

Making Presentations

CMU 2022-23

1. What is a presentation?
2. What and how should be presented?
3. Using slides

3 Key Aspects

- Audience: Who is this for?
- Goal: What do I want to change?
- Time : How long do I have?

⇒ **Message**

The Audience and the Goal

- Who are they?
- What do they know?
- What is the best possible result?
- How are they reacting?
- *Tip: Beware of content reuse!*

Time

- Commitment with the audience.
- Key: Rehearsing (with less time).
- *Tip: 2 minutes per slide (on average).*

Presentation

Communicate one concept to an audience in a limited amount of time.

The Presenter

- Key Aspects: Clarity, Discretion, **Enthusiasm**.
- **Important**: what we know and can teach.
- **Not important**: who we are. (what is necessary to say).
- The speaker is us, not the univ./company.

- **Verbalize** your train of thought.
- **Don't verbalize** the presentation process.

The Channel

- Keep it clean = Avoid noise.
- Prepare the location, yourself and devices.
- Avoid verbal or physical tics and quirks.
- Emphasize the essential.
- Who is feeling less involved?

The Message: A Standard Paper Presentation

- The message must serve the goal!
- Tip: Constantly ask:
Is this needed?
Is this working toward the goal?
Can I reinforce the message?

A Standard Paper Presentation

- Title and Summary:
 - Informative: avoid “introduction”, “conclusion”,...
 - Include credits.
 - *Tip: summary in the title slide.*

A Standard Paper Presentation

- Title and Summary.
- Problem, Motivation, Goal:
 - Generally most important part!
 - Varies with the presentation type:
 - Science: Context
 - Commercial: Advantages, Gains
 - Academic: Place in Knowledge, Future Application.

A Standard Paper Presentation

- Title and Summary.
- Problem, Motivation, Goal.
- Solutions:
 - Science: Contribution.
 - Commercial: Product.
 - Class: Concepts, Techniques.

A Standard Paper Presentation

- Title and Summary.
- Problem, Motivation, Goal.
- Solutions.
- Practice:
 - Science: Implementation, Tests, Properties.
 - Commercial: Use Cases.
 - Class: Examples and Exercises.

A Standard Paper Presentation

- Title and Summary.
- Problem, Motivation, Goal.
- Solutions.
- Practice.
- Conclusions:
 - Restate the relation problem <-> solution.
 - Give pointers to further information.

A Standard Paper Presentation

- Title and Summary.
- Problem, Motivation, Goal.
- Solutions.
- Practice.
- Conclusions.
- Q & A:
 - Listen to the question.
 - Repeat the question.
 - Be honest.

Slides

- **Expose vs. Teach**
- Main **goal**: emphasys & time saving.
- Make them **simple**.
- Be very careful with **reuse**.
- Use graphical elements carefully.
- Be careful with contrast, detail and fonts:
 - “Those tiny numbers are...”

Slide Design Models

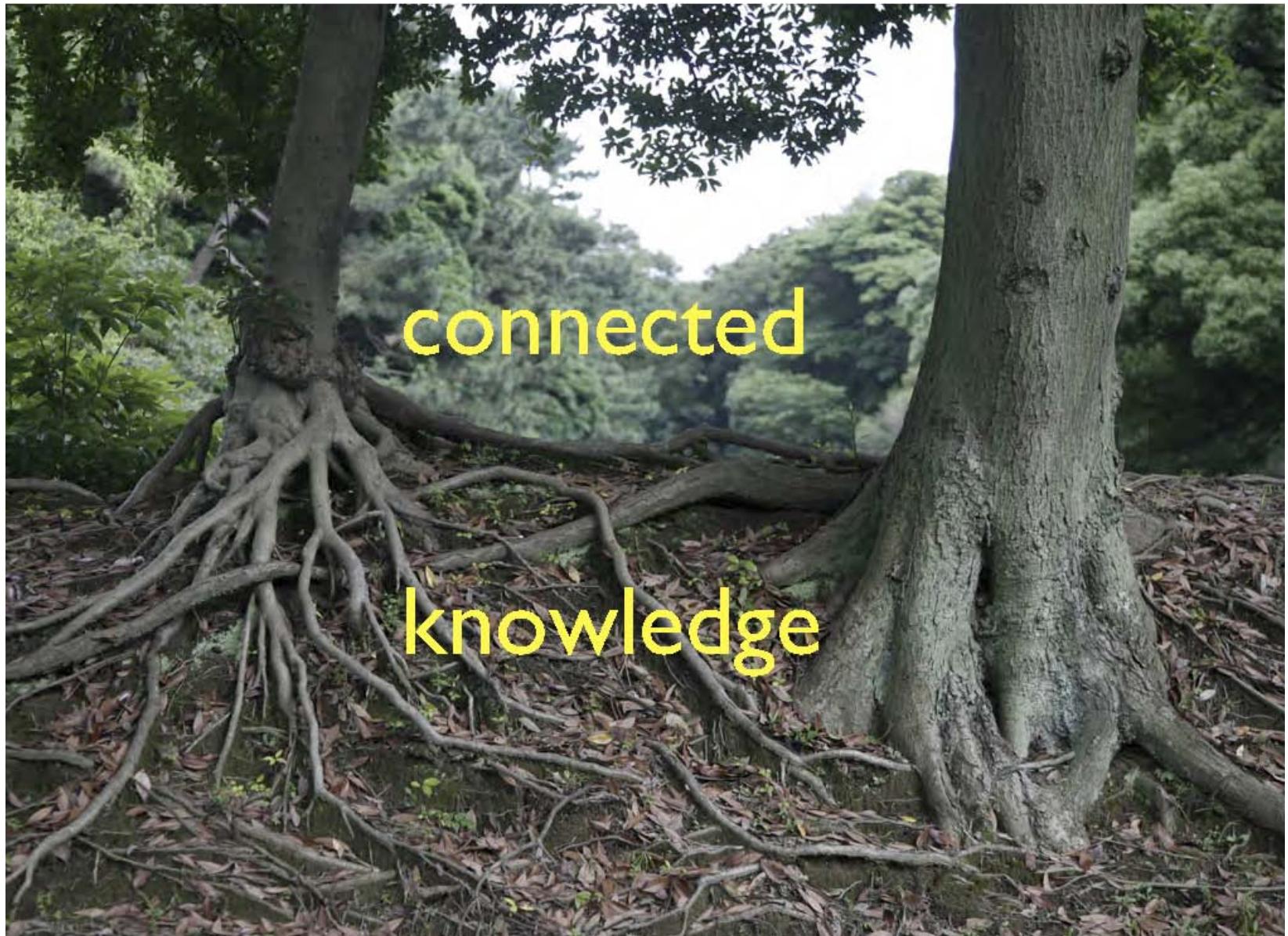
- Models:
 - 2 minutes per slide, 6 words by 6 bullets.
 - “10, 20, 30”
 - Others: see videos in the references...

Slides

- Emotions and critical points:
 - *Tip: Slides should not be handouts.*
- Speaker: detailed content.
- Practice transitions:
 - *Does the next slide work?*

Figures

- Great speakers (almost) only need figures.
- Figures should be memorable models!!!
- Better **boring and clear** than funny and confusing.



A photograph showing two hands in dark suit jackets and white shirts. They are holding a black smartphone between them. The hand on the left has a gold ring on the ring finger. The word "connected" is overlaid in yellow text at the top of the image.

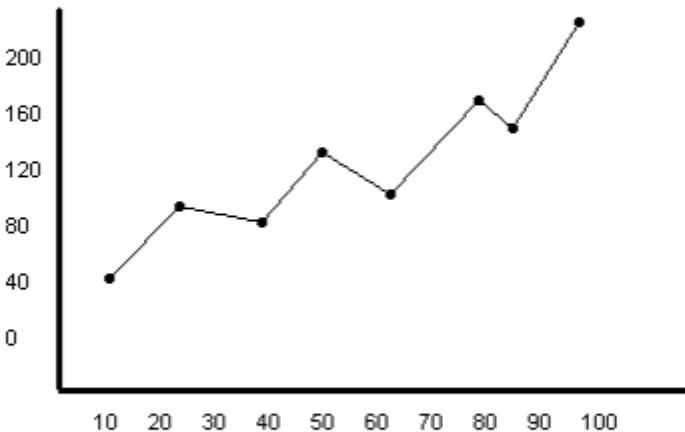
connected

knowledge

Graphs

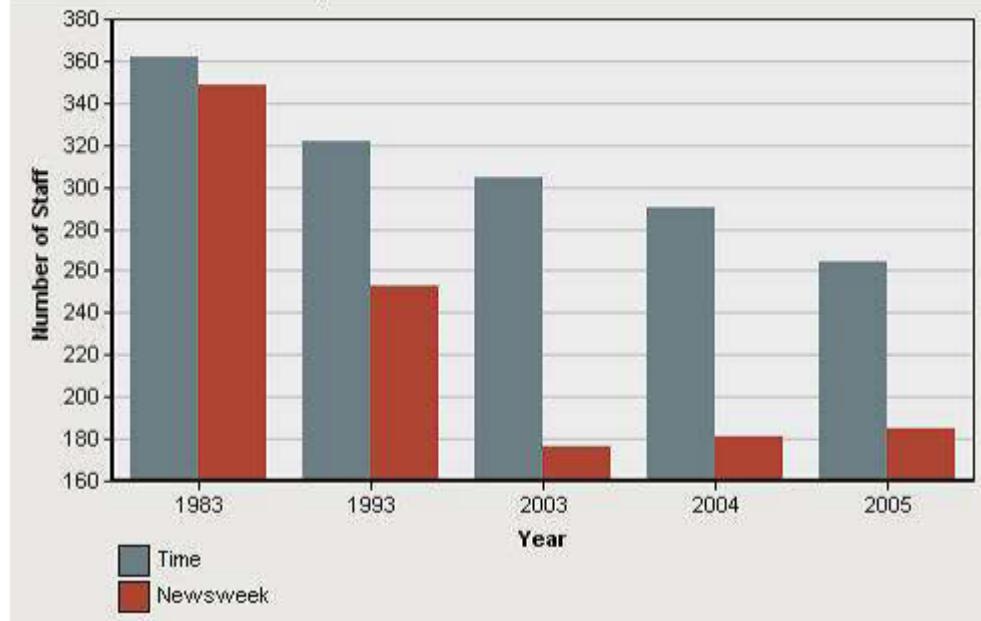
- Graphs: 1 message, readable, with a legend, with units!
- *Tip: explain the figure before making your point.*

Bad Graphs



NEWS MAGAZINE STAFF SIZE OVER TIME

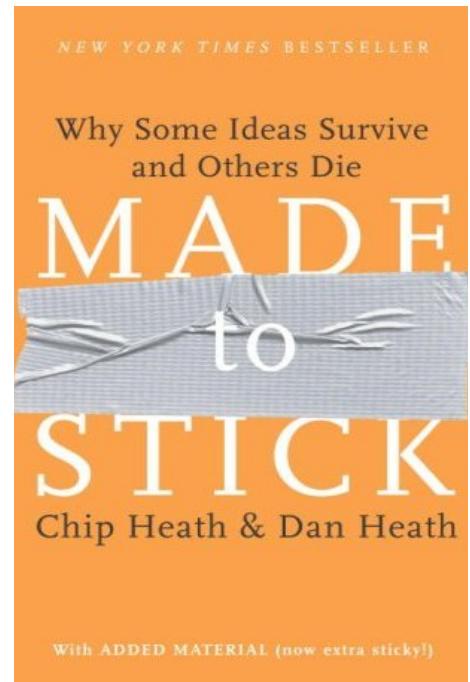
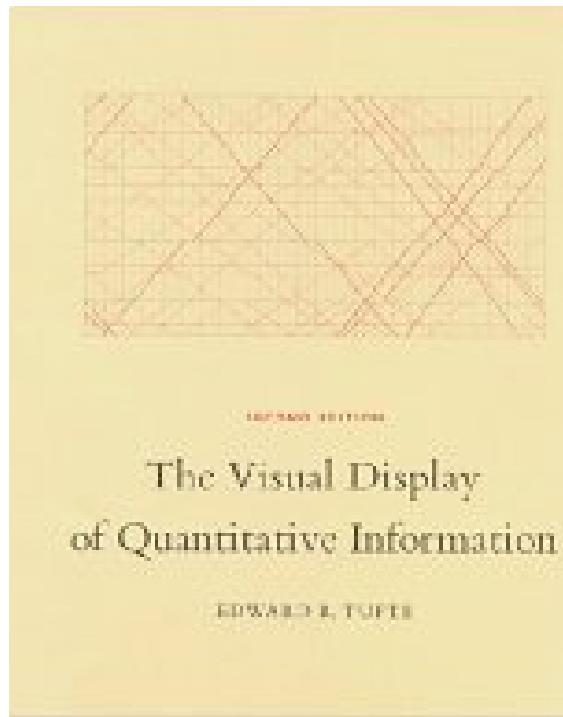
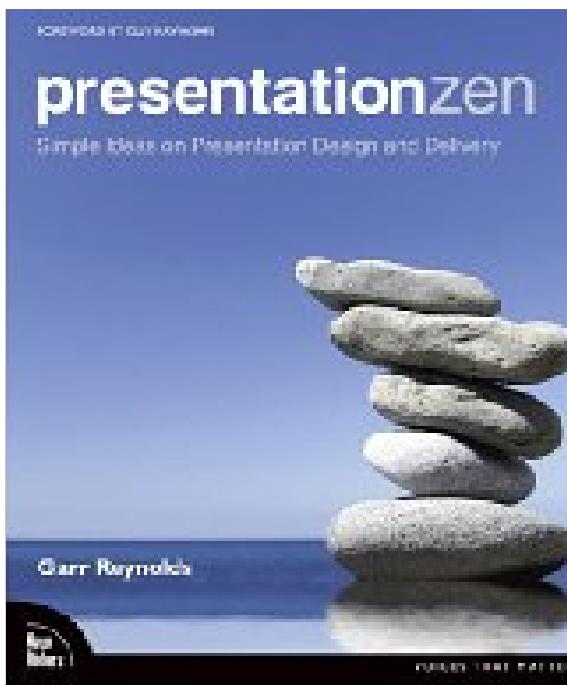
Time and Newsweek select years 1983 - 2005



Conclusions

- A good presentation is:
 - Tailored for the audience.
 - Coordinated to reach a particular goal(s).
 - On time.
 - Clear.

References



Video References

- 3 min thesis:
[https://www.youtube.com/watch?
v=dh0pJdgY6Lc](https://www.youtube.com/watch?v=dh0pJdgY6Lc)
- Identity 2.0:
www.youtube.com/watch?v=RrpajcAgR1E
- Clifford Stoll @ TED:
www.youtube.com/watch?v=Gj8IA6xOpSk

Mobile and Ubiquitous Computing

2022-2023

MEIC/METI- Alameda & Taguspark

A Global Overview

A Glimpse of the Past...



Fig. 5 -The Whirlwind computer, the large-scale supercomputer of the 1950s and 1960s developed for automated strategic air defense applications. It could display real time video. The last Whirlwind computer was shut down in 1983.

Fig. 4 -The console and a few peripheral units from an IBM System/360 model 30.



Fig. 7 -The DEC PDP-8, a popular minicomputer of the late 1960s and early 1970s. The yellow box identifies the processors; the components above are hard disk drives, which wouldn't be necessary in an embedded system.

A Glimpse of Today...



A Glimpse of Today...

JAN
2022

ESSENTIAL DIGITAL HEADLINES

OVERVIEW OF THE ADOPTION AND USE OF CONNECTED DEVICES AND SERVICES



7.91
BILLION

URBANISATION

57.0%



5.31
BILLION

vs. POPULATION

67.1%



4.95
BILLION

vs. POPULATION

62.5%



4.62
BILLION

vs. POPULATION

58.4%

A Glimpse of Today...

JAN
2022

UNCONNECTED POPULATIONS

COUNTRIES AND TERRITORIES WITH THE LARGEST UNCONNECTED POPULATIONS AND THE LOWEST LEVELS OF INTERNET ADOPTION



ABSOLUTE: LARGEST UNCONNECTED POPULATIONS

#	LOCATION	UNCONNECTED POPULATION	% OF POP. OFFLINE
01	INDIA	742,003,000	53.0%
02	CHINA	421,432,000	29.1%
03	PAKISTAN	144,434,000	63.5%
04	BANGLADESH	114,511,000	68.5%
05	NIGERIA	104,888,000	49.0%
06	ETHIOPIA	89,502,000	75.0%
07	DEM. REP. OF THE CONGO	77,293,000	82.4%
08	INDONESIA	73,047,000	26.3%
09	BRAZIL	49,375,000	23.0%
10	TANZANIA	46,794,000	75.0%

RELATIVE: LOWEST LEVELS OF INTERNET ADOPTION

#	LOCATION	% OF POP. OFFLINE	UNCONNECTED
232	NORTH KOREA	>99.9%	25,938,000
231	CENTRAL AFRICAN REPUBLIC	92.9%	4,613,000
230	ERITREA	92.0%	3,341,000
229	COMOROS	91.5%	822,000
228	SOUTH SUDAN	89.1%	10,248,000
227	SOMALIA	86.3%	14,333,000
226	NIGER	85.5%	21,881,000
225	KIRIBATI	85.4%	105,000
224	BURUNDI	85.4%	10,623,000
223	DEM. REP. OF THE CONGO	82.4%	77,293,000

A Glimpse of Today...

JAN
2022

DEVICE OWNERSHIP

PERCENTAGE OF INTERNET USERS AGED 16 TO 64 WHO OWN EACH KIND OF DEVICE



ANY KIND OF
MOBILE PHONE



GWI.

96.6%

YEAR-ON-YEAR CHANGE
-0.5% (-50 BPS)

SMART
PHONE



we
are
social

96.2%

YEAR-ON-YEAR CHANGE
-0.4% (-40 BPS)

FEATURE
PHONE



GWI.

8.8%

YEAR-ON-YEAR CHANGE
-2.2% (-20 BPS)

LAPTOP OR
DESKTOP COMPUTER



GWI.

63.1%

YEAR-ON-YEAR CHANGE
-2.0% (-130 BPS)

TABLET
DEVICE



34.8%

YEAR-ON-YEAR CHANGE
+1.5% (+50 BPS)

GAMES
CONSOLE



SMART WATCH OR
SMART WRISTBAND



GWI.

20.3%

YEAR-ON-YEAR CHANGE
-5.1% (-110 BPS)

TV STREAMING
DEVICE



15.5%

YEAR-ON-YEAR CHANGE
+7.6% (+110 BPS)

SMART HOME
DEVICE



GWI.

14.1%

YEAR-ON-YEAR CHANGE
+14.6% (+180 BPS)

VIRTUAL REALITY
DEVICE



4.8%

YEAR-ON-YEAR CHANGE
+9.1% (+40 BPS)

SOURCE: GWI (Q3 2021). FIGURES REPRESENT THE FINDINGS OF A BROAD GLOBAL SURVEY OF INTERNET USERS AGED 16 TO 64. SEE [GWI.COM](#) FOR FULL DETAILS.
NOTE: PERCENTAGE CHANGE VALUES REFLECT RELATIVE CHANGE. "BPS" VALUES SHOW THE CHANGE IN BASIS POINTS, AND REFLECT ABSOLUTE CHANGE.

A Glimpse of Today...

JAN
2022

SHARE OF WEB TRAFFIC BY DEVICE

PERCENTAGE OF TOTAL WEB PAGES SERVED TO WEB BROWSERS RUNNING ON EACH KIND OF DEVICE



K
KEPIOS

53.96%

YEAR-ON-YEAR CHANGE

+2.0%

+104 BPS



we
are
social

43.53%

YEAR-ON-YEAR CHANGE

-1.5%

-66 BPS



o
o

2.47%

YEAR-ON-YEAR CHANGE

-12.4%

-35 BPS



0.03%

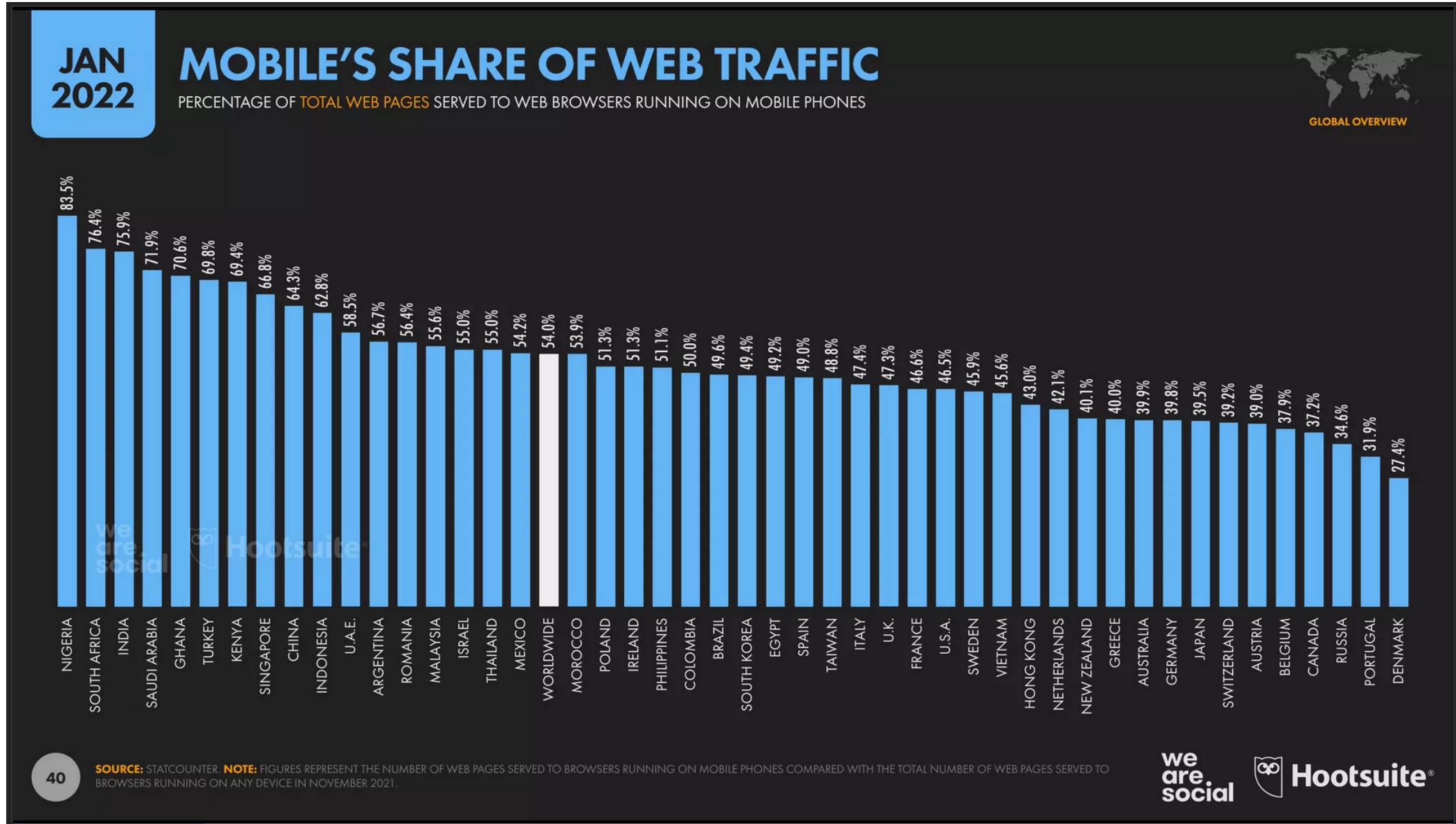
YEAR-ON-YEAR CHANGE

-50.0%

-3 BPS

SOURCE: STATCOUNTER. NOTES: FIGURES REPRESENT THE NUMBER OF WEB PAGES SERVED TO BROWSERS RUNNING ON EACH TYPE OF DEVICE COMPARED WITH THE TOTAL NUMBER OF WEB PAGES SERVED TO BROWSERS RUNNING ON ANY DEVICE IN NOVEMBER 2021. PERCENTAGE CHANGE VALUES REPRESENT RELATIVE CHANGE (I.E. AN INCREASE OF 20% FROM A STARTING VALUE OF 50% WOULD EQUAL 60%, NOT 70%). "BPS" VALUES REPRESENT BASIS POINTS, AND INDICATE THE ABSOLUTE CHANGE. FIGURES MAY NOT SUM TO 100% DUE TO ROUNDING.

A Glimpse of Today...

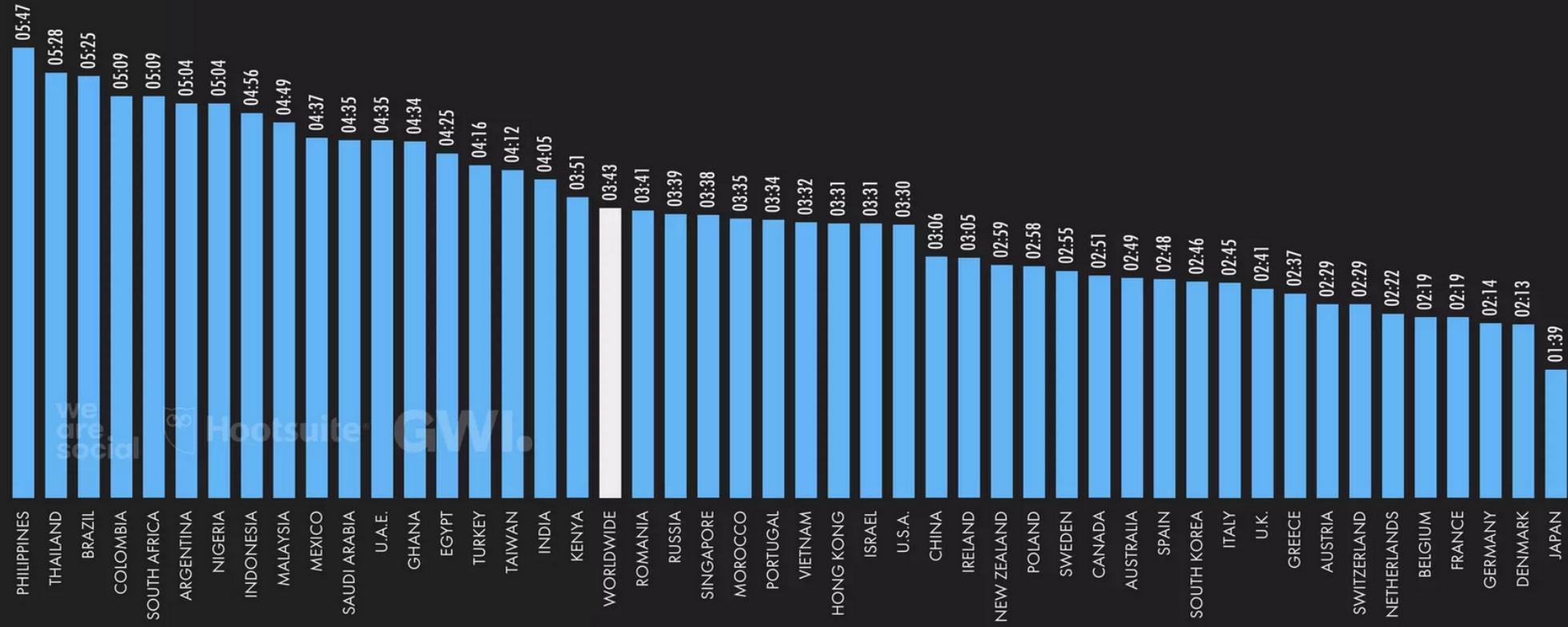


A Glimpse of Today...

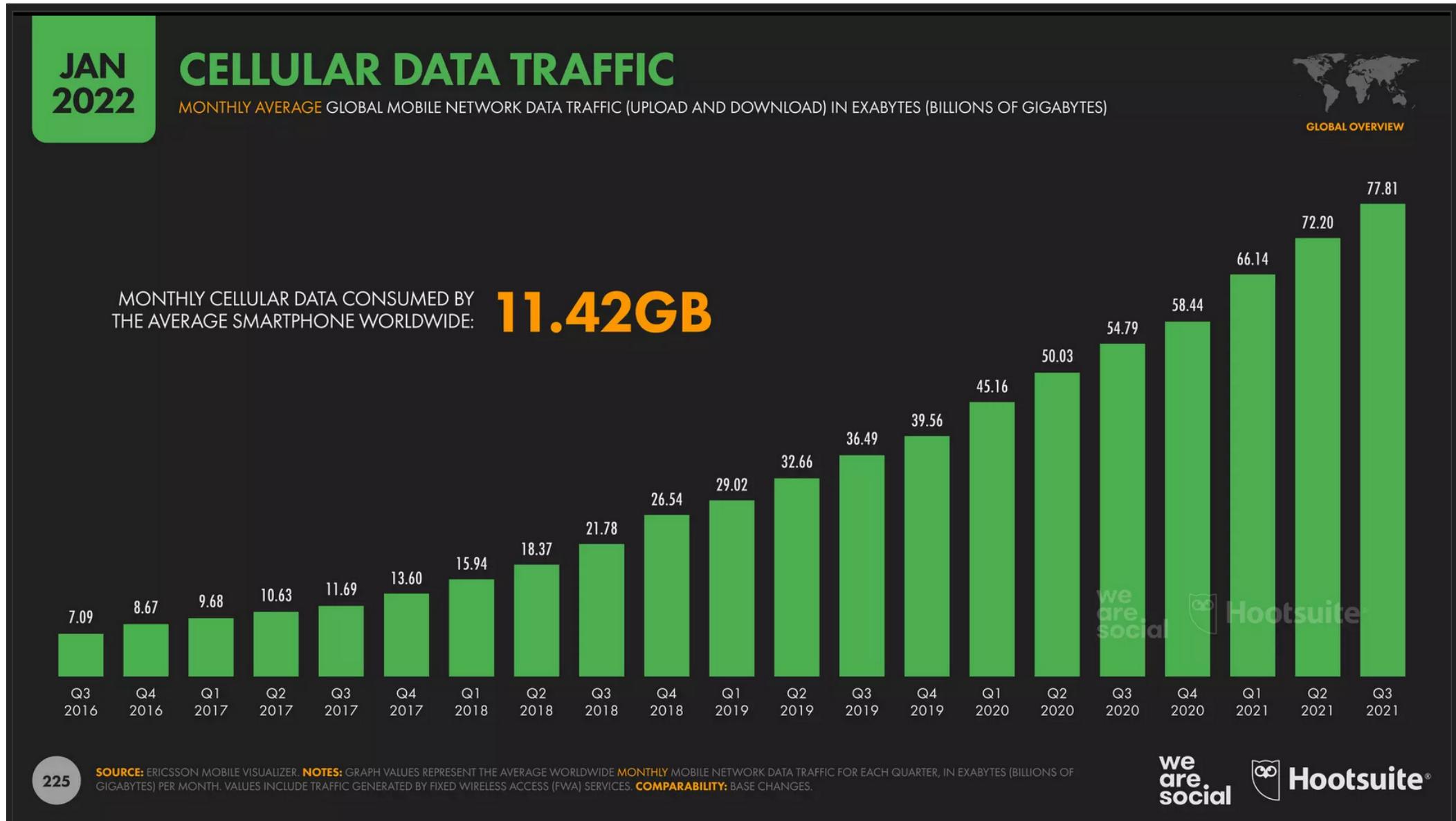
JAN
2022

TIME SPENT USING THE INTERNET ON MOBILES

AVERAGE AMOUNT OF TIME PER DAY THAT INTERNET USERS AGED 16 TO 64 SPEND USING THE INTERNET ON MOBILE PHONES



A Glimpse of Today...



Mobile Computing Today

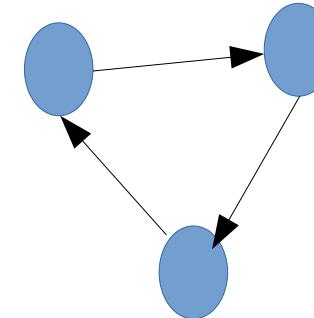
- Where do we stand ?
- Where we do not want to go !
- How far away are we ?
- Are we going in the right direction ?
- What are the challenges ?

A Glimpse of the Future?...

Google Glasses

<http://www.youtube.com/watch?v=JSnB06um5r4>

Where do we stand ?



- Thanks to cellular connections and Wi-Fi networks:
 - we have near-constant access to computing power and online data,
 - giving us what might be called *near-ubiquitous computing*
- We seem to be producing an attention economy that lives off users attention and data mining:
 - “If the service is free, you are the product.”
- It's not quite the ambient intelligence that was envisioned but it's a step in that direction.

Where are we going to? Where we do not want to go!

- A fundamental desired characteristic is *calm computing*, i.e. the system:
 - remains in the background
 - does what we want it to do seamlessly.
- However, today's technology are more likely *jittery technology*:
 - constant beeping for messages, posts, updates, news, etc....
 - phantom vibration: the perception of a phone's vibration in the absence of an event
 - even watching TV is no longer as it used to be due to second screening and chatterboxing.

Where are we going to? Where we do not want to go!

- We create new technology but technology changes us also!
- Pros:
 - Having so much information ready at the fingertips.
 - Cheap access to quickly deployed telecoms.
 - Increasing productivity.
- Cons:
 - Attention Economy
 - Privacy Loss
 - Focus Loss

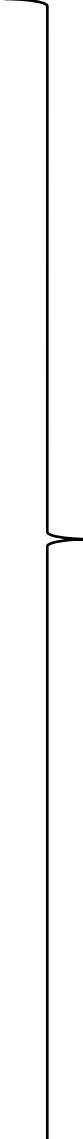
A Glimpse of the (would like to be) Future...

The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it...Today's multimedia machine makes the computer screen into a demanding focus of attention rather than allowing it to fade into the background.

*By Mark Weiser, Scientific American
September 1991*

Challenges

- Context-awareness
- Adaptability
- Cyberforaging
- Resource Discovery
- Offloading
- Replication
- Energy
- Security
- Operating system
- Networking
- Etc.



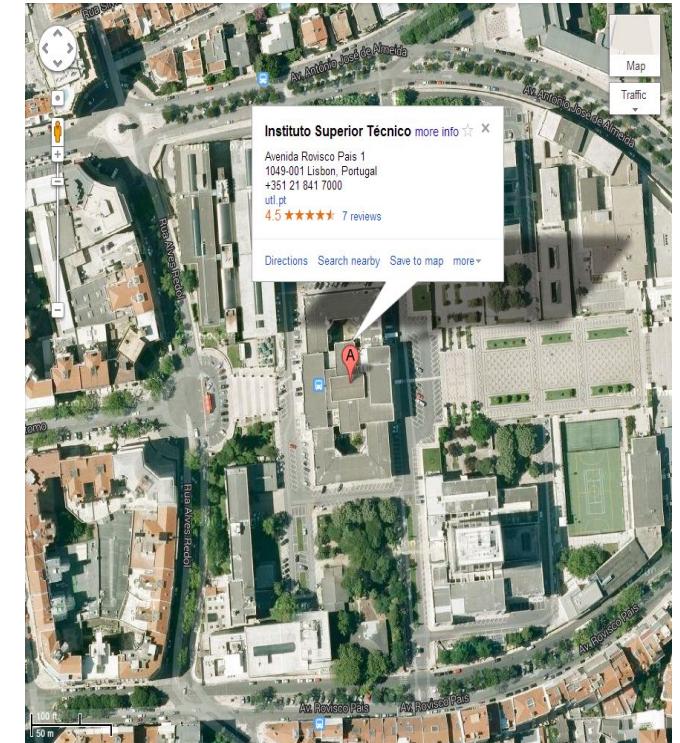
Applications
Middleware
Operating System

Context-Awareness

- **Invisibility** implies no configuration, no interaction. The system simply makes what the user wants
- This requires the system to be **pro-active** and **correct**.
- The system must get all the **context information** to **predict** what the user wants and act accordingly.
- Context information includes many aspects; from the most obvious which is location to many others as user's agenda, temperature, presence, etc...
- Basically, based on context, the system captures the user-intent.

Location

- Triangulation:
 - Iteration, which uses multiple distance measurements between known points,
 - angulation, which measures angle or bearing relative to points with known separation.
- Proximity:
 - measures nearness to a known set of points.
- Scene analysis:
 - examines a view from a particular point.
- Privacy, Symbolic, Scale, Cost, Absolute/Relative, Accuracy and Precision.



Adaptability

- The system must be capable of dealing with the variability of the environments.
- Such variability ranges all aspects:
 - network on /off/slower/faster, memory available or not, CPU available or not, etc... all resources may vary with time and location
- Capability of automatically adapting to the environment while remaining invisible.
- Ubiquitous (invisible) systems must be able to handle all the spectrum of resource variability, thus including fault-tolerance aspects (when resources are unavailable).
- Balancing pro-activity and invisibility.





It is not the strongest of the species that survives, nor the most intelligent;

It is the one that is most adaptable to change.

(Charles Darwin)

Resource Discovery

- Protocols to discover nearby resources.
- Use those resources if that is needed or Advantageous.
- Examples:
 - use a printer at the hotel where I've just arrived;
 - using more powerful CPU from a nearby laptop;
 - use a higher quality screen (HD, bigger screen, etc.);
 - use storage available from a local server or from the cloud.
- Scalable and correct discovery protocols (flooding, false positives, completeness).



Cyberforaging

- Dealing with variable, uncertain or unavailable resources can be done by taking advantage of other equivalent resources in the surroundings.
- E.g. lack of network availability may be solved with a shared hotspot.
- Invisibility implies:
 - discovering available resources
 - use such resources
 - automatically, without bothering the user
- Opportunities provided by cyberforaging:
 - take advantage any resource available
(e.g. network, printer, **CPU**, storage, screen, etc.)



Replication

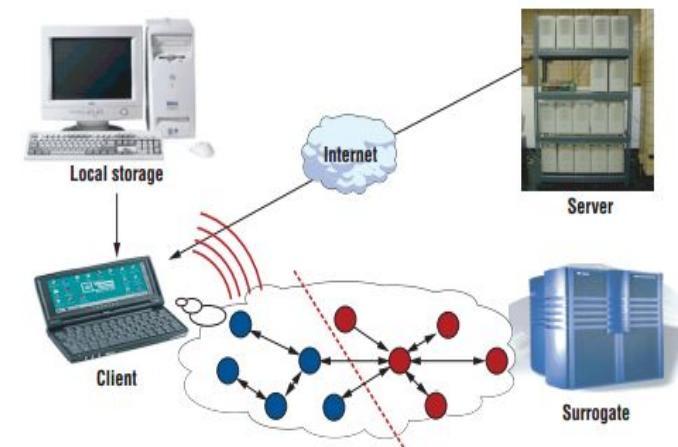


- Maintaining multiple consistent copies of data for accessibility and fault tolerance.
- Accessing data and code anywhere
- Data staging on untrusted surrogates
- File hoarding
- Object replication



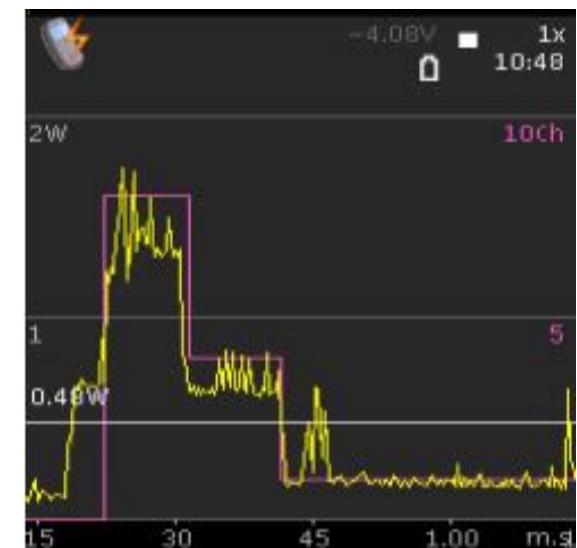
Offloading

- Offload computation from the mobile device to a more powerful CPU
- Implies deciding if it is really advantageous:
 - Time/Cost to compute.
 - Time/Cost to send and receive (what to transmit).
 - Time/Cost to decide what to offload (static and/or dynamic).
 - Run-time code changes to support offloading.



Energy

- Find where battery being mostly used.
- Minimize what most consumes energy:
 - Reduce/aggregate number of messages.
 - Redesign communication protocols.
(e.g. minimize wait-state)
 - Better match between messages sending and communication protocol.



App	Run-time	#Routine calls (#Threads)	% Battery	3rd-Party Modules Used	<i>Where is the energy spent inside an app?</i>
browser	30s	1M (34)	0.35%	-	38% HTTP; 5% GUI; 16% user tracking; 25% TCP cond.
angrybirds	28s	200K (47)	0.37%	Flurry[7],Khronos[41]	20% game rendering; 45% user tracking; 28% TCP cond.
fchess	33s	742K (37)	0.60%	AdWhirl[42]	50% advertisement; 20% GUI; 20% AI; 2% screen touch
nytimes	41s	7.4M (29)	0.75%	Flurry[7],JSON[43]	65% database building; 15% user tracking; 18% TCP cond.
mapquest	29s	6M (43)	0.60%	SHW[44],AOL,JSON[43]	28% map tracking; 20% map download; 27% rendering

Conclusions

- Hardware Evolution
- Ubiquity of Devices
- Vision: Invisible Computing
- Challenges to Make Invisible Computing a Reality

- Context-awareness
- Adaptability
- Cyberforaging
- Resource Discovery
- Offloading
- Replication
- Battery Usage
- Security
- Etc.

Applications
Middleware
Operating System

Mobile and Ubiquitous Computing

2022-2023

**Definitions, Ubiquitous and
Pervasive Computing Overview**

Evolution

- Hardware evolution
- Ubiquity of devices
- Ubiquity nightmare
- Invisible computing
- Some examples



Specifications



Ring

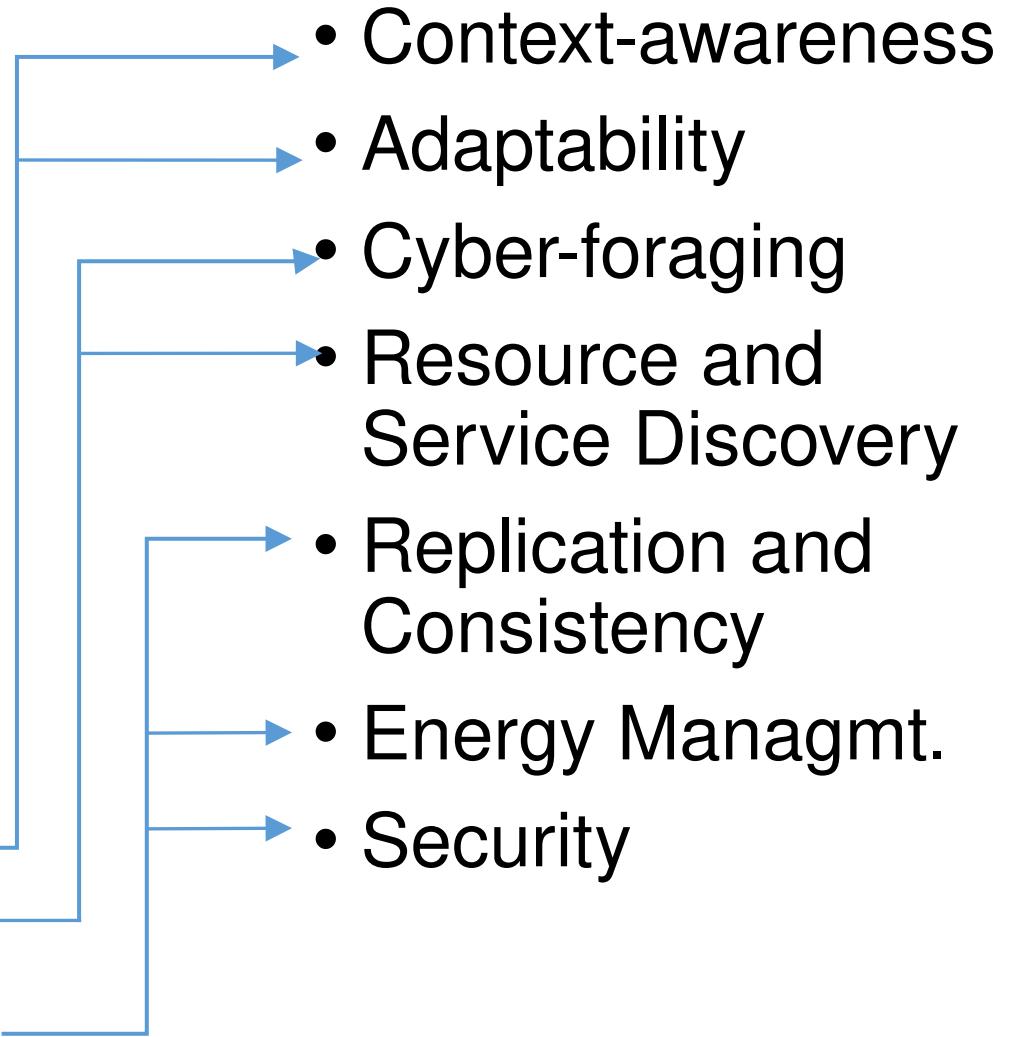
Ring Size

We are planning to provide 6 different sizes for Ring and will contact you to inquire which size you desire.



Goal/Requirements and Challenges

- Goal: invisible computing
- Requirements common to other areas:
 - Scalability in the large
 - Performance
 - Availability
- Requirements specific to CMU:
 - Support Variability
 - Deal with Resource Constraints
 - Provide Constant Access to Devices
 - Support Localized Scalability



Basic Definitions

- Distinction from classical distributed systems
- Fundamental concepts
- Mobile computing
- Ubiquitous computing
- Pervasive computing
- Localized scalability
- Smart spaces

Mobile Computing

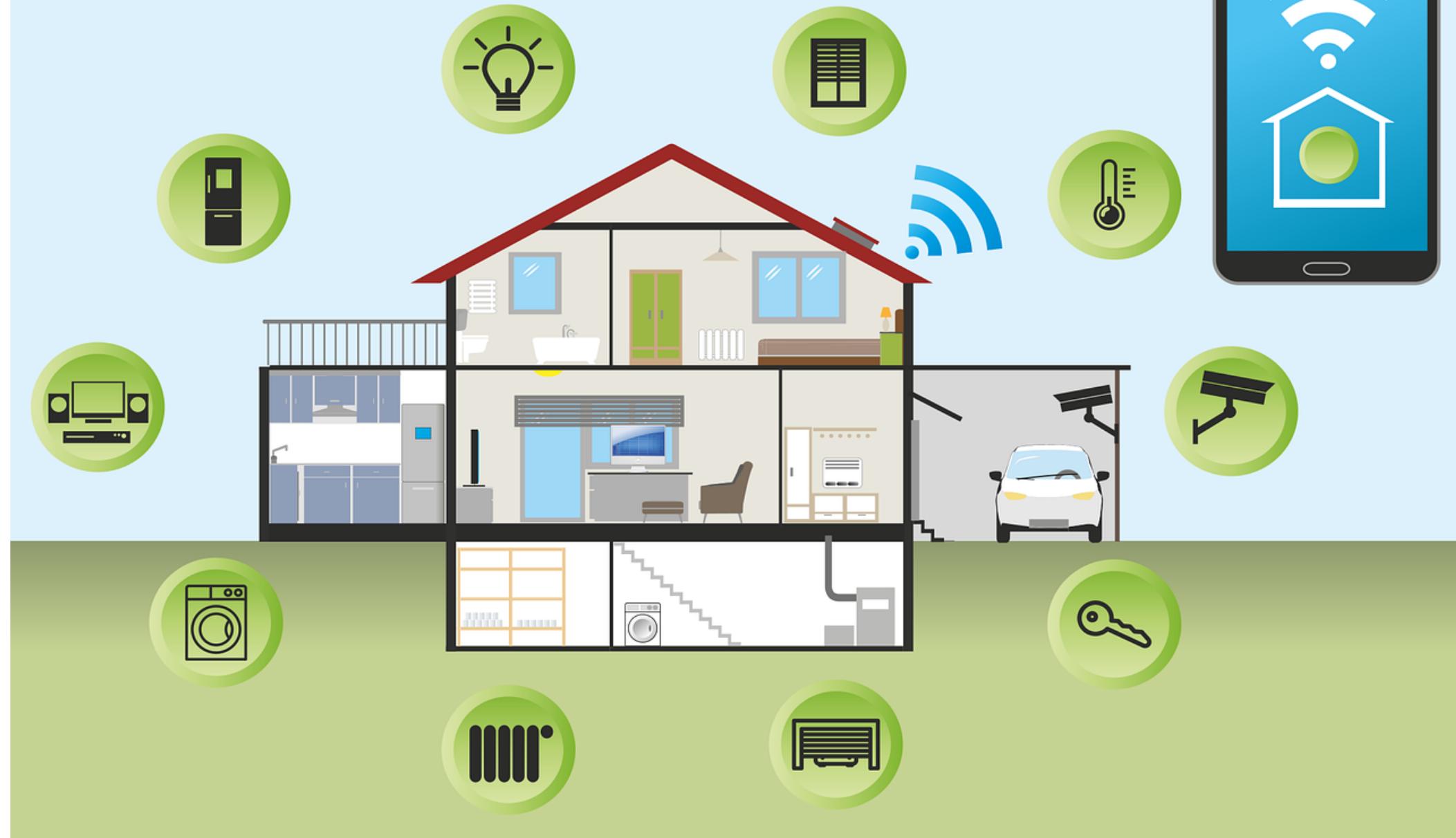
- Mobile computing is a computer science and engineering domain that deals with:
 - computing and communication software and hardware aspects
 - related to the use of mobile devices (e.g smartphones, tablets, laptops, etc.)
- This domain includes several areas:
 - We are concerned with **software** (hardware and network protocols are out of the scope in this course):
 - Middleware, sensors, distributed support, application support.
- Thus, in the scope of this course, we refer to mobile and ubiquitous computing to:
 - designate all **system-level software issues** related to the architectural design of solutions, including applications, middleware and operating systems,
 - which allow a **mobile** device to run mobile applications efficiently, while being scalable, secure, and energy efficient.

Ubiquitous Computing

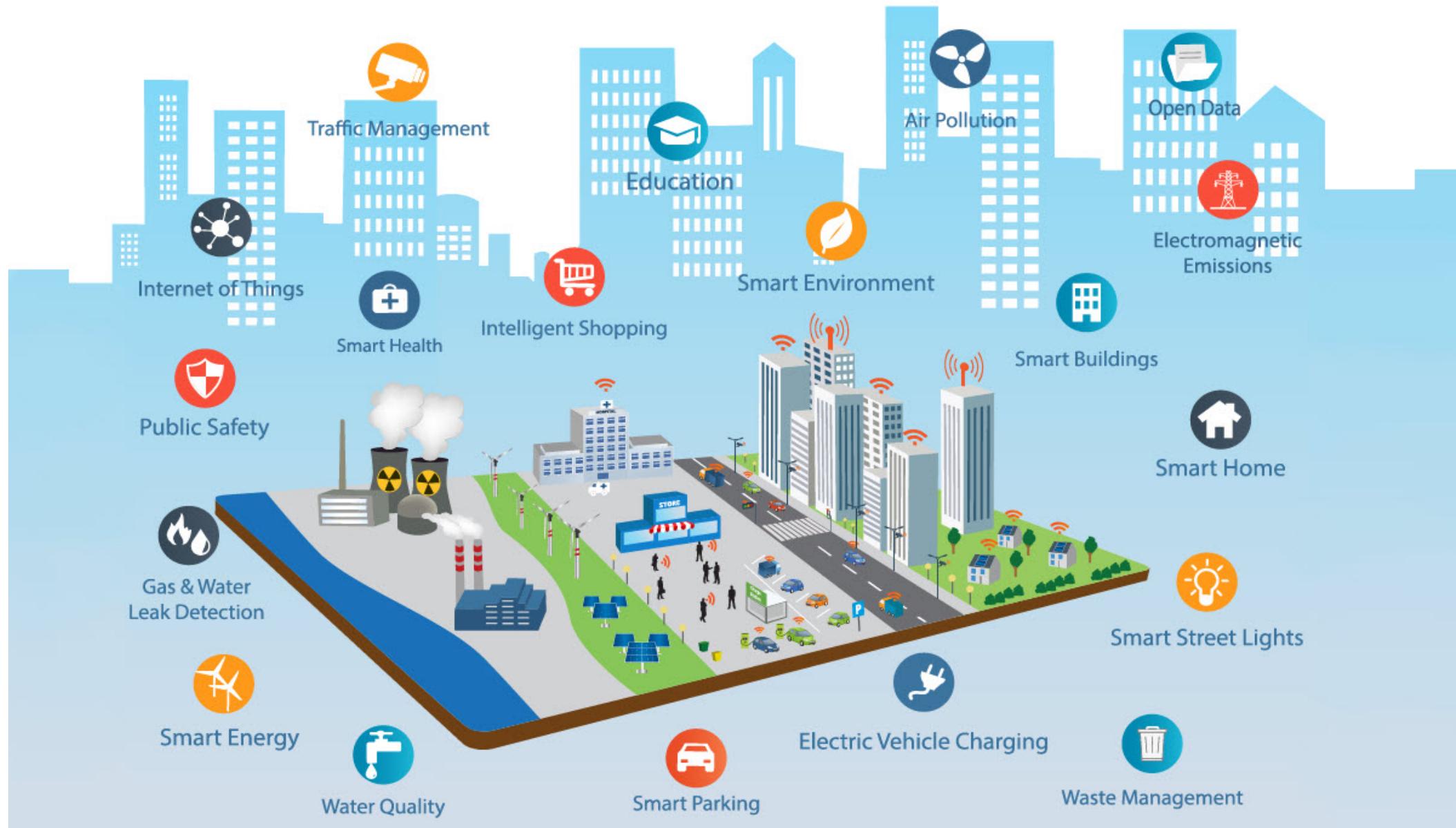
- Sometimes also designated as pervasive computing.
- Hardware evolution has created a large number of devices with a large variety in sizes, characteristics, processing power, etc. (e.g. tablets, laptops, smart-watches, etc.):
 - thus contributing to the **ubiquity of computing and communication devices** in the world
- Specifically, ubiquitous computing is a computer science and engineering domain that deals with:
 - all computing and communication software and hardware aspects
 - related to the use of ubiquitous devices
- Given that most of such ubiquitous devices are mobile, it is clear that there is an overlap between these two fundamental concepts, but:
 - in some cases, we may be in presence of an ubiquitous system in which the devices are not mobile
 - thus, we can have an ubiquitous system that is not mobile and, on the other way around, a mobile system which is not ubiquitous

Examples of Ubiquitous Environments

Smart Home



SMART CITY





Pervasive Computing

- In the literature, sometimes there is no distinction between ubiquitous and pervasive computing.
- Pervasive computing implies the **embedding of computing devices into everyday analog objects**:
 - e.g. the smart-mug that changes its color according to the temperature of the liquid it contains.
- One could be in a pervasive computing environment which is not ubiquitous:
 - unless, there are several “stupid” objects, with increased embedded computing capabilities, all over.

