Managing State with RxJava

Jake Wharton

Why Reactive?

Unless you can model your entire system synchronously...

Why Reactive?

Unless you can model your entire system synchronously, a single asynchronous source breaks imperative programming.

Why Reactive?

```
interface UserManager {
  User getUser();
}
```

Why Reactive?

```
interface UserManager {
  User getUser();
  void setName(String name);
  void setAge(int age);
}
```

Why Reactive?

```
interface UserManager {
  User getUser();
  void setName(String name);
  void setAge(int age);
}

UserManager um = new UserManager();
```

Why Reactive?

```
interface UserManager {
  User getUser();
  void setName(String name);
  void setAge(int age);
}

UserManager um = new UserManager();
System.out.println(um.getUser());
```

Why Reactive?

```
interface UserManager {
  User getUser();
  void setName(String name);
  void setAge(int age);
}

UserManager um = new UserManager();
System.out.println(um.getUser());

um.setName("Jane Doe");
```

Why Reactive?

```
interface UserManager {
  User getUser();
  void setName(String name);
  void setAge(int age);
}

UserManager um = new UserManager();
System.out.println(um.getUser());

um.setName("Jane Doe");
System.out.println(um.getUser());
```

Why Reactive?

```
interface UserManager {
  User getUser();
  void setName(String name); // <-- now async
  void setAge(int age); // <-- now async
}
```

Why Reactive?

```
interface UserManager {
  User getUser();
  void setName(String name);
  void setAge(int age);
}

UserManager um = new UserManager();
System.out.println(um.getUser());

um.setName("Jane Doe");
System.out.println(um.getUser());
```

Why Reactive?

```
interface UserManager {
  User getUser();
  void setName(String name, Runnable callback);
  void setAge(int age, Runnable callback);
}
```

Why Reactive?

```
interface UserManager {
  User getUser();
  void setName(String name, Runnable callback);
  void setAge(int age, Runnable callback);
}

UserManager um = new UserManager();
System.out.println(um.getUser());

um.setName("Jane Doe", () -> {
  System.out.println(um.getUser());
});
```

Why Reactive?

```
interface UserManager {
  User getUser();
  void setName(String name, Listener listener);
  void setAge(int age, Listener listener);

  interface Listener {
    void success(User user);
    void failure(IOException e);
  }
}
```

Why Reactive?

```
UserManager um = new UserManager();
System.out.println(um.getUser());

um.setName("Jane Doe");
```

Why Reactive?

```
UserManager um = new UserManager();
System.out.println(um.getUser());

um.setName("Jane Doe", new UserManager.Listener() {
  @Override public void success() {
    System.out.println(um.getUser());
  }

  @Override public void failure(IOException e) {
    // TODO show the error...
  }
});
```

Why Reactive?

```
UserManager um = new UserManager();
System.out.println(um.getUser());

um.setName("Jane Doe", new UserManager.Listener() {
  @Override public void success() {
    System.out.println(um.getUser());
  }
  @Override public void failure(IOException e) {
    // TODO show the error...
  }
});

um.setAge(40, new UserManager.Listener() {
  @Override public void success() {
    System.out.println(um.getUser());
  }
  @Override public void failure(IOException e) {
    // TODO show the error...
  }
});
```

Why Reactive?

```
UserManager um = new UserManager();
System.out.println(um.getUser());

um.setName("Jane Doe", new UserManager.Listener() {
  @Override public void success() {
    System.out.println(um.getUser());

    um.setAge(40, new UserManager.Listener() {
      @Override public void success() {
        System.out.println(um.getUser());
      }
      @Override public void failure(IOException e) {
        // TODO show the error...
      }
    });
  }
  @Override public void failure(IOException e) {
    // TODO show the error...
  }
});
```

Why Reactive?

```
public final class UserActivity extends Activity {
  private final UserManager um = new UserManager();

  @Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.user);
    TextView tv = (TextView) findViewById(R.id.user_name);
    tv.setText(um.getUser().toString());

    um.setName("Jane Doe", new UserManager.Listener() {
      @Override public void success() {
        tv.setText(um.getUser().toString());
      }
      @Override public void failure(IOException e) {
        // TODO show the error...
      }
    });
  }
}
```

Why Reactive?

```
public final class UserActivity extends Activity {
  private final UserManager um = new UserManager();

  @Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.user);
    TextView tv = (TextView) findViewById(R.id.user_name);
    tv.setText(um.getUser().toString());

    um.setName("Jane Doe", new UserManager.Listener() {
      @Override public void success() {
        tv.setText(um.getUser().toString());
      }
      @Override public void failure(IOException e) {
        // TODO show the error...
      }
    });
  }
}
```

Why Reactive?

```
public final class UserActivity extends Activity {
  private final UserManager um = new UserManager();

  @Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.user);
    TextView tv = (TextView) findViewById(R.id.user_name);
    tv.setText(um.getUser().toString());

    um.setName("Jane Doe", new UserManager.Listener() {
      @Override public void success() {
        if (isDestroyed()) {
          tv.setText(um.getUser().toString());
        }
      }
      @Override public void failure(IOException e) {
        // TODO show the error...
      }
    });
  }
}
```

Why Reactive?

```
public final class UserActivity extends Activity {
  private final UserManager um = new UserManager();

  @Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.user);
    TextView tv = (TextView) findViewById(R.id.user_name);
    tv.setText(um.getUser().toString());

    um.setName("Jane Doe", new UserManager.Listener() {
      @Override public void success() {
        if (isDestroyed()) {
          tv.setText(um.getUser().toString());
        }
      }
      @Override public void failure(IOException e) {
        // TODO show the error...
      }
    });
  }
}
```

Why Reactive?

```
public final class UserActivity extends Activity {
  private final UserManager um = new UserManager();

  @Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.user);
    TextView tv = (TextView) findViewById(R.id.user_name);
    tv.setText(um.getUser().toString());

    um.setName("Jane Doe", new UserManager.Listener() {
      @Override public void success() {
        if (isDestroyed()) {
          tv.setText(um.getUser().toString());
        }
      }
      @Override public void failure(IOException e) {
        // TODO show the error...
      }
    });
  }
}
```

Why Reactive?

```
public final class UserActivity extends Activity {
  private final UserManager um = new UserManager();

  @Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.user);
    TextView tv = (TextView) findViewById(R.id.user_name);
    tv.setText(um.getUser().toString());

    um.setName("Jane Doe", new UserManager.Listener() {
      @Override public void success() {
        runOnUiThread(new Runnable() {
          @Override public void run() {
            if (isDestroyed()) {
              tv.setText(um.getUser().toString());
            }
          }
        });
      }
      @Override public void failure(IOException e) {
        // TODO show the error...
      }
    });
  }
}
```

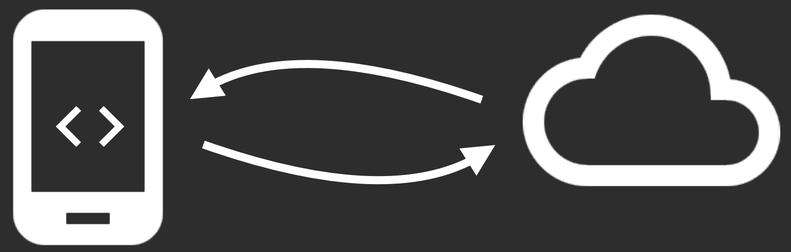
Why Reactive?

```
public final class UserActivity extends Activity {
  private final UserManager um = new UserManager();

  @Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.user);
    TextView tv = (TextView) findViewById(R.id.user_name);
    tv.setText(um.getUser().toString());

    um.setName("Jane Doe", new UserManager.Listener() {
      @Override public void success() {
        runOnUiThread(new Runnable() {
          @Override public void run() {
            if (isDestroyed()) {
              tv.setText(um.getUser().toString());
            }
          }
        });
      }
      @Override public void failure(IOException e) {
        // TODO show the error...
      }
    });
  }
}
```

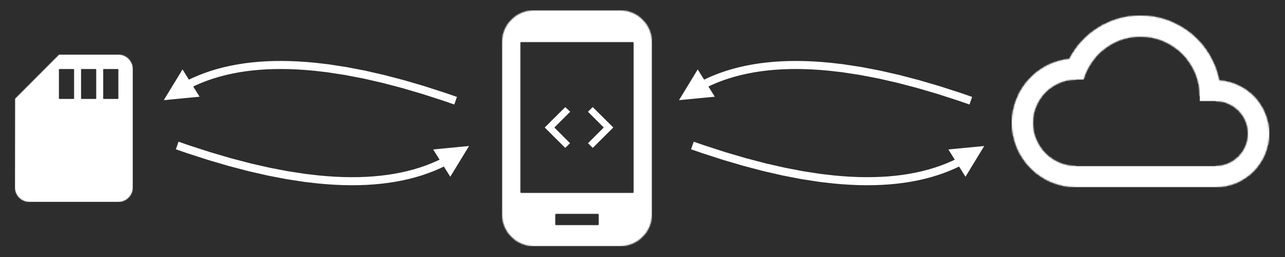
Why Reactive?

Why Reactive?

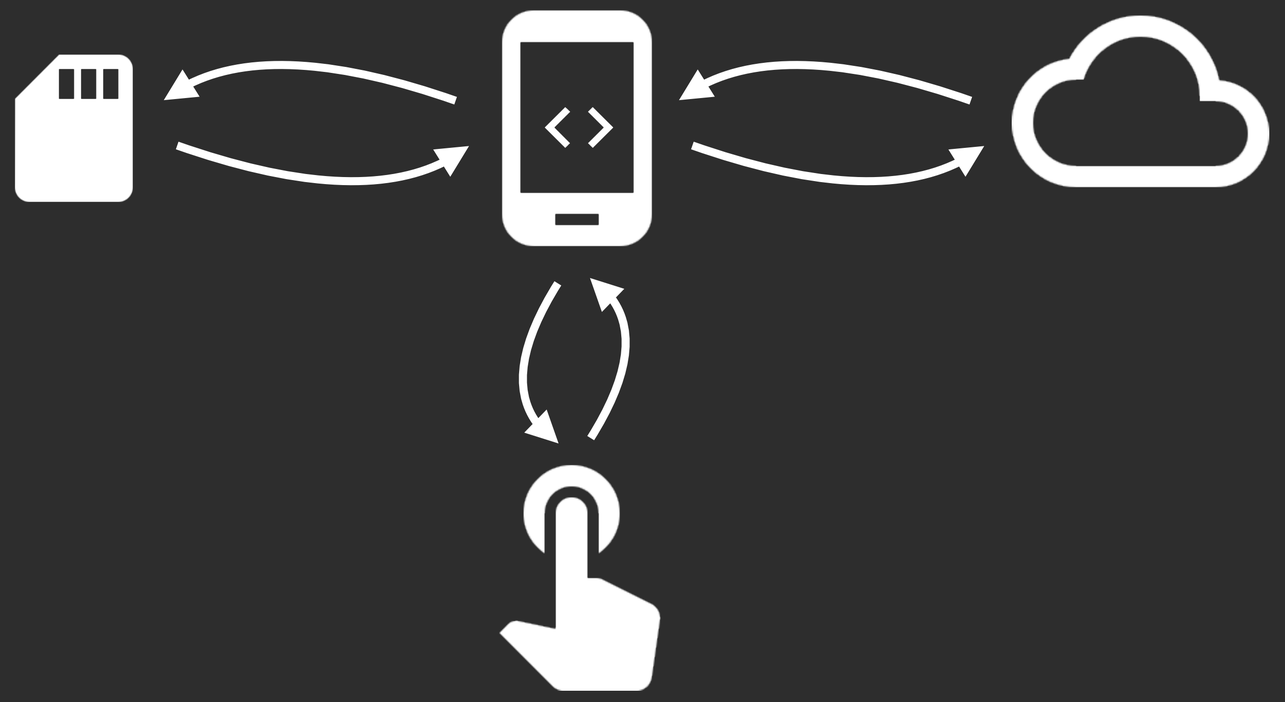
Why Reactive?



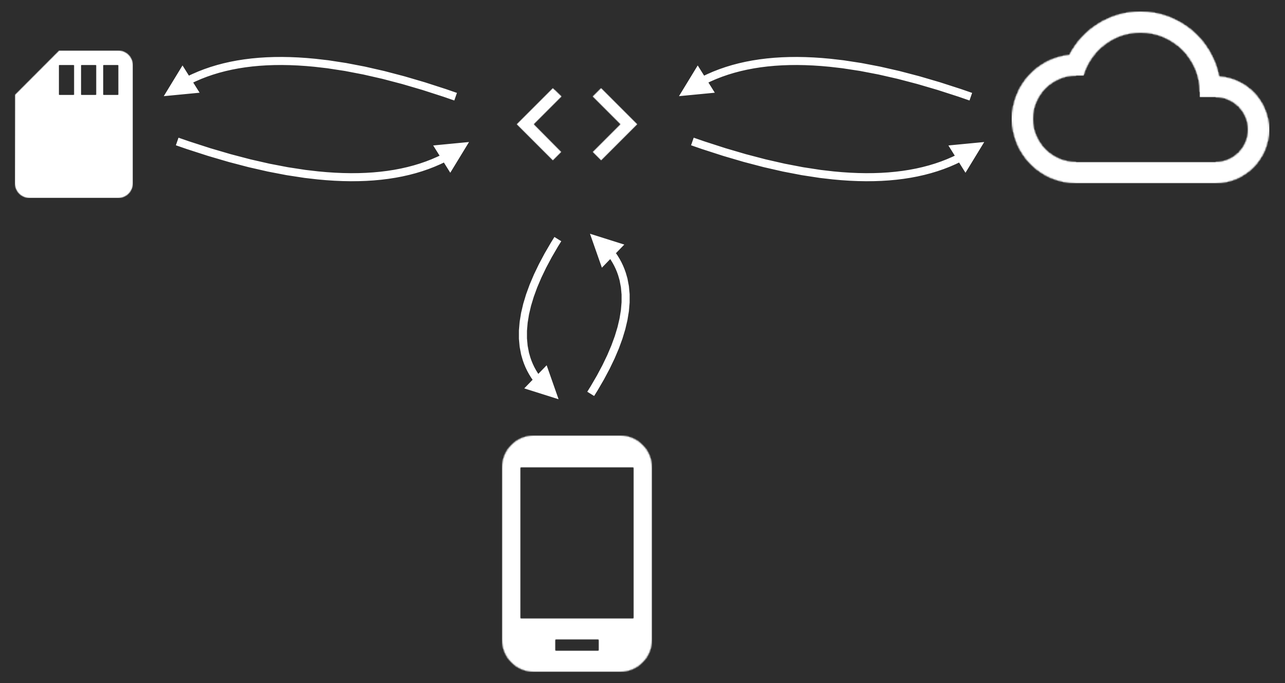
Why Reactive?



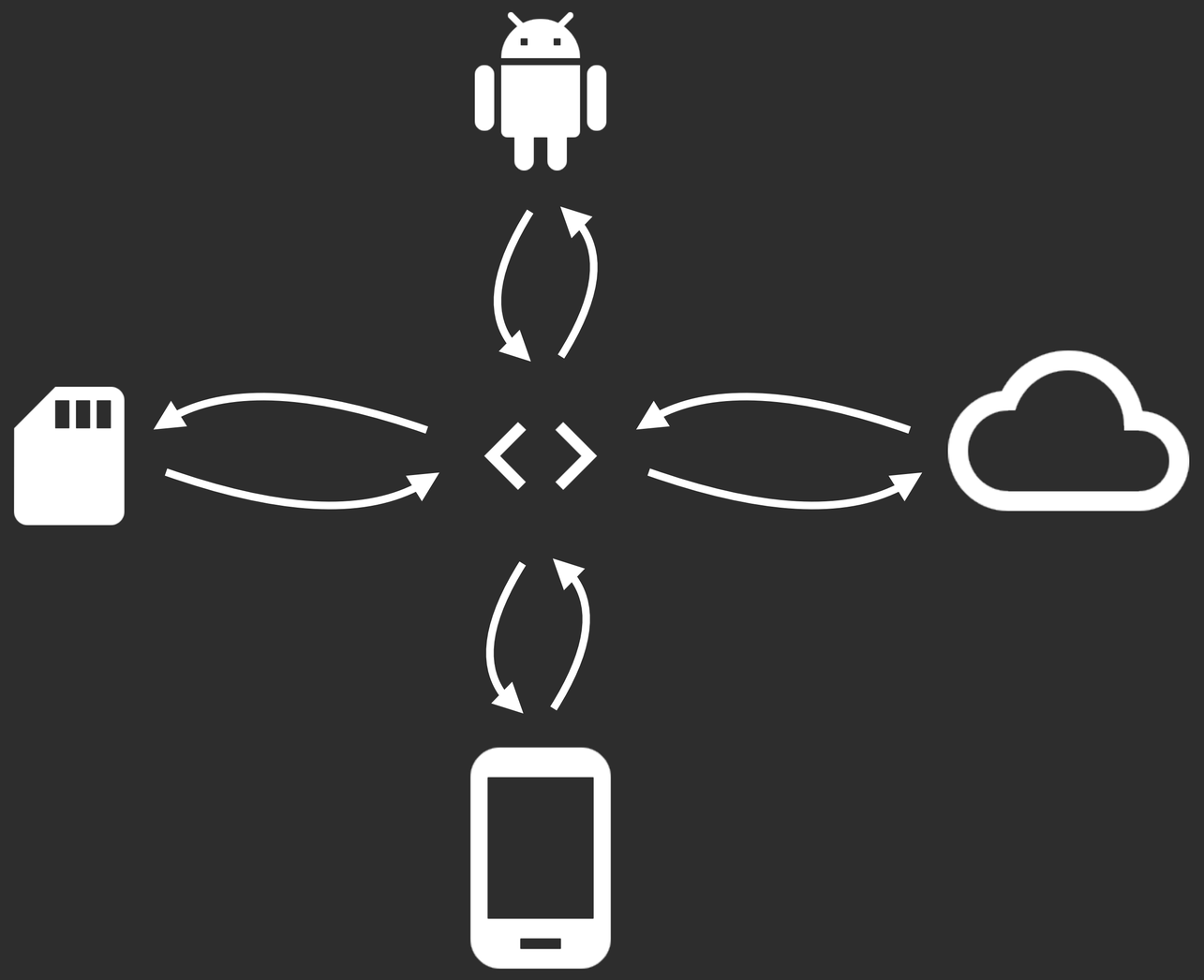
Why Reactive?



Why Reactive?



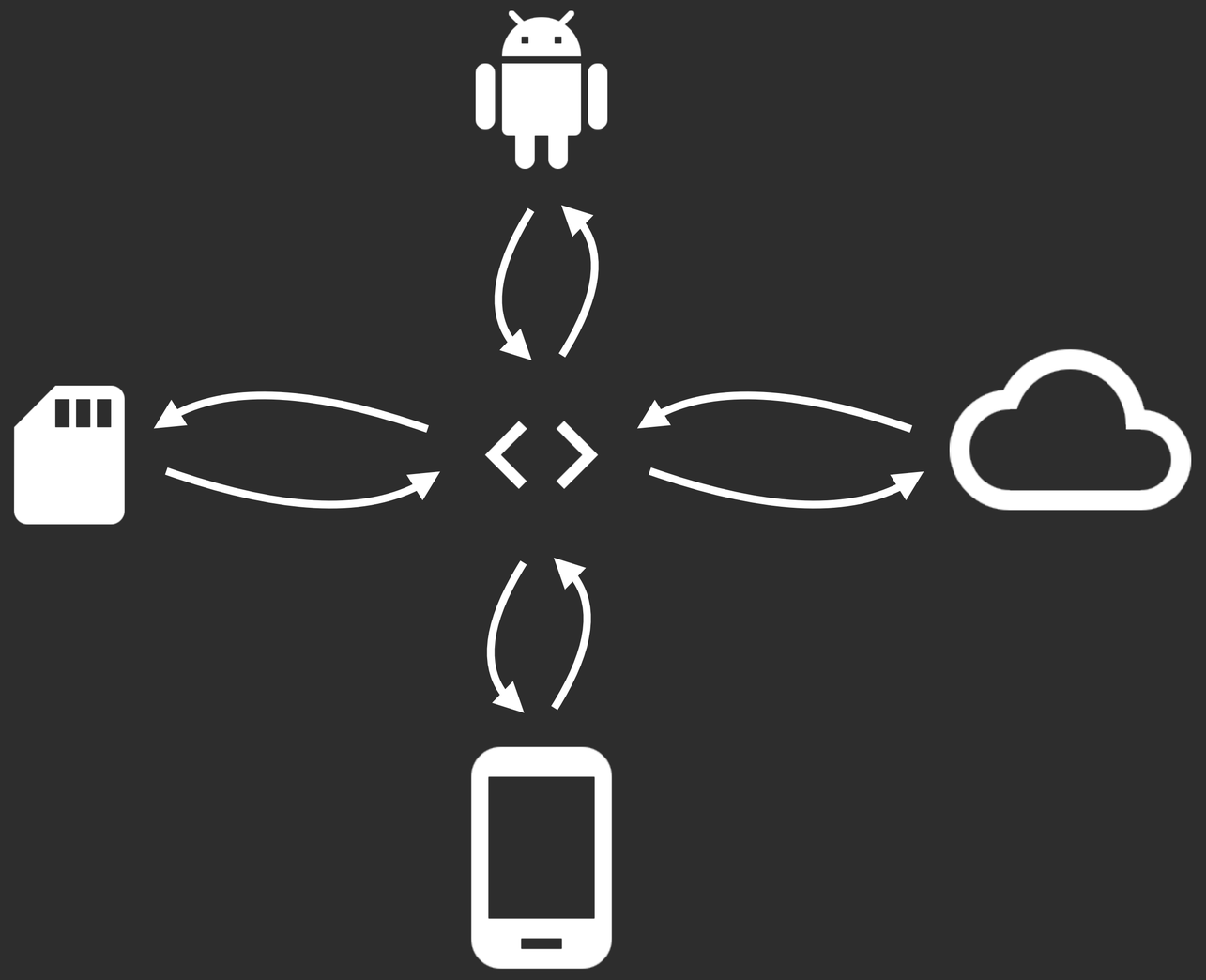
Why Reactive?



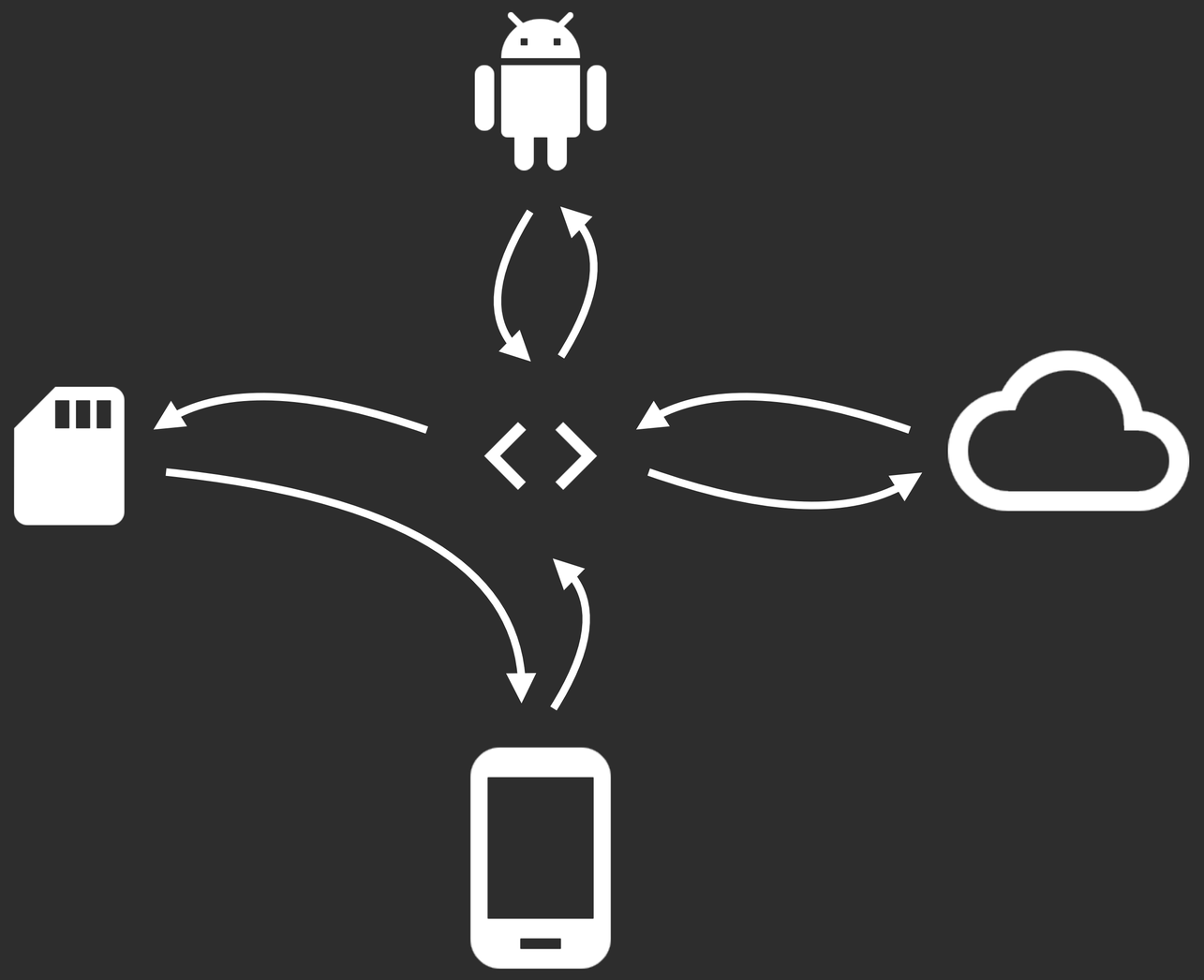
Why Reactive?

Unless you can model your entire system synchronously, a single asynchronous source breaks imperative programming.

