# CROSS-BROWSER EXTENSION DEVELOPMENT

## Managing complexity of cross-platform code

Sergey Yavnyi / @glatteis


May, 2018

# About me 👋

- User-facing products @ Grammarly
- Extensions tech lead, github.com/grammarly/focal
- Programming (F#, Python, TypeScript, Elm, etc.)
- 10+ yrs, 2+ on the web front-end

github.com/blacktaxi, reversemicrowave.me


- A writing app *(grammar, spelling and more!)*
- Web, desktop, mobile
- 10M+ active users

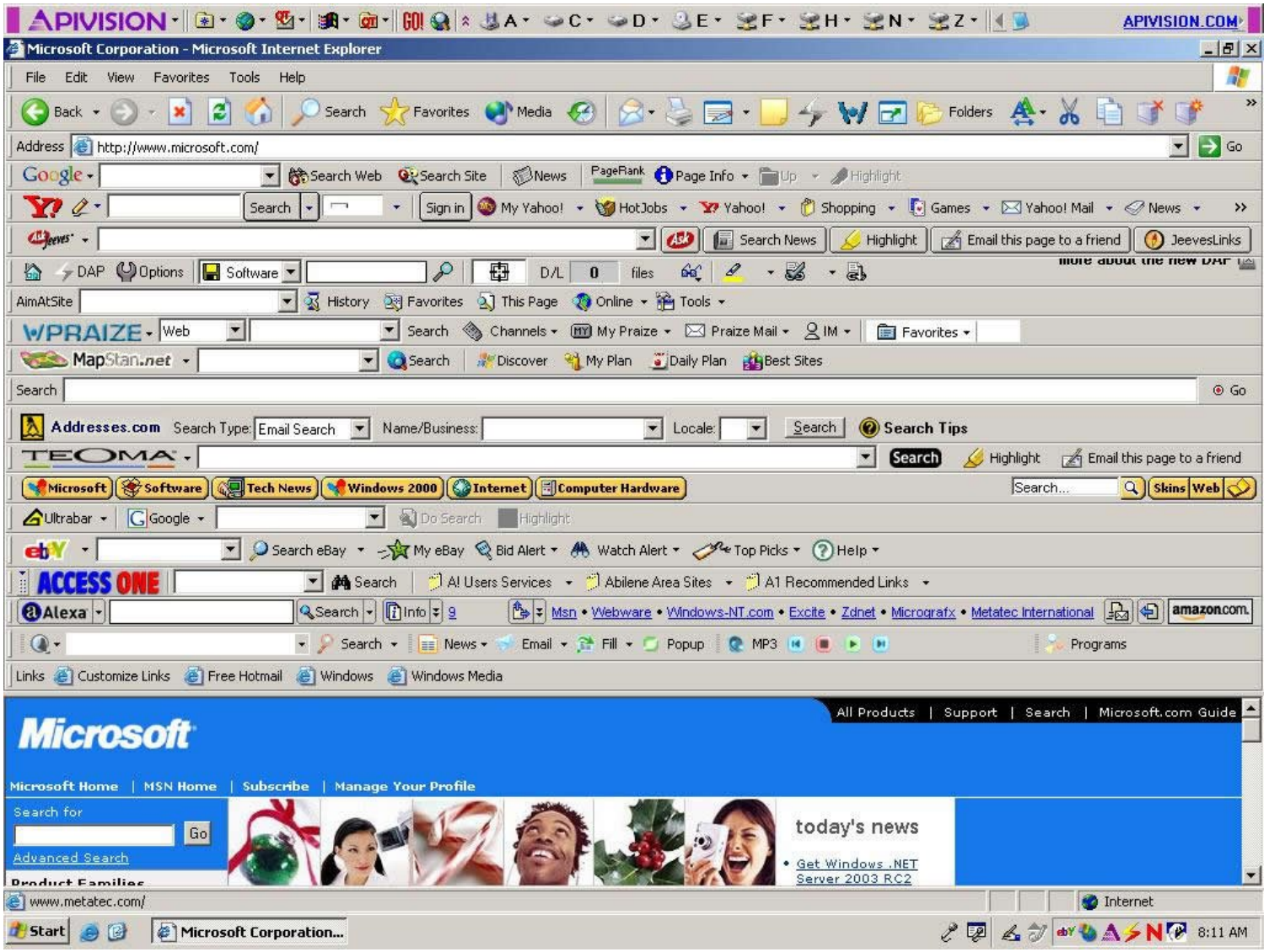
# THIS TALK: CROSS-BROWSERNESS

- Why
- Coding side of 'how'

# WHY EXTENSIONS?

Everywhere you go!

