Software Engineering 2013

Keio University, SFC, Shuichi Kurabayashi

Version 2.5

Introduction

- In this class, you will learn practical and effective software development techniques.
- Programming is a technique for **transforming an objective into a structure**.
  - It is necessary to think in a highly abstract way, but write using a low degree of abstraction.
- In this class, we will learn techniques for reusing software modules, and a practical software development technique known as software design patterns with HTML5 + JavaScript as the focus.

Characteristics of this Lecture

- Using JavaScript, we will learn design patterns, as well as software architectures and the fundamental skills for mid-level/large-scale software development, using typical web application creation as an example.
- As an example, we will not only explain basic web applications but also multimedia data processing, which is made possible by HTML5 and its relationship with design patterns.

Next-Generation Fundamental Technologies

- Smart devices change the end-user computing environment
- Massively parallel computing by GPGPU
- Next-generation GUI environment by HTML5 + CSS3 + JavaScript

Smart Phone

(Hardware Platform)

- End-user computing environment shifts from PC to smart terminals

HTML5 +

JavaScript + CSS3

(Software Platform)

- An interactive web environment = the spread of web applications

GPGPU & FPGA

(Non-CPU Hardware)

- Massively parallel computing due to GPGPU, which is 200 times faster than traditional CPU, revolutionizes multimedia data processing

Next-Generation Fundamental Applications

- The realization of data processing, semantic analysis, and information-sharing functions, the object of which is "dynamic data" (dynamic images, music, and sensor data) endowed with a sense of narrative, and accompanying spatial and temporal changes

Visualization	• Spatial and temporal changes in "dynamic data," and the intuitive display of a sense of narrative.
Personalization	• Selective distribution of information responding to individual user context
Collective Intelligence	• Data mining is applied to historical information accumulated by the user, and significant knowledge is acquired

The Next Generation Web's Important Elemental Technology

Combining the following four, WebSocket + Web Workers + WebGL + WebCL JavaScript, obtains a higher speed calculation ability than C++.

Web Socket	Realizes bidirectional real-time transmissions (full duplex transmission) between the client and the server
Web Worker	In Javascript, the realization of a program's parallel execution
WebGL	Makes it possible to use accelerated 3D CG on a browser through the use of hardware
WebCL	Realizes massively parallel computing (executing at the same time dozens to several hundreds of calculations)

WebGL
OpenGL ES 2.0 for the Web

- WebGL is a formal specification for displaying 3D computer graphics on a web browser.
- WebGL makes it possible to display 3D graphics, accelerated using hardware, on a platform that supports 3DCG specifications called OpenGL ES 2.0, without a special browser plug-in.
- It is supported on major platforms such as Windows, MacOSX, iOS, and Android.

WebCL
GPU Computing on the Web

- WebCL implements, by binding JavaScript and OpenCL (Open Computing Language), high-calculation load parallel program development and execution.
- As yet, it remains unimplemented on the browser, and plug-ins for testing are currently being offered.
- Nokia's Firefox Add-on: http://webcl.nokiaresearch.com/
  - Samsung's WebKit Expansion : http://code.google.com/p/webcl/

```
__kernel void vector_add(
    __global int *p_vec_in1_gpu,
    __global int *p_vec_in2_gpu,
    __global int *p_vec_out_gpu,
    int vec_dim) {
    int tid = get_global_id(0);
    if(tid < vec_dim) {
        p_vec_out_gpu[tid] =
        p_vec_in1_gpu[tid] +
        p_vec_in2_gpu[tid];
}}
```

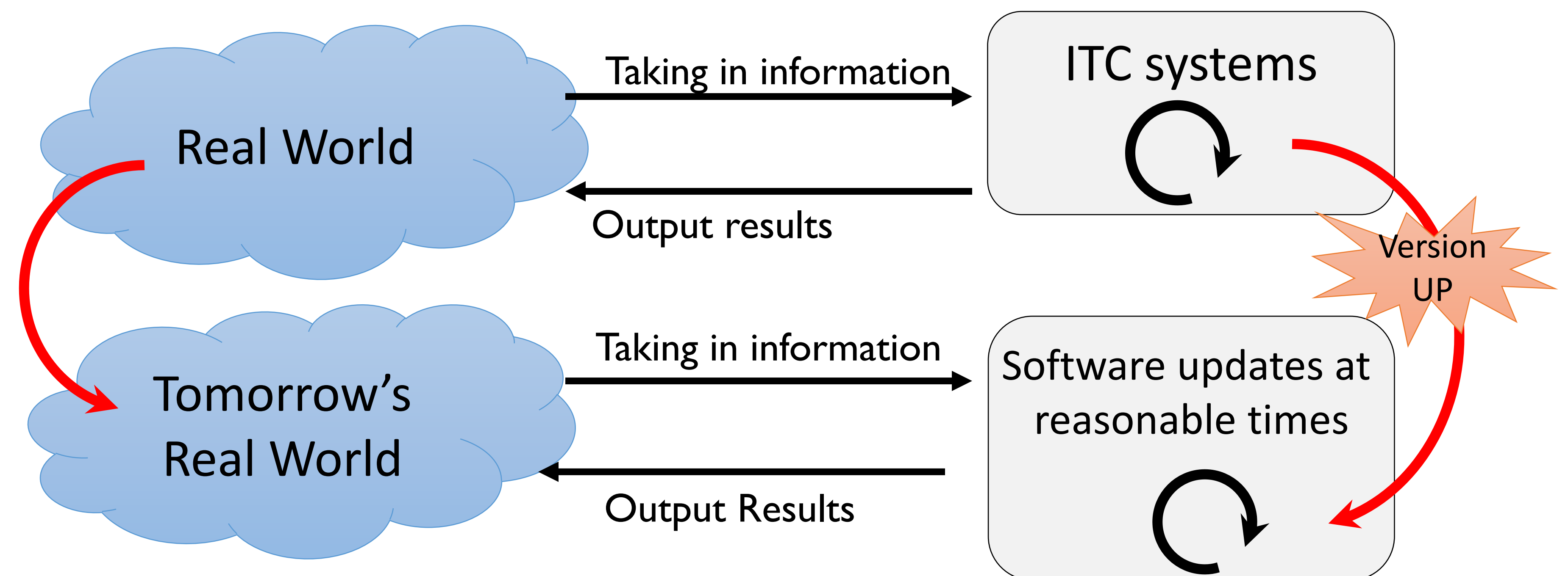
Open CL Language

References

- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, “Design Patterns Elements of Reusable Object-Oriented Software,” Soft Bank Publishing, 1995, ISBN 978-4797311129.
- Meyer, Bertrand (2000), “Object-Oriented Software Construction,” Prentice Hall. ISBN 978-0136290315.
- Crockford, Douglas (2008), “JavaScript: The Good Parts,” O’reilly Japan, ISBN 978-4873113913.
- Stefanov, Stoyan (2010), “JavaScript Pattern,” O’reilly Japan, ISBN 978-4873114880.
- Fifth Edition of ECMA-262, http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf

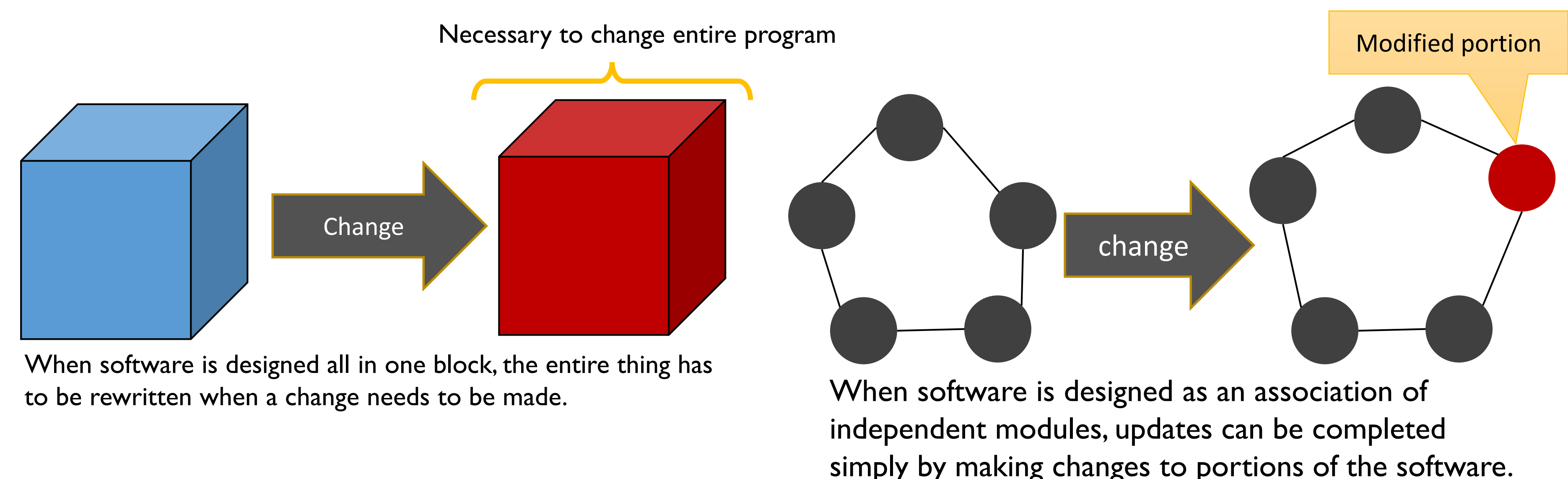
IT System Basics

- Basic functions of a computer
  - Input (taking in information), Calculation, and Output (implementing results)
- If the system is not updated to respond to changes in the real world, it is impossible to obtain a beneficial output.



Modularization

- If software can be realized as an association of independent modules, one can design software that responds well to change.
- Programming languages are full of functions that support “modularization,” and it is important to understand these various functions.



Software Engineering at SFC

- Learning software engineering as practical learning
  - We will learn software engineering not through mere theory, but through learning methods for creating software that has a "high level of reusability and responds well to change" while actually in the process of creating software.
  - We will learn how to use "tools" in order to efficiently develop software.
- Learning the ability to design typical information systems.
  - Ability to create a real-world model
  - Ability to design a system from a model
  - Ability to implement the designed system while making full use of the tools.

Schedule

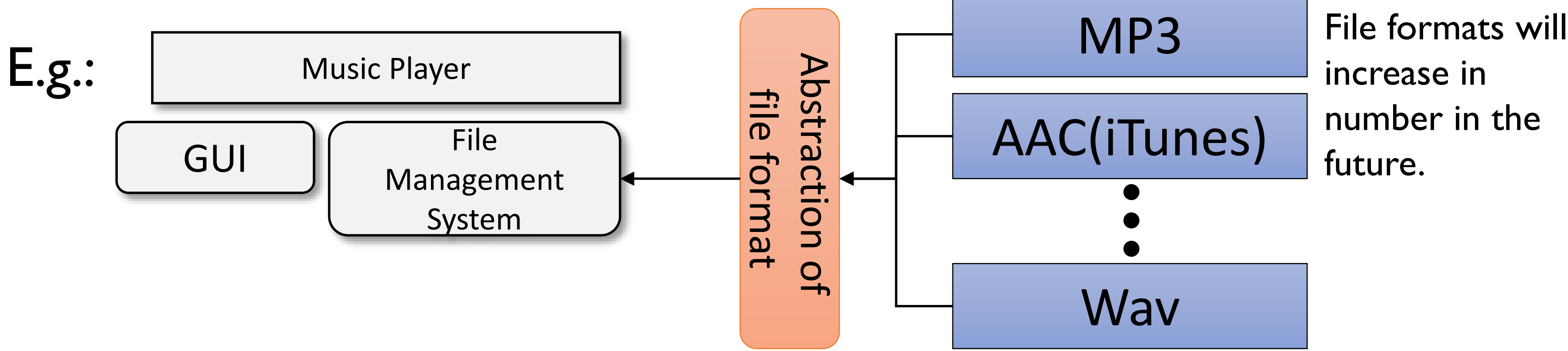
- Overview: Clarifies the position of this lecture, contents, how to give lectures, and tools required in this class.
- Introduction to Software Engineering: Explains the basic idea of making a large-scale software development. Describes the design of large-scale software, development, and maintenance, to understand the software development cycle.
- Object-Oriented Programming: As the basis for software development, learn object-oriented programming. using the JavaScript language.
- Object-Oriented Programming: As the basis for software development, learn object-oriented programming. using the JavaScript language.
- Design Pattern: Learn about object-oriented design patterns which is a "patterns for typical problem solving." Learn the pattern of the building user interface called MVC and (Model-View-Control) pattern, and build your own web application based on MVC.
- Design Pattern: Learn design patterns, using JavaScript, which is a prototype-based object-oriented programming language. In particular, describe the prototype chain mechanism, which is a software reuse mechanism in the prototype-based object-oriented language. Learn Chain of Responsibility pattern and Iterator pattern.
- Practice for midterm report: Build Web applications using design patterns to practice on how to implement an efficient client / server model. We will give an intermediate assignment to design a MVC-based web application.

Schedule

- Presentation: Oral presentation of intermediate assignment.
- Server Development with JavaScript: learn the basic skills of medium and large scale software development, through building a full-fledged Web applications by combining HTML5 and node.js, which is a server-side JavaScript engine.
- AJAX, WebS Socket, and Design Pattern using JavaScript: Learn how to apply design patterns into AJAX (Asynchronous JavaScript + XML) and WebSocket, for performing asynchronous communication in a web browser, which is a core technology of the Web site called Web2.0. Describes how to effectively implement AJAX functionality using a design pattern.
- Advanced Web Applications and Design Patterns: As advanced topics, this course deals with the design patterns for multimedia data processing in HTML5, including video element and canvas element.
- Final Reports: Design and implementation of software as a final project, develop your own software by using of object-oriented programming, design patterns, software module reuse techniques. In final project (1),you design your own software, and implement it using JavaScript.
- Final Reports: Design and implementation of software as a final project, develop your own software by using of object-oriented programming, design patterns, software module reuse techniques.
- Final Presentation and Summary: Made a presentation of the final project. In addition, we consider the outlook for the state of software engineering in the future.

First of all, what is a program?

- Programming is the management of the state of affairs based on future predictions.
  - Program = Pro (in advance) + Gram(to describe)
  - Programming describes what you should do in advance, so as not to fall into confusion during the actual event (which is essentially the management of the state of affairs).
  - When you program, you are thinking about the future.
- Regardless of computers, the basic of programming is abstraction.
  - If you find yourself captivated by tangible things, you cannot predict the future.
  - Abstraction allows you to handle uncertain elements.



Design Pattern: From construction technique to software development technique

- The architect Christopher Alexander proposed, as an architectural technique, the idea that building and city design “should be created through a combination of various fragments of past designs.”
- Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides applied the idea that designs “should be created through a combination of various fragments of past designs” as a software development technique.
    - Doctoral Thesis：「Object-Oriented Software Development based on ET++: Design Patterns, Class Library, Tools 」

Design Patterns as a Good Practice for Abstraction

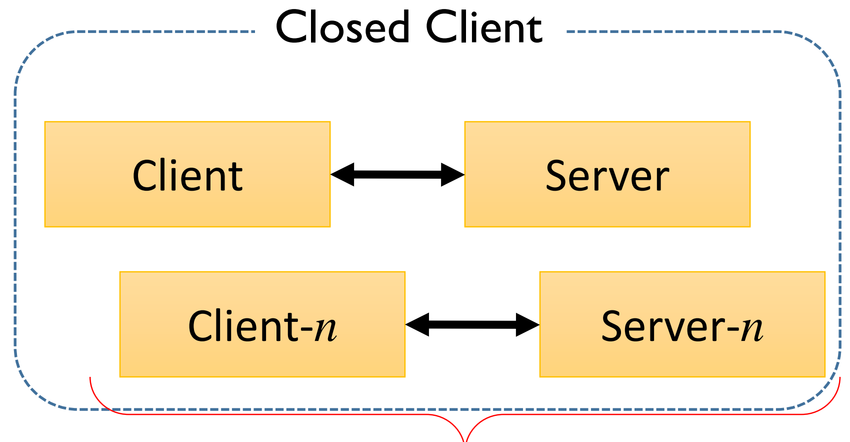
- Generic design patterns that can be reused in a variety of programs: Design Patterns
  - “In computer programming, there is a significant difference in terms of productivity between a novice and an expert; however, a significant portion of that difference is due to the difference in experience. Experts have faced and overcome various difficulties numerous times. When these experts tackle a similar issue, they typically arrive at a solution with same pattern. These are design patterns.” (Cited from the GoF book)
- Most famous Bible: “GoF Design Patterns”
  - Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides collected design patterns useful for object-oriented programming and published them as “Design Patterns: Elements of Reusable Object-Oriented Software (SoftBank Publishing Edition)”
- Design Patterns are Language Neutral
  - Among designs that most often use design patterns, Java class libraries are the most famous. However, design patterns themselves are not dependent on any specific language.

Main Design Patterns

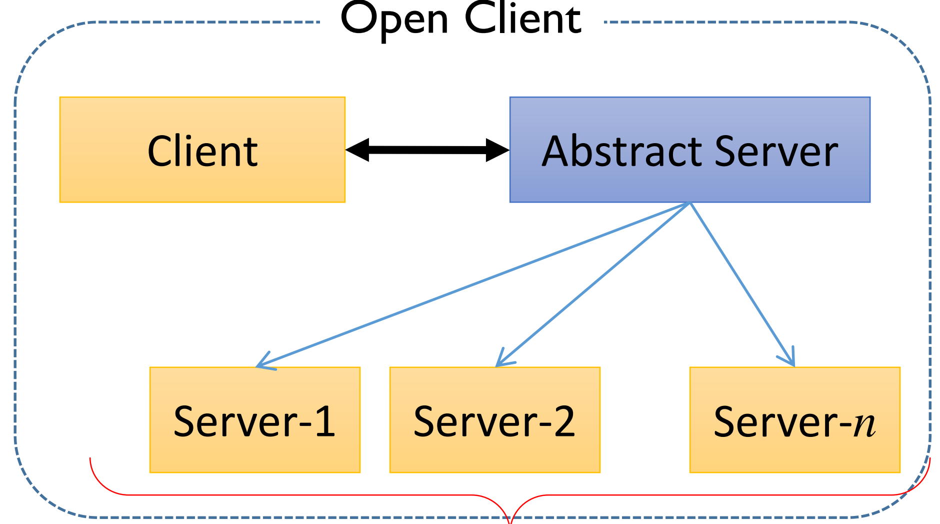
- Iterator
- Adapter
- TemplateMethod
- FactoryMethod
- Singleton
- Prototype
- Builder
- AbstractFactory
- Bridge
- Strategy
- Composite
- Decorator
- Visitor
- ChainOfResponsibility
- Facade
- Mediator
- Observer
- Memento
- State
- Flyweight
- Proxy
- Command
- Interpreter

Open/Closed Principle

- Modules must be open for “extension,” and closed for “modification.”
- Modules are connected through an abstract interface.



In response to modifications in the server module, the client module must also be modified.



In response to extension of the server module, the client module does not need extension (it is open).

Model–View–Controller Model

The MVC model is a concept utilized in a GUI design written in an object-oriented language known as Smalltalk. It is the most fundamental architecture when realizing the "open/closed principle," a principle with features similar to the GoF's design patterns, for GUI.

Model

Model defines data structure.
Model is independent of the View or Controller

View supplies a "way of showing" data.
Corresponds to GUI components

View

Controller

Controller supplies the "method of operation."
Corresponds to processing such as the operation when a button is clicked

Conventional Client/Server Architecture

- Server that collects information
- Client that implements application logic (status of the process)
- Composed of the above two layers

Database Server

Client

Client

Client

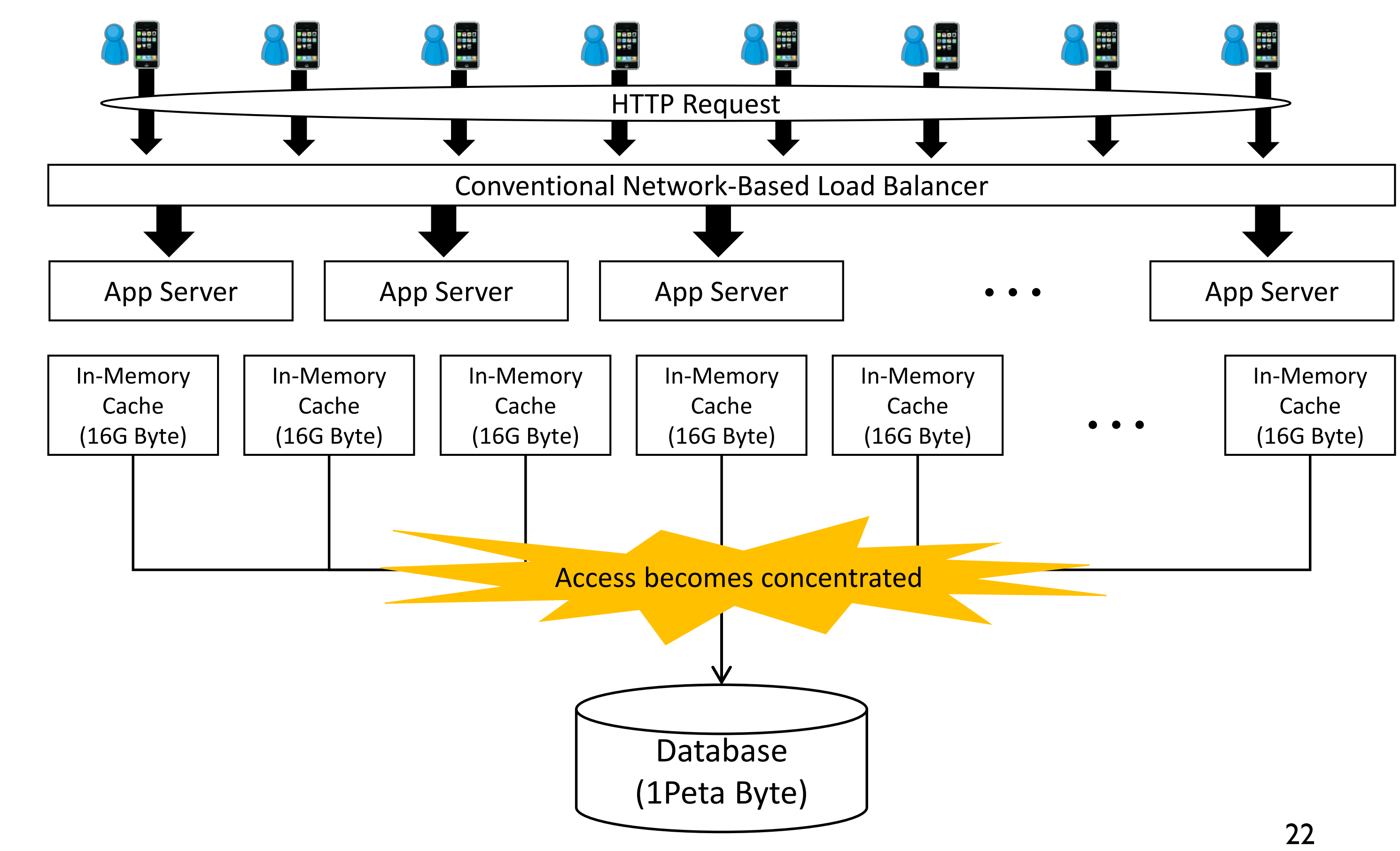
Data storage becomes bottlenecked in the three-level model

Many connections from clients are handled by the load balancer.

App Servers are distributed and highly scalable.

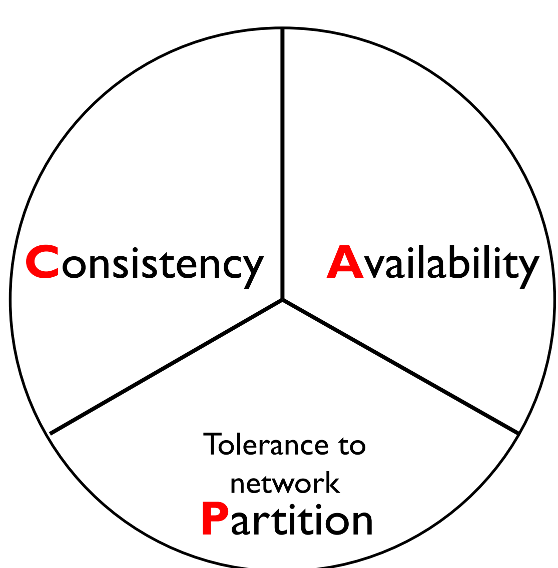
Cache is distributed but must be synchronized with data store

To keep consistency of data, data store can be the bottleneck.



CAP Theorem

- The CAP Theorem according to Dr. Brewer of UC Berkeley states the following:
- C: Consistency (consistency of data)
  - After data has been updated, if something else references that data, it will always be guaranteed that the updated data can be referenced.
- A: Availability (availability of the system)
  - No matter what the current situation, the system will continue to operate.
- P: Tolerance to network partitions (tolerance to network partitions)
  - Data can be distributed into multiple nodes, allowing for data reference/updates from other nodes (replication), even in the cases where there is malfunction in an individual node and damage to the data.
- Out of these three guarantees, only two can be fulfilled at a time.



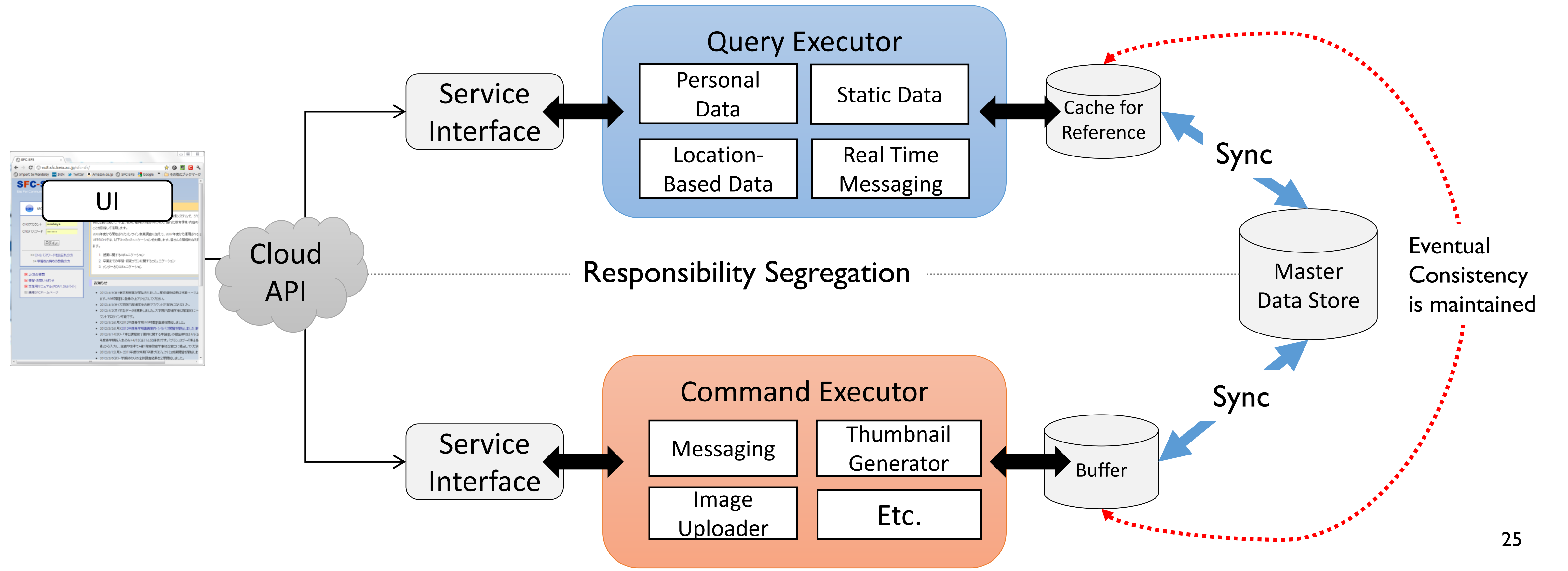
Availability and Service Level Agreement

- The facility to be able to continuously keep a system in operation.
  - In the cloud, the numerical value of availability is regulated through the Service Level Agreement (SLA)
- Representation of the Quantitative Value of Availability: Rate of Operation
  - The rate of operation is the uptime divided by the sum of the uptime and downtime

Rate of Operation	Yearly Downtime
99.9999%	32 s
99.999%	5 min 15 s
99.99%	52 min 34 s
99.9%	8 hr 46 min
99%	3 days 15 hr 36 min

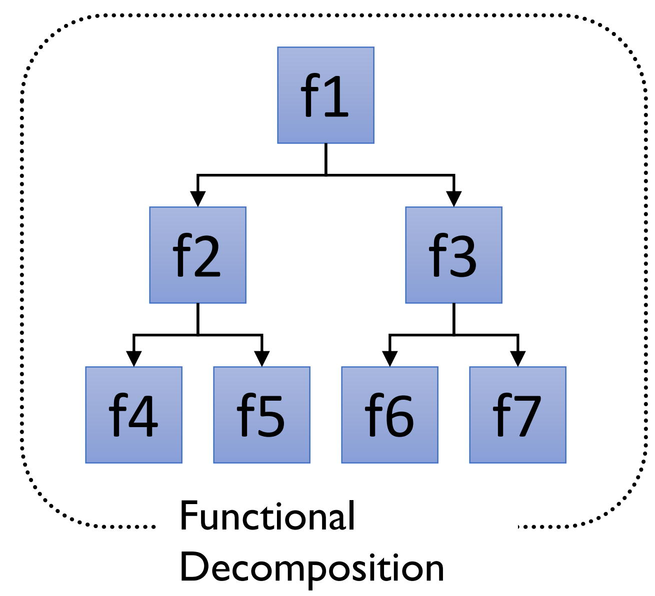
CQRS: Command and Query Responsibility Segregation

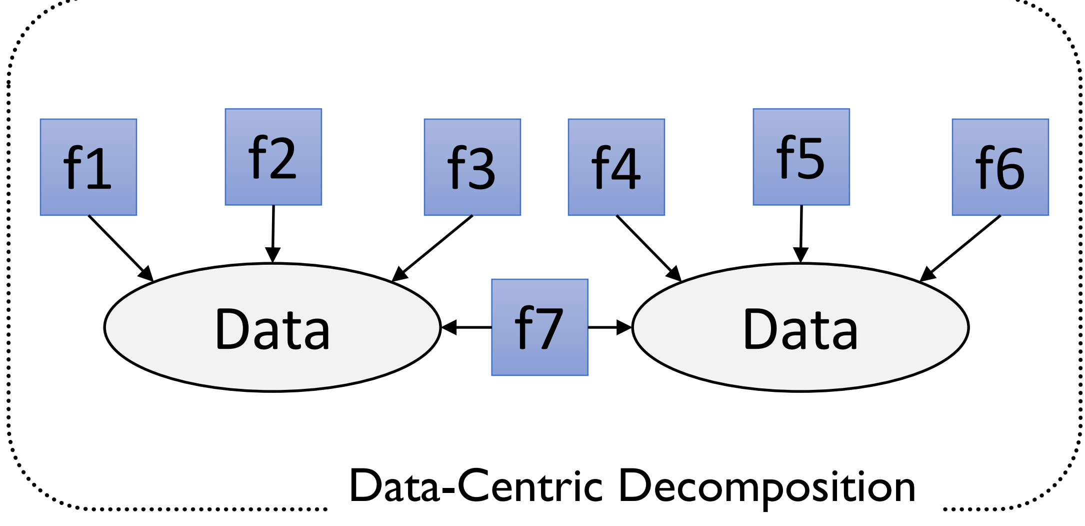
"All methods, whether a command executing an action or a query returning data in response to a call, must carry out both these responsibilities. This is the same as when questions are asked, an answer must be returned." Bertrand Meyer



Problems of Modularization: How should it be divided?

- Functional Decomposition
  - Software is designed by dividing a large function into lowering levels of abstraction.
- Data-Centric Decomposition
  - Software is designed to clarify its subject data and to describe the operation of that data and the links between its different pieces.





Introduction to Object-Oriented Programming using JavaScript

JavaScript for Beginners

JavaScript

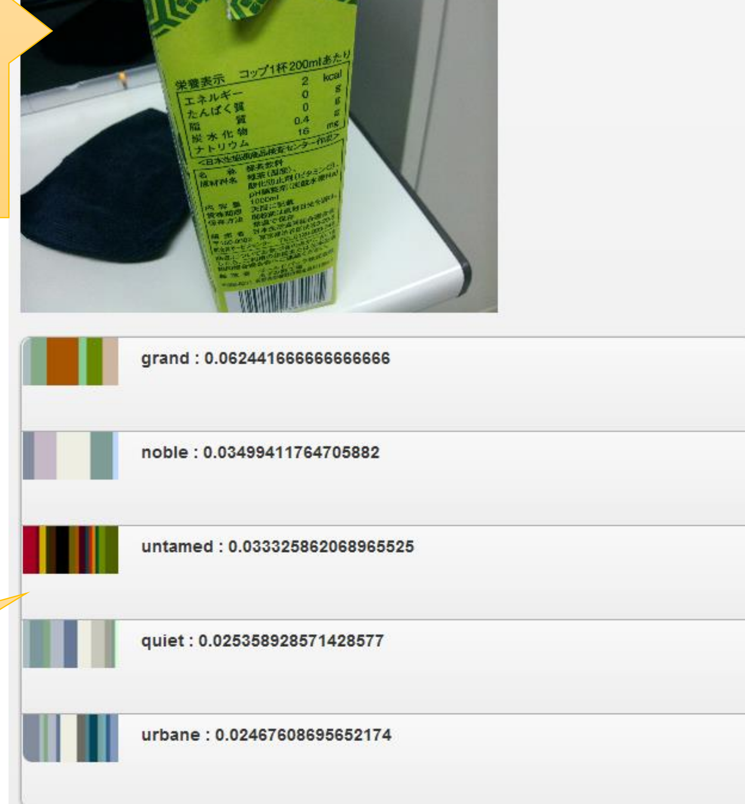
- JavaScript
  - Designed by Brendan Eich (Mozilla Corporation CTO)
  - A programming language designed with influences from the three languages: Java, Scheme, and Self.
  - Contrary to popular understanding, JavaScript can be used for developing programming.
- With the adoption of JIT Compiler, the execution speed of JavaScript on every company's browser has advanced and it is possible to execute it with a an adequately practical level of efficiency.
- A strong client system can be developed by combining HTML5 + CSS3 + JavaScript + jQuery

HTML5 + JavaScript Sample Application

http://goo.gl/ZsrKK

① Click this button and take a picture or choose a local image file.

② The system analyzes color impression of your image

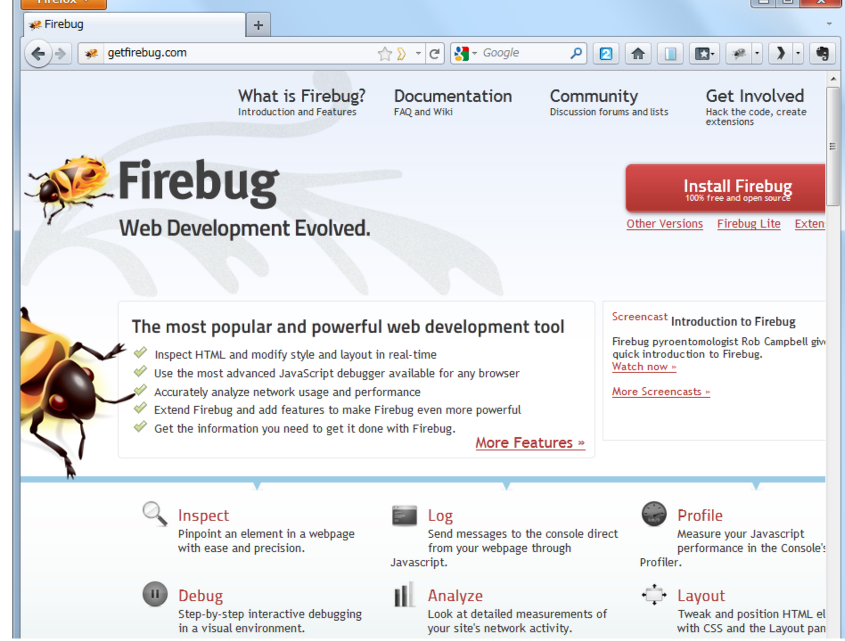


③ The system shows the color-impression metadata for your image.

Exercise Environment

- Firefox (Web browser) + Firebug (JavaScript development environment operational in Firefox)

Step 1: Install the latest version of Firefox
http://mozilla.jp/firefox/



Step 2: Install the latest version of Firebug, which is an add-on for Firefox
http://getfirebug.com/

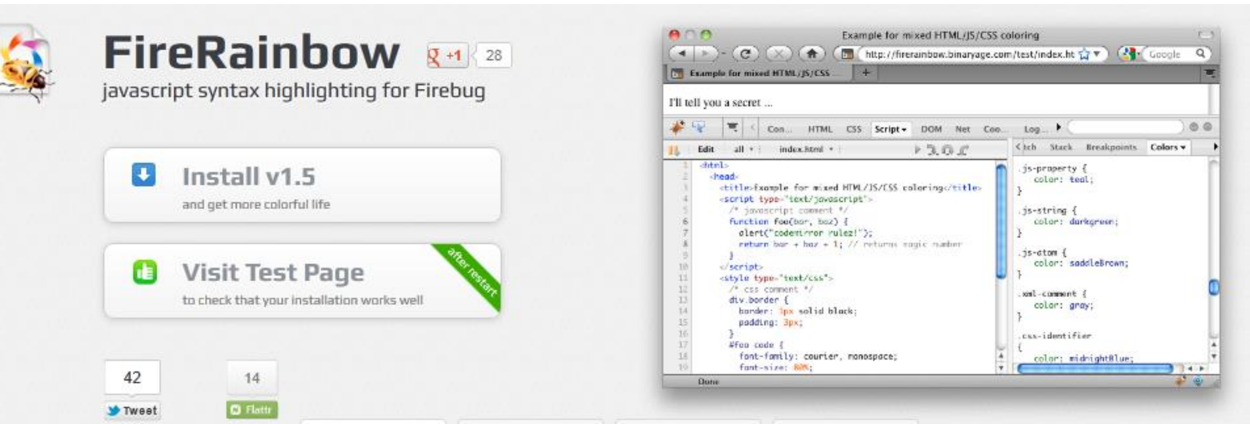
Recommended Add-on

http://www.flailingmonkey.com/acebug/

You are able to highlight elements in the JavaScript program on the JavaScript consol, making them easier to see.

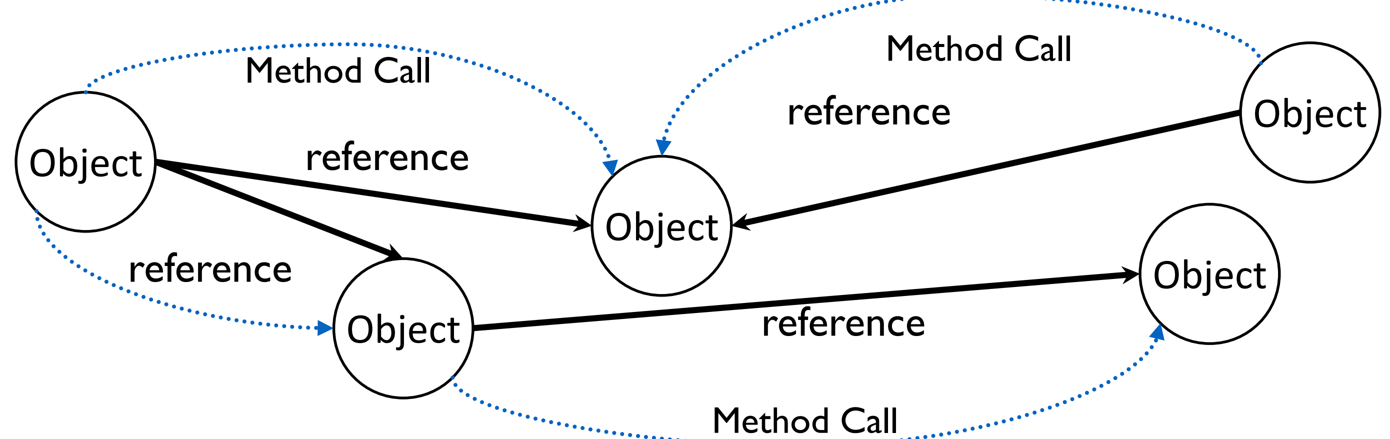
http://firerainbow.binaryage.com/

This lets you highlight elements in Firebug's JavaScript program, making them easier to see.



