

# Web講習会2021 ワールドワイドウェブ発展

第1回: Webの実装パターンの歴史

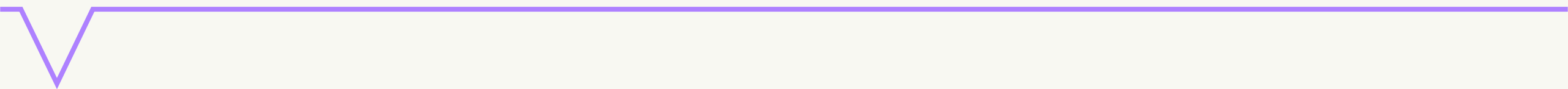


Arthur

# 今回の目標

- 変化し続けているWebの実装パターンの歴史を追う
- 本講習会の最終目標として扱うアーキテクチャの概観を知る

# 静的Webと動的Web



# Webサーバの大別

## 静的Webサーバ

サーバに保存されたファイルをクライアントにそのまま送る

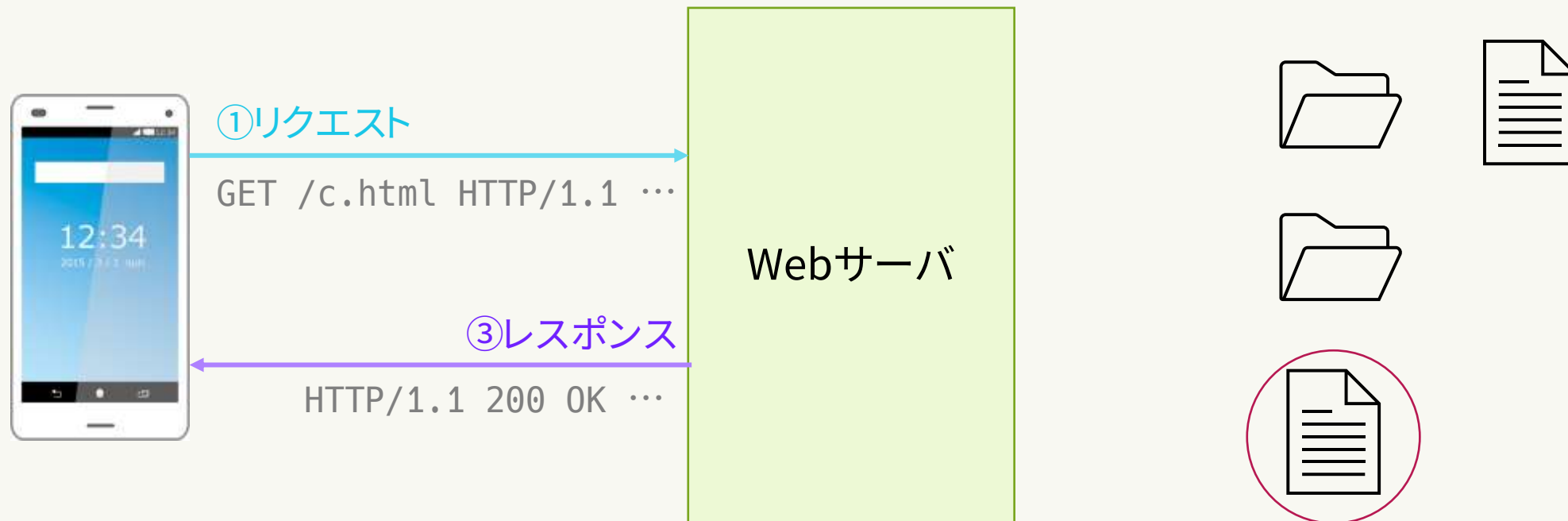
## 動的Webサーバ

リクエストのたびにファイルをアプリケーションで生成して送る

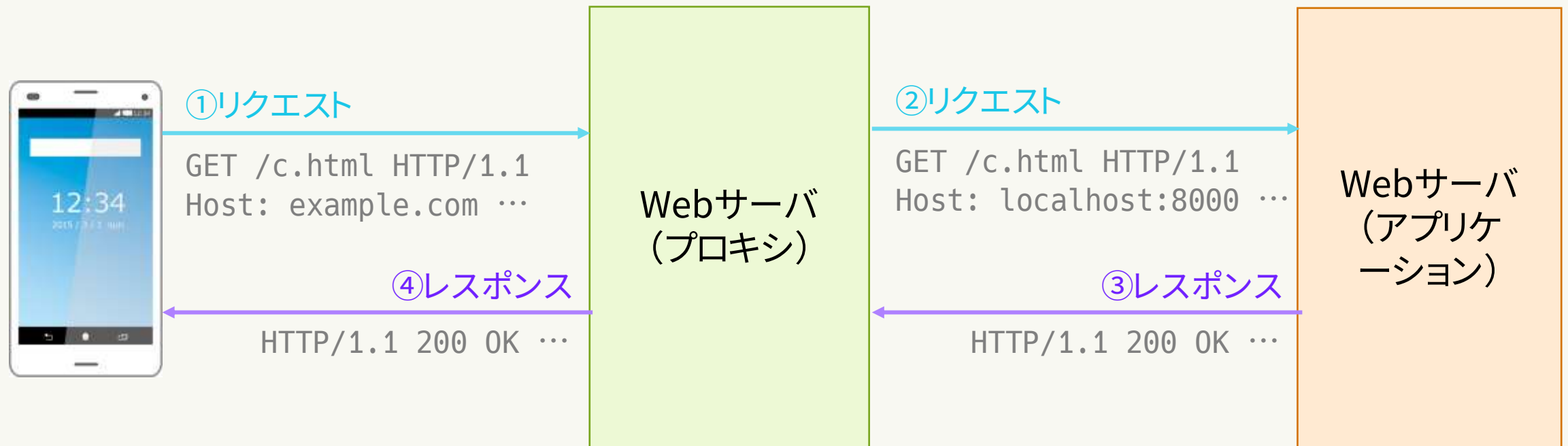
[例] ログイン後自分の名前が表示されるサイト

# 静的Webサーバの仕組み

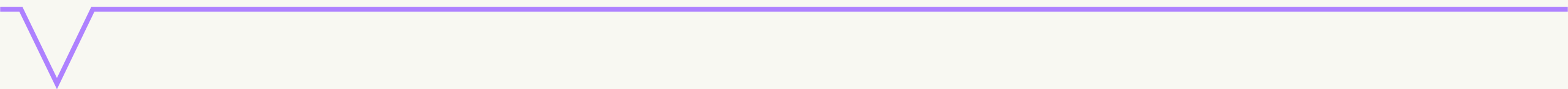
②リクエストの内容(URLなど)から、目的のファイルを探し出す



