



Zaawansowane Techniki WWW (HTML, CSS i JavaScript)

Dr inż. Marcin Zieliński

Środa 15:30 - 17:00 sala: A-1-04

WYKŁAD 8

Wykład dla kierunku: **Informatyka Stosowana II rok**

Rok akademicki: **2015/2016 - semestr zimowy**

Przypomnienie z poprzedniego wykładu

Wprowadzenie do biblioteki jQuery

Pobranie i instalacja jQuery

Pobieramy plik w najnowszej wersji ze strony:

<http://jquery.com/download/>



[jquery-2.1.4.js](#)

[jquery-2.1.4.min.js](#)

```
<!DOCTYPE html>
<html>
<head>
<title>Pierwsza strona z jQuery</title>
<script type="text/javascript" src="jquery-2.1.4.min.js"></script>
</head>
<body>
  <h1>Testujemy bibliotekę jQuery</h1>
</body>
</html>
```

Modyfikacja drzewa DOM



Do bardziej złożonych operacji które możemy wykonywać za pomocą biblioteki jQuery jest dowolna modyfikacja całej struktury drzewa DOM. Mówiliśmy już o jednej metodzie która na to pozwala [.html\(\)](#) . Natomiast metod do wykonywania operacji na drzewie DOM jest dużo więcej:

Modyfikacja drzewa DOM



Do bardziej złożonych operacji które możemy wykonywać za pomocą biblioteki jQuery jest dowolna modyfikacja całej struktury drzewa DOM. Mówiliśmy już o jednej metodzie która na to pozwala [.html\(\)](#) . Natomiast metod do wykonywania operacji na drzewie DOM jest dużo więcej:

```
.append( )
```

```
.appendTo( )
```

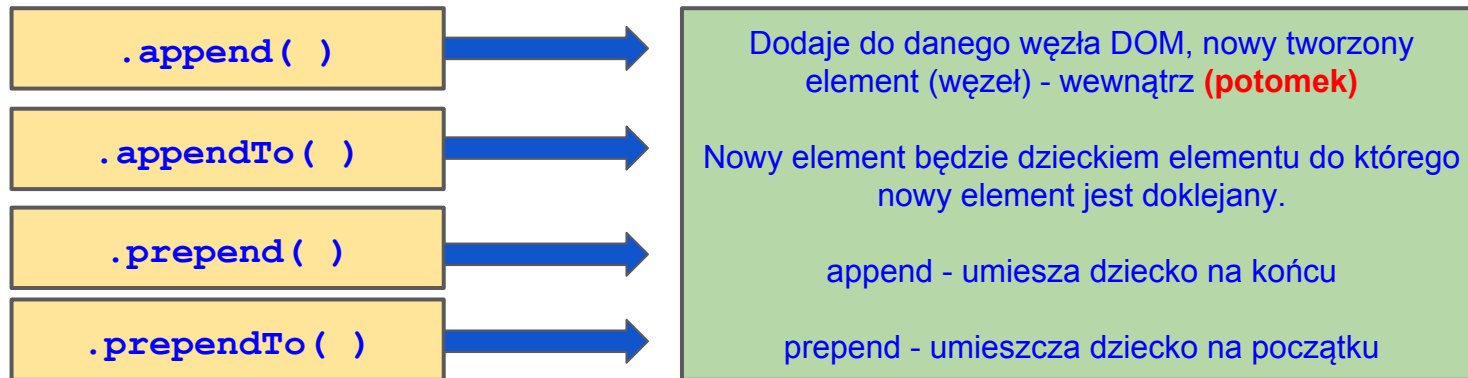
```
.prepend( )
```

```
.prependTo( )
```

Modyfikacja drzewa DOM



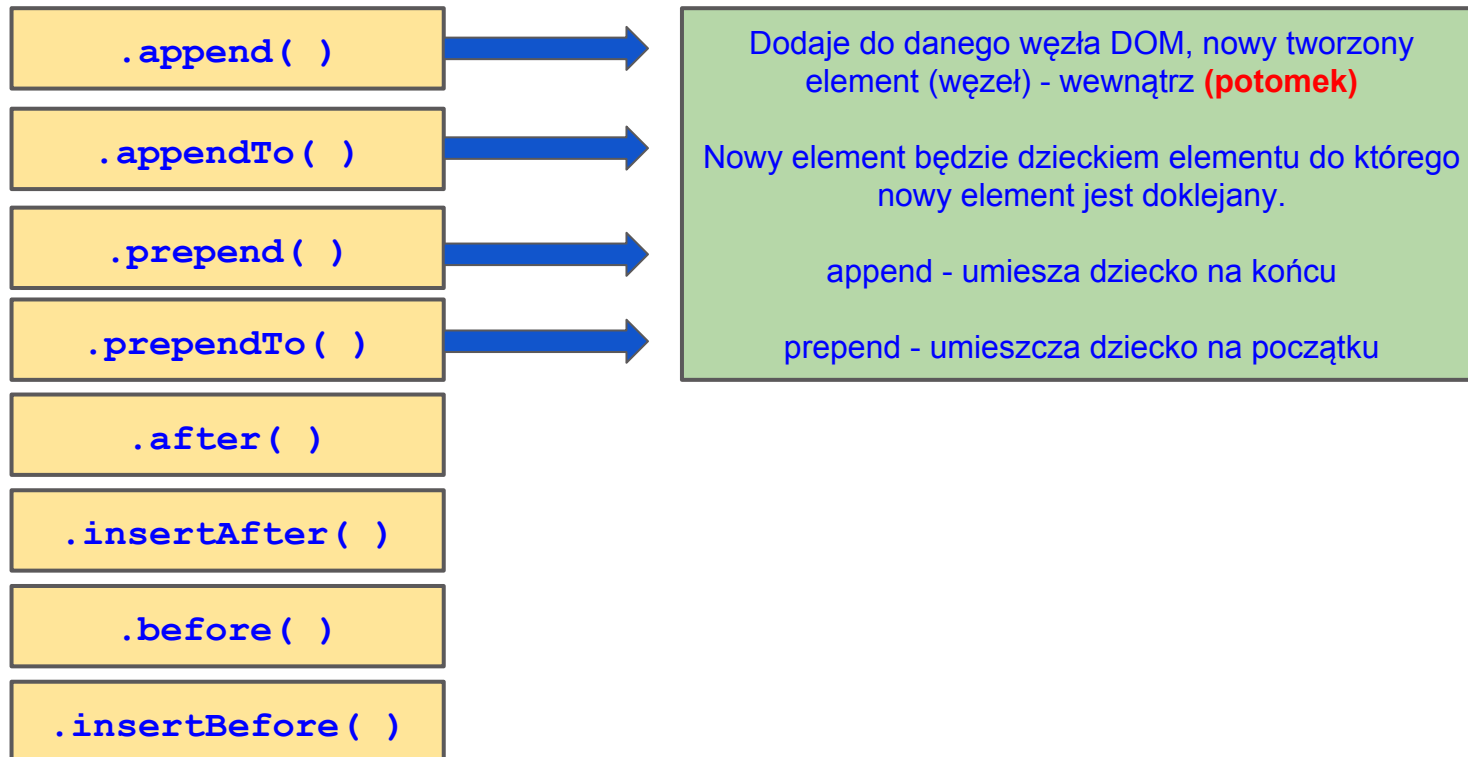
Do bardziej złożonych operacji które możemy wykonywać za pomocą biblioteki jQuery jest dowolna modyfikacja całej struktury drzewa DOM. Mówiliśmy już o jednej metodzie która na to pozwala `.html()`. Natomiast metod do wykonywania operacji na drzewie DOM jest dużo więcej:



Modyfikacja drzewa DOM



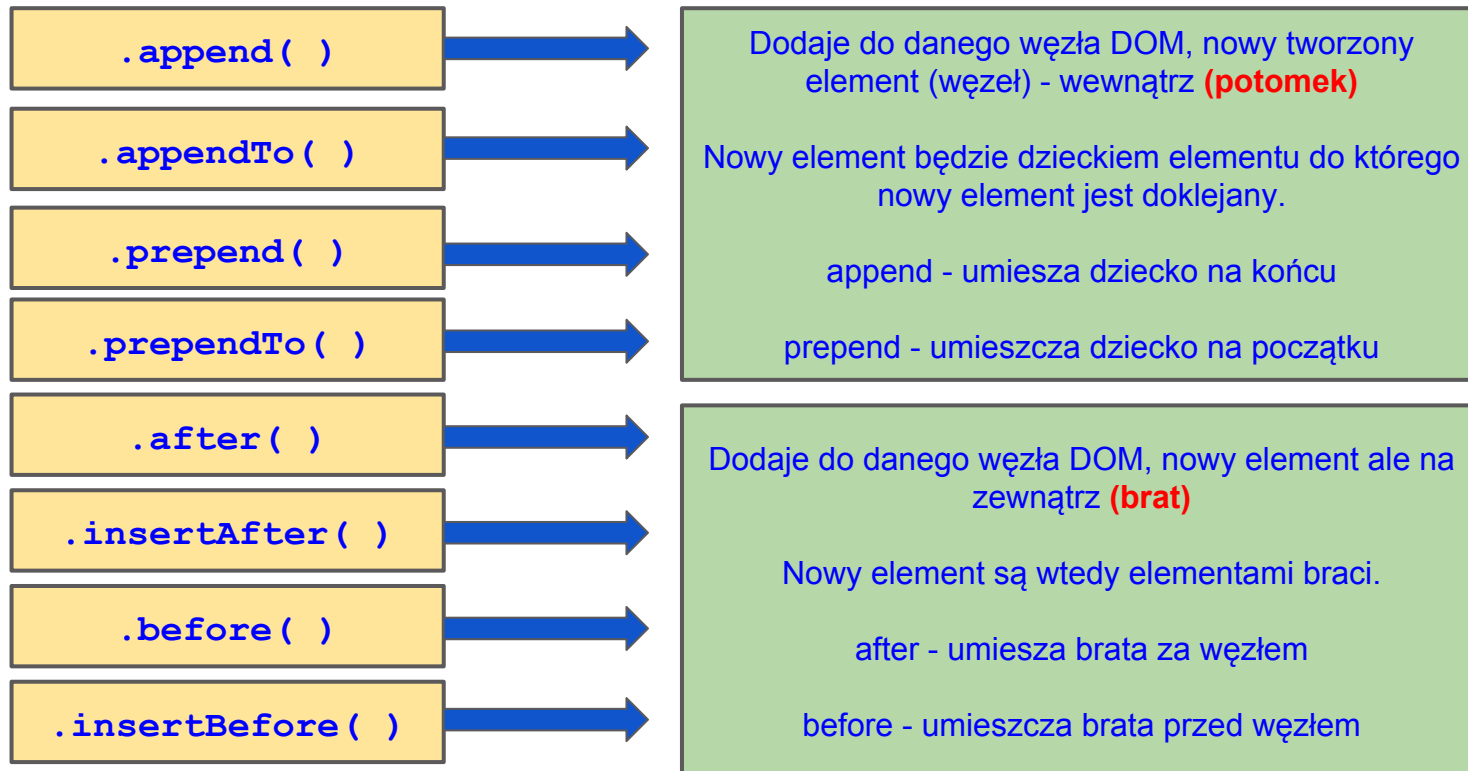
Do bardziej złożonych operacji które możemy wykonywać za pomocą biblioteki jQuery jest dowolna modyfikacja całej struktury drzewa DOM. Mówiliśmy już o jednej metodzie która na to pozwala `.html()`. Natomiast metod do wykonywania operacji na drzewie DOM jest dużo więcej:



Modyfikacja drzewa DOM



Do bardziej złożonych operacji które możemy wykonywać za pomocą biblioteki jQuery jest dowolna modyfikacja całej struktury drzewa DOM. Mówiliśmy już o jednej metodzie która na to pozwala [.html\(\)](#) . Natomiast metod do wykonywania operacji na drzewie DOM jest dużo więcej:



Modyfikacja drzewa DOM



Różnica pomiędzy metodami z postfixem “To”, a bez niego:

`.append ()`

`.appendTo ()`

Modyfikacja drzewa DOM



Różnica pomiędzy metodami z postfixem “To”, a bez niego:

`.append ()`



`.appendTo ()`

W tych metodach najpierw podajemy nazwę węzła który chcemy zmodyfikować, a dopiero później węzeł dodawany.

Modyfikacja drzewa DOM



Różnica pomiędzy metodami z postfixem “To”, a bez niego:

`.append ()`



W tych metodach najpierw podajemy nazwę węzła który chcemy zmodyfikować, a dopiero później węzeł dodawany.

`.appendTo ()`



W tych metodach najpierw podajemy węzeł dodawany, a dopiero jako drugi argument nazwę węzła który chcemy zmodyfikować.

Modyfikacja drzewa DOM



Różnica pomiędzy metodami z postfixem “To”, a bez niego:

`.append()`



W tych metodach najpierw podajemy nazwę węzła który chcemy zmodyfikować, a dopiero później węzeł dodawany.

```
$(function() {  
    $('body').append('<div></div>');  
});
```

`.appendTo()`



W tych metodach najpierw podajemy węzeł dodawany, a dopiero jako drugi argument nazwę węzła który chcemy zmodyfikować.

```
$(function() {  
    $('<div>  
        </div>').appendTo('body');  
});
```

Modyfikacja drzewa DOM



Różnica pomiędzy metodami z postfixem “To”, a bez niego:

`.append()`



W tych metodach najpierw podajemy nazwę węzła który chcemy zmodyfikować, a dopiero później węzeł dodawany.

```
$(function() {  
    $('body').append('<div></div>');  
});
```

1. Wybranie elementu <body> z drzewa DOM.
2. Wywołanie metody append().
3. Utworzenie węzła <div>.
4. Dowiązanie elementu <div> jako dziecka.

`.appendTo()`



W tych metodach najpierw podajemy węzeł dodawany, a dopiero jako drugi argument nazwę węzła który chcemy zmodyfikować.

```
$(function() {  
    $('<div>  
        </div>').appendTo('body');  
});
```

1. Utworzenie węzła <div>.
2. Wywołanie metody appendTo().
3. Wybranie elementu <body> z drzewa DOM.
4. Dowiązanie elementu <div> jako dziecka.

Modyfikacja drzewa DOM



Różnica pomiędzy metodami z postfixem “To”, a bez niego:

`.append()`



W tych metodach najpierw podajemy nazwę węzła który chcemy zmodyfikować, a dopiero później węzeł dodawany.

```
$(function() {  
    $('body').append('<div></div>');  
});
```

1. Wybranie elementu <body> z drzewa DOM.
2. Wywołanie metody append().
3. Utworzenie węzła <div>.
4. Dowiązanie elementu <div> jako dziecka.



```
<body>  
  <div></div>  
</body>
```

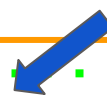
`.appendTo()`



W tych metodach najpierw podajemy węzeł dodawany, a dopiero jako drugi argument nazwę węzła który chcemy zmodyfikować.

```
$(function() {  
    $('<div>  
    </div>').appendTo('body');  
});
```

1. Utworzenie węzła <div>.
2. Wywołanie metody appendTo().
3. Wybranie elementu <body> z drzewa DOM.
4. Dowiązanie elementu <div> jako dziecka.



```
<body>  
  <div></div>  
</body>
```

Modyfikacja drzewa DOM



Różnica pomiędzy metodami z postfixem “To”, a bez niego:

`.append()`



W tych metodach najpierw podajemy nazwę węzła który chcemy zmodyfikować, a dopiero później węzeł dodawany.

```
$(function() {  
    $('body').append('<div></div>');  
});
```

1. Wybranie elementu <body> z drzewa DOM.
2. Wywołanie metody append().
3. Utworzenie węzła <div>.
4. Dowiązanie elementu <div> jako dziecka.



```
<body>  
  <div></div>  
</body>
```

`.appendTo()`



W tych metodach najpierw podajemy węzeł dodawany, a dopiero jako drugi argument nazwę węzła który chcemy zmodyfikować.

```
$(function() {  
    $('<div>  
    </div>').appendTo('body');  
});
```

1. Utworzenie węzła <div>.
2. Wywołanie metody appendTo().
3. Wybranie elementu <body> z drzewa DOM.
4. Dowiązanie elementu <div> jako dziecka.



Wynik działania
jest identyczny!

