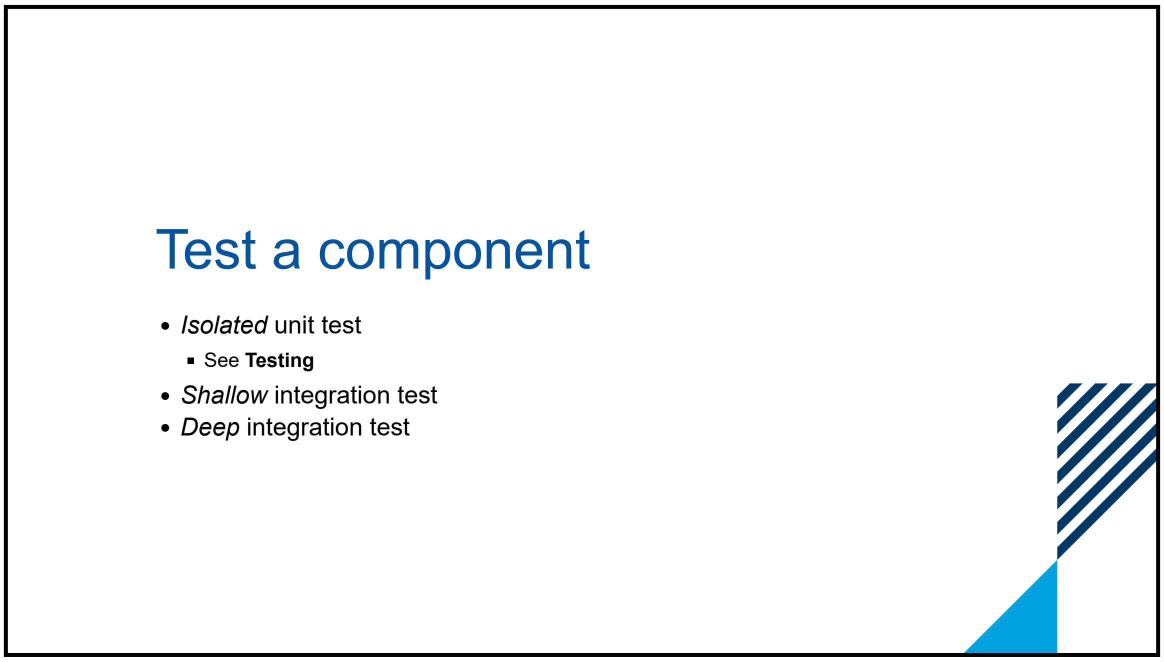
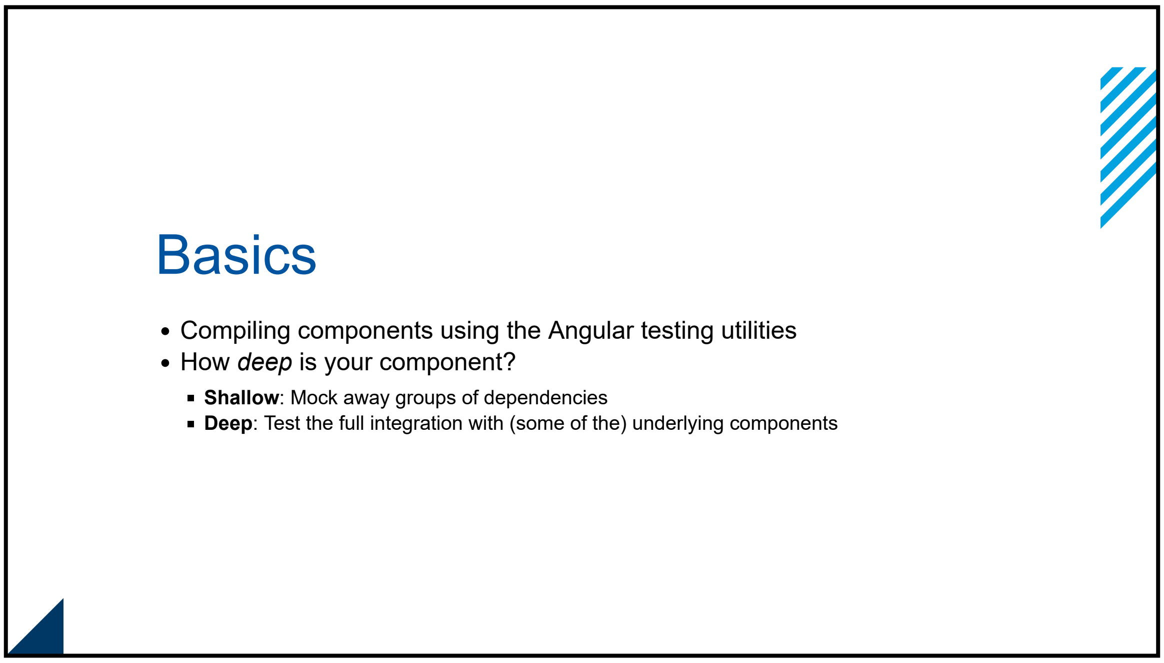
# Integration test



