Flux de Relax :)

ANDROID ALLSTARS #2 @dots.
Masaki Ogata

About me

Masaki Ogata

CyberAgent, Inc. / AbemaTV, Inc.

ogaclejapan

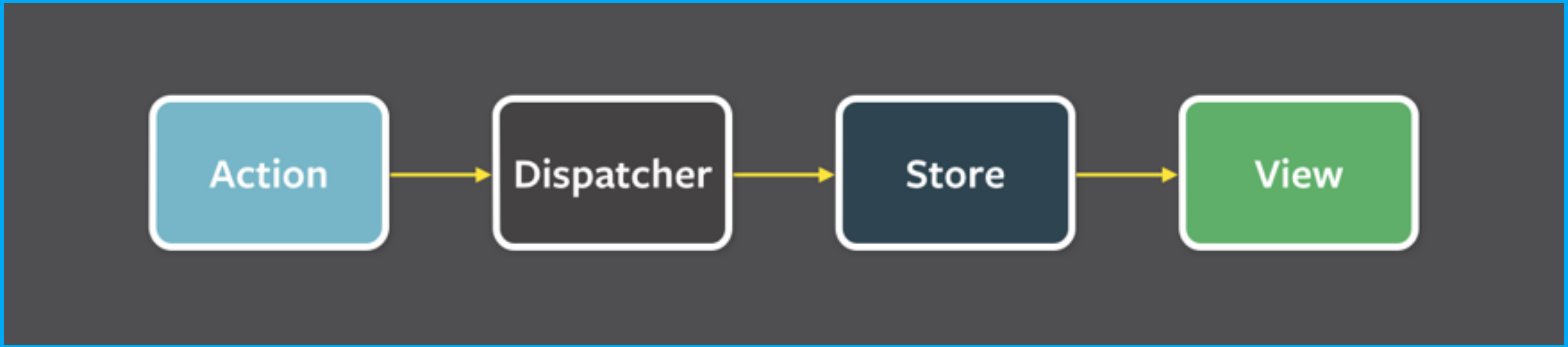
@ogaclejapan

Flux de Relax :)

What is Flux?

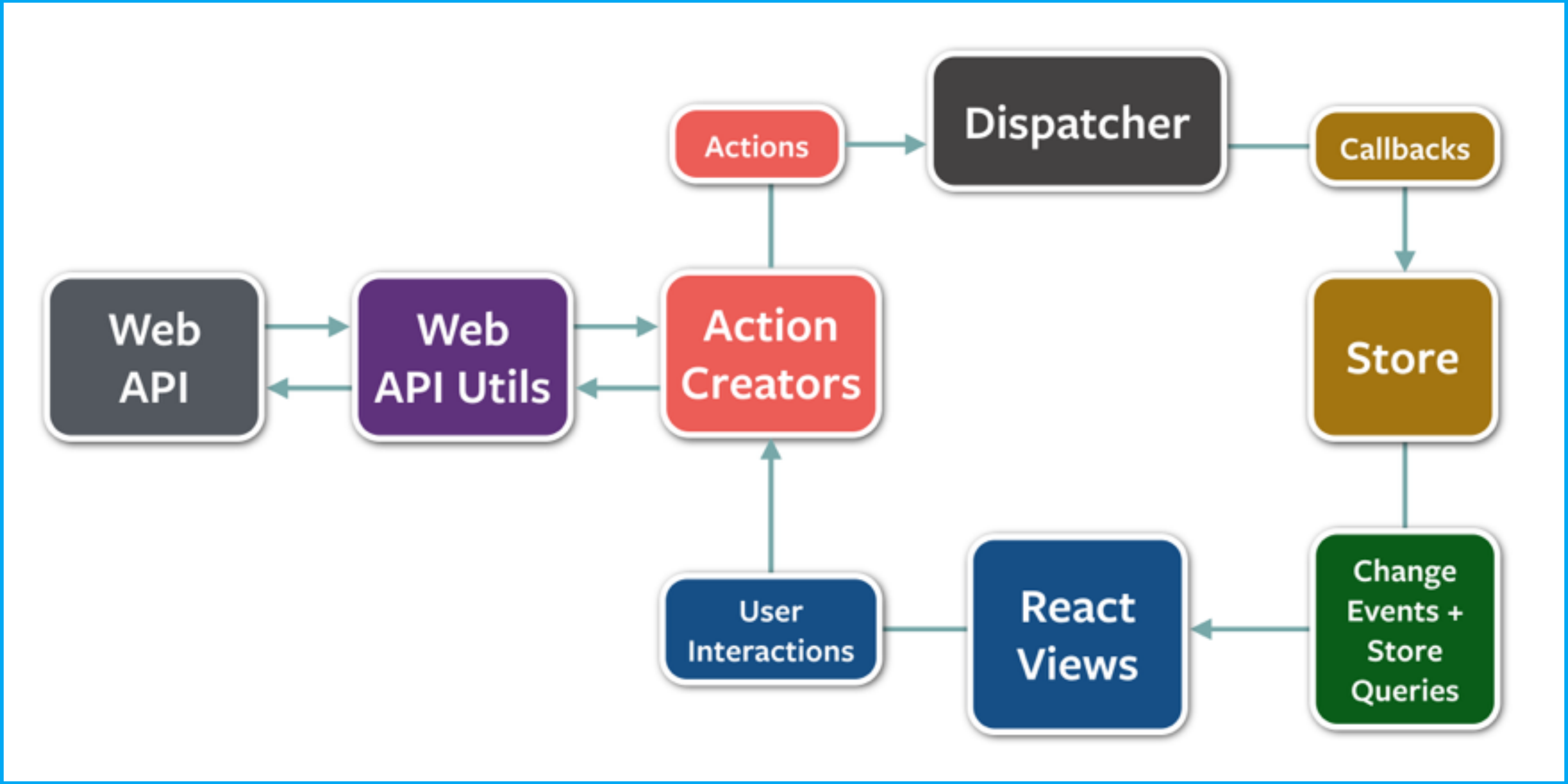
Facebook Flux Architecture

“Data in a Flux application flows in a single direction”



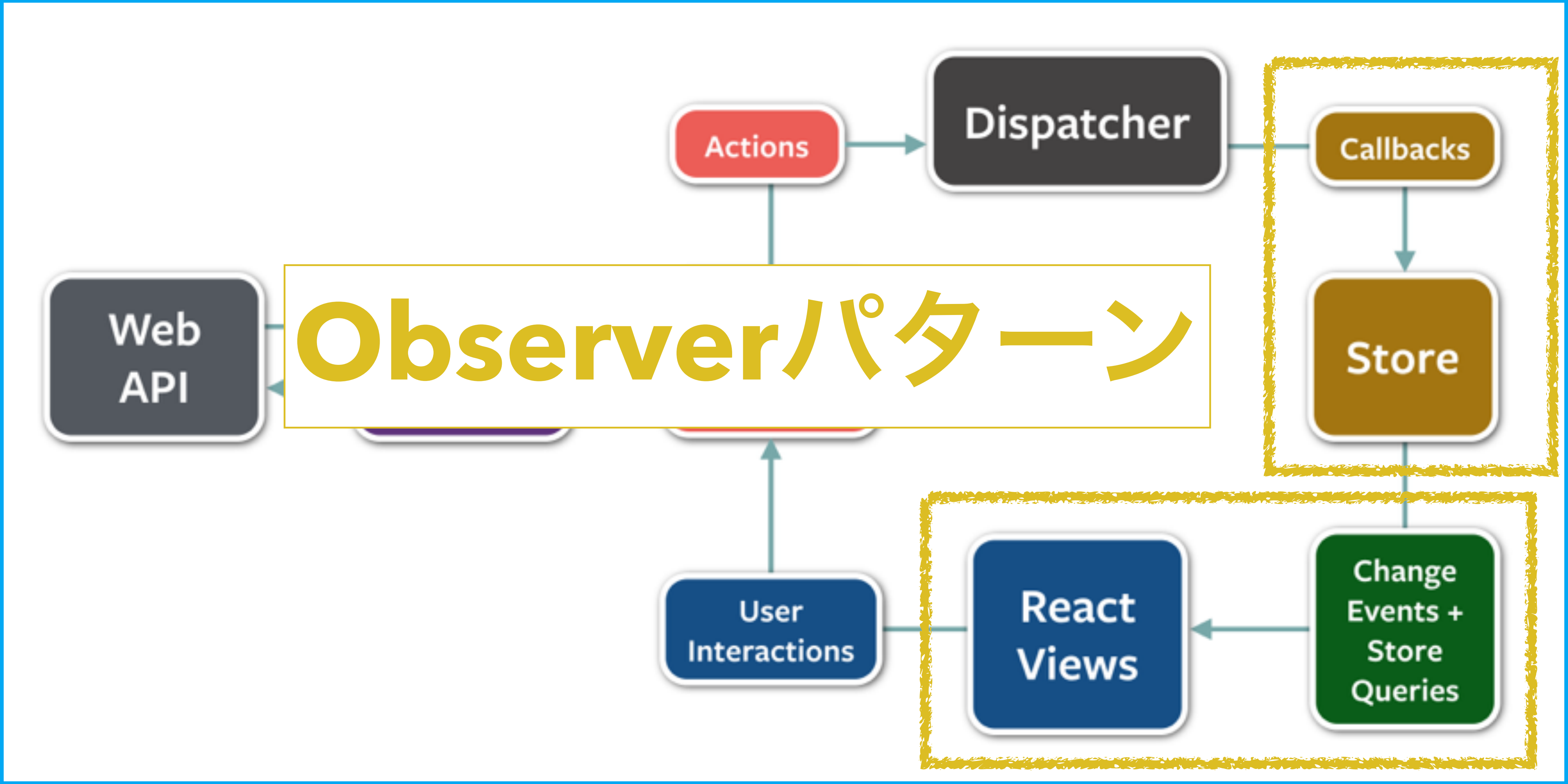
https://facebook.github.io/flux/docs/overview.html

Facebook Flux Architecture



https://github.com/facebook/flux

Facebook Flux Architecture



https://github.com/facebook/flux

Why Flux?

Why Flux?

アプリケーションの開発で
Viewの状態管理が一番難しい :(

Why Flux?

AbemaTVで必要になるViewの状態管理:

Cast

課金

CM

フィラー

画質

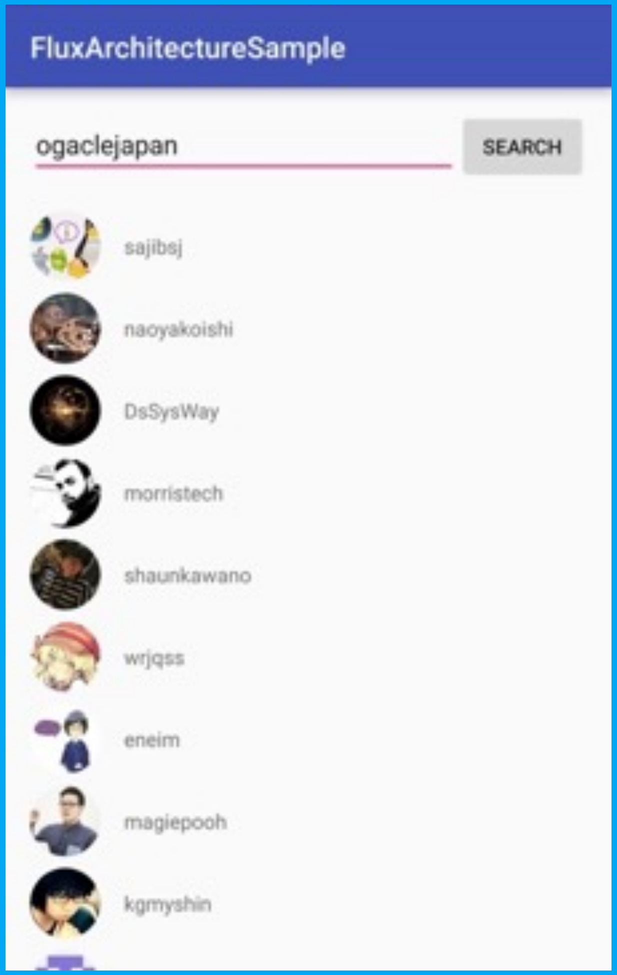
オンデマンド

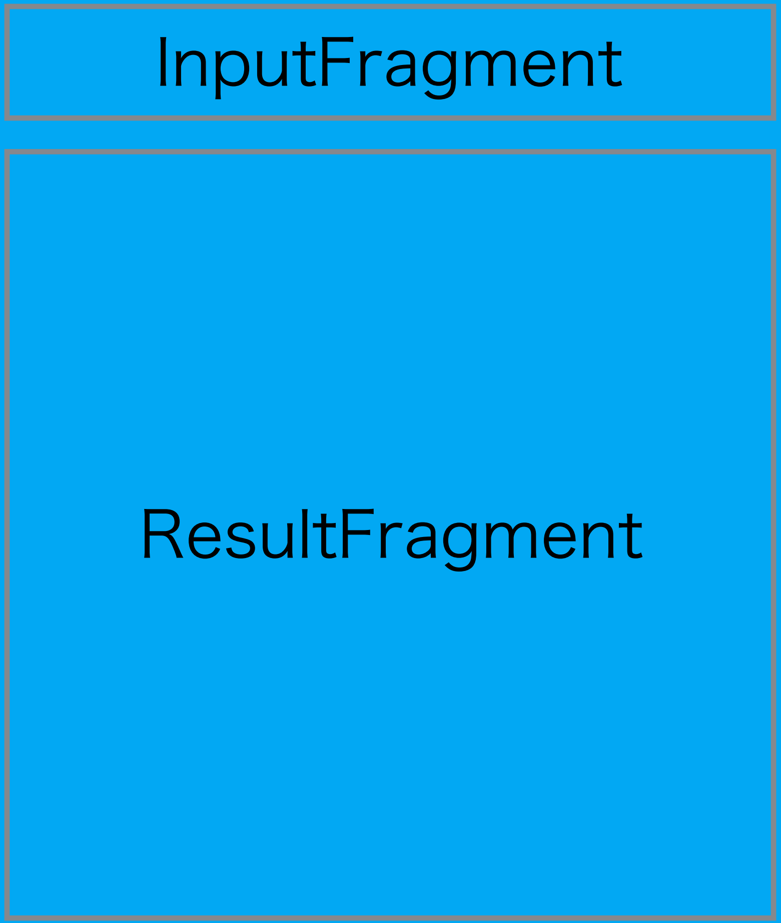
視聴予約

etc…

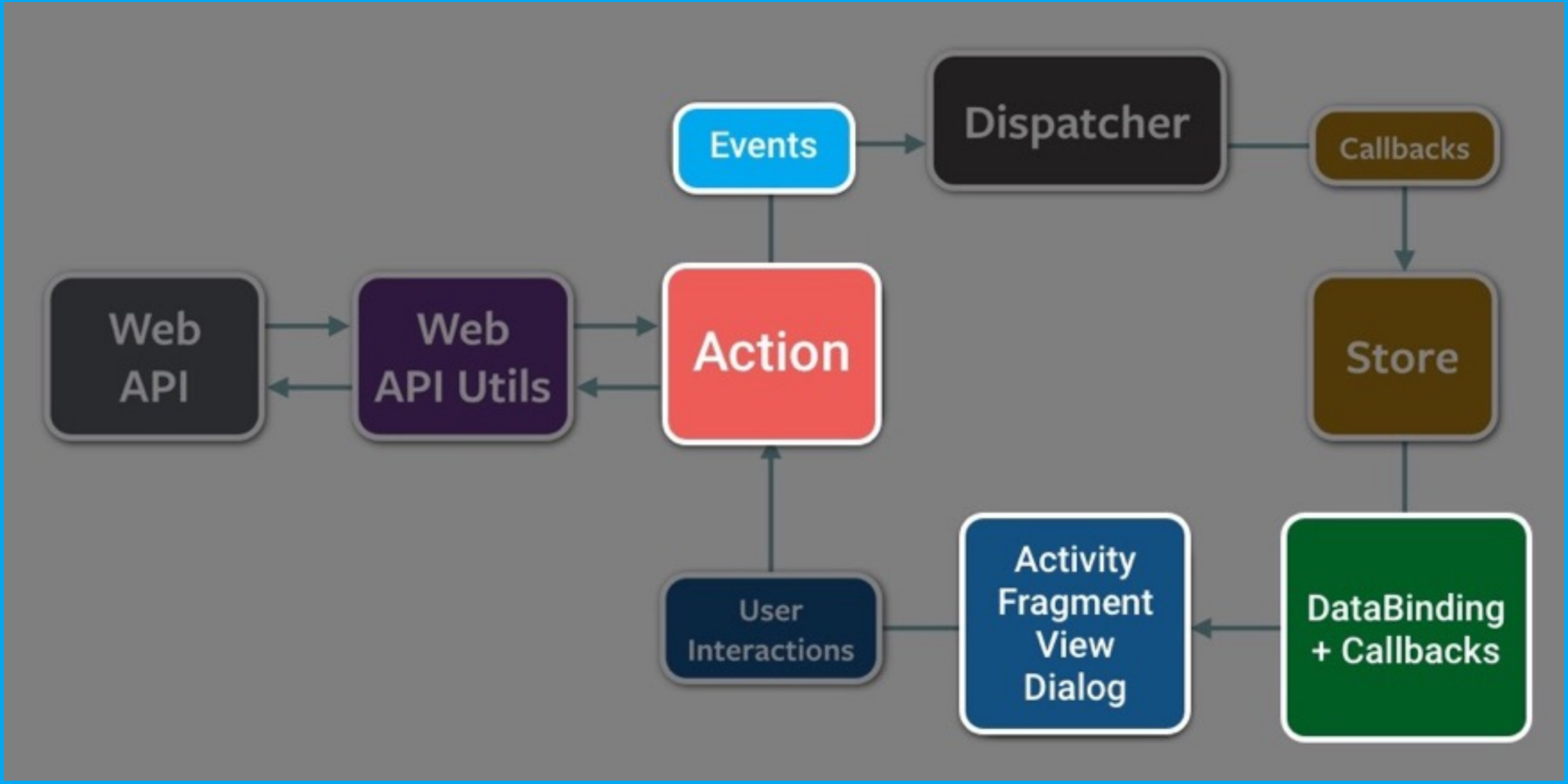
Sample code

https://github.com/ogaclejapan/FluxArchitectureSample

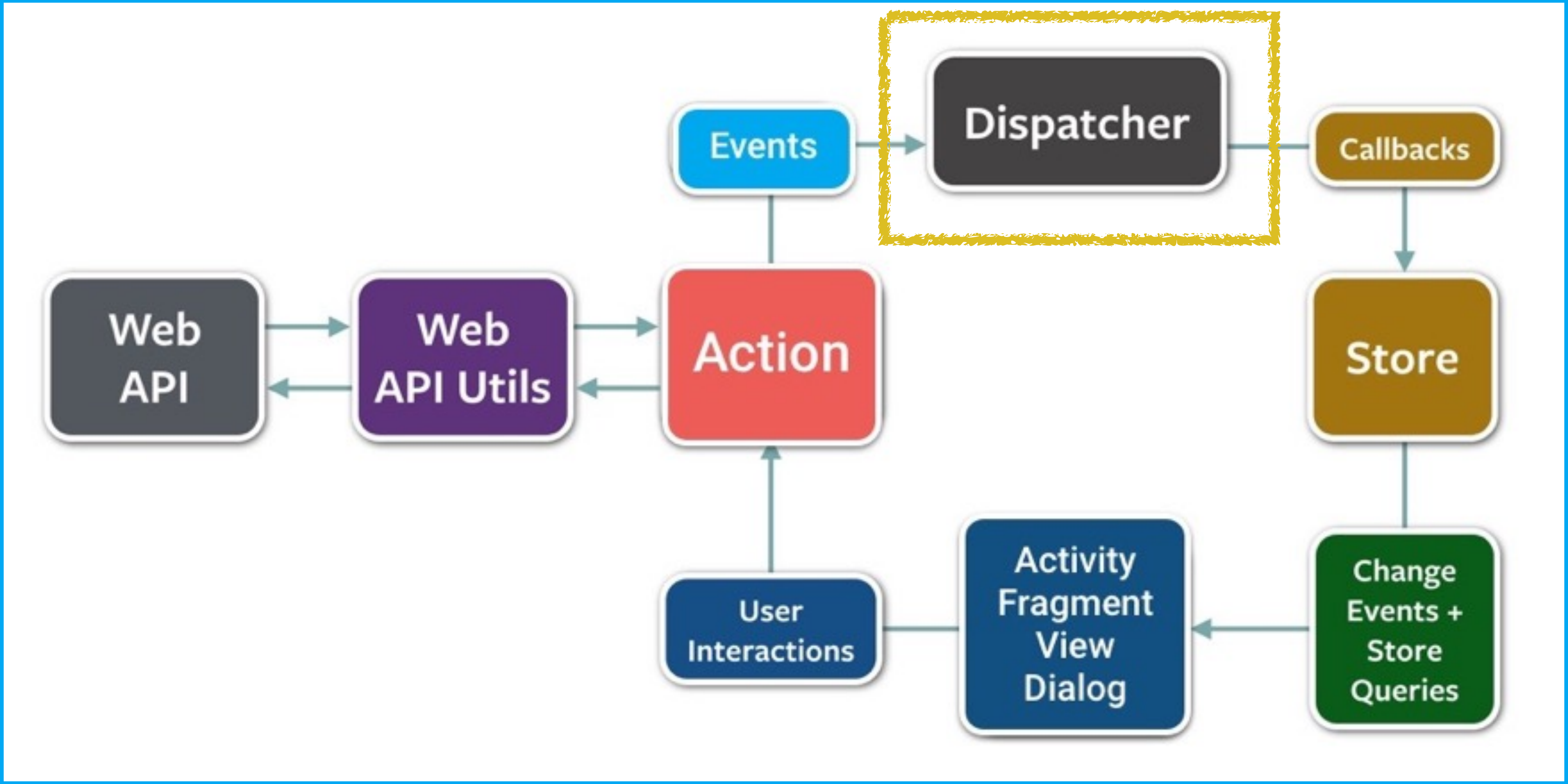




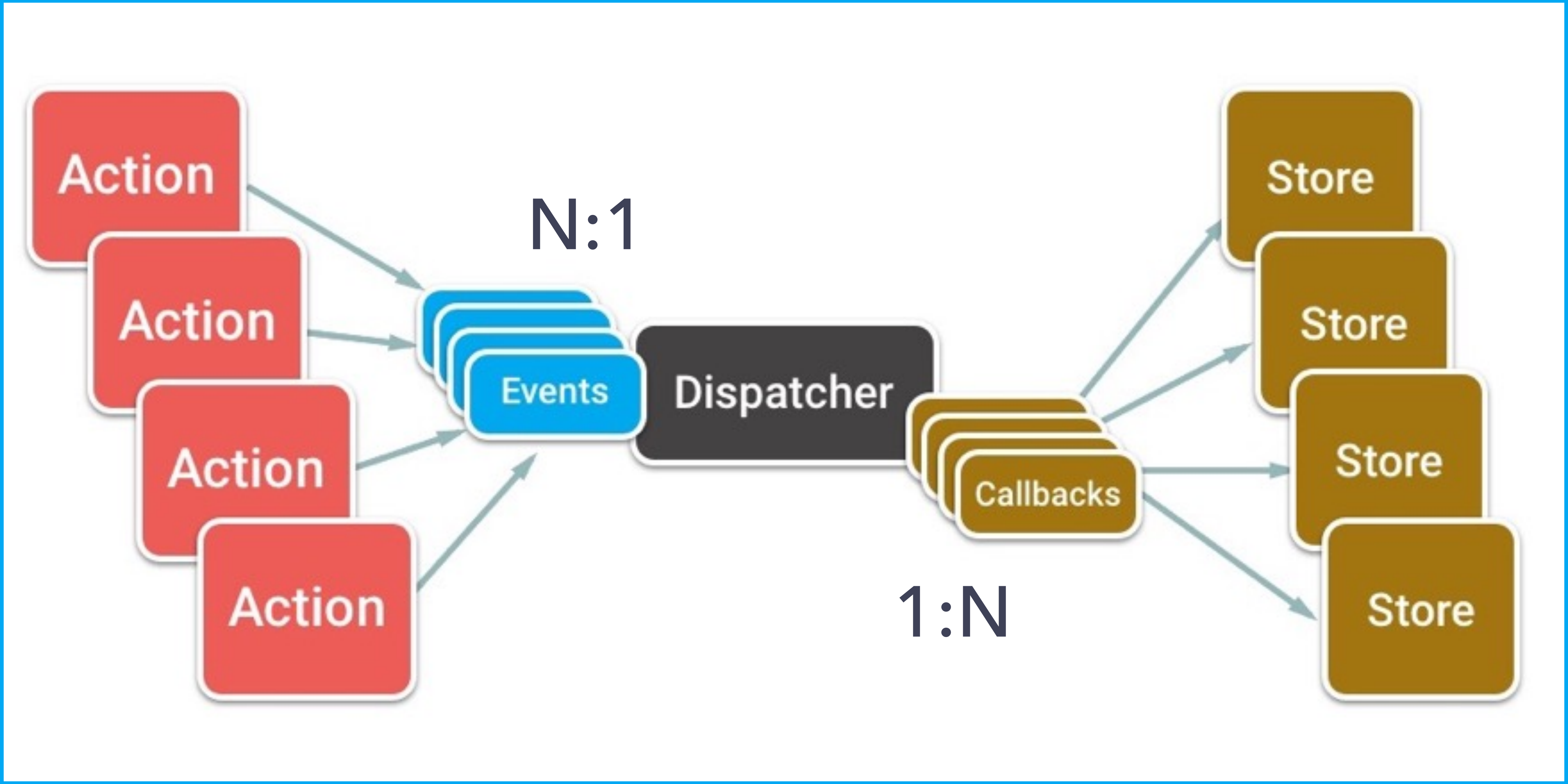
Flux Architecture Sample



Flux: Dispatcher



Flux: Dispatcher



Flux: Dispatcher

本家Fluxの実装を参考にしてみる

```
/* Action */
Dispatcher.dispatch({
  actionType: 'foo',
  payload: 'value'
});


/* Store */
Dispatcher.register(function(payload) {
    switch(payload.actionType) {
      case 'foo':
        ... = payload.data
        // Do something
    }
}
```

