エクストリームフラグメント UIプログラミング

あんざいゆき

あんざいゆき

- blog : Y.A.M の雑記帳
    - y-anz-m.blogspot.com
- twitter : @yanzm （やんざむ）
- Android女子部副部長
- uPhyca Inc. (株式会社ウフィカ)

- 3/16 発売
- Android 4.0 向けアプリ開発に必要な要素いっぱい

デザインっ設計

Button

TextView

~~Fragment~~

widget

Application

Activity

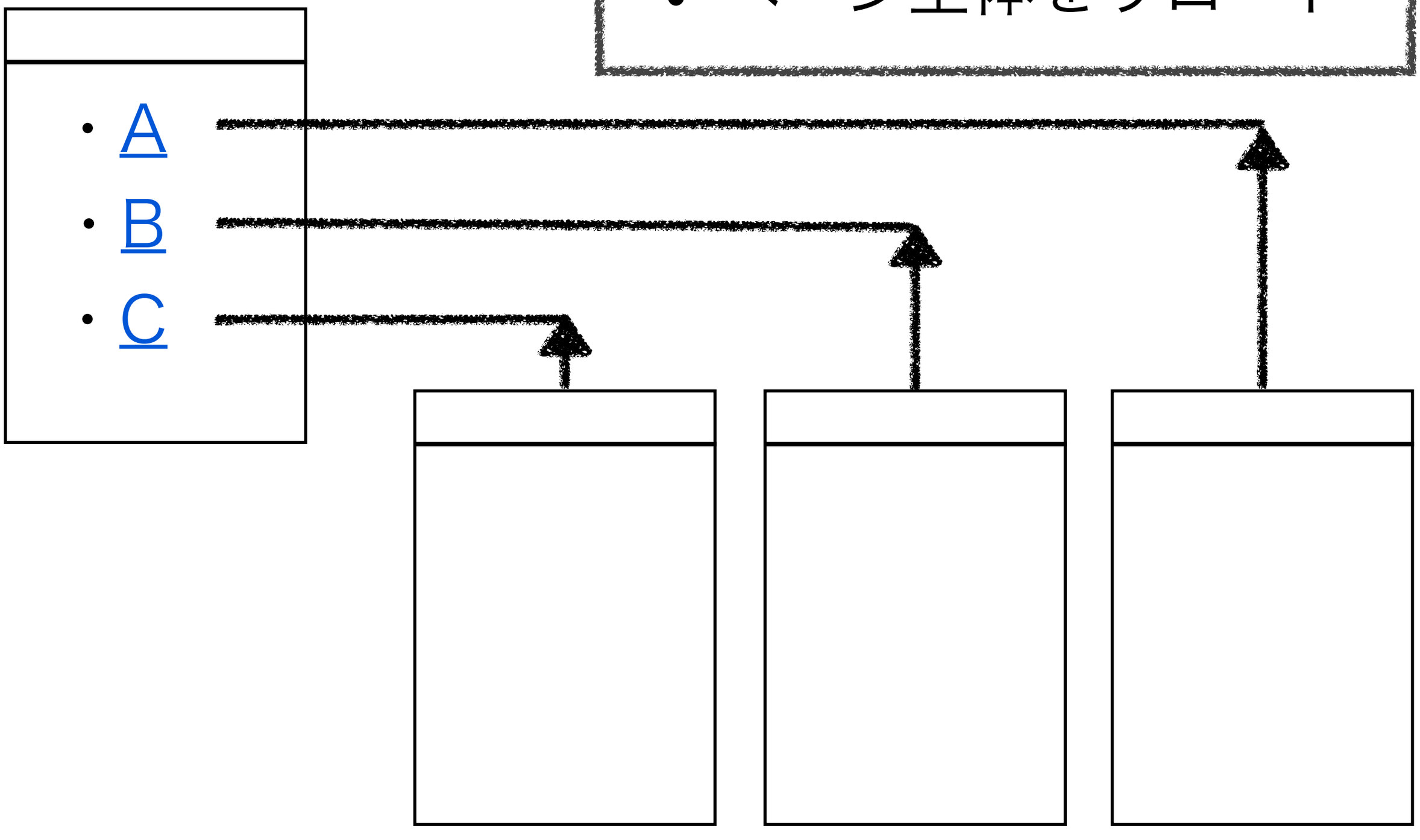
Fragment

設計

ちょっと web の話

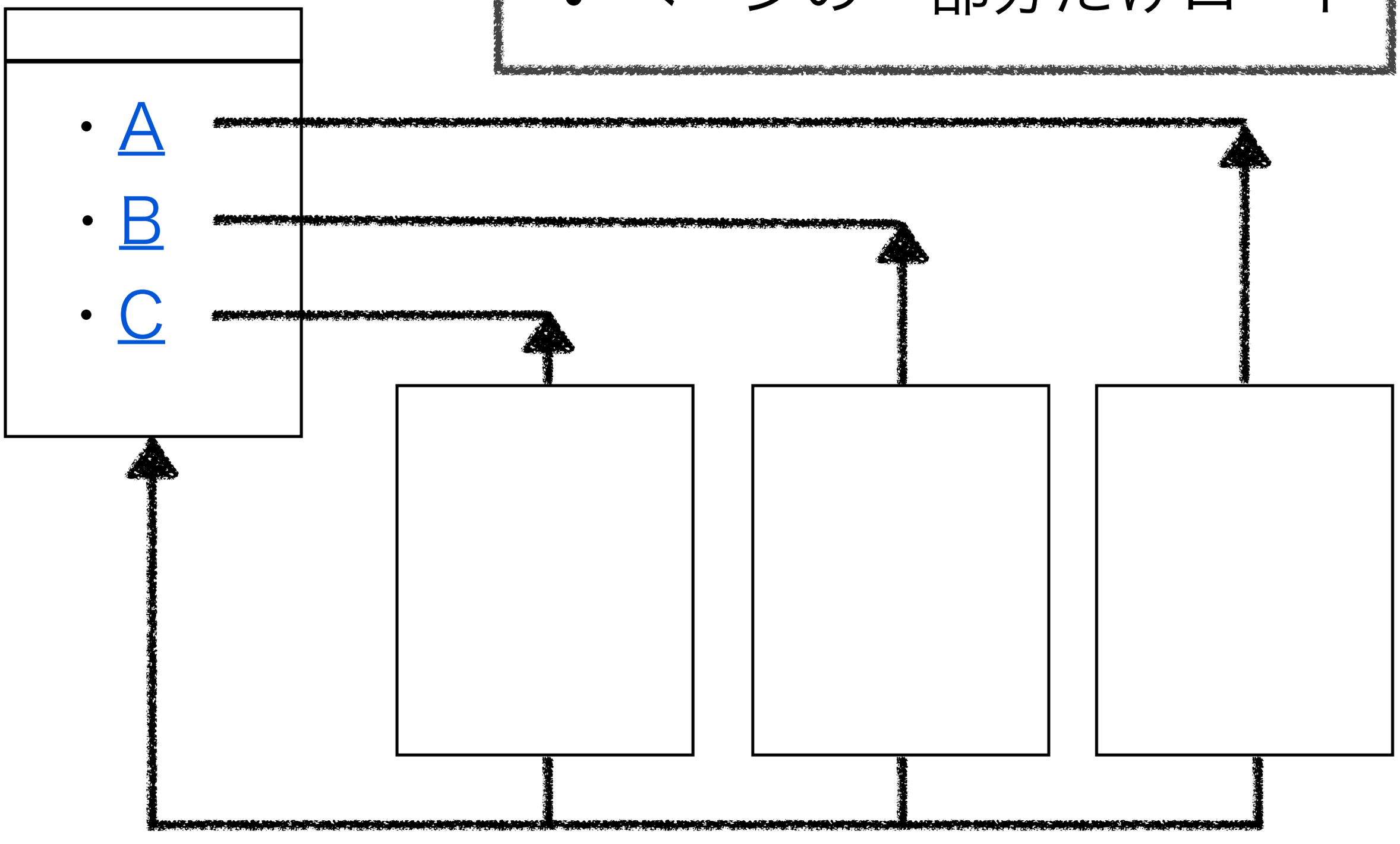
Ajax 以前

- 1画面 1html
- ページ全体をリロード



Ajax 後

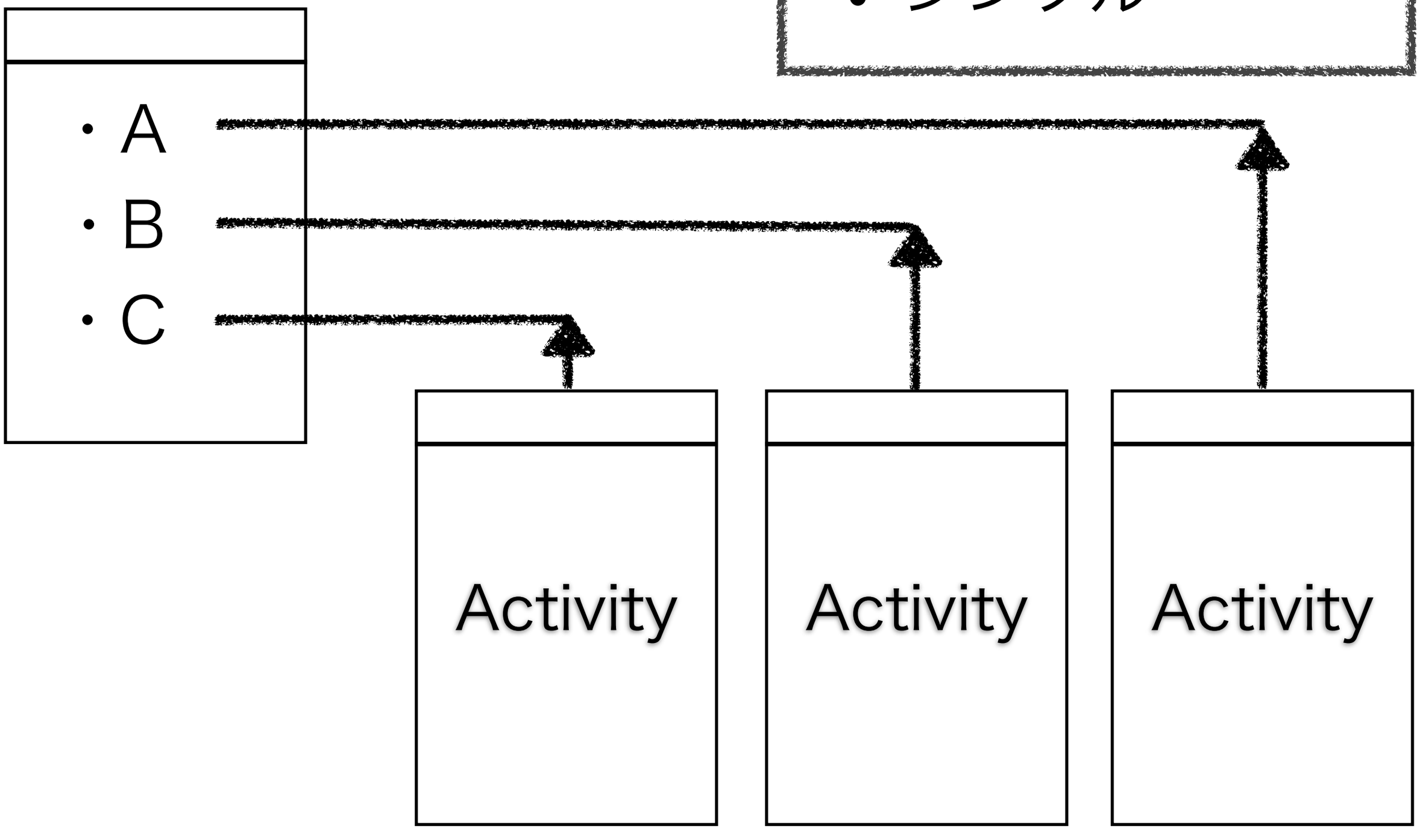
- 複数画面 1html
- ページの一部分だけロード



Fragment の
キモは同じ

2.x 時代

- 1画面 1Activity
- シンプル

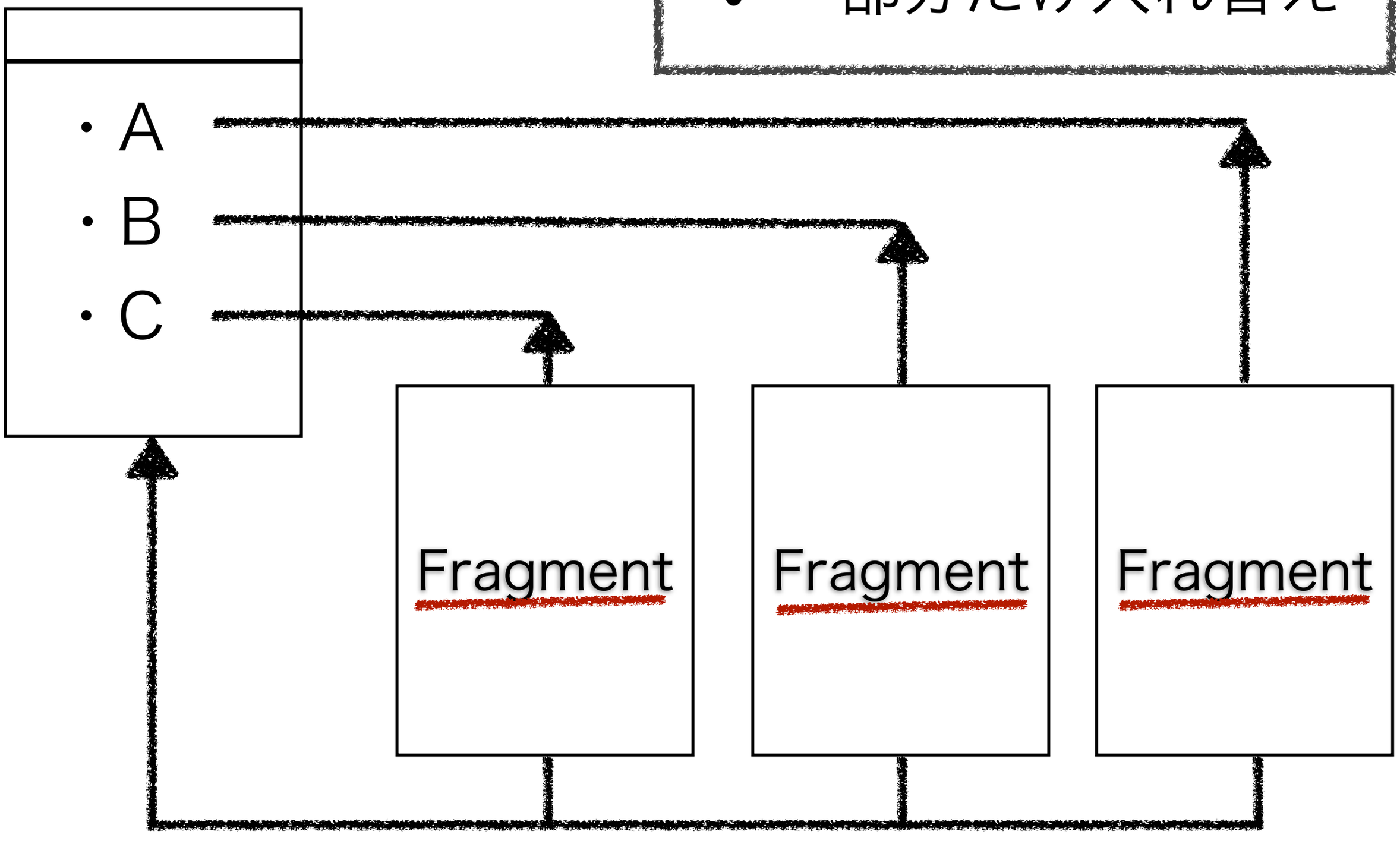


画面遷移に時間がかかる

- Ajax 以前のwebのページ全体ロードと同じ問題
- これまでの対処法
  - タブを使う
    - タブA からタブB の中がみれない
  - View の Visibility の変化を利用する
    - Activity で管理する View が増える

