ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Angular Testing

3 - Asynchrony and Mocking

Asynchrony

Potential Problems

- Expects not running
- Timeouts
- Cryptic error messages

Native Approaches

- done callback
- return Promise
- return expect().resolves
- async/await

done

```
it("should test with done", (done) => {
  let a = 1;
  Promise.resolve()
    .then(() => {
      a++;
      expect(a).toBe(1);
    })
    .then(done, done);
});
```

return the Promise

```
it("should return the promise", () => {

  let a = 1;

  return Promise.resolve().then(() => {

    a++;

    expect(a).toBe(2);

  });

});
```

return expect().resolves

```
it("should test with expect.resolves", () => {

  let a = 1;

  const promise = Promise.resolve().then(() => a + 1);

  return expect(promise).resolves.toBe(2);

});
```

Use async/await

```
it("should test with done", async () => {

  let a = 1;

  await Promise.resolve().then(() => {

    a++;

  });

  expect(a).toBe(2);

});
```

Angular-based Approaches

- **waitForAsync**: automatic done callback
- **fakeAsync**: transforms async to sync task
  - flushMicrotasks: run all microtasks
  - tick: move forward in time
  - flush: run all asynchronous tasks (skips periodic timers)

waitForAsync: Automatic done callback

```
test('async', waitForAsync(() => {
    expect.hasAssertions();
    let a = 1;
    Promise.resolve().then(() => {
      a++;
      expect(a).toBe(2);
    });

    window.setTimeout(() => {
      a++;
      expect(a).toBe(3);
    }, 1000);
  })
);
```

fakeAsync: Turn asynchrony into synchrony

```
test("microtasks", fakeAsync(() => {
  let a = 1;
  Promise.resolve().then(() => (a = 2));
  expect(a).toBe(1);


  flushMicrotasks();
  expect(a).toBe(2);
}));
```

fakeAsync

```
test("immediate macrotasks", fakeAsync(() => {
  let a = 1;
  window.setTimeout(() => (a = 2));
  expect(a).toBe(1);


  tick();
  expect(a).toBe(2);
}));
```

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

fakeAsync

```
test("delayed macrotasks", fakeAsync(() => {

  let a = 1;

  window.setTimeout(() => (a = 2), 2000);

  expect(a).toBe(1);



  tick(2000);

  expect(a).toBe(2);

}), 1000);
```

