BRAVE NEW WORLD!?

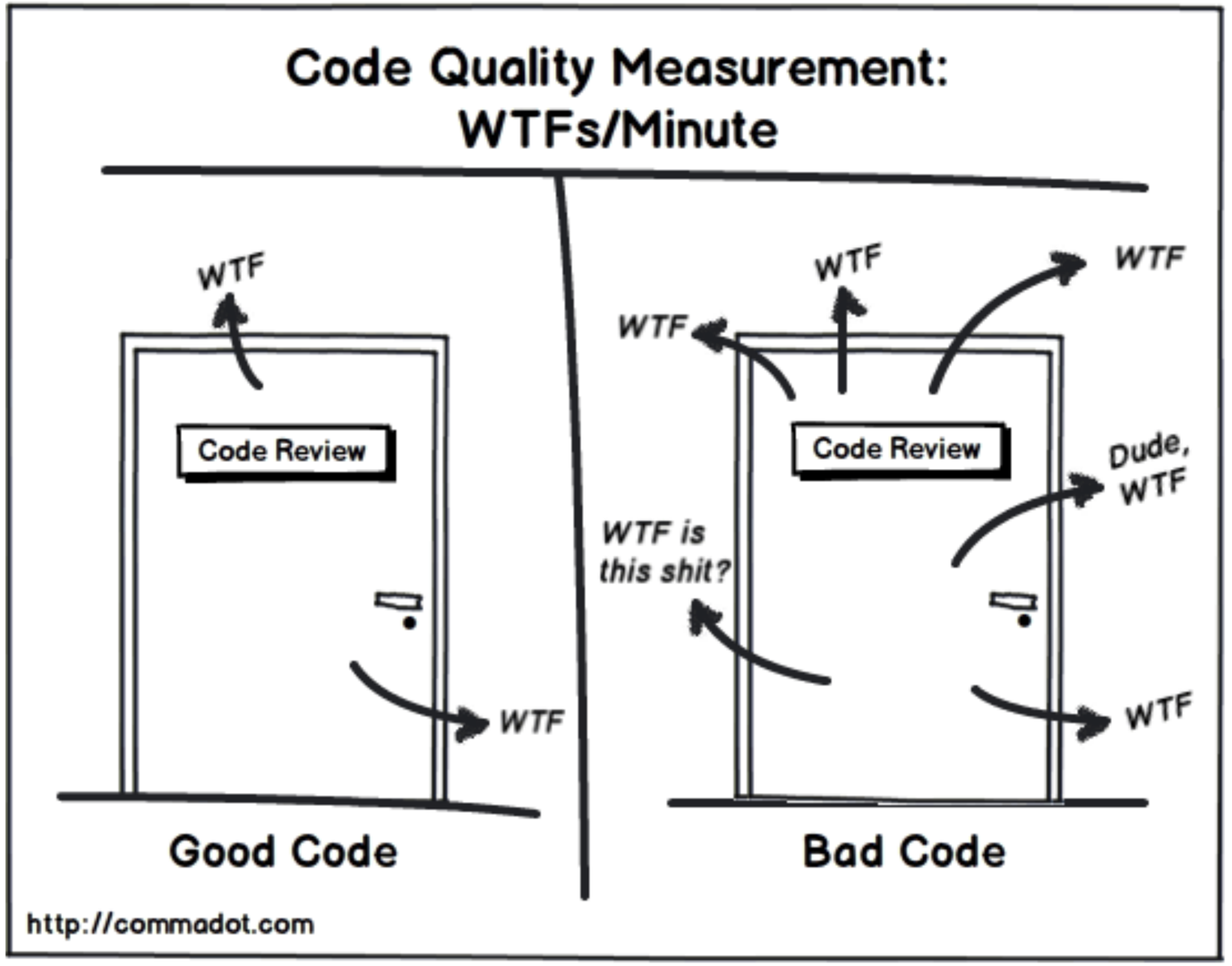
EXPLORING APP DESIGN WITH ARCHITECTURE COMPONENTS AND KOTLIN

Hendrik Kokocinski

Hendrik.Kokocinski@weltn24.de

WELT

github.com/hkokocin

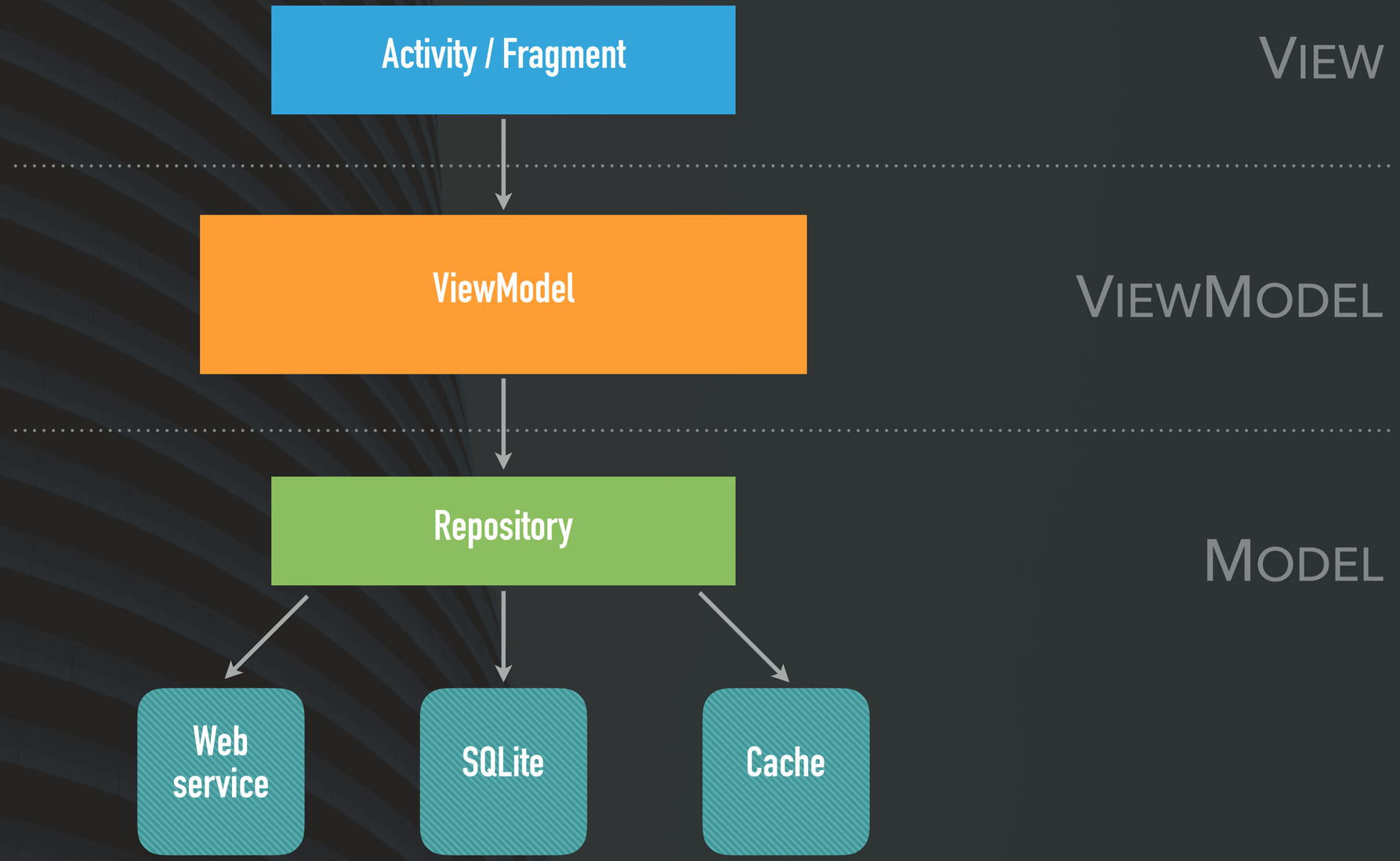

Code Quality Measurement:
WTFs/Minute
WTF
Code Review
WTF
Good Code
WTF
WTF
WTF
WTF
Code Review
Dude,
WTF
WTF is
this shit?
WTF
Bad Code
http://commadot.com


WTF
WTF
WTF
DUDE, WTF
WTF IS THIS SHIT
OMG, WTF
WTF
WTF
WTF
WTF


WTF
WTF
DUDE,
WTF
WTF IS THIS
SHIT
WTF
WTF
WTF


WTF
DUDE,
WTF
WTF
WTF


WTF
WTF
Available
Lot#


Activity / Fragment
ViewModel
Repository
Web service
SQLite
Cache
View
ViewModel
Model

