Alma Mater Studiorum – University of Bologna
CdS Laurea Magistrale (MSc) in
Computer Science Engineering

Mobile Systems M course (8 ECTS)
II Term – Academic Year 2022/2023

05 – Mobile Middleware Principles and Android

Paolo Bellavista
paolo.bellavista@unibo.it

Mobile Middleware

Managing many aspects (in particular mobility), not strictly application-specific, is a complex problem. For instance, mobility of:

- Nodes
- Networks
- ***Transport connections***
- ***Session***
- ***Objects (passive, active)***
- ***Services***
- ***Users***

Several solutions are needed, at multiple levels (link, network, transportation, application) and ***cross-layer***

Mobile middleware solutions to indicate all support solutions typically positioned from OSI level 4 to upper

Middleware

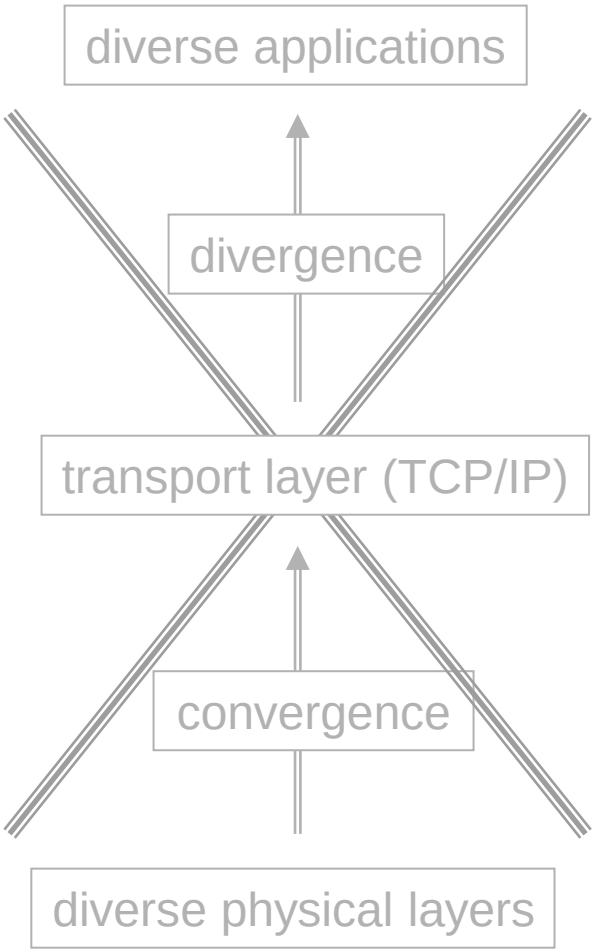
"Pop" and largely employed term, sometimes also in a partially imprecise way

- A possible definition:
  - "A set of ***service elements above the operating system and the communications stack***"
- An alternative definition:
  - "Software that provides a ***programming model above the basic building blocks of processes and message passing***" (Colouris, Dollimore, Kindberg, 2001)

- We define it as ***support software stack, typically cross-layer and application-agnostic***, which targets issues and challenges at OSI levels >= 4

Why the Need of Middleware?

- Development of distributed apps as ***complex and time-consuming process***
  - Any developer has to ***code from scratch his/her protocols for naming services, transactions***, ...?
  - How to manage the unavoidable ***heterogeneity of execution ens***?
- Need of middleware
  - To reduce development time and to favour rapid prototyping
  - To simplify application development and thus reducing the costs
  - To support heterogeneous envs., by partially masking differences at hardware/OS/app levels



hourglass model

Mobile Middleware

Middleware is traditionally designed, implemented, and optimized for ***fixed network hosts***

- Large bandwidth, low latency, reliable communications
- Persistent storage, good computing capabilities, no constraint associated with energy consumption
- No mobility

Mobile systems call for ***novel middleware solutions***

- Existing middleware is not suitable for resource-limited devices and sometimes exhibits limited scalability

Mobile middleware goals:

- ***fault-tolerance, adaptiveness, heterogeneity support, scalability, resource sharing***

Mobile Middleware

Mobile middleware

- ***Context dynamically changes***
- Need of ***decoupling in space and in time***
  - ***Asynchronous events, spaces for data sharing***
- Basic approach in many cases of wireless computing
  - ***Adoption of proxies***
- In addition, provisioning of ***some level/degree of visibility*** of running conditions to lower layers (sometimes *reflective middleware*)
  - Transparency to location in RPC/RMI
  - In mobile envs, partial visibility of ***location change, modifications in the QoS levels currently available***, ...

Not only mobile... Internet Principles

- **End-to-End Principle**
  - In its origin expression, referred to the opportunity to ***maintain state and intelligence only on network borders (edges)***
  - ***Internet*** connects these edges ***without keeping state, in order to achieve maximum efficiency and simplicity***
  - Today in real scenarios: firewalls, Network Address Traversal, caches for Web content. These are relevant modifications to this general principle
  - Towards Trust-to-Trust principle (application logics always run on trusted nodes?)
- **Robustness Principle**
  - *"Be conservative in what you do, be liberal in what you accept from others"* (attributed to Jon Postel, RFC 793 TCP)
  - Internet stack developers should ***be careful in respecting what specified in existing RFCs*** when they implement their functions, but be ***ready to process less rigidly and to accept non-compliant input*** from others (even non-compliant with existing RFCs)

Not only mobile... Web Principles

Web principles follow the guidelines of the ones at the basis of the underlying TCP/IP stack

- ***Simplicity, modularity, decentralization, and robustness***

In particular, about access, data representation, and data transformation for Web resources:

- Principle of application of ***least powerful language*** to implement any feature (maximum accessibility and URL mechanism)

Central and recognized problem is ***state maintenance for stateful interactions***. For instance, *Representational State Transfer (REST- see later), based on URIs, HTTP, XML*

- ***Client-server, stateless, cacheable, layered***
- ***Uniform interface to transfer state between client and resource*** (limited set of well-defined operations, possibly using code on demand)

Not only mobile... SOA Principles

Service Oriented Architecture (SOA) as the sw architecture where ***features and functions*** are ***structured*** around ***business processes*** and implemented via ***interoperable loosely-coupled services. Strongly based on message-oriented communications***

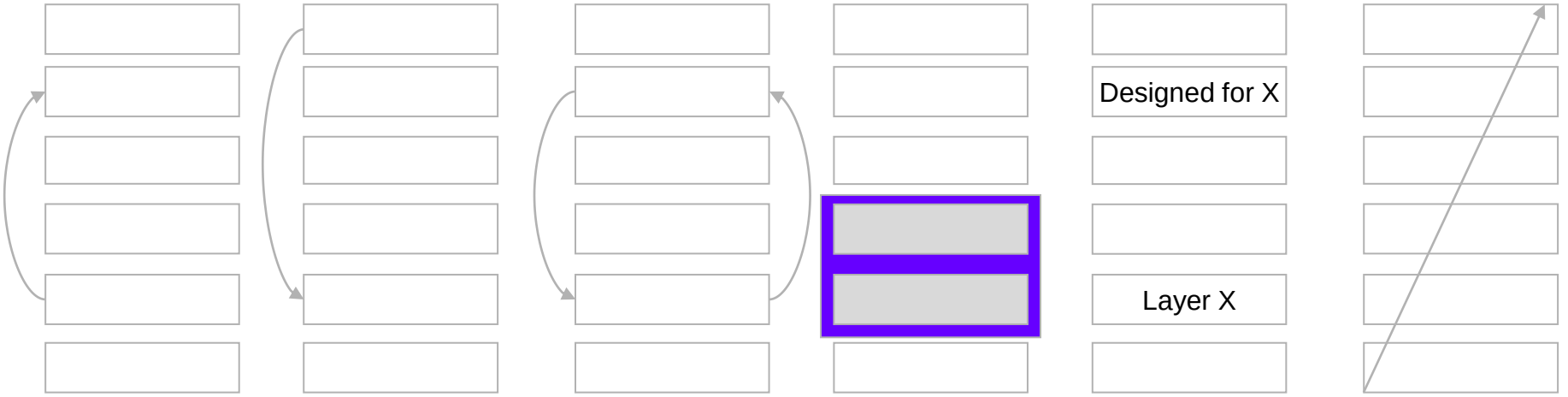
- ***Re-usability, granularity, modularity, possibility of dynamic composition, component-based organization, interoperability***
- Compliance with standards
- Identification and classification in terms of categories of services, provisioning and delivery, monitoring and tracking

Specific Principles for Mobile: Cross-Layering

Different types of interaction for different ways of cross-layering in a protocol stack

- ***Upward info flow***, with info propagated from lower to higher layers, in opposite to classical principles for layered architectures
- ***Downward info flow***, with info propagated from higher to lower layers, typically to configure lower-layer parameters and settings
- ***Back-and-forth info flow***, with info propagated in both directions
- ***Merging of adjacent layers***, allows the combination of different adjacent layers into a single super-layer
- ***Coupling*** with no addition of new interfaces. Two or more layers are ***coupled at design time*** in a finalized way, ***with no possibility of maintaining independency/abstraction from lower layer***
- ***Vertical calibration*** between layers. Usually to achieve layers-joint optimization of the configuration of parameters (***joint tuning***) and to obtain better overall performance

Cross-Layering Principle



Upward information flow

Downward information flow

Back and forth flow

Merging of adjacent layers

Design coupling

Vertical coupling

Not only mobile... Architectural Patterns

Several architectural patterns of general applicability and usage, not only in distributed systems:

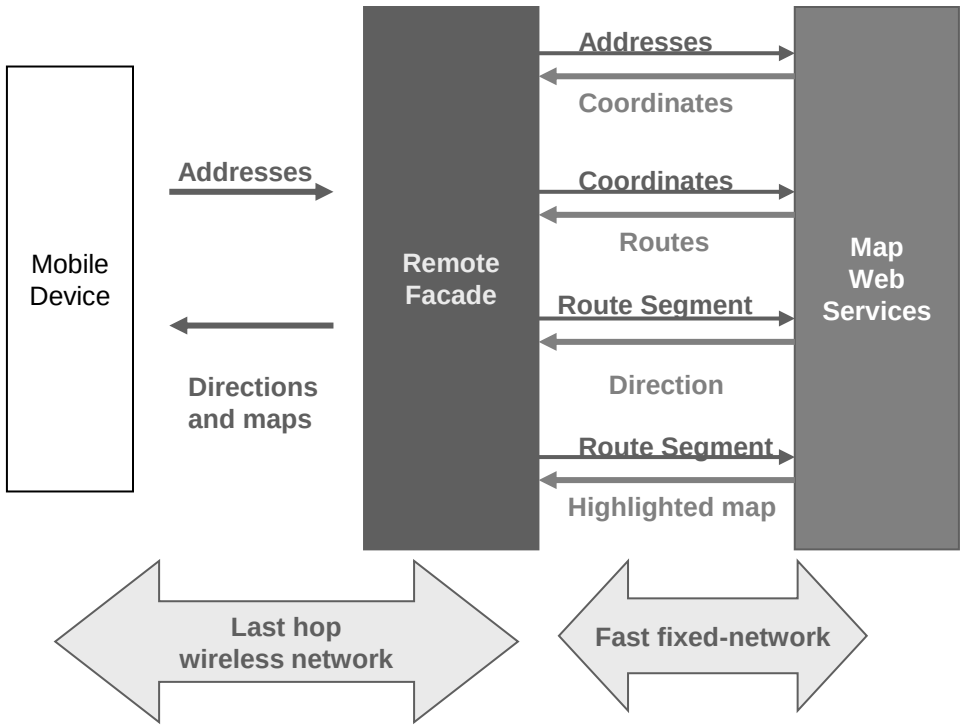
- ***Level-based***. Multi-layer sw architecture with ***different responsibilities "rigidly" allocated to different layers***
- ***Client-Server***. Most frequent pattern in distributed computing: clients use resources and services offered by server
- ***Peer-to-peer***. Any node can dynamically play ***the role of either client or server***; functionality could be more or less symmetric
- ***Pipeline*** (or pipe&filter). Pipeline as chain of processing elements aligned in such a way that ***output of one is offered as input for the successive one in the chain***
- ***Multi-tier***. Client-server architecture where applications are run by a ***multiplicity of different software agents***
- ***Blackboard***. A common knowledge base (*blackboard*) is updated iteratively by different knowledge sources, starting from including problem specification and then evolving to solution results
- ***Publish/Subscribe***. 1) Event channel and 2) Notifier (abstracting from location/distribution of broker)

Patterns for Mobile Computing

Three pattern categories for mobile middleware and applications:

- ***For distribution*** (how to distribute and access resources in runtime execution environment)
  - remote facade, data transfer object, remote proxy, observer
- ***For resource management and synchronization***
  - session token, caching, eager acquisition, lazy acquisition, synchronization, rendezvous, state transfer
- ***For communication***
  - connection factory, client-initiated connection

Distribution: Remote Facade



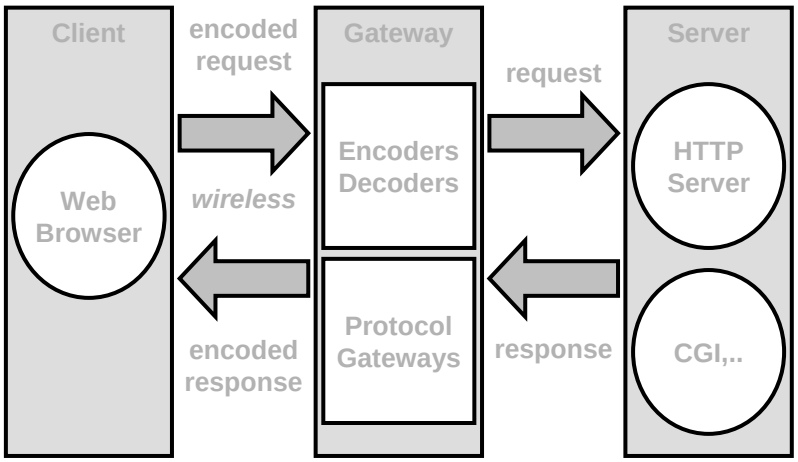
- ***Coarse-grained interface towards single or multiple fine-grained objects***
- Interface provided via ***remote gateway***
  - Accepts incoming requests that are compliant with facade interface
  - Successive interactions are fine-grained between remote facade (gateway) and object interfaces
- Application adopting the pattern does NOT need to know which specific remote servers or functions are used to implement the requested operations

Distribution: Data Transfer Object (DTO)

Data Transfer Object (DTO) pattern to offer a ***serializable container*** to transfer ***multiple data elements*** among distributed processes

- Primary goal is to ***reduce the number of remote method calls***
- Single DTO contains all the data that have to be transferred because of interest for an overall application
- Usually implemented as a simple ***serializable object*** that includes a set of fields with the corresponding *"getter & setter"* methods

Distribution: Remote Proxy



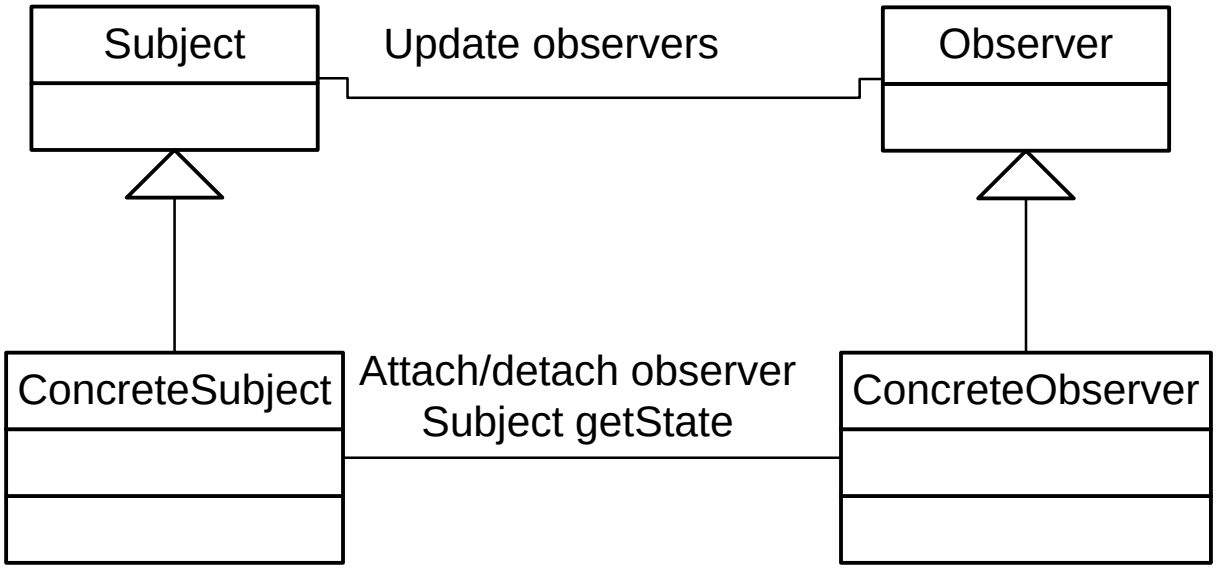
- ***Proxy (or gateway) interposed between device and network***

  By the way ☺, it is clear the difference between proxy and gateway, isn't it?
- ***All messages/packets*** (or a selected subset of them) *pass through the proxy*, which can evaluate them (also their content) and ***perform operations on them***
- Proxies also to perform computationally intensive operations in place of client device
- Proxy works as an ***adapter***, by also enabling server-side interaction ***with no need of implementing terminal-specific protocols and solutions***
- Typically need for discovery solutions to identify proxies at runtime

Distribution: Observer

Supports definition of ***one-to-many dependencies between objects***

- All objects that are considered ”dependent” are notified at the occurrence of modifications of the ***state of objects ”under observation”***
- ***Decoupling via subject and observer***
- Support to group-oriented communications, but limited scalability
- Adopted in Java and Jini event models



Resource Management: Session Token

Makes simpler and more lightweight the duties and requirements for ***server-side state management***

- ***Token emitted by server and sent to client*** (it includes data referring to the active session of the client with that server)
- ***Token includes session identifier*** (sometimes also related security info and time-to-live in order to avoid replay attacks)
- When client presents again the token to server, the server can correctly associate the client to its proper session (stato, …)

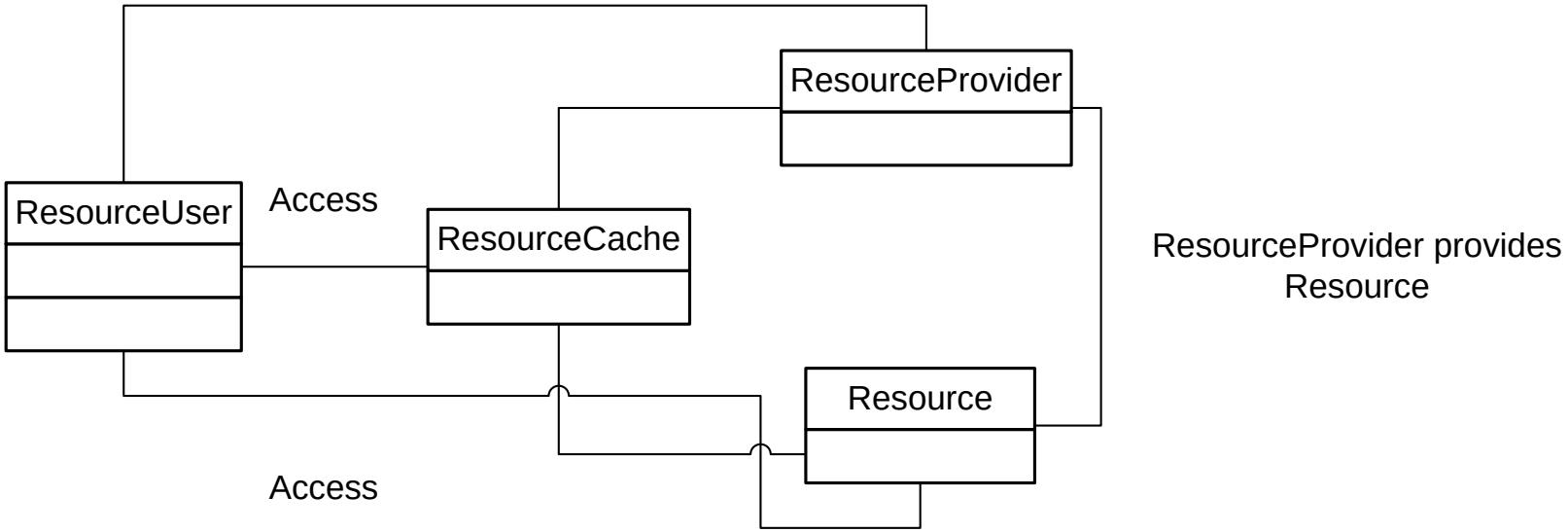
By the way, in which technologies have you already seen the application of such a solution pattern?