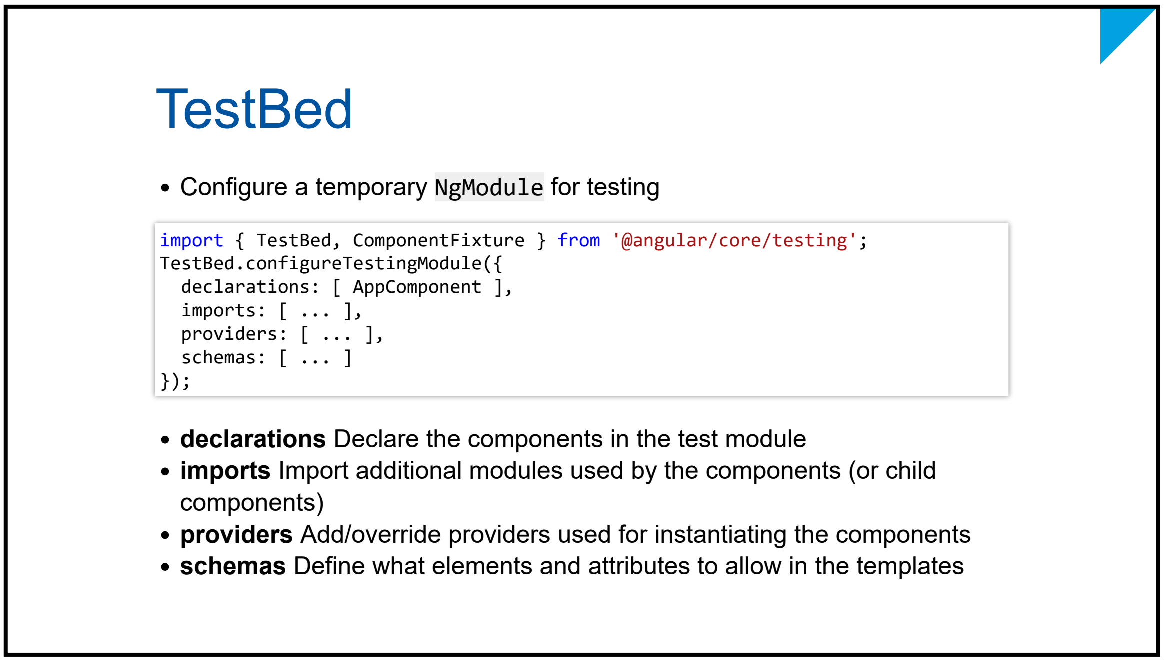
## Test a component

- *Isolated* unit test
  - See **Testing**
- *Shallow* integration test
- *Deep* integration test



## Basics

- Compiling components using the Angular testing utilities
- How *deep* is your component?
  - **Shallow**: Mock away groups of dependencies
  - **Deep**: Test the full integration with (some of the) underlying components

