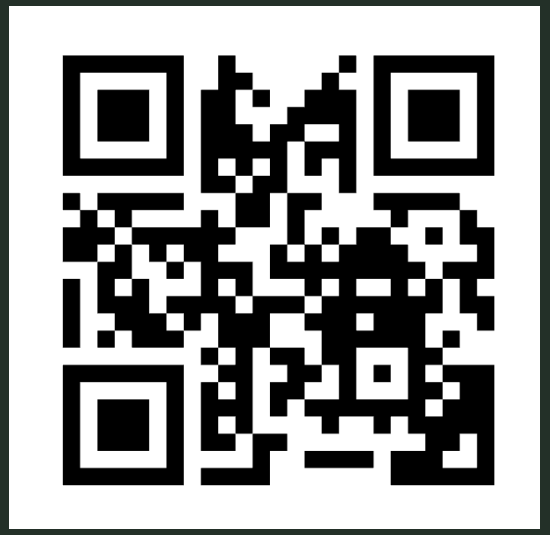
Ted M. Young

Java Trainer, Coach, Live Coder, and Creator of JitterTed's TDD Game



Testable Architecture

Keep 'em Separated

Get in touch: https://**ted.dev**/about

Twitch: https://JitterTed.**Stream**

YouTube: https://JitterTed.**TV**

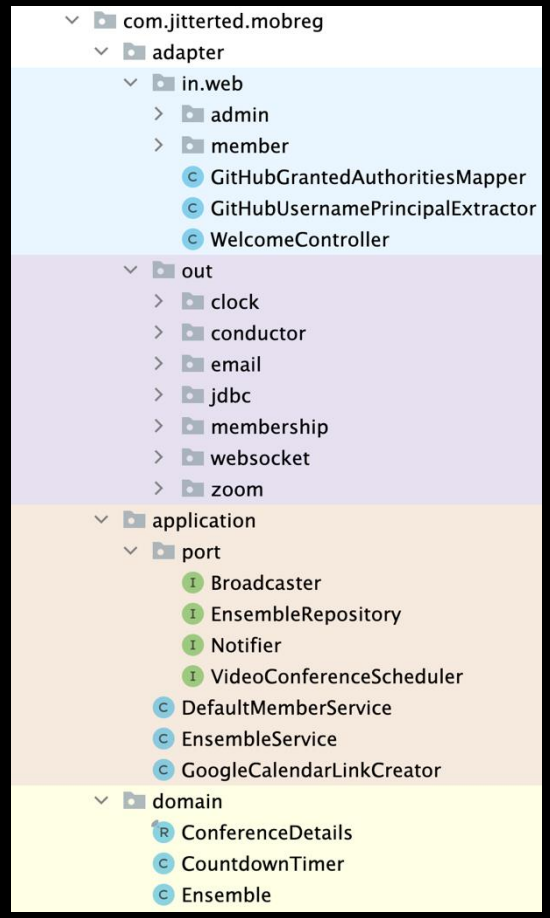
Source Code? Slides? Go here:

https://ted.dev/talks

What is Architecture?

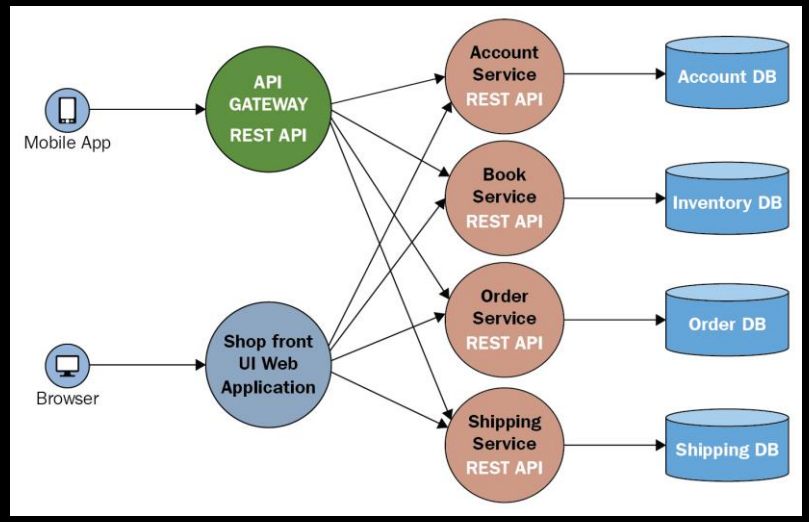
Architecture is…

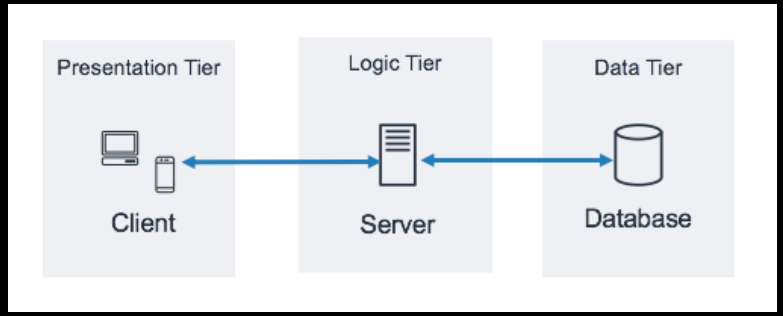
Organization of Code



layers

Deployment/Operational





tiers

Writing tests is hard because…

Classes are poorly organized	Classes mix logic and I/O access
Too much state Too many responsibilities	Rule/Calculation entangled with Hardware

https://ted.dev/about

Two Rules for Testability…

#1

Keep I/O Separate

#2

I/O Depends on Domain

#1: Keep I/O Separate

I/O = Anything Outside the Current Process

I/O includes…

- Hardware
  - System clock, Random (shuffle collection), Local files
- Remote Services
  - via network: HTTP, Queues, Sockets
- Databases
  - Including "in-memory" databases (H2, SQLite, Redis)
- Frameworks (Spring, Quarkus, etc.)