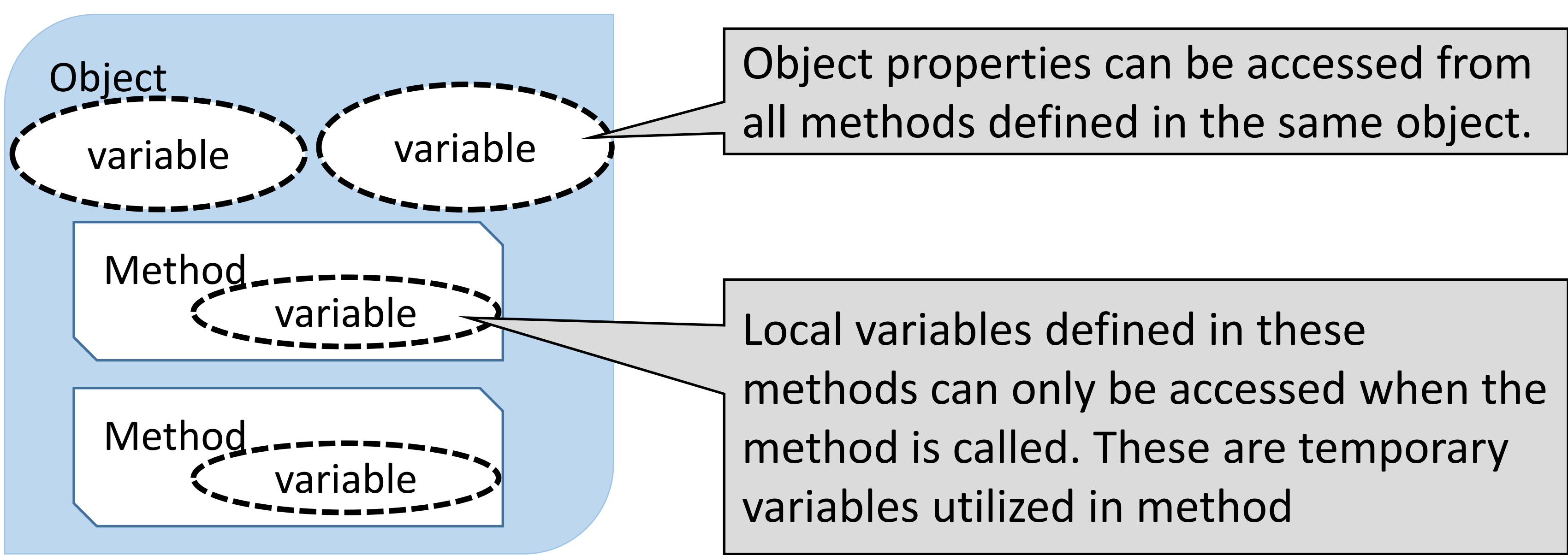
What is Object-Oriented Programming?

- Object-Oriented Programming: decompose and organize a system through the a unit known as an object, and realize it through a message passing between objects.
- Object = data structure + procedure for data operations
- Highly-Condensed & Loosely-Coupled modules
  - High condensation means the state in which mutually related methods are all collected into a single module.
  - Loosely coupling means the state in which dependence between modules is low and the exchange of information between modules is limited.



What is an Object?

- Object is the smallest unit of a program that makes up software.
- A object contains data (attributes, property, variable, property, value, or member)
- A object contains methods (behavior, feature, function, or implementing) for controlling contained data.



“Object” has the Fundamental Three Roles

Information Hiding through Encapsulation	• The internal data of an object is cannot be directly accessed from outside; moreover, that data structure is also not visible from outside.
Aggregation	• The feature in which multiple objects are contained within a single object, and a unified interface for various features of each object is supplied. Through this, general complexity can be reduced.
Inheritance	• The feature that allows for reuse, such as the re-implementation of a variety of features supplied by existing objects to other objects.

JavaScript Basics

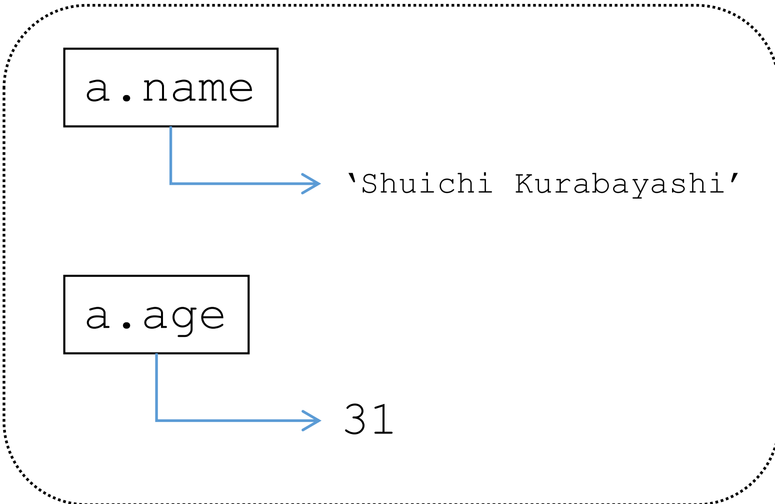
- Data is represented as objects.
- Features are represented as functions

```
var a = {};
```

An empty object containing no attributes

```
var a = {
      name: 'Shuichi Kurabayashi' ,
      age: 31
};
```

An object containing the attributes name and age



Essential 7 Elements in JavaScript

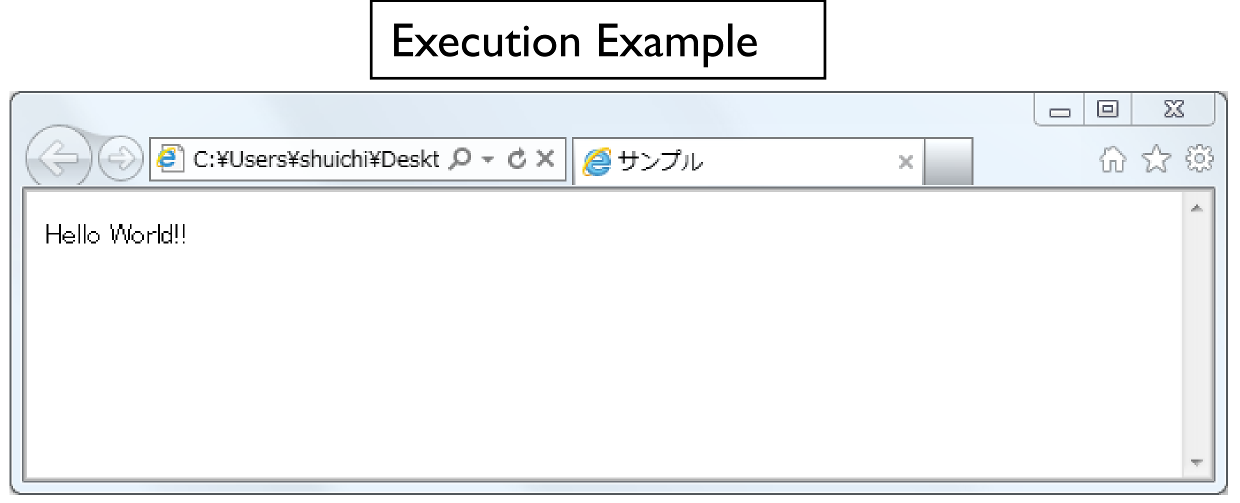
Element	Example
Variable Declaration	• var obj;
Object Creation	• var obj = {name: value};
Array Creation	• var array = [];
Function Definition	• function func(){};
“this” context	• this.attribute
If statement/for statement/ while statement	• if(){}, for(;;){}, while(){}

First JavaScript Program

- Write up the following program in a text editor such as notepad, save it under a name such as sample1.html, and display it in a web browser.

```
<html>
<head>
<title>Sample</title>
</head>
<body>
<script type="text/javascript">
document.write("Hello World!!");
</script>
</body>
</html>
```

Execution Example



JavaScript on a Browser

- JavaScript programs are enclosed in <script type=“text/javascript”></script>.
- The <script> tag can be inserted in an arbitrary location in the HTML.

```
<html>
<head>
<title>Sample</title>
</head>
<body>
<script type="text/javascript">
document.write("Hello World!!");
</script>
</body>
</html>
```

Displaying the Date

```
<html>
<head>
<title>Sample</title>
</head>
<body>
<script type="text/javascript">
document.write(new Date());
</script>
</body>
</html>
```

Displays the current time

C:¥Users¥shuichi¥Deskt
サンプル
Thu May 19 10:50:00 UTC+0900 2011

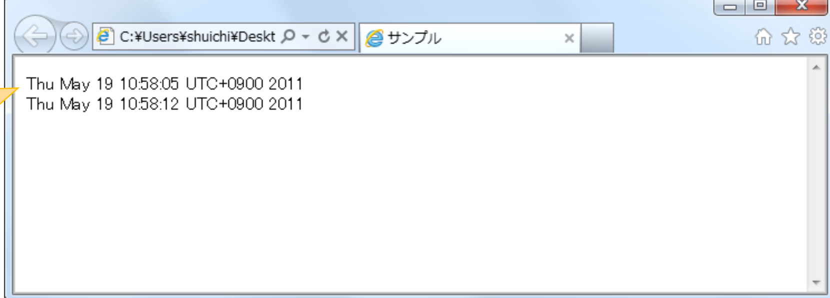
HTML and JavaScript (1)

- The <script> tag can be inserted in any arbitrary location any number of times.

```
<html>
<head>
<title>Sample</title>
</head>
<body>
<script type="text/javascript">
document.write(new Date());
</script>

<script type="text/javascript">
</script>
```

```
var dummy = 0;
var i = 0;
for (i = 0; i < 1000000000; i+=1) {
  dummy += 1;
}
document.write("<br />");
<script type="text/javascript">
document.write(new Date());
</script>
</body>
</html>
```

The current time is displayed twice. The second time is carried out after a “for loop,” which explains the passage of time.



HTML and JavaScript (2)

- You can combine the <script> tag with other HTML tags.

```
<html><head><title>Sample</title></head><body>
<h1>Extreme JavaScript Learning</h1>
<p>Lorem ipsum dolor sit amet, …..</p>
<script type="text/javascript">
document.write("<br />");
document.write(new Date());
</script>
</body></html>
```

A string of characters described as HTML

A string of characters created by JavaScript

C:¥Users¥shuichi¥Deskt

Sample

Extreme JavaScript Learning

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Thu May 19 11:06:39 UTC+0900 2011

HTML and JavaScript (3)

- The <script> tag can be combined as a child element of other HTML tags.

```
<html>
<head>
<title>Sample</title>
</head>
<body>
<h1>Extreme JavaScript Learning: <script
type="text/javascript">document.write(new Date());</script></h1>
<p>Lorem ipsum dolor sit….</p>
</body>
</html>
```

A string of characters described as HTML, and a string of characters created by JavaScript

Extreme JavaScript Learning: Thu May 19 11:11:18 UTC+0900 2011

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Functions and Variables

- JavaScript
  - Program: model as function, manage
  - Data Structure: model as object, manage
- The role of functions
  - Guarantee of reusability: allows for the reusability of a procedure with a completed series.
  - Variable scope management: all variables belong to a function.

Functions and Variables (1)

- Everything can be assigned to a variable.
- Variable Declaration
  - var name;
- Variable Assignment
  - var name = value;

```
var x = 100;
x = 'Hello';
x = 0.9876;
```

Functions and Variables (2)

- Functions are **executable** variables.
- Variable Declaration
  - var name;
- Variable Assignment
  - var name = function(….){….};
- Function Invocation
  - Name();

```
var buttonAction = function() {
  alert("The button was clicked.");
};
```

Functions and Variables (3)

- All elements of JavaScript are variables, and objects are the substance of variables.
- Object types are (1) HTML elements, (2) integer, (3) character string, (4) function, and (5) objects created by users.

```
<html>
<head>
<title>Sample</title>
</head>
<body>
<input type="button" id="button1" value="Click"/>
</body>
<script type="text/javascript">
  var buttonAction = function() {
  alert("The button was clicked.");
  };

  var button1 = document.getElementById("button1");
  button1.onclick = buttonAction;
</script>
</html>
```

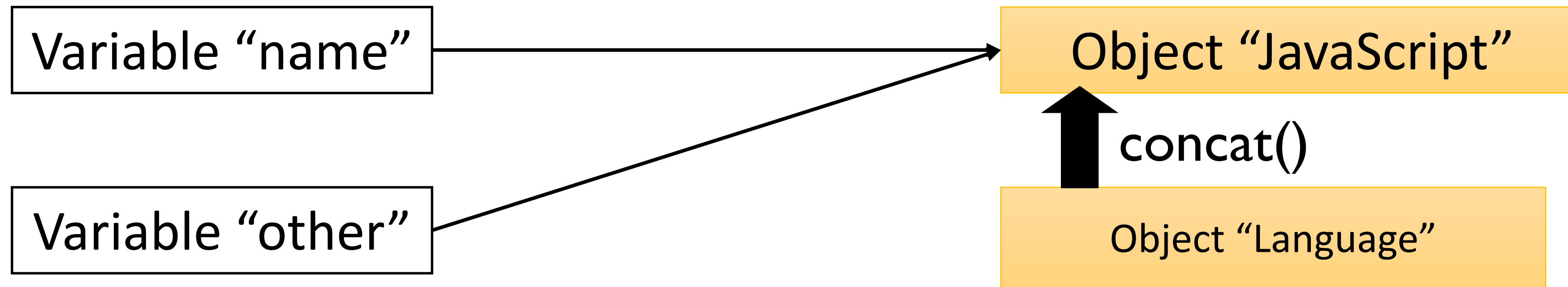
Using the function(){} syntax, a function object is created and assigned to the variable “buttonAction”

Substance of the “buttonAction” variable (function object) is assigned to the Button1 object’s onclick event

What is a Variable?

- A variable is a reference to an object.
  - In JavaScript programming, “variables” are used to operate on objects.
- A variable is really *variable* because it can refer to objects, numbers, functions, strings.

```
var name = “JavaScript”;
var other = name;
other.concat(“Language”);
```



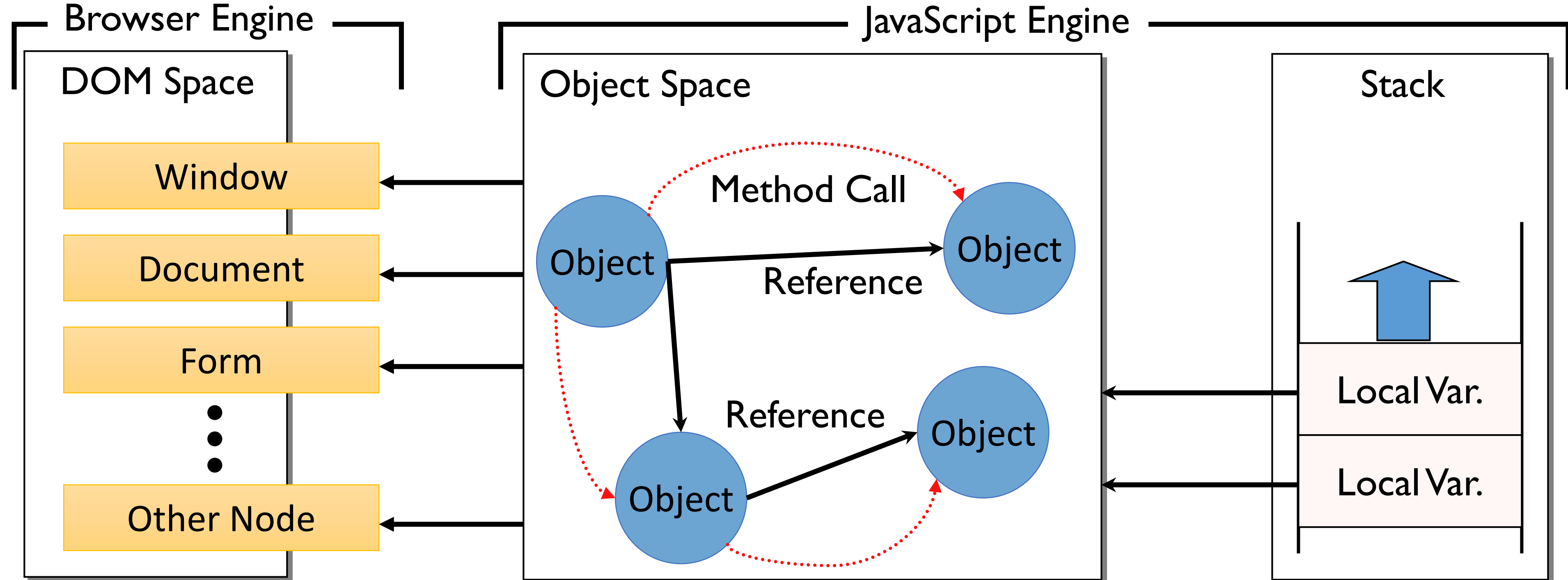
Variable Declaration and Initialization

- Declaration is registration of a name.
  - Global variables are registered in the system.
  - Local variables within the function are registered in the function.
- Variables temporarily place names on program components.
  - Things without names cannot be handled in programming.
  - Ultimately, programming is “the work of placing names on things.”
- Variable Declaration
  - var name;
- Assignment
  - name = value;

```
var sample = {};
```

JavaScript Memory space

- DOM Space: the space where the Document Object Model representing the HTML's layered structure is represented.
- Object Space: the space where all JavaScript objects are located.
- Stack: short-term memory



Exercise 1

- Try to put multiple <script> tags in your HTML and check the result.
- How does your program executed?

```
<html>
<head>
<title>Sample</title>
</head>
<body>
<script type="text/javascript"> var x = "Hello World"; </script>
<h1>
<script type="text/javascript"> document.write(x); </script>
</h1>
<script type="text/javascript"> x = 100; </script>
<h2>
<script type="text/javascript"> document.write(x); </script>
</h2>
</body>
</html>
```

Variable Scope

- Variables are managed in the name space.
- Variables defined in the top of <script> tag are belonging to the global scope.
- Variables defined in a function are belonging to the function.
  - Those variables are called “local” variables.
- Multiple <script> tags are belonging to the same global scope, thus any variables defined in the top of those <script> tags shares the same scope.

Those “x” are the same variable.

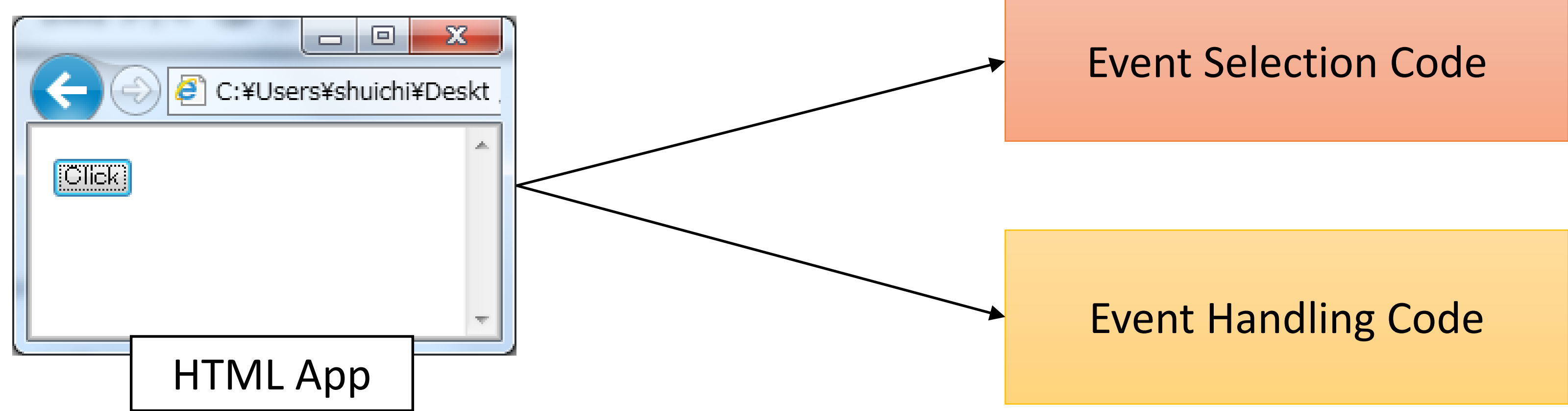
```
<script type="text/javascript"> var x = "Hello World"; </script>
<script type="text/javascript"> document.write(x); </script>
<script type="text/javascript"> x = 100; </script>
<script type="text/javascript"> document.write(x); </script>
```

Event Listeners & Basic GUI Programming in JavaScript

Introduction to Event-Driven Programming in HTML

What is Event?

- “Event” is a data object representing something occurrence - e.g. user actions (mouse clicks, key presses) or messages from other programs (Web Socket and Web Worker).
- Web application is modelled as Event-driven architecture which consists of the following two sections:
  - Event Selection
  - Event Handling

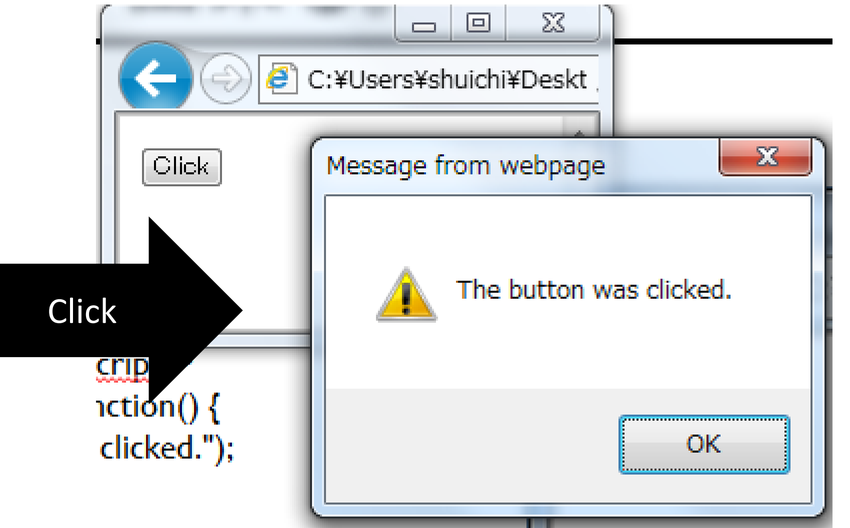


Event Selection

- Assign a function object to the special attributes of HTML elements, in order to react to the input of HTML elements (Form element, P tag, H tag, etc.).
- document.getElementById() method returns the HTML element specified by ID.

```
<html><body>
<input type="button" id="button1" value="Click"/>
<script type="text/javascript">
  var buttonAction = function() {
    alert("The button was clicked.");
  };

  var button = document.getElementById("button1");
  button.addEventListener("click“, buttonAction);
</script>
</body></html>
```

Button reacting to click.



Event Listener Definition using Function Literal

- Function Literals

```
var listener = function (event) {
    alert(“message for a user");
}
```

- Function (event) indicates that the function accepts one argument, “event” representing something occurrence in DOM.
- Inside the { }, the process of the function is described.
- Functions are also objects and can be assigned to variables.

Event Listener Example

```
<html>
<body>
<button id="button1">Button1</button>
<button id="button2">Button2</button>
<script type="text/javascript">

document.getElementById("button1").addEventListener("click",
    function(event){
        alert("Button 1");
    }
);

document.getElementById("button2").addEventListener("click",
    function(event){
        alert("Button 2");
    }
);
</script>
</body>
</html>
```

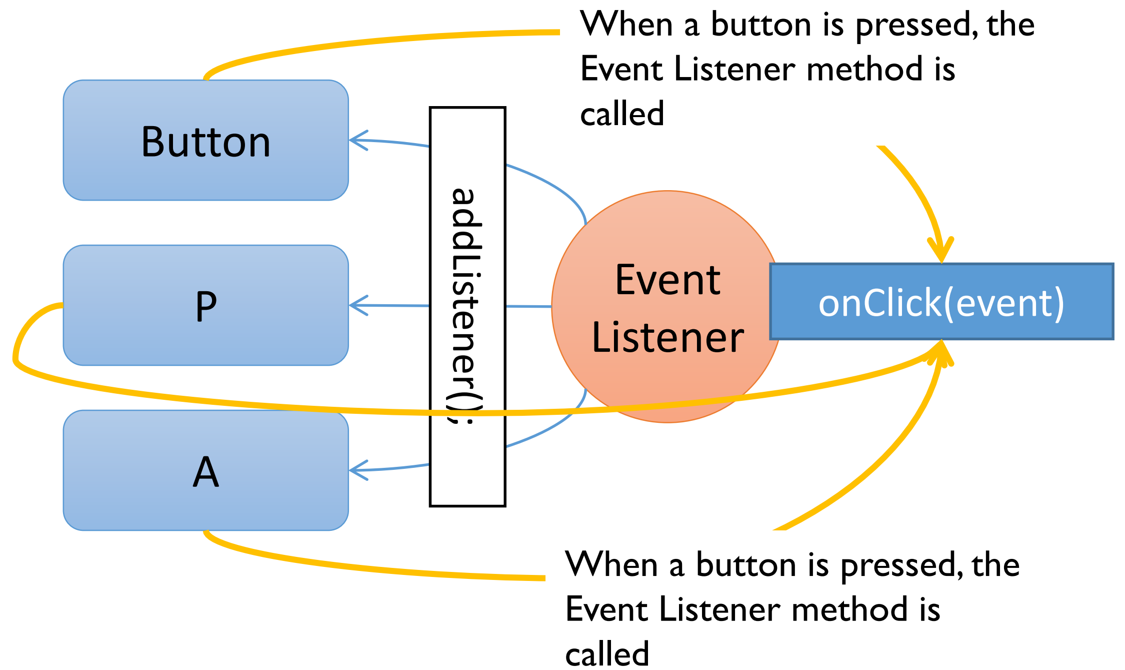
Button1 Button2

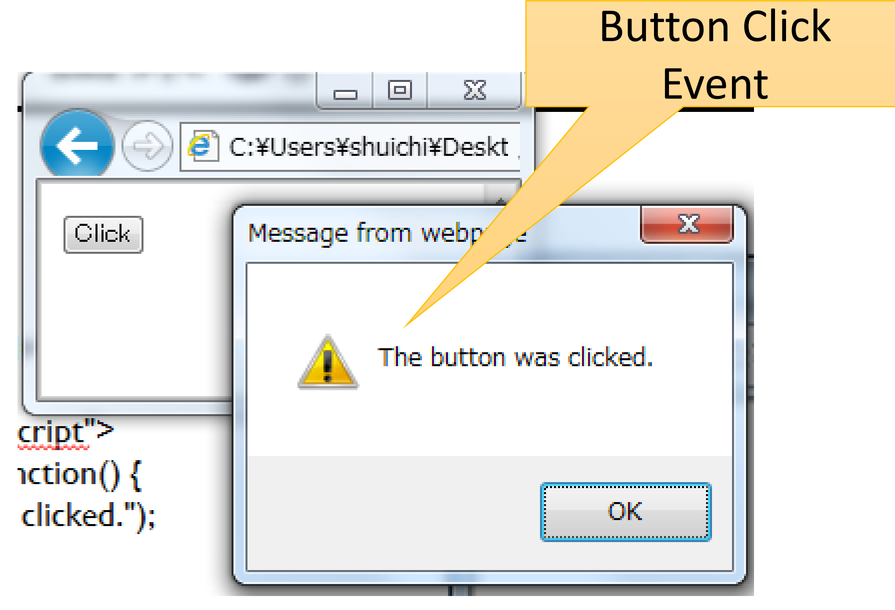
Clicking on Button1 causes the function to run.

Clicking on Button2 causes the function to run.

Event Listener Mechanism

- The Event Listener is called when an action occurs in a button or text-field GUI component, such as a button click or input of the return key.





DOM for Event-Driven Application

- This example HTML shows four DOM elements as follows: INPUT button, SPAN, P, and INPUT text.
- By clicking them, text and color will be changed by event listeners.

Button1 This is a SPAN tag.

This is a P tag.

This is an input[type=text] tag.

```
<html>
    <body>
        <input type="button" id="button1" value="Button1"/>
        <span id="span1">This is a SPAN tag.</span>
        <p id="p1"> This is a P tag.</p>
        <input id="text1" type="text" size="30" value="This is an input[type=text] tag.">
        <script type="text/javascript" src="ex2.js"></script>
    </body>
</html>
```

Event Listener Definition

- clickAction
  - This function will react to the clicking event. This function changes the clicked DOM element text to “clicked!” and also changes its color to “red”.
- mouseOverAction
  - This function will be invoked when the mouse enters into a DOM element. This function changes the target DOM element’s background color to “blue”.
- mouseOutAction
  - This function will be invoked when the mouse gets out from a DOM element. This function changes the target DOM element’s background color to “white”.

```
var clickAction = function(evt) {
    var domElement = evt.target;
    if (domElement.value) {
        domElement.value = "Clicked!";
    } else {
        domElement.innerText = "Clicked!";
    }
    domElement.style.color = "red";
};

var mouseOverAction = function(evt) {
    var domElement = evt.target;
    domElement.style.backgroundColor = "blue";
};

var mouseOutAction = function(evt) {
    var domElement = evt.target;
    domElement.style.backgroundColor = "white";
};
```

Event Selection: “addEventListener” method

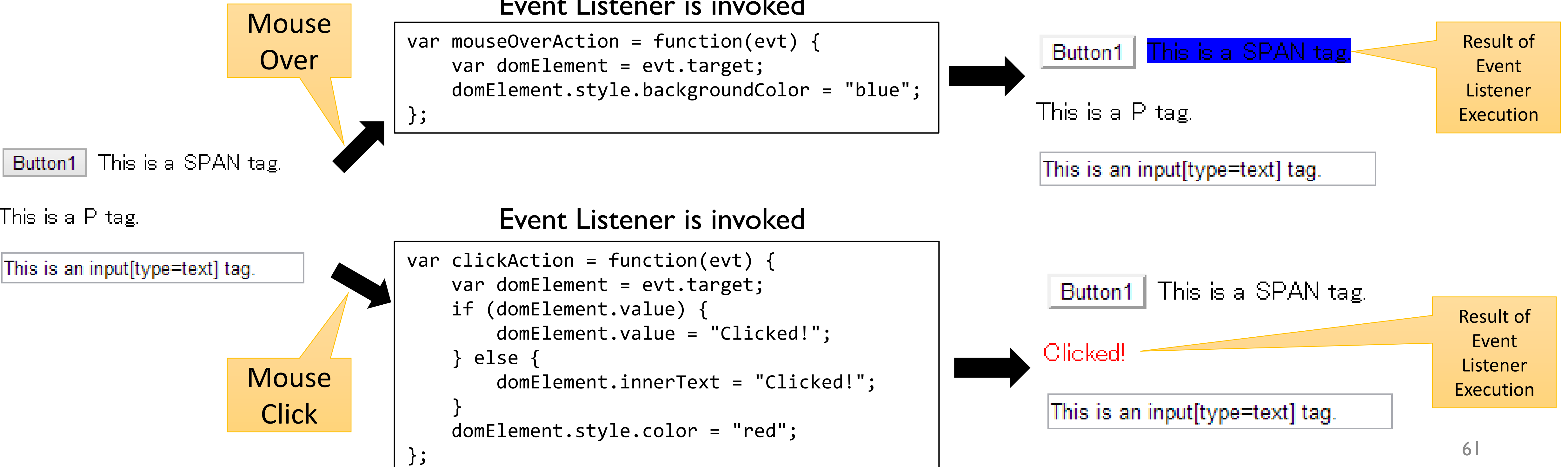
- Every DOM element provide a method, called “addEventListener”, to associate an event and an event listener.
- `dom.addEventListener("click", clickAction)` associates “click” event and the event listener function clickAction.

```
var ids = ["button1", "p1", "span1", "text1"];
var i;
for (i = 0; i < ids.length; i++) {
    var dom = document.getElementById(ids[i]);
    dom.addEventListener("click", clickAction);
    dom.addEventListener("mouseover", mouseOverAction);
    dom.addEventListener("mouseout", mouseOutAction);
}
```

ids is an array that stores each DOM element’s ID.

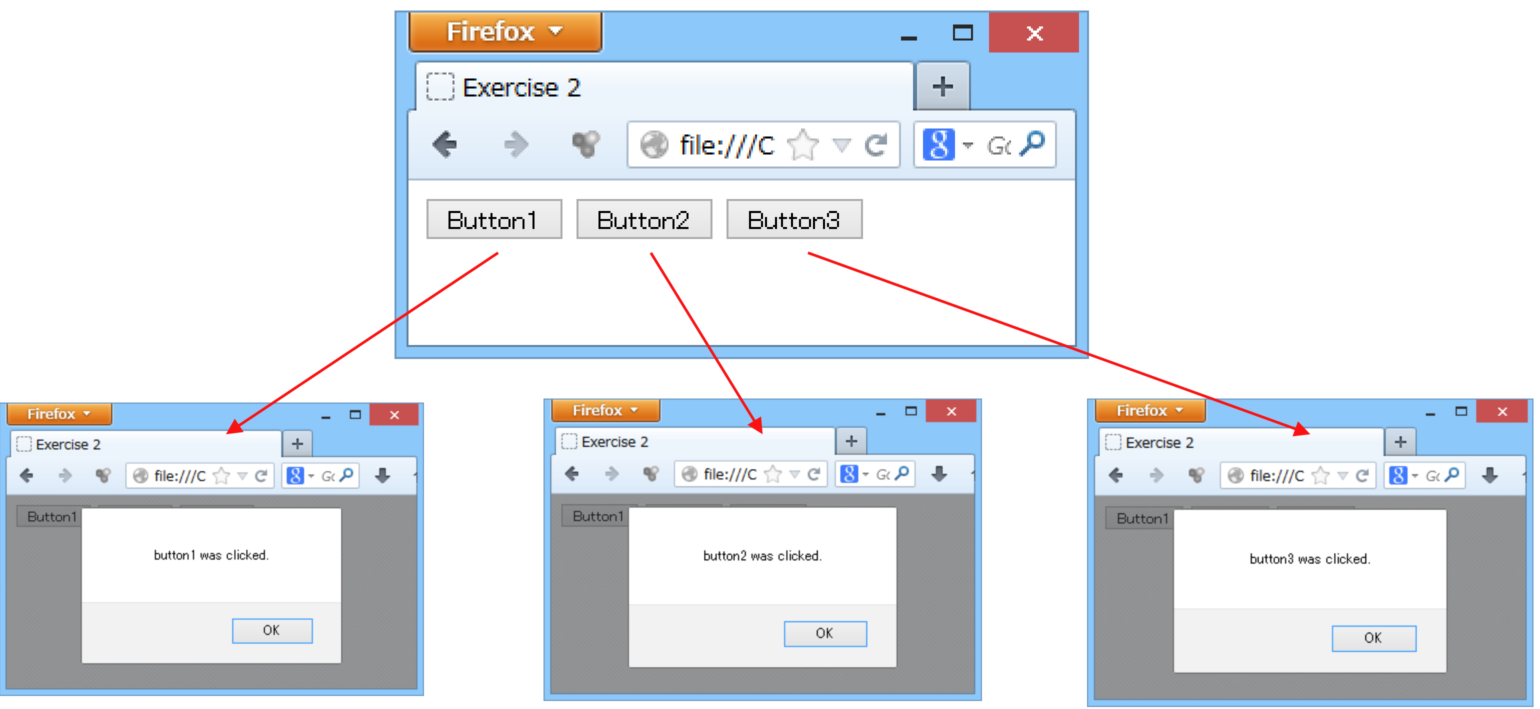
Event Lister and Dynamic HTML (DHTML)

- A web browser invokes event listeners associated with DOM elements when a specified event, such as click and mouse over, is detected.
- When an event listener modifies DOM elements or CSS, a browser updates the screen. This dynamic update mechanism is called “Dynamic HTML”.



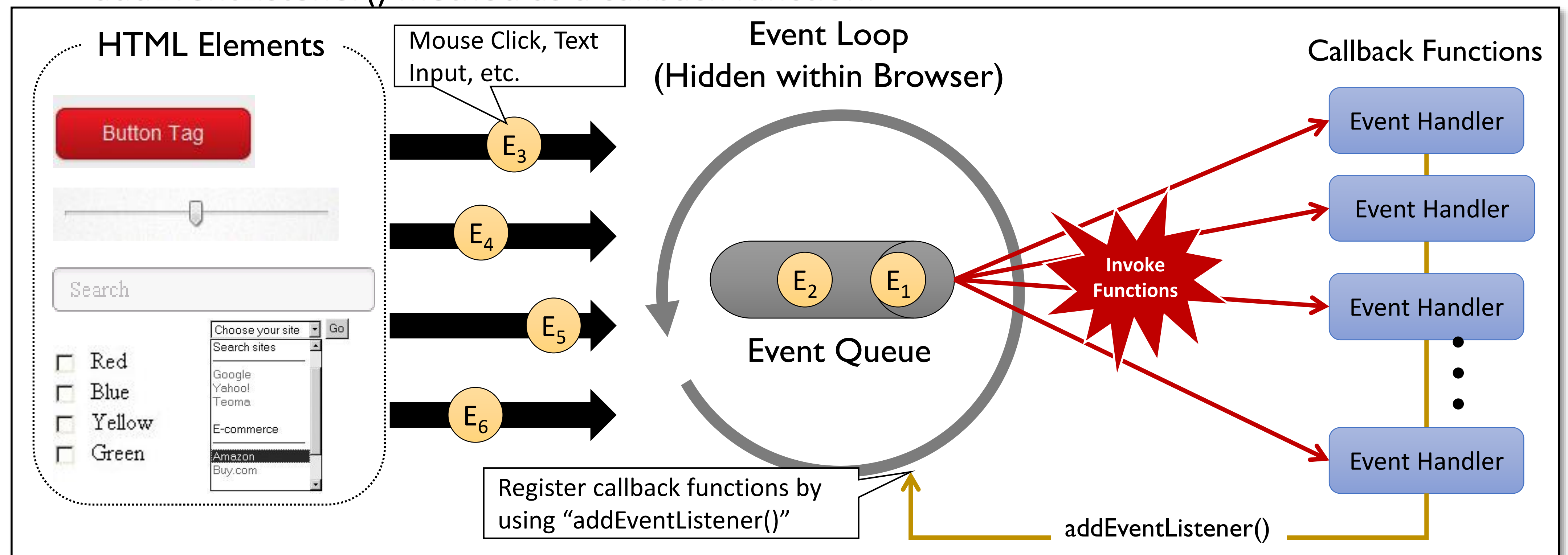
Exercise 2

- Create a program that provides three buttons(Button1, Button2, and Button3). Each button is assigned a corresponding event listener for showing dialog when the button is clicked.



Event Loop

- The web browser provides an “Event Loop” as the central mechanism to execute GUI-based interaction with users.
- In an event loop,
    - “events” (mouse clicks, etc.) sent from DOM are stored in the event queue
    - and carry out a method for the processing of each successive event.
- This event processing function is called an Event Handler and is registered through the addEventListener() method as a callback function.



Objects, Prototypes, and Methods

Fundamental Object-Orientation Mechanisms in JavaScript

JavaScript Objects

- Objects are a set of values aligned through a key (character string).
- A value can be an integer, a character string, Boolean, null, undefined, and an object.

An example of a person object that has the attributes firstName and lastName

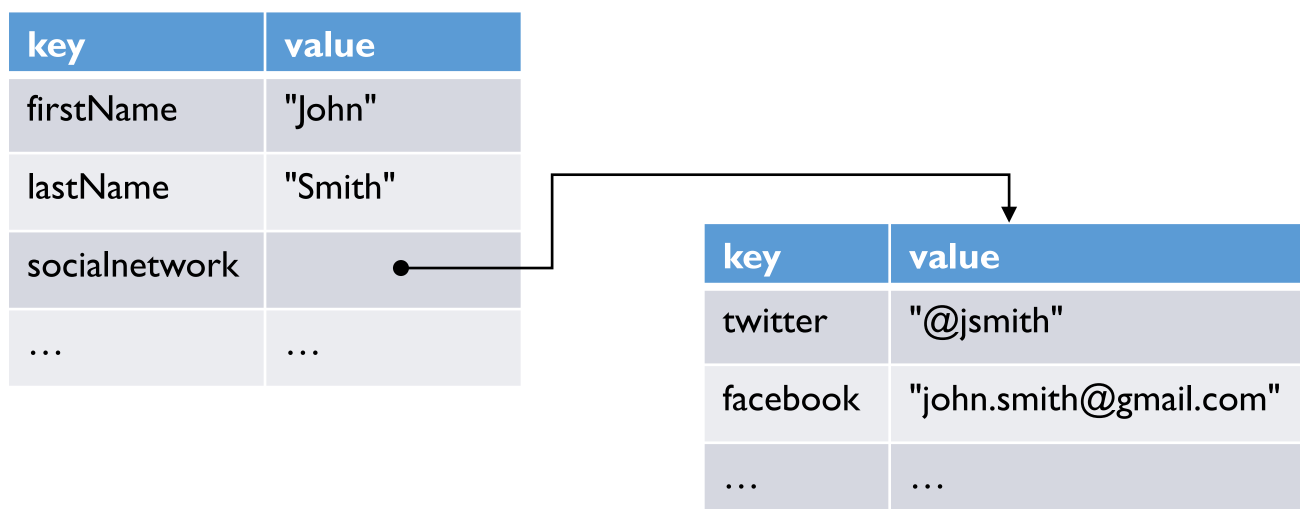
```
var person = {
    firstName: "John",
    lastName: "Smith"
};
```

An example of a person object that has the attributes firstName and lastName

```
var person = {
    firstName: "John",
    lastName: "Smith",
    socialnetwork: {
        twitter: "@jsmith",
        facebook: "john.smith@gmail.com"
    }
};
```

JavaScript Object Structure

- An objects is a set of values aligned through a key (character string), adding, deleting, changing attributes, and linking to other objects can be freely set.
- The technique of representing data structures as object networks is a basic feature of object-oriented programming.