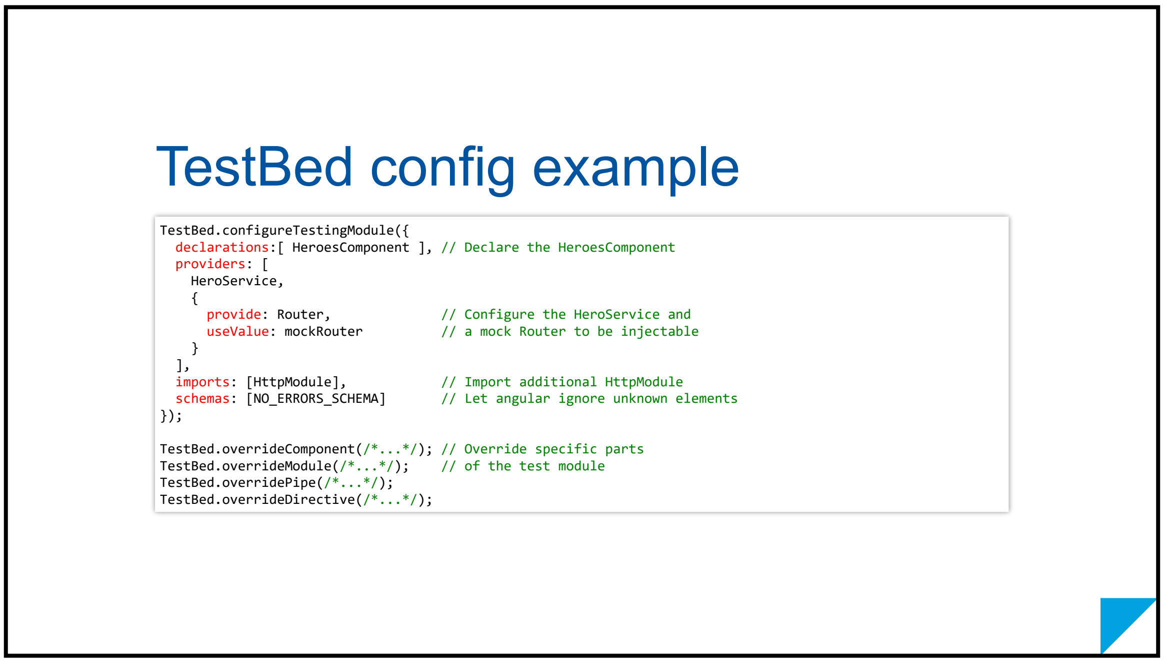


## TestBed

- Configure a temporary `NgModule` for testing

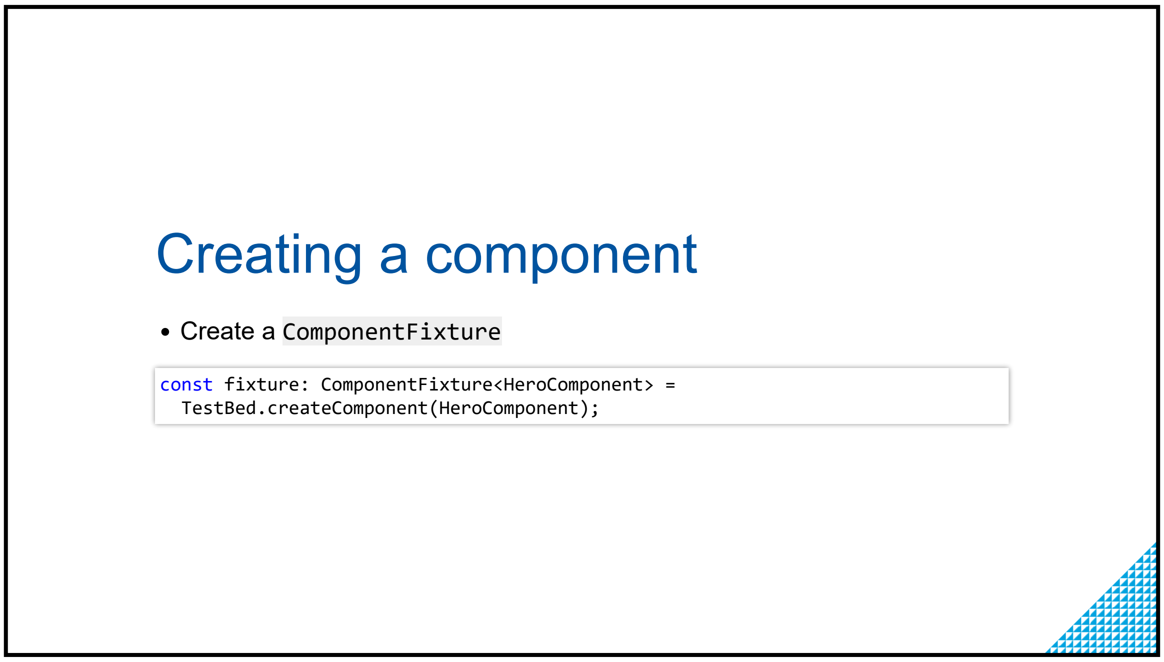
```
import { TestBed, ComponentFixture } from '@angular/core/testing';
TestBed.configureTestingModule({
  declarations: [ AppComponent ],
  imports: [ ... ],
  providers: [ ... ],
  schemas: [ ... ]
});
```

- **declarations** Declare the components in the test module
- **imports** Import additional modules used by the components (or child components)
- **providers** Add/override providers used for instantiating the components
- **schemas** Define what elements and attributes to allow in the templates



## TestBed config example

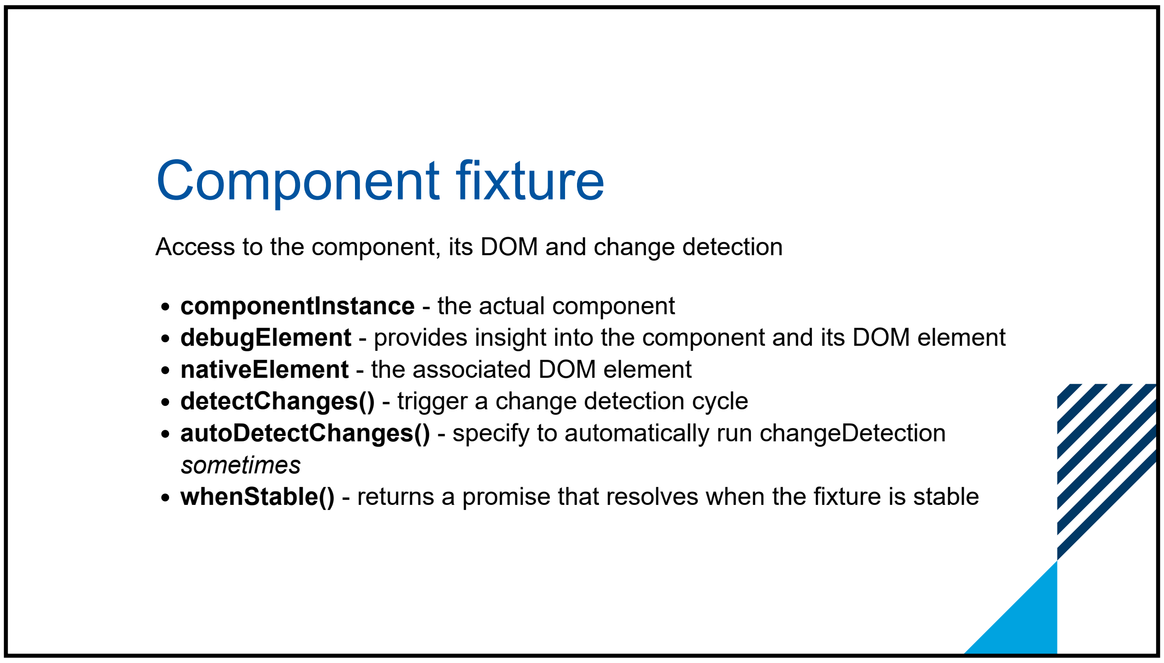
```
TestBed.configureTestingModule({
  declarations:[ HeroesComponent ], // Declare the HeroesComponent
  providers: [
    HeroService,
    {
      provide: Router,              // Configure the HeroService and
      useValue: mockRouter          // a mock Router to be injectable
    }
  ],
  imports: [HttpModule],            // Import additional HttpModule
  schemas: [NO_ERRORS_SCHEMA]       // Let angular ignore unknown elements
});

TestBed.overrideComponent(/*...*/); // Override specific parts
TestBed.overrideModule(/*...*/);    // of the test module
TestBed.overridePipe(/*...*/);
TestBed.overrideDirective(/*...*/);
```