Pervasive Computing Examples



Mobile, Ubiquitous, and Pervasive Computing

- The concepts underlying mobile, ubiquitous and pervasive computing are mostly related to the properties of:
 - mobility (i.e., devices can be easily moved from one place to another by the user),
 - ubiquity (i.e., devices are everywhere), and
 - pervasiveness (i.e., “stupid” everyday objects with embedded computing devices), respectively.
- The most interesting scenarios are those in which a mix of these properties exist.
- The challenges being addressed, previously mentioned, apply to all such cases as they all share a common view:
 - “information at your fingertips anywhere, anytime”.

Localized Scalability and Smart Spaces

- It is the equivalent to the concept of scalability when applied to classic distributed systems:
 - the difference is that we are concerned with the large number of devices that may co-exist in a confined/small space (e.g., in a room)
- Localized scalability means that a system must scale in the local space:
 - i.e., it must be able to handle a growing amount of devices and the resulting interaction
- Such a room is also called a smart-space:
 - its “smartness” results from the existence of a large number of computing devices that, while being invisible, perform the work needed

Joseph Marie Jacquard (1752-1834)



Example: Jacquard (1/2)

Welcome to Project Jacquard - <https://www.youtube.com/watch?v=qObSFfdfe7I>

Example: Jacquard (2/2)

Levi's® Commuter™ x Jacquard by Google Trucker Jacket
- <https://www.youtube.com/watch?v=yJ-lcdMfziw>

Mobile and Ubiquitous Computing

2022-23

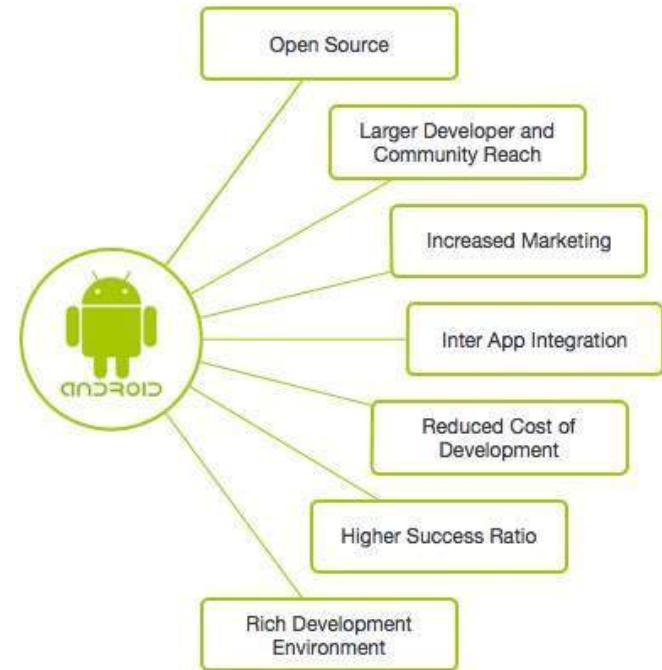
MEIC-A, MEIC-T, METI
Introduction to Android



ANDROID BASICS

The Beginning

- Android was the flagship product of Android Inc.
- it was a **small startup founded in 1999** to develop a new user-friendly location-aware mobile platform
- **acquired by Google in 2005**
- The **first Android** phone ran Android version 1.0:
- it was launched in **September 2008** (HTC Dream)
- Android rapid evolution:
 - fast sequence of **updates**
 - growing number of **partners** launching their own mobile devices
 - also used across a wide **variety of mobile devices** (e.g. tablets, mp3 players, netbooks, in-flight entertainment systems)
 - Android runs around **2000 million devices** around the world and is still growing at an unparalleled rate

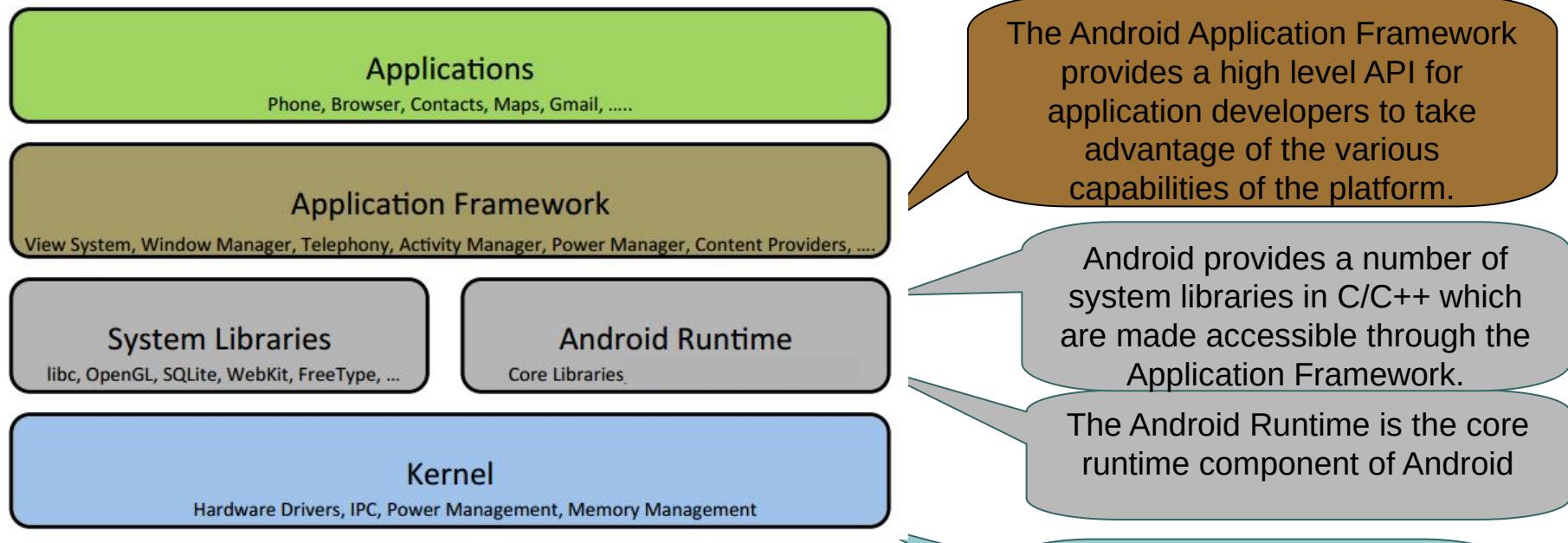


Android Main Features

Feature	Description
Beautiful UI	Android OS basic screen provides a beautiful and intuitive user interface.
Connectivity	GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC and WiMAX.
Storage	SQLite, a lightweight relational database, is used for data storage purposes.
Media support	H.263, H.264, MPEG-4 SP, AMR, AMR-WB, AAC, HE-AAC, AAC 5.1, MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, and BMP
Messaging	SMS and MMS
Web browser	Based on the open-source WebKit layout engine, coupled with Chrome's V8 JavaScript engine supporting HTML5 and CSS3.
Multi-touch	Android has native support for multi-touch which was initially made available in handsets such as the HTC Hero.

Feature	Description
Multi-tasking	User can jump from one task to another and same time various application can run simultaneously.
Resizable widgets	Widgets are resizable, so users can expand them to show more content or shrink them to save space
Multi-Language	Supports single direction and bi-directional text.
GCM	Google Cloud Messaging (GCM) is a service that lets developers send short message data to their users on Android devices, without needing a proprietary sync solution.
Wi-Fi Direct	A technology that lets apps discover and pair directly, over a high-bandwidth peer-to-peer connection.
Android Beam	A popular NFC-based technology that lets users instantly share, just by touching two NFC-enabled phones together.

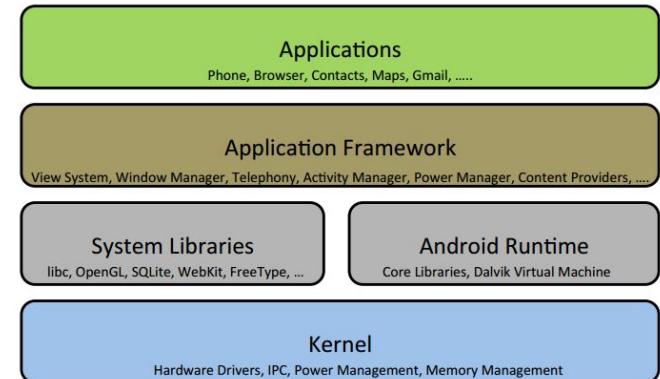
Android Overall Architecture (1/2)



- Android is a **Linux-based platform**
- It uses a heavily **customized Linux kernel**
- It is **not another flavor of Linux** because:
 - it **does not support the complete set of standard GNU libraries**, and
 - it uses its own **proprietary windowing system** instead of X-Windows

Android Linux Kernel

- Contains:
 - the hardware abstraction layer (**HAL**)
 - components for **memory management** and **inter-process communication** amongst other low-level functionalities
- It provides:
 - **drivers** for the display, touch input, networking, power management and storage
- One of the main features is:
 - YAFFS2 (Yet Another Flash File System 2) which enables support for **flash-based file systems**.
- Another important feature is the **WakeLock**:
 - it can be used to force a device from going into a low power state
 - it is useful from a user experience perspective since it makes the device more responsive to user interaction
 - it is mostly used by applications and services to request CPU resources
 - it implies that if the OS does not have any active WakeLocks, then the CPU will be shutdown
- Binder:
 - proprietary mechanism for **IPC** (RMI included)



Android Runtime

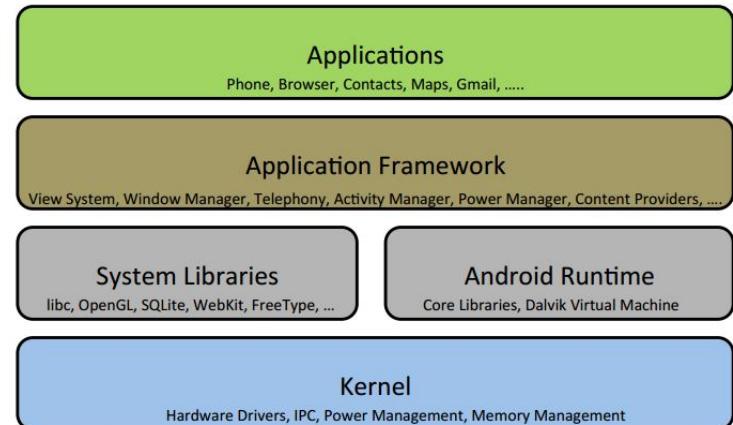
- It is a **process-based VM** optimized for low memory footprint and performance efficiency

- Applications are written in a **dialect of Java**:

- compiled into bytecodes
- stored as JVM compatible .class files

- Differences between standard Java and Android APIs:

- absence of Abstract Window Toolkit (AWT)**, and
Swing libraries in Android



- The JVM class files are converted into **Dalvik Executable (.dex)** files:

- these are executed by the Android Runtime when the application runs on Android

- Each Android application executes within its own instance of the **Android Runtime**:

- in turn the Android Runtime runs as a **kernel managed process**

- Ensures that multiple instances of the VM can run at the same time:

- this **sandboxed** approach ensures security, and if an application requires access to data outside its own sandbox (e.g. such as contacts, text messages, Bluetooth communications)
- it needs to explicitly request permission during installation on a specific device

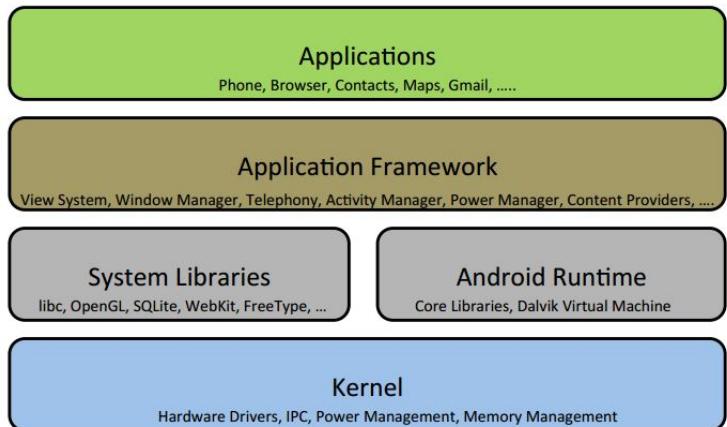
System Libraries

- Android system libraries:

- in C/C++
- made accessible through the Application Framework
- do not provide the complete functionality required of the standard GNU C libraries in Linux

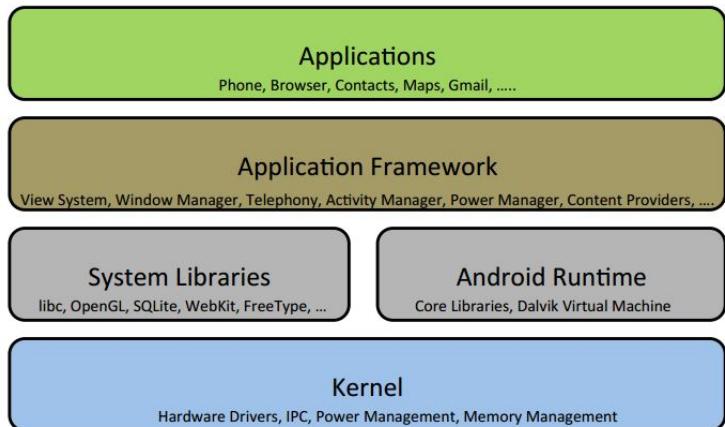
- Some examples:

- libc typical of C/C++ environments
- FreeType library provides bitmap and vector-based font rendering
- SQLite library (database capabilities)
- OpenGL/ES for 2D and 3D rendering
- Scalable Graphics Library (2D rendering tasks)
- compositing of 2D and 3D content is handled by the Surface Manager library
- LibWebCore library provides a WebKit-based browser engine which can be embedded as a web view within user interfaces of other applications
- Android Media Library (based on the OpenCORE multimedia framework) handles both images and multimedia content:
 - provides capabilities for recording and playback of commonly used audio, video and still-image formats (e.g. MP3, AAC, H.264, MP4, JPG and PNG)



Application Framework (1/5)

- **Activity Manager** – Controls all aspects of the application lifecycle and activity:
 - interacts with the overall activities running in the system.



activity: <http://developer.android.com/reference/android/app/ActivityManager.html>

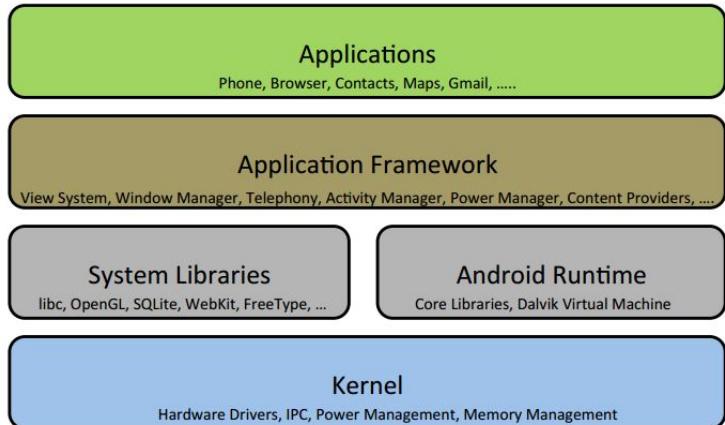
- **ContentProvider** - allows applications to access and share data with other applications:
 - allows an application to publish its data for access by others
 - the information is provided through a single ContentResolver interface

content providers: <http://developer.android.com/guide/topics/providers/content-providers.html>

Application Framework (2/5)

◦ **Resource Manager** - Provides access to non-code embedded resources such as strings, color settings and user interface layouts:

- resources can be animations, layouts, strings and even image files
- allow to customize the application based on the type of device it is deployed in or various device settings, such as locale information and language



resources: <http://developer.android.com/guide/topics/resources/index.html>

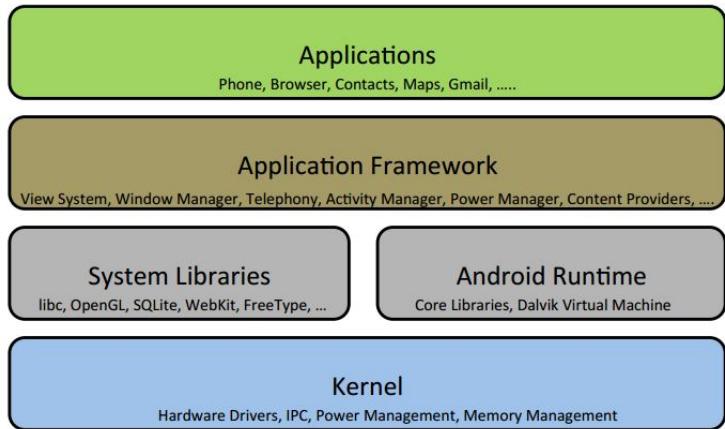
◦ **Notifications Manager** – Allows applications to display alerts and notifications to the user:

- any application can notify the user about specific events when they occur
- handled by the NotificationManager class, which allows an application to issue notifications (e.g. an icon on the status bar, flashing the device LEDs or the backlight of the device play or by playing a sound or vibrating the device)

notifications: <http://developer.android.com/reference/android/app/NotificationManager.html>

Application Framework (3/5)

- **View System** – An extensible set of views used to create application user interfaces
- It provides the basic building blocks for creating the user interface of an application:
 - a View object in Android represents a rectangular region of the device's display and is responsible for all the rendering and event handling tasks within that region
 - visual user interface elements in Android are called Widgets and are derived from the View class

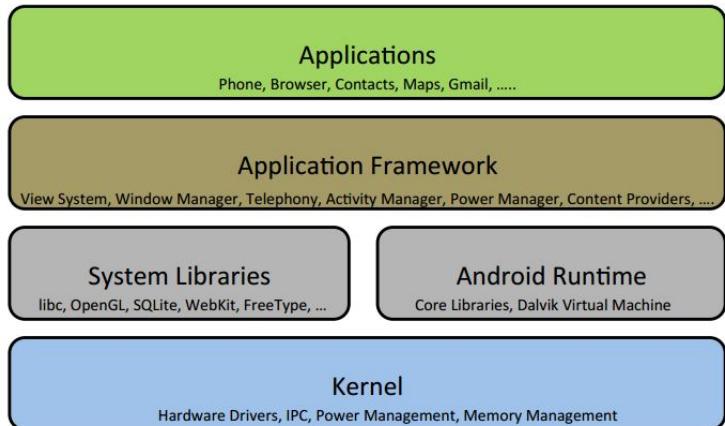


user interface: <http://developer.android.com/guide/topics/ui/index.html>

Application Framework (4/5)

◦ **LocationManager** - provides access to the localization services available on the device:

- supports mapping capabilities resulting from the availability of free high-quality turn-by-turn navigation and map applications such as Google Maps
- third-party developers can utilize this class to create location-based services (LBS)



location:<http://developer.android.com/reference/android/location/LocationManager.html>

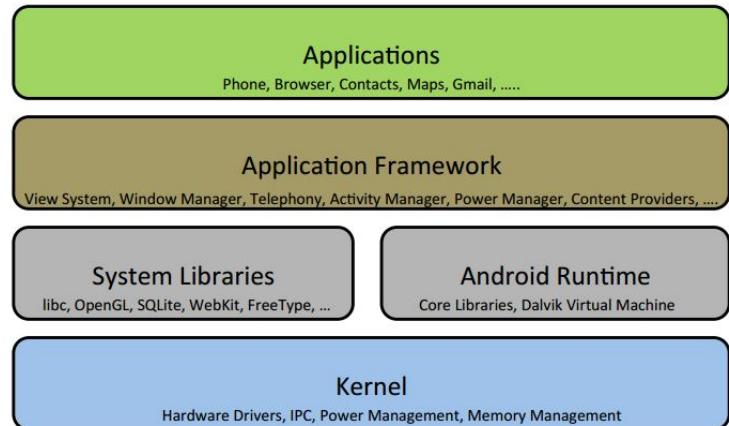
◦ **InputMethodService** - enables to implement custom software keyboards, keypads and even pen input:

- input is converted into text and passed on to the target UI element

input:<http://developer.android.com/reference/android/inputmethodservice/InputMethodService.html>

Application Framework (5/5)

- **TelephonyManager** class provides applications with the ability to determine telephony services on the devices and access specific subscriber information
- **SmsManager** allows applications to send data and text messages using the (SMS) protocol



telephony: <http://developer.android.com/reference/android/telephony/TelephonyManager.html>

- **PowerManager** class provides applications with the ability to control the power state of the device and use a feature called WakeLocks:
 - a WakeLock forces a device to remain on and not go into power-saving mode
 - can provide superior user experience by ensuring the app's interface is immediately responsive to user actions
 - misuse of this capability can lead to extremely poor battery life on the device

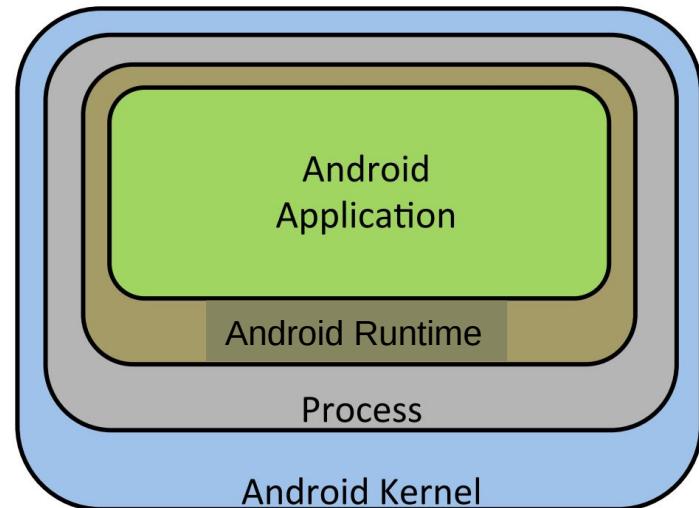
power: <http://developer.android.com/reference/android/os/PowerManager.html>



ANDROID APPLICATIONS

What Kind of System Entity is an Android Application?

- Android applications written in Java, executed by the Android Runtime (VM)
- Android is not fully compliant with standard Java:
 - there are major differences especially in the user interface libraries
- Each application runs in a sandboxed environment:
 - with its own instance of an Android Runtime which runs within its own kernel managed process
- Android application is installed as a single file of type Android Package (extension: .apk):
 - it contains the compiled code along with data and resource files
- Android package files can be signed or unsigned



What are the Components of an Application?

- Four types of components:



- Components communicate through Intents

What do the Main Application Components Do?

◦ Activities

- Represent a **screen** with a visual user interface (Activity class)
- Dictate the UI and handle the **user interaction** to the smart phone screen

```
public class MainActivity extends Activity {  
}
```

◦ Services:

- Used for **background tasks** such as **time intensive** tasks or inter-application functionalities which do not require direct user interaction (Service class).

```
public class MyService extends Service {  
}
```

◦ Content Providers:

- Enable applications to **store and share data** with other applications (ContentProvider class)
- they handle data and database management issues.

```
public class MyContentProvider extends ContentProvider {  
  
    public void onCreate(){  
  
    }  
}
```

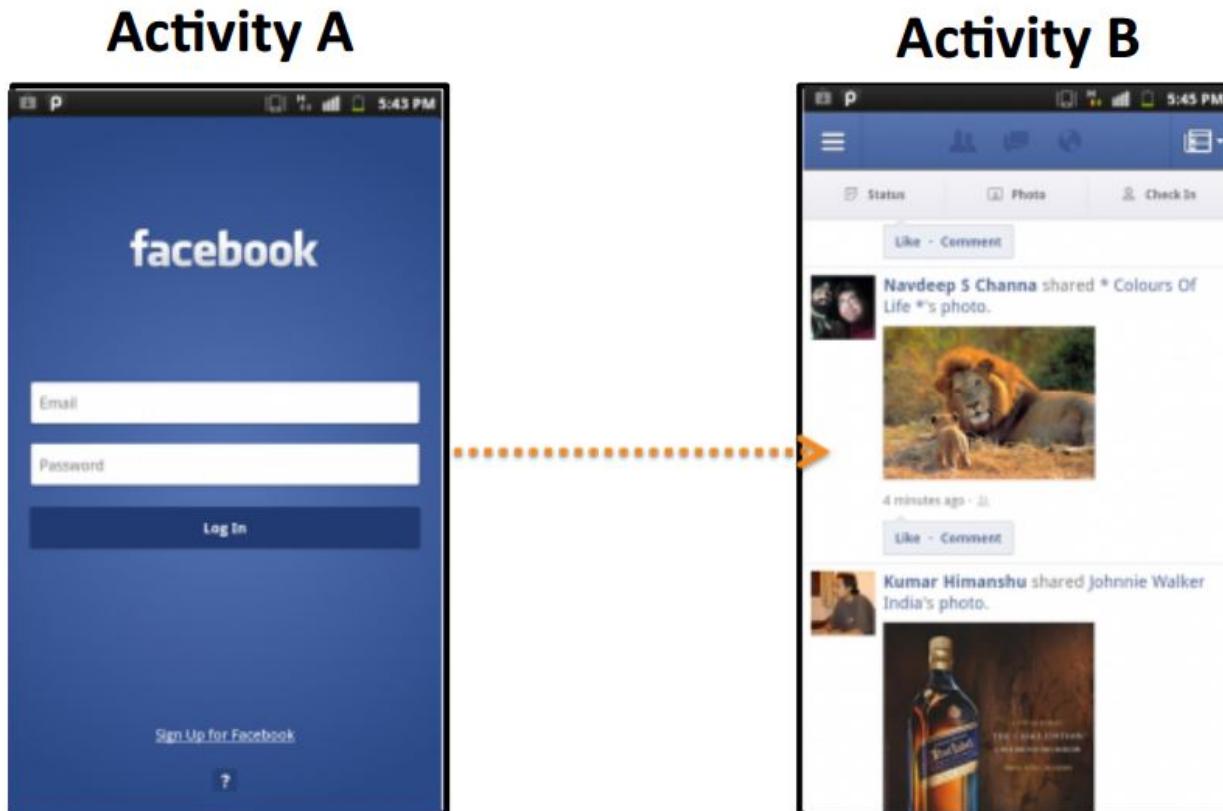
◦ Broadcast Receiver:

- Respond to system-wide broadcast announcements (BroadcastReceiver class)
- Handle communication between Android OS and applications.

```
public class MyReceiver extends BroadcastReceiver {  
  
    public void onReceive(context,intent){  
  
    }  
}
```

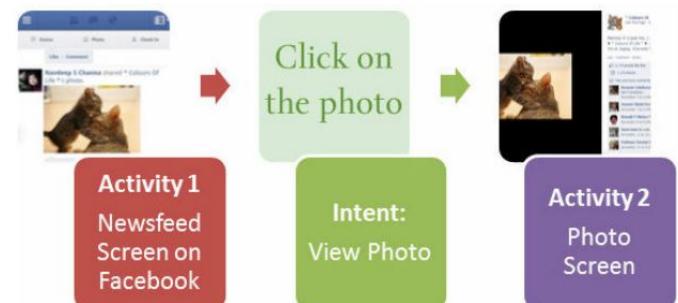
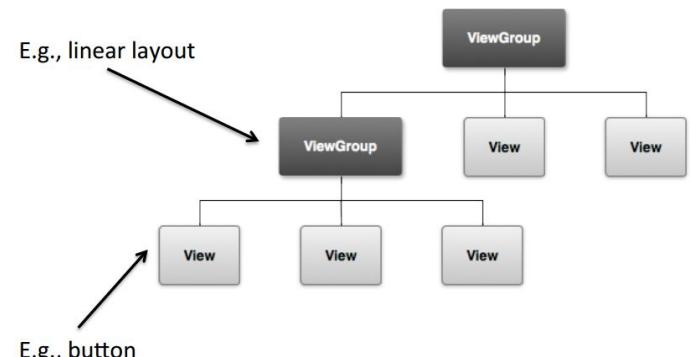
How do Activities Relate to Applications?

- A typical Android application consists of one or more activities
- Launching an app results in executing one predefined activity (called main activity)
- Facebook App:
 - Activity A allows user to login on Facebook
 - Activity B displays user's Facebook wall

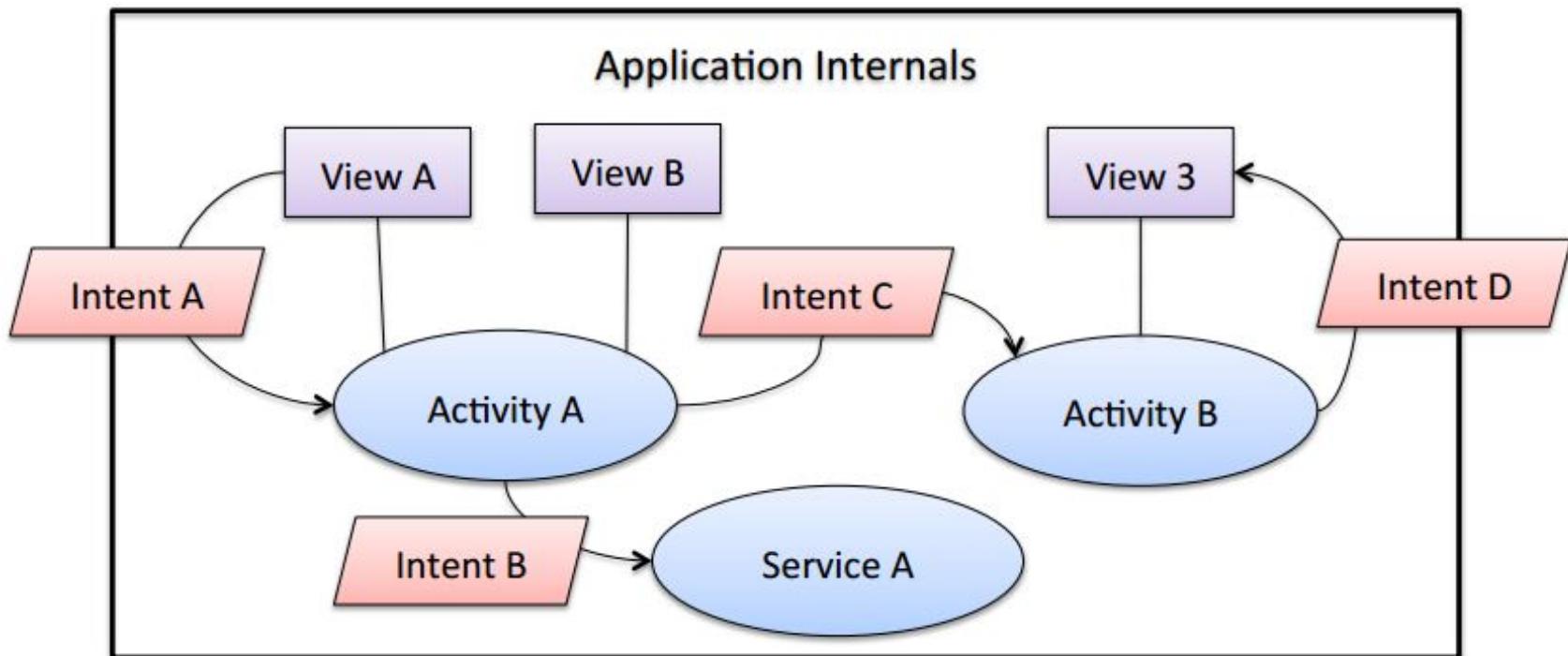


Activities, Views, and Intents: Why are they the base of an application's logic?

- Activities are “similar” to web pages:
 - they provide an interface for users to interact with an app and take actions
- Views are UI elements hierarchically organized in two types:
 - layouts, and widgets
- Intents are similar to messages that enable communication across components



Activities, Views, and Intents: How do they combine to form an application?



Components

Modules

Views

UI elements (e.g., button)

Intents

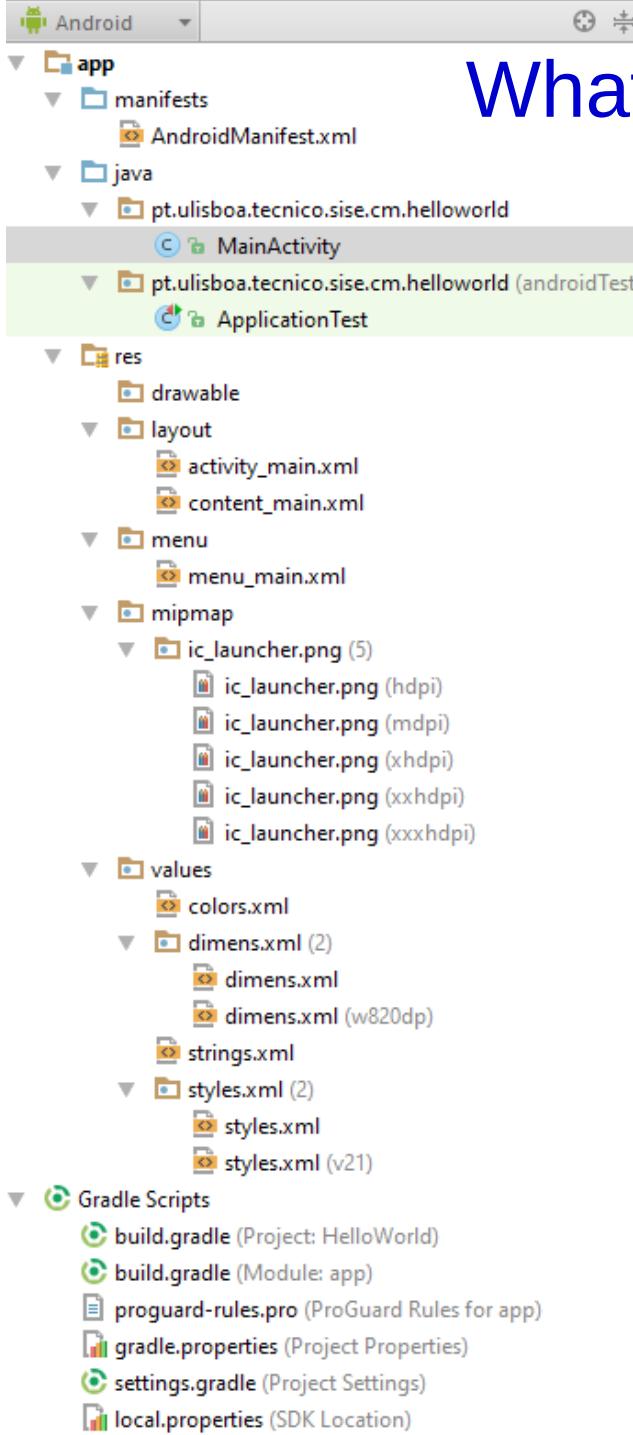
Messages

How do Activities Relate to Each Other?

- If an **application** has **multiple** user interface **screens**, then it will contain multiple activity components, e.g., a music player application user interface consists of:
 - one screen for audio playback, and
 - one screen for selecting albums or audio files will have two activity components
- Any **application** can ask the Application Framework to **start** a specific activity in **another application**, e.g. an email application receives a message with an MP3 audio attachment:
 - then, it requests the framework to start the activity representing the music player's audio playback screen
 - so that the user can listen to the contents of the attachment
 - this is done asynchronously through the use of Intents
 - an Intent object provides an abstract description of an action be performed in the form of launching an Activity component or a specific type of component



DEVELOPMENT AND TESTING

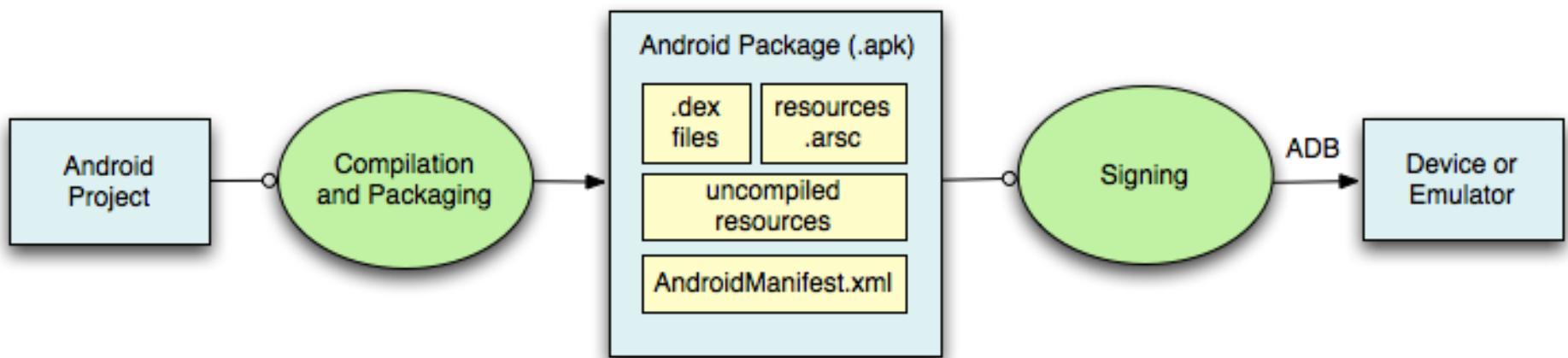


What does an Android Project look like?

S.N.	Folder, File & Description
1	src This contains the .java source files for your project. By default, it includes an <i>MainActivity.java</i> source file having an activity class that runs when your app is launched using the app icon.
2	gen This contains the .R file, a compiler-generated file that references all the resources found in your project. You should not modify this file.
3	bin This folder contains the Android package files .apk built by the ADT during the build process and everything else needed to run an Android application.
4	res/drawable-hdpi This is a directory for drawable objects that are designed for high-density screens.
5	res/layout This is a directory for files that define your app's user interface.
6	res/values This is a directory for other various XML files that contain a collection of resources, such as strings and colours definitions.
7	AndroidManifest.xml This is the manifest file which describes the fundamental characteristics of the app and defines each of its components.

What Does the Android Development Process Look Like?

- The Android build process provides:
 - project and module build settings so that
 - your Android modules are compiled and packaged into .apk files, the containers for your application binaries, based on your build settings
- The apk file for each app contains all of the information necessary to run your application on a device or emulator, such as:
 - compiled .dex files (.class files converted to Dalvik byte code),
 - a binary version of the AndroidManifest.xml file,
 - compiled resources (resources.arsc) and uncompiled resource files for your application



What is the Entry Point for an Android App?

The Main Activity

- The main activity code is a Java file MainActivity.java:

- this is the actual application file which ultimately gets converted to a DEX executable and runs your application
- R.layout.activity_main refers to the activity_main.xml file located in the res/layout folder
- the onCreate() method is one of many methods that are executed when an activity is loaded.

code generated by the application wizard for Hello World application

```
package pt.ulisboa.tecnico.sise.cm.helloworld;  
  
import android.os.Bundle;  
import android.support.design.widget.FloatingActionButton;  
import android.support.design.widget.Snackbar;  
import android.support.v7.app.AppCompatActivity;  
import android.support.v7.widget.Toolbar;  
import android.util.Log;  
import android.view.View;  
import android.view.Menu;  
import android.view.MenuItem;  
  
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);  
        setSupportActionBar(toolbar);  
  
        Log.d("MainActivity", "This is my first debugging message.");  
    }  
}
```

How are App Components Organized?

Manifest File

- All components present in an application must be declared:
 - this is done in a manifest file named `AndroidManifest.xml`
- For each component, the manifest file can also declare an **Intent Filter**:
 - to list its capabilities so that it can respond to intents of specific types
- Apart from this, the manifest file also:
 - declares the **user permissions** required by the application during run time such as access to user data or network access
 - specifies the **hardware and software services** required and the **external libraries** that need to be linked with the application
 - specifies the **API Level** used by the application (given that Android has been evolving rapidly; e.g. with four major versions launched within the last three years)
- API level corresponds to a release version of the Android platform:
 - e.g.: the API level associated with the Ice Cream Sandwich release of Android (version 4.0) is 14

Manifest File (1/3)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="pt.ulisboa.tecnico.sise.cm.helloworld">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name"
            android:theme="@style/AppTheme.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

- AndroidManifest.xml defines the skeleton of an application
- Whatever component you develop as a part of your application:
 - you must declare all its components in a *manifest.xml*
 - this file resides at the root of the application project directory
 - this file works as an interface between Android OS and your application
 - if you do not declare your component in this file, then it will not be considered by the OS

Manifest File (2/3)

- <application>...</application> enclose the components related to the application:
- Attribute android:icon points to the application icon available under res/drawable-hdpi
- the application uses the image named ic_launcher.png located in the drawable folders.

- The <activity> tag is used to specify an activity:

- android:name attribute specifies the fully qualified class name of the Activity subclass

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="pt.ulisboa.tecnico.sise.cm.helloworld">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name"
            android:theme="@style/AppTheme.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

- android:label attribute specifies a string to use as the label for the activity

- You can specify multiple activities using <activity> tags

Manifest File (3/3)

- The action for the intent filter is named android.intent.action.MAIN:
 - it indicates that this activity serves as the entry point for the application
- The category for the intent-filter is named android.intent.category.LAUNCHER:
 - it indicates that the application can be launched from the device's launcher icon

- @string/app_name

refers to the app_name
string defined in the
strings.xml file, which is
"HelloWorld"

- The @string refers to
the strings.xml file
(see next slide)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="pt.ulisboa.tecnico.sise.cm.helloworld">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name"
            android:theme="@style/AppTheme.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

- List of tags in the manifest file to specify different Android application components:

- <activity> elements for activities
- <service> elements for services
- <receiver> elements for broadcast receivers
- <provider> elements for content providers

Strings File

- strings.xml file is located in the res/values folder:
- it contains all the text that your application uses
- e.g., the names of buttons, labels, default text, and similar types of strings go into this file
- this file is responsible for their textual content

```
<resources>
    <string name="app_name">Hello World</string>
    <string name="action_settings">Settings</string>
</resources>
```

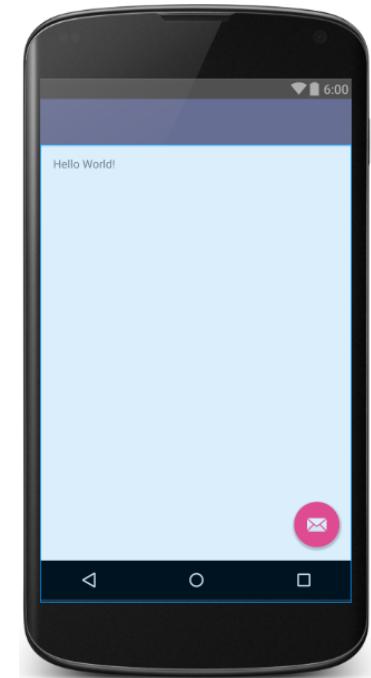
Layout File

- The activity_main.xml / content_main.xml is a layout file available in res/layout directory:
- it is referenced by your application when building its interface
- you will modify this file very frequently to change the layout of your application

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:context=".pt.ulisboa.tecnico.sise.cm.helloworld.MainActivity"
    tools:showIn="@layout/activity_main">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
</RelativeLayout>
```

"Hello World"
application
layout file



- The TextView is an Android control used to build the GUI:
 - it has various attributes like android:layout_width, android:layout_height, etc
 - these are being used to set its width and height etc..
- The @string refers to the strings.xml file located in the res/values folder:
 - hence, @string/hello_world refers to the hello string defined in the strings.xml file, which is "Hello World"

Other Resources

- There are many more items which you use to build a good Android application

- Apart from coding for the application, you take care of various other resources like static content that your code uses, such as:

- bitmaps,
- colors,
- layout definitions,
- user interface strings,
- animation instructions,
- etc.

- These resources are always maintained separately in various sub-directories under res/ directory of the project

```
MyProject/
    src/
        MyActivity.java
    res/
        drawable/
            graphic.png
        layout/
            main.xml
            info.xml
        mipmap/
            icon.png
        values/
            strings.xml
```

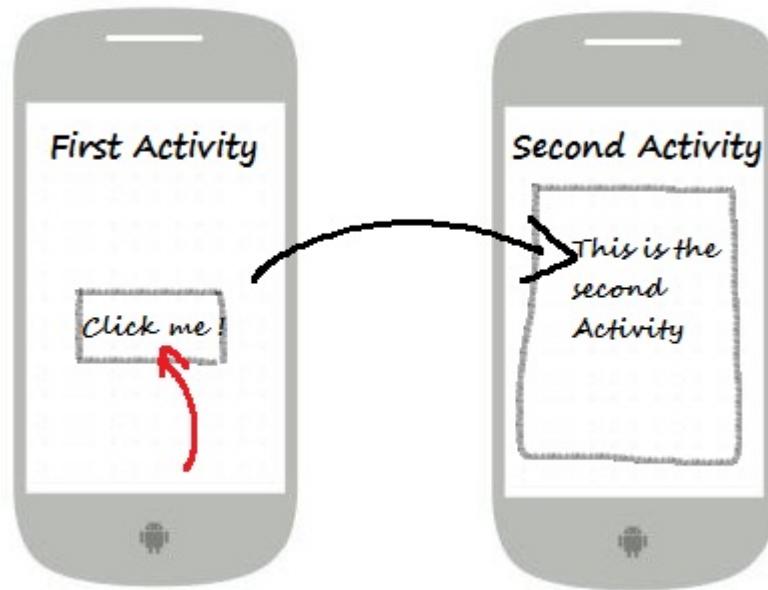
Resources in Android Studio

Resource directories supported inside project res/ directory

Directory	Resource Type
animator/	XML files that define property animations .
anim/	XML files that define tween animations . (Property animations can also be saved in this directory, but the <code>animator/</code> directory is preferred for property animations to distinguish between the two types.)
color/	XML files that define a state list of colors. See Color State List Resource
drawable/	Bitmap files (<code>.png</code> , <code>.9.png</code> , <code>.jpg</code> , <code>.gif</code>) or XML files that are compiled into the following drawable resource subtypes: <ul style="list-style-type: none">• Bitmap files• Nine-Patches (re-sizable bitmaps)• State lists• Shapes• Animation drawables• Other drawables See Drawable Resources .
mipmap/	Drawable files for different launcher icon densities. For more information on managing launcher icons with <code>mipmap/</code> folders, see Managing Projects Overview .
layout/	XML files that define a user interface layout. See Layout Resource .
menu/	XML files that define application menus, such as an Options Menu, Context Menu, or Sub Menu. See Menu Resource .
raw/	Arbitrary files to save in their raw form. To open these resources with a raw <code>InputStream</code> , call <code>Resources.openRawResource()</code> with the resource ID, which is <code>R.raw.filename</code> . However, if you need access to original file names and file hierarchy, you might consider saving some resources in the <code>assets/</code> directory (instead of <code>res/raw/</code>). Files in <code>assets/</code> are not given a resource ID, so you can read them only using <code>AssetManager</code> .
Directory	Resource Type
values/	XML files that contain simple values, such as strings, integers, and colors. Whereas XML resource files in other <code>res/</code> subdirectories define a single resource based on the XML filename, files in the <code>values/</code> directory describe multiple resources. For a file in this directory, each child of the <code><resources></code> element defines a single resource. For example, a <code><string></code> element creates an <code>R.string</code> resource and a <code><color></code> element creates an <code>R.color</code> resource. Because each resource is defined with its own XML element, you can name the file whatever you want and place different resource types in one file. However, for clarity, you might want to place unique resource types in different files. For example, here are some filename conventions for resources you can create in this directory: <ul style="list-style-type: none">• arrays.xml for resource arrays (typed arrays).• colors.xml for color values• dimens.xml for dimension values.• strings.xml for string values.• styles.xml for styles. See String Resources , Style Resource , and More Resource Types .
xml/	Arbitrary XML files that can be read at runtime by calling <code>Resources.getXML()</code> . Various XML configuration files must be saved here, such as a searchable configuration .

Accessing Resources

- You should always externalize application resources such as images and strings from your code:
 - so that you can maintain them independently
- You should also provide alternative resources for specific device configurations:
 - by grouping them in specially-named resource directories.
- At runtime, Android uses the appropriate resource based on the current configuration:
 - e.g., you might want to provide a different UI layout depending on the screen size or different strings depending on the language setting.
- Once you externalize your application resources:
 - you can access them using resource IDs that are generated in your project's R class
- Once you provide a resource in your application:
 - you can apply it by referencing its resource ID
 - all resource IDs are defined in your project's R class, which the aapt tool automatically generates



ACTIVITIES

Components Lifecycle

- Each application runs in a sandboxed environment with its own instance of an Android Runtime
- An Android application does not completely control the completion of its lifecycle:
 - hardware resources may become critically low and OS could order early termination of any process
- Components are different points through which the system can enter into an application.
- **Each component has a distinct lifecycle that defines how the component is created and destroyed.**
- **They all follow a master plan that consists of:**
 - **Begin:** respond to request to instantiate them
 - **End:** when instances are destroyed
 - **In between states:** depends on the component type

Activity Lifecycle (1/4)

- An activity can exist in essentially three states:

- Resumed, Paused, and Stopped

- **Resumed** (or Running):

- the activity is in the **foreground** of the screen and has user **focus**

- **Paused**:

- another activity is in the foreground and has focus, but this one is **still visible** i.e., another activity is visible on top of the paused one, and
- that activity is partially transparent or doesn't cover the entire screen
- a paused activity is completely alive (the Activity object is retained in memory, it maintains all state and member information, and remains attached to the window manager), but
- can be killed by the system in extremely low memory situations

- **Stopped**:

- the activity is **completely obscured** by another activity (it is in the "background")
- a stopped activity is also still alive (the Activity object is retained in memory, it maintains all state and member information, but it is not attached to the window manager)
- however, it is no longer visible to the user, and it can be killed by the system when memory is needed elsewhere

Activity Lifecycle (2/4)

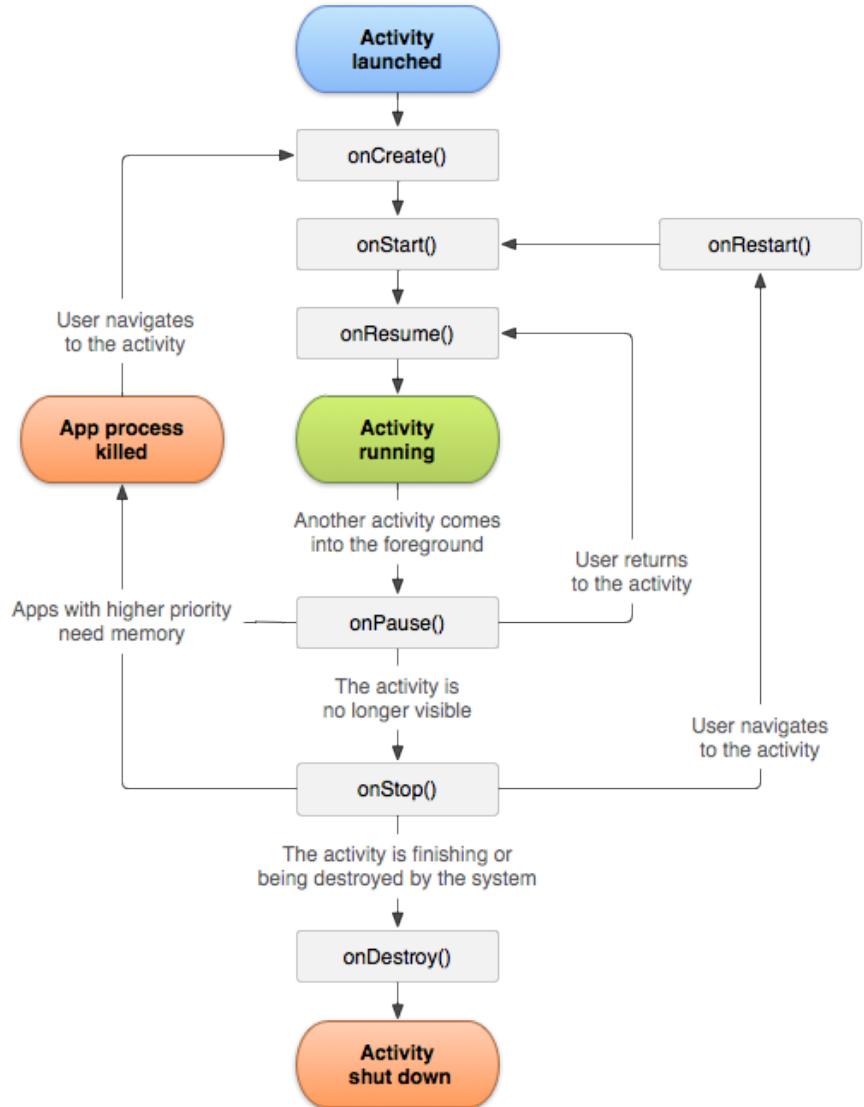
- An application usually consists of multiple activities that are loosely bound to each other
- Typically, one activity in an application is specified as the "main" activity:
 - it is presented to the user when launching the application for the first time.
- Each activity can then start another activity in order to perform different actions:
 - each time a new activity starts, the previous activity is stopped, but
 - the system preserves the activity in a stack (the "back stack")
- When a new activity starts, it is pushed onto the back stack and takes user focus
- The back stack abides to the basic "last in, first out" stack mechanism:
 - when the user is done with the current activity and presses the Back button, it is popped from the stack (and destroyed) and the previous activity resumes

Activity Lifecycle (4/4)

- To create an activity, you must:
 - create a subclass of Activity (or an existing subclass of it)
 - in your subclass, you need to implement callback methods that the system calls when the activity transitions between various states of its lifecycle
- The two most important callback methods are: onCreate and onPause
- **onCreate:**
 - you **must** implement this method
 - the system calls this when creating your activity
 - within your implementation, you should initialize the essential components of your activity
 - most importantly, this is where you must call **setContentView()** to define the layout for the activity's user interface
- **onPause:**
 - the system calls this method as the first indication that the user is leaving your activity (though it does not always mean the activity is being destroyed)
 - this is usually **where you should commit** any changes that should be persisted beyond the current user session (because the user might not come back)

Activity Lifecycle Timings (1/2)

- Entire lifetime of an activity happens between:
 - the call to onCreate(), and
 - the call to onDestroy()
- Visible lifetime of an activity happens between:
 - the call to onStart(), and
 - the call to onStop()
- Foreground lifetime of an activity happens between:
 - the call to onResume(), and
 - the call to onPause()



Activity Lifecycle Methods

```
public class ExampleActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // The activity is being created.
    }

    @Override
    protected void onStart() {
        super.onStart();
        // The activity is about to
        // become visible.
    }

    @Override
    protected void onResume() {
        super.onResume();
        // The activity has become
        // visible (it is now "resumed").
    }
}
```

```
@Override
protected void onPause() {
    super.onPause();
    // Another activity is
    // taking focus (this
    // activity is about to
    // be
    // "paused").
}

@Override
protected void onStop() {
    super.onStop();
    // The activity is no
    // longer visible
    // (it is now "stopped")
}

