

# Kotlin, one language to build them all!

Rodrigo Oliveira & Paul Merlin



[gradle.github.io/imagine](https://gradle.github.io/imagine)

[video](#) - [pdf](#) - [sources](#)

 Gradle

# Agenda

**Imaginate**

An imaginary image generator

**Gradle & Kotlin ❤**

Build logic

**Speed-up Loop Bonus**

Configuration cache

# Who are we?





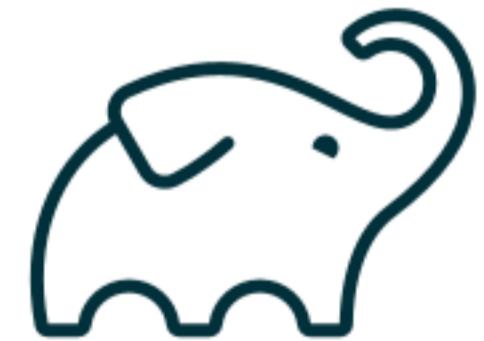
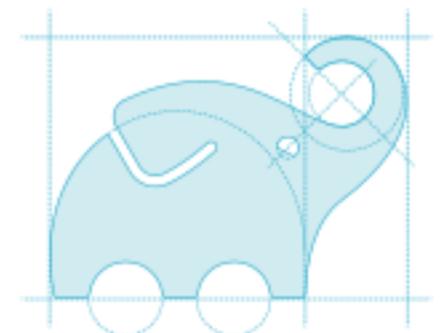
```
speaker {  
    name = "Paul Merlin"  
    company = "Gradle"  
    joined = 2015  
    position = "Kotlin DSL Project Lead, again \o/"  
    previously = "Configuration Cache Project Lead"  
    github = "eskatos"  
    mastodon = "@eskatos@mastodon.social"  
}
```

```
speaker {  
    name = "Rodrigo B. de Oliveira"  
    company = "Gradle"  
    joined = 2015  
    position = "Configuration Team Lead"  
    previously = "Kotlin DSL Project Lead"  
    github = "bamboo"  
    mastodon = "@rbo@mastodon.social"  
}
```

# Gradle



Since 2008, our mission is to accelerate developer productivity.





# Gradle Build Tool

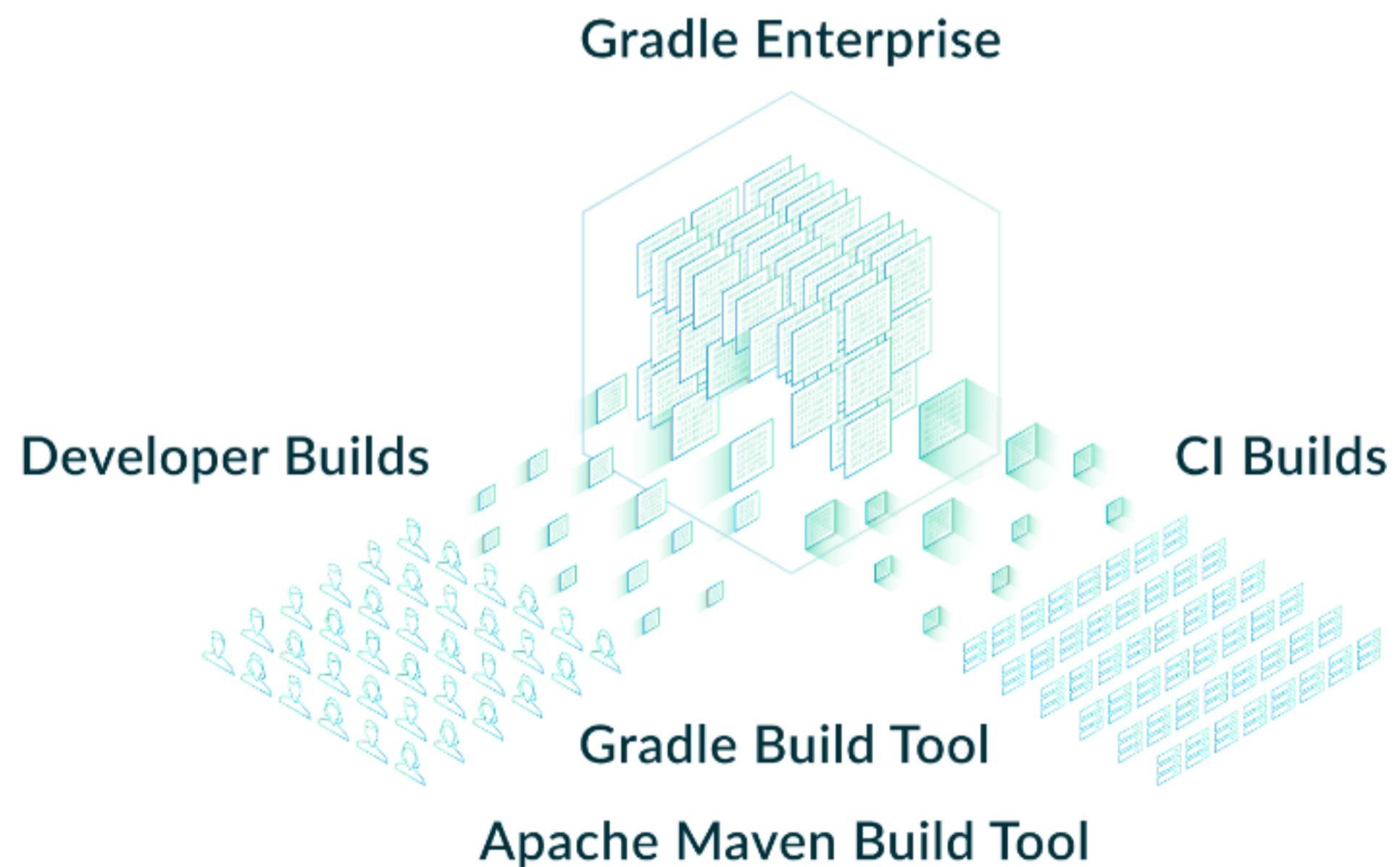
Apache licenced software build tool

With 30M+ monthly downloads, this is one of the top 20 popular open source projects according to [TechCrunch](#).

