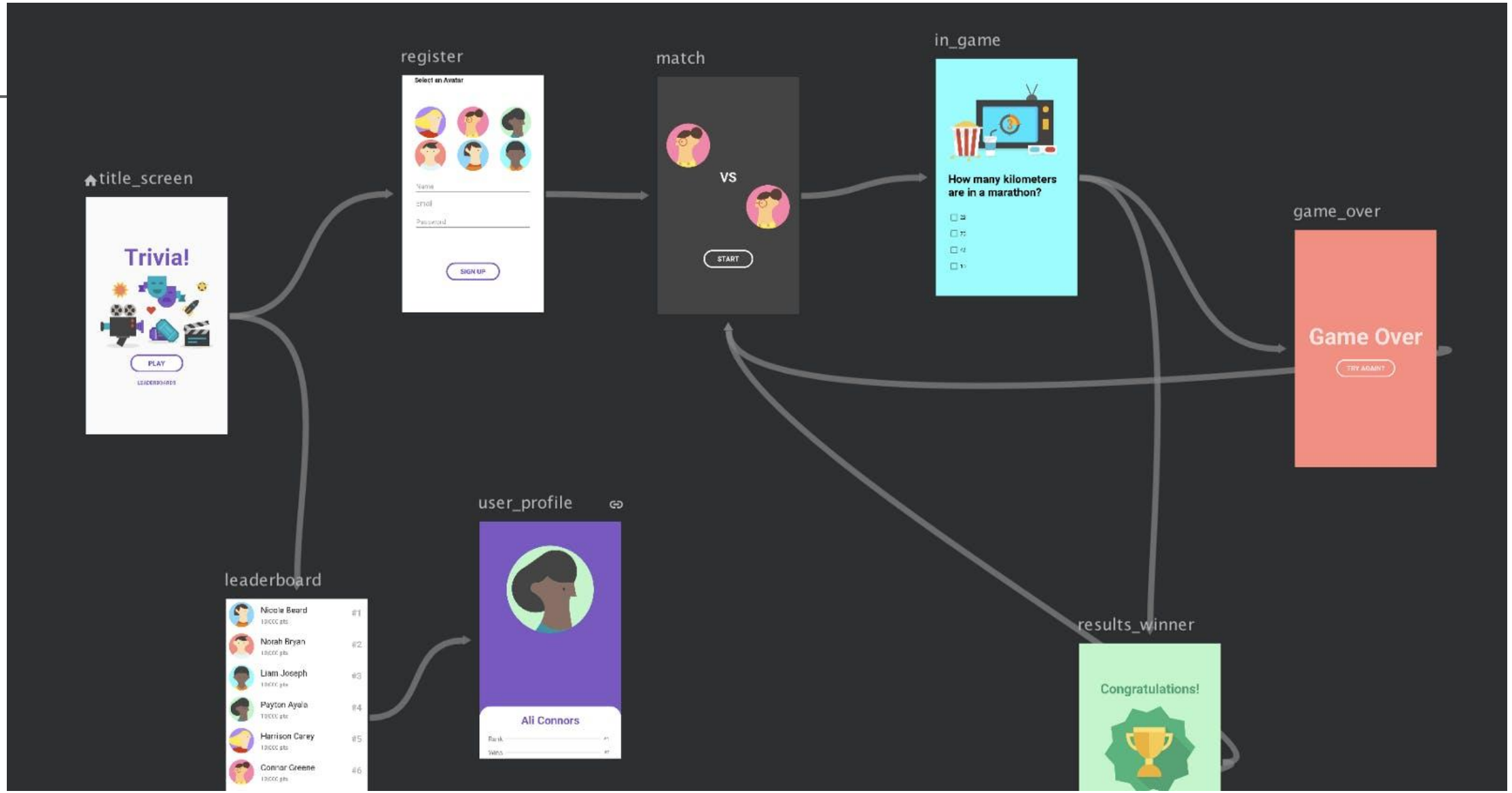
# Navigation Component

Lezione 7


title_screen
Trivia!
PLAY
register
Select an Avatar
SIGN UP
match
VS
START
in_game
How many kilometers are in a marathon?
game_over
Game Over
leaderboard
Nicole Beard
Norah Bryan
Liam Joseph
Payton Ayala
Harrison Carey
Connor Greene
user_profile
Ali Connors
results_winner
Congratulations!