ViewModel

LiveData

Lifecycle Handling

Room

ViewModel

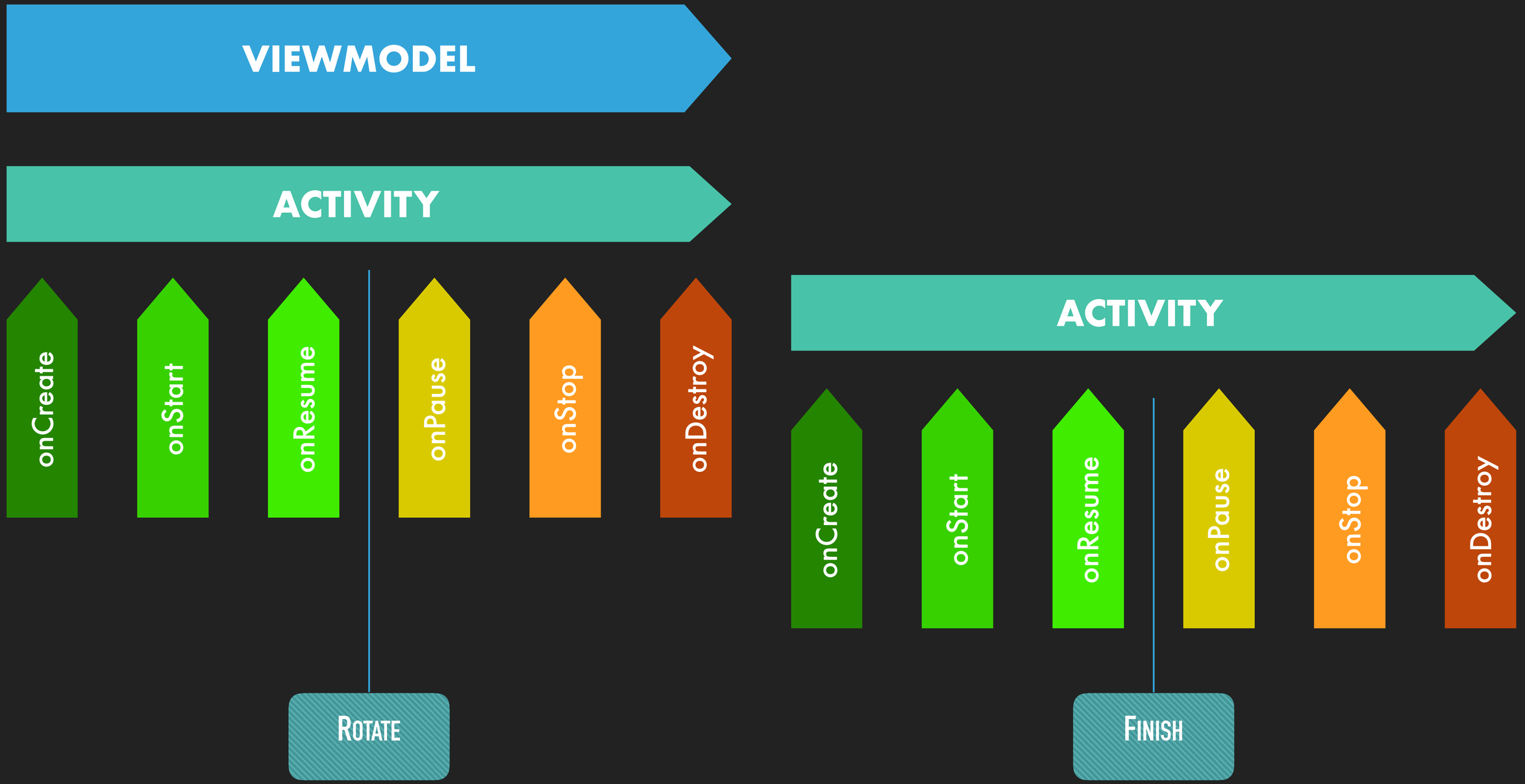
LiveData

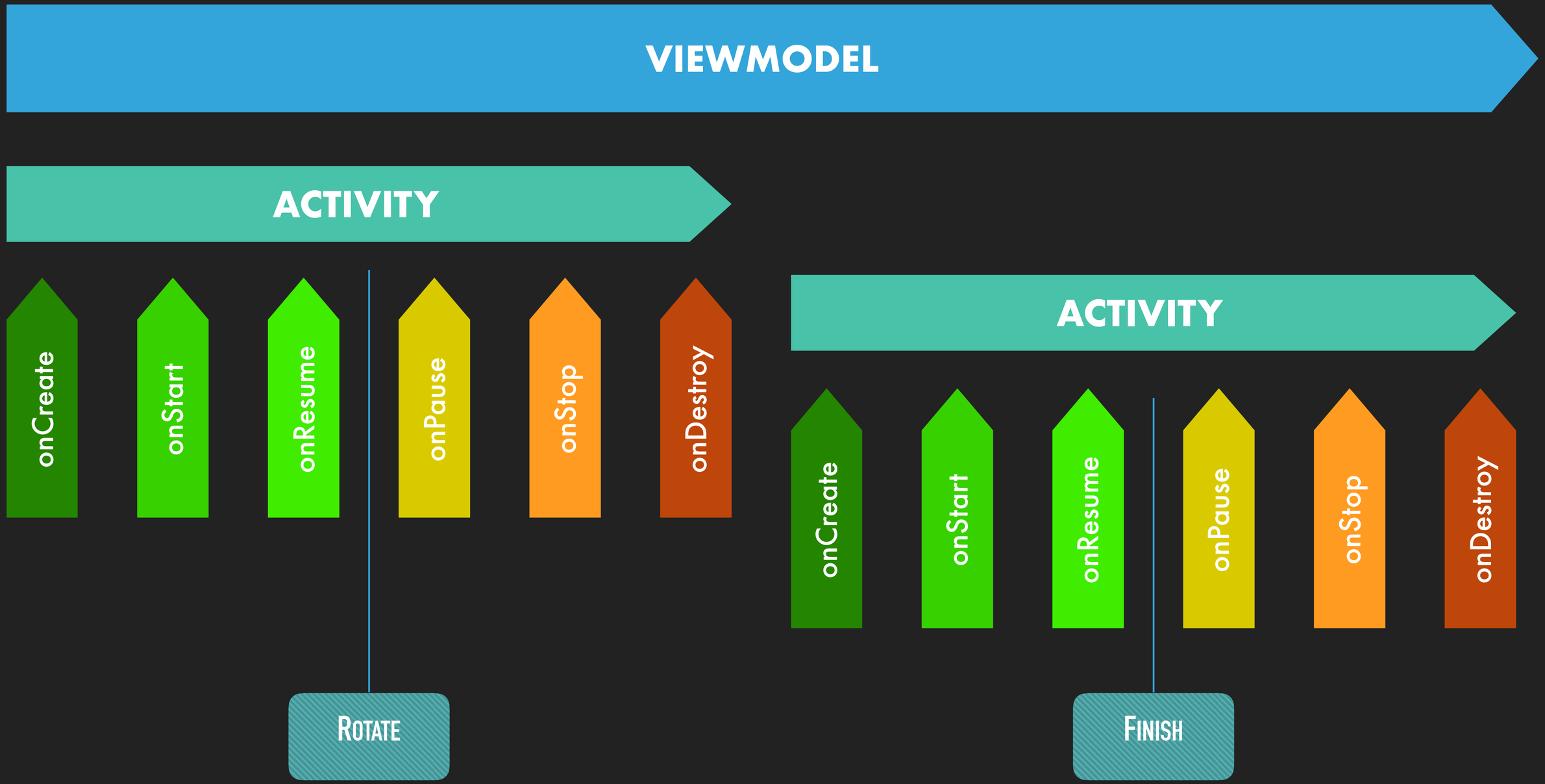
Lifecycle Handling

ViewModel

LiveData

Lifecycle Handling


VIEWMODEL
ACTIVITY
onCreate
onStart
onResume
onPause
onStop
onDestroy
ROTATE
ACTIVITY
onCreate
onStart
onResume
onPause
onStop
onDestroy
FINISH