@Override
protected void onDestroy() {
    super.onDestroy();
    // The activity is about
    // to be destroyed.
}
```

Tasks and Back Stack (1/6)

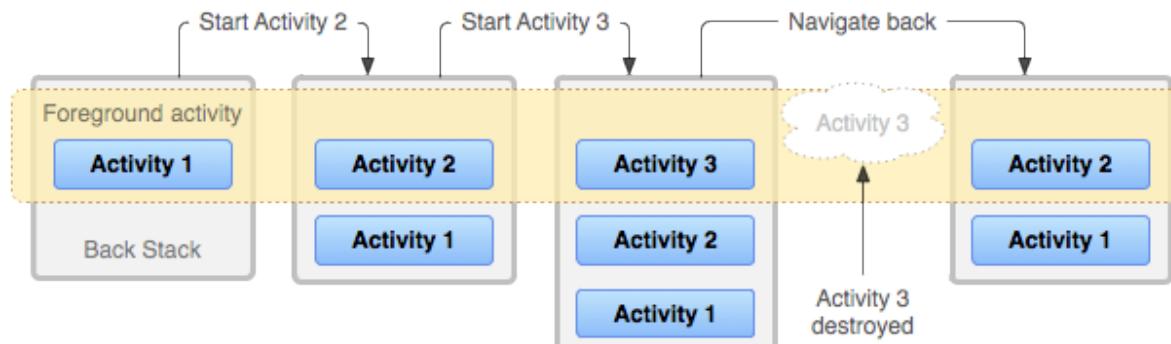
- An application usually contains multiple activities
- Each activity should be designed around a specific kind of action the user can perform and can start other activities:
 - e.g., an email application might have one activity to show a list of new messages
 - when the user selects a message, a new activity opens to view that message
- An activity can even start activities that exist in other applications on the device:
 - e.g. you can define an intent to perform a "send" action and include some data, such as an email address and a message
 - an activity from another application that declares itself to handle this kind of intent then opens
 - **the intent is to send an email, so an email application's "compose" activity starts (if multiple activities support the same intent, then the system lets the user select which one to use)**
 - when the email is sent, your activity resumes and it seems as if the email activity was part of your application
 - even though the activities may be from different applications, Android maintains this seamless user experience by keeping both activities in the same *task*

Tasks and Back Stack (2/6)

- A task is a collection of activities that users interact with when performing a job
- The activities are arranged in a stack (the *back stack*), in the order in which each activity is opened
- The device Home screen is the starting place for most tasks
- When the user touches an icon in the application launcher (or a shortcut on the Home screen):
 - that application's task comes to the foreground
- If no task exists for the application (the application has not been used recently), then:
 - a new task is created and the "main" activity for that application opens as the root activity in the stack

Tasks and Back Stack (3/6)

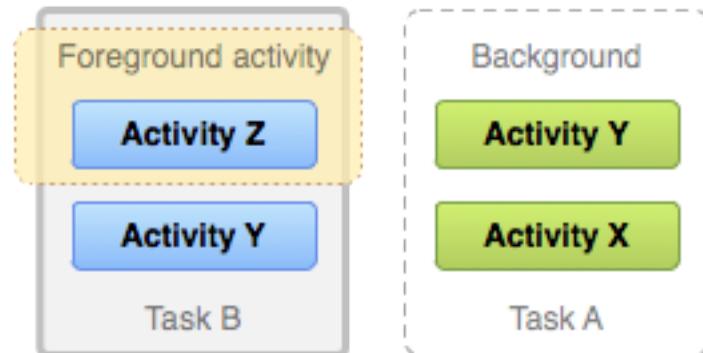
- When the current activity starts another:
 - the new activity is pushed on the top of the stack and takes focus
 - the previous activity remains in the stack, but is stopped
- When an activity stops, the system retains the current state of its user interface
- When the user presses the *Back* button:
 - the current activity is popped from the top of the stack (the activity is destroyed), and
 - the previous activity resumes (the previous state of its UI is restored)
- Activities in the stack are never rearranged, only pushed and popped from the stack:
 - pushed onto the stack when started by the current activity,
 - and popped off when the user leaves it using the *Back* button
- As such, the back stack operates as a "last in, first out" object structure



Tasks and Back Stack (4/6)

- If the user continues to press *Back*, then:
 - each activity in the stack is popped off to reveal the previous one, until the user returns to the Home screen (or to whichever activity was running when the task began)

- When all activities are removed from the stack, the task no longer exists



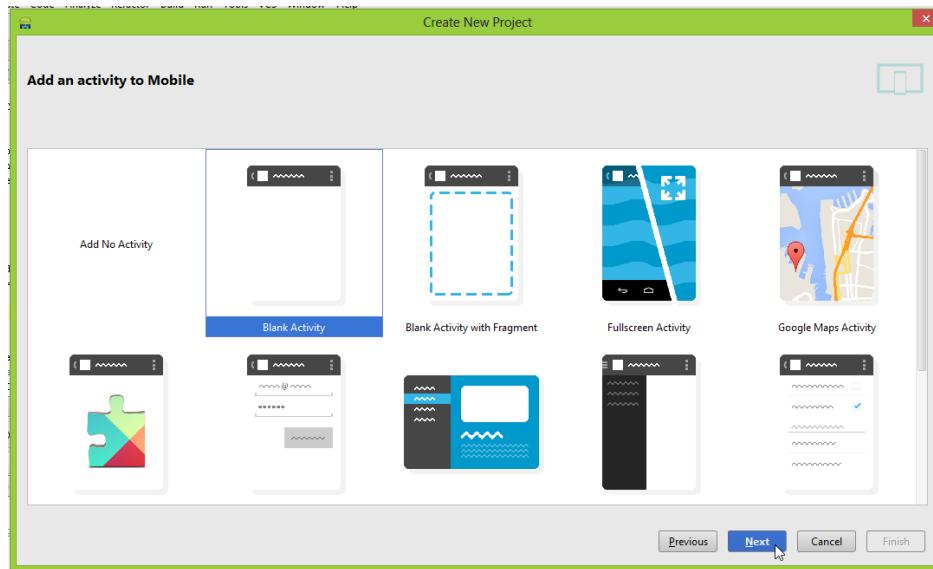
- A task is a cohesive unit that can move to the "background" when users begin a new task or go to the Home screen, via the *Home* button
- While in the background, all the activities in the task are stopped, but the back stack for the task remains intact:
 - the task has simply lost focus while another task takes place

Tasks and Back Stack (5/6)

- A task can return to the "foreground" so users can pick up where they left off
- Suppose, for example, that the current task (Task A) has three activities in its stack:
 - the user presses the *Home* button, then starts a new application from the application launcher
 - when the Home screen appears, Task A goes into the background
 - when the new application starts, the system starts a task for that application (Task B) with its own stack of activities
 - after interacting with that application, the user returns Home again and selects the application that originally started Task A
 - now, Task A comes to the foreground—all three activities in its stack are intact and the activity at the top of the stack resumes
 - at this point, the user can also switch back to Task B by going Home and selecting the application icon that started that task (or by selecting the app's task from the overview screen)

Activities and Tasks - summary

- To summarize the default behavior for activities and tasks:
 - when Activity A starts Activity B, Activity A is stopped, **but the system retains its state (such as scroll position and text entered into forms)**
 - if the user presses the *Back* button while in Activity B, Activity A resumes with its state restored
 - when the user leaves a task by pressing the *Home* button, the current activity is stopped and its task goes into the background
 - **the system retains the state of every activity in the task**
 - if the user later resumes the task by selecting the launcher icon that began the task, the task comes to the foreground and resumes the activity at the top of the stack
 - if the user presses the *Back* button, the current activity is popped from the stack and destroyed
 - the previous activity in the stack is resumed
 - when an activity is destroyed, the system *does not* retain the activity's state
 - Activities can be instantiated multiple times, even from other tasks.



CREATING ACTIVITIES

Creating and Activity and its Interface (1/2)

- To create an activity:
 - create a subclass of `Activity` (or an existing subclass of it)
 - in the subclass, implement callback methods that the system calls when the activity transitions between various states of its lifecycle (e.g. when the activity is being created, stopped, resumed, or destroyed)
- The user interface for an activity is provided by a hierarchy of views:
 - these are objects derived from the `View` class
 - each view controls a particular rectangular space within the activity's window (it responds to the user)
 - e.g. a view might be a button that initiates an action when the user touches it
- Provides a number of ready-made views that can be used to design and organize the layout:
 - "Layouts" are views derived from `ViewGroup`
 - they provide a unique layout model for its child views (e.g. linear layout, grid layout)
 - you can also subclass the `View` and `ViewGroup` classes (or existing subclasses) to create your own widgets and layouts and apply them to your activity layout

Creating and Activity and its Interface (2/2)

- The most common way to define a layout using views is with an XML layout file saved in your application resource:
- this way, you can maintain the design of your user interface separately from the source code that defines the activity's behavior
- you can set the layout as the UI for your activity with `setContentView()`, passing the resource ID for the layout
- you can also create new Views in your activity code and build a view hierarchy by inserting new Views into a ViewGroup, then use that layout by passing the root ViewGroup to `setContentView()`
- "Widgets" are views that provide a visual (and interactive) elements for the screen, such as a button, text field, checkbox, or just an image

Declaring an Activity

- Declare your activity in the manifest file for it to be accessible to the system:
 - open your manifest file and add an <activity> element as a child of the <application> element
 - e.g. :

```
<manifest ... >
    <application ... >
        <activity android:name=".ExampleActivity" />
        ...
    </application ... >
    ...
</manifest>
```

- There are several other attributes that you can include in this element to define properties:
 - e.g. the label for the activity, an icon for the activity, or a theme to style the activity's UI
- The android:name attribute is the only required attribute:
 - it specifies the class name of the activity
- Once you publish your application, you should not change this name:
 - if you do, you might break some functionality, such as application shortcuts

Intent Filters (1/2)

- An `<activity>` element can also specify various intent filters in order to declare how other application components may activate it:
 - simply use the `<intent-filter>` element
- When you create a new application (using the Android SDK tools):
 - the stub activity that's created automatically includes an intent filter
 - this filter declares that the activity responds to the "main" action and should be placed in the "launcher" category

```
<activity android:name=".ExampleActivity"
    android:icon="@drawable/app_icon">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category
            android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

- The `<action>` element specifies that this is the "main" entry point to the application
- The `<category>` element specifies that this activity should be listed in the system's application launcher (to allow users to launch this activity)

Intent Filters (2/2)

- If you intend for your application to be self-contained and not allow other applications to activate its activities:
 - you don't need any other intent filters
 - only one activity should have the "main" action and "launcher" category, as in the previous example
 - activities that you don't want to make available to other applications should have no intent filters and you can start them yourself using explicit intents
- If you want your activity to respond to implicit intents that are delivered from other applications (and your own):
 - you must define additional intent filters for your activity
 - for each type of intent to which you want to respond, you must include an
 - <intent-filter> that includes an
 - <action> element and, optionally, a
 - <category> element and/or a
 - <data> element
 - these elements specify the type of intent to which your activity can respond

Starting an Activity (1/2)

- You can start an activity by:
 - calling `startActivity()`, and
 - passing it an Intent that describes the activity you want to start
 - the intent specifies:
 - either the exact activity you want to start, or
 - describes the type of action you want to perform (and the system selects the appropriate activity for you, which can even be from a different application)
 - an intent can also carry small amounts of data to be used by the activity that is started
- When working within your own application, you'll often need to simply launch a known activity:
 - you can do so by creating an intent that explicitly defines the activity you want to start, using the class name.
 - e.g. one activity starts another activity named `SignInActivity`

```
Intent intent = new Intent(this,  
    SignInActivity.class);  
startActivity(intent);
```

Starting an Activity (2/2)

- If your application wants to perform some action using data from your activity, and it does not have its own activities to perform such actions:
 - e.g. send an email, text message, or status update
 - you can instead leverage the activities provided by other applications on the device, which can perform the actions for you
 - you can create an intent that describes an action you want to perform and the system launches the appropriate activity from another application
 - e.g., if you want to allow the user to send an email message, you can create the following intent:

```
Intent intent = new Intent(Intent.ACTION_SEND);  
intent.putExtra(Intent.EXTRA_EMAIL,  
recipientArray);  
startActivity(intent);
```

- The EXTRA_EMAIL added to the intent:
 - string array of email addresses to which the email should be sent
 - when an email application responds to this intent, it reads the string array provided in the extra and places them in the "to" field of the email composition form
 - in this situation, the email application's activity starts and when is done, your activity resumes

Starting an Activity for a Result (1/3)

- You can also start another activity and receive a result back
- To receive a result:
 - call `startActivityForResult()` (instead of `startActivity()`)
 - e.g. your app can start a camera app and receive the captured photo as a result
- The activity that responds must be designed to return a result:
 - when it does, it sends the result as another Intent object
 - your activity receives it in the `onActivityResult()` callback

Starting an Activity for a Result (2/3)

- There's nothing special about the Intent object you use when starting an activity for a result:
 - you do need to pass an additional integer argument to the `startActivityForResult()` method
- The integer argument is a "request code":
 - it identifies your request
 - when you receive the result Intent, the callback provides the same request code so that your app can properly identify the result and determine how to handle it
- E.g., start an activity that allows the user to pick a contact:

```
static final int PICK_CONTACT_REQUEST = 1; // The request code
...
private void pickContact() {
    Intent pickContactIntent = new Intent(Intent.ACTION_PICK,
Uri.parse("content://contacts"));
    // Show only contacts w/ phone numbers
    pickContactIntent.setType(Phone.CONTENT_TYPE);
    startActivityForResult(pickContactIntent,
PICK_CONTACT_REQUEST);
}
```

Starting an Activity for a Result (3/3)

- When the user is done with the subsequent activity and returns:

- the system calls your activity's `onActivityResult()` method
- this method includes three arguments:
 - the request code you passed to `startActivityForResult()`
 - a result code specified by the second activity:
 - this is either `RESULT_OK` if the operation was successful, or
 - `RESULT_CANCELED` if the user backed out or the operation failed for some reason.
 - an Intent that carries the result data

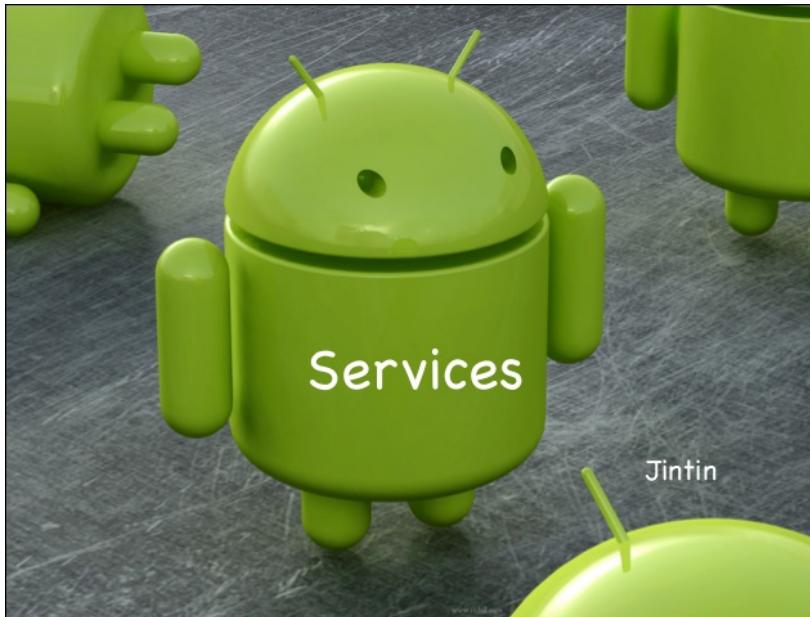
- E.g., handle the result for the "pick a contact" intent:

- the result Intent returned by Android's Contacts or People app provides a content Uri that identifies the contact the user selected

```
@Override  
protected void onActivityResult(int requestCode, int resultCode, Intent data) {  
    // Check which request we're responding to  
    if (requestCode == PICK_CONTACT_REQUEST) {  
        // Make sure the request was successful  
        if (resultCode == RESULT_OK) {  
            // The user picked a contact.  
            // The Intent's data Uri identifies which contact was selected.  
            // Do something with the contact here  
        }  
    }  
}
```

Shutting Down an Activity

- You can shut down an activity by calling its `finish()` method.
- You can also shut down a separate activity that you previously started by calling `finishActivity()`.
- The Android system manages the life of an activity for you, so you do not need to finish your own activities.
- Calling these methods could adversely affect the expected user experience:
 - thus, it should only be used when you absolutely do not want the user to return to this instance of the activity



SERVICES

Services

- A service is a component that:

- runs in the background
- to perform long-running operations
- without needing to interact with the user

- A service can essentially take two states:

Started A service is **started** when an application component, such as an activity, starts it by calling `startService()`.

Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.

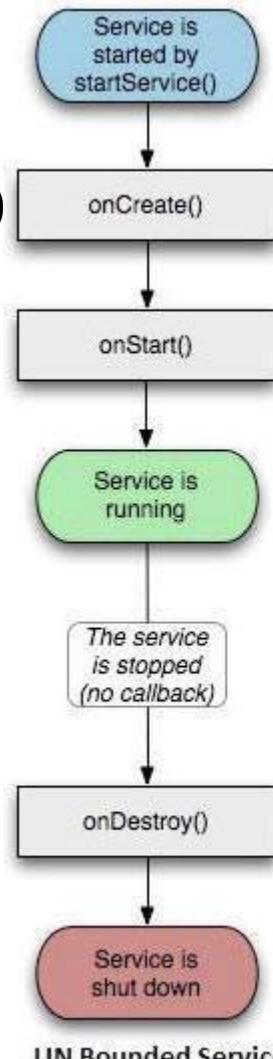
Bound A service is **bound** when an application component binds to it by calling `bindService()`.

A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with interprocess communication (IPC).

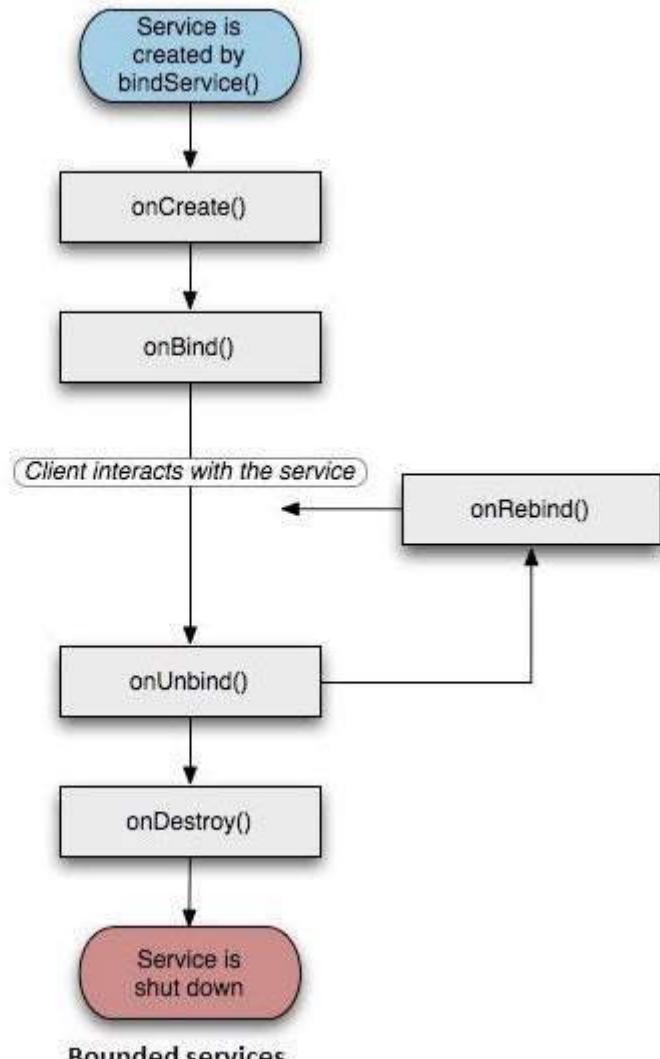
Services Lifecycle

- A service has life cycle callback methods that you can implement to:
 - monitor changes in the service's state, and

- Left: life cycle when the service is created with `startService()`



- Right: life cycle when the service is created with `bindService()`



UN Bounded Service

Bounded services

Creating a Service

- To create a service:

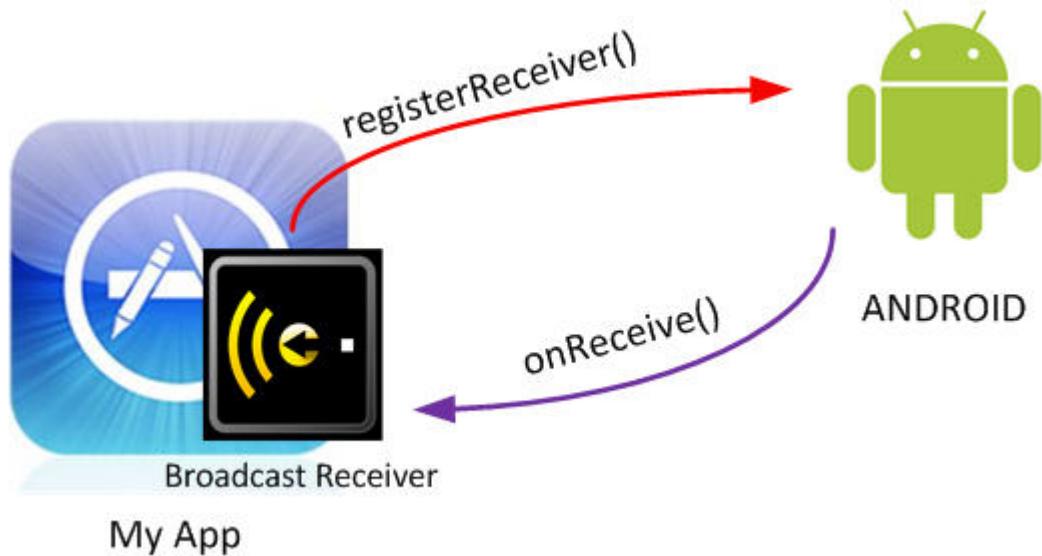
- you create a Java class that extends the Service base class or one of its existing subclasses

- The Service base class defines various callback methods

- You don't need to implement all the callbacks methods:

- it's important that you understand each one and implement those that ensure your app behaves the way users expect

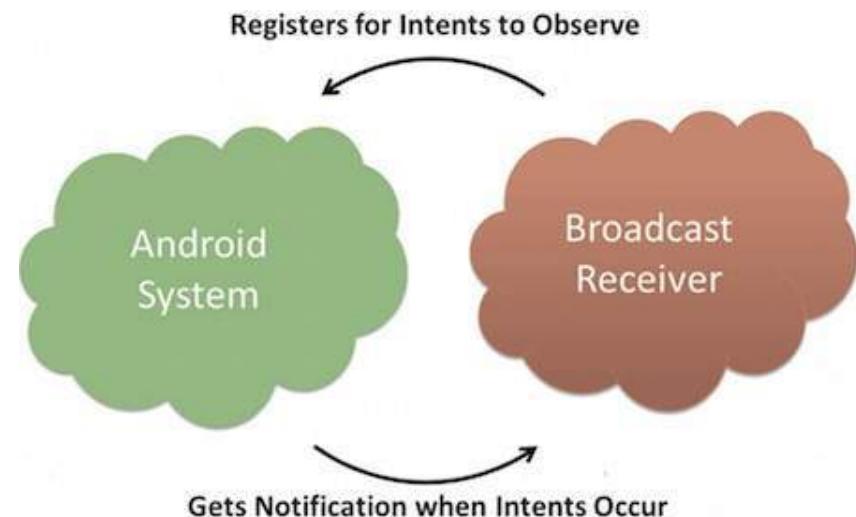
Callback	Description
onStartCommand()	The system calls this method when another component, such as an activity, requests that the service be started, by calling <i>startService()</i> . If you implement this method, it is your responsibility to stop the service when its work is done, by calling <i>stopSelf()</i> or <i>stopService()</i> methods.
onBind()	The system calls this method when another component wants to bind with the service by calling <i>bindService()</i> . If you implement this method, you must provide an interface that clients use to communicate with the service, by returning an <i>IBinder</i> object. You must always implement this method, but if you don't want to allow binding, then you should return <i>null</i> .
onUnbind()	The system calls this method when all clients have disconnected from a particular interface published by the service.
onRebind()	The system calls this method when new clients have connected to the service, after it had previously been notified that all had disconnected in its <i>onUnbind(Intent)</i> .
onCreate()	The system calls this method when the service is first created using <i>onStartCommand()</i> or <i>onBind()</i> . This call is required to perform one-time set-up.
onDestroy()	The system calls this method when the service is no longer used and is being destroyed. Your service should implement this to clean up any resources such as threads, registered listeners, receivers, etc.



BROADCAST RECEIVERS

Broadcast Receivers

- Broadcast Receivers simply respond to broadcast messages from other applications or from the system itself.
- For example:
 - Applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device exists, and is available for them to use,
 - so that the broadcast receiver intercepts this communication and initiates an appropriate action
- There are two important steps to make BroadcastReceiver works for the system broadcast intents:
 - Creating the Broadcast Receiver
 - Registering Broadcast Receiver



Creating and Registering a Broadcast Receiver

- A broadcast receiver is implemented:
 - subclass of BroadcastReceiver class, and
 - overriding the onReceive() method where each message is received as an Intent object parameter

```
public class MyReceiver extends BroadcastReceiver {  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        Toast.makeText(context, "Intent Detected.", Toast.LENGTH_LONG).show();  
    }  
}
```

A toast is a view containing a quick little message for the user.

- An application listens for specific broadcast intents by:

- registering a broadcast receiver in AndroidManifest.xml file

- E.g. registering MyReceiver for system generated event

ACTION_BOOT_COMPLETED:

- fired by the system once the Android system has booted

```
<application  
    android:icon="@drawable/ic_launcher"  
    android:label="@string/app_name"  
    android:theme="@style/AppTheme" >  
    <receiver android:name="MyReceiver">  
        <intent-filter>  
            <action android:name="android.intent.action.BOOT_COMPLETED">  
            </action>  
        </intent-filter>  
    </receiver>  
</application>
```

System Events

Event Constant	Description
android.intent.action.BATTERY_CHANGED	Sticky broadcast containing the charging state, level, and other information about the battery.
android.intent.action.BATTERY_LOW	Indicates low battery condition on the device.
android.intent.action.BATTERY_OKAY	Indicates the battery is now okay after being low.
android.intent.action.BOOT_COMPLETED	This is broadcast once, after the system has finished booting.
android.intent.action.BUG_REPORT	Show activity for reporting a bug.
android.intent.action.CALL	Perform a call to someone specified by the data.
android.intent.action.CALL_BUTTON	The user pressed the "call" button to go to the dialer or other appropriate UI for placing a call.
android.intent.action.DATE_CHANGED	The date has changed.
android.intent.action.REBOOT	Have the device reboot.

Broadcasting Custom Intents

- An application can generate and send custom intents:
- Create and send those intents by using the sendBroadcast() method.

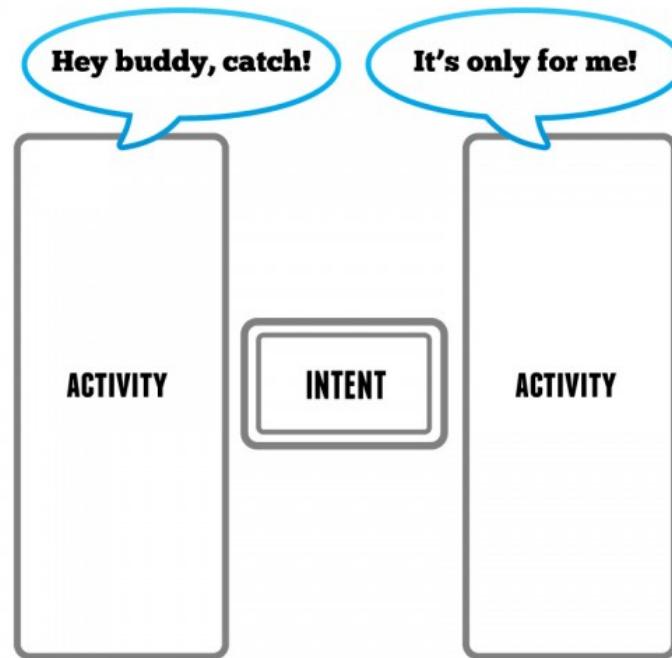
```
public void broadcastIntent(View view)
{
    Intent intent = new Intent();
    intent.setAction("com.tutorialspoint.CUSTOM_INTENT");
    sendBroadcast(intent);
}
```

- The intent com.tutorialspoint.CUSTOM_INTENT can also be registered as a broadcast message (like system-wide intents).

```
<application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <receiver android:name="MyReceiver">

        <intent-filter>
            <action android:name="com.tutorialspoint.CUSTOM_INTENT">
            </action>
        </intent-filter>

    </receiver>
</application>
```



INTENTS

Intents and Intent Filters (1/2)

- An Intent is a messaging object you can use to request an action from another app component
- An intent can be seen as abstract description of an operation to be performed; thus it can be used with:
 - `startActivity` to launch an Activity,
 - `startService(Intent)` or `bindService(Intent, ServiceConnection, int)` to communicate with a background Service
 - `broadcastIntent` to send it to any interested BroadcastReceiver components, and

The intent itself, an Intent object, is a passive data structure holding an abstract description of an operation to be performed

- Start an activity:
 - an Activity represents a single screen in an app
 - start a new instance of an Activity by passing an Intent to `startActivity()`
 - the Intent describes the activity to start and carries any necessary data
 - if you want to receive a result from the activity when it finishes, call `startActivityForResult()`
 - your activity receives the result as a separate Intent object in your activity's `onActivityResult()` callback

Intents and Intent Filters (2/2)

- Start a service:

- a Service is a component that performs operations in the background without a user interface
- you can start a service to perform a one-time operation (such as download a file) by passing an Intent to startService()
- the Intent describes the service to start and carries any necessary data
- if the service is designed with a client-server interface, you can bind to the service from another component by passing an Intent to bindService()

- A broadcast is a message that any app can receive:

- the system delivers various broadcasts for system events (e.g. when the system boots up or the device starts charging)
- e.g. you can deliver a broadcast to other apps by passing an Intent to sendBroadcast()

Intent Types (1/2)

- There are two types of intents:

- explicit and implicit

- Explicit intents:

- Specify the component to start by name (the fully-qualified class name)
- You'll typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start
- E.g., start a new activity in response to a user action or start a service to download a file in the background.

- Implicit intents:

- do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it
- e.g., if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map

Intent Types (2/2)

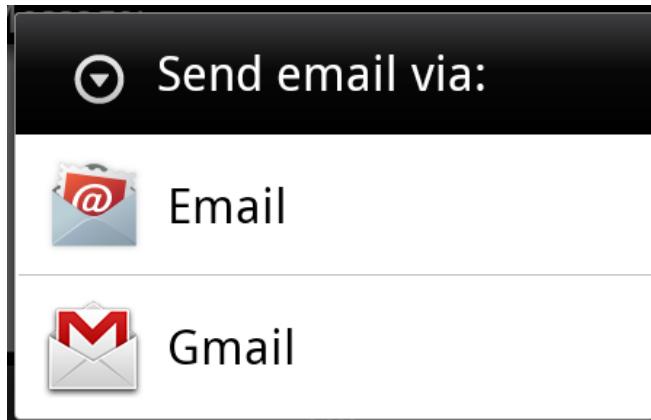
- When you create an explicit intent to start an activity or service:
 - the system immediately starts the app component specified in the Intent object
- When you create an implicit intent:
 - the system finds the appropriate component to start by comparing the contents of the intent to the intent filters declared in the manifest file of other apps on the device
 - if the intent matches an intent filter, the system starts that component and delivers it the Intent object
 - if multiple intent filters are compatible, the system displays a dialog so the user can pick which app to use

Example: email Intent

- You have an Activity that needs to launch an email client and sends an email using your Android device
- For this purpose, your Activity would send an ACTION_SEND along with appropriate chooser, to the Android Intent Resolver
- The specified chooser gives the proper interface for the user to pick how to send your email data

```
Intent email = new Intent(Intent.ACTION_SEND, Uri.parse("mailto:"));
email.putExtra(Intent.EXTRA_EMAIL, recipients);
email.putExtra(Intent.EXTRA_SUBJECT, subject.getText().toString());
email.putExtra(Intent.EXTRA_TEXT, body.getText().toString());
startActivity(Intent.createChooser(email, "Choose an email client from..."));
```

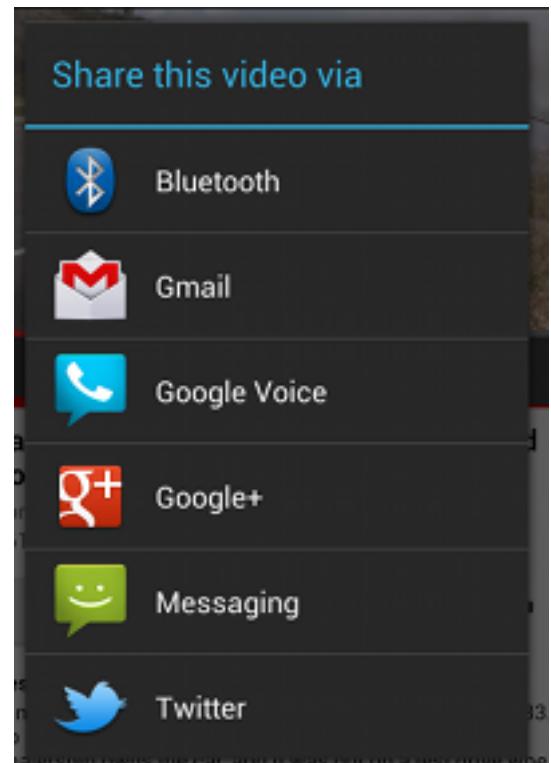
- It calls the startActivityForResult method to start an email activity and result should be as shown:



Forcing an App Chooser

- When there is more than one app that responds to your implicit intent:
 - the user can select which app to use, and
 - make that app the default choice for the action

- If multiple apps can respond to the intent,
the user might want to use a different
app each time:
 - you should explicitly show a chooser dialog
 - the chooser dialog asks the user to select
which app to use for the action every time



Explicit Intent

- An explicit intent is one that:

- you use to launch a specific app component, such as a particular activity or service in your app
- to create an explicit intent, define the component name for the Intent object—all other intent properties are optional

- Example of a service (in your app) named DownloadService:

- designed to download a file from the web,
- you can start it with the following code:

```
// Executed in an Activity, so 'this' is the Context
// The fileUrl is a string URL, such as "http://www.example.com/image.png"
Intent downloadIntent = new Intent(this, DownloadService.class);
downloadIntent.setData(Uri.parse(fileUrl));
startService(downloadIntent);
```

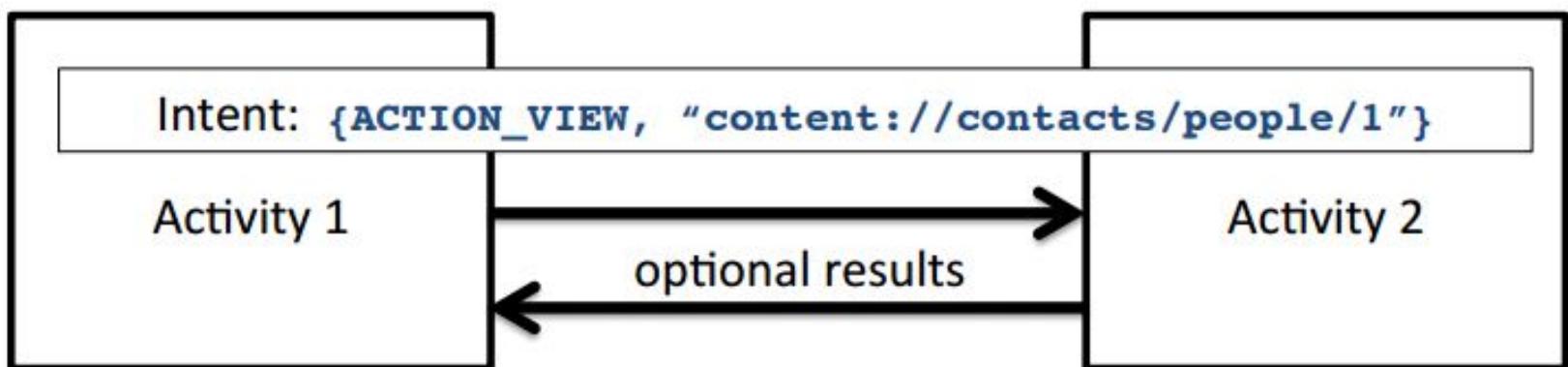
- The Intent(Context, Class) constructor supplies the app Context and the component a Class object:
- thus, this intent explicitly starts the DownloadService class in the app

Implicit Intent

- An implicit intent specifies an action that can invoke any app on the device able to perform the action
- Using an implicit intent is useful when:
 - your app cannot perform the action, but
 - other apps probably can and you'd like the user to pick which app to use
- E.g., if you have content you want the user to share with other people:
 - create an intent with the ACTION_SEND action, and
 - add extras that specify the content to share
 - when you call startActivity() with that intent, the user can pick an app through which to share the content

Main Arguments of Intents

- Action: the built-in action to be performed, such as ACTION_VIEW, or user-created-action
- Data: the primary data to operate on, such as a phone number to be called (expressed as a URI)

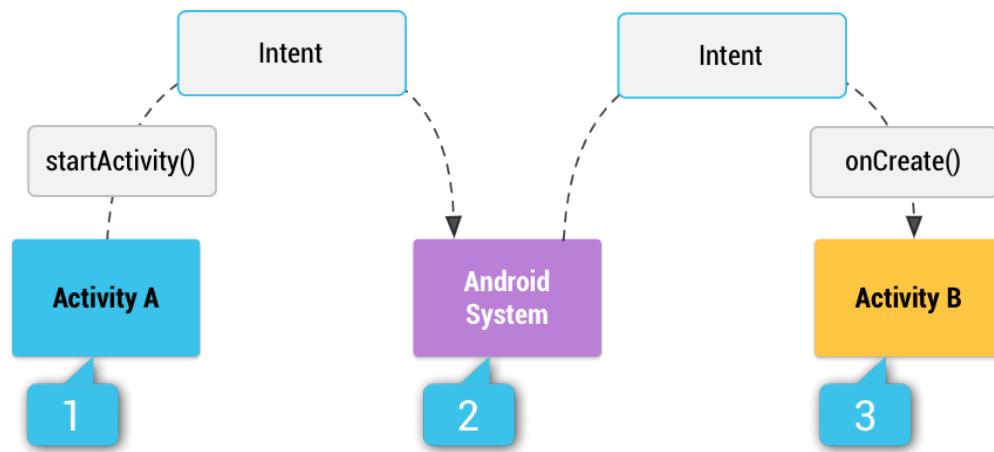


Intent Filter

- An intent filter is an expression in an app's manifest file that specifies the type of intents that the component would like to receive:
 - e.g., by declaring an intent filter for an activity, you make it possible for other apps to directly start your activity with a certain kind of intent
 - likewise, if you do *not* declare any intent filters for an activity, then it can be started only with an explicit intent

- How an implicit intent is delivered through the system to start another activity:

- [1] Activity A creates an Intent with an action description and passes it to `startActivity()`
- [2] The Android System searches all apps for an intent filter that matches the intent
- When a match is found, [3] the system starts the matching activity (Activity B) by invoking its `onCreate()` method and passing it the Intent.



Building an Intent

- An Intent object carries information that the Android system uses to determine which component to start:
 - the exact component name or component category that should receive the intent, plus
 - information that the recipient component uses in order to properly perform the action (such as the action to take and the data to act upon)
- The primary information contained in an Intent is the following:
 - Component name
 - Action
 - Data
 - Category
 - Extras
 - Flags

Intent – component

- Component name:

- the name of the component to start
- this is optional, but it's the critical piece of information that makes an intent **explicit**,
- it means that the intent should be delivered only to the app component defined by the component name
- without a component name, the intent is **implicit** (the system decides which component should receive the intent based on the other intent information, e.g. action, data, and category)

- Action:

- a string that specifies the generic action to perform
- in the case of a broadcast intent, this is the action that took place and is being reported
- the action largely determines how the rest of the intent is structured (particularly what is contained in the data and extras)
- you can specify your own actions for use by intents within your app (or for use by other apps to invoke components in your app), but
- you should usually use action constants defined by the Intent class or other framework classes
- Example:
 - ACTION_VIEW - use this action in an intent with `startActivity()` when you have some information that an activity can show to the user (e.g. a photo to view in a gallery app, or an address to view in a map app)
 - ACTION_SEND - also known as the "share" intent, you should use this in an intent with `startActivity()` when you have some data that the user can share through another app, such as an email app or social sharing app

Intent – data and category

○ Data:

- the URI (a Uri object) that references the data to be acted on and/or the MIME type of that data
- the type of data supplied is generally dictated by the intent's action
- e.g., if the action is ACTION_EDIT, the data should contain the URI of the document to edit
- when creating an intent, it's often important to specify the type of data (its MIME type) in addition to its URI
- e.g., an activity that's able to display images probably won't be able to play an audio file, even though the URI formats could be similar
- so specifying the MIME type of your data helps the Android system find the best component to receive your intent
- however, the MIME type can sometimes be inferred from the URI—particularly when the data is a content: URI, which indicates the data is located on the device and controlled by a ContentProvider, which makes the data MIME type visible to the system

○ Category:

- a string containing additional information about the kind of component that should handle the intent
- any number of category descriptions can be placed in an intent, but most intents do not require a category
- some common categories:
 - CATEGORY_BROWSABLE - the target activity allows itself to be started by a web browser to display data referenced by a link—such as an image or an e-mail message
 - CATEGORY_LAUNCHER - the activity is the initial activity of a task and is listed in the system's application launcher



Mobile and Ubiquitous Computing

2022-2023

MEIC/METI - Alameda & Taguspark

Location I & II



Contents

- Basic definitions
- GPS and Differential GPS
- Infrared and ultrasonic
- Wi-Fi and Cellular
- Others
- Location-Based Applications and Services
- Challenges and opportunities

Type of location solutions:

- Triangulation, ToF, AoA, etc.
- Proximity
- Scene analysis (fingerprinting, modelling)

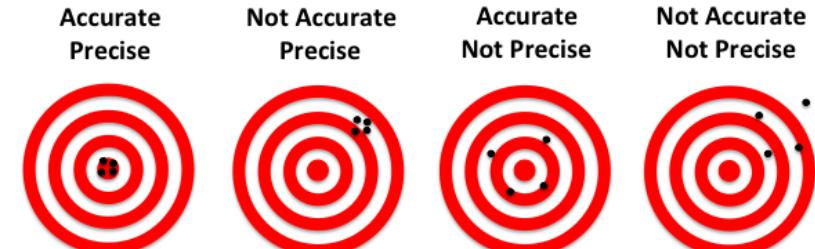


Introduction

- Location is at the core of a number of **high-value applications** including:
 - emergency response, navigation, asset tracking, ground surveying, and many others
- The **market for GPS products and services** alone was approx. **US\$200 billion in 2015**
- **Location** is also commonly used to **infer other contexts**:
 - **symbolic location** is often a good **proxy for activity** (e.g., grocery store is indicative of shopping)
 - **social roles and interactions** can be learned from **patterns of colocation**, and
 - **physical activities and modes of transportation** can often be inferred from the changes in coordinate-based location

Basic Definitions

- Location:
 - **symbolic** - symbolic location encompasses abstract ideas of where something is: kitchen, room 609, on a train arriving at Lisbon, etc.
 - **physical** - physical location provides coordinates such as latitude and longitude
- Accuracy and Precision:
 - some inexpensive GPS receivers can locate positions to within 10 meters for approximately 95 percent of measurements.
 - **correction is accuracy.**
 - **variability is precision.**
- To assess the **scale of a location system**, we consider:
 - its **coverage area per unit of infrastructure**, and
 - the **number of objects the system can locate per unit of infrastructure per time interval**
- Infrastructure **cost**
- Per-client **cost**
- Privacy:
 - location of a user is **obtained locally or on a central service.**





Outdoor Location: GPS



GPS Context (1/2)

- NAVSTAR Global Positioning System (GPS) is by far **the most widely** used outdoor location technology.
- It is estimated that in 2020, the number of **GPS chipsets** approached **three billion**
- Dates:
 - the U.S. Department of Defense **approved the project in 1973**
 - the system was declared **fully operational in 1995**
 -
- Cost:
 - **development** of GPS has been reported at **\$14 billion**
 - its **annual operation** and maintenance cost is estimated at **\$500 million**
 -
- It is a **passive one-way** ranging system where all signals are transmitted by earth-orbiting **satellites**, and **position determination happens at the receivers**
- It provides:
 - worldwide coverage
 - scales to an unlimited number of users,
 - preserves user privacy
 - supports a range of location services with accuracies that range from several meters to a few millimeters

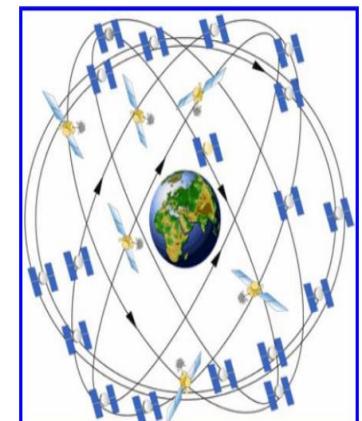


GPS Context (2/2)

- Different types of GPS receivers vary in pricing:
 - from a **hundred dollars** for an inexpensive mass market model, to
 - tens of **thousands of dollars** for the more accurate kinematic solutions used for land surveying
- GPS was designed as a dual-use technology for both **military** and **civilian** use:
 - to maintain an accuracy advantage for the military, the original GPS design included a feature known as **Selective Availability**
 - it added **intentional noise** to the signals to degrade the accuracy of civilian GPS models
- Shortage of military-grade GPS receivers during the first Persian Gulf War in **1991**:
 - the U.S. Pentagon **disables Selective Availability** for the duration of the war,
 - and used **civilian receivers** to make up the shortfall
- Selective Availability was:
 - permanently **discontinued in May of 2000**, and
 - in **September 2007**, the U.S. Government announced its decision to **eliminate the feature** from future GPS satellites
- In December of **2004**:
 - the U.S. government reiterated its commitment to provide **GPS access free of direct user fees** for civil, commercial, and scientific uses

Architecture (1/3)

- It consists of three distinct parts:
 - a constellation of Earth-orbiting **satellites** that broadcast a continuous ranging signal,
 - **ground stations** that update the satellites' coordinate projections and clocks and,
 - the **receivers** that use the GPS signals to estimate their position
- Earth-orbiting satellites:
 - **31 satellites** organized into **six non-geostationary circular orbits** 26,560 km above the Earth with a **12-h period**
 - full GPS coverage requires **24 GPS satellites**
 - the **additional satellites** operate as **active spares** to **accommodate occasional maintenance downtime** and to assure **system robustness**



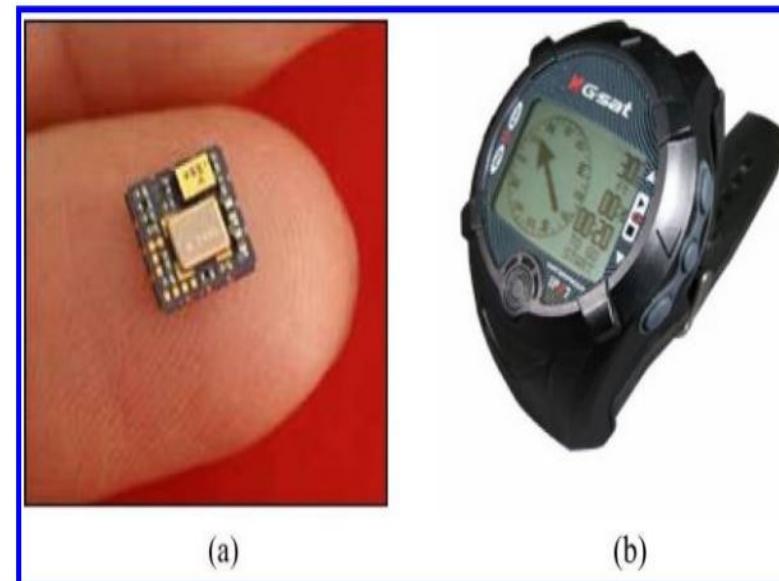
Architecture (2/3)

- Ground stations:
 - responsible for **monitoring satellite positions** and providing satellites with **clock corrections and satellite orbit updates**
 - there are enough ground monitoring stations to allow **each satellite to be simultaneously tracked by at least two monitoring stations**
 - simultaneous satellite tracking improves the precision of orbit calculations **increasing localization accuracy**



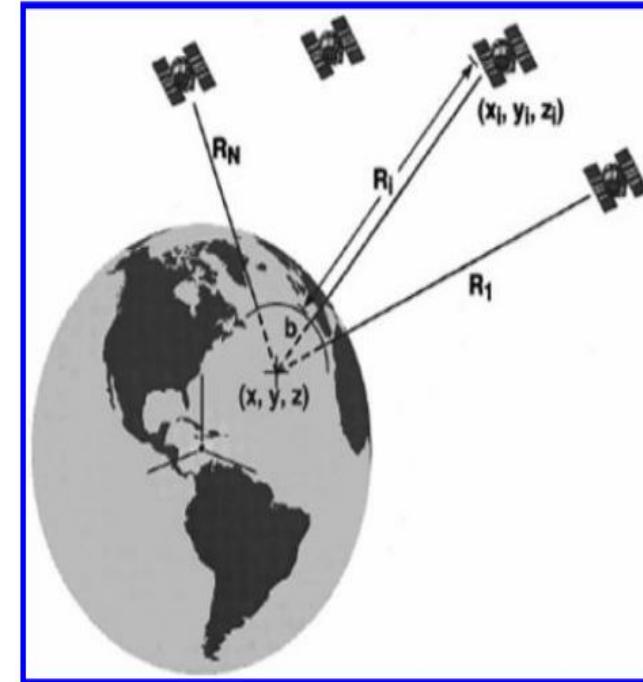
Architecture (3/3)

- GPS receivers:
 - determine their position by **simultaneously tracking at least 4**, but commonly up to 12, satellites
 - can be **augmented with other sensors** (e.g. altimeters, accelerometers, and gyroscopes) to compensate for gaps in GPS coverage
- Receivers:
 - the original mobile units tested by the **U.S. Army in the 1970s weighed approx. 12kg** and filled a backpack
 - today, typically around the **size of a cell phone**, and new single chip GPS implementations have made form factors as small as a wristwatch as possible



Algorithm (1/3)

- Supported by all GPS receivers and allows the receiver to:
 - **estimate its position in three dimensions** (latitude, longitude, altitude)
- Location estimated:
 - computed from the **estimated positions of the satellites** and the **ranges from the receiver to those satellites**
 -
- **Satellite locations are learned from the broadcasts from the satellite:**
 - **R_i** is the **distance between the receiver and satellite i,**
 - it is inferred by measuring the **transit time of the signal between the satellite and the receiver and multiplying by the speed of light**





Algorithm (2/3)

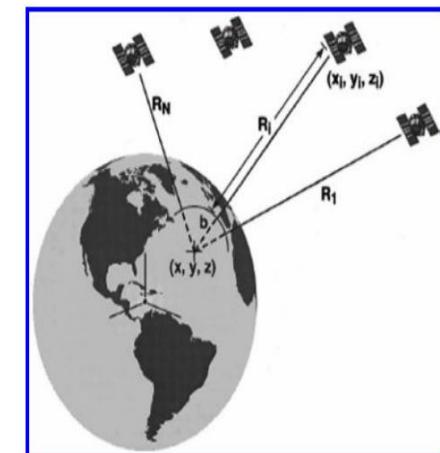
- Measuring the signal transit time accurately requires:
 - the satellite and receiver's **clocks to be tightly synchronized**
- In practice, the use of **low-cost crystal oscillators in the receiver** introduces:
 - a bias that makes the distance from the satellite appear **shorter or longer than the real value**
 - the receiver-induced bias will be the **same across all of the satellites**
 - **satellite** induced clock bias is less of an issue because of their **extremely accurate atomic clocks**
 - thus, the effect of the **receiver's clock bias can be removed** by treating it as an extra unknown in the location calculation

Algorithm (3/3)

- The **receiver's location** in three-dimensional space (x, y, z) and **receiver clock bias b** is determined by solving the following equation for at least four satellites (just three satellites would be required with perfect clocks):

$$R_i = \sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} - b$$

It requires at least **four satellites** (as opposed to the **three satellites that would be required with perfect clocks**)

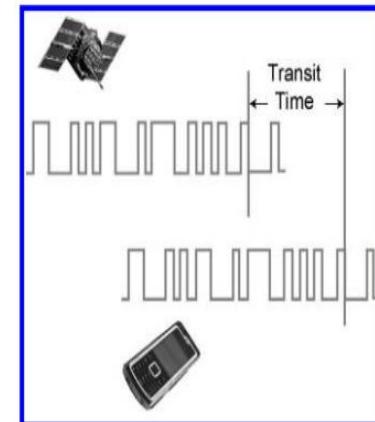


- In cases where more than four satellites signals are available to the receiver:
 - the **redundant data** is used to try to **identify and eliminate error** in the location estimate
- Commercial GPS units using this basic algorithm estimate location with **median accuracy of around 10 m**



Algorithm - Satellite Range Estimation

- To allow the **distance between the satellite and the receiver** to be estimated:
 - GPS satellites **transmit radio signals modulated** by pseudorandom noise (**PRN codes**),
 - such codes consist of **binary sequences** that appear to be randomly generated but are **specific** to each satellite and **known** by the receiver.
- GPS receivers continually **compare the signals they are receiving** with a **locally generated replica of the satellite's PRN code**:
 - the time **delta between the received signal and the local PRN code** represents the **signal's travel time**
 - range is computed by simply **multiplying the travel time by the speed of light** (299,729,458 m/s)
- The use of orthogonal PRN codes allows all GPS satellites to use the same frequencies:
 - while still allowing receivers to differentiate their transmissions
 - and tracking them in parallel
- The **theoretical accuracy of GPS** ranging is:
 - 3 m for the civilian
 - 0.3 m for the military



Algorithm - Satellite Coordinate Estimation

- The GPS receiver obtains the satellite's coordinates from a **navigation message** which is also modulated on 1,575 and 1,227 MHz signals
- The **message is encoded within the PRN code** at the low bit rate of 50 bits per second
- The navigation **message is 37,500 bits long** and **takes 12.5 min to transmit**
- The **message contains all of the relevant information** about the GPS constellation:
 - the coordinates of the satellites as a function of time,
 - satellite clock correction parameters,
 - a satellite directory,
 - constellation health status, and
 - parameters for ionospheric error correction
- To speed location calculation at the GPS receiver:
 - the **satellite coordinates and clock offset are repeated** in the navigation message **every 30s**

