ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Angular Testing

5 - Testing Strategies

Two Competing Schools of Unit Testing

Differences

London

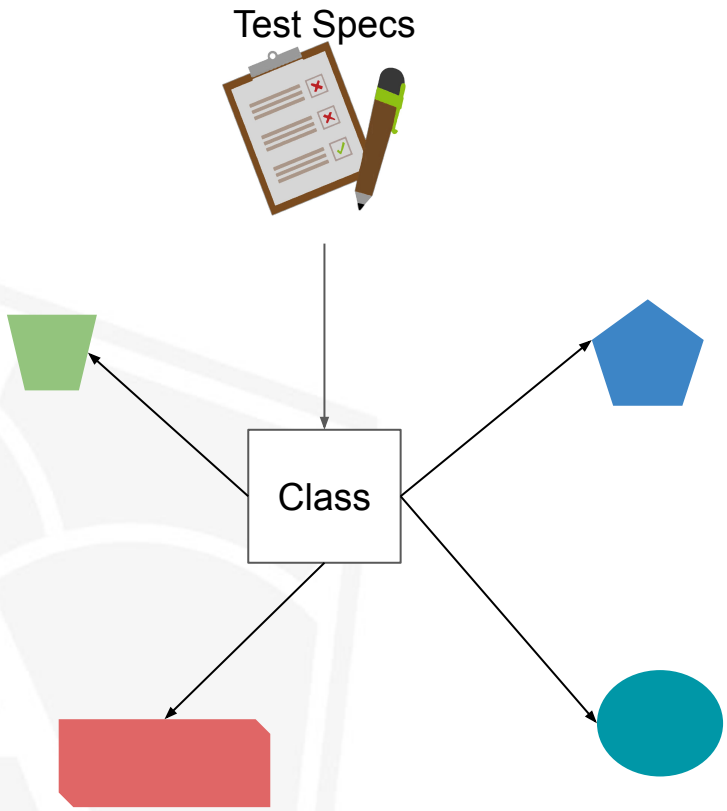
- Unit is a **class**
- Mock everything except the class
  - Very tightly coupled to implementation
- Disadvantages
  - No refactoring
  - Lots of code for mocking
  - No interplay testing
- Advantages
  - Edge cases, finding bugs, exploratory
  - Great code quality (FP)
  - Fast

Detroit (Chicago)

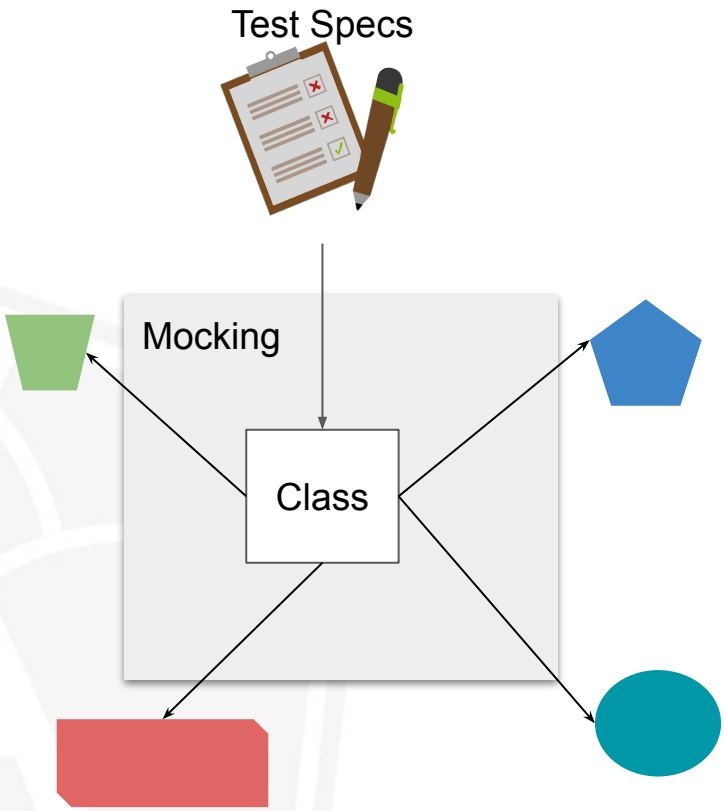
- Unit is a **behaviour**
- Mock out-of-system dependencies
  - Runs against an API (UI)
- Advantages
  - Great for refactoring
  - Efficient (coverage)
- Disadvantages
  - Large setup required
  - Slow

