Tactical Domain-Driven Design with Java & Spring

Oliver Drotbohm / / odrotbohm info@odrotbohm.de

github.com/odrotbohm

odrotbohm (Oliver Drotbohm)

Oliver Drotbohm

odrotbohm · he/him

Frameworks & Architecture in the Spring engineering team, OpenSource enthusiast, all things Java, DDD, REST, software architecture, drums & music

Edit profile

3.9k followers · **32** following

- Spring Open Source Engineering
- Dresden, Germany
- 17:46 (UTC +02:00)
- info@odrotbohm.de
- www.odrotbohm.de
- @odrotbohm.de
- @odrotbohm@chaos.social
- odrotbohm
- in/odrotbohm

Sponsoring

Pinned

Customize your pins

spring-projects/spring-modulith Public
Modular applications with Spring Boot
Java · 1k · 173

xmolecules/jmolecules Public
Libraries to help developers express architectural abstractions in Java code
Java · 1.4k · 110

xmolecules/jmolecules-integrations Public
Technology integration for jMolecules
Java · 106 · 27

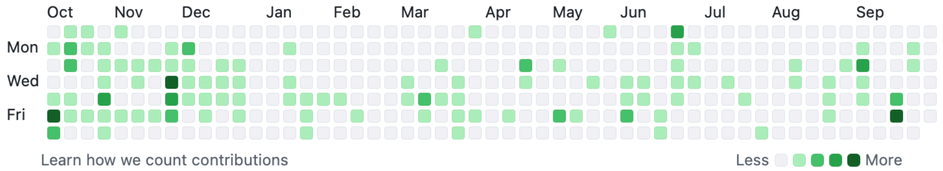
spring-restbucks Public
Implementation of the sample from REST in Practice based on Spring projects
Java · 1.3k · 428

spring-playground Public
A collection of tiny helpers for building Spring applications
Java · 102 · 11

lectures Public
Lecture scripts and slides I use during the Software Engineering course at TU Dresden
Java · 74 · 25

572 contributions in the last year

Contribution settings



Learn how we count contributions

Less More

2025
2024
2023
2022
2021
2020
2019
2018
2017
2016

@xmolecules @st-tu-dresden @FasterXML More

Activity overview

Contributed to
xmolecules/jmolecules-integrat... ,
odrotbohm/spring-modulith-pri... ,
odrotbohm/spring-restbucks
and 43 other repositories

Logistics

Part I – DDD Building Blocks

- Value Objects
- Entities
- Aggregates
- Repositories

- jMolecules
- Verification
- Integration
- Persistence Boilerplate

Part II – Modularity

- Establishing Modules
- Integration Testing
- Documentation

- Module interaction
- Consistency

Related Talks

- **What's new in Spring Modulith?**

  Thursday, Oct. 9th – 13:50, Room 5

- **Domain-driven? Why Hexagonal and Onion Architecture are Answers to the Wrong Question**

  Thursday, Oct. 9th – 16:30, Room 5

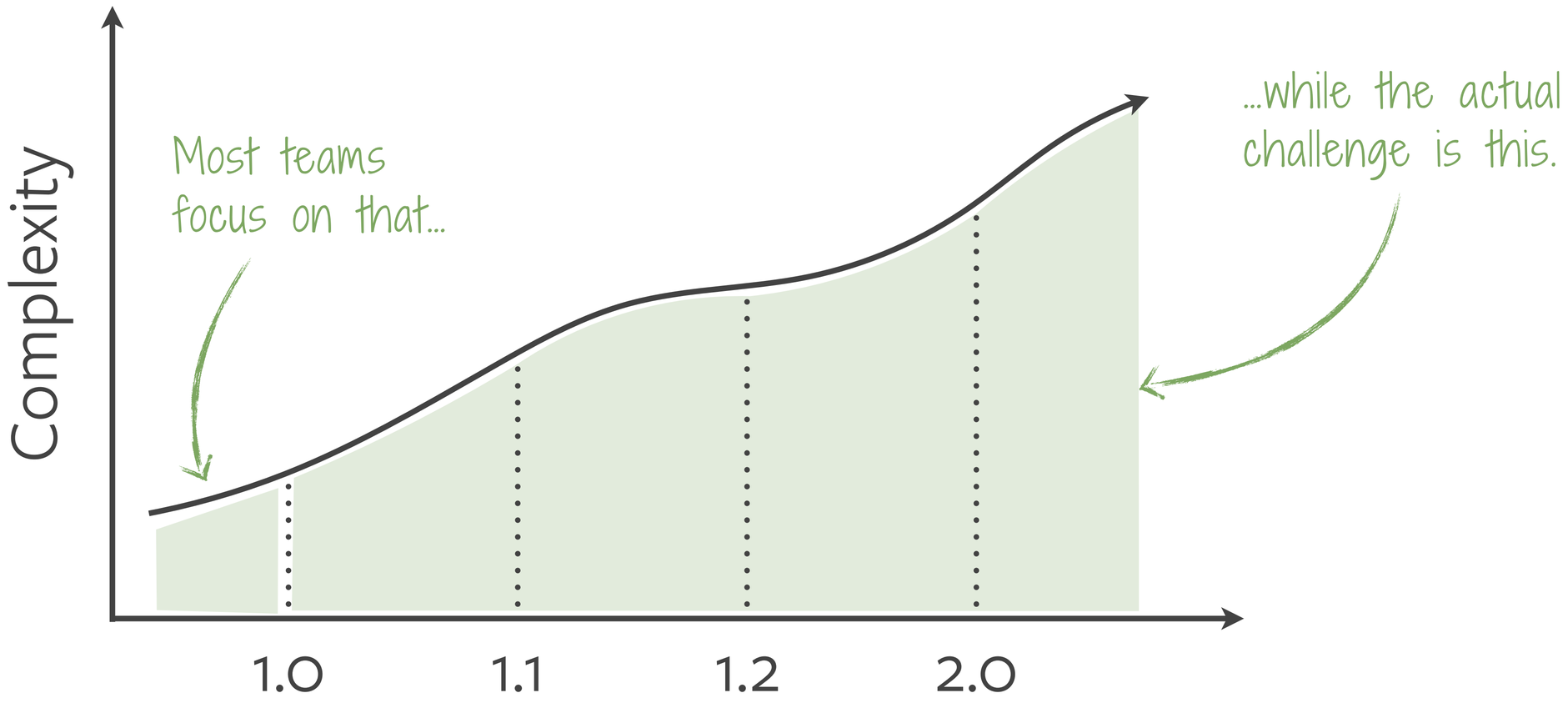
- **Spring Team BOF**

  Thursday, Oct. 9th – 17:40, BOF 1

Source Code

https://github.com/odrotbohm/tactical-ddd-workshop
https://github.com/odrotbohm/arch-evident-spring

We want to build evolvable systems.


Complexity
Most teams
focus on that...
...while the actual
challenge is this.
1.0
1.1
1.2
2.0

Chunking

Hierarchization

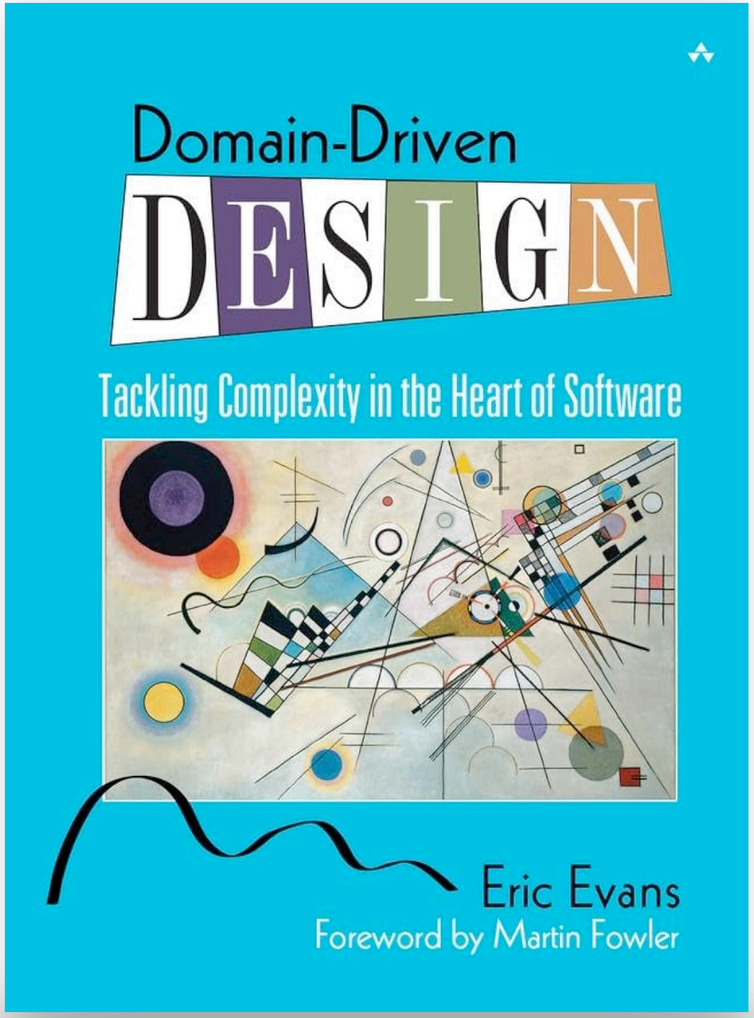
Pattern languages

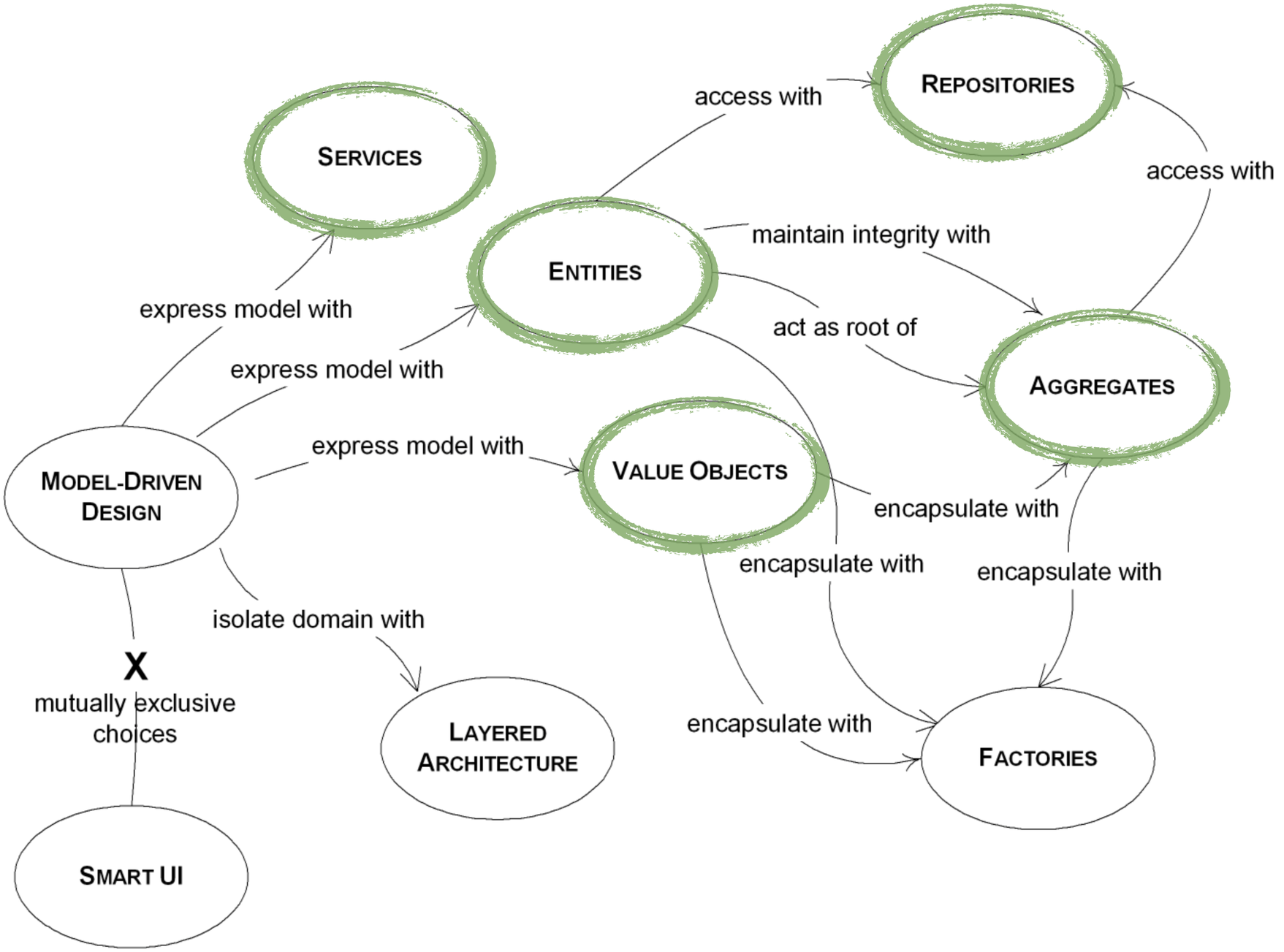
Chunking

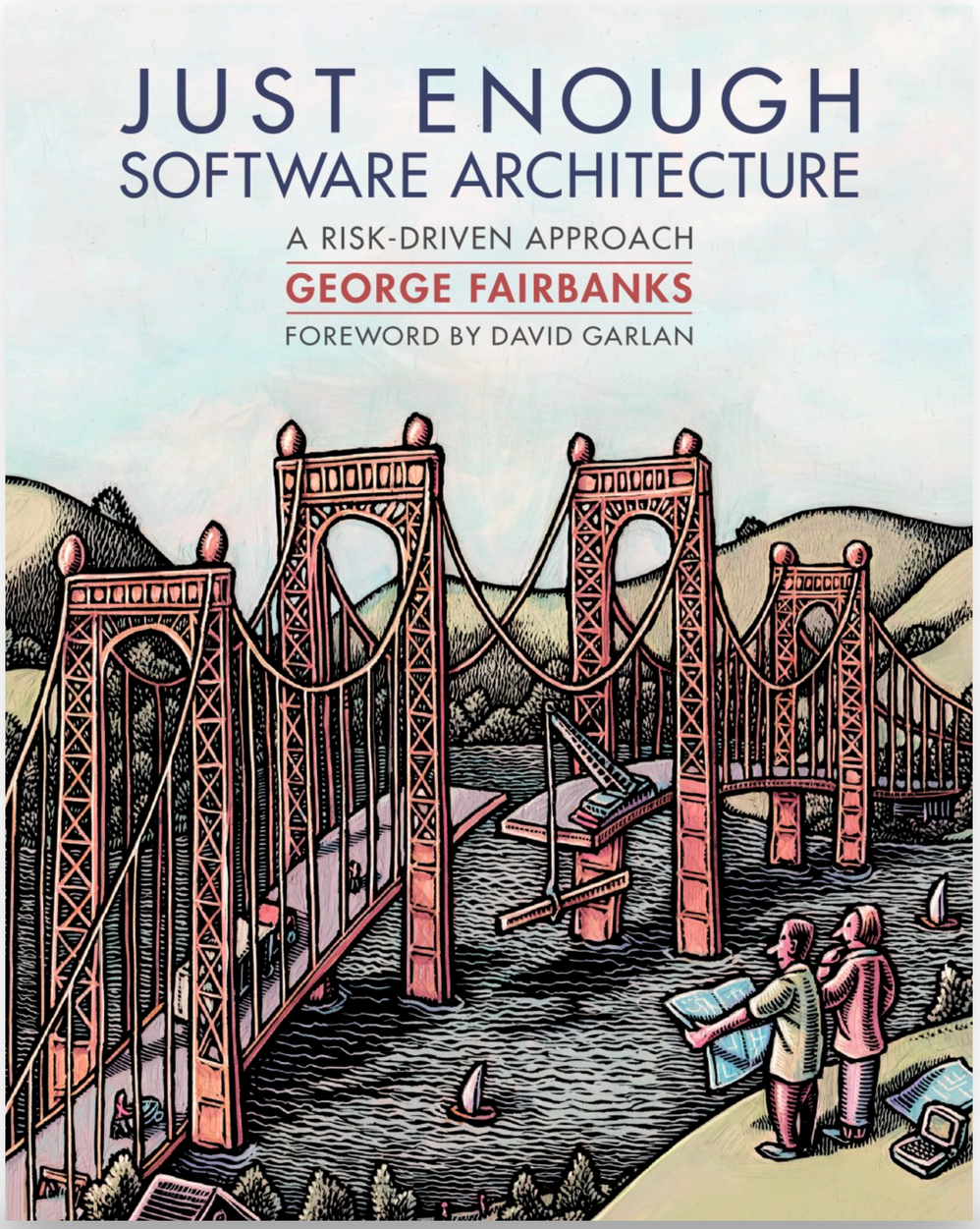
Hierarchization

Pattern languages


WPS WORKPLACE SOLUTIONS
dpunkt.verlag
Carola Lilienthal
Sustainable Software Architecture
Analyze and Reduce Technical Debt


Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler


REPOSITORIES
access with
SERVICES
access with
ENTITIES
maintain integrity with
express model with
act as root of
express model with
AGGREGATES
express model with
MODEL-DRIVEN DESIGN
VALUE OBJECTS
encapsulate with
encapsulate with
encapsulate with
isolate domain with
X
mutually exclusive choices
LAYERED ARCHITECTURE
encapsulate with
FACTORIES
SMART UI


JUST ENOUGH
SOFTWARE ARCHITECTURE
A RISK-DRIVEN APPROACH
GEORGE FAIRBANKS
FOREWORD BY DAVID GARLAN

Architecturally-Evident Code

Architecturally-Evident Code?

