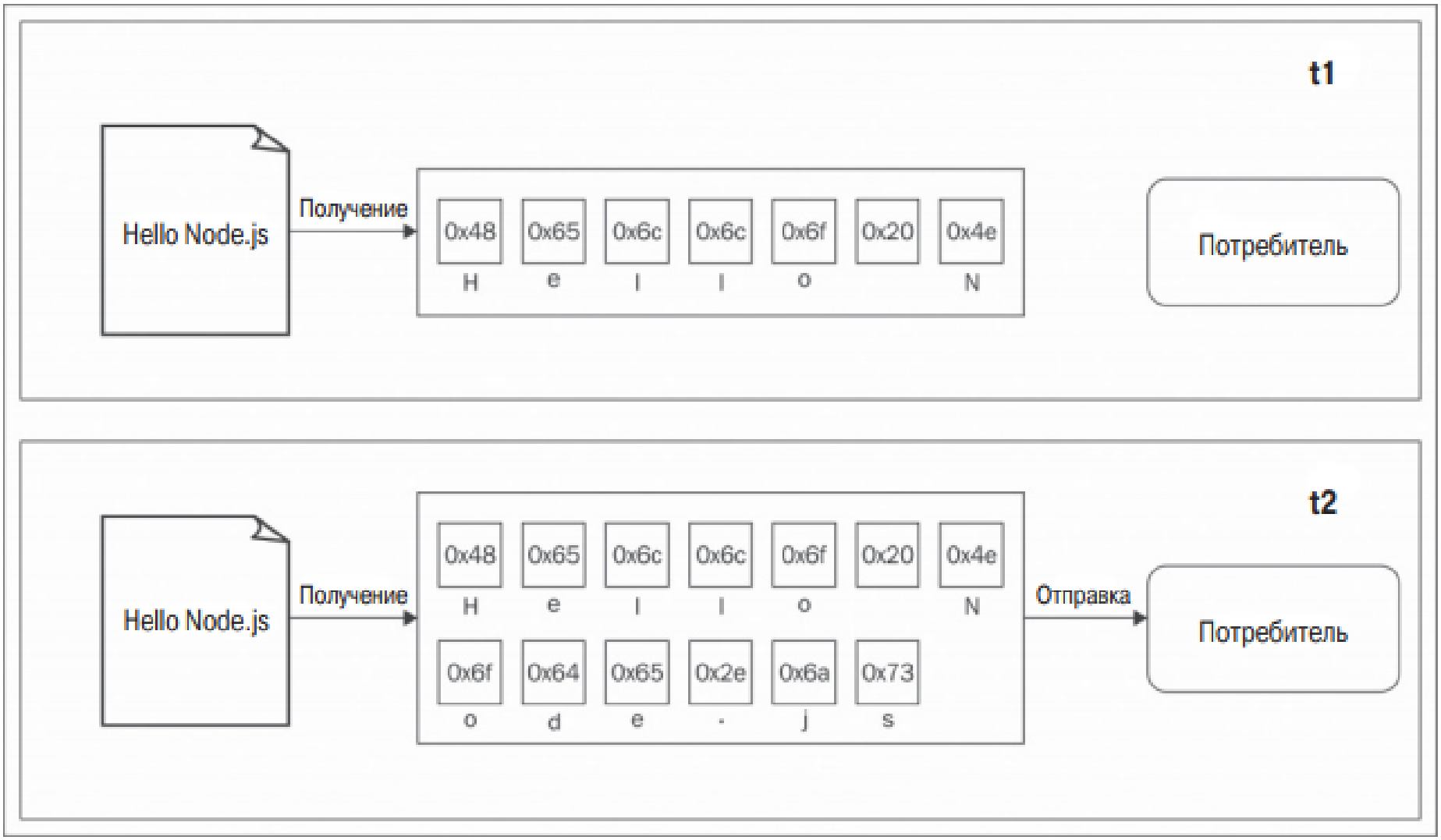
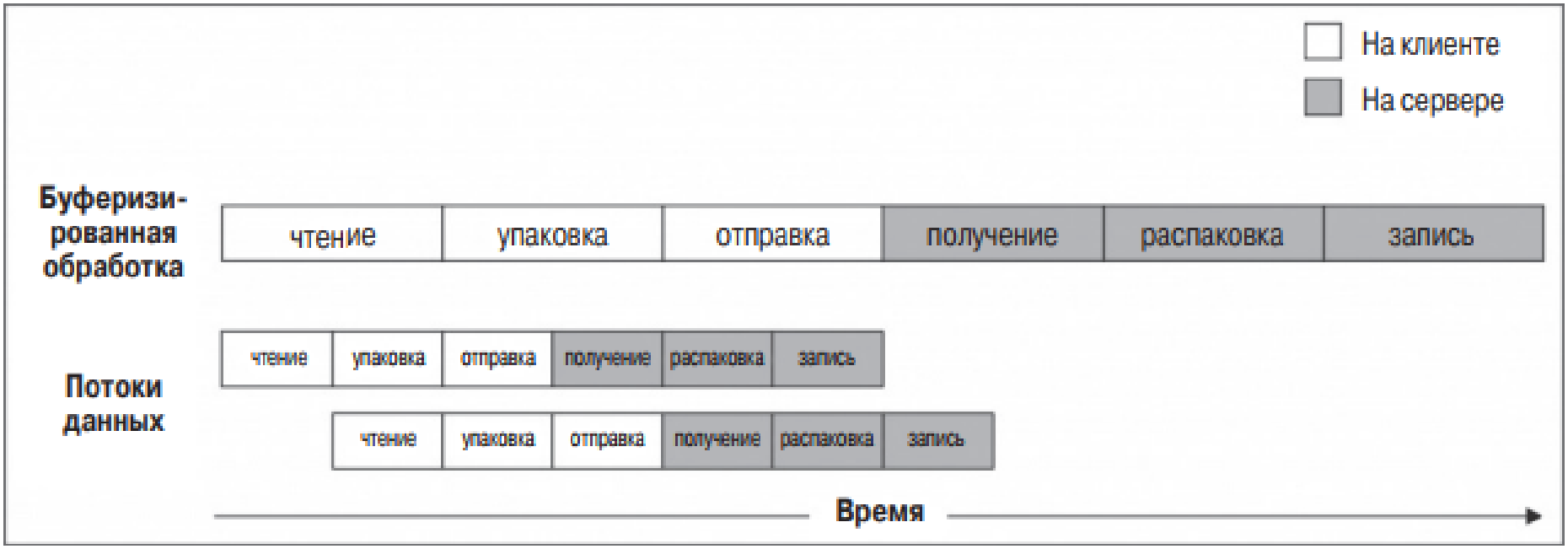
Stream - потоки данных, отправка данных методом POST