Modyfikacja drzewa DOM



Przykłady zastosowania metod “before()” i “after()” :

```
$(function() {  
    $('p').before('<div id="przed"></div>');  
});  
  
$(function() {  
    $('p').after('<div id="po"></div>');  
});  
  
<body>  
    <p>Tu jest akapit</p>  
</body>
```


Modyfikacja drzewa DOM



Przykłady zastosowania metod “before()” i “after()” :

```
$(function() {  
    $('p').before('<div id="przed"></div>');  
});  
  
$(function() {  
    $('p').after('<div id="po"></div>');  
});  
  
<body>  
    <p>Tu jest akapit</p>  
</body>
```

Modyfikacja drzewa DOM



Przykłady zastosowania metod “before()” i “after()” :

```
$(function() {  
    $('p').before('<div id="przed"></div>');  
});  
  
$(function() {  
    $('p').after('<div id="po"></div>');  
});  
  
<body>  
    <p>Tu jest akapit</p>  
</body>
```



```
<body>  
    <div id="przed"></div>  
    <p>Tu jest akapit</p>  
    <div id="po"></div>  
</body>
```

Modyfikacja drzewa DOM



Przykłady zastosowania metod “insertBefore()” i “insertAfter()” :

```
$(function() {  
    $('<div id="przed"></div>').insertBefore('p');  
});  
  
$(function() {  
    $('<div id="po"></div>').insertAfter('p');  
});  
  
<body>  
    <p>Tu jest akapit</p>  
</body>
```

Modyfikacja drzewa DOM



Przykłady zastosowania metod “insertBefore()” i “insertAfter()” :

```
$(function() {  
    $('<div id="przed"></div>').insertBefore('p');  
});  
  
$(function() {  
    $('<div id="po"></div>').insertAfter('p');  
});  
  
<body>  
    <p>Tu jest akapit</p>  
</body>
```



```
<body>  
    <div id="przed"></div>  
    <p>Tu jest akapit</p>  
    <div id="po"></div>  
</body>
```

Modyfikacja drzewa DOM



Przykłady dodawania nowego elementu z “appendTo()”:

```
$(function() {  
  $('<div/>', {  
    'class': 'nowy',  
    text: 'Nowy element blokowy',  
    click: function() {  
      $(this).css('color': 'red');  
    }  
  }).appendTo('body');  
});
```

```
<body>  
</body>
```

Dodawanie nowego elementu i określanie jego atrybutów i własności.

Modyfikacja drzewa DOM



Przykłady dodawania nowego elementu z “appendTo()”:

```
$(function() {  
  $('<div/>', {  
    'class': 'nowy',  
    text: 'Nowy element blokowy',  
    click: function() {  
      $(this).css('color': 'red');  
    }  
  }).appendTo('body');  
});
```

```
<body>  
</body>
```

Dodawanie nowego elementu i określanie jego atrybutów i własności.



```
<body>  
  <div class="nowy">Nowy element blokowy</div>  
</body>
```

Modyfikacja drzewa DOM



Poprzednich metod można używać do dodawania nowych węzłów na początku i na końcu dowolnego istniejącego węzła. Istnieją natomiast funkcje które pozwalają na “owijanie” węzłów innymi węzłami:

```
.wrap( )
```

```
.wrapInner( )
```

Modyfikacja drzewa DOM



Poprzednich metod można używać do dodawania nowych węzłów na początku i na końcu dowolnego istniejącego węzła. Istnieją natomiast funkcje które pozwalają na “owijanie” węzłów innymi węzłami:

`.wrap()`



Metoda ta “owija” istniejący element w nowy element podany jako argument omawianej metody.

`.wrapInner()`

Modyfikacja drzewa DOM



Poprzednich metod można używać do dodawania nowych węzłów na początku i na końcu dowolnego istniejącego węzła. Istnieją natomiast funkcje które pozwalają na “owijanie” węzłów innymi węzłami:

`.wrap()`



Metoda ta “owija” istniejący element w nowy element podany jako argument omawianej metody.

`.wrapInner()`



Metoda ta owija zawartość wybranego elementu elementem podanym jako argument omawianej metody.

Modyfikacja drzewa DOM



Poprzednich metod można używać do dodawania nowych węzłów na początku i na końcu dowolnego istniejącego węzła. Istnieją natomiast funkcje które pozwalają na “owijanie” węzłów innymi węzłami:

`.wrap()`



Metoda ta “owija” istniejący element w nowy element podany jako argument omawianej metody.

`.wrapInner()`



Metoda ta owija zawartość wybranego elementu elementem podanym jako argument omawianej metody.

`<p>Tekst akapitu</p>`

```
$(function() {
  $('p').wrap('<div></div>');
});
```

```
$(function() {
  $('p').wrap('<strong></strong>');
});
```

Modyfikacja drzewa DOM



Poprzednich metod można używać do dodawania nowych węzłów na początku i na końcu dowolnego istniejącego węzła. Istnieją natomiast funkcje które pozwalają na “owijanie” węzłów innymi węzłami:

`.wrap()`



Metoda ta “owija” istniejący element w nowy element podany jako argument omawianej metody.

`.wrapInner()`



Metoda ta owija zawartość wybranego elementu elementem podanym jako argument omawianej metody.

`<p>Tekst akapitu</p>`

```
$(function() {
  $('p').wrap('<div></div>');
});
```



```
<div>
  <p>Tekst akapitu</p>
</div>
```

```
$(function() {
  $('p').wrapInner('<strong></strong>');
});
```



```
<p><strong>Tekst akapitu</strong></p>
```

Modyfikacja drzewa DOM



Poprzednich metod można używać do dodawania nowych węzłów na początku i na końcu dowolnego istniejącego węzła. Istnieją natomiast funkcje które pozwalają na “owijanie” węzłów innymi węzłami:

`.wrap()`



Metoda ta “owija” istniejący element w nowy element podany jako argument omawianej metody.

`.wrapInner()`



Metoda ta owija zawartość wybranego elementu elementem podanym jako argument omawianej metody.

`<p>Tekst akapitu</p>`

```
$(function() {
  $('p').wrap('<div></div>');
```

```
$(function() {
  $('p').wrap('<strong></strong>');
```

Możliwy jest jeszcze bardziej zwięzły zapis...

`</div>`

Modyfikacja drzewa DOM



Poprzednich metod można używać do dodawania nowych węzłów na początku i na końcu dowolnego istniejącego węzła. Istnieją natomiast funkcje które pozwalają na “owijanie” węzłów innymi węzłami:

`.wrap()`

Metoda ta “owija” istniejący element w nowy element podany jako argument omawianej metody.

```
$(function() {
  $('p').wrap('<div/>');
});
```

```
<div>
  <p>Tekst akapitu</p>
</div>
```

`.wrapInner()`

Metoda ta owija zawartość wybranego elementu elementem podanym jako argument omawianej metody.

Można stosować zapis skrócony znaczników...

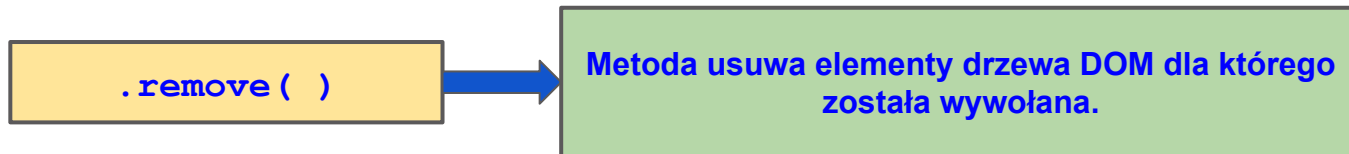
```
$(function() {
  $('p').wrap('<strong/>');
});
```

```
<p><strong>Tekst akapitu</strong></p>
```

Modyfikacja drzewa DOM



Do usuwania elementów drzewa DOM służy tylko jedna metoda:



Modyfikacja drzewa DOM



Do usuwania elementów drzewa DOM służy tylko jedna metoda:

`.remove()`

Metoda usuwa elementy drzewa DOM dla którego została wywołana.

```
$(function() {  
    $('p').remove();  
});  
  
<body>  
  <div>  
    <p>Tekst akapitu </p>  
  </div>  
</body>
```

Modyfikacja drzewa DOM



Do usuwania elementów drzewa DOM służy tylko jedna metoda:

`.remove ()`

Metoda usuwa elementy drzewa DOM dla którego została wywołana.

```
$(function() {  
    $('p').remove();  
});  
  
<body>  
  <div>  
    <p>Tekst akapitu </p>  
  </div>  
</body>
```



```
<body>  
  <div>  
  </div>  
</body>
```


Modyfikacja drzewa DOM



Do usuwania elementów drzewa DOM służy tylko jedna metoda:

`.remove()`

Metoda usuwa elementy drzewa DOM dla którego została wywołana.

```
$(function() {  
    $('#tytul').click(function() {  
        $(this).remove();  
    });  
});  
  
<body>  
    <div>  
        <h1>Tekst akapitu </h1>  
        <h1 id="tytul"> Tytul akapitu </h1>  
    </div>  
</body>
```

Modyfikacja drzewa DOM



Do usuwania elementów drzewa DOM służy tylko jedna metoda:

`.remove()`

Metoda usuwa elementy drzewa DOM dla którego została wywołana.

```
$(function() {  
    $('#tytul').click(function() {  
        $(this).remove();  
    });  
});  
  
<body>  
  <div>  
    <h1>Tekst akapitu </h1>  
    <h1 id="tytul"> Tytul akapitu </h1>  
  </div>  
</body>
```



```
<body>  
  <div>  
    <h1>Tekst akapitu </h1>  
  </div>  
</body>
```

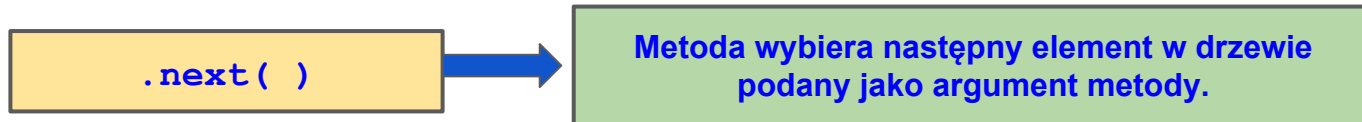
Poruszanie się po strukturze DOM



Do poruszania się i przechodzenia po elementach struktury drzewa DOM służą cztery metody (jedną z nich już poznaliśmy):

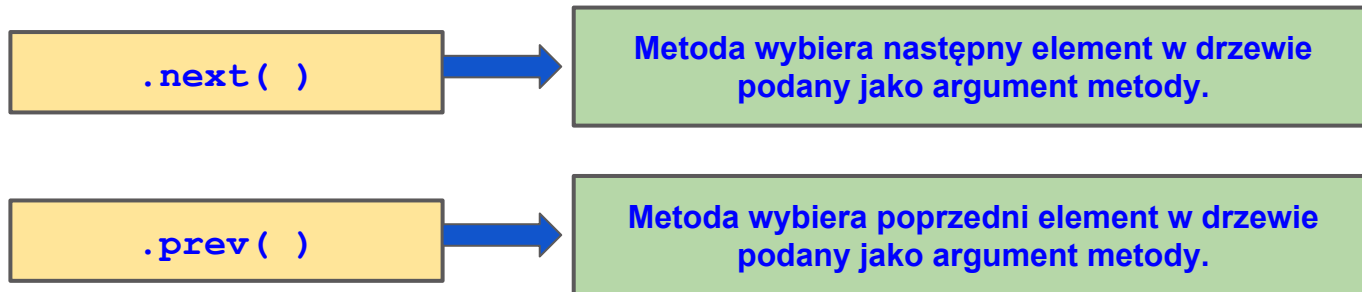
Poruszanie się po strukturze DOM

Do poruszania się i przechodzenia po elementach struktury drzewa DOM służą cztery metody (jedną z nich już poznaliśmy):



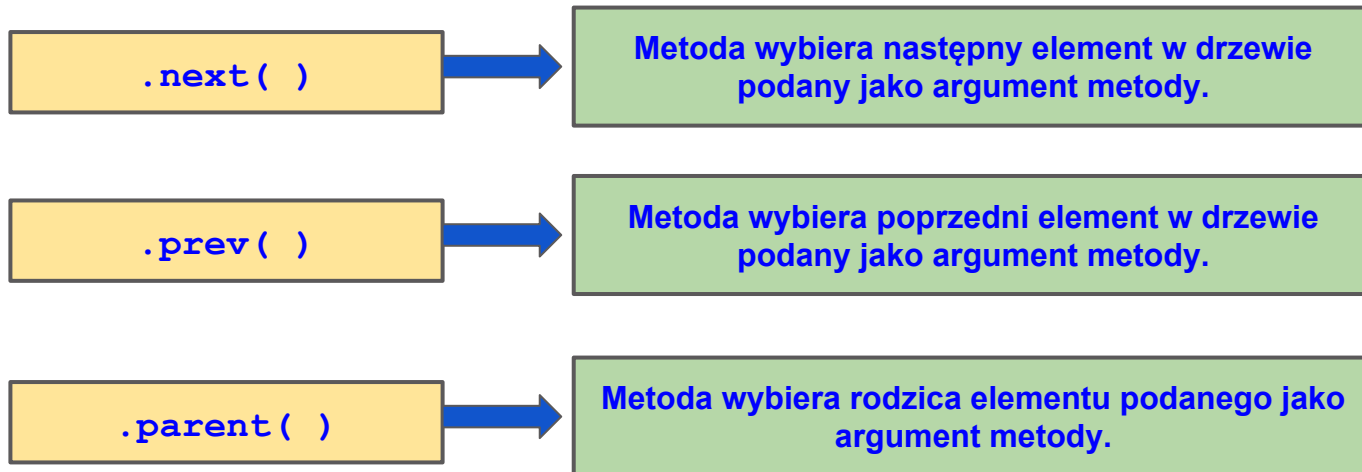
Poruszanie się po strukturze DOM

Do poruszania się i przechodzenia po elementach struktury drzewa DOM służą cztery metody (jedną z nich już poznaliśmy):



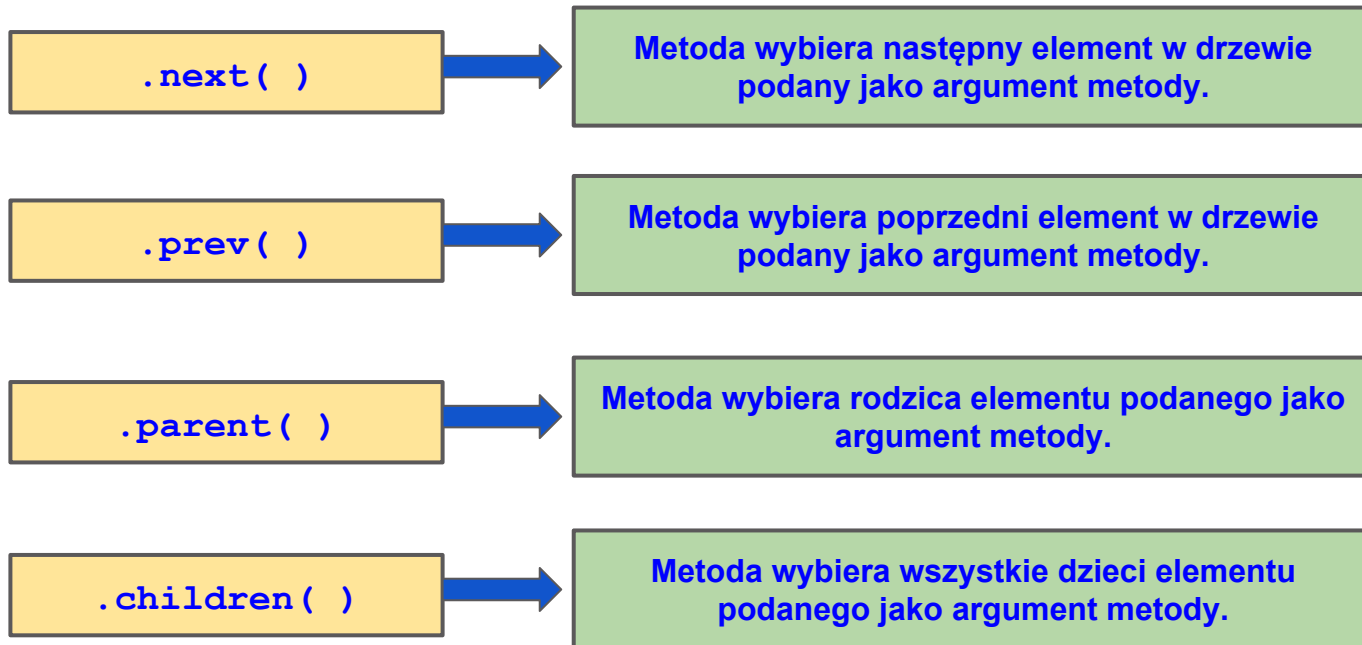
Poruszanie się po strukturze DOM

Do poruszania się i przechodzenia po elementach struktury drzewa DOM służą cztery metody (jedną z nich już poznaliśmy):



Poruszanie się po strukturze DOM

Do poruszania się i przechodzenia po elementach struktury drzewa DOM służą cztery metody (jedną z nich już poznaliśmy):



Poruszanie się po strukturze DOM

Przykład:

```
<body>
  <div id="kontener">
    <h1 id="naglowek">Naglowek akapitu</h1>
    <div id="akapit">
      <p> Tresc akapitu 1 </p>
      <p> Tresc akapitu 2 </p>
      <p> Tresc akapitu 3 </p>
    </div>
    <div id="stopka"></div>
  </div>
</body>
```