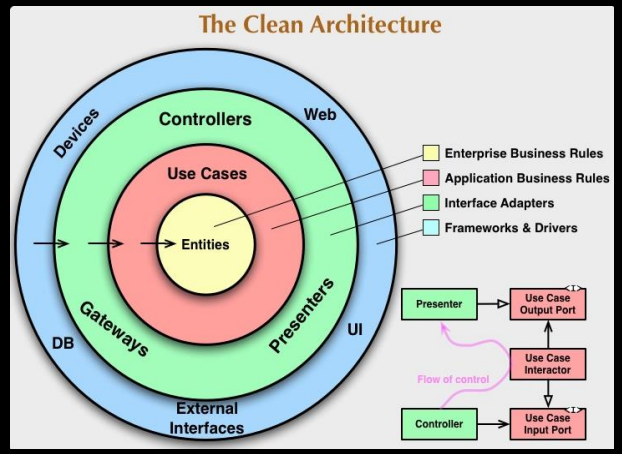
https://ted.dev/about

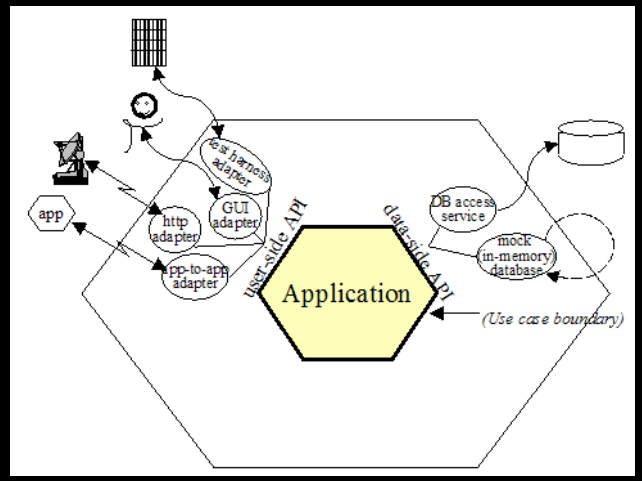
SCION-T

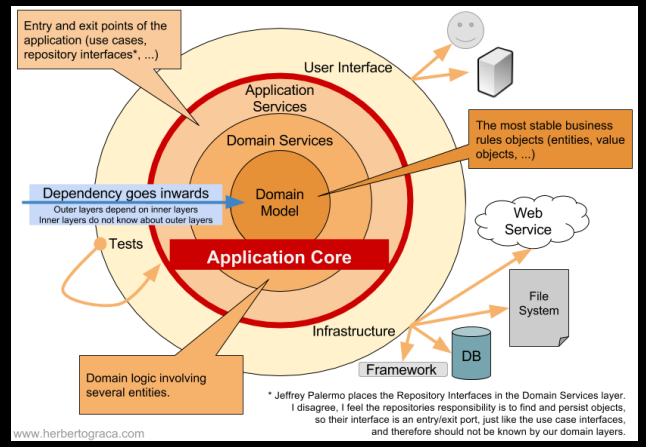
Separating
Concerns of
I/O and
Non-I/O for
Testability

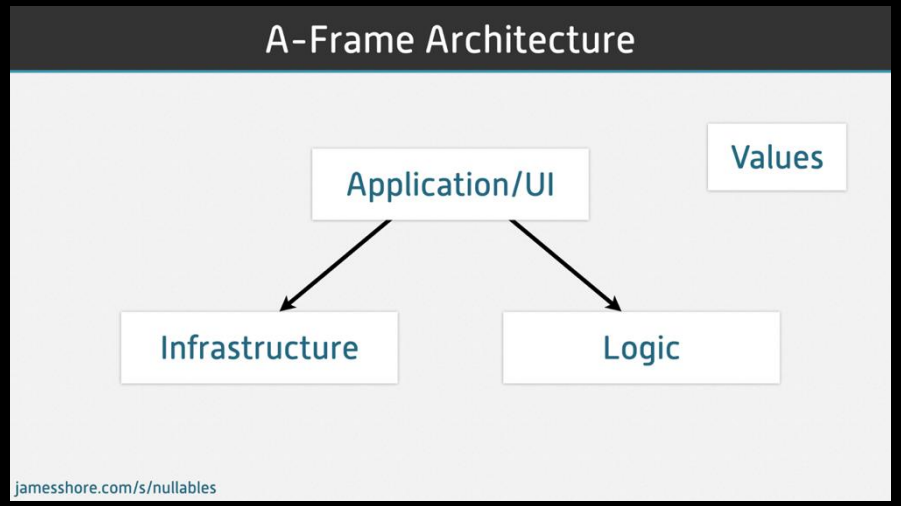
Testable Architecture

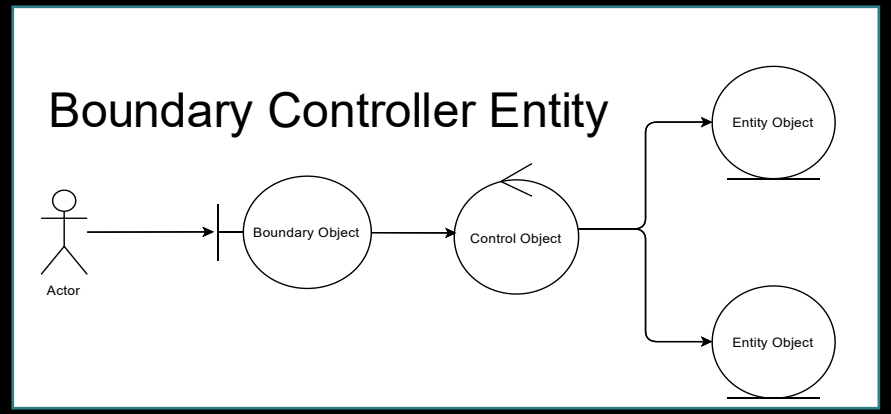
Some Testable Architectures...

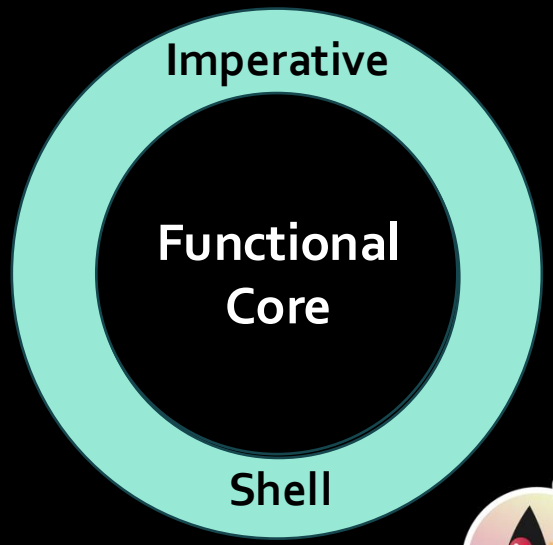












https://ted.dev/about

Inside the Code

A Simple Example

```java
public class MixedMeal {
    private static final NumberFormat CURRENCY_FORMATTER = NumberFormat.getCurrencyInstance();
    private static final double SALES_TAX_PERCENT = 8.25;
    private static final double SALES_TAX_RATE = SALES_TAX_PERCENT / 100;

    public static void main(String[] args) {
        System.out.println("""
                Choose an item to purchase:
                1) Burger        $11.95
                2) Pizza         $ 8.95
                3) Fried Chicken $ 9.95""");

        Scanner scanner = new Scanner(System.in);
        int choice = scanner.nextInt();

        double price = switch (choice) {
            case 1 -> 11.95;
            case 2 -> 8.95;
            case 3 -> 9.95;
            default -> throw new IllegalArgumentException("Invalid choice: " + choice);
        };

        double taxAmount = price * SALES_TAX_RATE;
        double total = price + taxAmount;

        String receipt = """
                Subtotal: %s
                     Tax: %s
                   Total: %s""".formatted(
                CURRENCY_FORMATTER.format(price),
                CURRENCY_FORMATTER.format(taxAmount),
                CURRENCY_FORMATTER.format(total));
        System.out.println(receipt);
    }
}
```

Mixed Concerns

https://ted.dev/about

Testing Mixed Concerns

```
18  class MixedMealTest {
19
20  ...
44
45      private static void sendChoiceOf(String simulatedInput) {
46          String simulatedInputWithNewline = simulatedInput + "\n";
47          System.setIn(new ByteArrayInputStream(
48                  simulatedInputWithNewline.getBytes(StandardCharsets.UTF_8)));
49      }
50
51      private String outputString() {
52          return testOut.toString(StandardCharsets.UTF_8);
53      }
54
55      @Test
56      void burgerChoiceTaxedAtCorrectRateWithTotals() {
57          sendChoiceOf("1"); // choose Burger via injecting to System.in
58
59          MixedMeal.main(new String[0]);
60
61          String output = outputString(); // get captured System.out
62          assertThat(output)
63                  .as("Receipt should include tax and totals for Burger choice")
64                  .contains("Subtotal: $11.95",
65                            "     Tax: $0.99",
66                            "   Total: $12.94");
67      }
68  }
```

https://ted.dev/about

Refactor Out Calculations

Extract "Domain" or "Business" Logic (Decisions)