Extensional

Components / Modules

Invoicing,
Shipment

Deployables / Build modules / Packages

Domain language

EmailAddress,
ZipCode

Classes, methods, fields

Intensional

Concepts & Rules

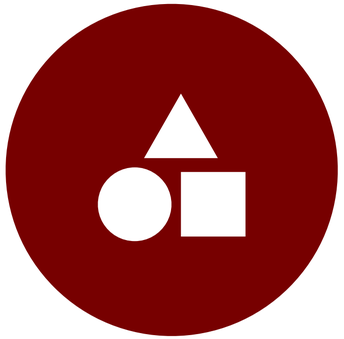
ValueObject,
Entity,
Aggregate

Layers,
Rings

Naming conventions

What else? 🤔

User Code

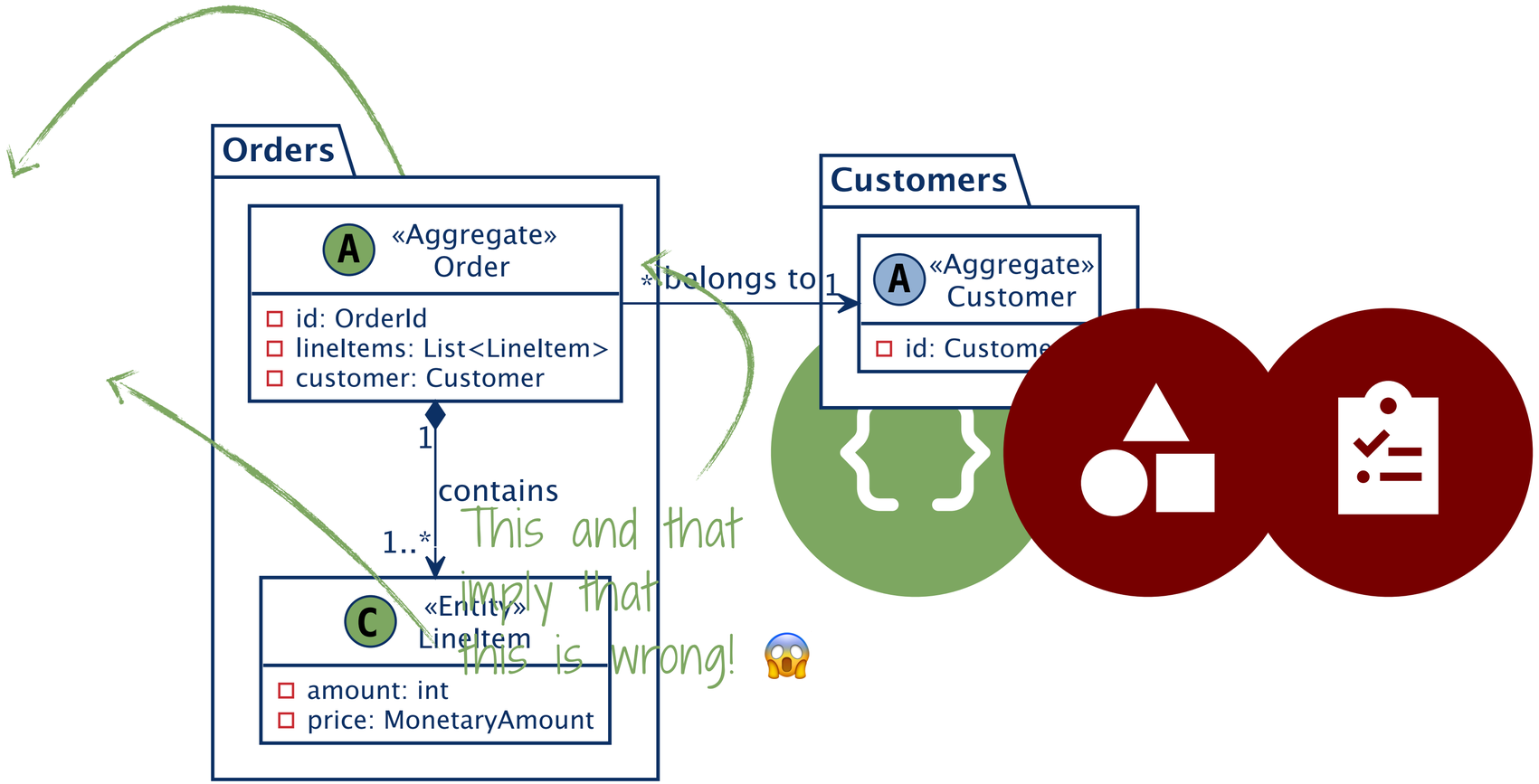


Concepts

Code

Architecture

Technology

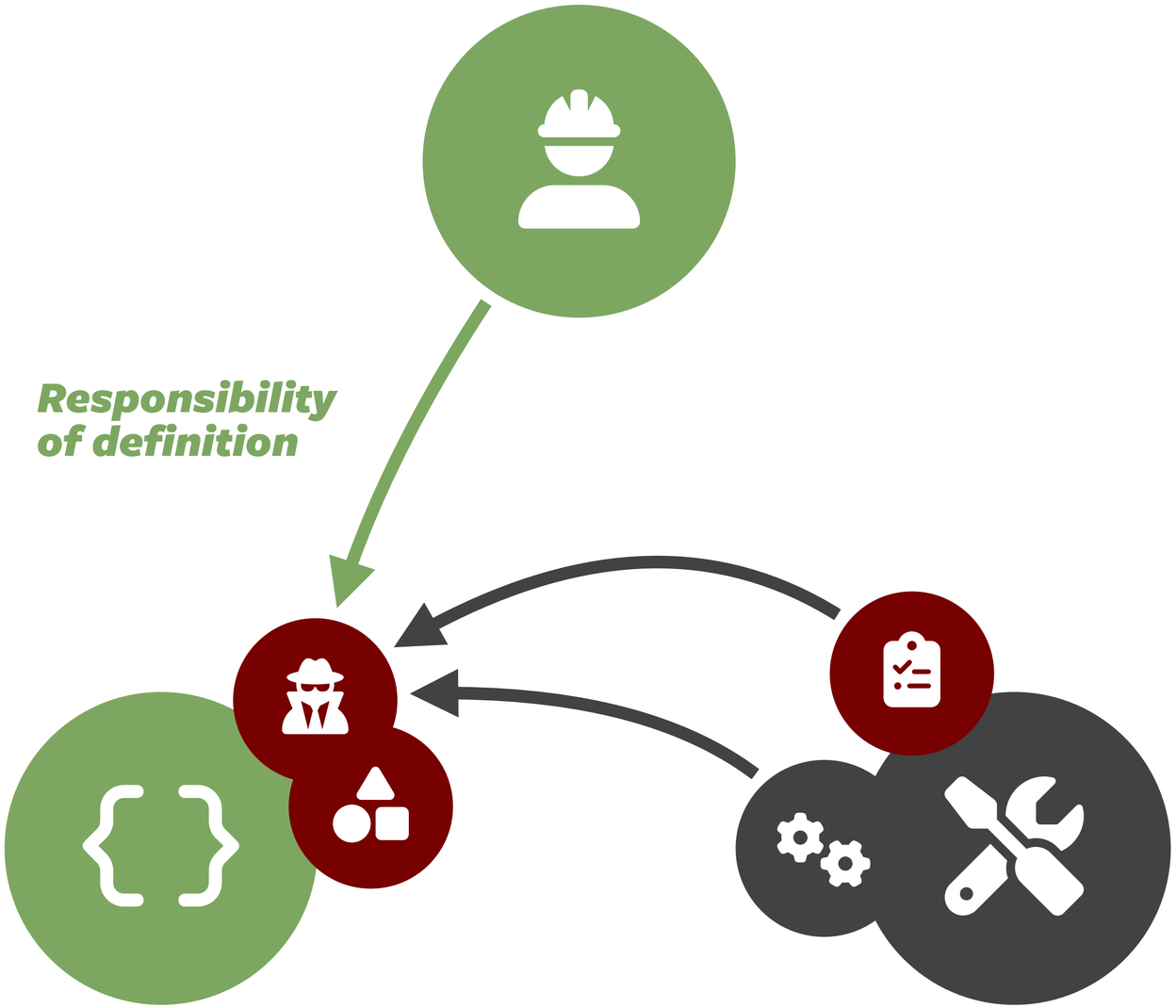


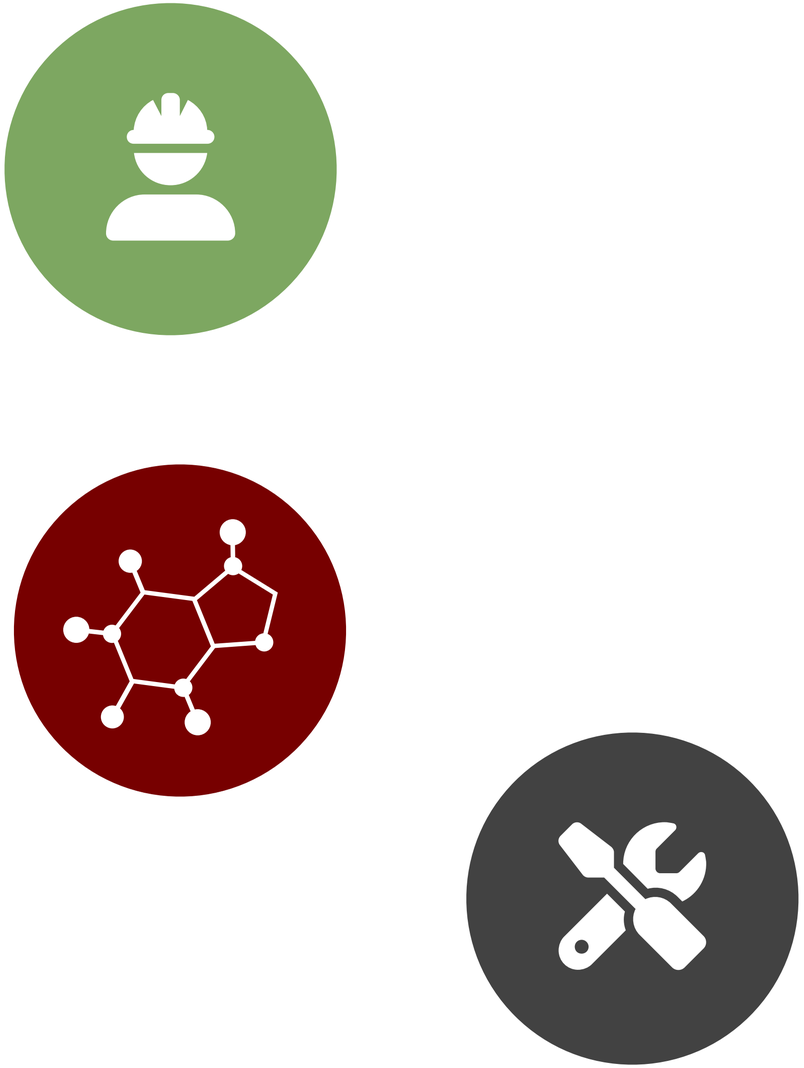
A simple Aggregate arrangement

Code

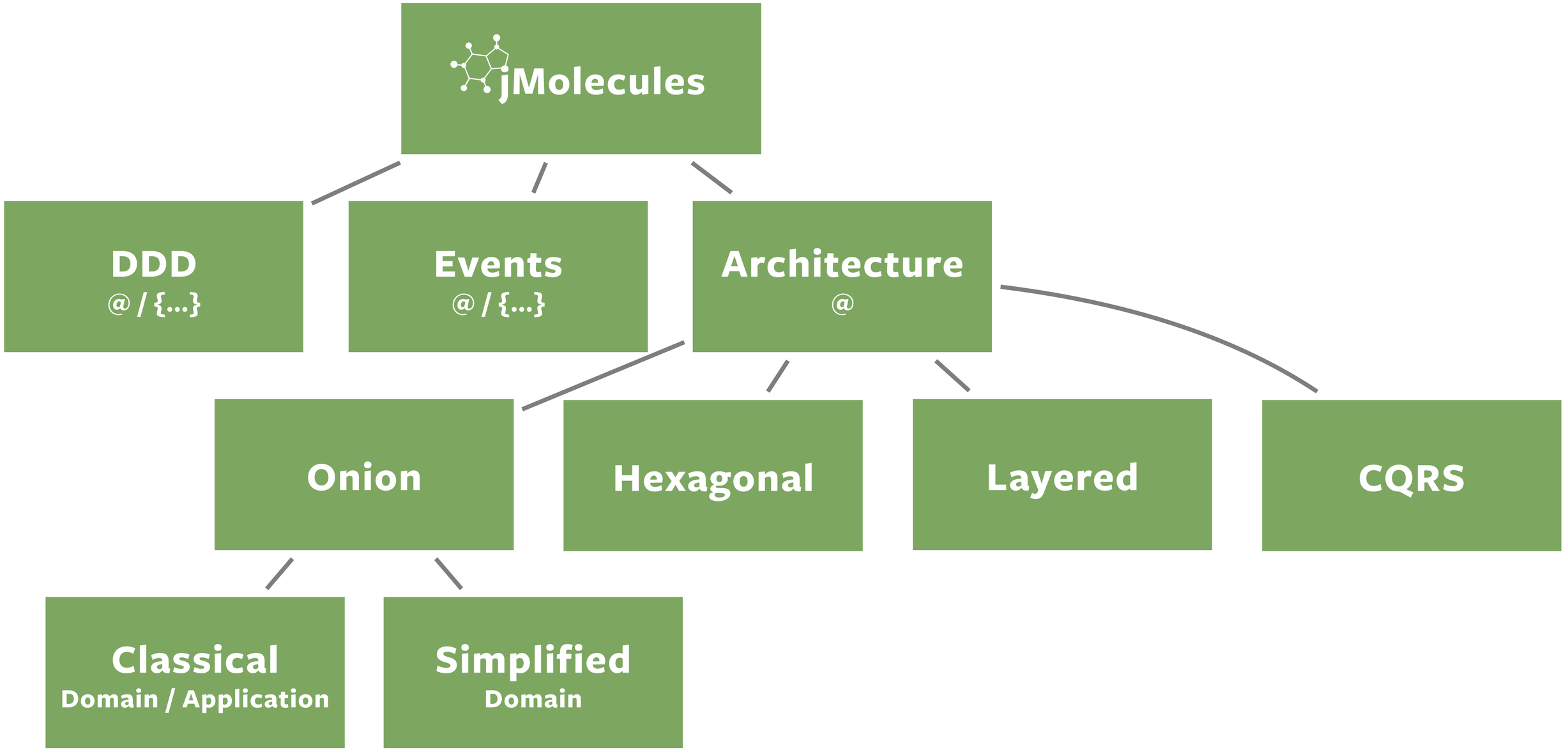
Architecture

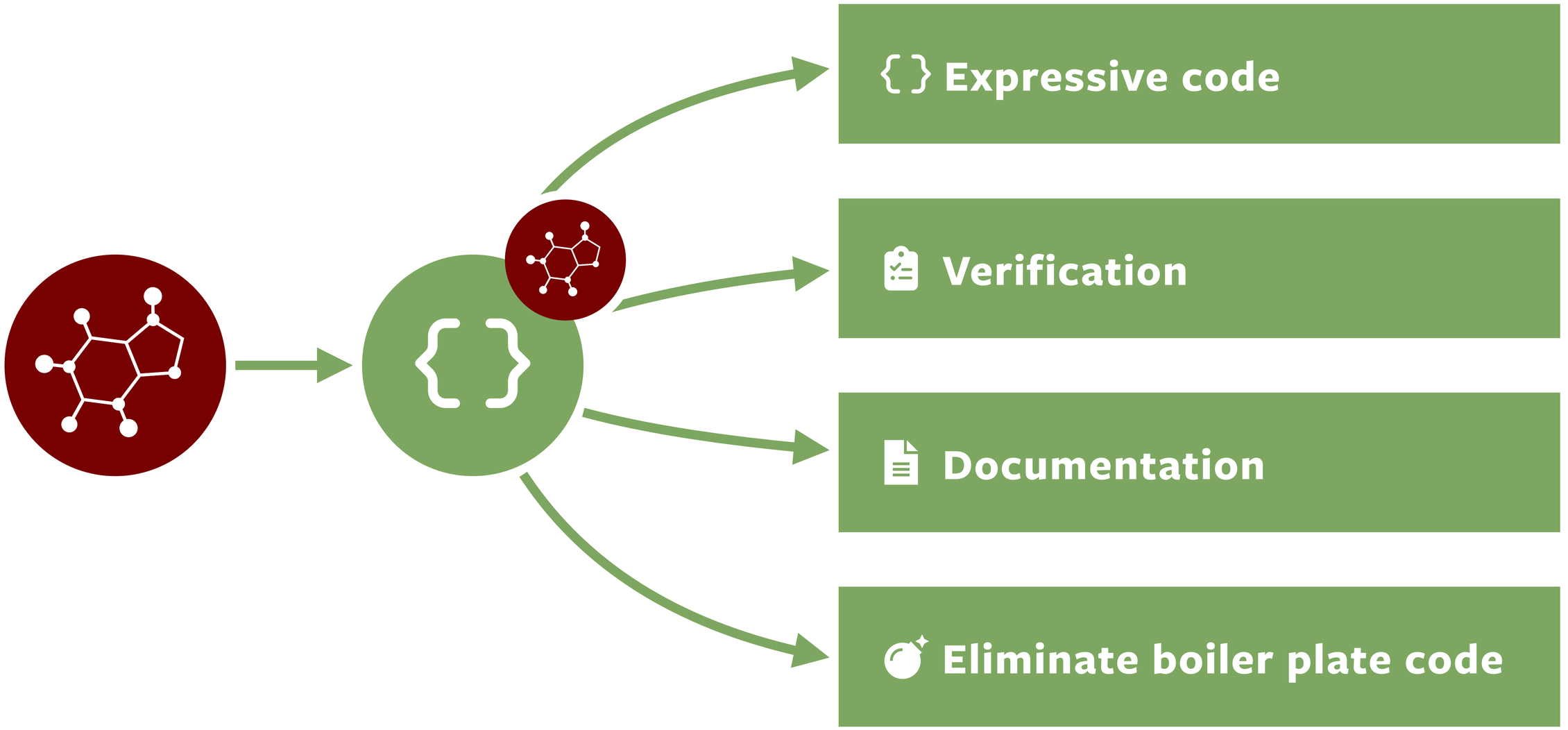
Technology


Responsibility
of definition



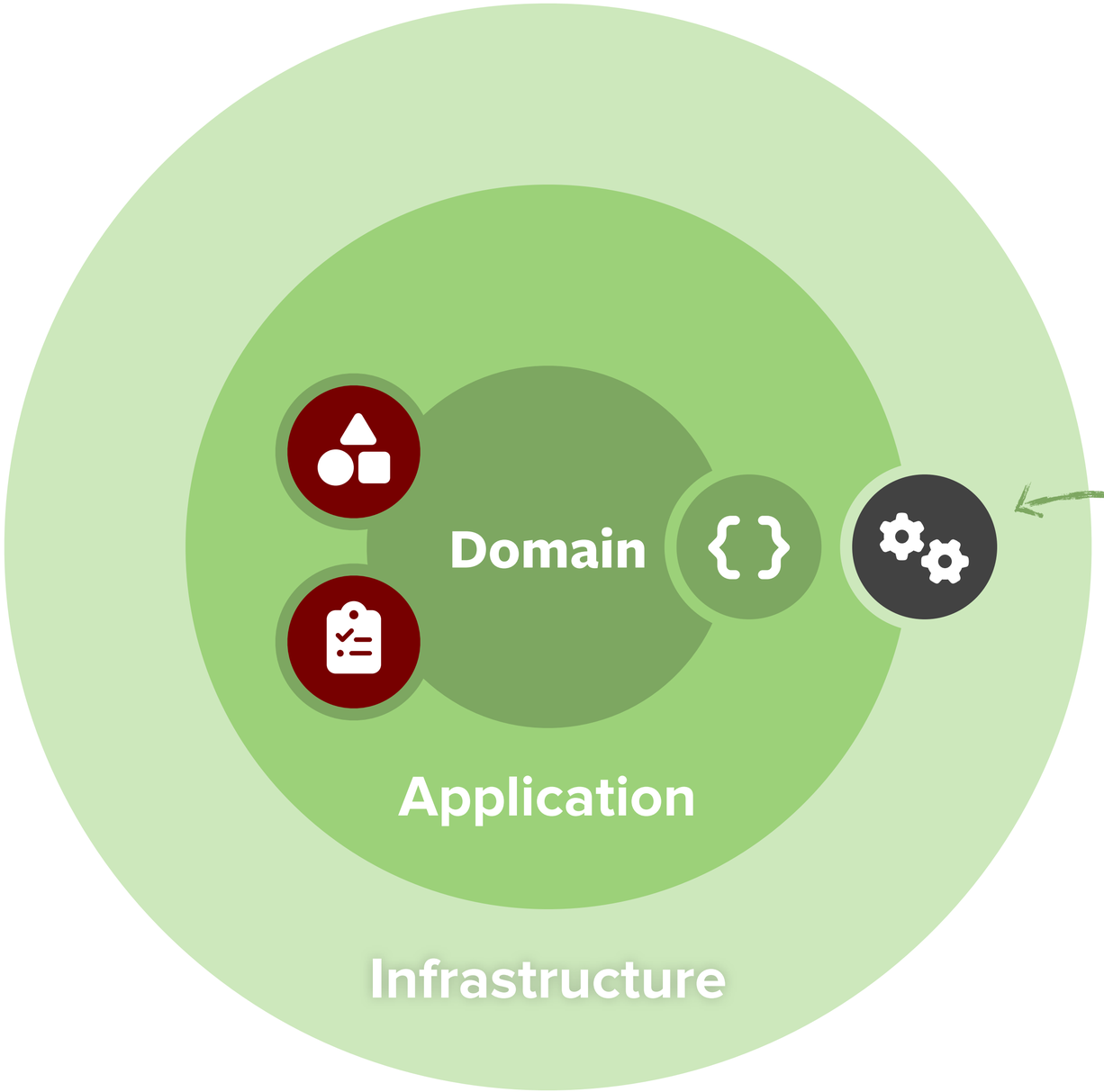
jMolecules


jMolecules
DDD
@ / {...}
Events
@ / {...}
Architecture
@
Onion
Hexagonal
Layered
CQRS
Classical
Domain / Application
Simplified
Domain


Expressive code
Verification
Documentation
Eliminate boiler plate code

Demo

Eliminate boilerplate



- Spring Framework
- JPA
- Jackson

Persistent model

VS.

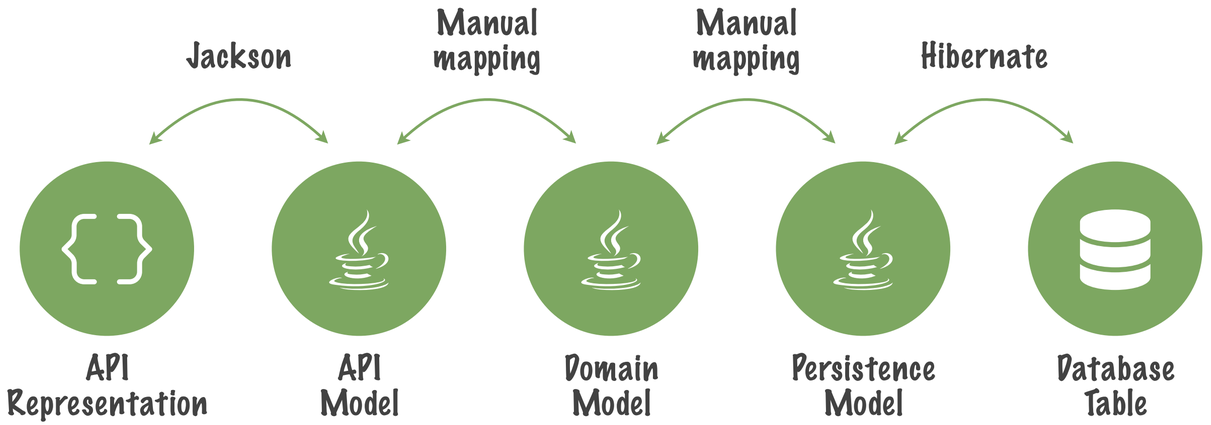
Dedicated persistence model

```
@Entity
class Order { … }
```

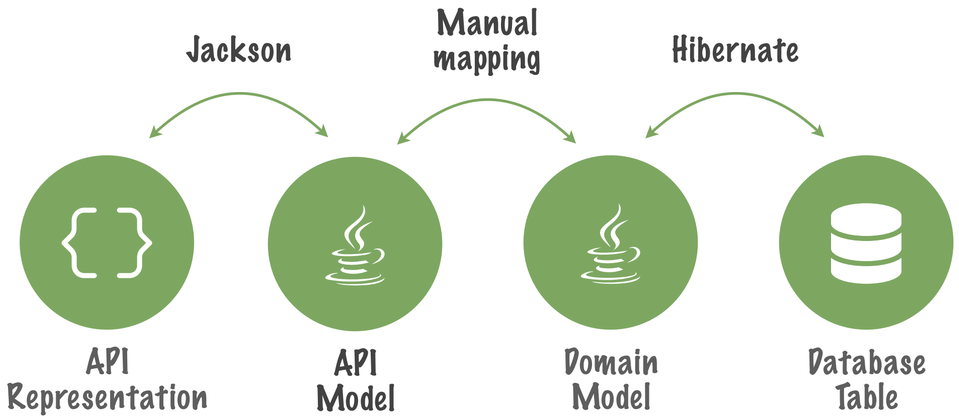
```
class Order { … }
```

```
@Entity
class JpaOrder { … }
```

Some mapping code, somewhere…


Jackson
Manual mapping
Manual mapping
Hibernate
API Representation
API Model
Domain Model
Persistence Model
Database Table


Legacy applications
Modern applications
Shared ownership
Team ownership


Jackson
Manual mapping
Hibernate
API Representation
API Model
Domain Model
Database Table

```java
@Entity
@NoArgsConstructor(force = true)
@EqualsAndHashCode(of = "id")
@Table(name = "SAMPLE_ORDER")
@Getter
public class Order {

  private final @EmbeddedId OrderId id;

  @OneToMany(cascade = CascadeType.ALL, …)
  private List<LineItem> lineItems;
  private CustomerId customerId;

  public Order(CustomerId customerId) {
    this.id = OrderId.of(UUID.randomUUID());
