Разработка мобильных приложений

Лекция 2

Архитектура

Ключев А.О. к.т.н., доцент ФПИиКТ Университета ИТМО

Санкт-Петербург

2025

Контакты

- Ключев Аркадий Олегович
- ВК: https://vk.com/aokluchev
- E-mail: kluchev@yandex.ru
- Закрытая группа в Telegram «РМП 2025 весна»

Литература

- Hatley D.J., Pirbhai I.A. Strategies for Real-Time System Specification. - N.Y. Dorset House Publishing, 1988.
- Роберт Мартин. Чистая архитектура.
- Фредерик Брукс. Мифический человеко-месяц, или как создаются программные системы.
- Непейвода Н. Н. ,Скопин И. Н. Основания программирования.
- Гради Буч. Объектно-ориентированный анализ и проектирование с примерами приложений на C++
- Э. Рэймонд Искусство программирования для Unix
- Б. Керниган, Пайк Практика программирования
- Линус Торвальдс, Дэвид Даймонд. Just for FUN. Рассказ нечаянного революционера
- Ключев А.О., Кустарев П.В., Ковязина Д.Р., Петров Е.В. Программное обеспечение встроенных вычислительные систем https://books.ifmo.ru/file/pdf/499.pdf

Цель лекции

Показать важность изучения архитектуры ПО, методов проектирования архитектуры ПО, а также добавить мотивации к дальнейшему изучению компьютерных наук.

Проблемы

- В последнее время преподавателя стали замечать, что студенты, немного освоившие программирование на начальном уровне и нашедшие работу по специальности с неплохой зарплатой возомнили себя крупными специалистами. Необходимость в большом количестве низко квалифицированных программистов связана с возникновением сиюминутного бума в области разработки информационных систем в нашей стране.
  - Мотивация к изучению чего-то нового в университете слабая
  - ChatGPT и его аналоги помогут написать практически любую программу для новичка
  - На любой простой вопрос по программированию есть ответ на сайте stackoverflow
  - Отношение к занятиям в университете крайне скептическое
  - Посещаемость занятий низкая
- Результат – однобокое, поверхностное образование и как следствие – невозможность использования таких «специалистов» в более-менее сложных проектах.

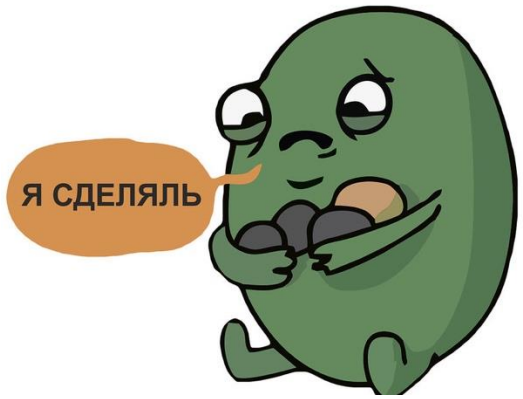
Размышления о стартовой позиции

- Кому проще стать:
  - миллионером?
  - инженером?
  - торговцем?
  - художником, музыкантом, певцом,…

Начальный толчок

- Вас подталкивает к тому или иному роду деятельности ваше окружение, давая правильные шаблоны поведения и работающие методики достижения целей в сумбурном и непонятном мире, больше всего напоминающем большое минное поле.
  - В семье ученых у ребенка есть шанс попасть в науку
  - В семье инженеров ребенку проще потом будет найти работу в области инженерной деятельности
  - В семье торговцев у ребенка скорее всего со временем появится свой бизнес и он будете поражать культурных людей своими безграмотными постами на форумах, где обсуждаются дорогие автомобили (зайдите на автомобильные чаты БМВ,Ауди или Мерседес)... У этого человека будет процветающий бизнес без всяких MBA и дорогих бесполезных курсов по бизнесу.
  - У людей занимающихся инвестициями и дети скорее всего будут понимать в акциях и облигациях гораздо больше сверстников.
  - Если ребенку посчастливилось родиться в с семье сильных мира всего, то ему открыта дорога в лучшие университеты мира, государственные или корпоративные правящие структуры (**почитайте книгу Михаила Хазина «Лестница в небо»,** она сильно пошатнет ваше представление о мироустройстве). Все дело в связях и понимании того, как устроен этот мир на самом деле.
- Переход между этими вариантами развития возможен, но он весьма трудоемок. Например, переход из среднего класса в богатый (или в верх среднего) может потребовать десятки лет. Идея перешагнуть эту пропасть быстро хороша, но мало осуществима на практике. В год создается примерно 50 миллионов стартапов. Единорогами (капитализация от миллиарда долларов) стали около 1000. Доля успешных (не разорившихся) стартапов тоже не очень велика.

Компоненты успешного бизнеса



- Допустим, вы захотели создать стартап в ближайшем будущем. Что вас ждет?
- Основные компоненты бизнеса:
  - Ваши компетенции
    - Диванный эксперт по многим вопросам, умею пользоваться ChatGPT и Stack Overflow, а еще у меня лапки?
  - Ресурсы (деньги, специалисты, материалы)
    - Денег нет, специалистов нет, ничего нет…
- На момент создания стартапа (например, через год) оба ваших компонента (компетенции и ресурсы) как правило близки к нулевой отметке, а вероятность провала близка к 100%

Современная «кастовая система» и социальный лифт

- Многие заметили, что в западном мире постепенно зарождается «кастовая система», разделяющая общество на три группы:
  - Необразованное большинство, с низкой оплатой
  - Средний класс, имеющий высшее образование и научные степени (инженеры, ученые, врачи, программисты, менеджеры, деятели культуры...)
  - Богатые люди, мультимиллионеры и миллиардеры, владельцы бизнеса (бизнесов), инвесторы
- Фактически «общество равных возможностей» перестало быть таковым.
- Несмотря на то, что Россия еще недавно была частью СССР где были равны все, процессы расслоения общества начались и у нас. К счастью, попасть в средний класс у нас еще достаточно просто, так как образование в России пока еще бесплатное (например, в США стоимость обучения в не самом лучшем колледже примерно на порядок выше стоимости контрактного обучения в ИТМО). Точно также у нас пока нет «Лиги Плюща». Диплом этих университетов является фактически готовым входным билетом в мир инвестиций при создании своего стартапа (если вы закончили какой-то неизвестный университет вы скорее всего просто не получите денег от инвесторов). Для справки, стоимость обучения в Гарварде около $55к в год (а еще нужно что-то есть и платить за проживание), что автоматически отсекает от такого социального лифта всех, кто не входит в высший или средний класс.

Можно ли идти напролом через это «минное поле»?

- Можно, если вы гений (или близки к этому) и обладаете огромной силой воли. Яркий пример такого - Михаил Ломоносов. У него получилось пройти путь от неграмотного крестьянина до академика.
- У меня есть для вас плохая новость: к сожалению, до финиша дойдут не все.

В чем суть проблемы?

- Если отсечь финансовый фактор (у нас в РФ это пока не является совсем уж большой проблемой), вероятность правильного выбора маршрута своего развития (с обходом всех подводных камней и ловушек) довольно низка.
- Например, я знаю короткий путь в высшее образование и последующее трудоустройство по специальности. Но я родился в семье инженеров и меня с детства готовили быть этим самым инженером, пусть и неосознанно. Даже в мыслях не было другого пути, кроме технического высшего образования.
  - Разные конструкторы с раннего детства
  - Электрический конструктор в 7 лет (я для школы сам сделал работающий игрушечный светофор).
  - Конструктор «250 опытов по физике»
  - Популярная литература по технике, космическим исследованиям, физике (например, «Занимательная физика» Перельмана), астрономии, биологии и т.п.
  - Куча научно-популярных журналов, доступных мне с детства
  - Научная фантастика (с 10 лет я начал читать книги с утра до вечера), а когда я был маленький, отец читал мне книги Стругацких вместо сказок.
  - Радиокружок, радиолюбительство с 4-го класса и т.д.
- На самом деле, стать инженером после школы было мейнстримом во времена СССР. На развитие детей государство тратило огромные деньги. Сейчас такого давления со стороны государства нет, хотя возможностей по развитию стало гораздо больше. Но сейчас, возможностей для того, чтобы свернуть «не туда» тоже гораздо больше, «минное поле» нашей жизни имеет большую площадь, да и мин стало значительно больше.

