Зачем нужен ТАиФЯ?

Лекция №1

Зачем нужна теория автоматов?

Классификация программного обеспечения

Классификация программного обеспечения

1. **Серверное ПО** — бэкенд веб-сервисов и мобильных приложений

Классификация программного обеспечения

1. **Серверное ПО** — бэкенд веб-сервисов и мобильных приложений
2. **Приложения** — фронтенд, мобильные и десктопные приложения, игры

Классификация программного обеспечения

1. **Серверное ПО** — бэкенд веб-сервисов и мобильных приложений
2. **Приложения** — фронтенд, мобильные и десктопные приложения, игры
3. **Трансляторы** — компиляторы и инструменты

Классификация программного обеспечения

1. **Серверное ПО** — бэкенд веб-сервисов и мобильных приложений
2. **Приложения** — фронтенд, мобильные и десктопные приложения, игры
3. **Трансляторы** — компиляторы и инструменты
4. **Управляющее ПО** — в технике, в промышленном оборудовании, в детских игрушках

Что важно в разных типах ПО

1. **Серверное ПО** — ???
2. **Приложения** — ???
3. **Трансляторы** — ???
4. **Управляющее ПО** — ???

Что важно в разных типах ПО

1. **Серверное ПО** — целостность данных, масштабируемость, SLA, latency
2. **Приложения** — ???
3. **Трансляторы** — ???
4. **Управляющее ПО** — ???

Что важно в разных типах ПО

1. **Серверное ПО** — целостность данных, масштабируемость, SLA, latency
2. **Приложения** — UX, отзывчивость, экономия ресурсов компьютера
3. **Трансляторы** — ???
4. **Управляющее ПО** — ???

Что важно в разных типах ПО

1. **Серверное ПО** — целостность данных, масштабируемость, SLA, latency
2. **Приложения** — UX, отзывчивость, экономия ресурсов компьютера
3. **Трансляторы** — корректность, диагностика ошибок, общее время обработки данных
4. **Управляющее ПО** — ???

Что важно в разных типах ПО

1. **Серверное ПО** — целостность данных, масштабируемость, SLA, latency
2. **Приложения** — UX, отзывчивость, экономия ресурсов компьютера
3. **Трансляторы** — корректность, диагностика ошибок, общее время обработки данных
4. **Управляющее ПО** — полная автономность, максимальная устойчивость к ошибкам и корректность

Теоретическая база

1. **Серверное ПО** — ???
2. **Приложения** — ???
3. **Трансляторы** — ???
4. **Управляющее ПО** — ???

Теоретическая база

1. **Серверное ПО** — базы данных, теория распределённых систем, ООП, DDD и Чистая Архитектура
2. **Приложения** — ???
3. **Трансляторы** — ???
4. **Управляющее ПО** — ???

Теоретическая база

1. **Серверное ПО** — базы данных, теория распределённых систем, ООП, DDD и Чистая Архитектура
2. **Приложения** — UX, ООП (местами ФП)
3. **Трансляторы** — ???
4. **Управляющее ПО** — ???

Теоретическая база

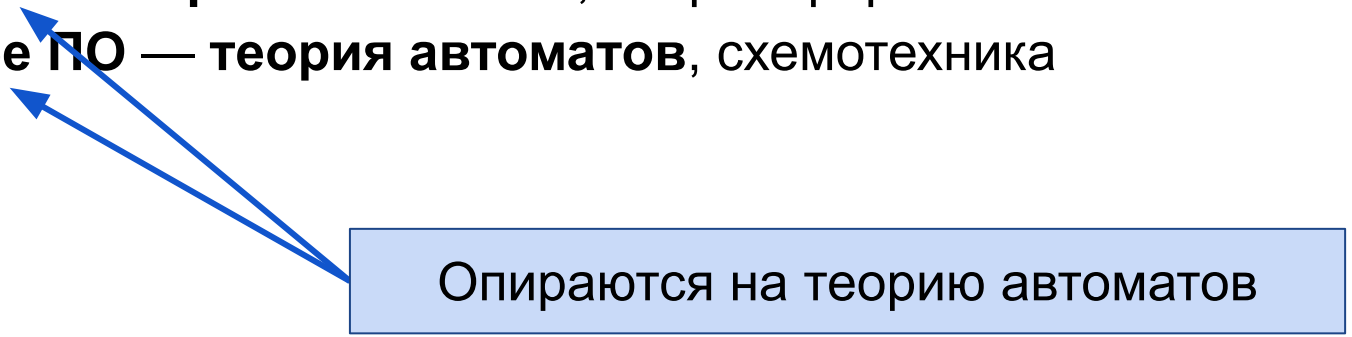
1. **Серверное ПО** — базы данных, теория распределённых систем, ООП, DDD и Чистая Архитектура
2. **Приложения** — UX, ООП (местами ФП)
3. **Трансляторы** — **теория автоматов**, теория формальных языков
4. **Управляющее ПО** — ???

