

# АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

СПбГУТ)))

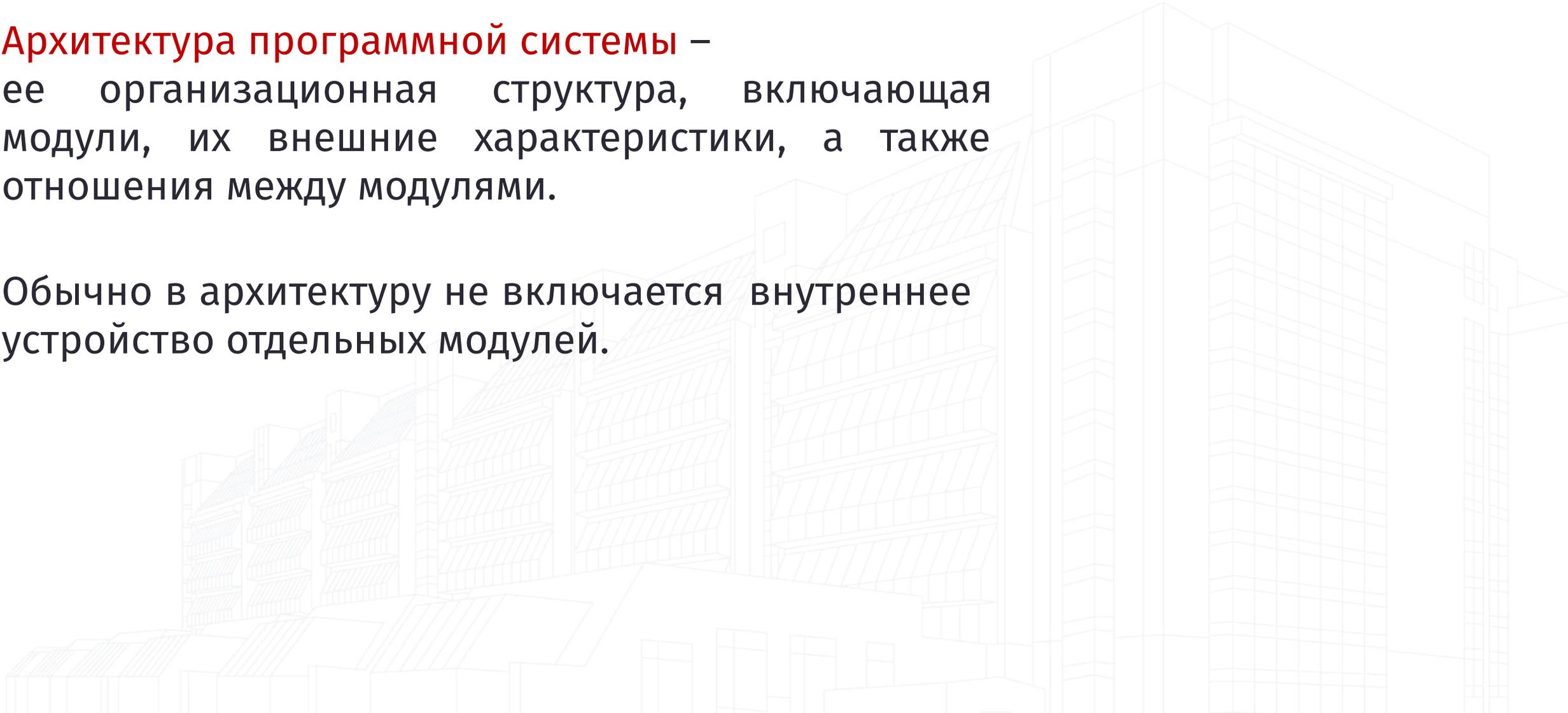




## Определение

**Архитектура программной системы** – ее организационная структура, включающая модули, их внешние характеристики, а также отношения между модулями.

Обычно в архитектуру не включается внутреннее устройство отдельных модулей.





