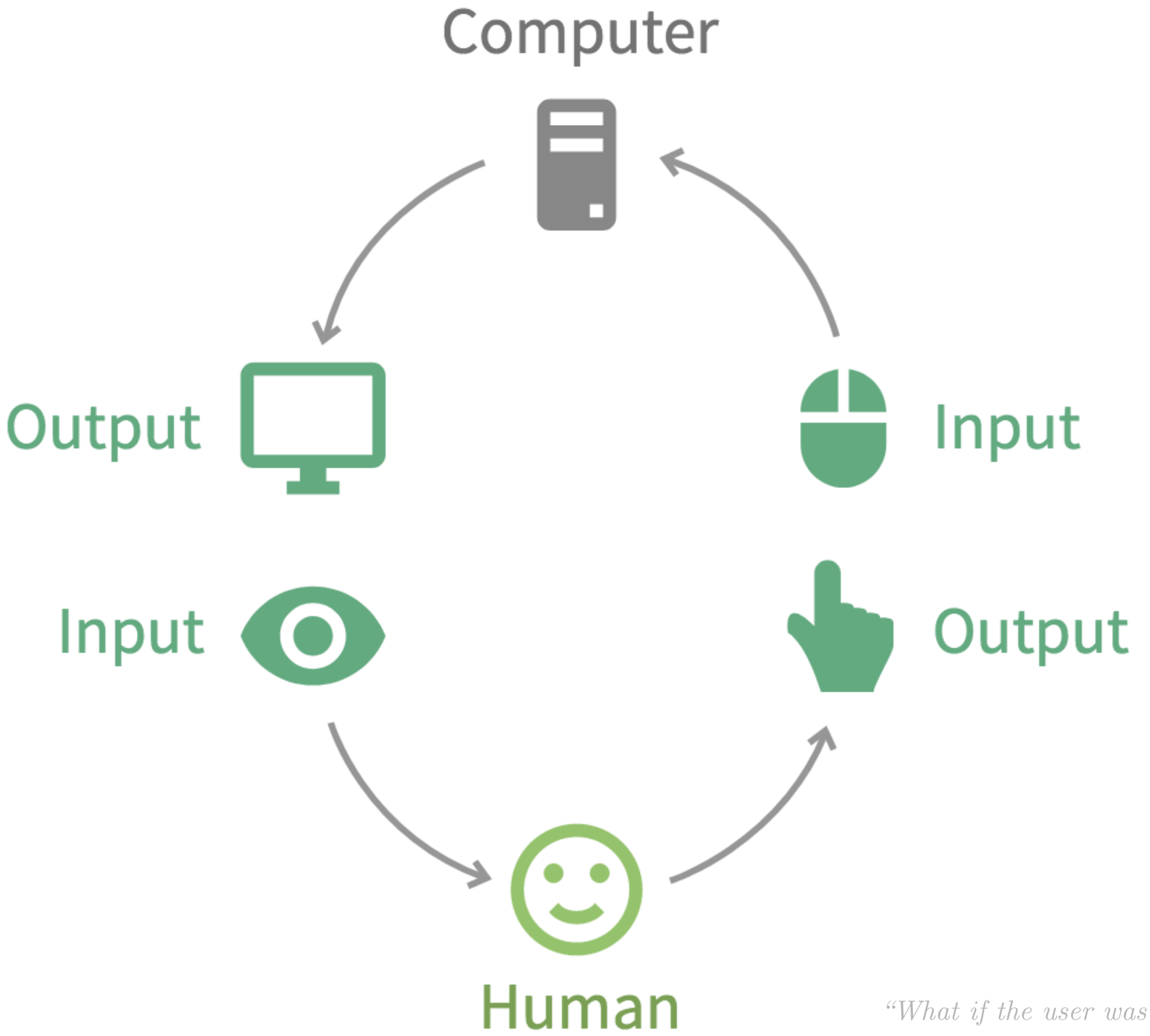
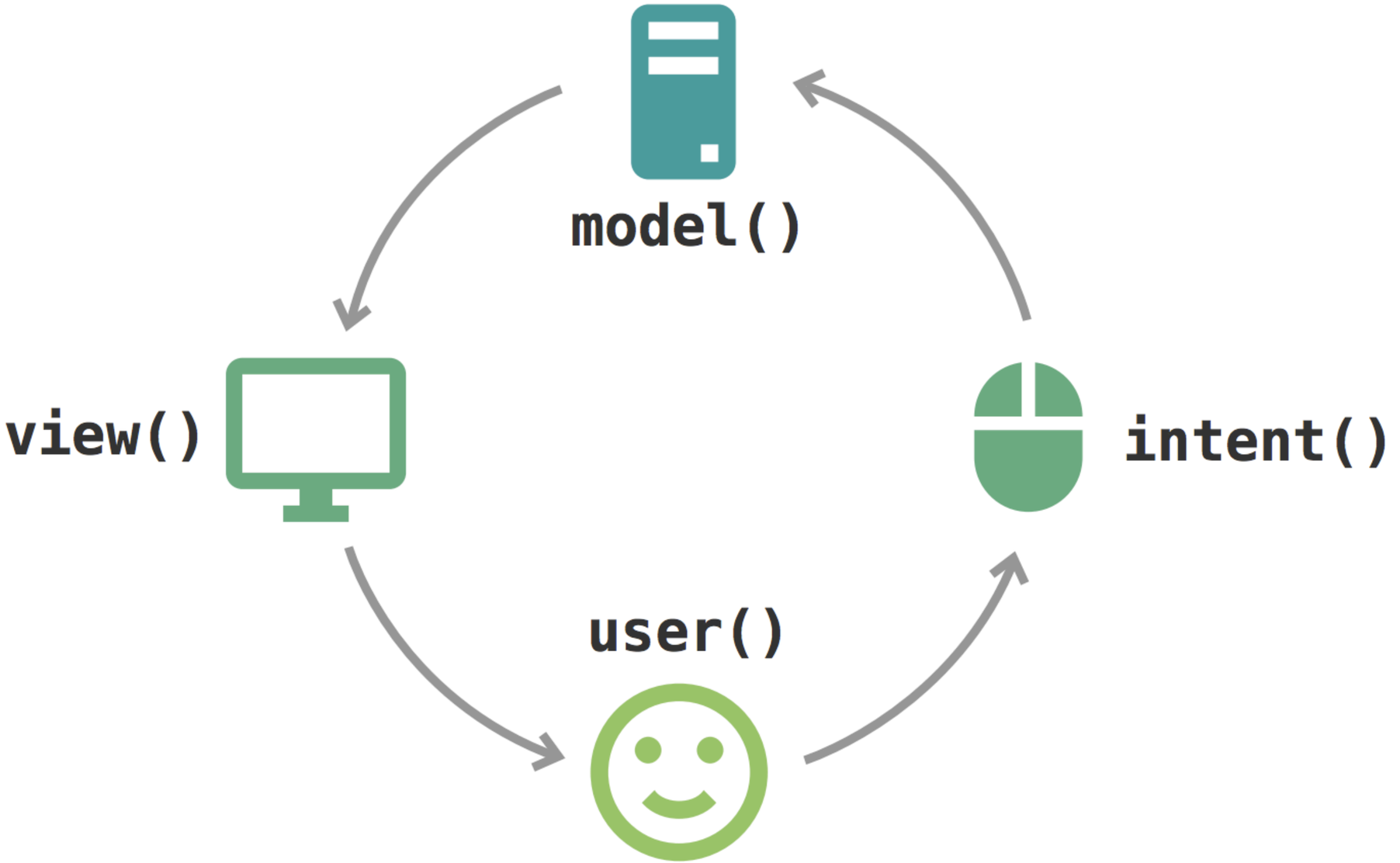
Model-View-Intent

for Android

Benoît Quenaudon @oldergod



"What if the user was a function?" by Andre Staltz



"What if the user was a function?" by Andre Staltz

```
user()
```

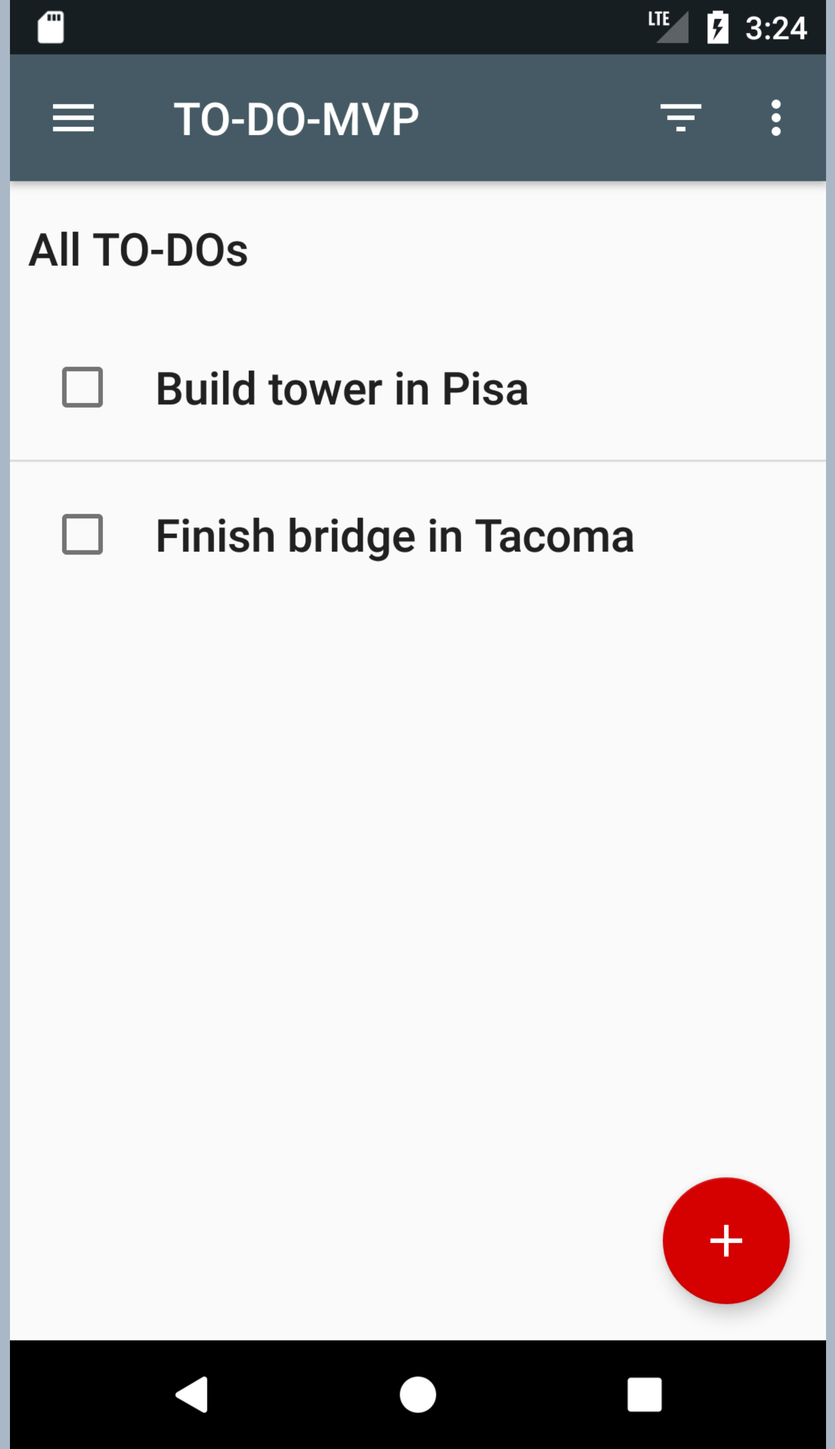
```
intent(user())
```

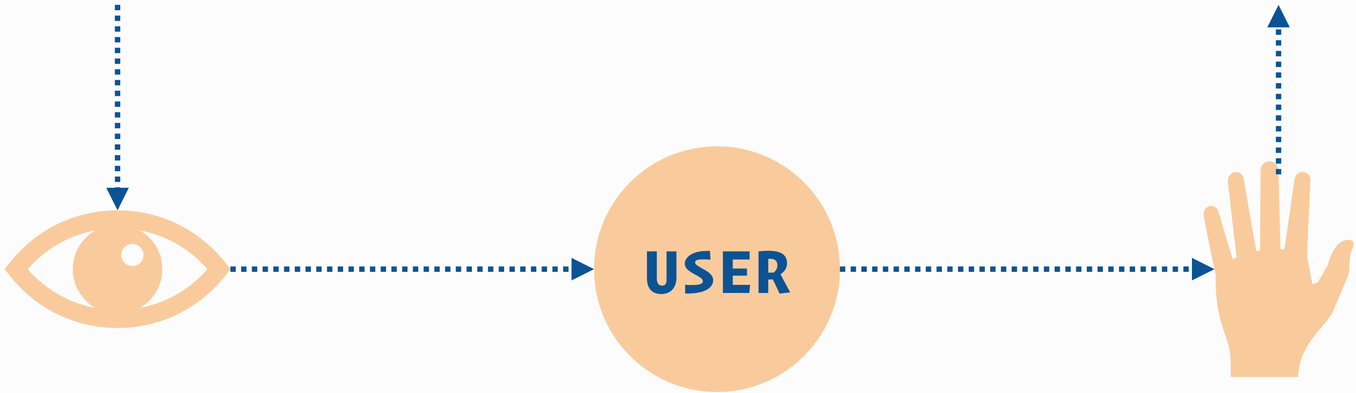
```
model(intent(user()))
```

```
view(model(intent(user())))
```

```
user(view(model(intent(user()))))
```

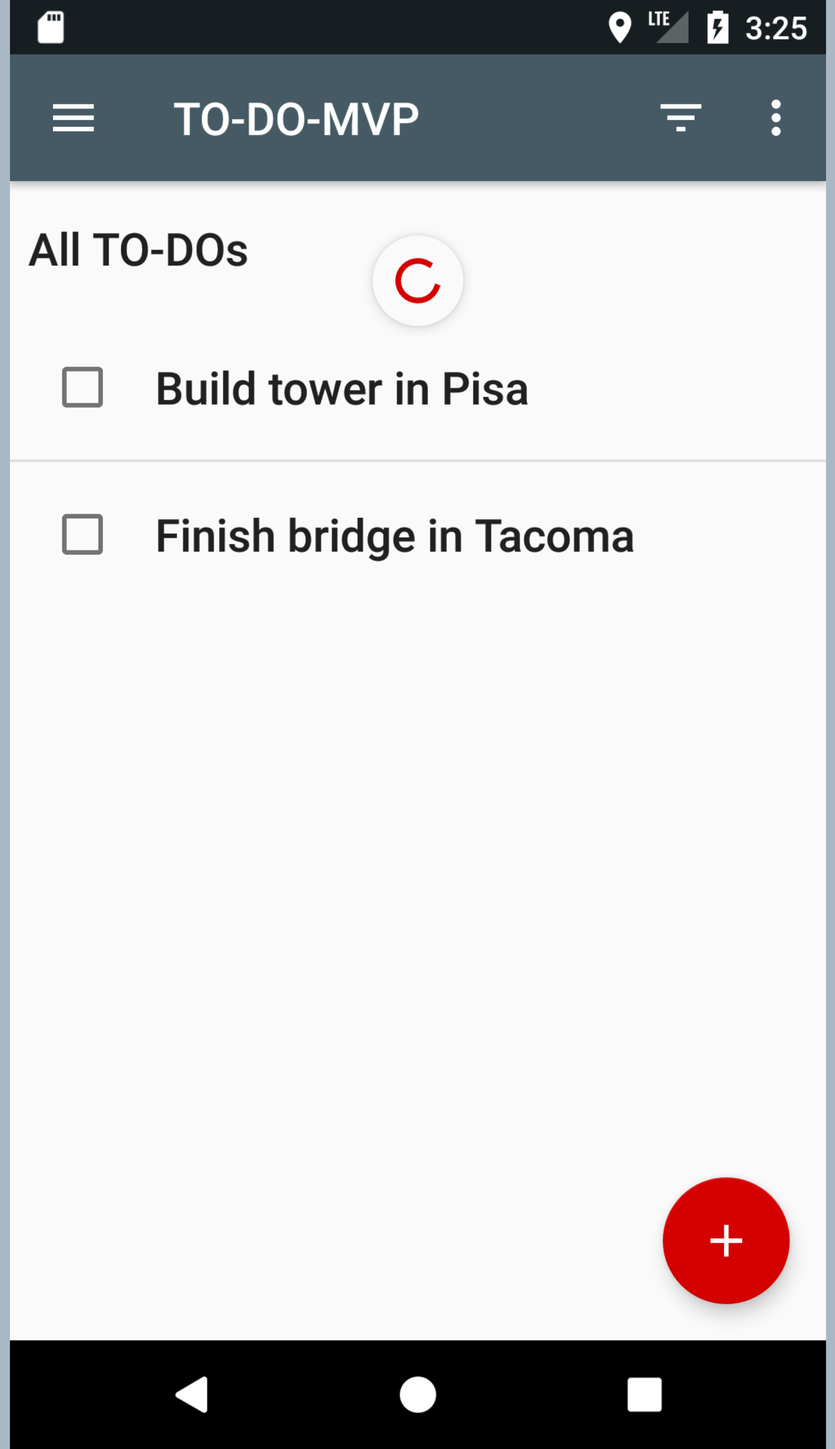
```
view(model(intent()))
```


3:24
TO-DO-MVP
All TO-DOs
Build tower in Pisa
Finish bridge in Tacoma

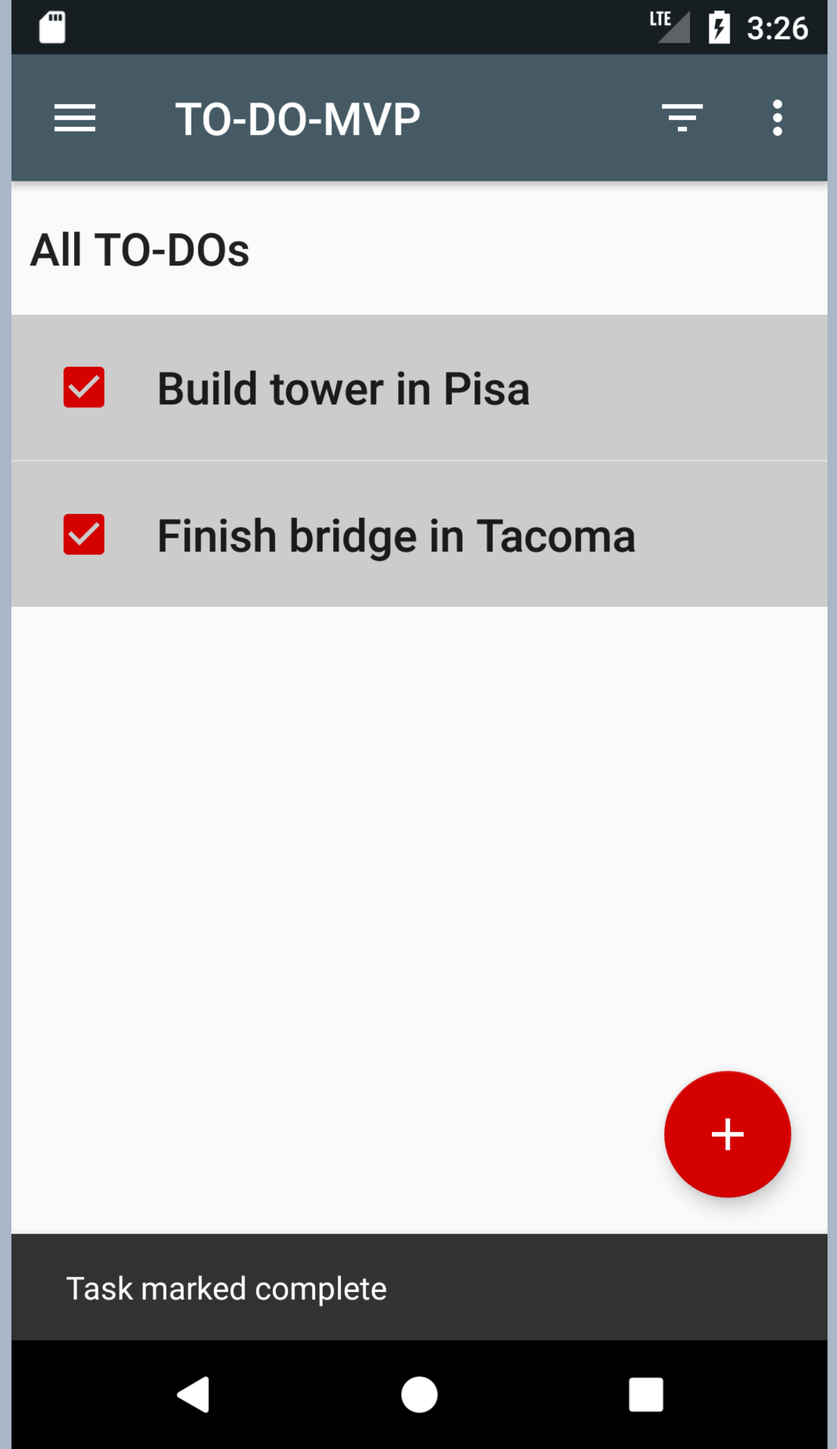

USER

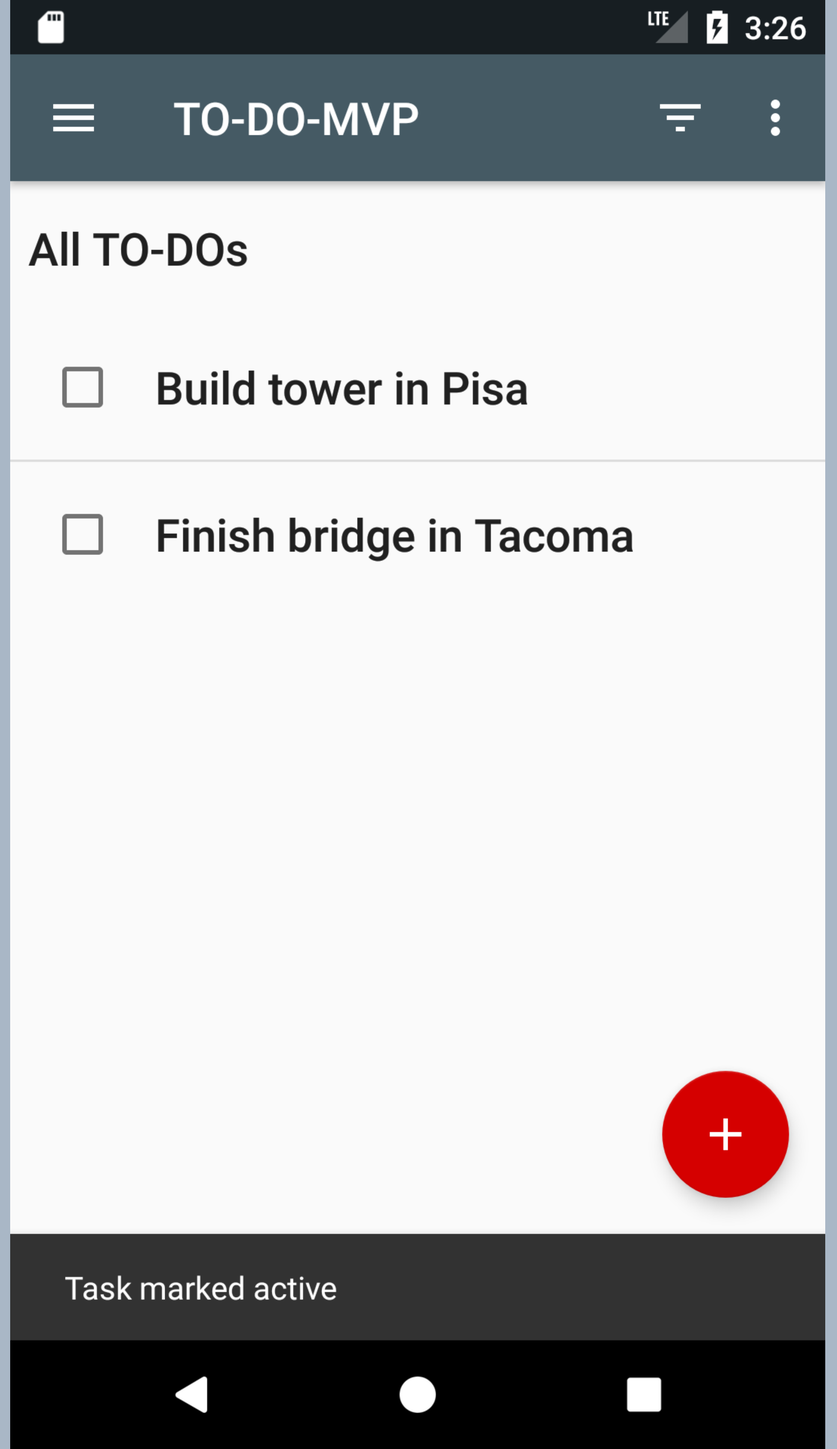
```
sealed class TasksIntent {
}
```

```
sealed class TasksIntent {
  object InitialIntent : TasksIntent()
}
```

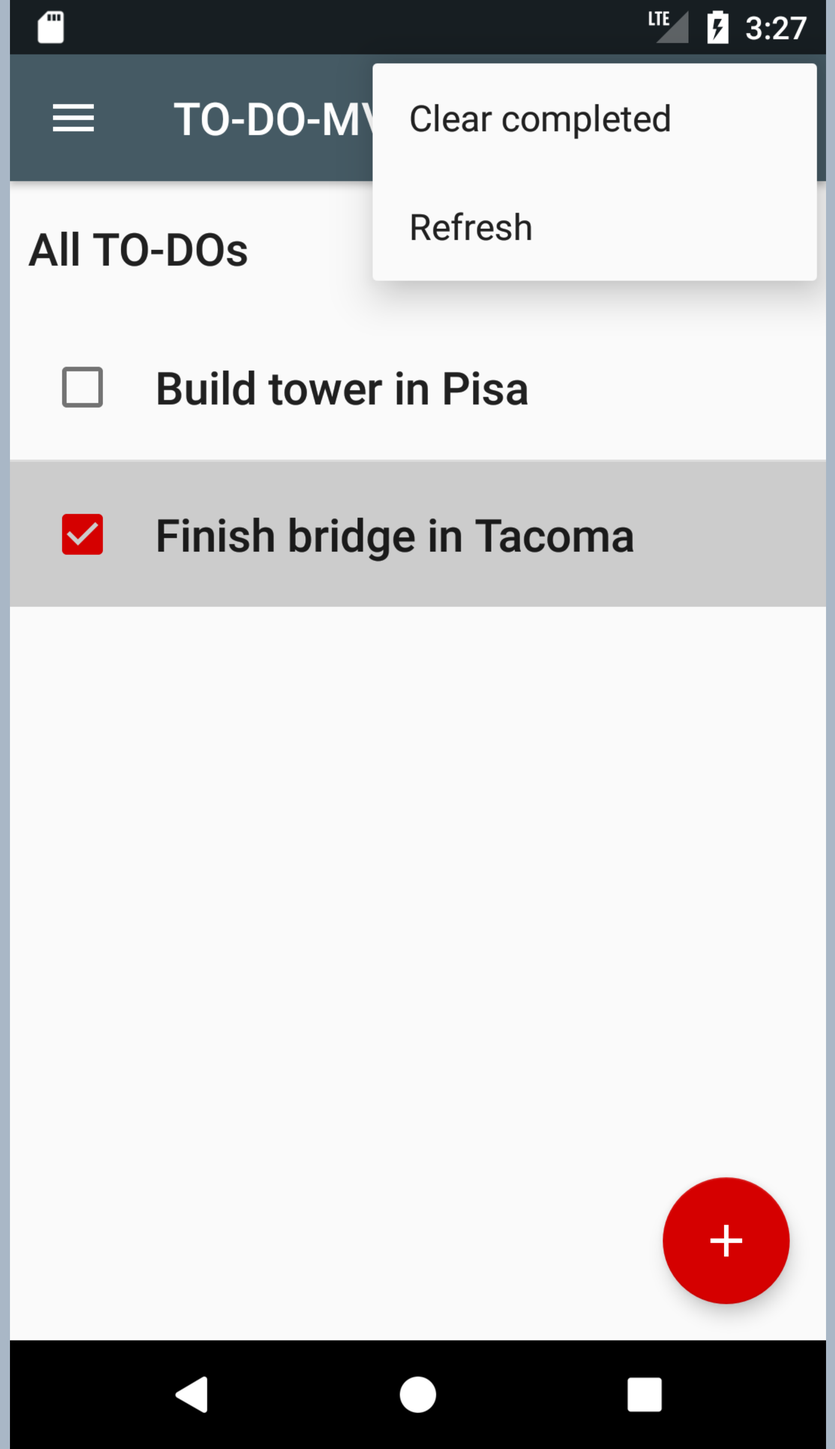

3:25
TO-DO-MVP
All TO-DOs
Build tower in Pisa
Finish bridge in Tacoma

```
sealed class TasksIntent {
  object InitialIntent : TasksIntent()

  object RefreshIntent : TasksIntent()
}
```

