フロントエンド基礎 #1

Tomoyuki Takata@DeNA

勉強会の目的

開発メンバー全員がフロントエンドに対する意識を高め、
結果それがプロダクトの品質を向上させる

勉強会の流れ

1.ブラウザの仕組み、レンダリングの仕組み

2.chrome dev toolsの使い方

3.pure javascript

4.pure javascript2

5.css設計 SMACSS

6.パフォーマンスチューニング

勉強会の流れ

1.ブラウザの仕組み、レンダリングの仕組み

2.chrome dev toolsの使い方

3.pure javascript

4.pure javascript2

5.css設計 SMACSS

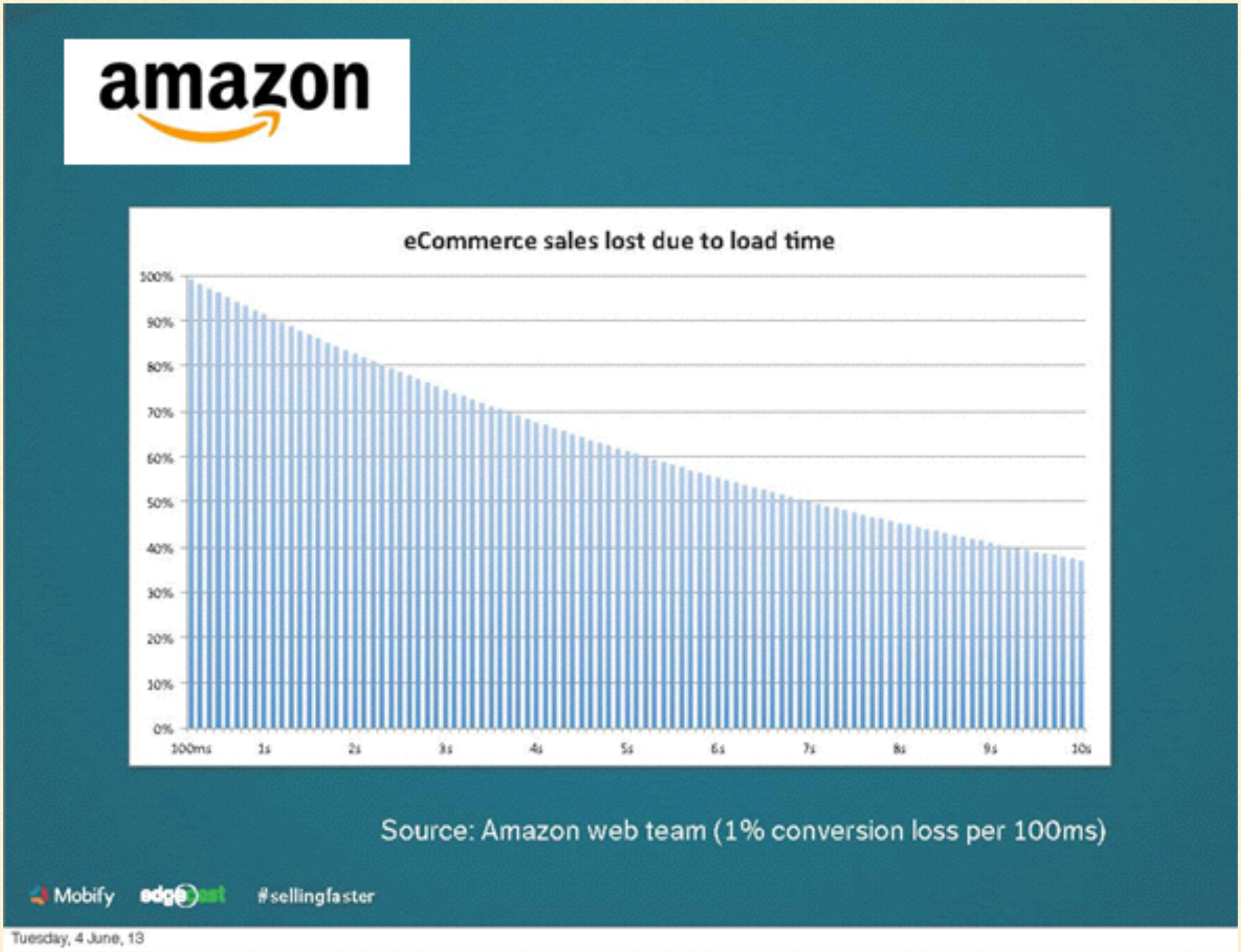
6.パフォーマンスチューニング

モバイルWEBにおけるパフォーマンスの重要性

“画面表示までに3秒以上かかった場合の離脱率は実に57%
5秒以上かかると74%の人が離脱する”

