CSSによる「スタイリング」コントロールの苦悩と変遷

半田 惇志

半田 惇志

- フリーランス
  - HubSpotテックエキスパート
  - フロントエンドエンジニア
- CRM（顧客管理システム）/
  SFA（営業支援システム）/
  MA（マーケティングオートメーション）
  の導入支援をしたり、
  ウェブサイトを構築したり

【第一章】
CSSの誕生

“

CSSはつらい

-全人類

引用元：独断と偏見

```
<p>
  <font
    size="4"
    color="blue"
    face="Arial"
    >青色で、フォントサイズが4、Arialを使用
</font></p>

<table border="1" bgcolor="yellow">
  <tr>
      <td>このセルは背景色が黄色</td>
  </tr>
</table>
```

スタイリングの歴史の始まり

「〇〇へようこそ」「あなたは〇〇番目のお客様です」というテキストも散見された

HTMLは「文書」

HTMLはそもそもCERN（欧州原子核研究機構）の
ティム・バーナーズ＝リーによって、
研究データや文書を著すために開発された

→ Webサイトのような使われ方、スタイリングまでは想定されていなかった

WWWの発達・時代の流れとともに
HTMLがウェブサイトに使われることが
一般化する

“

属性でスタイリングするのつらい

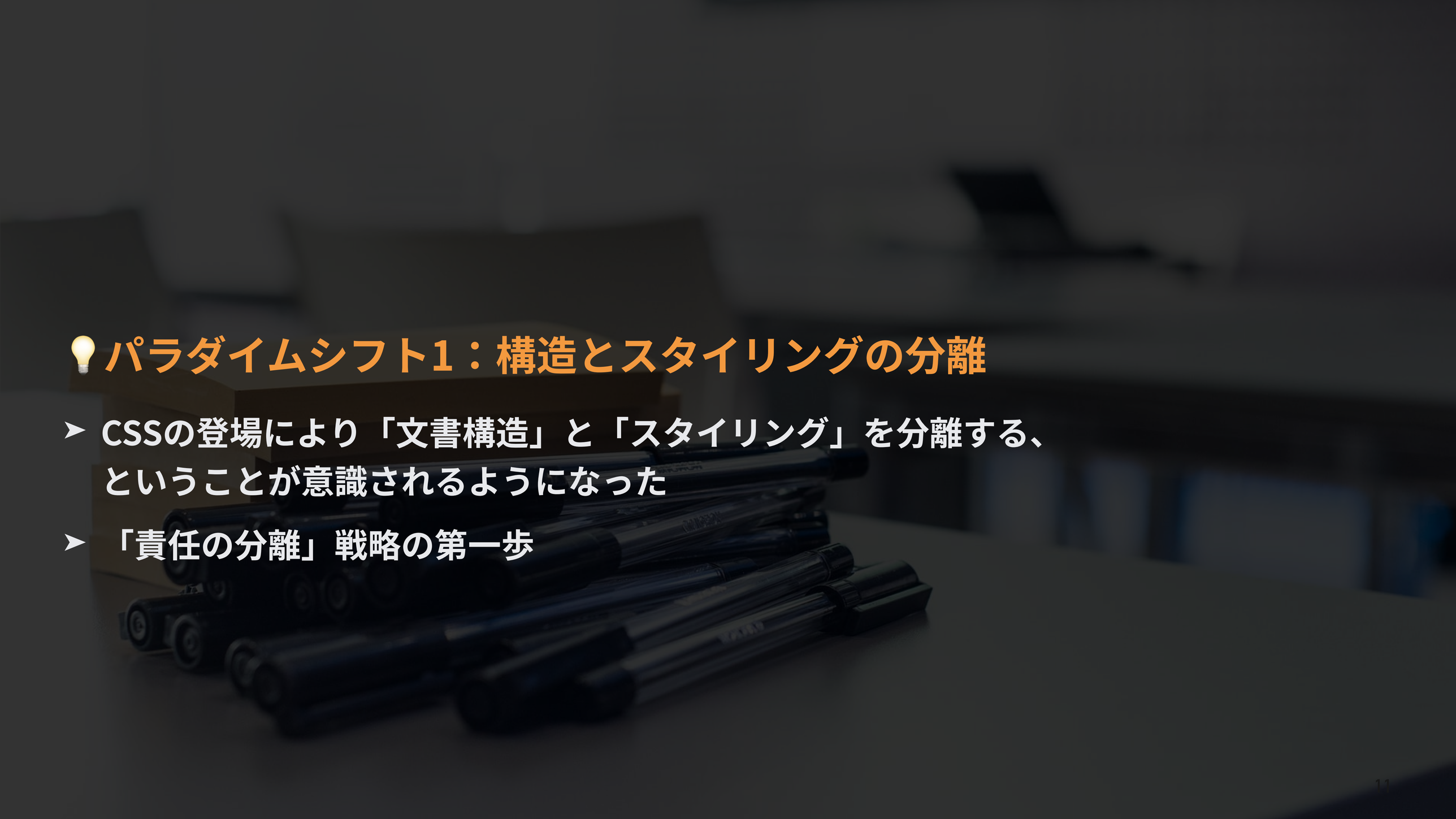
-全人類

タグ・属性スタイリングが抱えていた課題

- 文書構造（HTML）とスタイリング（見た目）が一緒くたになってしまっている
- とにかく使い回しができない

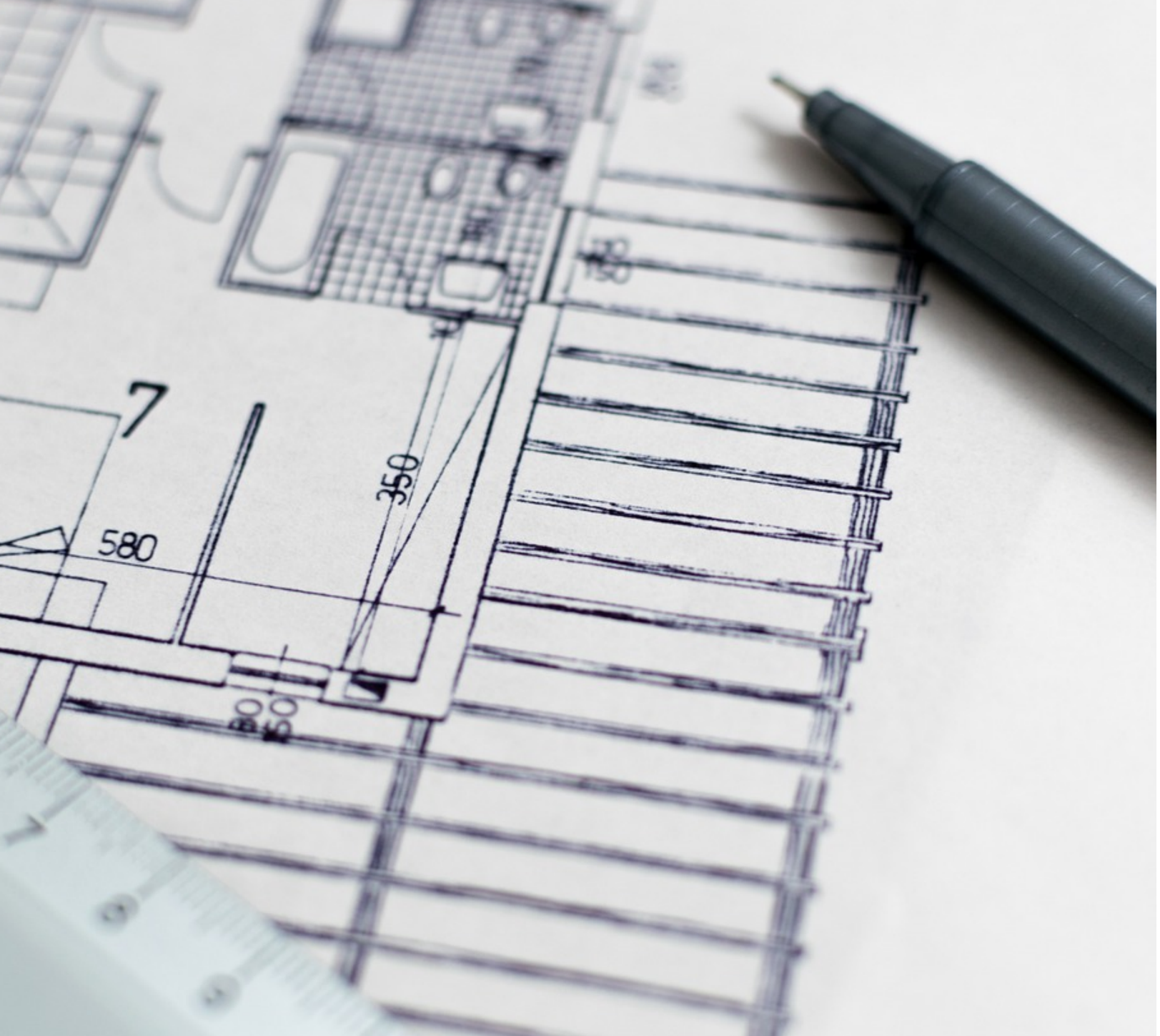
CSSの誕生 👶

1996年にW3Cによって
Cascading Style Sheets, level 1としてCSSが勧告された