Separated Concerns

```
11      public static void main(String[] args) {
12          System.out.println("""
13                  Choose an item to purchase:
14                  1) Burger        $11.95
15                  2) Pizza         $ 8.95
16                  3) Fried Chicken $ 9.95""");
17
18          Scanner scanner = new Scanner(System.in);
19          int choice = scanner.nextInt();
20
21          String receipt = computeReceiptFor(choice);
22
23          System.out.println(receipt);
24      }
```

```
26  @   public static String computeReceiptFor(int choice) {
27          double price = switch (choice) {
28              case 1 -> 11.95;
29              case 2 -> 8.95;
30              case 3 -> 9.95;
31              default -> throw new IllegalArgumentException(
32                              "Invalid choice: " + choice);
33          };
34
35          double taxAmount = price * SALES_TAX_RATE;
36          double total = price + taxAmount;
37
38          return """
39                  Subtotal: %s
40                       Tax: %s
41                     Total: %s""".formatted(
42                  CURRENCY_FORMATTER.format(price),
43                  CURRENCY_FORMATTER.format(taxAmount),
44                  CURRENCY_FORMATTER.format(total));
45      }
46  }
```

https://ted.dev/about

Testing Separated Concerns

```
7   class SeparatedMealTest {
8
9       @Test
10      void burgerChoiceTaxedAtCorrectRateWithTotals() {
11          String output = SeparatedMeal.computeReceiptFor(1); // choose Burger
12
13          assertThat(output)
14                  .as("Receipt should include tax and totals for Burger choice")
15                  .contains("Subtotal: $11.95",
16                            "     Tax: $0.99",
17                            "   Total: $12.94");
18      }
19  }
```

https://ted.dev/about

Using Separated Concerns

```
8   @RestController
9   public class ReceiptController {
10
11      @GetMapping("/receipt")
12      public ResponseEntity<String> receipt(@RequestParam("choice") String choice) {
13          try {
14
15              int choiceInt = Integer.parseInt(choice);
16              String receipt = SeparatedMeal.computeReceiptFor(choiceInt);
17              return ResponseEntity.ok(receipt);
18
19          } catch (NumberFormatException nfe) {
20              return ResponseEntity.badRequest().body("choice must be an integer");
21          } catch (IllegalArgumentException iae) {
22              return ResponseEntity.badRequest().body(iae.getMessage());
23          }
24      }
25  }
```

Hexagonal + DDD Architecture

also known as PORTS & ADAPTERS

Disclaimer...

Hexagonal Architecture is not a Specification...

...it's a Pattern

“Allow an application to equally be driven by users, programs, [and] automated tests... and to be developed and tested in isolation from its eventual run-time devices and databases.”

Ports & Adapters Pattern
Alistair Cockburn

“Allow an application to equally be driven by users, programs, [and] automated tests... and to be developed and tested in isolation from its eventual run-time devices and databases.”

Ports & Adapters Pattern
Alistair Cockburn

“Allow an application to equally be driven by users, programs, [and] automated tests... and to be developed and tested in isolation from its eventual run-time devices and databases.”

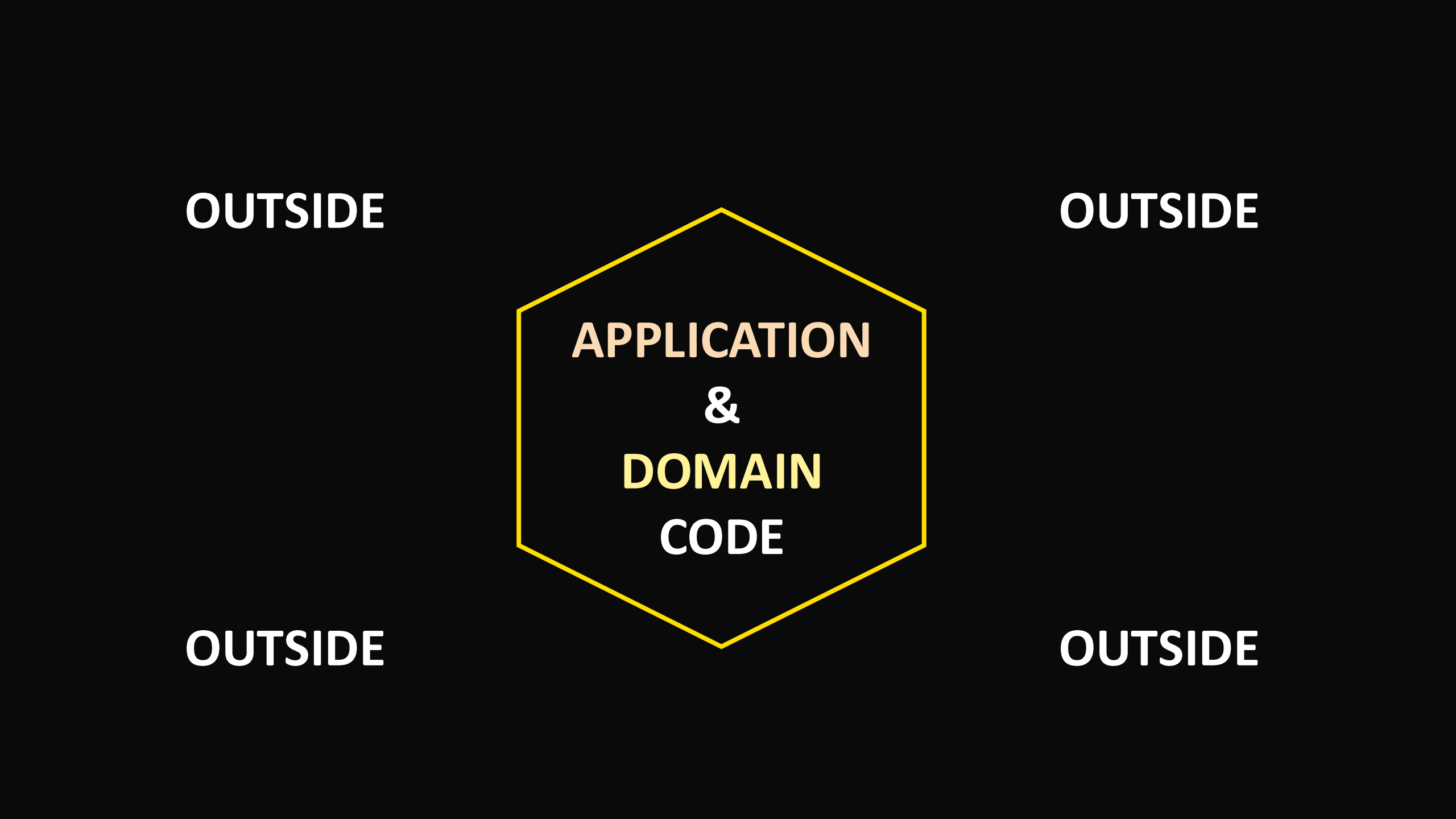
Ports & Adapters Pattern
Alistair Cockburn