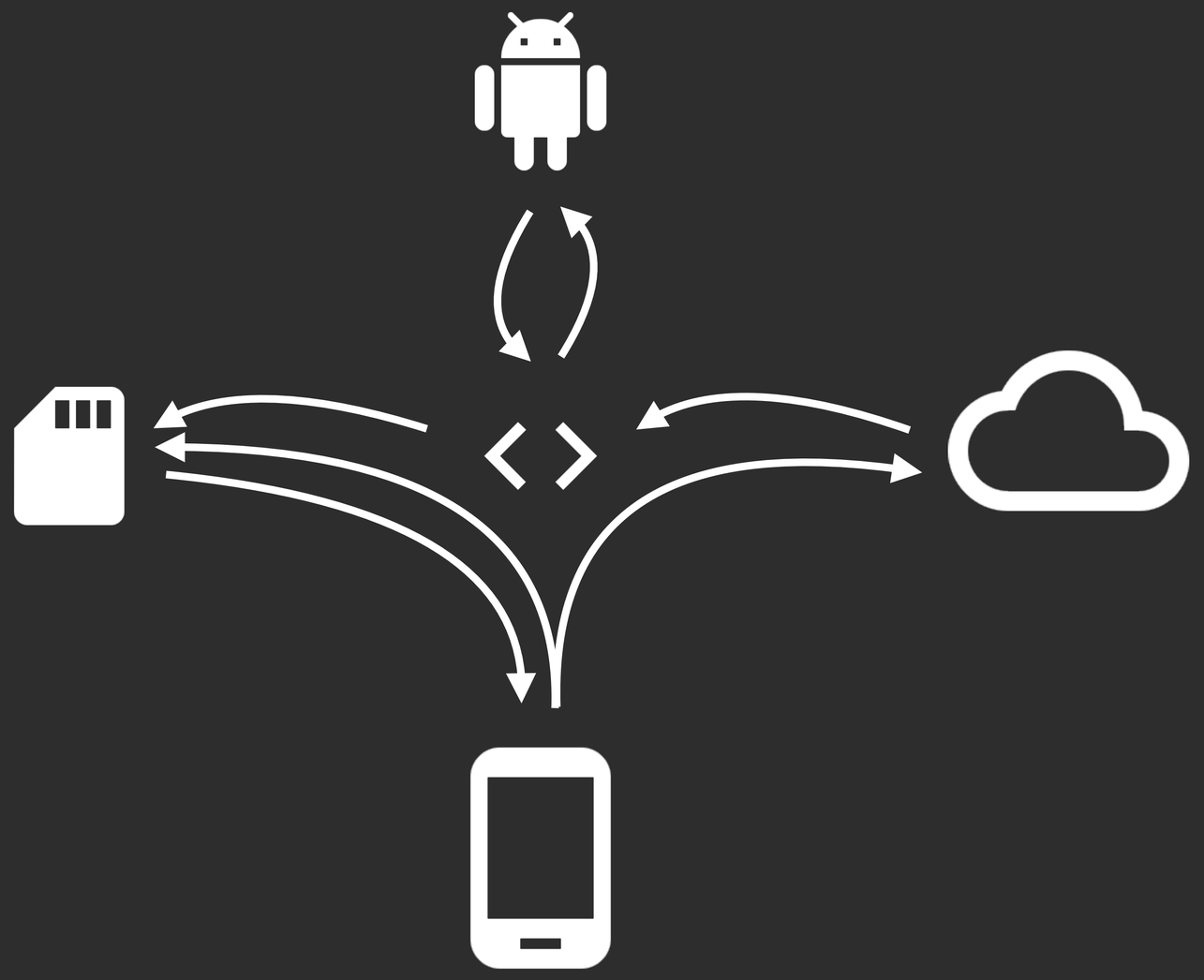
Why Reactive?



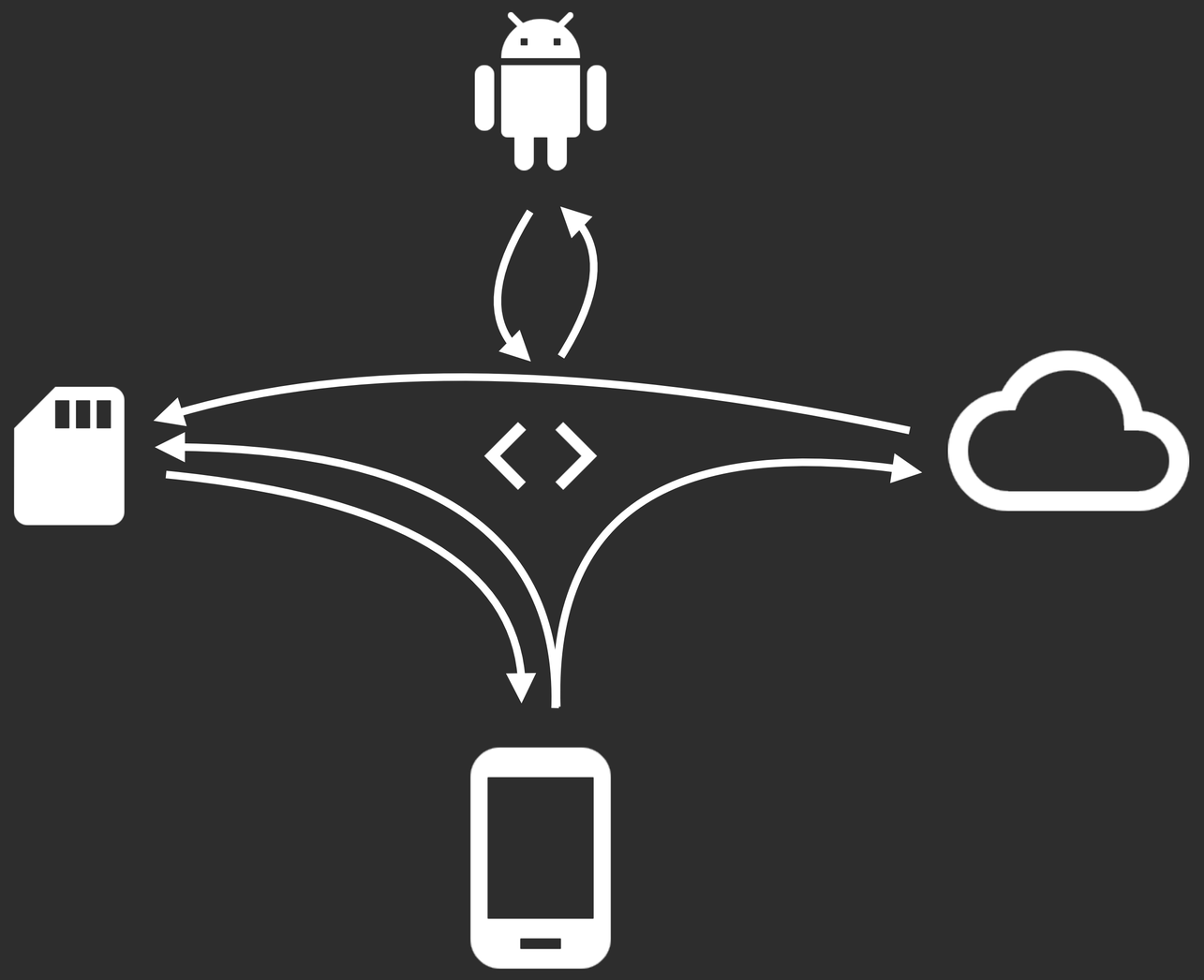
Why Reactive?



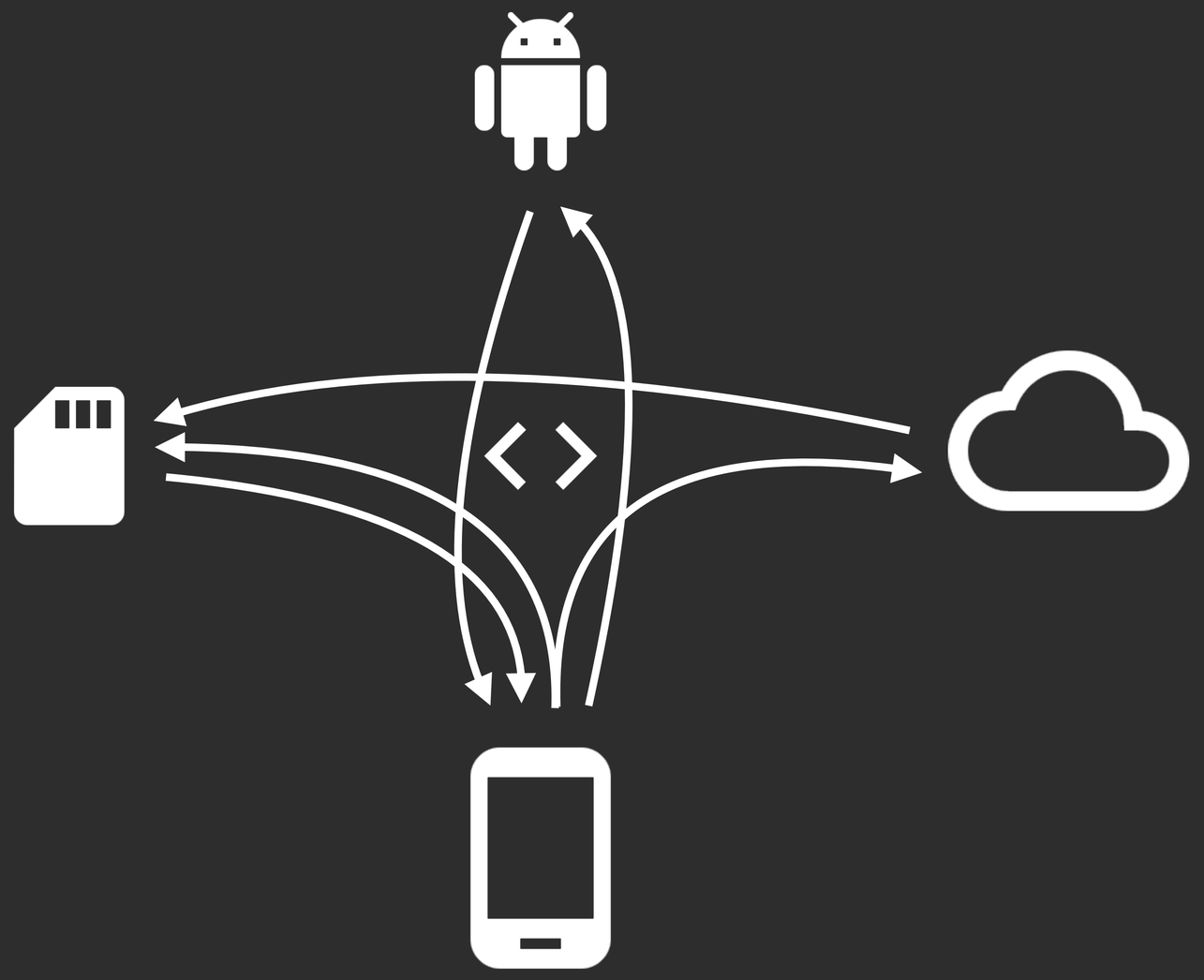
Why Reactive?



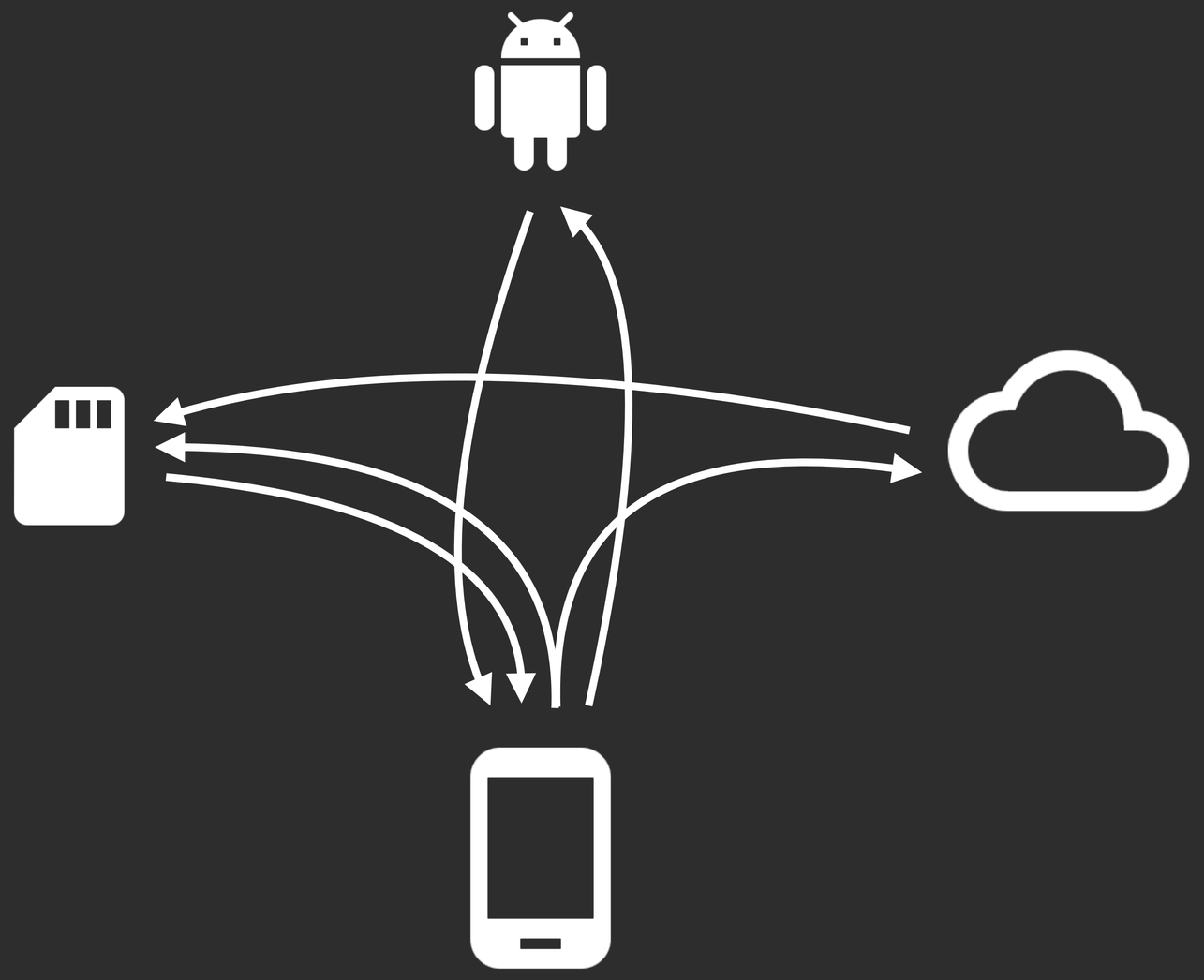
Why Reactive?



Why Reactive?



Why Reactive?



Being Reactive

```
interface UserManager {
  User getUser();
  void setName(String name);
  void setAge(int age);
}
```

Being Reactive

```
interface UserManager {
  Observable<User> getUser();
  void setName(String name);
  void setAge(int age);
}
```

Being Reactive

```
interface UserManager {
  Observable<User> getUser();
  Completable setName(String name);
  Completable setAge(int age);
}
```

Being Reactive

```
um.getUser()
```

Being Reactive

```
um.getUser()
  .observeOn(AndroidSchedulers.mainThread())
```

Being Reactive

```
um.getUser()
  .observeOn(AndroidSchedulers.mainThread())
  .subscribeWith(new DisposableObserver<User>() {
    @Override public void onNext(User user) {
    }
    @Override public void onComplete() { /* ignored */ }
    @Override public void onError(Throwable t) { /* crash or show */ }
  });
```

Being Reactive

```
um.getUser()
  .observeOn(AndroidSchedulers.mainThread())
  .subscribeWith(new DisposableObserver<User>() {
    @Override public void onNext(User user) {
      tv.setText(user.toString());
    }
    @Override public void onComplete() { /* ignored */ }
    @Override public void onError(Throwable t) { /* crash or show */ }
  });
```

Being Reactive

```
disposables.add(um.getUser()
  .observeOn(AndroidSchedulers.mainThread())
  .subscribeWith(new DisposableObserver<User>() {
    @Override public void onNext(User user) {
      tv.setText(user.toString());
    }
    @Override public void onComplete() { /* ignored */ }
    @Override public void onError(Throwable t) { /* crash or show */ }
  }));
```

Being Reactive

```
// onCreate
disposables.add(um.getUser()
  .observeOn(AndroidSchedulers.mainThread())
  .subscribeWith(new DisposableObserver<User>() {
    @Override public void onNext(User user) {
      tv.setText(user.toString());
    }
    @Override public void onComplete() { /* ignored */ }
    @Override public void onError(Throwable t) { /* crash or show */ }
  }));

// onDestroy
disposables.dispose();
```

Being Reactive

```
um.setName("Jane Doe")
```

Being Reactive

```
um.setName("Jane Doe")
  .subscribeOn(Schedulers.io())
```

Being Reactive

```
um.setName("Jane Doe")
  .subscribeOn(Schedulers.io())
  .observeOn(AndroidSchedulers.mainThread())
```

Being Reactive

```
um.setName("Jane Doe")
  .subscribeOn(Schedulers.io())
  .observeOn(AndroidSchedulers.mainThread())
  .subscribeWith(new DisposableCompletableObserver() {
    @Override public void onComplete() {
    }
    @Override public void onError(Throwable t) {
      // retry or show
    }
  });
```

Being Reactive

```
um.setName("Jane Doe")
  .subscribeOn(Schedulers.io())
  .observeOn(AndroidSchedulers.mainThread())
  .subscribeWith(new DisposableCompletableObserver() {
    @Override public void onComplete() {
      // success! re-enable editing
    }
    @Override public void onError(Throwable t) {
      // retry or show
    }
  });
```

Being Reactive

```
disposables.add(um.setName("Jane Doe")
  .subscribeOn(Schedulers.io())
  .observeOn(AndroidSchedulers.mainThread())
  .subscribeWith(new DisposableCompletableObserver() {
    @Override public void onComplete() {
      // success! re-enable editing
    }
    @Override public void onError(Throwable t) {
      // retry or show
    }
  }));
```

Being Reactive

```
// onCreate
disposables.add(um.getUser()
  .observeOn(AndroidSchedulers.mainThread())
  .subscribeWith(new DisposableObserver<User>() {
    @Override public void onNext(User user) {
      tv.setText(user.toString());
    }
    @Override public void onComplete() { /* ignored */ }
    @Override public void onError(Throwable t) { /* crash or show */ }
  }));

// button click listener
disposables.add(um.setName("Jane Doe")
  .subscribeOn(Schedulers.io())
  .observeOn(AndroidSchedulers.mainThread())
  .subscribeWith(new DisposableCompletableObserver() {
    @Override public void onComplete() {
      // success! re-enable editing
    }
    @Override public void onError(Throwable t) {
      // retry or show
    }
  }));

// onDestroy
disposables.dispose();
```

Being Reactive

```
// onCreate
disposables.add(um.getUser()
  .observeOn(AndroidSchedulers.mainThread())
  .subscribeWith(new DisposableObserver<User>() {
    @Override public void onNext(User user) {
      tv.setText(user.toString());
    }
    @Override public void onComplete() { /* ignored */ }
    @Override public void onError(Throwable t) { /* crash or show */ }
  }));

// button click listener
disposables.add(um.setName("Jane Doe")
  .subscribeOn(Schedulers.io())
  .observeOn(AndroidSchedulers.mainThread())
  .subscribeWith(new DisposableCompletableObserver() {
    @Override public void onComplete() {
      // success! re-enable editing
    }
    @Override public void onError(Throwable t) {
      // retry or show
    }
  }));

// onDestroy
disposables.dispose();
```

Push-based updates

Being Reactive

```
// onCreate
disposables.add(um.getUser()
  .observeOn(AndroidSchedulers.mainThread())
  .subscribeWith(new DisposableObserver<User>() {
    @Override public void onNext(User user) {
      tv.setText(user.toString());
    }
    @Override public void onComplete() { /* ignored */ }
    @Override public void onError(Throwable t) { /* crash or show */ }
  }));

// button click listener
disposables.add(um.setName("Jane Doe")
  .subscribeOn(Schedulers.io())
  .observeOn(AndroidSchedulers.mainThread())
  .subscribeWith(new DisposableCompletableObserver() {
    @Override public void onComplete() {
      // success! re-enable editing
    }
    @Override public void onError(Throwable t) {
      // retry or show
    }
  }));

// onDestroy
disposables.dispose();
```

Push-based updates

Declarative threading

Being Reactive

```
// onCreate
disposables.add(um.getUser()
  .observeOn(AndroidSchedulers.mainThread())
  .subscribeWith(new DisposableObserver<User>() {
    @Override public void onNext(User user) {
      tv.setText(user.toString());
    }
    @Override public void onComplete() { /* ignored */ }
    @Override public void onError(Throwable t) { /* crash or show */ }
  }));

// button click listener
disposables.add(um.setName("Jane Doe")
  .subscribeOn(Schedulers.io())
  .observeOn(AndroidSchedulers.mainThread())
  .subscribeWith(new DisposableCompletableObserver() {
    @Override public void onComplete() {
      // success! re-enable editing
    }
    @Override public void onError(Throwable t) {
      // retry or show
    }
  }));

// onDestroy
disposables.dispose();
```

Push-based updates

Declarative threading

Easy error-handling

Being Reactive

```
// onCreate
disposables.add(um.getUser()
  .observeOn(AndroidSchedulers.mainThread())
  .subscribeWith(new DisposableObserver<User>() {
    @Override public void onNext(User user) {
      tv.setText(user.toString());
    }
    @Override public void onComplete() { /* ignored */ }
    @Override public void onError(Throwable t) { /* crash or show */ }
  }));

// button click listener
disposables.add(um.setName("Jane Doe")
  .subscribeOn(Schedulers.io())
  .observeOn(AndroidSchedulers.mainThread())
  .subscribeWith(new DisposableCompletableObserver() {
    @Override public void onComplete() {
      // success! re-enable editing
    }
    @Override public void onError(Throwable t) {
      // retry or show
    }
  }));

// onDestroy
disposables.dispose();
```

Push-based updates

Declarative threading

Easy error-handling

Specialized callbacks

Being Reactive

```
// onCreate
disposables.add(um.getUser()
  .observeOn(AndroidSchedulers.mainThread())
  .subscribeWith(new DisposableObserver<User>() {
    @Override public void onNext(User user) {
      tv.setText(user.toString());
    }
    @Override public void onComplete() { /* ignored */ }
    @Override public void onError(Throwable t) { /* crash or show */ }
  }));

// button click listener
disposables.add(um.setName("Jane Doe")
  .subscribeOn(Schedulers.io())
  .observeOn(AndroidSchedulers.mainThread())
  .subscribeWith(new DisposableCompletableObserver() {
    @Override public void onComplete() {
      // success! re-enable editing
    }
    @Override public void onError(Throwable t) {
      // retry or show
    }
  }));

// onDestroy
disposables.dispose();
```