- Inherent value for the user
- Stickiness and retention for your product

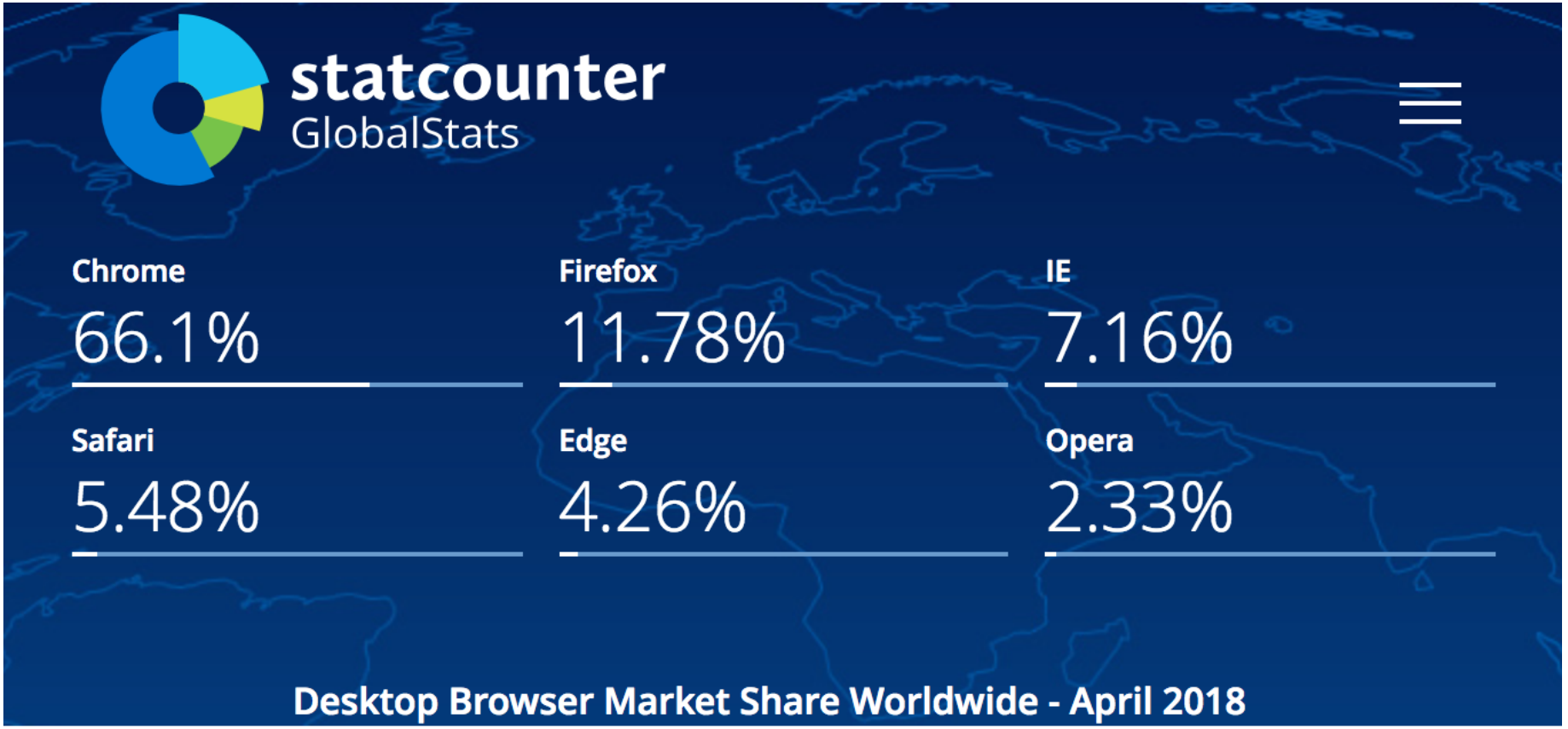

APIVISION
APIVISION.COM
Microsoft Corporation - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media Folders
Address http://www.microsoft.com/
Go
Google Search Web Search Site News PageRank Page Info Up Highlight
Search Sign in My Yahoo! HotJobs Yahoo! Shopping Games Yahoo! Mail News
Search News Highlight Email this page to a friend JeevesLinks
DAP Options Software D/L 0 files
AimAtSite History Favorites This Page Online Tools
WPRAIZE Web Search Channels My Praize Praize Mail IM Favorites
MapStan.net Search Discover My Plan Daily Plan Best Sites
Search Go
Addresses.com Search Type: Email Search Name/Business: Locale: Search Search Tips
TEOMA Search Highlight Email this page to a friend
Microsoft Software Tech News Windows 2000 Internet Computer Hardware
Search... Skins Web
Ultrabar Google Do Search Highlight
Search eBay My eBay Bid Alert Watch Alert Top Picks Help
ACCESS ONE Search A! Users Services Abilene Area Sites A1 Recommended Links
Alexa Search Info 9 Msn • Webware • Windows-NT.com • Excite • Zdnet • Micrografx • Metatec International amazon.com
Search News Email Fill Popup MP3 Programs
Links Customize Links Free Hotmail Windows Windows Media
Microsoft
All Products | Support | Search | Microsoft.com Guide
Microsoft Home | MSN Home | Subscribe | Manage Your Profile
Search for
Go
Advanced Search
today's news
Get Windows .NET Server 2003 RC2
www.metatec.com/
Internet
Start
Microsoft Corporation...
8:11 AM

# CROSS-BROWSER?

Not when:

- Platform-specific product
- Market fit uncertain
- Limited dev resource

# HOWEVER...


statcounter
GlobalStats
Chrome
66.1%
Firefox
11.78%
IE
7.16%
Safari
5.48%
Edge
4.26%
Opera
2.33%
Desktop Browser Market Share Worldwide - April 2018

# WHY CROSS-BROWSER?

- User reach
- Availability as value proposition (see 1Password, Evernote, Pocket)

# NO UNIFIED PLATFORM

JavaScript, HTML, CSS can work differently:

- API surface (caretPositionFromPoint vs caretRangeFromPoint)
- Behavior (`fetch` vs `XHR` wr/t `Origin`)
- Rendering (newer CSS features)

caniuse.com

# THE WEB: STANDARDIZED

- W3C: standards for HTML, CSS, JS
- TC39: standard for ECMAScript

# BROWSER EXTENSIONS

Standards? Not so much.

