

Tuning Android app performance for 150 MILLION devices

by Ryan Clements, Owner of Byte Bot 



Home

News by Fire TV

Press ⚙ to manage

Enjoy customized news from multiple providers and categories for free. Just say "Alexa, play the news".

YOUR APPS & CHANNELS



SPONSORED

A thumbnail for the TV show "fixer upper Welcome Home" featuring Chip and Joanna Gaines. To the left is the HGTV logo. To the right are buttons for "start free trial" and the "discovery+" logo.

prime AMAZON ORIGINALS



My News Live Trending Categories Preferences

MY CHANNELS



LIVE

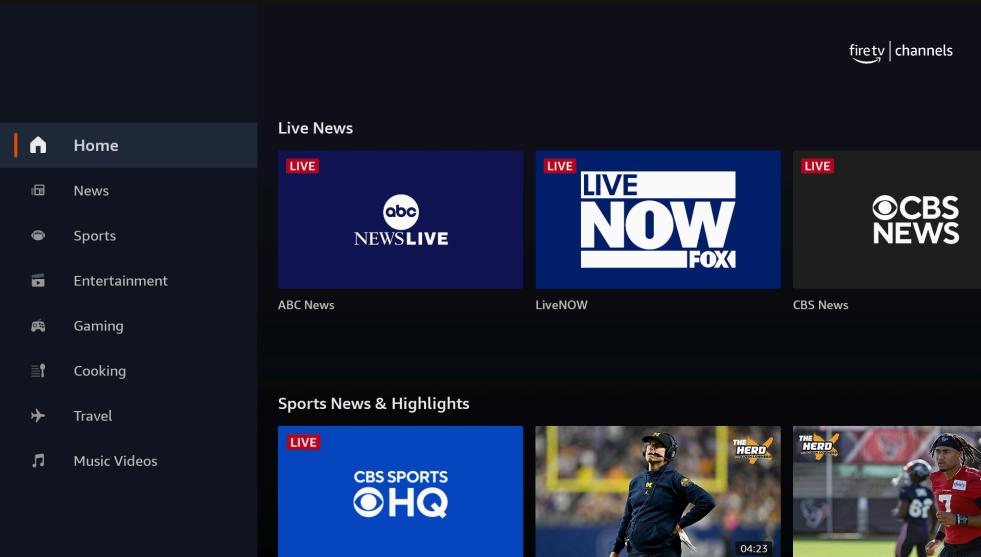


MISSION JURASSIC: SEARCHING FOR DINOSAUR BONES
NOW PLAYING
TEAM OF SCIENTISTS HAVE BEGUN EXCAVATING A TREASURE TROVE OF NEW DINOSAUR FOSSILS IN WYOMING

LIVE
CBSN

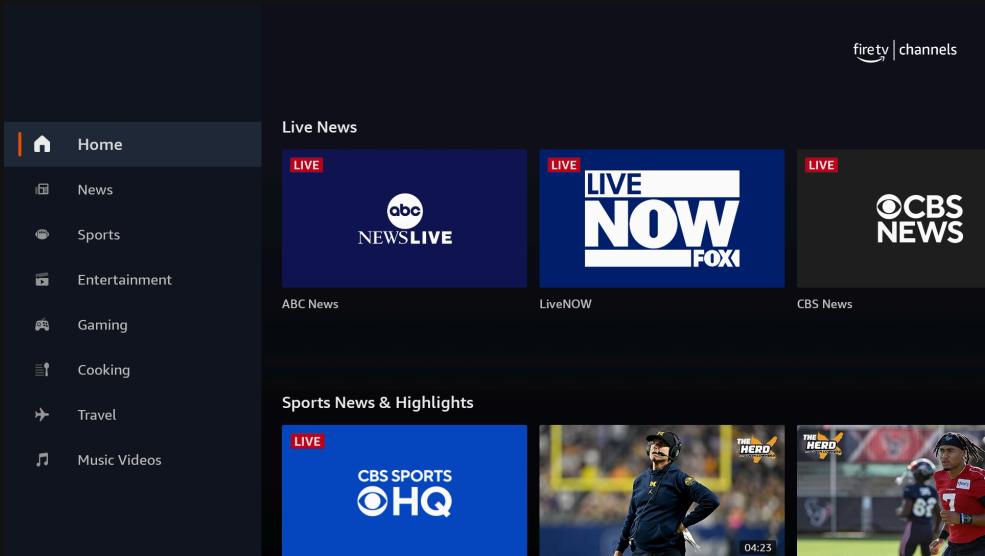
We wanted...

The same app, but genericized to other types of content like



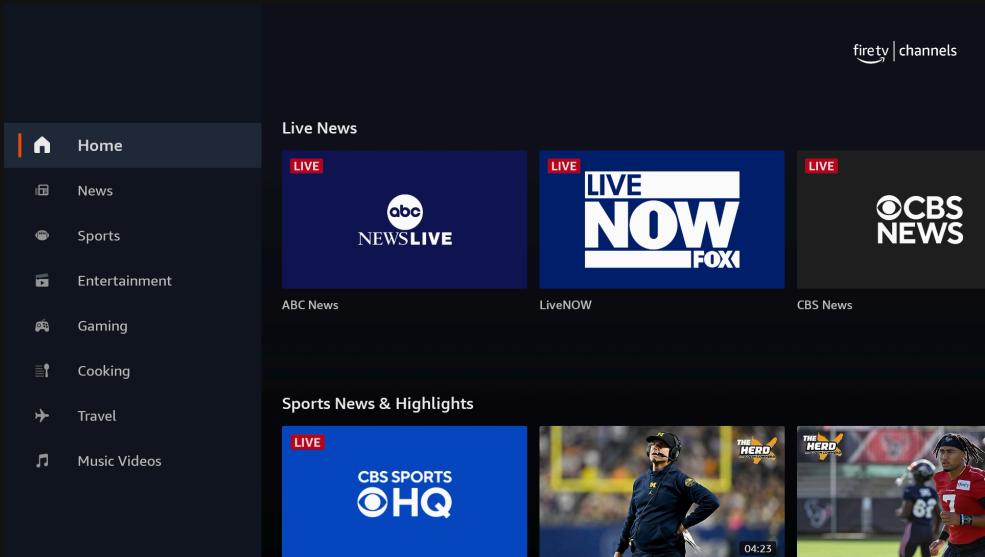
We wanted...

The same app, but genericized to other types of content like



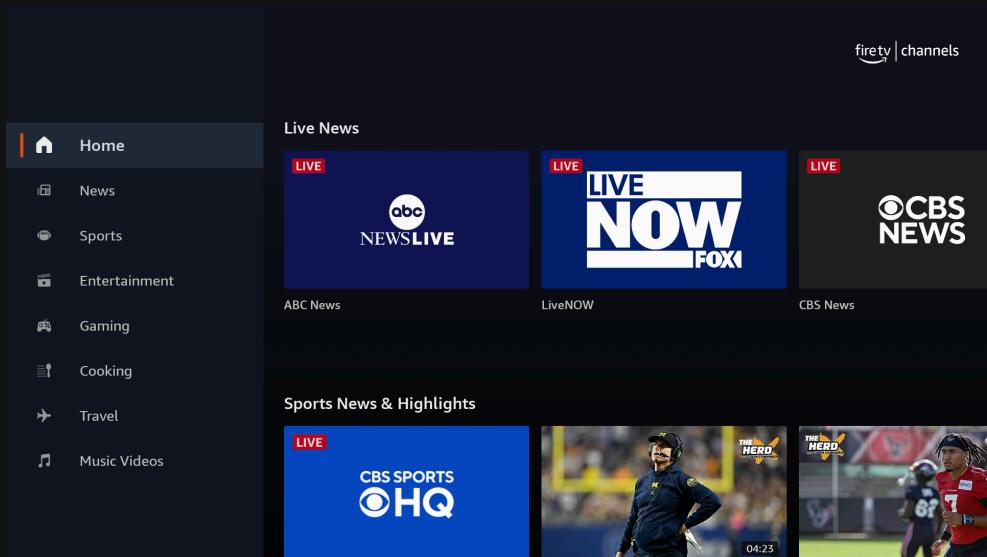
We wanted...

