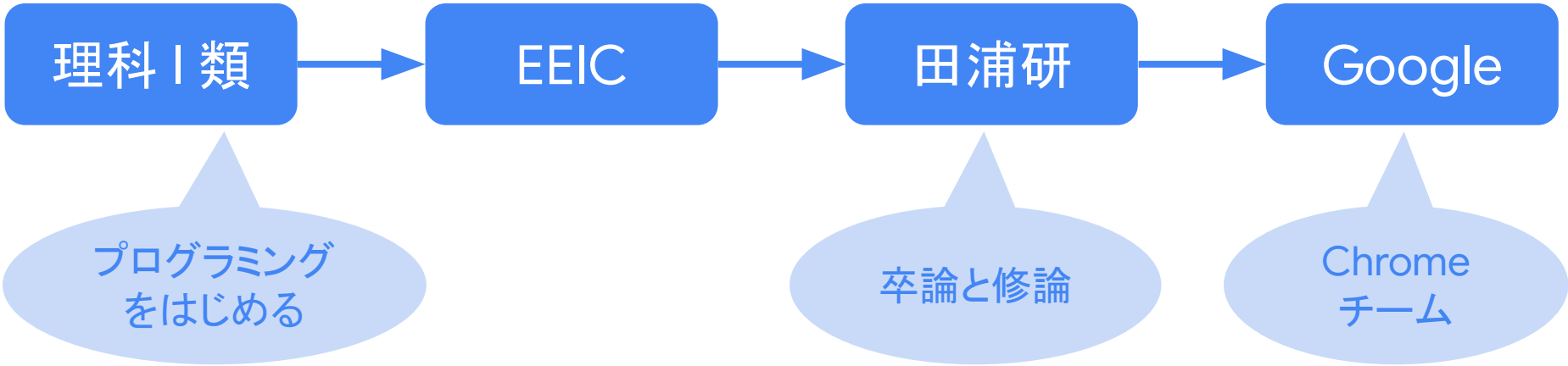
電子情報学特論：
Chromium のアーキテクチャを解き明かす
～ EEIC の授業が生きるプロダクトの世界～

Kentaro Hara

2024 May

(◎>◡<◎)

自己紹介



教養〜EEIC 時代

- プログラミングをはじめたのは大学に入ってから
  - 「情報」の授業がきっかけ

- 田浦先生の授業の課題で出たソーティングの高速化にはまる
  - 半年間をソーティングに費やす => 論文を出す

- 田浦先生の「言語処理系」の実験に夢中になる
  - プログラミング言語を自作

田浦研時代

- 並列分散プログラミング言語処理系の開発で卒論と修論を書く

- 他にやってたこと
  - プログラミングコンテスト
  - 学科用 PC の設定・管理
  - 大規模クラスタ（600台）のサーバ管理
  - TA（このとき作った実験はまだ受け継がれてます！）

Google へ

- 研究開発が好きだったので Google か博士かで迷った
- 当時 Google といえば並列分散処理の最先端（MapReduce、GFS）だったけど、そのチームが東京にはなかった・・・(´・ω・`)
- ならば全く違うことをやってみようと思って Chrome チームへ！
- あれから 13 年
- エンジニア → テックリード → ディレクター（でもコードはたくさん見てます）

趣味：日本料理を極めること

道場六三郎さんの YouTube チャンネルで料理してきた

第8回ジビエ料理コンテストのお店部門で「全国日本調理技能士会連合会会長賞」受賞

本題: Chromiumの話

Chromium と Chrome

Chromium

Google Chrome

Chromium と Chrome

- Chromium = オープンソースのウェブブラウザプロジェクト

- Chromiumを利用しているプロダクト
  - Google Chrome
  - Microsoft Edge
  - Opera
  - Brave
  - ...

「4 つの S」

- Chromium が最も大事にしている「4 つの S」とは何でしょう？

S....

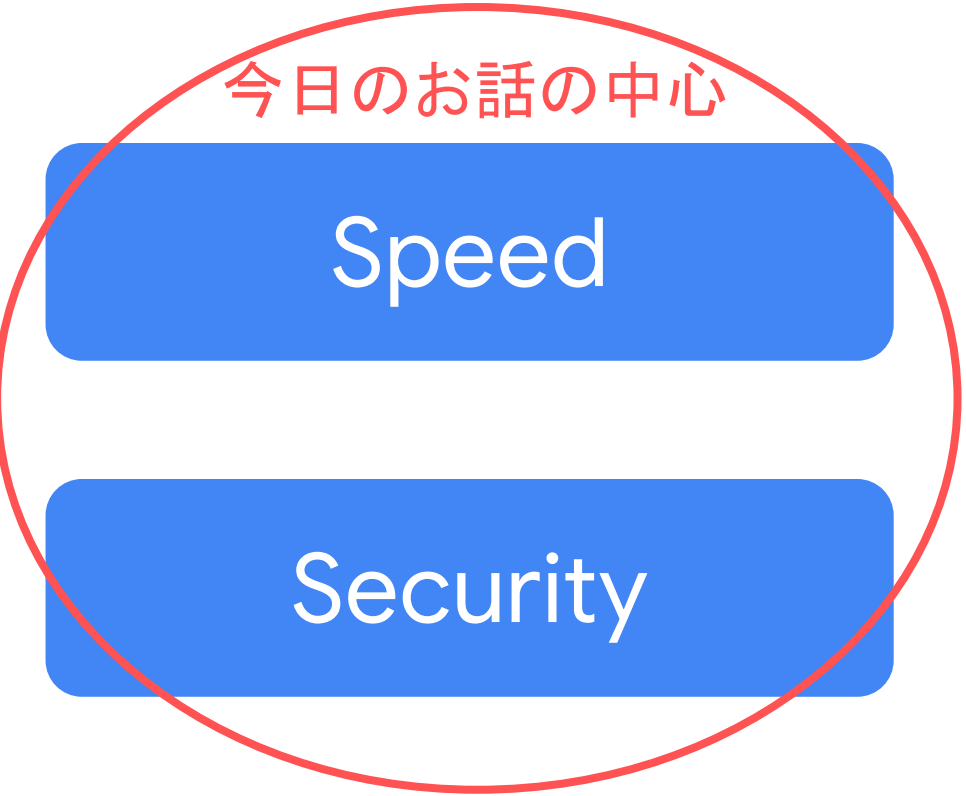
S....

S....

S....

「4 つの S」

- Chromium が最も大事にしている「4 つの S」とは何でしょう？



今日のお話

1. プロセスモデル
2. メモリモデル
3. レンダリングモデル

プロセスモデル

プロセス vs. スレッド

クイズ：違いは何だっけ？ (๑>◡<)ﾉ

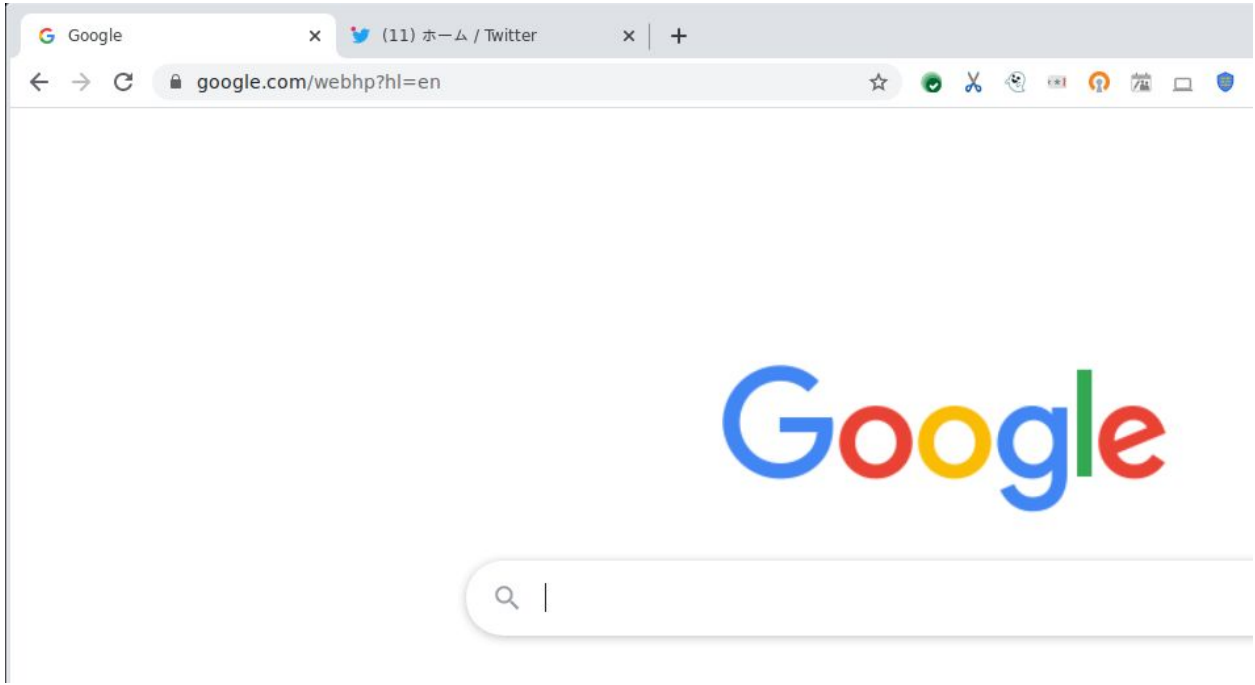
プロセス vs. スレッド

- プロセス
  - プロセス間でメモリ空間が共有されない
  - 1個のプロセスがクラッシュしても・クラックされても、他のプロセスには影響が及ばない

- スレッド
  - スレッド間でメモリ空間が共有される
  - そのぶんプロセスより軽い（メモリ使用量、コンテキストスイッチ）

プロセス vs. スレッド

- クイズ：Chromium で google.com と twitter.com を開いたとき、いくつのプロセス or スレッドを作るべきか？

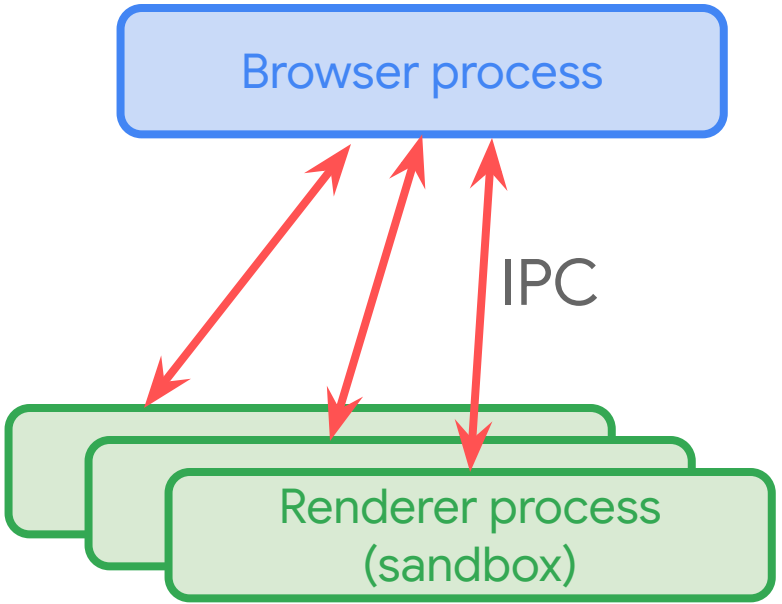


マルチプロセスアーキテクチャ

- 重要な仮定：
  - 「悪意のある Web ページからユーザを守る」
    - ユーザのクッキーを盗む
    - OS にウィルスを送り込む
    - 他のタブの Web ページを改ざんしてフィッシング詐欺
    - …
- 帰結（マルチプロセスアーキテクチャ）：
  - Web ページごとにプロセスレベルで分離してサンドボックス内で走らせる