Что я понял?

- Инженеру нужна фантазия. Мне в этом смысле очень помогла научная фантастика в огромных количества. Инженер должен жить в будущем (я мечтал о таких штуках как Интернет, ИИ, ноутбук, смартфон или планшет примерно с 70-х годов, поэтому их появление для меня не стало шоком) и думать о таких вещах, которые в голову обычному человеку никогда не придут. Сериал «Теория большого взрыва» смотрится как родной, несмотря на то, что он показывает жизнь в США, а я жил и учился еще во времена СССР.
- Инженеру очень важно уметь думать: обобщать, сопоставлять (находить зависимости) , приводя огромные объемы информации в онтологии и создавать компактные словари терминов. Для этого нужно уметь обрабатывать огромные потоки информации, а для этого, в свою очередь, нужно очень много читать.
- Инженеру нужно уметь разбирать системы на подсистемы и потом собирать их в другом порядке, это возможно благодаря так называемому комбинаторному мышлению. Очень хорошо, если оно у вас появится к концу четвертого курса. Замечательно, если оно у вас уже есть, тогда вам нужно идти на следующую ступень своего развития.

На что я намекаю?

- Итак:
  - Если у вас нет фантазии,
  - Если вы не умеете обобщать, сопоставлять, строить классификации, давать определения, делать выводы,
  - Если вы не обладаете комбинаторным мышлением,
  - Если вы не хотите все это научиться делать

Если все это так, то хочу вас огорчить, вы вошли не в ту дверь и ваш предел это позиция вечного джуна в крупной корпорации или судьба офисного планктона.

Уровни знаний и умений (Непейвода)

- Условные рефлексы
- Комбинаторное планирование
- Стратегическое мышление
- Методическое мышление

Подробности читайте:

- Н.Н. Непейвода, «**Основания программирования**», глава «Знания, данные, умения».
- Н.Н. Непейвода, «**Инфософия, введение в системный и логический анализ**», курс лекций

На что влияет уровень мышления?

- На масштаб и сложность решаемой задачи
- На качество и оптимальность решения
- На риск (например, при комбинаторном решении риск очень велик, при стратегическом он меньше, а при методическом он обычно мал)

Ваши перспективы на позиции «джуна»

- В крупной корпорации у вас практически нет шансов на рост, если вы сами не будете прилагать титанические усилия в области своего профессионального роста. Ну и не забываем про классическое «ведро с крабами», если вокруг вас будут одни неудачники.
  - «Джунов» можно использовать как взаимозаменяемый расходный материал, крупная фирма может себе позволить держать много низкосортных сотрудников и специальных людей, которые могут формулировать им примитивные задачи.
  - Компаниям выгодно держать у себя закредитованных неудачников
- В мелкой компании у вас шансов на рост существенно больше, но в таких компаниях джуны вообще не нужны (они не приносят прибыль и обходятся очень дорого). Чтобы вы начали приносить хоть какую-то пользу вы должны расти и вас будут скорее всего стимулировать это делать. Другой вопрос, что в такую фирму будет гораздо труднее попасть.
- Имеет ли смысл иметь опыт? Наверно да, но если вы не тянете на позицию выше джуна, то у вас нет шанса перемещаться по карьерной лестнице вверх, только по горизонтали между корпорациями.

К чему все это?

- Я просто подвожу вас к мысли, что разработка ПО это трудоемкая, сложная, интеллектуальная деятельность, доступная далеко не всем. «Войти в IT» просто только до позиции джуна, дальше начнутся очень капитальные проблемы. Если у вас нет достаточно развитого мозга (или на момент попадания в потогонную систему корпорации вы его не успели развить), то вам грозит выгорание.
- Тут полная аналогия с компьютером. Вы можете запустить одну и ту же игру на одноядерном CPU с низкой тактовой частотой и на многоядерном процессоре с высокой тактовой частотой. В первом случае ресурсы процессора будут использованы полностью, во втором лишь частично.
- Если вы хорошо подготовлены, то нагрузка на работе вам будет не очень заметна и вы сможете тратить время на развитие своих способностей. В случае плохой подготовки вы будете годами работать на пределе своих возможностей, будете хронически уставать и не сможете развиваться и расти по карьерной лестнице.

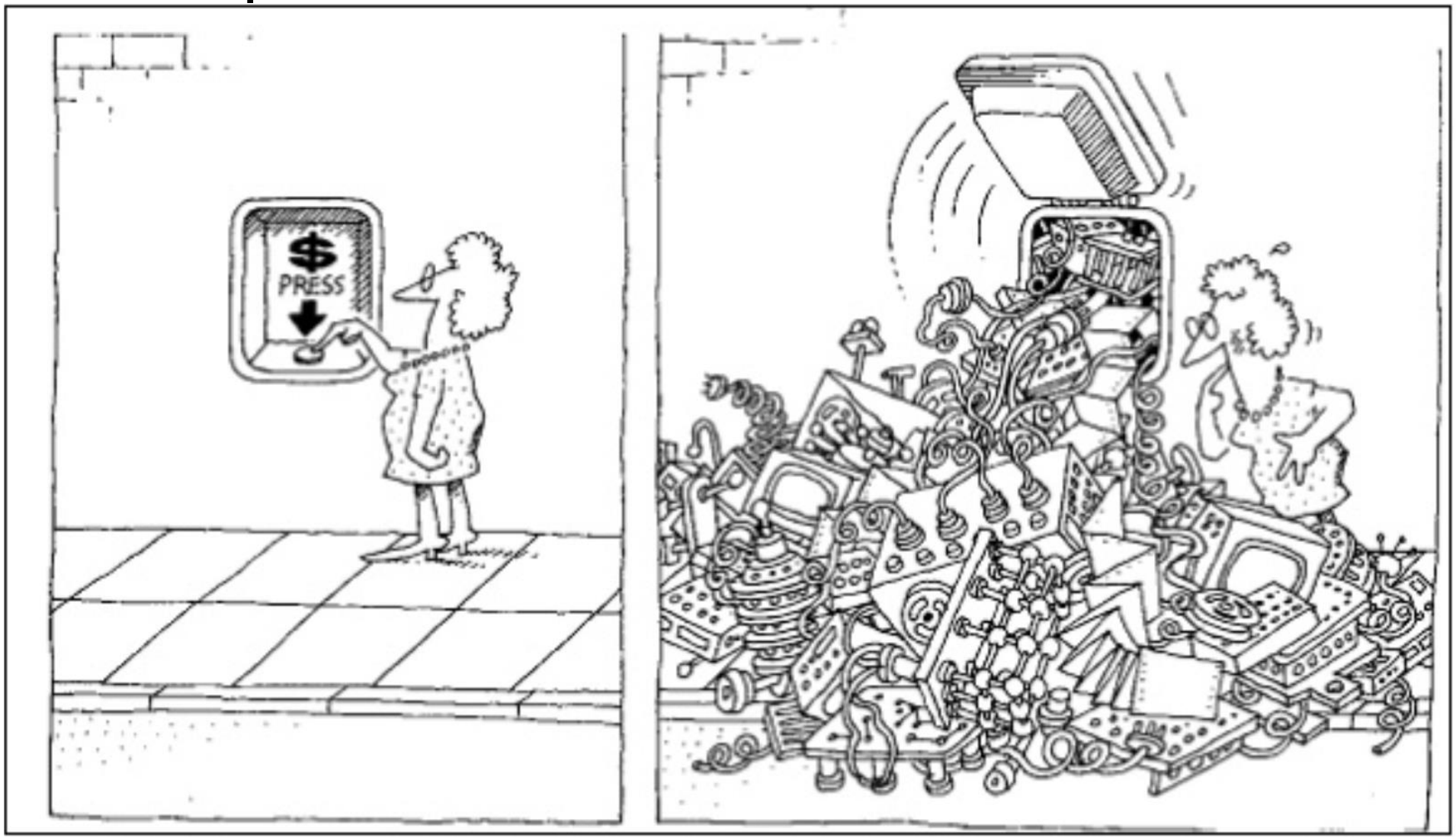
Что мы имеем в сухом осадке?