3.0 以降

- 複数画面 1Activity
- 一部分だけ入れ替え



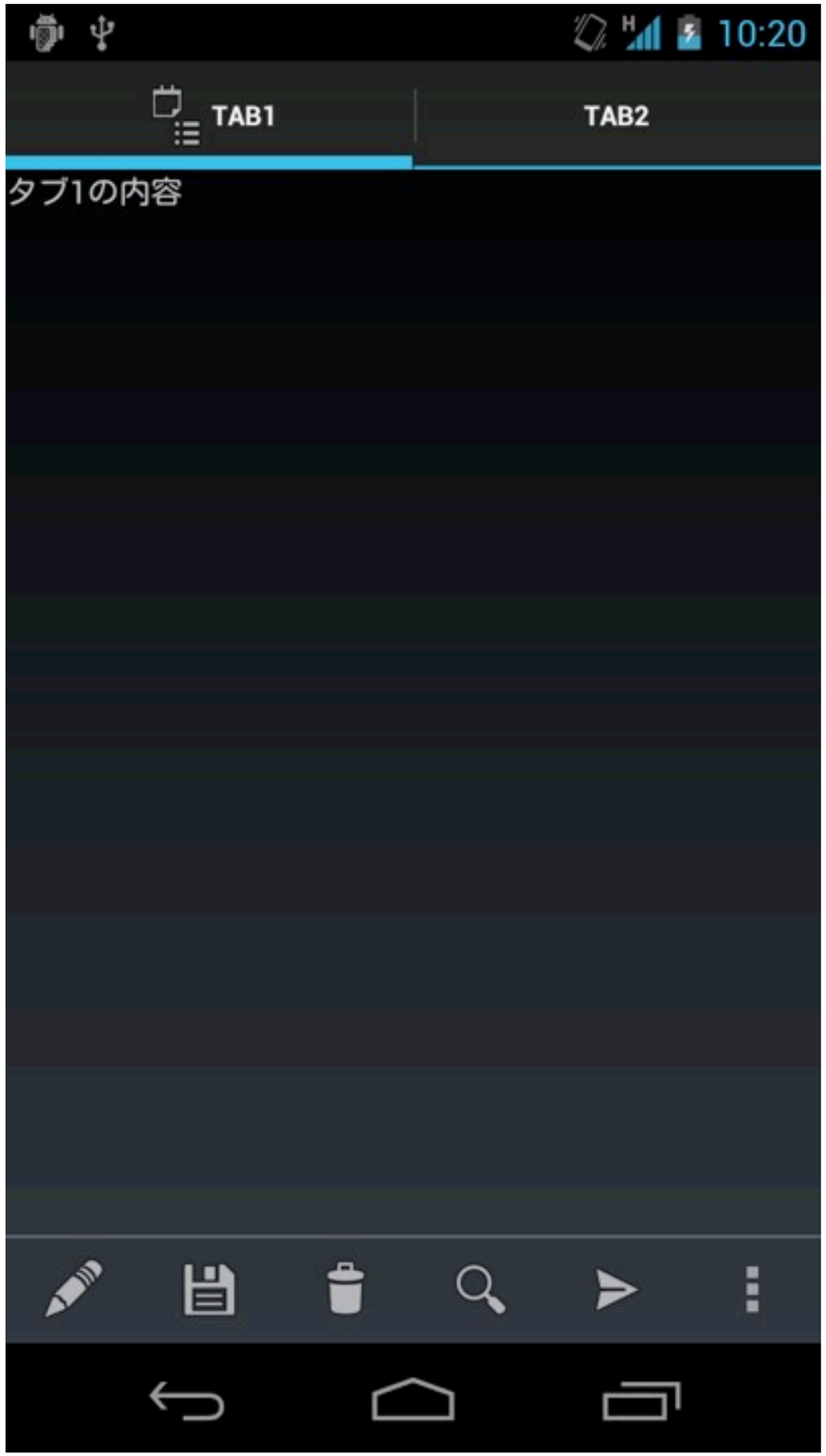
ハンドセットでの

Fragment の使いどころ

タブレイアウト

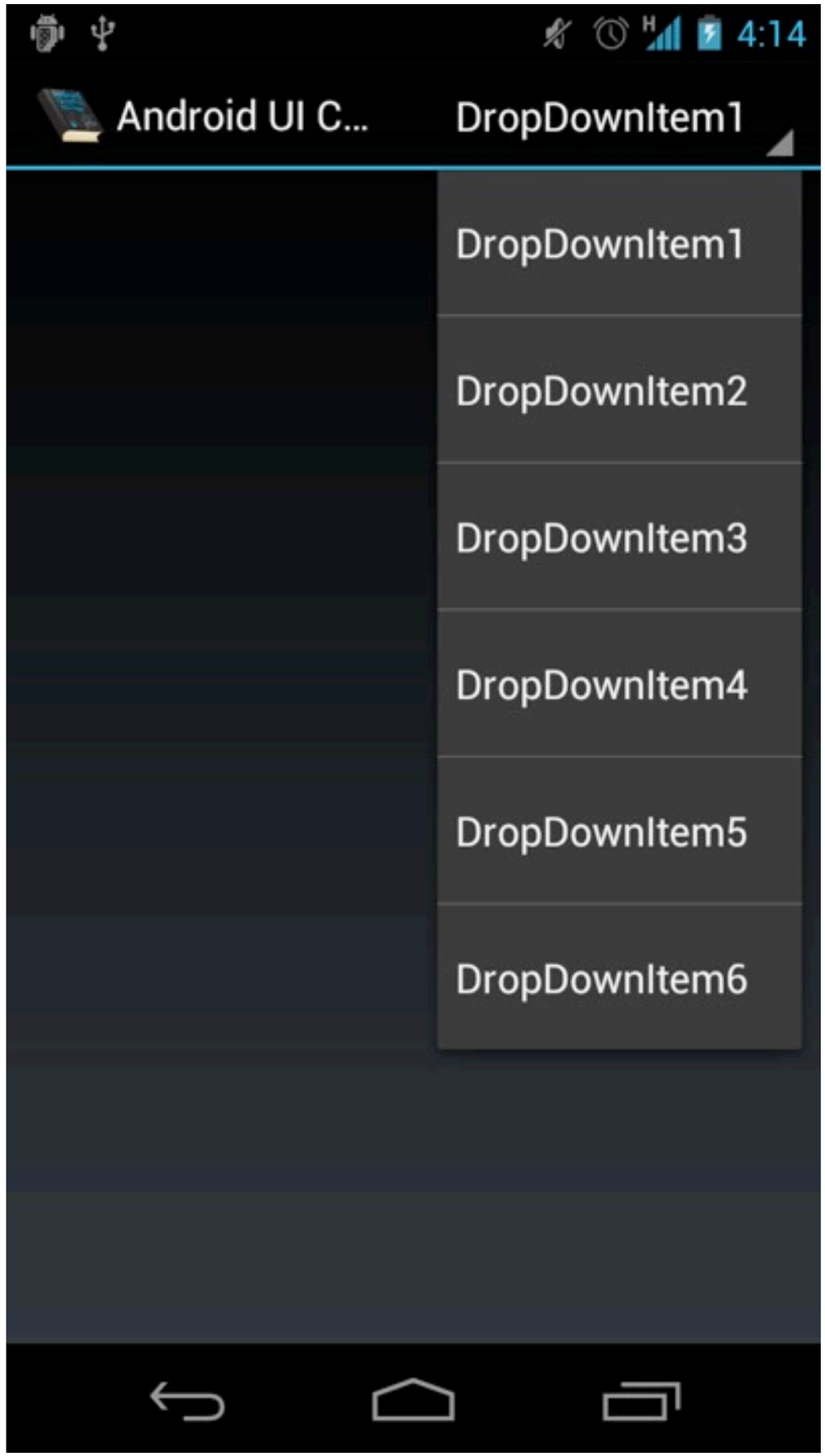
- x : タブの中が Activity
- o : タブの中が Fragment

