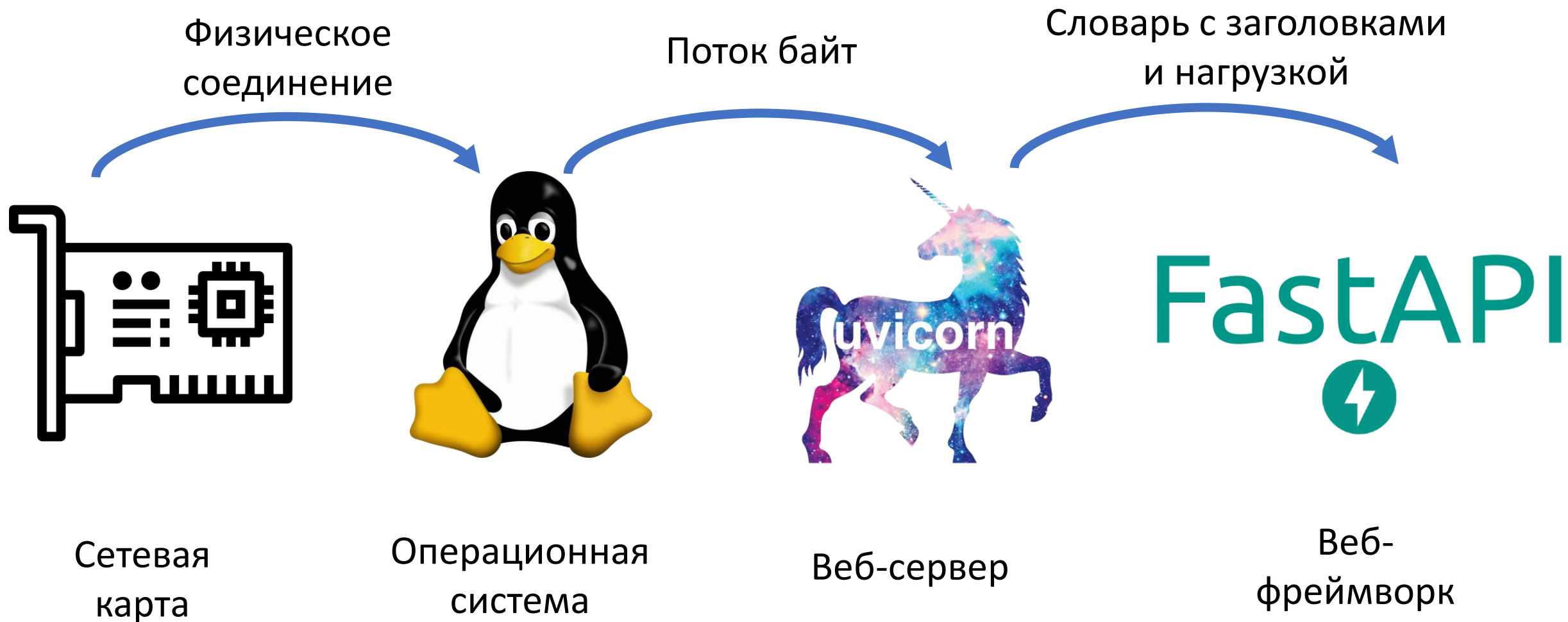


Веб-приложение:
обработка HTTP запросов

План

- Порядок обработки веб-запроса
- Веб-фреймворки
- Серверное приложение с использованием веб-фреймворка FastApi

Порядок обработки



Сетевая карта

- Оцифровывает физический сигнал и помещает его в RX(reception) буфер, доступный ОС
- Для оповещения о новых данных вызывает прерывание (interrupt) ОС
- Процедура обработки прерываний в ОС, считывает данные из буфера в inode ядра ОС



```
lrx----- 1 user user 64 Feb 12 12:34 3 -> socket:[1234567]
```

Операционная система


```
int sockfd = socket(AF_INET, SOCK_STREAM, 0);  
bind(sockfd, '127.0.0.1:80');  
listen(sockfd, 10);  
int clientfd = accept(sockfd, NULL, NULL);|
```

- Создает файл tcp-сокета и файловый дескриптор для процесса сервера
- Присваивает пару (server_ip,server_port) сокету, задающий сетевой интерфейс
- Начинает опрос интерфейсов. Новое подключение = новый файл, где записывается поток байт
- Ожидает подключения клиента и возвращает дескриптор для чтения и записи

Веб-сервер

```
POST /endpoint HTTP/1.1
Host: example.com
Content-Type: application/json
Content-Length: 34
```

```
{
  "name": "Alice",
  "age": 30
}
```