    this.customerId = customerId;
  }

  @Value
  @RequiredArgsConstructor(staticName = "of")
  @NoArgsConstructor(force = true)
  public static class OrderId implements Serializable {
    private static final long serialVersionUID = …;
    private final UUID orderId;
  }
}
```

Demo

```
@Entity
@NoArgsConstructor(force = true)
@EqualsAndHashCode(of = "id")
@Table(name = "SAMPLE_ORDER")
@Getter
public class Order {

  private final @EmbeddedId OrderId id;

  @OneToMany(cascade = CascadeType.ALL)
  private List<LineItem> lineItems;
  private CustomerId customerId;

  public Order(Customer customer) {
    this.id = OrderId.of(UUID.randomUUID());
    this.customerId = customer.getId();
  }

  @Value
  @RequiredArgsConstructor(staticName = "of")
  @NoArgsConstructor(force = true)
  public static class OrderId implements Serializable {
    private static final long serialVersionUID = …;
    private final UUID orderId;
  }
}
```

```
@Table(name = "SAMPLE_ORDER")
@Getter
public class Order implements AggregateRoot<Order, OrderId> {

  private final OrderId id;
  private List<LineItem> lineItems;
  private Association<Customer, CustomerId> customer;

  public Order(Customer customer) {
    this.id = OrderId.of(UUID.randomUUID());
    this.customer = Association.forAggregate(customer);
  }

  @Value(staticConstructor = "of")
  public static class OrderId implements Identifier {
    private final UUID orderId;
  }
}
```

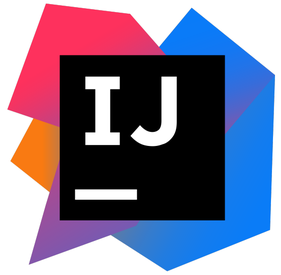
Architectural and Design Concepts...

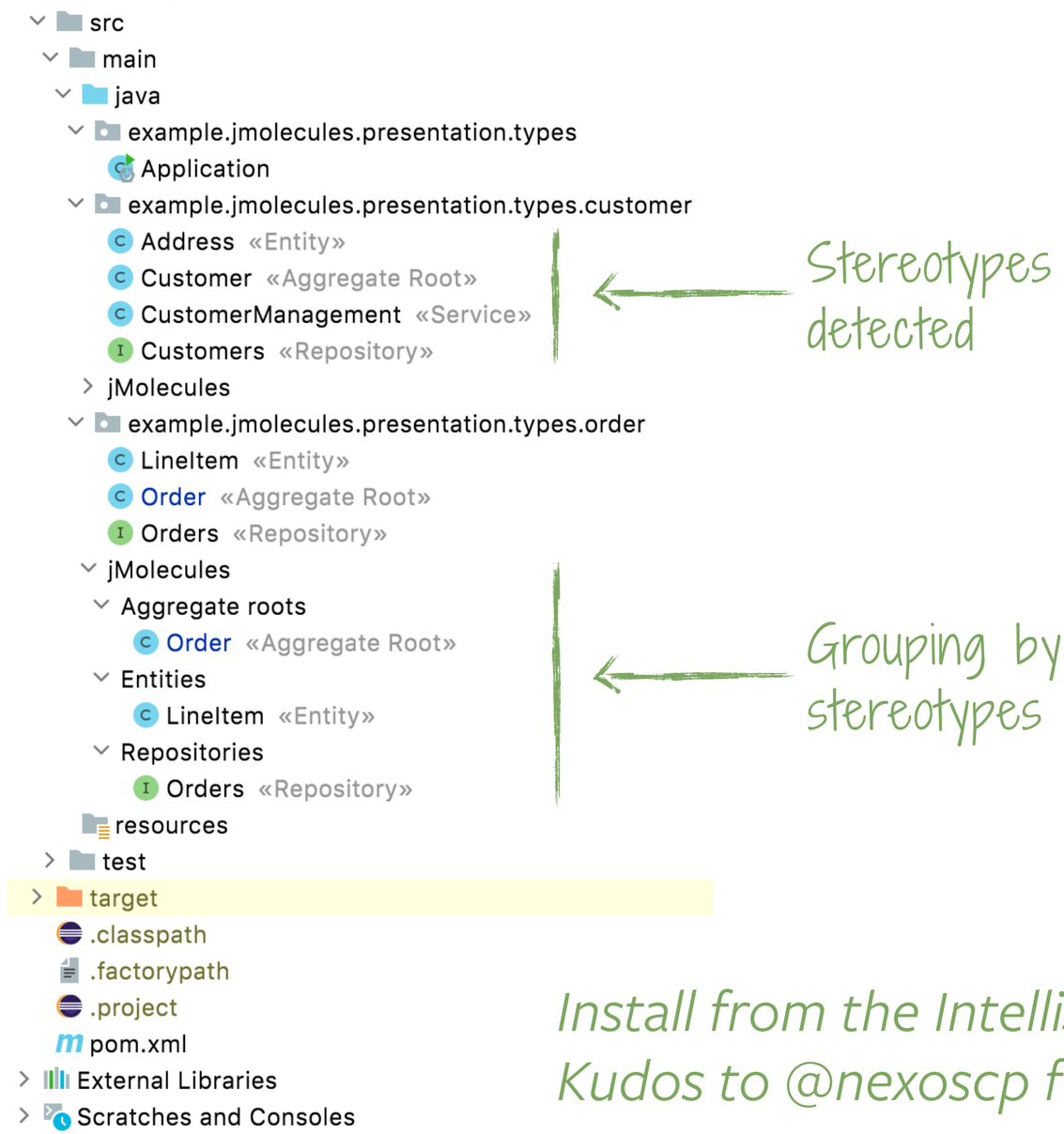
... are explicitly expressed in the code.

... are predefined based on established terms.

... are defined by jMolecules (concepts)
and tool integration (rules).

IDE support



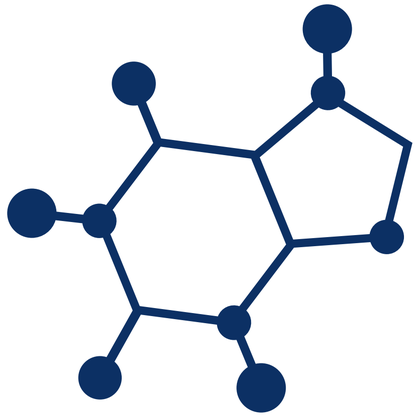


Stereotypes detected

Grouping by stereotypes

Install from the IntelliJ IDEA plugin portal.
Kudos to @nexoscp for the contributions!

