# Введение

**Django** — синхронный framework: 1 запущенный процесс django будет обрабатывать одного клиента.

Это неплохо, если мы выполняем CPU bound операции. Мы занимаем вычислительную мощность процессора и под эту операцию выделяем отдельный процесс. Но это неэффективно, если выполнять блокирующие IO bound операции. Процесс/поток будет заблокирован на все время ожидания операции.

Чтобы обслужить больше одного клиента одновременно, нужно запустить дополнительные процессы. Получается, что максимальное количество одновременно обслуживаемых клиентов будет равно количеству одновременно запущенных процессов. При этом процессы в большинстве случаев будут ожидать, когда выполнится IO-операция и ничего не делать, а процессор будет тратить ресурсы на переключение между «спящими» процессами.

Большинство монолитных приложений на Django работают только с БД. Зачастую база данных находится на том же сервере, и все запросы к ней выполняются быстро. Но если увеличивается время выполнения запросов к БД, или добавляется логика обращения к другим микросервисам, то производительность и эффективность утилизации ресурсов драматически падает.

В ситуации, когда приложение в основном ожидает выполнения IO bound операций (выполнения запроса в базу или ответа другого микросервиса), подойдет асинхронное программирование и асинхронные фреймворки — например, aiohttp.

Давайте посмотрим, в каких случаях асинхронный подход приносит пользу.

**Микросервисная архитектура.** Используется микросервисный подход. Для формирования ответа пользователю нужно выполнить много обращений к другим микросервисам.

**Задачи доставки информации в реальном времени.** Например, уведомления или чаты.

Для реализации таких задач используется технология web socket. Клиент открывает соединение и ждет, когда сервер пришлет новое сообщение или уведомление. В случае с синхронным подходом каждому клиенту нужно выделять отдельный поток/процесс, даже если он не получает сообщений. При таком подходе один сервер может обрабатывать тысячи соединений, а при асинхронном подходе десятки или сотни тысяч.

**Стриминговая загрузка.** Например, в s3-хранилище.

Обычный синхронный подход для загрузки файлов — получить весь файл, затем отправить его дальше в s3-хранилище. Такой подход удовлетворительно работает, когда размеры файлов сравнительно небольшие: 10-100 Мб. Но если файл весит гигабайты, использовать веб-сервер как буфер будет плохой идеей.

Решение проблемы загрузки больших файлов — сразу отправлять полученные данные дальше в пункт назначения, то есть «стримить». Aiohttp позволяет эффективно решать такую задачу и параллельно обрабатывать других клиентов.