Firefox OS App Days

Overview

and High Level Architecture

Author: José M. Cantera (@jmcantera)
Last update: March 2013

bravo!

TELEFÓNICA I+D

Telefónica

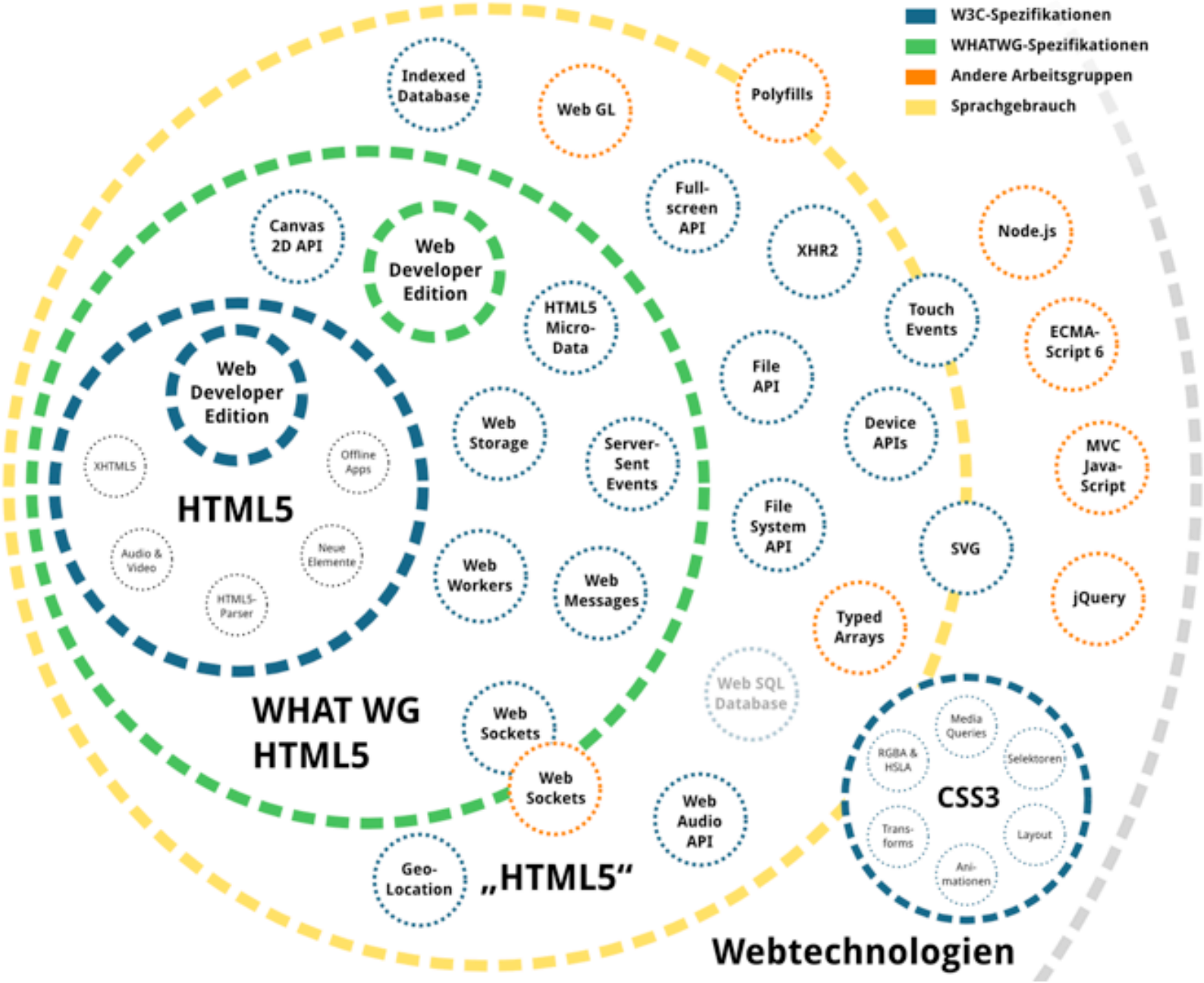
Introduction

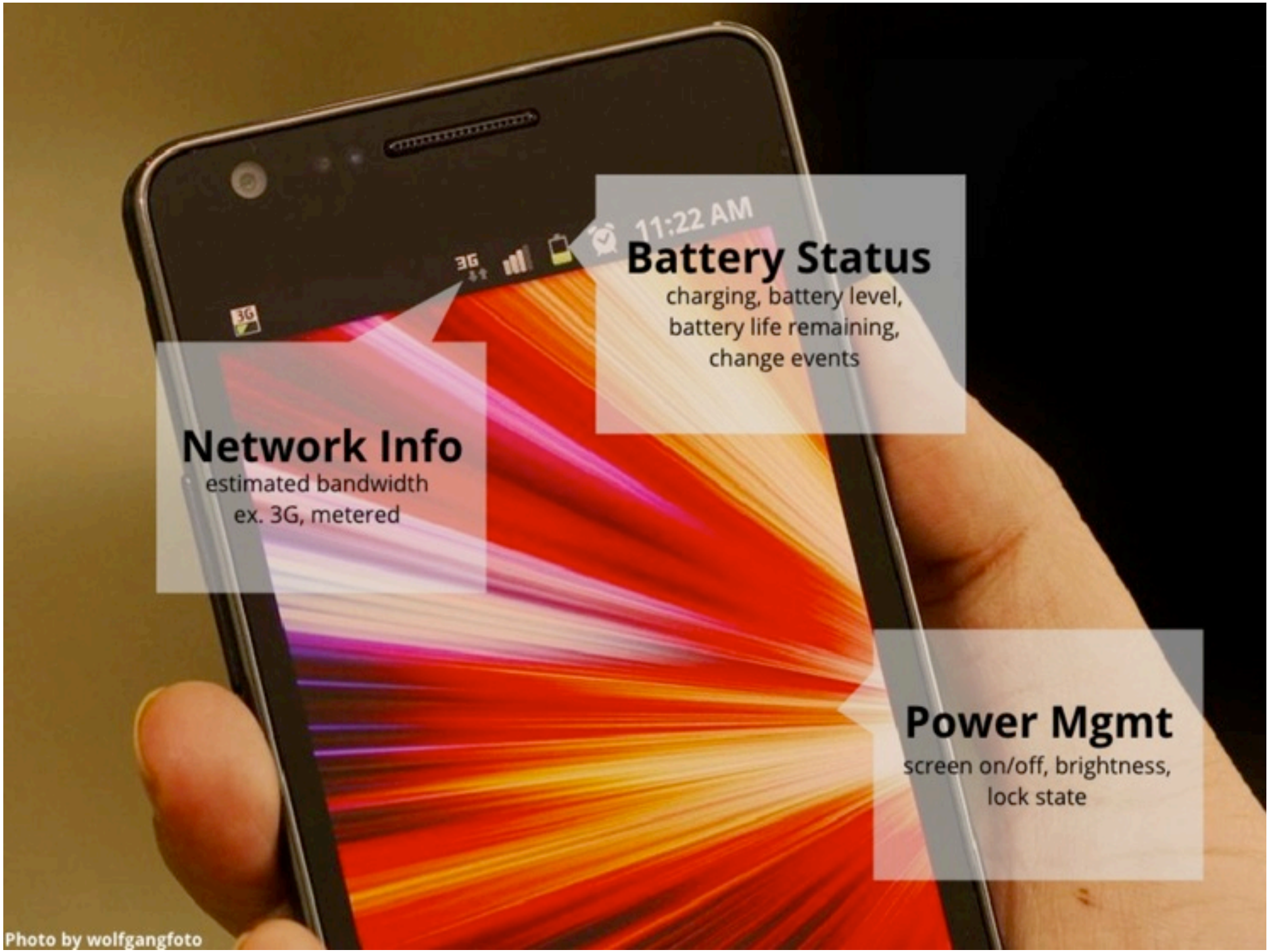
What is Firefox OS?

bravo!

A new mobile open OS fully based on the contemporary Web Platform (HTML5)

bravo!


W3C-Spezifikationen
WHATWG-Spezifikationen
Andere Arbeitsgruppen
Sprachgebrauch
Indexed Database
Web GL
Polyfills
Full-screen API
XHR2
Node.js
Canvas 2D API
Web Developer Edition
HTML5 Micro-Data
Touch Events
ECMA-Script 6
File API
Web Developer Edition
Web Storage
Server-Sent Events
Device APIs
MVC Java-Script
XHTML5
Offline Apps
HTML5
File System API
SVG
Audio & Video
Neue Elemente
Web Workers
Web Messages
jQuery
HTML5-Parser
Typed Arrays
Web SQL Database
WHAT WG HTML5
Web Sockets
Media Queries
RGBA & HSLA
Selektoren
Web Sockets
CSS3
Web Audio API
Trans-forms
Layout
Ani-mationen
Geo-Location
„HTML5“
Webtechnologien