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

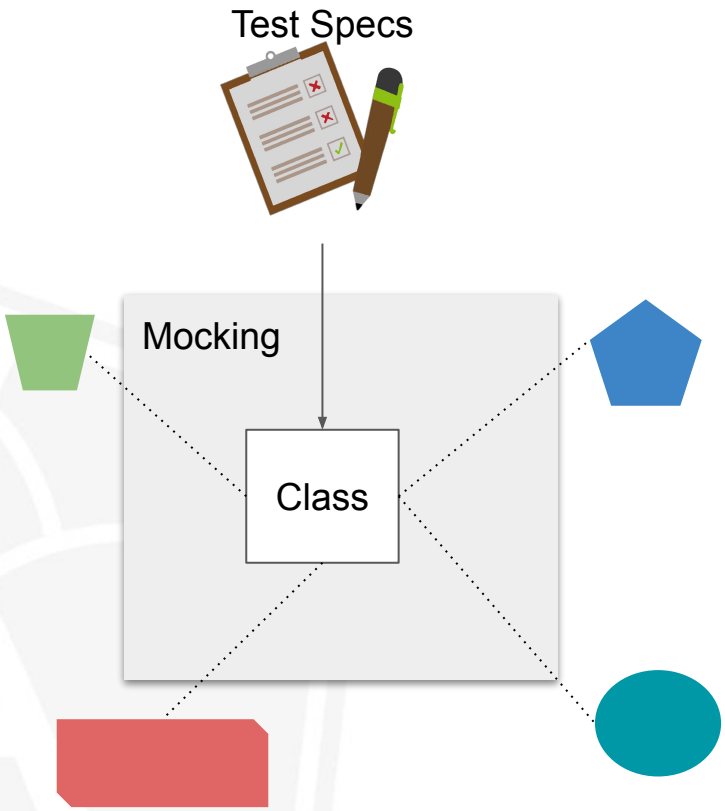
London Style



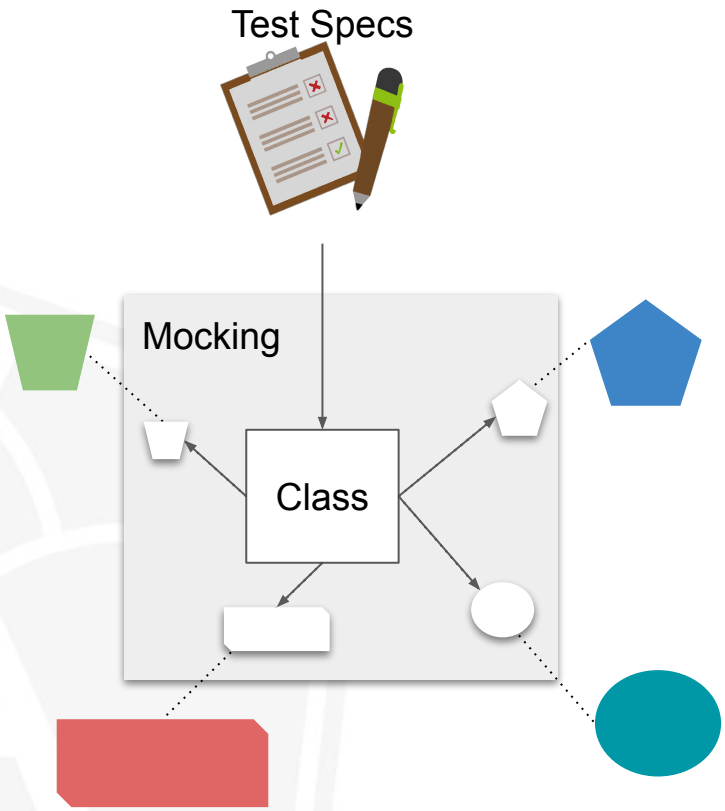
London Style



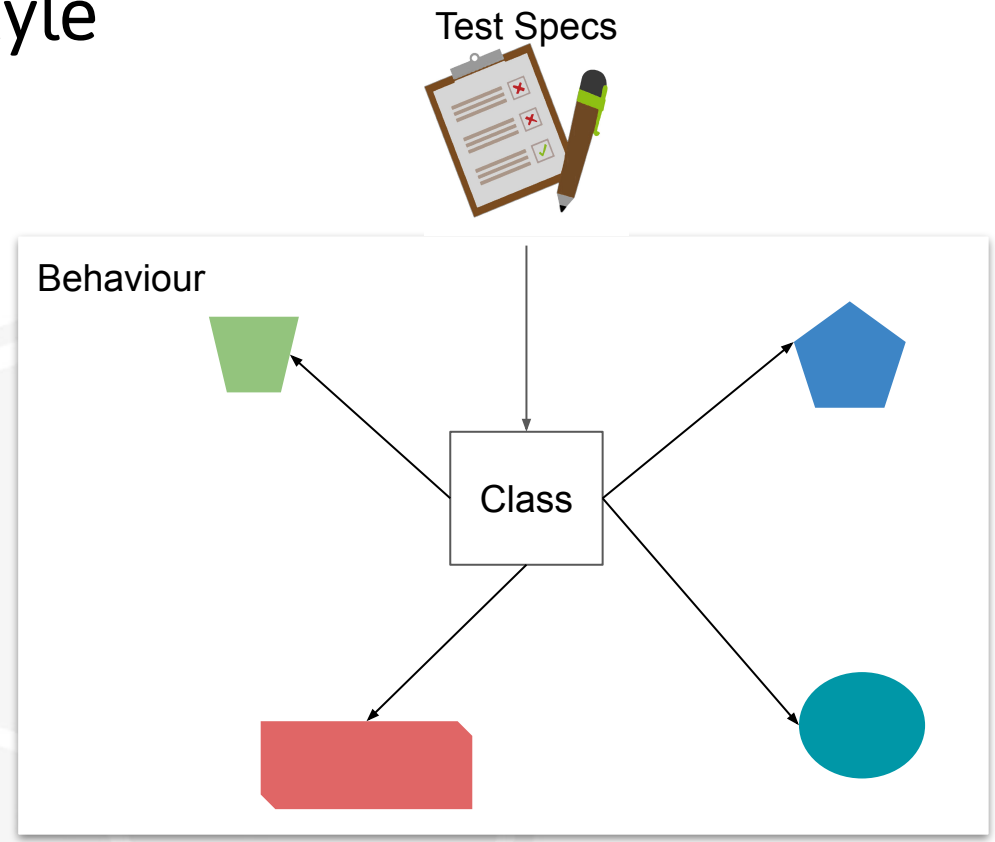
London Style



