Testing in the Clean Architecture

CSC 207 Software design

Beyond simple unit testing

We are familiar with the idea of unit testing from Python.

Now that we have our more complicated Clean Architecture structure to work with, how can we test our code?

For example, how can we test a Use Case Interactor since it depends on implementations of various interfaces to function correctly?

Three types of tests

- **Unit Test** — Tests the smallest unit of cohesive code, often a method. We can create a test class called `CourseTests` of unit tests for all the methods in class `Course`, for example.

- **Integration Test** — Tests how two components (say classes) interact with each other.

- **End-to-end Test** — Tests an entire path through the code from input to output. This can start and end with the View.

Thought Question

Which type of test is most important to ensure that our programs behave correctly?

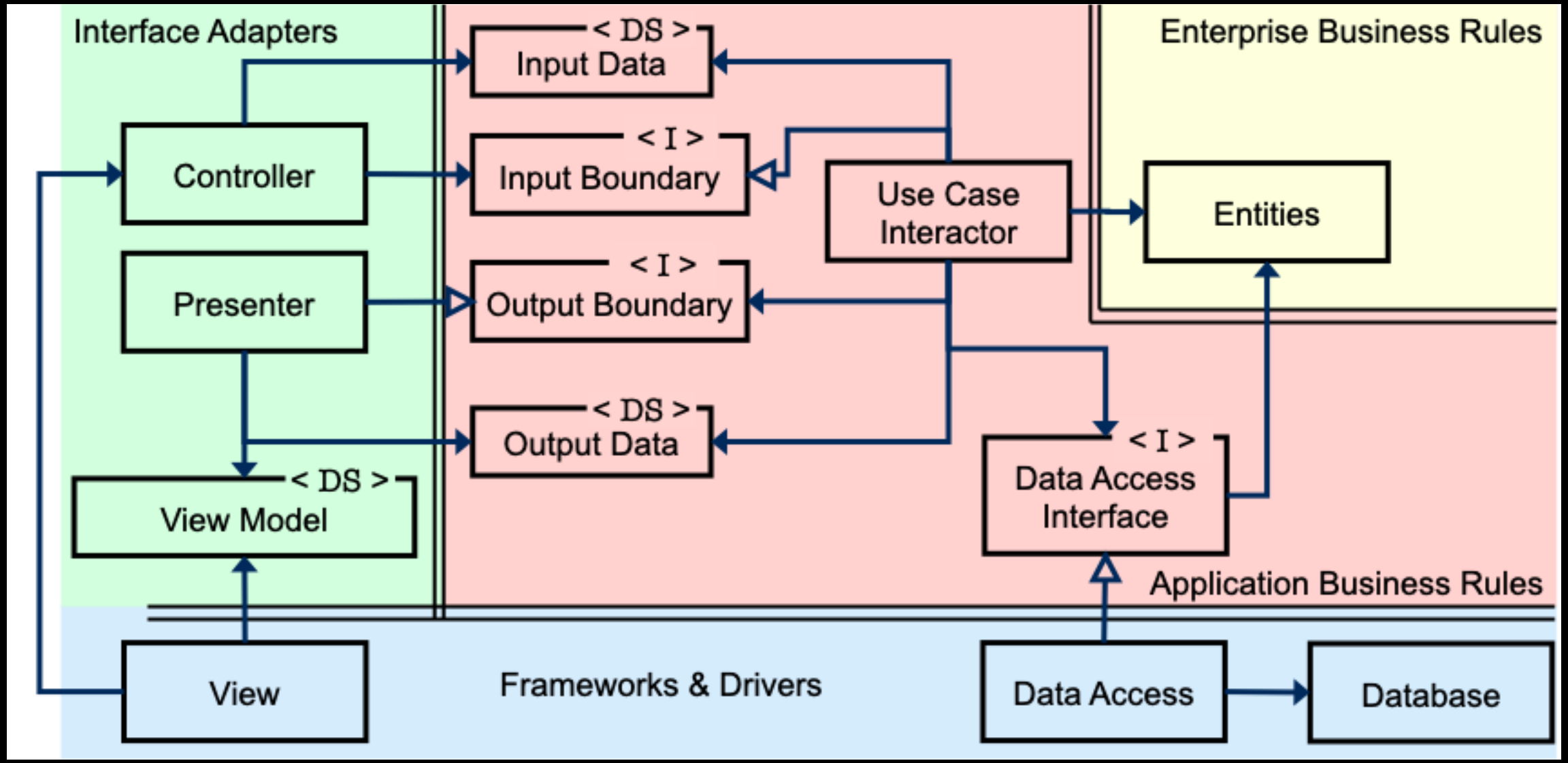
A – unit tests

B – integration tests

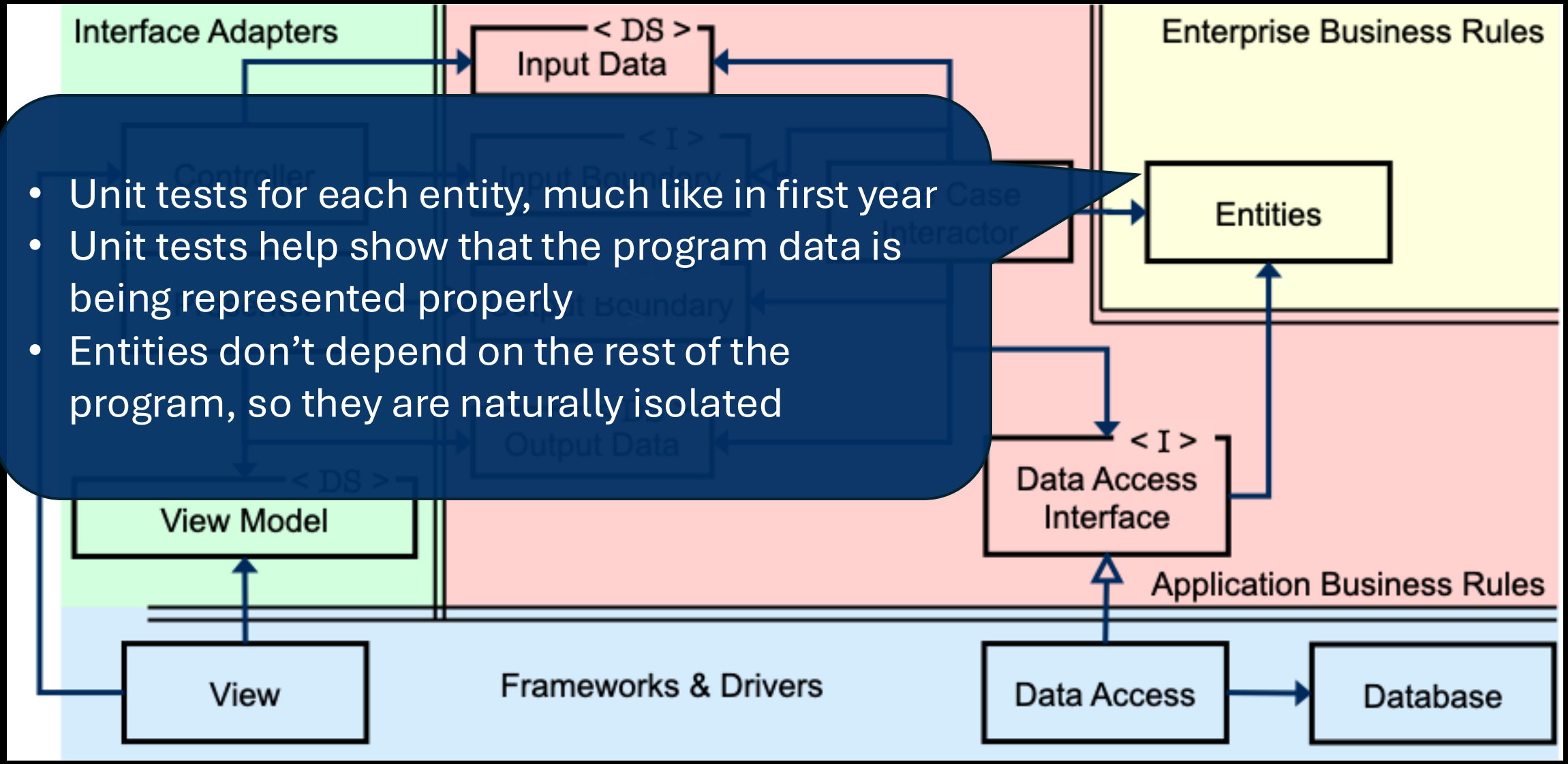
C – end-to-end tests

D – they are all equally important

How can we test this?

