APRIL 08+09
DROIDCON BOS19

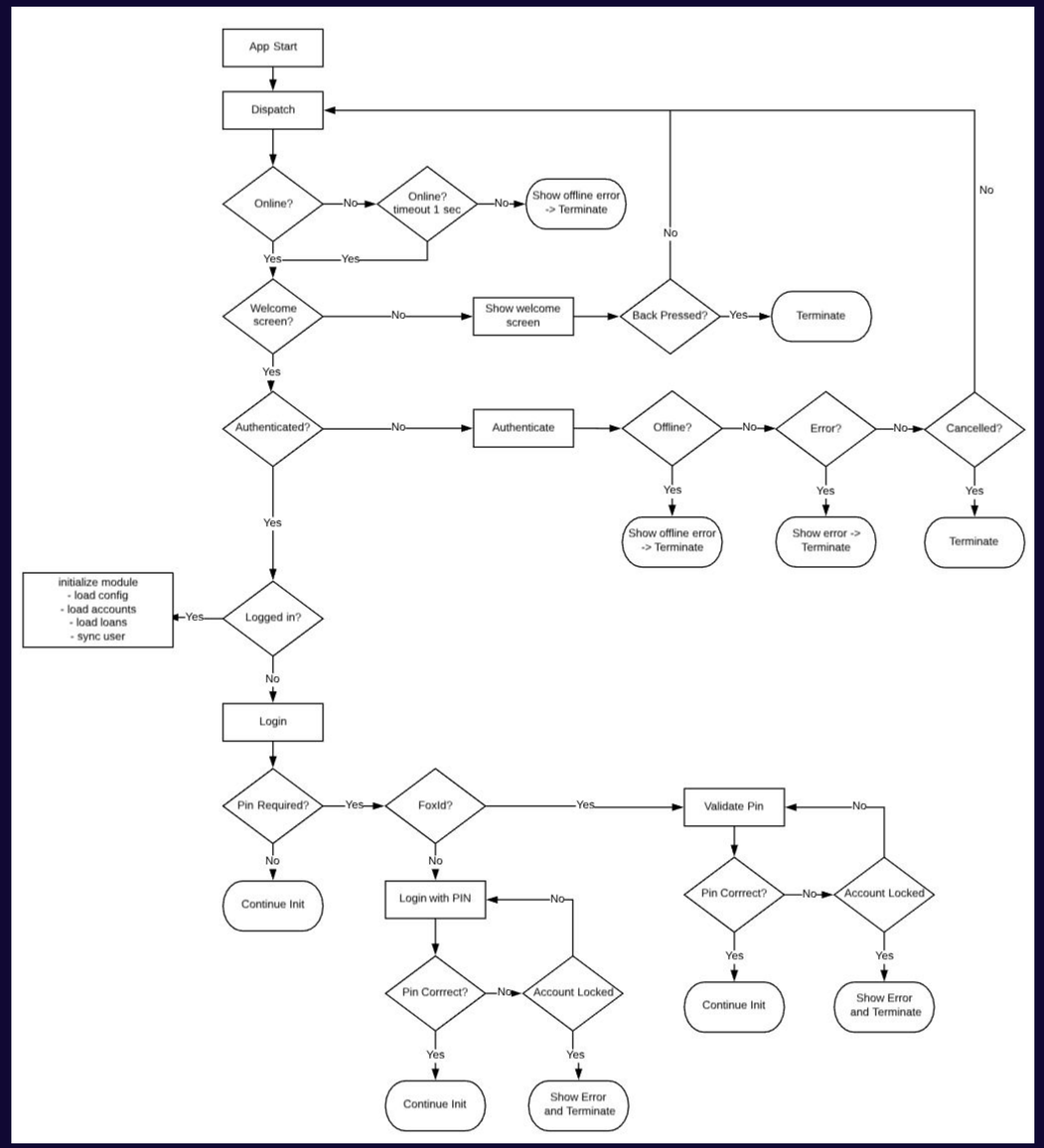
# Finite State Machine to the rescue

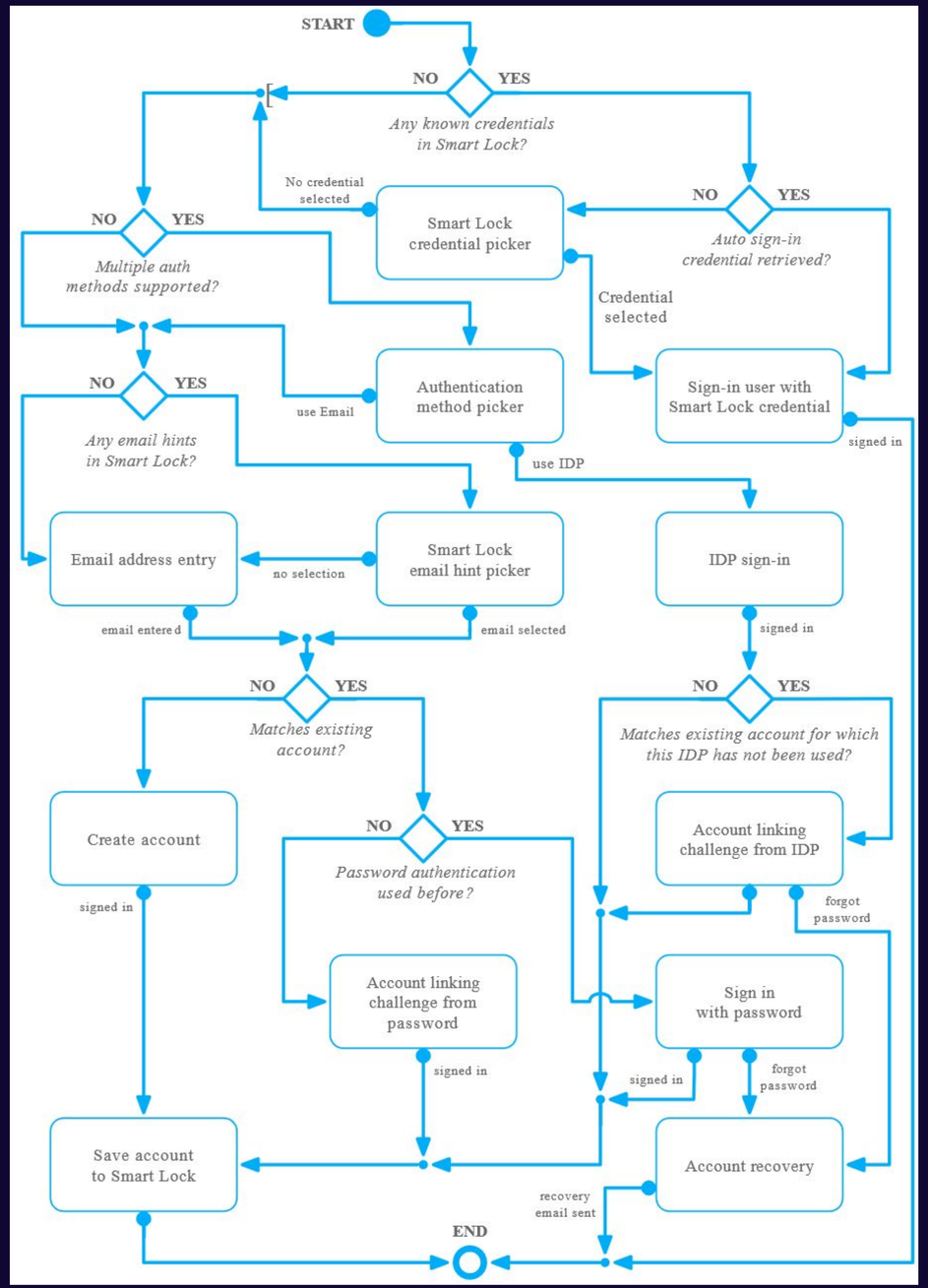
How to get complex workflows under control

**Emanuel Moecklin, CTO**
Stackoverflow: https://bit.ly/2YyXzYz
Github: https://github.com/1gravity
LinkedIn: https://bit.ly/2YxH5zX

