Testing with Mock Store in NgRx v8

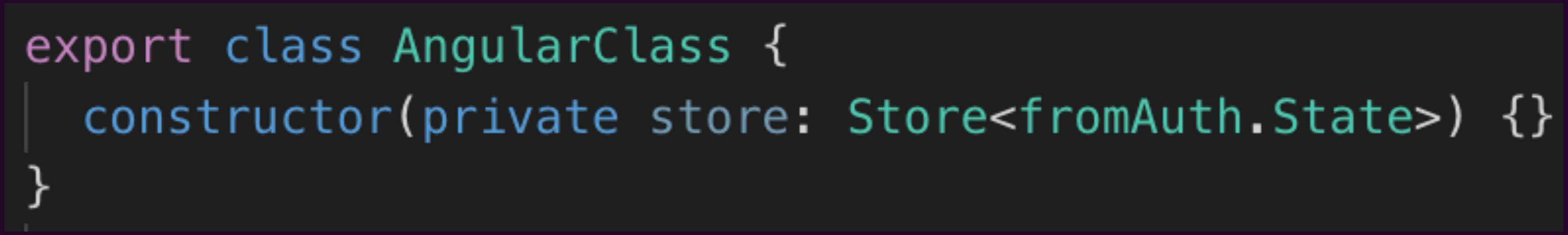
John Crowson

@john_crowson

ANGULAR UP CONFERENCE

Agenda

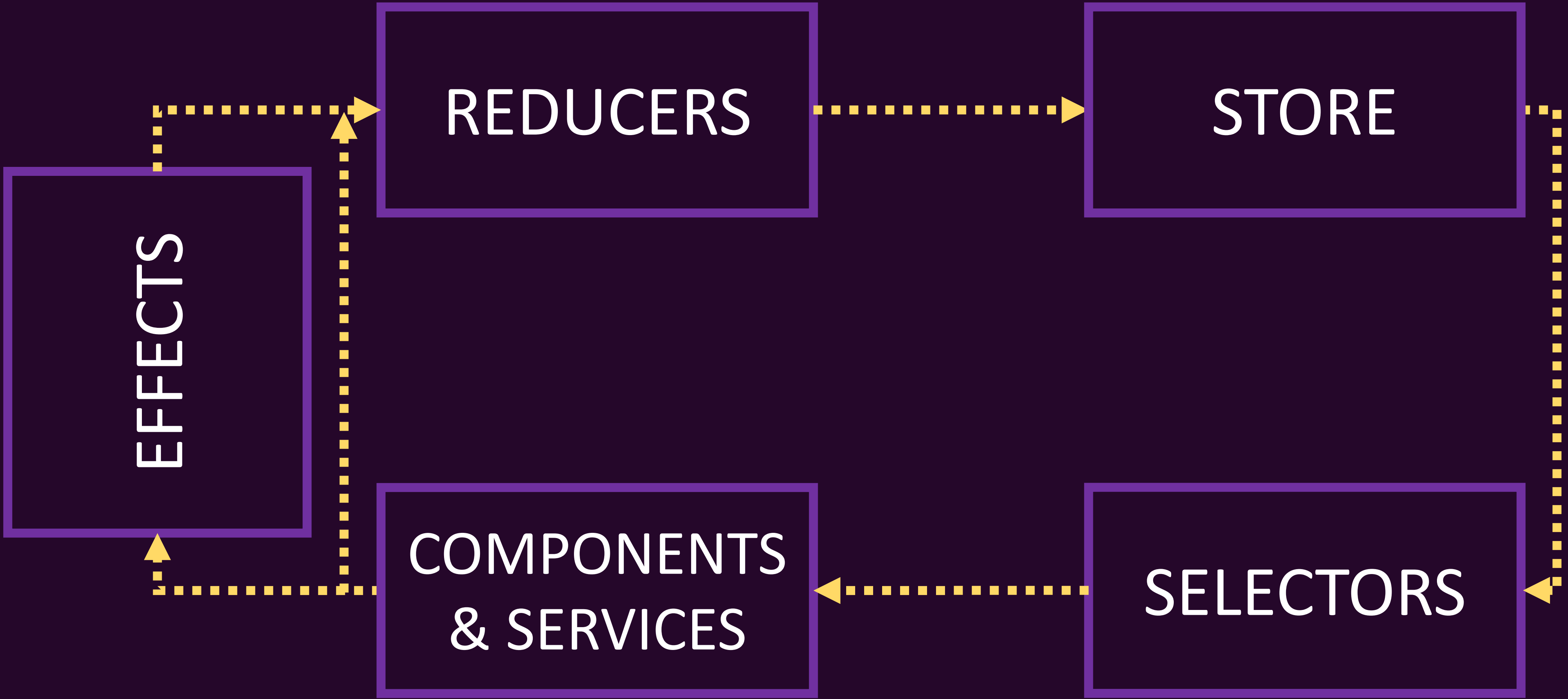
- Testing Angular classes that inject the NgRx store