# 動的Webサーバの仕組み



# Webの実装パターンの歴史



# 純粋な静的Web

各ページごとhtmlファイルを作る

Webサーバは各パスを静的ファイルにマッピング



/20201201.html



/20200727.html



/20171222.html



# 問題点

ヘッダーなどのマークアップは各ページ共通  
コピペする? 😞



/20201201.html



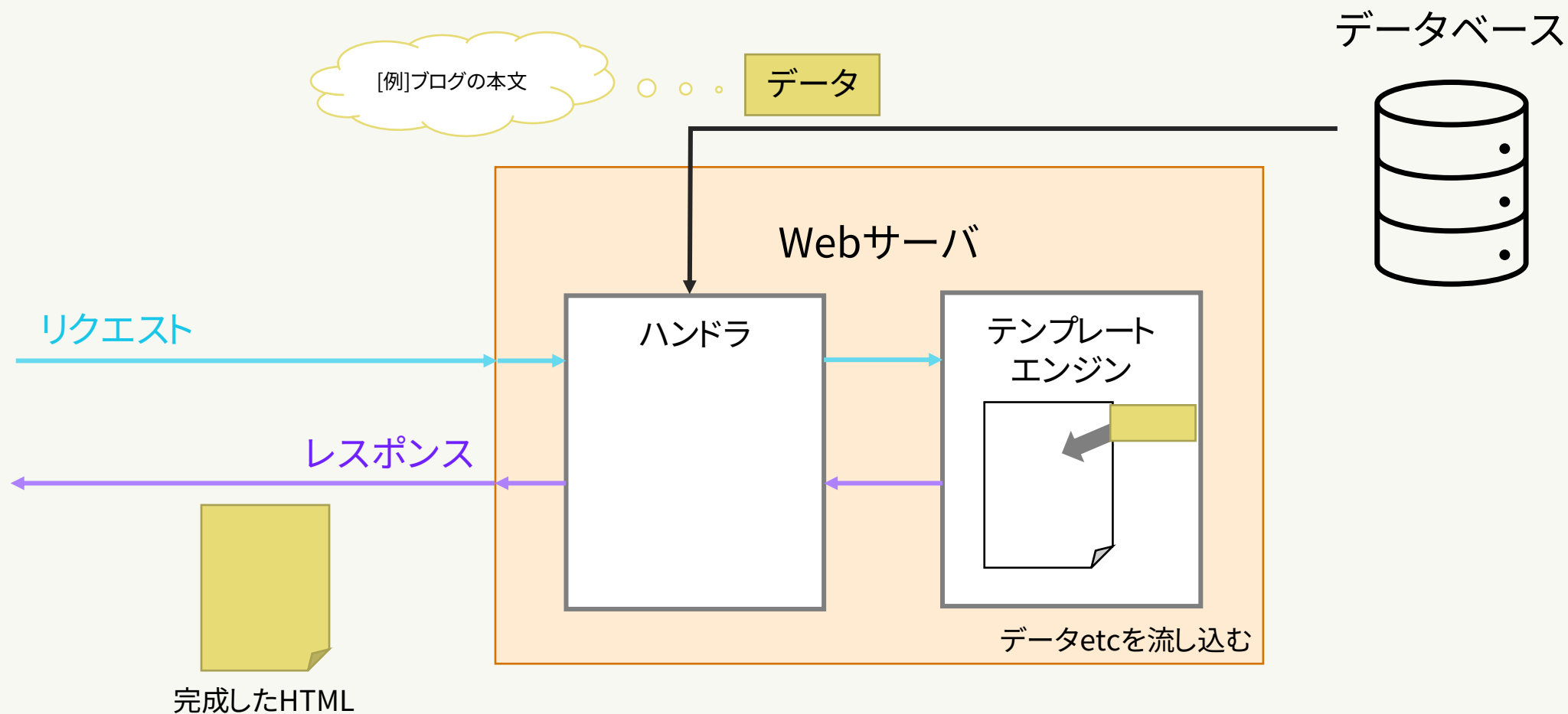
