Управление сложностью состояния

Сергей Опивалов

Gradle Inc.

mobius

Знакомство

Senior SWE @ Gradle Inc.

ex. SQUIRE

ex. Yandex Pro

`#android-dev` `#dev-prod` `#builds`

`#JVM` `#tools` `#CI/CD`

`#making software softer`

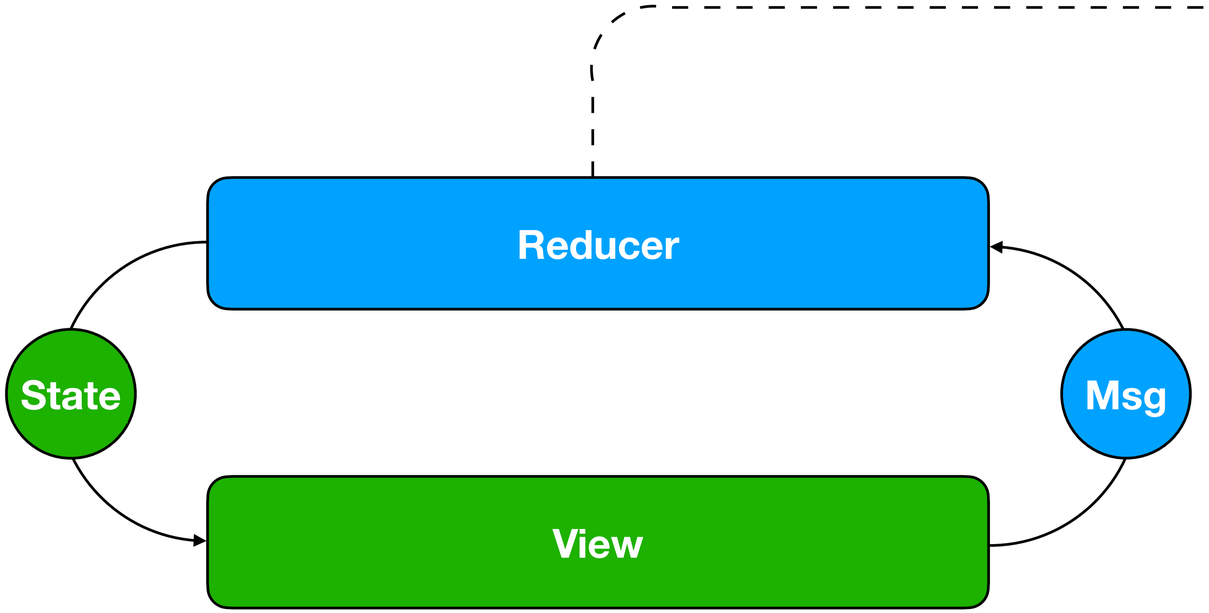
@sergey_opivalov

@6hundreds

Структура

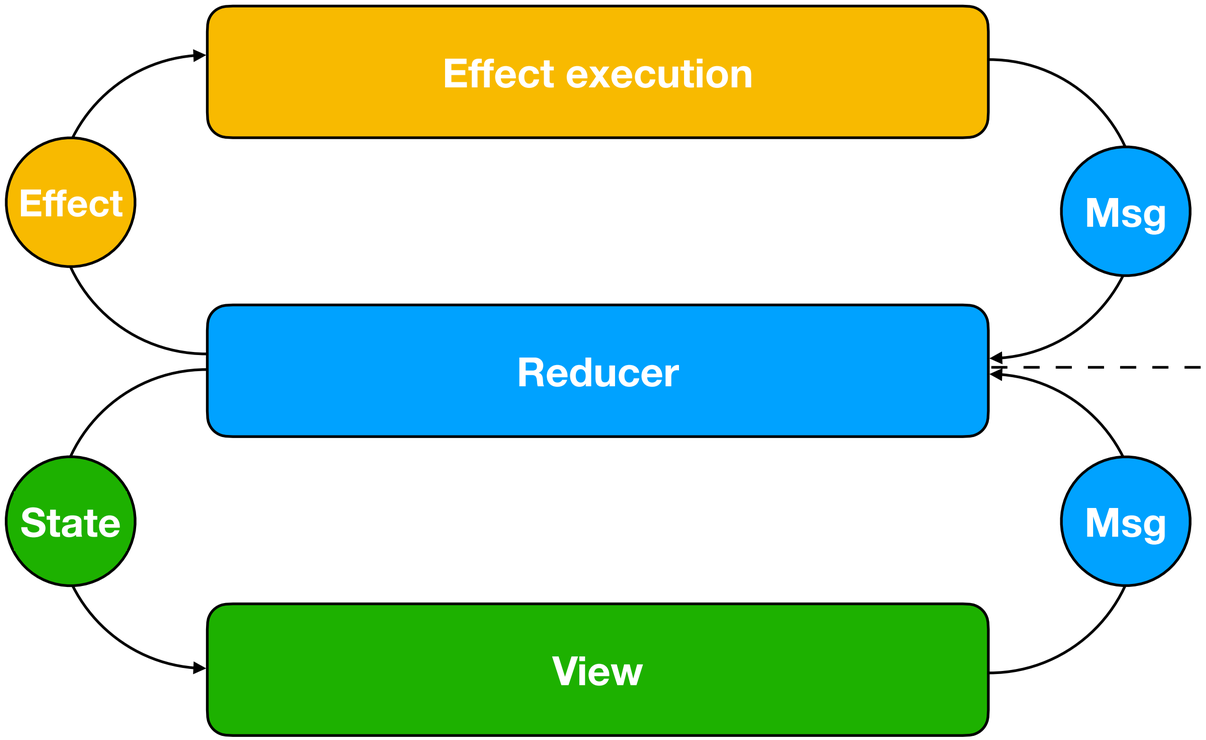
1. Краткое описание UDF архитектур
    1.1. UDF 101
    1.2. Почему UDF?
    1.3. TEA-like UDF
    1.4. Моделируем сложное состояние с UDF
2. Сложность, что это?
    2.1. Simple vs Easy
3. Сложность алгоритма
    3.1. Нотация сложности алгоритма
    3.2. Структурные гипотезы сложности алгоритма
4. Что делает алгоритм сложным?
5. Управляем сложностью. Визуальный признак
    5.1. NestedFragmentsHost
6. Можно ли привыкнуть к сложному алгоритму?
7. Управляем сложностью. Логический признак
    7.1. HeadlessFeaturesHost
8. Выводы

UDF




```
fun update(msg : Msg, state : State) : State =
    when(msg) {
        is Foo → state.copy(isFoo = true)
        is Bar → state.copy(isBar = true)
    }
```

UDF




```
typealias Update = Pair<State, Set<Effect>>

fun update(msg : Msg, state : State) : Update =
    when(msg) {
        is Foo →
         state.copy(isFoo = true) with Effects.MakeFoo
        is Bar →
         state.copy(isBar = true) with Effects.MakeBar
    }
```

Feature

```
object Feature {
    class Deps()
    data class State()
    sealed class Msg{}
    fun update(msg : Msg, state: State) : Update
}
```

Runtime

```
class EffektRuntime<State : Any, Msg : Any, Deps : Any>(
    private val update: (Msg, State) → Update,
    private val deps: Deps
)
```

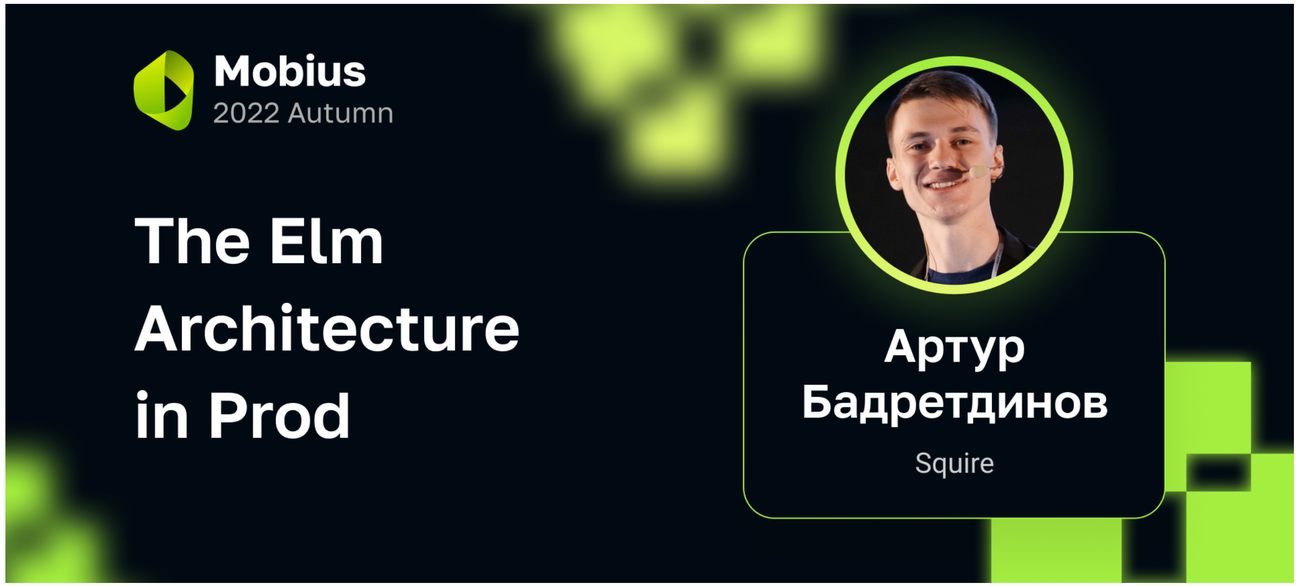
ViewModel

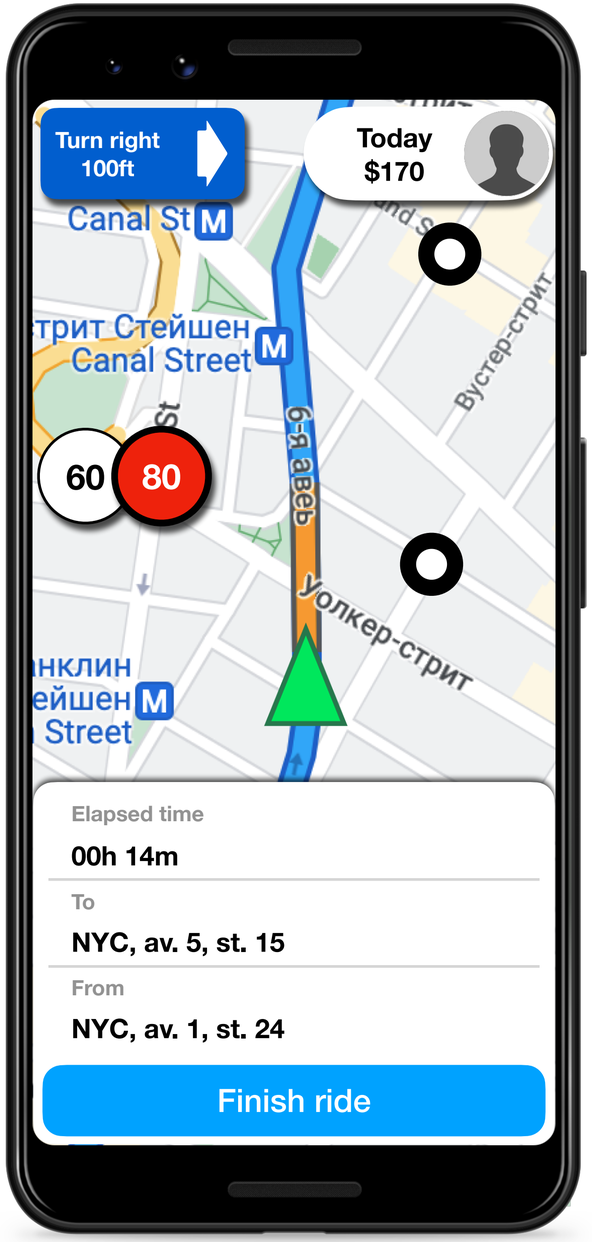
```
abstract class EffektViewModel<State : Any, Msg : Any, Deps : Any>(
    update: (Msg, State) → Update,
    deps: Deps,
) : ViewModel() {

private val runtime by lazy { EffektRuntime(update, deps) }
}
```

