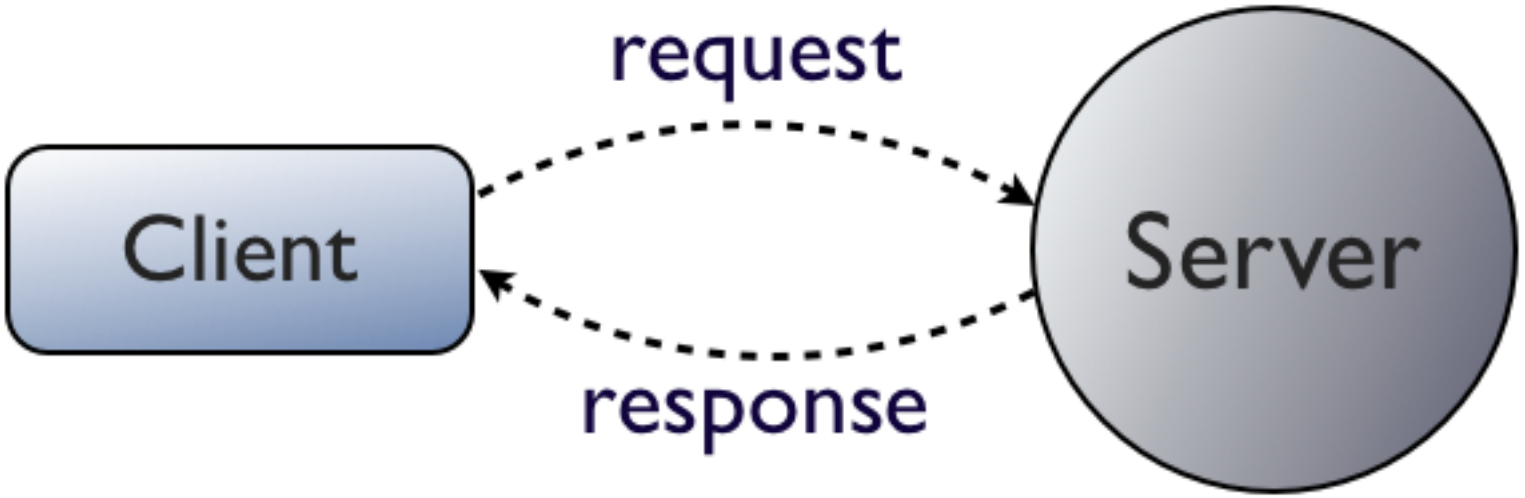
Comunicação Assíncrona na Web

Kiev Gama

kiev.gama@gmail.com

@kievgama

slideshare.net/kievgama



Historicamente, padrões da Web são orientados a comunicação síncrona

Na IoT, o cenário típico é ter dados de tempos em tempos.

Que cenário seria mais adequado?