## Разделение ответственности

**Разделение ответственностей (Separation of Concerns, SoC)** – процесс разделения (программной) системы на составные части, как можно меньше дублирующие функциональность друг другу.

Это основной принцип (программной) инженерии, сформулирован Э. Дейкстрой.

**Формулировка для запоминания:** Принцип DRY (Don't Repeat Yourself)

# Разделение абстракций



Абстракция + инкапсуляция + модульность = SoC

При правильном разделении ответственностей получившиеся составные части (модули) представляют собой абстракции, скрывающие внутреннее устройство и предоставляющие сравнительно простой внешний интерфейс

# Уровни абстракции



Нередко разделение абстракций представляет «слоеный пирог», тогда говорят об **уровнях абстракции**.

JVM
OS Kernel
Firmware
Hardware

Application
Presentation
Session
Transport
Network
Data link
Physical

# Виды ответственностей (concerns)



**Ответственности 1-го класса:** бизнес-логика приложения, следует непосредственно из функциональных требований

**Ответственности 2-го класса** (сквозная функциональность, cross-cutting concerns): функциональность, не относящаяся к бизнес-логике, проистекающая из нефункциональных требований



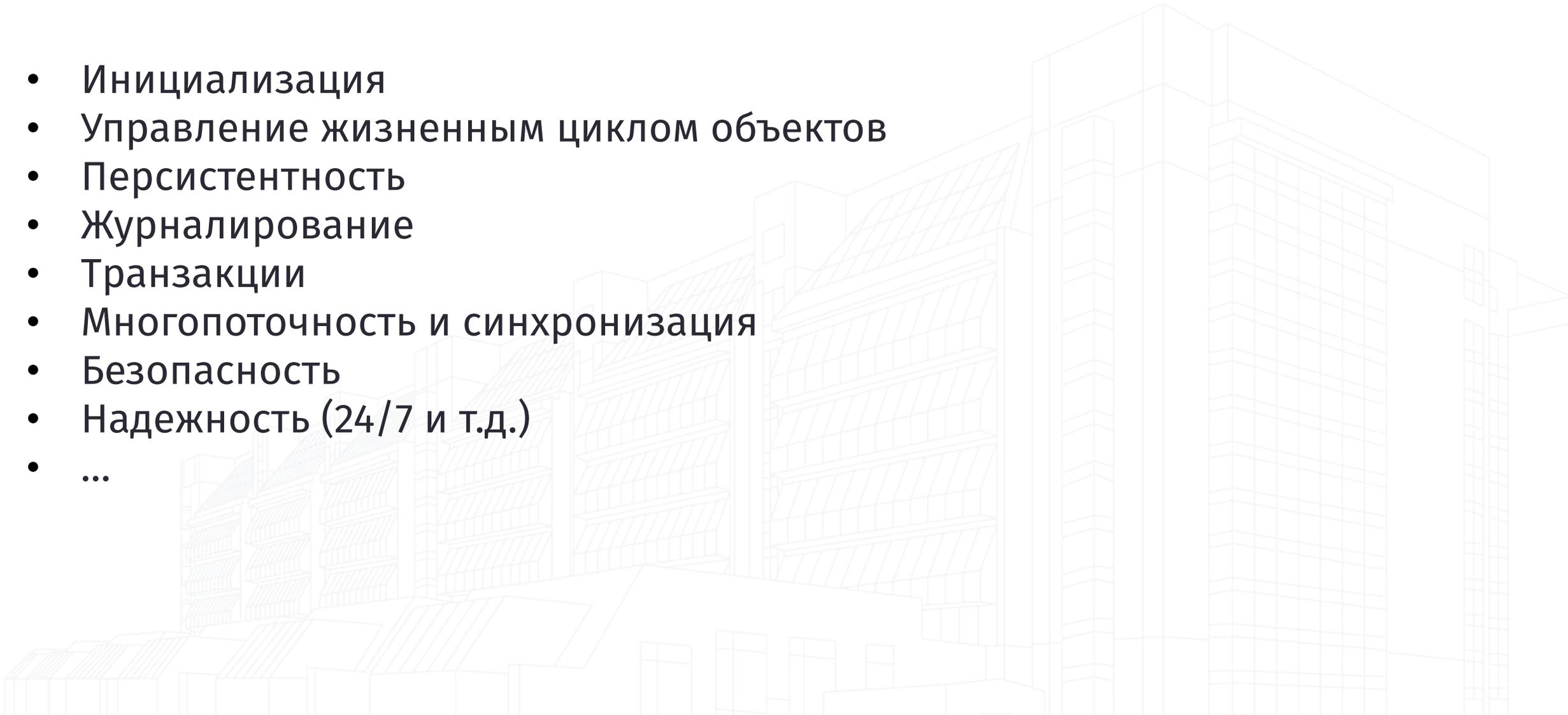
