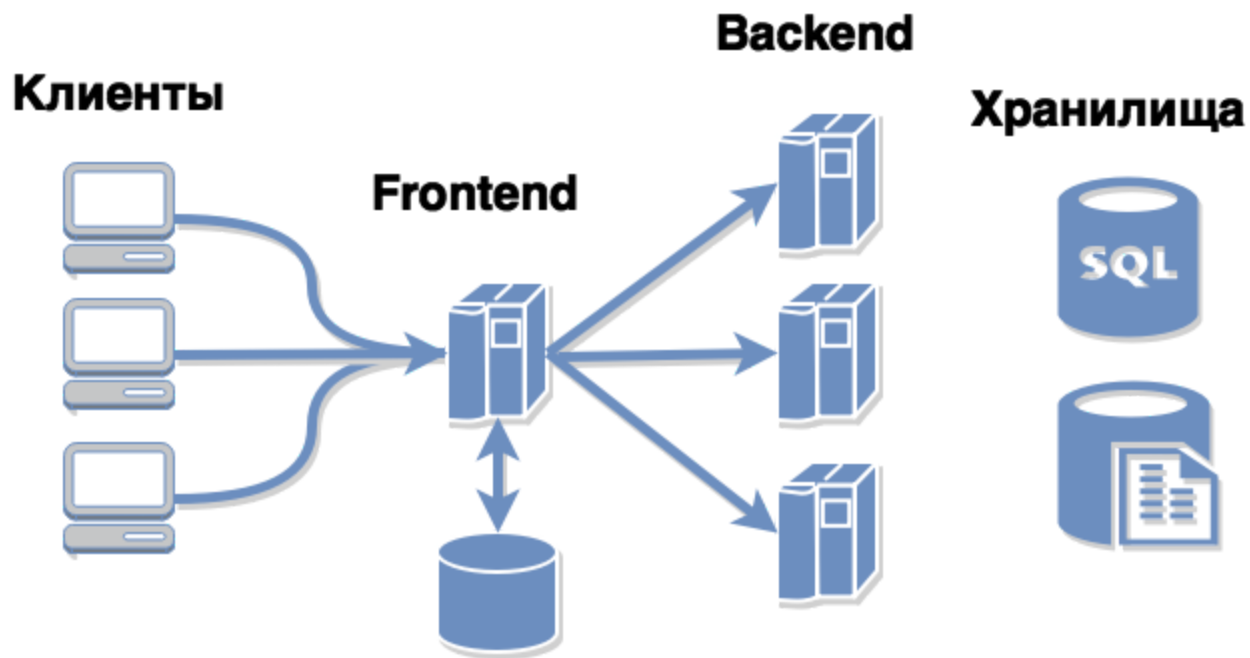


Frontend и  
Backend

# Общая архитектура



# Задачи Frontend (web) сервера

- отдача статических документов
- проксирование (reverse proxy)
- балансировка нагрузки
- кеширование
- сборка SSI
- авторизация, SSL, нарезка картинок, gzip

# Reverse proxy

- frontend (медленно) читает запрос от клиента
- frontend (быстро) передает запрос свободному backend
- backend генерирует страницу
- backend (быстро) возвращает ответ frontend серверу
- frontend (медленно) возвращает ответ клиенту

Результат: backend занят минимально возможное время.

# Настройка проксирования в nginx

# Настройка проксирования в nginx

```
proxy_set_header Host      $host;  
proxy_set_header X-Real-IP $remote_addr;  
location / {  
    proxy_pass http://backend;  
}  
location /partner/ {  
    proxy_pass http://www.partner.com;  
}  
location ~ /\.w\.w\.w?\w?$ {  
    root /www/static;  
}
```

# Настройка upstream в nginx

```
upstream backend {  
    server back1.example.com:8080 weight=1 max_fails=3;  
    server back2.example.com:8080 weight=2;  
    server unix:/tmp/backend.sock;  
    server backup1.example.com:8080 backup;  
    server backup2.example.com:8080 backup;  
}
```

Application  
сервер



# Backend (application) сервер

Роль application сервера заключается в исполнении бизнес логики приложения и генерации динамических документов.

На каждый HTTP запрос application сервер запускает некоторый обработчик в приложении. Это может быть функция, класс или программа, в зависимости от технологии.