Resource Management: Caching

Suggests ***temporary storage of resources in local memory after their employment***, instead of their immediate destruction or de-allocation

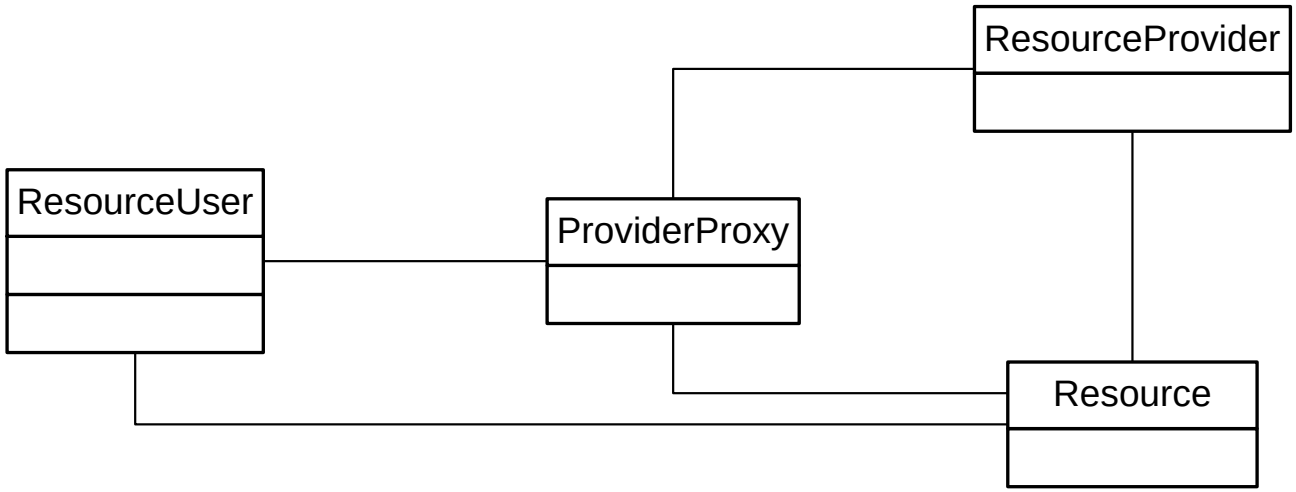
- First, ***local check in resource cache***, whenever a new request for resource/service is received
- If cache hit, immediate delivery to requesting application
- If cache miss, request is propagated and new entry created in cache

Examples of file systems, such as Coda and Aura, for pervasive computing



Resource Management: Eager Acquisition

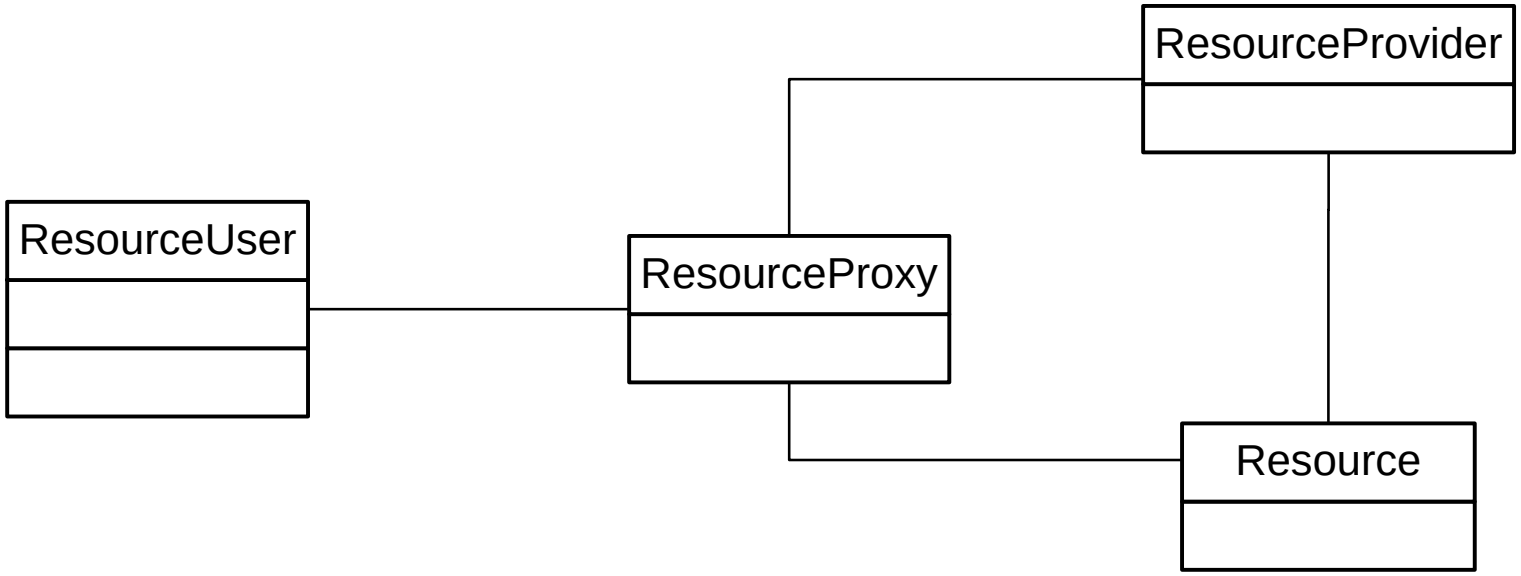
- If needed resources are known by an application from the beginning, possibility to exploit this info to ***operate pre-fetching of required resources***
- Result is that ***resources are already available locally when actual need will occur***; no necessity of remote requests
  - Examples of resources such as memory, network connections, file handlers, threads, and sessions



Resource Management: Lazy Acquisition

To optimize system resource usage, this pattern suggest to ***postpone resource acquisition till the end*** *(latest possible time)*

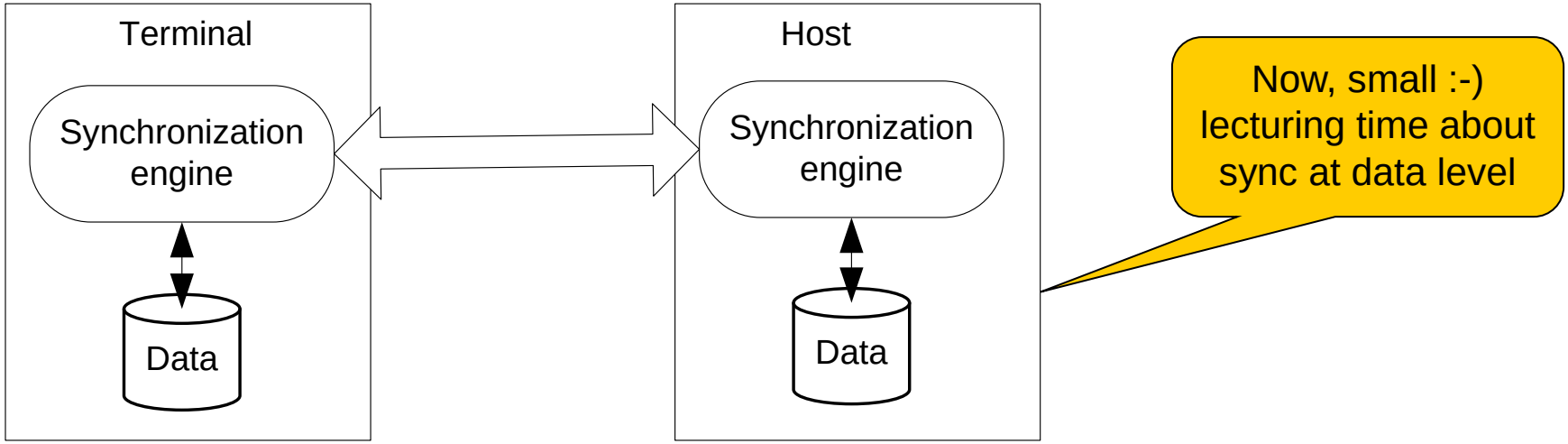
- Resource Proxy ***is responsible for intercepting*** all user requests for resources
- Resource Proxy ***does NOT acquire resources***, transparently, until the user ***does not access them explicitly***



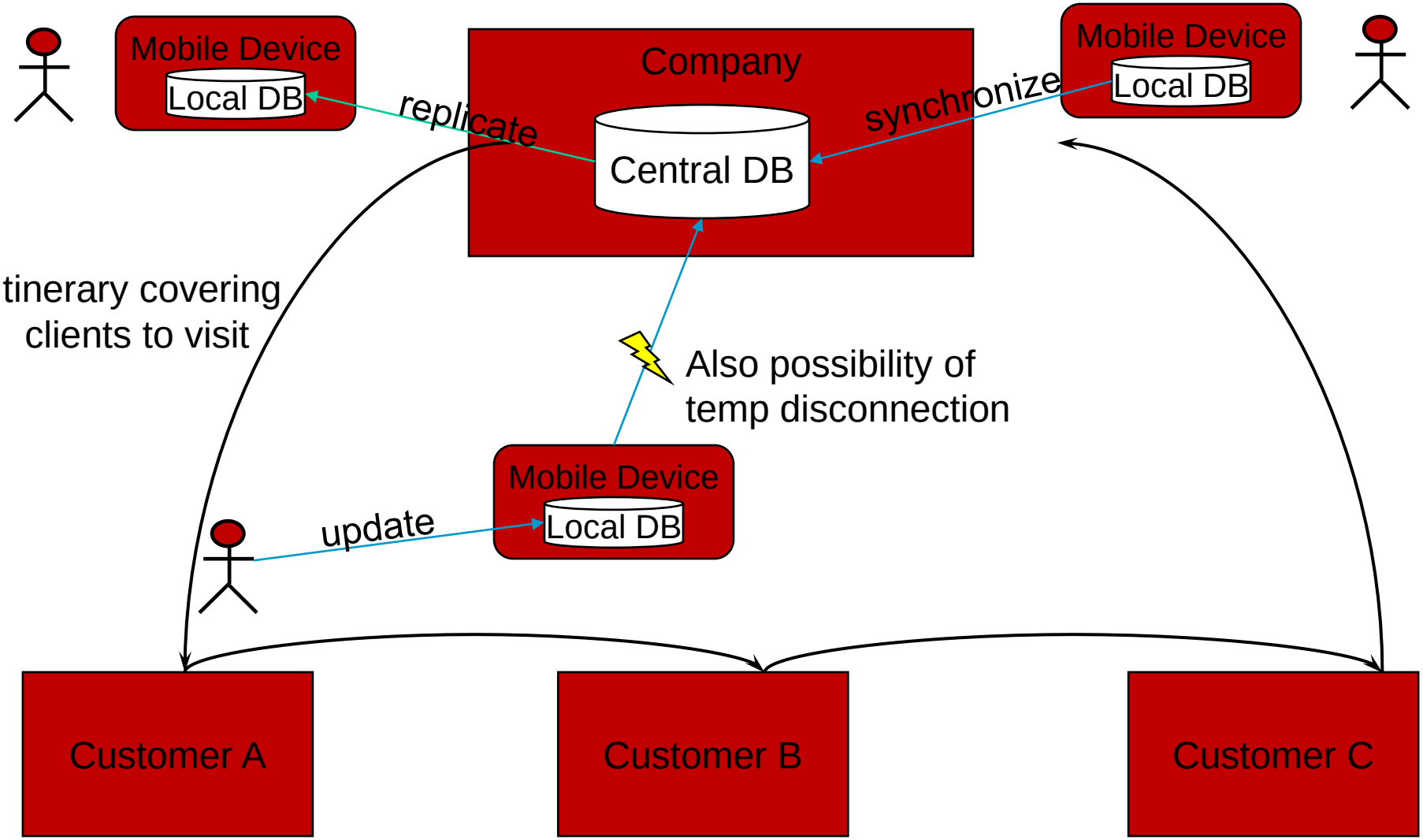
Synchronization

To efficiently manage multiple instances of data for a set of devices, this pattern suggests to ***realize and use a synchronization engine***

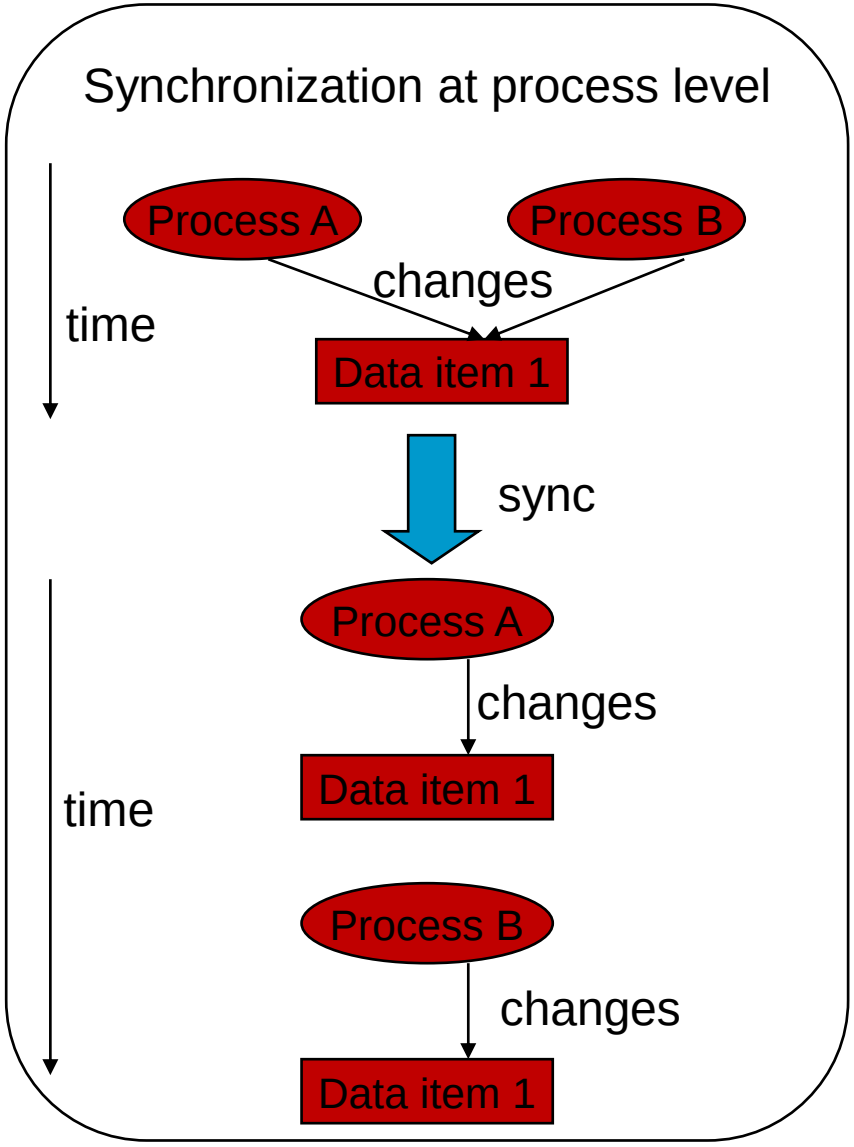
- *Sync engine* ***keeps track of modifications*** made on employed data, by exchanging modification metadata transparently (***coordination of different engines***) and by updating data ***when appropriate, e.g., connectivity is available***
- *Sync engine* ***responsible for conflict detection and resolution*** during sync process
- Possibility to access stale data and conflicts

