EPFL

CS-311: The Mobile Platform

Prof. George Candea
School of Computer & Communication Sciences

Mobile Platform

The ingredients that come together to enable the creation, distribution, and operation of mobile content and services

Operating Systems

Devices

- smartphones, tablets
- wearables (smartwatches, fitness trackers)
- wellness (health monitors, glucose meters)
- smart-home devices
- automotive (connected cars)
- ...

Infrastructure & Services

- app distribution (play/app stores)
- push notification
- authentication & identity services
- content delivery networks
- cloud computing platforms
- find-my-device, MDM
- 5G, WiFi
- ...

App Dev Frameworks

Outline

- Mobile Devices
- Mobile Operating Systems
- Mobile Infrastructure & Services
- Mobile Applications

Mobile Devices

The Mobile SoC

- CPU (up to 16 cores!): general-purpose app and OS logic
- GPU: graphics rendering, gaming, UI
- NPU (neural processing unit): AI/ML tasks
- ISP (image signal processor): camera processing
- DSP (digital signal processor): sensor data processing, audio
- Modem (5G/LTE): cellular connectivity
- HSM (hardware security module): crypto, biometrics data protection
- Memory controller, video encode/decode, display engine, etc.

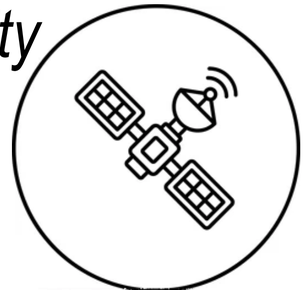
Device Architecture: Data

- Memory (RAM)
  - *more RAM → more apps can be active simultaneously without slowdown*
  - *mobile app lifecycle and memory mgmt is different from desktop apps*
  - *high-end phones can have as much as 24GB*
- Storage
  - *Internal storage holds the OS, installed apps, and user data*
  - *NAND flash with UFS*
  - *high-end phones can have as much as 2TB*

https://www.ghacks.net/wp-content/uploads/2023/03/MWC-2023-Nokia-unveils-an-affordable-new-Android-smartphone-2.png

Device Architecture: Battery & Sensors

- Battery
  - *Capacity: 1'000 mAh to 10'000 mAh*
  - *energy consumed depends on usage, device's power efficiency, screen, processors*
  - *fast charging and wireless charging add convenience*
- Sensors
  - *Accelerometers, gyroscopes, proximity sensors, etc.*
  - *Ambient light sensors adjust screen brightness*
  - *Fingerprint readers, facial recognition, and other biometrics enhance security*
- Satellite connectivity



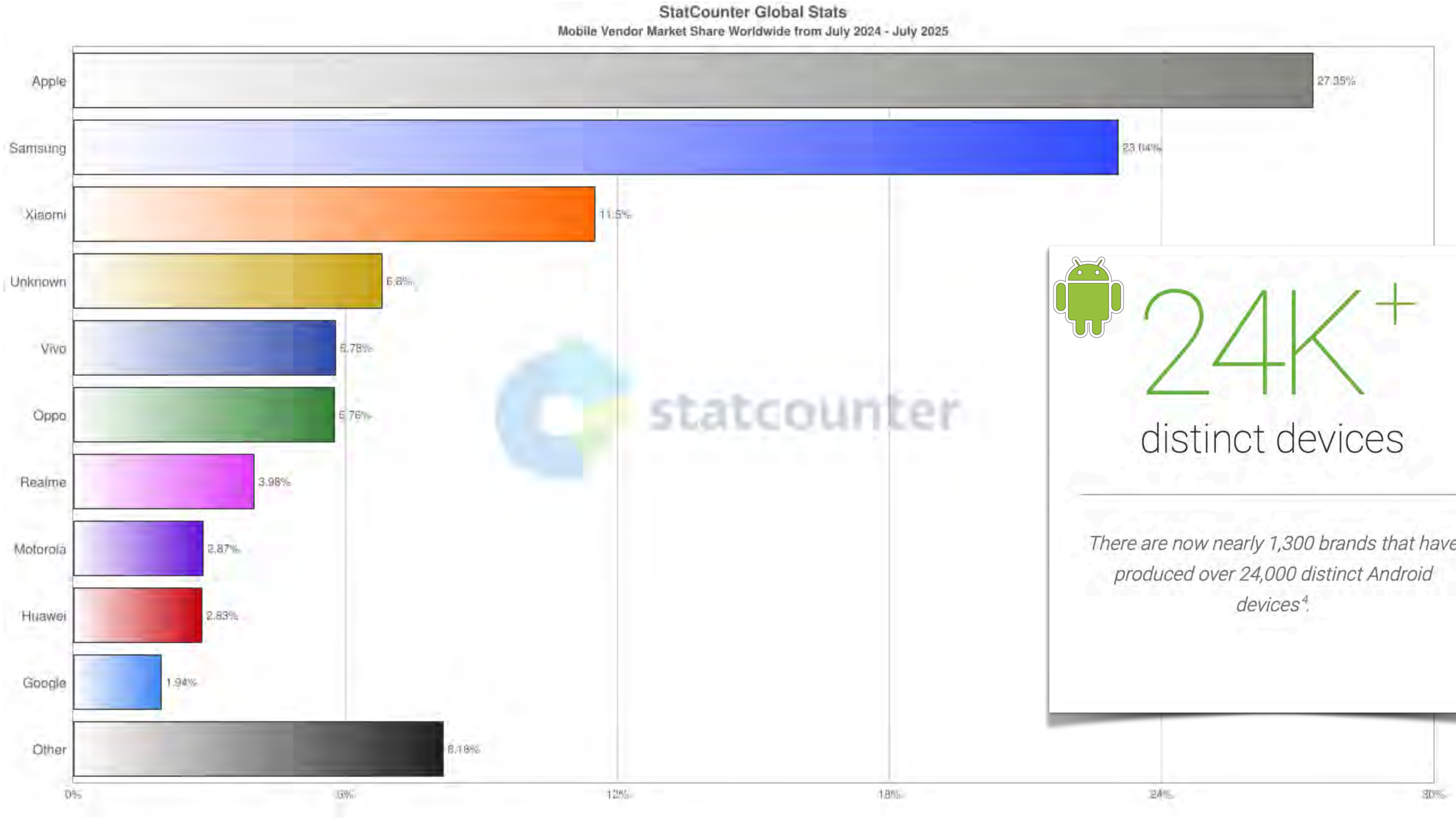
https://www.theverge.com/2021/1/11/22223494/qualcomm-ultrasonic-fingerprint-reader-samsung-galaxy-s21-announcement https://mybroadband.co.za/news/smartphones/500497-smartphones-with-removable-batteries-could-make-a-comeback.html

Device Architecture: Interaction with the Outside

- Display
  - *Display sizes: 4" to 6.9" for phones, bigger for tablets*
  - *Pixel density: 180 to 800 PPI for currently selling phones*
  - *Resolution: 480x854 to 1644x3840 (4K)*
  - *LTPO (Low Temperature Polycrystalline Oxide) → adaptive refresh rates 1 - 120Hz*

https://static1.srcdn.com/wordpress/wp-content/uploads/2023/01/samsung-galaxy-s-22-ultra-camera.jpg?q=50&fit=crop&w=825&dpr=1.5

- Cameras
  - *Rear cameras: wide-angle, telephoto, ultra-wide, and macro lenses*
  - *Megapixels: 2MP to 200MP*
  - *Computational photography advances, periscope zoom lenses up to 10x optical*
- Connectivity
  - *GPS, Cellular (5G), WiFi, Bluetooth, NFC*



24K+

distinct devices

There are now nearly 1,300 brands that have produced over 24,000 distinct Android devices[4].

Outline

- Mobile Devices
- Mobile Operating Systems
- Mobile Infrastructure & Services
- Mobile Applications

Mobile OS vs. Desktop/Server OS

User Model and Security

Feature	Mobile OS	Server OS
User model	Single-user	Multi-user with roles/permissions
App/process isolation	App sandboxing	Process-level isolation (UIDs/GIDs)
Permission system	Explicit runtime permissions	Access control via file and system permissions
Admin/root access	Not available to user	Sudo/admin access available
Trust model	Distrusts apps	Distrusts users

Power Management

Feature	Mobile OS	Server OS
Power	Battery-powered, energy-constrained	Always-on power
Background processing	Throttled/deferred	Unrestricted (daemons, services)
Thermal limits	Aggressively managed	Passive or active cooling, less constrained

Execution Model & App Lifecycle

Feature	Mobile OS	Server OS
Process lifecycle	OS can kill/restart apps anytime	Processes expected to run indefinitely
Lifecycle control	App lifecycle events (onPause, etc.)	None; processes control their own lifetime
Scheduling priority	UI threads get CPU priority	Throughput-optimized scheduling

Networking & Connectivity

Feature	Mobile OS	Server OS
Network stability	Intermittent or mobile	Stable, high-bandwidth
Network types	Cellular, WiFi, Bluetooth, NFC	Ethernet, bonded interfaces, high-speed
Network policy	Battery-sensitive, may block traffic	No such constraints

UI & Input/Output

Feature	Mobile OS	Server OS
User interaction	Touch, gestures, voice, sensors	None (headless); CLI or remote GUI
UI importance	Must be real-time and smooth	Not applicable
Output interfaces	Small screens, haptics, speakers	Logs, stdout, APIs, remote outputs

Storage & Filesystem

Feature	Mobile OS	Server OS
Storage type	Flash (UFS), constrained size	Large-scale SSD/HDD, RAID
File access	Per-app private storage	Shared/global filesystem
Access control	App permission-based	User/group permissions

Hardware Environment

Feature	Mobile OS	Server OS
Architecture diversity	Extreme (ARM SoCs, sensors, NPUs)	Standardized (x86_64 or server-grade ARM)
Input/output diversity	Cameras, touchscreens, GPS, accelerometers	Basic I/O, remote-managed
Integration level	Highly integrated (SoC = CPU+GPU+NPU)	Modular hardware, discrete components

Software Updates & Maintenance

Feature	Mobile OS	Server OS
Update mechanism	OTA updates via OEM/carrier	Admin- or automation-driven updates
Control over updates	Limited (esp. on Android)	Full control (scripts, policies)
Update timing	Varies by vendor/device	Regular, scheduled maintenance

Android vs. iOS

- open-source, highly customizable
- tens of thousands of devices
- variety of UI skins
  - *Samsung One UI, Xiaomi MIUI, ...*
- ("everyone", i.e., all possible users)
- based on Linux

- closed, controlled ecosystem
- only iPhones and iPads (and a few others)
- uniform and consistent user experience
- (high-revenue users)
- based on XNU

ANDROID

Kotlin / Java source

↓

Java bytecode (.class files)

↓

DEX bytecode (.dex files)

↓

Distributed to device

↓

Compiled to device CPU at install time

(ARMv7, ARM64, x86, x86-64, RISC-V, ...)

- ART managed runtime
- JIT compilation & profile-guided optimization

Swift / Objective-C source

↓

LLVM IR

↓

ARM64 native machine code

↓

Distributed to device

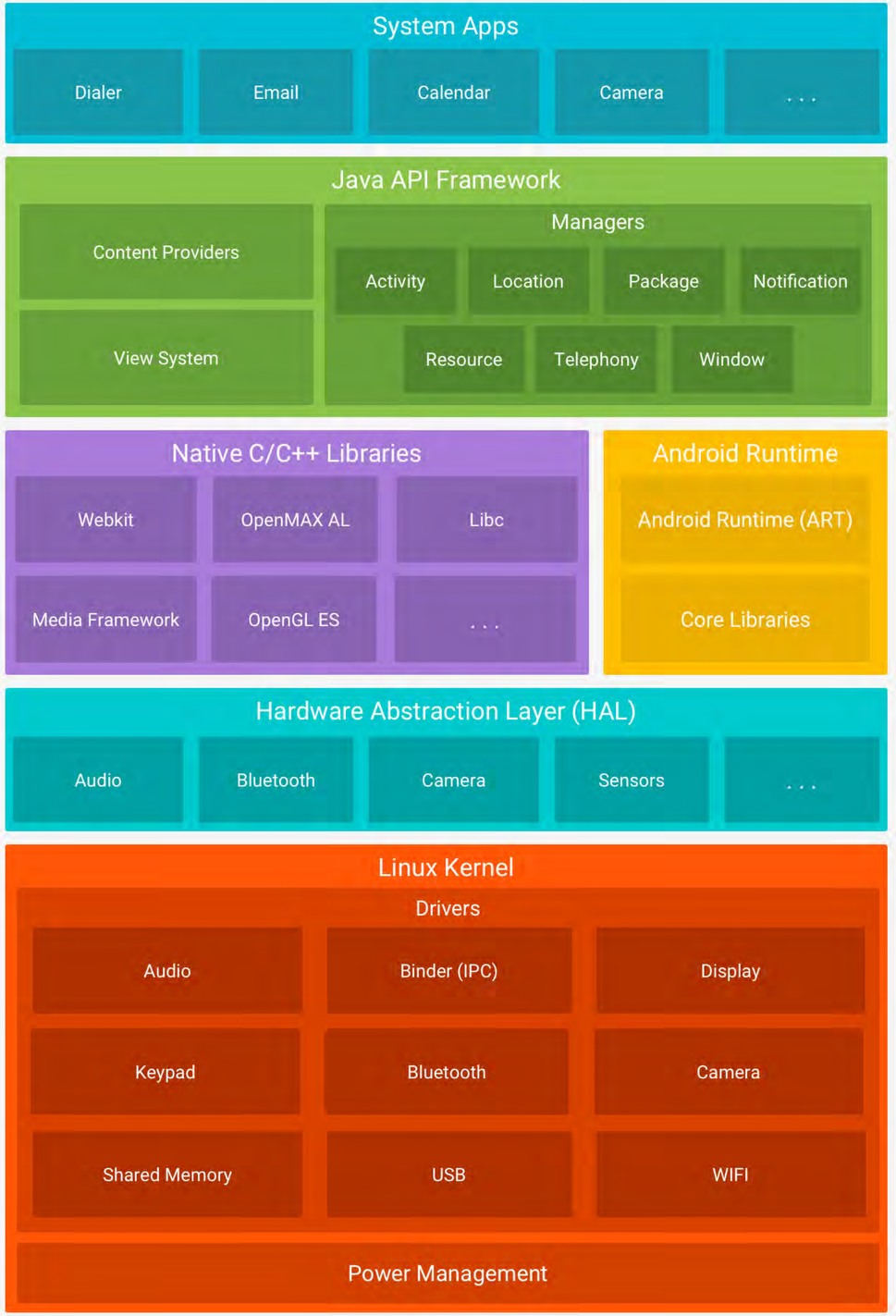
↓

Direct execution as native binary

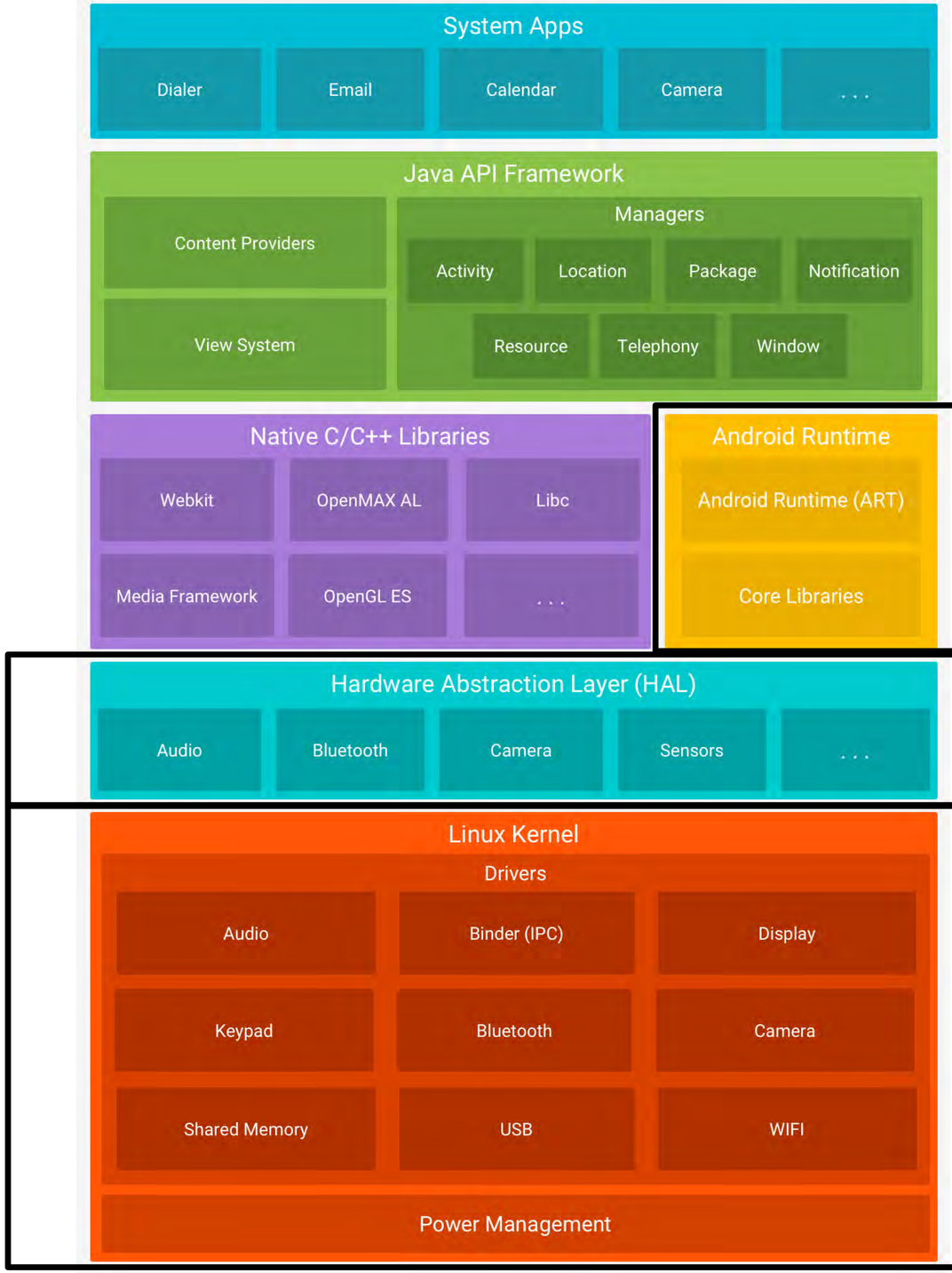
- No managed runtime
- Uses dynamically linked libraries

Mobile OS Internals
(illustrated with Android)