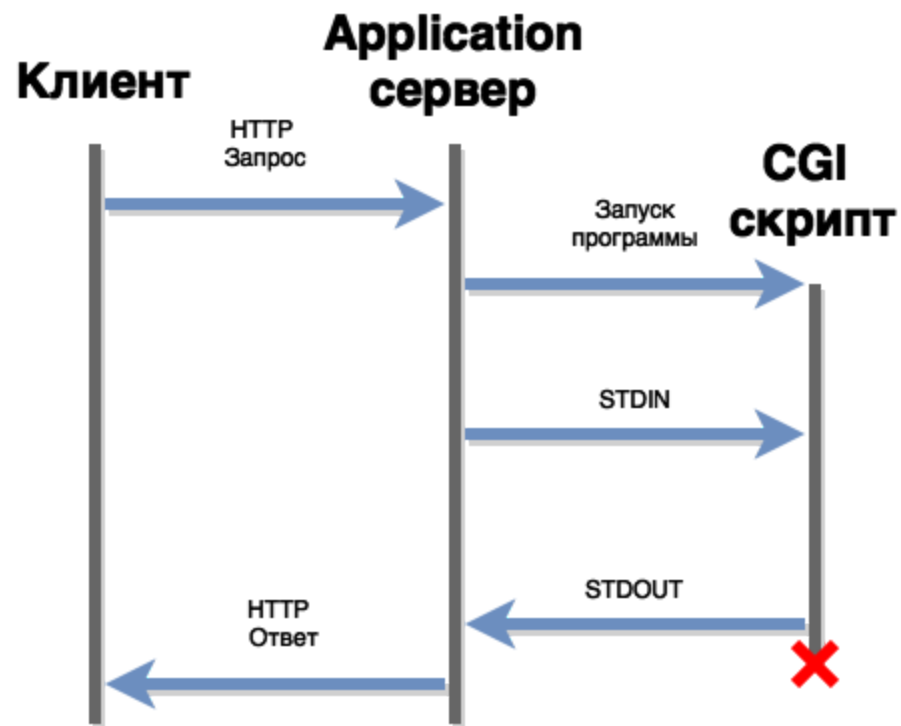
# Протоколы запуска приложения

- Servlets и др. специализированное API
- mod\_perl, mod\_python, mod\_php
- CGI
- FastCGI
- SCGI
- PSGI, **WSGI**, Rack

CGI

# CGI - Common Gateway Interface

- Метод, QueryString, заголовки запроса - через **переменные окружения**
- Тело запроса передается через **STDIN**
- Заголовки и тело ответа возвращаются через **STDOUT**
- HTTP код ответа передается через псевдозаголовок **Status**
- Поток ошибок **STDERR** направляется в лог ошибок сервера



# Переменные окружения CGI

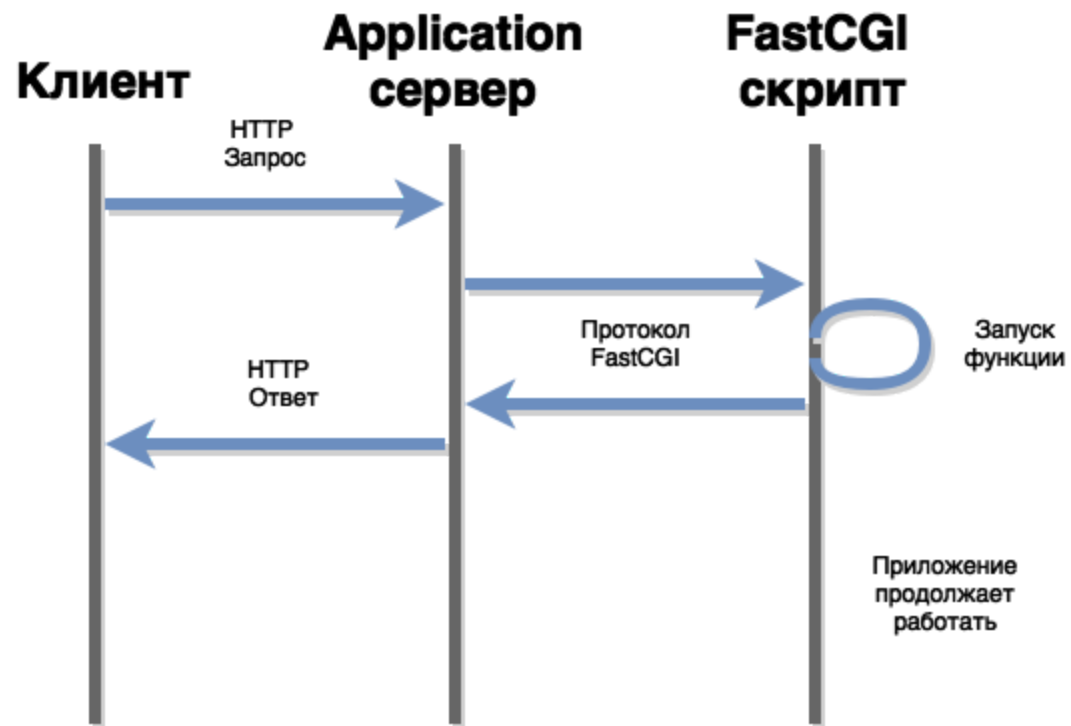
- `REQUEST_METHOD` - метод запроса
- `PATH_INFO` - путь из URL
- `QUERY_STRING` - фрагмент URL после `?`
- `REMOTE_ADDR` - IP адрес пользователя
- `CONTENT_LENGTH` - длина тела запроса
- `HTTP_COOKIE` - Заголовок `Cookie`
- `HTTP_ANY_HEADER_NAME` - любой другой HTTP заголовок

