Improving the Kotlin Developer Experience in Koin 3.2

Fosdem 2023

Arnaud Giuliani
@arnogiu

Koin Project Lead

Google Dev Expert - Kotlin

JetBrains Certified

@arnogiu

Arnaud GIULIANI

kotzilla.io

help structure your app,
with ease and efficiency

Koin in 2 minutes

```
class ClassA()
class ClassB(val a : ClassA)
```

```
class ClassA()
class ClassB(val a : ClassA)

module {


}
```

Configure your components with Koin DSL

```
class ClassA()
class ClassB(val a : ClassA)

module {
    single { ClassA() }

}
```

Configure your components with Koin DSL

```
class ClassA()
class ClassB(val a : ClassA)

module {
    single { ClassA() }
    single { ClassB() }
}
```

Configure your components with Koin DSL

```
class ClassA()
class ClassB(val a : ClassA)

module {
    single { ClassA() }
    single { ClassB(???) }
}
```

Configure your components with Koin DSL

```
class ClassA()
class ClassB(val a : ClassA)

module {
    single { ClassA() }
    single { ClassB(get()) }
}
```

Configure your components with Koin DSL

```
class ClassA()
class ClassB(val a : ClassA)

module {
    single { ClassA() }
    single { ClassB(get()) }
}
```

Koin creates instances:

- Constructors
- Kotlin functions

```
class MyApp {

    val classB : ClassB

}
```

```
class MyApp : KoinComponent {

    val classB : ClassB

}
```

```
class MyApp : KoinComponent {

    val classB by inject<ClassB>()

}
```

```
class MyApp : KoinComponent {

    val classB by inject<ClassB>()

}
```

Inject your dependency by field

```
class MyActivity : AppCompatActivity() {

    val classB by inject<ClassB>()

}
```

Inject your dependency by field - on Android

```
fun main(){




}
```

```
fun main(){

    startKoin {
        modules(myModule)
    }

}
```

Koin DSL can become verbose 🤨

```
single { ClassB(get(),get(),get(),get() …) }
```

Improving Developer Experience

```
class ClassA()
class ClassB(val a : ClassA)

module {
    single { ClassA() }
    single { ClassB(get()) }
}
```

```
class ClassA()
class ClassB(val a : ClassA)

module {
    singleOf(::classA)
    singleOf(::classB)
}
```

- Easy to read/write
- Koin Semantic
- Consistent changes
- KMP Compatible

```
class MyClassA(val id : String)

module {
    single { (id : String) -> MyClassA(id) }
}
```

```
class MyClassA(val id : String)

module {
    single { (id : String) -> MyClassA(id) }
}
```

```
// in calling code
val classA : MyClassA by inject { parametersOf("_id_")}
```

```
class MyClassA(val id : String)

module {
    singleOf(::MyClassA)
}
```

```
// in calling code
val classA : MyClassA by inject { parametersOf("_id_")}
```

Warnings

- Named parameters
- Default Values
- Complex Kotlin Expression (builders ...)

Koin DSL Options

```
single { MyClass() }
```

```
single(
    named("_qualifier_")
) { MyClass() }
```

```
single(
    named("_qualifier_"),
    createdAtStart = true
) { MyClass() }
```

```
single(
    named("_qualifier_"),
    createdAtStart = true
) { MyClass() } bind MyComponent::class
```

```
single(
    named("_qualifier_"),
    createdAtStart = true
) { MyClass() } bind MyComponent::class
```

New DSL => Options DSL 🤔

```
singleOf(::MyClassA)
```

```
singleOf(::MyClassA){
    named("_qualifier_")
}
```

```
singleOf(::MyClassA){
    named("_qualifier_")
    createdAtStart()
}
```

```
singleOf(::MyClassA){
    named("_qualifier_")
    createdAtStart()
    bind<MyComponent>()
}
```

```
single(
    named("_qualifier_"),
    createdAtStart = true
) { MyClass() } bind MyComponent::class
```

```
singleOf(::MyClassA){
    named("_qualifier_")
    createdAtStart()
    bind<MyComponent>()
}
```

