

ПРОГРАММИРОВАНИЕ В INTERNET

ВВЕДЕНИЕ

Лектор

Дубовик Марина Владимировна

старший преподаватель кафедры ИСиТ



ауд. 115-1



dubovik@belstu.by

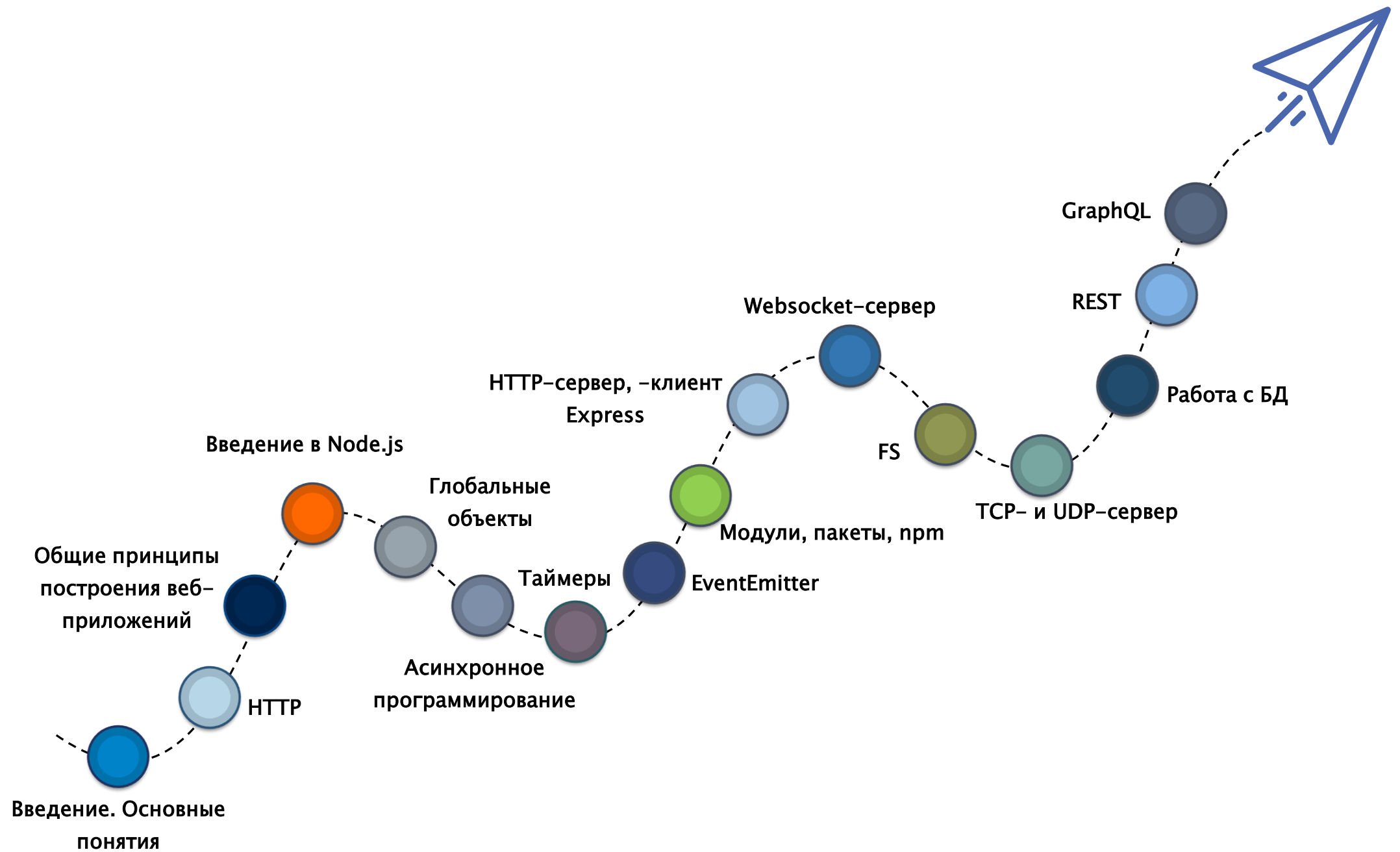


https://vk.com/marina_dubovik



Учебный план

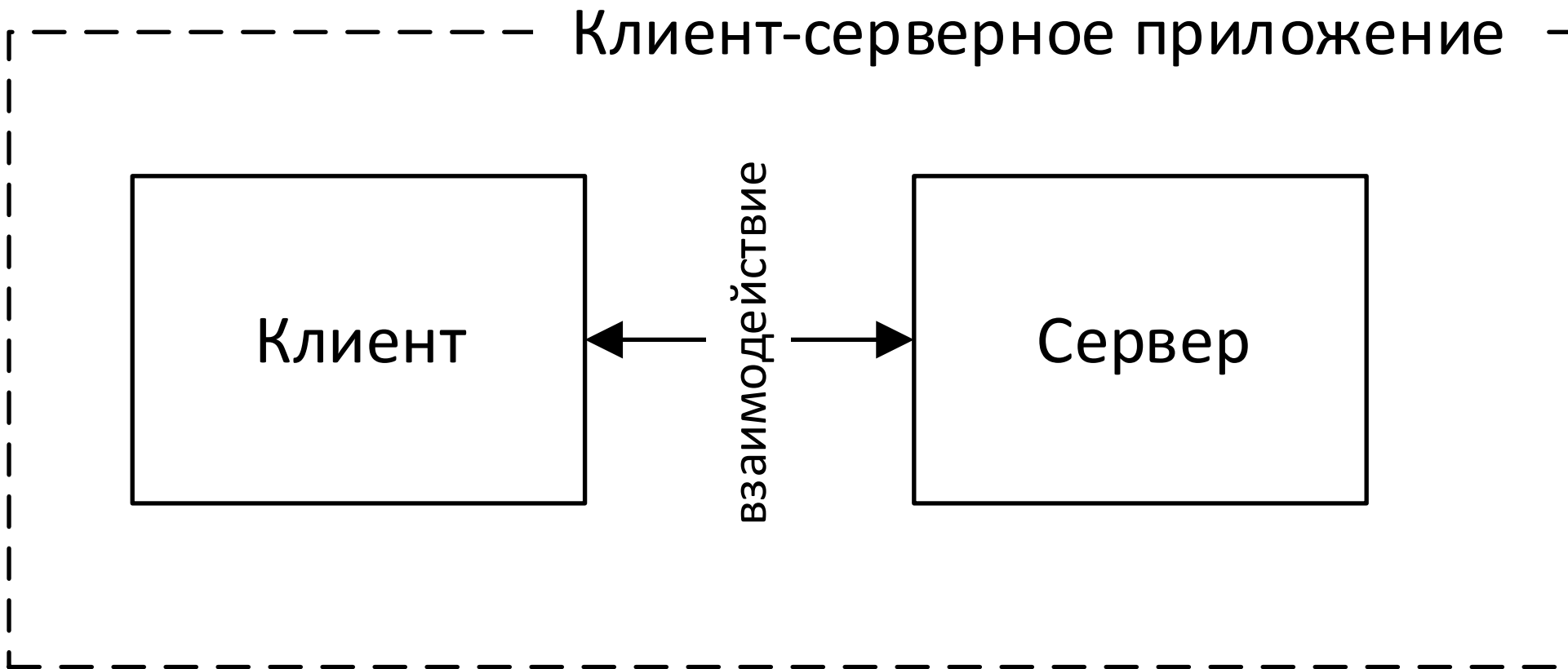
Семестр	Лекции (час)	Лабораторные работы (час)	Итого (час)	Форма контроля
4	34	26	60	зачет



Клиент– серверное приложение =

приложение (программа) с клиент–серверной архитектурой:

- состоит из двух компонент – **клиента** и **сервера**;
- клиент и сервер взаимодействуют между собой в соответствии с заданными правилами (**протоколами**);
- для взаимодействия между клиентом и сервером в соответствии с правилами (протоколом) должно быть установлено **соединение**;
- **инициатором соединения – клиент.**

