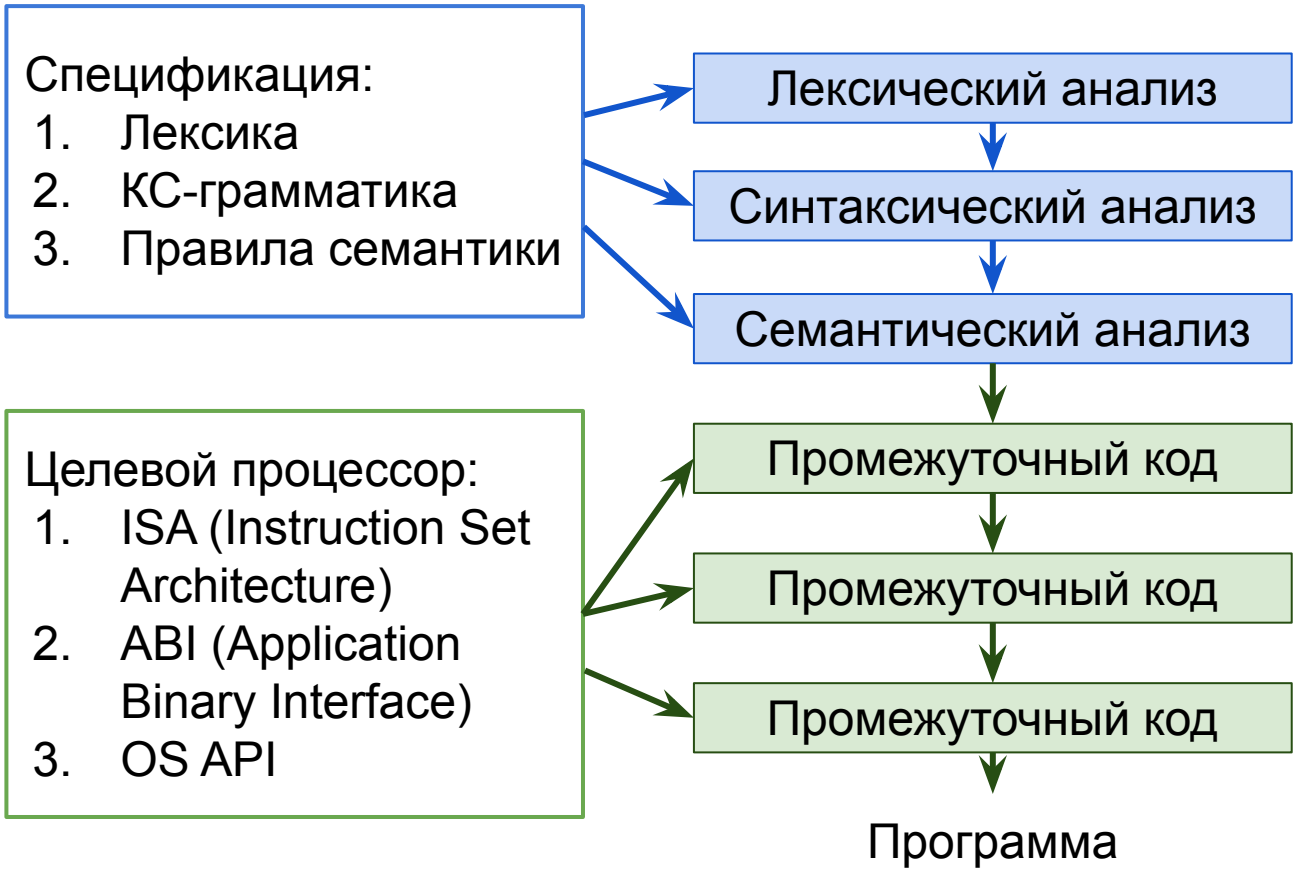
Как работает язык программирования



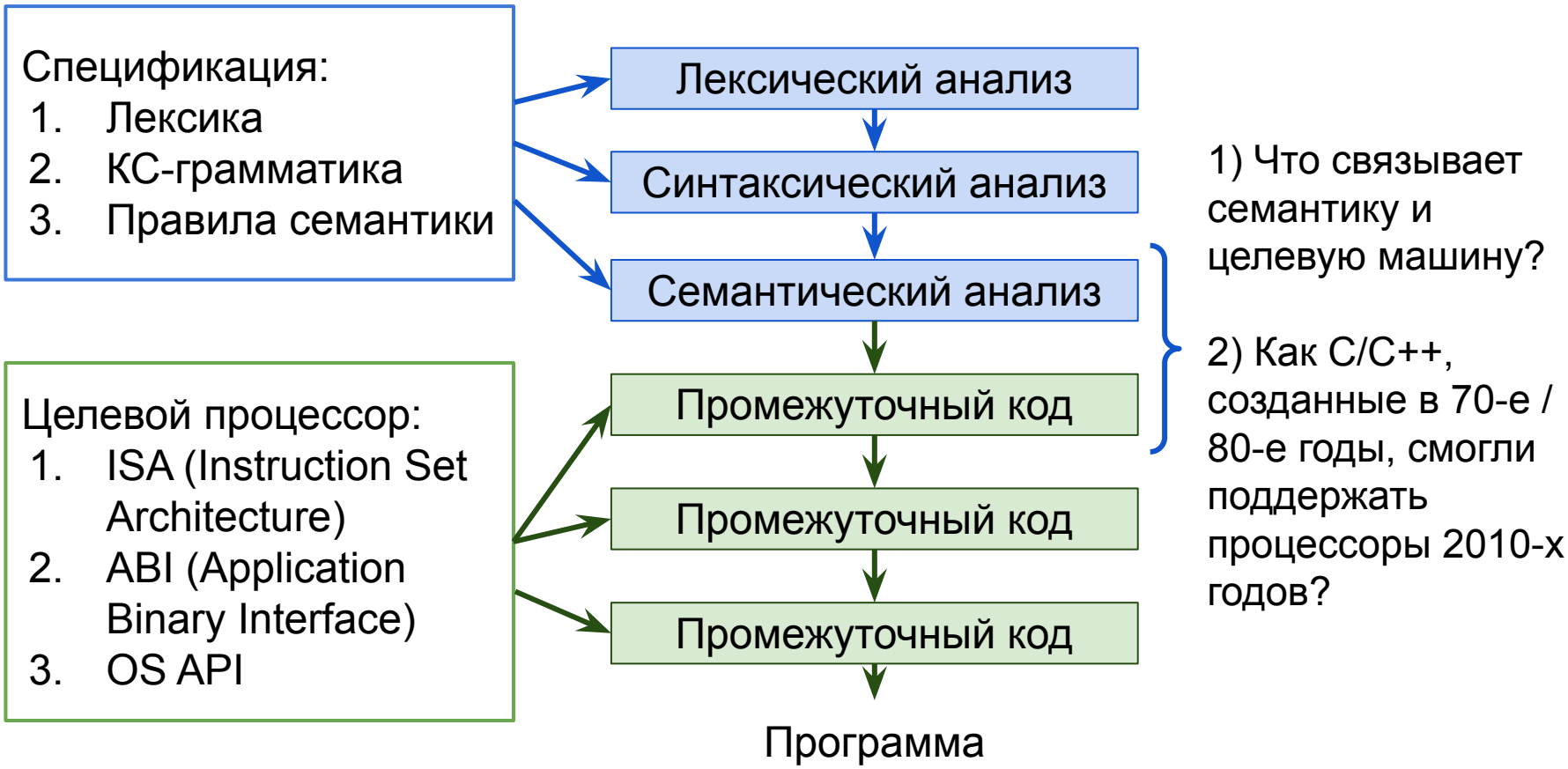
Как работает язык программирования



Как работает язык программирования



Как работает язык программирования



Мышление через абстрактную машину

Хороший программист при чтении кода может:

Мышление через абстрактную машину

Хороший программист при чтении кода может:

1. Проверять соответствие правилам языка
    - … используя семантику

Мышление через абстрактную машину

Хороший программист при чтении кода может:

1. Проверять соответствие правилам языка
    - … используя семантику
2. Проверять корректность реализации
    - … используя абстрактную машину в своей голове

Мышление через абстрактную машину

Хороший программист при чтении кода может:

1. Проверять соответствие правилам языка
    - … используя семантику
2. Проверять корректность реализации
    - … используя абстрактную машину в своей голове
3. Оценивать алгоритмическую сложность
    - … используя абстрактную машину в своей голове

Мышление через абстрактную машину

Хороший программист при чтении кода может:

1. Проверять соответствие правилам языка
   - … используя семантику
2. Проверять корректность реализации
   - … используя абстрактную машину в своей голове
3. Оценивать алгоритмическую сложность
   - … используя абстрактную машину в своей голове
4. Оценивать реальную эффективность
   - … используя знания об особенностях реальных процессоров — x86, AMD64, ARM AArch64, E2K и т.д.

Подытожим

Вопросы этой лекции

1. Почему интерпретатор и компилятор генерируют код для **машины** (физической или виртуальной)?
2. Как программист понимает, правильно ли работает программа?

Вопросы этой лекции

1. Почему интерпретатор и компилятор генерируют код для **машины** (физической или виртуальной)?
2. Как программист понимает, правильно ли работает программа?

1. Модель целевой машины определяет как вычислимость, так эффективность вычислений
2. Программист мыслит абстрактными машинами.

Ссылки

1. Abstract Machine Models Also: what Rust got particularly right — о мышлении программистов через абстрактные машины
2. RAM-машина — о том, как собрать RAM-машину из базовых элементов (сумматоры, память и т.д.)
3. Машина Тьюринга