## Creating a component

- Create a `ComponentFixture`

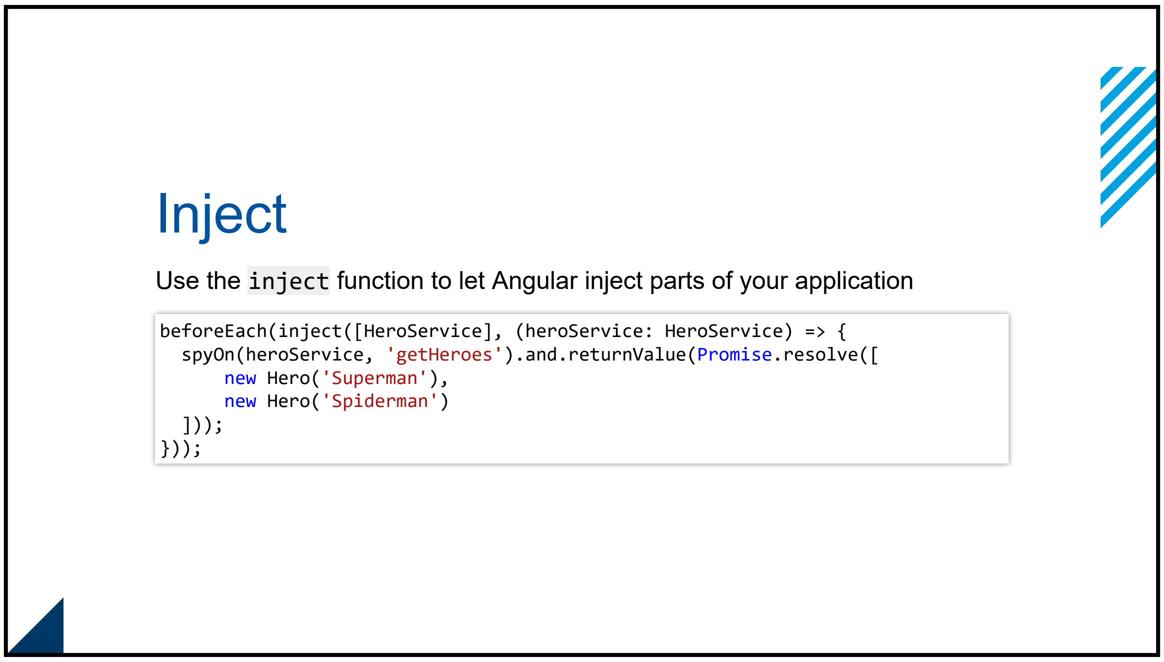
```
const fixture: ComponentFixture<HeroComponent> =
  TestBed.createComponent(HeroComponent);
```



## Component fixture

Access to the component, its DOM and change detection

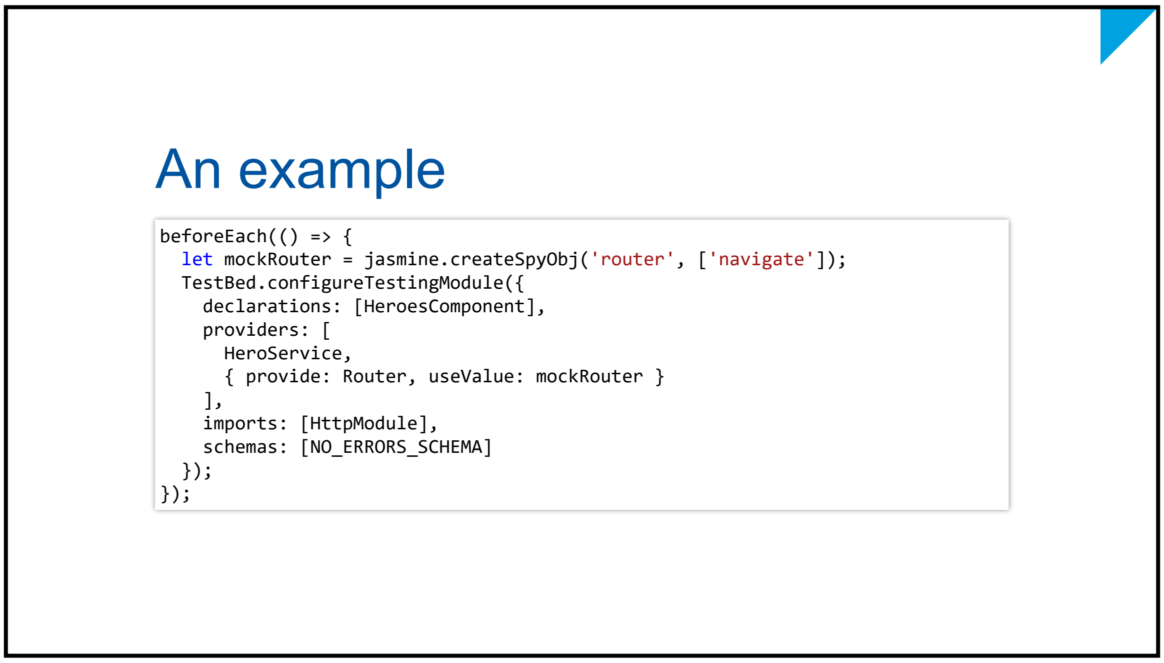
- **componentInstance** - the actual component
- **debugElement** - provides insight into the component and its DOM element
- **nativeElement** - the associated DOM element
- **detectChanges()** - trigger a change detection cycle
- **autoDetectChanges()** - specify to automatically run changeDetection *sometimes*
- **whenStable()** - returns a promise that resolves when the fixture is stable



