Accelerating App Startup with the Jetpack Startup Library

John Li

@tbfJohn

in/JohnLeeroy

Overview

- What is App Startup?
- Android Vitals
- Jetpack App Startup Library
- Parallelization Performance Pattern
- Codelab Walkthrough
- Course Summary

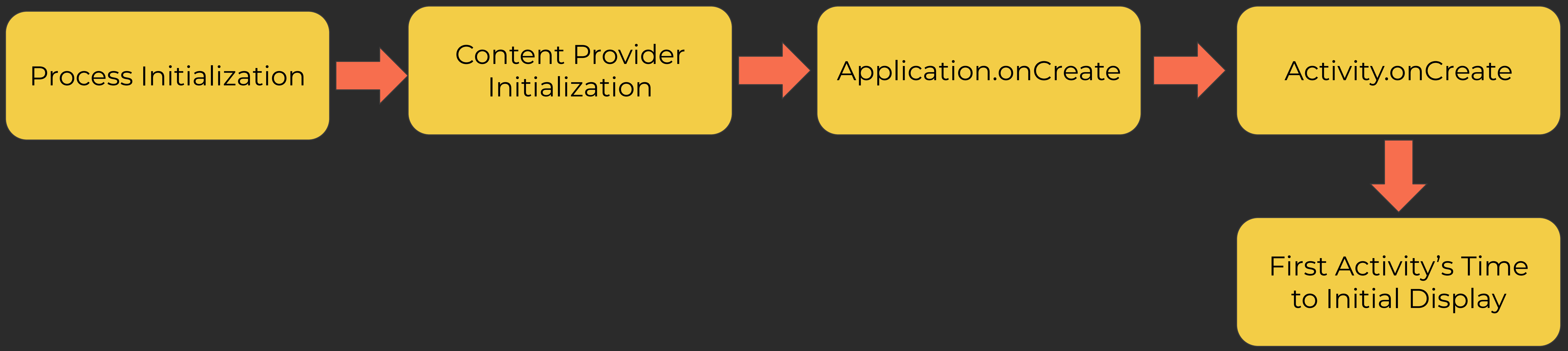
What is App Startup?

Application.onCreate

```kotlin
class MyApplication: Application() {

   fun onCreate(name: String) {
       super.onCreate()
       // Startup code
   }
}
```

App Startup Process


Process Initialization
Content Provider Initialization
Application.onCreate
Activity.onCreate
First Activity's Time to Initial Display

Android Vitals

- User-perceived Crash Rate
- User-perceived ANR
- Slow Rendering
- **App Startup Duration**
- Excessive Wakeups
- and more

App Startup Thresholds

- Cold startup < 5s
- Warm startup < 2s
- Hot startup < 1.5s

Jetpack App Startup Library

Up Next

Jetpack App Startup Features

- Provides a simple API to manage initializing components
- Provides a form of control with initializing content providers
- Intelligently manage the order of how components initialize based on dependencies
- Automatically or manually manage initialization code

Initializer

```
interface Initializer {
 fun create(context: Context): T
 fun dependencies(): (Mutable)List<Class<Initializer<Any!>!>!>
}
```

Initializer Example

```
class RoomDatabaseInitializer: Initializer {
    override fun create(context: Context): AppDatabase {
        val db = Room.databaseBuilder(
            context,
            AppDatabase::class.java,
            "database-name"
        ).build()
       return db
    }

    override fun dependencies(): (Mutable)List<Class<Initializer<Any!>!>!> {
        emptyList()
    }
}
```

Content Provider in AndroidManifest.xml

```xml
<application>
 <provider
     android:name="androidx.startup.InitializationProvider"
     android:authorities="${applicationId}.androidx-startup"
     android:exported="false"
     tools:node="merge">
     <!-- This entry makes RoomDatabaseInitializer discoverable. -->
     <meta-data  android:name="com.example.RoomDatabaseInitializer"
           android:value="androidx.startup" />
 </provider>
</application>
```