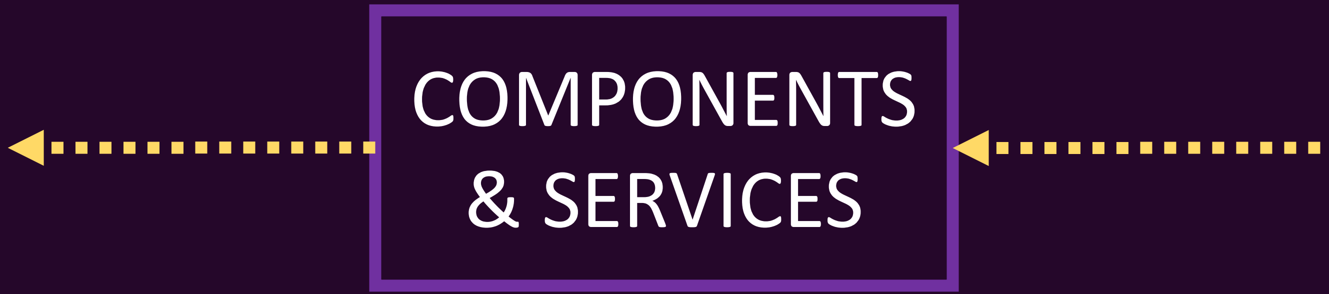
```
export class AngularClass {
  constructor(private store: Store<fromAuth.State>) {}
}
```

- What to Test
- Component Testing
- Unit Testing in NgRx v6
- Improved Unit Testing with MockStore + Mock Selectors
  - Example Component Testing
  - Example Service Testing

NgRx Data Flow



What do we test?



- Actions + payload are dispatched
- Behavior based on a given NgRx state

Component Testing

Integration Test(s)

Assert Interactions of Container Component with Real Dependencies (NgRx: Store).

- Do not mock any related effects, store, or selectors
- Set up using StoreModule and EffectsModule

Unit Test(s)

Assert Behavior of Container Component with Mocked Dependencies (NgRx: Store).

- Mock the store or individual selectors
- But how?

Unit Testing in v6

Goal:

Condition mock state to assert component behavior

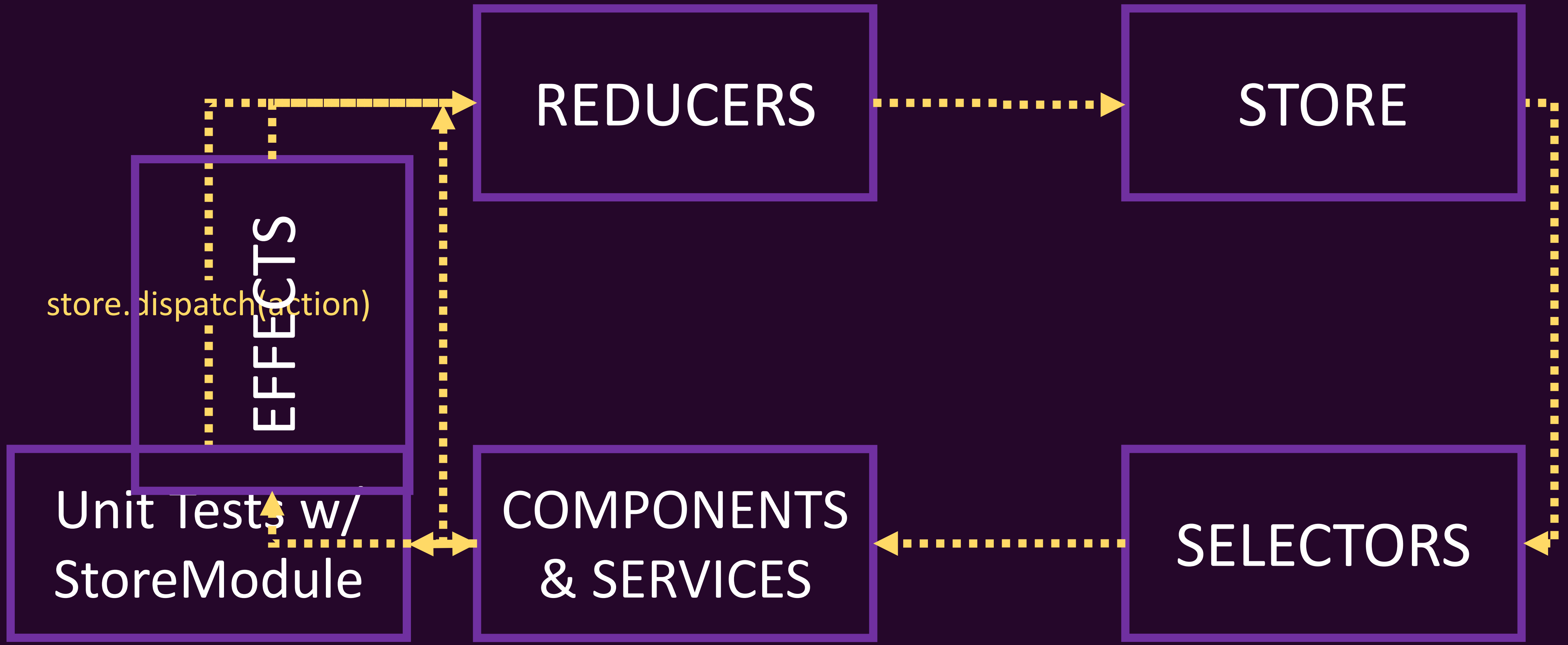
Documentation:

1. Import StoreModule in testing module
2. Dispatch sequence of actions to condition state

Unit Testing in v6

```
TestBed.configureTestingModule({
  // ...
  imports: [ StoreModule.forRoot({ feature: featureReducer }) ]
  // ...
});

it('should display a list of items after the data is loaded', () => {
  store.dispatch(new LoadDataSuccess({ items: [1, 2, 3] }));
  store.dispatch(new LoadOtherDataSuccess({ items: [4, 5, 6] }));
  store.dispatch(new LoadMoreDataSuccess({ items: [7, 8, 9] }));
  // ...

  component.items$.subscribe(data => {
    expect(data.length).toBe(items.length);
  });
});
```