–Google

Amazonの調査では0.1秒反応が遅くなると売上が1%減る



WEBサイトの表示高速化メリット

- コンバージョン率の最適化
- UXの向上
- 直帰率の減少
- 運用費の節約
- etc

WEBサイトの表示速度を遅くする要因

- レイテンシー（遅延）
- ペイロード（コンテンツサイズ）
- キャッシュ（ブラウザキャッシュ）
- レンダリング（ブラウザ上の描画プロセス）
- etc

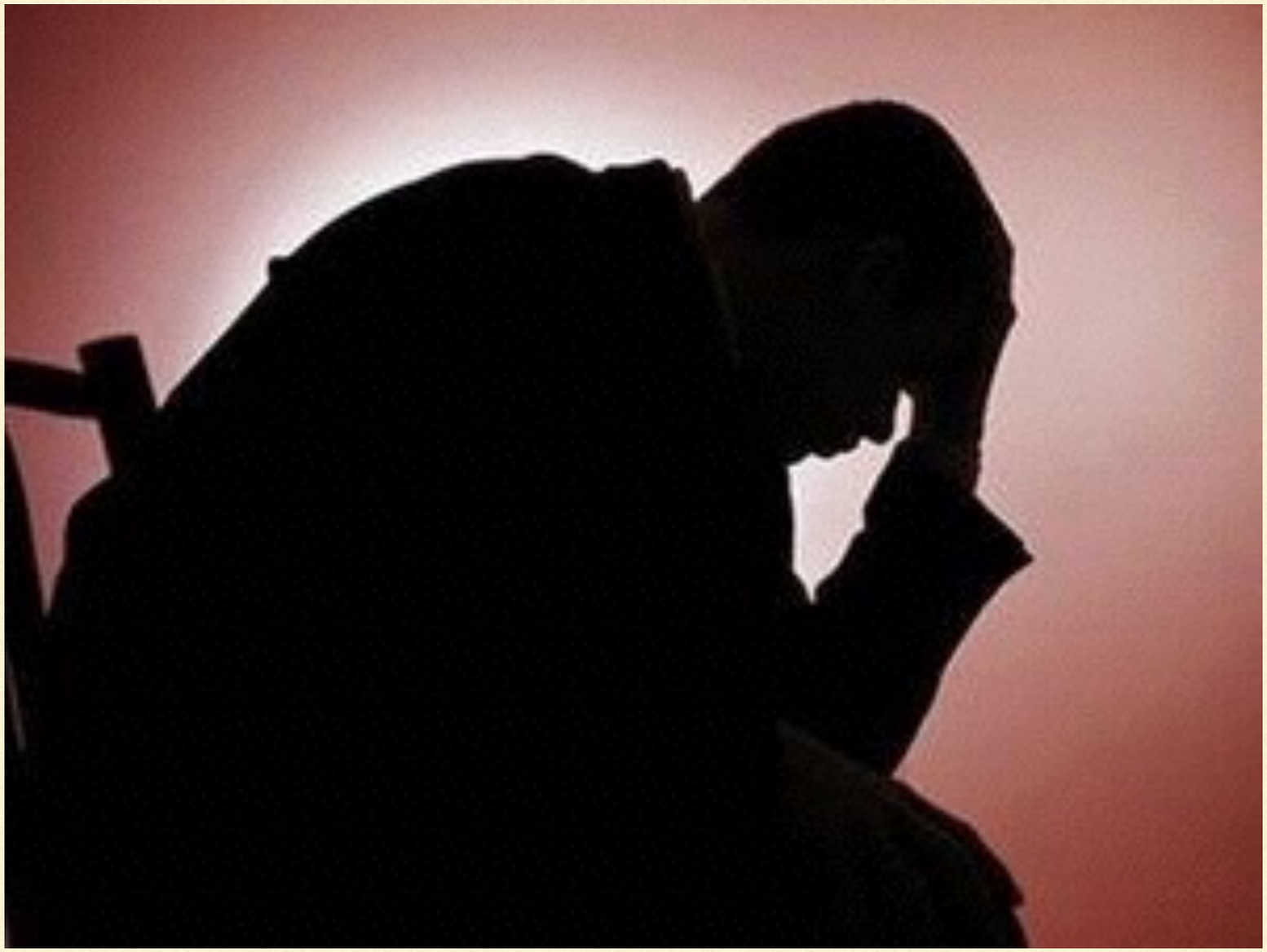
表示待ち時間＝通信量＝サーバーコスト＝ユーザーの携帯料金

パフォーマンスを最適化するステップ

- HTTPリクエストを減らす
- ブラウザキャッシュを利用する
- コンテンツサイズを最適化する
- GZIP圧縮を使う
- レンダリングコストを減らす
- etc

やることは分かった。が、、

どこから取りかかればいいんだ！



便利な試験ツールあります！

パフォーマンスを最適化するステップ

- HTTPリクエストを減らす
- ブラウザキャッシュを利用する
- コンテンツサイズを最適化する
- GZIP圧縮を使う
- レンダリングコストを減らす
- etc