The same app, but genericized to other types of content like



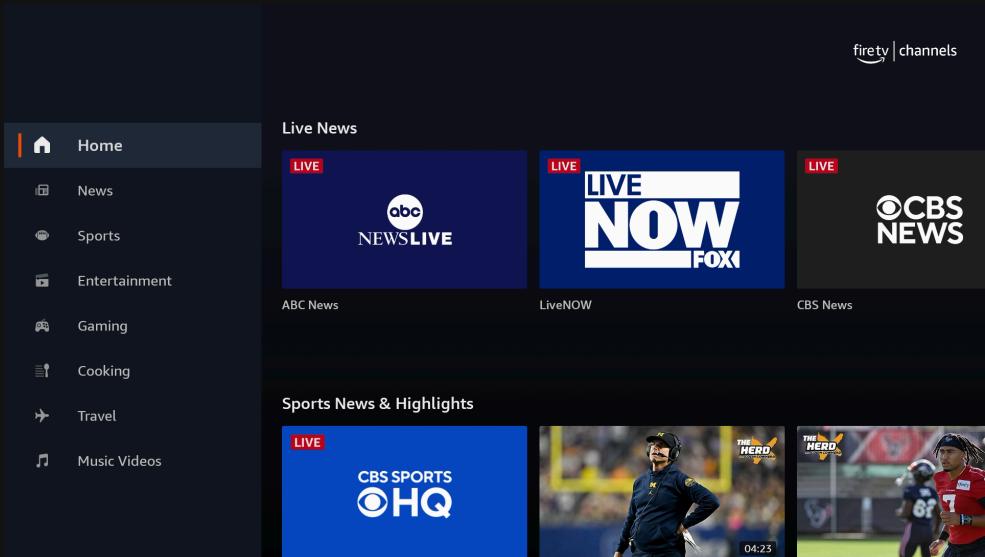
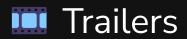
We wanted...

The same app, but genericized to other types of content like



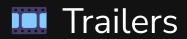
We wanted...

The same app, but genericized to other types of content like

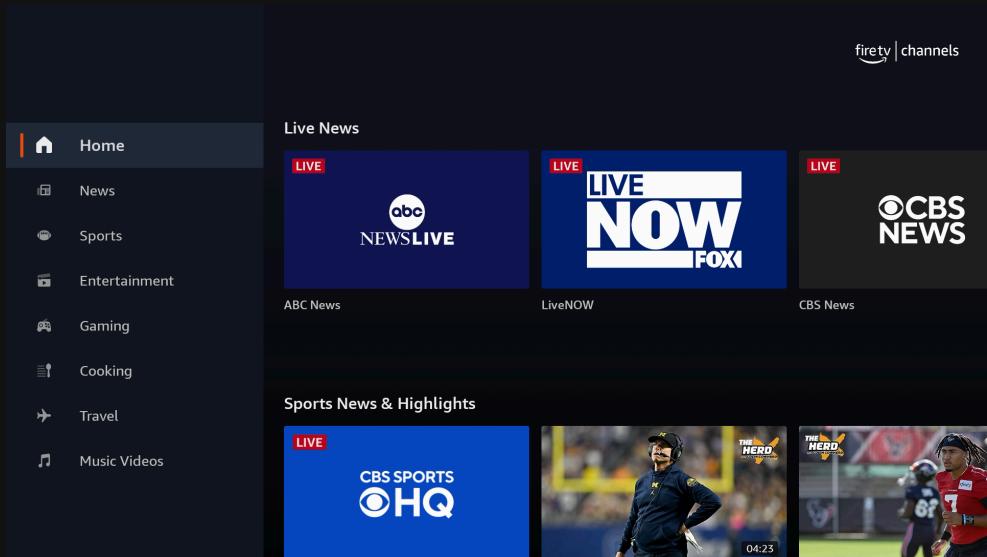


We wanted...

The same app, but genericized to other types of content like



...





MakeA**GIF**.com



Who am I?



Who am I?

Ryan Clements, Owner of Byte Bot 🤖



Who am I?

Ryan Clements, Owner of Byte Bot 🤖

a software agency that helps teams ship full stack solutions to their users through **training** and **consulting** 🚀



Who am I?

Ryan Clements, Owner of Byte Bot 🤖

a software agency that helps teams ship full stack solutions to their users through **training** and **consulting** 🚀



Who am I?

Ryan Clements, Owner of Byte Bot 🤖

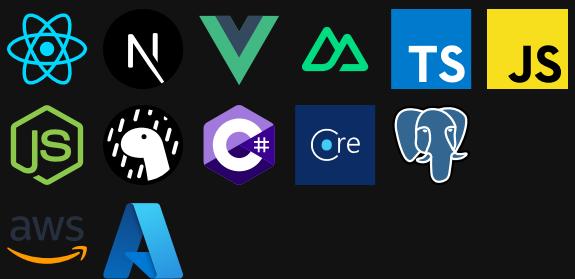
a software agency that helps teams ship full stack solutions to their users through **training** and **consulting** 🚀



Who am I?

Ryan Clements, Owner of Byte Bot 🤖

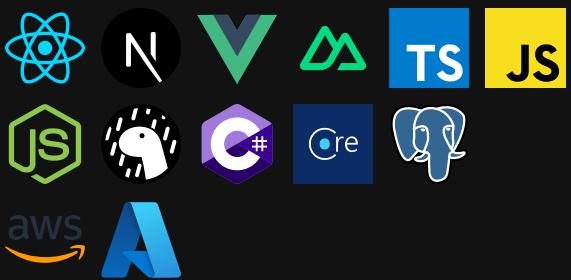
a software agency that helps teams ship full stack solutions to their users through **training** and **consulting** 🚀



Who am I?

Ryan Clements, Owner of Byte Bot 🤖

a software agency that helps teams ship full stack solutions to their users through **training** and **consulting** 🚀



📅 calendly.com/byte-bot

✉️ info@bytebot.io

🐦 [@RyanClementsHax](https://twitter.com/RyanClementsHax)

🔗 bytebot.beehiiv.com/subscribe

🔗 linkedin.com/in/ryan-clements-hax



Our first findings were...low tech

Lesson #1: Control your (de)serialization



REFLECTION



```
public class ItemSerializer extends StdSerializer<Item> {

    public ItemSerializer() {
        this(null);
    }

    public ItemSerializer(Class<Item> t) {
        super(t);
    }

    @Override
    public void serialize(
        Item value,
        JsonGenerator jgen,
        SerializerProvider provider
    ) throws IOException, JsonProcessingException {
        jgen.writeStartObject();
        jgen.writeNumberField("id", value.id);
        jgen.writeStringField("itemName", value.itemName);
        jgen.writeNumberField("owner", value.owner.id);
        jgen.writeEndObject();
    }
}
```

```
Item myItem = new Item(1, "theItem", new User(2, "theUser"));

ObjectMapper mapper = new ObjectMapper();

SimpleModule module = new SimpleModule();
module.addSerializer(Item.class, new ItemSerializer());
mapper.registerModule(module);

String serialized = mapper.writeValueAsString(myItem);
```


50%!

Lesson #2: Keep your dependencies up to date

1000 foot view of the app

1000 foot view of the app

1. Create a separate app per genre 

1000 foot view of the app

1. Create a separate app per genre 
2. Rewrite the existing app 

1000 foot view of the app

1. Create a separate app per genre ✗
2. Rewrite the existing app ✗
3. Create a new app that had its UI configured from the backend ✓

1000 foot view of the app

1. Create a separate app per genre ✗
2. Rewrite the existing app ✗
3. Create a new app that had its UI configured from the backend ✓

...but it has to be Trojan horse'd into the News app's apk... 😬

1000 foot view of the app

1. Create a separate app per genre ✗
2. Rewrite the existing app ✗
3. Create a new app that had its UI configured from the backend ✓

...but it has to be Trojan horse'd into the News app's apk... 😊



Behold! It's technically functional

Here is a rough architecture of the app.

Old parts in orange

New parts in green



Behold! It's technically functional

Here is a rough architecture of the app.