VIEWMODEL
ACTIVITY
onCreate
onStart
onResume
onPause
onStop
onDestroy
ROTATE
ACTIVITY
onCreate
onStart
onResume
onPause
onStop
onDestroy
FINISH

Demo

```
class GitHubActivity : LifecycleActivity() {

}
```

Kodein

https://github.com/SalomonBrys/Kodein

```
class GitHubActivity : LifecycleActivity() {

    private val injector = gitHubActivityScope(this)
}
```

```
class GitHubActivity : LifecycleActivity() {

    private val injector = gitHubActivityScope(this)

    private val adapter by lazy { injector.instance<WidgetAdapter>() }
    private val inputMethodManager by lazy { injector.instance<InputMethodManager>() }
}
```

```
class GitHubActivity : LifecycleActivity() {

    private val injector = gitHubActivityScope(this)

    private val adapter by lazy { injector.instance<WidgetAdapter>() }
    private val inputMethodManager by lazy { injector.instance<InputMethodManager>() }
}
```

```
class GitHubActivity : LifecycleActivity() {

    private val injector = gitHubActivityScope(this)

    private val adapter by lazy { injector.instance<WidgetAdapter>() }
    private val inputMethodManager by lazy { injector.instance<InputMethodManager>() }
}
```

```
class GitHubActivity : LifecycleActivity() {

    private val injector = gitHubActivityScope(this)

    private val adapter by lazy { injector.instance<WidgetAdapter>() }
    private val inputMethodManager by lazy { injector.instance<InputMethodManager>() }

    private val rvResult by lazy { findViewById<RecyclerView>(R.id.rv_result) }
}
```

```
class GitHubActivity : LifecycleActivity() {

    private val injector = gitHubActivityScope(this)

    private val adapter by lazy { injector.instance<WidgetAdapter>() }
    private val inputMethodManager by lazy { injector.instance<InputMethodManager>() }

    private val rvResult by lazy { findViewById<RecyclerView>(R.id.rv_result) }
    private val etSearch by lazy { findViewById<EditText>(R.id.et_search) }
    private val ivSearch by lazy { findViewById<ImageView>(R.id.iv_search) }
    private val progressBar by lazy { findViewById<ProgressBar>(R.id.progress) }
}
```

```
class GitHubActivity : LifecycleActivity() {
    […]

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.github_activity)
        initRecyclerView()
   }
}
```

```java
public interface LifecycleOwner {
    Lifecycle getLifecycle();
}
```

```
class GitHubActivity : LifecycleActivity() {
    […]

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.github_activity)
        initRecyclerView()

        ViewModelProviders.of(this)
    }
}
```

```
class GitHubActivity : LifecycleActivity() {
    […]
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.github_activity)
        initRecyclerView()

        ViewModelProviders.of(this, ViewModelFactory(injector))
    }
}
```

```
@Suppress("UNCHECKED_CAST")
class ViewModelFactory(private val injector: Kodein) : ViewModelProvider.Factory {

    override fun <T : ViewModel> create(modelClass: Class<T>): T = when (modelClass) {
        UserSearchViewModel::class.java -> injector.instance<UserSearchViewModel>()
        else                            -> throw IllegalAccessException(“…”)
    } as T
}
```

```
@Suppress("UNCHECKED_CAST")
class ViewModelFactory(private val injector: Kodein) : ViewModelProvider.Factory {

    override fun <T : ViewModel> create(modelClass: Class<T>): T = when (modelClass) {
        UserSearchViewModel::class.java -> injector.instance<UserSearchViewModel>()
        else                            -> throw IllegalAccessException(“…”)
    } as T
}
```

```
@Suppress("UNCHECKED_CAST")
class ViewModelFactory(private val injector: Kodein) : ViewModelProvider.Factory {

    override fun <T : ViewModel> create(modelClass: Class<T>): T = when (modelClass) {
        UserSearchViewModel::class.java -> injector.instance<UserSearchViewModel>()
        else                            -> throw IllegalAccessException(“…”)
    } as T
}
```

```
@Suppress("UNCHECKED_CAST")
class ViewModelFactory(private val injector: Kodein) : ViewModelProvider.Factory {

    override fun <T : ViewModel> create(modelClass: Class<T>): T = when (modelClass) {
        UserSearchViewModel::class.java -> injector.instance<UserSearchViewModel>()
        else                            -> throw IllegalAccessException("…")
    } as T
}
```

```
class GitHubActivity : LifecycleActivity() {
    […]
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.github_activity)
        initRecyclerView()

        val viewModel = ViewModelProviders
            .of(this, ViewModelFactory(injector))
            .get(UserSearchViewModel::class.java)
    }
}
```

```
class GitHubActivity : LifecycleActivity() {
    […]
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.github_activity)
        initRecyclerView()

        val viewModel = ViewModelProviders
            .of(this, ViewModelFactory(injector))
            .get(UserSearchViewModel::class.java)

        ivSearch.setOnClickListener {
            val search = etSearch.text.toString()
            if (search.isNotBlank()) {
                viewModel.searchUsers(search)
                inputMethodManager.hideSoftInputFromWindow(etSearch.windowToken, 0)
            }
        }
    }
}
```

```
class GitHubActivity : LifecycleActivity() {
    […]
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.github_activity)
        initRecyclerView()

        val viewModel = ViewModelProviders
            .of(this, ViewModelFactory(injector))
            .get(UserSearchViewModel::class.java)

        ivSearch.setOnClickListener {
            val search = etSearch.text.toString()
            if (search.isNotBlank()) {
                viewModel.searchUsers(search)
                inputMethodManager.hideSoftInputFromWindow(etSearch.windowToken, 0)
            }
        }
    }
}
```

```
class UserSearchViewModel() : ViewModel() {}
```

```
class UserSearchViewModel() : ViewModel() {

    override fun onCleared() {}
}
```

```
class UserSearchViewModel() : ViewModel() {

    private val disposables = CompositeDisposable()

    override fun onCleared() = disposables.clear()
}
```

```
class UserSearchViewModel(
        private val repository: GitHubRepository
) : ViewModel() {

    private val disposables = CompositeDisposable()

    override fun onCleared() = disposables.clear()
}
```

```
class UserSearchViewModel(
        private val repository: GitHubRepository
) : ViewModel() {

    private val disposables = CompositeDisposable()

    fun searchUsers(search: String) {
        repository.searchUser(search)
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe { (items) -> }
    }

    override fun onCleared() = disposables.clear()
}
```