VSCode



jMolecules

Logistics

Part I – DDD Building Blocks

- Value Objects
- Entities
- Aggregates
- Repositories

- jMolecules
- Verification
- Integration
- Persistence Boilerplate

Part II – Modularity

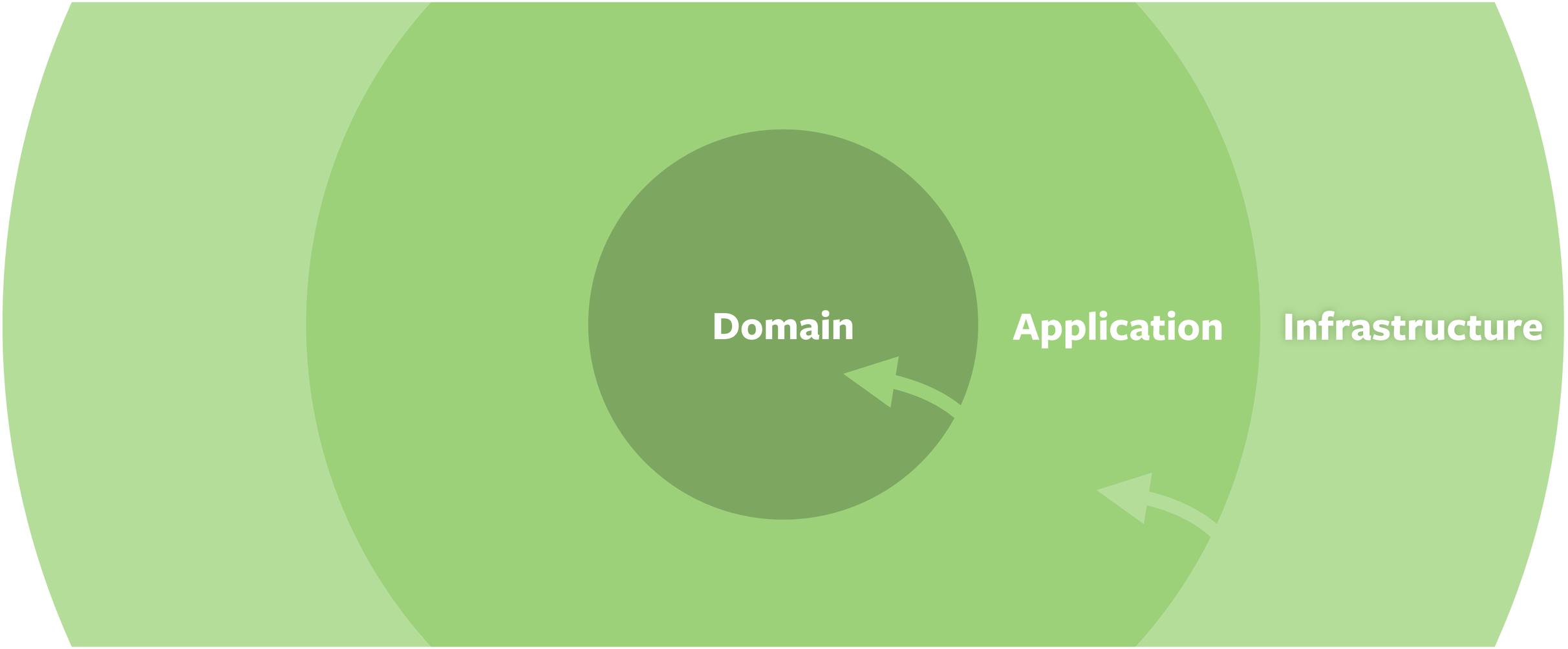
- Establishing Modules
- Integration Testing
- Documentation