- タブの中が Fragment
  → Activity, Fragment間でのやり取りが楽



ドロップダウン

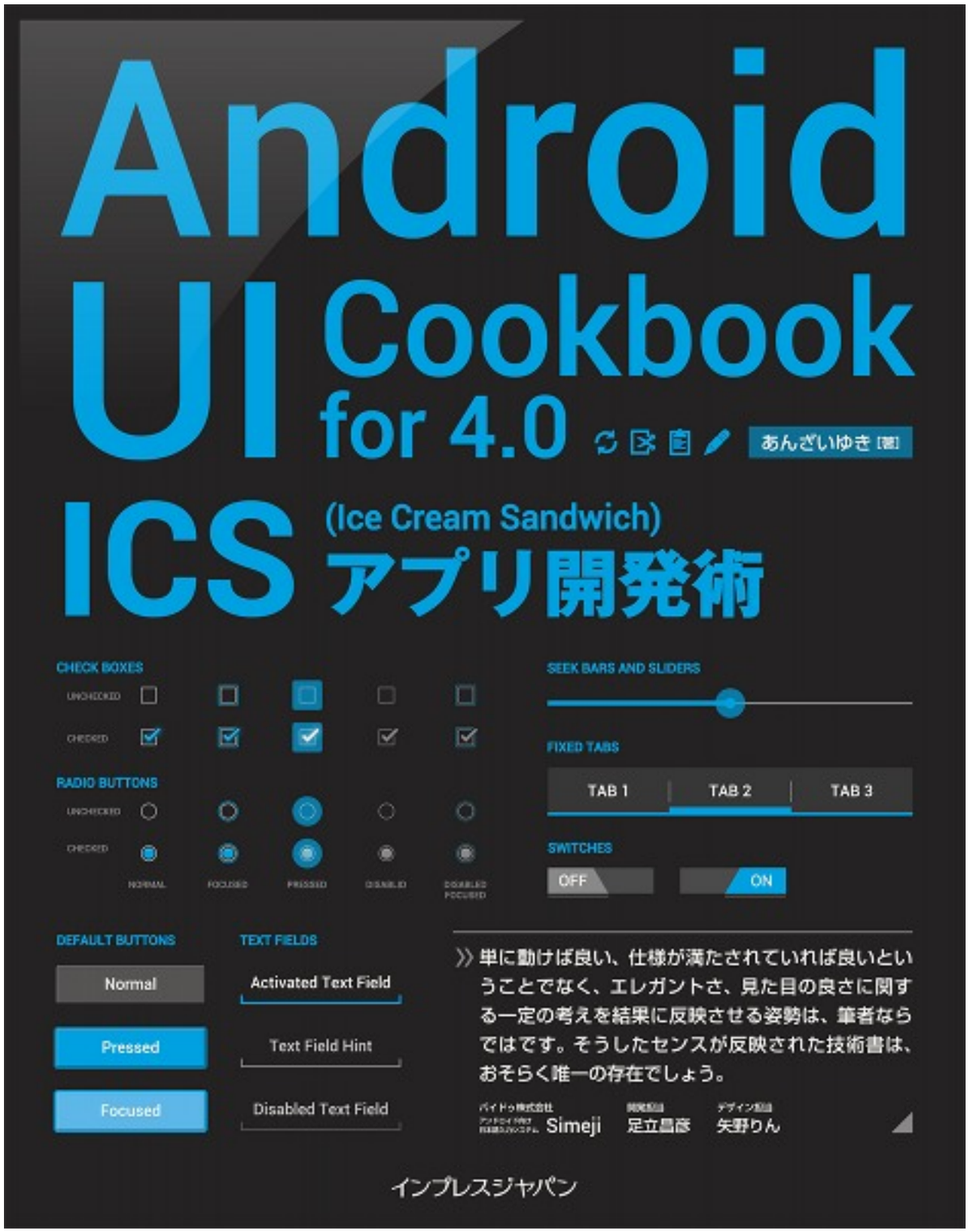
- ドロップダウンで画面切り替え
- Fragment を入れ替える



ダイアログ

- DialogFragment
- ライフサイクルを管理して
  もらえる
- setArgument() を活用

- 2章と3章をよみましょう。



設計について

Fragment を使った設計をするときに考えること

- Activity の単位
- Fragment の単位
- ライフサイクル

Activity の単位

「1画面 1Activity でなくてもいいなら
どこまでを 1Activity にするの？」

Activity の単位

- 機能・処理としてその画面で完結している
- ショートカットなど直接起動される入り口になっている
- 暗黙的 Intent を受取る入り口になっている

Fragment の単位

「どこまでを 1Fragment にするの？」

Fragment の単位

- データの取得・加工・表示
- 画面全体ではなく、一分だけ切り替えるときの領域

ライフサイクル

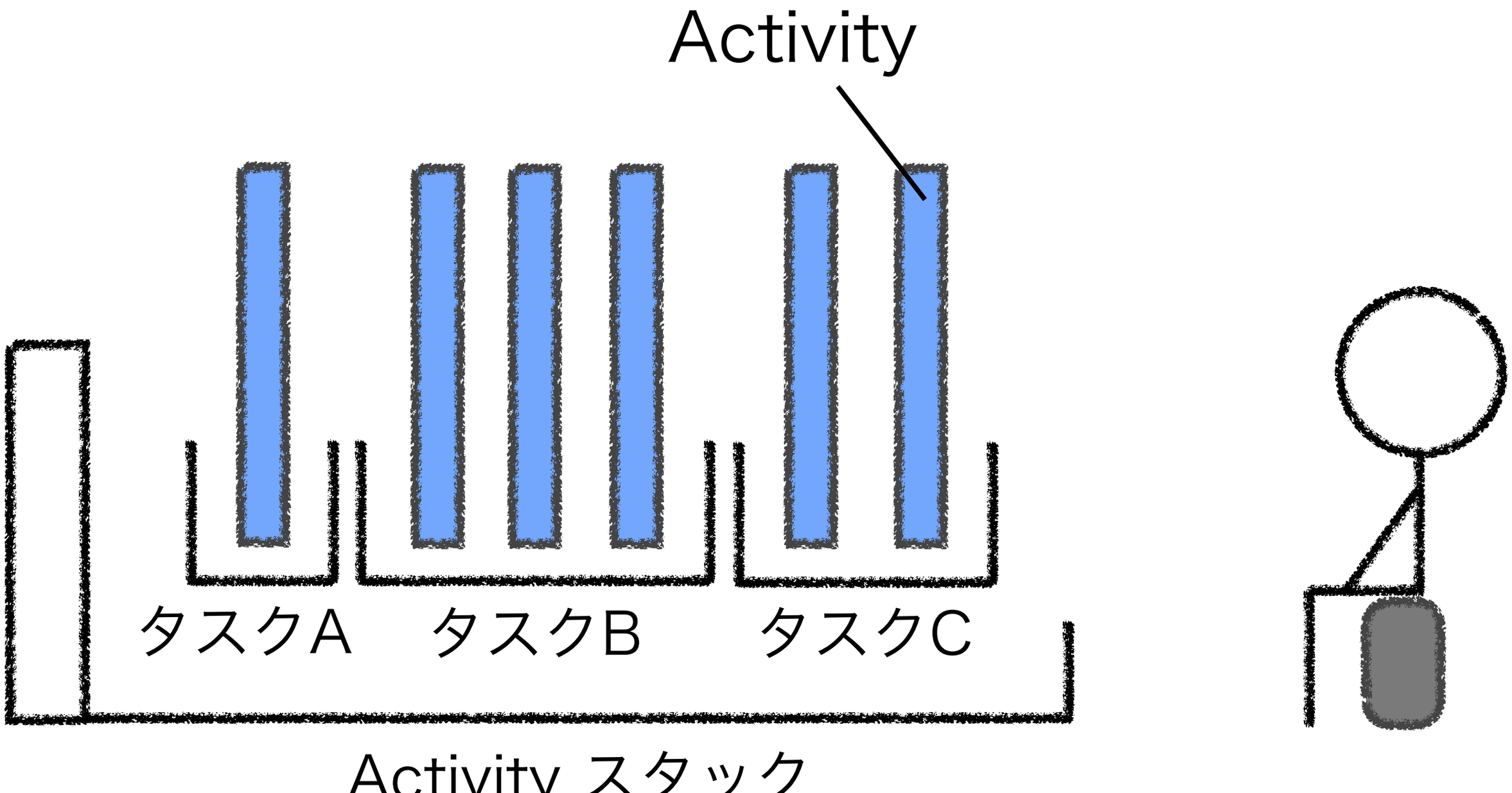
とスタック

これが一番重要

押さえるポイント

- Activity スタック
- Launch mode
- Activity のライフサイクル
- Fragment のライフサイクル
- Fragment の BackStack

Activity スタック

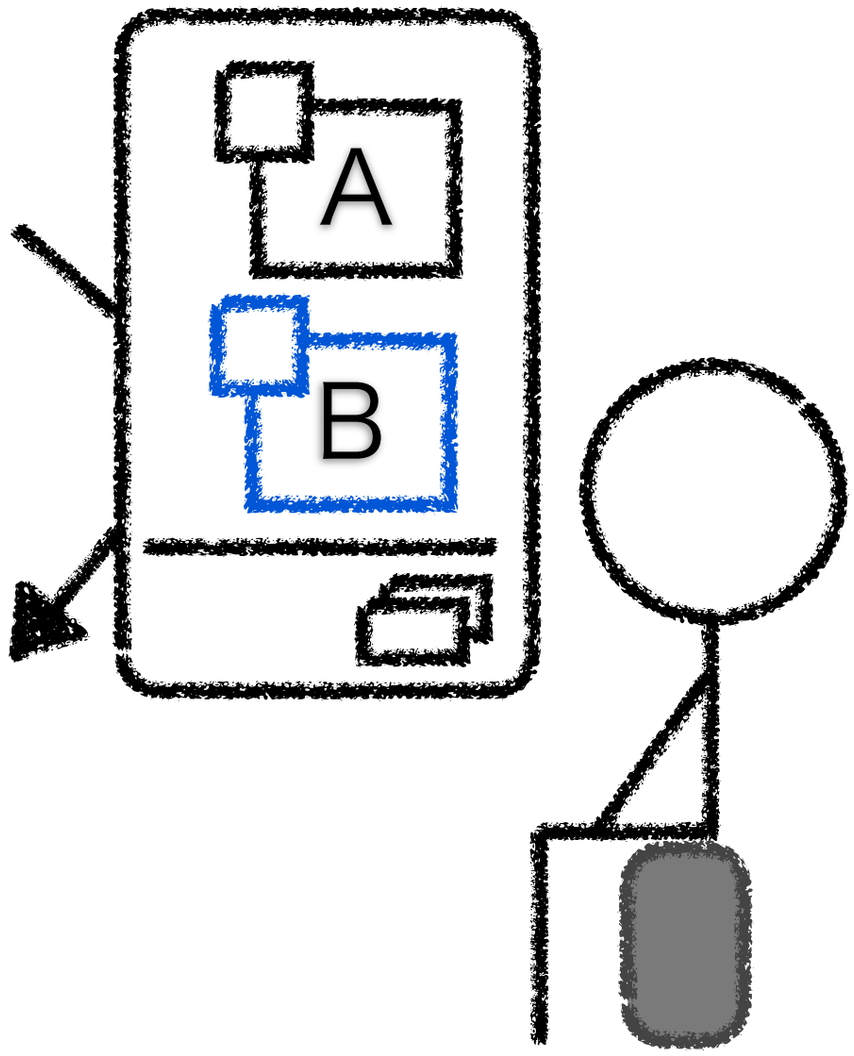


Activity

タスクA　タスクB　タスクC

タスクA　タスクC　タスクB

Activity スタック

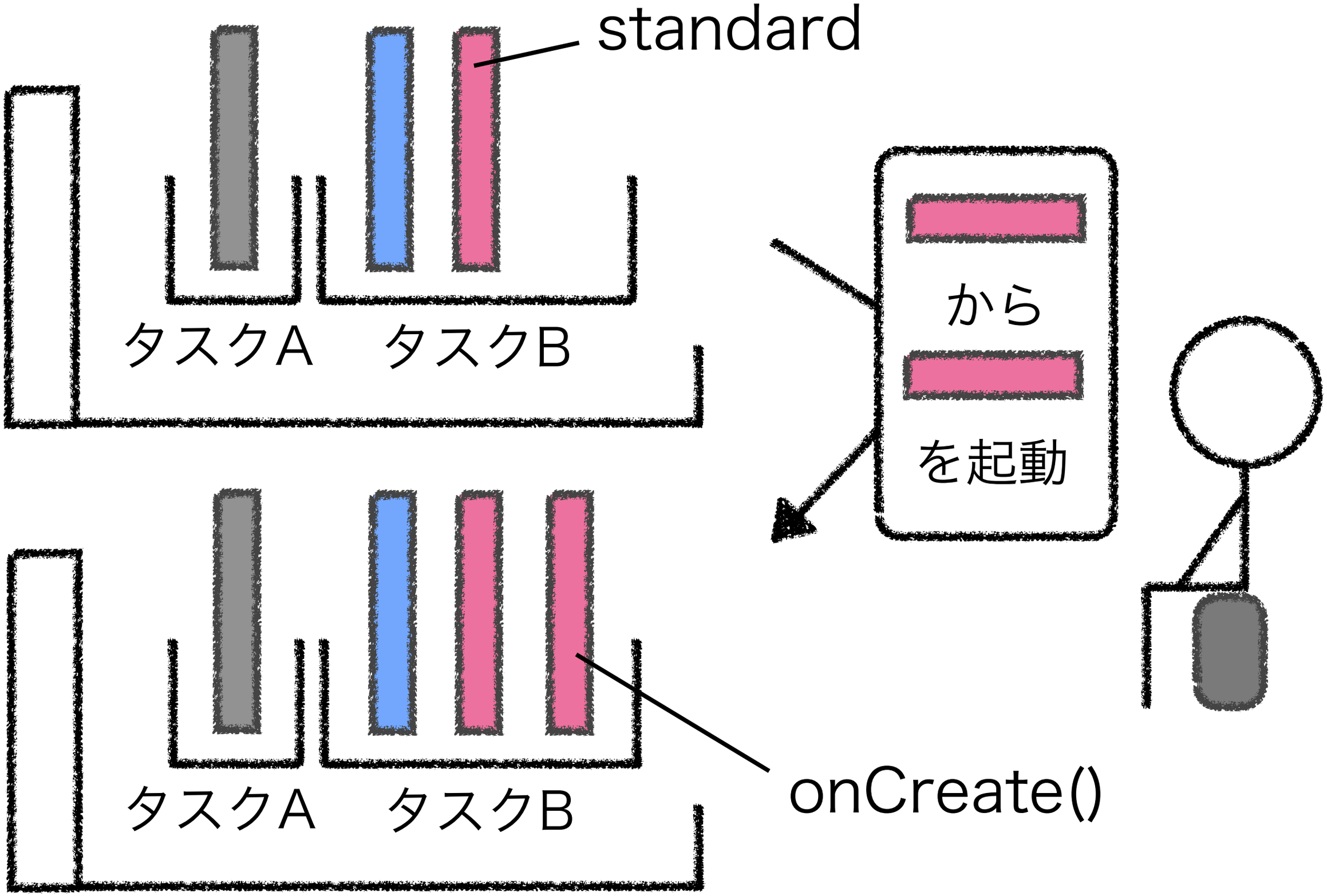


Launch mode

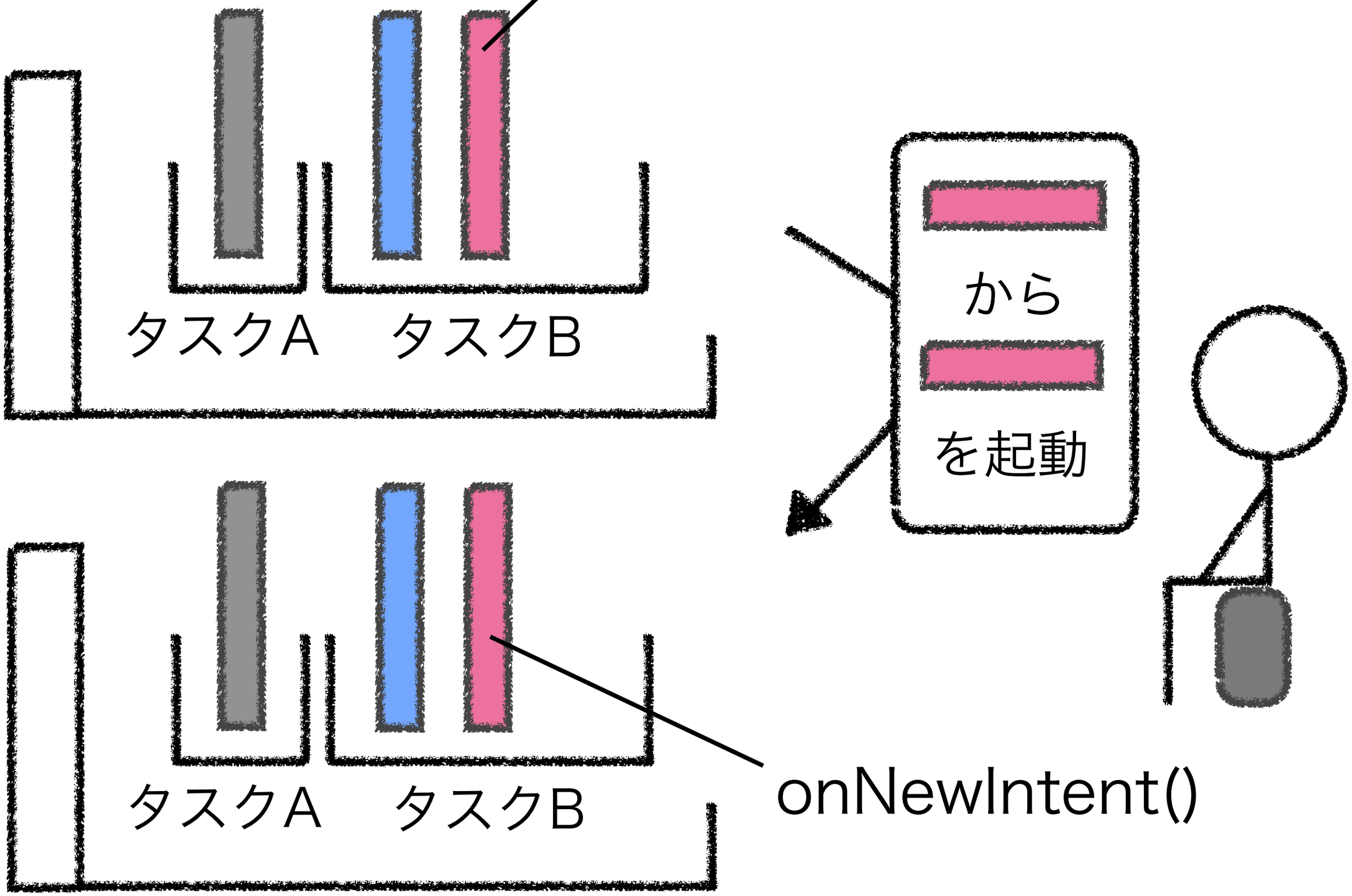
- Activity の起動時の振る舞いを決める
- standard
- singleTop
- singleTask
- singleInstance