Poruszanie się po strukturze DOM

Przykład:

```
<body>
  <div id="kontener">
    <h1 id="naglowek">Naglowek akapitu</h1>
    <div id="akapit">
      <p> Tresc akapitu 1 </p>
      <p> Tresc akapitu 2 </p>
      <p> Tresc akapitu 3 </p>
    </div>
    <div id="stopka"></div>
  </div>
</body>
```

`$('#akapit').next()`



`<div id="stopka"></div>`

Poruszanie się po strukturze DOM

Przykład:

```
<body>
  <div id="kontener">
    <h1 id="naglowek">Naglowek akapitu</h1>
    <div id="akapit">
      <p> Tresc akapitu 1 </p>
      <p> Tresc akapitu 2 </p>
      <p> Tresc akapitu 3 </p>
    </div>
    <div id="stopka"></div>
  </div>
</body>
```

`$('#akapit').next()`



`<div id="stopka"></div>`

`$('#akapit').prev()`



`<h1 id="naglowek"></h1>`

Poruszanie się po strukturze DOM

Przykład:

```
<body>
  <div id="kontener">
    <h1 id="naglowek">Naglowek akapitu</h1>
    <div id="akapit">
      <p> Tresc akapitu 1 </p>
      <p> Tresc akapitu 2 </p>
      <p> Tresc akapitu 3 </p>
    </div>
    <div id="stopka"></div>
  </div>
</body>
```

`$('#akapit').next()`

`<div id="stopka"></div>`

`$('#akapit').prev()`

`<h1 id="naglowek"></h1>`

`$('#akapit').parent()`

`<div id="kontener"></div>`

Poruszanie się po strukturze DOM

Przykład:

```
<body>
  <div id="kontener">
    <h1 id="naglowek">Naglowek akapitu</h1>
    <div id="akapit">
      <p> Tresc akapitu 1 </p>
      <p> Tresc akapitu 2 </p>
      <p class="opis"> Opis opsowy akapitu </p>
    </div>
    <div id="stopka"></div>
  </div>
</body>
```

`$('#akapit').children()`

`<p> Tresc akapitu 1 </p>`
`<p> Tresc akapitu 2 </p>`
`<p> Tresc akapitu 3 </p>`

Poruszanie się po strukturze DOM

Przykład:

```
<body>
  <div id="kontener">
    <h1 id="naglowek">Naglowek akapitu</h1>
    <div id="akapit">
      <p> Tresc akapitu 1 </p>
      <p> Tresc akapitu 2 </p>
      <p class="opis"> Opis opsowy akapitu </p>
    </div>
    <div id="stopka"></div>
  </div>
</body>
```

`$('#akapit').children()`

`<p> Tresc akapitu 1 </p>`
`<p> Tresc akapitu 2 </p>`
`<p> Tresc akapitu 3 </p>`

`$('#akapit').children().eq(0)`

`<p> Tresc akapitu 1 </p>`

`$('#akapit').children().eq(1)`

`<p> Tresc akapitu 2 </p>`

`$('#akapit').children().eq(2)`

`<p class="opis"> Opis opsowy akapitu </p>`

Poruszanie się po strukturze DOM

Przykład:

```
<body>
  <div id="kontener">
    <h1 id="naglowek">Naglowek akapitu</h1>
    <div id="akapit">
      <p> Tresc akapitu 1 </p>
      <p> Tresc akapitu 2 </p>
      <p class="opis"> Opis opsowy akapitu </p>
    </div>
    <div id="stopka"></div>
  </div>
</body>
```

`$('#akapit').children()`

`<p> Tresc akapitu 1 </p>`
`<p> Tresc akapitu 2 </p>`
`<p> Tresc akapitu 3 </p>`

`$('#akapit').children().eq(0)`

`<p> Tresc akapitu 1 </p>`

`$('#akapit').children().eq(1)`

`<p> Tresc akapitu 2 </p>`

`$('#akapit').children().filter('.opis')`

`<p class="opis"> Opis opsowy akapitu </p>`

Wykorzystanie poznanych metod do zrobienia okna “pop-up”



```
<body>
  <h1 id="tytul">Profesor Adam Strzałkowski</h1>
  <div id="akapit">
    <p>Prof. dr hab. Adam Strzałkowski urodził się 26 listopada 1923 roku w
      Tenczynku. Studia fizyki ukończył na Uniwersytecie Jagiellońskim w 1948
      roku. Karierę zaczynał od astronomii w Obserwatorium Astronomicznym UJ u
      profesora Tadeusza Banachiewicza, którego uważa za swojego pierwszego
      mistrza.
    </p>
  </div>
</body>
```

Wykorzystanie poznanych metod do zrobienia okna “pop-up”



Chcemy aby po najchechniu
na ten napis pojawiał się
opis z dodatkowymi
wyjaśnieniami np. jako okno
pop-up.

```
<body>
  <h1 id="tytul">Profesor Adam Strzałkowski</h1>
  <div id="akapit">
    <p>Prof. dr hab. Adam Strzałkowski urodził się 26 listopada 1923 roku w
      Tenczynku. Studia fizyki ukończył na Uniwersytecie Jagiellońskim w 1948
      roku. Karierę zaczynał od astronomii w Obserwatorium Astronomicznym UJ u
      profesora Tadeusza Banachiewicza, którego uważa za swojego pierwszego
      mistrza.
    </p>
  </div>
</body>
```


Wykorzystanie poznanych metod do zrobienia okna “pop-up”



Chcemy aby po najchechniu
na ten napis pojawiał się
opis z dodatkowymi
wyjaśnieniami np. jako okno
pop-up.

```
<body>
  <h1 id="tytul">Profesor Adam Strzałkowski</h1>
  <div id="akapit">
    <p>Prof. dr hab. Adam Strzałkowski urodził się 26 listopada 1923 roku w
      Tenczynku. Studia fizyki ukończył na Uniwersytecie Jagiellońskim w 1948
      roku. Karierę zaczynał od astronomii w Obserwatorium Astronomicznym UJ u
      profesora Tadeusza Banachiewicza, którego uważa za swojego pierwszego
      mistrza.
    </p>
  </div>
</body>
```

Wykorzystanie poznanych metod do zrobienia okna “pop-up”



Chcemy aby po najchechniu
na ten napis pojawiał się
opis z dodatkowymi
wyjaśnieniami np. jako okno
pop-up.

```
<body>
  <h1 id="tytul">Profesor Adam Strzałkowski</h1>
  <div id="akapit">
    <p>Prof. dr hab. Adam Strzałkowski urodził się 26 listopada 1923 roku w
      Tenczynku. Studia fizyki ukończył na Uniwersytecie Jagiellońskim w 1948
      roku. Karierę zaczynał od astronomii w Obserwatorium Astronomicznym UJ u
      profesora Tadeusza Banachiewicza, którego uważa za swojego pierwszego
      mistrza.
    </p>
  </div>
</body>
```

Na początek musimy zmodyfikować kod HTML

Wykorzystanie poznanych metod do zrobienia okna “pop-up”



```
<body>
  <h1 id="tytul">Profesor Adam Strzałkowski</h1>
  <div id="akapit">
    <p>Prof. dr hab. Adam Strzałkowski urodził się 26 listopada 1923 roku w
      Tenczynku. Studia fizyki ukończył na
      <span class="przypis">
        <span class="slowo">Uniwersytecie Jagiellońskim</span>
        <span class="wyraz">Uniwersytet Jagielloński w Krakowie</span>
        <span class="opis">Najstarszy uniwersytet w Polsce</span>
      </span>
      w 1948 roku. Karierę zaczynał od astronomii w Obserwatorium Astronomicznym
      UJ u profesora Tadeusza Banachiewicza, którego uważa za swojego pierwszego
      mistrza.
    </p>
  </div>
</body>
```

Wykorzystanie poznanych metod do zrobienia okna “pop-up”



To co zaznaczone zostało na kolor czerwony będzie treścią okna “pop-up” po najechaniu na przypis kursorem myszy.

```
<body>
  <h1 id="tytul">Profesor Adam Strzałkowski</h1>
  <div id="akapit">
    <p>Prof. dr hab. Adam Strzałkowski urodził się 15 marca 1948 roku w
      Tenczyнку. Studia fizyki ukończył w 1970 roku na Uniwersytecie
      Jagiellońskim</p>
    <span class="przypis">
      <span class="slowo">Uniwersytecie Jagiellońskim</span>
      <span class="wyraz">Uniwersytet Jagielloński w Krakowie</span>
      <span class="opis">Najstarszy uniwersytet w Polsce</span>
    </span>
    w 1948 roku. Karierę zaczynał od astronomii w Obserwatorium Astronomicznym
    UJ u profesora Tadeusza Banachiewicza, którego uważa za swojego pierwszego
    mistrza.
  </p>
</div>
</body>
```

Wykorzystanie poznanych metod do zrobienia okna “pop-up”



```
$(function() {  
    $('div').attr('id', 'popup').prependTo('.przypis').hide();  
    $('.przypis').mouseover(function() {  
        $('#popup').text('').  
            append('<h3>' + $(this).children().eq(1).text() + '</h3>').  
            append($(this).children().eq(2).text()).  
            show();  
    }).mouseout(function() {  
        $('#popup').hide();  
    }).mousemove(function(e) {  
        $('#popup').css('left', e.pageX + 10).css('top', e.pageY + 10);  
    });  
});
```

Wykorzystanie poznanych metod do zrobienia okna “pop-up”



```
$(function() {  
    $('div').attr('id', 'popup').prependTo('.przypis').hide();  
    $('.przypis').mouseover(function() {  
        $('#popup').text('').  
            append('<h3>' + $(this).children().eq(1).text() + '</h3>').  
            append($(this).children().eq(2).text()).  
            show();  
    }).mouseout(function() {  
        $('#popup').hide();  
    }).mousemove(function(e) {  
        $('#popup').css('left', e.pageX + 10).css('top', e.pageY + 10);  
    });  
});
```

**Sam kod jQuery nie wystarczy trzeba jeszcze napisać reguły CSS
sytlizujące wygląd okna “pop-up”.**

AJAX i jQuery



Już wcześniej poznaliśmy różnice pomiędzy synchronicznym i asynchronicznym modelem przetwarzania danych poprzez protokół HTTP.

Wykonywanie asynchronicznych zapytań AJAX było możliwe dzięki wykorzystaniu własności obiektu “XMLHttpRequest()”. Dzięki tej technologii możliwe jest np. uzupełnienie danego dokumentu HTML od dane pobrane z serwera bez konieczności przeładowania danego dokumentu HTML. Podobną funkcjonalność oferuje biblioteka jQuery.

AJAX i jQuery



Już wcześniej poznaliśmy różnice pomiędzy synchronicznym i asynchronicznym modelem przetwarzania danych poprzez protokół HTTP.

Wykonywanie asynchronicznych zapytań AJAX było możliwe dzięki wykorzystaniu własności obiektu “XMLHttpRequest()”. Dzięki tej technologii możliwe jest np. uzupełnienie danego dokumentu HTML o dane pobrane z serwera bez konieczności przeładowania danego dokumentu HTML. Podobną funkcjonalność oferuje biblioteka jQuery.

Możemy asynchronicznie pobrać dodatkowe dane z serwera a następnie korzystając z metod dzięki którym możliwe jest modyfikowanie struktury DOM wyświetlić te dane użytkownikowi.

AJAX i jQuery



Już wcześniej poznaliśmy różnice pomiędzy synchronicznym i asynchronicznym modelem przetwarzania danych poprzez protokół HTTP.

Wykonywanie asynchronicznych zapytań AJAX było możliwe dzięki wykorzystaniu własności obiektu “XMLHttpRequest()”. Dzięki tej technologii możliwe jest np. uzupełnienie danego dokumentu HTML o dane pobrane z serwera bez konieczności przeładowania danego dokumentu HTML. Podobną funkcjonalność oferuje biblioteka jQuery.

Możemy asynchronicznie pobrać dodatkowe dane z serwera a następnie korzystając z metod dzięki którym możliwe jest modyfikowanie struktury DOM wyświetlić te dane użytkownikowi.

W jQuery metoda `.load()` służy do asynchronicznego pobierania danych z serwera i ich wstawienia do dokumentu. Pozwala na ustalenie zawartości elementu treścią pobraną z adresu URL podanego jako argument metody.

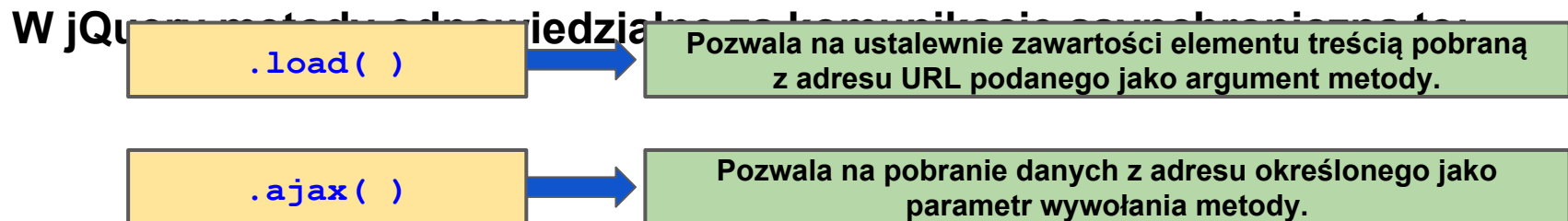
AJAX i jQuery



Już wcześniej poznaliśmy różnice pomiędzy synchronicznym i asynchronicznym modelem przetwarzania danych poprzez protokół HTTP.

Wykonywanie asynchronicznych zapytań AJAX było możliwe dzięki wykorzystaniu własności obiektu “XMLHttpRequest()”. Dzięki tej technologii możliwe jest np. uzupełnienie danego dokumentu HTML od dane pobrane z serwera bez konieczności przeładowania danego dokumentu HTML. Podobną funkcjonalność oferuje biblioteka jQuery.

Możemy asynchronicznie pobrać dodatkowe dane z serwera a następnie korzystając z metod dzięki którym możliwe jest modyfikowanie struktury DOM wyświetlić te dane użytkownikowi.



AJAX i jQuery



Spróbujmy najpierw określić możliwości działania metody “load()”:

```
$( selektor ).load( URL, paramtery-URL, function() );
```

AJAX i jQuery



Spróbujmy najpierw określić możliwości działania metody “load()”:

```
$( selektor ).load( URL, paramtery-URL, function() );
```

Przykład wywołania - zakładamy że ładujemy treść z strony “tresc.html”

```
$(function(){
    $('#tresc').load('tresc.html', 'bcs=34&sort=asc', function(html){
        $('#span#info').text('Tresc zostala zaladowana!');
    });
});

<body>
  <div id="tresc"></div>
  <span id="info"></span>
</body>
```

AJAX i jQuery



Spróbujmy najpierw określić możliwości działania metody “load()”:

```
$( selektor ).load( URL, paramtery-URL, function() );
```

Przykład wywołania - zakładka

Postać wywoływanego adresu:

```
"tresc.html?bcs=34&sort=asc"
```

“tresc.html”

```
$(function() {  
    $('#tresc').load('tresc.html', 'bcs=34&sort=asc', function(html) {  
        $('#span#info').text('Tresc zostala zaladowana!');  
    });  
});
```

```
<body>  
    <div id="tresc"></div>  
    <span id="info"></span>  
</body>
```

AJAX i jQuery



Spróbujmy najpierw określić możliwości działania metody “load()”:

```
$( selektor ).load( URL, paramtery-URL, function() );
```

Przykład wywołania - zakładamy że ładujemy

W argumencie przechowywane są pobrane dane z adresu URL.

```
$(function() {  
    $('#tresc').load('tresc.html', 'bcs=34&sort=asc', function(html) {  
        $('#span#info').text('Tresc zostala zaladowana!');  
    });  
});  
  
<body>  
    <div id="tresc"></div>  
    <span id="info"></span>  
</body>
```

AJAX i jQuery



Spróbujmy najpierw określić możliwości działania metody “load()”:

```
$( selektor ).load( URL, paramtery-URL, function() );
```

Przykład wywołania - zakładamy że ładujemy

W argumencie przechowywane są pobrane dane z adresu URL.

```
$(function() {  
    $('#tresc').load('tresc.html', 'bcs=34&sort=asc', function(html) {  
        $('#span#info').text('Tresc została załadowana!');  
    });  
});  
  
<body>  
    <div id="tresc"></div>  
    <span id="info"></span>  
</body>
```

Działanie metody “load()” przypomina funkcjonalnością metodę “include()” z języka PHP.

AJAX i jQuery



Użycie metody “ajax()” jest nieco trudniejsze:

```
$.ajax( {
```

```
});
```


AJAX i jQuery



Użycie metody “ajax()” jest nieco trudniejsze:

```
$.ajax( {  
    url: URL,  
    timeout: ms,  
    cache: true/false,  
    success: function(html){},  
    beforeSend: function(){},  
    error: function(){}  
});
```

AJAX i jQuery



Użycie metody “ajax()” jest nieco trudniejsze:

```
$.ajax( {  
    url: URL,  
    timeout: ms,  
    cache: true/false,  
    success: function(html){},  
    beforeSend: function() {},  
    error: function() {}  
});
```

Metoda “ajax()” nie działa na żadnym obiekcie drzewa DOM !!!!