Accessing Object Attribute

Value setting

- dot notation

```
person.email = "john.smith@gmail.com";
```

- bracket notation

```
person["email"] =
"john.smith@gmail.com";
```

Value reference

- dot notation

```
alert(person.email);
```

- bracket notation

```
alert(person["email"]);
```

Accessing Object Attribute: Combination of dot notation and bracket notation

- Dot notation and bracket notation can be combined.
- `person.firstName` and `person["firstName"]` indicate the same attribute

```
var person = {
    firstName: "John",
    lastName: "Smith",
    socialnetwork: {
        twitter: "@jsmith"
    }};
```

```
console.log(person.firstName);                    → John
console.log(person["lastName"]);                  → Smith
console.log(person.socialnetwork.twitter);        → @jsmith
console.log(person["socialnetwork"].twitter);     → @jsmith
console.log(person.socialnetwork["twitter"]);     → @jsmith
console.log(person["socialnetwork"]["twitter"]);  → @jsmith
```

Those four expression indicate the same attribute

Accessing Object Attribute: Combination of dot notation and bracket notation

- A parameter for bracket notation can be a variable.
- In the following example, variable target stores a string "firstName". So person[target] indicate the firstName attribute of person.

```
var person = {
    firstName: "John",
    lastName: "Smith"
};
```

person[target] indicate the firstName attribute of person

```
var target = "firstName"
console.log(person[target]);
```

→ John

How to create an object

- {} Object literal
  - {} generates an empty object.
- Object.create() method
  - Object.create() method generates a new object as a prototype of a designated object.
- “new” operator
  - “new” operator launches a designated function as a constructor and generates an object.

```
var person = {
   firstName: "John",
   lastName: "Smith"
};
```

Attribute Definition

- Object attribute can be defined by using the following two method.

Object Literal “{ }”

```
var jsmithsns = {
    twitter: "@jsmith",
    facebook: "js@gmail.com"
};

var person = {
    firstName: "John",
    lastName: "Smith",
    socialnetwork: jsmithsns
};
```

Combining dot “.” notation and assignment operator “=“

```
var person = {};
person.firstName = "John";
person.lastName = "Smith";
person.socialnetwork = {};
person.socialnetwork.twitter = "@jsmith";
person.socialnetwork.facebook= "js@gmail.com";
```

This approach is effective to define recursive relationship because it is easy to refer to self object.

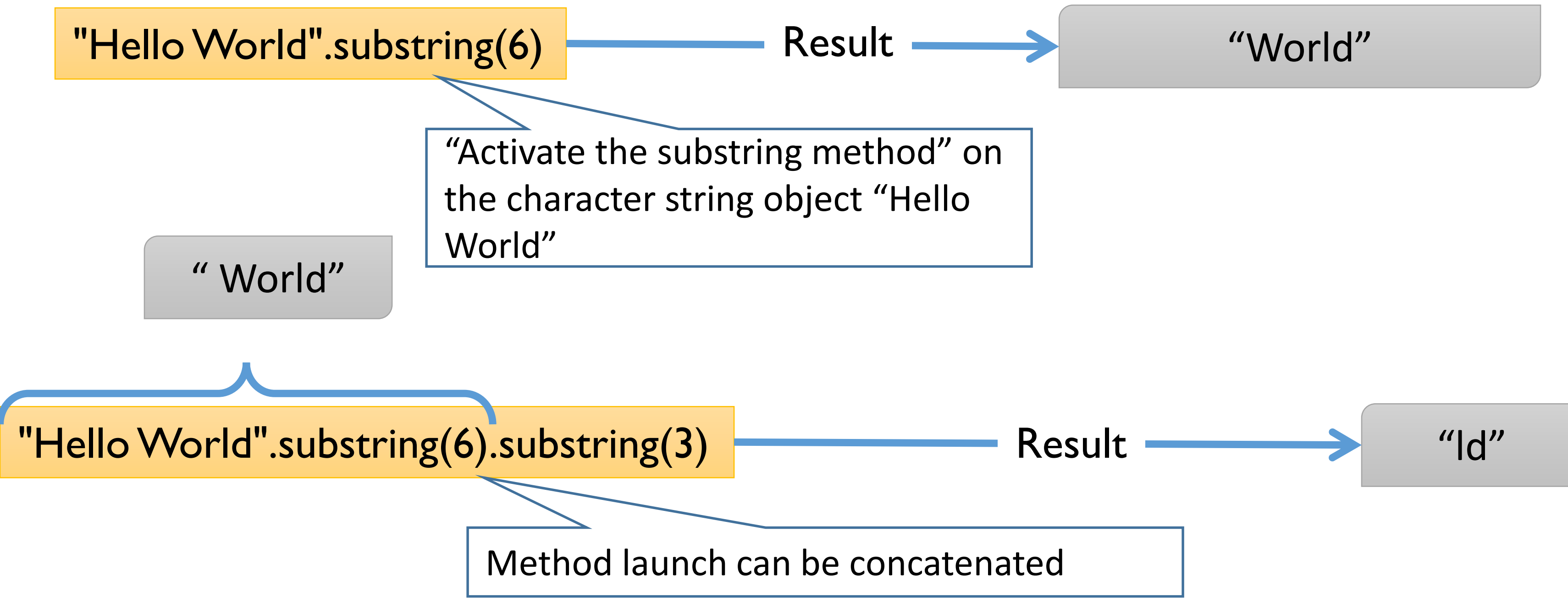
Exercise 4

- By modifying the right-side “person” object definition, try to create 3 persons objects as follows: John Smith, Joanna Doe, and Hans Schmidt.
- John Smith has a “friend” feature that is a reference to Joanna Doe.
- Joanna Doe has a “friend” feature that is a reference to Hans Schmidt.
- Hans Schmidt has a “friend” feature that is a reference to John Smith.
- Dot and assignment notation is required to define the above recursive relationship.

```
var jsmithsns = {
    twitter: "@jsmith",
    facebook: "js@gmail.com"
};

var person = {
    firstName: "John",
    lastName: "Smith",
    socialnetwork: jsmithsns
};
```

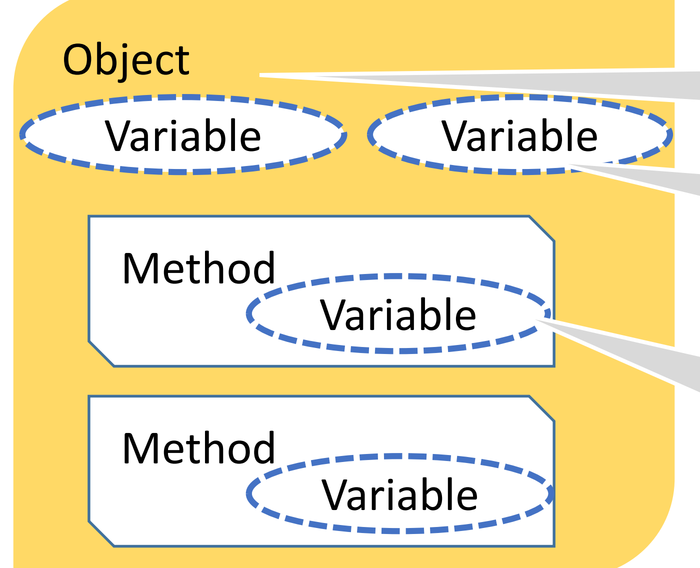
Message Passing

- By placing the dot operator on an object and entering the method name and parameters, you can invoke the appropriate object method.



Terms You Should Remember for Object Definition

Object	• All programs use objects as components.
Method	• Methods describe the programs movements (behavior).
Variable	• Objects and methods can both define a variable. Variables are a slot for keeping references to objects and data.



Objects are generated by passing prototypes to a literal notation or Object.create() method.

Variables defined in objects can be commonly accessed by all methods. This method is termed as an "object attribute."

Variables defined in a method can only be accessed when the method is called. These are temporary variables used for calculation in the method and are localized; therefore, they are termed as local variables.

Array Object

Managing multiple objects in numerical order.

Array Object

- An array object (array) is an object that has attributes placed in order according to integers.

- Array generation
  - Literals noted through empty brackets. (this is recommended for this lecture)

```
var list = [ ];
```

  - Generating an array object.

```
var list = new Array();
```

Array Object Operation concat(), join(), length, pop(), push(), reverse(), shift(), slice(), sort(), splice(), unshift()

Assigning, adding, and deleting elements

- designating and generating elements

```
var list = ["Hello", "World"];
```

- Adding elements

```
var list = [];
list.push("Hello");
list.push("World");
```

- Deleting Elements

```
var list = ["Hello", "World"];
list.pop()  //  deletes the last element
list.shift()  //Deletes the first element
```

Referencing elements

- Reference by index

```
alert( list[0] );
```

- for statement

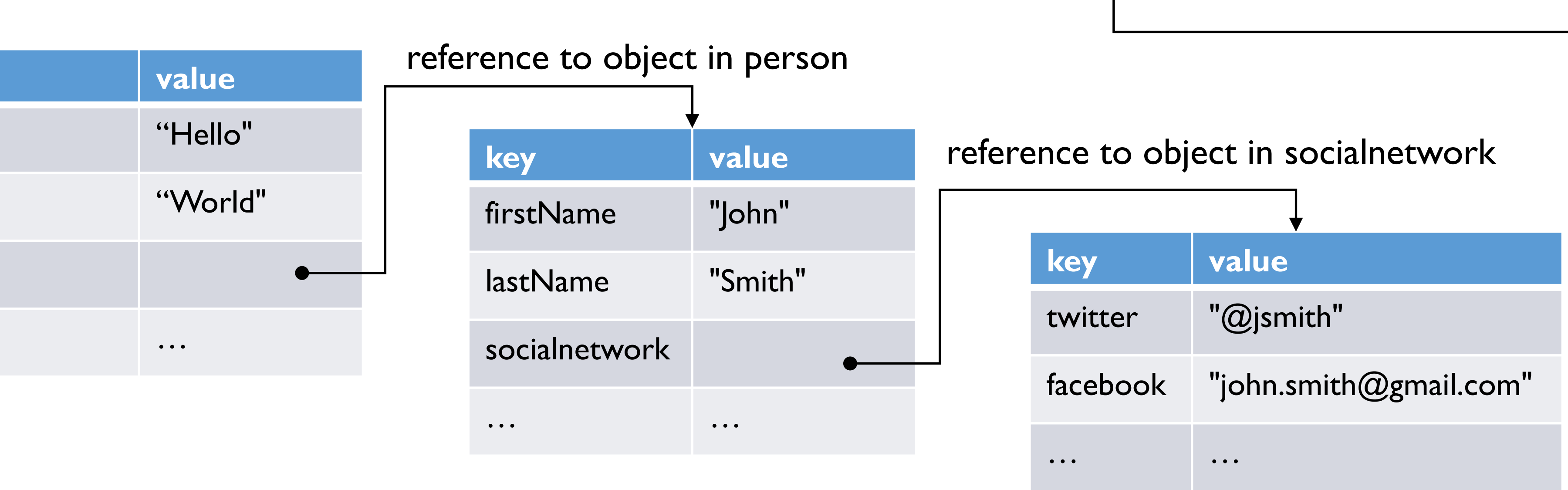
```
var i;
for (i = 0; i < list.length; i
+= 1) {
    console.log(list[i]);
}
```

  - list.length shows the length of the entire array

Array Object Structuring

- An array object is a variety of object, and is a set of values ordered according to integers making up the key.
- Other objects can be assigned to array objects.

```
var list = [];
list.push("Hello");
list.push("World");
list.push(person);
```



Exercise 5

- By reusing the right-side “person” object definition, try to create an array object storing at least 10 persons objects.

```
var person = {
    firstName: "John",
    lastName: "Smith",
};
```

Array Practice 1: Creation

- forEach Method (Array)

- Syntax

- Callback Function Syntax

Array Practice 1: Iteration by length property

```
var ary = ['A', 'B', 'C', 'D', 'E'];
var i;
for (i = 0; i < ary.length; i++) {
    console.log(ary1[i]);
}
```

- An array object has “length” property that indicates the size of itself.
- In the above example, array1.length is “5”.
- ary[0] is ‘A’, ary[1] is ‘B’, ary[2] is ‘C’, ary[3] is ‘D’, ary[4] is ‘E’.
- Thus, *i* will be incremented as follows: 0, 1, 2, 3, 4.
- When *i* becomes 5, *i* < *ary.length* fails and *for* iteration stops.

Array Practice 2: Iteration by “forEach” method

```
var array1 = ['A', 'B', 'C', 'D', 'E'];
array1.forEach(function (item) {
    console.log(item);
});
```

- forEach Method (Array)
  - The forEach method calls the callback function one time for each element present in the array, in ascending index order.
- Syntax

```
array1.forEach(callback)
```

- Callback Function Syntax

```
function callback(value, index, array1)
```

Advanced Example

```
['A', 'B', 'C', 'D', 'E'].forEach(function (item) {
    console.log(item);
});
```

- [] operator generates an array object.
- Thus, we can invoke forEach method for the generated object immediately.

Prototypes

Generating objects with similar functions and attributes of exiting objects

Class-Based and Prototype-Based

- Class-Based Object-Oriented Language
  - A way of thinking that considers all objects belongs to a blue-print called a “class,” and that objects are generated by materializing this blue-print.
  - Software designers first design classes, and next create a program for generating instances of those classes
- Prototype-Based Object-Oriented Language
  - A way of thinking that considers a specific object to be a model (prototype) for other objects and effectively generates objects with similar functions and attributes.
  - Software designers create a program while operating pre-existing objects.

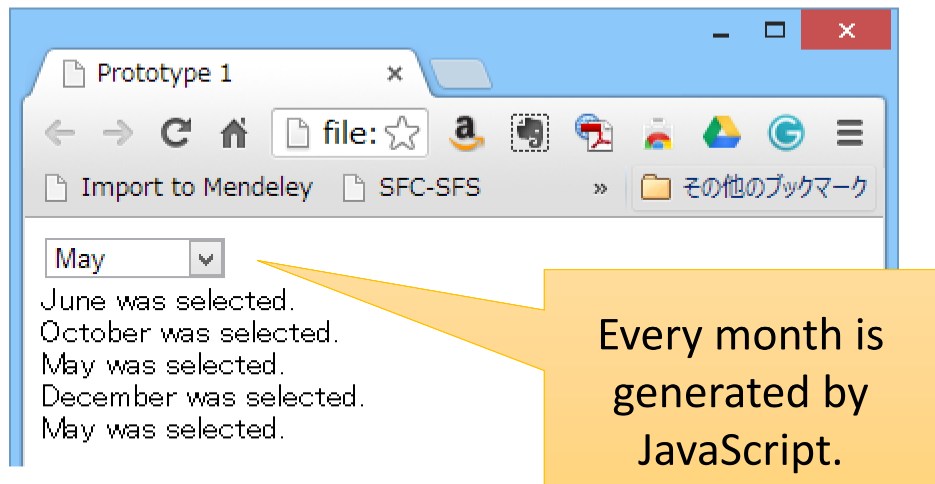
Prototype Object Definition

- MonthSelector is a prototype object. It has one attribute “months” and one method “init”.
- “months” is an array that stores names of twelve months.
- “init” is a function that renders “months” array as a option tag and assigns event listener for the select tag.

```
var MonthSelector = {
   months: ["January", "February", "March", "April", "May", "June",
      "July", "August", "September", "October", "November", "December"],
   init: function (target, log) {
      var select = document.createElement("select"),
         log = document.getElementById(log);

      document.getElementById(target).appendChild(select);
      this.months.forEach(function (m) {
         var option = document.createElement("option");
         option.innerText = m;
         option.value = m;
         select.add(option);
      });

      select.addEventListener("change", function (evt) {
         log.innerHTML += evt.target.selectedOptions[0].value + " was selected.<br>";
      });
   }
};

MonthSelector.init("target", "log");
```

Practice on Arrays

- This program shows a selector for choosing a month. Every month is generated by JavaScript.
- When you choose a month, the selected month are shown in the below DIV area.




```
<html>
<body>
<select id="items">
</select>
<div id="log">
</div>

<script type="text/javascript">
    var months = ["January", "February",
        "March", "April", "May", "June",
        "July", "August", "September",
        "October", "November", "December"];

    var select = document.getElementById("items");
    for (var i = 0; i < months.length; i++) {
        var option = document.createElement("option");
        option.innerText = months[i];
        option.value = months[i];
        select.add(option);
    }

    select.addEventListener("change", function (evt) {
        var selected = evt.target.selectedOptions[0];
        var log = document.getElementById("log");
        log.innerHTML += selected.value + " was selected.<br>";
    });
</script>
</body>
</html>
```

Reusing Prototype Object

- MonthSelectorJP is an extended version of MonthSelector. It re-defines “months” attribute in Japanese. “init” method are reused from MonthSelector.

Japanese version of Month selector is generated by MonthSelectorJP

August 8月
5月 was selected.
August was selected.
8月 was selected.

Load prototype2.js and prototype3.js

```
<body>
<div id="target"></div>
<div id="log"></div>
<script type="text/javascript" src="prototype2.js"></script>
<script type="text/javascript" src="prototype3.js"></script>
</body>
```

```
var MonthSelectorJP = Object.create(MonthSelector, {
  months: {
    value: ["1月", "2月", "3月", "4月", "5月", "6月",
      "7月", "8月", "9月", "10月", "11月", "12月"]
  }
});

MonthSelectorJP.init("target", "log");
```

prototype3.js defines only difference of MonthSelector and MonthSelectorJP

Exercise 6

- Let’s create a French version of MonthSelector.

Prototype-Oriented, Object-Oriented

- Prototype-oriented, object-oriented is a method for making a certain object the prototype of another object to efficiently generate objects with similar features and attributes.

Prototype

```
var Person = {
    firstName: "",
    lastName: "",
    email: "",
};

var list = [];

var p = Object.create(Person);
p.firstName = "Shuichi";
p.lastName = "Kurabayashi";
p.email =
"kurabaya@sfc.keio.ac.jp";

list.push(p);
```

```
p = Object.create(Person);
p.firstName = "John";
p.lastName = "Smith";
p.email = "js@sfc.keio.ac.jp";

list.push(p);

p = Object.create(Person);
p.firstName = "Foo";
p.lastName = "Baa";
p.email = "fb@sfc.keio.ac.jp";

list.push(p);

var i;
for (i = 0; i < list.length; i+= 1) {
    console.log(list[i]);
}
```

Contents displayed on console

```
Object { firstName="Shuichi",
lastName="Kurabayashi",
email="kurabaya@sfc.keio.ac.jp"}

Object { firstName="John",
lastName="Smith",
email="js@sfc.keio.ac.jp"}

Object { firstName="Foo",
lastName="Baa",
email="fb@sfc.keio.ac.jp"}
```

Defining Person

- The init() method initializes an objects attributes.
- The equals() method judges whether objects are same.
- comparedTo() determines the size relationship of objects.

```
var Person = {
    firstName: "",
    lastName: "",
    email: "",
    init: function(first, last, email) {
        this.firstName = first;
        this.lastName = last;
        this.email = email;
    },
    equals: function(obj) {
        if (obj === null) {
            return false;
        }
        if (this.firstName !== obj.firstName) {
            return false;
        }
        return true;
    },
    compareTo: function(obj) {
        return this.firstName > obj.firstName ? 1 : -1;
    }
}
```

Object creation method

```
var a = Object.create(Person);
a.init("Shuichi", "Kurabayashi",
"kurabaya@sfc.keio.ac.jp");
```

Using Person

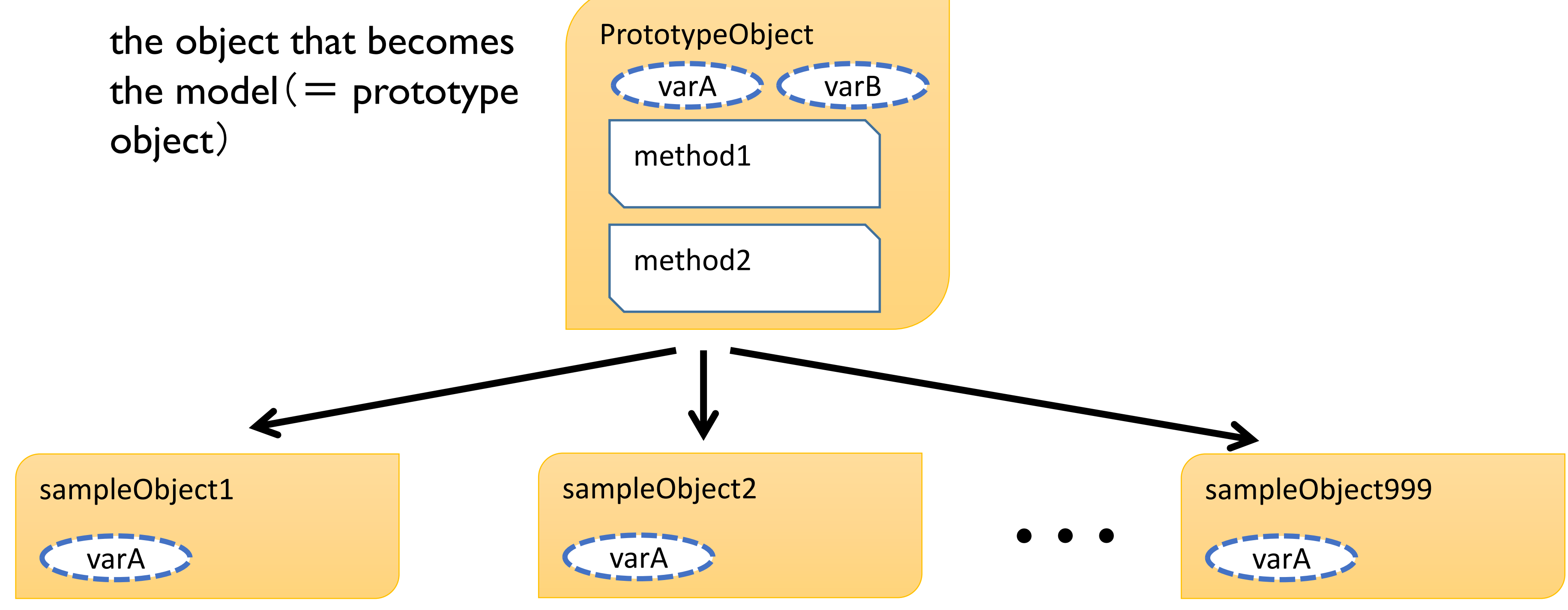
```
var list=[], p, i;

p = Object.create(Person);
p.init("Shuichi", "Kurabayashi", "kurabaya@sfc.keio.ac.jp");
list.push(p);

p = Object.create(Person);
p.init("John", "Smith", "js@sfc.keio.ac.jp");
list.push(p);

p = Object.create(Person);
p.init("Foo", "Baa", "fb@sfc.keio.ac.jp");
list.push(p);

for (i = 0; i < list.length; i+= 1) {
    console.log(list[i]);
}
```

Output Results

```
Object { firstName="Shuichi", lastName="Kurabayashi",
email="kurabaya@sfc.keio.ac.jp"}
Object { firstName="John", lastName="Smith", email="js@sfc.keio.ac.jp"}
Object { firstName="Foo", lastName="Baa", email="fb@sfc.keio.ac.jp"}
```

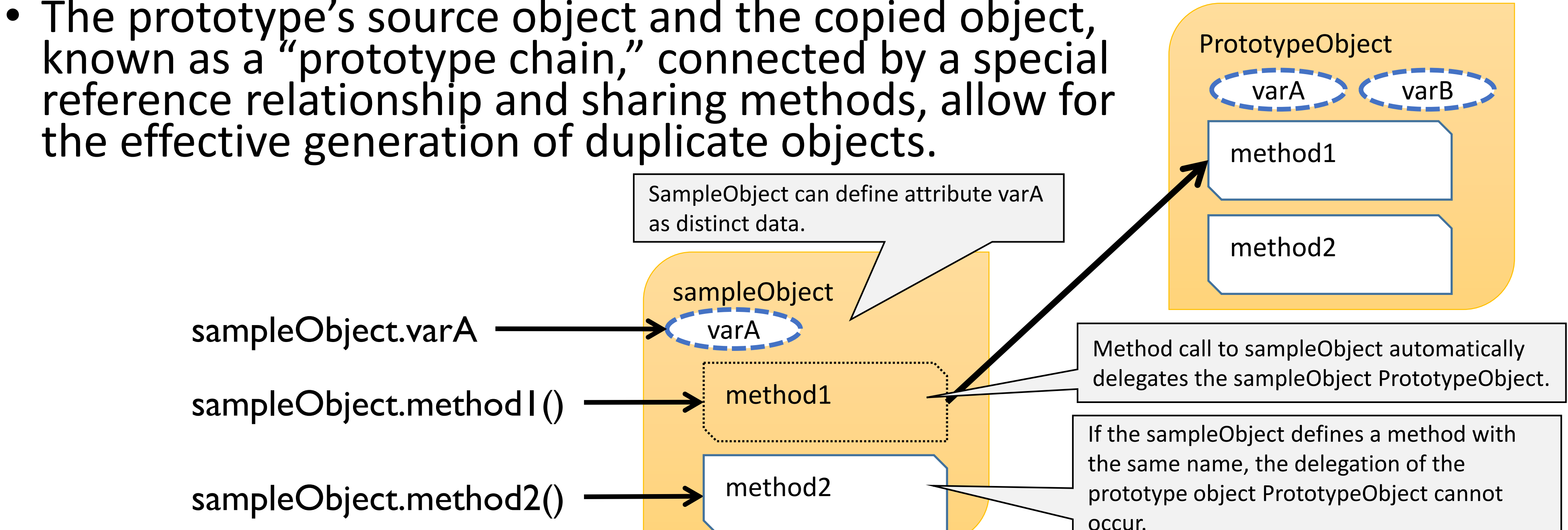
What is a Prototype?

- When one wants to mass-produce a similar object as a model of a certain object, JavaScript is able to duplicate this object. This model object is called a “prototype.”



Why Make a Prototype?

- Defining large quantities of objects causes the efficiency of memory to worsen and the efficiency of maintenance to lower on account of source code copies being developed.
- In particular, prototypes are used as a mechanism that allows various differing data to be kept, sharing only a function object (= method)
- The prototype's source object and the copied object, known as a "prototype chain," connected by a special reference relationship and sharing methods, allow for the effective generation of duplicate objects.



Prototype-Based Object Orientation in JavaScript: Three methods supporting prototype-based object orientation in JavaScript

Object.create()	• Object generation
Function.call()	• Change this context and **execute function.**
Function.bind()	• For a given function, **generates a new function**, set with a static this context.

Object.create()

- The Object.create() method accepts, as an argument, an object that will become a prototype, generates a new object that will take the object as a prototype, and returns it as a return value.
- Example: Person object duplication

```
var p = Object.create(Person);
p.firstName = "John";
p.lastName = "Smith";
p.email = "js@sfc.keio.ac.jp";
```

In the left example, a new object is generated that takes the Person object as a prototype, and is assigned to the variable p. In this process, the Person object never changes, and receives no influence from anything. “John” and “Smith” are only assigned to the newly generated object.

Associating a Function with an Object (Methodizing a Function)

- By declaring a function literal in an object literal, it is possible to create an object that has internalized the function.
    - Functions are also objects and object literals can contain other objects.
- A “method” is a function that has been internalized in an object.
- In a method, it is possible to use the other attributes of an object that the method is associated with through the use of “this” variable.

```
var calc = {
    a: 1,
    b: 2,
    add:  function () {
        return this.a + this.b;
    }
}

alert (  calc.add()  );
```

Prototype Chain Experiment

- Two objects are generated from the Person prototype, the attributes that each object has (variables and functions) are compared, and it is confirmed that only the functions are shared.

```
var Person = {
    firstName: "",
    lastName: "",
    email: "",
    init: function(first, last, email) {
        this.firstName = first;
        this.lastName = last;
        this.email = email;
    },
    equals: function(obj) {
        if (obj === null) {
            return false;
        }
        if (this.firstName !== obj.firstName) {
            return false;
        }
        return true;
    },
    compareTo: function(obj) {
        return this.firstName > obj.firstName ? 1 : -1;
    }
}
var p1 = Object.create(Person);
p1.init("Shuichi", "Kurabayashi", "kurabaya@sfc.keio.ac.jp");

var p2 = Object.create(Person);
p2.init("John", "Smith", "js@sfc.keio.ac.jp");

console.log("Variable: " + (p1.firstName === p2.lastName));
console.log("Function: " + (p1.equals === p2.equals));
```

The fact that (`p1.firstName === p2.lastName`) is false, that is to say that `p1.firstName` and `p2.lastName` are different, which means that the variables of the objects generated from the prototype keep separate data for each object.

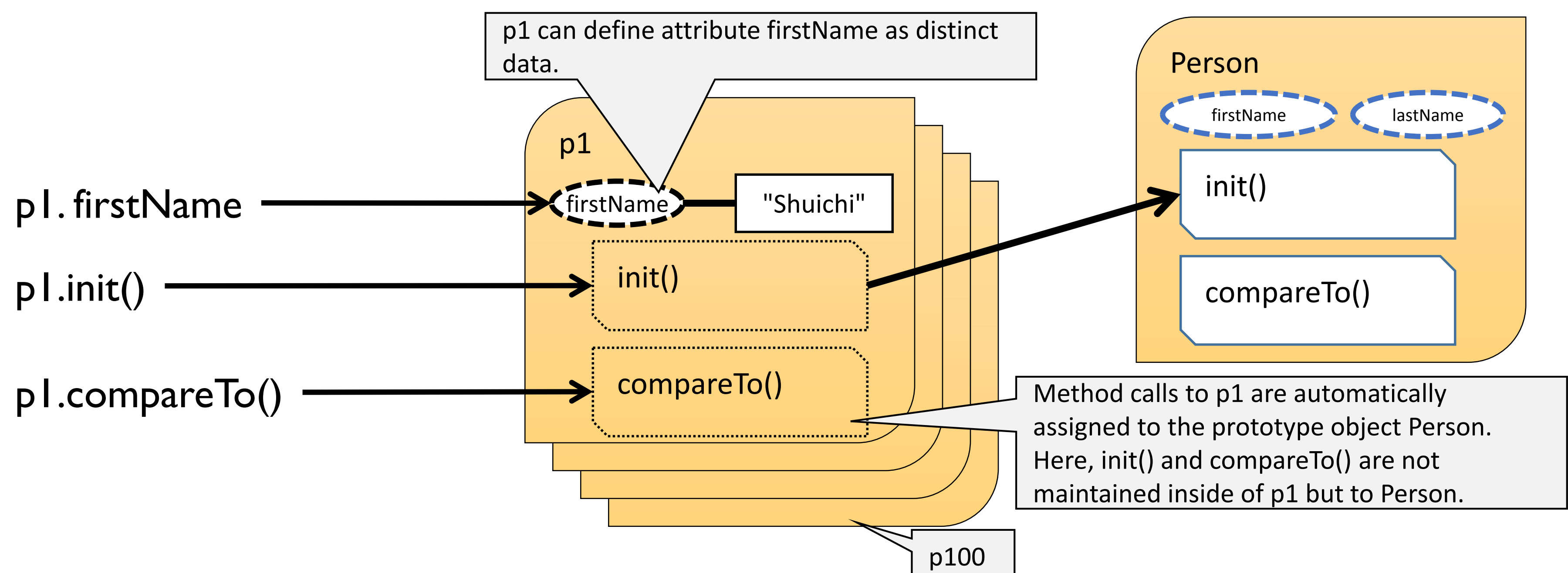
On the contrary, for functions, (`p1.equals === p2.equals`) is true, that is to say the `p1.equals` function is the same as the `p2.equals` function. This means that the variables of the objects generated from the prototype share a similar function.

Results Displayed on the Console

```
Variable: false
Function: true
```

Example of Prototype Use

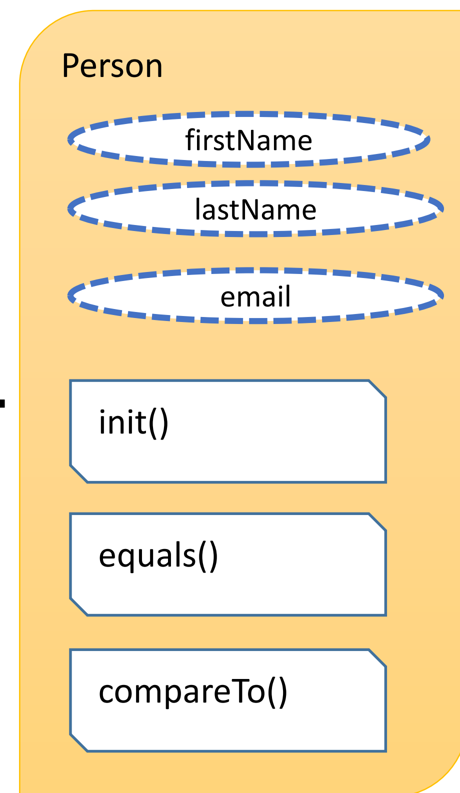
- Using the Person object as a prototype, consider a case where a large number of objects, object p1–p100, with the same features as Person, are generated.
- In each object only different data is stored in each of the 100 objects, and the method shares the prototype amongst all objects.



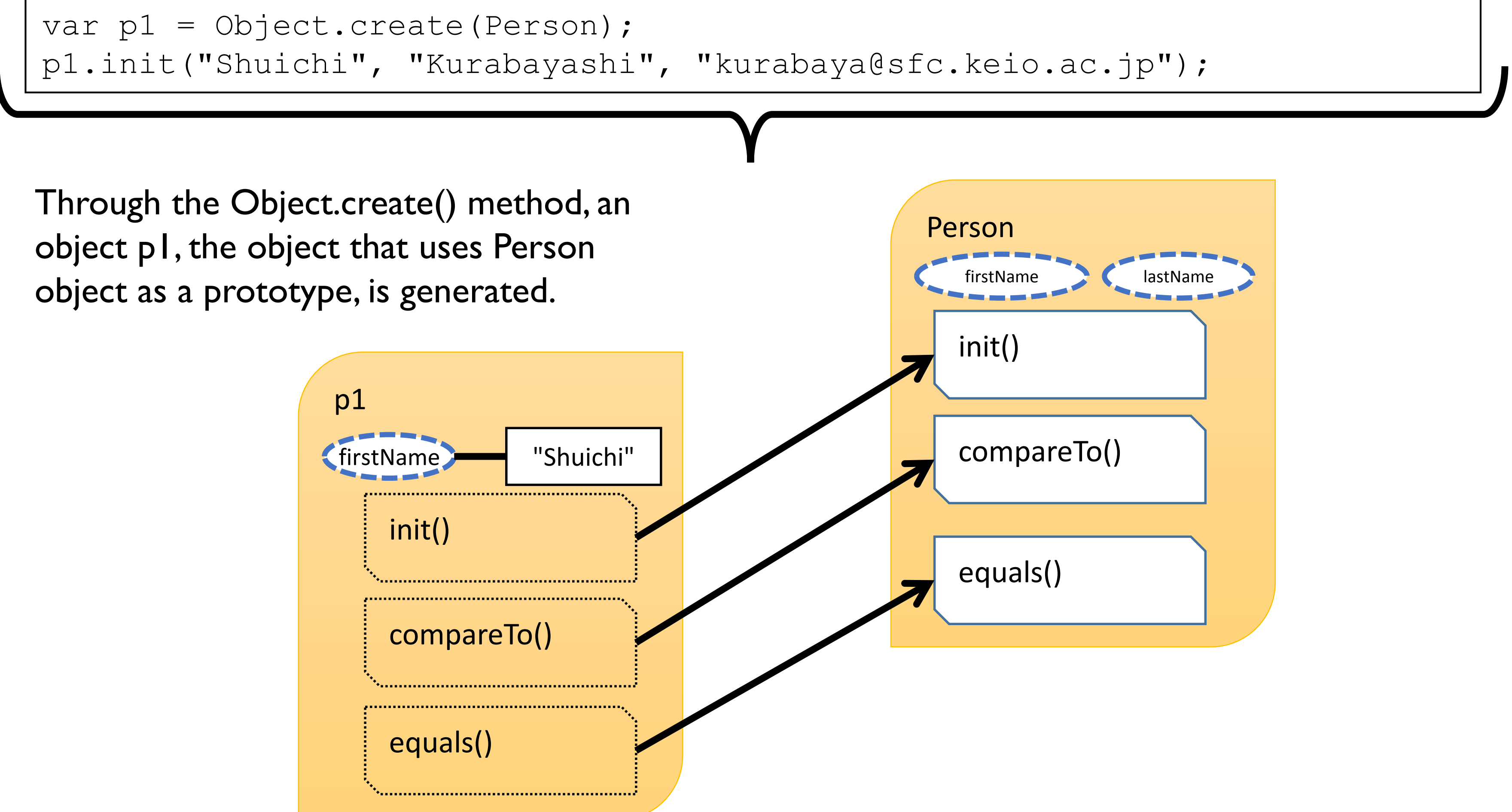
Prototype Use Procedure (1)
Prototype Object Generation

```
var Person = {
    firstName: "",
    lastName: "",
    email: "",
    init: function(first, last, email) {
        this.firstName = first;
        this.lastName = last;
        this.email = email;
    },
    equals: function(obj) {
        if (obj === null) {
            return false;
        }
        if (this.firstName !== obj.firstName) {
            return false;
        }
        return true;
    },
    compareTo: function(obj) {
        return this.firstName > obj.firstName ? 1 : -1;
    }
}
```

Person object is generated through the object literal {}.

