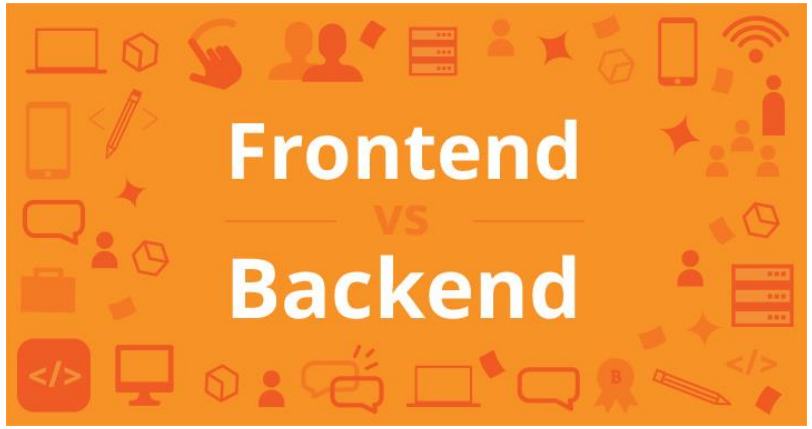
Типовые архитектуры GUI-приложений



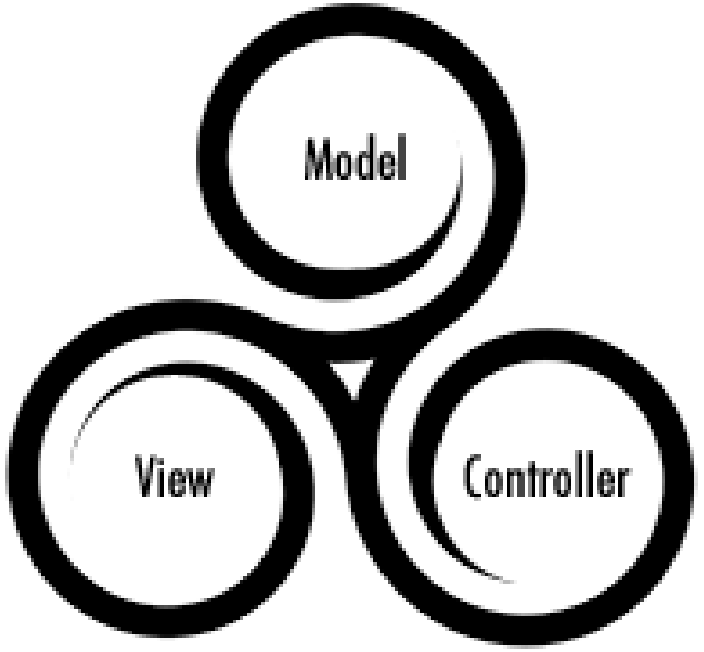
Что такое паттерн?

паттерн — повторимая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Паттерны проектирования

Семейство «MVC-паттернов» :

- MVC
- MTV
- MPV
- MVVM
- MVPVM
- …….

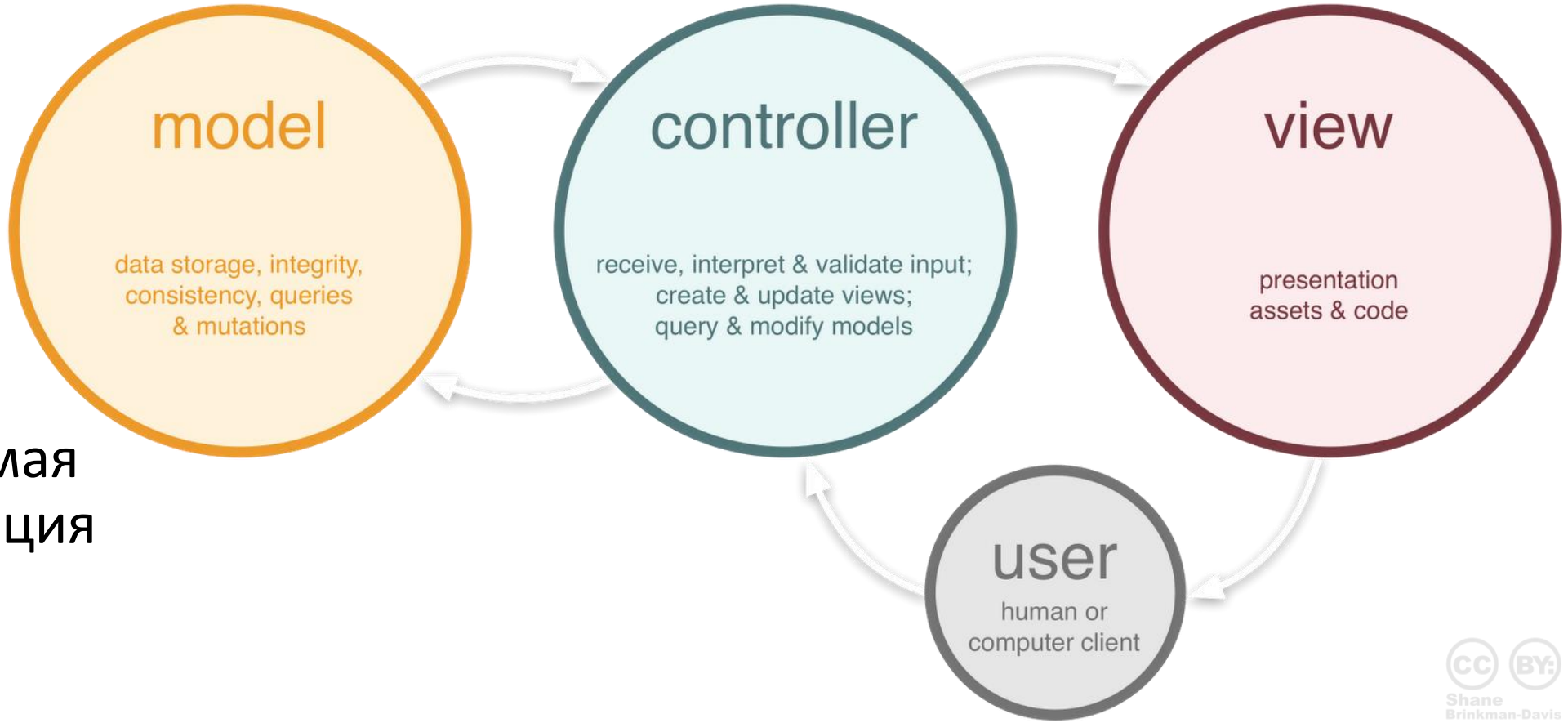


MVC

Model
View
Controller

Трюгве Реенскауг в 1979 году, язык Smalltalk

MVC: Model-View-Controller



- Независимая модификация

MVC: Model-View-Controller

View — интерфейс.