11:22 AM
Battery Status
charging, battery level,
battery life remaining,
change events
Network Info
estimated bandwidth
ex. 3G, metered
Power Mgmt
screen on/off, brightness,
lock state
Photo by wolfgangfoto

we are going to address cost driven customers following the disruptive innovation model

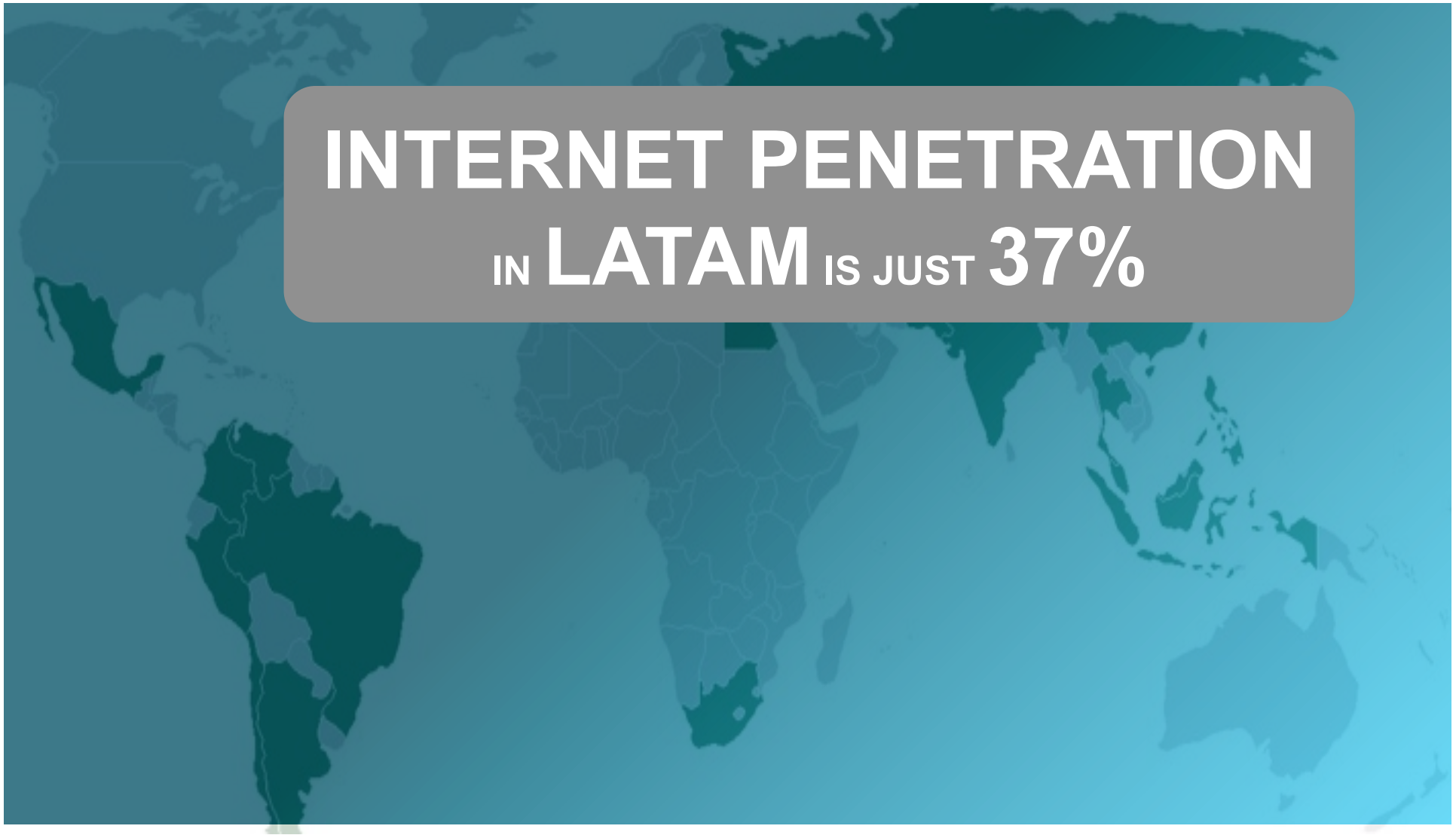
bravo!