Теоретическая база

1. **Серверное ПО** — базы данных, теория распределённых систем, ООП, DDD и Чистая Архитектура
2. **Приложения** — UX, ООП (местами ФП)
3. **Трансляторы — теория автоматов**, теория формальных языков
4. **Управляющее ПО — теория автоматов**, схемотехника

Теоретическая база

1. **Серверное ПО** — базы данных, теория распределённых систем, ООП, DDD и Чистая Архитектура
2. **Приложения** — UX, ООП (местами ФП)
3. **Трансляторы** — **теория автоматов**, теория формальных языков
4. **Управляющее ПО** — **теория автоматов**, схемотехника



Зачем нужны формальные языки?

Преимущество формальных языков

100% корректность при любой сложности

1. IDE, компиляторы, интерпретаторы, линтеры

Преимущество формальных языков

100% корректность при любой сложности

1. IDE, компиляторы, интерпретаторы, линтеры
2. Автоматное программирование

Преимущество формальных языков

100% корректность при любой сложности

1. IDE, компиляторы, интерпретаторы, линтеры
2. Автоматное программирование
3. Регулярные выражения

Преимущество формальных языков

100% корректность при любой сложности

1. IDE, компиляторы, интерпретаторы, линтеры
2. Автоматное программирование
3. Регулярные выражения
4. Системы автоматизированного доказательства
    - Теорема о 5 красках — доказана человеком в 1890 году
    - Теорема о 4 красках — доказана компьютером в 1976 году

ИИ-модели (Generative Pre-Trained Transformer)

Преимущества:

1. Понимают смысл языков
2. Запоминают и комбинируют знания

ИИ-модели (Generative Pre-Trained Transformer)

Преимущества:

1. Понимают смысл языков
2. Запоминают и комбинируют знания

Недостатки:

1. Нет самокритики
2. Галлюцинируют
3. Энтропия

ИИ-модели (Generative Pre-Trained Transformer)

Преимущества:

1. Понимают смысл языков
2. Запоминают и комбинируют знания

Недостатки:

1. Нет самокритики
2. Галлюцинируют
3. Энтропия

Главная проблема: верификация

ИИ-модели (Generative Pre-Trained Transformer)

Преимущества:

1. Понимают смысл языков
2. Запоминают и комбинируют знания

Сильные области:

1. Перевод текстов
2. Распознавание речи
3. Программирование
4. Математика

Недостатки:

1. Нет самокритики
2. Галлюцинируют
3. Энтропия

Главная проблема: верификация

ИИ-модели (Generative Pre-Trained Transformer)

Преимущества:

1. Понимают смысл языков
2. Запоминают и комбинируют знания

Недостатки:

1. Нет самокритики
2. Галлюцинируют
3. Энтропия

Сильные области:

1. Перевод текстов
2. Распознавание речи
3. Программирование
4. Математика

Верификация благодаря формальным языкам

Главная проблема: верификация

Цель курса

Цель курса — проект

ТАиФЯ (5-й семестр, зачёт)

Цель — интерпретатор

ТЯП (6-й семестр, экзамен)