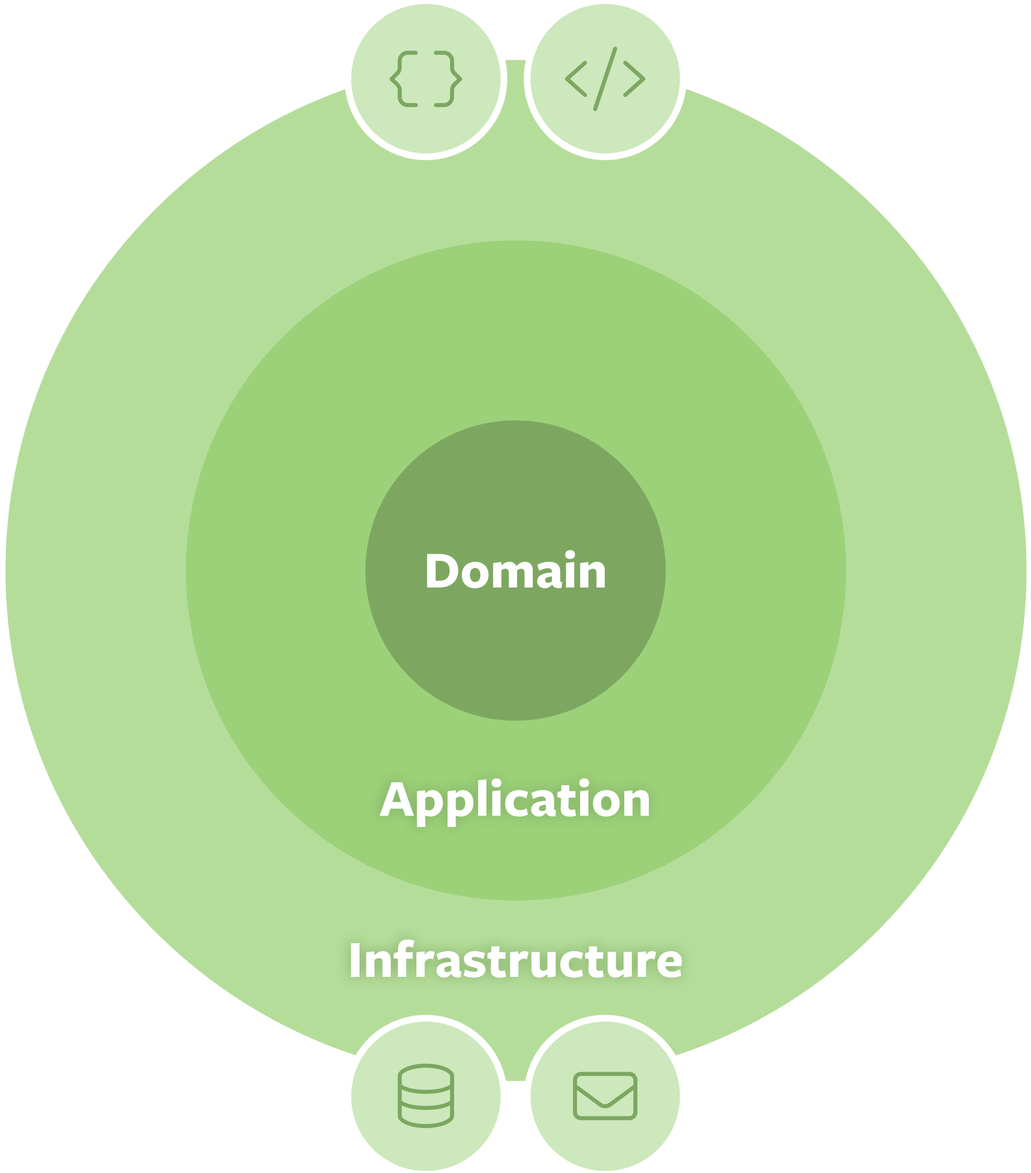
- Module interaction
- Consistency

Spring Modulith

https://projects.spring.io/spring-modulith

Modular Architecture

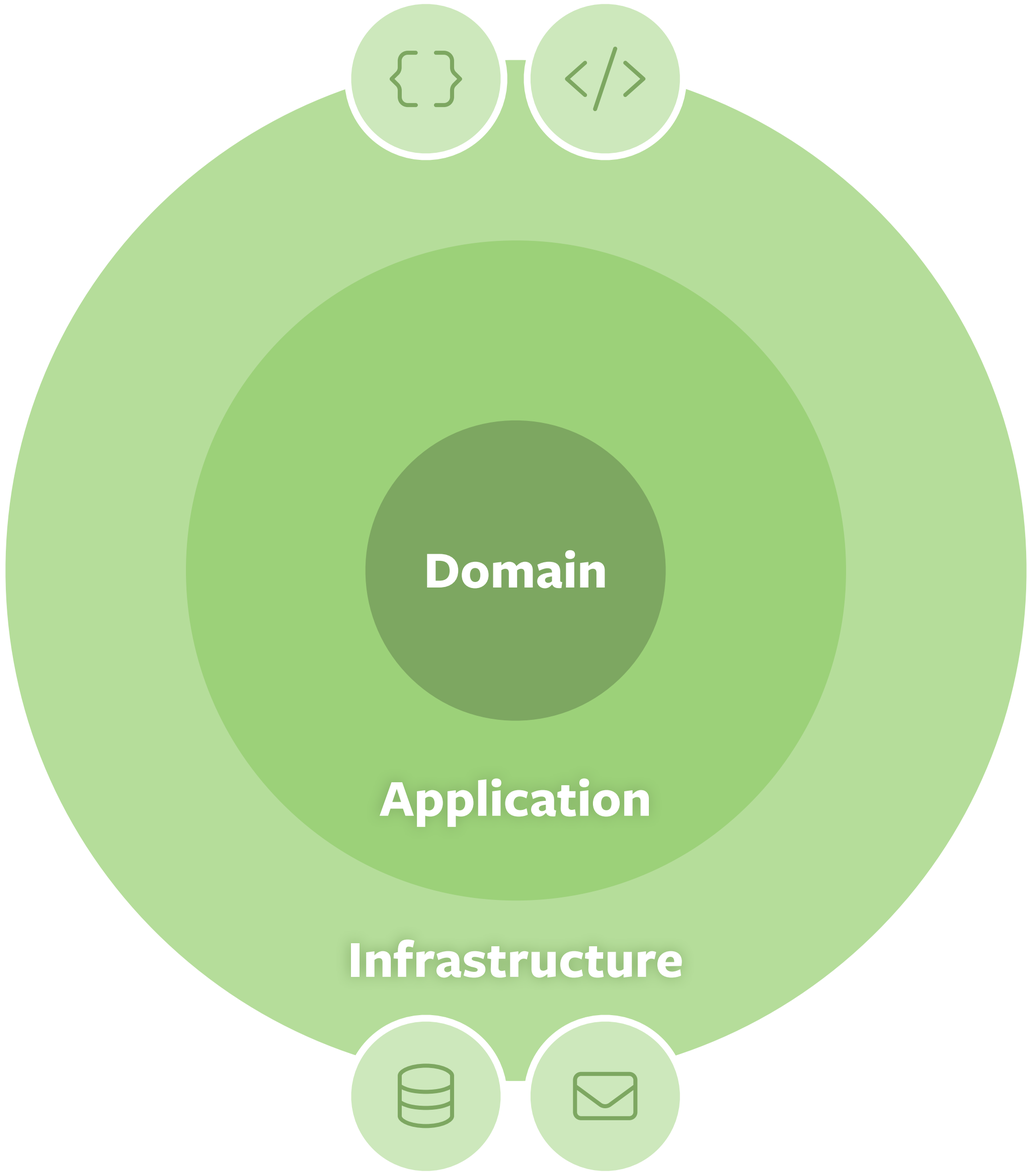

Domain
Application
Infrastructure

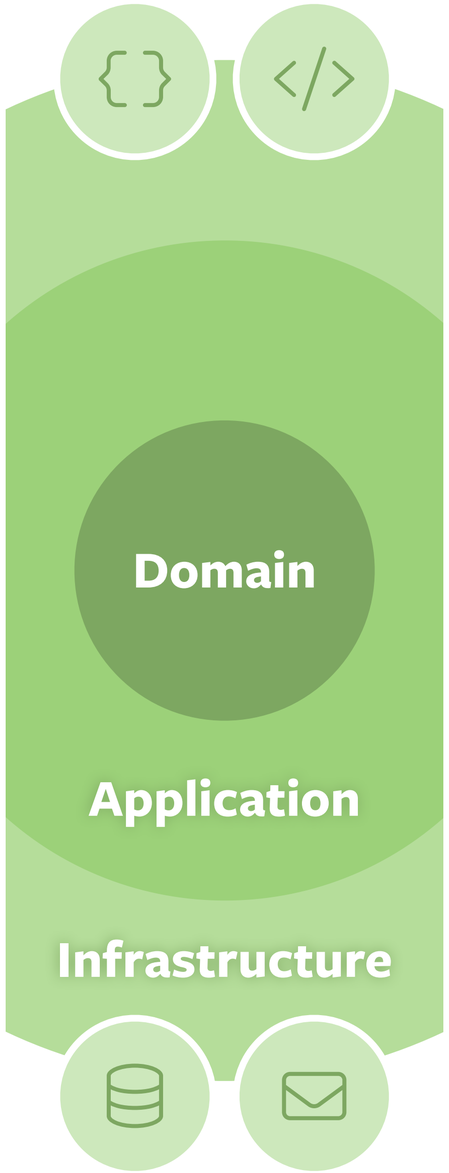

Domain
Application
Infrastructure

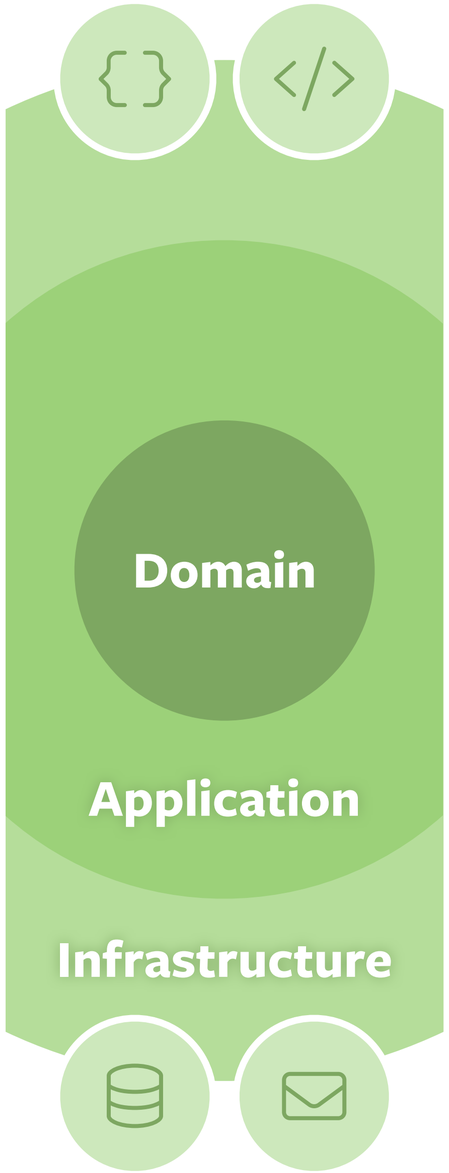
Onion-/Hexagonal Architecture

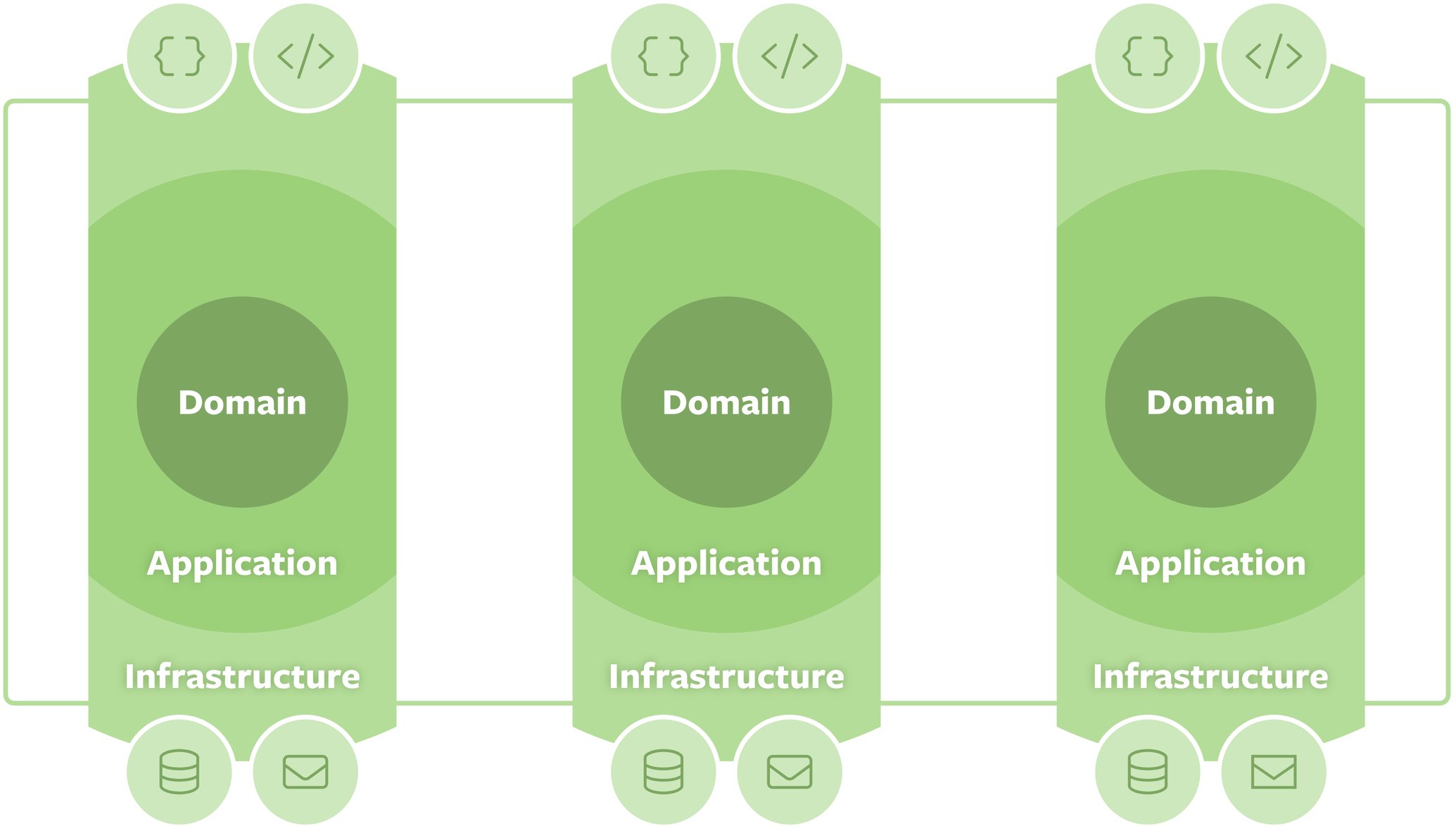
≠

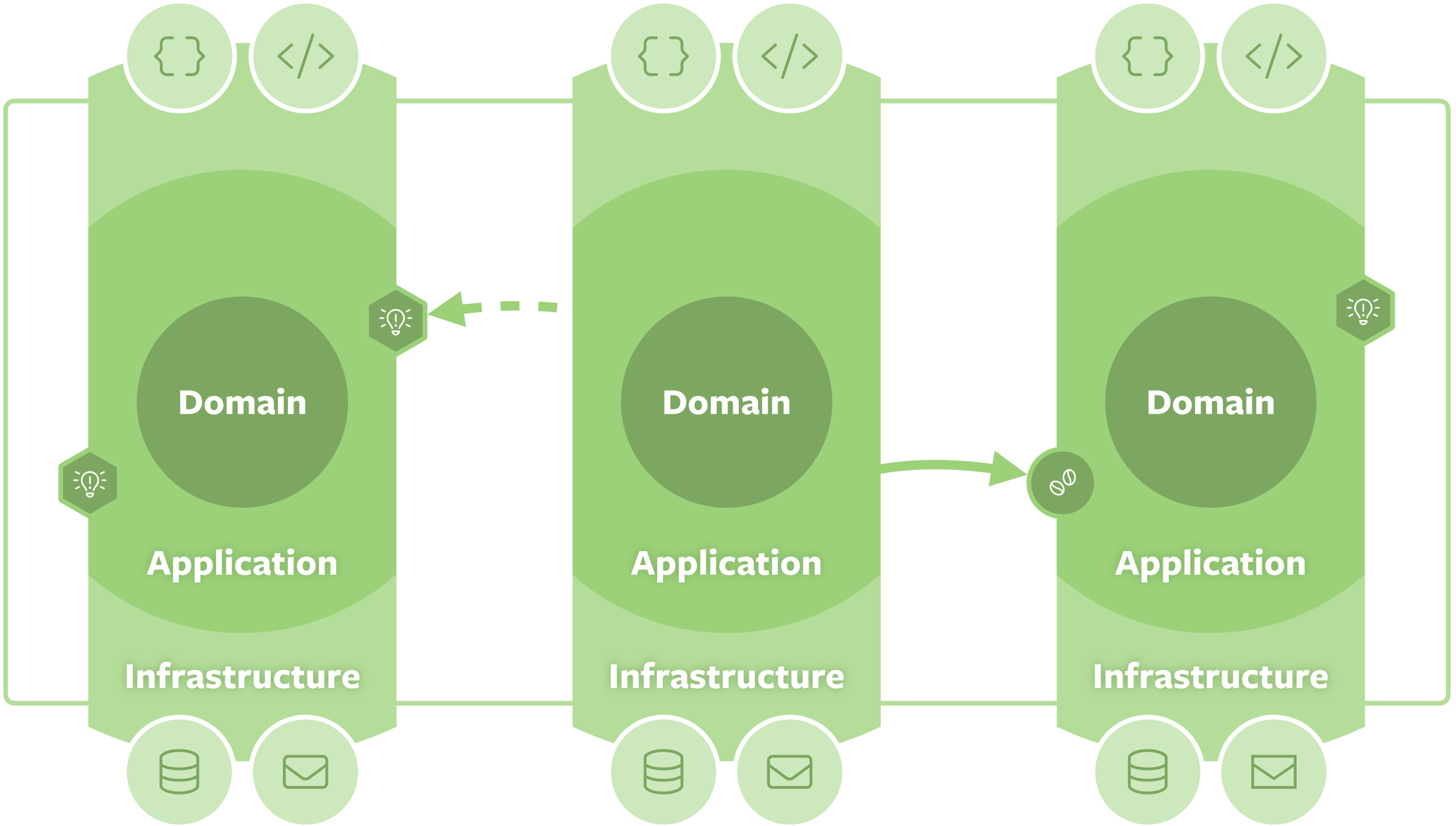
Modularization


Domain
Application
Infrastructure

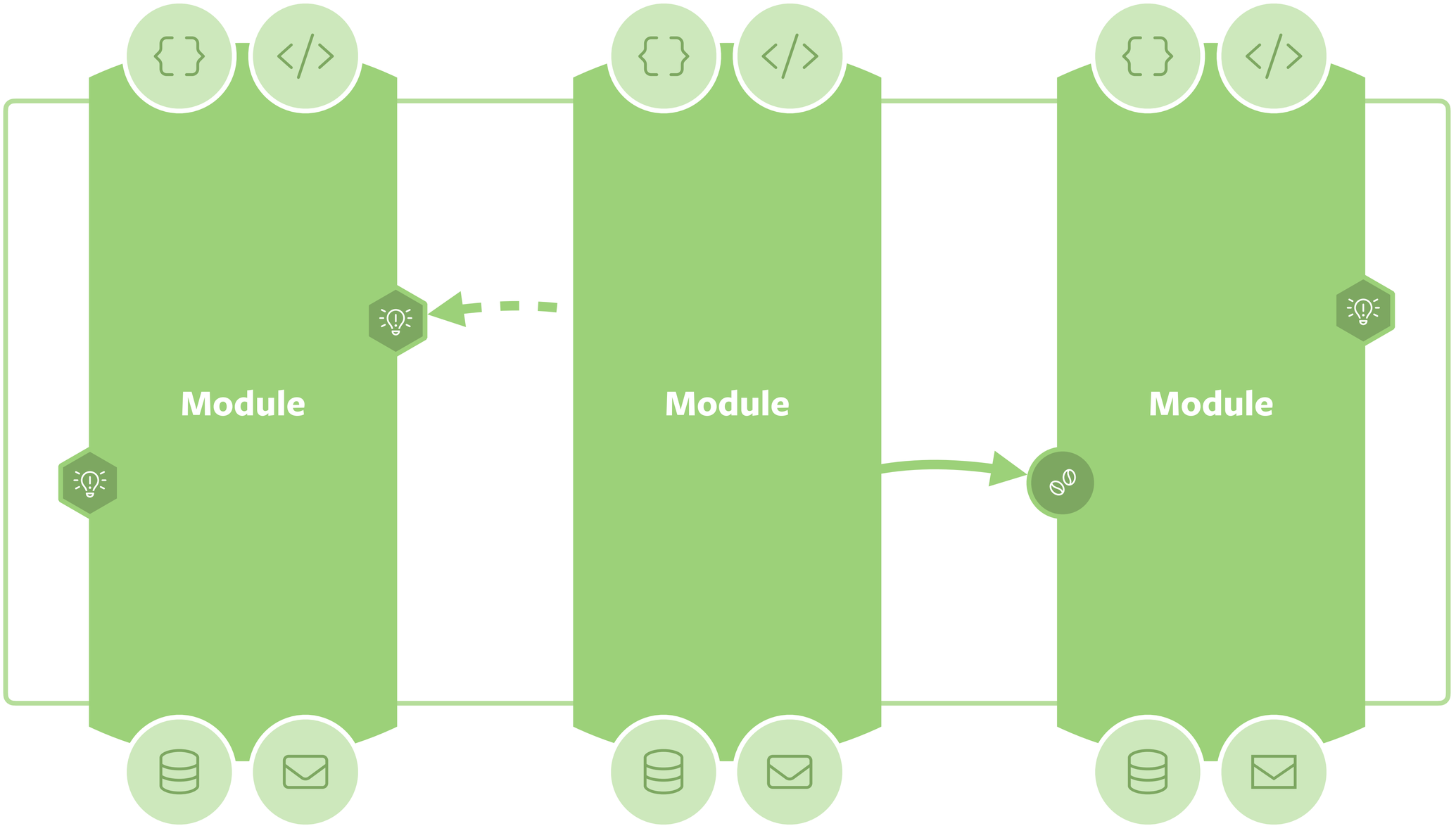

Domain
Application
Infrastructure


Domain
Application
Infrastructure


Domain
Application
Infrastructure
Domain
Application
Infrastructure
Domain
Application
Infrastructure



https://odrotbohm.de/2023/07/sliced-onion-architecture


Module
Module
Module

Demo

Spring Modulith

1. **A convention to map contexts to packages**

   Packages directly nested underneath the main application package are considered context ones.

2. **Simple set of access rules and API to verify**

   Only access components in module API packages.

3. **Test support to bootstrap modules**

   Integration with Spring test context framework and Spring Boot integration test support to limit bootstrap (component & entity scanning) to only the involved packages.

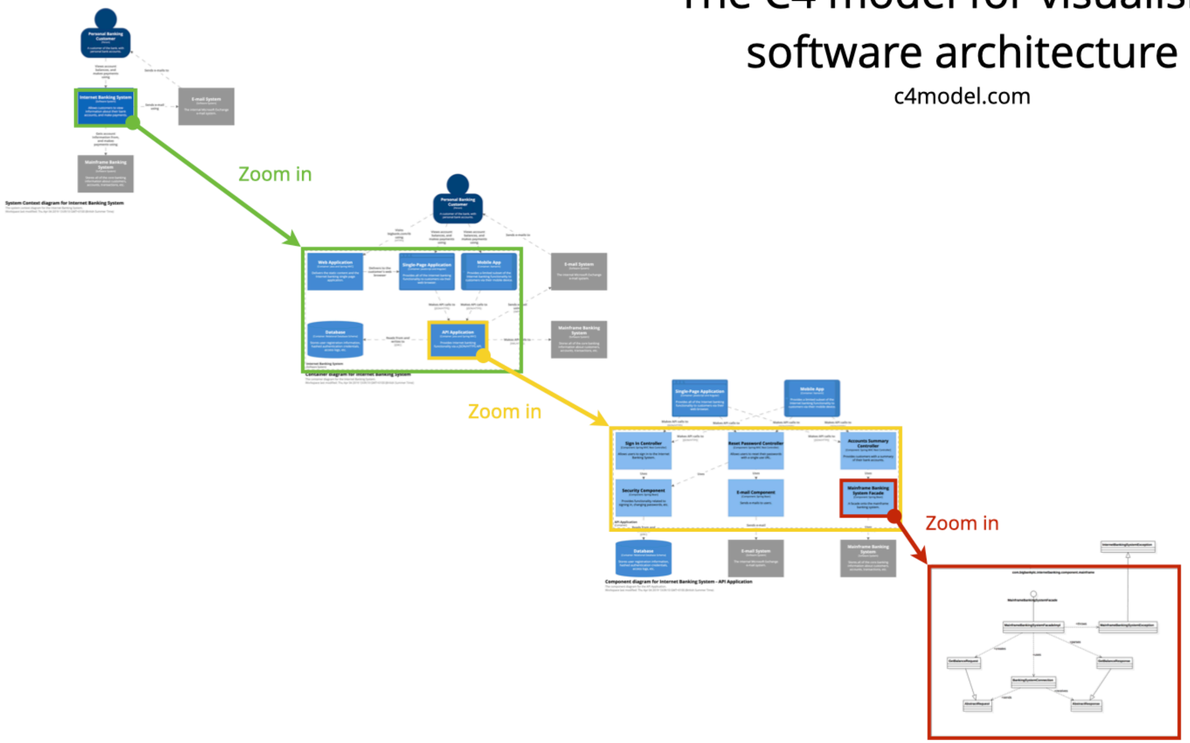
4. **Documentation support**

   PlantUML integration via Simon Brown's Structurizr to document module structure and dependency types.

Documentation

The C4 model for visualising software architecture

c4model.com



Level 1
Context

Level 2
Containers

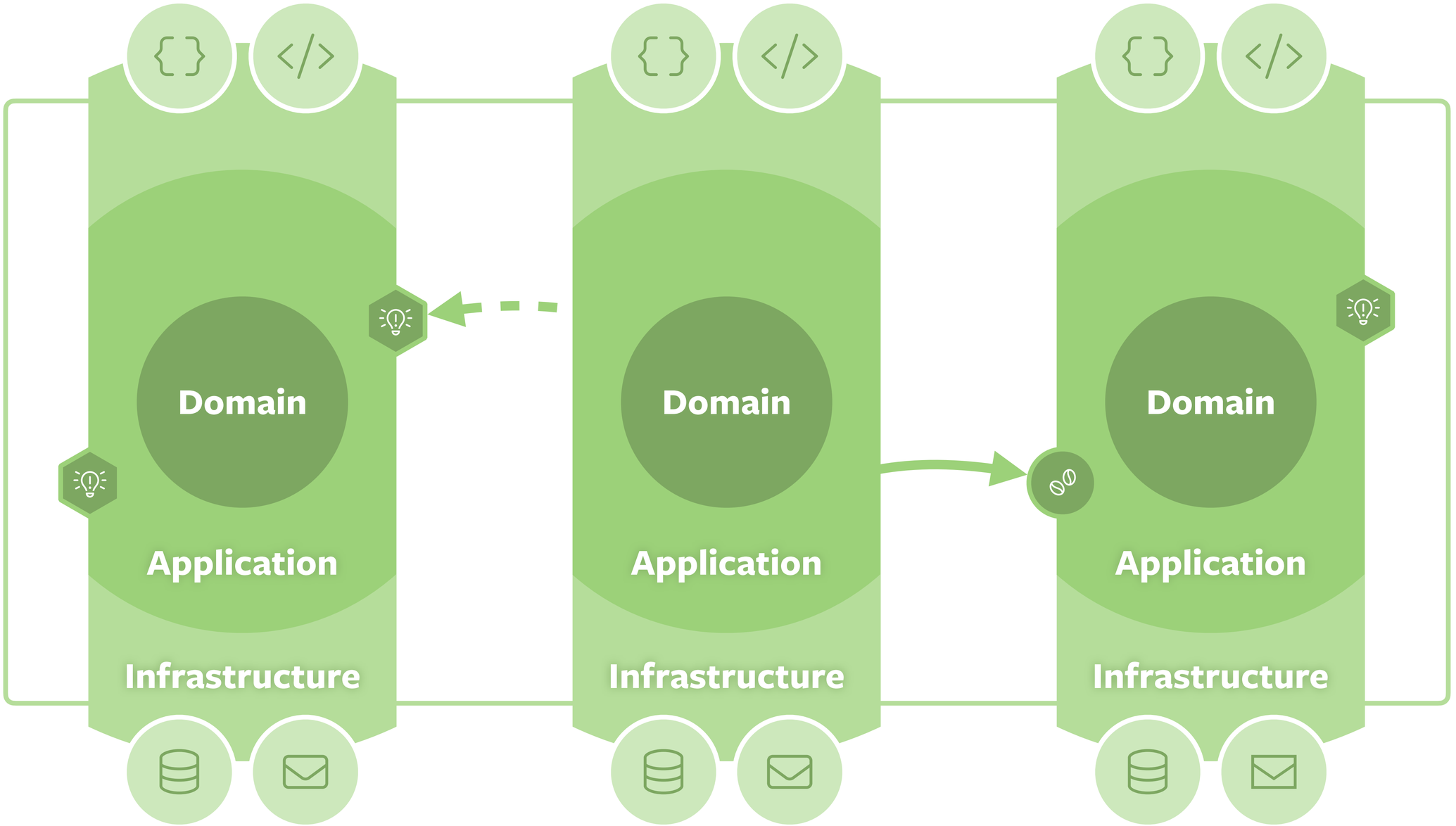
Level 3
Components

Level 4
Code

Pace of change

Level of detail

Effort of manual work


Domain
Application
Infrastructure
Domain
Application
Infrastructure
Domain
Application
Infrastructure