マルチプロセスアーキテクチャ

- Browser process =「親玉」プロセス
  - 1 個だけ存在
  - ファイルやユーザのクッキーなどシステムリソースにアクセス可能
- Renderer process = Webページをレンダリングするプロセス
  - N 個存在
  - サンドボックス内で走る
  - Browser process とはプロセス間通信でやりとり
- 今となっては当然に思えるアーキテクチャだが Chrome が出た 2008 年当時は斬新だった



しかし話は終わらない（｜◡｜✿）

インターネット黎明期に Web を成功へ導いたもの

- Openness
  - 世界中の情報にオープンにアクセスできる
- Linkability
  - リンクを通じて情報を結び付けられる
- Ephemerality
  - インストール不要
- Embeddability
  - いろんな人たちが作ったコンテンツをセキュアに埋め込める

Embeddability

- <iframe> を使えば他のサイトが作ったコンテンツを埋め込める
  - Google Maps、Facebook のいいねボタン、Web 広告

```
<html><body>
こんにちは！ようこそ私のホームページへ！
あなたは 1206 番目の訪問者です(◍•ᴗ•◍)
<iframe src="https://maps.google.com/..."></iframe>
<iframe src="https://facebook.com/..."></iframe>
<iframe src="https://g.doubleclick.net/..."></iframe>
</body></html>
```

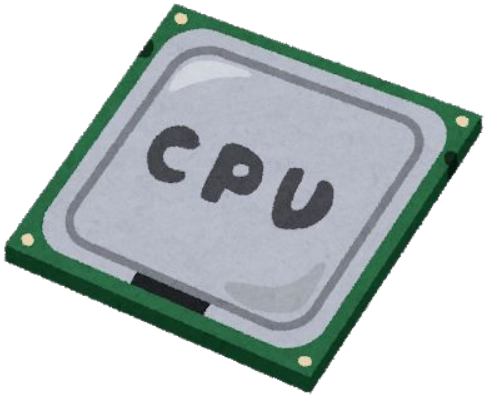
Embeddability

- 問題点：他のサイトが作ったコンテンツは信用できない
  - citibank.com が Web 広告を埋め込んでいて、これらが同一プロセス内に存在したとき（＝メモリ空間を共有していたとき）、もし何かしらのバグを悪用して Web 広告が citibank.com のパスワードを盗めたとしたら...

- ところで「何かしらのバグ」って何よ？
  - Chromium のメモリ関連のバグ（後述）
  - 重大な問題になったのが、2018年に発見された Spectre & Meltdown と呼ばれる CPU のバグ

Spectre とは

- CPU の投機的実行と分岐予測を悪用して、任意のメモリアドレスの値を読み込めてしまうバグ
  - Google の Project Zero（セキュリティチーム）と Paul Kocher が独立に発見
  - 現代のほぼすべての CPU に存在する致命的なバグ



Spectreの原理

```
// こういうArray構造体があるとする
struct Array {
  char at(int i) {
    if (i < length)
      return buffer[i];
    else
      return -1;
  }
  int length;
  char* buffer;
};
```

```
// 以下のコードで攻撃する
int data[2000];
// 狙ったアドレスを指定
char v = array->at(12345678);
if (v == -1) { return; }
else if (v & 1) { data[0]; }
else { data[1000]; }
```

投機的実行がなければ　　の部分が実行される

Spectreの原理

```
// こういうArray構造体があるとする
struct Array {
  char at(int i) {
    if (i < length)
      return buffer[i];
    else
      return -1;
  }
  int length;
  char* buffer;
};
```

```
// 以下のコードで攻撃する
int data[2000];
// 狙ったアドレスを指定
char v = array->at(12345678);
if (v == -1) { return; }
else if (v & 1) { data[0]; }
else { data[1000]; }
```

投機的実行があると、(あとで外れることが判明する)分岐予測によって `return buffer[i];` が実行されて、`buffer[12345678]`の値によって、`data[0]` と `data[1000]` のどっちかがキャッシュに読み込まれる

Spectreの原理

```
// こういうArray構造体があるとする
struct Array {
  char at(int i) {
    if (i < length)
      return buffer[i];
    else
      return -1;
  }
  int length;
  char* buffer;
};
```

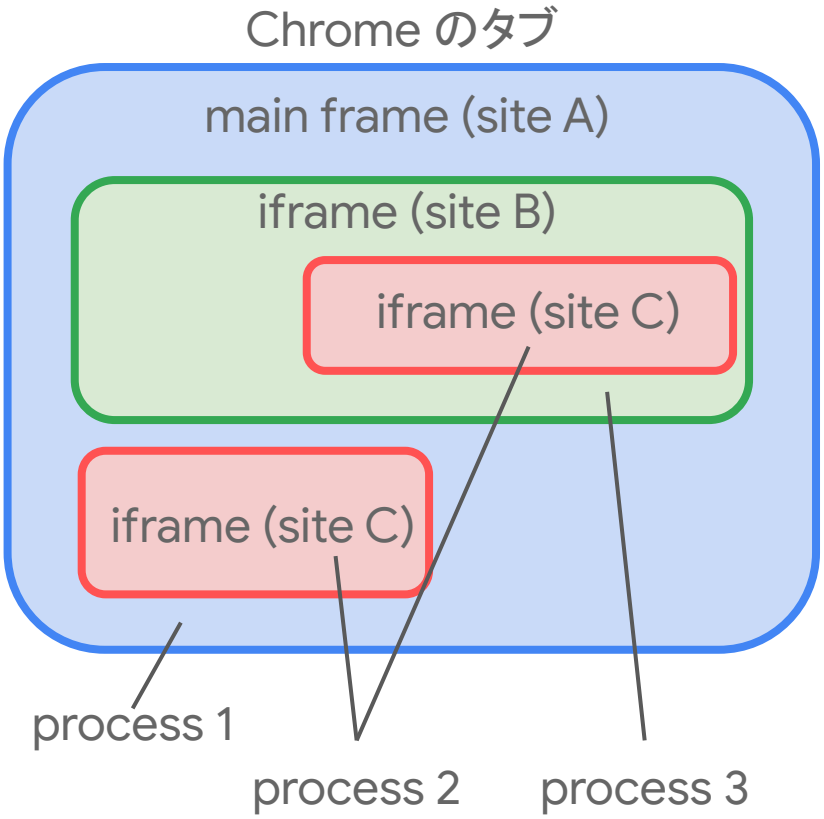
```
// 以下のコードで攻撃する
int data[2000];
// 狙ったアドレスを指定
char v = array->at(12345678);
if (v == -1) { return; }
else if (v & 1) { data[0]; }
else { data[1000]; }
```

分岐が外れたとわかった時点で投機的実行の結果は捨てられて　　の部分がやり直されるのだが･･･ data[0] と data[1000] のどっちかがキャッシュに残ってしまう

ということは data[0] と data[1000] のアクセス速度を調べれば、buffer[12345678] の中身を確定できるじゃん！！ => Web広告からcitibank.comのパスワードが読めるかも

解決策：Site Isolation

- <iframe>ごとにプロセス分離する
  - Site Isolation と呼ぶ
- Chrome では 1 個のタブに見えるものが実は多数のプロセスの集合としてうまく連携して動いている
  - 相当難しいことをやってます
  - すべては Embeddability と Security を両立させるため



ところで、
Chromeってメモリ使いすぎじゃない？？

メモリ使いすぎ問題

- セキュリティの確保が最大の理由
  - cnn.com を開くだけで <iframe> をプロセス分離するために 10 個以上のプロセスができる
  - Chrome拡張を分離するためのプロセス
  - メモリアロケータのセキュリティ強化
  - ...

- メモリ削減の努力もたくさんしているが・・・

意外なことが判明する・・・

- ユーザがタブを開きすぎる問題（30個とか）
  - 最近使ってないタブをこっそり消す A/B テストをやってみた
  - メモリ使用量は大きく減ったが、2ヶ月後になぜか元の水準に戻ってしまった・・・なぜ？
  - ユーザが開くタブの個数が大幅に増えていた
  - どうやら「ユーザは遅いと感じるまでタブを開き続ける（そして遅いと言う）」ことが判明

参考 : Jevons paradox

- イギリス産業革命期の 1865 年に経済学者 Jevons が発表したパラドックス
  1. 技術進歩によって石炭をより効率的に利用できるようになる
  2. より広範な産業で石炭が使われるようになる
  3. 石炭消費量が増加する

- 重要なこと
  - メモリ使用量は減らなくてもユーザは
    より多くのことをできるようになっている

メモリモデル

クイズ

- Chromiumのソースコードは何行くらいあるでしょう？

(a) 25万行
(b) 250万行
(c) 2500万行
(d) 250行

ちなみに Linux カーネルが2800万行

クイズ

- 2500 万行のうち空行は何行あるでしょう？

(a) 5 行
(b) 50万行
(c) 500万行
(d) 2500 万行

クイズ

- Chromium の大半は何の言語で書かれているでしょう？

(a) C++

(b) Java

(c) Rust

(d) ピダハン語

バグはつきもの

- 現在、90000個くらいバグが報告されているよ(´･ω･`)
- おもしろいやつもある

On the logo of cheeseburger in the "users" cheese located under the a cutlet. As I know, cheese never put under the cutlet.

1.png
31.4 KB View Download

検証してみた

結論：どっちもおいしかった

C++という選択

- 利点 : Speed
- 欠点 : Security

Security 上の懸念をどう補っていくか？

Use-after-free

- Security 上深刻なのは Use-after-free と呼ばれるタイプのメモリバグ
  - Free したメモリ領域にアクセスしてはいけません^^

```
Object* obj =
    malloc(sizeof(Object));
free(obj);
obj->Foo(); // Use-after-free
```

```
// もう少し現実的な例
class A {
  A(B* b) : b_(b) { }
  void Foo() {
    b_->Bar(); // この時点でb_が生きていることはどう保証されるのか...?
  }
  B* b_; //生ポインタは常に危険を伴う
};
```

Use-after-free の何が問題なのか

```
class A {
  A(B* b) : b_(b) { }
  void Foo() {
    b_->Bar();
  }
  B* b_;
};
```

1. オブジェクト B を解放して、b_->Bar()が Use-after-free を起こす状態にしておく
2. 「b_->Bar()を呼ぶと任意のコードが実行されるようなバイト列」からなるオブジェクトを大量に生成する。運が良ければそのうちのひとつは b_ が指すアドレスに確保される。
3. A::Foo()を呼び出す
4. 任意のコードを実行できてしまう(ユーザパスワードを盗み出すなど)