БУФЕРИЗАЦИЯ



ПОТОК ДАННЫХ

ЭФФЕКТИВНОСТЬ ПОТОКОВ ДАННЫХ



Отличие от буферизированной обработки:

- меньше использует оперативную память;
- меньше время выполнения из-за отсутствия ожидания полного завершения этапа обработки;
- можно объединять потоки данных

МОДУЛЬ STREAM

stream – абстрактный интерфейс для работы с потоками данных в Node.js.

Потоки бывают на чтение, на запись, одновременно на чтение и запись, потоки трансформации.

Чтобы использовать данный модуль его необходимо подключить с помощью require:

```
const stream = require('stream');
```

КЛАСС STREAM.READABLE

Класс **stream.Readable** – создаёт поток на чтение.
Основные методы экземпляра:

- on(eventName, listener) или addListener(eventName, listener) – задание функции обработчика listener на указанное событие eventName;
- pipe(streamWrite) – объединяет с потоком обладающем интерфейсом потока записи.
- read([size]) – читает данные из потока.

Примеры событий eventName:

- 'readable' – генерируется, когда есть доступные данные для чтения из потока;
- 'data' – генерируется, когда поток готов отдать часть данных;
- 'end' – генерируется при окончании чтения.

КЛАСС STREAM.READABLE

Класс **stream.Writable** – создаёт поток на запись.
Основные методы экземпляра:

- on(eventName, listener) или addListener(eventName, listener) – задание функции обработчика listener на указанное событие eventName;
- write(chunk[, encoding][, callback]) – запись части данных в поток записки.
- end([chunk][, encoding][, callback]) – сигнализирует о том, что в открытый для записи поток больше не поступит данных на запись, завершение потока.

ОТПРАВКА POST ЗАПРОСА

Особенности отправки формы методов POST:

- данные формы (пары «имя-значение») отправляются в теле запроса, а не в URL;
- POST запросы никогда не кэшируются
- Запросы POST не сохраняются в журнале обозревателя
- Запросы POST не могут быть закладками
- Запросы POST не имеют ограничений по длине данных

ОТПРАВКА POST ЗАПРОСА

Особенности отправки формы методов POST:

❖ Доступны три кодировки, задаваемые через атрибут enctype:

application/x-www-form-urlencoded – в виде пар ключ=значение:

name1=value1&name2=value2

multipart/form-data – в этой кодировке поля пересылаются одно за другим, через строку-разделитель

...Заголовки...
Content-Type: multipart/form-data; boundary=RaNdOmDeLiMiTeR
--RaNdOmDeLiMiTeR
Content-Disposition: form-data; name="name1"
value1
--RaNdOmDeLiMiTeR
Content-Disposition: form-data; name="name2"
value2
--RaNdOmDeLiMiTeR--

text-plain – текстовой формат

Content-Type: text/plain
name1=value1
name2=value2

ОТПРАВКА ФОРМЫ НА КЛИЕНТ

Вариант 1:

index.html:

```
<form name="person" method="POST" url="/"
        enctype="application/x-www-form-urlencoded">
      <input name="name" value="Иван">
      <input name="surname" value="Иванович">
      <input type="submit" value="Отправить">
</form>
```

ОТПРАВКА ФОРМЫ НА КЛИЕНТ

Вариант 2:

index.html:

```
<script src ="script.js" type= "text/javascript"></script>
<form name="person">
        <input name="name" value="Иван">
        <input name="surname" value="Иванович">
        <input type="submit" value="Отправить">
</form>
```

script.js:

```
document.forms.person.addEventListener('submit', (e)=>{
    e.preventDefault();
    let formData = new  FormData(document.forms.person);
    // отослать
    let xhr = new XMLHttpRequest();
    xhr.open("POST", "/");
    xhr.send(formData);
});
```

ОТПРАВКА ФОРМЫ НА КЛИЕНТ

Вариант 3:

index.html:

```
<script src ="script.js" type= "text/javascript"></script>
<form name="person">
        <input name="name" value="Иван">
        <input name="surname" value="Иванович">
        <input type="submit" value="Отправить">
</form>
```

script.js:

```
document.forms.person.addEventListener('submit', (e)=>{
    e.preventDefault();
    let msg = $('form[name="person"]').serialize();
    $.ajax({
        type: 'POST',
        url: '/',
        data: msg,
        success: function(data) {
            $('body').html(data);
        }
    });
});
```

ОТПРАВКА ФОРМЫ НА КЛИЕНТ

Дискретный режим чтения потока:
server.js (часть функции обработчика http запросов на сервере):

```
if (request.method === 'POST') {
        var strData = '';

        request.on('readable', function(){
                var chunk;
                while((chunk = request.read()) !== null) {
                        strData+=chunk;
                }
        });
        request.on('end', function(){
                console.log(strData);
                console.log(decodeURI(strData));
                response.writeHead(200);
                response.end('Data save!');
        });
}
```