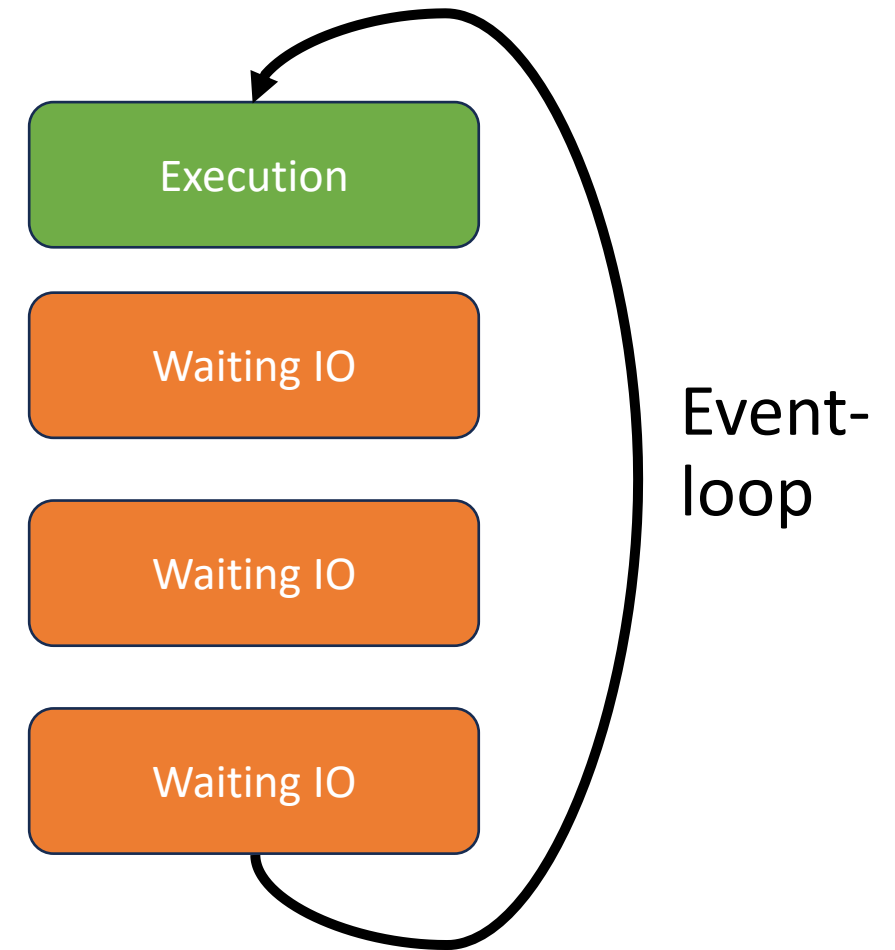
```
scope = {
    "type": "http",
    "method": "POST",
    "path": "/endpoint",
    "query_string": b"",
    "headers": [
        (b"host", b"example.com"),
        (b"content-type", b"application/json"),
        (b"content-length", b"34")
    ],
    "client": ("127.0.0.1", 5000),
    "server": ("127.0.0.1", 8000),
}
```



```
{
    "type": "http.request",
    "body": b'{"name": "Alice", "age": 30}',
    "more_body": False
}
```

Веб-сервер

- Запускает веб-фреймворк и серелизует для него входящее сообщение:
 - определяет начало и конец http запроса;
 - валидирует формат сообщения;
 - выделяет заголовки и нагрузку в отдельные словари;
- Организует модель асинхронной обработки с использованием мультиплексоров ввода/вывода;
- В случае TLS высылает сертификат севера, шифрует и дешифрует сообщение;



Веб-фреймворк

- Связывает метод обработки (handler) на ЯП с запросом по словарю, предоставленному веб-сервером (routing);
- Серилизует нагрузку и заголовки в переменные ЯП;
- Обрабатывает ошибки:
 - в клиентских запросах;
 - базы данных;
 - методов обработки.

```
@app.get("/task/{id}")
async def tasks_handler(id: int):
    task = await db.get_task(id)
    if not task:
        raise HTTPException(
            status_code=400,
            detail="Task not found"
        )
    return task
```


Обсуждение

В ethernet-кабеле возникла наводка от стороннего канала связи. Кто из цепи обработки веб-запроса заметит помехи и попробует исправить их?

- сетевая карта;
- операционная система;
- веб-сервер;
- веб-фреймворк.