ViewModel

LiveData

Lifecycle Handling

```
class UserSearchViewModel(
        private val repository: GitHubRepository
) : ViewModel() {

    private val disposables = CompositeDisposable()

    fun searchUsers(search: String) {
        repository.searchUser(search)
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe { (items) -> }
    }

    override fun onCleared() = disposables.clear()
}
```

```
class UserSearchViewModel(
        private val repository: GitHubRepository
) : ViewModel() {
    val users = MutableLiveData<List<GitHubUser>>()
    val showProgress = MutableLiveData<Boolean>()

    private val disposables = CompositeDisposable()

    fun searchUsers(search: String) {
        showProgress.value = true
        repository.searchUser(search)
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe { (items) -> }
    }

    override fun onCleared() = disposables.clear()
}
```

```
class UserSearchViewModel(
        private val repository: GitHubRepository
) : ViewModel() {
    val users = MutableLiveData<List<GitHubUser>>()
    val showProgress = MutableLiveData<Boolean>()

    private val disposables = CompositeDisposable()

    fun searchUsers(search: String) {
        showProgress.value = true
        repository.searchUser(search)
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe { (items) ->
                users.value = items
                showProgress.value = false
            }
    }

    override fun onCleared() = disposables.clear()
}
```

```
class UserSearchViewModel(
        private val repository: GitHubRepository
) : ViewModel() {
    val users = MutableLiveData<List<GitHubUser>>()
    val showProgress = MutableLiveData<Boolean>()

    private val disposables = CompositeDisposable()

    fun searchUsers(search: String) {
        showProgress.value = true
        repository.searchUser(search)
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe { (items) ->
                users.value = items
                showProgress.value = false
            }
    }

    override fun onCleared() = disposables.clear()
}
```

```
class UserSearchViewModel(
        private val repository: GitHubRepository
) : ViewModel() {
    val users = MutableLiveData<List<GitHubUser>>()
    val showProgress = MutableLiveData<Boolean>()

    private val disposables = CompositeDisposable()

    fun searchUsers(search: String) {
        showProgress.value = true
        repository.searchUser(search)
            .subscribeOn(Schedulers.io())
            .subscribe { (items) ->
                users.postValue(items)
                showProgress.postValue(false)
            }
    }

    override fun onCleared() = disposables.clear()
}
```

```
class UserSearchViewModel(
        private val repository: GitHubRepository
) : ViewModel() {
    val users = MutableLiveData<List<GitHubUser>>()
    val showProgress = MutableLiveData<Boolean>()

    private val disposables = CompositeDisposable()

    fun searchUsers(search: String) {
        showProgress.value = true
        repository.searchUser(search)
            .subscribeOn(Schedulers.io())
            .subscribe { (items) ->
                users.postValue(items)
                showProgress.postValue(false)
            }
    }

    override fun onCleared() = disposables.clear()
}
```

```
class GitHubActivity : LifecycleActivity() {
    […]
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.github_activity)
        initRecyclerView()

        val viewModel = ViewModelProviders
            .of(this, ViewModelFactory(injector))
            .get(UserSearchViewModel::class.java)

        ivSearch.setOnClickListener {
            val search = etSearch.text.toString()
            if (search.isNotBlank()) {
                viewModel.searchUsers(search)
                inputMethodManager.hideSoftInputFromWindow(etSearch.windowToken, 0)
            }
        }
    }
}
```

```
class GitHubActivity : LifecycleActivity() {
    […]
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.github_activity)
        initRecyclerView()

        val viewModel = ViewModelProviders
            .of(this, ViewModelFactory(injector))
            .get(UserSearchViewModel::class.java)

        ivSearch.setOnClickListener {…}

        viewModel.users.observe(
                this,
                Observer { adapter.setItems(it ?: emptyList()) }
        )
    }
}
```

```
class GitHubActivity : LifecycleActivity() {
    […]

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.github_activity)
        initRecyclerView()

        val viewModel = ViewModelProviders
            .of(this, ViewModelFactory(injector))
            .get(UserSearchViewModel::class.java)

        ivSearch.setOnClickListener {…}

        viewModel.users.observe(
                this,
                Observer { adapter.setItems(it ?: emptyList()) }
        )
    }
}
```

```
public interface LifecycleOwner {
    Lifecycle getLifecycle();
}
```

```
class GitHubActivity : LifecycleActivity() {
    […]
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.github_activity)
        initRecyclerView()

        val viewModel = ViewModelProviders
            .of(this, ViewModelFactory(injector))
            .get(UserSearchViewModel::class.java)

        ivSearch.setOnClickListener {…}

        viewModel.users.observe(
                this,
                Observer { adapter.setItems(it ?: emptyList()) }
        )
    }
}
```

```
public interface Observer<T> {
    void onChanged(@Nullable T t);
}
```

```
class GitHubActivity : LifecycleActivity() {
    […]

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.github_activity)
        initRecyclerView()

        val viewModel = ViewModelProviders
            .of(this, ViewModelFactory(injector))
            .get(UserSearchViewModel::class.java)

        ivSearch.setOnClickListener {…}

        viewModel.users.observe(
                this,
                Observer { adapter.setItems(it ?: emptyList()) }
        )

        viewModel.showProgress.observe(
                this,
                Observer { progressBar.visibility =
                        if (it == true) View.VISIBLE else View.GONE
                }
        )
    }
}
```

```
class GitHubActivity : LifecycleActivity() {
    […]
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.github_activity)
        initRecyclerView()

        val viewModel = ViewModelProviders
            .of(this, ViewModelFactory(injector))
            .get(UserSearchViewModel::class.java)

        ivSearch.setOnClickListener {…}

        viewModel.users.observe(
                this,
                Observer { adapter.setItems(it ?: emptyList()) }
        )

        viewModel.showProgress.observe(
                this,
                Observer { progressBar.visibility =
                        if (it == true) View.VISIBLE else View.GONE
                }
        )
    }
}
```