- Ваша успешность как разработчика зависит от нескольких факторов:
  1. Умение думать (тут нужны умения, которые вам бы помогли развить ваши родственники или друзья, работающие в схожей области до поступления в университет или в процессе обучения)
  2. Биологические способности вашего головного мозга (ну, тут мы бессильны). Различие может составлять до нескольких десятков раз и зависит от площади определенных участков мозга. Разные люди имеют разную конфигурацию мозга.
  3. Наличие знаний, умений и навыков.
- Если вы круглый отличник, но у вас не очень развиты п. 1-3, то вы будете всю жизнь работать офисным планктоном, а ваш красный диплом будет пылиться на полке. Это к вопросу о ценности ваших оценок. Эта ценность стремится к нулю. Никому не нужна говорящая википедия, которая не понимает того, что она говорит, для этой цели проще использовать какой-нибудь ChatGPT. Исходя из этого корректируйте свой способ обучения. Вас спасет большой объем практики и чтение, с последующими размышлениями.

При чем тут этот курс?

- Это была мотивация. Заставить делать студентов что-то нужное для них самих ОЧЕНЬ СЛОЖНО. Почти невозможно, так как вы знаете как устроен мир лучше преподавателя и вообще, вы самые умные на свете.
- Дело в том, что сложные системы не помещаются в мозг обычного человека даже по частям. Сложные системы - сложные, непонятные, нудные, контринтуитивные, их сложно разрабатывать, программировать, отлаживать, эксплуатировать и главное, их сложно делать по заданному ТЗ, в рамках заданного бюджета в заданные сроки. Для вас я бы пока заменил слово «сложно» на «невозможно в принципе».
- Проект, который мы сейчас разрабатываем – пример сложной системы и если вы хотите получить пользу от курса, вам придется напрягать голову и думать, даже если у вас уже есть опыт работы (опыт «джуна» вам точно вам не поможет).
- Ну а для того, чтобы разрабатывать сложные системы, вам нужно разбираться в такой штуке как АРХИТЕКТУРА...
- Причем желательно ПОНЯТЬ, что такое архитектура, а не пытаться применять разные инструменты наобум.

Сложность проекта



Архитектура

- Архитектура — логическая и физическая структура системы, сформированная всеми стратегическими и тактическими проектными решениями. (Г. Буч.)
- Архитектура программного обеспечения – это ряд значительных решений об организации программной системы, выборе структурных элементов и их интерфейсов, с помощью которых собирается система, а так же их поведение во взаимодействии с этими элементами, объединение этих структурных и поведенческих элементов в большие системы и архитектурный стиль этой организации – этих элементов и их интерфейсов, их взаимодействия и их объединения.
- Архитектура объясняет концепцию работы вашей системы, как ваша система устроена внутри, как работает, как ее делать, отлаживать, сопровождать и использовать.

Структура и архитектура

- Структура это совокупность элементов и связей между ними.
- Архитектура – более ёмкое понятие, включающее в себя множество структур.
- В рамках любой модели вычислений систему можно представить как структуру, в виде совокупности акторов и связей между ними.

Структура и графы

- Любую структуру в вычислительной технике можно изобразить в виде графа
- Граф это совокупность вершин (кружочков или квадратиков) и ребер (стрелочек)
- Вопрос в том, что вы должны понимать, что означают эти кружочки/квадратики, а что означают стрелочки в ВАШЕМ конкретном случае. Если вы не понимаете семантику элементов схемы, ваша схема не имеет смысла!
- От наполнения (семантики, смысла) этих элементов графа зависит то, что за структуру вы нарисовали. Это может быть:
  - Структурная схема вашего оборудования
  - Диаграмма потоков данных или сеть процессов Кана
  - Конечный автомат
  - Блок-схема алгоритма
  - Электрическая принципиальная схема
  - Диаграмма классов
  - и многое другое... (см. книгу Г. Буча, там много интересного)

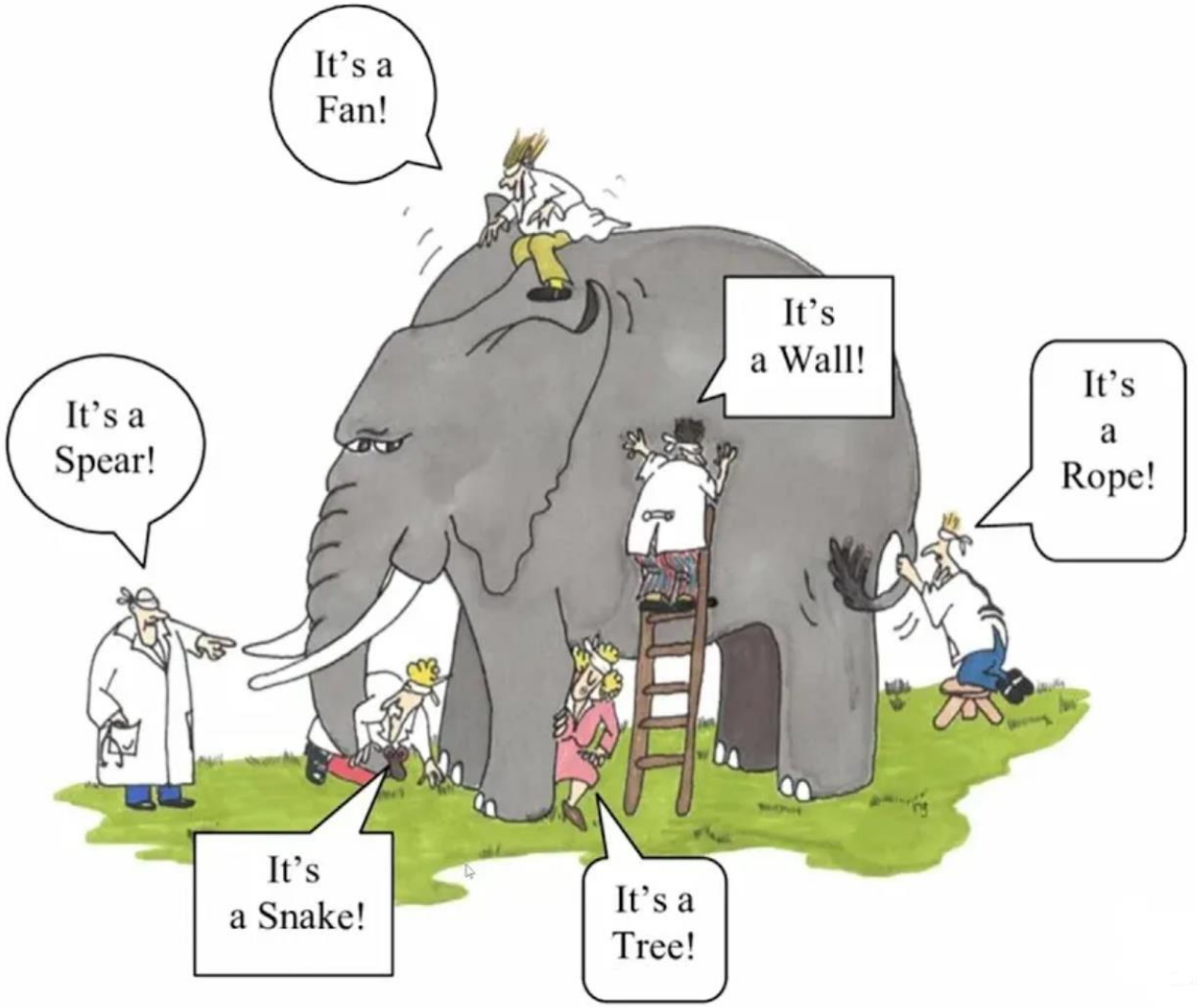
Зачем нужна архитектура

- Позволяет понять систему
- Позволяет упростить систему
- Позволяет объяснить другому специалисту как устроена и как работает система

- Если вы пишете программу и не очень себе представляете, что у вас получится в конце – то это скорее всего кончится провалом.
- Знаете в чем проблема джуна? Он думает о решении проблем не выходя за рамки своего языка программирования (Java, C#, C++, Go,... – не важно, какой язык), а также используемых библиотек и фреймворков.
- Нужно учиться думать на различных уровнях абстракции, пользоваться разными описаниями системы. Язык описания архитектуры – выше по уровню чем любой язык программирования, он позволяет увидеть систему целиком, с высоты «птичьего полета».