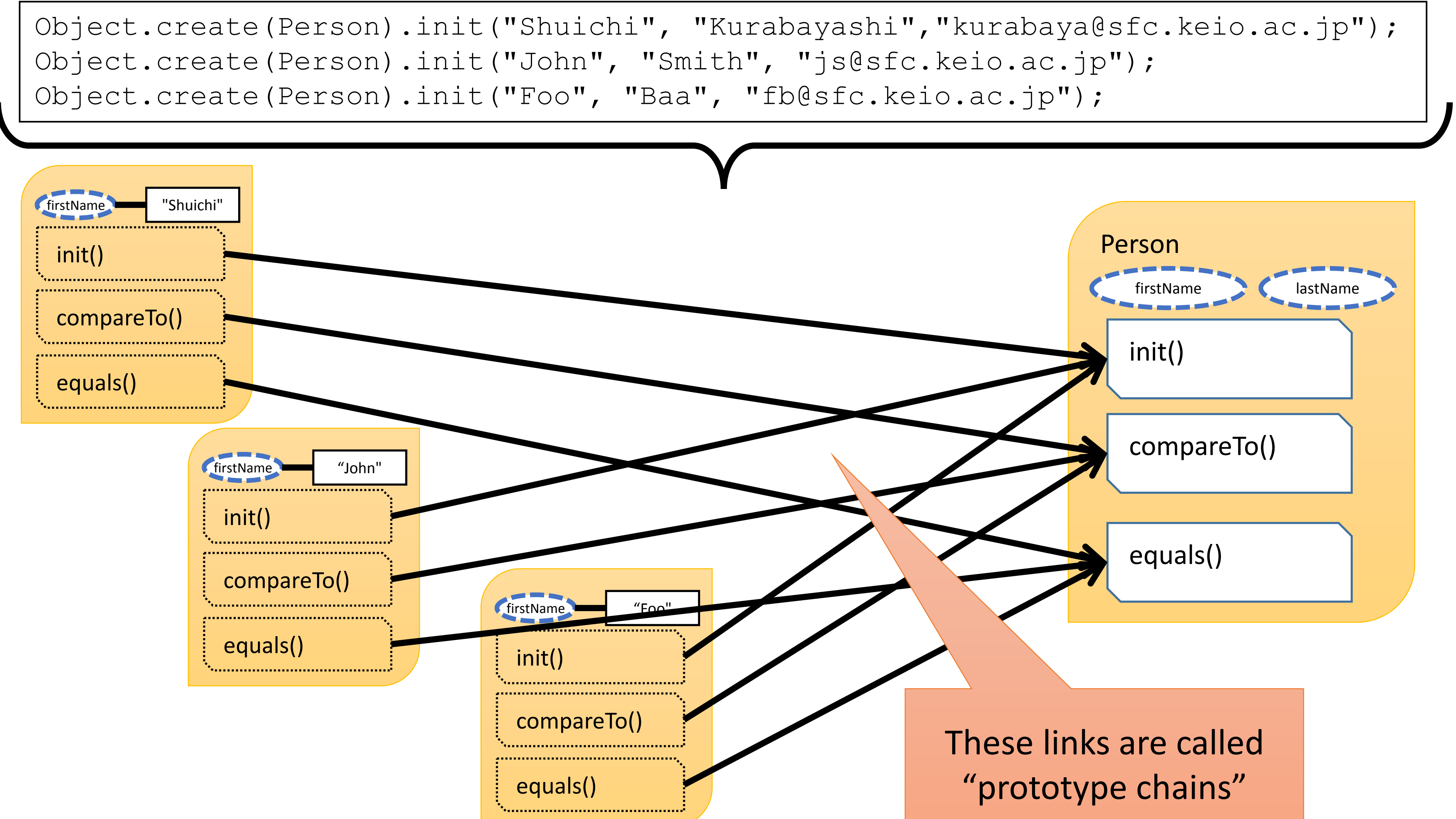


Prototype Use Procedure (2) Object Generation and Prototypes



Prototype Use Procedure (3)
Object Generation and Prototypes

```
Object.create(Person).init("Shuichi", "Kurabayashi","kurabaya@sfc.keio.ac.jp");
Object.create(Person).init("John", "Smith", "js@sfc.keio.ac.jp");
Object.create(Person).init("Foo", "Baa", "fb@sfc.keio.ac.jp");
```



Person Definition that Sorts by Last Name

- Change compareTo(), and try to carry out an age field comparison.
- By changing the definition of Person, all objects generated by Person are influenced. This is the effect of a prototype chain.

```
var Person = {
    firstName: "",
    lastName: "",
    email: "",
    compareTo: function(obj) {
        return this.lastName > obj.lastName ? 1 : -1;
    }
}
```

Output Results

```
Object { firstName="Foo", lastName="Baa", email="fb@sfc.keio.ac.jp"}
Object { firstName="Shuichi", lastName="Kurabayashi",
email="kurabaya@sfc.keio.ac.jp"}
Object { firstName="John", lastName="Smith", email="js@sfc.keio.ac.jp"}
```

Sorting Program

- By passing the variables used for sorting to the array sort( ) method, it is possible to organize variables in a fixed sequence.

```
var list=[], p, i;

p = Object.create(Person);
p.init("Shuichi", "Kurabayashi",
        "kurabaya@sfc.keio.ac.jp");
list.push(p);

p = Object.create(Person);
p.init("John", "Smith", "js@sfc.keio.ac.jp");
list.push(p);

p = Object.create(Person);
p.init("Foo", "Baa", "fb@sfc.keio.ac.jp");
list.push(p);

list.sort(function(a,b){return a.compareTo(b)});

for (i = 0; i < list.length; i+= 1) {
    console.log(list[i]);
}
```

Execution Results

```
Foo Baa { firstName="Foo", lastName="Baa", email="fb@sfc.keio.ac.jp"}
Shuichi Kurabayashi { firstName="Shuichi", lastName="Kurabayashi", email="kurabaya@sfc.keio.ac.jp"}
John Smith { firstName="John", lastName="Smith", email="js@sfc.keio.ac.jp"}
```

An Even Smarter Sorting Program

- By adding a method for sorting to the Array.prototype, it is possible to add the sorting feature corresponding to compareTo() to all arrays.

```
Array.prototype.objectSort = function(){
    this.sort(function(a,b){return a.compareTo(b)});
};

var list=[], p, i;
p = Object.create(Person);
p.init("Shuichi", "Kurabayashi",
"kurabaya@sfc.keio.ac.jp");
list.push(p);
p = Object.create(Person);
p.init("John", "Smith", "js@sfc.keio.ac.jp");
list.push(p);
p = Object.create(Person);
p.init("Foo", "Baa", "fb@sfc.keio.ac.jp");
list.push(p);

list.objectSort();

for (i = 0; i < list.length; i+= 1) {
    console.log(list[i]);
}
```

Execution Results

```
Foo Baa { firstName="Foo", lastName="Baa", email="fb@sfc.keio.ac.jp"}
Shuichi Kurabayashi { firstName="Shuichi", lastName="Kurabayashi", email="kurabaya@sfc.keio.ac.jp"}
John Smith { firstName="John", lastName="Smith", email="js@sfc.keio.ac.jp"}
```

Complete Version of the Person Prototype

- The Person prototype has three attributes, firstName, lastName, and email, and the methods init(), equals(), compareTo(), and toString().

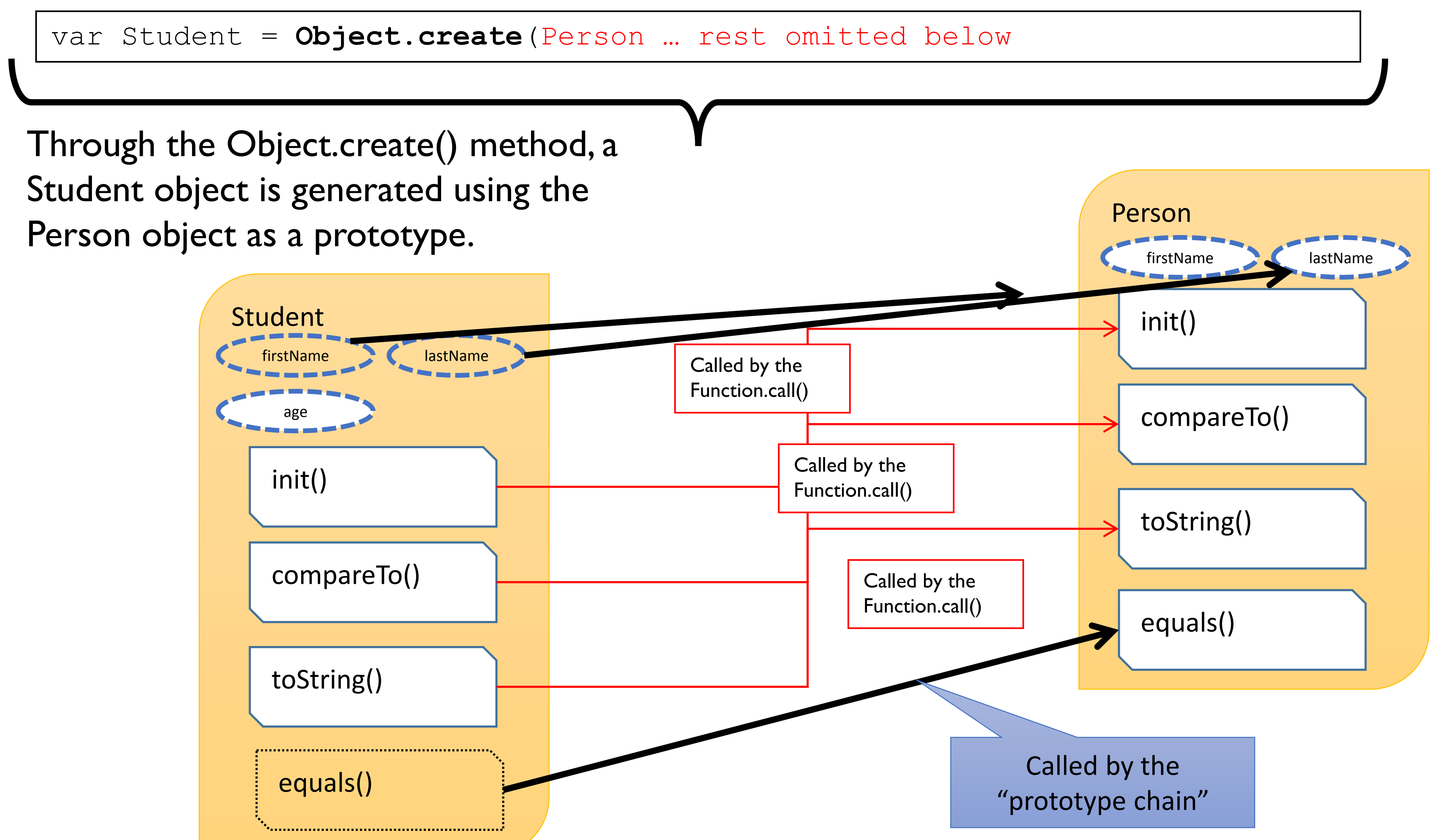
```
var Person = {
    firstName: "",
    lastName: "",
    email: "",
    init: function(first, last, email) {
        this.firstName = first;
        this.lastName = last;
        this.email = email;
    },
    equals: function(obj) {
        if (obj === null) {
            return false;
        }
        if (this.firstName !== obj.firstName) {
            return false;
        }
        return true;
    },
    compareTo: function(obj) {
        return this.firstName > obj.firstName ? 1 : -1;
    },
    toString: function() {
        return this.firstName + " " + this.lastName;
    }
}
```

Prototype Inheritance

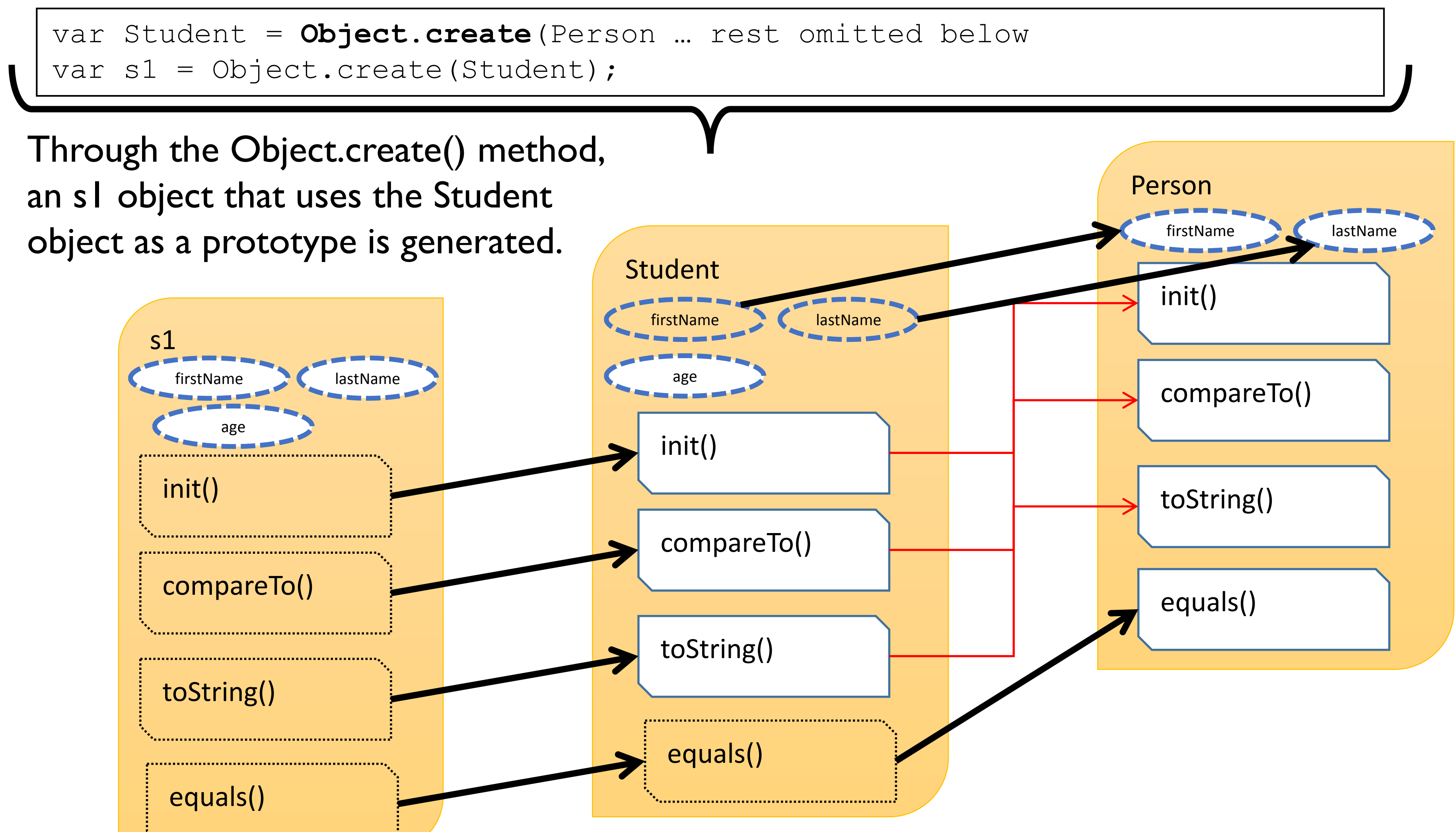
- The Student object is generated using Person as a prototype that has a new age field, and can be sorted in order of age.

```
var Student = Object.create(Person, {
  age: {
    value: 0,
    writable: true
  },
  init: {
      value: function(first, last, email, age) {
          Person.init.call(this, first, last, email);
          this.age = age;
      }
  },
  compareTo: {
    value: function(obj) {
        var result;
        if (this.age === obj.age) {
            result = Person.compareTo.call(this, obj);
        } else {
            result = this.age > obj.age ? 1 : -1;
        }
        return result;
    }
  },
  toString: {
    value: function() {
        return Person.toString.call(this) + " (" + this.age + ")";
    }
  }
});
```

Prototype Inheritance



Object Generation after Prototype Inheritance



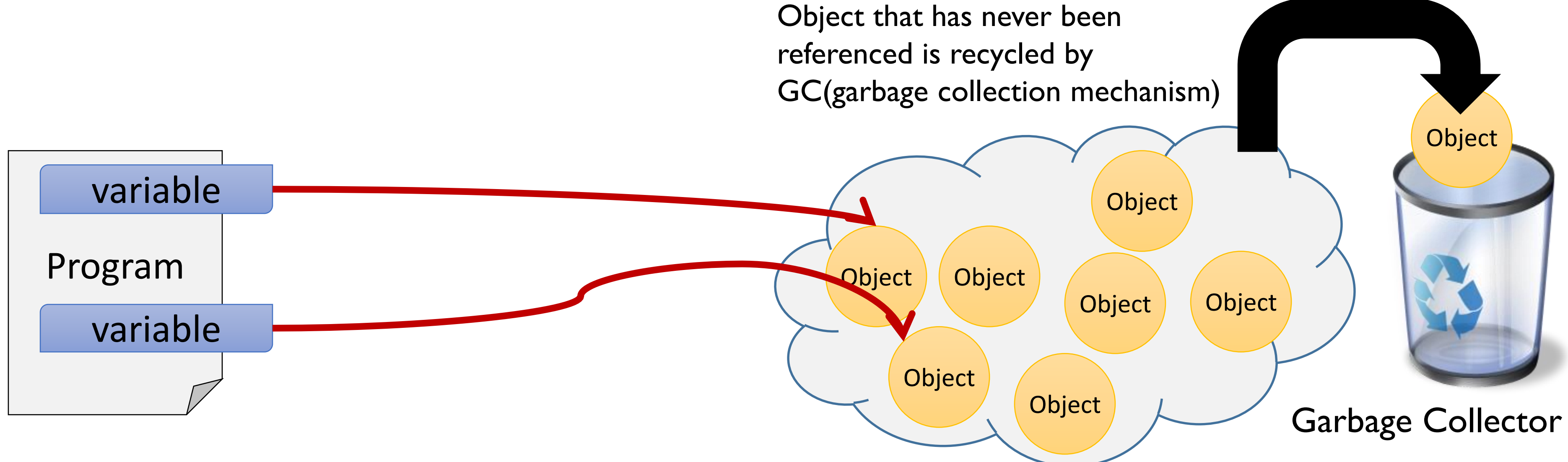
Object Generation using Object.create()

- By passing the empty object literal {} as a parameter of the method Object.create(), it is possible to build objects starting from zero.
- The advantage of using the Object.create() method is that you can finely control the object field attributes.

```
var Person = Object.create({}, {
    firstName: {
        value: "",
        writable: true
    },
    lastName: {
        value: "",
        writable: true
    },
    email: {
        value: "",
        writable: true
    },
    equals: {
        value: function(obj) {
            if (obj === null) {
                return false;
            }
            if (this.firstName !== obj.firstName) {
                return false;
            }
            return true;
        }
    },
    compareTo: {
        value: function(obj) {
            return this.firstName > obj.firstName ? 1 : -1;
        }
    },
    toString: {
        value: function() {
            return this.firstName + " " + this.lastName;
        }
    }
});
```

The Relationship between Variables and Objects

- Objects that are not being used are automatically deleted as garbage.
- Variables exist to reference objects.
  - Variables are structures for naming and holding objects.
  - No variables, No objects. Objects must be referenced by variables. Non-referenced objects are garbage.



Function.bind()

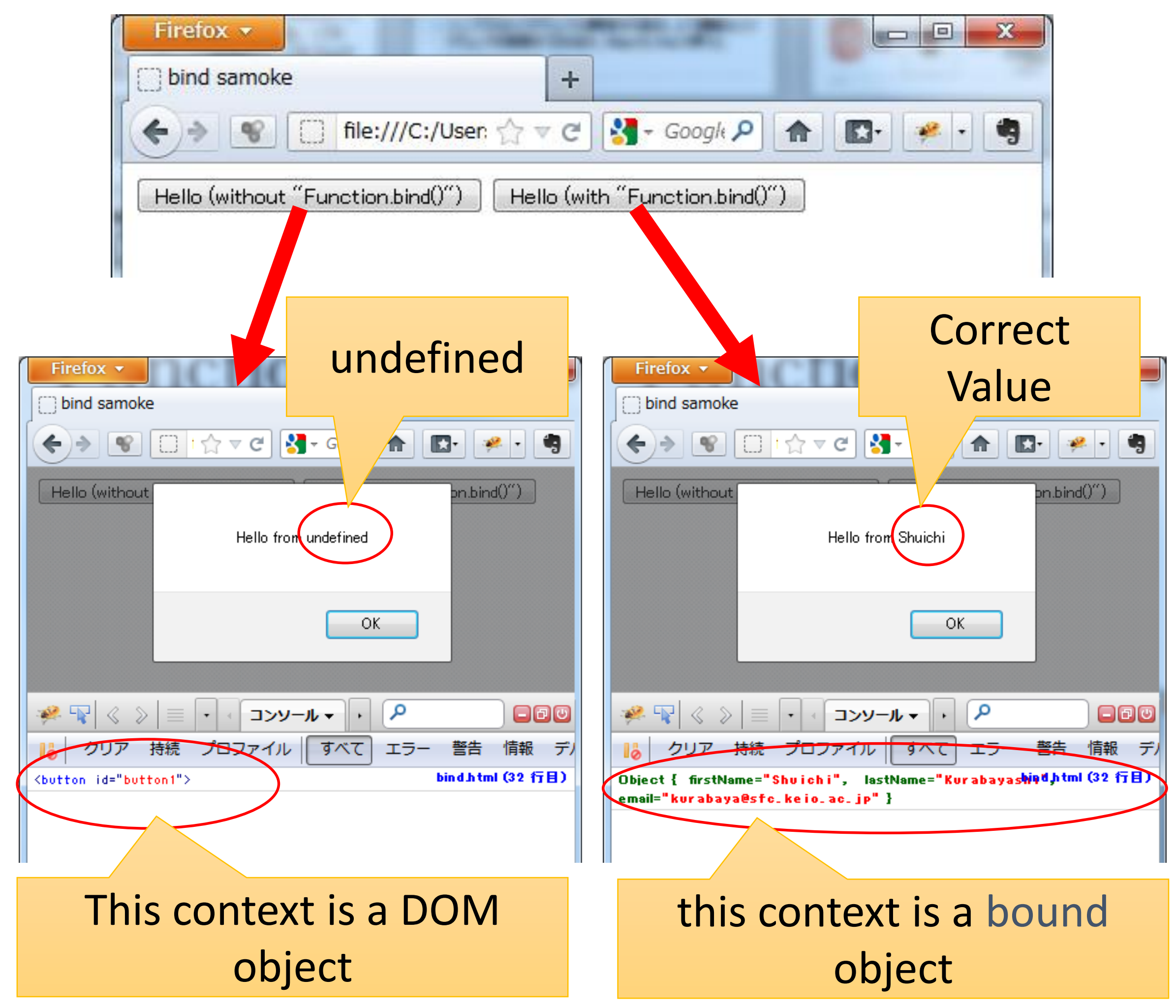
- When linking to the web browser, a function unit called a “callback function” links to the web browser but cannot be linked at the object unit.
  - Halves object-oriented merits.
- Therefore, using Function.bind() generates an event handler that can call an object-oriented method

```
<html>
<head>
<title>bind samoke</title>
</head>
<body>
<button id="button1">Hello (without "Function.bind()")</button>
<button id="button2">Hello (with "Function.bind()")</button>

<script type="text/javascript">
var Person = {
    firstName: "",
    lastName: "",
    email: "",
    init: function(first, last, email) {
        this.firstName = first;
        this.lastName = last;
        this.email = email;
    },
    equals: function(obj) {
        if (obj === null) {
            return false;
        }
        if (this.firstName !== obj.firstName) {
            return false;
        }
        return true;
    },
    compareTo: function(obj) {
        return this.firstName > obj.firstName ? 1 : -1;
    },
    helloEvent: function(event) {
        console.log(this);
        alert("Hello from " + this.firstName);
    }
};

var p1 = Object.create(Person);
p1.init("Shuichi", "Kurabayashi", "kurabaya@sfc.keio.ac.jp");

document.getElementById("button1").addEventListener("click", p1.helloEvent);
document.getElementById("button2").addEventListener("click", p1.helloEvent.bind(p1));

</script>

</body>
</html>
```

Function.bind() Demo

http://web.sfc.keio.ac.jp/~kurabaya/lecture/javascript/bind.html

- If you press the Hello button (without "Function.bind()") because Function.bind() is not used, a pertinent object cannot be referenced.
  - this context lost
- If you press the Hello button (with "Function.bind()"), because Function.bind() is being used, a pertinent object can be referenced.



bind.html's Full Source

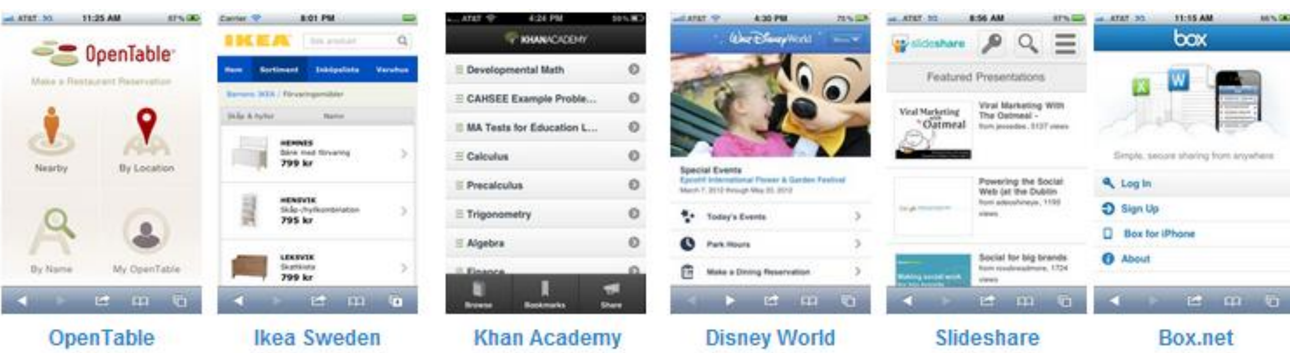
```
<html>
<head>
<title>bind samoke</title>
</head>
<body>
<button id="button1">Hello (without "Function.bind()")</button>
<button id="button2">Hello (with "Function.bind()")</button>

<script type="text/javascript">
var Person = {
    firstName: "",
    lastName: "",
    email: "",
    init: function(first, last, email) {
        this.firstName = first;
        this.lastName = last;
        this.email = email;
    },
    equals: function(obj) {
        if (obj === null) {
            return false;
        }
        if (this.firstName !== obj.firstName) {
            return false;
        }
        return true;
    },
    compareTo: function(obj) {
        return this.firstName > obj.firstName ? 1 : -1;
    },
    helloEvent: function(event) {
        console.log(this);
        alert("Hello from " + this.firstName);
    }
};

var p1 = Object.create(Person);
p1.init("Shuichi", "Kurabayashi", "kurabaya@sfc.keio.ac.jp");

document.getElementById("button1").addEventListener("click", p1.helloEvent);
document.getElementById("button2").addEventListener("click", p1.helloEvent.bind(p1));

</script>

</body>
</html>
```

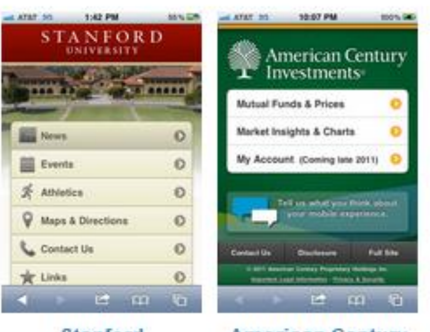
Objects and Frameworks

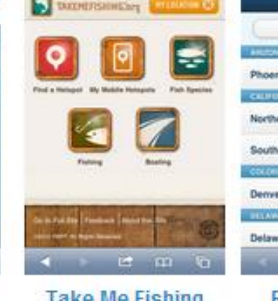
jQuery and jQuery Mobile for Developing Advanced Mobile Applications

Introduction to jQuery Mobile

- jQuery Mobile is a web application framework built on the top of jQuery.
- jQuery Mobile provides a rich GUI widgets for web applications, such as Navigation Bar, Buttons, Pages and Dialogs.
- You can customize the default behavior of jQuery Mobile by writing JavaScript codes.







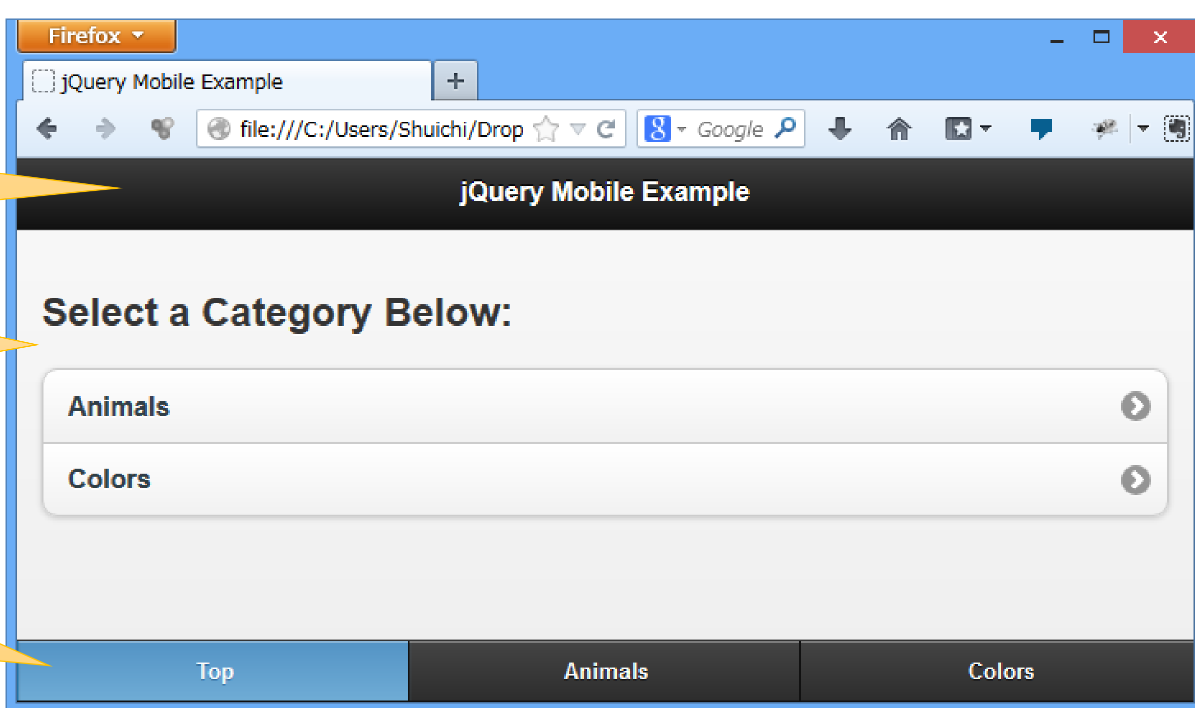
Your First jQuery Mobile Application (http://web.sfc.keio.ac.jp/~kurabaya/lectures/jqmobile.html)

- jQuery Mobile application consists of three elements:
  - Header
  - Content
  - Footer (Navigation Bar)

Header shows title and subtitle, additional buttons

Content shows main information of the application. It contains various widgets such as buttons, list, images.

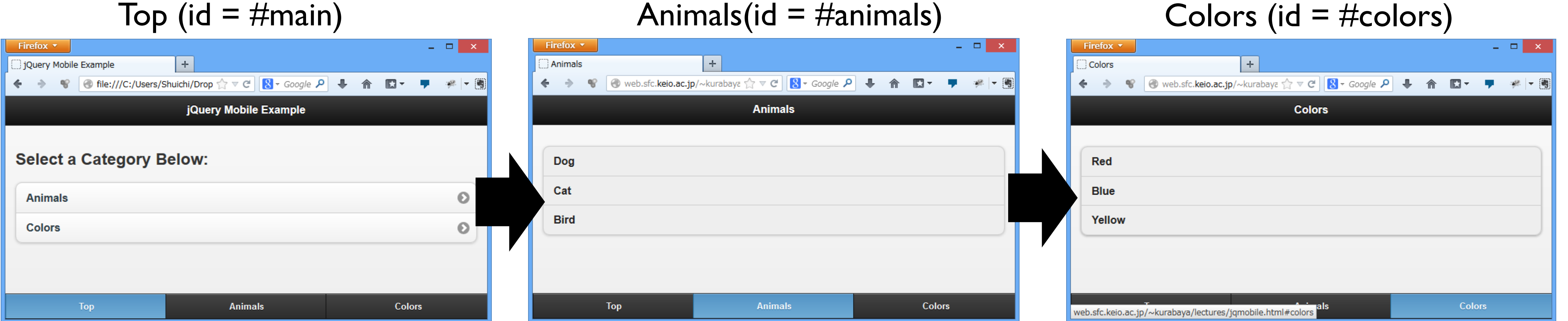
Footer, also called as Navigation Bar, shows several buttons for navigating pages and dialogs.



Page Navigation

- jQuery Mobile provides a page navigation function even if multiple pages are written in a single HTML.
- You can define multiple pages using <div data-role="page"> tag.
- You can link to other page by defining hyperlink for other <div data-role="page">. For example, a user will go to main page by clicking Top.

Top (id = #main) Animals(id = #animals) Colors (id = #colors)



Internal Structure of the First Application

- To use jQuery Mobile, you must link jquery.mobile-*.min.css and loads jquery-*.min.js and jquery.mobile-*.min.js .
- Main content can be defined using <div data-role="page" id="main"> tag.
- Main div tag consists of header, content, and footer DIV tag.

```
<!DOCTYPE html>
<html>
<head>
    <title>jQuery Mobile Example</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.1/jquery.mobile-1.3.1.min.css"/>
    <script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
    <script src="http://code.jquery.com/mobile/1.3.1/jquery.mobile-1.3.1.min.js"></script>
</head>
<body>

<div data-role="page" id="main">

    <div data-role="header">
        <h1>jQuery Mobile Example</h1>
    </div>

    <div data-role="content">
        <h2>Select a Category Below:</h2>
        <ul data-role="listview" data-inset="true">
            <li><a href="#animals" class="animals">Animals</a></li>
            <li><a href="#colors" class="colors">Colors</a></li>
        </ul>
    </div>

    <div data-role="footer" data-id="navbar" data-position="fixed">
        <div data-role="navbar">
            <ul>
                <li><a href="#main" class="ui-btn-active ui-state-persist">Top</a></li>
                <li><a href="#animals">Animals</a></li>
                <li><a href="#colors">Colors</a></li>
            </ul>
        </div>
    </div>
</div>

</body>
</html>
```

Header

- You can define a header by specifying data-role of DIV tag as "header".
- Header consists of H1 tag showing title of this page, and optional buttons. In this example, I omit the optional buttons.
- Header must display current page's name because it is important for users to understand where they are at.

```
<div data-role="page" id="main">
    <div data-role="header">
        <h1>jQuery Mobile Example</h1>
    </div>
</div>
```

Content

- You can define a content area by specifying data-role of DIV tag as "content".
- You can use a lot of widget (please see http://view.jquerymobile.com/1.3.1/dist/demos/examples/) in content area.
- The most convenient widget is "listview" that provides a rich UI for <ul> tag.

```
<div data-role="content">
  <h2>Select a Category Below:</h2>
  <ul data-role="listview" data-inset="true">
      <li><a href="#animals" class="animals">Animals</a></li>
      <li><a href="#colors" class="colors">Colors</a></li>
  </ul>
</div>
```

Footer (Navigation Bar)

- Footer is a special component that persists on the bottom of the page.
- Footer provides several buttons for navigating among pages.
- You can define the navigation bar by specifying data-role="navbar"
- Inside of the navbar, class="ui-btn-active ui-state-persist“ means the button is selected currently.

```
<div data-role="footer" data-id="navbar" data-position="fixed">
    <div data-role="navbar">
        <ul>
            <li><a href="#main" class="ui-btn-active ui-state-persist">Top</a></li>
            <li><a href="#animals">Animals</a></li>
            <li><a href="#colors">Colors</a></li>
        </ul>
    </div>
</div>
```

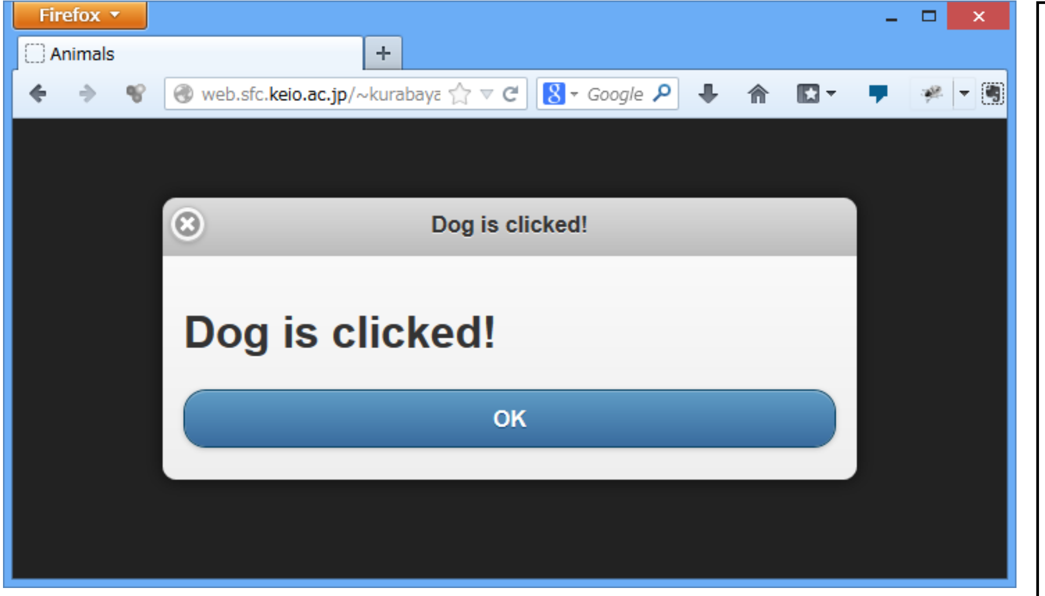
JavaScript for jQuery Mobile

- This example generates data set dynamically.
- This example also binds event listeners for each list item. When a user clicks them, the dialog will show the corresponding data.
- $() is a core function provided by jQuery.
- A variety of operations can executed via $()-wrapped objects.
    - empty() deletes all child nodes
    - append() adds HTML elements

```
var animals = ['Dog', 'Cat', 'Bird'];
var colors = ['Red', 'Blue', 'Yellow'];

animals.forEach(function (item) {
    var li = $('<li class="ui-li"></li>');
    li.text(item);
    $("#animals ul[data-role=listview]").append(li);
    li.click(function (e) {
        $("div[data-role=dialog] h1").html(item + " is clicked!");
        $.mobile.changePage('#dialog', 'pop', true, true);
    });
});

colors.forEach(function (item) {
    var li = $('<li class="ui-li"></li>');
    li.text(item);
    $("#colors ul[data-role=listview]").append(li);
    li.click(function (e) {
        $("div[data-role=dialog] h1").html(item + " is clicked!");
        $.mobile.changePage('#dialog', 'pop', true, true);
    });
});
```

Dialog

- You can define a popup dialog by specifying data-role of DIV tag as “dialog".
- You can open the dialog by calling the following JavaScript method.
  - `$.mobile.changePage('#dialog', 'pop', true, true);`




```
<div data-role="dialog" id="dialog">
  <div data-role="header" data-theme="d">
    <h1>Dialog</h1>
  </div>

  <div data-role="content">
    <h1></h1>
    <a data-role="button" data-rel="back" data-theme="b">OK</a>
  </div>
</div>
```

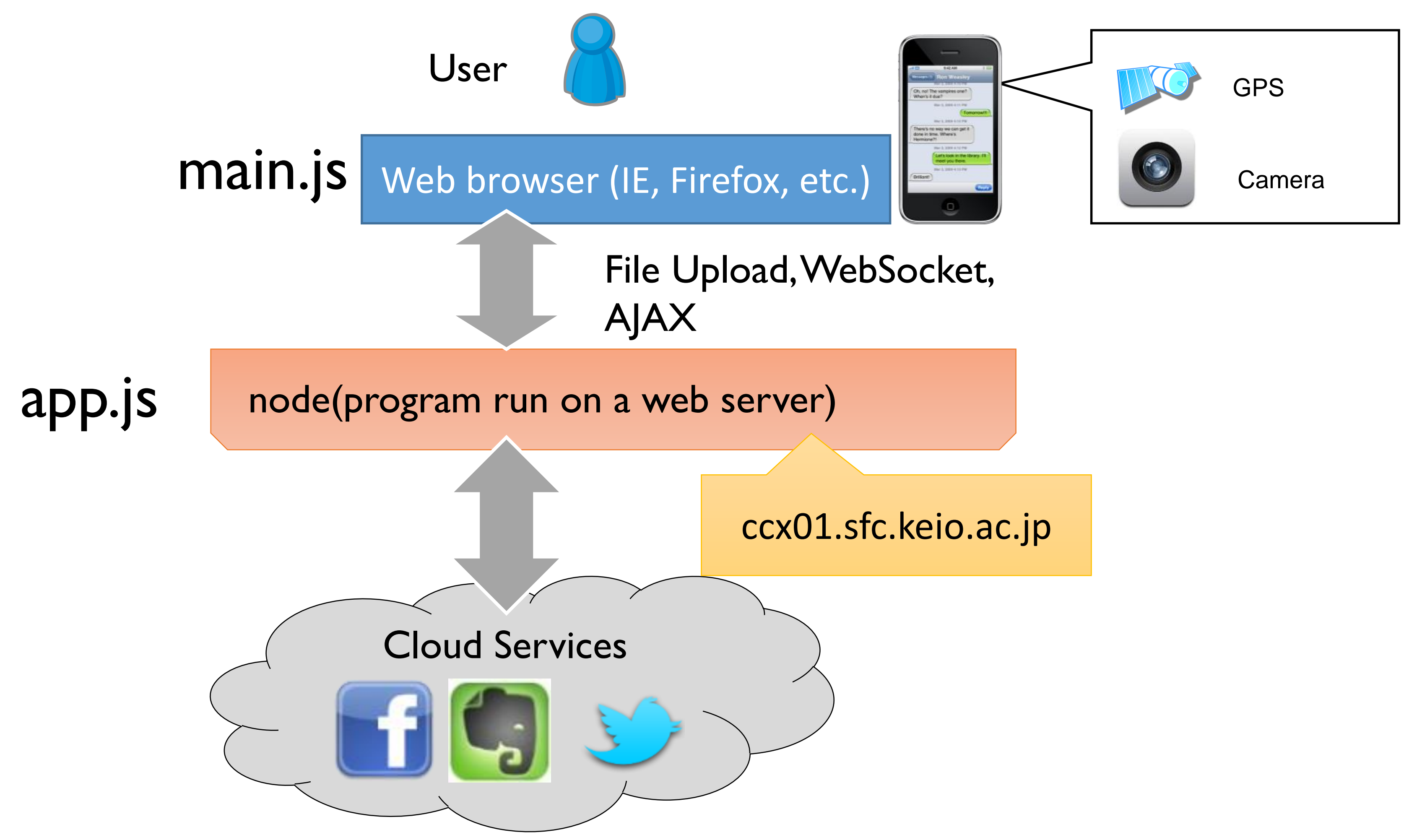
Exercise 7

- Let’s append a 4th page for the example.
- Define another dialog.
- Add event listener for the 4th page for showing your own dialog.