OUTSIDE

OUTSIDE

OUTSIDE

OUTSIDE

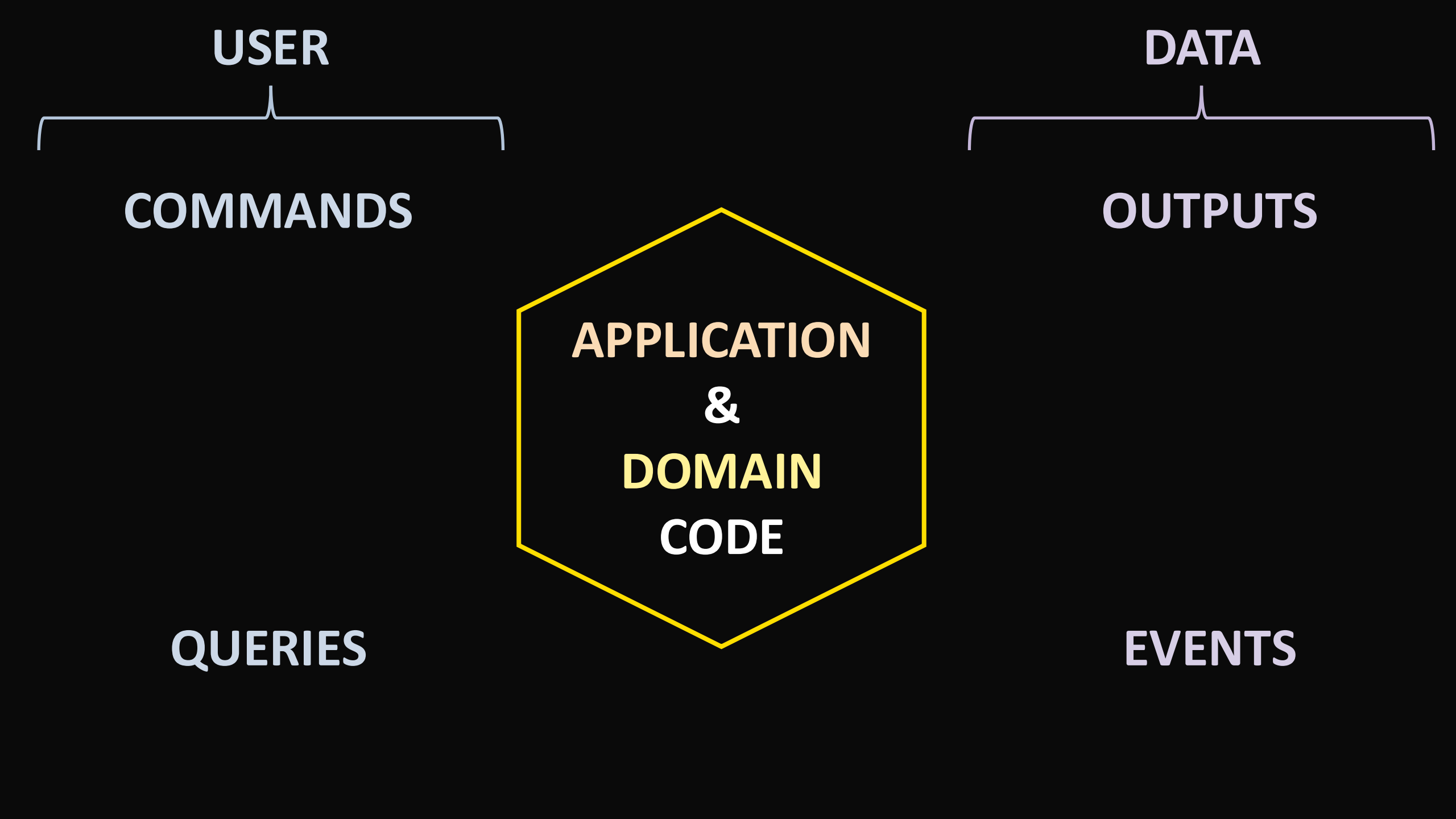

OUTSIDE
OUTSIDE
APPLICATION
&
DOMAIN
CODE
OUTSIDE
OUTSIDE

REST

OF

THE

WORLD


USER
DATA
COMMANDS
OUTPUTS
APPLICATION
&
DOMAIN
CODE
QUERIES
EVENTS

COMMANDS

OUTPUTS

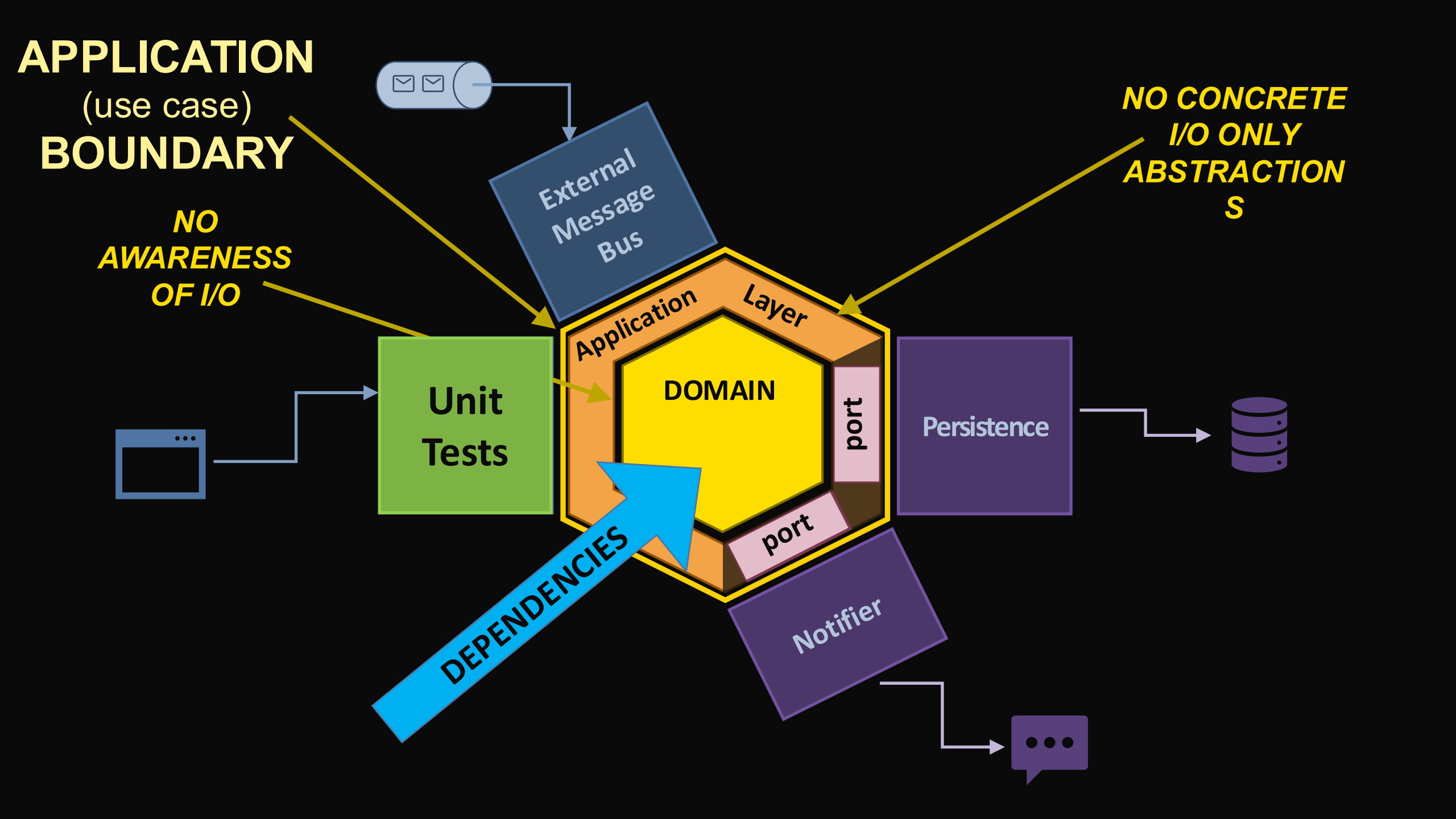
INBOUND

APPLICATION
&
DOMAIN
CODE

OUTBOUND

QUERIES

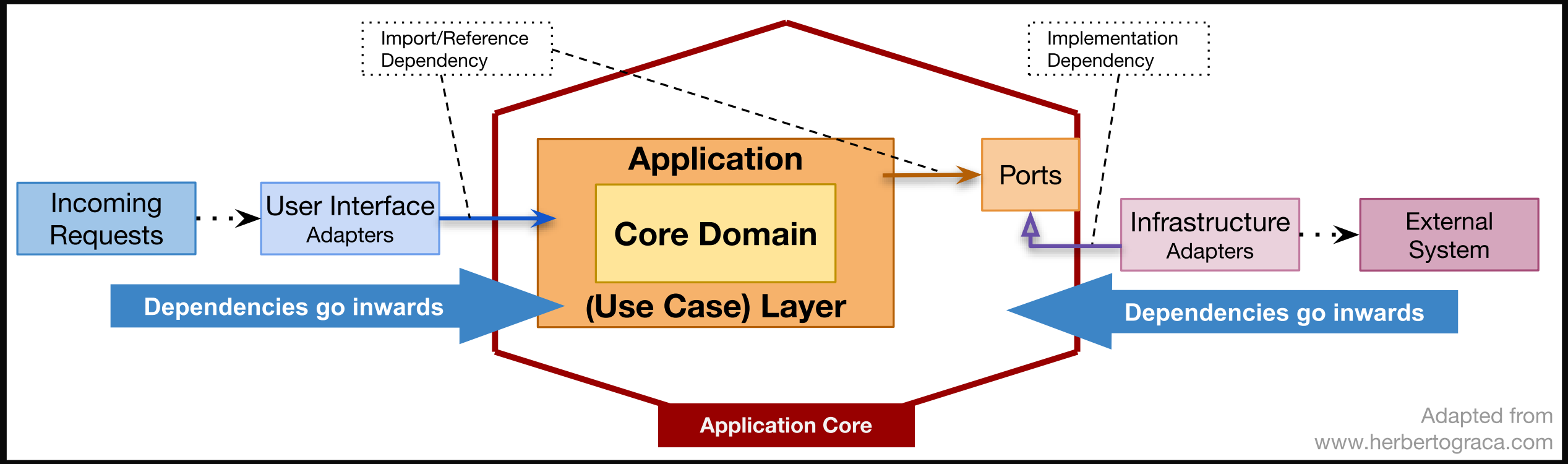
EVENTS


APPLICATION
(use case)
BOUNDARY
NO AWARENESS OF I/O
NO CONCRETE I/O ONLY ABSTRACTIONS
External Message Bus
Application Layer
DOMAIN
port
port
Unit Tests
Persistence
Notifier
DEPENDENCIES