Что будет, если вы не будете разрабатывать или понимать архитектуру?

- Байка про слона и муравья: что увидит муравей сидя на спине слона?
- Притча о слепых мудрецах и слоне.
- Анекдот про ключи, которые человек искал под фонарем, потому что там светло.



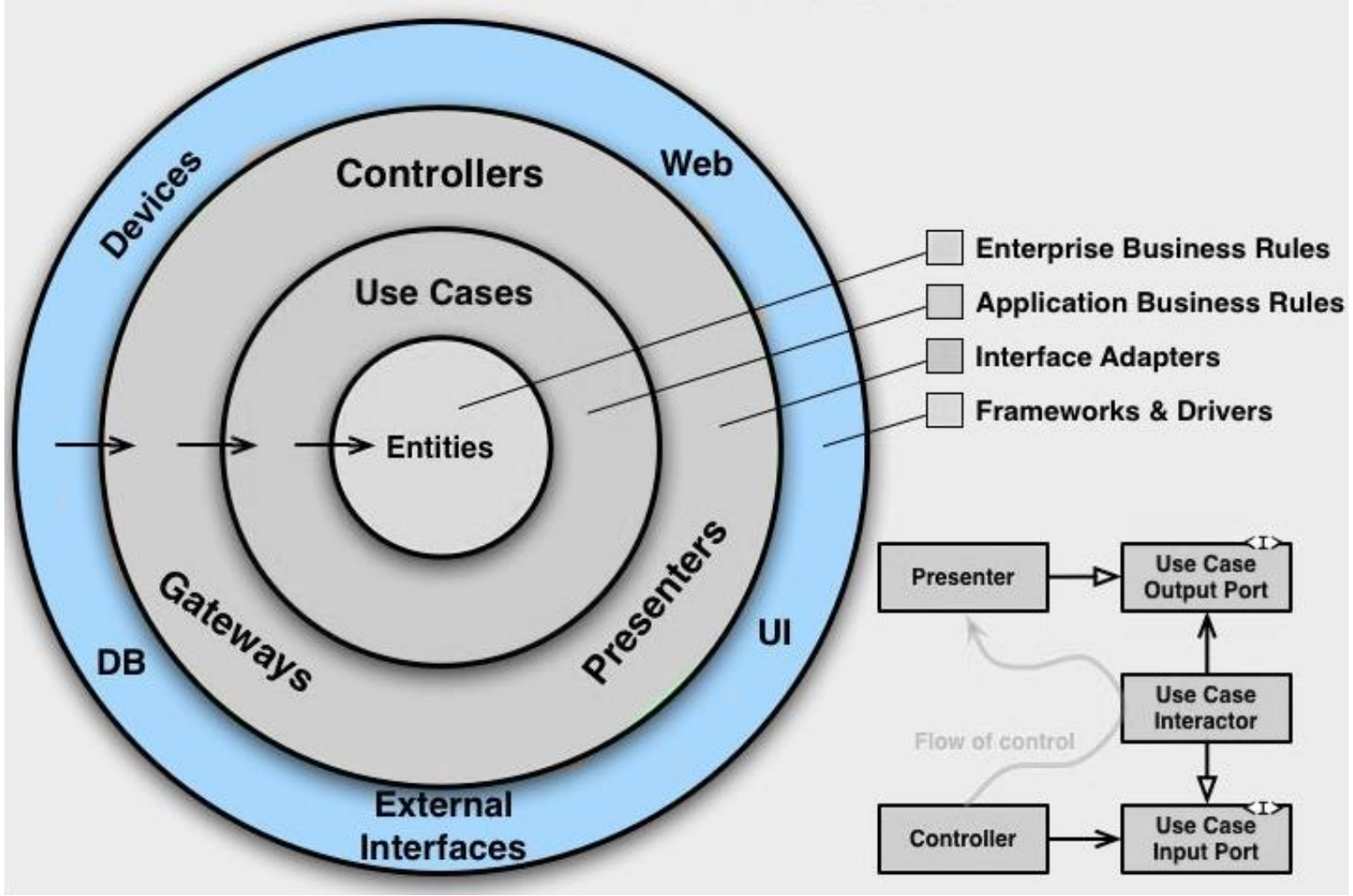
Как будем описывать проект?

- Максимально просто, но не проще, чем это необходимо.
- Используйте принцип KISS (Keep it simple, stupid» — «Делай проще, тупица») по максимуму. В этом смысле я вам рекомендую обратиться к трудам авторов языка программирования C, а также операционных систем Unix и Linux.
- Не мудрите, не пытайтесь меня поразить своими познаниями в UML или еще чем-либо. Я таки могу подумать, что вы зубрилы-отличники и перестану вас уважать. ;)
- Мы станем использовать максимально простые средства описания архитектуры, а для начала попытаемся понять концепцию модели вычислений.
- Нельзя чему-то научиться в области проектирования сложных систем просто тупо копируя чужие решения и изучая всякие UML (и другие ужасы). Ребенок с молотком и гвоздями не построит дом, для начала этому ребенку нужно стать строителем. Мы попытаемся начать с детского конструктора.

Уровни системы

- Важный момент. Почитайте книгу «**Чистая архитектура**»! Без понимания концепций изложенных в этой книге все ваши более-менее сложные программы будут годиться только для помойки. Эх, если бы у меня 30 лет назад были такие книги...
- Я лет 5 программировал без понимания уровней и архитектуры, пока не уперся в сложную задачу при разработке системы управления наружным освещением громадного завода.
- Работа встала, я не смог продвинуться ни на миллиметр (где-то 6 месяцев), пока до меня не дошло, как поделить систему на уровни.

Уровни системы (чистая архитектура)
(не принимайте это как единственный вариант!)



Пример организации уровней в цифровом вольтметре

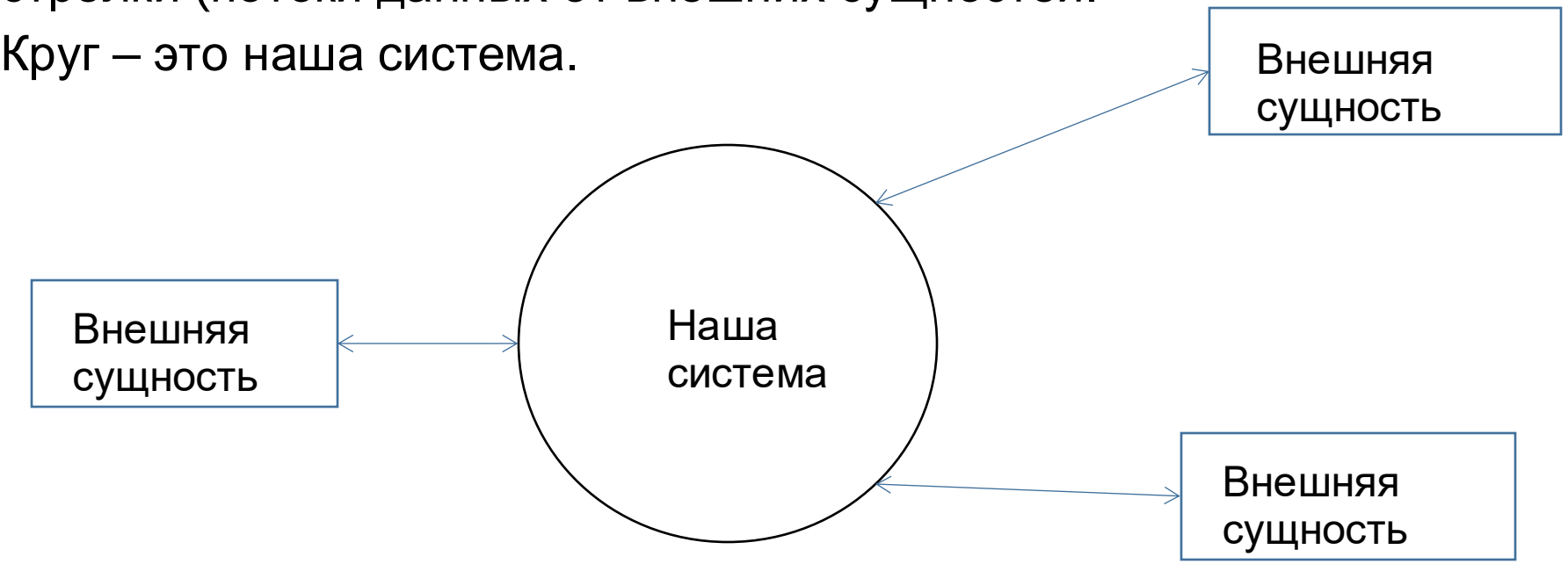
- Допустим, мы хотим сделать точный цифровой вольтметр на базе аналого-цифрового преобразователя (АЦП) и микроконтроллера. Для этого мы пишем несколько драйверов, которые последовательно обрабатывают данные и позволяют программе вывести результат на дисплей напряжение в вольтах:
  - Драйвер SPI или I^2C позволяет добраться до регистров микросхемы АЦП
  - Драйвер АЦП принимает данные от микросхемы АЦП
  - Фильтр убирает помехи
  - Система калибровки позволяет учесть уникальные характеристики аналогового тракта измерителя напряжения (у резисторов и конденсаторов есть разброс параметров)
  - Преобразователь отсчетов АЦП в Вольты позволяет получить на выходе драйвера напряжение в Вольтах
  - Прикладная программа выводит полученное значения напряжения на дисплей вольтметра