# Navigation

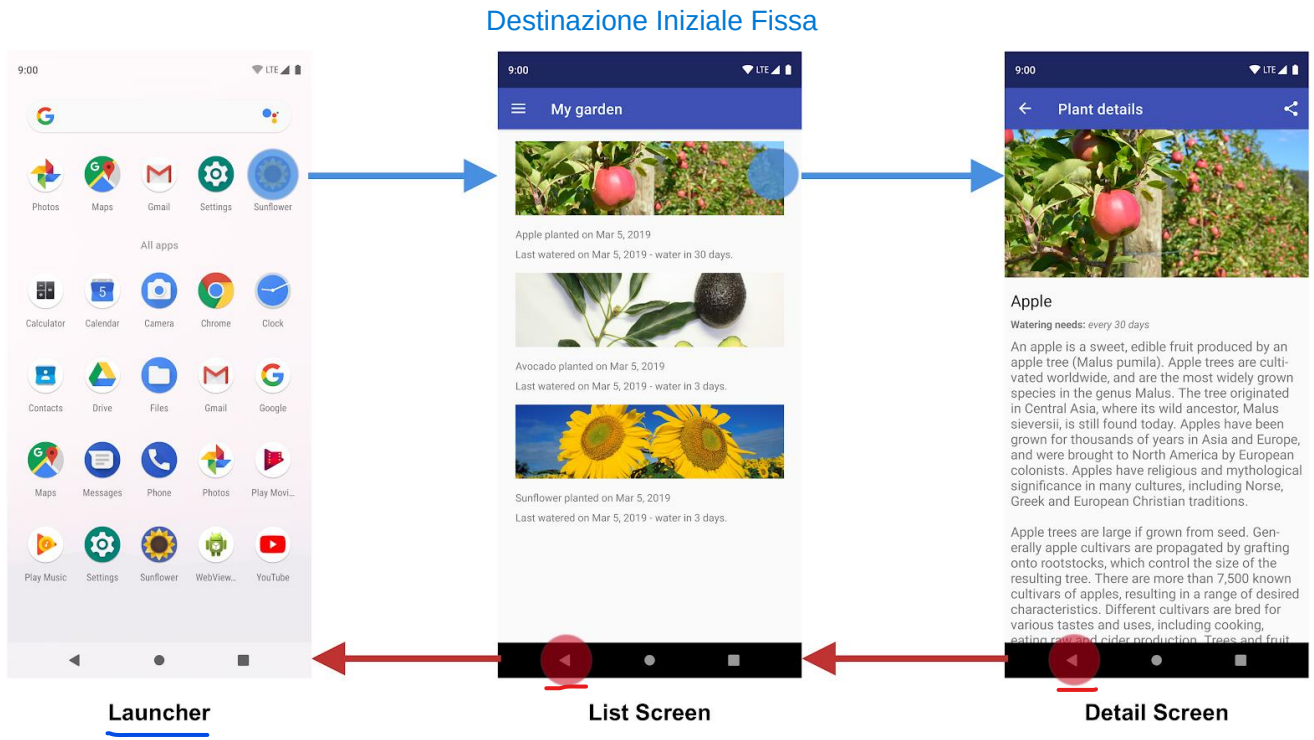
- *La navigazione* fa riferimento alle interazioni che consentono agli utenti di spostarsi, entrare e uscire dai diversi contenuti all'interno dell'app
- Il componente di navigazione (Navigation component) di Android Jetpack ti aiuta a implementare la navigazione, dai semplici clic sui pulsanti a schemi più complessi, come le barre delle app e il cassetto di navigazione
- Il componente Navigation garantisce inoltre un'esperienza utente coerente e prevedibile aderendo a una serie di principi di navigazione ben definiti

# Principi di navigazione

- La navigazione tra diverse schermate e app è una parte fondamentale dell'esperienza dell'utente
- Il navigation component è progettato per implementare questi principi per impostazione predefinita, garantendo che gli utenti possano applicare gli stessi criteri euristici e gli stessi modelli nella navigazione mentre si spostano tra le app
- I principi presentati nelle prossime slide definiscono una linea di base per un'esperienza utente coerente e intuitiva tra le app

# 1. Destinazione di partenza fissa

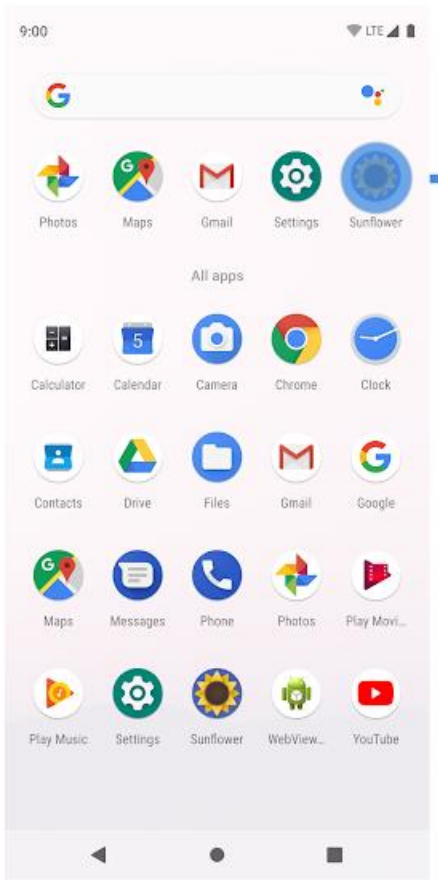
- Ogni app che si crea ha una destinazione iniziale fissa
  - Questa è la prima schermata che l'utente vede quando avvia la tua app dal programma di avvio
  - Questa destinazione è anche l'ultima schermata che l'utente vede quando torna al programma di avvio dopo aver premuto il pulsante Indietro



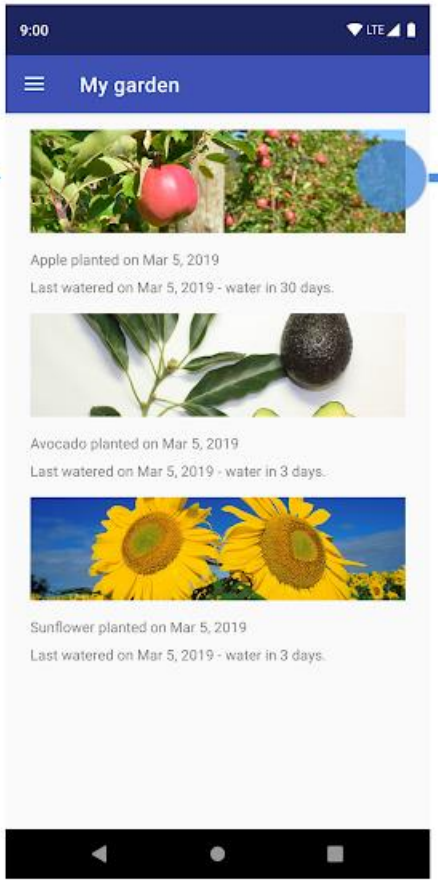
# 2. Lo stato di navigazione è rappresentato come una pila di destinazioni

- Quando l'app viene avviata per la prima volta, viene creata una nuova attività per l'utente e l'app visualizza la sua destinazione iniziale
- Questa diventa la destinazione di base di ciò che è noto come back stack ed è la base per lo stato di navigazione della tua app
  - La parte superiore dello stack è la schermata corrente e le destinazioni precedenti nello stack rappresentano la cronologia delle schermate in cui sei stato
  - Il back stack ha sempre la destinazione iniziale dell'app nella parte inferiore dello stack
  - Le operazioni che modificano il back stack operano sempre in cima allo stack, spingendo una nuova destinazione in cima allo stack o estraendo la destinazione più in alto dallo stack
  - La navigazione verso una destinazione spinge tale destinazione in cima allo stack
- Il componente Navigation **gestisce per te** tutti gli ordini del tuo back stack, sebbene tu possa anche scegliere di gestire tu stesso il back stack