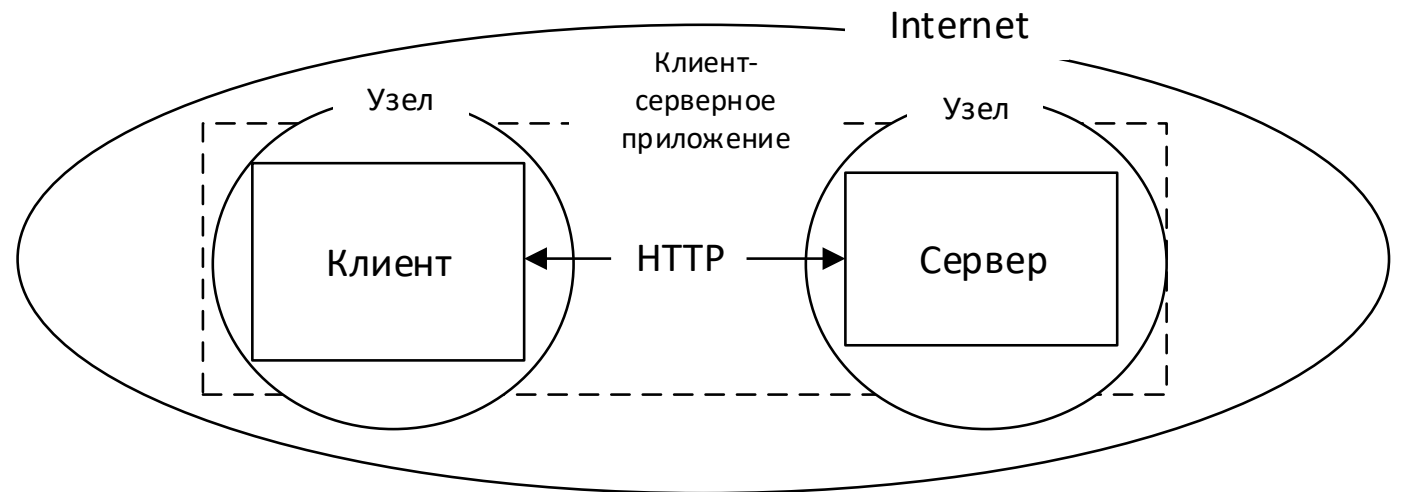


Web-приложение =

клиент-серверное приложение, у которого клиент и сервер взаимодействуют по протоколу HTTP.

Когда говорят о разработке web-приложения, говорят о разработке **frontend** (клиента) и **backend** (сервера)

Курс посвящен разработке серверной части web-приложения или иначе разработке web-сервера (backend).



Узел сети Интернет

=

устройство, имеющее IP-адрес и подключенное сети Интернет (обычно к сети Интернет-провайдера). Каждый узел характеризуется своей программно-аппаратной платформой – аппаратурой и операционной системой.

Сеть Интернет

=

- 1) сеть на основе TCP/IP;
- 2) стандарты Internet (RFC, STD);
- 3) службы Интернет (DNS, SMTP/POP3/IMAP, WWW, FTP, Telnet, SSH,...);
- 4) организации, управляющие сетью Internet (ISOC, IETF, W3C, ICANN, IANA, ...).

Кроссплатформенное приложение =

приложение, способное
работать на **более чем одной
программно-аппаратной**
(аппаратура + операционная
система) **платформе**.



Кроссплатформенность может быть достигнута различными способами:

- 1) на уровне компилятора (C, C++); 2) на уровне среды (или фреймворка) исполнения (Java/JVM, C#/.NET CORE/CLR, JS/Node, ...)

C++



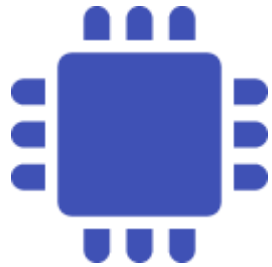
Java/C#



Технологии для разработки кроссплатформенных web-серверов

- PHP / Apache, LAMP;
- Java / JVM / Application Server;
- C# / ASP.NET CORE;
- Python / Django;
- Ruby on Rails;
- JS / Node.js,

Ресурсы, потребляемые web-сервером



Процессор

**Оперативная
память**



**Жесткий
диск**



**Сетевой
интерфейс**

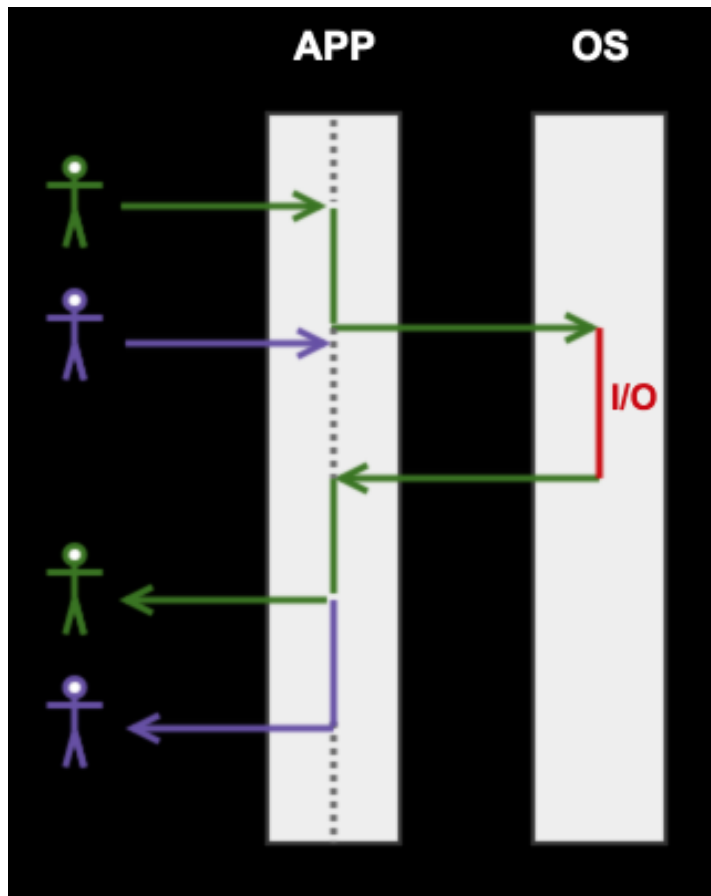
Вычислительные задачи можно разделить на две категории:

- **CPU bound** – нагружающие процессор (*вычисление факториала, вычисление хэша, процесс шифрования...*). Такие операции **нагружают вычислительные мощности** текущего устройства.
- **I/O bound** – требующие **ввода/вывода** (*поиск по файлу, подсчет числа строк в файле, копирование директории...*). В своей базовой форме Node.js лучше всего подходит для этого типа вычислений.

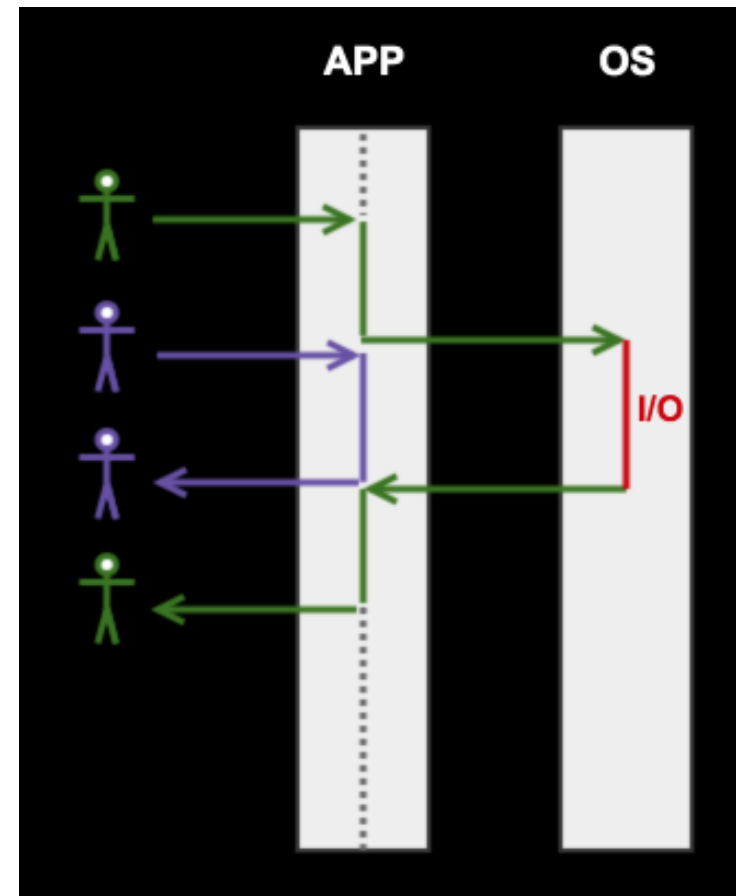
ЧТЕНИЕ HTTP-запроса	I/O
ПАРСИНГ HTTP-запроса	CPU
ЗАПРОСЫ к БД	I/O
ЗАПРОСЫ к другим СЕРВЕРМ	I/O
ВЫЧИСЛЕНИЯ	CPU
ФОРМАТИРОВАНИЕ ответа	CPU
ОТПРАВКА HTTP-Ответа	I/O