Parametry do wywołania tej metody przekazywane są w postaci obiektu (tablicy asocjacyjnej) określającej parametry połączenia oraz zdania jakie mają zostać wykonane po zakończeniu żądania.

AJAX i jQuery



Użycie metody “ajax()” jest nieco trudniejsze:

```
$.ajax( {  
  url: URL,  
  timeout: ms,  
  cache: true/false,  
  success: function(html){},  
  beforeSend: function(){},  
  error: function(){}  
});
```




AJAX i jQuery



Użycie metody “ajax()” jest nieco trudniejsze:

```
$.ajax( {  
  url: URL,  
  timeout: ms,  
  cache: true/false,  
  success: function(html){},  
  beforeSend: function(){},  
  error: function(){}  
});
```



AJAX i jQuery



Użycie metody “ajax()” jest nieco trudniejsze:

```
$.ajax( {  
  url: URL,  
  timeout: ms,  
  cache: true/false,  
  success: function(html){},  
  beforeSend: function(){},  
  error: function(){}  
});
```

adres URL

maksymalny czas oczekiwania
na odpowiedź serwera

AJAX i jQuery



Użycie metody “ajax()” jest nieco trudniejsze:

```
$.ajax( {  
  url: URL,  
  timeout: ms,  
  cache: true/false,  
  success: function(html){},  
  beforeSend: function(){},  
  error: function(){}  
});
```

adres URL

maksymalny czas oczekiwania
na odpowiedź serwera

wyłącza stosowanie pamięci
podręcznej przeglądarki

AJAX i jQuery



Użycie metody “ajax()” jest nieco trudniejsze:

```
$.ajax( {  
  url: URL,  
  timeout: ms,  
  cache: true/false,  
  success: function(html) {},  
  beforeSend: function() {},  
  error: function() {}  
});
```

adres URL

maksymalny czas oczekiwania
na odpowiedź serwera

wyłącza stosowanie pamięci
podręcznej przeglądarki

wywołanie zwrotne
realizowane w przypadku
powodzenia żądania

AJAX i jQuery



Użycie metody “ajax()” jest nieco trudniejsze:

```
$.ajax( {  
  url: URL,  
  timeout: ms,  
  cache: true/false,  
  success: function(html) {},  
  beforeSend: function() {},  
  error: function() {}  
});
```

adres URL

maksymalny czas oczekiwania
na odpowiedź serwera

wyłącza stosowanie pamięci
podręcznej przeglądarki

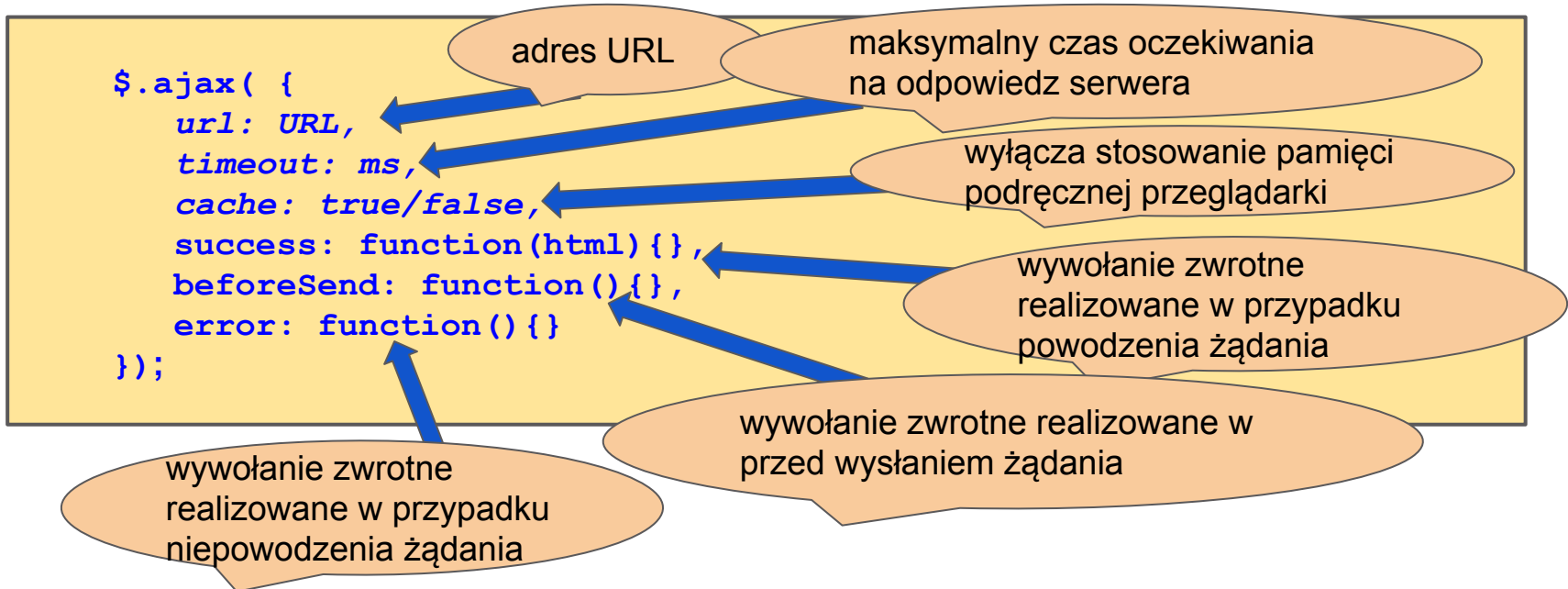
wywołanie zwrotne
realizowane w przypadku
powodzenia żądania

wywołanie zwrotne realizowane w
przed wysłaniem żądania

AJAX i jQuery



Użycie metody “ajax()” jest nieco trudniejsze:



AJAX i jQuery



Przykład 1:

```
$(function() {  
    $.ajax({  
        url: "localhost/api/getDataHtml",  
        success: function(html) {  
            $("#tresc").html(html);  
        }  
    });  
});  
<body>  
    <div id="tresc"></div>  
</body>
```

Pobieramy asynchronicznie dane ze adresu URL i w razie sukcesu zakończenia żądania wyświetlamy treść (html) w elemencie #tresc.

AJAX i jQuery



Przykład 2:

```
$(function() {  
    $.ajax({  
        url: "localhost/api/getDataJson",  
        dataType: "json",  
        success: function(json) {  
            $("#tresc").html(JSON.stringify(json));  
        }  
    });  
});  
<body>  
    <div id="tresc"></div>  
</body>
```

określamy typ danych
przychodzących

Pobieramy asynchronicznie dane ze adresu URL i w razie sukcesu zakończenia żądania wyświetlamy treść (json) w elemencie #tresc.

AJAX i jQuery



Przykład 3:

```
$(function(){  
  $.ajax({  
    url: 'localhost/api/getData',  
    type: 'GET',  
    data: 'Username=jquery4u',  
    success: function(data) {  
      $('#results').html(data);  
    },  
    error: function(e) {  
      console.log(e.message);  
    }  
  });  
});
```

określamy metody wysyłania
danych

przesyłane dane
w metodzie GET


AJAX i jQuery



Przykład 4:

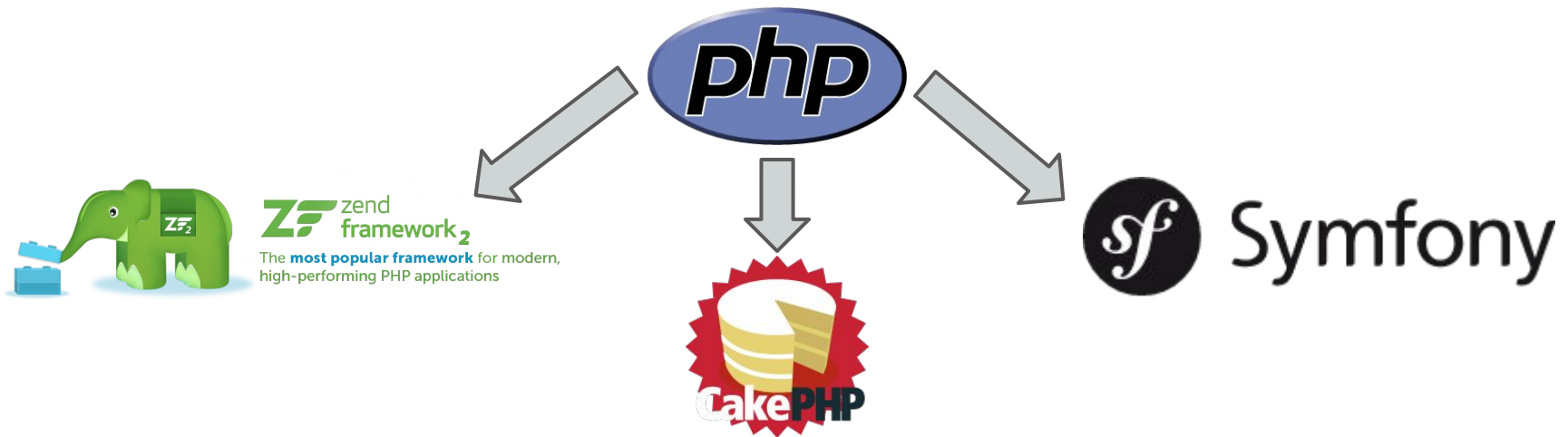
```
$(function(){  
  $.ajax({  
    method: 'POST',  
    url: 'localhost/api/sendData',  
    data: { name: "Jan", fname: "Kowalski" },  
    success: function() {  
      alert( 'Dane zostały przesłane ' );  
    }  
  });  
});
```

określamy metody wysyłania
danych



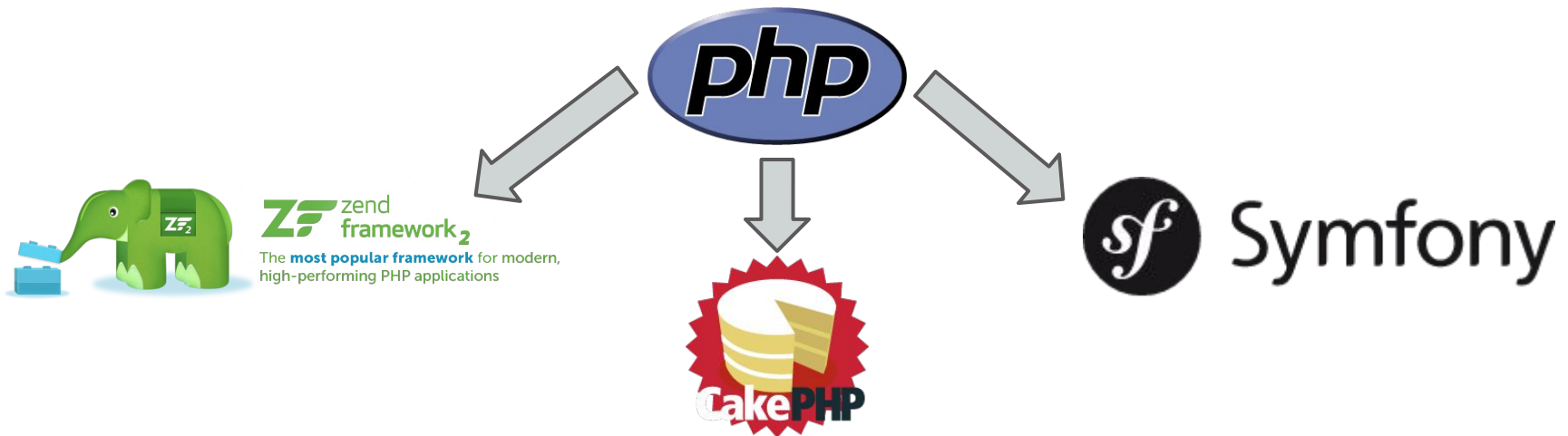
Popularne dostępne rozwiązania

Najpopularniejsze środowiska programistyczne:



Popularne dostępne rozwiązania

Najpopularniejsze środowiska programistyczne:



oraz systemy CMS (Content Manager System):



Dlaczego NODE.JS a nie PHP



vs.



Obecnie PHP jest wykorzystywane na około 75% wszystkich stron internetowych.

Dlaczego NODE.JS a nie PHP



vs.



Obecnie PHP jest wykorzystywane na około 75% wszystkich stron internetowych.

Wiele nowych projektów odchodzi od PHP na rzecz NODE.JS jako rozwiązania bardziej wydajnego, elastycznego i niezależnego.

Dlaczego NODE.JS a nie PHP



vs.

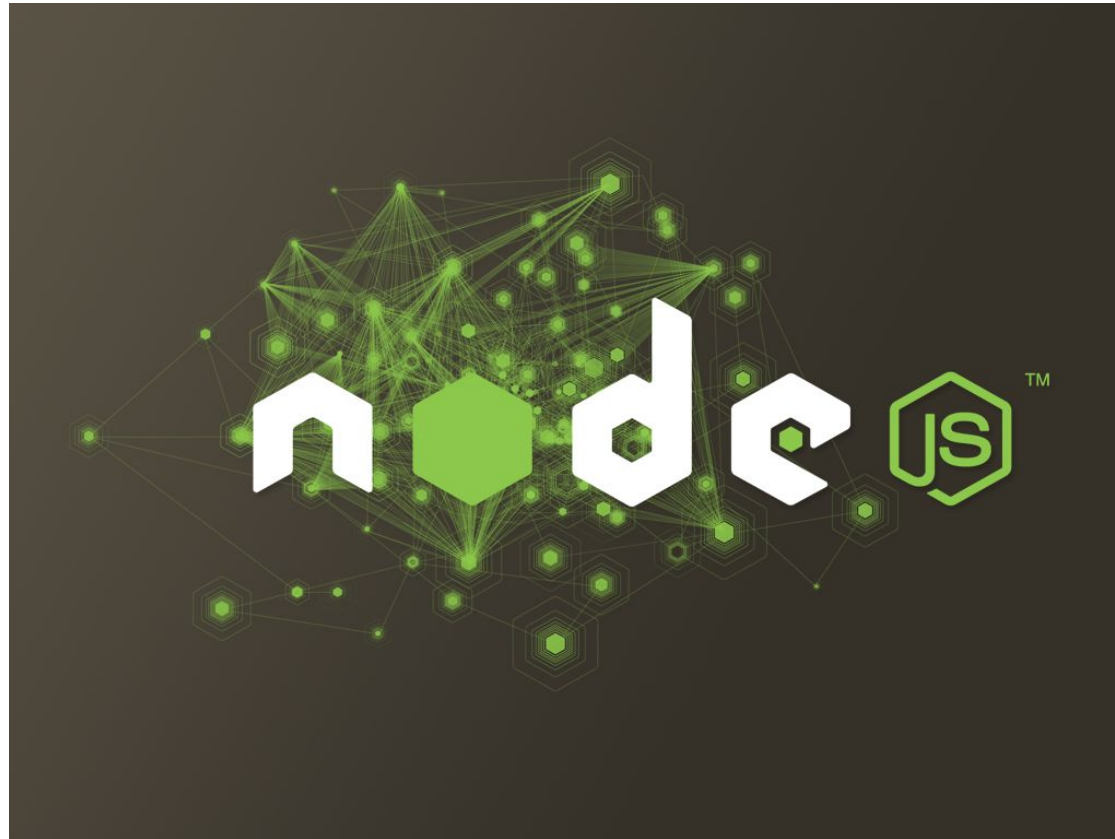


Obecnie PHP jest wykorzystywane na około 75% wszystkich stron internetowych.

