

# Webの振り返りとHTTP

最初に

# 最近のWebアプリ開発事情を棚卸し

Webアプリ開発? → HTTPに乗るもの、関連するものすべて

# 最近のWebアプリ開発事情

- フロントエンドWebアプリフレームワーク (React、Vue、Angular)
- バックエンドWebアプリフレームワーク (Laravel、Rails、Express)
- データベース (RDB、NoSQL)
- クラウド (AWS、GCP、Azure)
- コンテナ (Docker、Kubernetes)
- プロトコル (HTTP、TCP/IP、SSL/TLS、gRPC)
- ...

↑ 混沌 (この内容はほとんど出てきません)

# この資料について

- 目的
  - Webについて分かった気にさせる
    - 前編: Webを振り返って技術の流れを把握する
    - 後編: HTTP、TCPについての理解を深める

# 前編：Webを振り返って技術の流れを把握する

①ブラウザの仕事

②ブラウザとWeb技術の移り変わり

# ブラウザの仕事



Google 検索

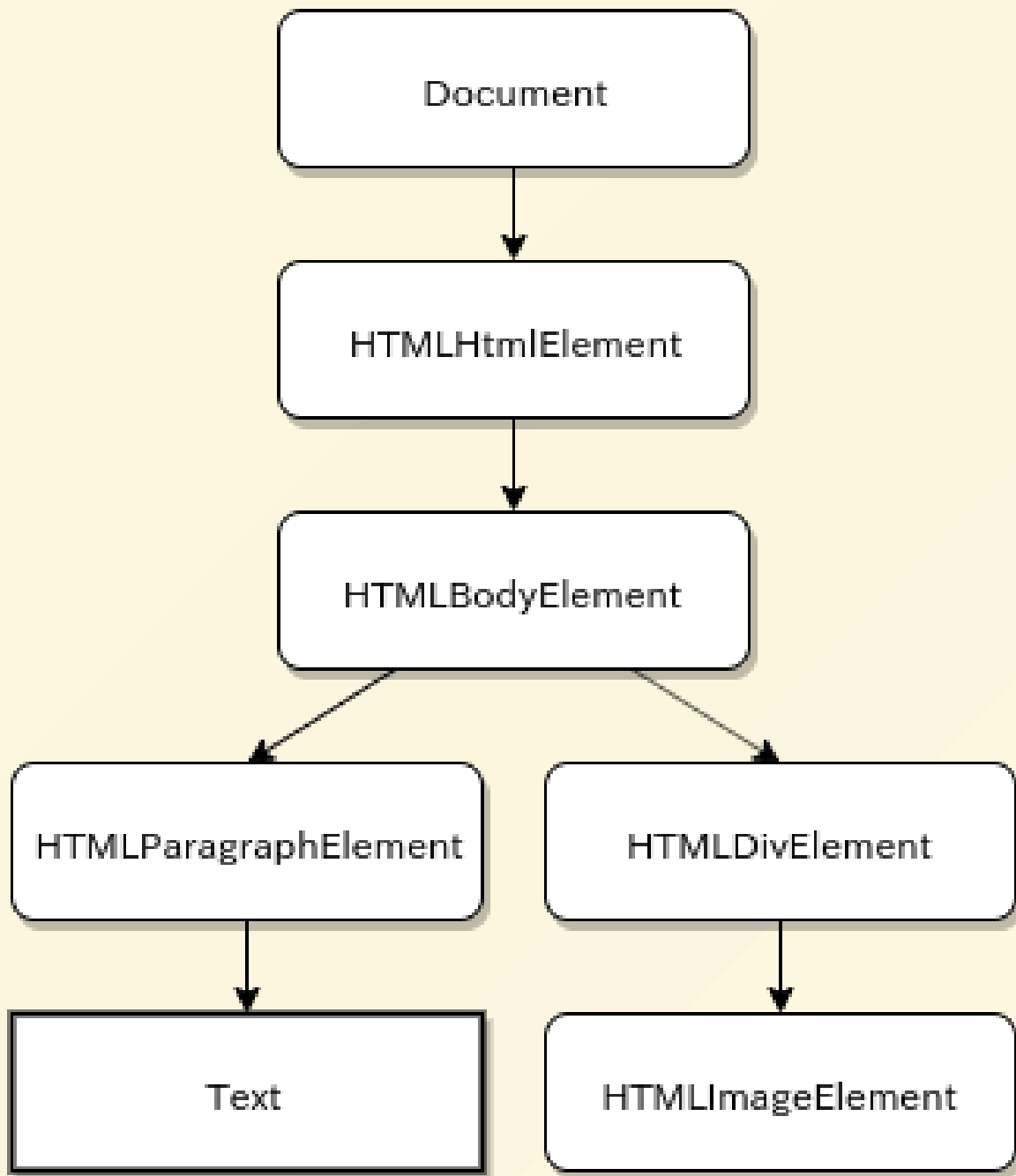
I'm Feeling Lucky

Google の無料オンライン コースを今年のうちに始めよう

# 主な機能

HTMLやCSSなどを取得し、  
パースして、画面に表示する。  
JavaScriptの実行もする。

- レンダリングエンジン
  - HTML解析 → DOM Tree
  - CSS解析 → Style Rules
  - 画面表示
- JavaScriptエンジン



# DOM

Document Object Model

ブラウザ内部でHTMLは  
DOMツリーとして保持される。

JavaScriptからDOMのルート  
「Document」オブジェクト  
を通じてDOMを操作できる！

```
document.getElementById('foo')
```

