CS5331: Web Security

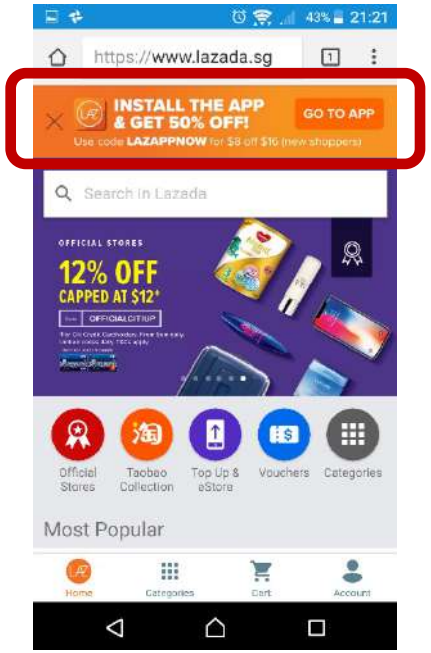
Lecture 9: Mobile Security

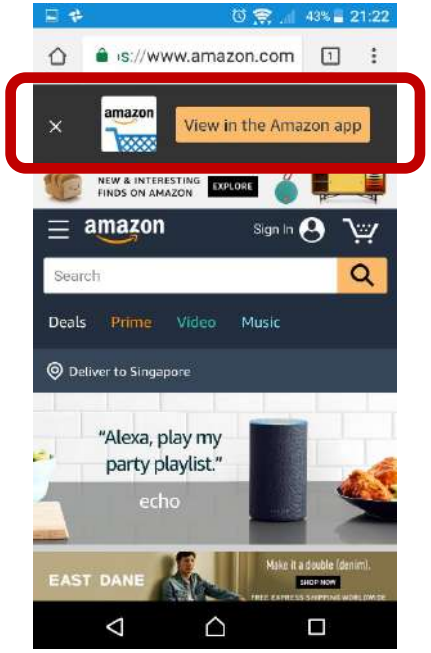
Modern Mobile OSs

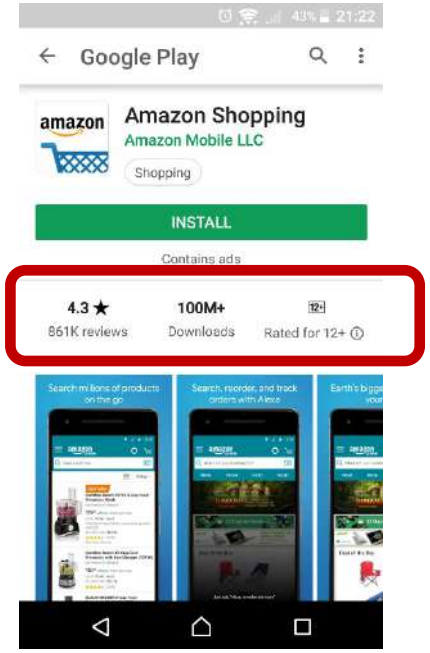
- Worldwide smartphone-subscription projection: 1.9 billion in 2013 to 14.91 billion devices in 2021 (Ericsson Mobility Report, Nov, 2013)
- Dominated by Android (Google) vs iOS (Apple)
- Android
  - OS kernel: Linux
  - Open ecosystem: Android Open Source Project (AOSP), various manufacturers, possible customization
- iOS
  - OS kernel: Darwin (shared with OS X)
  - Closed ecosystem

Web and Mobile Apps

- Both browser and mobile apps as “universal” clients
- *Appification*: mobile apps especially for commonly-used, feature-rich web sites
  (The Appification Of Everything Will Transform The World's 360 Million Web Sites)