Push-based updates

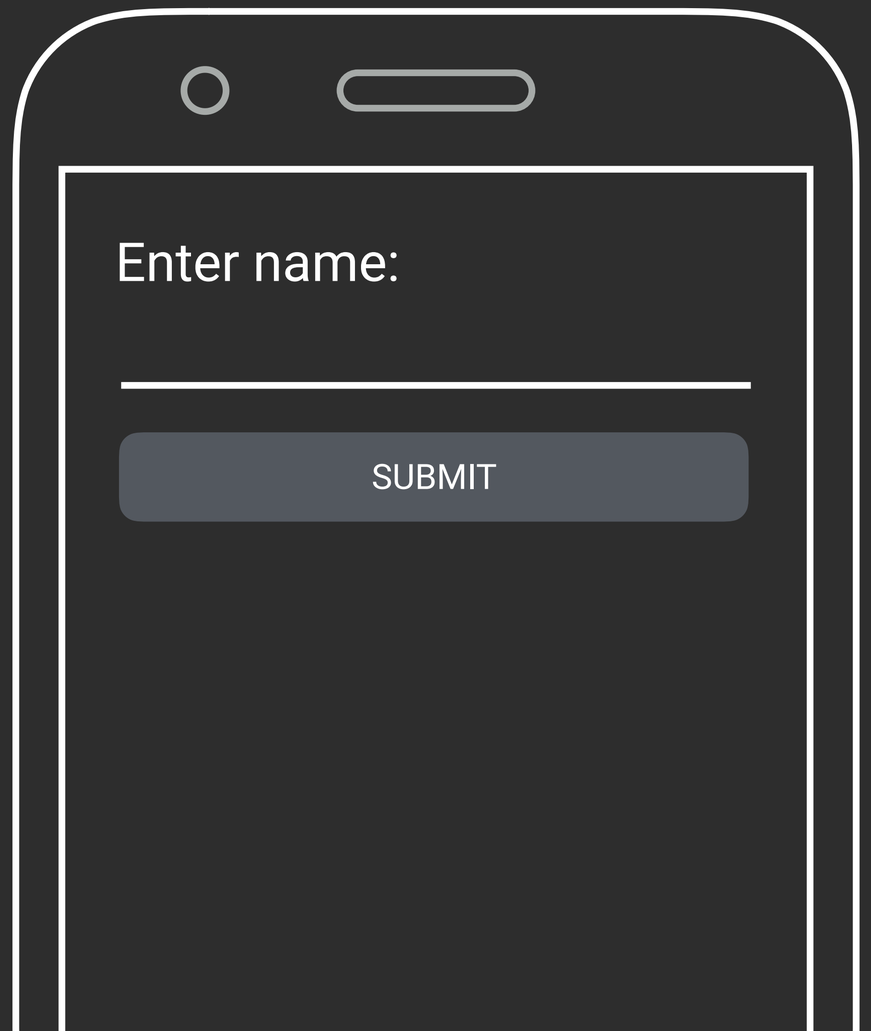
Declarative threading

Easy error-handling

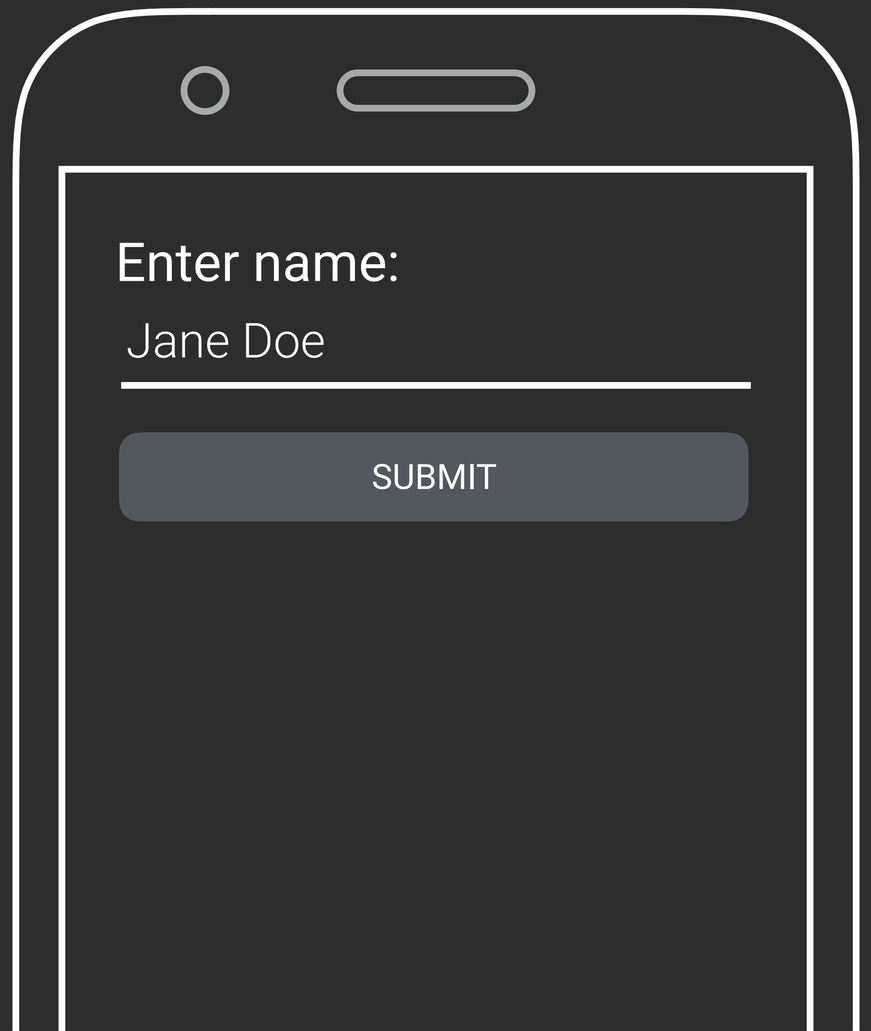
Specialized callbacks

Lifecycle cancelation

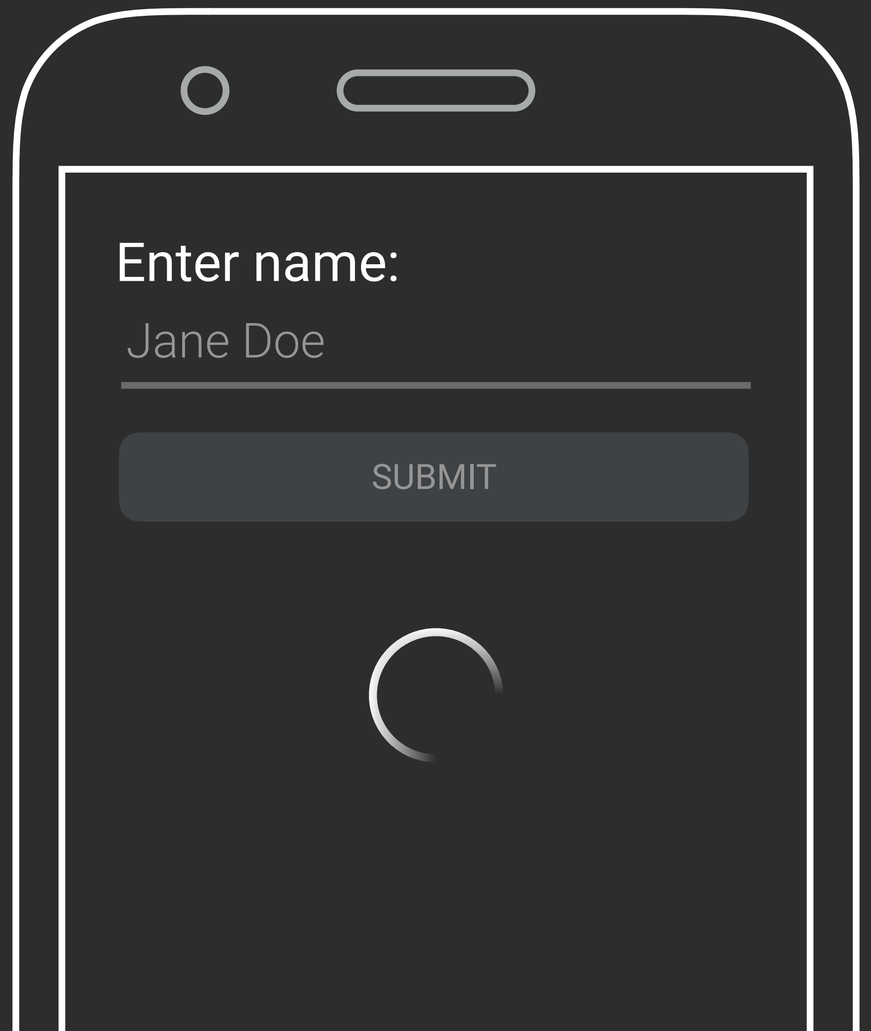
Managing State



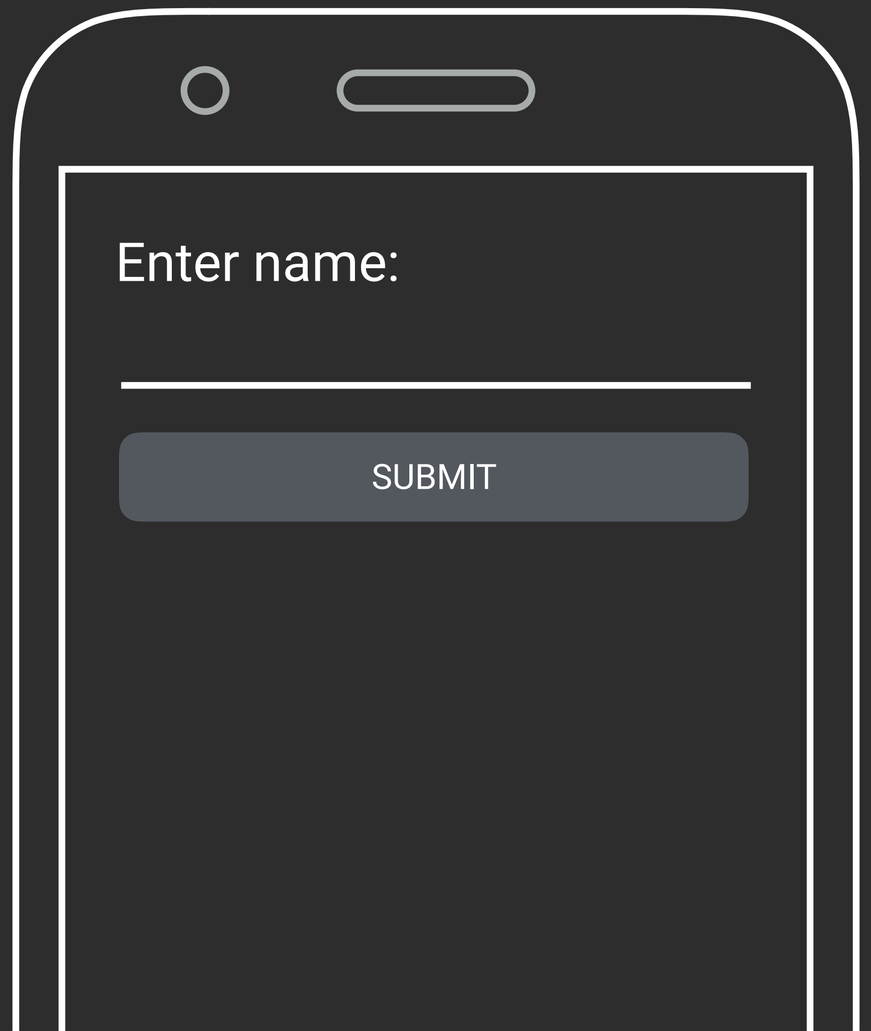
Managing State



Managing State



Managing State



Managing State

```
disposables.add(RxView.clicks(submitView)
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(ignored -> service.setName(nameView.getText().toString()))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Managing State

```
disposables.add(RxView.clicks(submitView)
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(ignored -> service.setName(nameView.getText().toString()))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Managing State

```
disposables.add(RxView.clicks(submitView)
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(ignored -> service.setName(nameView.getText().toString()))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Managing State

```
disposables.add(RxView.clicks(submitView)
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(ignored -> service.setName(nameView.getText().toString()))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Managing State

```
disposables.add(RxView.clicks(submitView)
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(ignored -> service.setName(nameView.getText().toString()))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Managing State

```
disposables.add(RxView.clicks(submitView)
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(ignored -> service.setName(nameView.getText().toString()))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Managing State

```
disposables.add(RxView.clicks(submitView)
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(ignored -> service.setName(nameView.getText().toString()))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Managing State

```
disposables.add(RxView.clicks(submitView)
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(ignored -> service.setName(nameView.getText().toString()))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Managing State

```
disposables.add(RxView.clicks(submitView)
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(ignored -> service.setName(nameView.getText().toString()))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Managing State

```
disposables.add(RxView.clicks(submitView)
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(ignored -> service.setName(nameView.getText().toString()))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Managing State

```
disposables.add(RxView.clicks(submitView)
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(ignored -> service.setName(nameView.getText().toString()))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Managing State

```
disposables.add(RxView.clicks(submitView)
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(ignored -> service.setName(nameView.getText().toString()))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Managing State

```
disposables.add(RxView.clicks(submitView)
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(ignored -> service.setName(nameView.getText().toString()))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Managing State

```
disposables.add(RxView.clicks(submitView)
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(ignored -> service.setName(nameView.getText().toString()))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Managing State

```
disposables.add(RxView.clicks(submitView)
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(ignored -> service.setName(nameView.getText().toString()))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Managing State

```
disposables.add(RxView.clicks(submitView)
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(ignored -> service.setName(nameView.getText().toString()))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Managing State

```
disposables.add(RxView.clicks(submitView)
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(ignored -> service.setName(nameView.getText().toString()))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Managing State

```
disposables.add(RxView.clicks(submitView)
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(ignored -> service.setName(nameView.getText().toString()))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Managing State

```
disposables.add(RxView.clicks(submitView)
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(ignored -> service.setName(nameView.getText().toString()))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Managing State