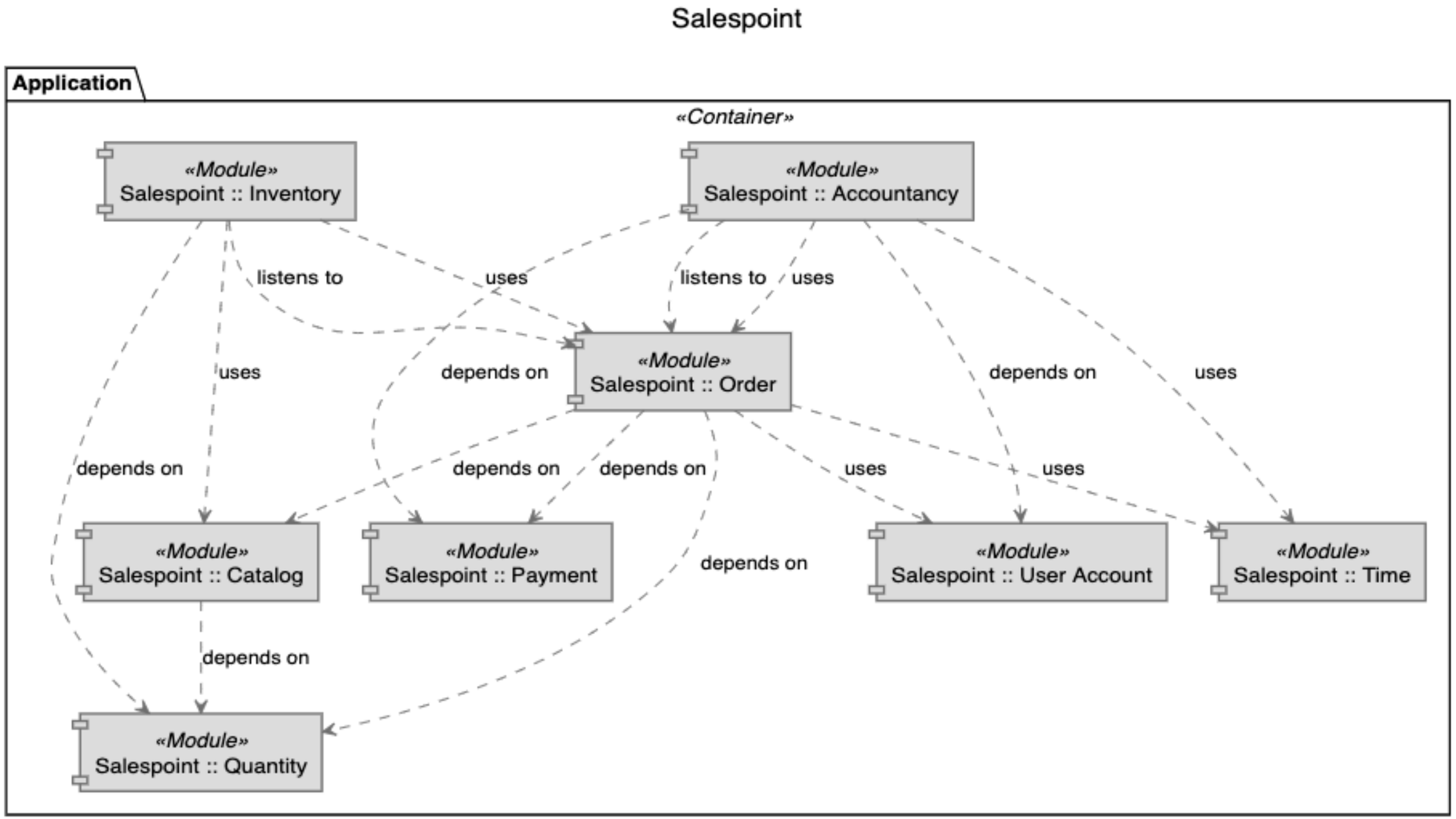


Figure 1. Salespoint component overview

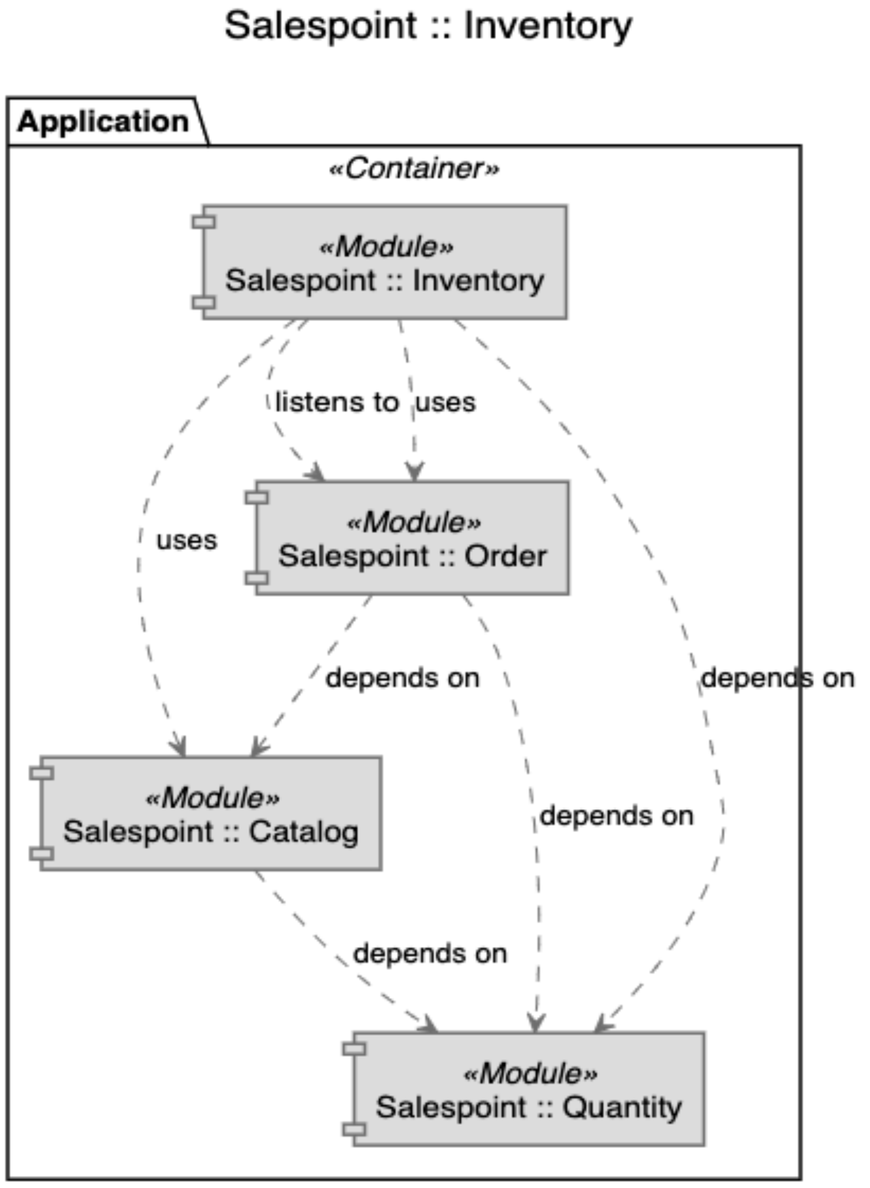
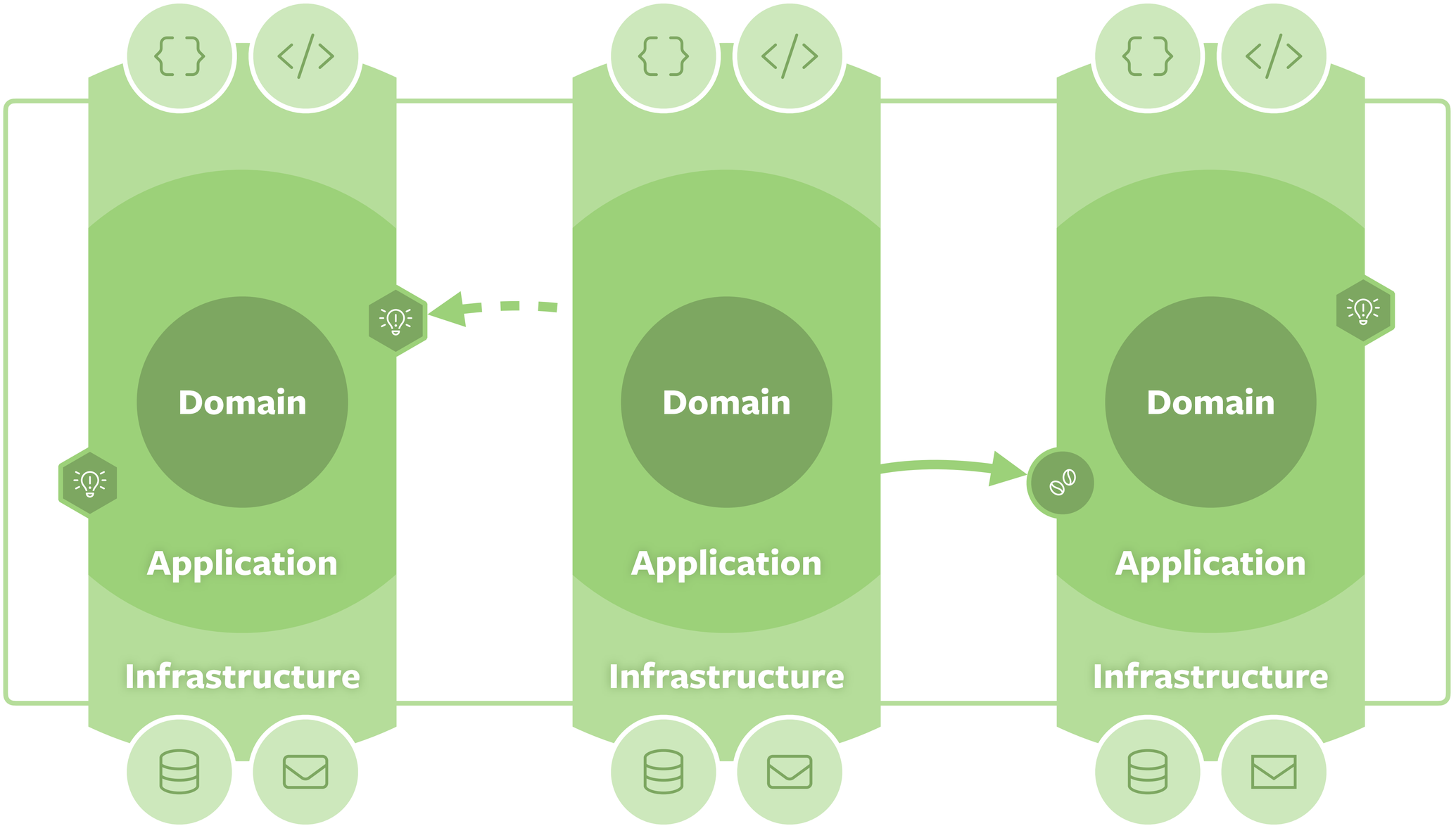
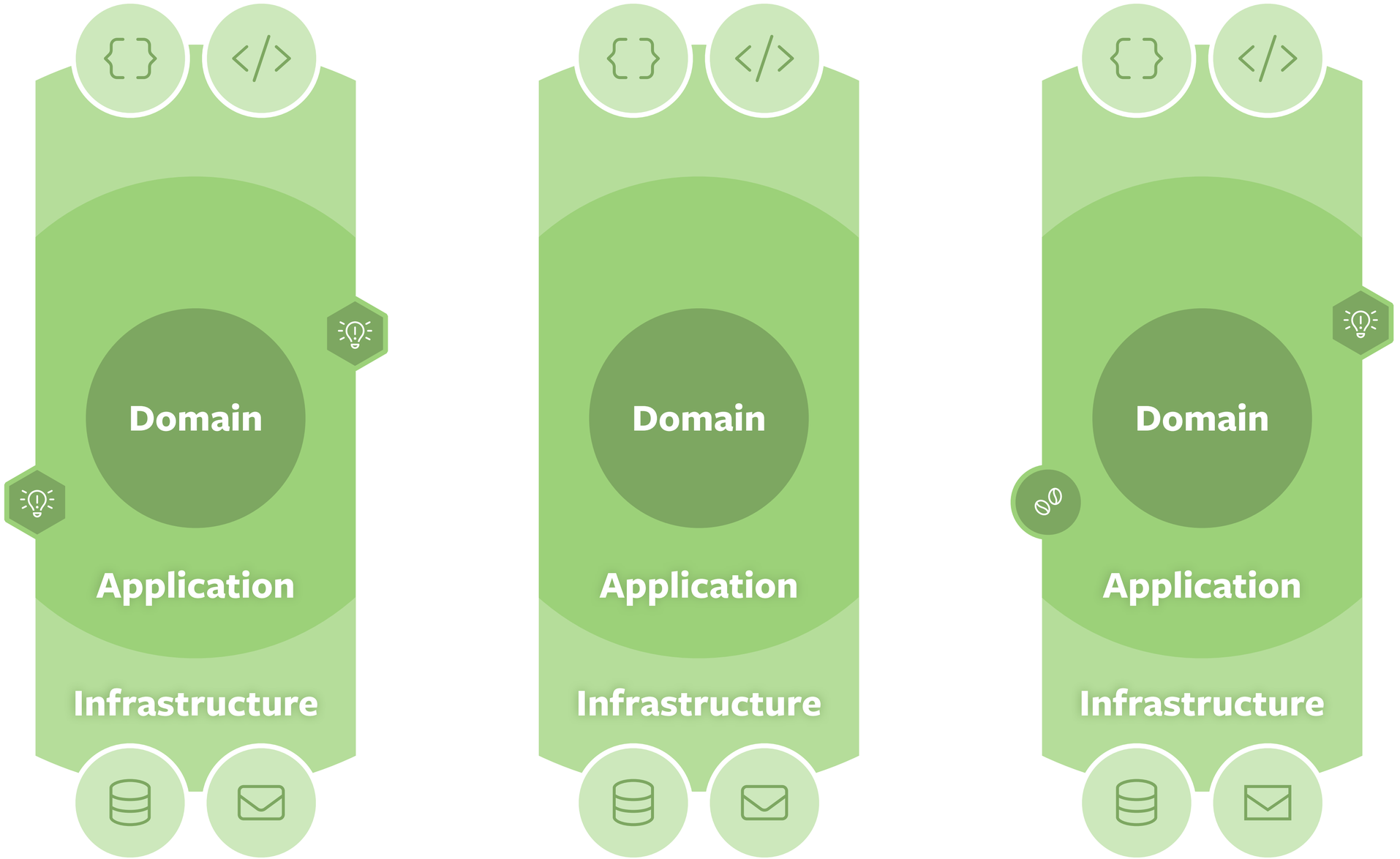
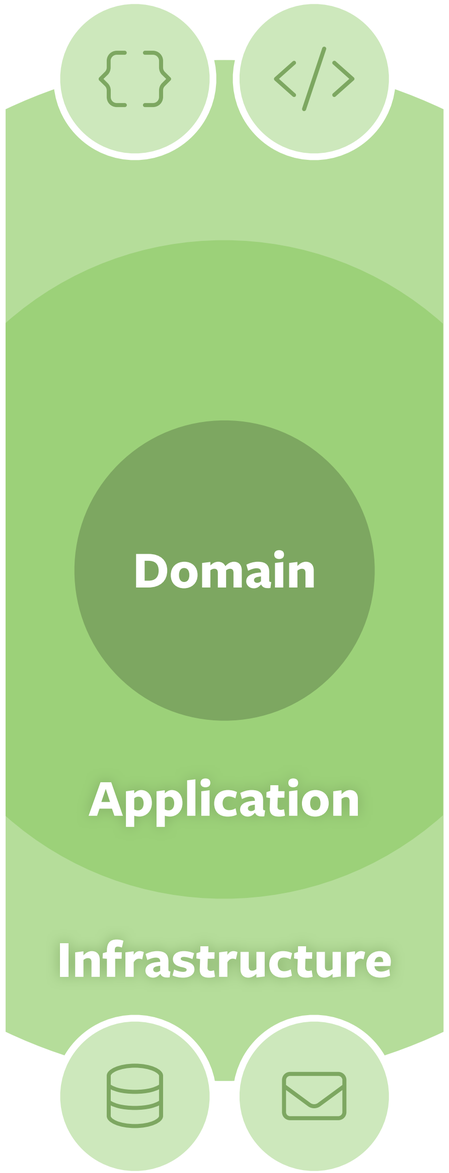