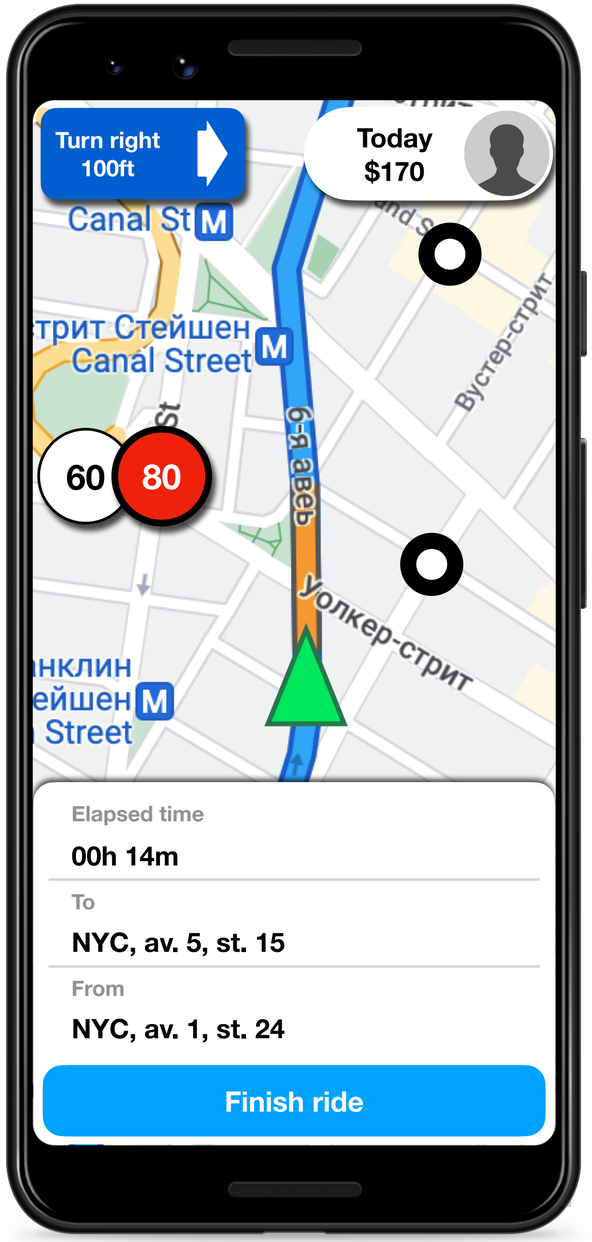
Fragment

```
abstract class EffektFragment<State : Any, Msg : Any, Deps : Any> : Fragment {

    abstract val viewModel: EffektViewModel<State, Msg, Deps>
}
```

The Elm Architecture in Prod

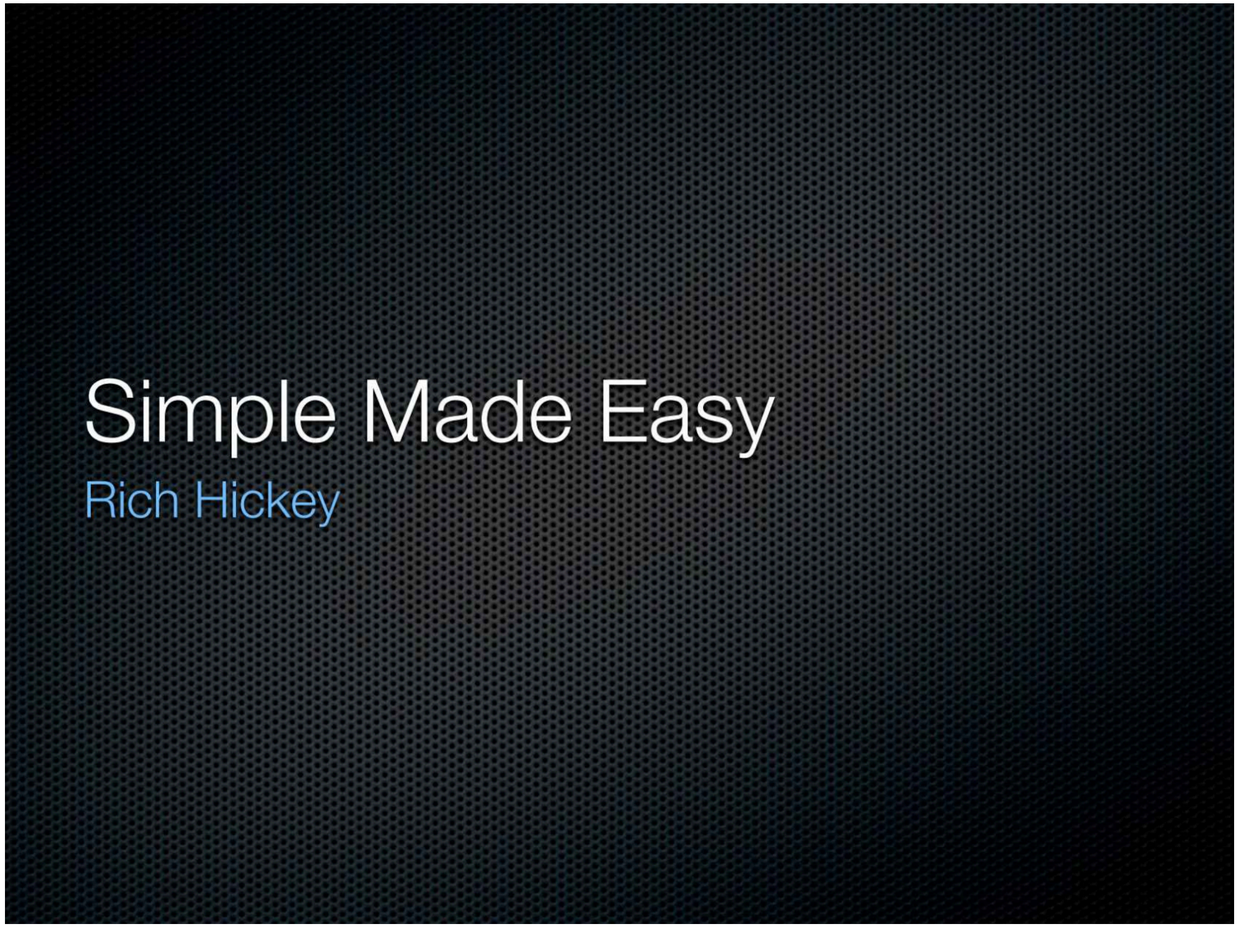



Turn right
100ft
Today
$170
Canal St
M
трит Стейшен
Canal Street
M
60
80
6-я авеь
Вустер-стрит
Уолкер-стрит
анклин
ейшен
M
Street
Elapsed time
00h 14m
To
NYC, av. 5, st. 15
From
NYC, av. 1, st. 24
Finish ride




```
data class State(
  val navigationData : NavigationData,
  val currentPosition : LatLng,
  val route : List<Polygon>,
  val pois : List<Poi>,
  val profileData : ProfileData,
  val rideData : RideData,
  val speedData : SpeedData
)

sealed class Msg {
    class CurrentPositionUpdated(val position : LatLng) : Msg()
    class RouteUpdated(val route : List<Polygon>) : Msg()
    class NavigationDataUpdated(val data : NavigationData) : Msg()
    class ProfileUpdated(val data : ProfileData) : Msg()
    class PoiUpdated(val pois : List<Poi>) : Msg()
    class OnPoiClicked(val poi : Poi) : Msg()
    object OnProfileClicked() : Msg()
    object OnFinishRideClicked() : Msg()
}
```

Simple made easy

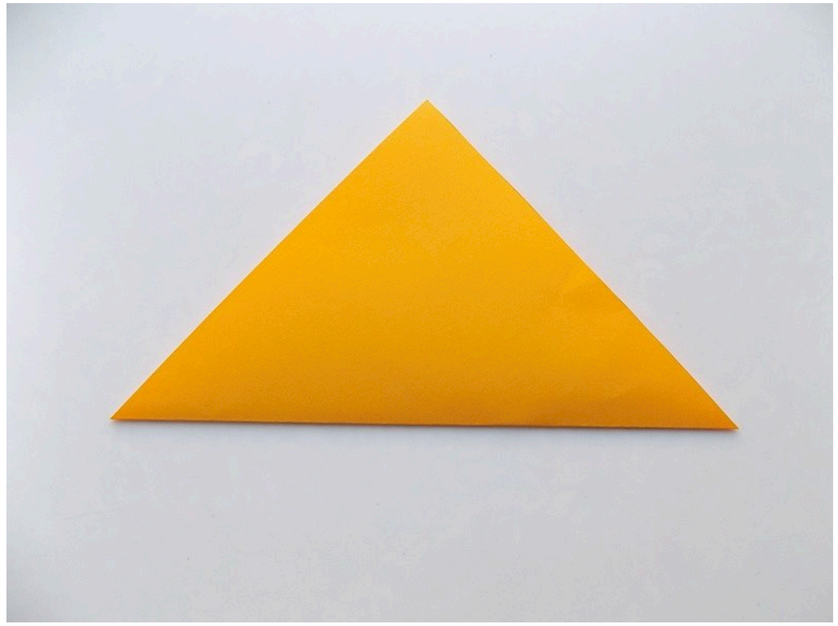


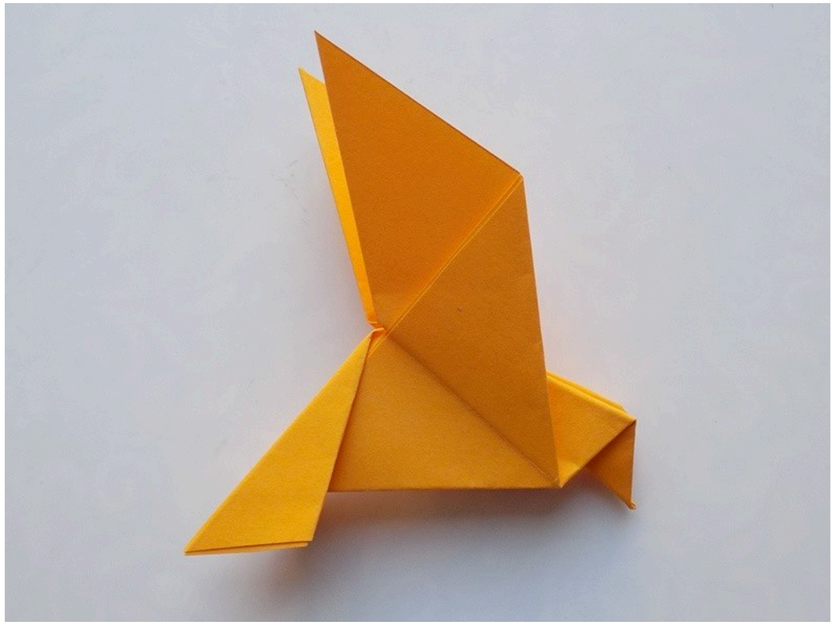
Simple

Sim- plex
Одна складка/одно сплетение

vs.