```
disposables.add(RxView.clicks(submitView)
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(ignored -> service.setName(nameView.getText().toString()))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Managing State

```
disposables.add(RxView.clicks(submitView)
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(ignored -> service.setName(nameView.getText().toString()))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Managing State

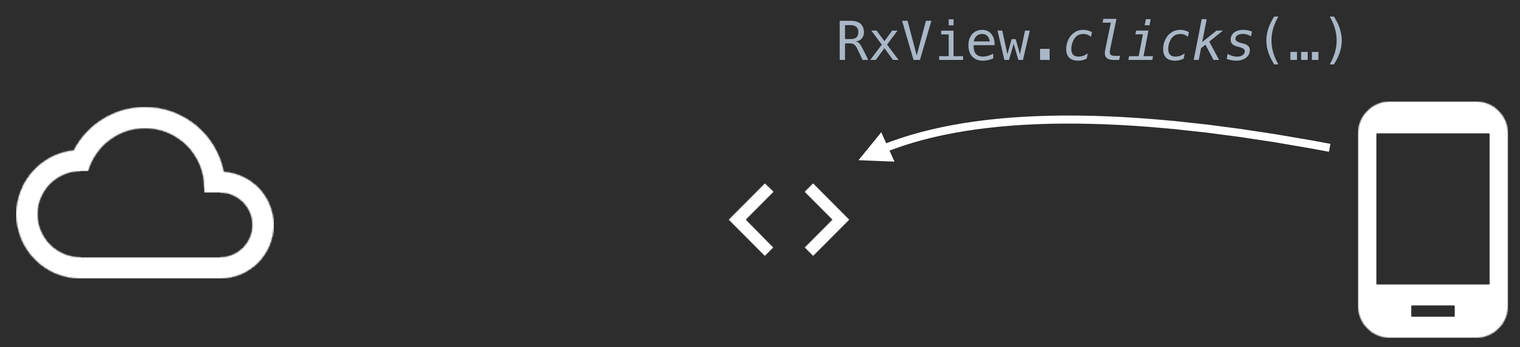
```
disposables.add(RxView.clicks(submitView)
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(ignored -> service.setName(nameView.getText().toString()))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Managing State

```
disposables.add(RxView.clicks(submitView)
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(ignored -> service.setName(nameView.getText().toString()))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

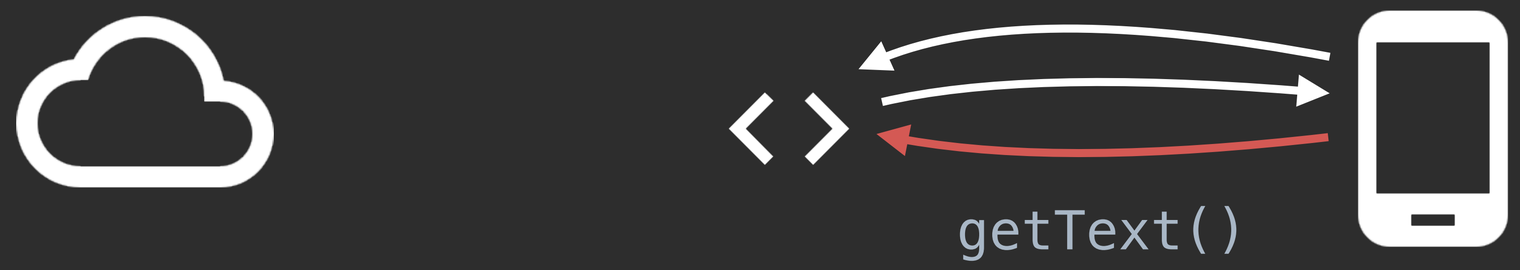
Managing State

Managing State



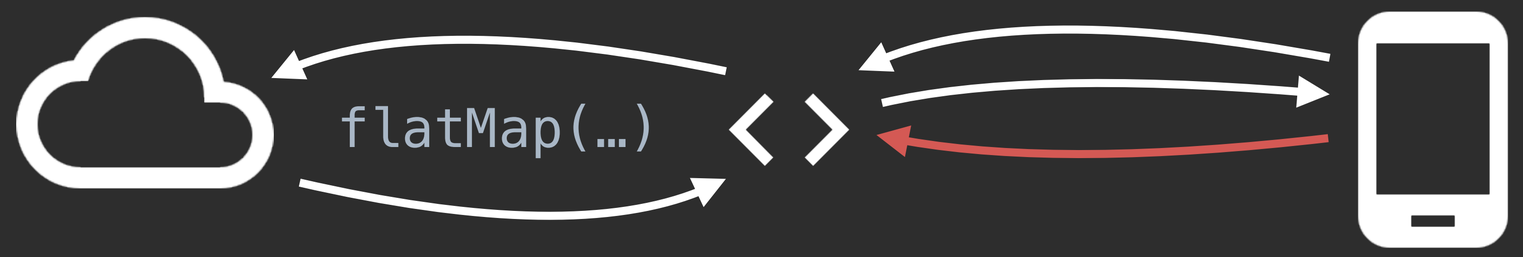
Managing State

Managing State

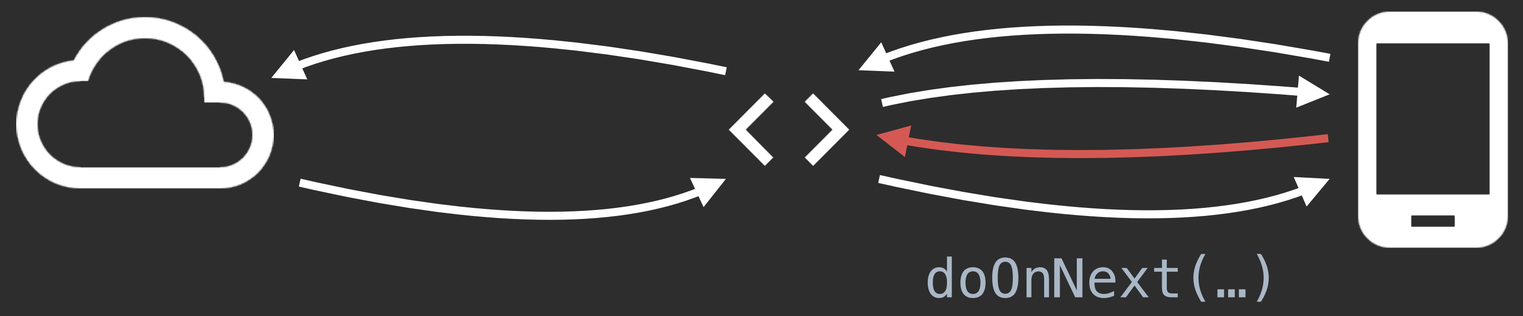


Managing State

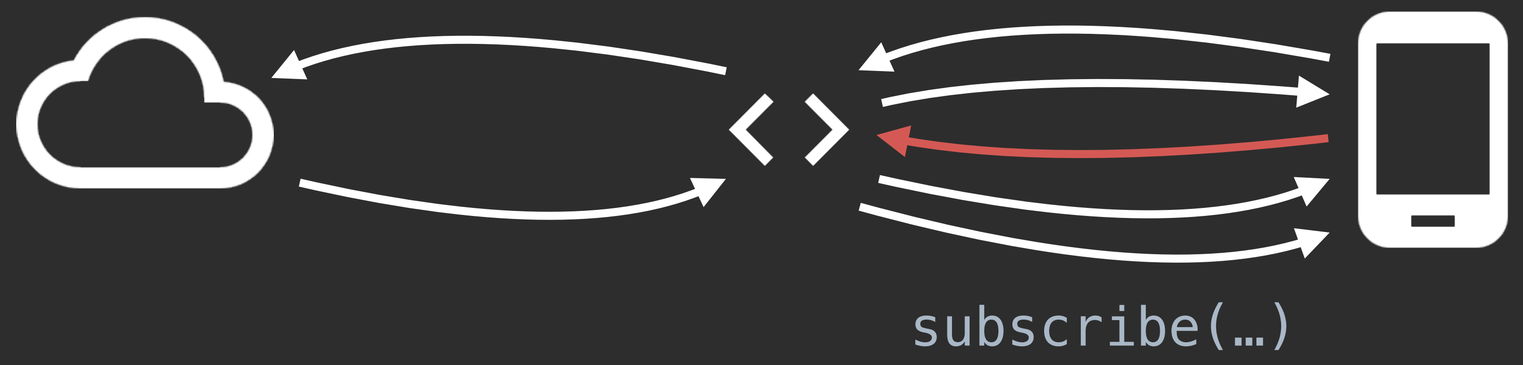
Managing State



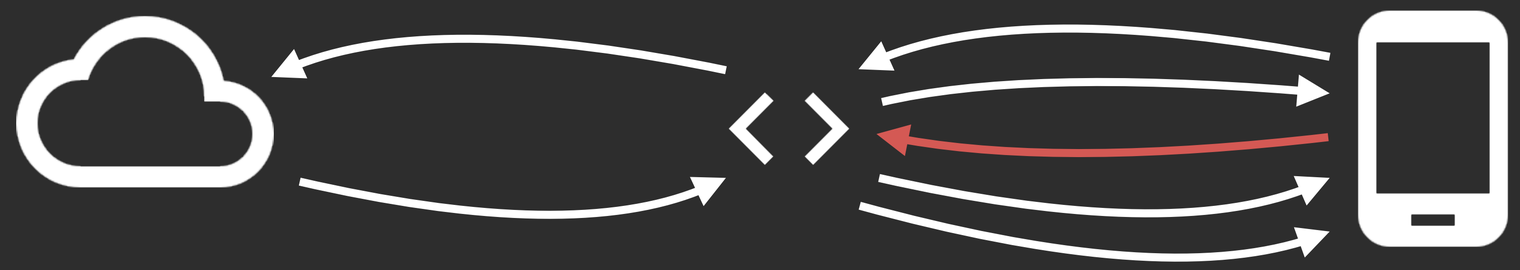
Managing State



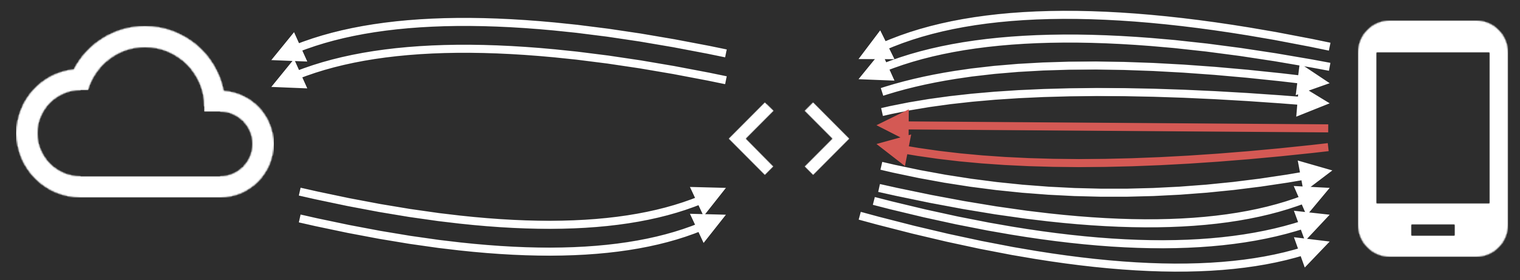
Managing State



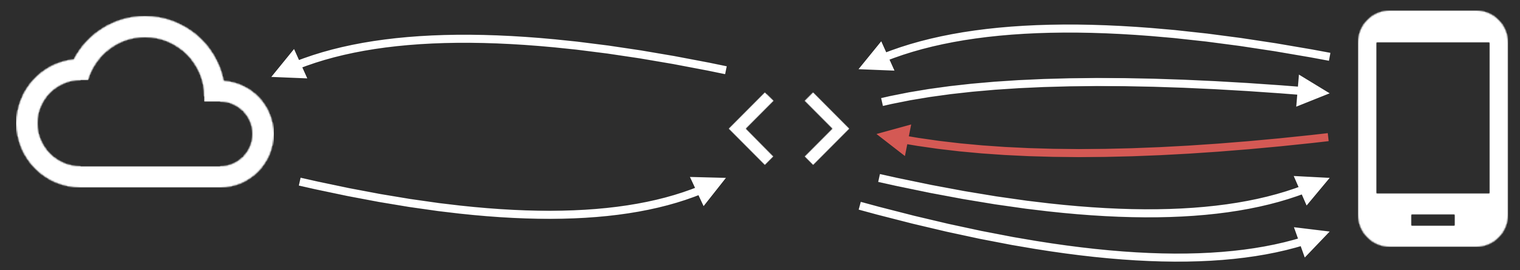
Managing State



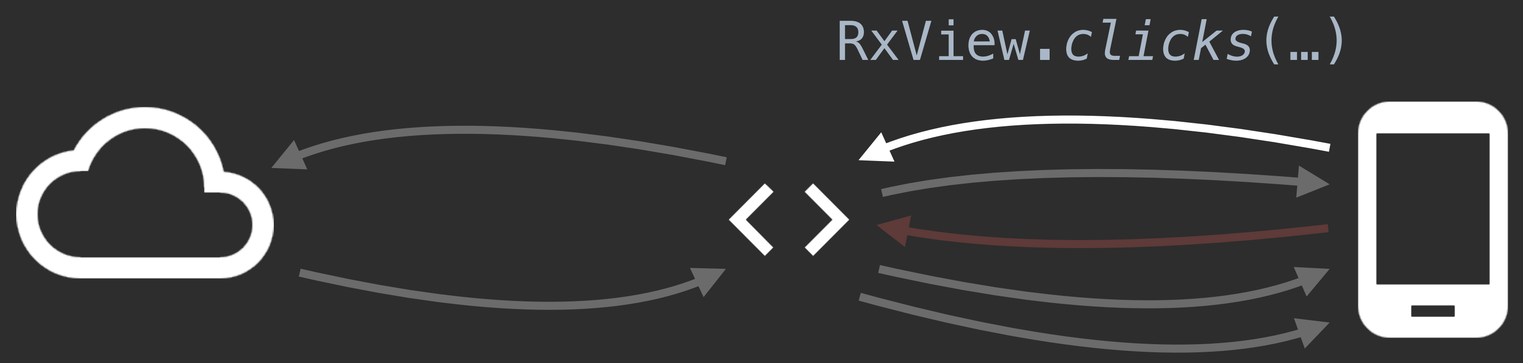
Managing State



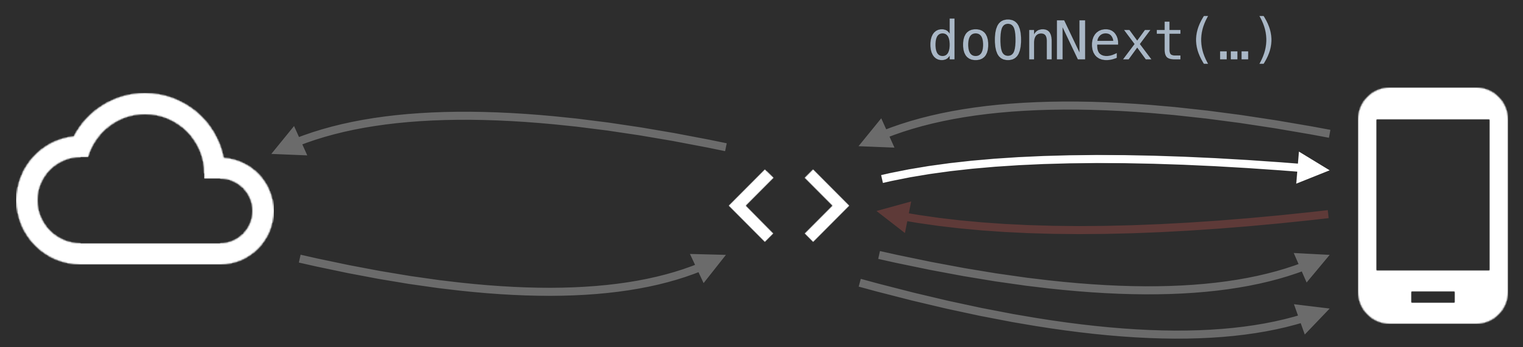
Managing State



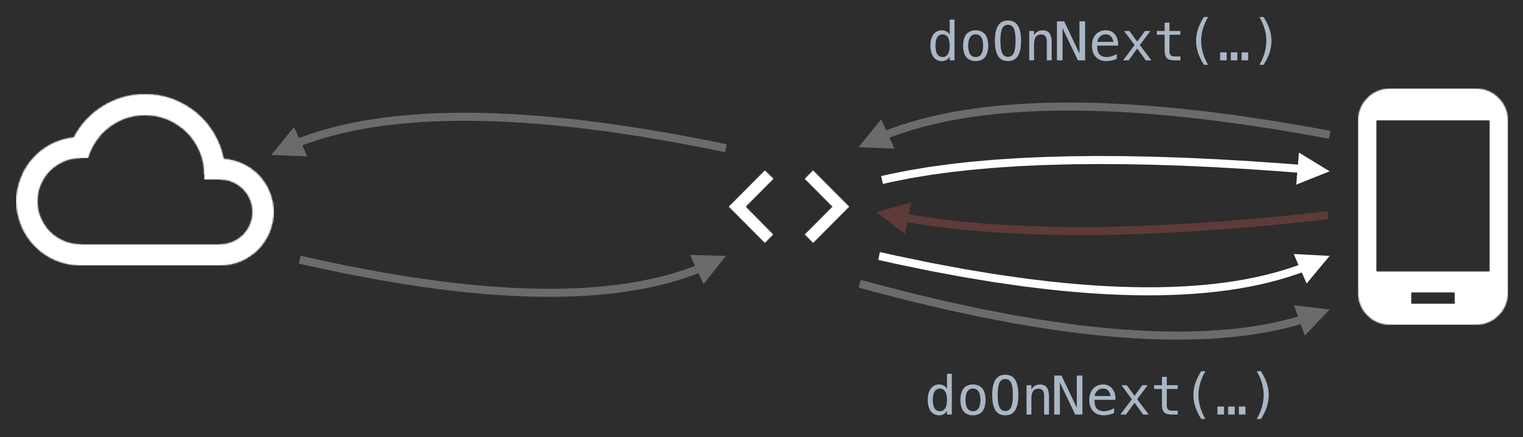
Managing State



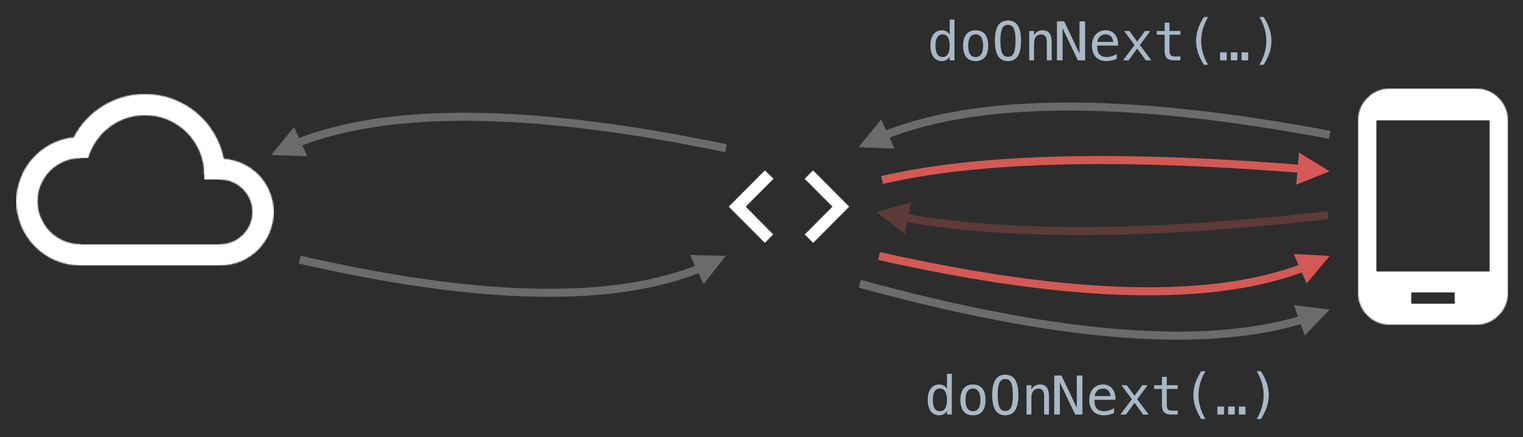
Managing State



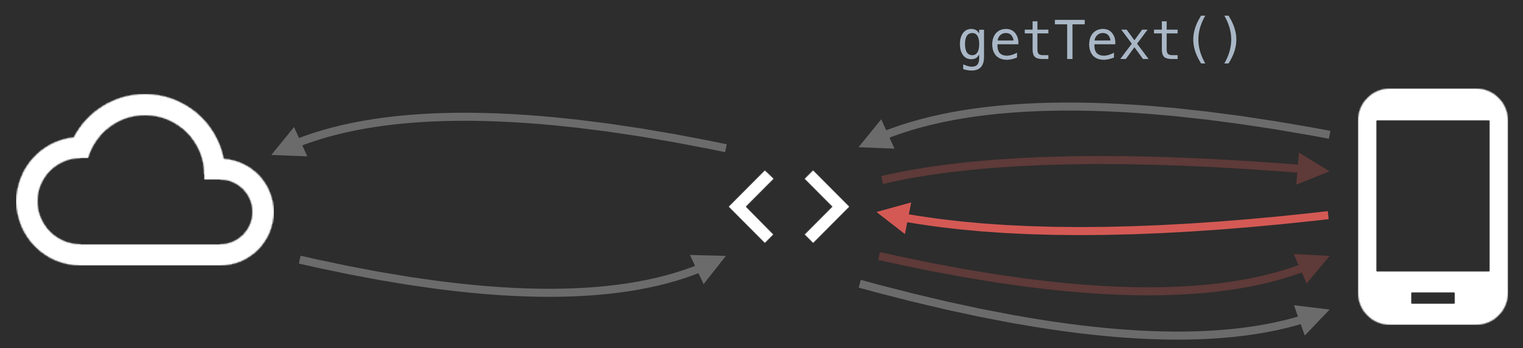
Managing State



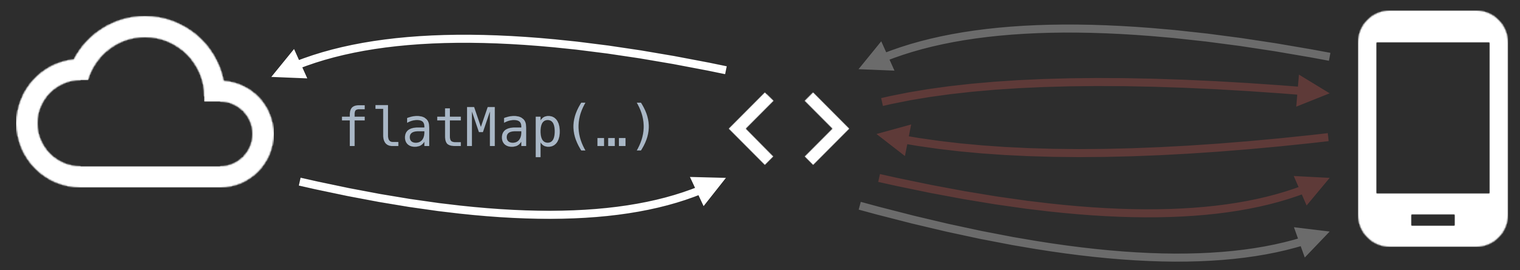
Managing State



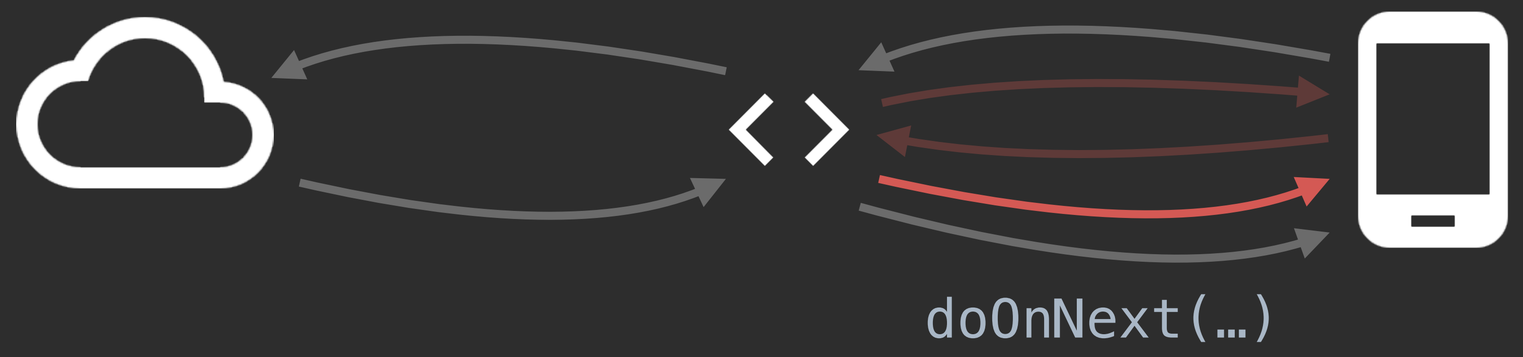
Managing State



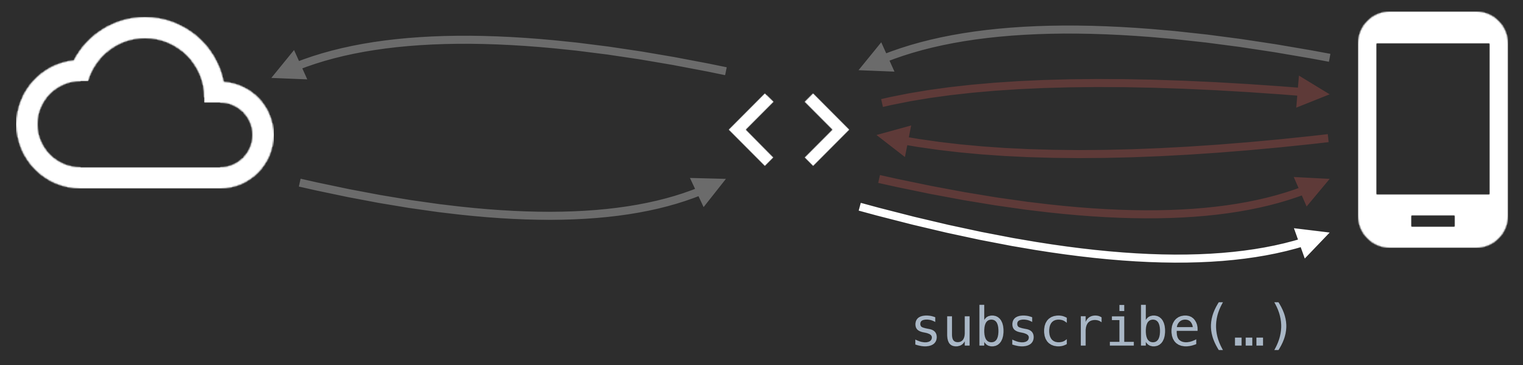
Managing State



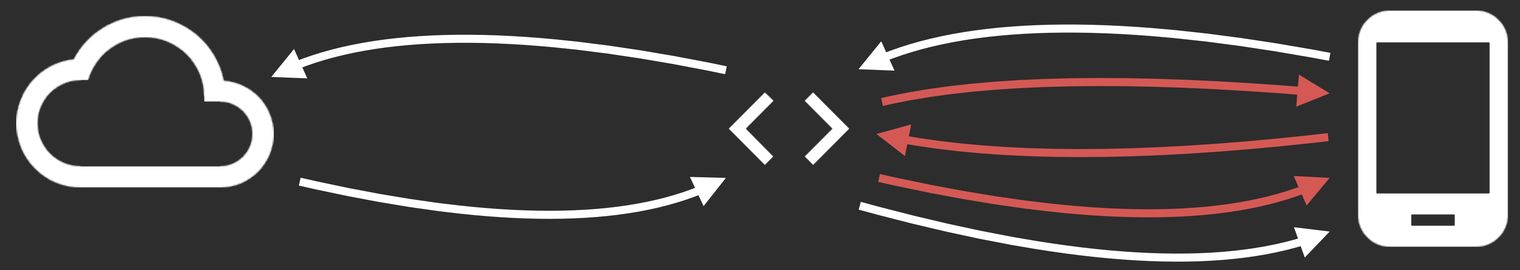
Managing State



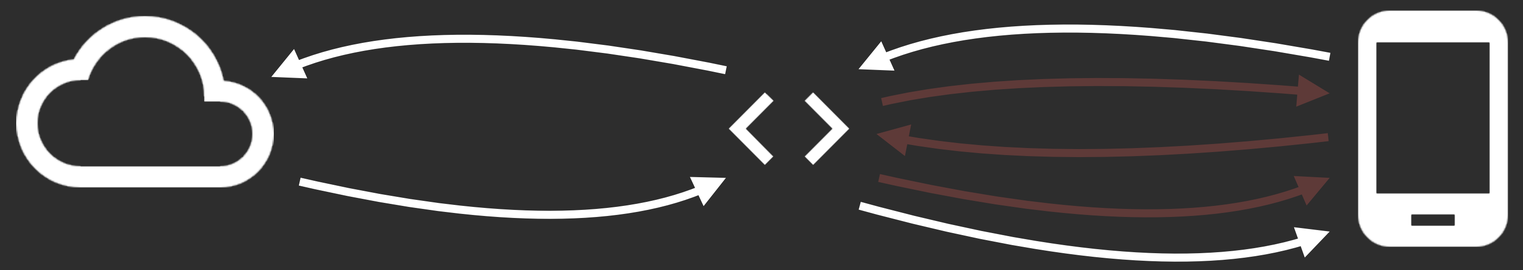
Managing State



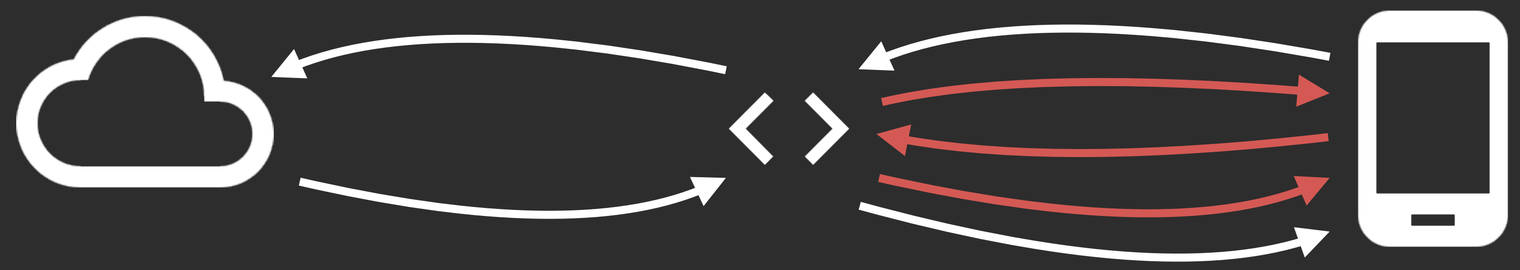
Managing State



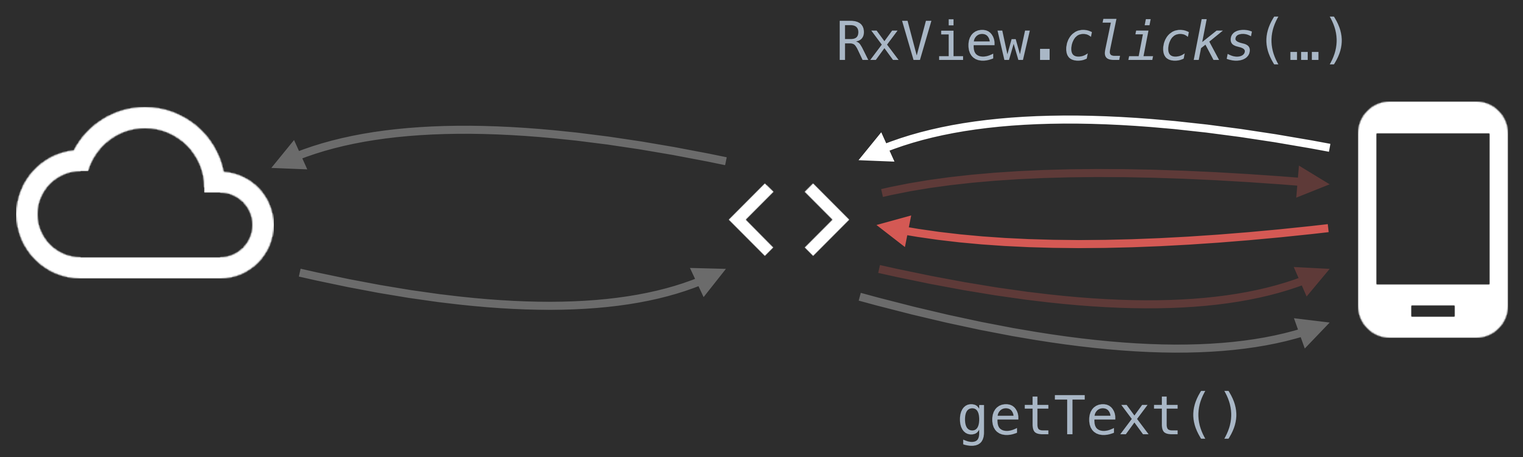
Managing State



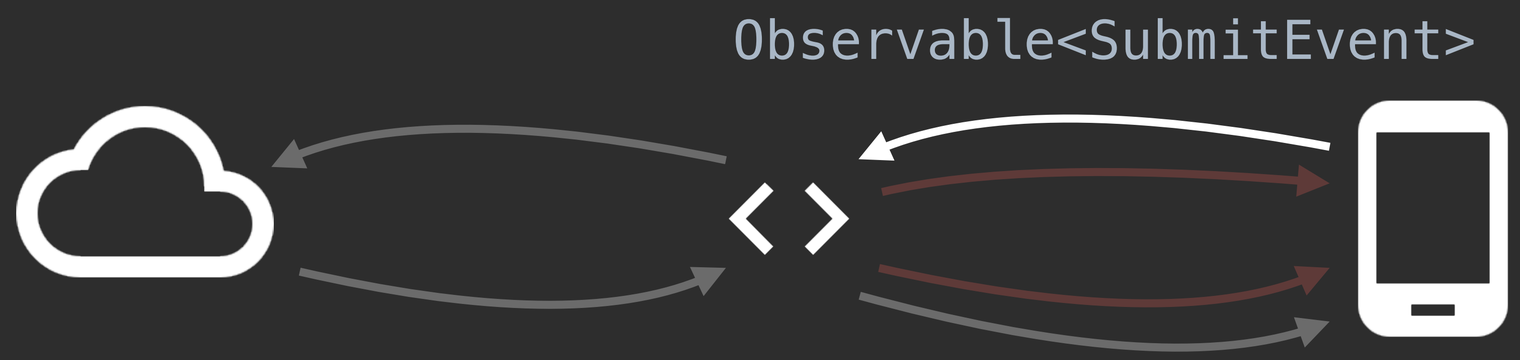
Managing State



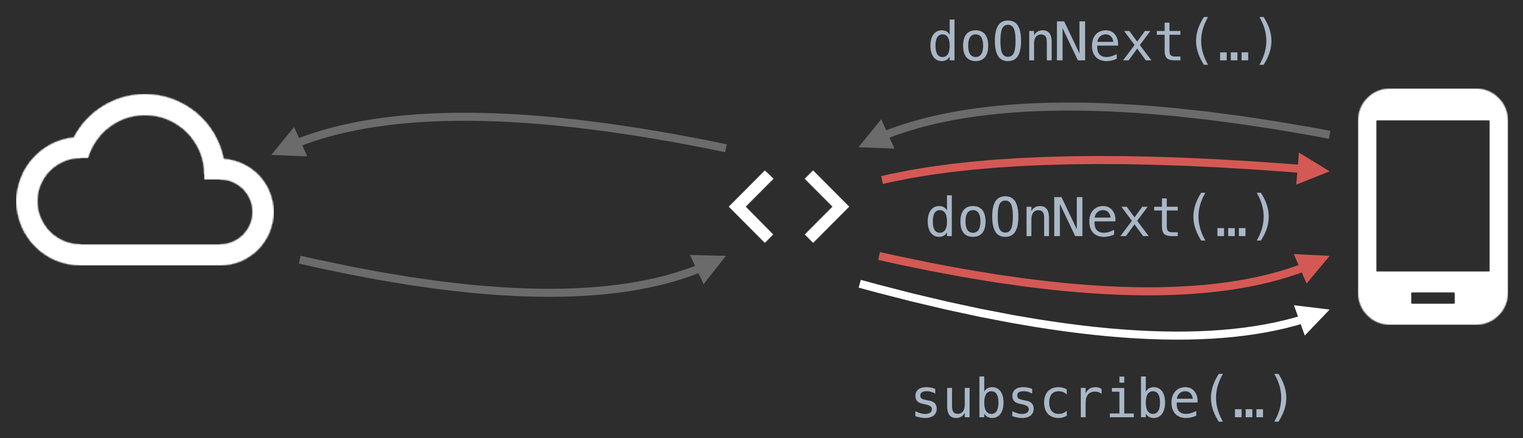
Managing State



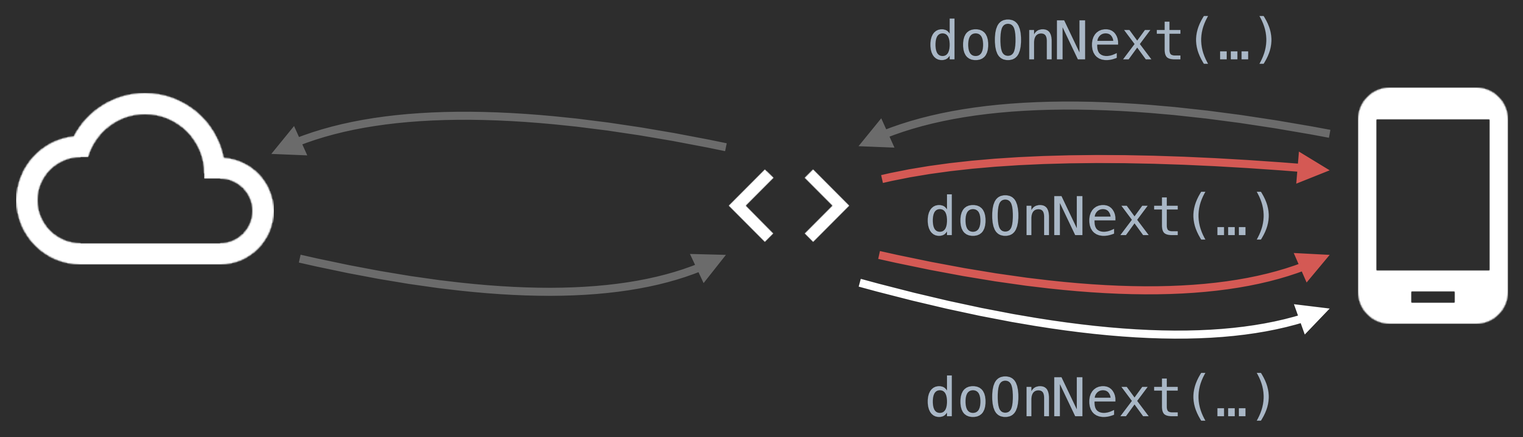
Managing State



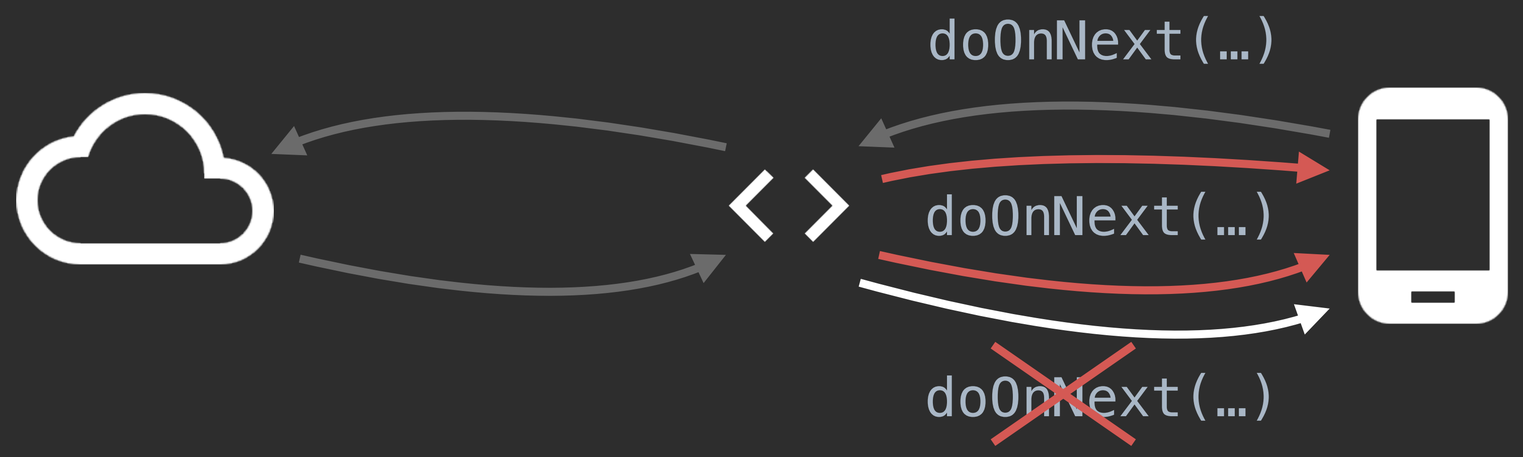
Managing State



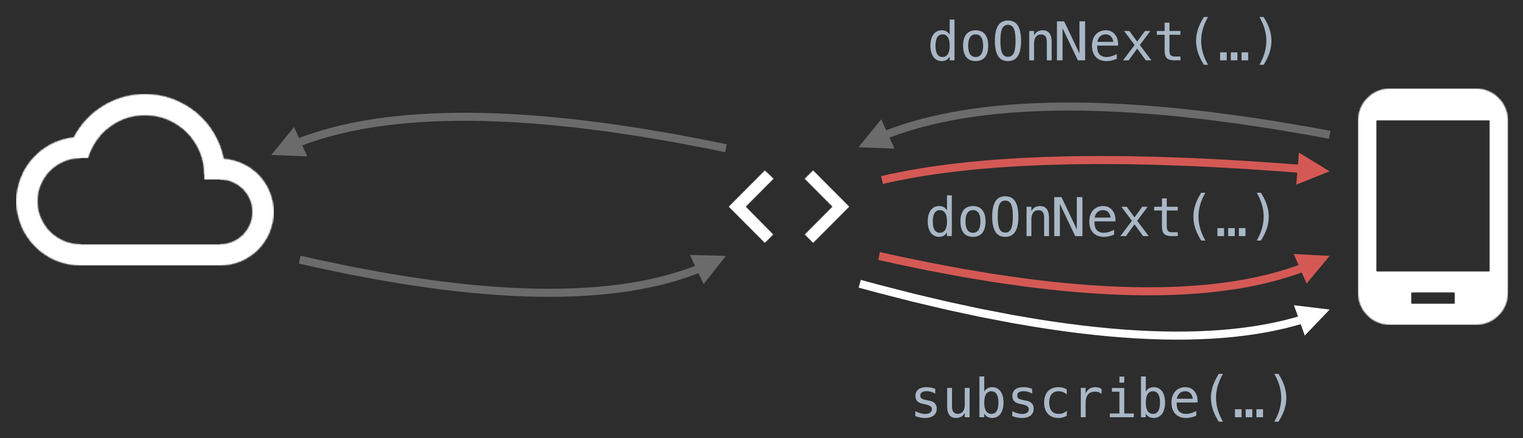
Managing State



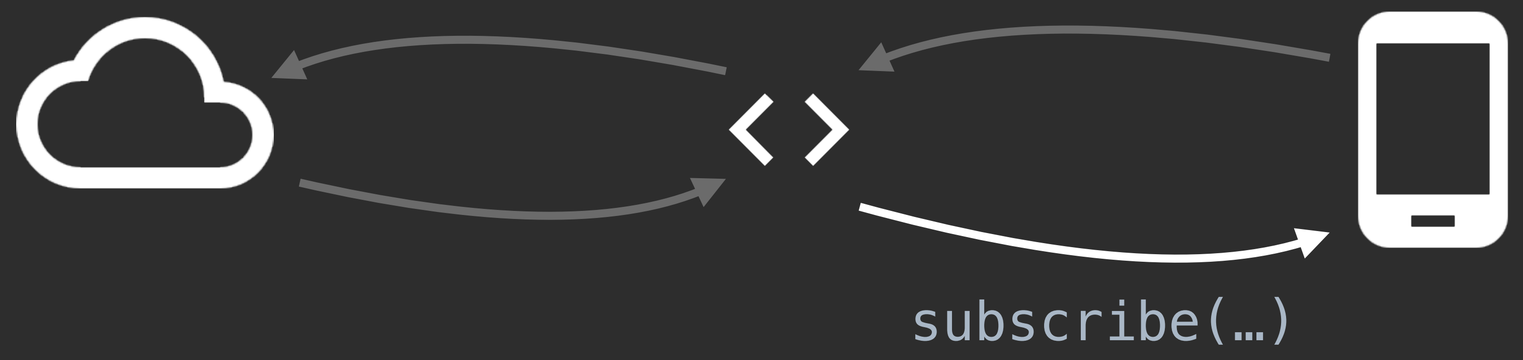
Managing State



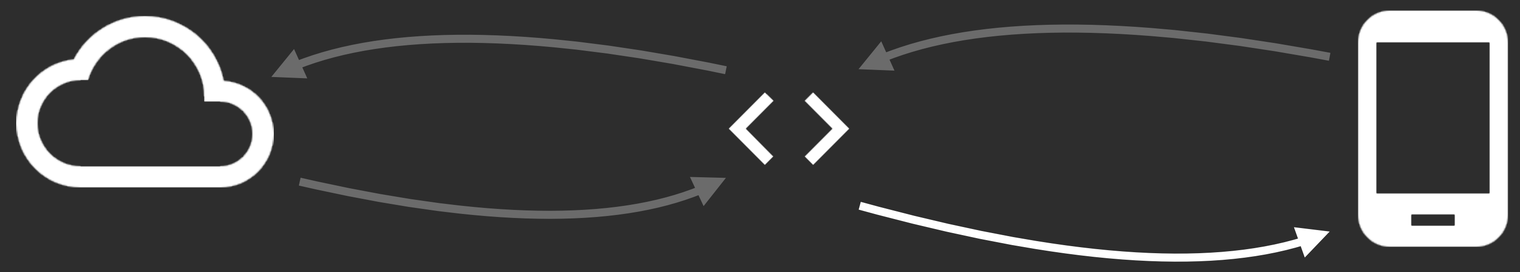
Managing State



Managing State

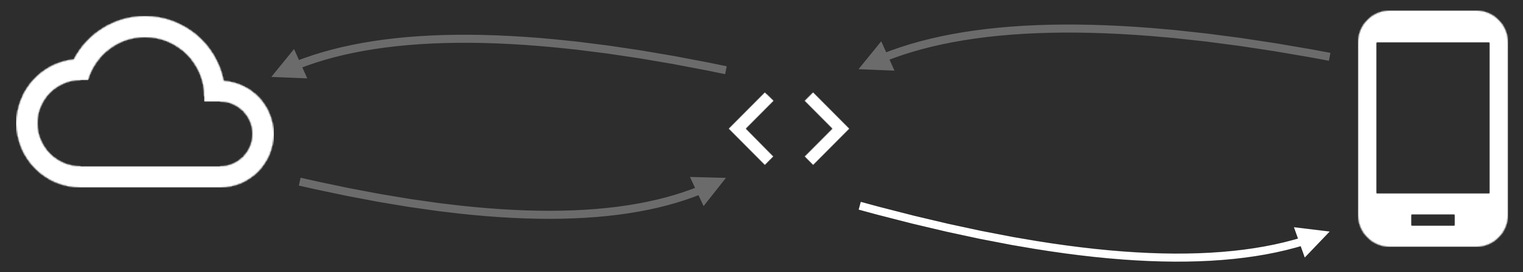


Managing State



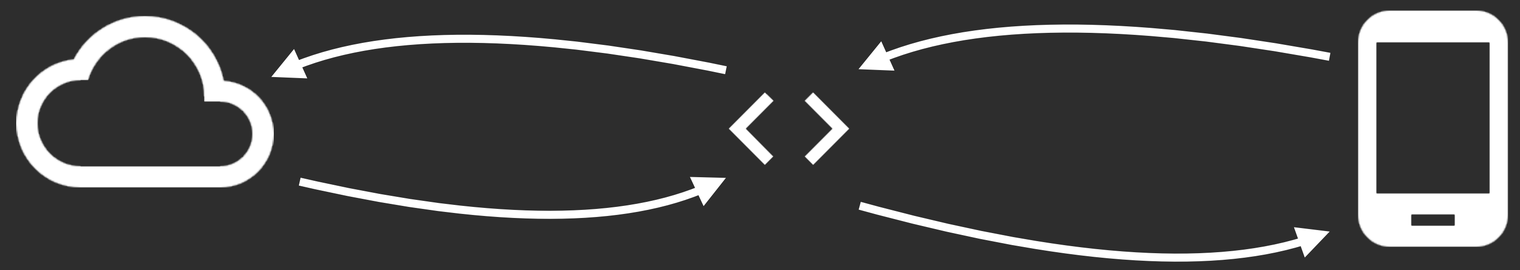
```
subscribe(ignored -> …)
```

Managing State

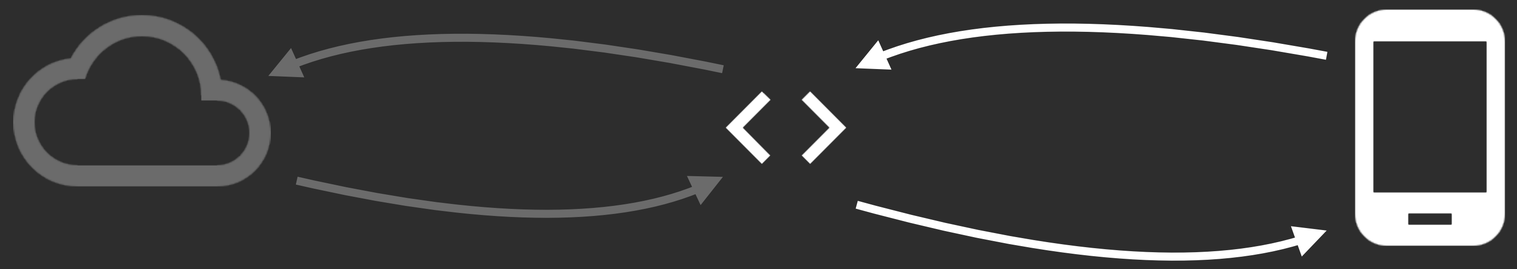