💡パラダイムシフト1：構造とスタイリングの分離

- CSSの登場により「文書構造」と「スタイリング」を分離する、ということが意識されるようになった
- 「責任の分離」戦略の第一歩



【第二章】
CSS設計の誕生

CSSはとてもよく受け入れられた

CSSは記法がとにかくシンプルであるため、標準的に利用され始めた

“

CSSがあればスタイリングの使い回しができる！

CSSがあれば何でもできる！

- コーダー

1, 2, 3, ダー！

A FEW YEARS LATER...

何も考えずに書くとスタイルが衝突しまくる

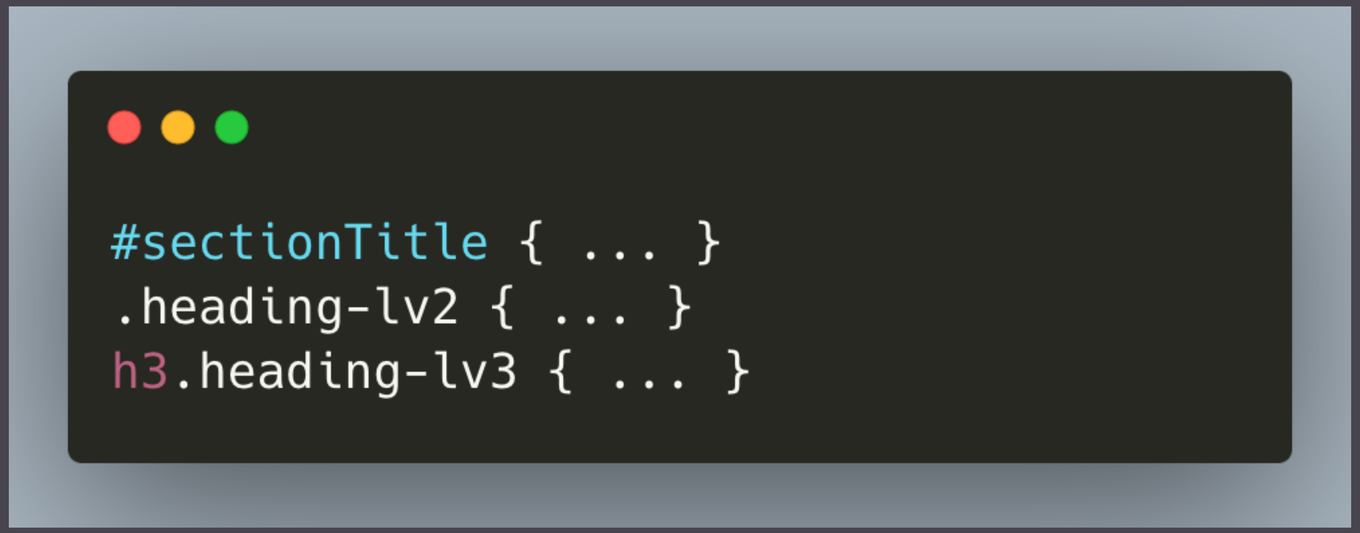
```css
h2 {
  font-size: 2rem;
  border-bottom: 1px solid;
}

.heading-lv2 {
  font-size: 3rem;
  border-bottom: none;
}
```