Android Stack



Android Stack



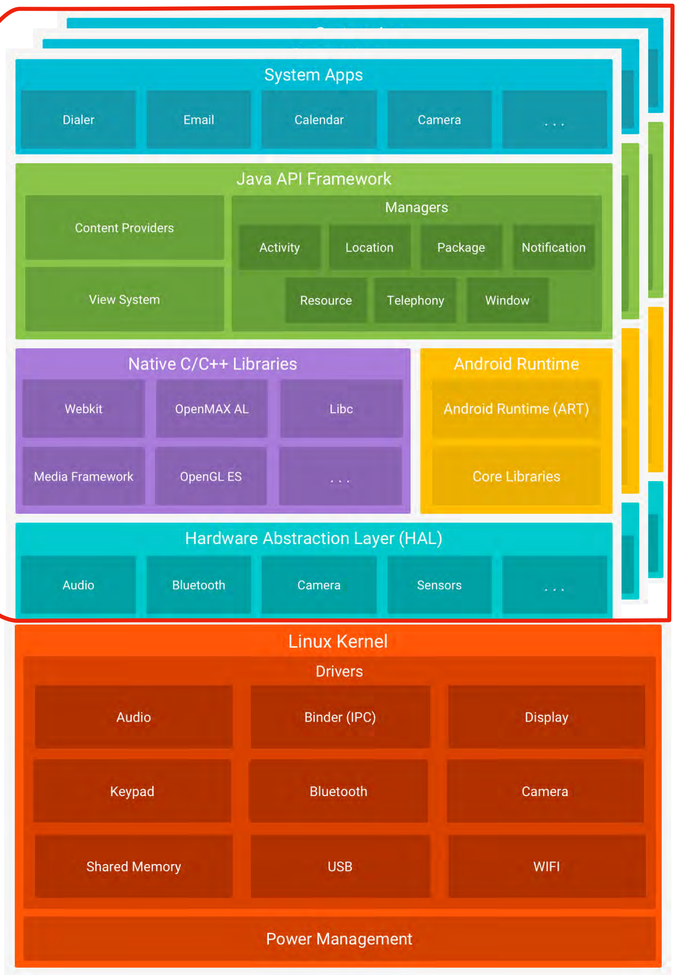
Env. for executing app bytecode

Interface to HW capabilities

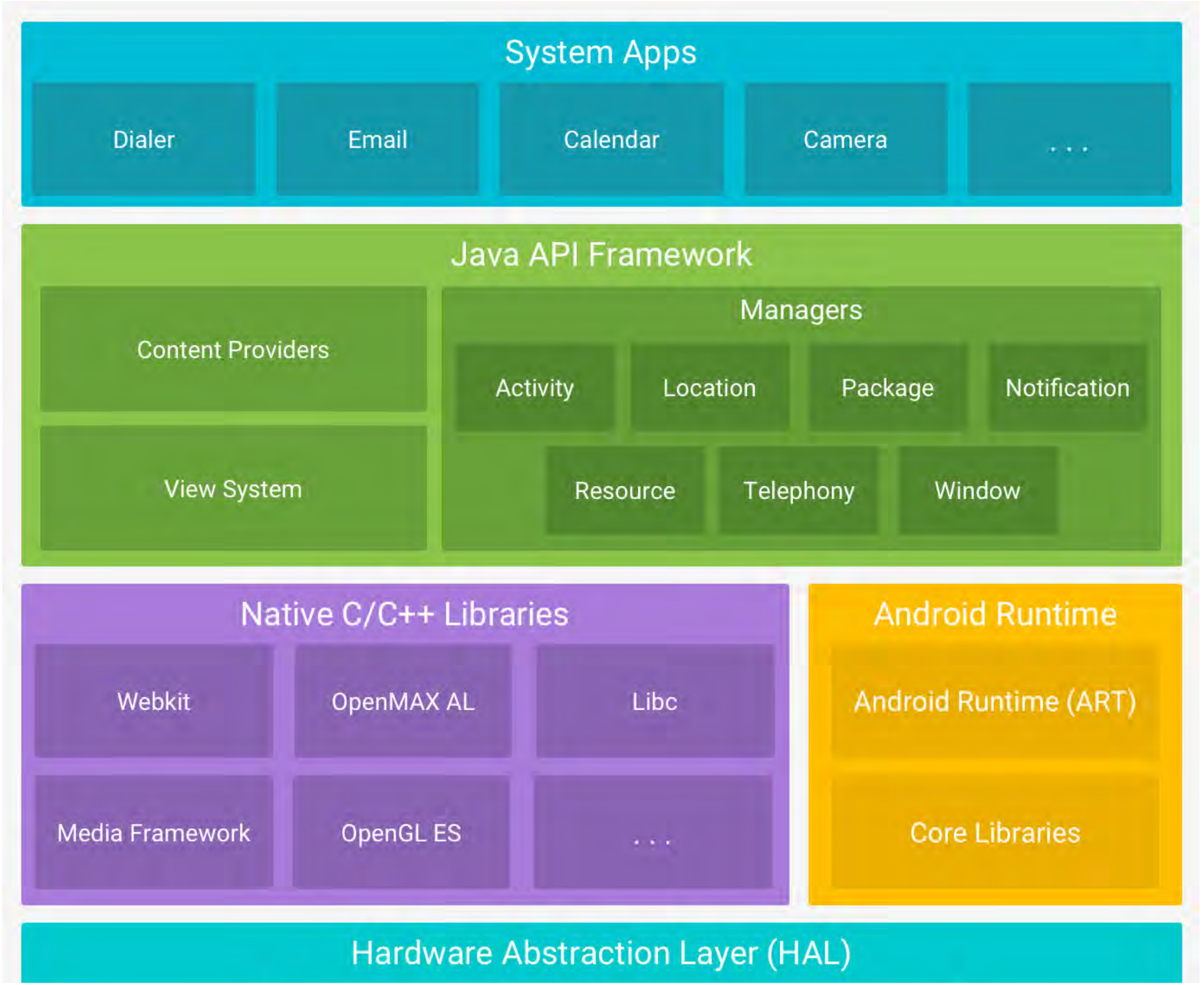
Inner core of the operating system

Apps executes in a per-app security sandbox

- Apps written in Kotlin (or Java, C++)
- Deployment: on device (APK) or in app store (AAB)
- App executes in a per-app security sandbox
  - *1 Linux user per app, 1 Linux process per app*
    => replication of top of Android stack
  - *Follows principle of least privilege*
- Sharing is possible through explicit mechanisms
  - *Apps from same developer can directly share files*
  - *With the user's permission, apps can access device data*
- Priority-based scheduling + process containers (cgroups)



Top of Android Stack



App Data (Persistence Levels)

Service	Usage examples	Lifespan	Implementation
SharedPreferences	user prefs (dark mode, toggles), simple game high scores	Until app is uninstalled or data is cleared	Key–value pairs, lightweight
SQLite / Jetpack Room	draft emails, chat history, notes, complex leaderboards	Until app is uninstalled or data is cleared	Structured relational DB
Internal storage (files)	app-specific documents, private media, email attachments	Until app is uninstalled or data is cleared	Private to the app
External storage (files)	shared media (photos, audio, video), downloads	Until user deletes it or external storage is wiped	Accessible to other apps (with permission)
Cache directories	cached images, temporary API responses	May be cleared anytime by OS or user	For temporary but re-creatable data
Content providers	contacts, calendar events, media store entries	Until user deletes (system-level state)	Lets apps read/write shared system data
Cloud / backend services	synced drafts, saved games, messaging data	Persists across devices, as long as cloud account exists	Not local; depends on server availability

Outline

- Mobile Devices
- Mobile Operating Systems
- Mobile Infrastructure & Services
- Mobile Applications

Mobile Infrastructure & Services

Split-App Model

- Client-side (frontend)
  - *runs on the user's device, provides the UI and UX*
- Server-side (backend)
  - *runs on a server or in the cloud*
- Client interacts with cloud services
  - *typically REST APIs or GraphQL*
  - *send HTTP request*
  - *get back JSON or XML*

https://hacker-news.firebaseio.com/v0/newstories.json

[39456086,39456083,39456079,39456069,39456026,
39456015,39455995,39455986,39455982,39455981,3
9455961,39455952,39455939,39455928,39455918,39
455915,39455911,39455903,39455902,39455891,394
55874,39455873,39455839,39455827,39455825,3945
5821,39455810,39455783,39455770,39455750,39455
746,39455745,39455724,39455683,39455682,394556
76,39455633,39455632,39455617,39455607,3945559
3,39455576,39455572,39455553,39455546,39455536
,39455535,39455533,39455522,39455503,39455473,
39455421,39455336,39455319,39455305,39455301,3
9455294,39455275,39455271,39455267,39455265,39
455257,39455230,39455220,39455187,39455185,394
55184,39455177,39455173,39455170,39455167,3945
5146,39455131,39455122,39455108,39455103,39455
102,39455080,39455075,39455057,39455044,394550
33,39455029,39455027,39455022,39454997,3945499
3,39454961,39454959]

https://hacker-news.firebaseio.com/v0/newstories.json

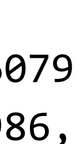
https://hacker-news.firebaseio.com/v0/item/39456069.json

[39456086,39456083,39456079,39456069,39456026,
39456015,39455995,39455986,39455982,39455981,3
9455961,39455952,39455939,39455928,39455918,39
455915,39455911,39455903,39455902,39455891,394
55874,39455873,39455839,39455827,39455825,3945
5821,39455810,39455783,39455770,39455750,39455
746,39455745,39455724,39455683,39455682,394556
76,39455633,39455632,39455617,39455607,3945559
3,39455576,39455572,39455553,39455546,39455536
,39455535,39455533,39455522,39455503,39455473,
39455421,39455336,39455319,39455305,39455301,3
9455294,39455275,39455271,39455267,39455265,39
455257,39455230,39455220,39455187,39455185,394
55184,39455177,39455173,39455170,39455167,3945
5146,39455131,39455122,39455108,39455103,39455
102,39455080,39455075,39455057,39455044,394550
33,39455029,39455027,39455022,39454997,3945499
3,39454961,39454959]

https://hacker-news.firebaseio.com/v0/newstories.json

[39456086,39456083,39456079,39456069,39456026,
39456015,39455995,39455986,39455982,39455981,3
9455961,39455952,39455939,39455928,39455918,39
455915,39455911,39455903,39455902,39455891,394
55874,39455873,39455839,39455827,39455825,3945
5821,39455810,39455783,39455770,39455750,39455
746,39455745,39455724,39455683,39455682,394556
76,39455633,39455632,39455617,39455607,3945559
3,39455576,39455572,39455553,39455546,39455536
,39455535,39455533,39455522,39455503,39455473,
39455421,39455336,39455319,39455305,39455301,3
9455294,39455275,39455271,39455267,39455265,39
455257,39455230,39455220,39455187,39455185,394
55184,39455177,39455173,39455170,39455167,3945
5146,39455131,39455122,39455108,39455103,39455
102,39455080,39455075,39455057,39455044,394550
33,39455029,39455027,39455022,39454997,3945499
3,39454961,39454959]

https://hacker-news.firebaseio.com/v0/item/39456069.json



[{

"by":"tosh",

"descendants":0,

"id":39456069,

"score":2,

"time":1708533765,

"title":"Imperfect Compression",

"type":"story",

"url":"https://mathspp.com/blog/problems/imperfect-compression"

}]

```
query {
  newStories(limit: 10) {
    id
    title
    url
    by
    score
    time
  }
}
```

```
POST /graphql HTTP/1.1
Content-Type: application/json

{
  "query": "query { newStories(limit: 10) { id title url by score time } }"
}
```

Web service

```
{
  "data": {
    "newStories": [
      {
        "id": 39456069,
        "title": "Imperfect Compression",
        "url": "https://mathspp.com/blog/problems/imperfect-compression",
        "by": "tosh",
        "score": 2,
        "time": 1708533765
      },
      ...
    ]
  }
}
```

Benefits of Split-App Model

Modularity + contracts between the frontend and backend ⇒

highly decoupled ⇒

- Independent development
- Scalability
- Flexibility to choose the most suitable technology and frameworks
- Can update app frontend independently of backend
- Reuse backend services across clients (mobile, Web, etc.)

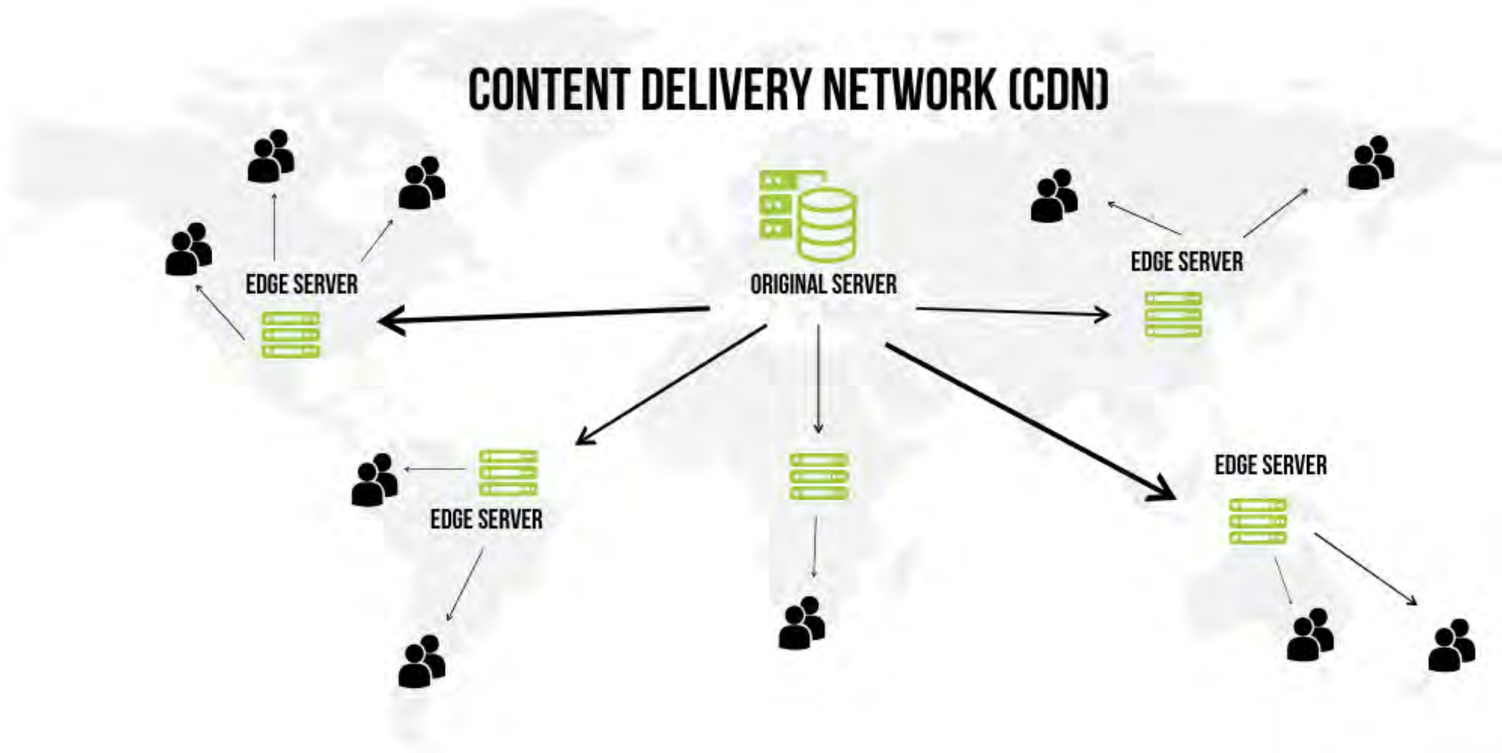
Backend/Network Services

- Content Delivery Networks (CDNs)
- Analytics and reporting
- Email and SMS
- Backup and recovery
- Geo-location
- Integration services
- ...

https://www.ssl.com/article/what-is-a-cdn/

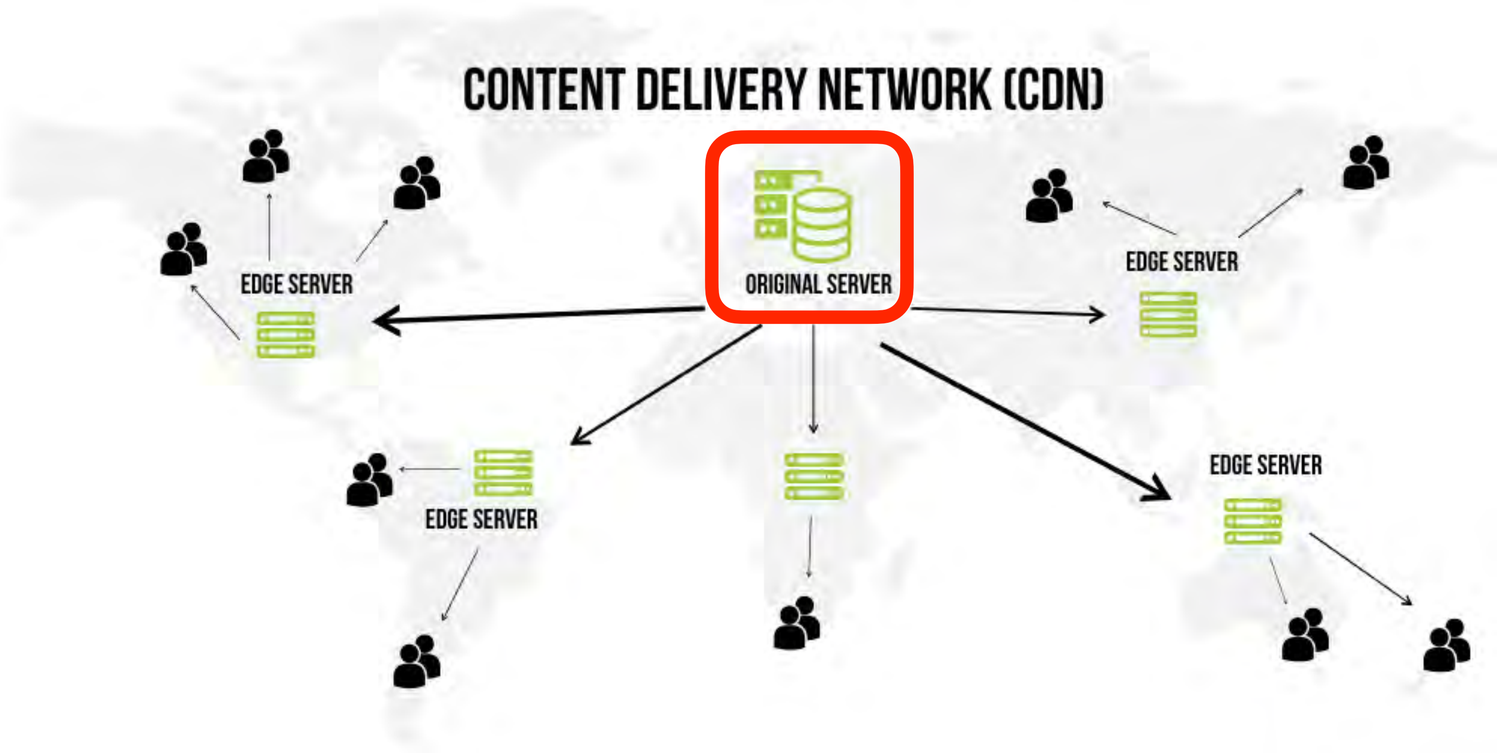
Backend/Network Services

- Content Delivery Networks (CDNs)
- Analytics and reporting
- Email and SMS
- Backup and recovery
- Geo-location
- Integration services
- ...



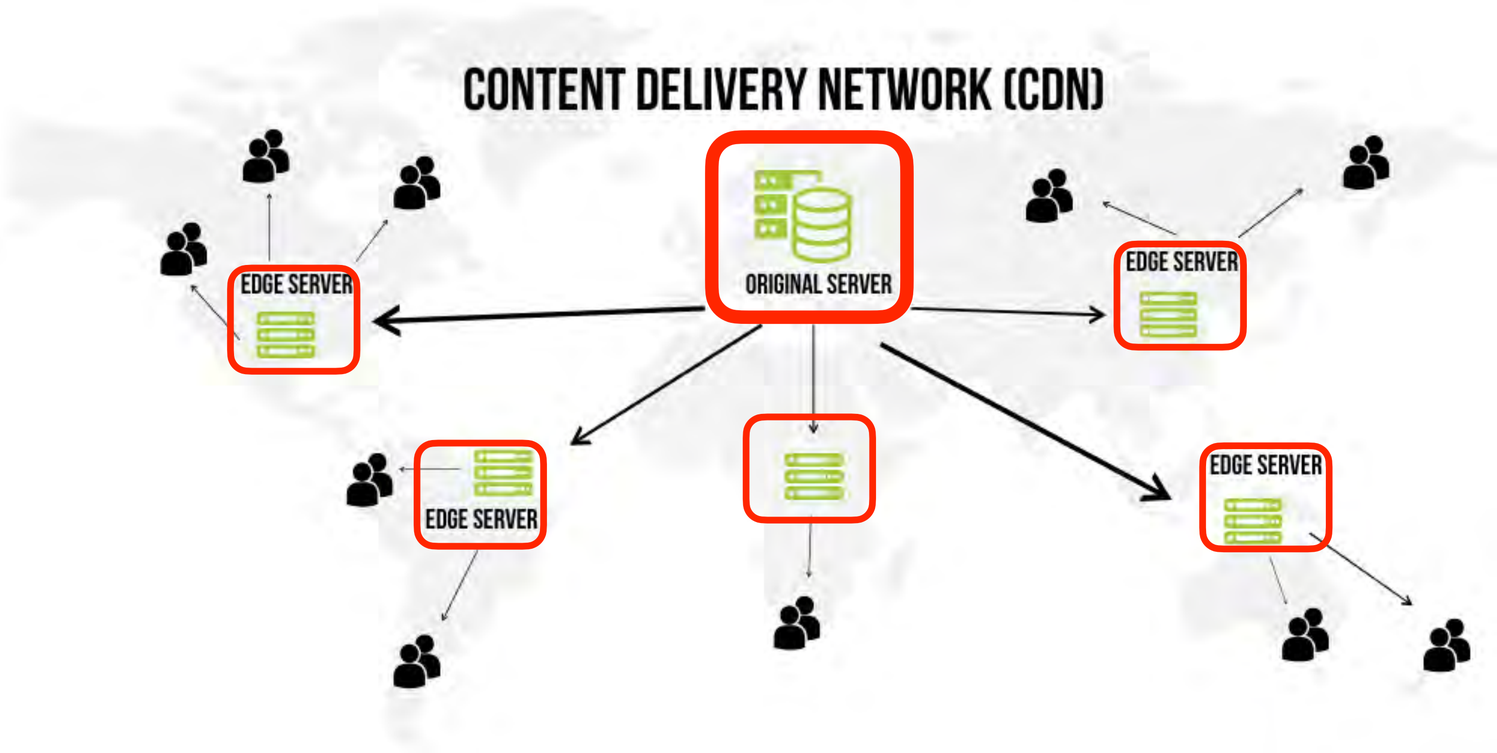
Backend/Network Services

- Content Delivery Networks (CDNs)
- Analytics and reporting
- Email and SMS
- Backup and recovery
- Geo-location
- Integration services
- ...



Backend/Network Services

- Content Delivery Networks (CDNs)
- Analytics and reporting
- Email and SMS
- Backup and recovery
- Geo-location
- Integration services
- ...