/20200727.html



/20171222.html

# SST: Server-Side Templating

テンプレートエンジンで生成したHTMLの内容を返す



# SSTの問題点

サーバ側でやる手続きが多い

- DBとのアクセス
- テンプレートエンジンによるHTMLの生成



同時アクセスに弱い

動的サーバを持つコストも大きい

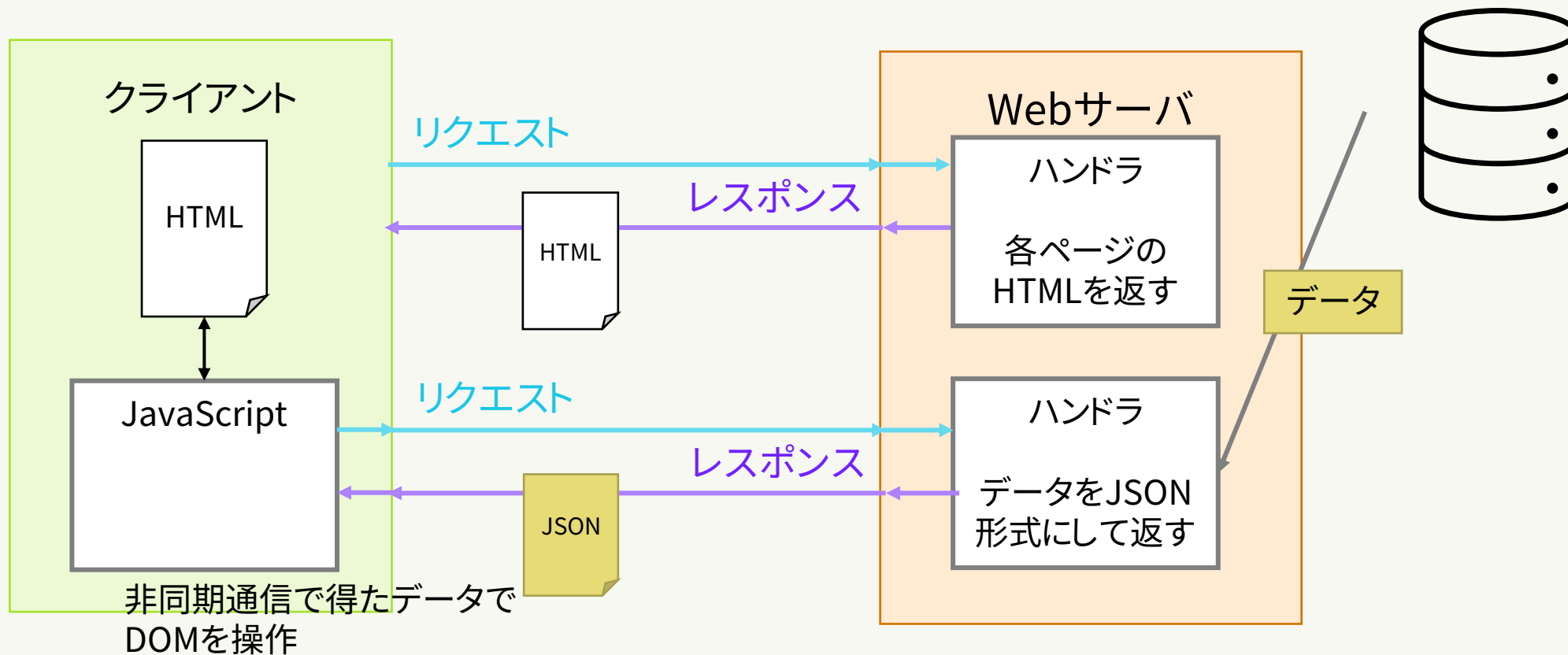
(静的Webなら無料でホスティングできることも多い)