- Chrome: in development, not ready to standardize
- Firefox: WebExtensions API
- Edge: a mix of WebExtensions and Chrome-like APIs
- Safari: replacing exclusive API with... exclusive API

# EVEN ON THE SAME BROWSER

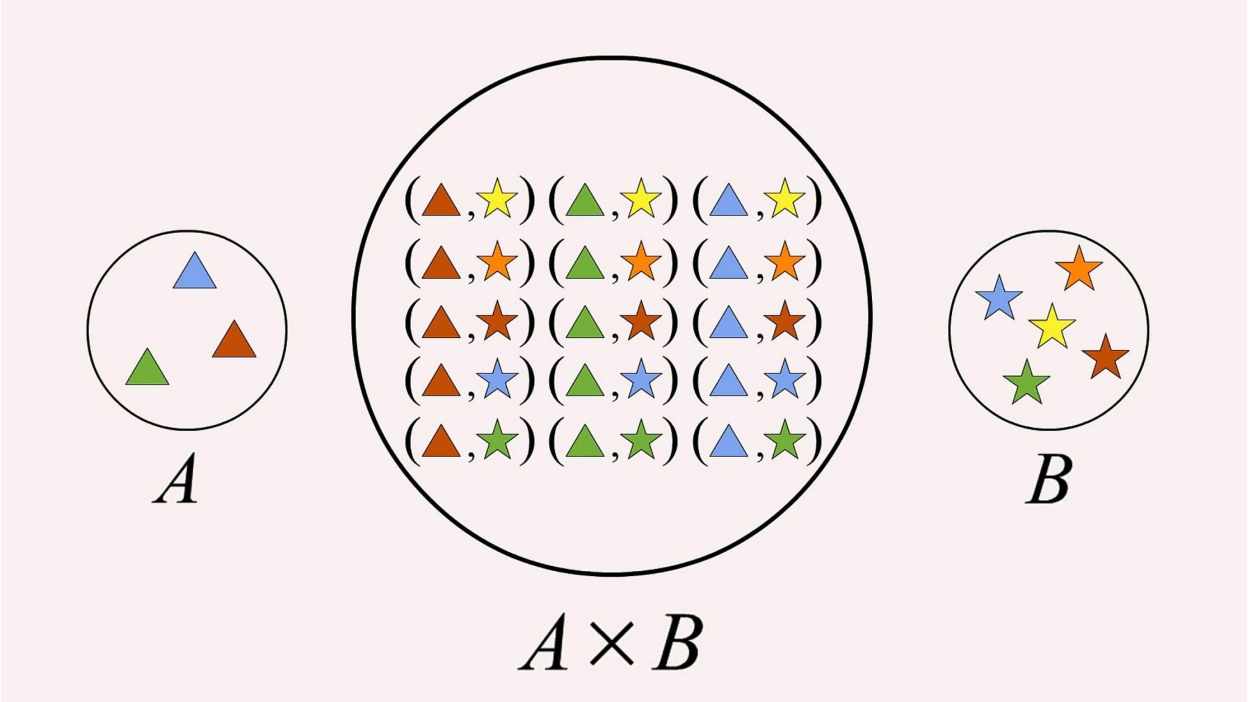
- Firefox: XUL vs WebExtensions, Quantum
- Edge: 14 vs 15 (vs 16?)
- Safari: Safari App Extensions

# DIFFERENT OSES


...

# PLATFORMS

Browser × Version × OS



# PER-PLATFORM CODEBASE

*when?*

- Different/separate product
- Very specialized *(e.g. API- or UX-dependent)*

Disadvantages:

- Expensive to develop *(at least N times more as)*
- Product fragmentation

Not a separate product?

Not specialized to platform?