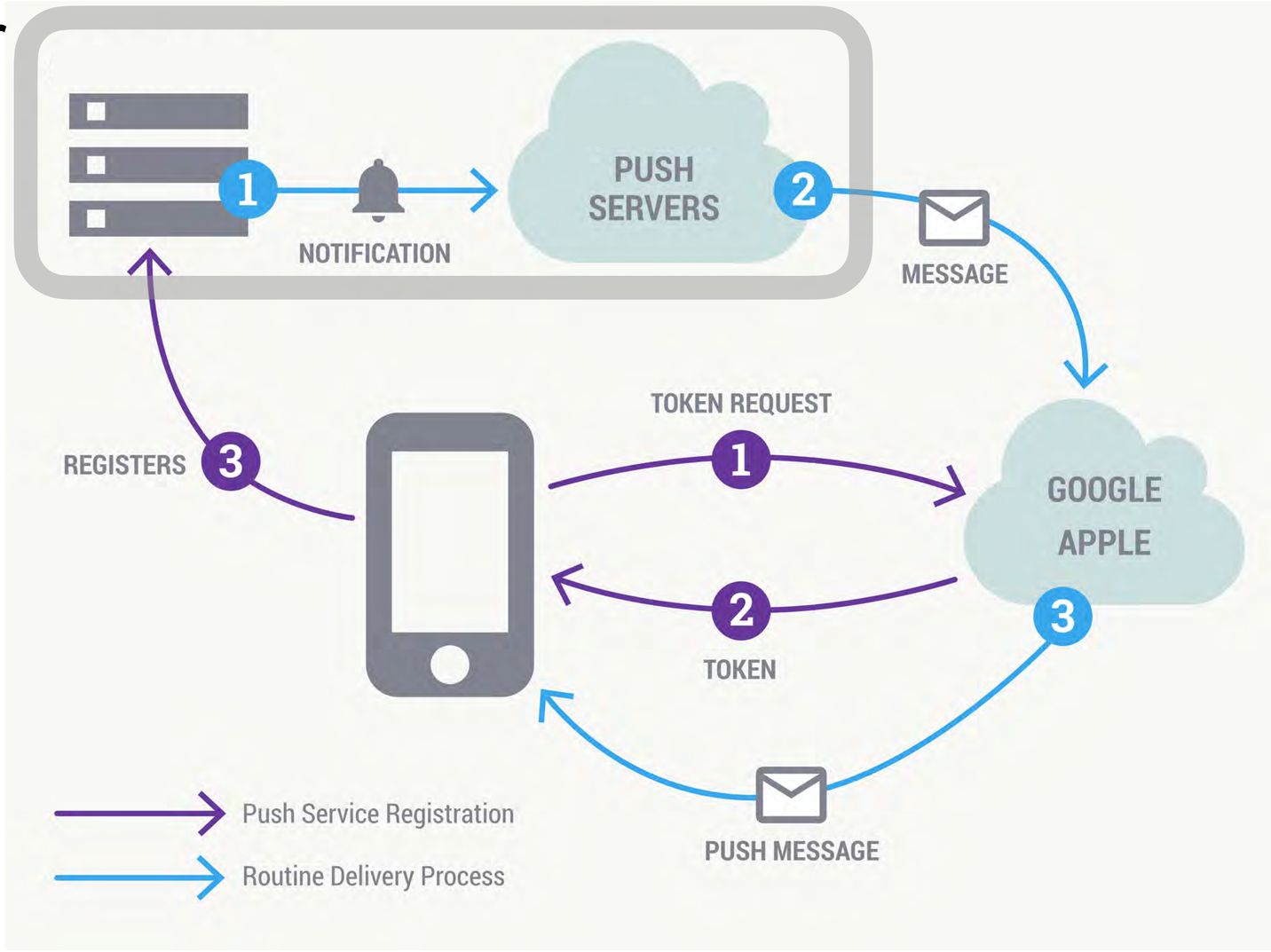


Push Notifications

- Messages sent to the user by the backend of the app
- Each platform has its own push notification service

Push Notifications

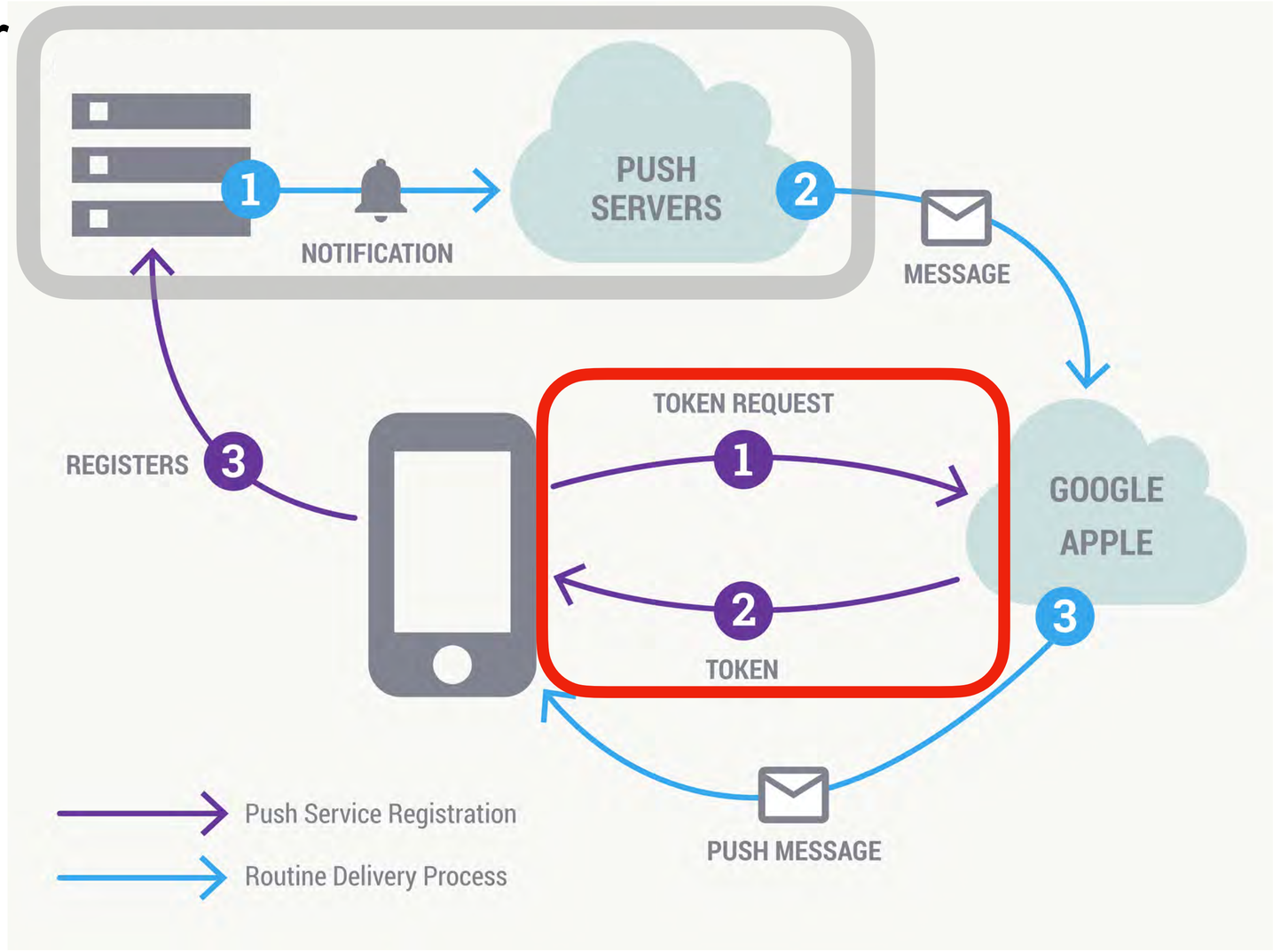
- Messages sent to the user by the backend of the app
- Each platform has its own push notification service



https://knowledge.opsview.com/docs/getting-started-with-push

Push Notifications

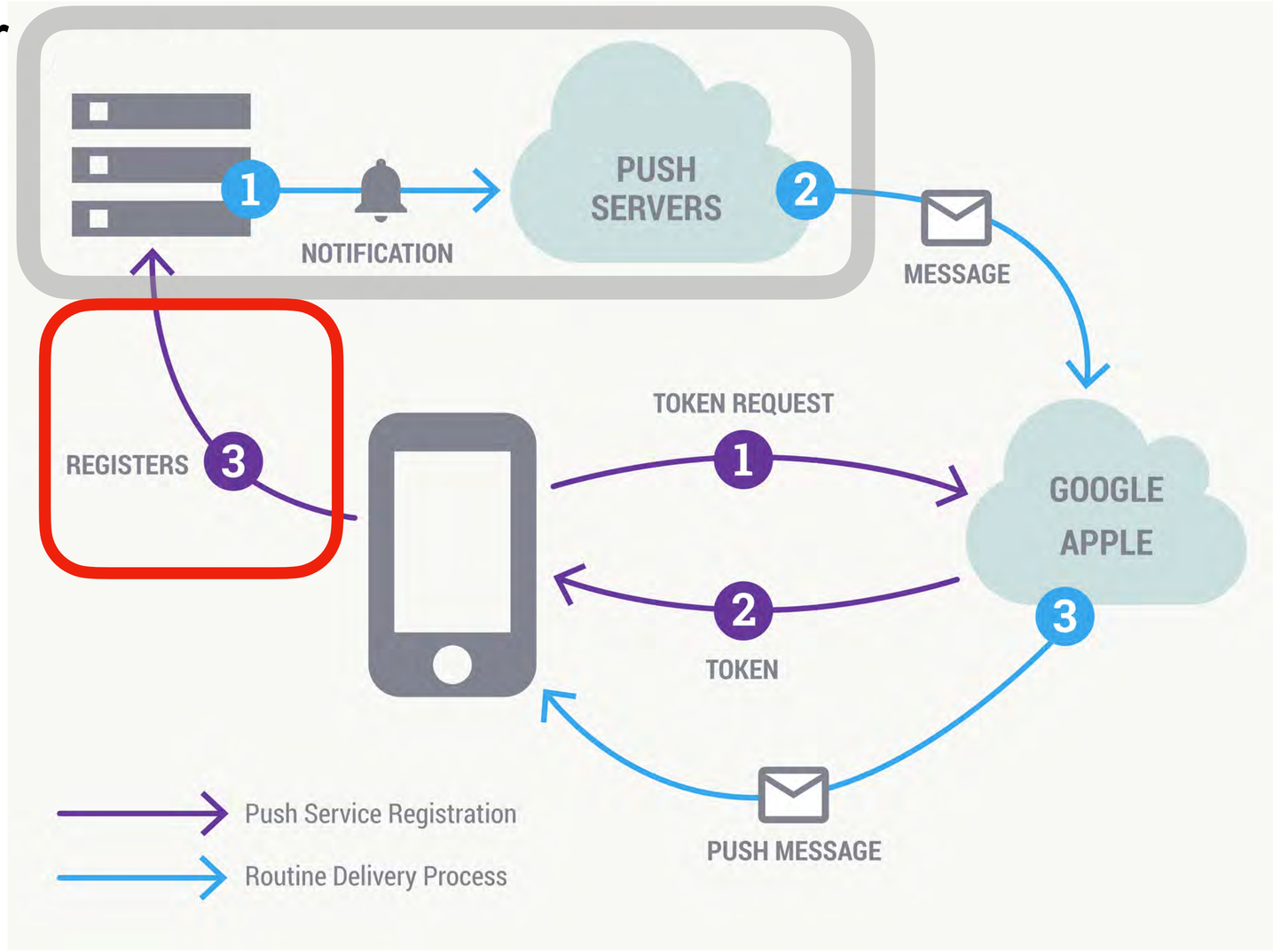
- Messages sent to the user by the backend of the app
- Each platform has its own push notification service



https://knowledge.opsview.com/docs/getting-started-with-push

Push Notifications

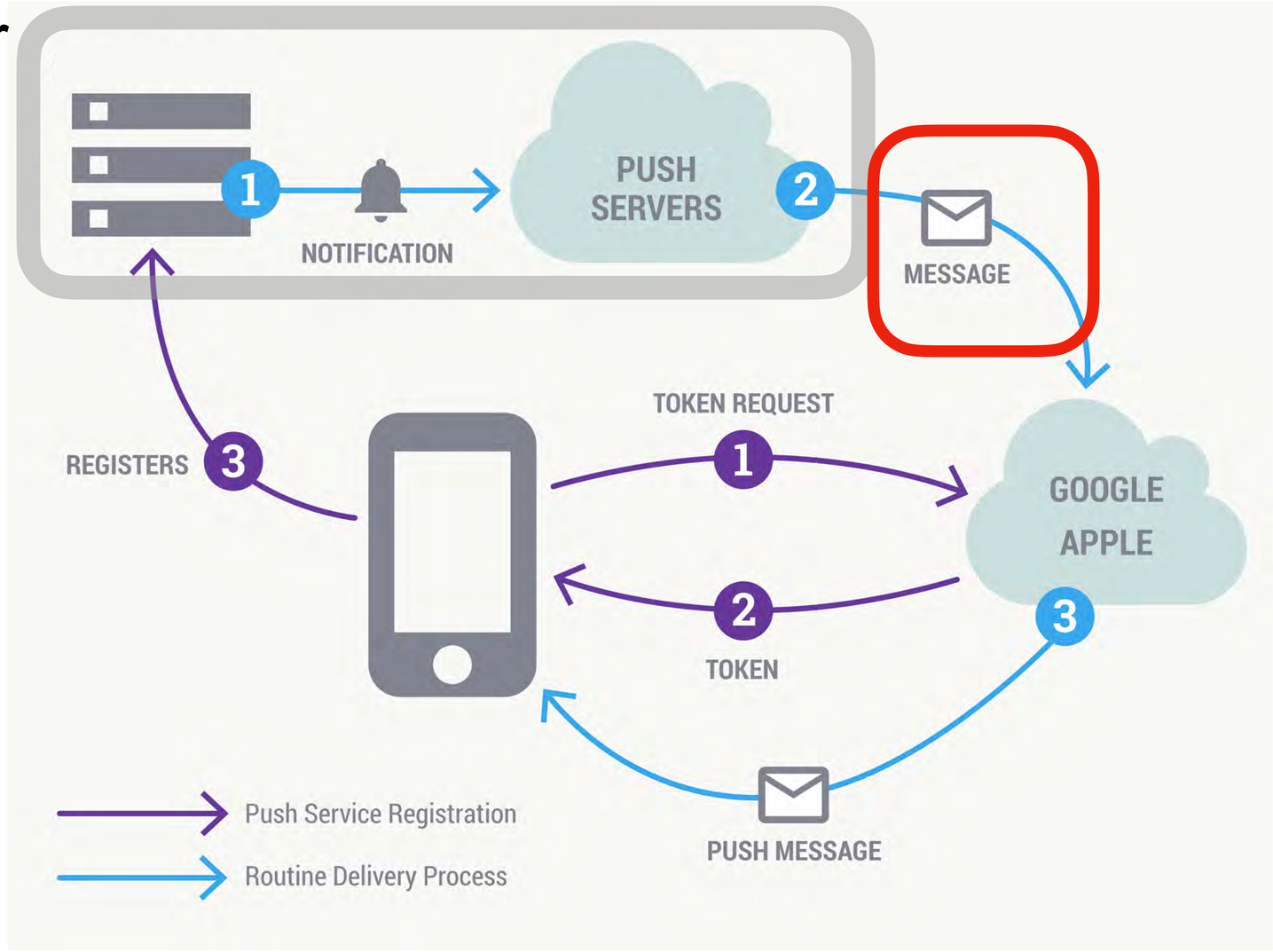
- Messages sent to the user by the backend of the app
- Each platform has its own push notification service



https://knowledge.opsview.com/docs/getting-started-with-push

Push Notifications

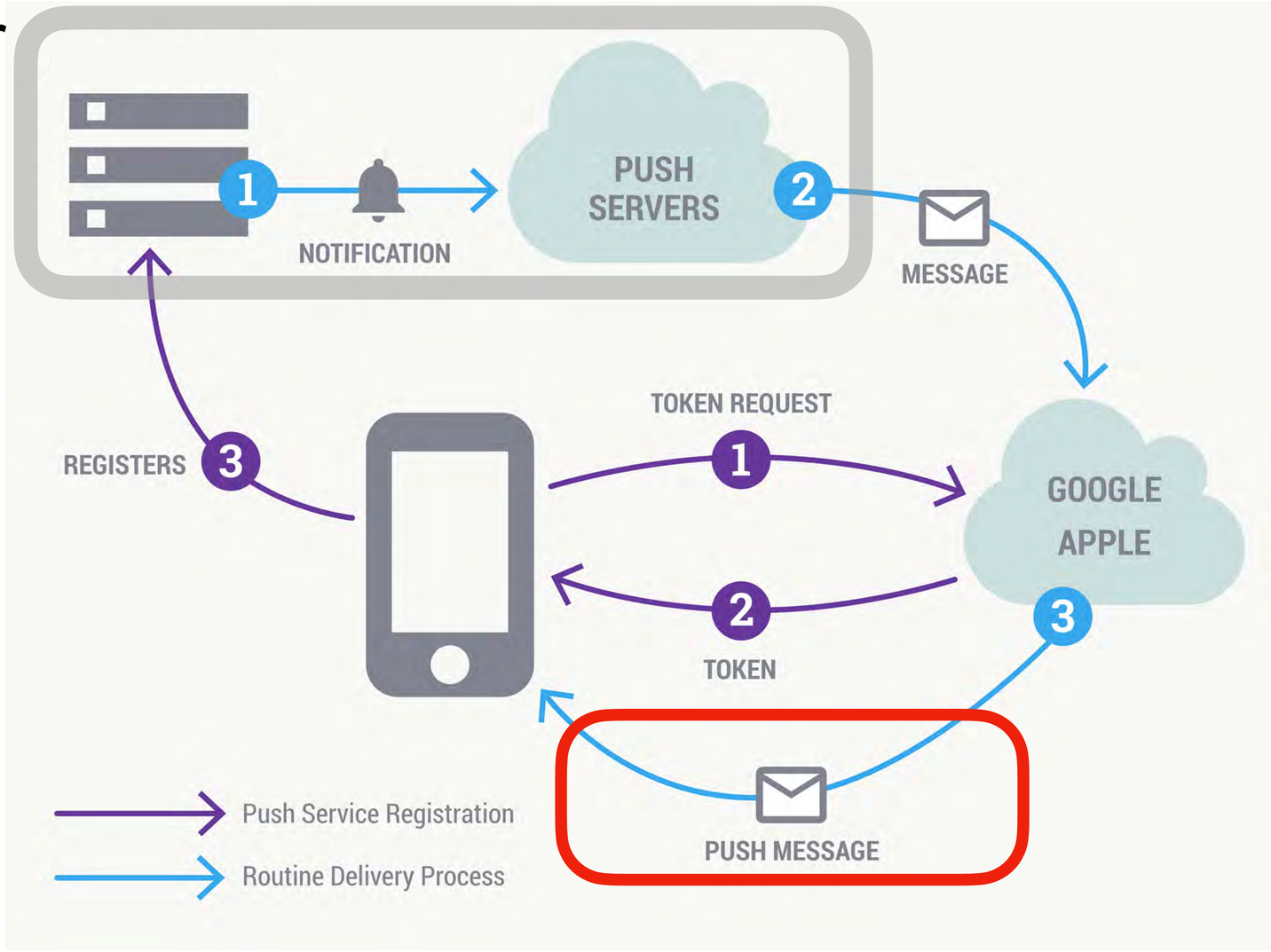
- Messages sent to the user by the backend of the app
- Each platform has its own push notification service