# AJAX: Asynchronous JavaScript + XML

**非同期通信:** ブラウザを通常通り動作させたまま裏で通信

ページ遷移なく表示する情報を更新できる

データベース



# AJAXの特徴

jQuery … 前述の構成が一番流行っていた時代のライブラリ

## メリット

- インタラクティブなWebサイトが作れる
- 追加の通信はデータのみなので、通信量の削減ができる

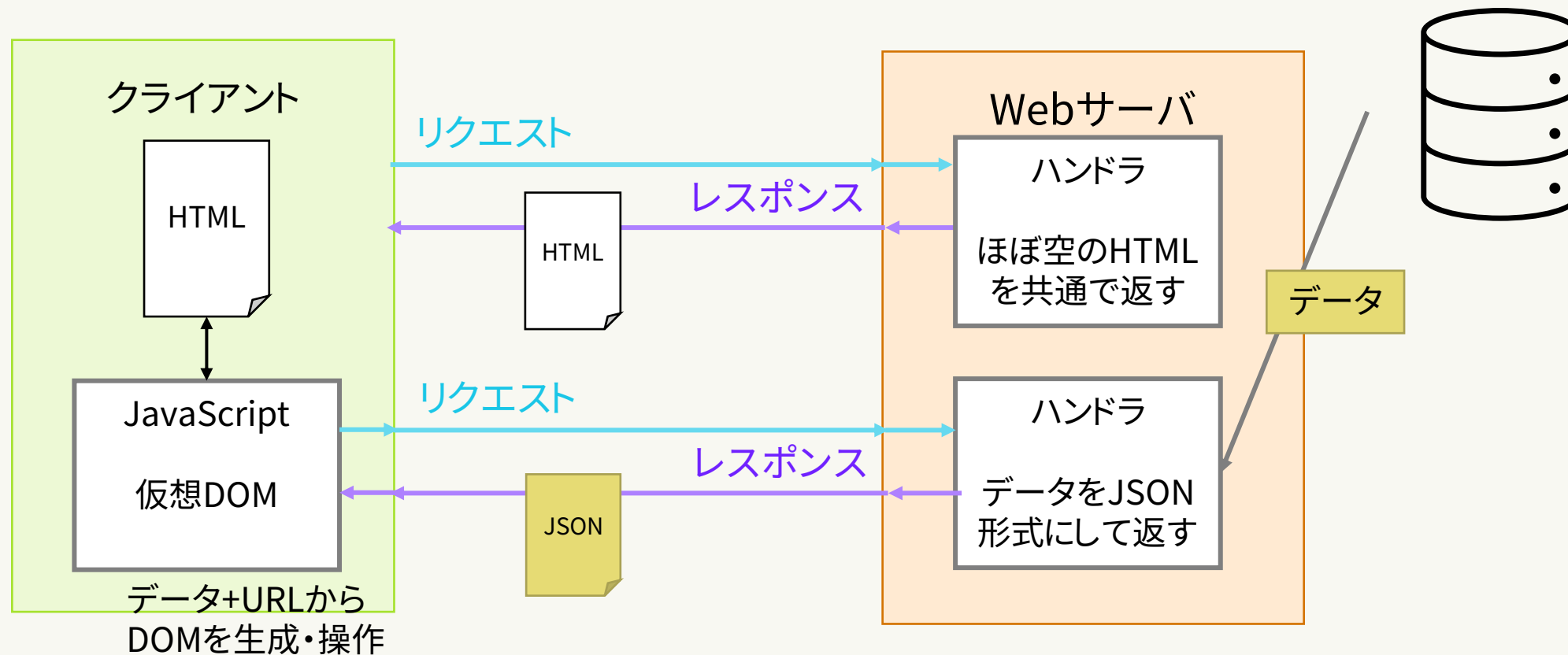
## デメリット

- 高機能になればなるほど、手続きが複雑になっていく  
DOMの直接変更が増えると収拾がつかない

# CSR: Client-Side Rendering

ルーティング (URLごとのページ切り替え) もJavaScriptにやらせる  
ページ遷移なく表示する情報を更新できる

データベース



# サーバサイドとフロントエンドの分離

CSRは別名 **SPA**: Single Page Applicationとも呼ばれる  
どのURLでもレスポンスのHTMLは単一だから

## サーバサイド

- データの操作と、JSON形式のデータ返却 → Web API  
(詳細は第4回⇒)

## フロントエンド

- URLごとに表示するViewを決める(ルーティング)
- JSONで受け取ったデータ→仮想DOMの紐付け

# CSRを実現するライブラリ

## 仮想DOM

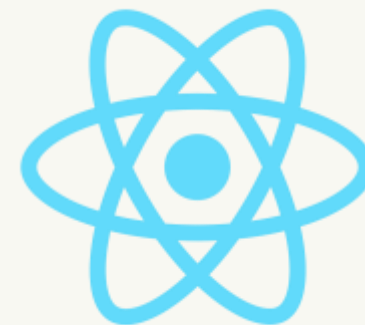
クライアント上で動的にDOMツリーを生成・編集するための仮想的なDOM

→ 直接DOMを触るより扱いやすい

ライブラリによって実際のDOMと同期するように更新

→ 差分更新の最適化が可能

- Vue.js
- React (第5・6回⇒)