# Gradle Enterprise



Gradle Enterprise, commercial product, is the first Developer Productivity Engineering (DPE) integrated solution.



# Developer Productivity Engineering

DPE is an emerging software practice that relies on acceleration technologies and data analysis to improve developer productivity.

- make builds and tests faster
- make problem research more efficient



[twitter.com/DpeShowdown](https://twitter.com/DpeShowdown)

# Build Scans

A permanent record  
of what happens during a build.



The screenshot shows the Gradle Enterprise Build Scan interface for a build named "example-build" from October 19, 2021. The left sidebar includes links for Summary, Console log, Timeline (which is selected), Performance, Tests, Projects, Dependencies, Build dependencies, Plugins, Switches, Infrastructure, and Delete Build Scan. The main area displays a timeline with two phases: Initialization & configuration and Execution. In the Execution phase, tasks like :app:test and :disttest are shown. A detailed view of the :app:test task is open, showing its duration (1.037s), type (org.gradle.api.DefaultTask), and path (:app:test). It also notes that this task is on the critical path. The task was started after 1.038s and took 0.754s.

build scan

Gradle & Maven build speed challenge



Get some swag :)





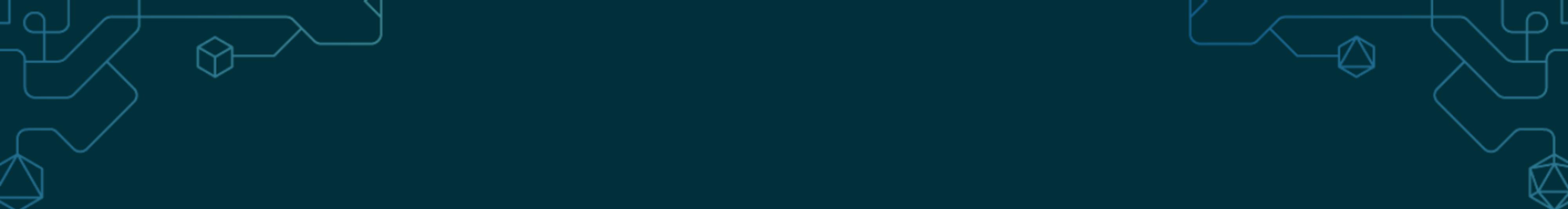
# We recruit!

If what we're going to talk about Today is of interest to you, come work with us!



[gradle.com/careers](http://gradle.com/careers)



