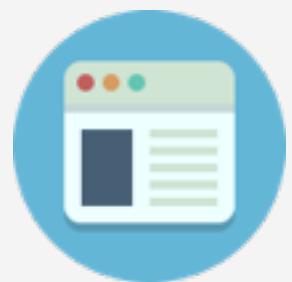


Progressive Web Apps の導入基礎

Recruit Sumai Company Ltd. / Yusuke Katayama

片山 雄介 (Yusuke Katayama)

株式会社リクルート住まいカンパニー
新テクノロジー商品開発チーム チームリーダー



Web / ブラウザ 等の最新技術を活用した
新たなカスタマーエクスペリエンスの企画・開発



SUUMOのスマホサイトを中心とした
UX Design / Interaction Design



VRを利用した商品の開発



SUUMOについて

全国のマンション、一戸建て、土地などの販売・売却情報から賃貸・リフォームまで、住まいに関するあらゆる情報が満載の「**不動産総合ポータルサイト**」



本日のコンテンツ



Add to
Home Screen



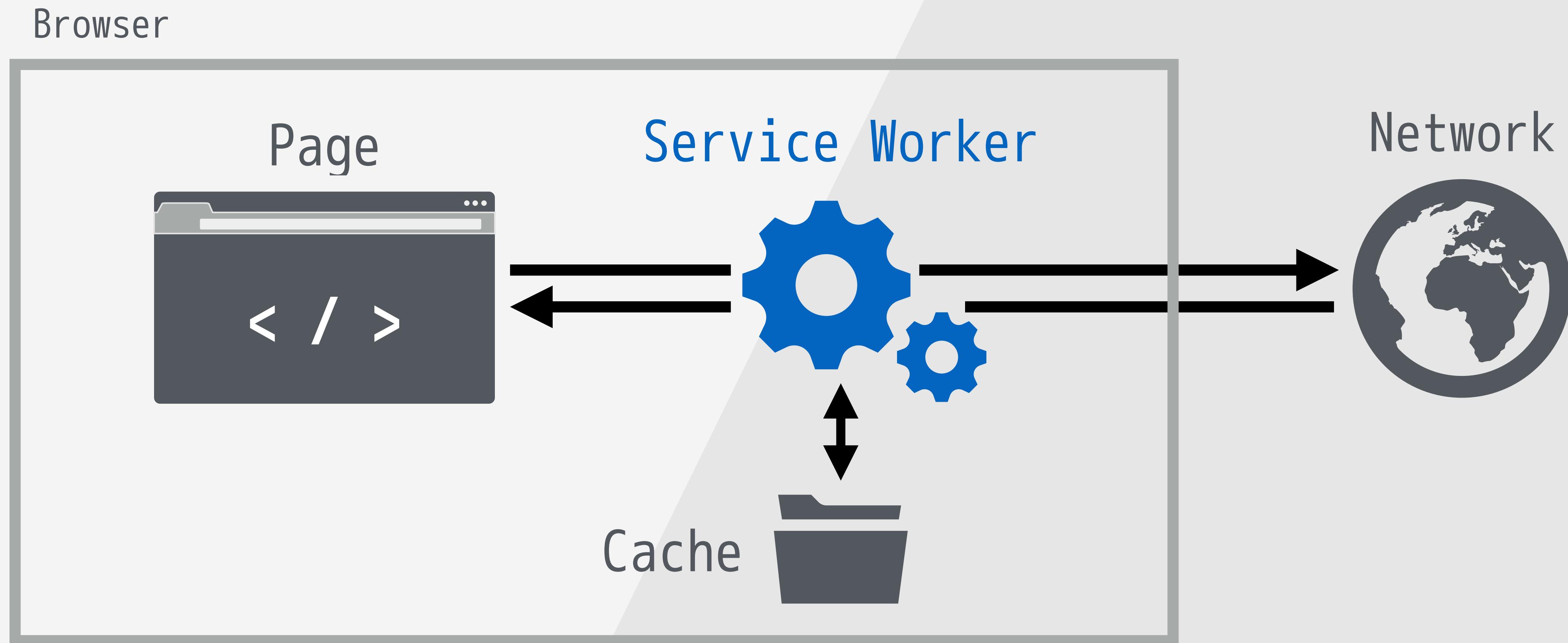
Push Notifications



Offline Cache
(+ App Shell)

Service Worker とは・・・

ブラウザのバックグラウンドで動くjavascript実行環境で、ネットワークプロキシとして動作することができるもの。





Add to Home Screen



Add to Home Screen – 実施の背景



Webユーザーはアプリユーザーよりも、
他の競合サービスへ横流れする可能性が高い傾向。

また、アプリをインストールせず、
Web のみで家探しをするユーザーも多い。

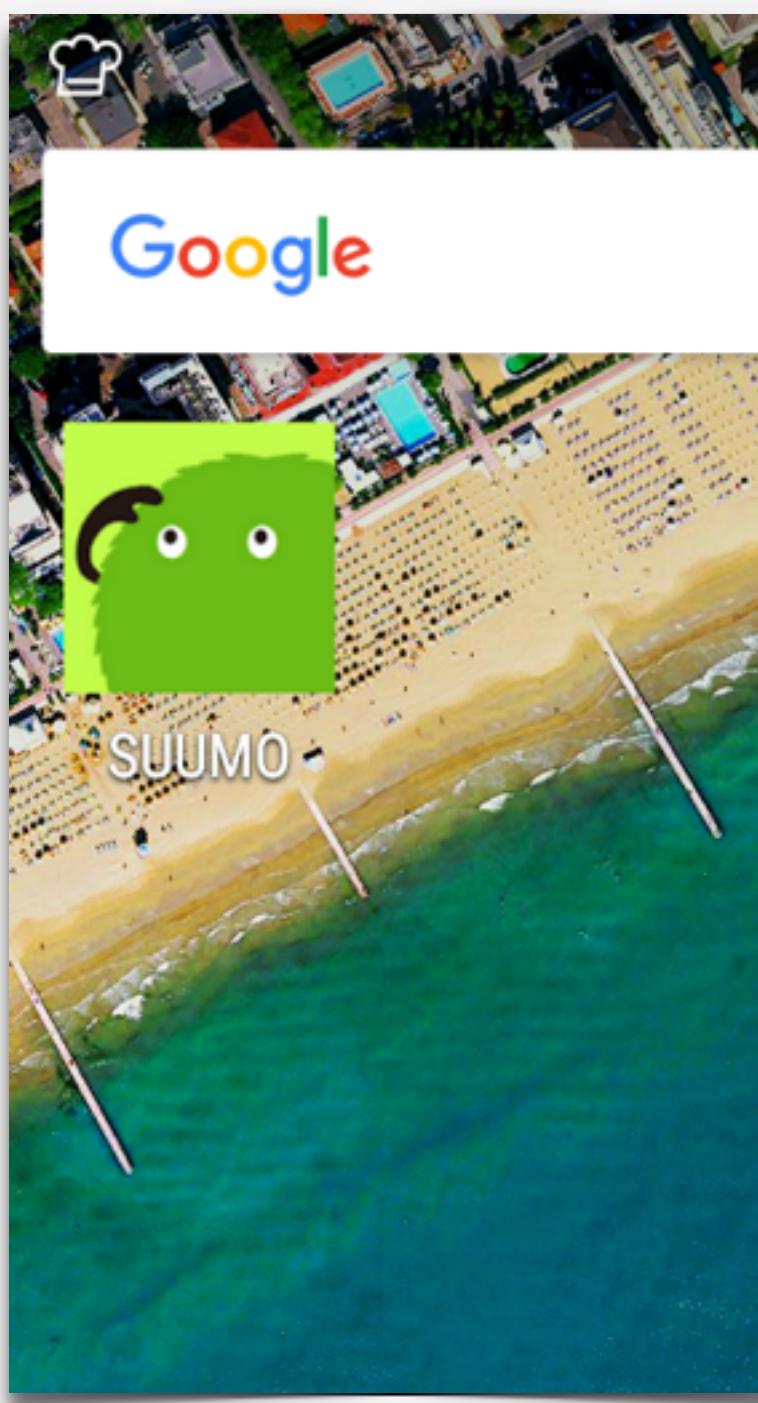
Webユーザーに SUUMOをメインサービス として利用していただき、
他サービスへの横流れを防ぐため、Add to Homescreen を実施。

Add to Home Screen – 施策内容

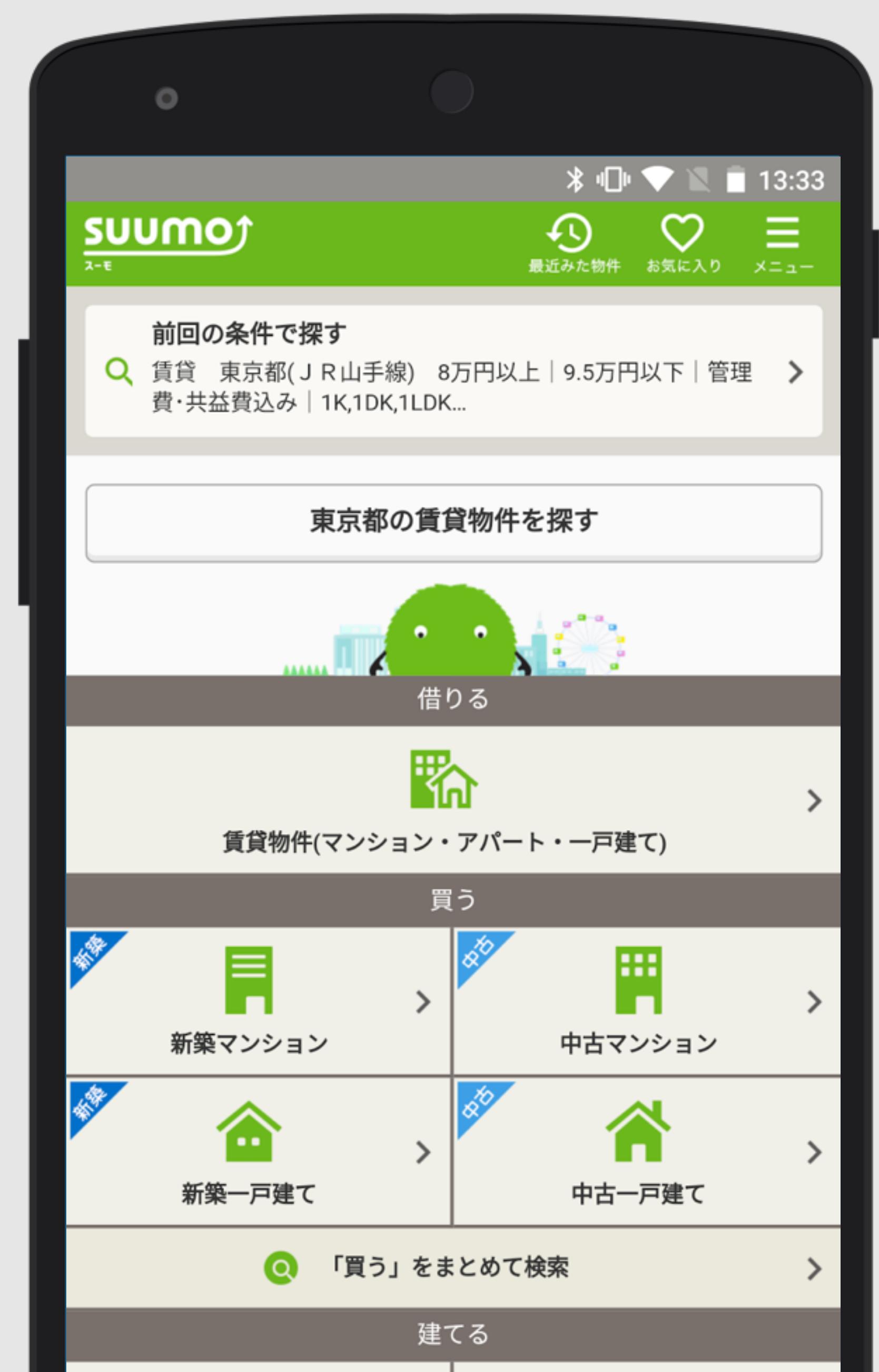
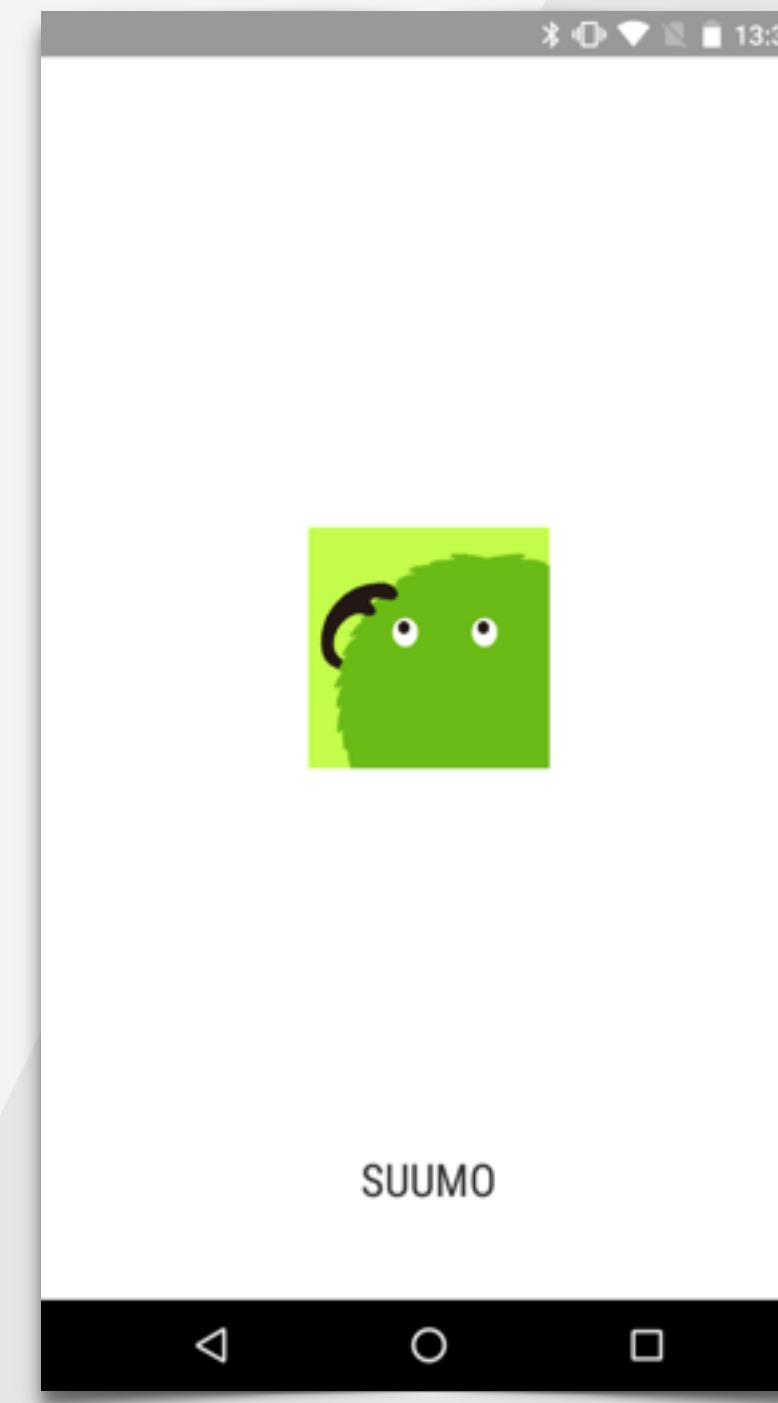
SUUMOに再来訪時に
プロンプト出現