Цель — компилятор

Компилятор — сложный проект

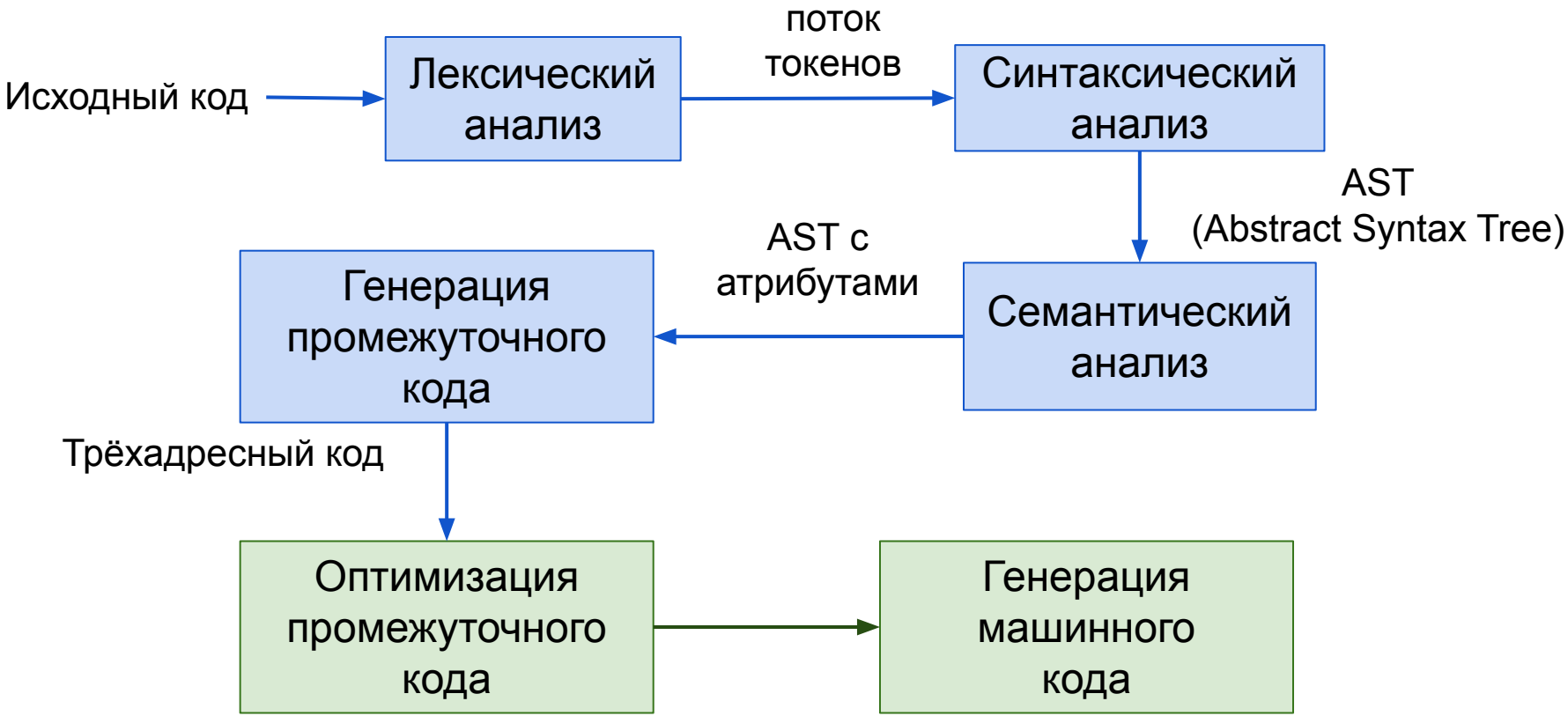
1. Первый компилятор Fortran (1957) — 19 человеко-лет
2. Компилятор Clang (2007) — тысячи человеко-лет

Компилятор — сложный проект

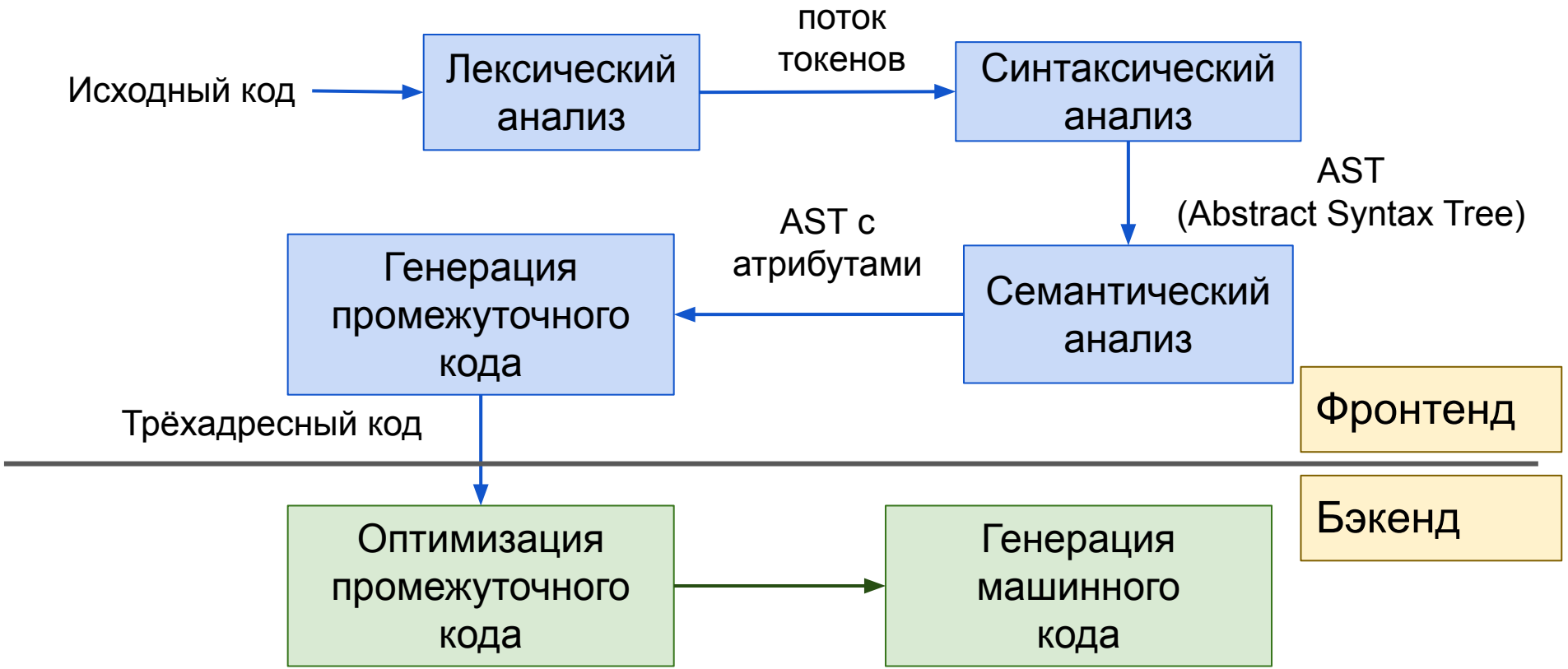
1. Первый компилятор Fortran (1957) — 19 человеко-лет
2. Компилятор Clang (2007) — тысячи человеко-лет