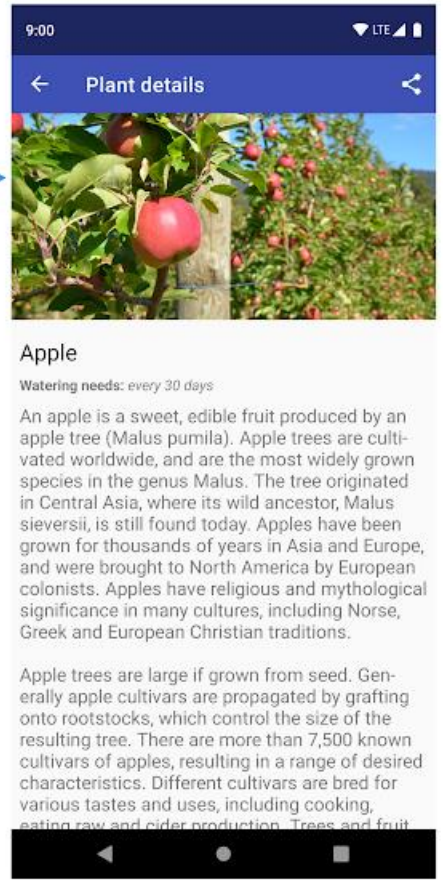
## Organic Navigation through Sunflower (Browsing to apple details)




Launch Screen

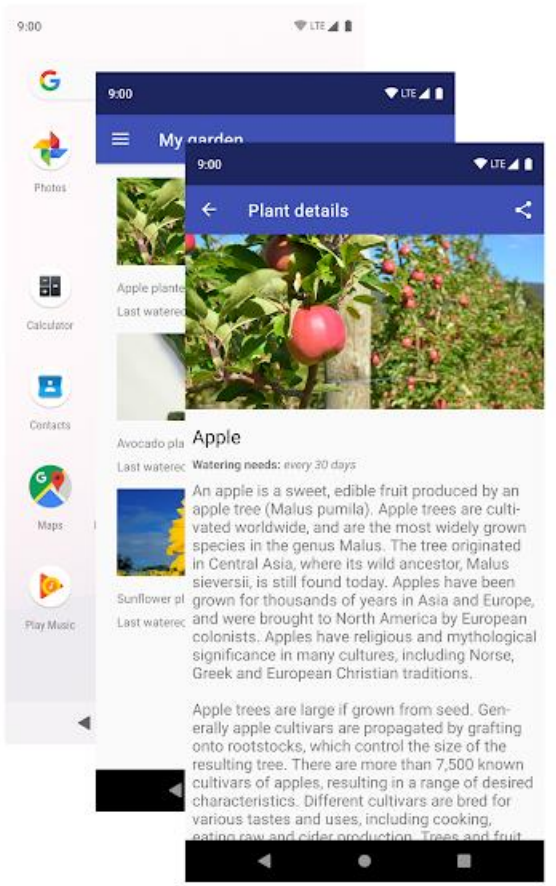



List Screen




Detail Screen

## Resulting Sunflower Task Back Stack



# 3. Up e back sono identici all'interno dell'attività della tua app

- Il pulsante Back viene visualizzato nella barra di navigazione del sistema nella parte inferiore dello schermo e viene utilizzato per navigare in ordine cronologico inverso attraverso la cronologia delle schermate con cui l'utente ha lavorato di recente
  - Quando si preme il pulsante Back, la destinazione corrente viene estratta dalla parte superiore del back stack e quindi si naviga verso la destinazione precedente
- Il pulsante Up viene visualizzato nella barra dell'app nella parte superiore dello schermo
- All'interno dell'attività dell'app, i pulsanti Up e Back si comportano in modo identico

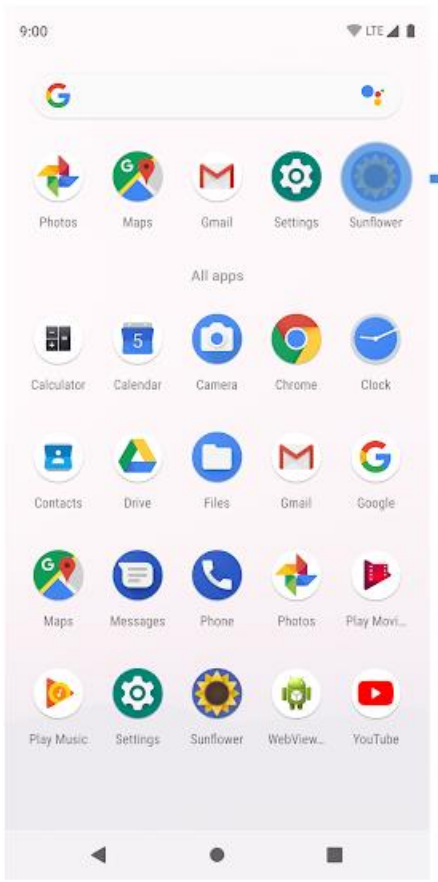
# 4. Il pulsante Up non chiude mai l'app

- Se un utente si trova nella destinazione iniziale dell'app, il pulsante Up non viene visualizzato perché il pulsante UP non permette mai di uscire dall'app
  - Il pulsante Back, tuttavia, viene visualizzato e chiude l'app
- Quando la tua app viene avviata utilizzando un collegamento diretto sull'activity di un'altra app, Up riporta gli utenti all'activity della tua app attraverso un back stack simulato e non all'app che ha attivato il collegamento diretto
  - Il pulsante Back, tuttavia, ti riporta all'altra app