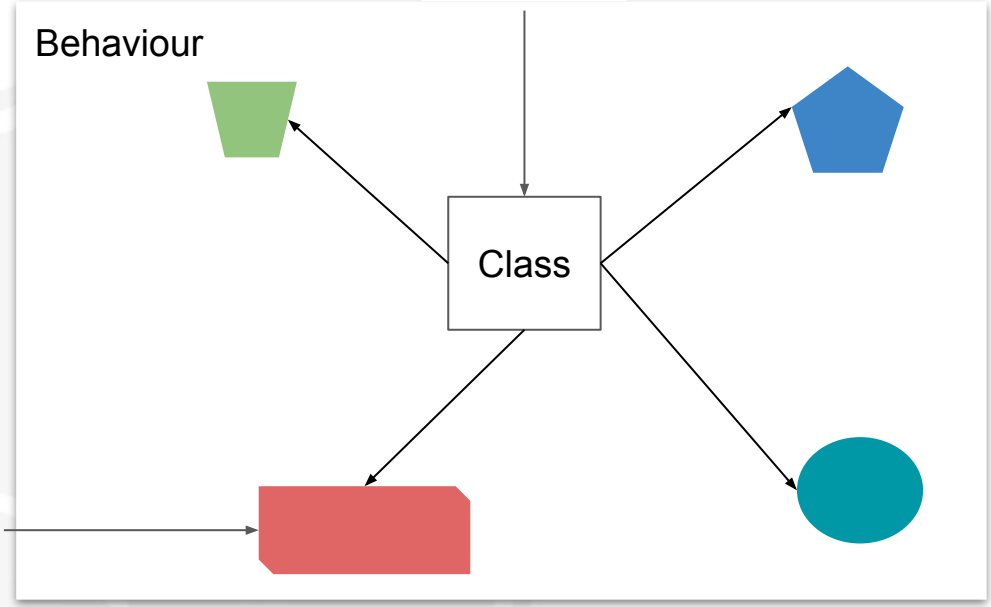
London Style



Detroit Style

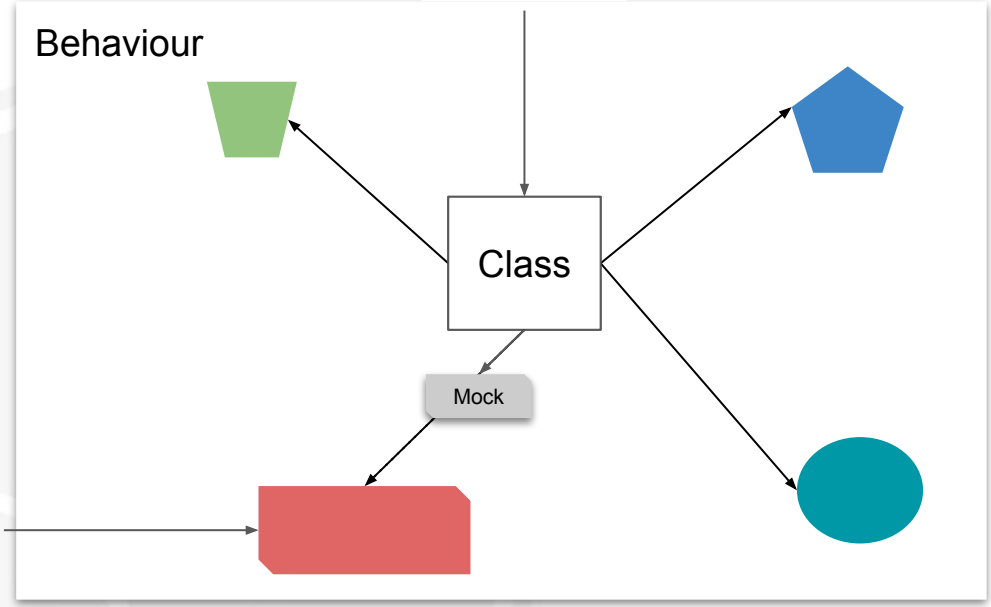


Detroit Style



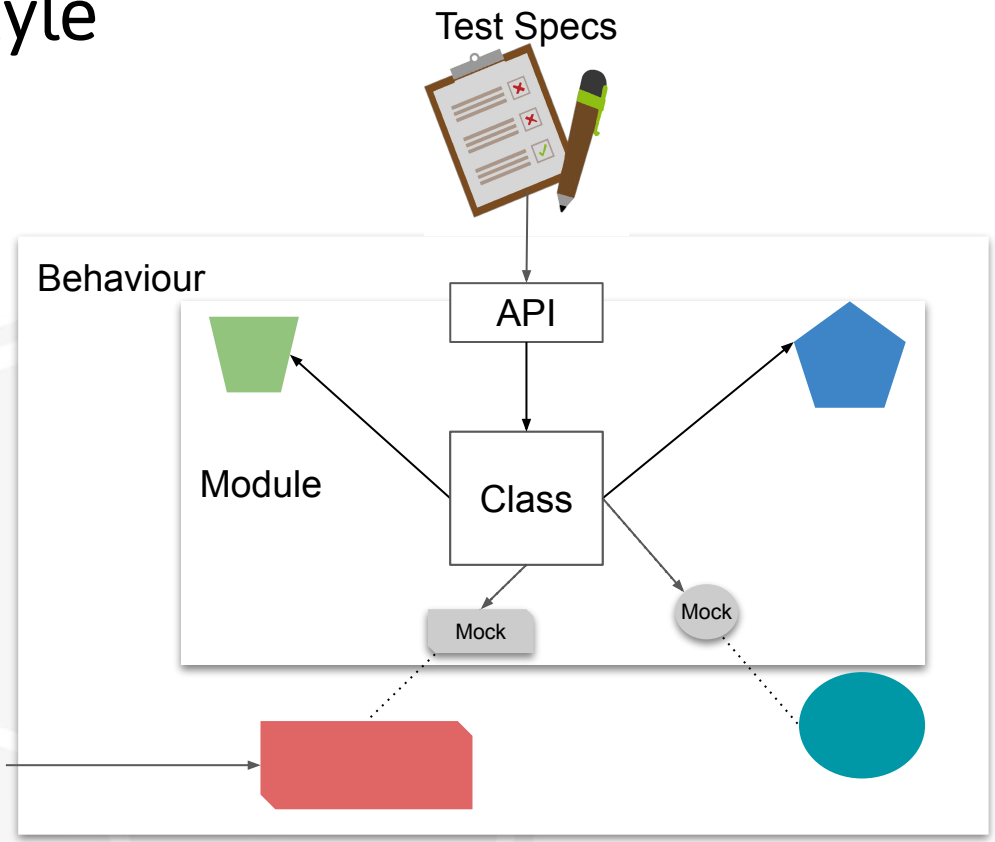
Out-of-System Dependency
e.g. HttpClient