Old parts in orange

New parts in green

Ugly parts in red 🐈



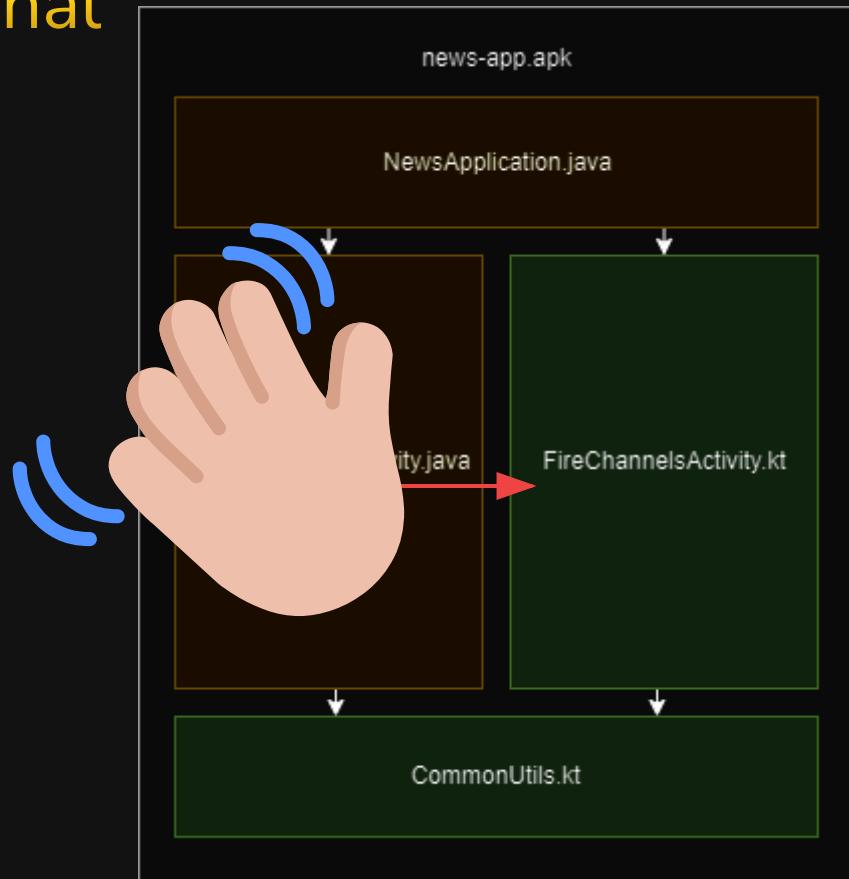
Behold! It's technically functional

Here is a rough architecture of the app.

Old parts in orange

New parts in green

Ugly parts in red 🤢



Our poor app

Our poor app



Kotlin 1.3

Our poor app

- 😢 Kotlin 1.3

- ☠️ Android support libraries

Our poor app

 Kotlin 1.3

 Android support libraries

 Gradle 3

Our poor app

 Kotlin 1.3

 Android support libraries

 Gradle 3.0

Nice things we can't have

Nice things we can't have

✗ Flow

Nice things we can't have

✗ Flow

✗ androidx.anything (including compose 😭)

Nice things we can't have

- ✖ Flow
- ✖ androidx.anything (including compose 😭)
- ✖ Anything that needs androidx

Nice things we can't have

- ✗ Flow
- ✗ androidx.anything (including compose 😭)
- ✗ Anything that needs androidx
- ✗ Android Studio

Nice things we can't have

- ✗ Flow
- ✗ androidx.anything (including compose 😭)
- ✗ Anything that needs androidx
- ✗ Android Studio
- ⚠ Databinding

Nice things we can't have

- ✗ Flow
- ✗ androidx.anything (including compose 😭)
- ✗ Anything that needs androidx
- ✗ Android Studio
- ⚠ Databinding
- ⚠ Documentation

So what serializers can we use?

So what serializers can we use?

✗ kotlinx.serialization

So what serializers can we use?

- ✗ kotlinx.serialization
- ✗ Moshi

So what serializers can we use?

- ✗ kotlinx.serialization
- ✗ Moshi
- ✓ Jackson



REFLECTION



Lesson #3: Use profiling tooling

 Developers Essentials ▾ Design & Plan ▾ Develop ▾ Google Play Community

Search

APP QUALITY

Overview Core value User experience **Technical quality** Privacy & Security

Overview

Performance

App performance guide

- ▶ Inspecting performance
- ▶ Improving performance
- ▶ Monitoring performance

On Google Play

Android vitals 

Healthy releases 

App performance guide

This guide provides an overview of libraries, tools, and best practices you can use to inspect, improve, and monitor performance on Android.

Users want apps to launch quickly, render smoothly, and require little memory and battery usage. This guide's sections provide information and insights into tools, libraries, and best practices that help you achieve better app performance.

Inspect performance

Learn about inspecting app performance during development.

[Learn more](#)

Improve performance

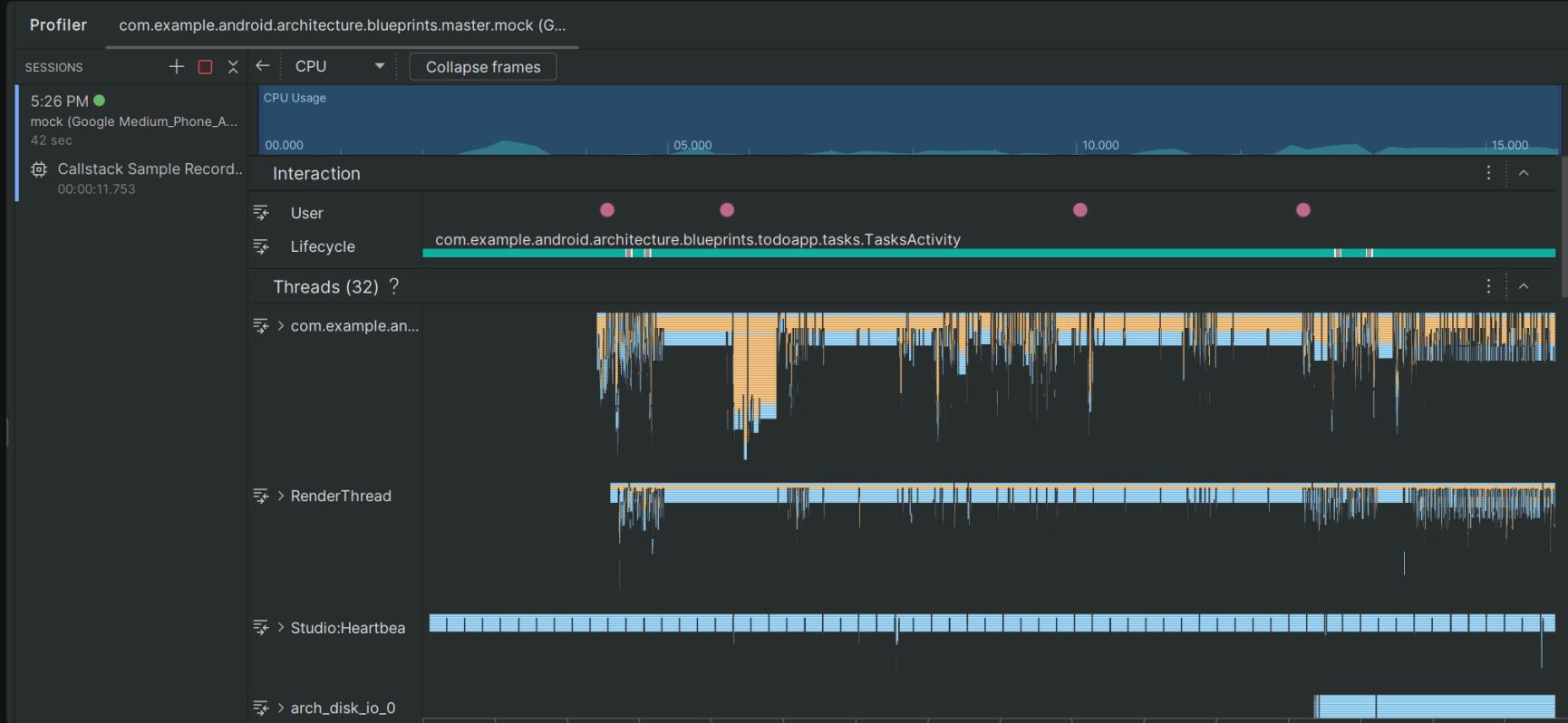
Improve app performance where it matters the most—in production.