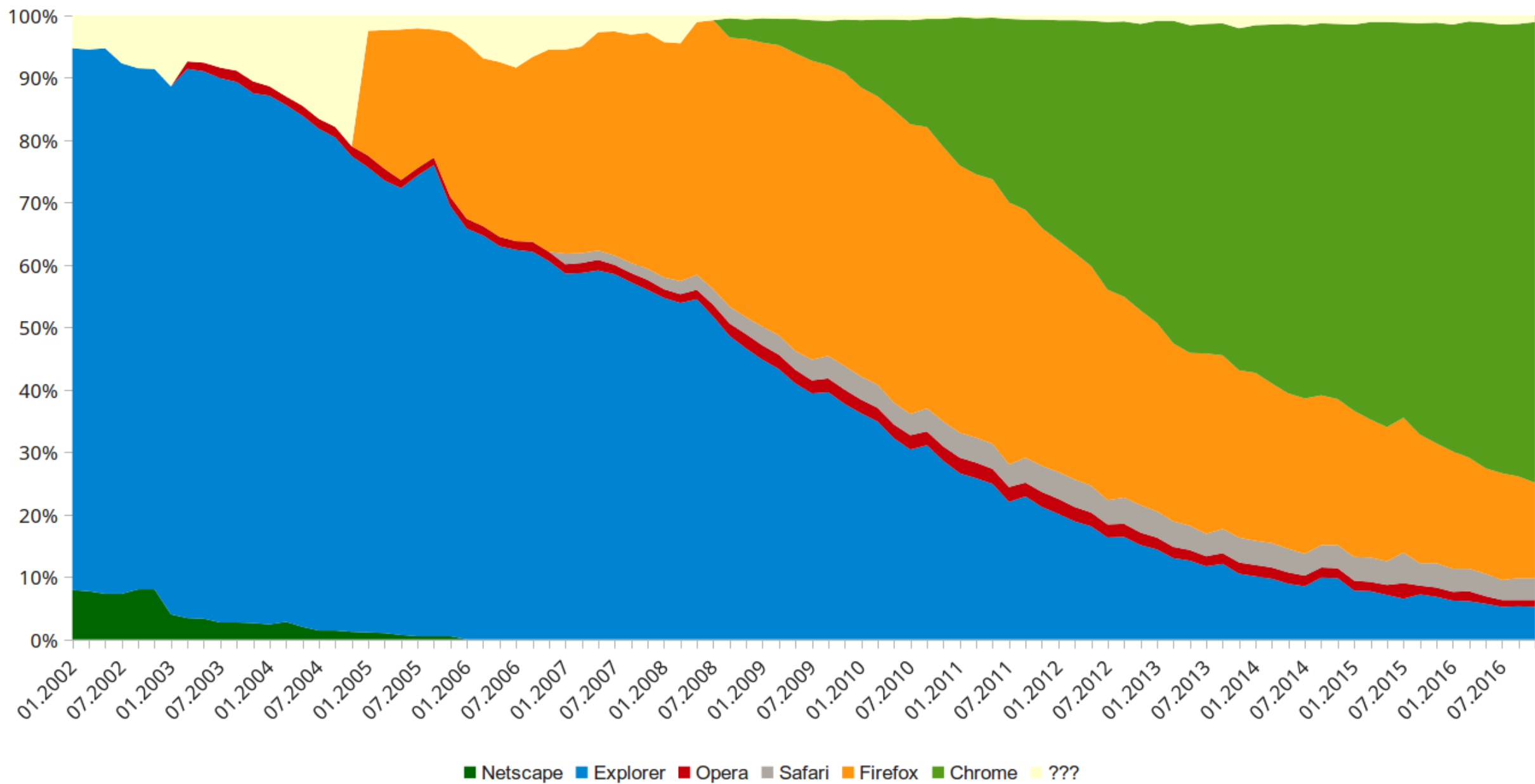
```
$('#bar').innerHTML = 'sample'
```

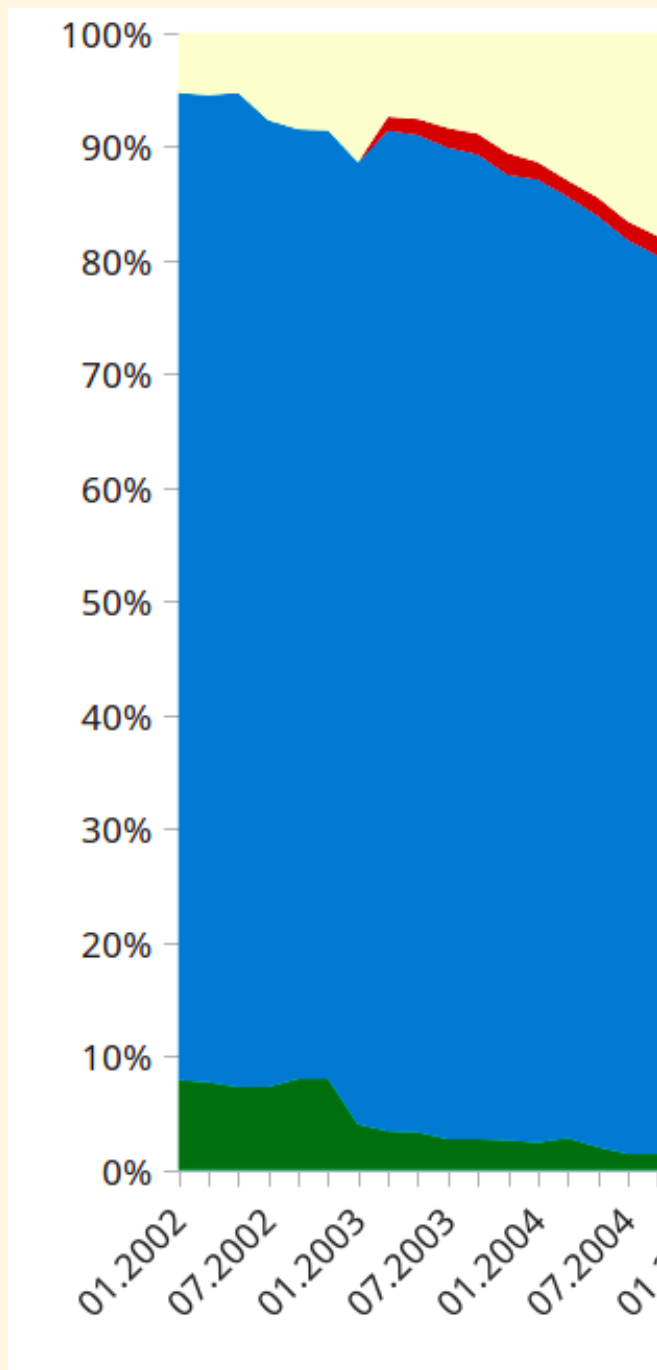


# JavaScript実行環境

- JIT型
  - Chakra Legacy (IE11)
  - Chakra (Edge)
  - SpiderMonkey (Firefox)
  - V8 (Chrome、Node.js)
- インタプリタ型
  - Ignition (V8) (Android Chrome)

# ブラウザとWeb技術の移り変わり





## ～2004年

- IE全盛期 (IE 6)
- FLASH黄金時代

この頃のJavaScriptは  
「ちょっと動きを加えるもの」

リッチなものは全部FLASH!  
JavaScriptは無効に設定!

