Современные платформы прикладной разработки

SIGNALR

signalR

ОБЩАЯ ИНФОРМАЦИЯ

ASP.NET Core SignalR — это библиотека с открытым исходным кодом, которая упрощает добавление веб-функций **в реальном времени** в приложения.

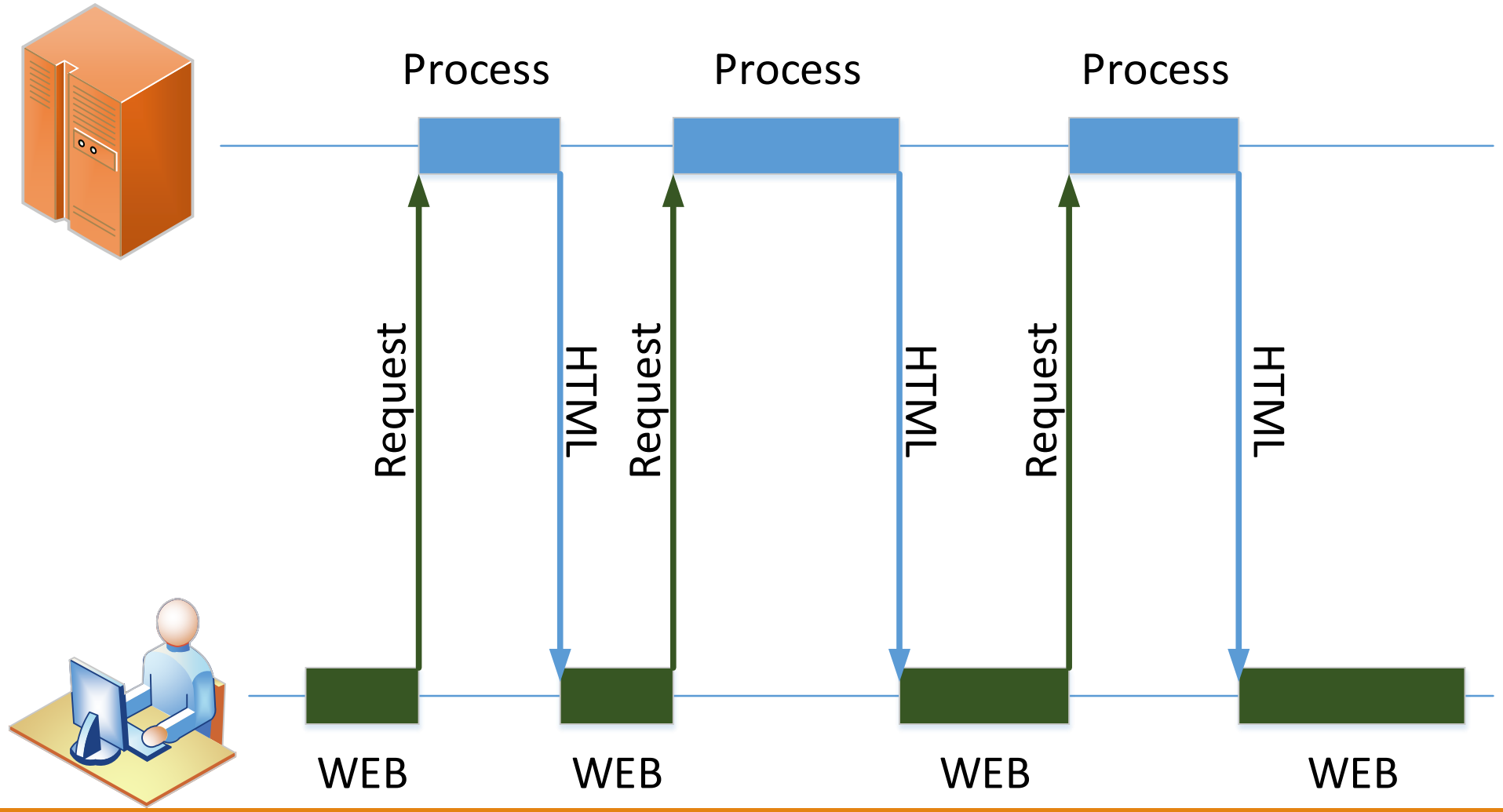
Веб-функции в режиме реального времени позволяют коду на стороне сервера мгновенно передавать контент клиентам.