## Inject

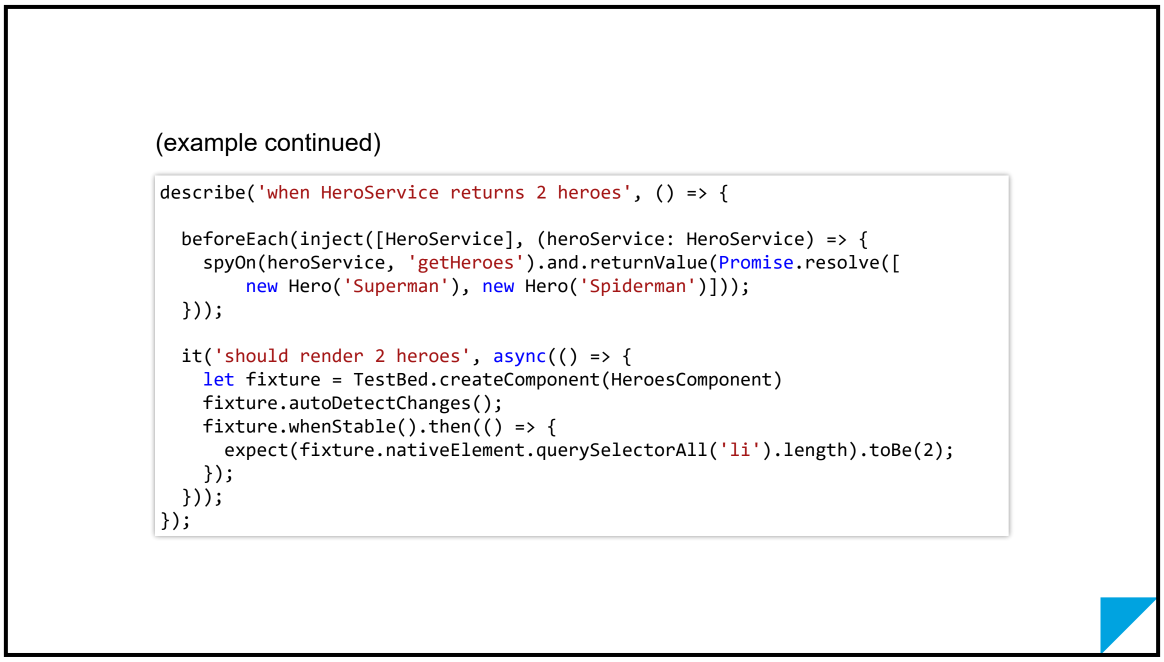
Use the `inject` function to let Angular inject parts of your application

```
beforeEach(inject([HeroService], (heroService: HeroService) => {
  spyOn(heroService, 'getHeroes').and.returnValue(Promise.resolve([
      new Hero('Superman'),
      new Hero('Spiderman')
  ]));
}));
```



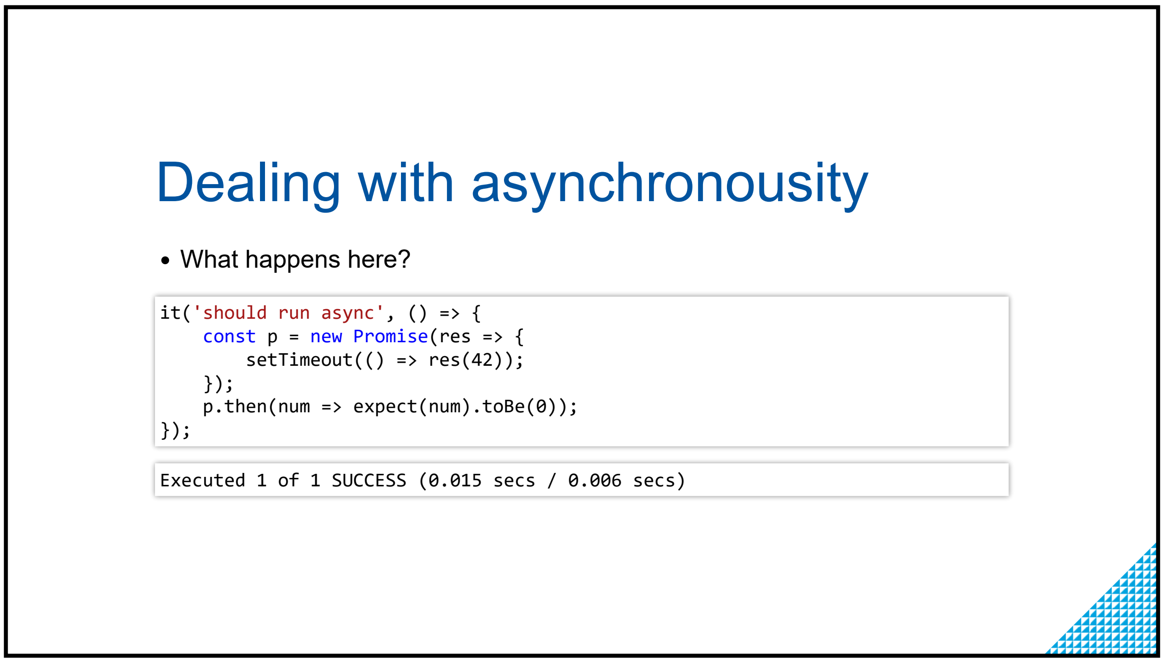
## An example

```
beforeEach(() => {
  let mockRouter = jasmine.createSpyObj('router', ['navigate']);
  TestBed.configureTestingModule({
    declarations: [HeroesComponent],
    providers: [
      HeroService,
      { provide: Router, useValue: mockRouter }
    ],
    imports: [HttpModule],
    schemas: [NO_ERRORS_SCHEMA]
  });
});
```