Com- plex
Сложенные/сплетенные вместе





Simple

Да

- **Одна задача**
- **Одна роль**
- **Одна концепция**

Нет

- **Одна операция**
- **Один экземпляр**

Simple

- **Не про мощность множества (cardinality)**
- **Про отсутствие пересечений**
- **Объективная характеристика**

Easy

Easy < aise(франц.) < adjacence
Лежит рядом, доступно

vs.

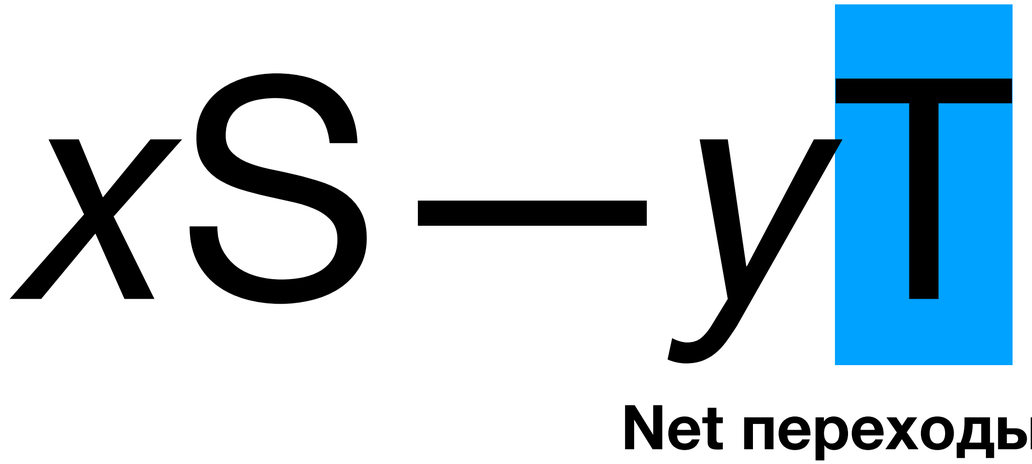
Hard

- **Доступно для наших способностей**
- **Доступно для наших инструментов**
- **Субъективная характеристика**

Нотация сложности алгоритма

$$xS - yT$$

Нотация сложности алгоритма

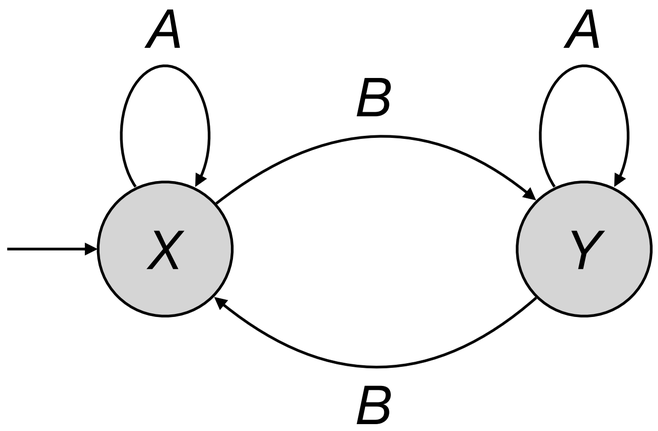


S-complexity

Правило **A** *сложнее, чем правило* **B** *если* **A** *имеет больше состояний, чем* **B**

Simple

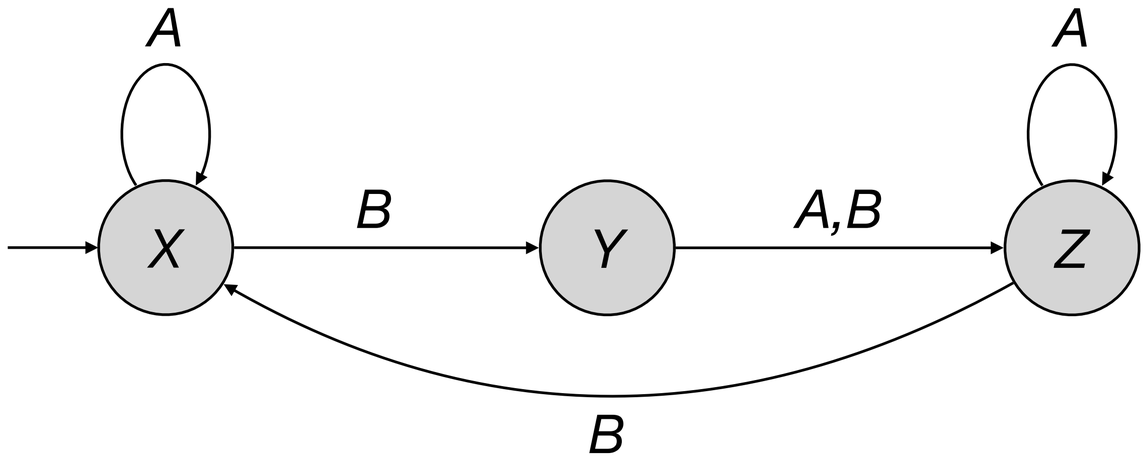
Правило B



2S-2T

Complex

Правило A



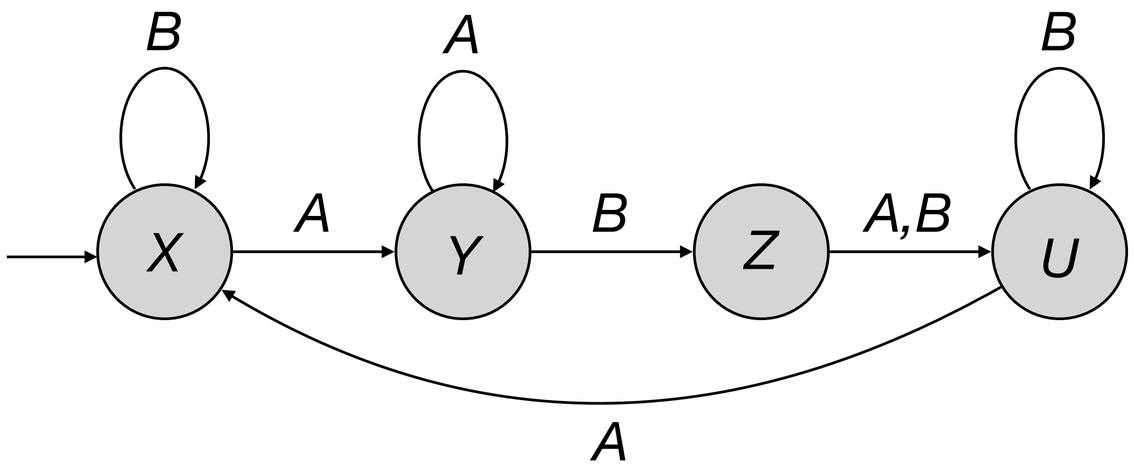
3S-2T

T-complexity

Правило **A** *сложнее, чем правило* **B** *если* **A** *имеет больше net переходов, чем* **B**

Simple

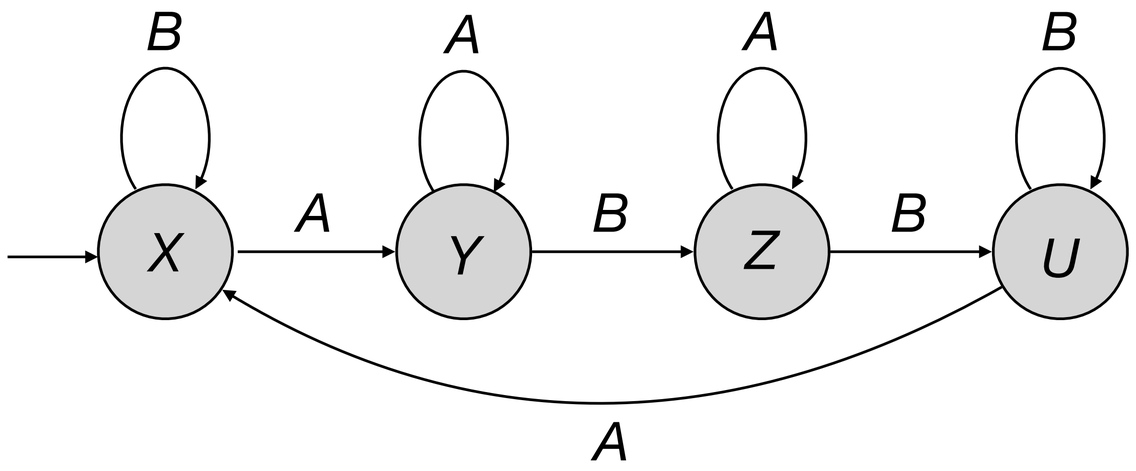
Правило B



4S-3T

Complex

Правило A



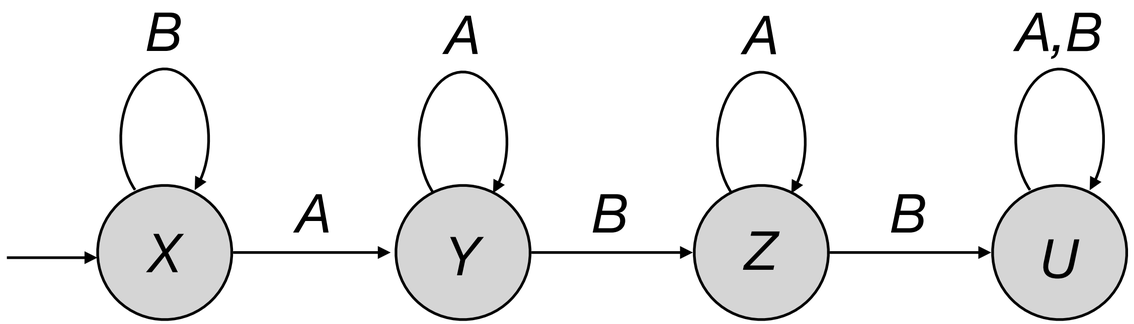
4S-4T

Absorption

При прочих равных, правило **A** *сложнее, чем* **B** *если* **B** *имеет абсорбирующее состояние, а* **A** *нет.*