Node.JS

Server-Side JavaScript Engine

Nodes and Web Application



Node.JS

- Server-side JavaScript Engine
  - This was developed by combining the V8 JavaScript engine, developed by Google, with an Event Loop library (libev/libeio, IOCP in the case of Windows) to be an environment for executing highly efficient server-side programs.
  - Node.js was developed in 2009 by Ryan Dhal, and later in November of 2010 was transferred to Joylent, an American Cloud Services Business.
- Event-driven programming
  - Single Thread, Nonblocking I/O
  - Does not require knowledge for carrying out complex multithread programming such as that used in C and Java.
- What is Nonblocking I/O?
  - As I/O operations are generally 10 to 1000 times slower than the CPU, data sending and receiving via a network or file input and output creates a CPU “wait” time until operations are completed.
  - Nonblocking I/O immediately continues to the next program action.
  - I/O operation completion is received after as an “event.”

Install

- http://nodejs.org/
- In the case of MacOSX, it is good to install homebrew, and install via homebrew (required for xcode4 and after).
  - http://mxcl.github.com/homebrew/In

Accessing ccx01 using TeraTerm

- Log into ccx01.sfc.keio.ac.jp using TeraTerm, and confirm that it is working. TeraTerm is terminal software for accessing a UNIX machine from Windows. TeraTerm was made available as open source and can be downloaded from http://ttssh2.sourceforge.jp. In order to access ccx01, the server where node.js is installed, input ccx01.sfc.keio.ac.jp for host and press the OK button as shown below.

Method for Launching a Server Process

- Combining the Nohup command >&, launches the server process.
- The Nohup command is a command that continues the process even after log out.

```
% nohup node app.js >& log.txt &
```

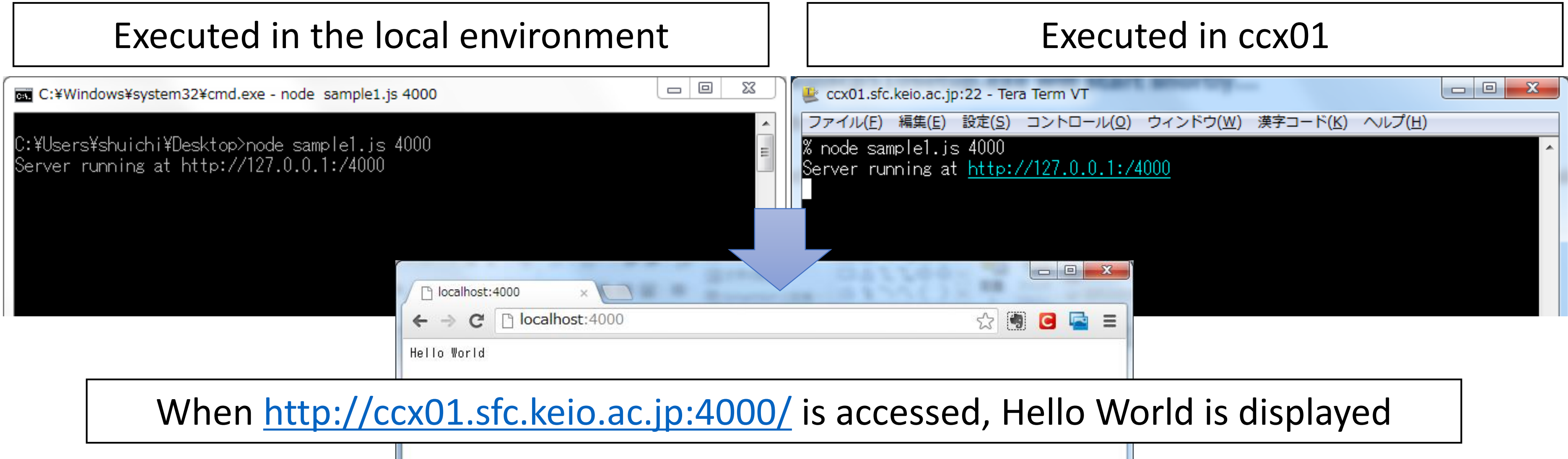
The Easiest Web Server

```
1  var http = require('http');
2  var server = http.createServer(function (req, res) {
3      res.writeHead(200, {'Content-Type':'text/plain'});
4      res.end('Hello World¥n');
5  });
6  server.listen(process.argv[2], '127.0.0.1');
7  console.log('Server running at http://127.0.0.1:'+process.argv[2]);
```

- var http = require('http');
  - The require function reads the various features of node.js. Here http module, which uses the web-related feature, is being read.
- http.createServer(...)
  - This calls the http module’s createServer function, and an object that governs the server feature is generated.
- function (req, res) {...}
  - A callback function dealing with request events is designated as an argument to the createServer function. “req” is an object representing HTTP request, and “res” is an object representing HTTP responses.
- res.writeHead(200, { ‘Content-Type’: ‘text/html’ });」
  - The res object’s writeHead function designates a status code and HTTP header, and transmits an HTTP response header.
- res.end(‘Hello, world!’);
  - The res object end function transmits the argument “Hello, world!” as a character string, and notifies node.js of the response generation’s completion.
- .listen(process.argv[2]);
  - The server object listen function begins the process of accepting the connection using the designated port number. Accordingly, by launching this function, access from the actual web server actually begins to be accepted.

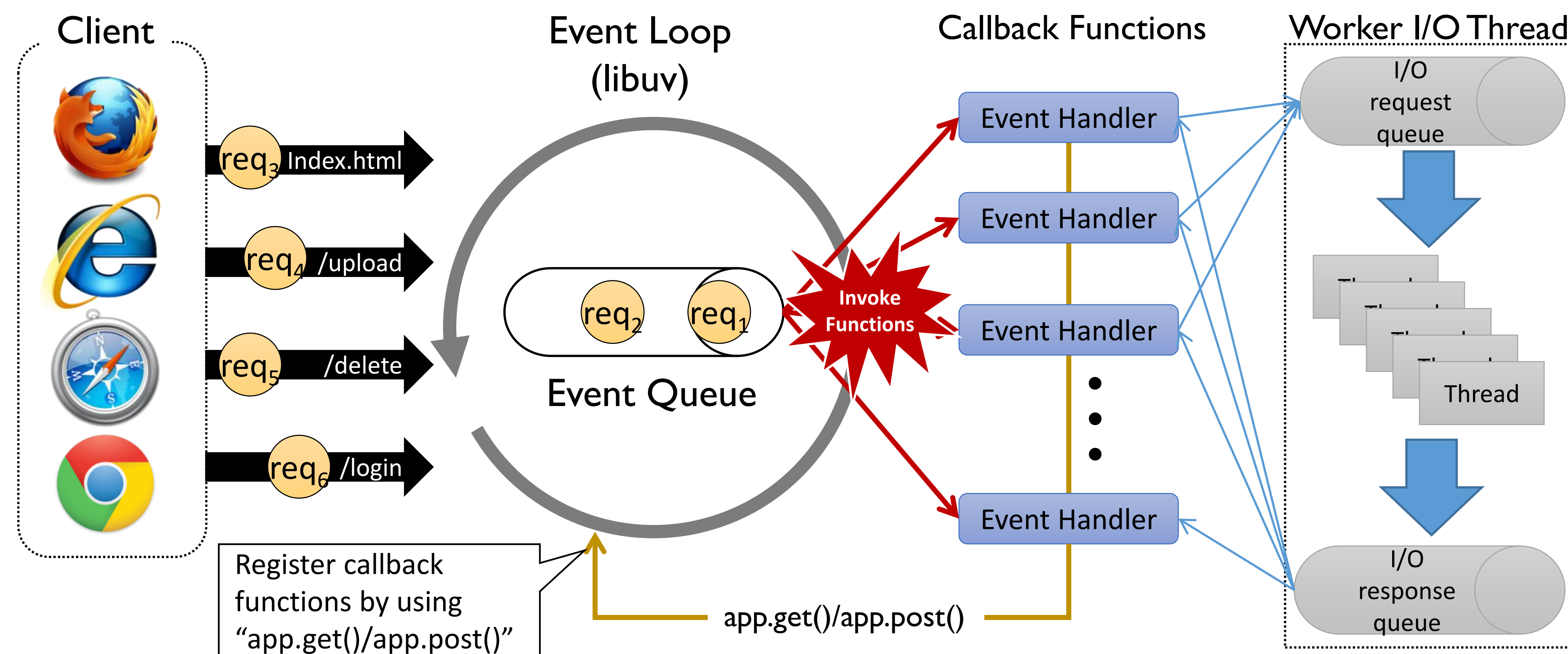
Web Server Execution

- The web server can be executed by inputting the node JavaScript file name and the port number.
- In the event it is executed in ccx01, the server on CNS inputs a number that will not overlap with another user's number, such as a school register number. This is necessary because port numbers cannot overlap.
- Accessing http://ccx01.sfc.keio.ac.jp:4000/, will allow you to check the web server's operation. (individually change the port number)



Node.JS Event Loop

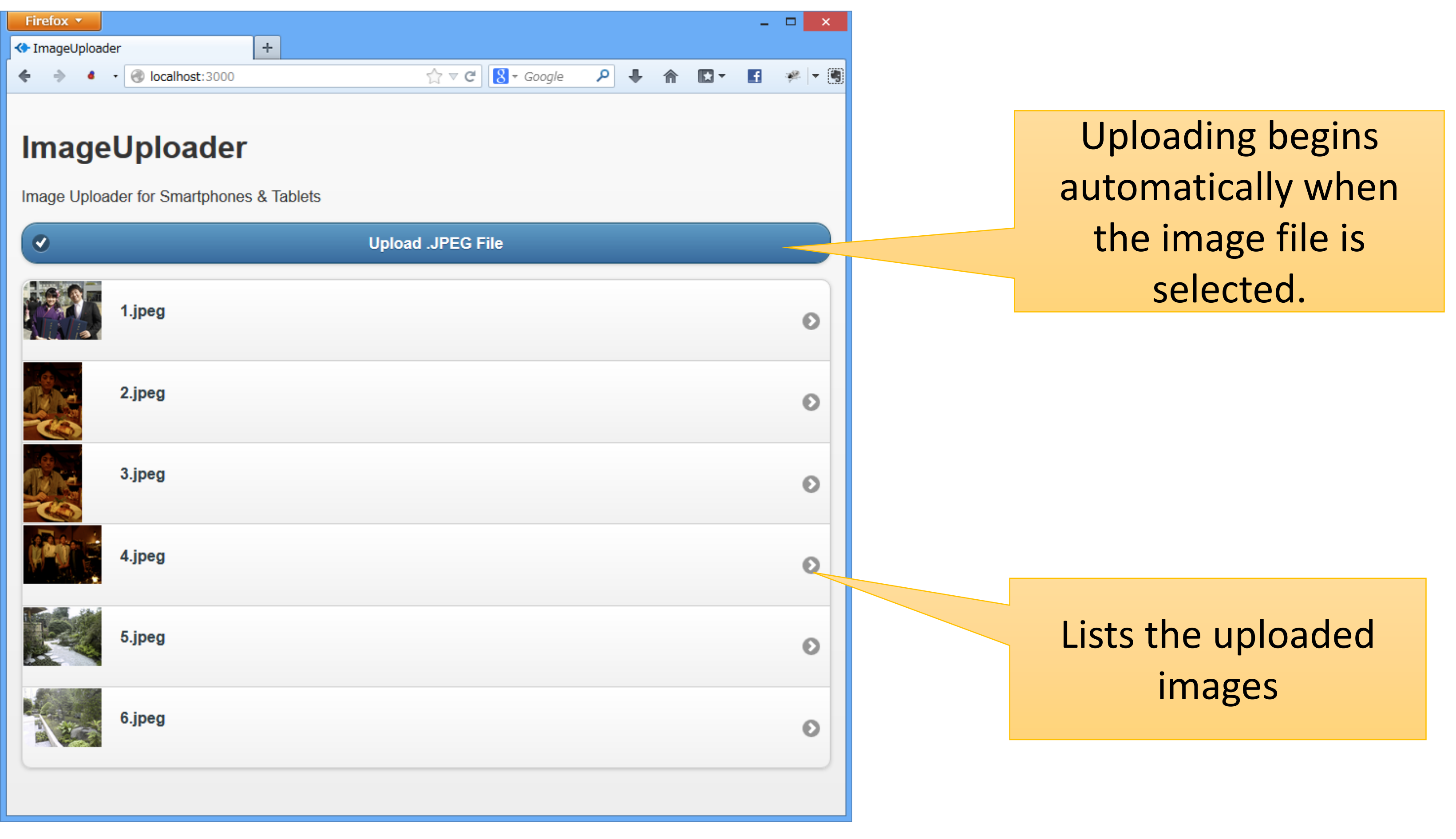
- "events" (get/post from URL, etc.) are transmitted in succession from the client and are stored in the event queue, which carries out a method for processing each successive event. This event processing function is called an Event Handler and is registered through the app.get()/app.post() method as a callback function.
- The main thread that executes the event loop and the worker I/O thread that carries out real-time processing exist, and each I/O real-time process is congruently processed.
- Callback is all processed in the event loop in the main thread.



Application Example

An Image Uploader that can be made using 100 lines of code.

An Image Uploader for use with Smart Phones and Tablets



Uploading begins automatically when the image file is selected.

Lists the uploaded images

Experimental Environments

- Full-Stacked JavaScript Environments are highly dynamic programming environments for creating mobile-based web applications.

http://jquerymobile.com/

http://nodejs.org/

express 3.0.0
web
application
framework for
node

http://expressjs.com/

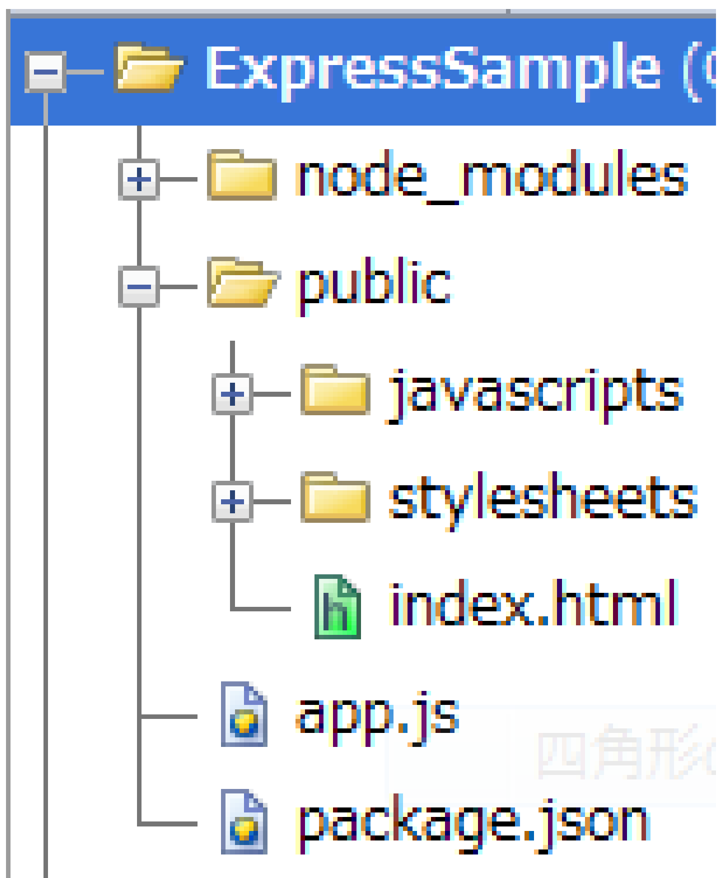
Client-Side JavaScript Framework

Server-Side JavaScript Framework

Web Application Framework

Structuring a Web Server Directory Using Express

- node_modules
  - A directory with the node.js module installed for the purpose of executing a web application. There is no need to change it.
- Public
  - A directory for storing files distributed as static files. Deploys outside JS libraries such as jquery, css file, or the client use.jp file prepared by you. When accessed from the client, files are directly accessed below public; therefore, for example, file "public/index.html" can be accessed as "http://localhost:4000/index.html."
- app.js
  - A node.js program that is executed as a server.
- package.json
  - A file for describing a web application's dependent module etc. There is no need to change it.



Index.html

Loads jQuery, jQueryMobile, and CSS

Markup according to jQueryMobile Use (data-role)

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>ImageUploader</title>
    <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.1/jquery.mobile-1.3.1.min.css"/>
    <script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
    <script src="http://code.jquery.com/mobile/1.3.1/jquery.mobile-1.3.1.min.js"></script>
</head>
<body>
<div data-role="page" id="home" class="type-home">
    <div data-role="content">
        <div>
            <h1>ImageUploader</h1>

            <p>Image Uploader for Smartphones &amp; Tablets</p>
            <button id="uploadButton" data-icon="check" data-theme="b">Upload .JPEG File</button>
        </div>
        <div id="listHolder"></div>
    </div>
</div>
<input type="file" name="file" style="visibility: hidden"/>
<script type="text/javascript">……</script>
</body>
</html>
```

Button for Uploading File

Invisible input[file] tag invoked by the above button

app.js is the main file in node.js Settings for Express Use

```
var express = require('express')
    , http = require('http')
    , path = require('path')
    , fs = require('fs');

var app = express();

app.configure(function () {
    app.set('port', process.argv[2] || 3000);
    app.use(express.favicon());
    app.use(express.logger('dev'));
    app.use(express.bodyParser());
    app.use(express.methodOverride());
    app.use(app.router);
    app.use(express.static(path.join(__dirname, 'public')));
});
```

Require() loads the node modules that are parameters, and returns objects.

Creates an express framework object

Express settings

Function Registering Using Get Function

```
app.get(url, function (req, res) {
    res.send({…});
});
```

- get()
  - Among HTTP methods, this deals with what method this function should react with. These methods are GET/POST/PUT/DELETE.
- url
  - This function points out the launched URL. For example, in the case of “list,” when http://localhost:3000/list is accessed, this function is executed.
- req
  - An object that stores request information for the client (header information, parameter, etc.)
- res
  - An object for carrying out a response to the client. In web applications, in response to information acquired from req, a process of writing to res is common. Writing to res uses the send() method.

app.js is the main file in node.js (dataDir = './public/images/';)

```
app.get('/list', function (req, res) {
    fs.readdir(dataDir, function (err, files) {
        res.json(JSON.stringify(files));
        res.end();
    });
});

app.post('/upload', function (req, res) {
    var files = fs.readdirSync(dataDir),
        max = 0;
    files.forEach(function (file) {
        max = Math.max(max, Number(file.slice(0, file.lastIndexOf('.'))));
    });
    req.pipe(fs.createWriteStream(dataDir + (++max) + '.jpeg'));
});
```

Returns a list of the uploaded. file.fs.readdir() method executes the accepted callback function with "files" parameter that is an array storing files in the directory. res.json() method sends the JSON file as a response to a client.

Extracts the number part of the uploaded files. For example, this expression extracts "10" of "10.jpeg" by slicing the file name from 0 to the position of '.'.

Pipe method connects a stream to other stream. Here we open the file stream to the new file and connects it to the uploaded file stream.

JavaScript for Client Use
The Update Function receives a list of images from the server.

```
function update() {
      $.getJSON("/list", function (data) {
          var ul =
          $('<ul data-role="listview" data-inset="true" data-theme="c" data-dividertheme="b"></ul>');
          $('#listHolder').empty();
          $('#listHolder').append(ul);
          data.forEach(function (elem) {
              ul.append("<li><a data-ajax='false' href='images/" + elem
                      + "'><img src='images/" + elem
                      + "'/><h3>"
                      + elem + "</h3></a></li>");
          });
          $('ul[data-role="listview"]').listview();
      });
   }
```

Receives a list of the uploaded files. $.getJSON method sends a request to the server and executes a callback function when the browser received a response data. $.getJSON assumes that a response data is JSON file.

Generates <ul> tag as a place holder for showing image files in a list view. We append the <ul> to the '#listHolder'

“data” is an array storing URLs of image files. data.forEach method iterates over the data array to append <li> tag for each URL. tag is annotated in a jQuery manner. So we executes $('ul[data-role="listview"]').listview(); to decorate the generated <ul> tag.

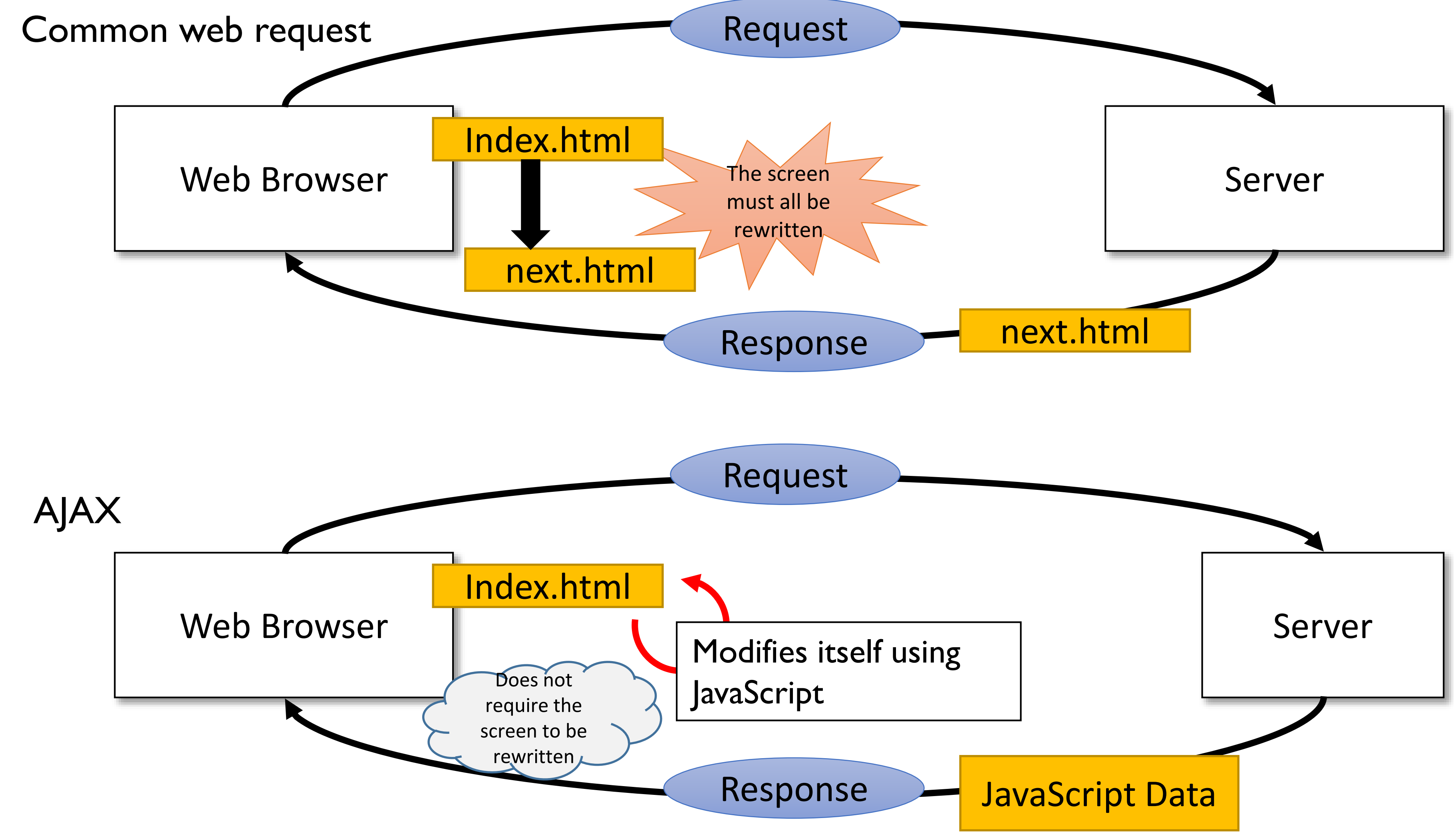
JavaScript for Client Use
Upload Process

```
$('#uploadButton').click(function (e) {
    $('input[type=file]').get(0).click();
});

$('input[type=file]').change(function (e) {
    var xhr = new XMLHttpRequest(),
            fileList = $('input[type=file]').get(0).files;
    xhr.open("POST", "/upload", true);
    xhr.setRequestHeader("Content-Type", "application/octet-stream");
    xhr.send(fileList[0]);
    update();
});
```

$('#uploadButton') is a pseudo button to bypass the click event to the input tag.
We use this trick due to the bad appearance of input tag in mobile browsers.

XMLHttpRequest (XHR) is an API used to send HTTP or HTTPS requests directly to a web server and load the server response data directly back into the script. Here the XHR sends a POST request to http://localhost:3000/upload URL. As a parameter for this request, we attach filelist[0] (the first selected file) to the XHR. To indicate that the file is a binary file, we set a request header “content-type” as “application/octet-stream”.

What is AJAX?



Design Patterns

Good Practices for Software Development

Design Pattern

- Patterns are validated solutions
  - Patterns have accumulated in the software development community, and these patterns are polished solutions, the merits/demerits of which are all investigated. By reusing this knowledge, it is possible to write high-quality software.
- Patterns have a high level of reusability
  - Patterns are written with an aim of having universal uses, in order to not be dependent on any specific application while having a high level of reusability.
- By combining patterns, they can display a high level of expressiveness
  - Patterns are a basic structure for configuring software and serve as the vocabulary for writing bigger structures (architecture).

Command Pattern

Managing various functions by encapsulating heterogeneity in an object

Command Pattern

- General purpose calculation program

```
var CalcCommand = {
    execute: function(command){
         this[command.name](command.value1, command.value2);
    },

    plus: function(value1, value2){
        return value1 + value2;
    },

    minus: function(value1, value2){
        return value1 - value2;
    }
};
```

this[command.name] selects a method, which is specified as the characters stored in the variable command.name. By attaching () to the selected results and launching the method, a function can be changed without the use of an if statement.

Command Pattern

- Enhances calculation methods and adds multiplication and division.

```
var CalcCommand = {
    execute: function(command){
        return this[command.name](command.value1, command.value2);
    },

    plus: function(value1, value2){
        return value1 + value2;
    },

    minus: function(value1, value2){
        return value1 - value2;
    },

    mult: function(value1, value2){
        return value1 * value2;
    },

    divide: function(value1, value2){
        return value1 / value2;
    }
};
```

Because this [command.name] selects a method that agrees with the characters stored in command.name, functions can be added just by adding a method.

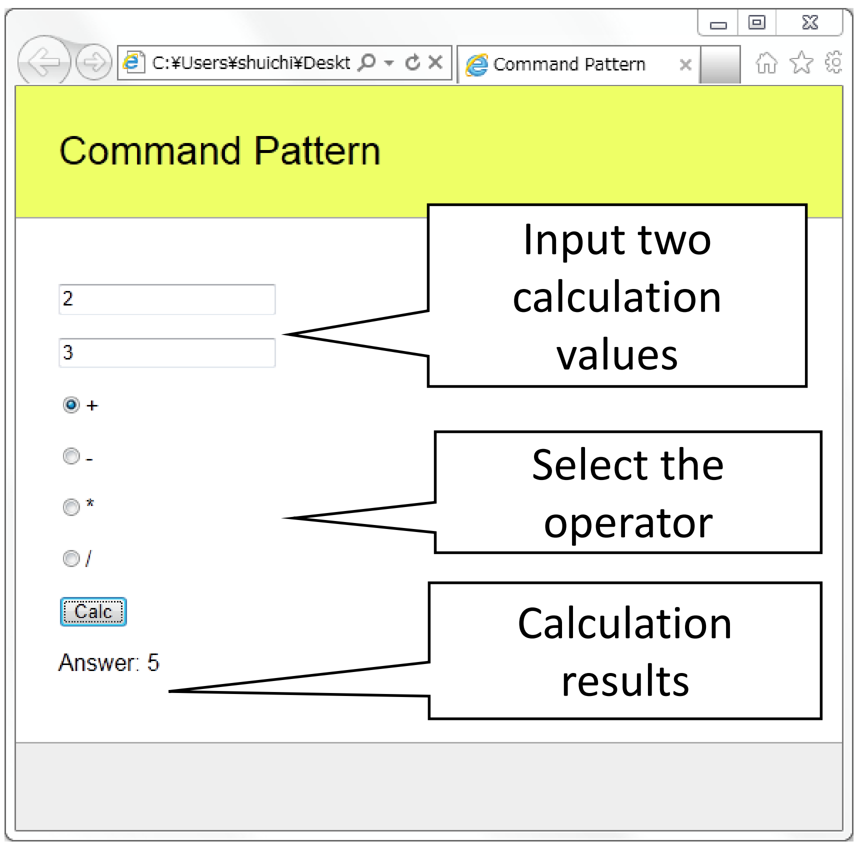
Command Pattern

http://web.sfc.keio.ac.jp/~kurabaya/lecture/javascript/command.html

Defining the calculation method in HTML elements

```
<input type="text" name="value1" value="2" />
<input type="text" name="value2" value="3" />
<input type="radio" name="mode" value="plus" />+
<input type="radio" name="mode" value="minus"  />-
<input type="radio" name="mode" value="mult"  />*
<input type="radio" name="mode" value="divide"  />/
<input type="button" id="calcButton" value="Calc" />
```

```
$("#calcButton").click(function (evt) {
        var converter = {plus: "+", minus: "-", mult: "*", divide: "/"};
        var command = Object.create(CalcCommand);
        var mode = $("input[name=mode]:checked").val();
        var v1 = parseInt($("input[name=value1]").val());
        var v2 = parseInt($("input[name=value2]").val());
        var answer = command.execute({name: mode, value1: v1, value2: v2});
        $("#answer").html(v1 + " " + converter[mode] + " " + v2 + " = " + answer);
        $.mobile.changePage('#dialog', 'pop', true, true);
    });
```



When the #calcButton is pushed, a value that is acquired from the two Texts (parseInt($("input[name=value1]").val());), and CalcCommand.execute is launched.

Method Without Using the Command Pattern

http://web.sfc.keio.ac.jp/~kurabaya/lecture/javascript/nocommand.html

- When we do not use the command pattern, we have to define many event listeners corresponding to actions
- In this case, we have to define 4 event listeners corresponding plus, minus, mult, and divide actions. If we need 100 actions, we have to define 100 event listeners and have to manage duplicated codes among those listeners

```
$("input[value=plus]").click(function () {
    var v1 = parseInt($("input[name=value1]").val());
    var v2 = parseInt($("input[name=value2]").val());
    var answer = v1 + v2;
    $("#answer").html(v1 + " + " + v2 + " = " + answer);
    $.mobile.changePage('#dialog', 'pop', true, true);
});

$("input[value=minus]").click(function () {
    var v1 = parseInt($("input[name=value1]").val());
    var v2 = parseInt($("input[name=value2]").val());
    var answer = v1 - v2;
    $("#answer").html(v1 + " - " + v2 + " = " + answer);
    $.mobile.changePage('#dialog', 'pop', true, true);
});
```

```
$("input[value=mult]").click(function () {
    var v1 = parseInt($("input[name=value1]").val());
    var v2 = parseInt($("input[name=value2]").val());
    var answer = v1 * v2;
    $("#answer").html(v1 + " * " + v2 + " = " + answer);
    $.mobile.changePage('#dialog', 'pop', true, true);
});

$("input[value=divide]").click(function () {
    var v1 = parseInt($("input[name=value1]").val());
    var v2 = parseInt($("input[name=value2]").val());
    var answer = v1 / v2;
    $("#answer").html(v1 + " / " + v2 + " = " + answer);
    $.mobile.changePage('#dialog', 'pop', true, true);
});
```

Exercise 8

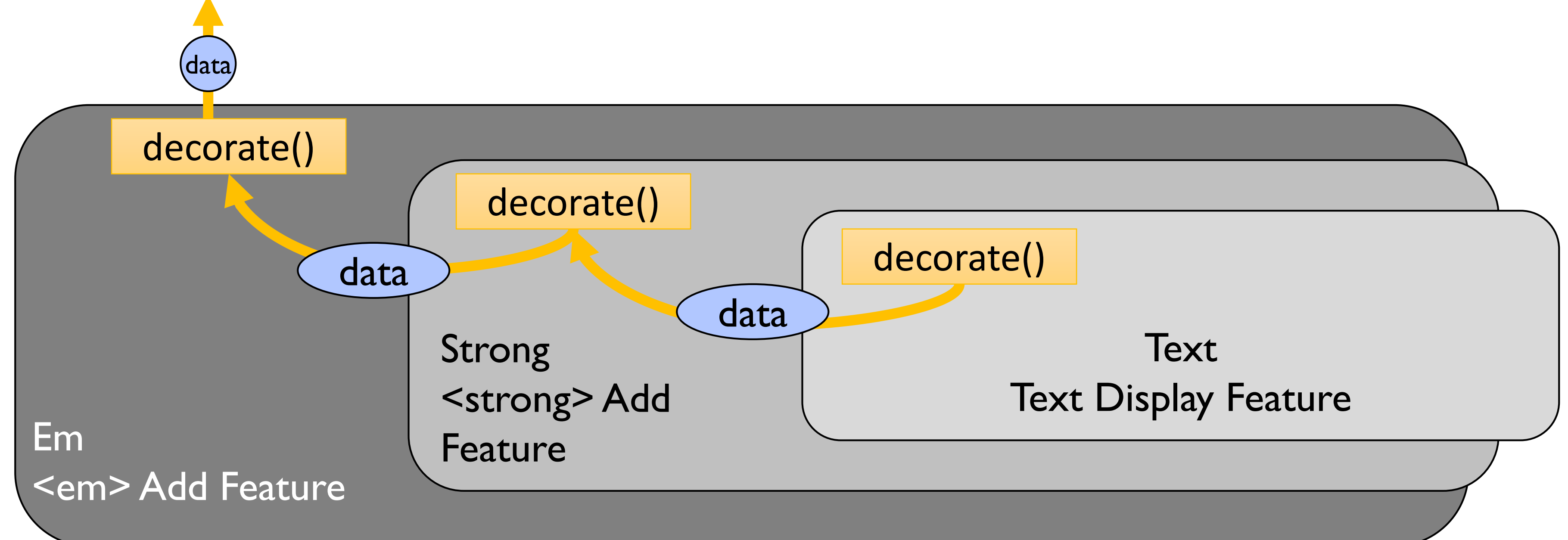
- Append logarithm, exponential, and modulus (division remainder) calculation to the example.
- You can use Math.log(A) for calculating the natural logarithm of A. Only single parameter val1 is used in this case.
- You can use Math.pow(base, exponent) for calculating base to the exponent Power, that is, $base^{exponent}$.
- You can use % operator for calculating a modulo (division remainder). For example, 7 % 2 == 1

Decorator Pattern

Incremental extension of objects

Decorator Pattern

- The principle concept of the decorator pattern is an incremental extension of object by wrapping (decorating) a simple stupid object with another object.
- Every object must have the same method and returns the same structure data
- Objects can be specialized as monofunctional, and you can combine different elements according to the aim and reusability.



Decorator Pattern

http://web.sfc.keio.ac.jp/~kurabaya/lectures/decorator.html

```
var Text = {
    init: function(t){
        this.text = t;
        return this;
    },
    decorate: function() {
        return this.text;
    }
};

var Em = Object.create(Text, {
    decorate: {
        value: function(){
            return "<em>" + this.text.decorate() + "</em>";
        }
    }
});

var Strong = Object.create(Text, {
    decorate: {
        value: function(){
            return "<strong>" + this.text.decorate() + "</strong>";
        }
    }
});
```

```
var Italic = Object.create(Text, {
    decorate: {
        value: function(){
            return "<i>" + this.text.decorate() + "</i>";
        }
    }
});

var Underline = Object.create(Text, {
    decorate: {
        value: function(){
            return "<u>" + this.text.decorate() + "</u>";
        }
    }
});

var Style = Object.create(Text, {
    css: {
        value: function(css){
            this._css = css;
            return this;
        }
    },
    decorate: {
        value: function(){
            return "<span style='" + this._css + "'>"
            + this.text.decorate() + "</span>";
        }
    }
});
```

6 Decorator Objects in the Example

Text

- Text object stores character strings. This is a base prototype object. The following objects utilizing Text as a prototype object. So they have the same and similar functionalities.

Em

- Appends “<em>” tag to the string.

Strong

- Appends “<strong>” tag to the string.

Italic

- Appends “<i>” tag to the string.

Underline

- Appends “<u>” tag to the string.

Style

- Appends “<span style>” tag to the string.

Decorator Pattern Usage

```
var decorator1 = Object.create(Text).init("Hello");
document.write(decorator1.decorate());

document.write("<br>");

var decorator2 = Object.create(Strong).init(decorator1);
document.write(decorator2.decorate());

document.write("<br>");

var decorator3 = Object.create(Em).init(decorator2);
document.write(decorator3.decorate());

document.write("<br>");

var decorator4 = Object.create(Underline).init(decorator3);
document.write(decorator4.decorate());

document.write("<br>");

var decorator5 =
Object.create(Style).init(decorator4).css("font-size:40px");
document.write(decorator5.decorate());
```

Executing the above program displays five hellos corresponding to the variables decorator1, decorator2, decorator3, decorator4, and decorator5.

decorator1

decorator2

decorator3

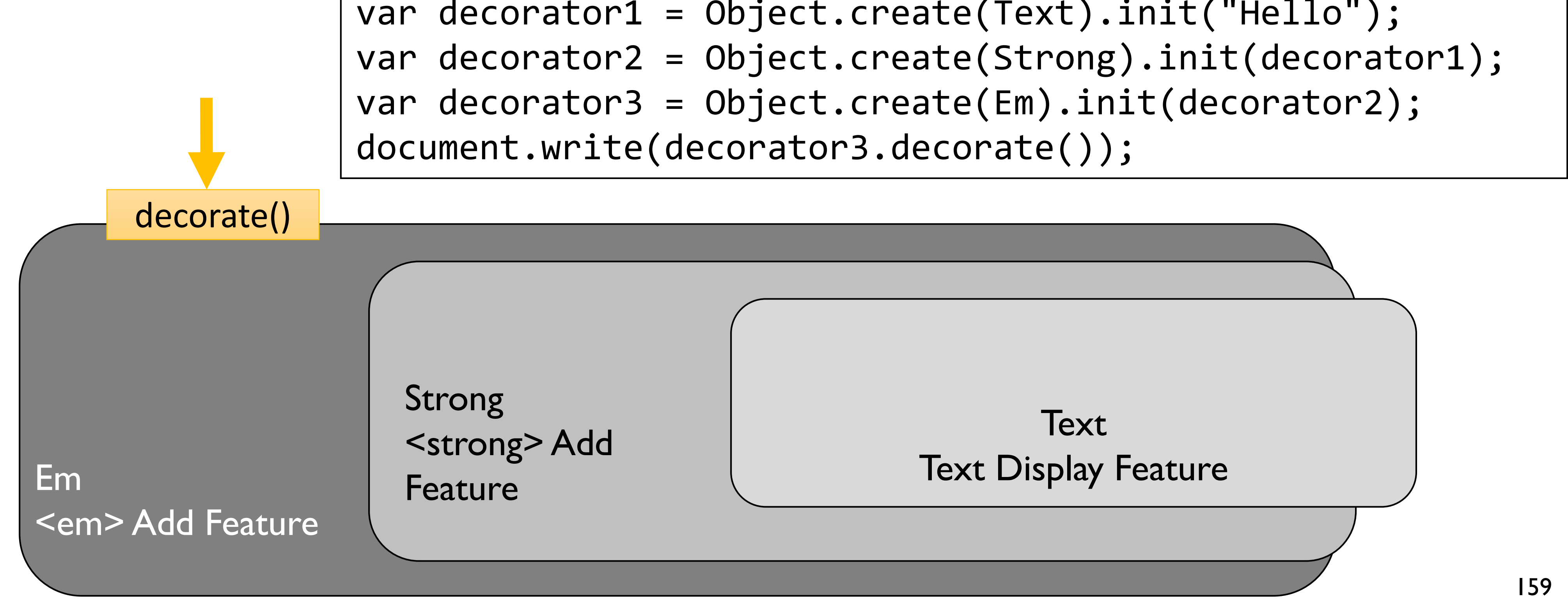
decorator4

decorator5

Decoration Process (1)

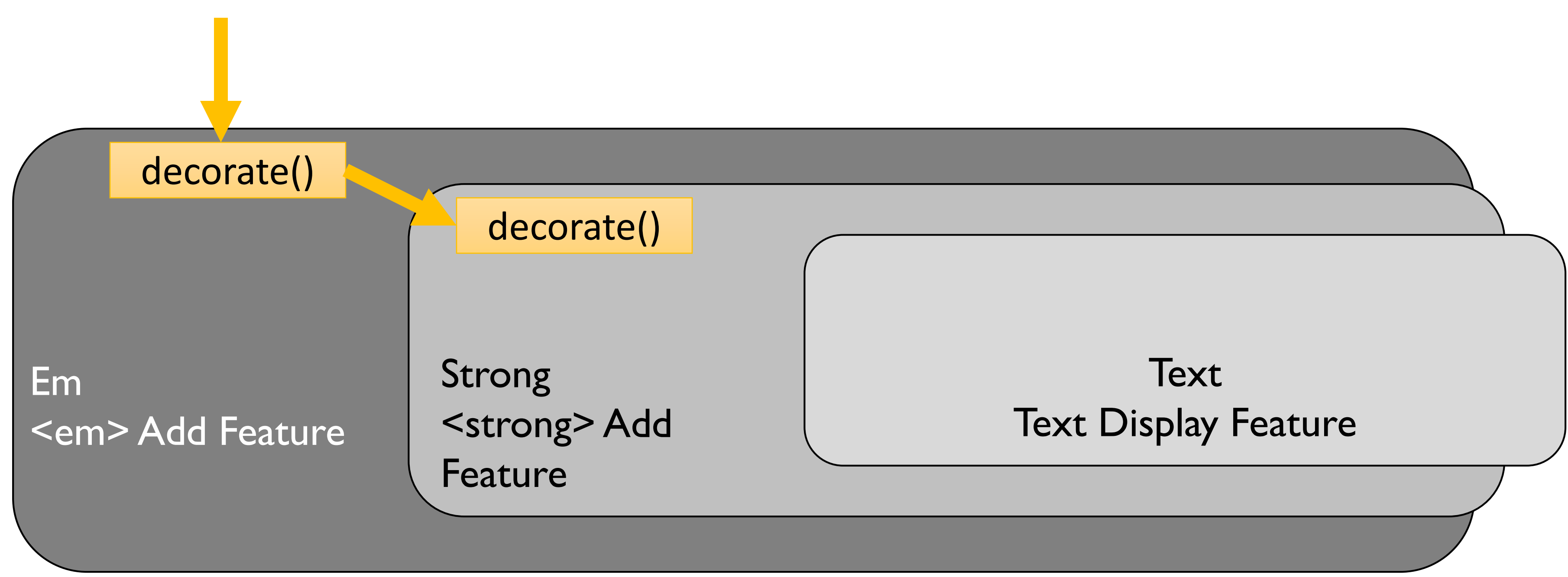
- We trace the process when executing a decorate() method.
- First, the top Em decoration is executed.