(example continued)

```
describe('when HeroService returns 2 heroes', () => {

  beforeEach(inject([HeroService], (heroService: HeroService) => {
    spyOn(heroService, 'getHeroes').and.returnValue(Promise.resolve([
        new Hero('Superman'), new Hero('Spiderman')]));
  }));

  it('should render 2 heroes', async(() => {
    let fixture = TestBed.createComponent(HeroesComponent)
    fixture.autoDetectChanges();
    fixture.whenStable().then(() => {
      expect(fixture.nativeElement.querySelectorAll('li').length).toBe(2);
    });
  }));
});
```

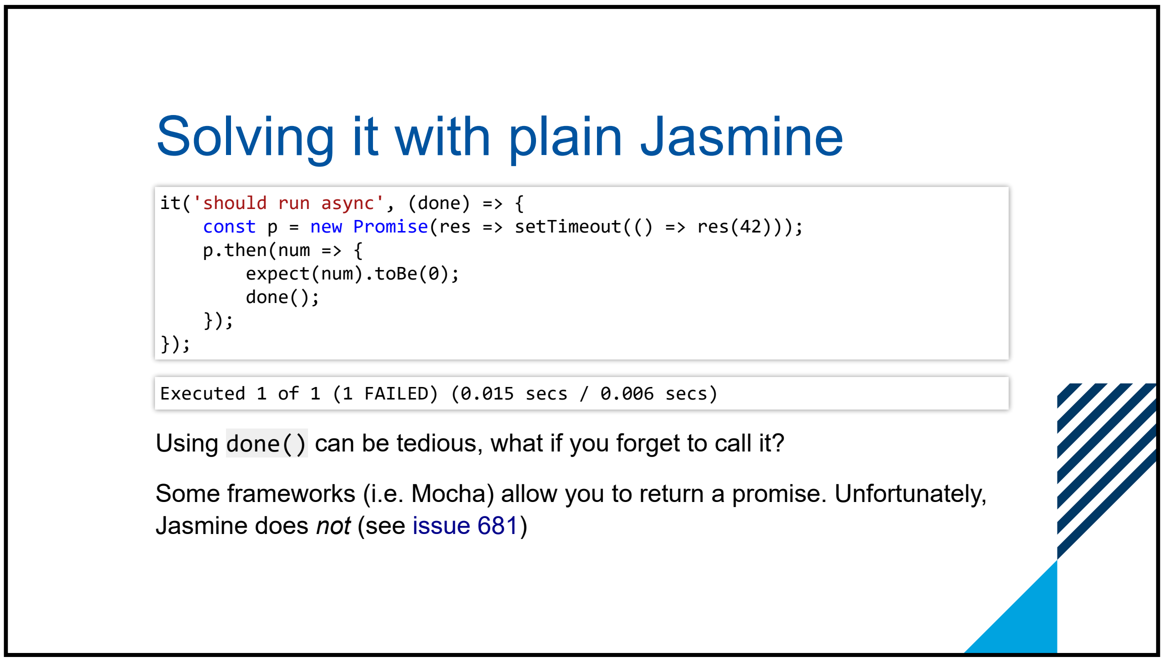


## Dealing with asynchronousity

- What happens here?

```
it('should run async', () => {
    const p = new Promise(res => {
        setTimeout(() => res(42));
    });
    p.then(num => expect(num).toBe(0));
});
```

```
Executed 1 of 1 SUCCESS (0.015 secs / 0.006 secs)
```