IO-операции

блокирующие



неблокирующие



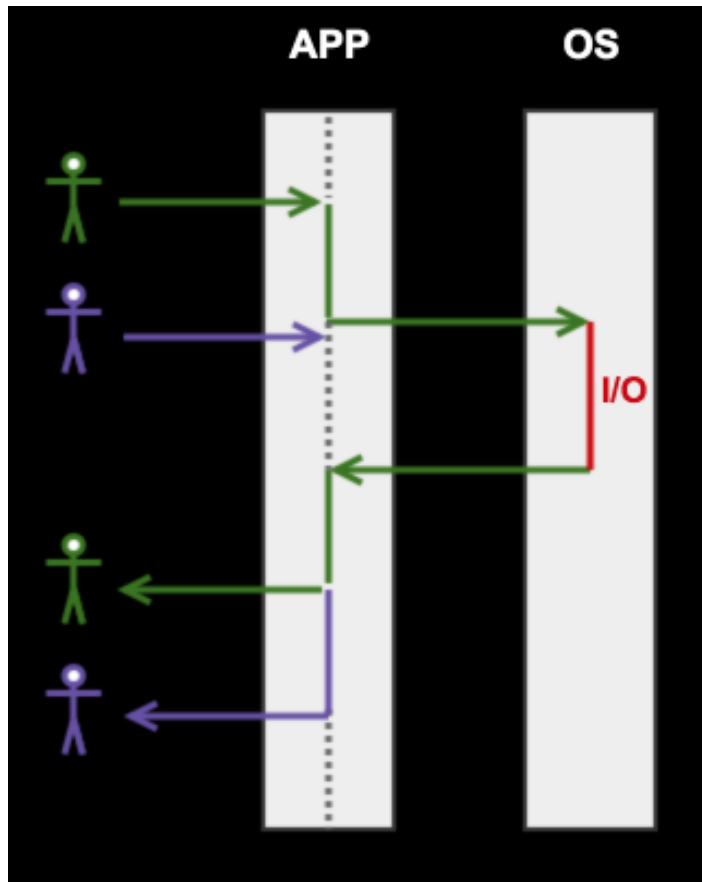
Подходы для решения проблем блокирующего ввода/вывода:

1. Применение **многопоточности** (ограничение по количеству потоков, каждый поток требует дополнительной памяти, синхронизация, отладка).
2. Применение паттерна **Reactor**.

Apache – многопоточность, Nginx – Reactor.

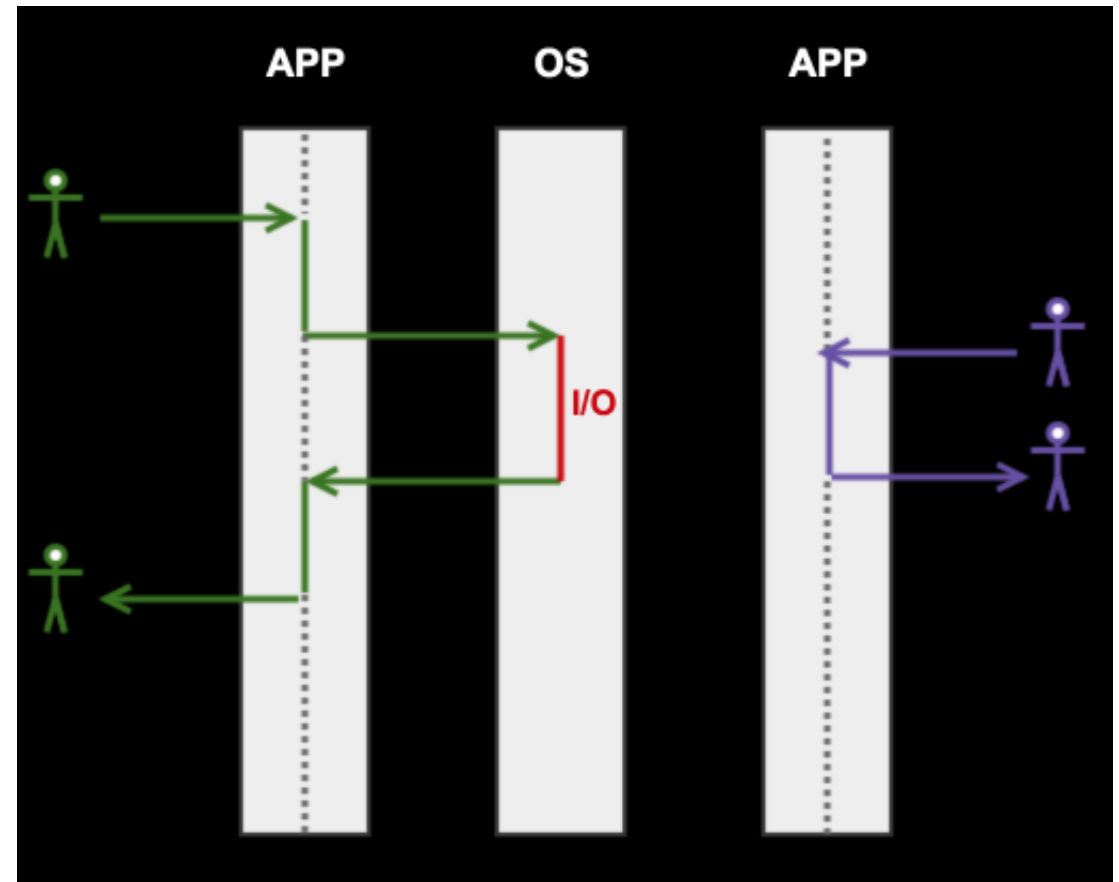
Однопоточность

(единственная задача в один момент времени)