https://knowledge.opsview.com/docs/getting-started-with-push

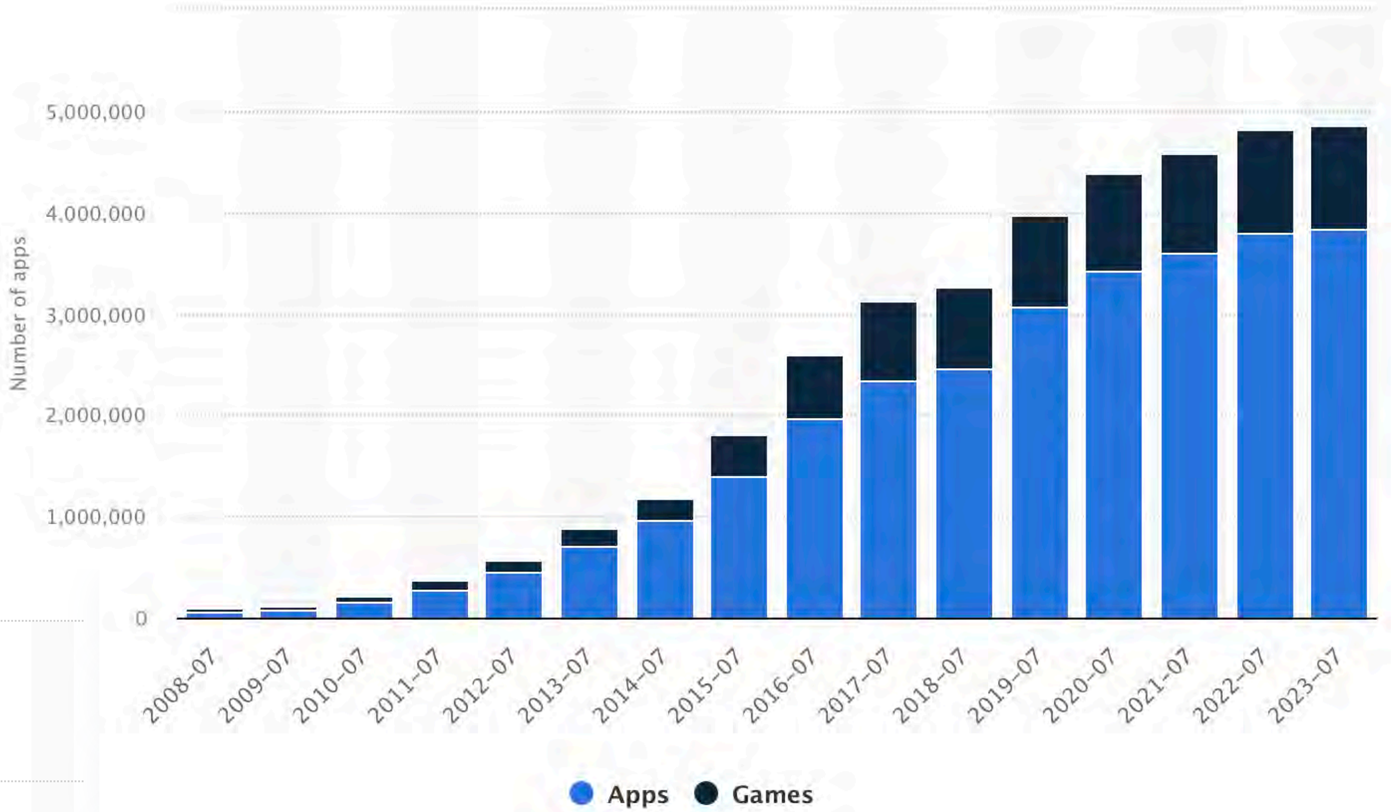
Push Notifications

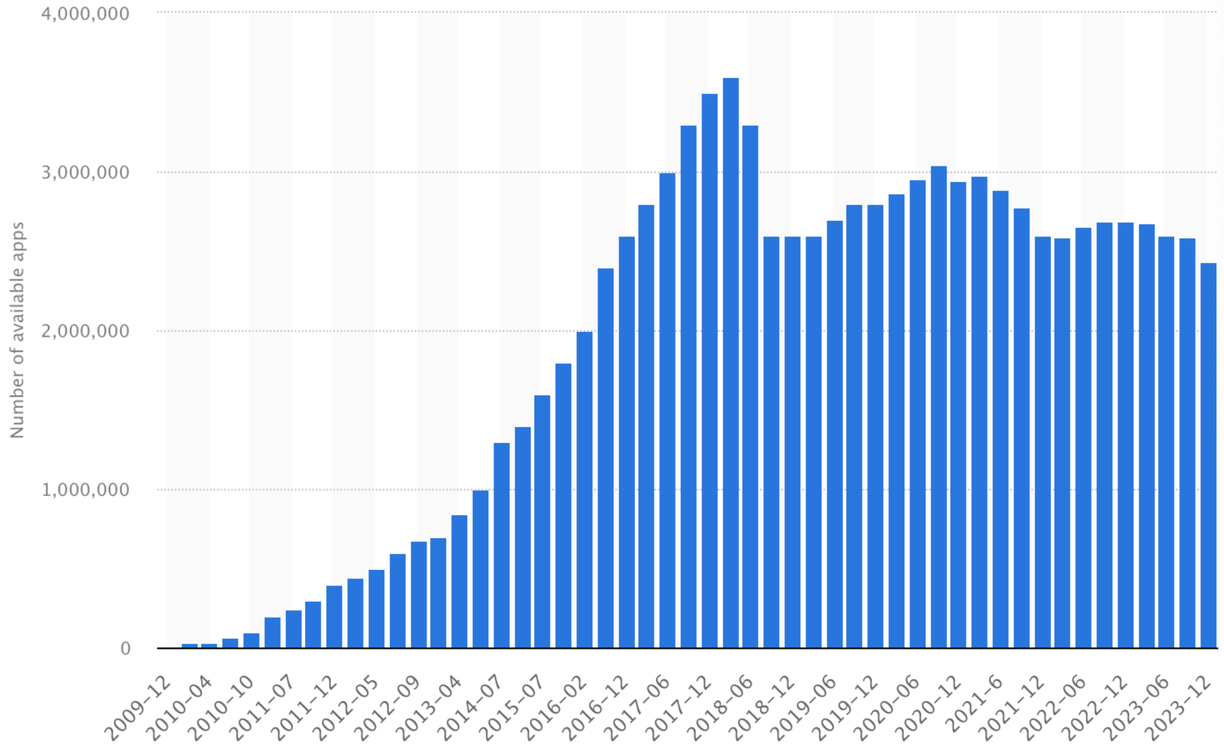
- Messages sent to the user by the backend of the app
- Each platform has its own push notification service



https://knowledge.opsview.com/docs/getting-started-with-push

Download on the App Store





Charts courtesy of Statista.com

GET IT ON Google Play

App Vetting

- Focus: Security, Privacy, UX
- Stringent guidelines on functionality, design, safety, monetization, legal
- Automated initial screening
- In-depth scrutiny by humans
- Common reasons to reject: bugs, crashes, inappropriate content, guideline violations, deception

- Focus: Openness, Flexibility, Scale
- Developer guidelines give more flexibility
- Automated screening
- Monitor apps post-release and rely on user reports to catch bad apps
- Applies more stringent reviews in sensitive categories (financial apps, kids apps, etc.)

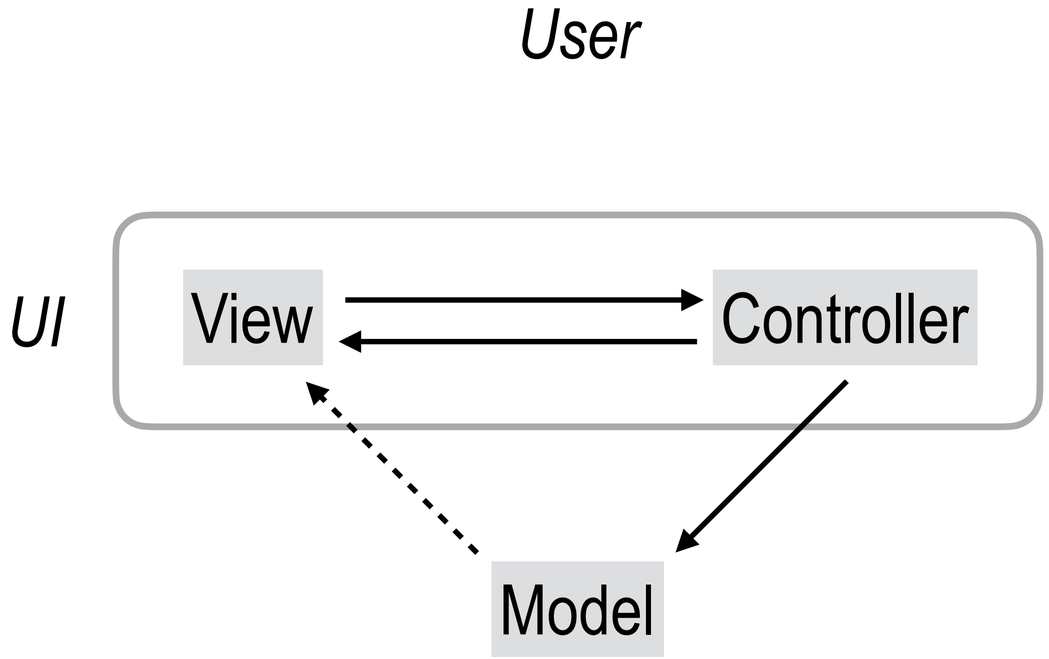
Outline

- Mobile Devices
- Mobile Operating Systems
- Mobile Infrastructure & Services
- Mobile Applications

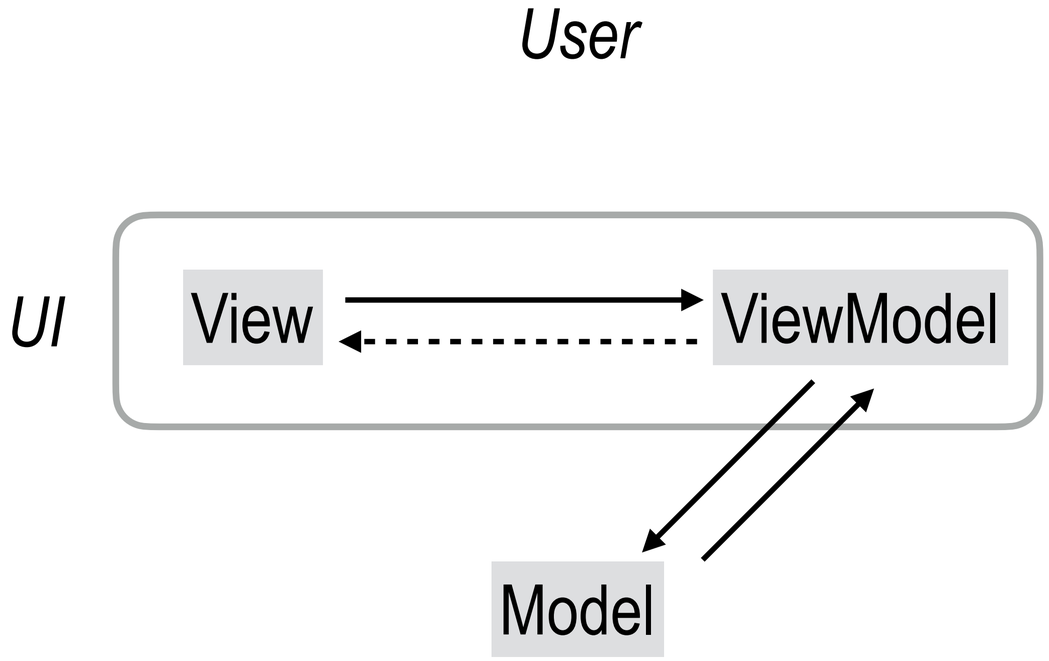
Mobile Applications
≠
Desktop/Server Applications

The MVVM Design Pattern

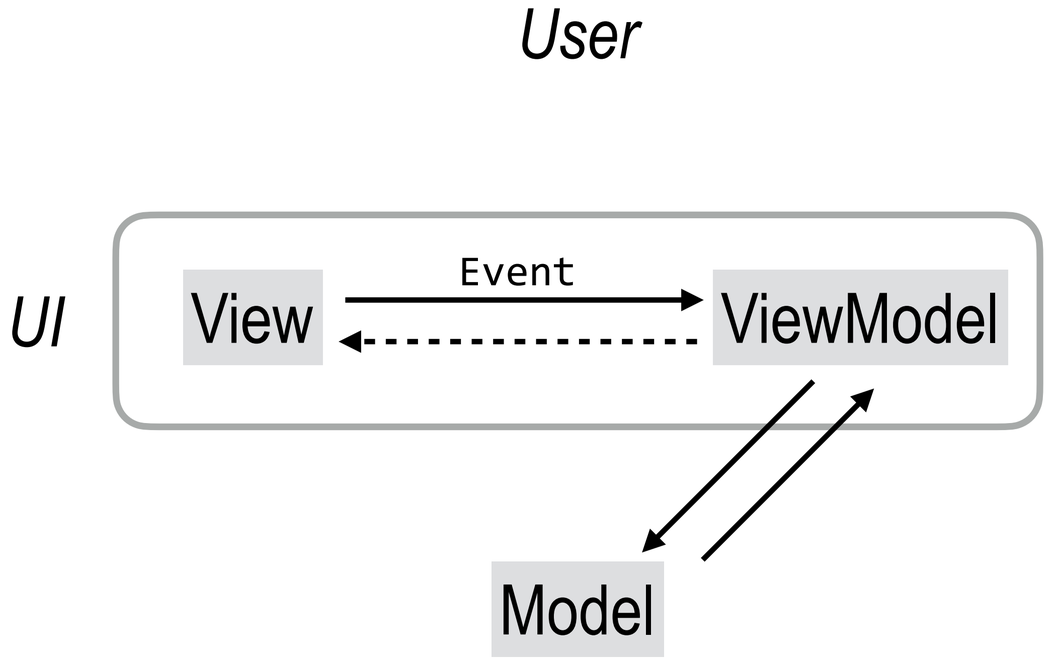
MVC (Recap)



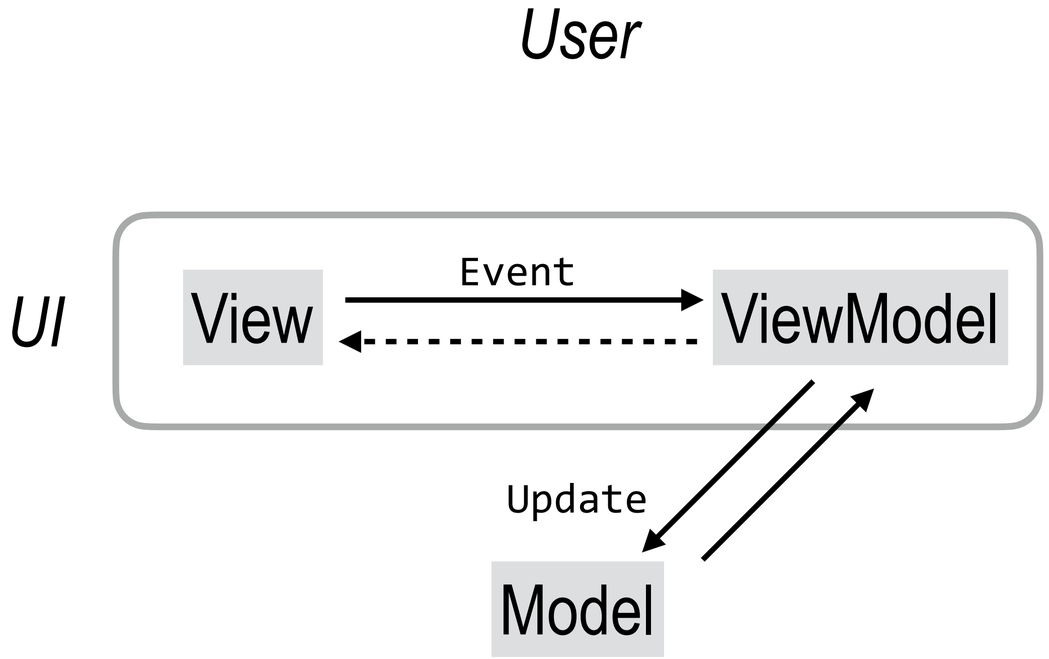
MVVM



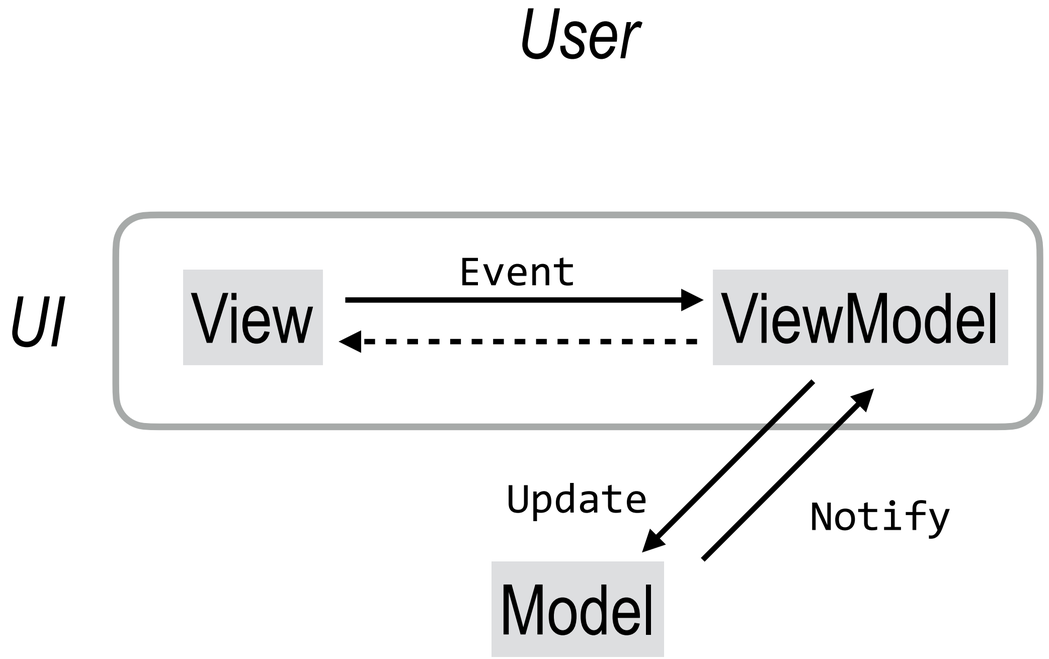
MVVM



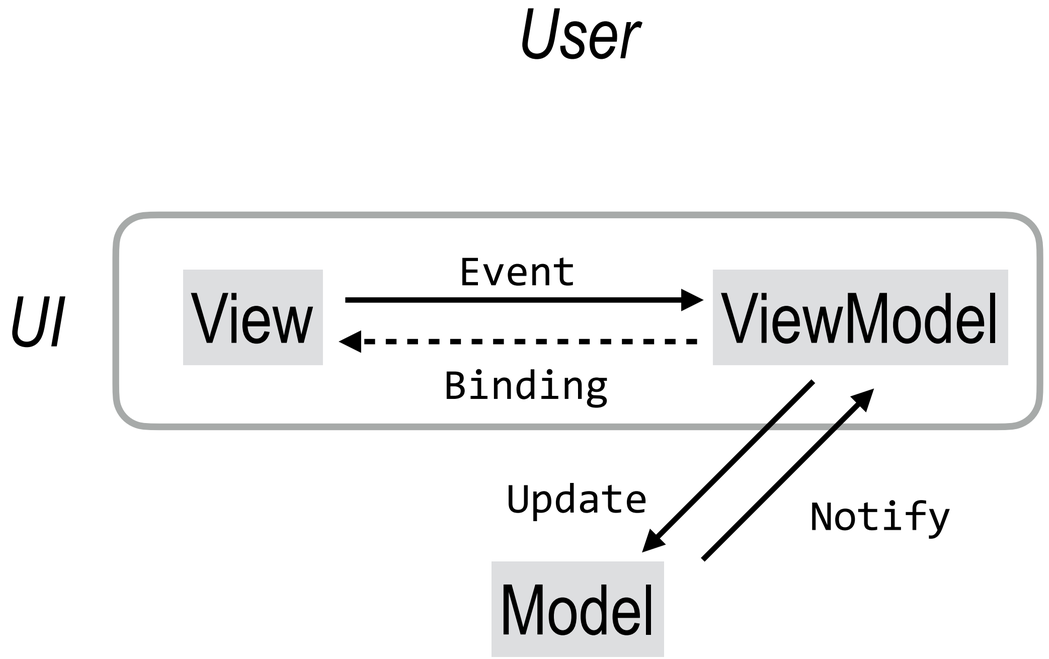
MVVM



MVVM



MVVM



Observability (Android / Jetpack Compose)

- Observability
  - = *ability of the UI (View) to observe changes in the app's data state (ViewModel)*
  - *ViewModel exposes UI state via* `State<T>`, `mutableStateOf()`, `StateFlow`, `Flow`
- Observers are connected to observables
  - *UI functions (*`@Composable`*) read exposed UI state*

    (e.g., via `collectAsState()` *or* `remember { mutableStateOf(...) }`*)*
  - *When state changes → framework recomposes affected UI functions automatically*
- For user input: UI events call ViewModel functions to update state

MVVM Example

View

```
@Composable
fun ChatScreen(vm: ChatVM) {
    val msgs by vm.messages.collectAsState()
    Column {
        msgs.forEach { Text((if (it.fromMe) "You" else "AI") + ": " + it.text) }
        TextField(vm.input, vm::input::set)
        Button(onClick = vm::send) { Text("Send") }
    }
}
```

ViewModel

```
class ChatVM(val llm: DumbLLM) : ViewModel() {
    val messages = llm.messages
    var input by mutableStateOf("")

    fun send() {
        val txt = input.trim()
        if (txt.isNotEmpty()) {
            input = ""
            viewModelScope.launch { llm.send(txt) }
        }
    }
}
```

Model

```
data class Message(val fromMe: Boolean, val text: String)

class DumbLLM {
    private val _msgs = MutableStateFlow<List<Message>>(emptyList())
    val messages: StateFlow<List<Message>> = _msgs

    suspend fun send(text: String) {
        _msgs.value += Message(true, text)
        delay(500)
        _msgs.value += Message(false, "You said \"$text\"")
    }
}
```

MVVM Example

View

```
@Composable
fun ChatScreen(vm: ChatVM) {
    val msgs by vm.messages.collectAsState()
    Column {
        msgs.forEach { Text((if (it.fromMe) "You" else "AI") + ": " + it.text) }
        TextField(vm.input, vm::input::set)
        Button(onClick = vm::send) { Text("Send") }
    }
}
```

ViewModel

```
class ChatVM(val llm: DumbLLM) : ViewModel() {
    val messages = llm.messages
    var input by mutableStateOf("")

    fun send() {
        val txt = input.trim()
        if (txt.isNotEmpty()) {
            input = ""
            viewModelScope.launch { llm.send(txt) }
        }
    }
}
```

Model

```
data class Message(val fromMe: Boolean, val text: String)

class DumbLLM {
    private val _msgs = MutableStateFlow<List<Message>>(emptyList())
    val messages: StateFlow<List<Message>> = _msgs

    suspend fun send(text: String) {
        _msgs.value += Message(true, text)
        delay(500)
        _msgs.value += Message(false, "You said \"$text\"")
    }
}
```