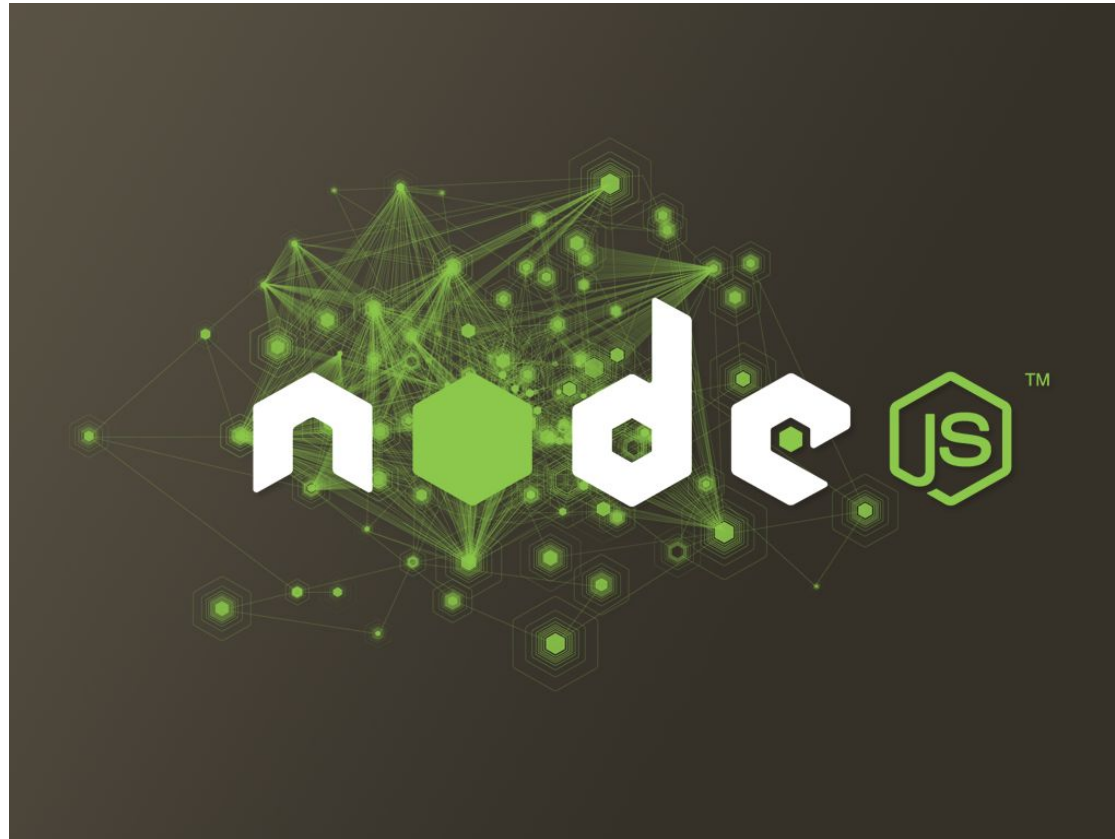
Wiele nowych projektów odchodzi od PHP na rzecz NODE.JS jako rozwiązania bardziej wydajnego, elastycznego i niezależnego.

Daje możliwości tworzenia łatwo skalowalnych aplikacji internetowych, których działanie jest sterowane zdarzeniowo wykorzystując żądania asynchroniczne czyli bez konieczności przeładowania danego zasobu.

Node.js



Node.js



<http://nodejs.org/>

Node.js

Co to jest NODE.JS ?

Node.js

Co to jest NODE.JS ?



```
graph TD; A[Co to jest NODE.JS ?] --> B[Nowoczesne środowisko programistyczne];
```

**Nowoczesne środowisko
programistyczne**

Środowisko programistyczne w sensie zestawu gotowych klas i metod których można używać do przygotowania własnych skalowalnych i wydajnych aplikacji internetowych.

Krótką historia Node.js

Środowisko Node.js zostało stworzone w 2009 roku, przez Ryana Dahla, który w tamtym czasie pracował w firmie Joyent.



Ryan Dahl



Krótką historia Node.js

Środowisko Node.js zostało stworzone w 2009 roku, przez Ryana Dahla, który w tamtym czasie pracował w firmie Joyent.



Ryan Dahl



Krótką historia Node.js

Node.js jest środowiskiem programistycznym napisanym w języku JavaScript. Aplikacje stworzone w środowisku Node.js są sterowane zdarzeniami wykorzystując system wejścia/wyjścia (I/O) który jest asynchroniczny.



Ryan Dahl

Krótką historia Node.js

Node.js jest środowiskiem programistycznym napisanym w języku JavaScript. Aplikacje stworzone w środowisku Node.js są sterowane zdarzeniami wykorzystując system wejścia/wyjścia (I/O) który jest asynchroniczny.



Ryan Dahl



NODE.JS ON THE ROAD

PRODUCTION NODE HITS 1
GET INSPIRED & INVOLVED

<http://nodejs.org/>

Krótką historia Node.js

Node.js jest środowiskiem programistycznym napisanym w języku JavaScript. Aplikacje stworzone w środowisku Node.js są sterowane zdarzeniami wykorzystując system wejścia/wyjścia (I/O) który jest asynchroniczny.



Ryan Dahl



NODE.JS ON THE ROAD

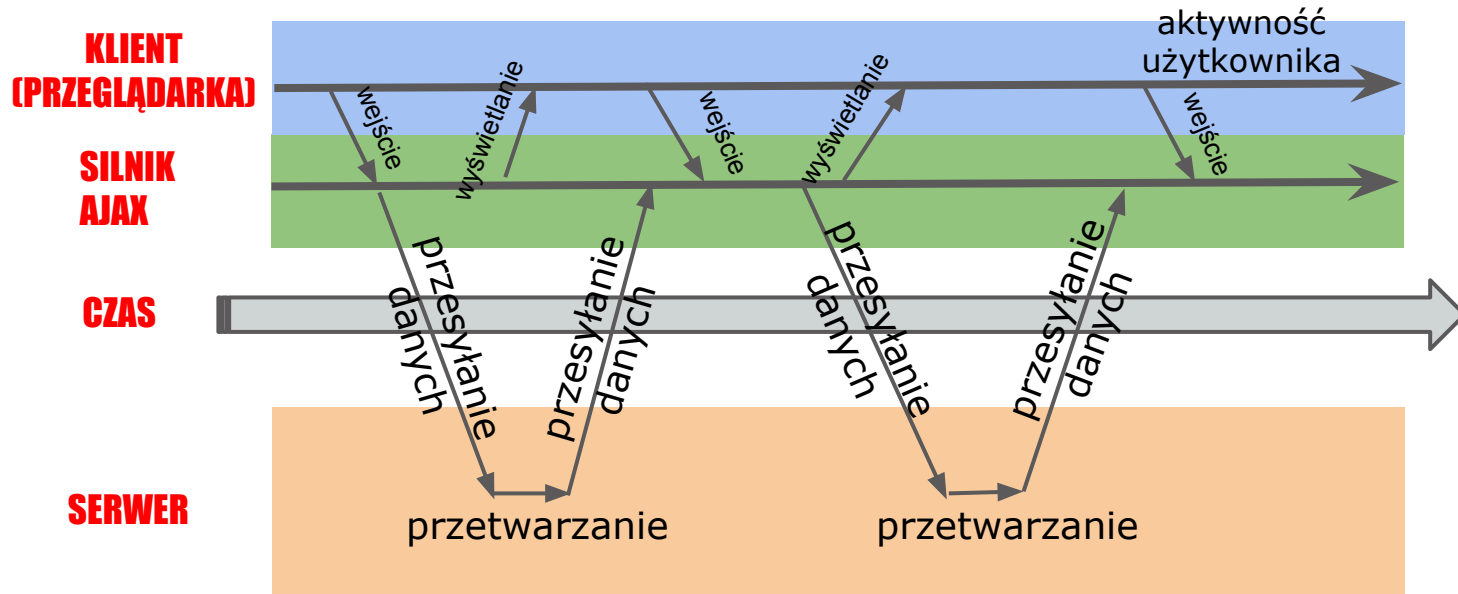
PRODUCTION NODE HITS
GET INSPIRED & INVOLVED

<http://nodejs.org/>

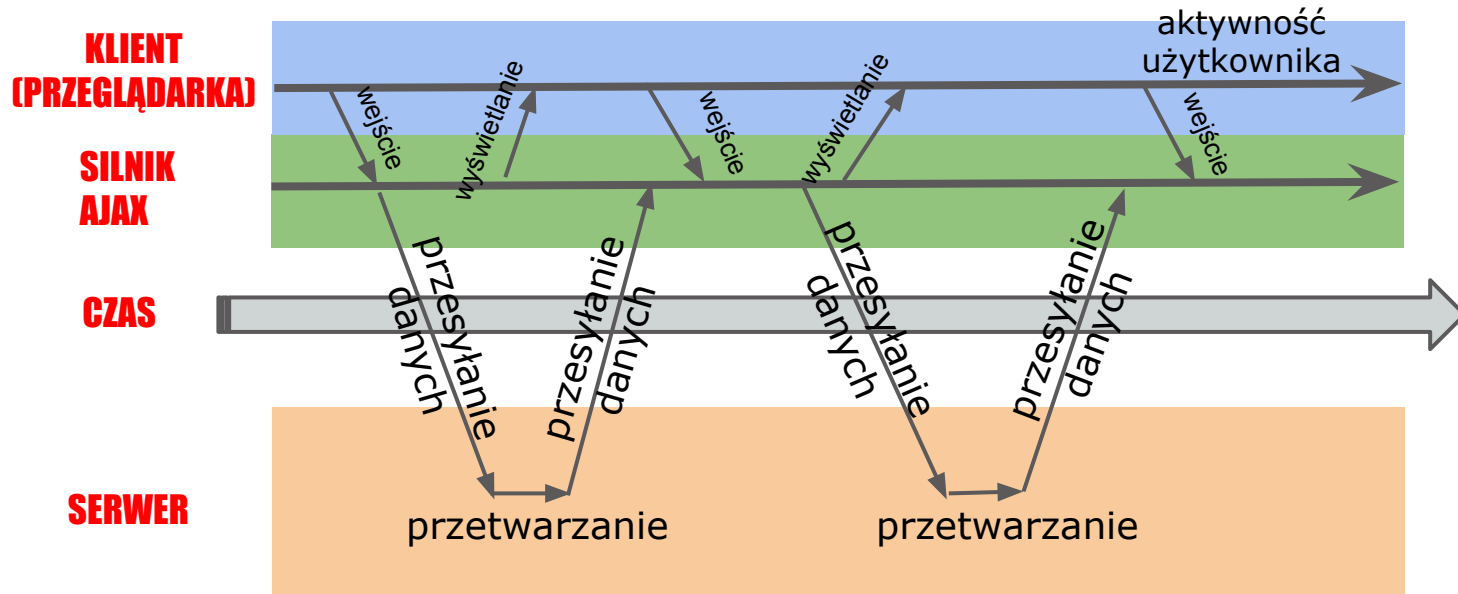
Najnowsza wersja: v.5.1.0

(17 październik 2015 r.)

Przypomnienie: asynchroniczność



Przypomnienie: asynchroniczność



Jest to model asynchronicznej komunikacji HTTP, gdzie klient (przeglądarka) nie czeka na przyjęcie odpowiedzi na żądanie serwera, a wykonuje dalsze żądania.

W takim modelu nie ma konieczności przeładowania strony przy każdej operacji klienta, wystarczy, że zostaną doczytane brakujące dane, a dzięki odpowiednim narzędziom zmodyfikowana zostanie zawartość strony.

Krótką historia Node.js

Powstanie Node.js było zainspirowane przez funkcjonalności “push” jakie oferuje np. poczta GMAIL...

Co to jest “push” ?

Krótką historia Node.js

Powstanie Node.js było zainspirowane przez funkcjonalności “push” jakie oferuje np. poczta GMAIL...

Co to jest “push” ?



Usługi push działają w oparciu o przekazane przez klienta wcześniej informacje, na podstawie których serwer dostarcza klientowi nowych danych w zależności od tego czy są one dostępne.

Krótką historia Node.js

Powstanie Node.js było zainspirowane przez funkcjonalności “push” jakie oferuje np. poczta GMAIL...

Co to jest “push” ?



Usługi push działają w oparciu o przekazane przez klienta wcześniej informacje, na podstawie których serwer dostarcza klientowi nowych danych w zależności od tego czy są one dostępne.

Przykład:

Przykładem usługi “push” jest synchroniczny chat.

Krótką historia Node.js

Node. jest czysto JavaScriptowym środowiskiem tworzenia stron internetowych działającym po stronie serwera.

Do tej pory nauczyliśmy się że...

Krótką historia Node.js

Node. jest czysto JavaScriptowym środowiskiem tworzenia stron internetowych działającym po stronie serwera.

Do tej pory nauczyliśmy się że...



JavaScript jest wykonywany po stronie klienta (przeglądarki)!

Krótką historia Node.js

Node. jest czysto JavaScriptowym środowiskiem tworzenia stron internetowych działającym po stronie serwera.

Do tej pory nauczyliśmy się że...



JavaScript jest wykonywany po stronie klienta (przeglądarki)!

Jak zatem jest możliwe że Node.js działa na serwerze ?

Google V8

Jak zatem jest możliwe że Node.js działa na serwerze ?

Google V8

Jak zatem jest możliwe że Node.js działą na serwerze ?



Silnik V8 został stworzony na potrzeby przeglądarki Chrome, do obsługi skryptów napisanych w języku JavaScript.

Google V8

Silnik V8 został stworzony przez firmę Google na potrzeby przeglądarki Chrome, do obsługi skryptów napisanych w języku JavaScript.



Google V8

Silnik V8 został stworzony przez firmę Google na potrzeby przeglądarki Chrome, do obsługi skryptów napisanych w języku JavaScript.

Jest to program napisany w języku C++, który “kompiluje” skrypty do kodu maszynowego, a dopiero następnie wykonuje. Dzięki takiej procedurze uzyskano wzrost wydajności i szybkości wykonywania skryptów JS.

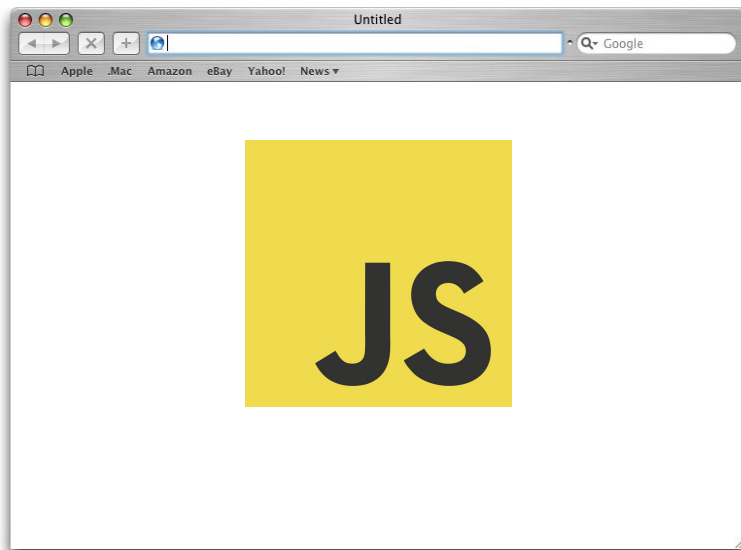


JavaScript po stronie serwera

**Dzięki zastosowaniu silnika V8 wprowadzono zupełnie
nowe podejście w myśleniu o wykonywaniu kodu
JavaScript**

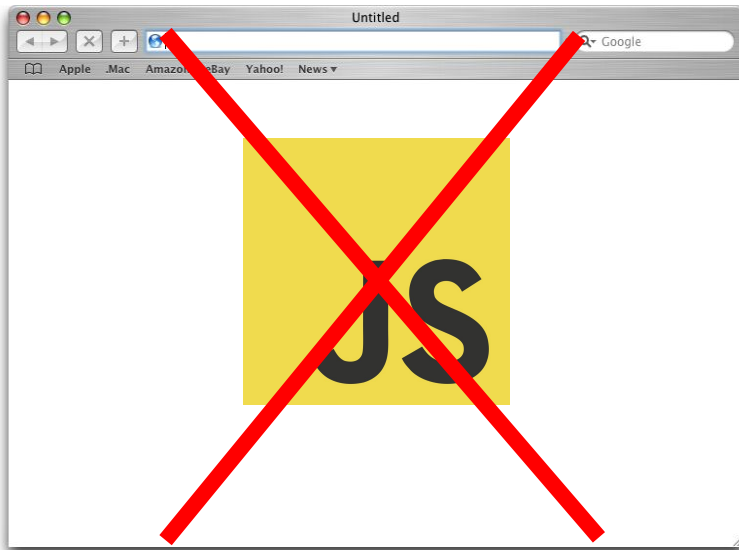
JavaScript po stronie serwera

Dzięki zastosowaniu silnika V8 wprowadzono zupełnie nowe podejście w myśleniu o wykonywaniu kodu JavaScript



JavaScript po stronie serwera

Dzięki zastosowaniu silnika V8 wprowadzono zupełnie nowe podejście w myśleniu o wykonywaniu kodu JavaScript



JavaScript po stronie serwera

**Musimy odwrócić sposób myślenia jeśli chodzi
wykonywanie JavaScriptu.**



JavaScript po stronie serwera

SERVER SIDE

CLIENT SIDE

JavaScript po stronie serwera

SERVER SIDE

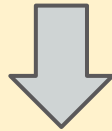
```
1 var edge = require('edge');
2
3 var hello = edge.func(function () {/*
4   async (input) =>
5     {
6       return ".NET welcomes " + input.ToString();
7     }
8   */});
9
10 hello('Node.js', function (error, result) {
11   if (error) throw error;
12   console.log(result);
13 });
```

CLIENT SIDE

JavaScript po stronie serwera

SERVER SIDE

```
1 var edge = require('edge');
2
3 var hello = edge.func(function () { /*
4   async (input) =>
5     {
6       return ".NET welcomes " + input.ToString();
7     }
8   */});
9
10 hello('Node.js', function (error, result) {
11   if (error) throw error;
12   console.log(result);
13 });
```