注:実際に行うには Heap spraying などの高度な手法が必要

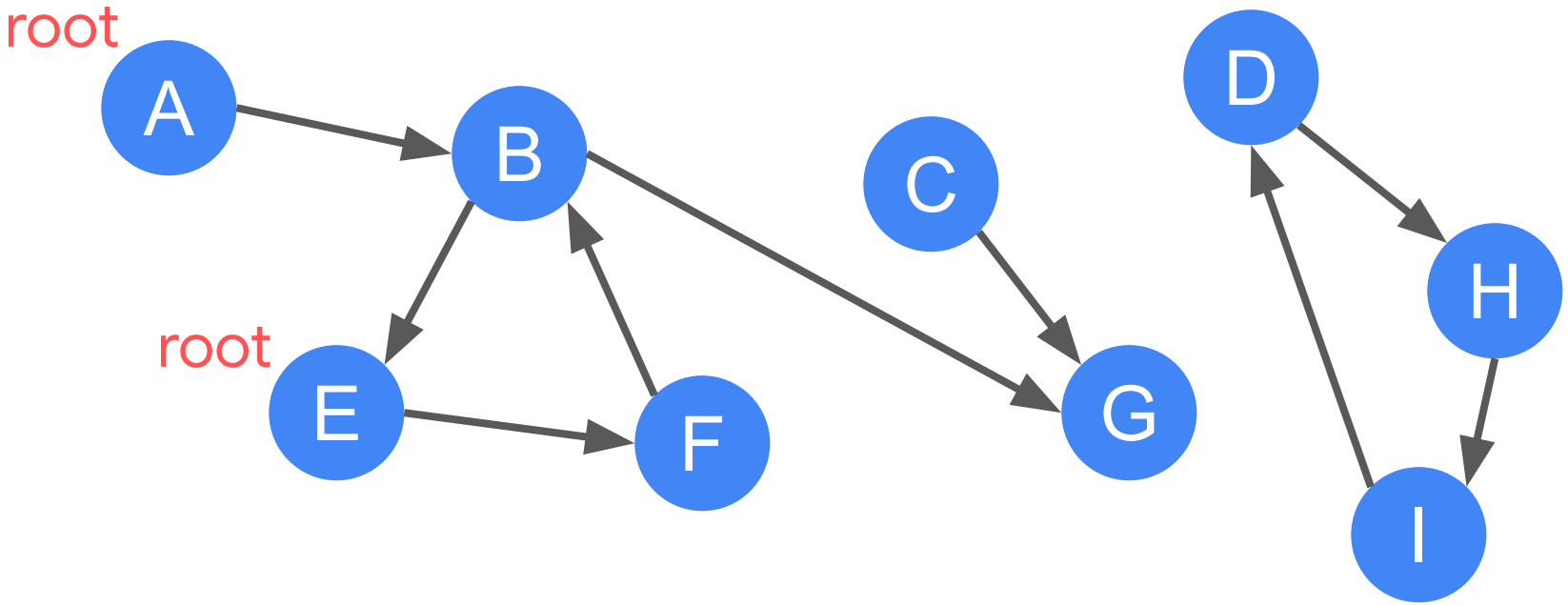
最も美しく正しい解決策：GC

- GC（ガベージコレクション）
  - メモリの自動管理
  - Java、Python、JavaScript などたいていの言語に搭載

- Chromium では Oilpan と呼ばれる C++ 用 GC を開発
  - オブジェクトを GC に移行した結果、Use-after-free の数が激減

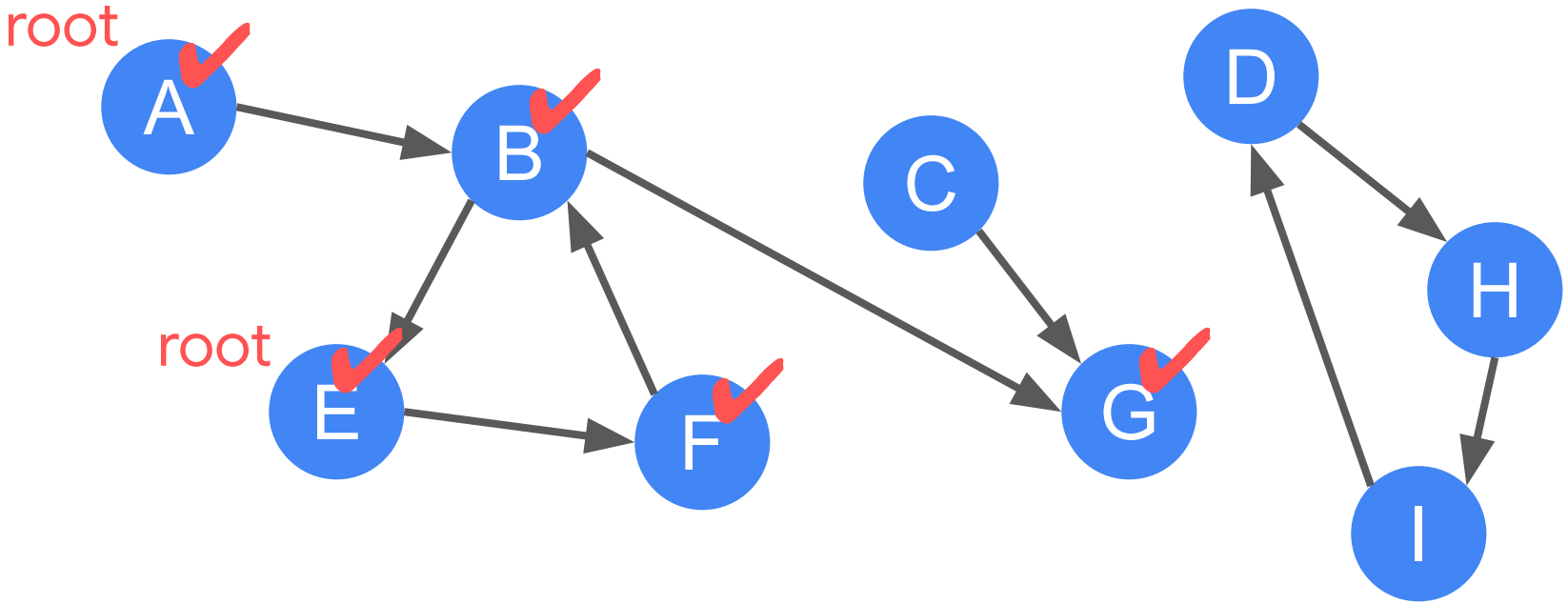
GCの基本：マーク & スイープ

- Step 1: ルートオブジェクト（グローバル変数、スタック上の変数など）から辿れるオブジェクトをマークしていく



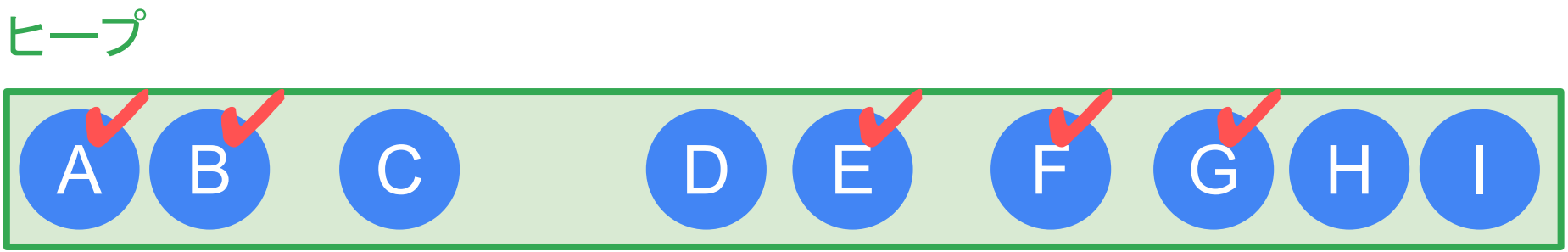
GCの基本：マーク & スイープ

- Step 1: ルートオブジェクト（グローバル変数、スタック上の変数など）から辿れるオブジェクトをマークしていく



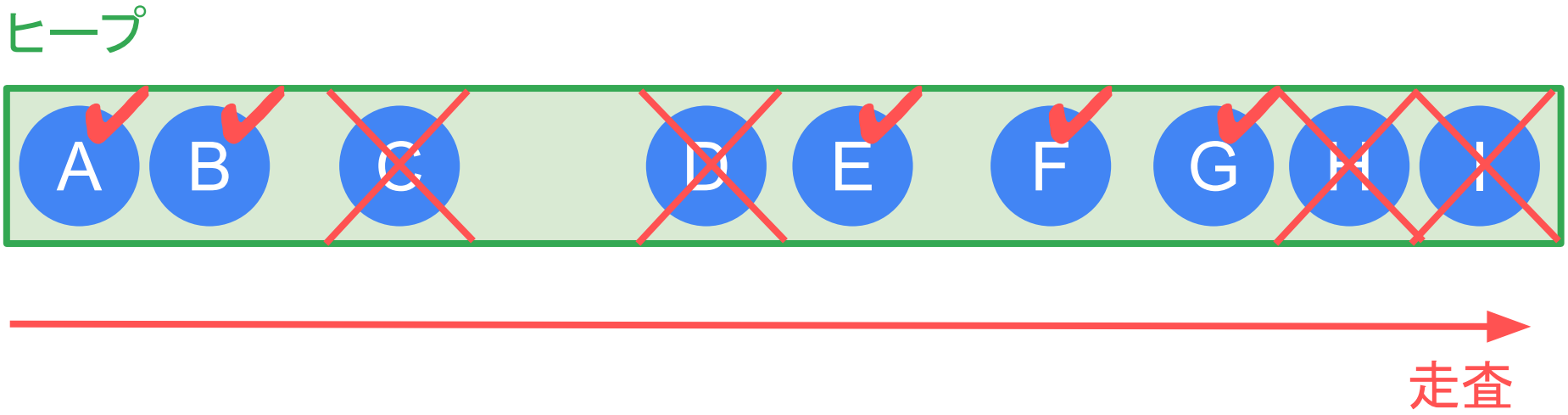
GCの基本：マーク & スイープ

- Step 2: ヒープを走査して、マークされていないオブジェクトを解放する（スイープ）



GCの基本：マーク & スイープ

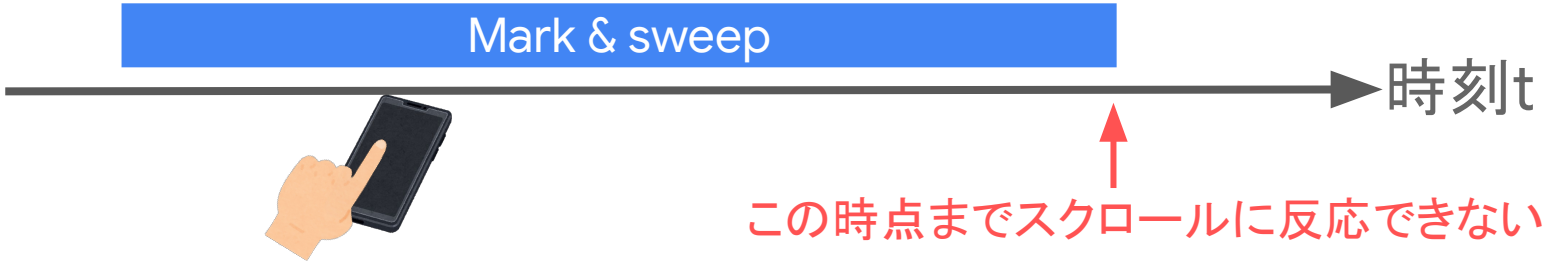
- Step 2: ヒープを走査して、マークされていないオブジェクトを解放する（スイープ）
  - reachable なオブジェクトは free されない => use-after-free が解決される