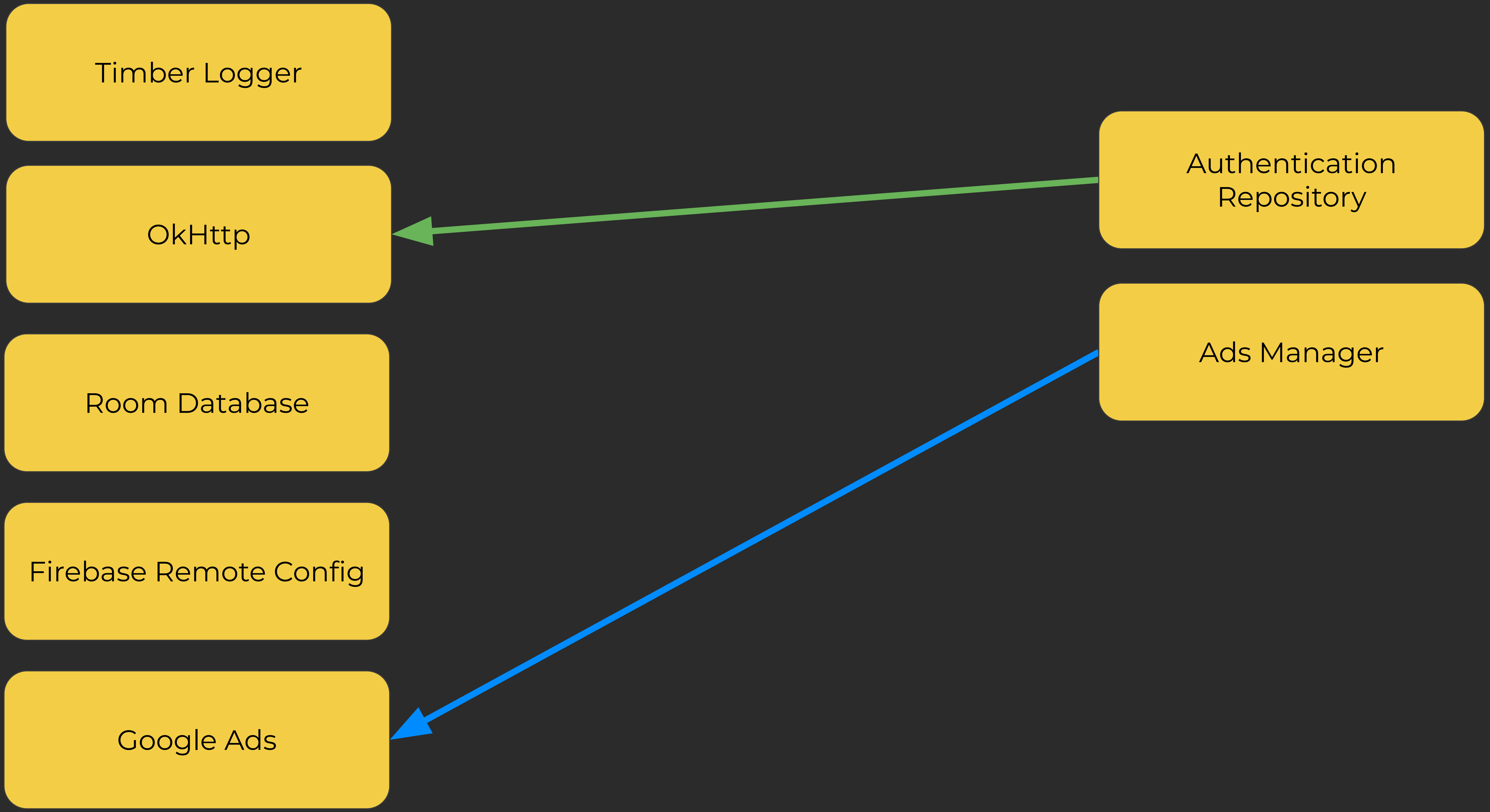
Parallelization Performance Pattern

Up Next

App Startup Work Example

Dagger	Authentication Repository	Firebase Remote Config	Room Database
Google Ads	Ads Manager	OkHttp	Timber Logger