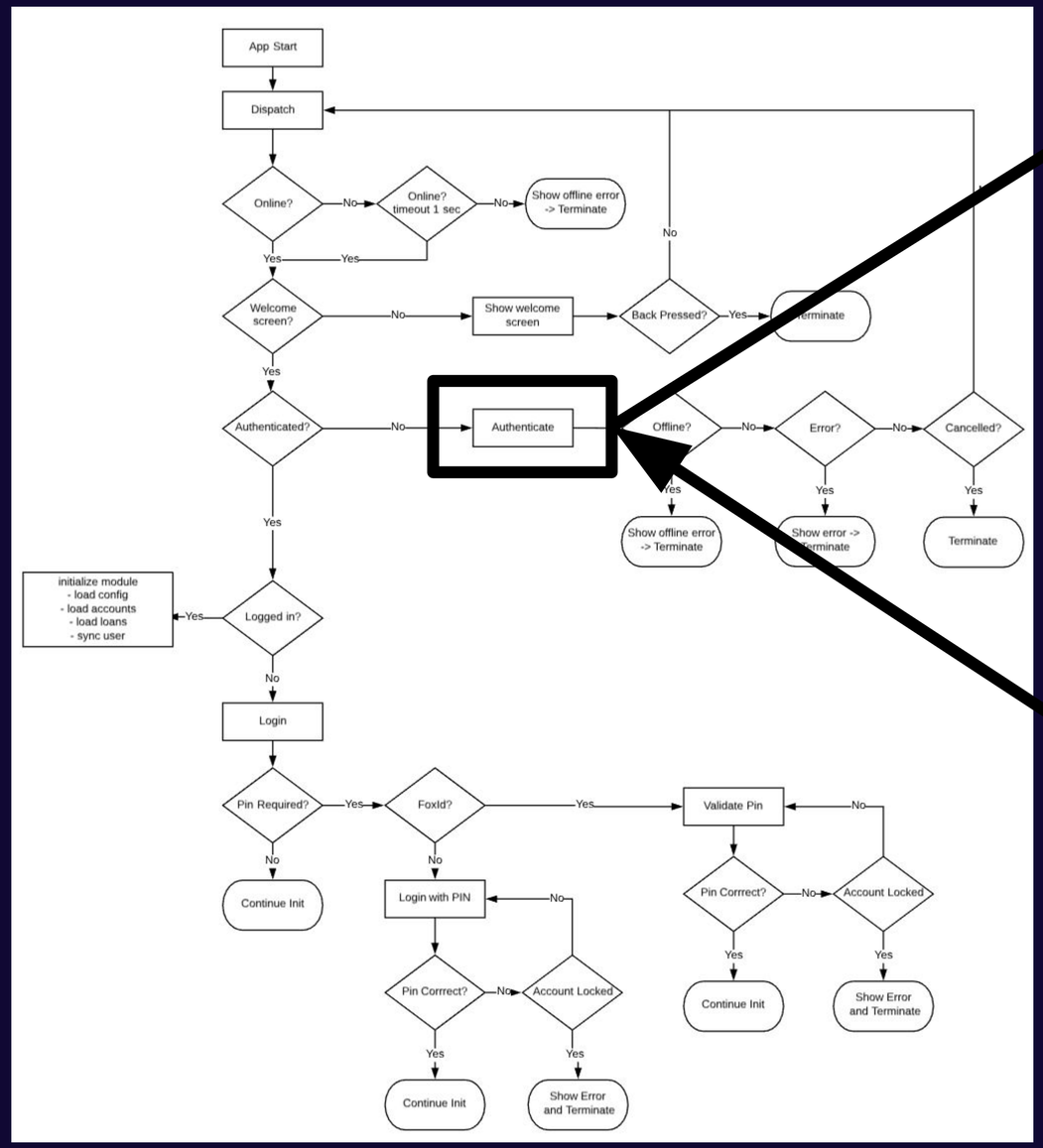
Airfox

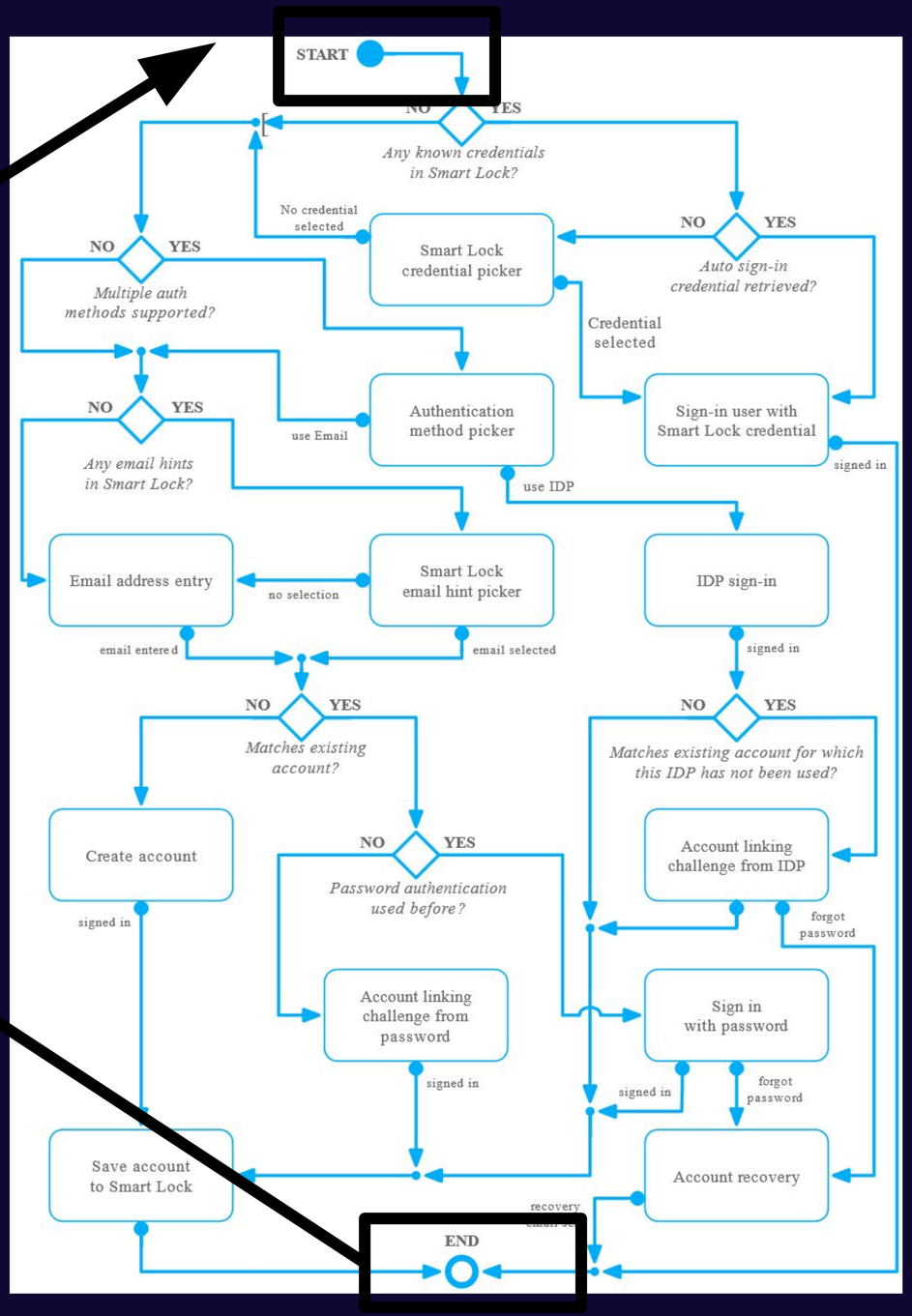
# The Problem


App Start
Dispatch
Online?
No
Online? timeout 1 sec
No
Show offline error -> Terminate
Yes
Yes
No
No
Welcome screen?
No
Show welcome screen
Back Pressed?
Yes
Terminate
Yes
Authenticated?
No
Authenticate
Offline?
No
Error?
No
Cancelled?
Yes
Yes
Yes
Yes
Show offline error -> Terminate
Show error -> Terminate
Terminate
initialize module
- load config
- load accounts
- load loans
- sync user
Yes
Logged in?
No
Login
Pin Required?
Yes
FoxId?
Yes
Validate Pin
No
No
No
Continue Init
Login with PIN
No
Pin Corrrect?
No
Account Locked
Yes
Yes
Pin Corrrect?
No
Account Locked
Continue Init
Show Error and Terminate
Yes
Yes
Continue Init
Show Error and Terminate