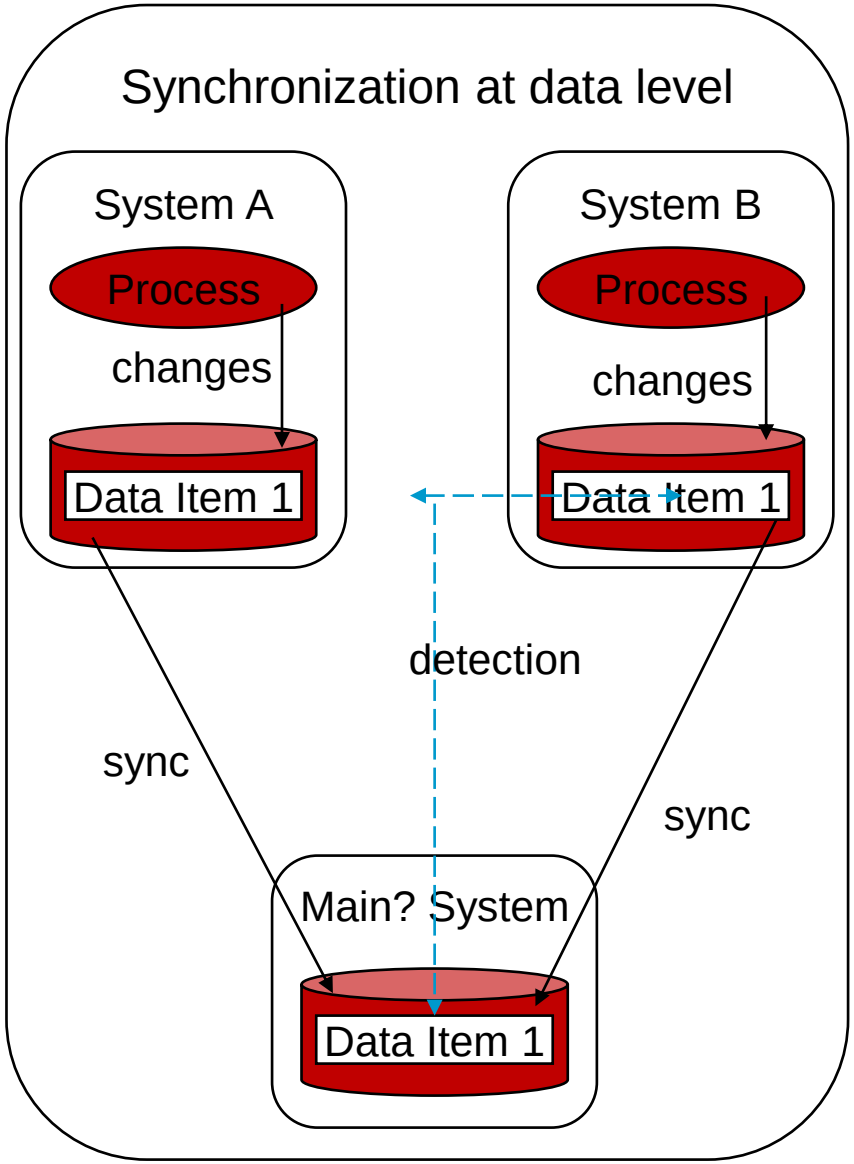


Synchronization at Data Level: Traveling Salesman Scenario

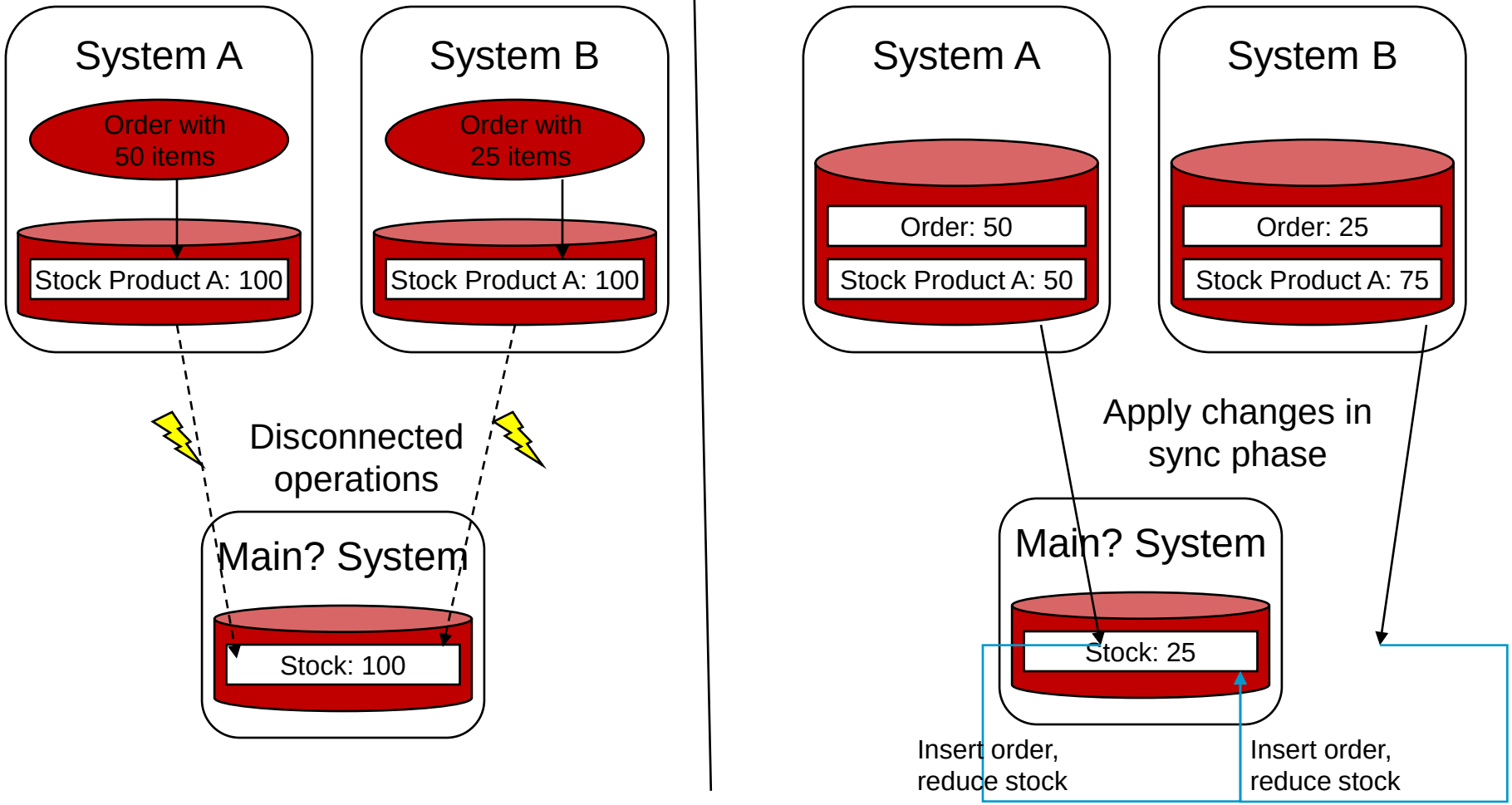


Two Primary Types of Possible Synchronization





Sometimes it is possible to merge data during synchronization



No conflicts in this case; always true?

Synchronization: Basic Models and Elements

Until an efficient implementation of the ideal concept of mobile cloud computing will not be available, seamlessly and with low cost, ***relevance of disconnected operations and synchronization in a second step/phase***

Apart from joking ☺, costs (in terms of money, energy, …) + connectivity bandwidth + discontinuous connectivity push towards ***disconnected operations and replicas/copies management***

Freedom degrees in synchronization:

- ***When*** to launch sync operations
  - Manual vs automated trigger

  See also similar problems in more traditional distributed replication
- ***How to*** synchronize (*sync styles*)
  - ***Pessimistic*** – single modifiable copy; synchronization as replication of the single modified instance
  - ***Optimistic*** – multiple copies may be modified; in a second step, trying to reconcile already performed modifications

  With which costs?

Synchronization: Basic Models and Elements

Often used basic mechanism: ***versioning***

Different possible solutions; in general, open issue on how to do versioning to improve sync process according to optimistic approach

Usually simple versioning mechanisms satisfy the following properties:

- If version B comes from a change in A, then version B is greater than version A
- If two copies have been concurrently modified in different locations, then version numbers are NOT comparable

⇒ ***Linear sequence of versions at each single location***; anyway possibility to detect concurrent modifications

Two phases (in addition to update propagation):

- ***Change detection*** (*clean or dirty copies*, also with ***conservative approach***)
- ***Reconciliation*** (need to consider syntax and semantics of data)
  - ***Log with modification operations*** – reconciliation as processing of overall log, which starts from the last common version before change
  - ***State comparison*** – it operates directly on changed data

Synchronization

In realistic cases, ***always synchronization as a process between ONLY two nodes*** that keep data copies

- ***Centralized architecture*** with one node playing the role of master for each data instance
- ***Tree-based architecture*** (sync between node and its single parent)
- More general architecture as ***cyclic connected graph*** (greater flexibility but more complex management: e.g., which previous versions to consider for state comparison?)

  Rarely used in nowadays' sync systems

Reconciliation algorithms may ***benefit from knowledge of the nature (e.g., structure) of shared data***

Growing reconciliation capabilities while passing from opaque info, to known structure, to unique id also for data parts, to known semantics, and finally to application-specific solutions

A couple of notable examples

File systems (e.g., Coda, just to mention one of the oldest):

Do you know the technical difference and distinction between *Networked File System (NFS)* and *Distributed File System (DFS)*, don't you?

- Typically DFS work on ***consistency for folder tree***, not at the level of data content for each file (delegated to an external synchronizer plug-in); often requests for explicit intervention by user
- Similarly to consistency management for an XML tree

Do you know the ancestor tools **`diff`** and **`patch`** in Unix?

Or more refined and sophisticated examples of tools, such as **`rsync`**?

A couple of notable examples

Concurrent Versions Systems (CVSs) for collaborative work, usually for sw development:

- First trivial versions with centralized server that releases a lock to a single participant at a time; NO need for reconciliation
- Now generally ***optimistic approach and centralized architecture***: ***3-way merge algorithm; conflict detection*** and possible request for user intervention
- Even first non-centralized approaches, with exploitation of ***change sets*** (each of them atomic and with unique id): synchronization as ***ordered and completed application of all change sets***

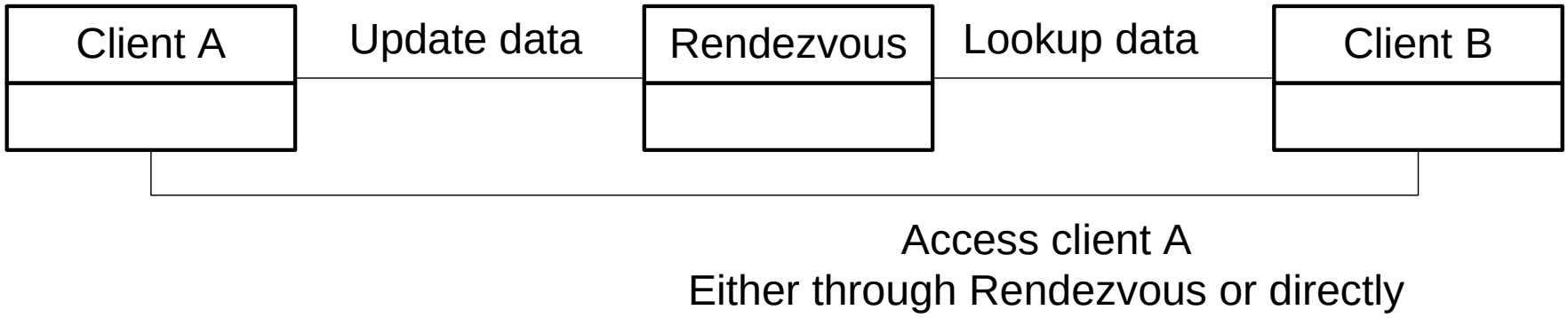
… or for example in mobile systems, ***Synchronization Markup Lamguage (SyncML)*** for mail/db synchronization

Synchronization: Rendezvous

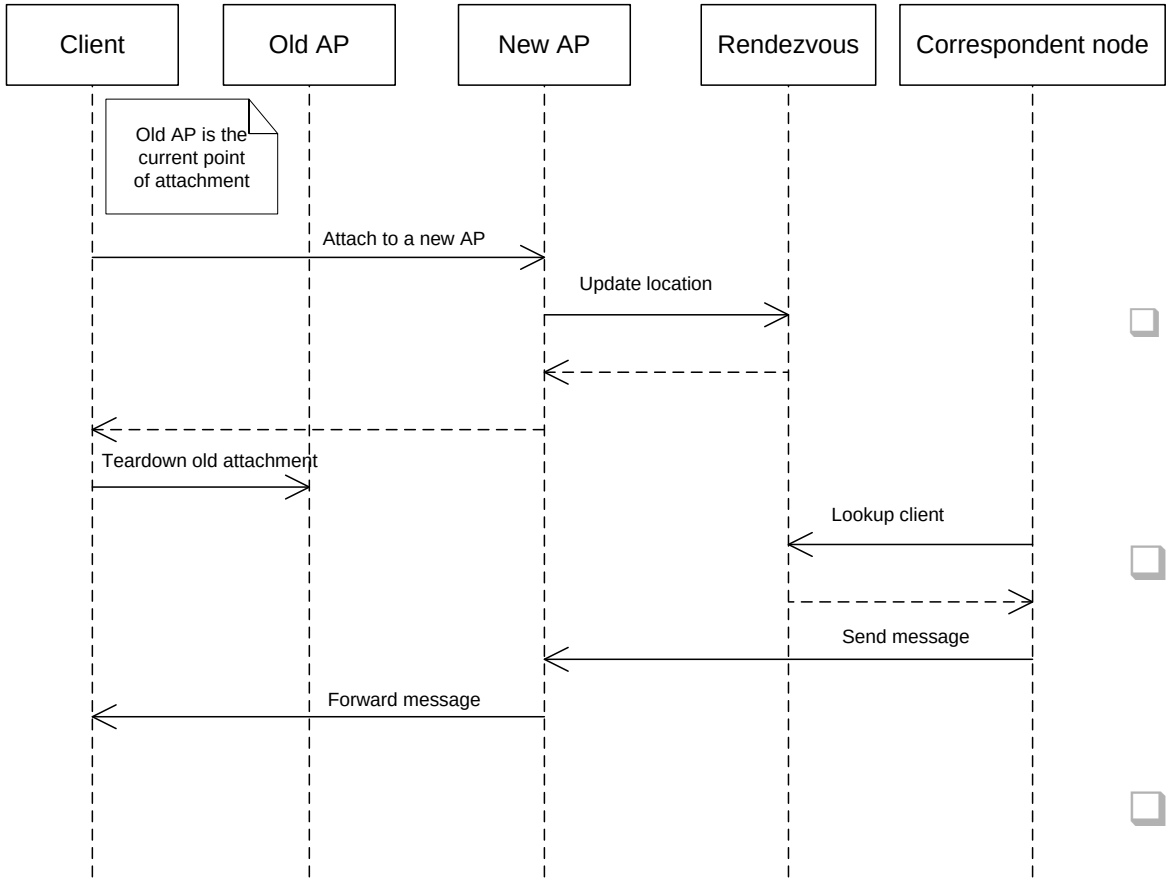
Rendezvous as frequently used pattern in network infrastructure elements (but not only) to realize management of mobile devices

- In general, ***rendezvous as a pattern to allow two or more entities to coordinate their activities***
- Typically implemented in distributed systems via *rendezvous points*
  - Entities that are ***logically centralized (indirection points)***
  - Accept messages/packets and ***keep state***, thus being able to respond, e.g., on where a mobile device is currently located

Examples: DNS, Mobile IP, HIP, ...



Resource Management & Synchro: State Transfer



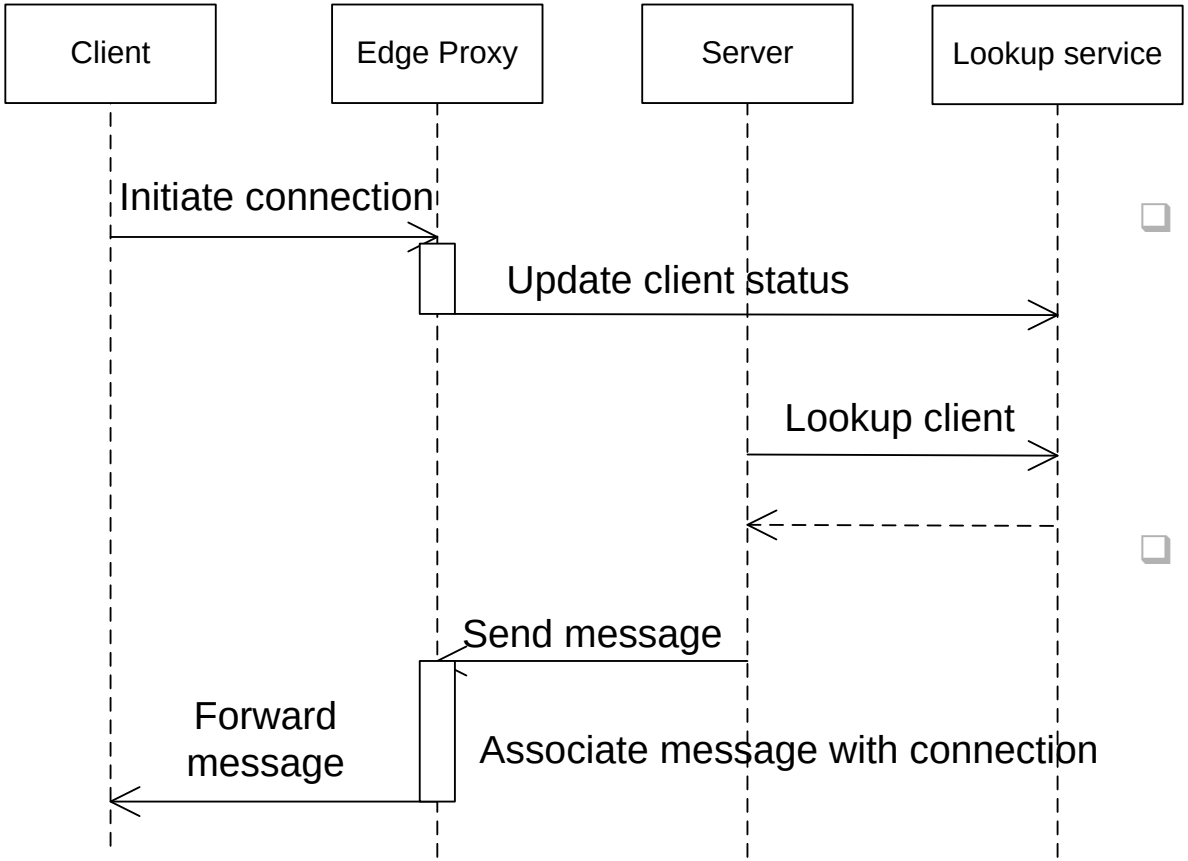
- As you already know, different types of handoff are possible
- Handoffs may need state transfer between APs
- Exploitaton of indirection point (rendezvous) to ensure reachability

Communication: Connection Factory

Suggests ***decoupling application and underlying communication system*** via introduction of a ***component used to instantiate, access, and terminate connections***

- The factory design pattern is widely used; in this case, it is employed to enable ***the efficient management and re-use of connections***
- ***Largely used in the Java world in general***. ***APIs for Java ME communications*** adopt this pattern

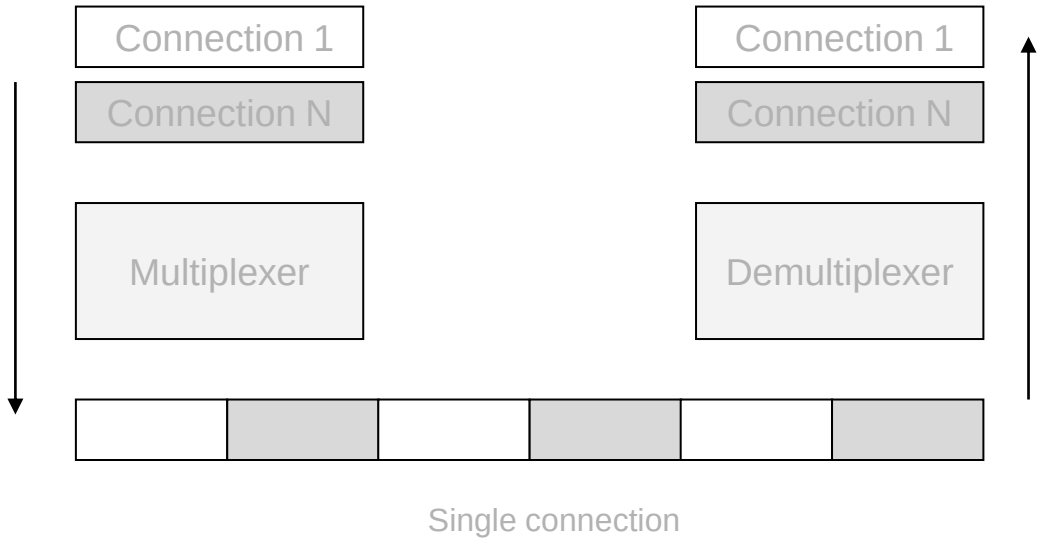
Communication: Client-initiated Connection for Push Model



- In many cases it is impossible to ***reach a mobile client due to firewall/NAT*** along the communication path
- These issues motivate the usage of ***client-initiated connections to a server with public IP address*** that then can operate ***message push towards client*** by using existing connection

  Examples: MS DirectPush, AJAX

Communication: Multiplexed Connection



- ***Inefficient to create many connections*** that may compete for network/system resource consumption
- ***Multiplexed Connection uses a single logical connection and multiplexes it to support multiple connections*** at a higher abstraction level
- Allows ***differentiated priorities*** to multiplexed messages

Example: Stream Control Transfer Protocol (SCTP)

Smartphone Platforms…

What is

And how should it be structured

a ***development/runtime platform*** to support mobile applications over mobile nodes such as ***smartphones***?

Overview on Development/Runtime Platforms

Many solutions have been proposed in the literature for development/runtime support of mobile systems middleware and applications. Why? Because in the past:

- ***Heterogeneity and fragmentation***
- Market-driven choice
- ***Very heterogeneous and differentiated characteristics*** of available devices, "platforms", and app requirements

Plethora of solutions: Symbian, Palm, RIM, Maemo/Meego, iOS, Android, Java Mobile Edition (J2ME), .NET Compact Framework (CF), Python, Lazarus, Brew, Flash Lite, Web runtime environment (micro-browser, HTML5, XHTML/CSS, JavaScript, Mobile Ajax, …), …

Let's try to put in order…

Which involved layers?