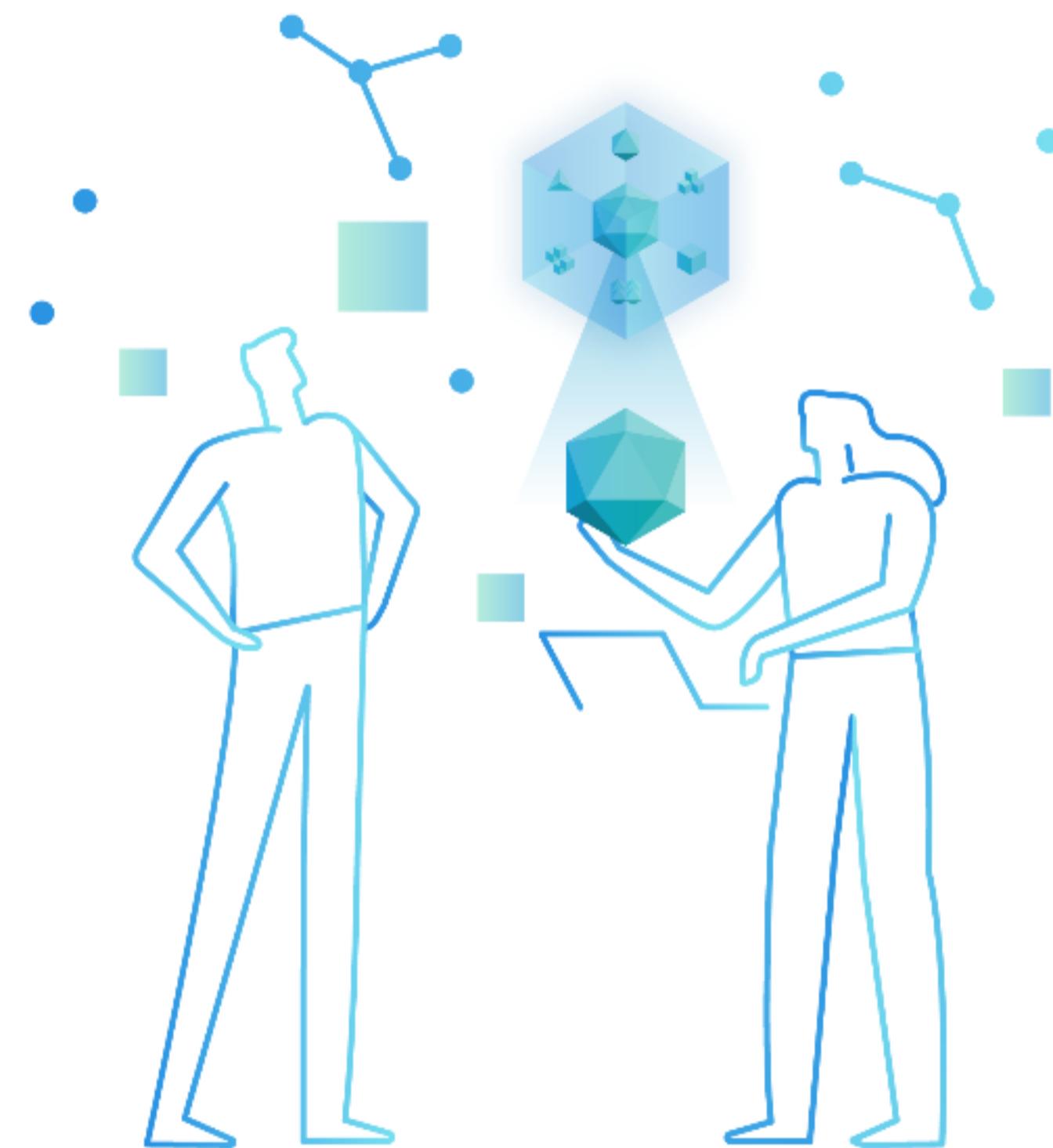


# Imagine

An imaginary image generator

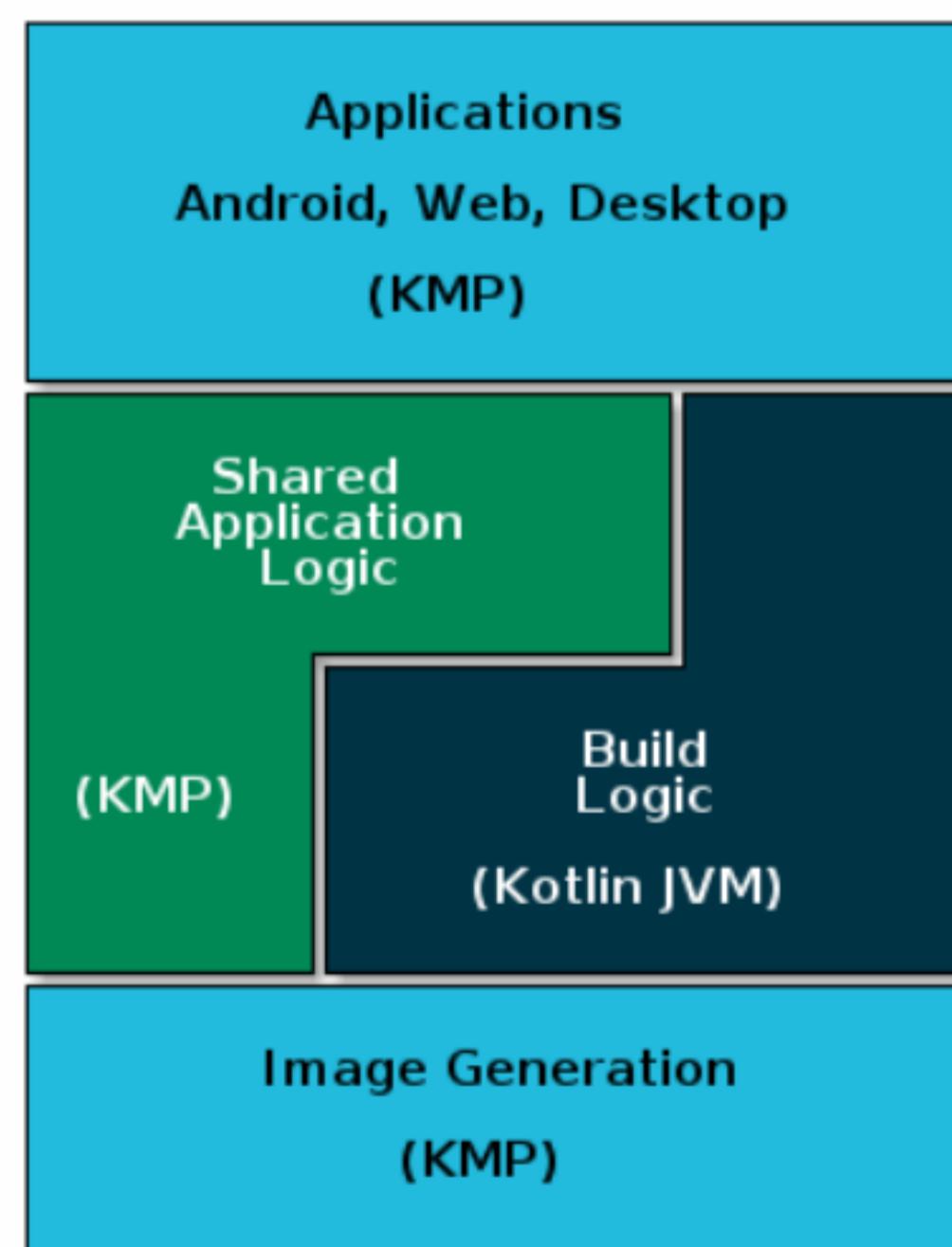
# Imagine - Demo

# Imaginate - Exploration



[gradle/imagine#DesktopMain.kt](https://gradle/imagine#DesktopMain.kt)