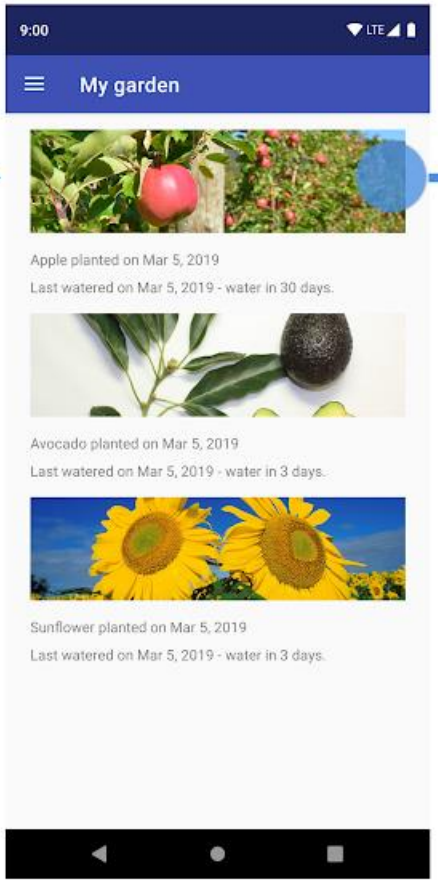
# 5. Il deep linking simula la navigazione manuale

- Che si tratti di deep linking (collegamento – URL - che porta direttamente a una destinazione specifica- activity - all'interno di un'app) o navigazione manuale verso una destinazione specifica, si può utilizzare il pulsante Up per navigare tra le destinazioni fino alla destinazione di partenza
- Quando si crea un collegamento diretto (deep linking) a una destinazione all'interno dell'attività dell'app, qualsiasi back stack esistente per l'attività dell'app viene rimosso e sostituito con lo stack back collegato in modo diretto

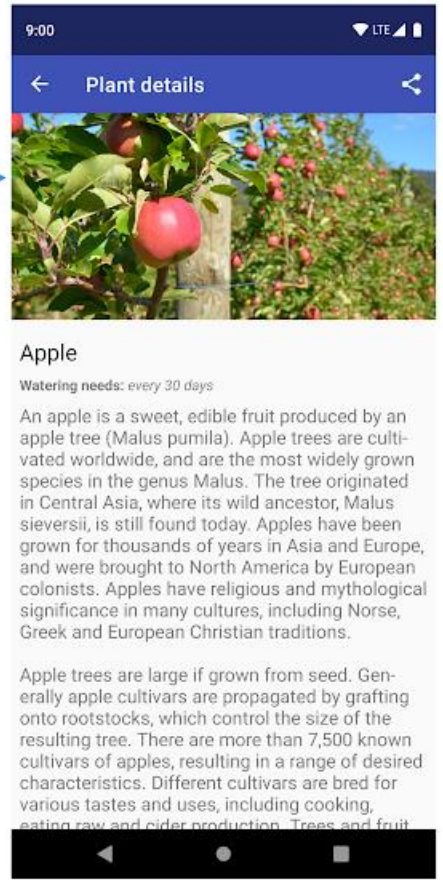
## Organic Navigation through Sunflower (Browsing to apple details)