しかし話は終わらない（｜◡｜✿）

問題点：GC の停止時間が致命的

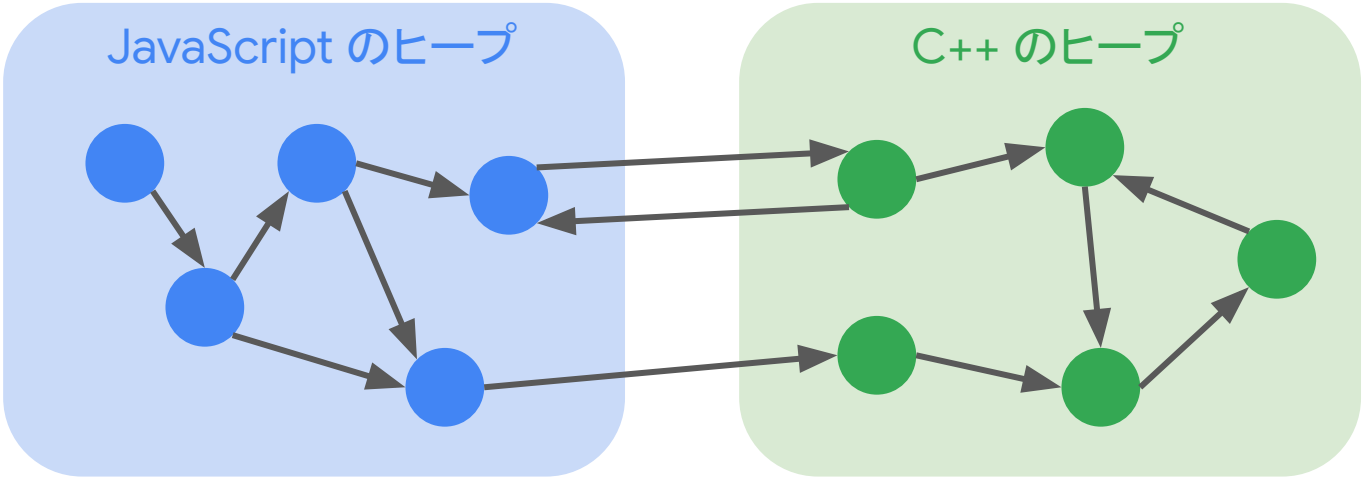
- ヒープサイズは ~GB、オブジェクト数は数百万個になる
- マーク & スイープをやっている間 Web サイトが止まる
  - 一昔前の Chrome で Gmail や Maps が「カクカク」した原因がこれ



- 解決策：インクリメンタル GC、コンカレント GC、世代別 GC などを実装して停止時間を大幅に短縮（詳細は田浦先生の講義）

問題点 : JavaScript と C++ の言語境界

- Web アプリは JavaScript で書かれている
- その Web アプリを実行する Chromium は C++ で書かれている
- その2つのオブジェクトグラフがお互いに参照している
  - 言語境界をまたがってGCを走らせる（論文出したよ）



GC できたよ！！
これで Use-after-free がゼロになる！！
(◍＞◡＜◍)

というほど、
この世界は単純ではない

GC が抱える問題点

- 数百万行のコードベースを全部 GC に対応させるエンジニアリングコストは非現実的

- 何が難しいのか？機械的に置き換えられないの？

GC が抱える問題点

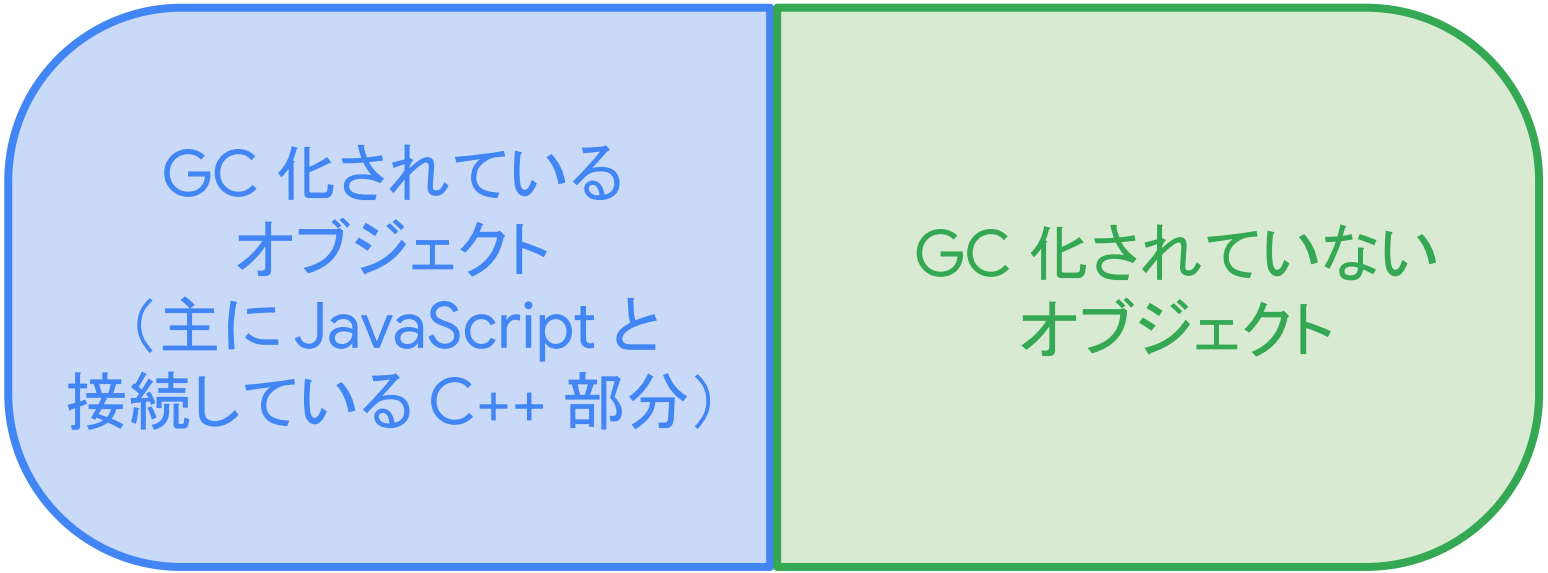
- 最大の難しさ: GC とデストラクタの相性が悪いこと
  - GC はデストラクタの順序を保証できない
  - ということは、デストラクタ内で他の GC オブジェクトに触るのはアウト
  - 実質的にコードベースからデストラクタを消し去る必要がある（大変）

```
class A : public GC {
};

class B : public GC {
  ~B() {
    a_->Clear(); // デストラクタの順序次第で a_ はもう解放されてるかもしれない
  }
  GCedPointer<A> a_;
};
```

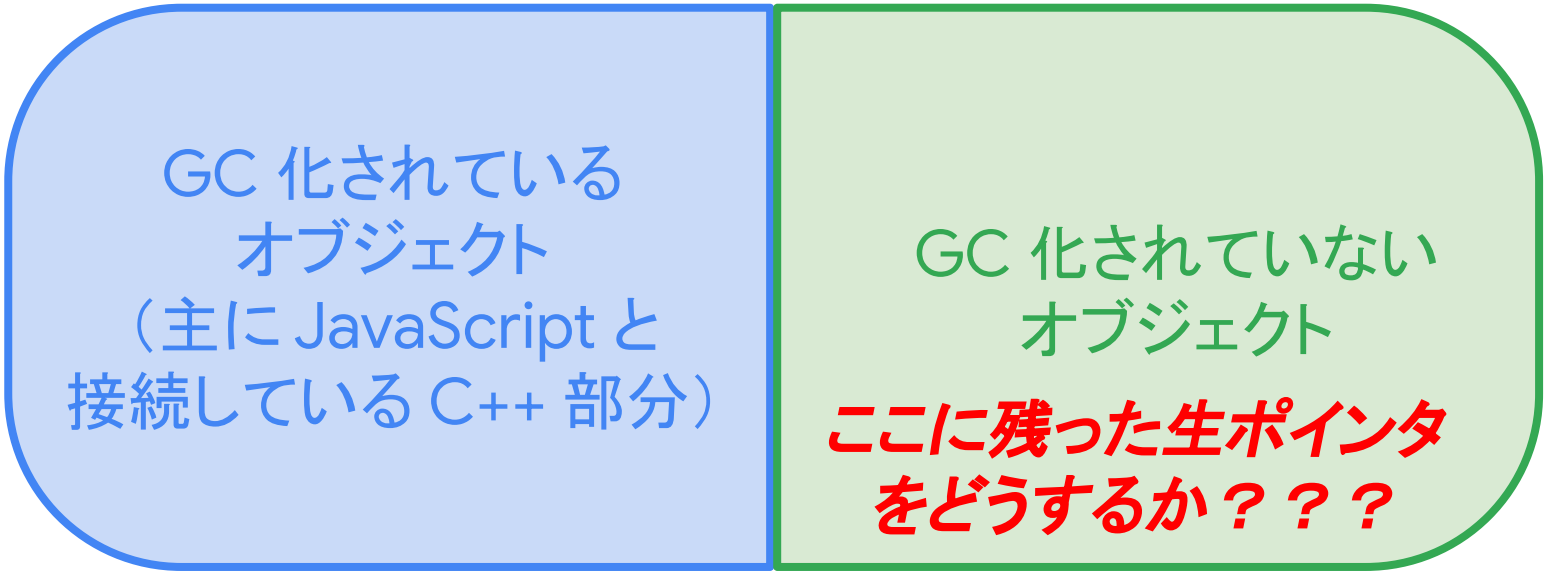
現実的な落としどころ

Chromiumのオブジェクト全体



現実的な落としどころ

Chromiumのオブジェクト全体



スマートポインタ（unique_ptr や scoped_refptr）は当然使っているが、
それで生ポインタがなくなるわけではない
（全部に使ったら参照サイクルが起きる）

ちょっと深いところに入ります

MiraclePtr（みらくる☆ぽいんた）の話

生ポインタの安全性をどう保証するか？

```
class A {
  A(B* b) : b_(b) { }
  void Foo() {
    b_->Bar();
  }
  B* b_; //生ポインタは常に危険を伴う
};
```

```
// 別のコード
std::unique_ptr<B> b = new B();
A* a = new A(b.get());
...
b = nullptr; // b を free
a->Foo(); // use-after-free!
```

生ポインタの安全性をどう保証するか？

- アイディア：生ポインタを参照カウントして、参照カウントが0になるまで free しなければいい

```
class A {
  A(B* b) : b_(b) { }
  void Foo() {
    b_->Bar();
  }
  MiraclePtr<B> b_;
};
```

```
// 別のコード
std::unique_ptr<B> b = new B();
A* a = new A(b.get());
...
b = nullptr;
a->Foo(); // use-after-freeが起きない
```

MiraclePtr<>のコンストラクタで参照カウント++
MiraclePtr<>のデストラクタで参照カウント--

参照カウントが0になるまで free を遅延させる

生ポインタの安全性をどう保証するか？

- Chromium の生ポインタ 15000 個を MiraclePtr に書き換えた
  - Use-after-free を 60 %削減！

- 性能とメモリへの影響は？
  - 12 個のアイディアからベストなものを選択
  - CPU命令レベルで徹底的に最適化

- MiraclePtrは「すでに存在している use-after-free からユーザを守る手段」にすぎない
  - use-after-free を検出して治す努力も必要

use-after-free を検出して治す努力

- PartitionAlloc
  - セキュリティ耐性と性能を大幅に強化したメモリアロケータ
  - malloc に対して 10 - 20 %以上のメモリ削減も達成
- Cluster Fuzz
  - 既存のテストケースや Web サイトのコードを字句に分割して、それらをランダムに結合して Use-after-free が起きるテストケースを自動生成してくれる
- Address Sanitizer
  - Valgrind の高速化版。Use-after-free、バッファオーバーフロー、メモリリークなどを検出可能（論文）。

use-after-free を検出して治す努力

- Chrome Vulnerability Reward Program
  - （Use-after-free などを利用して）Chrome の致命的なセキュリティ攻撃を発見すると、程度に応じて $500 ~ $150,000 をプレゼント！！(◍•ᴗ•◍)
  - これで生計を立てているハッカーもいる
  - 協力的なハッカーがほとんどで、攻撃手法と同時に解決策まで丁寧に教えてくれる優しい世界