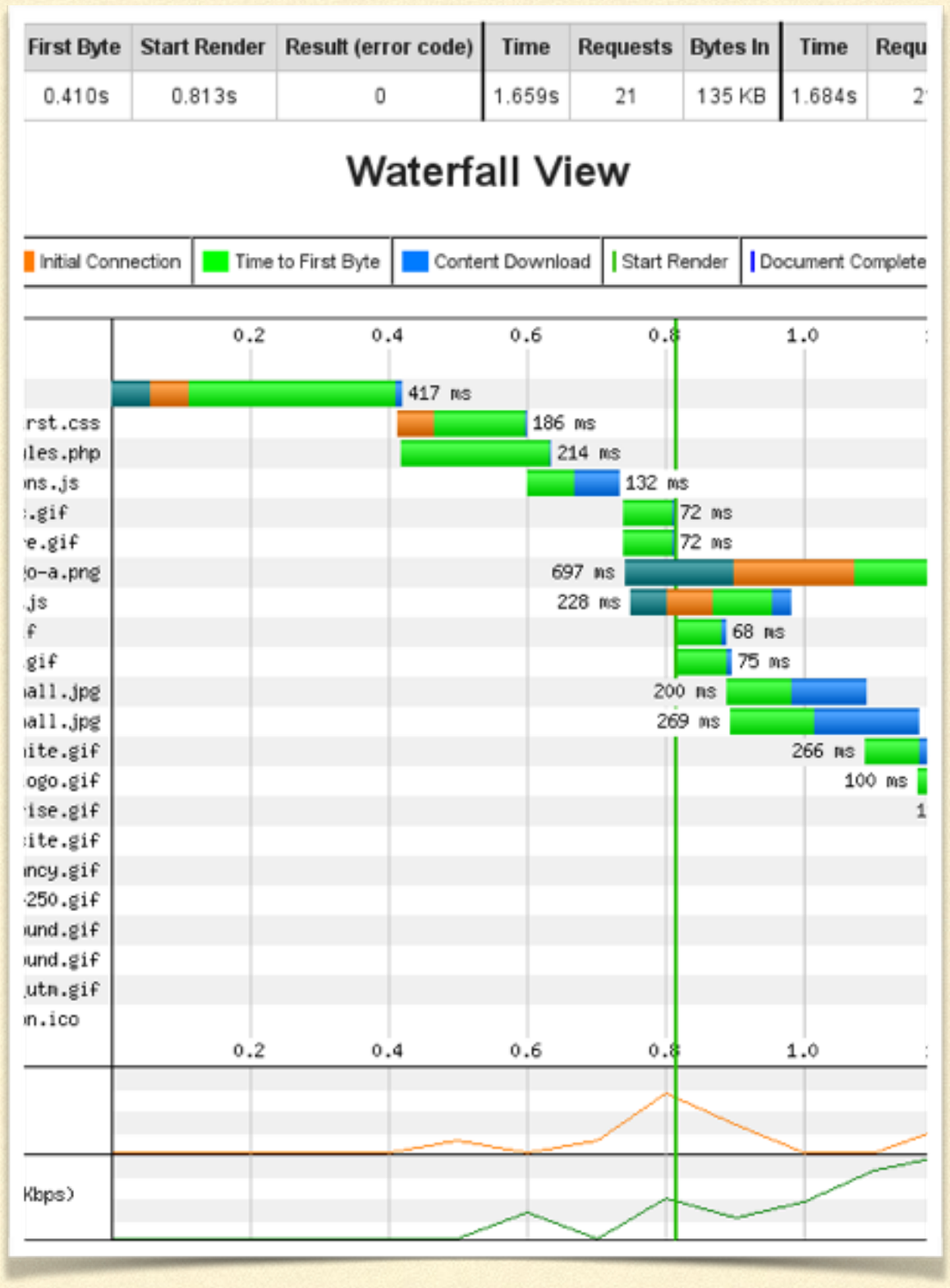
PAGESPEED INSIGHTS

https://developers.google.com/speed/pagespeed/insights/?hl=ja

デモ

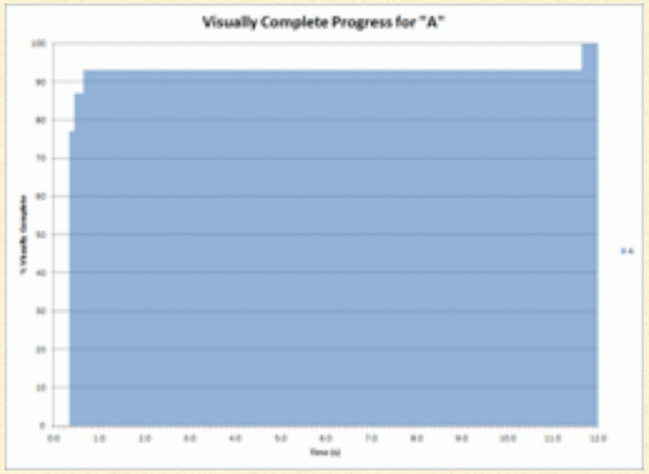
WEBPAGE TEST

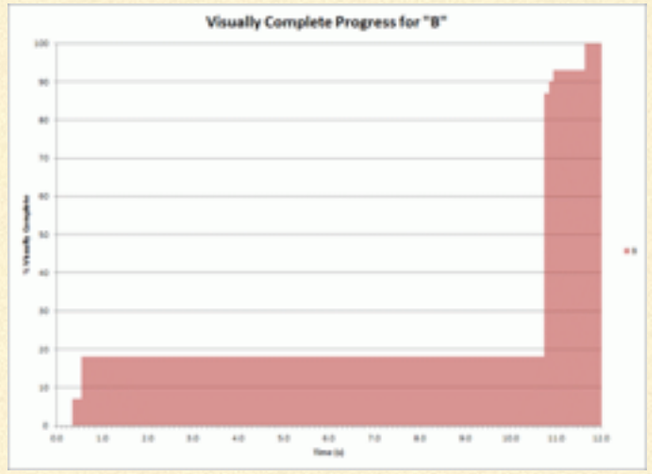
http://www.webpagetest.org/

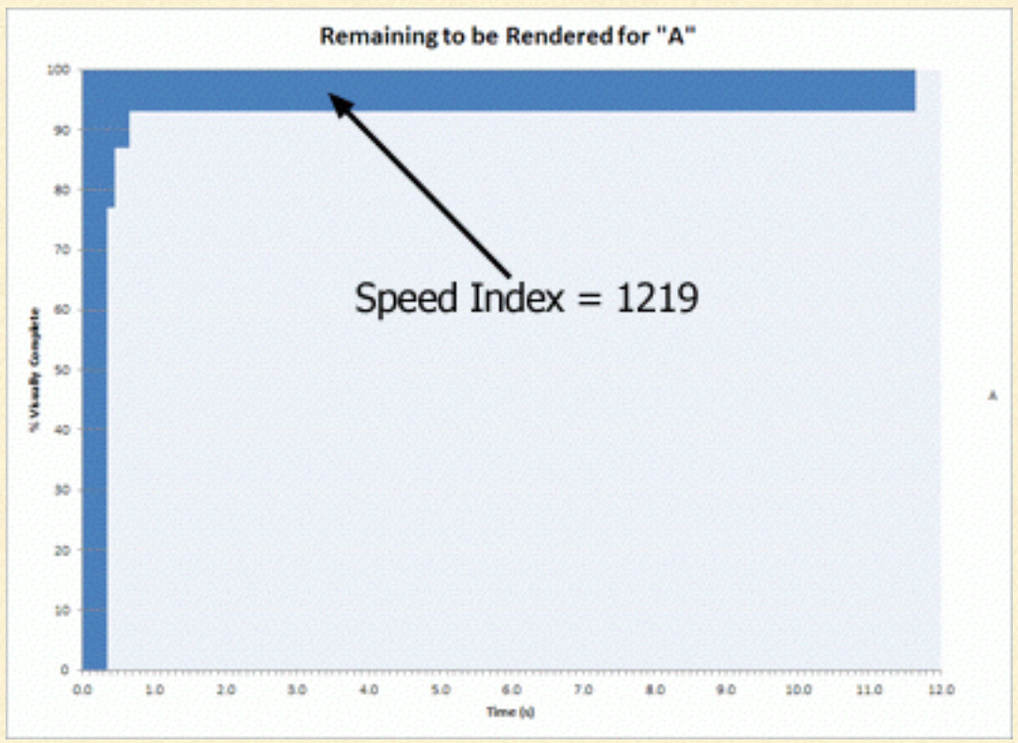


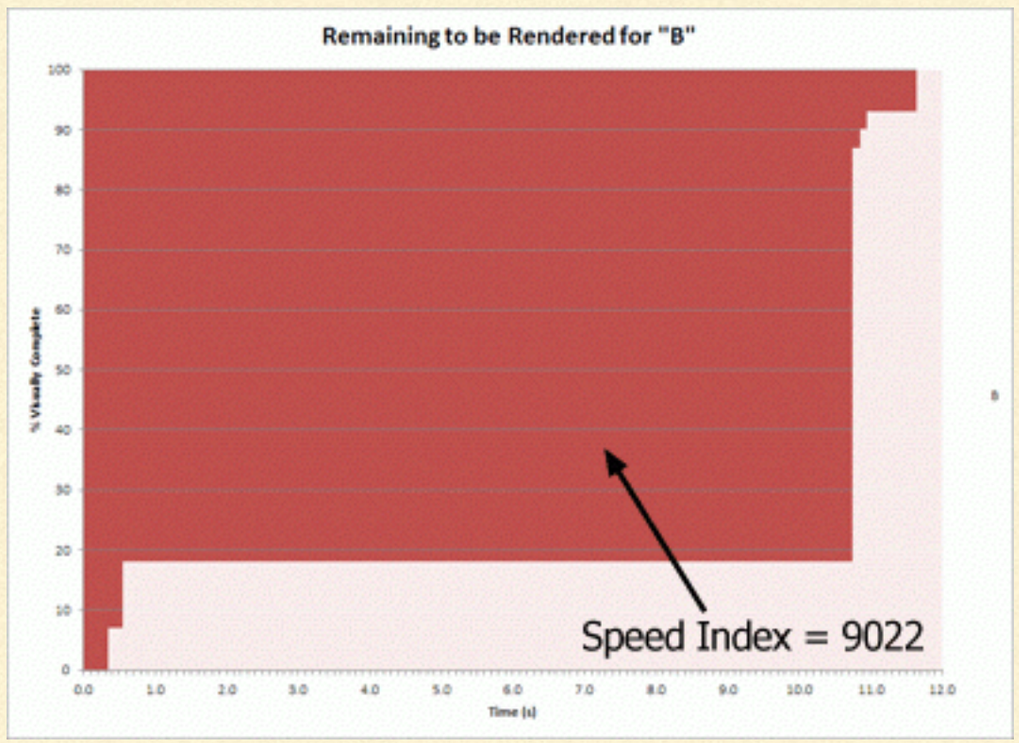
デモ

SPEED INDEX


Visually Complete Progress for "A"

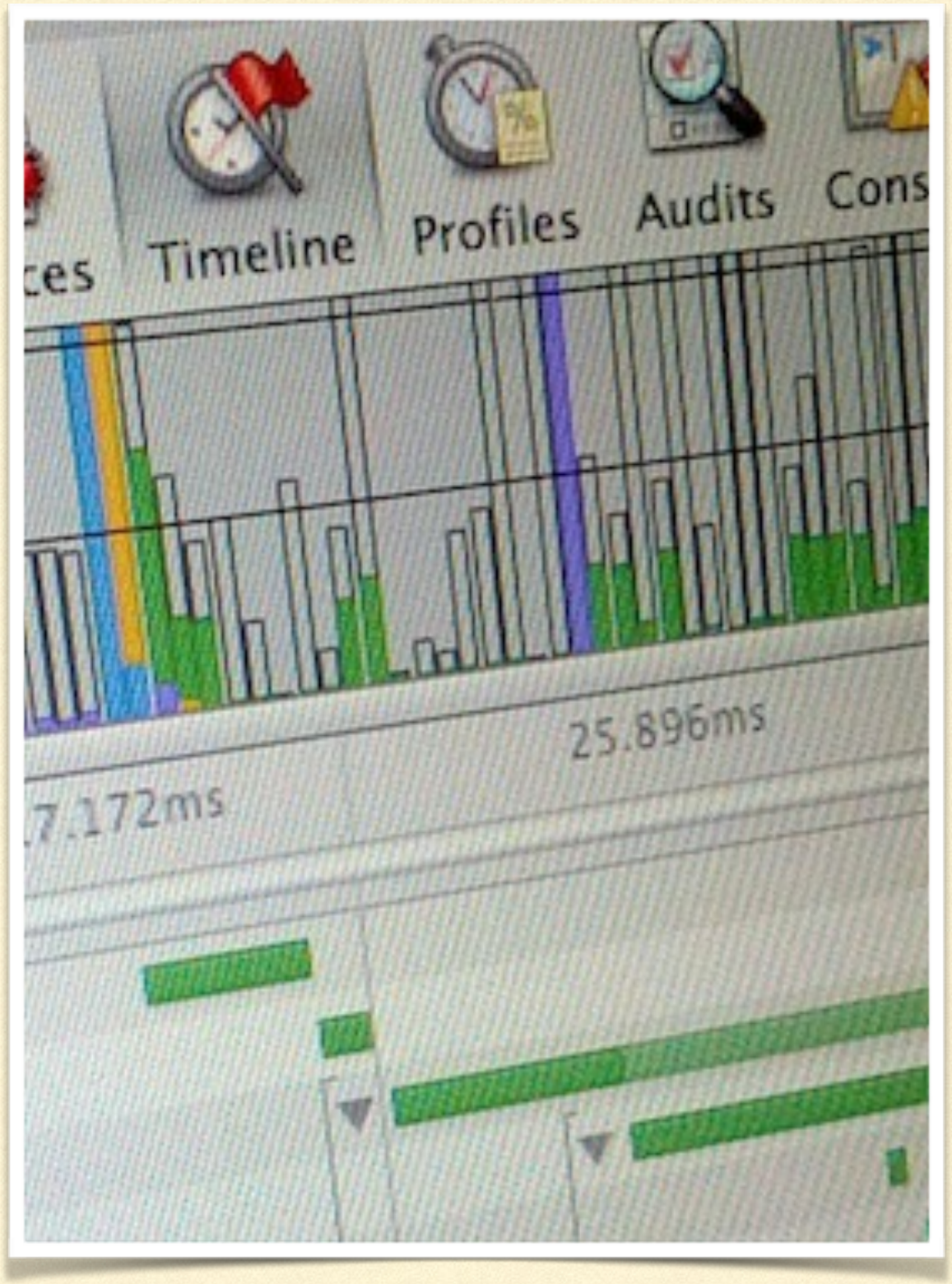

Visually Complete Progress for "B"