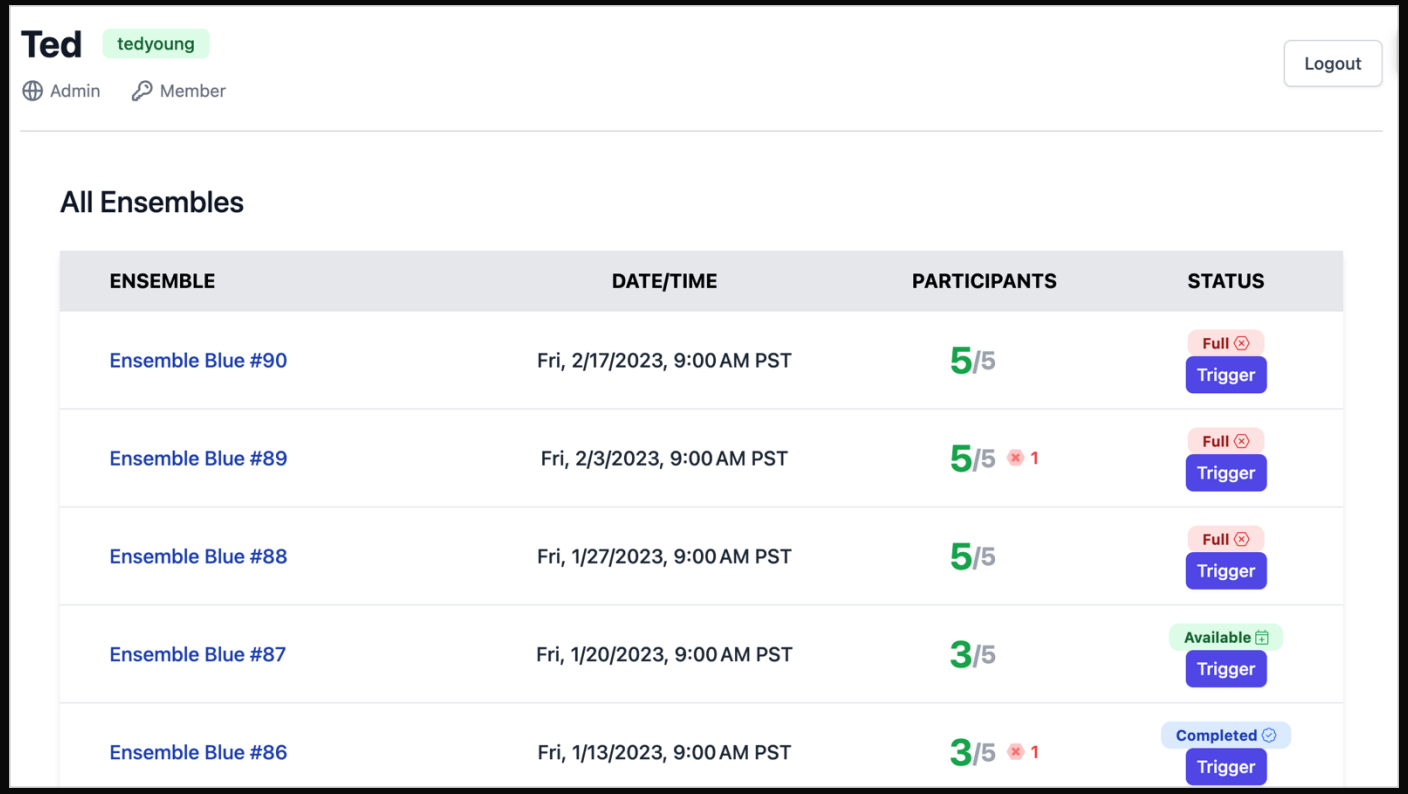
#2: Dependency Direction

Dependencies Point Inwards

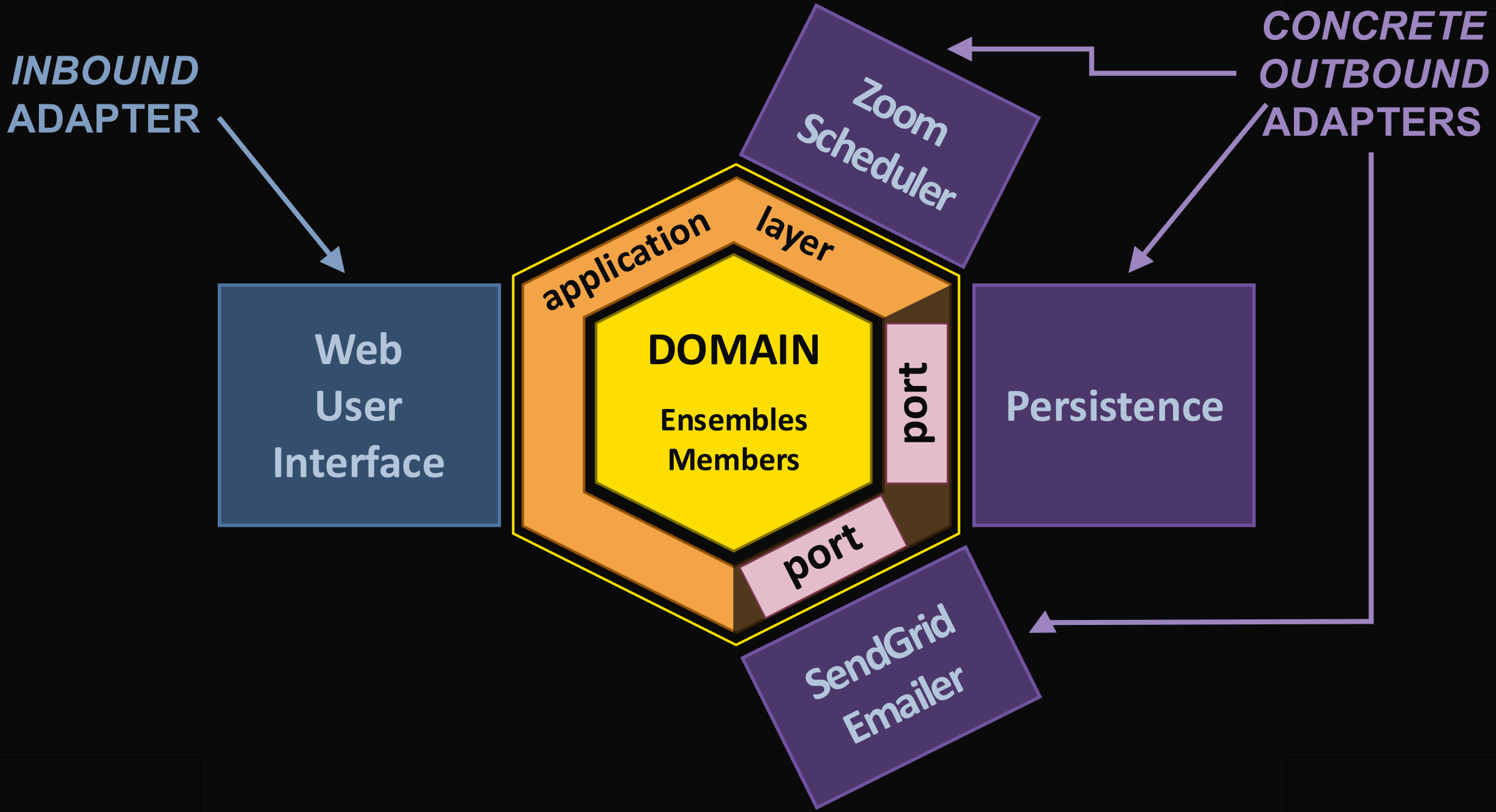
Dependencies Point Inward



ENSEMBLER
managing ensembling sessions