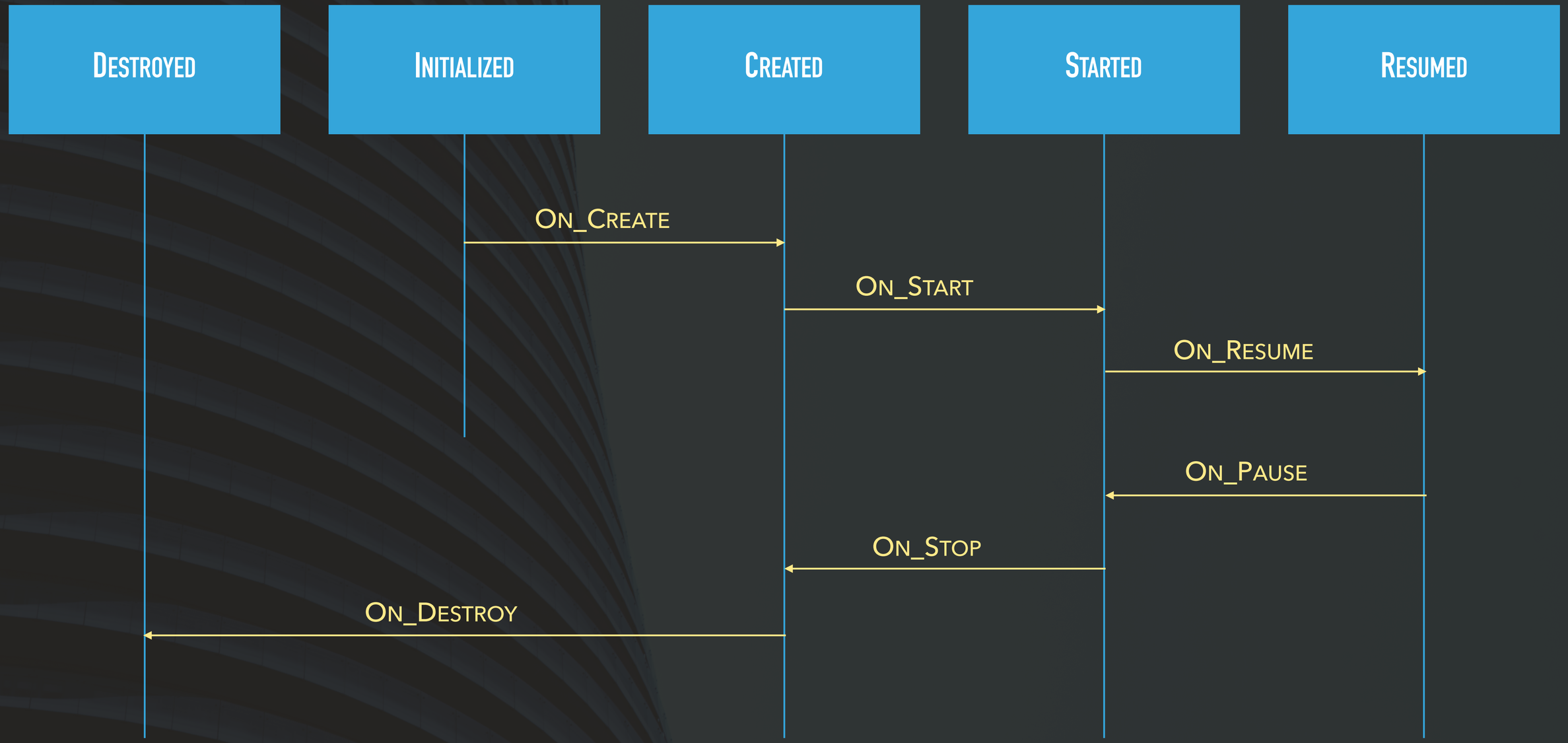
```
class GitHubActivity : LifecycleActivity() {
    […]

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.github_activity)
        initRecyclerView()

        val viewModel = ViewModelProviders
            .of(this, ViewModelFactory(injector))
            .get(UserSearchViewModel::class.java)

        ivSearch.setOnClickListener {…}

        viewModel.users.observe(
                this,
                Observer { adapter.setItems(it ?: emptyList()) }
        )

        viewModel.showProgress.observe(
                this,
                Observer { progressBar.visibility =
                        if (it == true) View.VISIBLE else View.GONE
                }
        )
    }
}
```

```
class GitHubActivity : LifecycleActivity() {
    […]
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.github_activity)
        initRecyclerView()

        val viewModel = ViewModelProviders
            .of(this, ViewModelFactory(injector))
            .get(UserSearchViewModel::class.java)

        ivSearch.setOnClickListener {…}

        viewModel.state.observe(
                this,
                Observer { it?.apply { updateViews(this) }}
        )
    }

    private fun updateViews(state: UserSearchViewState) {
        adapter.setItems(state.users)
        progressBar.visibility = if (state.showProgress) View.VISIBLE else View.GONE
    }
}
```