# Imaginate - Overview



# Gradle & Kotlin ❤

Build logic

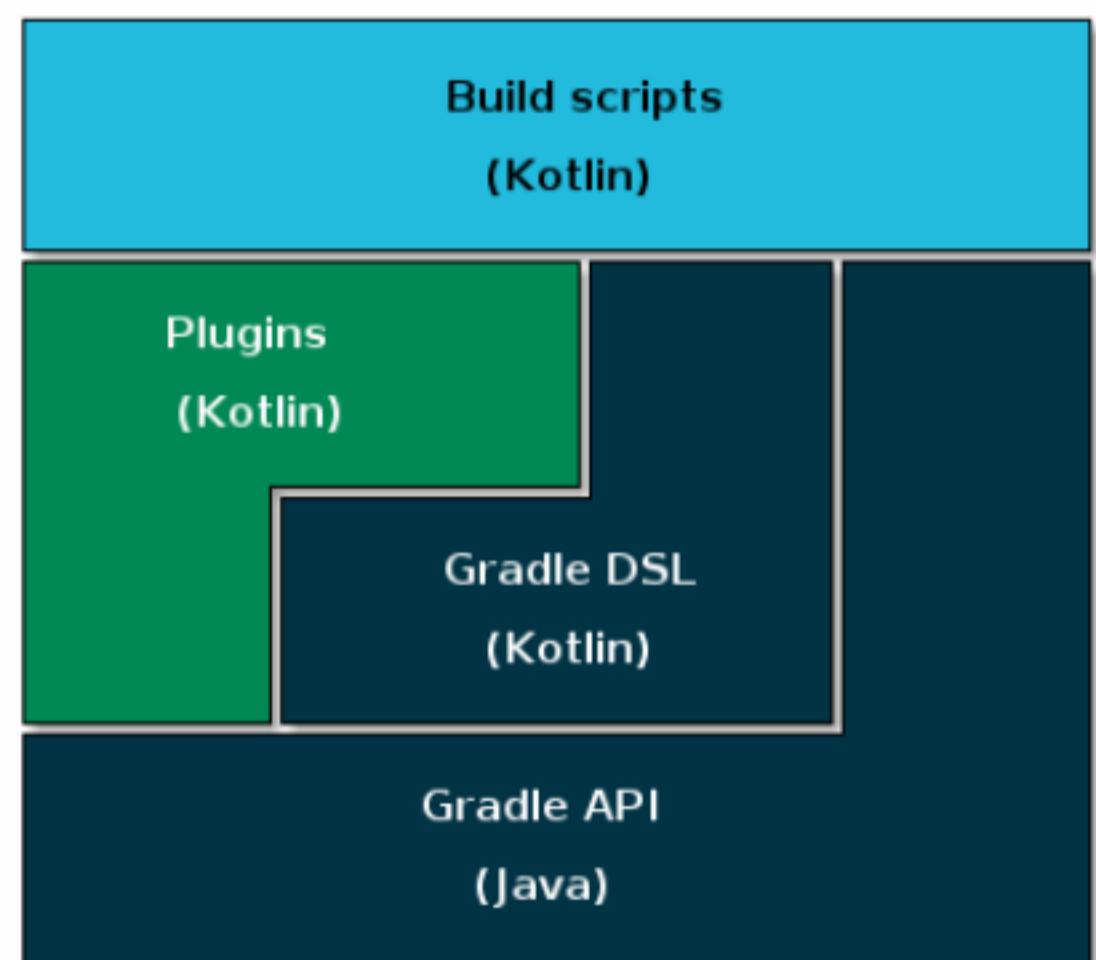


# Build logic - Kotlin!



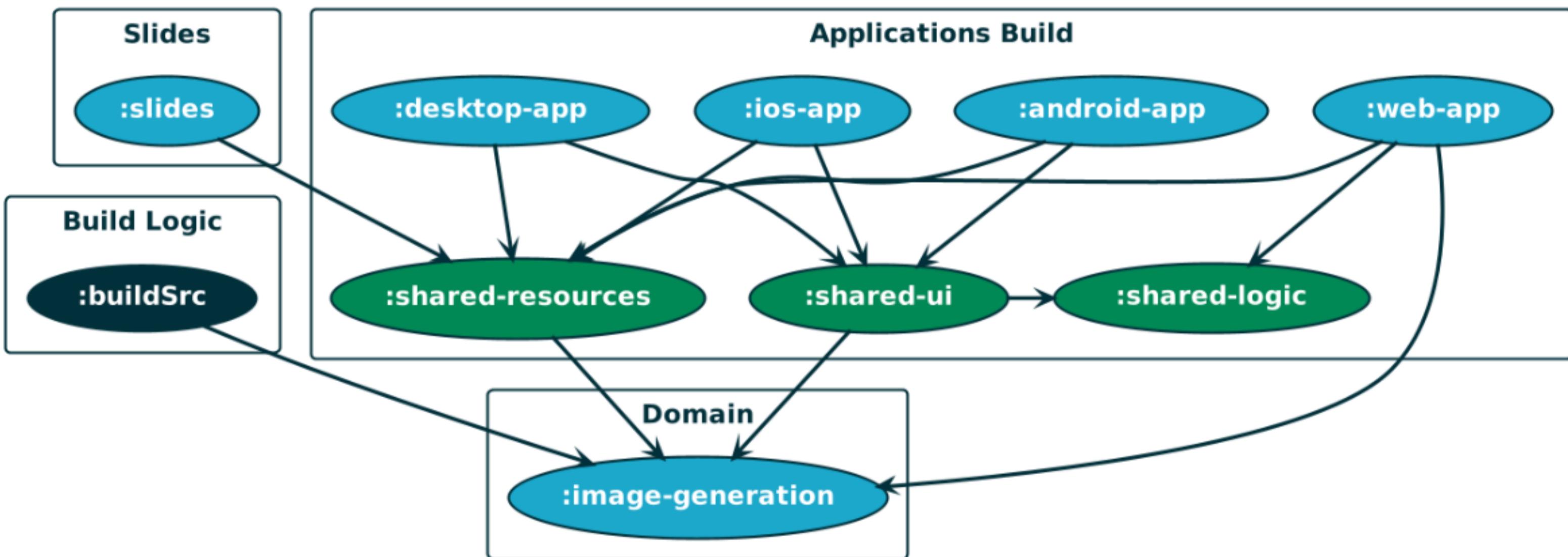
## Gradle's Kotlin DSL

- Kotlin DSL on top of Gradle's Java API
- Low ceremony thanks to Kotlin Scripting
- Official Kotlin compiler plugins
- IDE assistance, code navigation etc...



# Build logic - Structure

A composite build bringing 4 included builds together.



# Build logic - Slides

"capital letter K. elephant. pop art."



"elephant in the Kotlin island.  
caravaggio."