Detroit Style

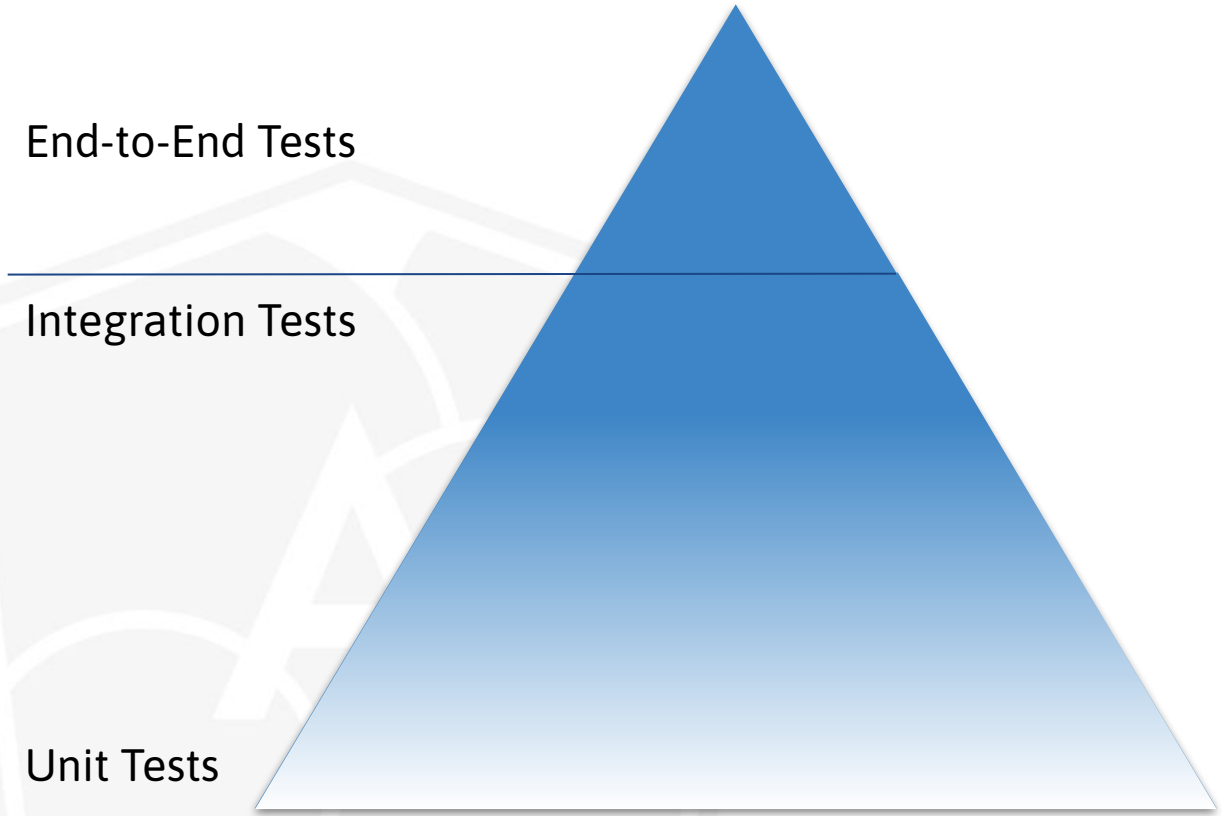


Out-of-System Dependency
e.g. HttpClient

Detroit Style

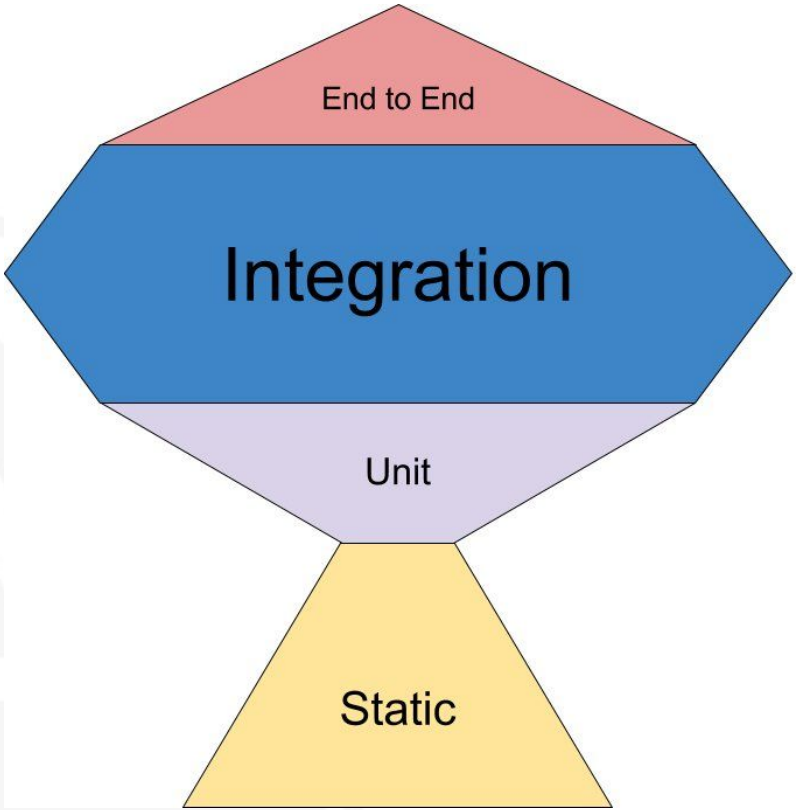


Testing Pyramid Revisited



ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

Testing According to ROI



Kent C. Dodds: https://twitter.com/kentcdodds/status/960723172591992832

HttpTest

```
it("should use Angular's http mock", () => {
  TestBed.configureTestingModule({
    declarations: [RequestInfoComponent],
    imports: [ReactiveFormsModule, HttpClientTestingModule],
  });

  const httpController = TestBed.inject(HttpTestingController);
  const fixture = TestBed.createComponent(RequestInfoComponent);
  fixture.componentInstance.search();

  const request = httpController.match((req) => !!req.url.match(/nominatim/))[0];
  request.flush([{ street: "Domgasse", streetNumber: 5 }]);
  expect(lookupResult.textContent.trim()).toBe("Address found");
});
```

Instead of HttpClientModule

Runs AFTER http request

RoutingTest

- RouterTestingModule provides routing functionality for tests
- Location can verify the expected url
- RoutingConfiguration is required

Testable Architecture

Different Testing Techniques

1. Unit / Integration Range
    a. Full mocking, no TestBed
    b. Selected mocking, without DOM interaction
    c. Selected mocking, DOM interaction
    d. Most minimal mocking, DOM Interaction
2. Exotic
    a. RxJs Marbles
    b. Visual Regression
    c. Component Tests via Storybook/Cypress (E2E)

Potential Problems

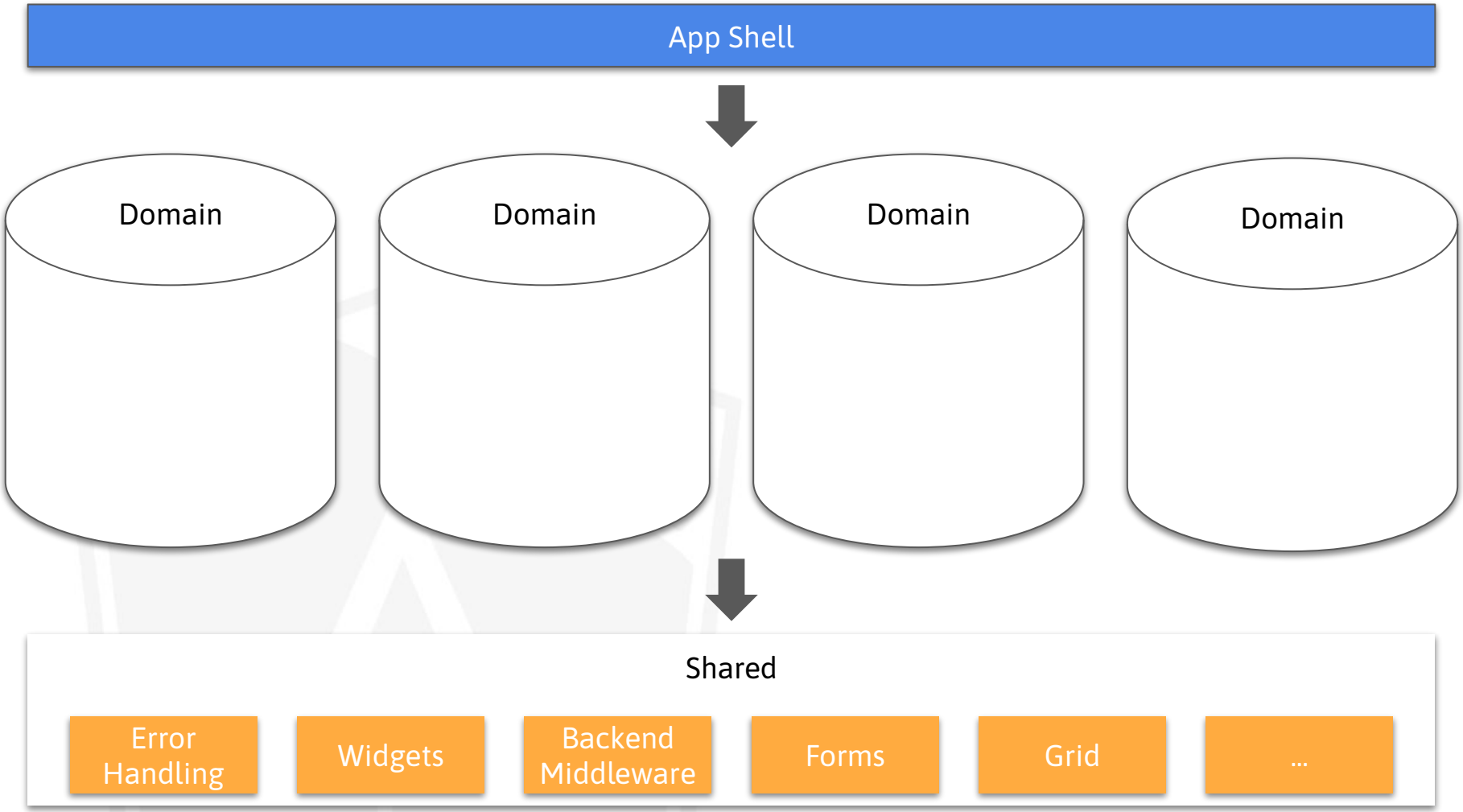
- Unit Tests (London)
  - What technique should be applied?
  - Too much mocking
  - Should I have unit test for everything?
- Integration Tests (Detroit Unit)
  - Too much setup required → feels like E2E
  - What should I mock?