Многопоточность

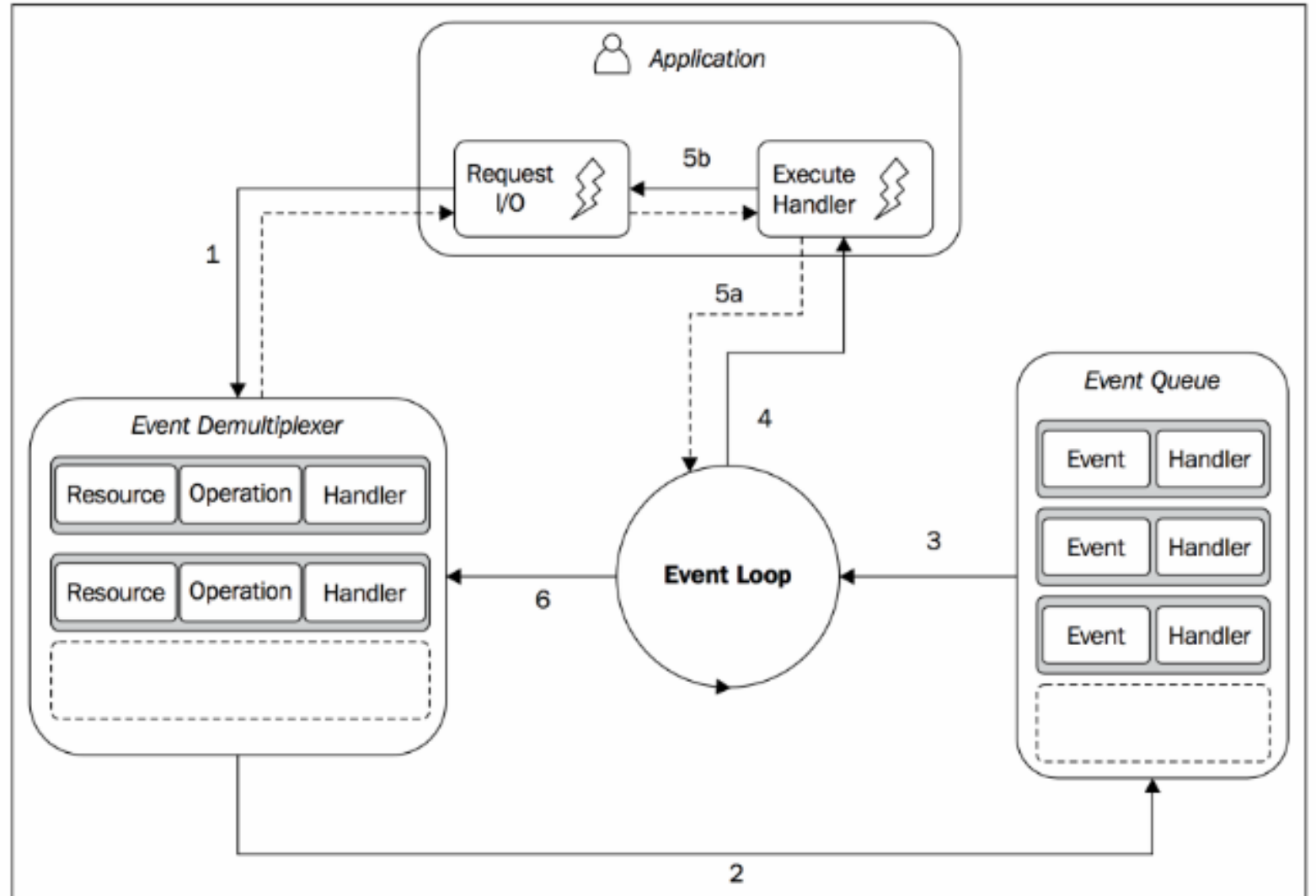
(различные задачи в один момент времени)



Паттерн Reactor

Используется при обработке параллельных запросов к сервису.

Демультимплексор разбирает прибывшие запросы и синхронно перенаправляет их на соответствующие обработчики запросов.



Какой подход используется в Node.js?

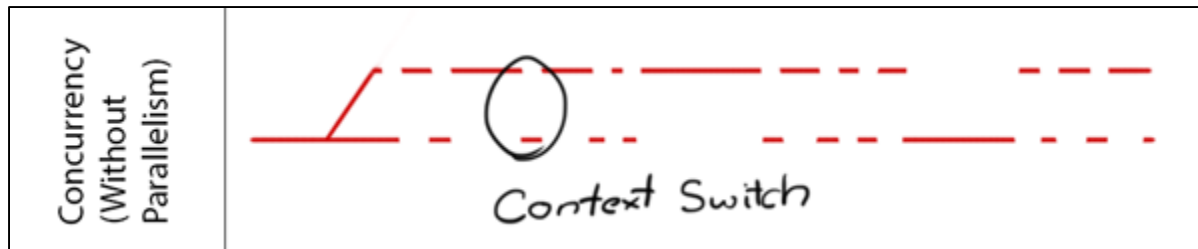
Многопоточность



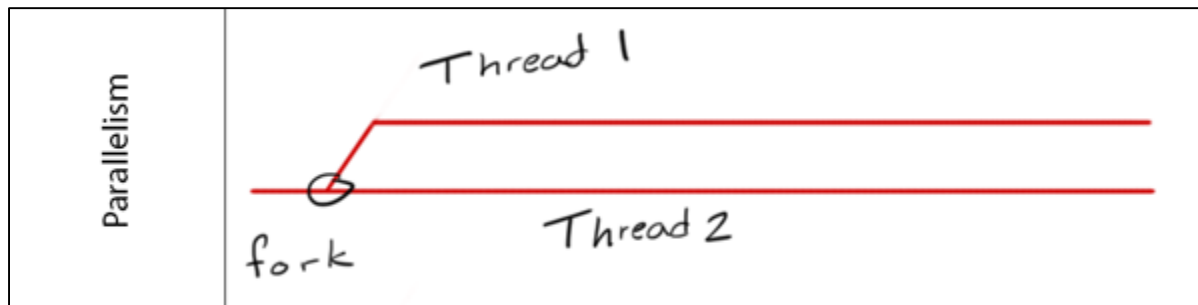
Паттерн Reactor



Конкурентность и параллельность



- Работа с несколькими **потоками управления**.
- Каждый поток получает **квант времени** на исполнение.



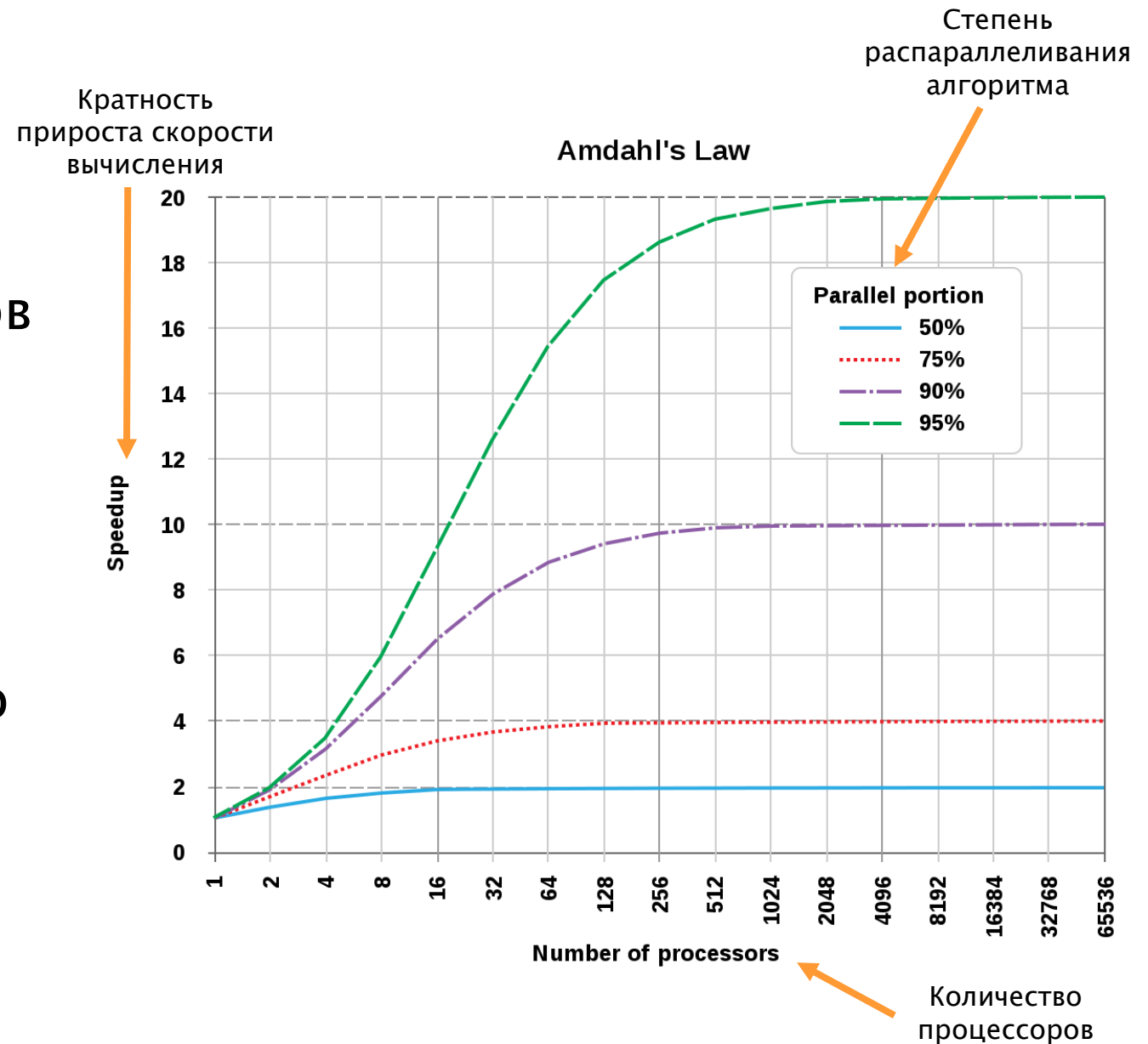
- Необходима **аппаратная составляющая**.
- Наличие потоков **необязательно**.