Simple

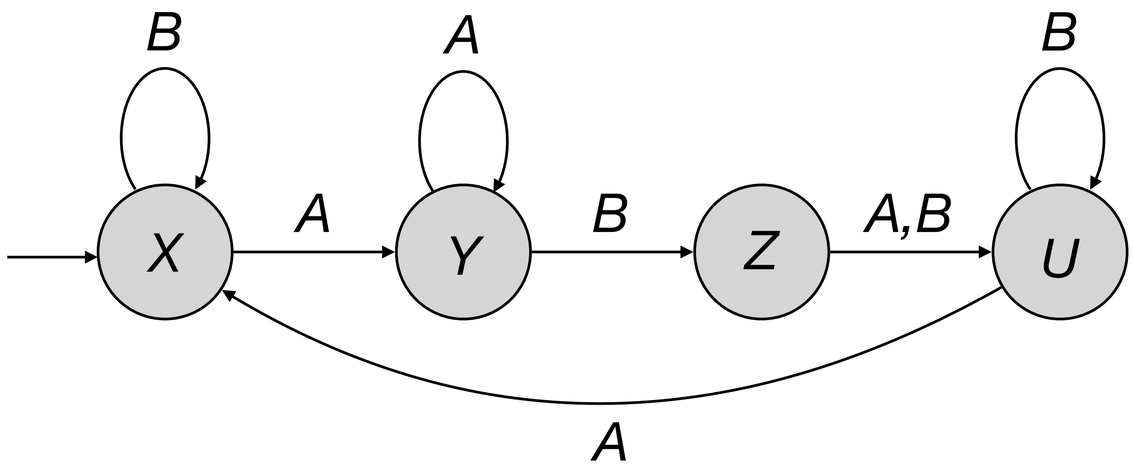
Правило B



4S-3Ta

Complex

Правило A

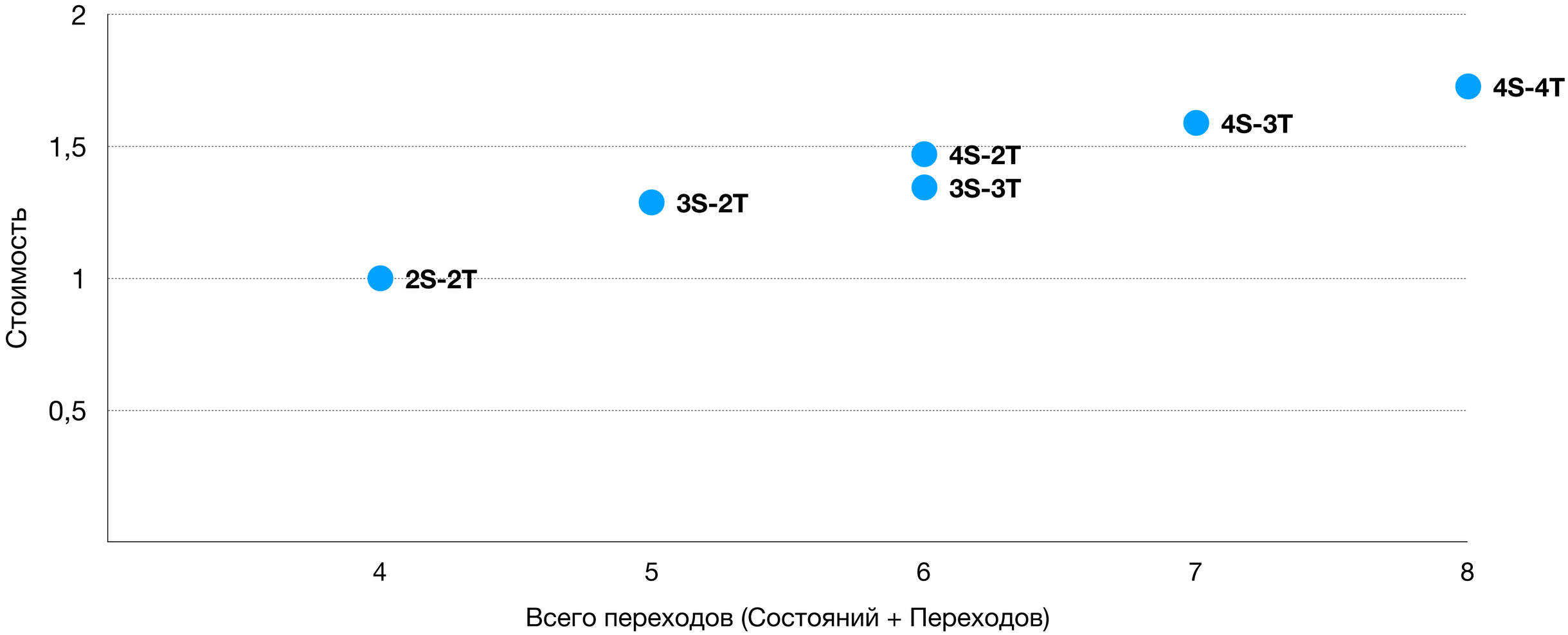


4S-3T

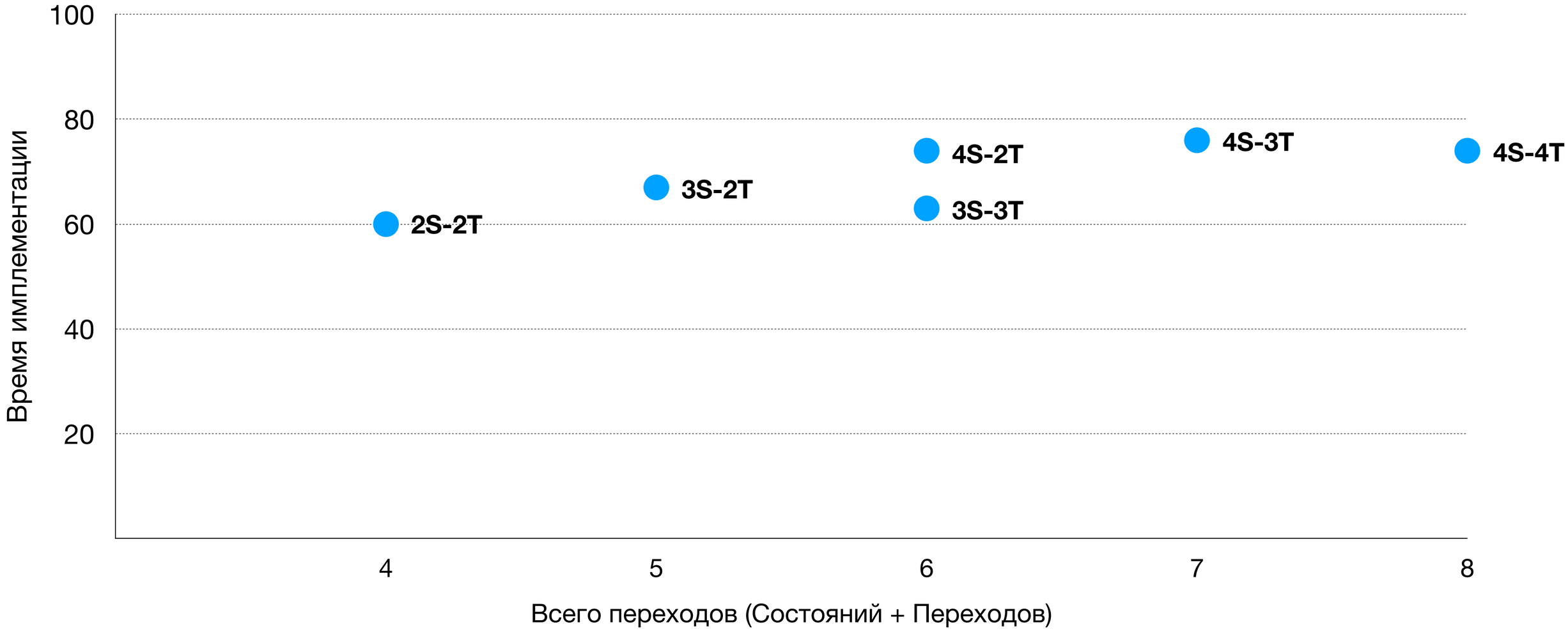
What makes a rule complex?