PUSH


PULL

Pull
Pool
Poll

As principais formas de **comunicação assíncrona** sendo utilizadas na **Web**

AJAX

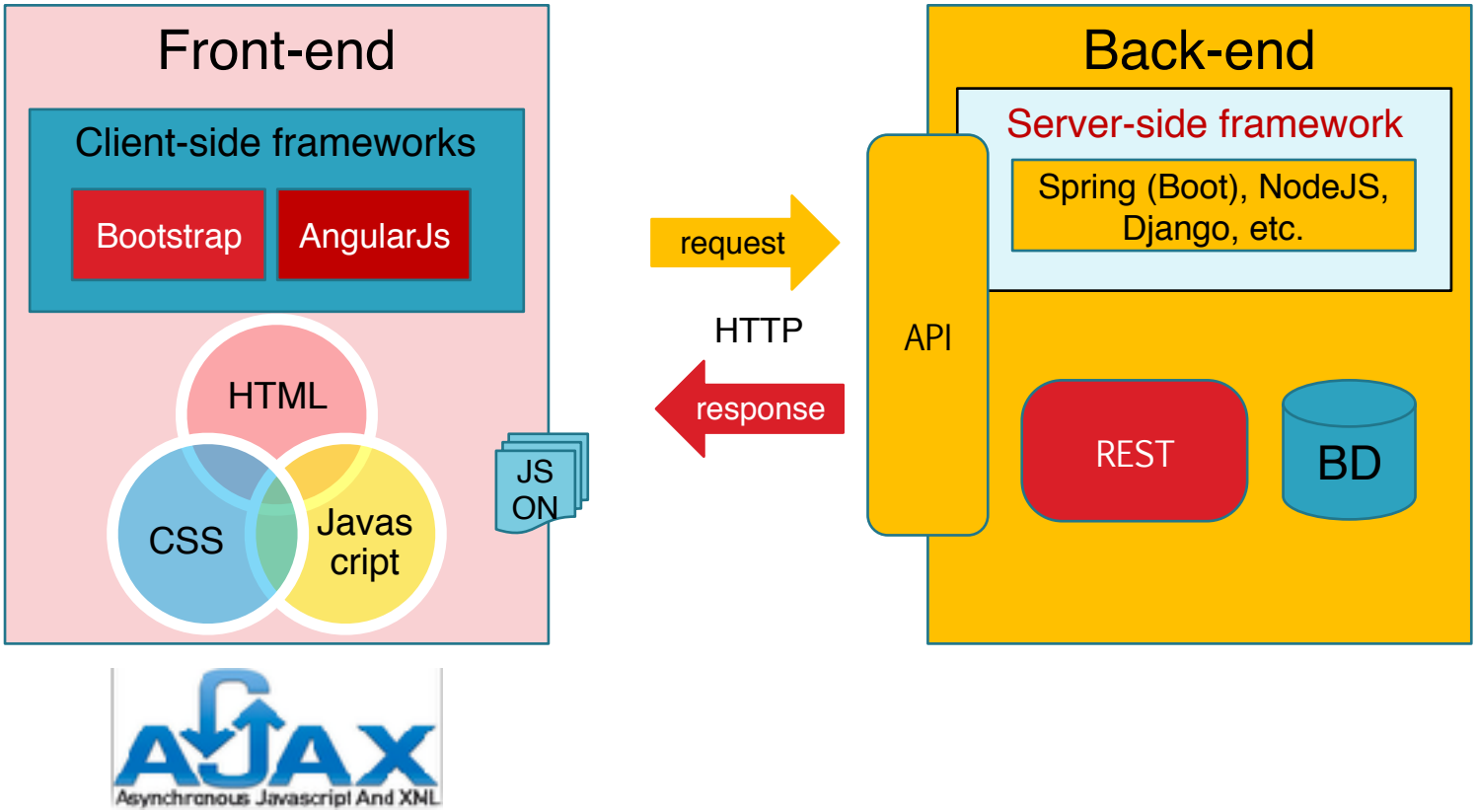
Asynchronous Javascript And XML

GUIs (Graphical User Interface) mais interativas em aplicações

Usa tecnologias abertas utilizadas em browsers

- HTTP
- HTML
- XML (DOM – Document Object Model)
- XMLHttpRequest object
- Javascript

One of the enablers of the « Web 2.0 »


Front-end
Client-side frameworks
Bootstrap
AngularJs
HTML
CSS
Javas
cript
JS
ON
request
HTTP
response
Back-end
Server-side framework
Spring (Boot), NodeJS,
Django, etc.
API
REST
BD

AJAX
Asynchronous Javascript And XML

Comunicação Assíncrona na Web

Abordagens “genéricas” baseadas em HTTP

- Comet
- WebSub(PubsubHubBub)