Overview on Development/Runtime Platforms

- ***Operating system layer*** (e.g., is Android an operating system?)
- ***Runtime execution support layer*** (frameworks, containers, virtual machines, …)
- ***Development support layer - SDK*** (libraries, support components, containers, …)

Again, layers with NO clear borders and NOT easily disjoint…

- Mainly at OS layer: Symbian, Palm, RIM (BlackBerry), Maemo/Meego, iOS, Linux?, …
- Mainly at runtime execution support layer: Kernel-based Virtual Machine (KVM) or Dalvik Virtual Machine for Java, Common Language Runtime (CLR) for .NET, Flash Lite, Web runtime environment

Overview on Development/Runtime Platforms

Generally speaking, for the smartphone market segment,

Two primary approach categories:

- Based on ***native applications***
- Based on ***Web integration*** (in particular, see standardization efforts associated with HTML5)

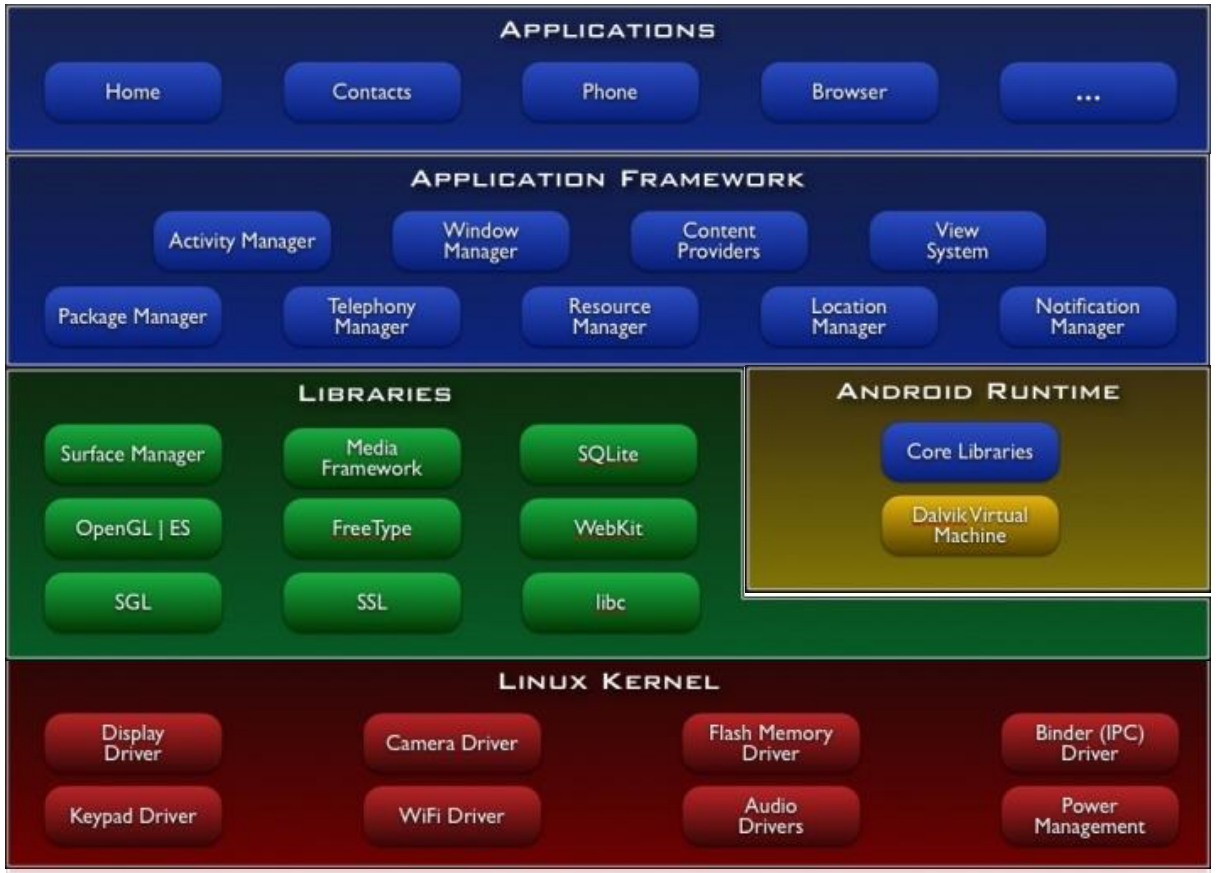
Let us start with the first approach (***native applications***), by choosing to focus primarily on ***Android*** and only rapidly and superficially on iOS

Android: Architecture

Classical ***hierarchical architecture***, structured into layers
(growing complexity from bottom to top)

Layers:

- Linux kernel
- Libraries (native Linux)
  + Android runtime
  (Dalvik VM + core libraries)
- Application Framework
- Applications



Android: Architecture

Kernel Layer

- Based on traditional Linux v3.x kernel and evolutions
- It introduces Hardware Abstraction Layer (HAL)

Libraries

- ***in native language (C/C++)***

Android Runtime

- Execution environment for ***applications, written in Java***
- Based on Dalvik VM / Android Run Time (ART)

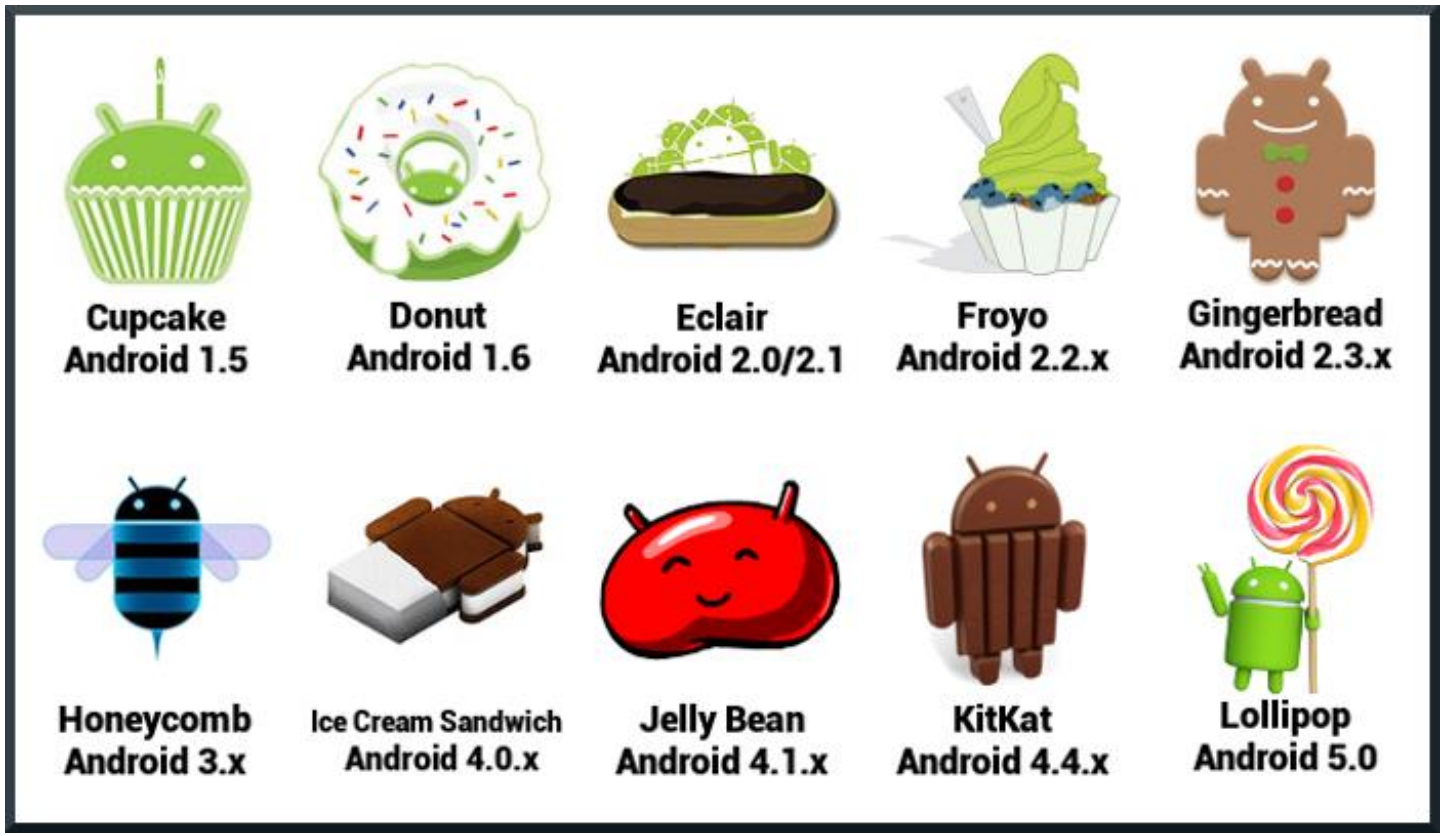
Application Framework

- It provides applications with advanced services, ***encapsulated in Java objects***,

Application

- Core Apps: provided by the «initial» onboard system
- Developers' written apps…

Android:
Elements about Version Evolution



Ice Cream Sandwich 4.0.1 (October 2011)

- Linux kernel 3.0.1; customizable launcher; "Contacts" app fully integrated with main social network applications; Android Beam (data exchange via NFC); ***Wi-Fi Direct***, etc. etc.

Android: Elements about Version Evolution

Jelly Bean 4.1.1 (July 2012)

- Linux kernel 3.1.10; new features for picture/video sharing via NFC; advanced voice recognition; official end of the support of Adobe Flash, etc. etc.

Kit Kat 4.4 (October 2013)

Support for **3 new types of sensors** (geomagnetic rotation vector, step detector and counter), added step counter feature; decreased battery consumption during audio playing; **Android RunTime** (ART) and new compiler, experimental in this version, that can be activated in Developers Options (not on all devices); optimized working on devices with limited RAM, …

Android: Elements about Version Evolution

Lollipop 5.0-5.1.1 (November 2014)

Introduction of ***Google Fit*** for physical activity; new ***Linux 3.10.x. kernel***; ***removal of Dalvik runtime, that is officially replaced by ART***; native 64 bit support; Bluetooth 4.1; improvement of graphical performance thanks to OpenGL ES 3.1 support; improvement of camera support thanks to dedicated APIs; ***added multi-user features on smartphones***; …

Marshmallow 6.0 (October 2015)

Doze mode, with CPU speed reduction when display is switched off; native support to fingerprint readers; Direct Share for sharing among apps; ***request of post-install/run-time permissions***; USB Type-C support; support to External storage to make it uniform to Internal Storage; MIDI support for music instruments; ***experimental multi-window feature***

Android: Elements about Version Evolution

Nougat «N» 7.1.2 (August 2016) – API 25

Ability to display ***multiple apps on-screen*** at once in a ***split-screen view***; support for inline replies to notifications; ***OpenJDK-based*** Java environment; support for the Vulkan graphics rendering API; "seamless" system updates on supported devices; …

Oreo 8.0 (August 2017) – API 26

Project Treble, i.e., modular architecture that makes it easier and faster for hardware makers to deliver Android updates; ***multi-display support***

Pie 9.0 (August 2018) – API 28

Experimental features (hidden within a menu called Feature Flags) such as automatic Bluetooth enabling while driving; ***DNS over TLS***; ***Vulkan 1.1*** - low-overhead, cross-platform 3D graphics and computing API

Android 10.0 (September 2019) – API 29

New permissions to access location in background; background apps can no longer jump into foreground; support for the WPA3 Wi-Fi security protocol; support for ***Notification Bubbles***; ***allows core OS components to be updated via Play Store***, ***with no complete system update***

Android: Elements about Version Evolution

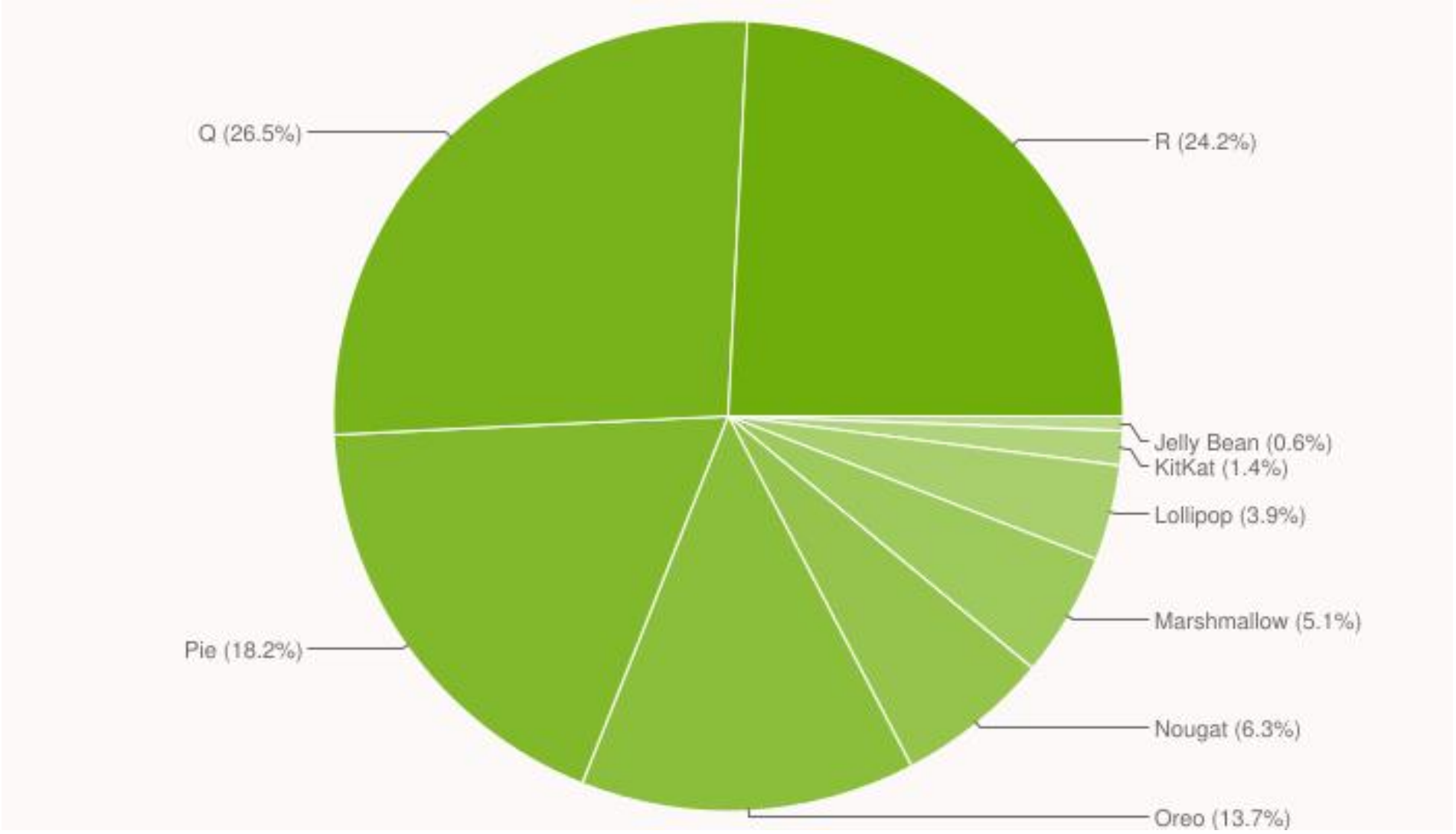
Android 11.0 (September 2020) – API 30

New permissions to access location, camera, microphone, …, and **more constrained** permissions also for background apps and apps not opened for long; support for more isolation between different apps (it removes the ability to see other installed apps and limits access to the local storage); support for ***floating Bubbles***; support for ***conversation-related system notifications***; ***contextual menu for smart devices control***

Android 12.0 (October 2021) – API 31

Option to choose ***precise or approximate location***; privacy dashboard; ***performance improvements to system services*** to improve transitions, ***power efficiency***, and reduce app startup times

Android versions distribution in 2021



Android: OS Kernel Layer

Kernel Linux 3.x

Hardware Abstraction Layer (HAL)

- Memory management
- Process management
- Network stack
- Standard Linux power management
- …

Kernel extensions

- ***Ashmem***: ***shared memory manager***; reference counting and automatic deallocation by the kernel
- ***IPC binder***: low overhead thanks to the usage of Ashmem (rigid access discipline via memory block descriptors)
- ***Advanced power management***: exploitation of different energy management policies via WakeLocks

It is NOT a real complete Linux OS kernel. It lacks:

- Native window manager system
- Complete support to GNU C library
- Complete support to standard Linux utilities

Power Management and WakeLocks

Evident example of cross-layering

Android apps (with permission to access Power Manager) can achieve ***control of energy consumption by enforcing the desired policy***:

- Always active CPU, also with switched-off display
- Prioritary CPU with display that is at least backlit
- …

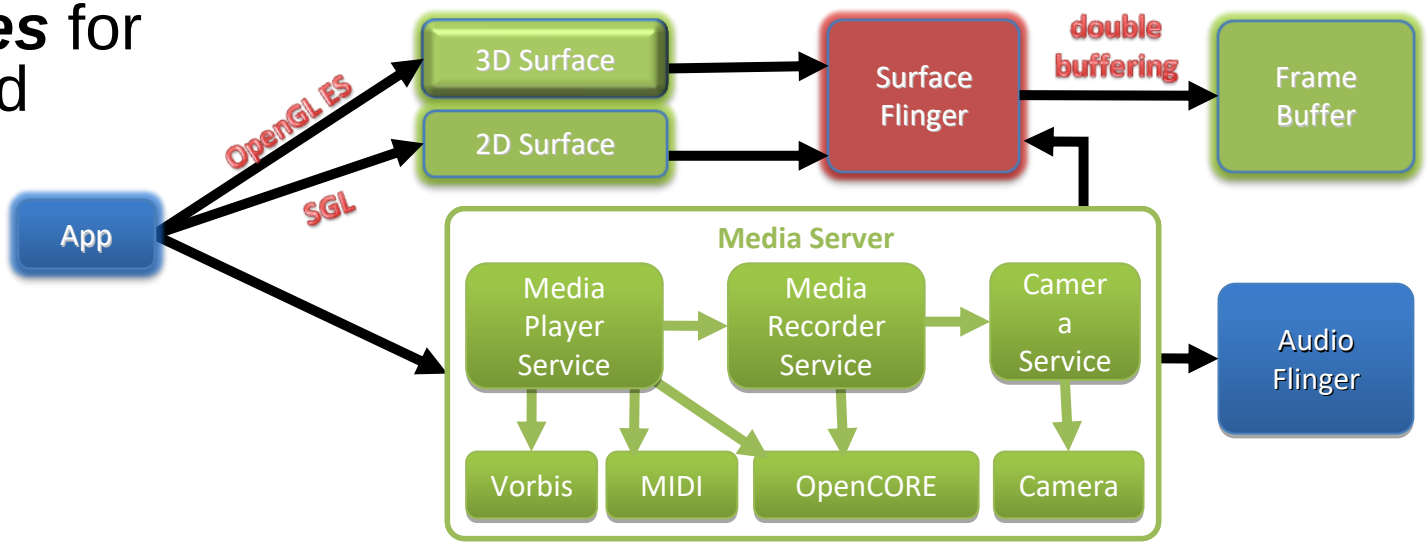
Via ***WakeLocks***: access locks to the features of Power Manager (different types of WakeLock). For instance:

```
… @Override
protected void onCreate(Bundle savedInstanceState) {
   PowerManager pm = (PowerManager)
       getSystemService(Context.POWER_SERVICE);
   wl = pm.newWakeLock(PowerManager.FULL_WAKE_LOCK,
       "DoNotDimScreen"); }

@Override
protected void onPause() { super.onPause(); wl.release(); }
```

