Mobile Application Architectures

Lecture 4

Goal for today

- Understand existing solutions for mobile application architectures
- Architectures classification
- App development approaches
- Types of app clients
- Higher order architectures

Introduction

What IS a software architecture?

… "the organization or structure of a system, while the system represents a collection of components that accomplish a specific function or set of functions."

Introduction

An architecture is focused on organizing components to support specific functionality.

- Application architectures are often modeled to highlight or illustrate the overall layout of the software (e.g., application code and platform) and hardware (e.g., client, server, and network devices).
- While there are many possible combinations of software and hardware, application architectures often fall into a series of recognizable patterns.

Introduction

Mobile app architectures often do not fulfill the requirements of concrete tasks.

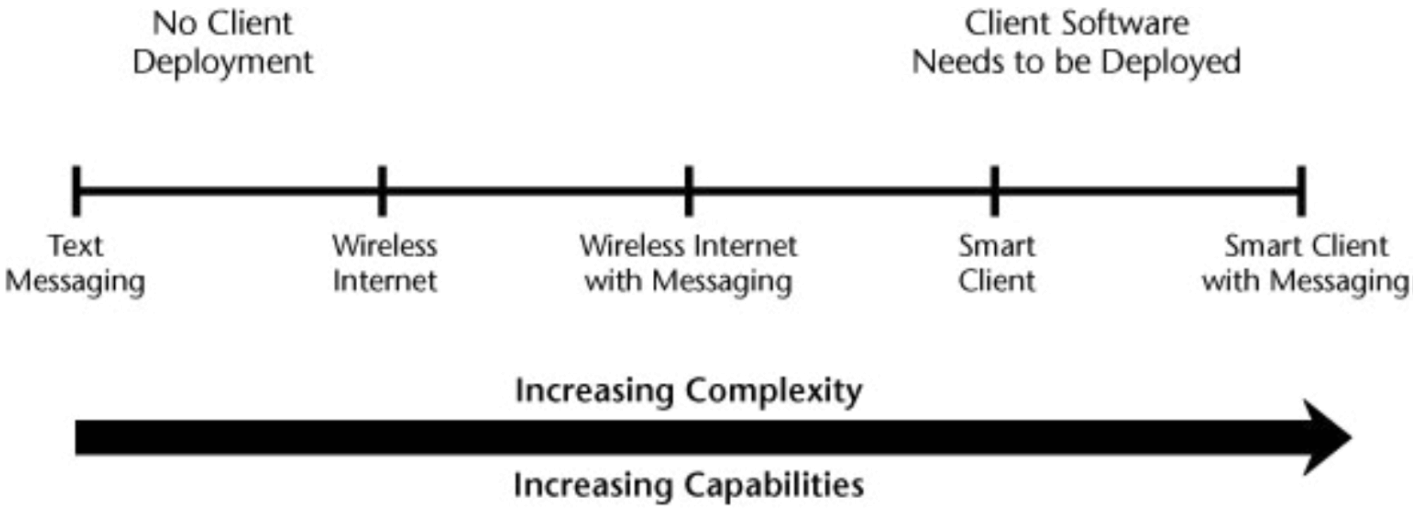
Elaborate trade-off between:

- Usability – *imposed by specific devices*
- Finalization time – *short vs. long term development*
- Portability (device types & OS) – *tablet, smartwatch also? multiple OS? Older OS? Newest release of OS?*
- Complexity – *WebView vs native app, e.g. Facebook <2012*
- Installation options – *dedicated/web app?*
- Performance – *power efficient vs. Pokemon Go*
- Cost
- Maintenance
- Access to resources – *server, local backup, offline capability?*

Classification criteria

Many types of architectures for mobile applications development

→ Classified according to the complexity of software modules that run on the mobile system



Architecture types

1. Messaging

   Data shipping

   (3G) SMS infrastructure

2. Thin client

   Both code and data are executed and stored on the server side

   E.g. accessed through internet browser

3. Rich client

   The application is stored on the device

   Data is not stored on the device

4. Thick client

   Both code and data are stored on the device

Architecture types

- Messaging – limited applicability
- Thin – web applications (HTML, XHTML, Flash Light), dependent on connectivity
- Rich – intermittent connectivity
- Thick – independent of connectivity

General Architecture of a Mobile Solution

A mobile application is built on several components:

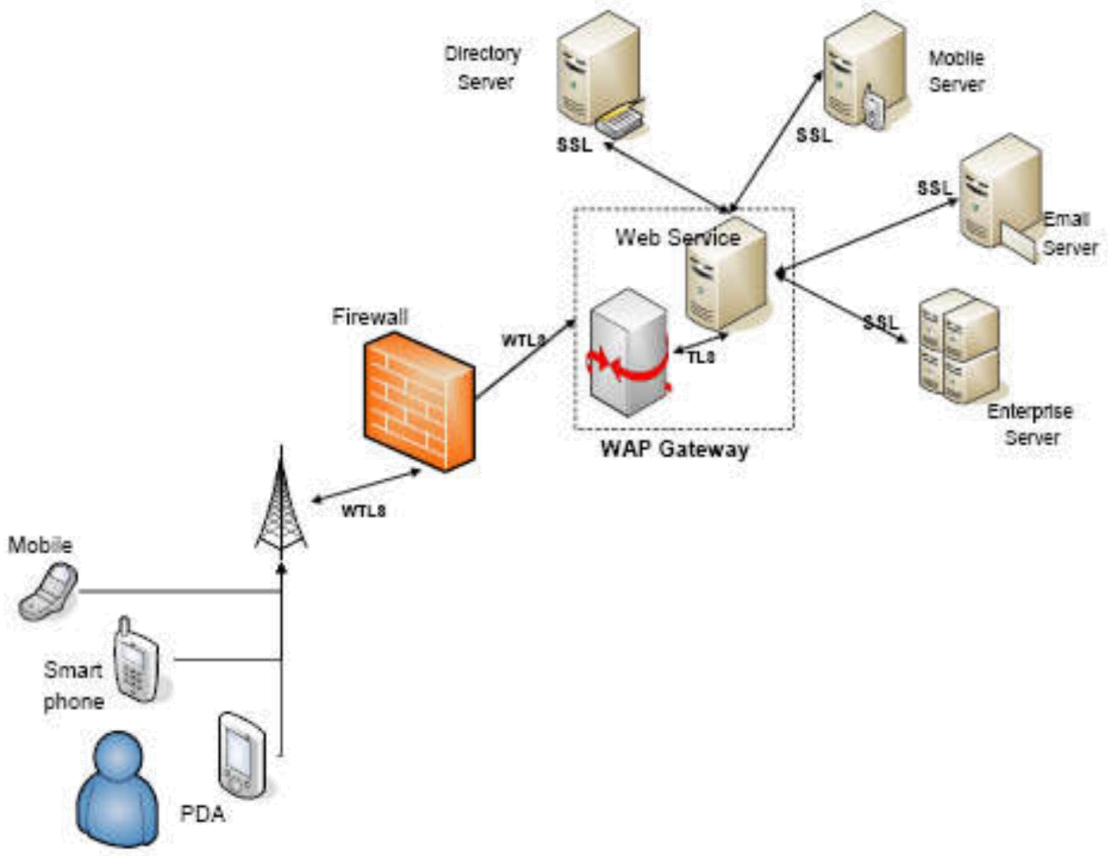
- Frontend: *web app, mobile app*
- Backend: *server, file server, database*

And layers:

- user interface
- application's logic
- data access

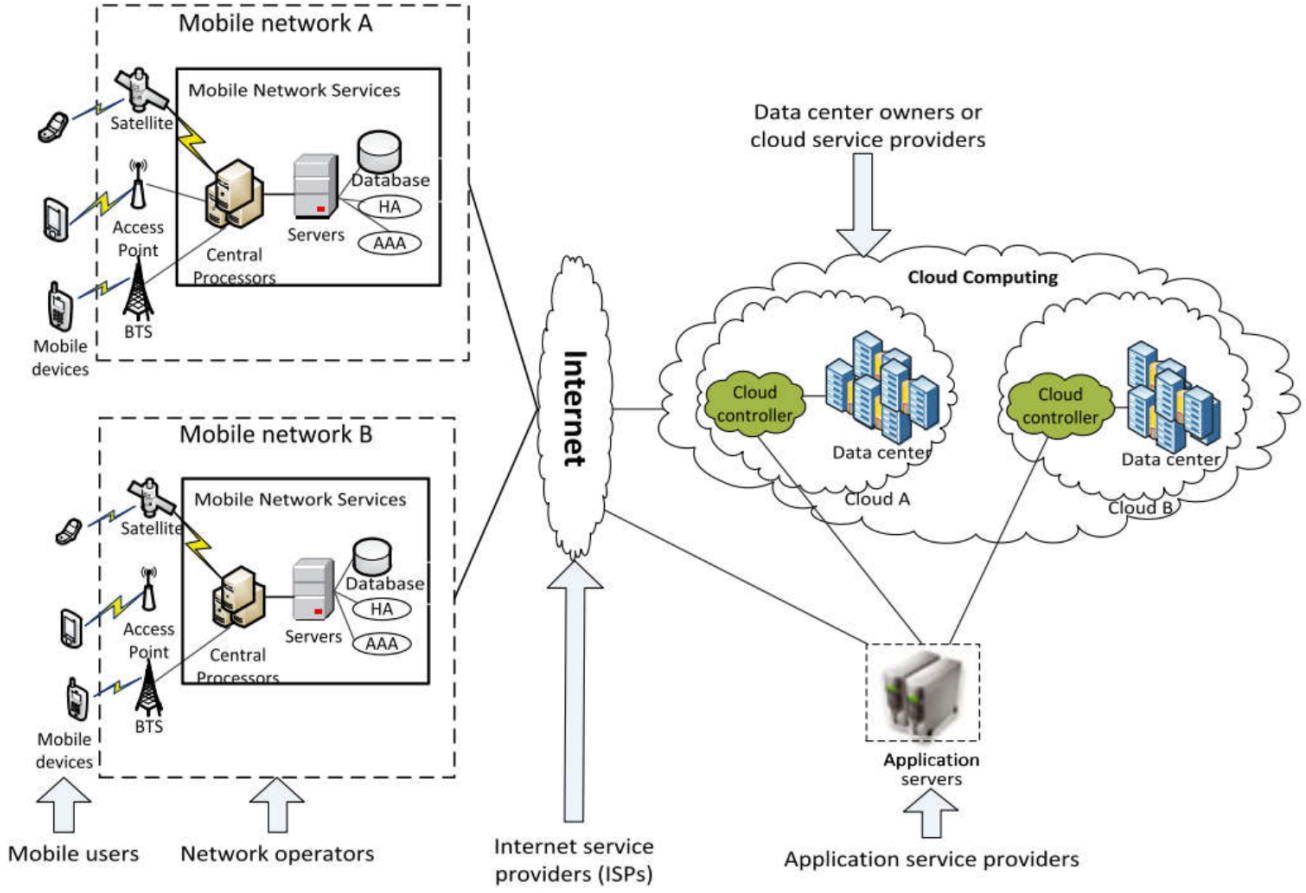
Simplified Architecture

Access local server directly



Actual Architecture

Managing global clients on cloud-replicated server apps



Messaging

SMS based text apps and services

→ Kronos, Dating, Games

+ Advantages

- Any mobile device
- Alerts
- Easy to install and manage

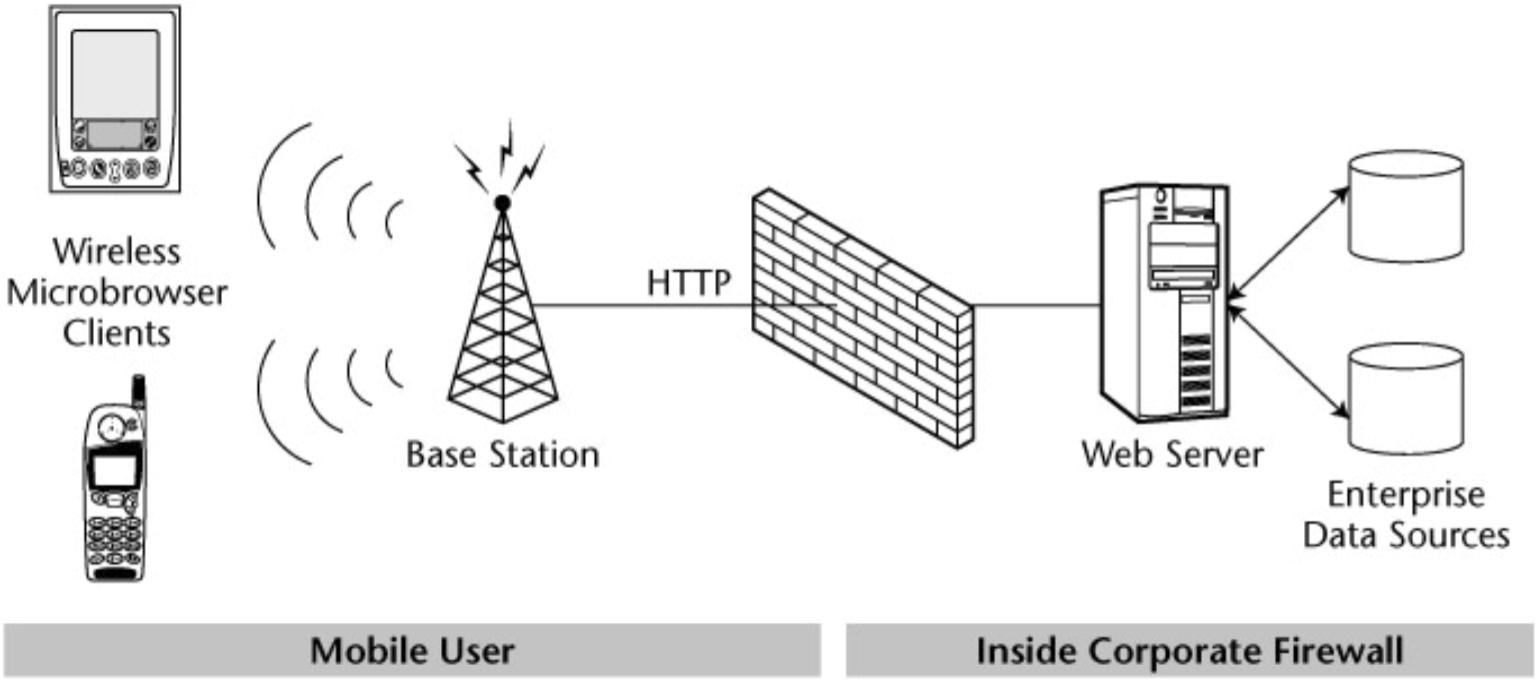
- Disadvantages

- Limited amount of data
- Limited to text
- Cost (at least international)

Thin clients (Internet clients)

User interface = micro web browser

The application code is running on the server, and data is stored on the server.



Thin clients (Internet clients)

The client application:

- On the mobile device, a web browser accesses the web pages content that are stored on a server.
- The device must have continuous connectivity to the network, otherwise the desired pages cannot be accessed or are retrieved from the cache.
- The protocols are usually based on IP protocol: HTTP, HTTPS.

Thin clients (Internet clients)

The server application:

- Listens to requests from client applications and provides the pages of information requested.
- Accesses the data to prepare information for customers.
- Analyzes the characteristics of the client application to provide data in a suitable form for it (e.g. based on Screen Size).

Thin clients (Internet clients)

The advantages of the model:

- ✓ Applications can be used without an installation on the client.
- ✓ Updating the application is done only on the server.
- ✓ Extend the internet applications model to the mobile devices.
- ✓ The user interface is familiar to most users.
- ✓ Integration with existing applications.
- ✓ Designed for a large amount of users.
- ✓ Data is updated in a centralized manner.
- ✓ Security – data stored on the server, not by the client.

Thin clients (Internet clients)

The disadvantages of the model:

- ✓ Applications require a continuous connection to the wireless network.
- ✓ The user interface is limited to data and simple elements.
- ✓ Performance is reduced due to the accesses to remote services.
- ✓ Testing applications under real conditions can be difficult.
- ✓ Availability – the application cannot be accessed if there is no connectivity or if the server is off.
- ✓ Security – wireless connection.
- ✓ Cost – continuous connecting to mobile networks costs.