Flux: Dispatcher

Javaで実装を書き直してみると…

```
public interface Action { String getType(); }
public class FooAction implements Action {...}

/* Action */
Dispatcher.dispatch(new FooAction(data));

/* Store */
Dispatcher.register(new Callback() {
  public void on(Action action) {
    switch (action.getType()) {
      case "foo":
        ... = ((FooAction) action).data;
        break;
    }
  }
}
```

Flux: Dispatcher

Javaで実装を書き直してみると…

```
public interface Action { String getType(); }
public class FooAction implements Action {...}

/* Action */
Dispatcher.dispatch(new FooAction(data));

/* Store */
Dispatcher.register(new Callback() {
  public void on(Action action) {
    switch (action.getType()) {
      case "foo":
        ... = ((FooAction) action).data;
        break;
    }
  }
}
```

Javaだと型変換が必要 :(

Flux: Dispatcher

…EventBusでよくねぇ？

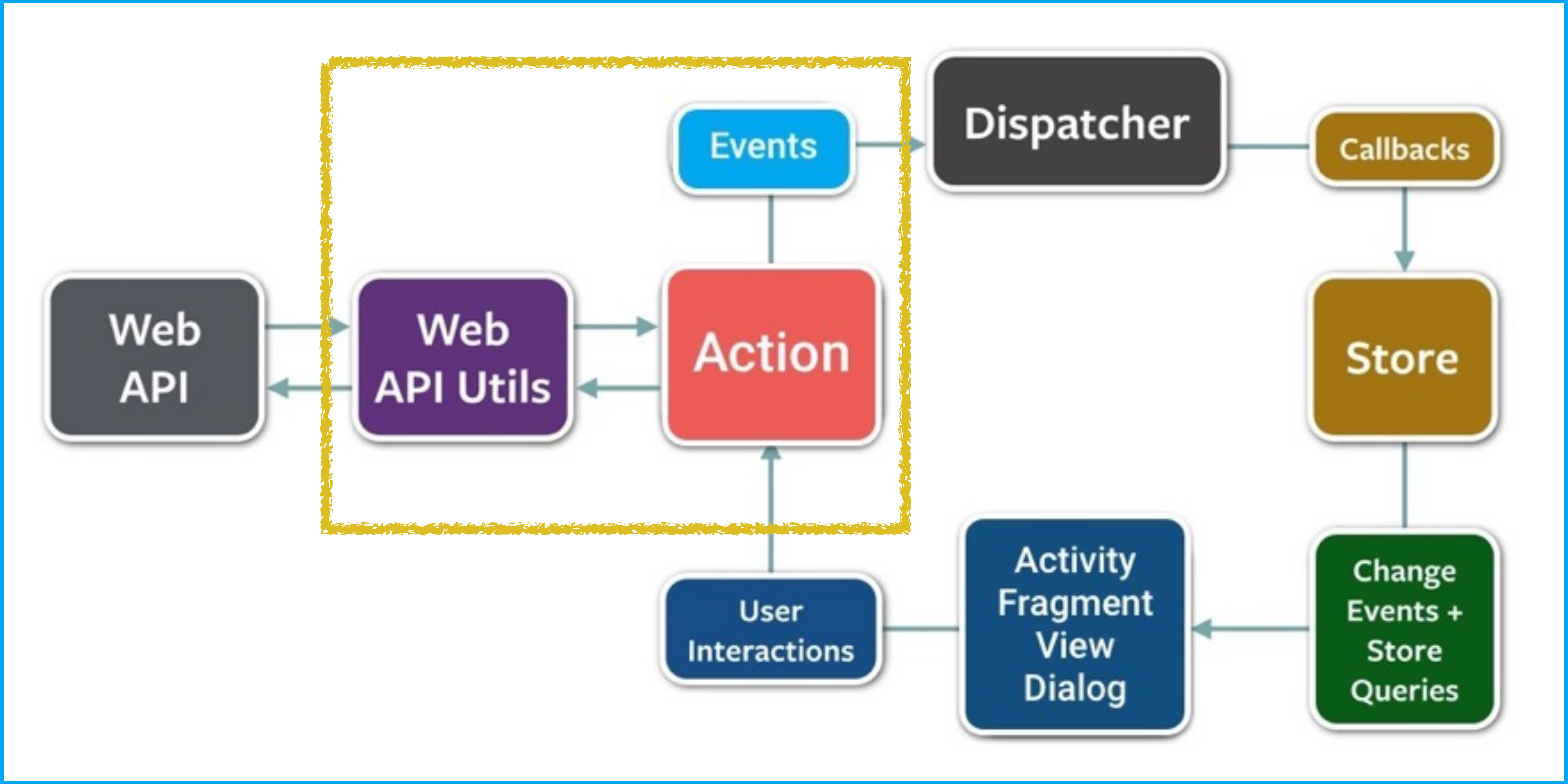
https://github.com/greenrobot/EventBus

```
// Define events:
public class MessageEvent { /* Additional fields if needed */ }

// Prepare subscribers: Register your subscriber
eventBus.register(this);

// Declare your subscribing method:
@Subscribe
public void onEvent(AnyEventType event) {/* Do something */};

// Post events:
eventBus.post(event);
```

Flux: Dispatcher

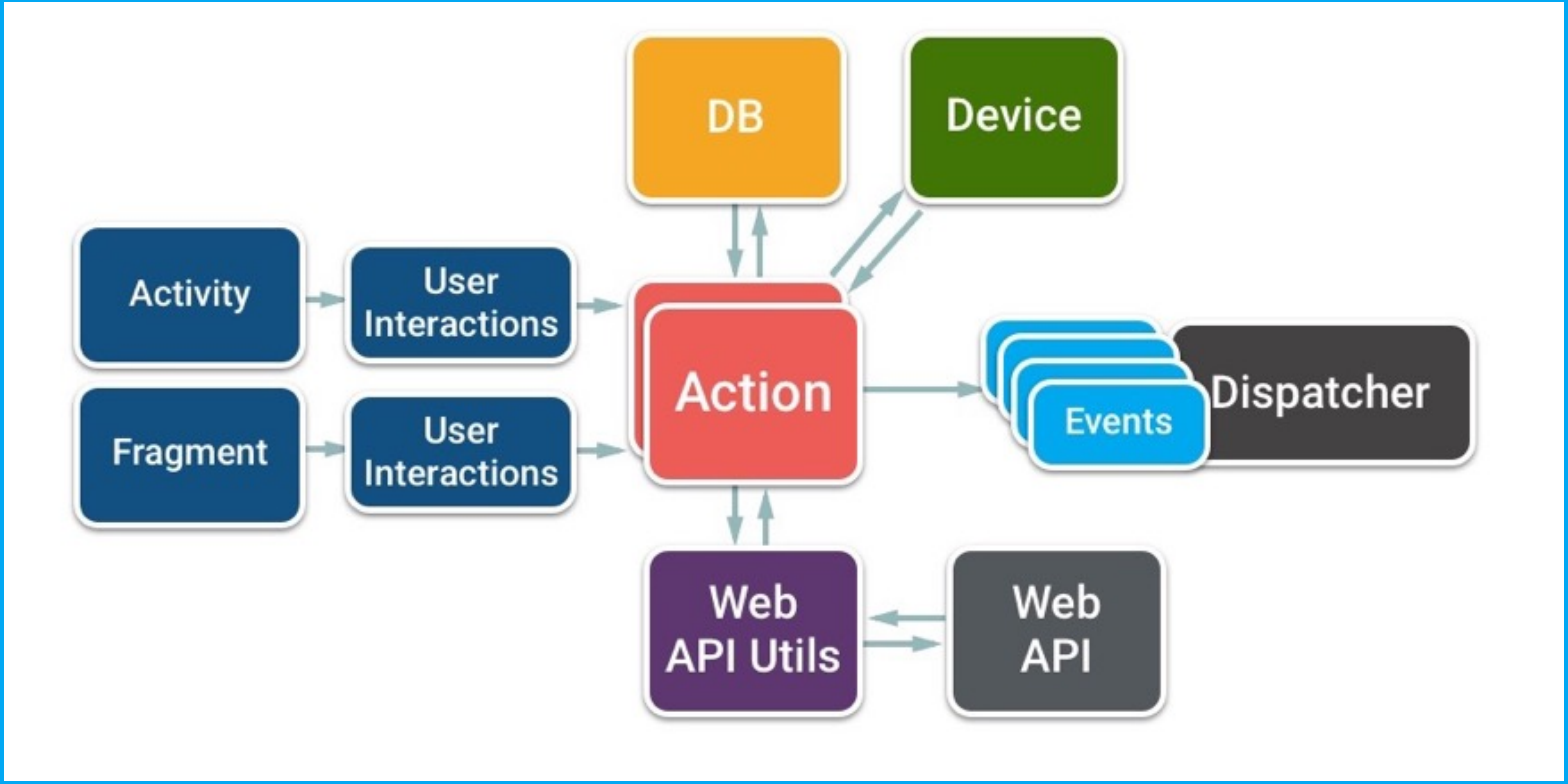
```java
public class Dispatcher {

  private final EventBus bus;

  public Dispatcher() {
    bus = EventBus.builder()
        ...
        .build();
  }

  public void dispatch(Object payload) {
    bus.post(payload);
  }

  public void register(Object observer) {
    bus.register(observer);
  }

  public void unregister(Object observer) {
    bus.unregister(observer);
  }
}
```

Flux: Action


Events
Dispatcher
Callbacks
Web API
Web API Utils
Action
Store
User Interactions
Activity Fragment View Dialog
Change Events + Store Queries

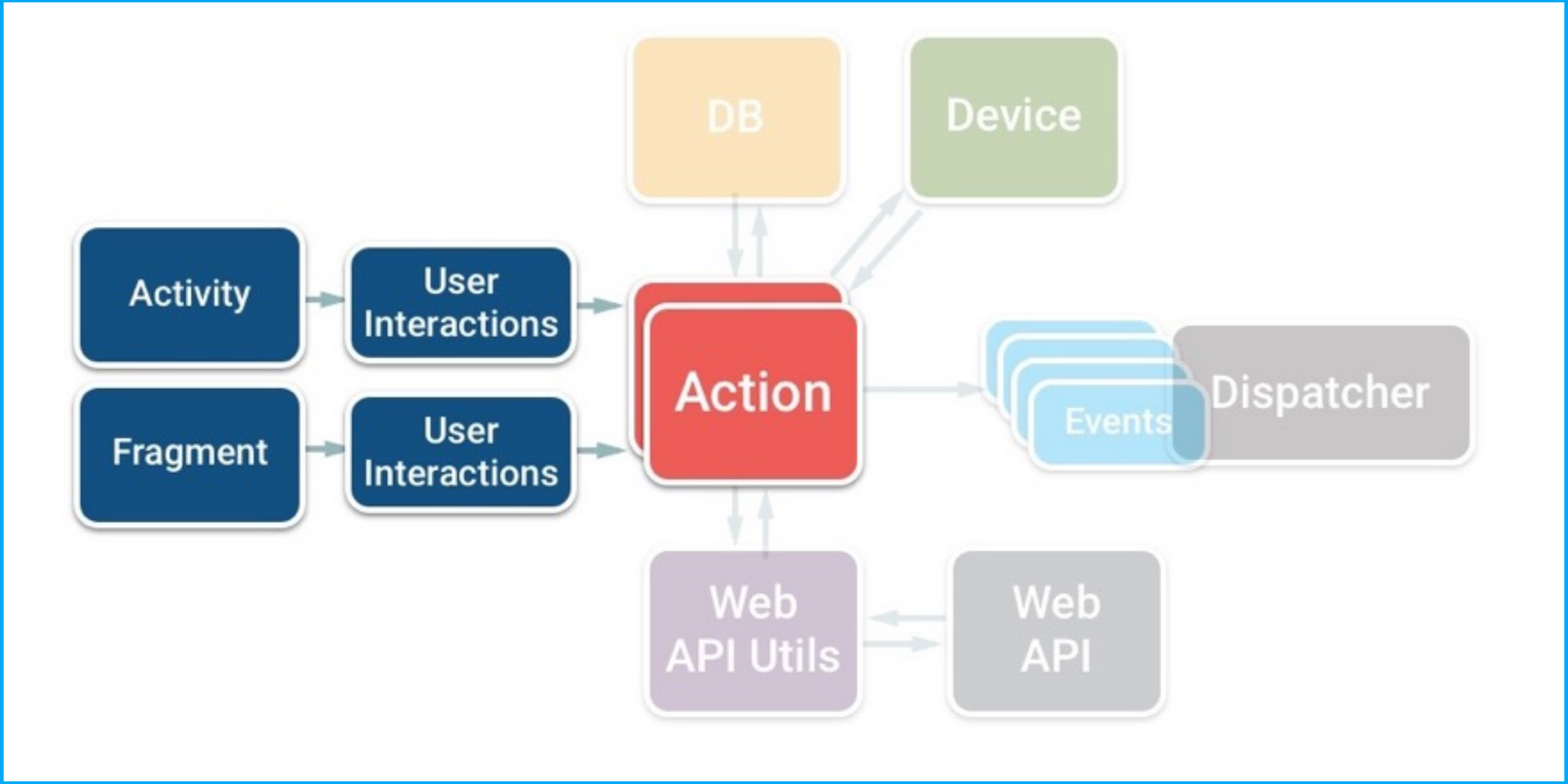
Flux: Action

Actionのデータフロー ٩(•ᴗ• ٩)