AMERICAN
ECONOMIC
ASSOCIATION

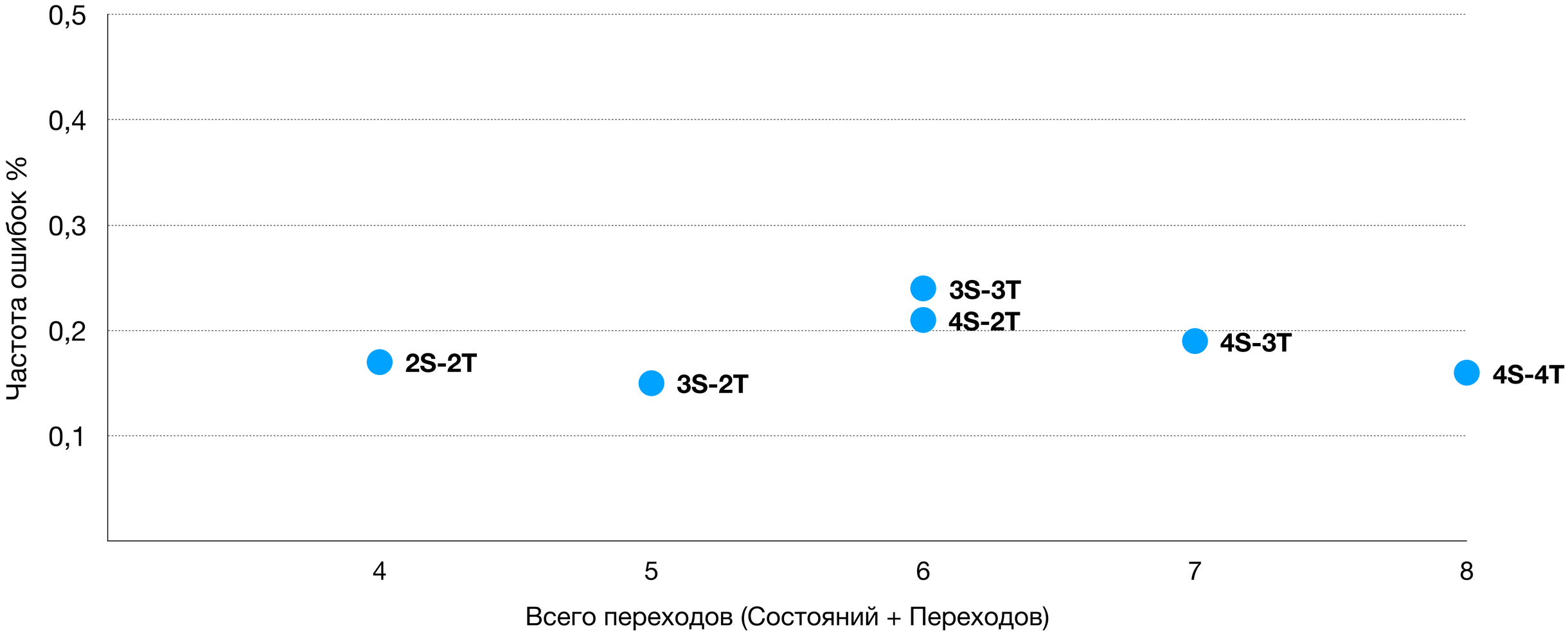
Метрика сложности: Стоимость



Метрика сложности: Время



Метрика сложности: Частота ошибок



Результаты

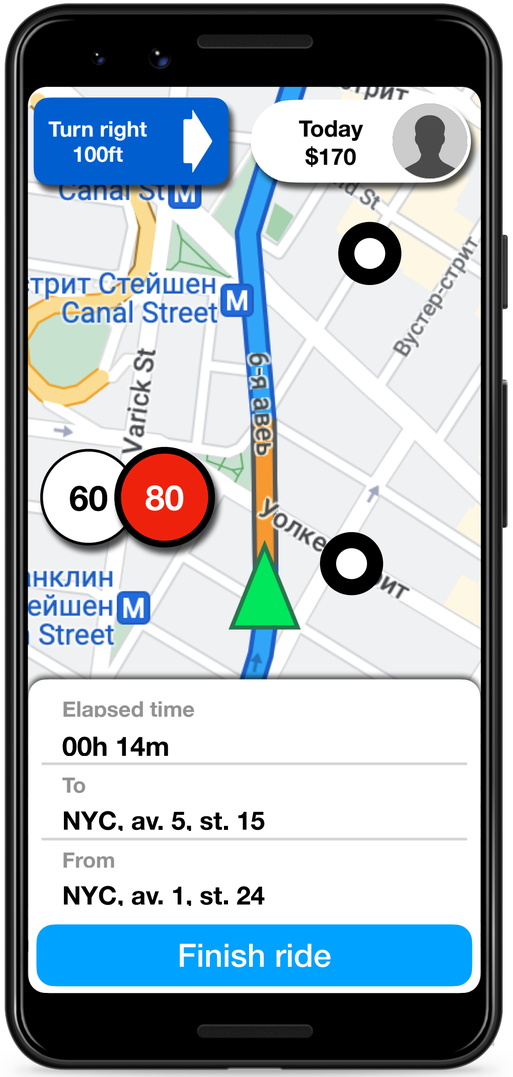
Фактор	Влияние на сложность
Количество состояний	+1 дополнительное состояние == ~25% к сложности алгоритма
Наличие абсорбирующего состояния	Минус 25% от сложности алгоритма
Количество переходов	+1 дополнительный переход == ~12% к сложности алгоритма

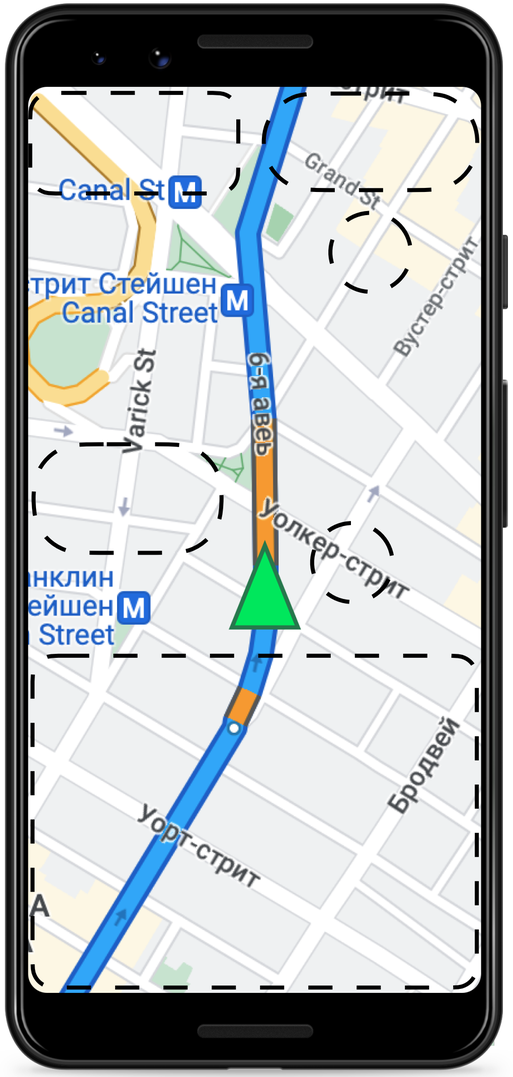
```
data class State(
  val navigationData : NavigationData,
  val currentPosition : LatLng,
  val route : List<Polygon>,
  val pois : List<Poi>,
  val profileData : ProfileData,
  val rideData : RideData,
  val speedData : SpeedData
)

sealed class Msg {
    class CurrentPositionUpdated(val position : LatLng) : Msg()
    class RouteUpdated(val route : List<Polygon>) : Msg()
    class NavigationDataUpdated(val data : NavigationData) : Msg()
    class ProfileUpdated(val data : ProfileData) : Msg()
    class PoiUpdated(val pois : List<Poi>) : Msg()
    class OnPoiClicked(val poi : Poi) : Msg()
    object OnProfileClicked() : Msg()
    object OnFinishRideClicked() : Msg()
}
```

7S

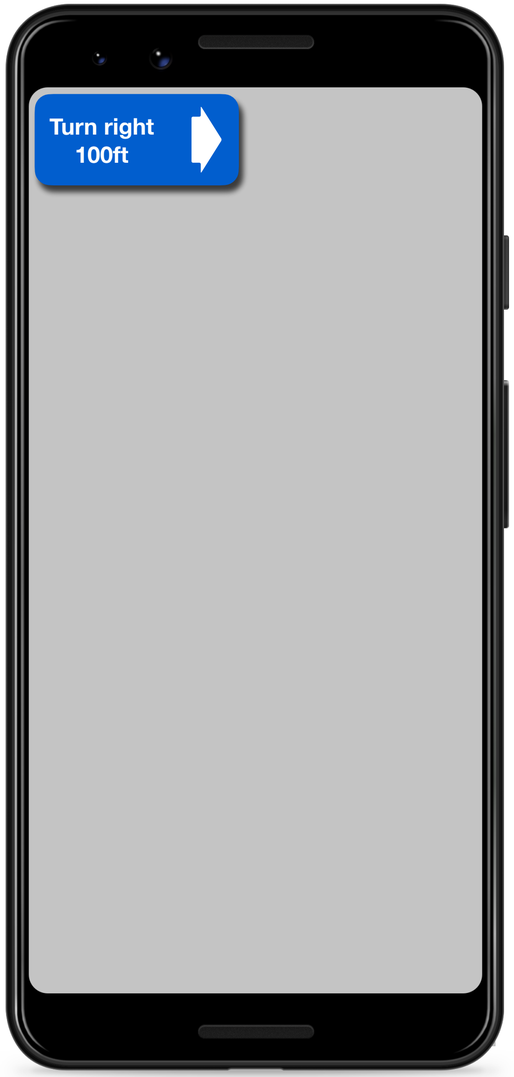
8T

Декомпозиция



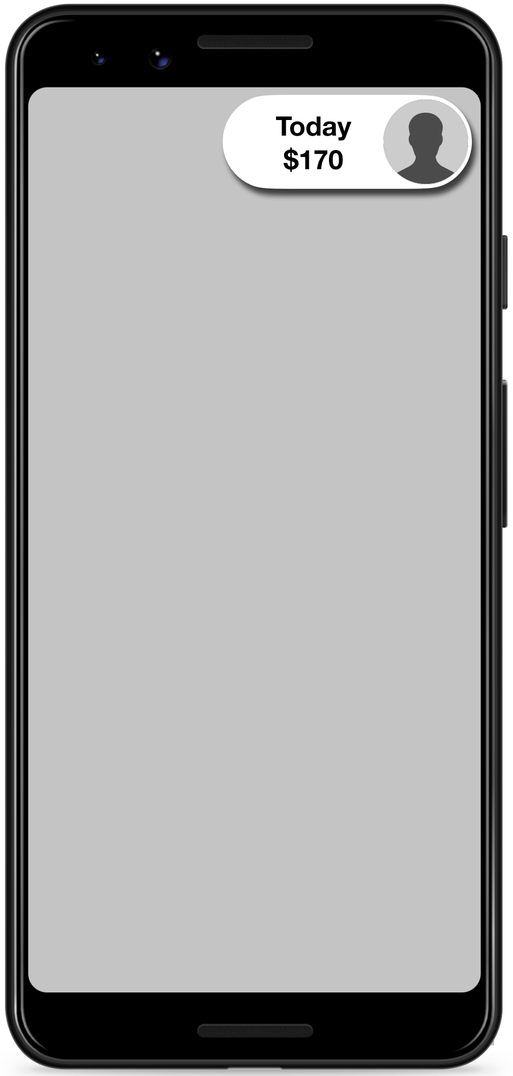


NavigationWidget



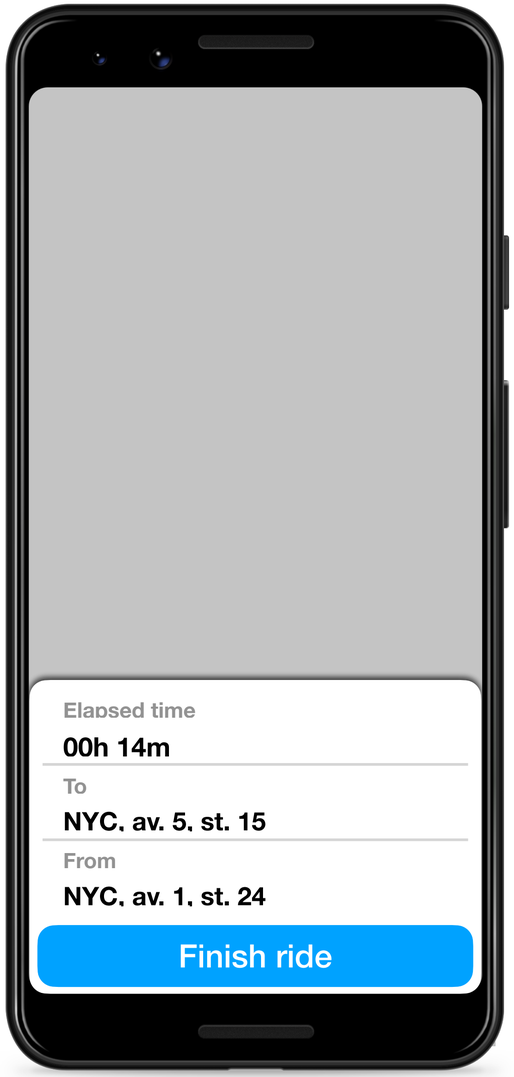

```
data class State(
  val navigationData: NavigationData,
)

sealed class Msg {
    class NavigationDataUpdated(val data : NavigationData) : Msg()
}
```

