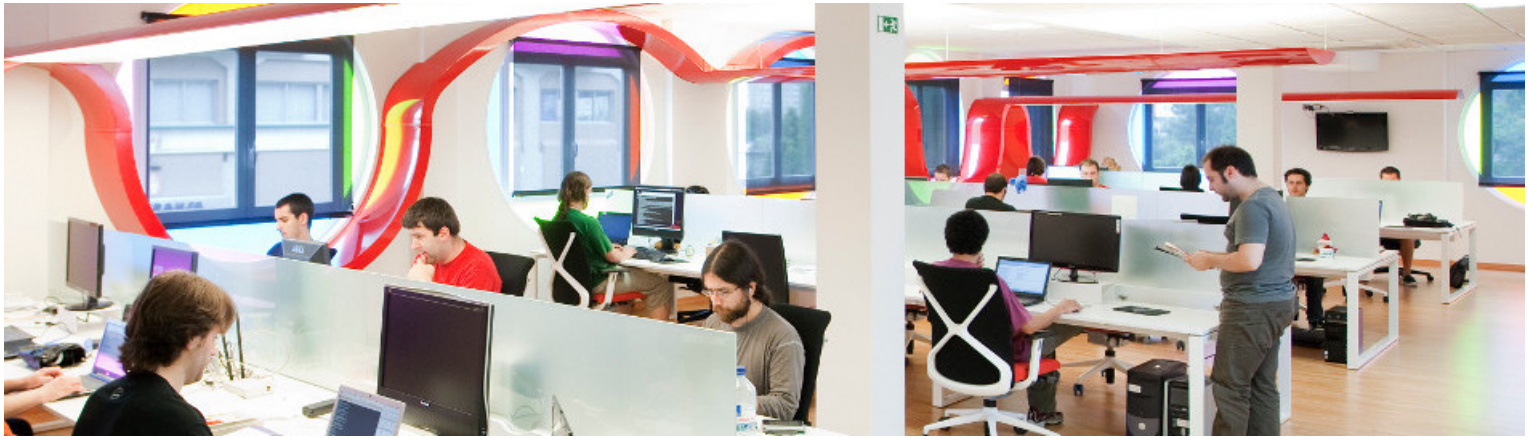
Building a Browser for Automotive: Alternatives, Challenges and Recommendations

Juan J. Sánchez

Automotive Linux Summit 2015, Tokyo

igalia

Myself, Igalia and Webkit/Chromium



- Co-founder of Igalia
- Open source consultancy founded in 2001
- Igalia is Top 5 contributor to upstream WebKit/Chromium
- Working with many industry actors: automotive, tablets, phones, smart tv, set-top boxes, IVI and home automation

Outline

1. A browser for automotive: requirements and alternatives
2. WebKit and Chromium, a historical perspective
3. Selecting between WebKit and Chromium based alternatives

PART 1
A browser for automotive: requirements and alternatives

Requirements

- Different User Experiences
  - UI modifications (flexibility)
  - New ways of interacting: accessibility support
- Support of specific standards (mostly communication and interfaces)
- Portability: support of specific hardware boards (performance optimization)
- Functionality and completeness can be less demanding in some cases (for now)
- Provide both browser as an application and as a runtime

igalia

Available alternatives

- Option 1) Licensing a proprietary solution: might bring a reduced time-to-market but involves a cost per unit and lack of flexibility
- Option 2) Deriving a new browser from the main open source browser technologies:
  - Firefox (Gecko)
  - Chromium
  - WebKit (Safari and others)
- Mozilla removed support in their engine for third party browser developers, so the two available choices are Chromium and WebKit (with various options for each of them)

igalia

Understanding the main alternatives

- When creating a new open source browser for automotive, a decission between Chromium and WebKit will need to be made
- Chromium and Webkit share a lot of history, design and code
- Learning how WebKit was created, and how Chromium emerged and derived from WebKit, improves the understanding of the pros and cons of each solution
- We will make a detailed historical review of both projects

PART 2
WebKit and Chromium, a historical perspective

PART 2.1: 2004-2013
WebKit, the first 9 years

The WebKit project

- **Web rendering engine** (*HTML*, *JavaScript*, *CSS*...)
- The engine is the product
- Started as a fork of KHTML and KJS in 2001
- *Open Source* since 2005
- Among other things, it's **useful for**:
  - Web browsers
  - Using web technologies for UI development

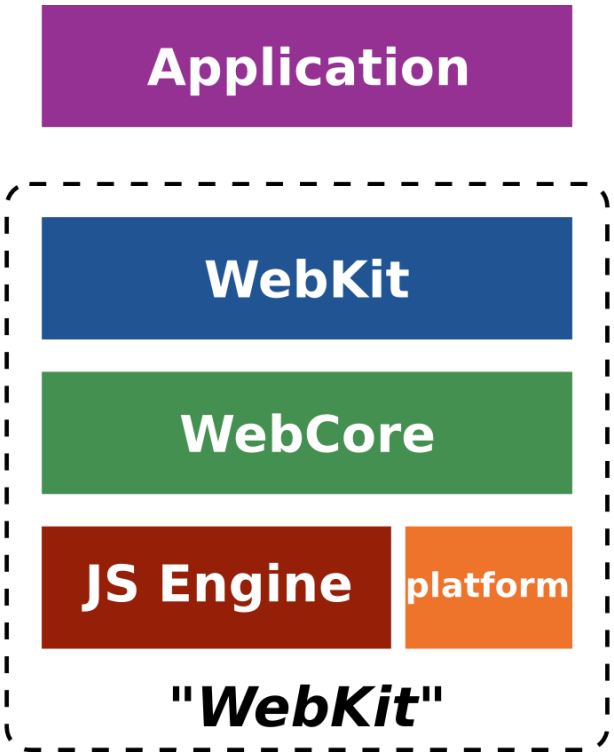
WebKit features

- HTML and XML support
- JavaScript support (ECMAScript 5.1)
- CSS 2.1, CSS 3 support
- SVG support
- Support for Plugins (NPAPI, WebKit Plugins)
- HTML5 support: multimedia, 3D graphics, advanced CSS animations and transformations, drag'n'drop, offline & local storage, connectivity...
- Accessibility support
- Q&A infrastructure: review process, continuous integration,
- Passing ACID3 with 100/100 tests since March 2008

igalia

WebKit Architecture

From a simplified point of view, WebKit is structured this way:



- **WebKit**: thin layer to link against from the applications
- **WebCore**: rendering, layout, network access, multimedia, accessibility support...
- **JS Engine**: the JavaScript engine. JavaScriptCore by default.
- **platform**: platform-specific *hooks* to implement generic algorithms

igalia

What is a WebKit port?

iOS

igalia

How many WebKit ports are there?