```
subscribe((SubmitUiModel model) -> …)
```

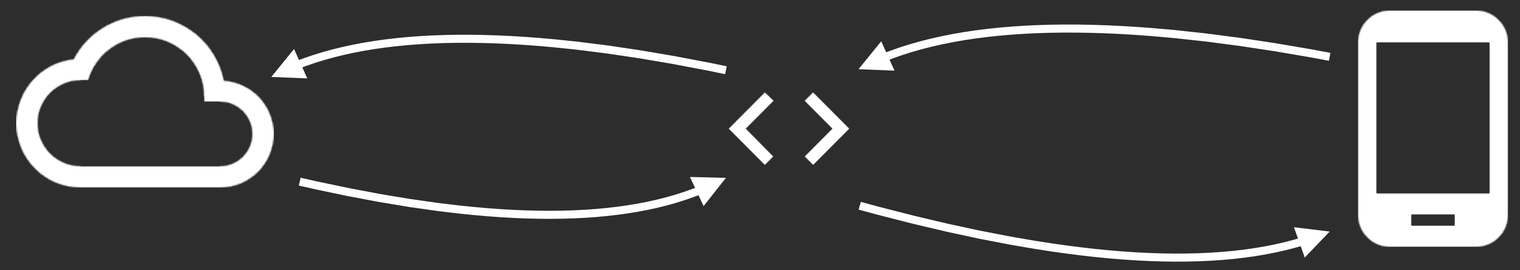
Managing State



Managing State



Reactive State



Reactive State

```
disposables.add(RxView.clicks(submitView)
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(ignored -> service.setName(nameView.getText().toString()))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Reactive State

```
disposables.add(RxView.clicks(submitView)
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(ignored -> service.setName(nameView.getText().toString()))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Reactive State

```
disposables.add(RxView.clicks(submitView)
    .map(ignored -> nameView.getText().toString())
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(name -> service.setName(name))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Reactive State

```
disposables.add(RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()))
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(event -> service.setName(event.name))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Reactive State

```
disposables.add(RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()))
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(event -> service.setName(event.name))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Reactive State

```
disposables.add(RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()))
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(event -> service.setName(event.name))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Reactive State

```
disposables.add(RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()))
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(event -> service.setName(event.name))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Reactive State

```
final class SubmitUiModel {
  final boolean inProgress;
  final boolean success;
  final String errorMessage;

  private SubmitUiModel(
      boolean inProgress, boolean success, String errorMessage) {
    // ...
  }

  static SubmitUiModel inProgress() { /* ... */ }
  static SubmitUiModel success() { /* ... */ }
  static SubmitUiModel failure(String errorMessage) { /* ... */ }
}
```

Reactive State

```
final class SubmitUiModel {
  final boolean inProgress;
  final boolean success;
  final String errorMessage;

  private SubmitUiModel(
      boolean inProgress, boolean success, String errorMessage) {
    // ...
  }

  static SubmitUiModel inProgress() { /* ... */ }
  static SubmitUiModel success() { /* ... */ }
  static SubmitUiModel failure(String errorMessage) { /* ... */ }
}
```

Reactive State

```
disposables.add(RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()))
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(event -> service.setName(event.name))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Reactive State

```
disposables.add(RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()))
    .doOnNext(ignored -> {
      submitView.setEnabled(false);
      progressView.setVisibility(VISIBLE);
    })
    .flatMap(event -> service.setName(event.name))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Reactive State

```
disposables.add(RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()))
    .doOnNext(ignored -> {
      SubmitUiModel.inProgress()
    })
    .flatMap(event -> service.setName(event.name))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Reactive State

```
disposables.add(RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()))
    .flatMap(event -> service.setName(event.name)
        .startWith(SubmitUiModel.inProgress()))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Reactive State

```
disposables.add(RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()))
    .flatMap(event -> service.setName(event.name)
        .startWith(SubmitUiModel.inProgress()))
    .observeOn(AndroidSchedulers.mainThread())
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Reactive State

```
disposables.add(RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()))
    .flatMap(event -> service.setName(event.name)
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitUiModel.inProgress()))
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Reactive State

```
disposables.add(RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()))
    .flatMap(event -> service.setName(event.name)
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitUiModel.inProgress()))
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Reactive State

```
final class SubmitUiModel {
  final boolean inProgress;
  final boolean success;
  final String errorMessage;

  private SubmitUiModel(
      boolean inProgress, boolean success, String errorMessage) {
    // ...
  }

  static SubmitUiModel inProgress() { /* ... */ }
  static SubmitUiModel success() { /* ... */ }
  static SubmitUiModel failure(String errorMessage) { /* ... */ }
}
```

Reactive State

```
final class SubmitUiModel {
  final boolean inProgress;
  final boolean success;
  final String errorMessage;

  private SubmitUiModel(
      boolean inProgress, boolean success, String errorMessage) {
    // ...
  }

  static SubmitUiModel inProgress() { /* ... */ }
  static SubmitUiModel success() { /* ... */ }
  static SubmitUiModel failure(String errorMessage) { /* ... */ }
}
```

Reactive State

```
disposables.add(RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()))
    .flatMap(event -> service.setName(event.name)
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitUiModel.inProgress()))
    .doOnNext(ignored -> progressView.setVisibility(GONE))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Reactive State

```
disposables.add(RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()))
    .flatMap(event -> service.setName(event.name)
        .map(response -> SubmitUiModel.success())
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitUiModel.inProgress()))
    .subscribe(s -> finish(), t -> {
      submitView.setEnabled(true);
      Toast.makeText(this, "Failed to set name: " + t.getMessage(),
          LENGTH_SHORT).show();
    }));
```

Reactive State

```
disposables.add(RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()))
    .flatMap(event -> service.setName(event.name)
        .map(response -> SubmitUiModel.success())
        .onErrorReturn(t -> SubmitUiModel.failure(t.getMessage()))
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitUiModel.inProgress()))
    .subscribe(s -> finish(), t -> {}));
```

Reactive State

```
disposables.add(RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()))
    .flatMap(event -> service.setName(event.name)
        .map(response -> SubmitUiModel.success())
        .onErrorReturn(t -> SubmitUiModel.failure(t.getMessage()))
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitUiModel.inProgress()))
    .subscribe(s -> finish(), t -> {}));
```

Reactive State

```
disposables.add(RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()))
    .flatMap(event -> service.setName(event.name)
        .map(response -> SubmitUiModel.success())
        .onErrorReturn(t -> SubmitUiModel.failure(t.getMessage()))
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitUiModel.inProgress()))
    .subscribe(s -> finish(), t -> {}));
```

Reactive State

```
disposables.add(RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()))
    .flatMap(event -> service.setName(event.name)
        .map(response -> SubmitUiModel.success())
        .onErrorReturn(t -> SubmitUiModel.failure(t.getMessage()))
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitUiModel.inProgress()))
    .subscribe(model -> {
      submitView.setEnabled(!model.inProgress);
      progressView.setVisibility(model.inProgress ? VISIBLE : GONE);
      if (!model.inProgress) {
        if (model.success) finish()
        else Toast.makeText(this, "Failed to set name: "
          + model.errorMessage, LENGTH_SHORT).show();
      }
    }, t -> {}));
```

Reactive State

```
disposables.add(RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()))
    .flatMap(event -> service.setName(event.name)
        .map(response -> SubmitUiModel.success())
        .onErrorReturn(t -> SubmitUiModel.failure(t.getMessage()))
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitUiModel.inProgress()))
    .subscribe(model -> {
      submitView.setEnabled(!model.inProgress);
      progressView.setVisibility(model.inProgress ? VISIBLE : GONE);
      if (!model.inProgress) {
        if (model.success) finish()
        else Toast.makeText(this, "Failed to set name: "
          + model.errorMessage, LENGTH_SHORT).show();
      }
    }, t -> {}));
```

Reactive State

```
disposables.add(RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()))
    .flatMap(event -> service.setName(event.name)
        .map(response -> SubmitUiModel.success())
        .onErrorReturn(t -> SubmitUiModel.failure(t.getMessage()))
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitUiModel.inProgress()))
    .subscribe(model -> {
      submitView.setEnabled(!model.inProgress);
      progressView.setVisibility(model.inProgress ? VISIBLE : GONE);
      if (!model.inProgress) {
        if (model.success) finish()
        else Toast.makeText(this, "Failed to set name: "
          + model.errorMessage, LENGTH_SHORT).show();
      }
    }, t -> { throw new OnErrorNotImplementedException(t); }));
```

Reactive State

```
disposables.add(RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()))
    .flatMap(event -> service.setName(event.name)
        .map(response -> SubmitUiModel.success())
        .onErrorReturn(t -> SubmitUiModel.failure(t.getMessage()))
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitUiModel.inProgress()))
    .subscribe(model -> {
      submitView.setEnabled(!model.inProgress);
      progressView.setVisibility(model.inProgress ? VISIBLE : GONE);
      if (!model.inProgress) {
        if (model.success) finish()
        else Toast.makeText(this, "Failed to set name: "
          + model.errorMessage, LENGTH_SHORT).show();
      }
    }, t -> { throw new OnErrorNotImplementedException(t); }));
```

Reactive State

```
Observable<SubmitEvent> events = RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()));

disposables.add(events.flatMap(event -> service.setName(event.name)
        .map(response -> SubmitUiModel.success())
        .onErrorReturn(t -> SubmitUiModel.failure(t.getMessage()))
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitUiModel.inProgress()))
    .subscribe(model -> {
      submitView.setEnabled(!model.inProgress);
      progressView.setVisibility(model.inProgress ? VISIBLE : GONE);
      if (!model.inProgress) {
        if (model.success) finish()
        else Toast.makeText(this, "Failed to set name: "
          + model.errorMessage, LENGTH_SHORT).show();
      }
    }, t -> { throw new OnErrorNotImplementedException(t); }));
```

Reactive State

```
Observable<SubmitEvent> events = RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()));

Observable<SubmitUiModel> models = events
    .flatMap(event -> service.setName(event.name)
        .map(response -> SubmitUiModel.success())
        .onErrorReturn(t -> SubmitUiModel.failure(t.getMessage()))
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitUiModel.inProgress()));

disposables.add(models.subscribe(model -> {
      submitView.setEnabled(!model.inProgress);
      progressView.setVisibility(model.inProgress ? VISIBLE : GONE);
      if (!model.inProgress) {
        if (model.success) finish()
        else Toast.makeText(this, "Failed to set name: "
          + model.errorMessage, LENGTH_SHORT).show();
      }
    }, t -> { throw new OnErrorNotImplementedException(t); }));
```

Reactive State

```
Observable<SubmitEvent> events = RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()));

ObservableTransformer<SubmitEvent, SubmitUiModel> submit = events -> events
    .flatMap(event -> service.setName(event.name)
        .map(response -> SubmitUiModel.success())
        .onErrorReturn(t -> SubmitUiModel.failure(t.getMessage()))
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitUiModel.inProgress()));

disposables.add(events.compose(submit).subscribe(model -> {
      submitView.setEnabled(!model.inProgress);
      progressView.setVisibility(model.inProgress ? VISIBLE : GONE);
      if (!model.inProgress) {
        if (model.success) finish()
        else Toast.makeText(this, "Failed to set name: "
          + model.errorMessage, LENGTH_SHORT).show();
      }
    }, t -> { throw new OnErrorNotImplementedException(t); }));
```

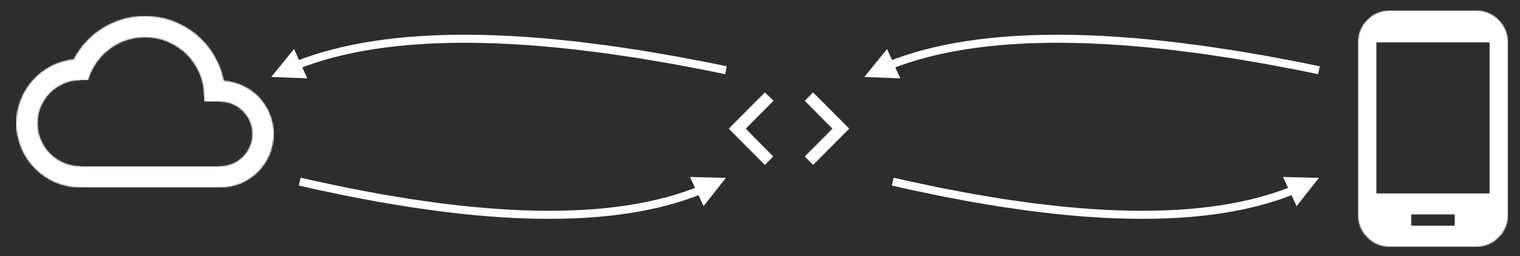
Reactive State

```
Observable<SubmitEvent> events = RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()));

ObservableTransformer<SubmitEvent, SubmitUiModel> submit = events -> events
    .flatMap(event -> service.setName(event.name)
        .map(response -> SubmitUiModel.success())
        .onErrorReturn(t -> SubmitUiModel.failure(t.getMessage()))
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitUiModel.inProgress()));

disposables.add(events.compose(submit).subscribe(model -> {
      submitView.setEnabled(!model.inProgress);
      progressView.setVisibility(model.inProgress ? VISIBLE : GONE);
      if (!model.inProgress) {
        if (model.success) finish()
        else Toast.makeText(this, "Failed to set name: "
          + model.errorMessage, LENGTH_SHORT).show();
      }
    }, t -> { throw new OnErrorNotImplementedException(t); }));
```

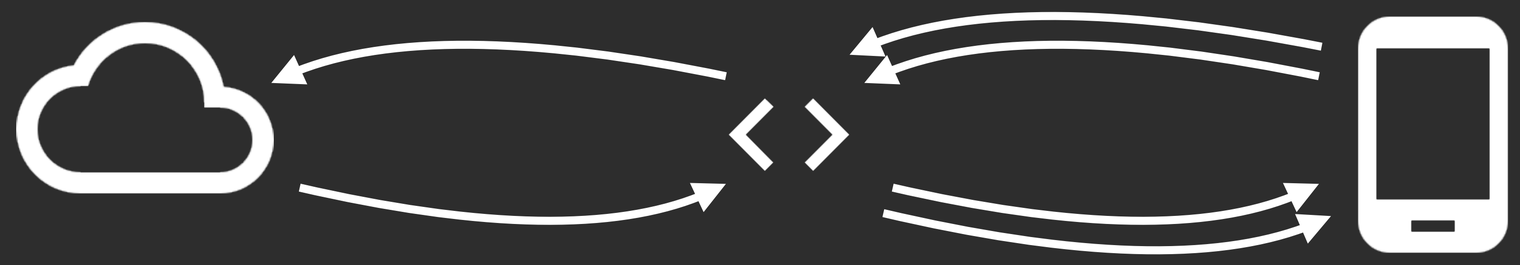
Reactive State

```
ObservableTransformer<SubmitEvent, SubmitUiModel> submit = events -> events
    .flatMap(event -> service.setName(event.name)
        .map(response -> SubmitUiModel.success())
        .onErrorReturn(t -> SubmitUiModel.failure(t.getMessage()))
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitUiModel.inProgress()));

disposables.add(RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()));
    .compose(submit)
    .subscribe(model -> {
      submitView.setEnabled(!model.inProgress);
      progressView.setVisibility(model.inProgress ? VISIBLE : GONE);
      if (!model.inProgress) {
        if (model.success) finish()
        else Toast.makeText(this, "Failed to set name: "
          + model.errorMessage, LENGTH_SHORT).show();
      }
    }, t -> { throw new OnErrorNotImplementedException(t); }));
```

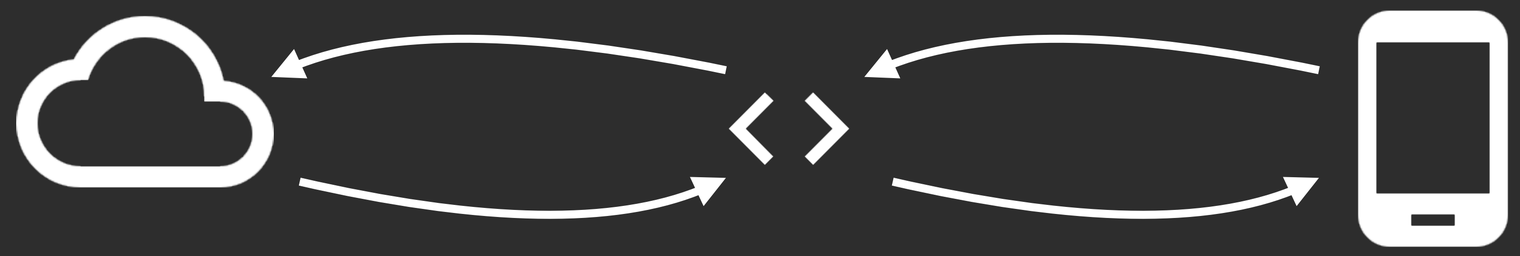
Reactive State



Reactive State



Reactive State



Reactive State

```
ObservableTransformer<SubmitEvent, SubmitUiModel> submit = events -> events
    .flatMap(event -> service.setName(event.name)
        .map(response -> SubmitUiModel.success())
        .onErrorReturn(t -> SubmitUiModel.failure(t.getMessage()))
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitUiModel.inProgress()));

disposables.add(RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()));
    .compose(submit)
    .subscribe(model -> {
      submitView.setEnabled(!model.inProgress);
      progressView.setVisibility(model.inProgress ? VISIBLE : GONE);
      if (!model.inProgress) {
        if (model.success) finish()
        else Toast.makeText(this, "Failed to set name: " + model.message,
          LENGTH_SHORT).show();
      }
    }, t -> { throw new OnErrorNotImplementedException(t); }));
```

