INSERT KOIN

for dependency injection

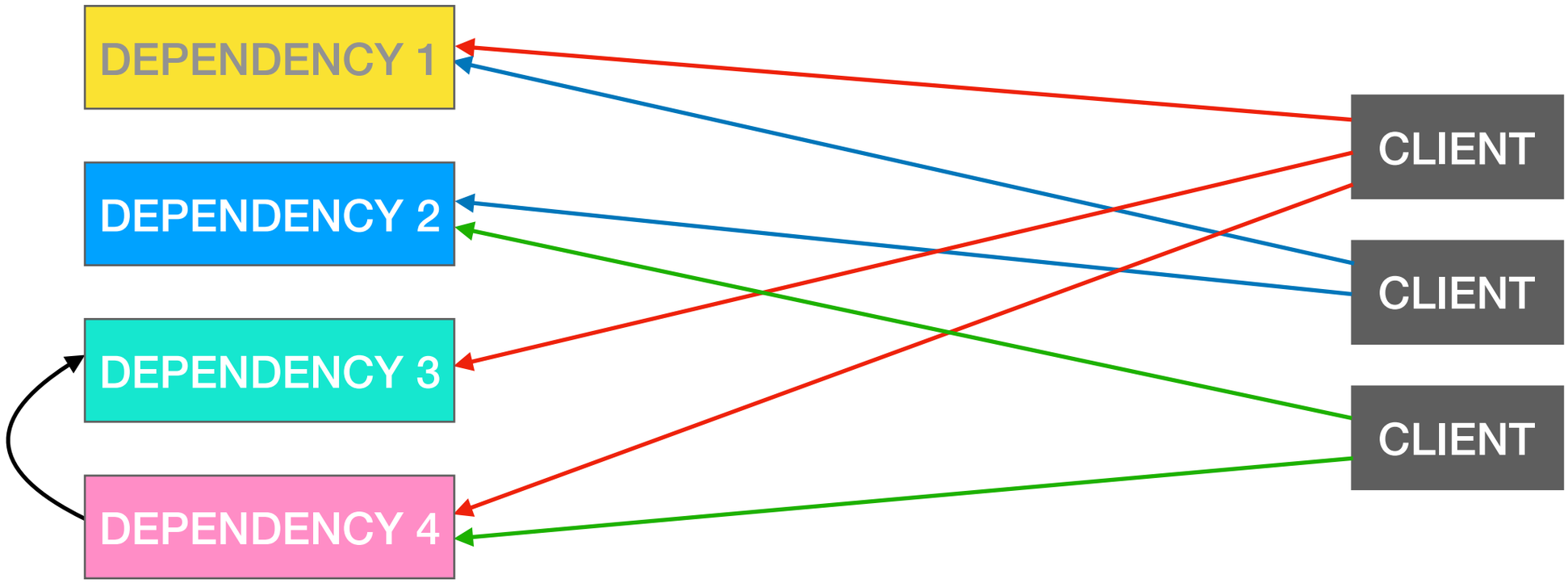
Кирилл Розов
Android Developer
APALON

Dependency Injection

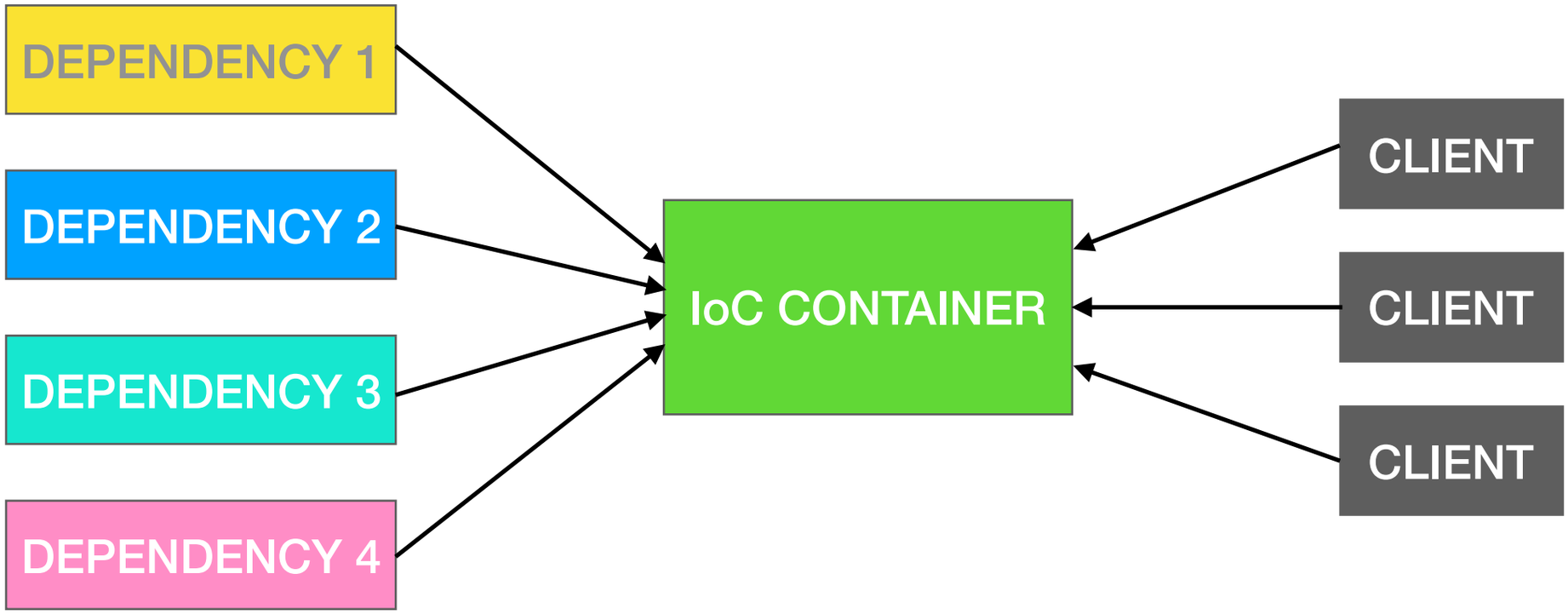
Inversion of Control (IoC) is a design principle in which custom-written portions of a computer program receive the flow of control from a generic framework