Source Code: https://github.com/jitterted/ensembler

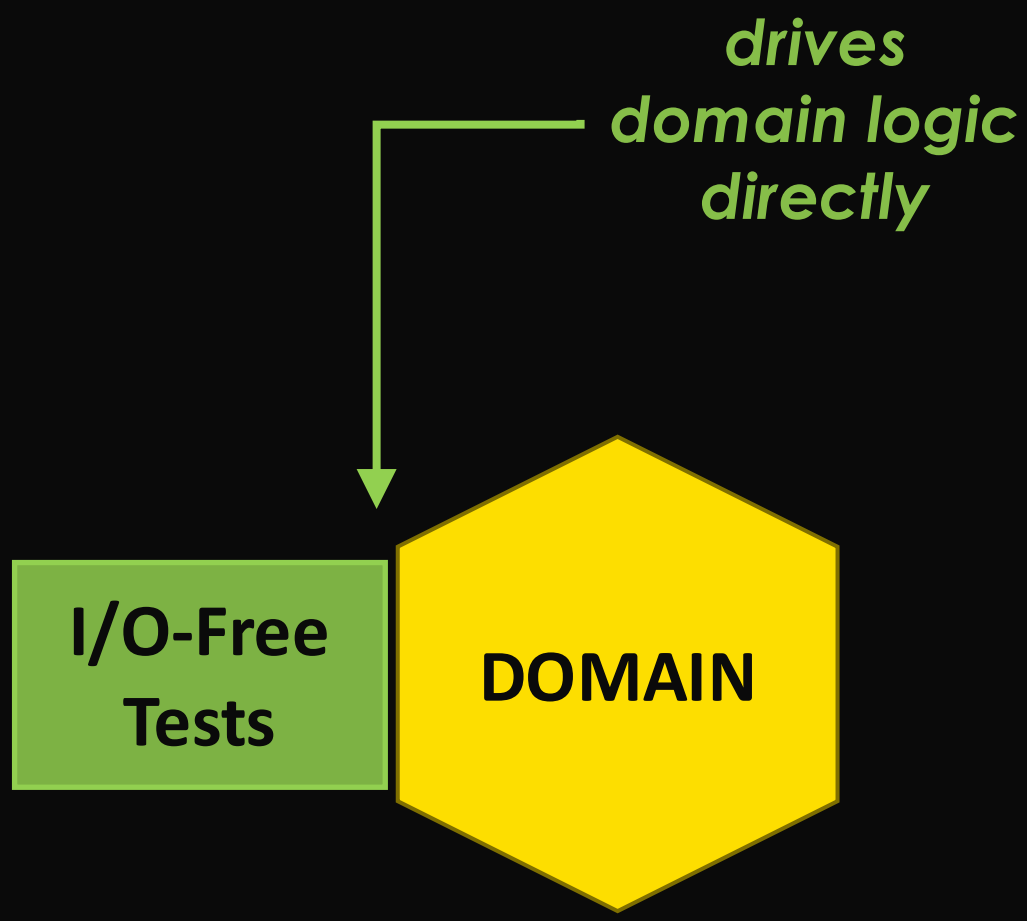

INBOUND
ADAPTER
CONCRETE
OUTBOUND
ADAPTERS
Zoom
Scheduler
application
layer
DOMAIN
Ensembles
Members
port
port
Web
User
Interface
Persistence
SendGrid
Emailer