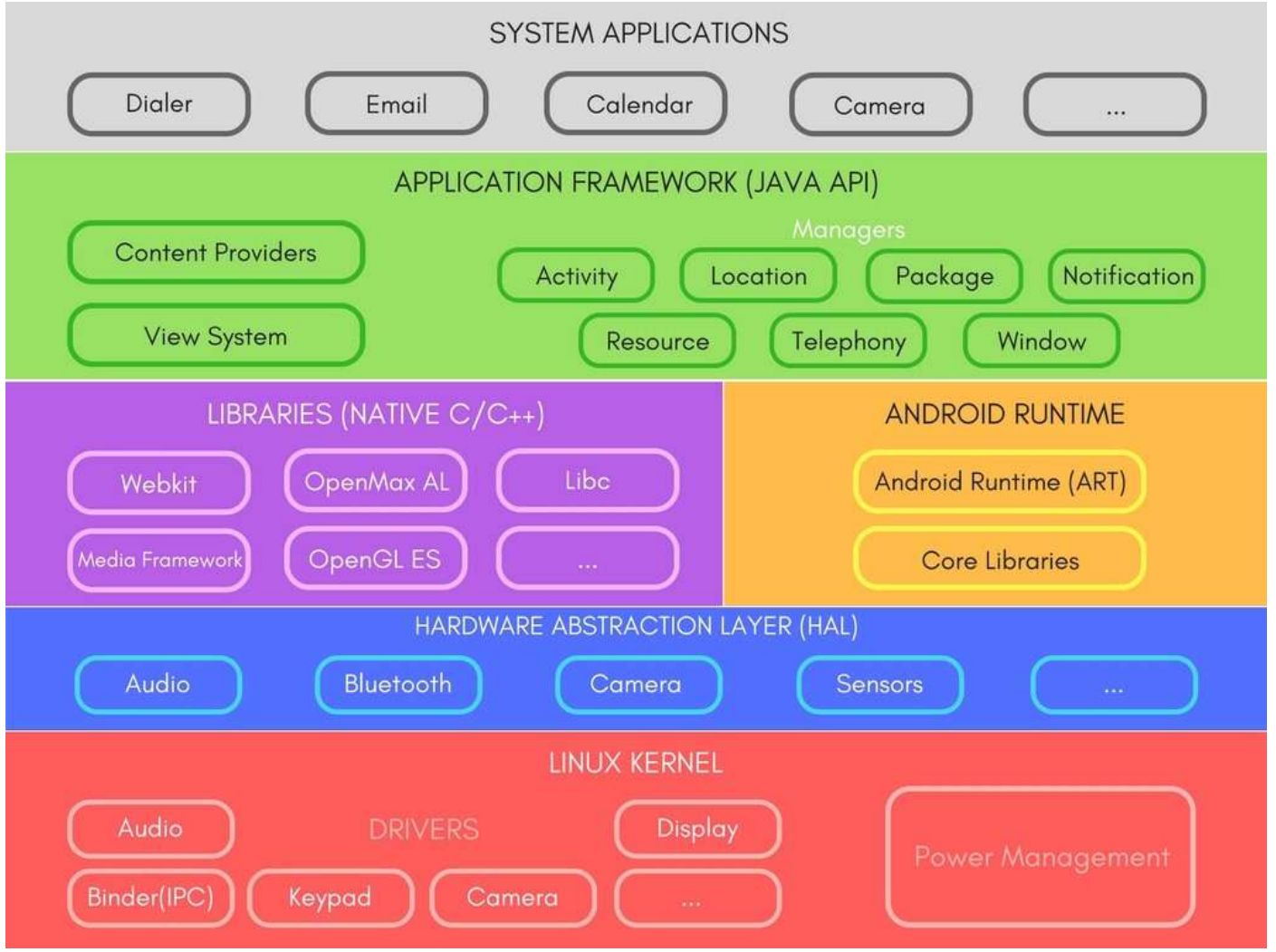




Android Mobile OS

- Android holds the biggest market share (Wikipedia):
  - In 2014: ~1.9 billion devices in use
  - As of February 2017: Google Play store has over 2.7 million Android apps published
  - As of May 2016: apps have been downloaded more than 65 billion times
- Android devices have also become prominent targets of malware/security attacks too:
  - Widely reported by various security companies
  - Cautioned by the U.S. Dept. of Homeland Security in 2013
  - Recent WikiLeaks says that CIA used Android exploits

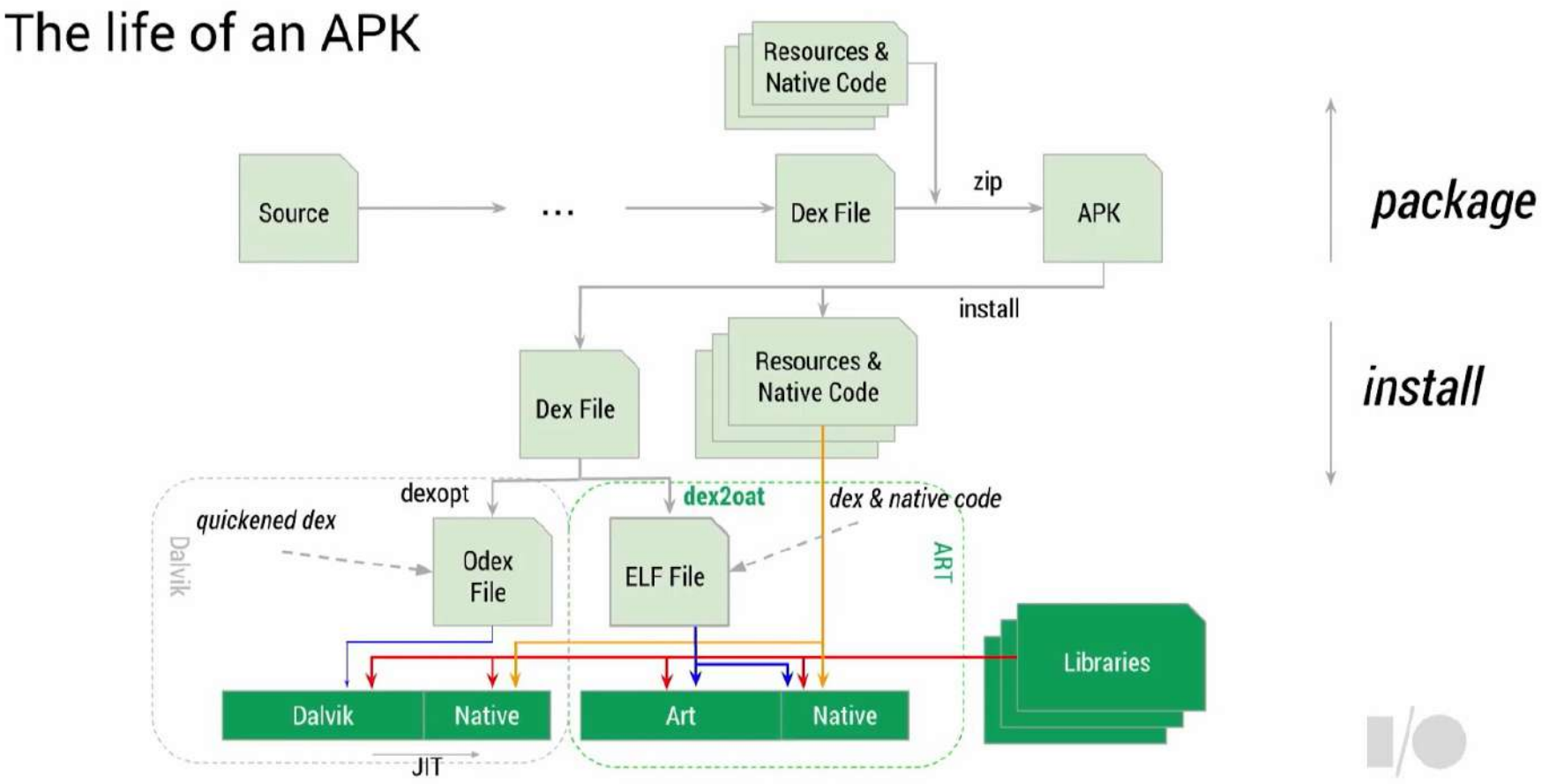
Android Platform Architecture



Android Apps

- Apps mostly written in Java (recently: Kotlin):
  - Java bytecode (.class) translated into Dalvik bytecode (.dex)
  - A possibility of native libraries, which are invoked via Java Native Interface (JNI)
  - All files are packaged and signed as a single APK file
- Two Android runtime systems:
  - Dalvik VM: prior to v5.0, uses just-in-time (JIT) compilation
  - ART (Android RunTime): on recent Android versions, uses ahead-of-time (AOT) compilation
- Dalvik/ART:
  - Register based VMs (JVM is stack based)
  - Differ from JDK – no Java security manager