Modules Dependencies

```
val uiModule = module { }
val dataModule = module { }
```

Let’s start those modules

```
val uiModule = module { }
val dataModule = module { }

startKoin {
    modules(uiModule, dataModule)
}
```

Let’s start those modules

```
val uiModule = module { }
val dataModule = module { }
val appModules = listOf(uiModule,dataModule)

startKoin {
    modules(appModules)
}
```

Let’s start those modules

```
val uiModule = module { }
val dataModule = module { }
val appModules = uiModule + dataModule

startKoin {
    modules(appModules)
}
```

Scaling development => good modules organisation

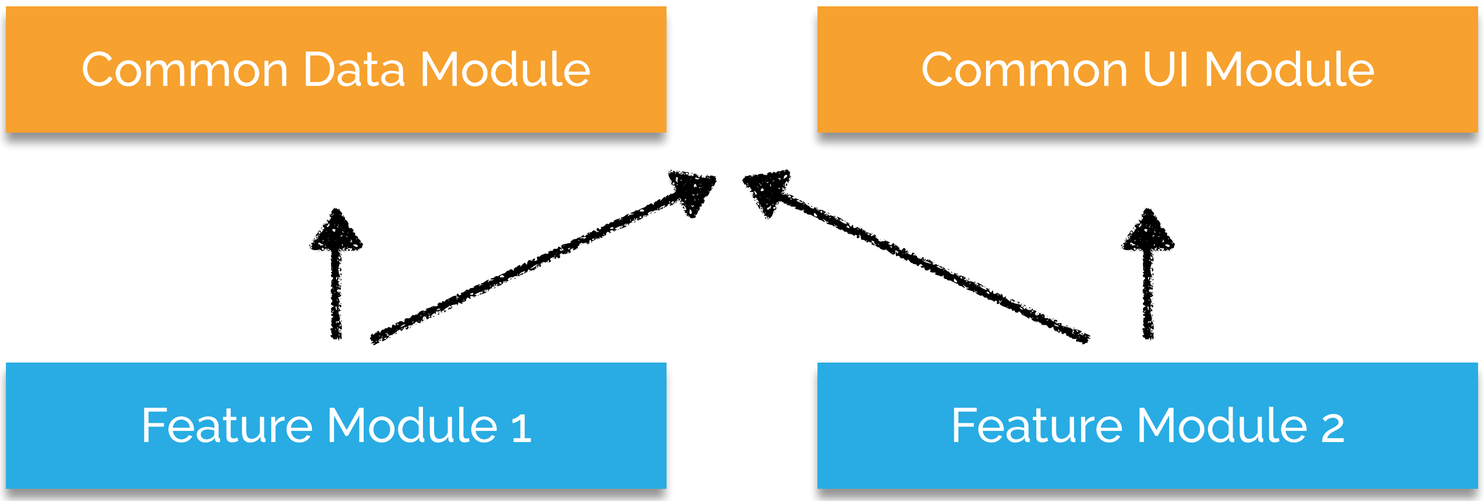
Let’s use includes()