# Build logic - Overview

settings.gradle.kts

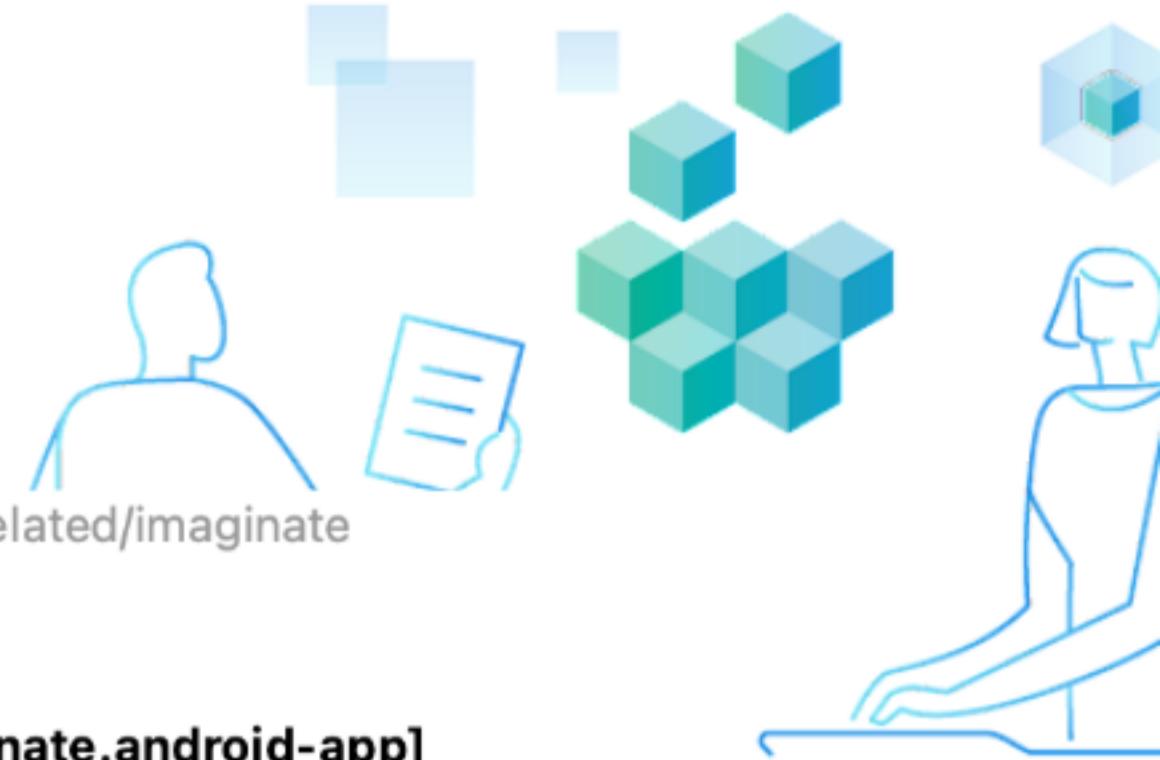
```
includeBuild("image-generation")
includeBuild("slides")

include("shared-resources")
include("shared-logic")
include("shared-ui")
include("desktop-app")
include("android-app")
include("ios-app")
include("web-app")

for (project in rootProject.children) {
    project.projectDir = file("applications/${project.name}")
}
```

buildSrc/settings.gradle.kts

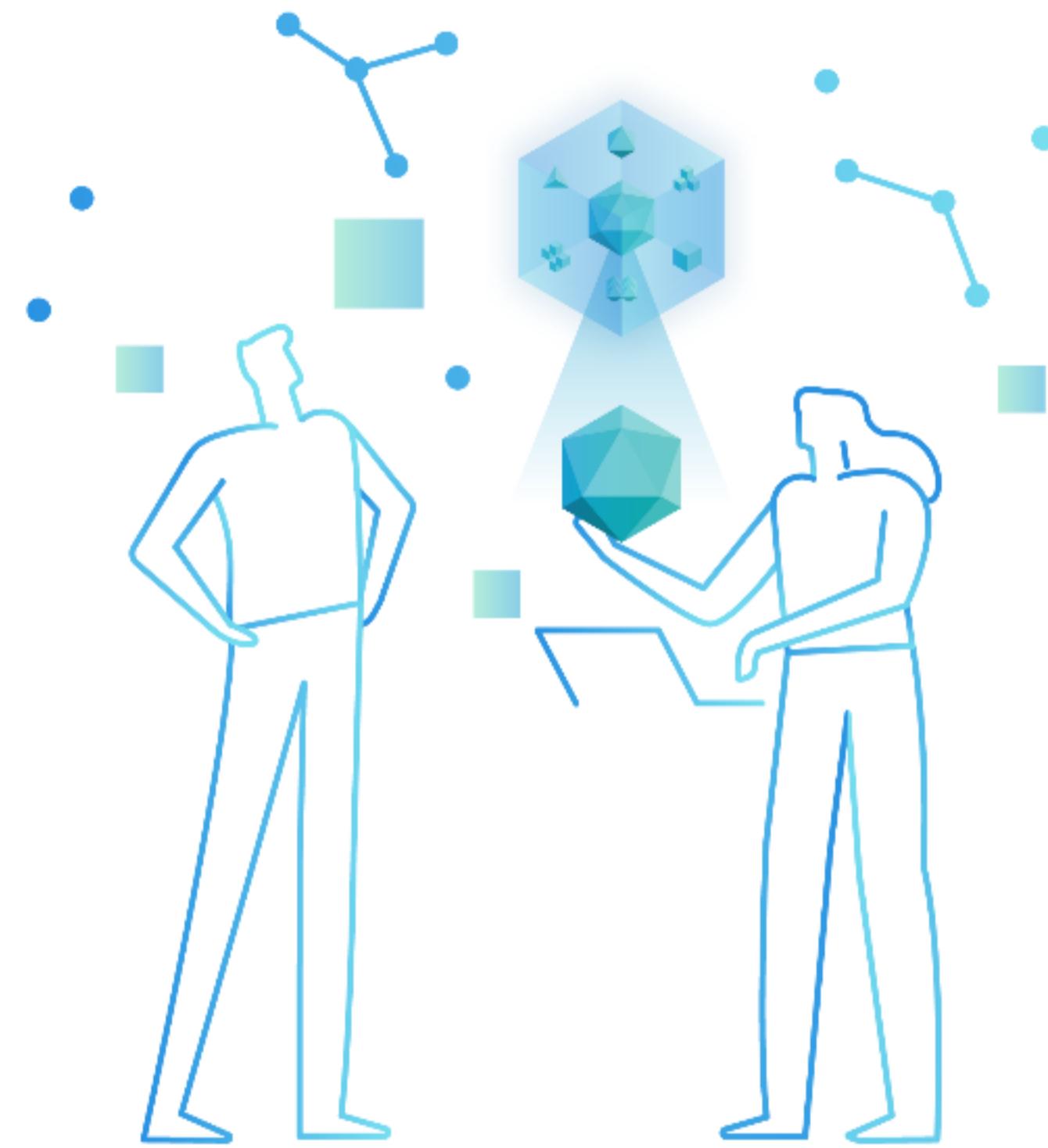
```
includeBuild("../image-generation")
```



```
✓  imaginate ~/src/gradle-related/imaginat
  >  .github
  >  applications
    >  android-app [imaginat.android-app]
    >  desktop-app [imaginat.desktop-app]
    >  ios-app [imaginat.ios-app]
    >  shared-logic [imaginat.shared-logic]
    >  shared-resources [imaginat.shared-resources]
    >  shared-ui [imaginat.shared-ui]
    >  web-app [imaginat.web-app]
  >  buildSrc
  >  gradle
  >  image-generation
  >  slides
  >  .gitignore
  >  build.gradle.kts
  >  gradle.properties
  >  gradlew
  >  gradlew.bat
  >  LICENSE
  >  local.properties
  >  README.md
  >  settings.gradle.kts
```