Android: Native Libraries and Dalvik VM

Native libraries for graphics and multimedia



Surface Manager + Media Framework

Dalvik VM

- designed for mobile devices: ***registry-based*** (exploiting the ARM architecture). Instead, how is the traditional JVM organized?
- It interprets and runs ***dex*** files, obtained via transformation of ***class files*** (30% reduction of necessary instructions and improvement of runtime perf)
- optimized support to garbage collection

Android: Application Framework

Activity

- A ***single action*** that one user can accomplish via a window (it corresponds usually to a single screen)
- It is the fundamental Android component
  - E.g., *home activity*

Intent

- Maximum reusability of activity
- ***Request*** to perform an operation (e.g., choice of a phone number)
- Received by a component that has a ***compatibile Intent Filter***

Service

- Running in ***background*** (no interaction with user, differently from activity)
- Usable by 1+ components
- No dedicated process/thread in background

Broadcast Receiver

- Responds to compatible Intents by executing the associated operations
- Typically, ***notify actions*** (calls, sms transmissions, …)
- Lifecycle is limited to the response
- Differently from activities and services, ***multiple receivers*** can be activated by ***a single intent***

Android: Application Framework

- ***Package and Activity Manager***

  Manage the lifecycle of Activities and apps, included in ***Android Packages*** (APK). Each APK includes a descriptor (manifest), executable dex, and the associated «static» resources (xml, png, …), according to a predefined file structure

- ***Window Manager and View System***

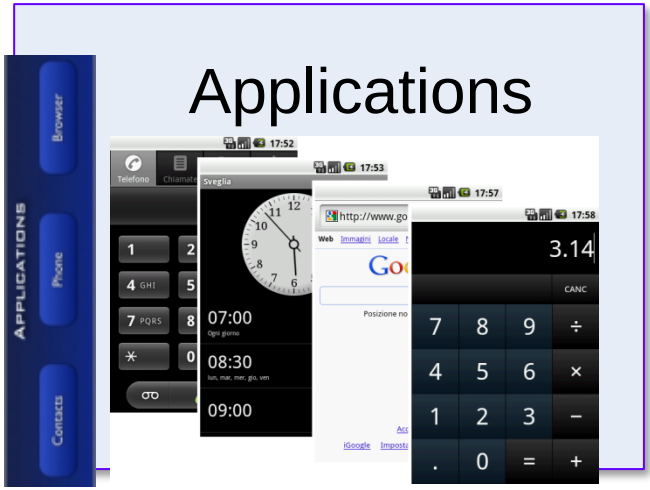
  Offer advanced graphics features that can be used directly in apps. View System is based on the ***View class***, and consists of GUI components that interact with user and with event managers (no Java Swing, no AWT)

- ***Resource Manager and Content Provider***

  Resource manager for any resource (all files except for code) and shared access to ***local data*** (SQLite RDBMS and persistence through files)

- ***Telephony, Notification, and Location Manager***

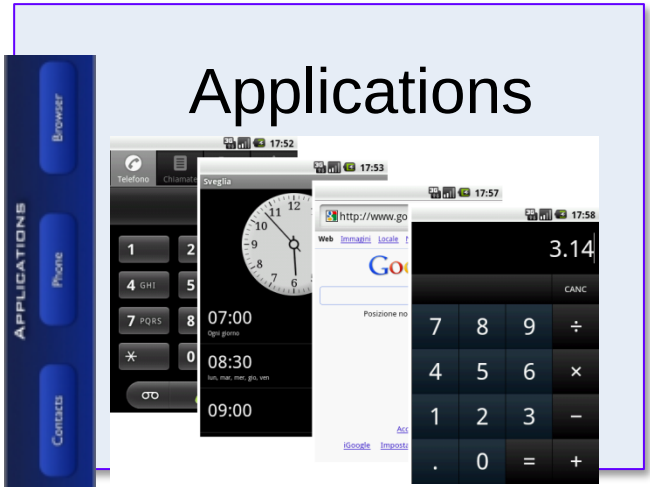
  Allow to access functionality for telephony, notification, and positioning



Core Applications

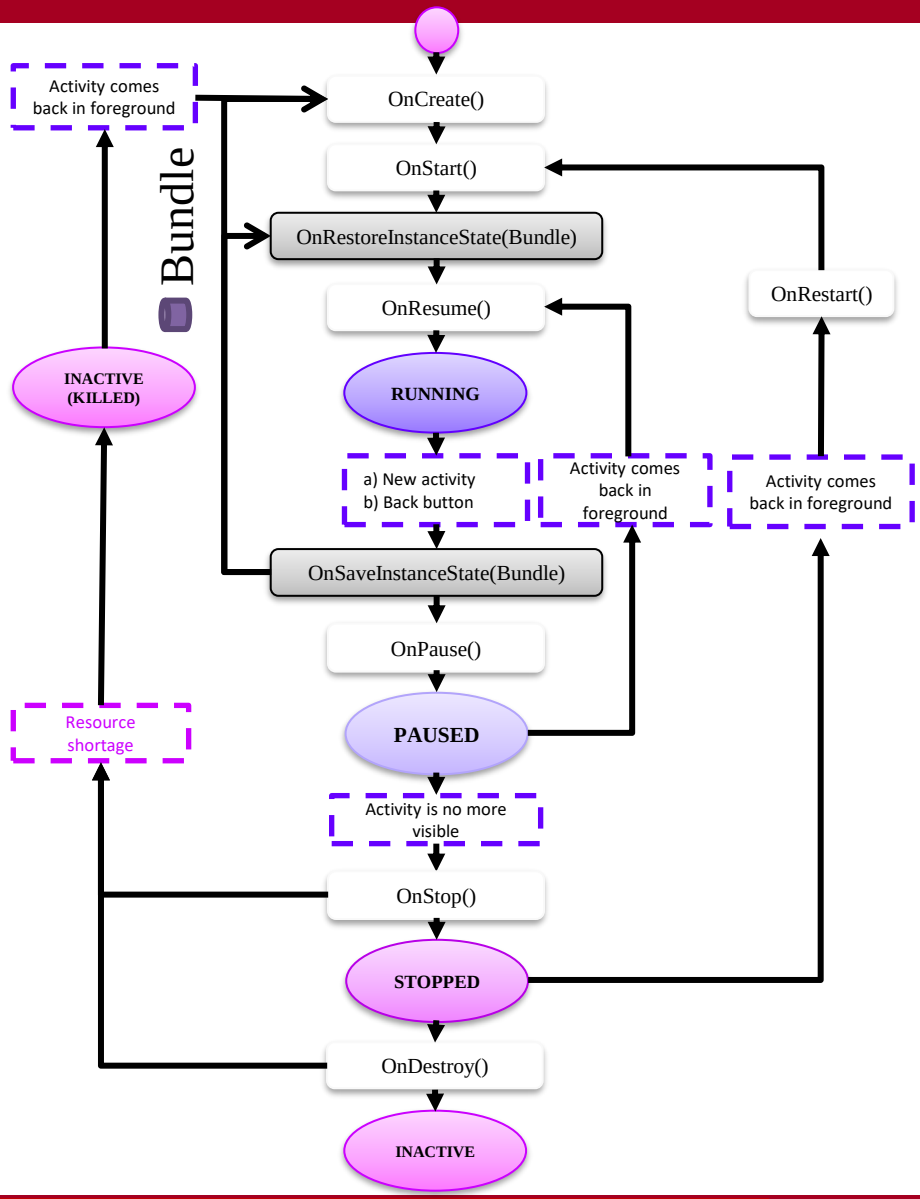
Real and regular apps (same execution model as for apps developed by third parties and downloaded by users), pre-installed at default over an Android device

- ***Home application*** (and associated activity): manager app frm which it is possible to launch execution of other apps. Which threading model?
- ***Message management***
- Client app for ***email***
- ***Contact book***
- ***Map management***
- ***Web browser***: WebKit browser engine (open-source); used also in Safari and Google Chrome
  Only browser engine (HTML parser + renderer + JavaScript engine)

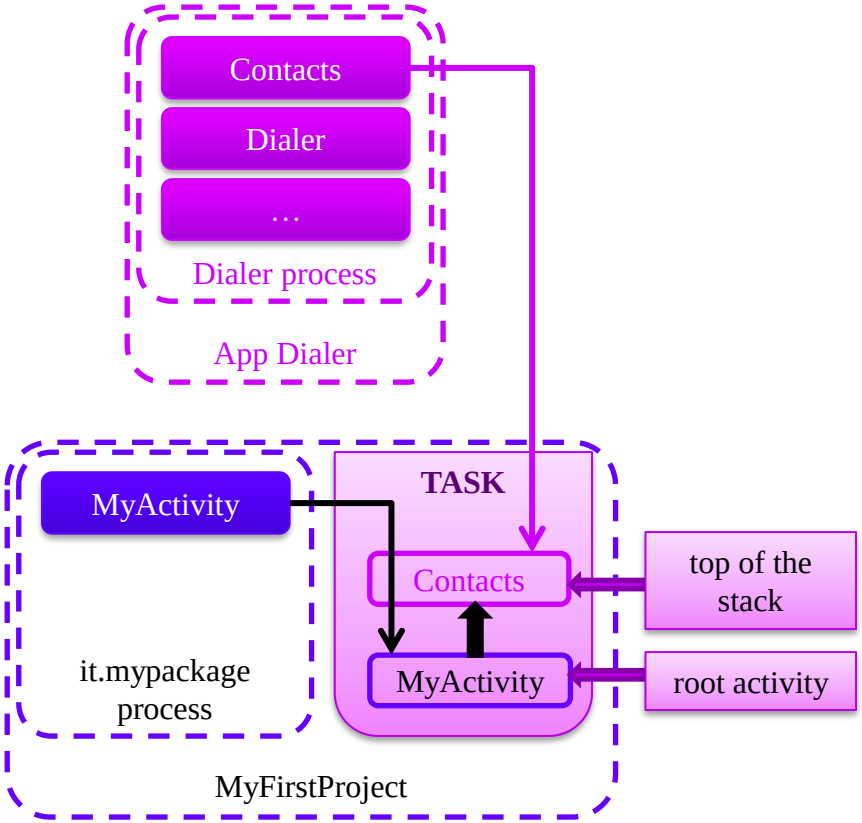


Lifecycle Management of Android Activities

- Activity is the component for user interaction
- It extends the `Activity` Java class
- In general, many activities are running concurrently: the active one (**RUNNING**) is only ONE. If visible but not active, it is **PAUSED**, otherwise **STOPPED**
- For lifecycle management, it is possible to redefine ***callback methods OnCreate, OnStart, OnResume***,…
- Activities use resources: an activity may be deallocated due to resource shortage → **KILLED state**
- Android offers an ***info container called Bundle***, where to save the state to be recovered at re-allocation



Conversation Concept: Android Task



One App may include several activities:

- Independent and disjoint
- Associated the one with the other

Activities are different from forms

- They should be simple
- They should be usable and reusable

→ there is the need to structure activities to compose a more ***complex conversation*** with user (as for Web pages)

A task models a conversation

- It includes an ***activity stack***, also of different apps: on top the only active activity in the task
- Starting an activity pushes it on top of the stack; closing it discards the activity from the stack
- It may be ***in foreground or in background***

Intent and Intent Filter

Activation of a component is triggered by an Intent

(usually to pass from an activity to the following one)

- ***explicit***: component to activate is known at compile time; it needs the Class descriptor of the component
- ***implicit***: component to activate is NOT known at compile time; it needs that the following data are specified
  - ***action and category***: they describe respectively action and type of the component to activate for execution
  - ***url***: it specifies data to be processed by the activated component
  - ***mime type***: it specifies the data type used

```
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("http://www.unibo.it"));
```

Component is selected ***based on applicable Intent Filters*** (description of which intents an activity can handle) ***that are declared in the manifest***, according to an algorithm of ***Intent Resolution***

Intent repository on www.openintents.org

Intent and Intent Filter

```
<activity android:name="IntentActivity">
        <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.
                LAUNCHER" />
        <category android:name="it.mypackage.intent.
                category.CAT_NAME" />
        <data android:mimeType="vnd.android.cursor.item/
        vnd.mytype" />
        </intent-filter>
</activity>
```

Example of manifest file

```
protected void onCreate(Bundle savedInstanceState) {
...
Intent intent = new Intent(); intent.setAction(MY_ACTION);
intent.addCategory(Intent.CATEGORY_ALTERNATIVE);
intent.addCategory(Intent.CATEGORY_BROWSABLE);
Uri uri = Uri.parse("content://it.mypackage/items/");
intent.setData(uri);
intent.setType("vnd.android.cursor.item/vnd.mytype");
startActivity(intent); ... }
```

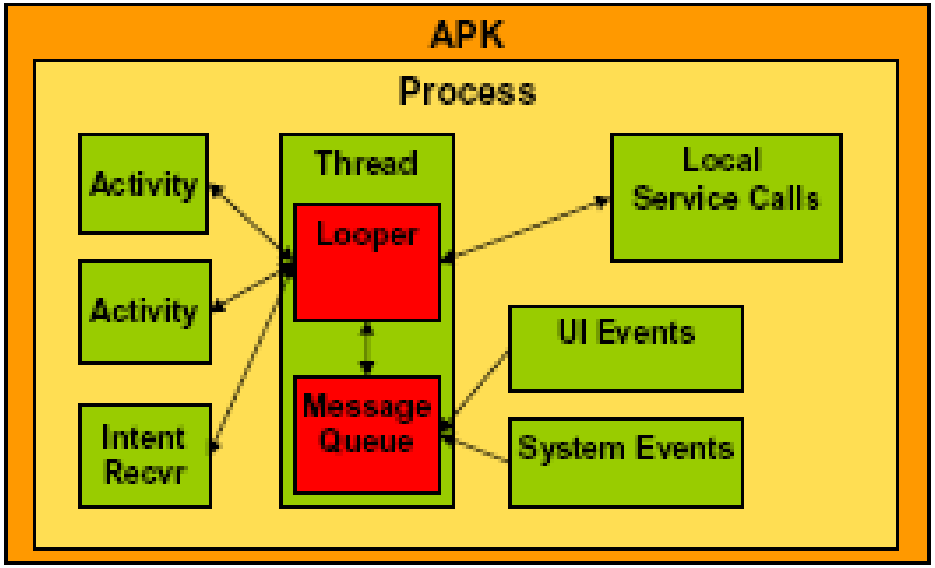
Threading Model in Android

Each application has ***a single thread (at default)*** => simple *single-threaded model* (1 app: 1 thread)

- 1 app: 1 system thread "assigned" to the app: multiple activities
- Possibility to ***save the state*** in info bundle (not very different from execution model of stateful session beans in J2EE)
- ***Each thread has a Looper*** for message queue management

Looper.loop

```
while(true){
    Message m=queue.next();
    // may block
    if(m!=null){
        m.target.dispatch-
                    Message(m);
        m.recycle();
    }
}
```



Threading Model in Android

Two options: *1) to* ***launch different apps in a single Dalvik VM (single heavy process)***; 2) to ***have a single Dalvik VM (process) dedicated to each single app***

The default is the second option: any app is put in execution into a separated VM (separated isolated process), also when triggered by `startActivity(intent)` or `startService(intent)`

How is it possible? Which potential issues?

For instance, how to perform response return to the «invoking» activity?

`startActivityForResult(Intent, int)`

second parameter identifies the call

callback function `onActivityResult(int, int, Intent)`

If it is crucial to optimize system resource consumption, ***there is the need to impose that different apps share the same userID***

`Android.sharedUserId="PaoloBellavista"` in manifest file

Which potential security issues?

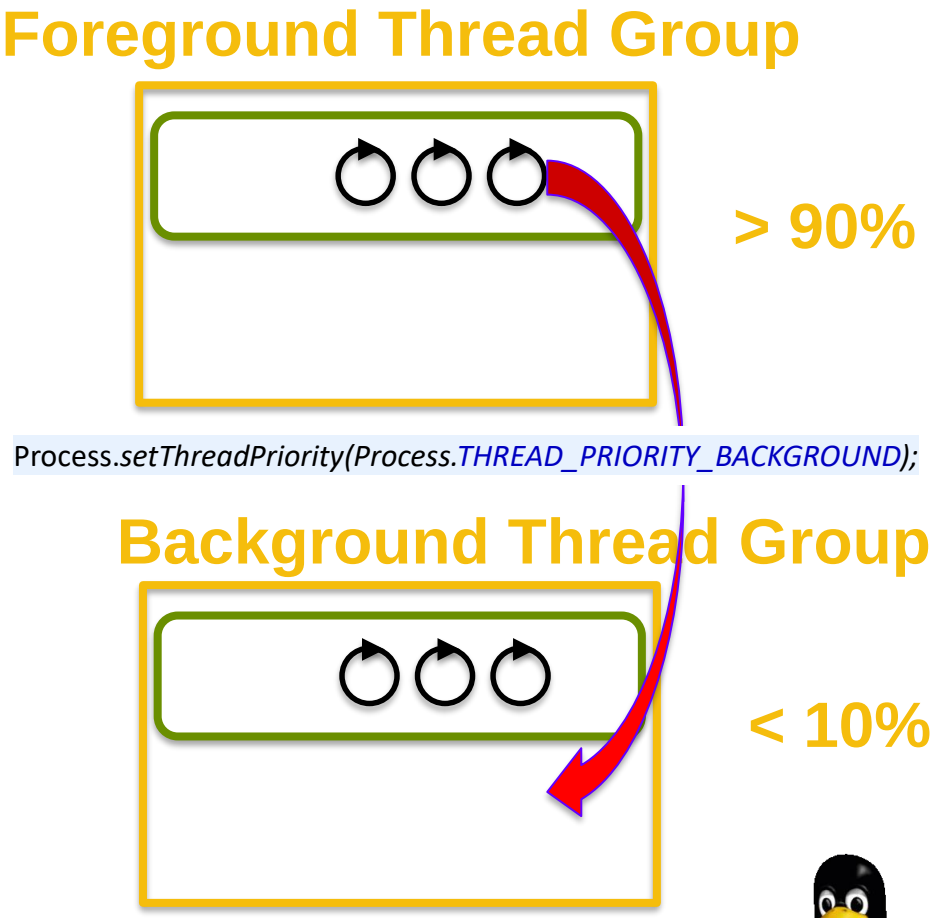
Android Scheduling

Process level:

1. Foreground
2. Visible
3. Service
4. Background

App A

App B



Asynchronous Techniques

- ***Thread***
- Executor
- ***HandlerThread***
- ***AsyncTask***
- ***Service*** (already partially discussed)
- IntentService
- AsyncQueryHandler
- Loader

Services for long tasks

Services are an ***advanced topic*** and considered complex in Android

A service is an ***application component*** that can perform ***long-running operations in the background*** and does ***NOT provide a user interface***