Mecanismos padronizados no HTML5

- WebSockets
- Server-sent events

Comet

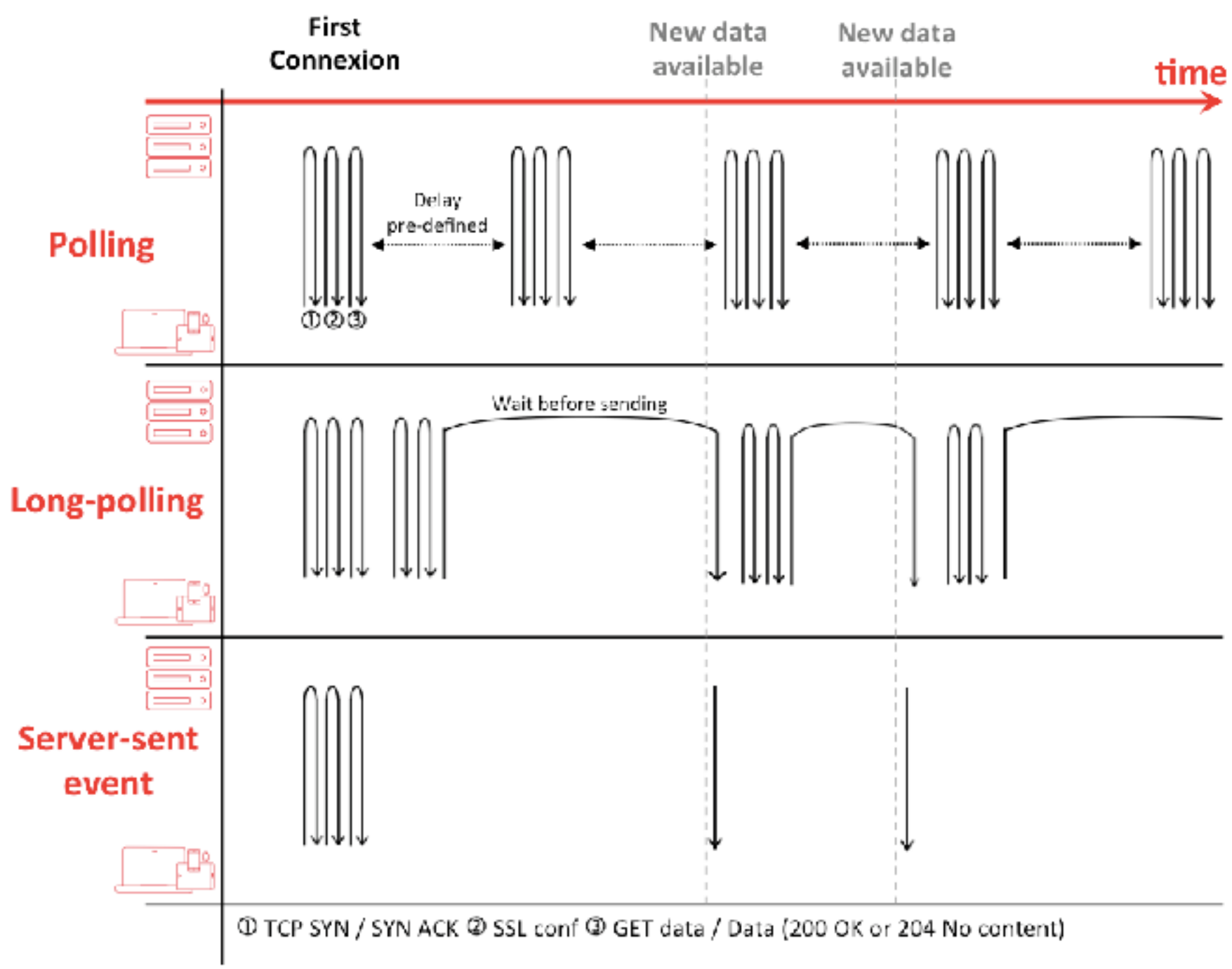
Também conhecido como “AJAX push”

Nome genérico para abordagem em que servidores continuam a enviar dados a clientes através de uma conexão HTTP

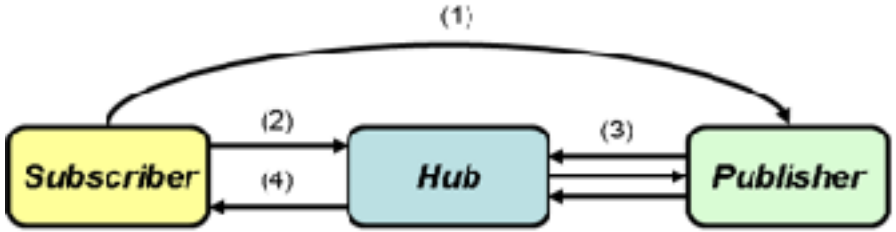
Tipicamente depende de código JavaScript (XMLHTTPRequest)

Diferentes formas de implementação

- Streaming
- Long Polling


First Connexion
New data available
New data available
time
Polling
Delay pre-defined
①②③
Long-polling
Wait before sending
Server-sent event
① TCP SYN / SYN ACK ② SSL conf ③ GET data / Data (200 OK or 204 No content)

WebSub (PubSubHubbub)