Применение signalR

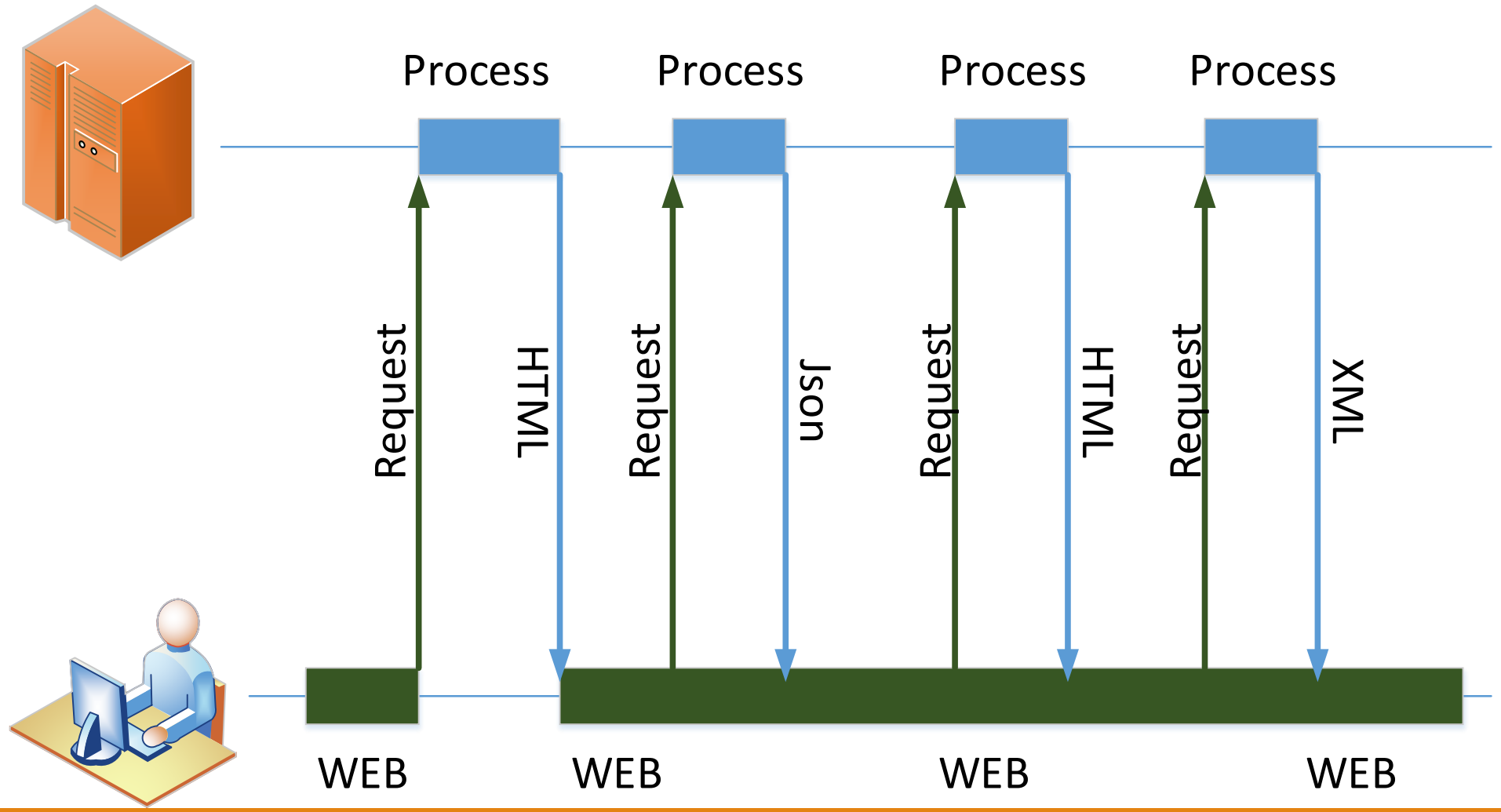
- Приложения, требующие частых обновлений с сервера. Примерами являются игры, социальные сети, голосования, аукционы, карты и приложения GPS.
- Панели мониторинга и приложения для мониторинга. Примеры включают информационные панели компании, мгновенные обновления продаж или оповещения о поездках.
- Совместные приложения. Приложения для доски и программное обеспечение для групповых собраний являются примерами приложений для совместной работы.
- Приложения, которые требуют уведомлений. Социальные сети, электронная почта, чат, игры, оповещения о поездках и многие другие приложения используют уведомления.

Протокол HTTP

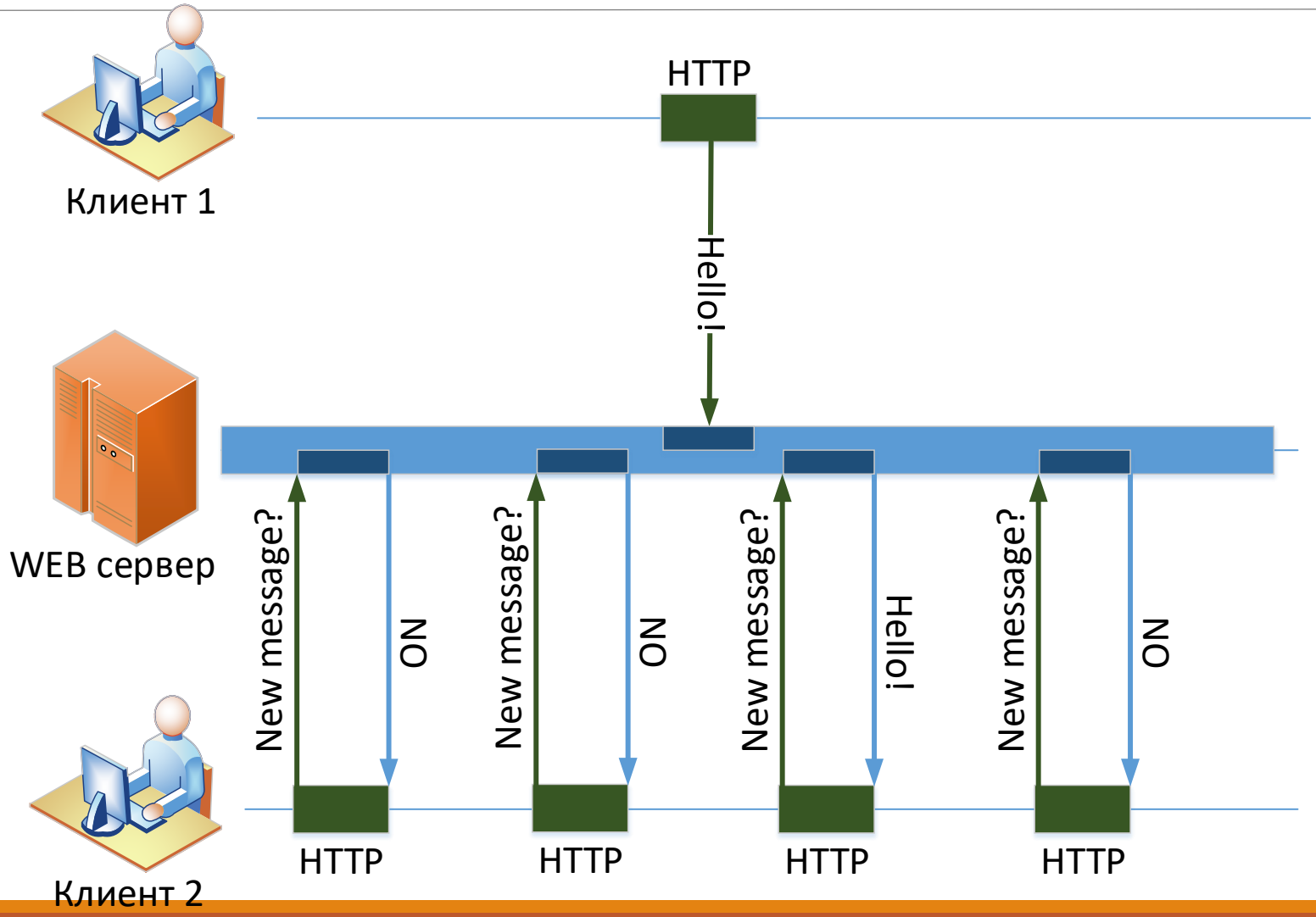
Протокол HTTP



Ajax


Process
Process
Process
Process
Request
HTML
Request
json
Request
HTML
Request
XML
WEB
WEB
WEB
WEB