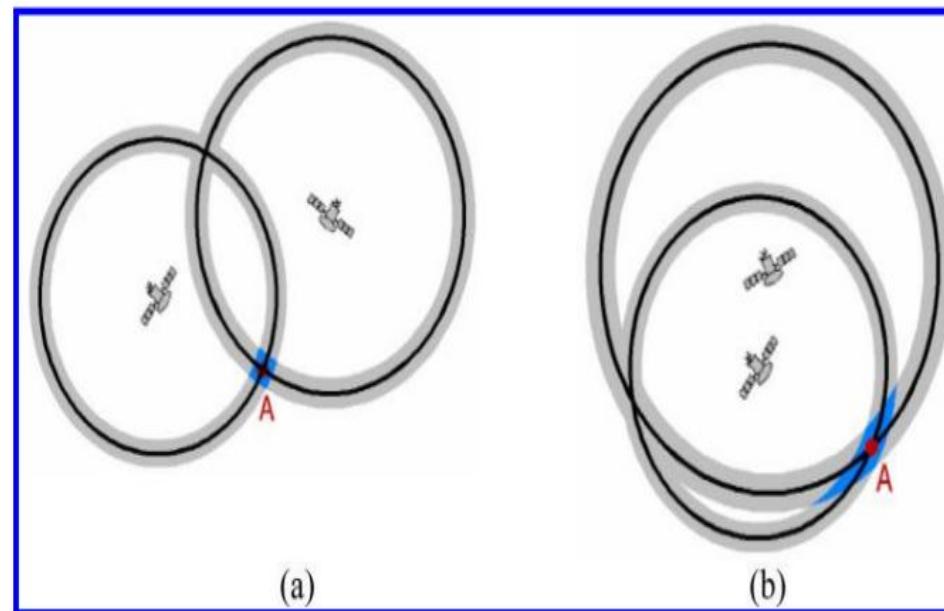


Errors and Biases – Ranging Errors

- Satellite's **atomic clocks are very stable**:
 - Still can **accumulate up to 17 ns of error per day**, which translates to a **range error of 5 m**
 - To correct for this, the **satellite clock is continually monitored by the ground monitoring stations**, and **clock corrections are periodically transmitted**
- **Satellite coordinates errors**:
 - On the order of **2 m**
 - Result of the **failure of the satellite position models** to account for all forces acting on the satellite
 - This can be **eliminated almost completely** by using precise orbit data (accurate within a few centimeters) that is made available over the Internet
- Ionospheric delay is the dominant source of GPS ranging error:
 - Interaction of the GPS signal with ionized gases in the upper atmosphere,
 - **Varies with time of day, time of year, solar flare activity**, and the **angle of entrance of signal** that affects the length of the path through the ionosphere
 - **Models** of ionospheric delay have been developed, leaving a **residual delay with a 4-m mean**

Errors and Biases – Satellite Geometry Errors

- The quality of the GPS location estimation depends on:
 - how well the tracked **satellites are spread across the sky**, and
 - in general, **satellite geometry improves as the distance between satellites increases**
- The **size of the uncertainty area** where the receiver could be located **decreases**, as the **distance between the satellites increases** (see blue bands in the figure)



GPS Drawbacks

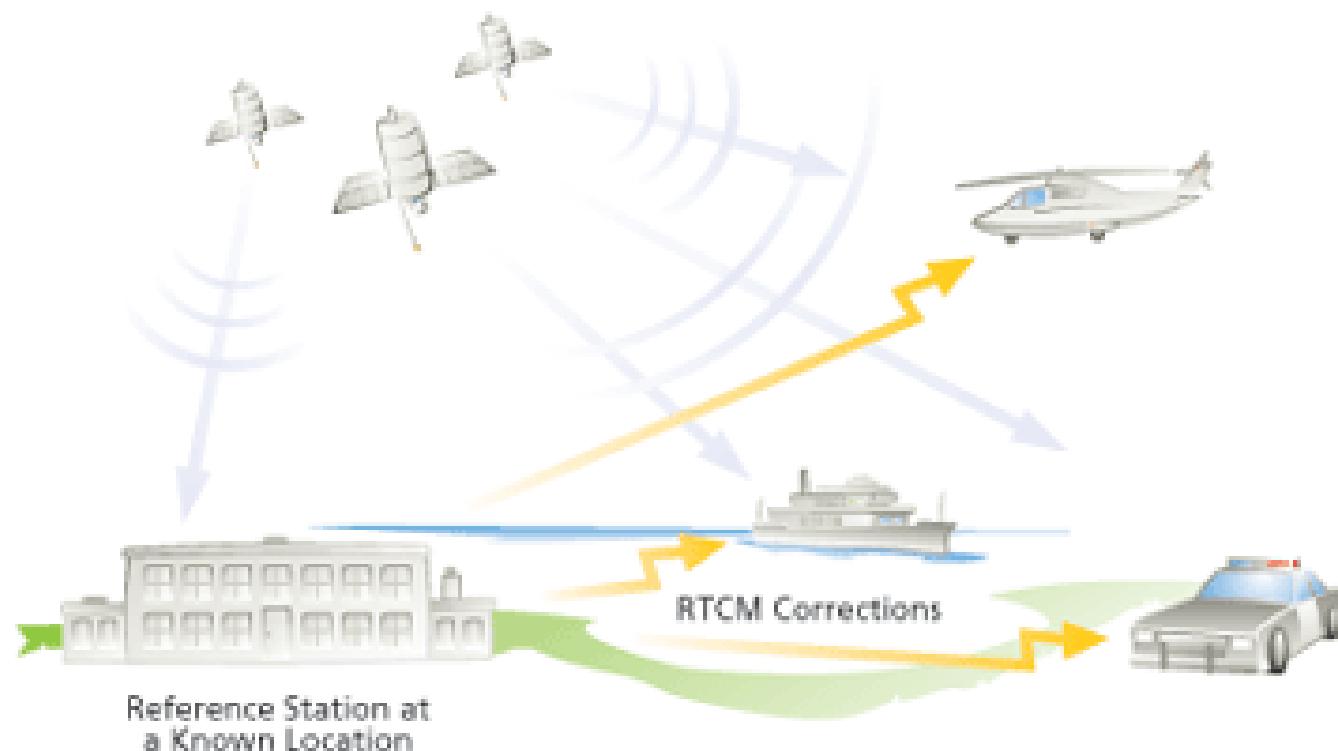
- Accurate GPS localization **requires an unobstructed view of at least four satellites**
- GPS signals do not penetrate well through walls, soil, and water:
 - thus, the system **cannot be used inside buildings, underground** (e.g., inside a mine or tunnel), or for **subsurface marine navigation**
- Signal can also be obstructed by large buildings in the so-called **urban canyons**





Real-Time Differential GPS

Real-Time Differential GPS





Main Idea of Differential GPS

- Differential GPS takes advantage of the fact that
 - satellite clock and coordinate errors, as well as ionospheric and tropospheric delays exhibit high temporal and spatial correlation and are very similar even hundreds of kilometers apart
- Thus, differential techniques take advantage of this by:
 - coordinating multiple GPS receivers that simultaneously track the same satellites
 - by having one or more GPS receivers fixed at known positions, the observed errors from those receivers can be transmitted to nearby roving GPS receivers
 - these roving units are then able to reduce their error in proportion to their proximity to the site at which the correction was measured



Real-Time Differential GPS

- Real-time differential GPS (**DGPS**) is a:
relative positioning technique that provides **sub-meter accuracy**

A fixed receiver determines the DGPS corrections by comparing its measured **satellite ranges** with ranges computed using its **known coordinates** and **the satellite coordinates** obtained from the **navigation message**

The **DGPS corrections are then transmitted** over a ground- or satellite-based wireless link to the rover

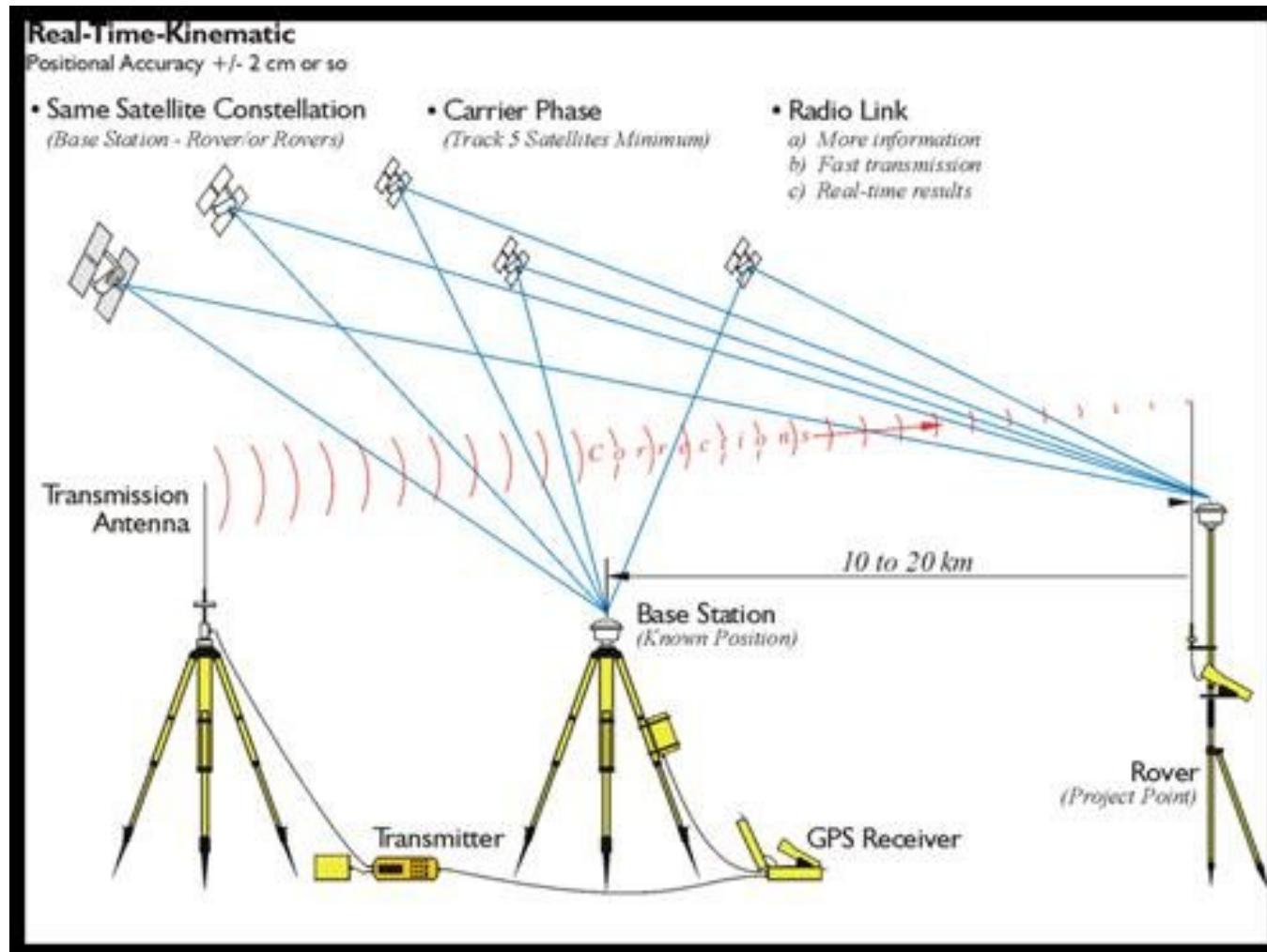
The **rover uses them to adjust** its ranging measurements

DGPS is offered both as a local and a wide-area service

Real-Time Differential GPS – examples

- Maritime DGPS:
 - it consists of a **network of stations installed at lighthouses along coastal areas** in several countries around the world
 - each station operates by independently **broadcasting real-time DGPS corrections** (285- to 325-kHz frequency band)
 - these corrections are **available at no cost**, but **require a GPS receiver augmented** to receive the corrections
 -
- Wide Area Augmentation System (WAAS):
 - it is a wide-area DGPS implementation that determines GPS range errors at **25 ground base stations**
 - these corrections are **broadcast from four geostationary satellites**
 - measurements from 27 U.S. airports show that WAAS provides **accuracy of 1.8 m at least 95%** of the time
 -
- Others:
 - European Geostationary Navigation Overlay System (EGNOS)
 - Japan's Multifunctional Transport Satellite Augmentation System (MSAS)
 - India's GPS and GEO Augmented Navigation (GAGAN)

Real-Time Kinematic GPS





Real-Time Kinematic GPS

- GPS variant that achieves **centimeter accuracy**
- It uses an **alternative technique for satellite ranging**:
 - it is based on **measuring the number of fractional and full signal cycles** that happen between the satellite and the receiver
 - the range is then **computed by multiplying this value by the carrier's wavelength**
 - while the fraction of the last cycle is easy to determine, the **number of full cycles is ambiguous**
 - this ambiguity can be resolved by taking **simultaneous measurements at two receivers** (there are several techniques)
- After receiving the **base's coordinates and measurements** and **resolving the ambiguity**:
 - roving RTK receivers obtain **centimeter-level positioning accuracy**
- RTK is used in applications that require **extreme accuracy**:
 - surveying, aircraft landing, and maritime construction
- High-end RTK systems cost **tens of thousands of euros** and require **line of sight between the coordinating GPS receivers**



Comparison

The vertical scale on the left indicates performance from worst (top) to best (bottom), with 10 stars in total.

Technology	Basic GPS	GPS + WAAS	Real-time Kinematic GPS
Accuracy	★★★☆☆ 3D coordinates with 10 m median accuracy	★★★★☆ 3D coordinates with 2 m median accuracy	★★★★★ 3D coordinates with 10 cm accuracy
Coverage	★★★★☆ Outdoors with clear view of 4+ GPS satellites	★★★★☆ Outdoors in USA — Requires clear view of 4+ GPS satellites and a WAAS satellite	★★★★☆ Outdoors with 4+ GPS satellites and requires line-sight between mobile unit and surveyed unit
Infrastructure cost	★★★★★ \$14 B US initial cost + \$500 M US yearly for global coverage	★★★★★ WAAS satellites needed in addition to GPS constellation	★★☆☆☆ Beyond GPS constellation, requires calibrated, surveyed ground unit
Per-client cost	★★★★☆ GPS antenna and chipset required	★★★★☆ GPS antenna and WAAS-capable chipset required	★★☆☆☆ Special RTK unit required
Privacy	★★★★★ Location is estimated passively on the GPS unit	★★★★★ Location is estimated passively on the GPS unit	★★★★★ Location is estimated passively on the GPS unit
Well-matched use cases	Outdoor navigation for land, sea and air, emergency response, turn-by-turn driving directions, outdoor mapping/information/tour guide services, personnel/pet tracking, fitness/activity tracking, gaming	Outdoor navigation for land, sea and air, emergency response, turn-by-turn driving directions, outdoor mapping/information/tour guide services, personnel/pet tracking, fitness/activity tracking, gaming	Outdoor navigation requiring extreme accuracy, surveying, aircraft landing, maritime construction



Indoor Location: Infrared and Ultrasonic Location Systems



Infrared and Ultrasonic Location Systems

- 1) Active Badge
- 2) Walrus
- 3) Cricket
- 4) Active Bat

Introduction (1/2)

- **Light and sound:**
 - move freely in open space.
 - are both largely **blocked by the materials** such as walls, curtains, and partitions that humans use to define the borders of their living spaces.
- These two types of signals are **well suited for indoor location systems:**
 - they can determine **which room a user is** in with high accuracy.
- High relevance of room-level location information for many **context-aware applications and services:**
 - While applications like navigation require absolute locations,
 - many others rely instead on the semantics associated with a room name or room type.
 - A location system with a median accuracy of 0.5 m may seem more desirable than a system which is only able to determine which room a user is in.
 - However, you may rethink this assessment once you consider that the former system may incorrectly infer that the user is in the adjoining kitchen 25% of the time, when in fact they are in the hallway.

Introduction (2/2)

- Possible indoor location approaches:
 - use **infrared or ultrasound beacons and listeners** to determine a location
 - the **beacons may be embedded in the environment**, and the **listeners remain mobile or vice-versa**
 - use **radio signals** to coordinate the location estimation
 - rely on the **ultrasound or infrared signals**
 - Provide symbolic room-level accuracy or
 - use **time-of-arrival** or **angle-of-arrival** to determine the user's location within the room with **5–10 cm accuracy**.



Room-Level Localization via Proximity - ActiveBadge

- The **first indoor location system** was the **Active Badge**:
 - developed in **1990** at the Olivetti Research Laboratory
 - **The badge** is the beacon; the **receiver** is in the room.
 - Determines the room in a building where people are located by having **each person wear a small (40-g) badge-like device that sends a unique pulse-width modulated infrared code every 15 s**
 - **these codes are received by base stations** mounted on the ceilings of offices, hallways, and meeting rooms
 - **base stations decode the received infrared signals** and forward the decoded **badge-IDs** to a **centralized location server** using the building's wired networking infrastructure
 - because **base stations are mounted in known locations** and because the **signals rarely propagate beyond a single room**, the **server can infer** with high probability in which room the badge is beaconsing
 - because the infrared **badges take only 0.1s to transmit their unique code**, collisions between simultaneously beaconsing badges are unlikely



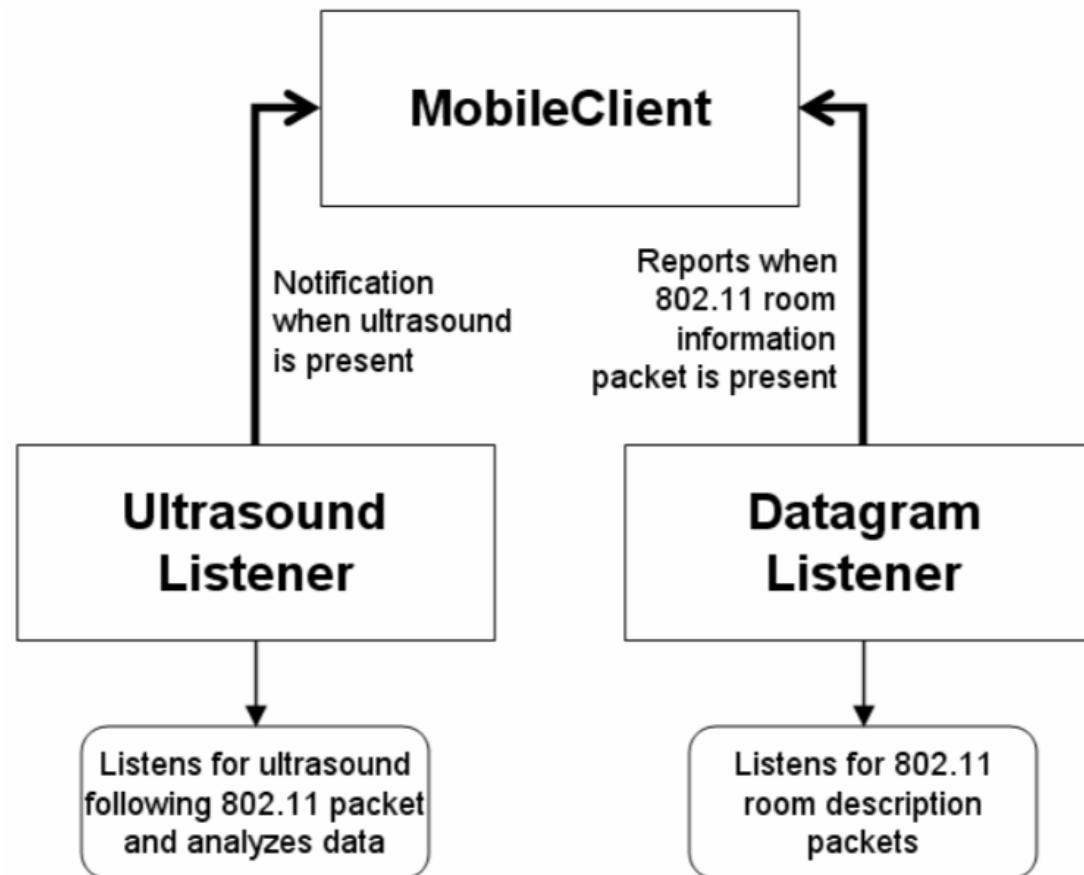


Room-Level Localization via Proximity- ActiveBadge

- Infrared signals are **easily blocked by fabric**:
 - the badges need to be worn on the **outside of people's clothing**
- Badges clipped to belts or pockets were often **obscured by tables and desks** when people were seated:
 - “**clipped to a breast pocket**” emerged as the best position in which to wear the Active Badge
- Simple design and use of low power parts enable:
 - an Active Badges to **run for more than a year** before requiring the batteries to be changed
- **Simplicity** in design and architecture:
 - commercial infrared badge systems as those sold by Versus have changed little in the nearly 20 years since the Active Badge system was developed
- Disadvantages:
 - **requires specialized infrastructure** in the form of base stations to listen for badge IDs
 - **requires users to purchase and carry the badge itself**



Room-Level Localization via Proximity- WALRUS





Room-Level Localization via Proximity- WALRUS

- WALRUS uses **PC and PDA speakers and microphones** to **send and receive ultrasonic signals** to convey proximity:
 - PCs are the ultrasound and radio **beacons**; badges are the **receivers** (mic and wireless)
 - it **does not encode the room's ID** using the ultrasonic signal
 - it **encodes the room's ID using a radio side channel**
 - the **PCs that are serving as beacons** periodically emit a 10-ms pulse of 21-kHz ultrasound
 - at the same time, **PCs send an 802.11 datagram containing the rooms' identifying information**
 - as the **ultrasounds and radio datagram are transmitted at the same time**, WALRUS clients do not need to constantly monitor their microphones
 - rather, **they can listen for an 802.11 datagram and, only when received, listen for an ultrasound signal**
- Given that there is no actual data encoded in the ultrasound:
 - **the client's audio task is reduced to simply looking for sufficient energy** in the 21-kHz band to denote proximity
 - this also extended the useful range of the ultrasound signals generated by the beacons to 13 m in their experiments



Room-Level Localization via Proximity- WALRUS

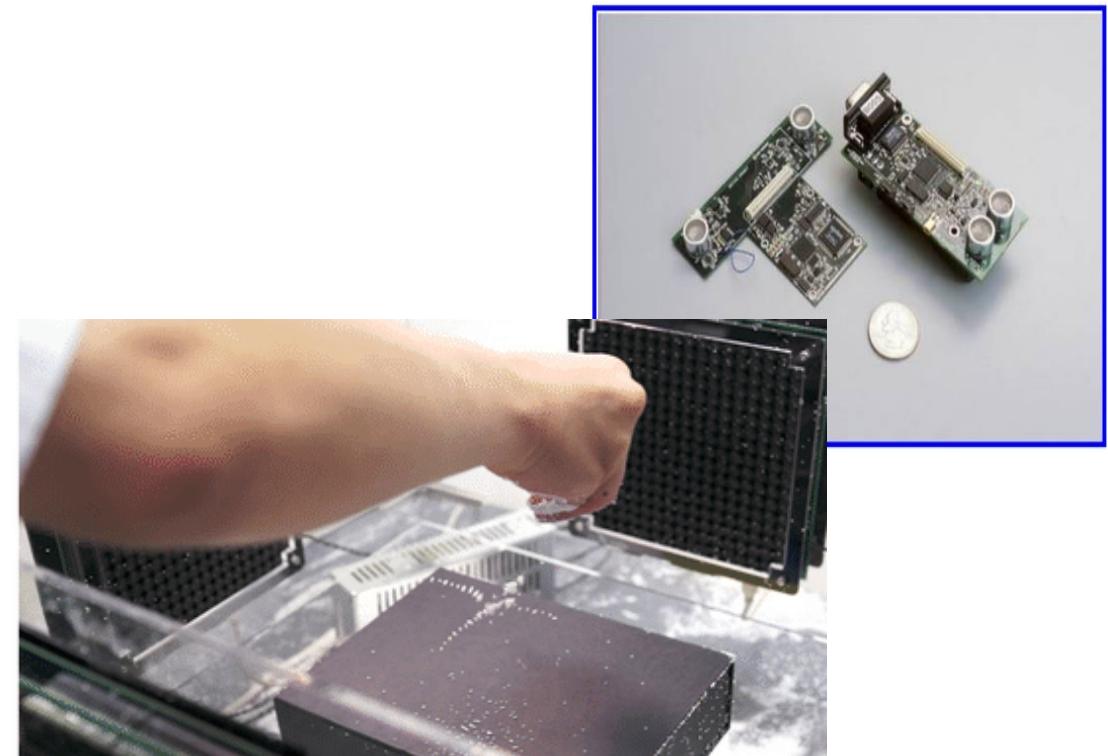
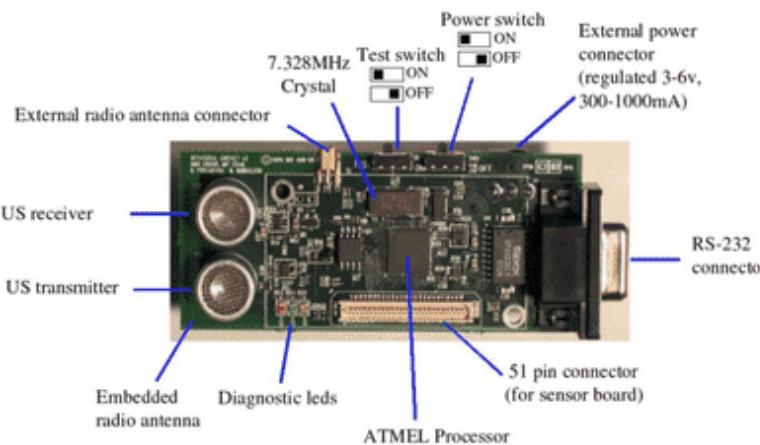
- The mobile client component is designed to run whenever it hears an 802.11 location packet:
 - it **records audio** from the microphone for enough time for a typical room size (e.g., 50-100ms can handle a good sized room of 18m maximum dimension at 25°C)
 - it then **looks for energy** in the received audio in a small band around 21KHz:
 - if there is a signal there, then **it is likely the device is in the room that generated the last location** information-bearing 802.11 packet
 - conversely, **if there is no energy at 21KHz, then it is likely that the device is in a different room**
 - several readings over a few seconds can quickly provide a high-confidence room-level location estimate
 - note that the client device only expends energy on ultrasound detection when it hears an 802.11 location packet; the rest of the time, it performs no ultrasound calculations at all



Room-Level Localization via Proximity- WALRUS

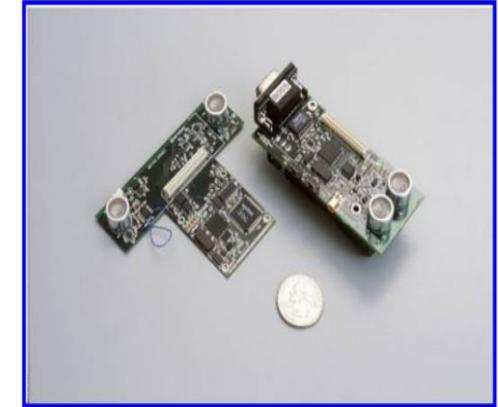
- Because **beaconing PCs are not coordinated** in WALRUS:
 - there is the potential for **client devices to receive the radio datagram from one beacon and associate it with the ultrasound from a different beacon**, in effect, incorrectly identifying which room it is in
 - the **experiments show that while this is unlikely** when only a small number of WALRUS beacons are within 802.11 range of each other, by the time **the number of beacons grows to 25, the error rate increases to more than 50%**
 - this issue can be addressed by **using a central coordinator to schedule the PC beacons**

Sub-room Accuracy – Cricket





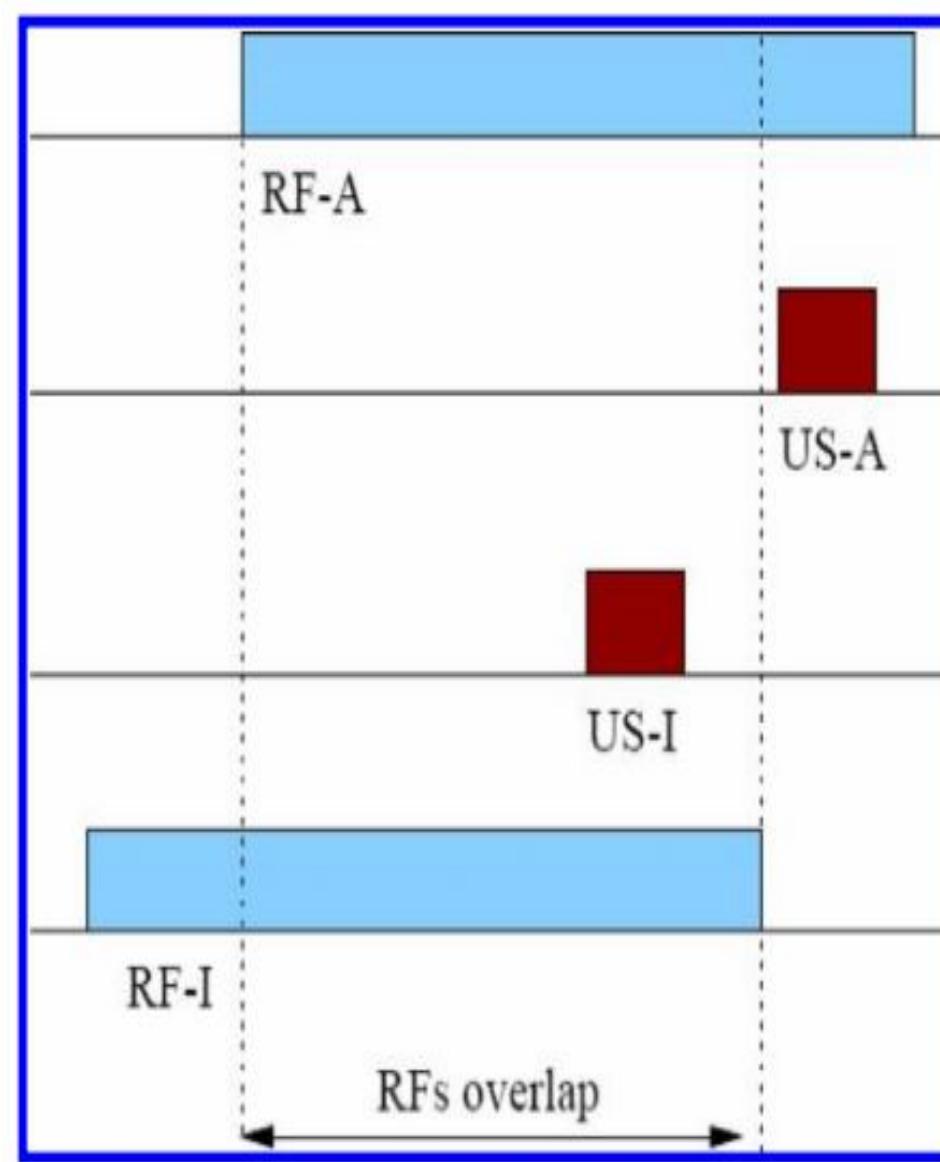
Sub-room Accuracy With Ultrasonic Time of Flight



- The **Cricket** system:
 - developed as part of the MIT Oxygen project
 - sought to enable **sub room-level location using ultrasound**, and
 - to do it in a **privacy preserving, decentralized fashion**
 - **badges** are the receivers/listeners; **beacons** are the infra-structure
- **Beacons are placed on ceilings or high on the walls in the environment and advertise their identity using a combination of RF and ultrasound:**
 - **Cricket beacons encode their identity using RF and send ultrasound pulses containing no digital data (similar to WALRUS)**
 - Cricket designers wanted to be able to place multiple Cricket beacons in a single room to allow Cricket listeners to figure out which part of the room (which seat at the table, which work area, which bed, etc.) the user was occupying
- Provides positioning precision to within 1m^2 regions within a room (since the ultrasound does not travel through walls).



Cricket





Sub-room Accuracy With Ultrasonic Time of Flight

- To do this, the Cricket system:
 - ultrasound travels very slowly (in room-temperature air, sound travels at only 0.34 m/ms.)
 - on receiving the radio ID from a beacon, **Cricket listeners not only listen for an ultrasound pulse, but time how long it takes to arrive**
 - because the radio signal travels almost instantaneously relative to the ultrasound:

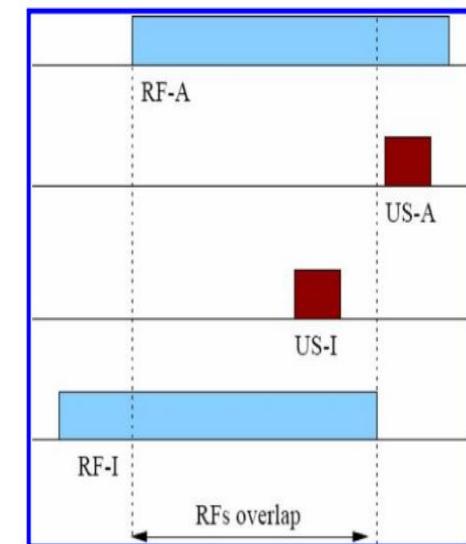
the delay between start of the radio transmission and the observation of the ultrasound pulse is a close approximation of the time of flight of the ultrasound which can, in turn, be converted into an estimate of the distance the ultrasound travelled

- by **estimating and comparing the distances of the beacons being observed**, a **Cricket listener can accurately predict which beacon is the closest**
- Thus, Cricket can be used as:
 - a **room-level location system**, when **beacons are sparsely deployed**, and
 - a **sub room location level system**, when **multiple beacons are placed in a room**
- Subsequent extensions to Cricket show that:
 - with **sufficient beacon density**, their system could **determine location within several centimeters and orientation to within 5°**

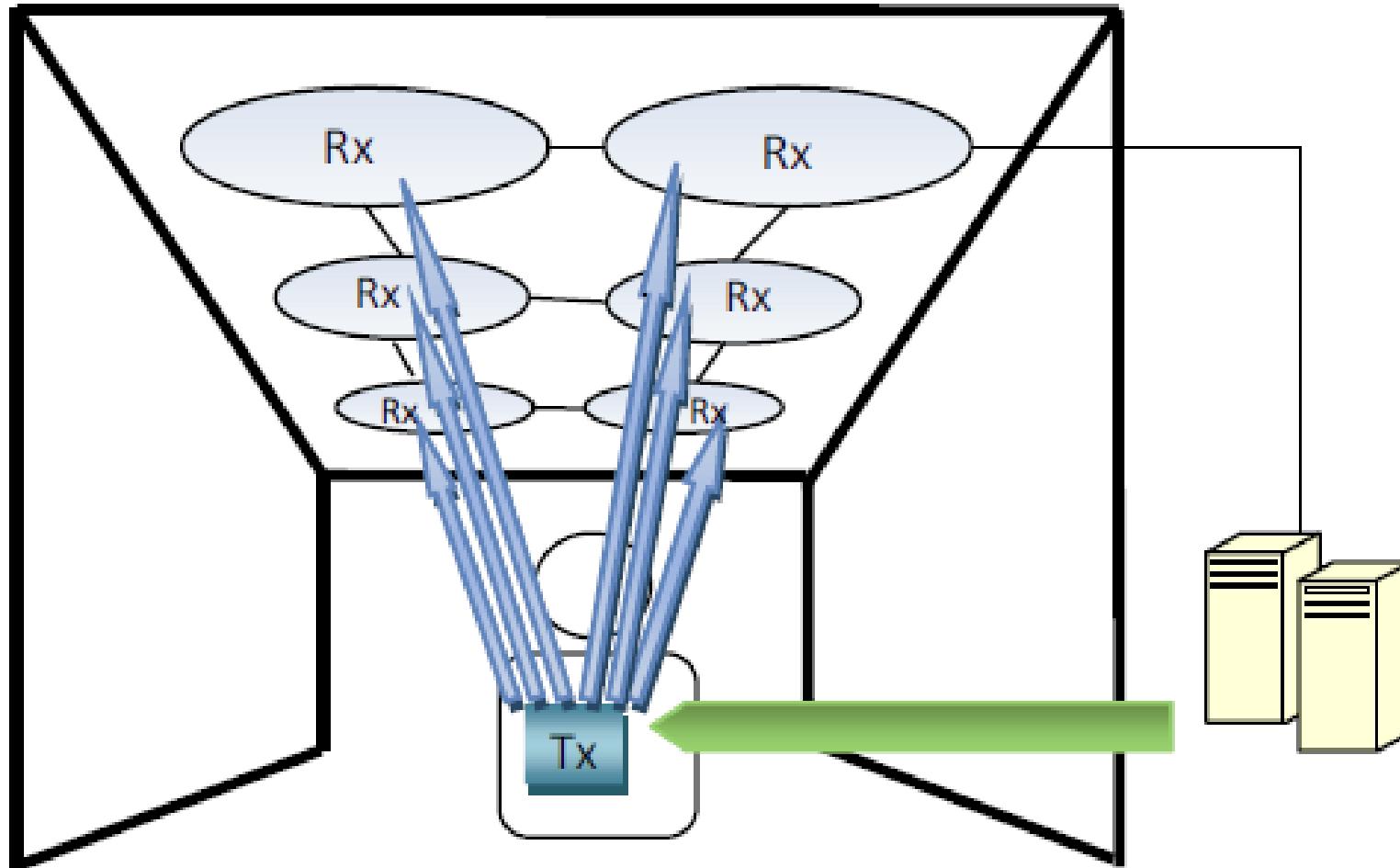


Sub-room Accuracy With Ultrasonic Time of Flight

- Preventing listeners from associating the ultrasound pulse from one beacon with the radio broadcast from another:
 - given that the **maximum effective ultrasonic range of the Crickets is 10 m**, the Crickets beacons intentionally transmit their ID using a **slow radio encoding that takes as long as ultrasound takes to propagate 10 m**
 - this ensures that the **radio transmission from each cricket beacon fully overlaps the flight of the ultrasound pulse** from the same source
 - this overlap guarantees that **at the time a listener receives an ultrasonic pulse from a beacon, it will still be receiving that beacon's ID via radio**
 - this allows the **listeners to detect (and ignore) the RF collision** that would occur **if two or more ultrasound packets were in flight at the same time**
 - this ensures that pulses will never be incorrectly associated with the wrong beacon ID
 - to reduce the chance that these collisions occur, **Cricket beacons randomize the time between transmissions** (e.g. beacons wait between 150 and 350 ms between broadcasts)



Absolute Location With ToF – Active Bat



Absolute Location With Time of Flight

- The goal is to estimate with **high accuracy** the absolute location of a device within a room:
 - e.g., 110 cm south, 50 cm east, and 35 cm up from the northwest corner of Conference Room 206
- Badges are the **beacons**; **receivers** are in the infra-structure
- The Active Bat system:
 - was developed by AT&T Research in 1999
 - it uses ultrasound receivers to localize small (35 g) pager-like devices called Bats in three-dimensional space
 - it is extremely accurate, estimating location correctly within 5 cm 50% of the time and within 9 cm 95% of the time
- Like Cricket:
 - Bats are located by measuring the travel time of an ultrasonic pulse
- In contrast to Cricket:
 - Bats emit the ultrasonic pulses, and receivers in the infrastructure listen for and time the pulse's travel





Absolute Location With ToF – Active Bat



- Accuracy of the Active Bat system derives from:
 - its **dense deployment of ultrasound receiver units**
- In the original AT&T deployment:
 - **100 ceiling-mounted receivers were used in a 100-m² office**
 - this density makes it likely that a **Bat's pulse would be heard by at least three** and likely many more receivers
- To estimate location:
 - **receivers pool their time-of-flight observations to a central server** that performs multilateration (a generalization of trilateration)
 - **location** is estimated from three or more ranging estimates from known locations
- Similar to Cricket:
 - reflections of ultrasound are common, and
 - the Bat system uses the redundancy of the grid of receivers to reject these reflections as outliers



Absolute Location With ToF – Active Bat

- The Active Bat system is sufficiently accurate that:
 - it is possible to place multiple Bats on a rigid object and,
 - by measuring the Bat's location, reconstruct not only the object's **location** but also its **orientation**
- Experiments in which two **Bats were** attached to a rigid object **22 cm apart** showed:
 - **orientation could be estimated within 10° over 80% of the time**
- The accuracy of the Bat system is dependent:
 - not only on the **density of ultrasound receivers**, but also
 - on **centimeter-accurate knowledge of each receiver's location**
- Active Bat system is centrally coordinated
- Bats wishing to be tracked register their interest with a central service via radio:
 - this service employs a slotted schedule to ensure that, at most, one Bat at a time has an ultrasonic pulse in flight
 - the system designers determined that ultrasonic reverberation can take up to 20 ms to die off in an indoor environment
 - thus, they use a schedule with 50 slots per second



Absolute Location With ToF – Active Bat

- At the start of a slot:
 - the **central server requests (via radio) that the Bat with a given ID send an ultrasound pulse**
 - then, **the server records the elapsed time and ID of each receiver that reports having heard the pulse**
- At the end of the slot:
 - **the server can compute the distance estimates and,**
 - **using the receiver's locations, can estimate the Bat's location**
- The Bat system:
 - does not need to employ a uniform schedule, and
 - Bats assigned to people can be localized with a higher frequency than those attached to less mobile objects like printers



Comparison

The vertical scale on the left indicates performance levels from worst at the top to best at the bottom. It features 10 star icons, with an upward arrow on the left side.

Technology	Infrared proximity (e.g., Active Badge)	Ultrasound proximity (e.g., WALRUS)	Ultrasound TOF (e.g., Active Bat)
Accuracy	★★★★☆ Room ID with high accuracy	★★★★☆ Room ID with high accuracy	★★★★★ 3D location with 5 cm accuracy
Coverage	★★★★☆ Indoor only in room fit with IR receiver/beacon	★★★★☆ Indoor only in room fit with ultrasonic beacons	★★★★☆ Indoor only in room fit with ultrasonic infrastructure
Infrastructure cost	★★★★☆ Infrared receiver or beacon required for each room	★★★★☆ 1 or more ultrasonic receiver or beacon required for each room	★★★★☆ Requires dense array of ultrasonic receivers
Per-client cost	★★★★☆ Inexpensive IR badge/dongle required	★★★★★ Software only solution on device with microphone	★★★★☆ Inexpensive ultrasonic badge/dongle required
Privacy	★★★★★ If localization is performed on the client. Otherwise ★★★★☆ Opt-out easy by removing badge	★★★★★ Localization is performed by mobile client	★★★★☆ Localization is performed by infrastructure. Opt-out easy by removing badge
Well-matched use cases	Asset and personnel tracking, indoor mapping/navigation/tour guides	Asset and personnel tracking, indoor mapping/navigation/tour guides	Asset and personnel tracking, tangible UIs, fine-grained info services



Location Estimation With 802.11



Introduction

- Estimating location with 802.11 was **first proposed in 2000**:
 - since then, many variants have been developed.
- At their most basic, **all of these systems work the same way**:
 - 802.11 radios and their supporting drivers allow a device to scan for nearby 802.11 APs
 - regardless of the variant of 802.11 (a, b, g, n) or whether encryption is enabled, these scans return a list of the nearby APs and their unique IDs called MAC addresses
 - all of the 802.11 location systems capitalize on the fact that 802.11 access points have limited range (typically less than 100 m), and **if a device can hear an access point, it knows that it is in the vicinity of that access point**.
- If a **device can see more than one AP**:
 - it can **estimate its own location more precisely**, and additional information, such as **received signal strength and packet loss rate**, can be used to improve the accuracy of location estimates further.



Introduction

- The appeal of using 802.11 for location estimation is strong:
 - **nearly all mobile devices** have built-in 802.11
 - AP density is high, e.g. over a **half-million known mapped 802.11 access points** in the Tokyo metropolitan area
 - densities of this kind raise the possibility of **high-coverage indoor/outdoor location with no additional location infrastructure**
- This near-pervasive client hardware and infrastructure for doing 802.11-based location is the reason why dozens of such location systems have been developed
- To allow client devices to discover and associate with an access point:
 - the 802.11 protocol includes **beacon frames that APs can send to alert clients to their presence**
 - the frequency with which these beacon frames are sent varies by model of AP and commonly **ranges from tens to hundreds of beacon frames per second**
 - these frames **contain the human readable SSID of the network** “Joe’s hotspot” and the **MAC address of the AP** and whether the **AP is running encryption or not**
- **Client** devices can:
 - passively **learn about nearby APs** by listening for a small window of time on each of the 802.11 channels
 - alternatively, **clients also have the option of initiating an active scan** by sending a probe request which prompts access points to reply with a probe response very similar to a beacon frame



Signal Strength Fingerprinting



Signal Strength Fingerprinting - Idea

- Location estimation using radio fingerprints involves **two phases**:

- the **mapping phase**, and the
 - the **location estimation phase**

- Mapping phase:

- a **site survey is performed**, in which the visible APs and their observed signal strengths are recorded along with the location in which the observation was taken
 - to provide good coverage, the radio map readings typically span the entire physical space
 - to provide good accuracy, the **readings need to be of sufficiently high density**; typically, a reading is collected every few square meters



- Given this radio map, estimating location is straightforward:

- a **client in an unknown location performs a radio scan** and estimates its **location to be the place on the radio map whose scan most closely matches its observation**
 - **spatial variation** ensures that there will be enough variety in the radio environment to distinguish places from each other
 - **temporal consistency** means these distinctions are likely not to change between the time of mapping and the location estimation



Signal Strength Fingerprinting - Radar

- The first 802.11 location system, RADAR, was developed at Microsoft Research
- RADAR and the other systems like it are the most accurate 802.11-based location techniques:
 - capable of tracking device with **1- to 3-m accuracy** depending on the environment
 - it takes advantage of two properties of the 802.11 signals observed by client devices:
 - spatial variability
 - temporal consistency
- Due to the short range of 802.11 and the way signals are blocked and reflected by obstructions in the environment:
 - the **strength with which an AP is observed varies considerably spatially**, even over distances as small as a meter (we have all observed this when we found we could not get good network connection in one place, and we moved over one seat, and the connection improved drastically)
 - the **strength of an 802.11 signal tends to be temporally consistent** (if a given seat at a table is a good place to establish a connection to a particular AP today, it will likely be a good place to connect in a few minutes, the next day, or the next month)

Signal Strength Fingerprinting - Radar

- Radar:
 - builds a map of the radio environment by collecting location-tagged “**radio fingerprints**”, and
 - later uses this map to perform localization of mobile 802.11 devices
 - as most other similar systems, it uses the Euclidean distance in **signal space** as a measure of the similarity of the two radio scans
- For example, the distance in signal space for the pair of scans is

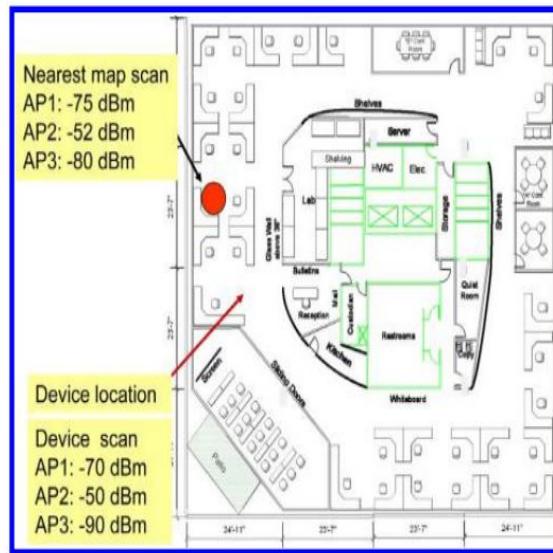
$$\sqrt{5 \cdot 5 + 2 \cdot 2 + 10 \cdot 10} = 11.4$$

AP	Scan 1	Scan 2	Difference
AP1	-70 db	-75 db	5 db
AP2	-50 db	-52 db	2 db
AP3	-90 db	-80 db	10 db

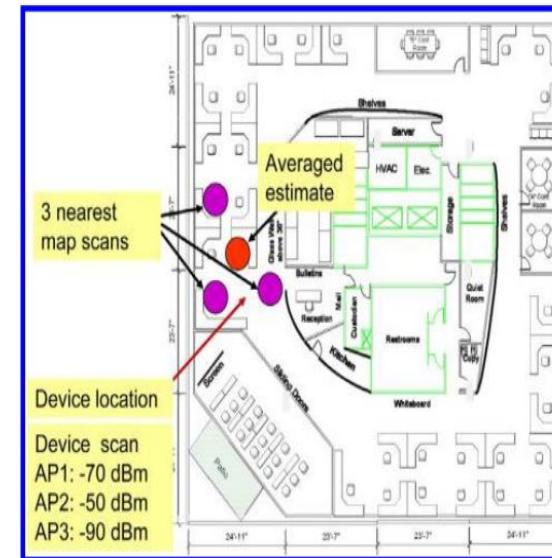


Signal Strength Fingerprinting - Radar

- The simplest algorithm for estimating location is called **nearest neighbor**:
 - the device's location is estimated to be the location of the mapping scan with the **smallest Euclidian distance from the client's scan**



nearest neighbor



nearest neighbor K=3

- Because there is some fluctuating interference from objects and radio sources:
 - **accuracy can be improved** by using more sophisticated algorithms
 - some systems including RADAR, smooth out variations by **averaging the locations of the K closest mapping scans** in Euclidian space
 - small K of 2-4 has been shown to yield good results



Signal Strength Fingerprinting - Accuracy

- The accuracy of any location system based on 802.11 radio fingerprinting is dependent on a number of factors:
 - **density** of the radio fingerprint map
 - **obstacles** in the mapped area
- E.g. the RADAR system was evaluated in:
 - a **1000-m²** office environment
 - with **70 fingerprint** locations,
 - averaging **4 m distance between neighboring fingerprints** and
 - yielded a **median localization error of 3 m** (versus a median error of 16 m for random)
- E.g. Horus and the commercial system Ekahau report:
 - **accuracies of 1 m** or better using **radio maps with 1–2 m between readings** on average
- A 2006 study using cars with GPS to collect 802.11 data showed that:
 - with a **mean distance of 10 m between fingerprints**,
 - it was possible to **estimate location with 15-m accuracy**



Signal Strength Fingerprinting - Accuracy

- Location systems are often used by people carrying mobile devices:
 - effect of human bodies on the accuracy of a fingerprinting location system
- To mitigate the effect of bodies attenuating the 802.11 signals:
 - RADAR's fingerprint maps were constructed with **four fingerprints from each location**,
 - with the person doing the mapping facing each of N, S, E, and W
- Rationale for taking these directional readings is:
 - to capture in the radio map **both different locations**,
 - and the **different ways a person may occlude the signals**
- With all four readings included in the radio map:
 - RADAR achieved **median accuracy of 3 m**
 - independent of the user orientation
 - when the fingerprint map was constrained to **only the north collected readings** and testing was done while **facing south**, errors increased by more than **60%**



Signal Strength Fingerprinting - Conclusion

- The effects of human occlusion on accuracy are indeed significant
- Other systems have varied in the number of mapping scans performed at each location:
 - in the case of maps generated from moving clients, from 1 to as many as 100
- The strength of 802.11 fingerprinting is that it is a sampling technique:
 - it does not need to understand where the APs are,
 - what the physical environment looks like, or
 - how 2.4-GHz signals propagate
- It works by simply:
 - creating an approximate snapshot of the radio environment, and
 - later sampling that snapshot
- Its simplicity is also its weakness:
 - site surveys are brittle, AP location changes are not that rare.
 - fingerprinting does not scale well.



Signal Strength Fingerprinting - Conclusion

- The radio maps produced by the site surveys are brittle:
 - if an **access point is moved**, the map needs to be recollected for the space within range of this AP,
 - even if the move is as insignificant as from the top of a filing cabinet to the desk beside it
 - as a result, fingerprinting systems are **best suited for spaces in which some degree of control over the radio infrastructure can be maintained** such as office spaces, hospitals, and other similar buildings or small campuses
- Fingerprinting does not scale well:
 - keeping a complete, up-to-date radio map for an office building or even a technology park might be fine, but
 - it is **not feasible at the scale of countries, states, or even large cities**
 - fingerprint maps can be collected sparsely with a predictable loss in accuracy in exchange for density reduction
- Other possible approach is to **model radio signal propagation**



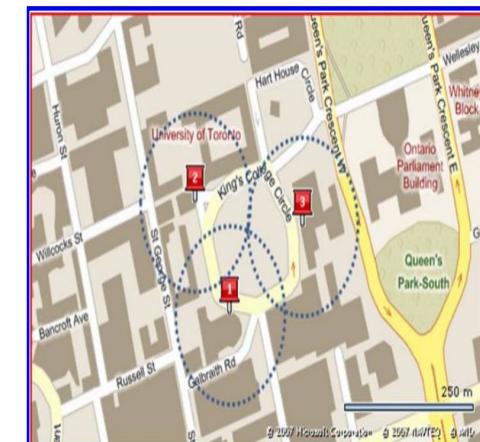
Signal Strength Modeling

Signal Strength Modeling - Idea

- At their core, all these systems take advantage of the same principle:
 - as **radio signals propagate in predictable ways**,
 - a **device's location can be predicted by modeling the propagation of the radio signals the device has observed**
- Models vary greatly, but all ask the same question:
 - “Based on what I know about the access points just observed, where is the device most likely to be?”