Protocolo aberto que utiliza a abordagem publish/subscribe

Baseado em ATOM/RSS e HTTP
https://code.google.com/p/pubsubhubbub/

Foco em atualização de sites de conteúdo

Cliente fala com fornecedor na primeira comunicação

Cabeçalho especial indica endereço do Hub
<link rel="hub" href="http://pubsubhubbub.appspot.com" />

Cliente "assina" atualizações no Hub

Hub notifica cliente das próximas atualizações
(não é mais necessário cliente falar com fornecedor)

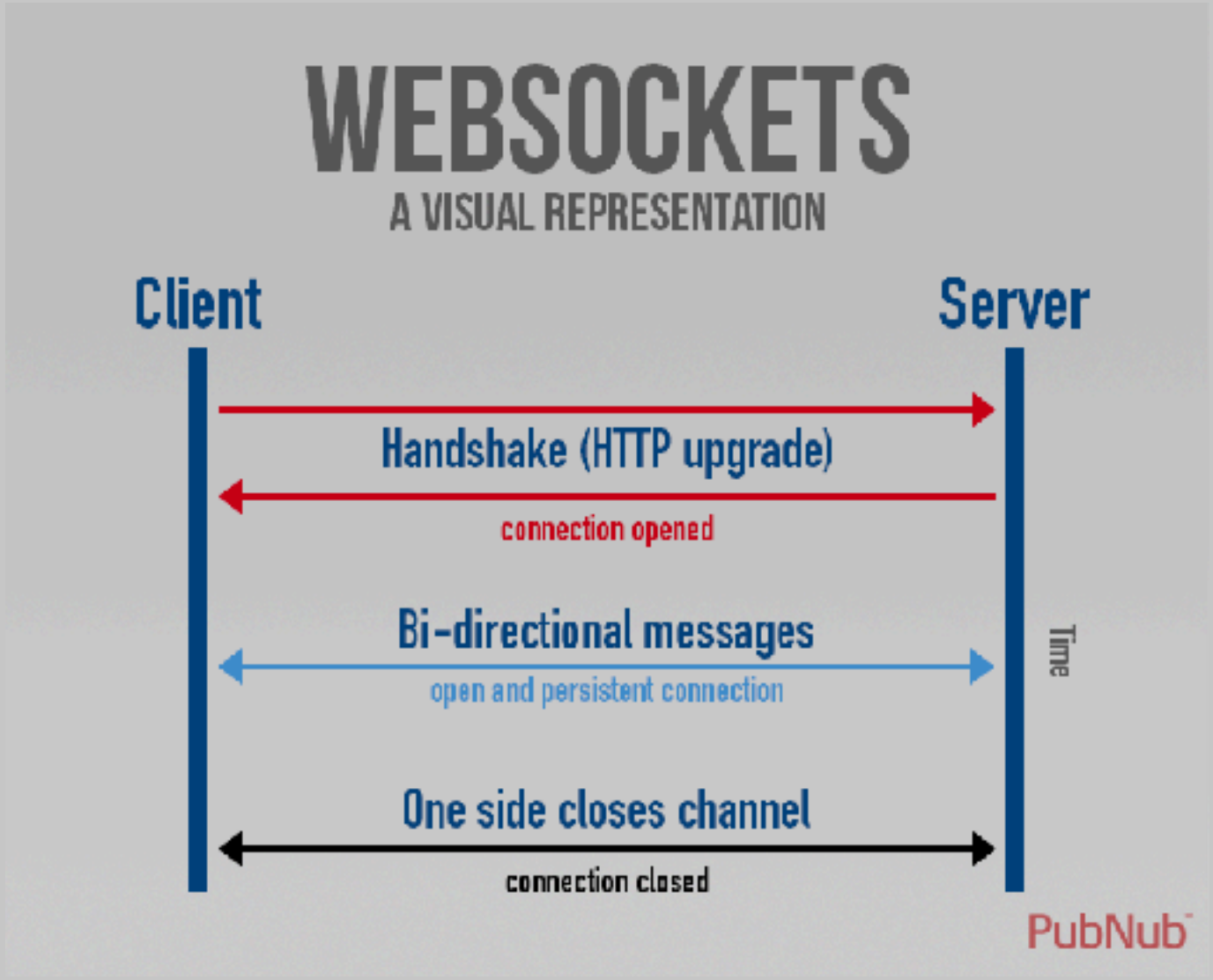
WebSockets

Procotolo fullduplex para comunicação na Web

Alternativa ao HTTP para troca de dados

Definido inicialmente pelo W3C como parte do HTML5
(Controle migrou para o IETF, entidade que cuida dos protocolos padrão da Internet)