Thin clients (Internet clients)

→Mobile web pages

→Mobile web widgets

→Mobile web applications (HTML5, CSS, JS)

In this approach, the application runs inside a mobile browser.

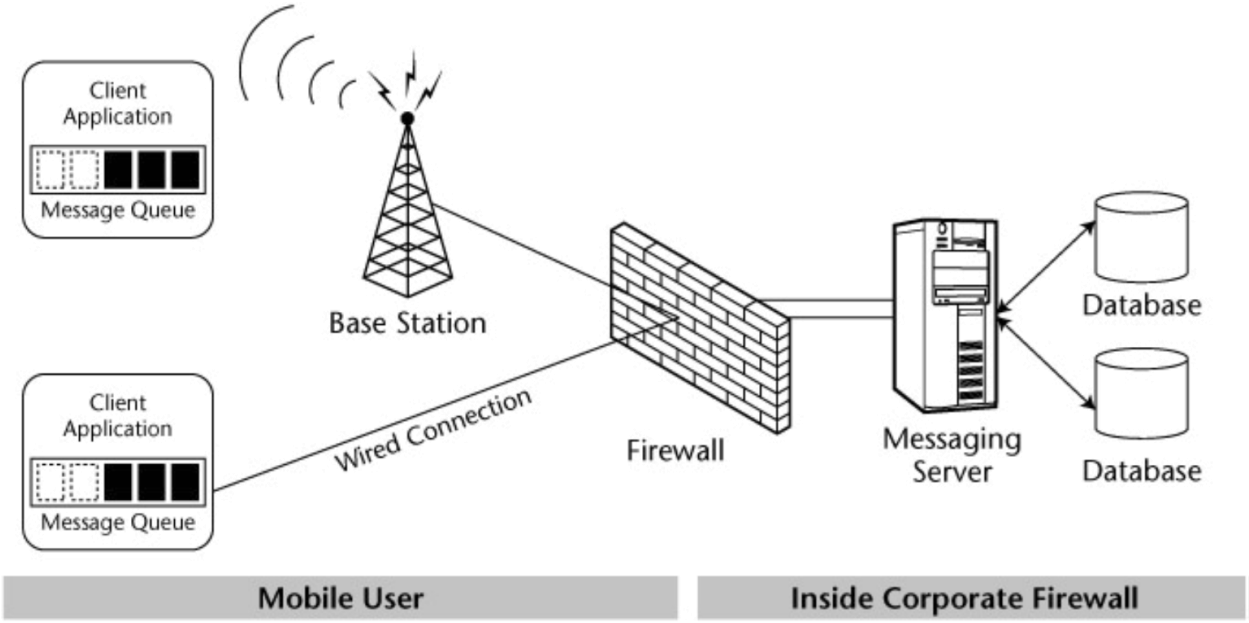
The browser only hosts the application's presentation layer that is designed using HTML5.

The interface typically looks and behaves like a traditional web site but is designed for the mobile device form factor.

Rich clients

Client/server mobile applications

Application layer protocol based on messages that support intermittent and limited connectivity.



Rich clients

Mobile–client system features:

- Implement application-level communication protocol
- Maintain a list of messages to be sent and then processed by the server when there is connectivity
- Store and forward communication
- Remote access to data, local data cache
- Complex operations can be sent for execution to the server

Rich clients

Server system features:

- Process the client messages
- Encapsulate the answers into messages
- Send response messages to the customer when there is a connectivity between them

Rich clients

The advantages of the model:

- ✓ Store-and-forward – communication model specific to environments with intermittent connectivity.
- ✓ Operations initiated by the server (sync, update, notification)
- ✓ Customized data transfer
- ✓ Radio (wireless) communication or wired communication

Rich clients

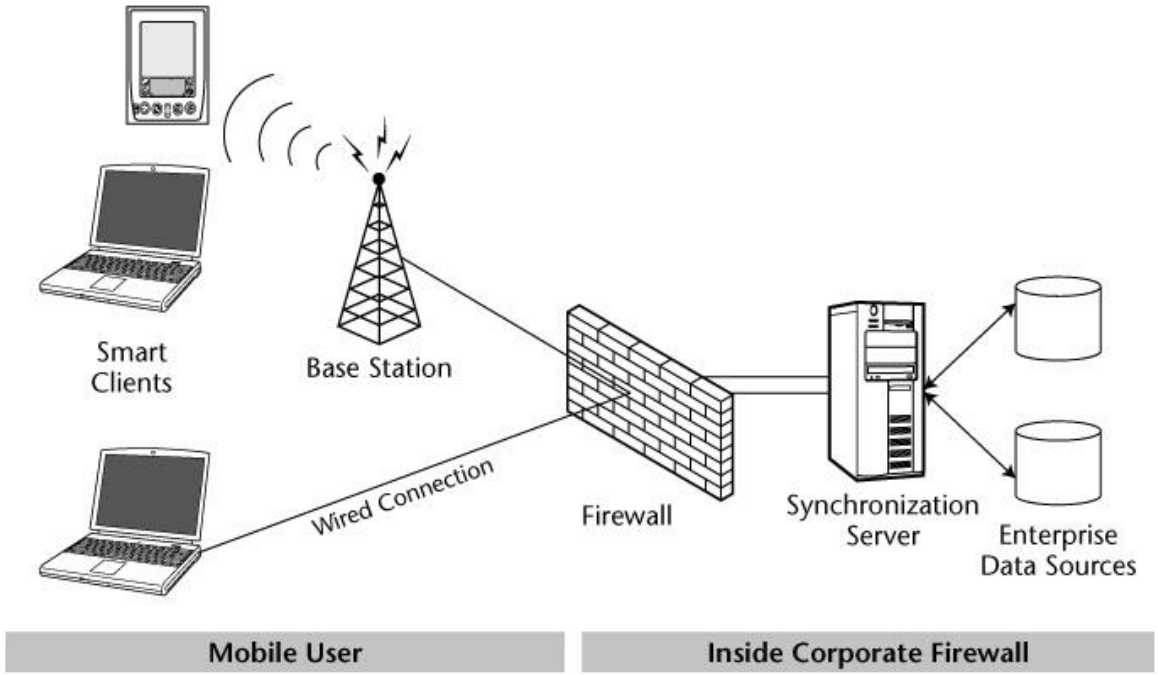
The disadvantages of the model:

- ✓ The cost of communication
- ✓ The energy used for communication
- ✓ The response latency (low bitrates, intermittent connectivity)

Thick clients

The application runs on a mobile, no matter the connectivity

The integration with other applications (enterprise) is done through synchronization



Thick clients

The device must have an OS that allows the installation of new applications.

→Code and data are installed and stored on the mobile device.

→Data can be synchronized with the persistence layer on the server through various communication techniques: wireless, *wired, USB*.

→The transfer and the installation of the applications on mobile devices has to be ensured.

Thick clients

The advantages of the model:

- ✓ The data is always available
- ✓ Enhanced graphical user interfaces
- ✓ Performance* (dependent on CPU, not on communication infrastructure)
- ✓ Distributed mobile computing
- ✓ Security / Data privacy
- ✓ Cost*

Thick clients

The disadvantages of the model:

- ✓ Integration with enterprise applications.
- ✓ The transfer, installation and configuration of the application on mobile devices.
- ✓ Cost and development time of mobile application.
- ✓ Multitude of HW, OS, SW platforms and different devices.

Smart clients: rich & thick

Apps pertaining to both categories are usually modeled through a client-server architecture.

→Mobile applications are client applications that use services provided by a server.

→Server applications implement services accessed by customers.

→Communication of request-response type

Choosing the right architecture

You'll often encounter terms like native app or web app, or even hybrid app. What's the difference?

Impacts:

- Development (language, technologies)
- Deployment (access on web or install from store)

Native apps (1/2)

They *live* on the device (icons on the home screen)

→Installed through an application store (Google Play, App Store).

→Developed specifically for one platform, and can take full advantage of all the device features.