Polling


HTTP
Клиент 1
Hello!
WEB сервер
New message?
ON
New message?
ON
New message?
Hello!
New message?
ON
HTTP
HTTP
HTTP
HTTP
Клиент 2

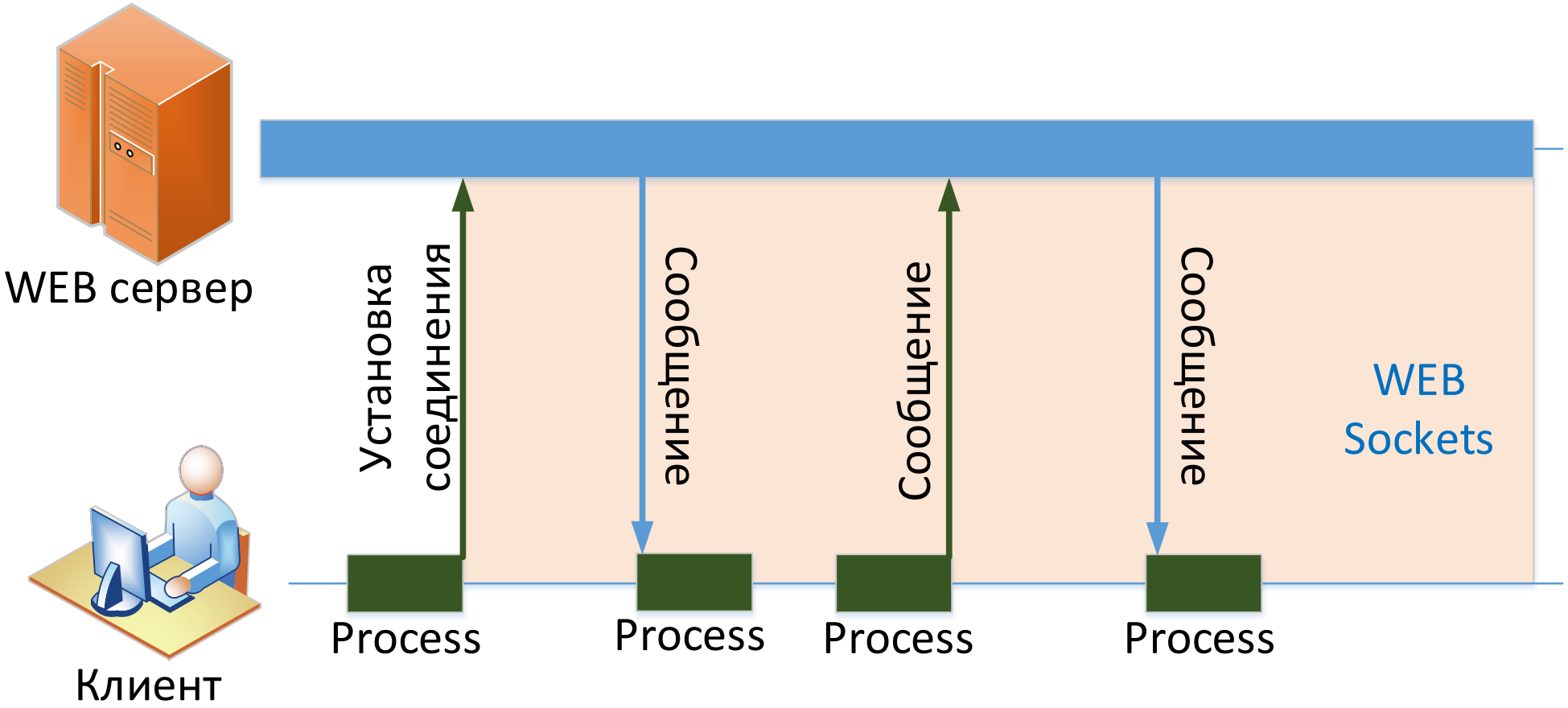
signalR

ВЗАИМОДЕЙСТВИЕ В РЕАЛЬНОМ ВРЕМЕНИ

WebSockets

WebSocket — протокол полнодуплексной связи поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и веб-сервером в режиме реального времени.

WebSockets


WEB сервер
Клиент
Установка соединения
Сообщение
Сообщение
Сообщение
WEB Sockets
Process
Process
Process
Process

WebSockets

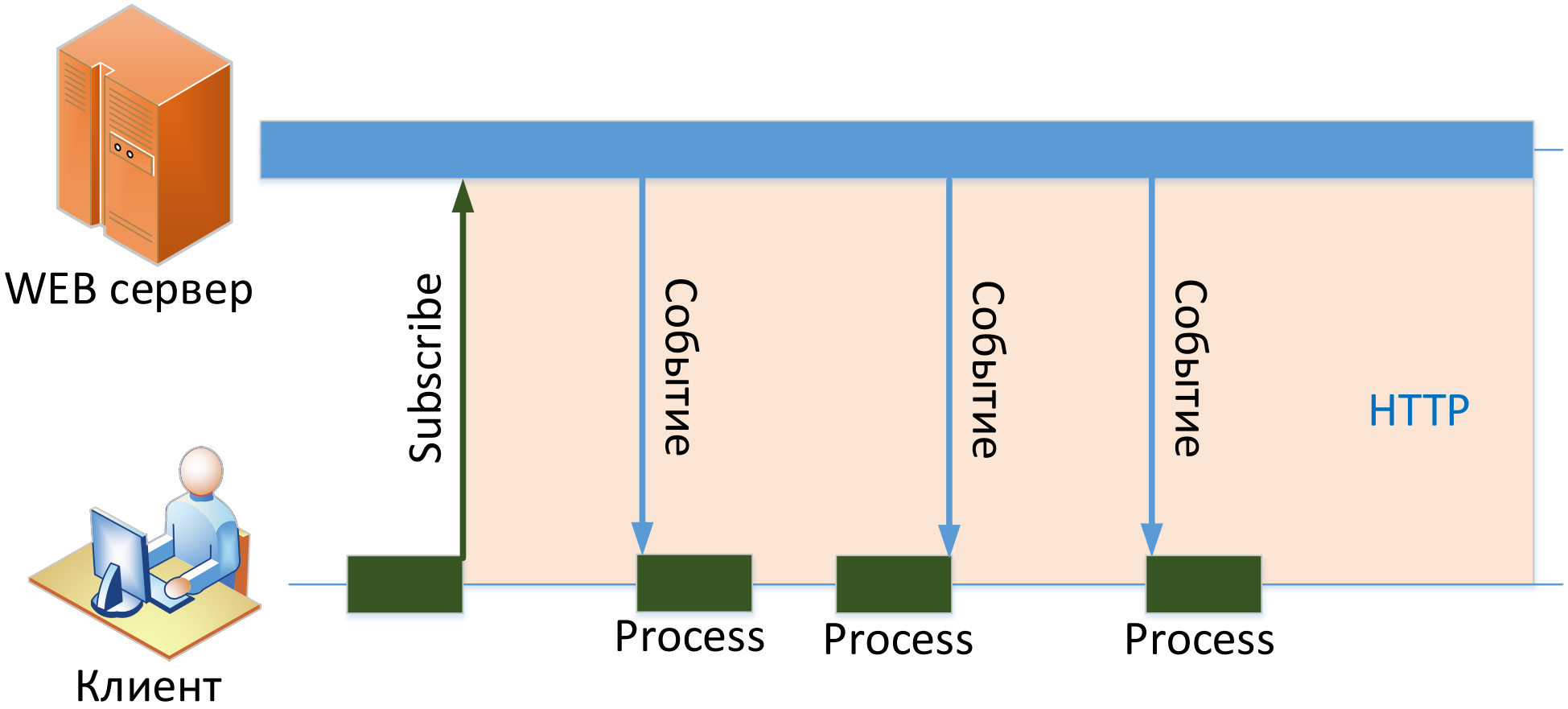
```
var ws = new WebSocket("ws://localhost:9998/echo");
ws.onopen = function () {
    // Web Socket is connected, send data using send()
    ws.send("Message to send");
    alert("Message is sent...");
};
ws.onmessage = function (evt) {
    var received msg = evt.data;
    alert("Message is received...");
};
ws.onclose = function () {
    // WebSocket is closed.
    alert("Connection is closed...");
};
```