# CSRの特徴

## メリット

- AJAXと同等以上のユーザ体験
- 仮想DOMによる、コードの実行速度の両立
- サーバサイドとフロントエンドの完全な分離（開発の分担）

## デメリット

- 初期レンダリングが遅い  
単体の比較的大きなサイズのJavaScriptを読み込み+実行まで真っ白
- OGPを各ページごとuniqueにできない

# [補足] OGP

## OGP: Open Graph Protocol

Webページの情報を検索エンジンやサービスに伝える

head内に特定のmetaタグを書いて伝える

[例] `<meta property="og:image" content=".....">`

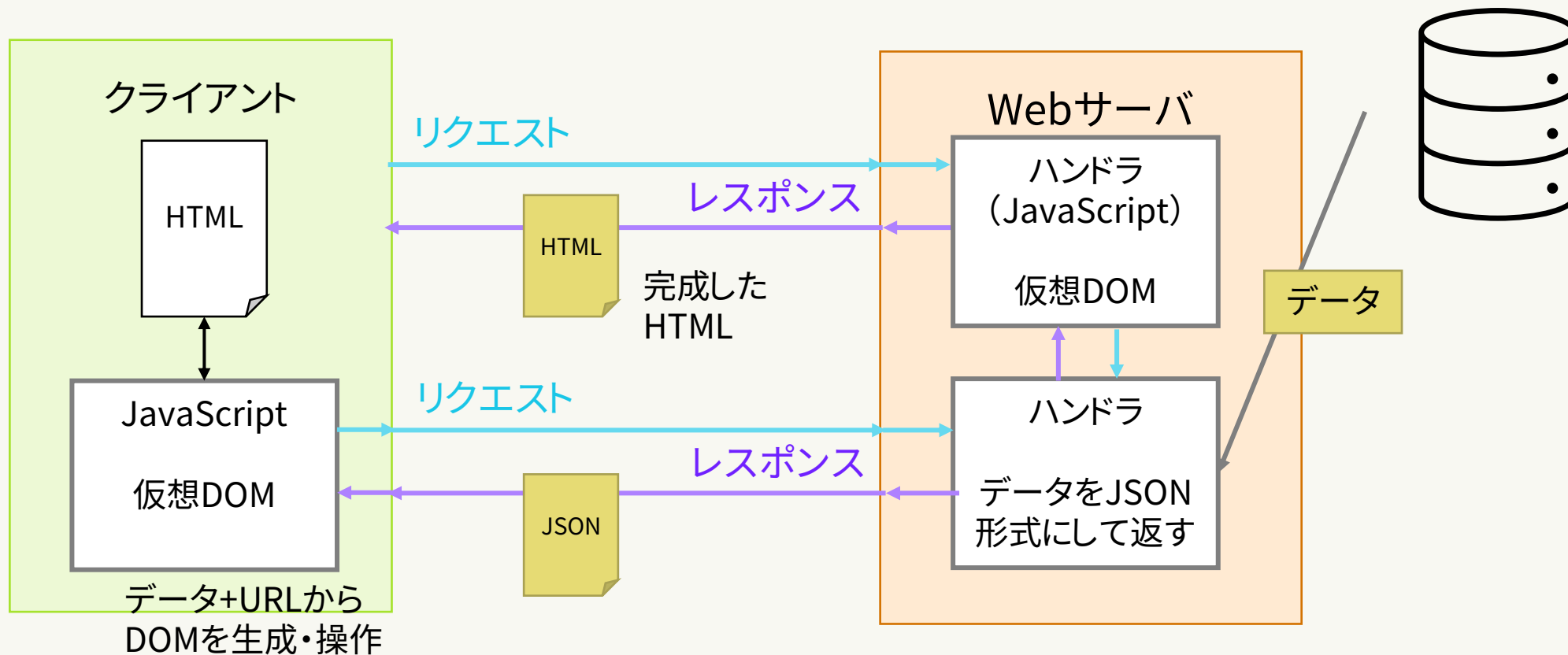
各サービスは、JavaScriptによって動的に書き換えたmetaタグを検知しない