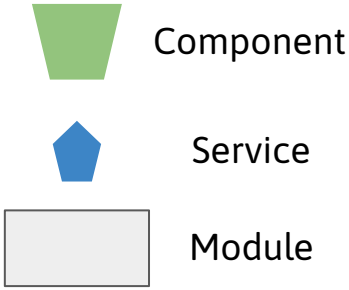
Unit vs. Integration

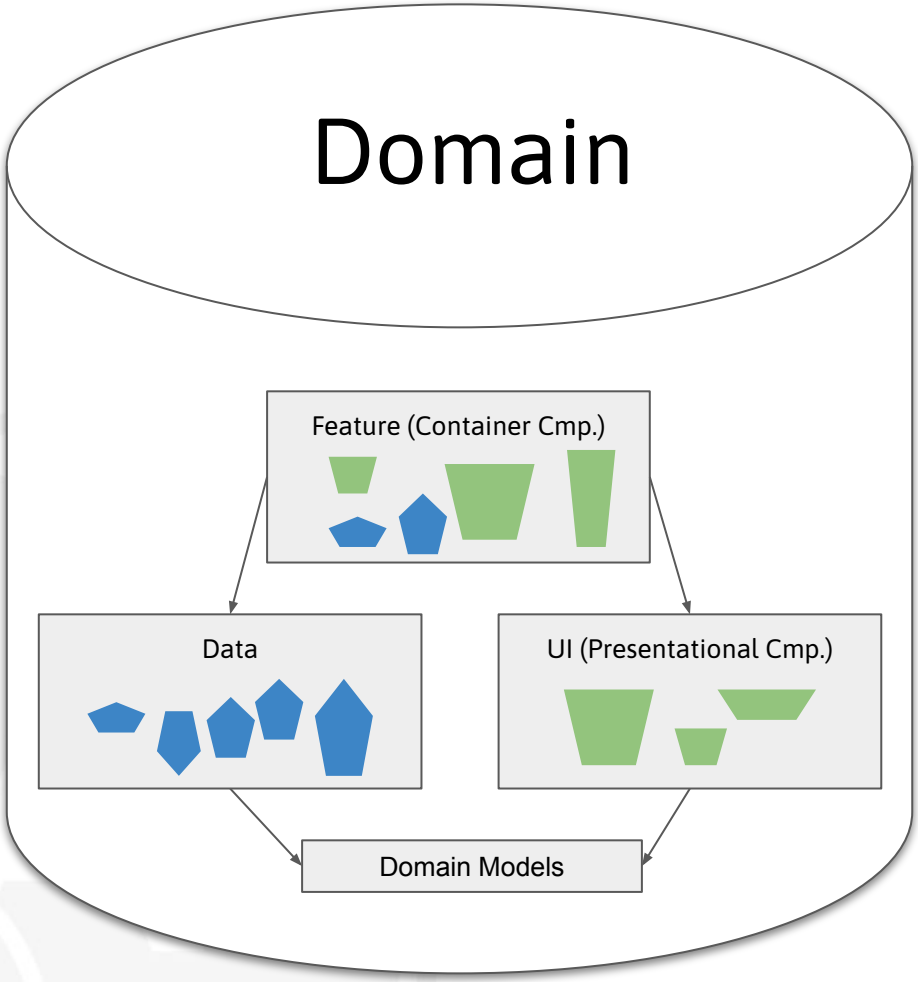
- Largely Depends on Application
- Most parts of data processing (unit test) done in backend
- Frontend as "proxy" → less logic
- Integration is King

Testable Architecture

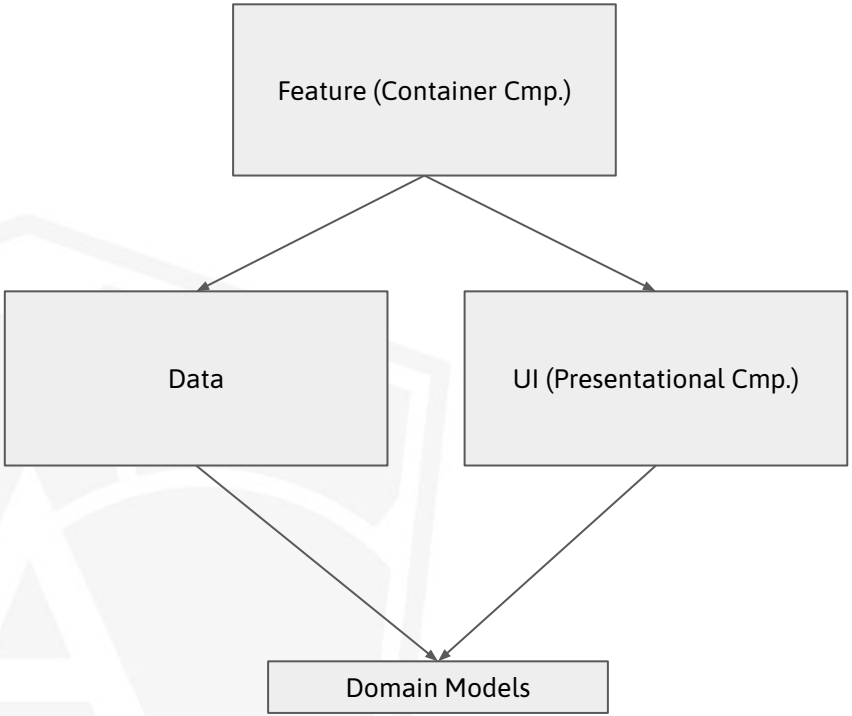
- Unit Tests
  - Class has a defined type
  - One testing technique per Type
- Integration Tests
  - Reduction of dependencies via domain/feature boundaries
  - Integration Test per Domain/Feature
  - Entry point is the feature component


App Shell
Domain
Domain
Domain
Domain
Shared
Error Handling
Widgets
Backend Middleware
Forms
Grid
...