レンダリングモデル

レンダリングエンジン

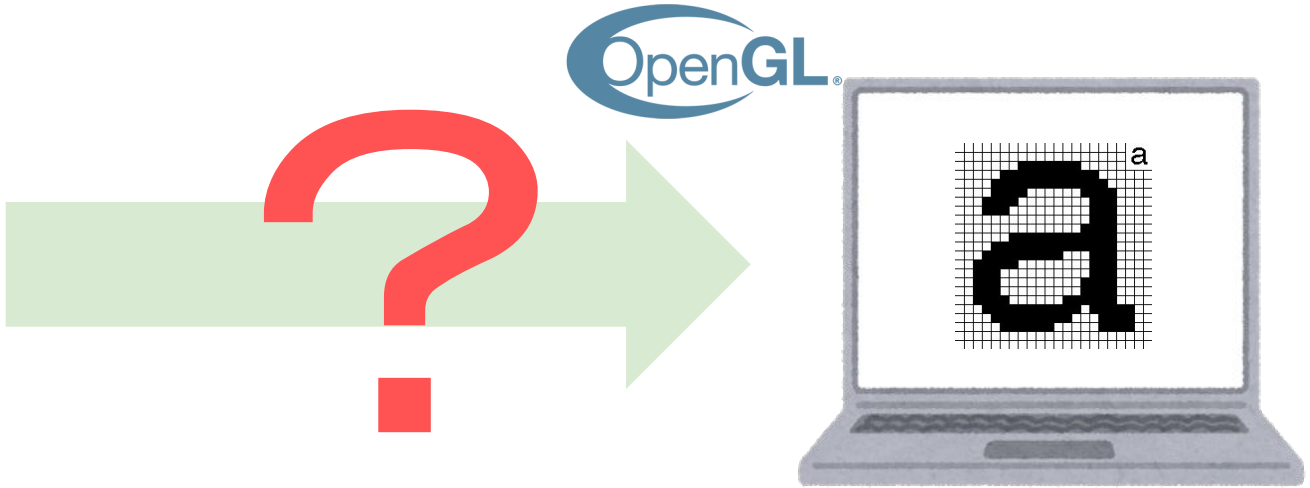
- Blink = Chromium のレンダリングエンジン
  - 「タブの中身」をレンダリングするソフトウェア
  - JavaScript の実行、HTML や CSS の解釈、画像描画など

レンダリングエンジン

- 「HTML や JavaScript のファイルがネットワークから受信されてから、Open GL で画面にピクセルが描かれるまでの壮大な物語」を 20 分で解説します(๑>◡<)ง

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta http-equiv="Content-Type" content=
            "text/html; charset=us-ascii" />
        <script type="text/javascript">
            function reDo() {top.location.reload();}
            if (navigator.appName == 'Netscape') {top.onresize = reDo;}
            dom=document.getElementById;
        </script>
    </head>
    <body>
    </body>
</html>
```



クイズ

- 一般的にソフトウェアが「サクサク」動いていると人間が感じるためには、毎秒何フレーム（Frame Per Second）で描画する必要があるでしょう？

(a) 10 FPS

(b) 30 FPS

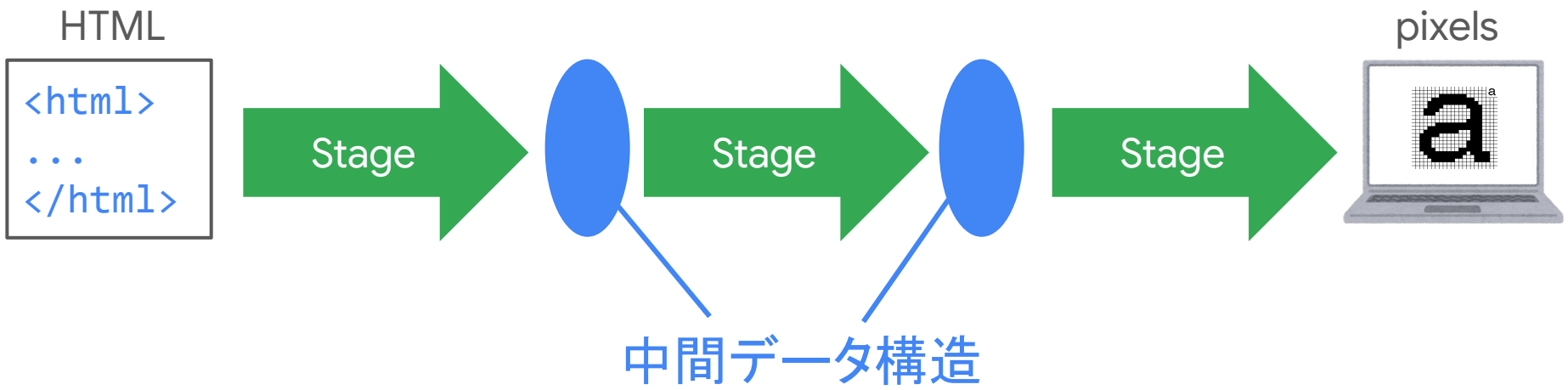
(c) 60 FPS

(d) 120 FPS

これを実現するために
いろんな工夫がいる

全体像

- 「ステージ」から「ステージ」へ中間データ構造を次々に渡していくことでレンダリングを進めていく（レンダリングパイプライン）



ステージ1: Parsing

- HTML ファイルを字句解析 => 構文解析

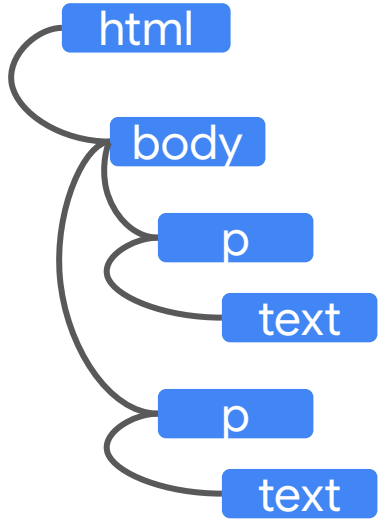
HTMLファイル

```
<html>
<body>
<p>aaa</p>
<p>bbb</p>
</body>
</html>
```

字句解析

```
"<" "html" ">"
"<" "body" ">"
"<" "p" ">"
"aaa"
"<" "/p" ">"
"<" "p" ">"
"bbb"
"<" "/p" ">"
"<" "/body" ">"
"<" "/html" ">"
```

構文解析



（構文木）

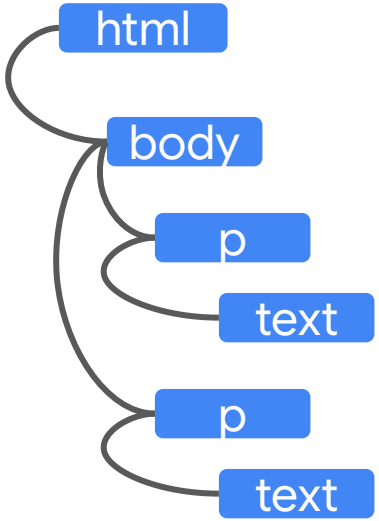
ステージ1：Parsing

- 入力：生の HTML ファイル
- 出力：DOM (Document Object Model) tree

HTMLファイル

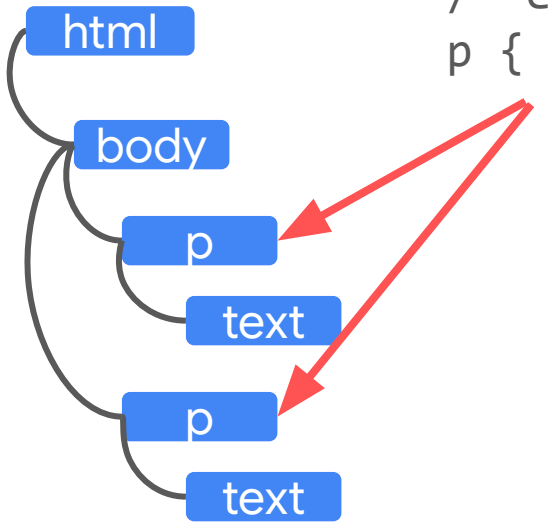
```
<html>
<body>
<p>aaa</p>
<p>bbb</p>
</body>
</html>
```

DOM tree



ステージ2：Style

- CSSのスタイルを適用する