Figure 7. Inventory component


Domain
Application
Infrastructure
Domain
Application
Infrastructure
Domain
Application
Infrastructure


Domain
Application
Infrastructure
Domain
Application
Infrastructure
Domain
Application
Infrastructure


Domain
Application
Infrastructure

Provided Interface

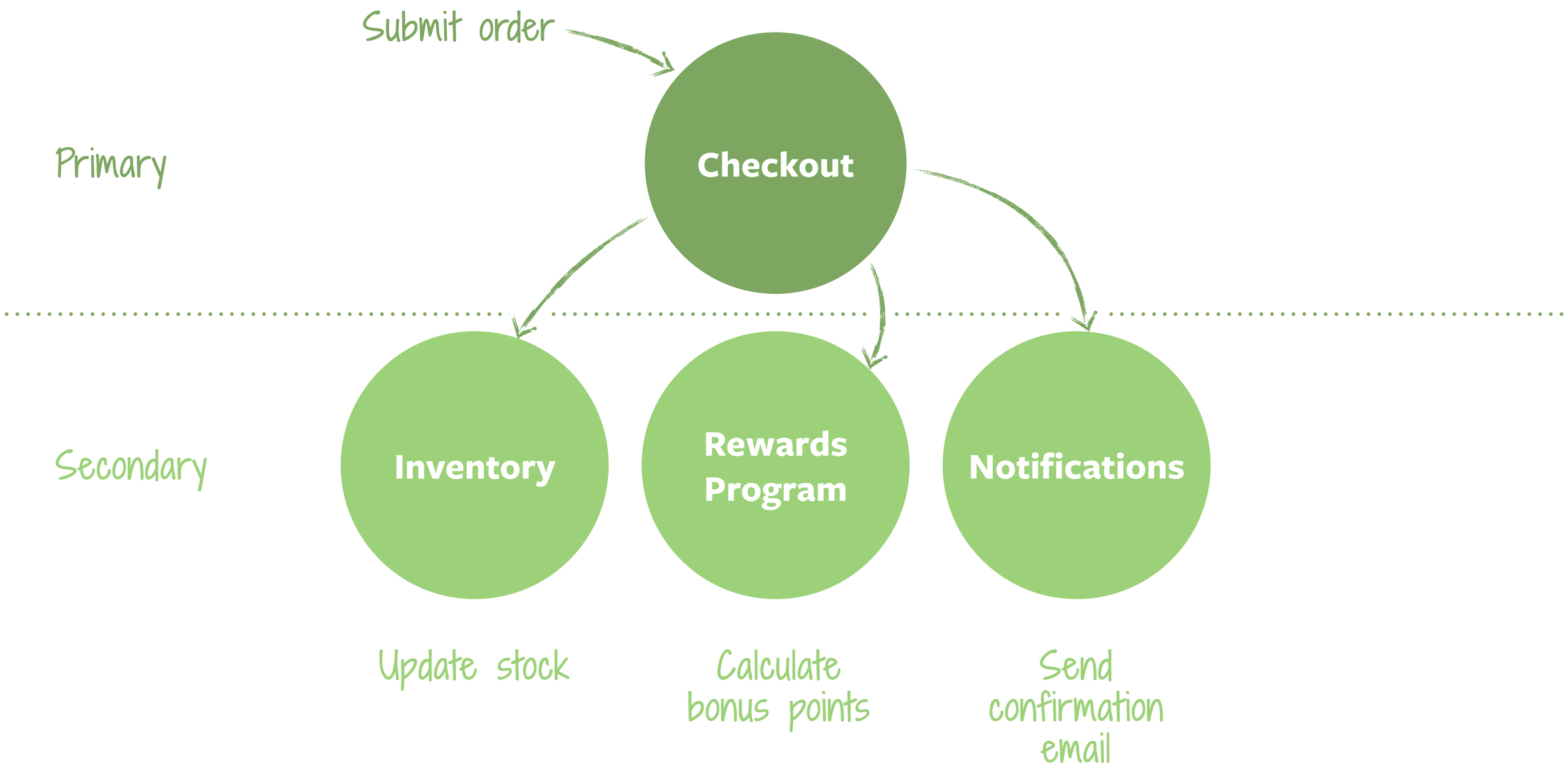
- **Exposed Service API**
  Spring Beans available for DI
- **Exposed Aggregates**
  Primary elements of the domain and constraints
- **Published events**
  Events the component emits

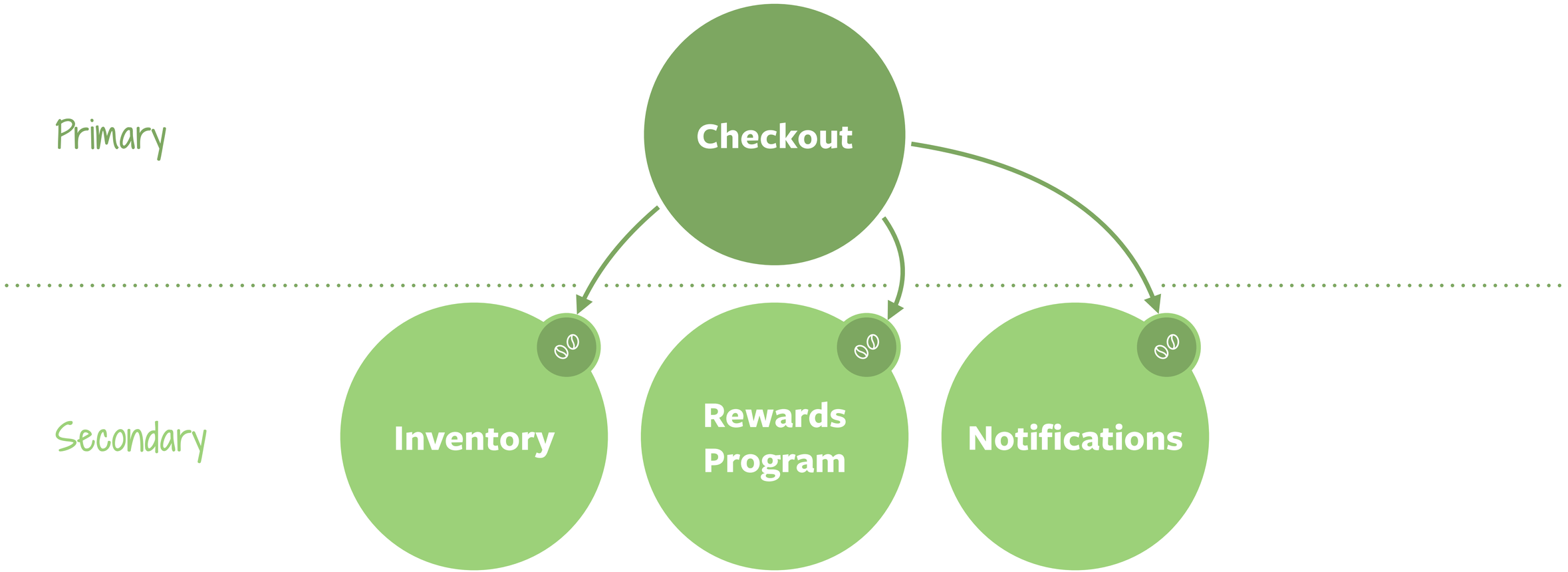
Required Interface

- **Consumed Service API**
  External dependencies of Spring beans
- **Configuration**
  Spring Boot configuration properties
- **Consumed events**
  Events that the component reacts to

Demo

It Takes Two To Tango...


Submit order
Primary
Checkout
Secondary
Inventory
Rewards Program
Notifications
Update stock
Calculate bonus points
Send confirmation email



Integration via DI

```java
@Service
@RequiredArgsConstructor
class Checkout {

  private final OrderRepository orders;

  @Transactional
  void complete(Order order) {

    orders.save(order.complete());

  }
}
```

Primary

Internal, technical decomposition

State transition

```java
@Service
@RequiredArgsConstructor
class Checkout {

  private final OrderRepository orders;

  @Transactional
  void complete(Order order) {

    orders.save(order.complete());

  }
}
```

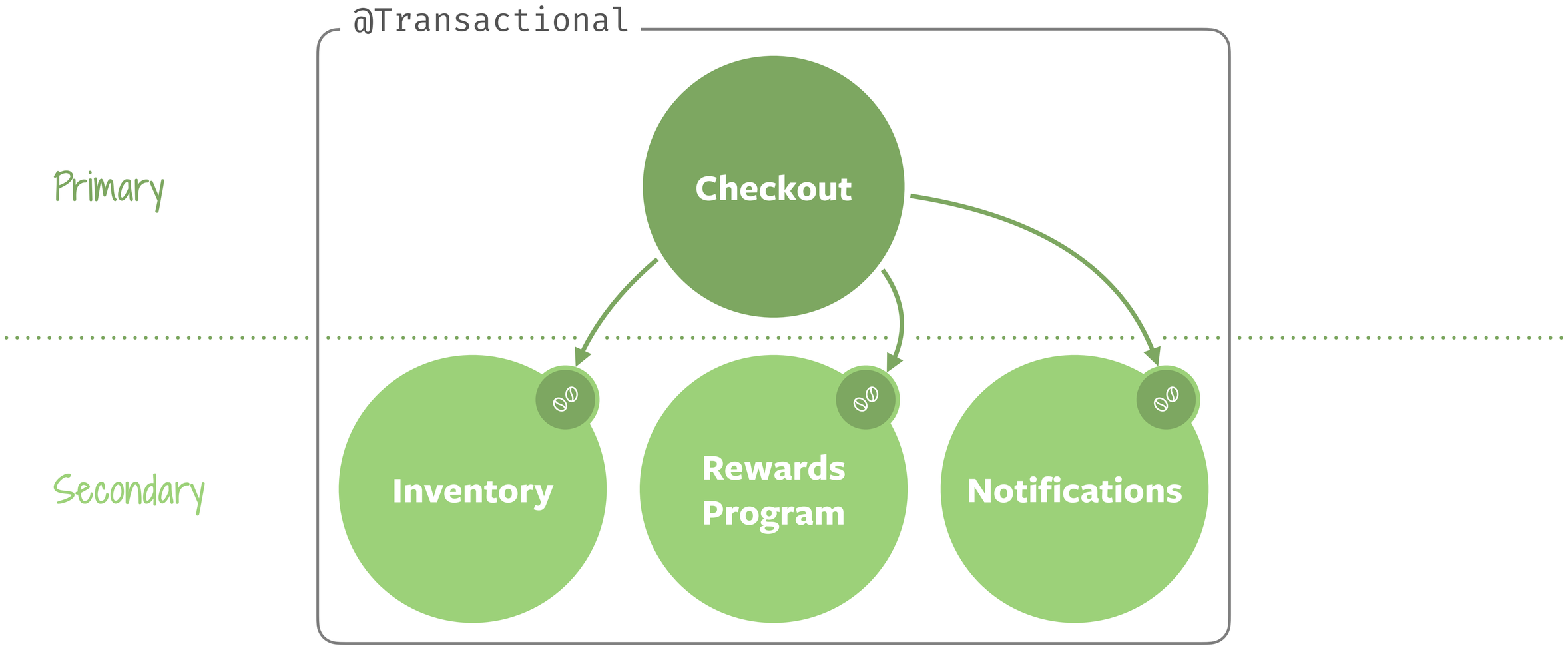
Primary

Secondary

Required at startup / in tests

```java
@Service
class Inventory { }

@Service
class RewardsProgram { }

@Service
class Notifications { }
```

Interaction expands transaction



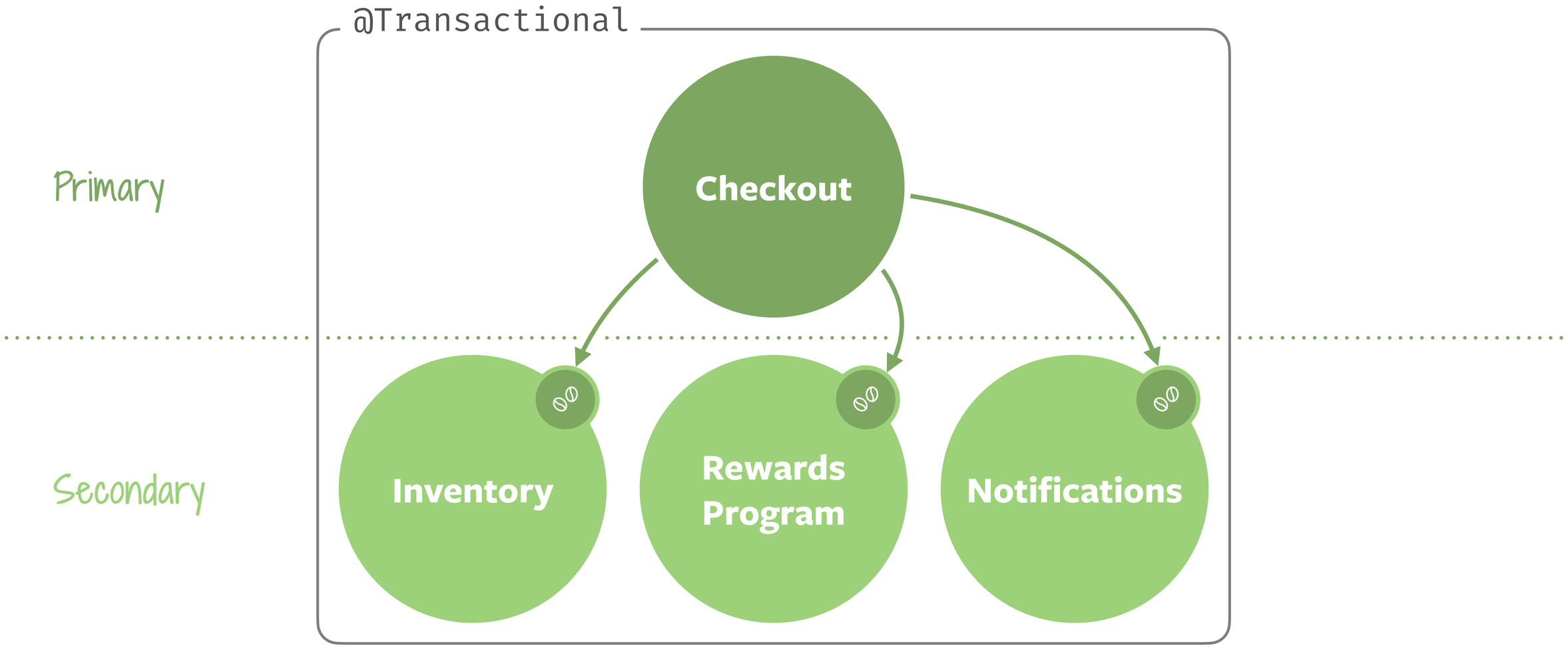
Scope of Consistency

```java
@SpringBootTest
class CheckoutTests {

  @Autowired Checkout checkout;

  @MockBean Inventory inventory;
  // Other mocks

  @Test
  void completesOrder() {

    var order = new Order(…);

    checkout.complete(order);

    verify(inventory).updateStock(…);
  }
}
```