Controller — обработчик событий, инициируемых пользователем (нажатие на кнопку, переход по ссылке, отправка формы).

Model — метод, который запускается обработчиком и выполняет все основные операции (получение записей из базы данных, проведение вычислений).

Модель может быть пассивной и активной (выше – активная).

MVC: Model-View-Controller

Для чего нужен MVC?

MVC: Model-View-Controller

Для независимости изменений

MVC: Model

- Модель — это бизнес-логика приложения
- Модель обладает знаниями о себе самой и не знает о контроллерах и представлениях - Почему?

MVC: Model-View-Controller

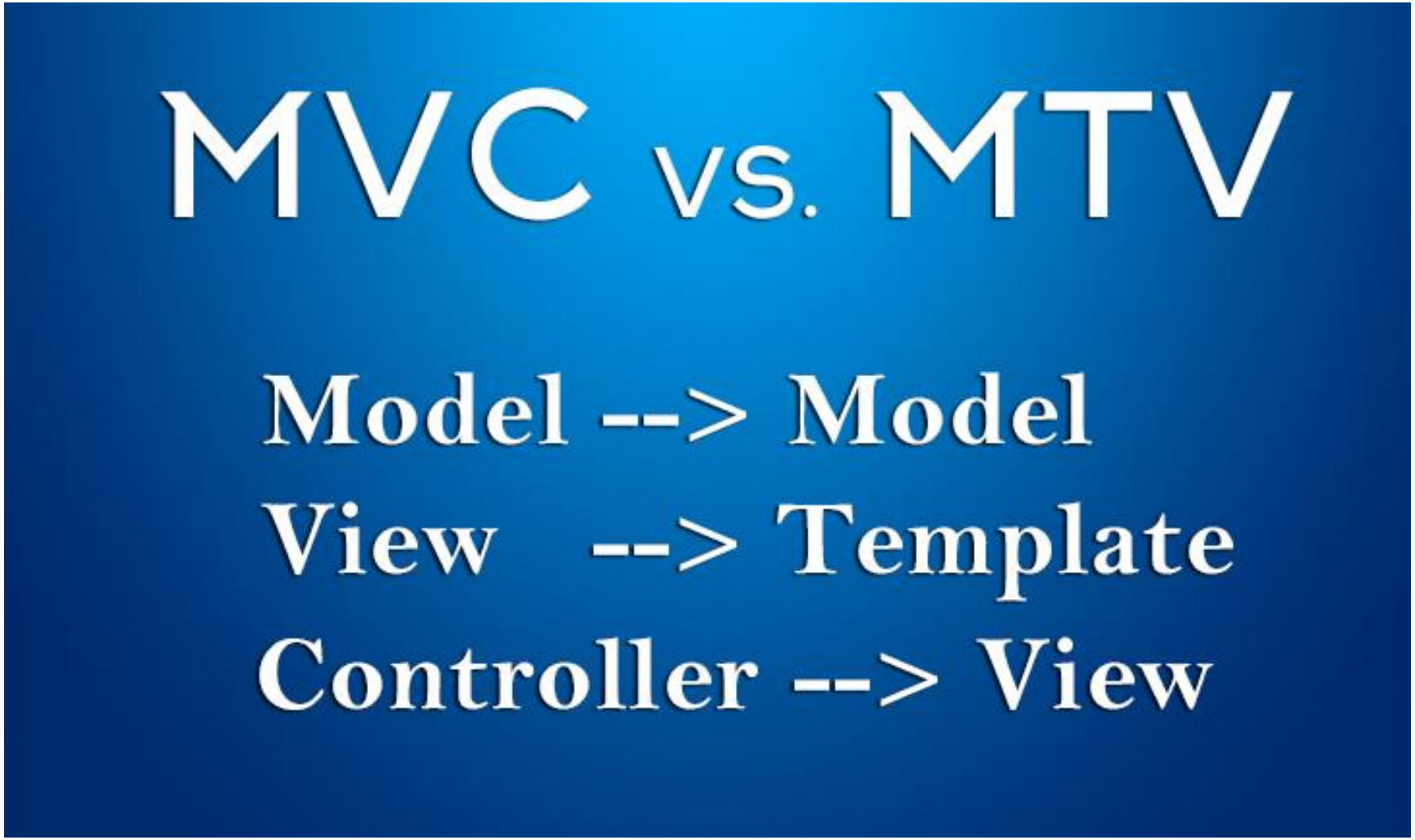
Признаки контроллера:

- Контроллер определяет, какое представление должно быть отображено в данный момент;
- События представления могут повлиять только на контроллер.
- контроллер может повлиять на модель и определить другое представление.
- Возможно несколько представлений только для одного контроллера;

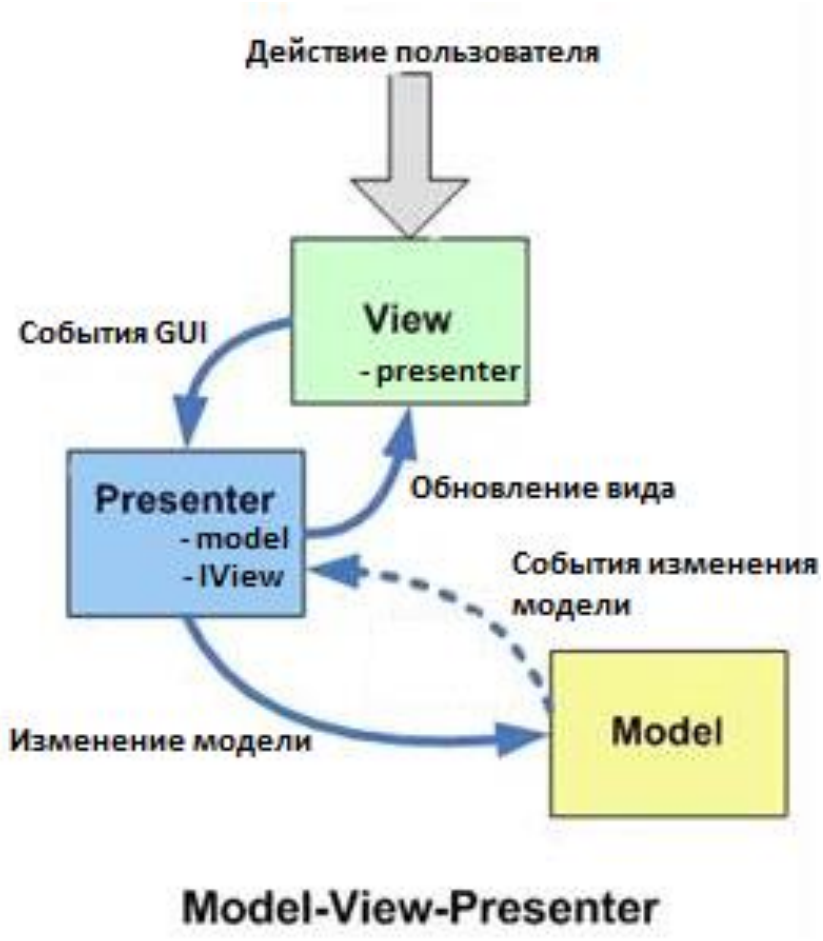
Пример использования – MVC ASP.NET

FSUC: Fat Stupid Ugly Controllers

MTV: Model-Template-View (Django-взгляд на MVC)