Can use the camera, the GPS, the accelerometer, the compass, the list of contacts, and so on. They can also incorporate gestures (OS or custom).

→Native apps can use the device's notification system and can work offline.

Native apps (2/2)

They *live* on the device (icons on the home screen)

→The mobile application is custom built for the target device operating system with a compiled programming language and using the native SDK.

→Offers the native look and feel.

→Best performance.

→Great user experience.

Mobile web apps

Are not *real* applications: they are websites that look and feel like native applications, but are not implemented as such.

→They are run by a browser and typically written in HTML, JS.

→Users first access them as they would access any web page (URL, bookmark).

→HTML5 makes it possible to obtain native-like functionality in the browser.

→Today, the distinction between web apps and regular web pages has become blurry.

Mobile web apps

e.g. In 2011 Financial Times replaced its native iOS app with a web app (app.ft.com) to circumvent costs.

- Hard to distinguish from a native app.
- Users can swipe horizontally to move on to new sections of the app.
- Due to browser caching, it's even possible to read the newspaper offline.

Mobile web apps

HTML5 offers GPS, tap-to-call feature, camera API etc, but

Native features that remain inaccessible in the browser:

- Notifications (progressive web apps can do this)
- Running in the background
- Accelerometer information (other than detecting landscape or portrait orientations)
- Complex gestures.

Hybrid apps

Emerged to address the inability of web apps to access device sensors (like Notifications and Bluetooth) while preserving its highly desirable cross-platform support.

→Achieved by building HTML5-based applications that run in the browser.

→Here, the browser is embedded inside a native container app that provides a bridge for the HTML5 pages to access the low-level device functions.

Hybrid apps

They live in an app store and can take advantage of the many device features available.

Companies build hybrid apps as wrappers for an existing web page

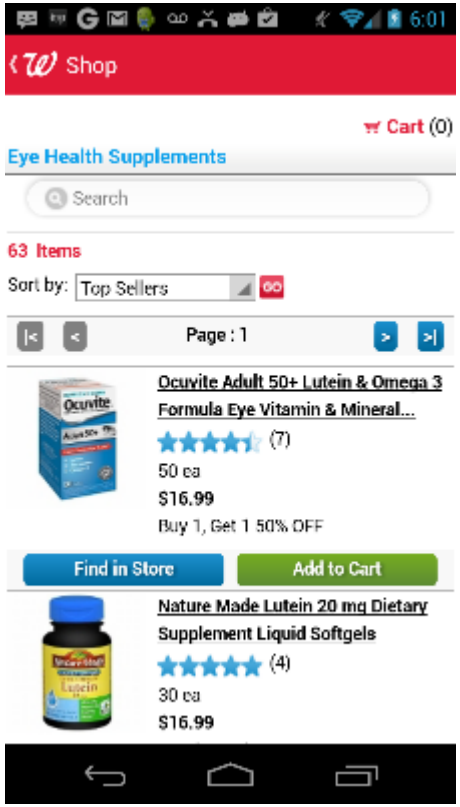
- Achieve a presence in the app store
- Allow cross-platform development
- Reduced development costs

  The same HTML code components can be reused on different mobile operating systems.

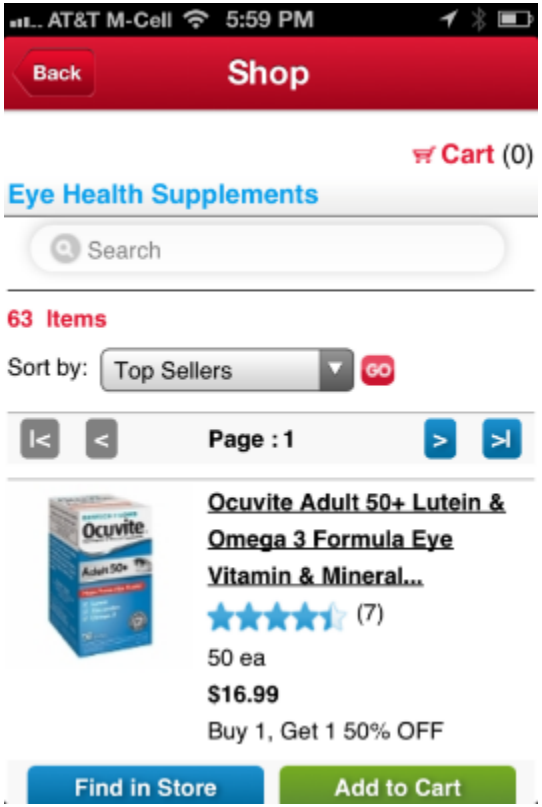
→ PhoneGap, Sencha allow people to design and code across platforms, using the power of HTML.

Hybrid app example 1

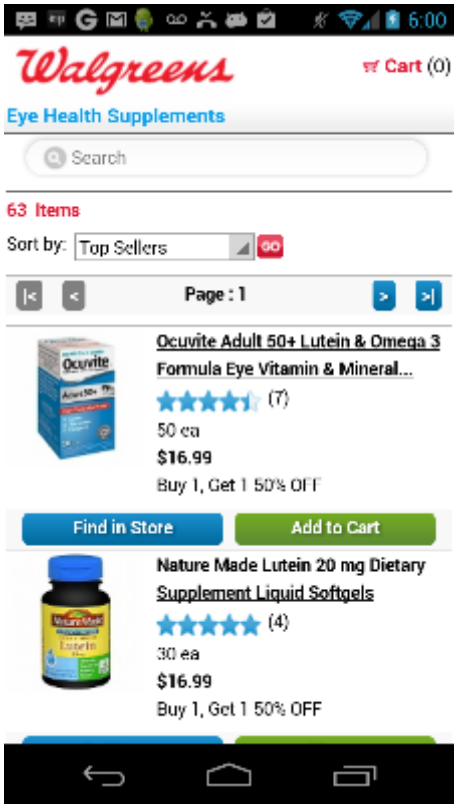
The *shop* section of Walgreens app uses a WebView (browser view).
Differences: top header, back button, logo on web page



Android



iPhone



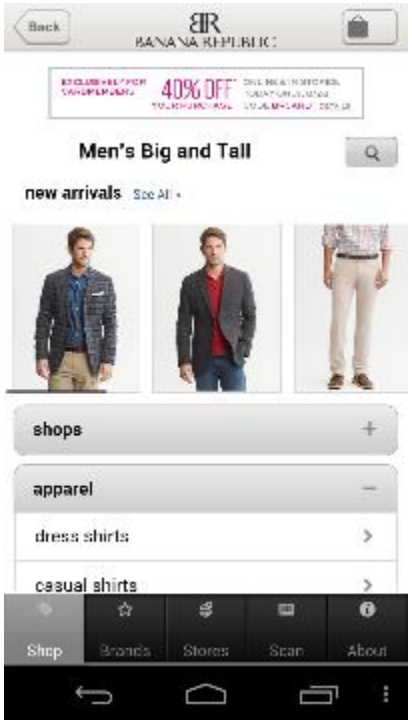
Web page

Hybrid app example 2

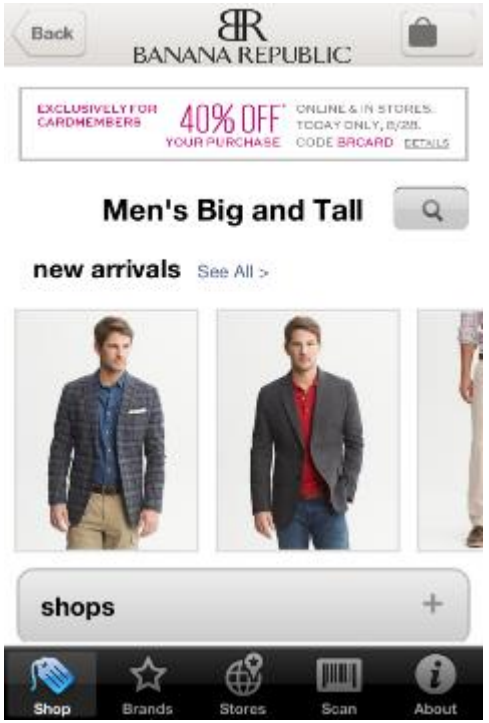
Banana Republic app.