Optimized to run hardware not suitable for latest smartphones OS

Bringing an affordable smartphone for the masses with a good UX and best mobile Web support

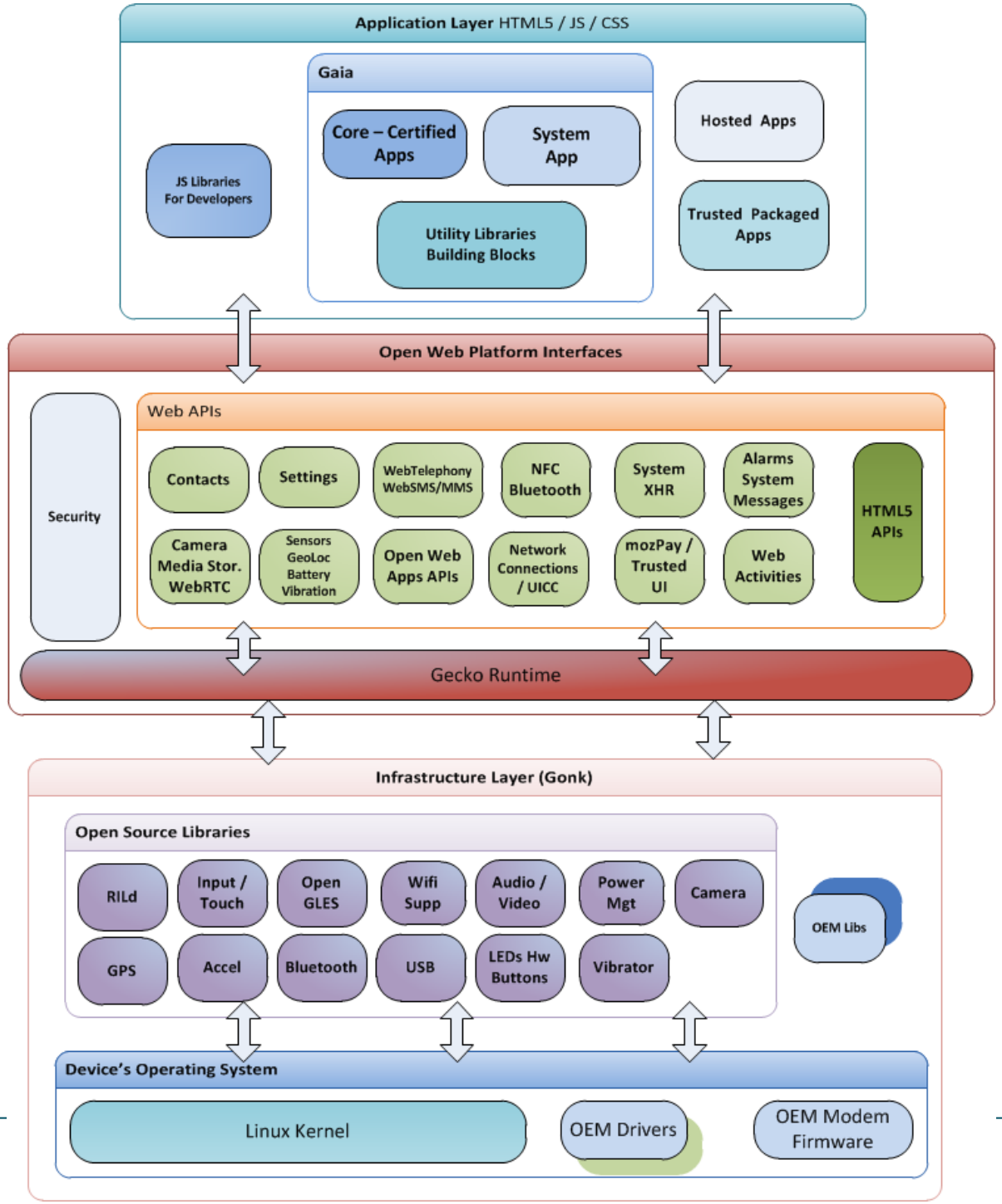
Better UX for the same price than alternative smartphone OS

Specially in emerging markets



Architecture

High-Level Architecure



Architecture : Layers

bravo!

- **Application Layer (Gaia)**
  - UI implementation based on building blocks and JS libraries
    - Jointly developed by Mozilla and TEF Digital

- **(Web) Platform Layer (Gecko)**
  - Runtime and middleware that provide the capabilities needed by the Application Layer
  - Developed by Mozilla (TEF Digital has strategically contributed)

- **Infrastructure Layer (Gonk)**
  - Lower-level OS services, libraries and other infrastructure based on Linux and other Open Source Software (OSS)
  - Device drivers or libraries provided by hardware vendors / OEM targeted to the specific hardware (i.e. not distributed as OSS)

- **Security**. Transversal layer for security & privacy

Application Layer

- ***Core - Certified Applications***
  - Applications which are the backbone of a contemporary mobile device
    - Dialer, Messaging, Contacts, Media, Gallery, Clock, etc.
  - Pre-installed on the device and certified by carrier / OEM

- Partner **Trusted** Applications
  - Complement core applications to incorporate compelling services demanded by users in target markets
  - Pre-installed and certified by OEM / carrier or the corresponding partner

- User Web Apps can be installed through
  - application store (market) → trusted packaged web apps (signed)
  - Web Page (HTML5 app cache) → untrusted hosted web apps
  - Search engine Ev.me (as web page bookmarks) → untrusted non-cached hosted web pages

Open Web Platform Interfaces

bravo!