MVVM Example

View

```
@Composable
fun ChatScreen(vm: ChatVM) {
    val msgs by vm.messages.collectAsState()
    Column {
        msgs.forEach { Text((if (it.fromMe) "You" else "AI") + ": " + it.text) }
        TextField(vm.input, vm::input::set)
        Button(onClick = vm::send) { Text("Send") }
    }
}
```

ViewModel

```
class ChatVM(val llm: DumbLLM) : ViewModel() {
    val messages = llm.messages
    var input by mutableStateOf("")

    fun send() {
        val txt = input.trim()
        if (txt.isNotEmpty()) {
            input = ""
            viewModelScope.launch { llm.send(txt) }
        }
    }
}
```

Model

```
data class Message(val fromMe: Boolean, val text: String)

class DumbLLM {
    private val _msgs = MutableStateFlow<List<Message>>(emptyList())
    val messages: StateFlow<List<Message>> = _msgs

    suspend fun send(text: String) {
        _msgs.value += Message(true, text)
        delay(500)
        _msgs.value += Message(false, "You said \"$text\"")
    }
}
```

MVVM Example

View

```
@Composable
fun ChatScreen(vm: ChatVM) {
    val msgs by vm.messages.collectAsState()
    Column {
        msgs.forEach { Text((if (it.fromMe) "You" else "AI") + ": " + it.text) }
        TextField(vm.input, vm::input::set)
        Button(onClick = vm::send) { Text("Send") }
    }
}
```

ViewModel

```
class ChatVM(val llm: DumbLLM) : ViewModel() {
    val messages = llm.messages
    var input by mutableStateOf("")

    fun send() {
        val txt = input.trim()
        if (txt.isNotEmpty()) {
            input = ""
            viewModelScope.launch { llm.send(txt) }
        }
    }
}
```

Model

```
data class Message(val fromMe: Boolean, val text: String)

class DumbLLM {
    private val _msgs = MutableStateFlow<List<Message>>(emptyList())
    val messages: StateFlow<List<Message>> = _msgs

    suspend fun send(text: String) {
        _msgs.value += Message(true, text)
        delay(500)
        _msgs.value += Message(false, "You said \"$text\"")
    }
}
```

MVVM Example

View

```
@Composable
fun ChatScreen(vm: ChatVM) {
    val msgs by vm.messages.collectAsState()
    Column {
        msgs.forEach { Text((if (it.fromMe) "You" else "AI") + ": " + it.text) }
        TextField(vm.input, vm::input::set)
        Button(onClick = vm::send) { Text("Send") }
    }
}
```

ViewModel

```
class ChatVM(val llm: DumbLLM) : ViewModel() {
    val messages = llm.messages
    var input by mutableStateOf("")

    fun send() {
        val txt = input.trim()
        if (txt.isNotEmpty()) {
            input = ""
            viewModelScope.launch { llm.send(txt) }
        }
    }
}
```

Model

```
data class Message(val fromMe: Boolean, val text: String)

class DumbLLM {
    private val _msgs = MutableStateFlow<List<Message>>(emptyList())
    val messages: StateFlow<List<Message>> = _msgs

    suspend fun send(text: String) {
        _msgs.value += Message(true, text)
        delay(500)
        _msgs.value += Message(false, "You said \"$text\"")
    }
}
```

MVVM Example

View

```
@Composable
fun ChatScreen(vm: ChatVM) {
    val msgs by vm.messages.collectAsState()
    Column {
        msgs.forEach { Text((if (it.fromMe) "You" else "AI") + ": " + it.text) }
        TextField(vm.input, vm::input::set)
        Button(onClick = vm::send) { Text("Send") }
    }
}
```

ViewModel

```
class ChatVM(val llm: DumbLLM) : ViewModel() {
    val messages = llm.messages
    var input by mutableStateOf("")

    fun send() {
        val txt = input.trim()
        if (txt.isNotEmpty()) {
            input = ""
            viewModelScope.launch { llm.send(txt) }
        }
    }
}
```

Model

```
data class Message(val fromMe: Boolean, val text: String)

class DumbLLM {
    private val _msgs = MutableStateFlow<List<Message>>(emptyList())
    val messages: StateFlow<List<Message>> = _msgs

    suspend fun send(text: String) {
        _msgs.value += Message(true, text)
        delay(500)
        _msgs.value += Message(false, "You said \"$text\"")
    }
}
```

MVVM Example

View

```
@Composable
fun ChatScreen(vm: ChatVM) {
    val msgs by vm.messages.collectAsState()
    Column {
        msgs.forEach { Text((if (it.fromMe) "You" else "AI") + ": " + it.text) }
        TextField(vm.input, vm::input::set)
        Button(onClick = vm::send) { Text("Send") }
    }
}
```

ViewModel

```
class ChatVM(val llm: DumbLLM) : ViewModel() {
    val messages = llm.messages
    var input by mutableStateOf("")

    fun send() {
        val txt = input.trim()
        if (txt.isNotEmpty()) {
            input = ""
            viewModelScope.launch { llm.send(txt) }
        }
    }
}
```

Model

```
data class Message(val fromMe: Boolean, val text: String)

class DumbLLM {
    private val _msgs = MutableStateFlow<List<Message>>(emptyList())
    val messages: StateFlow<List<Message>> = _msgs

    suspend fun send(text: String) {
        _msgs.value += Message(true, text)
        delay(500)
        _msgs.value += Message(false, "You said \"$text\"")
    }
}
```

MVVM Example

View

```
@Composable
fun ChatScreen(vm: ChatVM) {
    val msgs by vm.messages.collectAsState()
    Column {
        msgs.forEach { Text((if (it.fromMe) "You" else "AI") + ": " + it.text) }
        TextField(vm.input, vm::input::set)
        Button(onClick = vm::send) { Text("Send") }
    }
}
```

ViewModel

```
class ChatVM(val llm: DumbLLM) : ViewModel() {
    val messages = llm.messages
    var input by mutableStateOf("")

    fun send() {
        val txt = input.trim()
        if (txt.isNotEmpty()) {
            input = ""
            viewModelScope.launch { llm.send(txt) }
        }
    }
}
```

Model

```
data class Message(val fromMe: Boolean, val text: String)

class DumbLLM {
    private val _msgs = MutableStateFlow<List<Message>>(emptyList())
    val messages: StateFlow<List<Message>> = _msgs

    suspend fun send(text: String) {
        _msgs.value += Message(true, text)
        delay(500)
        _msgs.value += Message(false, "You said \"$text\"")
    }
}
```

MVVM Example

View

```
@Composable
fun ChatScreen(vm: ChatVM) {
    val msgs by vm.messages.collectAsState()
    Column {
        msgs.forEach { Text((if (it.fromMe) "You" else "AI") + ": " + it.text) }
        TextField(vm.input, vm::input::set)
        Button(onClick = vm::send) { Text("Send") }
    }
}
```

ViewModel

```
class ChatVM(val llm: DumbLLM) : ViewModel() {
    val messages = llm.messages
    var input by mutableStateOf("")

    fun send() {
        val txt = input.trim()
        if (txt.isNotEmpty()) {
            input = ""
            viewModelScope.launch { llm.send(txt) }
        }
    }
}
```

Model

```
data class Message(val fromMe: Boolean, val text: String)

class DumbLLM {
    private val _msgs = MutableStateFlow<List<Message>>(emptyList())
    val messages: StateFlow<List<Message>> = _msgs

    suspend fun send(text: String) {
        _msgs.value += Message(true, text)
        delay(500)
        _msgs.value += Message(false, "You said \"$text\"")
    }
}
```

MVVM Example

View

```
@Composable
fun ChatScreen(vm: ChatVM) {
    val msgs by vm.messages.collectAsState()
    Column {
        msgs.forEach { Text((if (it.fromMe) "You" else "AI") + ": " + it.text) }
        TextField(vm.input, vm::input::set)
        Button(onClick = vm::send) { Text("Send") }
    }
}
```

ViewModel

```
class ChatVM(val llm: DumbLLM) : ViewModel() {
    val messages = llm.messages
    var input by mutableStateOf("")

    fun send() {
        val txt = input.trim()
        if (txt.isNotEmpty()) {
            input = ""
            viewModelScope.launch { llm.send(txt) }
        }
    }
}
```

Model

```
data class Message(val fromMe: Boolean, val text: String)

class DumbLLM {
    private val _msgs = MutableStateFlow<List<Message>>(emptyList())
    val messages: StateFlow<List<Message>> = _msgs

    suspend fun send(text: String) {
        _msgs.value += Message(true, text)
        delay(500)
        _msgs.value += Message(false, "You said \"$text\"")
    }
}
```

MVVM Example

View

```
@Composable
fun ChatScreen(vm: ChatVM) {
    val msgs by vm.messages.collectAsState()
    Column {
        msgs.forEach { Text((if (it.fromMe) "You" else "AI") + ": " + it.text) }
        TextField(vm.input, vm::input::set)
        Button(onClick = vm::send) { Text("Send") }
    }
}
```

ViewModel

```
class ChatVM(val llm: DumbLLM) : ViewModel() {
    val messages = llm.messages
    var input by mutableStateOf("")

    fun send() {
        val txt = input.trim()
        if (txt.isNotEmpty()) {
            input = ""
            viewModelScope.launch { llm.send(txt) }
        }
    }
}
```

Model

```
data class Message(val fromMe: Boolean, val text: String)

class DumbLLM {
    private val _msgs = MutableStateFlow<List<Message>>(emptyList())
    val messages: StateFlow<List<Message>> = _msgs

    suspend fun send(text: String) {
        _msgs.value += Message(true, text)
        delay(500)
        _msgs.value += Message(false, "You said \"$text\"")
    }
}
```

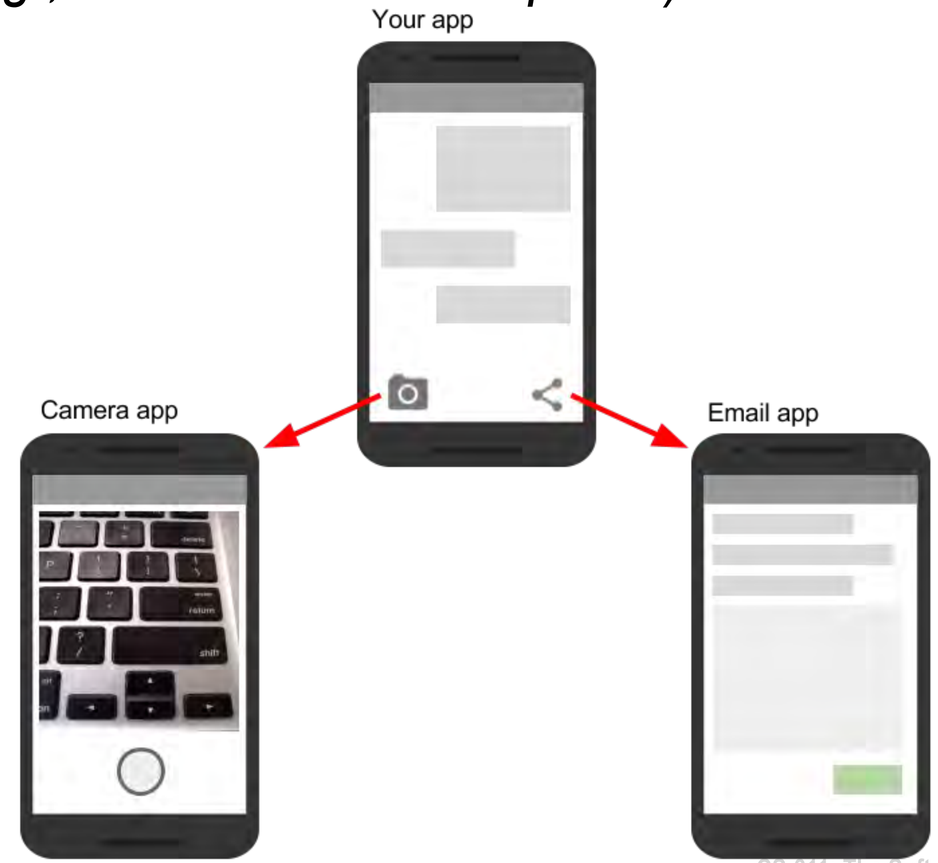
Benefits of MVVM

- Decouple UI logic from aplication's logic ("business logic")
  - *View is conceptually a function of the ViewModel => can be entirely computed from the ViewModel every time*
  - *E.g., on mobile: app may run in background and need no UI — can destroy View and re-create from the ViewModel whenever needed*
- Enables UIs that are reactive, responsive, dynamic
- Easy to test
- Easy to develop by separate people
- Easy to evolve, maintain, and reuse

Mobile Application Internals
(illustrated with Android)

Four Android App Components

- **[1] Activities**
  - *Entry point for interacting with the user (1 screen + UI)*
  - *One app can start an activity in another app (e.g., camera can email photo)*

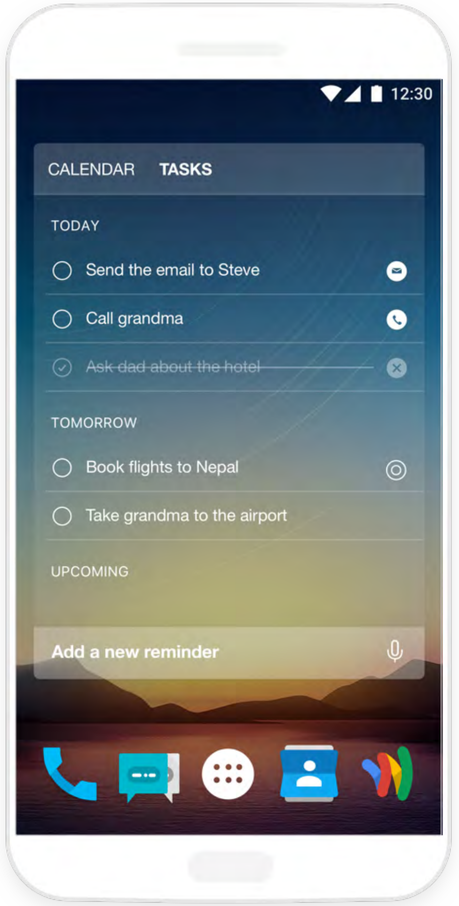


Four Android App Components

- **[1] Activities**
  - *Entry point for interacting with the user (1 screen + UI)*
  - *One app can start an activity in another app (e.g., camera can email photo)*
- **[2] Services**
  - *Run in background without UI*
  - *(often replaced or augmented by WorkManager for deferrable tasks)*

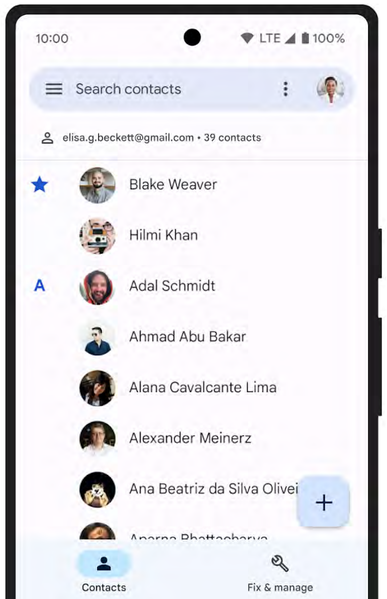
Four Android App Components

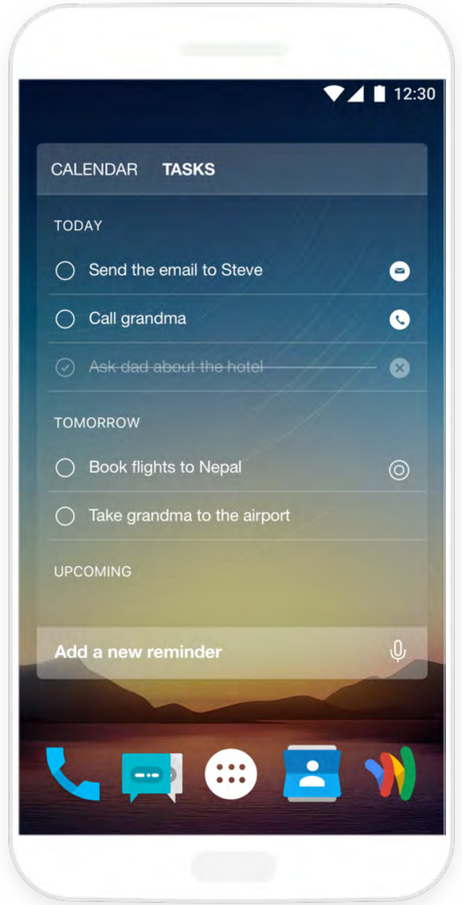
- **[3] Broadcast receivers**
  - *Receive "out of band" event broadcasts*
  - *App doesn't need to be running to receive broadcast*



Four Android App Components

- **[3] Broadcast receivers**
  - *Receive "out of band" event broadcasts*
  - *App doesn't need to be running to receive broadcast*
- **[4] Content providers**
  - *Manage data that is of interest to multiple apps*



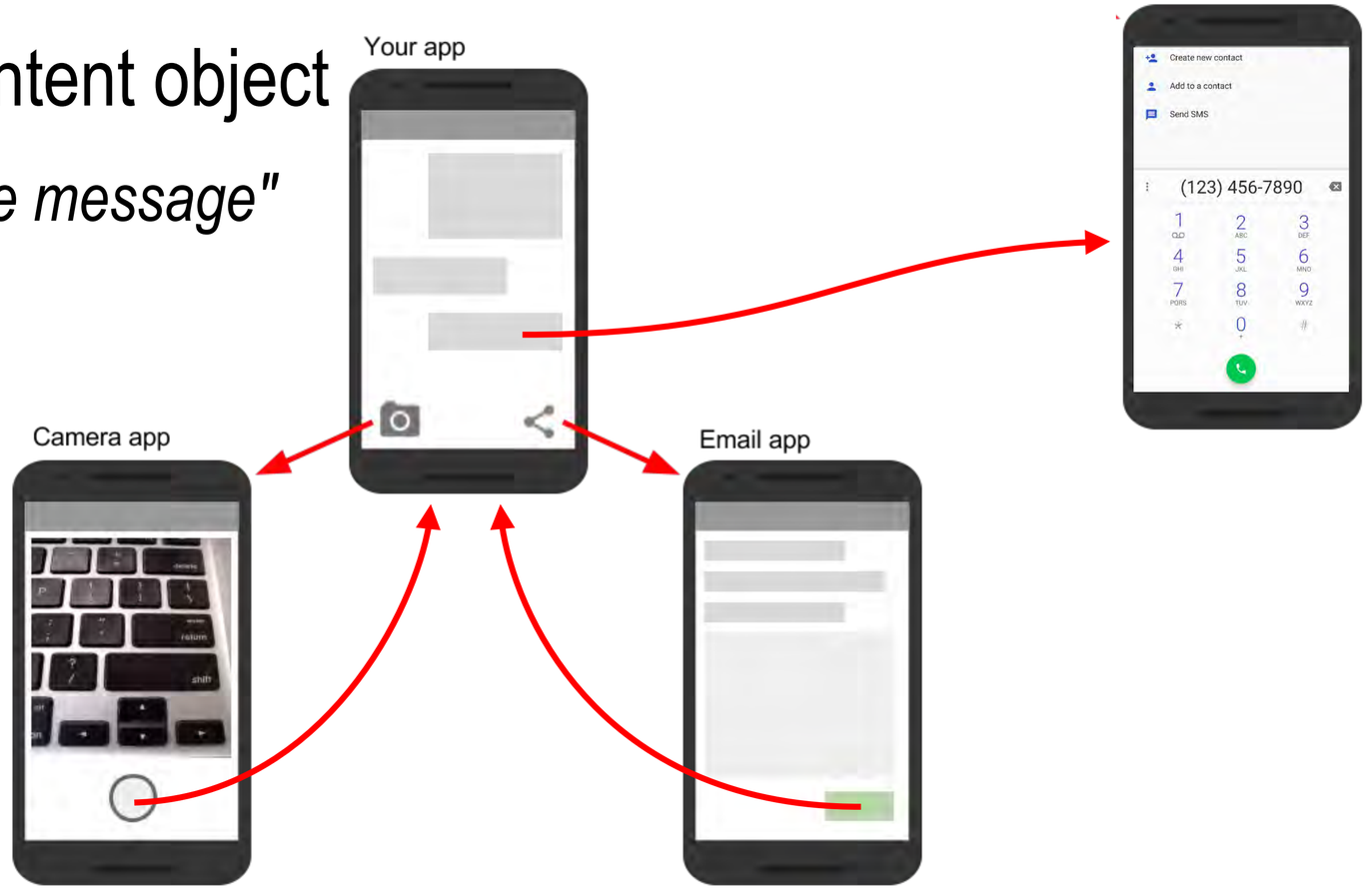


Activities

- Key part of an Android app
- Provides the window in which the app draws its UI
- UI consists of multiple screens
- Screens defined inside the main Activity