Unit Testing in v6



Unit Testing in v6

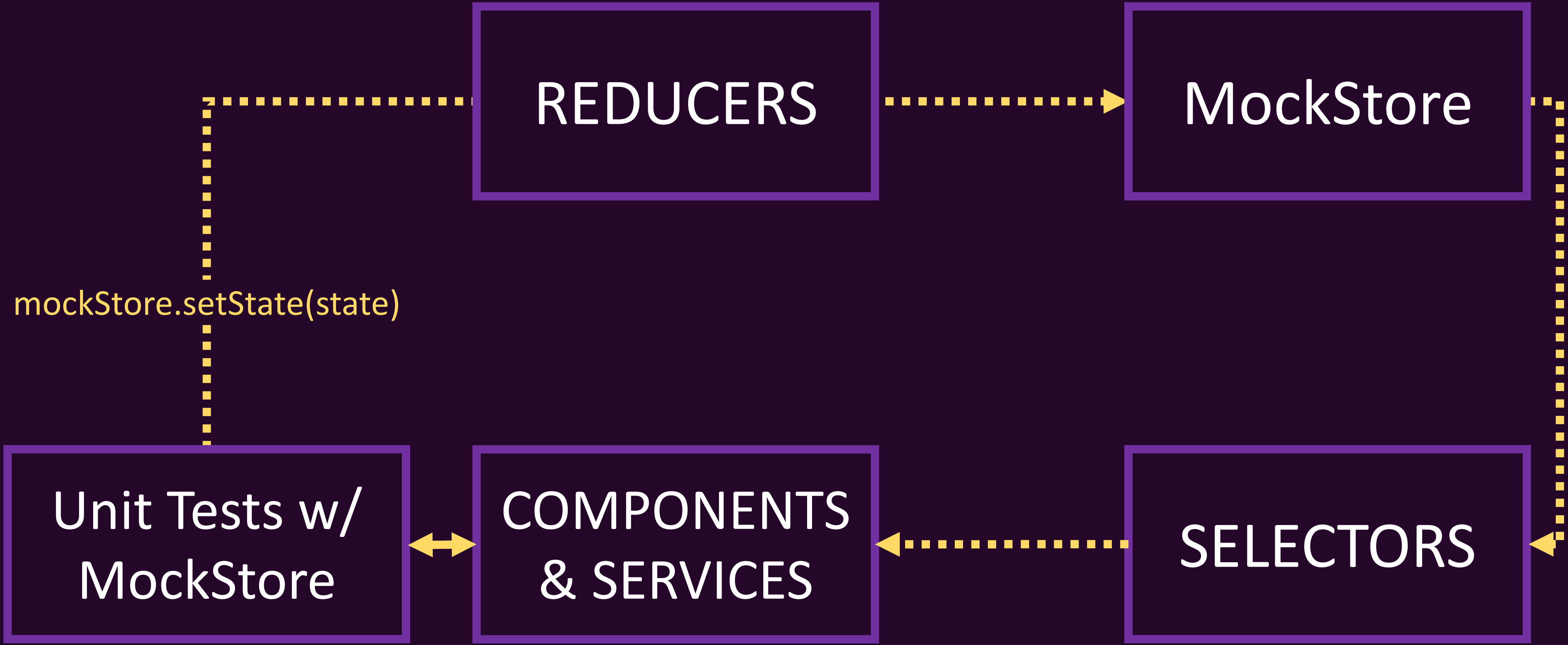
Wanted:

Simple way to mock the NgRx Store

import { MockStore } from '@ngrx/store/testing';

- Released in NgRx v7.0.0 (#1027 - Piotr Staniów)
- Condition a given mock state
- Selectors automatically use it
- Conditioning steps:
  1. Provide provideMockStore({ initialState }) in testing module
  2. Update state with mockStore.setState(state)

Unit Testing with MockStore



Example: Store-dependent Component

```
@Component({
  selector: 'app-cart',
  template: `<p>{{ text$ | async }}</p>`
})
export class CartComponent implements OnInit {
  cartSize$ = this.store.pipe(select(fromCart.getCartSize));
  loading$ = this.store.pipe(select(fromCart.getLoading));

  text$ = combineLatest(this.loading$, this.cartSize$).pipe(
    map(([loading, cartSize]) => {
      if (loading) {
        return 'Loading...';
      } else {
        return 'Cart size: ' + cartSize;
      }
    })
  );

  constructor(private store: Store<fromCart.State>) { }

  ngOnInit() {
    this.store.dispatch(loadCart());
  }
}
```

Test: Cart Loading State:

```
{
  loading: true
  products: [{ id: 5 }]
}
```

Expect: “Loading…”

Test: Cart Loaded State:

```
{
  loading: false,
  products: [{ id: 5 }]
}
```

Expect: “Cart size: 1”

Test: ngOnInit()

Expect: loadCart dispatched

Unit Testing with MockStore

```
import { provideMockStore, MockStore }
  from '@ngrx/store/testing';
// ...
describe('CartComponent', () => {
  let component: CartComponent;
  let mockStore: MockStore<fromCart.State>;
  const loadingState = {
    cart: {
      loading: true,
      products: [{ id: 5 }]
    },
  } as fromOrders.State;

  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [ OrdersComponent ],
      providers: [
        provideMockStore({ initialState: loadingState})
      ],
    }).compileComponents();

    mockStore = TestBed.get(Store);
    initComponent();
  });
```

```
it('should display loading text if the loading'
  + ' selector is true', () => {

  const expected = cold('a', { a: 'Loading...' });

  expect(component.text$).toBeObservable(expected);
});

it('should display the cart size if the loading'
  + ' selector is false', () => {

  const loadedState = {
    cart: {
      loading: false,
      products: [{ id: 5 }]
    },
  } as fromCart.State;
  mockStore.setState(loadedState);
  const expected = cold('a', { a: 'Cart size: 1' });

  expect(component.text$).toBeObservable(expected);
});
```