ホーム画面



スプラッシュ画面



Add to Home Screen – 施策内容

直前の検索履歴に応じてパーソナライズ
すばやく目的のコンテンツへ
※ **LocalStorage** を利用

通常のTOPページと同様に、
総合TOPとしての導線を配置



Add to Home Screen の実装方法



- ① manifestファイルを用意する ※ 詳細は後述
- ② <head> 内で manifestファイルを読み込む

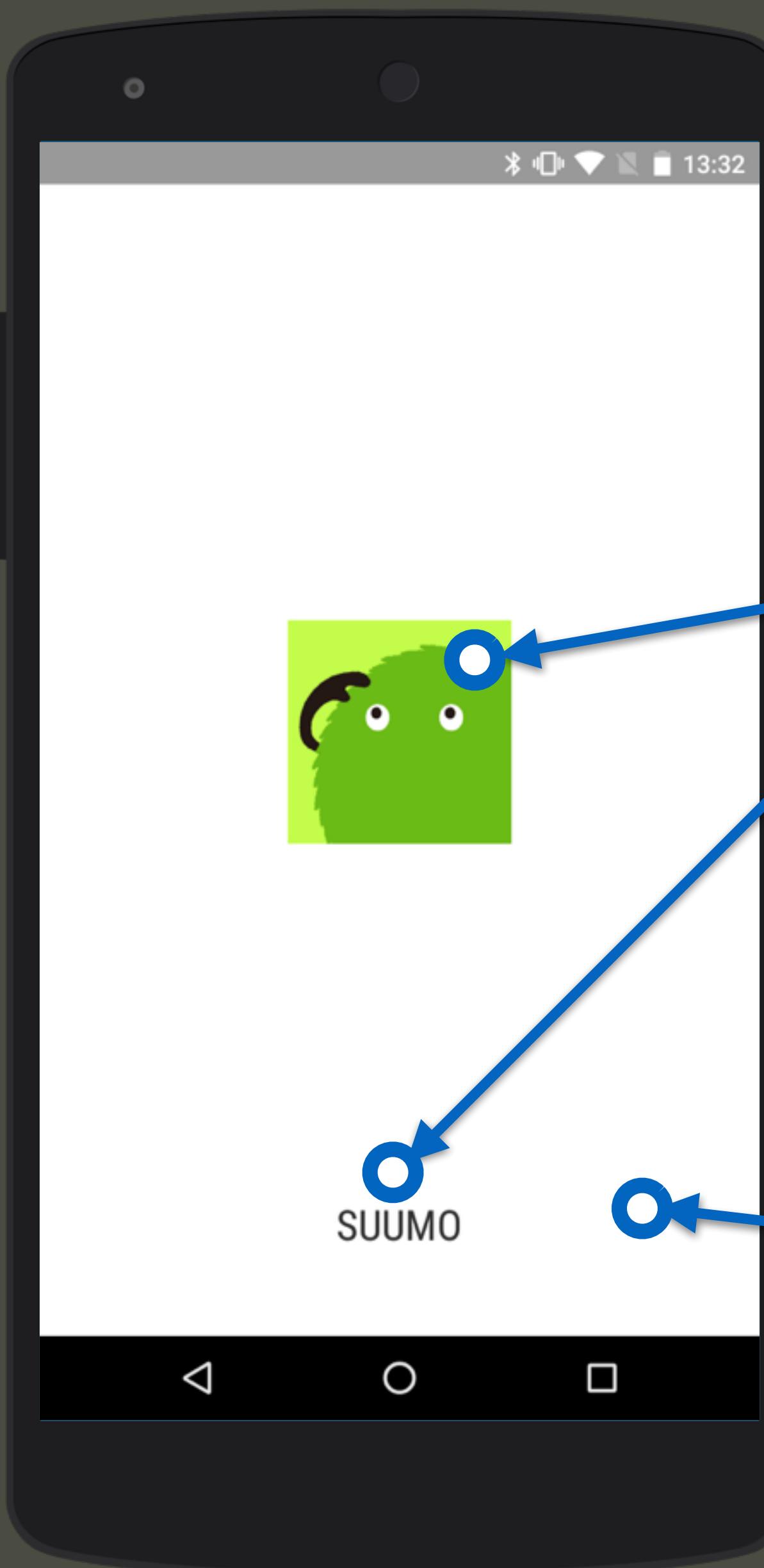
```
<head>
  <link rel="manifest" href="/manifest.json">
</head>
```

- ③ Service Worker (sw.js とする) の登録処理を記載

※ sw.js は空のファイルでOK

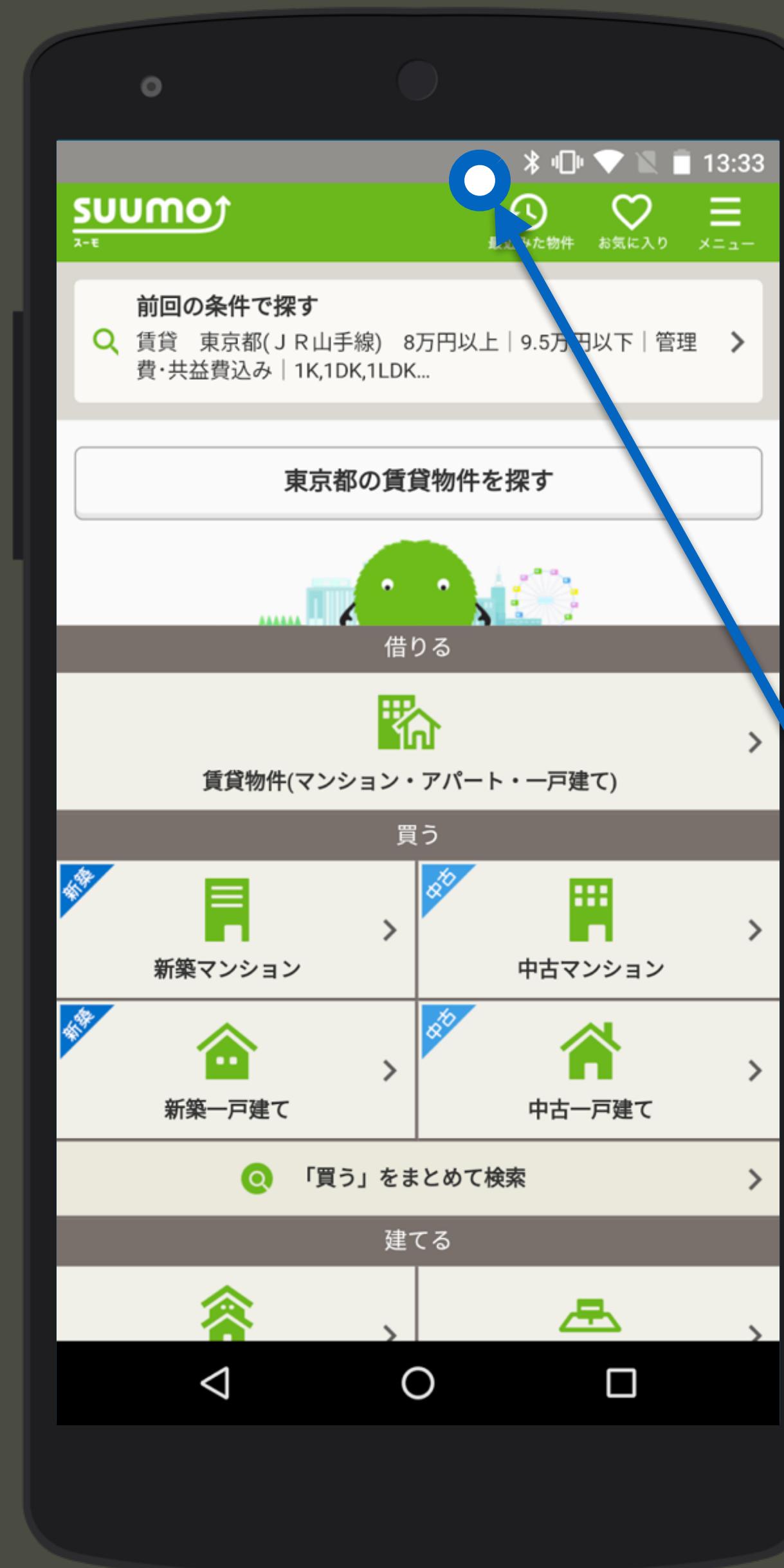
```
navigator.serviceWorker.register('/sw.js')
  .then(function(registration) { // 登録成功 })
  .catch(function(err) { // 登録失敗 });
```

■ manifest.json



```
{  
  "lang": "ja",  
  "name": "SUUMO",           # アプリ名  
  "short_name": "SUUMO",     # ホーム画面などの表示名  
  "icons": [  
    {  
      "src": "/icon-144x144.png",  
      "sizes": "144x144",  
      "type": "image/png"  
    }, {  
      "src": "/icon-192x192.png",  
      "sizes": "192x192",  
      "type": "image/png"  
    }],  
  "background_color": "#FFFFDD", # スplash画面の背景色  
  "theme_color": "#FFFFDD",     # アドレスバーの色  
  "start_url": "/homescreen",   # 起動する画面のURL  
  "display": "standalone"       # アドレスバーの表示有無  
}
```

■ manifest.json



```
{  
  "lang": "ja",  
  "name": "SUUMO",           # アプリ名  
  "short_name": "SUUMO",     # ホーム画面などの表示名  
  "icons": [<{br/>    "src": "/icon-144x144.png",  
    "sizes": "144x144",  
    "type": "image/png"  
  }, {  
    "src": "/icon-192x192.png",  
    "sizes": "192x192",  
    "type": "image/png"  
  }],  
  "background_color": "#FFFFDD", # スplashscreen画面の背景色  
  "theme_color": "#FFFFDD",    # アドレスバーの色  
  "start_url": "/homescreen", # 起動する画面のURL  
  "display": "standalone"     # アドレスバーの表示有無  
}
```

```
var deferredPrompt;

window.addEventListener('beforeinstallprompt', function(e) {
  e.preventDefault();
  deferredPrompt = e;
  return false;
});

btnSave.addEventListener('click', function() {
  if(deferredPrompt !== undefined) {
    deferredPrompt.prompt();
    deferredPrompt.userChoice.then(function(choiceResult) {
      if(choiceResult.outcome == 'dismissed') { // インストールをキャンセル }
      else { // インストール実施 \(^o^)/ }

      deferredPrompt = null;
    });
  }
});
});
```

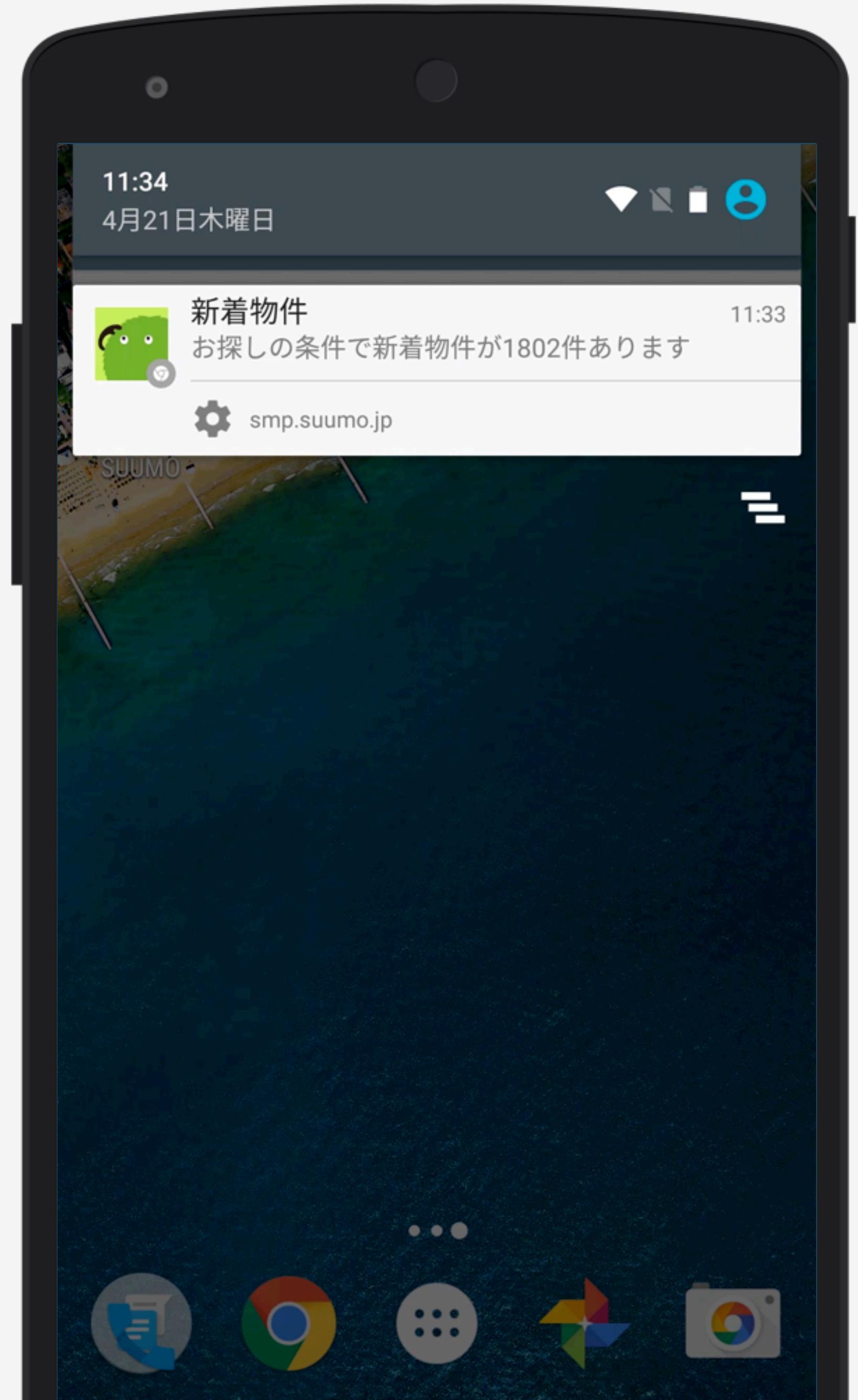
プロンプトを任意のタイミングで表示

ボタンをクリックした際に
プロンプトを表示させる

参考) <https://developers.google.com/web/fundamentals/engage-and-retain/app-install-banners/advanced?hl=en>