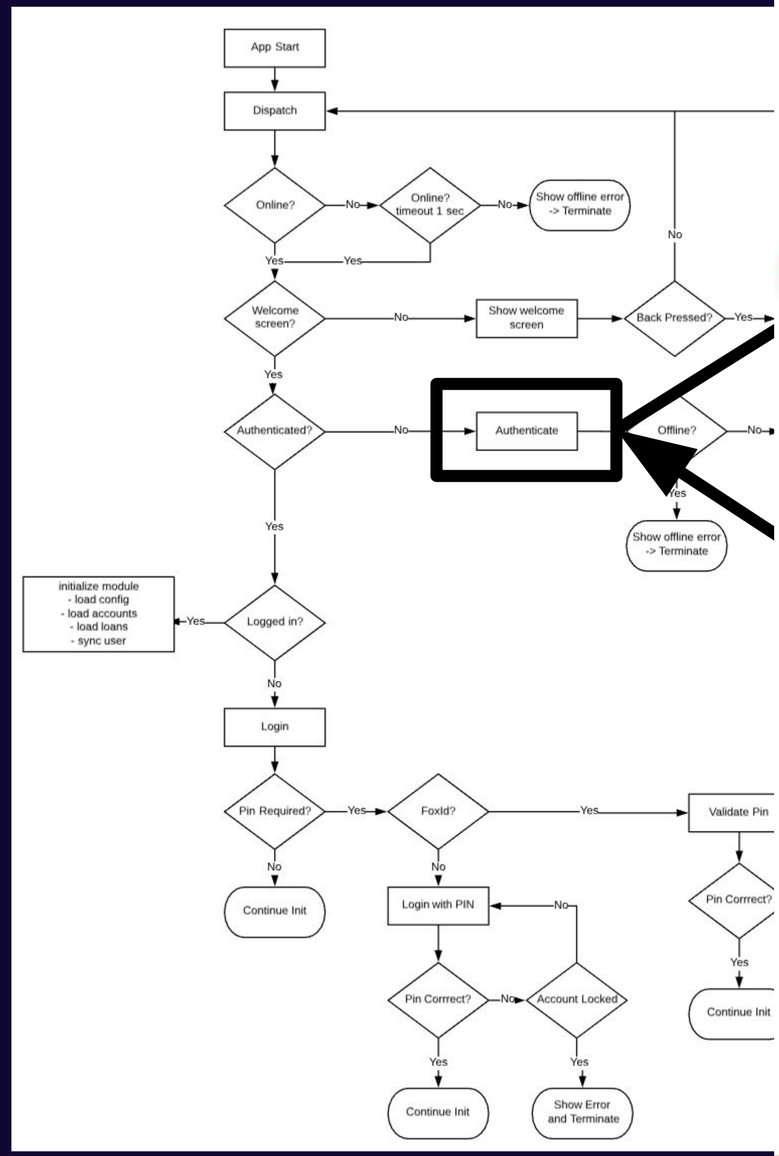

START
NO
YES
Any known credentials in Smart Lock?
No credential selected
Smart Lock credential picker
NO
YES
Auto sign-in credential retrieved?
NO
YES
Multiple auth methods supported?
Credential selected
NO
YES
Any email hints in Smart Lock?
use Email
Authentication method picker
Sign-in user with Smart Lock credential
signed in
use IDP
Email address entry
no selection
Smart Lock email hint picker
IDP sign-in
email entered
email selected
signed in
NO
YES
Matches existing account?
NO
YES
Matches existing account for which this IDP has not been used?
Create account
NO
YES
Password authentication used before?
Account linking challenge from IDP
forgot password
signed in
Account linking challenge from password
Sign in with password
signed in
signed in
forgot password
Save account to Smart Lock
Account recovery
recovery email sent
END

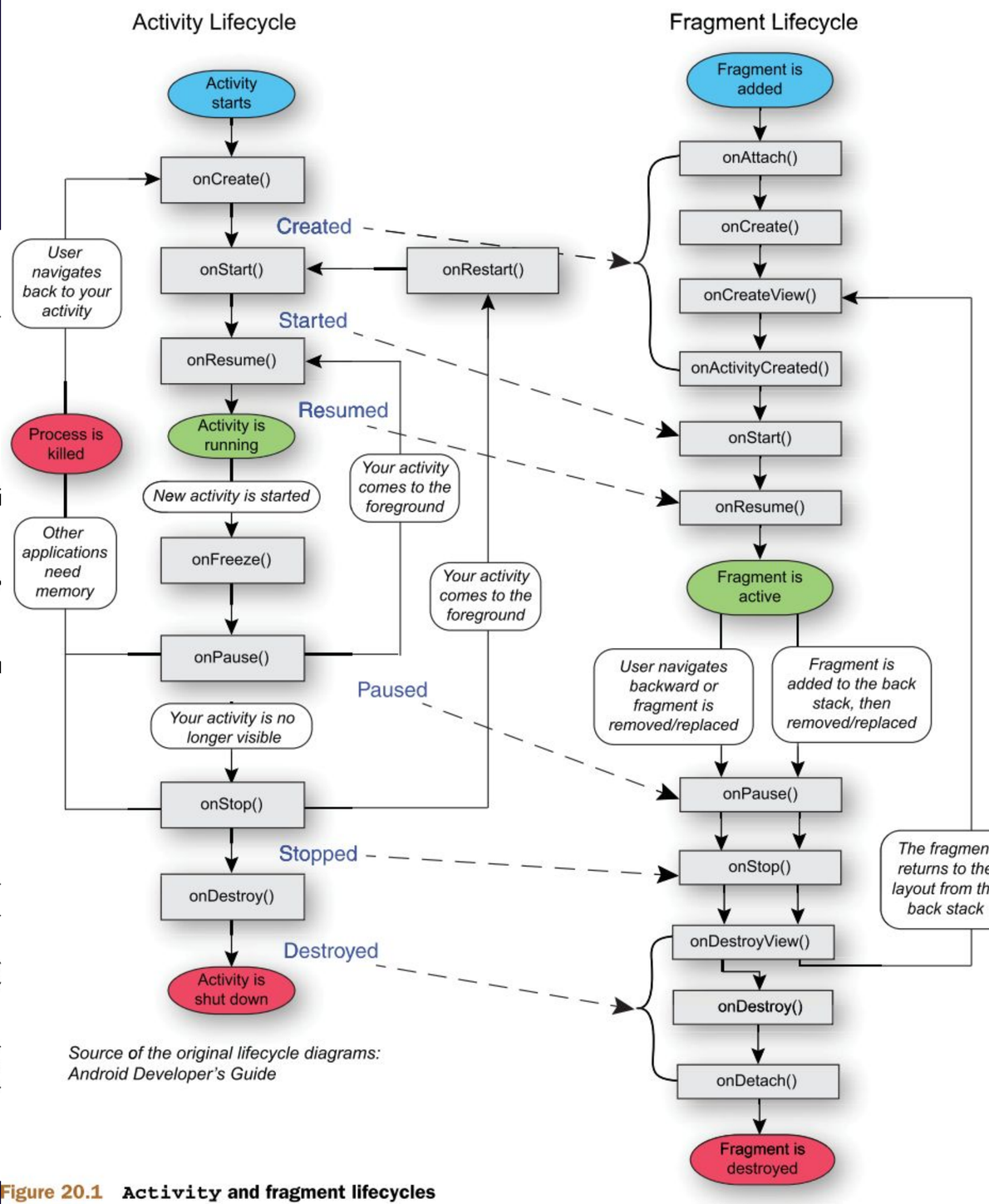
# The Problem


App Start
Dispatch
Online?
No
Online? timeout 1 sec
No
Show offline error -> Terminate
Yes
Yes
No
Welcome screen?
No
Show welcome screen
Back Pressed?
Yes
Terminate
Yes
Authenticated?
No
Authenticate
Offline?
No
Error?
No
Cancelled?
Yes
Yes
Yes
Yes
Show offline error -> Terminate
Show error -> Terminate
Terminate
Yes
initialize module
- load config
- load accounts
- load loans
- sync user
Yes
Logged in?
No
Login
Pin Required?
Yes
FoxId?
Yes
Validate Pin
No
No
No
Continue Init
Login with PIN
No
Pin Corrrect?
No
Account Locked
Yes
Yes
Yes
Pin Corrrect?
No
Account Locked
Continue Init
Show Error and Terminate
Yes
Yes
Continue Init
Show Error and Terminate


START
NO
YES
Any known credentials in Smart Lock?
No credential selected
Smart Lock credential picker
NO
YES
Auto sign-in credential retrieved?
NO
YES
Multiple auth methods supported?
Credential selected
NO
YES
Any email hints in Smart Lock?
use Email
Authentication method picker
Sign-in user with Smart Lock credential
signed in
use IDP
Email address entry
no selection
Smart Lock email hint picker
IDP sign-in
email entered
email selected
signed in
NO
YES
Matches existing account?
NO
YES
Matches existing account for which this IDP has not been used?
Create account
NO
YES
Password authentication used before?
Account linking challenge from IDP
signed in
forgot password
Account linking challenge from password
Sign in with password
signed in
signed in
forgot password
Save account to Smart Lock
Account recovery
recovery
END