Reactive State

```
Observable<SubmitEvent> events = RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()));

ObservableTransformer<SubmitEvent, SubmitUiModel> submit = events -> events
    .flatMap(event -> service.setName(event.name)
        .map(response -> SubmitUiModel.success())
        .onErrorReturn(t -> SubmitUiModel.failure(t.getMessage()))
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitUiModel.inProgress()));

disposables.add(events.compose(submit).subscribe(model -> {
      submitView.setEnabled(!model.inProgress);
      progressView.setVisibility(model.inProgress ? VISIBLE : GONE);
      if (!model.inProgress) {
        if (model.success) finish()
        else Toast.makeText(this, "Failed to set name: " + model.message,
          LENGTH_SHORT).show();
      }
    }, t -> { throw new OnErrorNotImplementedException(t); }));
```

Reactive State

```
Observable<SubmitEvent> events = RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()));

ObservableTransformer<SubmitEvent, SubmitUiModel> submit = events -> events
    .flatMap(event -> service.setName(event.name)
        .map(response -> SubmitUiModel.success())
        .onErrorReturn(t -> SubmitUiModel.failure(t.getMessage()))
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitUiModel.inProgress()));

disposables.add(events.compose(submit).subscribe(model -> {
      submitView.setEnabled(!model.inProgress);
      progressView.setVisibility(model.inProgress ? VISIBLE : GONE);
      if (!model.inProgress) {
        if (model.success) finish()
        else Toast.makeText(this, "Failed to set name: " + model.message,
          LENGTH_SHORT).show();
      }
    }, t -> { throw new OnErrorNotImplementedException(t); }));
```

Reactive State

```
class SubmitEvent { /* ... */ }

Observable<SubmitEvent> events = RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()));
```

Reactive State

```
abstract class SubmitUiEvent {}
class SubmitEvent extends SubmitUiEvent { /* ... */ }
class CheckNameEvent extends SubmitUiEvent { /* ... */ }

Observable<SubmitEvent> submitEvents = RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()));
```

Reactive State

```
abstract class SubmitUiEvent {}
class SubmitEvent extends SubmitUiEvent { /* ... */ }
class CheckNameEvent extends SubmitUiEvent { /* ... */ }

Observable<SubmitEvent> submitEvents = RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()));

Observable<CheckNameEvent> checkNameEvents =
    RxTextView.afterTextChanges(nameView)
        .map(text -> new CheckNameEvent(text));
```

Reactive State

```
abstract class SubmitUiEvent {}
class SubmitEvent extends SubmitUiEvent { /* ... */ }
class CheckNameEvent extends SubmitUiEvent { /* ... */ }

Observable<SubmitEvent> submitEvents = RxView.clicks(submitView)
    .map(ignored -> new SubmitEvent(nameView.getText().toString()));

Observable<CheckNameEvent> checkNameEvents =
    RxTextView.afterTextChanges(nameView)
        .map(text -> new CheckNameEvent(text));

Observable<SubmitUiEvent> events =
    Observable.merge(submitEvents, checkNameEvents);
```

Reactive State

```
Observable<SubmitUiEvent> events = /* ... */;

ObservableTransformer<SubmitEvent, SubmitUiModel> submit = events -> events
    .flatMap(event -> service.setName(event.name)
        .map(response -> SubmitUiModel.success())
        .onErrorReturn(t -> SubmitUiModel.failure(t.getMessage()))
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitUiModel.inProgress()));

disposables.add(events.compose(submit).subscribe(model -> {
      submitView.setEnabled(!model.inProgress);
      progressView.setVisibility(model.inProgress ? VISIBLE : GONE);
      if (!model.inProgress) {
        if (model.success) finish()
        else Toast.makeText(this, "Failed to set name: " + model.message,
          LENGTH_SHORT).show();
      }
    }, t -> { throw new OnErrorNotImplementedException(t); }));
```

Reactive State

```
Observable<SubmitUiEvent> events = /* ... */;

ObservableTransformer<SubmitEvent, SubmitUiModel> submit = events -> events
    .flatMap(event -> service.setName(event.name)
        .map(response -> SubmitUiModel.success())
        .onErrorReturn(t -> SubmitUiModel.failure(t.getMessage()))
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitUiModel.inProgress()));

disposables.add(events.compose(submit).subscribe(model -> {
      submitView.setEnabled(!model.inProgress);
      progressView.setVisibility(model.inProgress ? VISIBLE : GONE);
      if (!model.inProgress) {
        if (model.success) finish()
        else Toast.makeText(this, "Failed to set name: " + model.message,
          LENGTH_SHORT).show();
      }
    }, t -> { throw new OnErrorNotImplementedException(t); }));
```

Reactive State

```
ObservableTransformer<SubmitEvent, SubmitUiModel> submit = events -> events
    .flatMap(event -> service.setName(event.name)
        .map(response -> SubmitUiModel.success())
        .onErrorReturn(t -> SubmitUiModel.failure(t.getMessage()))
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitUiModel.inProgress()));
```

Reactive State

```
ObservableTransformer<SubmitEvent, SubmitUiModel> submit = events -> events
    .flatMap(event -> service.setName(event.name)
        .map(response -> SubmitUiModel.success())
        .onErrorReturn(t -> SubmitUiModel.failure(t.getMessage()))
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitUiModel.inProgress()));

ObservableTransformer<CheckNameEvent, SubmitUiModel> checkName = events -> events
    .switchMap(event -> event
        .delay(200, MILLISECONDS, AndroidSchedulers.mainThread())
        .flatMap(event -> service.checkName(event.name))
        .map(response -> ???)
        .onErrorReturn(t -> ???)
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(???));
```

Reactive State

```
ObservableTransformer<SubmitEvent, SubmitUiModel> submit = events -> events
    .flatMap(event -> service.setName(event.name)
        .map(response -> SubmitUiModel.success())
        .onErrorReturn(t -> SubmitUiModel.failure(t.getMessage()))
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitUiModel.inProgress()));

ObservableTransformer<CheckNameEvent, SubmitUiModel> checkName = events -> events
    .switchMap(event -> event
        .delay(200, MILLISECONDS, AndroidSchedulers.mainThread())
        .flatMap(event -> service.checkName(event.name))
        .map(response -> ???)
        .onErrorReturn(t -> ???)
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(???));
```

Reactive State

```
ObservableTransformer<SubmitEvent, SubmitUiModel> submit = events -> events
    .flatMap(event -> service.setName(event.name)
        .map(response -> SubmitUiModel.success())
        .onErrorReturn(t -> SubmitUiModel.failure(t.getMessage()))
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitUiModel.inProgress()));

ObservableTransformer<CheckNameEvent, SubmitUiModel> checkName = events -> events
    .switchMap(event -> event
        .delay(200, MILLISECONDS, AndroidSchedulers.mainThread())
        .flatMap(event -> service.checkName(event.name))
        .map(response -> ???)
        .onErrorReturn(t -> ???)
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(???));
```

Reactive State

```
ObservableTransformer<SubmitEvent, SubmitUiModel> submit = events -> events
    .flatMap(event -> service.setName(event.name)
        .map(response -> SubmitUiModel.success())
        .onErrorReturn(t -> SubmitUiModel.failure(t.getMessage()))
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitUiModel.inProgress()));

ObservableTransformer<CheckNameEvent, SubmitUiModel> checkName = events -> events
    .switchMap(event -> event
        .delay(200, MILLISECONDS, AndroidSchedulers.mainThread())
        .flatMap(event -> service.checkName(event.name))
        .map(response -> ???)
        .onErrorReturn(t -> ???)
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(???));
```

Reactive State

```
ObservableTransformer<SubmitEvent, SubmitUiModel> submit = /* ... */;

ObservableTransformer<CheckNameEvent, SubmitUiModel> checkName = /* ... */;
```

Reactive State

```
ObservableTransformer<SubmitEvent, SubmitUiModel> submit = /* ... */;

ObservableTransformer<CheckNameEvent, SubmitUiModel> checkName = /* ... */;

ObservableTransformer<SubmitUiEvent, SubmitUiModel> submitUi =
    events -> ???;
```

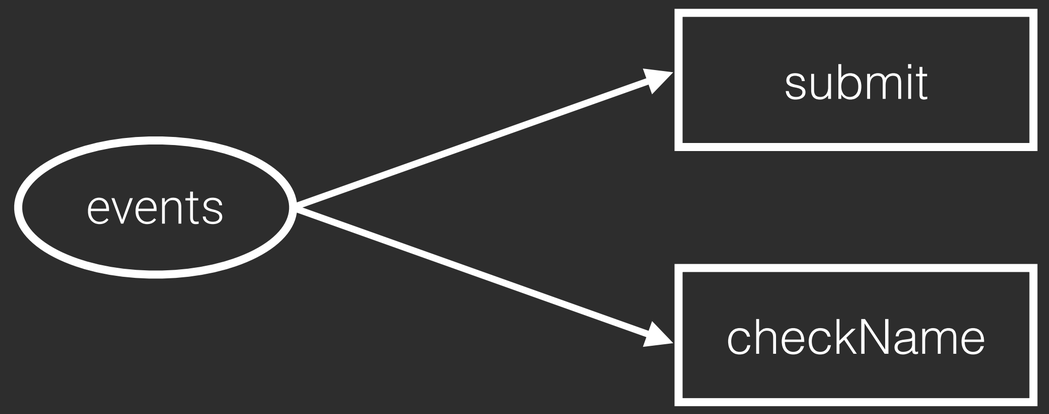
Reactive State

```
ObservableTransformer<SubmitEvent, SubmitUiModel> submit = /* ... */;

ObservableTransformer<CheckNameEvent, SubmitUiModel> checkName = /* ... */;

ObservableTransformer<SubmitUiEvent, SubmitUiModel> submitUi =
    events -> ???;
```

events

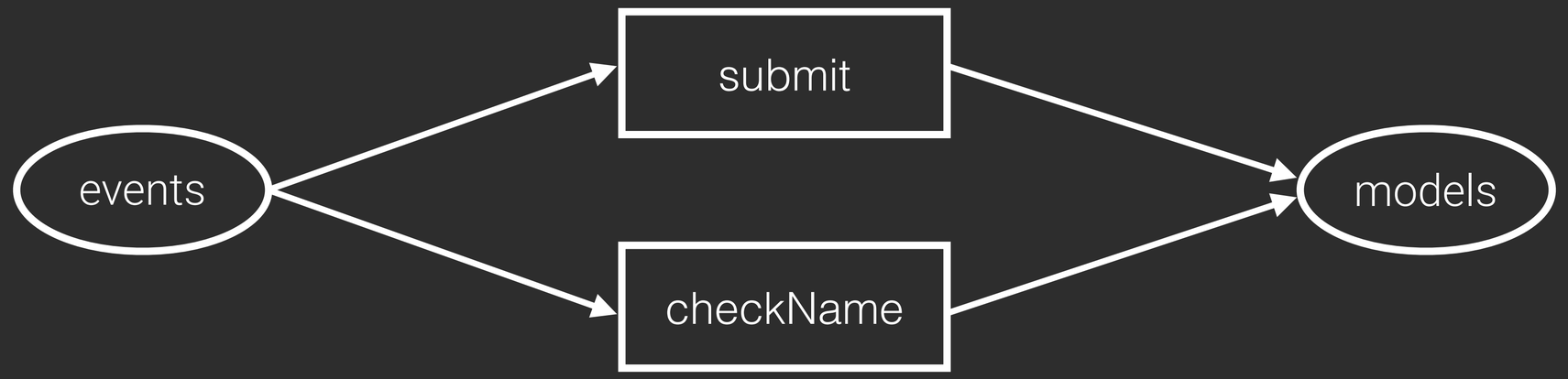
Reactive State

```
ObservableTransformer<SubmitEvent, SubmitUiModel> submit = /* ... */;

ObservableTransformer<CheckNameEvent, SubmitUiModel> checkName = /* ... */;

ObservableTransformer<SubmitUiEvent, SubmitUiModel> submitUi =
    events -> ???;
```



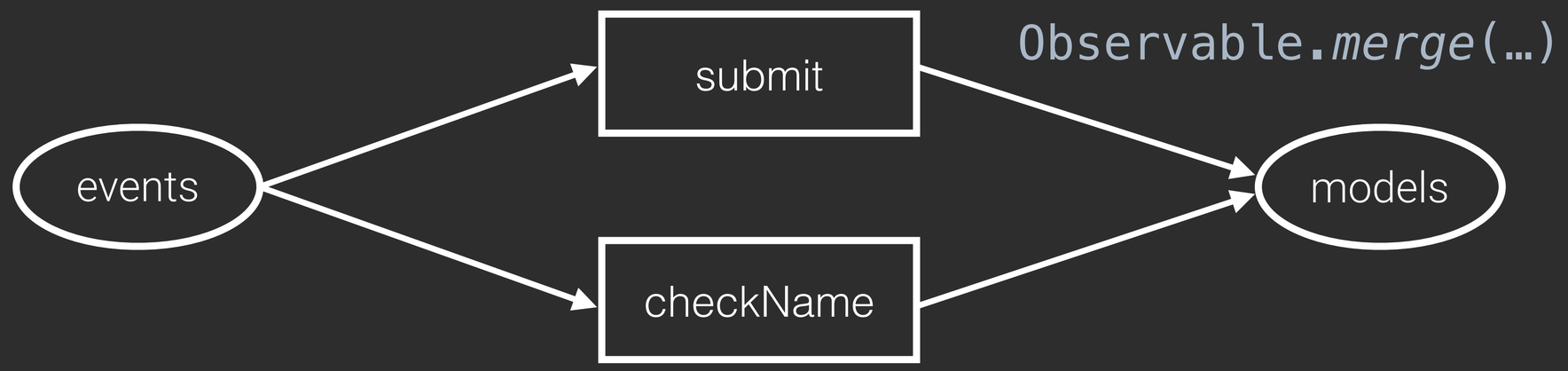
Reactive State

```
ObservableTransformer<SubmitEvent, SubmitUiModel> submit = /* ... */;

ObservableTransformer<CheckNameEvent, SubmitUiModel> checkName = /* ... */;

ObservableTransformer<SubmitUiEvent, SubmitUiModel> submitUi =
    events -> ???;
```



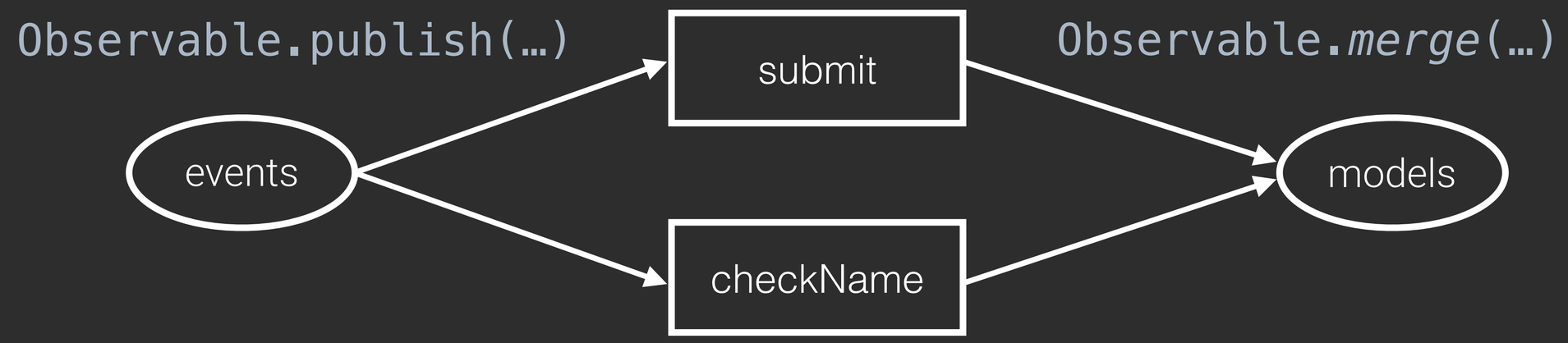
Reactive State

```
ObservableTransformer<SubmitEvent, SubmitUiModel> submit = /* ... */;

ObservableTransformer<CheckNameEvent, SubmitUiModel> checkName = /* ... */;

ObservableTransformer<SubmitUiEvent, SubmitUiModel> submitUi =
    events -> ???;
```



Reactive State

```
ObservableTransformer<SubmitEvent, SubmitUiModel> submit = /* ... */;

ObservableTransformer<CheckNameEvent, SubmitUiModel> checkName = /* ... */;

ObservableTransformer<SubmitUiEvent, SubmitUiModel> submitUi =
    events -> ???;
```



Reactive State

```
ObservableTransformer<SubmitEvent, SubmitUiModel> submit = /* ... */;

ObservableTransformer<CheckNameEvent, SubmitUiModel> checkName = /* ... */;

ObservableTransformer<SubmitUiEvent, SubmitUiModel> submitUi =
    events -> ???;
```

`Observable.publish(…)`

`Observable.merge(…)`

Reactive State

```
ObservableTransformer<SubmitEvent, SubmitUiModel> submit = /* ... */;

ObservableTransformer<CheckNameEvent, SubmitUiModel> checkName = /* ... */;

ObservableTransformer<SubmitUiEvent, SubmitUiModel> submitUi =
    events -> events.publish(shared -> Observable.merge());
```

Reactive State

```
ObservableTransformer<SubmitEvent, SubmitUiModel> submit = /* ... */;

ObservableTransformer<CheckNameEvent, SubmitUiModel> checkName = /* ... */;

ObservableTransformer<SubmitUiEvent, SubmitUiModel> submitUi =
    events -> events.publish(shared -> Observable.merge(
        shared.ofType(SubmitEvent.class).compose(submit),
        shared.ofType(CheckNameEvent.class).compose(checkName)));
```

Reactive State

```
Observable<SubmitUiEvent> events = /* ... */;

ObservableTransformer<SubmitUiEvent, SubmitUiModel> submitUi =
    events -> events.publish(shared -> Observable.merge(
        shared.ofType(SubmitEvent.class).compose(submit),
        shared.ofType(CheckNameEvent.class).compose(checkName)));

disposables.add(events.compose(submitUi).subscribe(model -> {
      submitView.setEnabled(!model.inProgress);
      progressView.setVisibility(model.inProgress ? VISIBLE : GONE);
      if (!model.inProgress) {
        if (model.success) finish()
        else Toast.makeText(this, "Failed to set name: " + model.message,
          LENGTH_SHORT).show();
      }
    }, t -> { throw new OnErrorNotImplementedException(t); }));
```

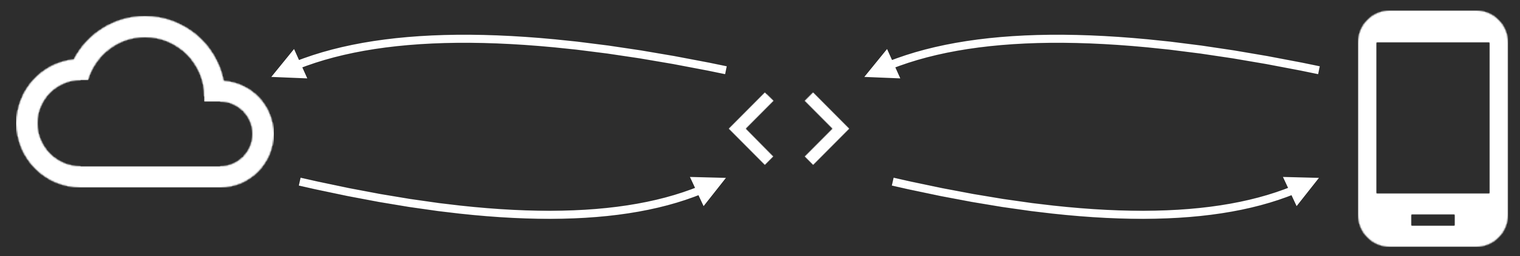
Reactive State

```
Observable<SubmitUiEvent> events = /* ... */;

ObservableTransformer<SubmitUiEvent, SubmitUiModel> submitUi =
    events -> events.publish(shared -> Observable.merge(
        shared.ofType(SubmitEvent.class).compose(submit),
        shared.ofType(CheckNameEvent.class).compose(checkName)));

disposables.add(events.compose(submitUi).subscribe(model -> {
      submitView.setEnabled(!model.inProgress);
      progressView.setVisibility(model.inProgress ? VISIBLE : GONE);
      if (!model.inProgress) {
        if (model.success) finish()
        else Toast.makeText(this, "Failed to set name: " + model.message,
          LENGTH_SHORT).show();
      }
    }, t -> { throw new OnErrorNotImplementedException(t); }));
```

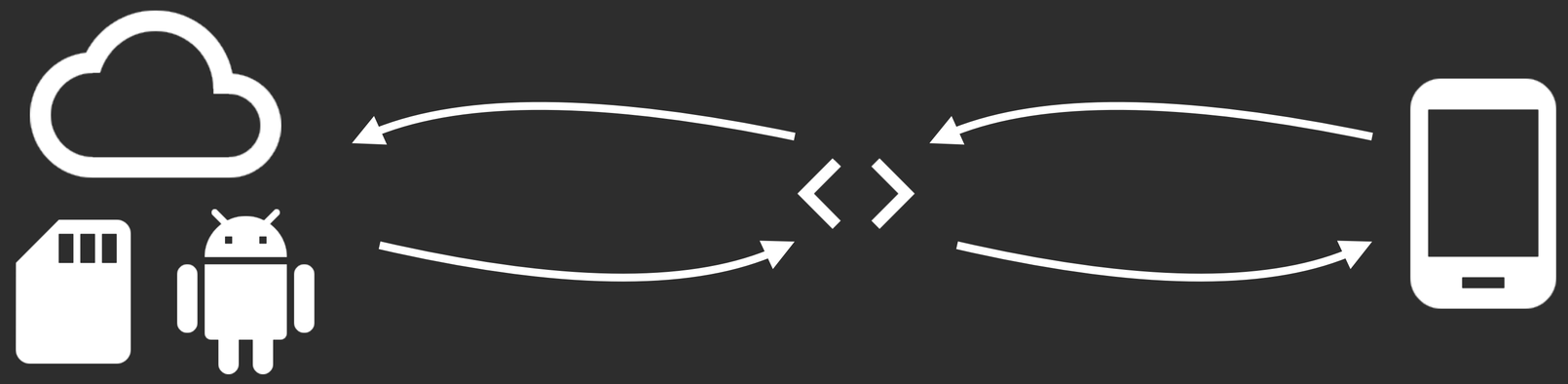
Reactive State

```
Observable<SubmitUiEvent> events = /* ... */;

ObservableTransformer<SubmitUiEvent, SubmitUiModel> submitUi =
    events -> events.publish(shared -> Observable.merge(
        shared.ofType(SubmitEvent.class).compose(submit),
        shared.ofType(CheckNameEvent.class).compose(checkName)));

disposables.add(events.compose(submitUi).subscribe(model -> {
      submitView.setEnabled(!model.inProgress);
      progressView.setVisibility(model.inProgress ? VISIBLE : GONE);
      if (!model.inProgress) {
        if (model.success) finish()
        else Toast.makeText(this, "Failed to set name: " + model.message,
          LENGTH_SHORT).show();
      }
    }, t -> { throw new OnErrorNotImplementedException(t); }));
```