wikipedia.org/wiki/Inversion_of_control

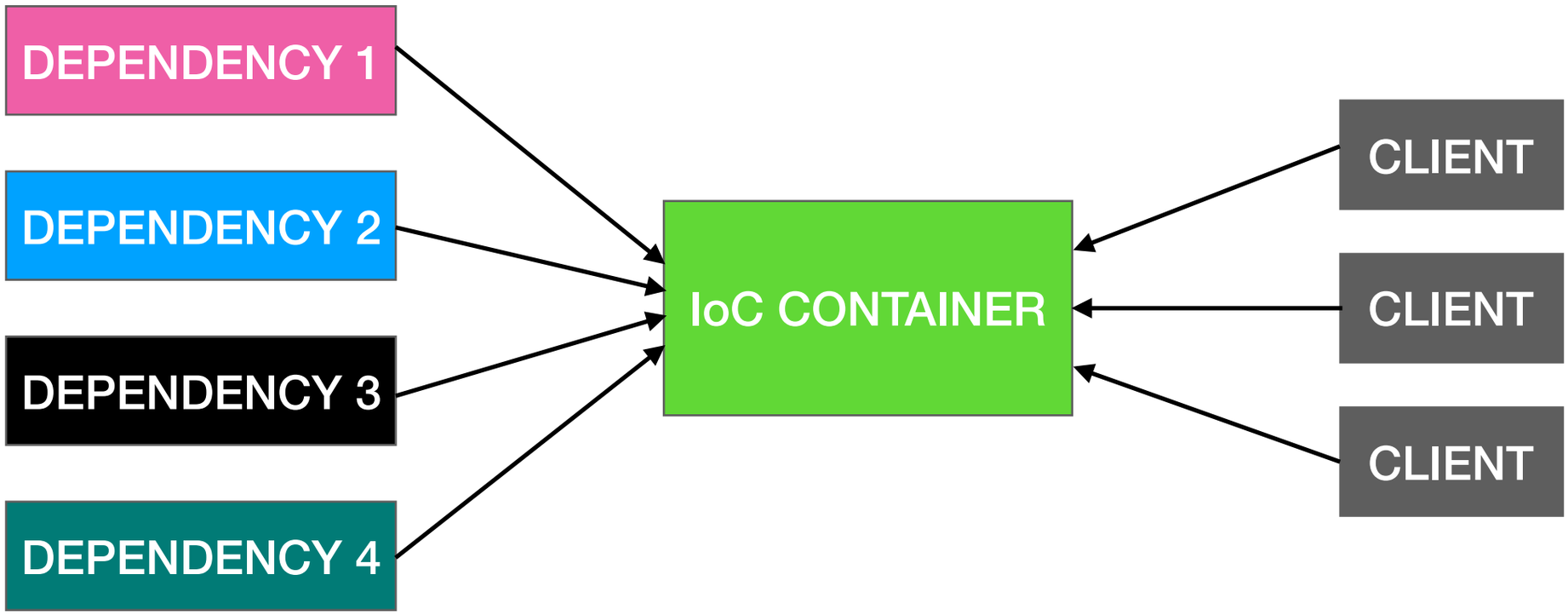
Inversion of Control



Inversion of Control



Inversion of Control



Popular DI

- Guice
- Spring DI
- Square Dagger
- Google Dagger 2
- Java EE CDI
- PicoContainer
- Kodein
- Koin