MVP: Model-View-Presenter



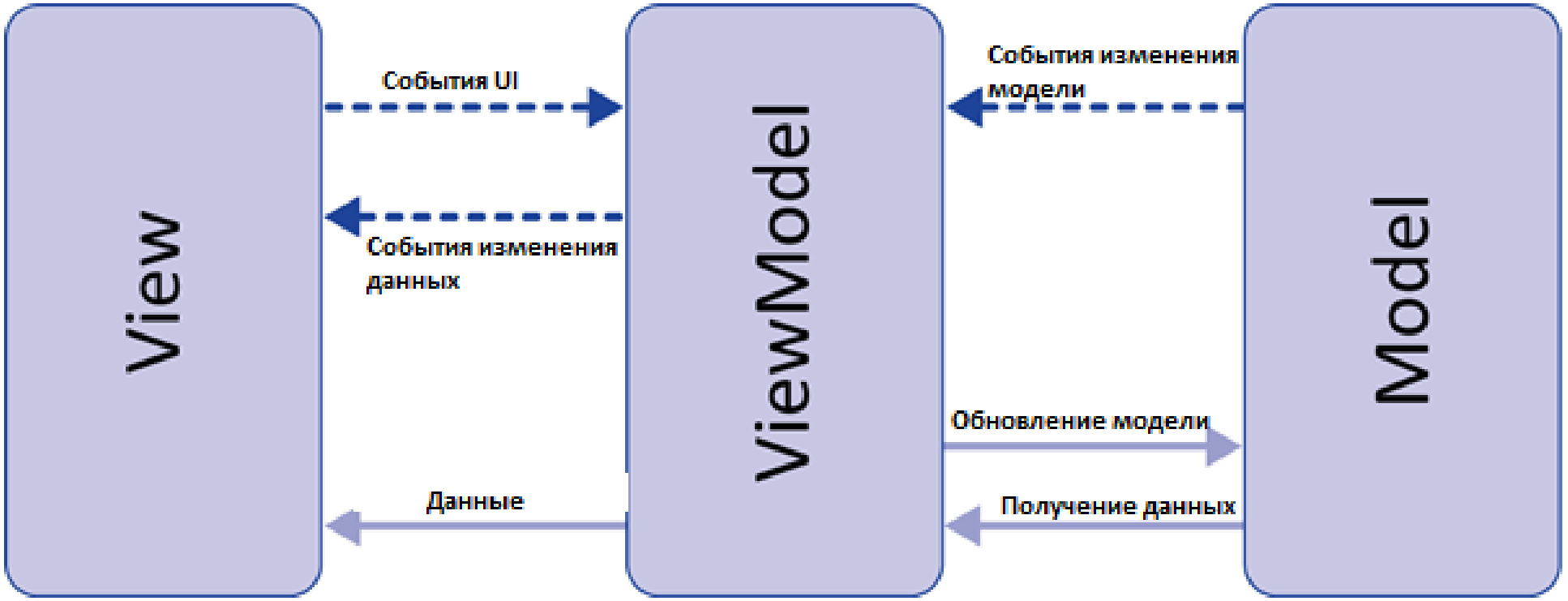
MVP: Model-View-Presenter

Признаки Presenter:

- Двухсторонняя коммуникация с View;
- View взаимодействует напрямую с Presenter, путем вызова соответствующих функций или событий экземпляра Presenter;
- Presenter взаимодействует с View путем использования специального интерфейса, реализованного View;
- Один экземпляр Presenter связан с одним View.

Пример использования – Win forms

MVVM: Model-View-View Model



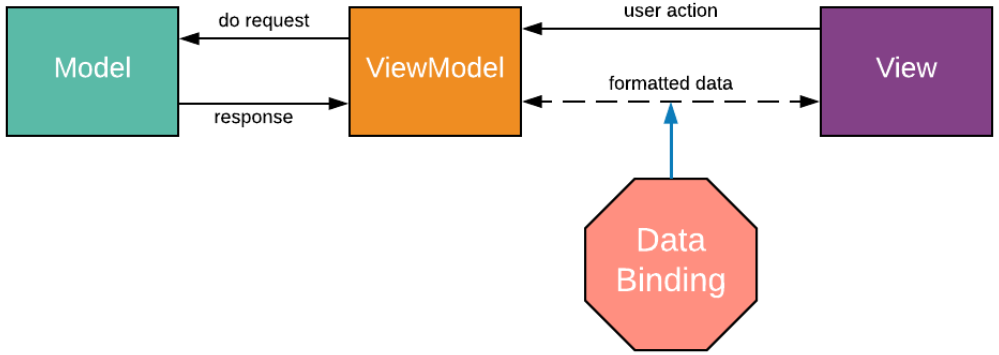
MVVM: Model-View-View Model

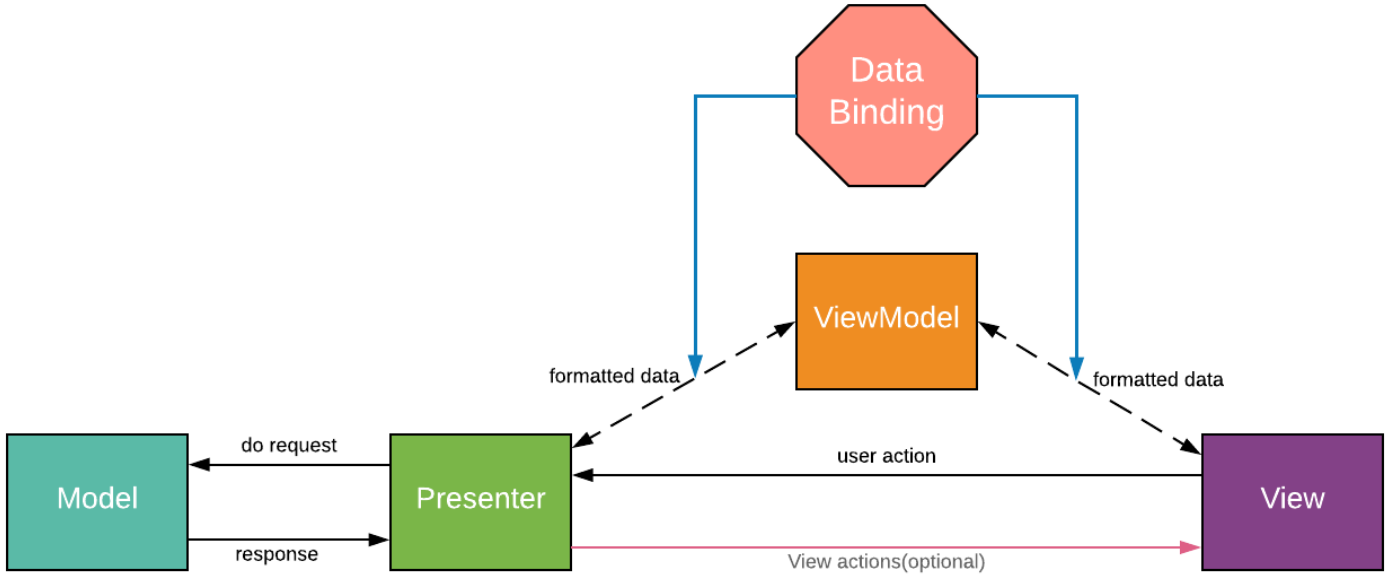
Признаки View-модели:

- Двухсторонняя коммуникация с представлением;
- View-модель — это абстракция View. Обычно означает, что свойства View совпадают со свойствами View-модели / модели
- View-модель не имеет ссылки на интерфейс представления (IView). Изменение состояния View-модели автоматически изменяет представление и наоборот, поскольку используется механизм связывания данных (Bindings).
- Один экземпляр View-модели связан с одним View.

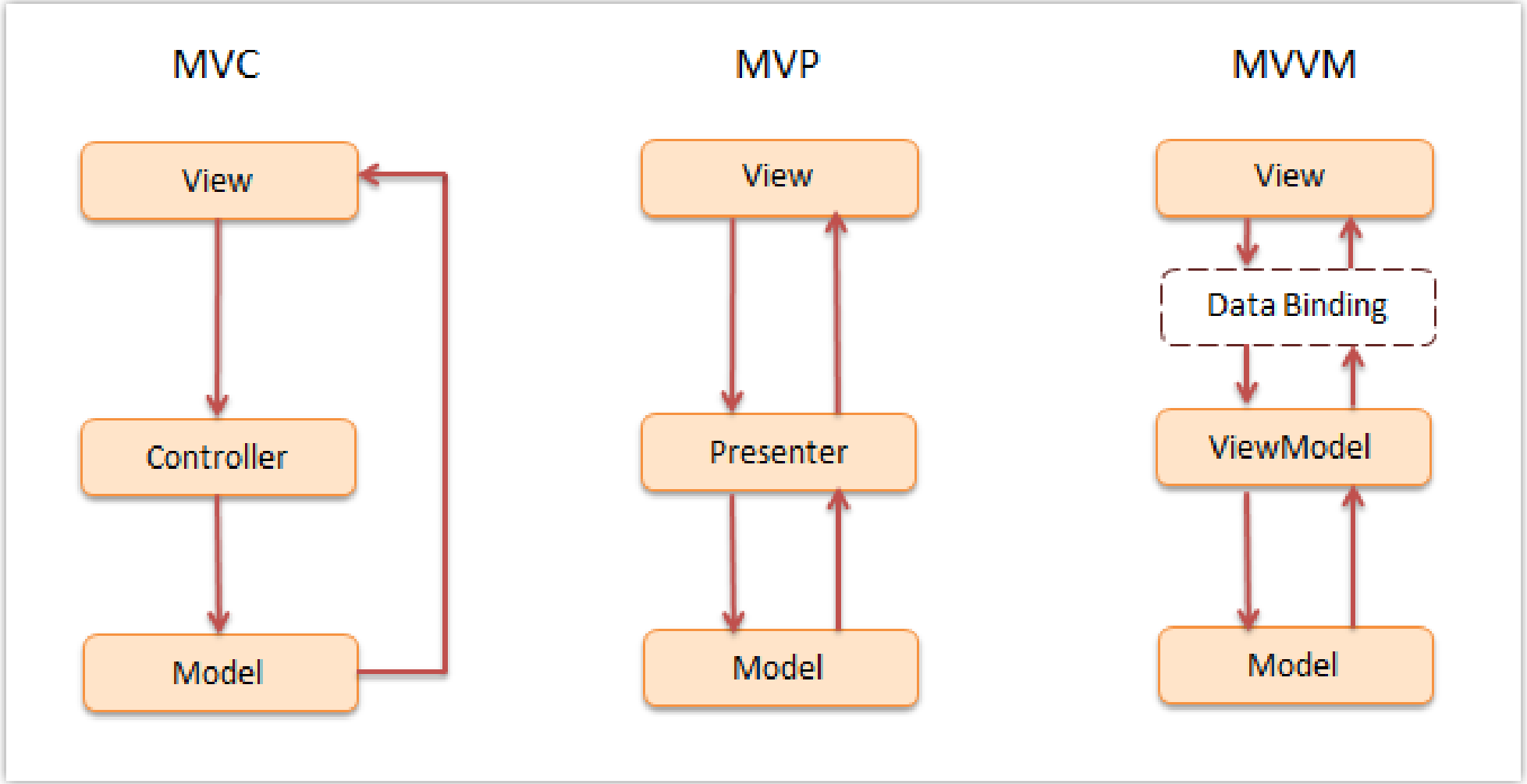
Пример использования – WPF

MVP vs MVVM vs MVPVM



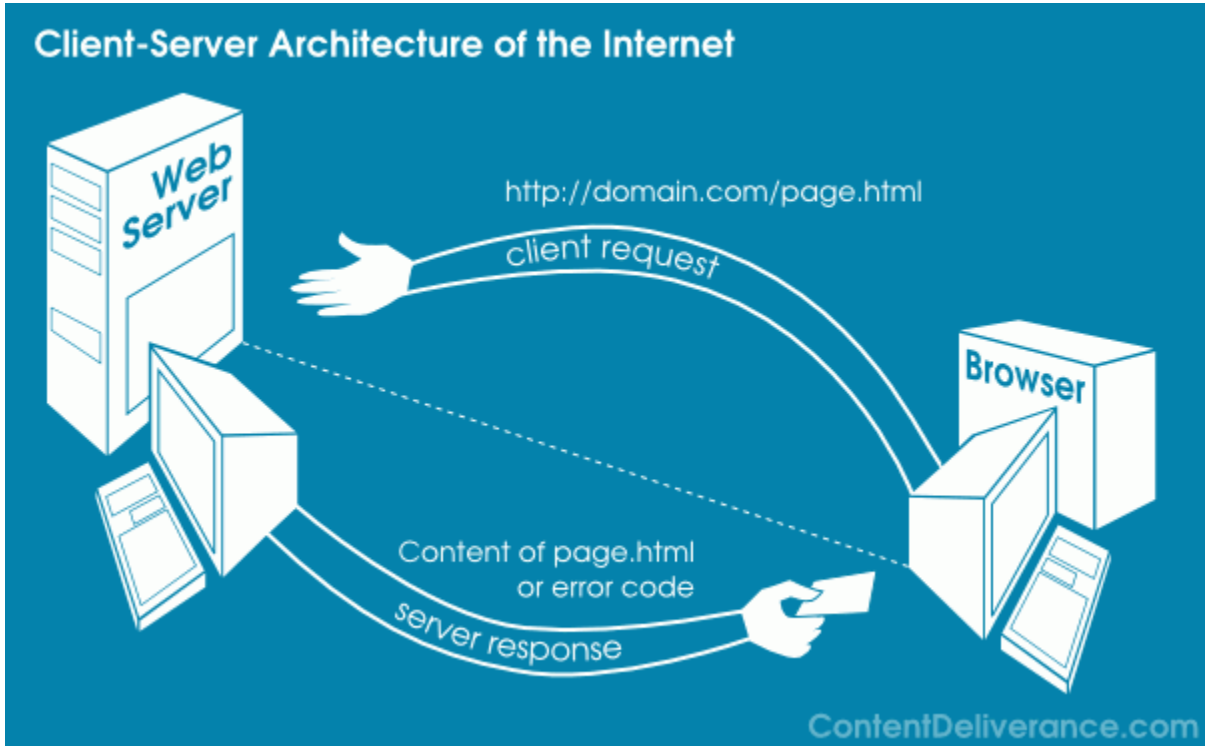


MVC vs MVP vs MVVM

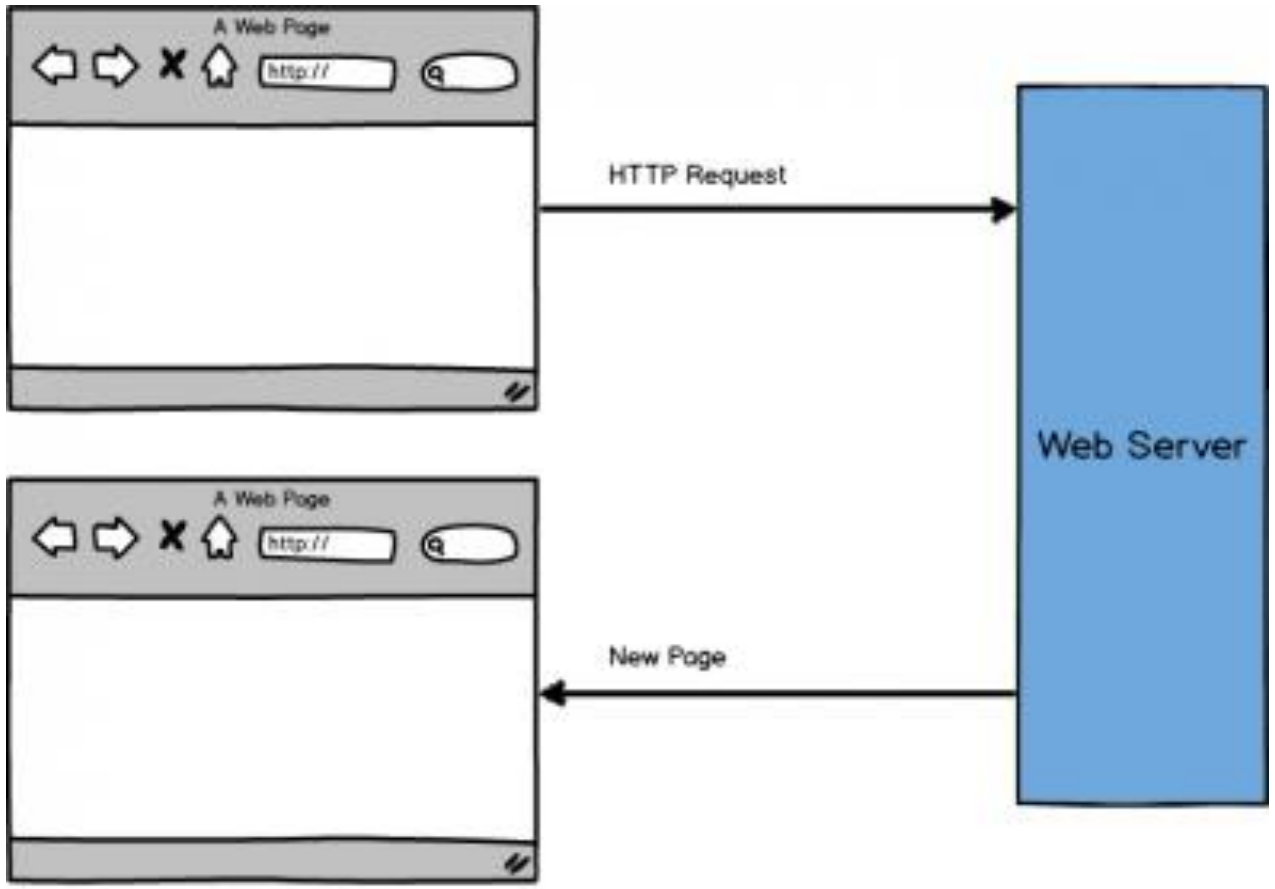


Классификации клиент-серверных архитектур в Вебе

- MPA-SPA
- Толстый-Тонкий
- Изоморфный