Launch Screen

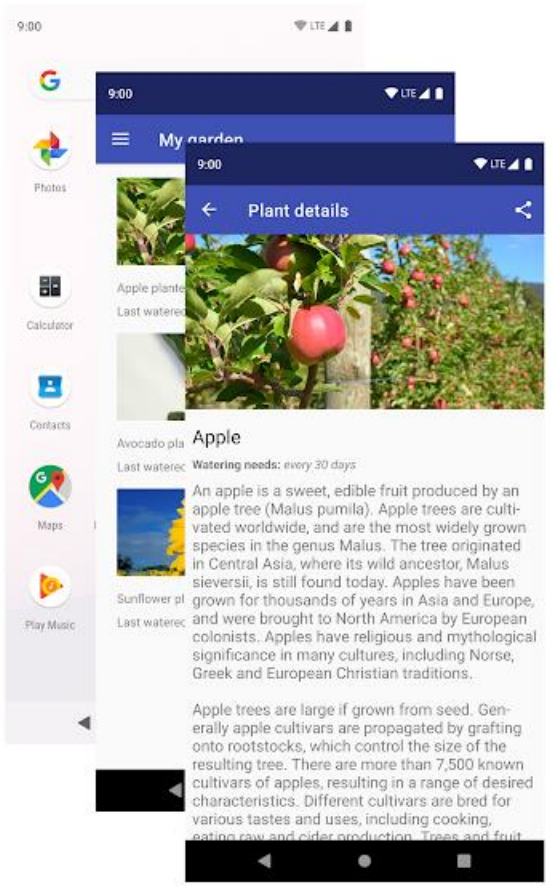



List Screen

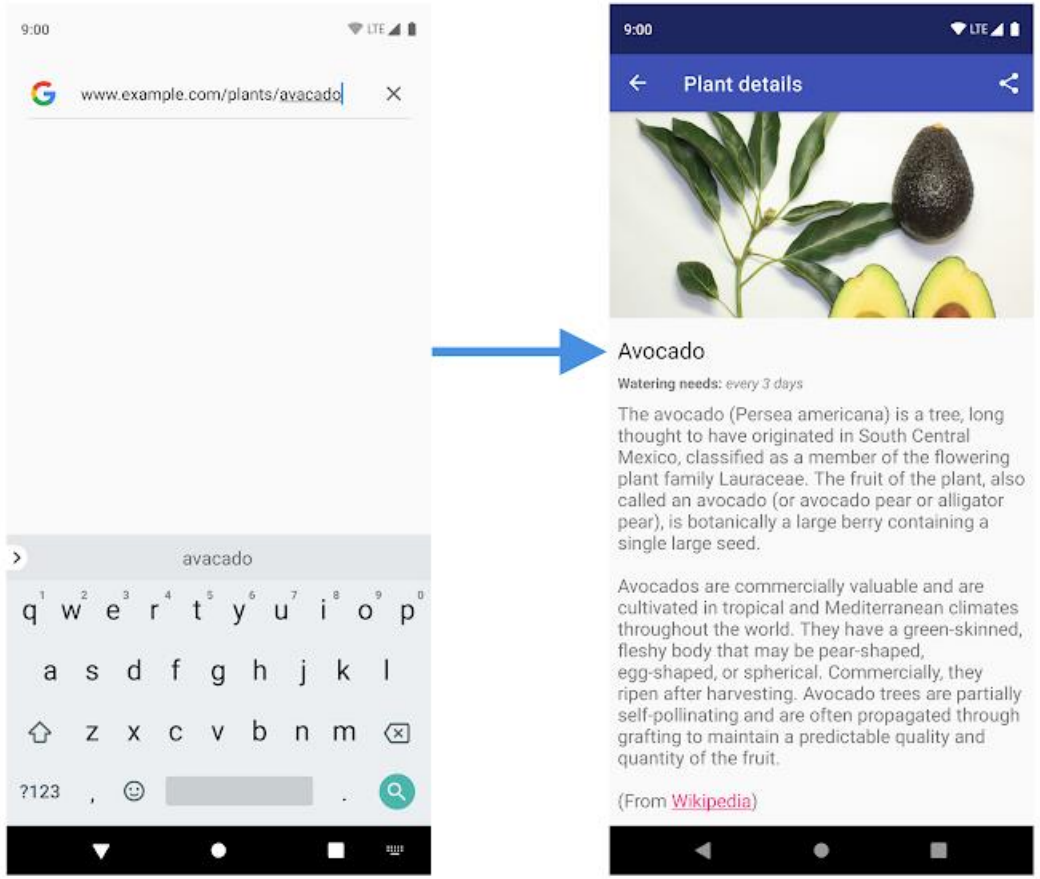



Detail Screen

## Resulting Sunflower Task Back Stack



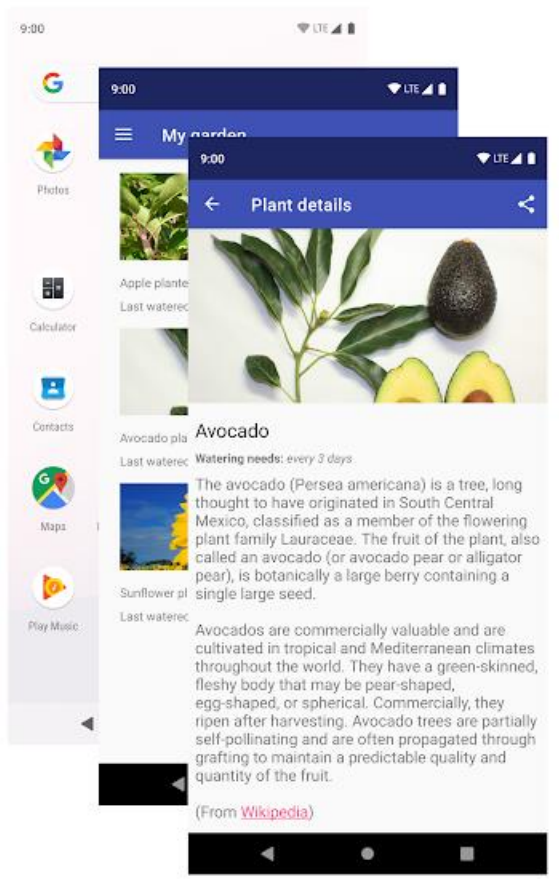
## Deep Link into Plant Details (Linking to avacado details)




Follow Deep Link

Detail Screen

## Resulting Sunflower Task Back Stack (after user enters through deep link)



# Navigation component

- Il Navigation component fornisce supporto per le applicazioni Jetpack Compose
- È possibile navigare tra i composable sfruttando l'infrastruttura e le funzionalità del Navigation component

# Concetti chiave

| Concept | Purpose | Type |
| --- | --- | --- |
| Host | A UI element that contains the current navigation destination. That is, when a user navigates through an app, the app essentially swaps destinations in and out of the navigation host. | • Compose: `NavHost`<br>• Fragments: `NavHostFragment` |
| Graph | A data structure that defines all the navigation destinations within the app and how they connect together. | `NavGraph` |
| Controller | The central coordinator for managing navigation between destinations. The controller offers methods for navigating between destinations, handling deep links, managing the back stack, and more. | `NavController` |
| Destination | A node in the navigation graph. When the user navigates to this node, the host displays its content. | `NavDestination`<br>Typically created when constructing the navigation graph. |
| Route | Uniquely identifies a destination and any data required by it.<br>You can navigate using routes. Routes take you to destinations. | Any serializable data type. |

# Navigation component

- E’ composto da tre parti principali:
  - *Grafico di navigazione*: contiene tutte le informazioni relative alla navigazione
    - Ciò include tutte le singole aree di contenuto all'interno della tua app, denominate «destinazioni», nonché i possibili percorsi che un utente può intraprendere attraverso la tua app
  - NavHost: un contenitore vuoto che mostra le destinazioni del tuo grafico di navigazione NavHost è un contenitore che tiene traccia della schermata attualmente visibile e permette di passare da una schermata all'altra.
  - NavController: un oggetto che gestisce la navigazione dell'app all'interno di un NavHost
    - Il NavController orchestra lo scambio del contenuto di destinazione nel NavHost man mano che gli utenti si spostano all'interno dell'app

# Setup

- Per supportare **Compose**, definire le dipendenze nel file *build.gradle.kts* del modulo dell'applicazione

```
dependencies {

val nav_version = " 2.8.9 "
implementation("androidx.navigation:navigation-
compose:$nav_version")
}
```

# Getting started

- **NavController è l'API centrale del componente Navigation**
  - È statico e tiene traccia nel back stack delle schermate dell'applicazione e dello stato di ciascuna schermata

- È possibile creare un *NavController* utilizzando il metodo rememberNavController() nel composable:

```
val navController = rememberNavController()
```

Il State Hoisting è un concetto chiave di Jetpack Compose che implica sollevare (hoist) lo stato in alto nella gerarchia dei composable invece di mantenerlo localmente in un singolo composable.
- Per la navigazione, significa che il NavController viene creato una sola volta in un punto superiore della gerarchia e poi passato ai composable che ne hanno bisogno.
- ERRORE DA EVITARE: Creare più NavController in diversi composable porta a comportamenti imprevedibili.