# Нефункциональные требования

- Платформа («железо», ОС, языки, библиотеки)
- Производительность (performance)
- Масштабируемость (scalability)
- Распределение функциональности по физическим узлам
- Простота поддержки, расширения, повторного использования модулей (maintainability, extensibility, reusability)

# Cross-cutting concerns



- Инициализация
- Управление жизненным циклом объектов
- Персистентность
- Журналирование
- Транзакции
- Многопоточность и синхронизация
- Безопасность
- Надежность (24/7 и т.д.)
- ...





# Представление архитектуры

Общие принципы и соглашения об организации системы:

- Парадигма
- Применение архитектурных шаблонов

Архитектурные представления (4+1 модель):

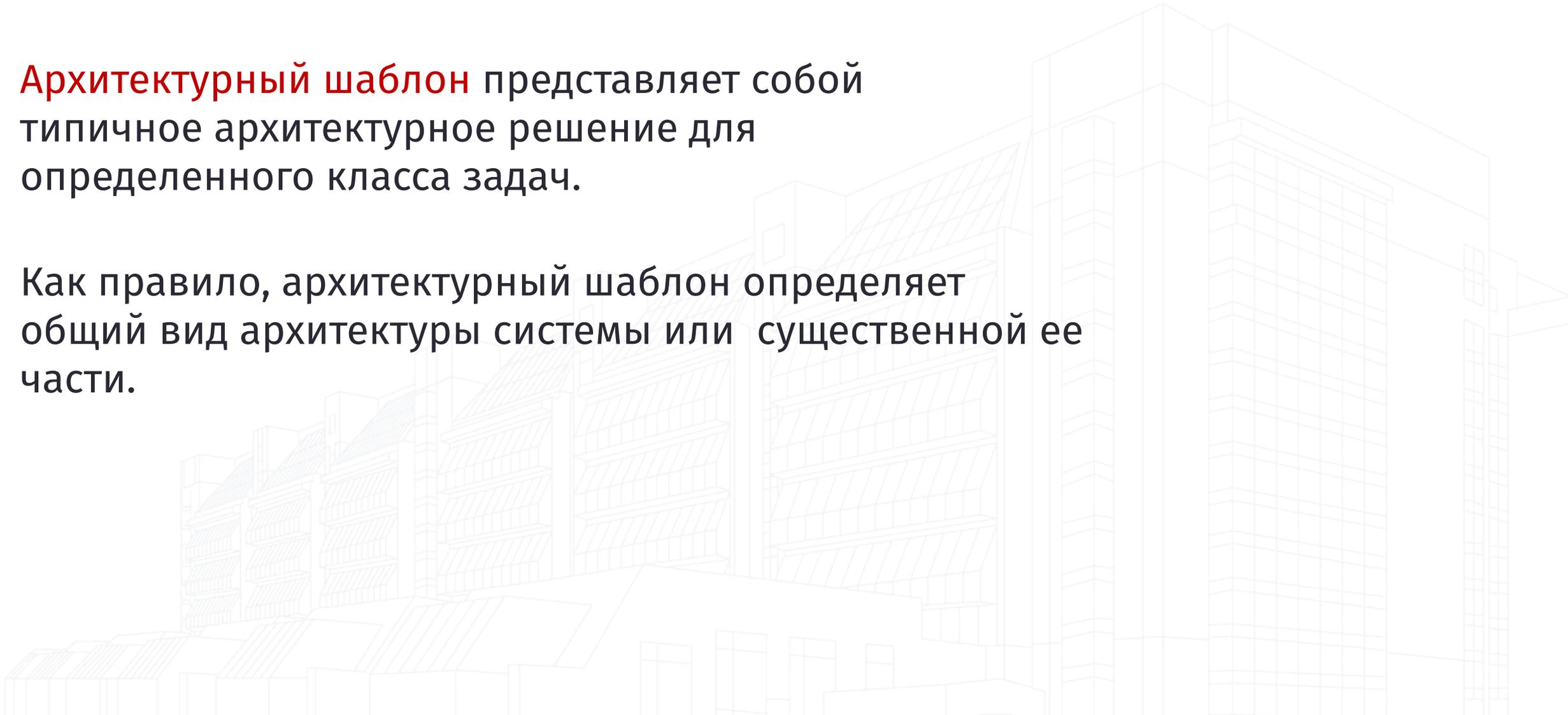
- Logical View (классы и пакеты)
- Process View (процессы и синхронизация)
- Physical View (компоненты и узлы)
- Development View (организация кода)
- Scenario View (варианты использования)