Push Notifications



Push Notification – 実施の背景

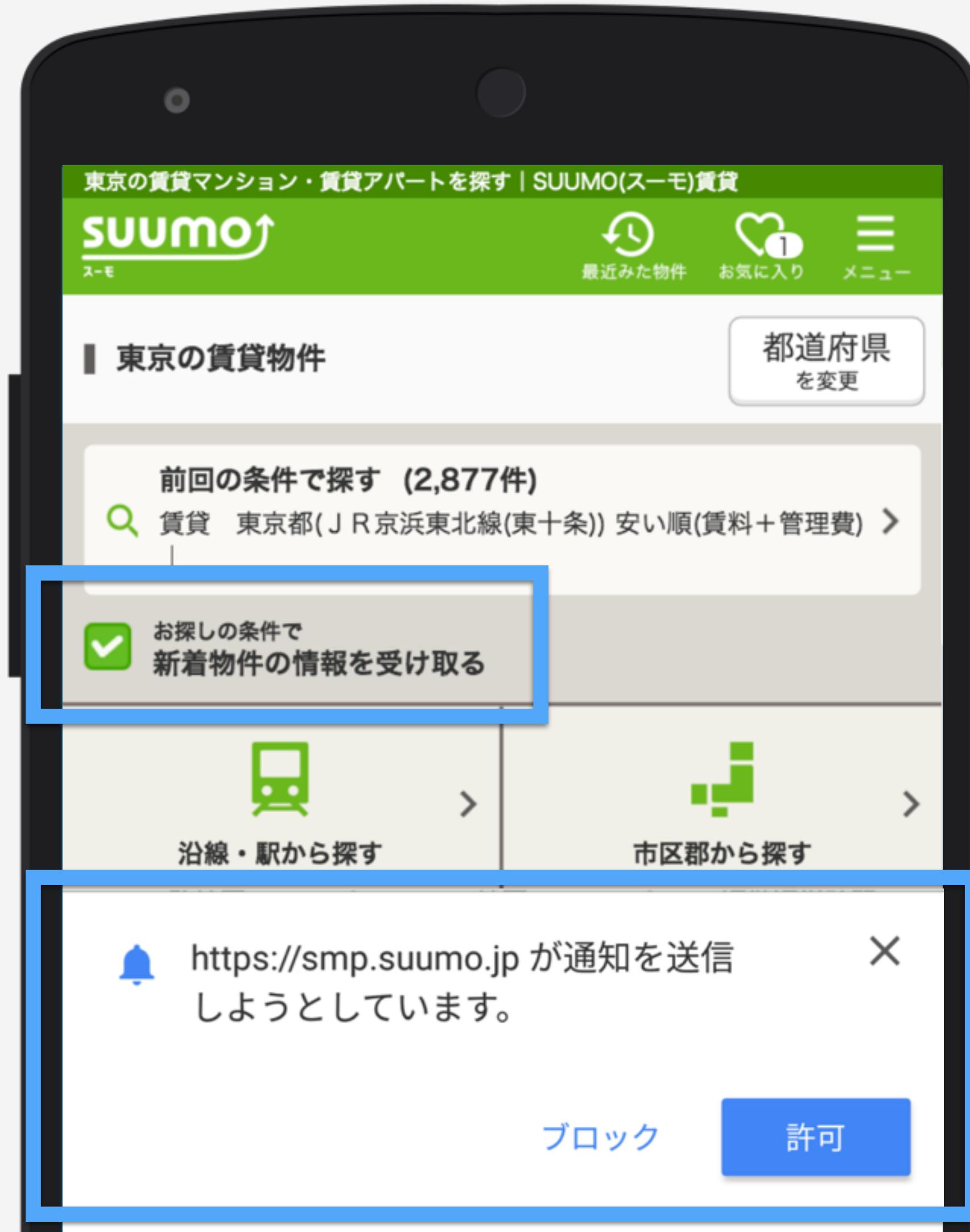


SUUMOは日々日々、物件情報が更新されるため、
情報の鮮度が高いうちにカスタマーにお届け することが、
カスタマーエクスペリエンス向上へつながる。

また、今までメールでしかできていなかった
Webカスタマーへのリテンションが、可能になった。

情報の鮮度が高いうちに、その情報を必要なカスタマーに対して
ピンポイントにお届けするため、Push Notification を実施。

Push Notification - 施策内容

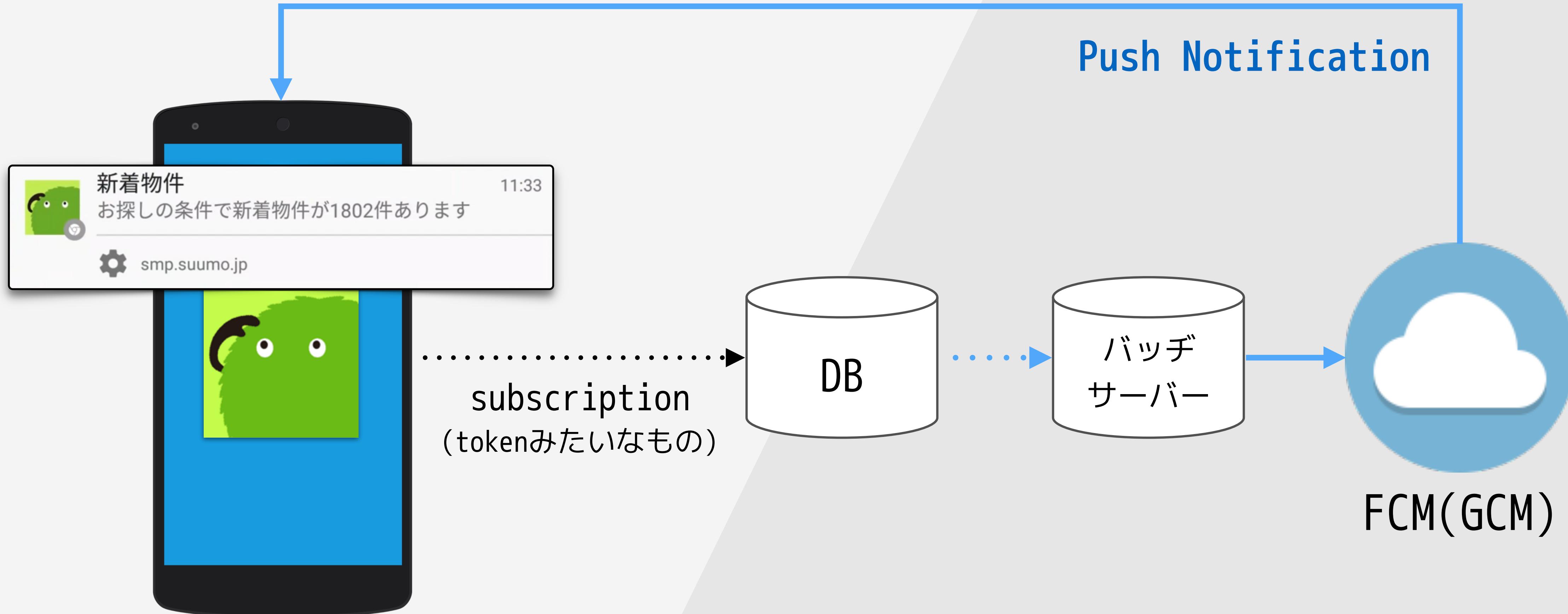


カスタマーが直近で探している条件で、
新着物件があった際に、その情報を通知。
カスタマーはその物件の一覧にアクセスできる。



新着物件
の一覧へ

Push Notification - システムとデータフロー



Push Notification の実装方法



① manifest に FCM(GCM) の プロジェクトNo を記載

※ Google Developer Console で作成。 ※ Chrome52以降で不要になりました

"gcm_sender_id": "012345678901234"

② subscribe() をし、endpoint 情報の取得処理

Service Worker 登録時のコールバック引数に渡される
registration オブジェクト内の、「pushManager」を利用

③ push を受け取った際 / 開く際 のイベントを定義

```
self.addEventListener('push', function(e){});  
self.addEventListener('notificationclick', function(e){});
```

④ push通知を送信する

① manifest ファイルに、FCM(GCM) の プロジェクトID を記載 (*Chrome 51まで)

■ manifest.json

```
{  
  "lang": "ja",  
  "name": "SUUMO",  
  "short_name": "SUUMO",  
  "icons": [{  
    "src": "/icon-144x144.png",  
    "sizes": "144x144",  
    "type": "image/png"  
  }],  
  :  
  :  
  :  
  :  
  "display": "standalone",  
  "gcm_sender_id": "012345678901234"  
}
```

② subscribe() をし、 endpoint 情報の取得処理

■ main.js

```
navigator.serviceWorker.ready.then(function(registration){  
    registration.pushManager.getSubscription().then(function(subscription){  
        if(subscription === null){  
  
            registration.pushManager.subscribe({userVisibleOnly: true})  
                .then(function(subscription){  
                    console.log(subscription.endpoint);  
                    // subscription.endpoint の値をサーバーへPOSTなどし、保存しておく  
                    // PUSHを送信する際に利用します  
                });  
  
        }  
    }).catch(function(e){});  
});
```

既にあるsubscription
を取得する処理

③ push を受け取った際 / 開く際 のイベントを定義



```
■ sw.js
```

PUSHを受け取った際のイベント

```
self.addEventListener("push", function(event){  
    self.registration.showNotification("通知タイトル", {  
        body: "メッセージ本文",  
        icon: "/icon.png",  
        tag: "new-bukken",  
        data: { url: "http://smp.suumo.jp/hogehoge" }  
    });  
});
```

通知をタップした際のイベント

```
self.addEventListener("notificationclick", function(event) {  
    event.notification.close();  
    clients.openWindow(event.notification.data.url);  
}, false);
```

③ push を受け取った際 / 開く際 のイベントを定義



■ sw.js

```
self.addEventListener("push", function(event){  
    self.registration.showNotification("通知タイトル", {
```

```
        body: "メッセージ本文",  
        icon: "/icon.png",  
        tag: "new-bukken",
```

```
        data: { url: "http://smp.suumo.jp/hogehoge" }
```

```
    });  
});
```

event.notification.dataで参照可能

```
self.addEventListener("notificationclick", function(event) {  
    event.notification.close();  
    clients.openWindow(event.notification.data.url)  
}, false);
```

④ push通知を送信する(curl)



```
curl --header "Authorization: key=<YOUR_API_KEY>" --header "Content-Type: application/json"  
https://android.googleapis.com/gcm/send -d "{\"registration_ids\":[\"<YOUR_REGISTRATION_ID>\"]}"
```

subscribe() した際の、subscription.endpoint の文字列から、
“https://android.googleapis.com/gcm/send/” を除いたもの

※ subscription.endpoint は、以下のような値になっています（GCMのときまで）
「https://android.googleapis.com/gcm/send/xxxxxxxxxxxxxxx」

※ 動的にPUSH通知の文章やURLを変える

```
var URL = "http://smp.suumo.jp/";

self.addEventListener("push", function(event){
  event.waitUntil(
    fetch(URL, {}).then(function(response){
      response.json().then(function(data){
        self.registration.showNotification( data.title, {
          body: data.body,
          icon: data.icon,
          tag: data.tag,
          data: { url: data.url }
        });
      });
    });
  );
});
```

pushイベント内で、fetchをし、
指定したURLから情報を取得

取得したデータで、
PUSH通知の情報をつくる

※ パラメータ付きのPUSHを送る (Payload)

参考

Chrome version 50 以降から、Payload付きのPUSH通知に対応。
(※メッセージの暗号化が必要)

<https://developers.google.com/web/updates/2016/03/web-push-encryption>

**これにより、サーバー側からクライアント側へ動的な
パラメータを送れるようになっています。**



Offline Cache (+ App Shell)



Offline Cache – 実施の背景



SUUMOはリアルタイムで「物件情報」を扱うサービス。
そのため、サービス性質上 **オンライン環境での利用は必須**

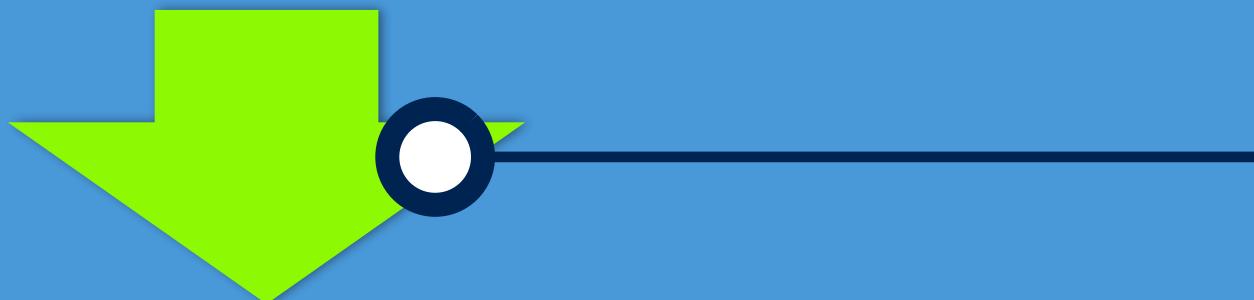
「画面表示の高速化」を目的として、 Offline Cache を実験的に導入。

(例) Offline Cache を利用した高速化



0.8475 ms

キャッシュ 非利用時

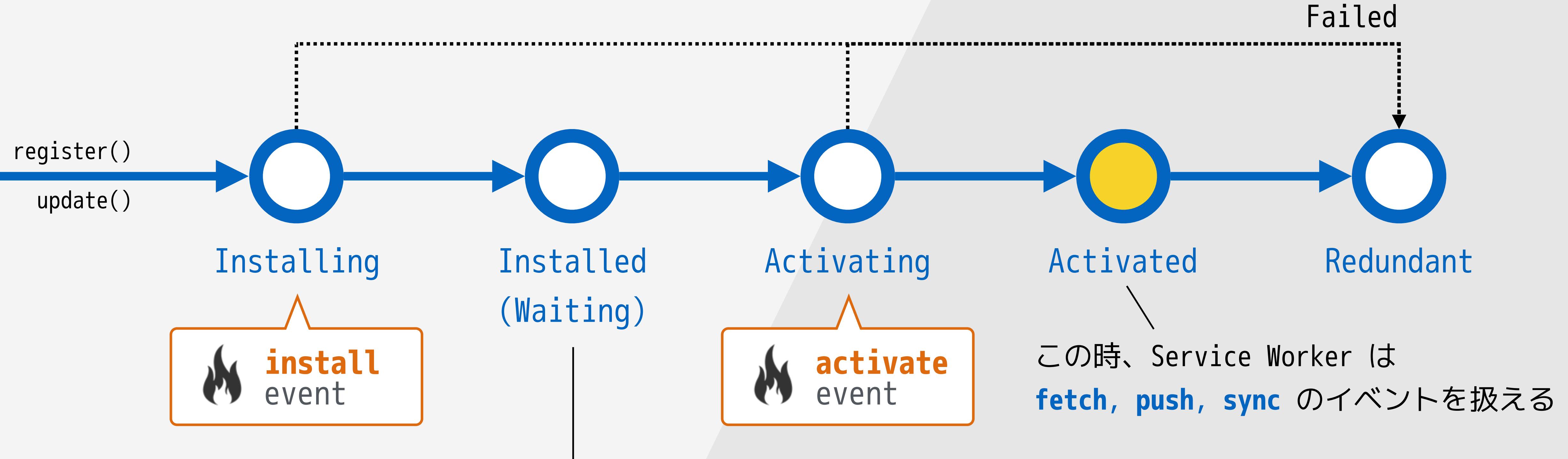


速度4倍！

0.2019 ms

オフラインキャッシュ 利用時

Service Worker Lifecycle



Service Workerがコントロールしているページが
すべて閉じられ、タスクが終了するまでは、この状態となる。
※ `skipWaiting()` でこの状態を飛ばすことも可能

Cache の追加の仕方

The screenshot shows the Chrome Developer Tools Application tab for the URL `http://localhost/sw/sw.js`. The left sidebar lists 'Manifest', 'Service Workers', and 'Clear storage'. Under 'Storage', it shows 'Local Storage', 'Session Storage', 'IndexedDB', 'Web SQL', and 'Cookies'. In the main area, there's a table of requests:

#	Request	Response
0	<code>http://localhost/sw/sample.png</code>	OK
1	<code>http://localhost/sw/test.html</code>	OK

Below the table, two annotations are present:

- An arrow points from the text "key:Request" to the Request column header in the table.
- An arrow points from the text "value:Response" to the Response column header in the table.