Server-Sent Events

SSE (*Server-Sent Events* — «события, посылаемые сервером») представляет собой технологию отправки уведомлений от сервера к веб-браузеру в виде DOM-событий.

Технология Server-Sent Events сейчас стандартизируется как часть HTML5 организацией W3C.

Server-Sent Events



Server-Sent Events

Связь осуществляется через протокол HTTP. Отличие состоит в том, что в ответе сервера указывается тип данных text/event-stream. Это означает, что соединение должно оставаться открытым, т.к. будет использоваться сервером для отправки непрерывного потока событий или сообщений.

Server-Sent Events

Стандарт SSE широко используется для посылки сообщений об обновлениях или для посылки непрерывных потоков данных браузеру клиента. Он спроектирован для улучшения кросс-браузерного вещания посредством JavaScript API под названием **EventSource**; с его помощью клиент задает URL для получения интересующего его потока событий.

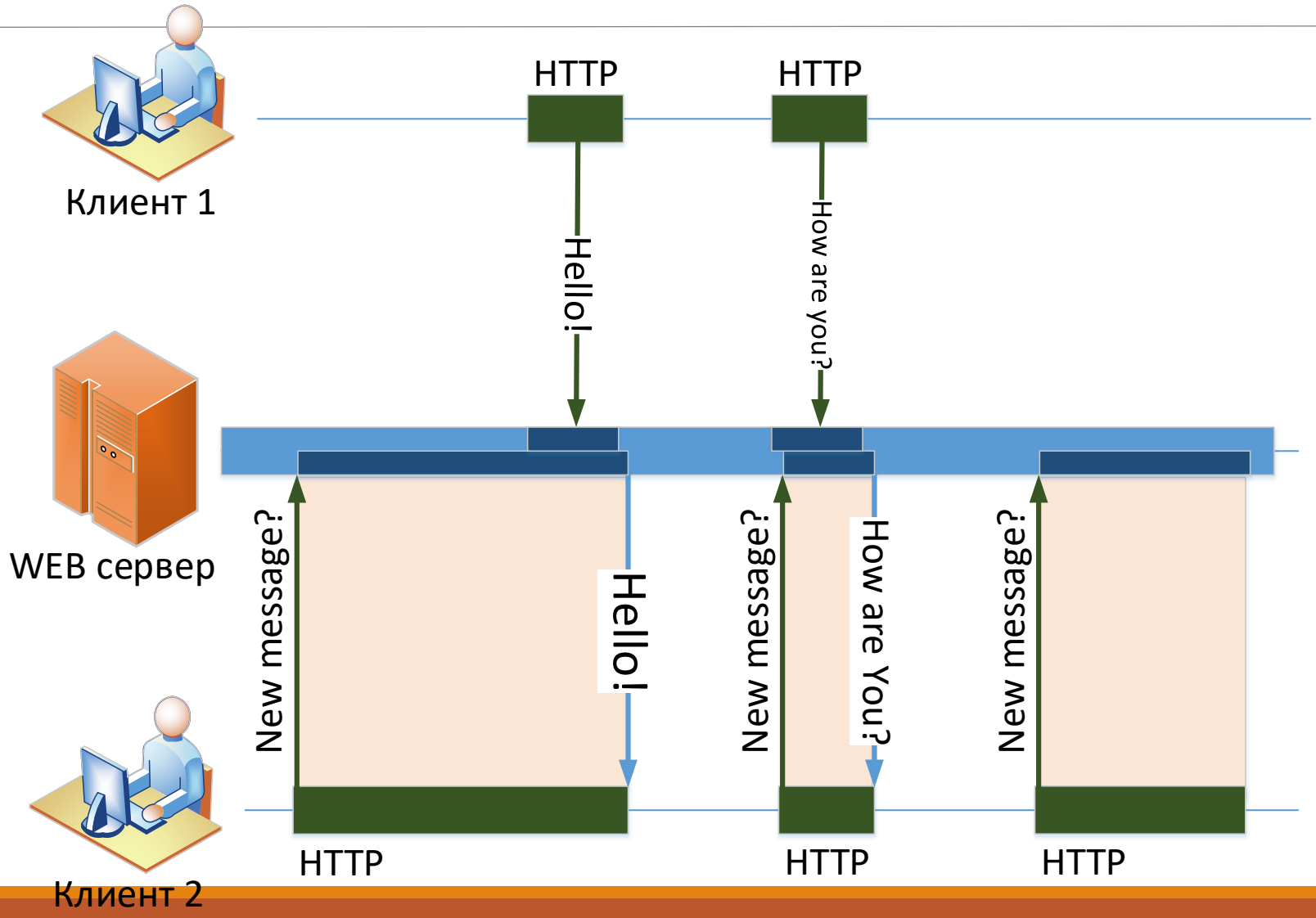
Server-Sent Events

```
var source = new EventSource("/getevents");
source.onmessage = function(event) {
    alert(event.data);
};
```