SSID	MAC	RSSI	Encrypted?
Capital City WiFi	00:0f:34:ab:0c:e0	-66 dB	No
First Security Trust	00:0f:f7:0c:e9:c0	-78 dB	Yes
First Security Trust	00:0f:f7:0c:4f:03	-105 dB	Yes

- Simplistic radio model:
 - assume that “**802.11 APs have a range of 100 m**”
- This model, combined with these AP locations, leads to the (perhaps incorrect) inference that:
 - the **client device is somewhere in the intersection of the three 100-m circles** centered at the coordinates of the three





Signal Strength Modeling - Active Campus

- The first wide-area deployment of an 802.11 location system was done:
 - as part of the **Active Campus** project
 - on the **University of San Diego** campus in 2002
- One of the goals was:
 - to offer **location-based social networking services** to students,
 - a campus-wide **indoor/outdoor location system**
- The campus:
 - have pervasive 802.11 coverage,
 - but was far **too large to perform a fingerprinting site survey**
- One resource at the team's disposal:
 - recent inventory recording the physical locations of the university's 1000 access points
- With this data set:
 - developed algorithms to use this AP map to approximate a device's location given an 802.11 scan



Signal Strength Modeling - Active Campus

- It succeeded in achieving high coverage with **accuracy in the wide-area of approximately 20 m**
- Basic trade-offs in using a modeling-based approach:
 - 1) it makes **simplifying assumptions** that make the computation easier at the cost of **decreased accuracy**
 - we are assuming that all APs can be heard from the same distance in all directions (which is almost never true)
 - 2) even if we wanted to use a **more sophisticated model, we could not do so** if the AP database did not give us additional key parameters such as transmit strength or degree of obstruction that the model needed
 - this creates a **strong coupling** between the **content of the AP database, the complexity of the radio model**, and the resulting accuracy of the location estimation



Signal Strength Modeling - Active Campus

- The Active Campus location system used an **AP database** storing:
 - latitude, longitude, and floor number for each known access point
- It used the **observed signal strength from each access point to estimate the client's distance from that AP**
- The **client's latitude and longitude** were estimated by:
 - **combining these distance estimates** using an average that heavily weighted the distance estimate from the strongest AP
- In the tests:
 - Average estimation **accuracy of 22 m outdoors and 11 m indoors**
 - **AP densities observed indoors** are bigger than **outdoors** which is why there is higher accuracy indoors,
 - although it was also likely due in part to indoor obstructions effectively reducing AP ranges and shrinking the set of likely client locations
- User's floor always estimated to be the floor of the AP with the strongest received signal:
 - due to the heavy attenuation of concrete floors and ceilings, this technique yielded the correct floor 95% of the time



Signal Strength Modeling - PlaceLab

- What if we want to use ActiveCampus **outside the campus**?
 - Place Lab employed a simple radio model based on an average of the latitude and longitude of the observed access points weighted by signal strength
- The key contribution of Place Lab was:
 - an access point database could be quickly built using a technique known as **“war driving”**
- War driving is:
 - **finding 802.11 networks by driving around** in a car equipped with a GPS and an 802.11 device that is continually scanning and logging
 - by **post-processing the log**, the GPS and 802.11 readings can be correlated to produce a cloud of locations in which a given AP was observed
 - using a **model of signal propagation**, this cloud of geo-tagged radio observations can be used to **estimate the 802.11 access point’s physical location**
- The observation of Place Lab was that:
 - war driving tools such as NetStumbler and Kismet that were primarily being used to allow people to find open networks could also be used to quickly build AP databases for 802.11 location systems



Signal Strength Modeling - PlaceLab

- The strength of this approach is that;
 - war driving **maps all observable access points**,
 - regardless of whether they are in a school, a business, or a private residence,
 - and independent of whether their traffic is encrypted or not
- This allowed Place Lab to provide coverage beyond that of a managed network:
 - it was the first system to potentially offer city or country scale 802.11-based location
- Experiments with Place Lab yielded accuracy numbers very similar to those of Active Campus:
 - in the neighborhood with the sparsest AP distributions, Place Lab has a median accuracy of 23 m, while in the densest, it had accuracies of 15–20m



Signal Strength Modeling – AP Database

- Active Campus and Place Lab illustrate two popular means of constructing an access-point database
- The advantage of the Active Campus approach is accuracy:
 - knowing an AP's location on a floor plan or a building schematic allows someone to estimate the AP's absolute location within a meter of its true location
- The downside of the Active Campus approach is that it only works when a network manager knows the location of their networking hardware:
 - note that this does not completely eliminate the possibility of achieving wide area or high coverage with this technique
 - cities deploying metropolitan-scale WiFi programs or even national “hotspot” providers like T-Mobile likely know the location of their APs
- War driving:
 - allows vast swaths of terrain to be quickly mapped at the cost of reduced accuracy in the AP location estimates



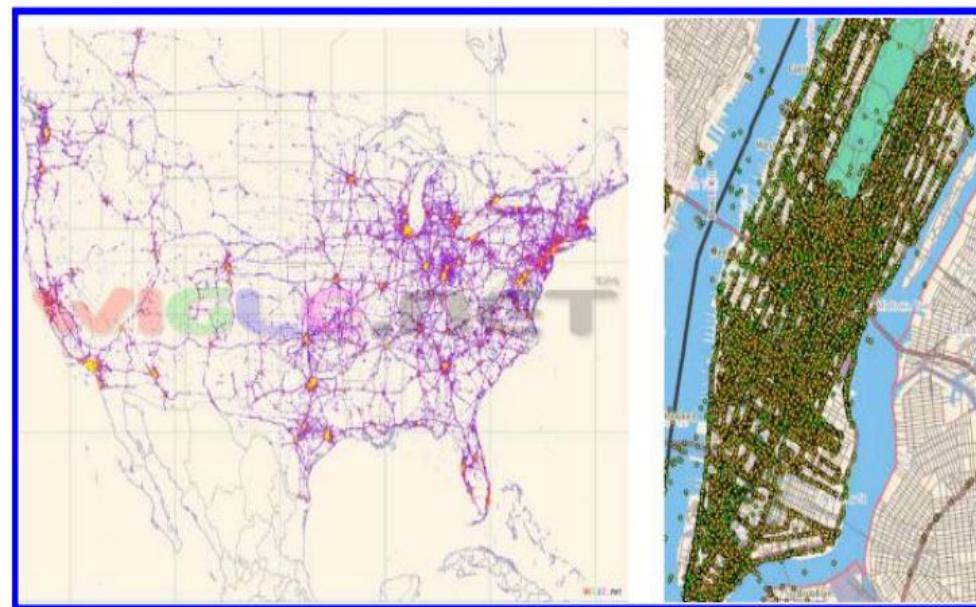
Signal Strength Modeling – AP Database

- In war driving, the errors in AP placement come from a variety of sources:
 - GPS is being used to record the location of radio observations, and **GPS itself has an 8 to 10m median error**
 - this is compounded by the **error introduced by the modeling** used to predict AP location from the set of observations
 - another source of error comes from that fact that all of the **popular war-driving programs ignore altitude** and estimate AP location in just latitude and longitude
 - while this may not introduce significant error for APs in one- or two-story buildings, Place Lab found that it had a **significant effect in downtown areas with tall buildings**
 - APs are **deployed, decommissioned, or moved** on a regular basis in the mapped area
 - this gives the data from a war drive an unpredictable, temporal fragility that a map of centrally controlled infrastructure could determine and even control
- Observations:
 - a comparison of war-driven AP locations to the true location of five APs in a residential area found a **median error of 26 m**
 - larger study of over 500 APs on Dartmouth campus - median error of **40 m for war-driven AP estimates**



Signal Strength Modeling – Conclusion

- Despite the drawbacks:
 - war driving has been far and away the largest source of AP data for location estimation
 - the commercial location system by Skyhook Wireless and Navizon employ nationwide war driving in the United States to build their AP maps
- In the public domain, the user-contributor war-driving repository “Wigle” contains the location of over 11 million APs



maps of Wigle.net's AP coverage for the continental United States and for Manhattan Island



Location Estimation With 802.11 - Comparison



Comparison



Technology	802.11 signal-strength fingerprinting (e.g., RADAR)	802.11 signal-strength modeling (e.g., Place Lab)
Accuracy	★★★★☆ 2D coordinates with 1-3 m median accuracy	★★★☆☆ 2D coordinates with 10-20m median accuracy
Coverage	★★☆☆☆ Building to campus scale. Requires 802.11 coverage and radio map. Best accuracy achieved when 3+ APs are visible.	★★★★☆ Areas with 802.11 coverage and radio map. Best accuracy achieved when 3+ APs are visible.
Infrastructure cost	★★☆☆☆ No additional infrastructure is needed beyond 802.11 APs. Creating radio map is time intensive and new/moved APs require remap.	★★★★☆ No additional infrastructure is needed beyond 802.11 APs. Creating radio maps is less work than for fingerprinting.
Per-client cost	★★★★★ Software-only solution for devices with 802.11 NICs.	★★★★★ Software-only solution for devices with 802.11 NICs.
Privacy	★★★★★ when localization is performed on the client. ★★☆☆☆ when localization is performed in the infrastructure.	★★★★★ when localization is performed on the client. ★☆☆☆☆ when localization is performed in the infrastructure.
Well-matched use cases	Asset and personnel tracking in indoor environments, indoor mapping/navigation/tour guides	Social networking, tour guides, indoor/outdoor navigation/tour guides, fitness/activity tracking



Cellular-Based Systems



Cellular-Based Systems - Introduction

- The development of location systems based on mobile phone technology was originally driven by:
 - the U.S. Federal Communication Commission E911 mandate, and
 - its European Community counterpart E112 to locate mobile phone calls to assist in the delivery of emergency services
- In addition, the wide adoption and ubiquitous connectivity of cellular phones makes them tempting platforms for the delivery of location-based services:
 - advertising, recommendation systems, and gaming
- A localization system based on cellular signals, such as GSM or CDMA, has the key advantages that:
 - it leverages the phone's existing hardware, and
 - can potentially provide location estimates anywhere voice service is available
- A number of mobile phone-based location systems have been developed, and they can be grouped into four categories:
 - cell ID-based approaches,
 - methods based on radio propagation modeling,
 - assisted GPS, and
 - surveying techniques based on radio fingerprinting
- Each type of location system strikes a different trade-off between ease of deployment, coverage, and accuracy



Cellular-Based Systems – Cell ID-Based

- A mobile-phone base station is typically equipped:
 - with a number of directional antennas that
 - define sectors of coverage or cells,
 - each of which is assigned a unique cell ID
- Cell ID-based location is a simple technique where:
 - the position of the mobile phone is estimated based on the ID of the cell currently providing service to the device
- Cell ID-based location is usually implemented on the network side, and its key advantage is:
 - it works for all phones, as no handset modifications are required
- Accuracy depends on the size of the cell, ranging:
 - from 150 m for microcells in urban cores to 30 km for cells in rural settings
- Such accuracy may be sufficient for some applications:
 - e.g. weather and traffic reports, but
 - falls short of the requirements for other application such as street navigation and the E911 guidelines (as these require a cell phone handset to be localized within 50 m 66% of the time)
 - accuracy can be improved by including in the position a calculation of the round-trip time (RTT)

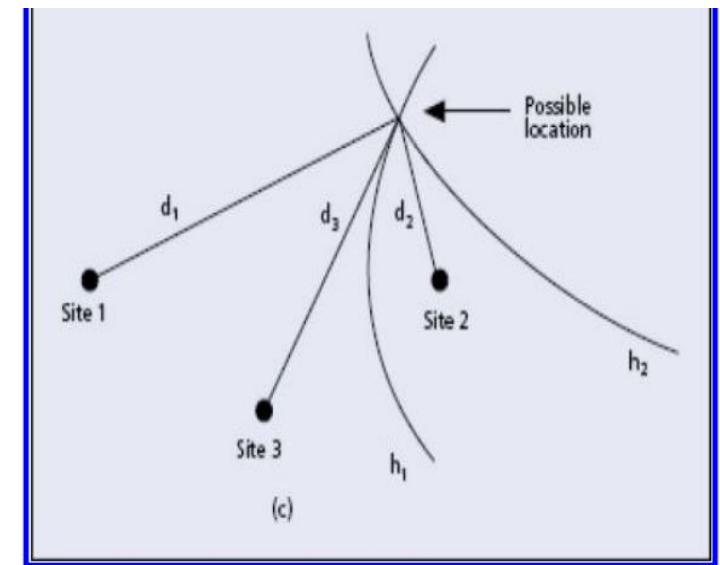


Cellular-Based Systems – Cell ID-Based

- The system by Laasonen is an example of a cell-ID based system:
 - it was implemented purely on the handset without network cooperation
 - it monitors the ID of the GSM cell the handset receives service from and uses the transition between cells to recognize the common places that a user goes to
 - it does not attempt to estimate absolute location, but rather assigns symbolic locations (e.g. Home and Grocery Store)
 - thus, it does not need to know the coordinates of the cell towers
 - the privacy improvement in this “handset-only” approach is marginal, as current protocols require the mobile phone to register with the cell tower to obtain the cell ID

Cellular-Based Systems – Radio Modelling

- A variety of location systems based on modelling of GSM and CDMA radio signals have been developed:
 - most are based on time-of-flight measurements (rather than signal strength)
- The underlying techniques used for computing ranges to cell towers depend on the specifics of the GSM or CDMA physical layer:
 - but the positioning algorithms used by GSM and CDMA are in fact very similar
- The main radio modelling positioning methods provided by GSM and CDMA systems are based on Time Difference of Arrival (**TDOA**) trilateration:
 - handset position is estimated by intersecting hyperbolic lines derived by taking differences between time measurements from pairs of base stations





Cellular-Based Systems – Radio Modelling

- TDOA requires base stations to be tightly synchronized:
 - this is not an issue for CDMA networks (follow a synchronous protocol)
 - in GSM, this requires the deployment of additional network elements to determine the clock difference between base stations
 - this information is then provided to the handset who uses it to calibrate the measurements from neighboring base stations
- TDOA can be implemented either on the **network** or the **client**
- Network-based approach:
 - three or more base stations **compare observations to determine the time differences** at which handset transmissions are heard at the cell towers
 - has the advantage that **it does not require modifications to the handset** hardware, and as a result, it works for existing, deployed handsets
- Client-based implementation:
 - the **handset measures the time differences in the arrival of training sequences** or pilot symbols transmitted by three or more stations
 - provides a **higher degree of privacy to users**, as time-difference is measured and the location estimated on the user's local device base stations



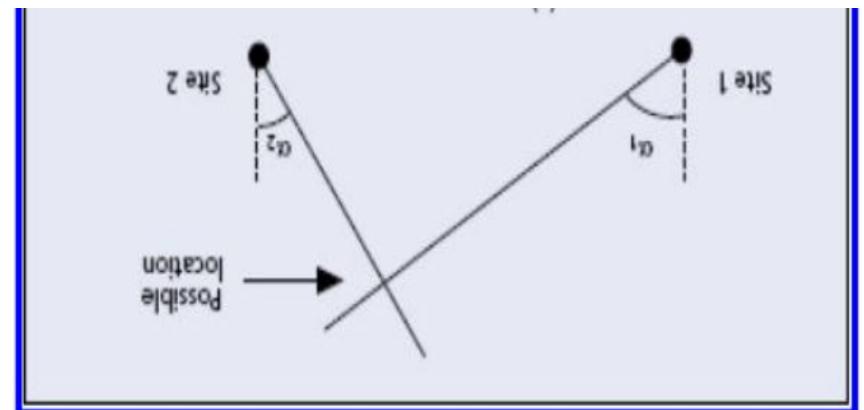
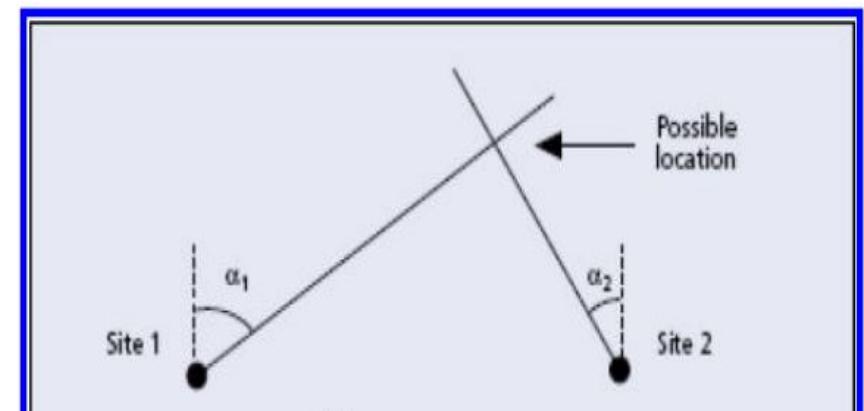
Cellular-Based Systems – Radio Modelling

- TDOA **accuracy ranges between 50 and 500 m** depending on interference, system geometry, and multipath effects:
 - handsets in close proximity to a base station may suffer from **interference** issues where the strong signal of the nearby base station prevents the handset from hearing the transmission of neighbouring nodes
 - this problem is more pronounced for CDMA systems, where different towers transmit on the same frequency, than on GSM networks, where towers are allocated different frequencies
 - to alleviate, base stations stop transmission for short idle periods that enable the handset to measure other neighbouring nodes
- **Multipath effects**, resulting from signal reflecting on obstacles degrade the accuracy of the technique by lengthening the perceived distance to the base station

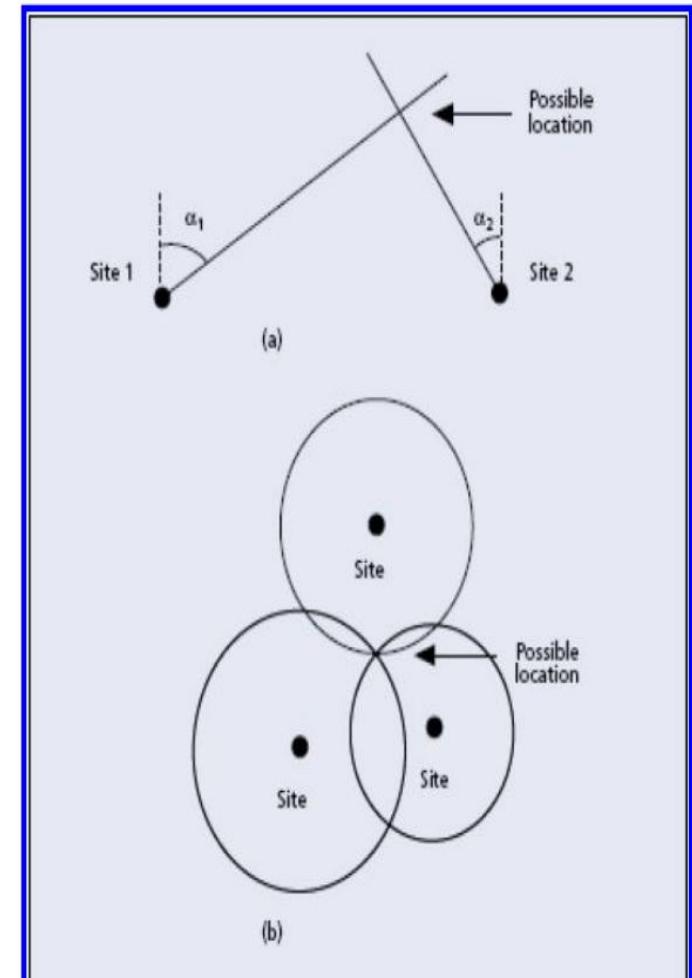


Cellular-Based Systems – Radio Modelling

- In addition to TDOA, GSM, and CDMA also provide positioning methods based on angle or arrival (AOA) and time of arrival (TOA)
- AOA is a method that determines position by electronically steering a directional antenna or an antenna array:
 - handset position is determined using triangulation, by intersecting a minimum of two directional lines of bearing



- The most significant drawback and advantage of AOA is:
 - it **requires specialized antennas and receivers**, which limits this approach to network-based implementations
 - it has the advantage that **it does not require system-wide synchronization** and is **more robust to multipath** propagation effects
- In TOA, the handset position is determined by intersecting range circles:
 - TOA systems can be implemented at either the network or the client
- A significant downside of TOA systems is that:
 - the measuring device has to have accurate knowledge of the time of transmission to determine the signal propagation time
 - note that TDOA systems get around this requirement by taking the difference between measurements



intersection of two hyperbolic curves derived from measurements to three base stations:

- (a) angle of arrival;
- (b) time of arrival;



Cellular-Based Systems – Radio Modelling

- GSM and CDMA location systems based on the measurement of **received signal strength** have also been developed:
 - it is less common than time-based approaches, as it typically achieves lower performance
 - e.g. Place Lab system, which complements their 802.11 signal models with a GSM model that predicts tower distance based on received signal strength
- In Place Lab:
 - GSM cell tower locations are estimated using war-driving tools similar to those used to estimate 802.11 base station locations
 - e.g. in three Seattle neighborhoods, Place Lab achieved median accuracy between 100 and 200 m depending on building and tower density; more densely populated neighborhoods, which tend to have a higher concentration of base stations, had better accuracy
 - the handset-based approach enables location determination without the need for cooperation from the network operator but requires the mobile to store a database of tower locations



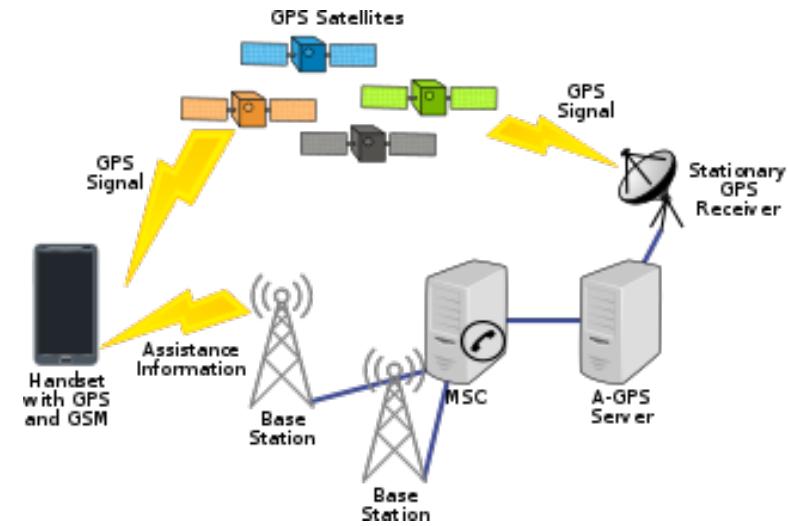
Cellular-Based Systems – Signal Strength Fingerprinting

- This solution is very similar to the fingerprinting systems based on 802.11 but there are with some differences:
 - due to higher coverage, cellular fingerprinting has the **potential to work in more places** than 802.11 fingerprinting
 - because cell towers are more dispersed across the covered area, a cellular-based localization system would **still work in situations where a building's electrical infrastructure has failed**
 - the significant expense and complexity of cellular base stations result in a network that **evolves slowly** and is only **reconfigured infrequently**
 - while this lack of flexibility (and high configuration cost) is certainly a drawback for the cellular system operator, it creates a stable environment, and **radio maps for cellular systems will degrade at a slower rate** than those based on 802.11
 - due to shorter range, **802.11 fingerprinting is more accurate** than cellular fingerprinting given the same number of radio sources

Cellular-Based Systems – A-GPS

- A-GPS:

- Provides GPS satellite information to accelerate and bootstrap mobile phone location.
- Provides accuracy in the tens of meters, but is still only available on a fraction of mobile handsets



- A-GPS and TDOA complement each other to a large extent:

- A-GPS is expected to perform best in rural and suburban environments where few obstructions provide for good satellite visibility; in this environment, however, low base station density leads to poor TDOA performance
- conversely, in downtown areas and other dense urban environments, TDOA has higher coverage than A-GPS, as it works (albeit at reduced accuracy) indoors and in other obstacle-rich environments

Cellular-Based Systems – Comparison

- Three location services: Cell-ID, TDOA, and A-GPS
- Each of these types of location services strikes a different trade-off between:
 - ease of deployment, coverage, and accuracy
- Cell ID provides:
 - the lowest accuracy, but
 - because it does not require additional hardware on the network side, it is the one method that is universally supported by all network providers
- TDOA:
 - provides higher accuracy, but
 - because it requires changes to the network, the handset, or both, it is not yet universally supported

Cellular-Based Systems – Comparison

Technology	GSM signal-strength fingerprinting	GSM TOF and signal-strength modeling	GSM/CDMA proximity	Assisted GPS (A-GPS)
Accuracy	★★★★☆ 2D coordinates with 4 m median accuracy in dense cell environment	★★★★☆ 3D coordinates with 100 - 200 m accuracy	★★★★☆ Accuracy dependant on cell tower density (150 m - 30 km)	★★★★☆ 3D coordinates with 10 -150 m accuracy depending on number of GPS satellites visible
Coverage	★★★★☆ Building to campus scale. Requires cell network coverage and radio map. Best accuracy when 3+ cells are visible	★★★★☆ Areas with GSM coverage and radio map. Best accuracy when 3+ cells are visible	★★★★★ Anywhere with cell coverage and cell-to-location map	★★★★☆ Outdoors with 4+ GPS satellites or indoors with cell network support + view of 1+ GPS satellite
Infrastructure cost	★★★★☆ No additional infrastructure is needed beyond cell network. Creating radio map is time intensive	★★★★★ No additional infrastructure is needed beyond cell network and map of tower locations	★★★★★ No additional infrastructure is needed beyond cell network and map of tower locations	★★★★☆ Beyond GPS constellation, requires deployment of fixed GPS receivers
Per-client cost	★★★★★ Software only solution	★★★★★ Software only solution	★★★★★ Software only solution	★★★★☆ GPS antenna and chipset required for handset
Privacy	★★★★☆ Even if location is computed on client device, the network still tracks a handset's associated cell	★★★★☆ Even if location is computed on client device, the network still tracks a handset's associated cell	★★★★☆ Even if location is computed on client device, the network still tracks a handset's associated cell	★★★★☆ Even if location is computed on client device, the network still tracks a handset's associated cell
Well-matched use cases	Asset and personnel tracking in indoor environments, indoor mapping/navigation/tour guides	Social networking, emergency response, neighborhood-scale information access, fitness tracking, outdoor mapping / navigation	Regional information access (weather, traffic, etc.)	Emergency response, indoor/outdoor information/tour guide services, personnel/pet tracking, activity tracking, gaming



Mobile and Ubiquitous Computing

2022-23

MEIC/METI - Alameda & Tagus

Privacy in Location Based Services

Introduction

- Increased popularity of mobile communication devices with embedded positioning capabilities (e.g., GPS) has generated **unprecedented interest in the development of location-based applications:**
 - browse through maps of their nearby areas and to ask queries about points of interest in their proximity
 - users can generate their own content with geospatial tags
 - location based social networks, or geosocial networks (GSN) allow users to share their whereabouts with friends, to find nearby contacts, and to provide/search for recommendations about points of interest that are close to their current geographical coordinates
 - geospatial crowdsourcing – mobile devices owners can act as mobile sensors (e.g. measure the levels of vehicle traffic congestion, the levels of air pollution, propagate instant information about the weather)
 - study of trajectory traces for mobility planning
- All the above exciting applications benefit from the **availability and potential for sharing of location information**
- However, uncontrolled **location sharing can have dire consequences**, when location data falls in the hands of malicious entities

Location Privacy

- With knowledge of user locations, an adversary can stage a **broad spectrum of attacks** against:
 - individuals, from physical surveillance and stalking, to identity theft, to inferring sensitive information, such as the individual's health status, alternative lifestyles, political and religious affiliations, etc.
- There are **three aspects of location information disclosure**:
 - position awareness,
 - sporadic queries, and
 - location tracking.
- **Position awareness** refers to the case:
 - where a device monitors an individual's location (e.g., an in-car GPS system), but no data is released to another party
 - the user's position is only used locally, to navigate a map for instance, hence no privacy threat occurs.
- The **sporadic (or one-time) queries** case refers to:
 - scenarios where **a user reports his/her current location to a service provider**, in order to find nearby points of interest (e.g., "find the closest restaurant").
- **Location tracking** occurs in:
 - applications that require **frequent updates of the user's position**, e.g., traffic monitoring.

Location Privacy

- Another important aspect in location disclosing is related to the **attacker capabilities**: weak and strong privacy
- **Weak privacy**:
 - requires that **no sensitive data should be *directly disclosed*** to a party that is not trusted
 - i.e. if the current location of the user does not reveal any sensitive information, it is safe to disclose
- **Strong privacy**:
 - **disallows the publication of location snapshots** which, although they do not represent a privacy violation by themselves, may be correlated to additional data to infer the presence of a user at a privacy-sensitive position
 - anonymizing trajectory data is a representative example where strong privacy is necessary
 - enforcing strong privacy must not have a significant negative impact on data accuracy, in the sense that the utility of the published data must be preserved
- These slides:
 - provide an overview of the state-of-the-art in location privacy protection from multiple perspective
 - reviewing the requirements and characteristics of several different location sharing application scenarios
 - solutions range from anonymization techniques using location generalization, to cryptographic techniques, geometric transformations, and differential privacy

Privacy Issues Tackled:

- Can a location based service (LBS) identify my location?
- Can an LBS be able to place me at a sensitive location?
- Can a third-party who accesses an LBS query database reconstruct my trajectory?
- Can a third-party with additional knowledge who accesses an LBS query database reconstruct my trajectory?

Privacy-Preserving Spatial Transformations

Two-Tier Spatial Transformations

Three-Tier Spatial Transformations

Introduction

- To preserve privacy, **the exact location of users that send queries to Location-Based Services (LBS) must not be disclosed:**
 - instead, **location data is first perturbed, and/or encrypted**
 - e.g. generate a few random fake locations and send a number of redundant queries to the LBS to prevent user identification
 - e.g. *spatial k-anonymity (SKA)* is enforced by generating a *Cloaking Region (CR)*—sometimes referred to as Anonymizing Spatial Region (ASR)—which includes the query source as well as $k - 1$ other users
 - e.g. obscure the location data using spatial or cryptographic transformations
- There is a **trade-off** that emerges **between privacy and performance:**
 - Location privacy techniques often degrade application functionality and usability.
 - achieving privacy **incurs an additional overhead in processing queries**
 - e.g. a **larger number of queries** need to be processed in the case of techniques that **generate redundant requests**
 - e.g. for **spatial k-anonymity** techniques, **query processing is performed with respect to the CR**, which is considerably **more expensive than processing point queries**

Introduction

- We can classify **existing privacy-preserving spatial transformation techniques** into **two categories**, according to the architecture they assume:
 - **two-tier spatial transformations**, and **three-tier spatial transformations**
- **Two-tier** spatial transformations:
 - do not require any trusted third party, and the **query anonymization is performed by the mobile user itself**
 - involve only two parties at query time: the user and the LBS provider
 - assume that no background knowledge is available to the attacker
- **Three-tier** spatial transformations:
 - assume the presence of a **trusted third-party anonymizer server**, and offer better protection against background knowledge attacks (e.g., an attacker may have additional information on user locations, from an external source)
- Trade-off:
 - methods in the **second category generate more runtime overhead**,
 - because they **require the users to constantly update their location with a central authority**, and
 - the algorithms used to generate **protecting cloaking regions are more computationally expensive**

Two-Tier Spatial Transformations

Dummy Locations

- Methods in this category involve only two parties at query time:
 - the **user** and the **LBS provider**
 - assume that **no background knowledge is available to the attacker**
- Simple solution to query privacy:
 - generate a number of redundant queries for each real query
 - e.g., user u could generate r random “fake” locations, and send r redundant queries to the LBS, in addition to the actual query containing u 's location
- Thus, **dummy locations** are generated such that:
 - the resulting trajectories mimic realistic movement patterns
- Dummy-generation algorithms can take into account movement parameters, such as velocity, and certain constraints, e.g., an underlying road network.

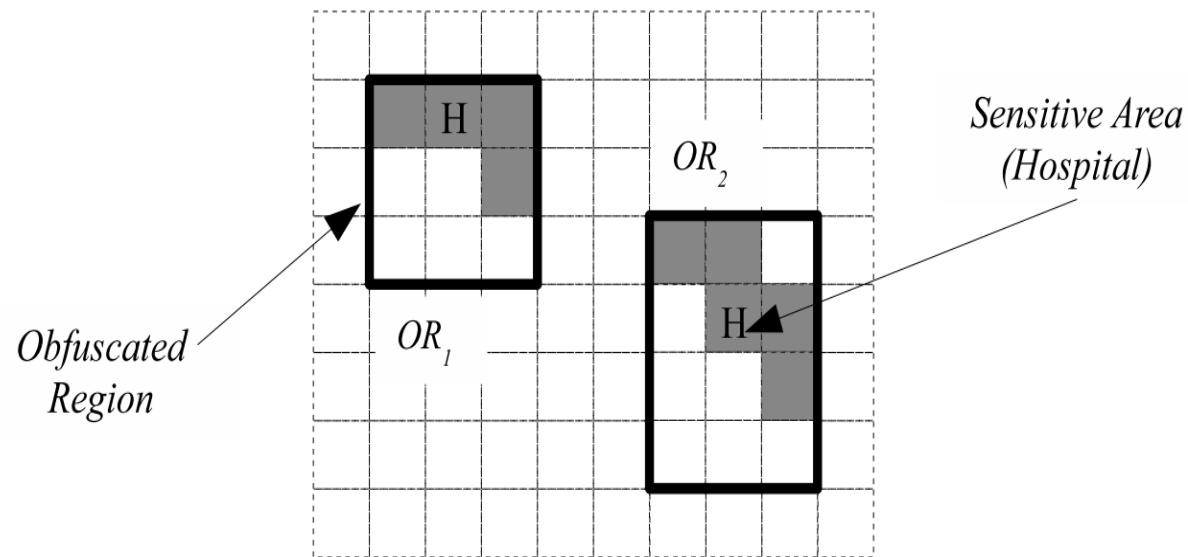
Probe (1/5)

Damiani, Maria, Elisa Bertino, and Claudio Silvestri. "PROBE: an obfuscation system for the protection of sensitive location information in LBS." TR2001-145, CERIAS (2008).

- It prevents the association between users and sensitive locations
- It is assumed that the attacker has access to all sensitive locations from a particular data space (e.g., a city, a country, etc.)
- Sensitive locations are represented by *features*, which are classified into *feature types* (e.g., hospitals, restaurants, etc.)
- In an off-line phase, an *obfuscated map* is constructed by:
 - partitioning the space into a set of disjoint regions such that
 - the probability of associating each region with a certain feature type is bounded by a threshold
- This process is called *obfuscation*:
 - may require an additional trusted third party, but
 - in the on-line phase (i.e., at query time) PROBE is a two-tier protocol

Probe (2/5)

- For example, ensure that no region can be associated with the “hospital” feature type with probability higher than 44%:
 - OR_1 contains nine grid cells in total, four of which are sensitive, and $4/9 = 0.44$

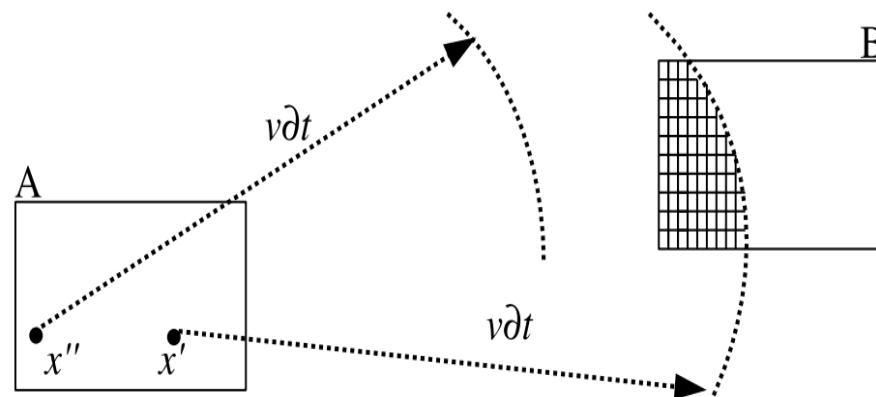


Probe (3/5)

- Another interesting aspect about PROBE is that **it can be extended to protect inference when users move across different obfuscated regions**
- Example:
 - consider the **division of US territory into zip-code areas**
 - the **map is partitioned into disjoint regions**, each of them covering an area of a few square miles; or, at a finer granularity level, a city can be sub-divided into block regions
 - **as the user moves, his/her location can be mapped to a city block identifier, and only the block identifier is disclosed**
- The *privacy requirement* in this case is:
 - **do not allow an attacker to pinpoint the user location within a sub-region of a reported obfuscated region**

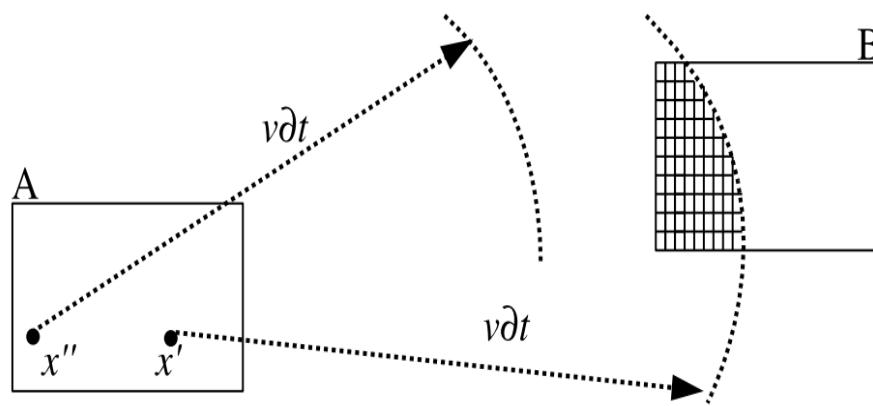
Probe (4/5)

- **Obfuscated regions A and B** are reported by user u at timestamps t_a and t_b , respectively where $t_a < t_b$
- Consider v the **maximum user velocity**, and let $\delta t = |t_b - t_a|$
- The **attacker** may try to **prune parts of A and B** to pinpoint u in two ways
- First:
 - determine if there is any location $x \in A$ from which the user cannot reach some location $y \in B$, even by traveling at maximum speed v
 - formally, an attack is successful iff $\exists x \in A$ s.t. $\forall y \in B, d(x, y) > v\delta t$
 - a user traveling from point x' is able to reach a point in the hatched region of B within time δt ; however, if the initial location of u were x'' , reaching B would not have been possible; therefore, an attacker can rule out a subset of A as possible positions for u , hence privacy is breached



Probe (5/5)

- Second:
 - determine if there is any location $y \in B$ which the user cannot reach from some initial location $x \in A$, even by traveling at maximum speed v
 - formally, $\exists y \in B \text{ s.t. } \forall x \in A, d(x, y) > v\delta t$
- To prevent privacy breaches, it is necessary to ensure that none of the two above equations ever holds



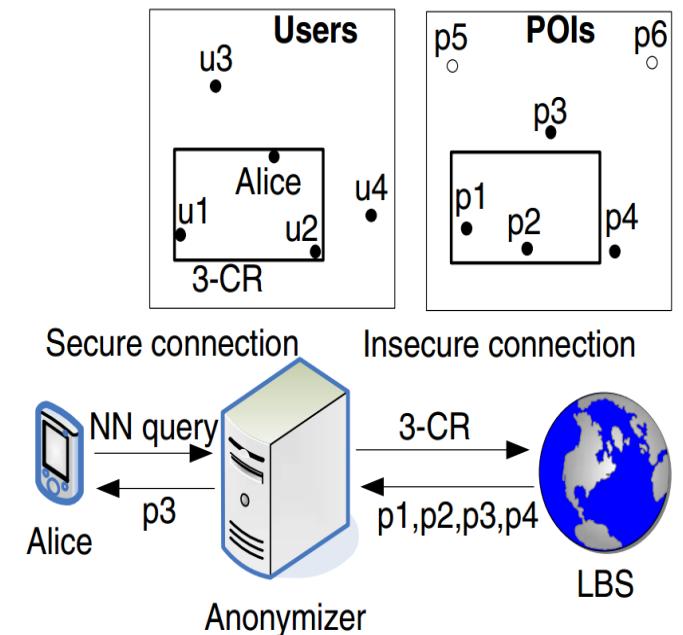
Limitation

- None of the two-tier spatial transformation solutions can prevent re-identification of the query source if an attacker has knowledge about specific user locations
- Example:
 - **if user u situated in a remote location issues a query, an attacker who knows that u is the only person residing in that area can associate u with the query, breaching user privacy**
 - the next category of query anonymization methods deals with this issue

Three-Tier Spatial Transformations

Spatial k -anonymity

- Methods in this category implement the **spatial k -anonymity paradigm**:
 - A **cloaking region (CR)** that contains $k - 1$ users in addition to the query source is generated, and
 - the LBS processes the query with respect to the CR
- Since **all the k locations enclosed by the CR correspond to actual users** (as opposed to “fake” locations in the previous category),
 - the **probability to identify the query source is at most $1/k$** , even if the attacker has knowledge about exact user locations
- Most solutions in this category employ a three-tier architecture

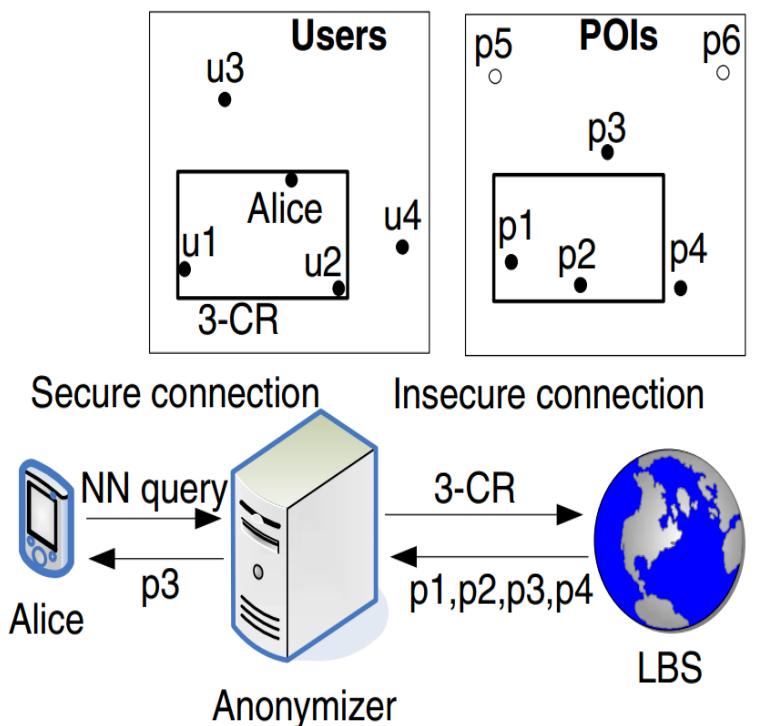


***k*-anonymity**

- ***k*-anonymity** was first discussed **in relational databases**, where **published data** (e.g., census, medical) **should not be linked to specific persons**:
 - methods for **computing aggregate functions** (e.g., *sum*, *count*) under the condition that **the results do not reveal any specific record**
 - compute **value distributions**, suitable for data mining, in confidential fields
- Recent work has focused on ***k*-anonymity** defined as:
 - a relation satisfies ***k*-anonymity** if every tuple is indistinguishable from at least ***k*-1 other tuples** with respect to a set of ***quasi-identifier*** attributes
- **Quasi-identifiers are attributes** (e.g., date of birth, gender, zip code) that **can be linked to publicly available data to identify individuals**
- **Records with identical quasi-identifiers** form an **anonymized group**

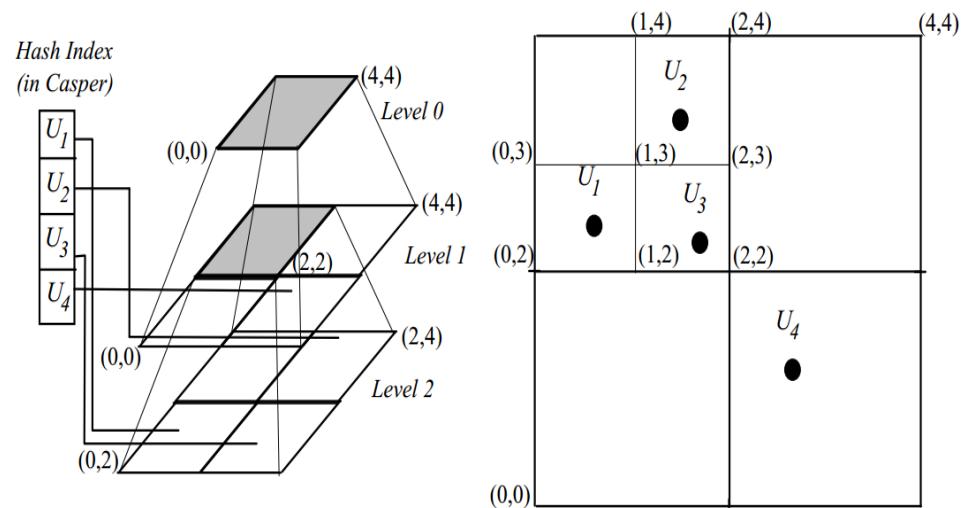
Spatial k -anonymity

- A trusted centralized *anonymizer* acts as an intermediate tier between the users and the LBS
- All users subscribe to the anonymizer and continuously report their location while they move
- Each user sends his query to the anonymizer, which constructs the appropriate CR and contacts the LBS
- The LBS computes the answer based on the CR, instead of the exact user location; thus, the response of the LBS is a superset of the answer
- Finally, the anonymizer filters the result from the LBS and returns the exact (?) answer to the user



Interval Cloak

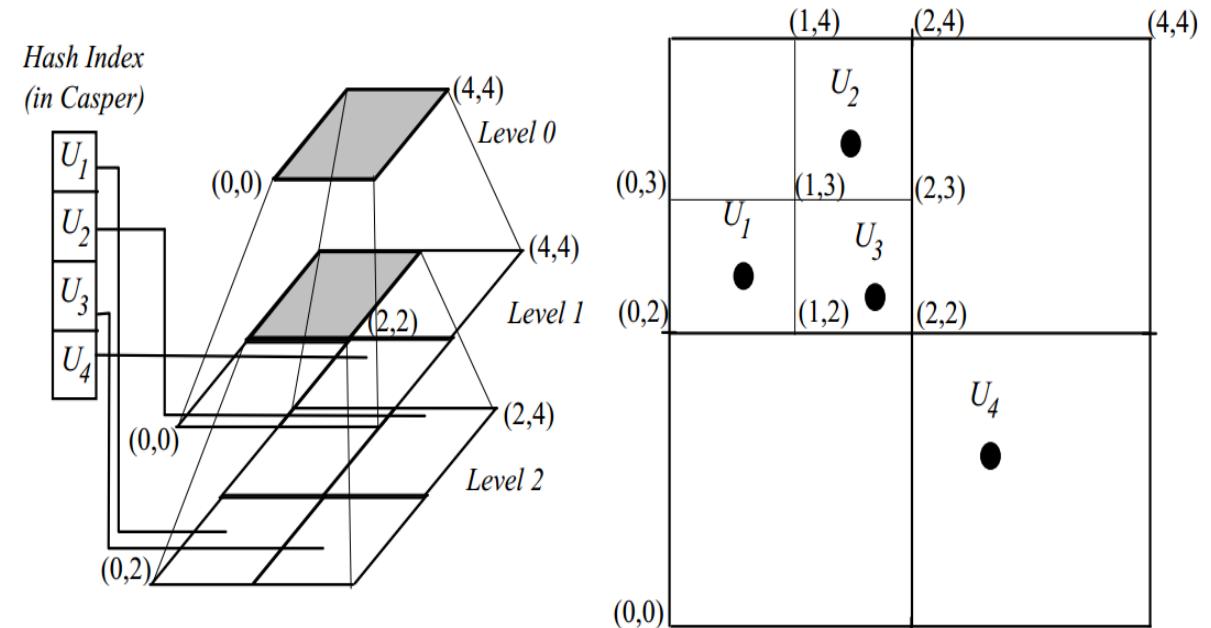
- Two techniques can be used to transform a relation to a k -anonymized one (both leading to information loss):
 - *suppression*, where some of the attributes or tuples are removed, and
 - *generalization*, which involves replacing specific values (e.g., phone number) with more general ones (e.g., only area code)
- **Interval Cloak** is based on quadtrees:
 - a quadtree recursively partitions the space in quadrants until the points in each quadrant fit in a page/node
- The anonymizer maintains a quadtree with the locations of all users
- Once it receives a query from a user U , it traverses the quadtree (top-down) until it finds the quadrant that contains U and fewer than $k-1$ users
- Then, it selects the parent of that quadrant as the k -CR and forwards it to LBS



space partitioning and a simple quadtree assuming that a node contains a single point

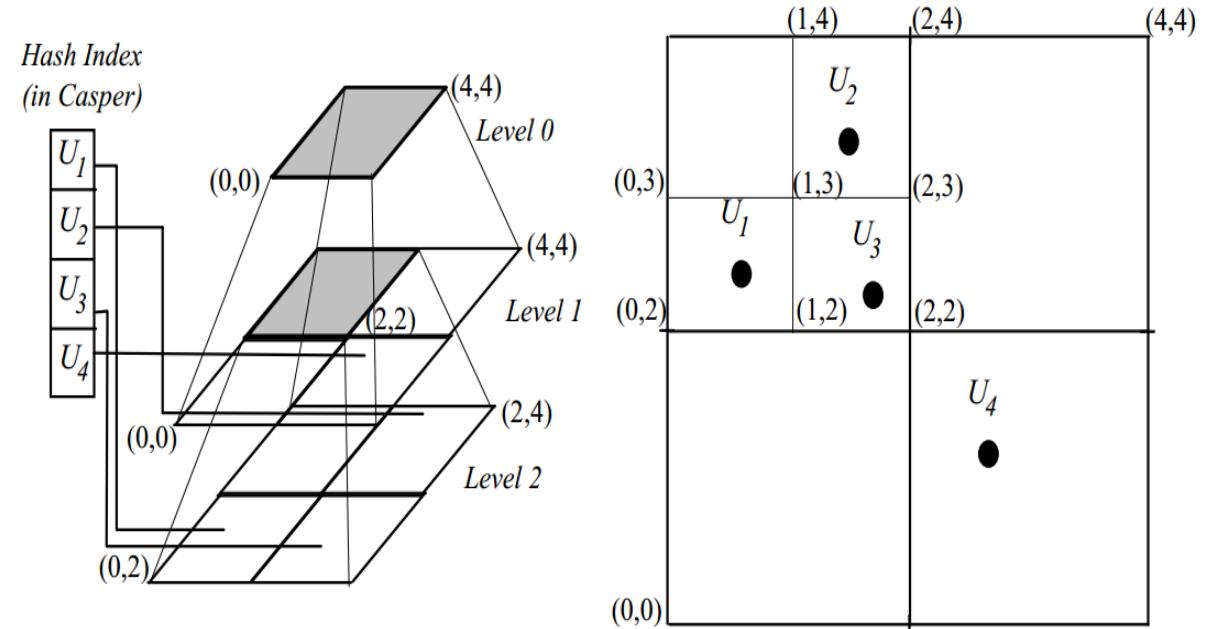
Interval Cloak

- Assume that U_1 issues a query with $k=2$
- Quadrant 2 $[(0, 2), (1, 3)]$ contains only U_1 , and its parent $[(0, 2), (2, 4)]$ becomes the 2-CR
- Note that the CR may contain more users than necessary:
 - in this example it includes U_1, U_2, U_3 , although two users would suffice for the privacy requirements
- A large CR burdens:
 - the query processing cost at the LBS, and
 - the network overhead for transferring a large number of candidate results from the LBS to the anonymizer



Casper

- Similar to *Interval Cloak*, Casper is based on quadtrees.
- The anonymizer uses a hash table on the user id pointing to the lowest-level quadrant where the user lies.
- Thus, each user is located directly, without having to access the quadtree top-down
- Furthermore, the quadtree can be adaptive, i.e., contain the minimum number of levels that satisfies the privacy requirements



- e.g., the second level for quadrant $[(0, 2), (2, 4)]$ is never used for $k \geq 2$ and can be omitted

Summary

- Methods in the category of **three-tier spatial transformations rely on the presence of other users to achieve spatial k -anonymity**
- Generally, these methods offer **stronger privacy guarantees than two-tier techniques.**
- The privacy features of PROBE and spatial k -anonymity methods are not directly comparable:
 - PROBE does not achieve k -anonymity, but it does provide spatial diversity
 - on the other hand, three-tier techniques may not always prevent association of users to sensitive locations
 - for instance, it is possible for an entire CR to fall within a sensitive region (e.g., hospital)

Mobile and Ubiquitous Computing
2022-23
MEIC/METI

Context-Awareness I

Context-Awareness (1/2)

- What is Context?
- What is the role and components of a Context-Aware Systems?
- Distributed Context-Aware Systems
- Local vs. Distributed Context-Aware Systems
- Collaborative vs . Non-collaborative Distributed Context-Aware Systems
- Examples of Sensors and Types of Context

Definition of Context

- The word “context” is subject to multiple interpretations and has been researched in various fields like psychology, philosophy, and computer science
- In the area of CS, context was initially perceived as user location
- In the last years, it has been enriched with other sources of information such as identity, activity, and state of people, groups, and objects
- *Context is any information that can be used to characterize the situation of an entity:*
 - *an entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*

Definition of Context (cont.)

- Consider only relevant information w.r.t. interaction between user and application
 - thus, application developers can focus on a subset of the user environment data for a given application scenario
- Example:
 - context needed by museum guide app to assist museum visitors is their location and native language
 - other environmental information (e.g. body temperature, marital status) is not relevant
- In mobile and ubiquitous systems, the idea is:
 - context supports systems to be cognizant of its user's state and surroundings and to modify their behavior based on this information
- A user's context can be quite rich, consisting of attributes such as:
 - physical location,
 - physiological state (such as body temperature and heart rate),
 - emotional state (such as angry, distraught, or calm),
 - personal history,
 - daily behavioral patterns

Definition of Context (cont.)

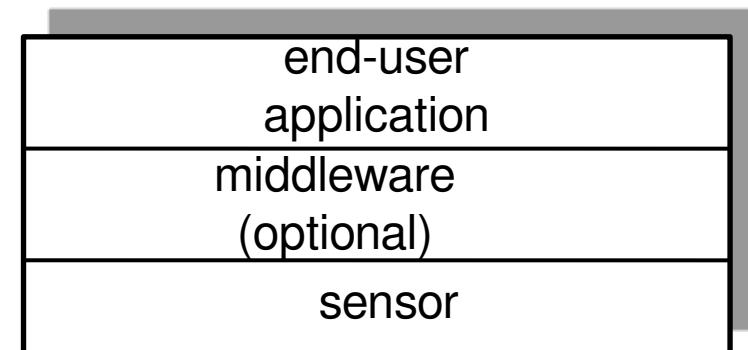
- Another definition (by enumeration of examples):
 - **computing context**, such as network connectivity, communication costs, and communication bandwidth, and nearby resources such as printers, displays, and workstations;
 - **user context**, such as the user's profile, location, people nearby, even the current social situation;
 - **physical context**, such as lighting, noise levels, traffic conditions, and temperature;
 - **temporal context**, such as time of a day, week, month, and season of the year
- Possible refinement:
 - there are certain types of context that are, in practice, more important than others for characterizing the situation of a particular entity:
 - *location, identity, activity, and time.*
- These context types not only answer the questions of **who, what, when, and where** but also act as identity indexes into other sources of contextual information.