Модель вычислений

- Модель вычислений, вычислительная модель (model of computation, MOC) — набор законов взаимодействия элементов вычислительной системы.
- Модель вычислений - набор правил организации вычислительного процесса, в рамках которых возможен его формальный анализ.
- Модель вычислений - набор формальных правил, в рамках которых организована взаимосвязь и поведение множества составляющих атомарных частей модели некоторой вычислительной системы.
- Модель вычислений - строго определенная парадигма (набор правил), описывающая протекание вычислительного процесса, способы обмена данными, взаимодействия между отдельными функциональными элементами.
- Модель вычислений - недвусмысленный формализм для представления спецификаций проекта и проектных решений.
- Модель вычислений - математическая модель, демонстрирующая пользователю вычислительные возможности вычислителя и правила их использования.
- *См. Ключев А.О., Кустарев П.В., Ковязина Д.Р., Петров Е.В. Программное обеспечение встроенных вычислительные систем https://books.ifmo.ru/file/pdf/499.pdf , раздел 2 «основные парадигмы и технологии программирования…»*

Контекст системы

- Контекст системы позволяет отделить то, что вы обязаны сделать в рамках ТЗ от того, что вы делать не должны.
- Контекст системы представлен в виде круга (это процесс) к которому подходят стрелки (потоки данных от внешних сущностей.
- Круг – это наша система.



Пример контекста системы

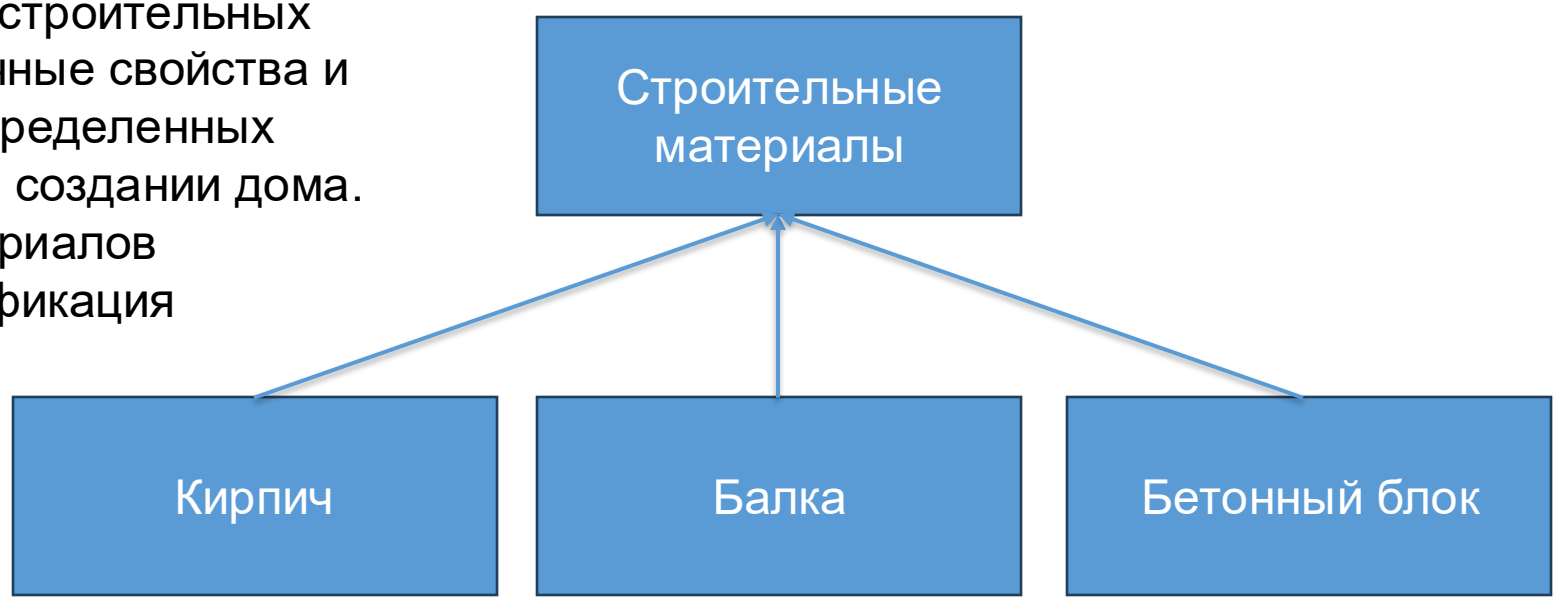
Внешняя сущность

- Это то, с чем взаимодействует наша система через какие-то каналы связи.
- Как эта внешняя сущность устроена и что она делает – нас не особо должно волновать.
- Для нас самое главное – как устроить корректный обмен данными с внешней сущностью, другими словами нас волнуют две вещи:
  - Интерфейс,
  - Протокол обмена.

Классификация

Классификация – декомпозиция системы на объекты, которые в свою очередь выстроены в виде иерархии
Например:

- Дом состоит из совокупности строительных материалов, имеющих различные свойства и требующих использования определенных строительных технологий при создании дома.
- Иерархия строительных материалов входящих в дом – это классификация материалов



Зачем нужно объектно-ориентированное программирование?

- Классификация объектов переносится с бумаги в исходный текст программы.
  - Это заставляет программиста лишний раз задуматься и сделать непротиворечивую классификацию
  - Система типов (каждый класс имеет свой тип) не дает делать грубые ошибки
  - Структура программы становится более прозрачной и простой
- Упрощается использование объектов за счет абстракции (выделения главного)
- Лишнее, ненужное и опасное прячется внутри объектов (инкапсуляция)
- Одинаковый по смыслу метод, например, ***send(message : String)***, позволяет в разных классах делать разные вещи: отправить сообщение в брокер сообщений, в очередь для связи с другим потоком, на принтер, в файл лога и т.п.
- Уменьшается объем кода (за счет того, что реализация частей сложных объектов как бы выносится за скобки)
- Упрощается отладка
- Упрощается сопровождение кода
- В результате – мы можем написать программу гораздо большего объема при той же квалификации программистов

Что не умеет делать ООП?