WebアプリはLAMPが最強



## ～2006年

- IEまだ強い (IE 7)
- Firefox
- Ajax、jQuery

2005年にGoogle Mapsが登場。

# 2005年:XMLHttpRequest (Ajax)

Webブラウザのスクリプト言語 (JavaScriptなど) からサーバとHTTP通信を行うために用意されたブラウザのAPI。

Goole MapsでXMLHttpRequestが有名になり、Ajaxという言葉が生まれる。(Asynchronous JavaScript + XML)

しかしまだクライアントプログラミングの敷居は高い...

.

※ ほかのWebブラウザのスクリプト言語

Javaアプレット、VBScript、JScript、ActionScript、Silverlight環境など

# 2006年:jQuery

クライアントプログラミングの敷居を一気に下げた存在

- かんたんDOM操作
- かんたんイベント処理
- かんたんAjax
- ブラウザによる挙動の差異を吸収

やりたいことがそこそこ良いカンジでできる。(まだまだ現役!)

```
$('#hoge')
```

# 2006年:jQuery②

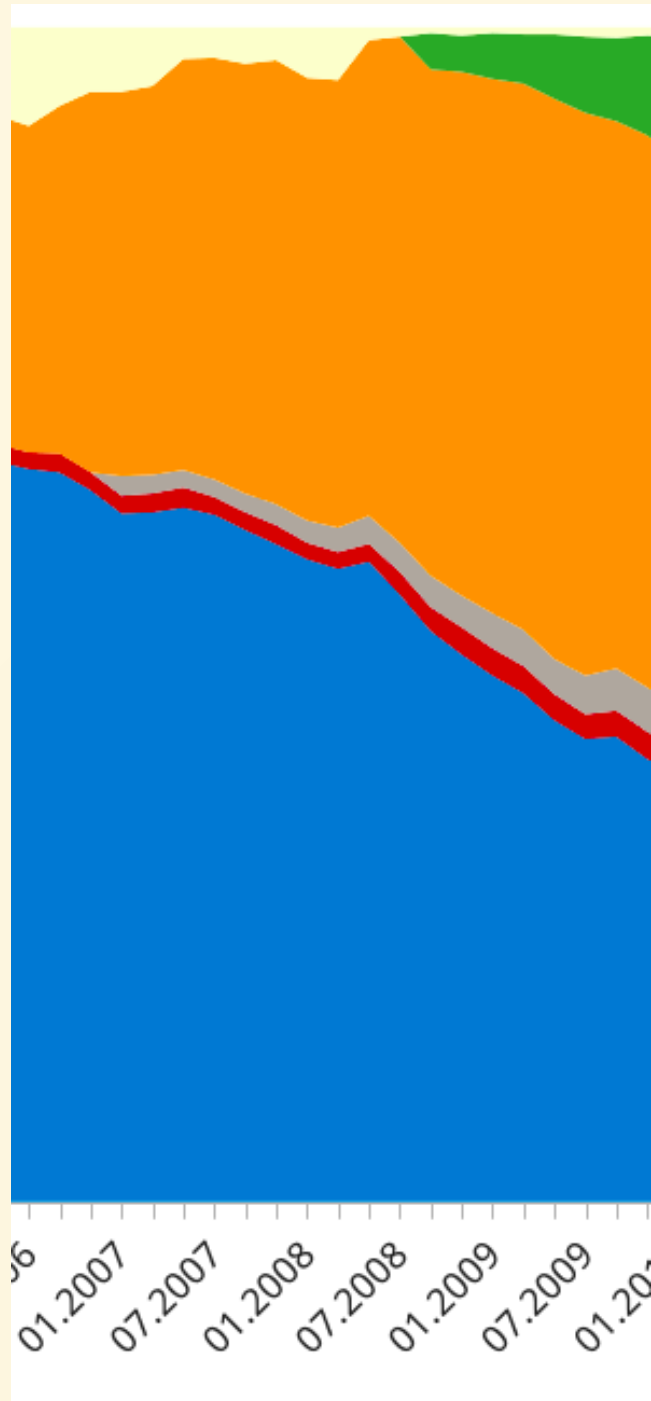
なにが辛いかな

- 値の管理
- DOMの状態管理
- イベントの発火管理
- ...