Теория, архитектура и тесты снижают сложность

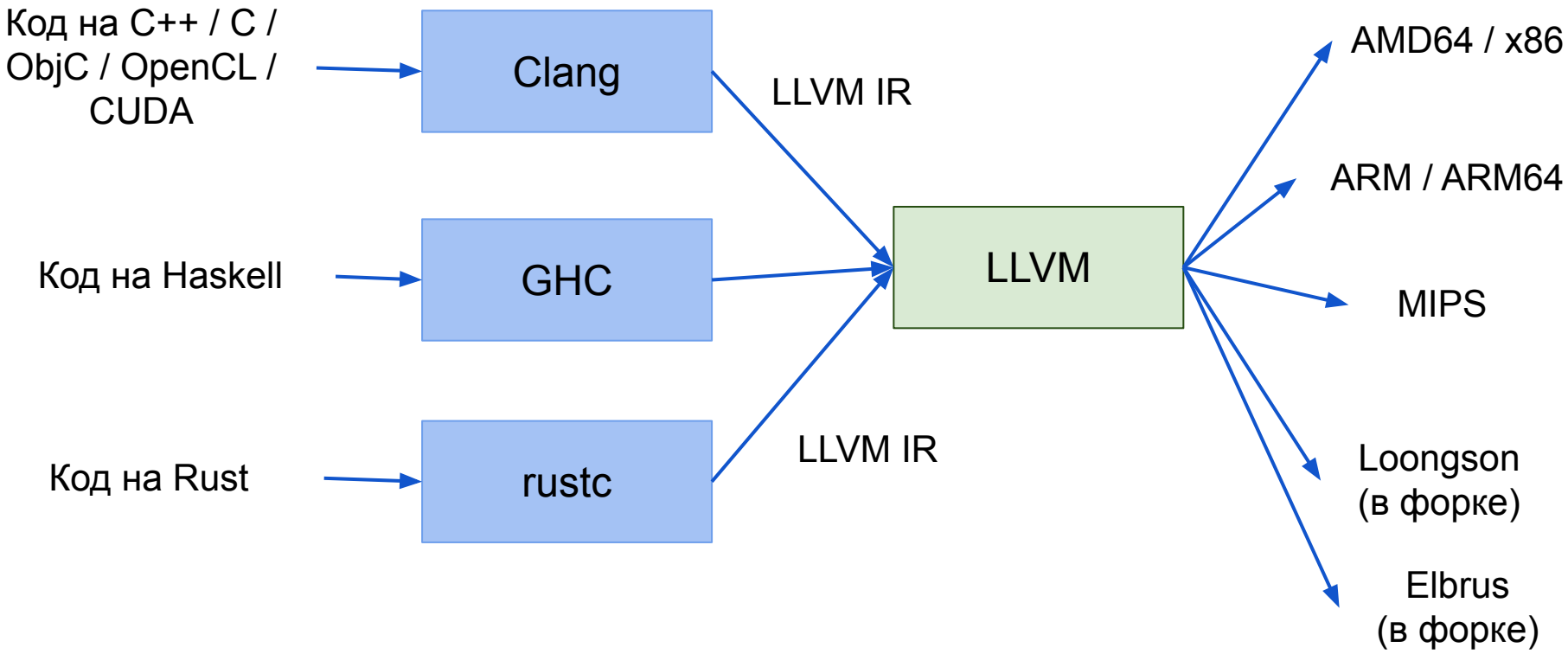
Архитектура компилятора



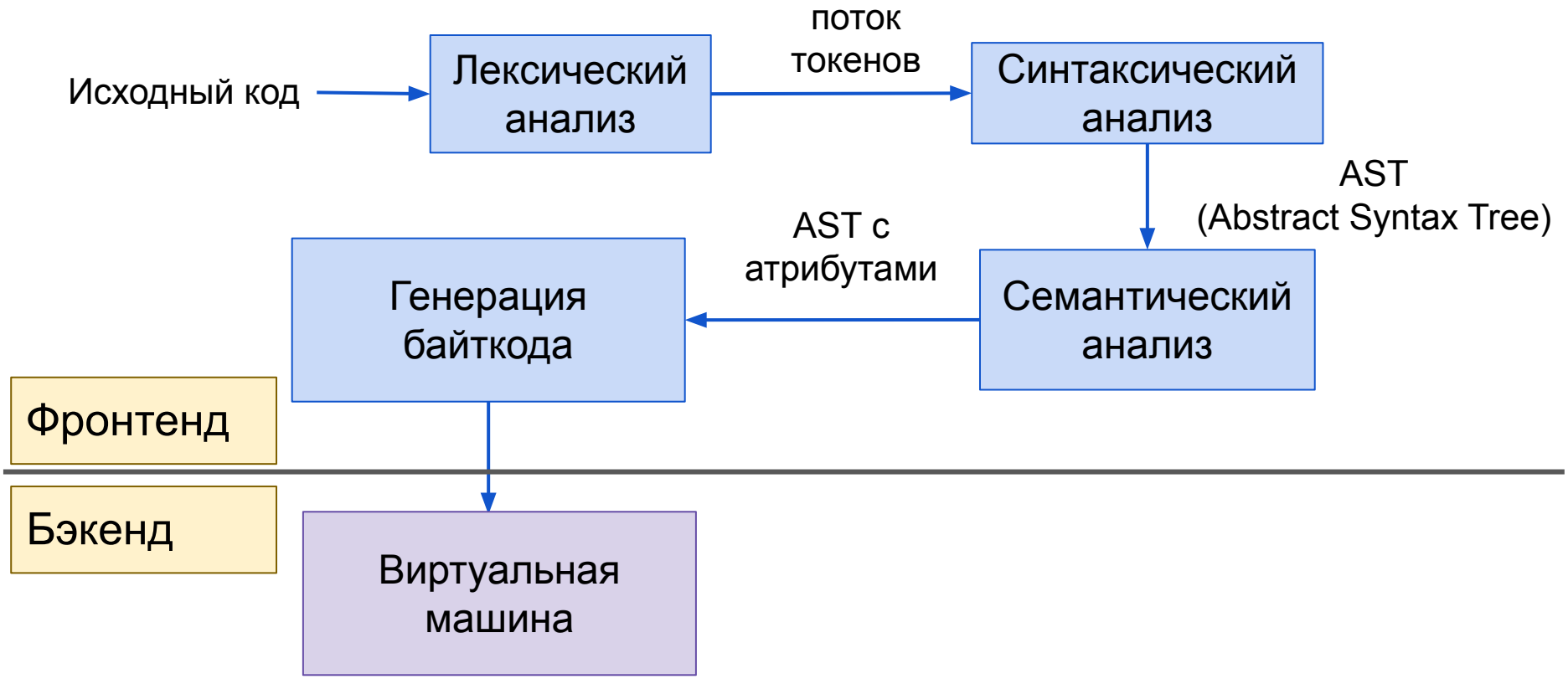
Архитектура компилятора



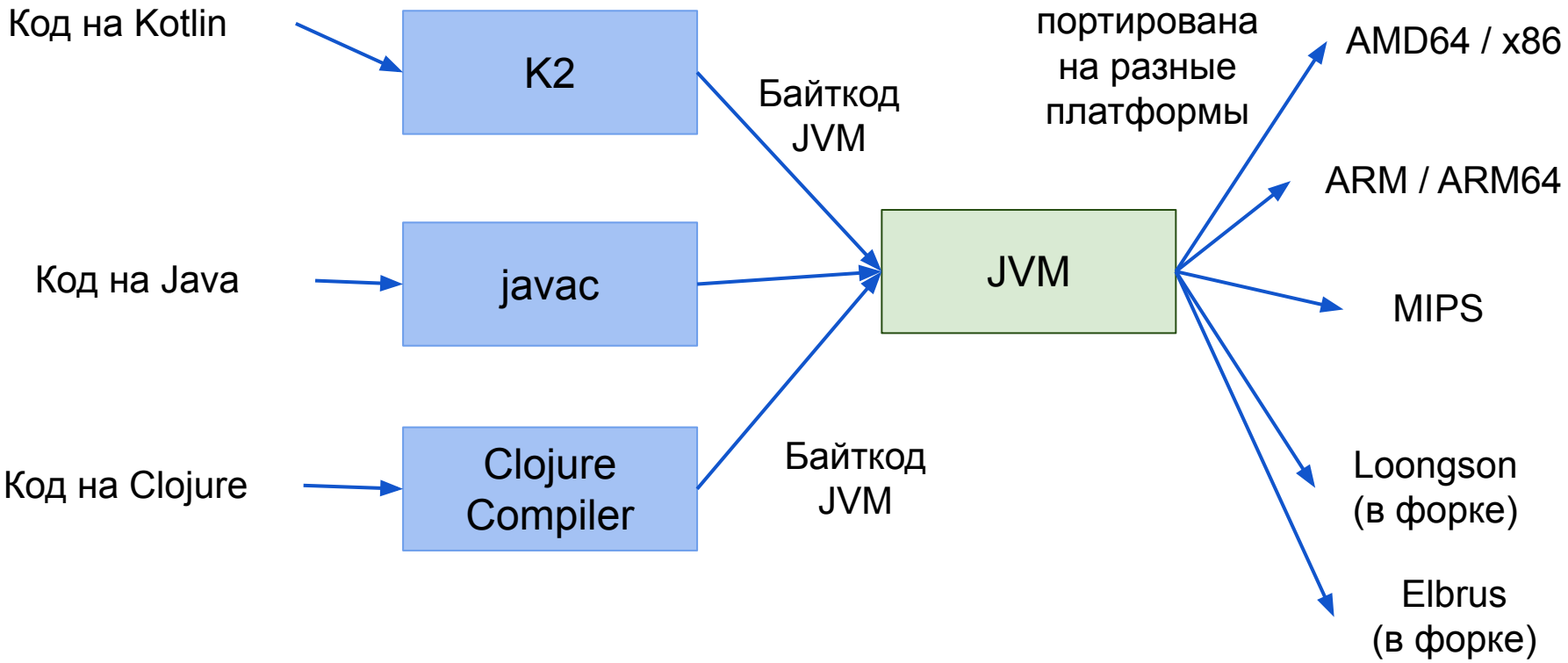
LLVM и его компиляторы



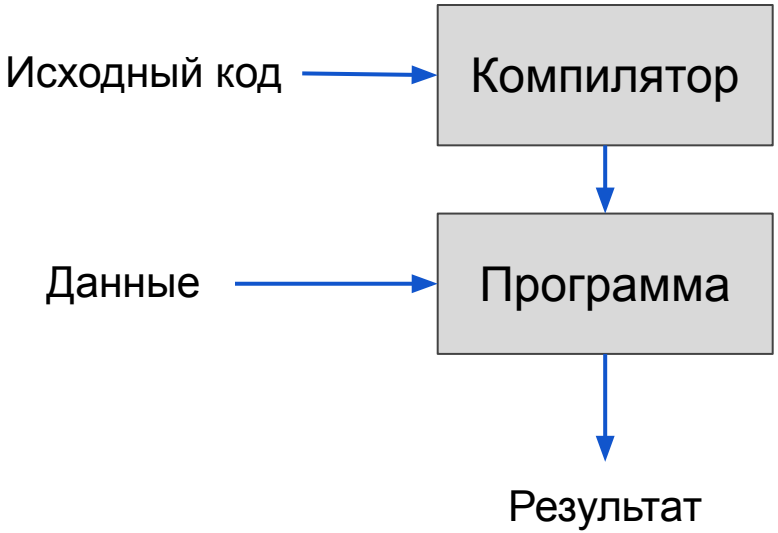
Компиляция для виртуальной машины



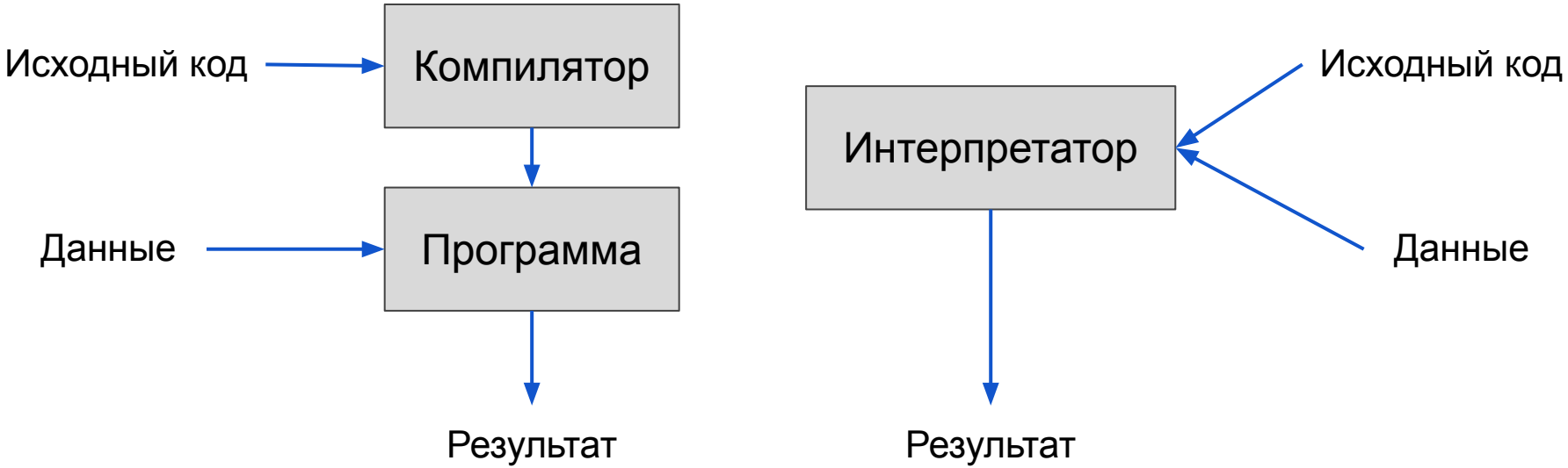