人によって書き方がバラバラ

IDセレクタを使ったり、要素型セレクタ使ったり……

なんで上書きされない？？？？？？？（詳細度）



“

ちょっといじったら他のページが崩れる 🤯

CSSわけわからん 🤯

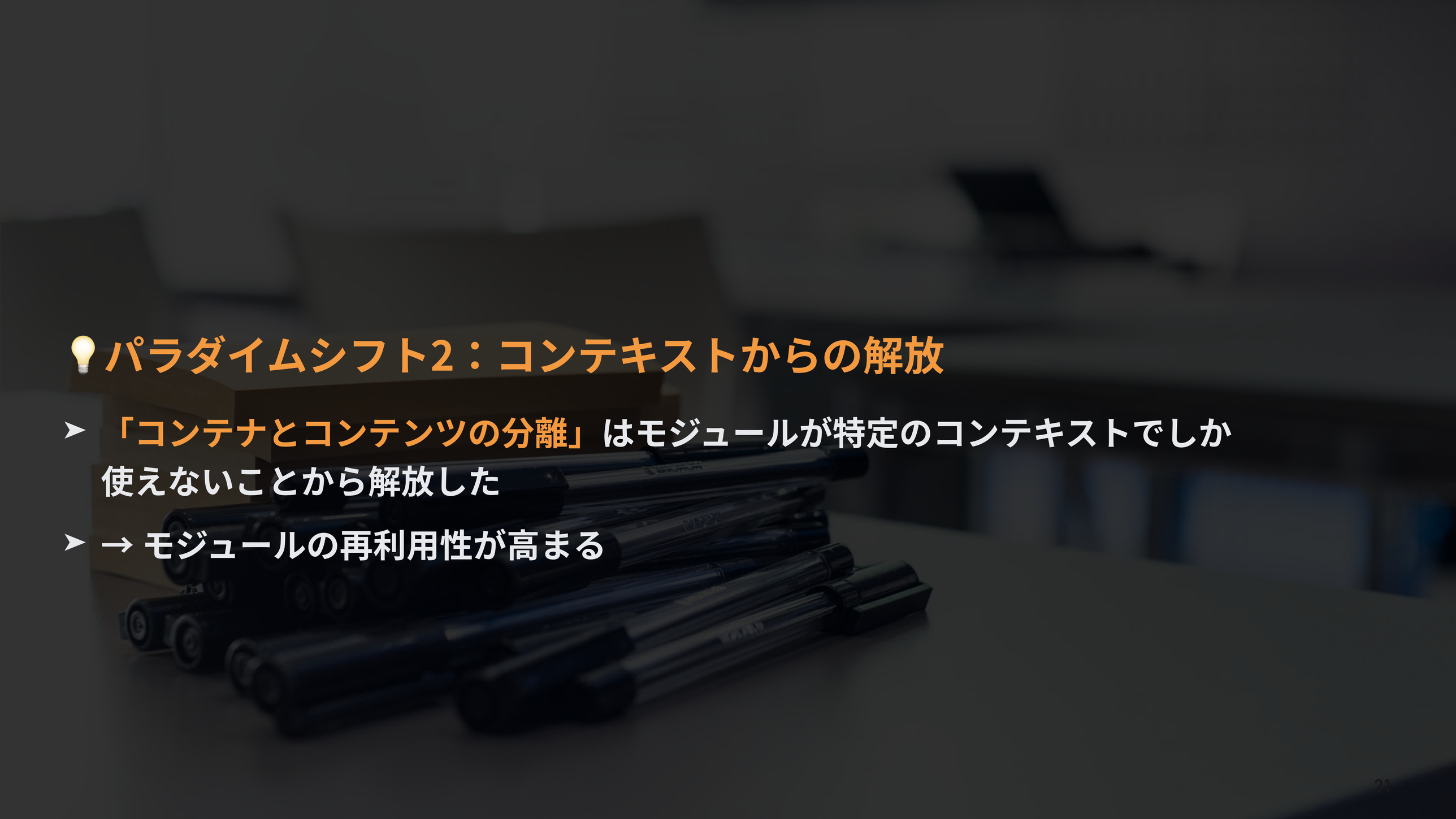
-コーダー

殺伐としたウェブ業界に救世主が！

OOCSS

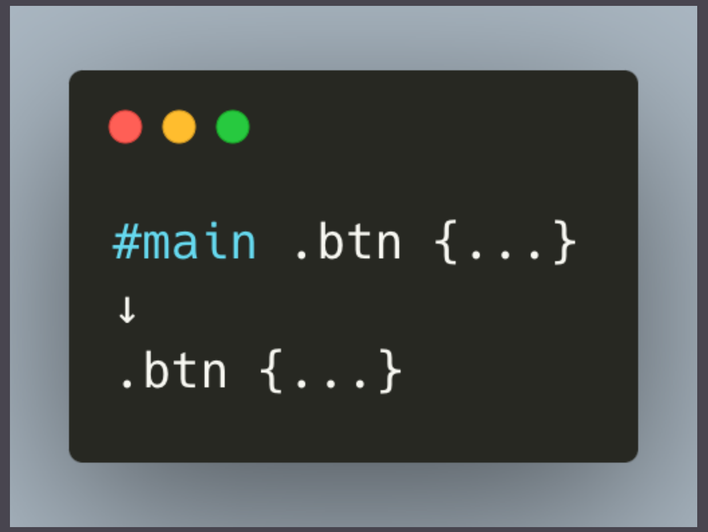
- 2009年から提唱され始めた
- CSS設計の中ではかなり古い方で、
  他のCSS設計にも少なからず影響を与えている
- 「コンテナとコンテンツの分離」
- 「ストラクチャーとスキンの分離」
- という2つの原則から、UIを
  “モジュール” として分解し再定義する、
  というアプローチが画期的だった