App Overview: Building & Installation



Android App: Sample Java Code

```
package com.example.helloworld;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

public class MainActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
  }

  …

}
```

Android App: Dalvik ByteCode (Smali)

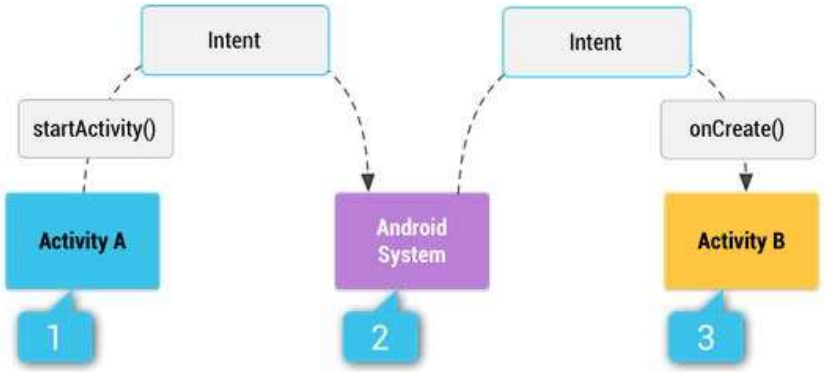
```
.class public Lcom/example/helloworld/MainActivity;
.super Landroid/app/Activity;
.source "MainActivity.java“

# virtual methods
.method protected onCreate(Landroid/os/Bundle;)V
  .locals 1
  .param p1, "savedInstanceState"  # Landroid/os/Bundle;

  .prologue
  .line 12
  invoke-super {p0, p1}, Landroid/app/Activity;
       ->onCreate(Landroid/os/Bundle;)V
  .line 13
  const/high16 v0, 0x7f030000
  invoke-virtual {p0, v0}, Lcom/example/helloworld/MainActivity;
       ->setContentView(I)V
  .line 14
  return-void
.end method
```

Android App Extra Details

- An app internally consists of multiple app components
- Four types of app components:
  - Activity
  - Service
  - Broadcast receiver
  - Content provider
- Android Framework (OS) invokes different *lifecycle methods* of an app component:
  - e.g. the activity lifecycle shown in the next slide
- Unlike Java programs, Android app has *multiple* entry points:
  - Make Android apps harder to analyse than Java programs!

Inter Component Communication

- Performed using *intent*: a messaging object containing the destination component's address or action string, and possibly data



Source: developer.android.com

- Interacting-app components:
  - *Unicast* intent: between two app components
  - *Broadcast* intent: a sender to multiple interested broadcast receivers, including system-based broadcasts for informing system events

Android APK File

- App packaged in a single APK file:
  - JAR (zip) file format generated by jarsigner tool
  - Signed – can be self-signed
    - So users may not necessarily trust the signature
    - Useful more for app updates & sharing
  - Manifest file
  - Resources
  - Dex files (for Dalvik/ART VM)
  - …

Android Manifest

- An app's XML file component that declares:
  - Java package for the app, which serves as a unique identifier for the app
  - App components
  - Required permissions:
    - Permissions that the app must have in order to access protected parts of the API and interact with other app
    - Permissions that others are required to have in order to interact with the app's components
  - Minimum level of the Android API required by the app
  - …
- Read https://developer.android.com/guide/topics/manifest/manifest-intro.html

Android Sample Manifest

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<manifest xmlns:android=http://schemas.android.com/apk/res/android
  package="com.example.browserapp" platformBuildVersionCode="21"
  platformBuildVersionName="5.0.1-1624448">

  <uses-permission android:name="android.permission.INTERNET"/>

  <application android:allowBackup="true" android:debuggable="true"
android:icon="@drawable/ic_launcher" android:label="@string/app_name"
android:theme="@style/AppTheme">

    <activity android:label="@string/app_name"
android:name=".BrowserActivity">|
       <intent-filter>
          <action android:name="android.intent.action.MAIN"/>
          <category android:name="android.intent.category.LAUNCHER"/>
       </intent-filter>