ANGULAR UP CONFERENCE

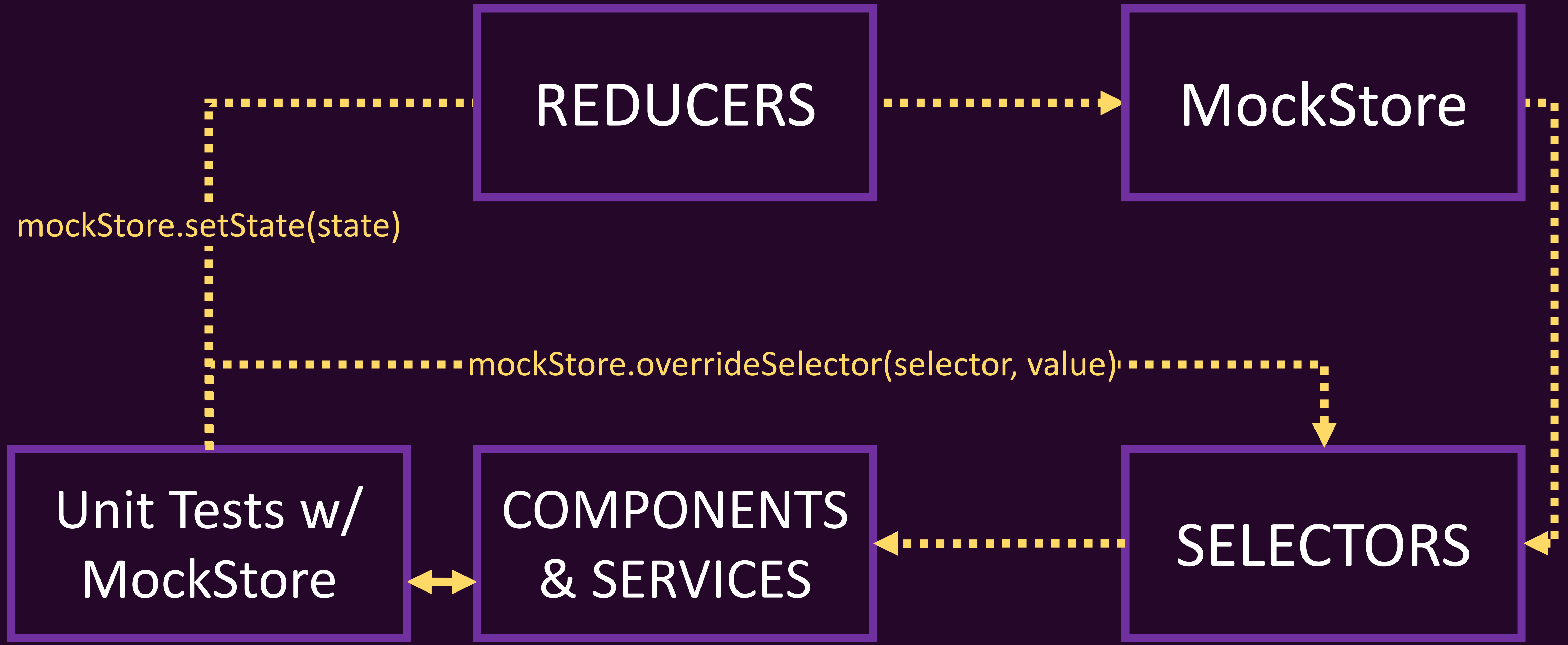
Mock Selectors

- Released in NgRx v8.0.0
  (#1688 – Brandon Roberts, #1836 – John Crowson)
- Condition a mock selector without mocking the entire state
- Test complexity does not increase with selector complexity:

```
export const getPriceAverage = createSelector(
  getPriceProduct1,
  getPriceProduct2,
  getPriceProduct3,
  (price1, price2, price3) =>
    (price1 + price2 + price3) / 3
);
```