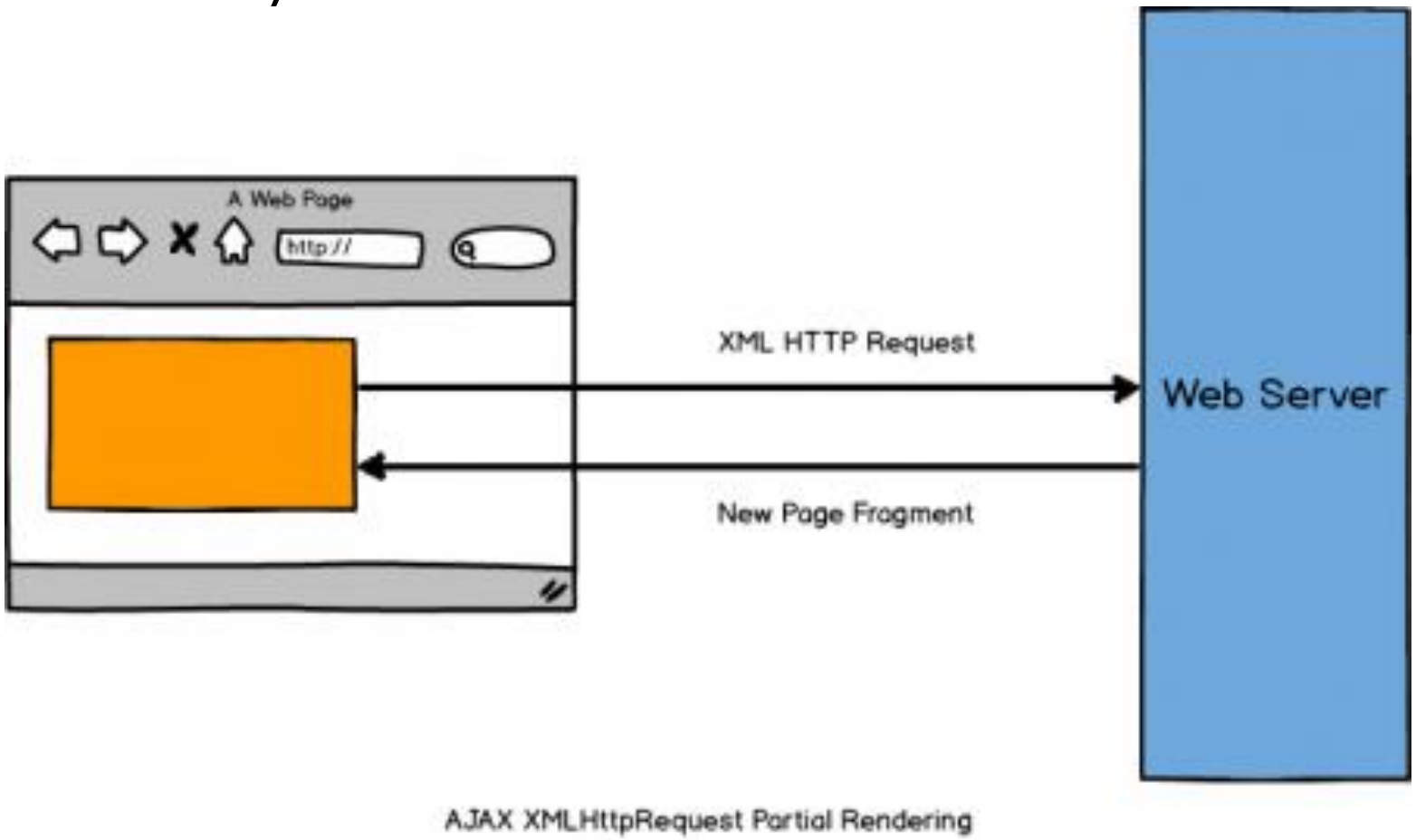


Многостраничное приложение (MPA, Multi Page Application)



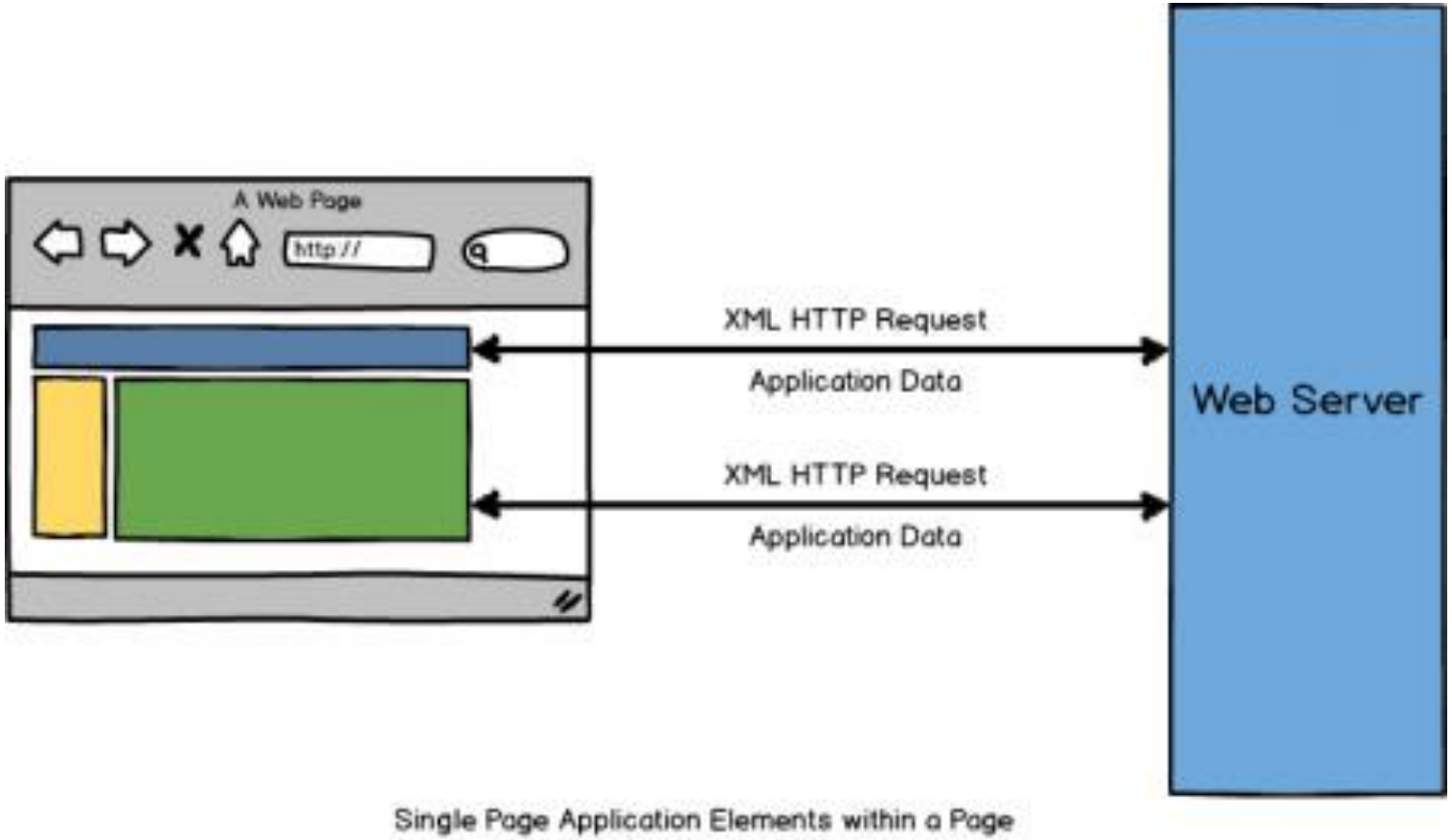
Traditional Full-Page Postback Operation

Многостраничное приложение с асинхронной загрузкой данных и частичным обновлением (MPA + AJAX)



AJAX XMLHttpRequest Partial Rendering

Одностраничное приложение (SPA, Single Page Application)