http://oocss.org/
https://www.slideshare.net/stubbornella/object-oriented-css など



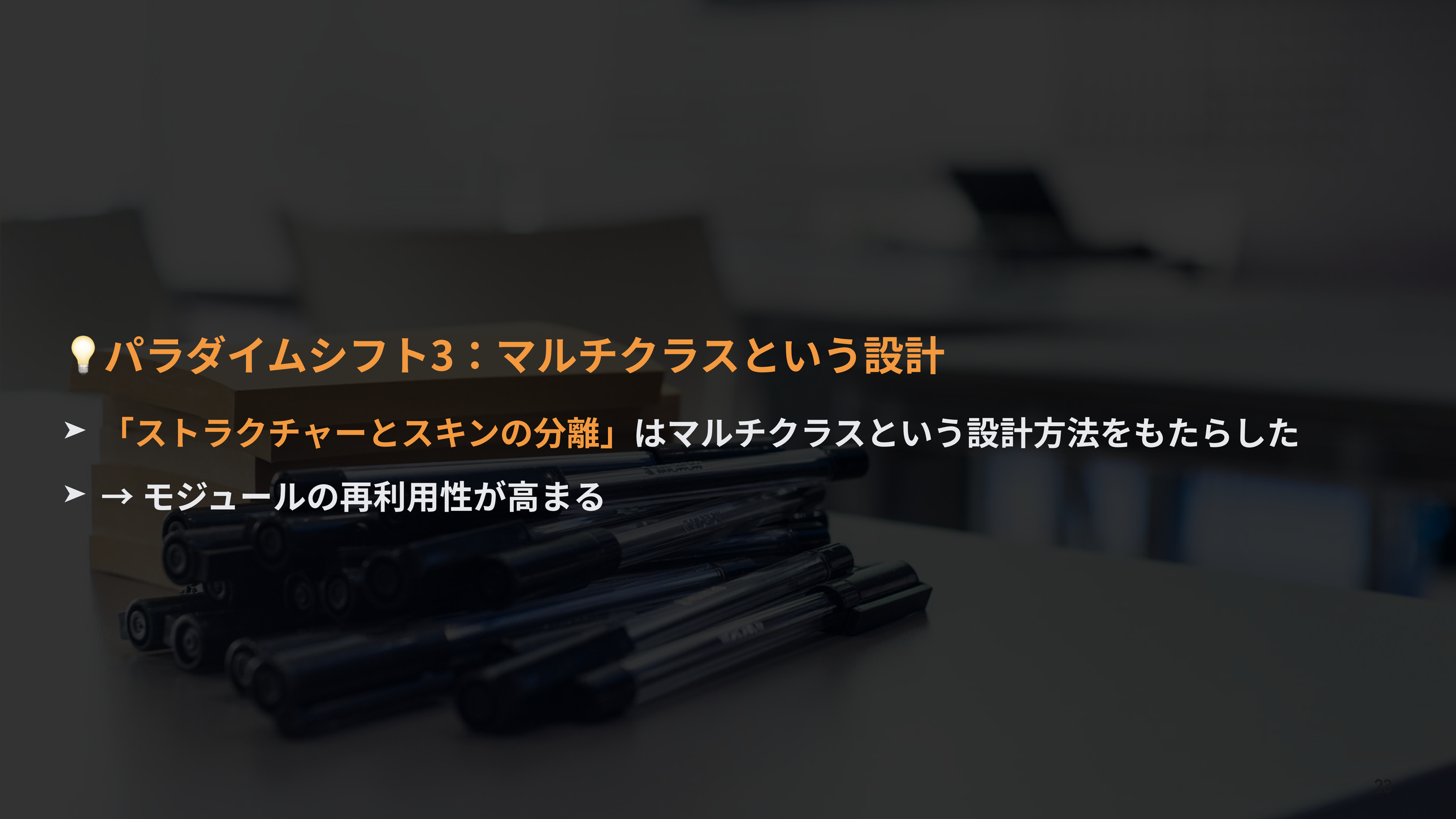
💡パラダイムシフト2：コンテキストからの解放

- **「コンテナとコンテンツの分離」**はモジュールが特定のコンテキストでしか使えないことから解放した
- → モジュールの再利用性が高まる



コンテナとコンテンツの分離

特定のコンテキスト（コンテナ）に依存させない



💡パラダイムシフト3：マルチクラスという設計

- 「ストラクチャーとスキンの分離」はマルチクラスという設計方法をもたらした
- → モジュールの再利用性が高まる

```html
<button
  class="btn"
  type="button">
  標準ボタン
</button>

<button
  class="btn warning"
  type="button">
  要注意ボタン
</button>
```

```css
.btn {
  display: flex;
  min-width: 200px
}

.warning {
  background-color: yellow;
}
```

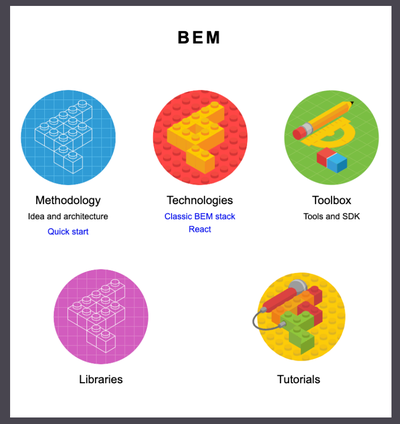
ストラクチャーとスキンの分離

構造とあしらいを分ける

A FEW YEARS LATER...


OBJECT ORIENTED CSS
for high performance web applications and sites.
Nicole Sullivan


"SMACSS is becoming one of the most useful contributions to front-end discussion in years"
Scalable and Modular Architecture for CSS
A flexible guide to developing sites small and large.
by Jonathan Snook


BEM
Methodology
Idea and architecture
Quick start
Technologies
Classic BEM stack
React
Toolbox
Tools and SDK
Libraries
Tutorials


Web制作者のための
CSS設計
の教科書
モダンWeb開発に欠かせない
「修正しやすいCSS」の設計手法
あとで困るコードを書いてない?
予測しやすい
保守しやすい
再利用しやすい
拡張しやすい
コンポーネントやCSSプリプロセッサを意識した設計・管理の実践など、
「ちゃんとCSSを書く」ためのプロの考え方を徹底解説!
インプレス