- Conditioning steps:
  1. Provide provideMockStore() in testing module
  2. Mock selectors with mockStore.overrideSelector(selector, val)
  3. Update selectors with selector.setResult(val)

Unit Testing with Mock Selectors



Unit Testing with Mock Selectors

```
import { provideMockStore, MockStore }
  from '@ngrx/store/testing';
// ...
describe('CartComponent', () => {
  let component: CartComponent;
  let mockStore: MockStore<fromCart.State>;
  let loading: MemoizedSelector<fromCart.State, boolean>;
  let cartSize: MemoizedSelector<fromCart.State, number>;

  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [ OrdersComponent ],
      providers: [
        provideMockStore()
      ],
    }).compileComponents();
    mockStore = TestBed.get(Store);
    loading = mockStore.overrideSelector(
      fromCart.getLoading,
      true
    );
    cartSize = mockStore.overrideSelector(
      fromCart.getCartSize,
      1
    );
    initComponent();
  });
```

```
it('should display loading text if the loading'
  + ' selector is true', () => {

  const expected = cold('a', { a: 'Loading...' });

  expect(component.text$).toBeObservable(expected);
});

it('should display the cart size if the loading'
  + ' selector is false', () => {

  loading.setResult(false);
  const expected = cold('a', { a: 'Cart size: 1' });

  expect(component.text$).toBeObservable(expected);
});
```

ANGULAR UP CONFERENCE

Selector Support

```
this.store.select('orders');

this.store.select(fromOrders.getOrders);

this.store.pipe(select('orders'));

this.store.pipe(select(fromOrders.getOrders));
```

Verifying Dispatched Actions

Component

```
ngOnInit() {
  this.store.dispatch(loadCart());
}
```

StoreModule
+ jasmine

```
it('should dispatch a loadCart action OnInit', () => {
  spyOn(store, 'dispatch').and.callThrough();
  component.ngOnInit();
  expect(store.dispatch).toHaveBeenCalledTimes(1);
  expect(store.dispatch).toHaveBeenCalledWith(loadCart());
});
```

MockStore

```
it('should dispatch a loadCart action OnInit', () => {
  component.ngOnInit();
  const expected = cold('a', { a: loadCart() });
  expect(store.scannedActions$).toBeObservable(expected);
});
```

Global

provideMockStore({ initialState, selectors })

- Setup MockStore in tests

MockStore

setState(nextState)

- Set state for MockStore

overrideSelector(selector, value): MemoizedSelector

- Override selector with mocked value

resetSelectors()

- Reset overridden selectors

scannedActions$: Observable<Action>

- Get actions dispatched to MockStore

MemoizedSelector

setResult(value)

- Update overridden selector's mock value

Store-dependent Effects

- Incorporate state using withLatestFrom + selector
- Should pass data as action payload when possible
- Potential use case:
  - Router State
  - Auth State

Example: Store-dependent Effect

```
loadCartAuthAlert$ = createEffect(
  () =>
    this.actions$.pipe(
      ofType(loadCart),
      withLatestFrom(
        this.store.pipe(
          select(fromAuth.getLoggedIn)
        )
      ),
      filter(([action, isLoggedIn]) =>
        !isLoggedIn
      ),
      tap(() =>
        window.alert(
          'Please log in'
        )
      )
    ),
  { dispatch: false }
);
```

When loadCart() dispatched...