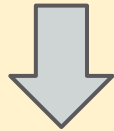


CLIENT SIDE

JavaScript po stronie serwera

SERVER SIDE

```
1 var edge = require('edge');
2
3 var hello = edge.func(function () { /*
4   async (input) =>
5     {
6       return ".NET welcomes " + input.ToString();
7     }
8   */});
9
10 hello('Node.js', function (error, result) {
11   if (error) throw error;
12   console.log(result);
13 });
```

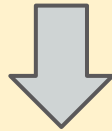


CLIENT SIDE

JavaScript po stronie serwera

SERVER SIDE

```
1 var edge = require('edge');
2
3 var hello = edge.func(function () { /*
4   async (input) =>
5   {
6     return ".NET welcomes " + input.ToString();
7   }
8 */});
9
10 hello('Node.js', function (error, result) {
11   if (error) throw error;
12   console.log(result);
13 });
```



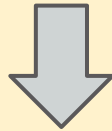
HTTP

CLIENT SIDE

JavaScript po stronie serwera

SERVER SIDE

```
1 var edge = require('edge');
2
3 var hello = edge.func(function () {/*
4   async (input) =>
5     {
6       return ".NET welcomes " + input.ToString();
7     }
8   */});
9
10 hello('Node.js', function (error, result) {
11   if (error) throw error;
12   console.log(result);
13 });
```



HTTP

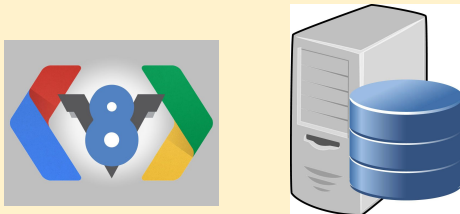
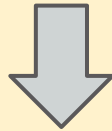
CLIENT SIDE

```
!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Sta
head>
<meta http-equiv="Content-Type" content="text/h
<title>CSS3</title>
<link href="style.css" rel="stylesheet" type="t
</head>
<body>
<div id="container"> <!-- Main Container -->
<div class="menu"> <!-- Menu here --></div>
<div class="article"><h2>CSS3 Sample</h2></div>
<div class="article"><h2>CSS3 & HTML5 are so good!</h2></div>
```

JavaScript po stronie serwera

SERVER SIDE

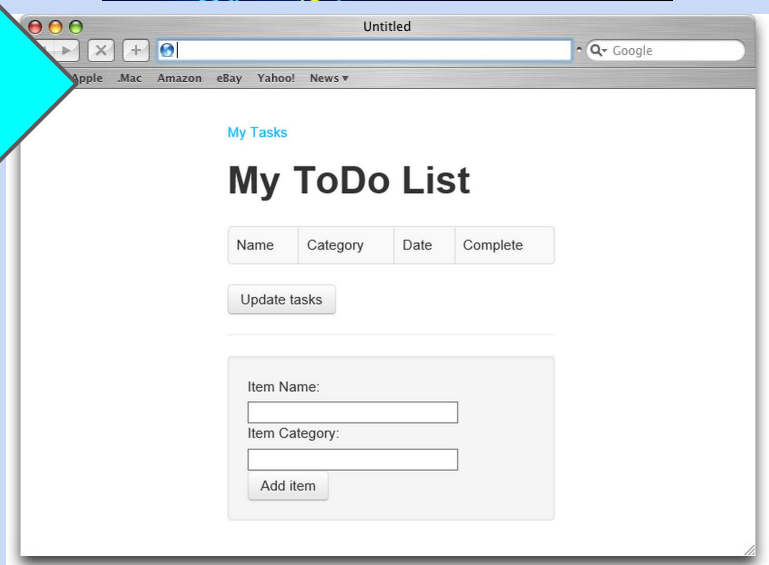
```
1 var edge = require('edge');
2
3 var hello = edge.func(function () { /*
4   async (input) =>
5     {
6       return ".NET welcomes " + input.ToString();
7     }
8   */});
9
10 hello('Node.js', function (error, result) {
11   if (error) throw error;
12   console.log(result);
13 });
```



CLIENT SIDE

```
!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html" />
<title>CSS3</title>
<link href="style.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="container"> <!-- Main Container -->
<div id="menu"> <!-- Menu here -->
<div class="article"><h2>CSS3 Sample</h2></div>
<div class="article"><h2>CSS3 & HTML5 are so good!</h2></div>
</div>
</body>
</html>
```

HTTP



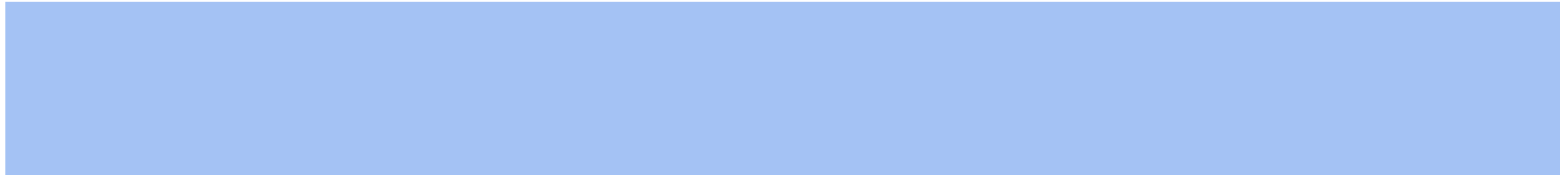
Obsługa żądań

MODEL TRADYCYJNY

KLIENCI



SERWER



Obsługa żądań

MODEL TRADYCYJNY

KLIENCI

Żądanie HTTP

SERWER

Wątek



Obsługa żądań

MODEL TRADYCYJNY

KLIENCI

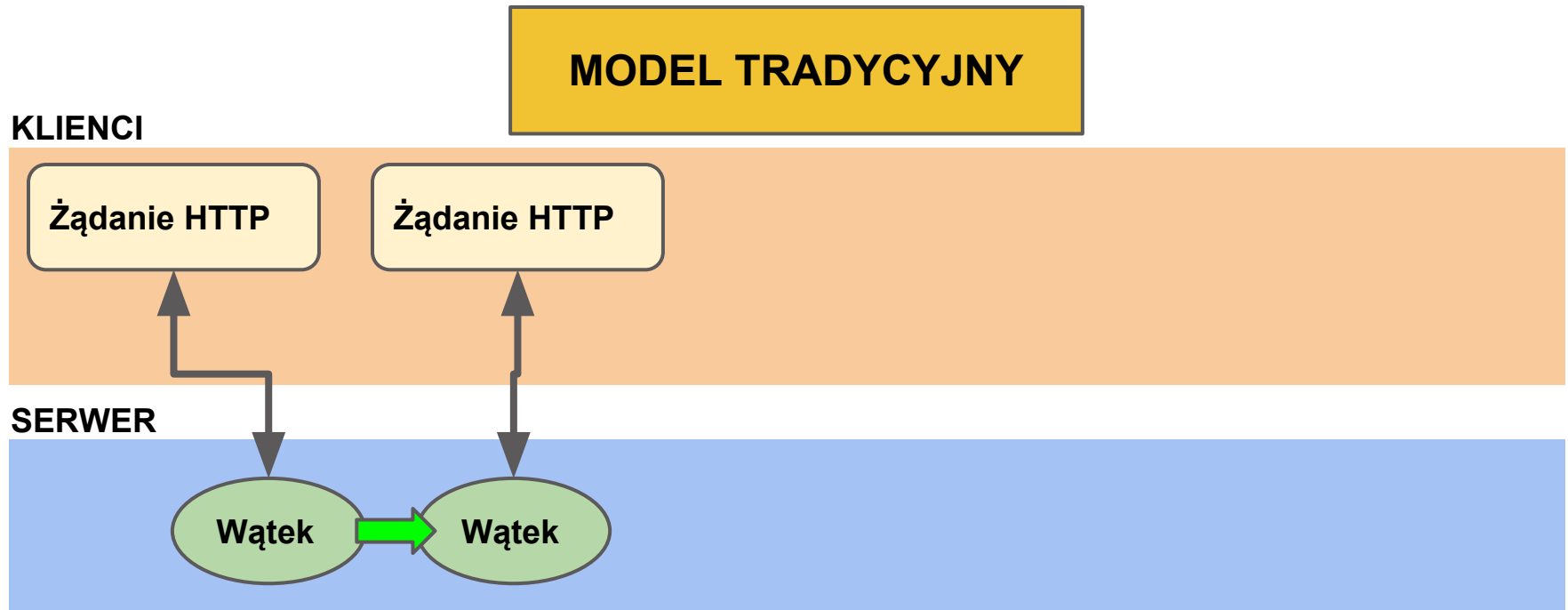
Żądanie HTTP

Żądanie HTTP

SERWER

Wątek

Wątek



Obsługa żądań

MODEL TRADYCYJNY

KLIENCI

Żądanie HTTP

Żądanie HTTP

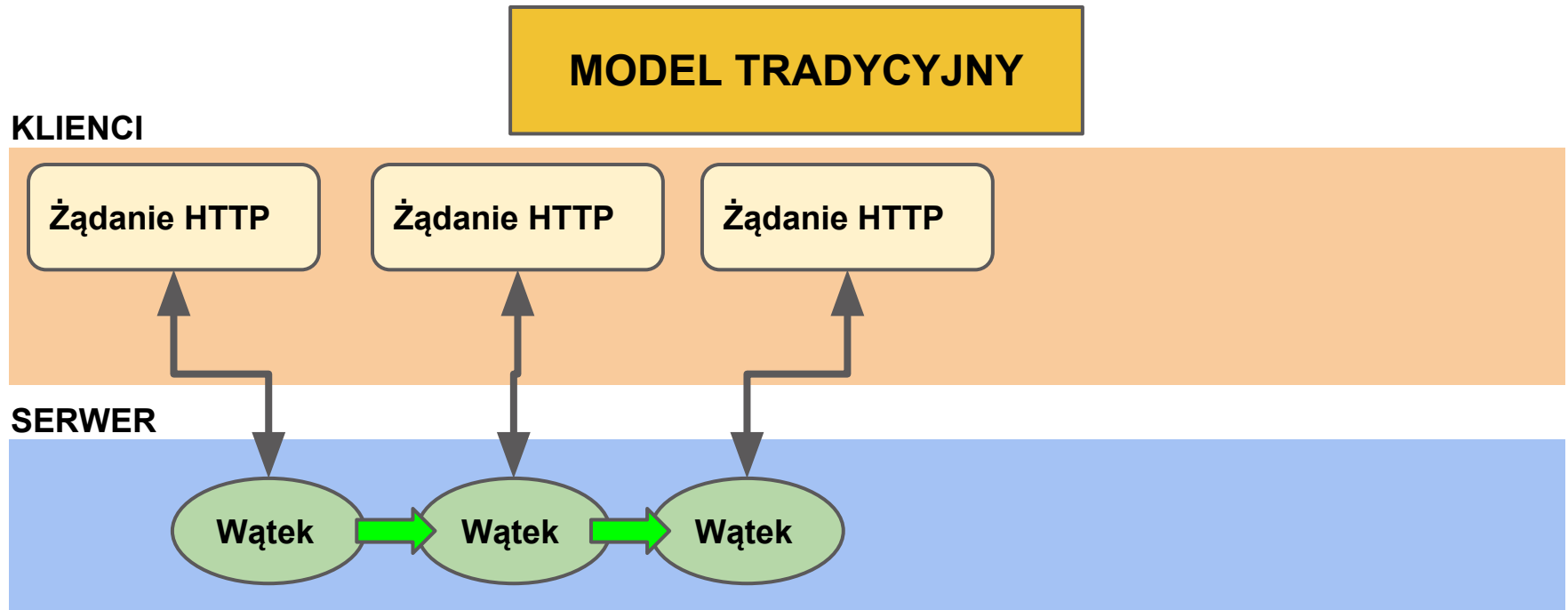
Żądanie HTTP

SERWER

Wątek

Wątek

Wątek



Obsługa żądań

MODEL TRADYCYJNY

KLIENCI

Żądanie HTTP

Żądanie HTTP

Żądanie HTTP

■ ■ ■ Żądanie HTTP

SERWER

Wątek

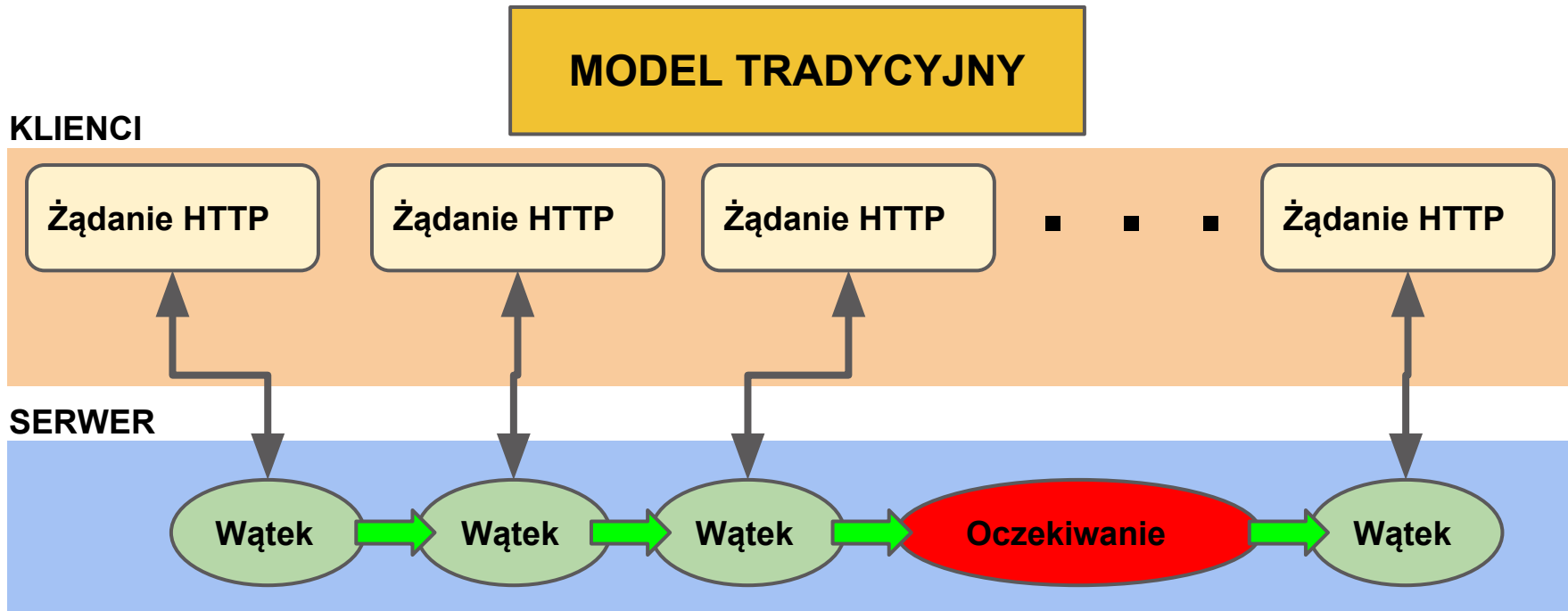
Wątek

Wątek

Oczekiwanie

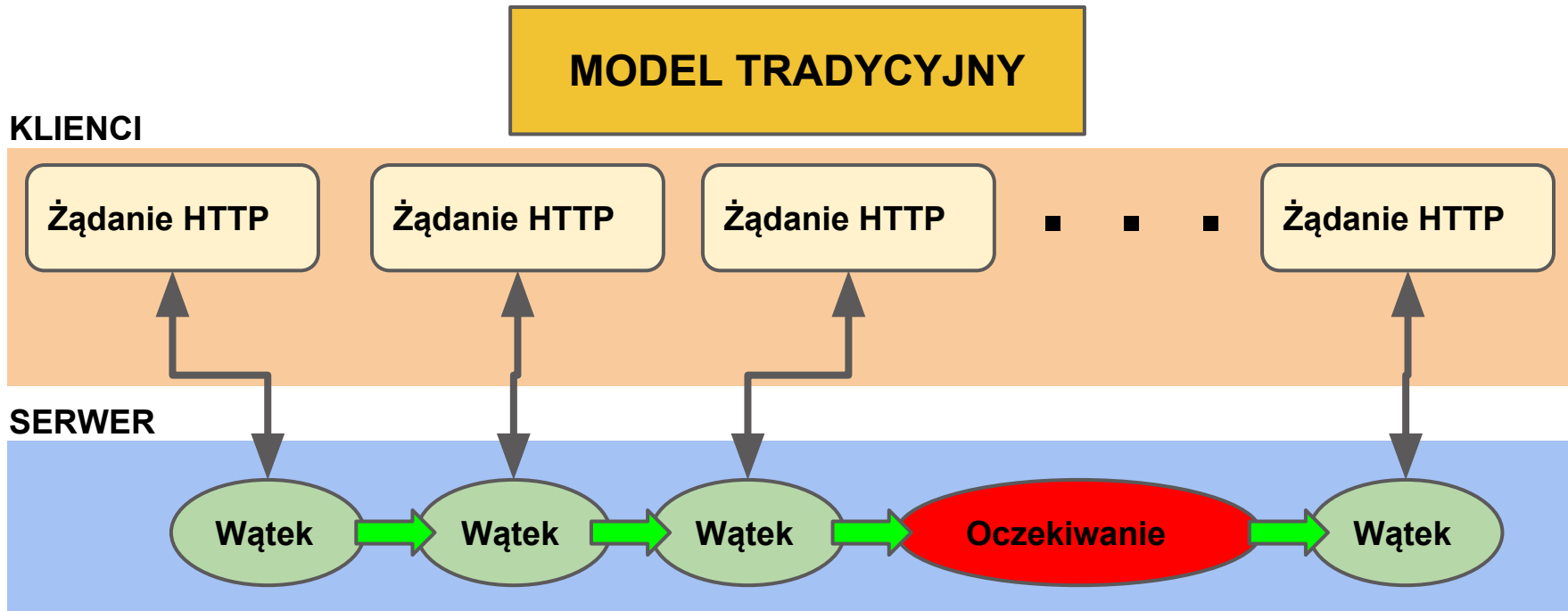


Obsługa żądań



W tym modelu serwer do obsługi każdego żądania musi stworzyć osobny wątek z ograniczonej puli jaką może obsłużyć CPU. Powoduje to że przy dużej ilości żądań niektóre z nich muszą czekać w "kolejce" na zrealizowanie. W wyniku czego serwer zaczyna działać wolniej co wydłuża czas oczekiwania na odpowiedź.