# The Problem




*Source of the original lifecycle diagrams: Android Developer's Guide*

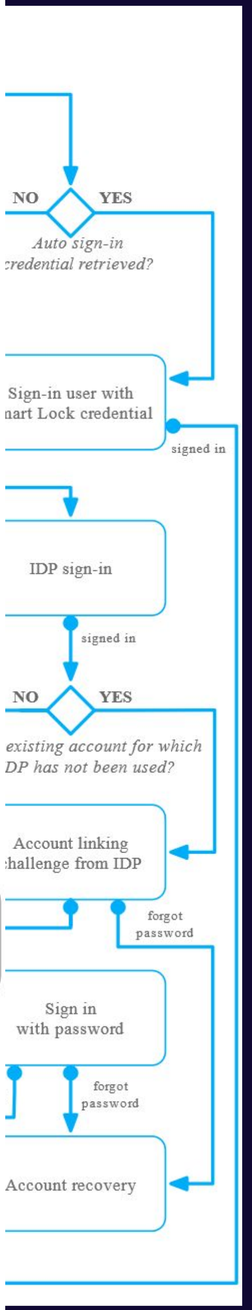



Figure 20.1 **Activity and fragment lifecycles**

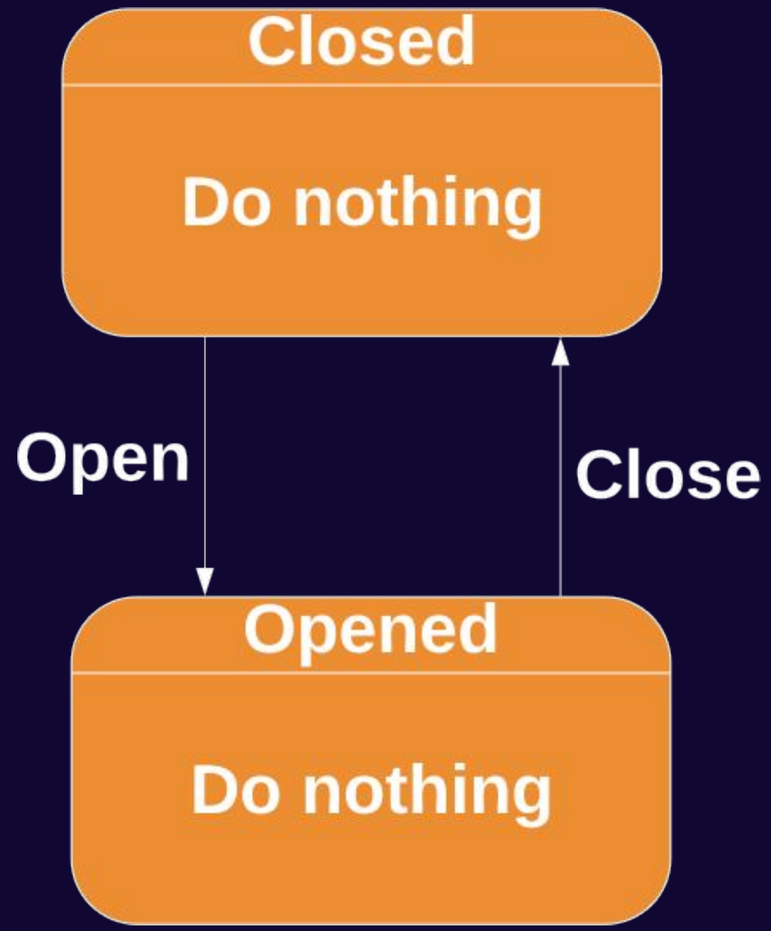
# The solution: Finite State Machine

A **finite state machine** (FSM) is an abstract machine that can be in exactly one of a finite number of **states** at any given time.
The FSM can change from one state to another in response to some **external inputs (actions)**.
The change from one state to another is called a **transition.**

Wikipedia - Finite-state machine

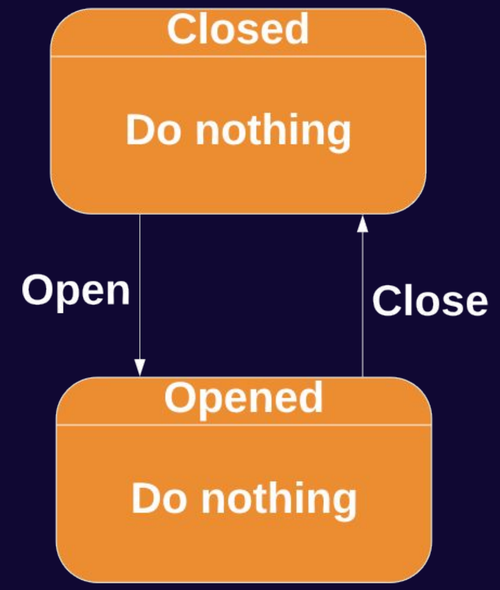
# Finite State Machine

A **finite state machine** (FSM) is an abstract machine that can be in exactly one of a finite number of **states** at any given time.
The FSM can change from one state to another in response to some **external inputs (actions)**.
The change from one state to another is called a **transition.**



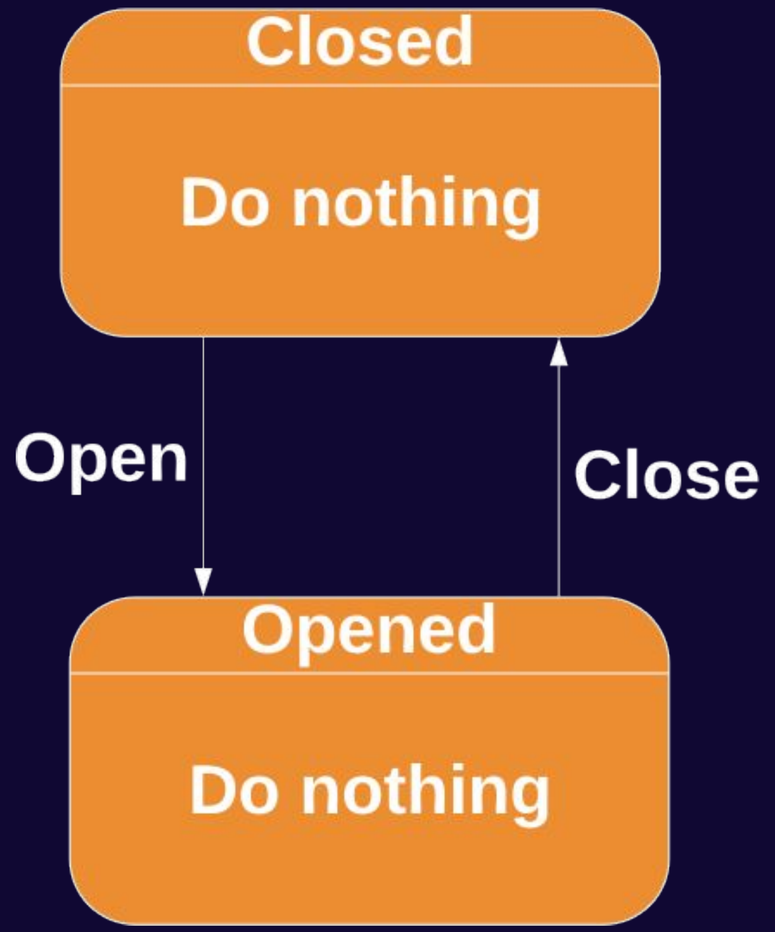

Wikipedia - Finite-state machine

# Door Finite State Machine




```
object Open : Action()
object Close : Action()
object Opened : StateImpl() {
    override fun exit(action: Action): State = if (action is Close) Closed.enter(this, action) else this
}
object Closed : StateImpl() {
    override fun exit(action: Action): State = if (action is Open) Opened.enter(this, action) else this
}
class Door @Inject constructor(initialState: State) : StateMachineImpl(initialState)
```

# Door Finite State Machine

# Finite State Machine design

**StateMachine**

- transition(action: Action): Observable<State>
- getState(): State
- getStates(): Observable<State>

**State**

- enter(previous: State, action: Action): State
- exit(action: Action): State

**Action**

**Actor(s)**

- Listen to state transitions and acts accordingly
- Triggers state transitions

# Finite State Machine design

- States decide on state transitions
    - no state machine God object for state and transition
    - decoupling of states and state machine
    - no dependency injection (circular references)

- Expose states / transitions as Rx Observables

# Finite State Machine design

**StateMachine**

- transition(action: Action): Observable<State>
- getState(): State
- getStates(): Observable<State>

**State**

- enter(previous: State, action: Action): State
- exit(action: Action): State

**Action**

**Actor(s)**

- Listen to state transitions and acts accordingly
- Triggers state transitions

vs.

```
val fsmGraph = TinderStateMachine.createGraph<State, Event, SideEffect> {
  initialState(State.Solid)
  state<State.Solid> {
    on<Event.OnMelted> {
      transitionTo(State.Liquid, SideEffect.LogMelted)
    }
  }
  state<State.Liquid> {
    on<Event.OnFroze> {
      transitionTo(State.Solid, SideEffect.LogFrozen)
    }
  }
  onTransition { transition ->
    when ((transition as? TinderStateMachine.Transition.Valid)?.sideEffect) {
      SideEffect.LogMelted -> { /* do something */ }
      SideEffect.LogFrozen -> { /* do something */ }
      else -> throw IllegalStateException("WRONG TRANSITION")
    }
  }
}
```

# Finite State Machine Code

```
interface StateMachine {

    fun getState(): State

    fun getStates(): Observable<State>

    fun transition(action: Action): Observable<State>

}
```