- **Web APIs**
  - Expose native device functionality to the Web Platform
  - Specified by Mozilla through an open process
  - Developed by Mozilla and partners
  - Pushed forward in W3C for standardization

Gecko Runtime

- Gecko is the "application runtime" of Firefox OS.
- Implements the open standards for HTML, CSS, and JS
- Gecko is a **middleware** composed by (among others)
  - a networking stack
  - graphics stack (which delegates operations to the GPU when needed)
  - layout engine (for rendering HTML content)
  - virtual machine (for running Javascript code)
  - porting layers to the different platforms

Infrastructure Layer (Gonk)

bravo!

- Gonk implements a **hardware abstraction layer for mobile devices**
  - Actually it is a simple *Linux Distribution*
  - Gonk is a **porting target** of Gecko

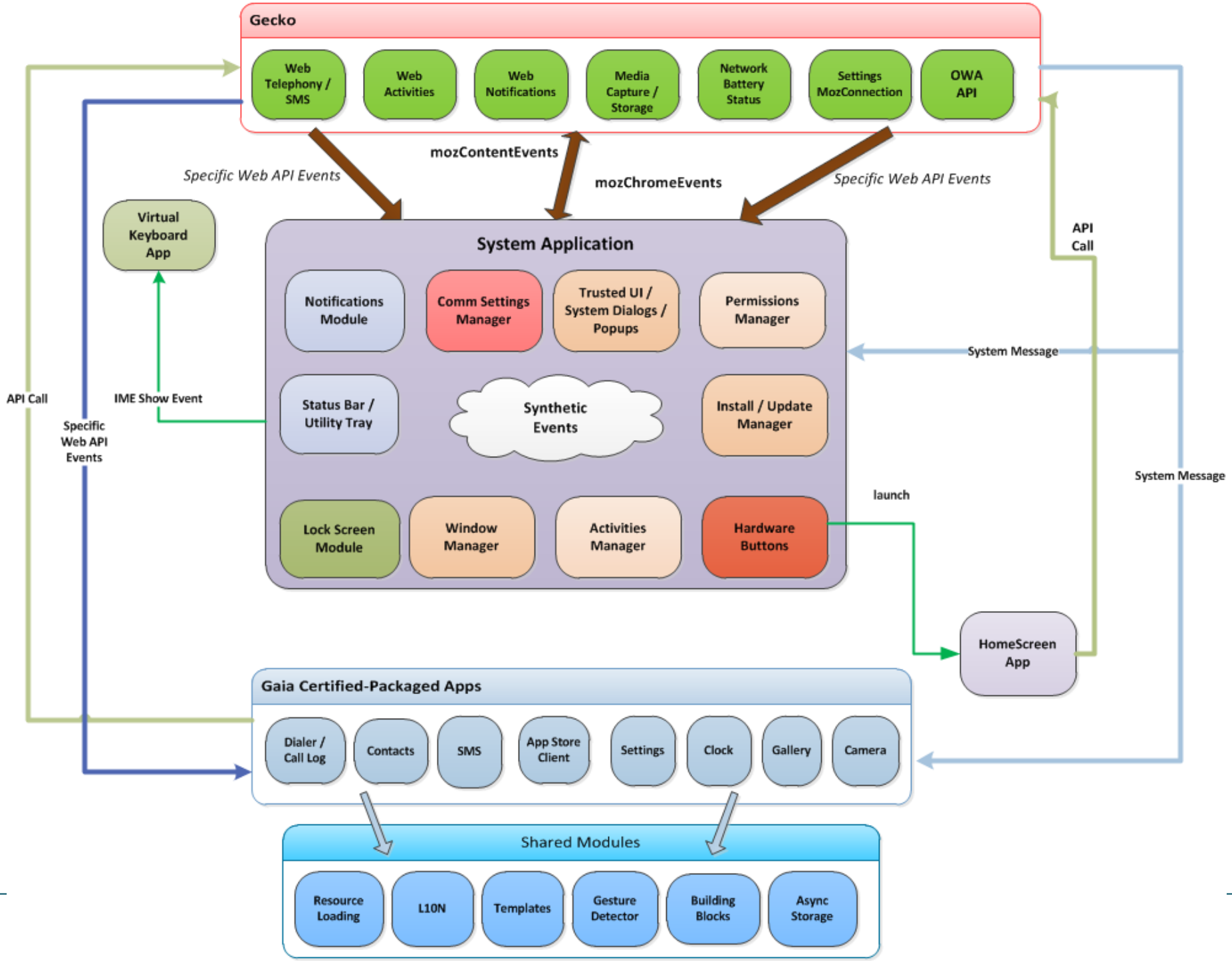
- Gonk is composed by
  - Libraries (both OSS and OEM-provided)
    - linux, libusb, bluez, etc.
    - GPS, camera and others from the Android OSS project.
  - Linux Kernel and user-space hardware abstraction layer (HAL)
  - Device Drivers provided by the OEM (including radio / modem)