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

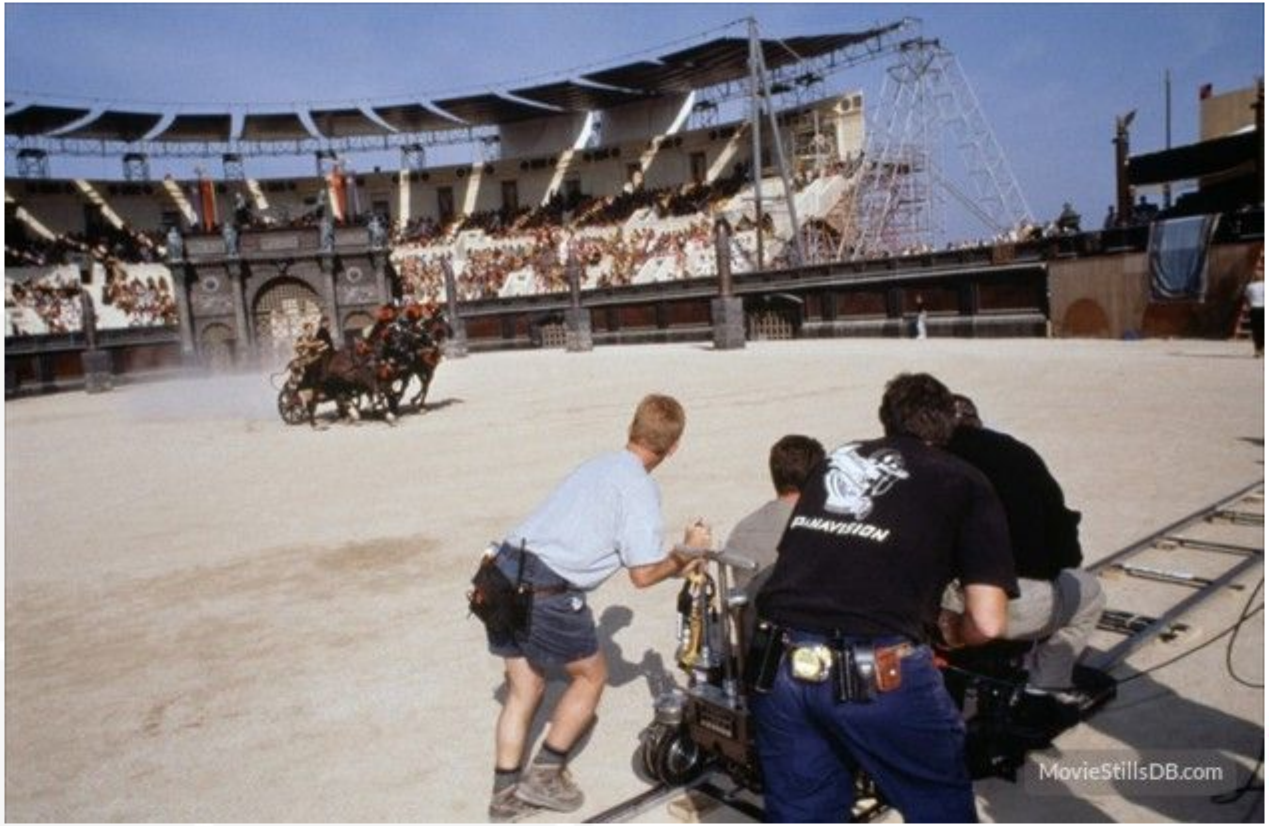
fakeAsync

```
test("delayed macrotasks", fakeAsync(() => {
  let a = 1;
  window.setTimeout(() => (a = 2), 2000);
  expect(a).toBe(1);



  flush();
  expect(a).toBe(2);
}), 1000);
```

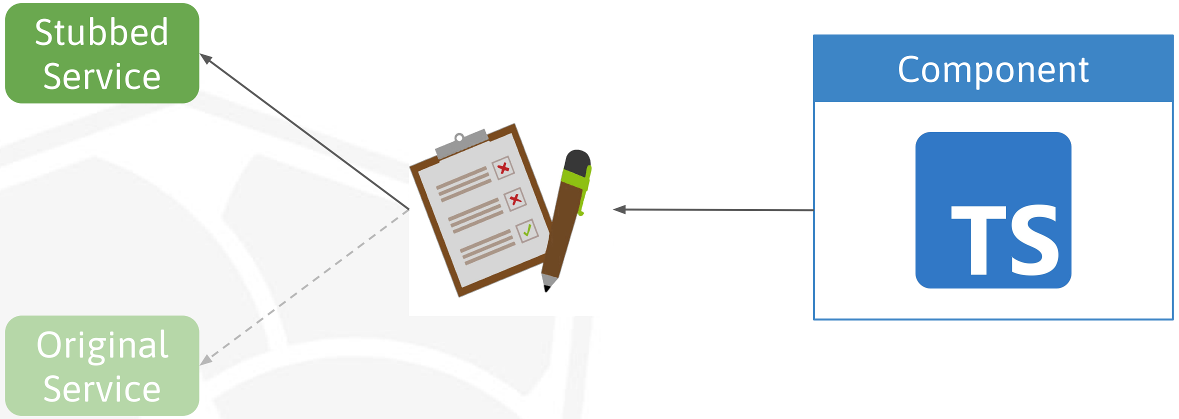
Why fakeAsync over waitForAsync?