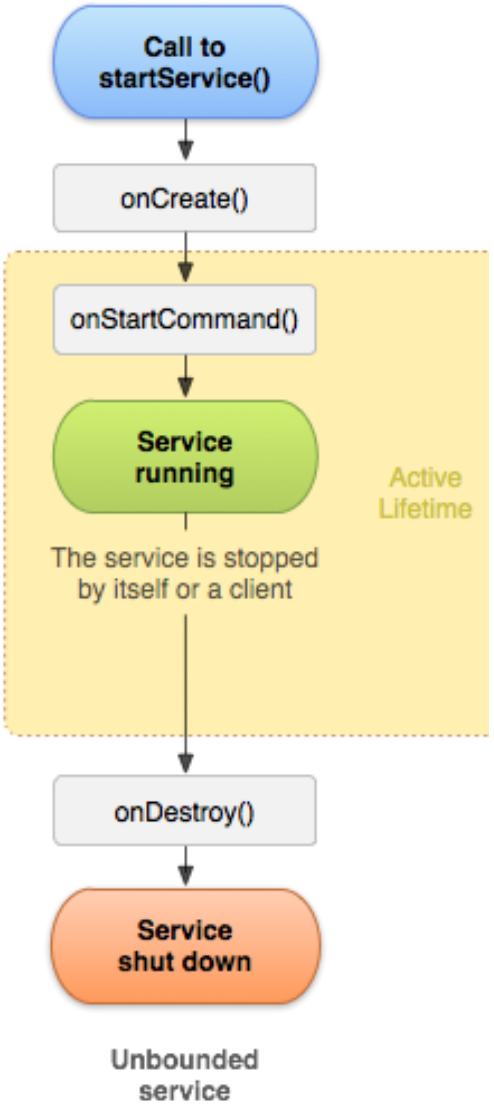
- Network transactions
- Play music
- Perform file I/O
- Interact with a database

Main Characteristics of Android Services

- ***Started with an Intent***
- Can stay running when user switches applications
- Lifecycle—which developers have to manage
- Other apps can use the service—manage permissions
- ***Runs in the main thread of its hosting process***

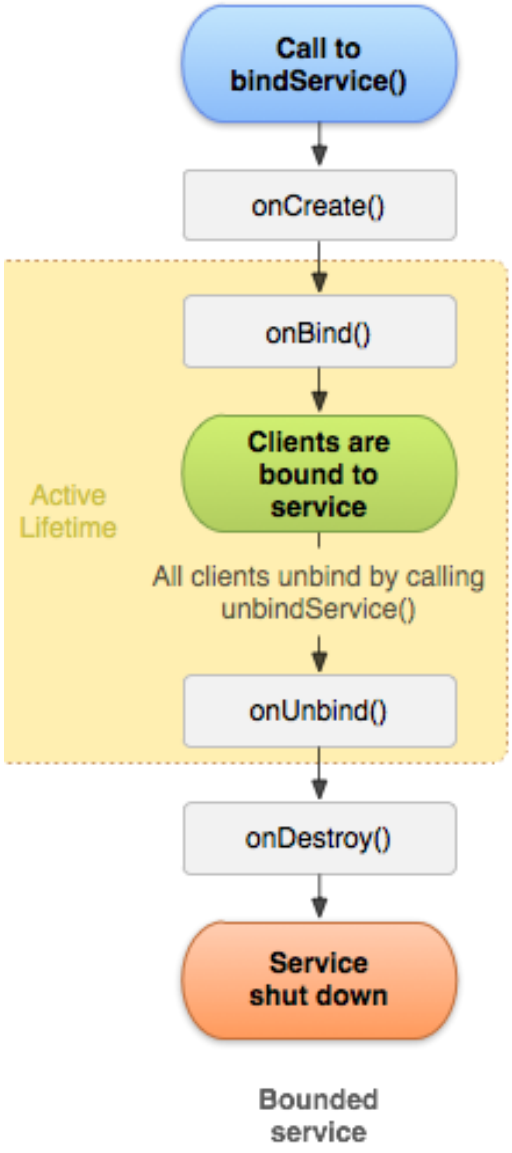
“Forms” of Android Services: started

- Started with **`startService()`**
- ***Runs indefinitely until it stops itself***
- Usually does ***NOT update the UI***



“Forms” of Android Services: bound

- Offers a ***client-server interface*** that allows components to interact with the service
- Clients send requests and get results
- Started with **`bindService()`**
- ***Ends when all clients unbind***



Services and Runtime Thread Model

- Although services are separate from the UI, they still run on the ***main thread*** by default (*except IntentService*)
- ***Offload CPU-intensive work to a separate thread within the service***

If the service can't access the UI, how do you update the app to show the results?

Use a broadcast receiver!

Foreground Services

Run in the background but ***require that user is actively aware it exists***—e.g. music player using music service

- ***Higher priority than background services*** since user will notice its absence—unlikely to be killed by the system
- ***Must provide a notification*** which the user cannot dismiss while the service is running

Background Service Limitations

- ***Starting from API 26,*** background app is NOT allowed to create a background service
- A foreground app, can create and run both foreground and background services
- When an app goes into the background, the system stops the app's background services
- The **`startService()`** method throws an IllegalStateException if an app is targeting API 26
- These limitations do not affect foreground services or bound services

Creating/Stopping a service

Creating a service:

- <service android:name=".ExampleService" />
- Manage permissions
- Subclass IntentService or Service class
- Implement lifecycle methods
- Start service from Activity
- Make sure service is stoppable

Stopping a service:

- A started service must manage its own lifecycle
- If not stopped, will keep running and consuming resources
- The service must stop itself by calling stopSelf()
- Another component can stop it by calling stopService()
- Bound service is destroyed when all clients unbound
- IntentService is destroyed after onHandleIntent() returns

IntentService

- ***Simple service with simplified lifecycle***
- Uses ***worker threads*** to fulfill requests
- ***Stops itself when done***
- Ideal for one long task on a single background thread

Intent service limitations:

- Cannot interact with the UI
- Can only run one request at a time
- Cannot be interrupted

Example of IntentService Implementation

```
public class HelloIntentService extends
IntentService {
    public HelloIntentService() {
super("HelloIntentService");}

    @Override
  protected void onHandleIntent(Intent intent) {
      try {
          // Do some work
      } catch (InterruptedException e) {
          Thread.currentThread().interrupt();
      }
  } // When this method returns, IntentService stops
the service, as appropriate
}
```

References for additional material

- Services overview
- Background Execution Limits

Android Threads

- Act much like ***usual*** Java Threads
- ***Cannot work*** directly on ***external User Interface objects*** (throw the Exception CalledFromWrongThreadException: “Only the original thread that created a view hierarchy can touch its views”)
- **Cannot be stopped by executing destroy() nor stop()**. Use instead interrupt() or join() (by case)

- As usual, two main ways of having a Thread that executes application code:
  - Providing a new class that extends Thread and overriding its run() method
  - Providing a new Thread instance with a Runnable object during its creation.

  In both cases, start() method must be explicitly called to actually execute the new Thread

Android Handler

- ***Associated with a single thread and that thread's message queue***
- Bound to the thread/message queue of the thread that has created it
- ***It delivers messages and runnables to that message queue***
- ***It executes them as they come out of the message queue***

Two main types of usage for a Handler:

- To ***schedule messages and runnables*** to be executed as some point in the future
- To add an ***action into a queue performed on a different thread***

Handler Example

Create Handler object

```
public Handler handleFbLogin = new Handler() {
        @Override
        public void handleMessage(Message msg) {
                if (msg.what == LOGIN_FACEBOOK) {
                        btnShareFacebook.setChecked(true);
                }

                if (msg.what == MESSAGE_LOGIN_FACEBOOK_ERROR) {
                        btnShareFacebook.setChecked(false);
                }
        }
};
```

Use Handler

```
Message message = handleFbLogin.obtainMessage(LOGIN_FACEBOOK, "");
handleFbLogin.sendMessage(message);
```

Android AsyncTask

- ***Created on the UI thread*** and can be executed only once
- ***Run on a background thread and result is published on the UI thread***
- The three parameters of an AsyncTask are:
  - ***Params***, the type of the parameters sent to the async task upon execution
  - ***Progress***, the type of the progress units published during the background computation
  - ***Result***, the type of the result of the background computation
- Go through 4 steps:
  - **onPreExecute()**: invoked on the UI thread immediately after the async task is started
  - **doInBackground(Param ...)**: invoked on the background thread immediately after onPreExecute() finishes executing
  - **onProgressUpdate(Progress...)**: invoked on the UI thread after a call to publishProgress(Progress...)
  - **onPostExecute(Result)**: invoked on the UI thread after background computation finishes

AsyncTask Example

Subclass of AsyncTask

```
public class LoadItemList extends AsyncTask<ArrayList<Item>, Void, ArrayList<Item>> {
	@Override
	protected ArrayList<Item> doInBackground(ArrayList<Item>... params) {
		//make something and return the result
		return itemList;
	}

	@Override
	protected void onProgressUpdate(Void... unsued) {

	}

	@Override
	protected void onPostExecute(ArrayList<Item> sResponse) {
		//update UI
	}
}
```

Use AsyncTask

```
LoadItemList loadItemList = new LoadItemList();
loadItemList.execute();
```

To Summarize: Pay Attention…

```
public void onClick(View v) {
  new Thread(new Runnable() {
    public void run() {
      Bitmap b = loadImageFromNetwork();
      mImageView.setImageBitmap(b);
    }
  }).start();
}
```

- ***Violate the single thread model***: the Android UI toolkit is not thread-safe and must always be manipulated on the UI thread
- In this piece of code, ***ImageView is manipulated on a worker thread***, which can cause really weird problems. Tracking down and fixing such bugs can be difficult and time-consuming

To Summarize: Pay Attention…

```
public void onClick(View v) {
  new Thread(new Runnable() {
    public void run() {
      final Bitmap b = loadImageFromNetwork();
      mImageView.post(new Runnable() {
        public void run() {
          mImageView.setImageBitmap(b);
        }
      });
    }
  }).start();
}
```

- ***Classes and methods*** also tend to make the code more complicated and ***more difficult to read***
- It becomes even worse when implementing ***complex operations that require frequent UI updates***

To Summarize: Pay Attention…

```
public void onClick(View v) {
  new DownloadImageTask().execute("http://example.com/image.png");
}

private class DownloadImageTask extends AsyncTask {
     protected Bitmap doInBackground(String... urls) {
         return loadImageFromNetwork(urls[0]);
     }

     protected void onPostExecute(Bitmap result) {
         mImageView.setImageBitmap(result);
     }
 }
```

Broadcasts vs Implicit Intents

Broadcasts are messages ***sent by Android system*** and other Android apps, when an event of interest occurs

Broadcasts are wrapped into an Intent object. This Intent object contains event details such as, android.intent.action.HEADSET_PLUG, sent when a wired headset is plugged or unplugged

Types of broadcast:

- System broadcast
- Custom broadcast

System Broadcasts

System broadcast are messages ***sent by the Android system, when a system event occurs***, that might affect your app. Few examples:

- An Intent with action, ACTION_BOOT_COMPLETED is broadcasted when the device boots
- An Intent with action, ACTION_POWER_CONNECTED is broadcasted when the device is connected to the external power

Custom Broadcasts

Custom broadcasts are broadcasts that your app sends out, similar to the Android system. For example, when you want to let other app(s) know that some data has been downloaded by your app, and its available for their use

Android provides three ways for sending a broadcast:

- Ordered broadcast
- Normal broadcast
- Local broadcast

Ordered Broadcast

- Ordered broadcast is delivered to ***one receiver at a time***
- To send an ordered broadcast, sendOrderedBroadcast()
- Receivers can propagate result to the next receiver or even abort the broadcast
- Control the broadcast order with android:priority attribute in the manifest file
- Receivers with same priority run in arbitrary order

Normal Broadcast

- Delivered to ***all the registered receivers*** at the same time, ***in an undefined order***
- Most efficient way to send a broadcast
- Receivers cannot propagate the results among themselves, and they cannot abort the broadcast
- The sendBroadcast() method is used to send a normal broadcast

Local Broadcast

- Sends broadcasts to ***receivers within your app***
- No security issues since no interprocess communication
- To send a local broadcast:
  - get an instance of LocalBroadcastManager
  - call **`sendBroadcast()`** on the instance

```
LocalBroadcastManager.getInstance(this).
        sendBroadcast(customBroadcastIntent);
```

Custom Broadcasts

- Sender and receiver must agree on unique name for intent (action name)
- Define in activity and broadcast receiver

```
private static final String ACTION_CUSTOM_BROADCAST =

"com.example.android.powerreceiver.ACTION_CUSTOM_BROADCAST";
```

What is a broadcast receiver?

- Broadcast receivers are app components
- They register for various system broadcast and or custom broadcast
- ***They are notified (via an Intent):***
  - By the system, when a system event occurs that your app is registered for
  - By another app, including your own if registered for that custom event

Broadcast receivers can be registered in two ways:

- Static receivers

  Registered in manifest, also called as Manifest-declared receivers
- Dynamic receivers

  Registered using app or activities' context in Java files

To create a broadcast receiver

- Subclass the BroadcastReceiver class and override its onReceive() method
- Register the broadcast receiver and specify the intent-filters:
  - Statically, in the Manifest
  - Dynamically, with registerReceiver()

Intent-filters specify the types of intents a broadcast receiver can receive; they filter the incoming intents based on the Intent values like action

To add an intent-filter:

- To your AndroidManifest.xml file, use <intent-filter> tag
- To your Java file use the IntentFilter object

Register statically in Android manifest

- <receiver> element inside <application> tag
- <intent-filter> registers receiver for specific intents

```
<receiver
    android:name=".CustomReceiver"
    android:enabled="true"
    android:exported="true">
    <intent-filter>
       <action
android:name="android.intent.action.BOOT_COMPLETED"/>
    </intent-filter>
</receiver>
```

Register dynamically

- Register your receiver in onCreate() or onResume()

```
// Register the receiver using the activity context
this.registerReceiver(mReceiver, filter);
```

- Unregister in onDestroy() or onPause()

```
// Unregister the receiver
this.unregisterReceiver(mReceiver);
```

Restricting broadcasts

- ***Restricting your broadcast is strongly recommended***
- An unrestricted broadcast can pose a security threat
  - For example: If your apps' broadcast is not restricted and includes sensitive information, an app that contains malware could register and receive your data

- If possible, ***use a LocalBroadcastManager, which keeps the data inside your app,*** avoiding security leaks
- ***Use the setPackage() method*** and pass in the package name. Your broadcast is restricted to apps that match the specified package name
- Access permissions can be enforced by sender or receiver

Enforce permissions by sender

To enforce a permission when sending a broadcast:

- Supply a non-null permission argument to sendBroadcast()
- Only receivers that request this permission using the <uses-permission> tag in their AndroidManifest.xml file can receive the broadcast

Enforce permissions by receiver

To enforce a permission when receiving a broadcast:

- If you register your receiver dynamically, supply a non-null permission to registerReceiver()
- If you register your receiver statically, use the android:permission attribute inside the <receiver> tag in your AndroidManifest.xml

References to additional material

- BroadcastReceiver Reference
- Intents and Intent Filters Guide
- LocalBroadcastManager Reference
- Broadcasts overview

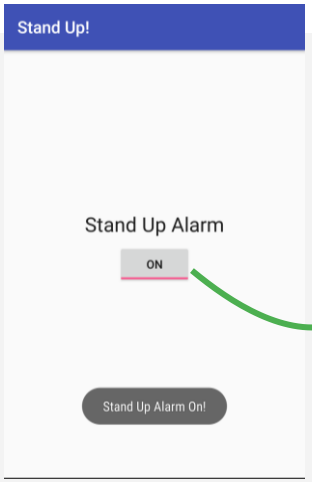
Alarms in Android

- Not an actual alarm clock
- ***Schedules something to happen*** at a set time
- Fire intents at set times or intervals
- Goes off ***once or recurring***
- Can be based on a real-time clock or elapsed time
- ***App does not need to run for alarm to be active***

Benefits:

- Device does not have to be awake
- Does not use resources until it goes off
- Use with BroadcastReceiver to start services and other operations

How alarms work with Android components

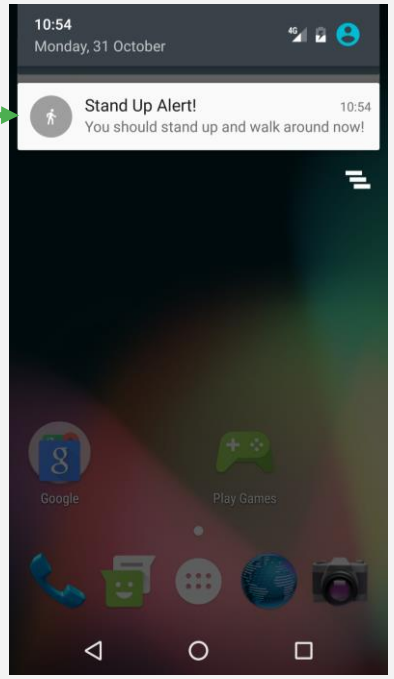


`Activity` creates a notification and sets an alarm

Alarm triggers and sends out `Intent`.

App may be destroyed so…

`BroadcastReceiver` wakes up the app and delivers the notification



Types of Android Alarms

	Elapsed Real Time (ERT)—since system boot	**Real Time Clock (RTC)—time of day matters**
Do not wake up device	`ELAPSED_REALTIME`	`RTC`
Wake up	`ELAPSED_REALTIME_WAKEUP`	`RTC_WAKEUP`

- Wakes up device CPU if screen is off
  - Used only for time critical operations; can drain battery
- Does not wake up device
  - Fires next time device is awake

Best practices…

Imagine an app with millions of users:

- Server sync operation based on clock time
- Every instance of app syncs at 11:00 p.m

Load on the server could result in high latency or even "denial of service"

- Add ***randomness*** to network requests on alarms
- Minimize alarm frequency
- Use ELAPSED_REALTIME, not clock time

Best practices…

Battery

- Minimize waking up the device
- Use inexact alarms
  - Android synchronizes multiple inexact repeating alarms and fires them at the same time
  - Reduces the drain on the battery
  - Use setInexactRepeating() instead of setRepeating()

Not using an alarm

- Ticks, timeouts, and while app is running—***Handler***
- Server sync—***SyncAdapter*** with Cloud Messaging Service
- Inexact time and resource efficiency—***JobScheduler***

AlarmManager

- AlarmManager provides access to system alarm services
- Schedules future operation
- When alarm goes off, registered Intent is broadcast
- Alarms are retained while device is asleep
- Firing alarms can wake device

```
AlarmManager alarmManager =
    (AlarmManager) getSystemService(ALARM_SERVICE);
```

How to schedule an alarm

- Type of alarm
- Time to trigger
- Interval for repeating alarms
- **`PendingIntent`** to deliver at the specified time

  (just like notifications)

What you need to to schedule an alarm

Schedule a single alarm:

- set()—single, inexact alarm
- setWindow()—single inexact alarm in window of time
- setExact()—single exact alarm

More power saving options AlarmManager API 23+

Schedule a repeating alarm:

- setInexactRepeating() - repeating, inexact alarm
- setRepeating()

Prior to API 19, creates a repeating, exact alarm

After API 19, same as setInexactRepeating()

User visible alarms

- setAlarmClock()
- System UI may display time/icon
- Precise
- Works when device is idle
- App can retrieve next alarm with `getNextAlarmClock()`

Supported starting from API 21

“Traditional” Sockets

- Classical Socket class available also in Android

  https://developer.android.com/reference/java/net/Socket

```
public void setPerformancePreferences (
        int connectionTime, int latency, int bandwidth)
```

- Classical ServerSocket class available also in Android

  https://developer.android.com/reference/java/net/ServerSocket

Points of attention:

- Sockets and ***MainThread***
- Sockets typically require the adoption of a ***multi-threading option*** (see previous slides)

Internet Connection

- Add permissions to Android Manifest
- ***Check Network Connection***
- ***Create Worker Thread***
- ***Implement background task***
  - Create URI
  - Make HTTP Connection
  - Connect and GET Data
- Process results
  - Parse Results

Internet permission

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

Check Network State permission

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

Network Management Information

- ConnectivityManager
  - Answers queries about state of network connectivity
  - Notifies applications when network connectivity changes
- NetworkInfo
  - Describes status of a network interface of a given type
  - Mobile or Wi-Fi

Network Check

```
ConnectivityManager connMgr = (ConnectivityManager)
            getSystemService(Context.CONNECTIVITY_SERVICE);

NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();

if (networkInfo != null && networkInfo.isConnected()) {
    // Create background thread to connect and get data
    new DownloadWebpageTask().execute(stringUrl);
} else { textView.setText("No network connection available."); }
```

```
NetworkInfo networkInfo =
    connMgr.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
boolean isWifiConn = networkInfo.isConnected();

networkInfo =
    connMgr.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);
boolean isMobileConn = networkInfo.isConnected();
```

Which async mode?