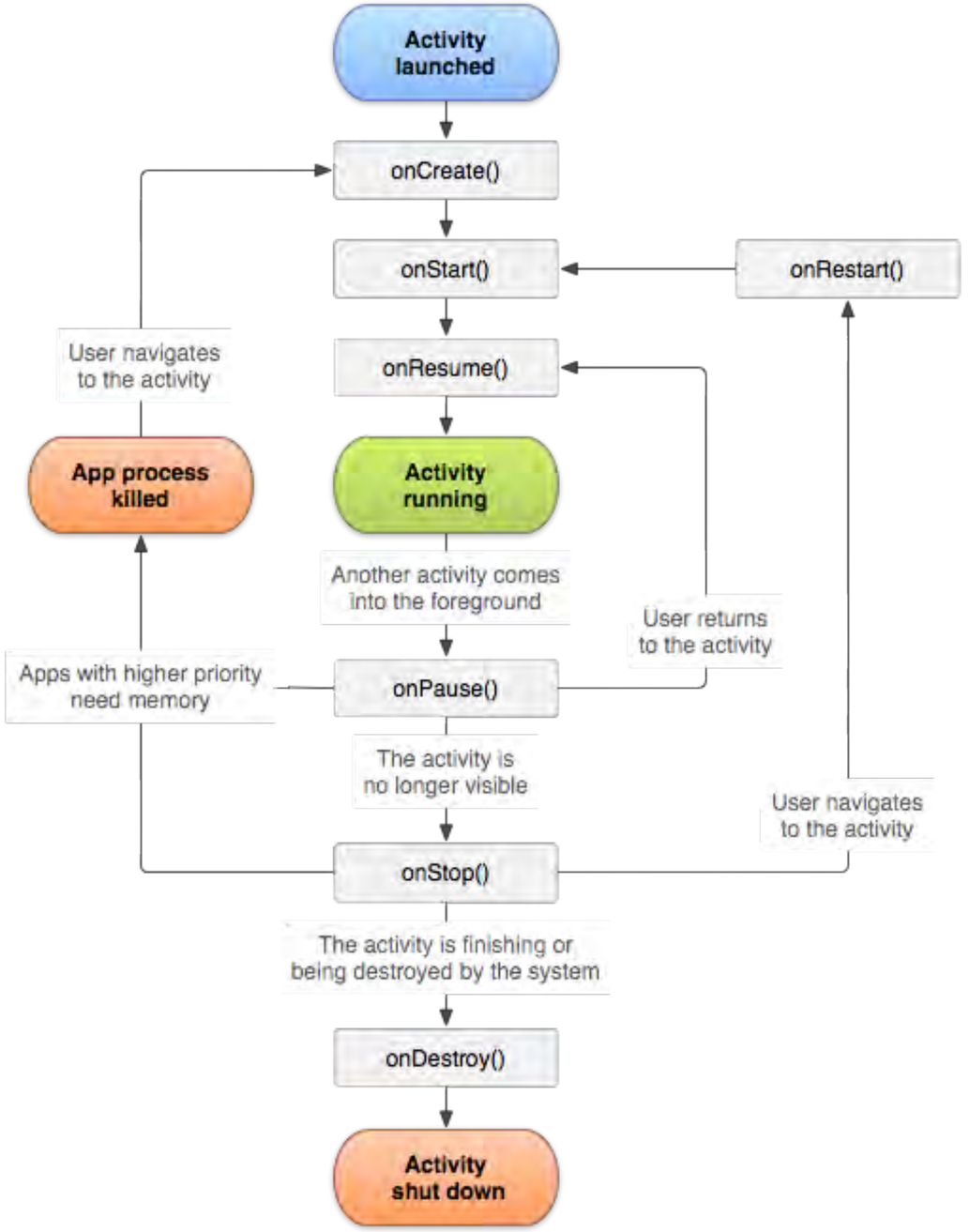
Inter-App Communication via Intents

- An app can cause another app's component to start
  - *E.g., WhatsApp takes a photo with the Camera app*
- Activation is via an Intent object
  - *Think of it as an "active message"*

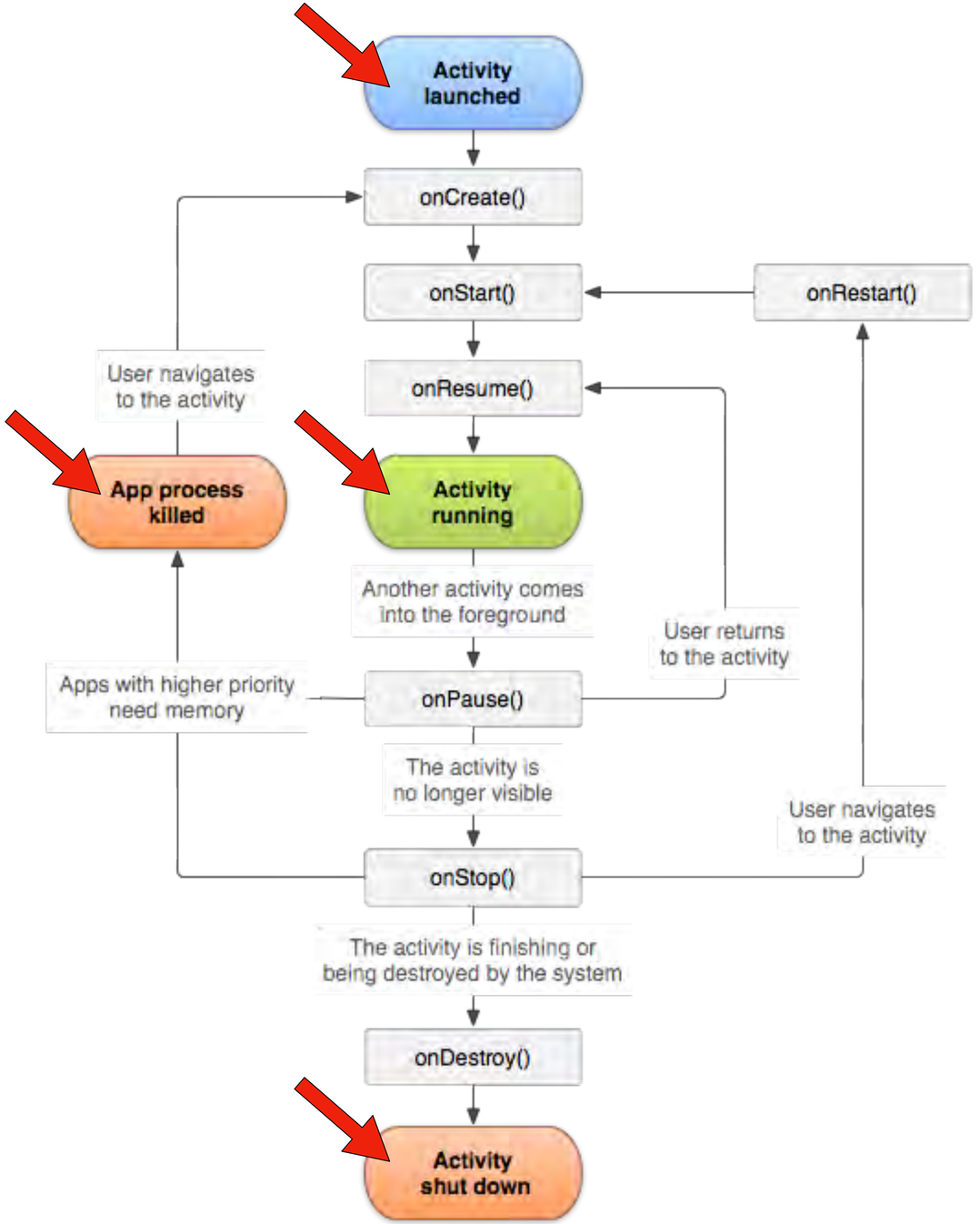


Inter-App Communication via Intents

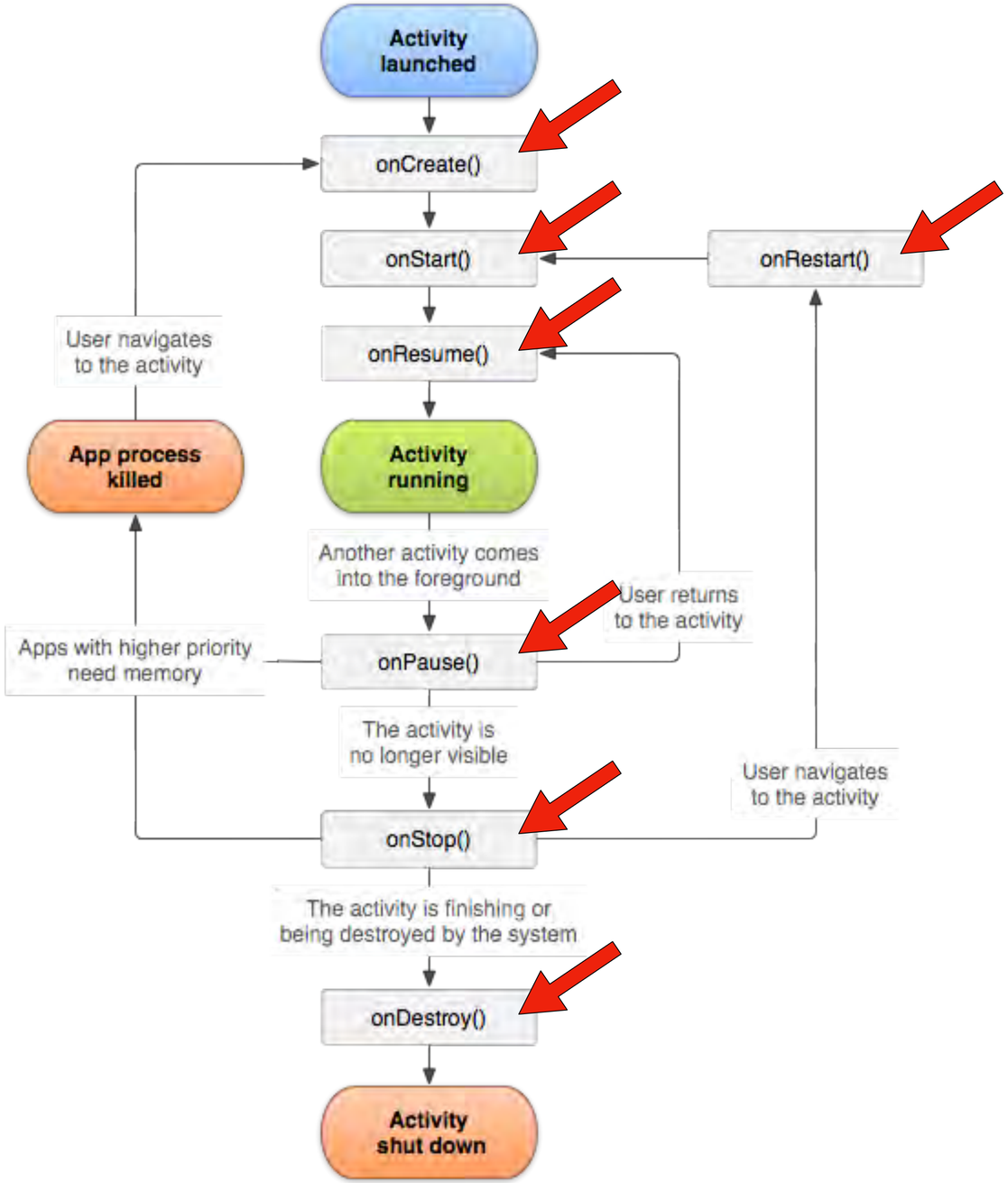
- An app can cause another app's component to start
  - *E.g., WhatsApp takes a photo with the Camera app*
  - *Activation is via an Intent object (think of it as an "active message")*
- To start activities and services: <action, URI> → explicit intent
  - *E.g., <ACTION_DIAL, "tel:123456789"), <ACTION_WEB_SEARCH, "app developers")*
- Register for broadcast announcements: <announcement> → implicit intent
  - *E.g., <ACTION_BATTERY_LOW>, <ACTION_POWER_DISCONNECTED>*
- For content providers: pull instead of push
  - *Activated by request from a ContentResolver*



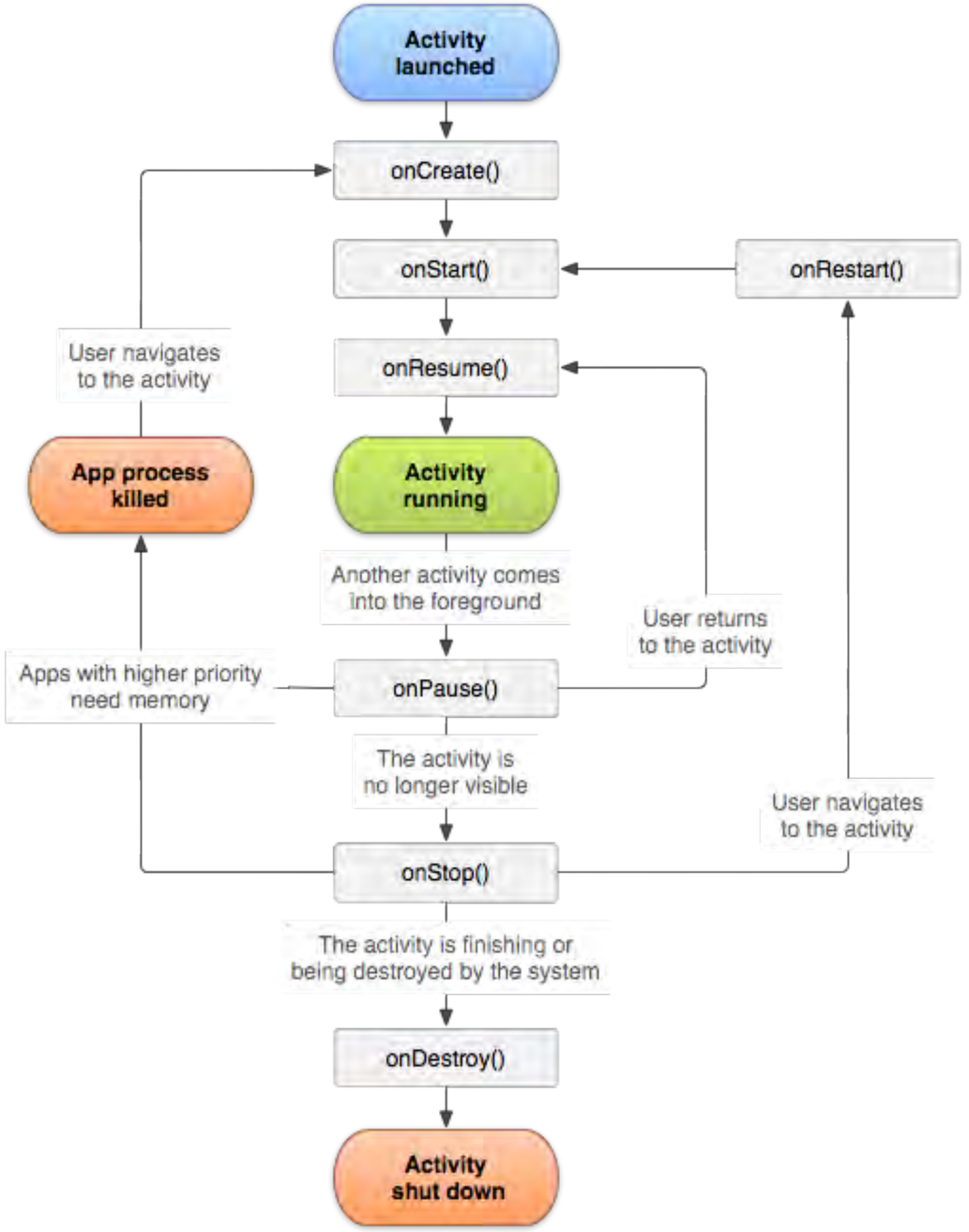
https://developer.android.com/guide/components/activities/activity-lifecycle



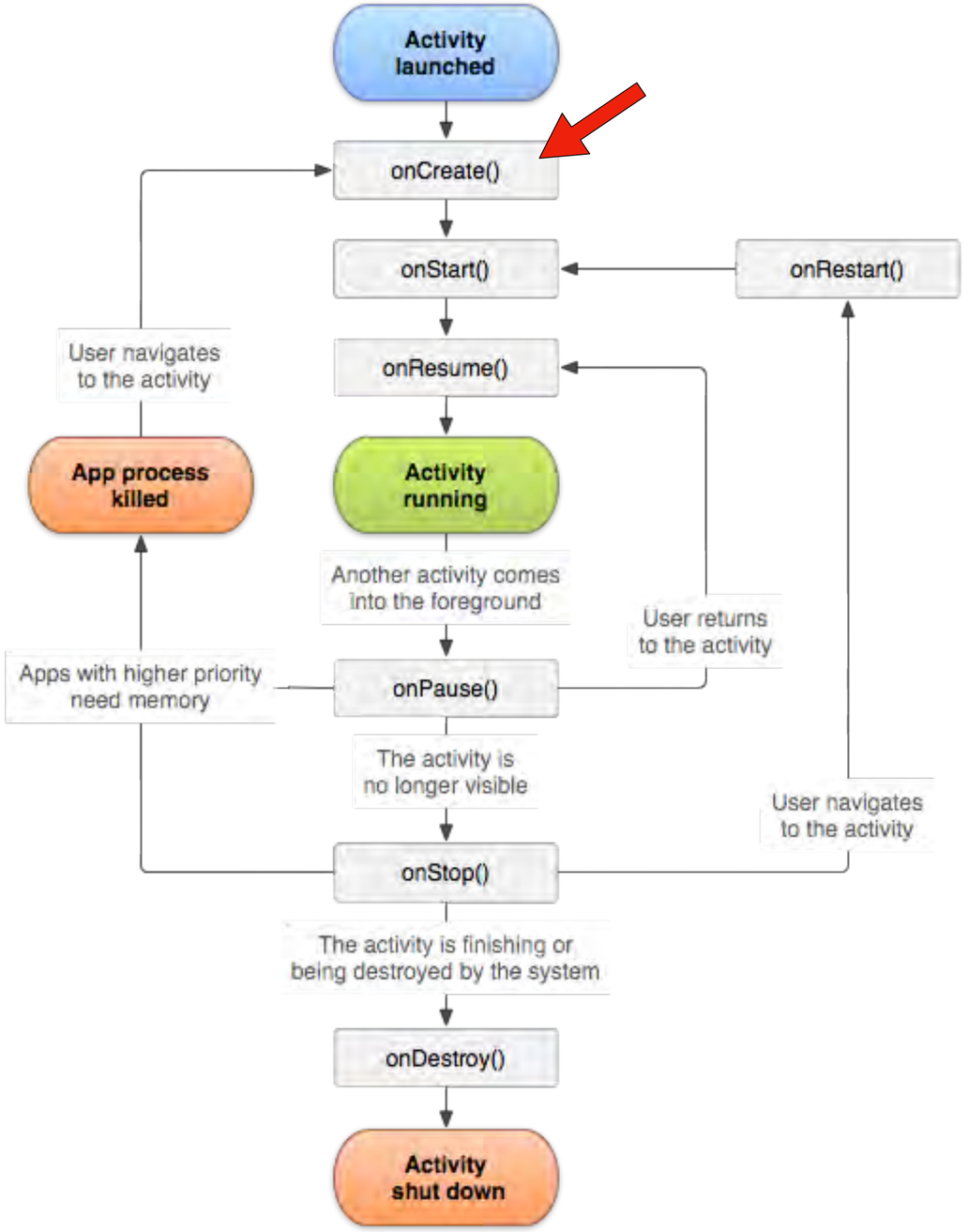
https://developer.android.com/guide/components/activities/activity-lifecycle