Disadvantages:

- Android devices come with a physical or virtual Back button.
- The tab bar at the bottom of the page works well in the iOS design, but is clunky and clearly non-native on Android.



Android



iPhone

Progressive web apps

Combination of web & native, no install required.

The user progressively builds a relationship with the app over time, it becomes more and more powerful. It loads quickly, even on flaky networks, sends relevant push notifications, has an icon on the home screen, and loads as a top-level, full screen experience.

Demo on Google's codelabs:

- Design and construct an app using the "app shell" method.
- Make your app work offline.
- Store data for later offline use.

What type of app to develop?

Device features → native

Although web apps can take advantage of some features, native apps (and the native components of the hybrid apps) have access to the full device-specific features, including sensors, camera, gestures, and notifications.

Offline functioning → native

A native app is best if your app must work when there is no connectivity. In-browser caching is available in HTML5, but it's still more limited than what you can get when you go native.

What type of app to develop?

Discoverability → web

Content is a lot more discoverable on the web than in an app

- Native apps imply going to the app store, searching for an app, download it, and then trying to find 'the answer' within the app.
- Need maintenance (updates), space on the device, useful only if it is used often.

What type of app to develop?

Speed → native

"Facebook's biggest mistake had been betting on the mobile web and not going native (2012)". Up to that point, the Facebook app had been a hybrid app with an HTML core; got replaced with a truly native app for better responsiveness.

Installation → web

Bookmarking (less familiar) versus installation from store.

What type of app to develop?

Maintenance → web

Native is more complicated not only for users, but also for developers (multiple versions, platforms).

Platform independence → web

Big parts of code can be reused when supporting web apps (different HTML5 versions vs. everything different)

What type of app to develop?

Content restrictions, approval process, and fees → web

Third parties impose rules on content and design can be taxing both in terms of time and money. Native apps must pass approval processes and content restrictions imposed by app stores, whereas the web is free for all.

Development cost → web

Mastering HTML5 vs. mastering specific languages. Native requires specialized talent.

What type of app to develop?

User interface → native

Native offers user experience that is consistent with the operating system.

Web - responsive design for all platforms.

What type of app to develop?

	Device Access	Offline capability	Discoverability	Installation	Speed	Maintenance	Dev Cost	App store	Approval
Native	Full	Good	Low	Easy	Very fast	Low	Expensive	Available	Mandatory
Hybrid	Full	Limited	Low	Easy	Native-like	High	Reasonable	Available	Low hassle
Web	Partial	Limited	High	Easiest	Fast	Highest	Reasonable	No	None

What type of app to develop?

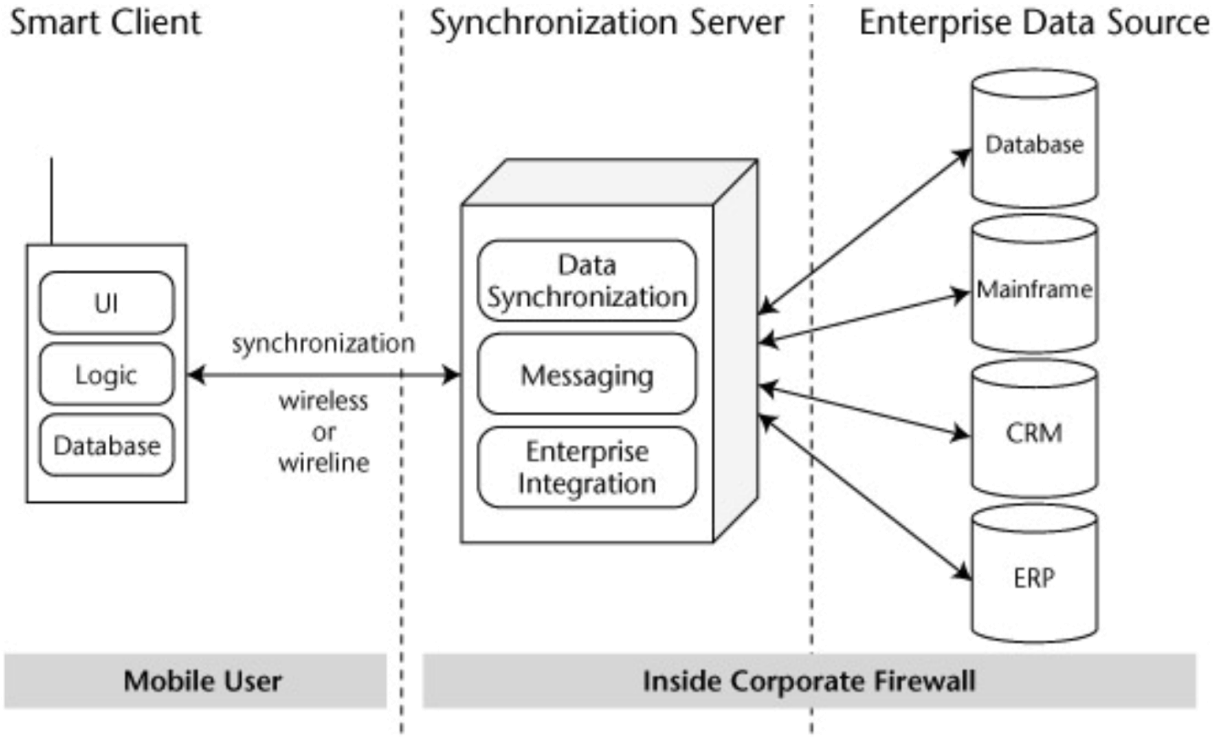
Summary: native and hybrid apps are installed from an app store, whereas web apps are mobile-optimized webpages that look like an app. Both hybrid and web apps render HTML web pages, but hybrid apps use app-embedded browsers to do that.

Conclusion: there is no best choice. The choice of one versus the other depends on unique needs.

Software layers

Software layers

The source code of the software application is usually organized on several levels called layers.



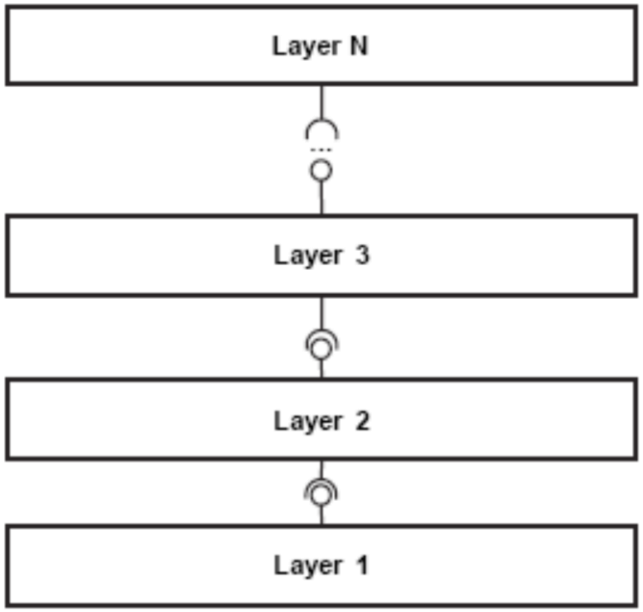
Software layers

Layer: group of software components which provide an abstract image of the system from a certain perspective.

Components of adjacent layers communicate with each other, providing or consuming services.