1. Run unreachable asynchronous tasks
2. Assertion at the end (AAA pattern)
3. No remaining macrotasks as "implicit" assertion

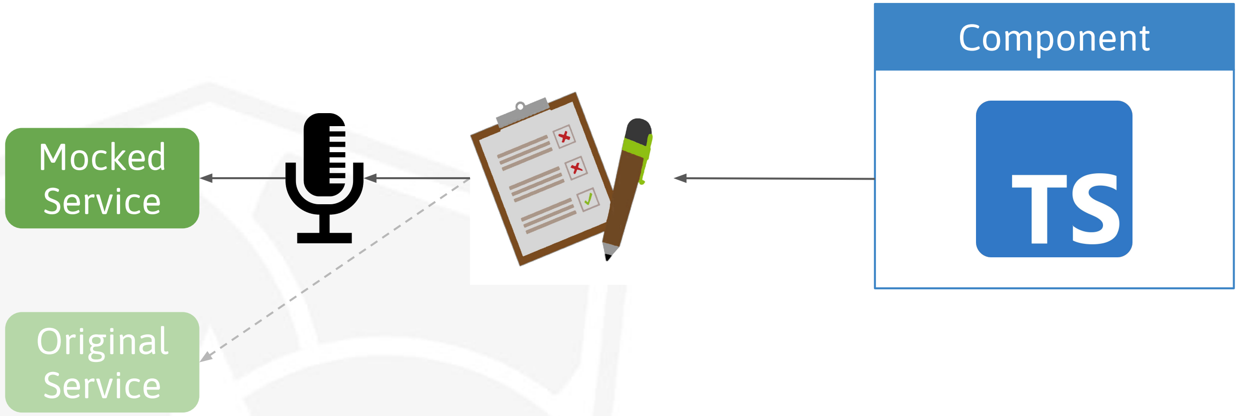
Mocking (Test Doubles)



Stub



Mock



Two Types (Academic)

1. Stub: Replaces a dependency
   a. When dependency returns a value
   b. e.g. HTTP Request
   c. Is enough in most cases
   d. Test doesn't verify the stub is called
2. Mock: Verifies a call to a dependency
   a. A "side-effect only" dependency
   b. Usage has to be verified
   c. e.g. SnackBar, Router navigation
   d. Test verifies the mock is called

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

```
export class ValidAddressLookuper {
  constructor(
    private addresses: () => AddressSource[],
    private addressValidator: AddressValidatorService
  ) {}

  lookup(query: string): boolean {
    return this.addresses()
      .filter((addressSource) => this.addressValidator.isValidAddress(addressSource))
      .some((address) => address.value.startsWith(query));
  }
}
```

Stub

```
it('should stub validator', () => {
  const validator = { isValidAddress: () => true };
  const lookuper = new ValidAddressLookuper(
    () => [
      {
        value: 'Domgasse 5',
        expiryDate: new Date(2000, 0, 1)
      }
    ],
    validator as AddressValidatorService
  );

  expect(lookuper.lookup('Domgasse 5')).toBe(true);
});
```

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

Mocking Functions

return value

arguments of functions

```
const validatorFn = jest.fn<boolean, [AddressSource]>(
  (addressSource) => true
);
```

optional implementation

Mock

```
it('should mock validator', () => {
  const validator = { isValidAddress: jest.fn<boolean, [AddressSource]>(() => true) };
  const lookuper = new ValidAddressLookuper(
    () => [
      {
        value: 'Domgasse 5',
        expiryDate: new Date(2000, 0, 1)
      }
    ],
    validator as AddressValidatorService
  );

  expect(lookuper.lookup('Domgasse 5')).toBe(true);
});
```

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

Mock

```
it('should mock validator', () => {
  const validator = { isValidAddress: jest.fn<boolean, [AddressSource]>(() => true) };
  const lookuper = new ValidAddressLookuper(
    () => [
      {
        value: 'Domgasse 5',
        expiryDate: new Date(2000, 0, 1)
      }
    ],
    validator as AddressValidatorService
  );

  lookuper.lookup('Domgasse 5')

~~  expect(lookuper.lookup('Domgasse 5')).toBe(true);~~
});
```