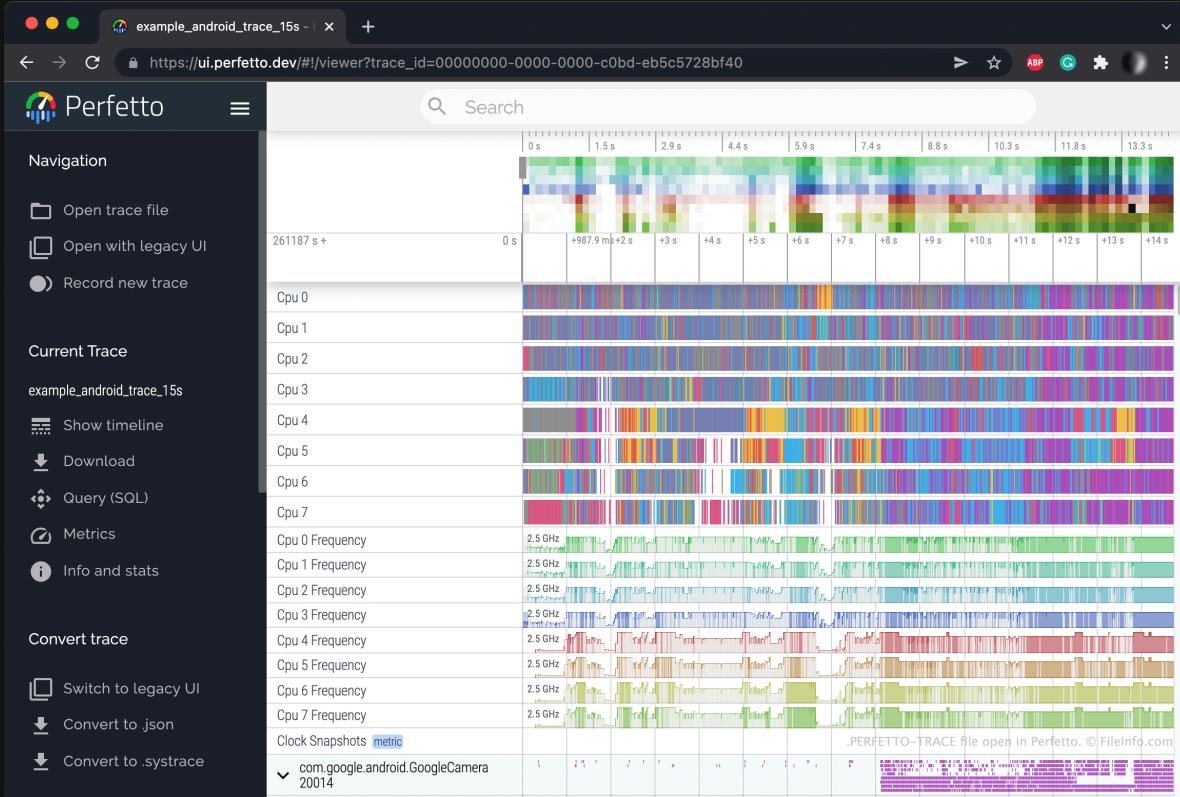
[Learn more](#)

Monitor performance

Monitor your app's performance in production to learn about potential bottlenecks.

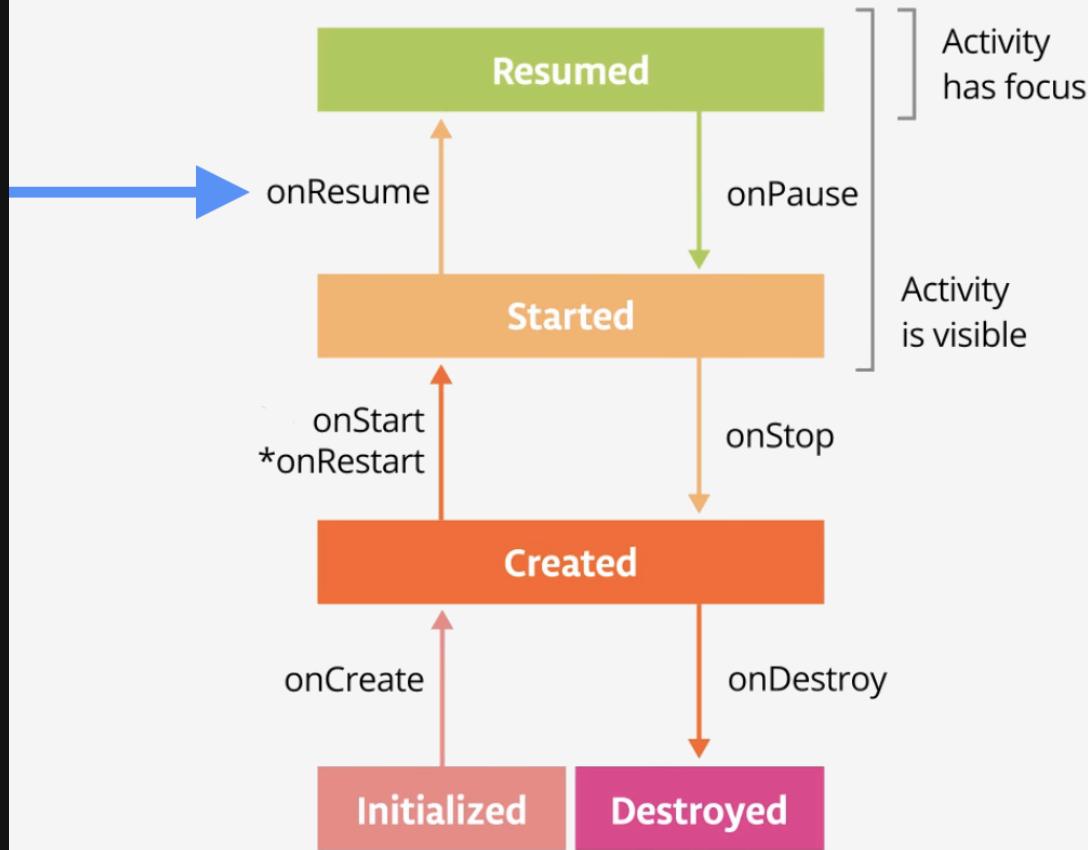
[Learn more](#)





Lesson #4: The startup path is sacred

The Activity Lifecycle



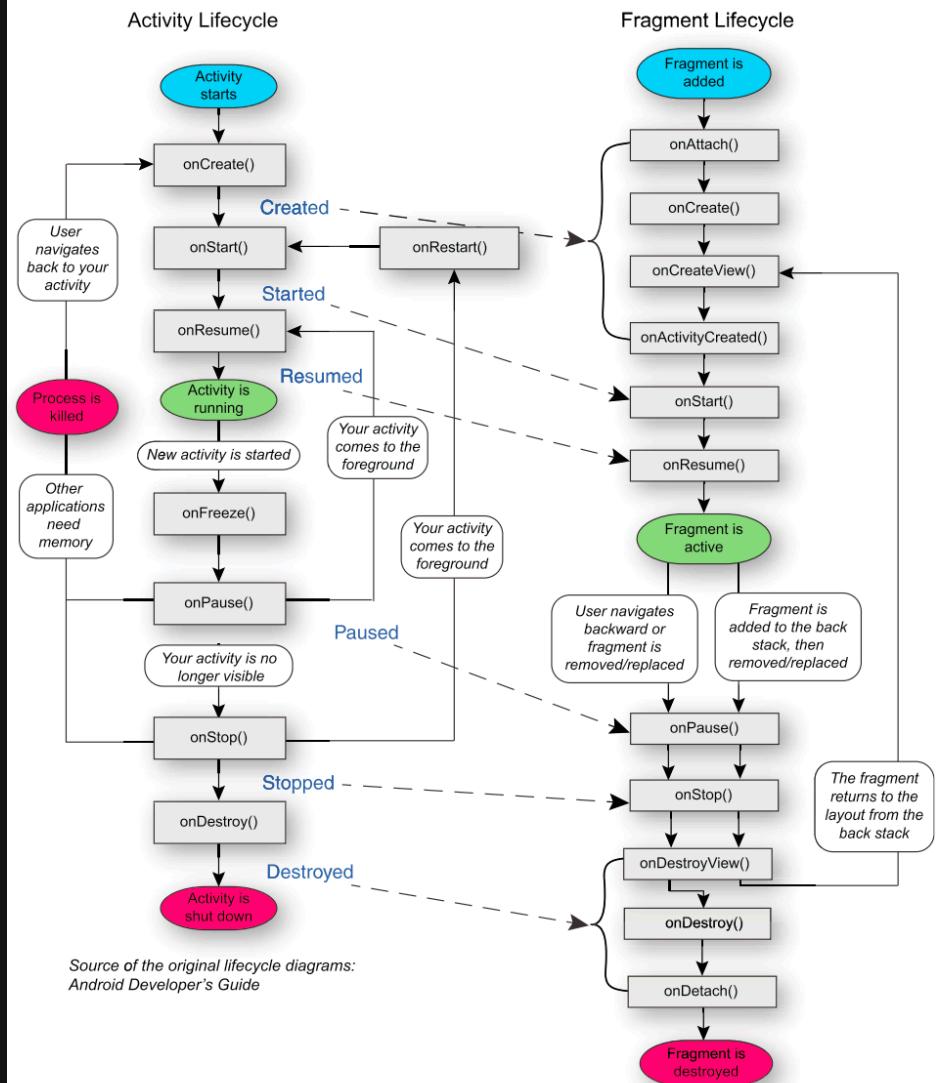


Figure 20.1 Activity and fragment lifecycles

More on my blog...