    </aZctivity>

  </application>

</manifest>
```

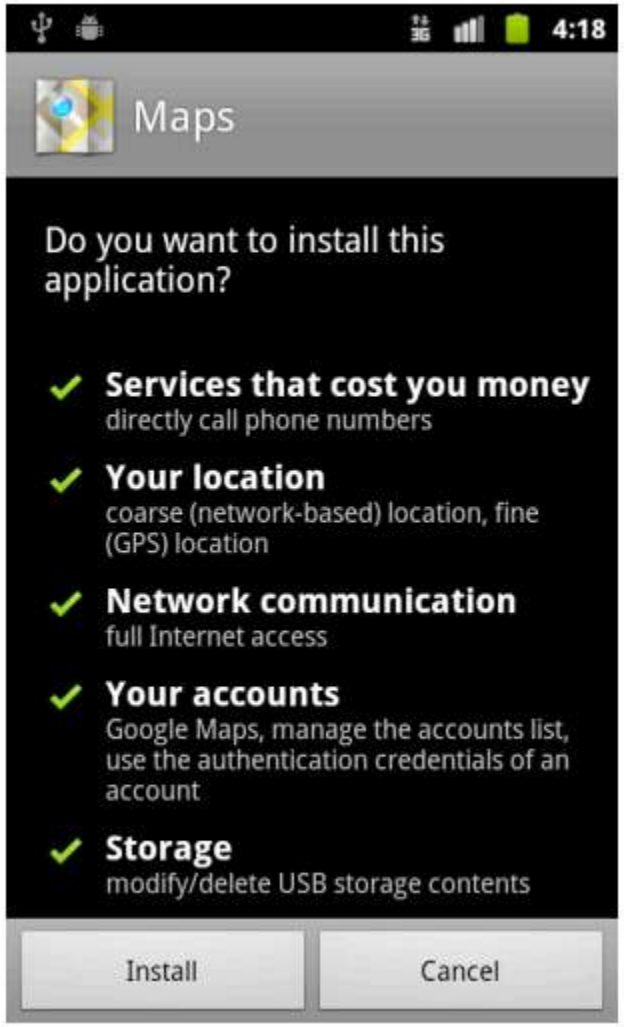
Android Security Mechanisms

- Linux kernel security:
  - Address Space Layout Randomization (ASLR) to randomize addresses on stack
  - Hardware-based No eXecute (NX) to prevent code execution on stack/heap
  - StackGuard derivative
  - Security Enhanced (SE) Android: some form of MAC over all processes, including root-privileged processes
- How about Android isolation mechanism?
  - Recall two types of isolations:
    - Isolate apps from each other
    - Regulate access to OS resources

Android Security Mechanisms

- App sandboxing:
  - Isolate apps from each other
  - Each app runs with unique user ID (UID) + group ID (GID)
  - Sandboxed to its IDs – doesn't affect other apps/system
- Permission system:
  - Regulate access to OS resources
  - Permissions needed to access resources/sensitive APIs (e.g. telephone function, network access, …)
  - App request permissions:
    - At install time – all/nothing: prior to v6.0
    - At runtime: since Android 6.0 (API level 23)

Permissions: Install-Time Request Approval



Top 10 Targeted Permissions

Commonly used permissions in malware: (discuss why?)