Long Polling

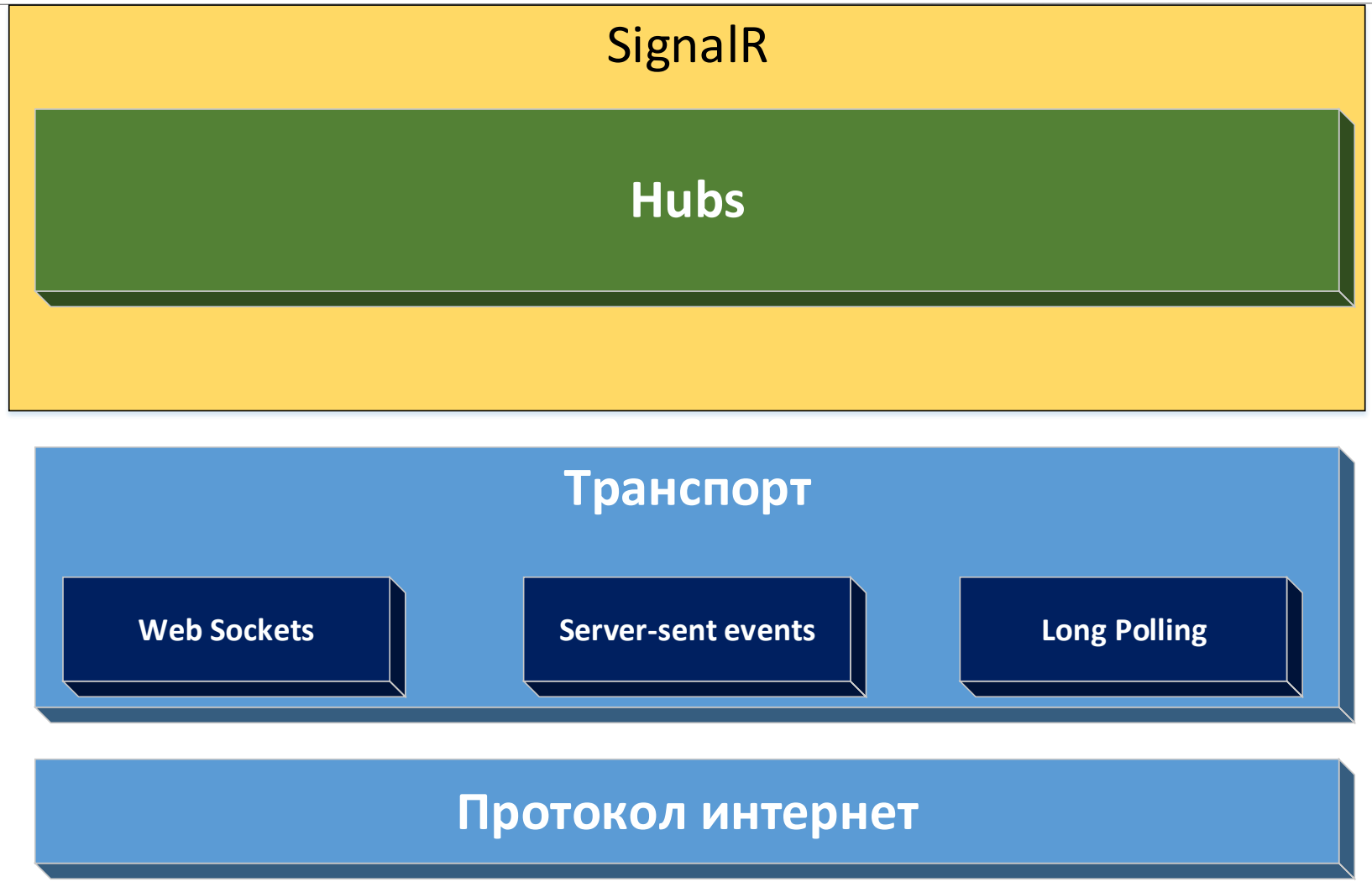
Long Polling - «Длинные опросы», или «Очередь ожидающих запросов