コンポーネントが増えるたび、やることが指数関数的に増えていく。

**一部の職人にしか成し得ない超絶技巧プログラミング**

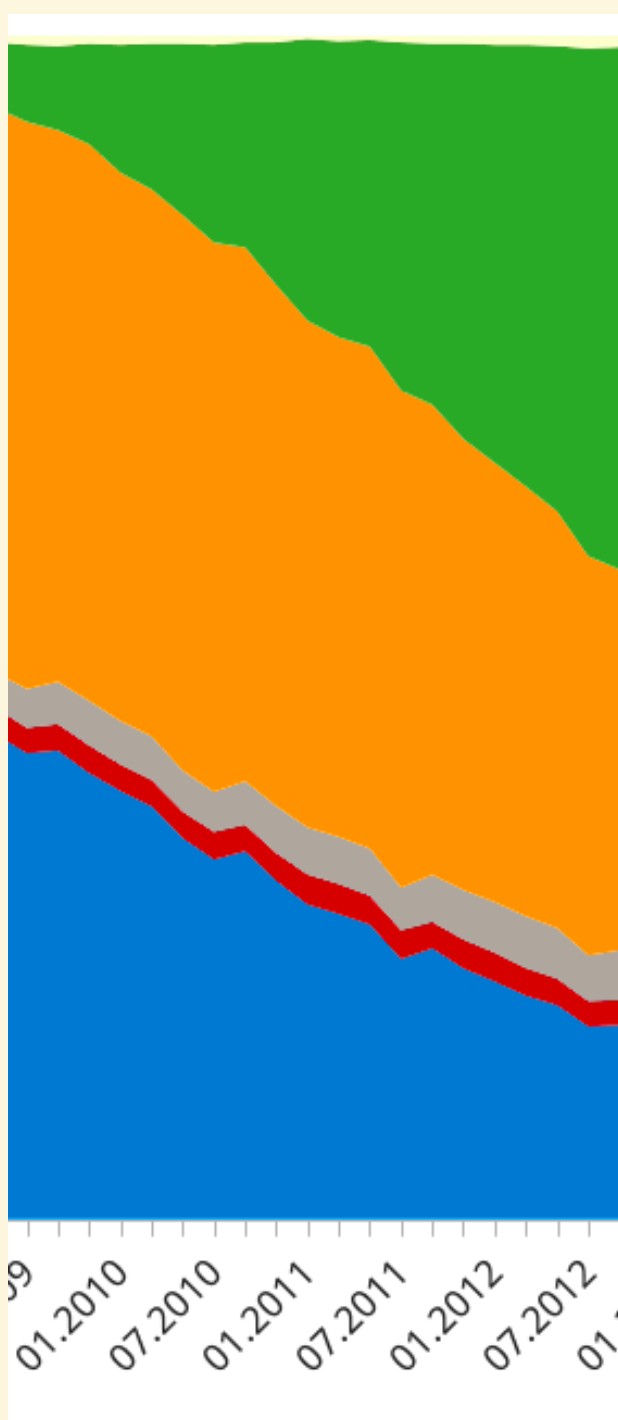




## ～2009年

- PHPフレームワーク乱立問題
- 2004年生まれのRuby on Railsが頭角を表す
- IE (IE 8) 以外のブラウザがシェアを伸ばし始めた時代

タブブラウジング、フィードリーダー、自前のレンダリングエンジン搭載のような独自機能を追加したブラウザがたくさん生まれた。



## ～2012年

- IE (IE 9、10) 完全に下火
- HTML5/CSS3の対応が進む
  - WebSocketが登場
- FuelPHP、Laravelはこのへん
- クラウドブーム

CSS3のメディアクエリ @media

→ レスポンシブデザインが主流

# HTML5とSingle Page Application

2011年の時点ですでに多くのブラウザがHTML5に対応していた。  
(IE 9、Firefox 3.5、Chrome 3.0など) (HTML5の正式な勧告は2014年)

HTML5では `history.pushState()` を使ってURLの動的書き換えが可能

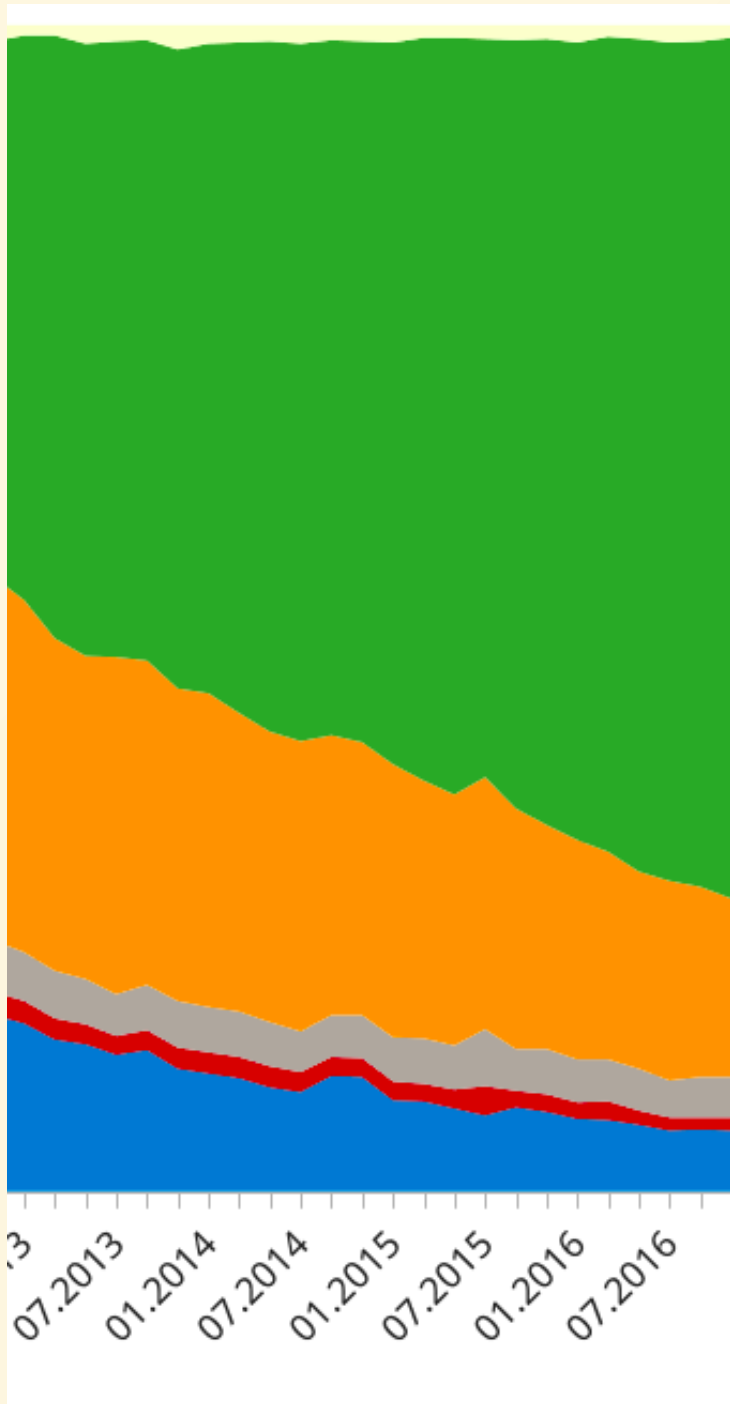
- ネイティブアプリのように、ブラウザのページ遷移を使わず複数ページあるWebアプリを作成することが可能に!
- シングルページアプリケーション!!

# jQueryとSingle Page Application

jQuery + Single Page Application...?