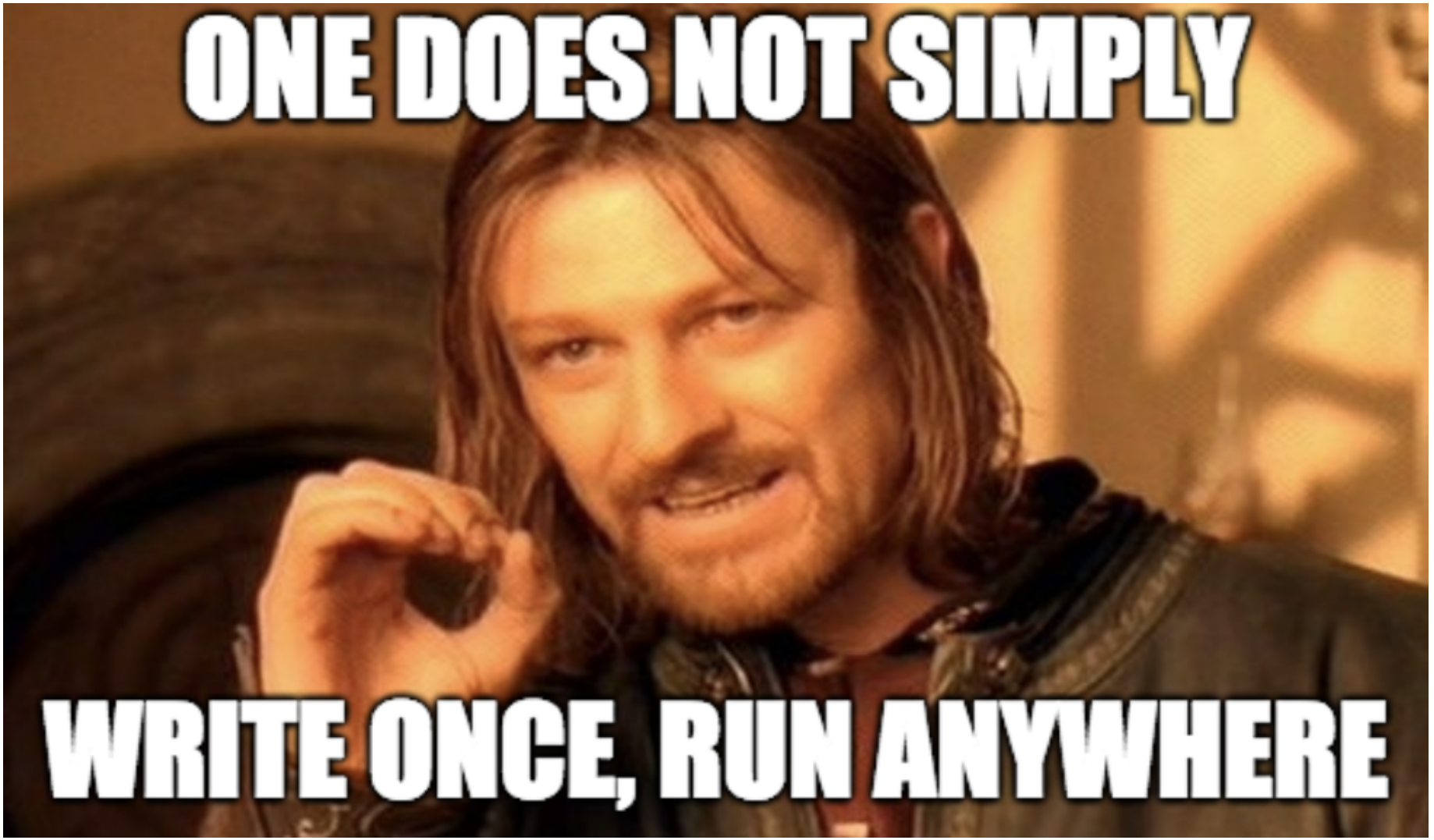
# SHARED CODEBASE

Advantages:

- Faster, cheaper development

Challenges:

- Growing complexity
- Difficult to add cross-platform support later on


ONE DOES NOT SIMPLY
WRITE ONCE, RUN ANYWHERE

# EXAMPLE: COUNTING DOMAIN VISITS

Initial Firefox implementation:

```
function handlePageVisit(domainName) {
  // read user preferences
  browser.storage.local.get('counters').then(storage => {
    const counters = storage.counters || {}
    counters[domainName] = (counters[domainName] || 0) + 1

    // write back the updated count
    browser.storage.local.set({ counters })
  })
}
```

# Add Chrome support?

```
// code reuse! because why not?
window.browser = browser || chrome

function handlePageVisit(domainName) {
  // read user preferences
  browser.storage.local.get('counters').then(storage => {
    const counters = storage.counters || {}
    counters[domainName] = (counters[domainName] || 0) + 1

    // write back the updated count
    browser.storage.local.set({ counters })
  })
}
```

Firefox: returns `Promise`

vs

Chrome: takes a callback

# Add Chrome support:

```
function handlePageVisit(domainName) {
  // assume we're on Firefox if `browser` is defined
  if (typeof browser !== 'undefined') {
    // read user preferences
    browser.storage.local.get('counters').then(storage => {
      const counters = storage.counters || {}
      counters[domainName] = (counters[domainName] || 0) + 1

      // write back the updated count
      browser.storage.local.set({ counters })
    })
  } else {
    // otherwise we're running Chrome
    chrome.storage.local.get('counters', storage => {
      const counters = storage.counters || {}
      counters[domainName] = (counters[domainName] || 0) + 1

      chrome.storage.local.set({ counters })
    })
  }
}
```

# Add Chrome support:

```
function handlePageVisit(domainName) {
  // assume we're on Firefox if `browser` is defined
  if (typeof browser !== 'undefined') {
    // read user preferences
    browser.storage.local.get('counters').then(storage => {
      const counters = storage.counters || {}
      counters[domainName] = (counters[domainName] || 0) + 1

      // write back the updated count
      browser.storage.local.set({ counters })
    })
  } else {
    // otherwise we're running Chrome
    chrome.storage.local.get('counters', storage => {
      const counters = storage.counters || {}
      counters[domainName] = (counters[domainName] || 0) + 1

      chrome.storage.local.set({ counters })
    })
  }
}
```

# DRY:

```
function handlePageVisit(domainName) {
  const increment = (counters = {}, name) => {
    counters[name] = (counters[name] || 0) + 1
    return counters
  }

  if (typeof browser !== 'undefined') { // firefox
    browser.storage.local.get('counters').then(storage => {
      browser.storage.local.set({
        counters: increment(storage.counters, domainName)
      })
    })
  } else { // chrome
    chrome.storage.local.get('counters', storage => {
      chrome.storage.local.set({
        counters: increment(storage.counters, domainName)
      })
    })
  }
}
```

# Propagate errors, return a `Promise`:

```
function handlePageVisit(domainName) {
  const increment = (counters = {}, name) => {
    counters[name] = (counters[name] || 0) + 1
    return counters
  }

  if (typeof browser !== 'undefined') { // firefox
    return browser.storage.local.get('counters').then(storage => {
      return browser.storage.local.set({
        counters: increment(storage.counters, domainName) })
    })
  } else { // chrome
    return new Promise(resolve => {
      chrome.storage.local.get('counters', storage => {
        chrome.storage.local.set({
          counters: increment(storage.counters, domainName) }, resolve)
      })
    })
  }
}
```