# SSR: Server-Side Rendering

SSTとCSRを足して割ったような構成

サーバでもブラウザと同じように仮想DOMからHTMLを生成 データベース



# SSRの特徴

## メリット

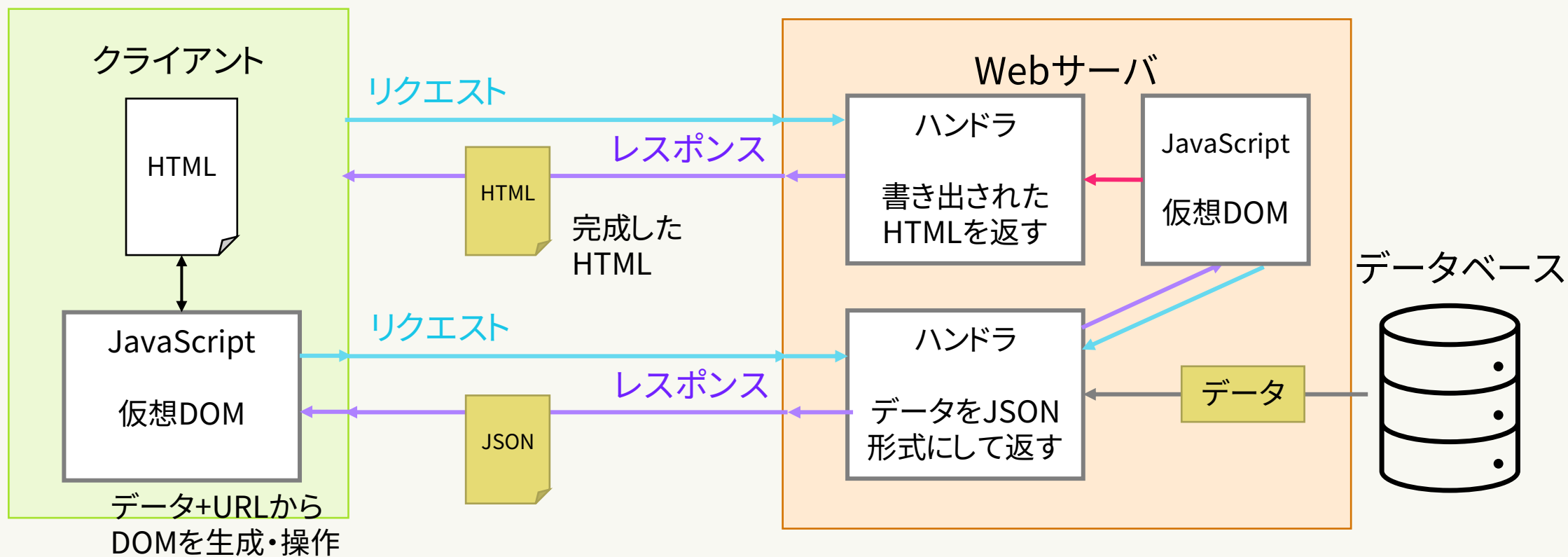
- CSRのパフォーマンスを維持しつつ、初期レンダリングが高速に
- OGPをページごと変えることが可能に

## デメリット

- SST同様、同時アクセスを捌ききれないことも
- 動的サーバを持つコスト
- サーバでもクライアントでも動くJavaScriptを書くのに一癖あり

# SSG: Static Site Generation

SSRの結果をあらかじめ静的ファイルに書き出す(ビルド)



# SSGの特徴

## メリット

- HTML生成のオーバーヘッドがない分、SSRより高速
- HTMLを配信するサーバの部分は静的でOK  
サーバ運用コストを減らせる

## デメリット

- ページ数が多いとビルドに時間がかかる  
数千とページ数がある場合には向いていない
- ビルド環境でもクライアントでも動くJavaScriptを書くのに一癖あり

# WWW発展の最終目標

## SSGを用いた高速なWebサイトの作成

Web講習会のWebサイトもSSGを利用している

<https://arthur1.github.io/tutorial-web-2021/>



# 次回予告

## Node.jsとES2015

- ES5とES2015の違い
- パッケージ管理ツール
- Promise API
- JavaScriptからHTTP通信

## 次回までにやること: Node.jsの実行環境を整える

環境の整え方は自由ですが、おすすめの方法をWebサイトに掲載します

Node.js v16系をインストールしてください