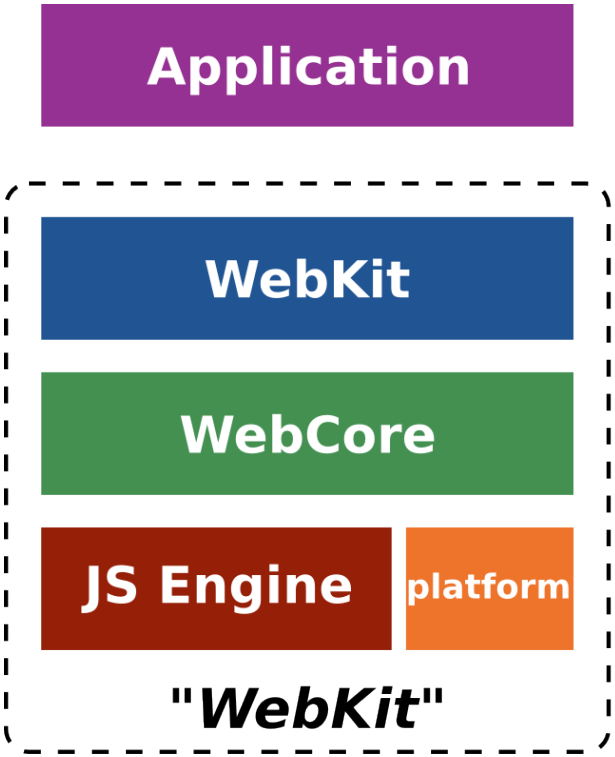
WebKit is available for different platforms:

- Main upstream ports in 2012/2013:
  - Mac OS X, iOS
  - GTK+ based platforms (GNOME)
  - Qt based platforms (KDE)
  - Enlightenment Foundation Libraries (EFL, Tizen)
  - Google Chromium / Chrome
  - WebKitNIX
- Other ports: wxWidgets, Brew MP, Symbian devices (S60), Win32, BlackBerry, Adobe Integrated Runtime (Adobe AIR)

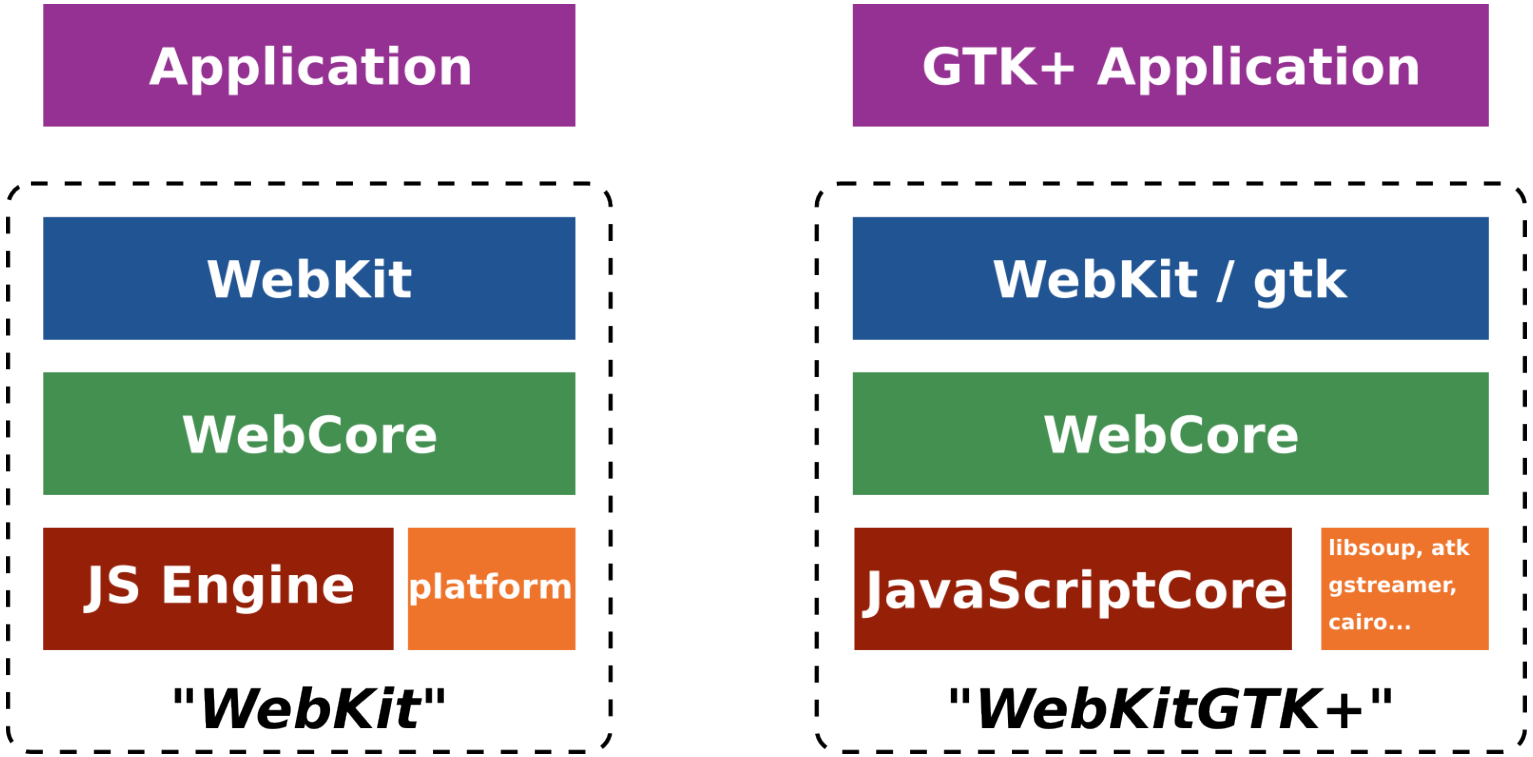
Some WebKit based browsers in 2013

- Amazon Kindle
- Arora
- BOLT browser
- Epiphany browser
- Google Chrome
- iCab (version >= 4)
- Iris Browser
- Konqueror
- Midori
- Nintendo 3DS
- OWB
- OmniWeb
- PS3 web browser
- RockMelt
- Safari
- SRWare Iron
- Shiira
- Sputnik for MorphOS
- Stainless
- Steel for Android
- TeaShark
- Uzbl
- Web Browser for S60 (Nokia)
- WebOS Browser

Architecture of a WebKit port



Architecture of a WebKit port



How do we use a WebKit port?

- **The WebView *widget***:
  A platform-specific widget that renders web content.

  It's the **main component** and it's useful for:
  - Loading URIs or data buffers pointing to HTML content
  - Go fullscreen, text/text+image zooming...
  - Navigate back and forward through history...

- **Events handling**:
  Allows embedders to get notified when something important happens or when some input is needed.