Закон Амдала

Идеальный случай:
система из n процессоров
могла бы ускорить
вычисления в n раз.

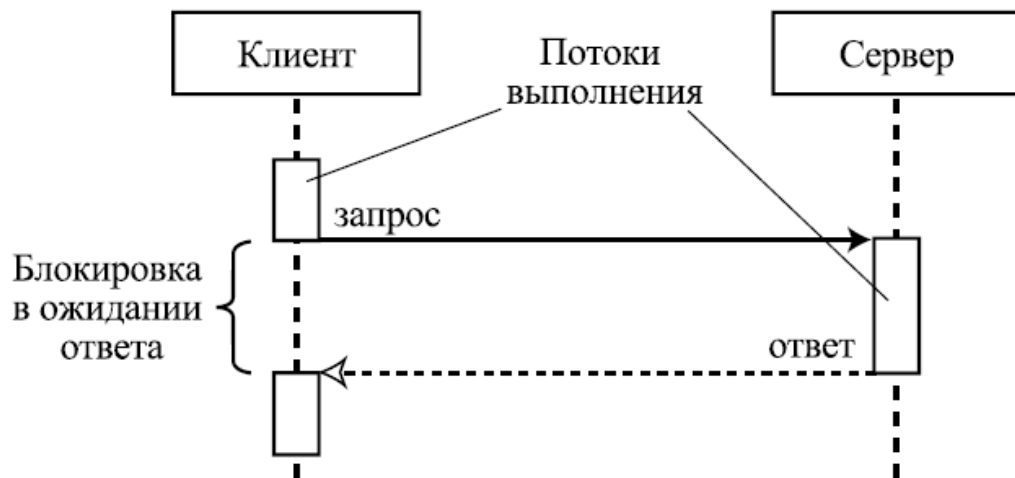
Проблемы:

- невозможность полного распараллеливания ни одной из задач;
- невозможность одинаковой загрузки каждого процессора.



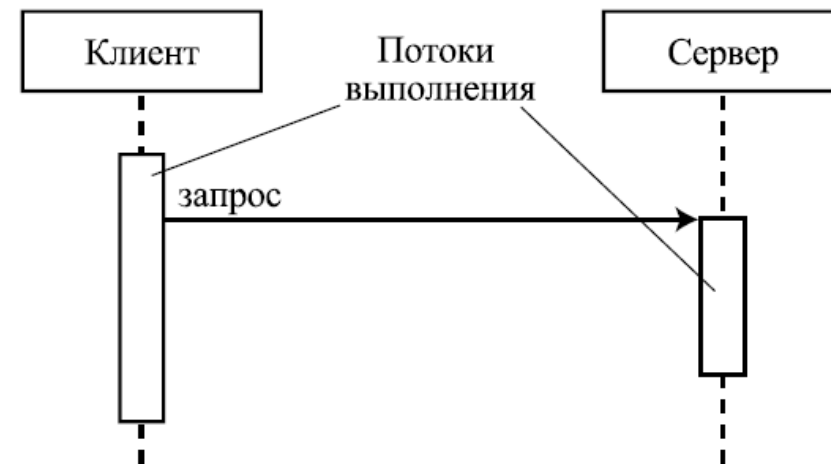
Синхронный запрос

запрос, при котором поток, выдавший http-запрос, блокируется до поступления ответа



Асинхронный запрос

запрос, при котором поток, выдавший http-запрос, не блокируется до поступления ответа; для обработки ответа применяется функция обратного вызова.



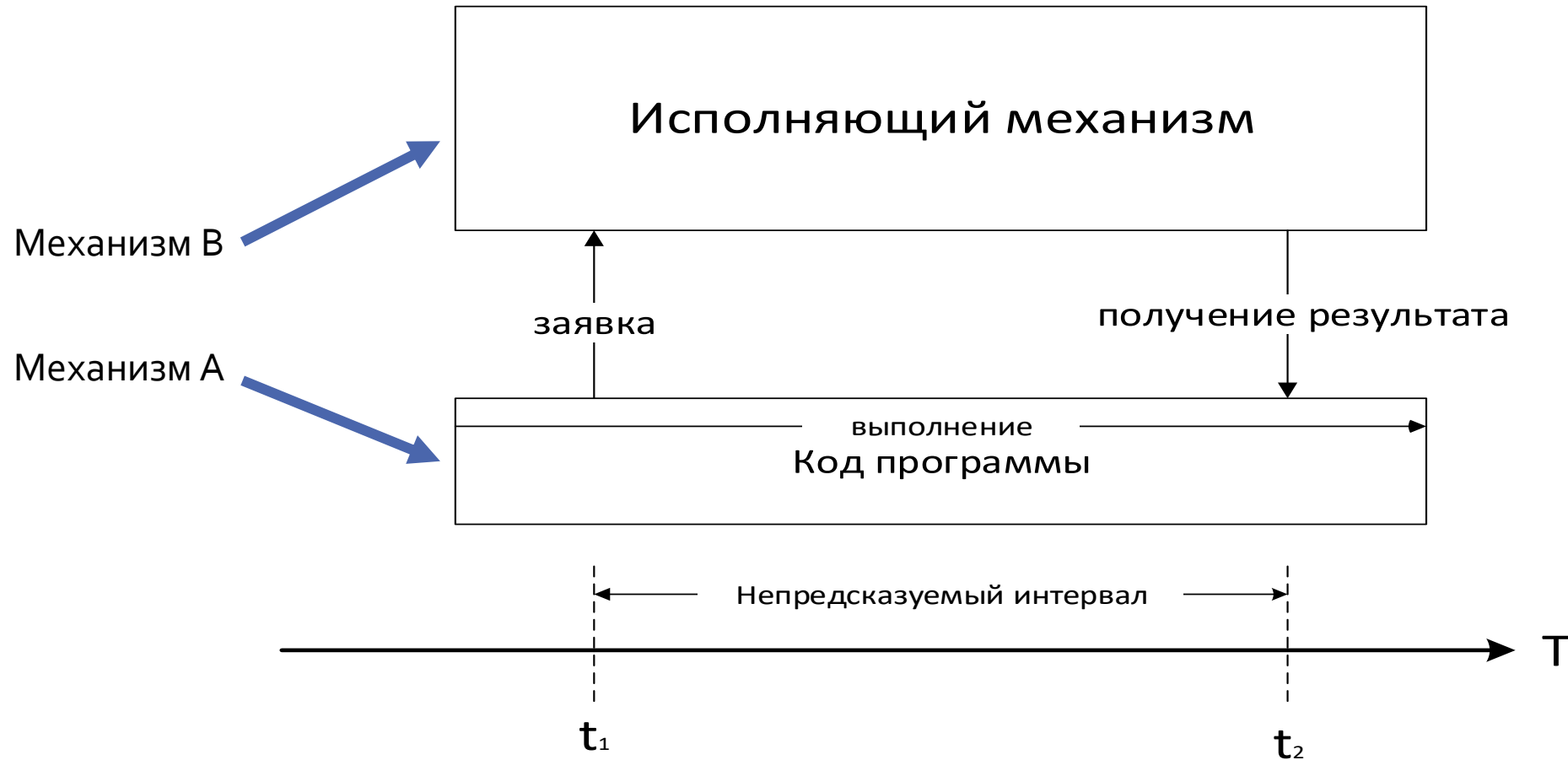
Асинхронность =

операция называется асинхронной, если ее выполнение осуществляется в 2 фазы:

- 1) **заявка на исполнение**;
- 2) **получение результата**; при этом участвуют два механизма: А–механизм, формирующий заявку и потом получающий результат; В–механизм, получающий заявку от А, исполняющий операцию и отправляющий результат А; продолжительность исполнения операции В–механизмом, как правило, непредсказуемо; в то время пока В–механизм исполняет операцию, А–механизм выполняет собственную работу.