launchMode	インテントに応答するアクティビティをどのタスクに保持するか	アクティビティのインスタンスを複数生成できるか	インスタンスのタスクに他のアクティビティを含めることができるか	クラスの新しいインスタンスを起動して新しいインテントを処理するかどうか
standard (default mode)	startActivity() を呼び出したタスクに保持 *1	アクティビティは複数回インスタンス化可能*2	タスクに複数のアクティビティを割り当てることが可能	新しいインテントに応答するときには必ず新しいインスタンスが作成
singleTop				クラスの既存のインスタンスが対象タスクのアクティビティ スタックの最上位にあれば、それを再利用して新しいインテントを処理 スタックの最上位にない場合は再利用されずに、新しいインスタンスが作成
singleTask	アクティビティが常にタスクのルート アクティビティになる 他のタスクの一部として起動されることはない	インスタンスは 1つに制限 アクティビティはタスクのルート		インスタンスは常に新しいインテントを処理
singleInstance			アクティビティは、そのタスク内の唯一のアクティビティとして単独で動作	

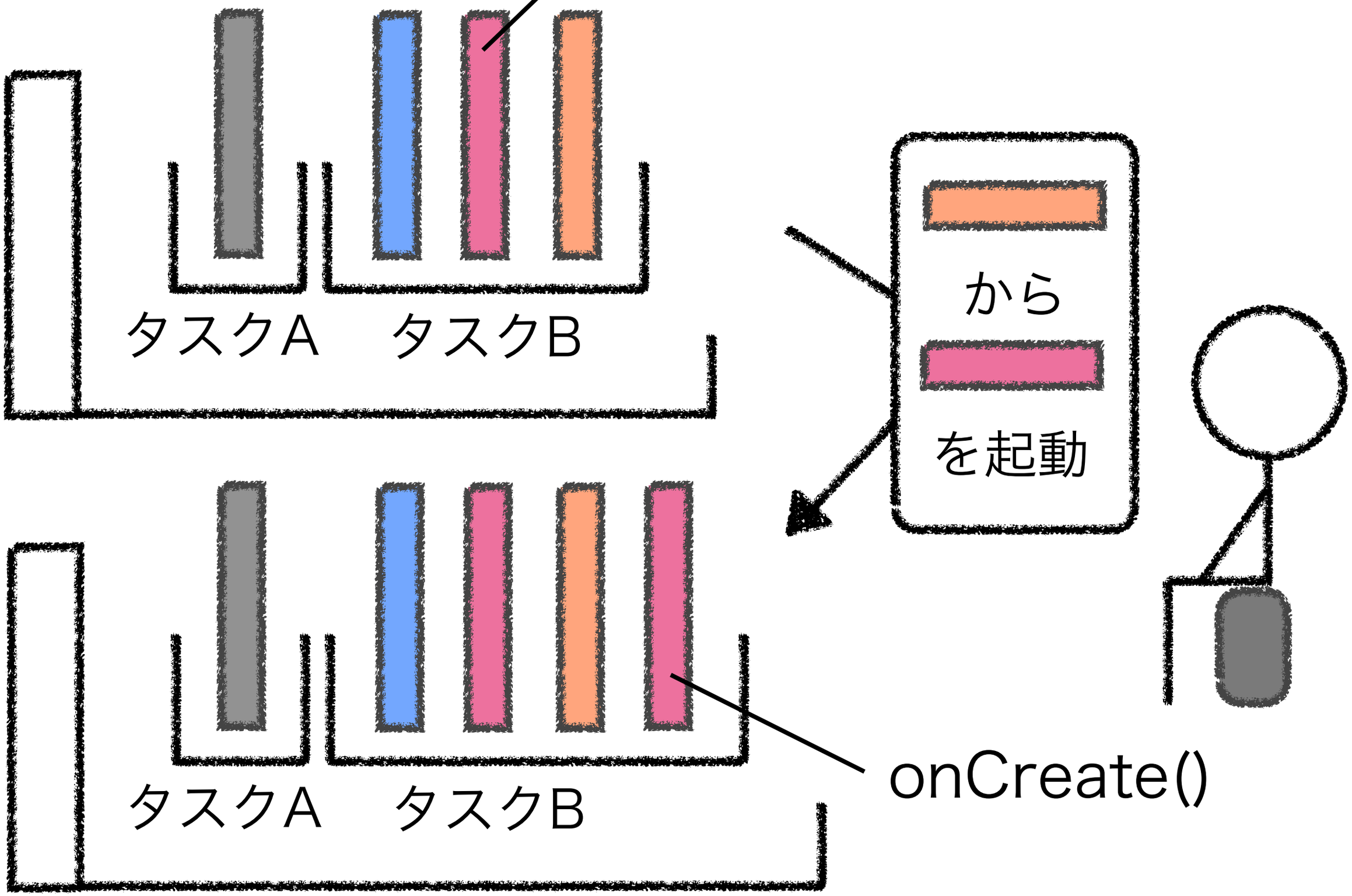
standard



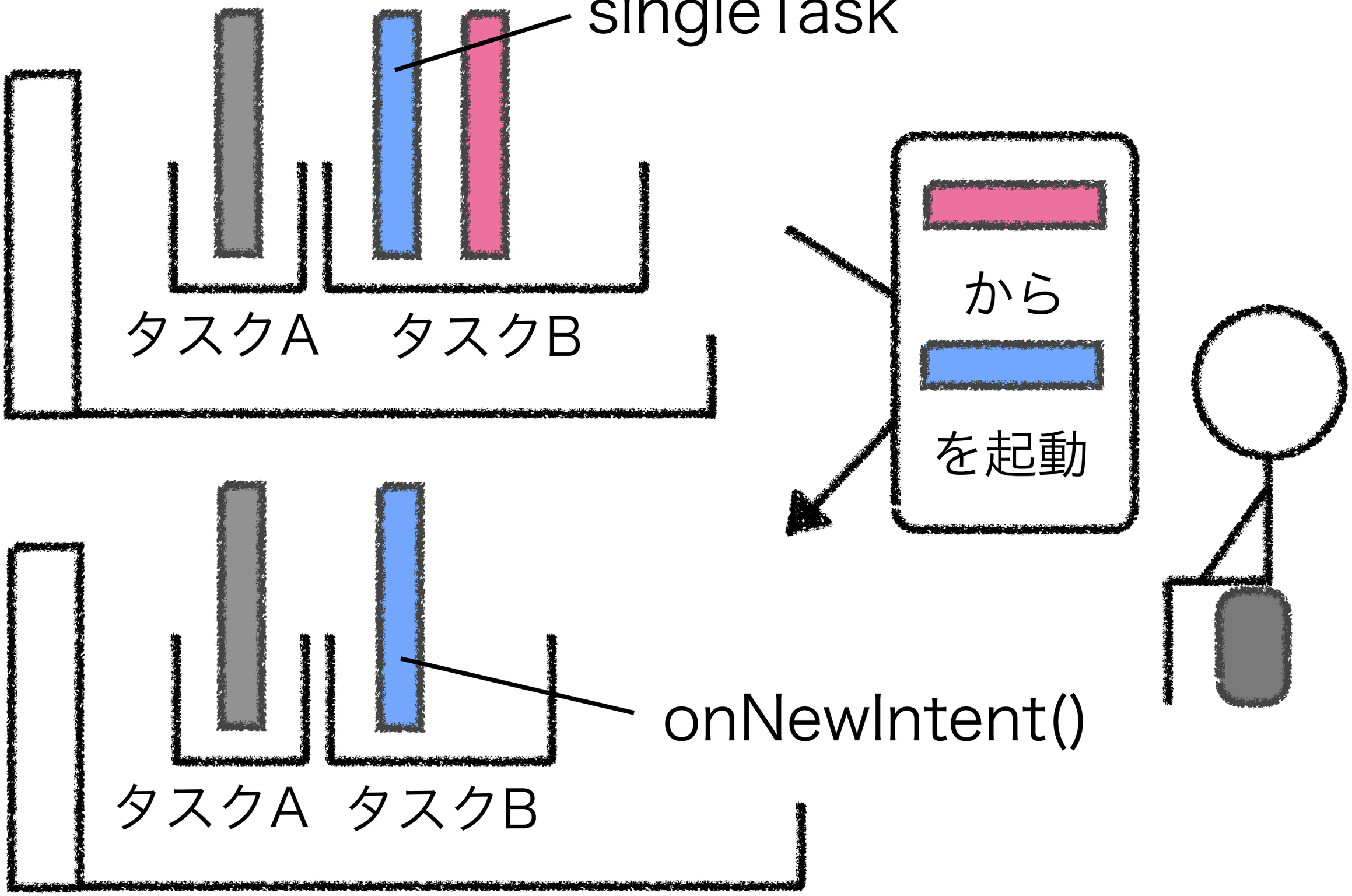
singleTop



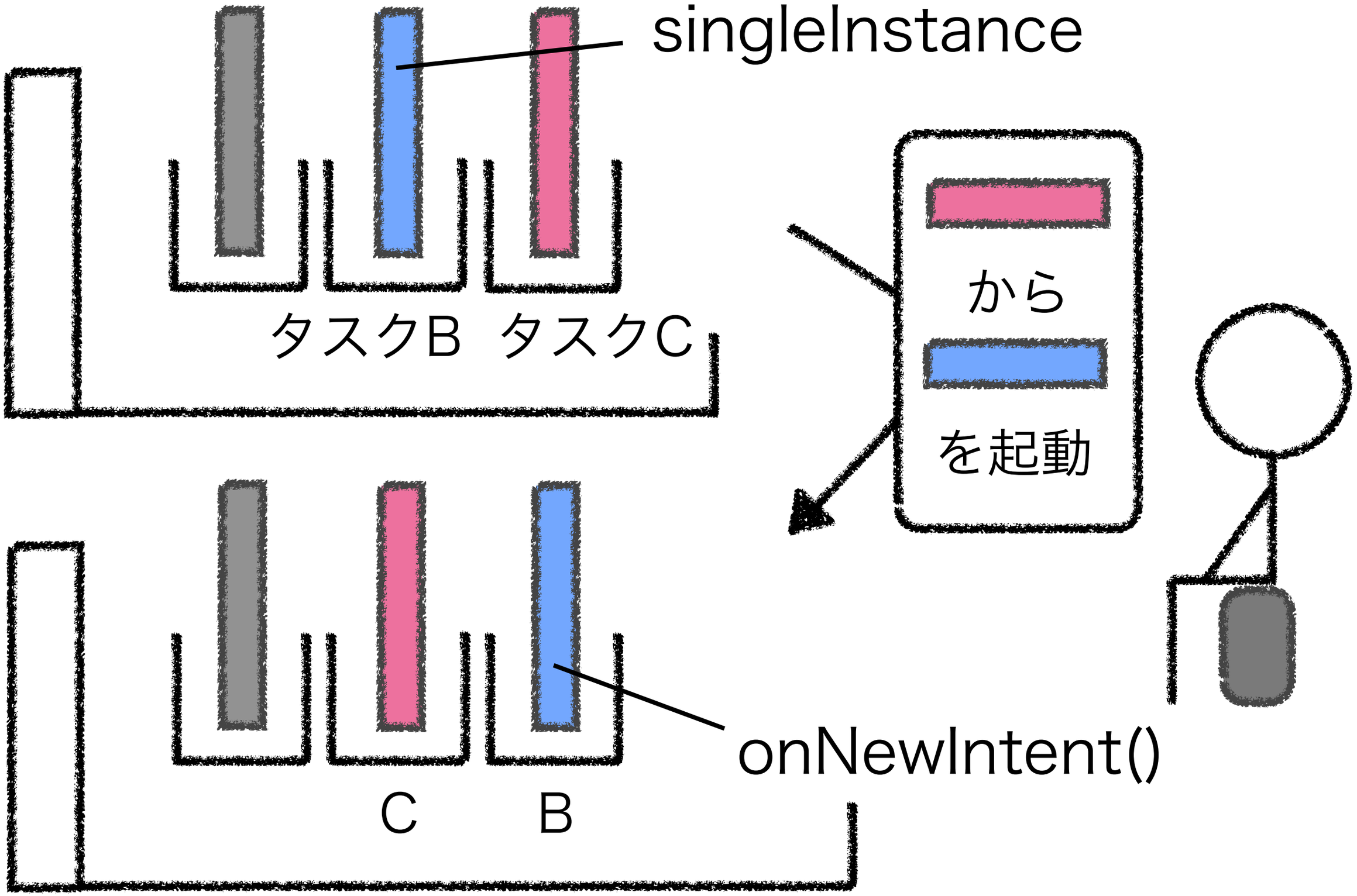
singleTop



singleTask



singleInstance



Activity のライフサイクル

おさらい

Activity 起動

onCreate()

onStart()

onResume()

Activity 稼働中

onPause()

onStop()

onDestroy()