Text annotations in the center area state:

**key:Request
value:Response
のセットでcacheに保存される**

**指定した名前のキャッシュが無い
場合は、自動的に追加されます**

```
var urlList = [
  "/sw/test.html",
  "/sw/sample.png"
];

// service worker の
// install イベント時に追加する例
self.addEventListener("install",
  function(event) {
    event.waitUntil(
      caches.open("my-site-cache-v1").then(
        function(cache) {
          return cache.addAll(urlList);
        }
      )
    );
  }
);
```

Cache の追加の仕方

The screenshot shows the Chrome Developer Tools interface with the 'Application' tab selected. On the left, there's a sidebar with sections for 'Manifest', 'Service Workers', and 'Clear storage'. Below that is 'Storage' with options for 'Local Storage', 'Session Storage', 'IndexedDB', 'Web SQL', and 'Cookies'. Under 'Cache', there's a dropdown menu for 'Cache Storage' which is currently expanded to show 'my-site-cache-v1 - http:' and 'Application Cache'. The main area has a table titled 'Request' with two rows:

#	Request	Response
0	http://localhost/sw/sample.png	OK
1	http://localhost/sw/test.html	OK

// 追加の方法は他にもあります

- ◆ `cache.addAll(urlList);`
- ◆ `cache.add(url);`
- ◆ `cache.put(request, response);`

// ちなみに、削除はこんな感じ

- ◆ `cache.delete(url);`

// そして、参照の仕方はこんな感じ

- ◆ `cache.match(url).then(…);`
- ◆ `cache.match(request).then(…);`
- ◆ `cache.matchAll(url).then(…);`

Cache を利用して Response を返す

■ sw.js

```
// fetch のイベント時 (リクエスト発生時)
self.addEventListener('fetch', function(event) {
  event.respondWith(
```

```
caches.match(event.request).then(function(response) {
  return response || fetch(event.request);
})
```

```
);
});
```

例) cache があれば cache を Response として返し、
なければネットワークリクエストを行う。

参考

「 The offline cookbook 」

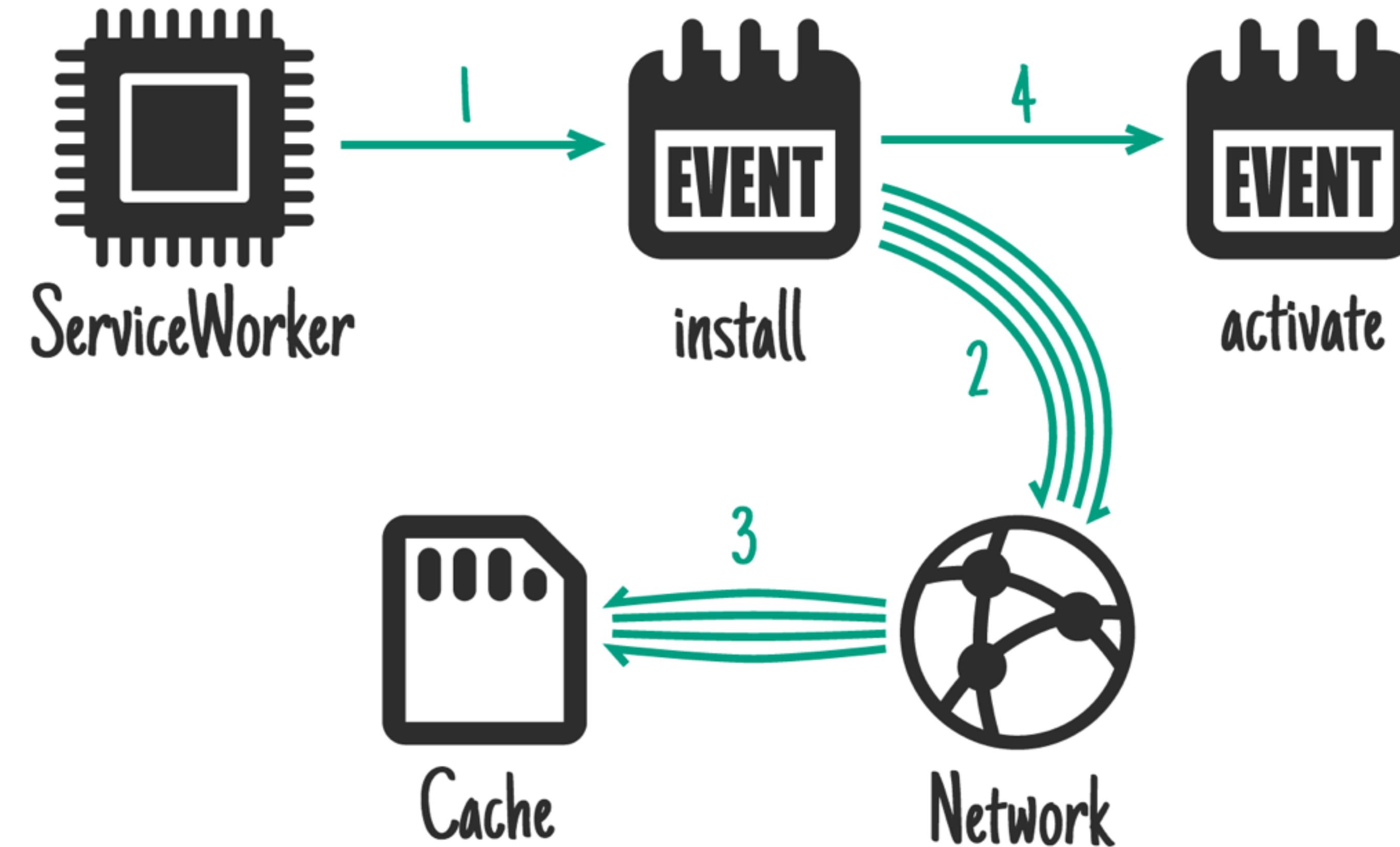
原文 (@Jake Archibald) : <https://jakearchibald.com/2014/offline-cookbook/>