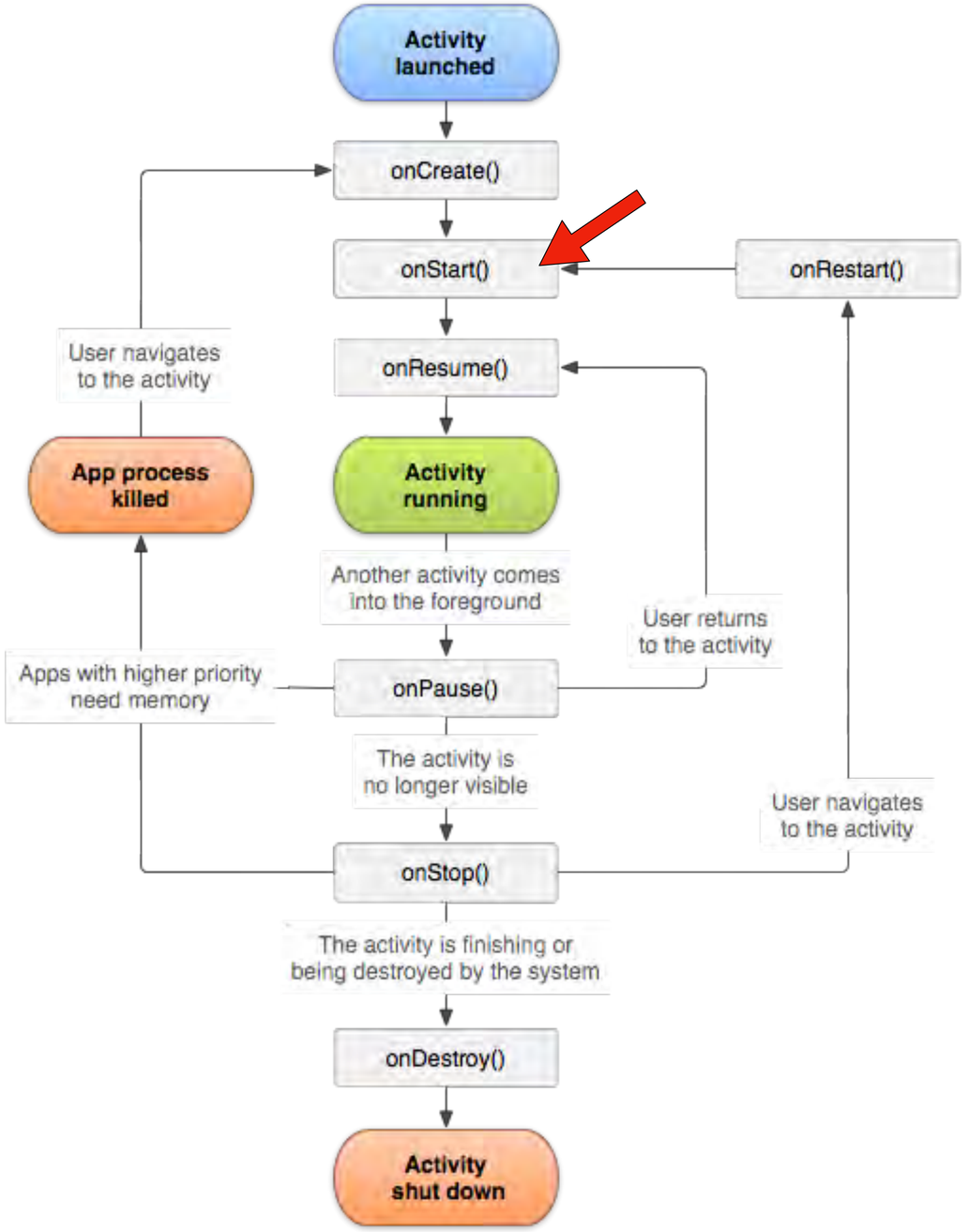
https://developer.android.com/guide/components/activities/activity-lifecycle



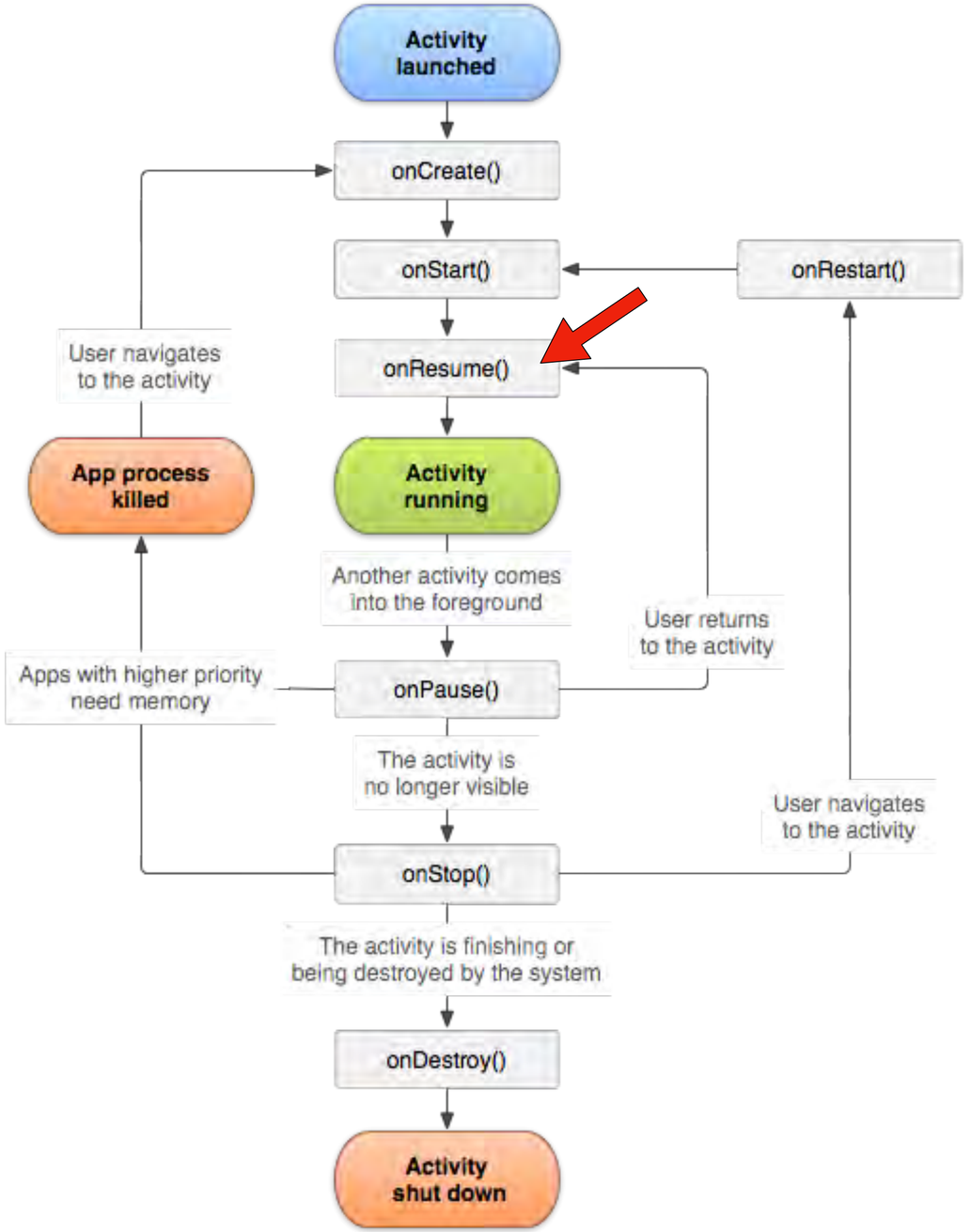
https://developer.android.com/guide/components/activities/activity-lifecycle



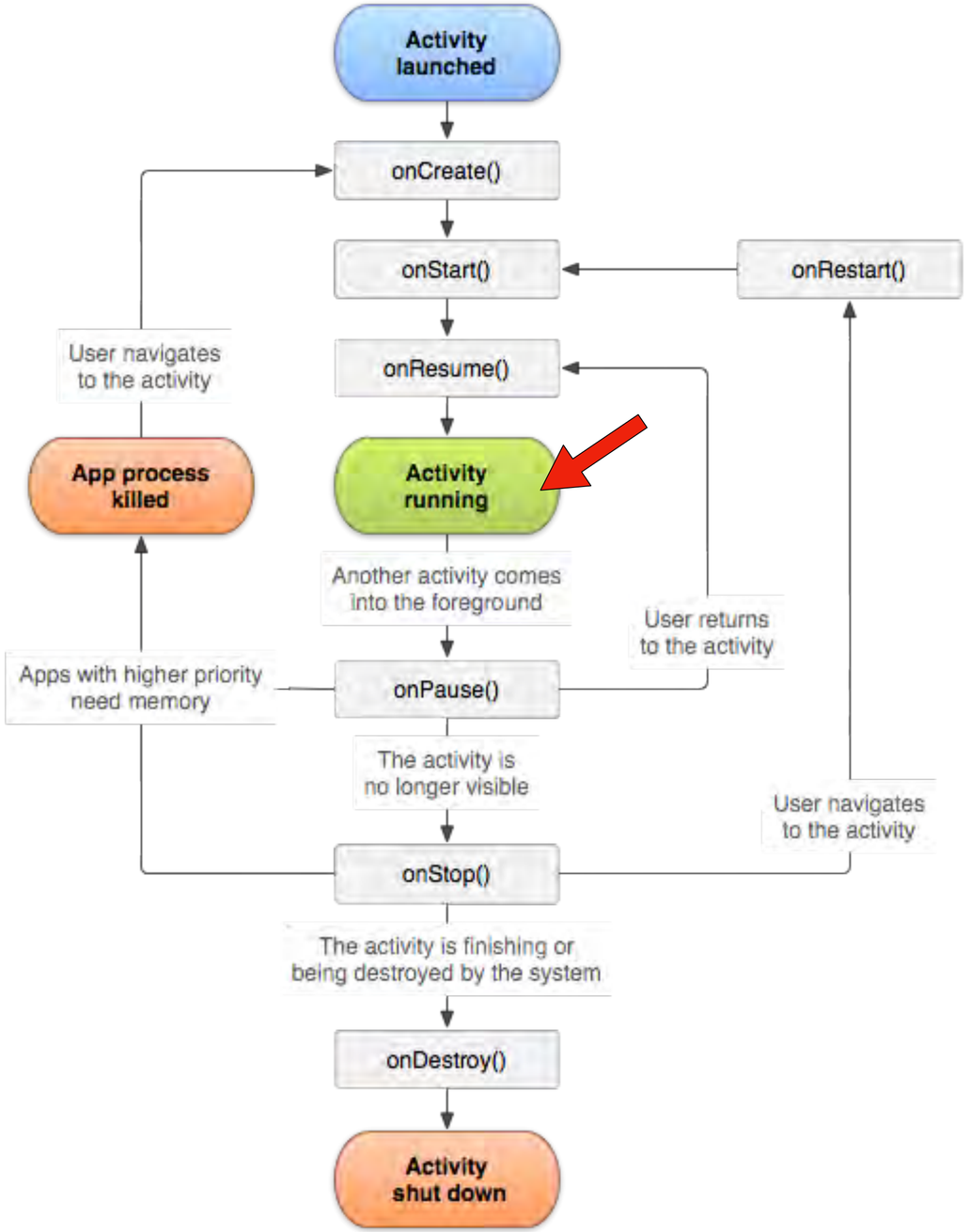
https://developer.android.com/guide/components/activities/activity-lifecycle



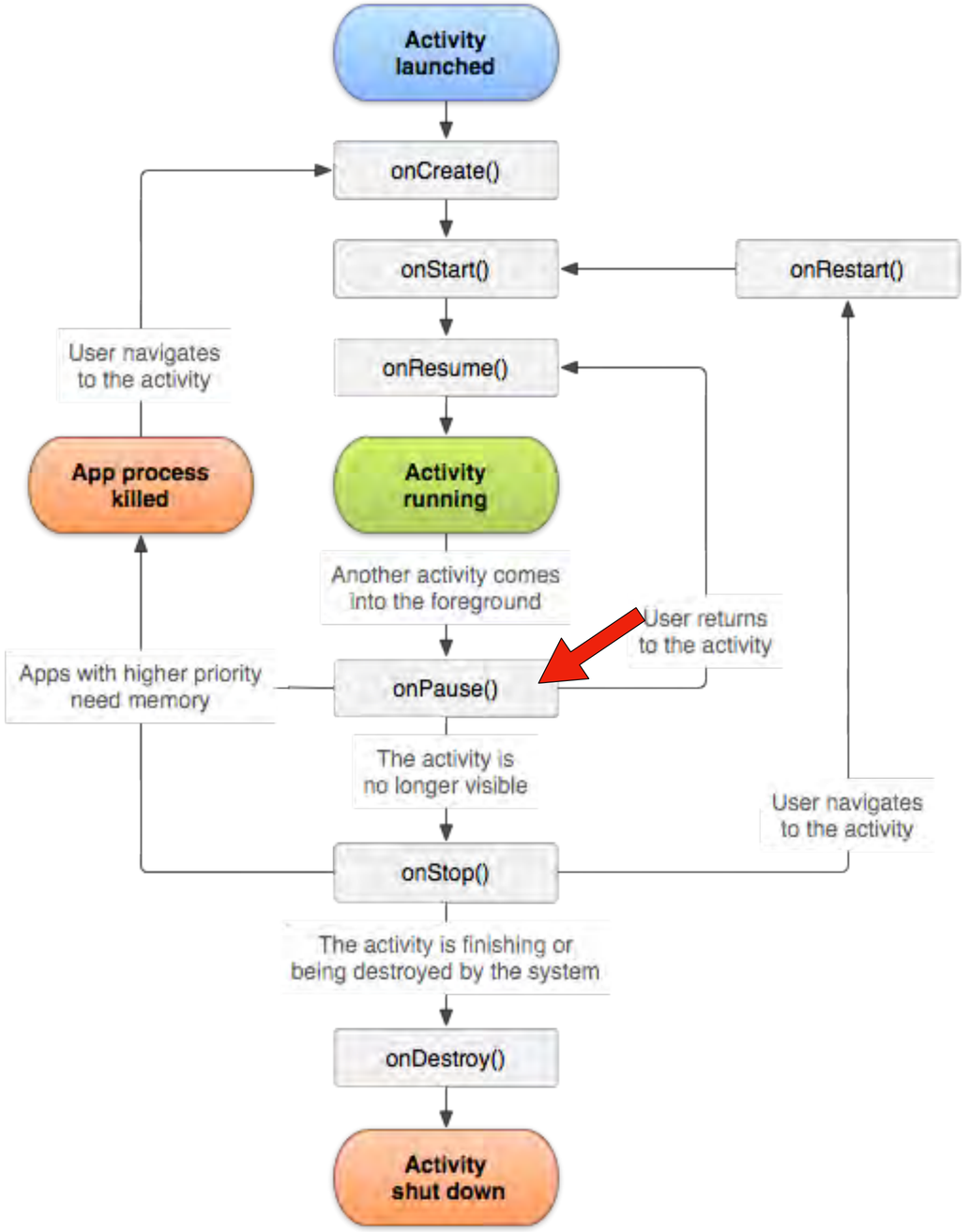
https://developer.android.com/guide/components/activities/activity-lifecycle



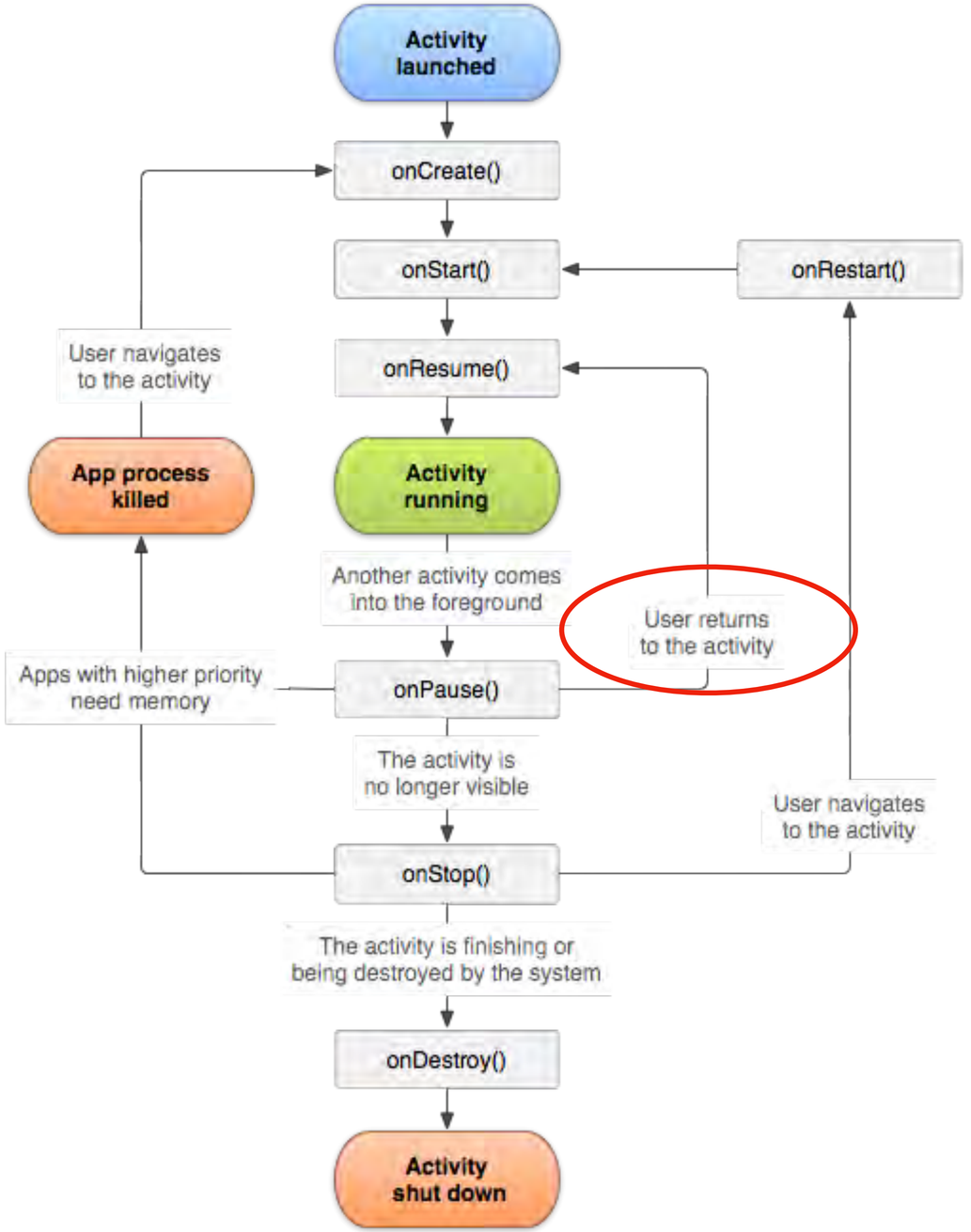
https://developer.android.com/guide/components/activities/activity-lifecycle



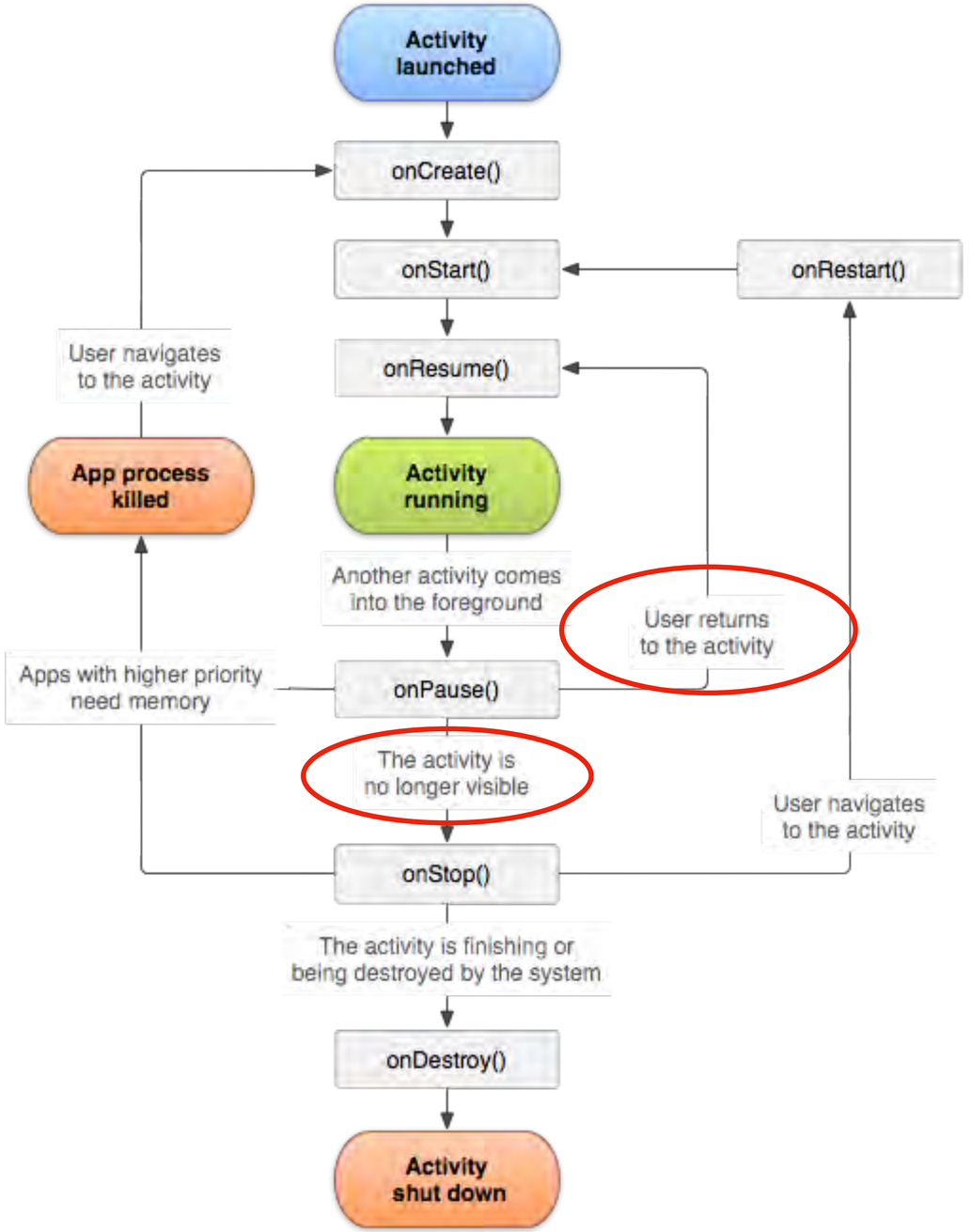
https://developer.android.com/guide/components/activities/activity-lifecycle