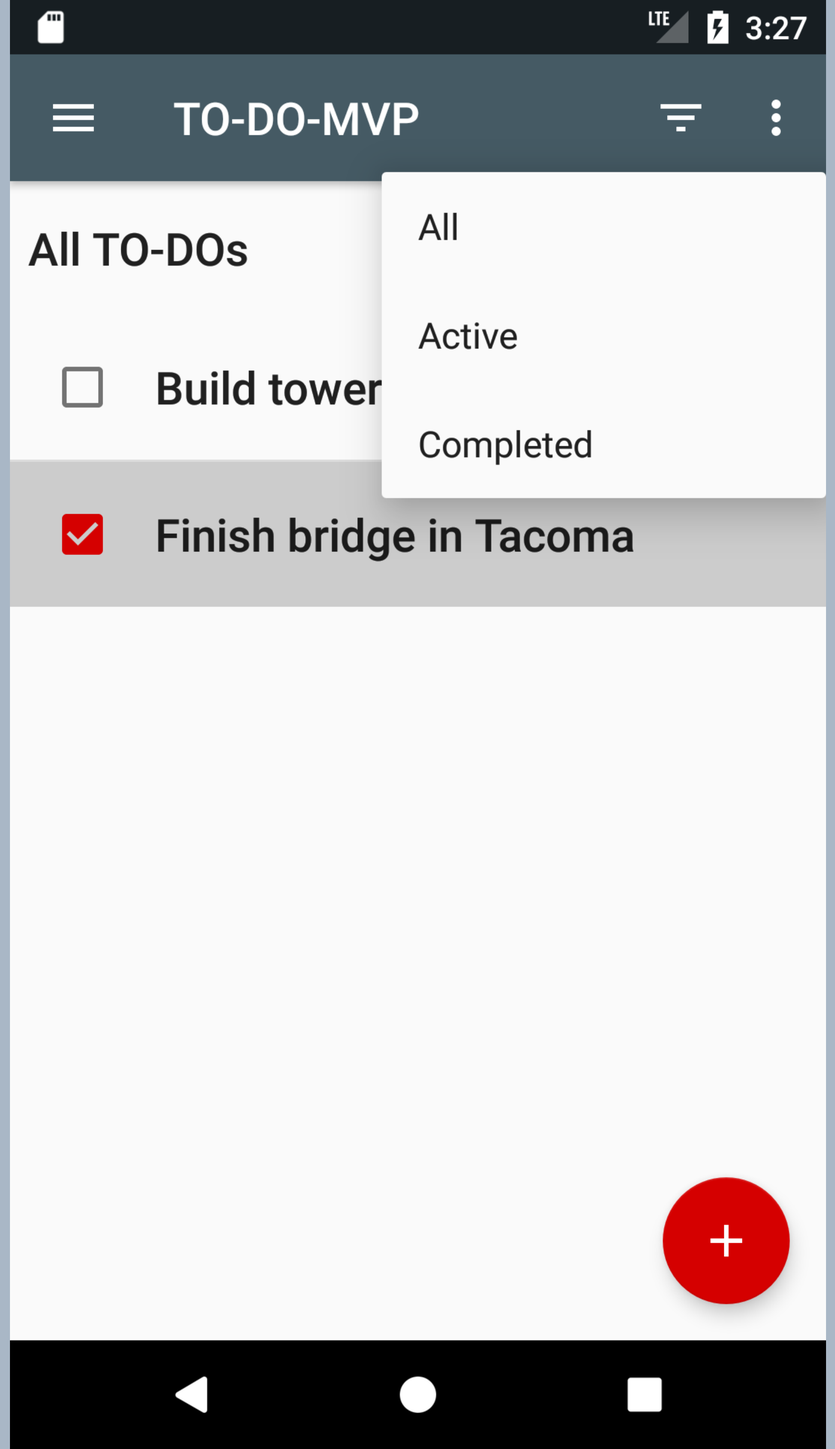

3:26
TO-DO-MVP
All TO-DOs
Build tower in Pisa
Finish bridge in Tacoma
Task marked complete


3:26
TO-DO-MVP
All TO-DOs
Build tower in Pisa
Finish bridge in Tacoma
Task marked active

```
sealed class TasksIntent {
  object InitialIntent : TasksIntent()

  object RefreshIntent : TasksIntent()

  data class ActivateTaskIntent(val task: Task) : TasksIntent()

  data class CompleteTaskIntent(val task: Task) : TasksIntent()
}
```


3:27
TO-DO-MV
Clear completed
Refresh
All TO-DOs
Build tower in Pisa
Finish bridge in Tacoma

```
sealed class TasksIntent {
  object InitialIntent : TasksIntent()

  object RefreshIntent : TasksIntent()

  data class ActivateTaskIntent(val task: Task) : TasksIntent()

  data class CompleteTaskIntent(val task: Task) : TasksIntent()

  object ClearCompletedTasksIntent : TasksIntent()
}
```


3:27
TO-DO-MVP
All TO-DOs
All
Active
Completed
Build tower
Finish bridge in Tacoma

```
sealed class TasksIntent {
  object InitialIntent : TasksIntent()

  object RefreshIntent : TasksIntent()

  data class ActivateTaskIntent(val task: Task) : TasksIntent()

  data class CompleteTaskIntent(val task: Task) : TasksIntent()

  object ClearCompletedTasksIntent : TasksIntent()

  data class ChangeFilterIntent(val filterType: TasksFilterType) : TasksIntent()
}
```

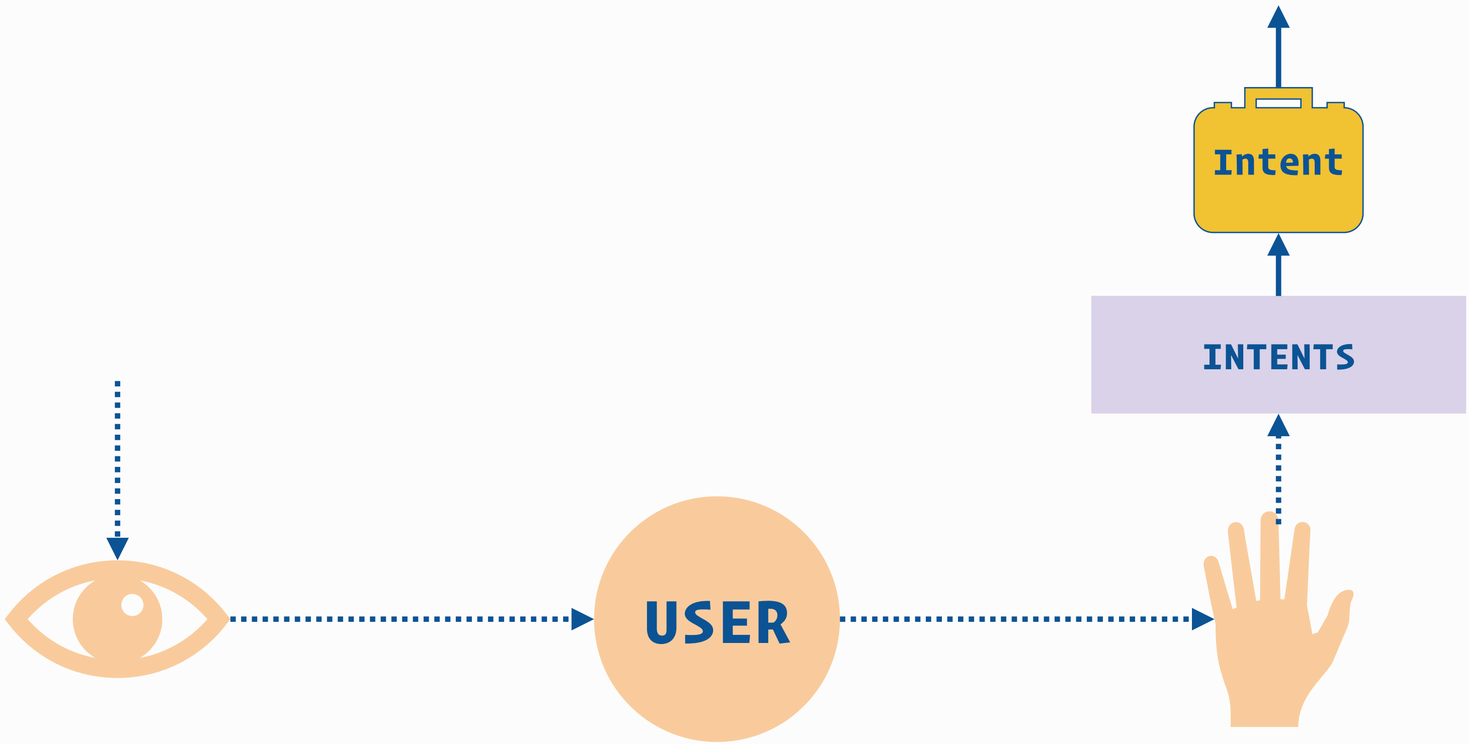
```
sealed class TasksIntent {
  object InitialIntent : TasksIntent()

  object RefreshIntent : TasksIntent()

  data class ActivateTaskIntent(val task: Task) : TasksIntent()

  data class CompleteTaskIntent(val task: Task) : TasksIntent()

  object ClearCompletedTasksIntent : TasksIntent()

  data class ChangeFilterIntent(val filterType: TasksFilterType) : TasksIntent()
}
```

```
class TasksFragment
  fun intents(): Observable<TasksIntent> {
  }
}
```

```
class TasksFragment
  fun intents(): Observable<TasksIntent> {
    return initialIntent()
  }

  private fun initialIntent(): Observable<InitialIntent> {
    return Observable.just(InitialIntent)
  }
}
```

```
class TasksFragment
  fun intents(): Observable<TasksIntent> {
    return Observable.merge(initialIntent(),
                            refreshIntent())
  }

  private fun refreshIntent(): Observable<RefreshIntent> {
    return RxSwipeRefreshLayout.refreshes(swipeRefreshLayout)
          .map { RefreshIntent }
  }
}
```

```
class TasksFragment
  fun intents(): Observable<TasksIntent> {
    return Observable.merge(initialIntent(),
                            refreshIntent(),
                            completeTaskIntent(),
                            activateTaskIntent(),
                            clearCompletedTaskIntent(),
                            changeFilterIntent())
  }
}
```


Intent
INTENTS
USER

```
intents // Observable<TasksIntent>
```

```
intents                                             // Observable<TasksIntent>
    .map { intent -> actionFromIntent(intent) } // Observable<TasksAction>
```

```
fun actionFromIntent(intent: TasksIntent): TasksAction =
    when (intent) {
      is InitialIntent ->
      is RefreshIntent ->
      is ActivateTaskIntent ->
      is CompleteTaskIntent ->
      is ClearCompletedTasksIntent ->
      is ChangeFilterIntent ->
    }
```

```
fun actionFromIntent(intent: TasksIntent): TasksAction =
    when (intent) {
      is InitialIntent ->
      is RefreshIntent ->
      is ActivateTaskIntent ->
      is CompleteTaskIntent ->
      is ClearCompletedTasksIntent ->
      is ChangeFilterIntent ->
    }
```

```
fun actionFromIntent(intent: TasksIntent): TasksAction =
    when (intent) {
      is InitialIntent ->
      is RefreshIntent ->
      is ActivateTaskIntent ->
      is CompleteTaskIntent ->
      is ClearCompletedTasksIntent ->
      is ChangeFilterIntent ->
    }
```

```
fun actionFromIntent(intent: TasksIntent): TasksAction =
    when (intent) {
      is InitialIntent -> LoadAndFilterTasksAction(TasksFilterType.ALL_TASKS)
      is RefreshIntent ->
      is ActivateTaskIntent ->
      is CompleteTaskIntent ->
      is ClearCompletedTasksIntent ->
      is ChangeFilterIntent ->
    }
```

```
fun actionFromIntent(intent: TasksIntent): TasksAction =
    when (intent) {
      is InitialIntent -> LoadAndFilterTasksAction(TasksFilterType.ALL_TASKS)
      is RefreshIntent -> LoadTasksAction
      is ActivateTaskIntent ->
      is CompleteTaskIntent ->
      is ClearCompletedTasksIntent ->
      is ChangeFilterIntent ->
    }
```

```
fun actionFromIntent(intent: TasksIntent): TasksAction =
    when (intent) {
      is InitialIntent -> LoadAndFilterTasksAction(TasksFilterType.ALL_TASKS)
      is RefreshIntent -> LoadTasksAction
      is ActivateTaskIntent -> ActivateTaskAction(intent.task)
      is CompleteTaskIntent ->
      is ClearCompletedTasksIntent ->
      is ChangeFilterIntent ->
    }
```

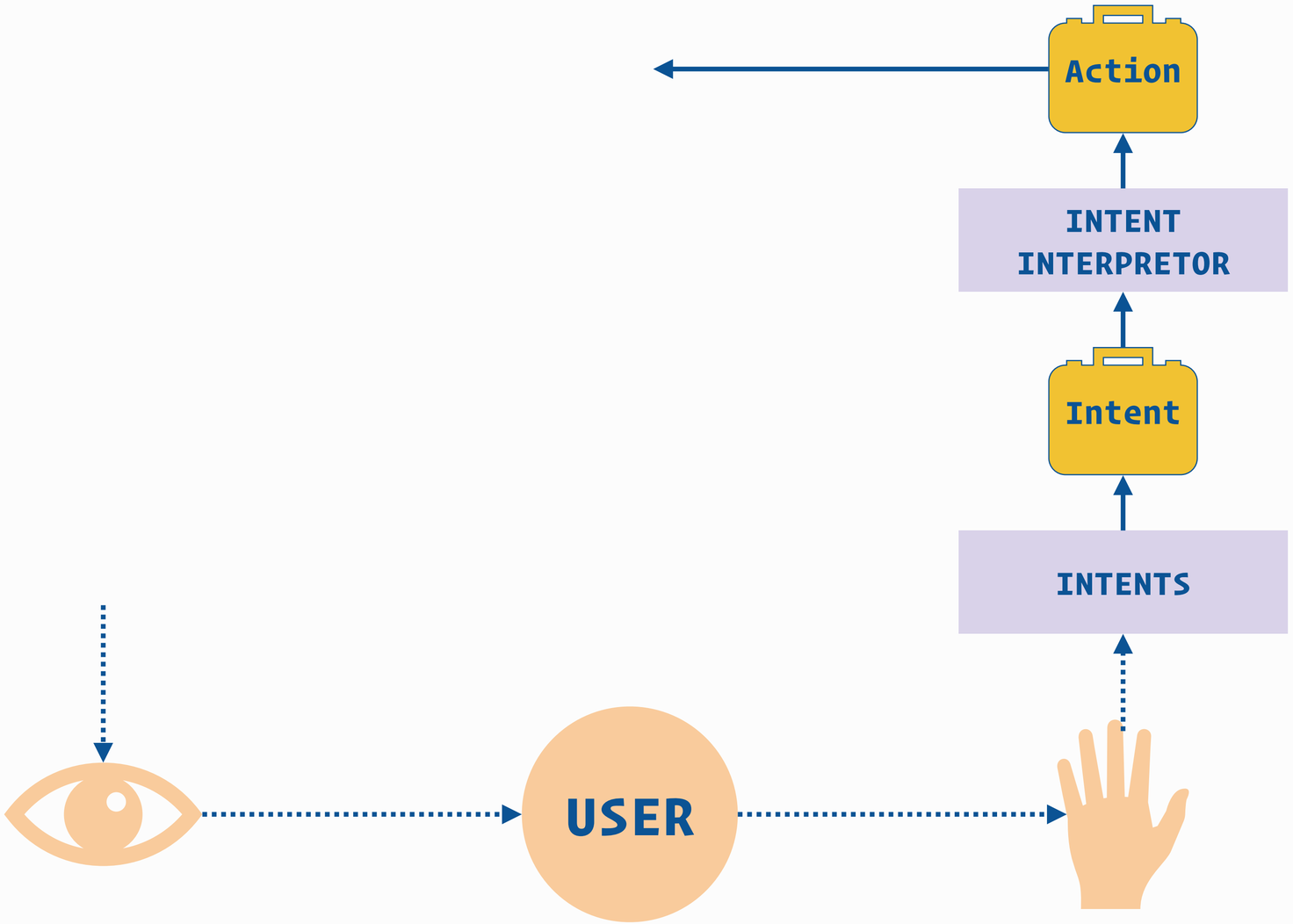
```
fun actionFromIntent(intent: TasksIntent): TasksAction =
    when (intent) {
      is InitialIntent -> LoadAndFilterTasksAction(TasksFilterType.ALL_TASKS)
      is RefreshIntent -> LoadTasksAction
      is ActivateTaskIntent -> ActivateTaskAction(intent.task)
      is CompleteTaskIntent -> CompleteTaskAction(intent.task)
      is ClearCompletedTasksIntent ->
      is ChangeFilterIntent ->
    }
```

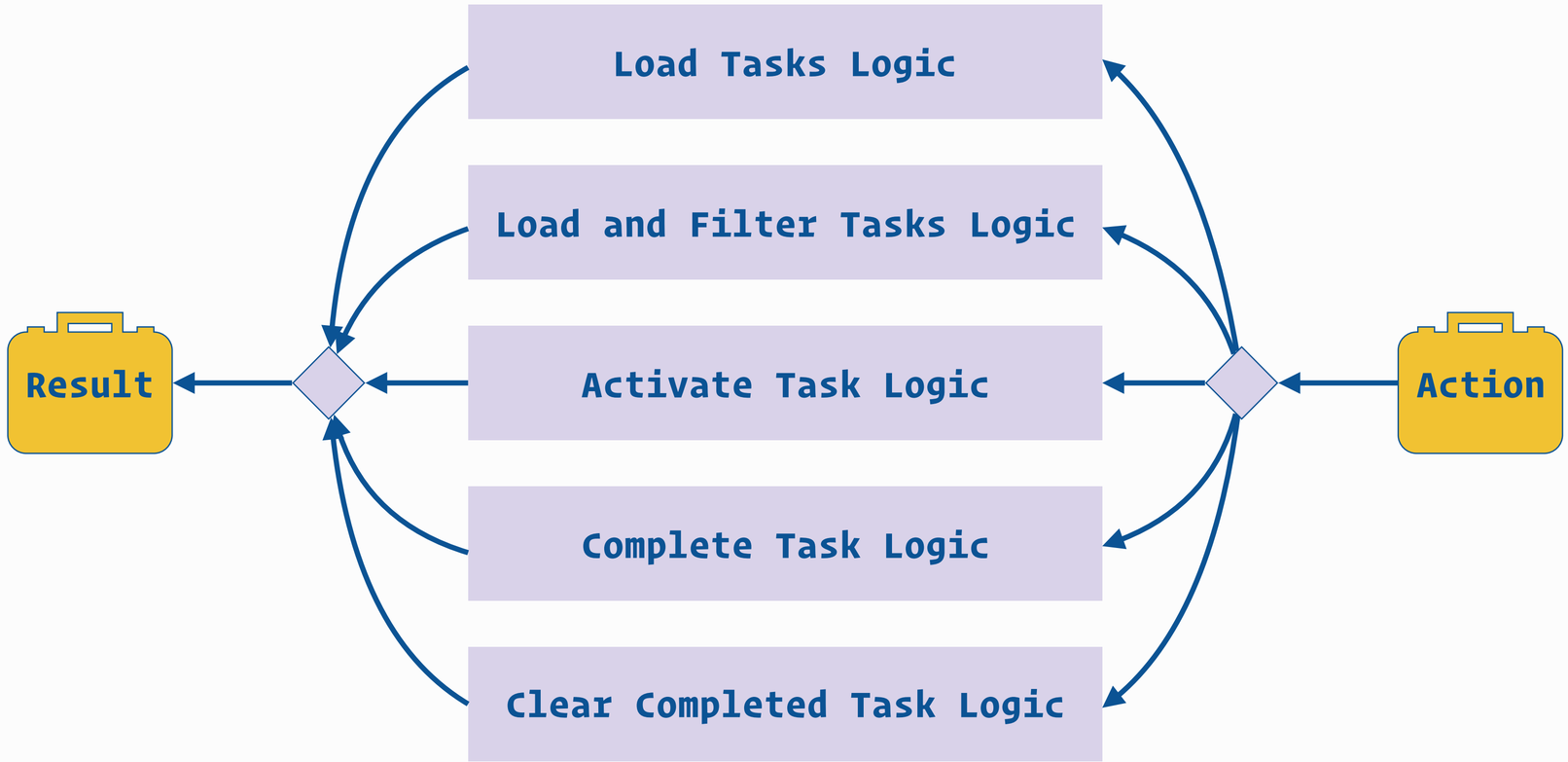
```
fun actionFromIntent(intent: TasksIntent): TasksAction =
    when (intent) {
      is InitialIntent -> LoadAndFilterTasksAction(TasksFilterType.ALL_TASKS)
      is RefreshIntent -> LoadTasksAction
      is ActivateTaskIntent -> ActivateTaskAction(intent.task)
      is CompleteTaskIntent -> CompleteTaskAction(intent.task)
      is ClearCompletedTasksIntent -> ClearCompletedTasksAction
      is ChangeFilterIntent ->
    }
```

```
fun actionFromIntent(intent: TasksIntent): TasksAction =
    when (intent) {
      is InitialIntent -> LoadAndFilterTasksAction(TasksFilterType.ALL_TASKS)
      is RefreshIntent -> LoadTasksAction
      is ActivateTaskIntent -> ActivateTaskAction(intent.task)
      is CompleteTaskIntent -> CompleteTaskAction(intent.task)
      is ClearCompletedTasksIntent -> ClearCompletedTasksAction
      is ChangeFilterIntent -> LoadAndFilterTasksAction(intent.filterType)
    }
```

```
fun actionFromIntent(intent: TasksIntent): TasksAction =
    when (intent) {
      is InitialIntent -> LoadAndFilterTasksAction(TasksFilterType.ALL_TASKS)
      is RefreshIntent -> LoadTasksAction
      is ActivateTaskIntent -> ActivateTaskAction(intent.task)
      is CompleteTaskIntent -> CompleteTaskAction(intent.task)
      is ClearCompletedTasksIntent -> ClearCompletedTasksAction
      is ChangeFilterIntent -> LoadAndFilterTasksAction(intent.filterType)
    }
```

```
sealed class TasksAction {
    data class LoadAndFilterTasksAction(val filterType: TasksFilterType) : TasksAction()

    object LoadTasksAction : TasksAction()

    data class ActivateTaskAction(val task: Task) : TasksAction()

    data class CompleteTaskAction(val task: Task) : TasksAction()

    object ClearCompletedTasksAction : TasksAction()
}
```


Action
INTENT
INTERPRETOR
Intent
INTENTS
USER


Load Tasks Logic
Load and Filter Tasks Logic
Result
Activate Task Logic
Action
Complete Task Logic
Clear Completed Task Logic

```
var actionProcessor: ObservableTransformer<TasksAction, TasksResult> =
    ObservableTransformer { actions: Observable<TasksAction> ->
    }
```

```
var actionProcessor: ObservableTransformer<TasksAction, TasksResult> =
    ObservableTransformer { actions: Observable<TasksAction> ->
    }
```

```
var actionProcessor: ObservableTransformer<TasksAction, TasksResult> =
    ObservableTransformer { actions: Observable<TasksAction> ->
      actions.publish { shared ->
        Observable.merge()
      }
    }
```

```
var actionProcessor: ObservableTransformer<TasksAction, TasksResult> =
    ObservableTransformer { actions: Observable<TasksAction> ->
      actions.publish { shared ->
        Observable.merge(
          shared.ofType(LoadTasksAction::class.java).compose(loadTasksProcessor)
        )
      }
    }
```

```
var actionProcessor: ObservableTransformer<TasksAction, TasksResult> =
    ObservableTransformer { actions: Observable<TasksAction> ->
      actions.publish { shared ->
        Observable.merge(
          shared.ofType(LoadTasksAction::class.java).compose(loadTasksProcessor),
          shared.ofType(LoadAndFilterTasksAction::class.java).compose(loadAndFilterTasksProcessor)
        )
      }
    }
```

```
var actionProcessor: ObservableTransformer<TasksAction, TasksResult> =
    ObservableTransformer { actions: Observable<TasksAction> ->
      actions.publish { shared ->
        Observable.merge(
          shared.ofType(LoadTasksAction::class.java).compose(loadTasksProcessor),
          shared.ofType(LoadAndFilterTasksAction::class.java).compose(loadAndFilterTasksProcessor),
          shared.ofType(ActivateTaskAction::class.java).compose(activateTaskProcessor)
        )
      }
    }
```

```
var actionProcessor: ObservableTransformer<TasksAction, TasksResult> =
    ObservableTransformer { actions: Observable<TasksAction> ->
      actions.publish { shared ->
        Observable.merge(
          shared.ofType(LoadTasksAction::class.java).compose(loadTasksProcessor),
          shared.ofType(LoadAndFilterTasksAction::class.java).compose(loadAndFilterTasksProcessor),
          shared.ofType(ActivateTaskAction::class.java).compose(activateTaskProcessor),
          shared.ofType(ClearCompletedTasksAction::class.java).compose(clearCompletedTasksProcessor)
        )
      }
    }
```

```
var actionProcessor: ObservableTransformer<TasksAction, TasksResult> =
    ObservableTransformer { actions: Observable<TasksAction> ->
      actions.publish { shared ->
        Observable.merge(
          shared.ofType(LoadTasksAction::class.java).compose(loadTasksProcessor),
          shared.ofType(LoadAndFilterTasksAction::class.java).compose(loadAndFilterTasksProcessor),
          shared.ofType(ActivateTaskAction::class.java).compose(activateTaskProcessor),
          shared.ofType(ClearCompletedTasksAction::class.java).compose(clearCompletedTasksProcessor),
          shared.ofType(CompleteTaskAction::class.java).compose(completeTaskProcessor)
        )
      }
    }
```

```
var actionProcessor: ObservableTransformer<TasksAction, TasksResult> =
    ObservableTransformer { actions: Observable<TasksAction> ->
      actions.publish { shared ->
        Observable.merge(
          shared.ofType(LoadTasksAction::class.java).compose(loadTasksProcessor),
          shared.ofType(LoadAndFilterTasksAction::class.java).compose(loadAndFilterTasksProcessor),
          shared.ofType(ActivateTaskAction::class.java).compose(activateTaskProcessor),
          shared.ofType(ClearCompletedTasksAction::class.java).compose(clearCompletedTasksProcessor),
          shared.ofType(CompleteTaskAction::class.java).compose(completeTaskProcessor)
        )
      }
    }
```

```
val loadTasksProcessor =
    ObservableTransformer { actions: Observable<LoadTasksAction> ->
      actions.switchMap {
        tasksRepository.getTasks() // Observable<List<Tasks>>
      }
    }
```

```
val loadTasksProcessor =
    ObservableTransformer { actions: Observable<LoadTasksAction> ->
      actions.switchMap {
        tasksRepository.getTasks() // Observable<List<Tasks>>
            .startWith(LoadTasksResult.InFlight)
      }
    }
```

```
val loadTasksProcessor =
    ObservableTransformer { actions: Observable<LoadTasksAction> ->
      actions.switchMap {
        tasksRepository.getTasks() // Observable<List<Tasks>>
            .startWith(LoadTasksResult.InFlight)
            .map { tasks -> LoadTasksResult.Success(tasks) }
      }
    }
```

```
val loadTasksProcessor =
    ObservableTransformer { actions: Observable<LoadTasksAction> ->
      actions.switchMap {
        tasksRepository.getTasks() // Observable<List<Tasks>>
            .startWith(LoadTasksResult.InFlight)
            .map { tasks -> LoadTasksResult.Success(tasks) }
            .onErrorReturn { t -> loadTasksResult.Failure(t) }
      }
    }
```