# Finite State Machine Code

```
abstract class StateMachineImpl(startState: State) : StateMachine {
  private val states = BehaviorSubject.create<State>()
  private var theState by Delegates.observable(startState) { _, oldState, newState ->
    if (oldState != newState) states.onNext(newState)
  }
  override fun getState() = theState
  override fun getStates() = states
  override fun transition(action: Action): Observable<State> {
    theState = theState.exit(action)
    return states
  }
}
```

# Finite State Machine Code

```
interface State {

    fun enter(previous: State, action: Action) : State

    fun exit(action: Action): State

}
```

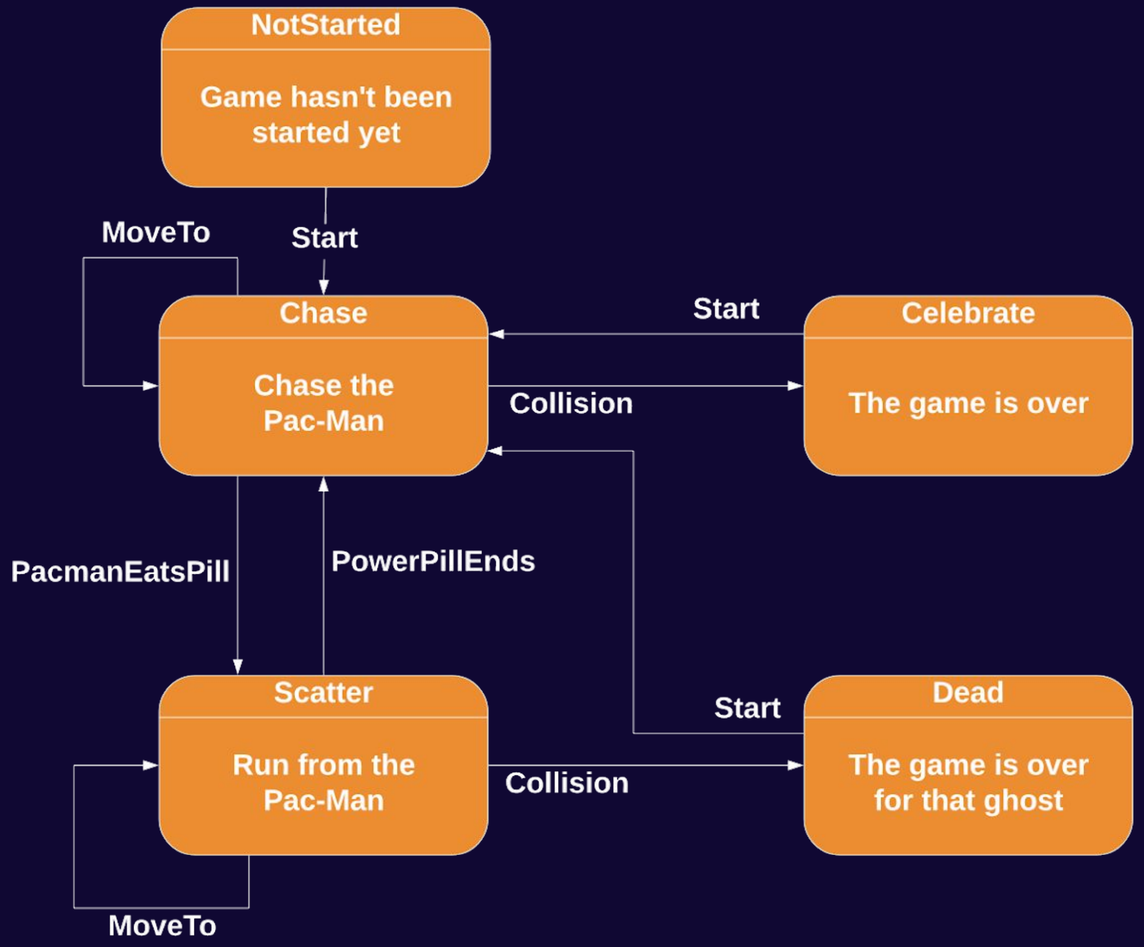
# Finite State Machine Code

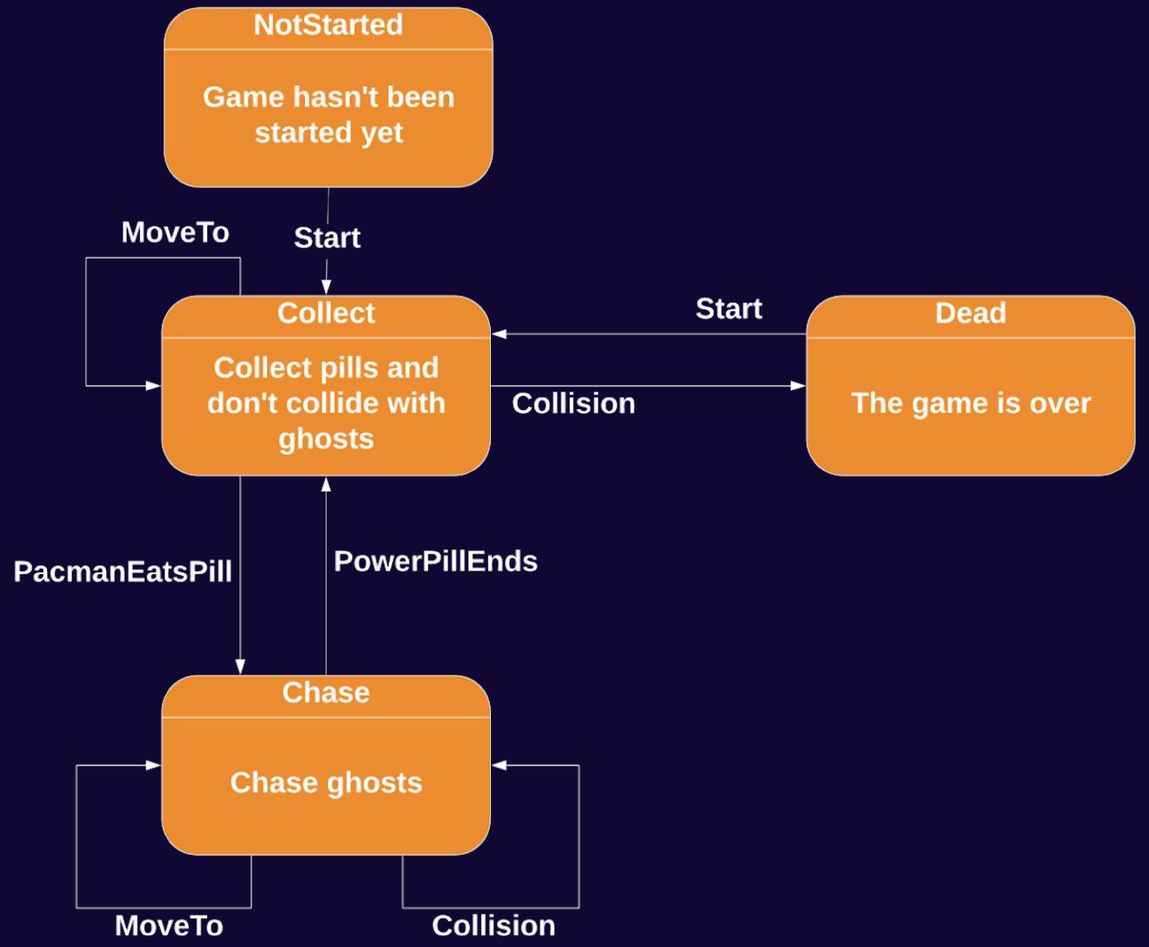
```
abstract class StateImpl: State {

    override fun enter(previous: State, action: Action) : State {
        return this
    }

    override fun exit(action: Action): State {
        return this
    }

}
```

# Finite State Machine Code

```
open class Action {


    override fun toString(): String {
        return javaClass.simpleName
    }


}
```