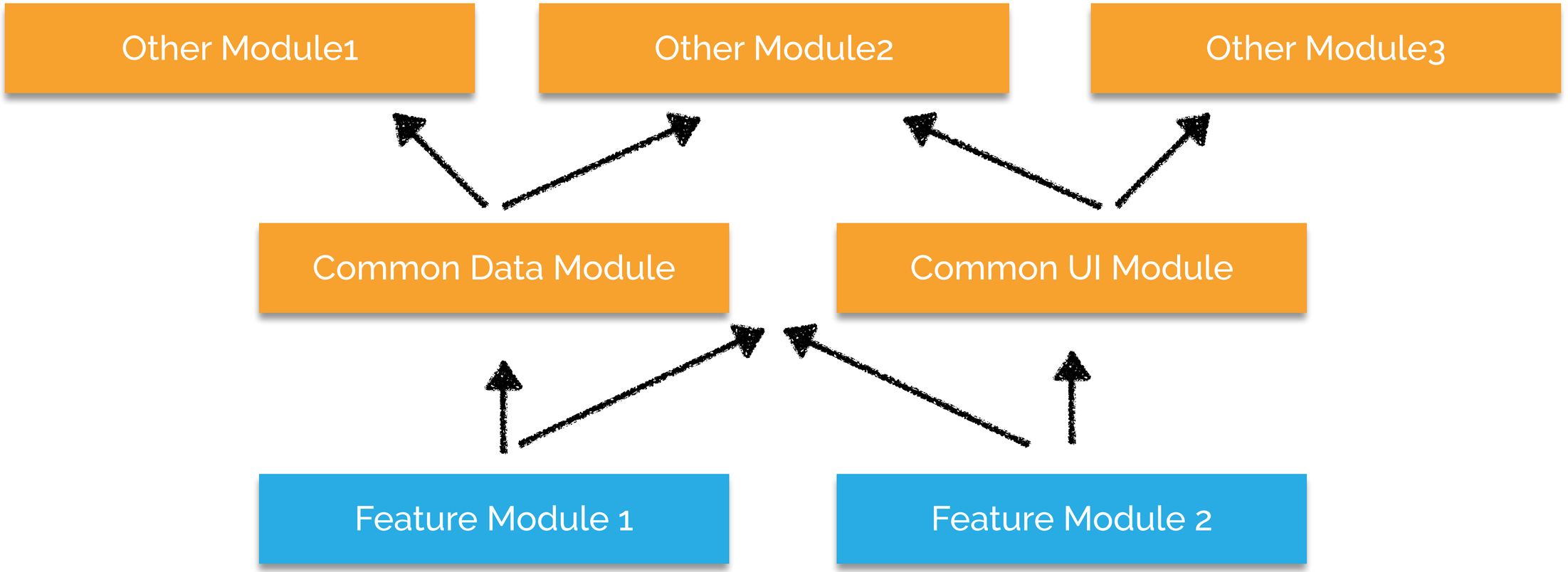
```
val uiModule = module { }
val dataModule = module { }
val appModules = listOf(uiModule,dataModule)
```

```
val uiModule = module { }
val dataModule = module { }

val appModules = module {
    includes(uiModule,dataModule)

 }
```

```
val uiModule = module { }
val dataModule = module { }
val appModules = module {
    includes(uiModule,dataModule)
}

startKoin{
    modules(appModules)
}
```

👉 Allow to reduce configuration complexity


Common Data Module
Common UI Module
Feature Module 1
Feature Module 2


Other Module1
Other Module2
Other Module3
Common Data Module
Common UI Module
Feature Module 1
Feature Module 2

All those new features
are Kotlin Multiplatform
Ready! 👍

```
// Latest version
koin_version = "3.2.3"  "3.3.2"
```

```
// Koin for Kotlin & Multiplatform
implementation “io.insert-koin:koin-core:$koin_version”
// Koin for Android
implementation “io.insert-koin:koin-android:$koin_version”
```

Keep your Koin config simple ✨

Less Code = Less Bugs

*Chet Haase

Why Annotations

Don’t reproduce existing solution

Bring new value!

Limits of Koin DSL 🤨

Kotlin Compiler Plugin! 🛠️

Kotlin Compiler Plugin… what for?

- Rewrite code on the fly
- Code generation
- Code analysis
- KMP compliance

…

✨ Google KSP ✨

- 👉 Configuration with Annotations
- 👉 Koin Semantic
- 👉 Generate Koin DSL for you

Definitions

```
class MyRepositoryImpl() : MyRepository
```

```
@Single
class MyRepositoryImpl() : MyRepository
```

```
@Single
class MyRepositoryImpl() : MyRepository


class MyPresenter(val repo : MyRepository)
```

```
@Single
class MyRepositoryImpl() : MyRepository

@Factory
class MyPresenter(val repo : MyRepository)
```

```
@Single
class MyRepositoryImpl() : MyRepository

@Factory
class MyPresenter(val repo : MyRepository)


class MyViewModel(val repo : MyRepository) : ViewModel()
```

```
@Single
class MyRepositoryImpl() : MyRepository

@Factory
class MyPresenter(val repo : MyRepository)

@KoinViewModel
class MyViewModel(val repo : MyRepository) : ViewModel()
```

```
@Single
class MyRepositoryImpl() : MyRepository

@Factory
class MyPresenter(val repo : MyRepository)

@KoinViewModel
class MyViewModel(val repo : MyRepository) : ViewModel()
```