Long Polling



Введение в SignalR

Уровни абстракции SignalR



SignalR: Hubs

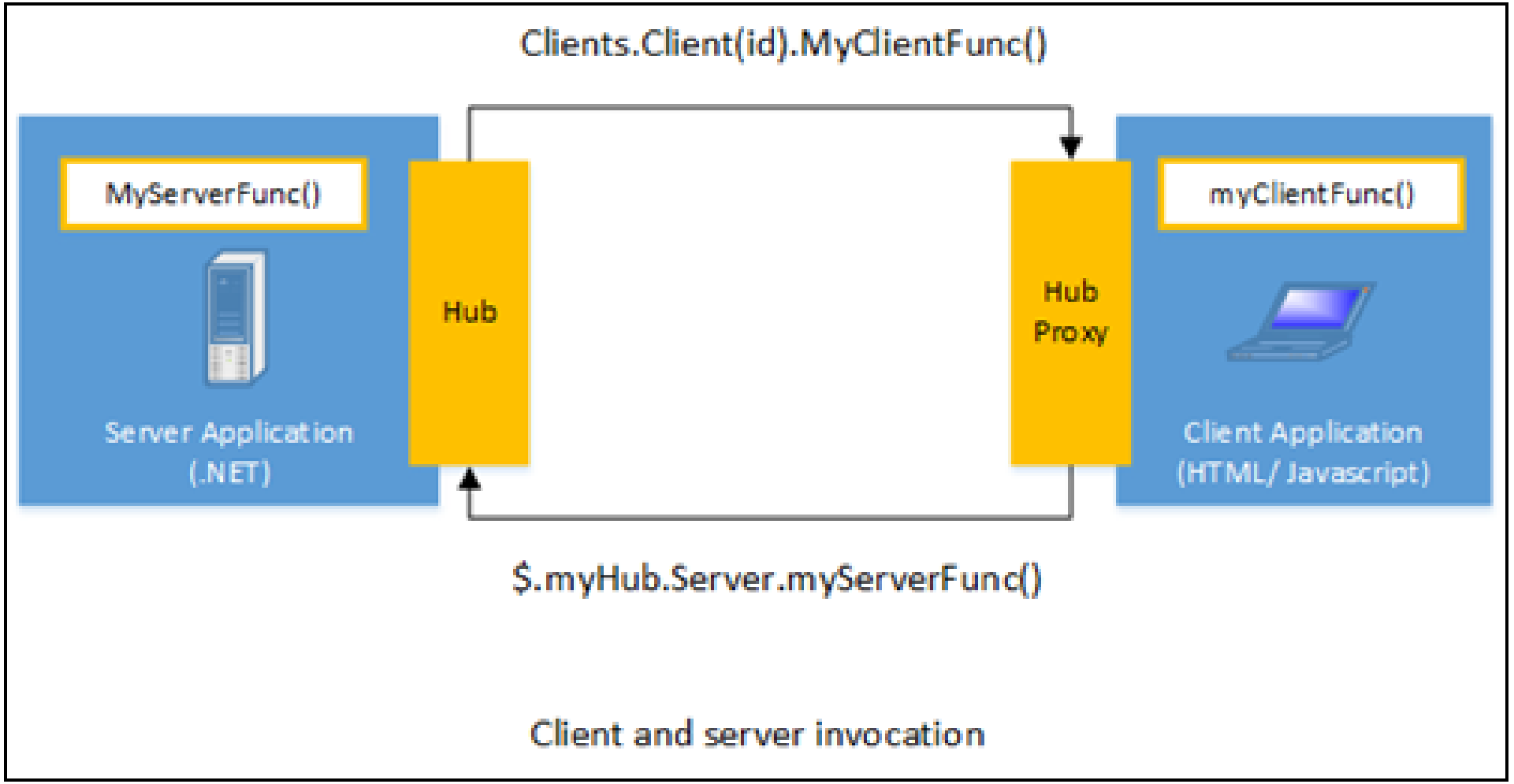
РЕАЛИЗАЦИЯ НА СТОРОНЕ СЕРВЕРА

SignalR: Hubs

Hubs – это конвейер высокого уровня, который позволяет клиенту и серверу вызывать методы друг друга.

Это модель двунаправленного RPC, позволяющая клиенту вызывать методы на сервере и наоборот, серверу вызывать методы на стороне клиента.

Вызов удаленных процедур



Создание класса Hub

```
using Microsoft.AspNetCore.SignalR;

namespace SPPR.Hubs
{
    public class ChatHub : Hub
    {
        public async Task SendMessage(string user, string message)
        {
            await Clients.All.SendAsync("ReceiveMessage", user, message);
        }
    }
}
```

```
public interface IChatClient
{
    Task ReceiveMessage(string user, string message);
}
```

```
public class ChatHub : Hub<IChatClient>
{
    public async Task SendMessage(string user, string message)
    {
        await Clients.All.ReceiveMessage(user, message);
    }
}
```

Hub, методы и свойства

Прием сообщений

Сообщения от клиента передаются посредством вызова открытых методов класса:

```
public class MyHub : Hub
{
    public async Task Hello() {. . .}
    public async Task Join(string name){. . .}
    public async Task<int> Sum(int a, int b){return a + b;}
}
```

Прием сообщений

Любой общедоступный метод Hub доступен извне.
Это может представлять собой угрозу безопасности приложения

Передача сообщений клиентам

```
public void Hello(string msg)
        {
            Clients.All.hello(msg);
        }
```

Передача сообщений клиентам

```
Clients.AllExcept(connections)
Clients.Caller
Clients.Connection(connectionId)
Clients.Others
Clients.Group(groupName, excludeConnectionIds)
Clients.User(userName)
```

Свойство Context

Context.ConnectionId - получает уникальный идентификатор для подключения, назначенный SignalR. Для каждого подключения существует один идентификатор подключения.

Context.UserIdentifier - получает идентификатор пользователя. По умолчанию SignalR использует ClaimTypes.NameIdentifier из ClaimsPrincipal, связанного с подключением, в качестве идентификатора пользователя.

Пример отправки сообщений

```
public Task SendMessage(string user, string message)
{
    return Clients.All.SendAsync("ReceiveMessage", user, message);
}

public Task SendMessageToCaller(string user, string message)
{
    return Clients.Caller.SendAsync("ReceiveMessage", user, message);
}

public Task SendMessageToGroup(string user, string message)
{
    return Clients.Group("SignalR Users").SendAsync("ReceiveMessage", user, message);
}
```