- ただでさえ辛いjQuery
- 考慮しないといけない点が増えすぎる
  - ページ管理
  - ページを跨いだデータ、イベント管理
  - 今までブラウザが管理していた情報をクライアントが管理
    - `history.back()` でのスクロール位置保持など

正気の沙汰ではない。



## ～2016年

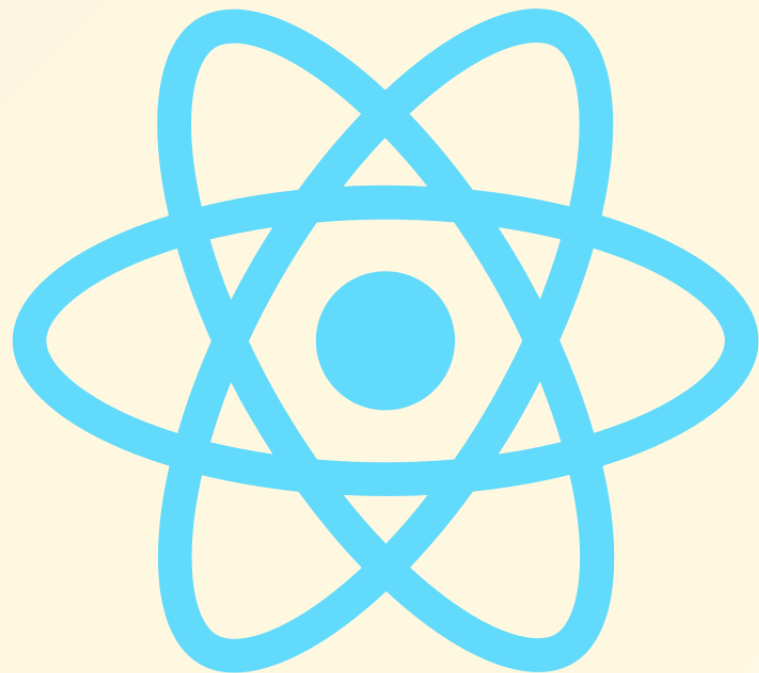
- **React** (2013年)
- Docker (2013年)
- Vue.js (2014年)
- TypeScript (2014年)
- Kubernetes (2015年)

Reactの台等もあり、SPA +  
APIサーバーのアプリが主流に

# React

- Facebook製ライブラリ
- ユーザインタフェースを構築
- コンポーネント指向
- **VirtualDOM**

jQueryを使って自分でDOMを操作しなくていい時代が到来！



# Virtual DOM (仮想DOM)

ブラウザのDOMと対になる、Reactが保持する構造体。

1. ブラウザでアクションが発生するとReactは仮想DOMを変更
2. 変更前の仮想DOMと変更後の仮想DOMを比較し、差分を抽出
3. ReactがブラウザのDOMを変更

Reactが内部でdiff/patchしてくれるため、直接DOMを触る必要がない。

→ 把握・管理しないといけないものが減り、SPAが作りやすくなった

# 前半の内容

- ブラウザの仕事
  - レンダリングエンジン
  - JavaScriptエンジン
- ブラウザとWeb技術の移り変わり
  - Ajax、jQuery
  - SPA、React
  - 主要なWeb技術の登場シーン