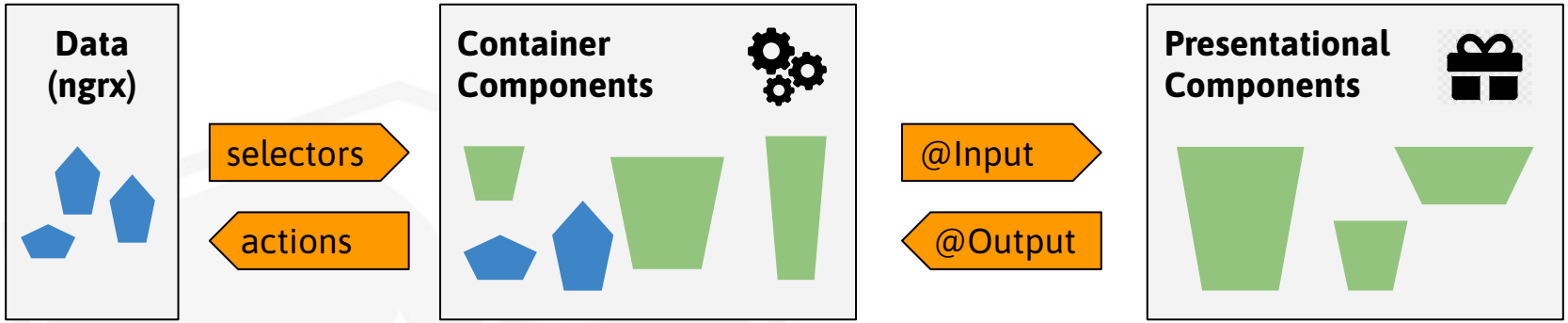




ANGULAR ARCHITECTS
INSIDE KNOWLEDGE



Container & Presentational Components

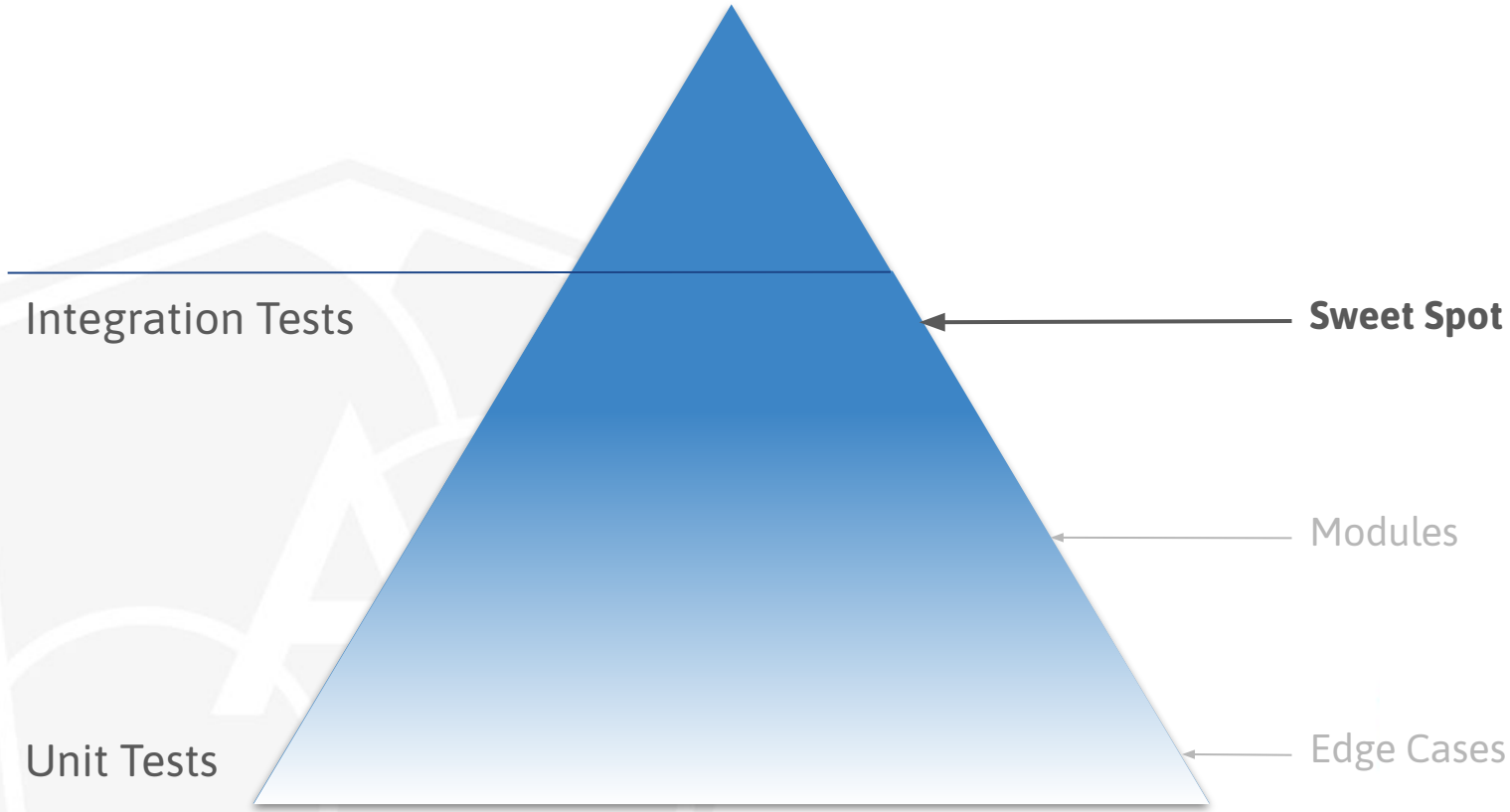


- Minimal HTML/CSS
- Lots of DI
- Facilitator

- HTML & CSS only
- Minimal Typescript
- No DI
- @Input & @Output
- No Observables

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

Sweet Spot


Integration Tests
Sweet Spot
Modules
Unit Tests
Edge Cases

Sweet Spot Examples

- Per Domain
- Per Feature
- Tests which are too hard for E2E

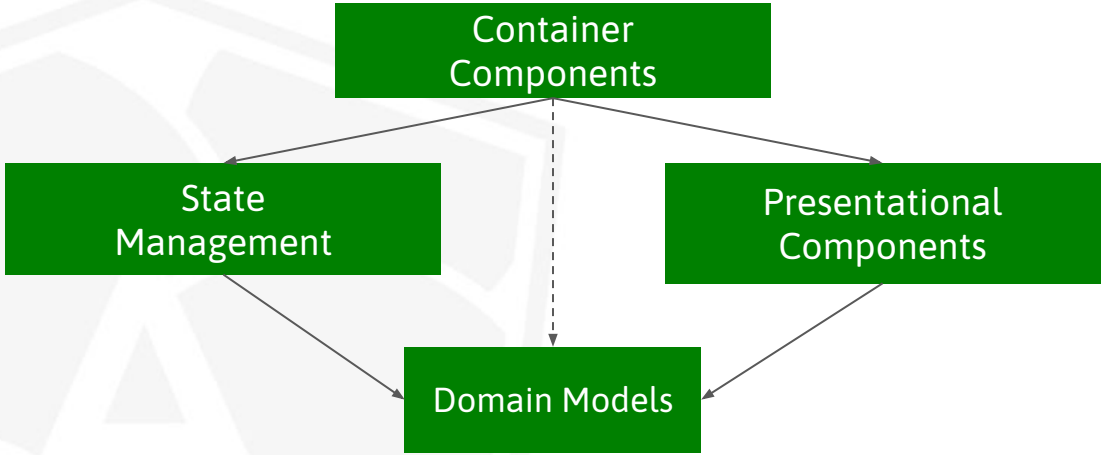
ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Sweet Spot: Testing Techniques