  Some examples of these events:
  - Getting notified when a load finished or failed
  - Asking permission for navigating to an URI
  - Requesting authorization for something..

A minibrowser written in Python

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import gtk
import webkit

def entry_activated_cb(entry, embed):
    embed.load_uri(entry.get_text())

# Widgets and signals
window = gtk.Window()
window.set_default_size(800, 600)
window.set_title("Mini browser written in Python")
embed = webkit.WebView(); # WebKit embed
entry = gtk.Entry()
entry.connect('activate', entry_activated_cb, embed)
scroller = gtk.ScrolledWindow()
scroller.add(embed)

# Pack everything up and show
vbox = gtk.VBox(False, 5)
vbox.pack_start(entry, False, False)
vbox.pack_start(scroller)
window.add(vbox)
window.show_all()

# Load a default URI and run
embed.load_uri("http://www.webkit.org")
gtk.main()
```

igalia

A minibrowser written in Python

What is WebKit2?

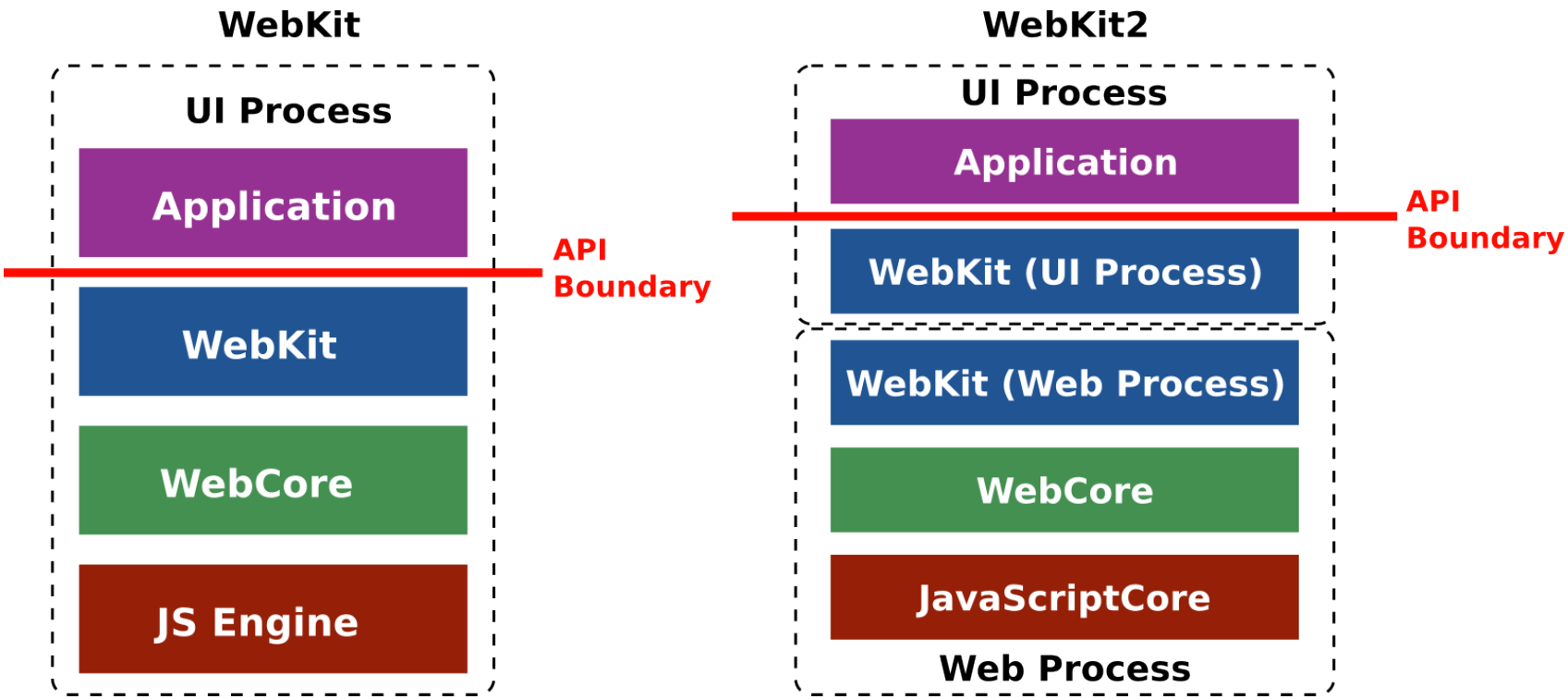
- New API layer designed to support a split process model (First release by Apple on April 8th, 2010[1]).
- Different to Chromium's multi-process implementation

  It's bundled in the framework (reusable)

- Different processes take care of different tasks:
  - **UI process**: the WebView *widget*, application UI
  - **Web process**: loading, parsing, rendering, layout...
  - **Plugin process**: each plugin type in a process
- It comes with Inter-Process Communication (IPC) mechanisms to communicate those processes bundled-in

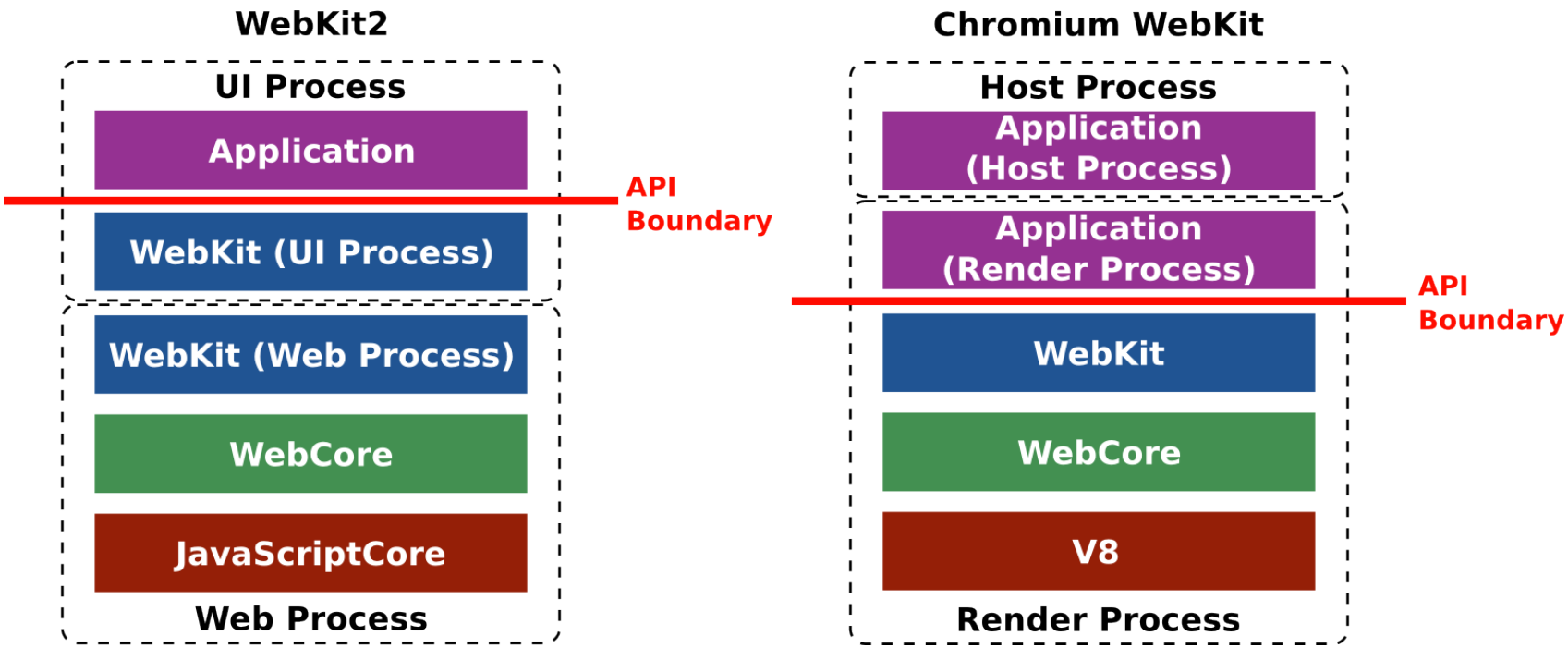
`http://trac.webkit.org/wiki/WebKit2`

WebKit VS WebKit2



Advantages: isolation, security, performance, stability.

WebKit2 VS Chromium