# Getting started

- Si dovrebbe creare il NavController in un punto della gerarchia dei composable in cui tutti i composable che devono farvi riferimento abbiano accesso
- Questo segue i principi di state hoisting e consente di utilizzare il NavController e lo stato che fornisce tramite currentBackStackEntryAsState() come *fonte di verità (single source of truth)* per l'aggiornamento dei composable

- Posiziona il NavController in alto nella gerarchia per evitare problemi di accesso.
- Usa currentBackStackEntryAsState() per osservare lo stato della navigazione e aggiornare la UI.
- Segui il principio di State Hoisting per mantenere una Single Source of Truth (SSOT) sulla navigazione.

# Creare un NavHost

- Ogni *NavController* deve essere associato ad un singolo *NavHost composable*
  Il NavHost tiene traccia della schermata attiva e cambia il composable visualizzato in base alla navigazione.
- Il NavHost collega il NavController con un grafo di navigazione che specifica le destinazioni composable tra le quali dovresti essere in grado di navigare
  Il grafo di navigazione è una struttura che definisce tutte le schermate (composable) e le regole per navigare tra di esse.
- Durante la navigazione tra i composable, il contenuto del NavHost viene ricomposto automaticamente
  Quando passi da una schermata all'altra, Jetpack Compose ricompone automaticamente solo il contenuto dentro il NavHost, senza ridisegnare tutto.
- Ogni destinazione composable nel grafo di navigazione è associata a un percorso (route)
  - Route definisce il percorso del tuo composable che conduce a una destinazione specifica
  - Ogni destinazione DEVE avere un percorso unico

# Creare un NavHost

- La creazione del NavHost richiede il NavController precedentemente creato tramite **rememberNavController()** e il percorso della destinazione iniziale del grafo
    - Esempio in slide successiva!

# Creare un NavHost

Route

```
@Serializable
object Profile
@Serializable
object FriendsList

val navController = rememberNavController()

NavHost(navController = navController, startDestination = Profile) {
    composable<Profile> { ProfileScreen( /* ... */ ) }
    composable<FriendsList> { FriendsListScreen( /* ... */ ) }
    // Add more destinations similarly.
}
```

# Nota!

- Ricordatevi che il Navigation component richiede di seguire i principi di navigazione e utilizzare una destinazione di partenza fissa!
  - Principio 1 visto all'inizio delle slide

# Navigare verso un composable

- Per navigare verso una destinazione composable nel grafo di navigazione, è necessario utilizzare il metodo *navigate*

Route

```
@Serializable
object FriendsList

navController.navigate(route = FriendsList)
```

# Navigare verso un composable: modifica back stack

- Per impostazione predefinita, Navigation aggiunge la tua nuova destinazione al back stack
- Puoi modificare il comportamento di *navigate* aggiungendo ulteriori opzioni di navigazione alla chiamata *navigate()*
  - Per esempio popUpTo() che permette di estrarre destinazioni aggiuntive dal back stack

- popUpTo rimuove tutte le schermate fino alla route specificata.
- Se aggiungi inclusive = true, rimuove anche la route indicata.
- Se non usi saveState, le schermate rimosse vengono distrutte e perse.
- Se vuoi salvare lo stato delle schermate rimosse dal back stack per poterle ripristinare dopo, usa saveState = true.
- Se hai salvato lo stato con saveState = true, puoi ripristinarlo con restoreState = true quando navighi di nuovo verso quella destinazione.

# Esempio

Salvare lo stato di una schermata significa conservare i suoi dati e la sua UI anche se viene rimossa dal back stack. Se non salvi lo stato, ogni volta che torni a una schermata, questa verrà ricreata da zero, perdendo tutti i dati.

```
navController.navigate(
    route = route,
    navOptions =  navOptions {
        popUpTo<A>{ saveState = true }
        restoreState = true
    }
)
```

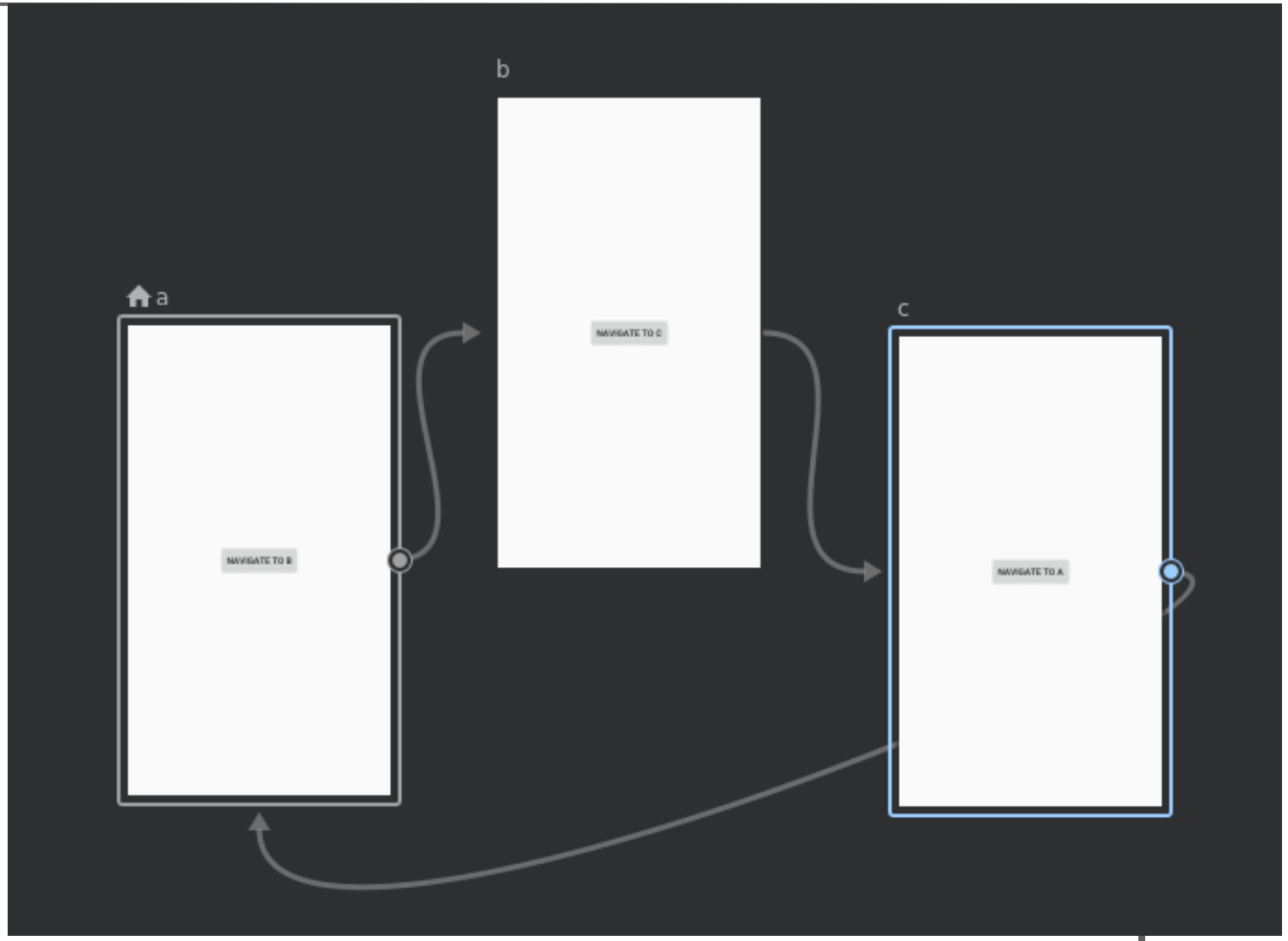
Nota: Quando utilizzi popUpTo per navigare verso una destinazione, puoi facoltativamente salvare il back stack e gli stati di tutte le destinazioni estratte dal back stack. Puoi quindi ripristinare il back stack e le destinazioni quando navighi verso quella destinazione in un secondo momento. Ciò ti consente di preservare lo stato per una determinata destinazione e di avere più back stack. Per farlo a livello di programmazione, specifica saveState = true quando aggiungi popUpTo alle tue opzioni di navigazione. Puoi anche specificare restoreState = true nelle tue opzioni di navigazione per ripristinare automaticamente il back stack e lo stato associato alla destinazione.