PRECSS

CSS設計 群雄割拠

こうして世界に平和が訪れた！

【第三章】
コンポーネント環境の誕生

“

やっぱりCSSわけわからん

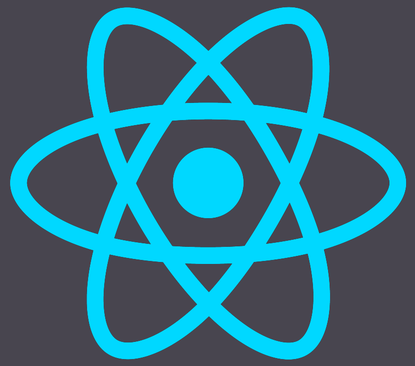
-疲弊したコーダー

CSS設計があれば完璧だったはずなのに……

Q. なぜこうなったのか？

A. CSS設計を「きちんと」行うのは非常に難しい

殺伐としたウェブ業界に救世主が！



JavaScriptフレームワークの登場

💡パラダイムシフト4：コンポーネント環境

- 技術の進化により、「コンポーネント環境」が整備された
- **コンポーネントごとにスタイリングのスコープを形成できる**
- → もう .header__nav-item--has-children のような長ったらしいクラス名を付けなくても、他の要素とスタイリングが干渉することがない

```
<style scoped>
.example {
  color: red;
}
</style>

<template>
  <div class="example">hi</div>
</template>
```

Vue.jsのSingle File Componentの例（コンパイル前）

公式ドキュメントより

```
<style>
.example[data-v-f3f3eg9] {
  color: red;
}
</style>

<template>
  <div class="example"
       data-v-f3f3eg9>hi</div>
</template>
```