The components of a layer run:

- in the same application,
- on the same hardware system



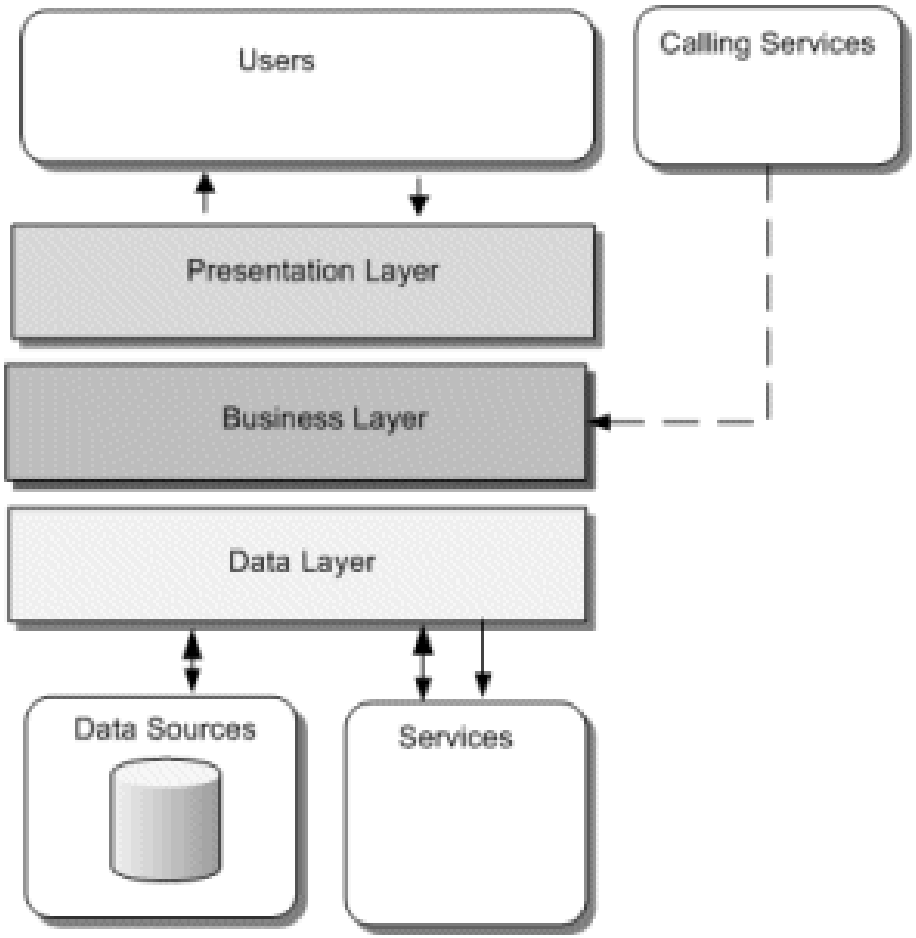
Software layers

Client applications are usually structured in 0 to 3 levels of code.

- A customer with 0 level of code is a thin client
- A client with 1-3 levels of code is a fat client:
  1. Presentation Layer
  2. Business Layer
  3. Data Layer

Software layers

Most developers that build applications will identify with presentation, business, and data layers.



Software tiers

Software tiers

→ Multi-tier architecture, n-tier architecture

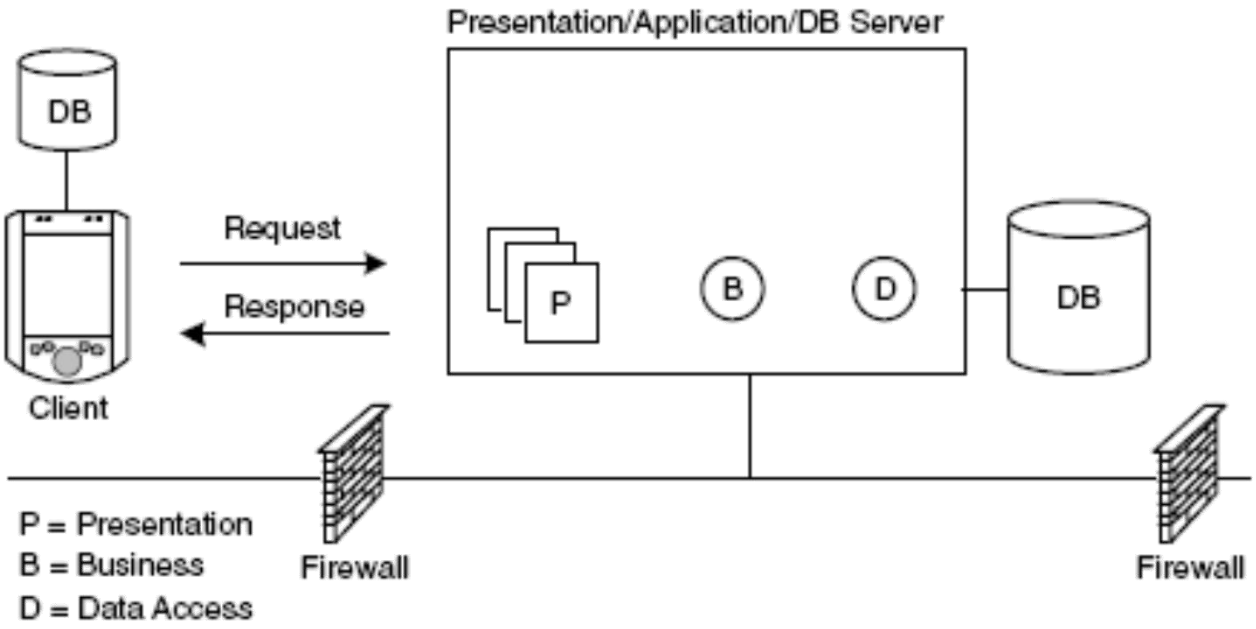
The tier refers to a collection of software components that run together in a separate process in order to provide an abstract view of a system's (service) perspective.

→ Each level (tier) runs in a separate process, possibly on a separate physical machine.

One-tier architecture

Code levels are running in a single process on a single machine.

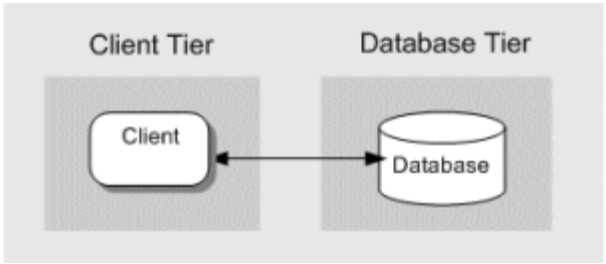
All software layers deployed in same tier.