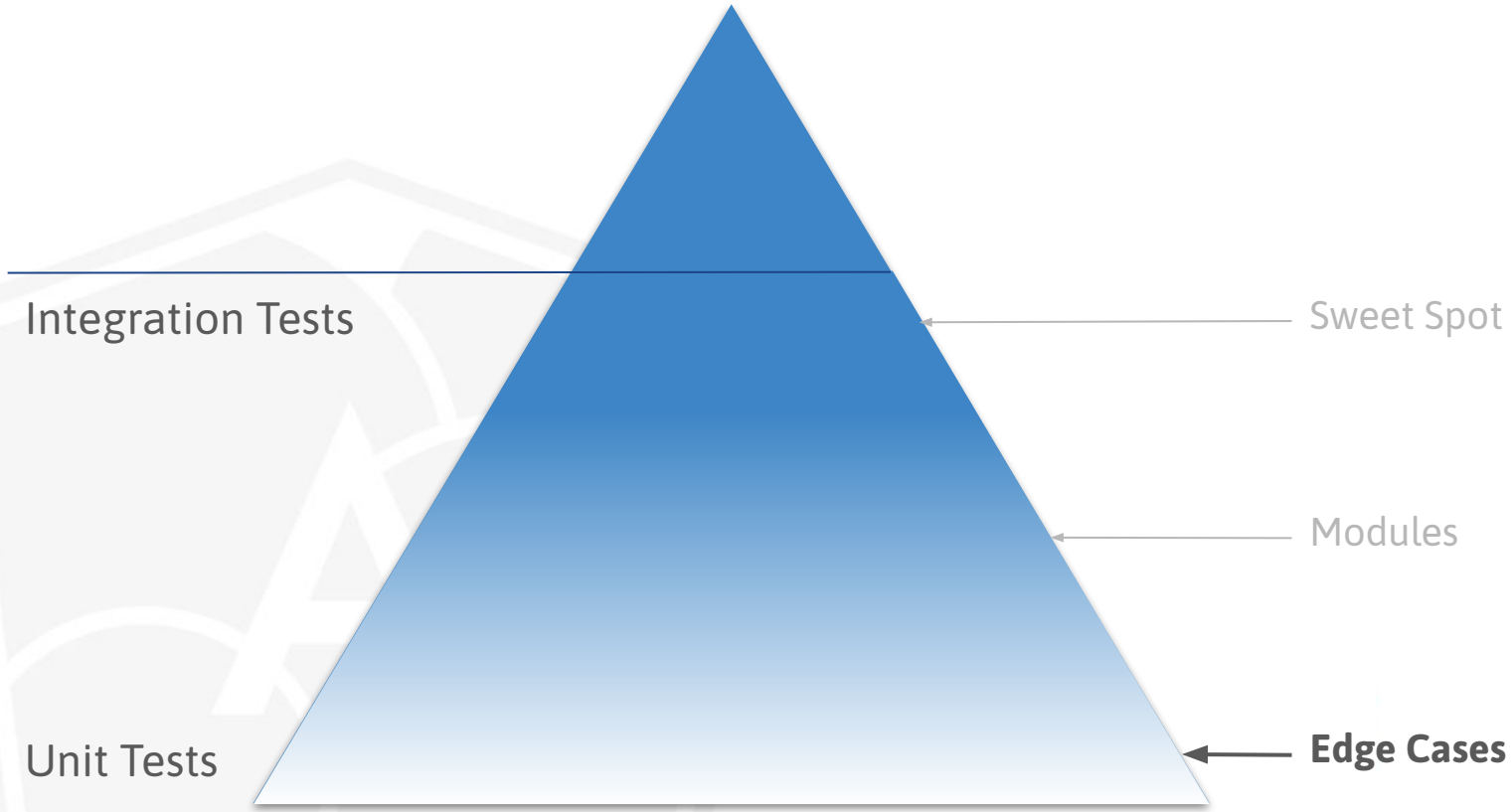
1. Unit / Integration Range
    a. Full mocking, no TestBed
    b. Selected mocking, without DOM interaction
    c. Selected mocking, DOM interaction
    d. Most minimal mocking, DOM Interaction
2. Exotic
    a. RxJs Marbles
    b. Visual Regression
    c. Component Tests via Storybook/Cypress (E2E)

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Sweet Spot



Edge Cases


Integration Tests
Sweet Spot
Modules
Unit Tests
Edge Cases

Edge Cases Examples

- Components
- Services
- Functions

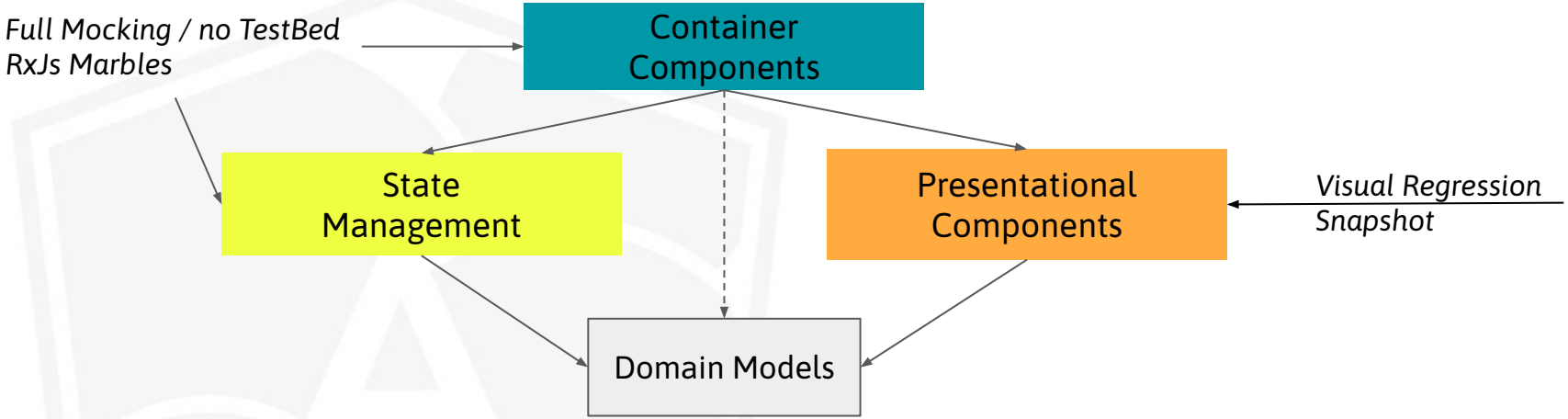
ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Edge Cases: Testing Techniques