ProfileWidget



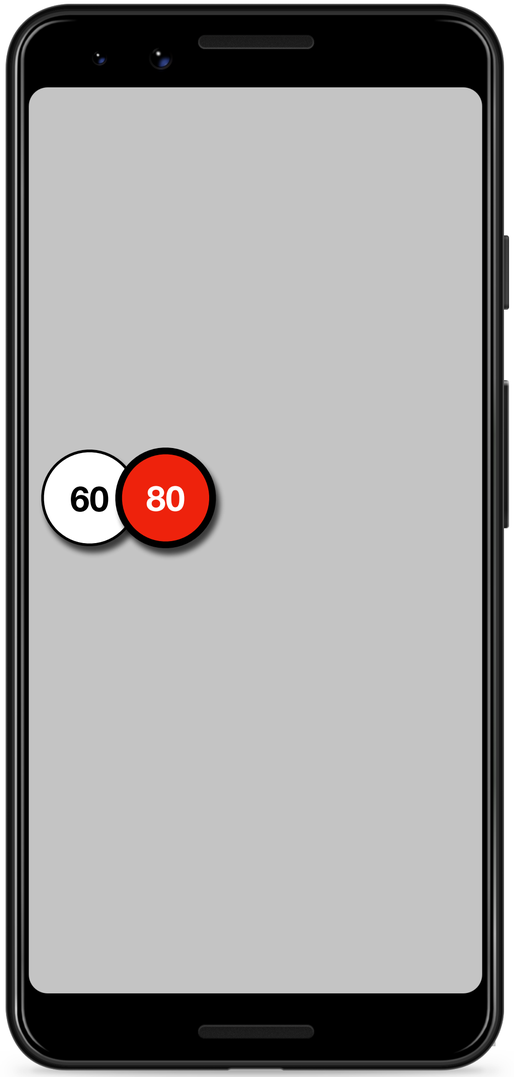

```
data class State(
  val profileData: ProfileData,
)

sealed class Msg {
    class ProfileUpdated(val data : ProfileData) : Msg()
    object OnProfileClicked() : Msg()
}
```

RidePanel



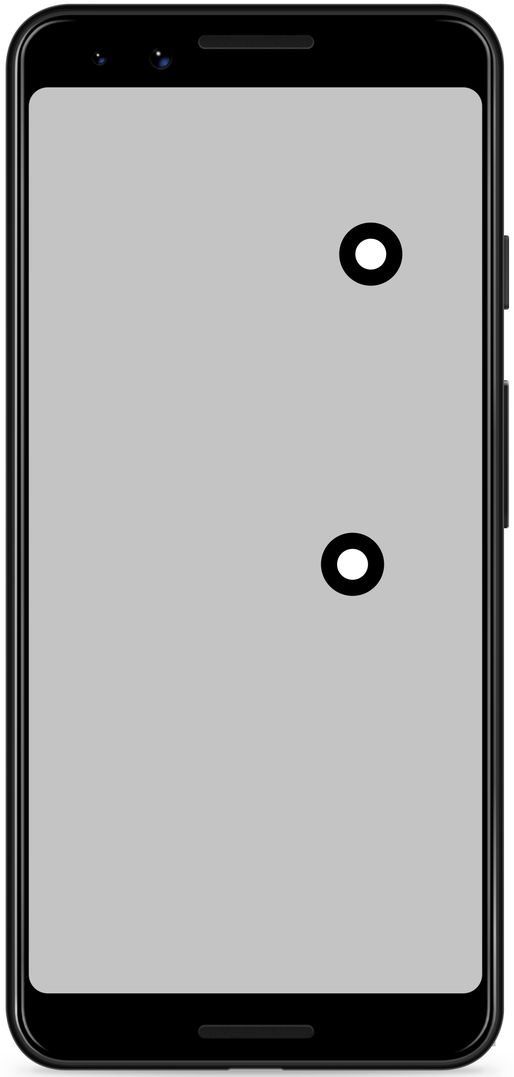

```
data class State(
  val rideData: RideData
)

sealed class Msg {
    class RideDataUpdated(val data: RideData) : Msg()
    object OnFinishRideClicked() : Msg()
}
```

SpeedWidget



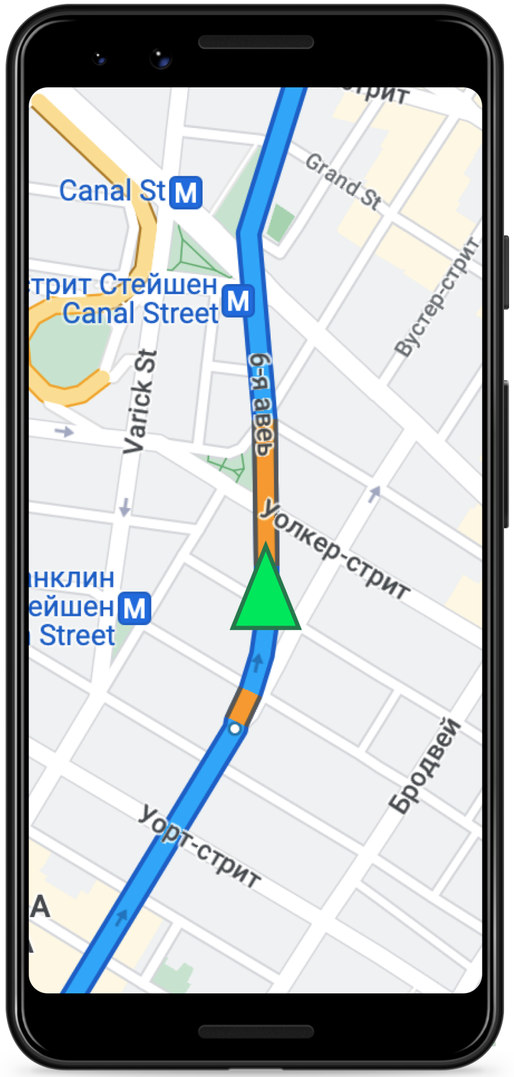

```
data class State(
  val speedData: SpeedData
)

sealed class Msg {
    class SpeedDataUpdated(val data: SpeedData) : Msg()
}
```

Poi



```
data class State(
  val pois : List<Poi>
)

sealed class Msg {
    class PoiUpdated(val pois : List<Poi>) : Msg()
    class OnPoiClicked(val poi : Poi) : Msg()
}
```

Ride




```
data class State(
  val currentPosition : LatLng,
  val route : List<Polygon>,
)

sealed class Msg {
    class CurrentPositionUpdated(val position : LatLng) : Msg()
    class RouteUpdated(val route : List<Polygon>) : Msg()
}
```

NestedFragmentsHost

```
typealias NestedFragments = Map<
  Class<out EffektFragment<*, *, *>>,
  FragmentContainerView
  >

interface NestedFragmentsHost {

   fun getNestedFragments(): NestedFragments
}
```

NestedFragmentsHost

```
class NestedFragmentHostDelegate(
  lifecycleOwnerLiveData: LiveData<LifecycleOwner>,
  private val fragmentManagerProvider: () → FragmentManager,
  private val nestedFragmentsProvider: () → NestedFragments,
  private val argumentsProvider: () → Bundle?
) {

  init {
    lifecycleOwnerLiveData.observeForever {
      it?.lifecycleScope?.launchWhenCreated { attachNestedFragments() }
    }
  }

  private fun attachNestedFragments() {
    // Attach nested fragments in single transaction
  }
}
```

NestedFragmentsHost

```
fun NestedFragmentsHost.nestedFragmentsHostDelegate(
  hostFragment: Fragment,
  argumentsProvider: () → Bundle? = hostFragment::getArguments
) = nestedFragmentsHostDelegate(
  lifecycleOwnerLiveData = hostFragment.viewLifecycleOwnerLiveData,
  fragmentManagerProvider = hostFragment::getChildFragmentManager,
  argumentsProvider = argumentsProvider
)

fun NestedFragmentsHost.nestedFragmentsHostDelegate(
  hostActivity: AppCompatActivity,
  argumentsProvider: () → Bundle?
) = nestedFragmentsHostDelegate(
  lifecycleOwnerLiveData = MutableLiveData(hostActivity),
  fragmentManagerProvider = hostActivity::getSupportFragmentManager,
  argumentsProvider = argumentsProvider
)
```

Визуальный признак


```
class NavigationWidgetFragment : EffektFragment<
    NavigationWidget.State,
    NavigationWidget.Msg,
    NavigationWidget.Deps
>(R.layout.fragment_widget_navigation)
```

```
class ProfileWidgetFragment : EffektFragment<
    ProfileWidget.State,
    ProfileWidget.Msg,
    ProfileWidget.Deps
>(R.layout.fragment_widget_profile)
```

```
class SpeedWidgetFragment : EffektFragment<
    SpeedWidget.State,
    SpeedWidget.Msg,
    SpeedWidget.Deps
>(R.layout.fragment_widget_speed)
```

```
class RidePanelFragment : EffektFragment<
    RidePanel.State,
    RidePanel.Msg,
    RidePanel.Deps
>(R.layout.fragment_panel_ride)
```

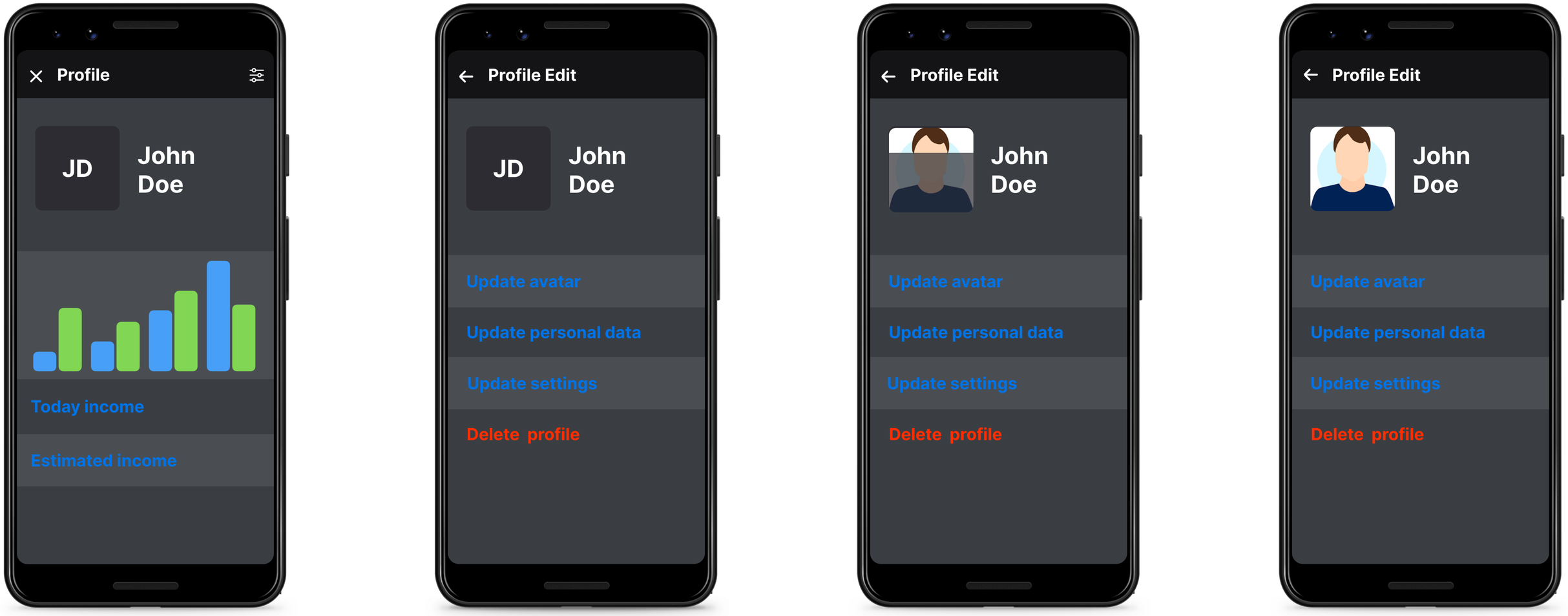
RideFragment

```
override fun getNestedFragments(): NestedFragments =
  mapOf(
    NavigationWidgetFragment::class.java to binding.navigationWidgetContainer,
    RidePanelFragment::class.java to binding.ridePanelContainer,
    ProfileWidgetFragment::class.java to binding.profileWidgetContainer,
    SpeedWidgetFragment::class.java to binding.speedWidgetContainer,
  )
```