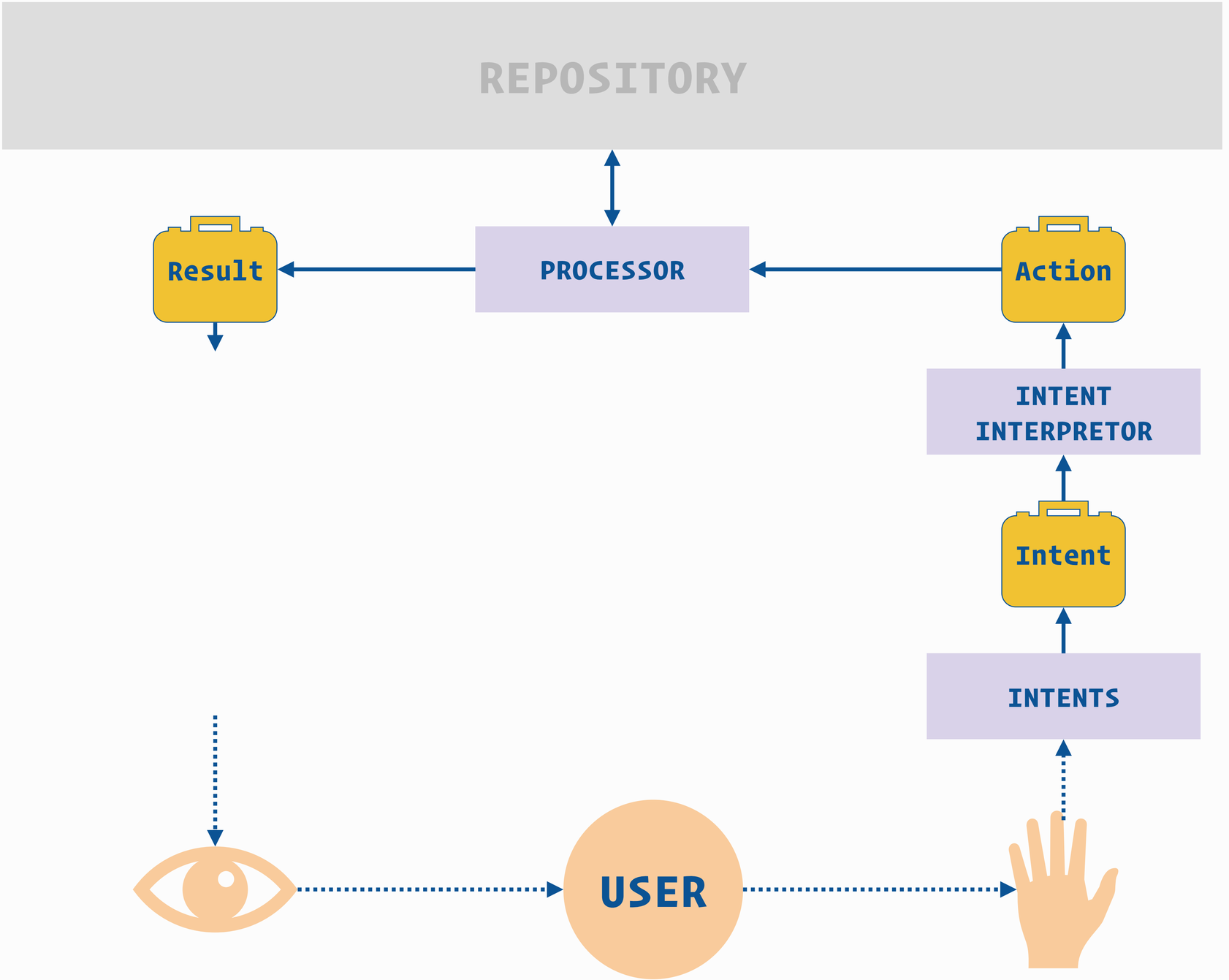
```
val loadTasksProcessor =
    ObservableTransformer { actions: Observable<LoadTasksAction> ->
      actions.switchMap {
        tasksRepository.getTasks() // Observable<List<Tasks>>
            .startWith(LoadTasksResult.InFlight)
            .map { tasks -> LoadTasksResult.Success(tasks) }
            .onErrorReturn { t -> loadTasksResult.Failure(t) }
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread())
      }
    }
```

```
val loadTasksProcessor =
    ObservableTransformer { actions: Observable<LoadTasksAction> ->
      actions.switchMap {
        tasksRepository.getTasks() // Observable<List<Tasks>>
            .startWith(LoadTasksResult.InFlight)
            .map { tasks -> LoadTasksResult.Success(tasks) }
            .onErrorReturn { t -> loadTasksResult.Failure(t) }
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread())
      }
    }
```

```
val loadTasksProcessor =
    ObservableTransformer { actions: Observable<LoadTasksAction> ->
      actions.switchMap {
        tasksRepository.getTasks() // Observable<List<Tasks>>
            .map { tasks -> LoadTasksResult.Success(tasks) }
            .onErrorReturn { t -> loadTasksResult.Failure(t) }
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread())
            .startWith(LoadTasksResult.InFlight)
      }
    }
```

```
var actionProcessor: ObservableTransformer<TasksAction, TasksResult> =
    ObservableTransformer { actions: Observable<TasksAction> ->
      actions.publish { shared ->
        Observable.merge(
          shared.ofType(LoadTasksAction::class.java).compose(loadTasksProcessor),
          shared.ofType(LoadAndFilterTasksAction::class.java).compose(loadAndFilterTasksProcessor),
          shared.ofType(ActivateTaskAction::class.java).compose(activateTaskProcessor),
          shared.ofType(ClearCompletedTasksAction::class.java).compose(clearCompletedTasksProcessor),
          shared.ofType(CompleteTaskAction::class.java).compose(completeTaskProcessor)
        )
      }
    }
```

```
intents                                             // Observable<TasksIntent>
    .map { intent -> actionFromIntent(intent) } // Observable<TasksAction>
    .compose(actionProcessor)                       // Observable<TasksResult>
```


REPOSITORY
Result
PROCESSOR
Action
INTENT
INTERPRETOR
Intent
INTENTS
USER

What do we need to render anything?

```
data class TasksViewState(
)
```

```
data class TasksViewState(
    val isLoading: Boolean
)
```

```
data class TasksViewState(
    val isLoading: Boolean,
    val tasksFilterType: TasksFilterType
)
```

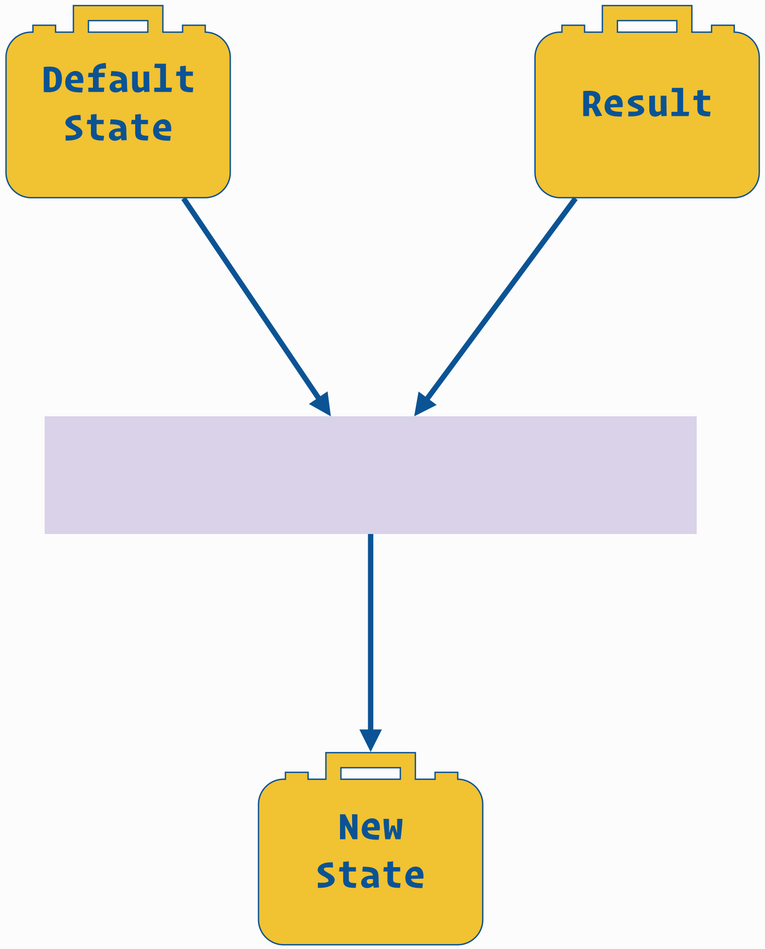
```
data class TasksViewState(
    val isLoading: Boolean,
    val tasksFilterType: TasksFilterType,
    val tasks: List<Task>
)
```

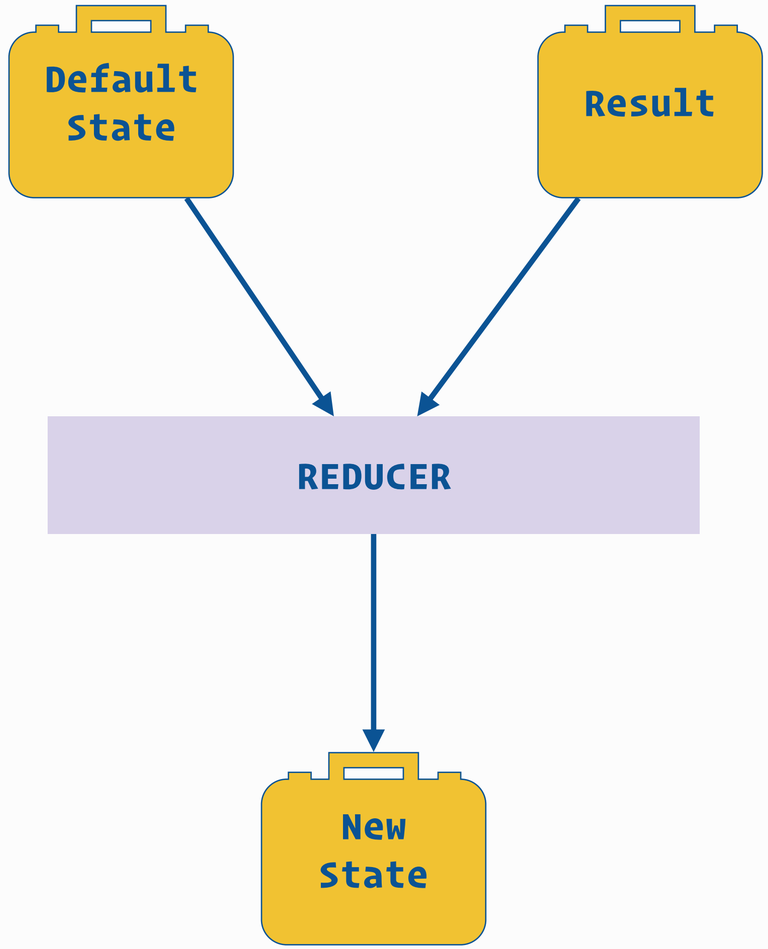
```
data class TasksViewState(
    val isLoading: Boolean,
    val tasksFilterType: TasksFilterType,
    val tasks: List<Task>,
    val error: Throwable?
)
```

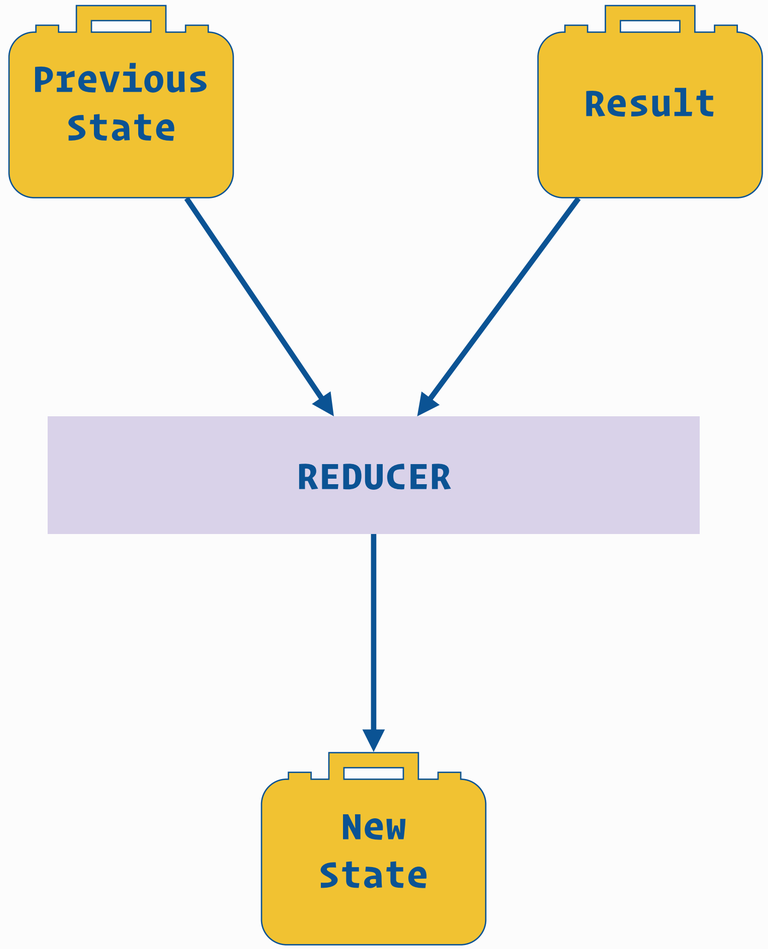
```
data class TasksViewState(
    val isLoading: Boolean,
    val tasksFilterType: TasksFilterType,
    val tasks: List<Task>,
    val error: Throwable?,
    val taskComplete: Boolean,
    val taskActivated: Boolean,
    val completedTasksCleared: Boolean
)
```

```
data class TasksViewState(
    val isLoading: Boolean,
    val tasksFilterType: TasksFilterType,
    val tasks: List<Task>,
    val error: Throwable?,
    val taskComplete: Boolean,
    val taskActivated: Boolean,
    val completedTasksCleared: Boolean
)
```

```kotlin
data class TasksViewState(
    val isLoading: Boolean,
    val tasksFilterType: TasksFilterType,
    val tasks: List<Task>,
    val error: Throwable?,
    val taskComplete: Boolean,
    val taskActivated: Boolean,
    val completedTasksCleared: Boolean
) {
    companion object Factory {
        fun default() = TasksViewState(
            isLoading = false,
            tasksFilterType = ALL_TASKS,
            tasks = emptyList(),
            error = null,
            taskComplete = false,
            taskActivated = false,
            completedTasksCleared = false)
    }
}
```


Default State
Result
New State


Default State
Result
REDUCER
New State


Previous State
Result
REDUCER
New State

```
intents                                             // Observable<TasksIntent>
    .map { intent -> actionFromIntent(intent) }     // Observable<TasksAction>
    .compose(actionProcessor)                       // Observable<TasksResult>
    .scan(TasksViewState.default(), reducer)        // Observable<TasksViewState>
```

```
val reducer = BiFunction<TasksViewState, TasksResult, TasksViewState>
  { previousState: TasksViewState, result: TasksResult ->
  }
```

```
val reducer = BiFunction<TasksViewState, TasksResult, TasksViewState>
  { previousState: TasksViewState, result: TasksResult ->
  }
```

```
val reducer = BiFunction<TasksViewState, TasksResult, TasksViewState>
  { previousState: TasksViewState, result: TasksResult ->
     when (result) {
       is LoadTasksResult -> /***/
       is LoadAndFilterTasksResult -> /***/
       is CompleteTaskResult -> /***/
       is ActivateTaskResult -> /***/
       is ClearCompletedTasksResult -> /***/
     }
   }
```

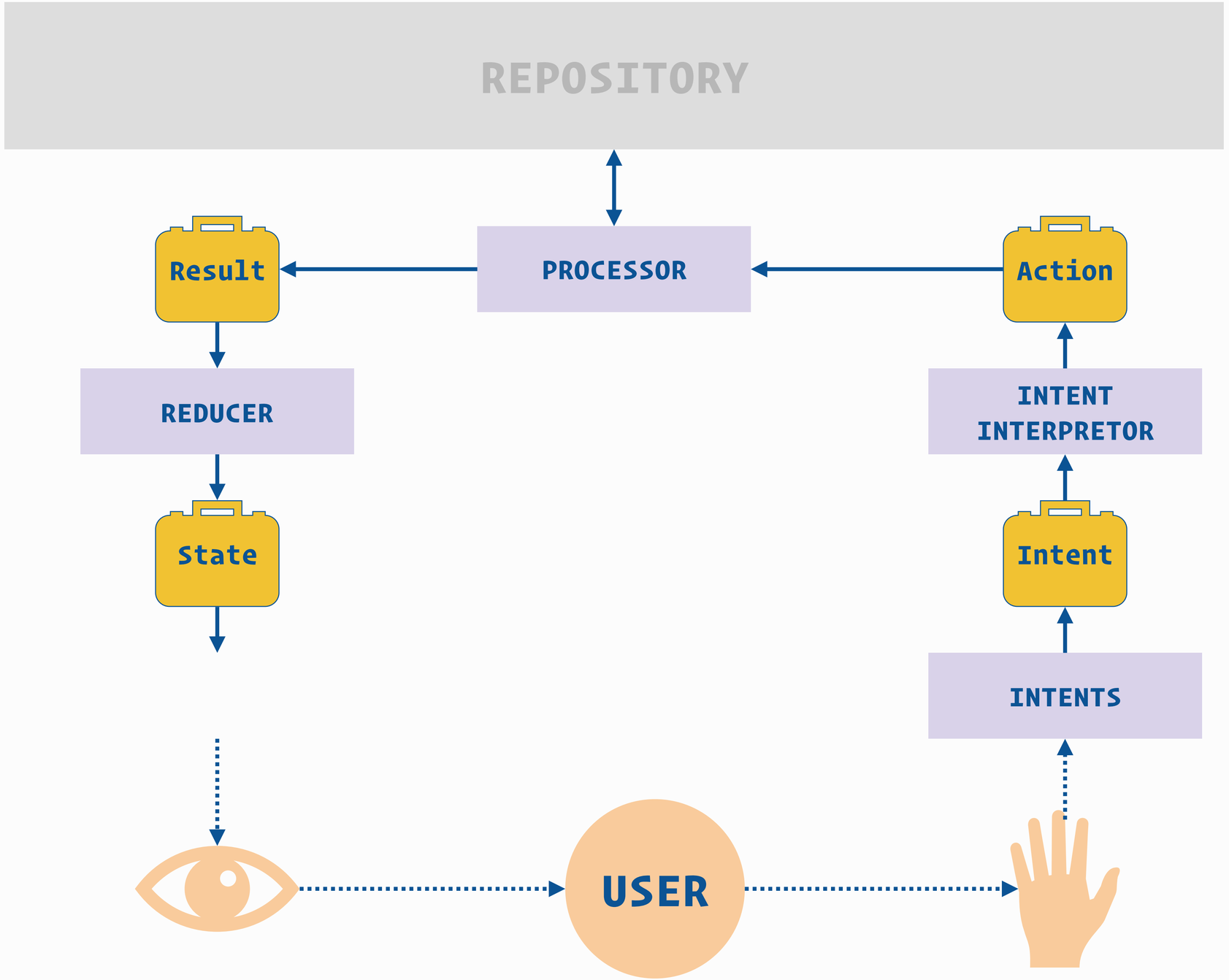
```
val reducer = BiFunction<TasksViewState, TasksResult, TasksViewState>
  { previousState: TasksViewState, result: TasksResult ->
    when (result) {
      is LoadTasksResult -> {
        when (result) {
          is LoadTasksResult.InFlight -> /***/
          is LoadTasksResult.Failure -> /***/
          is LoadTasksResult.Success -> /***/
        }
      }
      is LoadAndFilterTasksResult -> /***/
      is CompleteTaskResult -> /***/
      is ActivateTaskResult -> /***/
      is ClearCompletedTasksResult -> /***/
    }
  }
```

```
val reducer = BiFunction<TasksViewState, TasksResult, TasksViewState>
  { previousState: TasksViewState, result: TasksResult ->
     when (result) {
       is LoadTasksResult -> {
         when (result) {
           is LoadTasksResult.InFlight -> previousState.copy(isLoading = true)
           is LoadTasksResult.Failure -> /***/
           is LoadTasksResult.Success -> /***/
         }
       }
       is LoadAndFilterTasksResult -> /***/
       is CompleteTaskResult -> /***/
       is ActivateTaskResult -> /***/
       is ClearCompletedTasksResult -> /***/
     }
   }
```

```
val reducer = BiFunction<TasksViewState, TasksResult, TasksViewState>
  { previousState: TasksViewState, result: TasksResult ->
    when (result) {
      is LoadTasksResult -> {
        when (result) {
          is LoadTasksResult.InFlight -> previousState.copy(isLoading = true)
          is LoadTasksResult.Failure -> previousState.copy(isLoading = false,
                                                           error = result.error)
          is LoadTasksResult.Success -> /***/
        }
      }
      is LoadAndFilterTasksResult -> /***/
      is CompleteTaskResult -> /***/
      is ActivateTaskResult -> /***/
      is ClearCompletedTasksResult -> /***/
    }
  }
```

```
val reducer = BiFunction<TasksViewState, TasksResult, TasksViewState>
  { previousState: TasksViewState, result: TasksResult ->
     when (result) {
       is LoadTasksResult -> {
         when (result) {
           is LoadTasksResult.InFlight -> previousState.copy(isLoading = true)
           is LoadTasksResult.Failure -> previousState.copy(isLoading = false,
                                                            error = result.error)
           is LoadTasksResult.Success -> {
               previousState.copy(isLoading = false,
                                  tasks = result.tasks)
           }
         }
       }
       is LoadAndFilterTasksResult -> /***/
       is CompleteTaskResult -> /***/
       is ActivateTaskResult -> /***/
       is ClearCompletedTasksResult -> /***/
     }
   }
```

```
val reducer = BiFunction<TasksViewState, TasksResult, TasksViewState>
  { previousState: TasksViewState, result: TasksResult ->
     when (result) {
       is LoadTasksResult -> {
         when (result) {
           is LoadTasksResult.InFlight -> previousState.copy(isLoading = true)
           is LoadTasksResult.Failure -> previousState.copy(isLoading = false,
                                                            error = result.error)
           is LoadTasksResult.Success -> {
               previousState.copy(isLoading = false,
                                  tasks = result.tasks)
           }
         }
       }
       is LoadAndFilterTasksResult -> /***/
       is CompleteTaskResult -> /***/
       is ActivateTaskResult -> /***/
       is ClearCompletedTasksResult -> /***/
     }
   }
```


REPOSITORY
Result
PROCESSOR
Action
REDUCER
INTENT
INTERPRETOR
State
Intent
INTENTS
USER

```
intents                                                 // Observable<TasksIntent>
    .map { intent -> actionFromIntent(intent) }         // Observable<TasksAction>
    .compose(actionProcessor)                           // Observable<TasksResult>
    .scan(TasksViewState.default(), reducer)            // Observable<TasksViewState>
```

```
intents                                                 // Observable<TasksIntent>
    .map { intent -> actionFromIntent(intent) }         // Observable<TasksAction>
    .compose(actionProcessor)                           // Observable<TasksResult>
    .scan(TasksViewState.default(), reducer)            // Observable<TasksViewState>
    .subscribe { state -> render(state) }
```

```
fun render(state: TasksViewState) {
}
```

```
fun render(state: TasksViewState) {
    swipeRefreshLayout.isRefreshing = state.isLoading
}
```

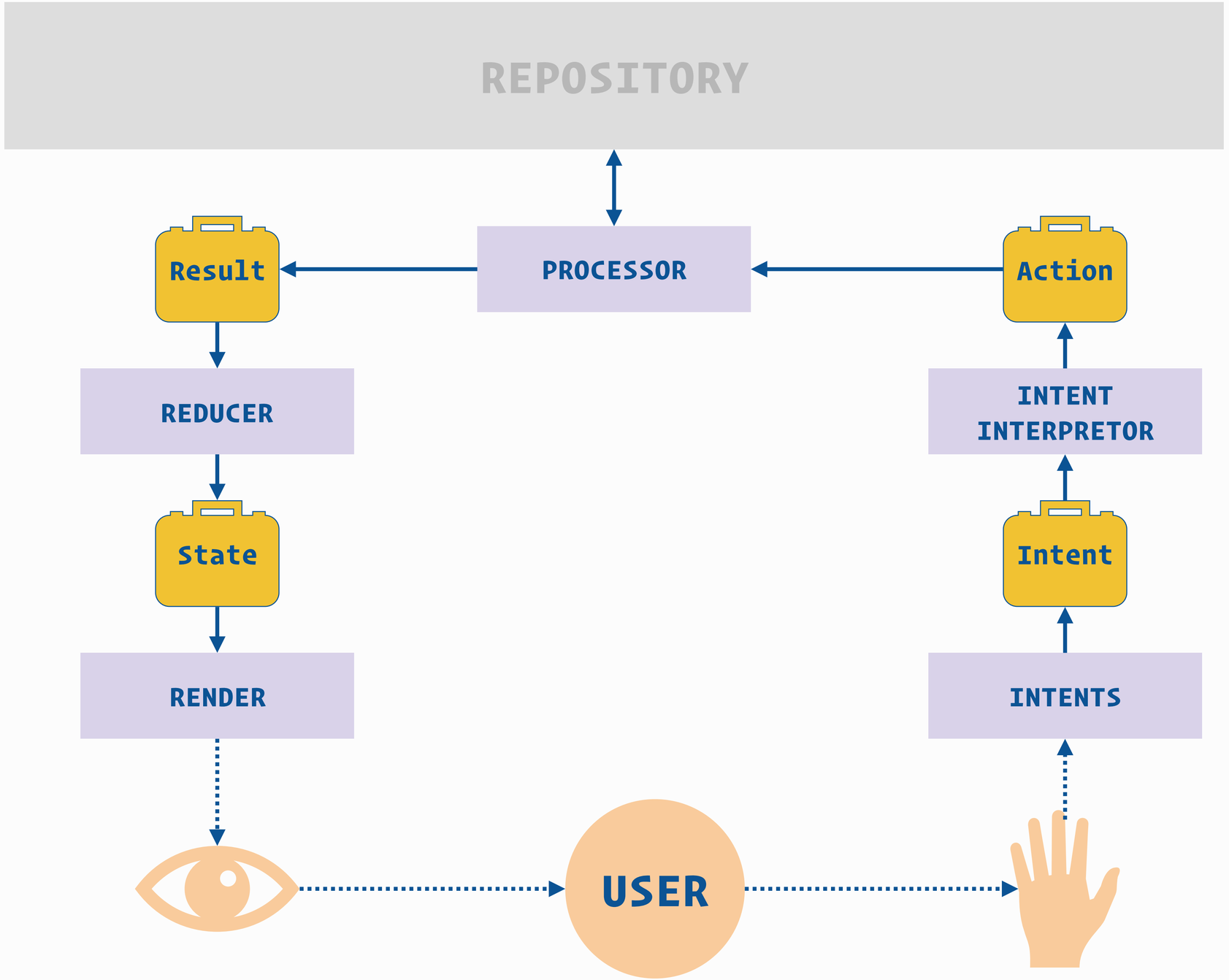
```
fun render(state: TasksViewState) {
    swipeRefreshLayout.isRefreshing = state.isLoading

    if (state.error != null) {
        showLoadingTasksError()
        return
    }
}
```

```
fun render(state: TasksViewState) {
    swipeRefreshLayout.isRefreshing = state.isLoading

    if (state.error != null) {
        showLoadingTasksError()
        return
    }

    if (state.taskActivated) {
        showMessage(getString(R.string.task_marked_active))
    }

    if (state.taskComplete) {
        showMessage(getString(R.string.task_marked_complete))
    }

    if (state.completedTasksCleared) {
        showMessage(getString(R.string.completed_tasks_cleared))
    }
}
```