Remaining to be Rendered for "A"
Speed Index = 1219
% Visually Complete
Time (s)
A


Remaining to be Rendered for "B"
Speed Index = 9022
% Visually Complete
Time (s)
B

CHROME DEVTOOLS

・2回目で詳しく！



パフォーマンスを最適化するステップ

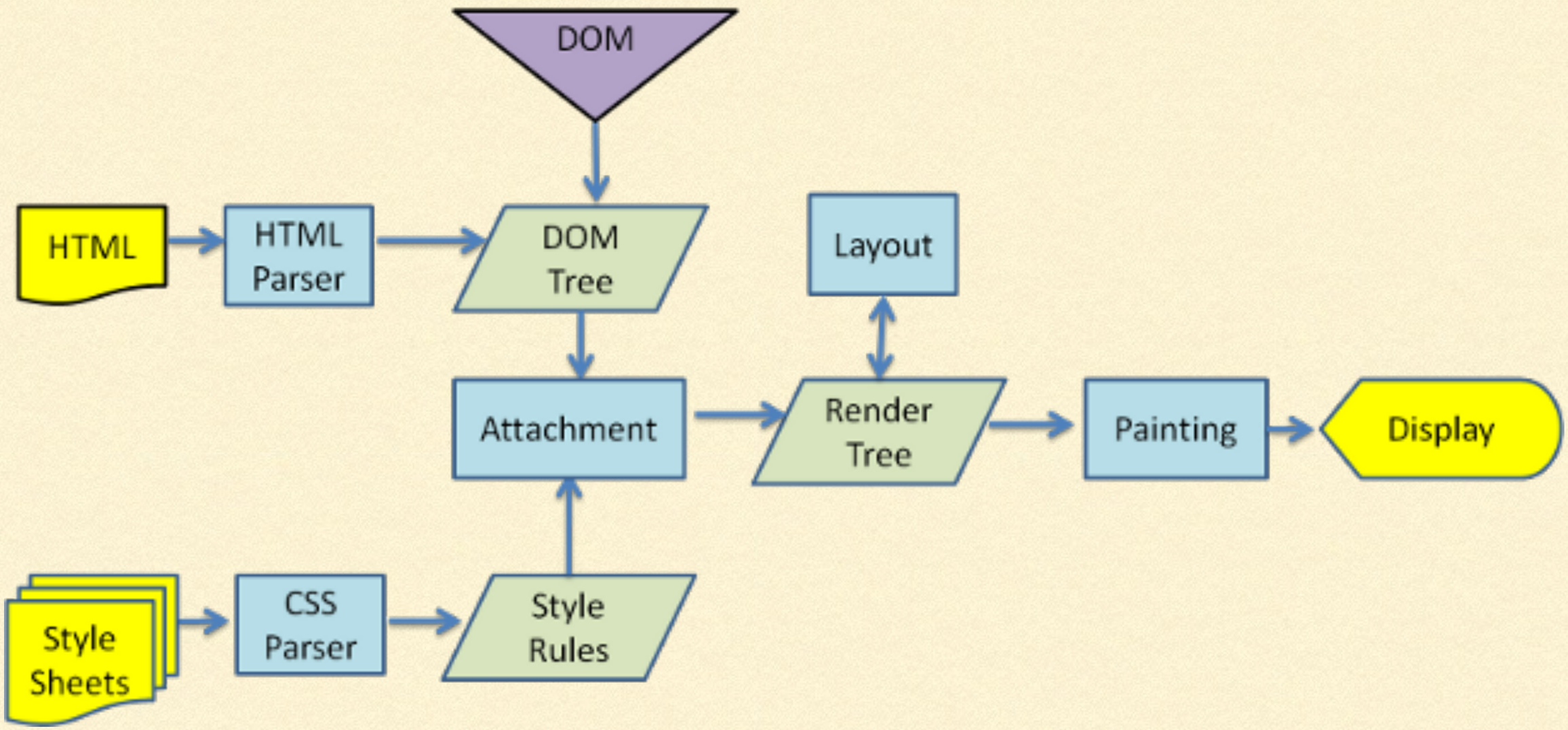
- HTTPリクエストを減らす
- ブラウザキャッシュを利用する
- コンテンツサイズを最適化する
- GZIP圧縮を使う
- レンダリングコストを減らす
- etc