- Automatic injection & bindings
- Nullable Types
- @InjectedParam

```
@Single
class MyRepositoryImpl() : MyRepository

@Factory
class MyPresenter(val repo : MyRepository)

@KoinViewModel
class MyViewModel(val repo : MyRepository) : ViewModel()
```

```
class MyActivity : AppCompatActivity() {

    val myPresenter : MyPresenter by inject()
    val myViewModel : MyViewModel by viewModel()
}
```

Modules

Module Class 👍

```
class MyModule
```

```
@Module
class MyModule
```

```
@Module
@ComponentScan("com.my.package")
class MyModule
```

```
@Module
class MyModule {

}
```

```
@Module
class MyModule {


    fun classB(a : ClassA) = ClassB(a)
}
```

```
@Module
class MyModule {

    @Single
    fun classB(a : ClassA) = ClassB(a)
}
```

```
@Module
class MyModuleA

@Module
class MyModuleB
```

```
@Module
class MyModuleA

@Module(includes = [MyModuleA::class])
class MyModuleB
```

Start Koin

```
@Module
class MyModule
```

```
@Module
class MyModule



startKoin {

    modules()
}
```

```
@Module
class MyModule



startKoin {

    modules(MyModule())
}
```

```
@Module
class MyModule

// .module generated in org.koin.ksp.generated.*
import org.koin.ksp.generated.*

startKoin {

    modules(MyModule().module)
}
```

Mix DSL & Annotations Modules

```
@Module
class MyModule

// .module generated in org.koin.ksp.generated.*
import org.koin.ksp.generated.*

startKoin {

    modules(myDSLModule, MyModule().module)
}
```