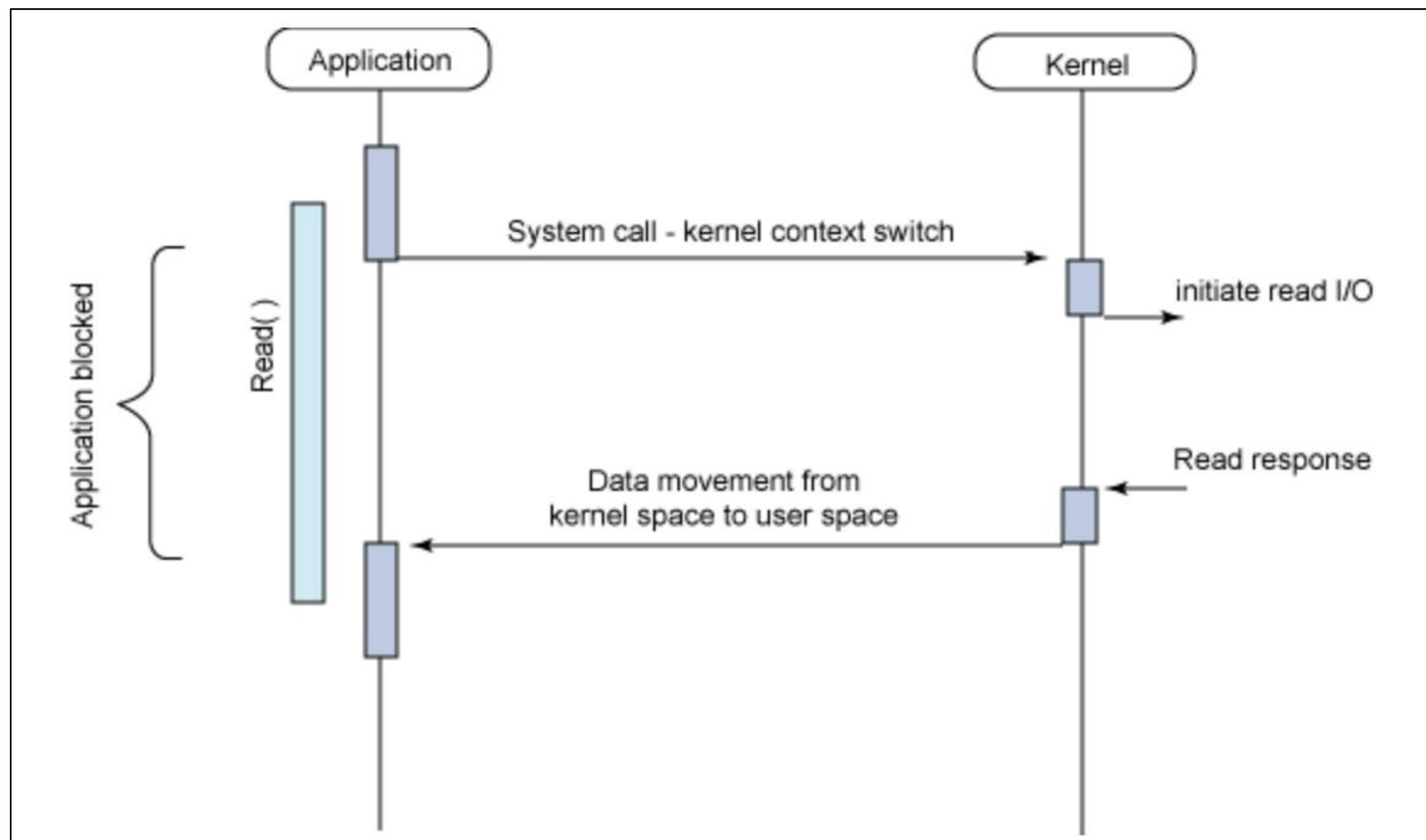
Применение асинхронности не противоречит применению многопоточности.

Асинхронность в программировании



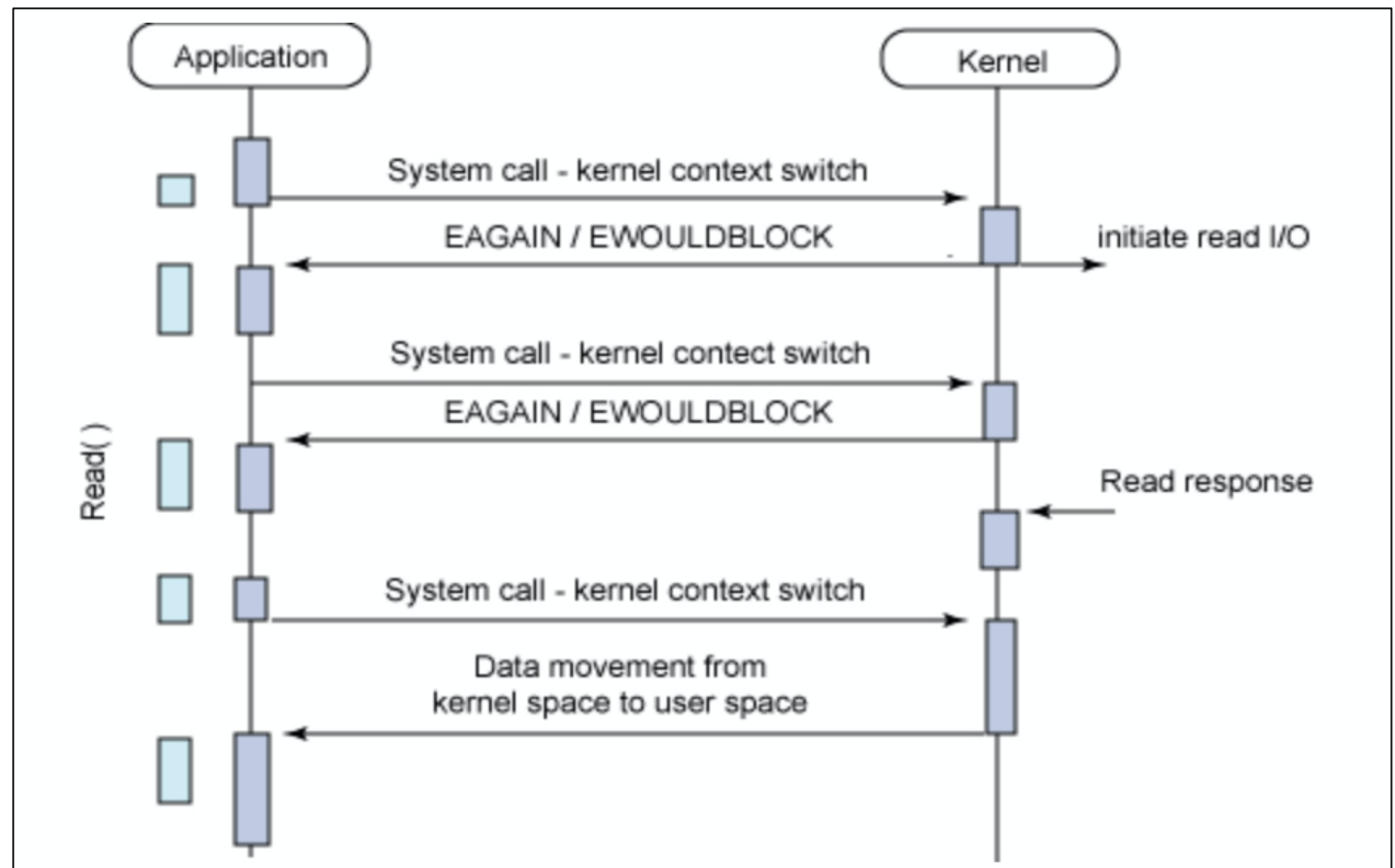
Синхронный блокирующий ВВОД/ВЫВОД

Приложение
блокируется до
тех пор, пока
операция
ввода/вывода не
будет завершена.