```
var decorator1 = Object.create(Text).init("Hello");
var decorator2 = Object.create(Strong).init(decorator1);
var decorator3 = Object.create(Em).init(decorator2);
document.write(decorator3.decorate());
```



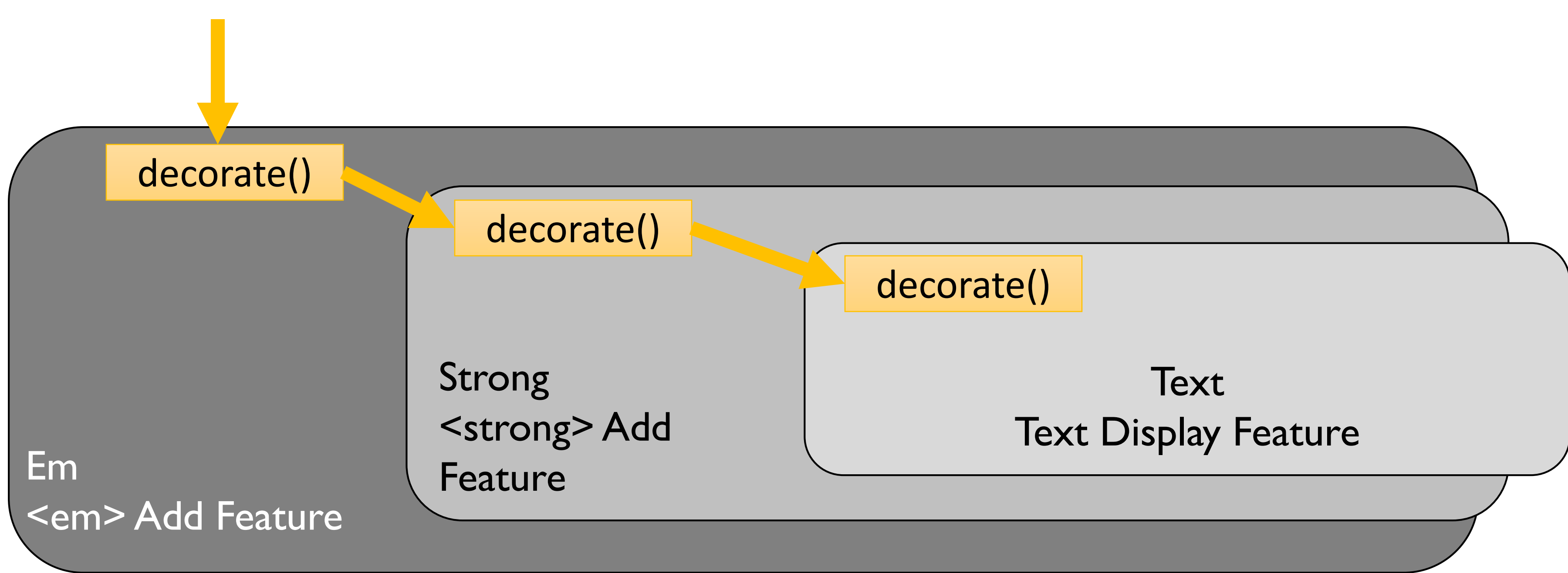
Decoration Process (2)

- The Em decorate() method executes the Strong decorate() method.



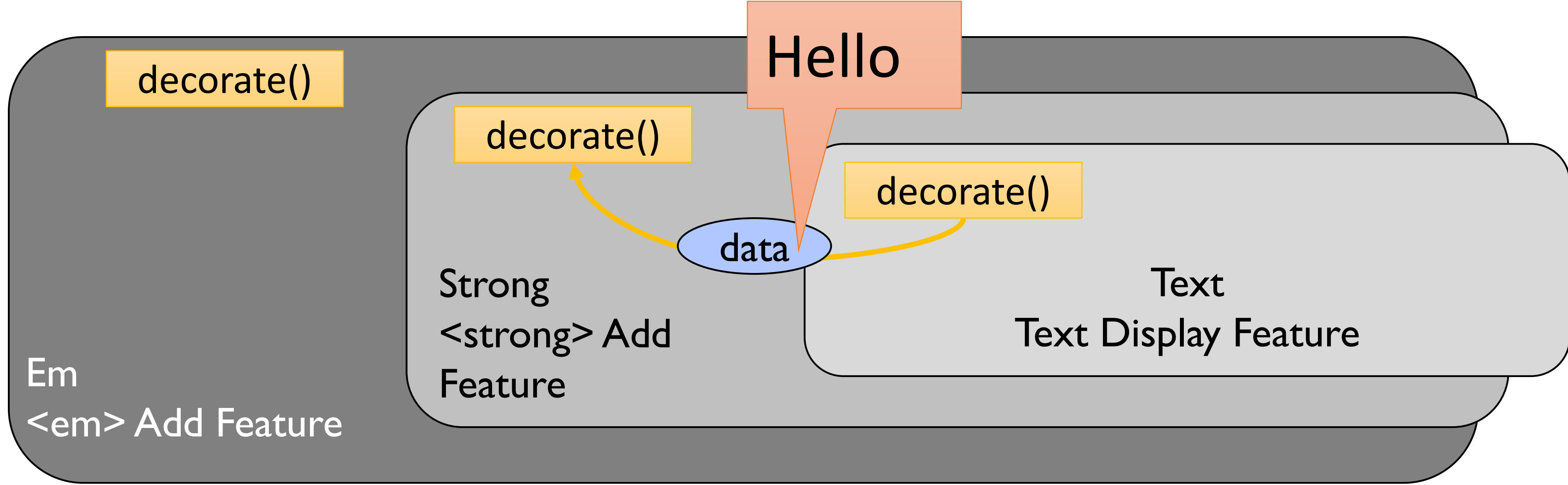
Decoration Process (3)

- The Strong decorate() method executes the Text decorate() method.



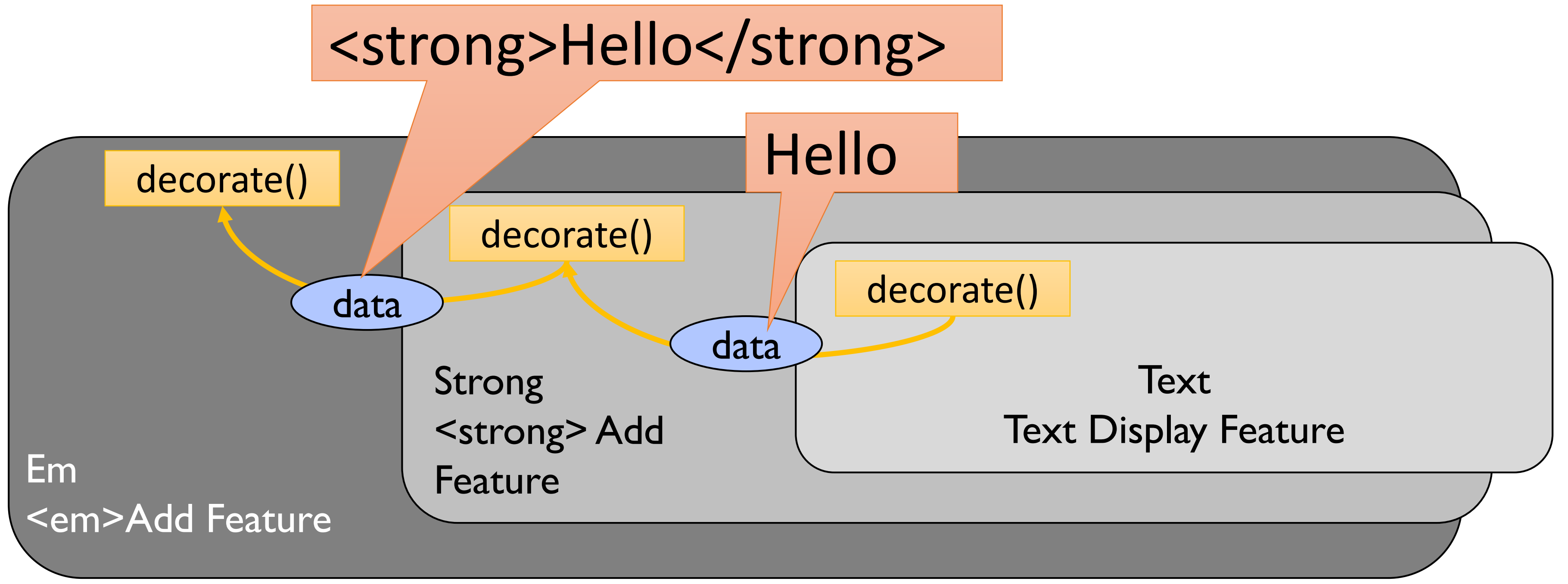
Decoration Process (4)

- The Text decorate() method returns the kept data as it is.
- As a result, the Strong decorate() bolds the retrieved data.



Decoration Process (5)

- The Strong decorate() method returns the bolded data.
- The Em highlights the bolded data returned by Strong decorate().



Decoration Process (6)

- The Em returns highlighted data.
- As a result, the process of “bolding and then highlighting text” is configured.

Hello

Hello

Hello

data

decorate()

data

decorate()

data

decorate()

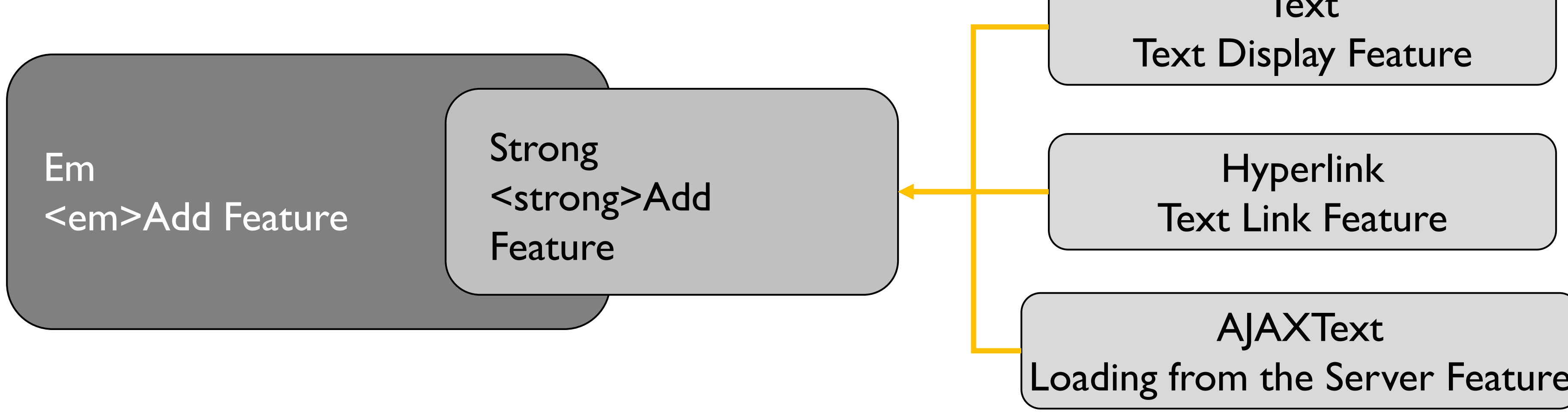
Strong
<strong> Add Feature

Text
Text Display Feature

Em
<em>Add Feature

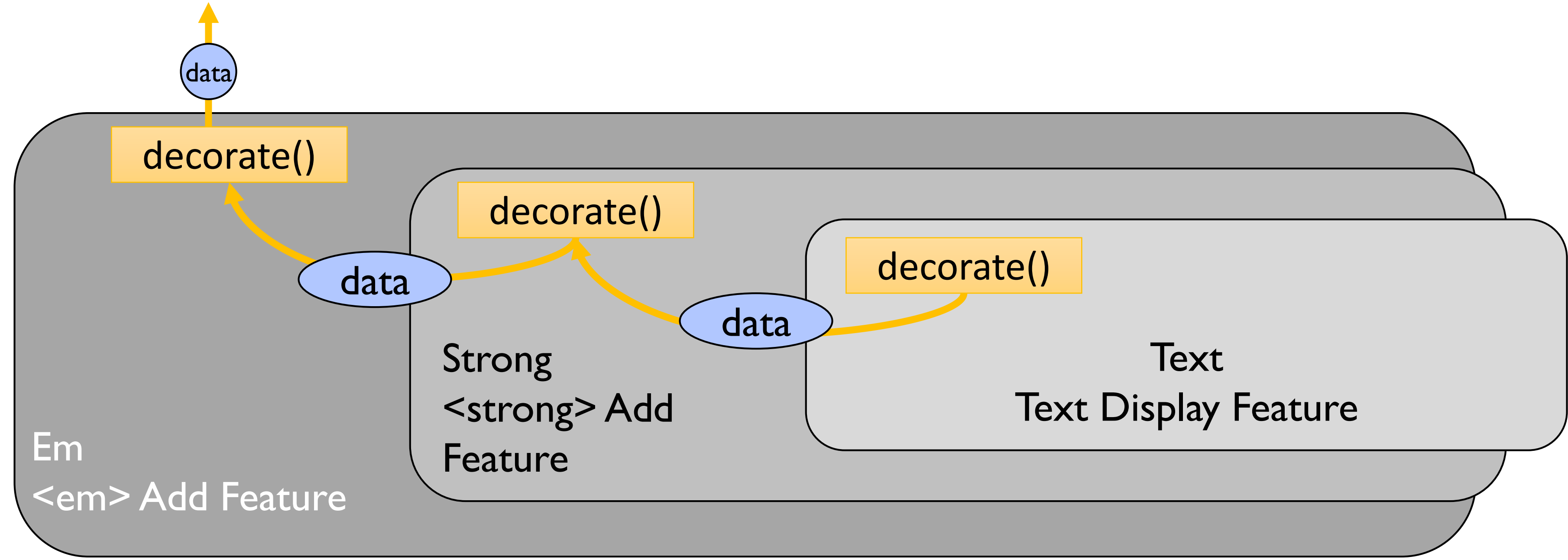
Configuring Decorators and Loosely-Coupling

- We can extend the base decorator for supporting various data sources. For example, HyperLink data, AJAX text data can be used as a data source.
- From the application that is calling Em, the category of base decorator can never be seen, so even the application is not influenced by the change.



Decoration Pattern

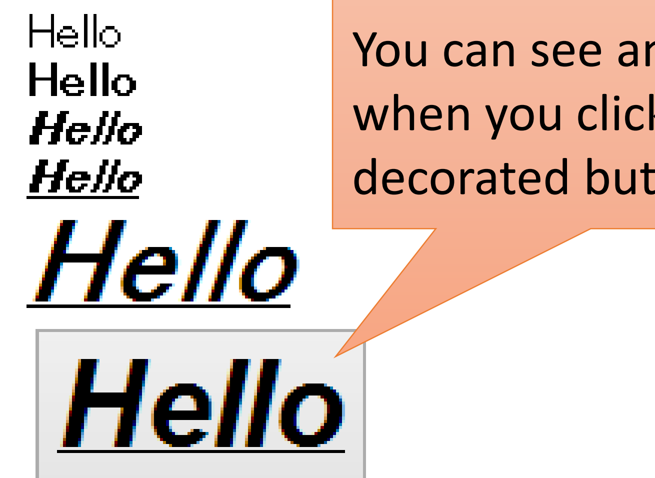
- A design that guarantees any combination by applying a constraint such as a “closure property” to the class.
- The object must have the same functions(methods) and must share the same data structure.



Exercise 9

- Append Button tag decorator for the example.
- Bind your own event listener for the appended input tag.
- For example, the following example code must be executed.

```
var decorator6 =
Object.create(Button).init(decorator5)
.click('alert("Clicked!");');

document.write(decorator6.decorate());
```



Module Pattern

Good Practices for Software Development

Module Pattern

- A pattern for concealing internal states.

```
var CounterModule = (function(){
    var counter = 0;
    var debug = function() {
        console.log("counter = " + counter);
    };
    return {
        count: function() {
            return counter;
        },
        incrementCounter: function() {
            counter += 1;
            debug();
        },
        resetCounter: function() {
            counter = 0;
            debug();
        }
    };
})();
```

In this example, the var counter value cannot be changed from an outside module. Furthermore, the debug() function has been made not callable from outside.

Technical Elements of the Module Pattern

- Executing an unnamed function

Executing an ordinary function

```
var foo = function() {
  …
};
foo();
```

Using the syntax of Function(){}, a function object is generated and the function object is assigned to the variable foo

```
var foo = function() {…};
foo();
```

() when calling functions.

Same () for when launching functions

Executing an ordinary function

```
(function() { … })();
```

(Function(){…}) encloses the function definition between (), and makes it recognized as a preferred function definition.

```
(function() { … })();
```

() when calling functions.

Even without a name it can be launched as a function.

Returning an Unnamed Object

- In JavaScript, it is possible to generate objects by using {}.
- The generated objects are not necessarily assigned to a variable, but can also pass as an argument of a function, or as a return value using return.

```
var CounterModule = (function(){
    var counter = 0;
    return {…};
})();
```

Same Meaning

```
var CounterModule = (function(){
    var counter = 0;
    var result = {…};
    return result;
})();
```

Module Pattern

- In module patterns, because it is possible to enclose a specific function or variable within a module, it becomes possible to write a program without the need to be concerned with things such as name collision.

```
//  Counter increases by 1.
CounterModule.incrementCounter();
console.log(CounterModule.count());

//  counter increases by 1.
CounterModule.incrementCounter();
console.log(CounterModule.count());
```

Package

- Packages like those on the left are defined so that name collision with other libraries in JavaScript can be prevented.
- In the example on the right, modules is defined in the package variable, and PrototypeObject is defined in the modules.

```
var package = {
    module: {}
};

package.modules.PrototypeObject = function(){};
```

Module Definition

```
package.modules.PrototypeObject = function(){

    var drawFlag = false;
    var oldX = 0;
    var oldY = 0;
    var canvas = null;
    var context = null;

    return {…

            };
}();
```

Variable defined here.

Function defined here.

Mediator Pattern

Encapsulating interaction among a set of objects

Mediator Pattern

https://github.com/ajacksified/Mediator.js

- Mediator Pattern encapsulates complex interaction between objects by providing a messaging mechanism.
- Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently *.

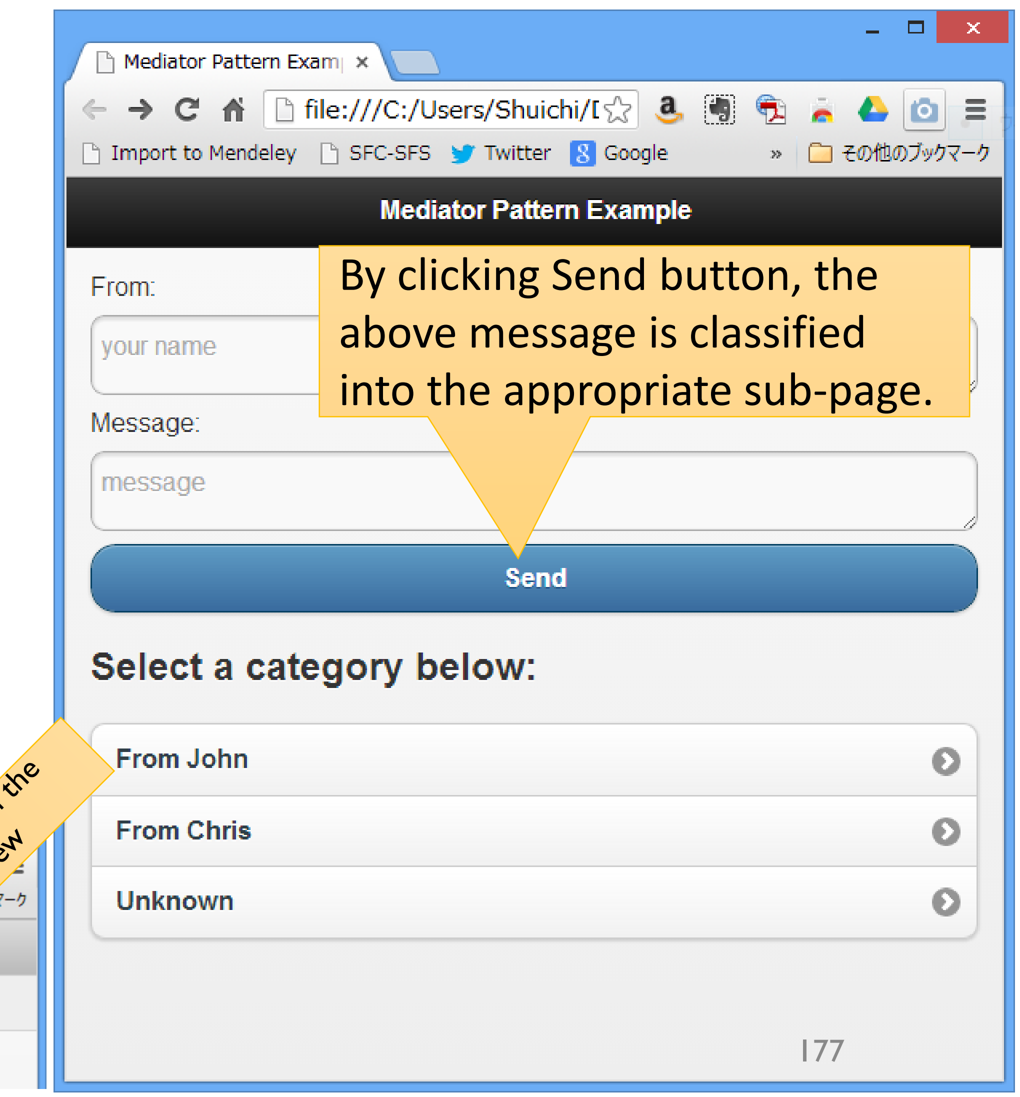
```
var mediator = new Mediator();
mediator.subscribe("message",
function(data){…});

…

mediator.publish("message", data);
```


* Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, “Design Patterns: Elements of Reusable Object-Oriented Software,” Addison-Wesley, 1994, ISBN 0-201-63361-2

Mediator Example

http://web.sfc.keio.ac.jp/~kurabaya/lectures/mediator.html

- We have implemented the message classification system that classifies a text message into three categories according to the sender information.
- The most important point is that the mediator objects decouples message input mechanism and classification mechanism.



Index.html

Loads jQuery, jQueryMobile, and CSS

Markup according to jQueryMobile Use (data-role)

```
<!DOCTYPE html>
<html>
<head>
    <title>jQuery Mobile Example</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.1/jquery.mobile-1.3.1.min.css"/>
    <script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
    <script src="http://code.jquery.com/mobile/1.3.1/jquery.mobile-1.3.1.min.js"></script>
</head>
<body>

<div data-role="page" id="main">
    <div data-role="header">
        <h1>Mediator Pattern Example</h1>
    </div>
    <div data-role="content">
        <textarea rows="1" placeholder="your name"></textarea>
        <textarea rows="3" placeholder="message"></textarea>
        <button data-theme="b">Send</button>
        <h2>Select a category below:</h2>
        <ul data-role="listview" data-inset="true">
            <li><a href="#dialog1">From John</a></li>
            <li><a href="#dialog2">From Chris</a></li>
            <li><a href="#dialog3">Unknown</a></li>
        </ul>
    </div>
</div>
```

Textarea for writing a message. The first one is for inputting sender, and the second one is for inputting message.

Category list
There are three categories: John, Chris, and Unknown.

Index.html (continued)

- We have prepared three sub-pages by utilizing jQuery mobile's pagination mechanism.
- Dialog1 shows the message list for John.
- Dialog2 shows the message list for Chris.
- Dialog3 shows the unclassified messages.

```
<div data-role="page" id="dialog1">
    <div data-role="header" data-theme="d">
        <h1>John</h1>
        <a data-role="button" data-rel="back" data-icon="back" data-theme="a">Back</a>
    </div>
    <div data-role="content">
        <ul data-role="listview"></ul>
    </div>
</div>
<div data-role="page" id="dialog2">
    <div data-role="header" data-theme="d">
        <h1>Cris</h1>
        <a data-role="button" data-rel="back" data-icon="back" data-theme="a">Back</a>
    </div>
    <div data-role="content">
        <ul data-role="listview"></ul>
    </div>
</div>
<div data-role="dialog" id="dialog3">
    <div data-role="header" data-theme="d">
        <h1>Unkown</h1>
    </div>
    <div data-role="content">
        <textarea></textarea>
        <a data-role="button" data-rel="back" data-theme="b">Back</a>
    </div>
</div>
```

Listview for showing the messages sent to John

Listview for showing the messages sent to Chris

Textarea for showing unclassified messages

Setup Mediator and Publication

- var mediator = new Mediator();
  - Mediator construction.
- document.querySelector("button").addEventListener("click", …
  - Binding a function to the click event of “Send” button.
- var data = { Message: message, From: name, Time: new Date()};
  - Creating message data by constructing object.
- mediator.publish("message", data);
  - Publishing data to the “message” channel. The subscriber of “message” channel will receive the same data.

```
var mediator = new Mediator();
document.querySelector("button").addEventListener("click", function (evt) {
    var name = document.querySelector("textarea:nth-of-type(1)").value;
    var message = document.querySelector("textarea:nth-of-type(2)").value;
    document.querySelector("textarea:nth-of-type(1)").value = "";
    document.querySelector("textarea:nth-of-type(2)").value = "";
    var data = { Message: message, From: name, Time: new Date()};
    mediator.publish("message", data);
});
```

Subscription

- Subscriber is a object that listens a “channel” in the mediator.
- Mediator object provides “subscribe” method for binding message channel and subscriber.
- mediator.subscribe(“channel", function (data) {}, {predicate:function(date){}}
  - channel: name of the message channel.
  - Function(data){} will be invoked when the subscriber receives data from the specified channel.
  - predicate:function(date){} will check the received data is appropriate for processing. This function is optional.
- In this case, three subscriber subscribes the same channel “message”, thus all the subscribers receive the same data simultaneously.
- Thus, we provided the predicate function for selecting only appropriate messages.

```
mediator.subscribe("message", function (data) {
    $("#dialog1 div ul").append("<li>" + "From " +
        data.From + ", " + data.Message + ", " + data.Time + "</li>");
    $("a[href=#dialog1]").text("From John (" +
        $("#dialog1 div ul").children().length + ")");
    $("#dialog1 div ul").listview('refresh');
}, {predicate: function (data) {
    return data.From === "John";
}});
mediator.subscribe("message", function (data) {
    $("#dialog2 div ul").append("<li>" + "From " + data.From + ", " +
data.Message + ", " + data.Time + "</li>");
    $("a[href=#dialog2]").text("From Chris (" + $("#dialog2 div
ul").children().length + ")");
    $("#dialog2 div ul").listview('refresh');
}, {predicate: function (data) {
    return data.From === "Chris";
}});
mediator.subscribe("message", function (data) {
    var textarea = $("#dialog3 div textarea");
    textarea.val(textarea.val() + "From " + data.From + ", " +
      data.Message + ", " + data.Time + "¥n");
    $("a[href=#dialog3]").text("Unknown (" +
      textarea.val().match(/¥n/g).length + ")");
}, {predicate: function (data) {
    return data.From != "Chris" && data.From != "John"; }});
```

Subscription (dialog1)

```
<div data-role="page" id="dialog1">
    <div data-role="header" data-theme="d"><h1>John</h1><a data-role="button" data-rel="back" data-icon="back" data-theme="a">Back</a></div>
    <div data-role="content">
        <ul data-role="listview"></ul>
    </div></div>
```

$("#dialog1 div ul")

- mediator.subscribe("message", function (data) {…}, {…option…});
  - subscribe method associates a channel specified as 1st argument and function.
  - When a new message comes to the channel, the corresponding function is invoked by the mediator. Option specifies how to validate and to select the appropriate message.
- $("#dialog1 div ul").append("
" + "From " + data.From + ", " + data.Message + ", " + data.Time + "
");
  - jQuery's append method creates a new DOM node and add it to the existing node $("#dialog1 div ul").
- $("#dialog1 div ul").listview('refresh');
  - Listview method refreshes the LI tag decoration.
- {predicate: function (data) {return data.From === "John";}}
  - This option selects only the method that satisfies the specific condition. In this case, the mediator delivers the message that satisfies data.From === "John" to the subscriber.

```
mediator.subscribe("message", function (data) {
    $("#dialog1 div ul").append("<li>" + "From " + data.From + ", " + data.Message + ", " + data.Time + "</li>");
    $("a[href=#dialog1]").text("From John (" + $("#dialog1 div ul").children().length + ")");
    $("#dialog1 div ul").listview('refresh');
}, {predicate: function (data) {return data.From === "John";}});
```

Subscription(dialog3)

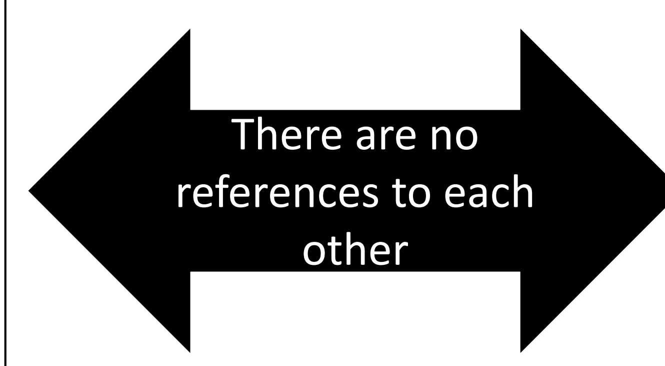
- $("a[href=#dialog3]").text("Unknown (" + textarea.val().match(/¥n/g).length + ")");
  - textarea.val().match(/¥n/g).length indicates the number of return code(¥n) in the text area.
- data.From != "Chris" && data.From != "John";
  - This predicate checks that the message is sent to neither Chris nor John.

```
mediator.subscribe("message", function (data) {
    var textarea = $("#dialog3 div textarea");
    textarea.val(textarea.val() + "From " + data.From + ", " + data.Message + ", " + data.Time + "¥n");
    $("a[href=#dialog3]").text("Unknown (" + textarea.val().match(/¥n/g).length + ")");
}, {predicate: function (data) {
    return data.From != "Chris" && data.From != "John"; }});
```

Summary

- The publisher and subscriber is completely decoupled. Thus, we can extend them without side-effects.
- Basically, publication module is coupled with data model, and subscriber module is coupled with View and GUI. As GUI and Data should be decoupled, the mediator pattern is a good solution to realizing loosely-coupling GUI and Data.

```
document.querySelector("button").addEventListener("click",
function (evt) {

    var name = document.querySelector("textarea:nth-of-
type(1)").value;

    var message = document.querySelector("textarea:nth-of-
type(2)").value;

    document.querySelector("textarea:nth-of-
type(1)").value = "";

    document.querySelector("textarea:nth-of-
type(2)").value = "";

    var data = { Message: message, From: name, Time: new
Date()};

    mediator.publish("message", data);

});
```



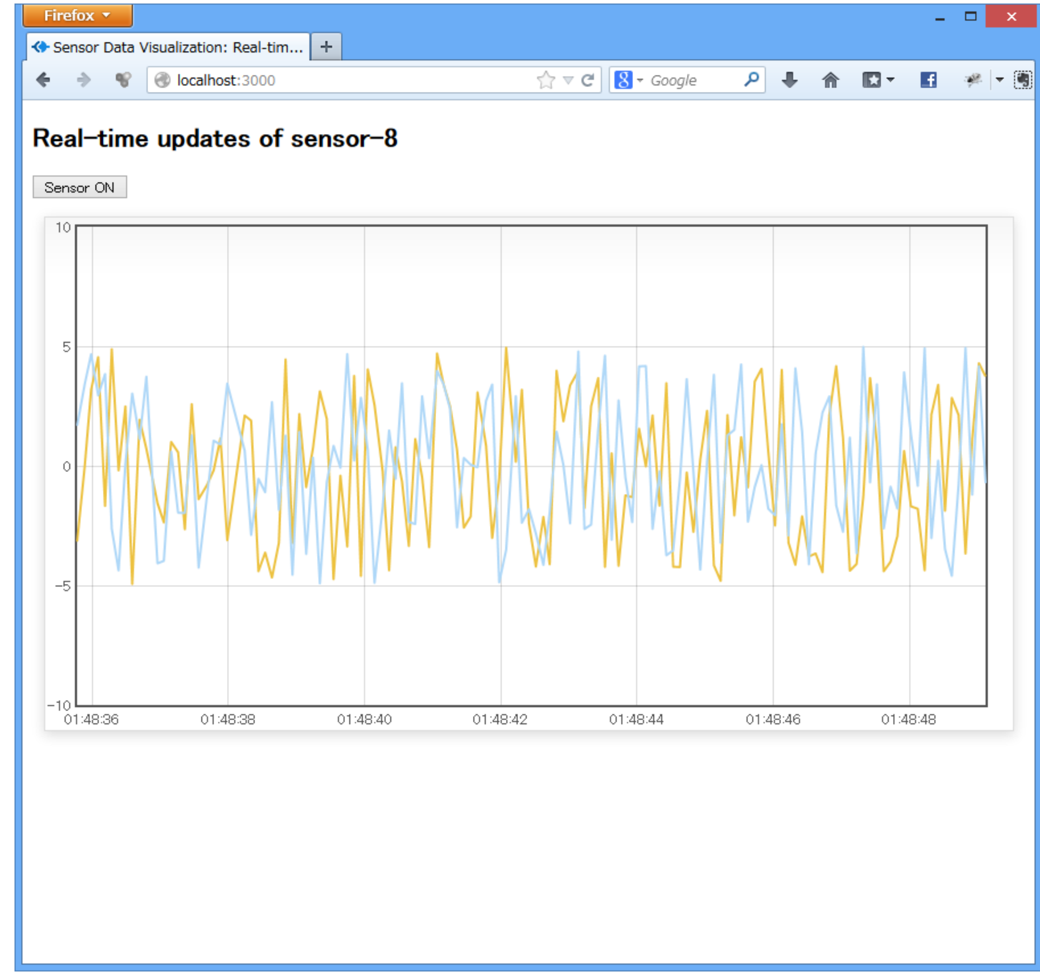

```
mediator.subscribe("message", function (data) {

    $("#dialog1 div ul").append("<li>" + "From " +
data.From + ", " + data.Message + ", " + data.Time
+ "</li>");

    $("a[href=#dialog1]").text("From John (" +
$("#dialog1 div ul").children().length + ")");

    $("#dialog1 div ul").listview('refresh');

}, {predicate: function (data) {return data.From
=== "John";}});
```

Real-Time Data Sharing by using Web Socket

Real-Time Sensor Data Sharing by using Web Socket

http://web.sfc.keio.ac.jp/~kurabaya/lectures/sensorwebsocket.zip

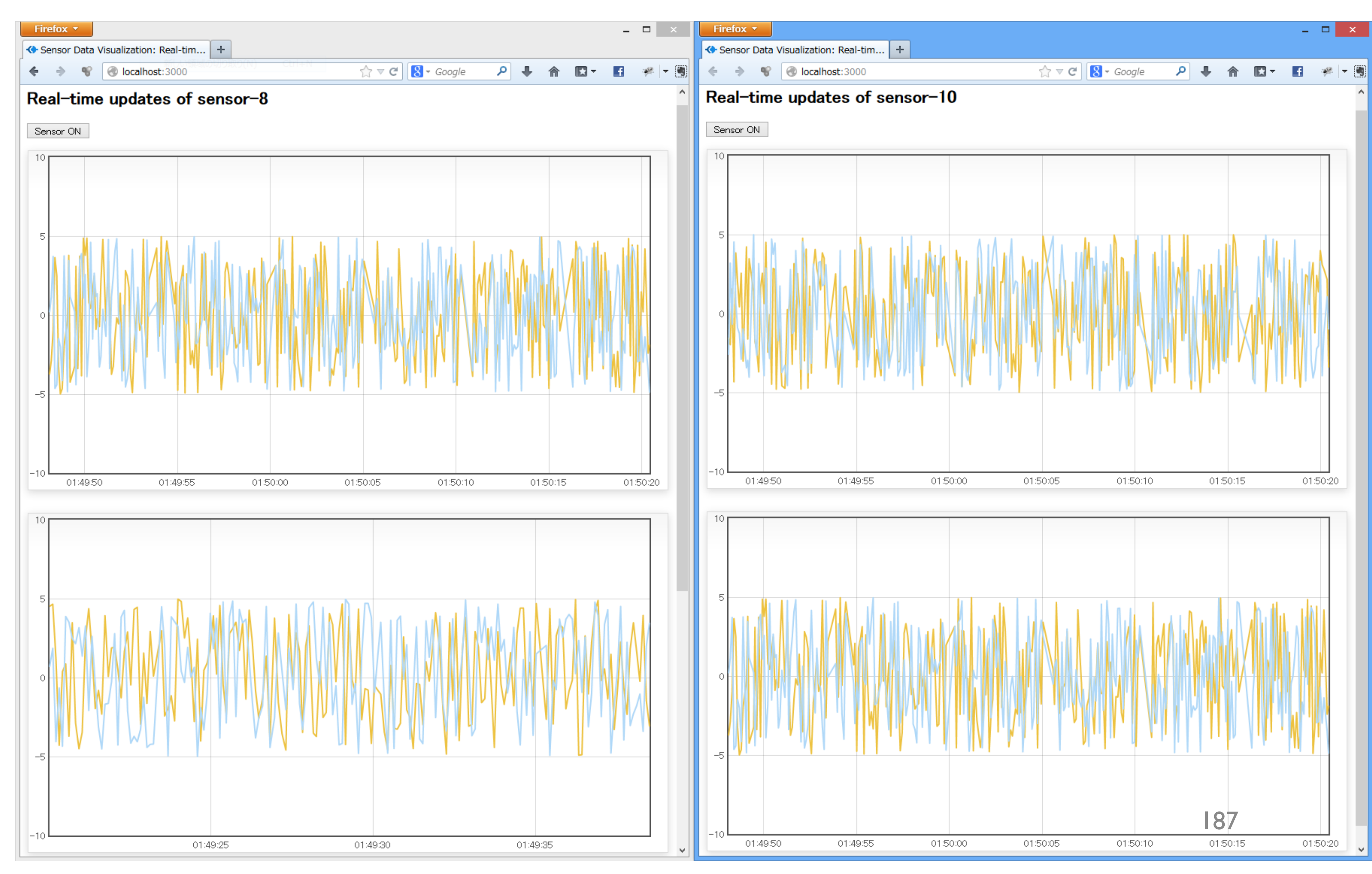
- Web Socket offers full-duplex communication channels between a web server and browsers.
  - Web Socket is standardized as RFC 6455 in December, 2011. IE 10 and after, Firefox 11 and after, Firefox Android 11 and after Chrome 16 and after, Safari 6 and after support it.
- We can send real-time sensor data from a browser to a web server and vice versa.
- The screenshot shows an example system that gathers sensor data from many clients and distributes the gathered data into multiple clients.



Real-Time Sensor Data Sharing by using Web Socket

http://web.sfc.keio.ac.jp/~kurabaya/lectures/sensorwebsocket.zip

- You can check that the example system can accept sensor data from multiple clients and can distribute the gathered sensor data for those clients, by opening multiple browsers.
- The system creates a new chart area when it accepts a new client sending sensor data.



Describing UI for Client Use:: Index.html

Loads style sheet stylesheets/style.css below the Public directory

Loads jquery framework and plugins for it.

Load main.js, which is a main module for this application. Main.js is executed here.

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href="stylesheets/style.css"/>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Sensor Data Visualization: Real-time updates</title>
    <script type="text/javascript" src="javascripts/flot/jquery.js"></script>
    <script type="text/javascript" src="javascripts/flot/jquery.flot.js"></script>
    <script type="text/javascript" src="javascripts/flot/jquery.flot.time.js"></script>
</head>
<body>
<div id="header">
    <h2>Real-time updates</h2>
</div>
<button onclick="sensorOn()">Sensor ON</button>
<div id="content">
</div>
<script type="text/javascript" src="main.js"></script>
</body>
</html>
```