Dependency Flow



Categorize by Dependency Count

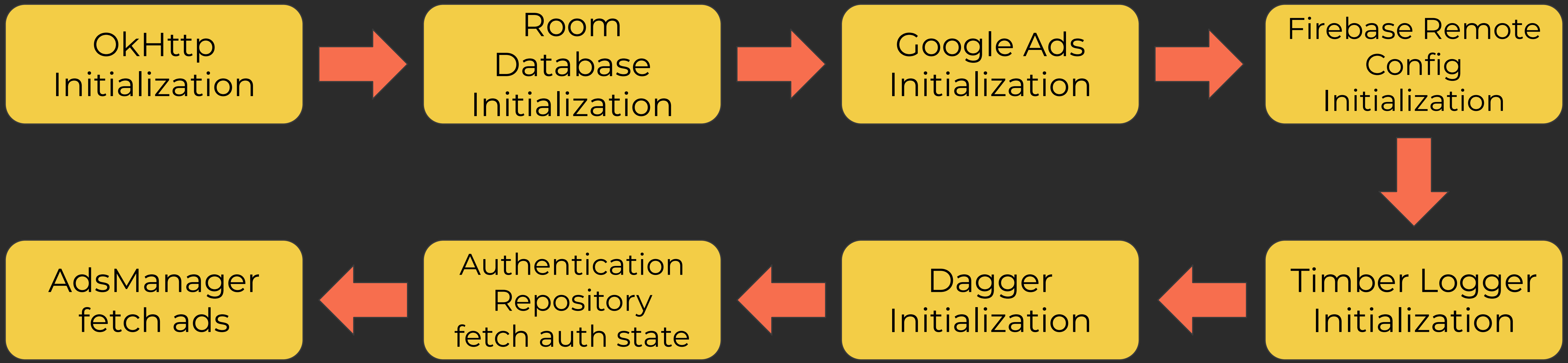
No Dependencies

- Google Ads
- OkHttp
- Firebase Remote Config
- Room Database
- Timber Logger

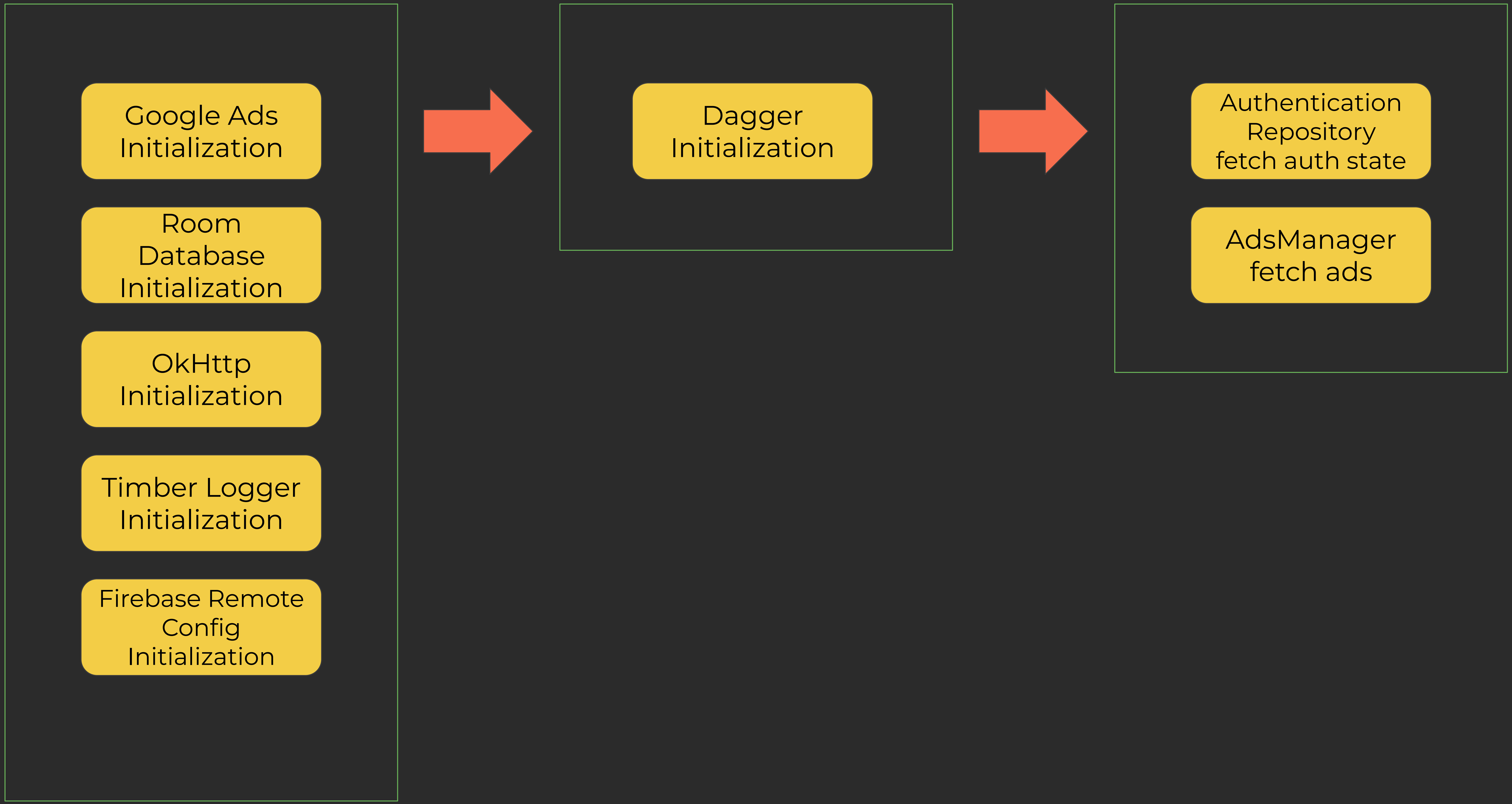
Has 1+ Dependencies

- Authentication Repository
- Ads Manager

Sequential Startup Timeline


OkHttp Initialization
Room Database Initialization
Google Ads Initialization
Firebase Remote Config Initialization
Timber Logger Initialization
Dagger Initialization
Authentication Repository fetch auth state
AdsManager fetch ads

Parallelized Startup Timeline


Google Ads Initialization
Room Database Initialization
OkHttp Initialization
Timber Logger Initialization
Firebase Remote Config Initialization
Dagger Initialization
Authentication Repository fetch auth state
AdsManager fetch ads

Codelab Walkthrough

Up Next

Parallelizing Initializers

```
fun List<Initializer>.parallelInit(context:
Context) {
  runBlocking {
        map { initializer ->
            async(Dispatchers.Default) {
              initializer.create(context)
            }
        }.awaitAll()
    }
}
```

Parallelizing Initializers

- **Extension method Initializer**

```
fun List<Initializer>.parallelInit(context:
Context) {
   runBlocking {
          map { initializer ->
               async(Dispatchers.Default) {
                  initializer.create(context)
               }
          }.awaitAll()
    }
}
```

Parallelizing Initializers

- Extension method Initializer
- **Runs on the current thread**

```
fun List<Initializer>.parallelInit(context:
Context) {
   runBlocking {
         map { initializer ->
              async(Dispatchers.Default) {
                 initializer.create(context)
              }
         }.awaitAll()
     }
}
```

Parallelizing Initializers

- Extension method Initializer
- Runs on the current thread
- **For every initializer, call create in the default thread pool**

```