Mock

```
it('should mock validator', () => {
  const validator = { isValidAddress: jest.fn(() => true) };
  const lookuper = new ValidAddressLookuper(
    () => [
      {
        value: 'Domgasse 5',
        expiryDate: new Date(2000, 0, 1)
      }
    ],
    validator as AddressValidatorService
  );

  lookuper.lookup('Domgasse 5')

  expect(validator.isValidAddress).toBeCalled();
});
```

Mock

```
it('should mock validator', () => {
  const validator = { isValidAddress: jest.fn(() => true) };
  const lookuper = new ValidAddressLookuper(
    () => [
      {
        value: 'Domgasse 5',
        expiryDate: new Date(2000, 0, 1)
      }
    ],
    validator as AddressValidatorService
  );

  lookuper.lookup('Domgasse 5')

  expect(validator.isValidAddress).toBeCalledWith({
    value: 'Domgasse 5',
    expiryDate: new Date(2000, 0, 1)
  });
});
```

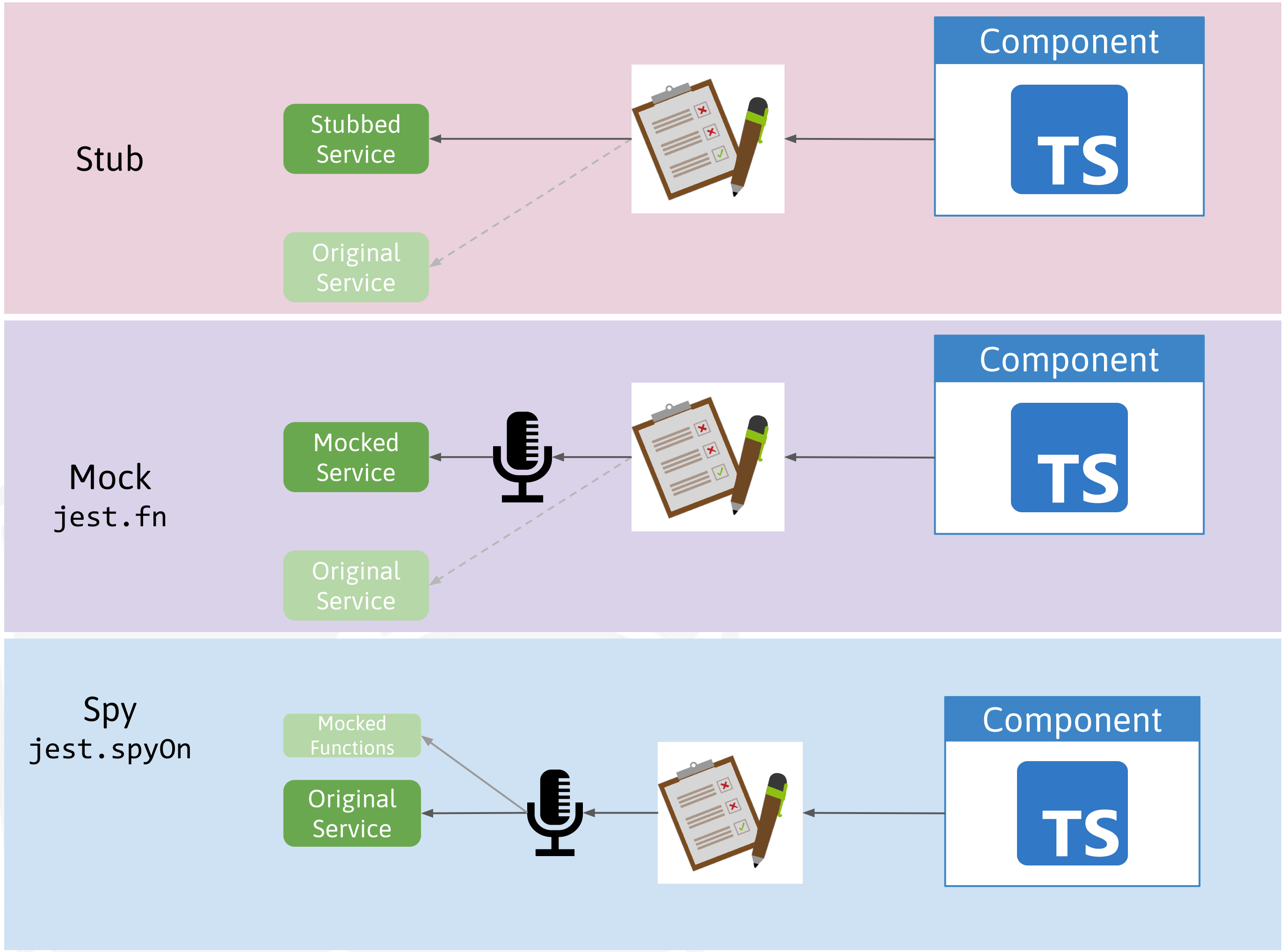
Mock

```
it('should mock validator', () => {
  const validator = { isValidAddress: jest.fn<boolean, [AddressSource]>(() => true) };
  const lookuper = new ValidAddressLookuper(
    () => [
      {
        value: 'Domgasse 5',
        expiryDate: new Date(2000, 0, 1)
      }
    ],
    validator as AddressValidatorService
  );

  lookuper.lookup('Domgasse 5')

  expect(validator.isValidAddress.mock.calls[0][0].value).toBe('Domgasse 5');
});
```