1. SEND_SMS, RECEIVE_SMS
2. SYSTEM_ALERT_WINDOW
3. READ_HISTORY_BOOKMARKS, WRITE_HISTORY_BOOKMARKS
4. READ_CONTACTS, WRITE_CONTACTS, READ_CALENDAR, WRITE_CALENDAR
5. CALL_PHONE
6. READ_LOGS
7. ACCESS_FINE_LOCATION
8. GET_TASKS
9. RECEIVE_BOOT_COMPLETED
10. CHANGE_WIFI_STATE

Android Security Mechanism

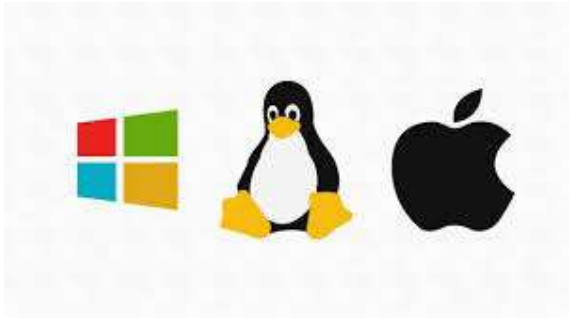
- App signing:
  - All Android apps need to be signed before installation
  - Self-signed certificates for 3-rd party apps are allowed! *Why?*
  - Verifies whether different apps *come from* the same developer
  - Updates are allowed if the same private key are used
  - Also used for booting process and system update
- App component encapsulation:
  restricts access to an app component

Android Security Mechanism

- Google Play store scans and approves apps:
  - Attacks on the scanner (Bouncer) were possible, e.g. by Oberheide & Miller, Summercon, 2012
- Android devices run an "app verification service" upon app installation:
  - Introduced in Android v4.2
  - Experiments showed that it was ineffective against malware (https://www.csc2.ncsu.edu/faculty/xjiang4/appverify/)
- For the details of Android security mechanisms: read http://source.android.com/tech/security/index.html