insert-Koin.io

Koin Supports

Android App

Arch Components

Standalone App

Spark Web App

Sample usage

```
val sampleModule = applicationContext {
    bean { ComponentA() }
}
```

```
class Application : KoinComponent {

    val component by inject<ComponentA>() // Inject (lazy)

    val component = get<ComponentA>() // Get (eager)
}
```

```
fun main(vararg args: String) {
    startKoin(listOf(sampleModule))
}
```

Init Koin

```
fun main(vararg args: String) {
    startKoin(listOf(sampleModule))
}
```

```
class SampleApplication : Application() {
    override fun onCreate() {
        startKoin(this, listOf(sampleModule))
    }
}
```

```
fun main(vararg args: String) {
    start(listOf(sampleModule)) { … }
}
```

Provide dependencies

```
applicationContext {

    bean { ComponentA() } // singletone

    factory { ComponentB() } // factory
}
```

Provide dependencies

```
applicationContext {

    bean { ComponentB() }

    bean { ComponentA(get<ComponentB>()) }
}
```

Named dependencies

```
applicationContext {

    bean(“debug”) { ComponentB() }

    bean(“prod”) { ComponentB() }

    bean { ComponentA(get(“debug”)) }
}
```

```
class Application : KoinComponent {

    val component by inject<ComponentB>(“prod”)
}
```

Type binding

```
applicationContext {

    bean { ComponentImpl() }

    bean { ComponentImpl() as Component }

    bean { ComponentImpl() } bind Component::class
}
```

Properties

```
class RestService(url: String)
```

```
applicationContext {

    bean { RestService(getProperty(“url”)) }
}
```

```
applicationContext {

    bean { RestService(getProperty(“url”, “http://localhost”)) }
}
```

```
val key1Property: String by property(“key1”)
```

Additional properties

```
// Properties has type Map<String, Any>
startKoin(properties =
    mapOf(“key1" to "value", “key2” to 1)
)
```

Properties Sources

- ***koin.properties*** in JAR resources
- ***koin.properties*** in assets 
- Environment properties 

Environment variables

```
startKoin(useEnvironmentProperties = true)
```


```
// Get user name from properties
val key1Property: String by property("USER")
```

Parameters

```
applicationContext {
    factory { params: ParametersProvider ->
        NewsDetailsPresenter(params[NEWS_ID])
    }
}
```

```
val presenter: NewsDetailsPresenter
      by inject { mapOf(NEWS_ID to "sample") }
```

```
interface ParametersProvider {

    operator fun <T> get(key: String): T

    fun <T> getOrNull(key: String): T?
}
```

Contexts

```
applicationContext {
    context("main") {
        bean { ComponentA() }

        bean { ComponentB() }
    }
}
```

```
class MainActivity : Activity() {
    override fun onStop() {
        releaseContext("main")
    }
}
```

Context isolation

```
applicationContext { // Root
    context("A") {
        context("B") {
            bean { ComponentA() }
        }
    }

    context("C") {
        bean { ComponentA() }
    }
}
```

Android Arch Components

Default Way

```
class ListFragment : Fragment() {

    val listViewModel =
          ViewModelProviders.of(this).get(ListViewModel::class.java)
}
```

Koin Way

```
applicationContext {

    viewModel { ListViewModel() }
}
```

```
class ListFragment : Fragment() {

    val listViewModel by viewModel<ListViewModel>()

    val listViewModel = getViewModel<ListViewModel>()
}
```

ViewModel with arguments

```
class DetailViewModel(id: String) : ViewModel()
```

```
class DetailFactory(val id: String) : ViewModelProvider.Factory {

    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        if (modelClass == DetailViewModel::class.java) {
            return DetailViewModel(id) as T
        }
        error("Can't create ViewModel for class='$modelClass'")
    }
}
```

```
class ListFragment : Fragment() {

    val listViewModel =
          ViewModelProviders.of(this, DetailFactory(id))
                .get(TeacherViewModel::class.java)
}
```