[Code spelunking: How to dive into unfamiliar code \(part 1\)](#)

Ever have problems making sense of code you didn't write? I'll help you with proven techniques in this post.



[Code spelunking: How to dive into unfamiliar code \(part 2\)](#)

Reading other people's code is hard. Here is how I do it.



```
class MyActivity : AppCompatActivity() {
    // Initialize DI
    val taskRepository: TasksRepository
        get() = ServiceLocator.provideTasksRepository(this)
    val appComponent: AppComponent by lazy {
        DaggerAppComponent.factory().create(applicationContext)
    }
    override fun onCreate() {
        super.onCreate()
        // Initializing logging
        if (BuildConfig.DEBUG) {
            Timber.plant(DebugTree());
        } else {
            Timber.plant(CrashReportingTree());
        }
        // Inflating views
        setContentView(R.layout.main);
        // Registering broadcast receivers
        val filter = IntentFilter();
        intentFilter.addAction(packageName + "android.net.conn.CONNECTIVITY_CHANGE");
        registerReceiver(MyReceiver(), filter);
        // Kicking off initial tasks
        taskRepository.fetchInitialData();
        appComponent.scheduler.queueBackgroundTasks();
        // ... and so on ...
    }
}
```

```
class MyActivity : AppCompatActivity() {
    // Initialize DI
    val taskRepository: TasksRepository
        get() = ServiceLocator.provideTasksRepository(this)
    val appComponent: AppComponent by lazy {
        DaggerAppComponent.factory().create(applicationContext)
    }
    override fun onCreate() {
        super.onCreate()
        // Initializing logging
        if (BuildConfig.DEBUG) {
            Timber.plant(DebugTree());
        } else {
            Timber.plant(CrashReportingTree());
        }
        // Inflating views
        setContentView(R.layout.main);
        // Registering broadcast receivers
        val filter = IntentFilter();
        intentFilter.addAction(packageName + "android.net.conn.CONNECTIVITY_CHANGE");
        registerReceiver(MyReceiver(), filter);
        // Kicking off initial tasks
        taskRepository.fetchInitialData();
        appComponent.scheduler.queueBackgroundTasks();
        // ... and so on ...
    }
}
```

```
class MyActivity : AppCompatActivity() {
    // Initialize DI
    val taskRepository: TasksRepository
        get() = ServiceLocator.provideTasksRepository(this)
    val appComponent: AppComponent by lazy {
        DaggerAppComponent.factory().create(applicationContext)
    }
    override fun onCreate() {
        super.onCreate()
        // Initializing logging
        if (BuildConfig.DEBUG) {
            Timber.plant(DebugTree());
        } else {
            Timber.plant(CrashReportingTree());
        }
        // Inflating views
        setContentView(R.layout.main);
        // Registering broadcast receivers
        val filter = IntentFilter();
        intentFilter.addAction(packageName + "android.net.conn.CONNECTIVITY_CHANGE");
        registerReceiver(MyReceiver(), filter);
        // Kicking off initial tasks
        taskRepository.fetchInitialData();
        appComponent.scheduler.queueBackgroundTasks();
        // ... and so on ...
    }
}
```



```
class MyActivity : AppCompatActivity() {
    // Initialize DI
    val taskRepository: TasksRepository
        get() = ServiceLocator.provideTasksRepository(this)
    val appComponent: AppComponent by lazy {
        DaggerAppComponent.factory().create(applicationContext)
    }
    override fun onCreate() {
        super.onCreate()
        // Initializing logging
        if (BuildConfig.DEBUG) {
            Timber.plant(DebugTree());
        } else {
            Timber.plant(CrashReportingTree());
        }
        // Inflating views
        setContentView(R.layout.main);
        // Registering broadcast receivers
        val filter = IntentFilter();
        intentFilter.addAction(packageName + "android.net.conn.CONNECTIVITY_CHANGE");
        registerReceiver(MyReceiver(), filter);
        // Kicking off initial tasks
        taskRepository.fetchInitialData();
        appComponent.scheduler.queueBackgroundTasks();
        // ... and so on ...
    }
}
```

```
class MyActivity : AppCompatActivity() {
    // Initialize DI
    val taskRepository: TasksRepository
        get() = ServiceLocator.provideTasksRepository(this)
    val appComponent: AppComponent by lazy {
        DaggerAppComponent.factory().create(applicationContext)
    }
    override fun onCreate() {
        super.onCreate()
        // Initializing logging
        if (BuildConfig.DEBUG) {
            Timber.plant(DebugTree());
        } else {
            Timber.plant(CrashReportingTree());
        }
        // Inflating views
        setContentView(R.layout.main);
        // Registering broadcast receivers
        val filter = IntentFilter();
        intentFilter.addAction(packageName + "android.net.conn.CONNECTIVITY_CHANGE");
        registerReceiver(MyReceiver(), filter);
        // Kicking off initial tasks
        taskRepository.fetchInitialData();
        appComponent.scheduler.queueBackgroundTasks();
        // ... and so on ...
    }
}
```


Beware of ContentProviders

ContentProviders run before other code during startup.

Look through your app's merged `AndroidManifest.xml` for lines like

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

```
// http://tinyurl.com/5n78k6cp
public class InitializationProvider extends ContentProvider {
    @Override
    public final boolean onCreate() {
        Context context = getContext();
        if (context != null) {
            // Many Initializer's expect the `applicationContext` to be non-null. This
            // typically happens when `android:sharedUserId` is used. In such cases, we postpone
            // initialization altogether, and rely on lazy init.
            // More context: b/196959015
            Context applicationContext = context.getApplicationContext();
            if (applicationContext != null) {
                // Pass the class context so the right metadata can be read.
                // This is especially important in the context of apps that want to use
                // InitializationProvider in multiple processes.
                // b/183136596#comment18
                AppInitializer.getInstance(context).discoverAndInitialize(getClass());
            } else {
                StartupLogger.w("Deferring initialization because `applicationContext` is null.");
            }
        } else {
            throw new StartupException("Context cannot be null");
        }
        return true;
    }

    // ...
}
```


Lesson #5: Define what "*fully loaded*" means

Lesson #5: Define what "*fully loaded*" means

Aggressively defer everything else

Playback page

Fully loaded = first video frame rendered



Playback page

Fully loaded = first video frame rendered

1. Instantiate NewsApplication



Playback page

Fully loaded = first video frame rendered

1. Instantiate NewsApplication
2. Parse intent, determine experience to start



Playback page

Fully loaded = first video frame rendered



1. Instantiate `NewsApplication`
2. Parse intent, determine experience to start
3. Start `FireChannelsActivity`

Playback page

Fully loaded = first video frame rendered



1. Instantiate `NewsApplication`
2. Parse intent, determine experience to start
3. Start `FireChannelsActivity`
4. Instantiate minimal dependency graph

Playback page

Fully loaded = first video frame rendered



1. Instantiate `NewsApplication`
2. Parse intent, determine experience to start
3. Start `FireChannelsActivity`
4. Instantiate minimal dependency graph
5. Fetch initial data

Playback page

Fully loaded = first video frame rendered



1. Instantiate `NewsApplication`
2. Parse intent, determine experience to start
3. Start `FireChannelsActivity`
4. Instantiate minimal dependency graph
5. Fetch initial data
6. Determine the page to load