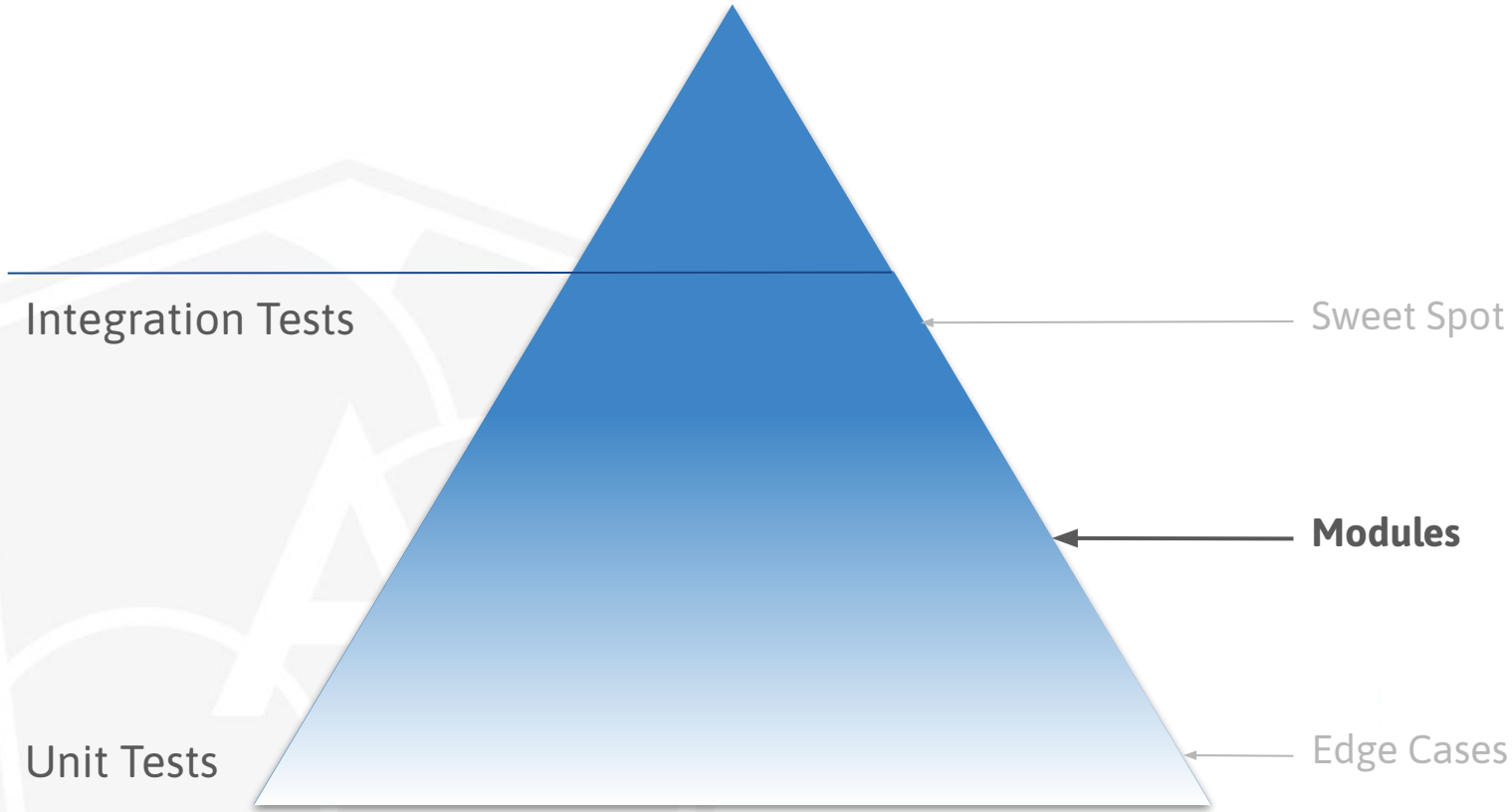
1. Unit / Integration Range
    a. **Full mocking, no TestBed**
    b. Selected mocking, without DOM interaction
    c. Selected mocking, DOM interaction
    d. Most minimal mocking, DOM Interaction
2. Exotic
    a. **RxJs Marbles**
    b. **Visual Regression**
    c. Component Tests via Storybook/Cypress (E2E)

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Edge Cases



Modules


Integration Tests
Sweet Spot
Modules
Unit Tests
Edge Cases

Modules Examples

- Interplay between Container & Presentational Components
- Complex Components (DataGrid)
- State Management
- Tests with Browser support (LocalStorage,...)

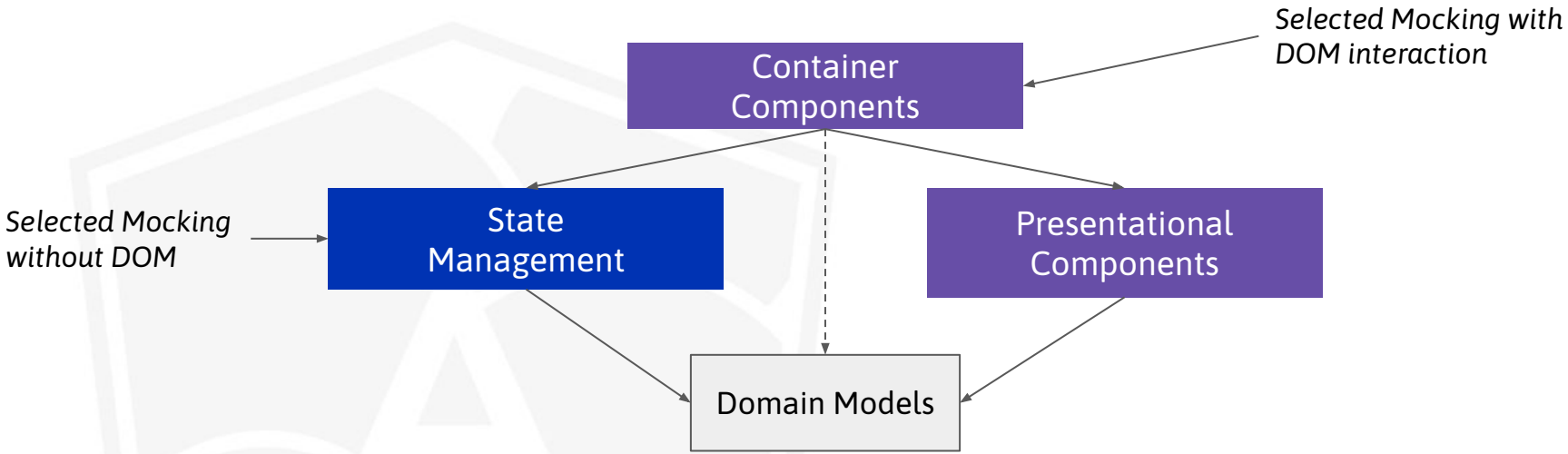
ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Modules: Testing Techniques

1. Unit / Integration Range
    a. Full mocking, no TestBed
    b. Selected mocking, without DOM interaction
    c. Selected mocking, DOM interaction
    d. Most minimal mocking, DOM Interaction
2. Exotic
    a. RxJs Marbles
    b. Visual Regression
    c. Component Tests via Storybook/Cypress (E2E)

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Logical Groups



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Summary

- Try to go for Integration Tests
- Use Code Coverage as Analysis Tool
  - Leave the edge cases for Unit Tests with full mocking
  - Rest with "Modules" Tests
- Apply the right Techniques:
  - Visual Regression: Presentational Components
  - RxJs Marbles: NgRx Effect
  - Storybook/Cypress: Component Groups

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Lab Time