ОТПРАВКА ФОРМЫ НА КЛИЕНТ

Непрерывный режим чтения потока:
server.js (часть функции обработчика http запросов на сервере):

```
if (request.method === 'POST') {
        var strData = '';

        request.on('data', function(chunk){
                strData += chunk;
        });
        request.on('end', function(){
                console.log(strData);
                console.log(decodeURI(strData));
                response.writeHead(200);
                response.end('Data save!');
        });
}
```

ОТПРАВКА ФАЙЛА НА КЛИЕНТ

index.html:

```
<input type="file" id="myFile" name="myFile" />
```

script.js:

```
myFile.addEventListener('change', takeFile, false);

function takeFile(e){
	var file = e.target.files[0];

	$.ajax({
		type: ‘POST',
		url: file.name,
		data: file,
		processData: false,
		success: function(response) {
			alert(response); //ответ от сервера
		}
	});
}
```

ДИСКРЕТНЫЙ РЕЖИМ ЧТЕНИЯ ИЗ ПОТОКА

```
if (request.method === 'POST') {
        var pathname = url.parse(request.url).path;
        pathname = pathname.substring(1, pathname.length);
        var newFileStream = fs.createWriteStream(pathname);

        request
                .on('readable', function(){
                        var chunk;
                        while((chunk = request.read()) !== null) {
                                newFileStream.write(chunk);
                        }
                })
                .on('end', function(){
                        newFileStream.end();
                        response.writeHead(200);
                        response.end();
                });
}
```

НЕПРЕРЫВНЫЙ РЕЖИМ ЧТЕНИЯ ИЗ ПОТОКА

```
if (request.method === 'POST') {
        var pathname = url.parse(request.url).path;
        pathname = pathname.substring(1, pathname.length);
        var newFileStream = fs.createWriteStream(pathname);

        request
                .on('data', function(chunk){
                        newFileStream.write(chunk);
                })
                .on('end', function(){
                        newFileStream.end();
                        response.writeHead(200);
                        response.end();
                });
}
```

ОБЪЕДИНЕНИЕ ПОТОКОВ

```
if (request.method === 'POST') {
        var pathname = url.parse(request.url).path;
        pathname = pathname.substring(1, pathname.length);
        var newFileStream = fs.createWriteStream(pathname);

        newFileStream.on('close', function(){
                response.writeHead(200);
                response.end();
        });

        request.pipe(newFileStream);
}
```

ОБЪЕДИНЕНИЕ ПОТОКОВ

```
if(request.method === 'GET') {
        var pathname = url.parse(request.url).path;
        if(pathname == '/')
                pathname = '/index.html';
        var extname = path.extname(pathname);
        var mimeType = mimeTypes[extname];
        pathname = pathname.substring(1, pathname.length);
        response.writeHead(200, {'Content-Type': mimeType});
        var newFileStream = fs.createReadStream(pathname);
        newFileStream.pipe(response);
        newFileStream.on('error', function(err){
                console.log('Could not find or open file '+
                        pathname + ' for reading\n');
        });
}
```

ТРАНСФОРМАЦИЯ ПОТОКОВ

```
const stream = require('stream');
const util = require('util');
class ToUpperStream extends stream.Transform {
        constructor() {
                super();
        }
        _transform(chunk, encoding, callback) {
                this.push(chunk.toString().toUpperCase());
                callback();
        }
}
module.exports = ToUpperStream;
```

ТРАНСФОРМАЦИЯ ПОТОКОВ

```
const ToUpperStream = require('./upperStream');
const tUS = new ToUpperStream();

//Пример №1
tUS.on('data', chunk =>
console.log(chunk.toString()));
tUS.write('Hello W');
tUS.write('orl');
tUS.end('d!');

//Пример №2
process.stdin.pipe(tUS).pipe(process.stdout);
```