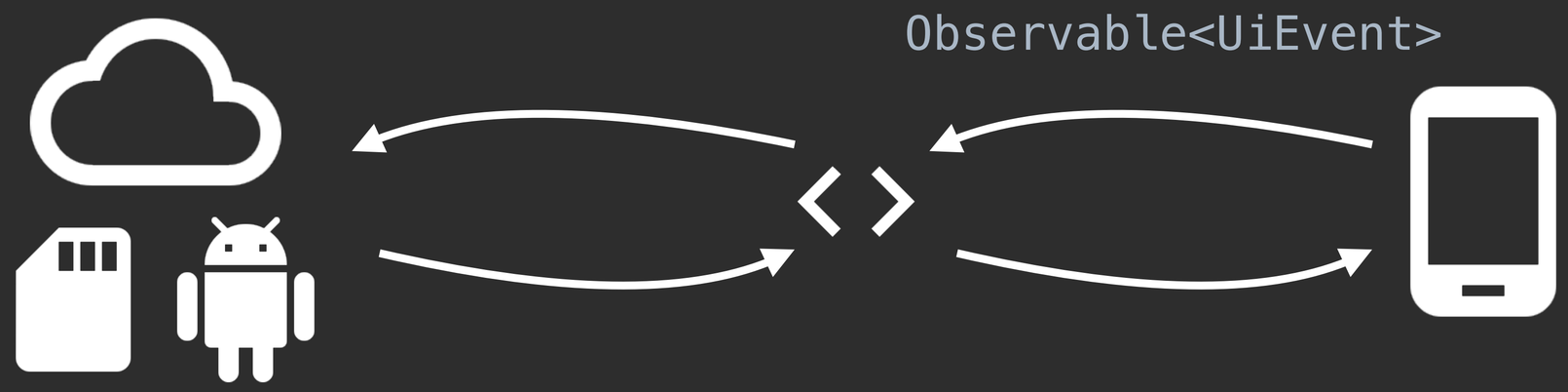
Reactive State



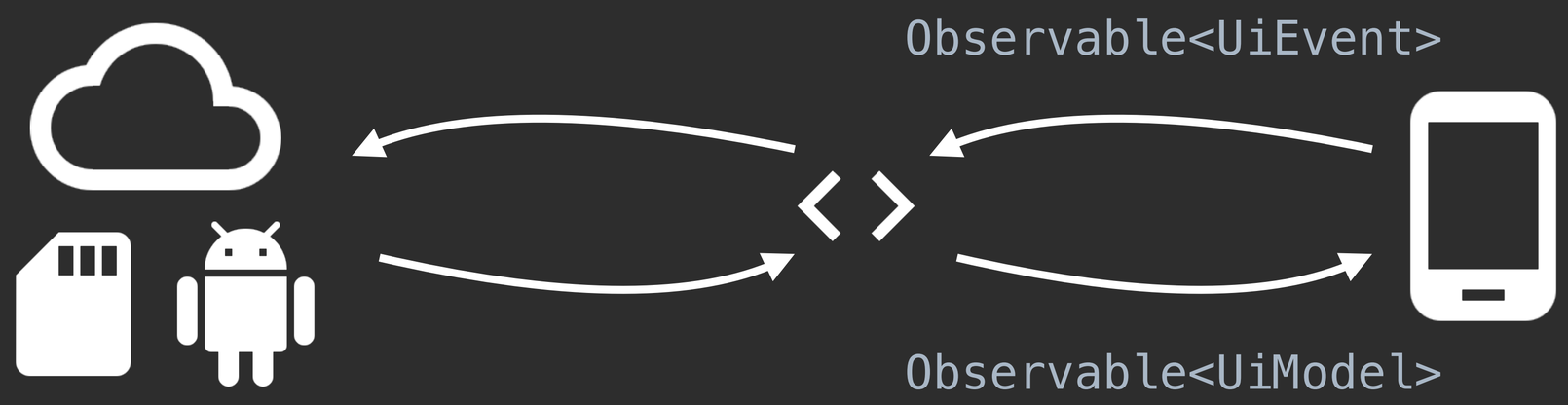
Reactive State



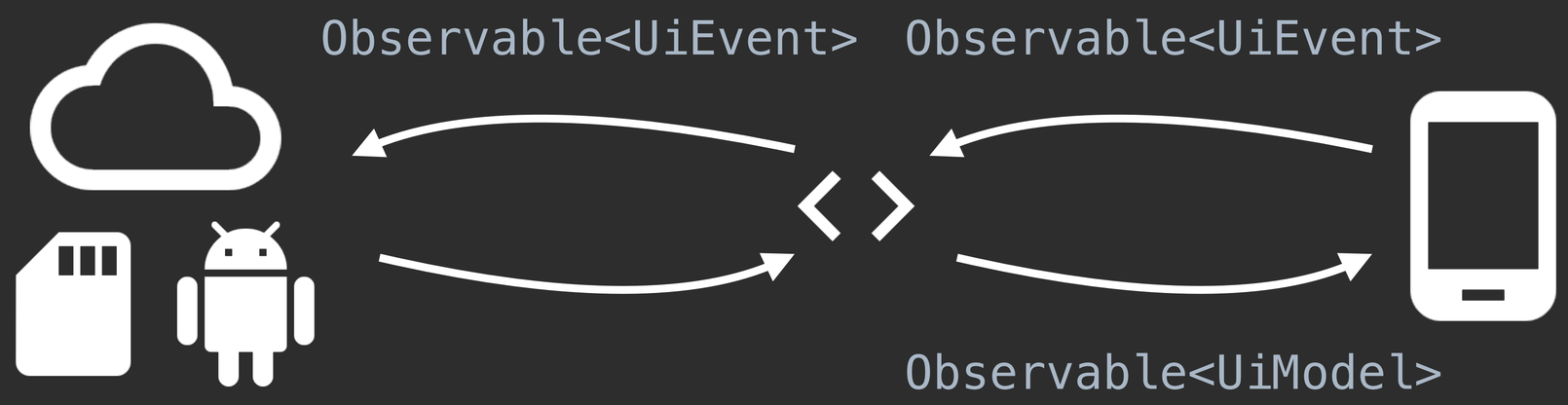
Reactive State



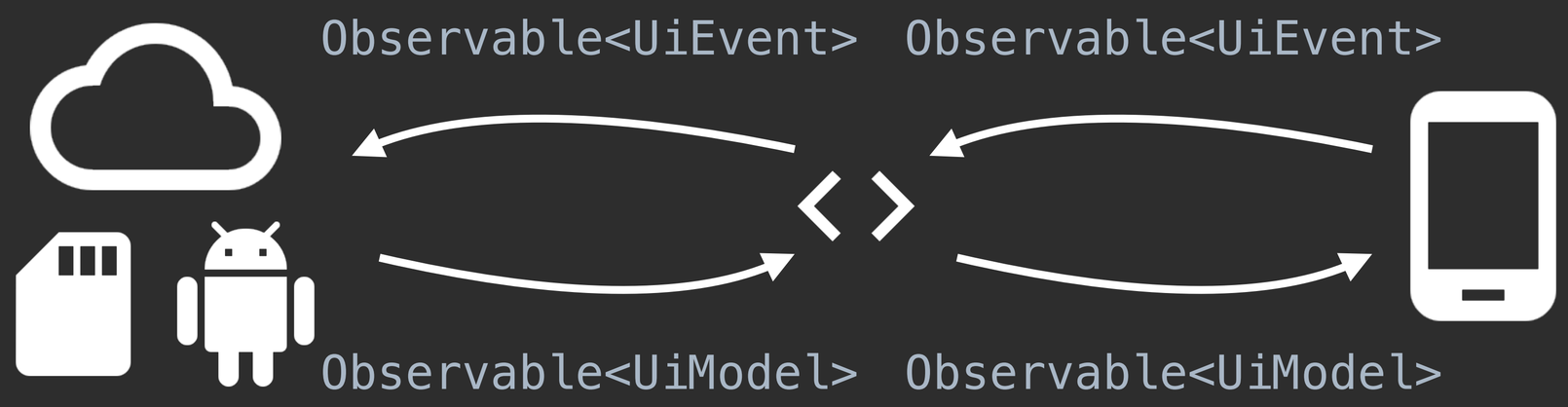
Reactive State



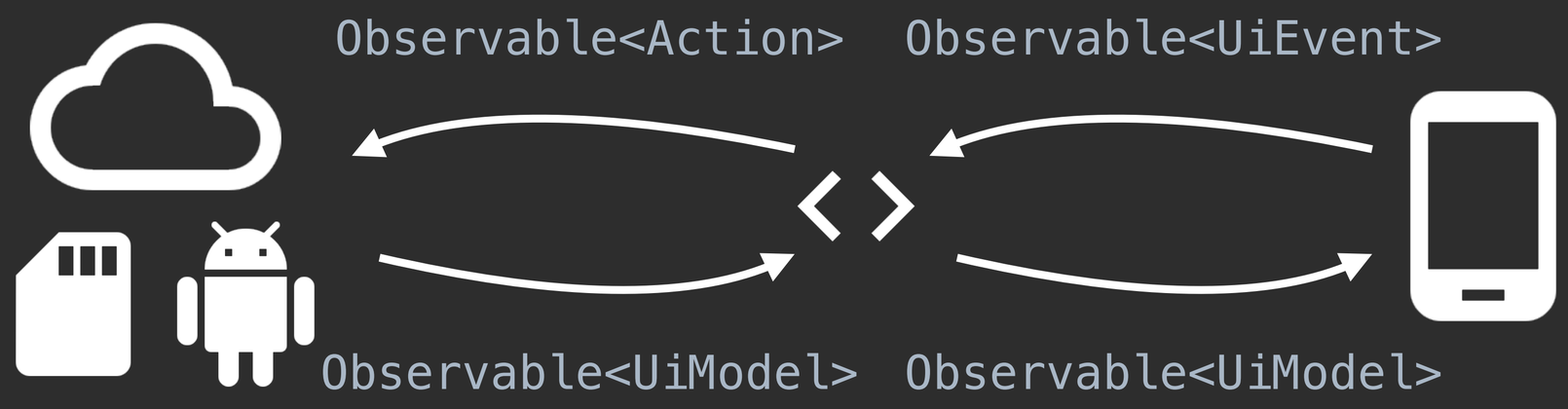
Reactive State



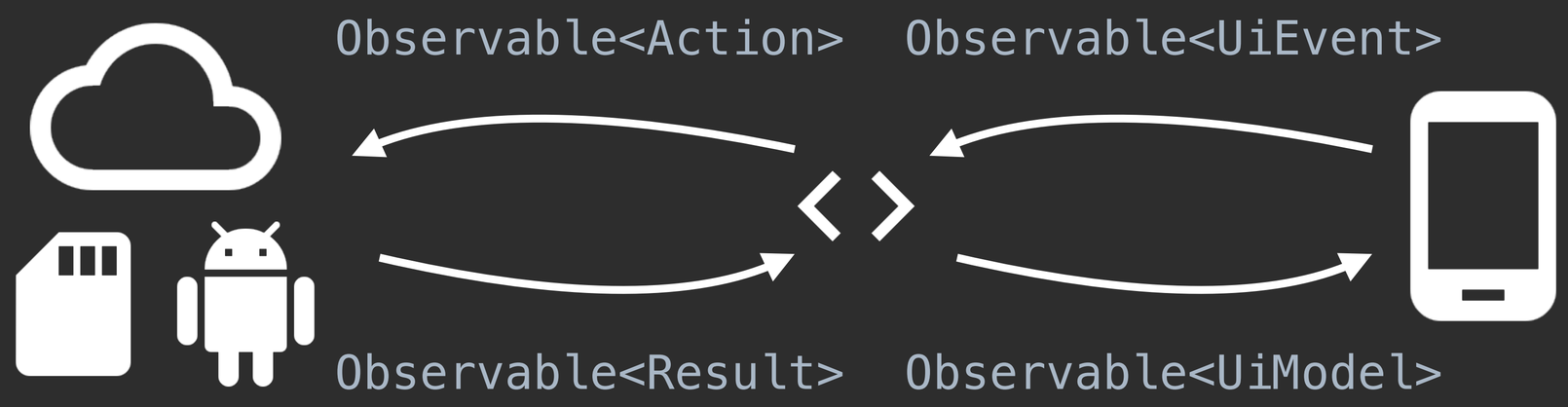
Reactive State



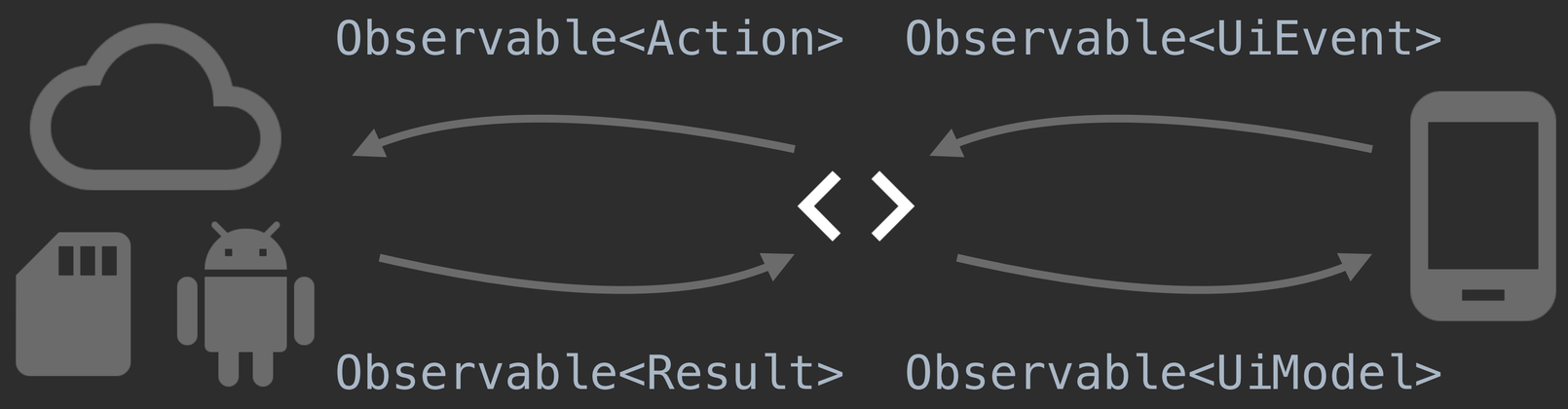
Reactive State



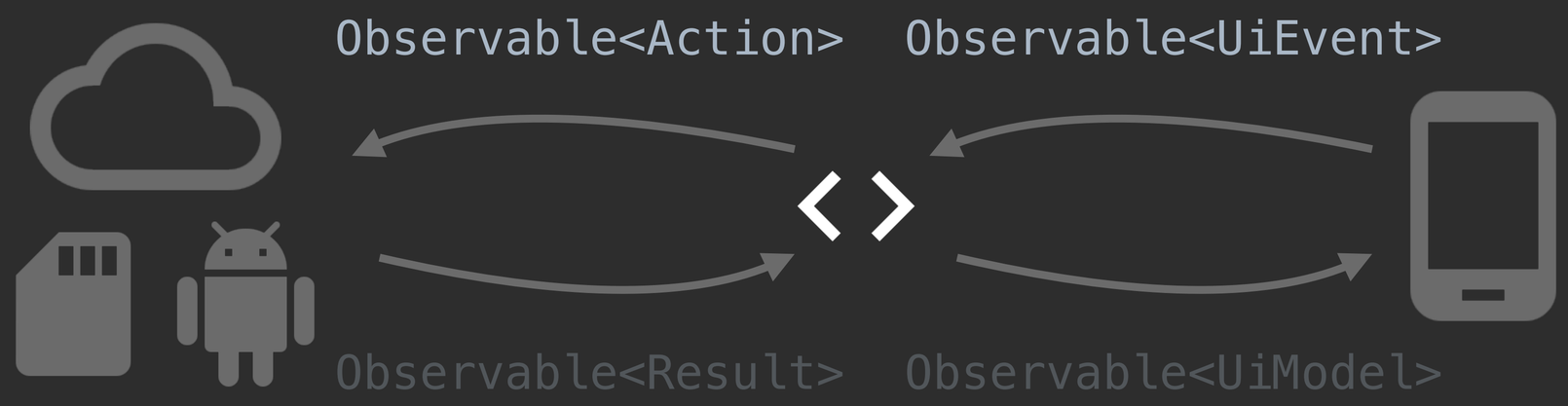
Reactive State



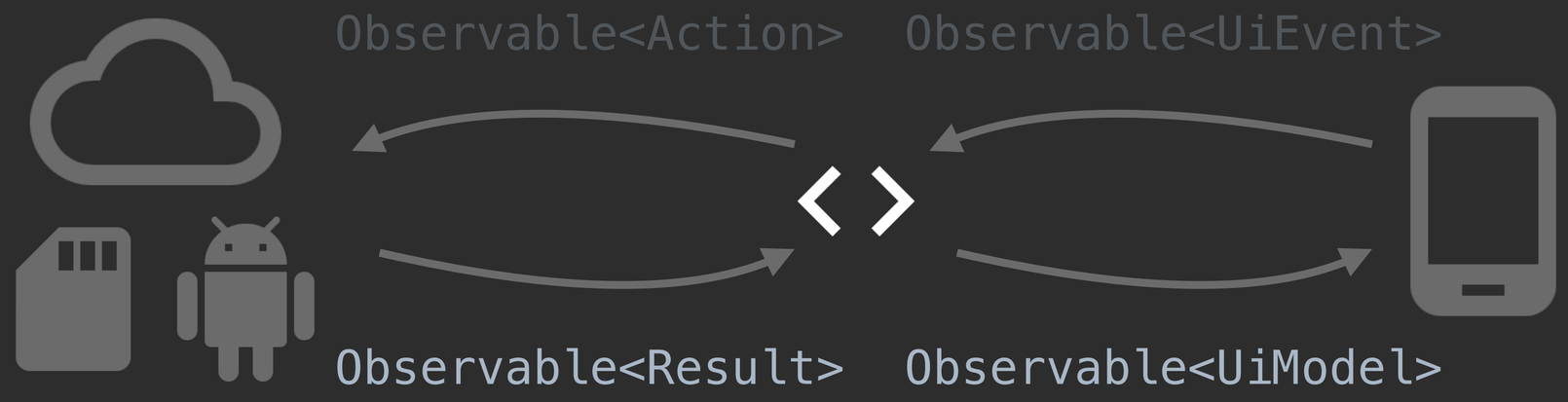
Reactive State



Reactive State



Reactive State



Reactive State

```
final class SubmitUiModel {
  final boolean inProgress;
  final boolean success;
  final String errorMessage;

  private SubmitUiModel(
      boolean inProgress, boolean success, String errorMessage) {
    // ...
  }

  static SubmitUiModel inProgress() { /* ... */ }
  static SubmitUiModel success() { /* ... */ }
  static SubmitUiModel failure(String errorMessage) { /* ... */ }
}
```

Reactive State

```
inProgress
false
success
false
errorMessage
null
```

Reactive State

inProgress
false

success
false

errorMessage
null

```
CheckNameResult.IN_FLIGHT
```

Reactive State

inProgress
false

success
false

errorMessage
null

CheckNameResult.IN_FLIGHT

inProgress
true

success
false

errorMessage
null

Reactive State

rogress
se

cess
se

orMessage
l

CheckNameResult.IN_FLIGHT

inProgress
true

success
false

errorMessage
null

CheckNameResult.SUCCESS

Reactive State

ckNameResult.IN_FLIGHT

inProgress
true

success
false

errorMessage
null

CheckNameResult.SUCCESS

inProgress
false

success
false

errorMessage
null

Reactive State

```
rogress
e
cess
se
orMessage
l
```

```
CheckNameResult.SUCCESS
```

```
inProgress
false
success
false
errorMessage
null
```

```
CheckNameResult.IN_FLIGHT
```

Reactive State

ckNameResult.SUCCESS

inProgress
false

success
false

errorMessage
null

CheckNameResult.IN_FLIGHT

inProgress
true

success
false

errorMessage
null

Reactive State

Progress
lse

ccess
lse

rorMessage
ll

CheckNameResult.IN_FLIGHT

inProgress
true

success
false

errorMessage
null

SubmitResult.IN_FLIGHT

Reactive State

eckNameResult.IN_FLIGHT

inProgress
true

success
false

errorMessage
null

SubmitResult.IN_FLIGHT

inProgress
true

success
false

errorMessage
null

Reactive State

inProgress
true

success
false

errorMessage
null

SubmitResult.IN_FLIGHT

inProgress
true

success
false

errorMessage
null

SubmitResult.SUCCESS

Reactive State

```
SubmitResult.IN_FLIGHT
```

inProgress
true

success
false

errorMessage
null

```
SubmitResult.SUCCESS
```

inProgress
false

success
true

errorMessage
null

Reactive State

```
SubmitUiModel initialState = SubmitUiModel.idle();
```

Reactive State

```
SubmitUiModel initialState = SubmitUiModel.idle();

Observable<Result> results = /* ... */;
```

Reactive State

```
SubmitUiModel initialState = SubmitUiModel.idle();

Observable<Result> results = /* ... */;

Observable<SubmitUiModel> uiModels = results
    .scan(initialState, (state, result) -> /* ... */);
```

Reactive State

```
SubmitUiModel initialState = SubmitUiModel.idle();

Observable<Result> results = /* ... */;

Observable<SubmitUiModel> uiModels = results
    .scan(initialState, (state, result) -> {
      if (result == CheckNameResult.IN_FLIGHT
          || result == SubmitResult.IN_FLIGHT)
        return SubmitUiModel.inProgress();
      if (result == CheckNameResult.SUCCESS)
        return SubmitUiModel.idle();
      if (result == SubmitResult.SUCCESS)
        return SubmitUiModel.success();
      // TODO handle check name and submit failures...
      throw new IllegalArgumentException("Unknown result: " + result);
    });
```

Reactive State

```
ObservableTransformer<SubmitEvent, SubmitUiModel> submit =
    events -> events.flatMap(event -> service.setName(event.name)
        .map(response -> SubmitUiModel.success())
        .onErrorReturn(t -> SubmitUiModel.failure(t.getMessage()))
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitUiModel.inProgress()));

ObservableTransformer<CheckNameEvent, SubmitUiModel> checkName =
    events -> events.switchMap(event -> event
        .delay(200, MILLISECONDS, AndroidSchedulers.mainThread())
        .flatMap(event -> service.checkName(event.name))
        .map(response -> ???)
        .onErrorReturn(t -> ???)
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(???));
```

Reactive State

```
ObservableTransformer<SubmitAction, SubmitUiModel> submit =
    actions -> actions.flatMap(action -> service.setName(action.name)
        .map(response -> SubmitUiModel.success())
        .onErrorReturn(t -> SubmitUiModel.failure(t.getMessage()))
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitUiModel.inProgress()));

ObservableTransformer<CheckNameAction, SubmitUiModel> checkName =
    actions -> actions.switchMap(action -> action
        .delay(200, MILLISECONDS, AndroidSchedulers.mainThread())
        .flatMap(action -> service.checkName(action.name))
        .map(response -> ???)
        .onErrorReturn(t -> ???)
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(???));
```

Reactive State

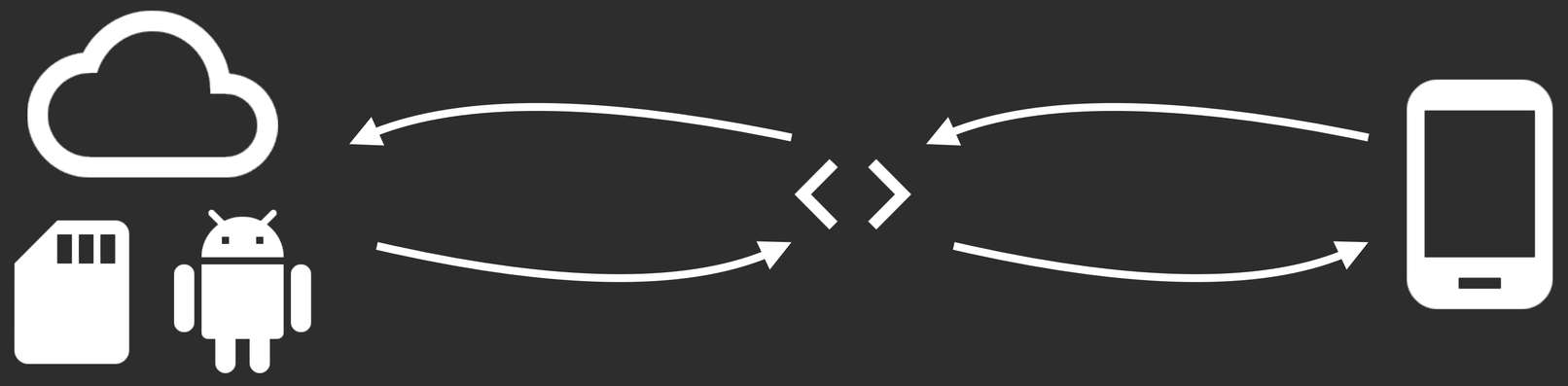
```
ObservableTransformer<SubmitAction, SubmitUiModel> submit =
    actions -> actions.flatMap(action -> service.setName(action.name)
        .map(response -> SubmitUiModel.success())
        .onErrorReturn(t -> SubmitUiModel.failure(t.getMessage()))
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitUiModel.inProgress()));

ObservableTransformer<CheckNameAction, SubmitUiModel> checkName =
    actions -> actions.switchMap(action -> action
        .delay(200, MILLISECONDS, AndroidSchedulers.mainThread())
        .flatMap(action -> service.checkName(action.name))
        .map(response -> ???)
        .onErrorReturn(t -> ???)
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(???));
```

Reactive State

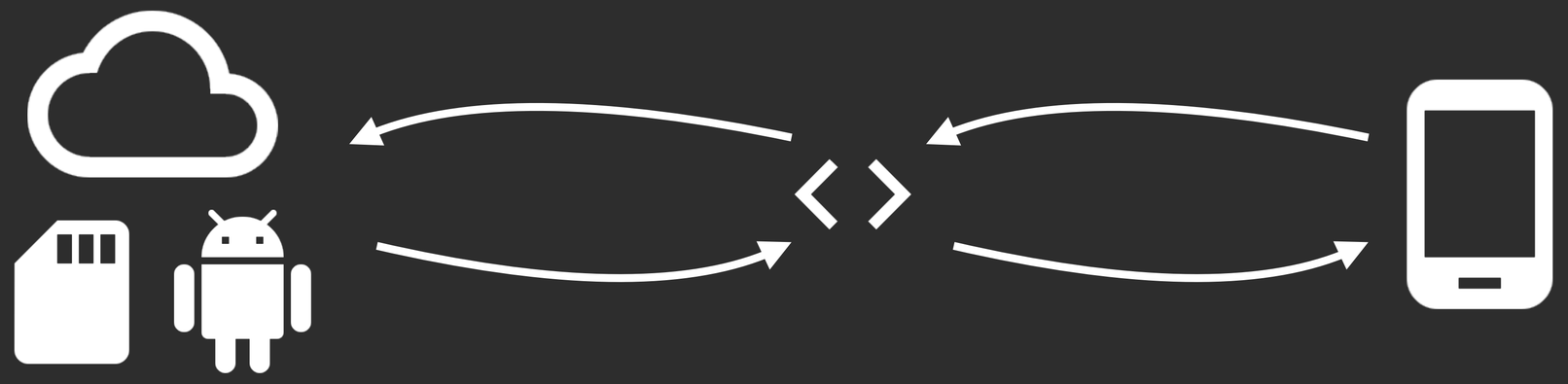
```
ObservableTransformer<SubmitAction, SubmitResult> submit =
    actions -> actions.flatMap(action -> service.setName(action.name)
        .map(response -> SubmitResult.SUCCESS)
        .onErrorReturn(t -> SubmitResult.failure(t.getMessage()))
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitResult.IN_FLIGHT));

ObservableTransformer<CheckNameAction, CheckNameResult> checkName =
    actions -> actions.switchMap(action -> action
        .delay(200, MILLISECONDS, AndroidSchedulers.mainThread())
        .flatMap(action -> service.checkName(action.name))
        .map(response -> CheckNameResult.SUCCESS)
        .onErrorReturn(t -> CheckNameResult.failure(t.getMessage()))
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(CheckNameResult.IN_FLIGHT));
```