- B2G has full control over Gonk and as a result …
  - Gecko has direct access to
    - the full telephony stack
    - display framebuffer
    - Other lower-level interfaces

Out of Process Architecture

- Two kind of processes in Firefox OS
  - One core process (b2g) aka “Chrome process”
    - runs with highly elevated privileges
    - controls the access to all resources and devices
    - modem, draws to the display frame buffer, and talks to GPS, cameras, and so forth.
    - Grants access to requested resources as per content process permissions (double check)
  - Many content process (at least one per web app running)
    - Communicates with b2g using inter-process communication (IPC)
    - Low-privileged sandboxed process (no access to OS services)
    - Request access to OS resources to the core process in accordance with the granted permissions
- It is up to the OS to kill content processes under low memory situations
  - Similar algorithm than Android

Application Layer


Gecko
Web Telephony / SMS
Web Activities
Web Notifications
Media Capture / Storage
Network Battery Status
Settings MozConnection
OWA API
mozContentEvents
mozChromeEvents
Specific Web API Events
Specific Web API Events
Virtual Keyboard App
System Application
Notifications Module
Comm Settings Manager
Trusted UI / System Dialogs / Popups
Permissions Manager
Status Bar / Utility Tray
Synthetic Events
Install / Update Manager
Lock Screen Module
Window Manager
Activities Manager
Hardware Buttons
API Call
IME Show Event
Specific Web API Events
API Call
System Message
System Message
launch
HomeScreen App
Gaia Certified-Packaged Apps
Dialer / Call Log
Contacts
SMS
App Store Client
Settings
Clock
Gallery
Camera
Shared Modules
Resource Loading
L10N
Templates
Gesture Detector
Building Blocks
Async Storage

App Layer – System App

- It is the main UA component implemented at application level
  - Avoids to hard code in Gecko those parts of the UI that are global to a specific device and UX
- Main responsibilities
  - Application Layer global initialization
    - Launching the Home Screen App
  - Keeping up to date the top Status Bar
  - Responding to device-related “global” events
    - Device On / Off / Sleep. Volume up – Down. Airplane mode.
    - Global Navigation.
    - Coverage / Wi-Fi / Network Manager events / Settings change events
    - Idle events (By loading the idle screen)
  - Responding to ‘IME’ events (virtual keyboard handling)

App Layer – System App

- More Responsibilities
  - Launching the Notifications bar at user request

  - Window management
    - Application management in mozapp iframes
    - Task switcher
    - Activity handling

  - System dialogs / Trusted UI / Popup Dialogs

  - Install / update management

  - Permissions Management (Prompting, etc.)

Application Management Principles

- **Application Management is based on Mozilla Open Web Apps**
  - It is being promoted in W3C with a view to standardizing
- **Each application runs on remote iframe with a Web origin**
  - including the “Home Screen”
  - HTML5 Visibility API allow apps to know if they are visible
- **App browsing contexts are isolated**
  - The number and stack of app browsing contexts is controlled by the OS
- **OS offer APIs to run applications in a new browsing context**
  - Web Activities (indirect API). For “non-system” apps
  - Direct API. For “Home Screen”. `navigator.mozApps.run(URI)`
  - System Messages (for instance when an alarm goes off)

Application Manifest

```
{ "name": "Gallery",
  "description": "Gaia Gallery",
  "type": "certified",
  "launch_path": "/index.html",
  "developer": {
    "name": "The Gaia Team",
    "url": "https://github.com/mozilla-b2g/gaia"
  },
  "fullscreen": true,
  "permissions": {
    "device-storage:pictures":{ "access": "readwrite" },
    "device-storage:videos":{ "access": "readwrite" },
    "settings":{ "access": "readonly" }
   },
icons": {
    "120": "/style/icons/Gallery.png",
    "60": "/style/icons/60/Gallery.png” }
```