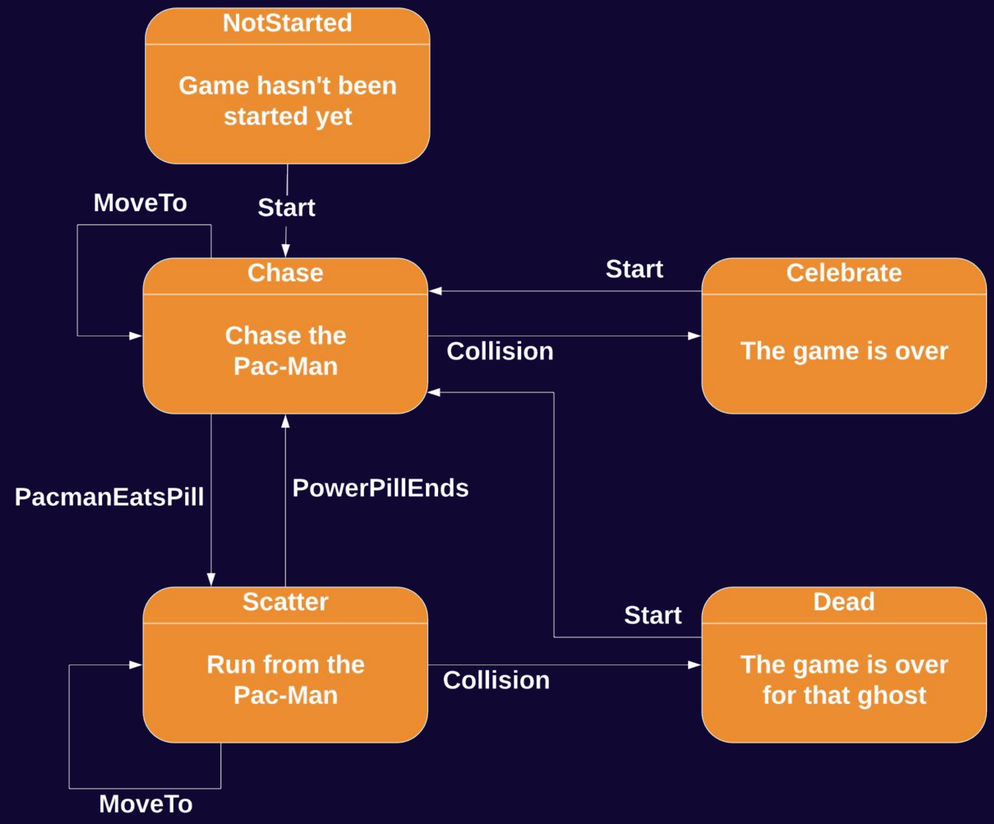
# Pacman Finite State Machine



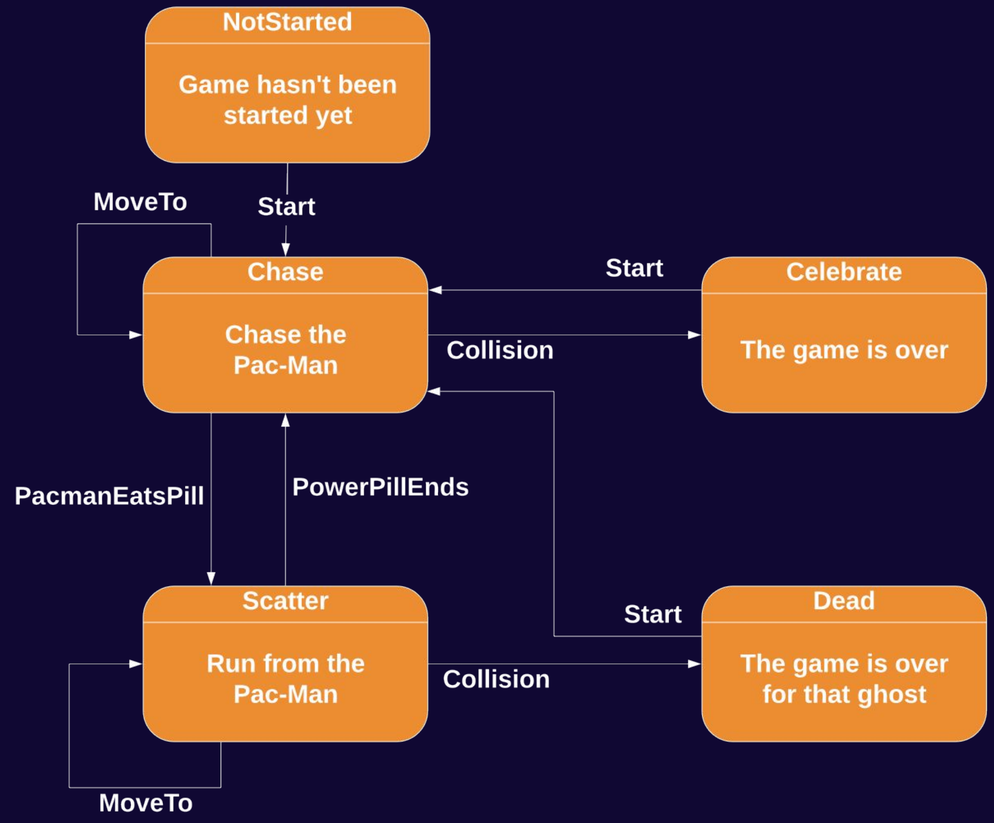



# Pacman Finite State Machine

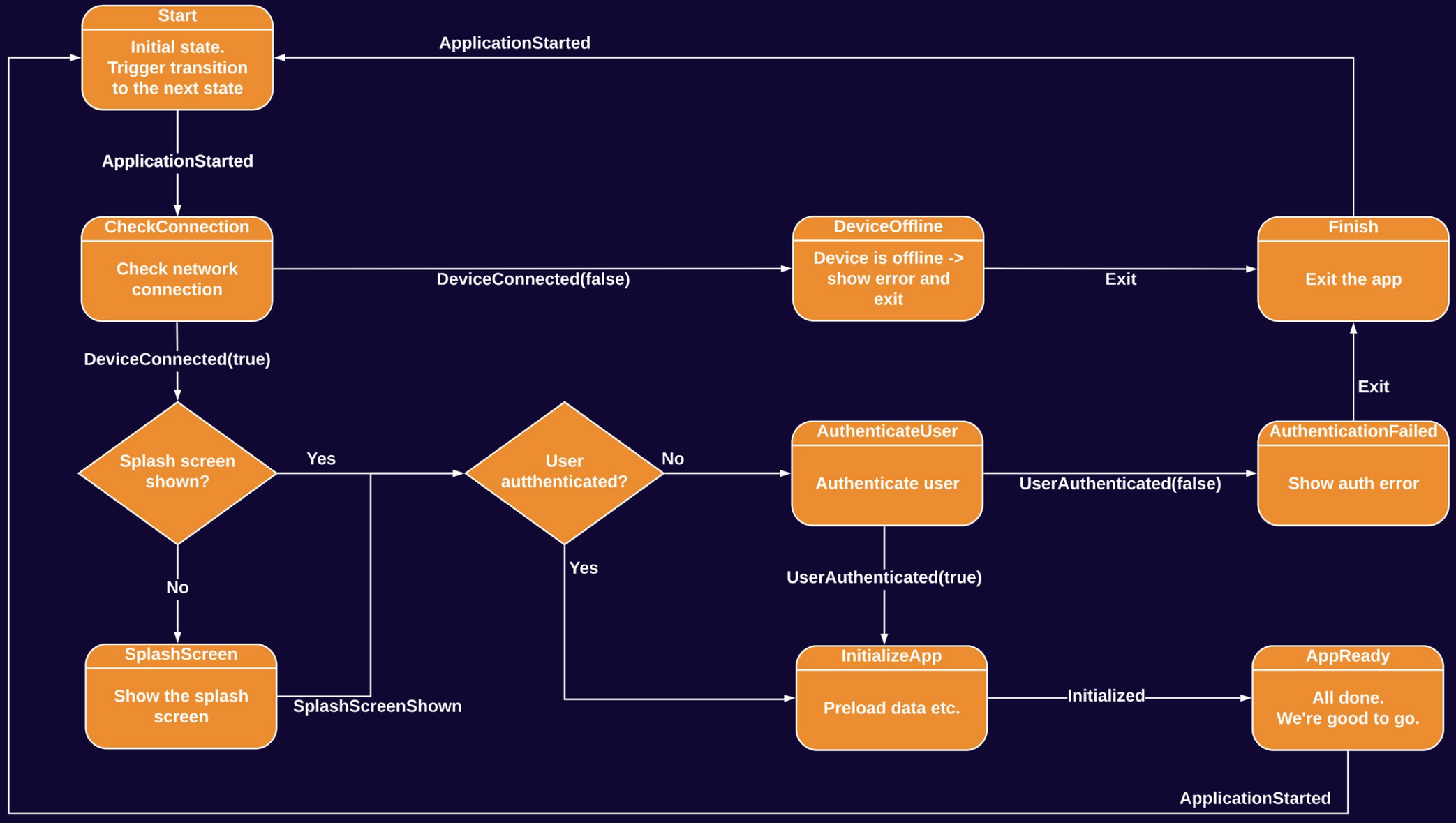
```
class Chase(val pos: Position) : State {


    override fun exit(action: Action): State {
        return when (action) {
            is MoveTo -> Chase(action.pos).enter(this, action)
            is PacmanEatsPill -> Scatter(pos).enter(this, action)
            is Collision -> Celebrate.enter(this, action)
            else -> this
        }


    }
```