ブラウザの仕組み

ブラウザの種類

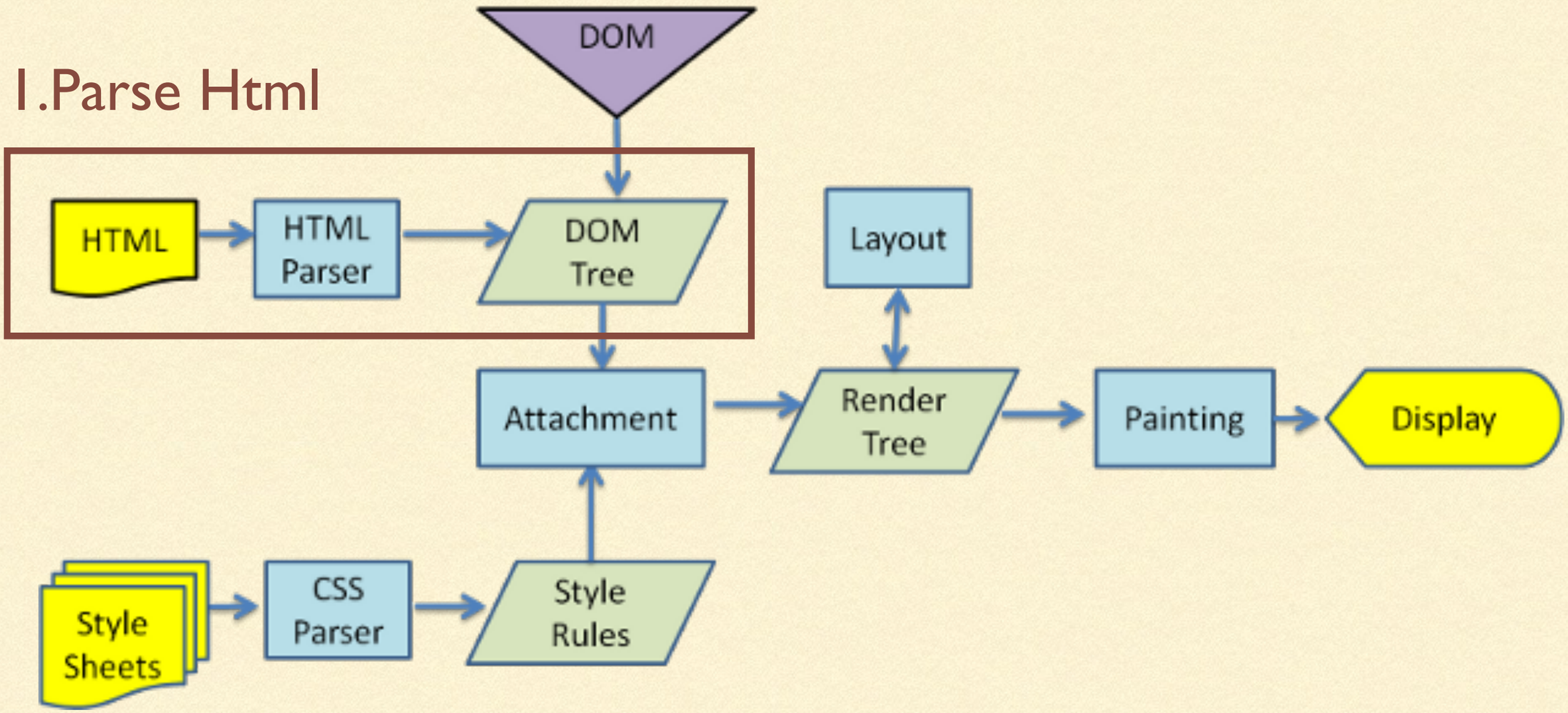
ブラウザの種類

レンダリングフロー（WEBKIT）



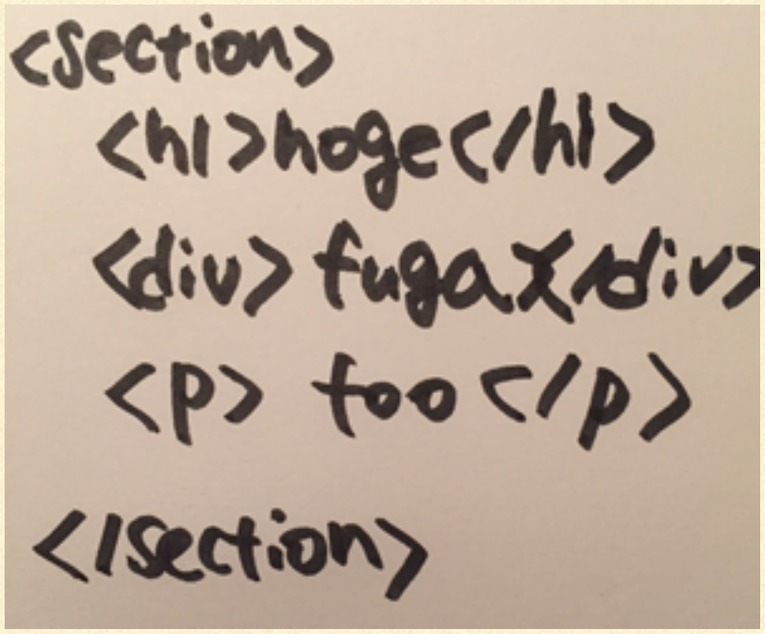
レンダリングフロー（WEBKIT）

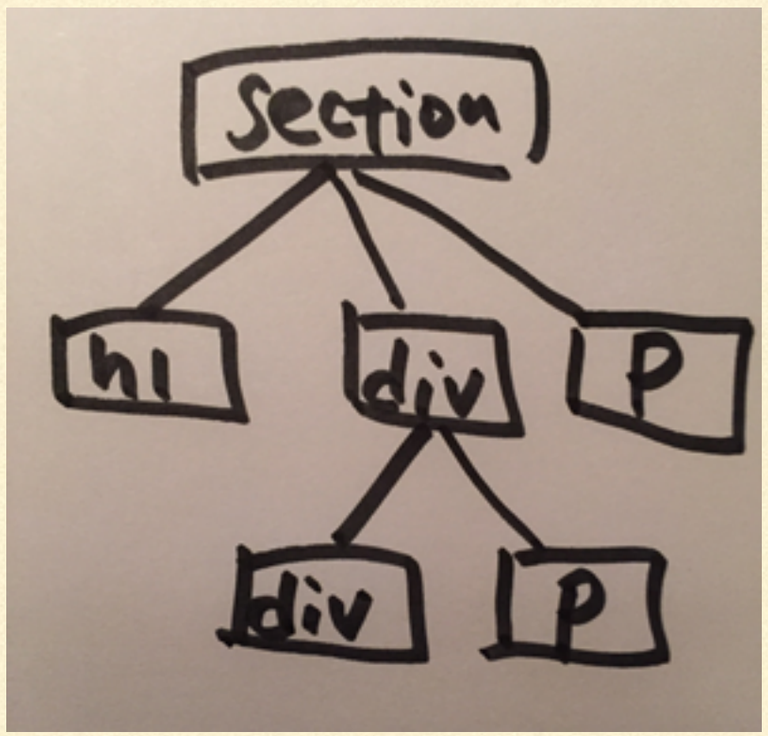
1.Parse Html



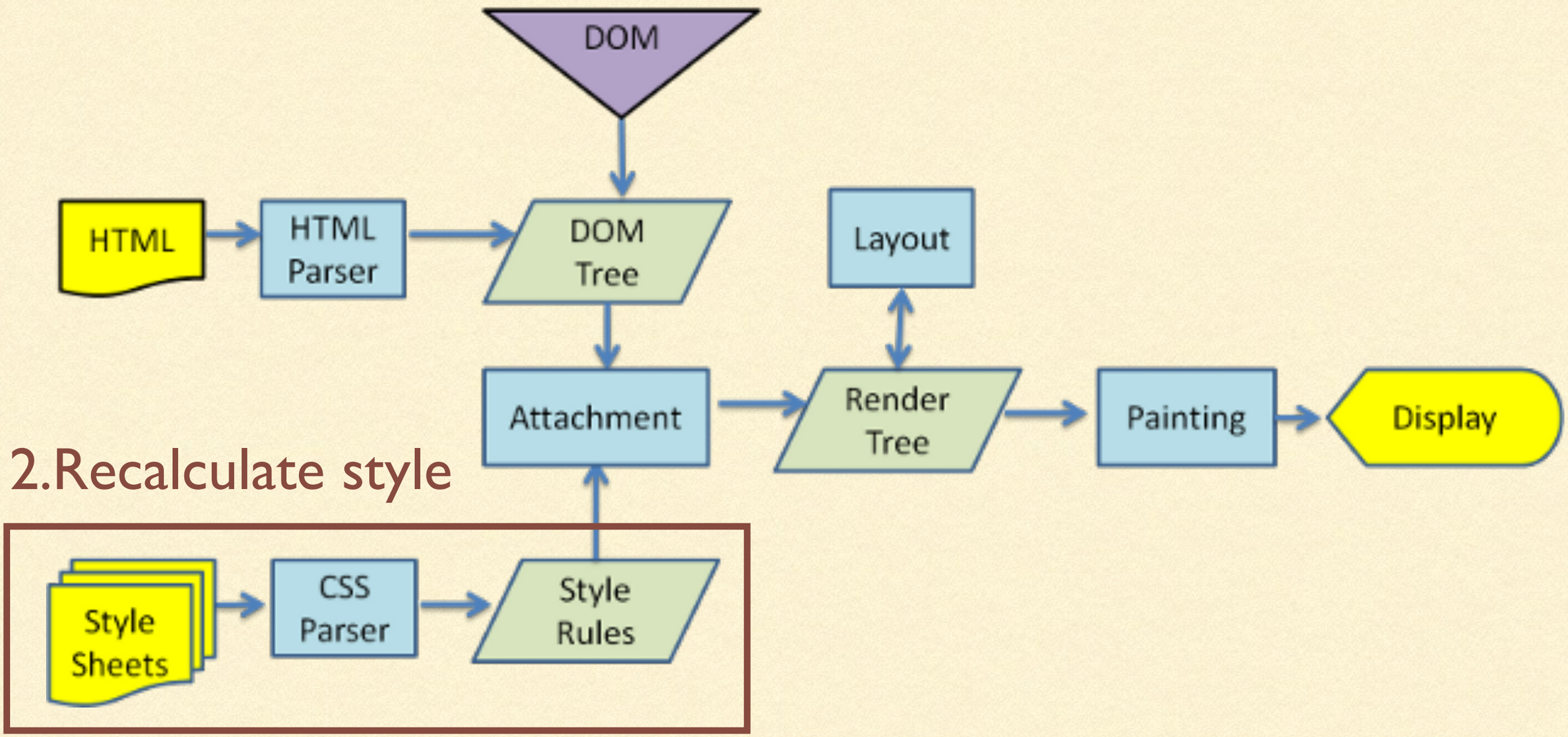
Parse HTML

Htmlテキストを DOM ツリーに変換する



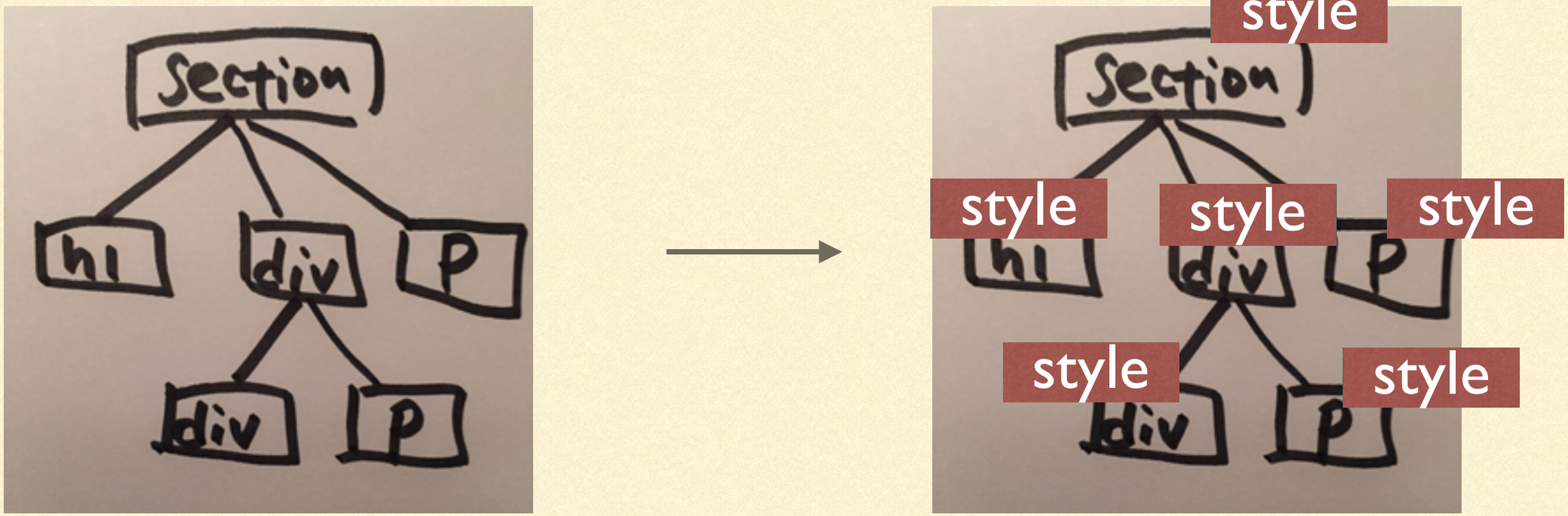


レンダリングフロー（WEBKIT）

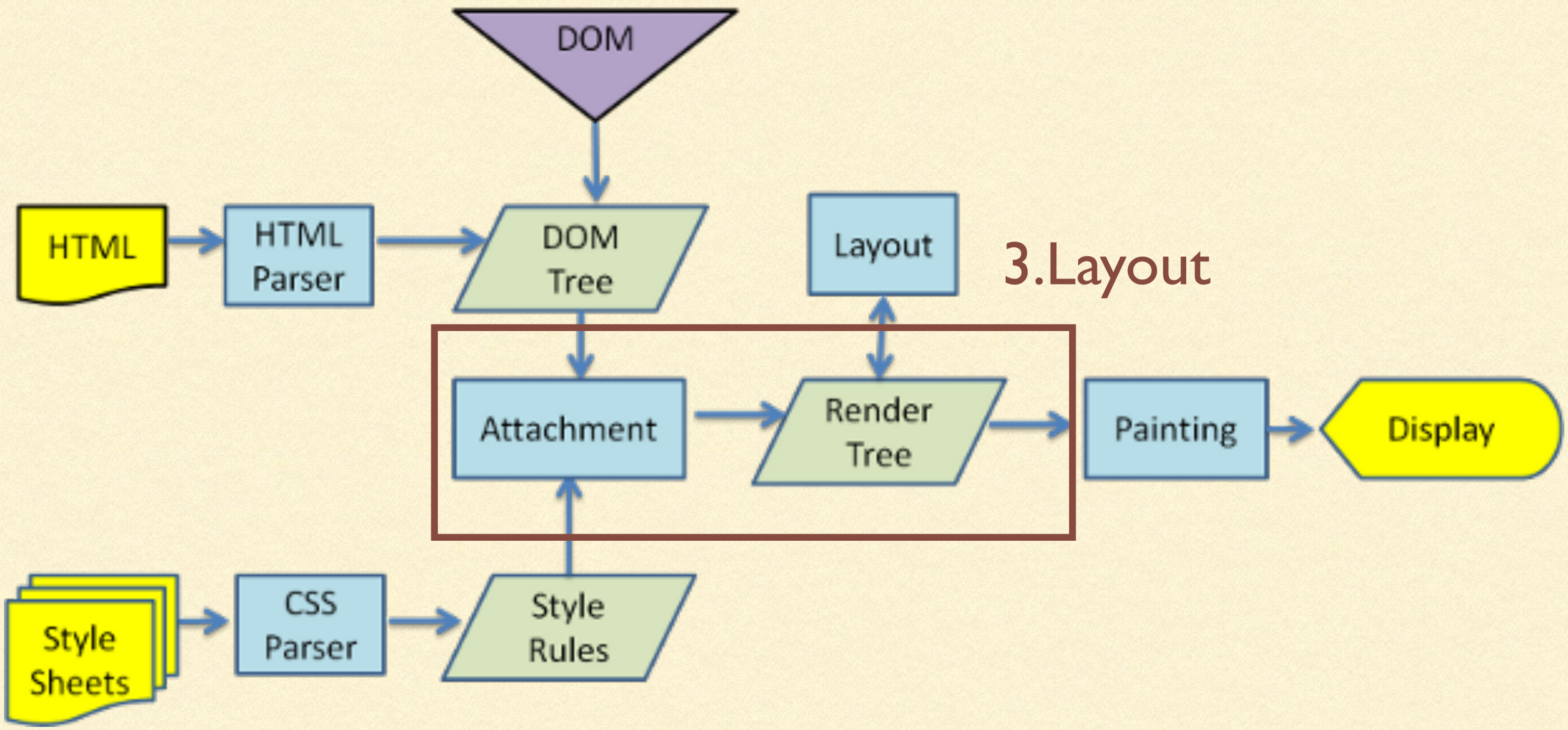


Recalculate style

DOM ツリーに対してそれぞれのdomにスタイル情報をマッピングしていく

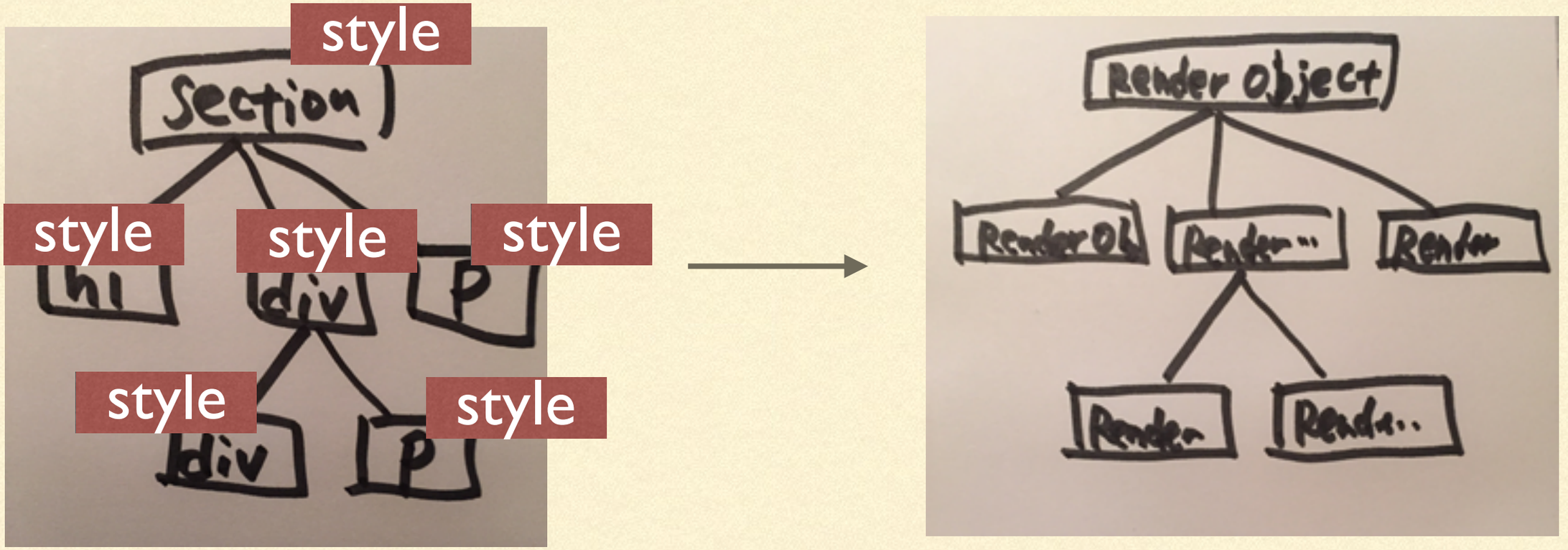