Activity 終了

プロセスkill

メモリ不足

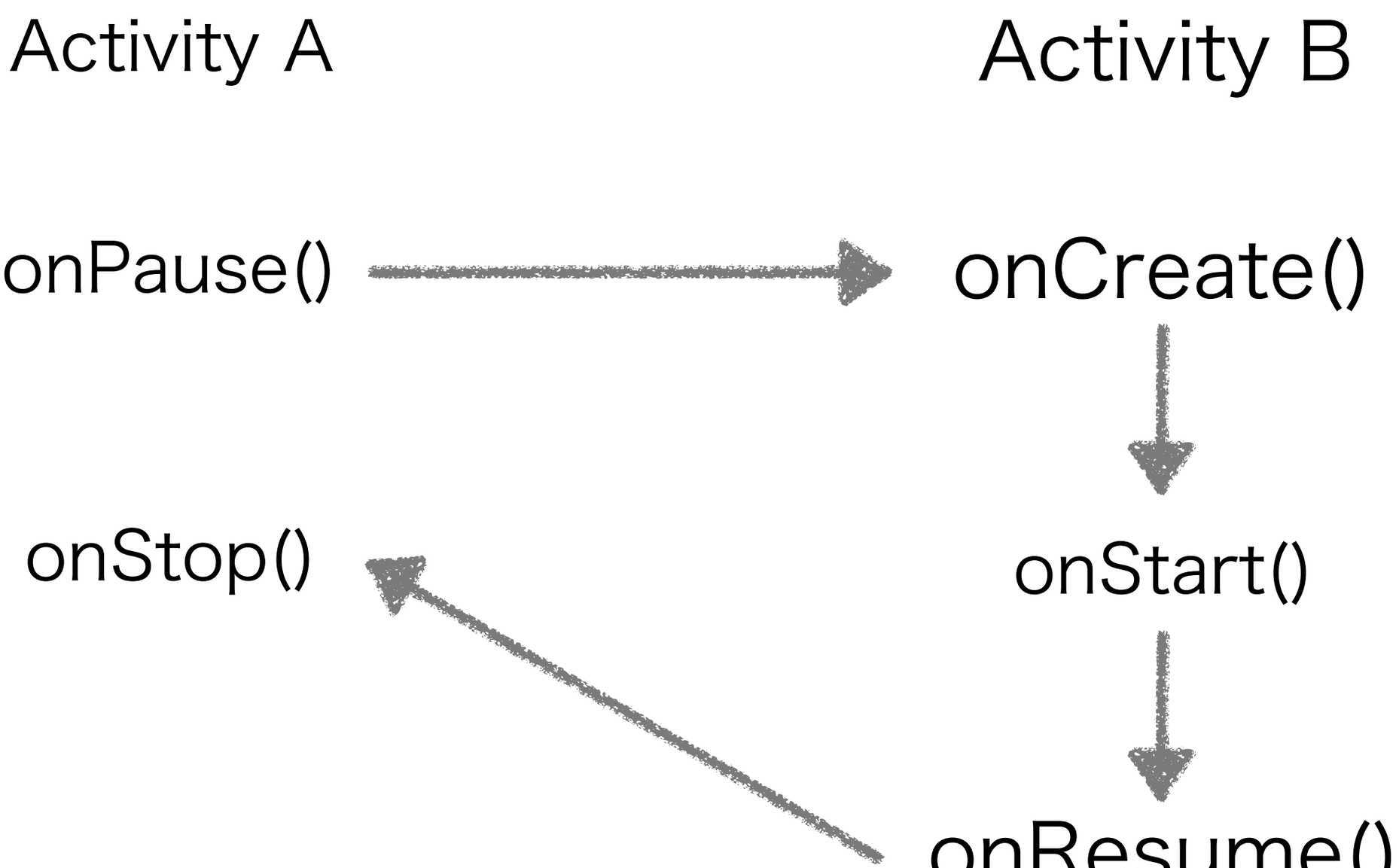
onRestart()

Activity が終了中 or
システムから破棄される

Activity が見えなく
なった

別のActivity が
foreground へ

Activity A から Activity B を起動するときのライフサイクル



ようやく Fragment

Fragment のライフサイクル

Fragment 追加

↓

onAttach()

↓

onCreate()

↓

onCreateView()

↓

onActivityCreated()

Activity の onCreate() が完了した

↓

onStart()

↓

onResume()

↘

Fragment がアクティブ

↗

onPause()

↑

onStop()

↑

onDestroyView()

View に関連するリソースの破棄が可能

↑

onDestroy()

↑

onDetach()

↑

Fragment 破棄

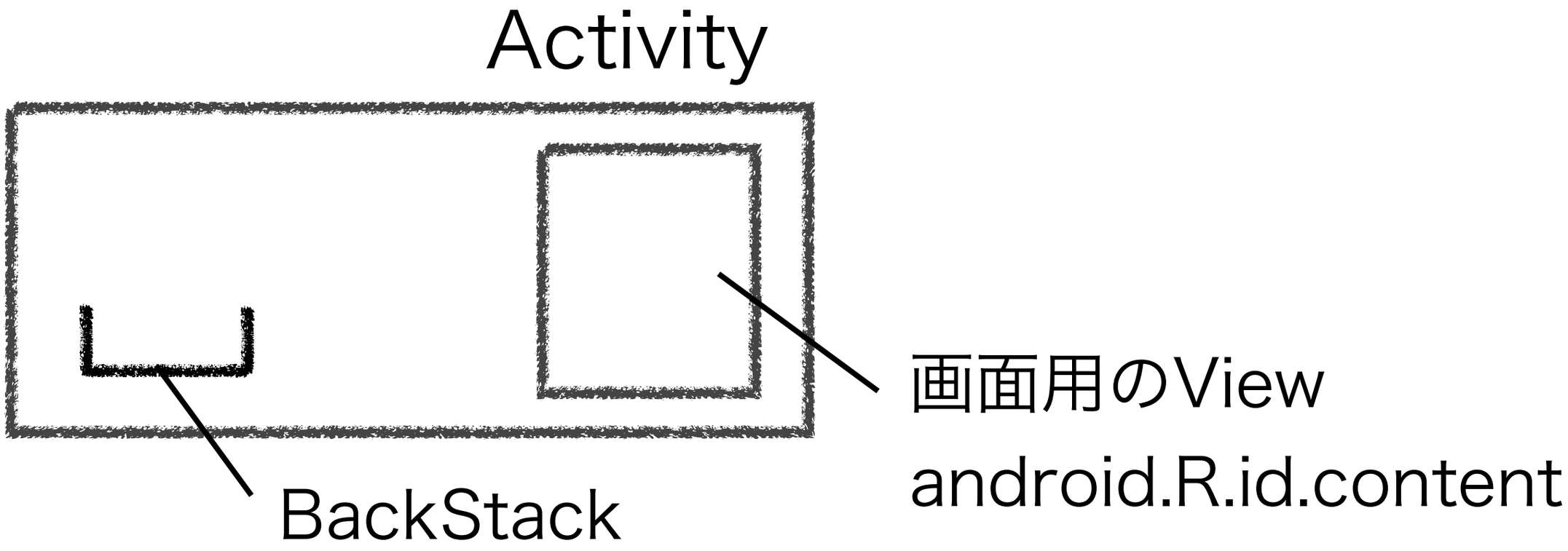
Fragment はライフサイクルの
コールバックが多い！

Fragment のライフサイクルは Fragment のトランザクションと関連

Fragment のトランザクション

- add(), remove()
- replace()
- attach(), detach()
- show(), hide()

詳しくみてみる。



add()

Fragment 追加

onAttach()

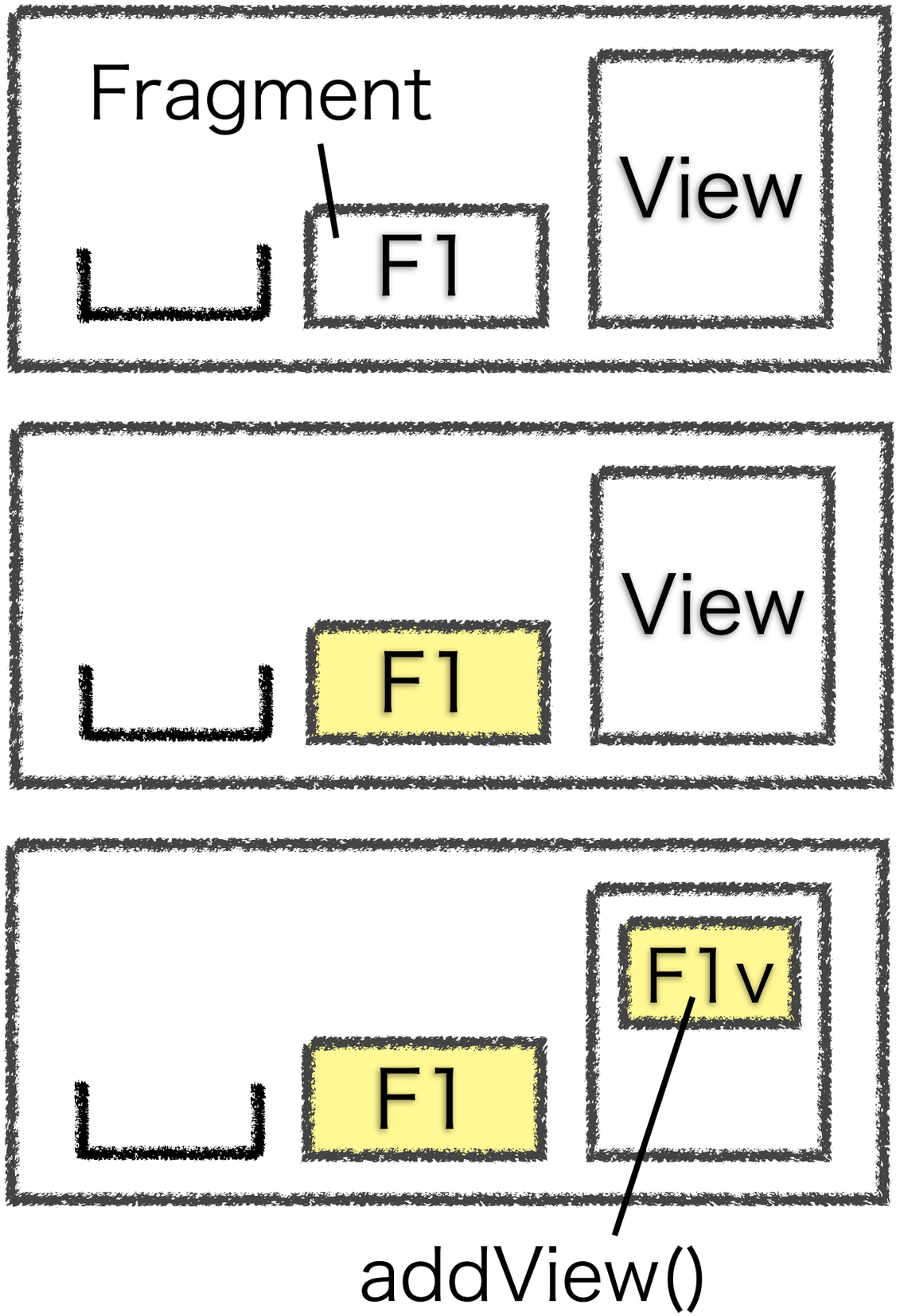
onCreate()

onCreateView()

onActivityCreated()

onStart()

onResume()



remove()

Fragment 破棄

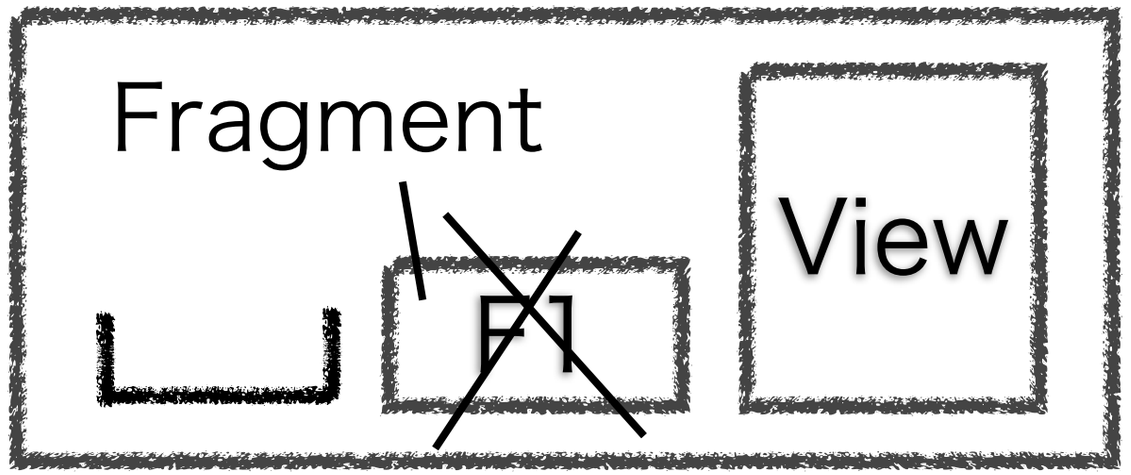
onDetach()

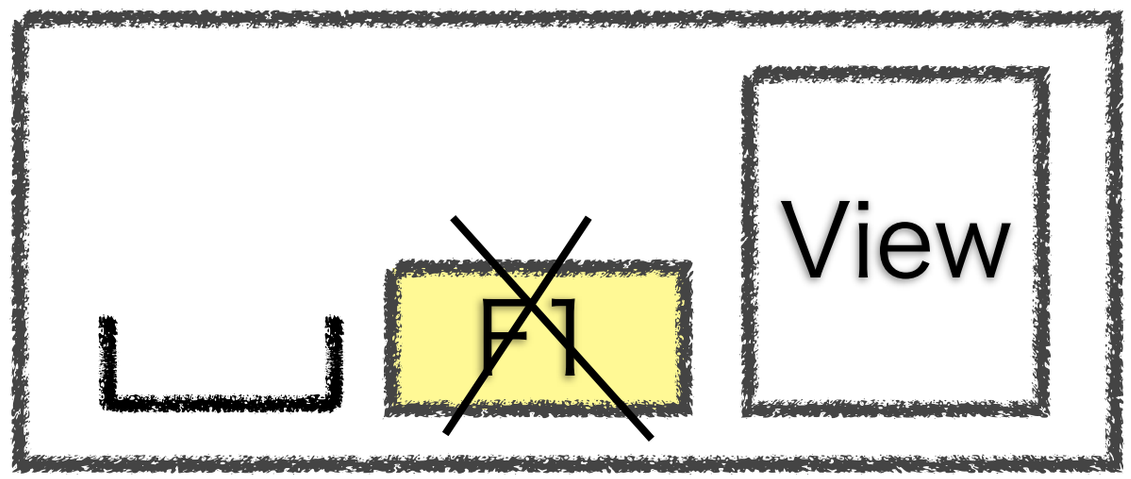
onDestroy()