Application Management API

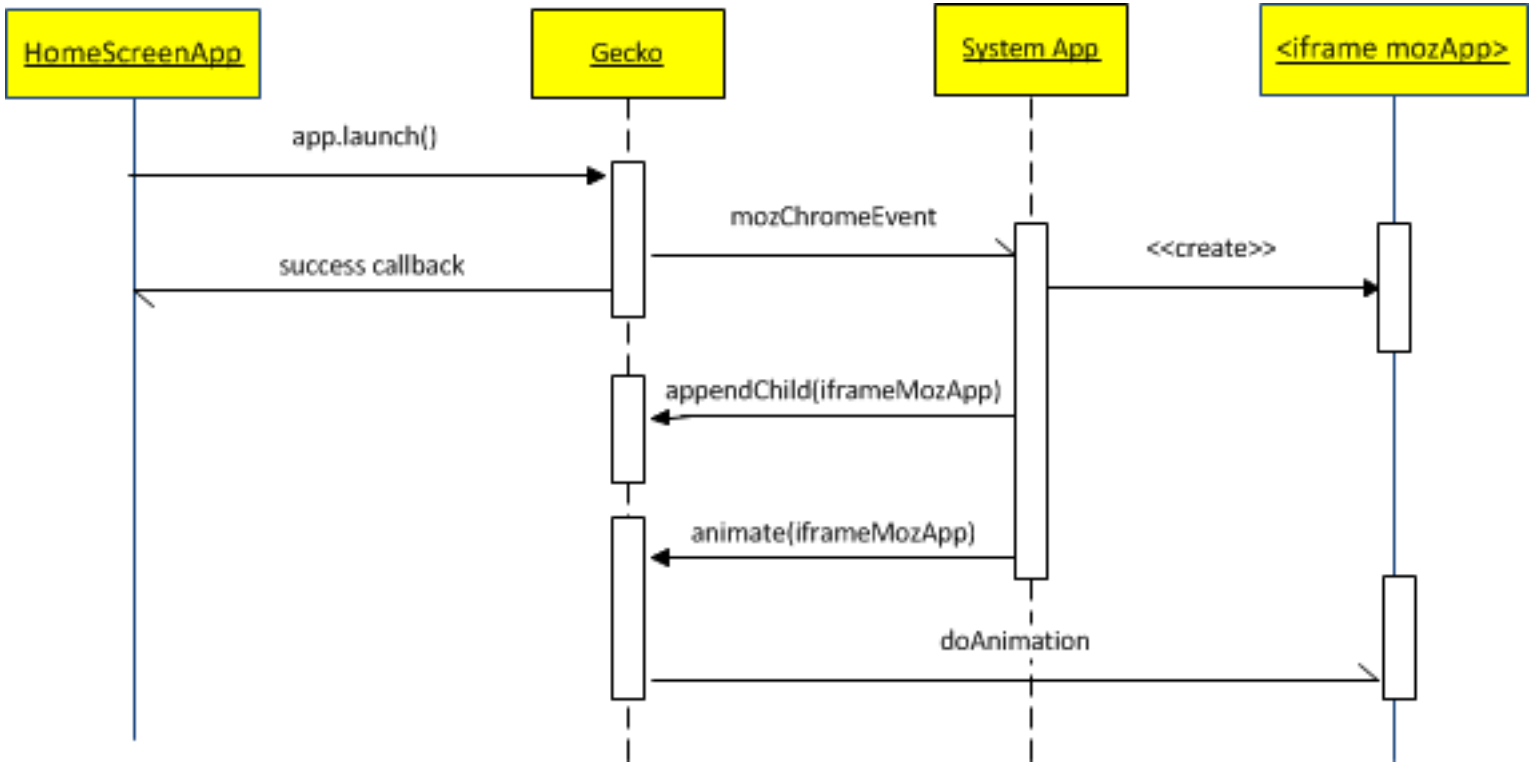
- `navigator.mozApps.install(uri, [package])`
  - URIs can be discovered through an application store or by the user
  - Applications will be installed by a trusted application
    - App Store Client, Browser App

- Running applications (direct API)
  - `navigator.mozApps.run(URI)` → "Home Screen" app

- Managing applications (App Manager)
  - `navigator.mozApps.uninstall, list, etc.`