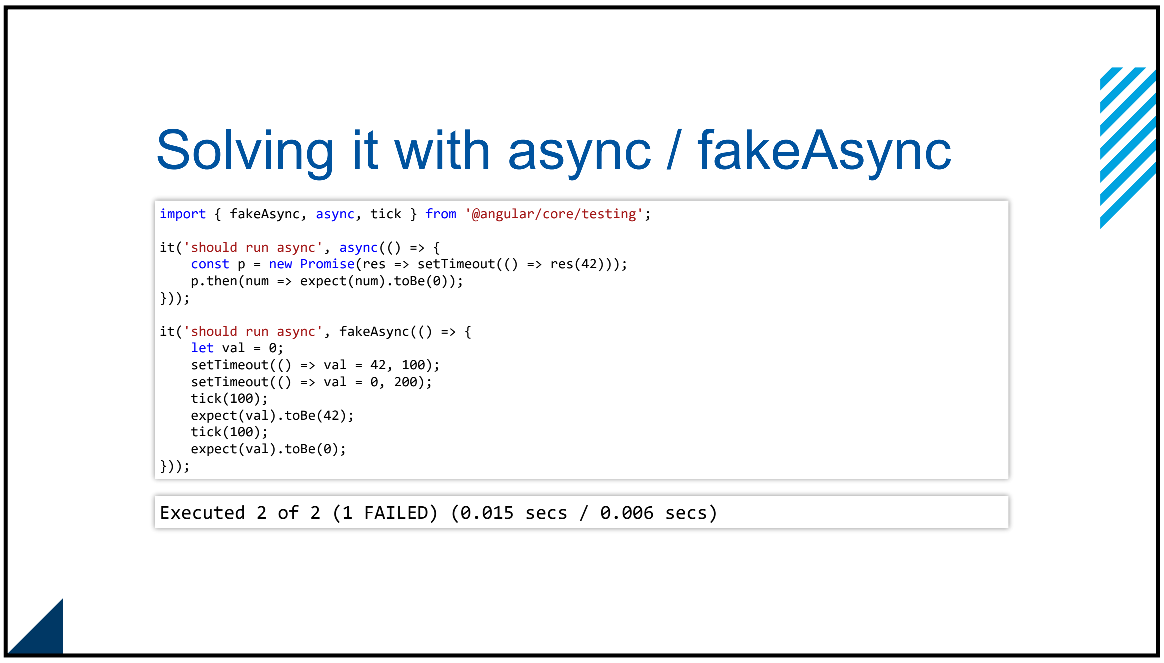
## Solving it with plain Jasmine

```
it('should run async', (done) => {
    const p = new Promise(res => setTimeout(() => res(42)));
    p.then(num => {
        expect(num).toBe(0);
        done();
    });
});
```

```
Executed 1 of 1 (1 FAILED) (0.015 secs / 0.006 secs)
```

Using `done()` can be tedious, what if you forget to call it?

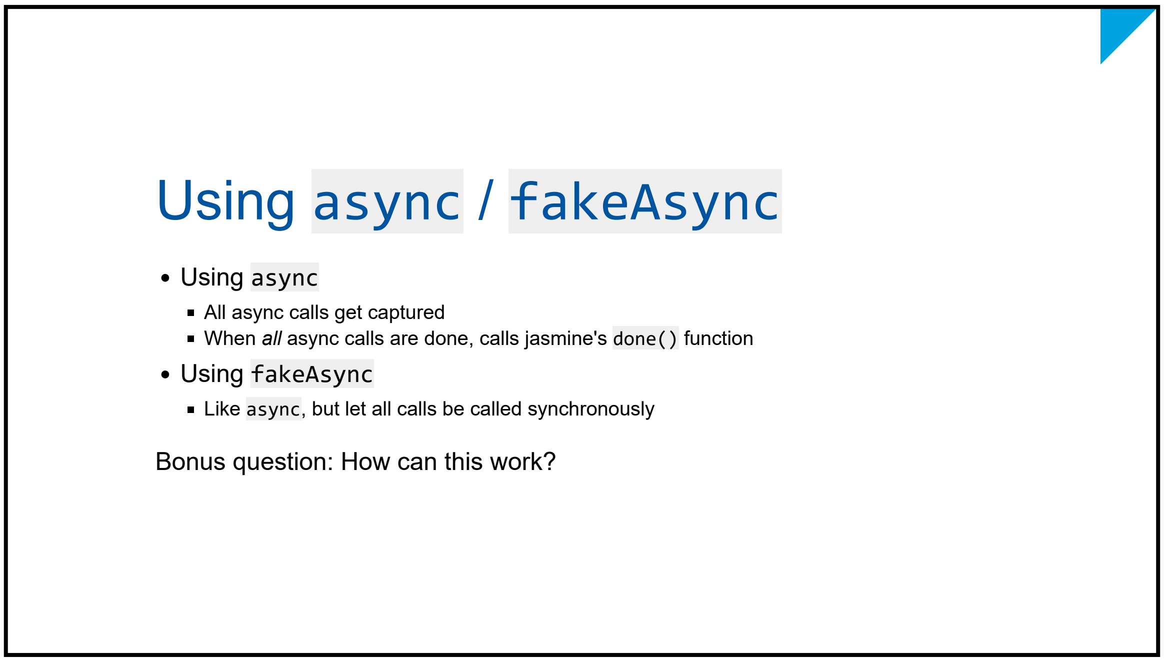
Some frameworks (i.e. Mocha) allow you to return a promise. Unfortunately, Jasmine does *not* (see issue 681)



## Solving it with async / fakeAsync

```
import { fakeAsync, async, tick } from '@angular/core/testing';

it('should run async', async(() => {
    const p = new Promise(res => setTimeout(() => res(42)));
    p.then(num => expect(num).toBe(0));
}));

it('should run async', fakeAsync(() => {
    let val = 0;
    setTimeout(() => val = 42, 100);
    setTimeout(() => val = 0, 200);
    tick(100);
    expect(val).toBe(42);
    tick(100);
    expect(val).toBe(0);
}));
```