# Handle errors properly:

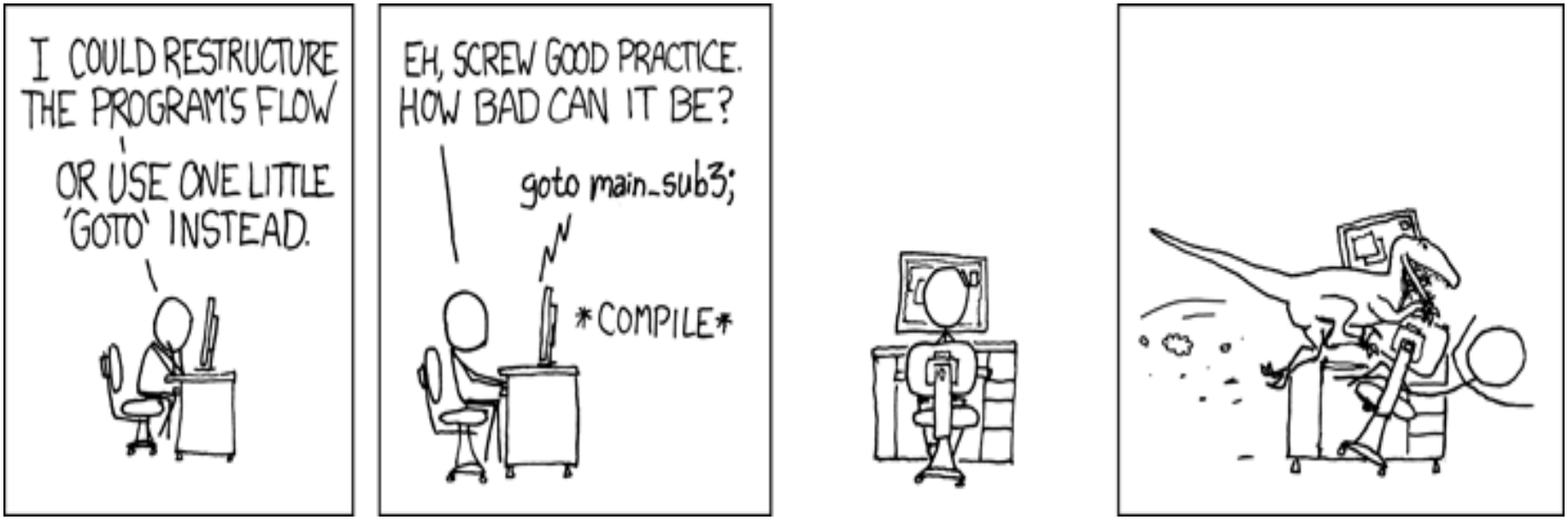
```
function handlePageVisit(domainName) {
  const increment = (counters = {}, name) => {
    counters[name] = (counters[name] || 0) + 1
    return counters
  }

  if (typeof browser !== 'undefined') { // firefox
    return browser.storage.local.get('counters').then(storage => {
      return browser.storage.local.set({ counters: increment(storage.counters, domainName) })
    })
  } else { // chrome
    return new Promise((resolve, reject) => {
      chrome.storage.local.get('counters', storage => {
        if (chrome.runtime.lastError) reject(chrome.runtime.lastError)
        else {
          chrome.storage.local.set({ counters: increment(storage.counters, domainName) }, () => {
            if (chrome.runtime.lastError) reject(chrome.runtime.lastError)
            else resolve()
          })
        }
      })
    })
  }
}
```

# Add Safari support:

```
function handlePageVisit(domainName) {
  const increment = (counters = {}, name) => {
    counters[name] = (counters[name] || 0) + 1
    return counters
  }

  if (typeof browser !== 'undefined') { // firefox
    return browser.storage.local.get('counters').then(storage => {
      return browser.storage.local.set({ counters: increment(storage.counters, domainName) })
    })
  } else if (typeof safari !== 'undefined') { // safari
    return new Promise(resolve => {
      const counters = safari.extension.settings.getItem('counters')
      safari.extension.settings.setItem('counters', increment(counters, domainName))
      resolve()
    })
  } else { // chrome
    return new Promise((resolve, reject) => {
      chrome.storage.local.get('counters', storage => {
        if (chrome.runtime.lastError) reject(chrome.runtime.lastError)
        else {
          chrome.storage.local.set({ counters: increment(storage.counters, domainName) }, () => {
            if (chrome.runtime.lastError) reject(chrome.runtime.lastError)
            else resolve()
          })
        }
      })
    })
  }
}
```


I COULD RESTRUCTURE THE PROGRAM'S FLOW
OR USE ONE LITTLE 'GOTO' INSTEAD.
EH, SCREW GOOD PRACTICE. HOW BAD CAN IT BE?
goto main_sub3;
*COMPILE*

Application code mixed with platform code?

# NOT SCALABLE

- Unnecessary code complexity (too much noise)
- Hard to maintain (tests are now mixed too)
- Hard to develop (adding new platforms, APIs)
- Hard to test (global dependencies)
- Shipping unreacheable code

```
function handlePageVisit(domainName) {
  const increment = (counters = {}, name) => {
    counters[name] = (counters[name] || 0) + 1
    return counters
  }

  if (typeof browser !== 'undefined') { // firefox
    return browser.storage.local.get('counters').then(storage => {
      return browser.storage.local.set({ counters: increment(storage.counters, domainName) })
    })
  } else if (typeof safari !== 'undefined') { // safari
    return new Promise(resolve => {
      const counters = safari.extension.settings.getItem('counters')
      safari.extension.settings.setItem('counters', increment(counters, domainName))
      resolve()
    })
  } else { // chrome
    return new Promise((resolve, reject) => {
      chrome.storage.local.get('counters', storage => {
        if (chrome.runtime.lastError) reject(chrome.runtime.lastError)
        else {
          chrome.storage.local.set({ counters: increment(storage.counters, domainName) }, () => {
            if (chrome.runtime.lastError) reject(chrome.runtime.lastError)
            else resolve()
          })
        }
      })
    })
  }
}
```

```
function handlePageVisit(domainName) {
  // actual application logic
  const increment = (counters = {}, name) => {
    counters[name] = (counters[name] || 0) + 1
    return counters
  }

  // ... platform-specific stuff skipped ...
}
```

# WHAT CAN WE DO?

Separate application and platform code

**Application**: increment the count

**Platform**: store the counter value

# RULE:

## REMOVE `if`S

*Shift branching from caller to callee*

# Perfect application logic code:

```
function handlePageVisit(domainName) {
  // call into our cross-platform API
  return updateStorage('counters', counters => {
    // domain-specific code
    counters[domainName] = (counters[domainName] || 0) + 1
    return counters
  })
}
```

# Platform: **storage update**

```
function updateStorage(name, updateFn) {
  if (typeof browser !== 'undefined') { // firefox
    return browser.storage.local.get(name).then(storage => {
      return browser.storage.local.set({ [name]: updateFn(storage[name]) })
    })
  } else if (typeof safari !== 'undefined') { // safari
    return new Promise(resolve => {
      const value = safari.extension.settings.getItem(name)
      safari.extension.settings.setItem(name, updateFn(value))
      resolve()
    })
  } else { // chrome
    return new Promise((resolve, reject) => {
      chrome.storage.local.get(name, storage => {
        if (chrome.runtime.lastError) reject(chrome.runtime.lastError)
        else {
          chrome.storage.local.set({ [name]: updateFn(storage[name]) }, () => {
            if (chrome.runtime.lastError) reject(chrome.runtime.lastError)
            else resolve()
          })
        }
      })
    })
  }
}
```

# NOT BAD!

- ~~Unnecessary code complexity~~ *(solved!)*
- ~~Hard to maintain~~ *(solved! more or less...)*
- Hard to develop (adding new platforms, APIs)
- Hard to test (global dependencies)
- Shipping unreacheable code

but...