Il back stack è la pila di schermate visitate dall'utente. Ogni volta che navighi a una nuova schermata, questa viene aggiunta alla pila. Nel back stack di Jetpack Compose non c'è mai la schermata che stai vedendo attualmente. Il back stack contiene solo le schermate già visitate (quelle che l'utente ha navigato precedentemente) e non la schermata che è attualmente visibile.

# Approfondimento: **popUpTo**

- popUpTo permette di rimuovere dal back stack tutte le destinazioni sopra quella definita (rimosse dall'altro verso il basso)
- Con *inclusive*, incluso quella definita (nell'esempio prima Home)
- Questo permette di evitare il loop che si vede in figura
- Pensate al login..

  Se un utente accede alla Home dopo il Login, la schermata di Login rimane nel back stack. Se preme "indietro", torna a Login invece di chiudere l'app. Con popUpTo, possiamo rimuovere Login dal back stack quando l'utente entra in Home.



# Chiamate navigate() attivate da altre funzioni composable

- La funzione navigate() del NavController modifica lo stato interno del NavController
- Per conformarsi il più possibile al principio dell'unica fonte di verità, solo la funzione composable o lo state holder dell'istanza di NavController e quelle funzioni composable che accettano il NavController come parametro dovrebbero effettuare chiamate di navigation

  Le chiamate a navigate() dovrebbero essere fatte solo nelle funzioni composable o negli state holder che gestiscono direttamente l'istanza del NavController. Questo assicura che lo stato di navigazione sia gestito in modo coerente e centralizzato, evitando chiamate di navigazione dispersive.
- Gli eventi di navigazione attivati da altre funzioni composable inferiori nella gerarchia dell'interfaccia utente (es. un bottone) devono esporre tali eventi al chiamante in modo appropriato

  Se hai composable inferiori (ad esempio, pulsanti o altre interazioni) che devono attivare la navigazione, è importante esporre correttamente gli eventi di navigazione al chiamante. Ciò significa che questi eventi non dovrebbero chiamare direttamente navigate() al loro interno, ma dovrebbero inviare un "segnale" al livello superiore (dove il NavController è accessibile) per eseguire l'azione di navigazione.

```kotlin
@Serializable
object Profile
@Serializable
object FriendsList

@Composable
fun MyAppNavHost(
    modifier: Modifier = Modifier,
    navController: NavHostController = rememberNavController(),

) {
    NavHost(
        modifier = modifier,
        navController = navController,
        startDestination = Profile
    ) {
        composable<Profile> {
            ProfileScreen(
                onNavigateToFriends = { navController.navigate(route = FriendsList) },
                /*...*/
            )
        }
        composable<FriendsList> { FriendsListScreen(/*...*/) }
    }
}

@Composable
fun ProfileScreen(
    onNavigateToFriends: () -> Unit,
    /*...*/
) {
    /*...*/
    Button(onClick = onNavigateToFriends) {
        Text(text = "See friends list")
    }
}
```

- L'esempio mostra la funzione composable MyAppNavHost come singola origine attendibile (single source of truth) per l'istanza NavController
- ProfileScreen espone un evento come una funzione che viene chiamata quando l'utente interagisce con un pulsante
- MyAppNavHost, che possiede la navigazione verso le diverse schermate nell'app, effettua la chiamata di navigazione verso la destinazione corretta quando si chiama ProfileScreen

# Navigate con argomenti

- Navigation Compose supporta anche il passaggio di argomenti tra destinazioni composable

```
@Serializable
data class Profile(val name: String)

@Serializable
data class Profile(val nickname: String? = null) //opzionali
```

# Navigate con argomenti

- È necessario estrarre gli argomenti da NavBackStackEntry (rappresentazione di un entry nel back stack) disponibili nel lambda della funzione composable()

```
@Serializable
data class Profile(val name: String)

val navController = rememberNavController()

NavHost(navController = navController, startDestination = Profile(name="John Smith")) {
    composable<Profile> { backStackEntry ->
        val profile: Profile = backStackEntry.toRoute()
        ProfileScreen(name = profile.name) }
}
```