# Pacman Finite State Machine

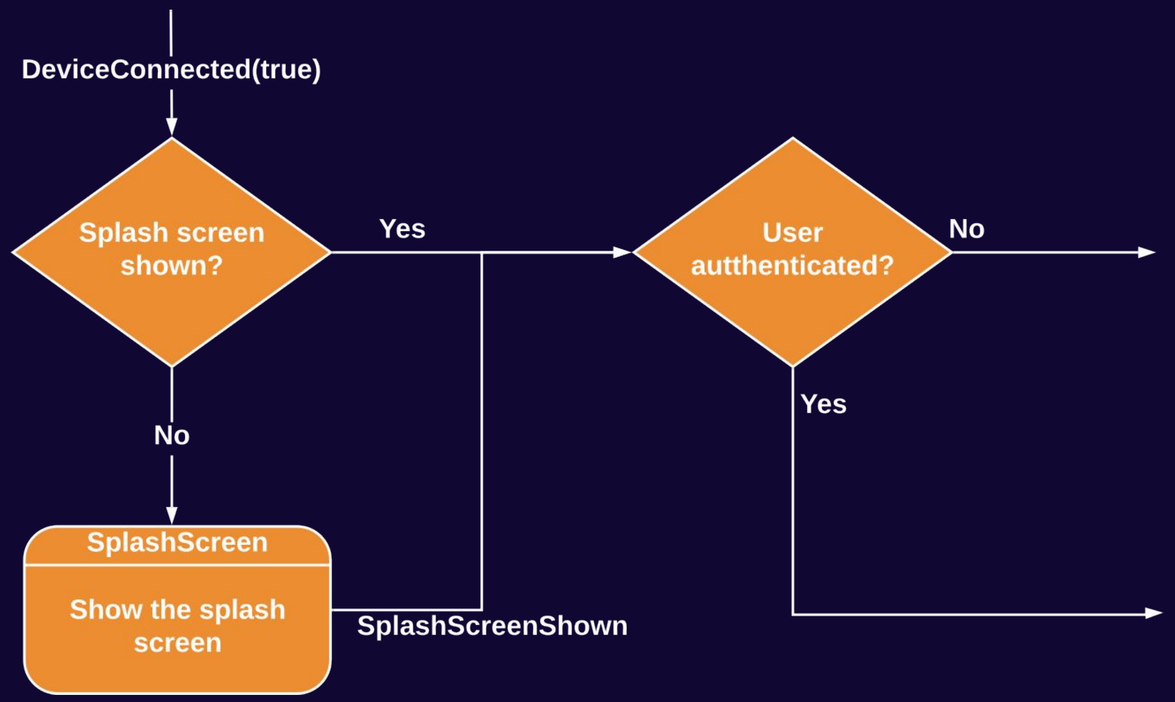



# Android Finite State Machine



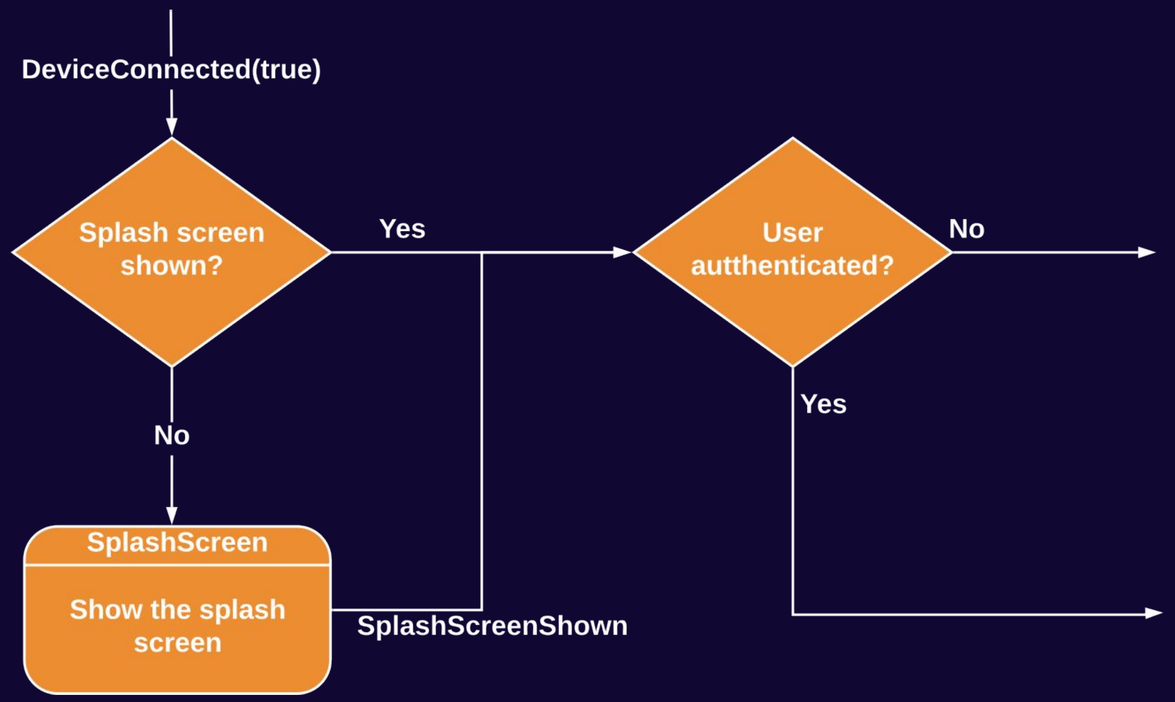
# Android Finite State Machine

```
object ShowSplashScreen : StateImpl() {


  override fun enter(previous: State, action: Action): State {
      return when (InitModel.isSplashScreenShown()) {
          true -> exit(SplashScreenShown(true))
          false -> this
      }
  }
 // more code here ...
```



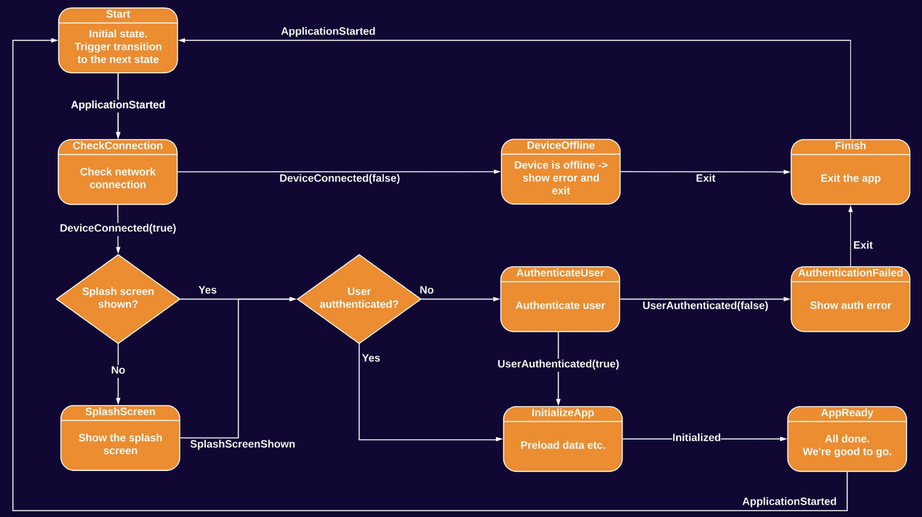
# Android Finite State Machine

```
// more code here ...
override fun exit(action: Action): State {
    return when (action) {
        is SplashScreenShown -> {
            InitModel.setSplashScreenShown()
            AuthenticateUser.enter(this, action)
        }
        else -> super.exit(action)
    }
}

}
```



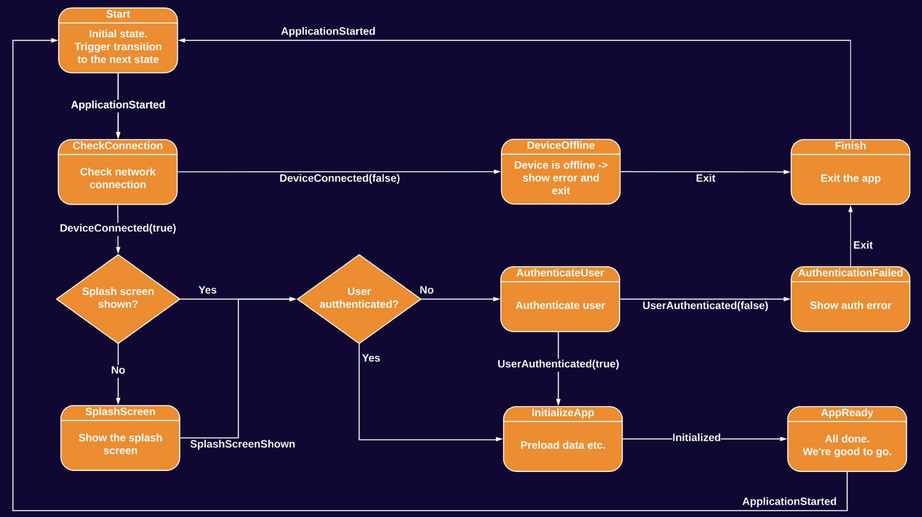
# Android Finite State Machine

```
class AndroidActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        stateMachine.getStates()
            .distinctUntilChanged()
            .doOnSubscribe { disposables.add(it) }
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe { dispatchState(it) }

        if (savedInstanceState == null) {
            stateMachine.transition(ApplicationStarted)
        }
```