Playback page

Fully loaded = first video frame rendered



1. Instantiate `NewsApplication`
2. Parse intent, determine experience to start
3. Start `FireChannelsActivity`
4. Instantiate minimal dependency graph
5. Fetch initial data
6. Determine the page to load
7. Inflate view and place the fragment

Playback page

Fully loaded = first video frame rendered



1. Instantiate `NewsApplication`
2. Parse intent, determine experience to start
3. Start `FireChannelsActivity`
4. Instantiate minimal dependency graph
5. Fetch initial data
6. Determine the page to load
7. Inflate view and place the fragment
8. Load video player

Playback page

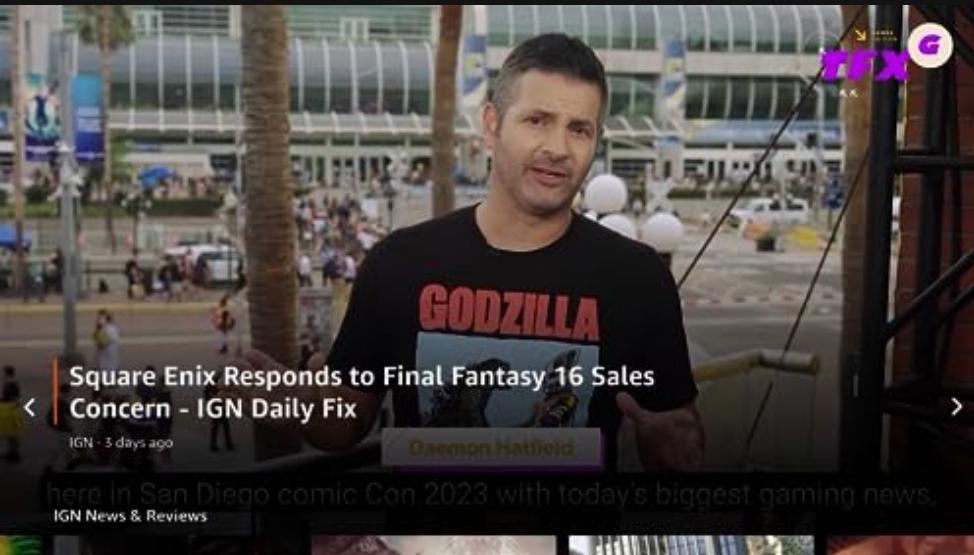
Fully loaded = first video frame rendered



1. Instantiate `NewsApplication`
2. Parse intent, determine experience to start
3. Start `FireChannelsActivity`
4. Instantiate minimal dependency graph
5. Fetch initial data
6. Determine the page to load
7. Inflate view and place the fragment
8. Load video player
9. `Play video`

Playback page

Fully loaded = first video frame rendered

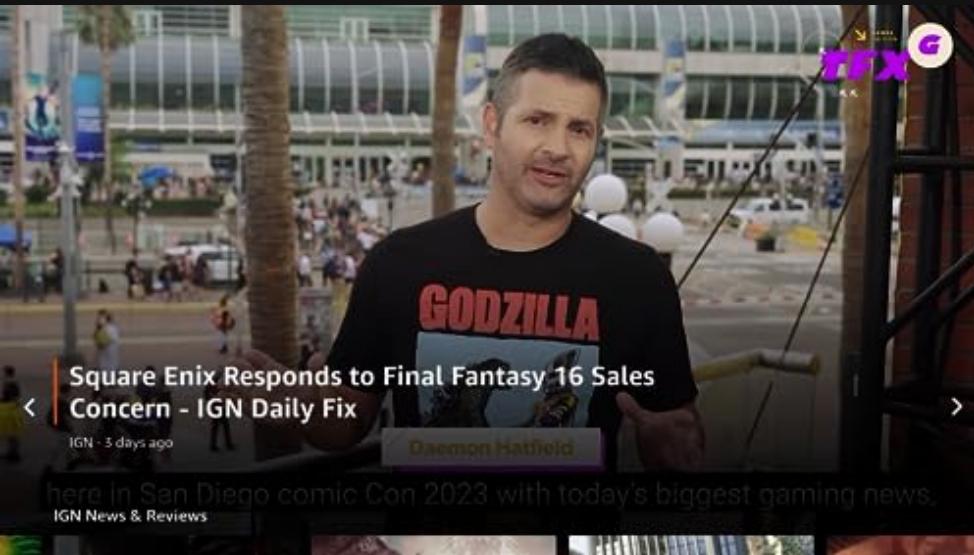


1. Instantiate `NewsApplication`
2. Parse intent, determine experience to start
3. Start `FireChannelsActivity`
4. Instantiate minimal dependency graph
5. Fetch initial data
6. Determine the page to load
7. Inflate view and place the fragment
8. Load video player
9. Play video // We need to get here

What were we actually doing?



What were we actually doing?



1. Instantiate NewsApplication

What were we actually doing?



1. Instantiate NewsApplication
2. Instantiate dependency graph *for everything*

What were we actually doing?



1. Instantiate NewsApplication
2. Instantiate dependency graph *for everything*
3. Start background processes *for everything*

What were we actually doing?



1. Instantiate `NewsApplication`
2. Instantiate dependency graph *for everything*
3. Start background processes *for everything*
4. Parse intent, determine experience to start

What were we actually doing?



1. Instantiate `NewsApplication`
2. Instantiate dependency graph *for everything*
3. Start background processes *for everything*
4. Parse intent, determine experience to start
5. Start `FireChannelsActivity`

What were we actually doing?



1. Instantiate `NewsApplication`
2. Instantiate dependency graph *for everything*
3. Start background processes *for everything*
4. Parse intent, determine experience to start
5. Start `FireChannelsActivity`
6. Instantiate dependency graph *for everything*

What were we actually doing?



1. Instantiate `NewsApplication`
2. Instantiate dependency graph *for everything*
3. Start background processes *for everything*
4. Parse intent, determine experience to start
5. Start `FireChannelsActivity`
6. Instantiate dependency graph *for everything*
7. Fetch initial data

What were we actually doing?



1. Instantiate `NewsApplication`
2. Instantiate dependency graph *for everything*
3. Start background processes *for everything*
4. Parse intent, determine experience to start
5. Start `FireChannelsActivity`
6. Instantiate dependency graph *for everything*
7. Fetch initial data
8. Do cache maintenance

What were we actually doing?



1. Instantiate `NewsApplication`
2. Instantiate dependency graph *for everything*
3. Start background processes *for everything*
4. Parse intent, determine experience to start
5. Start `FireChannelsActivity`
6. Instantiate dependency graph *for everything*
7. Fetch initial data
8. Do cache maintenance
9. Determine the page to load

What were we actually doing?



1. Instantiate `NewsApplication`
2. Instantiate dependency graph *for everything*
3. Start background processes *for everything*
4. Parse intent, determine experience to start
5. Start `FireChannelsActivity`
6. Instantiate dependency graph *for everything*
7. Fetch initial data
8. Do cache maintenance
9. Determine the page to load
10. Place the fragment

What were we actually doing?



1. Instantiate `NewsApplication`
2. Instantiate dependency graph *for everything*
3. Start background processes *for everything*
4. Parse intent, determine experience to start
5. Start `FireChannelsActivity`
6. Instantiate dependency graph *for everything*
7. Fetch initial data
8. Do cache maintenance
9. Determine the page to load
10. Place the fragment
11. Load video player

What were we actually doing?