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

Spying

```
it('should check with spied validator', () => {
 const addressValidator = new AddressValidator();
 const validatorSpy = jest.spyOn(addressValidator, 'isValidAddress');
 const addresses = ['Domgasse 15, 1010 Wien'];
 const lookuper = new AddressLookuper(() => addresses, addressValidator);

 lookuper.lookup('Domgasse 15');

 expect(validatorSpy).toHaveBeenCalledWith('Domgasse 15, 1010 Wien');
});
```



ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

Spy = Mocking in Jasmine

- Can be wrapped around a function or property
- Saves history of calls along their arguments
- Returns undefined by default
- Allows to replace implementation (fake) dynamically

Spy Strategy - Behaviours

spy.and.

- **stub**: default
- callThrough: uses original implementation
- fake: uses alternative implementation
- returnValue / returnValues: value or list of values to be returned

Factory methods

- spyOn
  - requires an object
  - attaches spy on one method
- jasmine.createSpy
  - used when dealing with functions
- jasmine.createSpyObj
  - creates an object with multiple spied functions

Spy in Action

```
it('should mock with spyOn', () => {
  const validator = { isValid: (query) => query === 'Domgasse 5' };
  const spy = spyOn(validator, 'isValid');

  expect(validator.isValid('Domgasse 5')).toBeUndefined();
  expect(spy).toHaveBeenCalledWith('Domgasse 5');

  spy.and.callThrough();
  expect(validator.isValid('Domgasse 5')).toBeTrue();

  spy.and.callFake((query) => query === 'Domgasse 15');
  expect(validator.isValid('Domgasse 15')).toBeTrue();
  expect(validator.isValid('Domgasse 5')).toBeFalse();

  spy.and.returnValue(true);
  expect(validator.isValid('unknown')).toBeTrue();
});
```

ANGULAR ARCHITECTS INSIDE KNOWLEDGE

`inject()` & TestBed

- `inject()` as successor to constructor-based DI
- `inject()` cannot be mocked
- `TestBed.inject(`*`Class`*`)` lets Angular instantiate Class
-
```
TestBed.configureTestingModule(
  {providers: [
    {provide: Class, useValue: mock}
  ]}
)
```
  - Overrides default services
  - Is not typesafe

Lab Time