Vue.jsのSingle File Componentの例（コンパイル後）

公式ドキュメントより

【最終章】
現在の状況

Q. コンポーネントの環境の登場により、
CSSに関する悩みは無くなったのか？

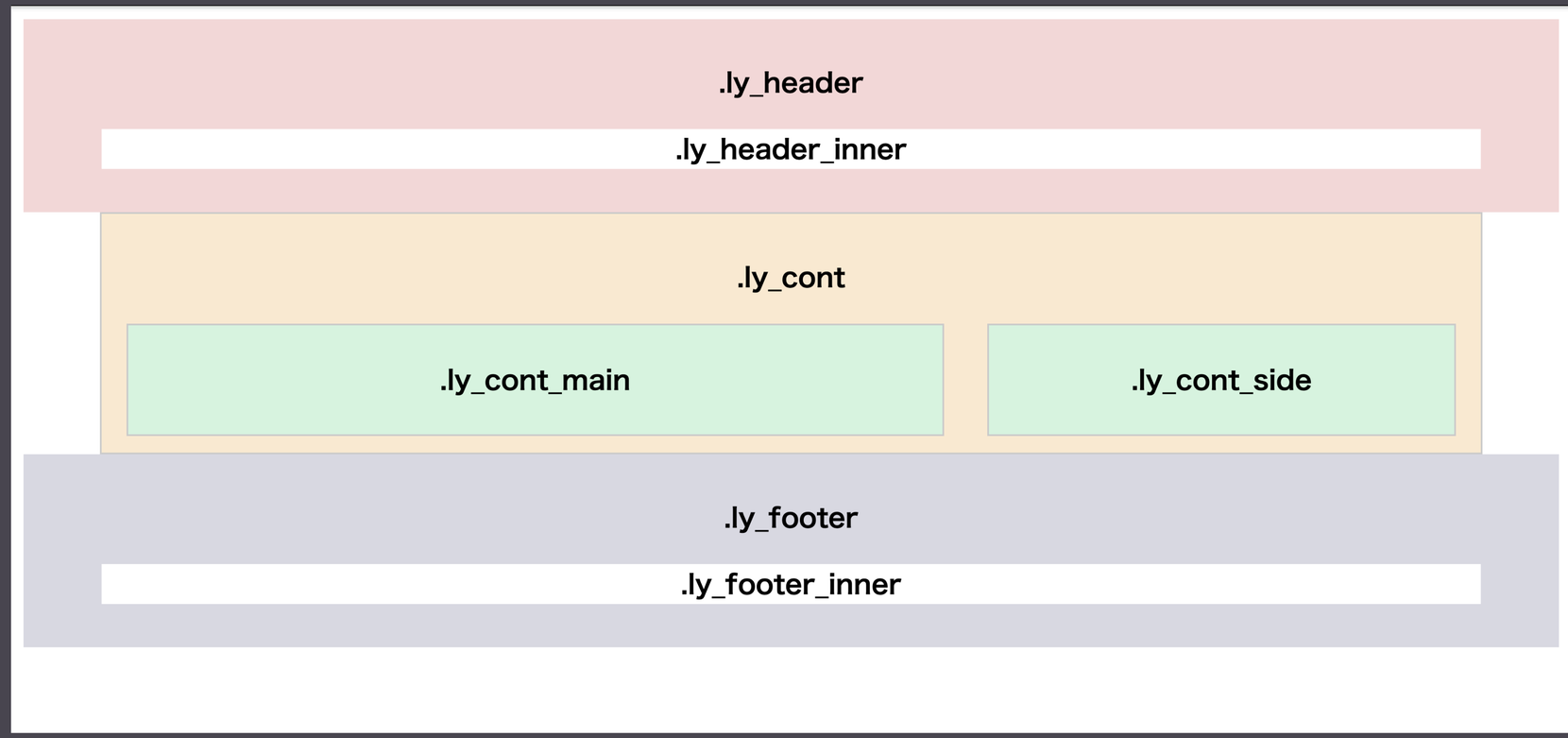
A. そんなことはない

CSS設計が担っていた役割

- ⏬ カスケード設計
  - スタイリングをどう分類し、詳細度を管理し、適用されるスタイルを予測しやすくするか
  - リセットスタイルの設計や、コンテナにあたるクラスの設計など
- ♻️ モジュール設計
  - ビジュアルをどのように分解し、モジュールとして独立させるか
- 🔠 名前設計
  - スタイルの衝突を避けるために、どう名前を付けるか
- 💅 スタイリング設計
  - モジュールが十分再利用しやすくあるために、どうスタイリングするか

スタイリングのレイヤー分けの例

- reset
  - * { margin: 0 }
- base
  - body { font-family: XX }
  - a { color: XX }
- layout
  - 右図
  - コンテンツ幅を管理するのは .ly_cont の役割で、
    コンポーネントにコンテンツ幅の設定などをしてはいけない



コンポーネント環境が担う範囲と担わない範囲

- **コンポーネント環境が担う範囲**
  - 🔠 **名前設計**