1. Instantiate `NewsApplication`
2. Instantiate dependency graph *for everything*
3. Start background processes *for everything*
4. Parse intent, determine experience to start
5. Start `FireChannelsActivity`
6. Instantiate dependency graph *for everything*
7. Fetch initial data
8. Do cache maintenance
9. Determine the page to load
10. Place the fragment
11. Load video player
12. Play video // We need to get here

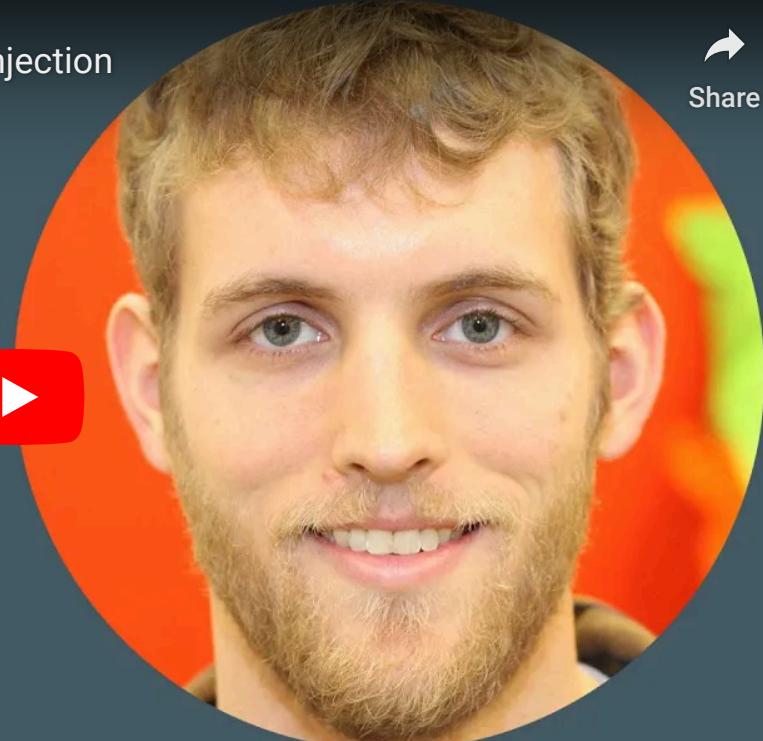
Lesson #6: Dependency injection



DAGGER 2 - A New Type of dependency injection



Dagger 2: A New Type of Dependency Injection



Watch on  YouTube

"The downside of tools that make things easy

"The downside of tools that make things easy is that they make them easy."

"The downside of tools that make things easy is that they make them easy."

- Me, while debugging this project

"The downside of tools that make things easy is that they make them easy."

- Me, while debugging this project 😬

What's the problem?

```
@Singleton
public class SharedPreferencesManager {
    @Inject
    public SharedPreferencesManager(Context context) {
        SharedPreferences sharedPref = context.getSharedPreferences(
            getString(R.string.preference_file_key), Context.MODE_PRIVATE);

        String bigJsonFile = sharedPref.getString("shared_preferences")
        Preferences preferences = new Gson().fromJson(bigJsonFile, Preferences.class);

        // ...
    }
}
```

What's the problem?

```
@Singleton
public class SharedPreferencesManager {
    @Inject
    public SharedPreferencesManager(Context context) {
        SharedPreferences sharedPref = context.getSharedPreferences(
            getString(R.string.preference_file_key), Context.MODE_PRIVATE);

        String bigJsonFile = sharedPref.getString("shared_preferences")
        Preferences preferences = new Gson().fromJson(bigJsonFile, Preferences.class);

        // ...
    }
}
```

Another example (a hunch)

```
@Singleton
public class AuthManager {
    private Executors executors;
    private JobScheduler jobScheduler;

    @Inject
    public AuthManager(Executors executors, JobScheduler jobScheduler) {
        this.executors = executors;
        this.jobScheduler = jobScheduler;
        kickOffBackgroundWork();
    }

    private void kickOffBackgroundWork() {
        this.executors.diskIO().submit(() -> {
            // ...
            this.executors.networkIO().submit(() -> {
                // ...
                jobScheduler.startJobs();
            })
            // ...
            jobScheduler.startOtherJobs();
        })
    }
}
```

Another example (a hunch)

```
@Singleton
public class AuthManager {
    private Executors executors;
    private JobScheduler jobScheduler;

    @Inject
    public AuthManager(Executors executors, JobScheduler jobScheduler) {
        this.executors = executors;
        this.jobScheduler = jobScheduler;
        kickOffBackgroundWork();
    }

    private void kickOffBackgroundWork() {
        this.executors.diskIO().submit(() -> {
            // ...
            this.executors.networkIO().submit(() -> {
                // ...
                jobScheduler.startJobs();
            })
            // ...
            jobScheduler.startOtherJobs();
        })
    }
}
```



droidcon NYC

droidcon NYC 2017 - Optimizing Dagger on Android: Developer Workflow and Ru...

September 25-26 2017



Share



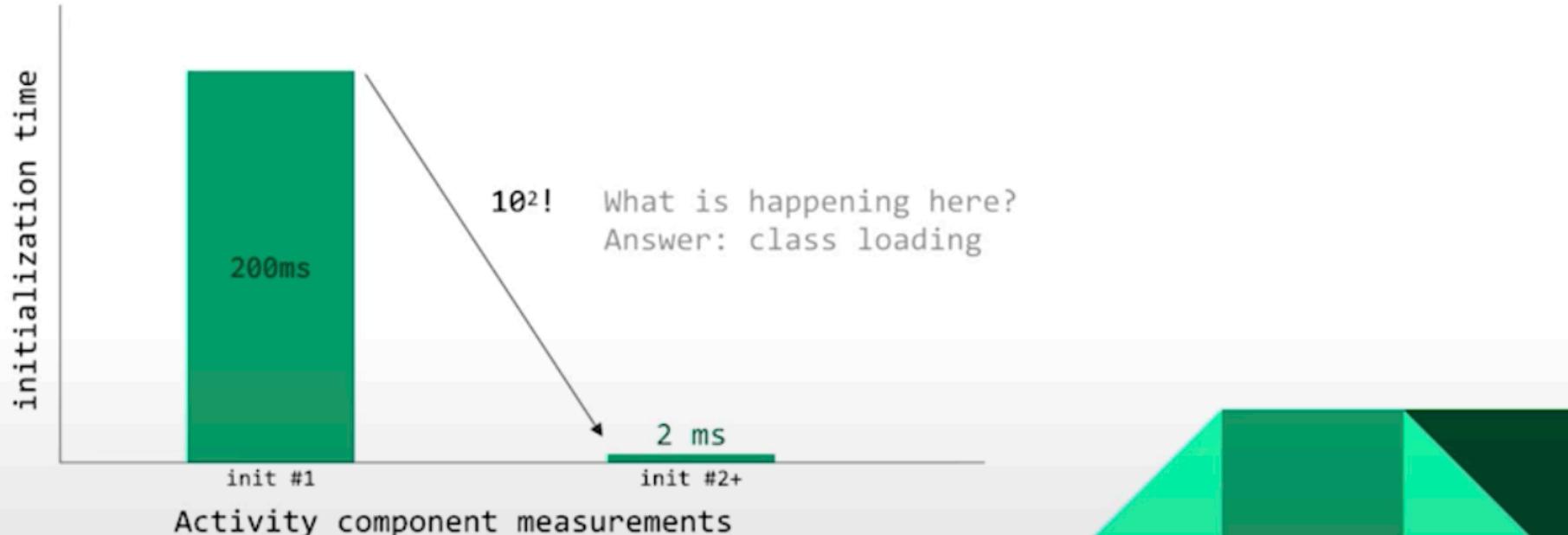
Why does this
matter?



Watch on YouTube

touchlab

But just *how slow* is it?



50%!

Lesson #7: Continuous testing

Macrobenchmark

```
@LargeTest
@RunWith(AndroidJUnit4::class)
class SampleStartupBenchmark {
    @get:Rule
    val benchmarkRule = MacrobenchmarkRule()

    @Test
    fun startup() = benchmarkRule.measureRepeated(
        packageName = TARGET_PACKAGE,
        metrics = listOf(StartupTimingMetric()),
        iterations = DEFAULT_ITERATIONS,
        setupBlock = {
            // Press home button before each run to ensure the starting activity isn't visible.
            pressHome()
        }
    ) {
        // starts default launch activity
        startActivityAndWait()
    }
}
```

Run: SampleStartupBenchmark ⚙️

Status 1 passed | 1 tests, 13 s 935 ms

Filter tests:

Tests

✓ Test Results

SampleStartupBenchmark

startup

✓ Test Results

Test Results

SampleStartupBenchmark

startup

01/25 16:46:25: Launching 'SampleStartupBenchmark' on Pixel 3a API 30.

Install successfully finished in 5 s 242 ms.