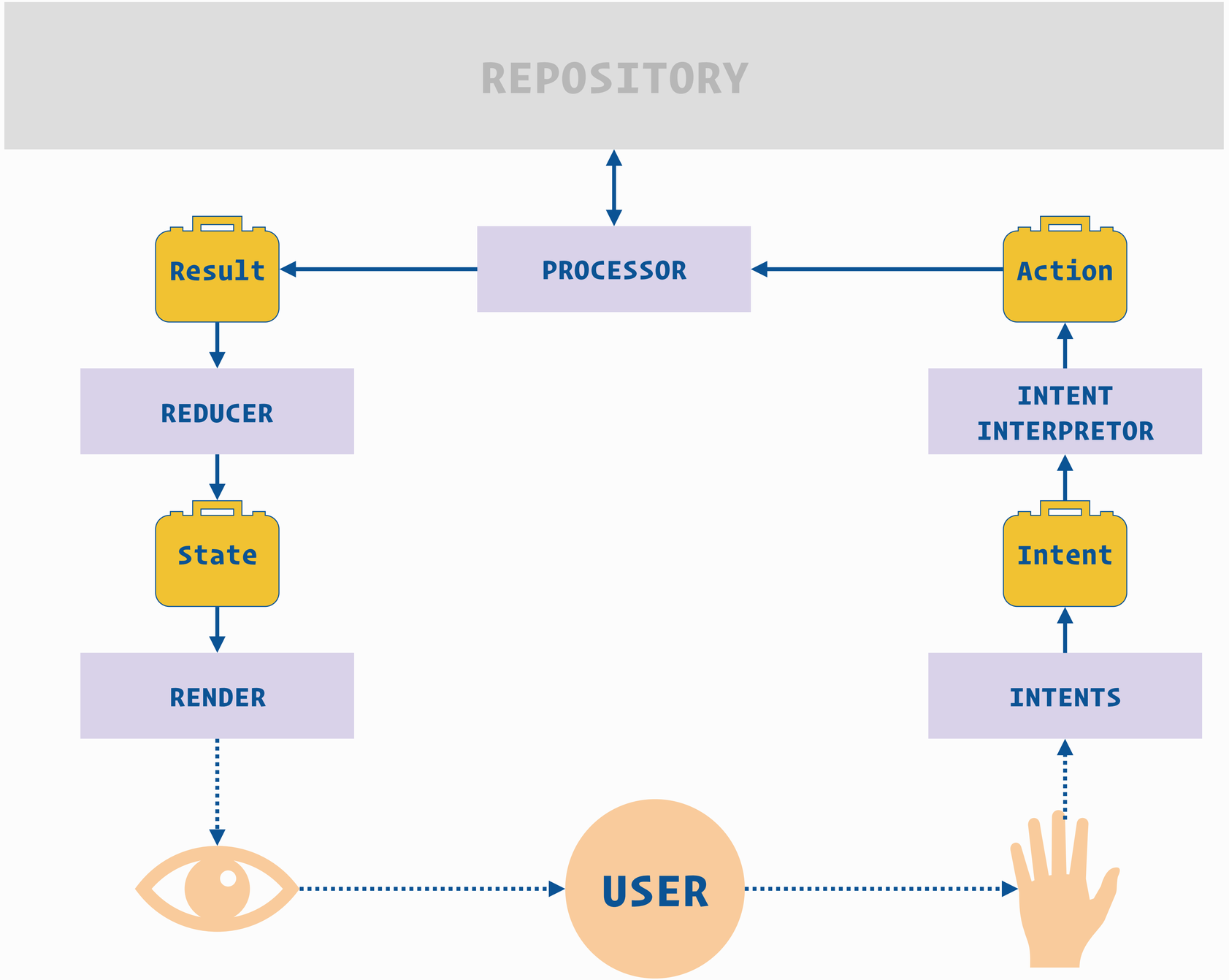
```
        if (state.taskActivated) {
            showMessage(getString(R.string.task_marked_active))
        }

        if (state.taskComplete) {
            showMessage(getString(R.string.task_marked_complete))
        }

        if (state.completedTasksCleared) {
            showMessage(getString(R.string.completed_tasks_cleared))
        }

        if (state.tasks.isEmpty()) {
            when (state.tasksFilterType) {
                ACTIVE_TASKS -> showNoActiveTasks()
                COMPLETED_TASKS -> showNoCompletedTasks()
                ALL_TASKS -> showNoTasks()
            }
        }
    }
```

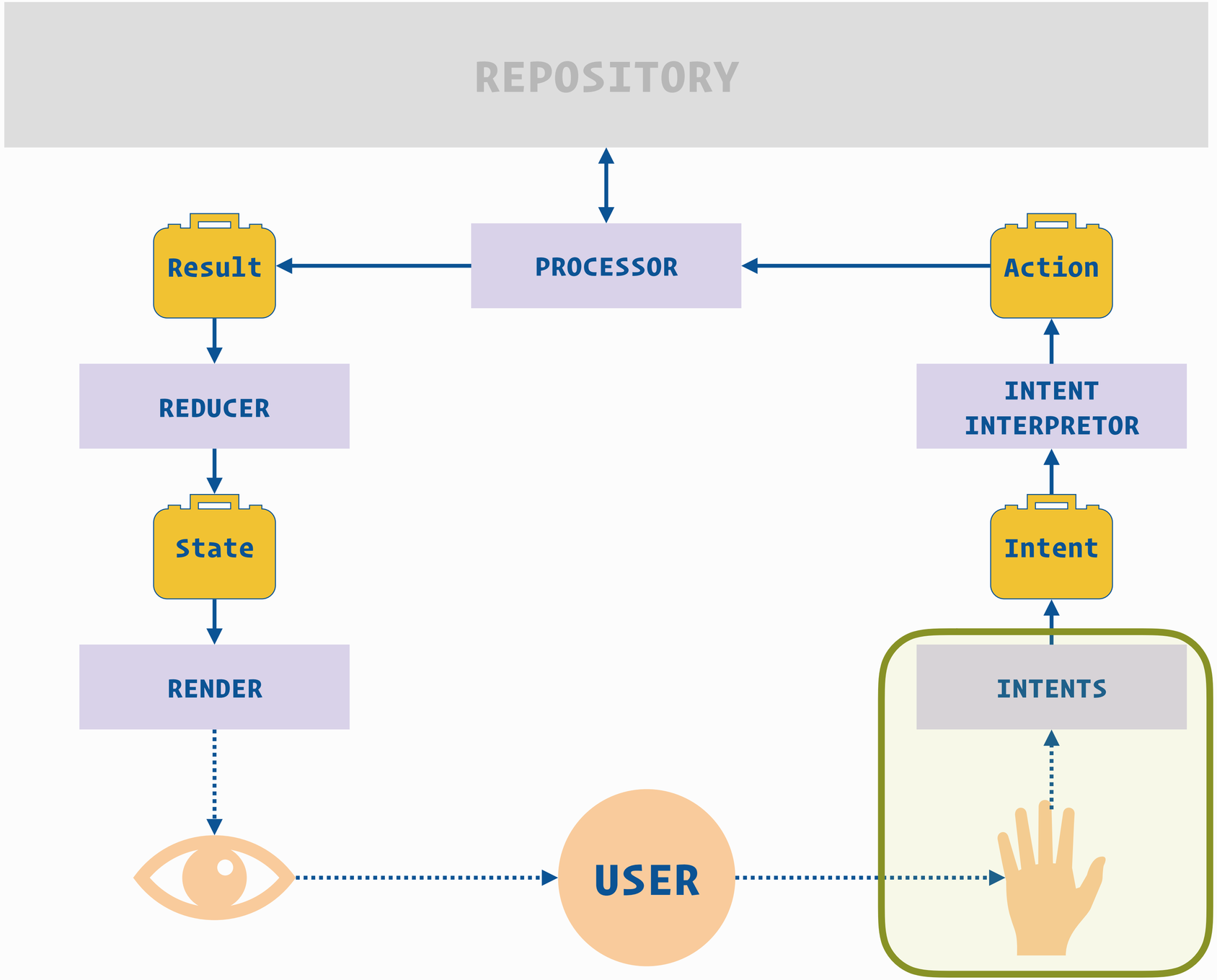
```
            showMessage(getString(R.string.completed_tasks_cleared))
        }

        if (state.tasks.isEmpty()) {
            when (state.tasksFilterType) {
                ACTIVE_TASKS -> showNoActiveTasks()
                COMPLETED_TASKS -> showNoCompletedTasks()
                ALL_TASKS -> showNoTasks()
            }
        } else {
            listAdapter.replaceData(state.tasks)

            tasksView.visibility = View.VISIBLE
            noTasksView.visibility = View.GONE

            when (state.tasksFilterType) {
                ACTIVE_TASKS -> showActiveFilterLabel()
                COMPLETED_TASKS -> showCompletedFilterLabel()
                ALL_TASKS -> showAllFilterLabel()
            }
        }
    }
```

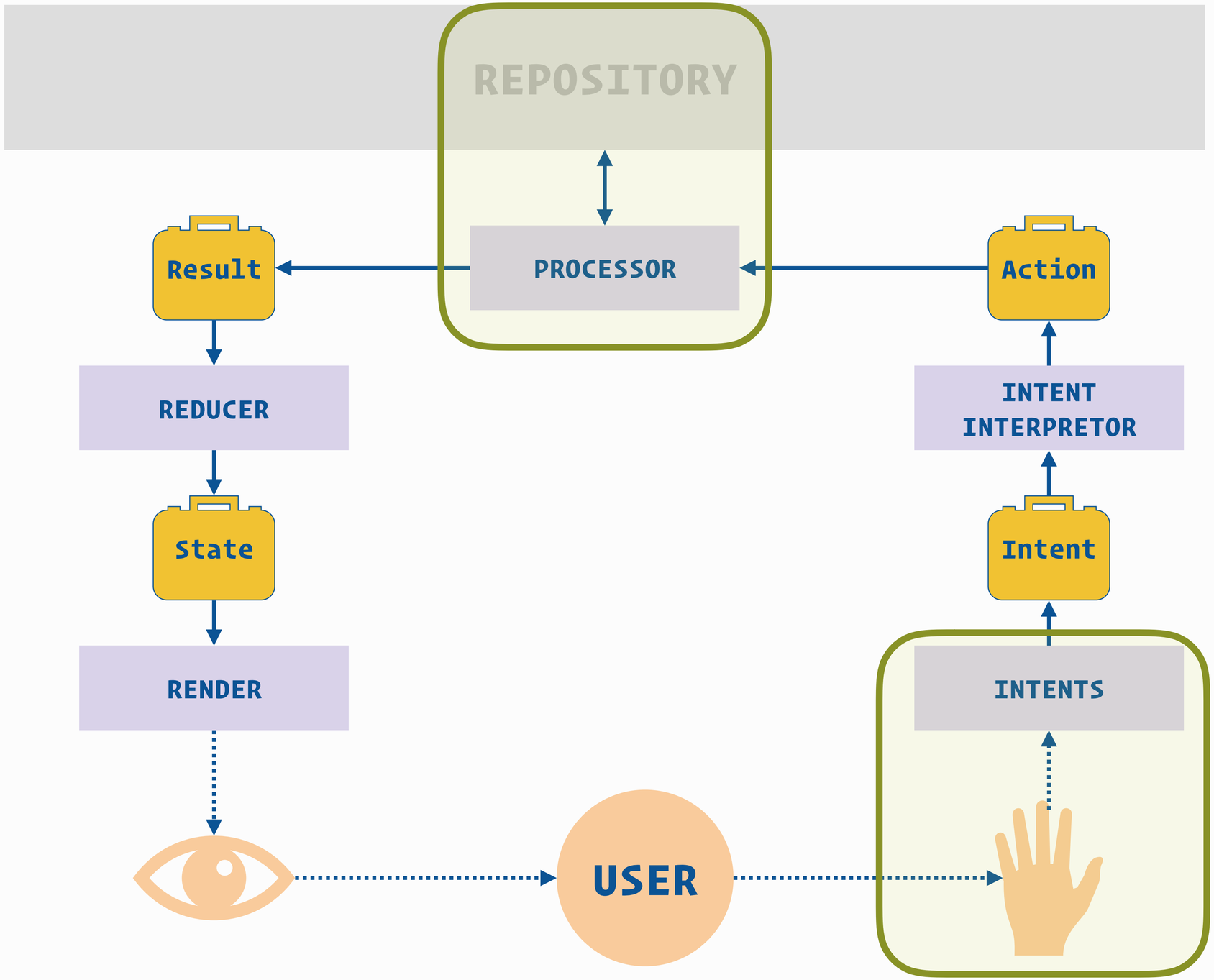
```
fun render(state: TasksViewState) {
    swipeRefreshLayout.isRefreshing = state.isLoading

    if (state.error != null) {
        showLoadingTasksError()
        return
    }

    if (state.taskActivated) {
        showMessage(getString(R.string.task_marked_active))
    }

    if (state.taskComplete) {
        showMessage(getString(R.string.task_marked_complete))
    }

    if (state.completedTasksCleared) {
        showMessage(getString(R.string.completed_tasks_cleared))
    }

    if (state.tasks.isEmpty()) {
        when (state.tasksFilterType) {
            ACTIVE_TASKS -> showNoActiveTasks()
            COMPLETED_TASKS -> showNoCompletedTasks()
            ALL_TASKS -> showNoTasks()
        }
    } else {
        listAdapter.replaceData(state.tasks)

        tasksView.visibility = View.VISIBLE
        noTasksView.visibility = View.GONE

        when (state.tasksFilterType) {
            ACTIVE_TASKS -> showActiveFilterLabel()
            COMPLETED_TASKS -> showCompletedFilterLabel()
            ALL_TASKS -> showAllFilterLabel()
        }
    }
}
```

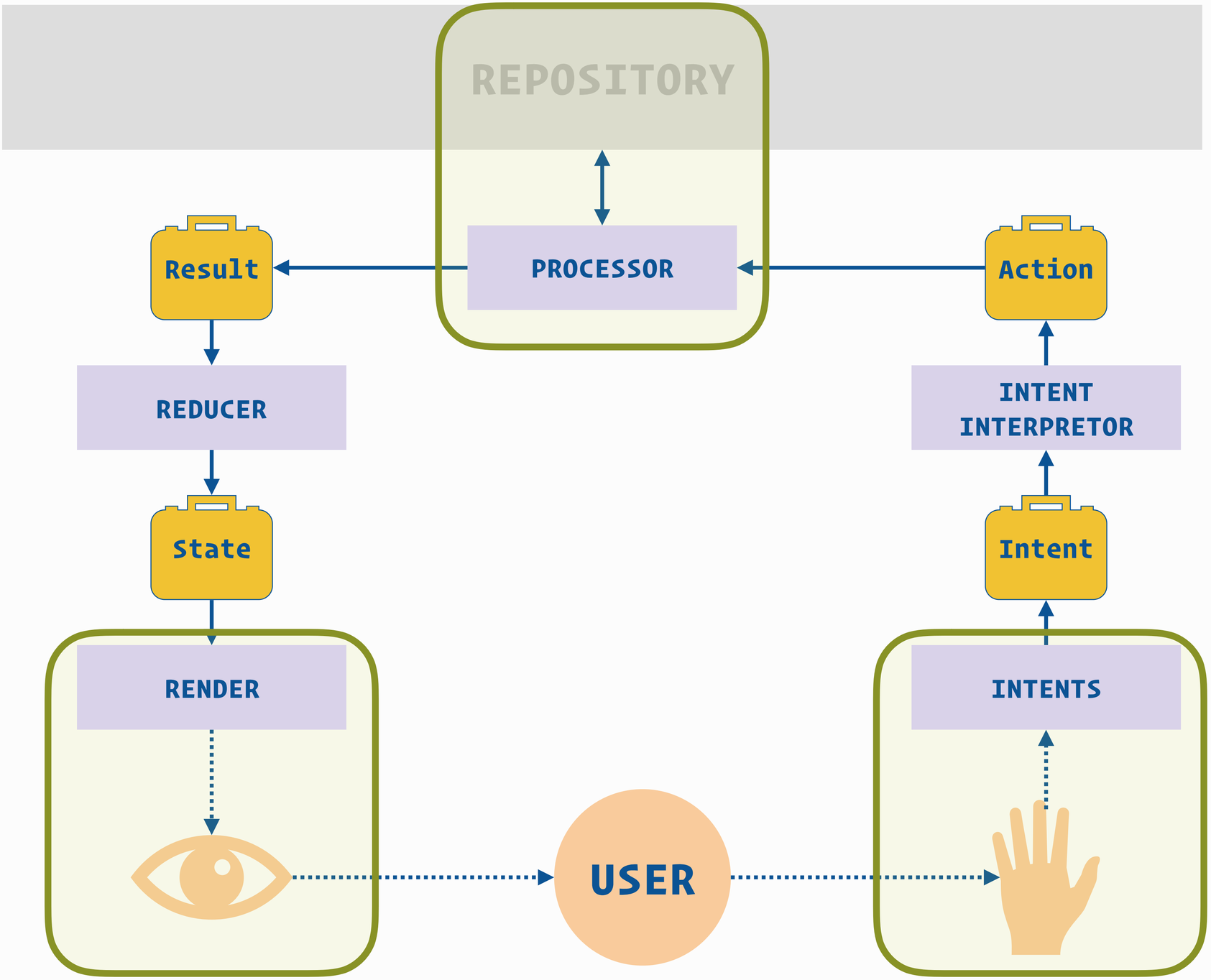

REPOSITORY
Result
PROCESSOR
Action
REDUCER
INTENT
INTERPRETOR
State
Intent
RENDER
INTENTS
USER

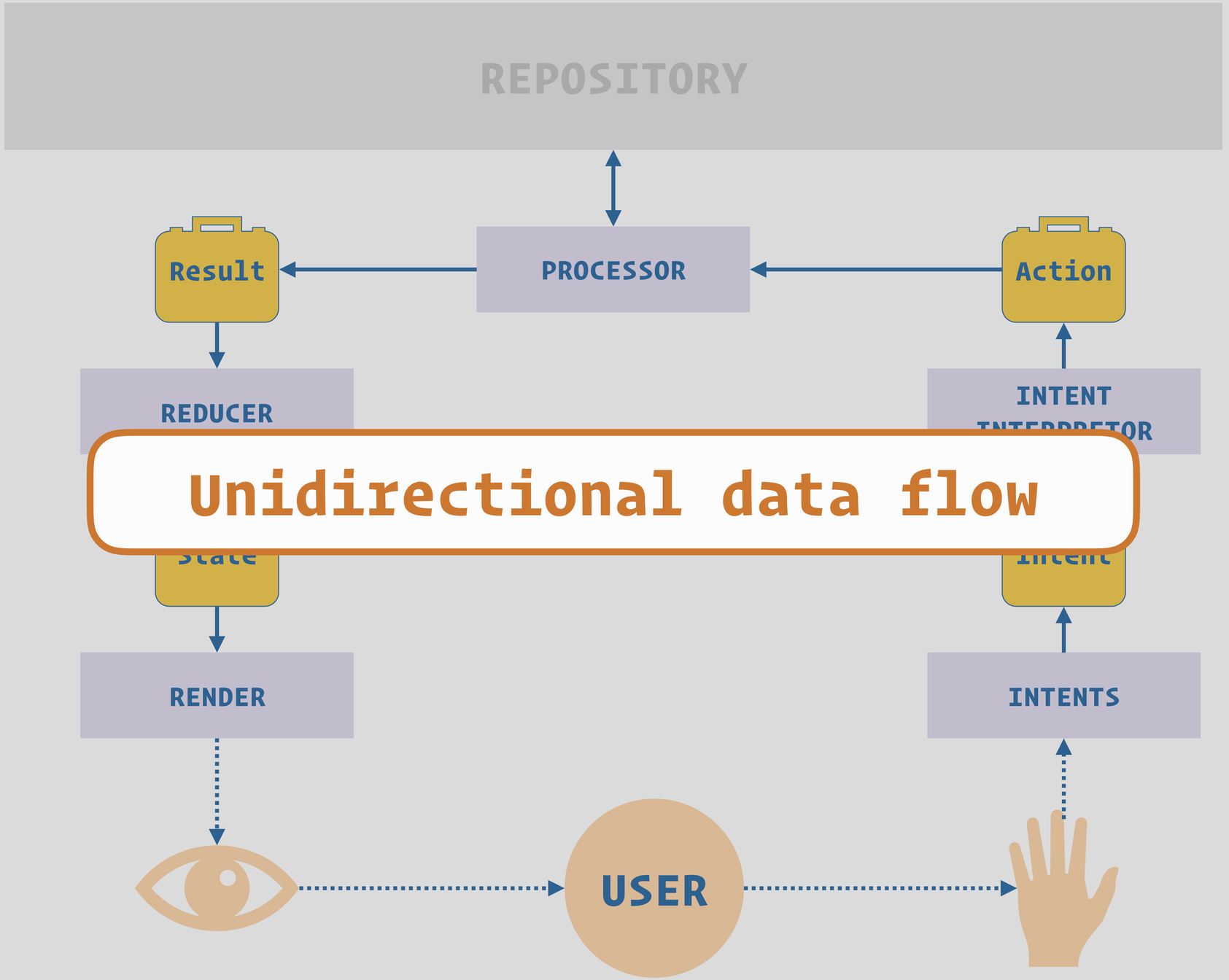
Yo Dawg, I heard you like side effects

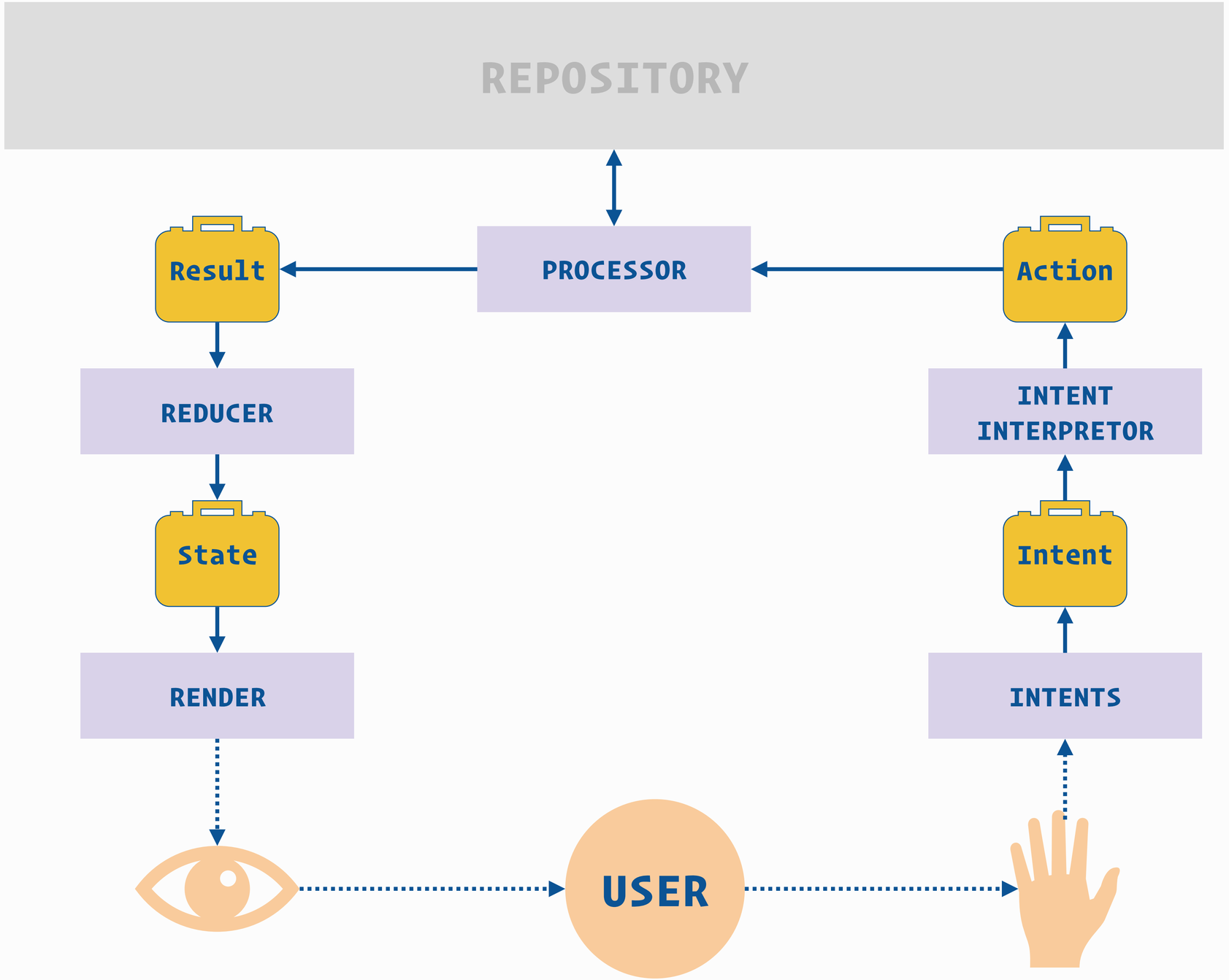

REPOSITORY
Result
PROCESSOR
Action
REDUCER
INTENT
INTERPRETOR
State
Intent
RENDER
INTENTS
USER

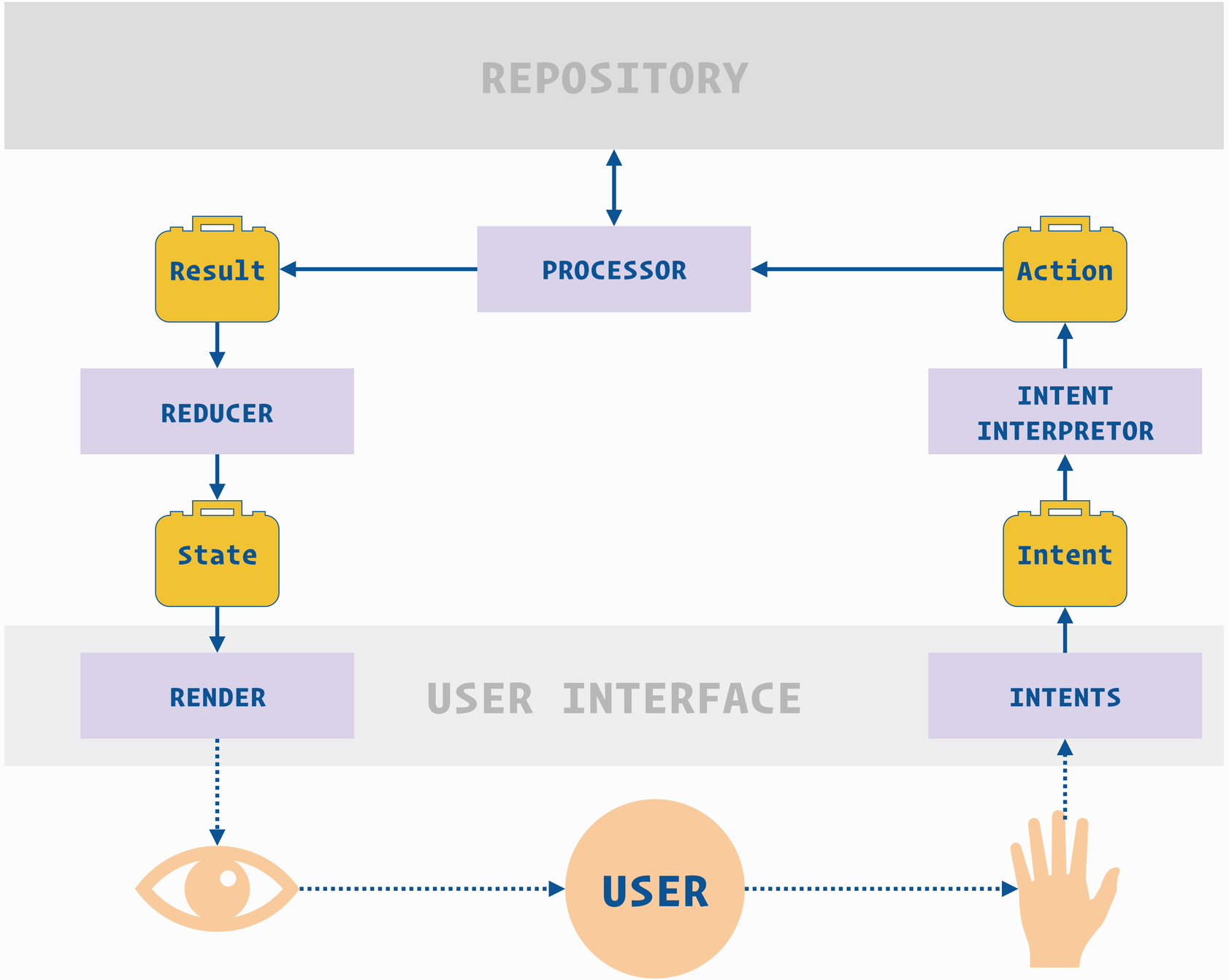

REPOSITORY
Result
PROCESSOR
Action
REDUCER
INTENT
INTERPRETOR
State
Intent
RENDER
INTENTS
USER