Получение ConnectionId

```
Context.ConnectionId
```

Обработка событий подключения

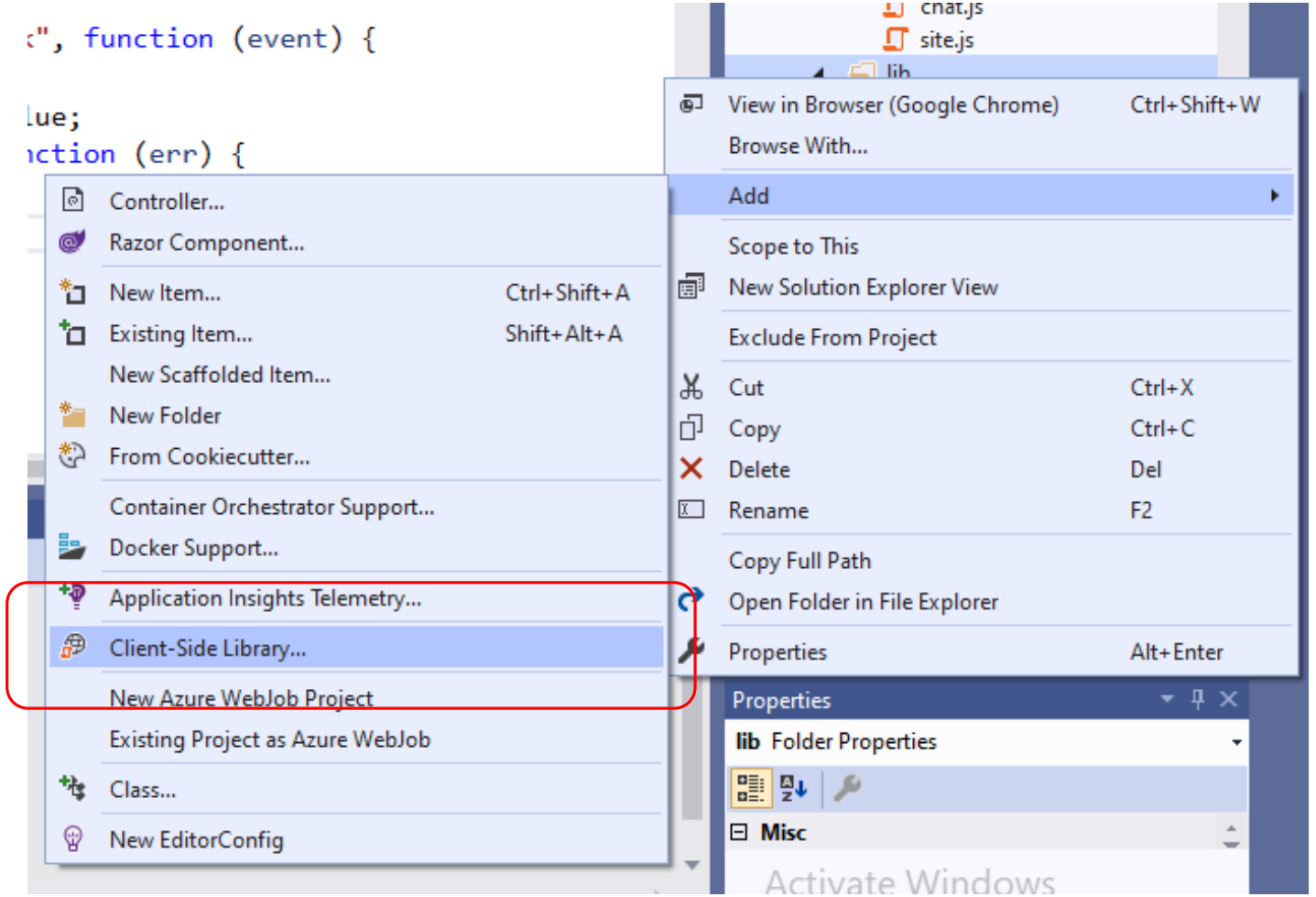
```
public virtual Task OnConnected();
public virtual Task OnDisconnected(bool stopCalled);
public virtual Task OnReconnected();
```

Работа с группами

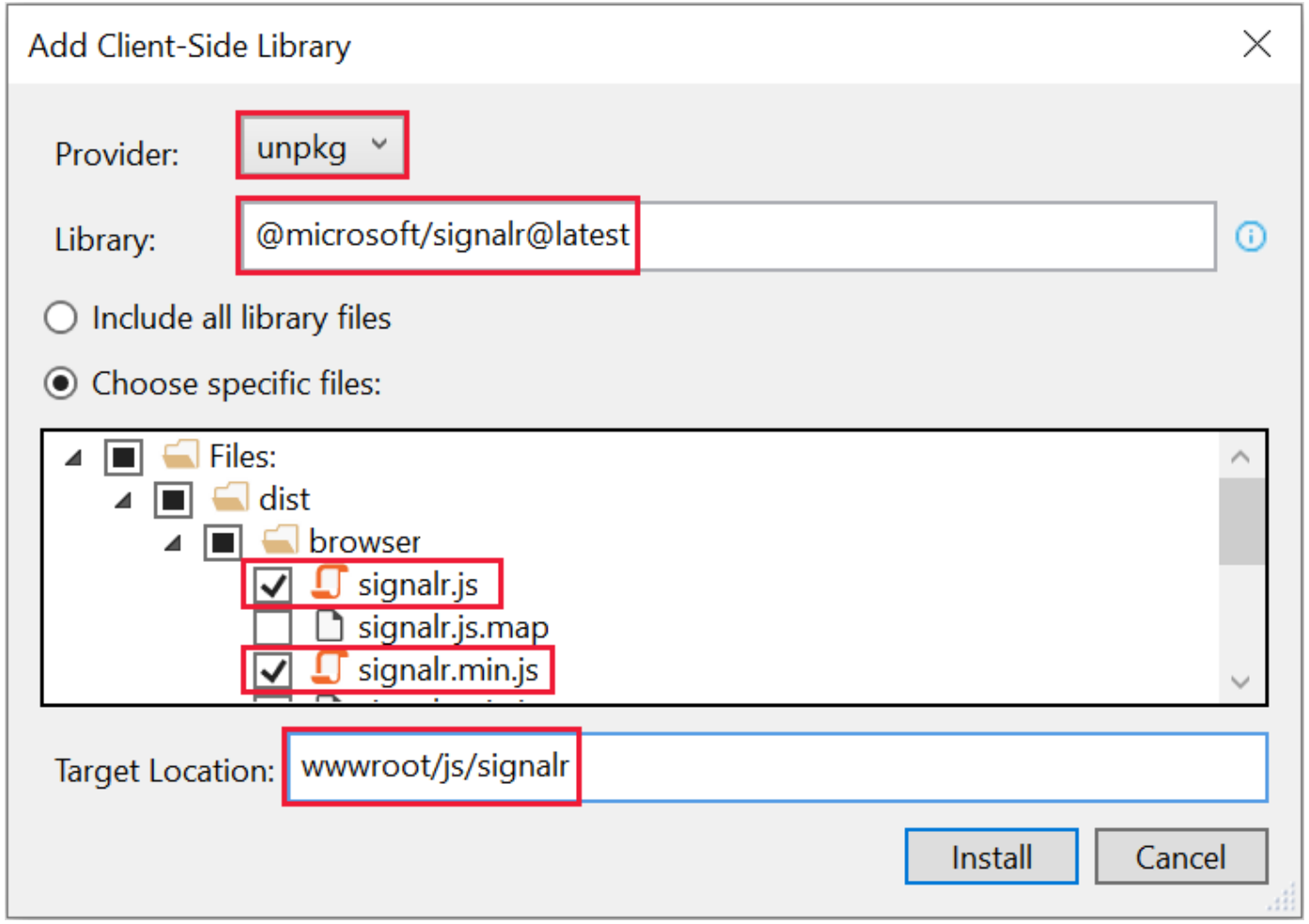
```
Groups.Add(Context.ConnectionId, groupName)

Clients.Group("game1", null).SendAsync(message)
```

Использование SignalR с JavaScript

Добавление клиентской библиотеки



Добавление клиентской библиотеки



Изменения в классе Program.cs

```
builder.Services.AddSignalR();

...

app.MapHub<ChatHub>("/chathub");
```