Java Virtual Machine



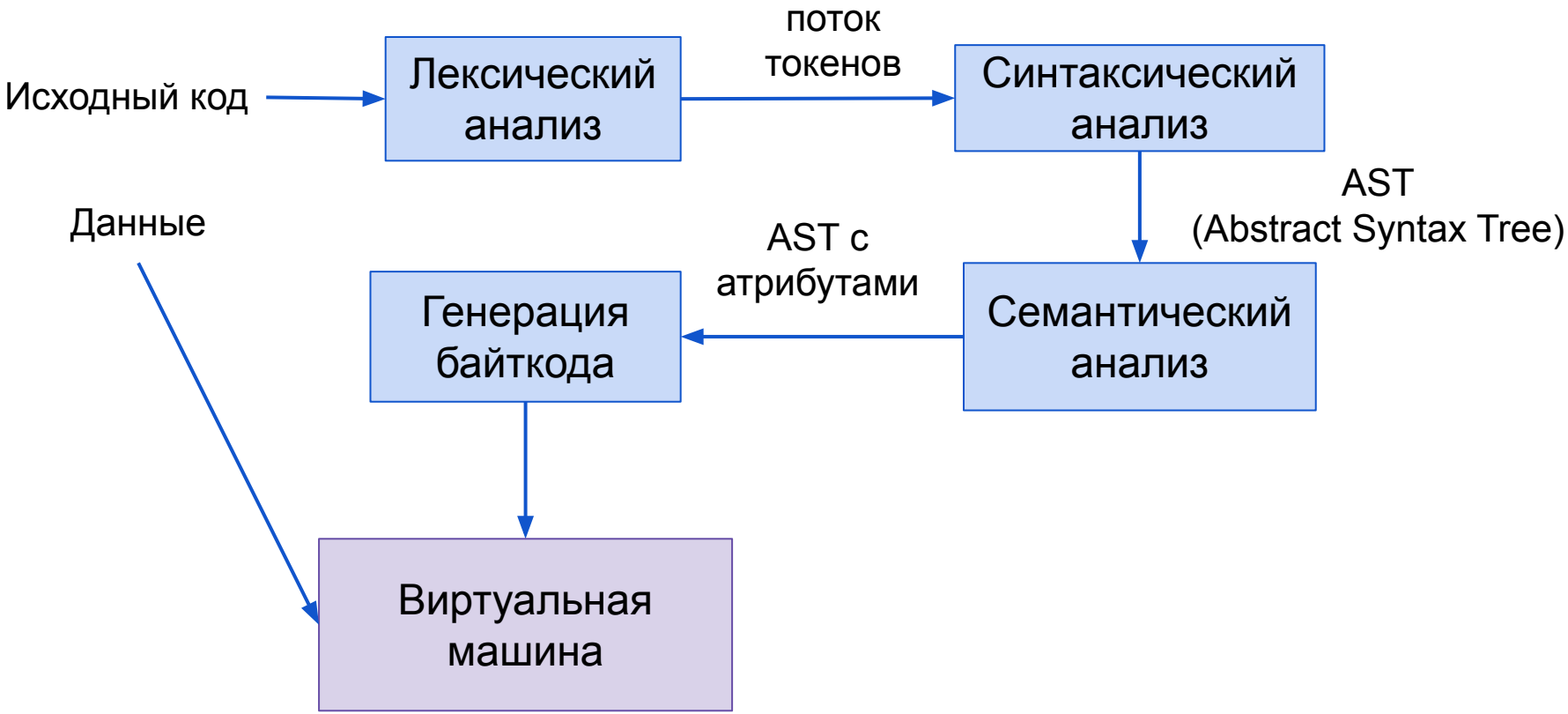
Компилятор и интерпретатор



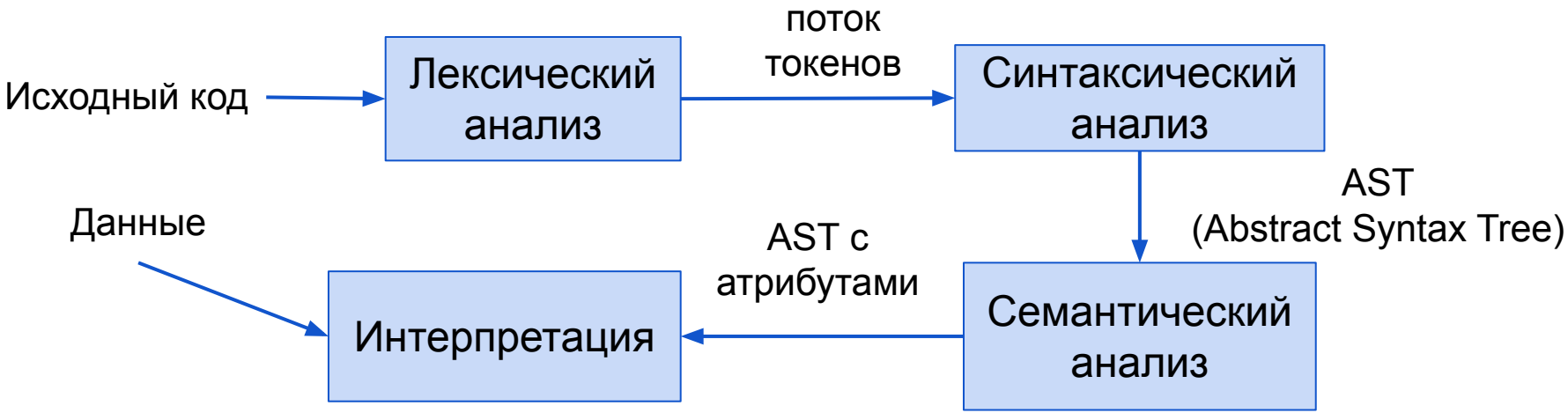
Компилятор и интерпретатор



Интерпретация через виртуальную машину



Интерпретация без виртуальной машины



Цель курса — проект

ТАиФЯ (5-й семестр, зачёт)

Цель — интерпретатор

ТЯП (6-й семестр, экзамен)

Цель — компилятор