ViewModel

LiveData

Lifecycle Handling


DESTROYED
INITIALIZED
CREATED
STARTED
RESUMED
ON_CREATE
ON_START
ON_RESUME
ON_PAUSE
ON_STOP
ON_DESTROY

ViewModel

LiveData

Lifecycle Handling

```kotlin
class GitHubActivity : LifecycleActivity() {

    private val injector = gitHubActivityScope(this)
    private val adapter by lazy { injector.instance<WidgetAdapter>() }
    private val inputMethodManager by lazy { injector.instance<InputMethodManager>() }

    private val rvResult by lazy { findViewById<RecyclerView>(R.id.rv_result) }
    private val etSearch by lazy { findViewById<EditText>(R.id.et_search) }
    private val ivSearch by lazy { findViewById<ImageView>(R.id.iv_search) }
    private val progressBar by lazy { findViewById<ProgressBar>(R.id.progress) }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.github_activity)
        initRecyclerView()

        val viewModel = ViewModelProviders
            .of(this, ViewModelFactory(injector))
            .get(UserSearchViewModel::class.java)

        ivSearch.setOnClickListener {
            val search = etSearch.text.toString()
            if (search.isNotBlank()) {
                viewModel.searchUsers(search)
                inputMethodManager.hideSoftInputFromWindow(etSearch.windowToken, 0)
            }
        }

        viewModel.state.observe(
                this,
                Observer { it?.apply { updateViews(this) } }
        )
    }

    private fun updateViews(state: UserSearchViewState) {
        adapter.setItems(state.users)
        progressBar.visibility = if (state.showProgress) View.VISIBLE else View.GONE
    }

    private fun initRecyclerView() {
        rvResult.layoutManager = LinearLayoutManager(this)
        rvResult.adapter = adapter
        adapter.addWidget { injector.instance<UserWidget>() }
    }
}
```

```
class GitHubActivity : LifecycleActivity() {

    private val injector = gitHubActivityScope(this)
    private val view: GitHubView by lazy { injector.instance<GitHubView>() }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        val root = layoutInflater.inflate(R.layout.github_activity, null)
        setContentView(root)

        view.root = root
        lifecycle.addObserver(view)
    }
}
```

```
class GitHubActivity : LifecycleActivity() {

    private val injector = gitHubActivityScope(this)
    private val view: GitHubView by lazy { injector.instance<GitHubView>() }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        val root = layoutInflater.inflate(R.layout.github_activity, null)
        setContentView(root)

        view.root = root
        lifecycle.addObserver(view)
    }
}
```

```
class GitHubActivity : LifecycleActivity() {

    private val injector = gitHubActivityScope(this)
    private val view: GitHubView by lazy { injector.instance<GitHubView>() }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        val root = layoutInflater.inflate(R.layout.github_activity, null)
        setContentView(root)

        view.root = root
        lifecycle.addObserver(view)
    }
}
```

```
class GitHubActivity : LifecycleActivity() {

    private val injector = gitHubActivityScope(this)
    private val view: GitHubView by lazy { injector.instance<GitHubView>() }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        val root = layoutInflater.inflate(R.layout.github_activity, null)
        setContentView(root)

        view.root = root
        lifecycle.addObserver(view)
    }
}
```

```
class GitHubActivity : LifecycleActivity() {

    private val injector = gitHubActivityScope(this)
    private val view: GitHubView by lazy { injector.instance<GitHubView>() }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        val root = layoutInflater.inflate(R.layout.github_activity, null)
        setContentView(root)

        view.root = root
        lifecycle.addObserver(view)
    }
}
```

```
class GitHubView() : LifecycleObserver
```

```
class GitHubView(
        private val adapter: WidgetAdapter,
        private val inputMethodManager: InputMethodManager,
        private val userWidgetProvider: () -> UserWidget,
        private val viewModel: UserSearchViewModel
) : LifecycleObserver
```

```
class GitHubView(
        private val adapter: WidgetAdapter,
        private val inputMethodManager: InputMethodManager,