- AsyncTask—very short task, or no result returned to UI
- AsyncTaskLoader—for longer tasks, returns result to UI
- Background Service

In the background task (for example in doInBackground())

- Create URI
- Make HTTP Connection
- Download Data

Make a connection from scratch

- Use HttpURLConnection
- Must be done on a separate thread
- Requires InputStreams and try/catch blocks

```
HttpURLConnection conn =
   (HttpURLConnection) requestURL.openConnection();
conn.setReadTimeout(10000 /* milliseconds */);
conn.setConnectTimeout(15000 /* milliseconds */);
conn.setRequestMethod("GET");
conn.setDoInput(true);
```

Make a connection from scratch

```
conn.connect();
int response = conn.getResponseCode();

InputStream is = conn.getInputStream();
String contentAsString = convertIsToString(is, len);
return contentAsString;
```

Make a connection using libraries

- Use a third party library like OkHttp or Volley
- Can be called on the main thread
- Much less code

Volley

```
RequestQueue queue = Volley.newRequestQueue(this);
String url ="http://www.google.com";

StringRequest stringRequest = new
StringRequest(Request.Method.GET, url,
            new Response.Listener<String>() {
    @Override
    public void onResponse(String response) {
        // Do something with response
    }
}, new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {}
});
queue.add(stringRequest);
```

OkHttp

```
OkHttpClient client = new OkHttpClient();
Request request = new Request.Builder()

.url("http://publicobject.com/helloworld.txt").build();
client.newCall(request).enqueue(new Callback() {
    @Override
    public void onResponse(Call call, final Response
response)
        throws IOException {
        try {
            String responseData =
response.body().string();
            JSONObject json = new
JSONObject(responseData);
            final String owner = json.getString("name");
        } catch (JSONException e) {}
    }
});
```

Parsing the results

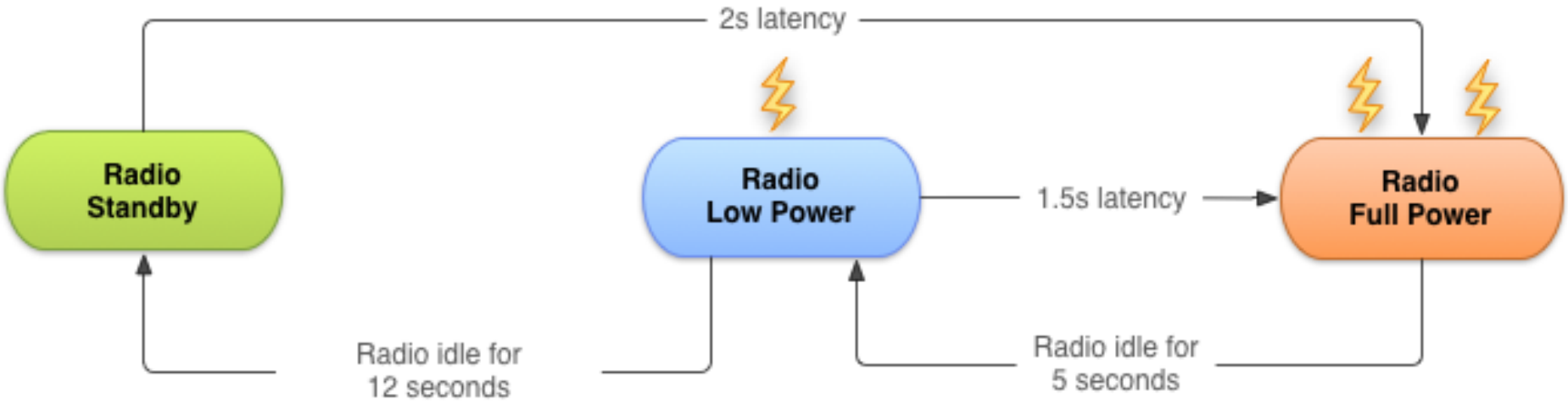
- Implement method to receive and handle results (onPostExecute())
- Response is often JSON or XML
- Parse results using helper classes
- JSONObject, JSONArray
- XMLPullParser—parses XML

Additional material:

- Connect to the Network Guide
- Managing Network Usage Guide
- HttpURLConnection reference
- ConnectivityManager reference
- InputStream reference

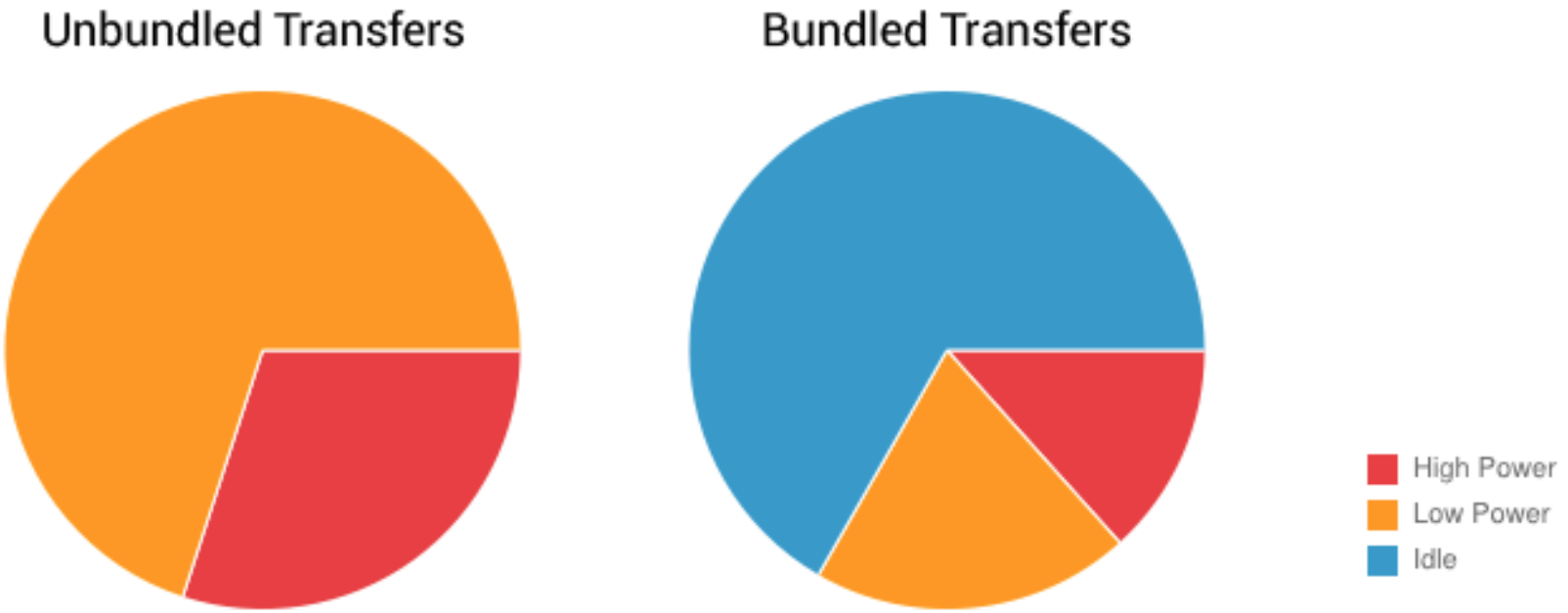
Efficient Data Transfer

- Full power—Active connection, highest rate data transfer
- Low power—Intermediate state that uses 50% less power
- Standby—Minimal energy, no active network connection



Bundle network transfers

- For a typical 3G device, every data transfer session, the radio draws energy for almost 20s
  - Send data for 1s every 18s—radio mostly on full power
  - Send data in bundles of 3s—radio mostly idle
- ***Bundle your data transfers***



Prefetch data

- ***Download all the data*** you are likely to need for a given time period ***in a single burst***, over a single connection, ***at full capacity***
- If guess is right, it reduces battery cost and latency
- If wrong, it may use more battery and data bandwidth

Monitor connectivity & battery

- Use ConnectivityManager to determine which radio is active and adapt your strategy
- Wait for specific conditions to initiate battery intensive operation
- BatteryManager broadcasts all battery and charging details in a broadcast Intent
- Use a BroadcastReceiver registered for battery status actions

Job Scheduler

- Used for ***intelligent scheduling of background tasks***
- Based on conditions, not a time schedule
- Much more efficient than AlarmManager
- ***Batches tasks together to minimize battery drain***

API 21+ (not in support library)

- JobService—Service class where task is initiated
- JobInfo—Builder pattern to set conditions for task
- JobScheduler—Schedule and cancel tasks, launch service

JobService

- JobService subclass
- Override
  - onStartJob()
  - onStopJob()
- ***Runs on main thread***

onStartJob()

- Called by system when conditions are met
- Runs on main thread
- ***Off-load heavy work to another thread***

onStartJob() returns a boolean

FALSE—Job finished

TRUE

- Work has been offloaded
- Must call **jobFinished()** from worker thread
- Pass in JobParams object from onStartJob()

onStopJob()

- Called if system has determined execution of job must stop… because requirements specified no longer met
  - For example, no longer on Wi-Fi, device not idle anymore
- Before jobFinished(JobParameters, boolean)
- Return TRUE to reschedule

```
public class MyJobService extends JobService {
    private UpdateAppsAsyncTask updateTask = new
UpdateAppsAsyncTask();
    @Override
    public boolean onStartJob(JobParameters params) {
        updateTask.execute(params);
        return true; // work has been offloaded
    }
   @Override
   public boolean onStopJob(JobParameters jobParameters) {
      return true;
   }    }
```

JobInfo

- Set conditions of execution
- JobInfo.Builder object

Arg 1: Job ID
Arg 2: Service component
Arg 3: JobService to launch

```
JobInfo.Builder builder = new JobInfo.Builder(
        JOB_ID,
        new ComponentName(getPackageName(),
        NotificationJobService.class.getName()));
```

Setting conditions

setRequiredNetworkType(int networkType)

setBackoffCriteria(long initialBackoffMillis, int backoffPolicy)

setMinimumLatency(long minLatencyMillis)

setOverrideDeadline(long maxExecutionDelayMillis)

setPeriodic(long intervalMillis)

setPersisted(boolean isPersisted)

setRequiresCharging(boolean requiresCharging)

setRequiresDeviceIdle(boolean requiresDeviceIdle)

Setting conditions

setRequiredNetworkType(int networkType)

- NETWORK_TYPE_NONE—Default, no network required
- NETWORK_TYPE_ANY—Requires network connectivity
- NETWORK_TYPE_NOT_ROAMING—Requires network connectivity that is not roaming
- NETWORK_TYPE_UNMETERED—Requires network connectivity that is unmetered

setOverrideDeadline(long maxExecutionDelayMillis)

- Maximum ms to wait before running the task, even if other conditions are not met

Setting conditions

setPeriodic(long intervalMillis)

- Repeats task after a certain amount of time
- Pass in repetition interval
- ***Mutually exclusive with minimum latency and override deadline conditions***
- Task is not guaranteed to run in the given period

setPersisted(boolean isPersisted)

- Sets whether the job is persisted across system reboots
- Pass in True or False
- Requires RECEIVE_BOOT_COMPLETED permission

Setting conditions

setRequiresCharging(boolean requiresCharging)

- Whether device must be plugged in
- Pass in True or False
- Defaults to False

setRequiresDeviceIdle(boolean requiresDeviceIdle)

- Whether device must be in idle mode
- Idle mode is a loose definition by the system, when device is not in use, and has not been for some time
- ***Use for resource-heavy jobs***
- Pass in True or False. Defaults to False

JobInfo code

```
JobInfo.Builder builder = new JobInfo.Builder(
        JOB_ID, new ComponentName(getPackageName(),
        NotificationJobService.class.getName()))
       .setRequiredNetworkType(JobInfo.NETWORK_TYPE_UNMETERED)
       .setRequiresDeviceIdle(true)
       .setRequiresCharging(true);
JobInfo myJobInfo = builder.build();
```

Scheduling the job

- Obtain a JobScheduler object from system
- Call schedule() on JobScheduler, with JobInfo object

```
mScheduler =   (JobScheduler)
      getSystemService(JOB_SCHEDULER_SERVICE);
mScheduler.schedule(myJobInfo);
```

References to additional material

- Transferring Data Without Draining the Battery Guide
- Optimizing Downloads for Efficient Network Access Guide
- Modifying your Download Patterns Based on the Connectivity Type Guide
- JobScheduler Reference
- JobService Reference
- JobInfo Reference
- JobInfo.Builder Reference
- JobParameters Reference
- Presentation on Scheduling Tasks

Example: Developing a Launcher for Replacing the usual Home

As already mentioned, ***home*** allows starting any app, serving as a ***launcher***
Android as an open platform – *"All Apps are created equal"* → it is possible to customize the system with our own home

A custom home can be created by:

1. Defining a ***new Activity***
2. ***Declaring the launcher's action MAIN in the Intent Filter***
3. Searching installed apps via Package Manager
4. Selecting the only apps that have an ***Intent Filter related to launch***
5. Creating a selection View (Button) for each suitable app
6. Defining the click event: Intent notification
7. Association of the event to the View

Example: Developing a Launcher for Replacing the usual Home

```
// Step 1: definition of a new activity
public class CategoryTestActivity extends Activity {
   public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        LinearLayout activitiesList;
        activitiesList = (LinearLayout)
                findViewById(R.id.activitiesList);

// Steps 3 and 4: app search and selection based on IntentFilter
        Intent intent = new Intent();
        intent.setAction(Intent.ACTION_MAIN);
        intent.addCategory(Intent.CATEGORY_LAUNCHER);
        PackageManager pkgManager = getPackageManager();
        List<ResolveInfo> activities =
                pkgManager.queryIntentActivities(intent, 0);
        …
```

Example: Developing a Launcher for Replacing the usual Home

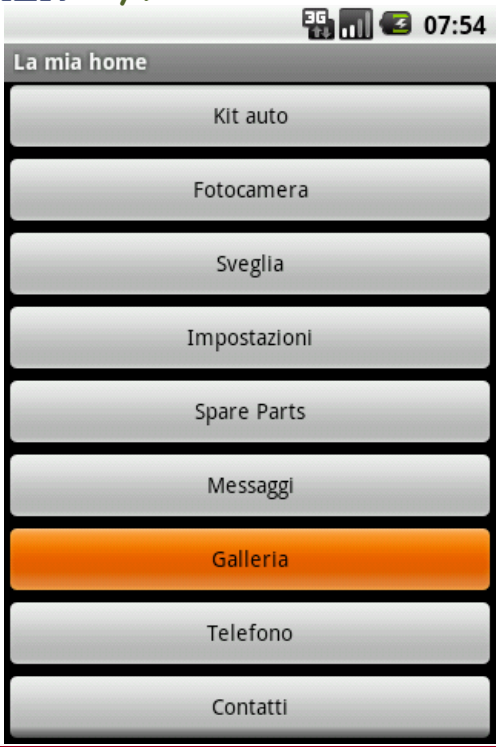
```
…
for (ResolveInfo resolveInfo : activities) {
    final ResolveInfo ri = resolveInfo;
    Button button = new Button(this); //Step 5: one button per app
    button.setText(resolveInfo.loadLabel(pkgManager));
    // Step 7: event-view association
    button.setOnClickListener(new OnClickListener() {

            public void onClick(View v) {
            Intent intent = new Intent(); //Step 6: intent launch
            ComponentName cn = new ComponentName(ri.
            activityInfo.packageName, ri.activityInfo.name);
            intent.setComponent(cn);
            intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
            intent.addFlags(Intent.FLAG_ACTIVITY_MULTIPLE_TASK);
            startActivity(intent);}});
    activitiesList.addView(button);}}}
```

Example: Developing a Launcher for Replacing the usual Home

```
<manifest xmlns:android="..." package="it.mypackage">
<application android:label="@string/app_name">
<activity android:name=".HomeActivity"
        android:label="@string/app_name">

<intent-filter> <!- Step 2 -->
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
<category android:name="android.intent.category.
        HOME"> </category>
<category android:name="android.intent.category.
        DEFAULT"></category>
</intent-filter>
</activity>
</application>
</manifest>
```



Android Security in a Single Slide

Typical guidelines for security in Linux

- At default any Android app runs in a dedicated Dalvik VM and within its ***own separated process***. Which UID/GID?
  - Process and group identifiers are selected from an interval that is defined system-level: **`FIRST_APPLICATION_UID`**, **`LAST_APPLICATION_UID`**
- Process-level permissions are assigned and controlled ***depending on user ID & group ID*** assigned to processes. *Typically what do you expect?*
- Finer-grained permissions are assigned (and revocable) even for a single operation via manifest file

```
<manifest xmlns:android="http://schemas.android.com/apk/res/
       android" package="com.google.android.app.myapp" >
<uses-permission id="android.permission.RECEIVE_SMS" />
</manifest>
```

Exercise on Android

To design and develop a simple ***context-dependent Android app*** (for example, able to play different audio/video files depending on current user location), by employing the APIs of the ***Media Framework library*** (native) and the APIs of the ***Location Manager*** (Java-based, application framework level)

Suggestion to use, but free personal choice ☺:

- ***Android Studio and SDK tools***
  https://developer.android.com/studio/index.html

It could be the initial starting seed for a possible project activity…

As additional examples of Android usage and project activities, please see also
http://lia.disi.unibo.it/Courses/sm2122-info/esercizi.html

Many and Many Available Sources for Additional Info

Many good (not only good ☹) books on Android:

- E. Hellman, ***"Android Programming: Pushing the Limits"***, Wiley, Nov. 2013
- Z. Mednieks, L. Dornin, ***"Programming Android: Java Programming for the New Generation of Mobile Devices"***, O'Reilly, Ott. 2012
- R. Meier, ***"Professional Android 4 Application Dev."***, Wrox, 2012
- F. Ableson, R. Sen, ***"Android in Action"***, Manning, Feb. 2011

Android SDK has good manuals and how-tos, e.g., description of available APIs, presentation of examples of apps, …

- http://developer.android.com/sdk/index.html

Further info and documents are available at the Android Studio Web site:

- https://developer.android.com/studio/index.html

iOS (or iPhoneOS): a Very Rapid Overview

Approach is ***very similar to Android from many perspectives*** and in terms of realization of a ***wide support ecosystem, i.e., development model + support API***:

- iOS exploits a variant of the ***XNU kernel, which is at the basis of MacOSX***
- Chain of development tools is similarly based on ***Xcode***

SDK includes APIs at different layers to support:

- Events and multi-touch controls
- ***Accelerometer***
- Localization (i18n)
- Camera and media in general (audio mixing&recording, video playback, several image file formats, OpenGL ES …)
- Networking
- ***Embedded SQLite database***
- ***Core Location*** (GPS, Skyhook WiFi, …)
- ***Thread***
- ***Power management***
- File system
- Security

SDK also includes ***iPhone Simulator***, i.e., a tool to emulate iPhone look&feel over developer desktop. It is not a real emulator: it runs code that is actually generated for another target (x86)

SDK requires a Mac OS X Leopard host (or more recent)

iOS (or iPhoneOS): Rules Rules Rules…

- 3.3.1 — Applications may only use ***Documented APIs in the manner prescribed by Apple*** and ***must not use or call any private APIs***. Applications must be ***originally written in Objective-C, C, C++, or JavaScript*** as executed by the iOS WebKit engine, and only code written in C, C++, and Objective-C may compile and directly link against the Documented APIs
- 3.3.2 — ***An Application may not itself install or launch other executable code by any means***, including without limitation through the use of a plug-in architecture, calling other frameworks, other APIs or otherwise. ***No interpreted code may be downloaded*** or used in an Application except for code that is interpreted and run by Apple's Documented APIs and built-in interpreter(s)

Even the SDK can be ***downloaded for free*** but ***calls for registration to iPhone Developer Program*** if developers are willing to release some software (***paying a fee + subject to Apple's approval***)

Apple has not announced ***any plan for supporting Java*** over iPhones; initially some partial support instead for ***J2ME over iOS***

How to Develop in iOS: First Steps

To start:

- http://developer.apple.com/iphone/
- Download the iOS SDK, which includes:
  - Xcode
  - iPhone emulation tools
  - Monitoring tools
  - Interface builder

Please note that apps are ***subject to Apple's approval*** (included in the signed agreement for SDK download), with the idea of performing reliability tests and other investigations…

Apps may be rejected if evaluated as of "limited utility"

Alternative Possibilities for iOS Development

Primary option: ***exploitation of Xcode and Objective C***, exactly same way as in traditional MacOSX

Alternatively:

- ***Web applications*** that employ ***AJAX/Javascript*** technologies; ***HTML5 set of solutions***; possibility to access via Safari

- Also (but rare) usage of Java
  - AlcheMo for iPhone
  - Xmlvm
  - Java installation over “unlocked” and “jailbroken” iPhone

By the way, do you know what the terms “unlocking” and “jailbreaking” mean?