# Архитектурные шаблоны



**Архитектурный шаблон** представляет собой типичное архитектурное решение для определенного класса задач.

Как правило, архитектурный шаблон определяет общий вид архитектуры системы или существенной ее части.





# Клиент-сервер

**Задача:** обеспечить коммуникацию в распределенной среде между клиентами

## Решение:

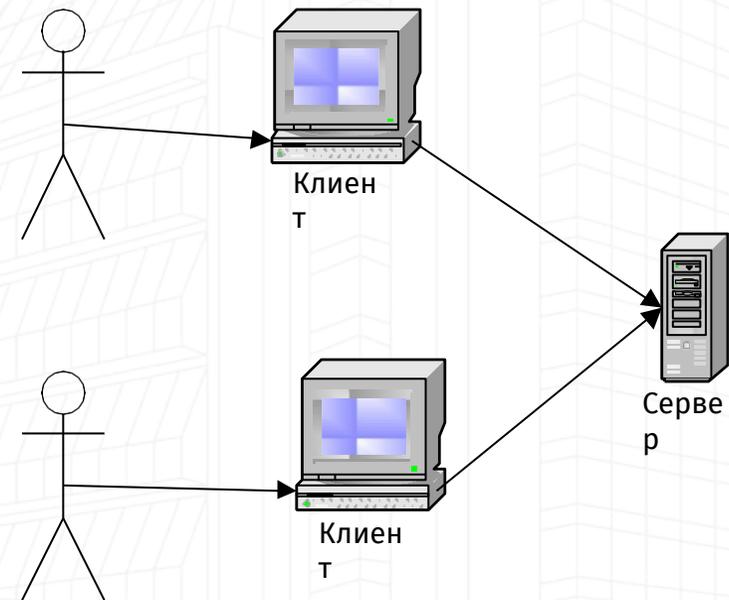
Клиенты взаимодействуют с единой сущностью – сервером. Между собой клиенты взаимодействовать не могут.