# 後編：HTTP、TCPについての理解を深める

①HTTP、TCPを知る

②もっとTCPを知る

# HTTP、TCPを知る

# HTTP

主にWebでブラウザ・サーバー間の通信に使われる**プロトコル**。

- <http://example.com>
- <https://google.com>

https (HTTP Secure) はHTTPの暗号化通信をするやつ。  
(最近のブラウザは `http` だと怒る



保護されていない通信

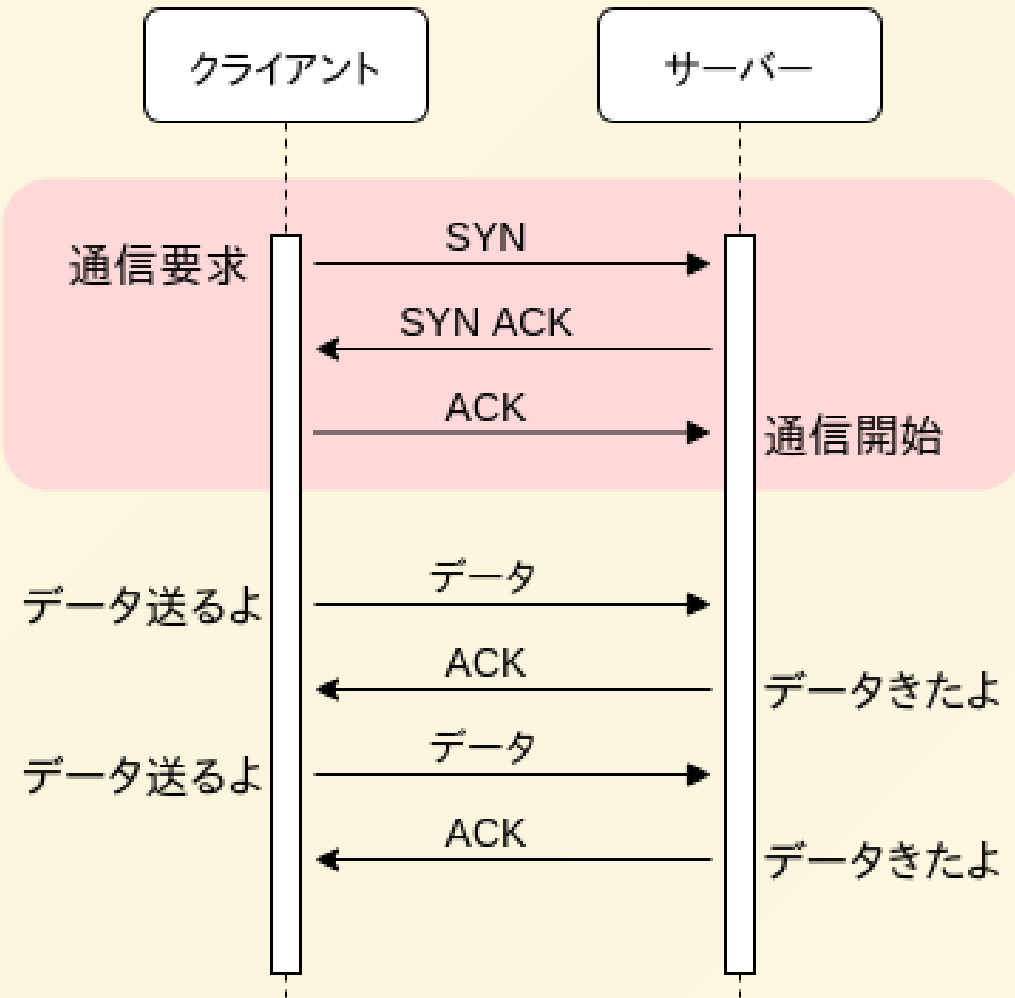
| example.com

# プロトコル？

現代のネットワーク技術を説明するにはレイヤーという概念がとても便利。  
プロトコルの仕事を分けた **TCP/IP参照モデル** などが存在する。

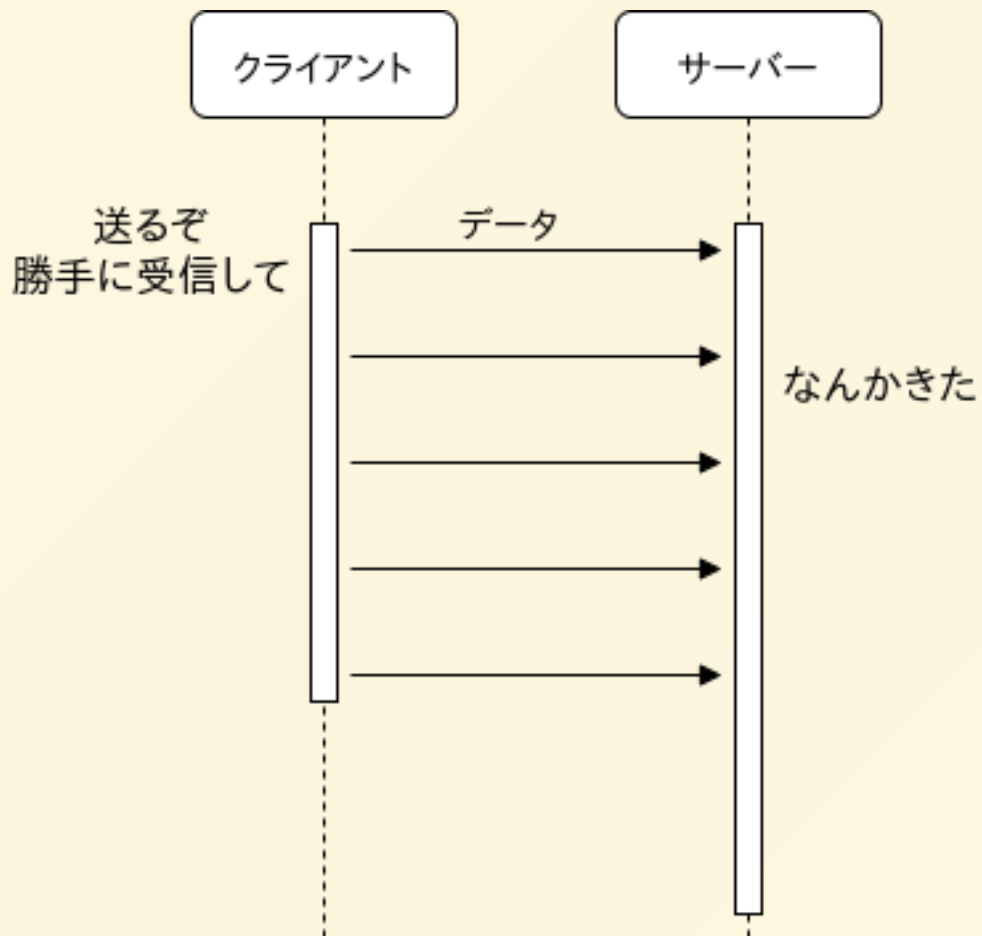
階層	担当	プロトコル例
アプリケーション層	アプリ	<b>HTTP</b> , TLS, SMTP, DNS
トランスポート層	OS	<b>TCP</b> , UDP
インターネット層	OS	IP (IPv4、IPv6)
ネットワークインターフェイス層	ドライバ	Ethernet, Wifi, PPP

# TCP



- 送受信の通信規約
- コネクションを繋げて通信
- データロスを検知し、再送  
(データの到着を保障)
- 到着順序を保障
- 通信速度を考慮
- いろいろ機能付いてる  
高機能プロトコルTCP