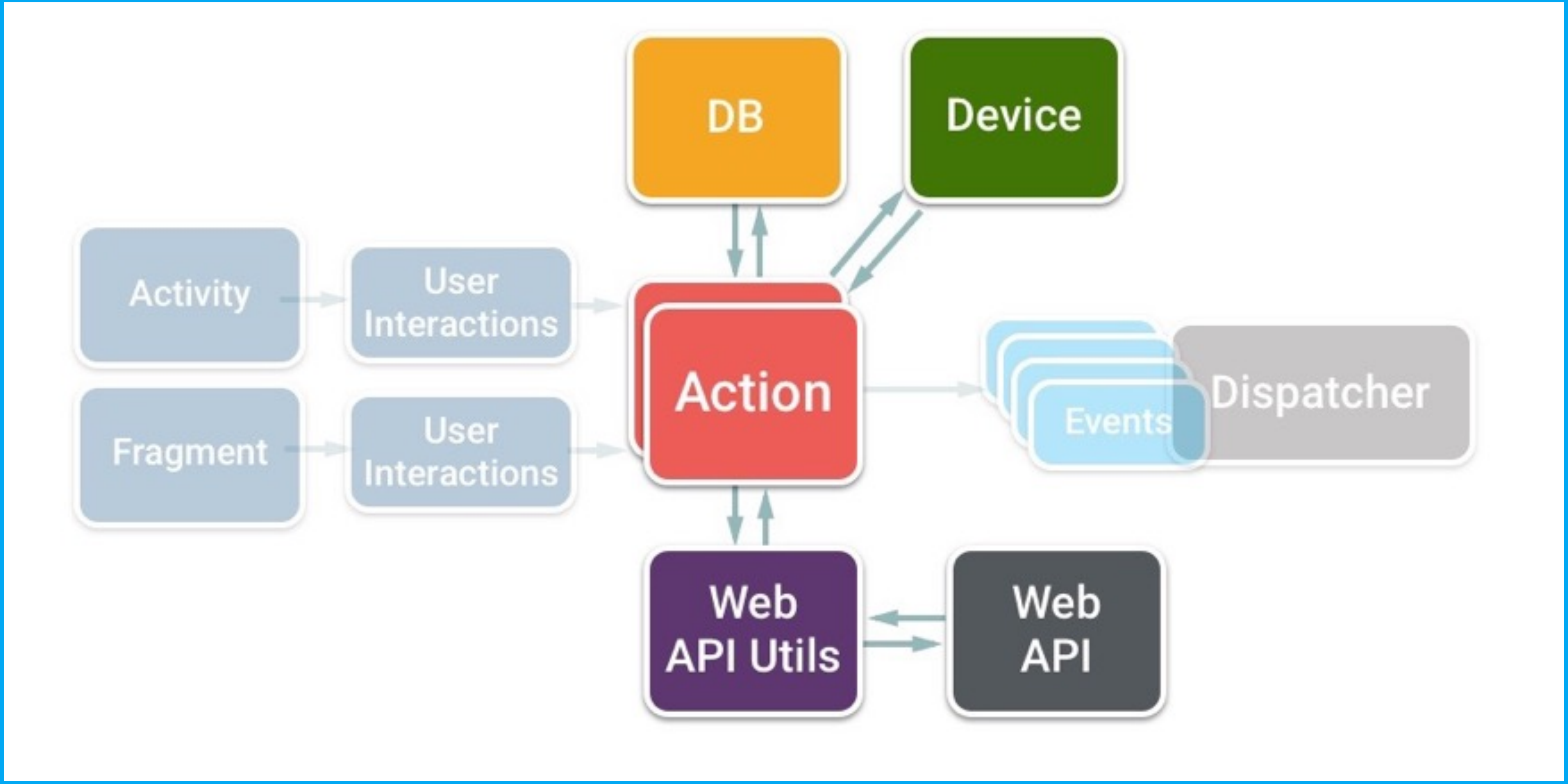
Flux: Action

Viewからの入力によりデータが流れてくる



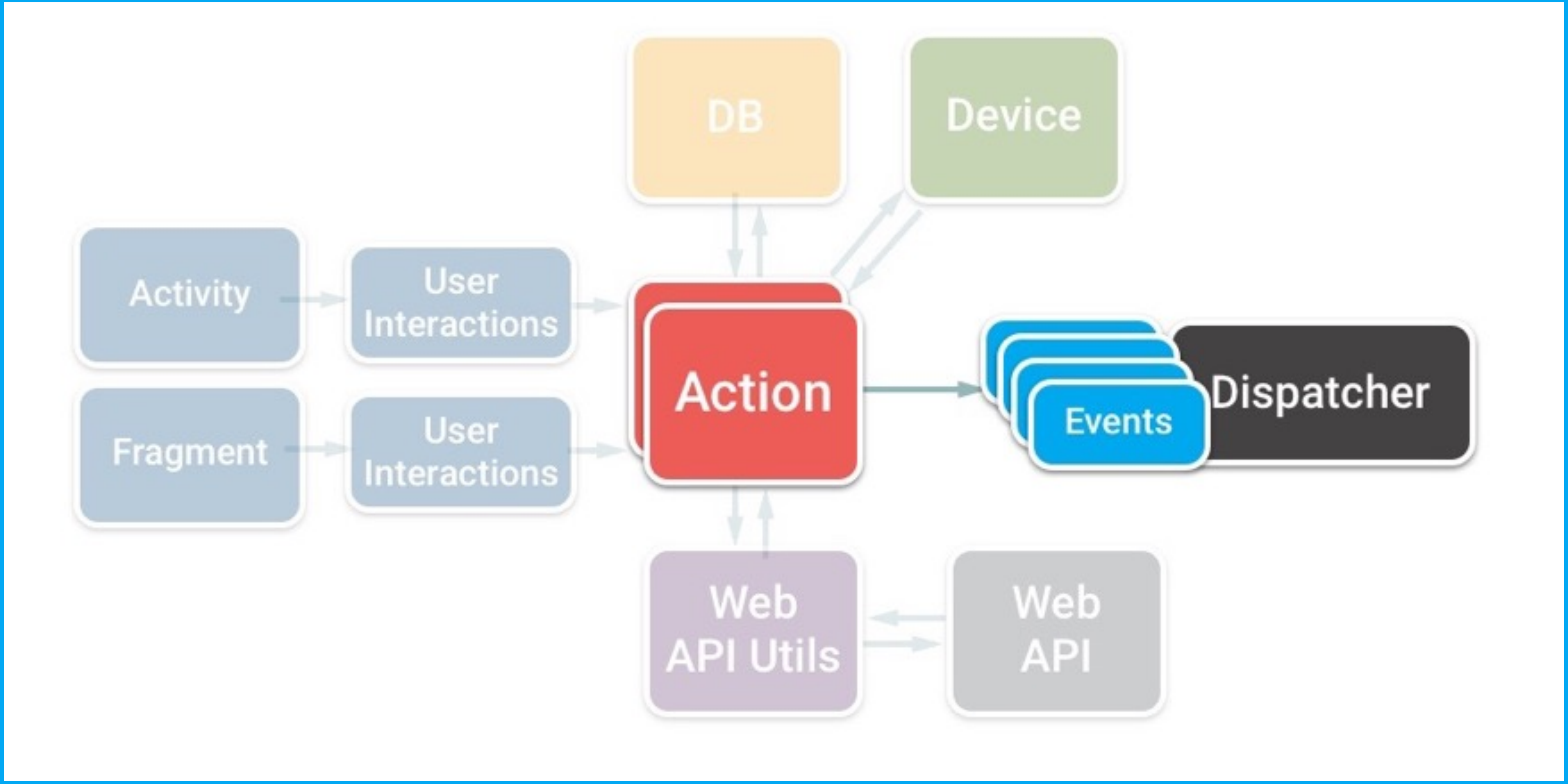
Flux: Action

データソースに必要となるデータを取りに行く



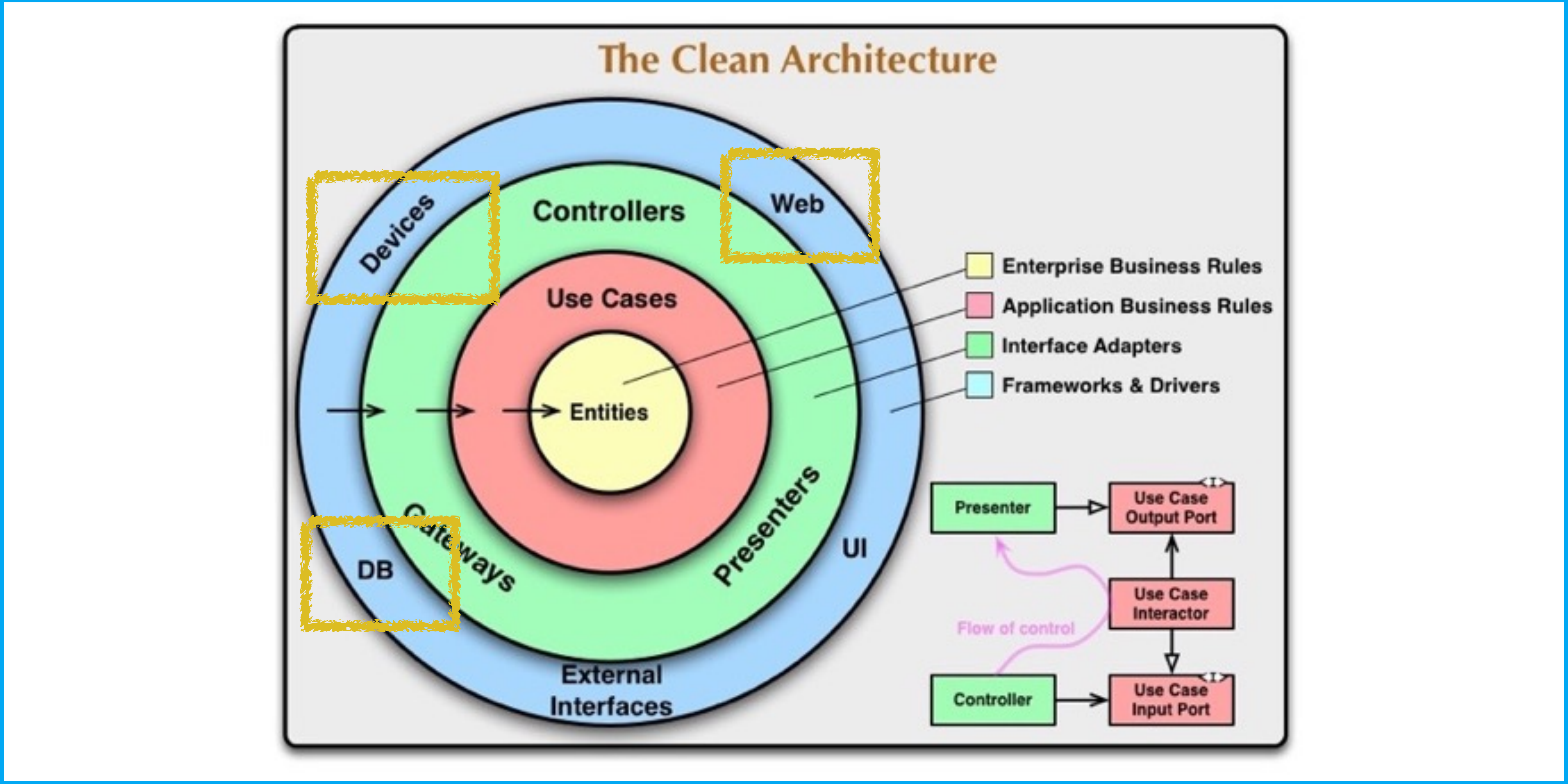
Flux: Action

データが集まったらDispatcherへデータを流す



Flux: Action

Web, DB, Devicesを外部サービスとして捉える



Flux: Action

```
@Inject GitHubApi gitHubApi;

private final Dispatcher dispatcher;

@Inject public UserSearchAction(Dispatcher dispatcher) {
  this.dispatcher = dispatcher;
}

public void findFollower(String userId) { ... }

public void findFollower(String userId, int nextPage) {
  gitHubApi.followers(userId, nextPage)
      .doOnSubscribe(() -> dispatchState(LoadingState.LOADING))
      .subscribe(users -> {
        dispatcher.dispatch(new SearchResultListChangedEvent(
            userId, users, users.nextPage()));
        dispatchState(users.hasMore()
            ? LoadingState.LOADABLE
            : LoadingState.FINISHED);
      }, ...);
}
```