Test: Logged In State:

```
{
  isLoggedIn: true
}
```

Expect: No alert

Test: Logged Out State:

```
{
  isLoggedIn: false
}
```

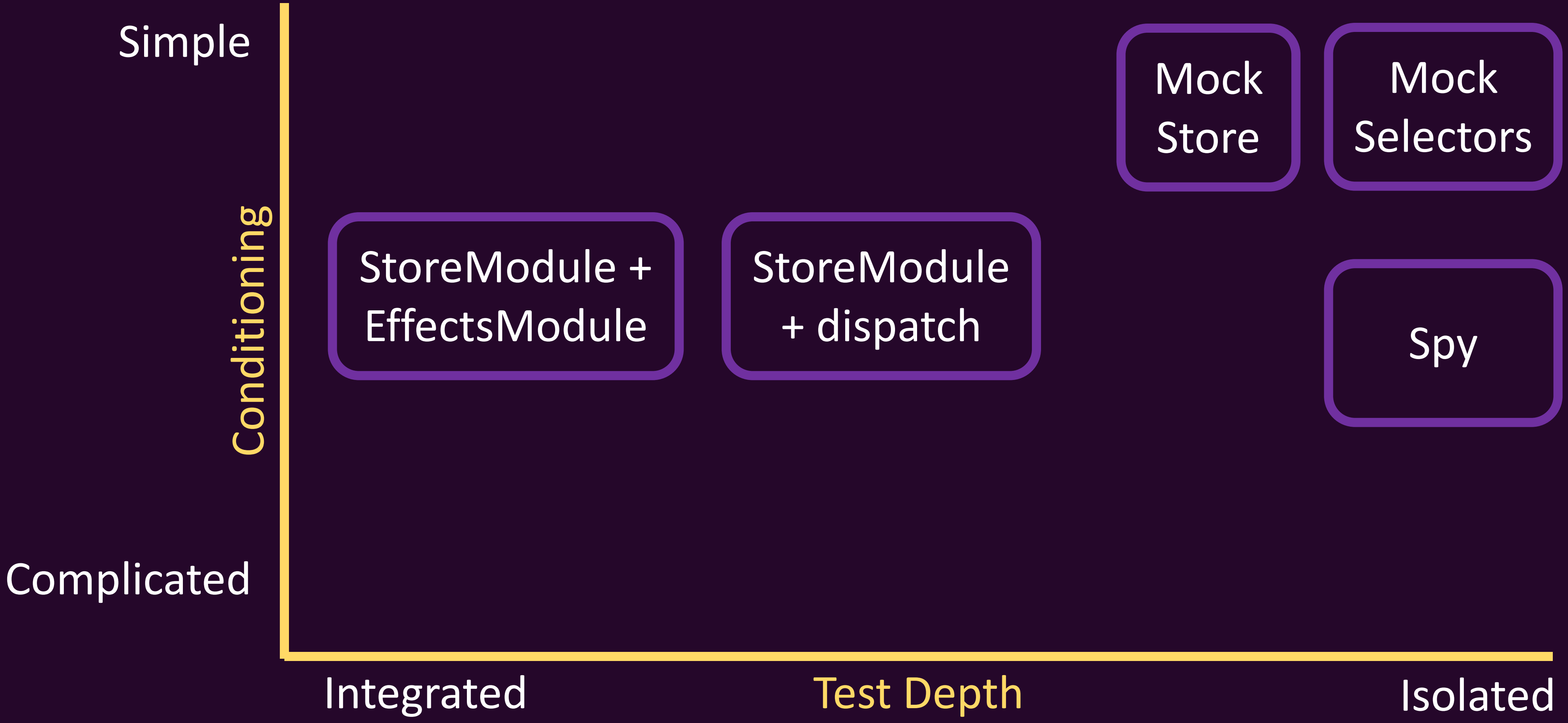
Expect: Alert: “Please log in”

Unit Testing Effect with Mock Selectors

```
import { provideMockStore, MockStore }
  from '@ngrx/store/testing';
// ...
describe('CartEffects', () => {
  let effects: CartEffects;
  let actions$: Observable<any>;
  let mockStore: MockStore<fromAuth.State>;
  let loggedIn: MemoizedSelector<fromAuth.State, boolean>;

  beforeEach(() => {
    TestBed.configureTestingModule({
      providers: [
        OrderEffects,
        provideMockOrderService(),
        provideMockActions(() => actions$),
        provideMockStore()
      ],
    });

    spyOn(window, 'alert');
    effects = TestBed.get(CartEffects);
    actions$ = TestBed.get(Actions);
    store = TestBed.get(Store);

    loggedIn = mockStore.overrideSelector(
      fromAuth.getLoggedIn,
      true
    );
  });
```

```
it('should not alert user if loading cart while'
  + ' logged in', () => {

  const action = loadCart();
  const expected = cold('--');
  actions$ = hot('-a', { a: action });

  expect(effects.loadCartAuthAlert$)
    .toBeObservable(expected);

  expect(window.alert).not.toHaveBeenCalled();
});

it('should alert user if loading cart while'
  + ' not logged in', () => {

  loggedIn.setResult(false);
  const action = loadCart();
  const expected = cold('-c', { c: [action, false] });
  actions$ = hot('-a', { a: action });

  expect(effects.loadCartAuthAlert$)
    .toBeObservable(expected);

  expect(window.alert).toHaveBeenCalledWith(
    'Please log in'
  );
});
```