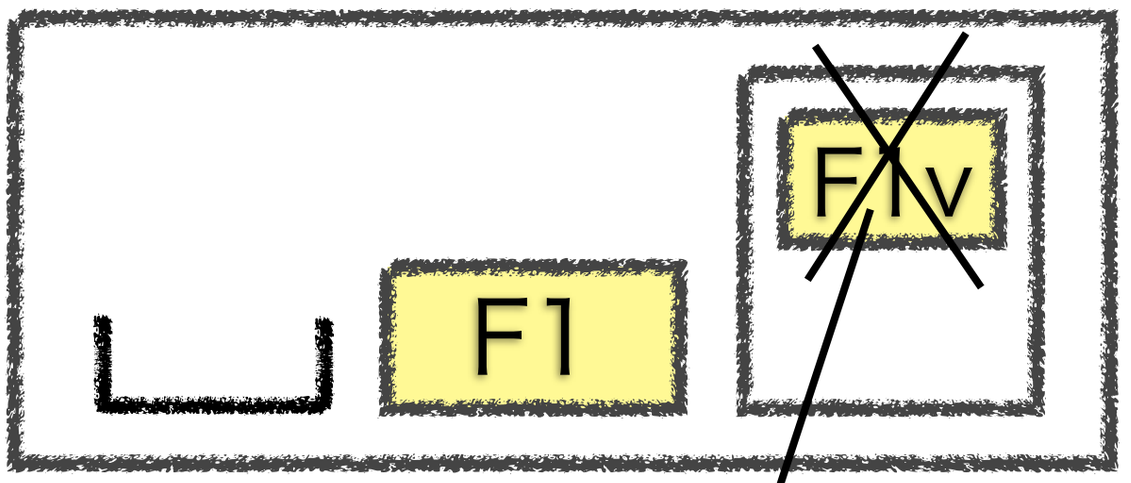
onDestroyView()

onStop()

onPause()





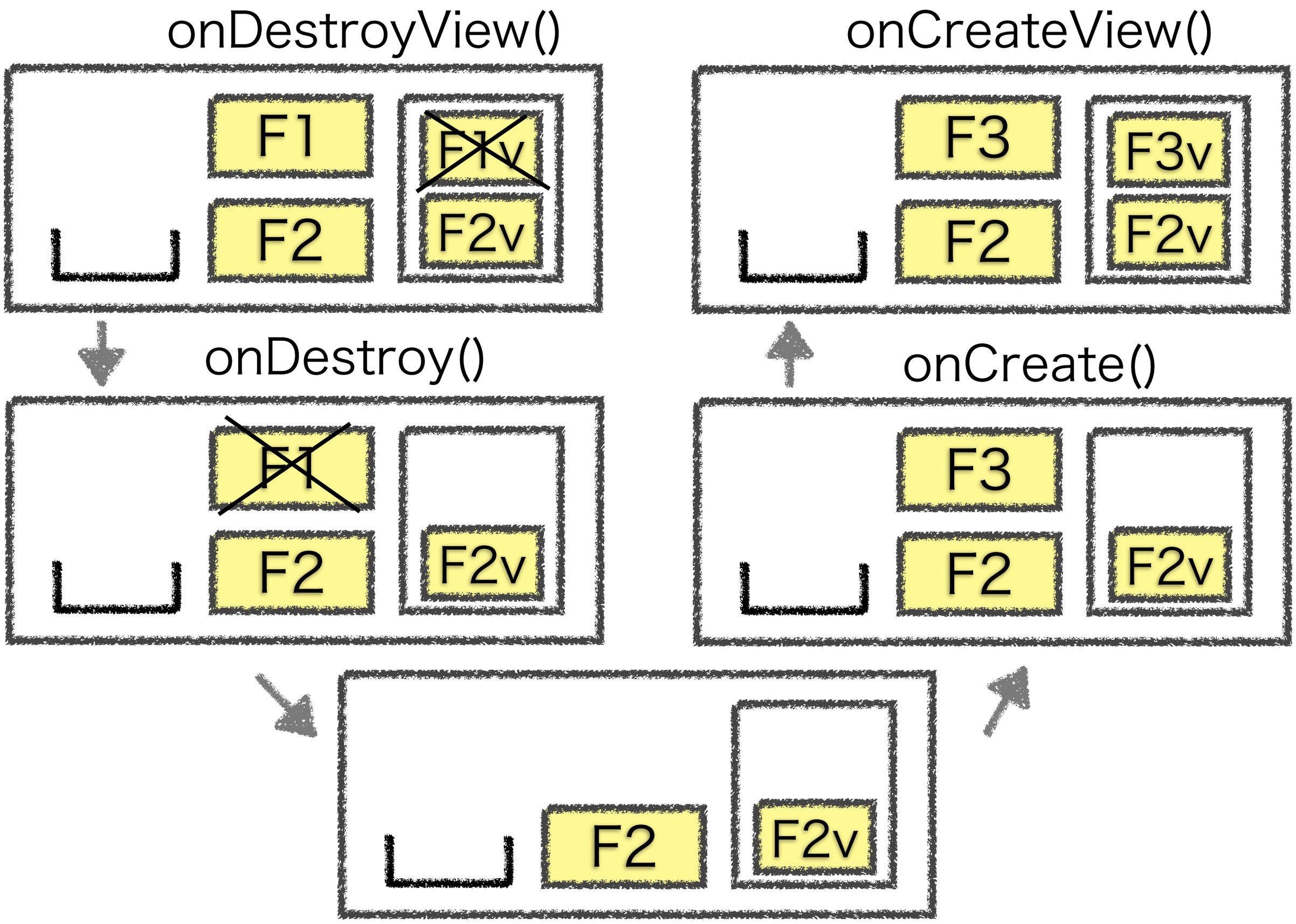


removeView()

Fragment の置き換え

replace

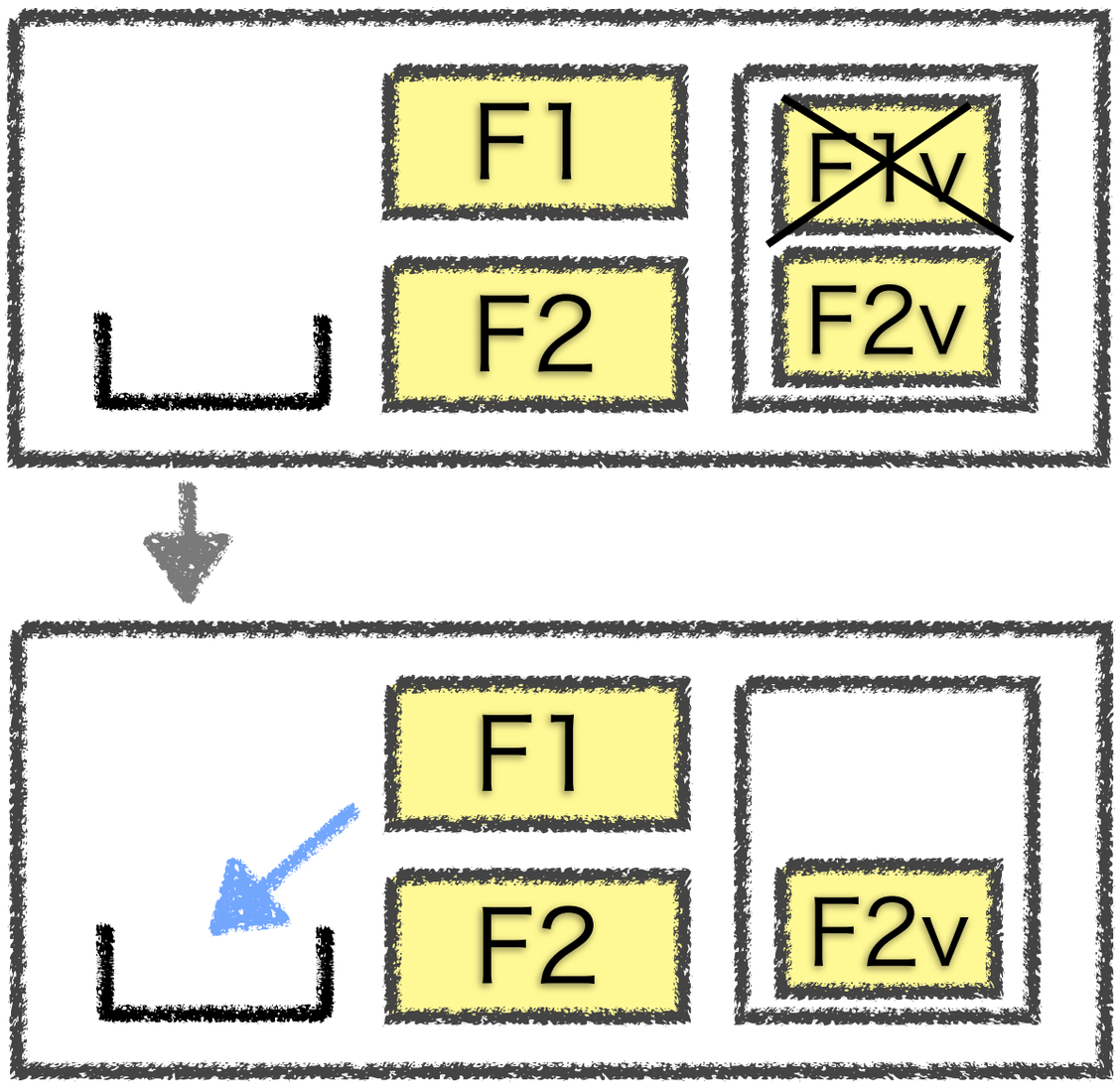
replace()

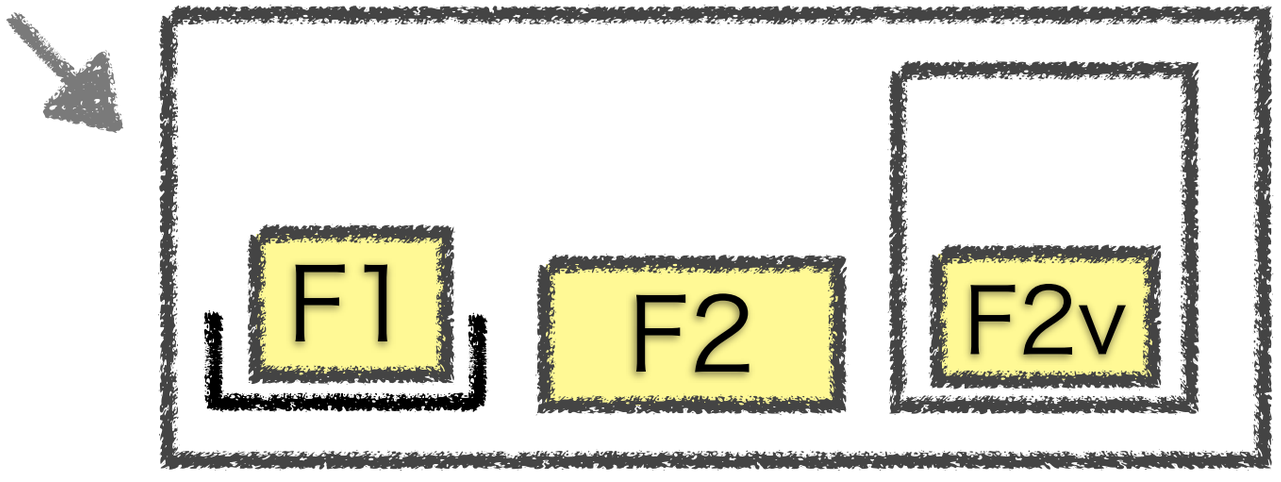

onDestroyView()
F1
F1v
F2
F2v
onDestroy()
onCreateView()
F3
F3v
F2
F2v
onCreate()
F1
F2
F2v
F3
F2
F2v
F2
F2v