Синхронный неблокирующий ввод/вывод

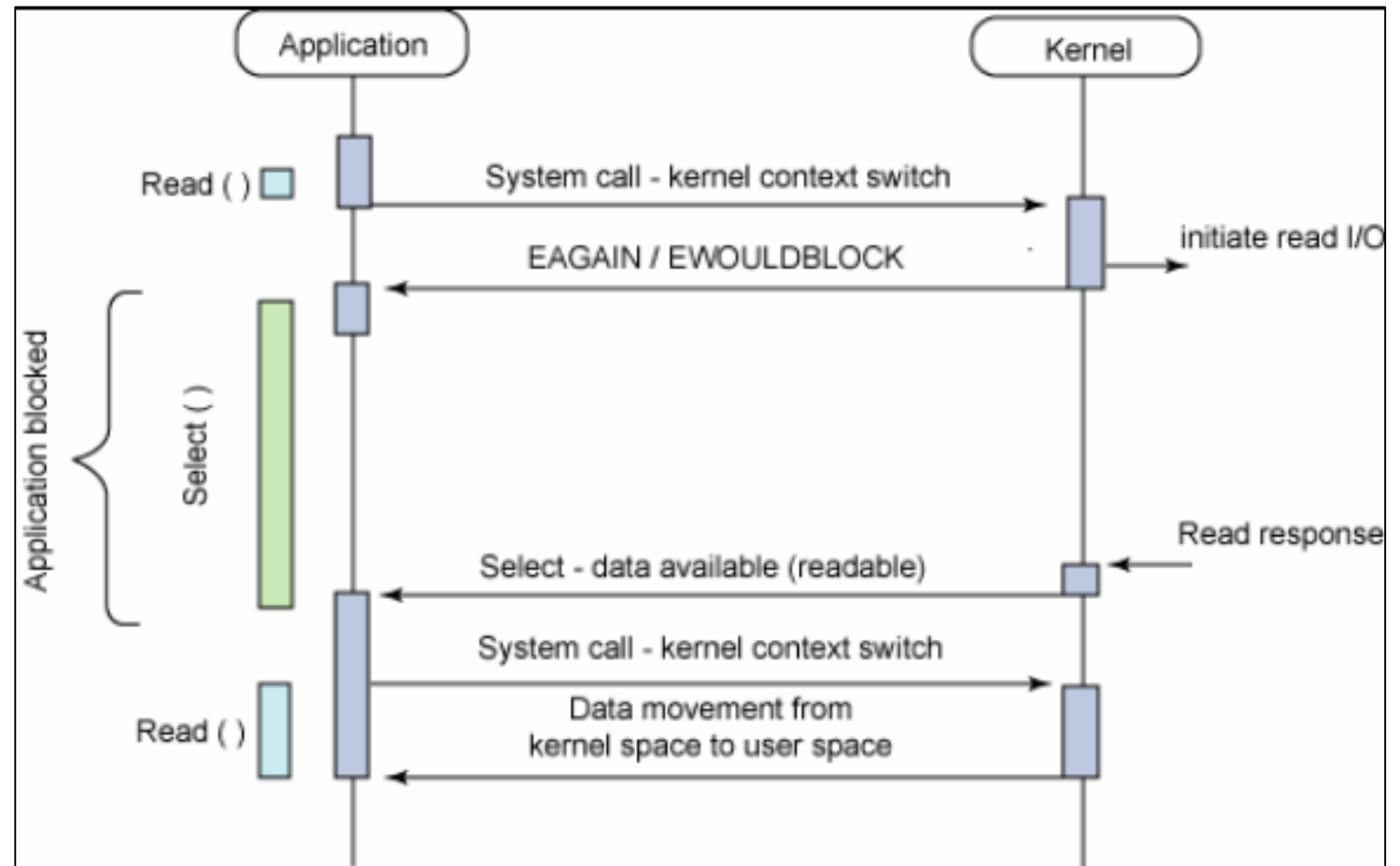
Приложение **может сделать что-то** еще вместо ожидания ввода/вывода, однако происходит **увеличение количества переключений контекста** между ядром и пользовательским пространством.



Асинхронный блокирующий ввод/вывод

Ввод-вывод инициируется как неблокирующий, но **уведомление принимается в режиме блокировки** с помощью системного вызова `select ()`.

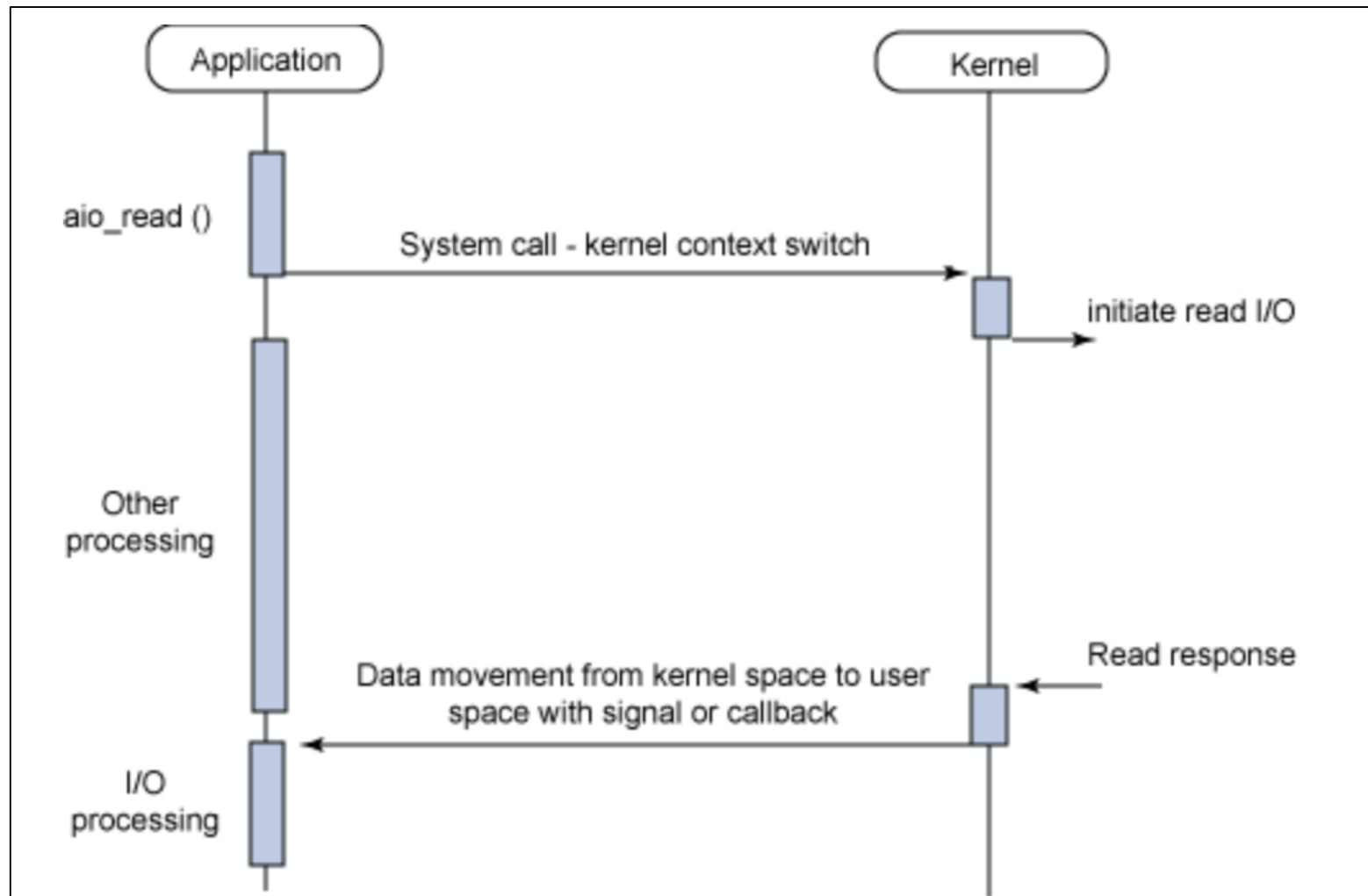
Как только файловый дескриптор готов к работе, вызов `read()` разблокируется, и приложение сможет извлечь и обработать данные.