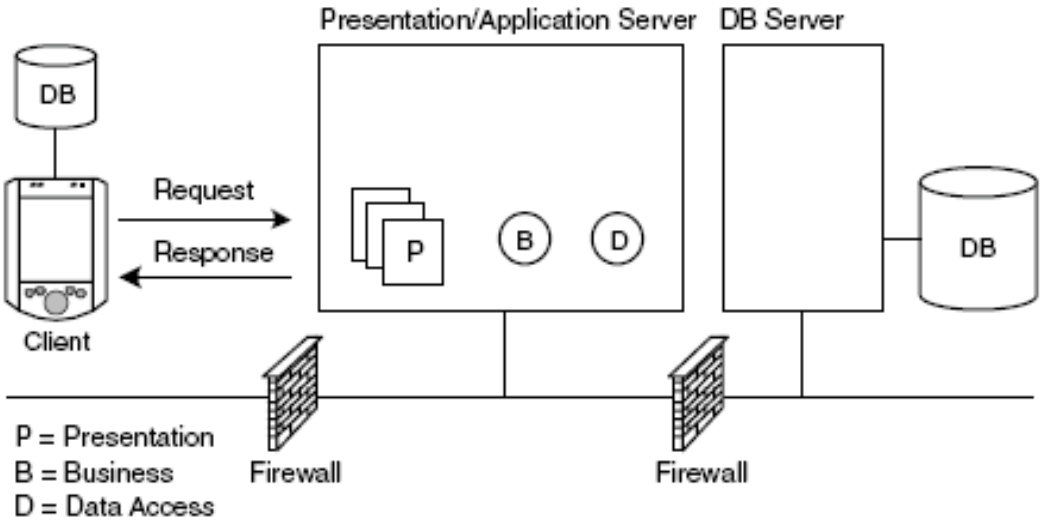


Two-tier architecture

Access to data is split into presentation and logic levels

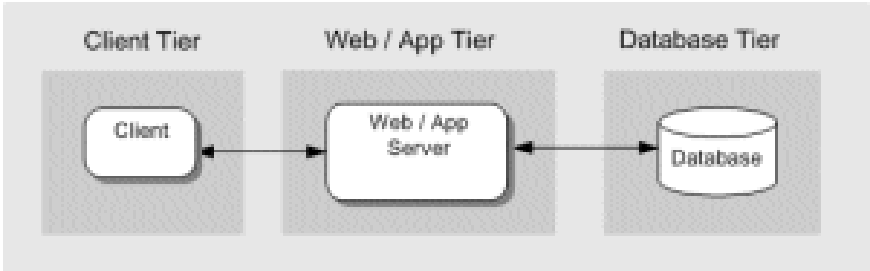
- Database server

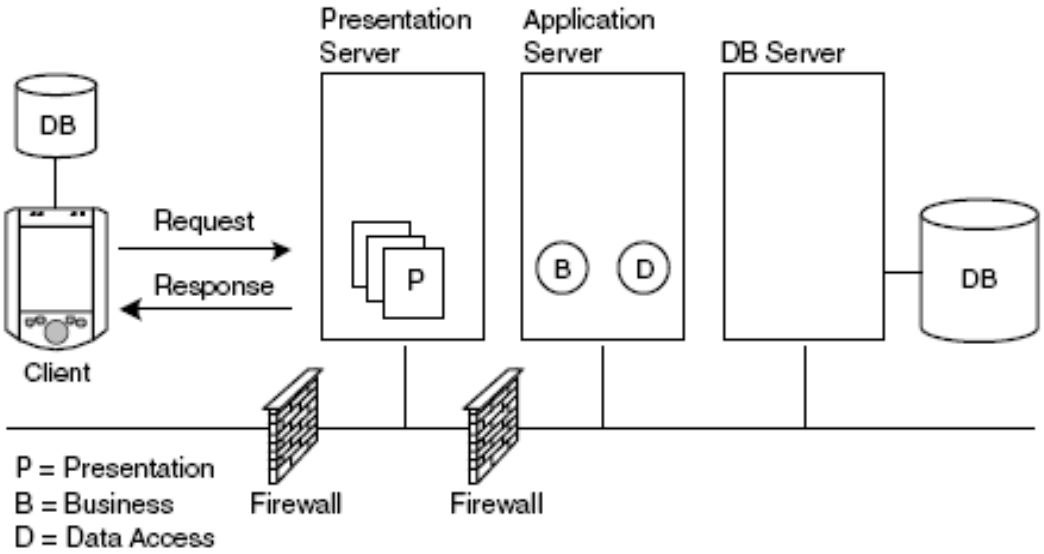




Three-tier architecture

The levels of presentation, application and data are completely separated.

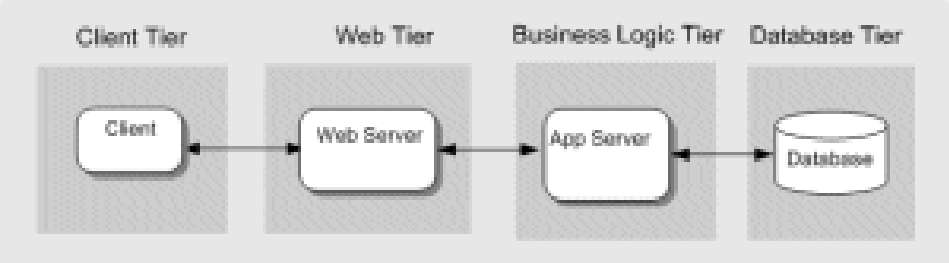




N-tier architecture

The levels of presentation, application and data are completely separated.

In mobile development: client (smartphone) is automatically an extra tier.



Layers versus tiers

Logical (software) layers are merely a way of organizing your code. We're not implying that these layers might run on different computers or in different processes on a single computer.

Physical (software) tiers are only about where the code runs. Tiers are places where layers are deployed and where layers run.

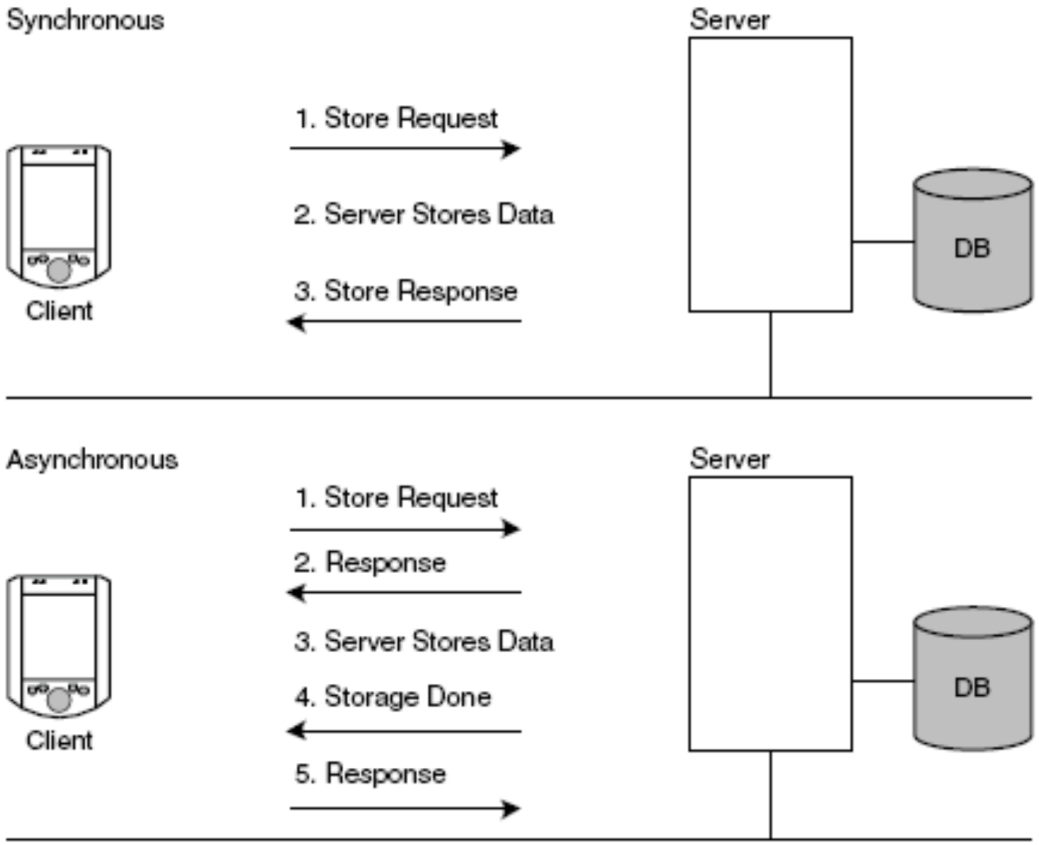
→ In other words, tiers are the physical deployment of layers.

Connection types

1. Always connected – mobile applications are always connected to the communication infrastructure, having continuous access to a central application.
2. Partially connected – mobile applications have continuous access to a central application, but can also operate outside login sessions.
3. Never connected – independent applications that do not require connection to a central application.