## Преимущества:

Сравнительная простота реализации и конфигурации

## Недостатки:

Сервер – потенциальное узкое место





# Одноранговая архитектура (P2P)

**Задача:** см. «клиент-сервер»

**Решение:**

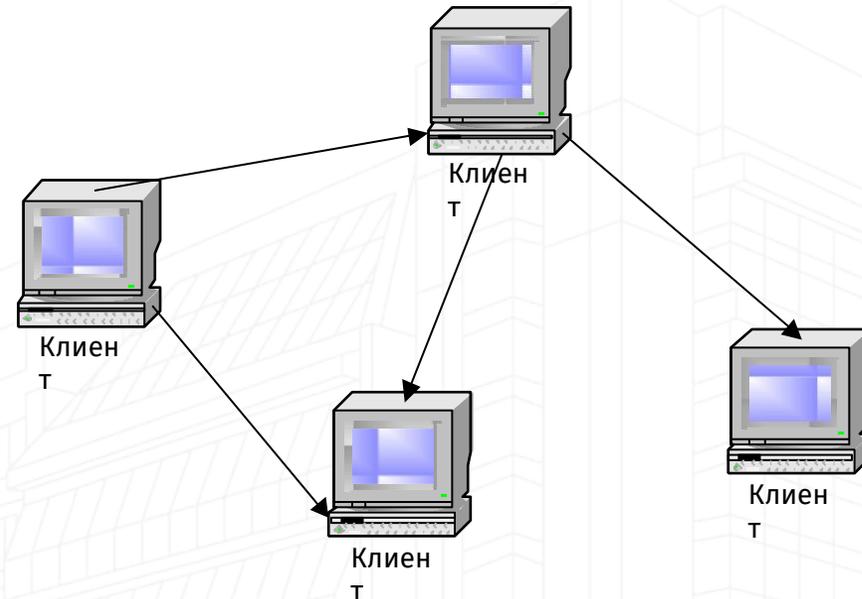
Клиенты непосредственно взаимодействуют друг с другом

**Преимущества:**

Нет явных узких мест

**Недостатки:**

- Сложность реализации и конфигурации
- Потенциальные проблемы видимости и маршрутизации



# Замечания по терминологии



В общем случае:

**Сервер** – сущность, предоставляющая функциональность

**Клиент** – сущность, использующая функциональность

Термины могут применяться безотносительно распределенных приложений

# Многоуровневая архитектура (N-tier Architecture)



Вариант архитектуры «клиент-сервер», где функциональность делится более чем на два уровня.

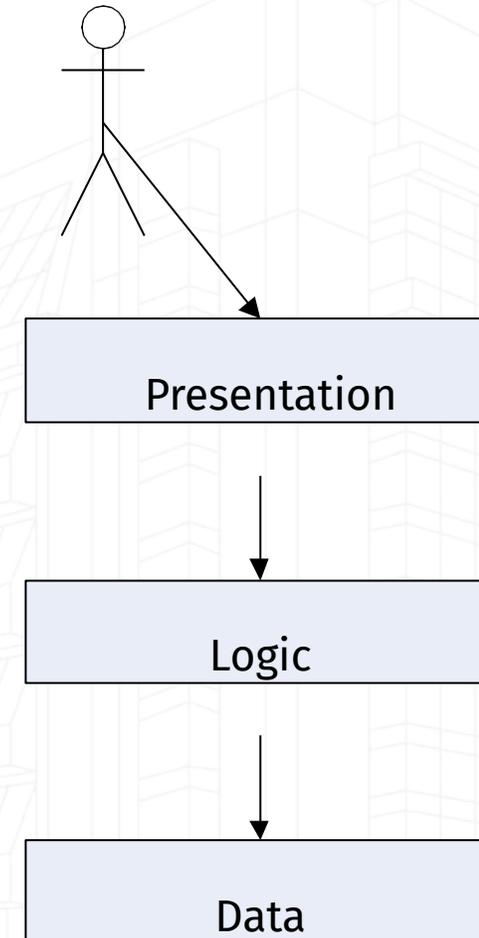
Каждый уровень является клиентом по отношению к нижележащему.

