JÖNKÖPING UNIVERSITY

School of Engineering

APPLICATIONS AND FRAMEWORKS

Server Side Web Development

TPWK16 Spring 2017

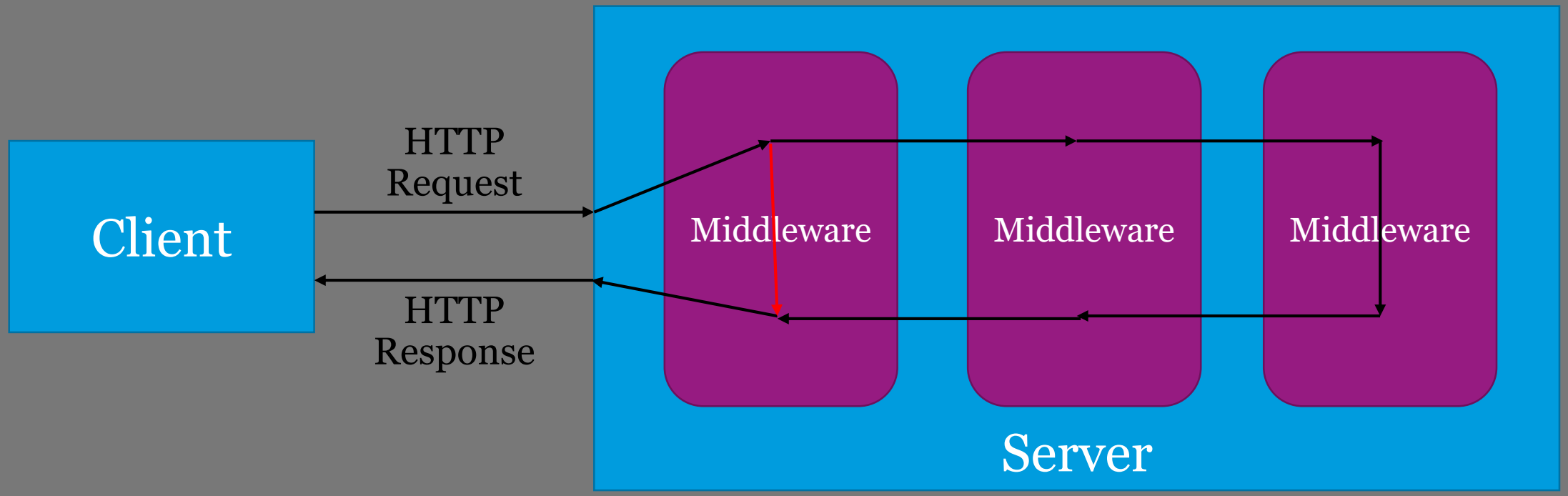
Peter Larsson-Green

JÖNKÖPING UNIVERSITY
School of Engineering

A BROADER POINT OF VIEW

- We use ASP.NET in this course.
- Companies also use other frameworks, such as:
  - Node.js (Express).
  - Python.
  - PHP (Slim).
  - Meteor.