Цель курса — проект

ТАиФЯ (5-й семестр, зачёт)

Цель — интерпретатор

1. Лексер — текст в токены
2. Парсер — разбор грамматики
3. AST — модель программы
4. Интерпретация данных

ТЯП (6-й семестр, экзамен)

Цель — компилятор

Цель курса — проект

ТАиФЯ (5-й семестр, зачёт)

Цель — интерпретатор

1. Лексер — текст в токены
2. Парсер — разбор грамматики
3. AST — модель программы
4. Интерпретация данных

ТЯП (6-й семестр, экзамен)

Цель — компилятор

1. Промежуточный код (IR) — MSIL, LLVM IR или иной
2. Простые оптимизации IR
3. Генерация машинного кода
4. Runtime языка

Команды и роли

Три роли:

1. Аналитик (A)
2. Разработчик (D)
3. Эксперт (E)

У каждой роли — свои задания.

Команды и роли

Три роли:

1. Аналитик (A)
2. Разработчик (D)
3. Эксперт (E)

У каждой роли — свои задания.

Команда — 2 или три человека

- Аналитик+Разработчик
- Аналитик+Разработчик+Эксперт

Языки программирования

Выберите язык, на котором вы пишете интерпретатор и компилятор.

Доступные варианты:

1. C# — доступен всем желающим
2. C++ — если была оценка 4 или 5 по ООП
3. Go — если пишете на Go на работе

Вопросы?