    - スコープ形成機能により、スタイリングのための名前はほぼ気にしなくてOK
    - なんなら要素型セレクターを使ってもよい
- **担わない範囲**
  - 💅 **スタイリング設計**
    - 「コンポーネントに当ててはいけないスタイル」や「子要素の制御の仕方」を理解していないと、使いづらい負債コンポーネントが簡単に出来上がる
    - 極端に言えばコンポーネントに position: absolute を付けていると地獄を見る

コンポーネント環境が担う範囲と担わない範囲

- どちらとも言えない範囲
  - ⏬ カスケード設計
    - スコープ機能により、よっぽど特大コンポーネントを作っていない限りは
      詳細度はそこまで気にしなくてよい（むしろ特大コンポーネントを見直しましょう）
    - しかし、コンポーネントの手前の reset / base / layout というような
      レイヤー分けの発想は引き続き重要
  - ♻️ モジュール設計
    - 「ビジュアルをどう分解するか」という観点は引き続きCSS設計の知見が重要

つまり現代の状況は……

- コンポーネント環境でラクはしつつ
- どのような単位でコンポーネントを区切るか
- 再利用しやすいコンポーネントにするにはどうすべきか
- コンポーネントのスタイリングや振る舞いの
  上書きにおいて、どう制御するか

などは引き続き、
CSS設計の知見や実際のトライアンドエラーが必要

ありがとうございました！

assialiholic