レンダリングフロー（WEBKIT）



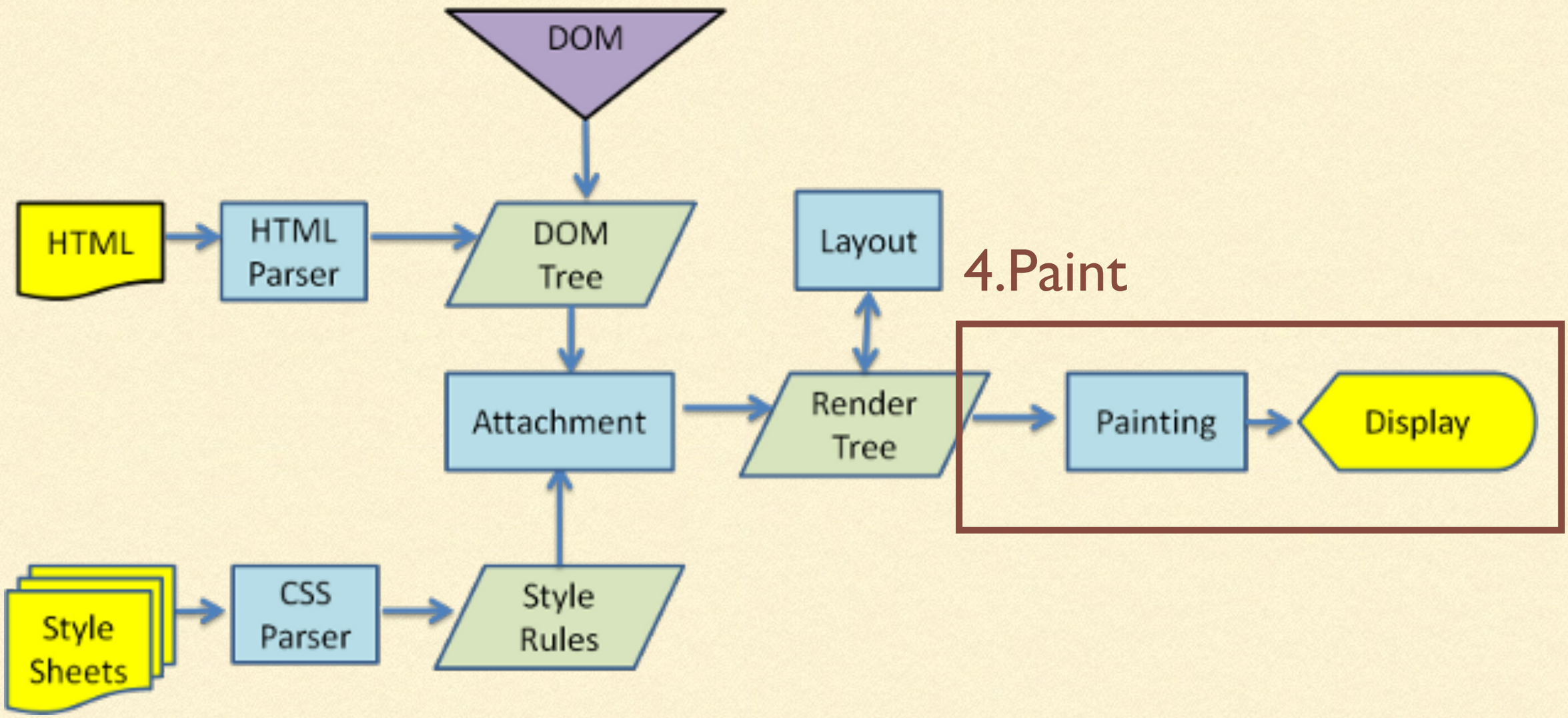
Layout

domエレメントに対してサイズ、座標の情報を与え、
レンダーツリーを構築する



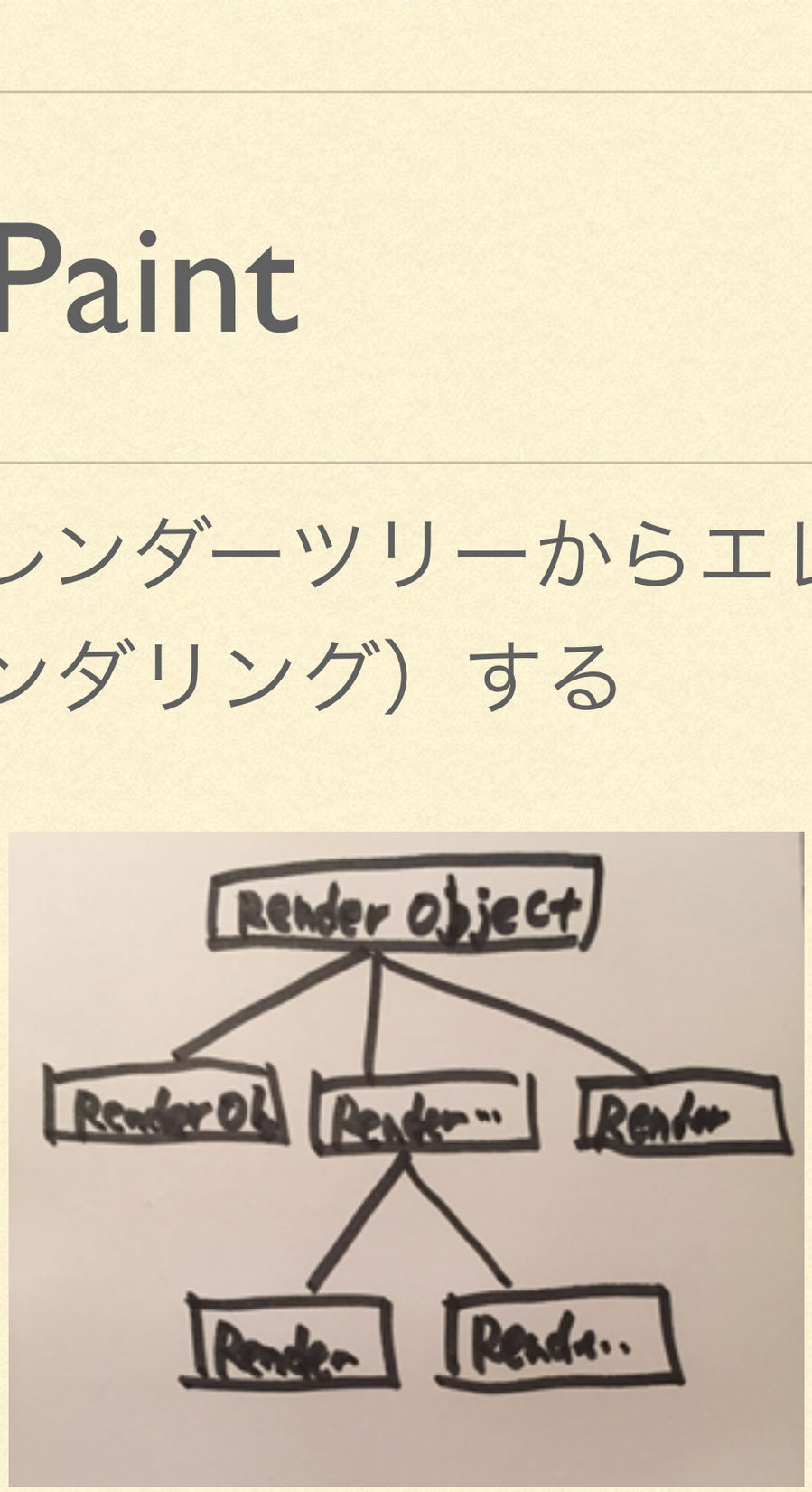
レンダーツリーには表示する要素のみが格納される（headタグやdisplay:noneの要素は含まれない）
domやスタイルの変更、ブラウザのリサイズなどで発生

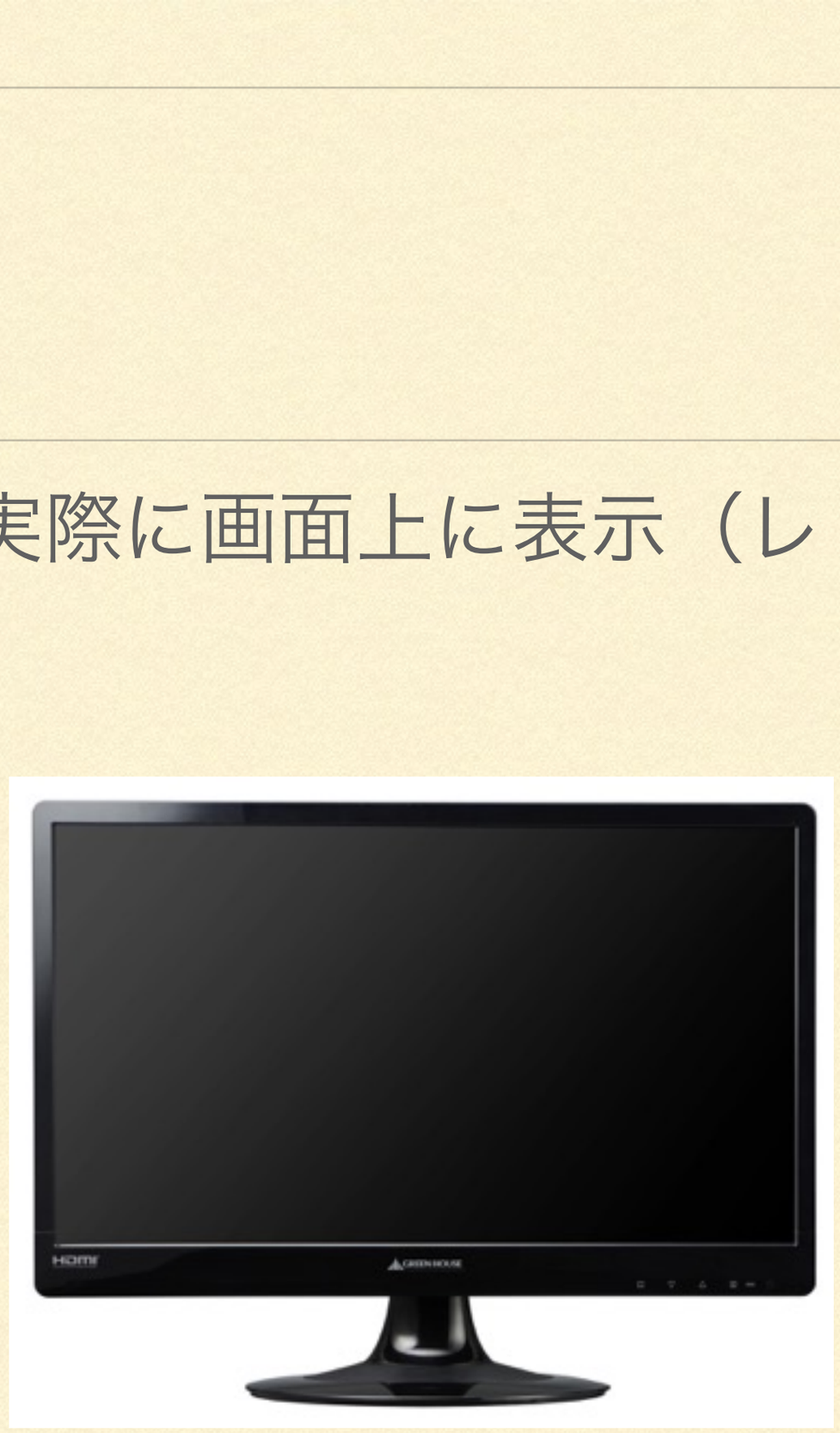
レンダリングフロー（WEBKIT）



Paint

レンダーツリーからエレメントを実際に画面上に表示（レンダリング）する





chrome dev toolsで実際に見てみましょう

表示におけるパフォーマンスを考慮する場合、

特に**Layout**と**Paint**に注目

Layoutを発生させる要因