Describing UI for Client Use:: main.js Global Variables

- debugOnDesktop
  - A flag indicating debug mode where sensor data is generated by using Math.random().
- myID
  - An integer indicating sensor ID in this system. Each client received a unique ID from the server when it connects to the server.
- sensorPlots
  - An array containing jQuery objects used for visualizing real-time charts.
- ws
  - Web Socket object.

```
var debugOnDesktop = true,
    myID,
    sensorsPlots = [],
    ws = new WebSocket("ws://" +
document.URL.substr(7).split('/')[0], "sensor");
```

A Function for Generating Dummy Data

- update() function generates random x and y values and sends them via Web Socket.
- It is executed periodically (executed 10 times in 1 second)
- `ws.send()` function received plain String object and send it to the web server via Web Socket protocol.

```
function update() {
    if (debugOnDesktop && myID) {
        ws.send(JSON.stringify(
            {
                sensorID: myID,
                x: Math.random() * 10 - 5,
                y: Math.random() * 10 - 5
            }
        ));
    }
    setTimeout(update, 100);
}
```

ws.onmessage (1)

- The most important event in Web Socket is “onmessage” that is triggered when a browser receives a message from a web server.
- This event listener parses the received string by invoking JSON.parse(event.data);
- This event listener checks the “command” attribute defined in the received message.
  - If command is “registered”, the listener bind the received sensorID to the global variable MyID.
  - If command is “value”, the listener visualizes it.

```
ws.onmessage = function (event) {
    var msg = JSON.parse(event.data);
    if (msg.command === "registered") {
        myID = msg.sensorID;
        $("h2").html("Real-time updates of sensor-" + msg.sensorID);
    } else if (msg.command === "value") {
        checkNewSensor(msg.sensorID);
        sensorsPlots[msg.sensorID].setData(msg.value);
        sensorsPlots[msg.sensorID].setupGrid();
        sensorsPlots[msg.sensorID].draw();
    }
};
```

ws.onmessage (2)

- checkNewSensor() checks the availability of chart area corresponding to the specified sensorID. When the chart is not available, this function creates a new chart for it.
- sensorsPlots is an array defined as a global function. This array stores a set of chart components. By invoking setDat(), setupGrid(), and draw() methods, the component updates its data.

```
ws.onmessage = function (event) {
    var msg = JSON.parse(event.data);
    if (msg.command === "registered") {
        myID = msg.sensorID;
        $("h2").html("Real-time updates of sensor-" + msg.sensorID);
    } else if (msg.command === "value") {
        checkNewSensor(msg.sensorID);
        sensorsPlots[msg.sensorID].setData(msg.value);
        sensorsPlots[msg.sensorID].setupGrid();
        sensorsPlots[msg.sensorID].draw();
    }
};
```

ws.onopen

- onopen event is triggered when the browser establishes the connection to the web server successfully. Here, we binds an event listener for devicemotion event, and the listener sends the sensed data to the server via Web Socket.
- update() function generates dummy sensor data and sends it to the server.
- ws.send() method sends String data to the server.
- JSON.stringify() method converts a JavaScript object into plain UTF-8 string.

```
ws.onopen = function (event) {
    update();
    window.addEventListener('devicemotion', function (e) {
        if (myID) {
            var data = {sensorID: myID,
                x: accelerationIncludingGravity.x,
                y: accelerationIncludingGravity.y
            };
            ws.send(JSON.stringify(data));
        }
    });
};
```

checkNewSensor(sensorID)

- checkNewSensor() checks the availability of chart area corresponding to the specified sensorID. When the chart is not available, this function creates a new chart for it.
- var placeholder = $('<div class="demo-container"><div class="demo-placeholder"></div></div>'); creates a new div tag dynamically.
- $("#content").append(placeholder); append the generated div tag to the existing div tag(#content).
- sensorsPlots[msg.sensorID] = $.plot(placeholder,... generates a new plot and stores it by associating sensorID.

```
function checkNewSensor(sensorID) {
    if (!sensorsPlots[sensorID]) {
        var placeholder =
$('<div class="demo-container"><div class="demo-
placeholder"></div></div>');
        $("#content").append(placeholder);
        sensorsPlots[msg.sensorID] =
$.plot(placeholder,
            [
                [],
                []
            ],
            {
                series: { shadowSize: 0 },
                yaxis: { min: -10, max: 10 },
                xaxis: { ticks: 5,
                         show: true,
                         mode: "time",
                         timeformat: "%H:%M:%S" }
            }
        );
    }
}
```

Server-Side Programming
App.js

- require() loads the node modules that are parameters, and returns objects.
- var server = http.createServer(app), wsserver = ws.attach(server); creates a web server and Web Socket server.

```
var express = require('express')
    , http = require('http')
    , path = require('path')
    , ws = require('websocket.io');

var app = express();
app.set('port', process.env.PORT || 3000);
app.set('view engine', 'ejs');
app.use(express.favicon());
app.use(express.logger('dev'));
app.use(express.bodyParser());
app.use(express.methodOverride());
app.use(app.router);
app.use(express.static(path.join(__dirname, 'public')));

var server = http.createServer(app),
    wsserver = ws.attach(server);
```

wsserver.on(‘connection’)

- Wsserver triggers ‘connection’ event when it accepts a connection from Web Socket client. When the connection event is triggered “socket” object, which represents a connection to the specific client,is created.
- socket.sensorID = ++id_serial; generates a unique serial ID for a client.
- database[socket.sensorID] = {data1: [], data2: []}; generates a data store place holder for the client.
- socket.send(JSON.stringify({command: "registered", sensorID: socket.sensorID})); sends the sensorID for the client.

```
wsserver.on('connection', function (socket) {
    socket.sensorID = ++id_serial;
    database[socket.sensorID] = {data1: [], data2: []};
    socket.send(JSON.stringify({command: "registered", sensorID: socket.sensorID}));
    socket.on("message", function (data) {
            var message = JSON.parse(data);
            var json = JSON.stringify(makeMessage(message.sensorID, message.x, message.y));
            wsserver.clients.forEach(function (client) {
                if (client) {
                    client.send(json);
                }
            });
        }
    );
});
```

socket.on('message')

- socket triggers 'message' event when it receives a message from Web Socket client.
- socket.sensorID = ++id_serial; generates a unique serial ID for a client.
- wsserver.clients is a special variables storing the all clients connected to the server. Thus, you can broadcast the data by iterating over this array.

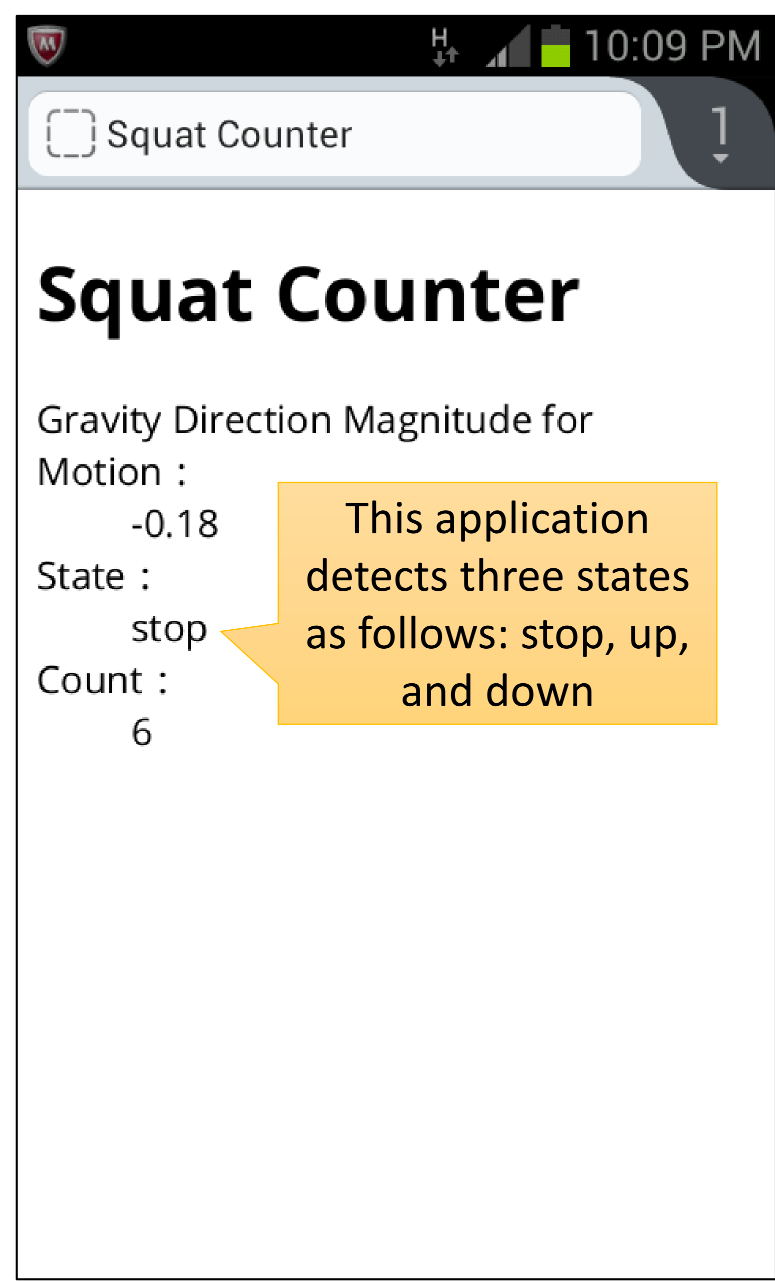
```
wsserver.on('connection', function (socket) {
    socket.sensorID = ++id_serial;
    database[socket.sensorID] = {data1: [], data2: []};
    socket.send(JSON.stringify({command: "registered", sensorID: socket.sensorID}));
    socket.on("message", function (data) {
            var message = JSON.parse(data);
            var json = JSON.stringify(makeMessage(message.sensorID, message.x, message.y));
            wsserver.clients.forEach(function (client) {
                if (client) {
                    client.send(json);
                }
            });
        }
    );
});
```

Integrating Client’s Sensor and Web Services

Squat Counter

http://web.sfc.keio.ac.jp/~kurabaya/lectures/counter.html

- This application detects your squat behavior and counts it up.
- The most important feature of this program is a noise reduction mechanism adapted for a gravity sensor equipped in smartphones.
- This application measures a magnitude of motion within a certain duration and detects the current state of a user.
- To implement the above mechanism, this application uses "setTimeout ()" timer method provided in a web browser.

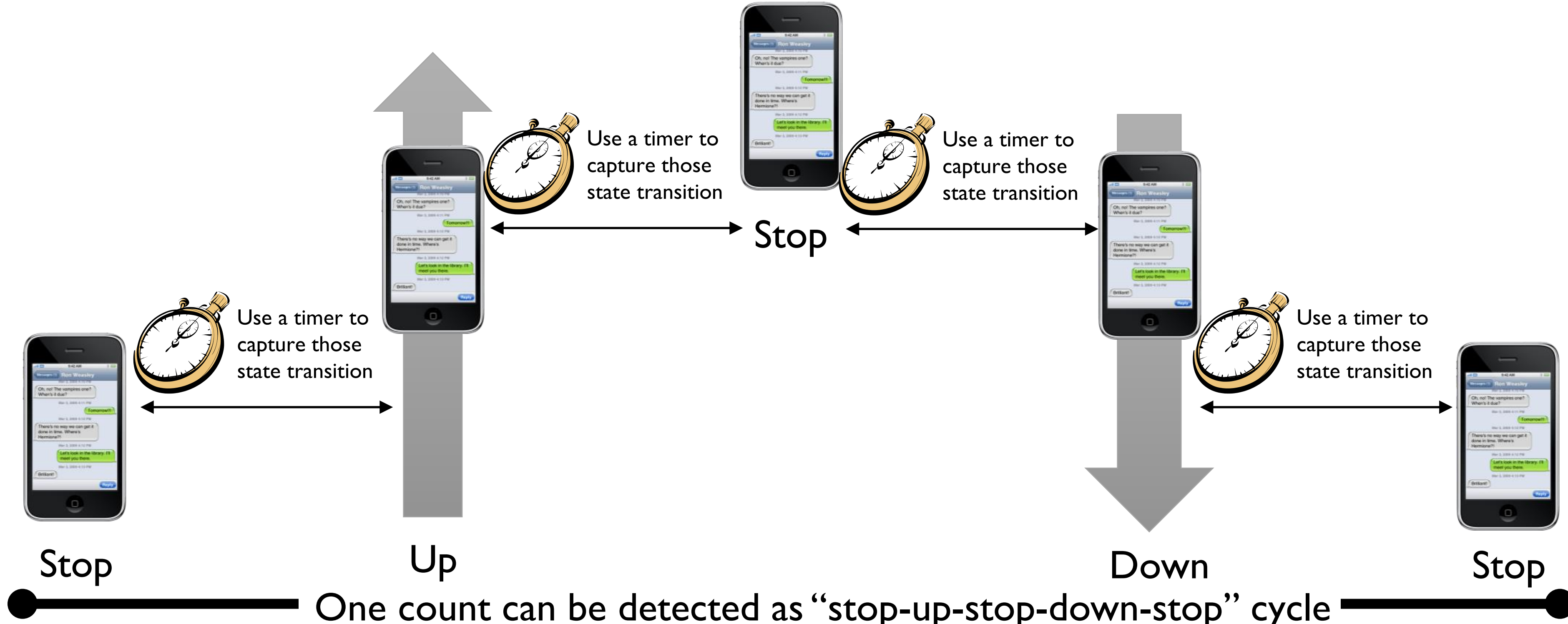


Implementation Strategy

- Use timer to measure the accumulated magnitude of device motion in a specific period.
  - setTimeout(func, delay) function calls a function or executes a code snippet after specified delay.
  - where, func is the function you want to execute after delay milliseconds. delay is the number of milliseconds (thousandths of a second) that the function call should be delayed by.
  - By invoking setTimeout() inside of a parameterized function, **we can implement a infinite loop that does not blocks the main UI thread**.
- Manage the state of device by using global variables

State Transition of Squat

- There are three states
  - Stop – the device is keeping its position.
  - Up – the device is moving up.
  - Down – the device is moving down.
- One squat count can be detected as “stop-up-stop-down-stop” cycle.



Squat Counter HTML

- There are three placeholder for values.
  - Gravity Direction Magnitude for Motion
  - State
  - Count
- <script type="text/javascript" src="counter.js"></script> loads our counter JavaScript program after initializing a web page.

```
<!DOCTYPE html>
<html>
<meta charset="UTF-8" lang="en"/>
<meta name=viewport content="width=device-width,
user-scalable=no,initial-scale=1.0,maximum-scale=1.0"/>
<meta name="apple-mobile-web-app-capable" content="yes"/>
<head>
    <title>Squat Counter</title>
</head>
<body>
<h1>Squat Counter</h1>

<dl>
    <dt>Gravity Direction Magnitude for Motion:</dt>
    <dd></dd>
    <dt>State:</dt>
    <dd></dd>
    <dt>Count:</dt>
    <dd></dd>
</dl>
<script type="text/javascript" src="counter.js"></script>
</script>
</body>
</html>
```

Global Variables

```
var dd = document.querySelectorAll("dd"),
    state = "stop",
    previous = "stop",
    log = [],
    count = 0;
```

- There are four global variables to manage current state of a user.
- dd
  - An array that stores the return value of document.querySelectorAll("dd"). So, this array stores placeholder DOM elements.
- state
  - Holds a string representing current state of a user. This variable should store “stop”, “up”, or “down”.
- previous
  - Holds a string representing a previous state of a user. This variable should store “stop”, “up”, or “down”. We needs this information to associate “down” state and the corresponding “up” state.
- Log
  - Log is an array that records magnitudes of device motion in a certain period. We measure the accumulated magnitude of device motion recorded in this array. This log variable is cleared every time when a new timer is setup.
- count
  - Holds an integer value representing a count of squat.

setTimeout()

- This program measure the accumulated magnitude of device motion recorded in this array. The accumulated value is calculated in every 100 milliseconds. This program has two branches three as follows:
- if (state != "stop" && Math.abs(sum) < 3.0)
  - When a user is moving and magnitude of motion is smaller than a threshold (3.0), the system considers that a user begins motion.
- else if (state == "stop" && sum >= 3.0)
  - When a user is stopping and magnitude of motion is greater than a threshold (3.0), the system considers that a user finishes motion.
- setTimeout(timer, 100)
  - By invoking setTimeout() inside of a parameterized function, we can implement a infinite loop that does not blocks the main UI thread.

```
function timer() {
    var sum = 0;
    log.forEach(function (v) {
        sum += v;
    });
    log = [];
    if (state != "stop" && Math.abs(sum) < 3.0) {
        state = "stop";
    } else if (state == "stop" && sum >= 3.0) {
        if (previous == "up") {
            state = "down";
            previous = state;
            count++;
        } else {
            state = "up";
            previous = state;
        }
    }
    dd[1].innerHTML = state;
    dd[2].innerHTML = count;
    setTimeout(timer, 100);
}

setTimeout(timer, 100);
```

Sensing From Web

JavaScript Programming for Sensing User's Context from Web Applications

Overview

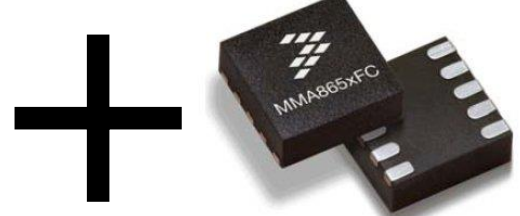
- Exploiting rich devices equipped in smartphones to capture environmental data.
- This week, we learn methods to access 6-axes sensors, optical sensor(camera) and web-based image processing techniques.
- Also we learn methods to access GPS.

Sensor Programming on the Web

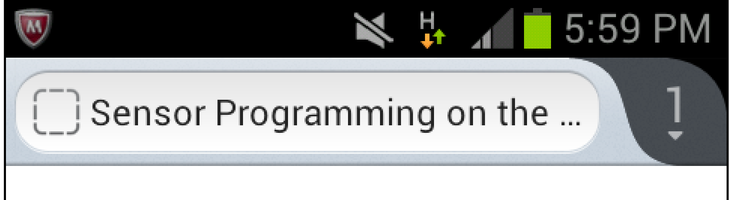
http://goo.gl/qs8YF

- Modern smartphone equips various sensors including 6-axes sensors, a barometer, a magnetometer GPS, and optical sensors (cameras).
- HTML5 offers APIs for accessing those sensors from web applications.
- By integrating web technologies and sensors, we can make social sensors that people sense environmental data by using their own smartphones.

+



Sensing Environmental Data



Sensor Programming on the web

Acceleration:
x=-0.12
y=-0.15
z=0.25

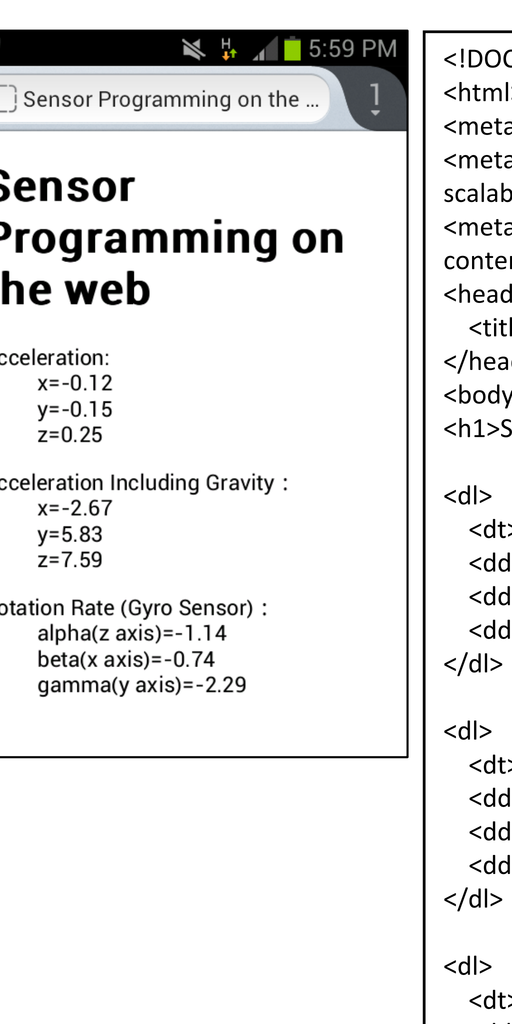
Acceleration Including Gravity :
x=-2.67
y=5.83
z=7.59

Rotation Rate (Gyro Sensor) :
alpha(z axis)=-1.14
beta(x axis)=-0.74
gamma(y axis)=-2.29

Sensor Programming on the web

- When a smartphone moves, “devicemotion” event is triggered.
- The querySelector method finds appropriate nodes sasifying CSS selector expression

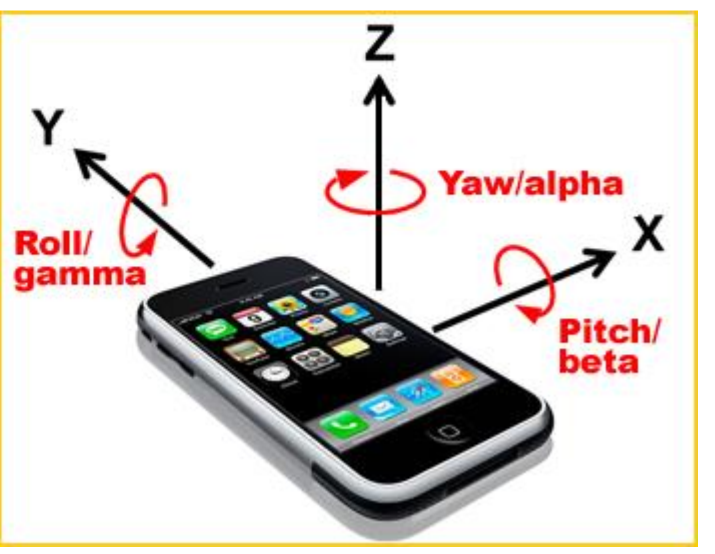
```
var acc_dd = document.querySelectorAll("dl:nth-of-type(1) dd");
var gra_dd = document.querySelectorAll("dl:nth-of-type(2) dd");
var rat_dd = document.querySelectorAll("dl:nth-of-type(3) dd");

window.addEventListener('devicemotion', function (e) {
    acc_dd[0].innerHTML = 'x:' + e.acceleration.x.toFixed(2);
    acc_dd[1].innerHTML = 'y:' + e.acceleration.y.toFixed(2);
    acc_dd[2].innerHTML = 'z:' + e.acceleration.z.toFixed(2);

    gra_dd[0].innerHTML = 'x:' + e.accelerationIncludingGravity.x.toFixed(2);
    gra_dd[1].innerHTML = 'y:' + e.accelerationIncludingGravity.y.toFixed(2);
    gra_dd[2].innerHTML = 'z:' + e.accelerationIncludingGravity.z.toFixed(2);

    rat_dd[0].innerHTML = 'a:' + e.rotationRate.alpha.toFixed(2);
    rat_dd[1].innerHTML = 'g:' + e.rotationRate.beta.toFixed(2);
    rat_dd[2].innerHTML = 'g:' + e.rotationRate.gamma.toFixed(2);
});
```




```
<!DOCTYPE html>
<html>
<meta charset="UTF-8" lang="en"/>
<meta name=viewport content="width=device-width,user-scalable=no,initial-scale=1.0,maximum-scale=1.0"/>
<meta name="apple-mobile-web-app-capable" content="yes"/>
<head>
   <title>Sensor Programming on the web</title>
</head>
<body>
<h1>Sensor Programming on the web</h1>

<dl>
   <dt>Acceleration:</dt>
   <dd>x:</dd>
   <dd>y:</dd>
   <dd>z:</dd>
</dl>

<dl>
   <dt>Acceleration Including Gravity : </dt>
   <dd>x:</dd>
   <dd>y:</dd>
   <dd>z:</dd>
</dl>

<dl>
   <dt>Rotation Rate (Gyro Sensor) : </dt>
   <dd>a:</dd>
   <dd>b:</dd>
   <dd>g:</dd>
</dl>

<script type="text/javascript">
</script>
</body>
</html>
```

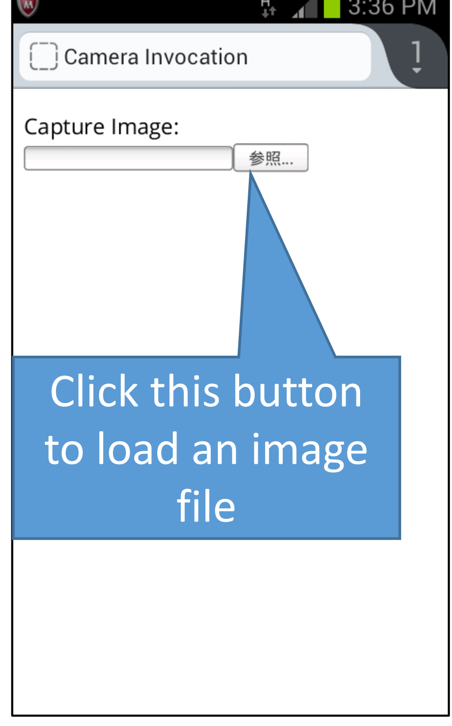
X,Y,Z,yaw, pitch, roll(alpha, beta, gamma)

- The orientation sensor establishes which way up the device is being held.
- The rotation around the X, Y, Z axes may respectively be referred to as roll, pitch and yaw as in planes and boats or in terms of degrees of beta, gamma and alpha rotation.



HTML Media Capture

- The HTML Media Capture specification defines an HTML form extension that facilitates user access to a device's media capture mechanism, such as a camera, or microphone, from within a file upload control.

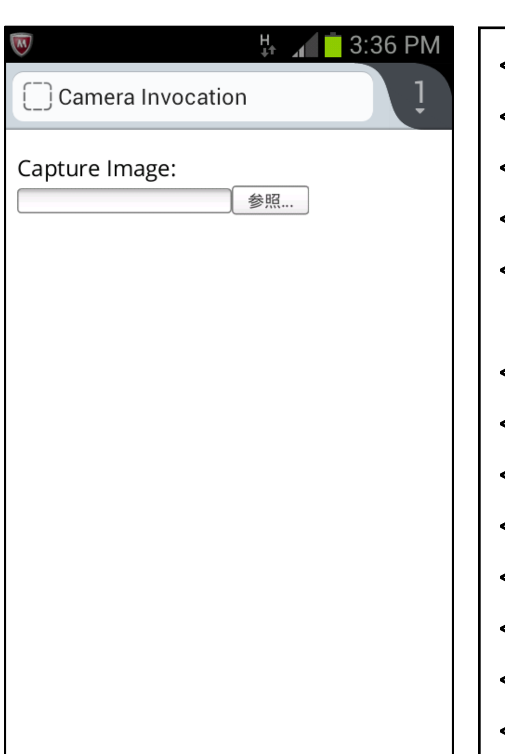


The browser load the specified image and renders it on a canvas.

HTML Media Capture

http://web.sfc.keio.ac.jp/~kurabaya/lectures/camera1.html

- <input> tag supports “capture” attribute for defining a method to get media files. You can use the following three parameter:
- “camera” for getting a still image
- “camcorder” for getting a video
- “microphone” for getting audio stream




```
<!DOCTYPE html>
<html>
<meta charset="UTF-8" lang="en"/>
<meta name=viewport content="width=device-width,user-scalable=no,initial-scale=1.0,maximum-scale=1.0"/>
<head>
  <title>Camera Invocation 1</title>
</head>
<body>
<p>Capture Image: <input type="file" id="capture1" accept="image/*" capture="camera" ></p>
<canvas id="canvas1"></canvas>
<script type="text/javascript">
</script>
</body>
</html>
```

File Access from HTML

- FileReader is an object to access a file selected in <input type="file"> tag. FileReader can access only the selected file due to security issue.
- reader.addEventListener('load', function (evt2) {…});
  - FileReader triggers 'load' event when it finishes loading a file. Variable evt2.target.result contains the loaded binary data in Data URL schema.
- reader.readAsDataURL(evt.target.files[0]);
  - readAsDataURL method starts to load the specified file.
  - Here, evt1.target corresponds to the <input type="file" id="capture1">. Thus, evt1.target.files[0] means the first selected file in the <input> tag.

- document.getElementById("capture1").addEventListener("change", function (evt){…});
- document.getElementById retrieves the DOM element specified by ID.
- <input type="file"> triggers "change" event when a user selects files.

```
document.getElementById("capture1").
addEventListener("change", function (evt1) {
  var reader = new FileReader();
  reader.addEventListener('load', function (evt2) {
    var img = new Image();
    img.addEventListener('load', function () {
      var canvas = createThumbnail(img, 400, 400);
    });
    img.src = evt2.target.result;
  });
  reader.readAsDataURL(evt1.target.files[0]);
});
```

Image Decoder in JavaScript

- Image is an object to decode and to render an image file such as JPEG and PNG.
- img.addEventListener('load', function () {…});
  - Image triggers 'load' event when it finishes decoding a image file.
- img.src = evt.target.result;
  - Image object starts to decode an image file when is was assigned "src" attribute. Here evt2.target corresponds to the loaded file (triggered by FileReader)

```
document.getElementById("capture1").
addEventListener("change", function (evt1) {
  var reader = new FileReader();
  reader.addEventListener('load', function (evt2) {
    var img = new Image();
    img.addEventListener('load', function () {
      var canvas = createThumbnail(img, 400, 400);
    });
    img.src = evt2.target.result;
  });
  reader.readAsDataURL(evt1.target.files[0]);
});
```

createThumbnail()

- Context is an object providing many methods to manipulate canvas. Context is similar to the concept of “pen” for the canvas. It is used for drawing lines, rectanbles, circles and images on the canvas.
- canvas.getContext('2d');
  - This generates the initial context for the canvas. 2d means a 2-dimensional. Only 2d is supported currently.
- We can change the size of canvas dynamically.
  - canvas.width = width;
  - canvas.height = height;
- context.drawImage(image, 0, 0, image.width, image.height, 0, 0, canvas.width, canvas.height);
  - This method draws an image object on the canvas.
- By drawing an image on the size-reduced canvas, we can generate a thunmbnail image.

```
function createThumbnail(image, maxWidth, maxHeight) {
    var width = image.width,
        height = image.height,
        canvas, context;

    maxWidth = maxWidth || 0;
    maxHeight = maxHeight || 0;
    maxWidth = maxWidth > image.width ? image.width : maxWidth;
    maxHeight = maxHeight > image.height ? image.height : maxHeight;

    if (width > maxWidth) {
      height *= maxWidth / width;
      width = maxWidth;
    }
    if (height > maxHeight) {
      width *= maxHeight / height;
      height = maxHeight;
    }

    canvas = document.getElementById('canvas1');
    context = canvas.getContext('2d');
    canvas.width = width;
    canvas.height = height;
    context.drawImage(image, 0, 0, image.width, image.height, 0, 0,
canvas.width, canvas.height);
    return canvas;
  }
```

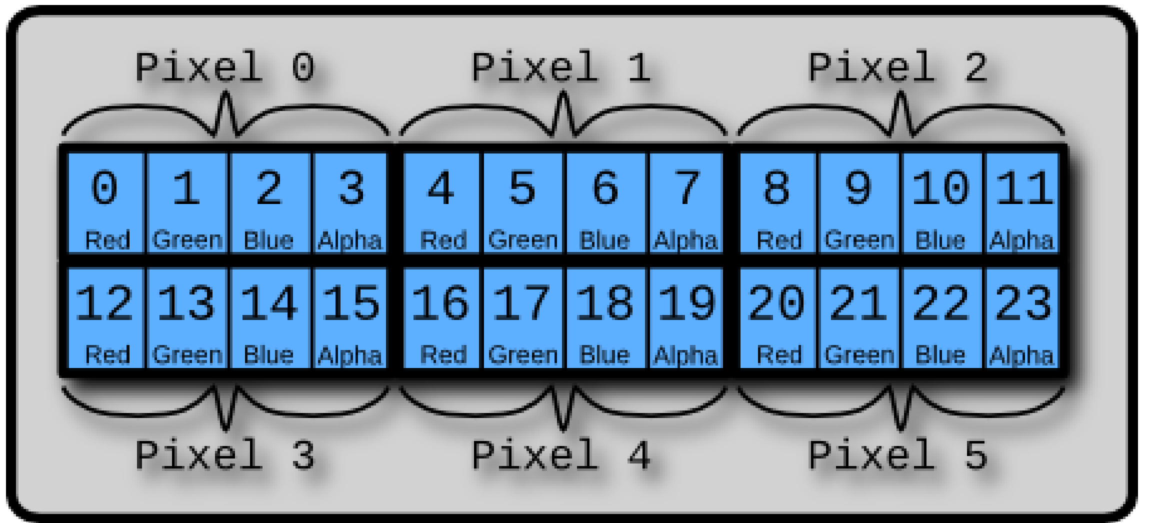
Color Histogram Analysis

http://web.sfc.keio.ac.jp/~kurabaya/lectures/camera2.html

- This program calculates the frequency of R, G, and B color component in the image.
- HTML canvas provides a pixel-level access method for the image content.
- context.getImageData() method returns a bitmap array containing each pixel in the canvas.

Pixel-Level Access to Canvas

- The pixels in the array are arranged in row-major order, and are represented as integers between 0 and 255, where each four-integer group represents the four color channels of one pixel: red, green, blue, and alpha (RGBA)
- Bitmap is represented as a long one-dimensional array consisting of width × height × 4(r,g,b,a) elements.



Pixel-Level Access to Canvas

- context.getImageData(0, 0, canvas.width, canvas.height) extracts bitmap array data from the canvas.
- pixels = bitmap.data.length / 4
  - This calculates the number of pixels in the bitmap.data. bitmap.data.length is devided by 4 because each pixel consists of r,g,b,a integers.
- r = bitmap.data[i * 4];
- g = bitmap.data[i * 4 + 1];
- b = bitmap.data[i * 4 + 2];
- a = bitmap.data[i * 4 + 3];

Pixel 0

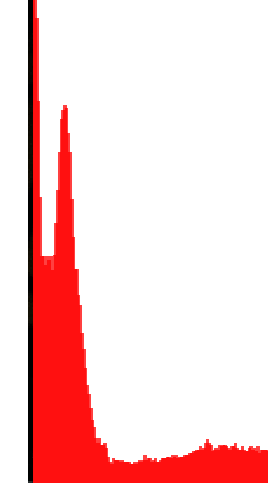
0	1	2	3
Red	Green	Blue	Alpha

```
bitmap = context.getImageData(0, 0, canvas.width,
                              canvas.height),
        pixels = bitmap.data.length / 4,
        i, r, g, b, a,
        r_array = [], g_array = [], b_array = [];

graphContext.clearRect(0, 0, graphCanvas.width,
                       graphCanvas.height)

for (i = 0; i < 255; i++) {
    r_array[i] = 0;
    g_array[i] = 0;
    b_array[i] = 0;
}

for (i = 0; i < pixels; i++) {
    r = bitmap.data[i * 4];
    g = bitmap.data[i * 4 + 1];
    b = bitmap.data[i * 4 + 2];
    a = bitmap.data[i * 4 + 3];

    r_array[r] += 1;
    g_array[g] += 1;
    b_array[b] += 1;
}
```

Drawing Histogram

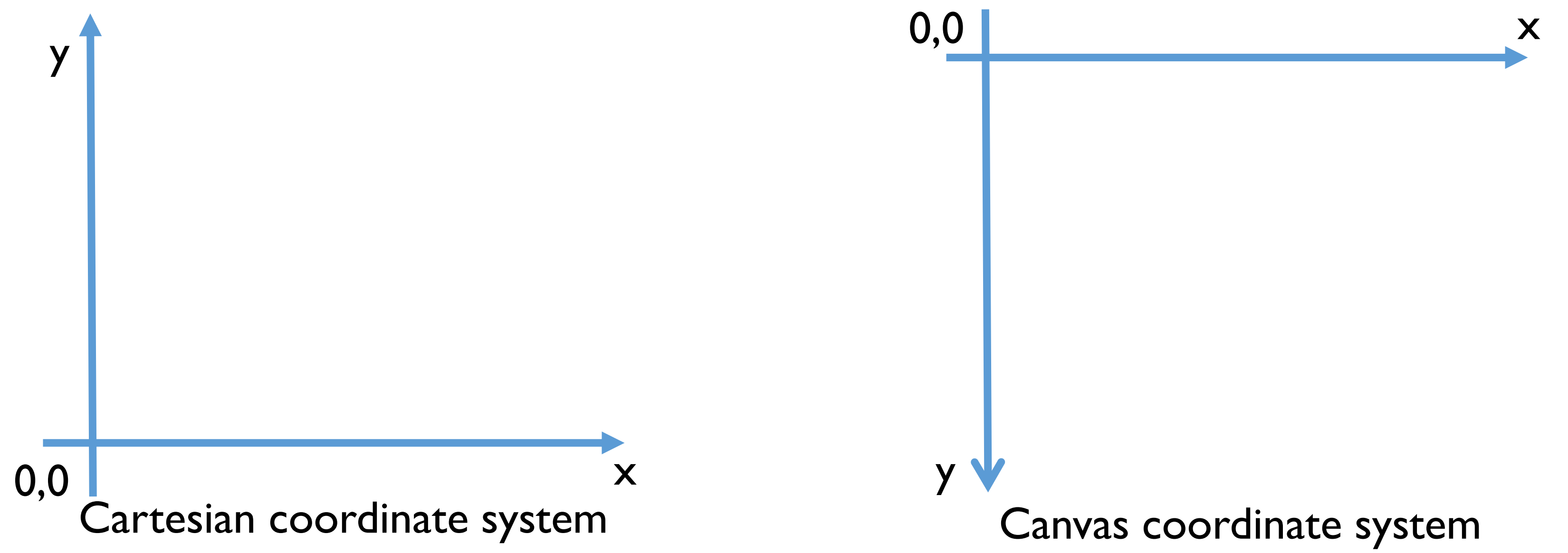
- Context object provides methods and attributes for drawing something on the canvas.
- strokStyle specifies a color for the context.
- beginPath() method starts defining path information.
- lineTo() method defines a line path. moveTo() does not write a line but moves a pen.
- closePath() method ends path definition. Stoke() method draws path actually.

```
for (i = 0; i < 255; i++) {
    graphContext.strokeStyle = 'red';
    graphContext.beginPath();
    graphContext.moveTo(i, graphCanvas.height –
           Math.round(r_array[i] * graphCanvas.height * 30));
    graphContext.lineTo(i, graphCanvas.height);
    graphContext.closePath();
    graphContext.stroke();
}
```



Canvas Coordinate System

- Canvas Coordinate System's origin (0,0) is at the top left corner.
- Everything are counted from top left. For example, if you want to move a line to right, you will have to increase your x-axis value; if you want to moving down, you will have to increase your y-axis value.
- Such coordinate system is suitable to web page rendering method because web pages' origin is at the top left corner.