Destinazione Iniziale del Grafo

Estrazione Argomenti da BackStack

# Navigate con argomenti

- Per passare l'argomento alla destinazione, devi aggiungerlo alla rotta quando effettui la chiamata di navigazione:

```
navController.navigate( route = Profile(name = ”Gino Pino")
```

```kotlin
@Serializable
data class Profile(val name: String)

@Serializable
object FriendsList

// Define the ProfileScreen composable.
@Composable
fun ProfileScreen(
    profile: Profile
    onNavigateToFriendsList: () -> Unit,
  ) {
  Text("Profile for ${profile.name}")
  Button(onClick = { onNavigateToFriendsList() }) {
    Text("Go to Friends List")
  }
}

// Define the FriendsListScreen composable.
@Composable
fun FriendsListScreen(onNavigateToProfile: () -> Unit) {
  Text("Friends List")
  Button(onClick = { onNavigateToProfile() }) {
    Text("Go to Profile")
  }
}
```

```kotlin
// Define the MyApp composable, including the `NavController` and `NavHost`.
@Composable
fun MyApp() {
  val navController = rememberNavController()
  NavHost(navController, startDestination = Profile(name = "John Smith")) {
    composable<Profile> { backStackEntry ->
        val profile: Profile = backStackEntry.toRoute()
        ProfileScreen(
            profile = profile,
            onNavigateToFriendsList = {
                navController.navigate(route = FriendsList)
            }
        )
    }
    composable<FriendsList> {
      FriendsListScreen(
        onNavigateToProfile = {
          navController.navigate(
            route = Profile(name = "Aisha Devi")
          )
        }
      )
    }
  }
}
```

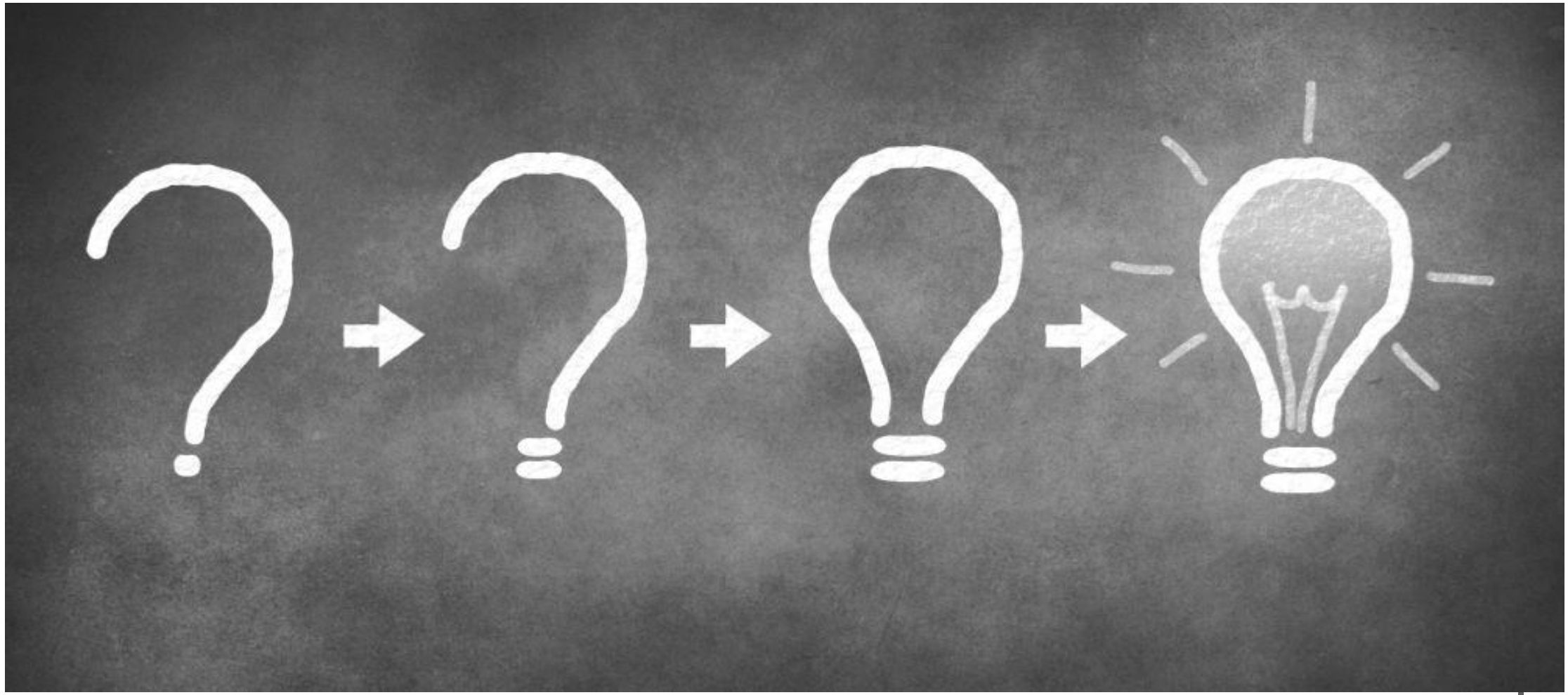
# Recupero di dati complessi durante la navigazione

- Si consiglia fortemente di non passare oggetti di dati complessi durante la navigazione, ma di passare invece le informazioni minime necessarie, come un identificatore univoco o altra forma di ID, come argomenti durante l'esecuzione delle azioni di navigazione

# Recupero di dati complessi durante la navigazione

- Gli oggetti complessi dovrebbero essere archiviati come dati in *un'unica fonte di verità*, come il livello Data
- Una volta nella destinazione dopo la navigazione, puoi quindi caricare le informazioni richieste dall'unica fonte di verità utilizzando l'ID trasmesso
- Per recuperare gli argomenti useremo il *ViewModel* (ne parleremo in modo dettagliato nella prossima lezione!) responsabile dell'accesso al livello dati
- Questo approccio aiuta a prevenire la perdita di dati durante le modifiche alla configurazione e qualsiasi incoerenza quando l'oggetto in questione viene aggiornato o modificato

# Domande?



# Riferimenti

- https://developer.android.com/jetpack/compose/navigation