Obsługa żądań



Przykład:

Dla systemu z 8GB pamięci RAM przydzielającego 2MB pamięci na wątek możemy obsłużyć maksymalnie w tym samym czasie **4000 żądań** (w rzeczywistości jest to mniej ponieważ zużywamy jeszcze pamięć na inne operacje).

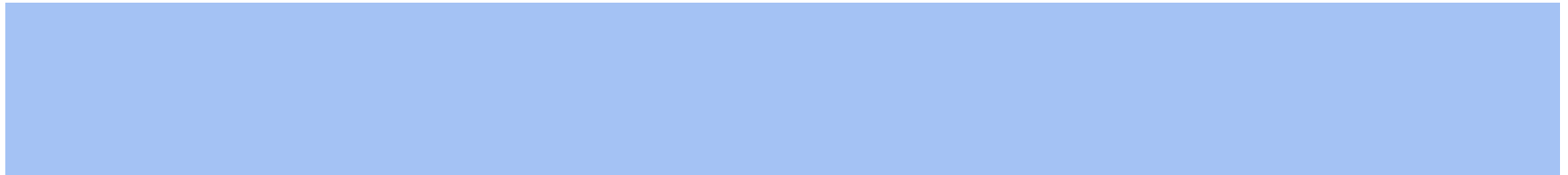
Obsługa żądań

MODEL ZDARZENIOWY

KLIENCI



SERWER



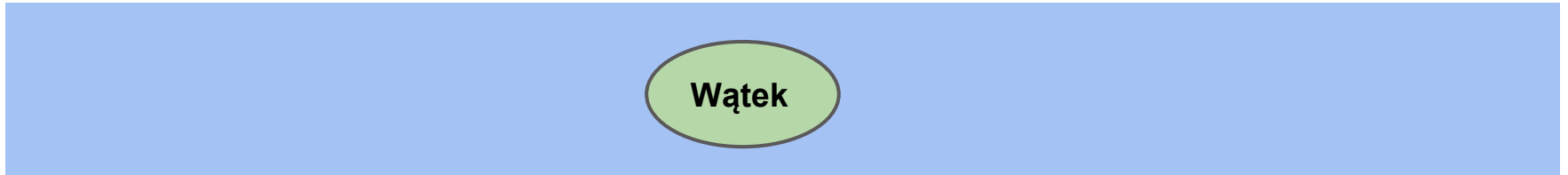
Obsługa żądań

MODEL ZDARZENIOWY

KLIENCI



SERWER



Obsługa żądań

MODEL ZDARZENIOWY

KLIENCI

Żądanie HTTP

SERWER

Wątek



```
graph TD; subgraph KLIENCI; direction TB; A[Żądanie HTTP]; end; subgraph SERWER; direction TB; B((Wątek)); end; A --> B;
```

The diagram illustrates the Event Model for request handling. It is divided into two horizontal sections: 'KLIENCI' (Clients) at the top, represented by an orange background, and 'SERWER' (Server) at the bottom, represented by a blue background. In the 'KLIENCI' section, there is a yellow rounded rectangle labeled 'Żądanie HTTP' (HTTP Request). In the 'SERWER' section, there is a green oval labeled 'Wątek' (Thread). A black arrow originates from the bottom of the 'Żądanie HTTP' box, points down, then turns right to point at the 'Wątek' oval, indicating the flow of the request from the client to the server's thread.

Obsługa żądań

MODEL ZDARZENIOWY

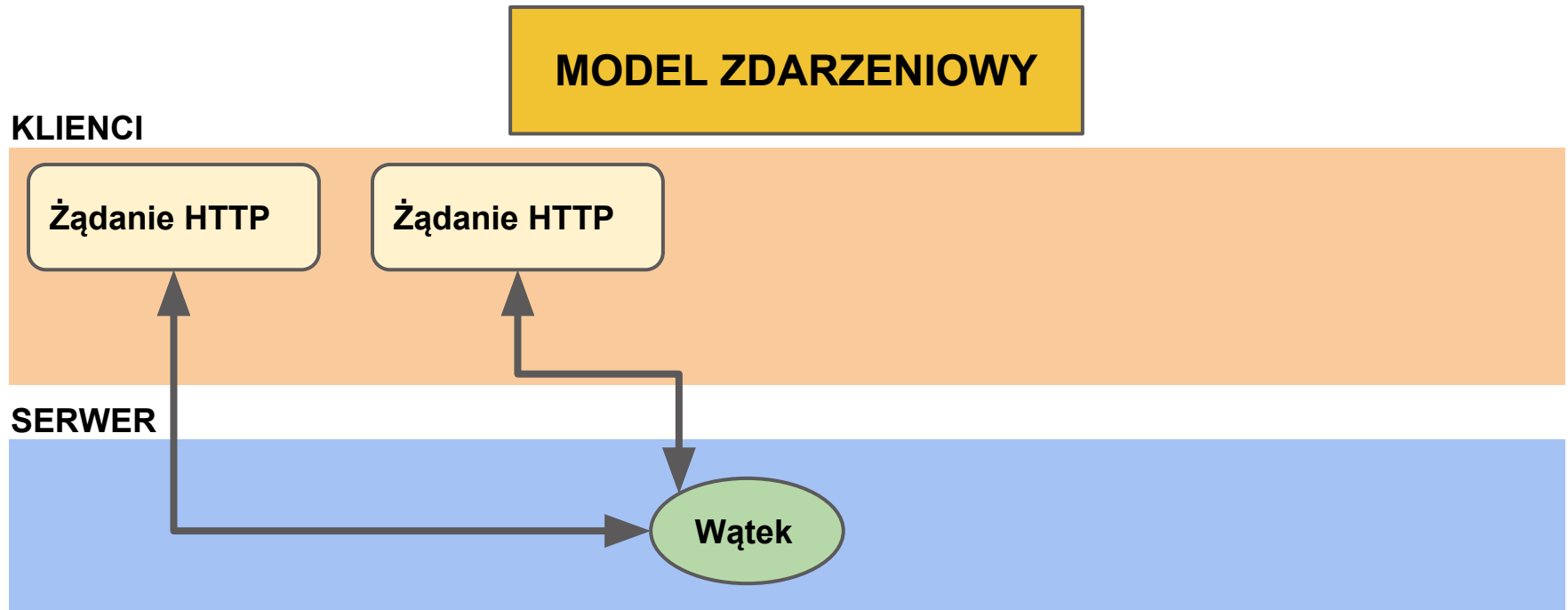
KLIENCI

Żądanie HTTP

Żądanie HTTP

SERWER

Wątek



Obsługa żądań

MODEL ZDARZENIOWY

KLIENCI

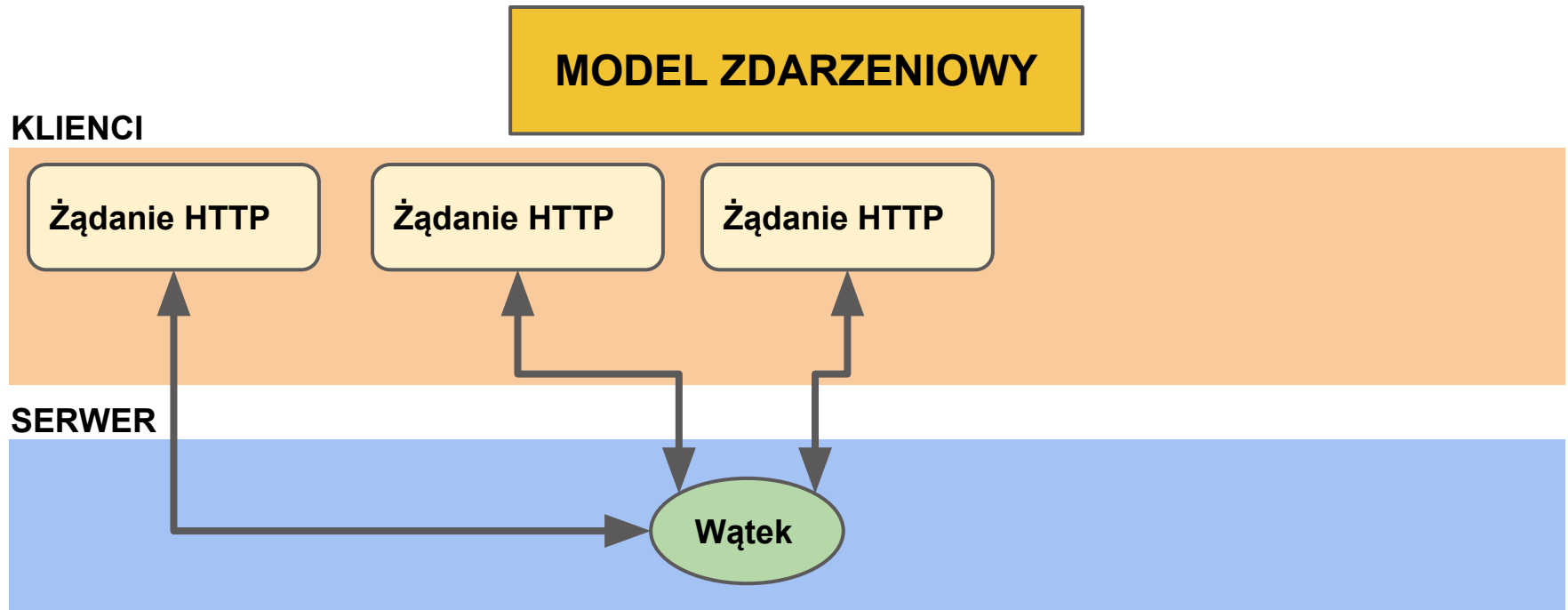
Żądanie HTTP

Żądanie HTTP

Żądanie HTTP

SERWER

Wątek



Obsługa żądań

MODEL ZDARZENIOWY

KLIENCI

Żądanie HTTP

Żądanie HTTP

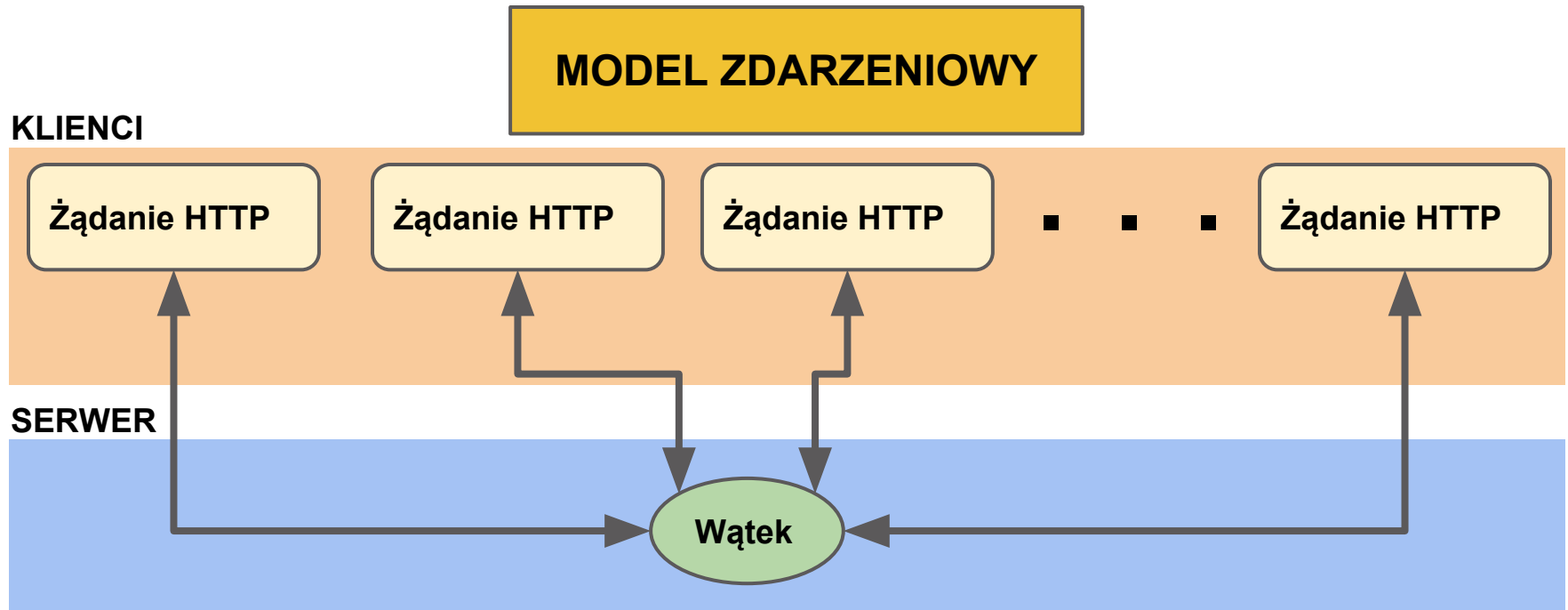
Żądanie HTTP

■ ■ ■

Żądanie HTTP

SERWER

Wątek



Obsługa żądań

MODEL ZDARZENIOWY

KLIENCI

Żądanie HTTP

Żądanie HTTP

Żądanie HTTP

■ ■ ■

Żądanie HTTP

SERWER

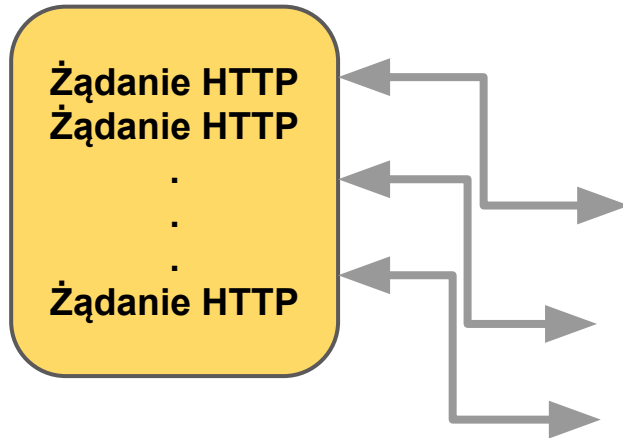
Wątek

W modelu zdarzeniowym Node.js wykorzystuje tylko jeden wątek do obsługi wielu żądań, oraz “pętłę zdarzeń” co powoduje że aplikacja taka jest bardzo wydajna i skalowalna. W praktyce przy żądaniach które nie wymagają złożonych operacji obliczeniowych można obsłużyć nawet do **1 miliona żądań** jednocześnie.