HTML5 atualmente padroniza apenas APIs de uso do protocolo
(ws:// e wss://)


WEBSOCKETS
A VISUAL REPRESENTATION
Client
Server
Handshake (HTTP upgrade)
connection opened
Bi-directional messages
open and persistent connection
One side closes channel
connection closed
Time
PubNub

Sockets (os puros, não os Web Sockets)
Funções primitivas

- socket() : cria novo socket
- bind() : vincula o socket a uma porta
- listen() : anuncia que irá aceitar conexões
- accept() : recebe conexões do cliente
- connect() : conecta-se a outro computador
- send() and receive() : efetua a comunicação
- close() : encerra a conexão

Sockets
Cenário de utilização

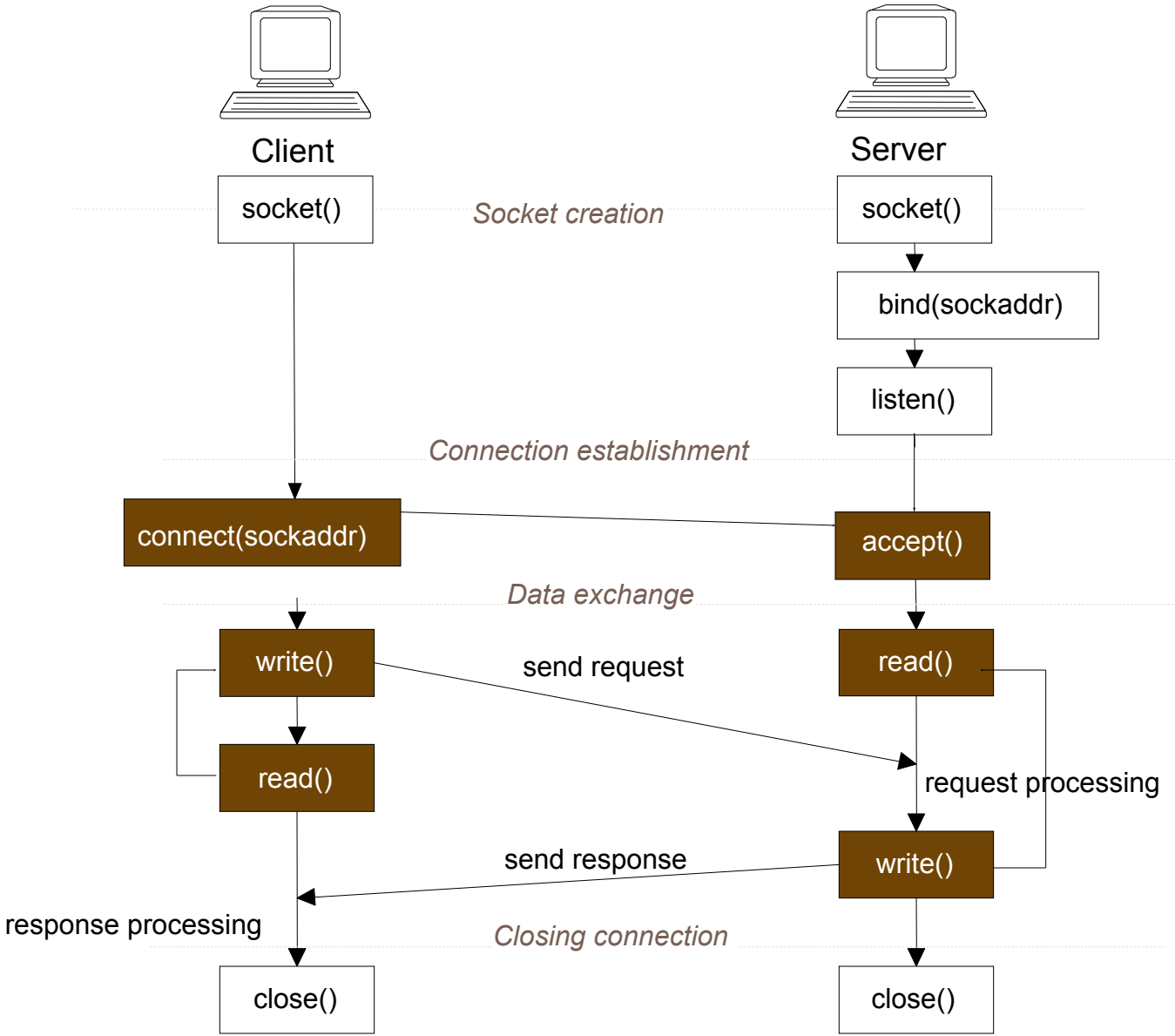
Non-blocking calls

Potentially blocking calls

Exemplo:

client / server,

connected type



WebSocket handshake

Request

```
GET /chat HTTP/1.1
   Host: server.example.com
   Upgrade: WebSocket
   Connection: Upgrade
   Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
   Origin: http://example.com
   Sec-WebSocket-Protocol: chat, superchat
   Sec-WebSocket-Version: 13
```

Response

```
HTTP/1.1 101 Switching Protocols
   Upgrade: WebSocket
   Connection: Upgrade
   Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
   Sec-WebSocket-Protocol: chat
```

Exemplo Cliente/servidor WebSockets

Código servidor (exemplo em Node.js)

```
var WebSocketServer = require('ws').Server;
wss = new WebSocketServer({port: 8080, path: '/myapp'});
wss.on('connection', function(ws) {
    ws.on('message', function(message) {
        console.log('Msg received in server: %s ', message);
    });
    console.log('new connection');
    ws.send('Msg from server');
});
```

Código cliente

```
var connection = new WebSocket('ws://example.org:8080/myapp');

//programação dos eventos (assíncronos)
connection.onopen = function(){
   /*Send a small message to the console once the connection is established */
   console.log('Connection open!’);
   //Envio de mensagens
   connection.send('Hey server, whats up?');
}
connection.onclose = function(){
   console.log('Connection closed');
}
connection.onmessage = function(e){
   var server_message = e.data;
   console.log(server_message);
}
```

Server-sent Events

Usa a EventSource API do HTML5

Envio de dados servidor-cliente (push)

Construído sobre o HTTP

Abordagem mais simples e direta que o WebSocket

Útil quando se necessita de *streams* de dados que são atualizados periodicamente (Posts do twitter, cotações de ações, etc)

Server-sent Events - Exemplo

Código Cliente

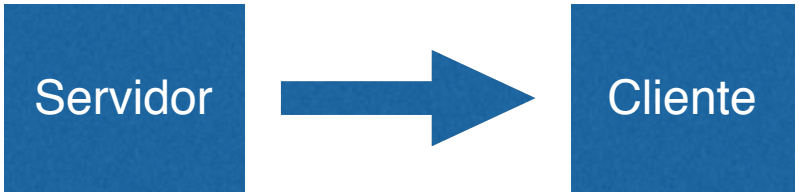
```
var source = new EventSource("demo_sse.php");
source.onmessage = function(event) {
    document.getElementById("result").innerHTML += event.data + "<br>";
};
```

Código Servidor (demo_sse.php)

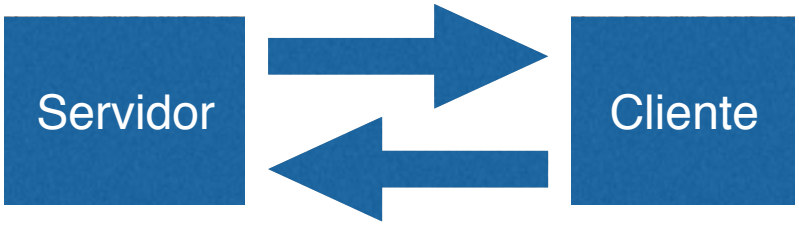
```
<?php
header('Content-Type: text/event-stream');
header('Cache-Control: no-cache');

$time = date('r');
echo "data: The server time is: {$time}\n\n";
flush();
?>
```

Server-sent events
(EventSource)



WebSockets



Outras abordagens

HTTP/2

XMPP

JQuery

PubSubJS

Resumo

Diferentes modelos de comunicação assíncrona na Web

Server-sent events (EventSource) são limitados ao uso em apenas em browsers.

WebSockets possuem suporte nativo em browsers quanto em outros tipos de aplicações. Útil quando se necessita uma conexão full duplex.