The Three OSes View

Web Browsers



Desktop OSes

Mobile Oses

Difference #1: Notion of Authorities

Web Browser	Unix	Android
- Web origin - PKI entities (for HTTPS) - Web extensions are different! - **No notions of "users"**	- UID: root vs non-root users, Groups (GIDs) - **No notion of apps**	- App (from the same developer) -**No notions of "users"** -But internally, UID: root vs non-root users, Groups (GIDs)

Difference #1(b): Notion of Sub-Authorities

Web Browser

- No.
- But see proposed plans for sub-origins, e.g. per-page sub-origins

Unix

- Yes.
- **E.g. Process-based isolation**

Android

- Yes.
- **E.g. Process-based isolation**

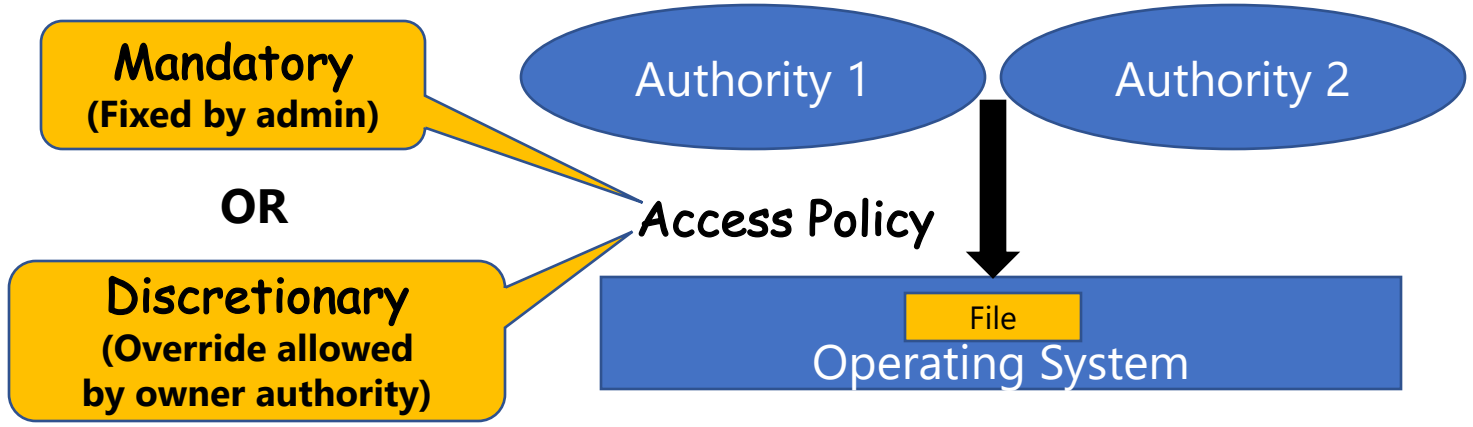
Difference #2: Authority Import (App Installation)

Web Browser	Unix	Android
- None (just click; or sometimes without a click, e.g. embedded pages)	- Via Package Managers (recommended) - Download from web (Nowadays with digital-signature verification feature)	**- Via app stores (recommended)** **- Side-loading (e.g. via ADB)**

Difference #3: Isolation Mechanisms

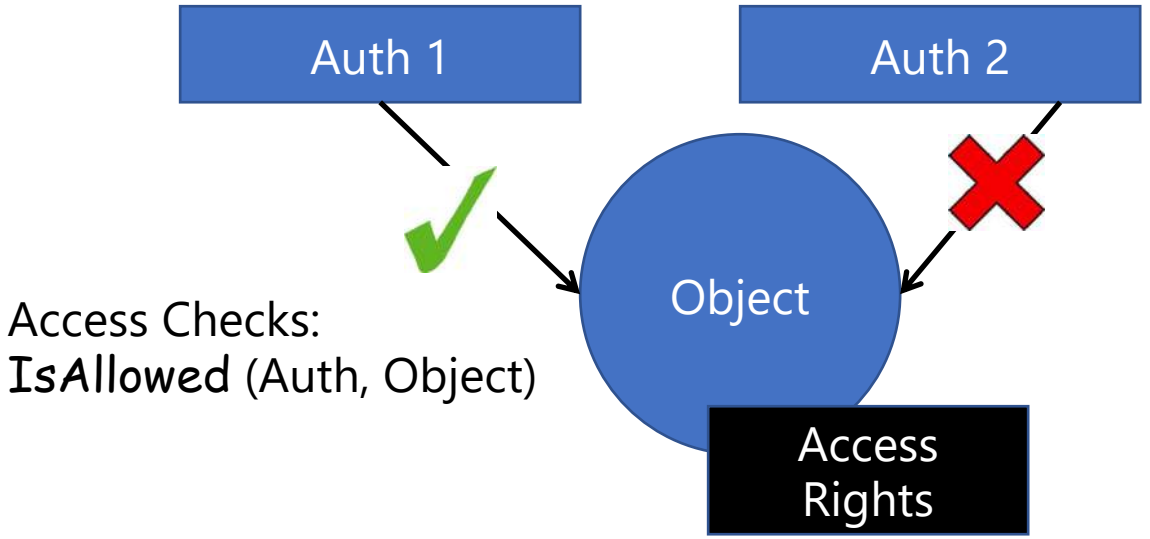
Web Browser	Unix	Android