# Build logic - Exploration



[gradle/imaginete#ImageGenerator.kt](#)



# Build logic - Kotlin DSL

Gradle plugins contribute to a dynamic model

Kotlin DSL provides static accessors

- extensions and tasks
- projects and version catalogs

IDE assistance, code navigation etc...

applications/web-app/build.gradle.kts

```
plugins {
    id("web-app")
}

dependencies {
    // Configuration defined in the
    // `shared-resources-consumer` convention plugin
    sharedBitmaps(projects.sharedResources)
}

kotlin {
    sourceSets {
        // Source set defined in `web-app` convention plugin
        jsMain {
            dependencies {
                // Static project accessor
                implementation(projects.sharedLogic)
                // Static version catalog accessors
                implementation(libs.imaginate.imageGeneration)
            }
        }
    }
}
```



# Build logic - Building blocks



## Convention plugins

- Put `.gradle.kts` files in `buildSrc/src/main/kotlin`
- Apply them in project build scripts

## Helpers

- Put `.kt` files in `buildSrc/src/main/kotlin`
- Prefer internal visibility

## Custom DSL

- Use Gradle extensions and containers

## Custom Tasks

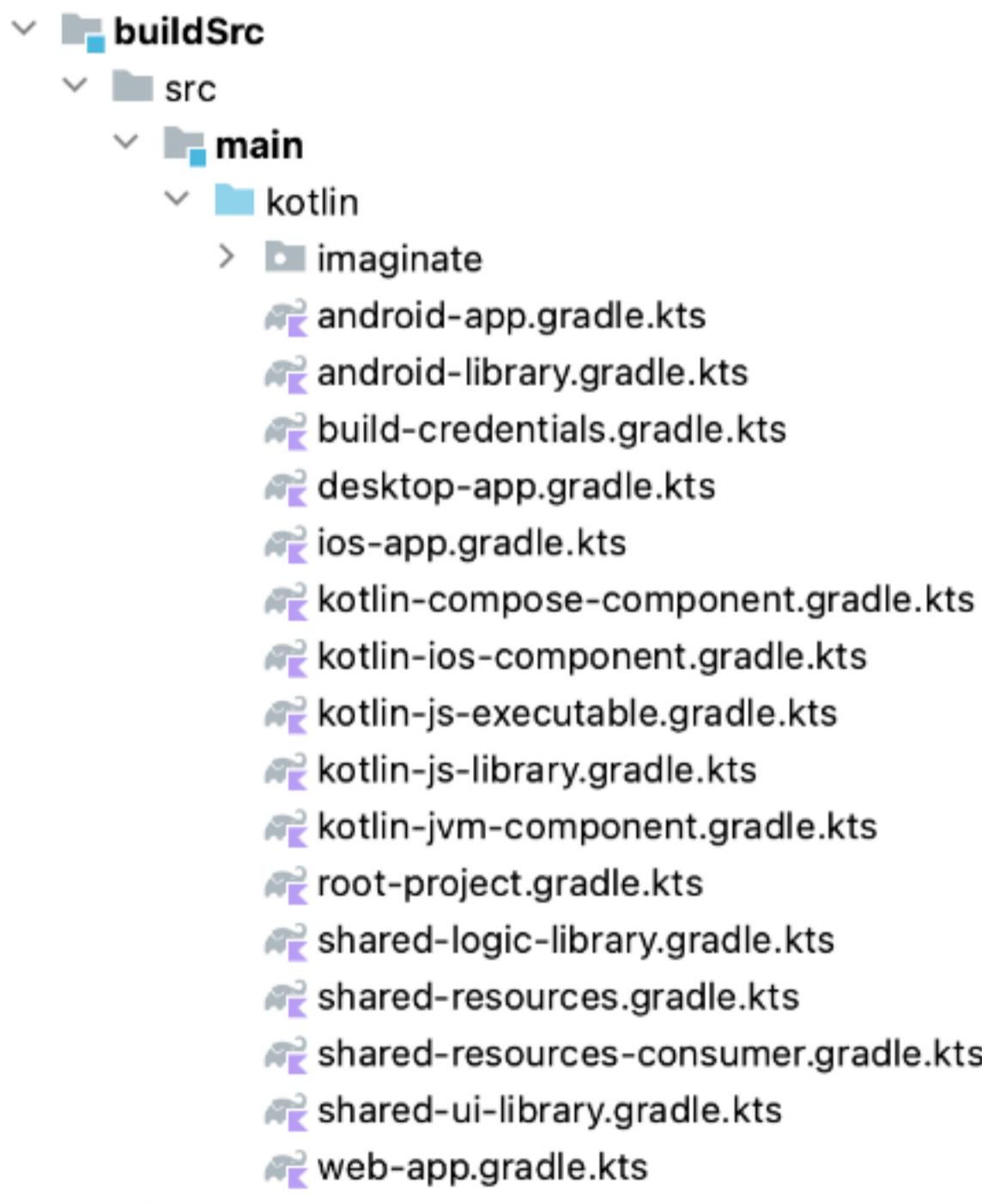
- Use the Worker API for isolation and parallelism