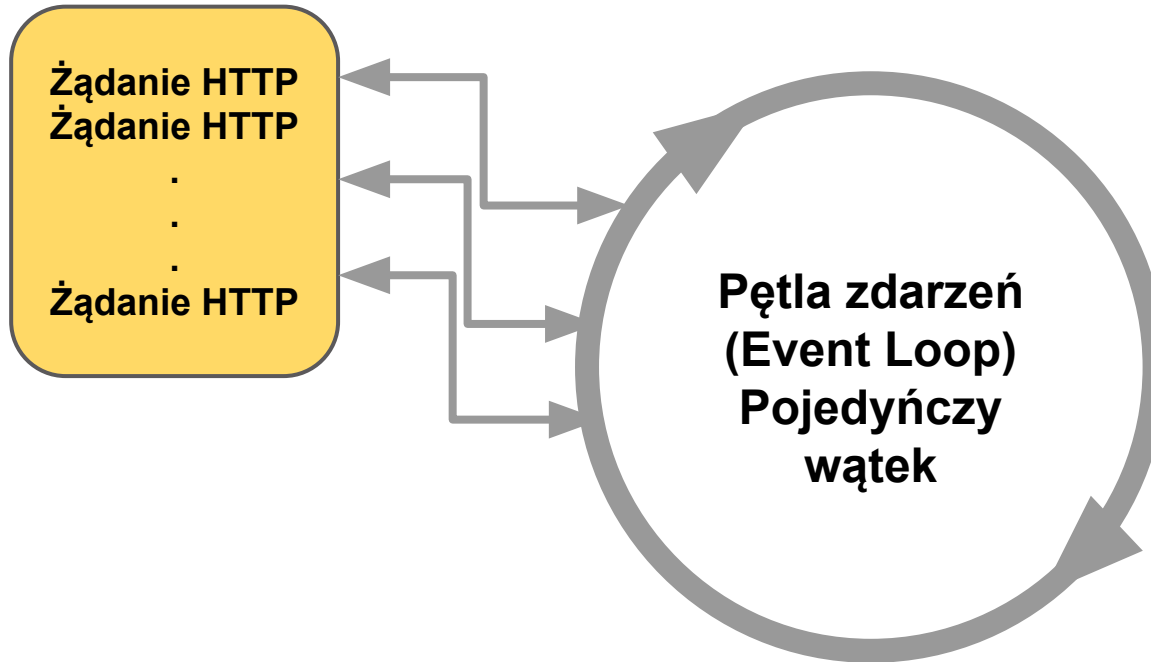
Pętla zdarzeń (Event loop)

Żądanie HTTP
Żądanie HTTP
.
.
.
Żądanie HTTP

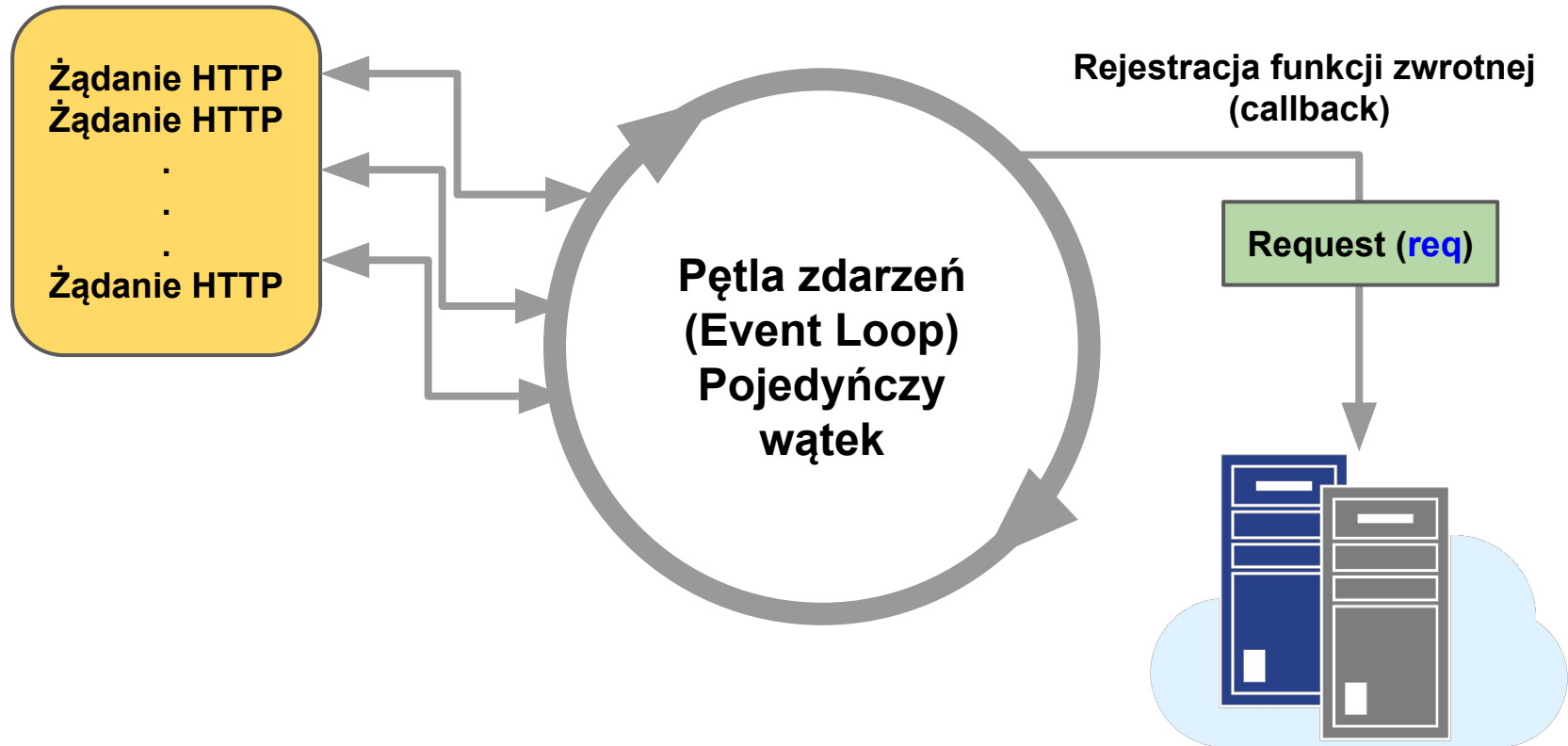
Pętla zdarzeń (Event loop)



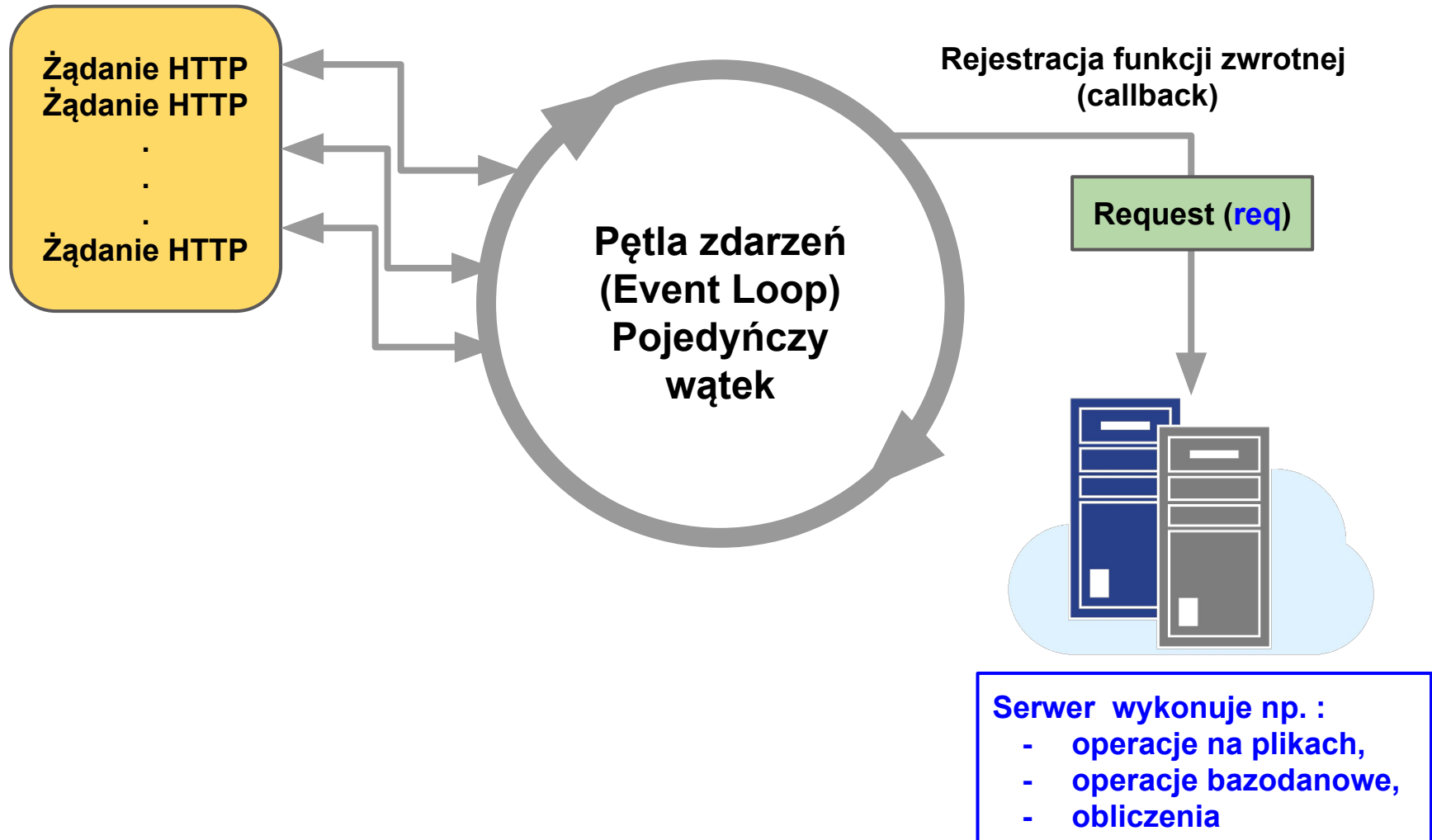
Pętla zdarzeń (Event loop)



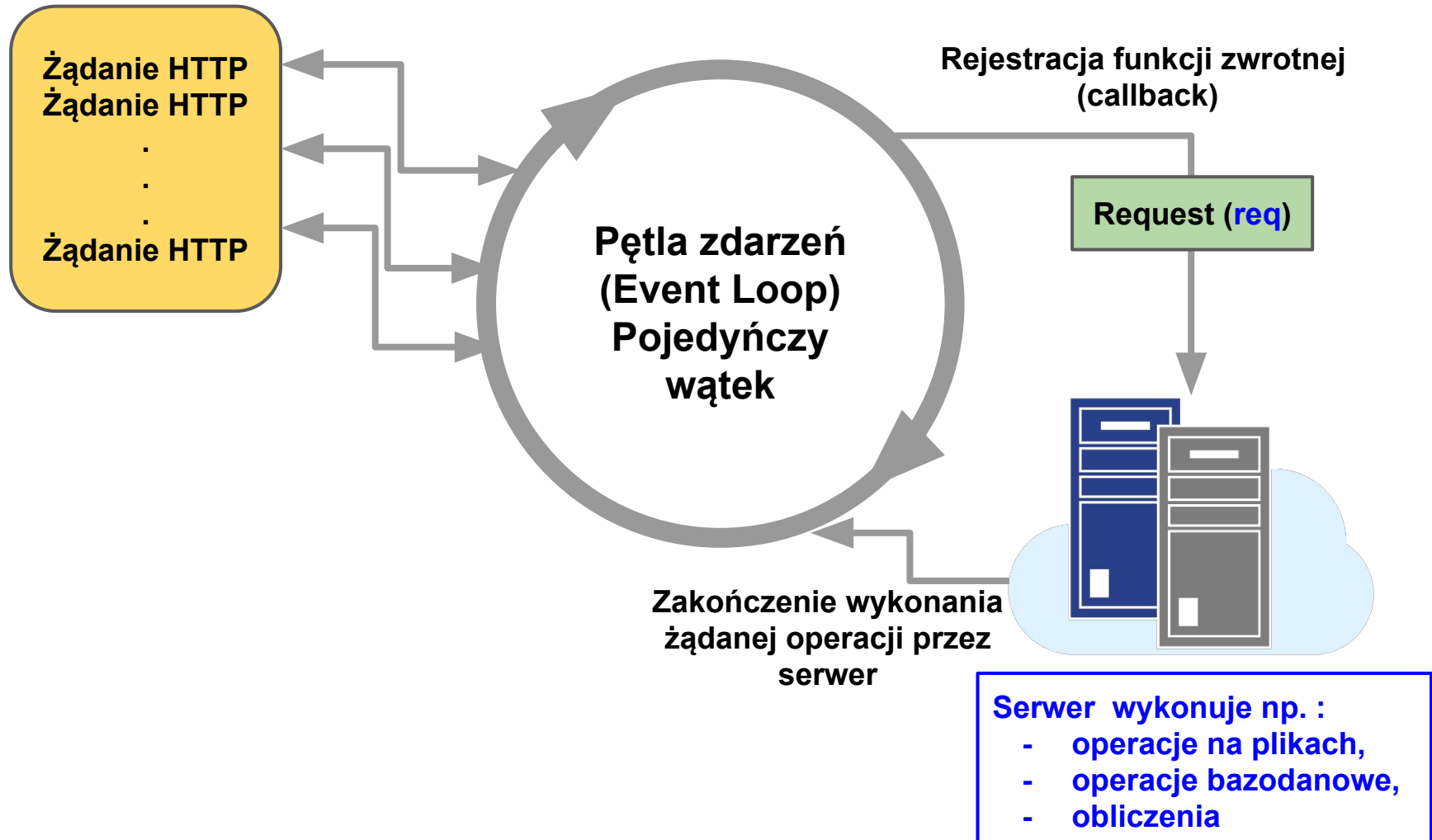
Pętla zdarzeń (Event loop)



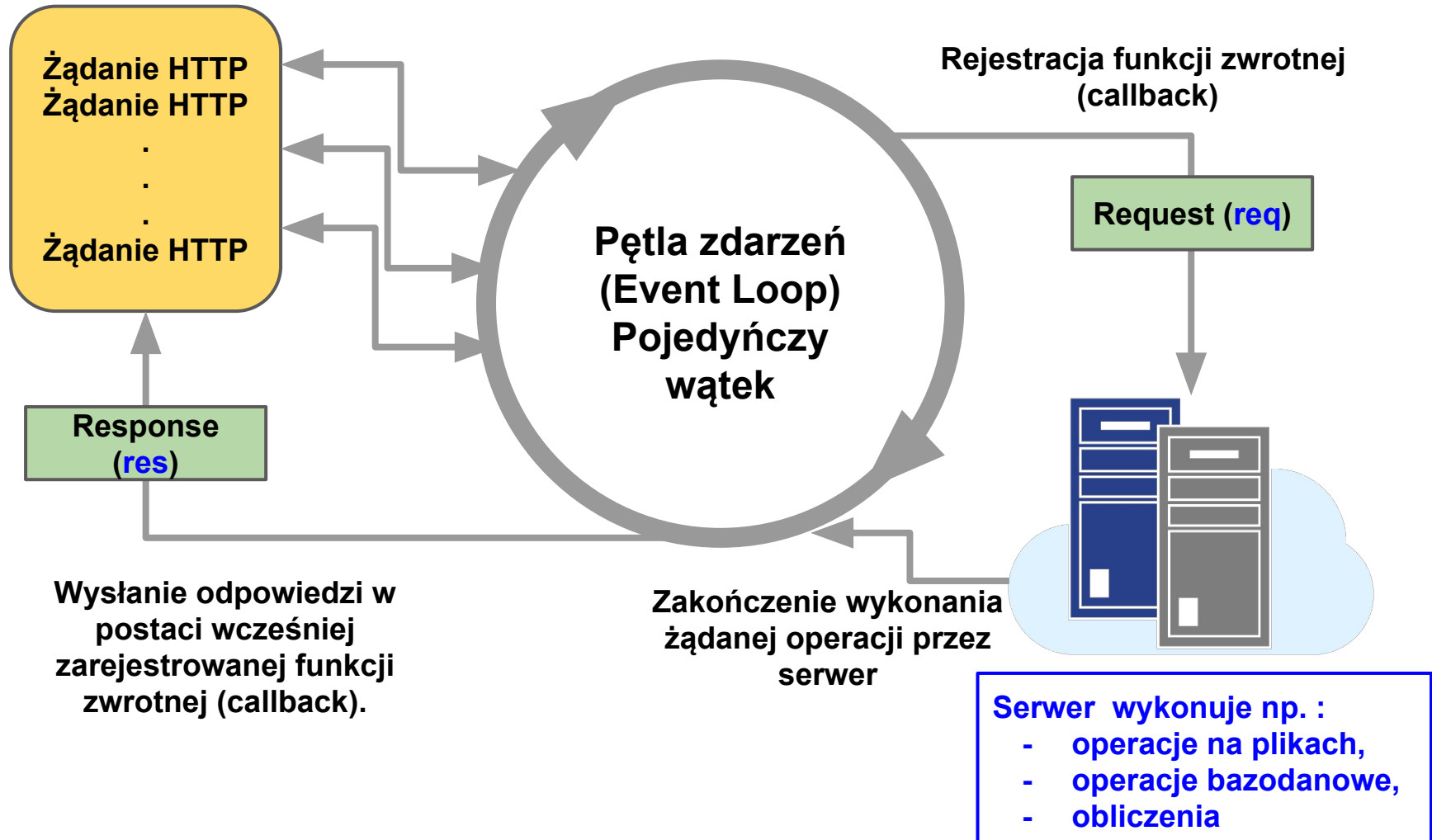
Pętla zdarzeń (Event loop)



Pętla zdarzeń (Event loop)



Pętla zdarzeń (Event loop)



KONIEC WYKŁADU 8