Unlocking & Jailbreaking

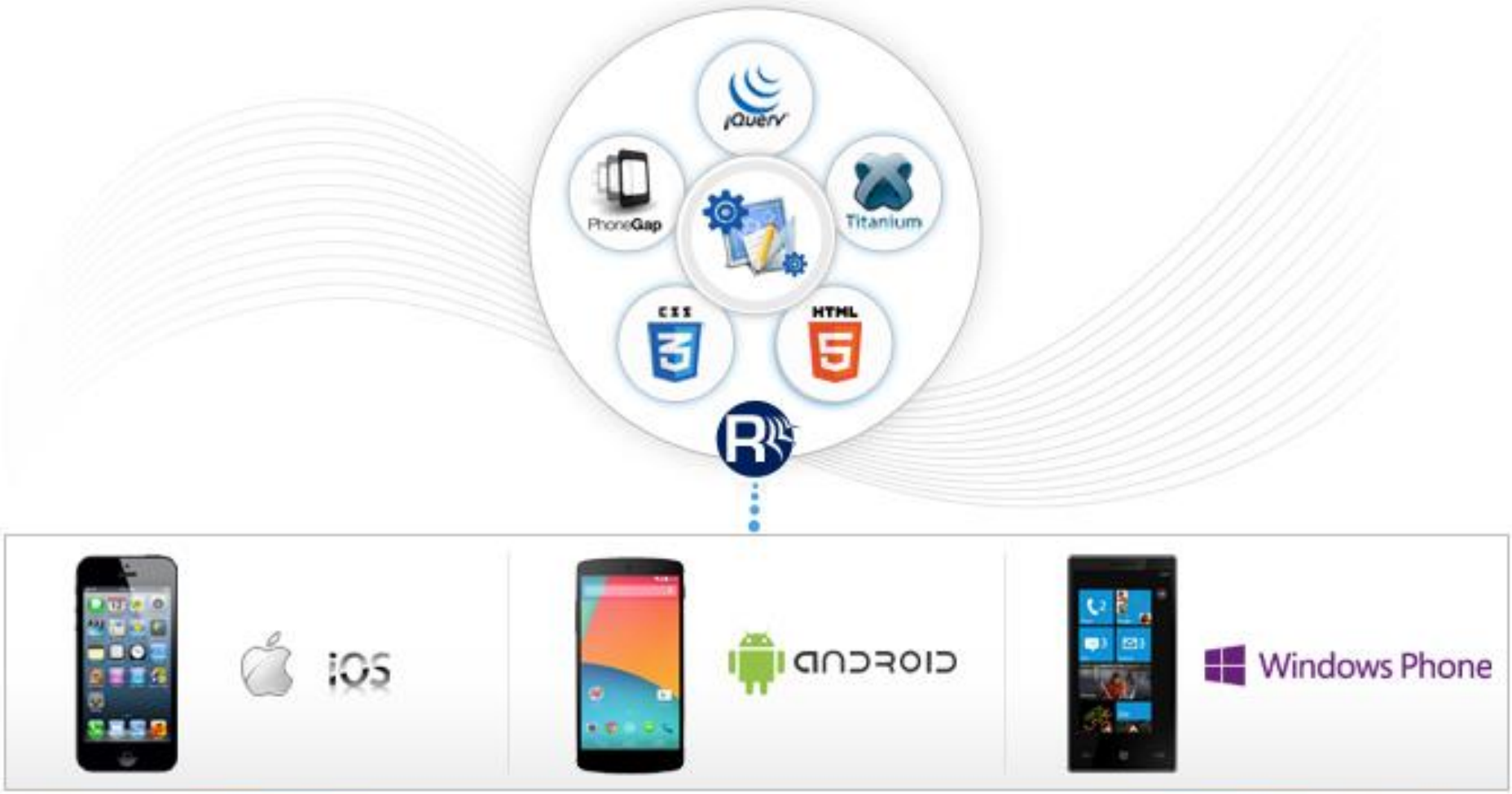
To be precise, ***unlocking and jailbreaking identify two different procedures***

- ***Unlocking*** is the process through which a device is ***made compatible with telco networks*** for which it was not specifically licensed (overcoming the locking with a given dedicated telco operator)
- ***Jailbreaking*** is the process through which a developer exits her own "jail" in UNIX-like OS and/or breaks the imposed system of Digital Right Management (DRM). ***It is a specific form of growth of execution privilege***
  - In iOS, it allows a user to ***execute arbitrary code and apps***, by bypassing the regular mechanism of distribution of Apple code (based on iTunes App Store and iTunes Application)

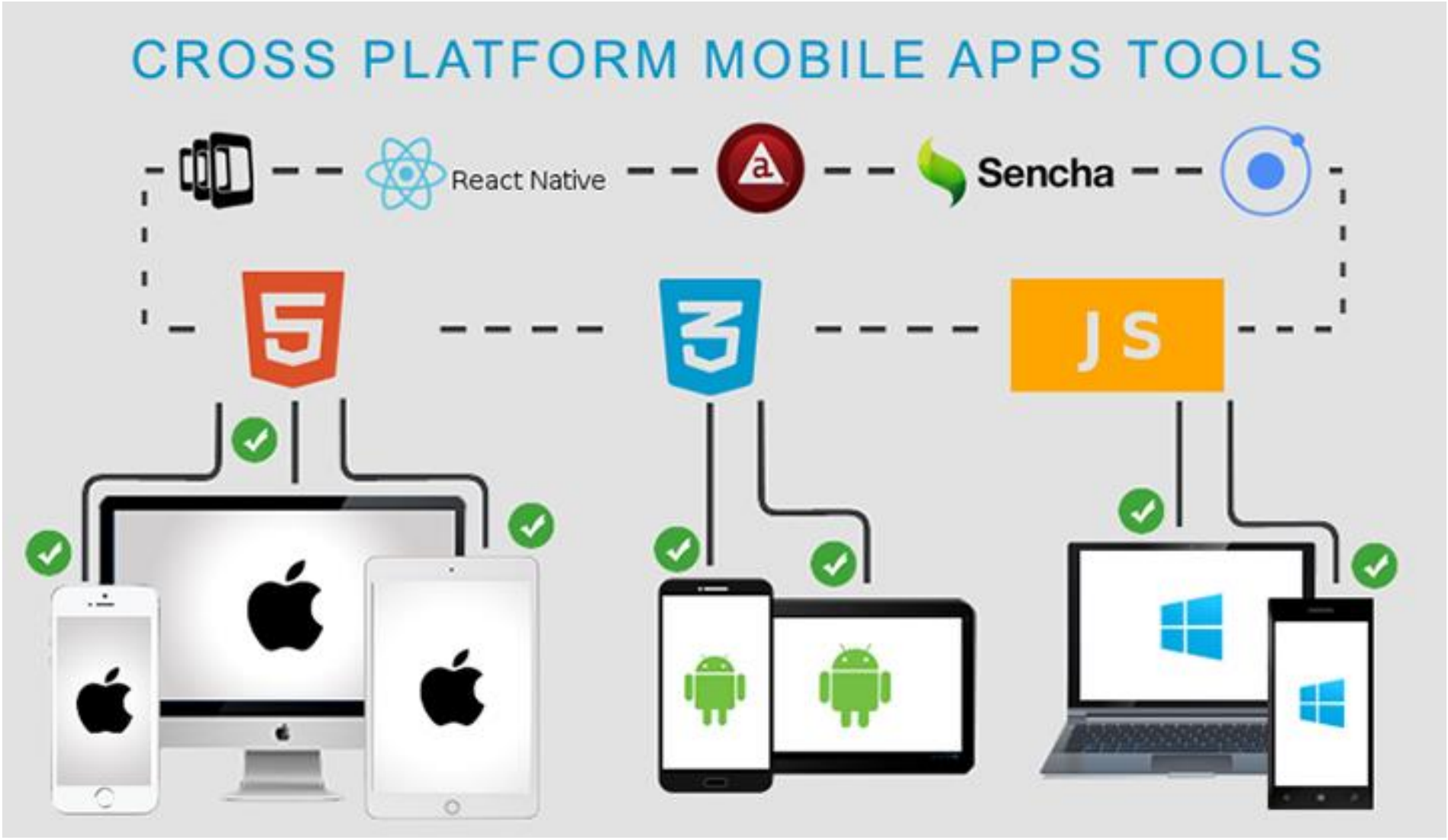
To enjoy yourself ☺ with experimentation, ***tools*** such as:

- PwnageTool, QuickPwn, Yellowsn0w http://blog.iphone-dev.org/
- Pusher - http://ripdev.com/pusher/
- Linux on iPhone - http://www.iphonelinux.org/index.php/Main_Page
- ZIPhone - http://www.ziphone.org/

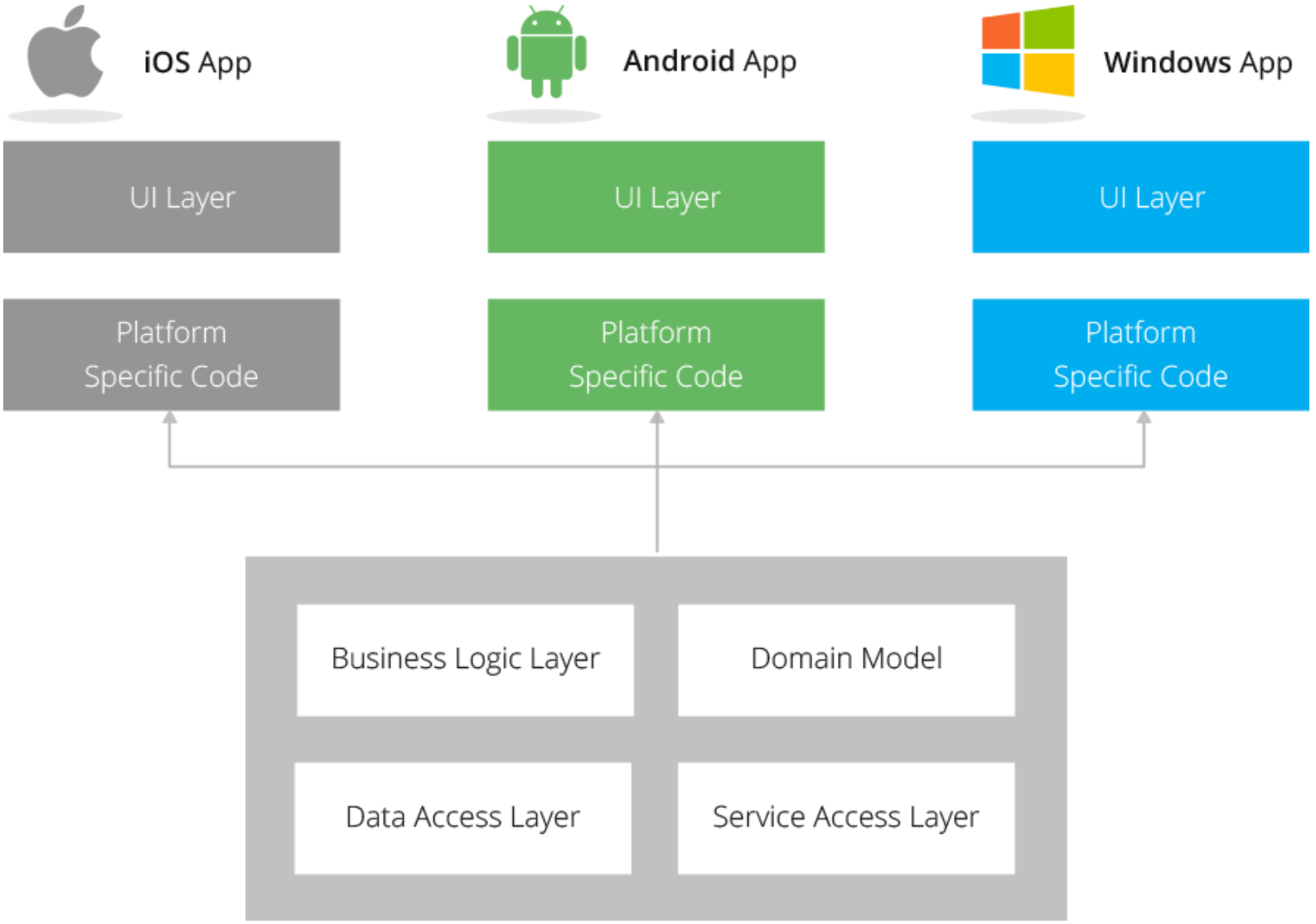
Cross platform based on HTML5



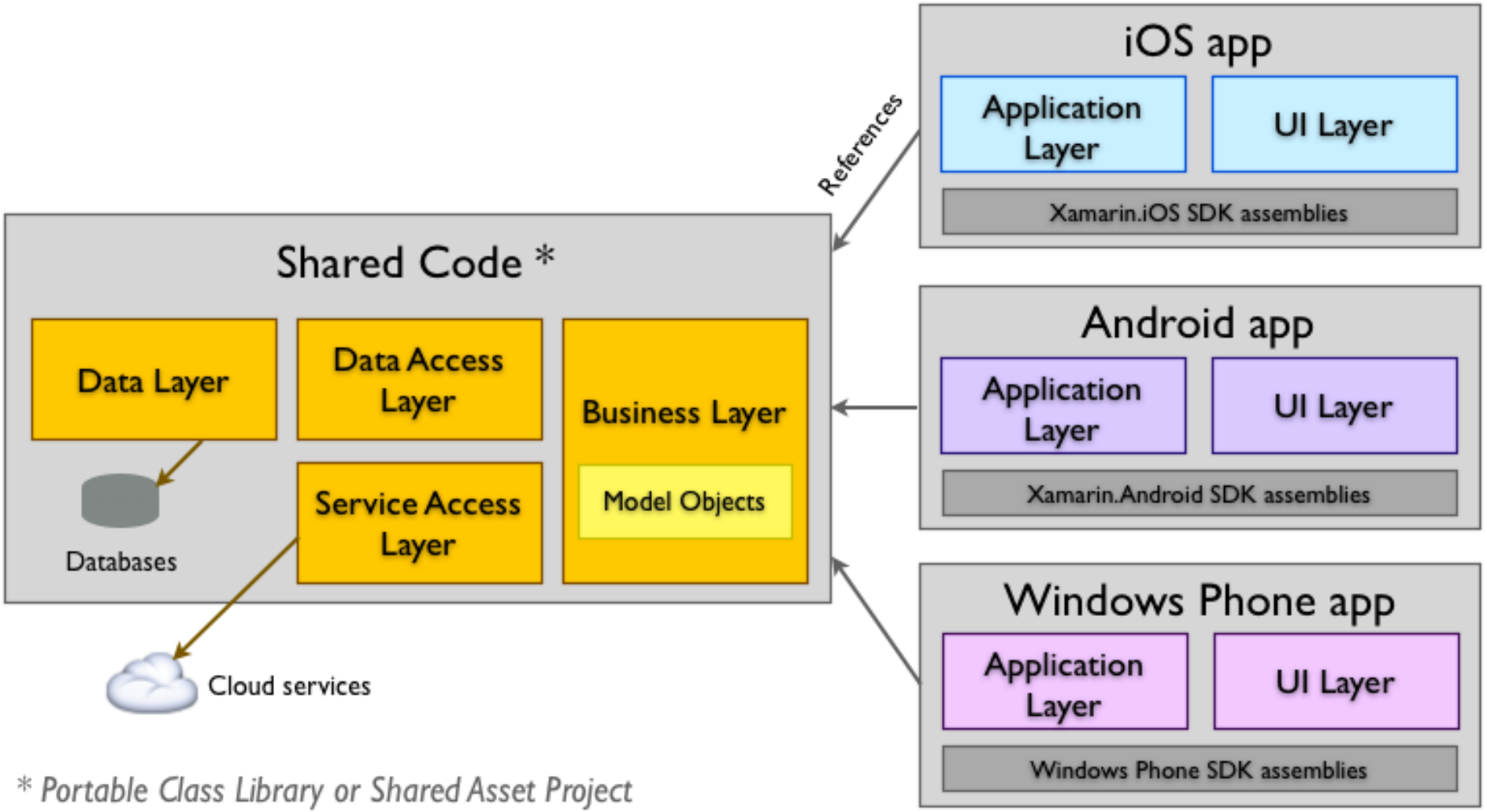
Cross platform based on HTML5



Cross platform - Xamarin



Cross platform - Xamarin



Web Applications: HTML5 in a Single Slide

Again, the fundamental question is pros&cons of

Web applications vs. native apps

HTML5

- Nothing of groundbreaking originality, it uses the traditional classical model of Web applications with rich interactivity
- ***HTML5 = HTML + CSS + JavaScript***
  - W3C finalized it (at the end :-)!) in October 2014

By delving into finer details:

- New tags for AJAX and DHTML
- New tags for embedded management of audio and video files (e.g., <video> tag)
  - To what extent are they currently supported? In a completely standard way?
- Better management of document structure

For example, see http://w3c.github.io/html/ (draft version 5.2 – April 18, 2017)

- http://slides.html5rocks.com/

Credits and Additional Material

Credits to ***Google Developer Training*** – Android Developer Fundamentals v2 for some material and pictures about Android programming (Creative Commons Attribution 4.0)

Additional (optional) material on the course Web site about:

- ❑ Activities and Intents
- ❑ Activity lifecycle and state
- ❑ Implicit Intents
- ❑ AsyncTask and AsyncTaskLoader
- ❑ Notifications
- ❑ Data Storage