日本語訳 : <https://github.com/kuu/the-offline-cookbook-ja/blob/master/ja.md>

キャッシュ更新と、リクエスト処理に関するパターンが紹介されています。

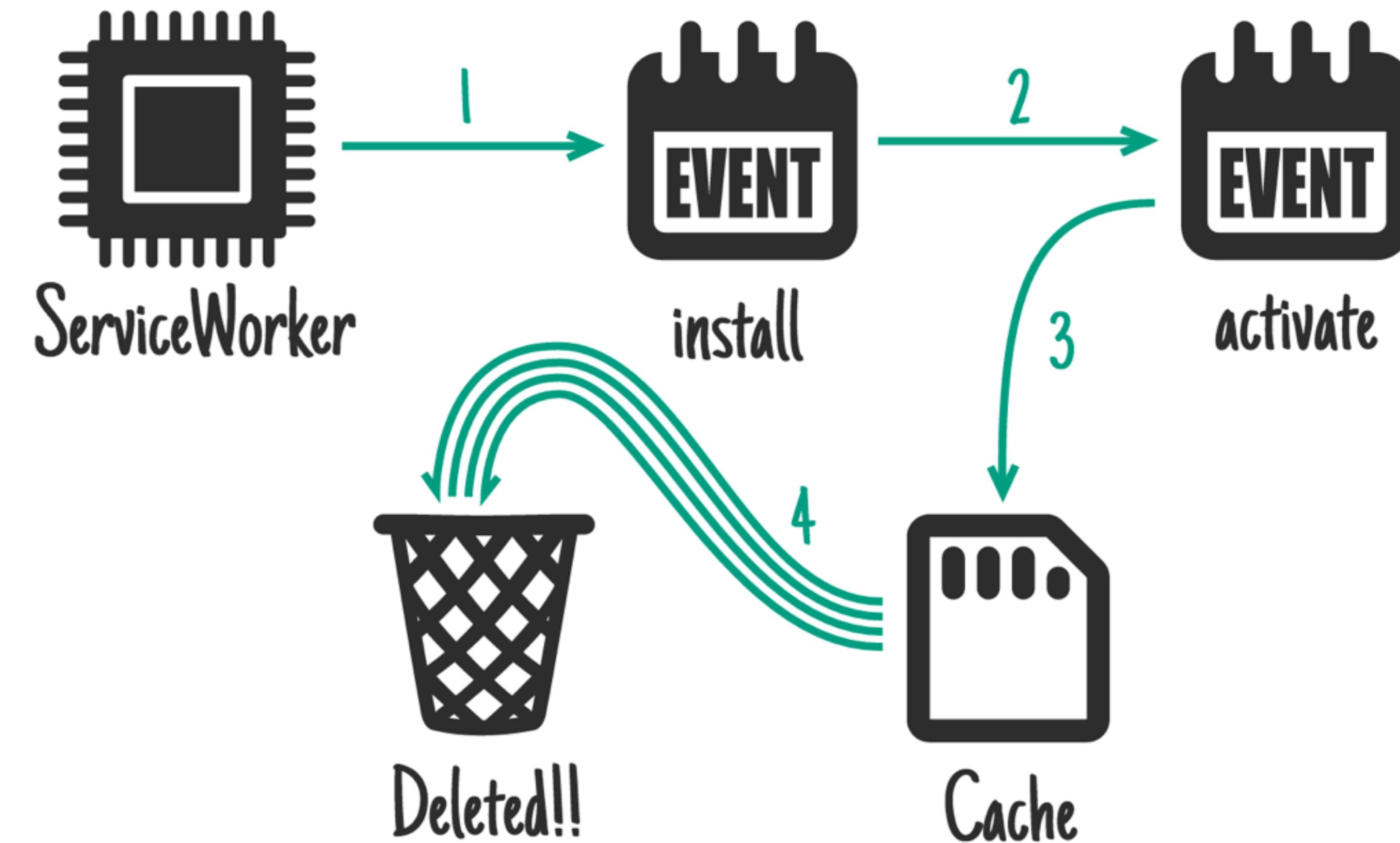
「 The offline cookbook 」

(例) install イベント時にリソースをキャッシュに保存する



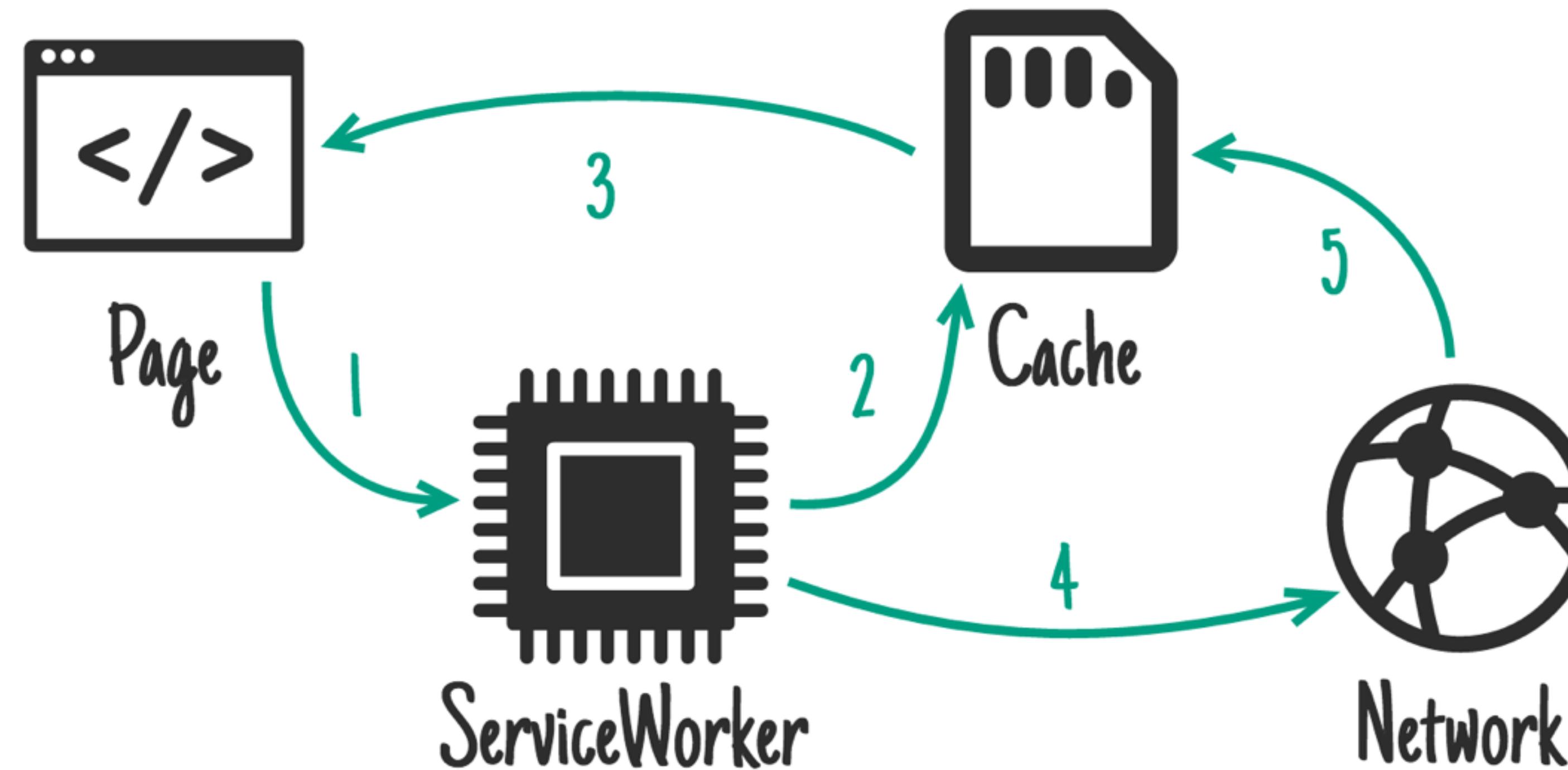
「 The offline cookbook 」

(例) activate イベント時に不要なリソースをキャッシュから削除する



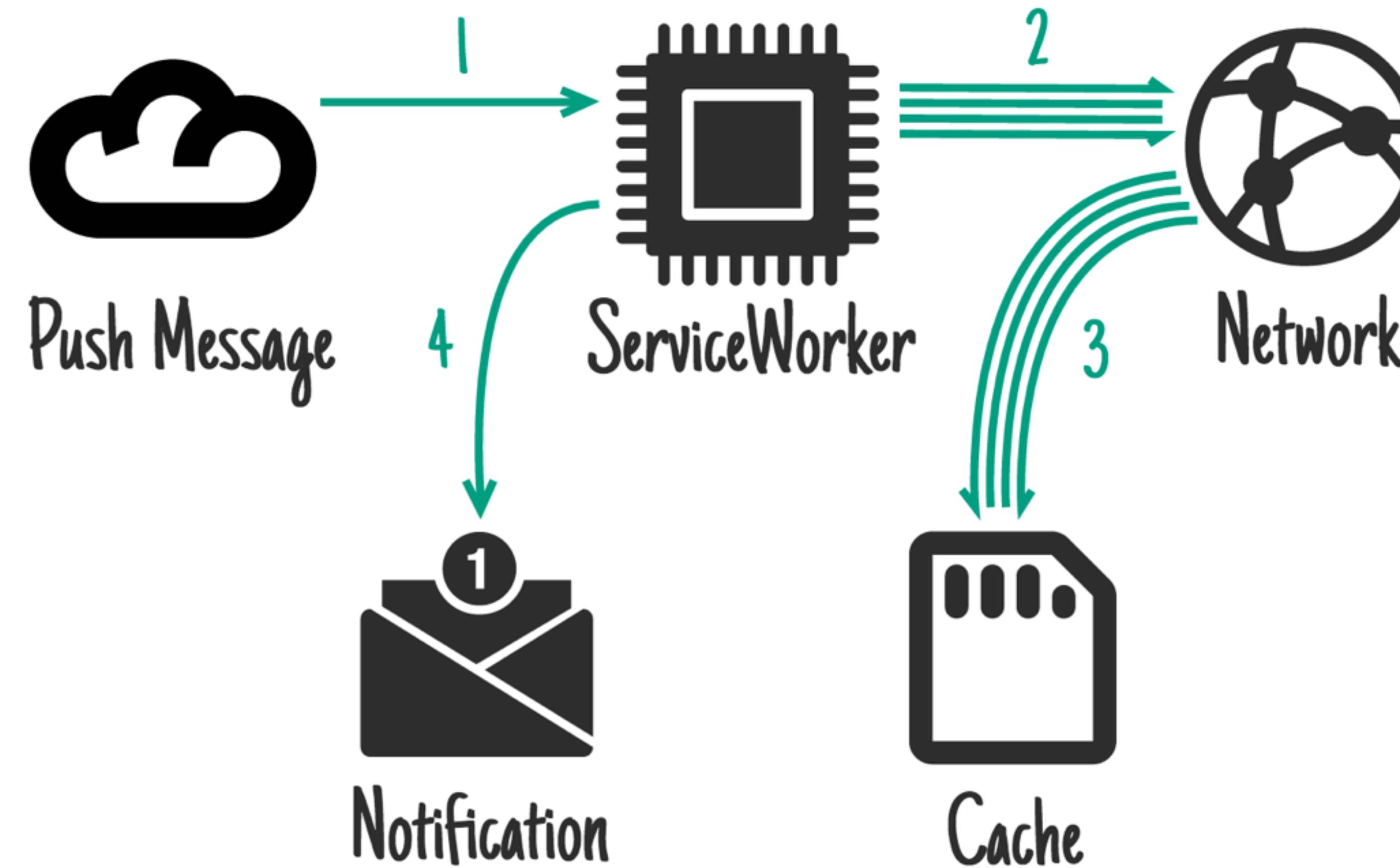
「 The offline cookbook 」

(例) キャッシュを使用すると同時に、次に備えキャッシュを更新させる



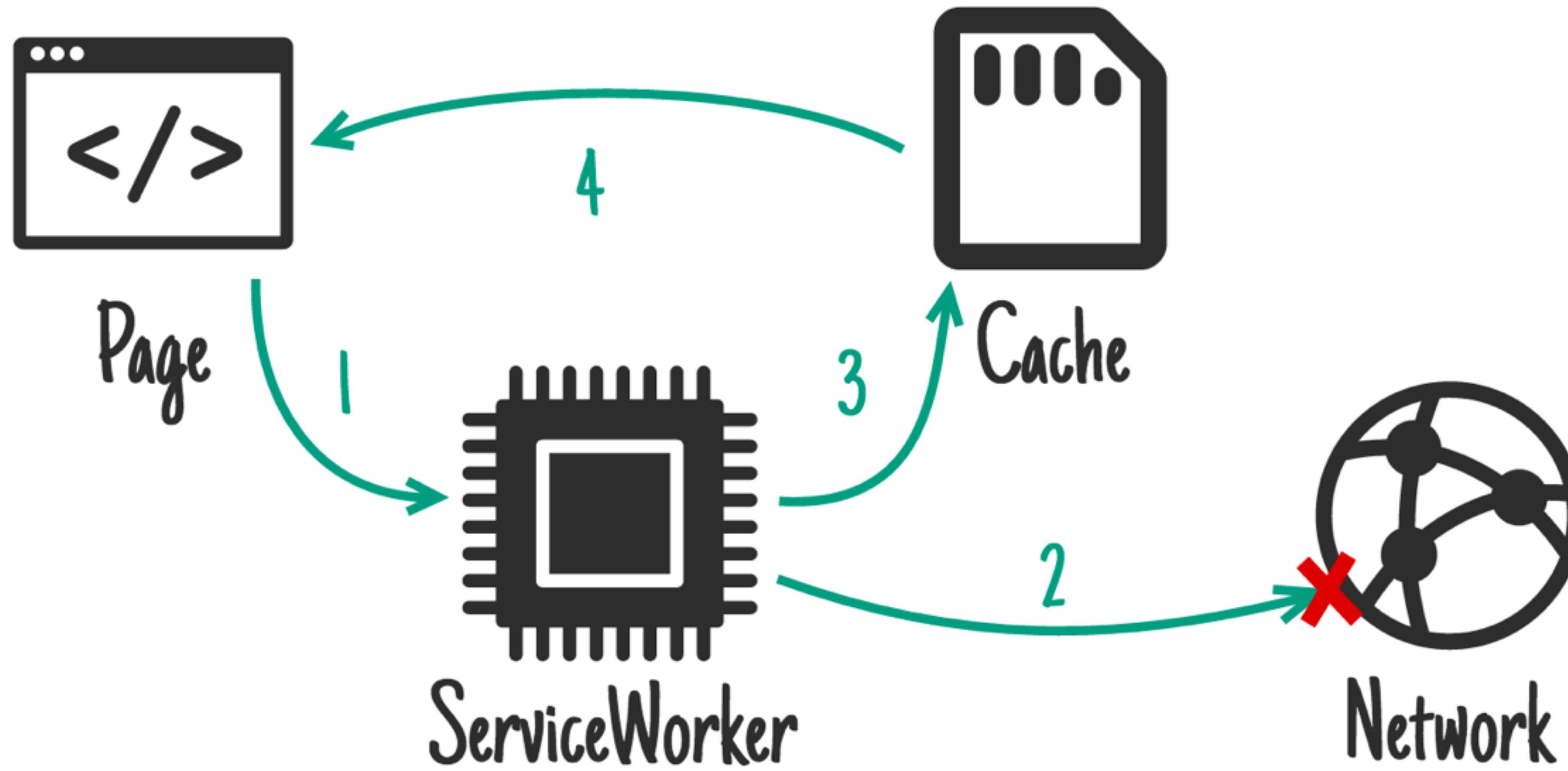
「 The offline cookbook 」

(例) PUSH通知を開く前に予めキャッシュさせておく



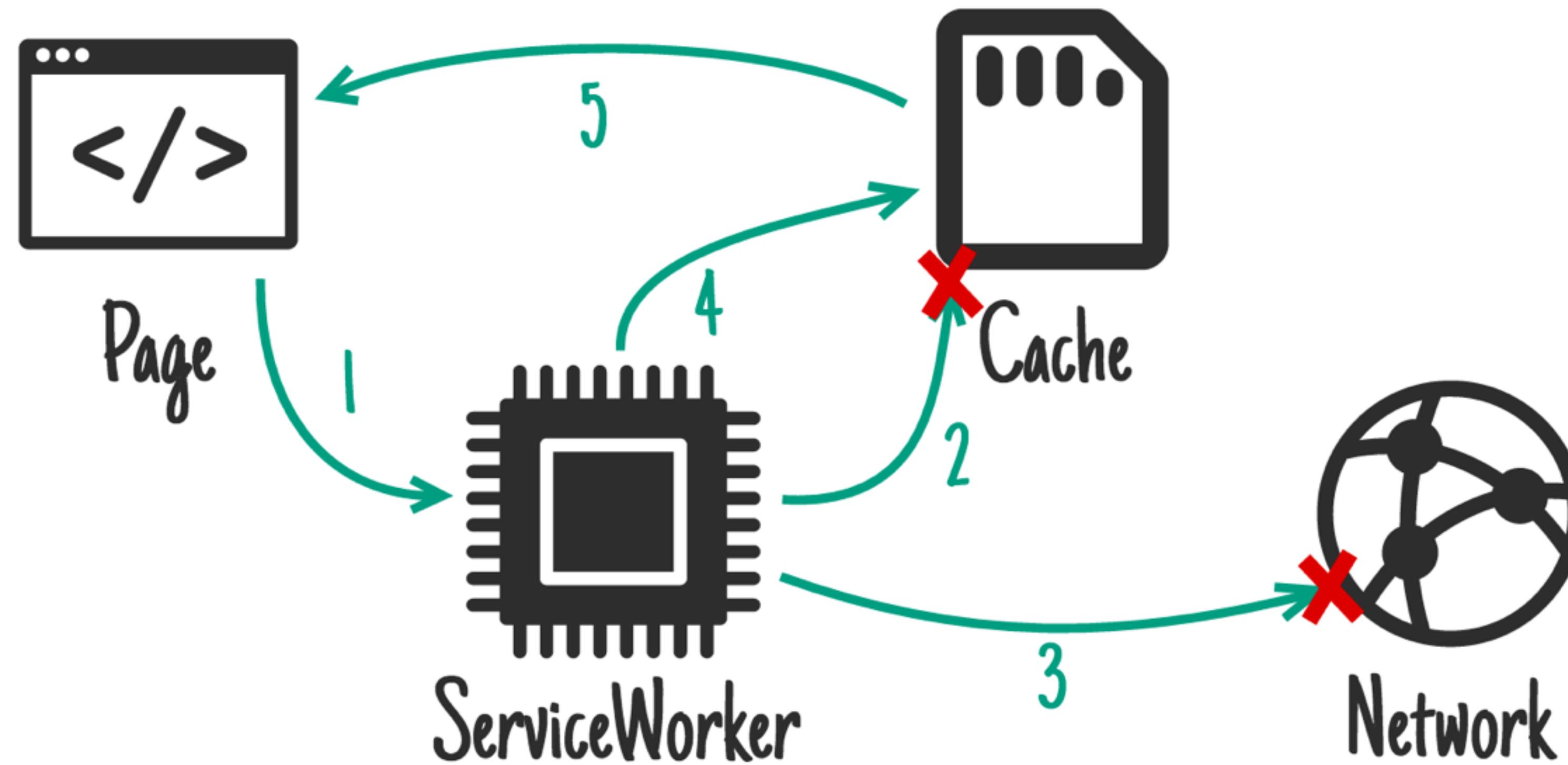
「 The offline cookbook 」

(例) ネットワークリクエストに失敗した時にキャッシュを利用する



「 The offline cookbook 」

(例) キャッシュ/ネットワークどちらも取得できなければデフォルトを利用



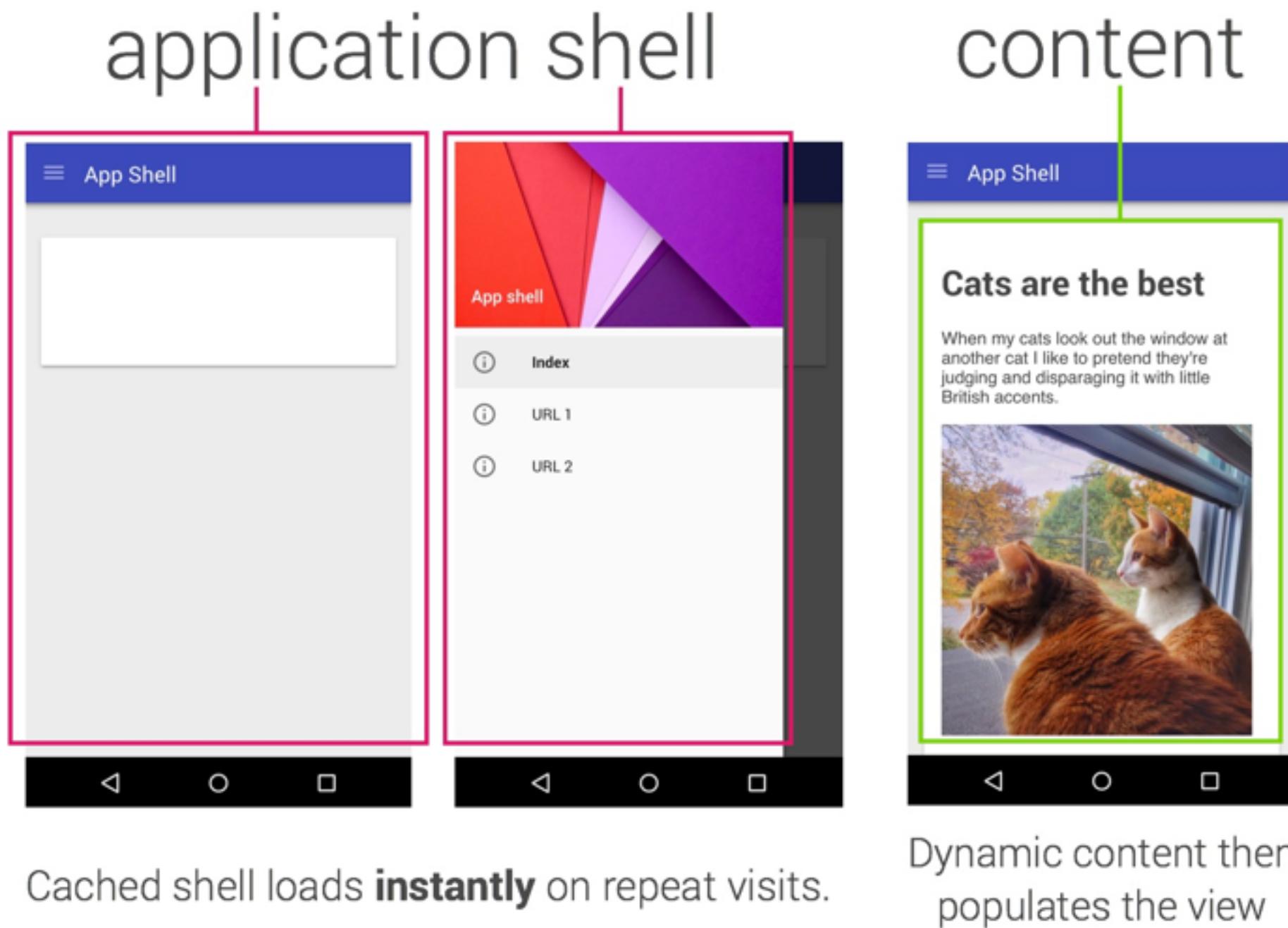
“App Shell” アーキテクチャ

App Shell アーキテクチャの概念について

“

App Shell のアーキテクチャでは、アプリケーションの核となるインフラストラクチャとユーザーインターフェースを、データから切り離して扱います。ユーザーインターフェースとインフラストラクチャはすべて、Service Worker によりローカルにキャッシュされる。

”