Faster Test Feedback

Majority Fast, Some Slow, A Few Slowest

Types of Tests

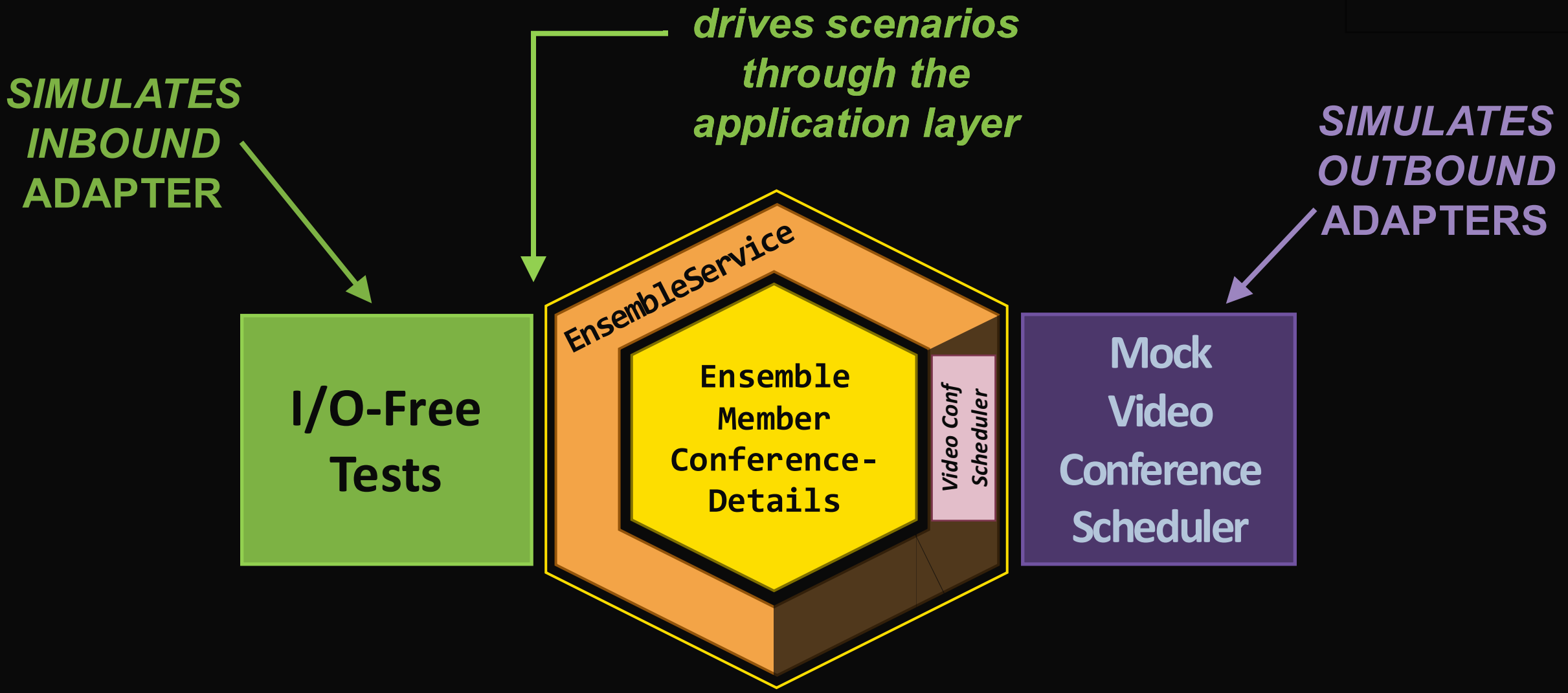
Hexagonal + Domain-Driven Design


drives
domain logic
directly
I/O-Free
Tests
DOMAIN

```java
@Test
void newEnsembleIsNotCompleted() throws Exception {
    Ensemble ensemble = new Ensemble("not completed", ZonedDateTime.now());

    assertThat(ensemble.isCompleted())
            .isFalse();
}

@Test
void whenCompletingEnsembleThenEnsembleIsCompleted() throws Exception {
    Ensemble ensemble = new Ensemble("completed", ZonedDateTime.now());

    ensemble.complete();

    assertThat(ensemble.isCompleted())
            .isTrue();
}

@Test
void whenCancelingScheduledEnsembleThenEnsembleIsCanceled() throws Exception {
    Ensemble ensemble = EnsembleFactory.withStartTimeNow();

    ensemble.cancel();

    assertThat(ensemble.isCanceled())
            .isTrue();
}
```

Testing Domain Directly

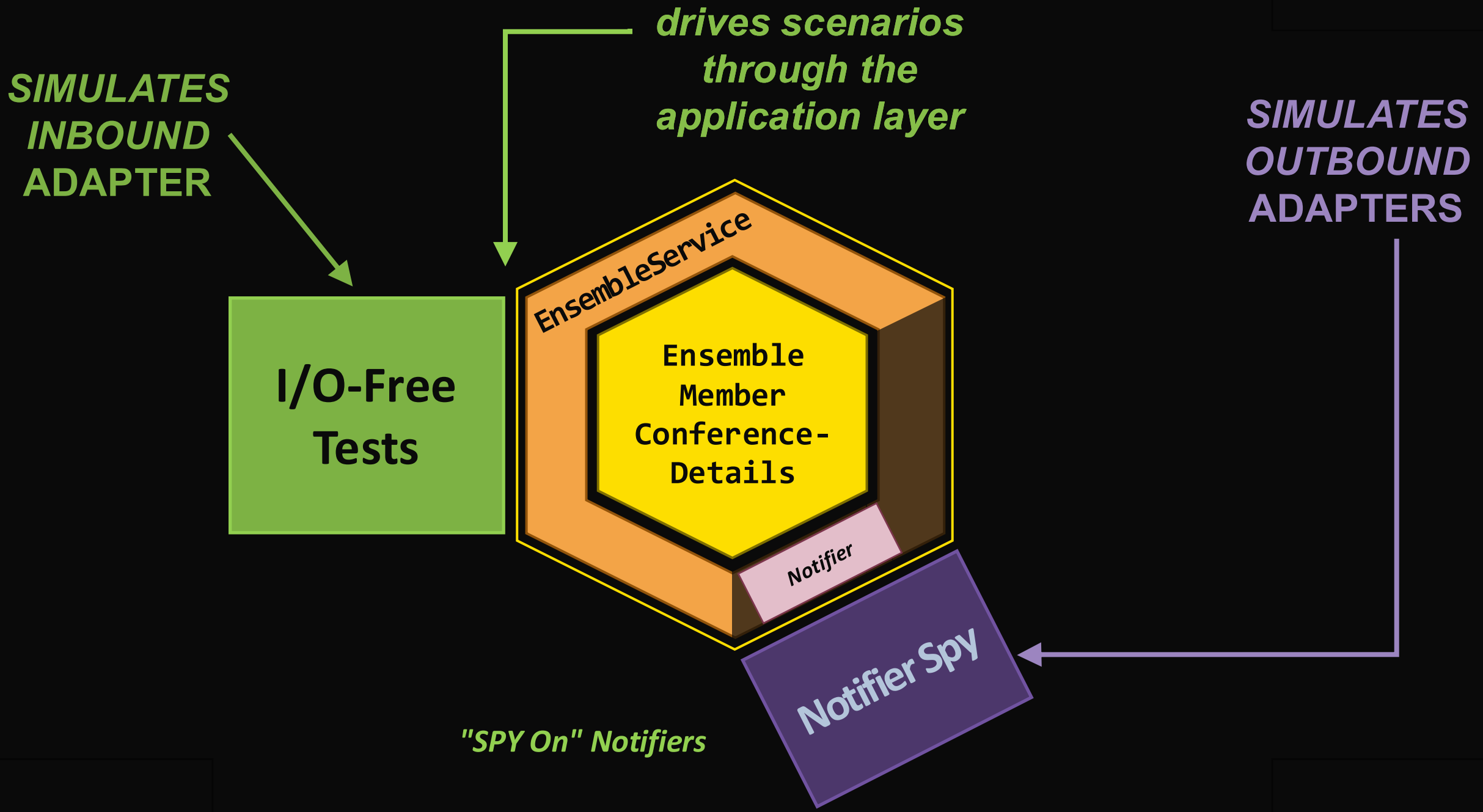
https://ted.dev/about


SIMULATES INBOUND ADAPTER
drives scenarios through the application layer
SIMULATES OUTBOUND ADAPTERS
I/O-Free Tests
EnsembleService
Ensemble Member Conference-Details
Video Conf Scheduler
Mock Video Conference Scheduler

```java
@Test
void canceledEnsembleDeletesVideoConferenceMeeting() throws Exception {
    Ensemble ensemble = new EnsembleBuilder()
            .withConferenceDetails("meetingId", "https://start.link", "https://join.link")
            .build();
    VideoConferenceScheduler succeedsIfMeetingIdMatchesEnsemble =
            new ZoomConferenceSchedulerDeletesExpectedMeetingId("meetingId");
    TestEnsembleServiceBuilder ensembleServiceBuilder =
            new TestEnsembleServiceBuilder()
                    .withVideoConferenceScheduler(succeedsIfMeetingIdMatchesEnsemble)
                    .saveEnsemble(ensemble);
    EnsembleId ensembleId = ensembleServiceBuilder.lastSavedEnsembleId();
    EnsembleService ensembleService = ensembleServiceBuilder.build();

    ensembleService.cancel(ensembleId);

    assertThat(ensemble.conferenceDetails())
            .isEqualTo(ConferenceDetails.DELETED);
}
```

Testing App Layer: Conference Scheduler

https://ted.dev/about


SIMULATES INBOUND ADAPTER
drives scenarios through the application layer
SIMULATES OUTBOUND ADAPTERS
I/O-Free Tests
EnsembleService
Ensemble Member Conference-Details
Notifier
Notifier Spy
"SPY On" Notifiers

```java
    @Test
    void whenEnsembleScheduledEnsembleOpenNotificationIsSent() throws Exception {
        MockEnsembleScheduledNotifier mockEnsembleScheduledNotifier =
                new MockEnsembleScheduledNotifier();
        EnsembleService ensembleService = new EnsembleService(
                new InMemoryEnsembleRepository(),
                new InMemoryMemberRepository(),
                mockEnsembleScheduledNotifier,
                new DummyVideoConferenceScheduler());

        ensembleService.scheduleEnsemble("Notifying Ensemble",
                ZonedDateTime.of(2021, 11, 10, 17, 0, 0, 0, ZoneOffset.UTC));

        mockEnsembleScheduledNotifier.verify();
    }
```