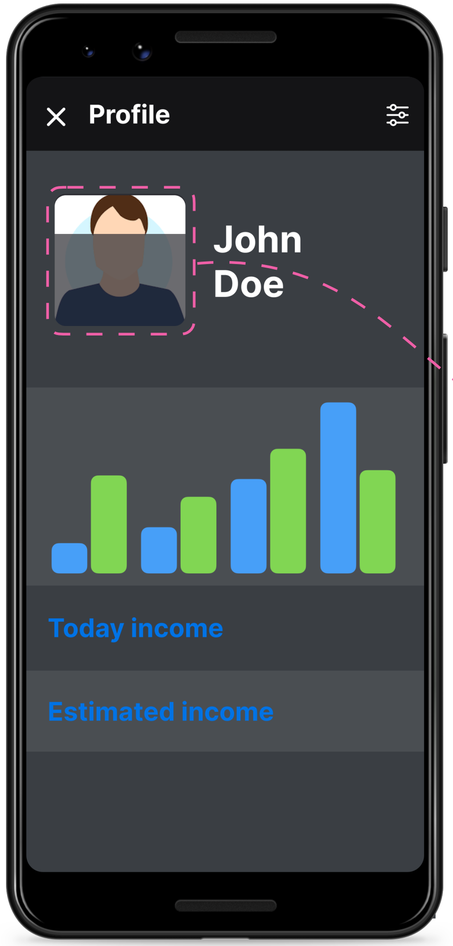
Что получаем?

- **Simple!**
- **Переиспользование компонентов**

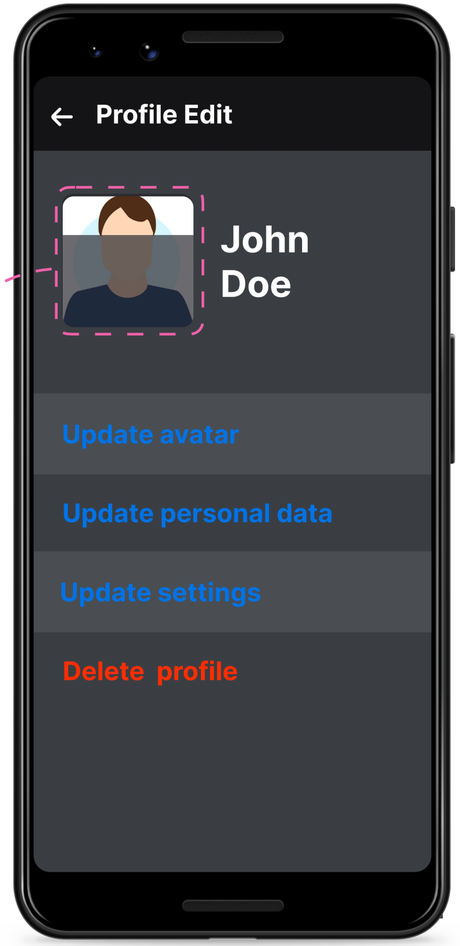
Переиспользование компонентов



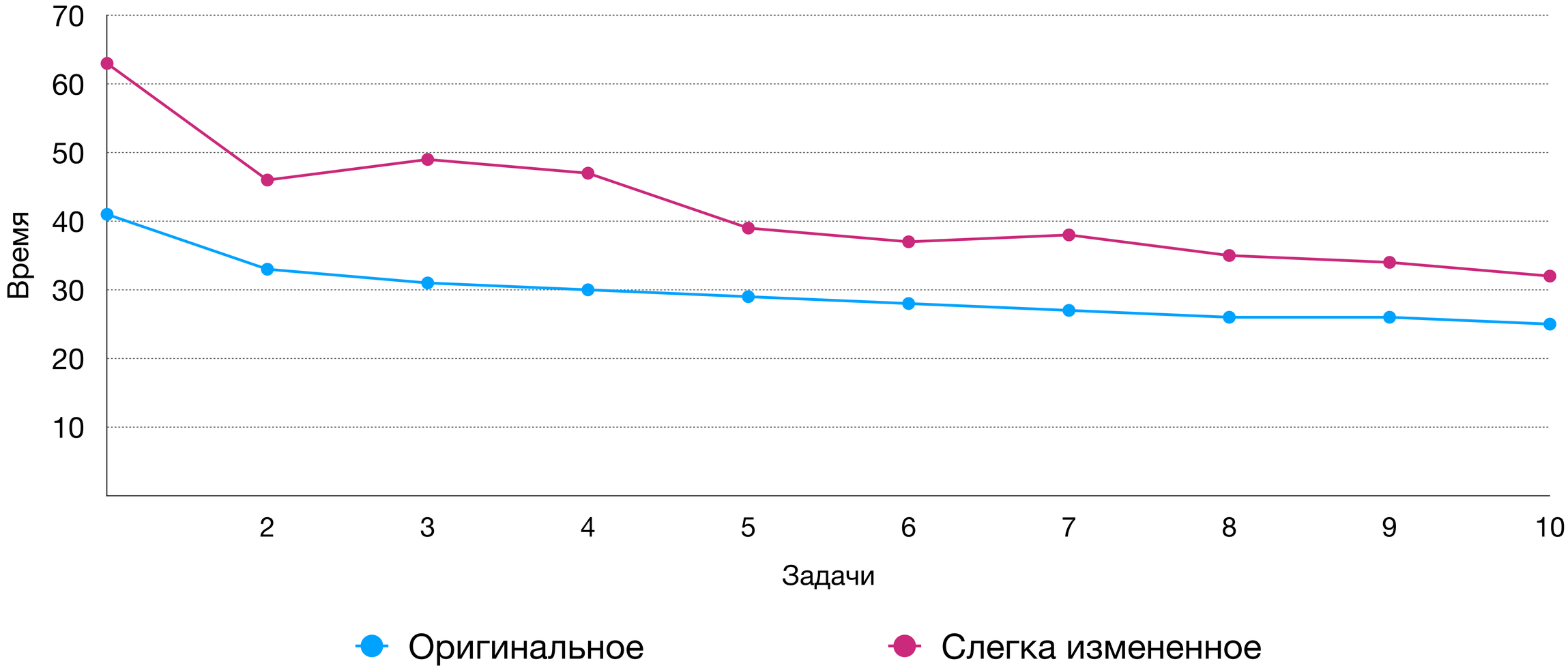
Переиспользование компонентов



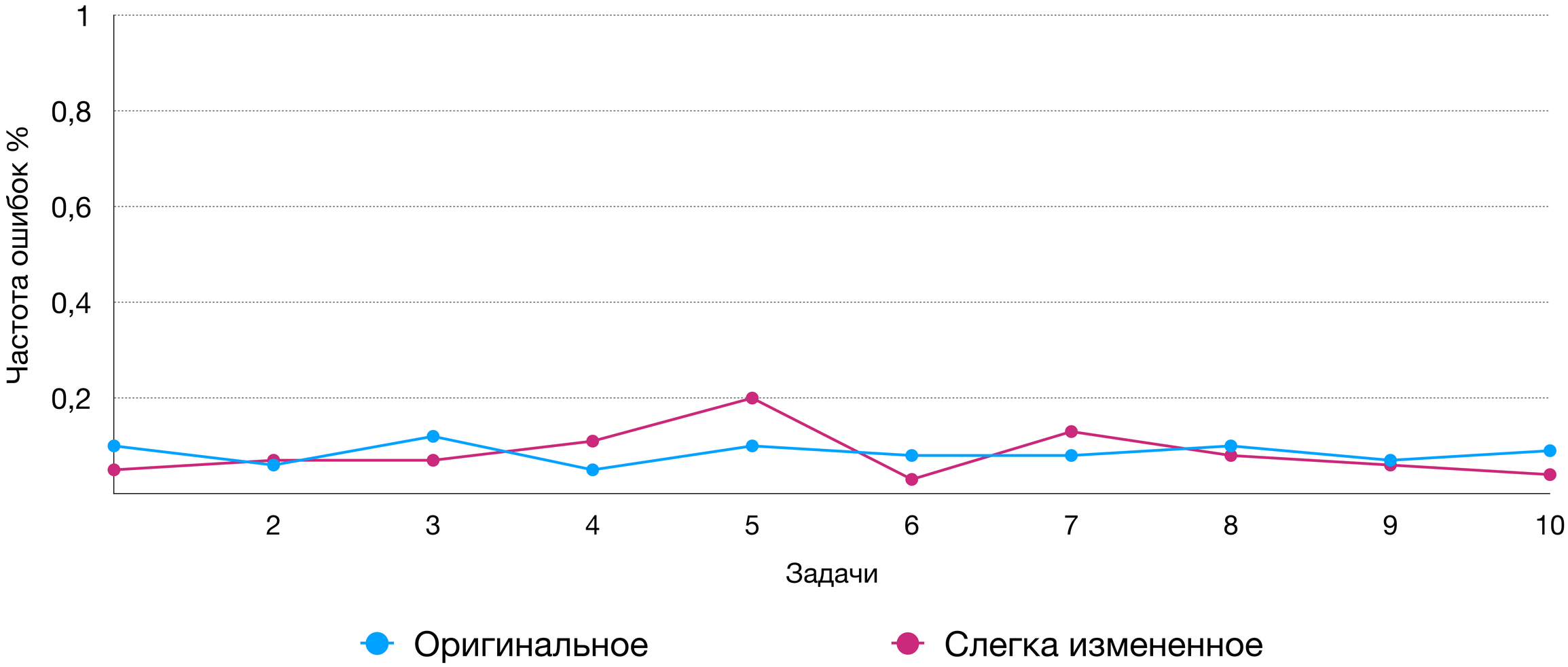

```
class UserAvatarFragment : EffektFragment<
    UserAvatar.State,
    UserAvatar.Msg,
    UserAvatar.Deps
>(R.layout.fragment_user_avatar)
```



Знакомство и привыкание



Знакомство и привыкание



Результаты

- **До 40% ускорения при реализации алгоритма => снижение сложности**
- **Эффект сильно зависит от алгоритма**
- **Эффект хрупок**
- **Слабо влияет на количество ошибок в реализации**

HeadlessFeaturesHost

```
interface HeadlessFeaturesHost {

  fun getHeadlessFeatures(): List<Class<out EffektViewModel<*, *, *>>>
}
```