Распространенный вариант: 3-уровневая архитектура

## Преимущества:

- распределение нагрузки между уровнями
- хорошая масштабируемость

**Замечание:** для эффективного применения уровни (tiers) должны соотноситься с уровнями абстракции (layers)





## 3-уровневая архитектура

**Data Tier/Layer** – отвечает за представление данных и персистентность (соответствует набору entity в аналитической модели)

**Logic Tier/Layer** – реализует основную часть функциональности системы, отвечает также за целостность данных (~ controls)

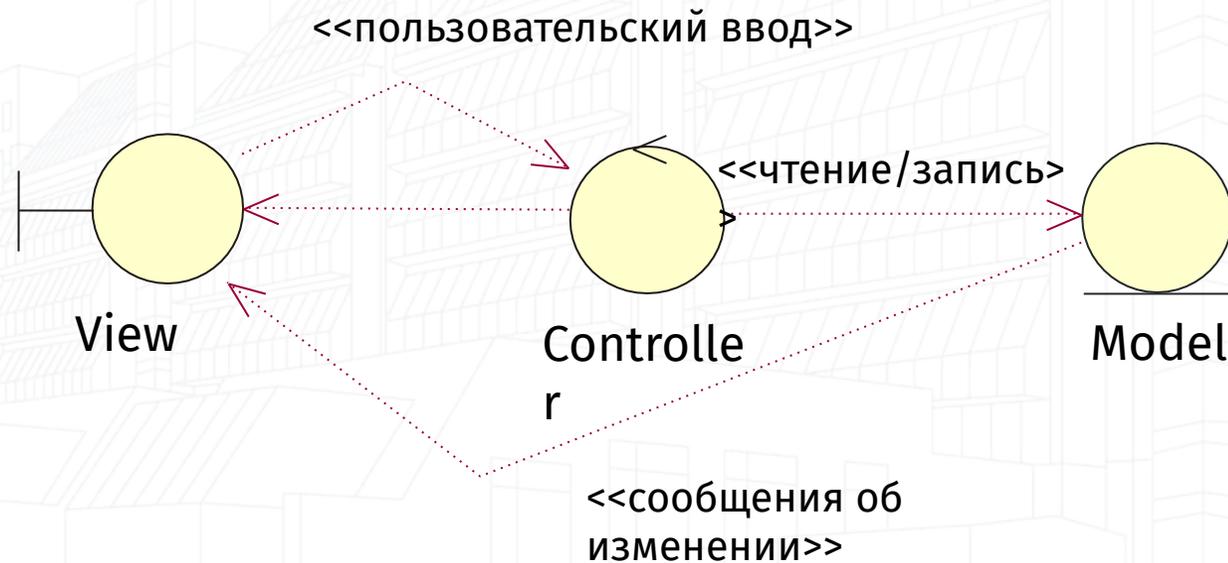
**Presentation Tier/Layer** – реализует интерфейс пользователя (~ boundaries)