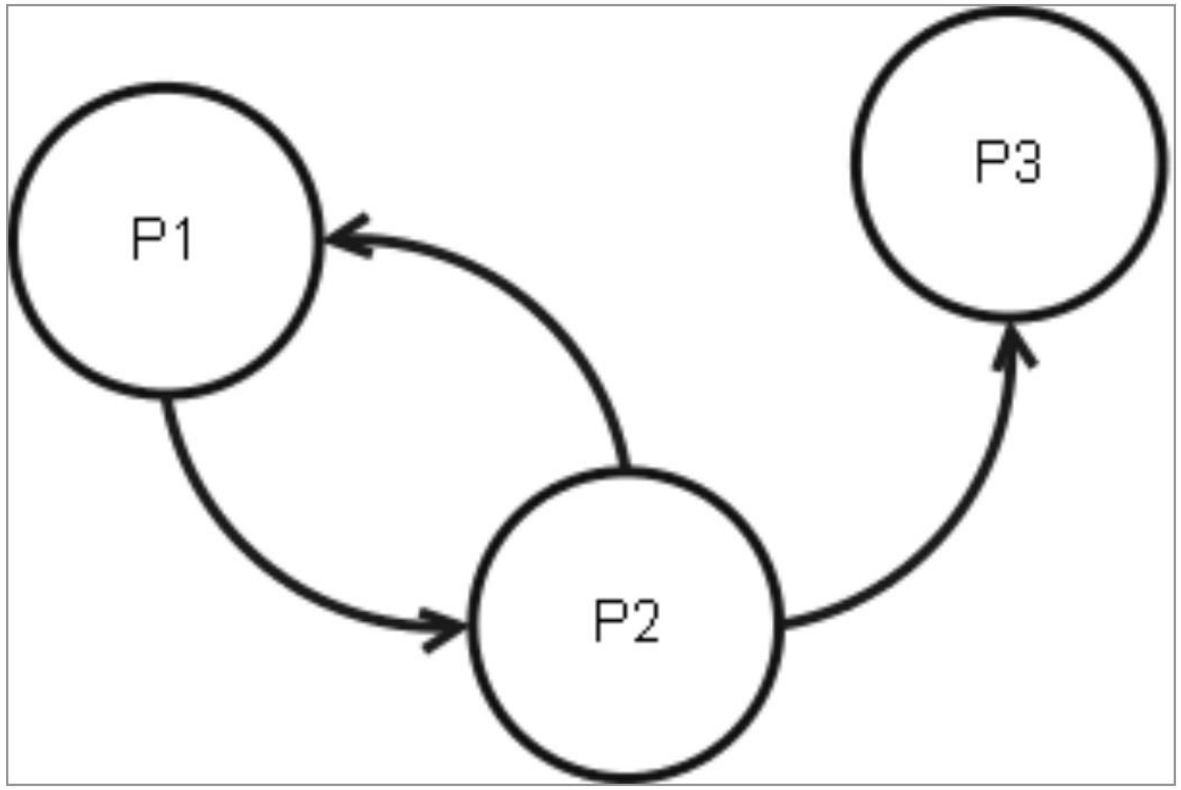
- Классификация дает вам возможность создать более-менее стройную иерархию строительных частей вашей программы, не более. Причем этот процесс не защищает вас от ошибок и разных противоречий (вы это увидите на практике в рамках данного курса).
- Чего не может ООП:
  - ООП не поможет вам справиться с асинхронными процессами
  - ООП не может решить проблему сложного поведения объектов (так же точно, как наличие диплома о высшем образовании не спасет вас от дождя, который вас застиг по пути на остановку автобуса).
  - ООП вам не поможет в создании алгоритмов связанных с решением задач в рамках заданного времени (системы реального времени)
  - ООП не даст вам способов решения задач экономии энергии или распределения процессов по вычислительным узлам

Что собой представляют кирпичики нашей программы?

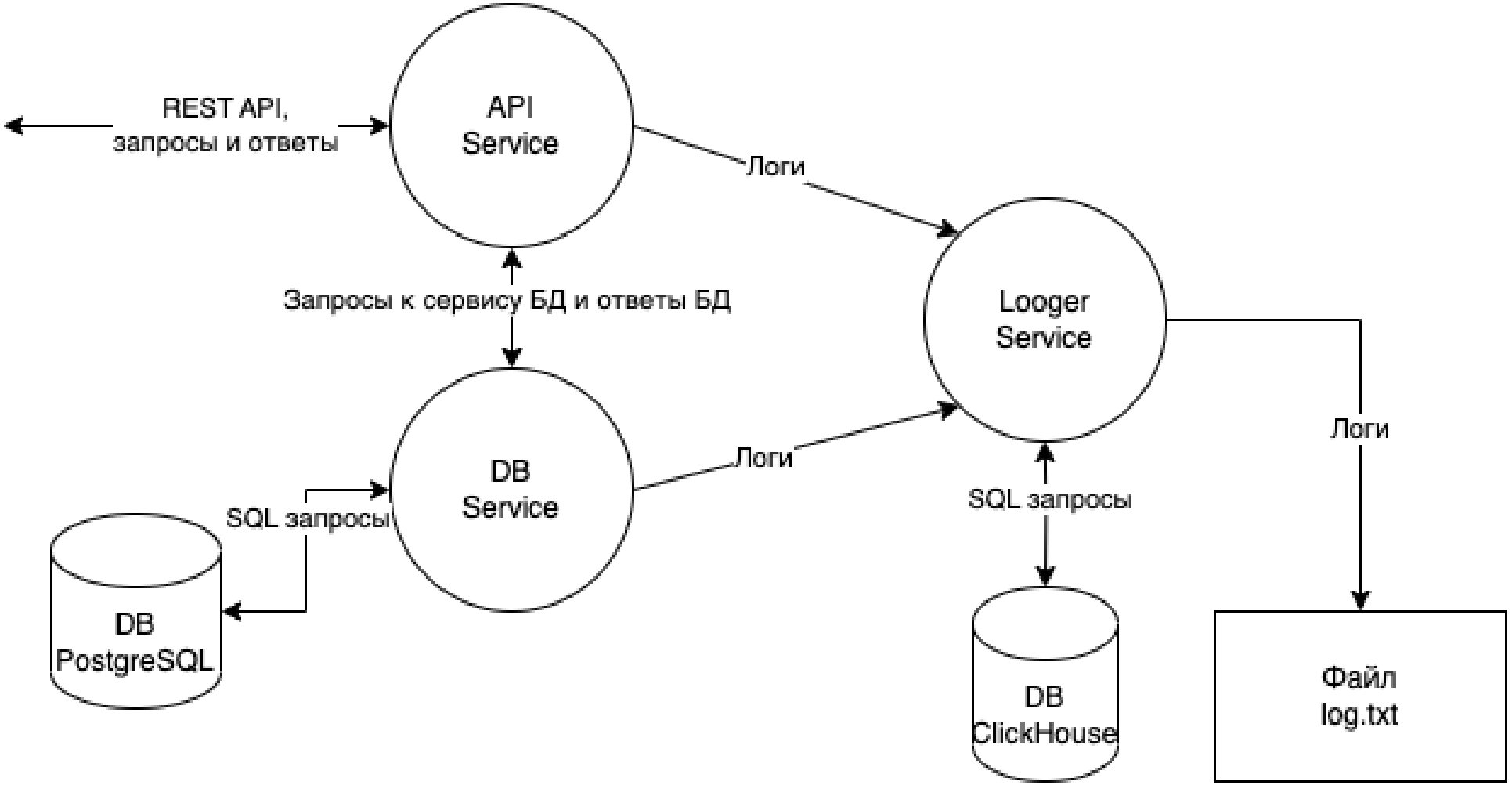
- Части из которых мы делаем программу (функции, объекты, переменные) – это части языка (не путайте с языком программирования!), с помощью которого мы пытаемся описать создаваемую систему. Язык программирования (C++, Java, Kotlin, C#, Go,…) задает лишь основу, в рамкой которой эти части позволяют нам создать нечто целое. В основе языка программирования, в свою очередь, лежит модель вычислений. Моделей вычислений может быть несколько, такие языки называют мультипарадигмальными.
- В процессе создания программы мы придумываем такие кирпичики, которые не противоречат концепции языка программирования (модели вычисления) и своему окружению.
- С помощью кирпичиков мы можем создать новую модель вычисления поверх имеющейся, например, реализовав переключатель задач (как в операционной системе) или виртуальную машину, с совершенно новыми свойствами. Создавая новые возможности мы создаем новый слой абстракции.
- Если в языке не хватает слов, то мы пытаемся рассказать о системе иносказательно, непонятно и многословно или не можем рассказать вообще ничего (например, на языке C++ сложно писать стихи, а на русском языке неудобно решать дифференциальные уравнения), а в языке древних людей вообще нет понятий, позволяющих говорить о создании программного обеспечения.
- Язык может хорошо описывать какие-то одни области и плохо описывать другие. Например, попробуйте с помощью набора понятий, входящих в язык школьника младших классов описать физические процессы внутри атомного реактора.
- Точно также в программировании вы не сможете описать какой-то специфический алгоритм, если набор доступных вам понятий не подходит для описания этого алгоритма (жители плоского мира Флэтляндия не смогут понять что такое трехмерный мир, в котором мы живем).