Веб-фреймворк FastApi

Компоненты

- Модель сериализации данных Pydantic;
- Модель рутинга FastApi;
- Автоматическая генерация документации Swagger;
- Запуск веб-сервера;

Модель сериализации Pydantic

- Структура модели объявляется с учетом типов ее аргументов;
- Допускается произвольная вложенность модели и контейнеры map, list;
- При валидации некорректных данных возвращается ошибка с указанием поля

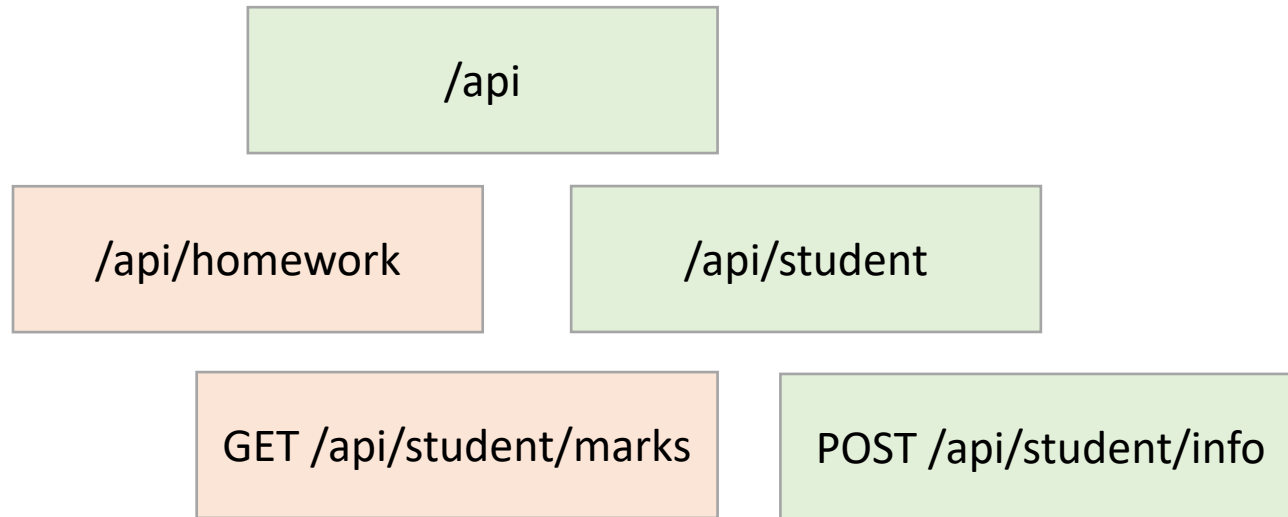
```
from pydantic import BaseModel

class Task(BaseModel):
    id: int
    text: str

class Homework(BaseModel):
    id: int
    tasks: list[Task]

Task.model_validate_json(
    """{
        "id": 1,
        "text": "Why ...?"
    }"""
)
```

Рутинг (префиксное дерево)



```
{  
  "method": "POST",  
  "path": "/api/student/info/10"  
}
```

- Название обработчика при регистрации помещается в префиксное дерево;
- При поступлении запроса проводится поиск по дереву. Параметры запроса передаются в функцию для получения ответа

Рутинг (код)

- Для организации префиксного древа используется рутеры
- Рутеры можно вкладывать друг в друга и в приложение
- Тэги позволяют организовать методы в документации

```
from fastapi import FastAPI, APIRouter

student_router = APIRouter(
    prefix='/api/student',
    tags=['Students']
)

@student_router.get('/info')
def student_info():...

@student_router.get('/marks')
def student():...

app = FastAPI()
app.include_router(student_router)
```

Документация

Запуск веб-сервера

- Uvicorn поддерживает неблокирующее чтение, обработку и запись запроса (ASGI запроса)
- Сам аллоцирует и связывает сокет с заданным (ip,port)
- Использует оптимизации цикла событий для ускорения приложения
- Логирует все входящие запросы

```
from fastapi import FastAPI
import uvicorn

app = FastAPI()

@app.get("/")
def wow():
    return "wow"

uvicorn.run(
    app,
    host="127.0.0.1",
    port=8080
)
```


Практика

Серверная часть приложения

- запись посещения студента. POST /attendance/record;
- посещаемость всех студентов. GET /attendance;
- отправка запроса в веб-браузере по нажатию кнопки и открытию веб-странички.

Домашнее задание

- Stepik
- Реализовать метод для получения информации только по одному студенту

Дополнительные слайды

Устройство процесса

Каждый процесс в Linux хранится в директории /proc, хранимой в RAM:

- /proc/<pid>/cmdline – команда запуска процесса;
- /proc/<pid>/status – статус процесса и его метаданные;
- /proc/<pid>/fd/ – Файловые дескрипторы, открытые процессом;
- /proc/<pid>/exe – Символическая ссылка на исполняемый файл процесса.