Single Page Application Elements within a Page

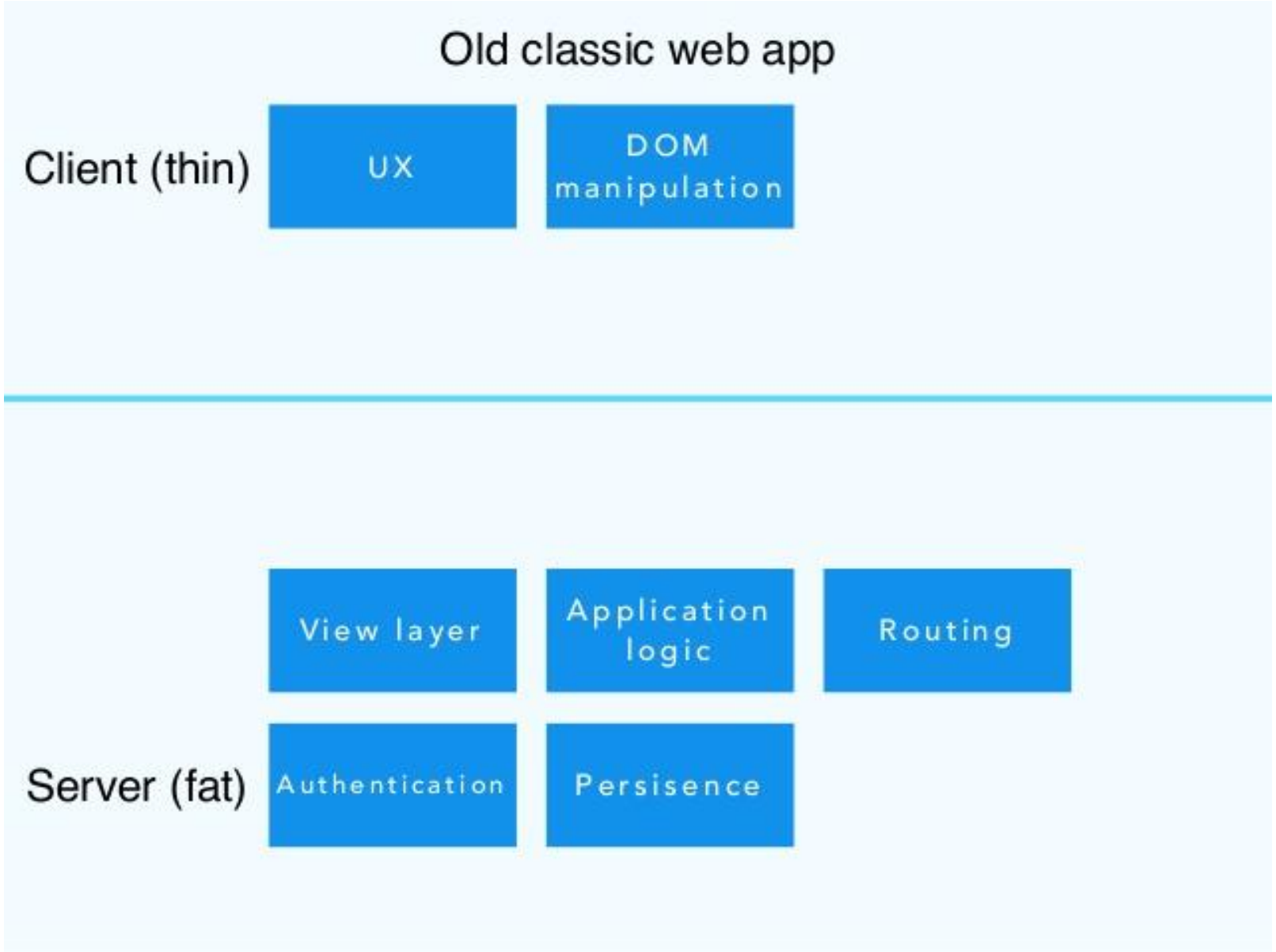
SPA

Достоинства	Недостатки
Быстрая загрузка/обновление страниц	Тяжелые фреймворки
Удобство пользователя	Интерпретируемые языки
Изолирование фронтенда и бэкенда	SEO