Fragment の BackStack

remove 時に BackStack に追加

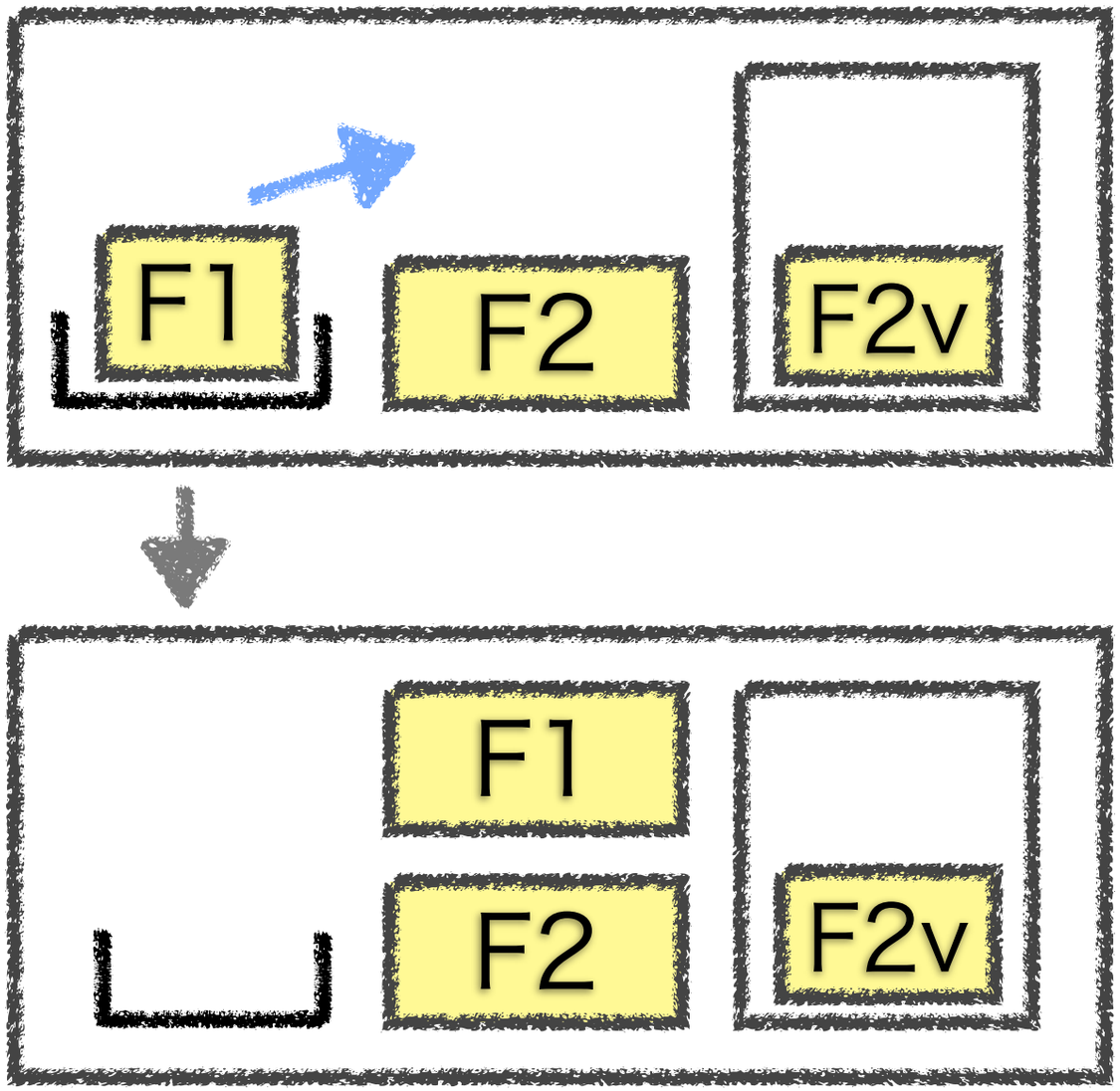
onDestroyView()





onDestroy()
onDetach()
は呼ばれない

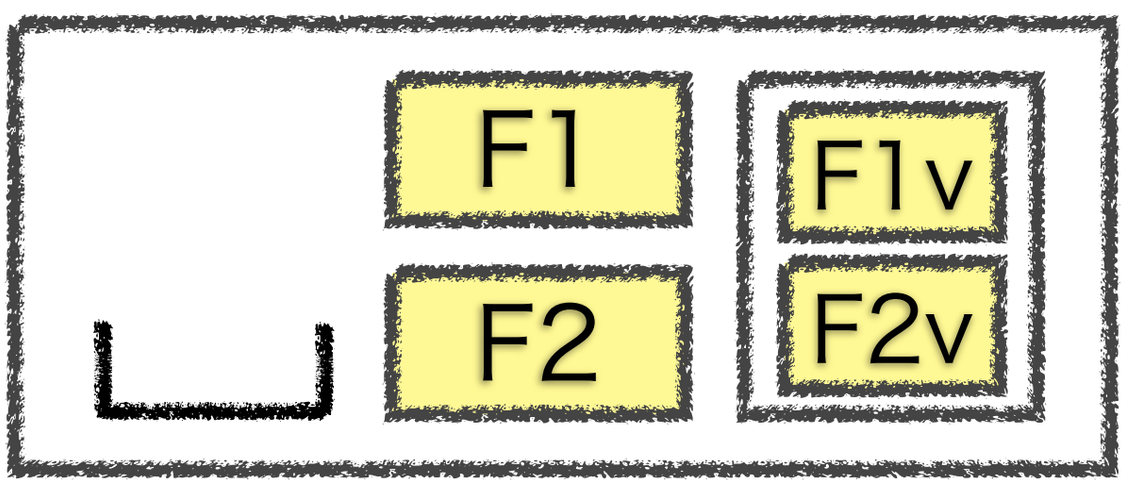
BackStack から pop



FragmentManager の
popBackStack() を呼ぶ
もしくはバックキー

onAttach(), onCreate()
は呼ばれない

onCreateView()

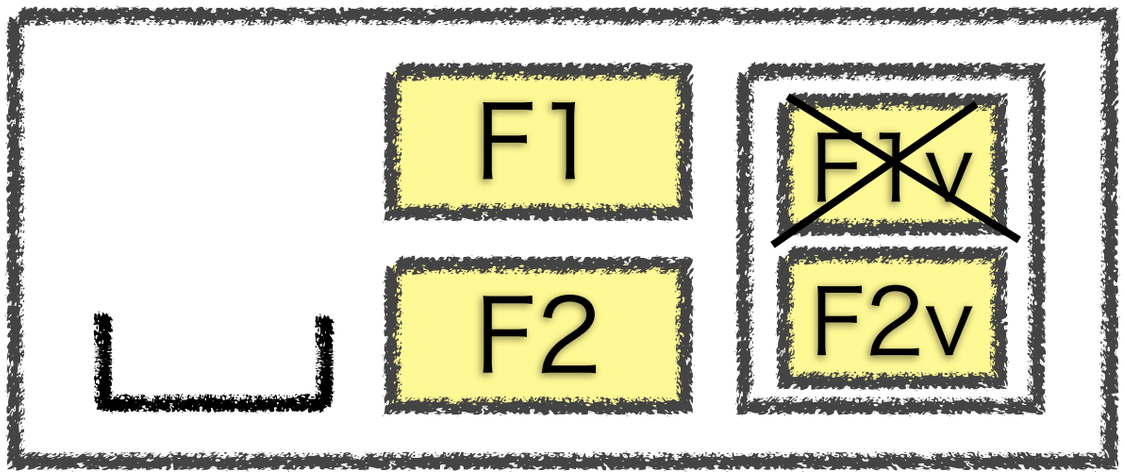


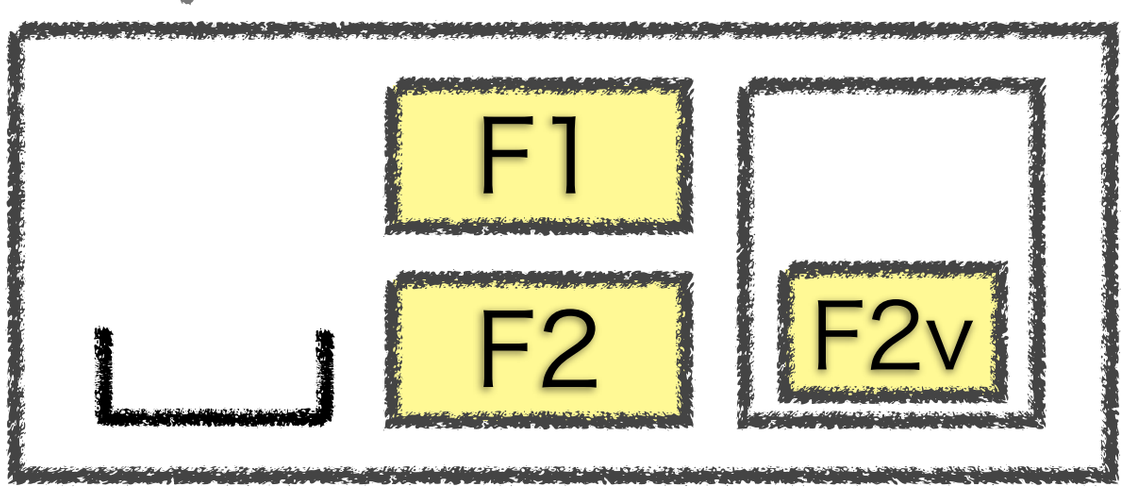
attach(), detach()

detach() 状態は BackStack に
入っているのと同じ状態

detach()

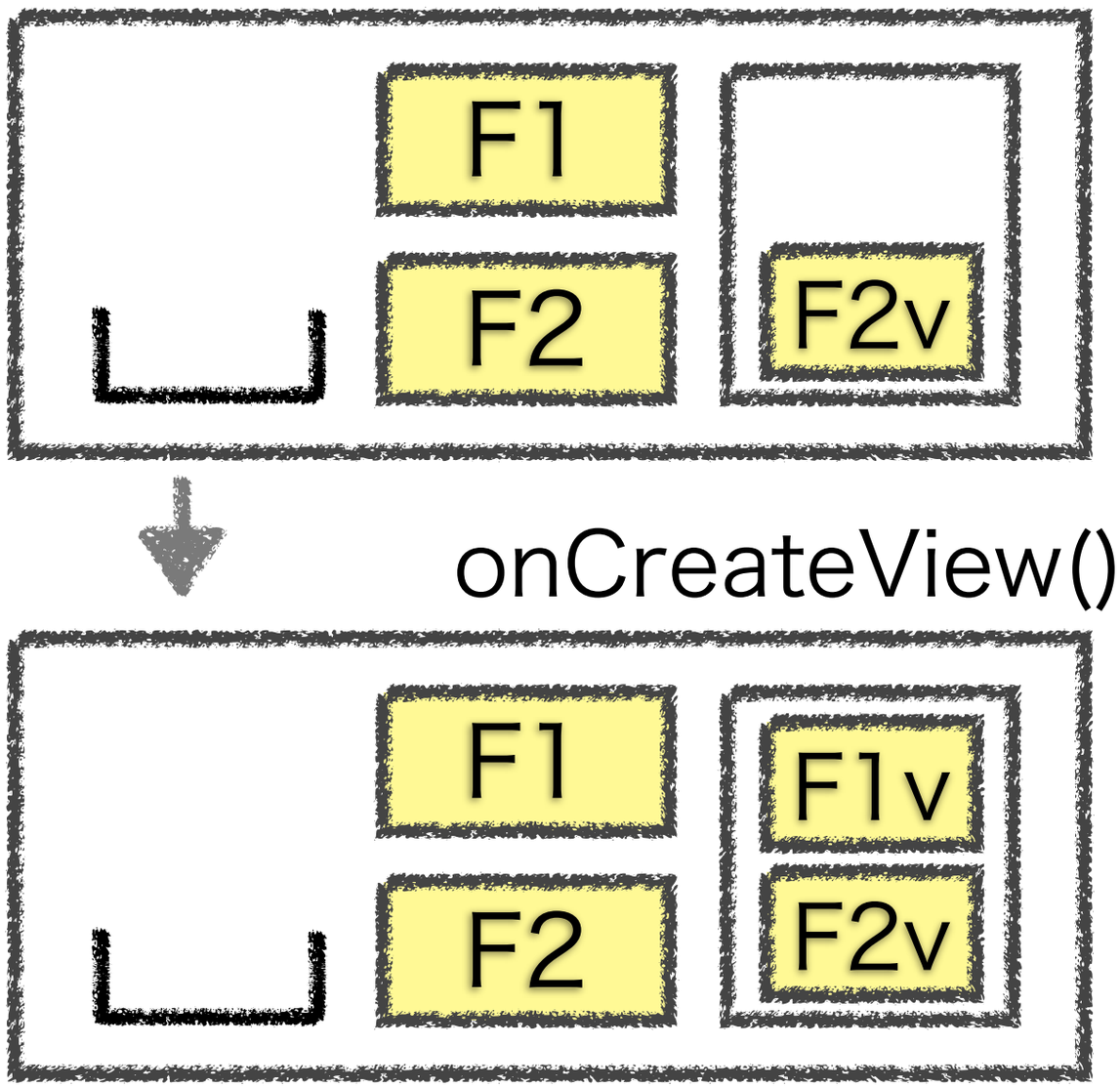
onDestroyView()





onDestroy()
onDetach()
は呼ばれない

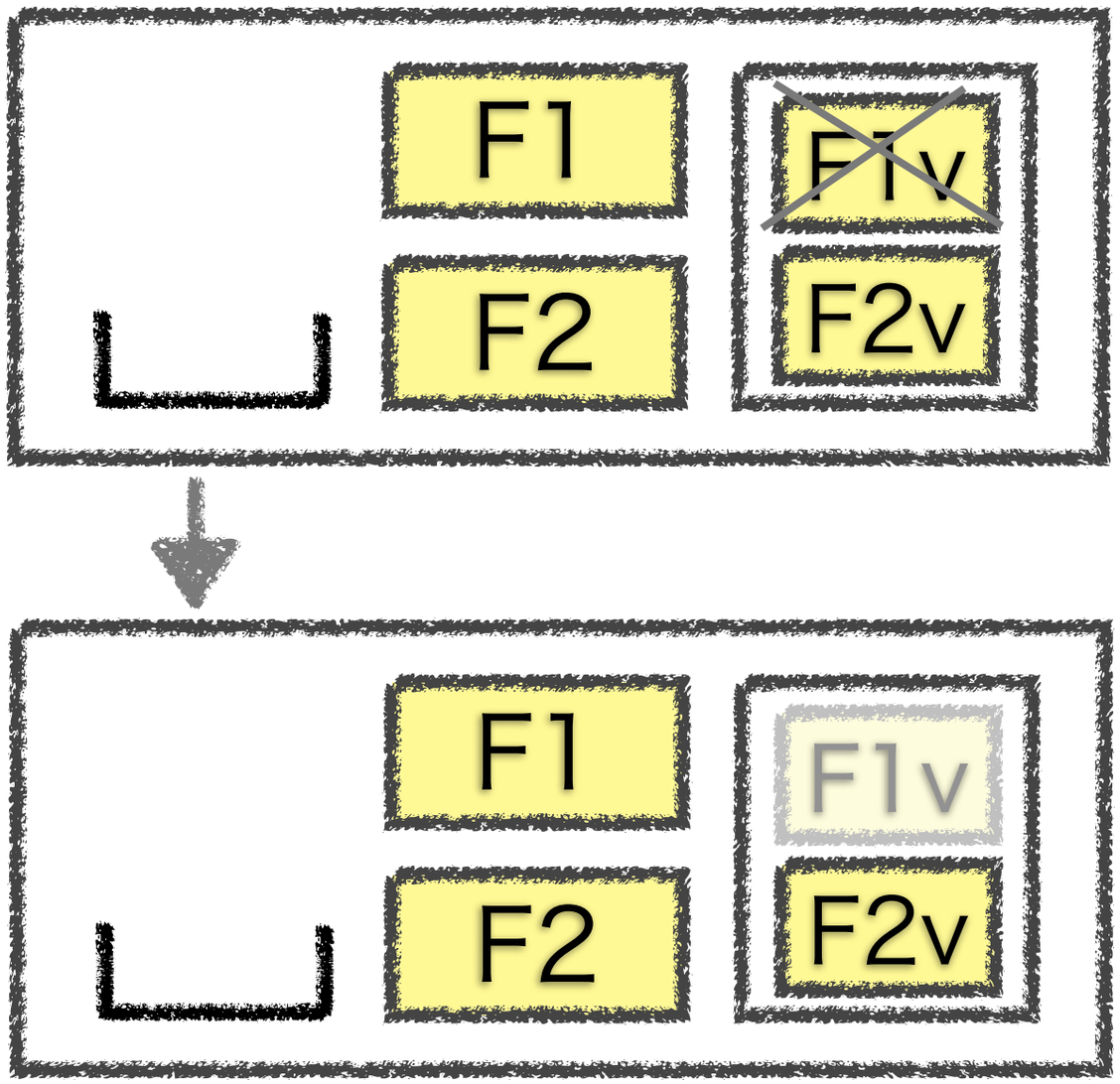
detach() した Fragment を attach()