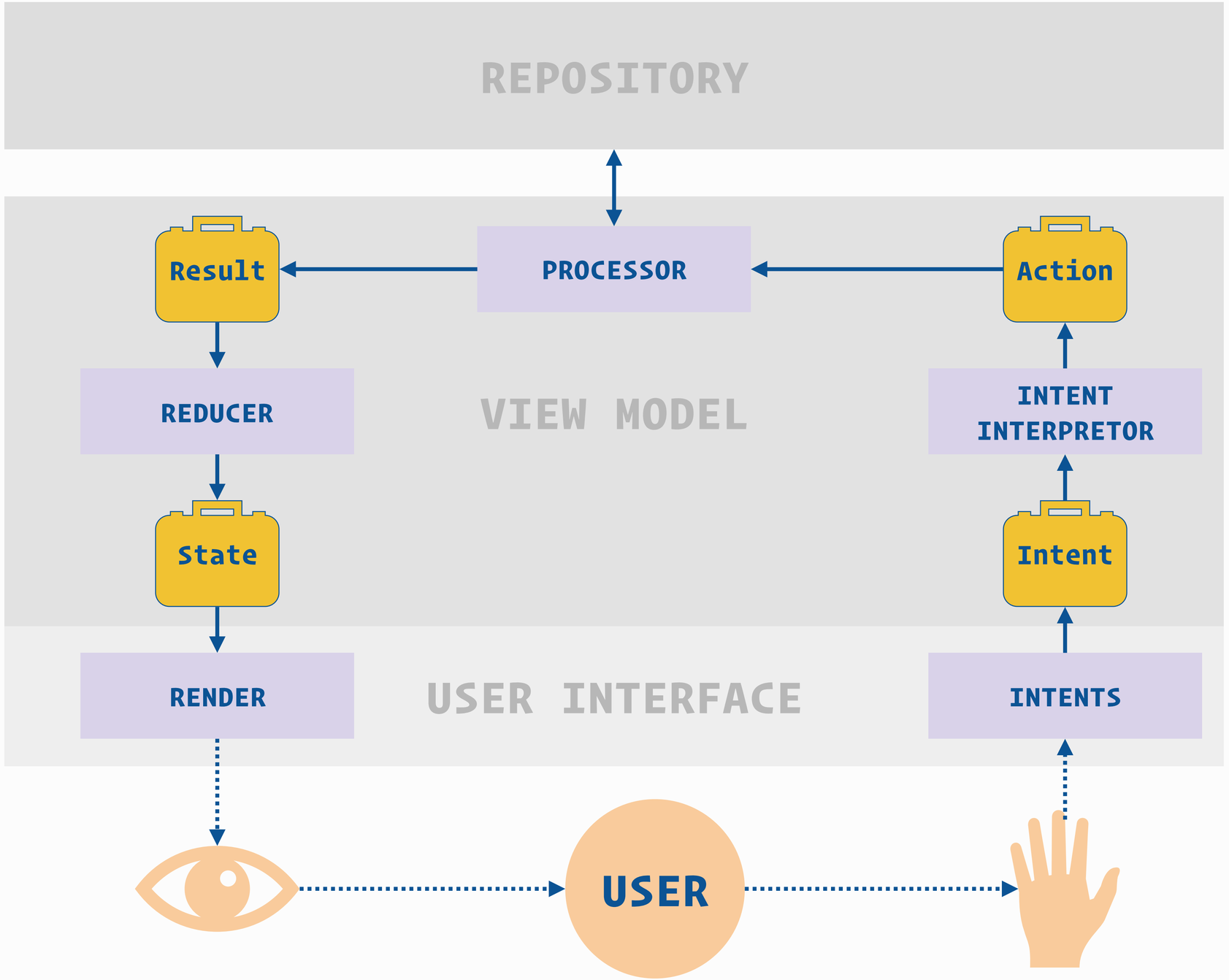

REPOSITORY
PROCESSOR
Result
Action
REDUCER
INTENT
INTERPRETOR
State
Intent
RENDER
INTENTS
USER


REPOSITORY
PROCESSOR
Result
Action
REDUCER
INTENT
INTERPRETOR
State
Intent
RENDER
INTENTS
USER


REPOSITORY
Result
PROCESSOR
Action
REDUCER
INTENT
Unidirectional data flow
RENDER
INTENTS
USER


REPOSITORY
Result
PROCESSOR
Action
REDUCER
INTENT
INTERPRETOR
State
Intent
RENDER
INTENTS
USER


REPOSITORY
Result
PROCESSOR
Action
REDUCER
INTENT
INTERPRETOR
State
Intent
RENDER
USER INTERFACE
INTENTS
USER


REPOSITORY
Result
PROCESSOR
Action
REDUCER
VIEW MODEL
INTENT
INTERPRETOR
State
Intent
RENDER
USER INTERFACE
INTENTS
USER

```
interface TasksUi {
    fun render(state: TasksViewState)

    fun intents(): Observable<TasksIntent>
}


interface TasksViewModel {
    fun processIntents(intents: Observable<TasksIntent>)

    fun states(): Observable<TasksViewState>
}
```

```
interface TasksUi {
    fun render(state: TasksViewState)

    fun intents(): Observable<TasksIntent>
}


interface TasksViewModel {
    fun processIntents(intents: Observable<TasksIntent>)

    fun states(): Observable<TasksViewState>
}
```

```
interface TasksUi {
    fun render(state: TasksViewState)

    fun intents(): Observable<TasksIntent>
}


interface TasksViewModel {
    fun processIntents(intents: Observable<TasksIntent>)

    fun states(): Observable<TasksViewState>
}
```

Mama Lova

User: initialIntent()

We: render()

User: initialIntent()

We: render()

User: activateTask(1)

User: activateTask(2)

User: refresh()

User: initialIntent()

We: render()

User: activateTask(1)

User: activateTask(2)

User: refresh()

Mum: 📞

User: initialIntent()

We: render()

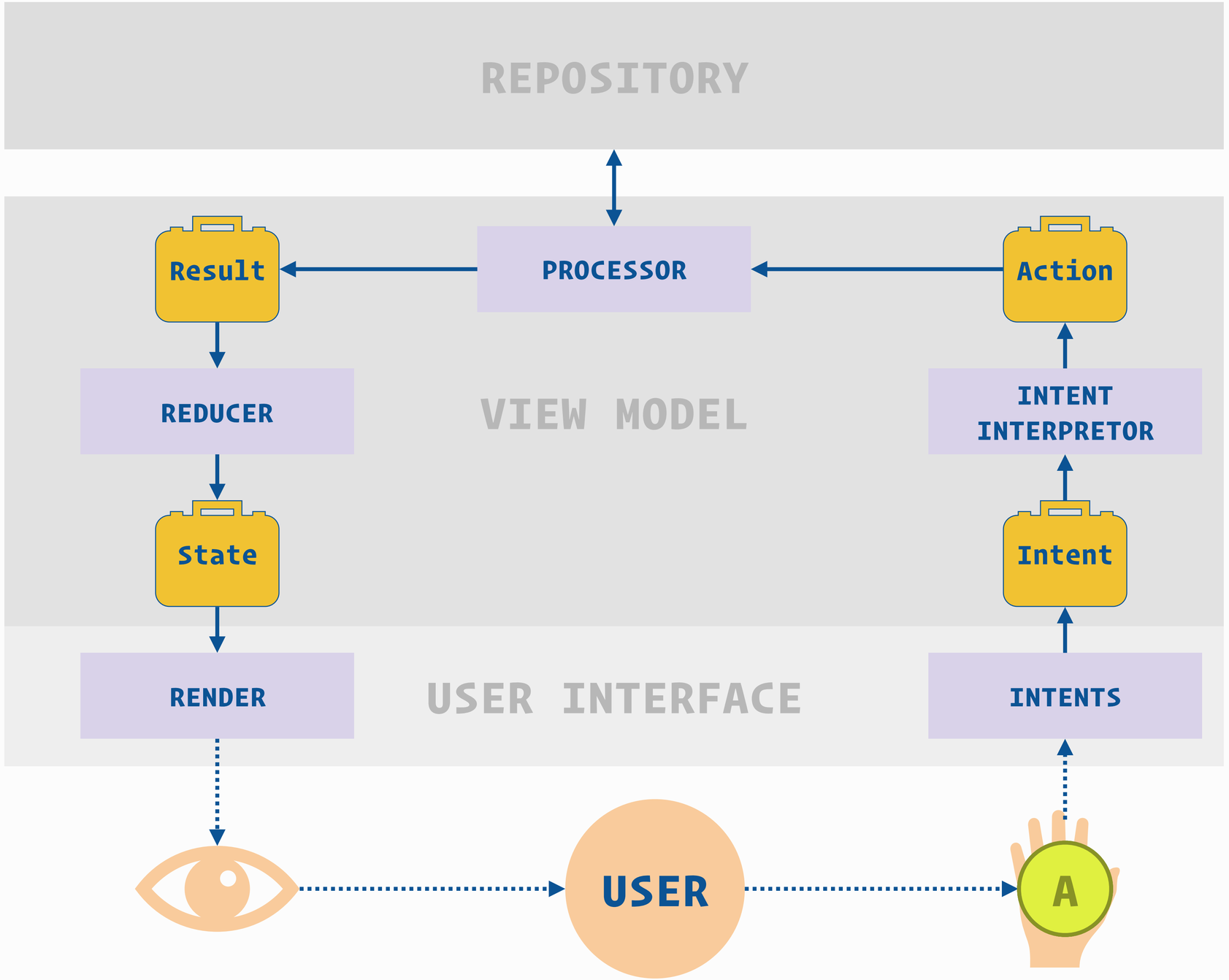
User: activateTask(1)

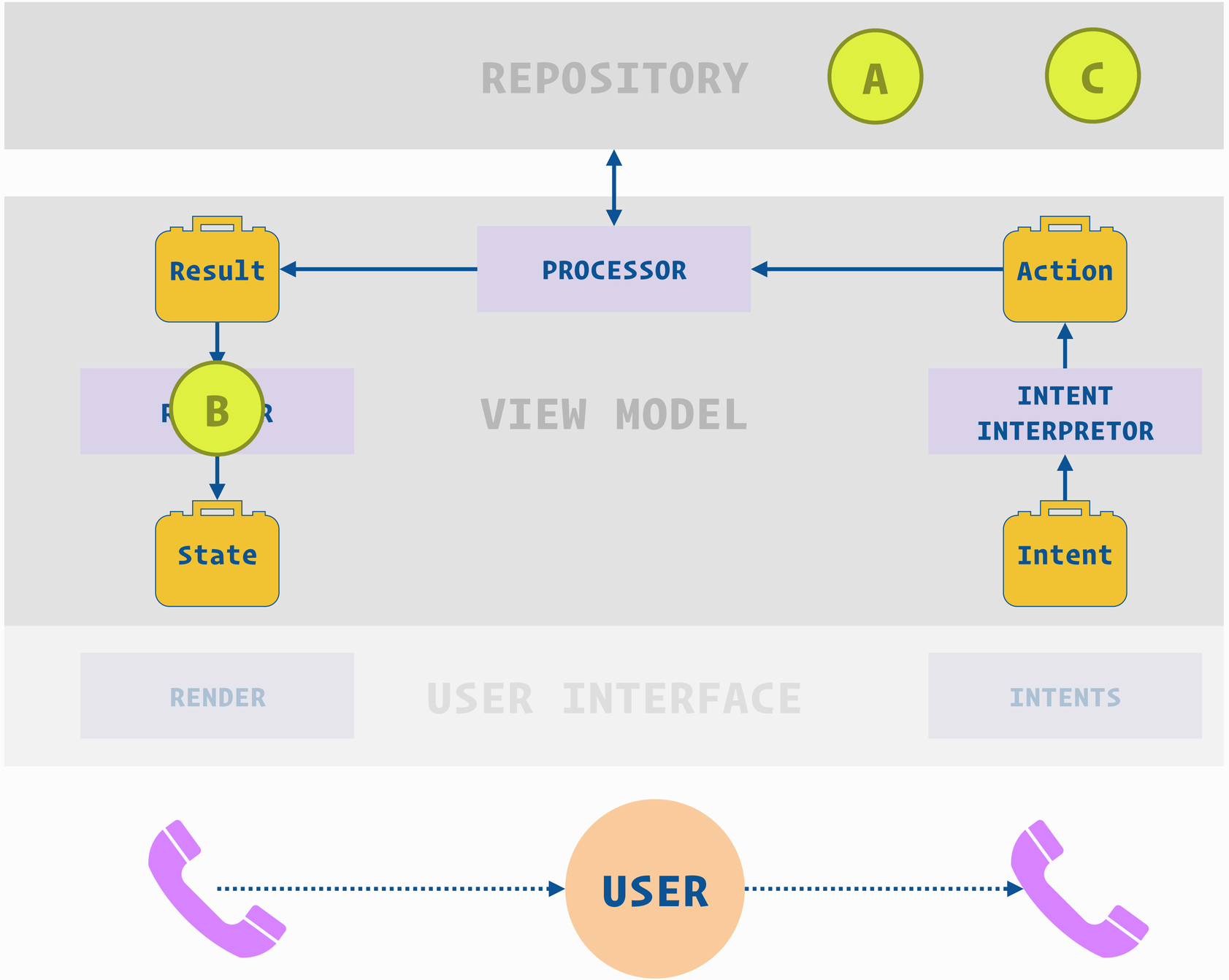
User: activateTask(2)

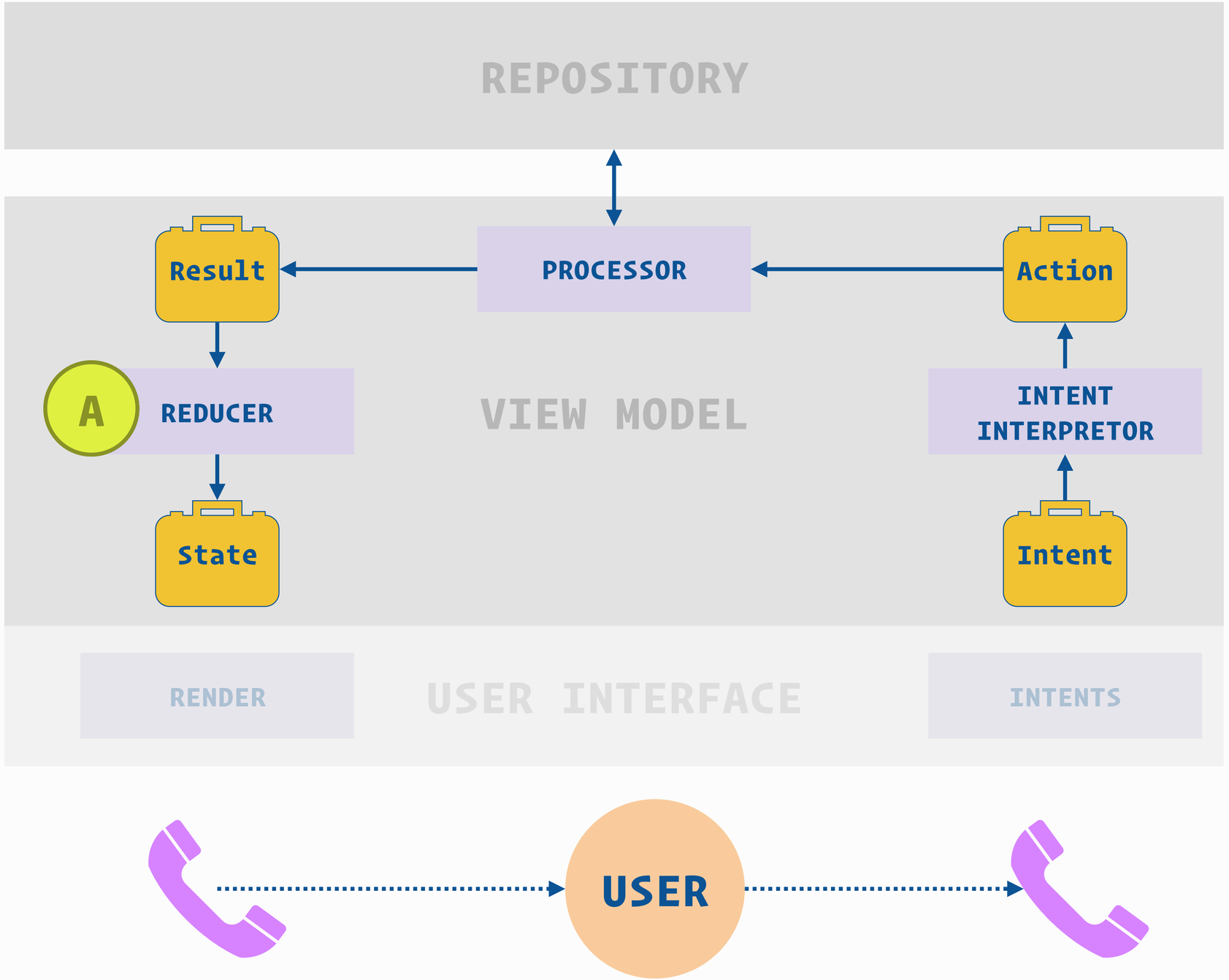
User: refresh()

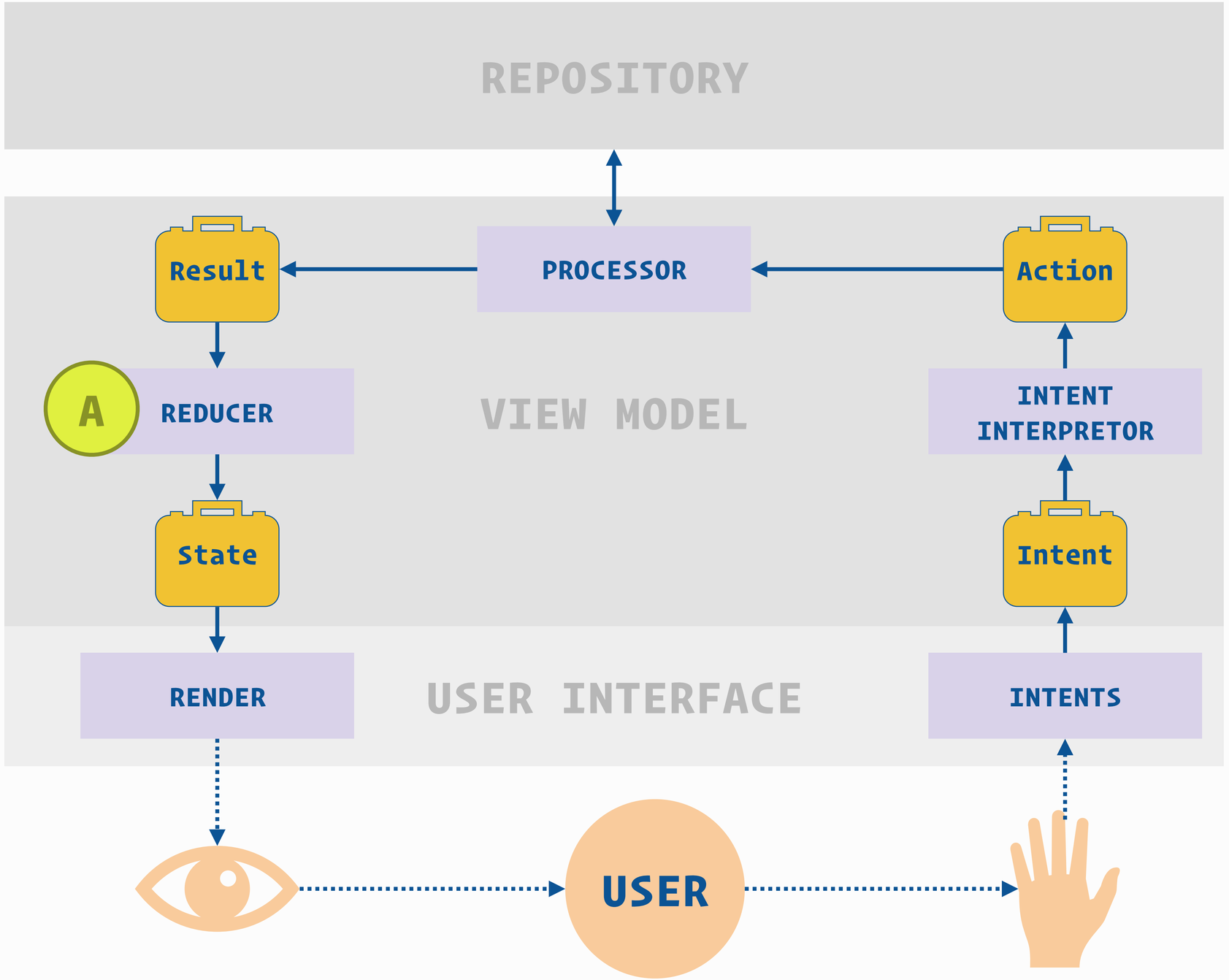
Mum: 📞

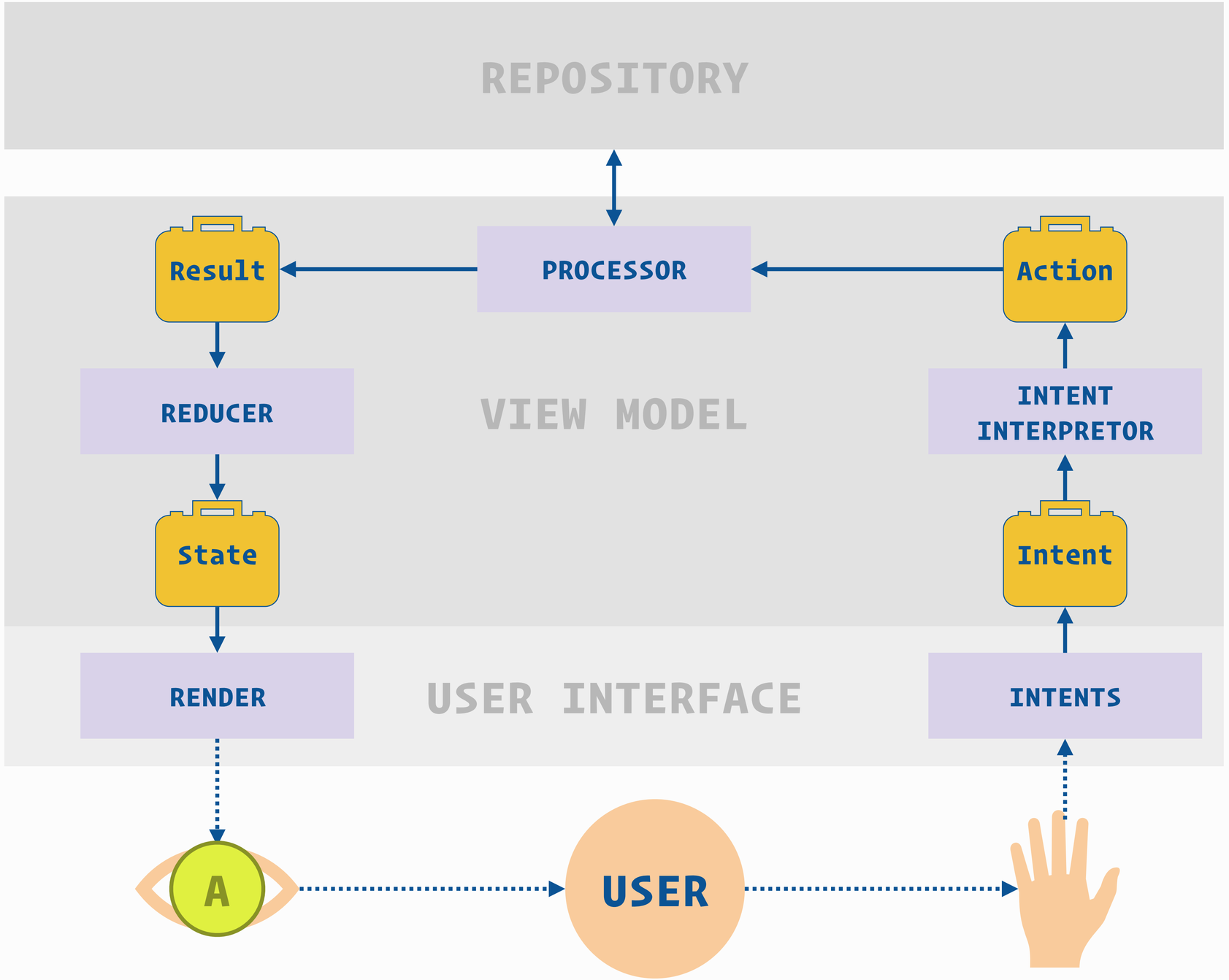
Android: we.*onStop*()


REPOSITORY
Result
PROCESSOR
Action
REDUCER
VIEW MODEL
INTENT
INTERPRETOR
State
Intent
RENDER
USER INTERFACE
INTENTS
USER
A