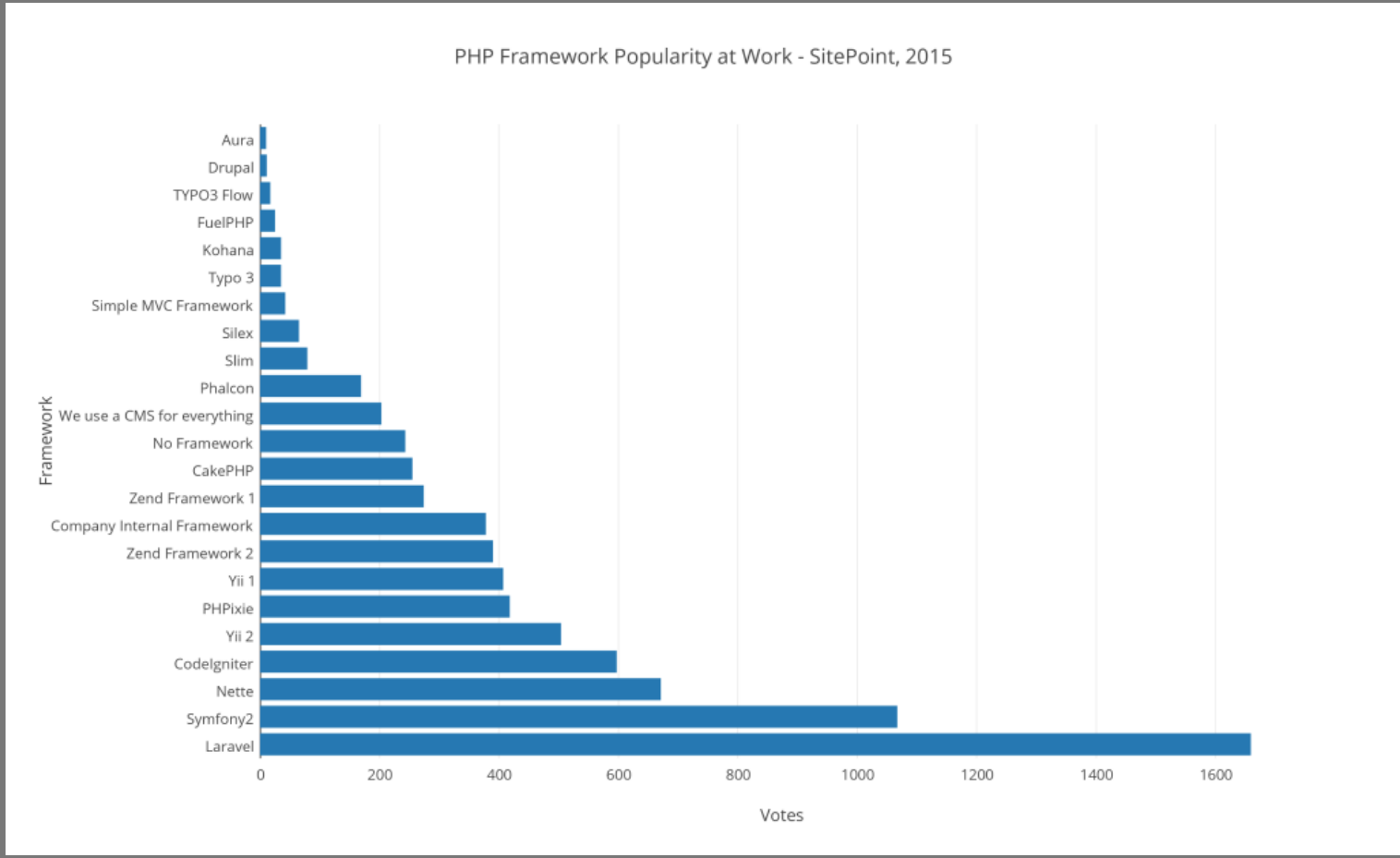
CONCEPT: MIDDLEWARE



PHP - PHP: HYPERTEXT PREPROCESSOR

- Before sending the hypertext to the client, process it!
  - Just as in Classic ASP.
- File extension: `.php`
- Put PHP code between `<?php` and `?>`.
- Version 1 released 1995.
  - First website on the web: 1990.
- Current version: 7 (released December 2015).
- Classes added in version 4.
- Facebook still uses PHP.
- "PHP won the Web Platform War".
  - https://www.phpclasses.org/blog/post/208-5-Reasons-Why-the-Web-Platform-War-is-Over-PHP-Won-with-75-says-Google.html

PHP FRAMEWORKS



http://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/

METEOR

Some characteristics:

- Single page application.
- Reactive programming on client.
- Single programming language: JavaScript.
- Database everywhere.
- Latency compensation.

SINGLE PAGE WEB APPLICATIONS

`GET / HTTP/1.1` → Gives you everything you need (except the data).

- Routing? Handled on the client!
- Views put in `.html` files.
  - Not ordinary HTML files, but spacebars files.
  - Inspired by the handlebars syntax.

SPACEBARS

```
<head>
  <!-- This will be inserted into the <head>-element. -->
<head>


<body>
  <!-- This will be inserted into the <body>-element. -->
</body>
```

SPACEBARS

```
<template name="message">
  <h1>Hi {{to}}</h1>
  <p>The message is: {{message}}</p>
</template>

<body>
  {{> message to="The King" message="Good job!"}}
  {{> message to="The Servant" message="Bad job!"}}
</body>
```

TEMPLATE LIFE CYCLES

```
<template name="message">
  <h1>Hi {{to}}</h1>
  <p>The message is: {{message}}</p>
</template>
```

```
Template["message"].onRendered(function(){
  this.find("h1").style.color = "red"
})
```

```
Template["message"].onCreated(function(){ })
```

```
Template["message"].onDestroyed(function(){ })
```