Flux: Action

Tips: 外部I/Fの戻り値をRxで統一しておく

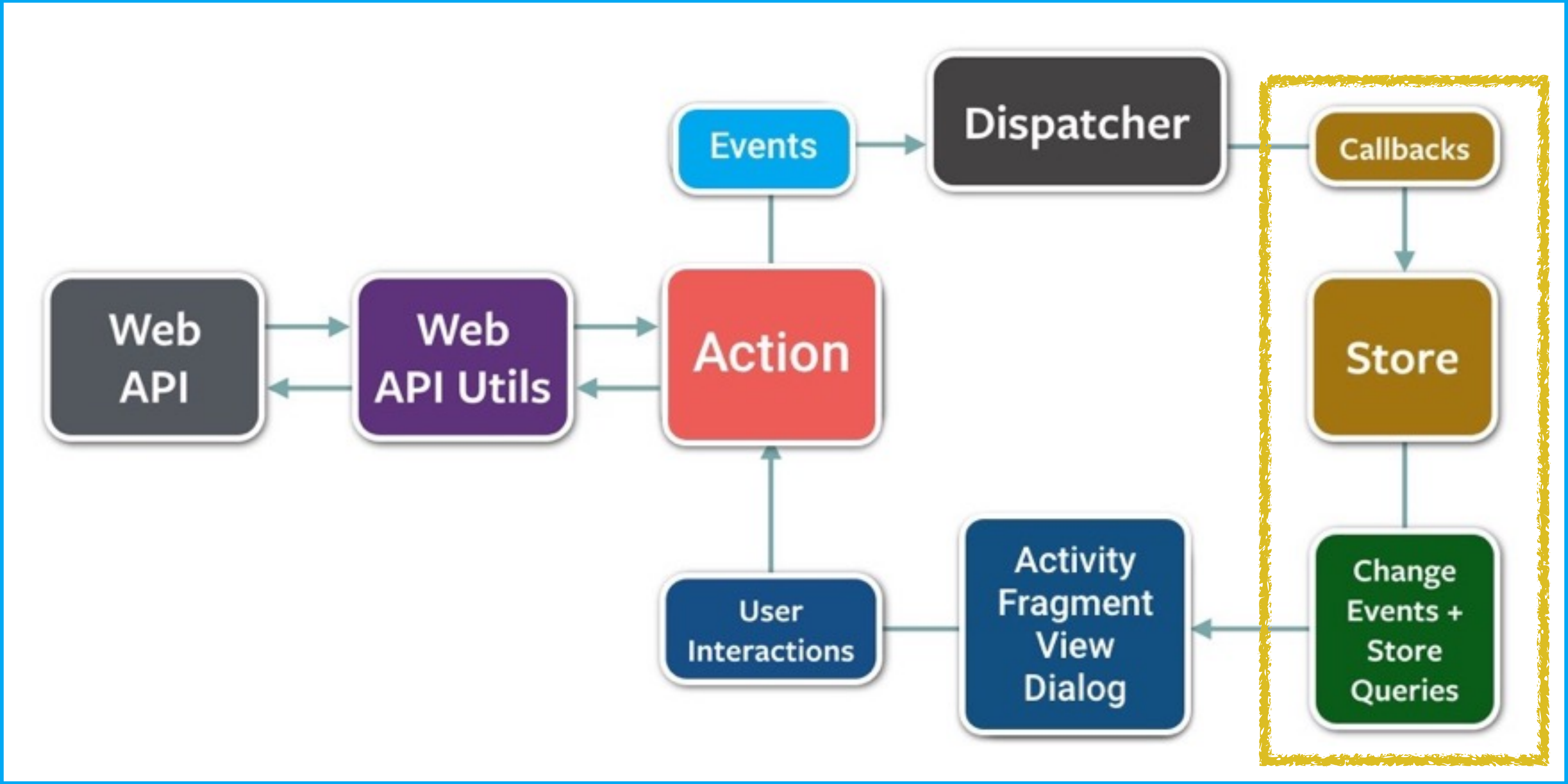
```
@Inject GitHubApi gitHubApi;

public void findFollower(String userId, int nextPage) {
  Observable.zip(
      gitHubApi.followers(userId, nextPage),
      gitHubApi.user(userId),
      this::doSomething)
    .subscribe(...)
}
```

Flux: Action

Tips: キャンセル処理が必要なとき

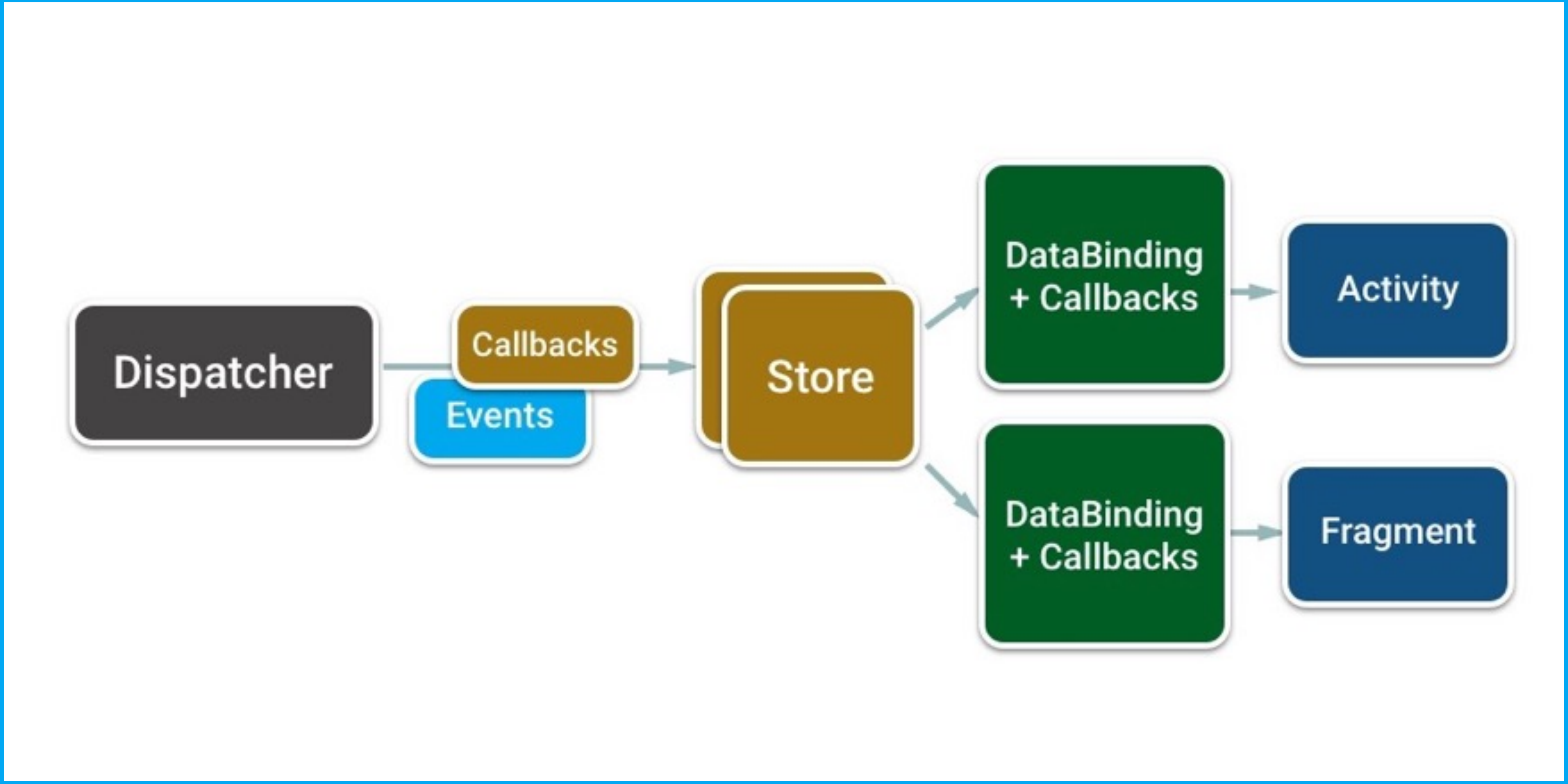
```
// e.g. 処理を呼び出し側でキャンセルする

public Subscription findFollower(String userId, int nextPage) {
  return gitHubApi.followers(userId, nextPage)
      .subscribe(...);
}

// e.g. 処理が実行中ならキャンセルする

private Subscription subs = Subscriptions.empty();

public void findFollower(String userId, int nextPage) {
  if (!subs.isUnsubscribed()) subs.unsubscribe();
  subs = gitHubApi.followers(userId, nextPage)
      .subscribe(...);
}
```

Flux: Store


Events
Dispatcher
Callbacks
Web API
Web API Utils
Action
Store
User Interactions
Activity Fragment View Dialog
Change Events + Store Queries