- ウィンドウのリサイズ、スクロール、DOMの操作、width, height,positionなどスタイルのサイズやポジションの変更

Paintを発生させる要因

- background-color, visibility, displayなど視覚的なスタイルの変更

例えばこういうクラスがあった場合

```
.hoge {
    width:50px;
    height:100px;
    background-color: #ff0000;
}
```

例えばこういうクラスがあった場合

```
.hoge {
    width:50px;       Layout発生
    height:100px;     Layout発生
    background-color: #ff0000;  Paint発生
}
```

CSS TRIGGERS...

http://csstriggers.com/

例えばこういう処理があった場合

document.body.style.color = ‘red’

例えばこういう処理があった場合

document.body.style.color = ‘red’　Paint発生

例えばこういう処理があった場合

document.body.style.color = ‘red’　Paint発生

```
document.body.style.padding = '100px';
```

例えばこういう処理があった場合

document.body.style.color = ‘red’　Paint発生

document.body.style.padding = '100px';　Layout + Paint発生

- Layoutが発生すると対象の要素が属してるノードの位置情報やサイズの情報が再計算されるため、非常にコストがかかる。
- 例えばvisiblity:hiddenはpaintingのみ発生するがdisplay:noneはLayoutとPaintingの両方が発生。位置的な変更がなされるdisplay:noneの方が実はコストがかかる。

point!

LayoutとPaintの回数を抑える

次回へ続く