# UDP

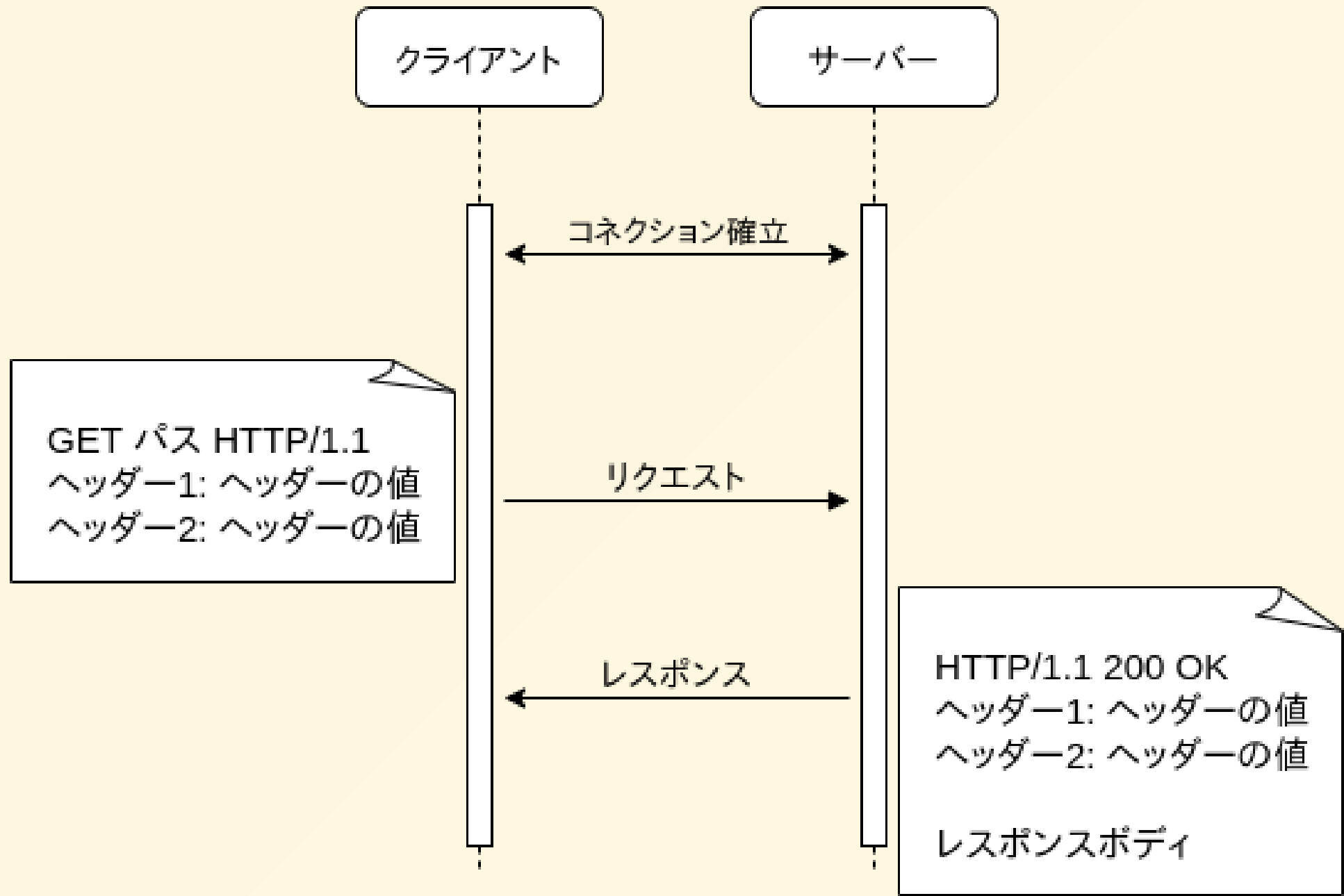


- コネクションレス  
繋がってる相手を管理しない
- 一方的にデータを送りつける
- データロスの検知なし
- 通信速度の制限なし
- 到着順序の管理なし
- 高機能なTCPと比べてかなりシンプルで早い

# HTTP通信

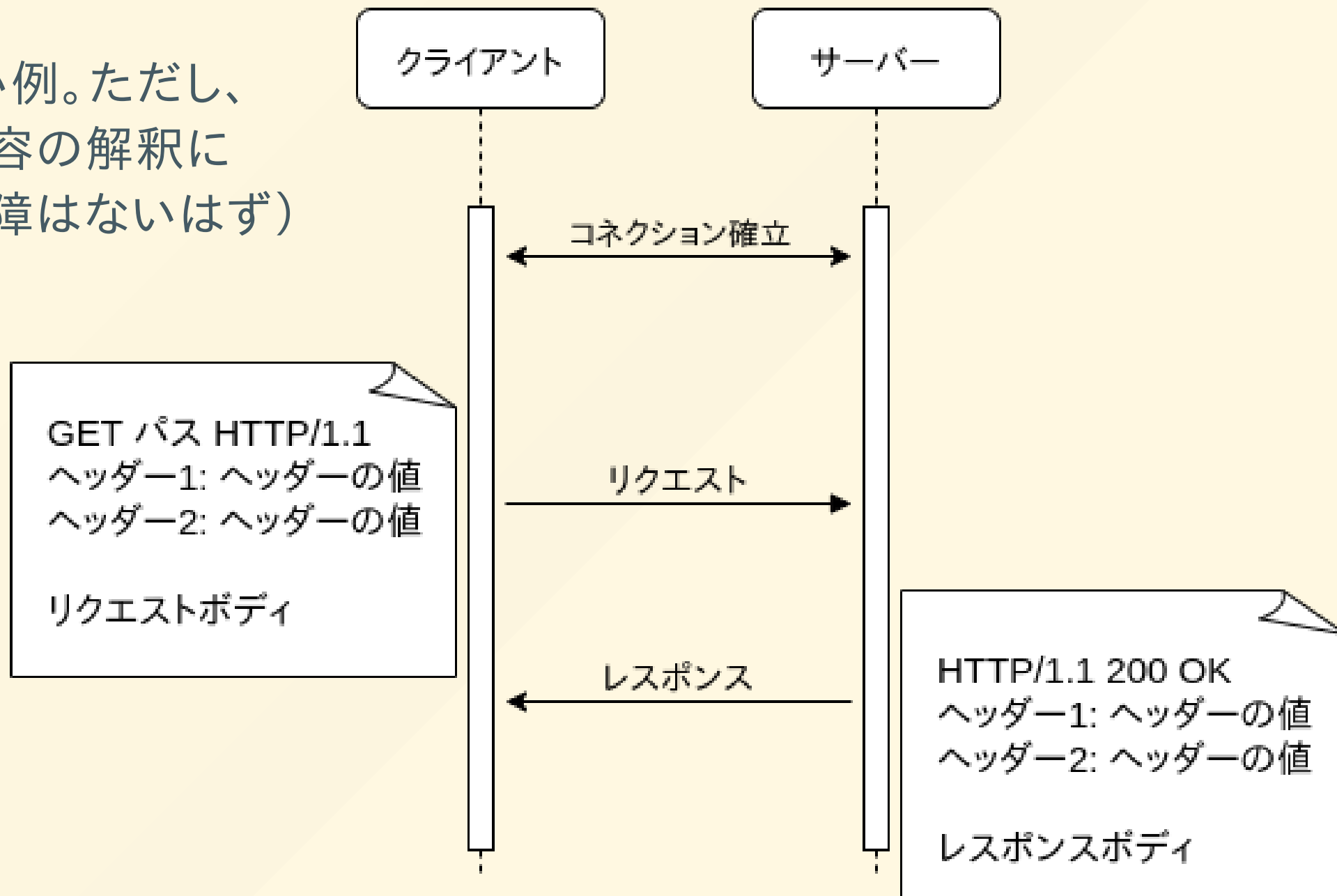
- HTTPはTCPの上に乗るプロトコル（今後登場するHTTP/3はUDP）
- HTTPリクエスト/レスポンスの書式、ヘッダーの項目などを定めている
- **データ通信のプロトコルではなく、  
送受信するデータをどう解釈するか定めたプロトコル**  
（インターネット間のデータ通信自体はTCP/IPで行われる）
  - どう解釈するかは使用するWebサーバーの実装次第