Platform-specific code still mixed together

```
function updateStorage(name, updateFn) {
  if (typeof browser !== 'undefined') { // firefox
    return browser.storage.local.get(name).then(storage => {
      return browser.storage.local.set({ [name]: updateFn(storage[name]) })
    })
  } else if (typeof safari !== 'undefined') { // safari
    return new Promise(resolve => {
      const value = safari.extension.settings.getItem(name)
      safari.extension.settings.setItem(name, updateFn(value))
      resolve()
    })
  } else { // chrome
    return new Promise((resolve, reject) => {
      chrome.storage.local.get(name, storage => {
        if (chrome.runtime.lastError) reject(chrome.runtime.lastError)
        else {
          chrome.storage.local.set({ [name]: updateFn(storage[name]) }, () => {
            if (chrome.runtime.lastError) reject(chrome.runtime.lastError)
            else resolve()
          })
        }
      })
    })
  }
}
```

# WHAT CAN WE DO?

Separate platform-specific implementations

 `browser.storage.local`

 `chrome.storage.local`

 `safari.extension.settings`

# REMOVE ifS

## api/firefox.js:

```
export function updateStorage(name, updateFn) {
  // ... use browser.storage.local ...
}
```

## api/chrome.js:

```
export function updateStorage(name, updateFn) {
  // ... use chrome.storage.local ...
}
```

## api/safari.js:

```
export function updateStorage(name, updateFn) {
  // ... use safari.extension.settings ...
}
```

# app.js:

```
import * as firefoxApi from './api/firefox'
import * as chromeApi from './api/chrome'
import * as safariApi from './api/safari'

let api
if (typeof browser !== 'undefined') api = firefoxApi
else if (typeof safari !== 'undefined') api = safariApi
else api = chromeApi

function handlePageVisit(domainName) {
  // call into our cross-platform API
  return api.updateStorage('counters', counters => {
    // domain-specific code
    counters[domainName] = (counters[domainName] || 0) + 1
    return counters
  })
}
```

# BETTER

- ~~Unnecessary code complexity~~
- ~~Hard to maintain~~
- ~~Hard to develop~~ *(fixed!)*
- Hard to test (global dependencies)
- Shipping unreacheable code

```
import * as firefoxApi from './api/firefox'
import * as chromeApi from './api/chrome'
import * as safariApi from './api/safari'

let api
if (typeof browser !== 'undefined') api = firefoxApi
else if (typeof safari !== 'undefined') api = safariApi
else api = chromeApi
```

- Global reference to the API
- Importing all of the API implementations

# REMOVE ifS

# REMOVE `if`S

- Inject the API dependency
- Separate entry point for each platform

# app.js:

```
export class App {
  constructor(api) {
    this.api = api
  }

  // ...
  handlePageVisit(domainName) {
    // call into our cross-platform API
    return this.api.updateStorage('counters', counters => {
      // domain-specific code
      counters[domainName] = (counters[domainName] || 0) + 1
      return counters
    })
  }
}
```

## firefox.js:

```
import { FirefoxApi } from './api/firefox'
import { App } from './app'

const app = new App(new FirefoxApi())
app.run()
```

## chrome.js:

```
import { ChromeApi } from './api/chrome'
import { App } from './app'

const app = new App(new ChromeApi())
app.run()
```

## safari.js:

```
import { SafariApi } from './api/safari'
import { App } from './app'

const app = new App(new SafariApi())
app.run()
```

# ✨ DONE! ✨

- ~~Unnecessary code complexity~~
- ~~Hard to maintain~~
- ~~Hard to develop~~
- ~~Hard to test~~
- ~~Shipping unreacheable code~~

# GETTING RID OF `if`S

Linear (`if`less) code is easier to work with[1]

# [1]NOT ALL `if`S ARE BAD

- Don't overdo it: in many cases an `if` is simpler
  - e.g. browser logo image in UI
- Make a judgement call

> *Do I make my code less or more complex by removing this `if`?*

# RED FLAGS

- `if (browser) ...`
- `if (background) ...`

# MORE `if`S TO REMOVE?

Depends.

# CROSS-CONTEXT CODE SHARING

- Does your extension have both background page and content scripts?
- Do you try to reuse code accross those?

*They are different execution contexts!*

# CONTEXT DIFFERENCES

- Different API surface
- Different behavior
- Not just extension APIs

# browser.storage.get

When used within a content script in Firefox versions prior to 52, the Promise returned by `browser.storage.local.get()` is fulfilled with an Array containing one Object. The Object in the Array contains the `keys` found in the storage area, as described above. The Promise is correctly fulfilled with an Object when used in the background context (background scripts, popups, options pages, etc.). When this API is used as `chrome.storage.local.get()`, it correctly passes an Object to the callback function.