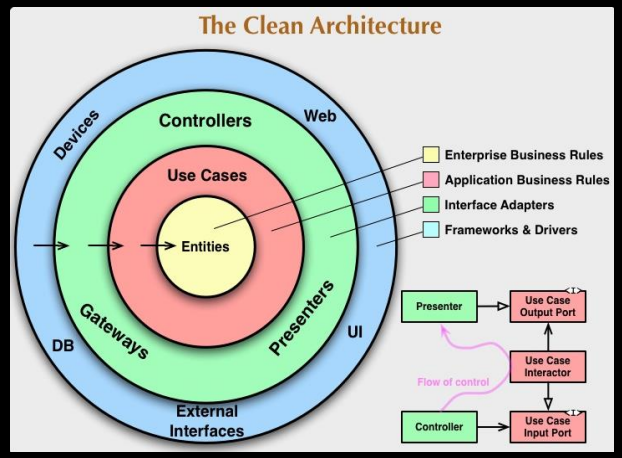
Testing App Layer: Notifier

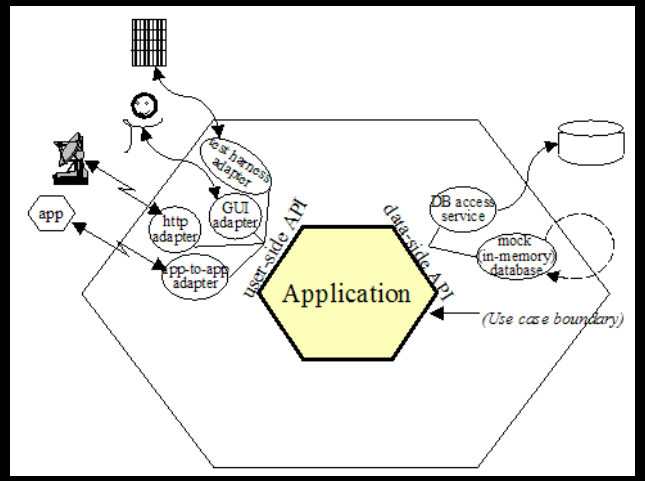
https://ted.dev/about

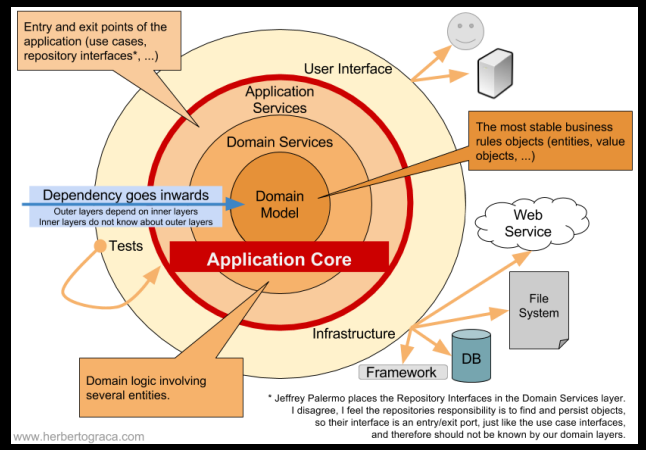
It's All Trade-offs

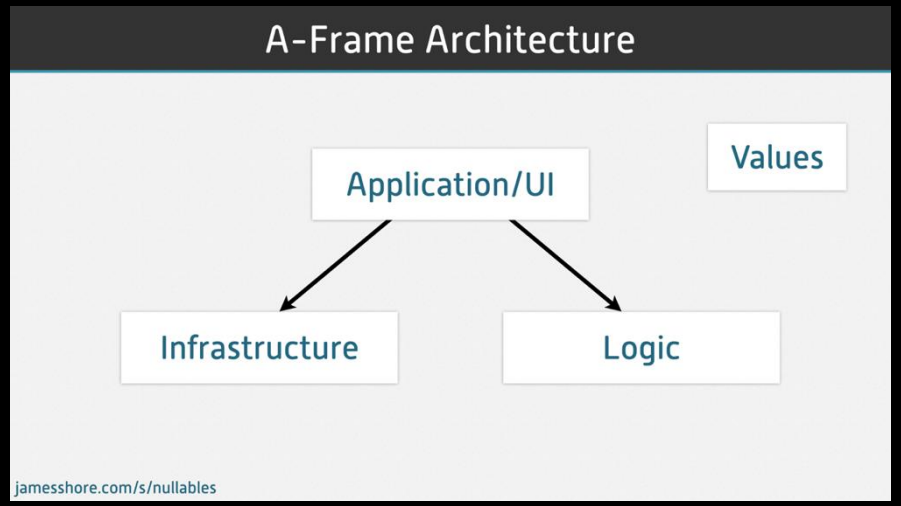
Increase Testability → May Lose "Simplicity"

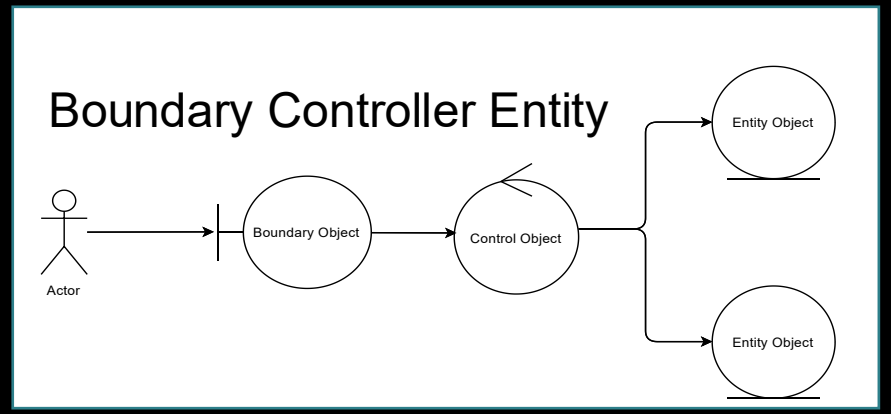
Use a Testable Architecture…

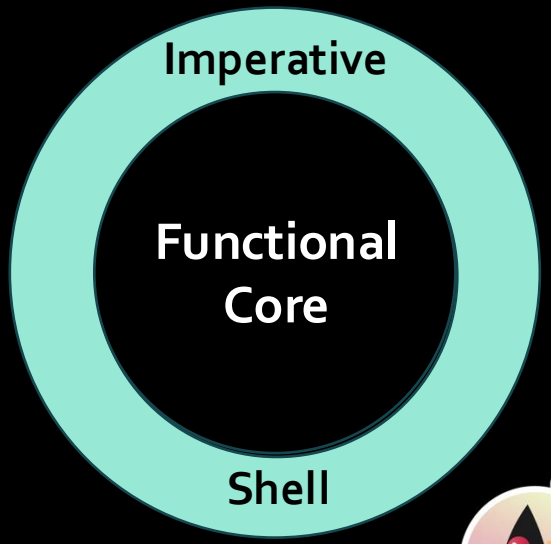












https://ted.dev/about

Ted M. Young
Java Trainer, Coach, & Live Coder

Get in touch: https://ted.dev/about

Want Your Code to be Easier to Test?

Keep 'em [I/O] Separated

Ted M. Young
Java Trainer, Coach, & Live Coder

Get in touch: https://**ted.dev**/about

Watch my talks and live coding:
Twitch: https://JitterTed.Stream
YouTube: https://JitterTed.TV

Thank You...

Source Code? Slides?
https://ted.dev/talks/