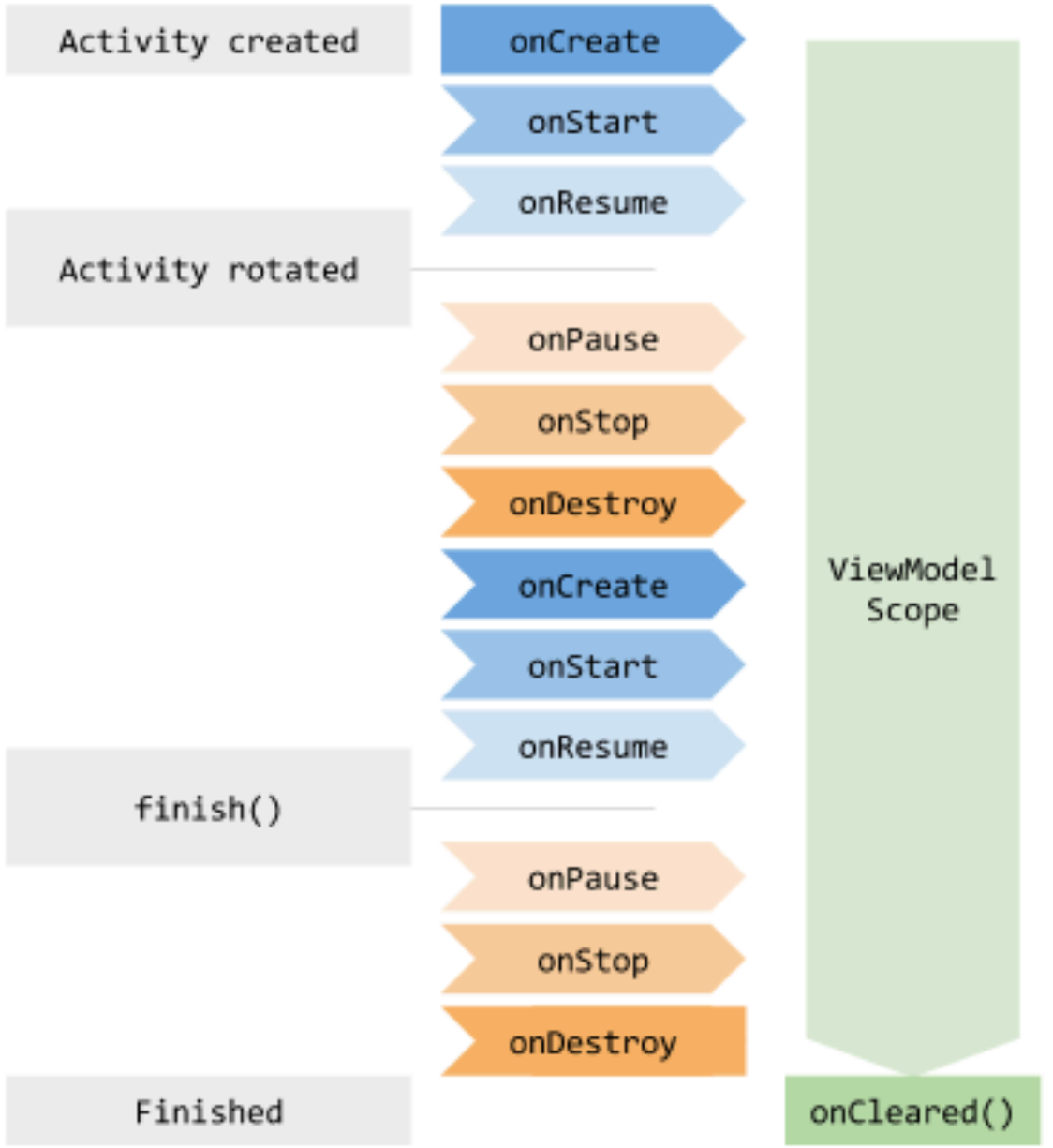

REPOSITORY
A
C
Result
PROCESSOR
Action
B
VIEW MODEL
INTENT
INTERPRETOR
State
Intent
RENDER
USER INTERFACE
INTENTS
USER

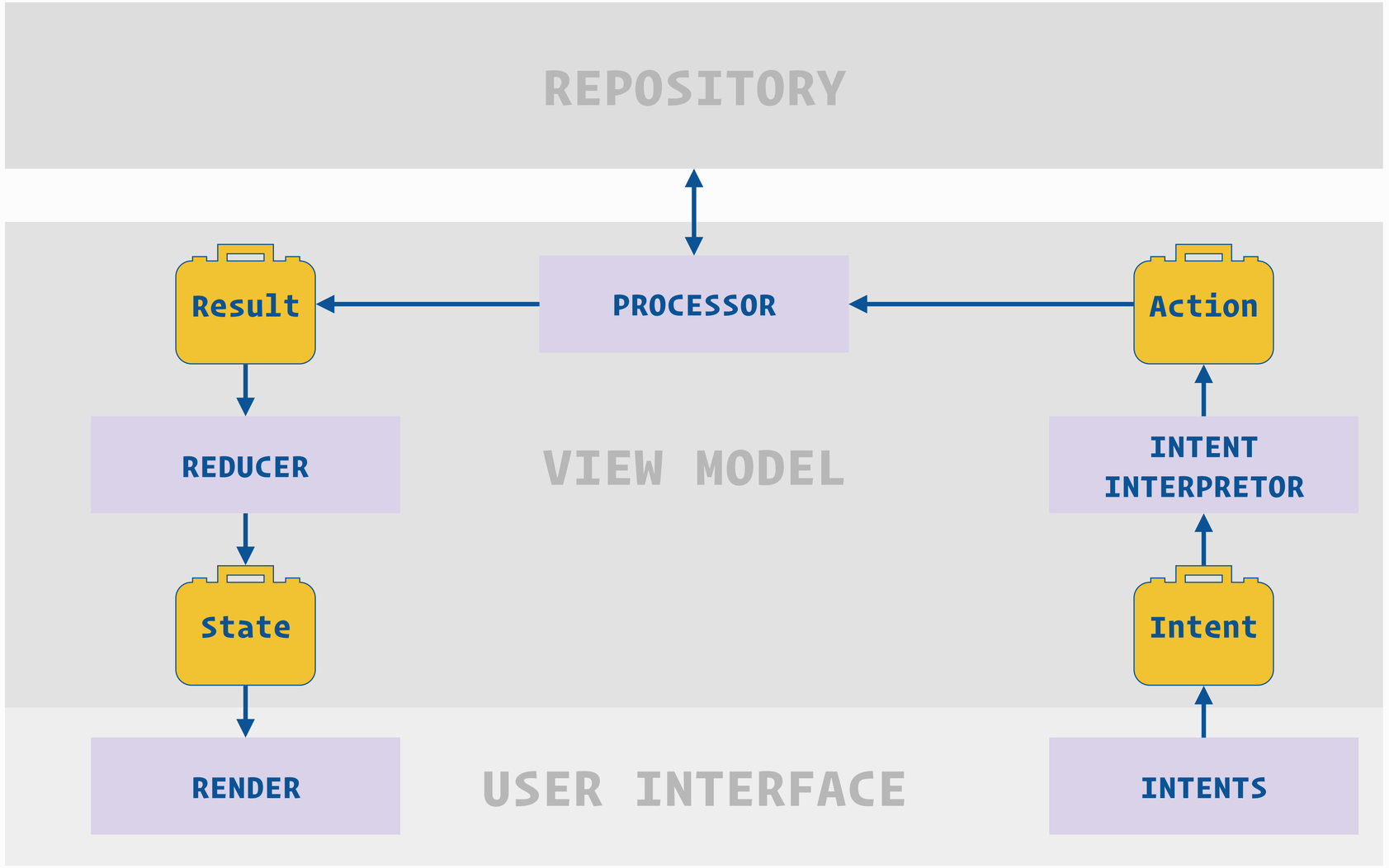

REPOSITORY
Result
PROCESSOR
Action
A
REDUCER
VIEW MODEL
INTENT
INTERPRETOR
State
Intent
RENDER
USER INTERFACE
INTENTS
USER

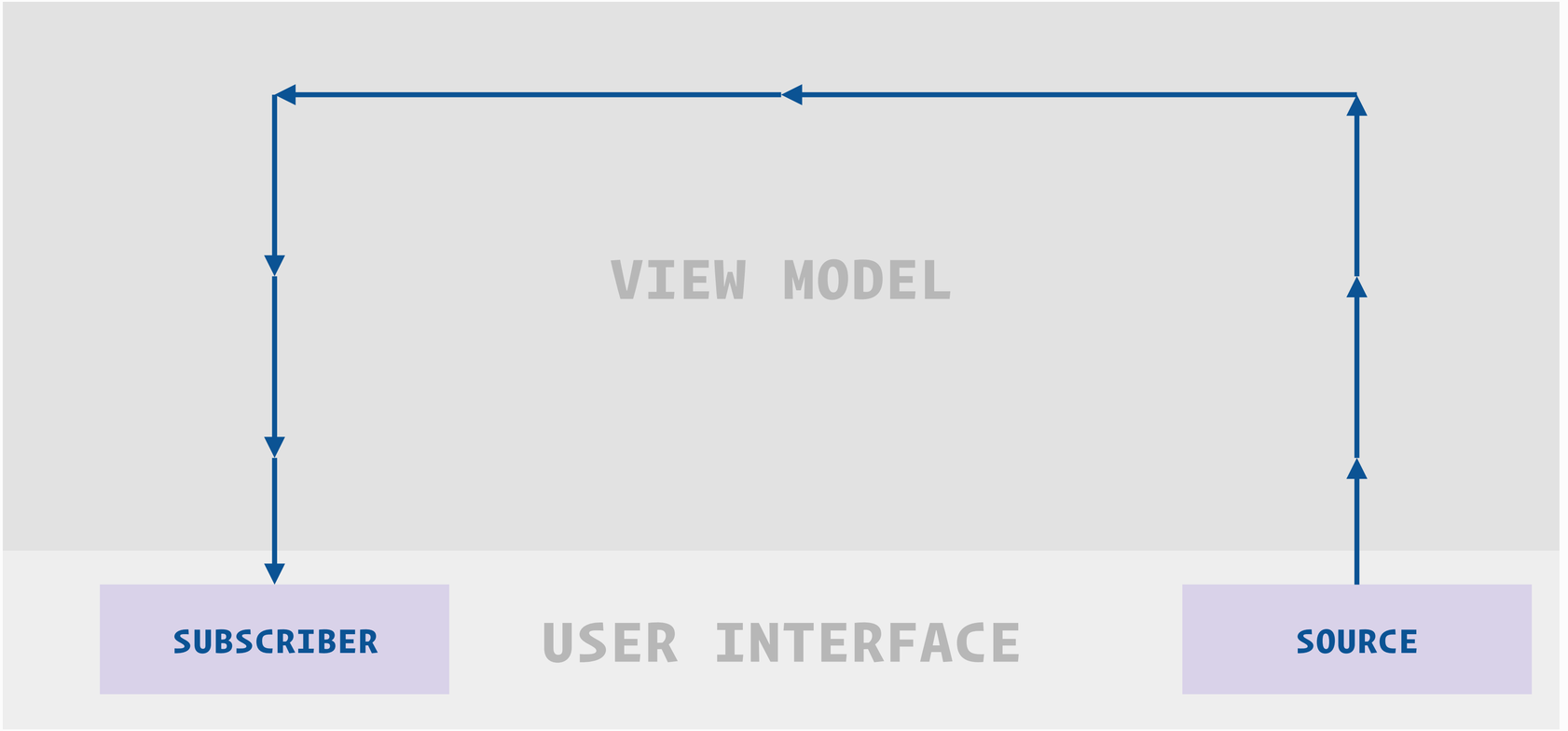

REPOSITORY
Result
PROCESSOR
Action
A
REDUCER
VIEW MODEL
INTENT
INTERPRETOR
State
Intent
RENDER
USER INTERFACE
INTENTS
USER

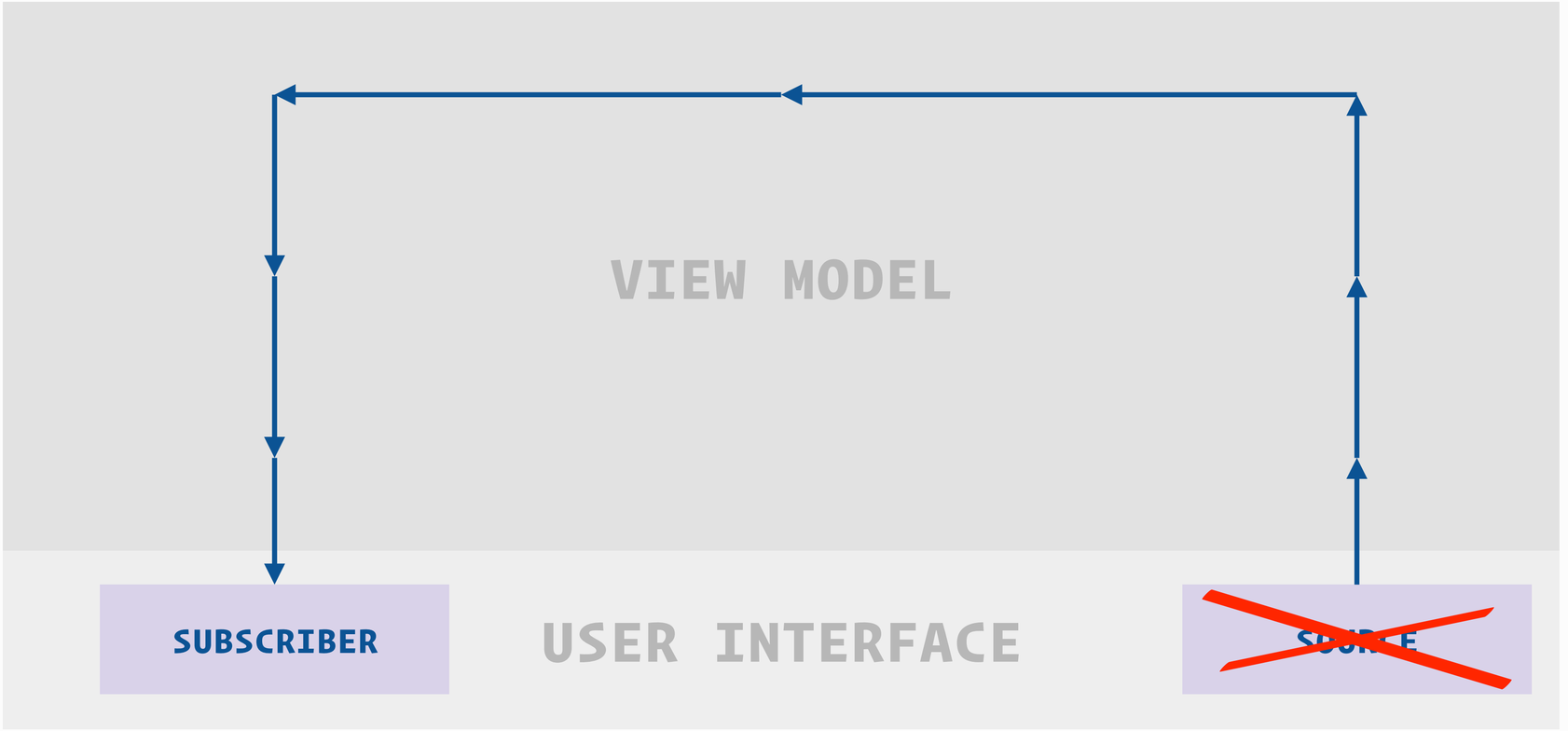

REPOSITORY
Result
PROCESSOR
Action
REDUCER
VIEW MODEL
INTENT
INTERPRETOR
State
Intent
RENDER
USER INTERFACE
INTENTS
A
USER

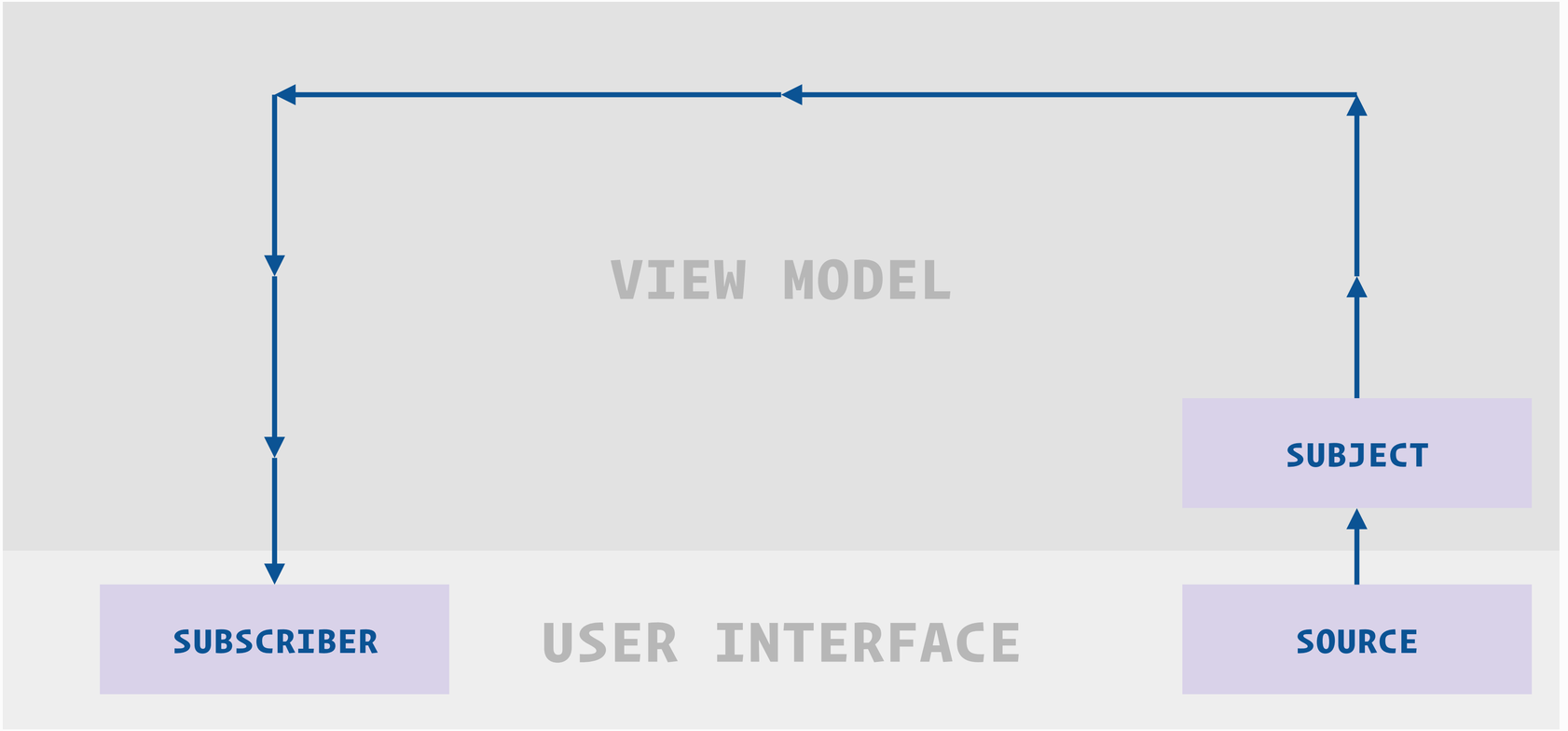

Config change everywhere

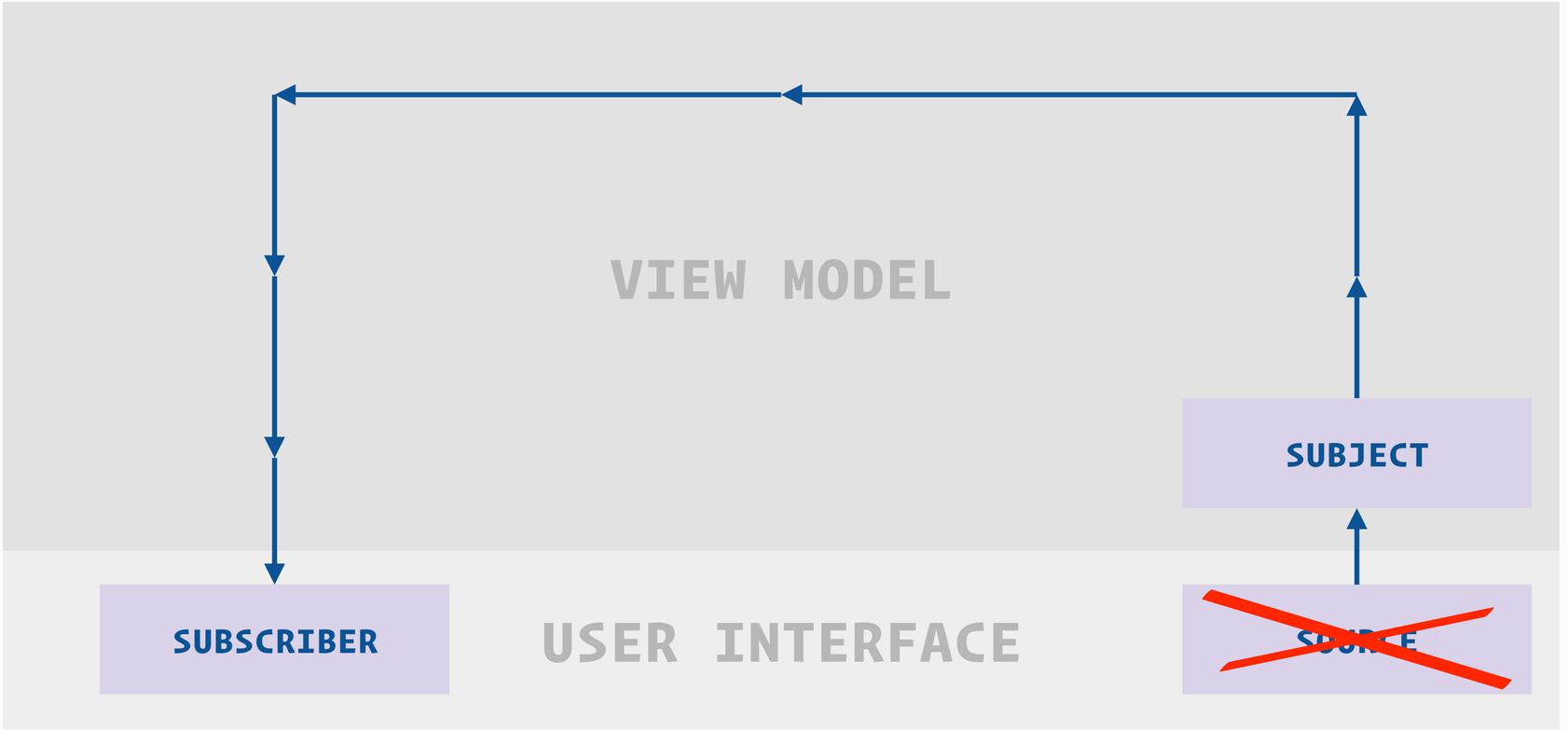

Activity created
onCreate
onStart
onResume
Activity rotated
onPause
onStop
onDestroy
onCreate
onStart
onResume
finish()
onPause
onStop
onDestroy
Finished
ViewModel Scope
onCleared()