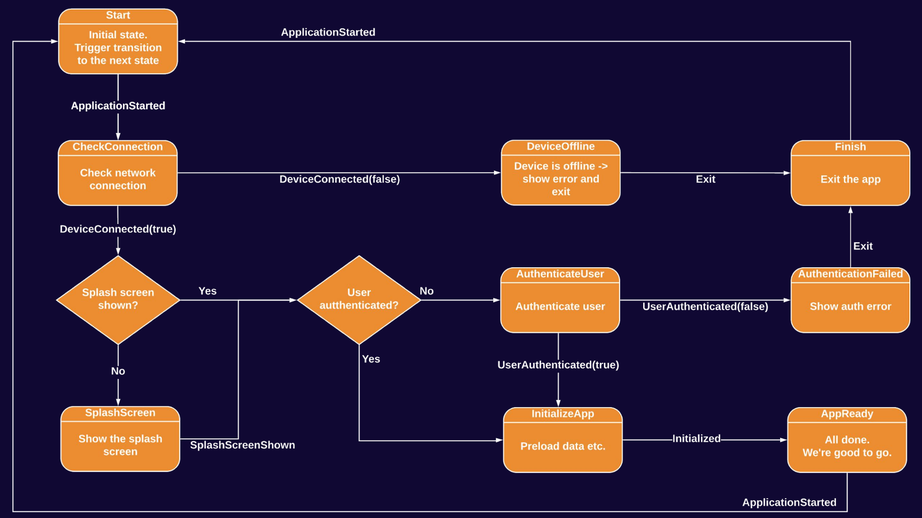
# Android Finite State Machine

```
private fun dispatchState(state: State) {
  when (state) {
      is Start -> stateMachine.transition(ApplicationStarted)
      is CheckConnection -> wait4Connection()
      is DeviceOffline -> deviceIsOffline()
      is ShowSplashScreen -> showSplashScreen()
      is AuthenticateUser -> startAuthentication()
      is AuthenticationFailed -> showAuthError()
      is InitializeApp -> initializeApp()
      is AppReady -> appIsReady()
      is Finish -> finish()
  }
```



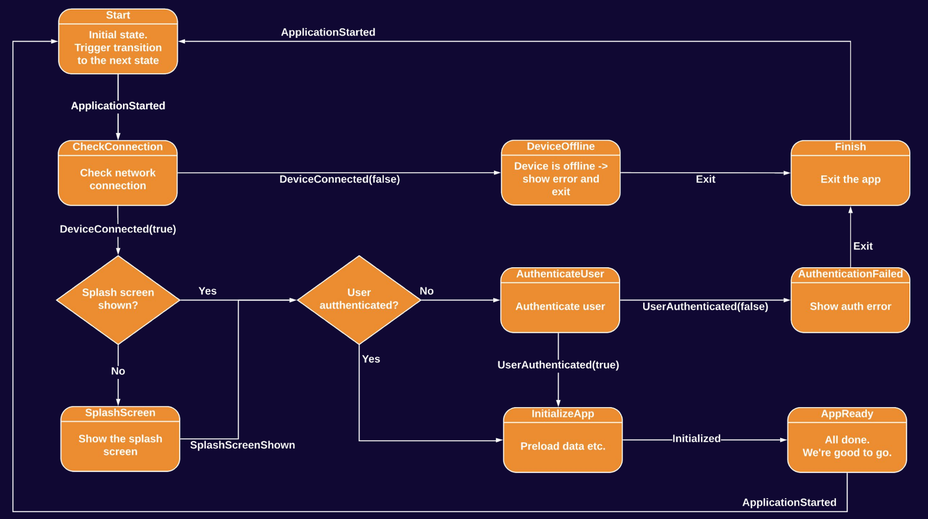
# Android Finite State Machine

```
private fun wait4Connection() {
    NetworkState.online()
        .subscribeOn(Schedulers.computation())
        .timeout(5, TimeUnit.SECONDS)
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe( {
            stateMachine.transition(DeviceConnected(true))
        }, {
            stateMachine.transition(DeviceConnected(false))
        } )
}
```

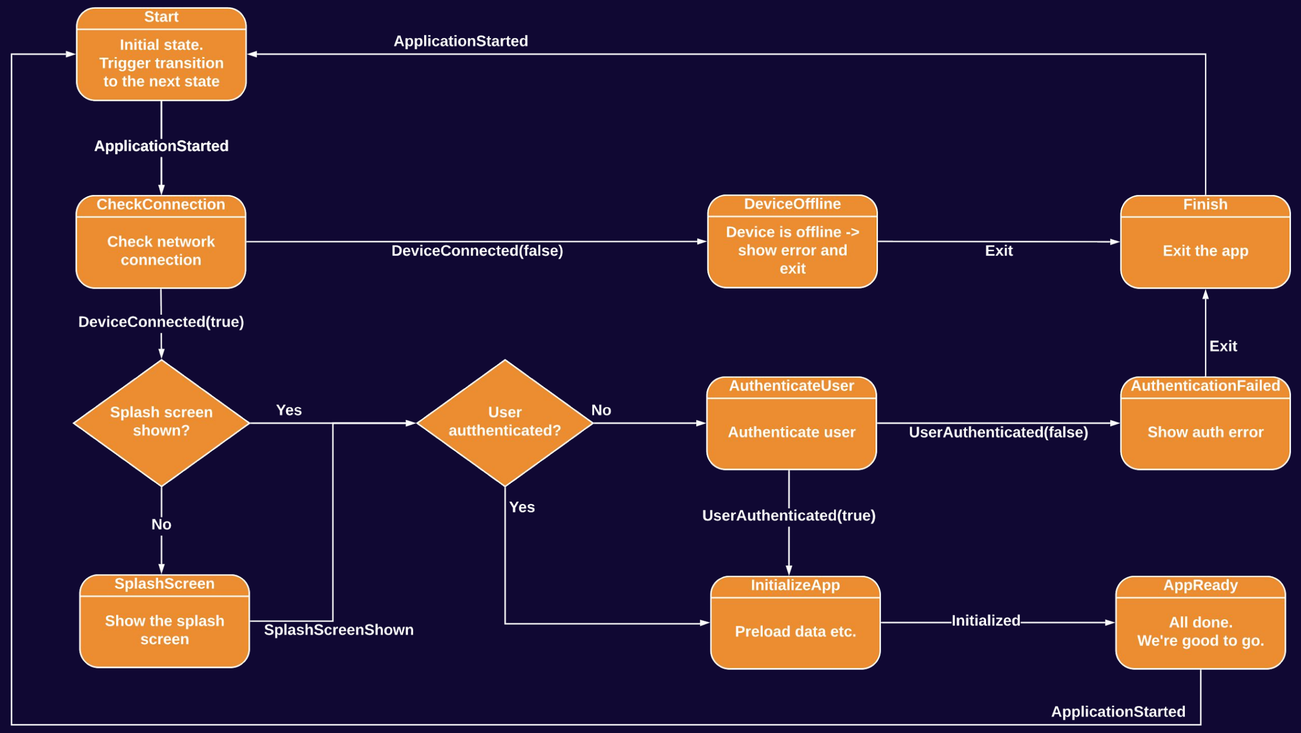


# Android Finite State Machine

```
private fun showSplashScreen() {
    showAndHide(false, true, false, true)
    tv_title.text = getString(R.string.splash_screen_title)
    tv_subtitle.text = getString(R.string.splash_screen_msg)
    btn_continue.setOnClickListener {
        stateMachine.transition(SplashScreenShown(true))
    }
}
```



# Android Finite State Machine

# Best practices

1. Don’t use for navigation only
2. Use for flows with different types of activities:

   ui, asynchronous calls, different life cycles...

3. **Draw the state diagram before writing code**

# Benefits

1. Clear separation of concerns
   - state & state transitions
   - actors
2. Reduced complexity
   → easier to understand
   → easier to maintain
   → easier to scale the team

# Benefits

3. Eliminates edge cases
    → less error prone
    → fewer regressions
4. Easier to change (change is inevitable)
    → easy to insert states
    → easy to remove states
    → easy to re-order states

# Sources

**Kotlin FSM with demo code**
https://github.com/1gravity/FiniteStateMachine
**F**uture: create loosely coupled FSM with fluent API (Kotlin DSL), add dependency injection

**Kotlin DSL with fluent API**
https://github.com/Tinder/StateMachine

**Kotlin FSM with fluent API**
https://github.com/ToxicBakery/kfin-state-machine

**Kotlin FSM**
https://thoughtbot.com/blog/finite-state-machines-android-kotlin-good-times


Q & A
Thank you!