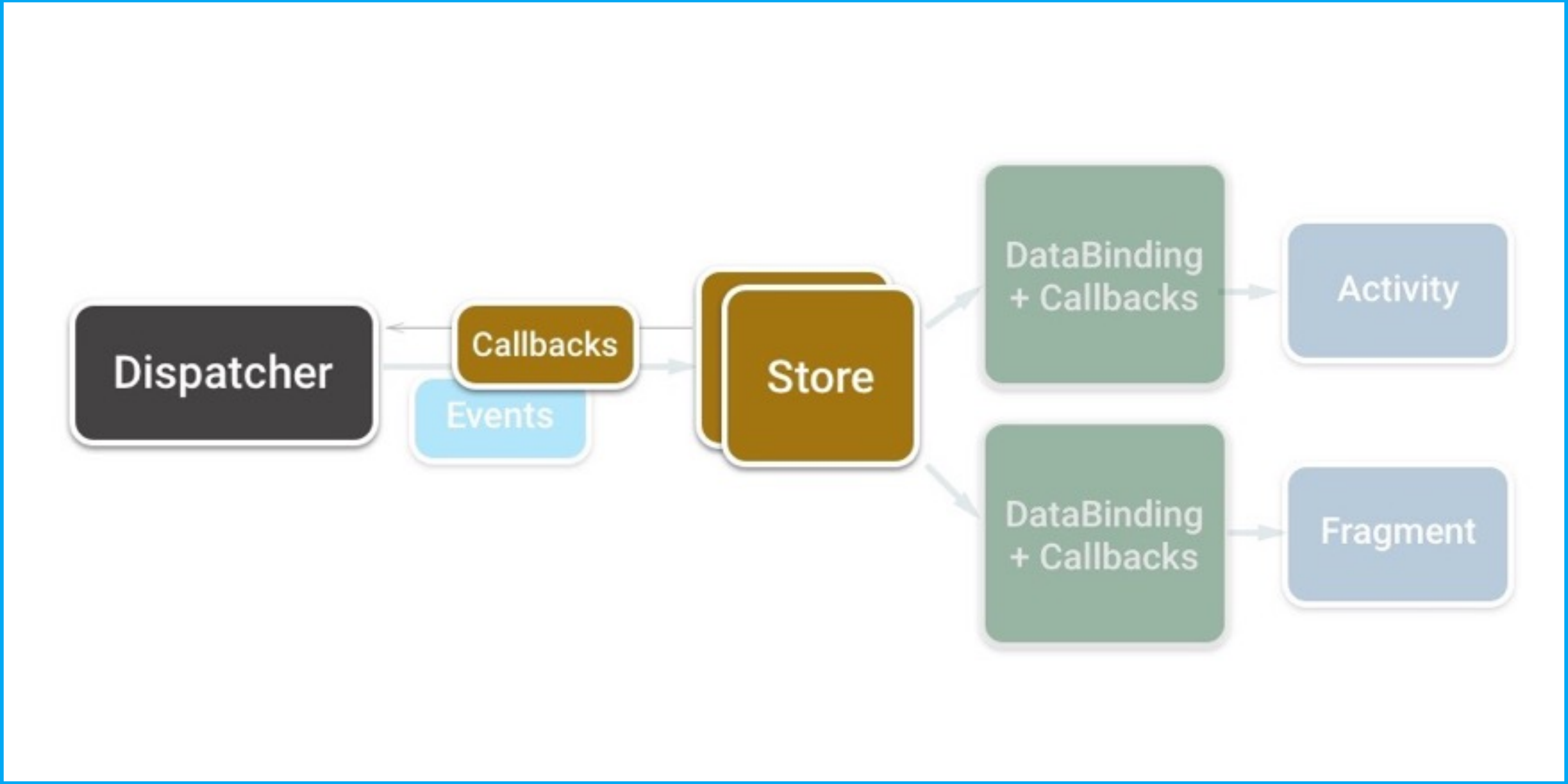
Flux: Store

Storeのデータフロー ₍₍ (ง ˙ω˙)ว ⁾⁾



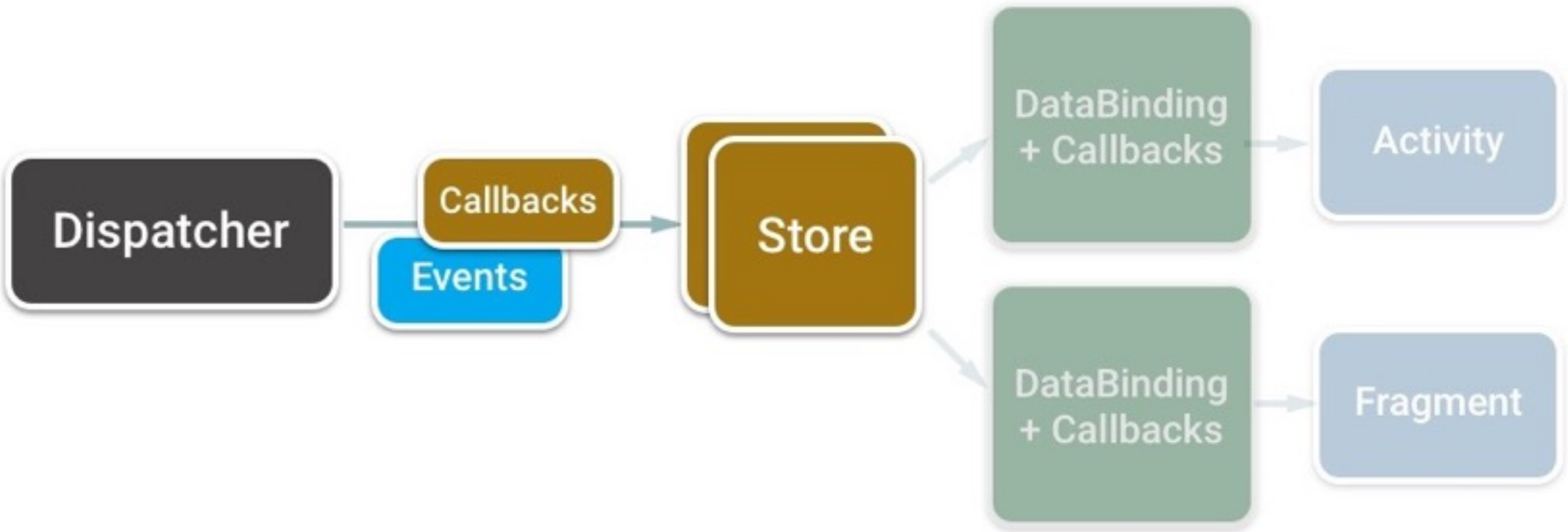
Flux: Store

データを受け取るためにCallbackを登録する



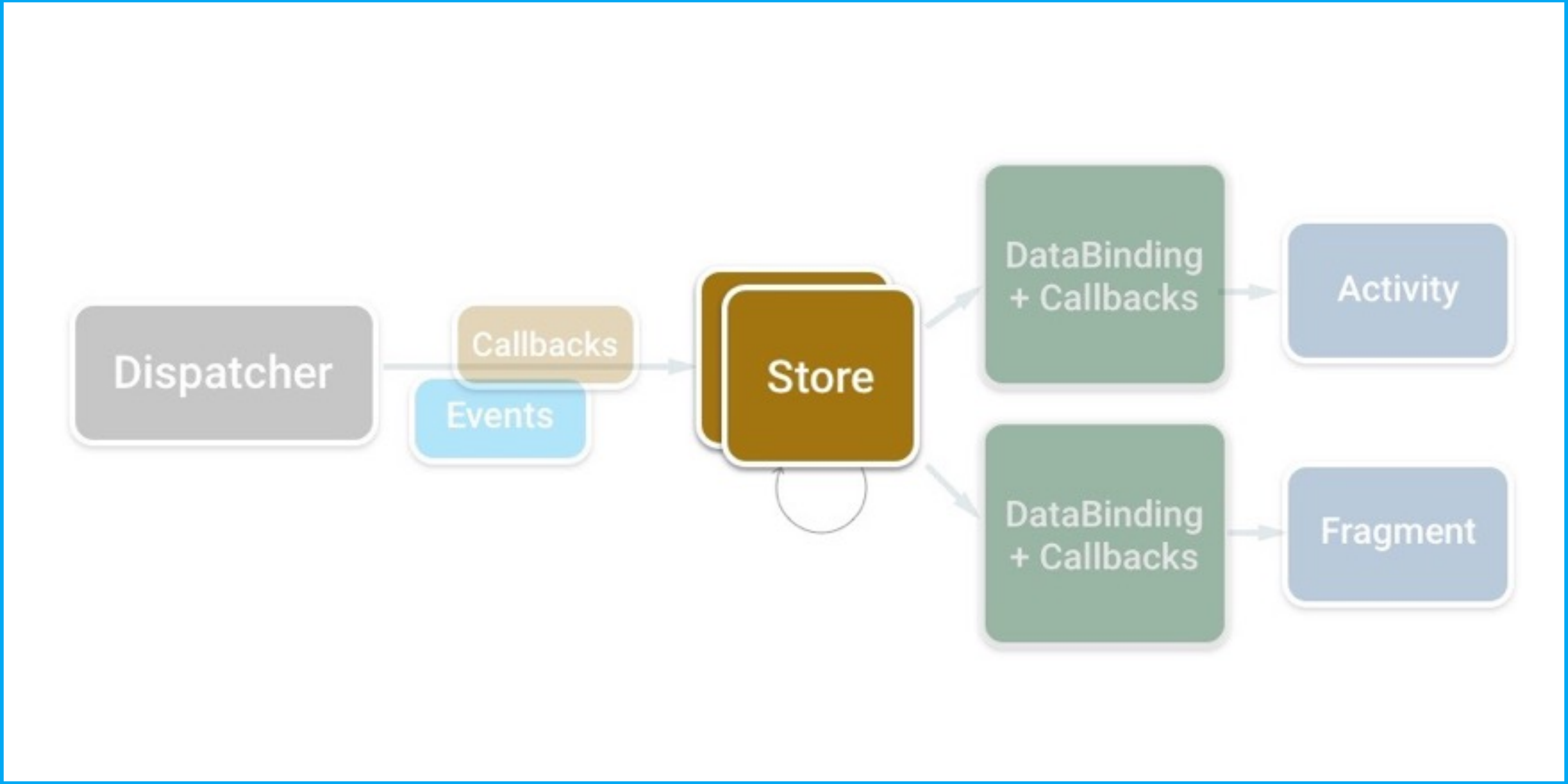
Flux: Store

ActionからデータStore流すとCallbackへ流れてくる



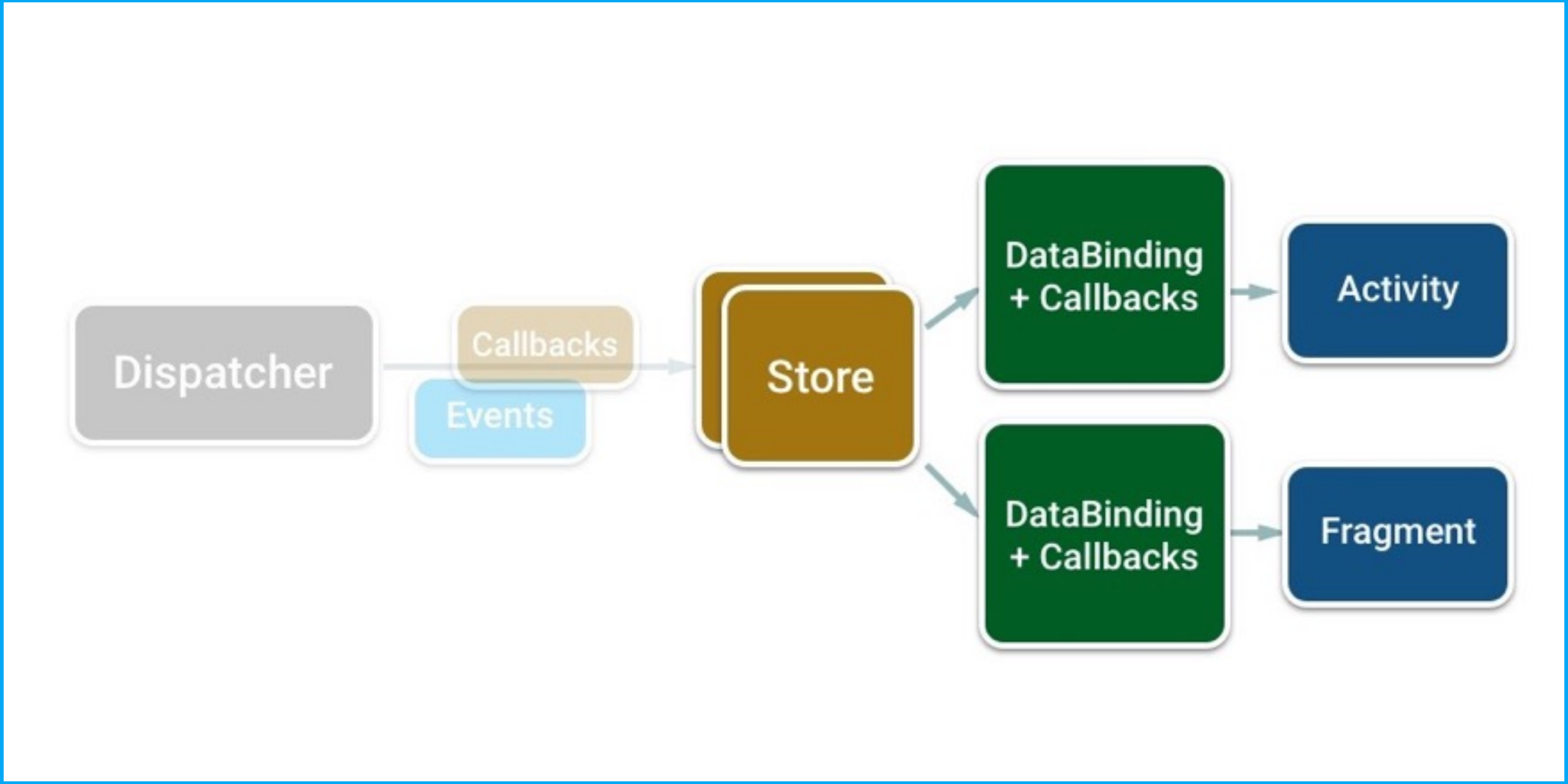
Flux: Store

データを受け取ったらStore内データを更新する



Flux: Store

データを更新したらViewへ変更を通知する



Flux: Store

DispatcherへCallbackを登録する

```
@ActivityScope
public class UserSearchStore {
  @Inject
  public UserSearchStore(Dispatcher dispatcher, ActivityLifecycleHook hook) {
    hook.addOnCreate(() -> dispatcher.register(this));
    hook.addOnDestroy(() -> dispatcher.unregister(this));
  }
```

Flux: Store

DispatcherへCallbackを登録する

```
@Singleton
~~@ActivityScope~~
public class UserSearchStore {
  @Inject
  public UserSearchStore(Dispatcher dispatcher~~, ActivityLifecycleHook hook~~) {
    dispatcher.register(this);
    ~~hook.addOnCreate(() -> dispatcher.register(this));~~
    ~~hook.addOnDestroy(() -> dispatcher.unregister(this));~~
  }
```

Flux: Store

Callback処理で自身の状態を更新する

```
private final ObservableField<LoadingState> state = new
ObservableField<>(LoadingState.LOADABLE);

@Subscribe(threadMode = ThreadMode.MAIN)
public void on(SearchLoadingStateChangedEvent event) {
  state.set(event.state);
}
```