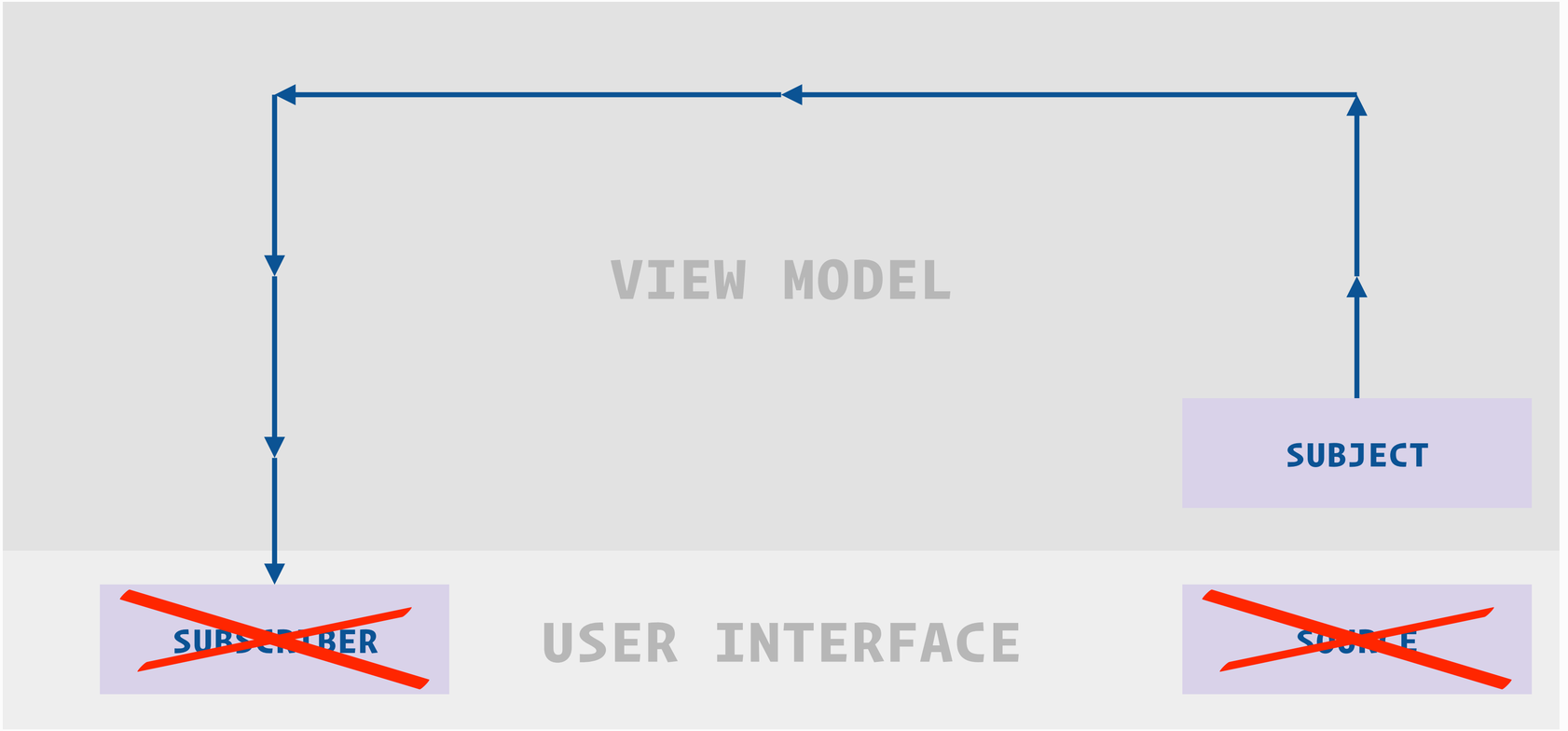

REPOSITORY
Result
PROCESSOR
Action
REDUCER
VIEW MODEL
INTENT
INTERPRETOR
State
Intent
RENDER
USER INTERFACE
INTENTS

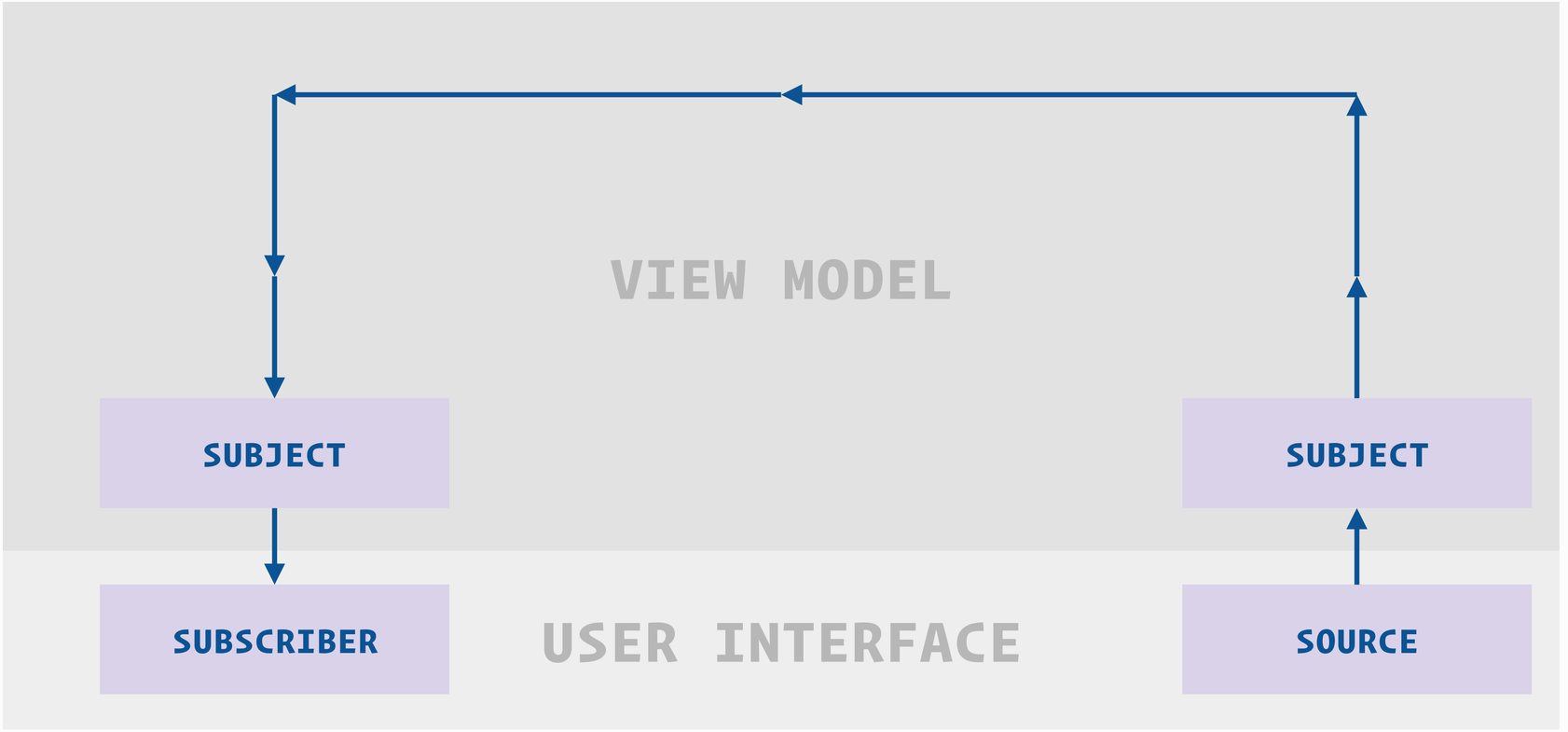

VIEW MODEL
SUBSCRIBER
USER INTERFACE
SOURCE

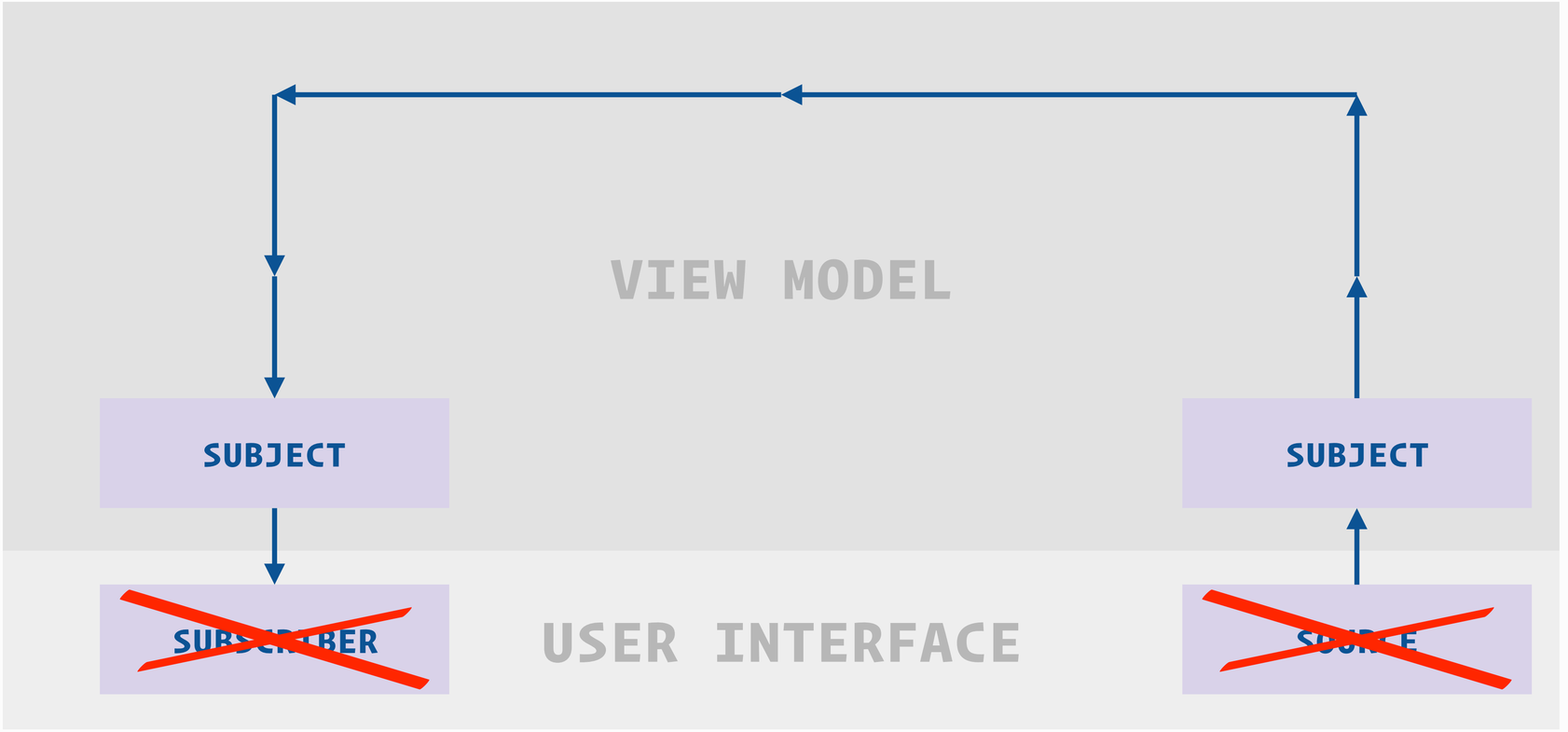

VIEW MODEL
SUBSCRIBER
USER INTERFACE

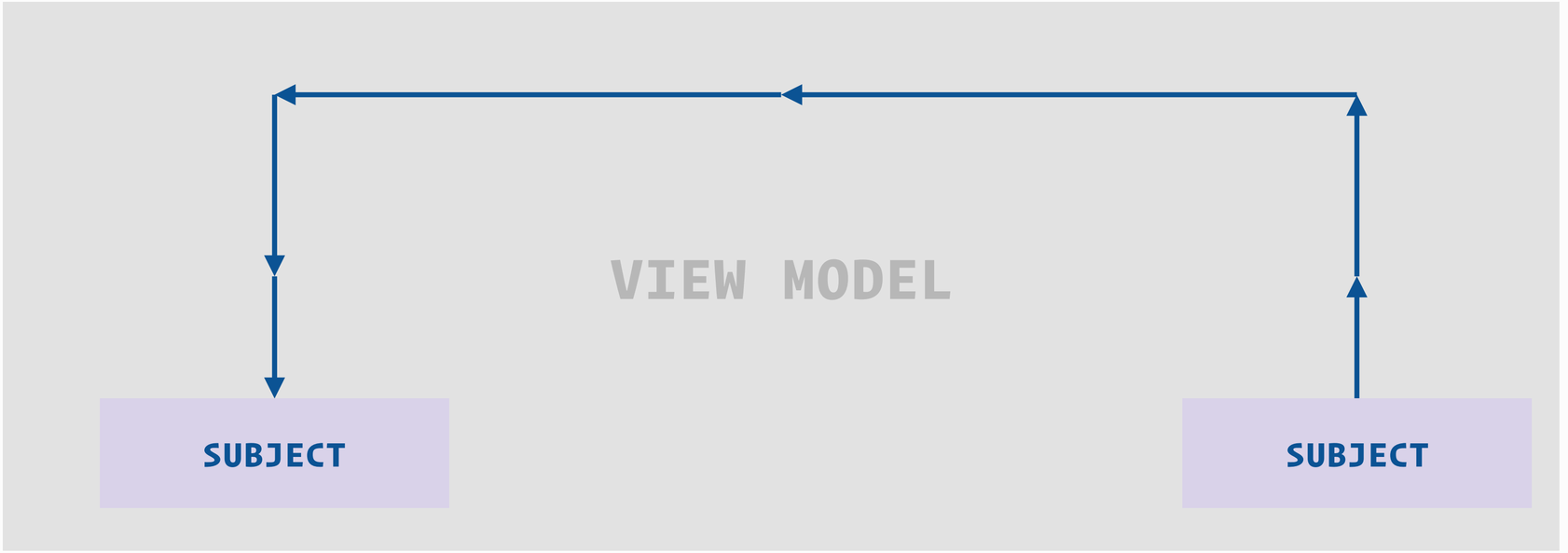

VIEW MODEL
SUBJECT
SUBSCRIBER
USER INTERFACE
SOURCE

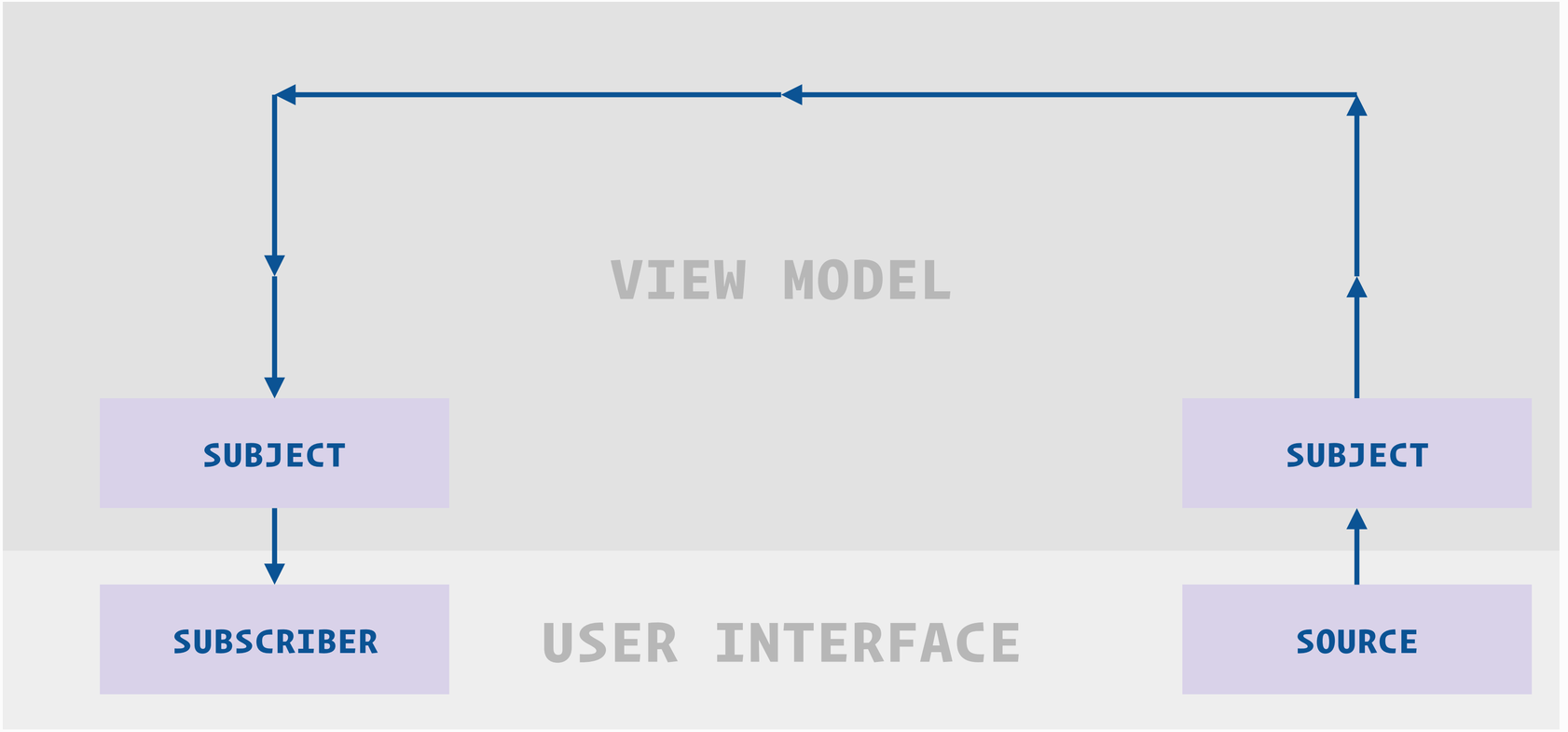

VIEW MODEL
SUBJECT
SUBSCRIBER
USER INTERFACE


VIEW MODEL
SUBJECT
USER INTERFACE


VIEW MODEL
SUBJECT
SUBJECT
SUBSCRIBER
USER INTERFACE
SOURCE


VIEW MODEL
SUBJECT
SUBJECT
USER INTERFACE


VIEW MODEL
SUBJECT
SUBJECT


VIEW MODEL
SUBJECT
SUBJECT
SUBSCRIBER
USER INTERFACE
SOURCE

```
class TasksViewModel : ViewModel() {
  fun processIntents(intents: Observable<TasksIntent>) {
  }

  fun states(): Observable<TasksViewState> {
  }
}
```

```
class TasksViewModel : ViewModel() {
  val intentsSubject: PublishSubject<TasksIntent> = PublishSubject.create()

  fun processIntents(intents: Observable<TasksIntent>) {
    intents.subscribe(intentsSubject)
  }

  fun states(): Observable<TasksViewState> {
  }
}
```

```
class TasksViewModel : ViewModel() {
  val intentsSubject: PublishSubject<TasksIntent> = PublishSubject.create()
  val statesSubject: PublishSubject<TasksViewState> = PublishSubject.create()

  fun processIntents(intents: Observable<TasksIntent>) {
    intents.subscribe(intentsSubject)
  }

  fun states(): Observable<TasksViewState> {
    return statesSubject
  }
}
```

```
class TasksViewModel : ViewModel() {
  val intentsSubject: PublishSubject<TasksIntent> = PublishSubject.create()
  val statesSubject: PublishSubject<TasksViewState> = PublishSubject.create()

  init {
    intentsSubject
        .map { intent -> actionFromIntent(intent) }
        .compose(actionProcessor)
        .scan(TasksViewState.default(), reducer)
  }

  fun processIntents(intents: Observable<TasksIntent>) {
    intents.subscribe(intentsSubject)
  }

  fun states(): Observable<TasksViewState> {
    return statesSubject
  }
}
```

```
class TasksViewModel : ViewModel() {
  val intentsSubject: PublishSubject<TasksIntent> = PublishSubject.create()
  val statesSubject: PublishSubject<TasksViewState> = PublishSubject.create()

  init {
    intentsSubject
        .map { intent -> actionFromIntent(intent) }
        .compose(actionProcessor)
        .scan(TasksViewState.default(), reducer)
        .subscribe(statesSubject)
  }

  fun processIntents(intents: Observable<TasksIntent>) {
    intents.subscribe(intentsSubject)
  }

  fun states(): Observable<TasksViewState> {
    return statesSubject
  }
}
```

```
class TasksViewModel : ViewModel() {
  val intentsSubject: PublishSubject<TasksIntent> = PublishSubject.create()
  val statesSubject: PublishSubject<TasksViewState> = PublishSubject.create()

  init {
    intentsSubject
        .map { intent -> actionFromIntent(intent) }
        .compose(actionProcessor)
        .scan(TasksViewState.default(), reducer)
        .subscribe(statesSubject)
  }

  fun processIntents(intents: Observable<TasksIntent>) {
    intents.subscribe(intentsSubject)
  }

  fun states(): Observable<TasksViewState> {
    return statesSubject
  }
}
```

```
class TasksFragment : Fragment() {
  fun intents(): Observable<TasksIntent> { /***/ }

  fun render(state: TasksViewState) { /***/ }
}
```

```
class TasksFragment : Fragment() {
  private val viewModel: TasksViewModel by lazy(NONE) {
    ViewModelProviders.of(this).get(TasksViewModel::class.java)
  }

  fun intents(): Observable<TasksIntent> { /***/ }

  fun render(state: TasksViewState) { /***/ }
}
```

```
class TasksFragment : Fragment() {
  private val viewModel: TasksViewModel by lazy(NONE) {
    ViewModelProviders.of(this).get(TasksViewModel::class.java)
  }

  fun intents(): Observable<TasksIntent> { /***/ }

  fun render(state: TasksViewState) { /***/ }

  override fun onStart() {
    viewModel.states().subscribe(this::render)
  }
}
```

```
class TasksFragment : Fragment() {
  private val viewModel: TasksViewModel by lazy(NONE) {
    ViewModelProviders.of(this).get(TasksViewModel::class.java)
  }
  private val disposables = CompositeDisposable()

  fun intents(): Observable<TasksIntent> { /***/ }

  fun render(state: TasksViewState) { /***/ }

  override fun onStart() {
    disposables.add(viewModel.states().subscribe(this::render))
  }
}
```

```
class TasksFragment : Fragment() {
  private val viewModel: TasksViewModel by lazy(NONE) {
    ViewModelProviders.of(this).get(TasksViewModel::class.java)
  }
  private val disposables = CompositeDisposable()

  fun intents(): Observable<TasksIntent> { /***/ }

  fun render(state: TasksViewState) { /***/ }

  override fun onStart() {
    disposables.add(viewModel.states().subscribe(this::render))
    viewModel.processIntents(intents())
  }
}
```

```
class TasksFragment : Fragment() {
  private val viewModel: TasksViewModel by lazy(NONE) {
    ViewModelProviders.of(this).get(TasksViewModel::class.java)
  }
  private val disposables = CompositeDisposable()

  fun intents(): Observable<TasksIntent> { /***/ }

  fun render(state: TasksViewState) { /***/ }

  override fun onStart() {
    disposables.add(viewModel.states().subscribe(this::render))
    viewModel.processIntents(intents())
  }

  override fun onStop() {
    disposables.dispose()
    super.onStop()
  }
}
```

```
class TasksFragment : Fragment() {
  private val viewModel: TasksViewModel by lazy(NONE) {
    ViewModelProviders.of(this).get(TasksViewModel::class.java)
  }
  private val disposables = CompositeDisposable()

  fun intents(): Observable<TasksIntent> { /***/ }

  fun render(state: TasksViewState) { /***/ }

  override fun onStart() {
    disposables.add(viewModel.states().subscribe(this::render))
    viewModel.processIntents(intents())
  }

  override fun onStop() {
    disposables.dispose()
    super.onStop()
  }
}
```

```
class TasksFragment : Fragment() {
  private val viewModel: TasksViewModel by lazy(NONE) {
    ViewModelProviders.of(this).get(TasksViewModel::class.java)
  }
  private val disposables = CompositeDisposable()

  fun intents(): Observable<TasksIntent> { /***/ }

  fun render(state: TasksViewState) { /***/ }

  override fun onStart() {
    disposables.add(viewModel.states().subscribe(this::render))
    viewModel.processIntents(intents())
  }

  override fun onStop() {
    disposables.dispose()
    super.onStop()
  }
}
```

```
class TasksFragment
  fun intents(): Observable<TasksIntent> {
    return Observable.merge(initialIntent(),
                            refreshIntent(),
                            completeTaskIntent(),
                            activateTaskIntent(),
                            clearCompletedTaskIntent(),
                            changeFilterIntent())
  }
}
```

```
class TasksFragment
  fun intents(): Observable<TasksIntent> {
    return Observable.merge(initialIntent(),
                            refreshIntent(),
                            completeTaskIntent(),
                            activateTaskIntent(),
                            clearCompletedTaskIntent(),
                            changeFilterIntent())
  }
}

fun actionFromIntent(intent: TasksIntent): TasksAction =
    when (intent) {
      is InitialIntent -> LoadAndFilterTasksAction
      /***/
    }
```

```
private fun compose(): Observable<TasksViewState> {
    return intentsSubject
            .map { this.actionFromIntent(it) }
            .compose(actionProcessorHolder.actionProcessor)
            .scan(TasksViewState.idle(), reducer)
}
```

```
private fun compose(): Observable<TasksViewState> {
    return intentsSubject
            .scan(initialIntentFilter)
            .map { this.actionFromIntent(it) }
            .compose(actionProcessorHolder.actionProcessor)
            .scan(TasksViewState.idle(), reducer)
}
```

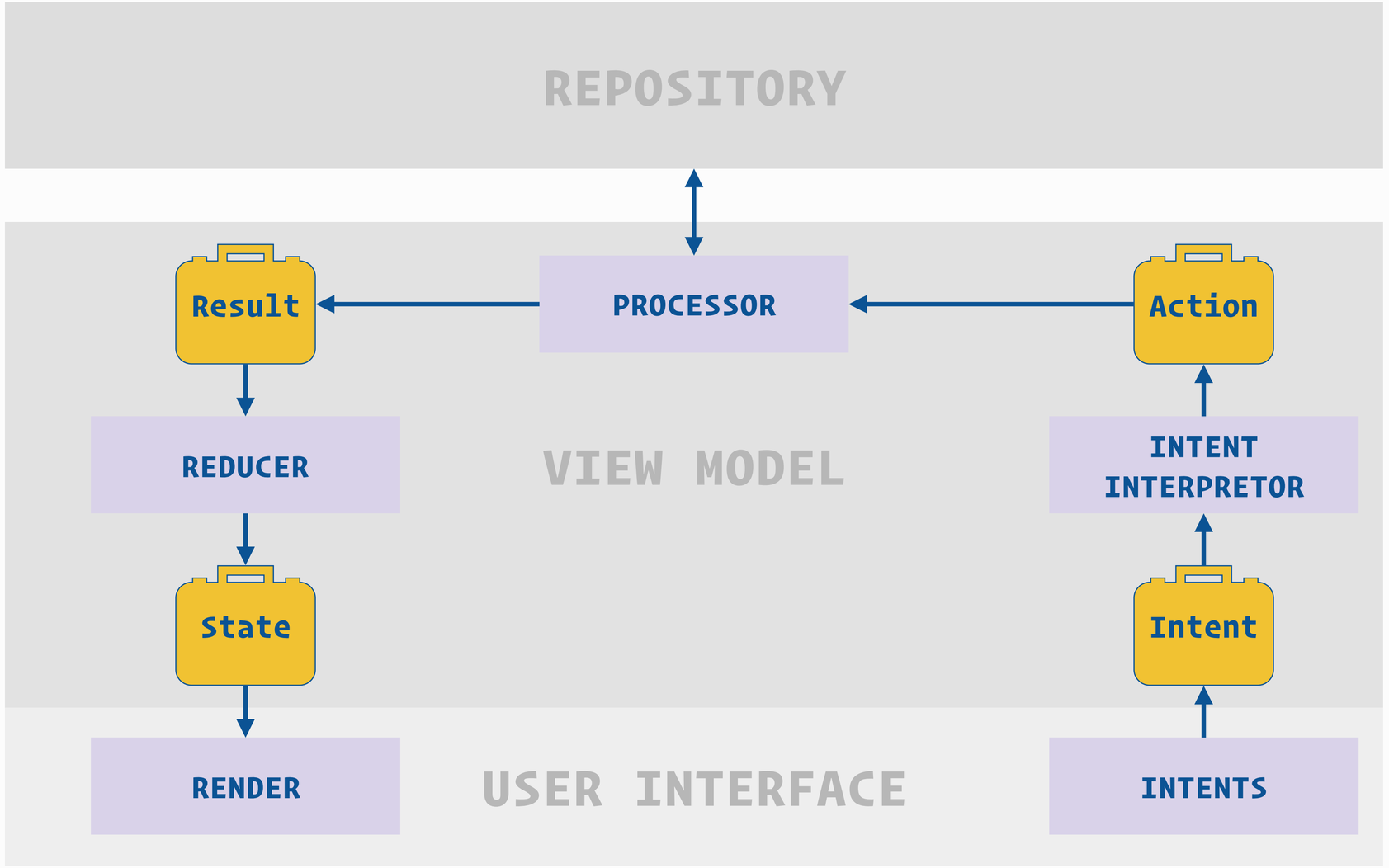
```
private fun compose(): Observable<TasksViewState> {
    return intentsSubject
            .scan(initialIntentFilter)
            .map { this.actionFromIntent(it) }
            .compose(actionProcessorHolder.actionProcessor)
            .scan(TasksViewState.idle(), reducer)
}

private val initialIntentFilter =
  BiFunction { _: TasksIntent, newIntent: TasksIntent ->
  }
```

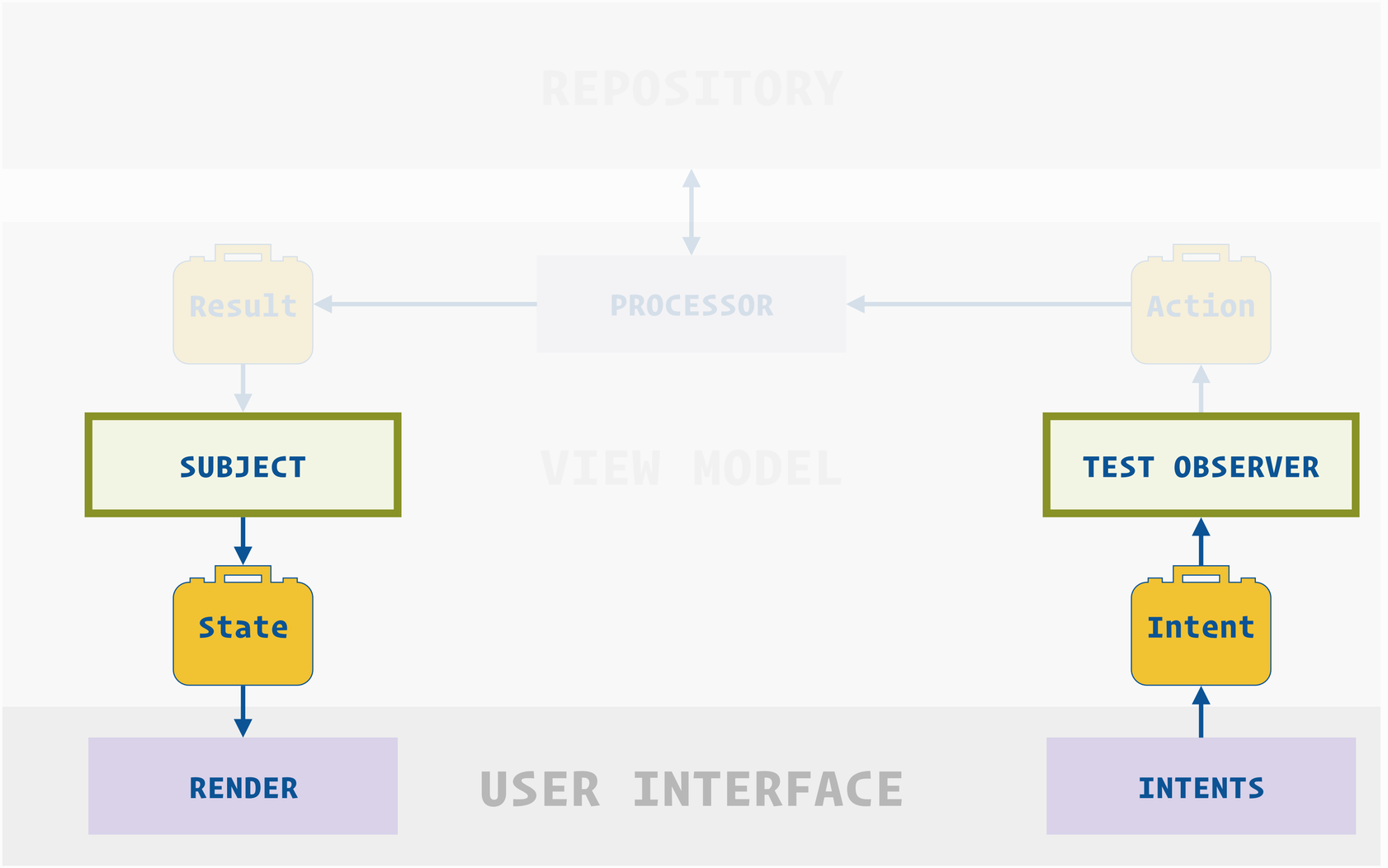
```
private fun compose(): Observable<TasksViewState> {
    return intentsSubject
            .scan(initialIntentFilter)
            .map { this.actionFromIntent(it) }
            .compose(actionProcessorHolder.actionProcessor)
            .scan(TasksViewState.idle(), reducer)
}

private val initialIntentFilter =
  BiFunction { _: TasksIntent, newIntent: TasksIntent ->
    if (newIntent is TasksIntent.InitialIntent) {
      TasksIntent.GetLastState
    }
  }
```