Context-Aware Systems

- A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.
- This results in three main features that context-aware systems may provide:
 - **presentation of information and services to a user**
 - systems that provide information to the user augmented with contextual information (e.g., phone's contact list enhanced with location information) or that provide services based on the current user's context (e.g., show me restaurants nearby);
 - **automatic execution of a service**
 - systems that execute a service automatically based on the current context (e.g., automatically updating my status on a social network based on accelerometer data—sleeping, walking, and running);
 - **tagging of context to information for later retrieval**
 - systems that are able to associate digital data with the user's context (e.g., virtual notes that are attached to certain locations, for others to see).
 - **System reconfiguration**

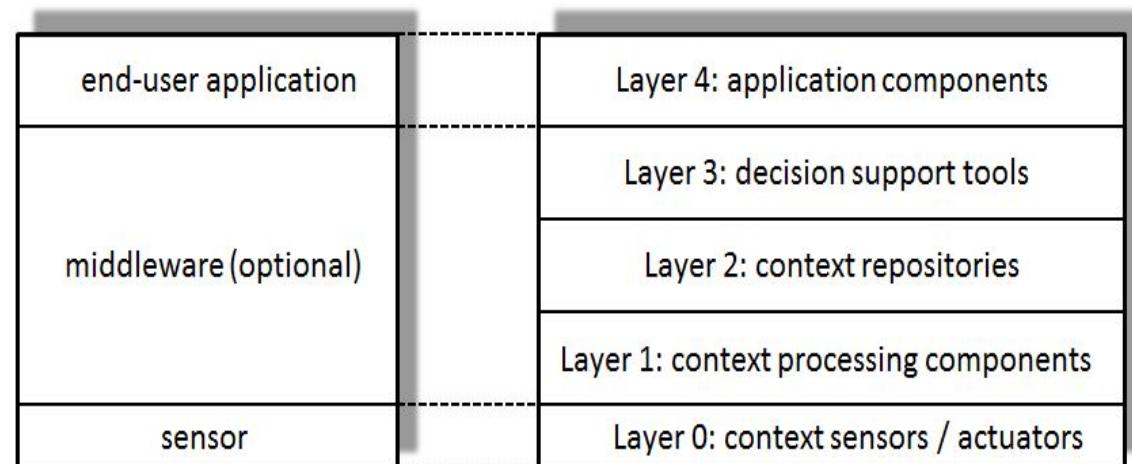
Context-Aware Systems

- These systems can be described as:
 - *Centralized or distributed system that uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task;*
 - *including end-user applications that use context information provided by sensors.*
- From an architectural point of view:
 - end-user applications layer - consumers of context information
 - middleware layer - communication and coordination issues between distributed components
 - sensors layer - producers



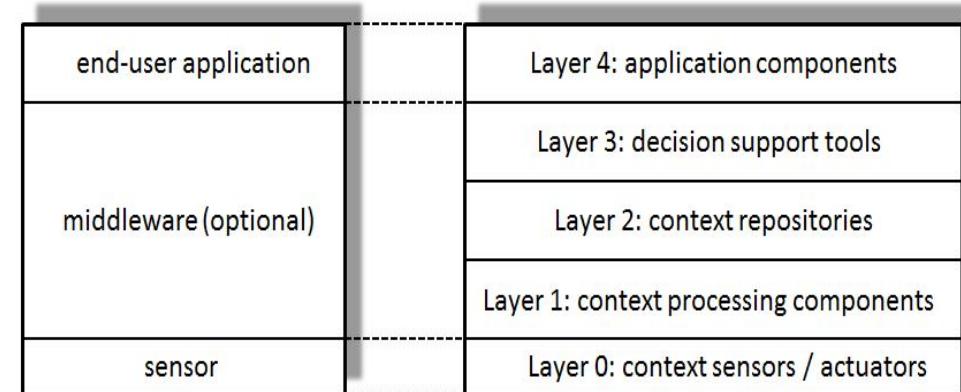
Context-Aware Systems (cont.)

- In simple context-aware systems, the end-user application communicates directly with the sensor:
 - e.g., a mobile phone that automatically switches off GPS when battery is below a certain capacity
- Today, it is widely acknowledged that additional infrastructural components are desirable:
 - to reduce the complexity of distributed context-aware applications,
 - improve maintainability, and
 - promote reuse.



Context-Aware Systems: Typical Stack

- layer 0 – context sensors and actuators that:
 - provide the interface with the environment, either by capturing it (sensors) or by modifying it (actuators);
- layer 1 – components that:
 - (i) assist with processing sensor outputs to produce context information that can be used by applications, and
 - (ii) map update operations on the higher-order information back down to actions on actuators;
- layer 2 - context repositories that:
 - provide persistent storage of context information and advanced query facilities;
- layer 3 – decision support tools that:
 - help applications to select appropriate actions and adaptations based on the available context information;
- layer 4 – application components that:
 - are integrated in client applications using programming toolkits.

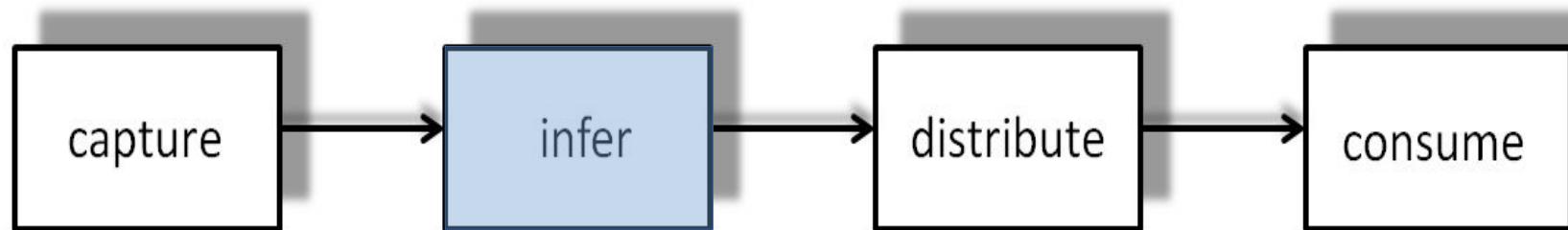
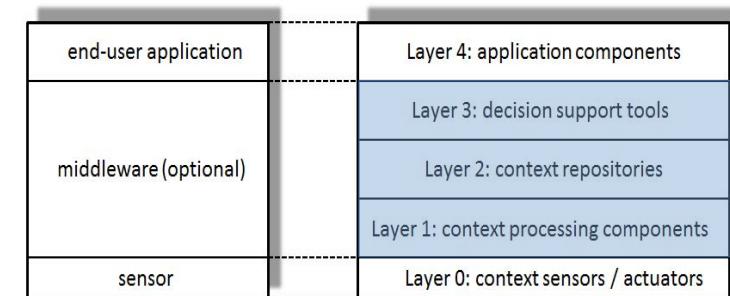


Local vs. Distributed Context-Aware Systems

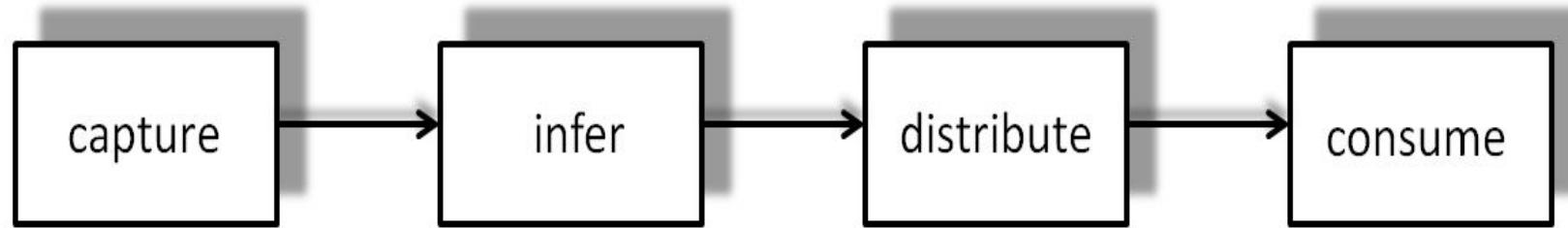
- Local:
 - sensors and applications are **tightly coupled** (usually through a direct physical connection)
 - e.g. a mobile phone application that sets the mode to silent, while its owner is jogging (the accelerometer that provides the “running” context is directly attached to the mobile phone as well as the application that uses that context to activate the silent mode)
- Distributed:
 - do **not have a direct physical connection** between the sensor and the application.
 - thus, it is possible to have multiple applications receiving information from the same sensor.
 - it is also possible that multiple dispersed sensors produce information to be consumed by a single application
 - e.g. a mobile phone may broadcast to a group of friends that its owner is currently walking, to decrease the probability of incoming calls during that period.
 - e.g. multiple thermometers in a home send periodic updates to a central thermostat.
 - in this case, the thermometers are not tightly coupled to the recipient of its information.

Context Processing: Taxonomy

- Clearly categorizes the architectural options available to the application developer, explaining the advantages and disadvantages of each approach;
- Analyzes the problems that are specific to distributed context-aware systems, whose (distributed) components have to communicate with each other, and how that affects availability and scalability of such systems;
- Four layers:



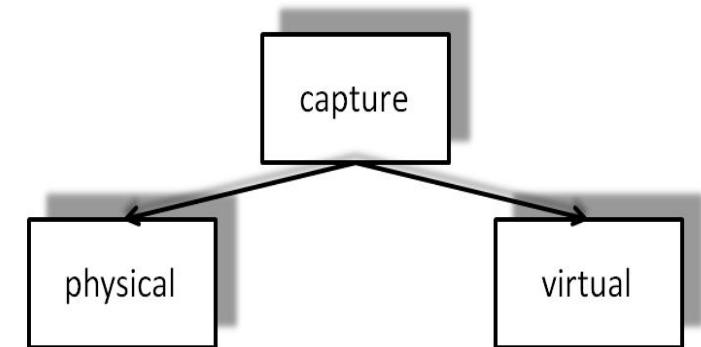
Context Processing: Taxonomy (cont.)



- Context data is **captured** from the environment using sensors
- A **context inference step** is needed to obtain higher-level aggregated data:
 - e.g. GPS device captures geographical coordinates from which a place (city, building, etc.) is inferred.
 - this step is also known as *feature extraction*.
- **Distribution of context data** among its components in an efficient and scalable manner
- Client applications **consume** this information in order to provide relevant services to their users.

Capture

- The Capture layer is responsible for acquiring context data from the environment using sensors.
- The word “sensor” not only refers to sensing hardware but also to every data source which may provide usable context information:
 - *physical*, and *virtual*
- **Physical sensors** are hardware sensors capable of capturing physical data such as light, audio, motion, and location.
 - location is, by far, the most researched type of physical sensor
- **Virtual sensors** acquire context data from software applications, operating systems, and networks.
 - detecting new appointments on an electronic calendar and watching the file system for changes are examples of virtual sensors



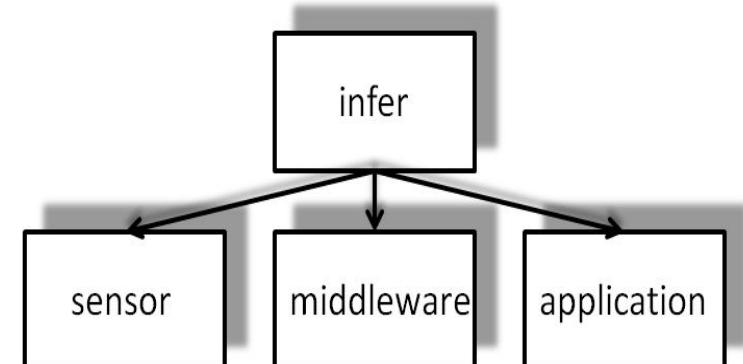
Examples of Sensors and Types of Context

- Examples of *location, identity, activity, and time sensors*.

	Physical sensors	Virtual sensors
Location	<i>Outdoor:</i> global positioning system (GPS), global system for mobile communications (GSM); <i>indoor:</i> Bluetooth, 802.11 cells	Networked calendar system, travel-booking system, user's login on location-aware computer, IP subnet
Identity	<i>Based on something you are:</i> fingerprint reader, retina scanner, microphone; <i>based on something you have:</i> smart card reader, RFId	Various authentication schemes at the operating system or application level
Activity	Mercury switch, accelerometer, motion detector, thermometer, UV sensors, camera	Keyboard or mouse activity, application usage
Time	Clock	Operating system timer

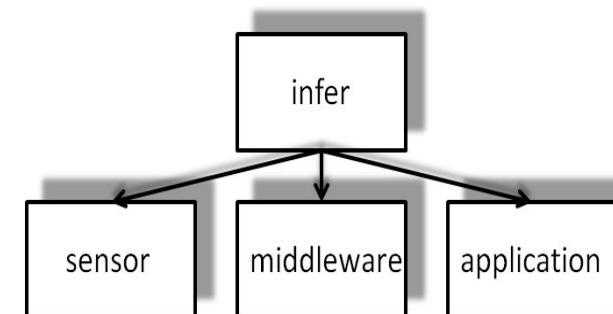
Infer

- It is responsible for reasoning and interpreting raw context information provided by sensors on the Capture Layer:
 - e.g. end-user applications do not need to know the exact latitude and longitude of an entity, being much more interested in knowing the place (city, street, etc.) in which the entity is located
- The *Context-Inference* layer, is also known as the *Preprocessing Layer*
- Context inference usually involves some kind of reasoning:
 - a transformation is needed to reach a higher level of abstraction
 - it is normally associated with classification techniques
 - e.g. inferring the user activity (walking, running) from an accelerometer
- Where should the Inference layer be placed?



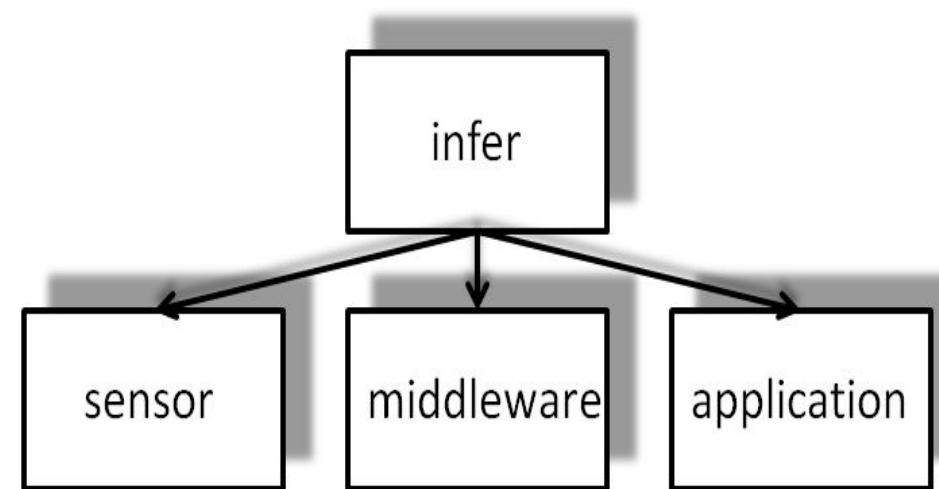
Infer (cont.)

- The type of information inferred from sensorial data is **specific** to each application:
 - e.g. taking the data provided by an **accelerometer**, an application might use that information to know whether a user is **walking** or running, while another may be more interested in detecting climbing stairs or rising in elevators.
 - it may make sense to move the inferring task to the application because of the specific semantic needs it tries to accomplish.
 - transforming raw sensorial data into high-level context information can be a resource-demanding task
 - (unsuitable for applications on constrained devices such as mobile phones)
 - some systems use a middleware component running in a machine with higher capability



Infer (cont.)

- The three locations for inference are intimately related to the following properties of a context-aware system:
 - Network bandwidth consumption
 - Complexity (CPU/memory consumption)
 - Reusability
 - Personalization



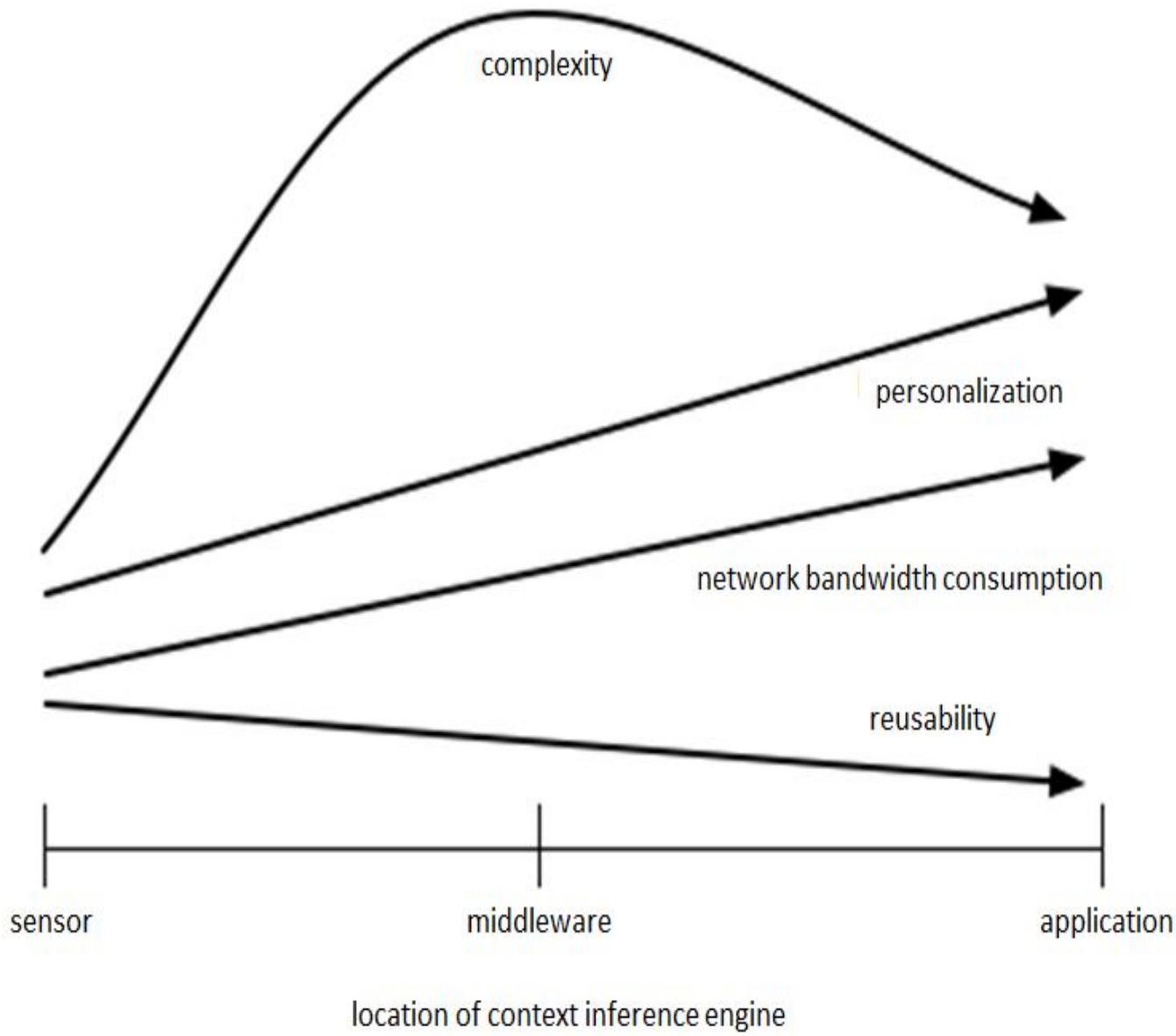
Infer (cont.)

- Network bandwidth consumption
 - context inference transforms fine grained sensorial information into coarse-grained high-level data
 - it reduces the amount of information needed to represent a context message
 - moving the inference layer closer to the sensor results in less network bandwidth consumption
- Complexity (CPU/memory consumption)
 - context inference mechanisms can lead to high CPU and RAM consumption
 - even if the device is capable of computing the information, it can lead to excessive and unsustainable battery consumption
 - thus, the developer has to deploy the inference engine in a resourceful machine (e.g., a server)
 - most physical sensors are attached to low-capability devices (to increase mobility) and do not provide context-inference mechanisms.
 - virtual sensors often run on servers or desktop computers (where they can access virtual context information from databases, shared calendars, etc.), so they are able to provide higher-level context information than their physical counterparts.

Infer (cont.)

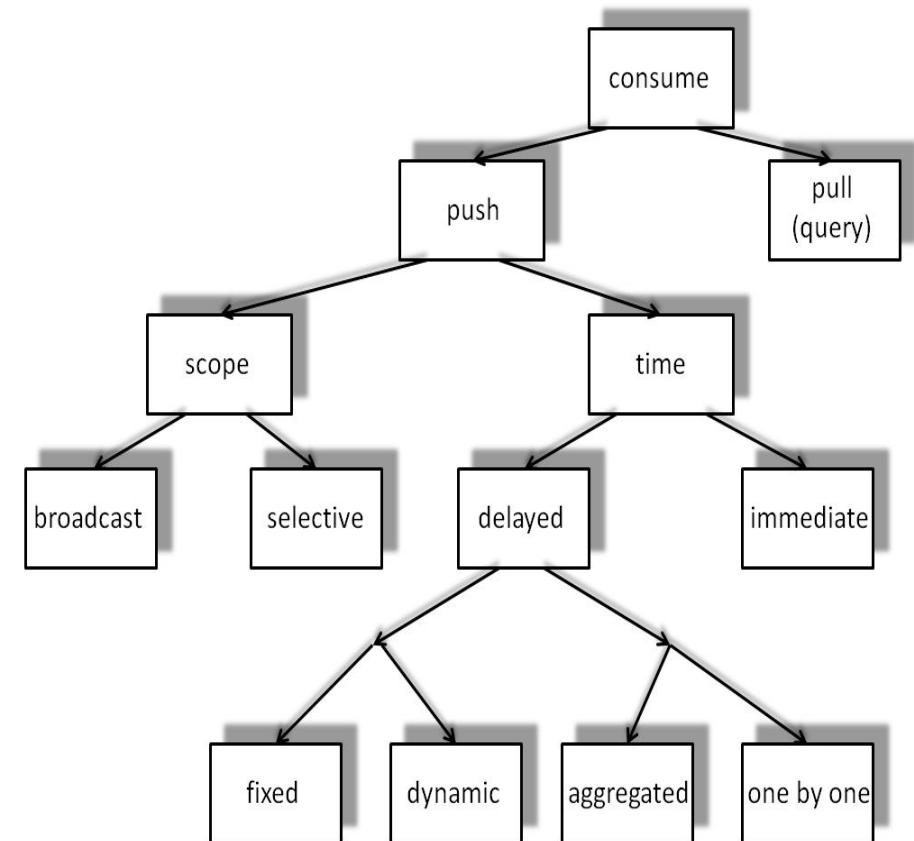
- Reusability
 - makes sense for context types as location, where the output is sufficiently *standard* to be used by multiple applications
 - e.g. inferring the street name from geographical GPS coordinates is a recurring requirement in location-aware systems and does not make sense to (re)implement in every end-user application
 - moving the inference layer closer to the sensor increases reusability
- Personalization
 - ability to personalize the inference engine to better suit individual needs
 - e.g. a phone application that allows its users to associate a certain movement (like drawing an imaginary circle with the phone) to some meaning or activity (e.g., going to lunch)
 - mediation of ambiguity - context inference includes dealing with ambiguous data and explicit user mediation is needed (thus adapting the inference mechanism to suit individual needs)
 - Can be circumvented by API with varied levels of abstraction.

Infer (cont.)



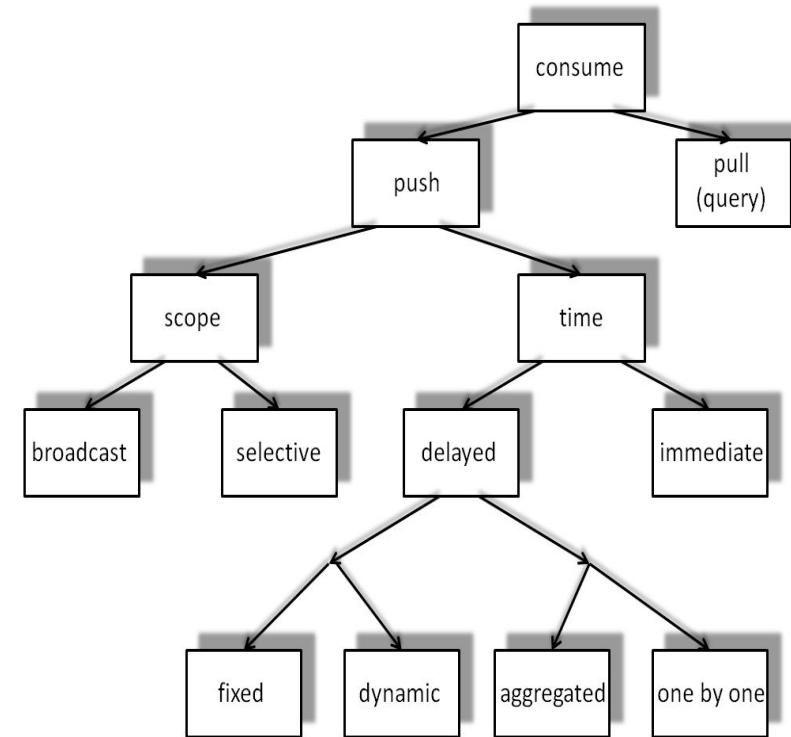
Consume

- After context information has been:
 - captured from sensors,
 - inferred into higher-level data,
 - distributed through the network,
 - it will be consumed by client applications.



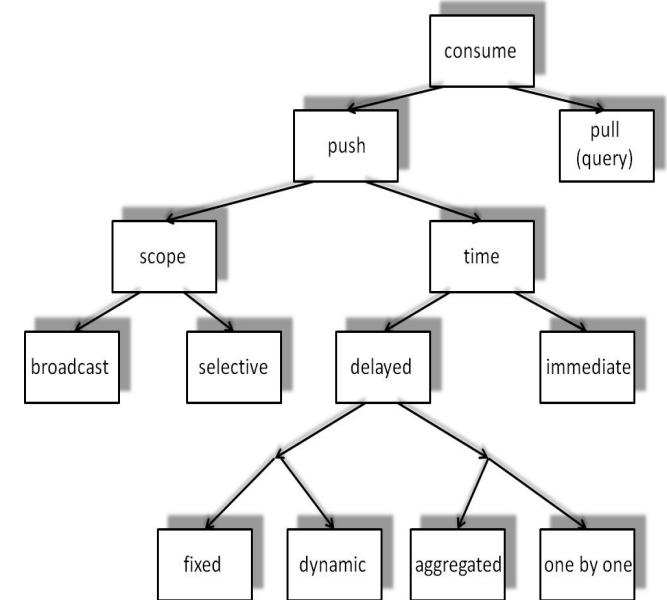
Consume (cont.)

- Pull-based (or query) systems:
 - *pull-based systems* - the consumer (client application) queries the component that has the information (sensor or middleware) for updates
 - this can be done manually (e.g., the user clicks the “refresh” button) or periodically (e.g., checking for updates every 5 min)
- Push-based systems:
 - the component that has the information is responsible for delivering it to interested client applications.
- The main distinction between these approaches lies on who initiates the communication:
 - if a client application initiates the communication, the system is pull-based;
 - otherwise it is push-based.



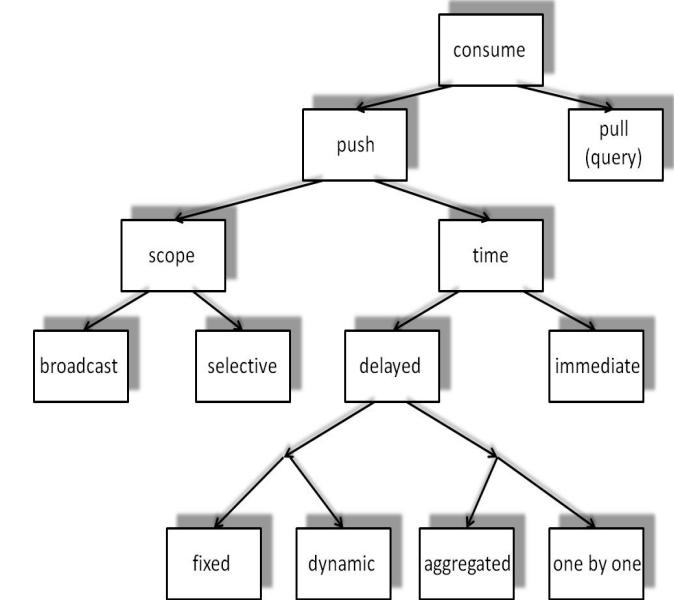
Consume (cont.)

- Pull-based mechanisms are simple to implement:
 - the client application only has to query a certain component with a certain periodicity.
 - the application can simply delegate the responsibility of refreshing the information to the user.
 - the polling interval can change according to the application current needs.
 - e.g., an application running on a mobile phone can increase the polling interval to save battery life, when the battery power falls below a certain threshold.



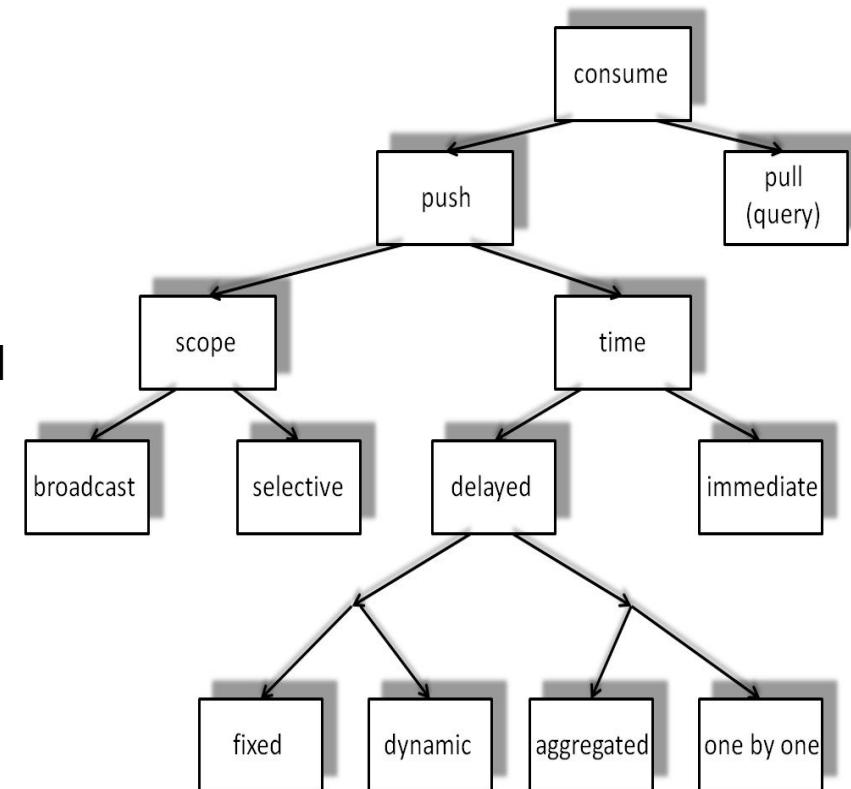
Consume (cont.)

- However, this mechanism suffers from two problems that can be critical:
 - context information reaches the application with a delay (may equal the polling interval)
 - if polling interval is 5 min, **the application may receive context information from 5 min ago**, compromising its usefulness.
 - the application can minimize this disadvantage by **reducing the polling interval**, but then, the second problem may start **occurring—most queries will be unnecessary** since there is no new context information available
 - since the application does not know when there will be updates, it has no alternative as to keep asking until there is new information, leading to **wasted network bandwidth and CPU consumption**



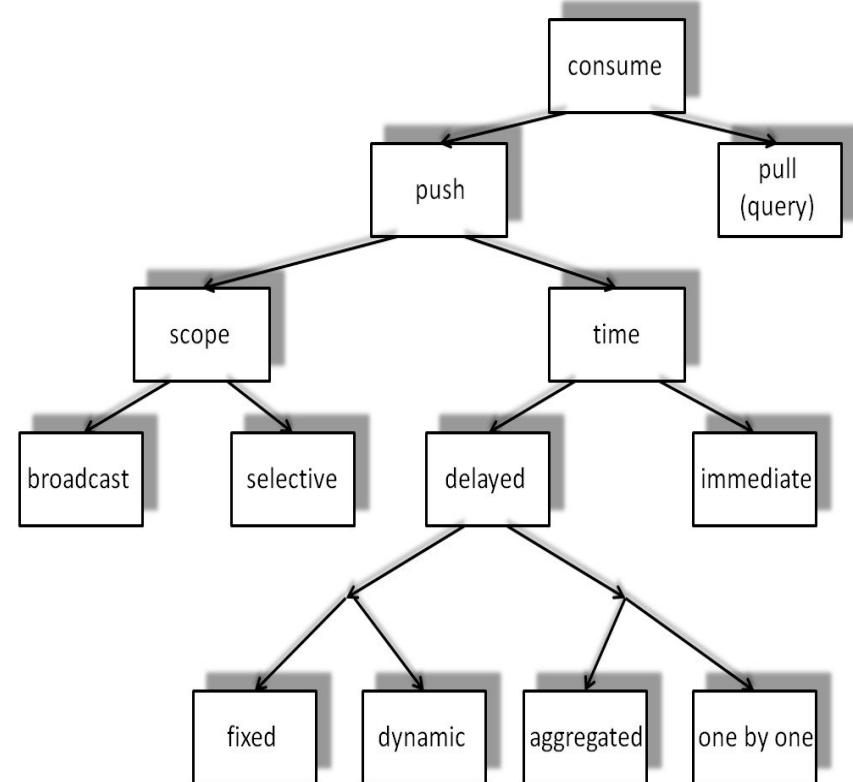
Consume (cont.)

- Push-based mechanisms are **more complex** than pull-based mechanisms.
- The component that has the information is also responsible for delivering it:
 - this component has to **know how to reach every possible consumer** that is interested in that information.
- Usually, a permanent connection with all the consumers is necessary.
- If the system has a large number of consumers, its **performance may degrade**.
- Furthermore, these systems have **poor scalability** as every new consumer degrades the performance.
- There are two measures to overcome the scalability problem: *reduce scope* and/or *relax delivery time*.



Consume (cont.)

- Users may be interested only in a **subset** of the system's available context information:
 - e.g. an application that enhances the traditional phone contact list with contextual information (such as location and availability) is **only interested in context updates for the people in its contact list.**
 - thus, if person A has person B in his list of contacts, A is notified if B moves from one location to another, but other people in the system without B in their contact list are not notified.

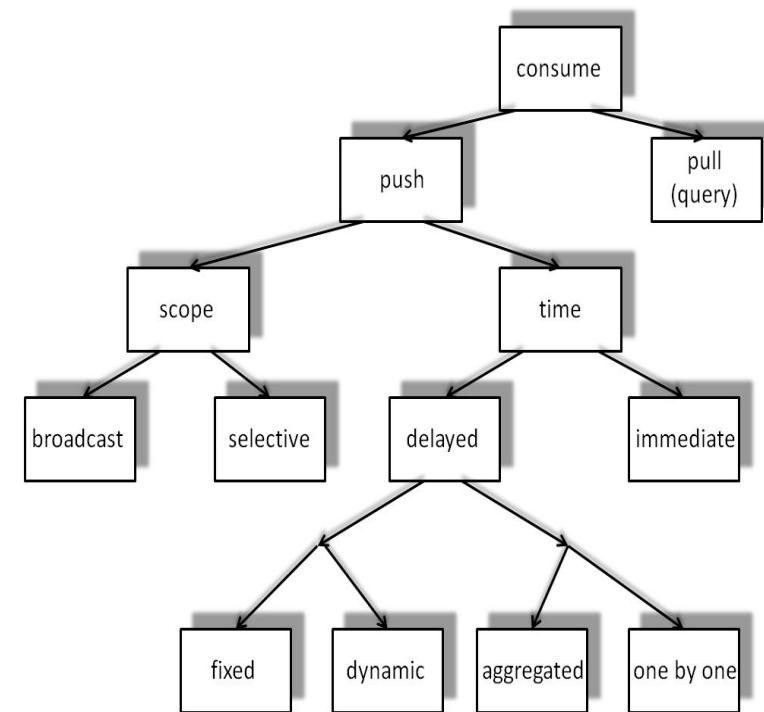


Consume (cont.)

Selective scope systems propagate only a **subset of the available context information** based on user's selection.

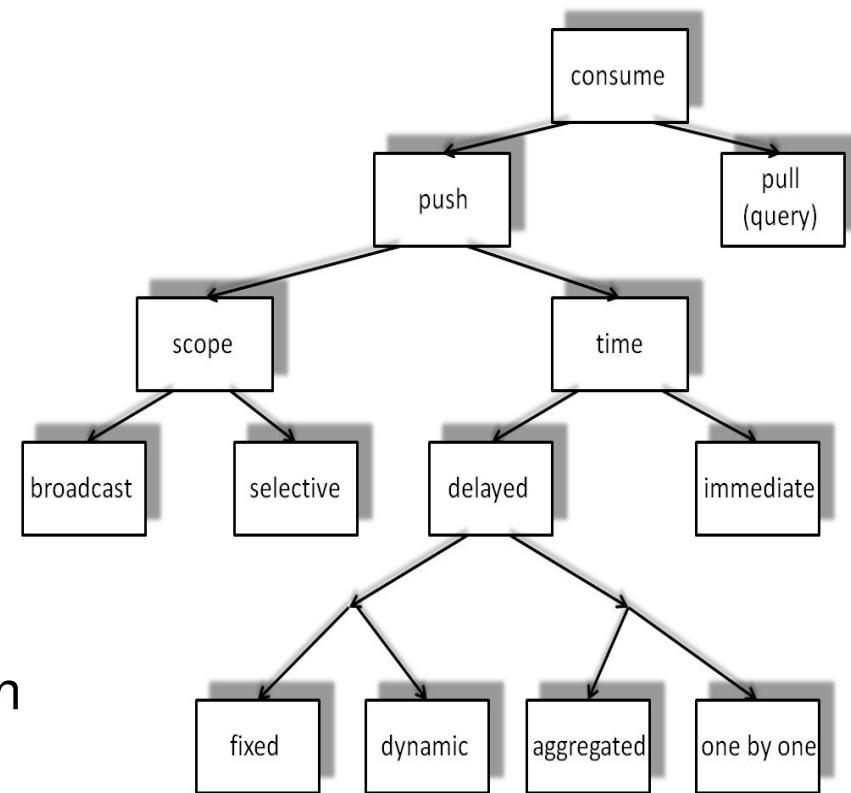
Most context-aware systems propagate all the available context information, thus falling into the broadcast scope category

- Using scope is an effective technique to reduce the number of pushed messages in the system:
 - reducing the required **outbound network bandwidth** on the component that has the information and improving overall scalability.
 - *quenching* - a mechanism that allows sensors to know whether client applications are interested in their information and only sending information in that case



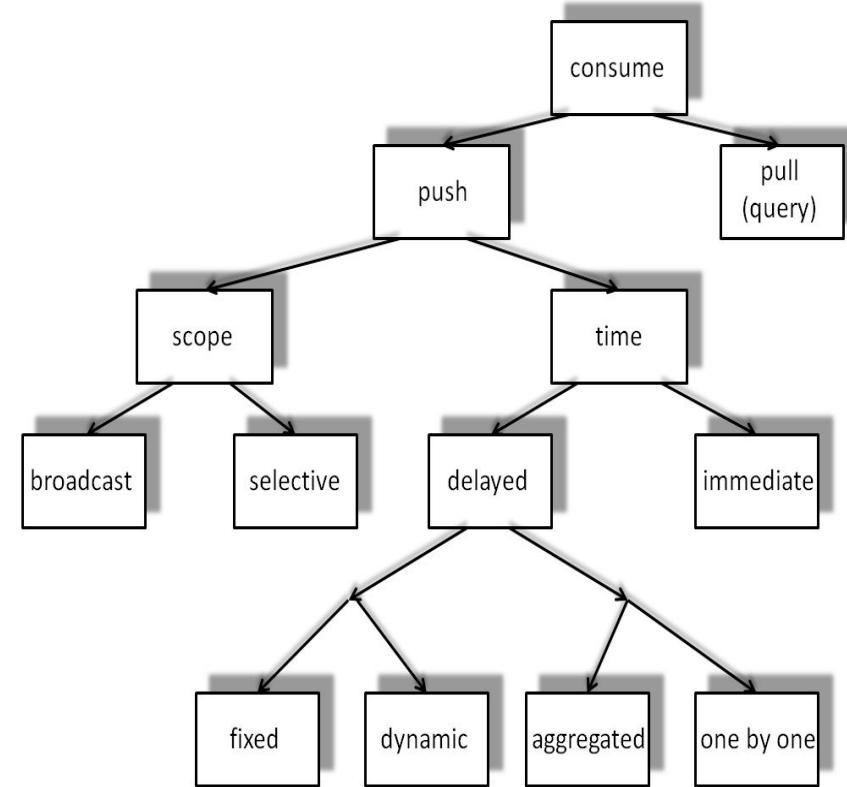
Consume (cont.)

- Time:
 - to improve push-based systems scalability,
 - assuming that certain context information **does not need to be immediately delivered**
- Users **tolerate some lag** as long as the information does not require urgent attention.
 - e.g., a context-aware system that shows friends near me is not required to be continuously up to date.
 - on the other hand, an application that shows the products around a user in a supermarket loses its usefulness if context information is not propagated as soon as possible
- Some of these issues have been studied in the context of *optimistic replication algorithms*:
 - increase availability and scalability of distributed data sharing systems by allowing replica contents to diverge in the short term



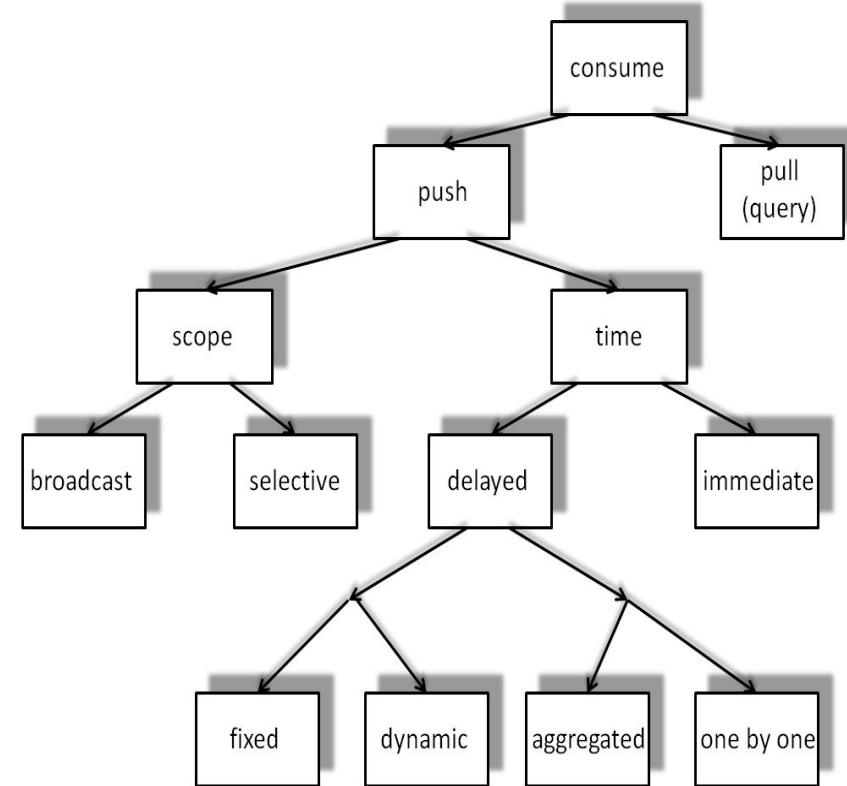
Consume (cont.)

- Fixed vs. dynamic:
 - must decide when to actually push the updates.
 - decision can be made based on a **fixed criteria such as time period** (push the update every x seconds) or **amount of retained information** (push the update when there is more than x Kbytes of context information)
 - obviously, both criteria can be used: e.g. to improve the scalability of a context-aware message application



Consume (cont.)

- Aggregated vs. one-by-one:
 - since there is a delay in context propagation, information must be retained and may start accumulating
 - when the system decides to push the update, there may be a considerable number of messages to transmit
 - simple approach is to **transmit the messages one-by-one**, as would be the case if there was not any delay (this can be extremely inefficient)
 - alternative solution is to **aggregate all delayed information in just one** (e.g. GPS locations during a certain amount of time)
 - even when all the retained messages are relevant, they can still be **aggregated in one big message**, achieving **higher levels of compression** and much **less round-trips** in the connection path



Consume (cont.)

- Note that:
 - in context-aware related literature, the term “aggregation” is often associated with the combination of information from different sensors, along with conflict resolution.
 - do not confuse both uses of the same term!
- More to read:
 - See several examples of context aware systems in the bibliography (Distributed Context-Aware Systems.

Paulo Ferreira, Pedro Alves. SpringerBriefs in Computer Science, 2014)

Example (1/2)

Estimote Smart Beacons - welcome to the contextual computing era-
<https://www.youtube.com/watch?v=SrsHBjzt2E8>

Example (2/2)

Estimote Sticker Beacons - Introducing Nearables -

<https://www.youtube.com/watch?v=JrRS8qRYXCQ>

Examples of Context-Aware Applications and Sensors

	Description	Sensors used
GUIDE [25]	Information for city visitors	WiFi (location)
Welbourne [97]	Mode of transit: walking, running, riding a vehicle	Clock, GSM/WiFi (location), accelerometer
ContextContacts [62]	Enhanced phone's contact list	GSM, phone activity, Bluetooth (nearby environment)
UbiqMuseum [15]	Assists museum visitors	Bluetooth (location)
AwareMedia [9]	Support coordination at an operation ward	Bluetooth (location), video camera, shared calendar
Stiefmeier [91]	Help workers perform critical and complex assembly tasks in a car production environment	Body-worn, car-mounted, and tool-mounted accelerometers
BikeNet [36]	Cyclist experience mapping	Magnetometer, inclinometer, speedometer, microphone, GPS, GSR stress monitor, CO ₂ meter
CenceMe [59]	Social activities (dancing, lunching)	Microphone, GPS, camera, Bluetooth (nearby environment), accelerometer
SoundSense [57]	Music events' sharing	Microphone, camera, GPS
Upcase [82]	Daily activities (working, driving, sleeping, resting, walking outside)	Luminosity, microphone, temperature, accelerometer

Mobile and Ubiquitous Computing
2022/2023
MEIC/METI

**Energy Issues in Mobile and
Ubiquitous Computing**

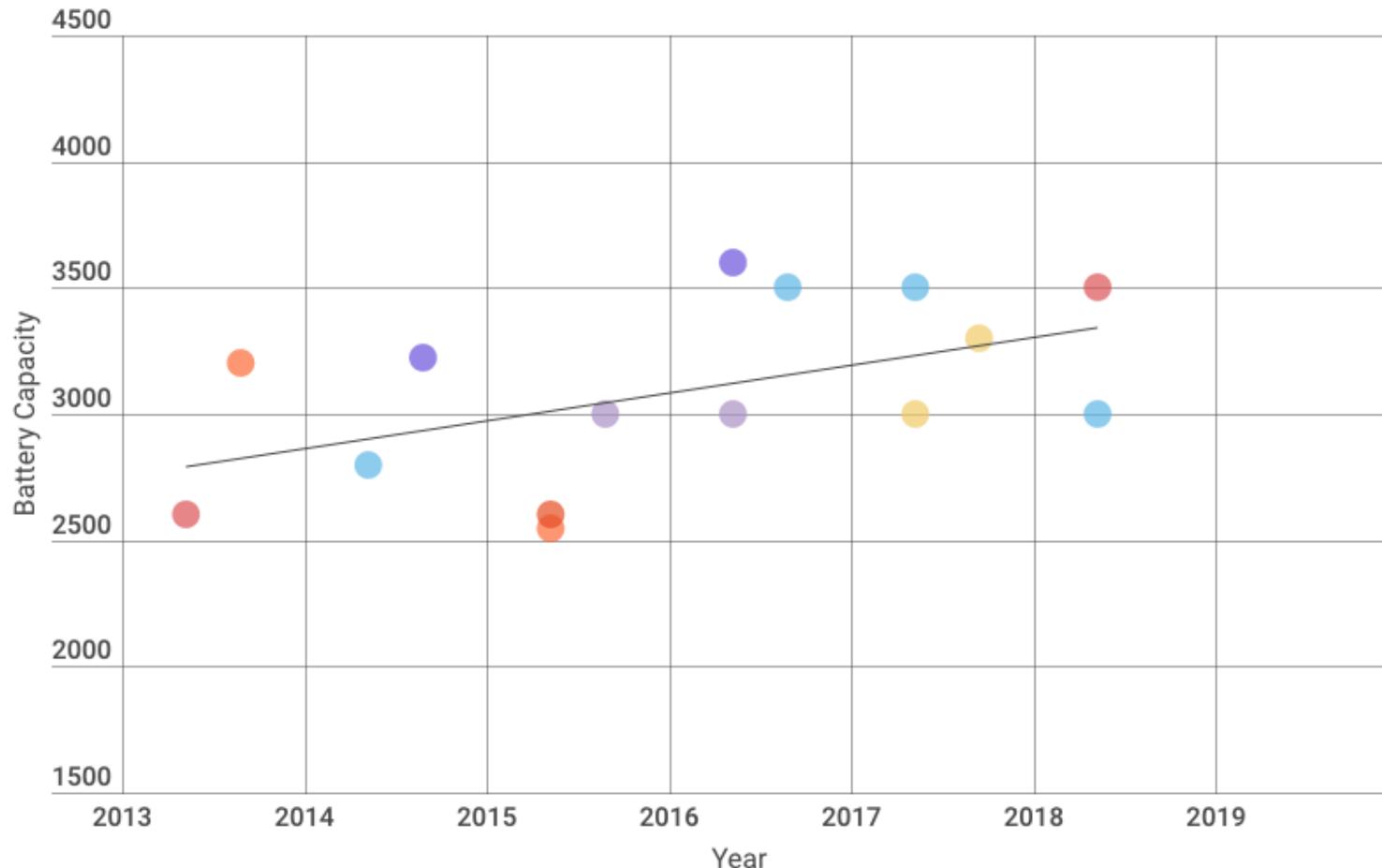
Context

- What is the problem? Matching:
 - Batteries
 - Devices
 - Applications
 - User:
 - App usage
 - Charging behaviour

Device Size, Battery Size

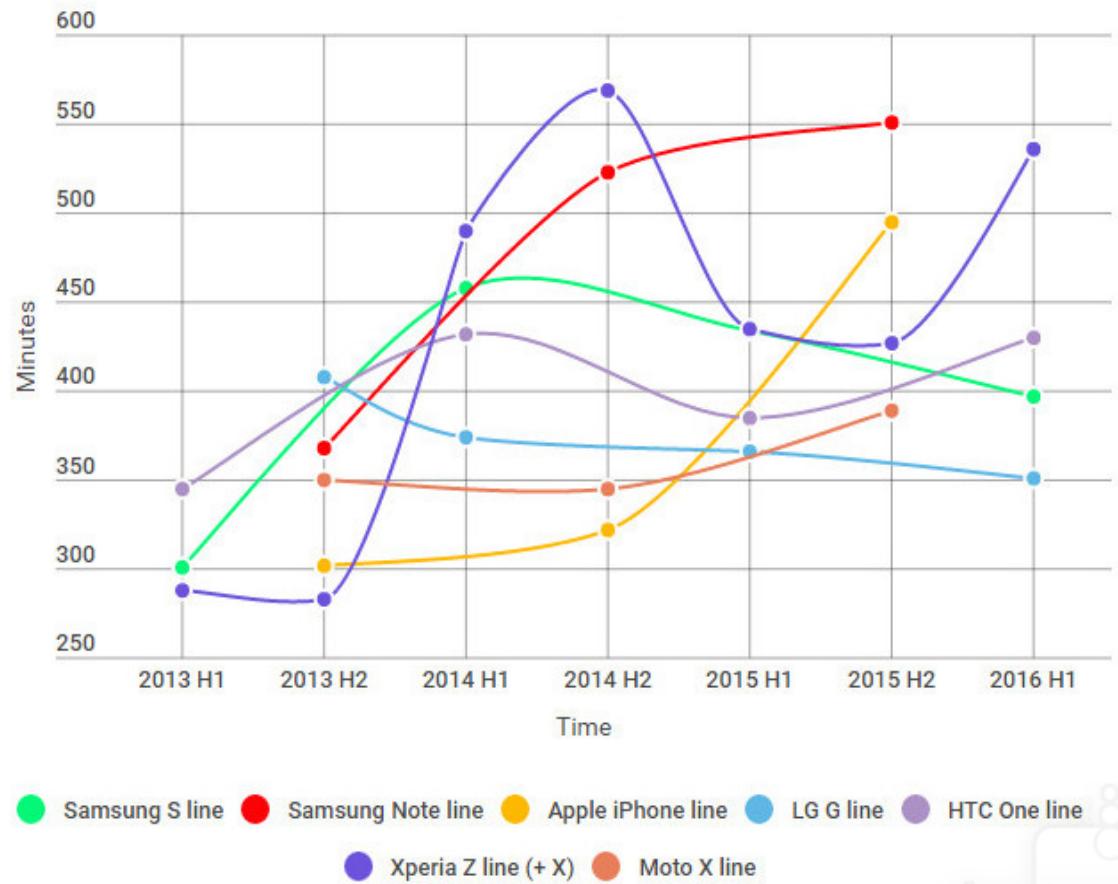
- Over the years mobile devices have become more **powerful**, more fully **featured**, and more essential.
- Resulting:
 - More sensors
 - Faster CPUs
 - Batteries than have grown but not kept up.

Progress in Batteries is Slow



- Example: Capacity on Samsung phones

Progress in Batteries is Slow



- Battery life numbers are mixed too.

Progress in Batteries is Slow

	mAh	Resolution
iPhone 7	1,960	750 x 1334
iPhone 8	1,821	750 x 1334
iPhone 7 plus	2,900	1080 x 1920
iPhone 8 plus	2,691	1080 x 1920
iPhone X	2,716	1125 x 2436
Pixel 2	2,700	1080 x 1920
Pixel	2,770	1080 x 1920
Pixel 2XL	3,520	1440 x 2880
Pixel XL	3,450	1440 x 2560
LG V30	3,300	1440 x 2880
LG V20	3,200	1440 x 2560
Galaxy Note 8	3,300	1440 x 2960
Galaxy Note 7	3,500	1440 x 2560

- Battery size and screens go hand in hand: screens create **volume (+)** and **power demand (-)**.

Where is Power spent?

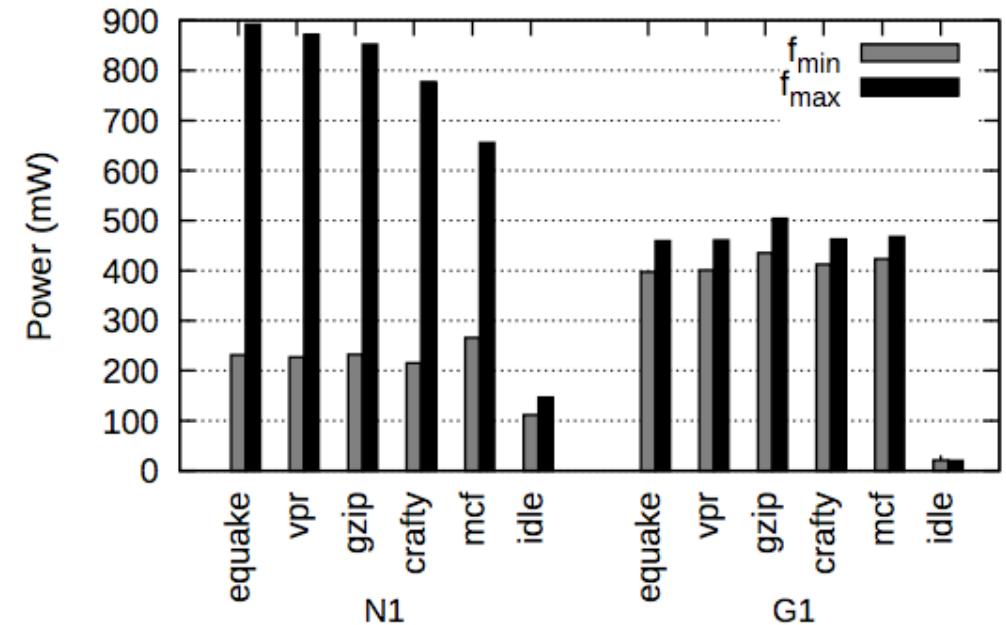
- Mobile devices are often power constrained.
- E.g. most smartphones can easily exhaust their batteries.
- Understanding power consumption in mobile devices is important to make choices.