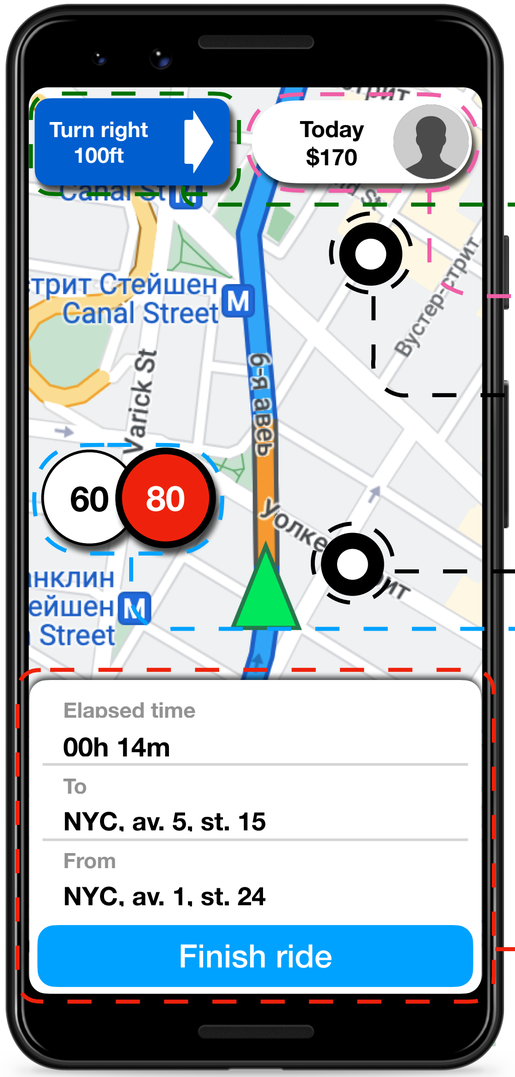
HeadlessFeaturesHost

```
class HeadlessFeaturesHostDelegate(
  lifecycleOwnerLiveData: LiveData<LifecycleOwner>,
  private val viewModelProvider: () -> ViewModelProvider,
  private val headlessFeaturesProvider: () -> List<Class<out EffektViewModel<*, *, *>>>,
) {

  init {
    lifecycleOwnerLiveData.observeForever {
      it?.lifecycleScope?.launchWhenCreated { attachHeadlessFeatures() }
    }
  }

  private fun attachHeadlessFeatures() {
    // pseudo: viewModels.forEach { it.onCreate }
  }
}
```

HeadlessFeaturesHost

```
fun HeadlessFeaturesHost.headlessFeaturesHostDelegate(hostFragment: Fragment) =
  HeadlessFeaturesHostDelegate(
    lifecycleOwnerLiveData = hostFragment.viewLifecycleOwnerLiveData,
    viewModelProvider = { ViewModelProvider(hostFragment) },
    headlessFeaturesProvider = this::getHeadlessFeatures
  )

fun HeadlessFeaturesHost.headlessFeaturesHostDelegate(hostActivity: AppCompatActivity) =
  HeadlessFeaturesHostDelegate(
    lifecycleOwnerLiveData = MutableLiveData(hostActivity),
    viewModelProvider = { ViewModelProvider(hostActivity) },
    headlessFeaturesProvider = this::getHeadlessFeatures
  )
```

Визуальный + логический признак




```
class NavigationWidgetFragment : EffektFragment<
    NavigationWidget.State,
    NavigationWidget.Msg,
    NavigationWidget.Deps
>(R.layout.fragment_widget_navigation)
```

```
class ProfileWidgetFragment : EffektFragment<
    ProfileWidget.State,
    ProfileWidget.Msg,
    ProfileWidget.Deps
>(R.layout.fragment_widget_profile)
```

```
class HeadlessPoiViewModel(deps: HeadlessPoi.Deps):
   EffektViewModel<
    HeadlessPoi.State,
    HeadlessPoi.Msg,
    HeadlessPoi.Deps
>(HeadlessPoi::update, deps)
```

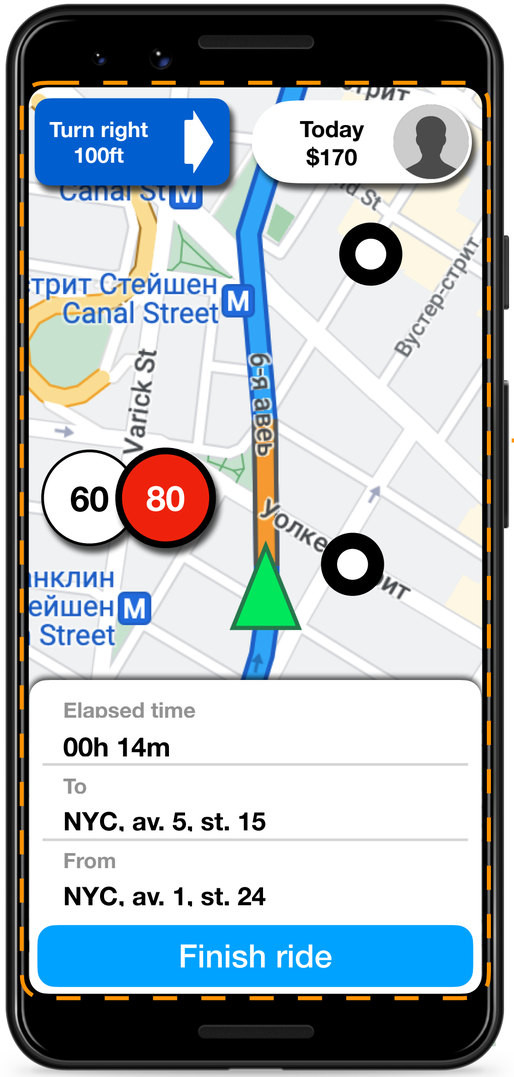
```
class SpeedWidgetFragment : EffektFragment<
    SpeedWidget.State,
    SpeedWidget.Msg,
    SpeedWidget.Deps
>(R.layout.fragment_widget_speed)
```

```
class RidePanelFragment : EffektFragment<
    RidePanel.State,
    RidePanel.Msg,
    RidePanel.Deps
>(R.layout.fragment_panel_ride)
```

RideFragment

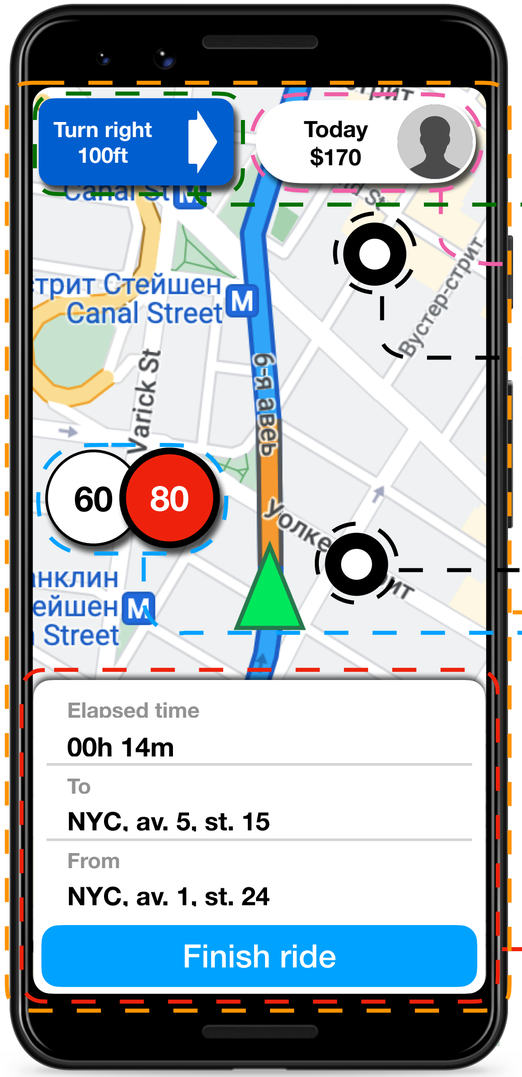
```
override fun getNestedFragments(): NestedFragments =
  mapOf(
    NavigationWidgetFragment::class.java to binding.navigationWidgetContainer,
    RidePanelFragment::class.java to binding.ridePanelContainer,
    ProfileWidgetFragment::class.java to binding.profileWidgetContainer,
    SpeedWidgetFragment::class.java to binding.speedWidgetContainer,
  )

override fun getHeadlessFeatures(): List<Class<out EffektViewModel<*, *, *>>> =
  listOf(HeadlessPoiViewModel::class.java)
```

Сложность до



7S-8T

Сложность после



1S-1T

1S-2T

1S-2T

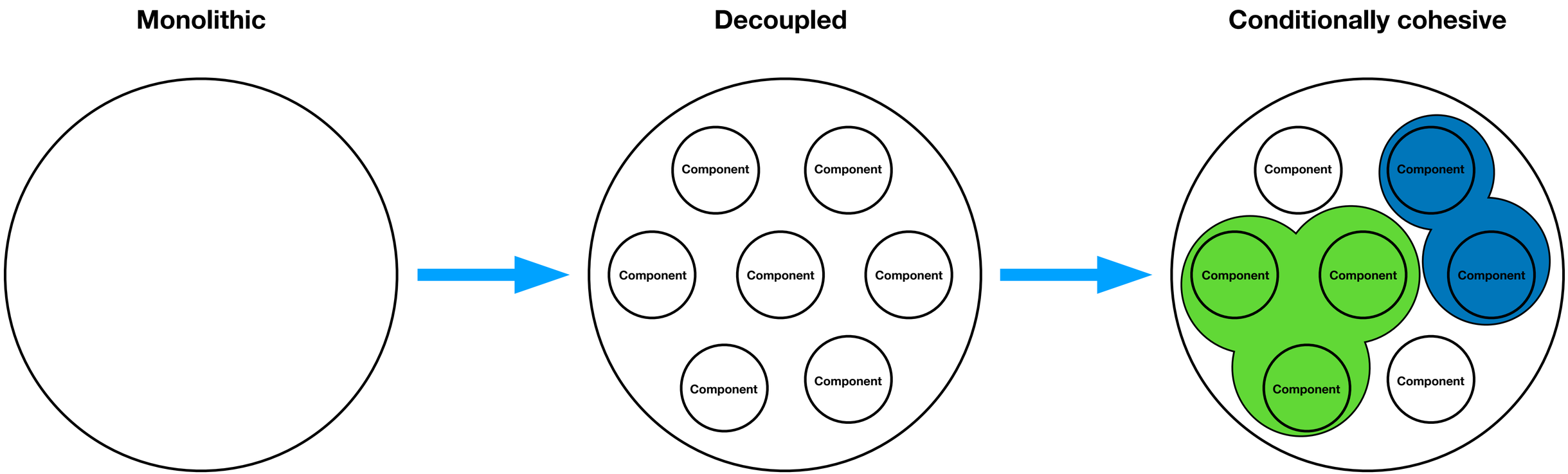
3S-4T

1S-1T

1S-2T

Выводы

Управление энтропией



Что получаем?

- **Simple!**
- **Управление энтропией**
- **Естественная модуляризация**
- **Переиспользование компонентов**

Чем платим?

- **Многословность**
- **Накладные расходы на связывание фич**
- **Статическая конфигурация**
- **Производительность?**

Простота — предпосылка надежности

Дейкстра Э.