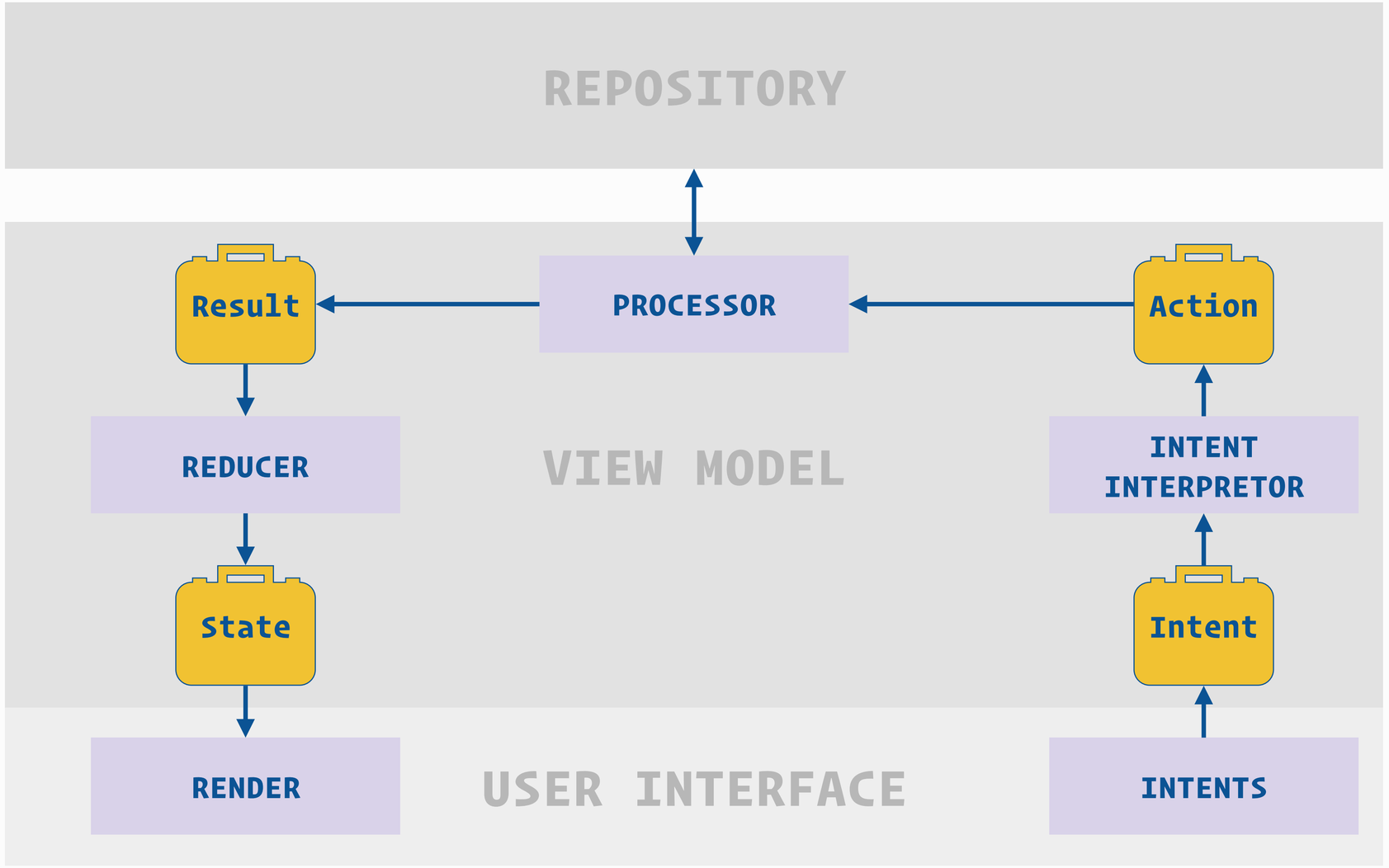
```
private fun compose(): Observable<TasksViewState> {
    return intentsSubject
            .scan(initialIntentFilter)
            .map { this.actionFromIntent(it) }
            .compose(actionProcessorHolder.actionProcessor)
            .scan(TasksViewState.idle(), reducer)
}

private val initialIntentFilter =
  BiFunction { _: TasksIntent, newIntent: TasksIntent ->
    if (newIntent is TasksIntent.InitialIntent) {
      TasksIntent.GetLastState
    } else {
      newIntent
    }
  }
```

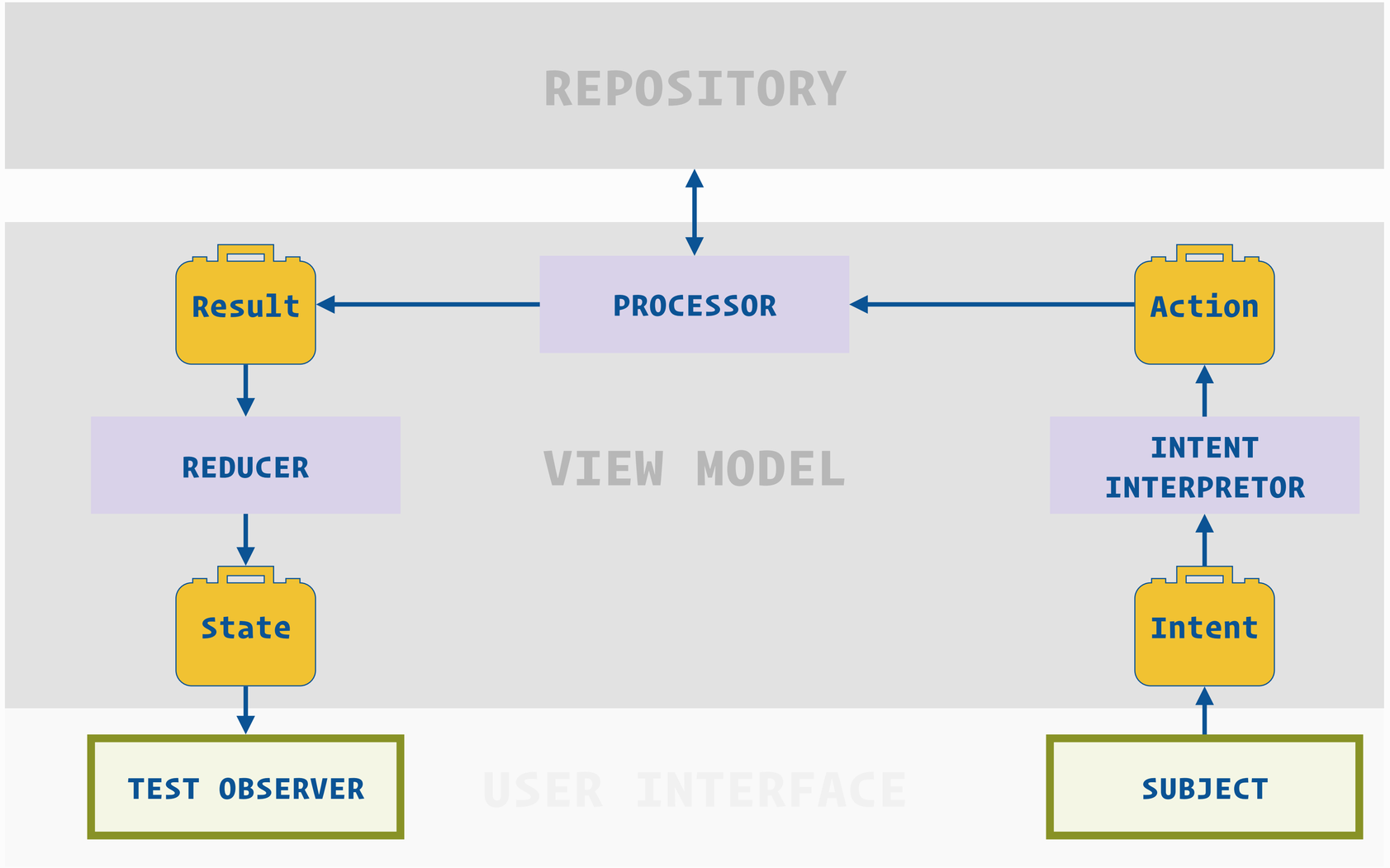
```
private fun compose(): Observable<TasksViewState> {
    return intentsSubject
            .scan(initialIntentFilter)
            .map { this.actionFromIntent(it) }
            .compose(actionProcessorHolder.actionProcessor)
            .scan(TasksViewState.idle(), reducer)
}

private val initialIntentFilter =
  BiFunction { _: TasksIntent, newIntent: TasksIntent ->
    if (newIntent is TasksIntent.InitialIntent) {
      TasksIntent.GetLastState
    } else {
      newIntent
    }
  }
```

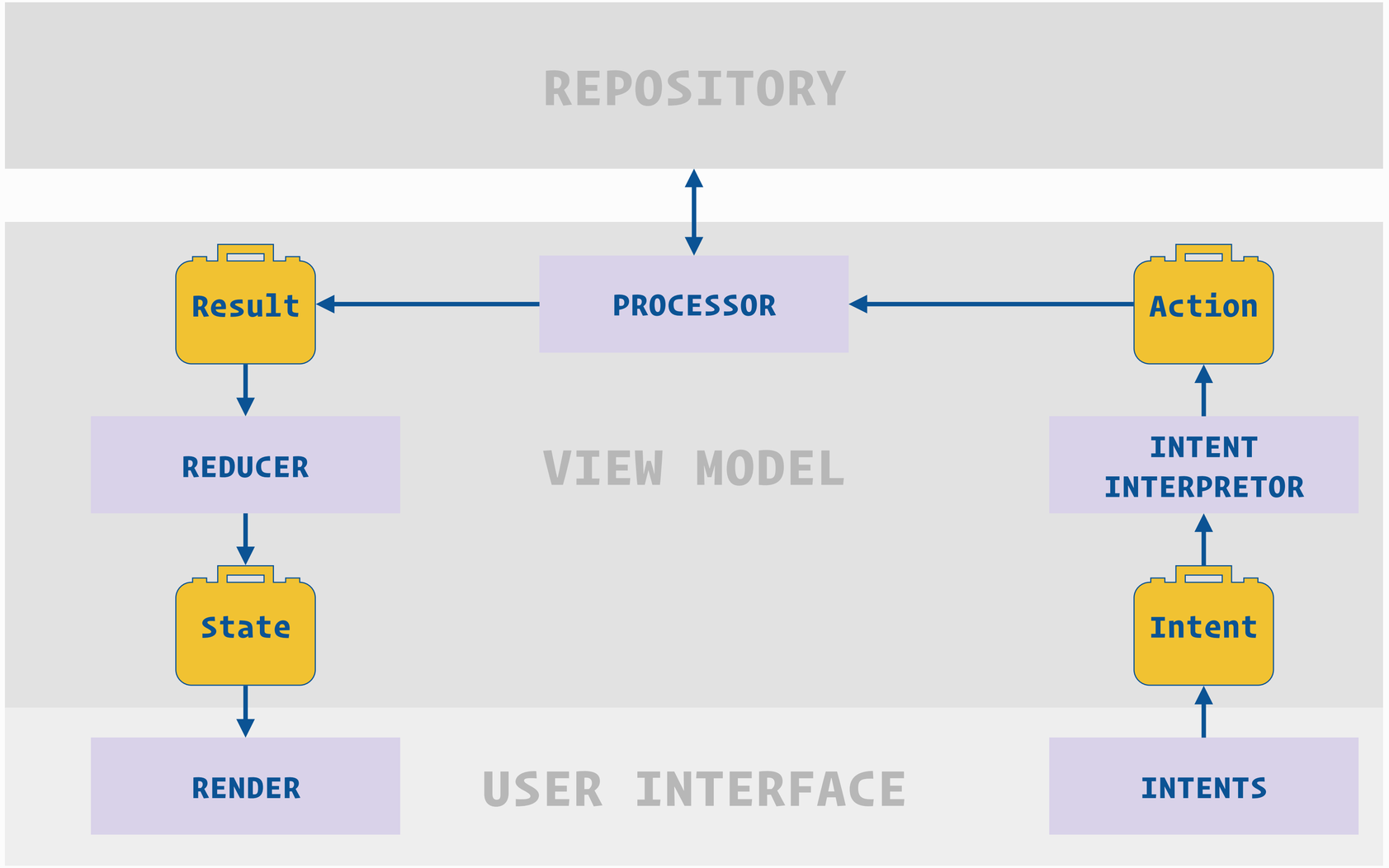
Do you even test?

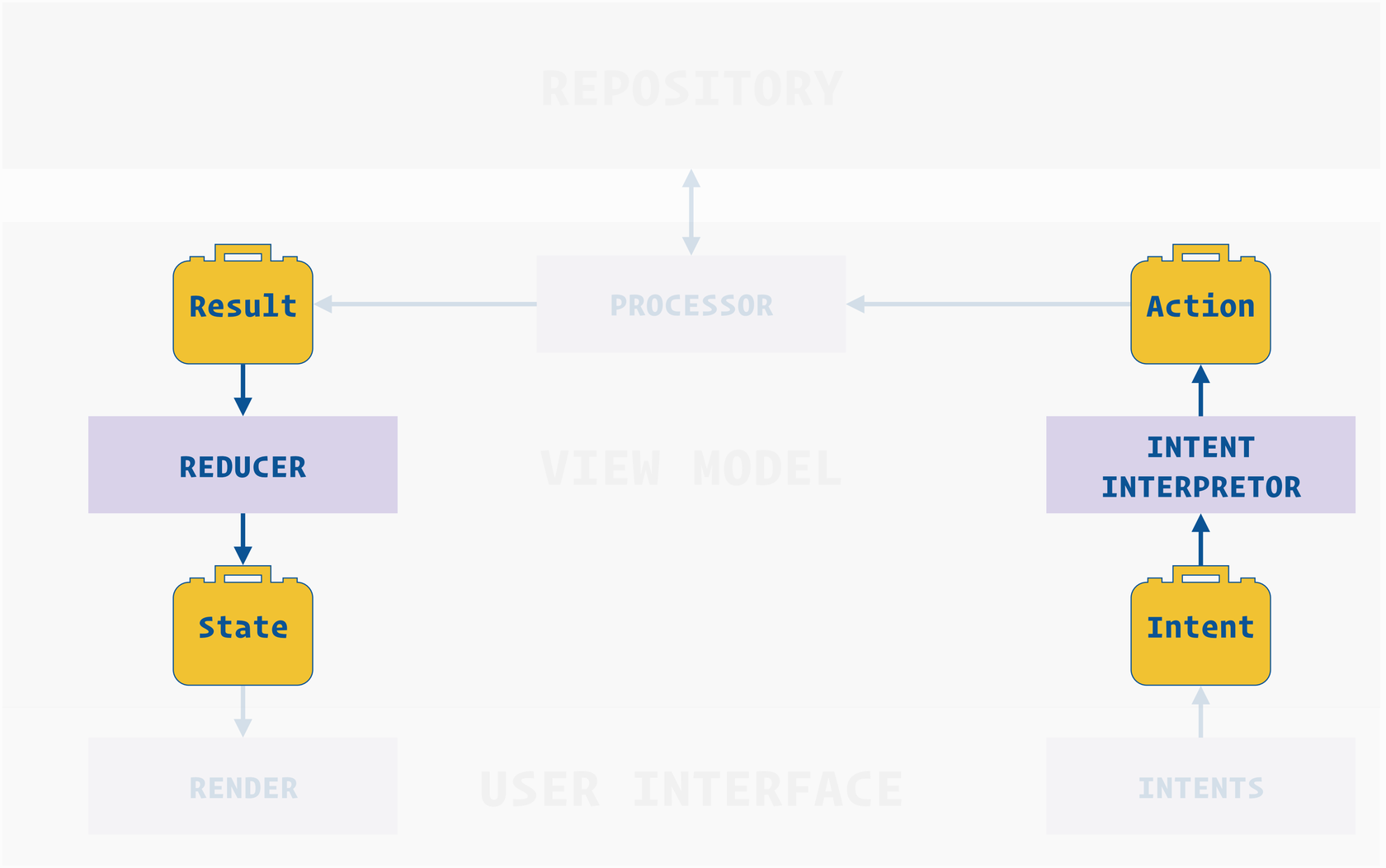

REPOSITORY
Result
PROCESSOR
Action
REDUCER
VIEW MODEL
INTENT
INTERPRETOR
State
Intent
RENDER
USER INTERFACE
INTENTS

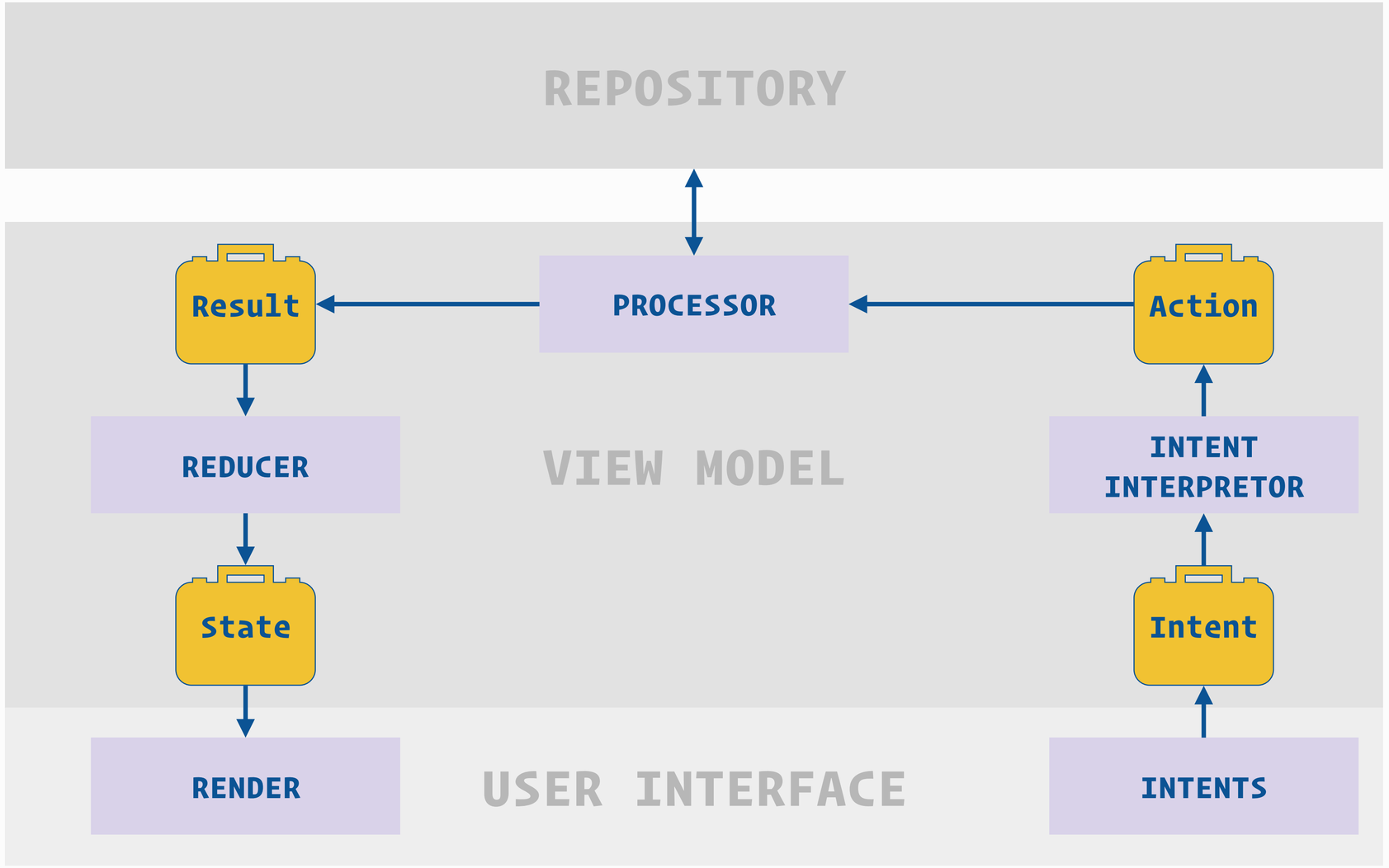

REPOSITORY
Result
PROCESSOR
Action
SUBJECT
VIEW MODEL
TEST OBSERVER
State
Intent
RENDER
USER INTERFACE
INTENTS

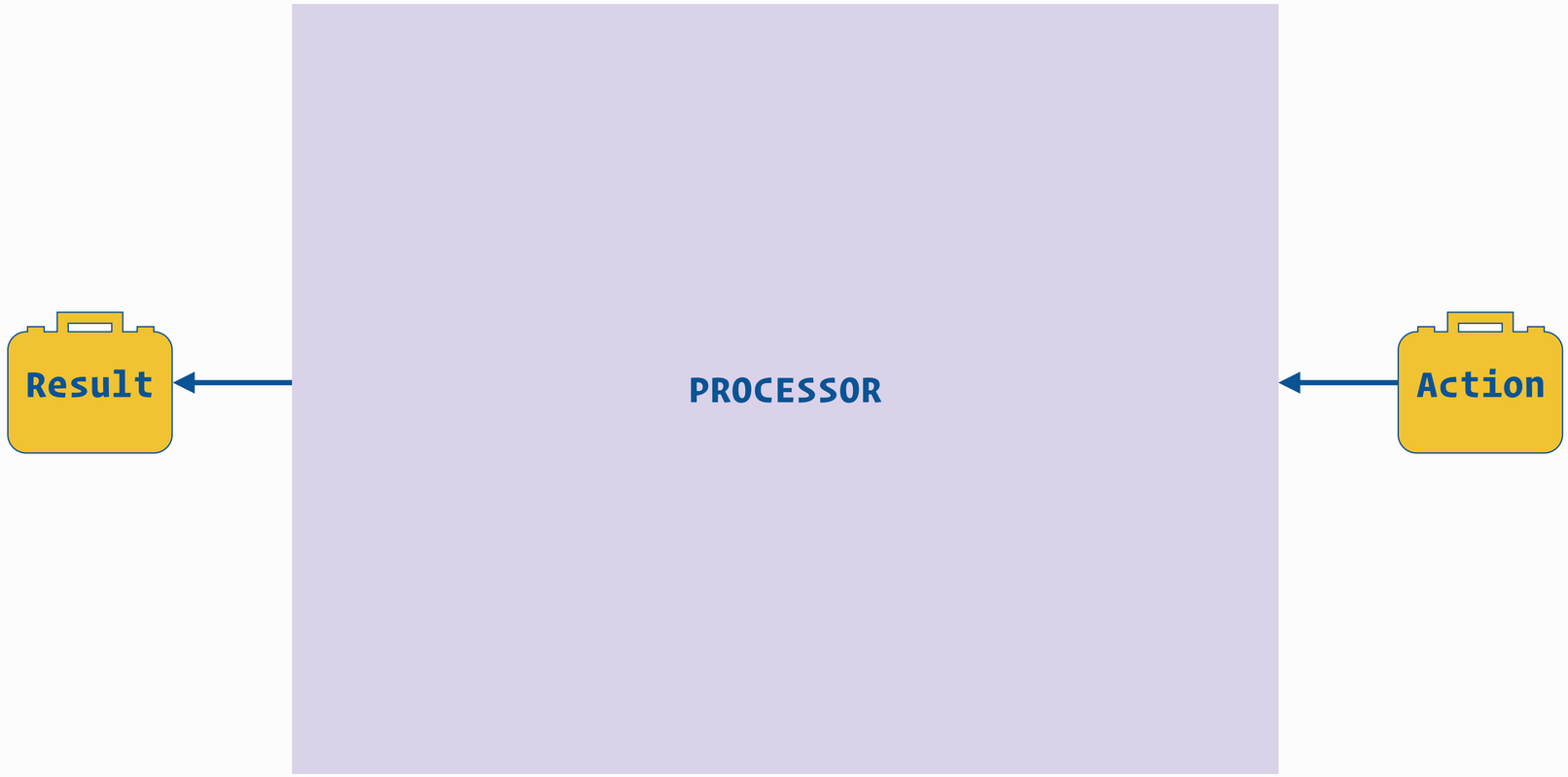

REPOSITORY
Result
PROCESSOR
Action
REDUCER
VIEW MODEL
INTENT
INTERPRETOR
State
Intent
RENDER
USER INTERFACE
INTENTS

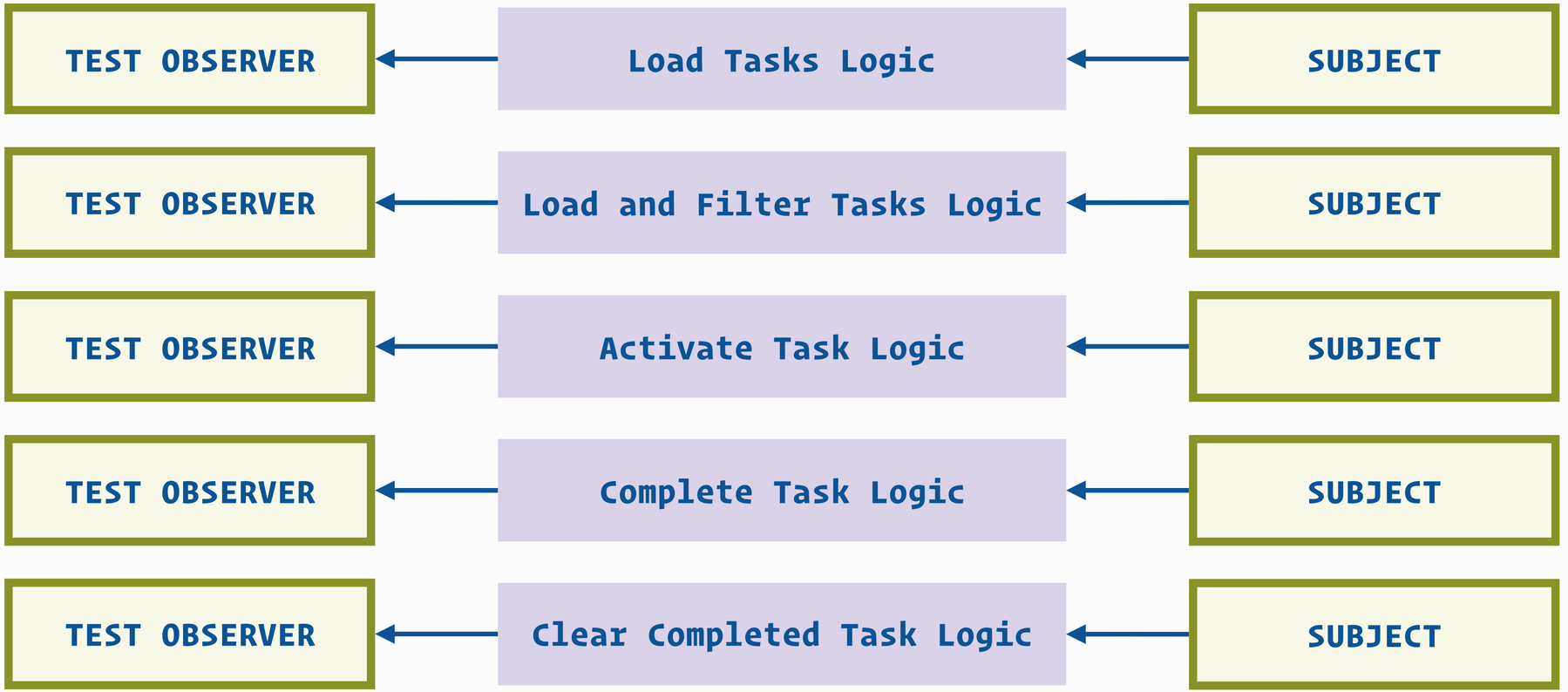

REPOSITORY
Result
PROCESSOR
Action
REDUCER
VIEW MODEL
INTENT
INTERPRETOR
State
Intent
TEST OBSERVER
USER INTERFACE
SUBJECT


REPOSITORY
Result
PROCESSOR
Action
REDUCER
VIEW MODEL
INTENT
INTERPRETOR
State
Intent
RENDER
USER INTERFACE
INTENTS


REPOSITORY
PROCESSOR
Result
Action
REDUCER
INTENT
INTERPRETOR
VIEW MODEL
State
Intent
RENDER
USER INTERFACE
INTENTS


REPOSITORY
Result
PROCESSOR
Action
REDUCER
VIEW MODEL
INTENT
INTERPRETOR
State
Intent
RENDER
USER INTERFACE
INTENTS


Result
PROCESSOR
Action


TEST OBSERVER
Load Tasks Logic
SUBJECT
TEST OBSERVER
Load and Filter Tasks Logic
SUBJECT
TEST OBSERVER
Activate Task Logic
SUBJECT
TEST OBSERVER
Complete Task Logic
SUBJECT
TEST OBSERVER
Clear Completed Task Logic
SUBJECT

oldergod / **android-architecture**

forked from googlesamples/android-architecture

Fin

Benoît Quenaudon @oldergod