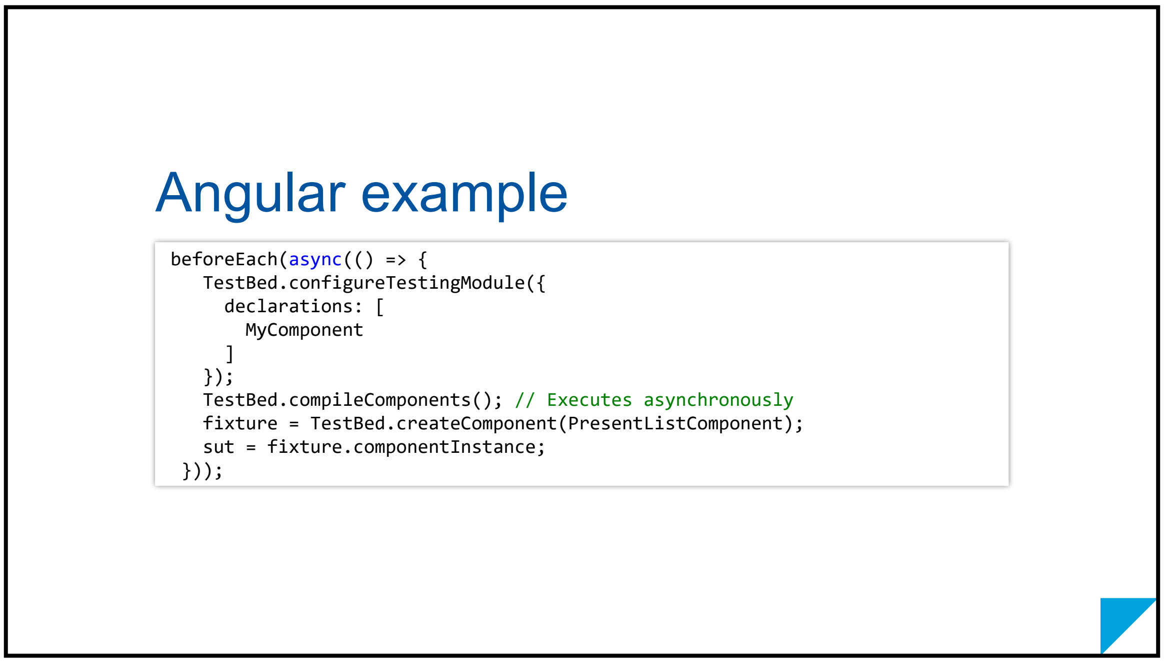
```
Executed 2 of 2 (1 FAILED) (0.015 secs / 0.006 secs)
```



## Using `async` / `fakeAsync`

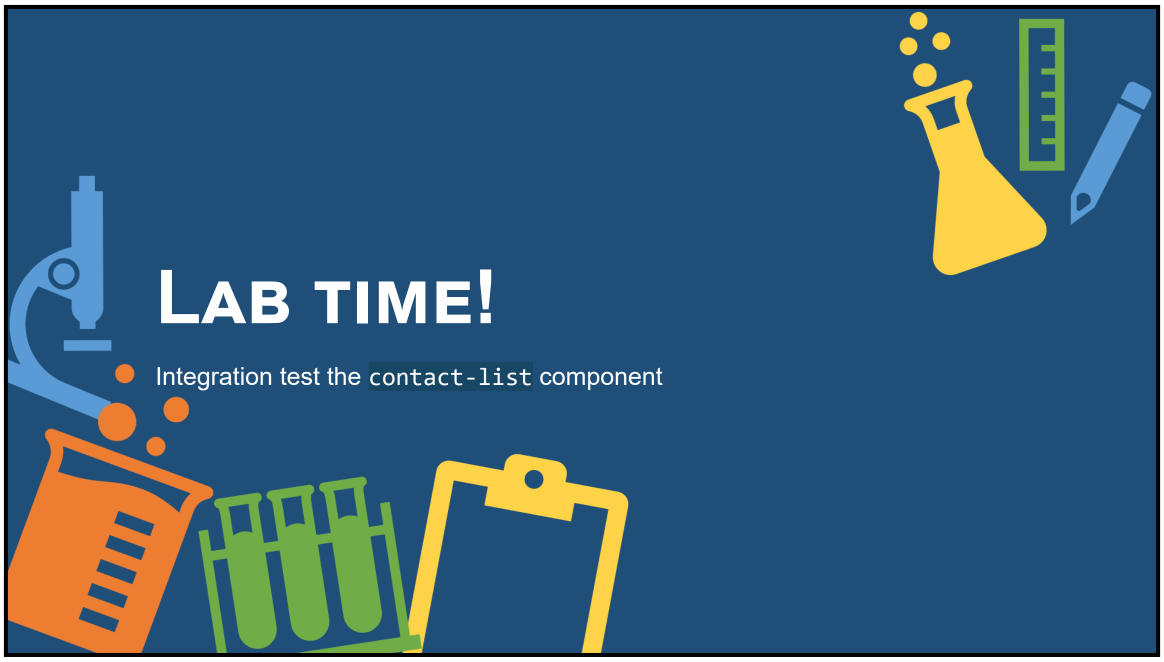
- Using `async`
  - All async calls get captured
  - When *all* async calls are done, calls jasmine's `done()` function
- Using `fakeAsync`
  - Like `async`, but let all calls be called synchronously

Bonus question: How can this work?



## Angular example

```
beforeEach(async(() => {
   TestBed.configureTestingModule({
     declarations: [
       MyComponent
     ]
   });
   TestBed.compileComponents(); // Executes asynchronously
   fixture = TestBed.createComponent(PresentListComponent);
   sut = fixture.componentInstance;
 }));
```



## LAB TIME!

Integration test the `contact-list` component