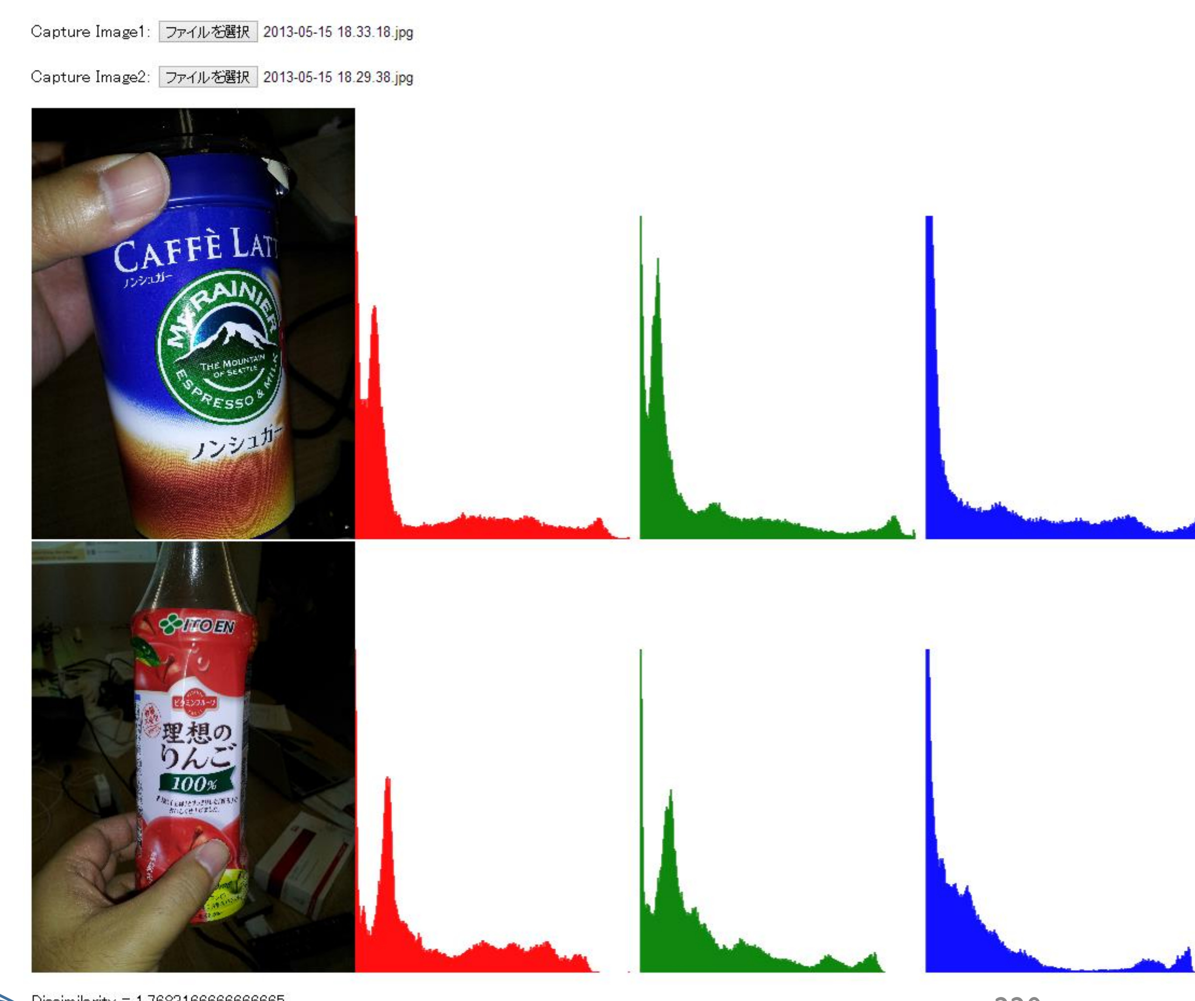
Dissimilarity Calculation

http://web.sfc.keio.ac.jp/~kurabaya/lectures/camera3.html

- We can calculate simple dissimilarity between two images by computing the following equation.

$$dissimilarity(r1, g1, b1, r2, g2, b2) :=$$
$$\sum_{i=0}^{255} |r1_i - r2_i| + \sum_{i=0}^{255} |g1_i - g2_i| + \sum_{i=0}^{255} |b1_i - b2_i|$$

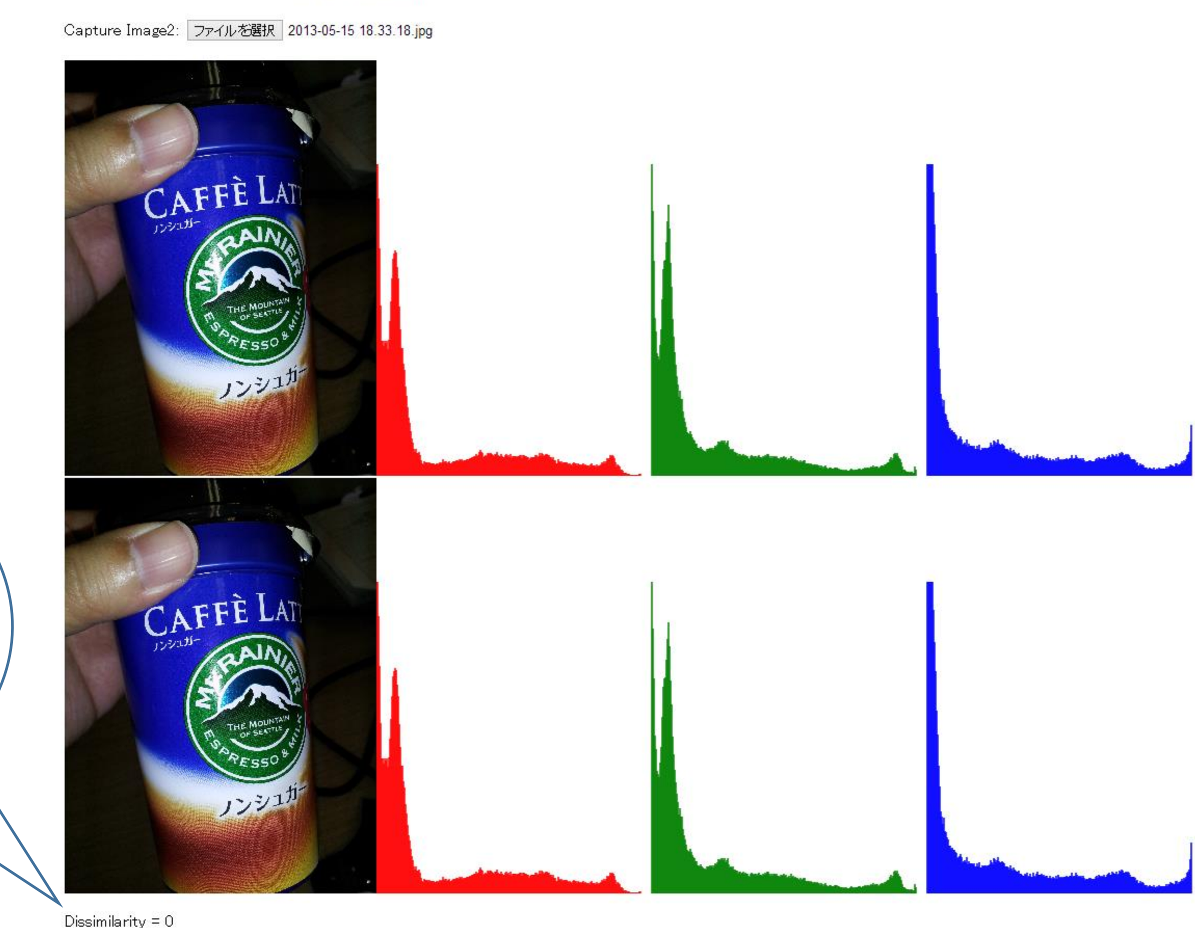
Dissimilarity = 1.7683166666666665



Dissimilarity Calculation

- “Dissimilarity = 0” means that two images are completely same.
- You can generate the similarity ranking by sorting the images in ascending order.

Dissimilarity = 0



function dissimilarity()

- r_array1, g_array1, b_array1 are array generated by applying analyze() function to the first image. They are stored in global.
- r_array2, g_array2, b_array2 are also array generated by applying analyze() function to the second image.

```
function dissimilarity(r_array1, g_array1, b_array1,
  r_array2, g_array2, b_array2) {
    var sum = 0, i;
    for (i = 0; i < 255; i++) {
      sum += Math.abs(r_array1[i] - r_array2[i]);
      sum += Math.abs(g_array1[i] - g_array2[i]);
      sum += Math.abs(b_array1[i] - b_array2[i]);
    }
    return sum;
  }
```

Global Variables

- var r_array1, g_array1, b_array1 are global variable because those are defined outside of functions.
- We can store data in the global variable for enable any functions to share the data.
- Here, we store the analysis result of the first image to the global variable. And then, the system compares the global variables and local variable storing the second image's analysis result to calculate dissimilarity.

```
var r_array1, g_array1, b_array1;

………

if (!r_array1) {
    r_array1 = r_array;
    g_array1 = g_array;
    b_array1 = b_array;
} else {
    var p = document.createElement("p");
    p.innerHTML = "Dissimilarity = " +
      dissimilarity(r_array1, g_array1,
        b_array1, r_array,
        g_array, b_array);
    document.body.appendChild(p);
}
```

GPS Access

http://web.sfc.keio.ac.jp/~kurabaya/lectures/gps.html

- Geolocation API enables web applications to access user's current location.
- navigator.geolocation.watchPosition() method accepts a callback function for monitoring current position.
- navigator.geolocation.clearWatch method stops monitoring.
- "if (pos.coords.accuracy < 300 || processingTime > 15000){}" checks that the accuracy is enough or not. It also checks timeout.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" lang="en"/>
    <meta name=viewport content="width=device-width,user-
scalable=no,initial-scale=1.0,maximum-scale=1.0"/>
    <title>GPS</title>
</head>
<body>
<h1>GPS Example</h1>

<div id="test"></div>
<script type="text/javascript">
    var watchId;
    var startTime;
    var currentTime;

    function display(pos, ptime) {
        var a = document.getElementById("test");
        var b = ["<ul>",
            "<li>time: ", ptime / 1000, "</li>",
            "<li>lat : ", pos.coords.latitude, "</li>",
            "<li>lon: ", pos.coords.longitude, "</li>",
            "<li>accuracy: ", pos.coords.accuracy, "</li>",
            "</ul>"];
        a.innerHTML = b.join('');
    }

    startTime = new Date();
    watchId = navigator.geolocation.watchPosition(function (pos) {
                currentTime = new Date();
                var processingTime = currentTime - startTime;
                display(pos, processingTime);
                if (pos.coords.accuracy < 300 || processingTime > 15000) {
                    console.log("end");
                    navigator.geolocation.clearWatch(watchId);
                }
            }, function (e) {
                alert(e.message);
            },
            {enableHighAccuracy: true, timeout: 1000, maximumAge: 0});
</script>
</body>
</html>
```

2:57 PM

GPS

GPS Example

- time: 3.852
- lat : 35.3967309
- lon: 139.4579551
- accuracy: 1445

First noise filtering in devicemotion event

- In devicemotion event listener, the program filters out small motion whose magnitude is smaller than 0.15.
- Math.abs(magnitude) > 0.15
  - This program records the data that is greater than 0.15 into the array log.

```
window.addEventListener('devicemotion', function (e) {
    var magnitude = gravityDirectionMagnitudeForMotion(e.acceleration,
                        e.accelerationIncludingGravity);
    if (Math.abs(magnitude) > 0.15) {
        log.push(magnitude);
        dd[0].innerHTML = magnitude.toFixed(2);
    }
});
```

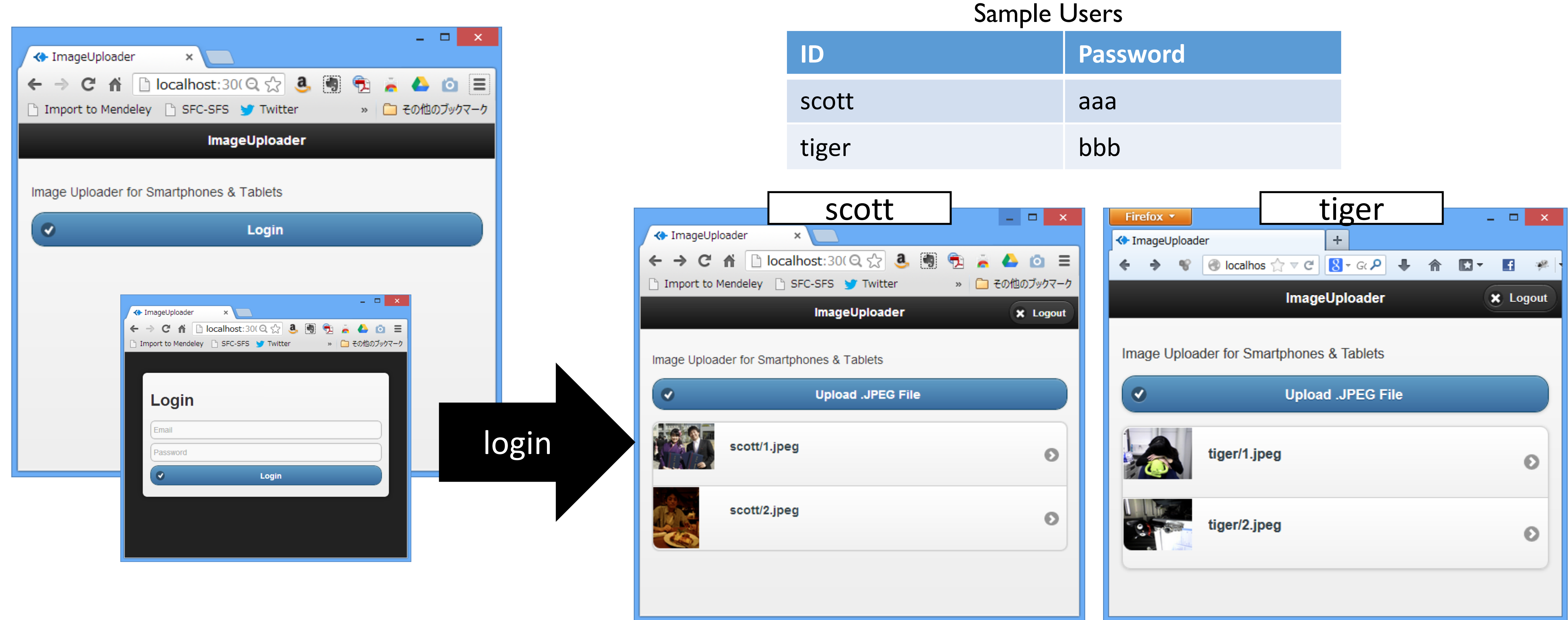
User Authentication

Session Management and User Authentication in Server-Side

Image Uploader (Multi-User Version)

http://web.sfc.keio.ac.jp/~kurabaya/lectures/ImageUploader2.zip

- As an example of user authentication, the previous image uploader application is extended to support multiple users.
- You can login to this application by using “scott” or “tiger” account.



Sample Users

ID	Password
scott	aaa
tiger	bbb

Authentication = Identification + Protection + Registration

Identification

- The system must recognize who are logging in continuously. Because HTTP is a stateless protocol, the session management mechanism must be provided to track a user between multiple HTTP requests. Express framework provides a session management mechanism utilizing HTTP cookie.

Protection

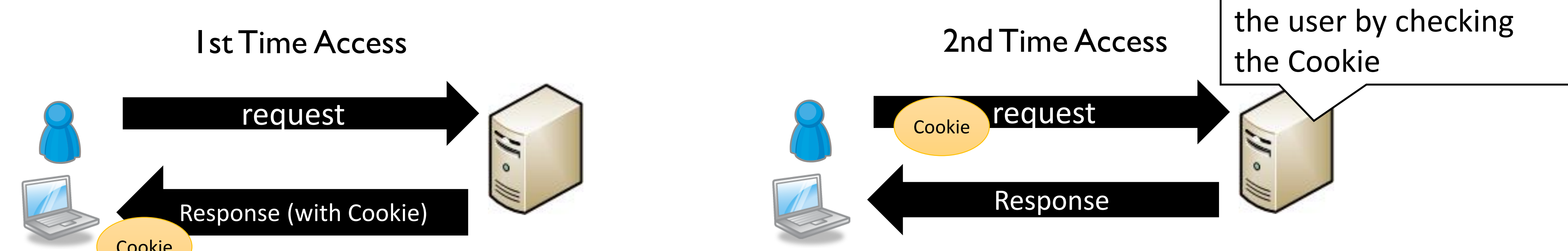
- Invalidating unauthorized access. For example, unauthorized user should not upload image files to the server.

Registration

- Adding a new user. In this example, we skip the registration process. You can implement user registration by using OAuth services provided by Facebook and Twitter.

HTTP Cookie

- HTTP Cookie is a client identification mechanism commonly available in modern web browsers.
  - Cookie is like the token supplied at a coat check counter, because its uniqueness allows it to be exchanged for the correct coat when returned to the coat check counter.
- When a user visits a website, the web server sends a response including a HTTP cookie, and the browser stores the received cookie. When the same user returns to the website, the stored cookie is sent to the web server for identifying the user.



Index.html

- We have modified the previous example image uploader by adding the following elements.
- Login Dialog
  - Login dialog is a modal dialog for inputting a user name and a password.
- Login Button
  - This button opens a login dialog.
- Logout Button
  - This is a invisible when a user has not logged in the site.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>ImageUploader</title>
    <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.1/jquery.mobile-
1.3.1.min.css"/>
    <script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
    <script src="http://code.jquery.com/mobile/1.3.1/jquery.mobile-1.3.1.min.js"></script>
</head>
<body>
<div data-role="page" id="home" class="type-home">
    <div data-role="header">
        <h1>ImageUploader</h1>
        <a href="#" id="logoutButton" data-icon="delete"
            class="ui-btn-right" style="visibility: hidden">Logout</a>
    </div>
    <div data-role="content">
        <div>
            <p>Image Uploader for Smartphones &amp; Tablets</p>
            <span id="buttonArea"></span>
        </div>
        <div id="listHolder"></div>
    </div>
</div>

<div data-role="dialog" id="openDialog">
    <div data-role="content">
        <div>
            <h1>Login</h1>
            <input type='text' name='email' placeholder='Email'/>
            <input type='password' name='password' placeholder='Password'/>
            <a href="#" data-role="button" data-icon="check" data-theme="b"
               data-rel="back">Login</a>
        </div>
    </div>
</div>

<input type="file" name="file" style="visibility: hidden"/>
```

Loads jQuery, jQueryMobile, and CSS

Invisible logout button

Login dialog provides input text field for user name and password.

Client-Side JavaScript authState()

- authState() function asks to the server the current authentication status of the client.
- Authentication is performed in the server-side, thus the client must not manage the authentication status.
- In this application, the client asks the current status whenever the page is rendered.

```
function authState() {
    $.ajax({
        type: "GET",
        url: "/auth",
        dataType: "text",
        success: function (data) {
            if (data == "true") {
                var uploadButton = $('<a data-icon="check" data-role="button" data-theme="b">Upload .JPEG File</a>');
                uploadButton.click(function (e) {
                    $('input[type=file]').get(0).click();
                });
                $("#buttonArea").empty();
                $("#buttonArea").append(uploadButton);
                $("#logoutButton").css("visibility", "visible");
                $("#logoutButton").button();
                uploadButton.button();
                update();
            } else {
                var loginButton = $('<a href="#openDialog" data-role="button" data-icon="check" data-theme="b">Login</a>');
                $("#buttonArea").empty();
                $("#buttonArea").append(loginButton);
                loginButton.button();
                $('#listHolder').empty();
                $("#logoutButton").css("visibility", "hidden");
            }
        }
    });
}
```

Login dialog provides input text field for user name and password.

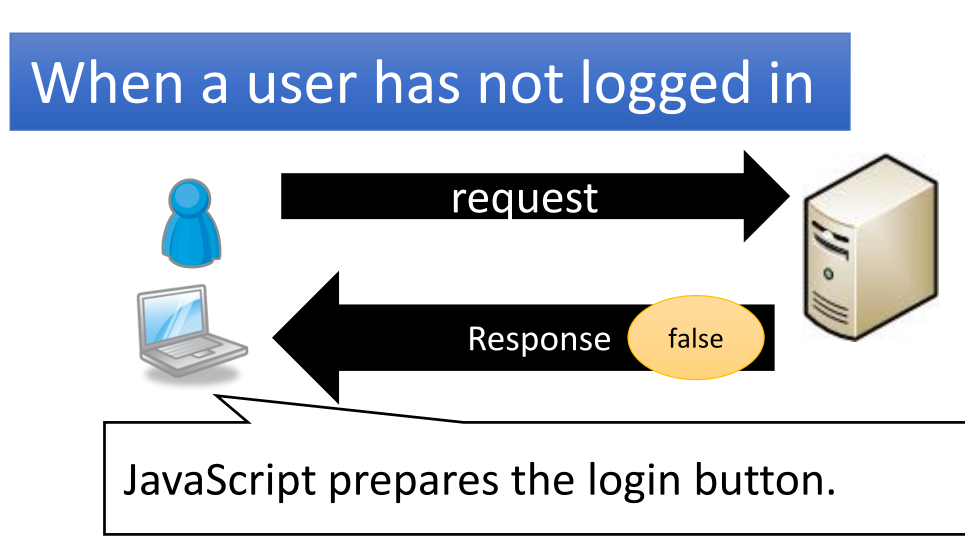
Login dialog provides input text field for user name and password.

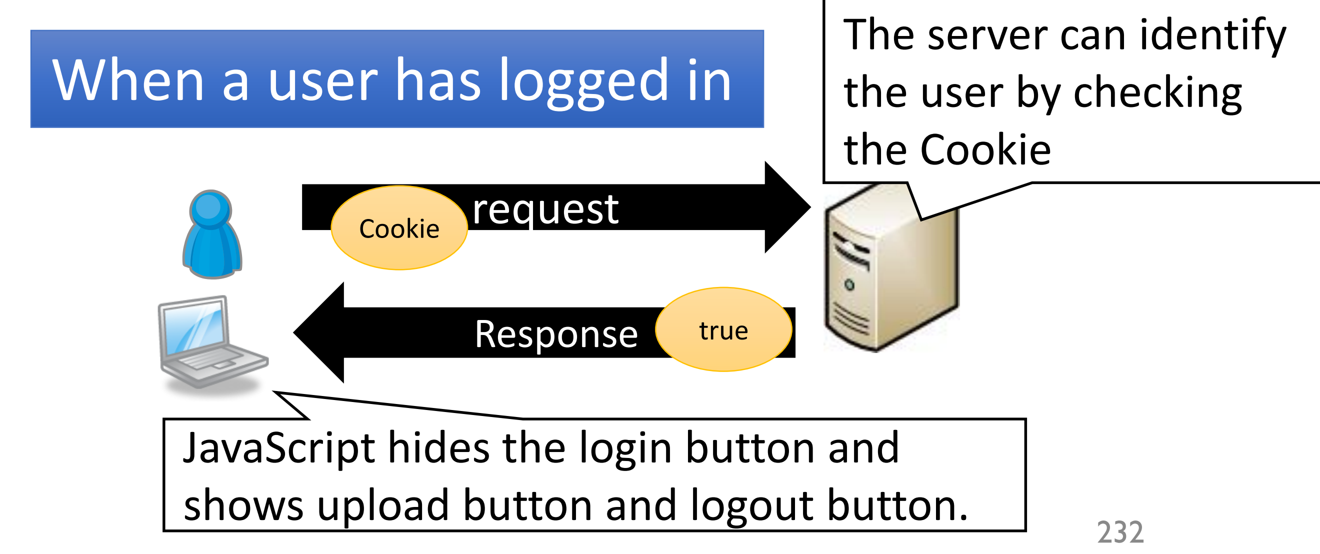
Login dialog provides input text field for user name and password.

Login dialog provides input text field for user name and password.

Authentication State Management

- When a user has not logged in, the server /auth API returns "false" response. So, the client-side JavaScript prepares the login button.
- When a user has logged in, the server /auth API returns "true" response. So, the client-side JavaScript hides the login button and shows upload button and logout button.





Client-Side JavaScript update()

- The multi-user version uses the update() function which is almost same as the previous version.
- When a user, who has not logged in, accesses to the /list API, the server sends “auth_required” message as a response.

```
function update() {
    $.getJSON("/list", function (data) {
        if (data == "auth_required") {
            console.log(data);
        } else {
            var ul = $('<ul data-role="listview" data-inset="true" data-theme="c" data-dividertheme="b"></ul>');
            $('#listHolder').empty();
            $('#listHolder').append(ul);
            data.forEach(function (elem) {
                ul.append("<li><a data-ajax='false' href='images/" + elem
                    + "'><img src='images/" + elem
                    + "'/><h3>"
                    + elem + "</h3></a></li>");
            });
            $('ul[data-role="listview"]').listview();
        }
    });
}
```

Client-Side JavaScript login process

- $("#openDialog a").click events is fired when a user clicks the “login” button.
- This event listener sends a user name and a password to /auth API.
- /auth API returns “true” when the login is successful.

{user: $(“input[type=text]”).val(), password: $(“input[type=password]”).val()} creates a data object consists of user and password.

/auth API returns “true” when the login is successful

```
$("#openDialog a").click(function (e) {
    $.ajax({
        type: "POST",
        url: "/auth",
        dataType: "text",
        data: {user: $("input[type=text]").val(), password: $("input[type=password]").val()},
        success: function (data) {
            if (data == "true") {
                authState();
            } else {
                alert("invalid user name or password");
            }
        },
        error: function (XMLHttpRequest, textStatus, errorThrown) {
            console.log(errorThrown);
        }
    });
});
```

Client-Side JavaScript
logout process and others

- When /auth API receives DELETE command, the server invalidates the login session by deleting the corresponding session information.
- Logout button's event listener sends DELETE command to /auth API.

```
$("#logoutButton").click(function (e) {
    $.ajax({
        type: "DELETE",
        url: "/auth",
        dataType: "text",
        success: function (data) {
            authState();
        }
    });
});

$('input[type=file]').change(function (e) {
    var xhr = new XMLHttpRequest(),
        fileList = $('input[type=file]').get(0).files;
    xhr.open("POST", "/upload", true);
    xhr.setRequestHeader("Content-Type", "application/octet-stream");
    xhr.send(fileList[0]);
    update();
});

authState();
```

Logout button's event listener sends DELETE command to /auth API.

File upload function

In this application, the client asks the current status whenever the page is rendered.

Server-Side Framework with Session Management

- To enable the session management mechanism in Express framework, app.use(express.session()) parameter should be inserted.
- {secret: "secretcode"} means an encryption salt for encrypting HTTP cookie for protecting ID from session hijacking.

```
app.configure(function () {
    app.set('port', process.argv[2] || 3000);
    app.use(express.favicon());
    app.use(express.logger('dev'));
    app.use(express.bodyParser());
    app.use(express.methodOverride());
    app.use(express.cookieParser());
    app.use(express.session({secret: "secretcode"}));
    app.use(app.router);
    app.use(express.static(path.join(__dirname,
'public')));
});
```

Server-Side API

/auth

- This API is for login, logout, and checking login status.
- When a client sends GET command, /auth checks the current login status.
- When a client sends POST command, /auth tries to login.
- When a client sends DELETE command, /auth invalidates the current login session.

/upload

- This API accepts the file to be uploaded.

/list

- This API returns a list of files. This API is modified to support multiple users.

/auth API implementation in Server-Side

- App.get invokes a callback function when the node.js server receives GET command. In this case, /auth API returns true when the session object (req.session) has user object.
- App.post invokes a callback function when the node.js server receives POST command. POST command can receive additional parameters such as a user name and a password. The received parameter can be accessed via req.body object. In this case, /auth API appends user attribute to the session object (req.session) when login process is successful.
- App.delete invokes a callback function when the node.js server receives DELETE command. Delete operator removes a specified attribute from an object. In this case, user attribute in req.session will be removed.

```
app.get('/auth', function (req, res) {
    res.setHeader('Content-Type', 'text/plain');
    if (req.session.user) {
        res.send("true");
    } else {
        res.send("false");
    }
});
```

```
app.delete('/auth', function (req, res) {
    res.setHeader('Content-Type', 'text/plain');
    delete req.session.user;
    res.send("true");
});
```

```
app.post('/auth', function (req, res) {
    var result = false;
    users.forEach(function (u) {
        if (u.user == req.body.user && u.password ==
req.body.password) {
            req.session.user = req.body.user;
            result = true;
        }
    });
    if (result) {
        res.send("true");
    } else {
        res.send("false");
    }
});
```

/list API

- /list API invokes loginCheck function for checking the current login status, before the execution of function (req, res) {}.
- This mechanism is called “middleware chain”.

```
app.get('/list', loginCheck, function (req, res) {
    fs.readdir(dataDir + req.session.user + "/", function (err, files) {
        res.setHeader('Content-Type', 'application/json');
        var i;
        for (i = 0; i < files.length; i++) {
            files[i] = req.session.user + "/" + files[i];
        }
        console.log(files);
        res.send(JSON.stringify(files));
    });
});
```

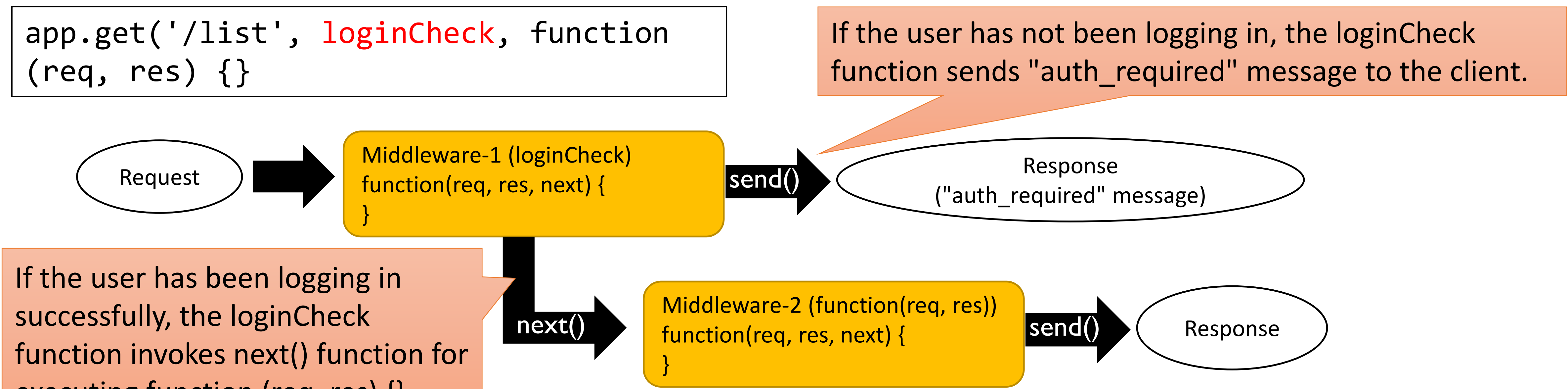
loginCheck function

- req.session is a special object automatically created by express framework. When a new client access to the server, the corresponding req.session is created automatically.
  - We can store any user-dependent attribute, such as login information and shopping cart information, to req.session object.
  - Express offers a persistent mechanism for req.session by storing it to MongoDB and Redis server.
- loginCheck function checks login status by inspecting the req.session.user attribute.
  - it invokes the next() function chain when the user has been logging in.
  - It sends "auth_required" message if the user has not been logging in.

```
var loginCheck = function (req, res, next) {
    if (req.session.user) {
        next();
    } else {
        res.send("auth_required");
    }
};
```

Middleware Chain in Express

- Middleware mechanism in express provides a chain of functions for processing request.
- app.get, app.post, app.delete function can accept multiple callback function. Those callback functions are executed sequentially, by executing next() function in each callback function.
- In this example, loginCheck function is executed before the execution of function(req, res).



/upload API

- /upload API also uses loginCheck middleware to protect the resource from unauthorized upload API call.

```
app.post('/upload', loginCheck, function (req, res) {
    var files = fs.readdirSync(dataDir + req.session.user + "/"),
        max = 0,
        outStream;

    files.forEach(function (file) {
        max = Math.max(max, Number(file.slice(0, file.lastIndexOf('.'))));
    });
    outStream = fs.createWriteStream(dataDir + req.session.user + "/" + (++max) + '.jpeg');
    req.pipe(outStream);
    outStream.on("close", function () {
        res.send("true");
    });
});
```

User Registration

- Facebook Login and Twitter Login make it easy to implement user authentication mechanism in your web applications.
- For example, password change, email-based password reminder, two-phase authentication mechanism can be integrated to your system.
- https://developers.facebook.com/docs/facebook-login/
- https://dev.twitter.com/docs/auth/sign-twitter

MVC Implementation using Backbone.js Practice

Model–View–Controller Model for Web Application from JavaScript

A web application from JavaScript is composed of a DOM structure as the Model, CSS as the View, and Form and JavaScript event listeners as the controller.

Model

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Command Pattern</title>
</head>
<body>
</body>
</html>
```

Defines the data structure. Model is independent of the View or Controller

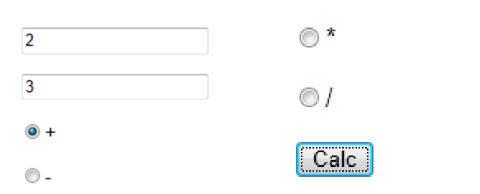
View supplies a “way of showing” data. Corresponds to GUI components

Controller supplies the “method of operation.” Corresponds to processing such as the operation when the button is clicked

View

```
<style>
html, body {
    margin: 0;
    padding: 0;
    font-family: sans-serif;
    background: #eeeeee;
}
</style>
```

Controller



What is Backbone.js?

- Client-side JavaScript MVC framework
- Model
  - In charge of data operation, and data generation, validation, discard, server-side data storage
- View
  - Receives an “event” from the “model” and draws up the model “data.” In Backbone.js, the model is changed, “view” automatically updates the display of these changes.

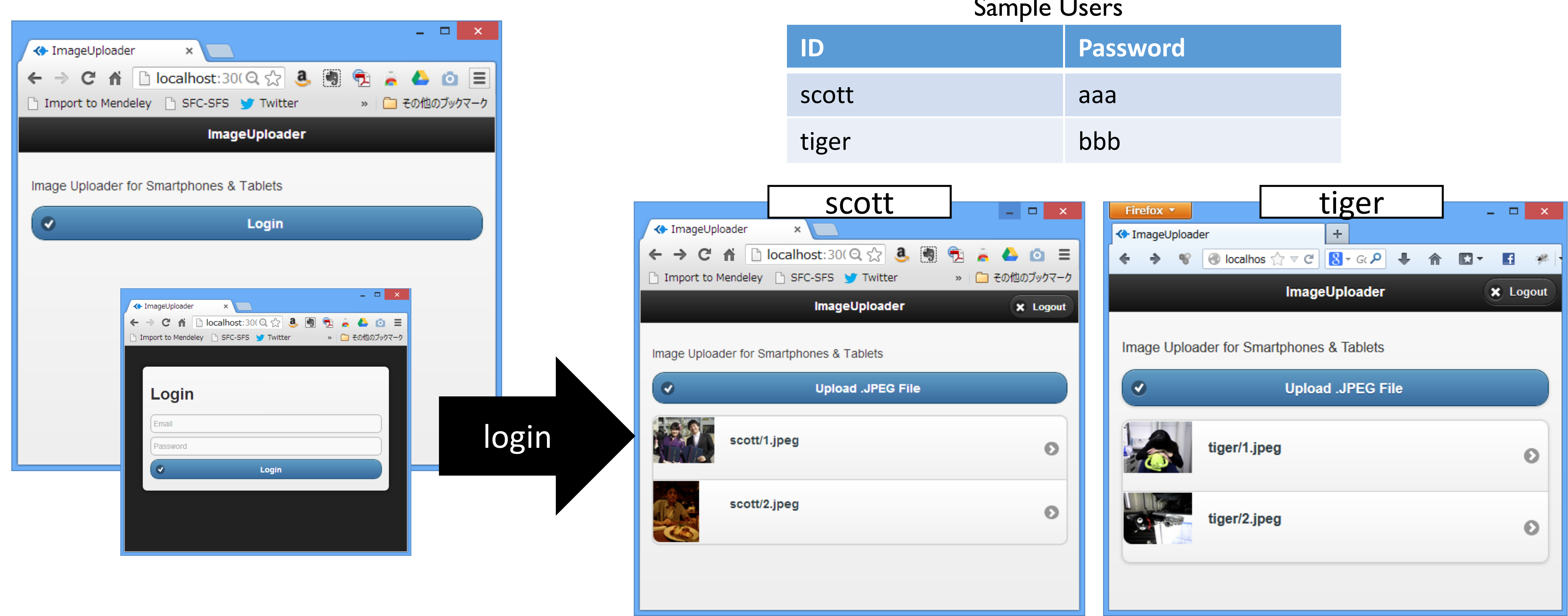
Sample

http://web.sfc.keio.ac.jp/~kurabaya/lectures/ImageUploader3.zip

- Source codes of the image uploader was slightly modified by using MVC model.

Sample Users

ID	Password
scott	aaa
tiger	bbb



Model

- An object that models the Image and an object that models the Image set.
- Model is created from JSON returned by the server.
  - To specify the URL of JSON, ImageCollection object has “url” attribute.
- Initialize method is called when the model object is created.

```
var ImageModel = Backbone.Model.extend({
    initialize: function (attrs, options) {
        this.set("id", this.id);
    }
});

var ImageCollection = Backbone.Collection.extend({
    model: ImageModel,
    url: '/images'
});
```

```
var ApplicationModel = Backbone.Model.extend({
    initialize: function (attrs, options) {
        this.images = new ImageCollection();
    }
});
```

View

- The object that decides the look of the image displayed on the screen
- This time, an HTML tag for displaying image is generated using the template.
- View also specifies events and event listeners. Here events “click a” triggers onClick method.
- Render method appends HTML tags generated by View to the main HTML tree.

```
var ImageView = Backbone.View.extend({
    tagName: "li",
    template: _.template('<a id="image_<%=id%>" href="#"><img src="<%=imageURL%>"/><h3>images/<%=id%>.jpg</h3></a>'),
    events: {
        "click a": "onClick"
    },
    onClick: function (evt) {
        var model = this.model;
        $("#detailPage h1").html(model.get('id'));
        $("#detailPage img").attr("src", model.get('imageURL'));
        $("#deleteButton").unbind();
        $("#deleteButton").click(function (e) {
            model.destroy();
            $.mobile.changePage($("#home"), {changeHash: false});
        });
        $.mobile.changePage($("#detailPage"), {changeHash: false});
    },
    render: function () {
        $(this.el).html(this.template(this.model.toJSON()));
        return this;
    }
});
```

Template

- Underscore.js provides a template method that generates the HTML by using values in JSON object.
- <%= %> is a special tags that are replaced with the corresponding values in JSON. For example, <%=id%> is replaced with obj.id.
- template(this.model.toJSON()) means that generating HTML using the model's JSON.

```
template: _.template('
<a id="image_<%=id%>" href="#">
  <img src="<%=imageURL%>"/>
  <h3>images/<%=id%>.jpg</h3>
</a>');
```

Initialization of Application

- jQuery Mobile automatically decorates HTML for creating rich UI. So we have to wait the completion of the jQuery Mobile's decoration process.
- jQuery Mobile's page object triggers pageinit event when it finishes the page decoration. The following event listener can detect the completion of page decoration.

```
$("#home").on("pageinit", function (evt) {
    var model = new ApplicationModel({});
    var view = new ApplicationView({model: model});
});
```

ApplicationView

- el represents ID of DOM node corresponding to this view.
- Initialize method is invoked when this object is created.
- authState method is same method in the previous example. It was tailored to MVC model.
- resetItems method is triggered when the model is initialized.
- removeItem method is invoked when the existing item is deleted from the model.
- appendItem method is invoked when a new item is inserted to the model.

```
var ApplicationView = Backbone.View.extend({
    el: "#home",
    initialize: function () {
        var self = this;
        this.model.images.bind("reset", this.resetItems, this);
        this.model.images.bind("add", this.appendItem, this);
        this.model.images.bind("remove", this.removeItem, this);
        this.authState();

        $('input[type=file]').change(function (e) {
            ……
        });

        $("#openDialog a").click(function (e) {
            ……
        });

        $("#logoutButton").click(function (e) {
            ……
        });
    },
    authState: function () {
            ……
    },
    resetItems: function (collection) {
        this.ul.empty();
        collection.each(function (model) {
            this.appendItem(model);
        }, this);
    },
    removeItem: function (model) {
        console.log(model);
        $("#image_" + model.get('id')).remove();
        $("ul[data-role=listview]").listview('refresh');
    },
    appendItem: function (model) {
        var view = new ImageView({model: model});
        $("ul[data-role=listview]").append(view.render().el);
        $("ul[data-role=listview]").listview('refresh');
    }
});
```

ApplicationView updates the contents in response to changes in the model.