# Модель-представление-управление (MVC (СПб ГУТ)))

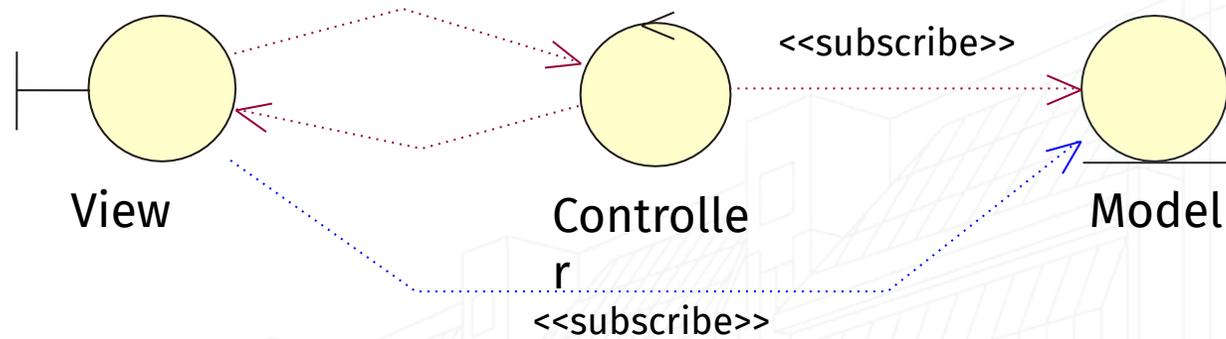


**Задача:** разделение бизнес-логики и интерфейса пользователя в соответствии с SoC

Как правило, речи не идет о распределенной системе

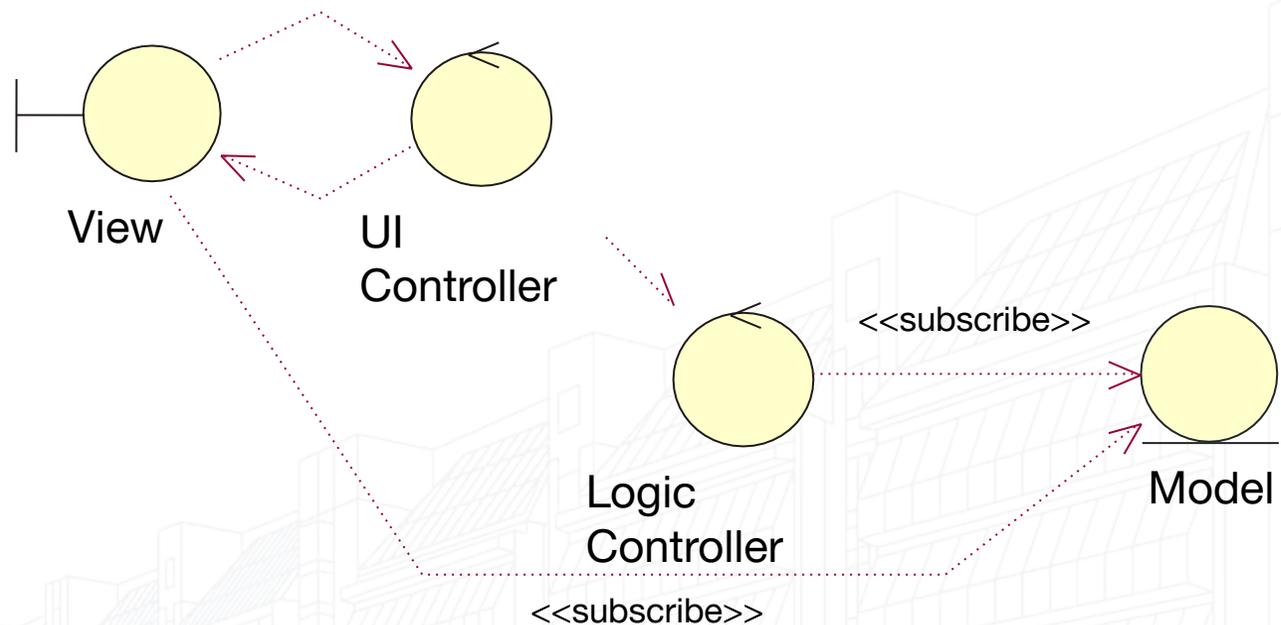


# Переход от MVC к 3-tier



**Шаг 1:** применение стереотипа  
subscribe

# Переход от MVC к 3-tier



**Шаг 2:** расщепление контроллера

# Событийно-ориентированная архитектура



## Задача:

Обеспечить взаимодействие между модулями без жесткой их привязки друг к другу

## Решение:

Модули взаимодействуют в терминах отправки сообщений и реакции на них. Как правило, сообщения асинхронны.

## Недостатки:

шина сообщений может быть узким местом

## Применение:

- Интерфейс пользователя
- Взаимодействие приложений
- Распределенные вычисления (MPI)