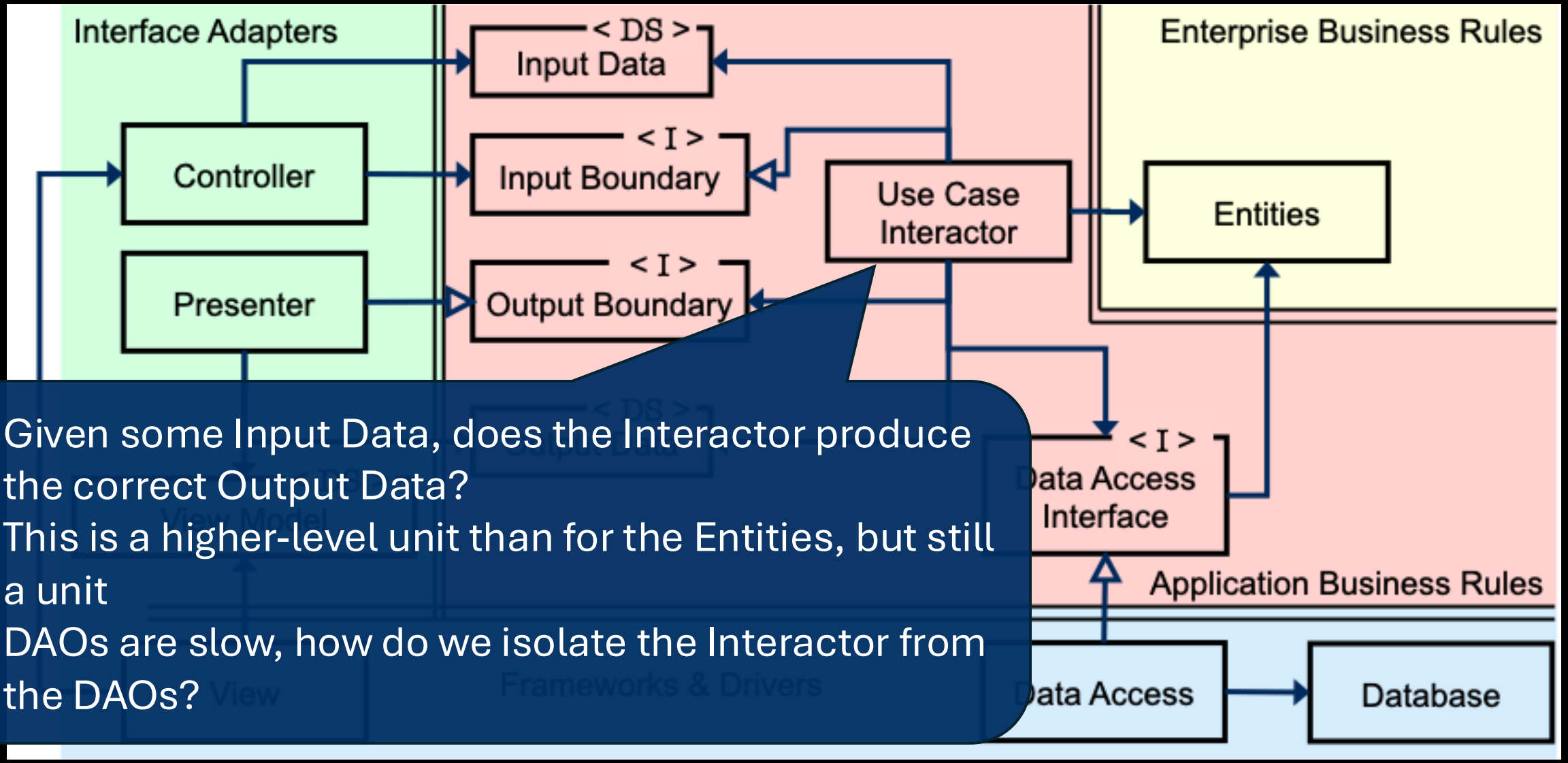


Entities: unit testing



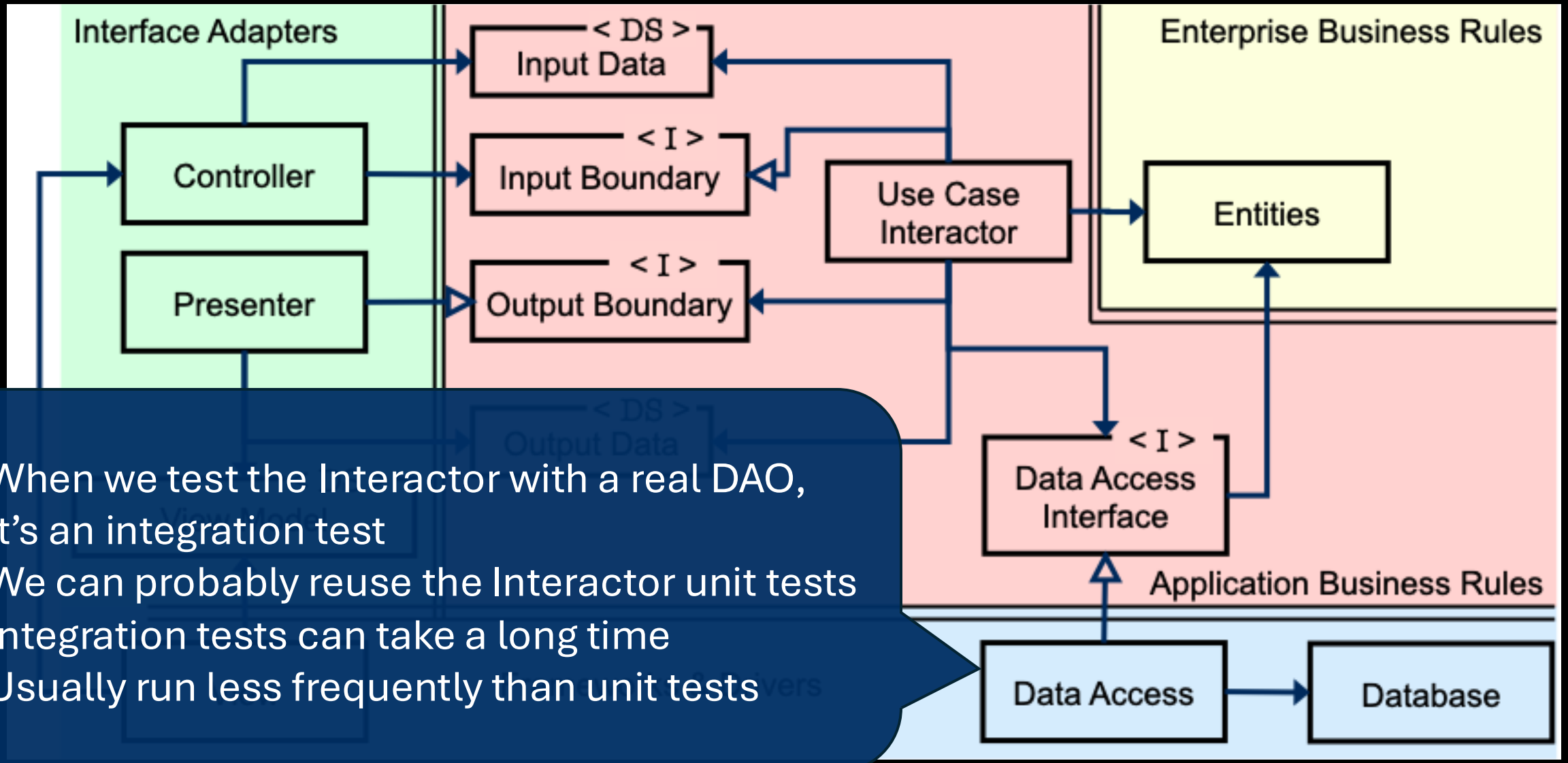
- Unit tests for each entity, much like in first year
- Unit tests help show that the program data is being represented properly
- Entities don't depend on the rest of the program, so they are naturally isolated

Interactors: unit testing



- Given some Input Data, does the Interactor produce the correct Output Data?
- This is a higher-level unit than for the Entities, but still a unit
- DAOs are slow, how do we isolate the Interactor from the DAOs?

Interactors and DAOs: integration testing



- When we test the Interactor with a real DAO, it's an integration test
- We can probably reuse the Interactor unit tests
- Integration tests can take a long time
- Usually run less frequently than unit tests

End-to-end testing the CA Engine