Flux: Store

状態変更を通知するためのメソッドを公開する

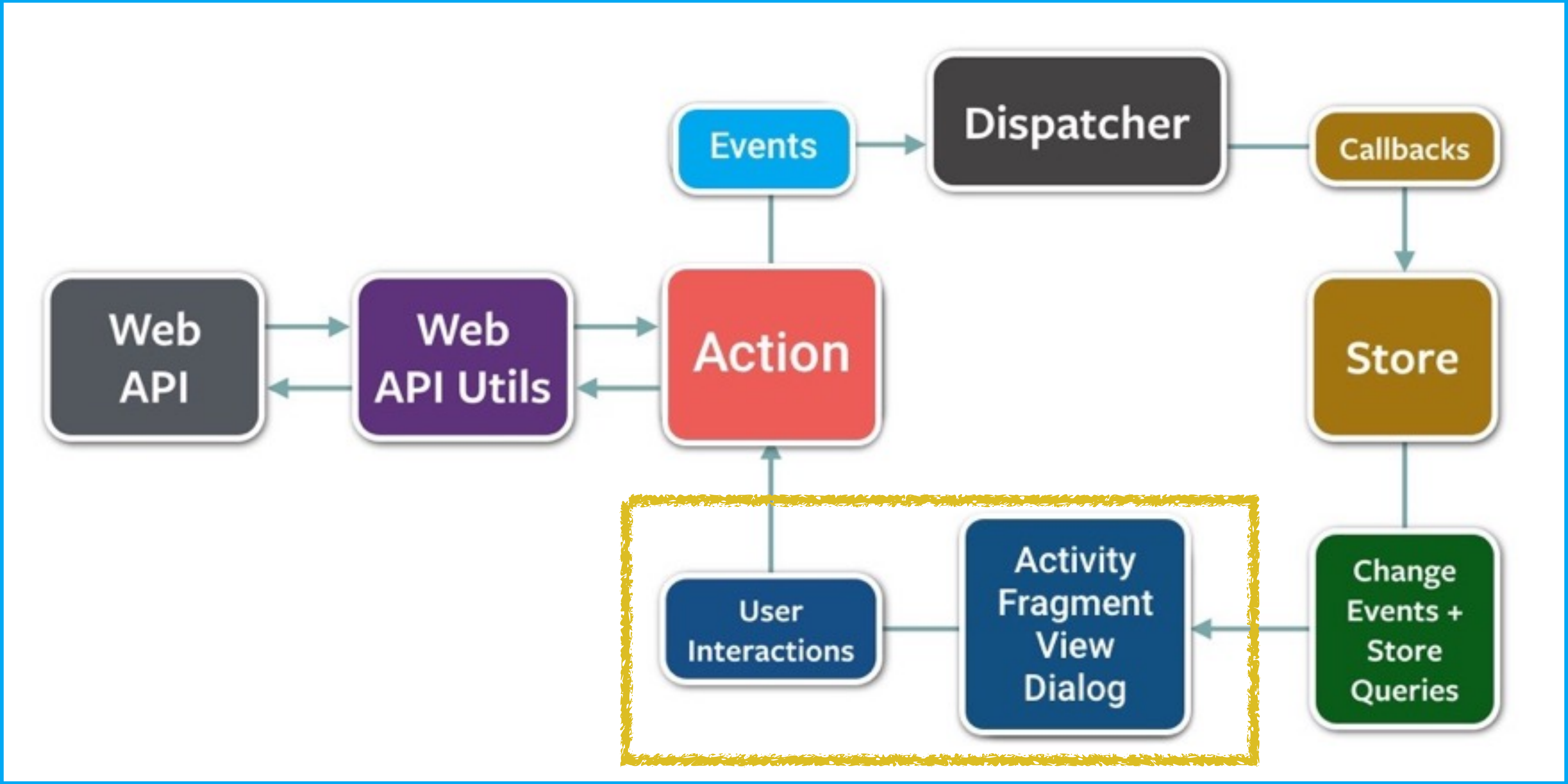
```
private final ObservableField<LoadingState> state = new
ObservableField<>(LoadingState.LOADABLE);

public Disposer addOnLoadingStateChanged(
    OnFieldChangedCallback<LoadingState> cb) {
  state.addOnPropertyChangedCallback(cb);
  return Disposers.from(() -> removeOnLoadingStateChanged(cb));
}

public void removeOnLoadingStateChanged(
    OnFieldChangedCallback<LoadingState> cb) {
  state.removeOnPropertyChangedCallback(cb);
}
```

Flux: Store

Tips: ObservableXXの代わりにRxを使う

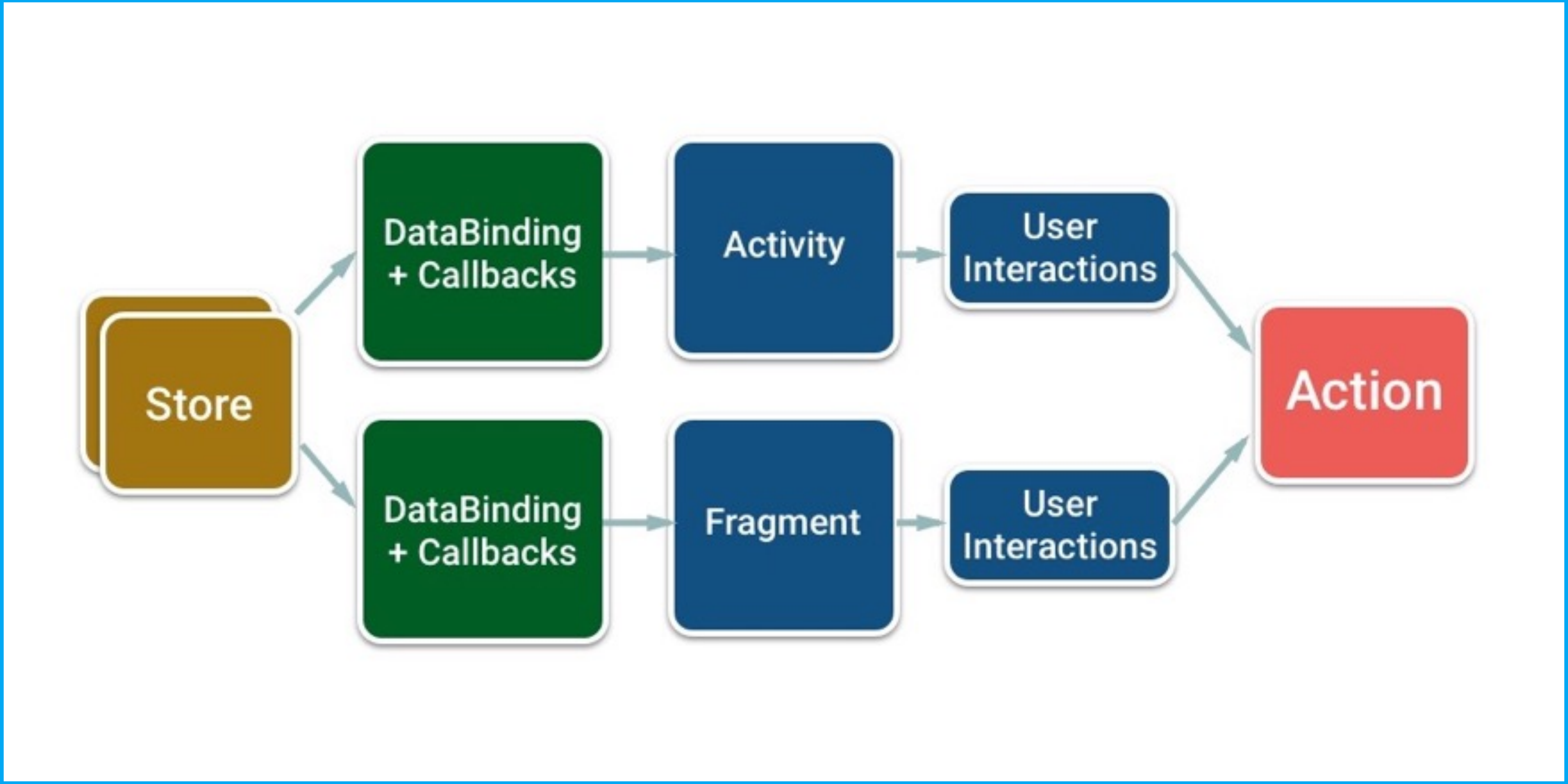
```
private final BehaviorSubject<LoadingState> state =
BehaviorSubject.create(LoadingState.LOADABLE);

public Observable<LoadingState> state() {
  return state.asObservable();
}

@Subscribe(threadMode = ThreadMode.MAIN)
public void on(SearchLoadingStateChangedEvent event) {
  state.onNext(event.state);
}
```

Flux: View


Events
Dispatcher
Callbacks
Web API
Web API Utils
Action
Store
User Interactions
Activity Fragment View Dialog
Change Events + Store Queries