# Trying to run content script code on background page?

## ❌ DON'T DO IT:

```
function fetchCounters() {
  // need to be on background page for correct
  // Origin header
  if (background) { // 👈 bad
    return fetch('https://visitcounter.com/api/counters')
  } else {
    // will use messaging to forward the call to background
    // page
    return api.callBg('fetchCounters')
  }
}
```

# MIXING CODE FROM ACCROSS CONTEXTS

Same issues:

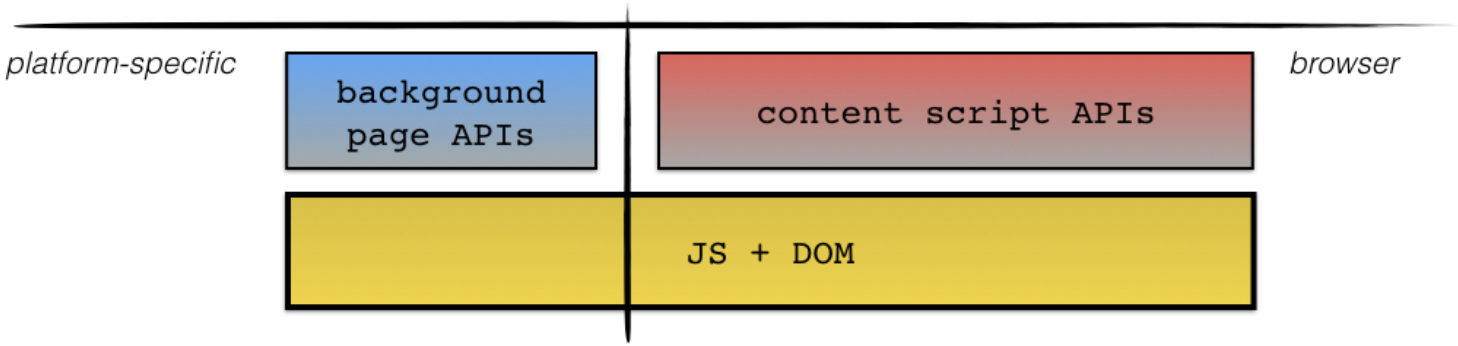
- Added complexity
- Hard to maintain/develop/test
- Shipping unreacheable code

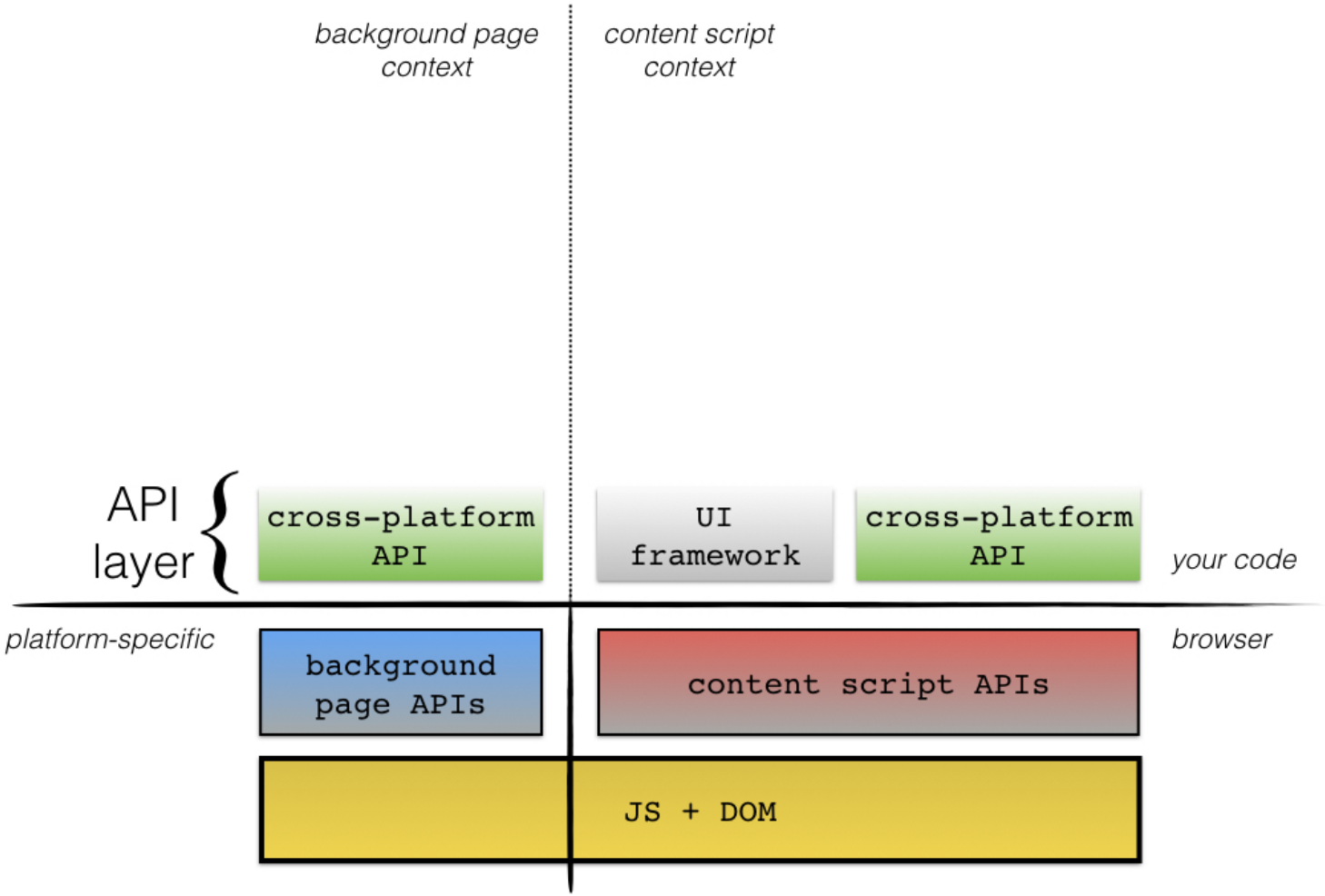
# SOLUTION?