CPU

- Highly variable as a power drain.

- Throttling:

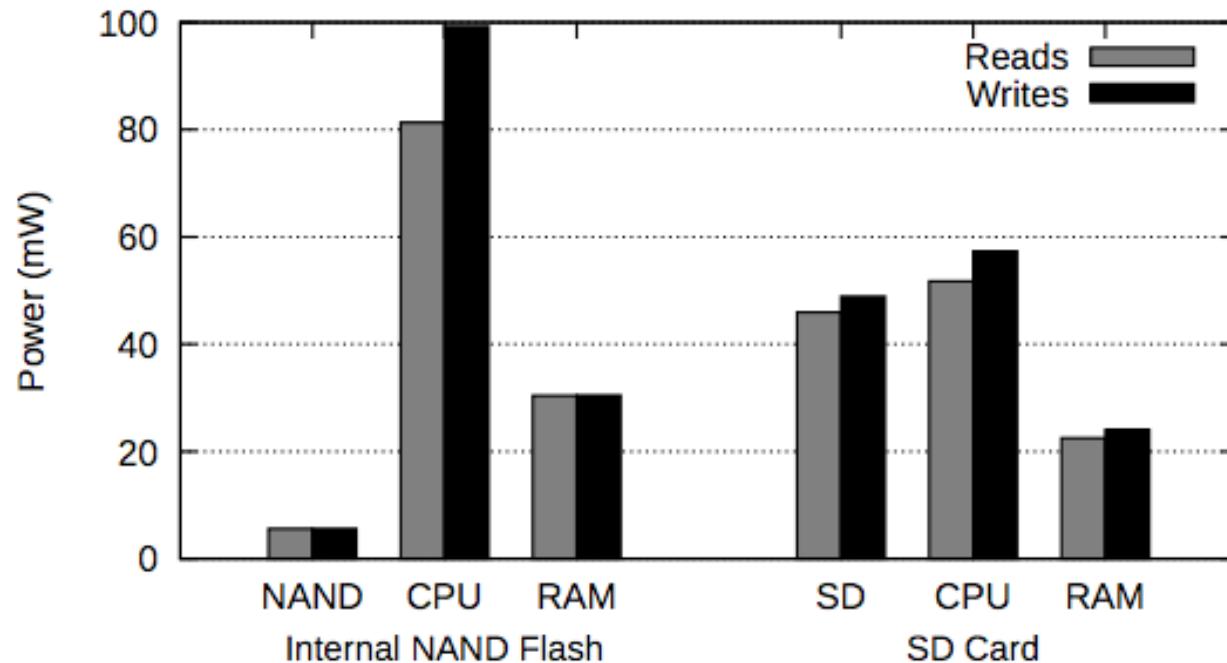
- Doesn't always work.
 - Can be annoying.
 - Is rarely user controlled.



- Adaptive fidelity can be successful both for CPU and network card power drain reduction.

Storage

- Power costs in access to persistent storage is small
- Circa 20% RAM access with the remaining 80% split between CPU and persistent memory.



Network

- Networking can be a significant source of power consumption, especially because it also triggers CPU activity.
- GSM module: ~800 mW
- WiFi module: ~700 mW
- However, WiFi consumes less power for the same bandwidth.

Other Peripherals

- Screen: Display power consumption in common smartphones is around 6-8 hours at full brightness (e.g. 1000 mW for 10000 mWh@3.7V battery)
- Other common sensors: power varies but tends to be smaller.
- GPS: 150 mW

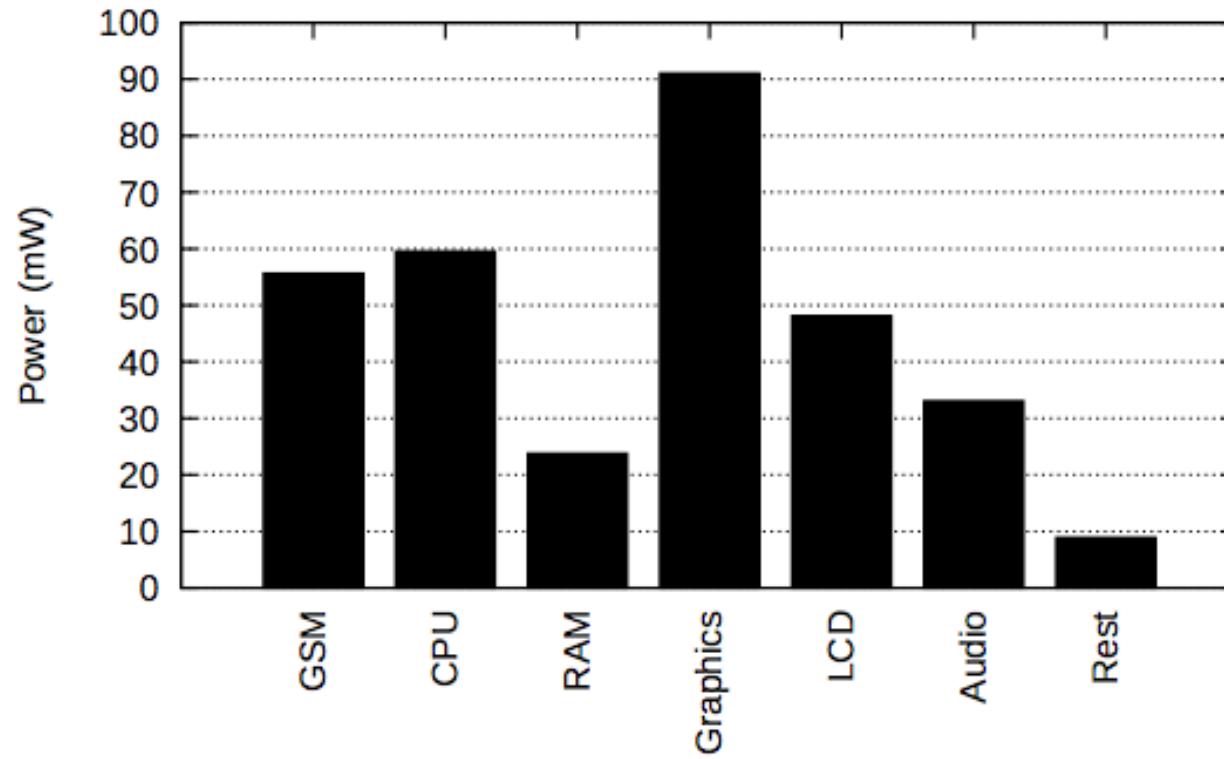
Optimizing Location Sensing

- Mobile applications are more and more location-aware.
- The need for constant location and constant GPS use can be minimized by:
 - **Combining multiple sensors** to reduce the energy consumption while minimising the error, e.g. rare GPS samples + accelerometer.
 - Rely on **probabilistic models** of users' location to infer future locations. (CPU mostly use less power than radio...)
 - Use **heuristics** to adapt the sampling rate, e.g. sampling more when the user is moving or cancelling sampling in areas known not to have GPS coverage.

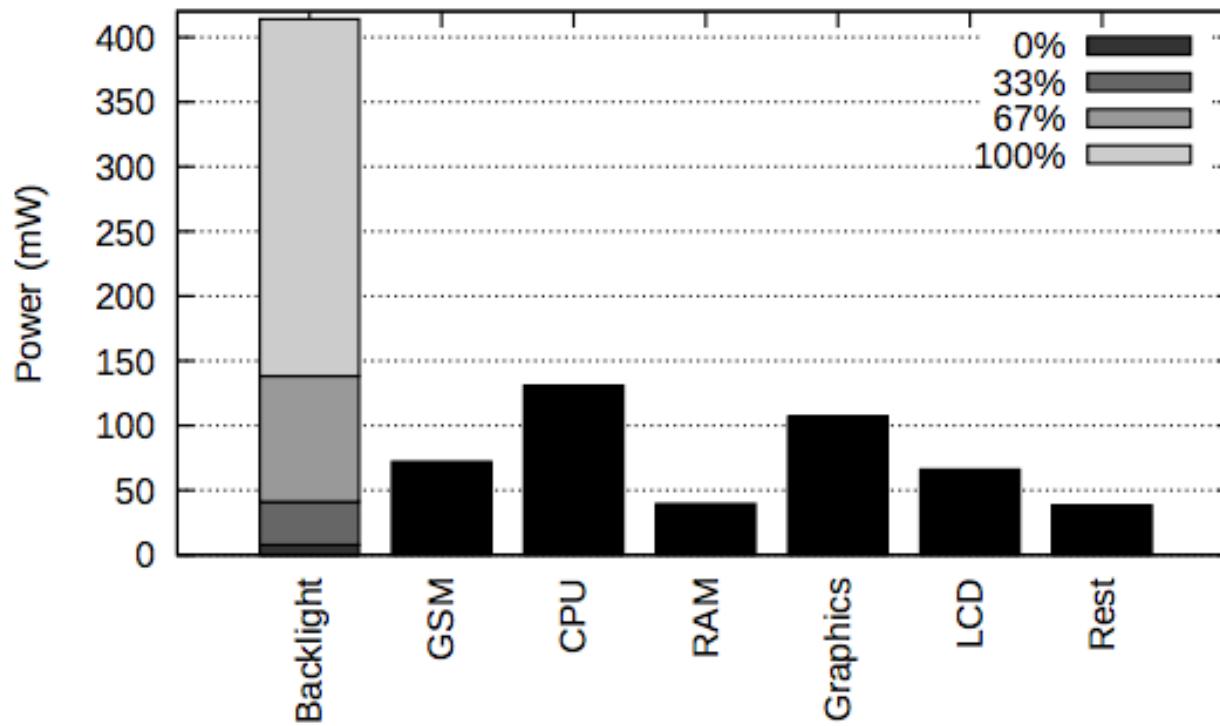
Sources of Power Consumption

- How do they aggregate?
- What are typical application profiles?

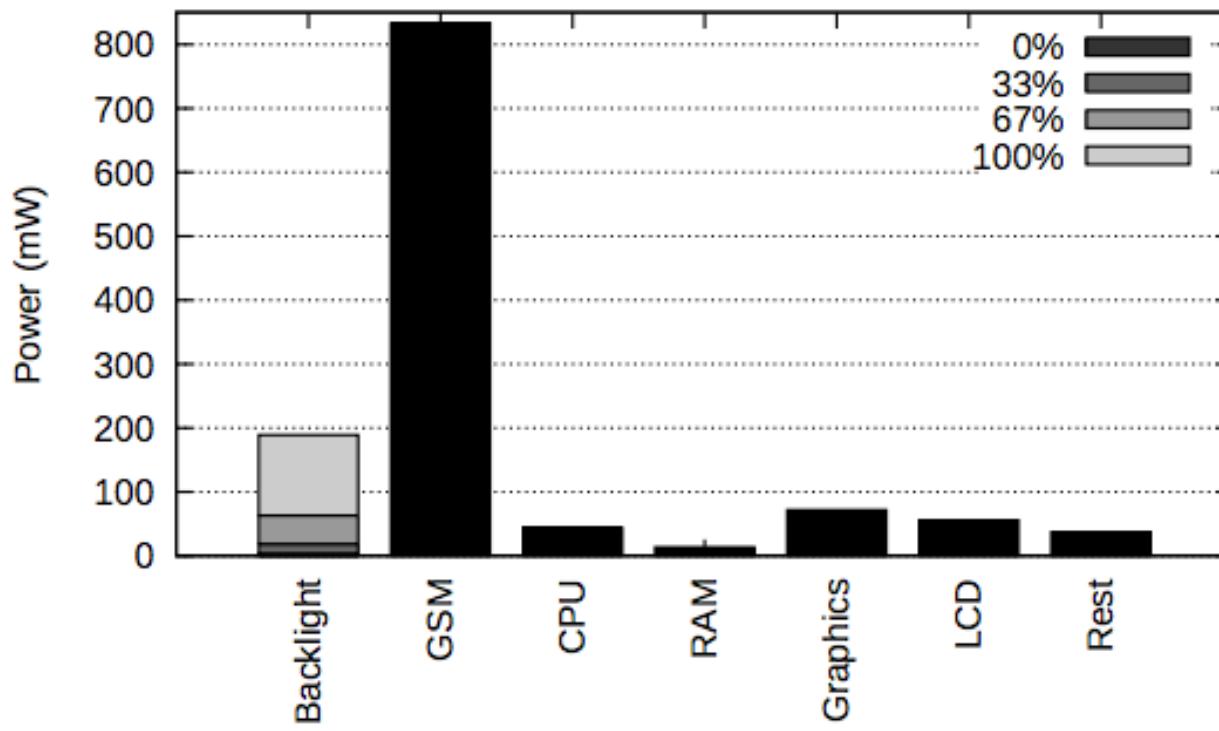
Power profile examples: audio playback



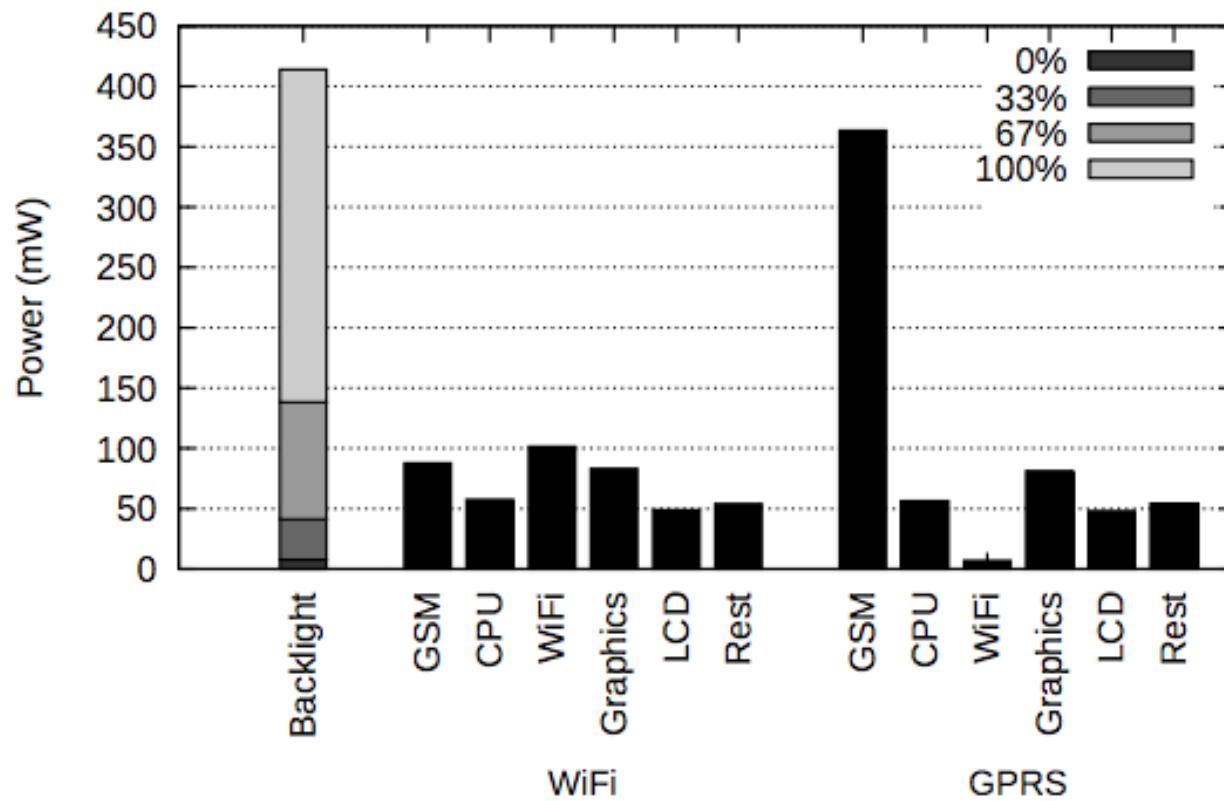
Power profile examples: video playback



Power profile examples: phone call



Power profile examples: email



How to adjust for power optimization

- Efficient power management in mobile platforms is a complex and challenging research problem:
 - due to the multitude of possible hardware configuration options and power states.
 - and the interdependencies between computing, sensing and networking resources caused by applications.

Understanding Users

- **Energy aware operating systems:**
- Who should be responsible for energy management: Applications or operating system? Probably both.
- At the OS level, the main idea is to reduce energy consumption by unifying resource and energy management and by using collaboration between applications and operating system.
- A key part of energy-efficient resources and energy management is having a good understanding of how resources are demanded by users and applications in the system.

OS approach:

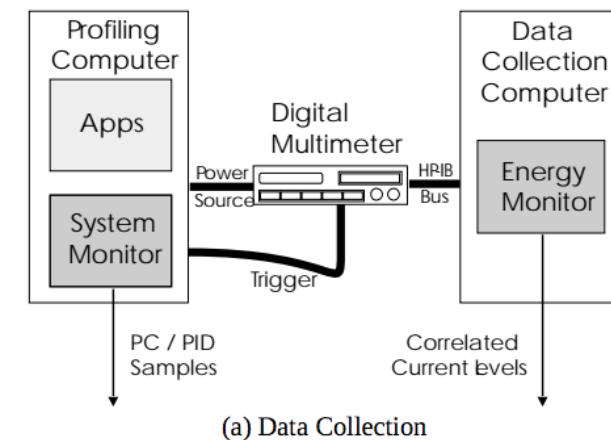
- Several systems adapt operating systems to:
- Disable unnecessary hardware devices.
- Provide fairness between applications using credit systems.
- Use power forecasting to do scheduling.

OS approach: Cinder

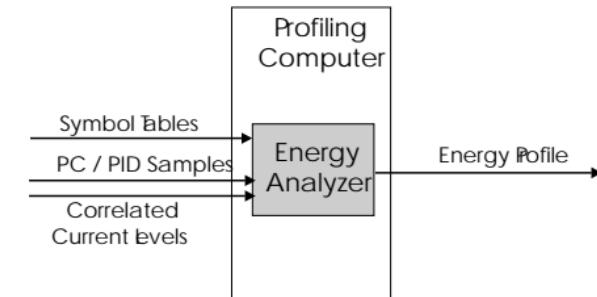
- Cinder [9] is a mobile OS designed on top of the HiStar exokernel that exploits device-level accounting and power modelling.
- Cinder allocates energy to applications using two abstractions called **reserves** (batteries) and **taps** to form a graph of resource consumption.
- When an application consumes a resource, the Cinder kernel reduces the values of the corresponding reserve and its scheduler only allows threads to run if they have enough reserves to run.
- The rate at which the reserves are being consumed is controlled by taps which are defined as special-purpose threads whose only role is to transfer energy between reserves (they support constant and proportional rates).
- Once an application has consumed all its reserves, the kernel prevents its threads performing more actions.

Power Modelling: Powerscope

- Mapping Energy to code procedures
- Powerscope: identifies applications behaving as energy sinks.
- Requires an external power meter and a second computer to reduce any possible interference at the profile stage.
- Uses statistical sampling to collect traces.
- This approach is not scalable since it requires repeating the off-line training for every single hardware configuration and machine.



(a) Data Collection



User Modelling

- User based energy management requires:
- 1) understanding the charging-discharging cycle the users impose on phones.
- 2) Knowing how users interacts with their handsets (and particular resources), and therefore how they demand energy.

Charging Prediction

- There are 3 types of users:
 - opportunistic chargers, light-consumers and night-time chargers.
- EET, a predictive model:
 - Tries to determine the subset of high-level energy characteristics that differentiate users.
 - It is possible to predict the energy level on a mobile handset within 7% error within an hour and within 28% error within 24 hours.

User Behaviour

- Users do typical 20-100 phone use sessions of 10 to 250 s. Sessions are longer at night than during the day.
- User app use is strongly related to location and time of day.
- App category and usage duration is correlated for individual users.

Conclusions

- Smartphones continue to become more powerful.
- The evolution of batteries has been slower.
- Power consumption is mostly dominated by the screen and networking.
- Power usage and charging patterns can be learned and harmonized.
- Any application can contribute to power saving. **Be frugal as a developer!**

**Mobile and Ubiquitous
Computing 2023-24
MEIC/METI - Alameda & Tagus**

Adaptability

Contents

- Definition of adaptability
- Variable context/resources
- Trading-off adaptability and invisibility
- Measuring the effectiveness of adaptability mechanisms
- Adaption layer (operating system, middleware, application)
- Architecture for adaptability
- Adaptability policies

Definition of Adaptability

- **Adaptability** (**Latin**: *adaptō* "fit to, adjust")
 - is a feature of a system or of a process
 - quality of being adaptable; a quality that renders adaptable
 - variability in respect to, or under the influence of, external conditions
- A computer system is said to have the capability of being **adaptable**:
 - if it changes **automatically** to suit **variable** working conditions,
 - while offering the best **quality of service** possible,
 - that is acceptable by the user.

Variable Context/Resources (1/2)

- Due to their intrinsic nature, execution environments in mobile computing suffer from **great and diverse variations** during applications execution.
- These variations can be:
 - **qualitative** (e.g., network connection or disconnection, specific devices such as printers in the device neighborhood, consistency and security constrains), or
 - **quantitative** aspects (e.g., amount of usable bandwidth, memory available).
- Applications should be able to **automatically deal with this variability**
- Applications programmers should **not be forced to account for every possible scenario** in their coding as this would be:
 - inefficient, error-prone, and limited to situations accounted for *a priori*

Variable Context/Resources (2/2)

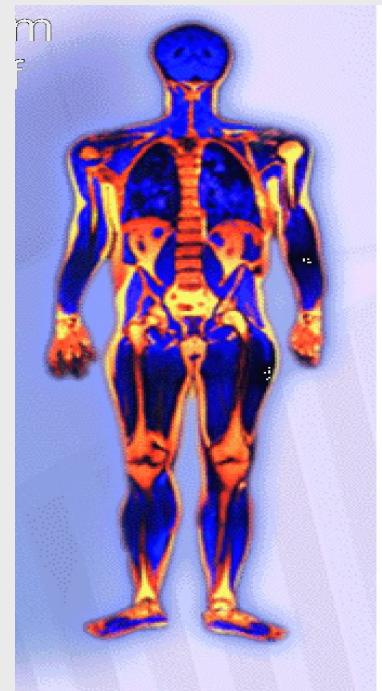
- Which resources should the system adapt to ?
 - **system**: network, energy, CPU, memory, etc.
 - **environment**: luminosity, temperature, etc.
 - etc.
- An adaptable system must take into account:
 - **all resources** whose quality/availability may change (with time/location/etc.) while the **application is running**.

Trading-off Adaptability and Invisibility

- Correct **adaptability** is key for **transparent operation**.
- If a system does not adapt at all:
 - **stops working**
- If a system adapts wrongly:
 - **It may malfunction**
- Both cases make the adverse conditions **visible!**
- Being adaptive implies:
 - **detecting changes** and **acting** accordingly.
 - **predicting changes** to allow **timely adaptation** (sometimes, in advance).
 - Ideally in an **autonomous** way...

Characteristics of an Autonomic System

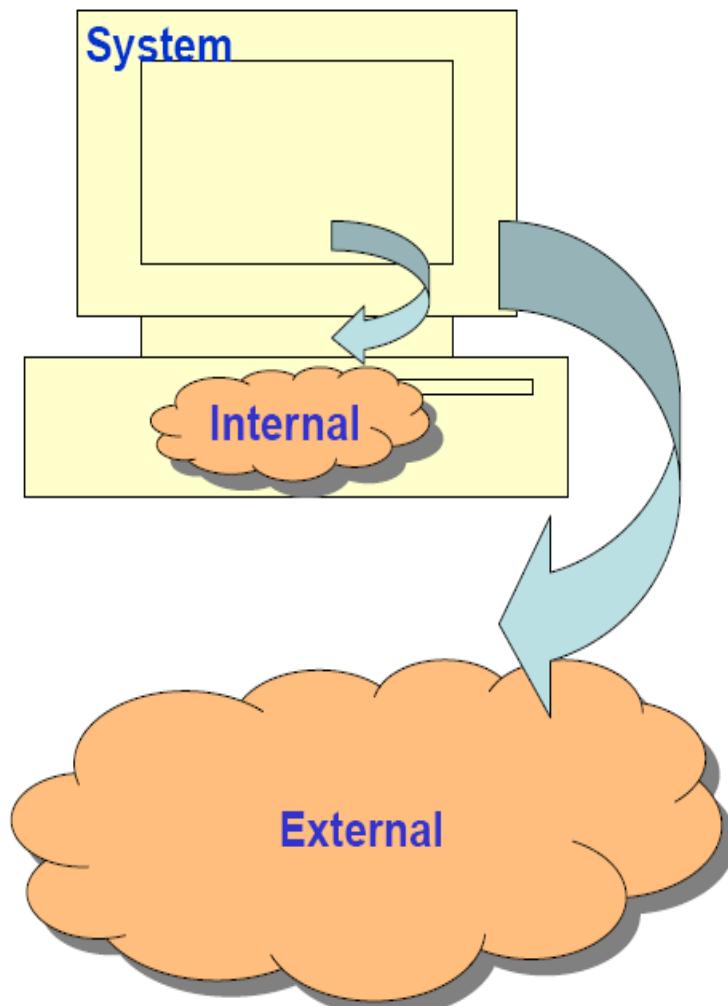
- In the original IBM proposal (“An architectural blueprint for autonomic computing”, a *de facto* standard):
 - Self-Configuration
 - Self-Optimization
 - Self-Repair
 - Self-Protection
- { Self-* or Self-X Properties



Self-Configuration

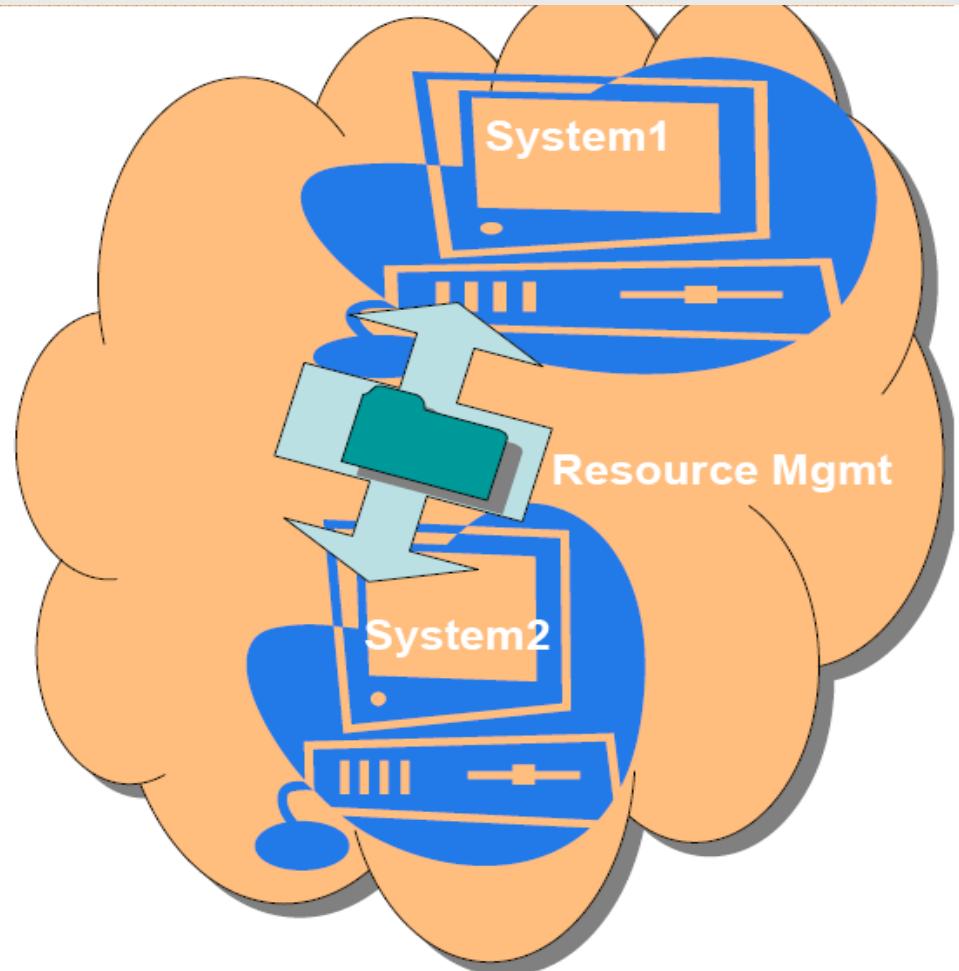
- Adapt automatically to the dynamically changing environment
- Internal adaptation
 - Add/remove new components (software)
 - *configures itself on the fly*
- External adaptation

Systems configure themselves into a global infrastructure



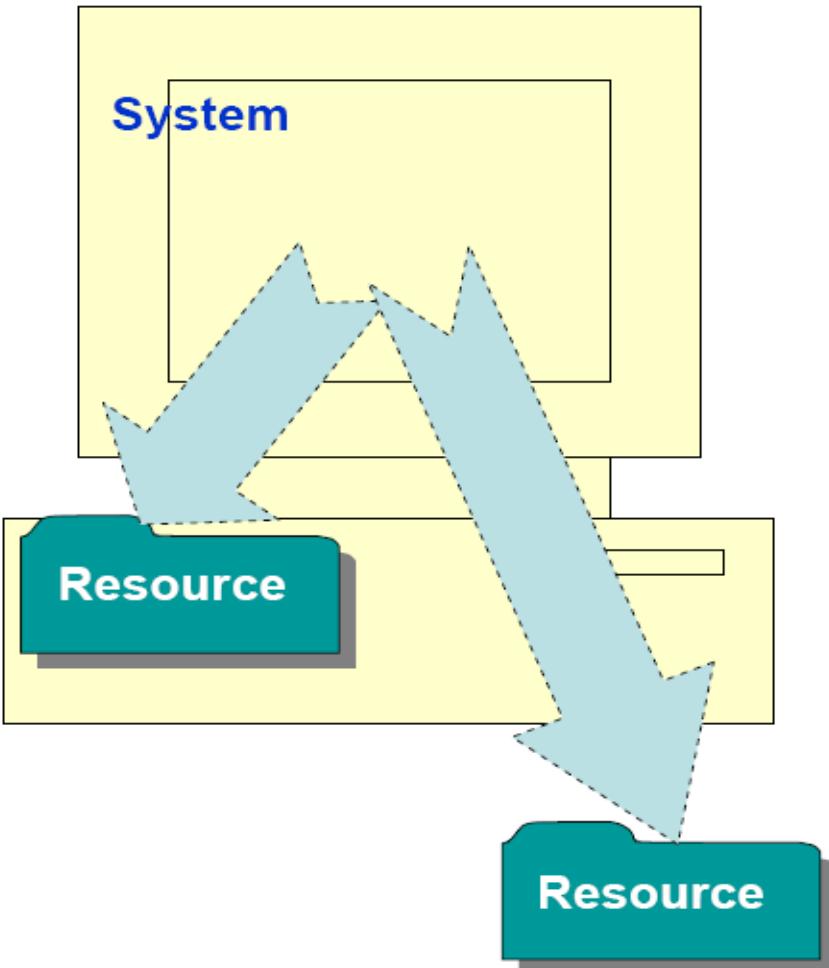
Self-Optimization

- Monitor and tune resources automatically
 - *Support operating in unpredictable environment*
 - *Efficiently maximization of resource utilization without human intervention*
- Dynamic resource allocation and workload management.
 - *Resource: Storage, databases, networks*
 - *For example, Dynamic server clustering*



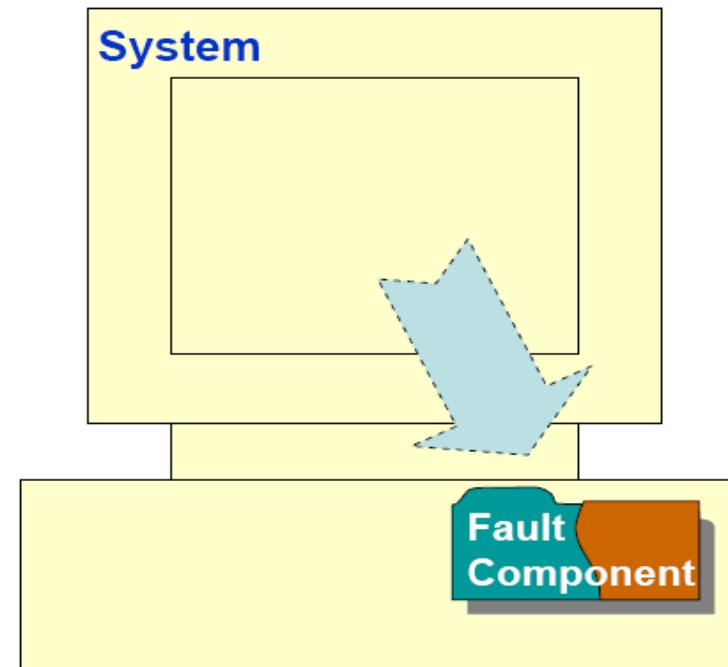
Self-Protection

- Anticipate, detect, identify and protect against attacks from anywhere
 - *Defining and managing user access to all computing resources*
 - *Protecting against unauthorized resource access, e.g. SSL*
 - *Detecting intrusions and reporting as they occur*



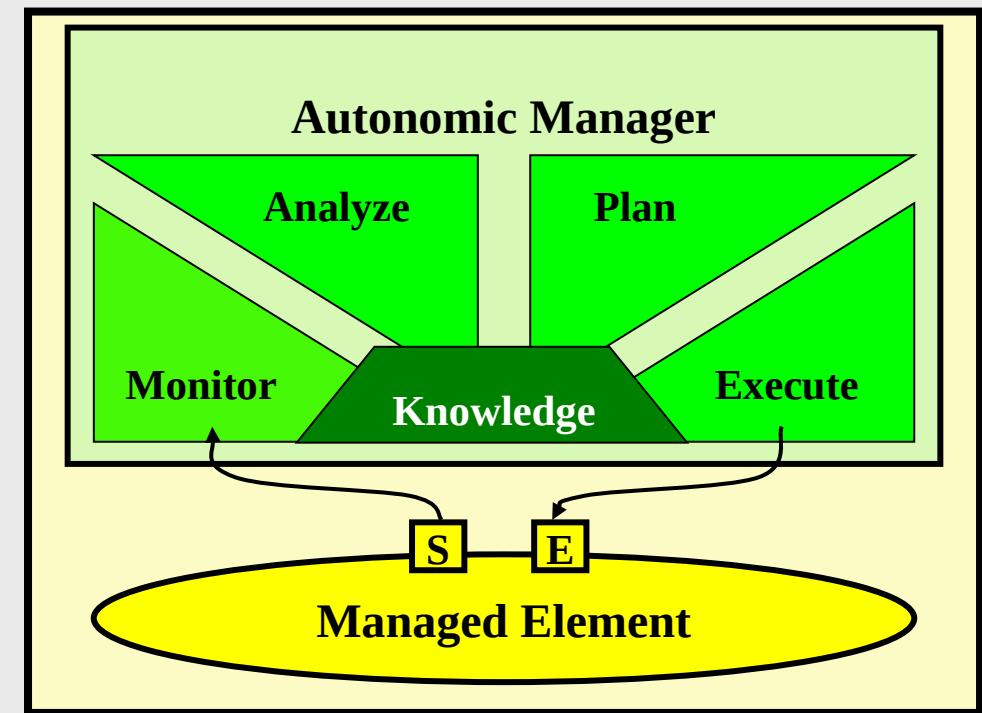
Self-Repair

- Discover, diagnose and react to disruptions without disrupting the service environment
- Fault components should be
 - *detected*
 - *Isolated*
 - *Fixed*
 - *reintegrated*



Autonomic Elements: Structure

- Fundamental atom of the architecture:
 - Managed element(s):
 - Database, storage system, server, software app, etc.
 - Plus *one* autonomic manager.
 - The **MAPE-K** model.
- Responsible for:
 - Providing its service.
 - Managing its own behavior in accordance with policies.
 - Interacting with other autonomic elements.



An Autonomic Element

Monitoring

- Capture environment properties (physical or virtual) that are relevant for decision.
- Highly implementation dependent component.
- Sensors read properties from the managed component: requests per second, power consumption,...
- Passive monitoring (no changes to system): e.g. /proc folder in Unix.
- Active monitoring (with changes to the system):
 - e.g. ProbeMeister, Pin (code injection for monitoring).
 - May require adaptive monitoring so as not to influence performance.

Analysis

- Combine monitoring symptoms into higher level descriptions.
- Process event streams according to policies:
 - Which sequences are to be interpreted?
 - Which sequences are discarded?
- May include the ECA (event-condition-action) component.
- Borders between Monitoring/Analysis/Planning are not rigid.

Planning

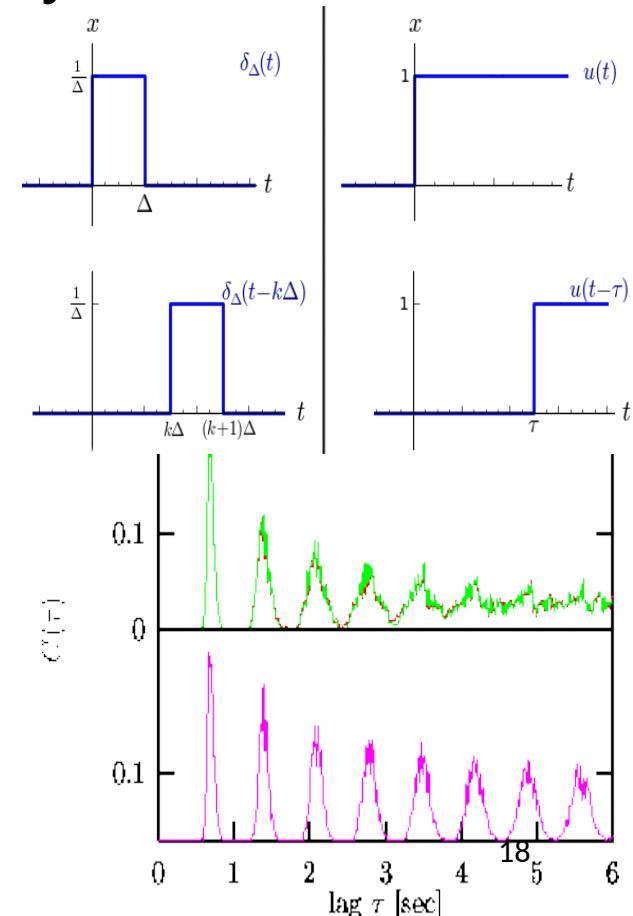
- Based on monitoring data decide changes to apply to the managed element.
- Two approaches:
 - ECA: a set of stateless rules to decide what actions to take. Often require conflict resolution mechanisms.
 - Model based approach: represent the managed element in a model. Actions are applied first on the model in order to detect problems and inconsistencies.

Execution

- Execution of the planning decisions.
- Based on dependencies between actions and intervention opportunities:
 - Convert the plan into a workflow.
 - Schedule executable actions.
 - Execute local and eventual remote actions.

Criteria for the Effectiveness of Adaptability Mechanisms

- User satisfaction
- “Dirac/Spike” changes on resources availability
- Reaction speed
- System stability
- Portability of the solutions
- Resource **usage** of the adaptability solutions



Adaptability Policies

- Adaptability can be provided through the enforcement of **declaratively-defined** policies supported by the middleware:
 - policies are not hard-coded in applications, and
 - can be deployed, enforced, and updated at any time
- Such middleware relies strongly on the following features:
 - the extensible capability to support the **specification and enforcement of runtime management policies**,
 - a set of **pre-defined policies** to control the mechanisms previously mentioned
 - for example: a pluggable set of basic mechanisms supporting object replication

Adaption Layer (operating system, middleware, application)

- At what level should the adaptability solution be implemented?
- Operating system:
 - oblivious to application semantics
 - global/equal solutions to all
 - implications with OS portability
 - no application awareness
- Middleware:
 - common mechanisms can be reused
 - interaction with applications to be semantic aware
 - portable
- Application:
 - semantic aware
 - specific for each application
 - not portable for different applications
 - With the issue of conflicting choices between applications

Conclusion

- What is adaptability ?
- Measuring how effective the system is w.r.t. automatic adaptability
- The autonomic computing model
- Adaption layer (operating system, middleware, application)
- A software architecture for adaptability
- Adaptability policies
- Access to objects (replication, consistency, transactions/sessions,failure)

**Mobile and Ubiquitous
Computing 2022-23
MEIC/METI - Alameda & Tagus**

**Replication & Consistency
Models**

Definitions: Device Connectivity

- System models:
 - how data is accessed, where it is stored, who is allowed to update it, how updated data propagates between devices, and what consistency is maintained
- A mobile system comprises a number of devices:
 - with computing, storage, and communication capabilities
 - can communicate with each other over a spectrum of networking technologies
- Two devices are *connected* if they can send messages to each other, either over a wireless or wired network
- *Weakly connected* devices can communicate, but only using a low-bandwidth, high latency connection
- A device is said to be *disconnected* if it cannot currently communicate with any other device
- Devices may experience *intermittent connectivity* characterized by alternating periods in which they are connected and disconnected

Definitions: items and collections

- An *item* is the basic unit of information that can be managed, manipulated, and replicated by devices
- Items include:
 - photos, songs, playlists, appointments, e-mail messages, files, videos, contacts, tasks, documents, and any other data objects of interest to mobile users
- Each item can be named by some form of *globally unique identifier*
- A *collection* is a set of related items, generally of the same type and belonging to the same person:
 - e.g., “Joe’s e-mail” is a collection of e-mail messages, etc.
 - it is an abstract entity that is not tied to any particular device or location or physical storage representation
 - each collection has a globally unique identifier so devices can refer to specific collections in replication protocols
- Collections can be shared and replicated among devices

Definitions: Full and Partial Replicas

- A *replica* is a copy of items from a collection that is stored on a given device
- A replica is a *full replica* if it contains all of the items in a collection
- As new items are added to a collection, copies of these items automatically appear in every full replica of the collection
- A *partial replica* contains a subset of the items in a collection
- Devices maintain their replicas in local, persistent storage, called data stores, so that the replicated items survive device crashes and restarts

Definitions: Operations

- Software applications running on a device can access:
 - the device's locally stored replicas, and
 - possibly replicas residing on other connected devices
- Such applications can perform four basic classes of operations on a replica:
 - CRUD: create, read, update, delete
- A *read* operation returns the contents of one or more items from a replica:
 - read operations include retrieving an item by its globally unique identifier, as in a conventional file system read operation, as well as querying items by content
- A *create* operation generates a new item with fresh contents and adds it to a collection:
 - this item is first created in the replica on which the create operation is performed, usually the device's local replica
 - it is then replicated to all other replicas for the same collection

Definitions: Operations

- An *update* operation changes the contents of an item (or set of items) in a replica, producing a new version of that item:
 - a file system write operation is an example of one that modifies an item
 - a SQL update statement on a relational database is also a modify operation
- A *delete* operation directly removes an item from a replica and the associated collection:
 - because the item is permanently deleted from its collection, it will be removed from all replicas of that collection
 - by contrast, a device holding a partial replica may choose to discard an item from its replica to save space without causing that item to be deleted from the collection

Summary

Replication Requirements for Basic Data-Oriented Systems

	REMOTE ACCESS	DEVICE- MASTER	PEER- TO-PEER	PUB-SUB
Continuous connectivity	✓			✓
Update anywhere		✓	✓	
Consistency		✓	✓	✓
Topology independence			✓	
Conflict handling		✓	✓	
Partial replication	✓		✓	✓

Summary

	DEVICES	DATA	READS	UPDATES
Remote access	Server plus mobile devices	Web pages, files, databases, etc.	Performed at server	Performed at server
Device caching	Server plus mobile devices	Web pages, files, databases, etc.	Performed on local cache; performed at server for cache misses	Performed at server and (optionally) in local cache
Device-master replication	Master replica plus mobile devices	Calendars, e-mail, address book, files, music, etc.	Performed on local replica	Performed on local replica; sent to master when connected
Peer-to-peer replication	Any mobile or stationary device	Calendars, e-mail, address book, files, music, etc.	Performed on local replica	Performed on local replica; synchronized with others lazily
Pub-sub	Publisher plus mobile devices	Events, weather, news, sports, etc.	Performed at subscriber's local replica	Performed at publisher; disseminated to subscribers

Data Consistency

Strong Consistency

- Consistency provided by a replicated system:
 - it is an indication of the extent to which users must be aware of the replication process and policies
- Systems that provide **strong consistency** try to exhibit **identical behavior to a non replicated system**:
 - this property is often referred to as **one-copy serializability**
 - it means that an application program, when performing a **read operation**, receives the data resulting from the **most recent update operation(s)**
 - update operations are performed at each device in some well-defined order, at least conceptually
 - maintaining strong consistency **requires substantial coordination** among devices that replicate a data collection which is not compatible with intermittent connectivity.
 - typically, **all or most replicas must be updated atomically** using multiple rounds of messages, such as a two-phase commit protocol

Weak Consistency

- Relaxed/optimistic consistency models have become popular for replicated systems:
 - due to their **tolerance of network** and **replica failures** and their **ability to scale** to large numbers of replicas
- Especially important in **mobile environments**:
 - rather than remaining mutually consistent at all times, replicas are allowed to **temporarily diverge** by accepting updates to local data items
 - such **updates propagate lazily** to other replicas
- Read operations performed on a device's local replica may return data that does not reflect recent updates made by other devices:
 - users and applications must be **able to tolerate potentially stale information**
- Mobile systems generally strive for **eventual consistency**:
 - guaranteeing that **each replica eventually receives the latest update** for each replicated item
- Other stronger (and weaker) consistency guarantees are possible

Eventual Consistency

- A system providing **eventual consistency** guarantees that:
 - replicas **eventually converge to a mutually consistent state**, i.e., to identical contents, if **update activity ceased**
- Ongoing updates may prevent replicas from ever reaching identical states:
 - especially in a mobile system where **communication delays between replicas can be large** due to intermittent connectivity
 - thus, a more pragmatic definition of eventual consistency is desired
- Practically, a mobile system provides eventual consistency if:
 - each update operation is eventually received by each device,
 - noncommutative updates are performed in the same order at each replica, and
 - the outcome of applying a sequence of updates is the same at each replica
- Eventually consistent systems make **no guarantees whatsoever about the freshness of data** returned by a read operation:
 - readers are simply assured of **receiving items that result from a valid update operation** performed sometime in the past
 - e.g. a person **might update a phone number** from her **cell phone** and then be **presented with the old phone number** when querying the address book on **her laptop**

Causal Consistency

- In a system providing causal consistency:
 - a user **may read stale data** but is at least guaranteed to observe states of replicas in which **causally related update operations** have been performed in the proper order
- Suppose an **update originates at some device that had already received and incorporated a number of other updates** into its local replica:
 - this new update is said to **causally follow** all of the previously received updates
 - a causally consistent replicated system ensures that

if update U2 follows update U1, then a user is never allowed to observe a replica that has performed update U2 without also performing update U1
- If two **updates are performed concurrently**, that is, without knowledge of each other:
 - they **can be incorporated into different devices in different observable orders**

Session Consistency

- One **potential problem** faced by users who **access data from multiple devices** is that they **may observe data that fluctuates in its staleness**:
 - e.g. a user may update a phone number on her cell phone and then read the new phone number from her tablet but later read the old phone number from her laptop
- Session guarantees have been devised to:
 - provide a **user** (or application) with a **view of a replicated database** that is **consistent with respect to the set of read and update operations** performed by that user **while still allowing temporary divergence** among replicas
- **Unlike most consistency models** which provide system wide guarantees, **session guarantees are individually selectable by each user or application.**
- **Application designers can choose the set of session guarantees** that they desire based on the **semantics of the data that they manage** and the expected access patterns

Definition of Session

- A **session** is an abstraction for the **sequence of read and write operations** performed during the execution of an application
- Sessions are **not intended to correspond to atomic transactions** that ensure atomicity and serializability
- Instead, the intent is to:
 - present individual applications with a **view of the database that is consistent with their own actions**,
 - even if they read and write from various, potentially inconsistent servers
- We want the **results of operations performed in a session to be consistent** with:
 - the model of a **single centralized server**,
 - possibly being **read and updated concurrently by multiple clients**

Session Guarantees

- Session guarantees can be **easily implemented on mobile devices**:
 - provided some **small state** can be **carried with the user** as she **switches between devices**
- More practically:
 - this state can be embedded in applications that access data from mobile devices
- However, systems providing session guarantees on top of an eventually consistent replication protocol may need to:
 - **occasionally prevent access** to some device's replica
 - i.e., **availability may be reduced to enforce the desired consistency properties**, which could adversely affect mobile users.
- One practical alternative is for the system to simply:
 - **inform the user** (or application) when an operation violates a session guarantee, but
 - allow that operation to **continue with weaker consistency**

Session Guarantees (from weakest to strongest)

- Read Your Writes:

- **read** operations reflect previous **writes**

Violation Example: after changing his password, a user would occasionally type the new password and receive an “invalid password” response

- Monotonic Reads:

- successive **reads** reflect a non decreasing set of **writes**

Violation Example: on a calendar recently added (or deleted) meetings may seem to appear and disappear

- Writes Follow Reads:

- **writes** are propagated after **reads** on which they depend

Violation Example: shared bibliographic database to which users contribute entries: a user reads some entry, discovers that it is inaccurate, and then issues a Write to update the entry and it's not in the DB

- Monotonic Writes:

- **writes** are propagated after **writes** that logically precede them

Violation Example: version N is written to some server, and version N+1 to a different server, and on some site version N+1 is applied before version N

Session Guarantees

- These properties are "**guaranteed**" in the sense that:
 - either the storage system **ensures them for each read and write operation belonging to a session**, or
 - it **informs the calling application that the guarantee cannot be met**
- The guarantees can easily be layered **on top of a weakly-consistent replicated data system**:
 - **each read or write operation is performed at a single server**, and
 - the **writes are propagated to other servers in a lazy fashion**
- To ensure that the guarantees are met:
 - **the servers at which an operation can be performed must be restricted to a subset of available servers that are sufficiently up-to-date**

System Model

- Basic assumption:
 - a **weakly consistent replicated storage system** to which the guarantees will be added
 - it consists of **a number of servers that each hold a full copy of some replicated database** and **clients that run applications desiring access to the database**
- The session guarantees are applicable to systems in which **clients and servers may reside on separate machines** and a **client accesses different servers over time**:
 - e.g, a mobile client may choose servers based on which ones are available in its region and can be accessed most cheaply

System Model

- Definition and implementation of session guarantees:
 - it is **unaffected by whether Writes are simple database updates or more complicated atomic transactions**
- Each **Write** has a **globally unique identifier**:
 - it is called a “**WID**”
 - the server that first accepts the Write, for instance, might be responsible for assigning its WID
- **Read and Write** operations may be **performed at any server or set of servers**
- The guarantees are presented assuming that **each Read or Write is executed against a single server’s copy of the database**:
 - i.e. for the most part, we discuss variants of a **read-any/write-any** replication scheme
 - however, the guarantees could also be used in systems that read or write multiple copies, such as all of the available servers in a partition

Terminology

- We define $\text{DB}(S,t)$ to be:
 - the ordered **sequence of Writes that have been received by server S at or before time t**
 - if it is known to be the current time, then it may be omitted leaving $\text{DB}(S)$ to represent the current contents of the server's database
- Conceptually:
 - **server S creates its copy of the database,**
 - **it uses it to answer Read requests,**
 - **it starts with an empty database and applies each Write in $\text{DB}(S)$ in the given order**
- In practice, a **server is allowed to process the Writes in a different order as long as they are commutative.**
- The **order of Writes in $\text{DB}(S)$ does not necessarily correspond to the order in which server S first received the Writes.**

Terminology

- Weak consistency permits database copies at different servers to vary:
 - **DB(S₁,t) is not necessarily equivalent to DB(S₂,t) for two servers S₁ and S₂**
- Practical systems generally desire eventual consistency:
 - servers converge towards identical database copies in the absence of updates
 - thus relying on two properties: **total propagation** and **consistent ordering**
- We **assume** that the **replicated system provides eventual consistency** and thus includes mechanisms to ensure these two properties as follows:
 - **Writes are propagated among servers** by a process called **anti-entropy**.
 - anti-entropy ensures that **each Write is eventually received by each server**
 - i.e., **for each Write W there exists a time t such that W is in DB(S,t) for each server S**
- There are no other assumptions about:
 - the anti-entropy protocol,
 - the frequency with which it happens,
 - the policy by which servers choose anti-entropy partners, or
 - other characteristics of the anti-entropy process

Terminology

- All servers apply non-commutative **Writes** to their databases in **the same order**:
 - let **WriteOrder(W1,W2)** be a boolean predicate indicating whether Write W1 should be ordered before Write W2
 - the system ensures that **if WriteOrder(W1,W2) then W1 is ordered before W2 in DB(S) for any server S that has received both W1 and W2**
- In a **strongly consistent** system:
 - WriteOrder would reflect the order in which individual Writes or transactions are committed
- In an **eventually consistent** system:
 - servers could use any of a variety of techniques to agree upon the order of Writes
 - e.g. using timestamps to determine the Write order does not imply that servers have synchronized clocks since there is no requirement that Writes be ordered by the actual time at which they were performed
- We make **no assumption** about:
 - how servers agree on the ordering of Writes, or
 - about how servers make their copies of the database conform to this ordering

Providing the Guarantees

- The implementations require only minor cooperation from the servers that process Read and Write operations
- Specifically, a server must be willing to return information about the:
 - **unique identifier** (WID) assigned to a **new Write**,
 - the **set of WIDs for Writes** that are **relevant to a given Read**, and
 - the **set of WIDs for all Writes** in its database
- The burden of providing the guarantees lies primarily with the **session manager on the client**:
 - through which all of a session's Read and Write operations are **serialized**
 - it is a **component of the client stub** that mediates communication with available servers
- For each session, the session manager it maintains two sets of WIDs:
 - **write-set** = set of WIDs for those Writes performed in the session
 - **read-set** = set of WIDs for the Writes that are relevant to session Reads

Session Guarantees – Read Your Writes

- The **Read Your Writes** guarantee is motivated by the fact that:
 - users and applications find it particularly confusing **if they update a database and then immediately read from the database only to discover that the update appears to be missing**
- This guarantee ensures that:
 - the **effects of any Writes made within a session are visible to Reads within that session**
 - thus, **Reads are restricted to copies of the database that include all previous Writes in this session**
- RYW-guarantee:
 - if **Read R follows Write W in a session, and**
 - **R is performed at server S at time t, then**
 - **W is included in DB(S,t)**
- Applications are **not guaranteed** that:
 - a **Read following a Write to the same data item will return the previously written value**
 - in particular, **Reads within the session may see other Writes that are performed outside the session**