Сети процессов Кана

- Сети процессов Кана являются удобным инструментом для изображения взаимодействующих процессов.
- Process Network (PN), сеть процессов Кана, сеть потоков данных - модель вычислений, в которой система представляется в виде ориентированного графа, вершины которого представляют собой процессы (вычисления), а дуги представляют собой упорядоченные последовательности элементов данных.



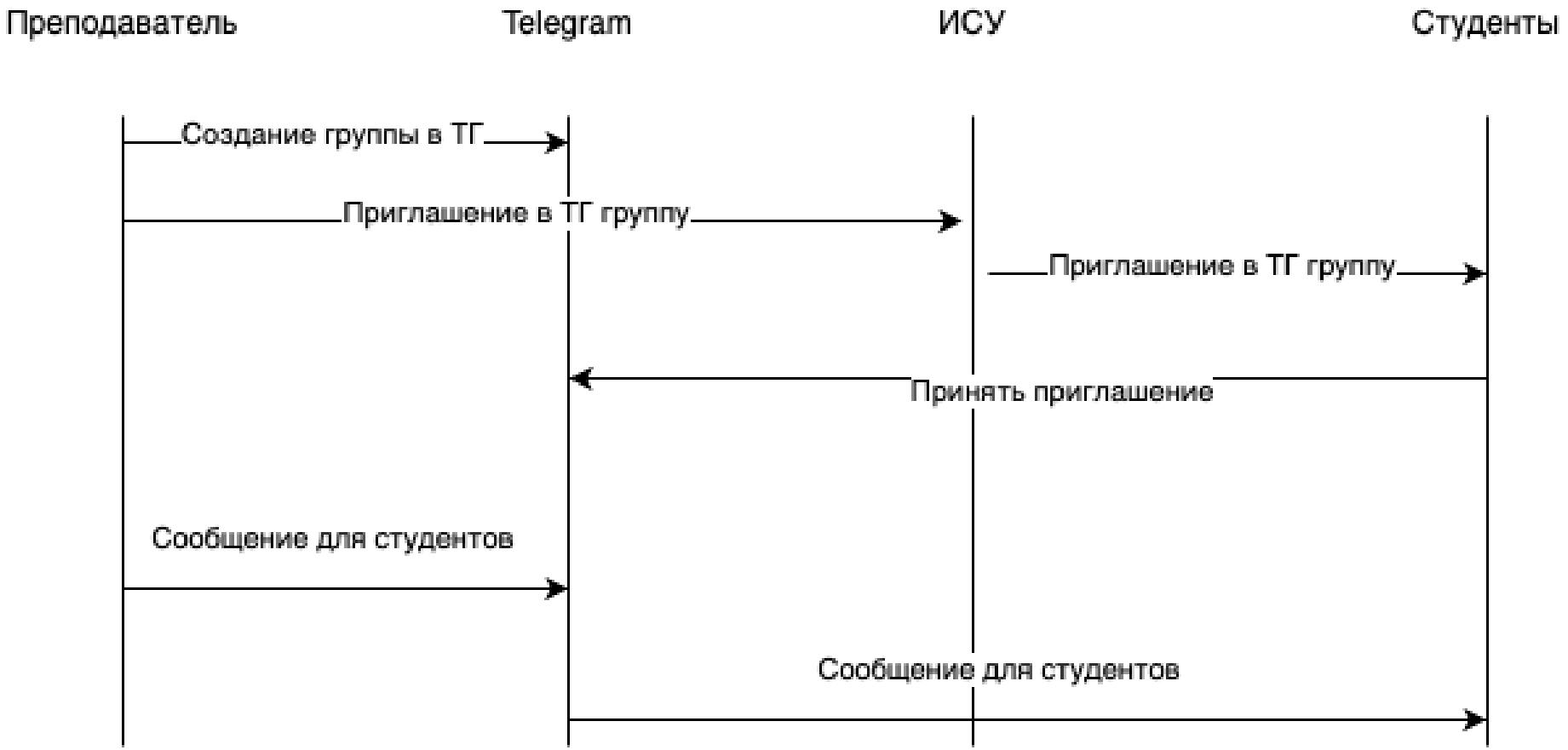
Пример использования сетей процессов Кана



Сценарий работы

- Сценарий работы (диаграмма последовательности) позволяет понять в какой последовательности процессы обмениваются сообщениями.
- Сценарий позволяет показать:
  - Тип сообщения
  - Какой процесс посылает сообщение, а какой принимает
  - Сколько времени занимает ответ на сообщение
  - Что будет, если сообщение не будет принято
- Сценарий показывает не все варианты обмена данными между процессами!

Сценарий работы (последовательности)



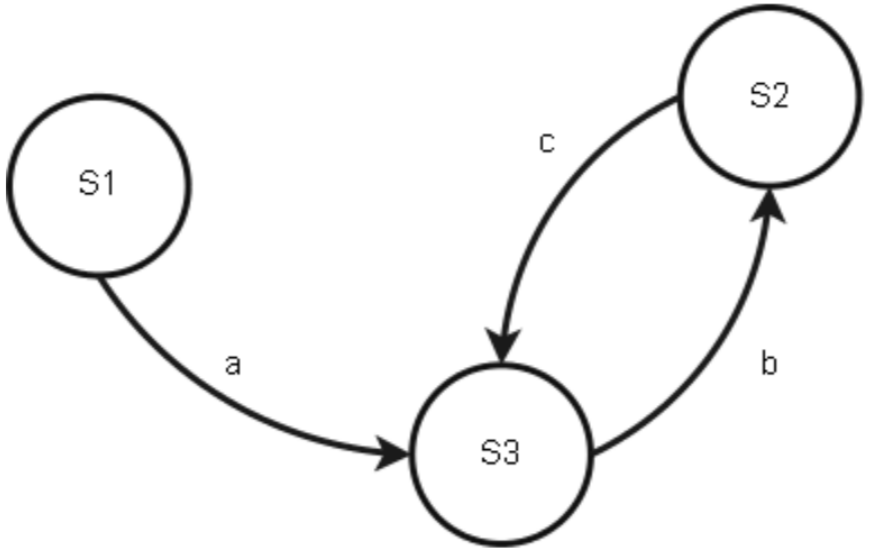
Конечный автомат

- FSM (finite state machine) , (finite automaton), конечный автомат, автомат — модель вычислений предполагающая наличие состояний объекта, переходов от состояния к состоянию и условий перехода.
- Конечный автомат - математическая модель устройства с конечной памятью. Конечный автомат перерабатывает множество входных дискретных сигналов в множество выходных сигналов. Различают синхронные и асинхронные конечные автоматы.
- Конечный автомат в теории алгоритмов - математическая абстракция, позволяющая описывать пути изменения состояния объекта в зависимости от его текущего состояния и входных данных, при условии что общее возможное количество состояний конечно. Конечный автомат является частным случаем абстрактного автомата.