```
/* every p has red text*/
p { color : red }
```

```
font-weight: bold;
```

hello

```
margin-left: 2cm;
```

hello

```
outline: dashed blue;
```

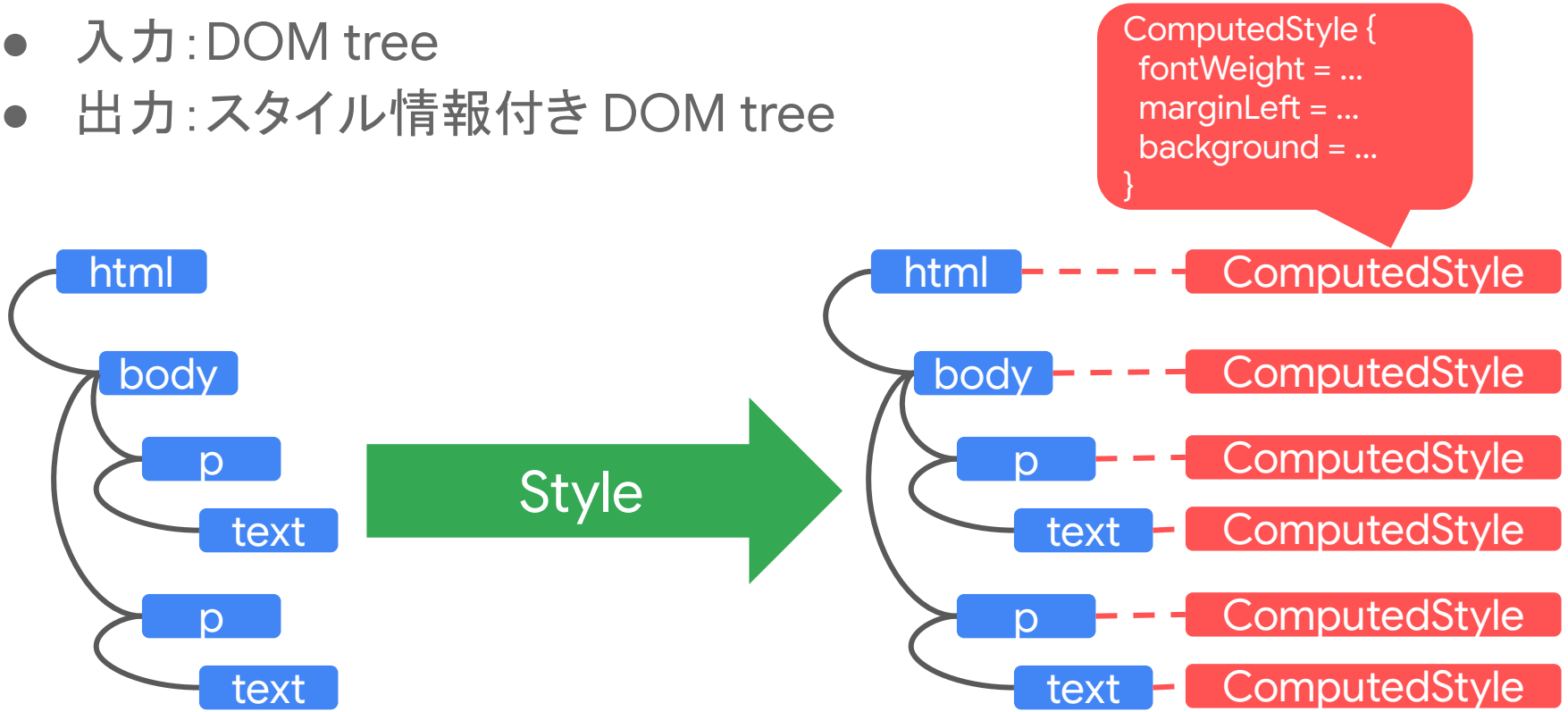
hello

```
transform: rotate(20deg);
```

hello

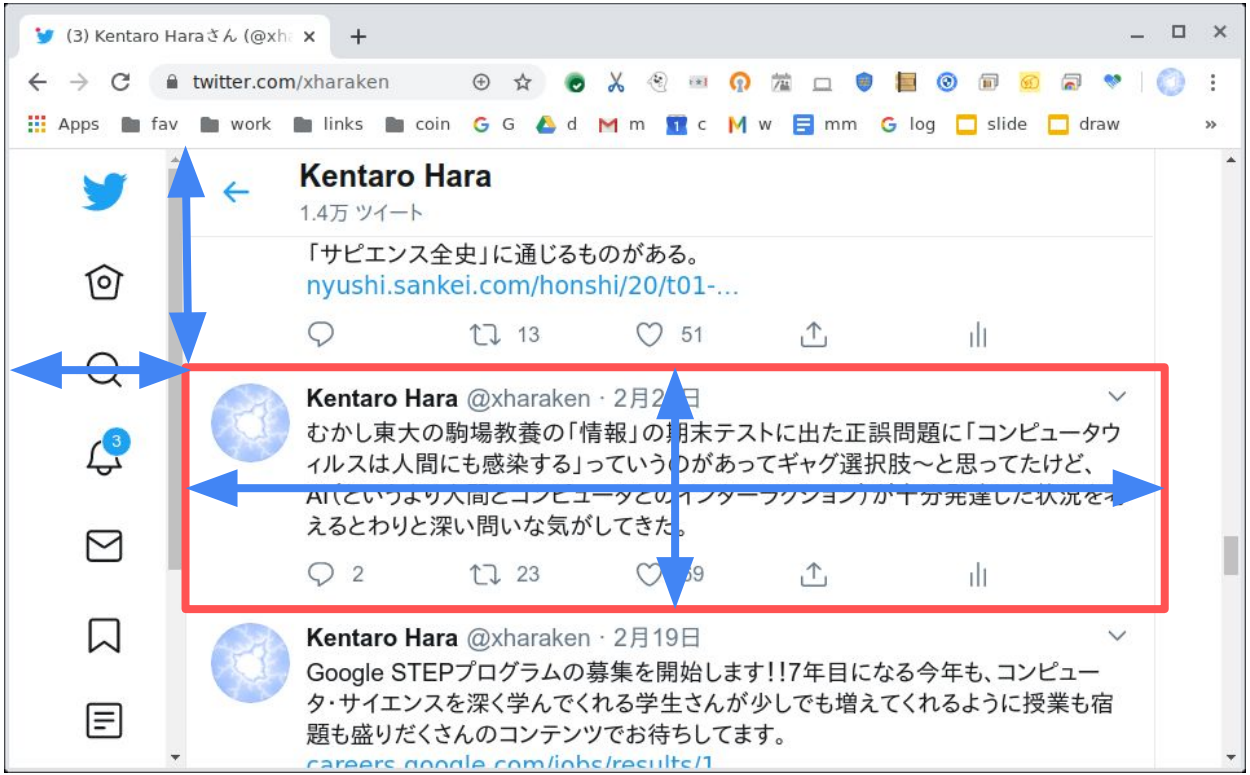
ステージ2：Style

- 入力：DOM tree
- 出力：スタイル情報付き DOM tree



ステージ3：Layout

- Layout = 各要素の位置座標を確定させること

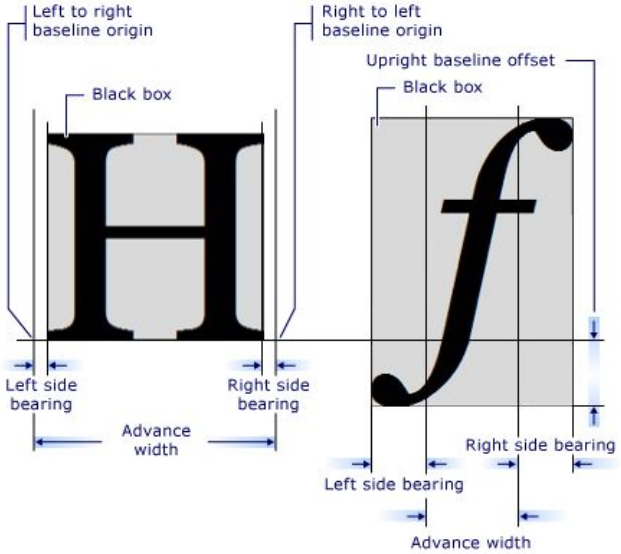


div