# Build logic - buildSrc

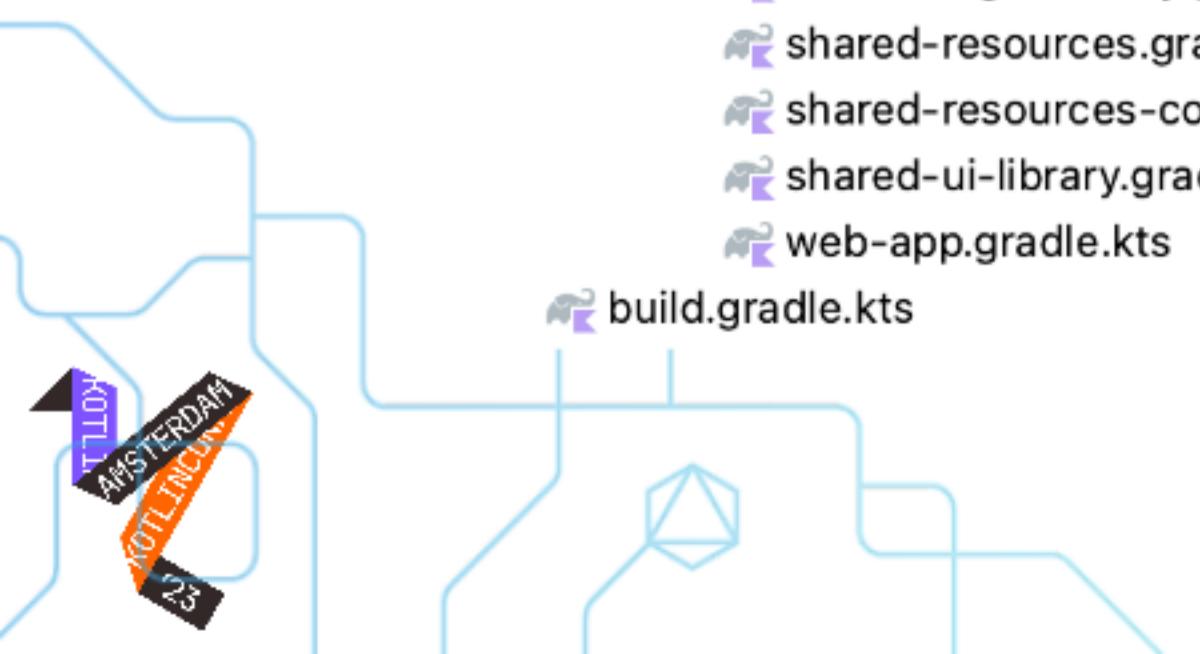
Keep imperative logic out of project build scripts.

Expose declarative DSL from convention plugins instead.



applications/android-app/build.gradle.kts

```
plugins {  
    id("android-app")  
}  
  
android {  
    namespace = "imagine.android"  
}  
  
dependencies {  
    implementation(projects.sharedLogic)  
}
```



# Build logic - buildSrc?

buildSrc vs includeBuild for build logic

## buildSrc

Everything is made available in the owner build's project scripts.

You can directly use anything.

## includeBuild

Nothing is made available by default.

You have to declare dependencies.

Convention plugins

```
plugins {  
    id("my-convention-plugin")  
}
```

From buildSrc

```
dependencies {  
    implementation("my:included-build:latest")  
}
```

Since Gradle 8.0, buildSrc is much more like included builds  
[docs.gradle.org/8.0/release-notes.html#improvements-for-buildsrc-builds](https://docs.gradle.org/8.0/release-notes.html#improvements-for-buildsrc-builds)



# Build logic - buildSrc first

Start with `buildSrc`,  
move to included builds when you need to

- share logic between builds,
- isolate a portion of build logic.

Transition is simple  
if you only use convention plugins

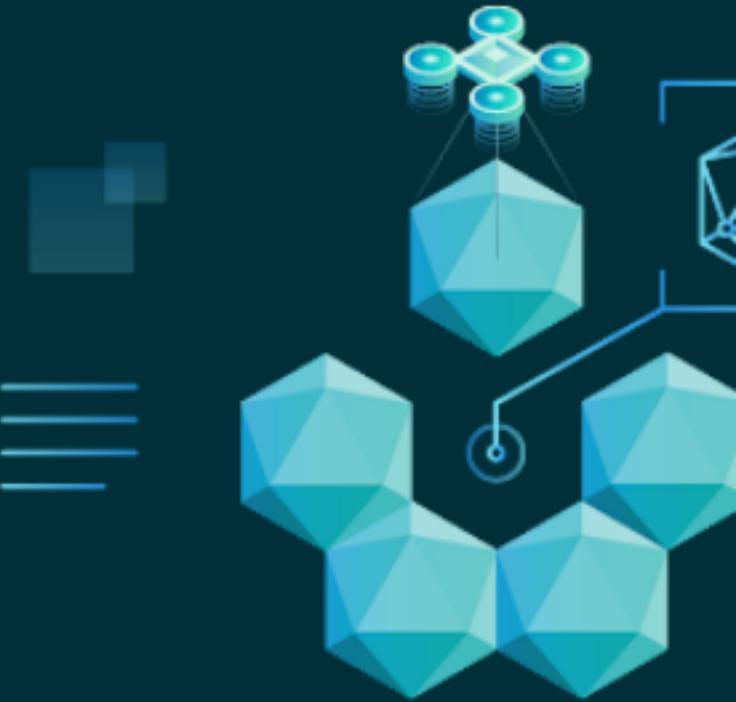
Publish your convention plugins for sharing them across repositories



# Build logic - Learn more

- Gradle's Kotlin DSL
  - Documentation
  - DSL Reference
- Build organization
  - Structuring individual builds
  - Structuring software products
  - Jendrik's [Understanding Gradle](#) videos
  - Tony's and Cédric's blog posts