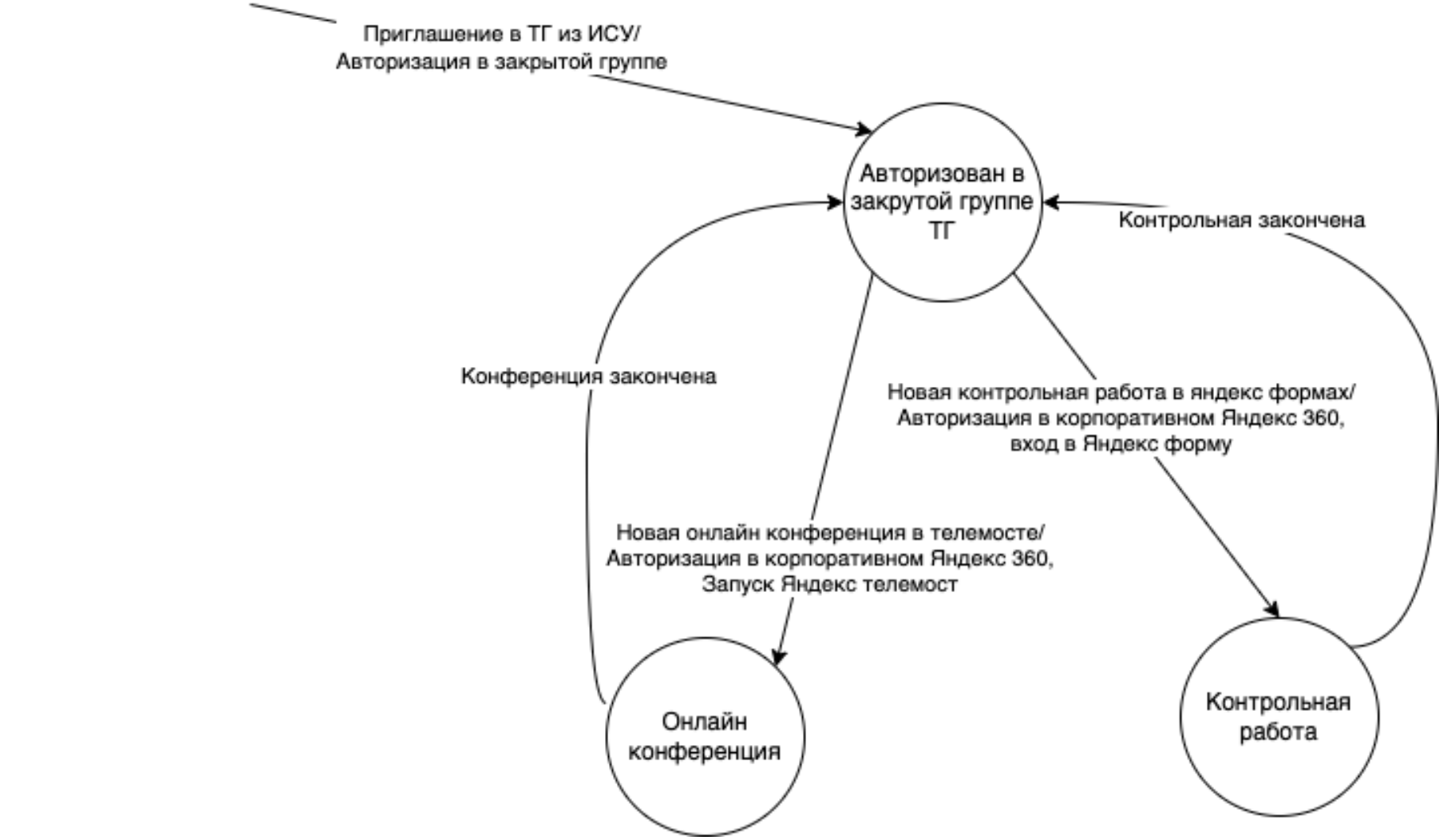
Конечный автомат

```
state = S1;

  while( true )
  {
     switch( state )
     {
        case S1: if( a ) state = S3;
        case S2: if( c ) state = S3;
        case S3: if( b ) state = S2;
     }
  }
```



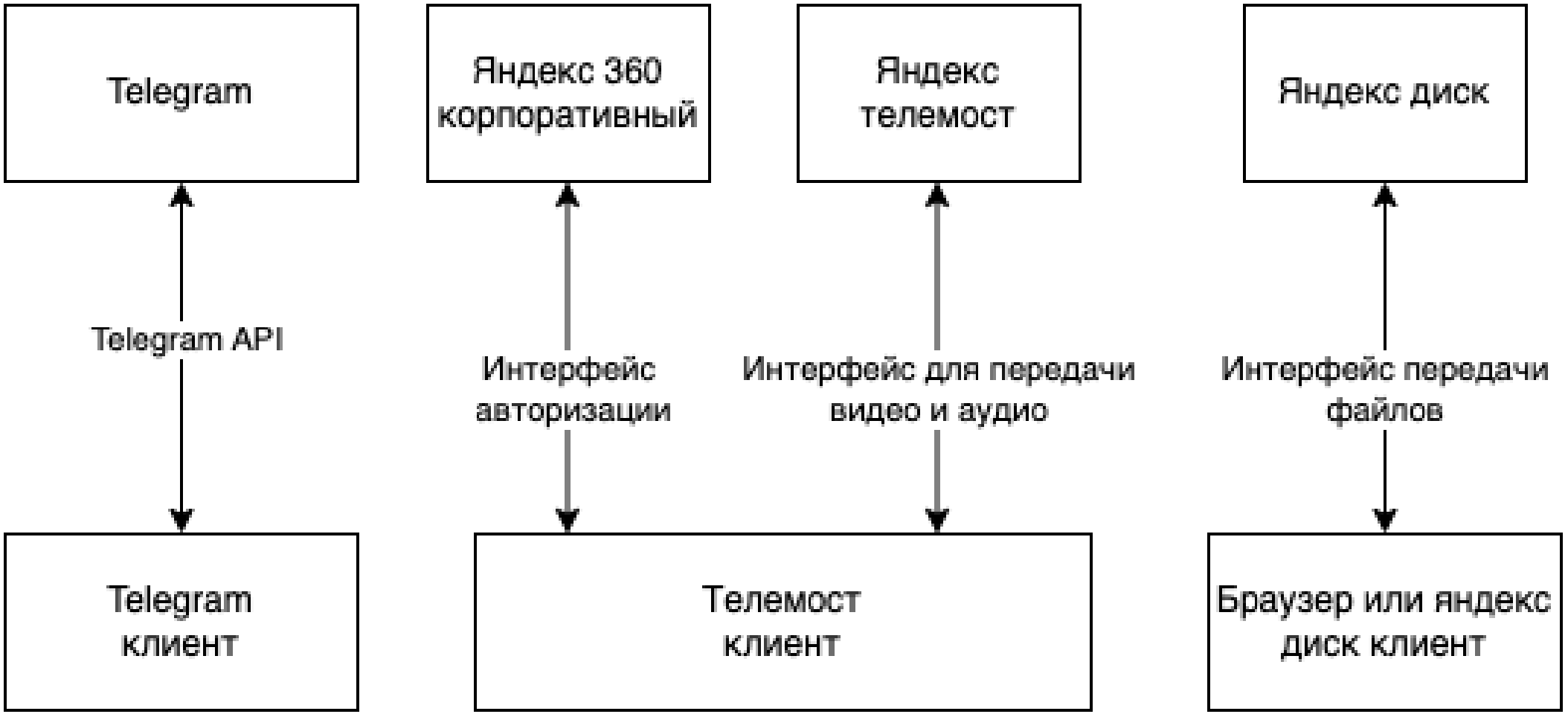
Пример конечного автомата



Структурная схема

- Очень часто нам нужно показать каким-то образом как соединяются различные части нашей системы.
- Нас не особо волнует что передается по проводам, но нас волнует какие провода нужны и какой стандарт передачи данных нам подходит.
- С помощью структурной схемы удобно демонстрировать то, из каких аппаратных блоков состоит наша система и какими интерфейсами эти блоки соединены.
- На самом деле структурная схема это такой же граф, как сеть процессов Кана или конечный автомат. Если не объяснить семантику элементов схемы (что такое квадрат или круг, а что означает стрелка), то схема не имеет какого либо смысла.
- Запомните, что нельзя в одну кучу сваливать элементы с разной семантикой, то есть нельзя нарисовать схему, в которой структурные компоненты аппаратного обеспечения смешаны вместе с состояниями и процессами.

Пример структурной схемы



Словарь вашей системы

- Словарь позволяет описать:
  - Аппаратные блоки
  - Интерфейсы
  - Процессы
  - Потоки данных
  - Состояния
  - Условия переходов и действия

Спасибо за внимание!