Running tests

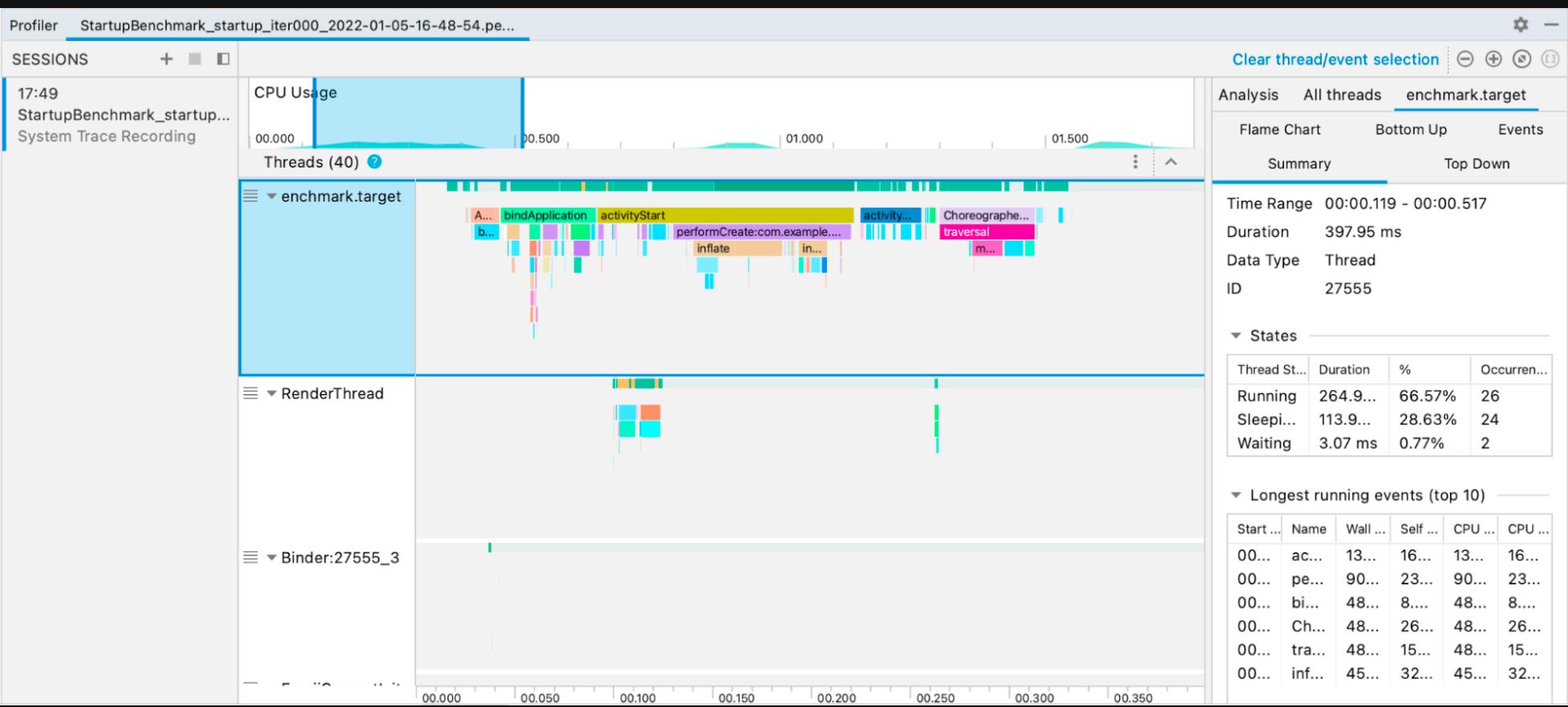
|

\$ adb shell am instrument -w -m -e androidx.benchmark.suppressErrors EM
SampleStartupBenchmark_startup

timeToFullDisplayMs [min 748.1](#), [median 748.1](#), [max 748.1](#)

timeToInitialDisplayMs [min 748.1](#), [median 748.1](#), [max 748.1](#)

Traces: Iteration [0](#) [1](#) [2](#) [3](#) [4](#)



Microbenchmark

```
@RunWith(AndroidJUnit4::class)
class SampleBenchmark {
    @get:Rule
    val benchmarkRule = BenchmarkRule()

    @Test
    fun benchmarkSomeWork() {
        benchmarkRule.measureRepeated {
            doSomeWork()
        }
    }
}
```

Lesson #8: Move critical steps earlier

Yup, that was the fix

```
class MyActivity : AppCompatActivity() {
    override fun onCreate() {
        super.onCreate()
        setContentView(R.layout.main);
        backend.fetchInitialData();
    }
}
~~~
class MyActivity : AppCompatActivity() {
    override fun onCreate() {
        super.onCreate()
        backend.fetchInitialData();
        setContentView(R.layout.main);
    }
}
```

Lesson #9: Improve cache hits

Lesson #10: Only improve what you need to

Lesson #11: Observability

More on my blog...

You Probably Don't Need Mocking

Mocking is overused and can lead to brittle tests. Here I go over alternatives and when you should reach for mocking.



Justifying Mocking

Testing techniques like mocking are often overused, but can be justified. In this post I break down what justifies such techniques.



Fire TV Channels - Free

by Amazon.com

4.5 out of 5 stars  413 customer ratings Guidance Suggested



Price: **Free Download**

Sold by: Amazon.com Services, Inc

Works with: Game Controllers, Fire TV Voice
Remote

Languages Supported: English

Get App

Learn how buying works

By placing your order, you agree to our [Terms of Use](#)





Bonus lesson #1: Enabling compression

Accept-Encoding: gzip, deflate

Bonus lesson #2: Plan for model evolution



REFLECTION





Bonus lesson #3: Remove network calls entirely

Performance isn't scary

Performance isn't scary



You don't need the fanciest tools

Performance isn't scary

 You don't need the fanciest tools

 You don't need it perfect to ship

Performance isn't scary

- 🔨 You don't need the fanciest tools

- 🚢 You don't need it perfect to ship

- 🔄 Iterate

Performance isn't scary

- 🔨 You don't need the fanciest tools

- 🚢 You don't need it perfect to ship

- 🔄 Iterate

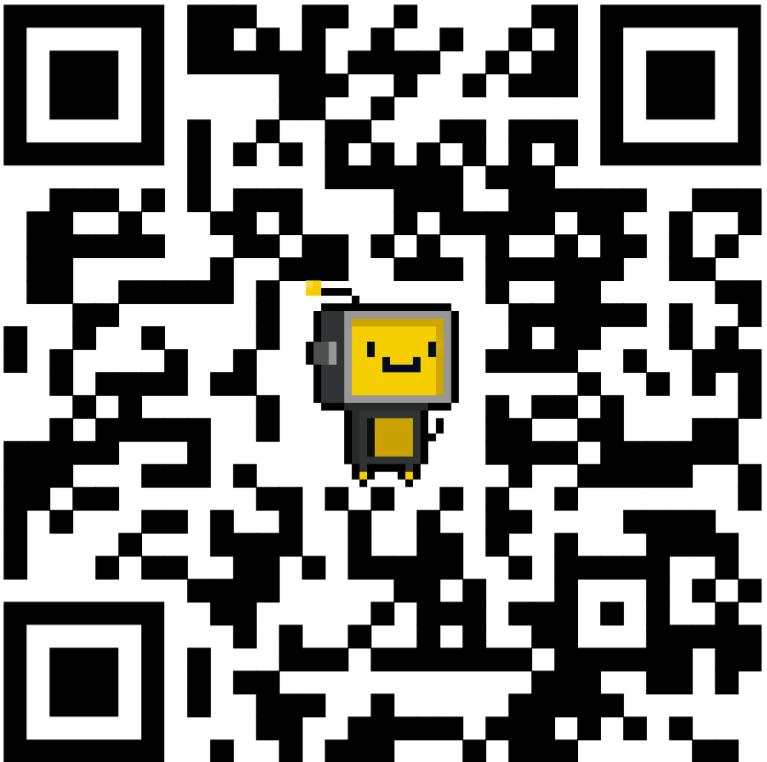
- 🧪 Experiment

Performance isn't scary

- 🔨 You don't need the fanciest tools
- 🚢 You don't need it perfect to ship
- 🔄 Iterate
- 🧪 Experiment
- ⌚ Continuously test

Performance isn't scary

-  You don't need the fanciest tools
-  You don't need it perfect to ship
-  Iterate
-  Experiment
-  Continuously test
-  Knowledge is transferrable



SCAN ME

Byte Bot helps
software teams ship
fullstack solutions 

Let's chat!

Also...

📅 calendly.com/byte-bot

✉️ info@bytebot.io

🐥 [@RyanClementsHax](#)

⬇️ bytebot.beehiiv.com/subscribe

🔗 linkedin.com/in/ryan-clements-hax