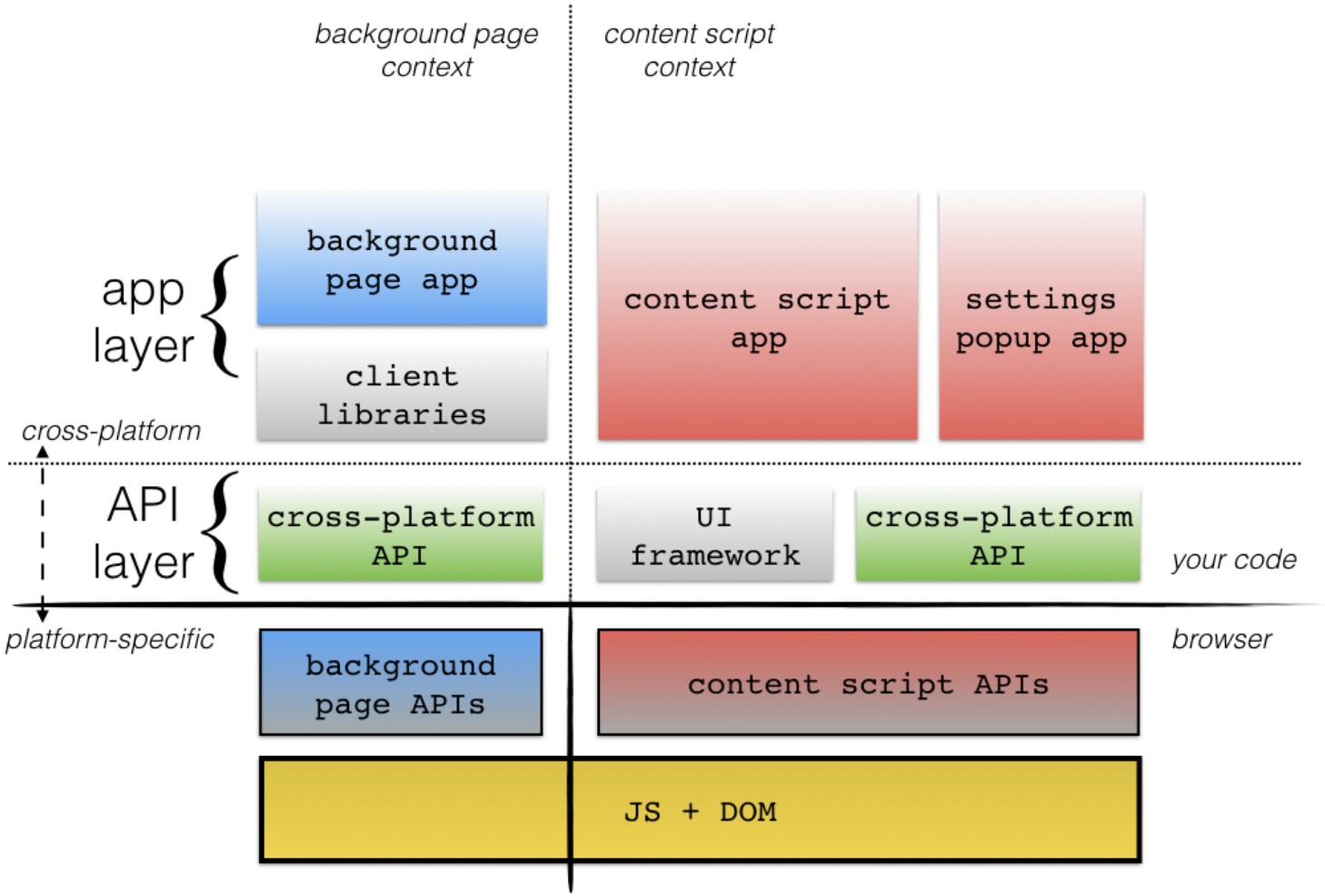
- Remove `if`s, put it in the API layer.
- Watch your bundle (e.g. webpack-bundle-analyzer)

# WHAT WE DID

- Separate application and platform code
- Separate entry points for each platform
- Separate entry points for each context
- Unified cross-platform API interface
- Modular, testable architecture


platform-specific
background page APIs
content script APIs
browser
JS + DOM


background page context
content script context
API layer
cross-platform API
UI framework
cross-platform API
your code
platform-specific
background page APIs
content script APIs
browser
JS + DOM


background page context
content script context
app layer
background page app
client libraries
content script app
settings popup app
cross-platform
API layer
cross-platform API
UI framework
cross-platform API
your code
platform-specific
background page APIs
content script APIs
browser
JS + DOM

# ALREADY EXISTS?

## Introducing OpenForge (2013)

We are now open-sourcing that codebase, calling it **OpenForge**.

This codebase will be far from stale – development of the browser add-on framework will continue with a full-time maintainer who will also provide support. In open-sourcing the framework we have also provided the ability to build locally without reliance on Trigger.io infrastructure.

## github.com/trigger-corp/browser-extensions

# NOT ANYMORE

- Not maintained
- Complicated build process
- Questionable architecture

```
if (background) {
  // background page code
} else {
  // content script code
}
```

# API DESIGN GUIDELINES

- Implement for both extension and general JS
- Sufficiently high-level *(hide complexity)*
- Uniform *(least common denominator)*
- Custom-tailored *(special, not general purpose)*
- Only necessary APIs *(don't do what you don't need)*
- Modular *(no globals, no implicit effects)*

# FURTHER IMPROVEMENTS

Use static typing (we use TypeScript)

- Easier to enforce the architecture

*Bonus:*

- Tests that you don't have to write
- Documentation that stays up to date
- Code intent expressed more clearly

# STATIC TYPING IS COOL

- https://jaredforsyth.com/type-systems-js-dev/

  Type systems will make you a better JavaScript programmer

- https://goo.gl/jkHcSQ

  Designing with types: Making illegal states unrepresentable (F#)

# CONCLUSION

- Design for cross-platform from the start
  - Good design will scale *(even with one platform)*
- Follow good practices
  - Separate concerns and try to keep code simple
- Embrace the tools
  - Compiler is your friend

# Q & A


WE’RE HIRING!
grammarly.com/jobs