Fast Compilation Time 

~2 sec - 1000 components

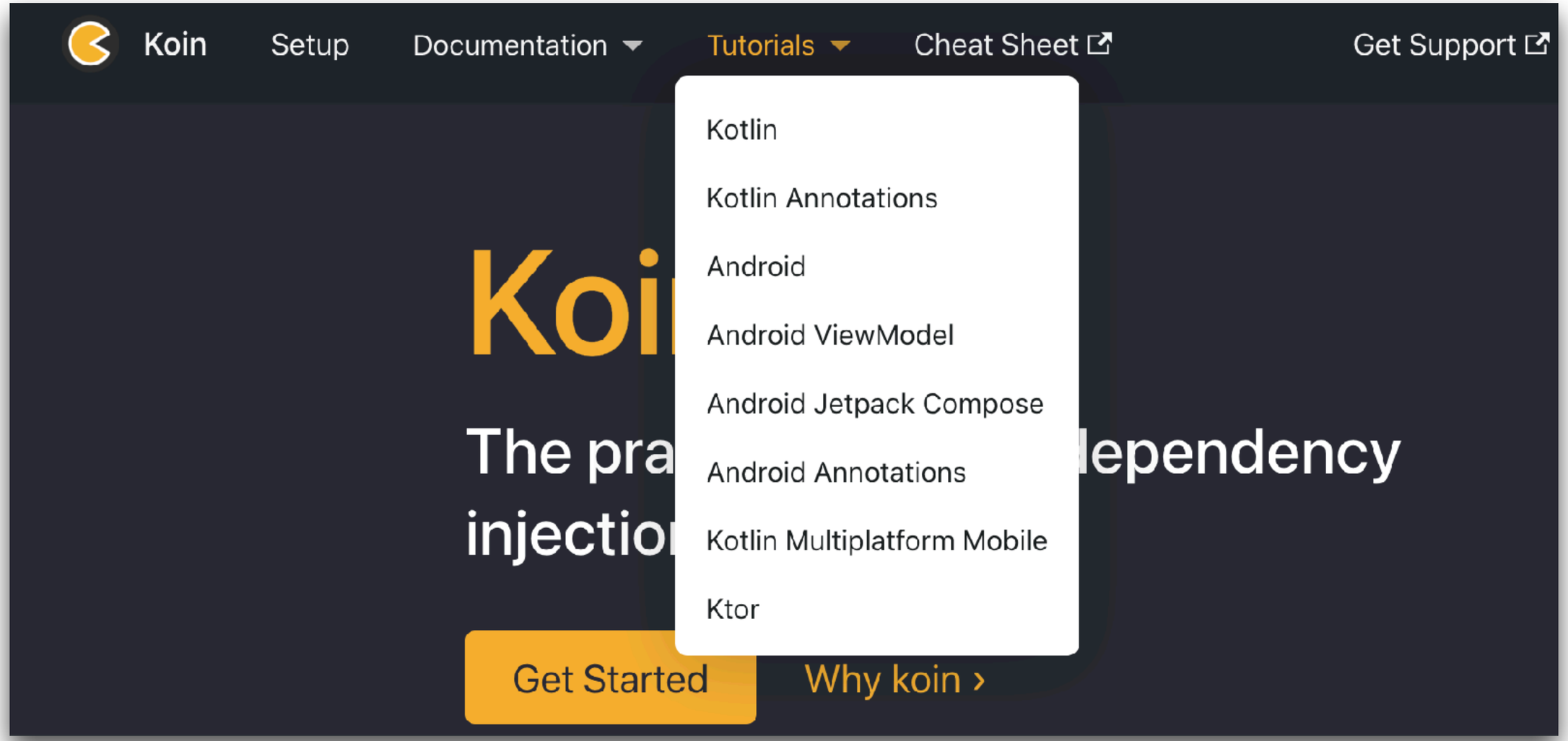
Easy to Debug 

Generated Koin DSL modules

DSL and Annotations

Choose what is best for you 👍

What's Next?