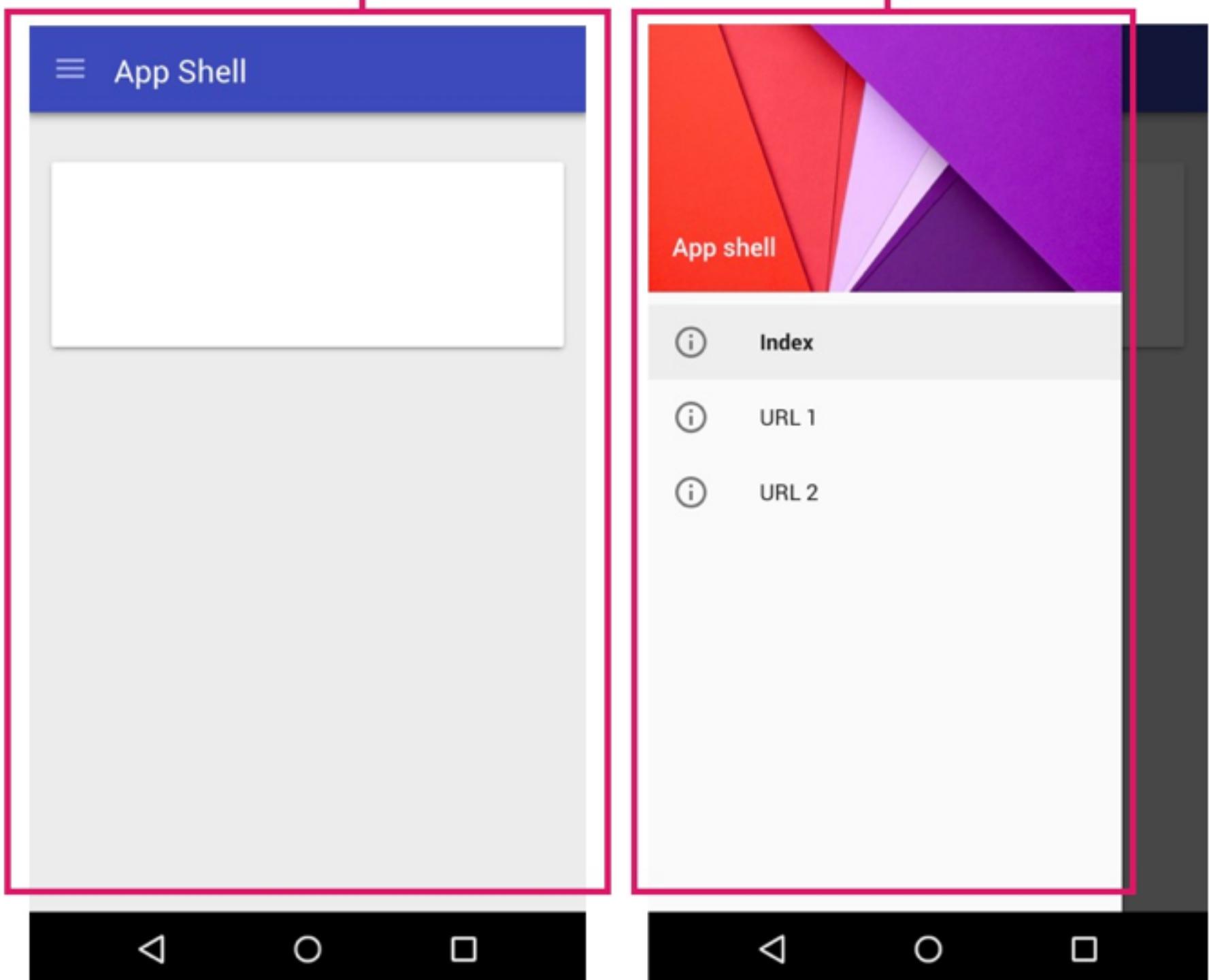
参照) 「App Shellを構築する」

<https://developers.google.com/web/fundamentals/>

テンプレートは
Service Workerでキャッシュ

データのみを必要な分だけ
取得して画面をつくる

application shell



Cached shell loads **instantly** on repeat visits.

UI と データを分離

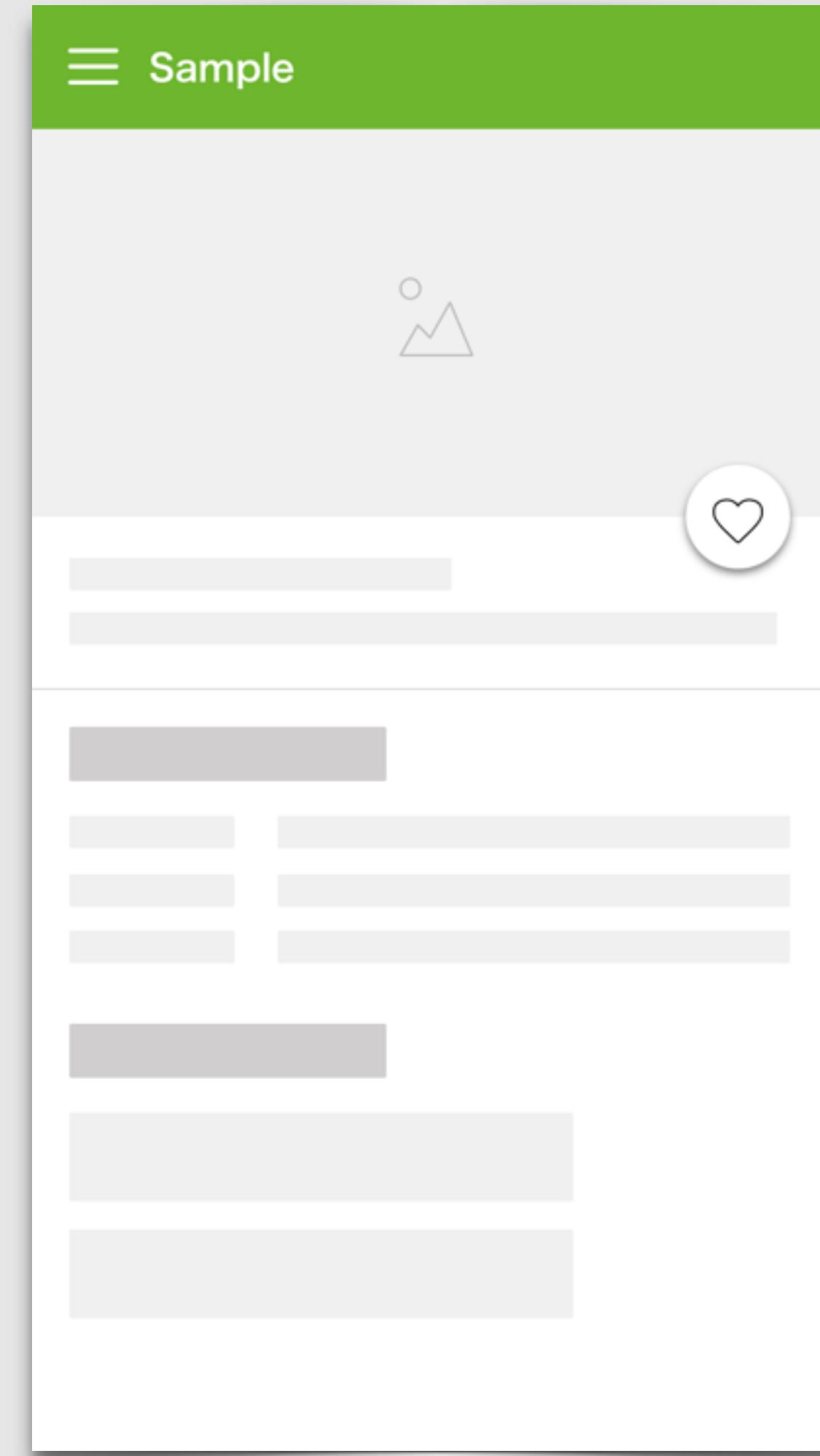
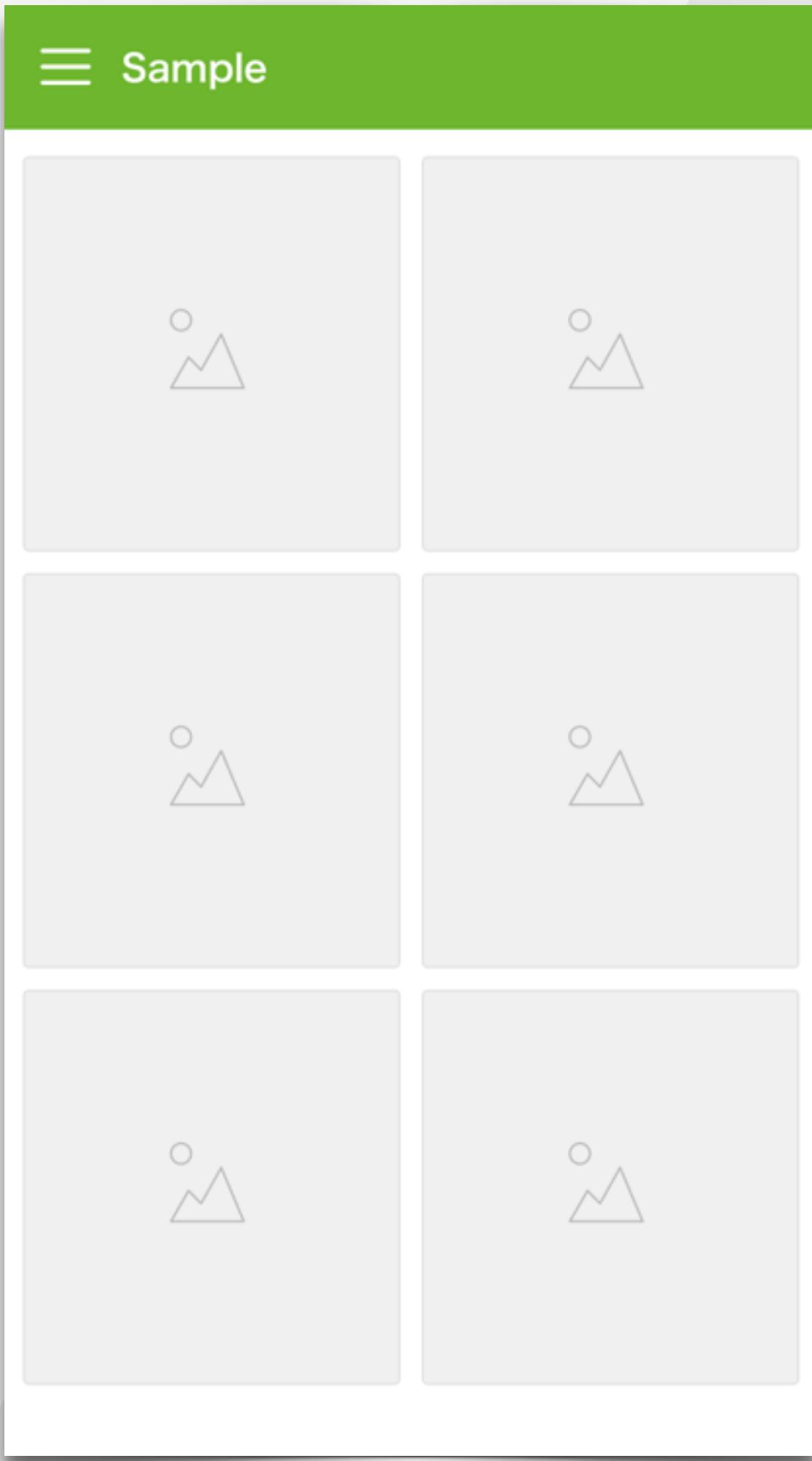
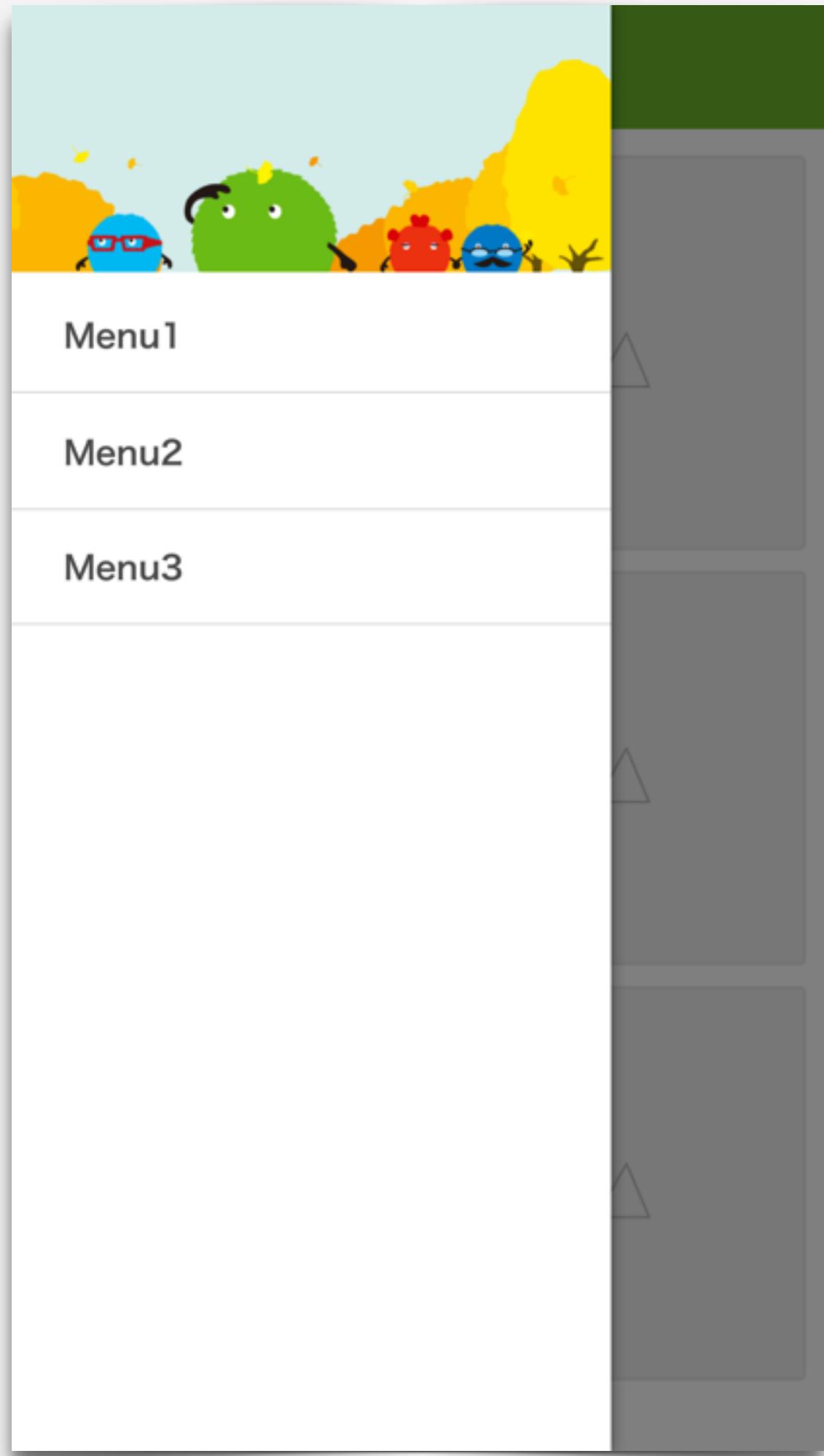
content