Conclusion


Simple
Complicated
Conditioning
Mock Store
Mock Selectors
StoreModule + EffectsModule
StoreModule + dispatch
Spy
Integrated
Test Depth
Isolated

What else is new in NgRx?

- Action Creators (v7)

```
// Action Class
export class Login implements Action {
  readonly type = '[Login Page] Login’;
  constructor(
    public payload: { username: string; password: string }
  ) {}
}
const action = new Login({ username: '', password: '' });
// Action Creator (v7)
import { createAction, props } from '@ngrx/store';
// ...
export const login = createAction(
  '[Login Page] Login’,
  props<{ username: string; password: string }>()
);
const action = login({ username: '', password: '' });
```

ANGULAR UP CONFERENCE

What else is new in NgRx?

- Effect Creators (v8)

```
@Effect({ dispatch: false })
loginSuccess$ = this.actions$.pipe(
  ofType(loginSuccess.type),
  tap(() => this.router.navigate(['/']))
);

// Effect Creators
loginSuccess$ = createEffect(
  () =>
    this.actions$.pipe(
    ofType(loginSuccess),
    tap(() => this.router.navigate(['/']))
  ),
  { dispatch: false }
);
```

What else is new in NgRx?

- Reducer Creators (v8)

```
export function reducer(
  state = initialState,
  action: OrdersActions.OrdersActionsUnion
): State {
  switch (action.type) {
    case OrdersActions.loadOrders.type: {
      return {
        ...state,
        error: null,
        loading: true,
        loaded: false
      };
    }
    default: { return state; }
  }
}
```

```
// Reducer Creator
export const reducer = createReducer(
  initialState,
  on(loadOrders, state => ({
    ...state,
    error: null,
    loading: true,
    loaded: false
  }))
);
```

What else is new in NgRx?

- @ngrx/data
  - angular-ngrx-data from John Papa and Ward Bell
  - Extension to simplify entity management
  - Abstracts NgRx for common entity patterns
  - https://ngrx.io/guide/data

Upgrading to NgRx 8

```
$ ng update @ngrx/store
```

References & Thanks

- https://ngrx.io/guide/store/testing
- NgRx Team
- AngularUP
- Questions?
  - @john_crowson

Appendix

Service Testing

```
@Injectable()
export class AuthGuard implements CanActivate {
  constructor(private store: Store<fromAuth.State>) {}

  canActivate(): Observable<boolean> {
    return this.store.pipe(
      select(fromAuth.getLoggedIn),
      take(1)
    );
  }
}
```

Service Testing

```
describe('Auth Guard', () => {
let guard: AuthGuard;
let store: MockStore<fromAuth.State>;
let loggedIn: MemoizedSelector<fromAuth.State, boolean>;

beforeEach(() => {
TestBed.configureTestingModule({
providers: [AuthGuard, provideMockStore()],
});

store = TestBed.get(Store);
guard = TestBed.get(AuthGuard);

loggedIn = store.overrideSelector(
  fromAuth.getLoggedIn,
  false
);
});
```

```
it('should return false if the user state is'
+ ' not logged in', () => {
const expected = cold('(a|)', { a: false });

expect(guard.canActivate())
.toBeObservable(expected);
});

it('should return true if the user state is'
+ ' logged in', () => {
const expected = cold('(a|)', { a: true });

loggedIn.setResult(true);

expect(guard.canActivate())
.toBeObservable(expected);
});
});
```

@john_crowson

ANGULAR UP CONFERENCE