Read Your Writes – example 1

- After **changing his password**, a user would occasionally type the new password and **receive an “invalid password” response**:
 - this annoying problem would arise because **the login process contacted a server to which the new password had not yet propagated**
 - the problem **can occur in any weakly consistent system** that manages passwords
- It **can be solved cleanly by having a session per user in which the RYW-guarantee is provided**:
 - such a session should be created for each new user and must exist for the lifetime of the user's account
 - by performing updates to the user's password as well as checks of this password within the session, users can use a new password without regard for the extent of its propagation
- The **RYW-guarantee ensures that the login process will always read the most recent password**
- Notice that this application requires a session to persist across logouts and machine reboots

Read Your Writes – example 2

- Consider a user whose **electronic mail is managed in a weakly consistent replicated database:**
 - as the **user reads and deletes messages**, those messages are **removed from the displayed “new mail” folder**
 - if the **user stops reading mail and returns sometime later**, she **should not see deleted messages** reappear simply because the mail reader refreshed its display from a different copy of the database
- The **RYW-guarantee can be requested within a session used by the mail reader:**
 - to ensure that the effects of any actions taken, such as deleting a message or moving a message to another folder, remain visible

Providing Read Your Writes

- It ensures that:
 - the effects of any Writes made within a session are visible to Reads within that session
 - thus, **Reads are restricted to copies of the database that include all previous Writes in this session**
- It involves two basic steps:
 - whenever a Write is accepted by a server, its assigned WID is added to the session's write-set
 - before each Read to server S at time t, **the session manager must check that the write-set is a subset of DB(S,t)**
- This check could be done:
 - **on the server** by passing the **write-set** to it, or
 - **on the client** by retrieving the **server's list of WIDs**
- The session manager can continue trying available servers until it discovers one for which the check succeeds:
 - if it cannot find a suitable server, then it reports that the guarantee cannot be provided

read-set = set of WIDs for the Writes that are relevant to session Reads
write-set = set of WIDs for those Writes performed in the session

Session Guarantees – Monotonic Reads

- The **Monotonic Reads** guarantee permits users to **observe a database that is increasingly up-to-date over time**:
 - it ensures that **Read operations are made only to database copies containing all Writes whose effects were seen by previous Reads within the session**
- Intuitively, a **set of Writes completely determines the result of a Read if**:
 - the **set includes “enough” of the database’s Writes**
 - so that the **result of executing the Read against this set is the same as executing it against the whole database**.
 - **These are the relevant Writes for that read R1: $\text{RelevantWrites}(S_1, t_1, R_1)$**
- Specifically, we say a Write set WS is complete for Read R and DB(S,t) if and only if:
 - WS is a subset of DB(S,t) and
 - for any set WS2 that contains WS and is also a subset of DB(S,t),
 - the result of R applied to WS2 is the same as the result of R applied to DB(S,t)
- Monotonic Reads Guarantee:
 - **if Read R1 occurs before R2 in a session, and**
 - **R1 accesses server S1 at time t1 and R2 accesses server S2 at time t2, then**
 - **$\text{RelevantWrites}(S_1, t_1, R_1)$ is a subset of $\text{DB}(S_2, t_2)$**

Monotonic Reads – example 1

- A user's appointment calendar is stored online in a replicated database:
 - where it can be updated by both the user and automatic meeting schedulers
- The user's calendar program periodically refreshes its display by reading all of today's calendar appointments from the database
 - if it accesses servers with inconsistent copies of the database, recently added (or deleted) meetings may appear to come and go
- The MR-guarantee can effectively prevent this since:
 - it disallows access to copies of the database that are less current than the previously read copy

Monotonic Reads – example 2

- Consider a **replicated electronic mail database**
- The mail reader issues a **query to retrieve all new mail messages** and **displays summaries** of these to the user
- When the user **issues a request to display one of these messages**, the mail reader **issues another Read** to retrieve the message's contents
- The MR-guarantee can be used by the mail reader to ensure that:
 - the **second Read is issued to a server that holds a copy of the message**
- Otherwise, the user, upon trying to display the message, might **incorrectly be informed that the message does not exist**

Providing Monotonic Reads

- It ensures that:
 - Read operations are made only to **database copies containing all Writes whose effects were seen by previous Reads within the session**
- It involves two basic steps:
 - before each Read to server S at time t, the session manager must **ensure that the read-set is a subset of $DB(S,t)$**
 - after each Read R to server S, the **WIDs for each Write in $RelevantWrites(S,t,R)$ should be added to the session's read-set**
- This assumes that the server can compute the relevant Writes and return this information along with the Read result

read-set = set of WIDs for the Writes that are relevant to session Reads
write-set = set of WIDs for those Writes performed in the session

Session Guarantees – Writes Follows Reads

- The Writes Follow Reads guarantee ensures that **traditional Write/Read dependencies are preserved** in the ordering of Writes at all servers:
 - in every copy of the database, **Writes made during the session are ordered after any Writes whose effects were seen by previous Reads in the session**
- WFR-guarantee:
 - if Read R1 precedes Write W2 in a session and
 - R1 is performed at server S1 at time t1, then,
 - for any server S2, if W2 is in DB(S2) then any W1 in RelevantWrites(S1,t1,R1) is also in DB(S2) and WriteOrder(W1,W2)
- This guarantee is different in nature from the previous two guarantees in that **it affects users outside the session**:
 - not only does the session **observe that the Writes it performs occur after any Writes it had previously seen**, but
 - also **all other clients will see the same ordering** of these Writes regardless of whether they request session guarantees

Writes Follows Reads – example

- Imagine a **shared bibliographic database** to which users contribute entries describing published papers:
 - suppose that a user **reads some entry**, discovers that it is inaccurate, and then **issues a Write to update the entry**
 - e.g., the person might discover that the page numbers for a paper are wrong and then correct them with a Write such as “UPDATE bibdb SET pages = ‘45-53’ WHERE bibid = ‘Jones93’.”
- The WFR-guarantee can ensure that:
 - the **new Write updates the previous bibliographic entry at all servers**
- The WFR-guarantee, as defined, associates two constraints on Write operations:
 - a constraint on Write order ensures that **a Write properly follows previous relevant Writes in the global ordering** that **all database replicas will eventually reflect**
 - a constraint on propagation ensures that **all servers (and hence all clients) only see a Write after they have seen all the previous Writes on which it depends**
- This example requires both these properties

Providing WFR and MW

- Providing the Writes Follow Reads and Monotonic Writes guarantees requires:
 - two additional, but reasonable, constraints on the servers' behavior
- C1
 - when a server S accepts a new Write W2 at time t, it ensures that $\text{WriteOrder}(W_1, W_2)$ is true for any W_1 already in $\text{DB}(S, t)$
 - that is, new Writes are ordered after Writes that are already known to a server
- C2
 - anti-entropy is performed such that if W_2 is propagated from server S_1 to server S_2 at time t then any W_1 in $\text{DB}(S_1, t)$ such that $\text{WriteOrder}(W_1, W_2)$ is also propagated to S_2
- Strictly speaking, the two conditions discussed above must hold for any Write W_1 in the session's read-set or write-set rather than for any Write in $\text{DB}(S, t)$:
 - this subtle distinction is not likely to have a practical consequence since the weaker requirements would require a server to keep track of clients' read-sets and write-sets.
 - the stronger requirements allow a server's behavior to be independent of the session state maintained by clients

read-set = set of WIDs for the Writes that are relevant to session Reads
write-set = set of WIDs for those Writes performed in the session

Providing Writes Follows Reads

- It ensures that traditional Write/Read dependencies are preserved in the ordering of Writes at all servers:
 - in every copy of the database, **Writes made during the session are ordered after any Writes whose effects were seen by previous Reads in the session**
- It involves two basic steps:
 - **each Read R to server S at time t results in $\text{RelevantWrites}(S,t,R)$ being added to the session's read-set**
 - **before each Write to server S at time t, the session manager checks that this read-set is a subset of $\text{DB}(S,t)$**

read-set = set of WIDs for the Writes that are relevant to session Reads

write-set = set of WIDs for those Writes performed in the session

Monotonic Writes

- The Monotonic Writes guarantee says that **Writes must follow previous Writes within the session**
- In other words:
 - a **Write is only incorporated** into a server's database copy **if the copy includes all previous session Writes**
 - the Write is ordered after the previous Writes
- MW-guarantee:
 - if Write W1 precedes Write W2 in a session, then,
 - for any server S2, if W2 in DB(S2) then W1 is also in DB(S2) and $\text{WriteOrder}(W1, W2)$
- This guarantee provides **assurances that are relevant both to the user of a session as well as to users outside the session**

Monotonic Writes – example 1

- The MW-guarantee could be used by a text editor when editing replicated files to ensure that:
 - if the user saves version N of the file and later saves version N+1 then version N+1 will replace version N at all servers
- In particular, it avoids the situation in which:
 - version N is written to some server, and
 - version N+1 to a different server, and
 - the versions get propagated such that version N is applied before N+1

Monotonic Writes – example 2

- Consider a **replicated database containing software source code**
- Suppose that a programmer **updates a library to add functionality in an upward compatible way:**
 - this new library can be propagated to other servers in a **lazy fashion since it will not cause any existing client software to break**
 - however, suppose that the programmer **also updates an application to make use of the new library** functionality
 - if the new application code gets written to servers that have not yet received the new library, then the code will not compile successfully
- To avoid this potential problem, the programmer can:
 - **create a new session that provides the MW-guarantee**, and
 - issue the Writes containing new versions of both the library and application code within this session

Providing Monotonic Writes

- It requires that:
 - a Write is only incorporated into a server's database copy **if the copy includes all previous session Writes**
- It involves two basic steps:
 - in order for a server S to accept a Write at time t, **the server's database, DB(S,t), must include the session's write-set**
 - also, whenever a **Write is accepted by a server, its assigned WID is added to the write-set**

read-set = set of WIDs for the Writes that are relevant to session Reads
write-set = set of WIDs for those Writes performed in the session

Read / Write Guarantees

- Operations on which a session is updated or checked

Guarantee	session state updated on	session state checked on
<i>Read Your Writes</i>	Write	Read
<i>Monotonic Reads</i>	Read	Read
<i>Writes Follow Reads</i>	Read	Write
<i>Monotonic Writes</i>	Write	Write

Version Vectors

- A **version vector** is a sequence of **<server, clock> pairs**, one for each server
- The **server portion is simply a unique identifier** for a particular copy of the replicated database
- The **clock is a value from the given server's monotonically increasing logical clock**
- The only constraint on this **logical clock** is that it **must increase for each Write accepted by the server**:
 - a Lamport clock
 - a real-time clock or
 - simply a counter
- A **<server,clock> pair serves nicely as a WID**, and we assume that **WIDs are assigned in this manner by the server that first accepts the Write**

Version Vectors at Each Server

- **Each server maintains its own version vector** with the following invariant:
 - if a server has $\langle S, c \rangle$ in its version vector, then **it has received all Writes that were assigned a WID by server S before or at logical time c on S's clock**
- For this invariant to hold, during anti-entropy:
 - **servers must transfer Writes in the order of their assigned WIDs**
- A **server's version vector is updated** as part of the anti-entropy process so that:
 - it precisely **specifies the set of Writes in its database**
- Assuming the use of version vectors by servers:
 - more practical implementations of the guarantees are possible in which **the sets of WIDs are replaced by version vectors** (next slide)

Replacing set of WIDs by Version Vectors

- To obtain a **version vector V providing a representation for a set of WIDs, Ws**:
 - set $V[S] =$ the time of the latest WID assigned by server S in Ws (or 0 if no Writes are from S)
- To obtain a version vector V that represents the **union of two sets of WIDs, Ws1 and Ws2**:
 - first obtain V1 from Ws1 and V2 from Ws2 as above
 - then, set $V[S] = \text{MAX}(V1[S], V2[S])$ for all S
- To check **if one set of WIDs, WS1, is a subset of another, WS2**:
 - first obtain V1 from WS1 and V2 from WS2 as above.
 - then, check that V2 “dominates” V1, where dominance is defined as one vector being greater or equal to the other in all components
- With these rules, **the state maintained for each session compacts into two version vectors**:
 - one to **record the session's Writes**, and
 - one to **record the session's Reads** (actually the Writes that are relevant to the session's Reads)

Finding a Server

- To find an acceptable server:
 - the **session manager must check that one or both of these session vectors are dominated by the server's version vector**
- Which session vectors are checked depends on the operation being performed and the guarantees being provided within the session
- **Servers return a version vector along with Read results to indicate the relevant Writes:**
 - in practice, servers may have difficulty computing the set of relevant Writes
 - 1) determining the relevant Writes for a complex query, such as one written in SQL, may be costly
 - 2) it may require servers to maintain substantial bookkeeping of which Writes produced or deleted which database items
- In real systems, servers typically do not remember deleted database entries:
 - they just store a copy of the database along with a version vector
 - for such systems, **a server is allowed to return its current version vector as a gross estimation of the relevant Writes**
 - this **may cause the session manager to be overly conservative when choosing acceptable servers**

Performance Improvement

- Checks for a suitable server can be amortized over many operations within a session:
 - in particular, the **previously contacted server** is always an acceptable choice for the server at which to perform the next Read or Write operation
 - if the session manager "latches on" to a given server, then the **checks can be skipped**
 - only when the session manager **switches to a different server**, like when the previous server becomes unavailable, must a server's current version vector be compared to the session's vectors
- To facilitate finding a server that is sufficiently up-to-date, **the session manager can cache the version vectors of various servers**
- Since **a server's database can only grow over time in terms of the numbers of Writes it has received and incorporated:**
 - cached version vectors represent a lower bound on a server's knowledge

Mobile and Ubiquitous Computing

2021/22

MEIC/METI - Alameda & Tagus

Cyberforaging

Introduction

**Should apps run entirely on
mobile devices ?**

Motivation – how to partition

- Mobile applications increasingly pervade our daily lives
- Innovation in computing hardware has continually provided mobile users with more computational resources in lighter and smaller form factors:
 - laptops have replaced personal computers, and
 - are now in turn being replaced by tablets and smart phones
- The portability of current mobile devices satisfies this demand by allowing their users to run applications anywhere at any time
- However:
 - most mobile applications do not run in isolation on the mobile device
 - they access data and computational resources in the cloud via wireless (e.g. cellular base stations and WiFi access points)
- Wired infrastructure components such as compute servers and data stores provide more computational resources than a mobile device:
 - they are not limited by stringent size, weight, and battery energy constraints

“How should one best partition applications across mobile platforms and fixed infrastructure?”

Motivation - static vs dynamic

- Static partitioning fails to account for the variability inherent in mobile and pervasive computing environments:
 - wireless **network quality can change** by one or more orders of magnitude as users move
 - mobile devices **may become disconnected** from the fixed infrastructure
 - shared computational **resources in the cloud may become overtaxed**
 - nearby computational **resources may become available** for opportunistic usage
 - **resources** available to different mobile devices **can vary substantially** from platform to platform
 - **battery/energy/cellular bandwidth are limited resources** and may need to be consumed within a budget
 - application **demand** and user **workloads** may **change over time**
- There is no “always-best” static partitioning of application functionality
- Thus, any static partitioning will inevitably lead to:
 - a poor user experience when the actual usage environment varies from that envisioned by the developer

Cyberforaging – partition, migration, replication

- It enables the seamless mobility of data and computation
- Users are given the illusion that their applications run entirely locally on their mobile devices
- This illusion preserves the pervasive, always available access to applications and data that has driven the growth of mobile computing
- Behind this illusion, however:
 - computation and data may be replicated and migrated between the mobile computer and fixed infrastructure

What is Partitioning

Given a specific application state and a specific computational environment, which portions of the application should run on the mobile computer and which should run on remote infrastructure?

- Dynamics depend on:
 - the application state may depend on specific inputs and the current location of data used in the computation
 - the mobile environment (server load, network bandwidth, etc.) is constantly changing
- Partitioning decision:
 - **objectives** that cyber foraging systems attempt to achieve through application partitioning and the **metrics** used to quantify those objectives
 - different approaches to enumerating the **set of candidate partitions**
 - strategies used to **select** one of those candidates
 - strategies to **estimate resource supply and demand**
 - how cyber foraging **recovers from failures** that occur during partitioned execution
 - which **characteristics** make applications particularly well suited or poorly suited for cyber foraging

What are the benefits of partitioning apps?

Potential Benefits

- There are many reasons why an application may benefit from ***executing a computation*** or ***storing data*** on remote infrastructure instead of or in addition to the mobile computer on which it is currently executing:
 - better performance, less battery used, data fidelity
- There is a fundamental performance gap between mobile and fixed computers:
 - even as computing power has increased rapidly over the past few decades, the computational resources available to mobile devices have substantially lagged behind their fixed counterparts

Year	Representative server		Representative handheld	
	Processor	Speed	Device	Speed
1997	Pentium II	266 MHz	PalmPilot	16 MHz
2002	Itanium	1 GHz	Blackberry 5810	133 MHz
2007	Core 2	9.6 GHz (4 cores)	Apple iPhone	412 MHz
2012	Xeon E3	14 GHz (2x4 cores)	Samsung Galaxy 3S	3.2 GHz (2 cores)

- Comparing the processing power of representative computer systems in five-year increments from 1997–2012:
 - as a rough estimate, processing power is given by the **product of core count and clock speed**
 - although both server and mobile computers both show rapid growth in processing power, the **performance gap between the two remains substantial** for every time period examined

Benefit - performance

- Due to the gap between mobile and infrastructure processing power:
 - **compute-intensive** applications can execute much faster on remote infrastructure than on mobile devices
- On the other hand:
 - **interactive activities** that demand few computational resources may execute almost as fast on a mobile computer as they do on a server
- Further, performance is not impacted solely by processor speed:
 - remote infrastructure may offer **more memory and storage**,
 - the ability to **parallelize computation** across multiple cores and servers, or
 - better **network connectivity**

Benefit - battery

- Designers strive to make mobile computing systems as energy-efficient as possible, for example:
 - employing hardware power-saving modes, or by
 - reducing the scope or quality of activities performed by mobile applications
- While these measures are essential to extending battery lifetime, they also:
 - noticeably degrade the mobile user experience
 - applications take longer to perform interactive activities and produce lower-quality results
- Use of fixed infrastructure is an attractive alternative for designers:
 - by **offloading computation or data storage** from a battery-powered computer to a remote computer with wall power,
 - the operational lifetime of the battery-powered mobile computer can sometimes be extended
 - without degrading the user experience

Benefit - fidelity

- Fidelity:
 - the degree to which the results produced by an application match the highest quality results that would be produced given ample computational resources
- Two reasons why a mobile computer operating alone may choose to reduce application fidelity:
 - to **achieve acceptable response times** for interactive applications
 - to **extend the mobile computer's battery** lifetime
- For demanding applications:
 - fidelity reduction may sometimes be absolutely necessary in order to get the application to run on the mobile device at all
 - e.g. a fixed computer may be able to execute a intense computation that a mobile computer is simply incapable of doing locally due to insufficient memory or storage.
- Offloading a computation or data query to a cluster in the cloud provides access to orders of magnitude more resource than a mobile computer has available locally

Costs of Using Remote Infrastructure (1/3)

- Offloading application from the mobile computer may degrade performance:
 - the mobile computer must **ship the inputs** to that computation to the remote infrastructure over a wireless network
 - after the computation completes, the mobile computer must **retrieve the results** of the computation
- Unless the computation is asynchronous and not on the critical path of the application, **performance is improved only if** the:
 - time saved by performing the computation remotely is greater than
 - the time spent communicating inputs and outputs at the start and end of the computation
- When
 - **network latency** between the mobile and remote computers is **high**,
 - **bandwidth is low**, or
 - the **amount of data shipped** per unit of computation is **large**
- it may be **quicker to perform the computation on the mobile computer** than it would be to perform the computation remotely

Costs of Using Remote Infrastructure (3/3)

Factor	Favorable for remote execution	Favorable for local execution
Network bandwidth	High	Low
Network latency	Low	High
Disparity between compute resources	High	Low
Granularity of computation	Large	Small
Parallelism in computation	High	Low
Memory/storage requirements	High	Low
Size of inputs/outputs	Small	Large
Load on remote computers	Low	High
Wireless network power	Low	High
Sensitivity of activity	N/A	Sensitive

Candidate Partitions

Where can an app be split?

Candidate Partitions

- How to pick a method for determining candidate partitions for applications?
- Theoretically:
 - the **number of possible application partitions is very large** since partitioning could employ granularities as fine as single instructions
- From a practical standpoint:
 - **very fine-grained partitions are ineffective** because substantial computation is needed to offset the performance and energy costs of using the network
 - further, the **performance overhead of the partitioning decision must be small** since it is executed dynamically; this limits the number of possible partitions that can be considered
- Thus, cyberforaging systems typically:
 - first **enumerate** a small number of possible partitions,
 - which we call the **candidate** partitions, and then
 - dynamically **select** the candidate partition that best achieves the system goal

Candidate Partitions - granularity

- There is considerable difference of opinion about how best to enumerate candidate partitions

Cyber Foraging System	Partition Granularity	Programmer Effort
RPF	Whole application	None
Spectra	Programmer-specified	Identify components
Chroma	Programmer-specified	Identify components
MAUI	Method	Annotate remotable methods
CloneCloud	Method	None
Odessa	Sprout component	None, but must use Sprout *

*Sprout is a stream processing system. Sprout models applications as a data flow graph in which the vertices are computational components called stages; Odessa dynamically decides where stages should be placed and when to employ parallel computation.

Candidate Partitions – programmer effort

- Ask a programmer to specify the ways of partitioning an application (Chroma):
 - this approach is based on the belief that for every application, the number of useful ways of splitting the application for remote execution is small
- The number of candidate partitions is small, and the granularity of partitioning is typically large application components:
 - e.g., in a natural language translator: four components for partitioning (three translation engines and a language modeler) and seven candidate partitions among those four components
 - a language (e.g. called Vivendi) allows developers to specify a compact static partition of all meaningful partitions of an applications
 - an application developer specifies code components called **remoteops** that may benefit from remote execution given the right circumstances
 - the developer also specifies **tactics**, each of which is a different way of combining remote procedure calls to produce a remoteop result
 - at runtime, the Chroma system selects one of the specified tactics
- Many other systems follow this approach:
 - e.g., Spectra asks the developer to specify candidate partitions explicitly
 - RPF uses a more extreme version of this approach in which it considers only two candidate partitions: one in which the core computation is performed on the mobile computer and one in which the computation is performed remotely

Candidate Partitions – no programmer effort

- The other extreme is to choose candidate partitions at method granularity without requiring **any programmer assistance**:
 - this approach takes advantage of modern language runtimes to **identify application components** that can be remotely executed and the data on which those components operate
- CloneCloud offers the best example of this approach:
 - it **identifies candidate partitions through static analysis** of code compiled to run in Android's Dalvik virtual machine
 - it **prunes the set of candidate partitions** by considering only those partitions that start and end at VM-layer method boundaries
 - it imposes **three additional restrictions**:
 - **methods that access platform-specific features** (such as the GPS device on a mobile phone) must execute on the platform that has those features
 - **methods that access the same native state** (e.g., a temporary file) must execute on the same computer
 - **nested migration is disallowed** — if a method is migrated from the mobile computer to remote infrastructure, the methods it invokes must also execute remotely
 - while one can imagine removing each of these restrictions (e.g., by supporting remote service invocation or migration of native state):
 - the **restrictions support the common case** while substantially **reducing the set of candidate partitions** that CloneCloud needs to consider

Candidate Partitions – method granularity

- Other systems use a **hybrid** of these two approaches
- E.g. MAUI:
 - it also considers **candidate partitions specified at method granularity**
 - however, it **asks** developers to specify which methods it should consider executing remotely by annotating those methods with the custom attribute feature of the Microsoft .NET Common Language Runtime
- This requires that the developer perform some of the checks that are done automatically by CloneCloud (for example):
 - **verifying that the method does not access functionality specific** to the mobile platform,
 - it also **allows the developer to pass along domain-specific knowledge** (for instance, that it seldom is a good idea to execute user interface methods remotely)

Candidate Partitions – component granularity

- Language runtimes represent only one way for a cyber foraging system to automatically extract application components
- If an **application has already been divided into components** to use distributed systems middleware:
 - the cyberforaging system can **leverage the same component structure** to identify candidate partitions
- E.g., Odessa targets applications that have already been modified to use the Sprout distributed stream processing system:
 - Sprout models applications as a **data flow graph** in which the vertices are computational components called stages
 - Odessa dynamically decides **where stages should be placed** and **when to employ parallel computation**

Partitioning Metrics

**How do we know we picked
the right partition?**

Partitioning Goals - maximize one metric

- There is some metric to be maximized:
 - **there is no consensus among cyber foraging systems about which of these metrics to employ**
 - most systems only consider a subset of them when deciding where to place application components

Cyber Foraging System	Goals	Method of Relating Goals
RPF [Rudenko et al., 1998]	Energy	N/A
Spectra [Flinn et al., 2002]	Execution time, energy, and fidelity	Utility function and goal-directed adaptation
Chroma [Balan et al., 2003]	Execution time and fidelity	Utility function
MAUI [Cuervo et al., 2010]	Execution time and energy	Optimize energy subject to execution time constraint
CloneCloud [Chun et al., 2011]	Execution time or energy	N/A
Odessa [Ra et al., 2011]	Makespan and throughput	Only execute remotely if both goals improve

Partitioning Goals – relating metrics

- One method is to:
 - **focus on one metric that must respect some constraint**
 - e.g. minimize energy usage on the mobile computer subject to the constraint that performance is no worse than 105% of the performance that would be achieved by executing the activity entirely on the mobile device
- Another method is to:
 - execute functionality **only if all performance metrics indicate that remote execution is favorable**
 - e.g. executes application components on remote infrastructure only if it expects to improve both throughput and time taken to finish all processing as a result
- A third possible method:
 - defines **conversion factors between metrics**
 - e.g. assign equal importance to fidelity and execution time
- Each of the above methods represents a **type of policy** for relating metrics expressed in **different currencies**:
 - none of the above take into account **user preferences** and variable goals

Partitioning Goals – user help

- A few methods attempt to solicit user input to help set the policy dynamically (e.g. Spectra):
 - it is based on the observation that the **relative importance of metrics may change** over time
 - it **asks its user to specify a target battery lifetime** for the mobile device at runtime
 - it uses a **feedback process** termed goal-directed adaptation to **dynamically adjust** the relative importance of energy in relation to fidelity and execution time
 - it is based on how well the **rate** at which the mobile computer is consuming power energy matches the target battery lifetime
- Thus:
 - when the mobile computer is projected to **run out of battery energy too soon**, e.g. Spectra **increases the relative importance of energy-efficiency**
 - if the mobile computer is projected to **have substantial energy left over**, e.g. Spectra **decreases the relative importance of energy efficiency**.
- A similar approach could potentially be used to address other budgeted resources for which a fixed maximum consumption is allowed over a time period:
 - e.g., **goal-directed adaptation** has also been used to ensure that a mobile phone₃₁ does not exceed a **monthly cap on a cellular data plan**

Partitioning Goals – off-line

- Aura uses **off-line profiling** to construct utility functions:
 - it allows **users to express thresholds** for satisfaction and starvation for multiple metrics
 - it then **interpolates** among those specifications using sigmoid functions
- In general, these systems illustrate that there is an inherent tradeoff in soliciting user input
- More user participation:
 - it can **improve the quality** of the partitioning, but
 - the participation may **distract the user** from the task at hand and prove **too burdensome**

Selecting a Partition

- Given a set of candidate partitions:
 - a cyber foraging system **selects** the one that it believes will best achieve its goals
- This should be the partition that:
 - **maximizes** the system's chosen metric or utility function while satisfying any constraints
- This is necessarily imperfect because making it correctly requires the cyber foraging system to **predict the future**
- An accurate prediction requires the system to:
 - **estimate** not only the conditions of the mobile computing **environment** (for example, network bandwidth, latency, and server load), but also
 - **application behavior** (e.g. the amount of computation that will be performed by a component for a specific set of inputs)
- Typically, the cyber foraging system assumes that **the past will predict the future**
- It observes application behavior and/or the resources available to the mobile computer and bases future predictions on those observations

Resource Measurement and Estimation

**How can we measure the
cost of offloading
computation?**

Resource Measurement and Estimation

- Resource supply:
 - the resources available in the mobile computer's computational environment
- Resource demand:
 - the requirements of each candidate partition
- Resources that cyber foraging systems most often use for **supply-and-demand estimation**:
 - CPU, network, battery energy, file cache state, and memory
- Cyber foraging systems (e.g. Spectra and Chroma) measure the demand (usage) of CPU for an application:
 - reading these statistics through interfaces such as Linux's /proc file system
- The per-application nature of the statistics:
 - enables the cyber foraging system
 - to distinguish the usage of the target application from other **concurrent activities** on the computer
 - although some **conflicting effects** (such as lock and cache contention) are very difficult to remove

How fast are the CPUs involved?

CPU Measurement and Estimation – first challenge

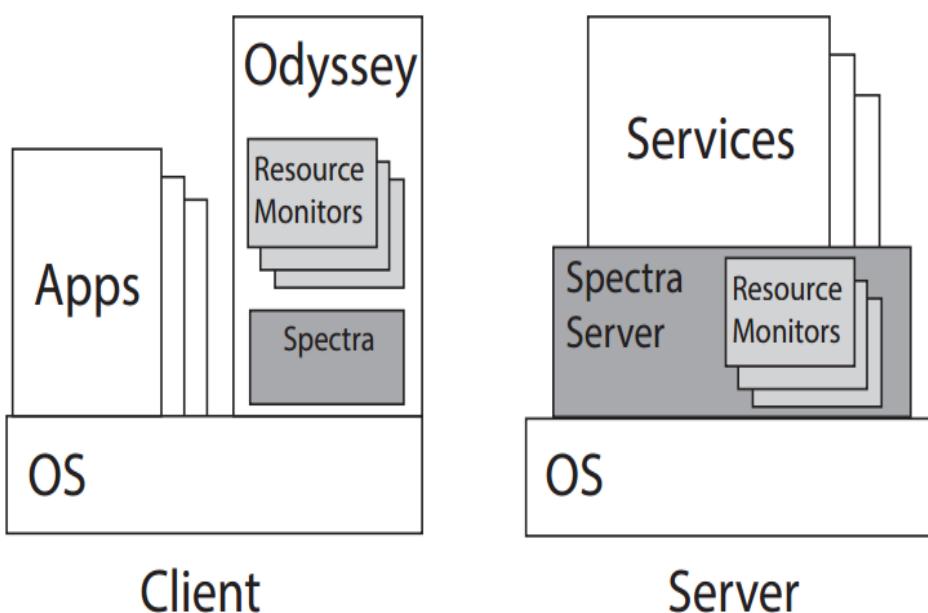
- **First challenge - measuring CPU supply and demand:**
 - such measurements must be performed on both the **mobile computer** and the **remote server(s)** used in cyber foraging
- The **remote server** employed for cyber foraging **may change** depending on such factors:
 - e.g. the location of the mobile computer
- This means that the cyber foraging system must:
 - either **thoroughly measure behavior on all computers** before making an accurate decision, or
 - it must **account for differences in processing power** across computers

CPU Measurement and Estimation – second challenge

- Second challenge - arises from the distributed nature of cyber foraging
- The partitioning **decision is made on one computer** (often, but not always the mobile computer) **using as an input the CPU load of other computers**
- Thus, when a partitioning decision arises:
 - the cyber foraging system could potentially query all other computers that might participate in the computation for their current CPU load
- Unfortunately, this involves a network round trip:
 - if the **computation being considered is relatively small**, the extra time to perform the query of remote state could **add an unacceptable overhead** relative to the compute time
 - the opposite is also true: if the **computation is long-running**, then the relative **overhead of a network round-trip** may be insignificant

Spectra

- Spectra uses slightly stale estimates of remote state:
 - it runs **resource monitors** on both the mobile computer and remote servers
 - each monitor is responsible for a **different resource** such as CPU, network, etc
 - resource monitors on the **remote server periodically and asynchronously report the state** of the server to the mobile computer via remote procedure calls
 - to avoid a network round trip, Spectra uses the last received estimate in its partitioning decision, even though this **estimate may be several seconds old**



How fast is the network?

Network Measurement and Estimation (1/3)

- Cyber foraging systems measure network supply (i.e., the available bandwidth and latency) through a variety of strategies:
 - **active** measurement,
 - **passive** measurement, or
 - a combination of the two
- Passive measurements observe traffic sent over the network for other purposes:
 - they **do not inject additional traffic**
 - this **conserves both battery energy and cellular network usage**
 - however, **passive measurements may become stale** if the mobile computer does not send or receive data for a time
- Active measurements:
 - **inject traffic** into the network solely for the purpose of measuring the network performance
 - provide more recent data

Network Measurement and Estimation (2/3)

- Spectra uses passive measurements:
 - its network monitor **predicts bandwidth and latency**
 - its RPC package **logs the sizes and elapsed time of short exchanges and bulk transfers**
 - the **small RPCs are used to predict latency**, and the **larger RPCs are used to predict throughput**
 - it periodically **examines the total bandwidth available** to the mobile computer
 - it then **estimates the bandwidth** likely to be available for communicating with each server
- Passive measurement of network demand (bytes transmitted) is accomplished in a similar manner:
 - Spectra simply **queries the RPC package to determine the bytes required to transfer state** for each RPC performed

Network Measurement and Estimation 1/3

- MAUI:
 - uses **both active and passive** measurements
 - it employs the simple strategy of **sending 10 KB of data to the remote server and measuring the throughput** of the transfer directly to obtain an average throughput
 - this works well because the **10 KB size is representative of the typical transfers** to the MAUI server
- MAUI also uses passive estimation to refine its active measurements:
 - **every time that a method is offloaded**, MAUI obtains a **more recent observation of network quality** by measuring the duration of the transfer
 - **active measurements are only employed if no transfers have provided recent passive estimates**
 - if more than a minute passes without a transfer, MAUI sends 10 KB to the server and observes the result

**What's the impact of
offloading on the battery?**

Battery Measurement and Estimation

- The **supply of battery energy** is one of the **simplest resource values to measure and estimate**:
 - the supply of battery energy is just the amount of charge remaining in the battery
 - most mobile computers provide standard interfaces for querying this value
- The **demand for battery energy** is one of the **most difficult**:
 - it is the amount of batter energy consumed by a particular computation
- Cyber foraging systems typically use one of two methods to estimate this value:
 - direct measurement (e.g. Spectra and Chroma)
 - model-based approach

Battery Measurement and Estimation - direct

- Direct measurement (e.g. Spectra and Chroma)
 - **measure** the amount of energy in the mobile computer's battery **before an operation begins** and **after the operation completes**
 - the difference between the two values is the amount of energy required to perform the computation
- Direct measurement has two drawbacks:
 - if **more than one activity occurs simultaneously**, it is difficult to separate the energy cost of each activity
 - Spectra adjusts for this issue by discarding all energy demand measurements in which operations executed concurrently.
 - many mobile computers **do not provide fine-grained measurements** of battery capacity
- Report battery energy at **granularities as coarse as 1% of total battery capacity** is insufficient to measure brief computational events

Battery Measurement and Estimation - model

- Many cyber foraging systems use a **model-based approach** to estimate **energy demand**:
 - executes a series of **micro benchmarks on the mobile computer in a laboratory setting**
 - while an **external power measurement device measures how much energy** the computer expends executing those benchmarks
 - an **energy model is constructed** using the results of these experiments
- The model assigns specific energy costs to measurable activities performed by the mobile computer:
 - e.g. **executing** a given amount of computation, **sending** or **receiving** a byte of data over a wireless network, or **enabling** the screen backlight

Battery Measurement and Estimation – model

- Such models are naturally **specific** to a particular brand and model of a mobile device
- Given such an energy model a cyber foraging system can:
 - measure the occurrence of events such as computation and network transmission that occur during application execution,
 - then use the model to assign an energy costs to those events
- The end result is an estimate of the energy used to perform the activity

Battery Measurement and Estimation - model

- Drawback of using energy models:
 - they necessarily **do not capture all the potential factors** that may influence mobile computer energy usage
- E.g.:
 - a model might capture the **average energy consumed per CPU cycle of computation**
- However:
 - **different instructions consume different amounts of energy**, so the mix of instructions can affect the total energy consumed by an operation
 - the **amount of memory accesses per instruction** substantially changes the energy expenditure
 - finally, **processor power management** will substantially alter energy usage

Applications - examples

- RPF executes **compiling tasks**
- Spectra runs a **speech recognizer, document preparation tools, and a natural language translator**
- Chroma supports **face recognition, text-to-speech synthesis, image filtering, 3D model viewing, speech recognition, music identification, natural language translation, lighting for 3D models, and creation of 3D scenes from 2D images**
- MAUI executes a **face recognizer, a video game, and a chess game**
- CloneCloud runs a **virus scanner, an image search application, and privacy-preserving targeted advertising**
- MARS targets three **computationally intensive applications: augmented reality pattern recognition, face recognition, and an augmented reality video game**
- Odessa supports **face recognition, object and pose recognition, and gesture recognition**

Applications – best tasks

- The tasks best suited for cyber foraging are **compute-intensive**
- Such tasks **benefit most from remote execution**, and they therefore **offer the best opportunity to recoup the costs of sending state over the wireless network**
- Cyber foraging is **not necessarily limited to batch tasks** like compiling and document processing:
 - most of the applications listed previously are interactive
 - however, **all have significant compute chunks** that **occur between interactions with the user**
 - the **length of such compute chunks must be substantial compared to the latency of the wireless network**, unless the computation can be performed asynchronously

Applications - multimedia

- Most of the recent applications that leverage cyber foraging involve **multimedia**:
 - either through **intense audio processing** (e.g., speech recognition and natural language translation), image processing (e.g., face recognition), or
 - **support for video** (e.g., games)
- Thus, the **trend to incorporate more multimedia** in everyday applications may **increase the opportunities to employ cyber foraging** in future applications
- Many of the applications listed above are especially relevant to mobile computer users:
 - applications such as **natural language translation** have clear applications for visitors traveling through a foreign country
 - **face recognition and music identification** are already used in popular mobile applications
- Thus, the future seems promising for cyber foraging:
 - **such applications demand capabilities that mobile computers may simply be unable to provide** with purely local resources while running with a limited battery capacity

Management

Management Issues

- Location and management of servers that host remote operations during cyber foraging
- We refer to a remote computer that may temporarily assist a mobile computer as a surrogate
- Where should surrogates be **located**?
- How should remote operations on a surrogate be **isolated**?
- How should surrogates acquire and manage the **state** associated with cyber foraging?

Location Decision Factors

- Factors that impact the location decision, including:
 - network latency and bandwidth, ease-of-management, cost, and privacy concerns
- Options for location:
 - deploying surrogates as close to the edge of the network as possible,
 - use of dedicated computers in a data centre, and
 - opportunistic use of computers in a data centre
- How any of these options may be best, depending on which of the factors listed above are considered the most important

Location Examples

- Network latency has a similar effect on energy usage:
 - the mobile computer expends power while waiting for the surrogate to return a result
- In the limit, it may not be feasible to offload any part of an application if a surrogate is located too far away
- Examples:
 - for a chess game, offloading methods to a surrogate improves performance only if the surrogate is located nearby — with a network round-trip time (RTT) of 25 ms or less
 - for face recognition and a video game, the performance difference between using a nearby (10 ms RTT) and a distant (220 ms RTT) surrogate can be as large as 50%
 - w.r.t. energy usage, the two applications (chess and the video game) consumed more energy when the mobile computer offloaded methods over 3G (incurring a 220 ms RTT) than if the method had not been offloaded,
 - whereas WiFi transmission, which led to lower round-trip times, enabled offloading to decrease the energy usage of a cell phone

Surrogates in the Cloud - benefits

- Ease of management:
 - much easier to replace a hardware component that has failed in a centralized data centre than it is to make a service call to a remote location to replace the same component
 - physical proximity may make it easier to troubleshoot software problems and repair mis-configurations
 - it is likely that network connectivity, power, and other external inputs to the surrogate are more reliable in a data centre than in a coffee shop or airport lounge
- Easier to provide physical security:
 - if surrogates are deployed in public locations, then additional measures, may be required to prevent tampering

Conclusions

- It's a hot topic and a research concept which has already practical use cases (mostly with static partitioning)
- Cyber foraging stands at the confluence of two strong trends in modern community:
 - every day more people access data and run computation from their mobile computers, and
 - data and computation is increasingly distributed among those devices and cloud data centers
- Cyber foraging is especially compelling in a post-PC world in which desktop and workstation computers are replaced by mobile and cloud devices
- The large variation in mobile computing environments argues that functionality should be dynamically, rather than statically, partitioned among mobile and cloud computers
- The limitations of locating computation in both cloud data centers (e.g., communication latency) and on mobile devices (e.g., limited resources):
 - create the need to insert a surrogate computational tier between these extremes that can mitigate both sets of limitations

Mobile and Ubiquitous Computing

2022/23

MEIC/METI - Alameda & Tagus

Augmented Reality - An Overview

Motivation

- Applications and hardware provide more and more immersive experiences.
- The physical and digital UI is a nuisance on the side.
- Reality is competing for our attention.

What is Augmented Reality?

- Extending visual reality with additional integrated elements.
- Creating a new UX environment without losing the context of reality.
- Integrating the UI in the environment.

History

- **1968:** Ivan Sutherland creates the first augmented reality system: optical see-through display with simple wireframe drawings.
- **1992:** term “augmented reality” coined.
- **1997:** augmented reality defined: real-virtual mix, real time, 3D.

History

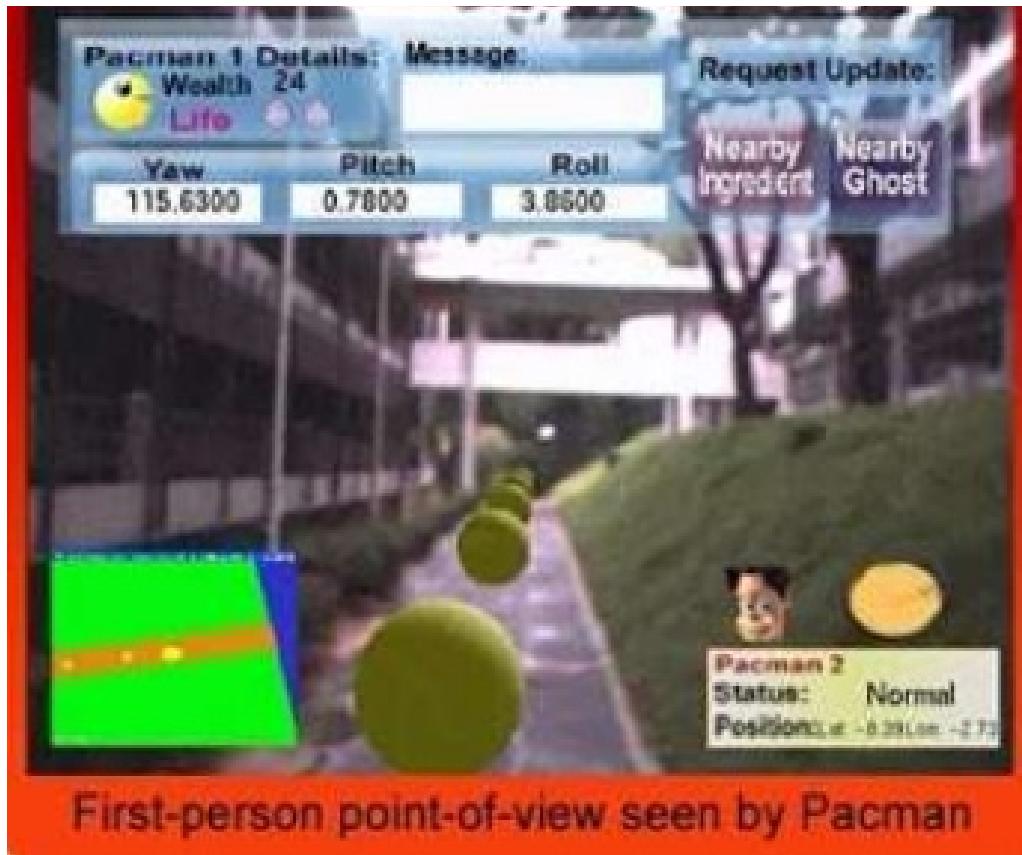
- 1997: Steve Feiner's Touring machine



- 2000: First commercial phone camera

History

- 2003: Human Pac-Man



History

- 2008: First museum AR guide



History

- 2013: Google launches Google Glass



Characteristics of Mobile AR Apps

- **Input:** Uses various device and companion device sensors, e.g. camera, gyroscope, microphone, GPS.
- **Processing:** Determines information that is going to render in the screen of the mobile device.
- **Output:** Projects its output to the screen of the mobile device together with the current view of the user (i.e. augments the user's reality).

Application Domains

- ▶ Tourism & Navigation: historical and practical information about new locations.
- ▶ Training & Education: complement reality with instructions, tips, collaboration hints. (see <https://www.youtube.com/watch?v=-DYqlaMWTVg>)
- ▶ Assembly & Maintenance: give hints on how to assemble a machine or what to observe, verify in maintenance procedures. (see <https://www.youtube.com/watch?v=Y5ywMb6SeGc>)

Application Domains

- Entertainment & Advertisement: in games extend the setting, the scenario or user costumes. Add ads to the environment or to a game.



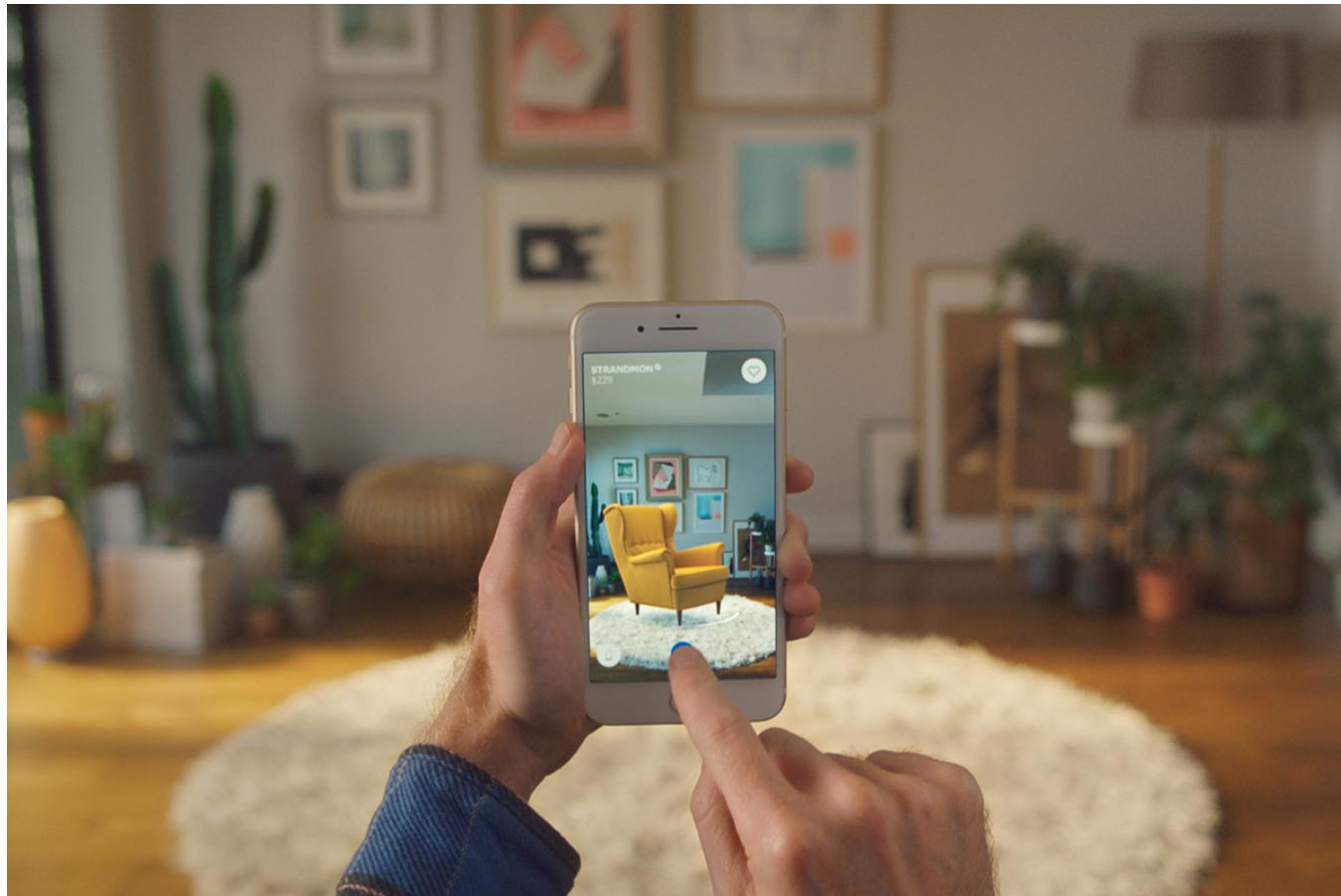
ABSOLUT TRUTHS EXPERIENCE
Explore the unnecessary lengths we go to

FLIP TAG PRINT

A screenshot of the Absolut Truths Experience website. The main title is "ABSOLUT TRUTHS EXPERIENCE" with the tagline "Explore the unnecessary lengths we go to". Below the title is a large image of a black Absolut Vodka bottle inside a white square frame. To the left of the image is a "FLIP TAG" button, and to the right is a "PRINT" button. At the bottom, there is a section titled "THE ABSOLUT TRUTHS AUGMENTED REALITY EXPERIENCE" with a note about the required iPhone app and tag.

Application Domains

- Scene Construction: for example, add items to an environment, e.g. furniture.



Application Domains

- Information Assistants: show related information to the observed environment, e.g. related species near plants, x-rays over the patient, building overlays.



Components: Display

- ▶ Visual see-through:
 - ▶ Blocks light
 - ▶ Contrast is difficult to tune.
- ▶ Video see-through
- ▶ Projected



Tracking and Registration

- ▶ Tracking and registration: identifying the location and position of the device to decide what to display.
- ▶ Can be Sensor-based or Vision-based.
- ▶ Sensor based tracking was covered in the “Location” section of the course.

Vision-based Tracking

- Vision-based tracking can be marker-based or feature-based.
- Markers are precise but impractical outdoors. Markers can be pre-input images, colored stickers or QR-codes.
- Feature tracking attempt to match DB real-world images to the camera image and then track them. It's generic but error-prone and compute (and energy) intensive.

Tracking: Sensor-Vision Hybrids

- Hybrid solutions explore the best of sensors and vision.
- Accelerometers deal well with brief sudden movements but drift in the long term.
- GPS are coarse but good for initial positioning.
- Image tracking is expensive and fragile (lighting, camera occlusion) but detailed.

How to match images to locations?

- Data acquisition: difficult to automate the conversion of acquired data to location databases. Can be manual, using topographical tools or the aid of existing DBs.
- One approach, SLAM:
 - Simultaneous Location and Mapping
 - Navigate an unknown space
 - Create a model
 - Place the device in it
 - The more previous knowledge (e.g. street view) the easier it becomes.
 - Area of extensive current research

SLAM

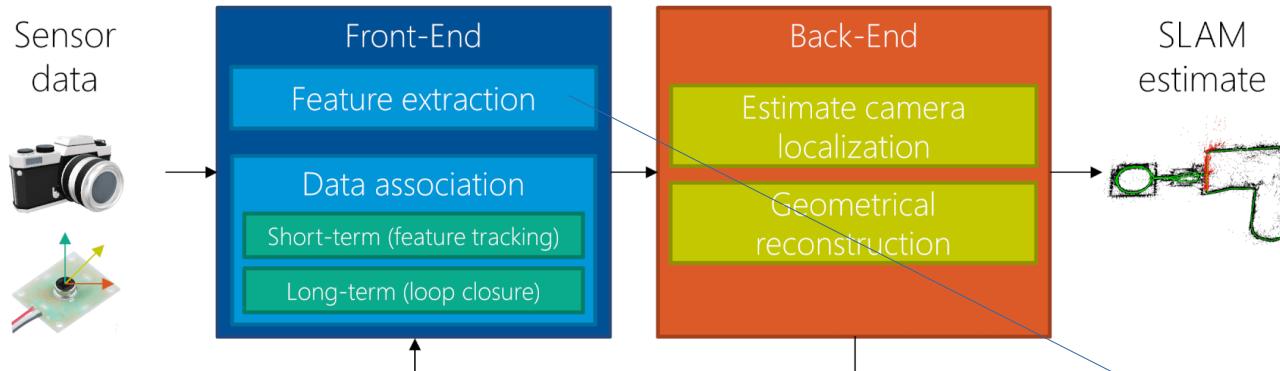
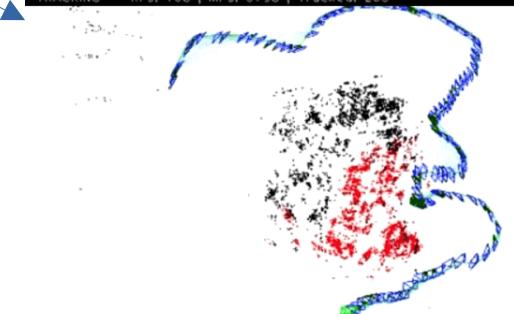


Diagram based on: Cadena, Cesar, et al. "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age." *IEEE Transactions on Robotics* 32.6 (2016): 1309-1332.
SLAM estimate: R. Mur-Artal, J. M. M. Montiel and J. D. Tardós, "ORB-SLAM: A Versatile and Accurate Monocular SLAM System," in *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147-1163, Oct. 2015.
3D Model of motion sensor by gerduleb: <https://www.remix3d.com/details/G009SVNP5RSM>
3D Model of camera by Microsoft: <https://www.remix3d.com/details/G009SXQ93TH>



Components: SDK

- ▶ Most established, mostly providing tracking and SLAM:
 - ▶ Wikitude
 - ▶ Vuforia
 - ▶ Apple ARKit
 - ▶ Android AR Core

Challenges for AR

- “Dangerous” mix of power drains: GPS + camera + intensive CPU/GPU use (scene recognition, object orientation and placement).
- Energy poor devices are the most interesting for AR.
- Great candidate for computational offloading. However, video stream processing is bandwidth intensive and may require preprocessing to reduce the stream’s bandwidth.
- Privacy issues of constant video acquisition.

References

- **Mobile augmented reality survey: From where we are to where we go.**
Chatzopoulos, Dimitris and Bermejo, Carlos and Huang, Zhanpeng and Hui, Pan. IEEE Access, v. 5, pp. 6917-6950. 2017
- The history of mobile augmented reality. C Arth, R Grasset, L Gruber, T Langlotz.
arxiv.org. 2015.
<https://arxiv.org/pdf/1505.01319>