Подключение скриптов на страницу

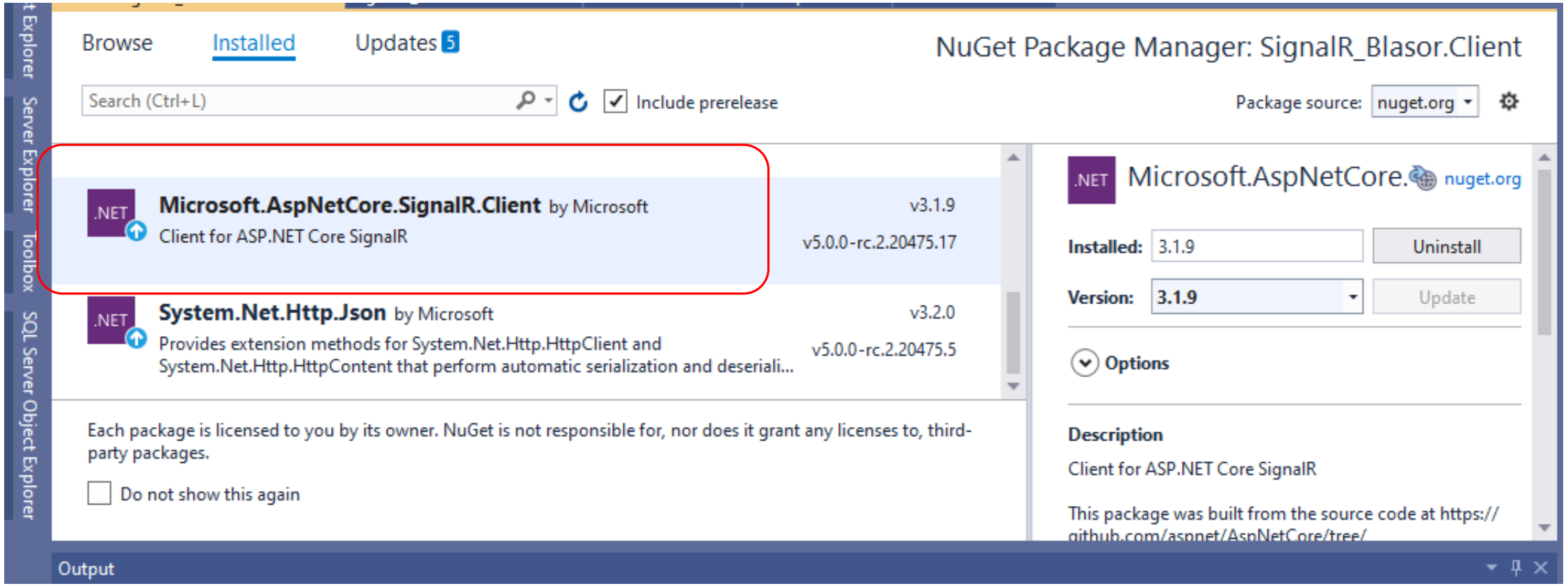
```
@section Scripts
{
    <script
   src="~/lib/signalr/dist/browser/signalr.js"></script>
    <script src="~/js/chat.js"></script>
}
```

Подключение к Hub в коде JavaScript

```
"use strict";
// Объект соединения с сервером
var connection = new signalR.HubConnectionBuilder().withUrl("/chathub").build();
// функция "ReceiveMessage" клиента
connection.on("ReceiveMessage", function (user, message) {. . .});
// Соединиться с сервером
connection.start().catch(function (err) { return console.error(err.toString());});
// Вызов функции "SendMessage" сервера
document.getElementById("sendButton").addEventListener("click", function (event) {
    . . .
    connection.invoke("SendMessage", user, message).catch(function (err) {
        return console.error(err.toString());
    });
    event.preventDefault();
});
```

Использование SignalR с Blazor Web Assembly

Добавление NuGet пакета в проект Client



Добавление Hub в проект Server

```
public class TttHub : Hub<TttClient>
{
    static Dictionary<string, GameGroup> groups = new();

    public async Task Register(string name)
    {
        . . .
       await Groups.AddToGroupAsync(Context.ConnectionId, groupId);
       await Clients.Caller.Registered(groupId, player.PlayerStyle);
    }
    . . .

    public async Task PlayerTurn(string groupId, int x, int y)
    {
       . . .
       await Clients.Group(groupId).BoardUpdated(game.Board.Board, game.Board.CurrentTurn);
        if (game.Board.GameComplete)
        {
                await Clients.Group(groupId).GameOver(game.Board.GetWinner());
        }
    }
}
```

Добавление сервиса в проекте Server

```
builder.Services.AddResponseCompression(opts =>
{
    opts.MimeTypes = ResponseCompressionDefaults
                .MimeTypes
                .Concat(new[] { "application/octet-stream" });
});
builder.Services.AddSignalR();
```

Добавление конечных точек в проекте Server

```
app.UseRouting();
app.UseResponseCompression();

app.MapRazorPages();
app.MapControllers();
app.MapFallbackToFile("index.html");
app.MapHub<TttHub>("/ttt");
```

Установка соединения клиента

```
@page "/"
@using Microsoft.AspNetCore.SignalR.Client
@inject NavigationManager NavigationManager
@implements IDisposable
```

Установка соединения клиента

```
HubConnection connection;


protected override async Task OnInitializedAsync()
{

    connection = new HubConnectionBuilder()
        .WithUrl(Navigation.ToAbsoluteUri("/ttt"))
        .WithAutomaticReconnect()
        .Build();
      . . .
    connection.On<GameSell[][],GameCellStyle>("BoardUpdated", (b, d) => BoardUpdated(b, d));
    connection.On<GameOverInfo>("GameOver", i => GameOver(i));
    connection.On<string, GameCellStyle>("Registered",(id, s) => Registered(id, s));

    await connection.StartAsync();
}
public async void Dispose()
    {
        await connection.DisposeAsync();
    }
```

Вызов метода сервера

```
async Task CellClicked(int x, int y)
{
    if (!CanMove()) return;

    await connection.InvokeAsync("PlayerTurn", GroupId, x, y);
}
```

Метод клиента

```
void Registered(string id, GameCellStyle style)
    {
        GroupId = id;
        PlayerStyle = style;
        StateHasChanged();
    }
```

Метод клиента