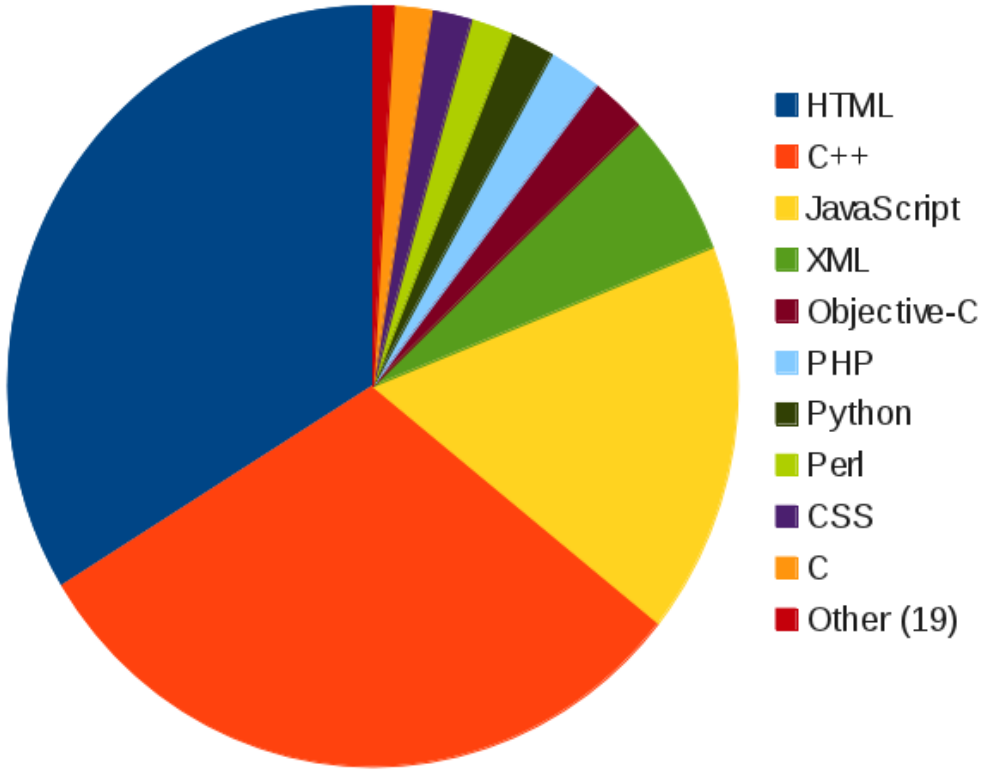


Chromium first released in late 2008.

The Source Code in numbers

According to Ohloh on May 17th, **lines of code per language**, without considering blank lines nor comments:

Language	LoC	%
HTML	1,955,561	32.4 %
C++	1,308,667	27.5 %
JavaScript	962,086	20.8 %
Objective-C	175,669	3.4 %
XML	158,555	2.5 %
C	121,951	3.0 %
PHP	100,345	2.3 %
CSS	93,247	1.6 %
Python	78,348	1.9 %
Perl	76,491	1.7 %
OpenGL Shad	52,234	1.8 %
Other (16)	50,000	1.1 %
Total	4,132,955	



`https://www.ohloh.net/p/WebKit/analyses/latest/language_summary`

Just considering C++, Objective-C and C >1.6M LoC!

Licenses: BSD 3-clause and LGPL

igalia

The WebKit Project in numbers

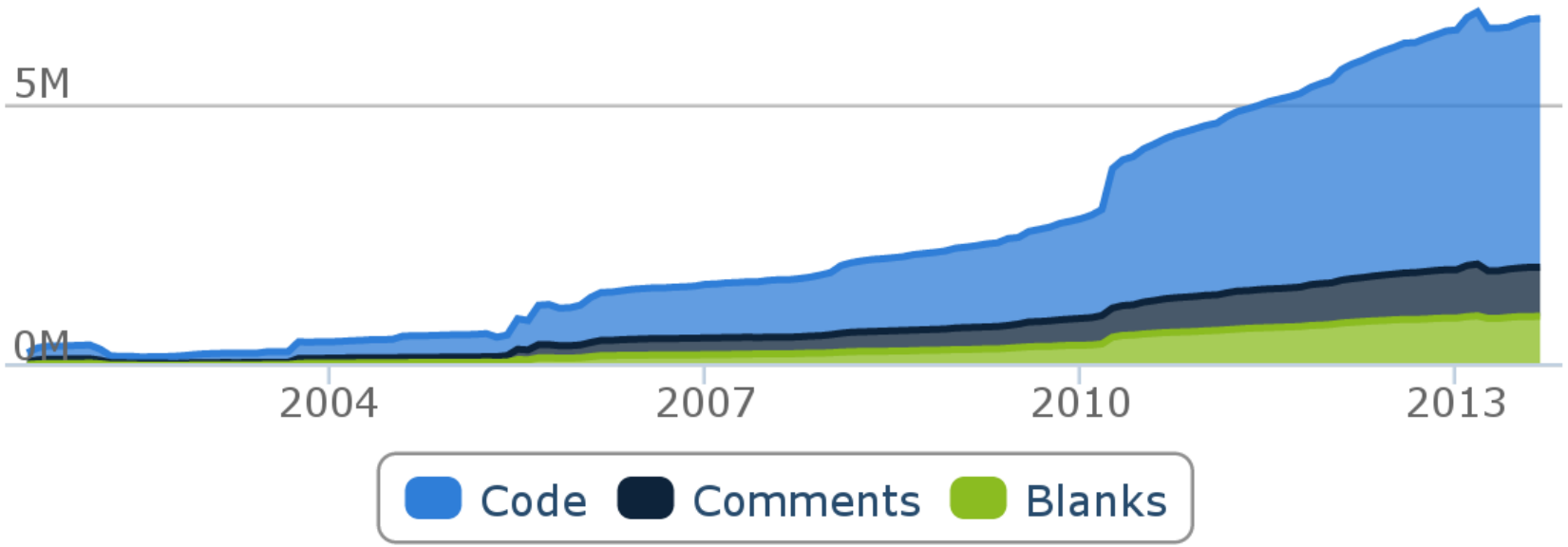
Commits per month till 2013:

The WebKit Project in numbers

Contributors per month::

The WebKit Project in numbers

Evolution in the number of lines of code



Activity of Companies by 2013

- Based on Bitergia's report[2]
- Based on reviewed commits
- "Gardening" commits filtered out
- From the beginning of the project till beginning of 2013

[2] http://blog.bitergia.com/2013/02/06/report-on-the-activity-of-companies-in-the-webkit-project/

Activity of Companies by 2013

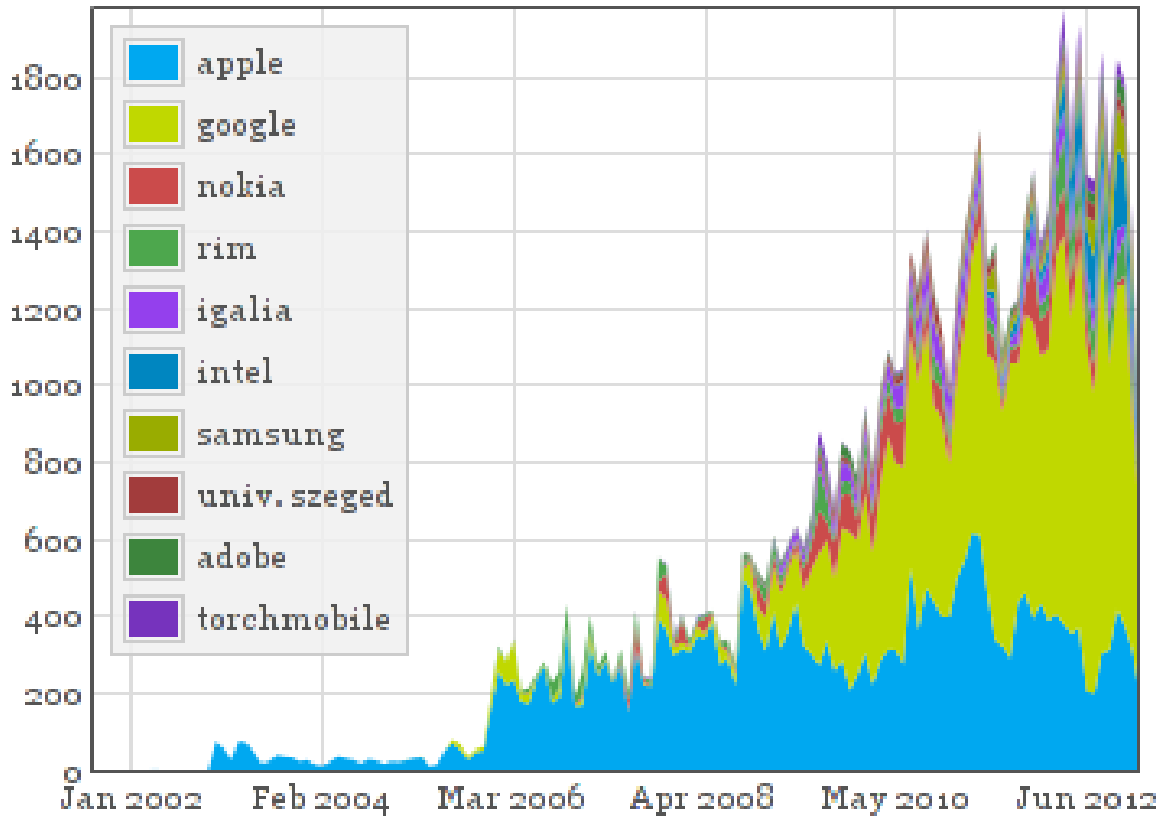


Figura : Commits per company (monthly)

Activity of Companies by 2013

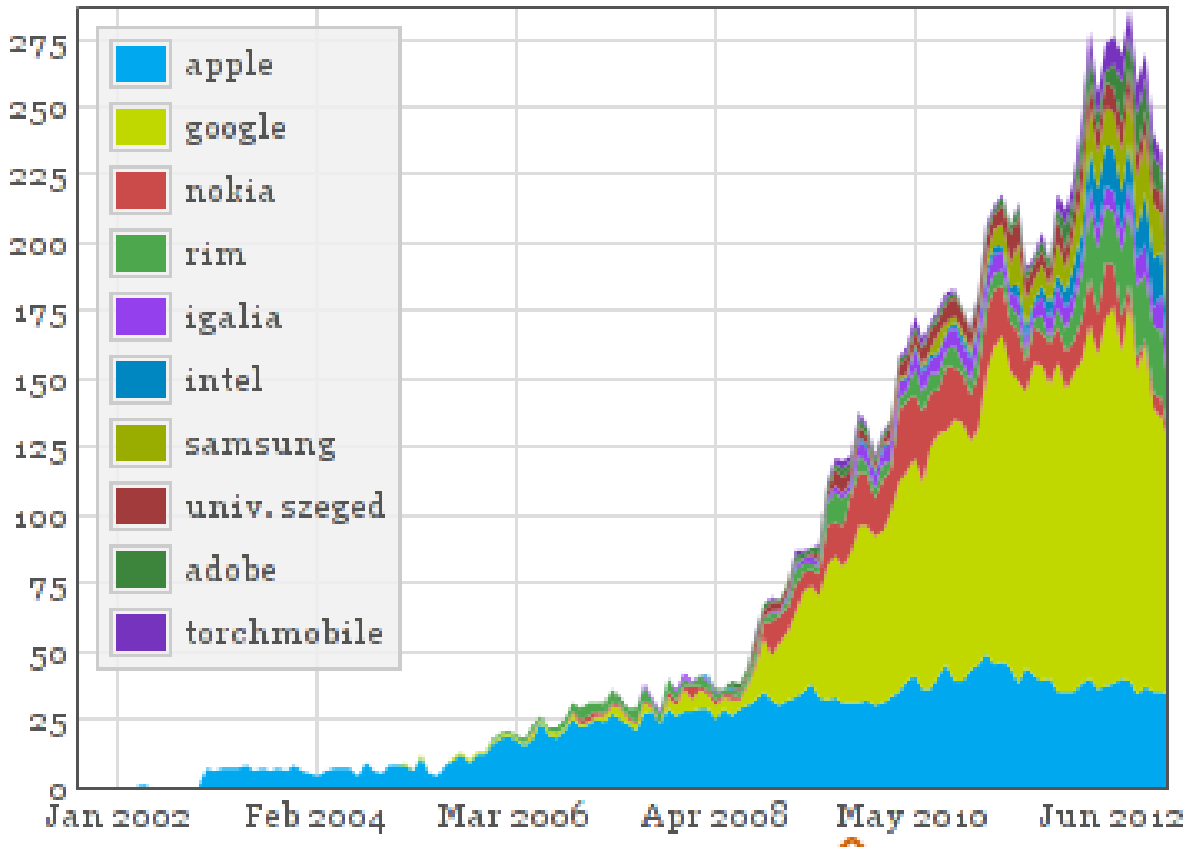


Figura : Active authors per company (monthly)

Activity of Companies by 2013

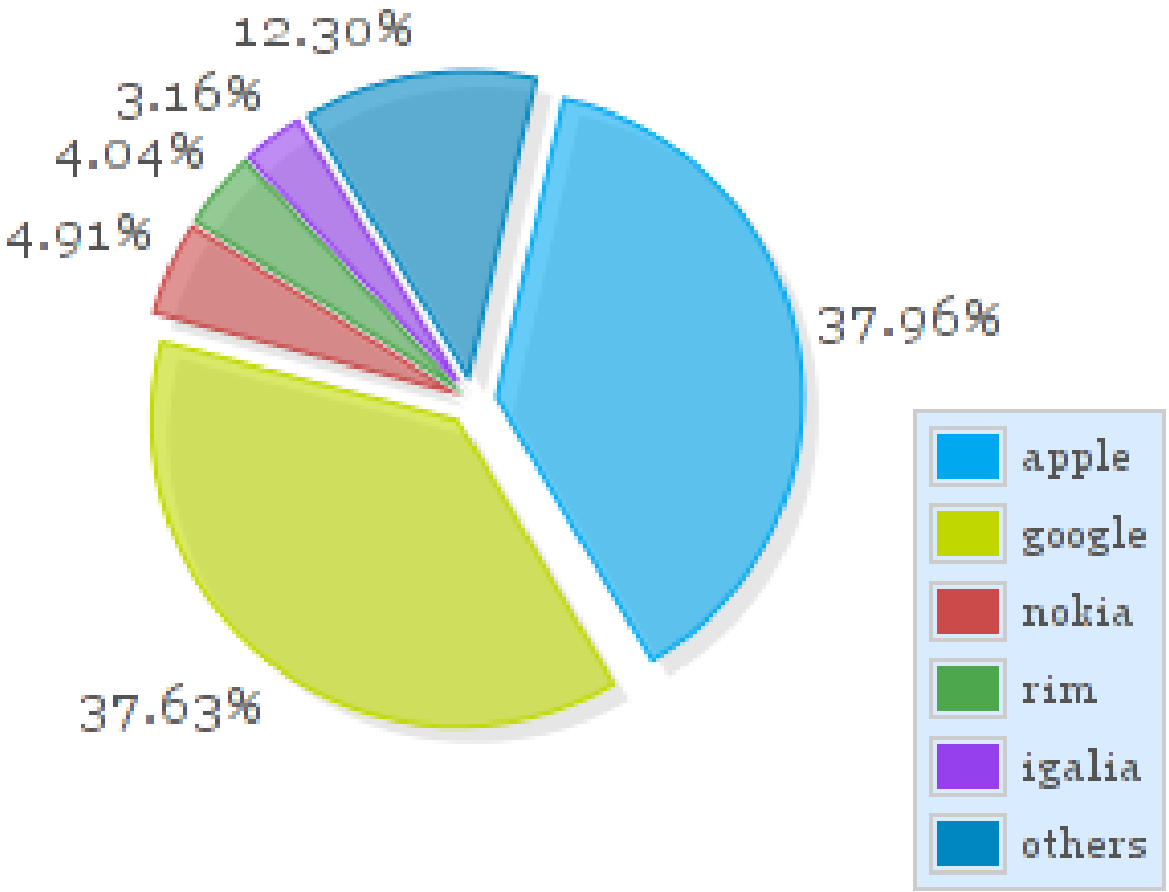


Figura : Commits per company

Activity of Companies by 2013

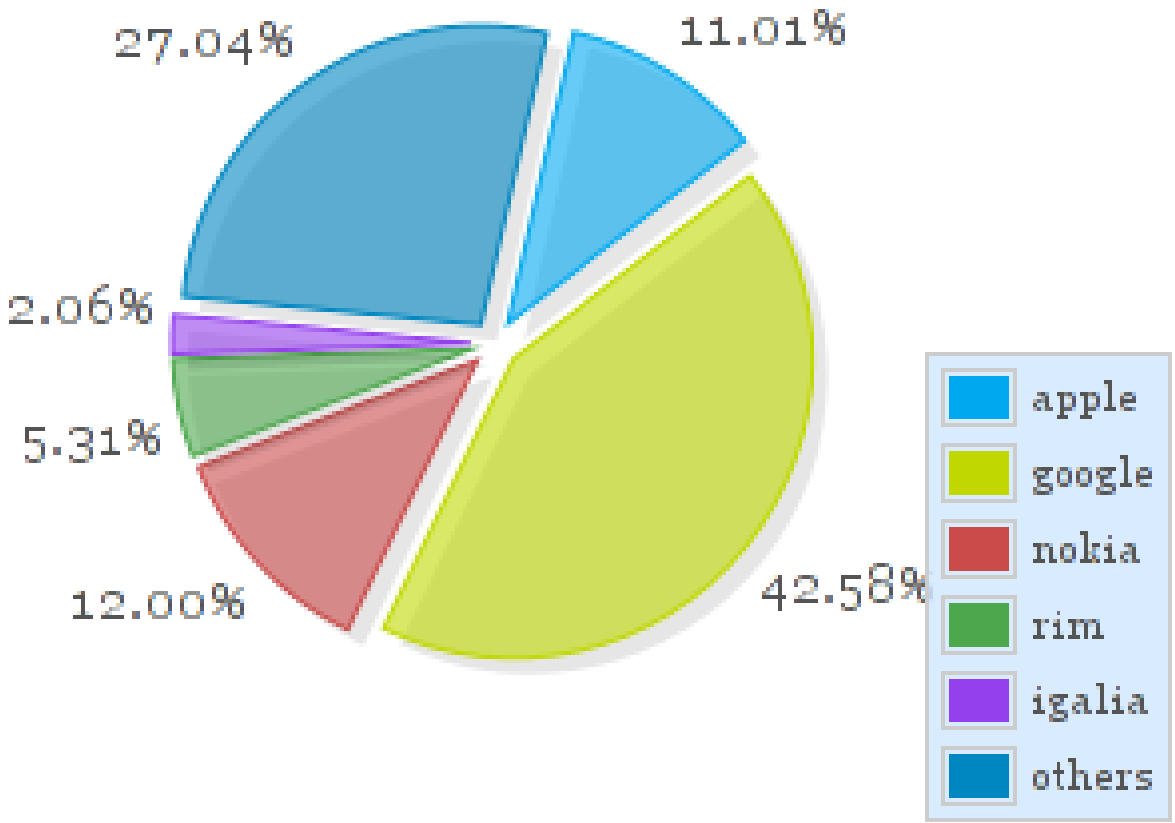


Figura : Active authors per company

Part 2.2

The creation of Blink (April 2013)

Google's Departure. Blink