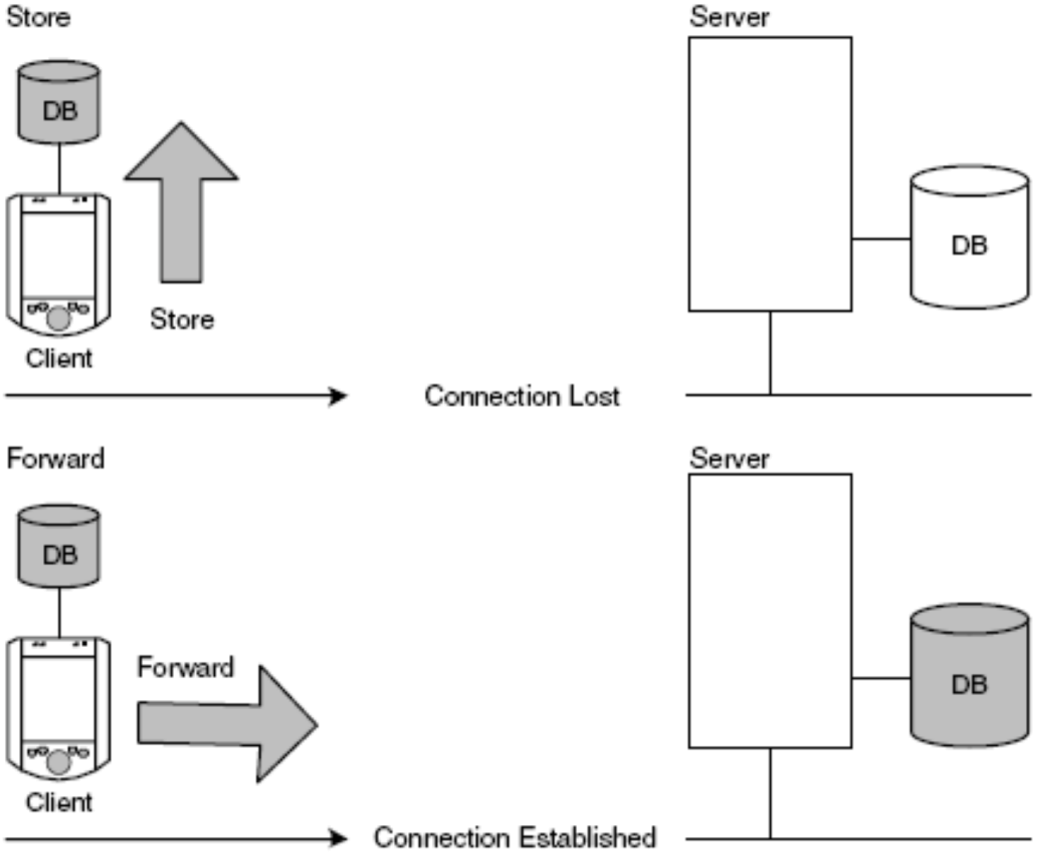
Synchronization types

1. Continuous communication



Synchronization types

2. Store-and-forward communication



Summary

Client type

- Thin client
- Rich client

} And combination of both into a 3 layer architecture

Connection type

- Connected
- Never connected

} Reliability of network vs. caching

Synchronization

- Continuous
- Store-and-forward

} Support for local storage

App type: platform independent vs. access to hardware

Higher Order Architectures

Services over the Internet

→Infrastructure as a Service (IaaS)

→Platform as a Service (PaaS)

→Software as a Service (SaaS)

IaaS

An "instant" computing infrastructure managed over the Internet.

- Scales up and down with demand
- Clients pay only for what they use

→Avoid the expense and complexity of buying and managing physical servers and other datacenter infrastructure.

→Each resource is offered as a separate service component

→The cloud computing service provider manages the infrastructure, while clients purchase, install, configure, and manage their own software—operating systems, middleware, and applications.

IaaS examples

- Amazon Web Services (AWS)
- Microsoft Azure
- Cisco Metacloud
- Google Compute Engine
- Rackspace

→Common use-case: extend current data center infrastructure for temporary workloads (e.g. increased Christmas holiday site traffic)

PaaS

A cloud computing model in which a third-party provider delivers hardware and software tools (needed for application development) to users over the internet.

→A PaaS provider hosts the hardware and software on its own infrastructure.

→Clients don't have to install in-house hardware and software to develop or run a new application.

PaaS examples

- Windows Azure
- Heroku
- Google App Engine
- Apache Stratos
- AWS Elastic Beanstalk

→Common use-case: increase developer productivity and utilization rates while also decreasing an application's time-to-market

IaaS → PaaS → SaaS

1. IaaS supplies the basic compute, storage and networking infrastructure along with the hypervisor (the virtualization layer). Users must then create virtual machines, install operating systems, support applications and data, and handle all of the configuration and management associated with those tasks.

2. PaaS offers more of the application stack than IaaS providers, adding operating systems, middleware (such as databases) and other runtimes into the cloud environment.

3. SaaS offers an entire application stack. Users simply log in and use the application that runs completely on the provider's infrastructure.

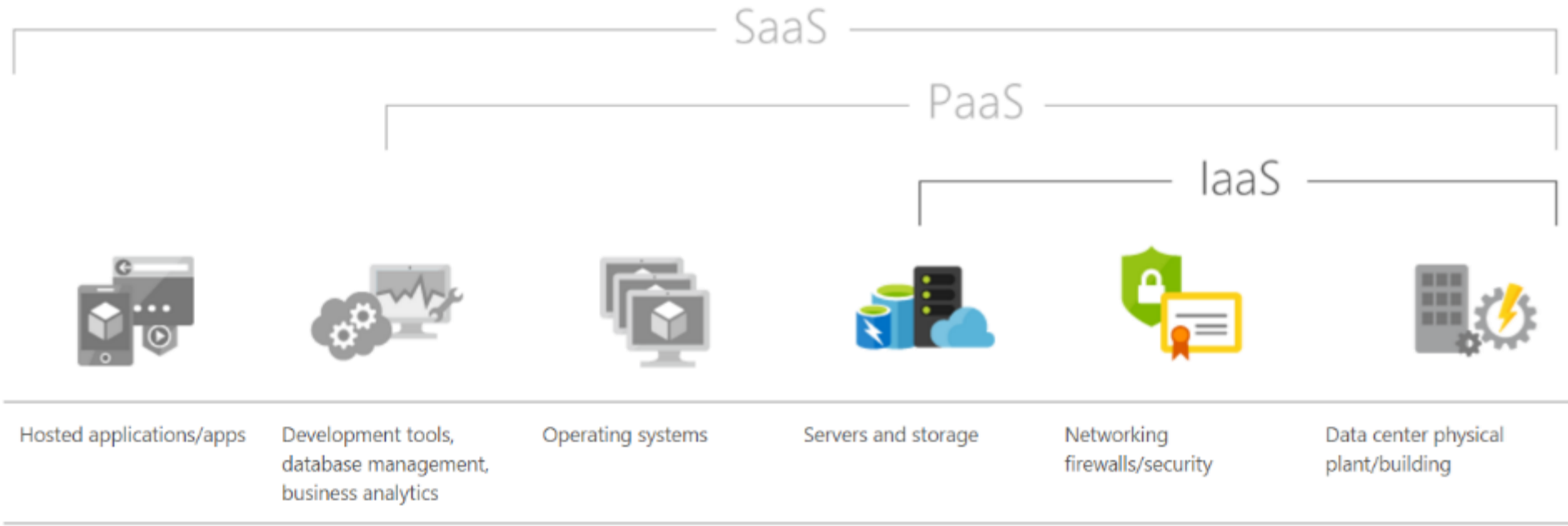
SaaS examples

- Google Workspace (Docs, Drive)
- Dropbox
- Salesforce
- Concur
- Cisco WebEx

→Common use-case: replaces traditional on-device software.

→No need for IT staff. All hw/sw problems are handled by the provider.

IaaS → PaaS → SaaS



Directions in Mobile Architectures

→Backend as a Service (BaaS)

In 2011 *Sravish Sridhar* (Kinvey) interviewed mobile developers on how they want to setup and operate backends for their mobile apps to host data, run business logic, etc.

IaaS – Amazon EC2, Windows Azure (too hard)

… dealing with Linux prompts and virtual machine setup

PaaS – Heroku, CloudFoundry, OpenShift

…building a backend platform stack from scratch

Backend as a Service (BaaS)

→ devs wanted all their backend features to be available out of the box, as a fully secure, scalable and managed service.

Cloud level: abstracting IaaS, on top of which creating and managing data stores (meta data and large files), connecting the data to cloud-based business logic, and 3rd-party cloud APIs.

Device level: ensure the accuracy and availability of data, context and security rules between an app on a device and the cloud, and across thousands of devices running that app.

Backend as a Service (BaaS)

Advantages:

- Scalability of app.
- No infrastructure hassles.
- Reduce software operating costs.
- Focus on frontend development.
- Remove repetitive tasks (backend can be 80% of effort that users don't see).

Features: Data Management, APIs, File Storage, Databases, Email Notifications, Push Notifications, Login Authentication, Social Media Integration (Twitter, FB), Infrastructure.

$$$ → after a certain threshold of API calls, DB size, or data transfer.

Mobile Ecosystem

An ecosystem rich with companies that form different parts of the mobile and cloud stack.

The service providers and the handset OEMs* are the 2 original players. Today, new cloud providers, API and SDK layers have inserted themselves in between the original two players of the ecosystem.

** original equipment manufacturer*

Backend as a Service (BaaS) Ecosystem Map

kinvey