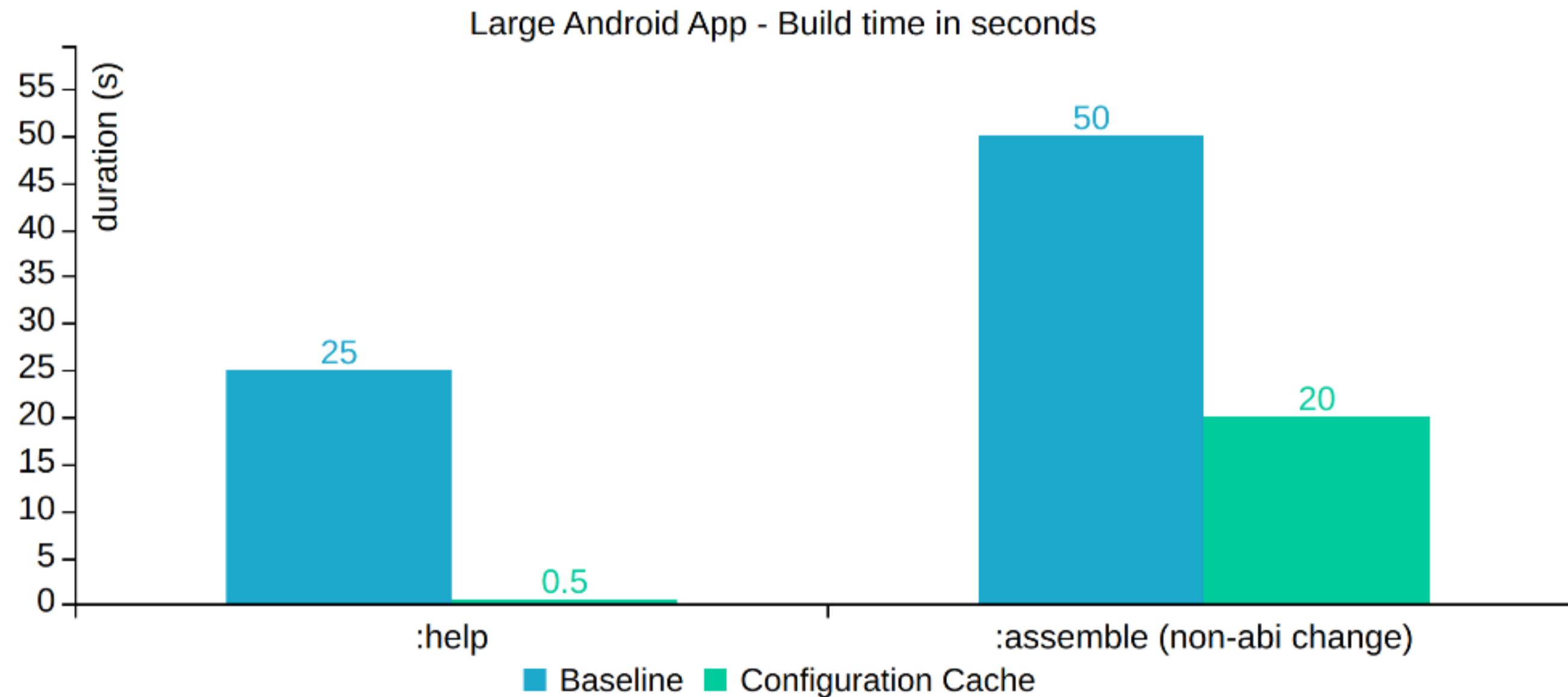
# Speed-up Loop Bonus

Configuration cache



# Configuration cache

Significantly improves build performance  
by caching the result of the configuration phase  
and reusing this for subsequent builds.



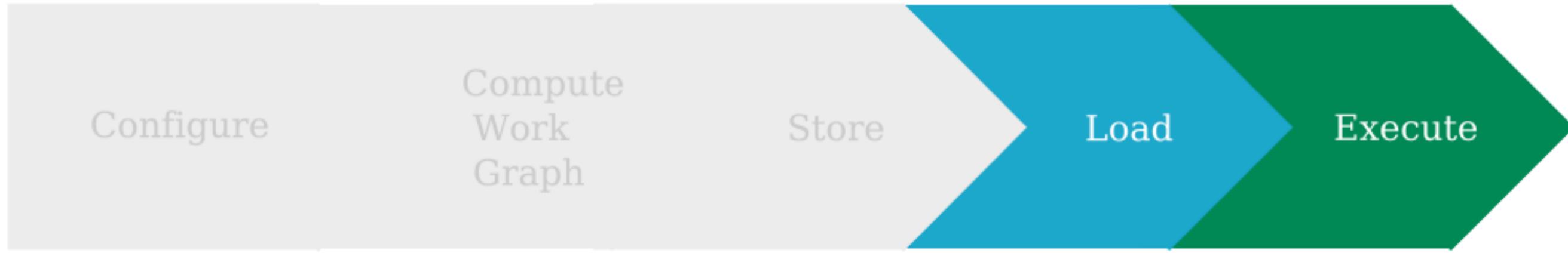


# Configuration cache - How does it work?

First run



Subsequent runs





# Configuration cache - What are the benefits?

- Configuration time becomes work graph loading time
- Increased parallelism
- Reduced memory consumption





# Configuration cache - Turn it on!

Stable in Gradle 8.1

[docs.gradle.org/8.1/release-notes.html](https://docs.gradle.org/8.1/release-notes.html)

On the command line

```
./gradlew --configuration-cache build
```

Persistently in gradle.properties

```
org.gradle.configuration-cache=true
```



# What's next?

## Usability

- Simpler container DSL
- Kotlin and Kotlin DSL version management
- DSL reference improvements
- And more...

## Performance

- Script compilation avoidance
- plugins {} block interpreter
- Faster precompiled script plugins build
- K2
- Quicker first-use
- Configuration cache by default for Android
- Project Isolation



# We recruit!

If what we talked about Today is of interest to you, come work with us!



[gradle.com/careers](http://gradle.com/careers)

# Thank you!

Don't forget to vote!