        private val userWidgetProvider: () -> UserWidget,
        private val viewModel: UserSearchViewModel
) : LifecycleObserver {

    lateinit var root: View
}
```

```
class GitHubView(
        private val adapter: WidgetAdapter,
        private val inputMethodManager: InputMethodManager,
        private val userWidgetProvider: () -> UserWidget,
        private val viewModel: UserSearchViewModel
) : LifecycleObserver {

    lateinit var root: View

    private val rvResult: RecyclerView by viewId(R.id.rv_result)
    private val etSearch: EditText by viewId(R.id.et_search)
    private val ivSearch: ImageView by viewId(R.id.iv_search)
    private val progressBar: ProgressBar by viewId(R.id.progress)
}
```

```
class GitHubView(
        private val adapter: WidgetAdapter,
        private val inputMethodManager: InputMethodManager,
        private val userWidgetProvider: () -> UserWidget,
        private val viewModel: UserSearchViewModel
) : LifecycleObserver {

    lateinit var root: View

    private val rvResult: RecyclerView by viewId(R.id.rv_result)
    private val etSearch: EditText by viewId(R.id.et_search)
    private val ivSearch: ImageView by viewId(R.id.iv_search)
    private val progressBar: ProgressBar by viewId(R.id.progress)

    private fun <T : View> viewId(resourcesId: Int): Lazy<T>
            = lazy { root.findViewById<T>(resourcesId) }
}
```

```
class GitHubView(…) : LifecycleObserver {

    (…)

    @OnLifecycleEvent(Lifecycle.Event.ON_CREATE)
    fun onCreate() {
    }
}
```

```
class GitHubView(…) : LifecycleObserver {

    (…)

    @OnLifecycleEvent(Lifecycle.Event.ON_CREATE)
    fun onCreate(lifecycleOwner: LifecycleOwner) {
    }
}
```

```
class GitHubView(…) : LifecycleObserver {

    (…)

    @OnLifecycleEvent(Lifecycle.Event.ON_CREATE)
    fun onCreate(lifecycleOwner: LifecycleOwner) {

        val state = lifecycleOwner.lifecycle.currentState
    }
}
```

```
class GitHubView(…) : LifecycleObserver {

    (…)

    @OnLifecycleEvent(Lifecycle.Event.ON_CREATE)
    fun onCreate(lifecycleOwner: LifecycleOwner) {

        val state = lifecycleOwner.lifecycle.currentState
        if(state == Lifecycle.State.STARTED)
    }
}
```

```
class GitHubView(…) : LifecycleObserver {

    (…)

    @OnLifecycleEvent(Lifecycle.Event.ON_CREATE)
    fun onCreate(lifecycleOwner: LifecycleOwner) {

        val state = lifecycleOwner.lifecycle.currentState
        if(state == Lifecycle.State.STARTED)
        if(state.isAtLeast(Lifecycle.State.CREATED))
    }
}
```

```
class GitHubView(…) : LifecycleObserver {

    (…)

    @OnLifecycleEvent(Lifecycle.Event.ON_CREATE)
    fun onCreate(lifecycleOwner: LifecycleOwner) {
        initRecyclerView()

        ivSearch.setOnClickListener {…}

        viewModel.state.observe(
                lifecycleOwner,
                Observer { adapter.setItems(it ?: emptyList()) }
        )
    }
}
```

```
class GitHubView(…) : LifecycleObserver {

    (…)

    @OnLifecycleEvent(Lifecycle.Event.ON_CREATE)
    fun onCreate(lifecycleOwner: LifecycleOwner) {
        initRecyclerView()

        ivSearch.setOnClickListener {…}

        viewModel.state.observe(
                lifecycleOwner,
                Observer { adapter.setItems(it ?: emptyList()) }
        )
    }
}
```

```
class GitHubView(…) : LifecycleObserver {

    (…)

    @OnLifecycleEvent(Lifecycle.Event.ON_CREATE)
    fun onCreate(lifecycleOwner: LifecycleOwner) {
        initRecyclerView()

        ivSearch.setOnClickListener {…}

        viewModel.state.observe(
                lifecycleOwner,
                Observer { adapter.setItems(it ?: emptyList()) }
        )
    }
}
```

```
class GitHubView(…) : LifecycleObserver {

    (…)

    @OnLifecycleEvent(Lifecycle.Event.ON_CREATE)
    fun initRecyclerView() {…}

    @OnLifecycleEvent(Lifecycle.Event.ON_CREATE)
    fun initSearchView() = ivSearch.setOnClickListener {…}

    @OnLifecycleEvent(Lifecycle.Event.ON_CREATE)
    fun observeUsers(lifecycleOwner: LifecycleOwner) = viewModel.state.observe(
            lifecycleOwner,
            Observer { adapter.setItems(it ?: emptyList()) }
    )
}
```

MVP

Databinding

SUMMARY

- Still in beta
- Designed to work on their own - pick what works for you
- Use Lifecycle handling to improve separation of concerns

THANK YOU!