FastCGI и SCGI

# FastCGI и SCGI

Основная проблема CGI - низкая производительность. Протоколы FastCGI и SCGI призваны решить эту проблему путем демонизации приложения. Иногда это возможно сделать даже без изменения кода CGI приложения.





WSGI

# WSGI - актуальный протокол

**WSGI**, PSGI, Rack - протоколы вызова функции обработчика из application сервера. Сам application server при этом может выполняться в отдельном процессе или совпадать с web сервером. Как правило, при использовании этих протоколов в качестве application сервера выступает отдельный легковесный процесс.

# WSGI - обработчик

```
def wsgi_application(environ, start_response):  
    # бизнес-логика  
    status = '200 OK'  
    headers = [  
        ('Content-Type', 'text/plain')  
    ]  
    body = 'Hello, world!'  
    start_response(status, headers )  
    return [ body ]
```

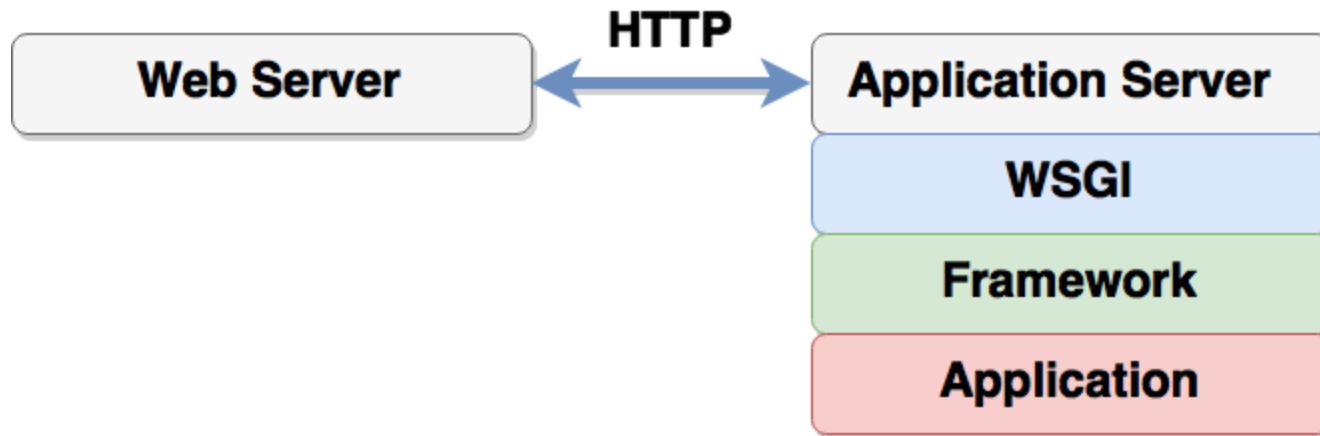
# Web Server Gateway Interface

- Обработчик - функция или класс (callable)
- Метод, QueryString, заголовки запроса - через аргумент **environ**
- Тело запроса передается через file-handle **wsgi.input**
- HTTP код ответа и заголовки ответа передаются через вызов функции **start\_response**
- Тело ответа возвращает в виде списка (iterable) из обработчика
- Поток ошибок должен быть направле в file-handle **wsgi.stderr**

# Переменные environ

- CGI-like переменные: `REQUEST_URI` , ...
- `wsgi.version` - версия WSGI протокола
- `wsgi.url_scheme` - схема текущего URL: https или http
- `wsgi.input` - file-handle для чтения тела запроса
- `wsgi.errors` - file-handle для вывода ошибок
- `wsgi.multithreaded` - ...
- `wsgi.multiprocess` - ...

# Развертывание WSGI



# Что ложится на приложение ?

- Анализ `PATH_INFO` и выбор конкретного обработчика
- Разбор конкретных заголовков, например `Cookie`
- Разбор `QUERY_STRING`
- Разбор тела запроса:
  - `x-www-form-urlencoded`
  - `multipart/form-data`
- Вывод правильных заголовков ответа