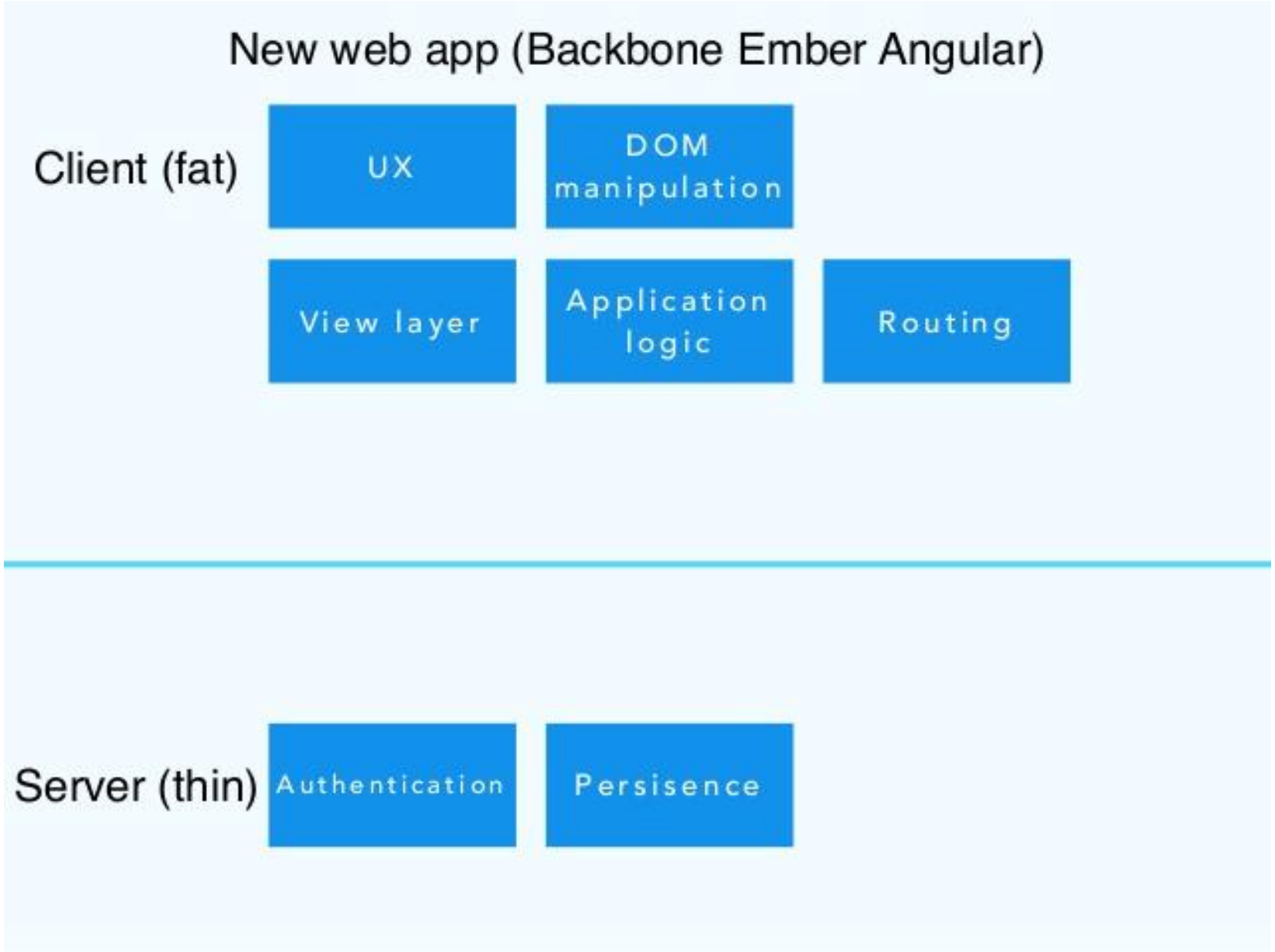
Толстый и тонкий

клиент

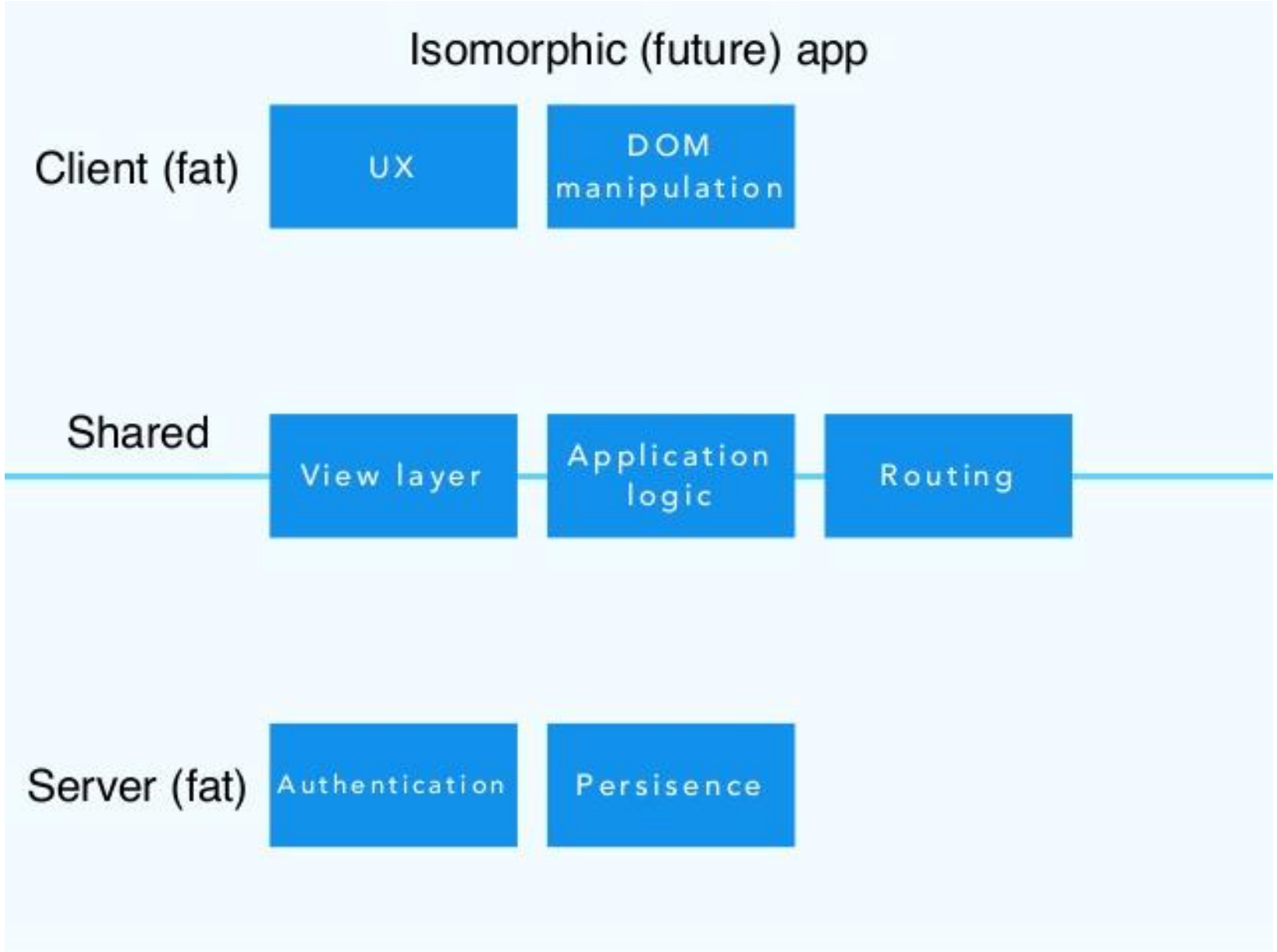
Тонкий клиент (Классический MPA)



Толстый клиент (RIA: Rich Internet Application)



Изоморфное приложение



Изоморфное приложение

- Единый код клиент-сервер
- Объединение достоинств MPA и SPA:
  - Скорость загрузки MPA
  - SEO-возможности MPA
  - Гибкость и удобство SPA
  - Тестируемость
  - Поддержка

Выбор архитектуры: от задач

Там где возможно – MPA

- Скорость разработки
- Скорость работы
- Нет проблем с SEO
- Дешево

Сайты-визитки, персональные страницы, сайты компаний

Выбор архитектуры: от задач

Там где невозможно MPA - SPA

- Гибкая архитектура
- Максимальное удобство пользователя
- Простота поддержки и модификации

Любое веб-приложение, веб-интерфейс сервисов и т.д.

Выбор архитектуры: от задач

Там где невозможно SPA – изоморфное

- Одни достоинства
- Слишком модно и современно
- Дорого
- Необходимо любить JS

Паттерн: Repository

Абстракция от источника/хранилища данных.

Упрощение тестирования за счет создания различных тестовых реализаций.

Выделение общей логики (DRY) – логгирование, проверки и т.д.

Репозиторий – пример

```
public interface IPostsRepository
{
    void Save(Post mypost);
    Post Get(int id);
    PaginatedResult<Post> List(int skip, int pageSize);
    PaginatedResult<Post> SearchByTitle(string title, int skip, int pageSize);
}
```

Generic Repository

```
public interface IRepository<T>
{
   IEnumerable<T> GetAll();
   T GetById(int id);
   void Add(T entity);
   void Update(T entity);
   void Delete(T entity);
}
```