TEMPLATE HELPERS

```
Template["listHumans"].helpers({
  humans: [
    {age: 10, name: "Alice"},
    {age: 15, name: "Bea"}
  ]
})
```

```
<template name="listHumans">
  <ul>
  {{#each humans}}
    <li>{{name}} is {{age}} years.</li>
  {{/each}}
  </ul>
</template>
```

TEMPLATE EVENTS

```
<template name="counter">
  Number: <span>0</span><br>
  <button>Inc!</button>
</template>
```

```
Template["counter"].events({
  'click button': function(event, template){
    var span = template.find('span')
    var counter = parseInt(span.innerHTML)
    span.innerHTML = counter + 1
  }
})
```

REACTIVE PROGRAMMING

Pseudo code

```
a = 1
b = 2
c = a + b
print(c) // Prints 3.
a = 2
print(c) // Prints 4!
```

Meteor

```
var a = new ReactiveVar(1)
var b = new ReactiveVar(2)
var c
Tracker.autorun(function(){
   c = a.get() + b.get()
})
console.log(c) // Logs 3.
a.set(2)
console.log(c) // Logs 4!
```

TEMPLATES ARE REACTIVE

```
Template["counter"].onCreated(function(){
  this.counter = new ReactiveVar(0)
})
Template["counter"].helpers({
  counter: function(){ return this.counter.get() }
})
Template["counter"].events({
  'click button': function(event, template){
    template.counter.set(template.counter.get() + 1)
  }
})
```

```
<template name="counter">
  Number: <span>{{counter}}</span><br>
  <button>Inc!</button>
</template>
```

SINGLE PROGRAMMING LANGUAGE

Write JavaScript code that runs on the client.	`/client/file.js`
Write JavaScript code that runs on the server.	`/server/file.js`
Write JavaScript code that runs on both!	`/file.js`

CLIENT → SERVER COMMUNICATION

/server/file.js

```
var numberOfCalls = 0
Meteor.methods({
  sum: function(a, b){
    numberOfCalls++
    return a + b
  }
})
```

/client/file.js

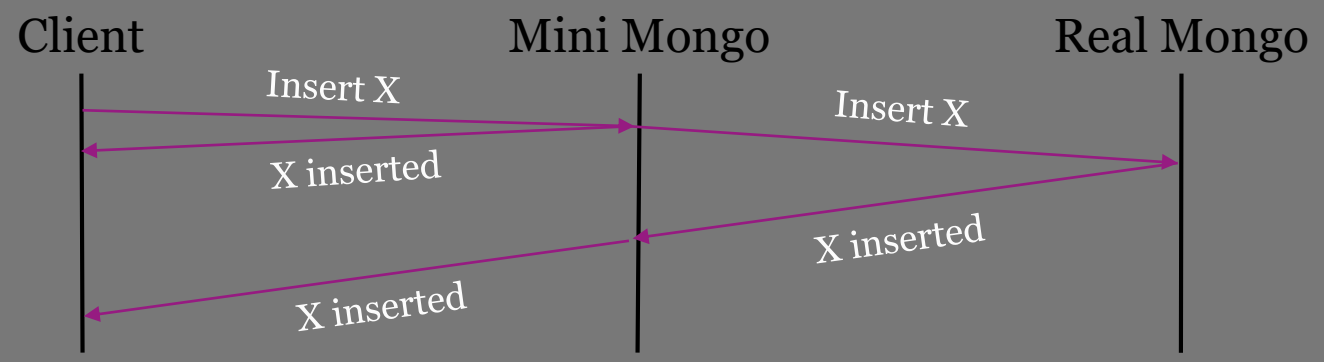
```
Meteor.methods({
  'sum': function(a, b){
    sum = a + b
  }
})
var sum
Meteor.call('sum', 1, 2, function(err, res){
  sum = res
})
```

Latency compensation!

DATABASE EVERYWHERE

Meteor supports MongoDB.

- A NoSQL database.
- The database consists of collections...
  - ..which in turn consists of documents.
- Mini Mongo on the client emulates the database on the server.
  - Latency compensation!



PUBLISHING RECORDS

/server/file.js

```
var Humans = new Mongo.Collection("humans")
Meteor.publish('allHumans', function(){
  return Humans.find()
})
```

/client/file.js

```
var Humans = new Mongo.Collection("humans")
Meteor.subscribe('allHumans')
```

ALLOWING CLIENT OPERATIONS

/server/file.js

```
var Humans = new Mongo.Collection("humans")
Humans.allow({
  insert: function(userId, doc){
    return doc.name != "" && 0 < doc.age
  }
})
```

Similar for update and remove!