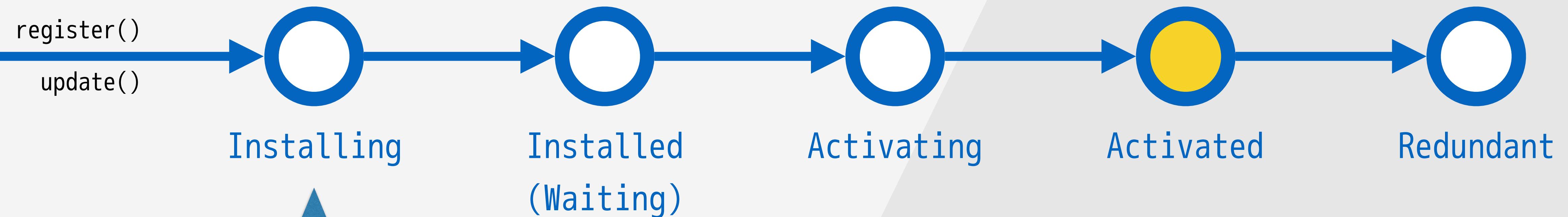
Dynamic content then
populates the view

App Shell の実装の流れ (例)

※ App shell Architecture

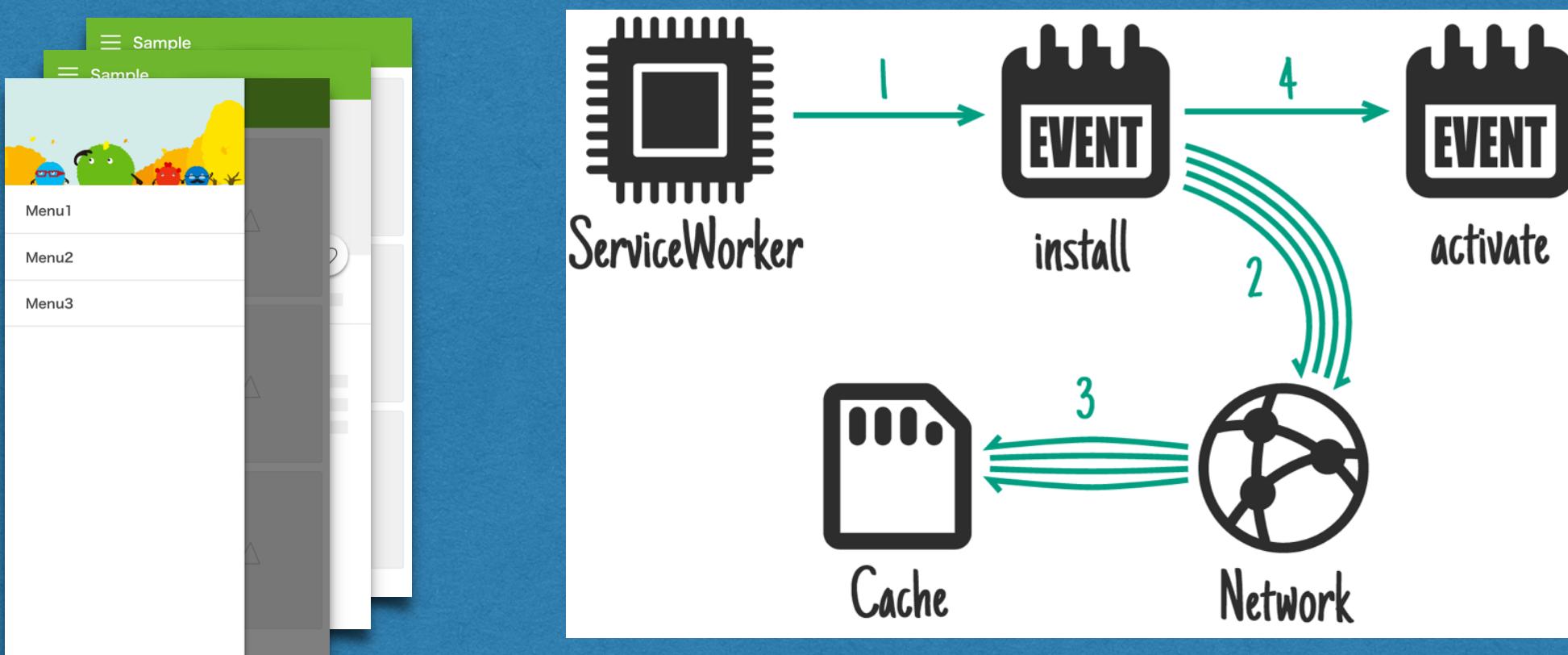


App Shell の実装の流れ (例)

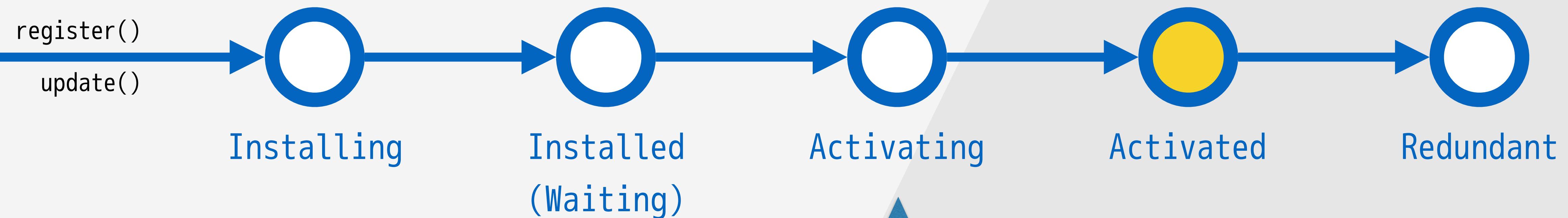


Templates をキャッシュ化

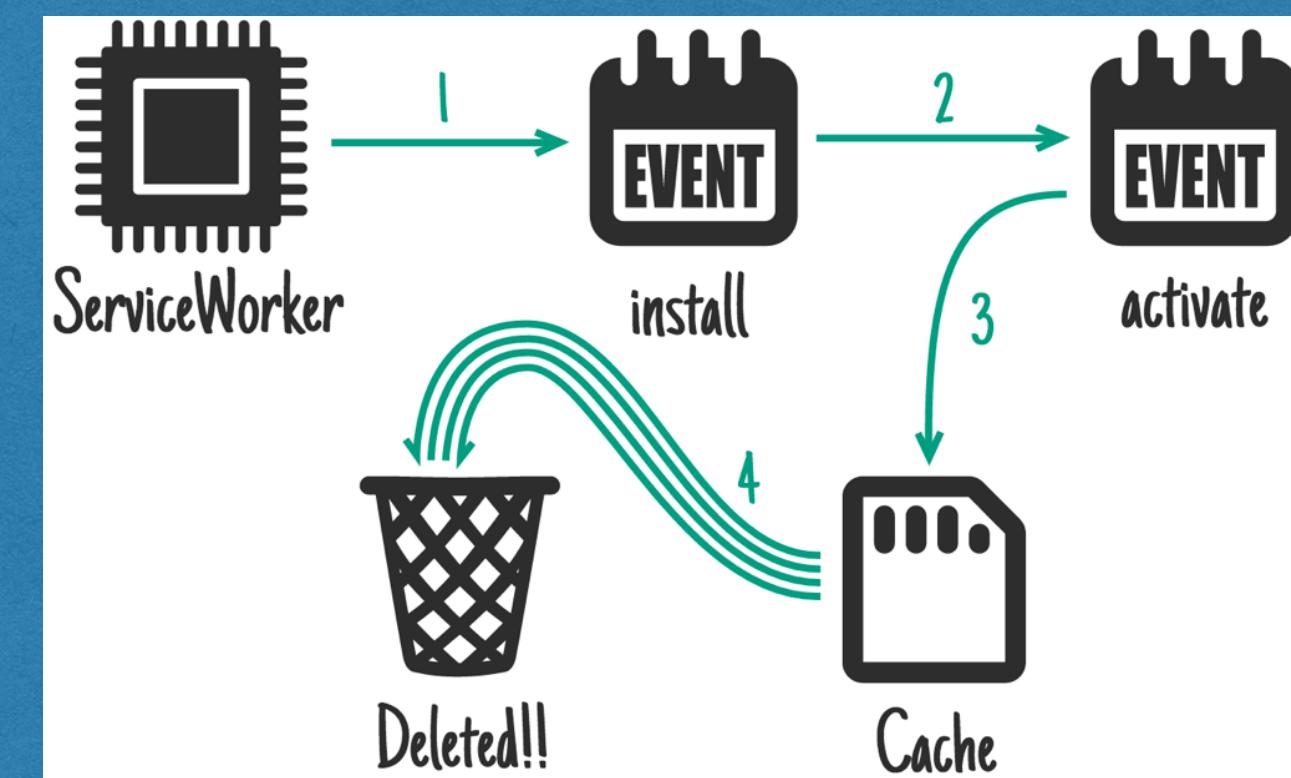
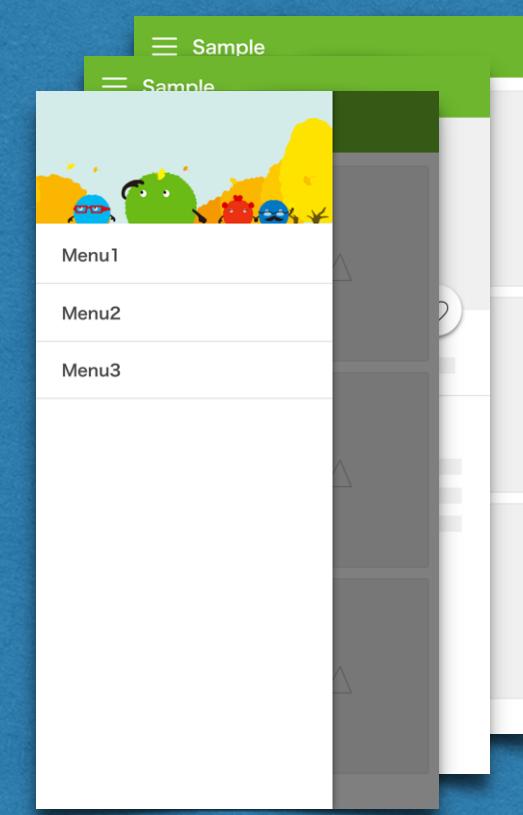
html, css, js
image, ...



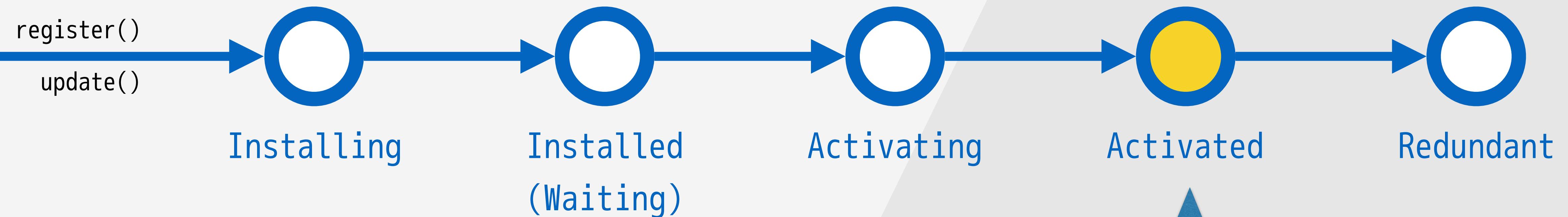
App Shell の実装の流れ (例)



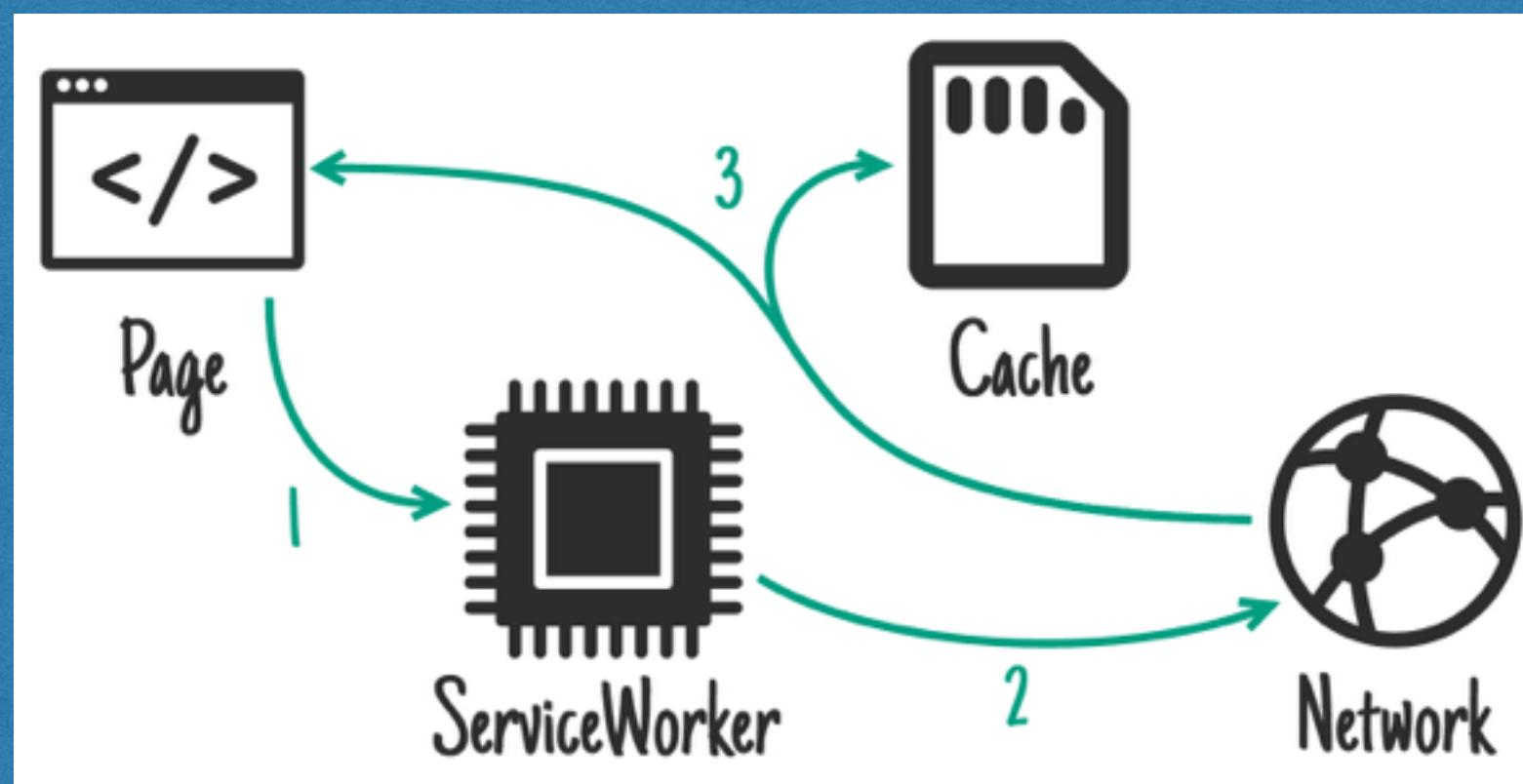
UI更新などがあって、古くなった Templates の
キャッシュがあれば消すなど。(migrate)