Reactive State

```
ObservableTransformer<SubmitAction, SubmitResult> submit =
    actions -> actions.flatMap(action -> service.setName(action.name)
        .map(response -> SubmitResult.SUCCESS)
        .onErrorReturn(t -> SubmitResult.failure(t.getMessage()))
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(SubmitResult.IN_FLIGHT));

ObservableTransformer<CheckNameAction, CheckNameResult> checkName =
    actions -> actions.switchMap(action -> action
        .delay(200, MILLISECONDS, AndroidSchedulers.mainThread())
        .flatMap(action -> service.checkName(action.name))
        .map(response -> CheckNameResult.SUCCESS)
        .onErrorReturn(t -> CheckNameResult.failure(t.getMessage()))
        .observeOn(AndroidSchedulers.mainThread())
        .startWith(CheckNameResult.IN_FLIGHT));
```

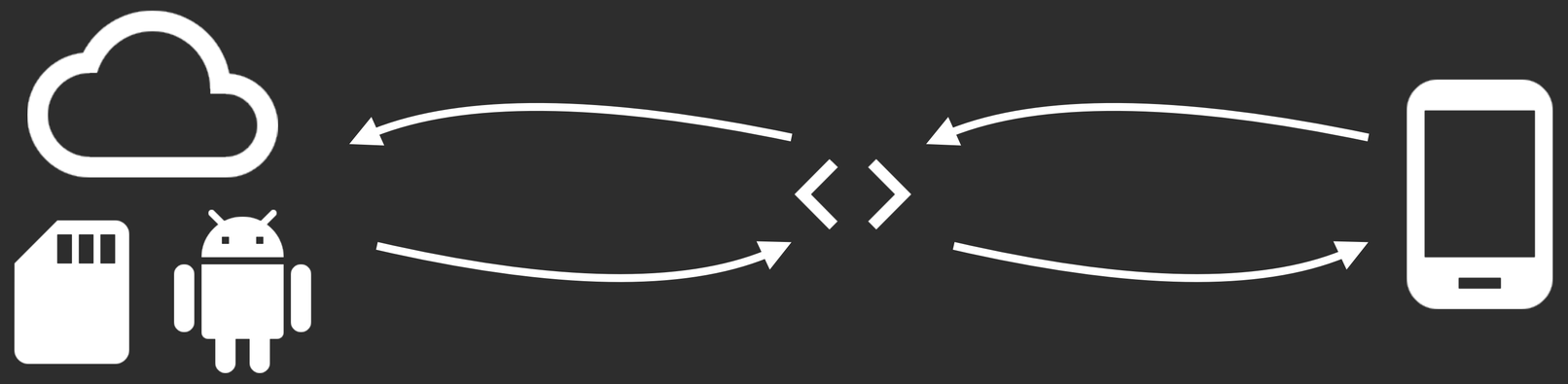


Observable<UiEvent>



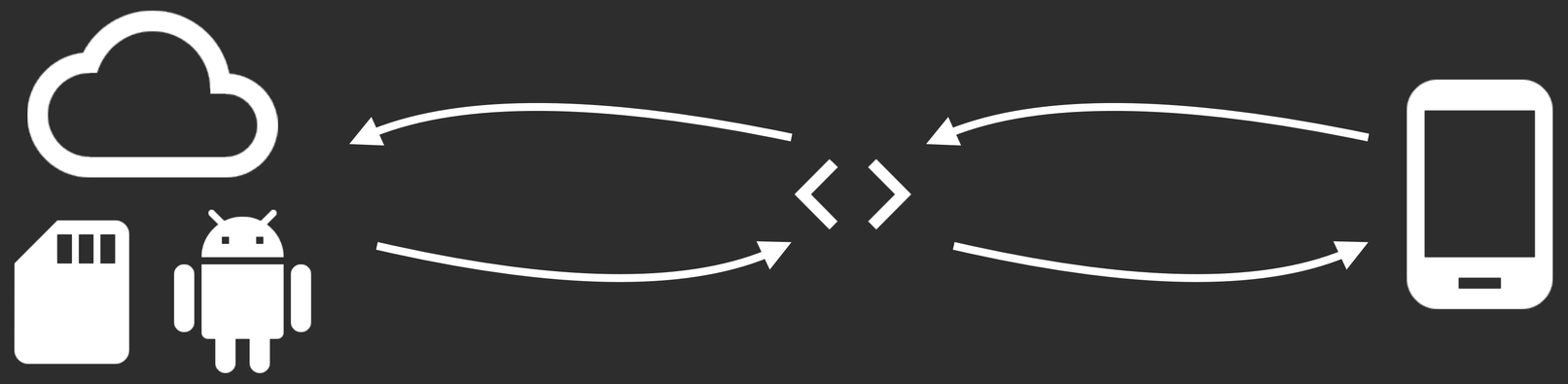
Observable<Action>

Observable<UiEvent>



Observable<Action>

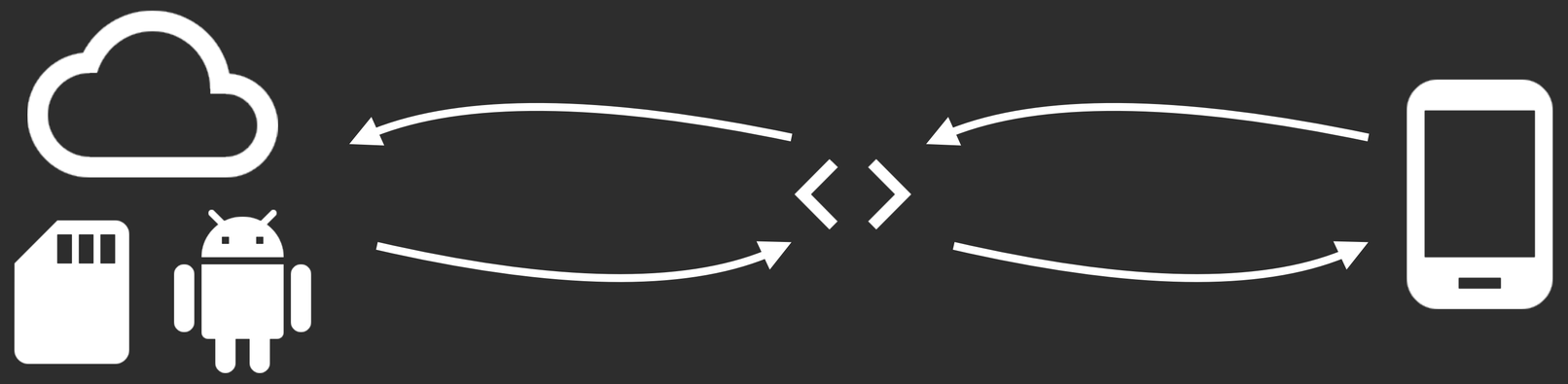
Observable<UiEvent>



Observable<Result>

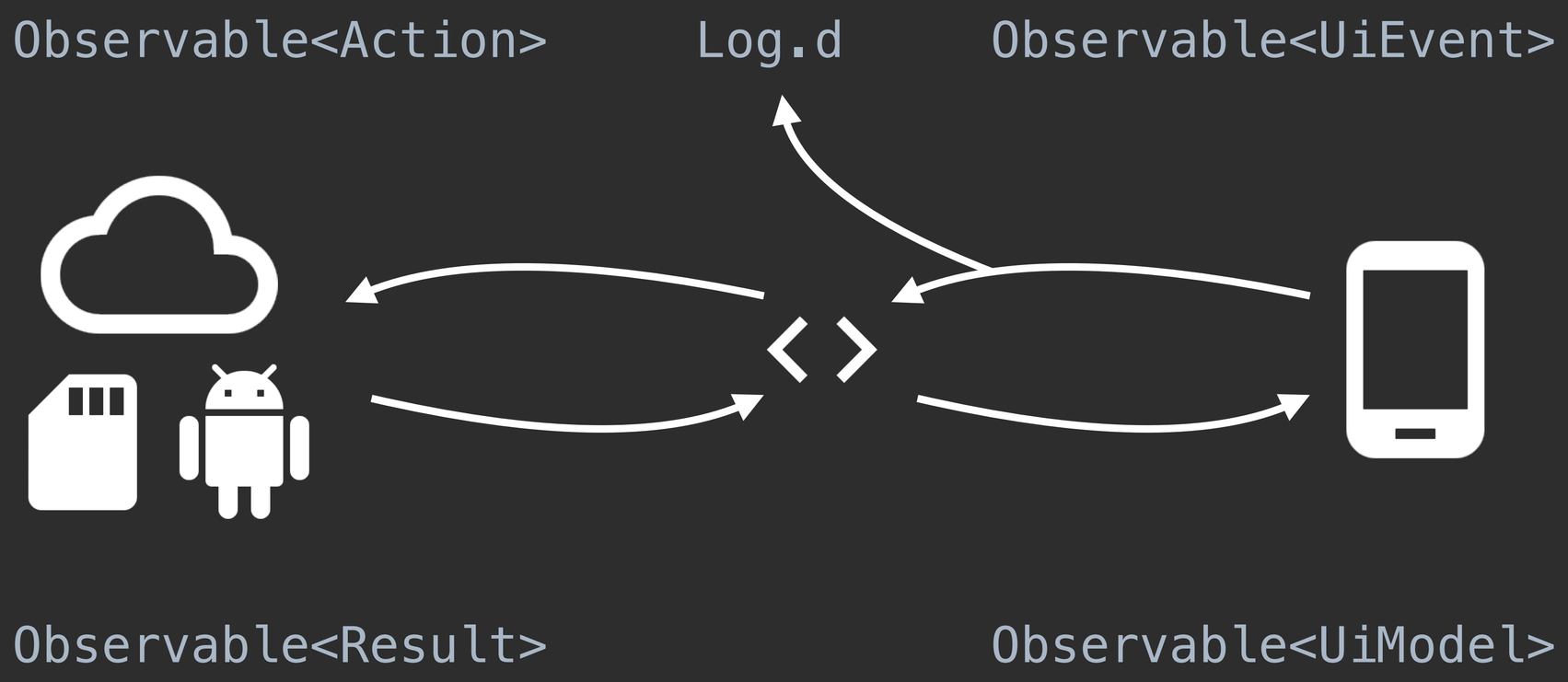
Observable<Action>

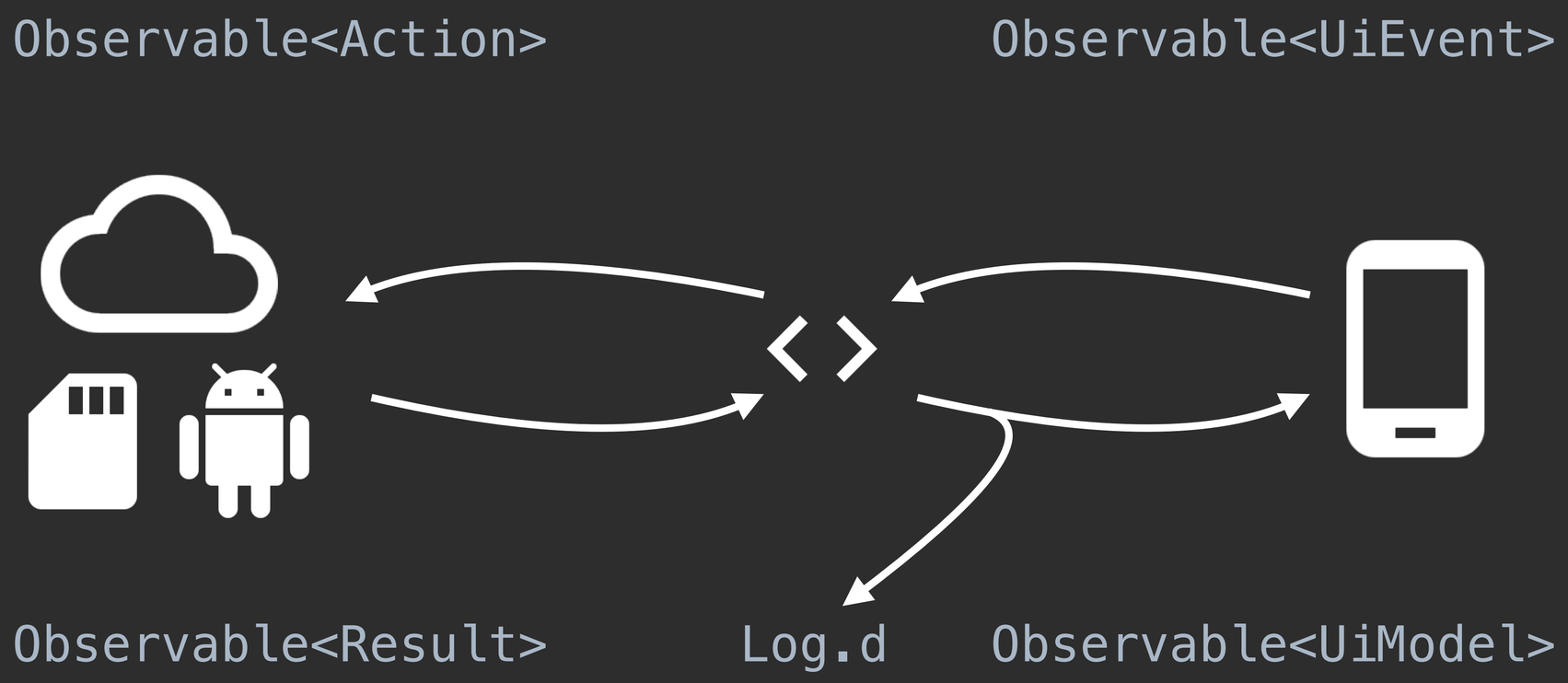
Observable<UiEvent>

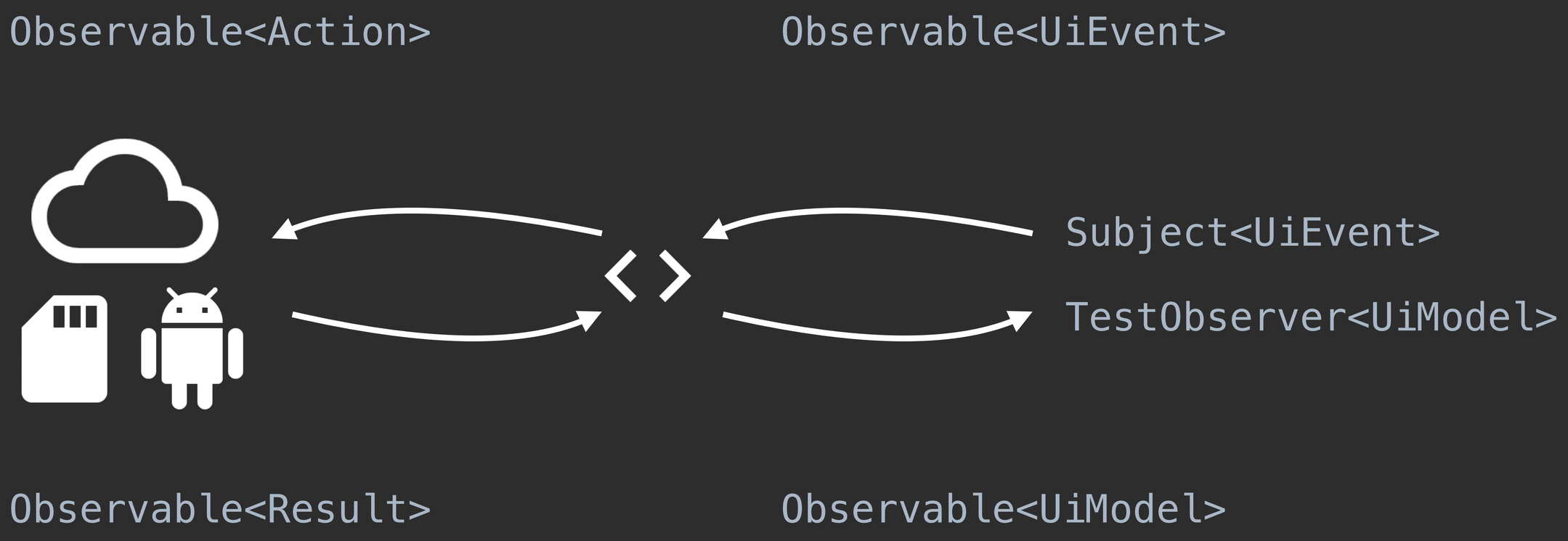


Observable<Result>

Observable<UiModel>

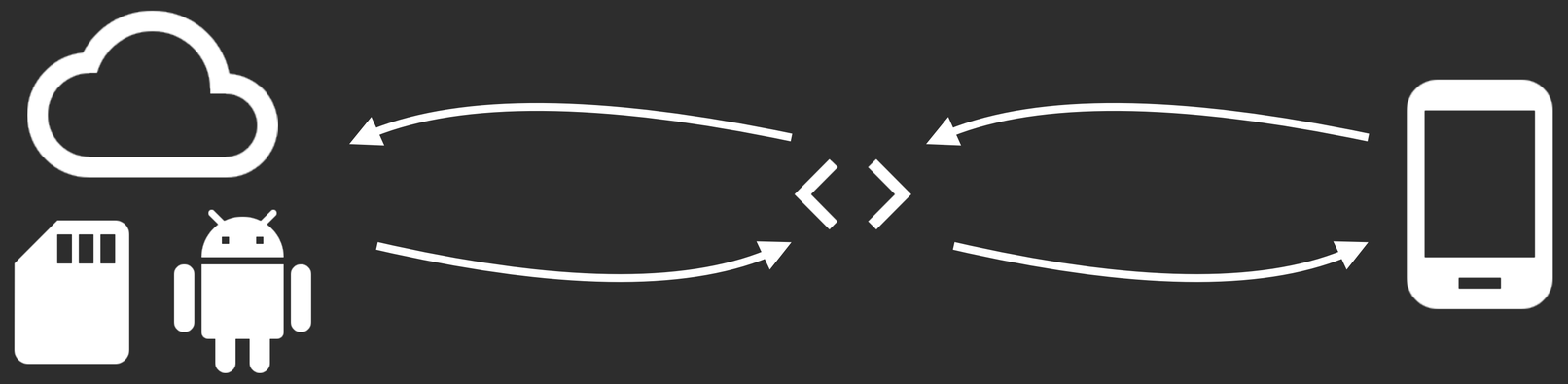

Observable<Action>
Log.d
Observable<UiEvent>
Observable<Result>
Observable<UiModel>


Observable<Action>
Observable<UiEvent>
Observable<Result>
Log.d
Observable<UiModel>


Observable<Action>
Observable<UiEvent>
Subject<UiEvent>
TestObserver<UiModel>
Observable<Result>
Observable<UiModel>

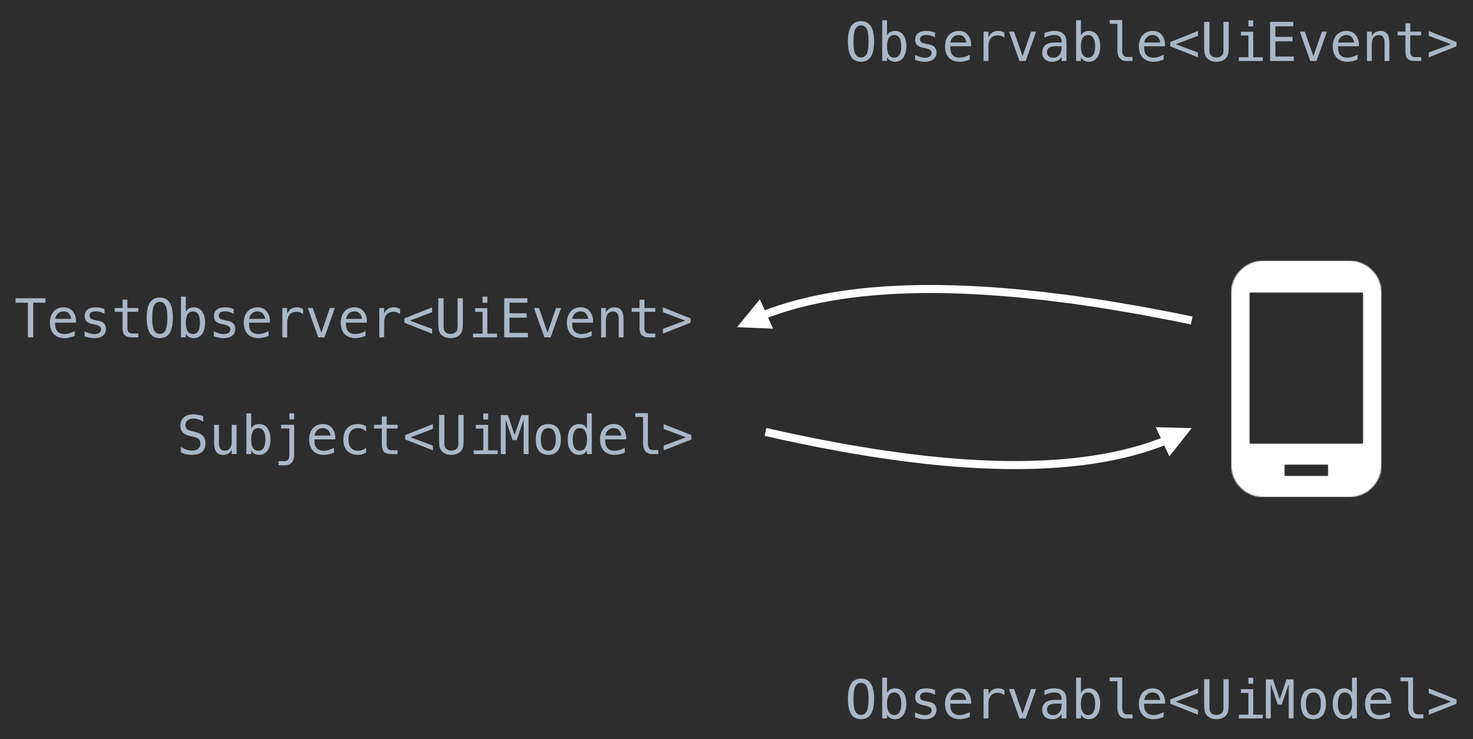
Observable<Action>

Observable<UiEvent>



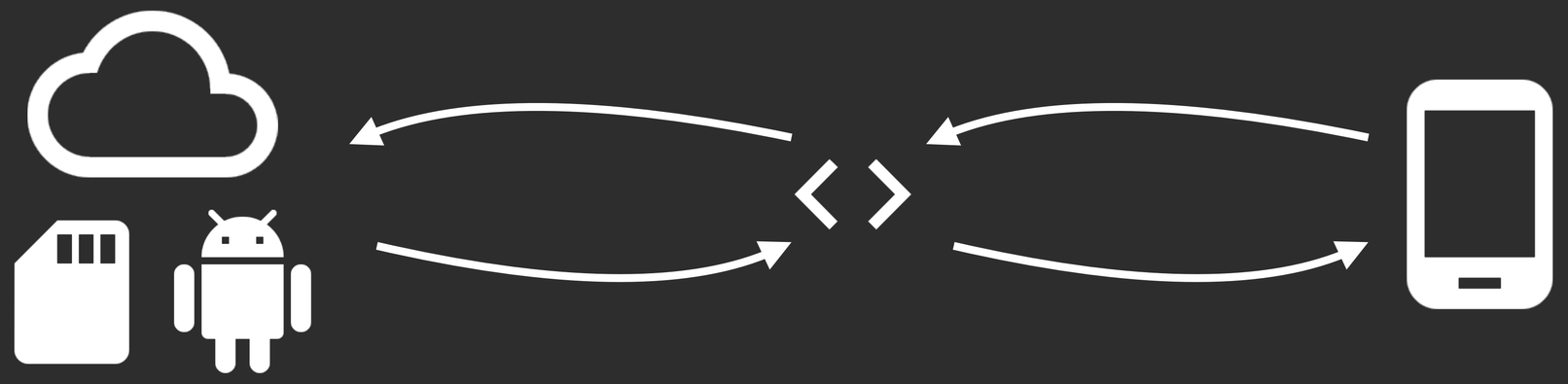
Observable<Result>

Observable<UiModel>


Observable<UiEvent>
TestObserver<UiEvent>
Subject<UiModel>
Observable<UiModel>

Observable<Action>

Observable<UiEvent>



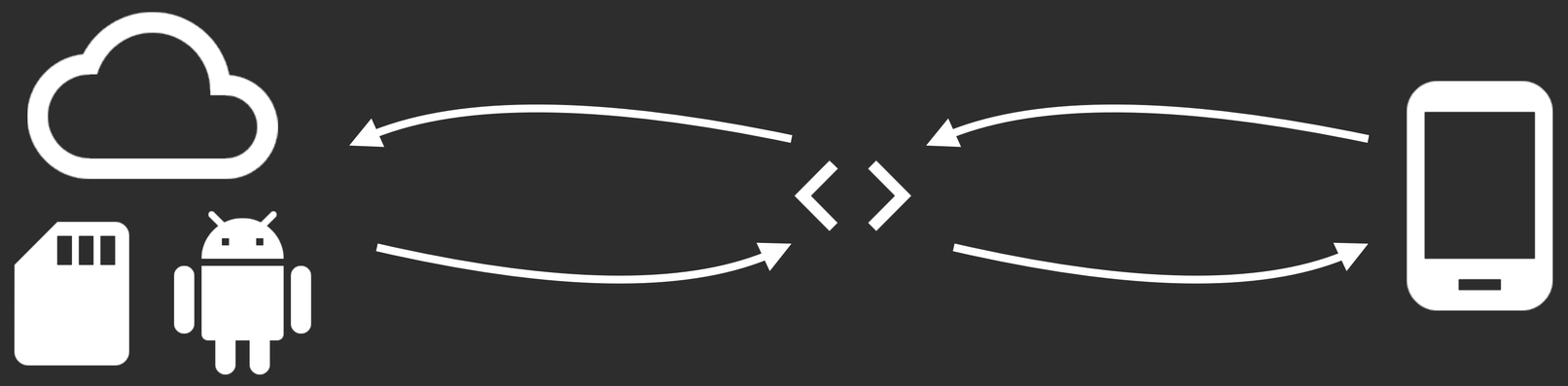
Observable<Result>

Observable<UiModel>

Redux / Cycle

Observable<Action>

Observable<UiEvent>



Observable<Result>

Observable<UiModel>

Managing State with RxJava

twitter.com/ jakewharton

github.com/ jakewharton

jakewharton .com