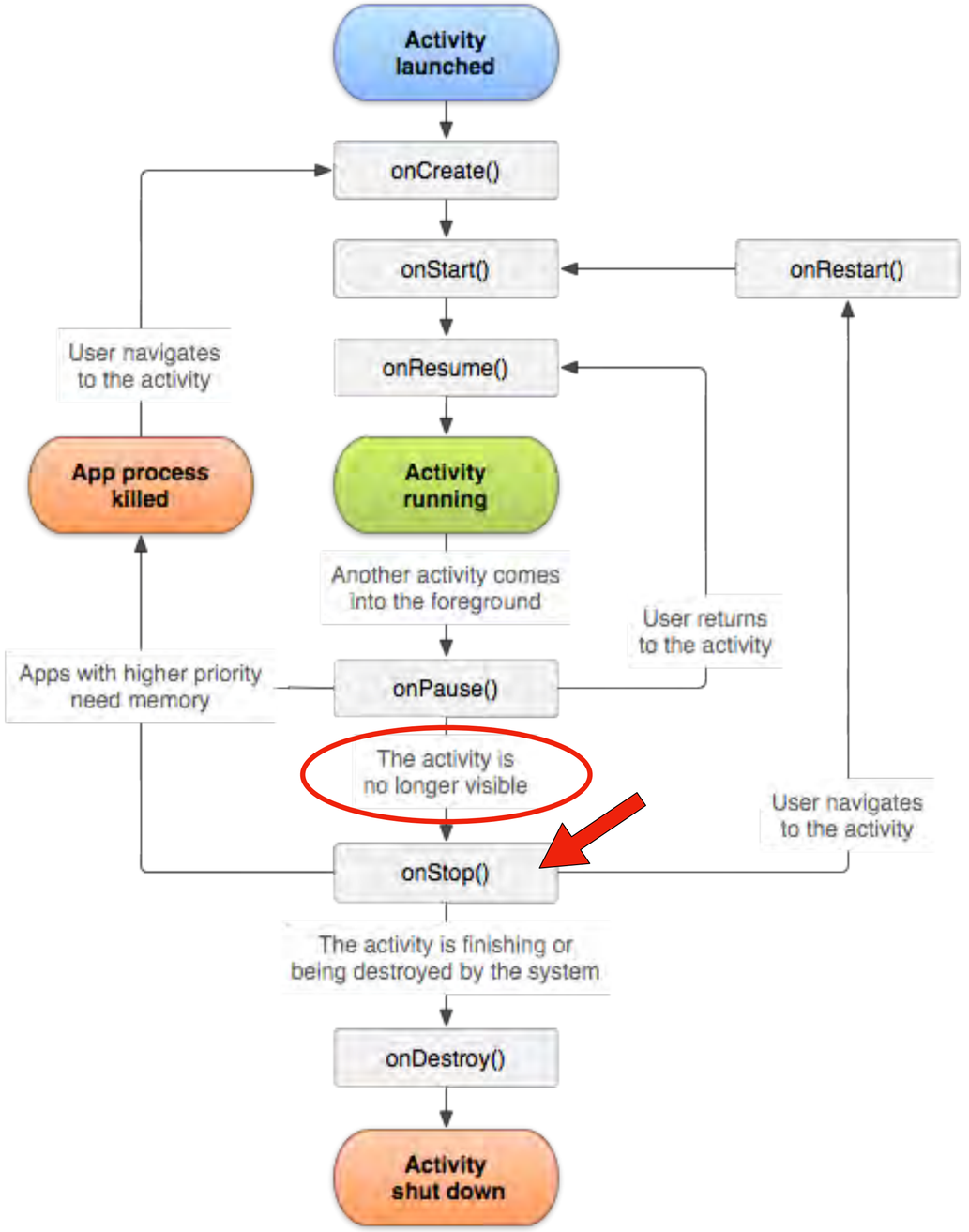
https://developer.android.com/guide/components/activities/activity-lifecycle



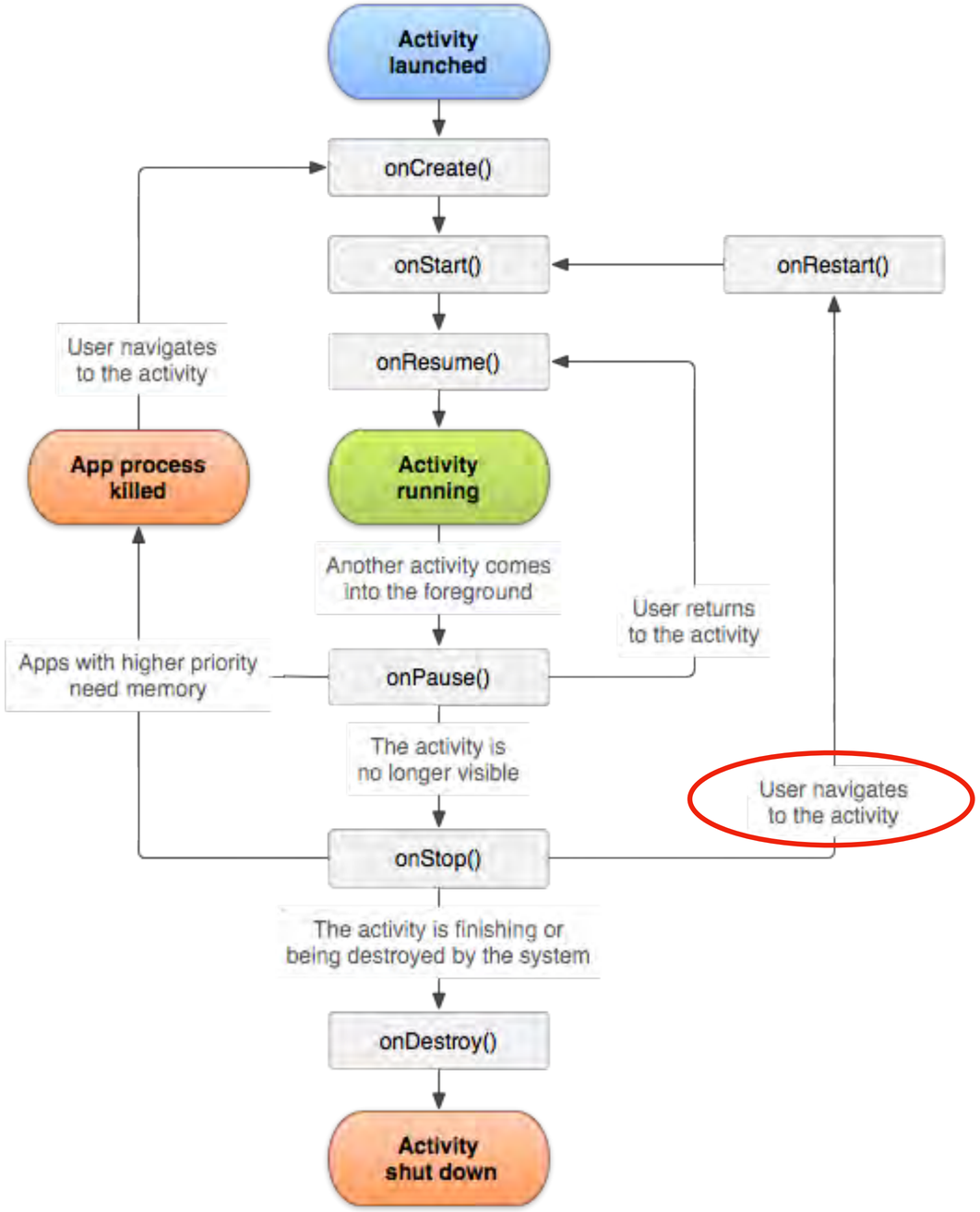
https://developer.android.com/guide/components/activities/activity-lifecycle



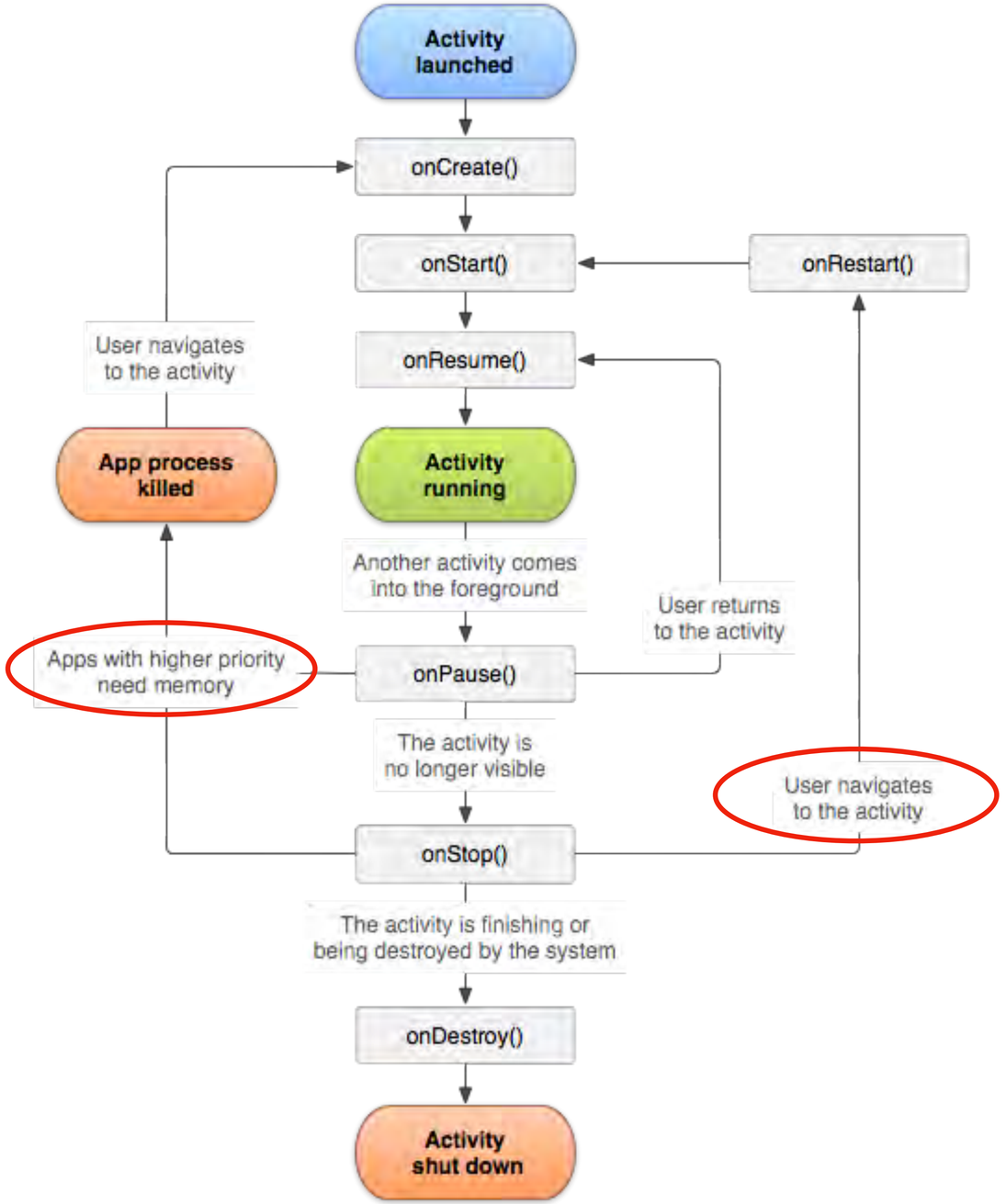
https://developer.android.com/guide/components/activities/activity-lifecycle



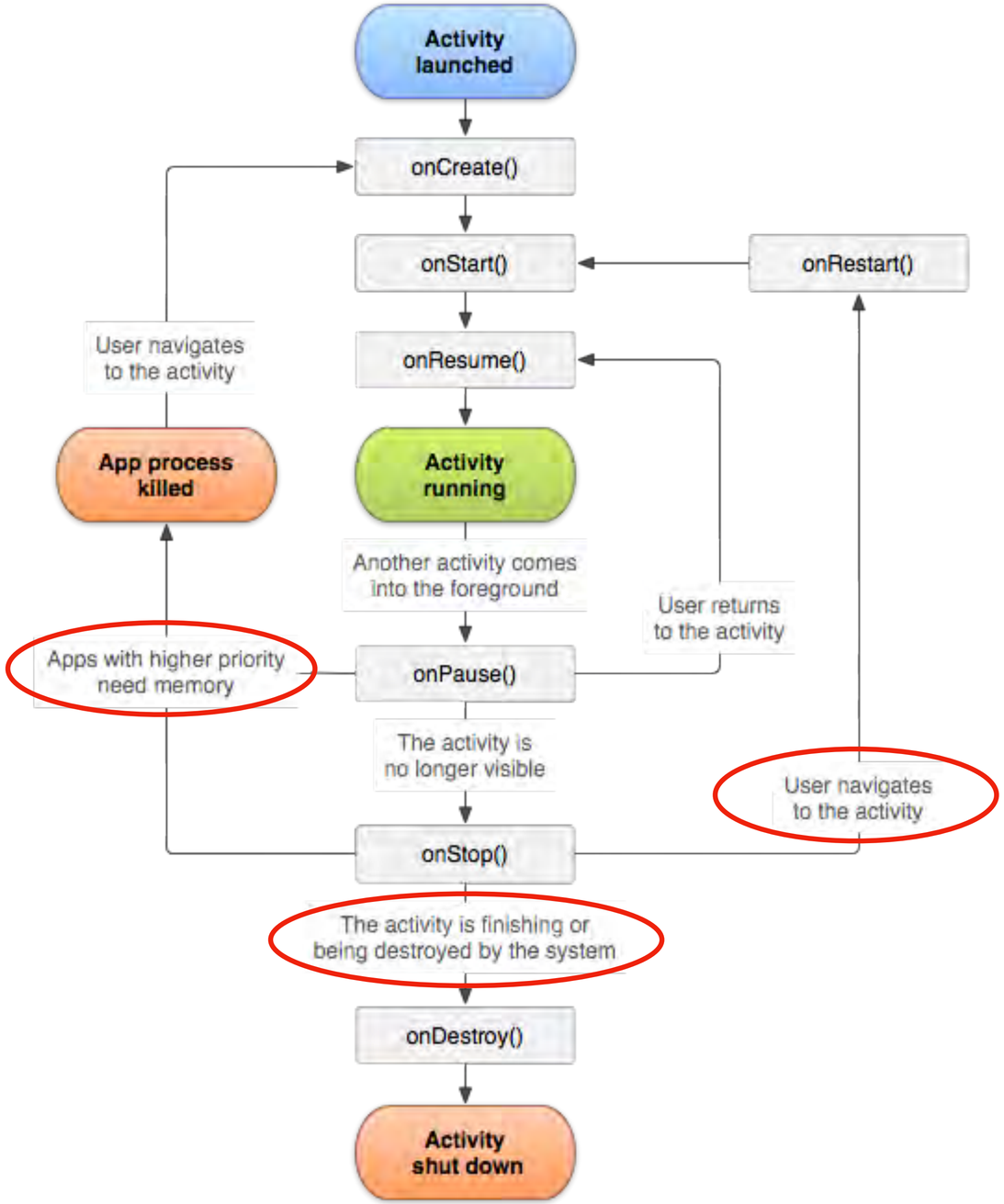
https://developer.android.com/guide/components/activities/activity-lifecycle



https://developer.android.com/guide/components/activities/activity-lifecycle



https://developer.android.com/guide/components/activities/activity-lifecycle



https://developer.android.com/guide/components/activities/activity-lifecycle

iOS Lifecycle

- Very similar to Android
  - *onCreate on Android → viewDidLoad on iOS*
  - *onStart → viewWillAppear*
  - *onResume → viewDidAppear*
  - *onPause → viewWillDisappear*
  - *onStop → viewDidDisappear*
  - *onRestart() & onDestroy → no true equivalent*
- iOS app may enter a "Suspended" state after being in the background
  - *remains in memory but is not executing code*
  - *Android may keep the process but doesn't have an explicit "Suspended" state*

Modern Android Development

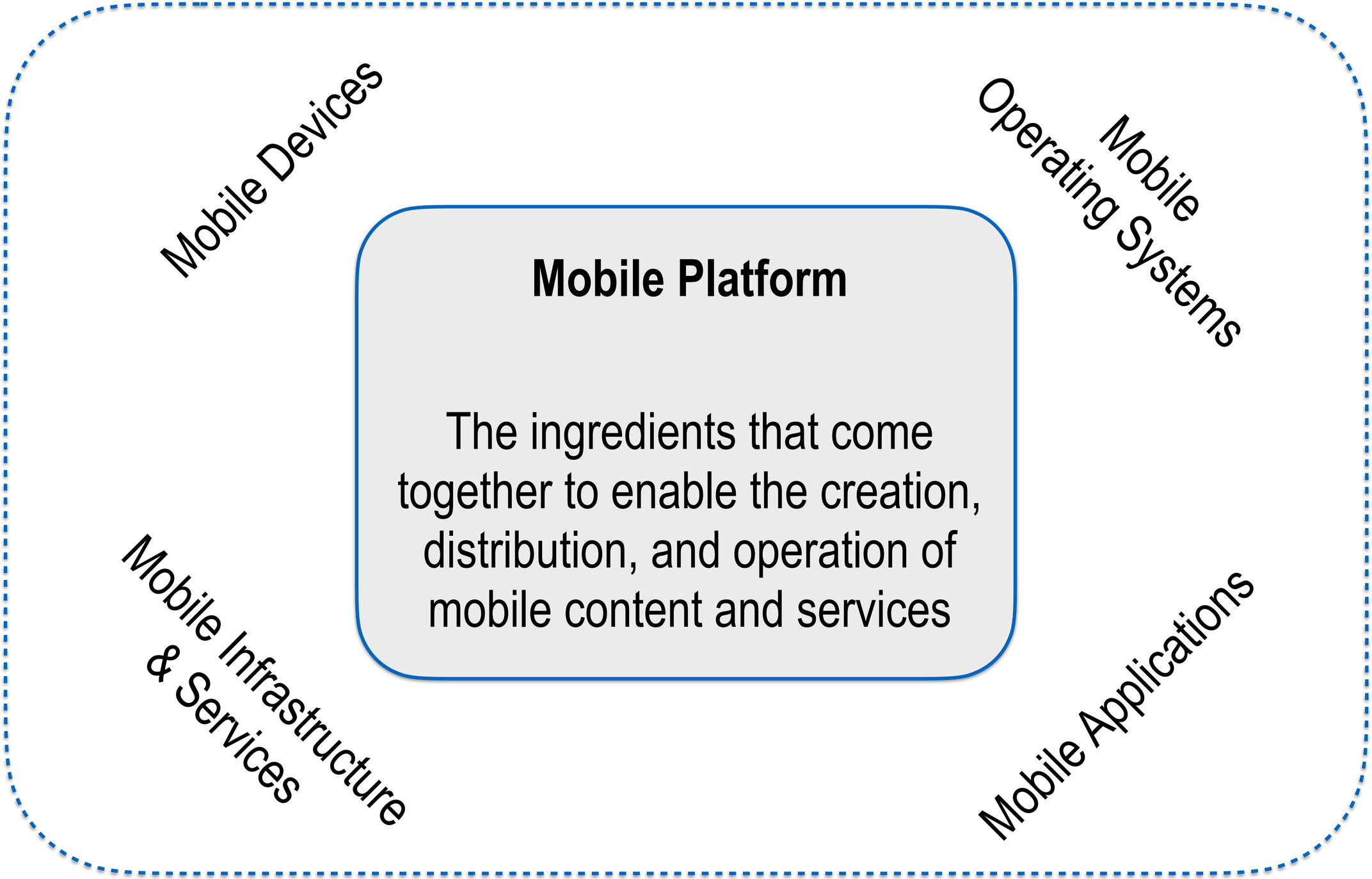
- Kotlin = primary programming lang for Android
  - *concise syntax*
  - *null safety features*
  - *coroutines for asynchronous programming*
  - *extension functions instead of inheritance*

```kotlin
fun String.isEmailValid(): Boolean {
    val emailPattern = Regex(
        "^[A-Za-z0-9._%+-]+@"+
        "[A-Za-z0-9.-]+\\.[A-Za-z]{2,6}$"
    )
    return emailPattern.matches(this)
}
```

- Jetpack
  - *libraries and tools that provide abstractions to reduce boilerplate code*
  - *components for architecture, UI, behavior, and foundations*
  - *e.g., Jetpack Compose for UI rendering and UI state management — works hand-in-hand with ViewModel for state, Navigation for screen switching, Room for data*

Cross-Platform Mobile Development

Framework	Language	Platforms	Notes
Flutter	Dart	Android, iOS, Web, Desktop	Major performance gains (Impeller engine), unified UI
React Native	JavaScript / TS	Android, iOS	New architecture (Fabric, TurboModules), huge ecosystem
Kotlin Multiplatform Mobile (KMM)	Kotlin	Android (native), iOS	Share logic, write UI per platfor
Jetpack Compose Multiplatform	Kotlin	Android, iOS, Desktop, Web	Declarative UI shared via Kotlin; Compose-first approach
.NET MAUI	C#	Android, iOS, macOS, Windows	Successor to Xamarin; XAML UI, good for .NET devs


Mobile Devices
Mobile Operating Systems
Mobile Platform
The ingredients that come together to enable the creation, distribution, and operation of mobile content and services
Mobile Infrastructure & Services
Mobile Applications