fun List<Initializer>.parallelInit(context:
Context) {
   runBlocking {
         map { initializer ->
              async(Dispatchers.Default) {
                 initializer.create(context)
              }
         }.awaitAll()
     }
}
```

droidcon academy

Parallelizing Initializers

- Extension method Initializer
- Runs on the current thread
- For every initializer, call create in the default thread pool
- **Block until all work is done**

```
fun List<Initializer>.parallelInit(context:
Context) {
   runBlocking {
          map { initializer ->
               async(Dispatchers.Default) {
                  initializer.create(context)
               }
          }.awaitAll()
     }
}
```

Parallel Init in Application.onCreate

```
class MyApplication: Application() {
   val roomDbInitializer = RoomInitializer()

   val zeroDependencyInitializers = listOf<Initializer>(roomDbInitializer, ...)


   fun onCreate(name: String) {
       super.onCreate()
      zeroDependencyInitializers.parallelInit(this)
    }
}
```

Parallel Init in Application.onCreate

```
class MyApplication: Application() {
   val roomDbInitializer = RoomInitializer()

   val zeroDependencyInitializers = listOf<Initializer>(roomDbInitializer, …)


   fun onCreate(name: String) {
       super.onCreate()
      zeroDependencyInitializers.parallelInit(this)
    }
}
```

droidcon academy

Course Summary

Up Next

Summary

- App startup contains many parts such as
  - Process Initialization
  - ContentProvider initialization
  - Application.onCreate
  - Activity.onCreate
  - First Frame Rendered
- Android Vitals provides app quality metrics
- App Startup Thresholds
  - Cold startup < 5s
  - Warm startup < 2s
  - Hot startup < 1.5s

Summary (cont.)

- Jetpack App Startup Library enables smart startup flows
  - Initializer interface
  - InitializationProvider
  - AppInitializer.initializeComponent
- Parallelize the initialization of components with zero dependencies