- Google announced on April 3rd that they would be forking WebKit and creating Blink
- Motivations according to Google:
  - They were not using WebKit2 anyway
  - Easier to do ambitious architectural changes after the fork
  - Simplification of the codebase in Blink
- Tension between Apple and Google before the fork
  - Architectural decisions: NetworkProcess
  - Code governance: Owners need to approve some core changes
- Big shock within the WebKit community

Differences between WebKit and Blink

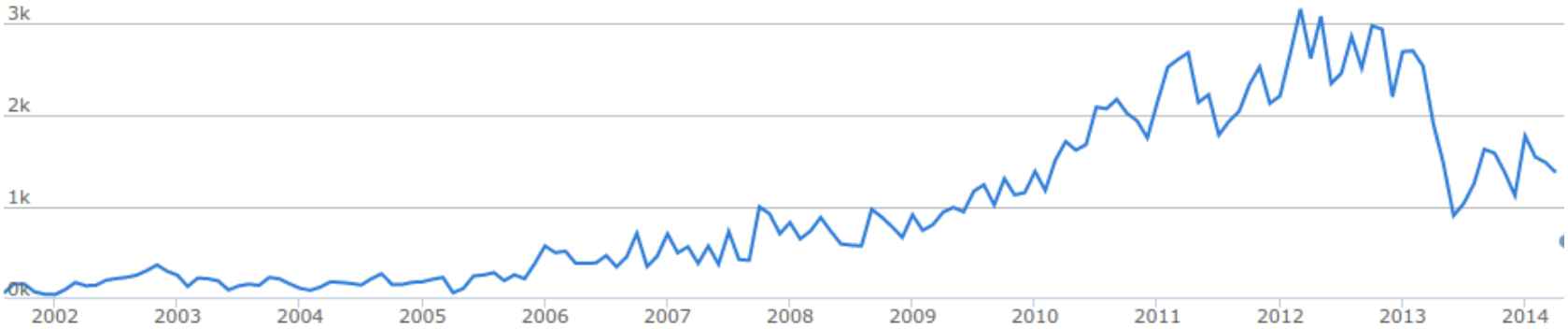
- Removes the concept of 'port' as it was defined in WebKit (deep platform integration): Skia, V8 and other libraries cannot be replaced
- Still possible to use Blink in other platforms, but now integration happens at Content level
- Only the rendering engine. Multi-process architecture is still in Chromium
- WebKit has committers, reviewers and owners (control some core areas). Blink only committers and owners (similar to WebKit reviewers)
- Peer review process a bit more relaxed in Blink
- Many architecture changes

Early consequences of Blink for WebKit

- Google was the main contributor by # of commits. Apple's position now more dominant
- Opera joined WebKit then moved to Blink. Other companies and communities started migrating (Tizen and Qt)
- Several WebCore modules left orphan. Other hackers assuming WebCore modules maintainership
- WebKit developers porting patches from/to Blink
- Many hacks to accomodate Chromium removed. Engines quickly starting to diverge at faster pace

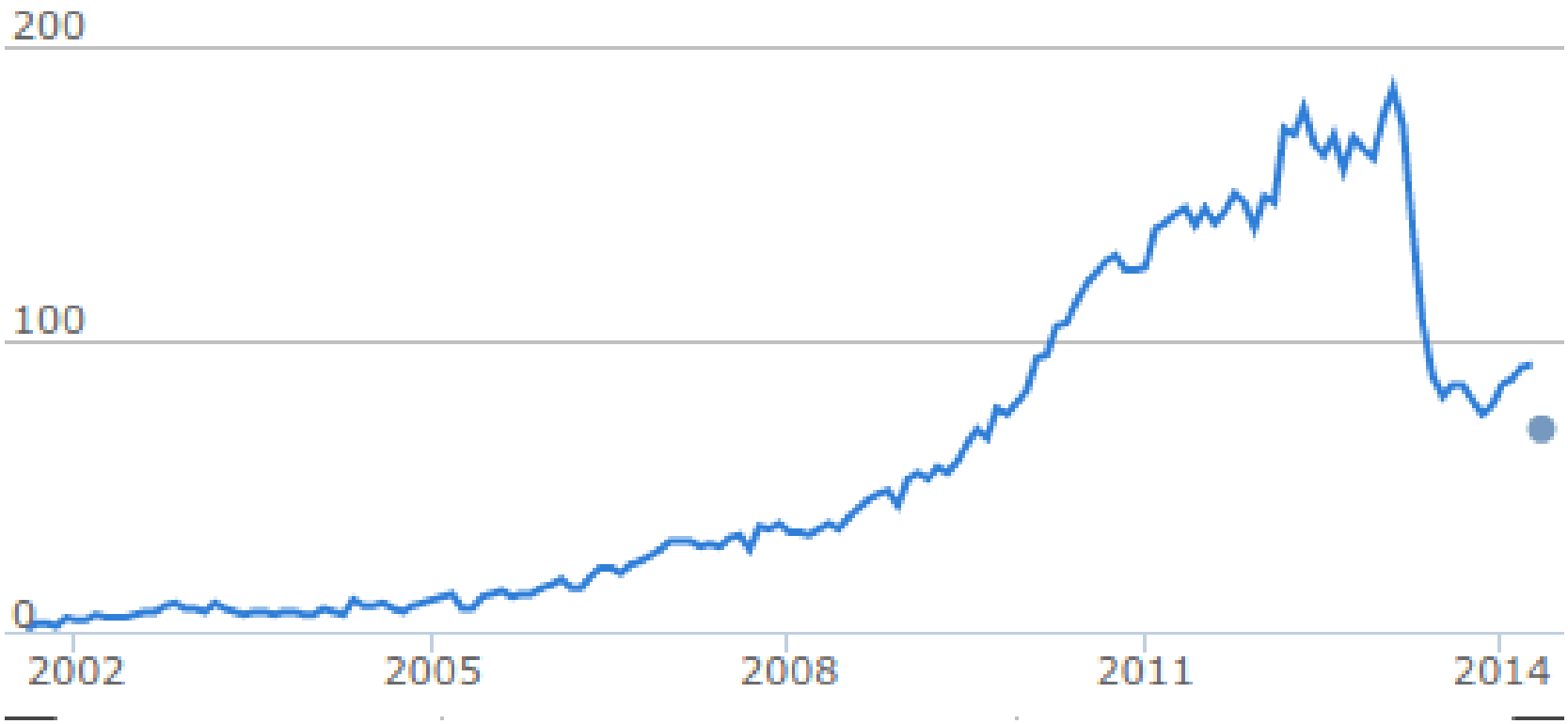
Impact of Blink in numbers

Commits per month in WebKit, including last months:



Impact of Blink in numbers

Contributors per month in WebKit, including first year of Blink:

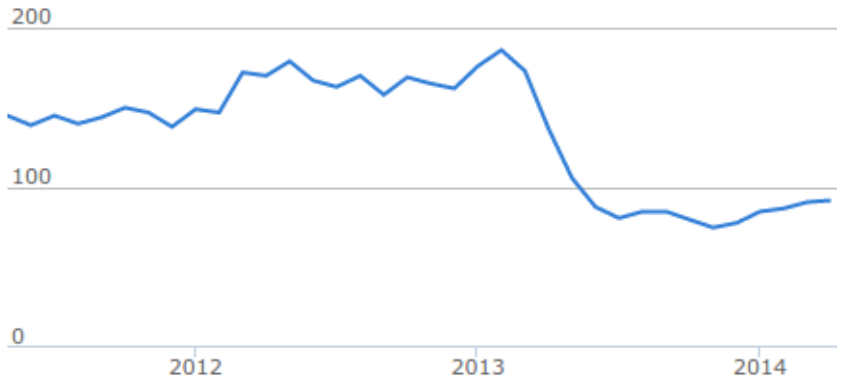


Impact of Blink in numbers

Contributors per month in 2013-2014:

Blink:

WebKit:



Impact of Blink in numbers

Commits per month in 2013-2014, Blink::

Commits per month in 2013-2014, WebKit::

igalia

Webkit and Chromium in 2015

- Less shared energy because of the split, but both projects very active and alive
- There is a recent trend towards more users for Blink and Chromium, but still quite a lot of open questions and challenges
- Both provide good building blocks for creating a browser for automotive

igalia

PART 3
Selecting the best alternative

Alternatives today

- In WebKit you need to select (or create) a port, in Chromium you need to define how to derive from upstream
- WebKit:
  - WebKitGTK+
  - WebKit for wayland
  - WebkitEFL and QtWebkit still in some (mostly) legacy projects
- Chromium:
  - Chromium directly
  - QtWebEngine
  - Crosswalk
  - Cromium Embedded Framework (CEF)

Webkit vs Chromium: pros and cons

- WebKit:
  - Pro: memory footprint is smaller
  - Pro: ports are upstream, easy to integrate core changes
  - Pro: very flexible architecture, easy to plug components
  - Con: less companies contributing (Apple very relevant)
  - Con: less innovation lately in some areas of the codebase
- Chromium:
  - Pro: more innovation happening in some areas, Google driving it with a lot of developers
  - Pro: trend of more and more companies using the technology and testing it
  - Con: no concept of ports, alternatives are not upstream
  - Con: difficult to contribute to some core areas (Google)
  - Con: versions of Chromium diverting a lot from Chrome

WebKitGTK+ and WebKit for Wayland