```
Layout {
x = 183
y = 250
width = 600
height = 140
}
```

ステージ3：Layout

- 要素の位置座標を決定するためには、文字グリフのサイズや改行位置なども決定する必要がある
  - HalfBuzz というオープンソースライブラリを利用



fi → ﬁ

fl → ﬂ

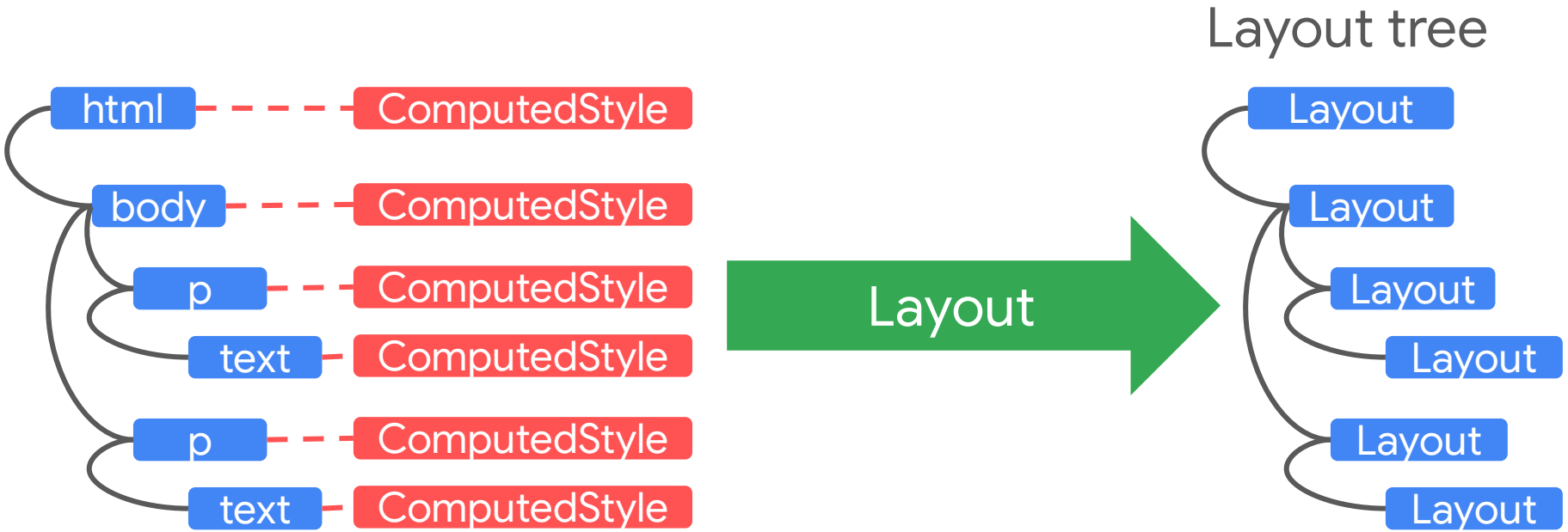
リガチャ

Is this a pen? No, it is a dog.

改行

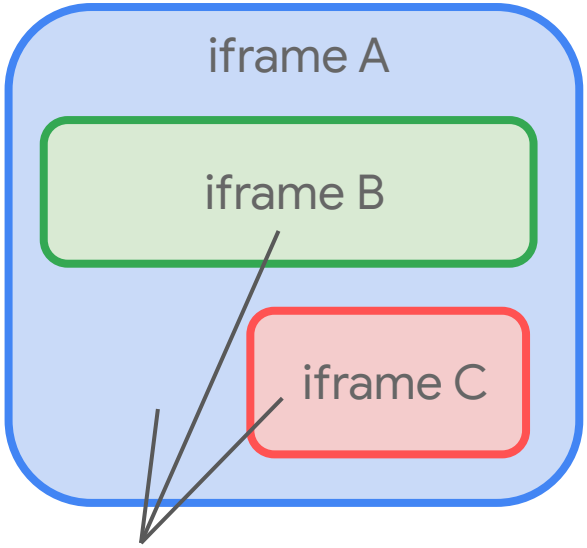
ステージ3 : Layout

- 入力 : スタイル情報付き DOM tree
- 出力 : Layout tree



ステージ4 : Layering

- Layering = 独立に描画可能な範囲をレイヤーに分割する作業

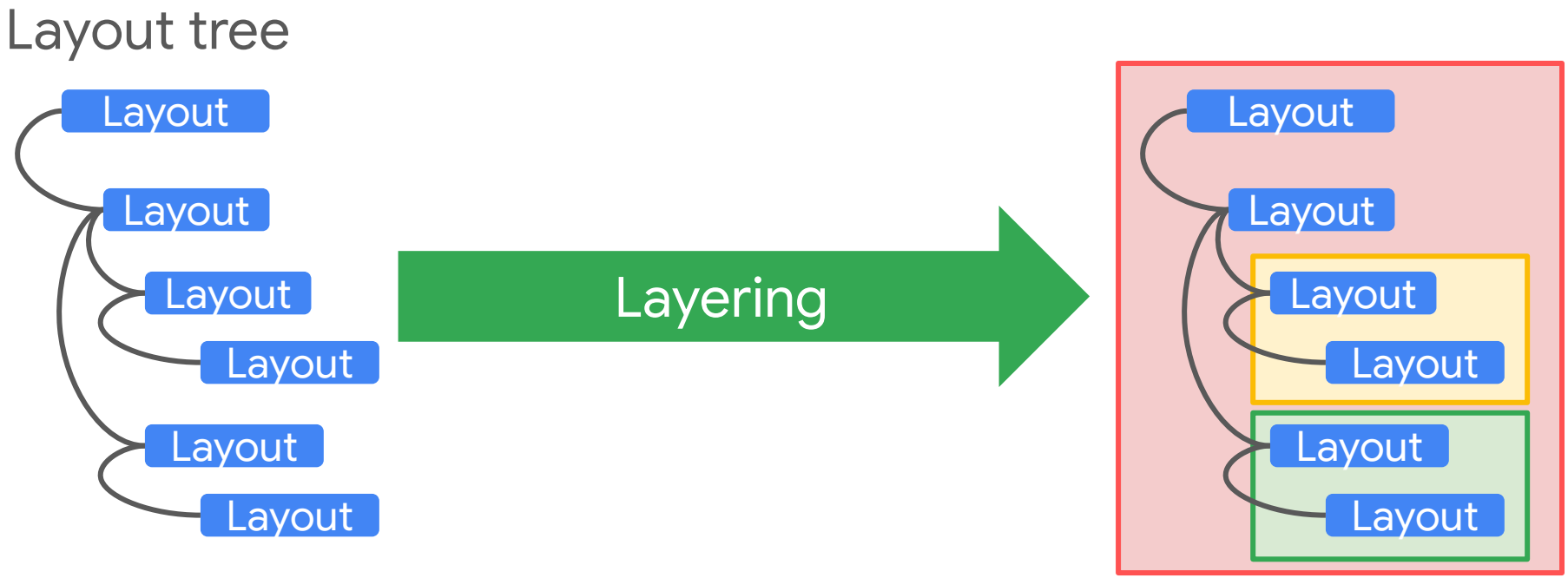


別レイヤーで描画可能

アニメーションする部分を
別レイヤーで動かす

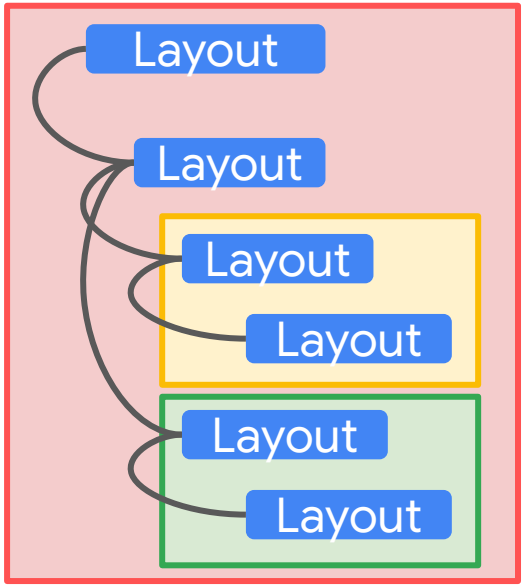
ステージ4 : Layering

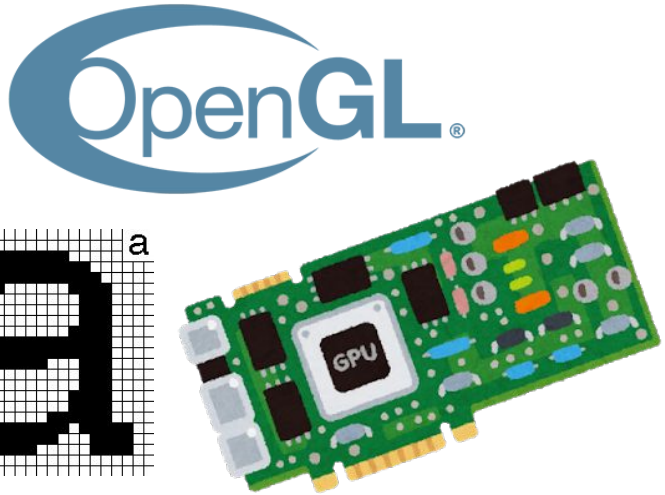
- 入力 : Layout tree
- 出力 : Layer



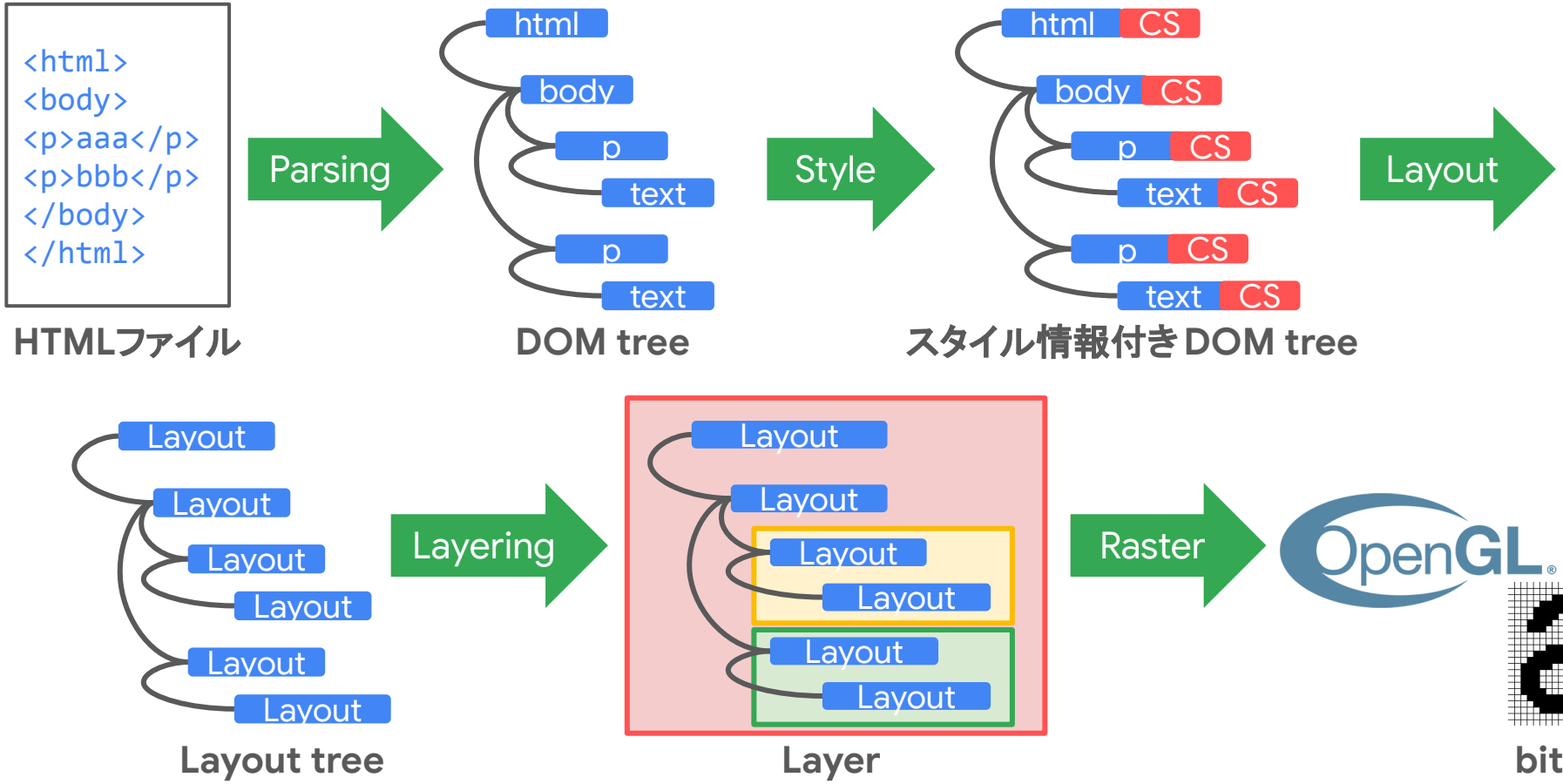
ステージ5 : Raster

- 入力 : Layer
- 出力 : OpenGL の描画コマンド
  - Skia というオープンソースライブラリを利用





全体像を振りかえる


<html>
<body>
aaa
bbb
</body>
</html>
Parsing
html
body
p
text
p
text
Style
CS
Layout
HTMLファイル
DOM tree
スタイル情報付きDOM tree
Layering
Raster
OpenGL
Layout tree
Layer
bitmap

なぜ「ステージ」から「ステージ」へ
中間データ構造を渡していくのか？

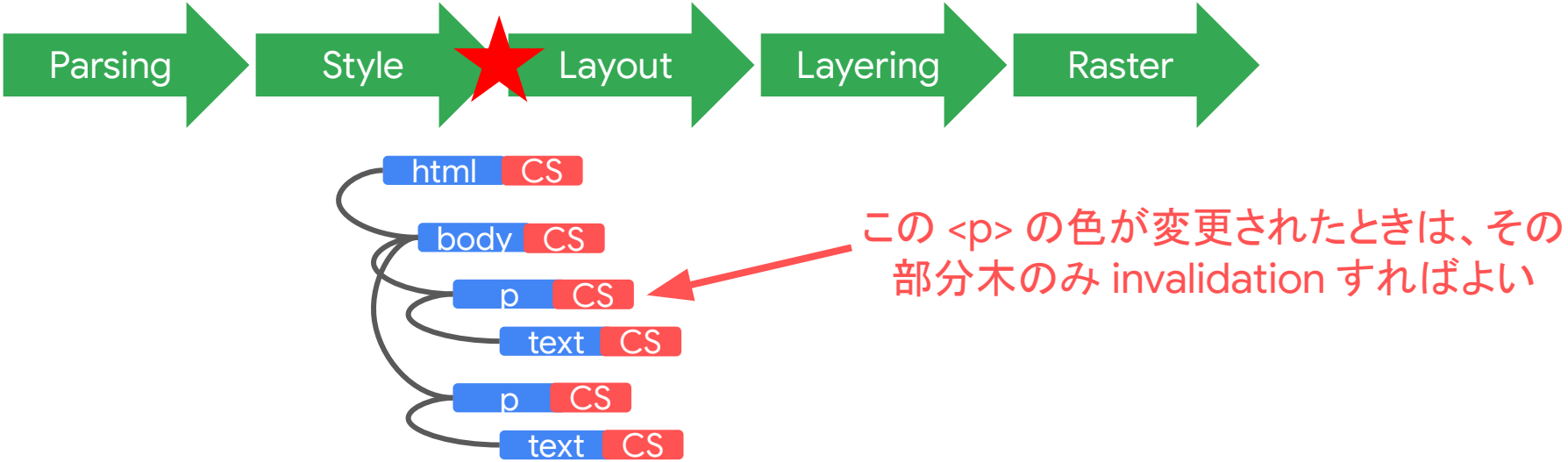
ステージングする理由1

- 動的レンダリングを高速化するため

Texのレンダリング	Webのレンダリング
静的	動的
レンダリングが完了して DVI ファイルができたらおわり	JavaScript が DOM を変更する JavaScript が CSS を変更する ユーザが画面をスクロールする アニメーション

ステージングする理由1

- ステージングすることで Partial invalidation が可能になる
  - 変更された部分（特定の部分木、特定のレイヤー etc）だけを無効化してその後の「ステージ」のみ実行しなおせばよい



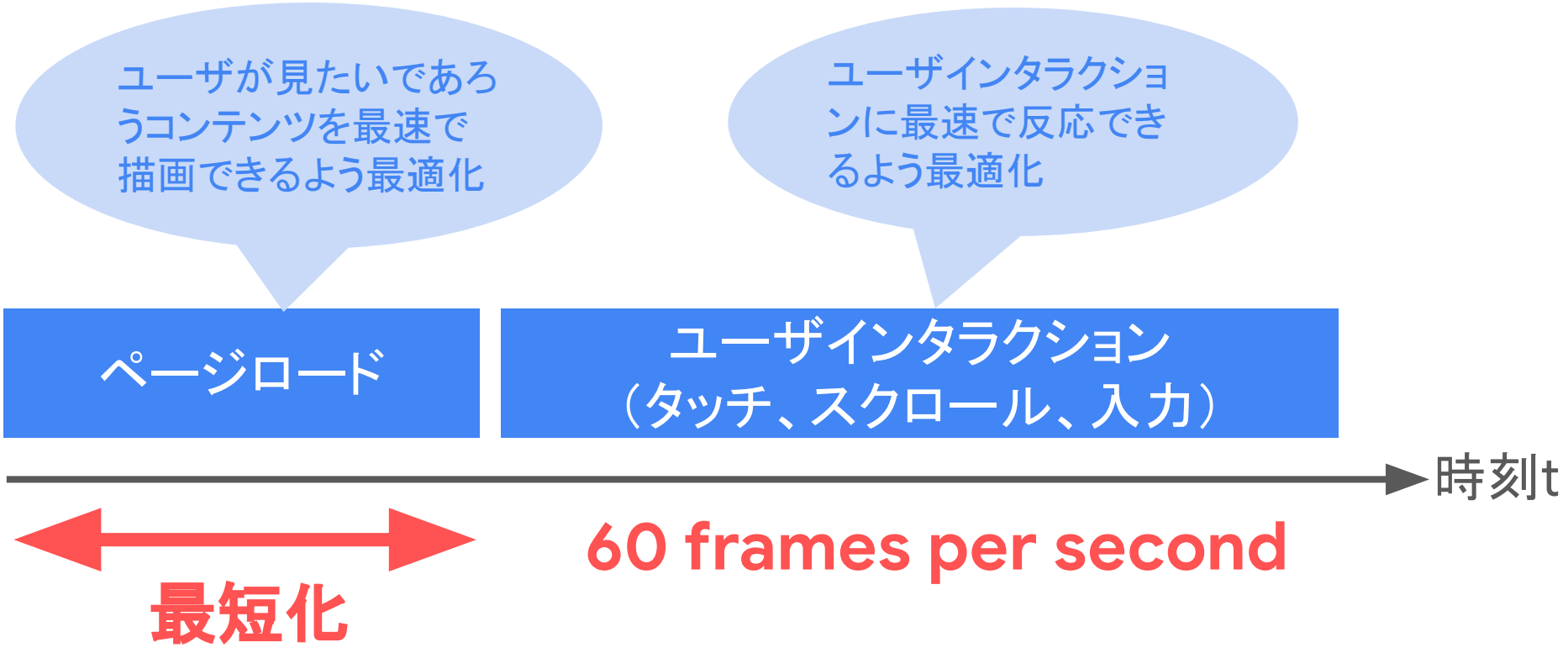
ステージングする理由1

- ステージングすることで Partial invalidation が可能になる
  - 変更された部分（特定の部分木、特定のレイヤー etc）だけを無効化してその後の「ステージ」のみ実行しなおせばよい

アニメーションの Layer のみ
Raster しなおせばよい

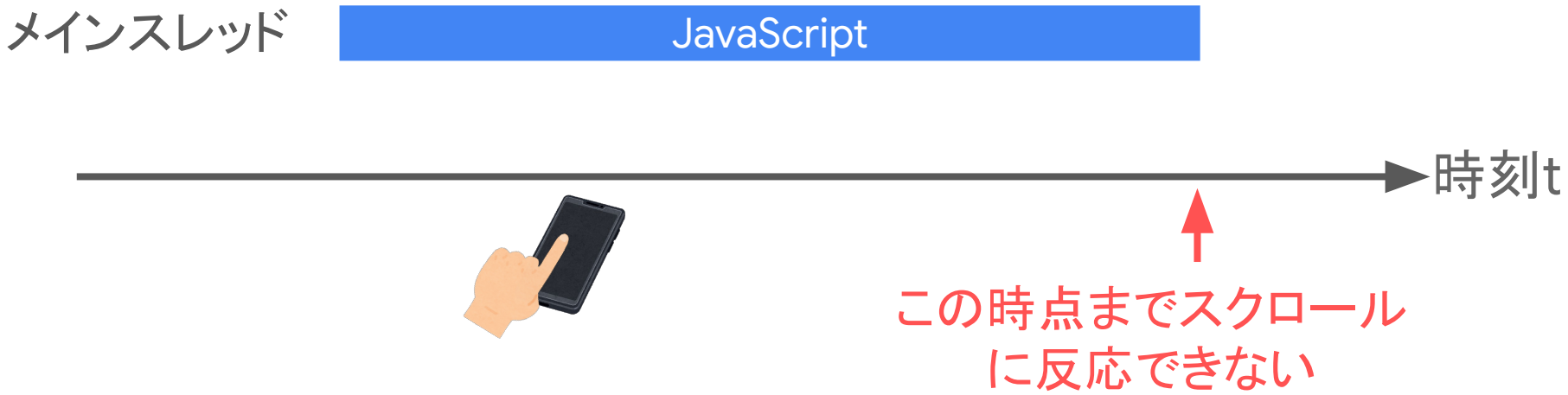
ステージングする理由2

- Web ページのライフサイクルと性能上の目標



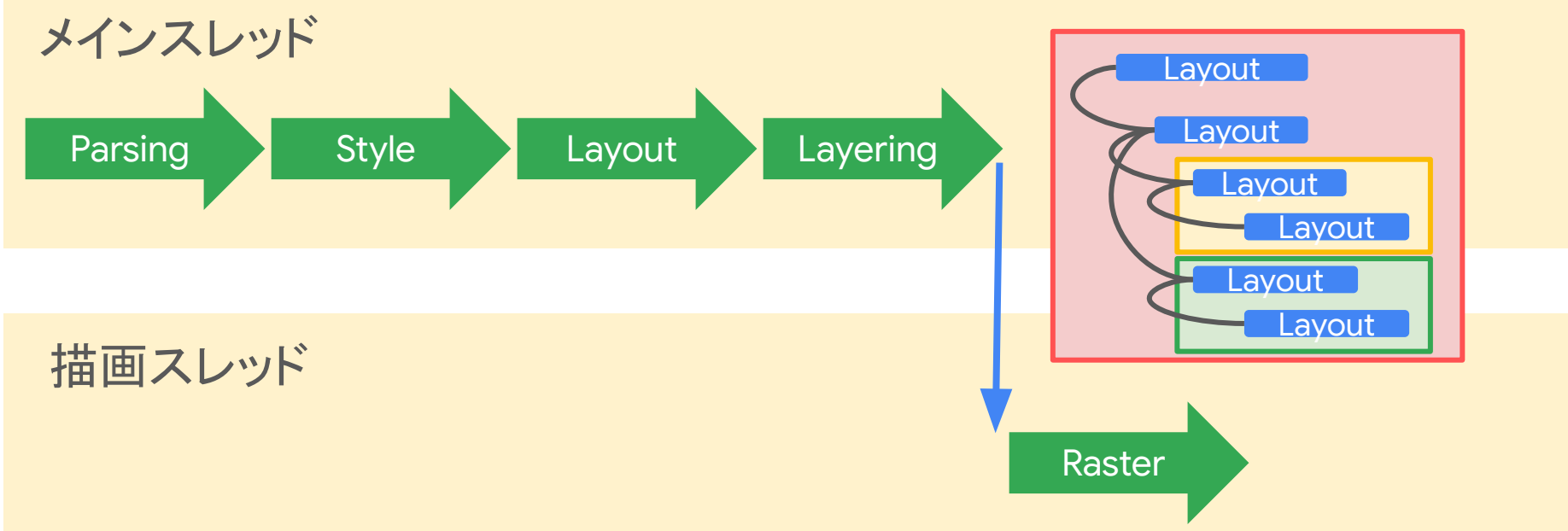
ステージングする理由2

- 問題点：メインスレッドが重たい JavaScript や GC を走らせていると、その間スクロールなどを処理できなくなる
  - 60 FPS が破綻 => 「カクカク」(´･ω･`)