- Sandbox baked in iframe / window (see the next slide)	- User domains - Within a domain: Process isolation - No sandboxing (default): e.g. on file access	- App domains - Within a domain: Process isolation -Sandboxing: e.g. on file access

Difference #4: Permission Delegation



Web Browser	Unix	Android
- **Mandatory Access Ctrl.** (e.g. Same-origin policy) - Some exceptions: - cross-origin resource	- **Discretionary Access Ctrl.** (e.g. files) - Defaults deny - But users can override	- **Mandatory Access Ctrl** (e.g. sandboxed apps, SEAndroid) - But users can override via *rooting*!

Difference #5: Authority Delegation



Access Checks:
IsAllowed (Auth, Object)

Web Browser	Unix	Android
- No explicit mechanism	- Yes, grant permissions - (e.g. groups) - (e.g sudo), setuid()	- No mechanism

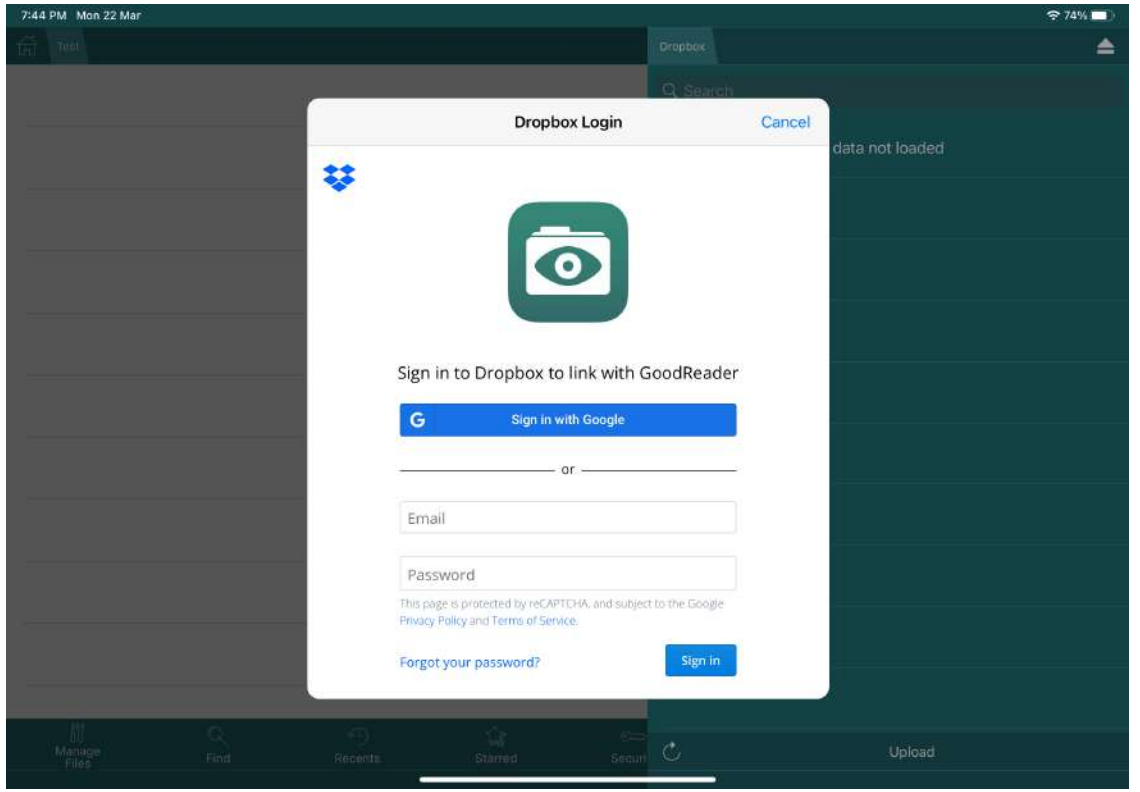
Blending Web and Mobile/System

Mobile Device as a Web Client

- Web authorization is used when connecting another web service, such as Dropbox
  - OAuth tokens are generated for follow up connections.
- In the address bar, you can check certificate. Can you do it in mobile?
  - Check the following examples.

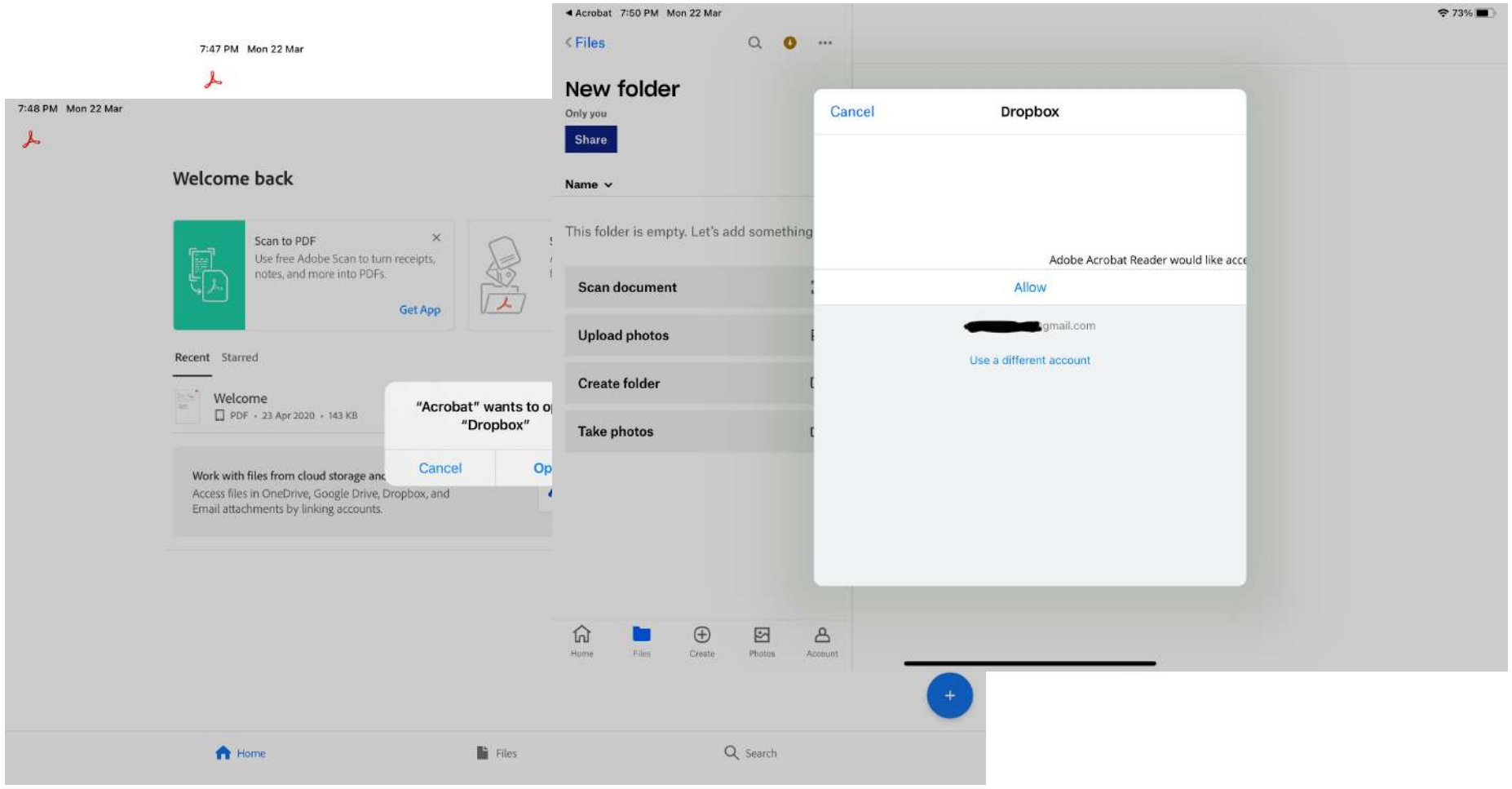
Connecting to Web Service

- GoodReader connects to Dropbox