- Pure open source projects, easy to influence their upstream development
- Reliable and well-known release process and quality maintenance procedures, strong API and ABI compatibility
- Possibility of modifying the whole stack, avoiding a big delta (e.g. gstreamer vs other media frameworks)
- Developed by a relatively small team (compared to Google's Chromium)
- Less widely tested in heterogeneous hardware platforms
- Webkit for wayland brings an interesting alternative to WebKitGTK+ for some use cases, but still not a mature project

Chromium directly

- All the features in the browser and flexibility to use them at the cost of increased maintenance complexity
- Browsing operations implemented interfacing Chromium's Content API. Browser services like history, bookmarks or incognito should be interfaced directly through internal (unstable) APIs
- High risk of ending with a big delta compared to upstream Chromium (which moves very fast)
- Changing the UI requires patching the UI code (no toolkit available)
- Chromium is officially supported on Intel-based Windows, Mac OS X and Linux with X11. Building on top of ARM devices is possible but less directly supported

igalia

Crosswalk

- Crosswalk is an HTML application runtime based on Chromium. It is available for Android as an embeddable webview container and for Tizen as the system-wide application runtime
- Crosswalk usage as a webview for Android difficult to port as it is mostly implemented in Java. Crosswalk reuses and adapts the multiprocess model of Chromium to its needs
- Crosswalk is intended to run applications and not web pages. Building a browser on top requires creating a quite big delta with upstream
- Still a quite new project (created in 2013). Not a big community outside Intel and Tizen