基本はHTTPリクエストを送り、HTTPレスポンスを受け取る1往復の通信。  
（リクエスト/レスポンスは書式通りに書かれた1つのファイルのようなもの）





(悪い例。ただし、  
内容の解釈に  
支障はないはず)



# HTTPリクエスト書式

```
メソッド パス HTTP/バージョン[改行]  
ヘッダー1: ヘッダーの値[改行]  
ヘッダー2: ヘッダーの値[改行]  
[改行]  
リクエストボディー(あれば)
```

- メソッド: GET、POST、PUT、DELETE、PATCH、HEAD、OPTION
- パス: `/`、`/index.html`、`/favicon.ico` などのパス
- ヘッダー: `Host`、`Accept`、`Connection`、`User-Agent` などの設定値
- リクエストボディー: POST、PUTなどでリクエストボディーが必要な場合

# HTTPリクエスト例

```
GET / HTTP/1.1
Host: example.com
User-Agent: curl/7.58.0
Accept: */*
```

```
POST / HTTP/1.1
Host: example.com
User-Agent: curl/7.58.0
Accept: */*
Content-Length: 20
Content-Type: application/json

{"message": "hello."}
```

# HTTPレスポンス書式

```
HTTP/バージョン ステータスコード(数値) ステータスコード(文字) [改行]  
ヘッダー1: ヘッダーの値[改行]  
ヘッダー2: ヘッダーの値[改行]  
[改行]  
サーバーレスポンス
```

- ステータスコード: `200 OK`、`400 Bad Request` などの決められたコード
- パス: `/`、`/index.html`、`/favicon.ico` などのパス
- ヘッダー: `Host`、`Accept`、`Connection`、`User-Agent` などの設定値
- リクエストボディ: POST、PUTなどでリクエストボディが必要な場合

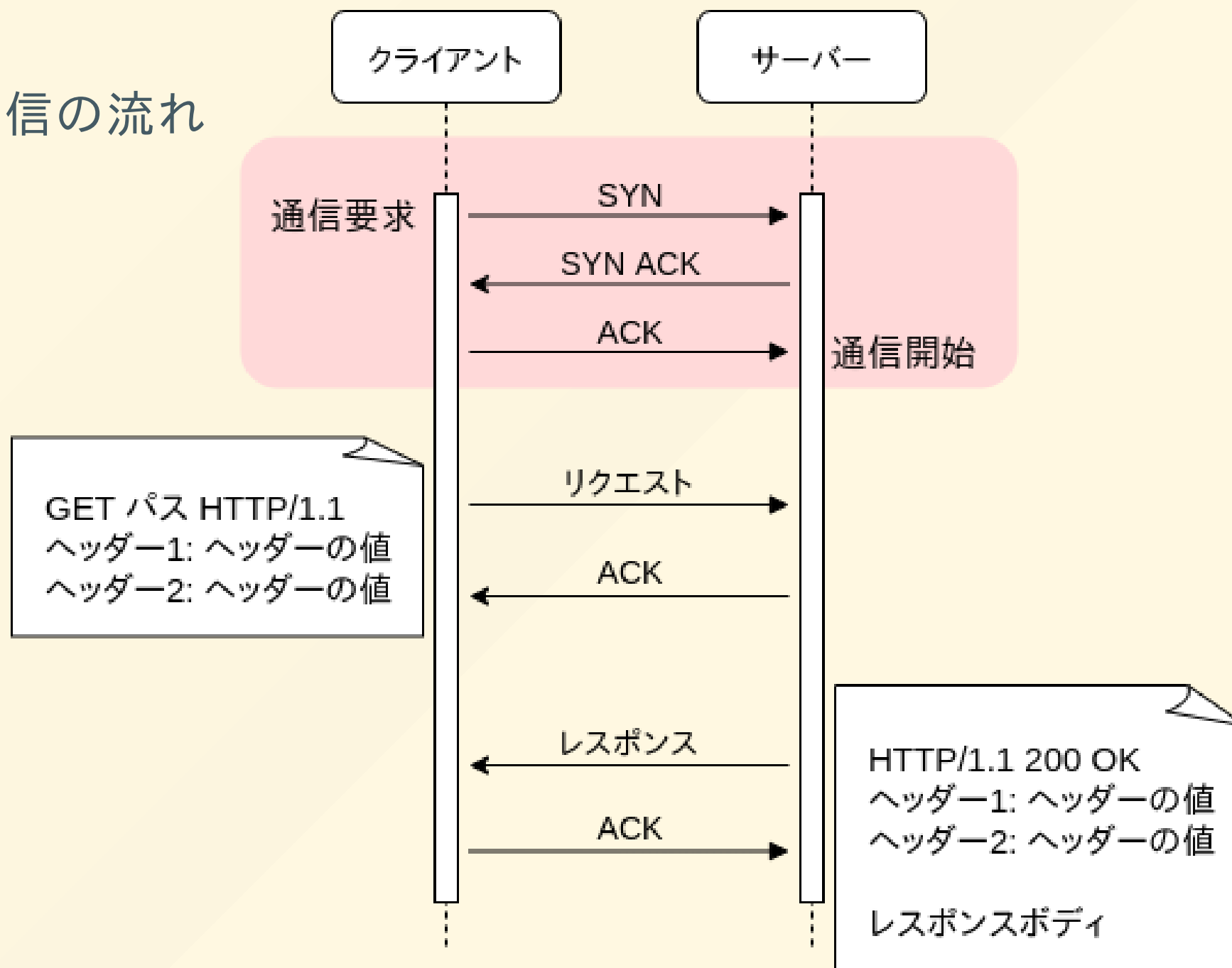
# HTTPレスポンス例

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Date: Fri, 01 Nov 2019 04:40:07 GMT
Content-Length: 1256
...

<!doctype html>
<html>
<head>
...
```

HTTPレスポンスの書式はGETやPOSTによって変化することがない。  
(サーバーレスポンスは多くの場合に存在するが、ない場合もある)

# HTTP通信の流れ



**もっとTCPを知る**

# !!何か切らないと...

長すぎる

- POSIX
- プロセス (プロセスディスクリプター)
- ファイルディスクリプター
- システムコール、シグナル
- ソケット通信



まとめ