onAttach()
onCreate()
は呼ばれない

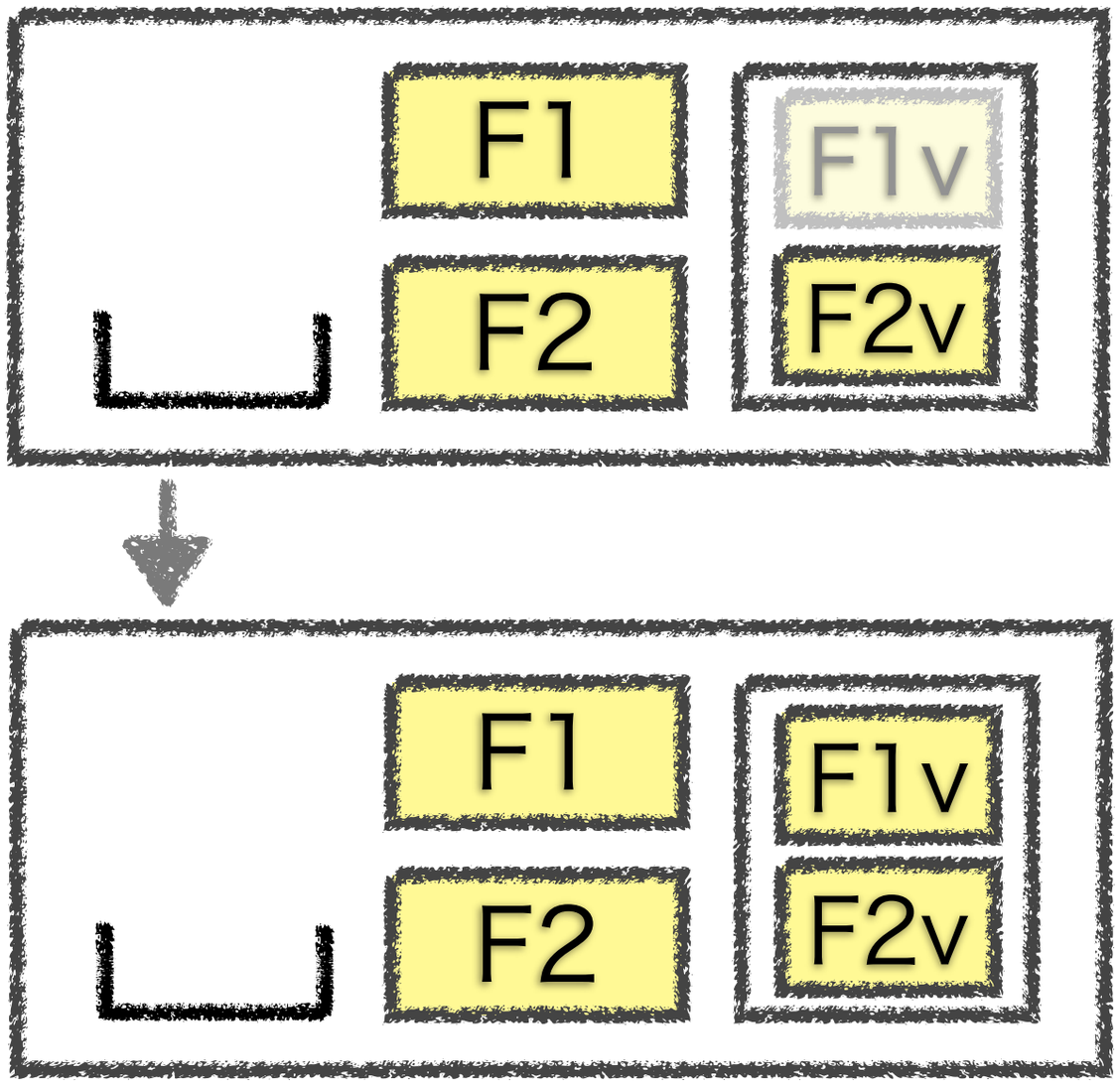
show(), hide()

hide()



View は破棄されないが
非表示になる
ライフサイクルは変化
しない

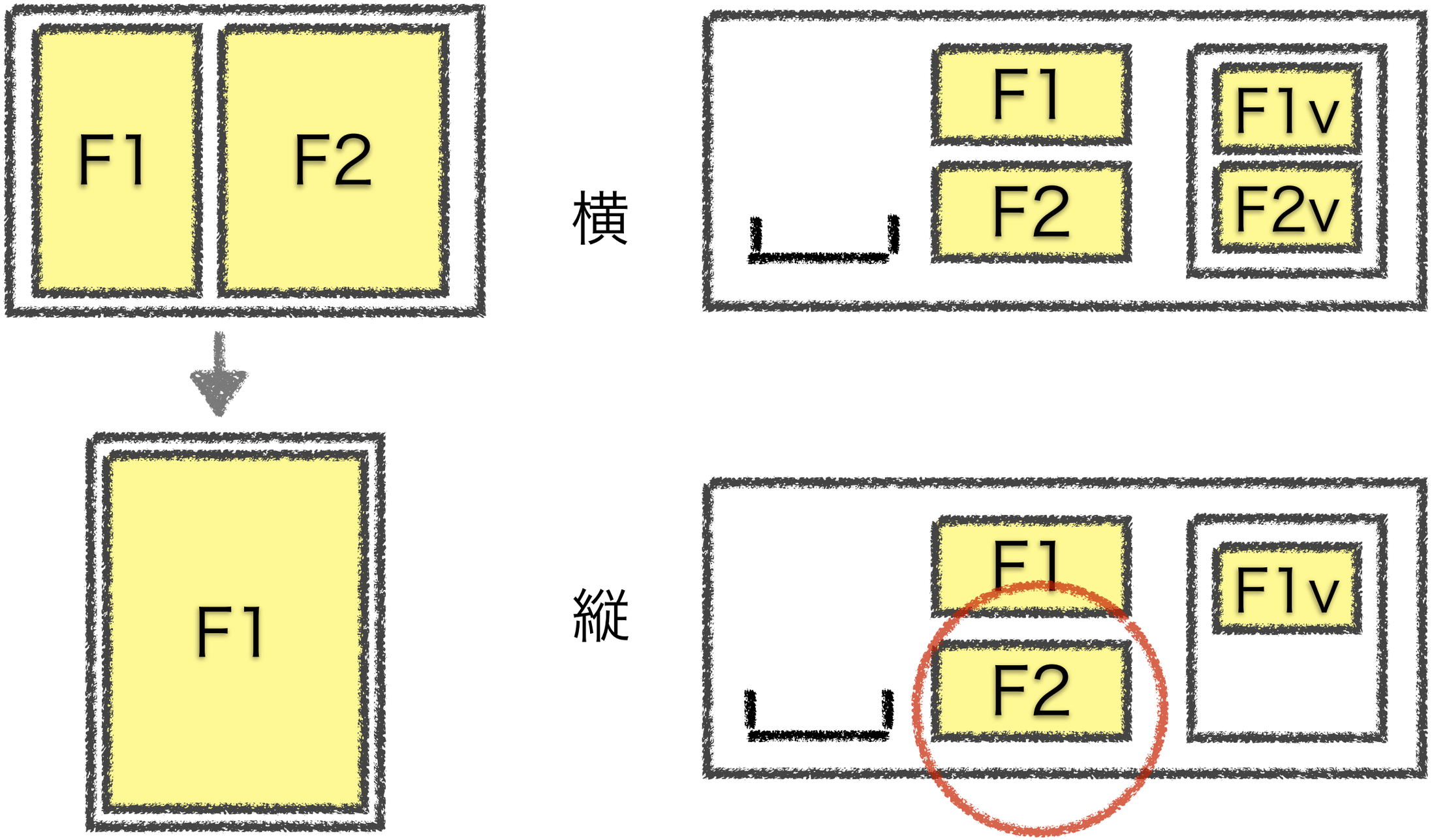
show()



hide() で非表示にした View が表示される
ライフサイクルは変化しない

Fragment トランザクション
での注意点

Activity に attach された Fragment は再生成される



Activity に attach された Fragment は再生成される

判定方法

- onCreateView() の第2引数の
  ViewGroup が変わる null かどうか
  null の場合 = 返した View は使われない

- isInLayout() メソッド

BackStack を使うとバックキーで前の画面に戻る UX が実現出来る

Activity による画面遷移の置き換え

Fragment による画面遷移の使いどころ1

- Activity で画面遷移できないところで使う
  - 例）launch mode が singleInstance の Activity は同じタスク内で Activity による画面遷移ができない
  - 例）Activity のスタックに残したくない

- Fragment を使って画面遷移が可能

Fragment による画面遷移の使いどころ2

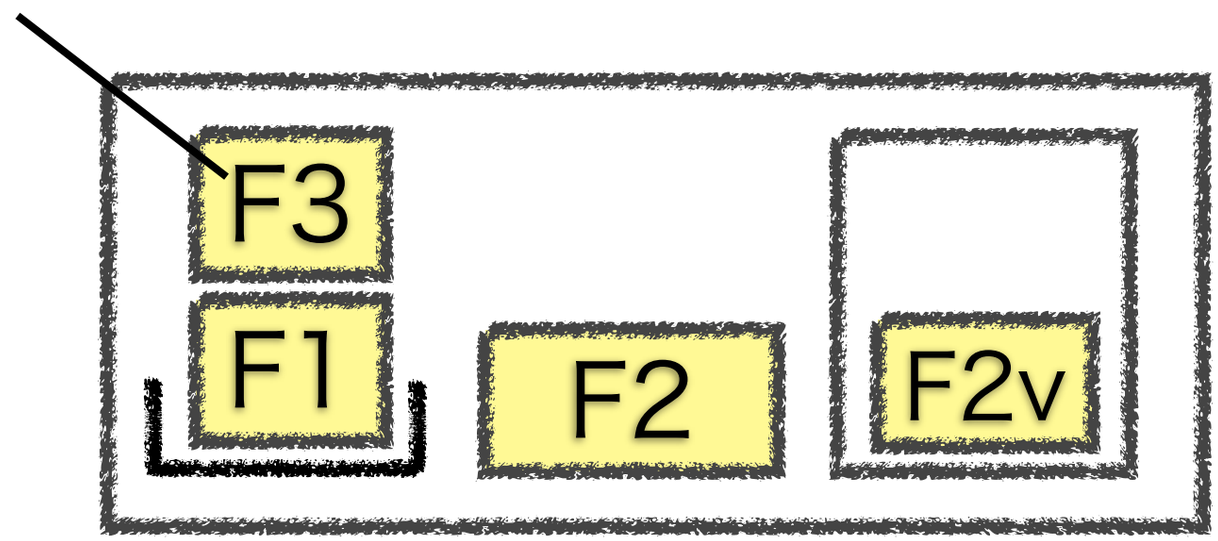
- 画面の一部分だけ切り替えたい
    - 例）Gmail の ドロップダウン
- その画面で行う処理が少ない
    - 例）２階層のリストから選択する　リスト → リスト → 画面
- パン屑リストを使いたい
    - FragmentBreadCrumbs

BackStack の管理は
FragmentManager で行う

BackStack の中身

FragmentManager.BackStackEntry

- ID
- Name
- パン屑リストのタイトル



FragmentManager

- getBackStackEntryCount()
- getBackStackEntryAt()

FragmentManager

- BackStack から Fragment のトランザクションを戻す
    - popBackStack()
    - popBackStackImmediate()
- BackStack の変更を検知
    - addOnBackStackChangedListener()
    - removeOnBackStackChangedListener()

FragmentManager

- Bundle に Fragment をリファレンスを格納できる = 状態として永続化出来る
    - putFragment(Bundle bundle, String key, Fragment fragment)
        - bundle に key で指定されたキーで fragment のリファレンスを格納
- getFragment(Bundle bundle, String key)
    - bundle から key で指定されたキーで fragment を取り出す

まとめ

- Fragment を有効に利用するには、ライフサイクルとトランザクションの理解が不可欠
- Activity は一連の完結した処理を単位にする
    - 外部から（暗黙的、明示的いずれでも）Intent として呼ばれる画面は Activity にする
    - Activity スタックとして残したくない画面遷移は Fragment にする
    - launch mode で考えるのも有効
- Fragment は、データの取得、処理、表示（あれば）で一単位にする