- No backwards compatibility in the new versions

QtWebEngine

- Evolution of the Qt webkit port, but using Chromium
- It was undergoing heavy development until very recently
- Some small open source browsers use it but not focused on being used for browsers, just for embedding small HTML5 parts in Qt 5 applications
- Potential issues with LGPLv3 license for some users. Alternative of paying a license for Qt

Chromium Embedded Framework

- Stable API for development of applications with embedded browsers
- All browser abstractions are preserved, and the multiprocess architecture of Chromium is preserved and properly interfaced
- A browser based on CEF would be an independent application that would incorporate CEF as a library and use its API
- Browser features from layers above the Content API are not present in CEF (history, bookmarks or incognito)
- Officially supported only on Intel-based Windows, Mac OS X and Linux with X11
- Created in 2009. Still mostly a one person project

Conclusions

- There are various alternatives both on WebKit and Chromium to create a derived browser for Automotive
- Different companies and projects are using different solutions. There is none that seems to be good at everything
- The choice largely depends on the weight of the different goals to be achieved with that project and its specific hardware and software needs
- In any case, and independently from the choice, 3 keys for success:
  - Long term analysis of hardware and software requirements
  - Respecting community and open source dynamics (minimum delta, as much upstream as possible)
  - Right team and project scope definition

igalia

Thank you!

Juan J. Sánchez
jjsanchez@igalia.com