Just give a try - insert-koin.io

🗓️ Roadmap 2023

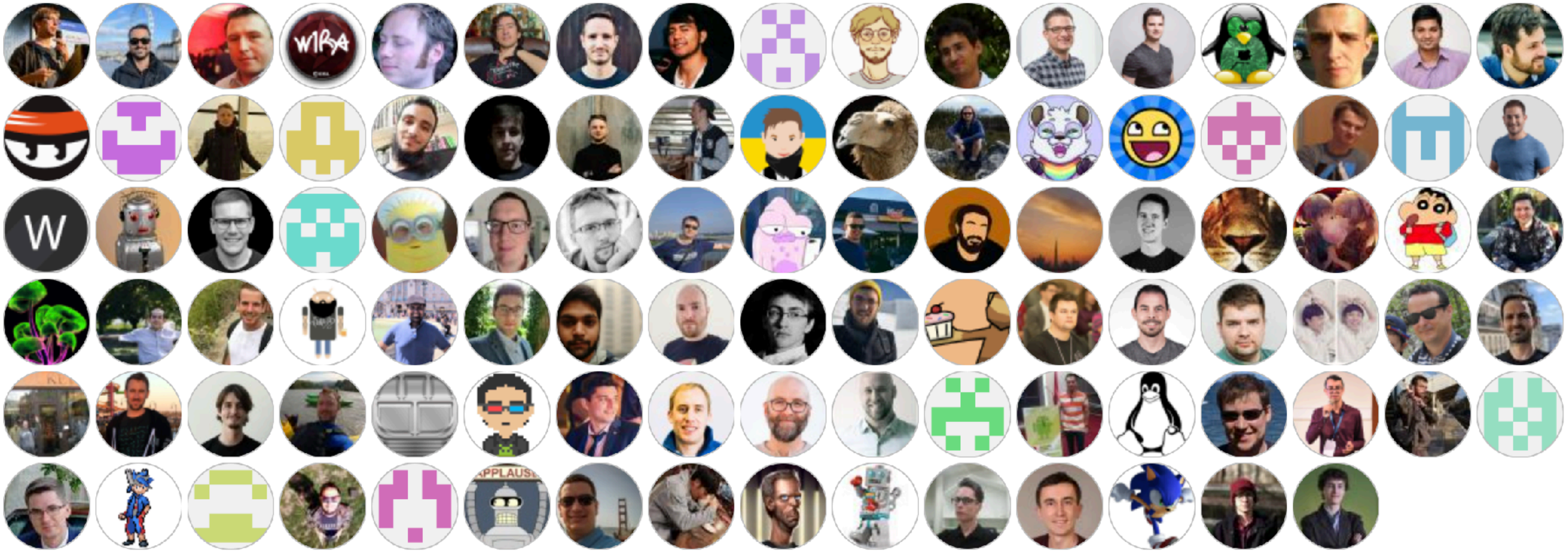
End of Track: Koin 3.2

Active Track: Koin 3.3

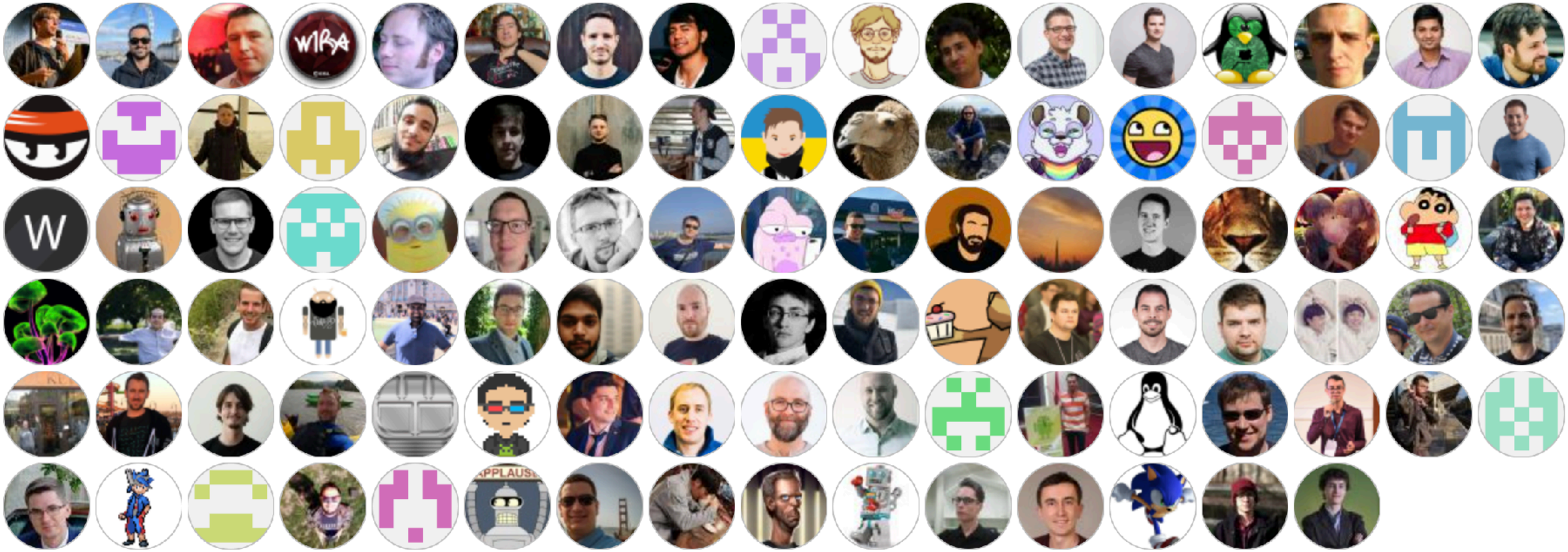
H1 Releases:
Koin & Koin Annotations 3.4

👉 Koin for Compose
👉 Kotlin Native interop
👉 Verify() API
👉 Ktor

Koin — open source
and community driven! 👍



Making things better, together!



insert-koin.io

@insertkoin_io

#koin

koin by kotzilla

Official Professional Support

kotzilla

kotzilla.io

@kotzilla_io

Thank You :)

Arnaud Giuliani
@arnogiu