App Shell の実装の流れ (例)



fetch イベント の際に Templates は キャッシュから。
データは常にネットワークから取得

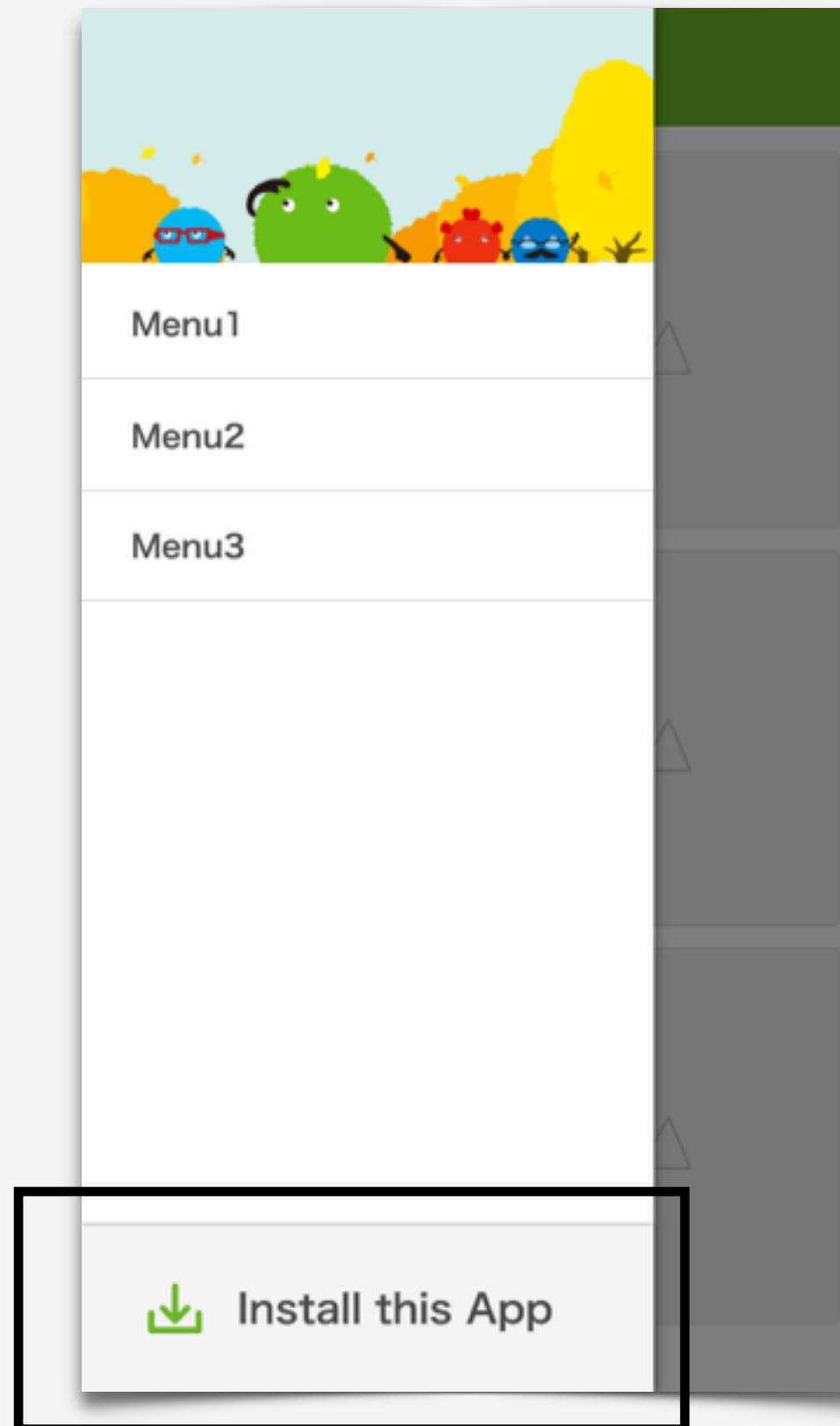


※ データをキャッシュさせておき、
オフライン時のみ、それを使って
コンテンツ閲覧を可能にすることも
できる

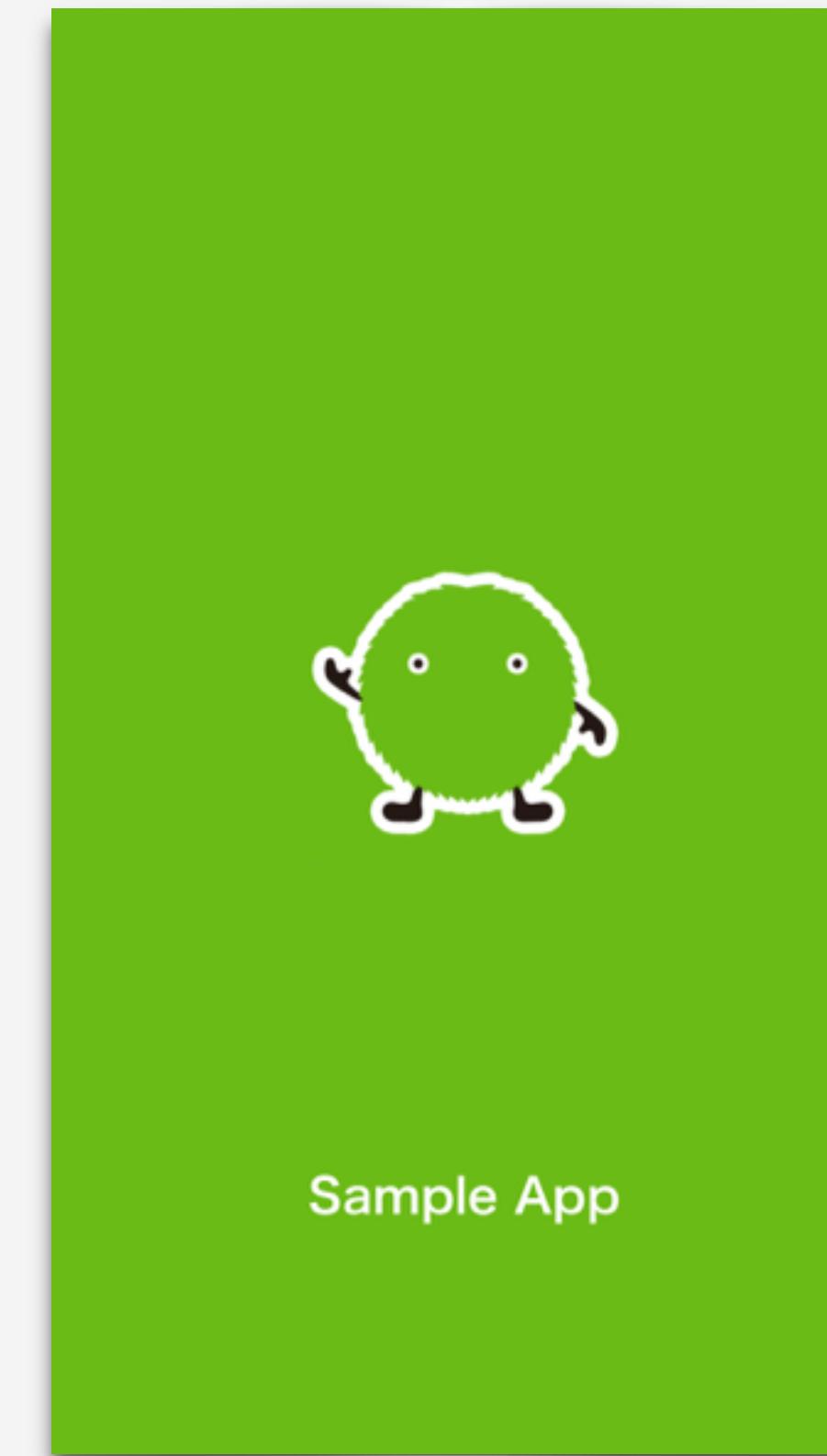
App Shell の実装の流れ (例)

Add to Home screen / PUSH と組み合わせると、さらにネイティブアプリに近づけられる。

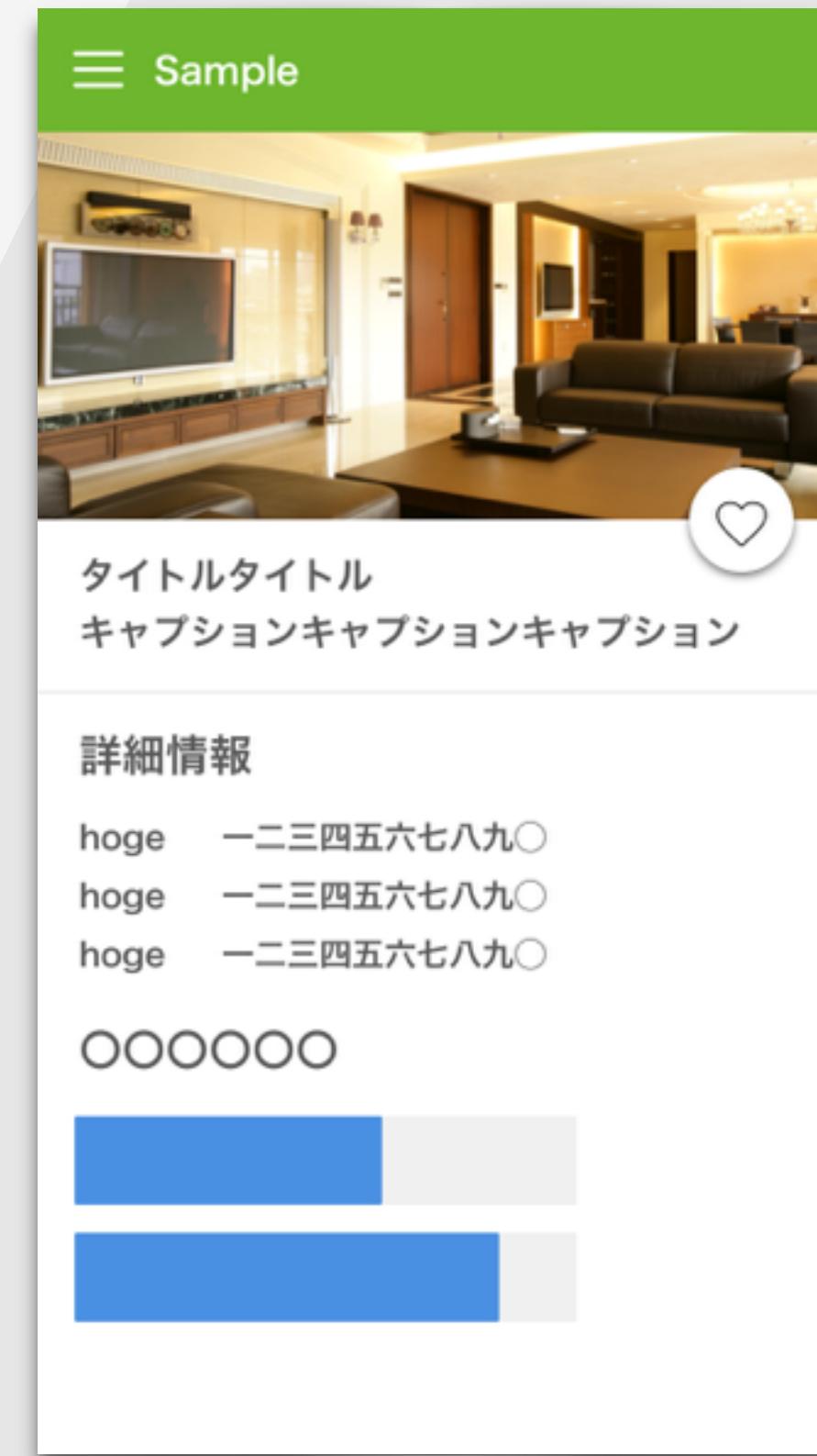
add to home screen で
web アプリとして
インストール



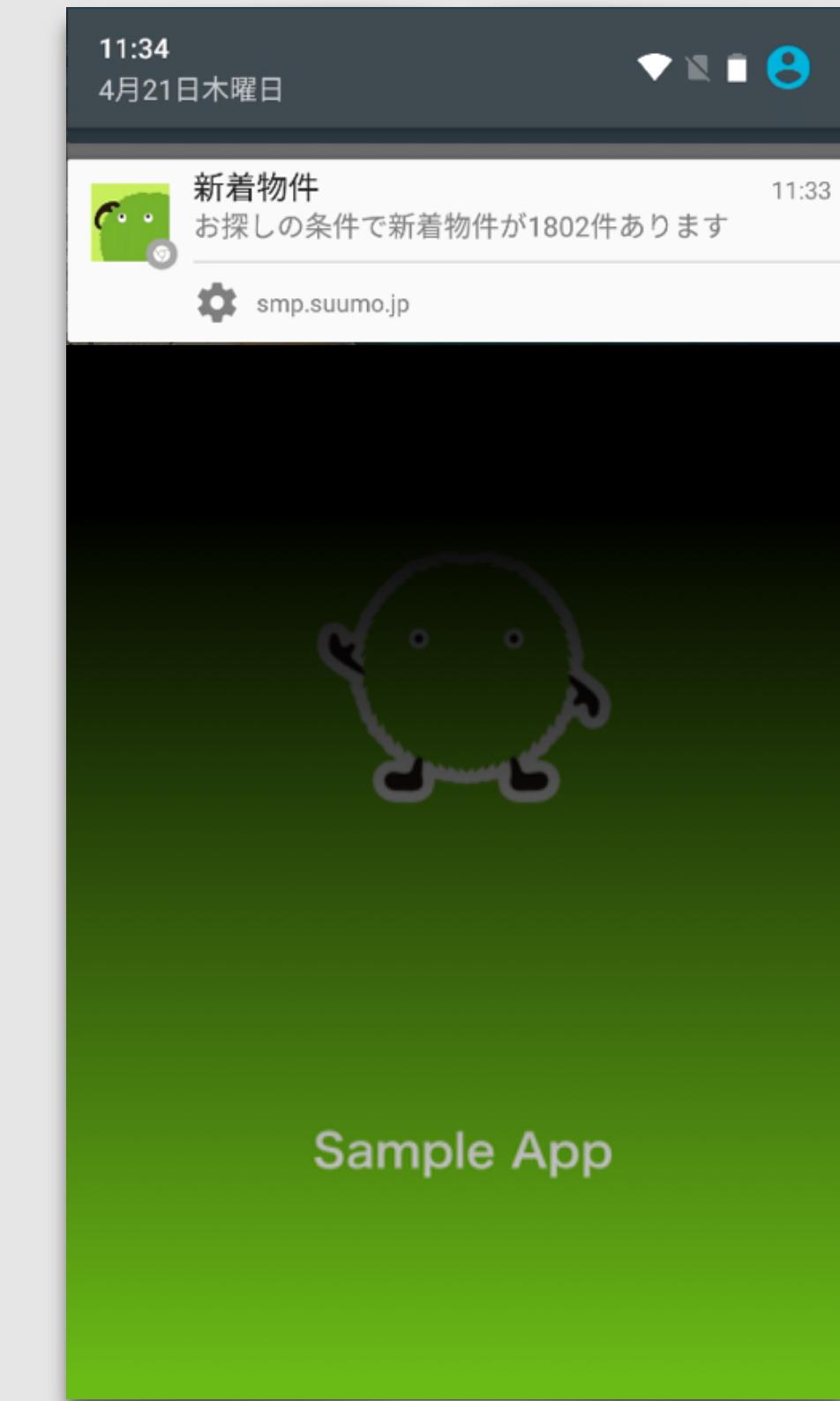
Home Screen からは
アプリのようにSplash画面が
立ち上がる



App Shell による
ネイティブアプリのような
パフォーマンス



PUSH通知による
リテンションも



Progressive Web Apps



Add to
Home Screen



Push Notifications



Offline Cache
(+ App Shell)



Thank you for your attention.