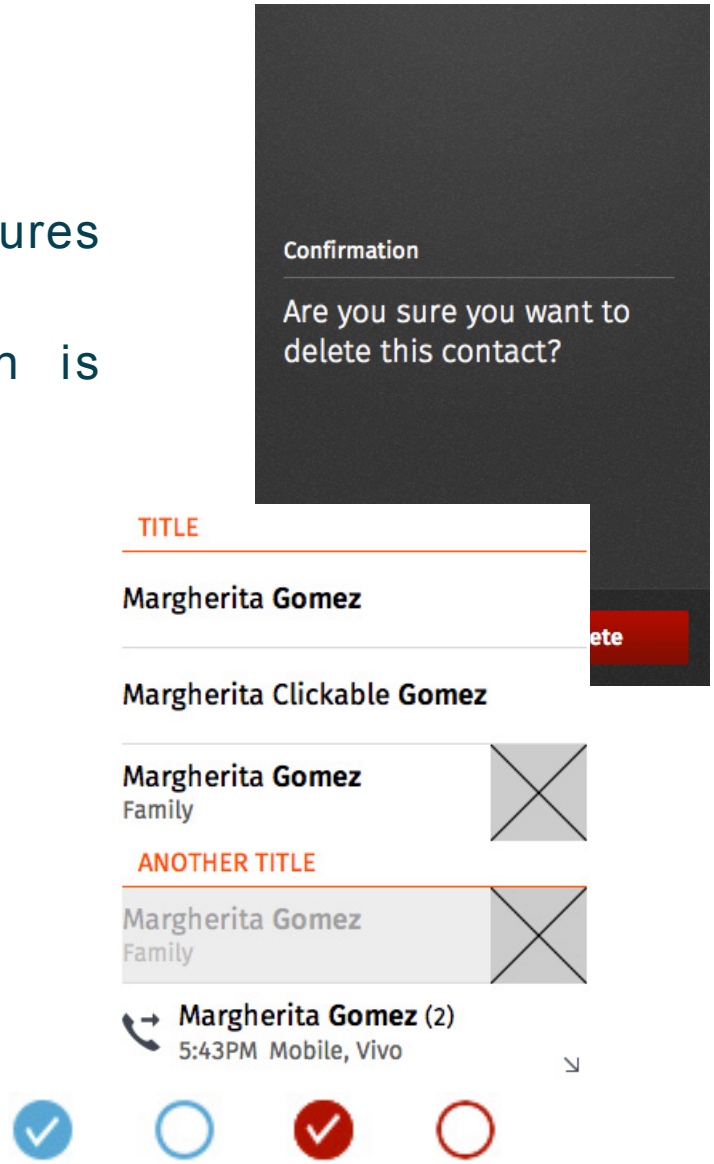
- An untrusted app may run on a HTML5 sandboxed iframe if the user wishes

Launching an app



Application UI

- **Building Blocks (shared/style)**
  - A library of common UI structures implemented using HTML + CSS
  - Enable that UI implementation is consistent across applications
  - Semantic and accessible mark-up
    - Buttons
    - Headers
    - Lists
    - Switches
    - Menus & Dialogs
    - ….
  - http://buildingfirefoxos.com



Security

Security Model

- The origin of a Web App determines what Web APIs is entitled to use
  - Applications must request on their manifest individual permissions for each privileged API they want to use.

- From an application level, security is determined by
  - a permissions matrix.
    - On one axis the application type
    - On the other axis the specific Web APIs.
    - On each cell the maximum permission level for that particular API and application type
  - Content Security Policy (CSP)

Security – Application Types

- **Certified**: The application is provided by the operator/device builder. The only way to distribute this kind of application is by pre-installation on the phone or by device updates.

- **Privileged**: This application is provided by a trusted App Store. On V1 the only trusted App Store is the Mozilla Marketplace. The application can be downloaded and installed from the store. The package must be signed.

- **Web**: Every other application out there. Web applications can be
  - installed (they're packages the same as Certified an Privileged apps but without the privileges)
  - hosted (so they're a cached view of a web page).

Low level protection

- One root process (B2G) that has access to the HAL. Only a small subset of certified apps can run in this process.

- N non privileged processes (one per app) called content processes
  - Non privileged processes request from the parent process access to privileged resources via IPC
  - The parent process knows what app is in each content process, and the permissions that app has
  - If a suspicious/invalid request arrives from a child process, the child process is assumed to be compromised and killed.
  - Each process runs as a different underlying OS user, similar as what happens on Android.
  - But there's not a one-on-one assignation between users and applications.

Telefónica