Асинхронный неблокирующий ввод/вывод

Приложение **может выполнять вычисления**, пока чтение выполняется в фоне ядром.

При готовности ответа генерируется сигнал или обратный вызов для завершения ввода-вывода.



AJAX =
Asynchronous JavaScript
and XML

методология (подход) построения динамических приложений, в которых **не осуществляется полная перезагрузка** HTML-страниц.

Пример асинхронных запросов

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="UTF-8">
  <title>Async requests</title>
</head>

<body>
  <h1>XMLHttpRequest</h1>
  <input type="submit" value="get users" onclick="getUsers()">
  <div id="divResult1"></div>

  <h1>Login</h1>
  <form id="myForm">
    Email: <br>
    <input type="text" name="email" required>
    <br> <br>
    Password: <br>
    <input type="password" name="password" required>
    <br> <br>
    <input type="submit" value="Login"></button>
  </form>
  <div id="divResult2"></div>

  <script>
    ...
  </script>
</body>

</html>
```

XMLHttpRequest

get users

Login

Email:

Password:

Login

XMLHttpRequest

```
<script>
function getUsers() {
  const xhr = new XMLHttpRequest();

  xhr.open("GET", "https://reqres.in/api/users?page=2", true);

  console.log("XMLHttpRequest ", 1);

  xhr.onreadystatechange = () => {
    console.log("XMLHttpRequest ", 2);
    if (xhr.readyState == 4) {
      if (xhr.status == 200)
        document.getElementById("divResult1").innerText = xhr.responseText;
    }
  };

  xhr.onerror = (e) => {
    console.log("XMLHttpRequest.onerror ", e)
  };

  console.log("XMLHttpRequest ", 3);

  xhr.send();

  console.log("XMLHttpRequest ", 4);
};

// ...

</script>
```

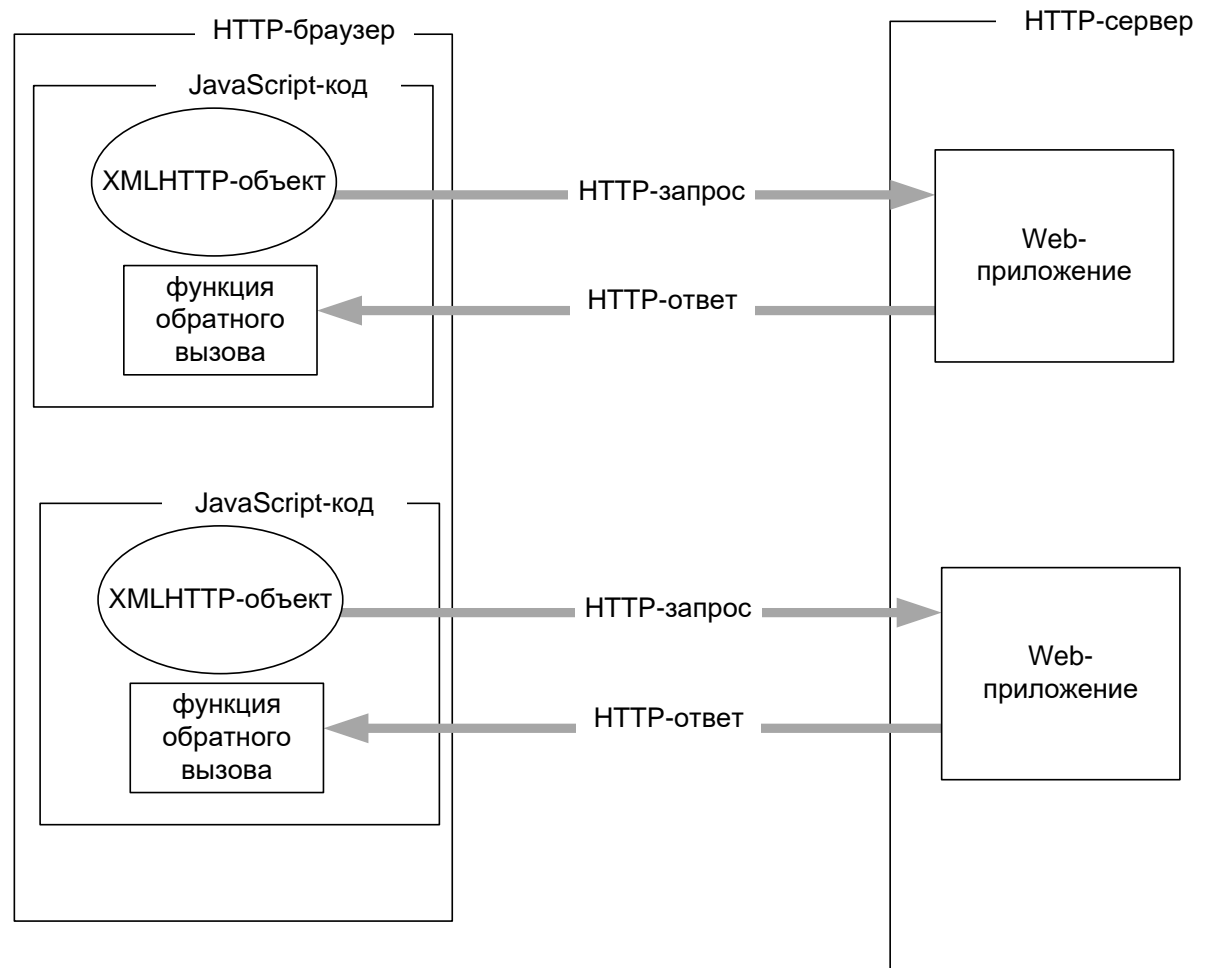
XMLHttpRequest

get users

```
{
  "page": 2,
  "per_page": 6,
  "total": 12,
  "total_pages": 2,
  "data": [
    {
      "id": 7,
      "email": "michael.lawson@reqres.in",
      "first_name": "Michael",
      "last_name": "Lawson"
    },
    {
      "id": 8,
      "email": "lindsay.ferguson@reqres.in",
      "first_name": "Lindsay",
      "last_name": "Ferguson"
    },
    {
      "id": 9,
      "email": "tobias.funke@reqres.in",
      "first_name": "Tobias",
      "last_name": "Funke"
    },
    {
      "id": 10,
      "email": "byron.fields@reqres.in",
      "first_name": "Byron",
      "last_name": "Fields"
    },
    {
      "id": 11,
      "email": "george.edwards@reqres.in",
      "first_name": "George",
      "last_name": "Edwards"
    },
    {
      "id": 12,
      "email": "rachel.howell@reqres.in",
      "first_name": "Rachel",
      "last_name": "Howell"
    }
  ],
  "support": {
    "url": "https://reqres.in/#support-heading",
    "text": "To support reqres.in visit our GitHub site!"
  }
}
```

	Elements	Console	Recorder
		top	Filter
		[ContentMain]	
		[ContentService] document.readyState	
		[ContentService.SetContentInitData]	
		1676367081, FrameId: 0}	
		XMLHttpRequest 1	
		XMLHttpRequest 3	
		XMLHttpRequest 4	
		XMLHttpRequest 2	
		>	

Асинхронный XMLHttpRequest запрос



fetch

```
<script>
  //...
  document.querySelector("#myForm").addEventListener("submit", (event) => {
    event.preventDefault();

    let formData = new FormData(event.target);
    let data = Object.fromEntries(formData);

    fetch("https://reqres.in/api/register", {
      method: "POST",
      mode: "cors",
      headers: {
        "Accept": "application/json",
        "Content-Type": "application/json"
      },
      body: JSON.stringify(data)
    })
      .then(resp => resp.text())
      .then(respText => document.getElementById("divResult2").innerText = respText)
      .catch(err => console.log(err))
  });
</script>
```

Login

Email:

Password:

```
{"id":4,"token":"QpwL5tke4Pnpja7X4"}
```