Koin Way

```
applicationContext {

    viewModel { params -> DetailViewModel(params["id"]) }
}
```

```
class ListFragment : Fragment() {

    val listViewModel
          by viewModel<ListViewModel> { mapOf("id" to "sample") }

    val listViewModel =
          getViewModel<ListViewModel> { mapOf("id" to "sample") }
}
```

Logging

```
(KOIN) :: Resolve [Presenter] ~ Factory[class=Presenter]
(KOIN) ::    Resolve [Repository] ~ Bean[class=Repository]
(KOIN) ::       Resolve [Datasource] ~ Bean[class=DebugDatasource, binds~(Datasource)]
(KOIN) ::       (*) Created
(KOIN) ::    (*) Created
(KOIN) :: (*) Created

(KOIN) :: Resolve [Repository] ~ Factory[class=Repository]
(KOIN) ::     Resolve [DebugDatasource] ~ Bean[class=DebugDatasource, binds~(Datasource)]
(KOIN) :: (*) Created
```

```
get<Presenter>()
get<Presenter>()
```

```
applicationContext {
    factory { Presenter(get()) }
    bean { Repository(get()) }
    bean { DebugDataSource() } bind DateSource::class
}
```

```
class Presenter(repository: Repository)
class Repository(dateSource: DateSource)
interface DateSource
class DebugDataSource() : DateSource
```

Crash Logs

Cyclic dependencies

```
org.koin.error.BeanInstanceCreationException: Can't create bean Bean[class=ComponentA] due to error :
  BeanInstanceCreationException: Can't create bean Bean[class=ComponentB] due to error :
  BeanInstanceCreationException: Can't create bean Bean[class=ComponentA] due to error :
  DependencyResolutionException: Cyclic dependency detected while resolving class ComponentA
```

```
get<ComponentA>()
```

```
applicationContext {
    bean { ComponentA(get()) }
    bean { ComponentB(get()) }
}
```

```
class ComponentA(component: ComponentB)

class ComponentB(component: ComponentA)
```

Missing dependency

```
org.koin.error.DependencyResolutionException:
    No definition found for ComponentC – Check your definitions and contexts visibility
```

```
get<ComponentC>()
```

```
applicationContext {
    bean { ComponentA(get()) }
    bean { ComponentB(get()) }
}
```

```
class ComponentA(component: ComponentB)

class ComponentB(component: ComponentA)
```

Graph Validation

Graph validation

```
class DryRunTest : KoinTest {

    @Test
    fun dryRunTest() {
        startKoin(listOf(sampleModule))
        dryRun()
    }
}
```

```
val sampleModule = applicationContext {
   bean { ComponentA(get()) }
   factory { ComponentB(get()) }
}
```

```
class ComponentA(component: ComponentB)
class ComponentB(component: ComponentC)
class ComponentC()
```

Declare

Write with the Koin DSL what you need to assemble:

```
// Given some classes
class Controller(val service : BusinessService)
class BusinessService()

// just declare it
val myModule = applicationContext {
  bean { Controller(get()) }
  bean { BusinessService() }
}
```

Learn Koin DSL in 5 min

Start

Use the **startKoin()** function to start Koin in your application:

Kotlin | Android

```
class MyApplication : Application() {
  override fun onCreate(){
    super.onCreate()
    // start Koin!
    startKoin(this, listOf(myModule))
  }
}
```

Write your app with Koin

Read the development guides

Inject

You're now ready! The Koin DSL help you make **injection by constructors**. Now just use it:

```
class Controller(val service : BusinessService){
  // service has been injected
}
```

Koin can be easily embedded with your favorite Java/Kotlin SDK, and already provide some dedicated support module.

Examples:

* Inject an Android Activity
* Architecture Component ViewModel

Choose a Koin module

Explore the documentation

Koin vs Dagger 2

	Koin	Dagger2
Simpler usage	**Yes**	-
How work	**No reflection or code generation**	Codegeneration
Support	**Kotlin, Java, Android, Arch Components, Spark, Koin**	Java, Android
Transitive dependency injection	-	**Yes**
Dependencies declaration	In modules, DSL	Annotated functions in module Annotations on class
Library Size	83 Kb (v 0.9.1)	38 Kb (v 2.15)
Generic type resolve	-	**Yes**
Named dependencies	**Yes**	**Yes**
Scopes	**Yes**	**Yes**
Lazy injection	**Yes**	**Yes**
Graph validation	Dry Run	**Compile time**
Logs	**Yes**	-
Additional Features	Environment property injection, parametrised injection	Multibindings, Reusable Scope

Thanks

insert-Koin.io

Кирилл Розов
Android Developer

APALON

krl.rozov@gmail.com

krlrozov