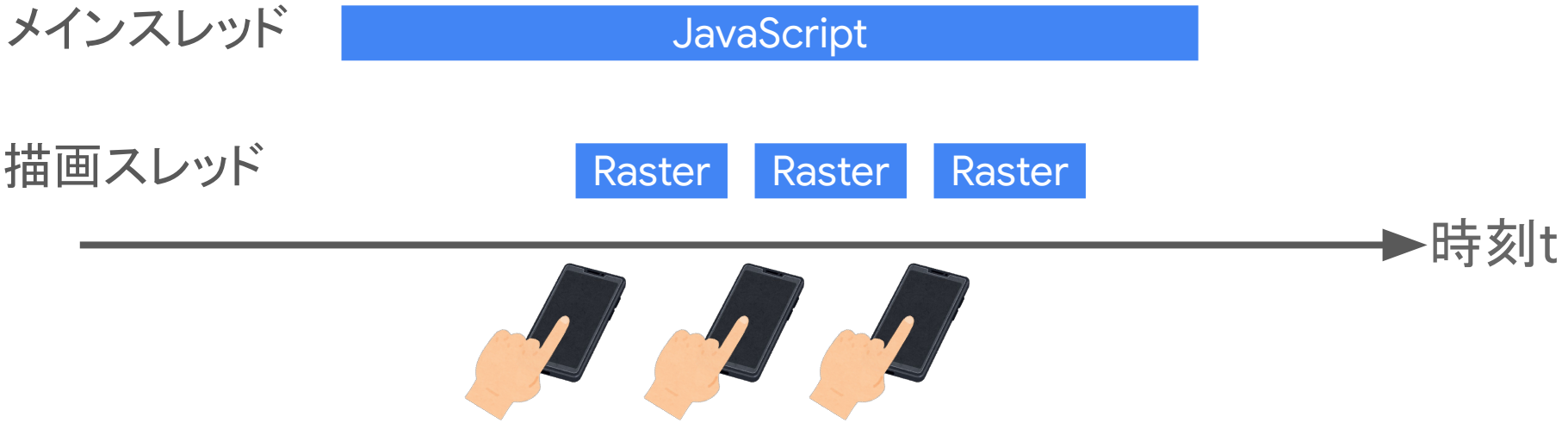
ステージングする理由2

- 中間データ構造の Layer を描画スレッドに送ることで、描画スレッドだけで Raster できるようにする



ステージングする理由2

- スクロール ＝ Layer 内の相対座標が変化するだけなので、描画スレッドだけで処理可能 => 「サクサク！！」(๑＞◡＜๑)



まとめ

ブラウザ = ただのブラウザアプリ

ではなく、もはや

ブラウザ = Web の OS

- プロセス管理
- メモリ管理
- レンダリング
- CPU タスクスケジューラ
- ファイル管理（ WebStorage etc）
- デバイス入出力（ WebAudio, WebUSB etc）
- アセンブリ（ WebAssembly）
- …

古典的な OS との決定的な違い ＝ Embeddability

Web 上に転がっているいろんな人や組織が作った
（よって信用を仮定できない）アプリをセキュアにホストする

（twitter.com はそれがどのページにどう埋め込まれようと、
twitter.com は twitter.com のデータにしかアクセスできないし、
twitter.com のデータは twitter.com にしかアクセスされないことを保証）

古典的な OS との決定的な違い ＝ Embeddability

Web 上に点がっているいろんなヤツや組織が作った
（よって信用を仮定できない）アプリをセキュアにホストする

（twitter.com よそからがどのページにどう埋め込まれようと、
twitter.com は twitter.com のデータにしかアクセスできないし、
twitter.com のデータは twitter.com にしかアクセスされないことを保証）

システム屋にはたまらなくおもしろい！

より深く学びたい人へ

- Chrome University YouTube channel（Chrome チームに入ってきた人向けのトレーニング資料）
- Chromium Browser Advent Calendar（Chrome Tokyoチームがやっている各種プロジェクトのお話）
- Code of conduct（開発時の心構え）
- Chromium project page

レポート課題

レポート課題

5. 評価

- 出席およびレポート
- レポートは全体(13回)から2回の講義を選んで執筆する. それぞれA4 5ページ程度のレポートにまとめる. 講義内容の簡単なまとめと, 自分なりの考察, 調査を行う.

- 方向性が決まっていたほうが書きやすいと思うので･･･テーマ課題を出しておきます！

Google のミッション：

世界中の情報を整理し、
世界中の人々がアクセスできて
使えるようにすること

レポート課題

- インターネット黎明期（1990年代〜2000年代）には情報やコンテンツは《Web×オープンな世界》に向かっていた

- ネイティブアプリ（Android アプリ、iOS アプリ）が出現すると情報は《ネイティブアプリ×クローズドな世界》に向かうようになった
  - LINE、WeChat、WhatsApp、Facebook etc
  - 情報が各ネイティブアプリ内部に囲い込まれる

- Web => ネイティブアプリへのシフトはとくに途上国で著しい
  - スマホ使用時間の何%を Web に使ってますか？

レポート課題

- 以下の3点についてあなたの考えを具体的に論じてください

1. なぜ情報は《Web×オープンな世界》から《ネイティブアプリ×クローズドな世界》に向かうようになったのか
2. Google のミッションを達成する観点から考えた場合、情報が《ネイティブアプリ×クローズドな世界》に向かうことは望ましいか、望ましくないか
3. 再び情報が《Web×オープンな世界》に向かうようになるために、Web に欠けているものは具体的に何だと思うか

Q&A