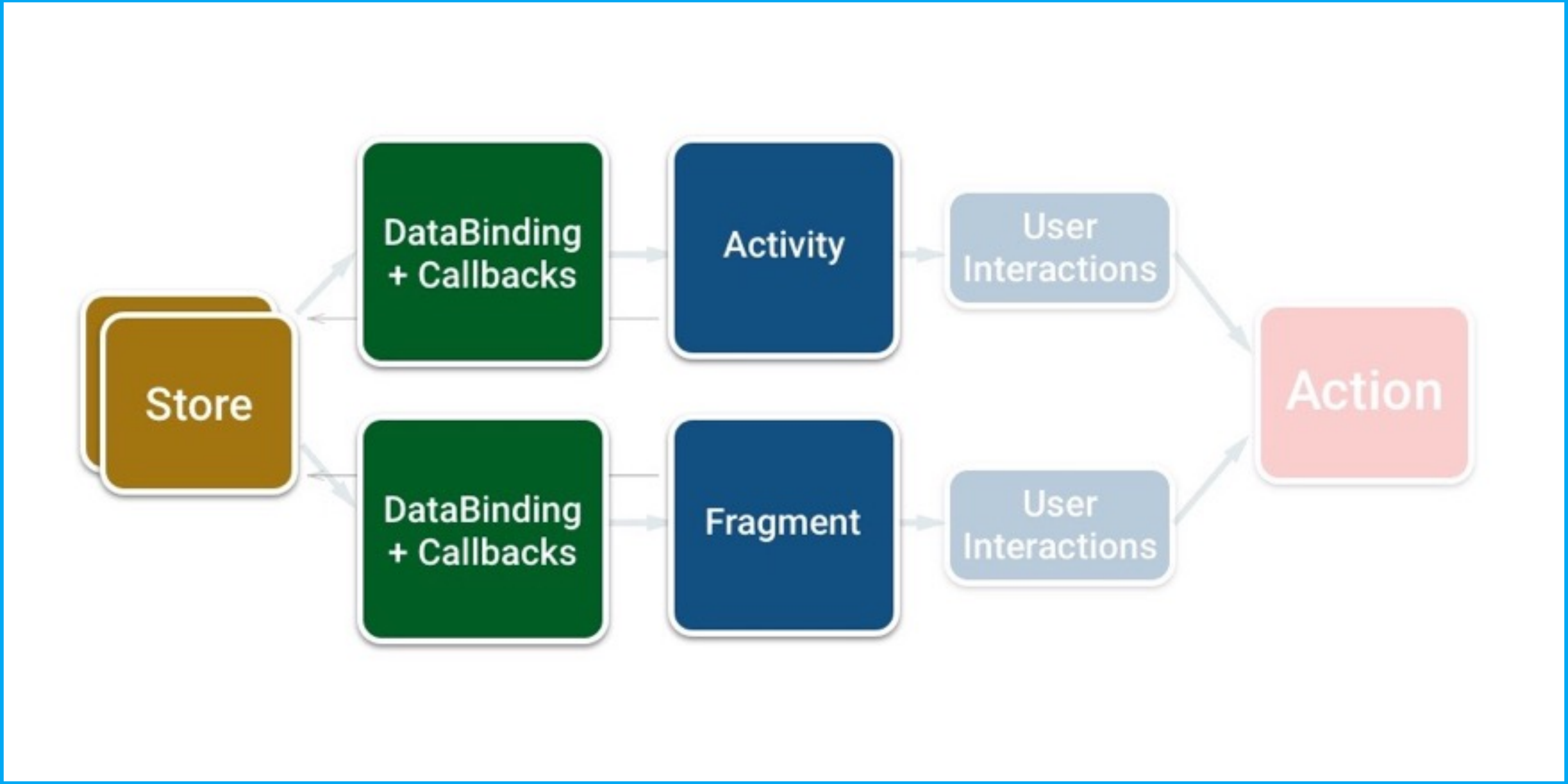
Flux: View

Viewのデータフロー ٩(๑´3｀๑)۶



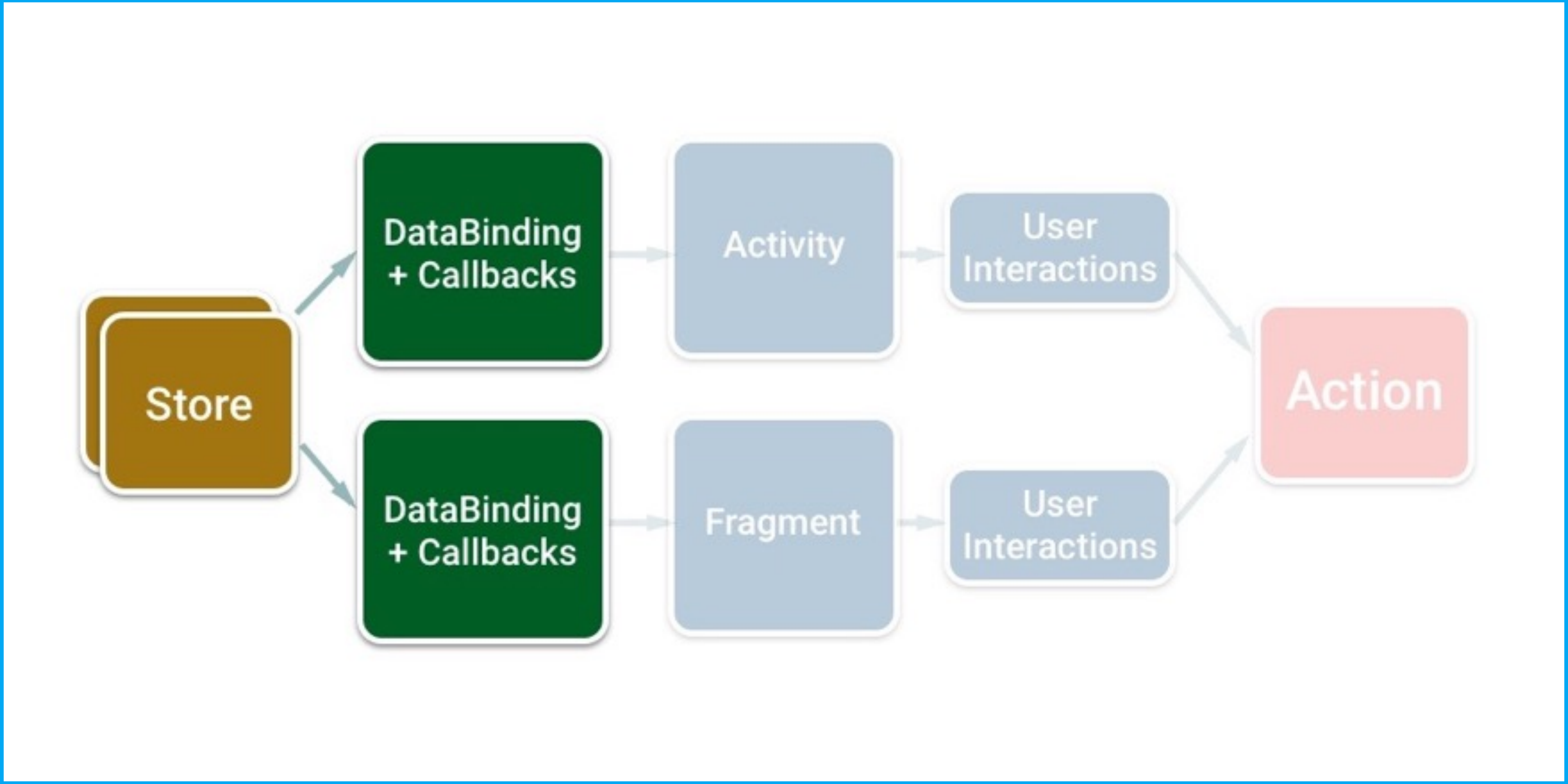
Flux: View

データを受け取るためStoreへCallbackを登録する



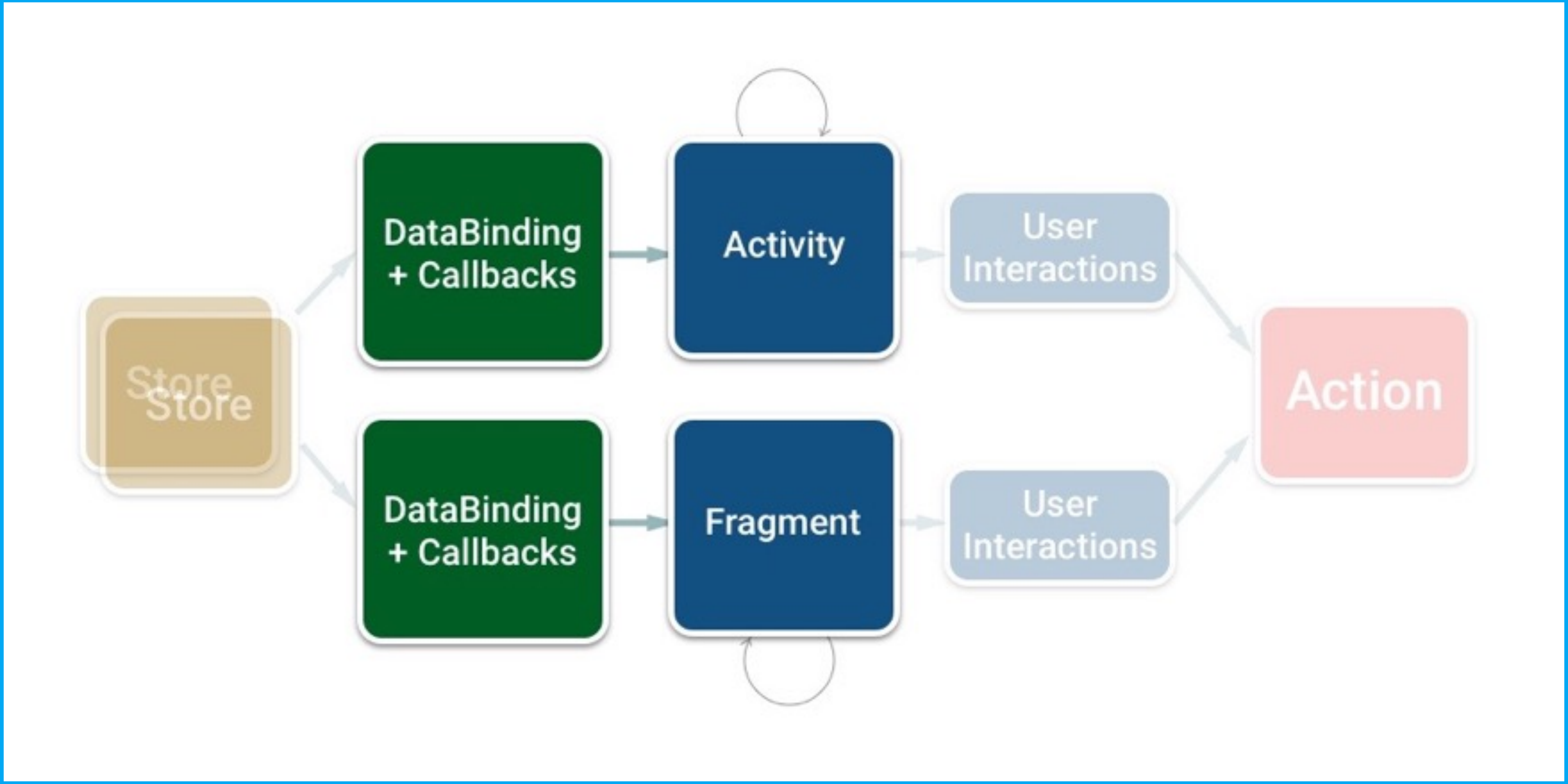
Flux: View

データが更新されたらCallbackが呼ばれる



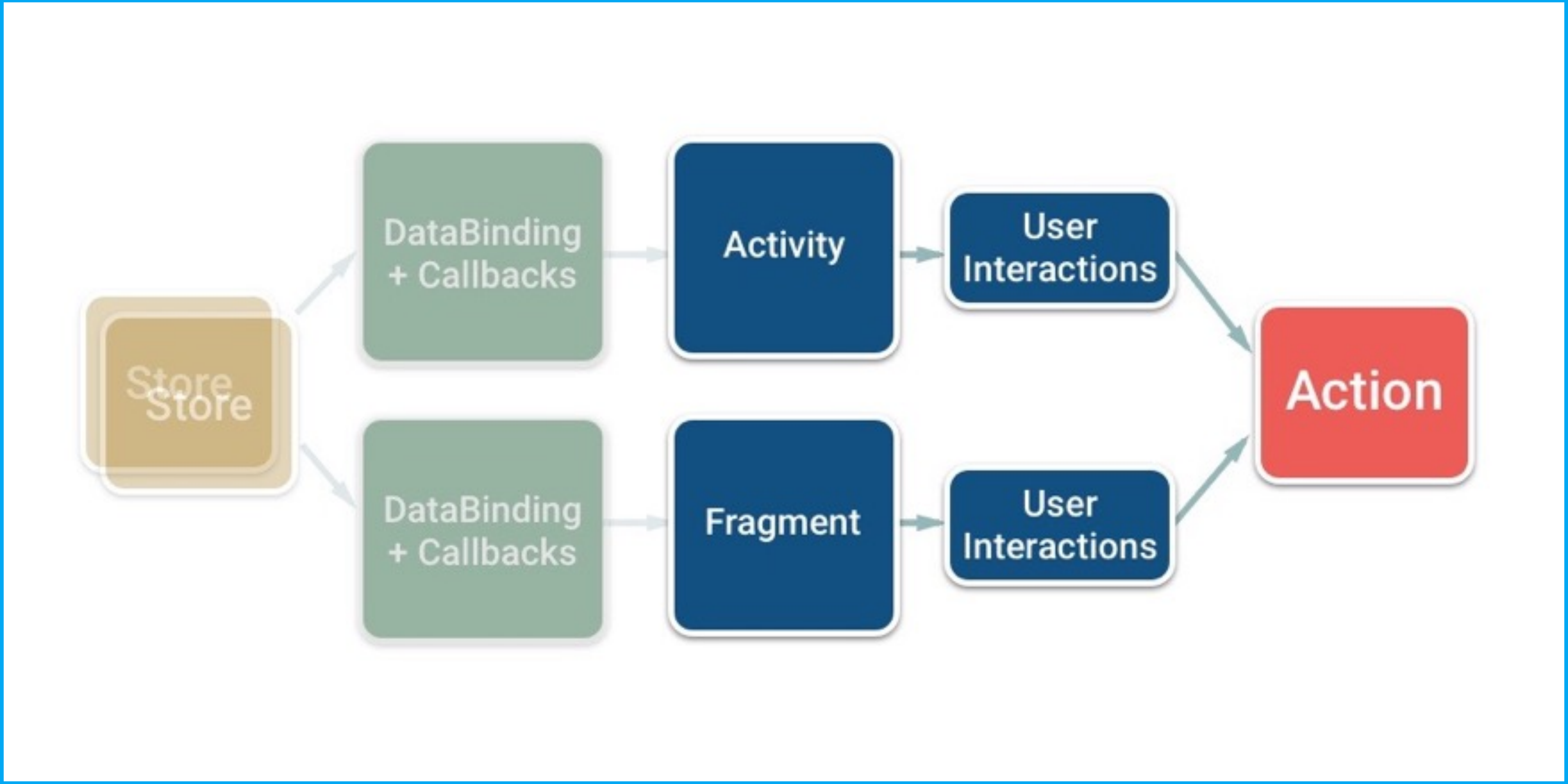
Flux: View

データを受け取ったら画面を更新する



Flux: View

新たな入力が発生したらActionへデータを流す



Flux: View

Storeの状態に応じてViewを更新する

```
@Inject UserSearchStore userSearchStore;

private final OnListChangedCallback<User> resultListChanged =
    new OnListChangedCallback<User>() {
      @Override
      public void onChanged(ObservableList<User> sender) {
        binding.setItemCount(sender.size());
      }
    };

public void onViewCreated(View view, @Nullable Bundle savedInstanceState) {
  ...
  userSearchStore.addOnListChanged(resultListChanged).addTo(this);
}
```

Flux: View

Storeの状態に応じてAdapterを更新する

```
@Inject
public UserSearchListAdapter(UserSearchStore store,
    ActivityLifecycleHook hook) {
  this.store = store;
  OnListChangedCallback<User> cb = OnListChangedCallback.delegateTo(this);
  hook.addOnCreate(() -> store.addOnListChanged(cb));
  hook.addOnDestroy(() -> store.removeOnListChanged(cb));
}

@Override
public void onBindViewHolder(ViewHolder holder, int position) {
  User user = store.getItemAt(position);
  ...
}

@Override
public int getItemCount() {
  return store.getItemCount();
}
```

Flux: View

Actionに処理を委譲する

```
@Inject UserSearchAction userSearchAction;

// SearchInputFragment
@Override public void onViewCreated(View view, …) {
  binding.searchButton.setOnClickListener(v -> {
    hideKeyboard(binding.searchInputText.getWindowToken());
    Optional.ofNullable(binding.searchInputText.getText())
        .map(Editable::toString)
        .filter(it -> !it.isEmpty())
        .ifPresent(userSearchAction::findFollower);
  });

// SearchResultFragment
@Override public void onLoadMore() {
  userSearchAction.findFollower(
      userSearchStore.getUserId(), userSearchStore.getNextPage());
}
```

Flux: View

Tips: ObservableXXの代わりにRxを使う

```
import com.trello.rxlifecycle.components.support.RxFragment;

public class SearchResultFragment extends RxFragment {

  @Inject UserSearchStore userSearchStore;

  @Override
  public void onViewCreated(View view, Bundle savedInstanceState) {
    ...
    userSearchStore.state()
        .map(it -> it == LoadingState.LOADING)
        .compose(bindToLifecycle())
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(binding::setIsLoading);
  }
```

Conclusion

Pros :)

- View間の依存が激減して、圧倒的感謝！
- 役割が明確なので開発者の実装が統一されやすい
- 単方向なのでコードが追いやすい

Conclusion

Cons :(

- シンプルな機能だと若干冗長に感じるときも…
- 解放ミスると即メモリリーク＼(^o^)／
- 基本トライ＆エラー(; ´∀｀)

Let's Flux de Relax :)