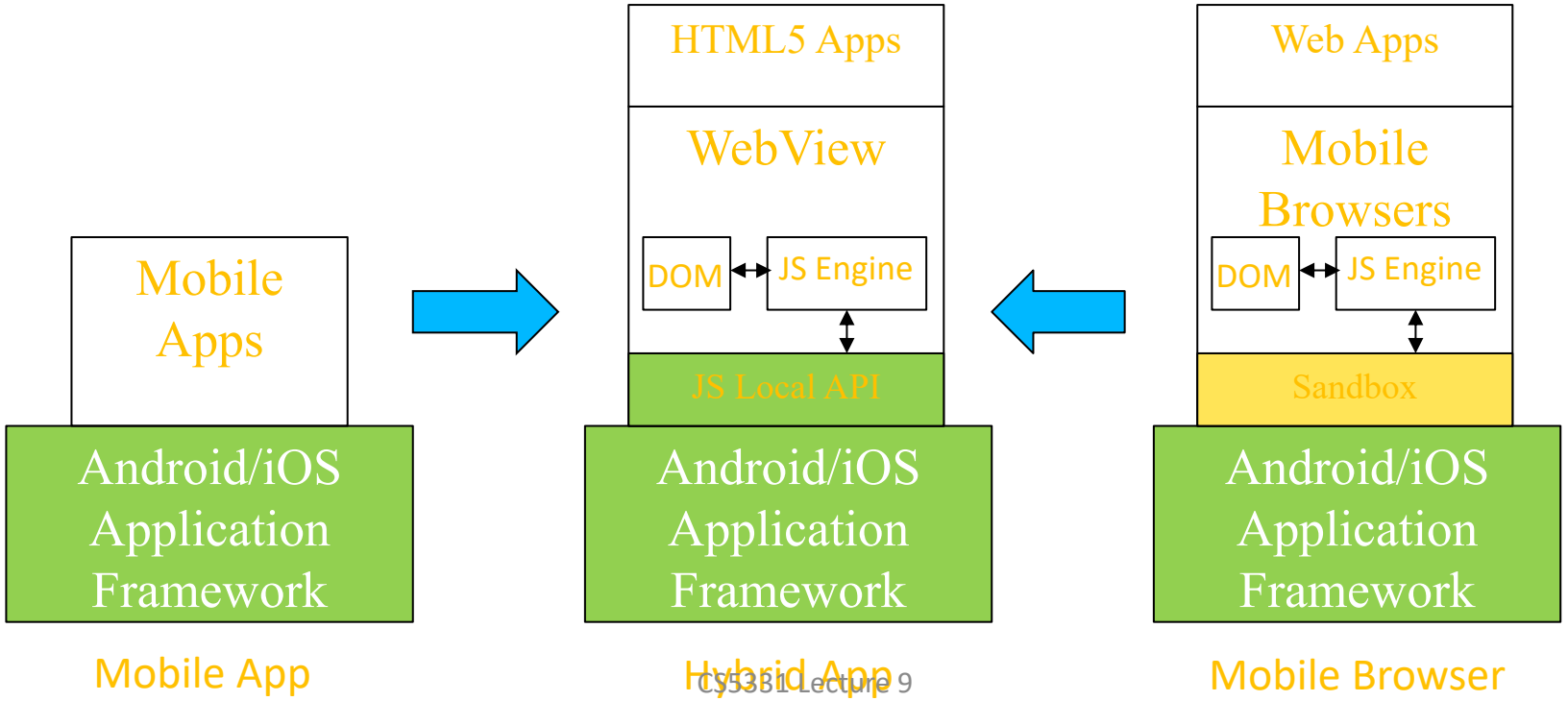
Authentic Connection Request from the App

- Acrobat Reader connects to Dropbox



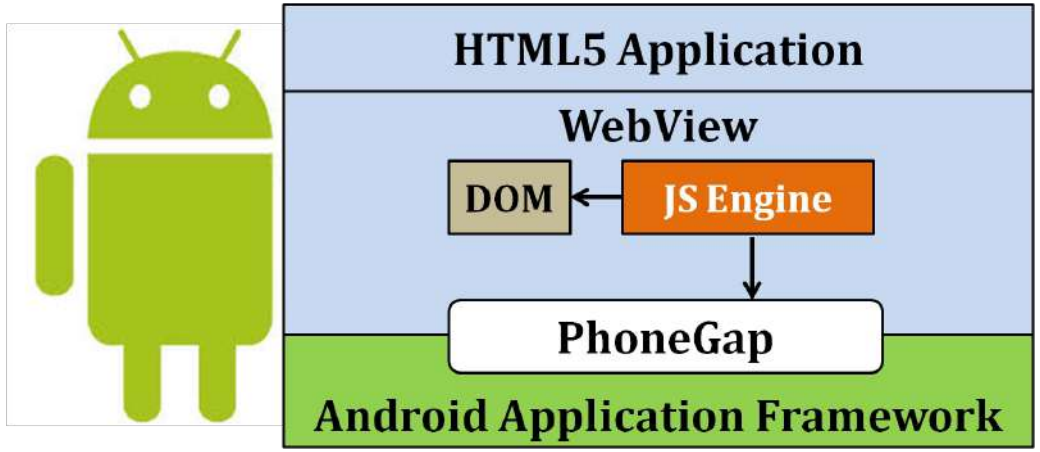
HTML5-based Mobile Apps

- Web applications have excellent cross-platform support
  - Using HTML/JavaScript to access system resources

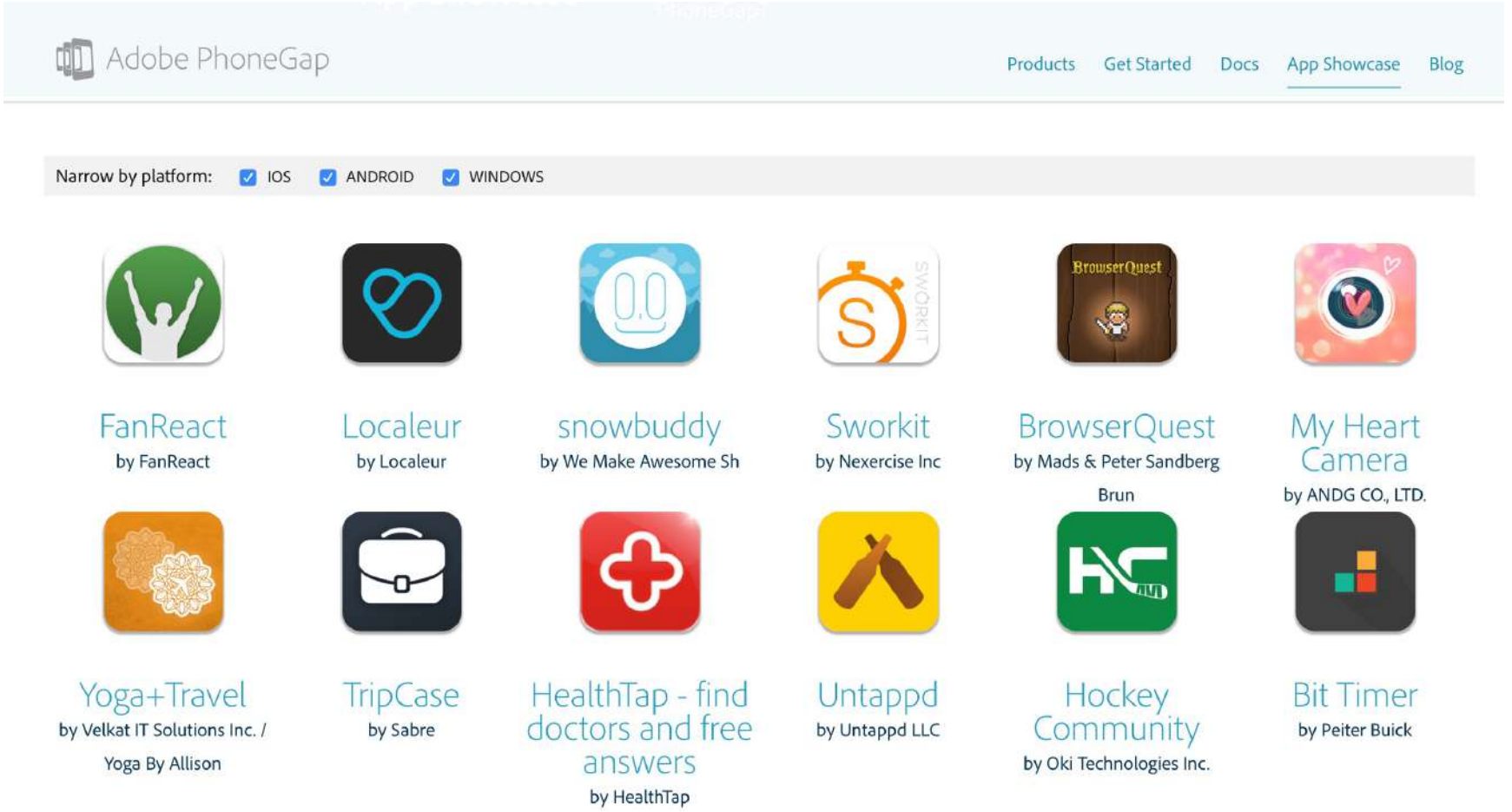


Hybrid Apps

- A new mobile application development framework based on web technology
  - HTML5, JavaScript, CSS
  - Plugin into WebView to give JavaScript privileges for local access.



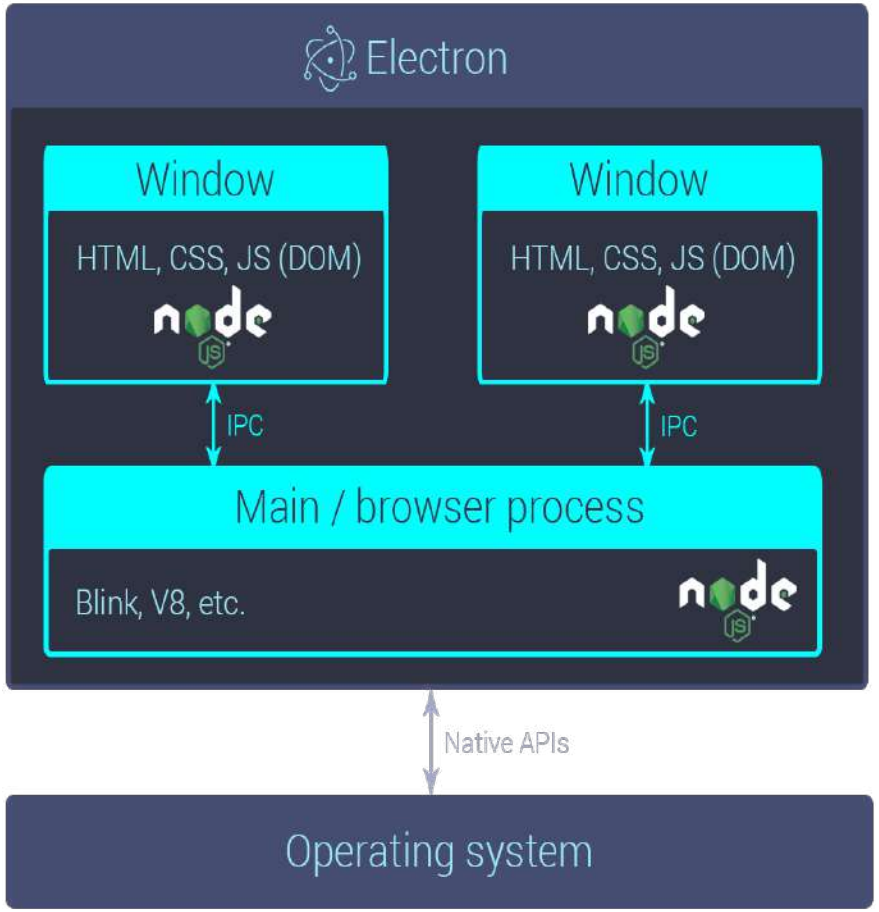
PhoneGap Framework



Electron Framework

- Cross-platform desktop applications from JavaScript, HTML, and CSS
  - https://www.electronjs.org/

DISCORD openfin slack
Microsoft ATLASSIAN
Abstract RingCentral aircall



Security Problems in Hybrid Apps

- What security problems in the web are carried over to Hybrid apps?
- What are the new security consequences?
  - We will discuss this research paper:

Code Injection Attacks on HTML5-based Mobile Apps: Characterization, Detection and Mitigation.
Xing Jin, Xunchao Hu, Kailiang Ying, Wenliang Du, Heng Yin and Gautam Nagesh Peri.
In ACM Conference on Computer and Communications Security (CCS), 2014.