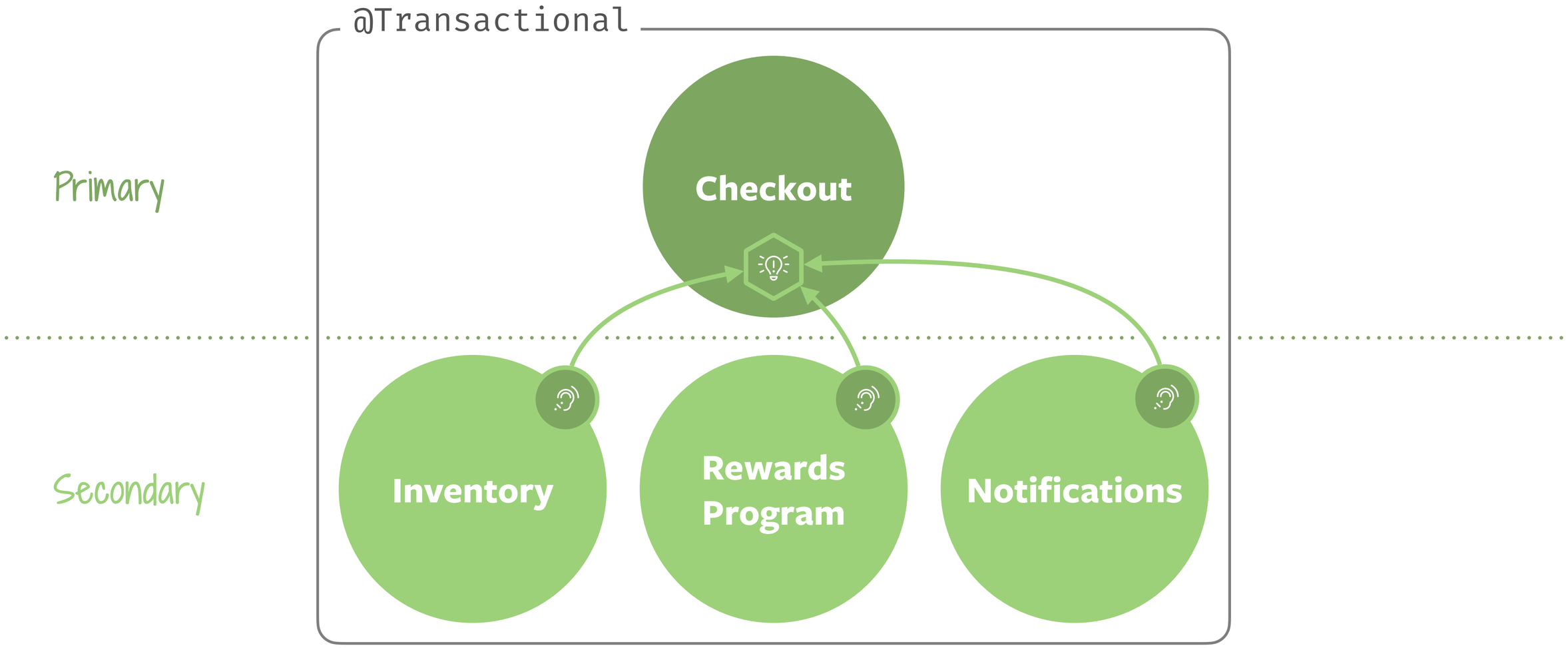
Collaborators are mocked

Given / When / Then

Testing focussed on orchestration



Scope of Consistency



Integration via Events

Only internal dependencies

```java
@Service
@RequiredArgsConstructor
class Checkout {

  private final OrderRepository orders;
  private final ApplicationEventPublisher events;

  @Transactional
  void complete(Order order) {

    orders.save(order.complete());

    events.publishEvent(
        OrderCompleted.of(order.getId()));
  }
}
```

Primary

Secondary

Invokes

```java
@Service
class Inventory {

  @EventListener
  void on(OrderCompleted event) {}
}

…
```

```java
@RecordApplicationEvents
/* or */
@ApplicationModuleTest
class CheckoutTests {

  private final Checkout checkout;

  @Test
  void completesOrder(AssertableApplicationEvents events) {

    var order = new Order(…);

    checkout.complete(order);

    assertThat(events)
      .contains(OrderCompleted.class)
      .matching(OrderCompleted::id, order.getId());
  }
}
```

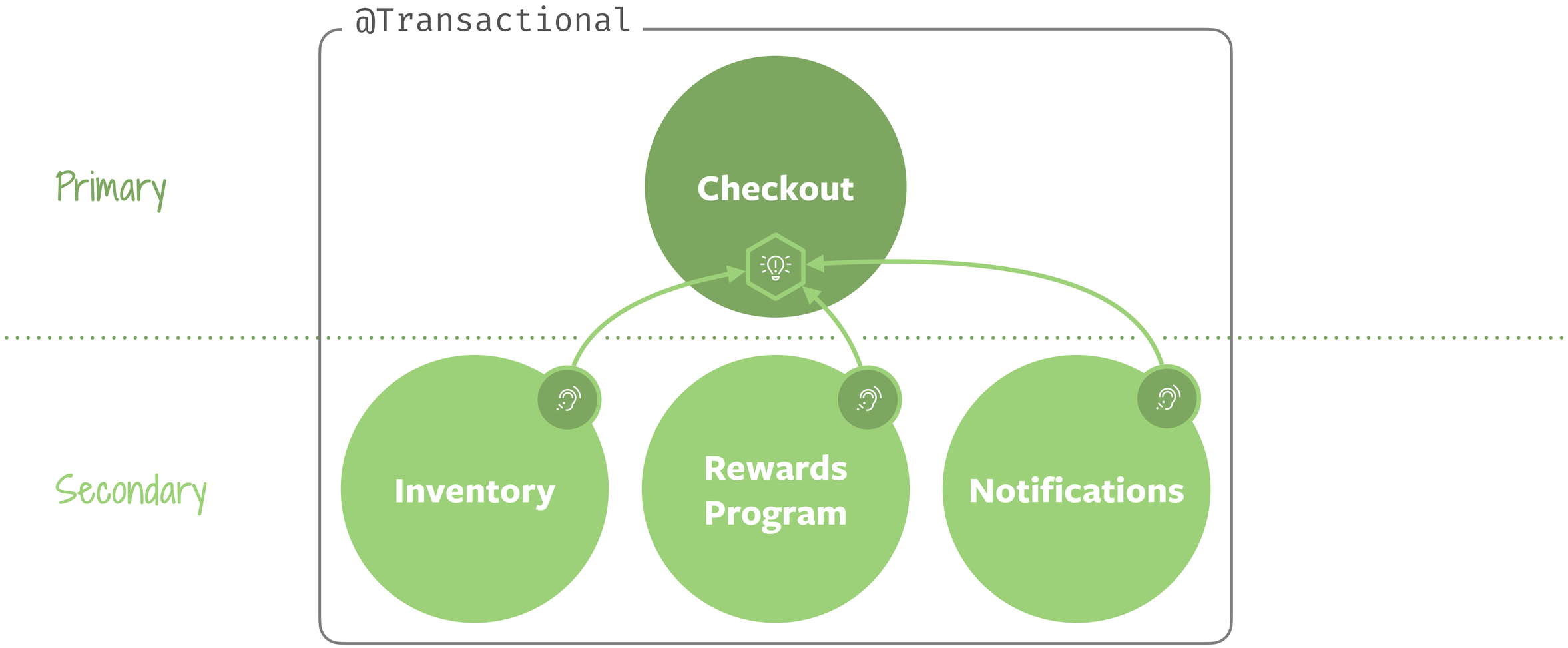
Spring Framework

Spring Modulith

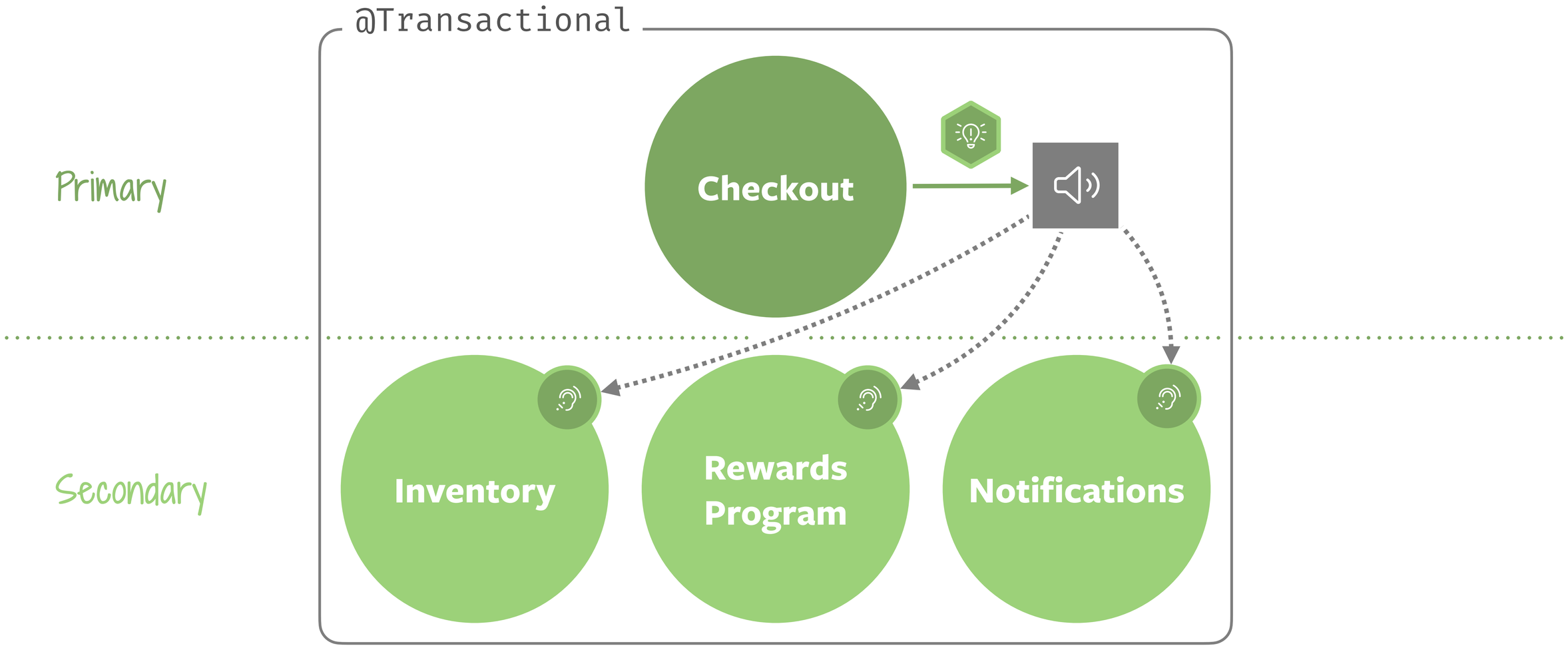
Records events published during test execution

Given / When / Then

Testing focussed on signal



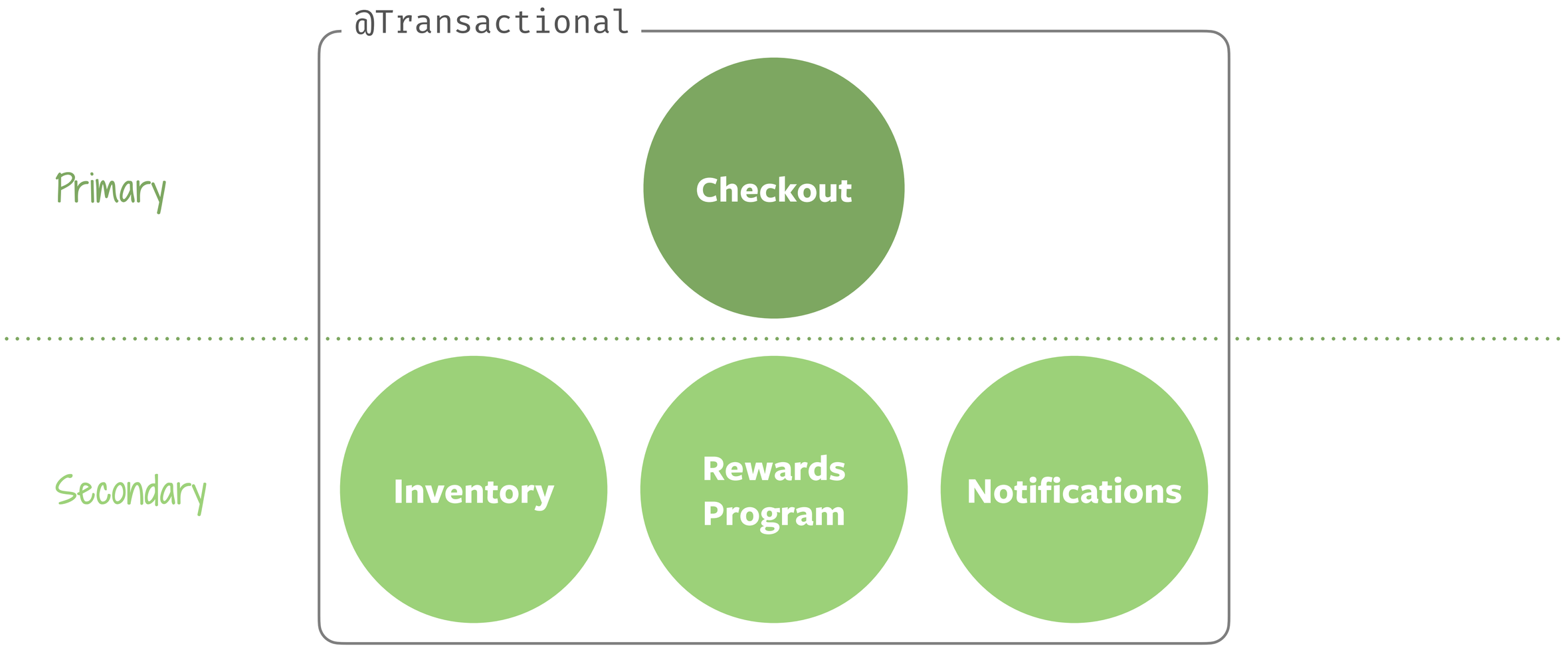
Integration via Events

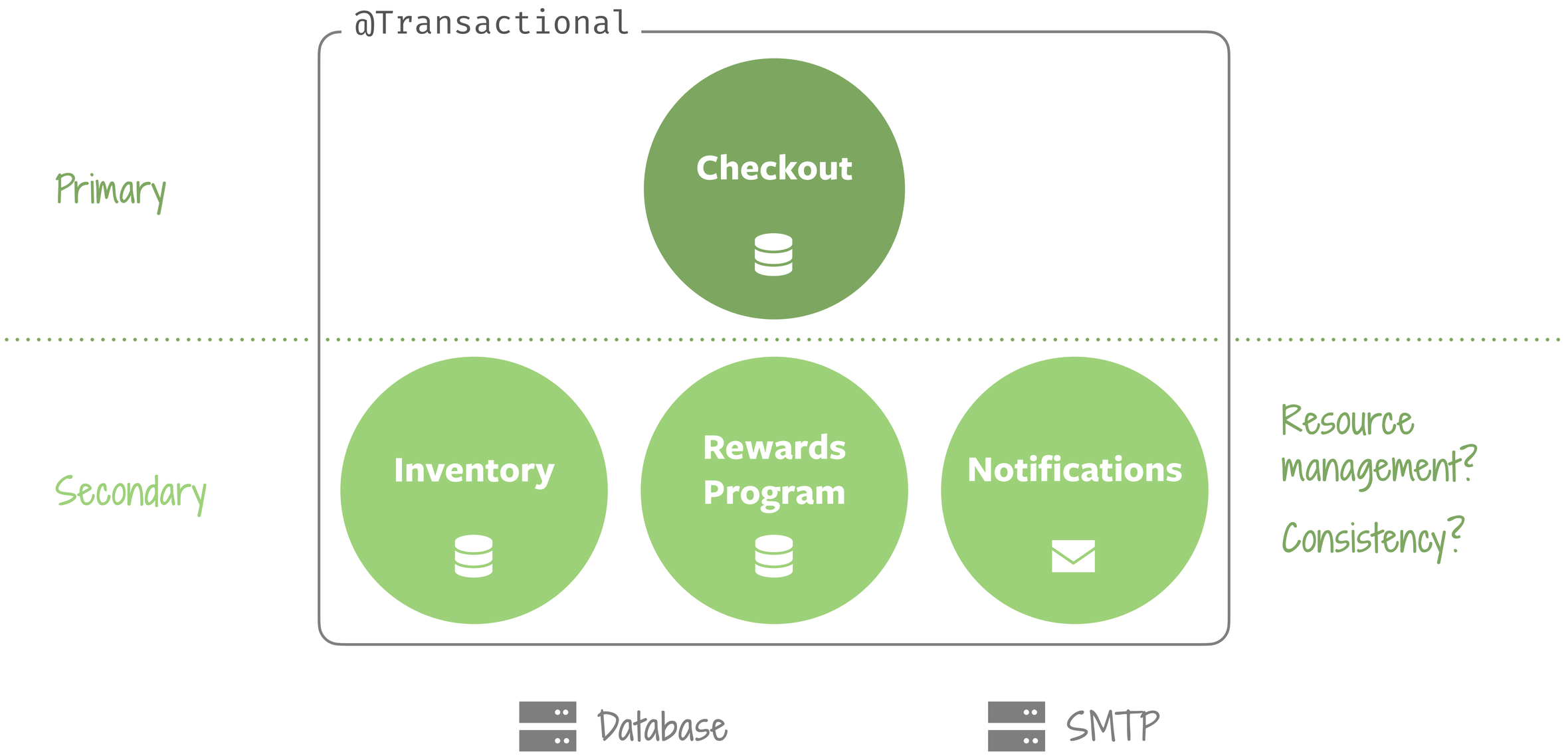


Invocation Flow for Events

Summary

- Change in structural arrangement
- Change in approach to testing
- ***No*** change to consistency arrangement


@Transactional
Primary
Checkout
Secondary
Inventory
Rewards Program
Notifications


@Transactional
Primary
Checkout
Secondary
Inventory
Rewards Program
Notifications
Resource management?
Consistency?
Database
SMTP

```java
@Service
@RequiredArgsConstructor
class Checkout {

  private final OrderRepository orders;
  private final ApplicationEventPublisher events;

  @Transactional
  void complete(Order order) {

    orders.save(order.complete());

    events.publishEvent(
        OrderCompleted.of(order.getId()));
  }
}
```

Primary

Secondary

```java
@Service
class Notifications {

  @EventListener
  void on(OrderCompleted event) {
    // Interact with SMTP server
  }
}
```

What if this takes long?

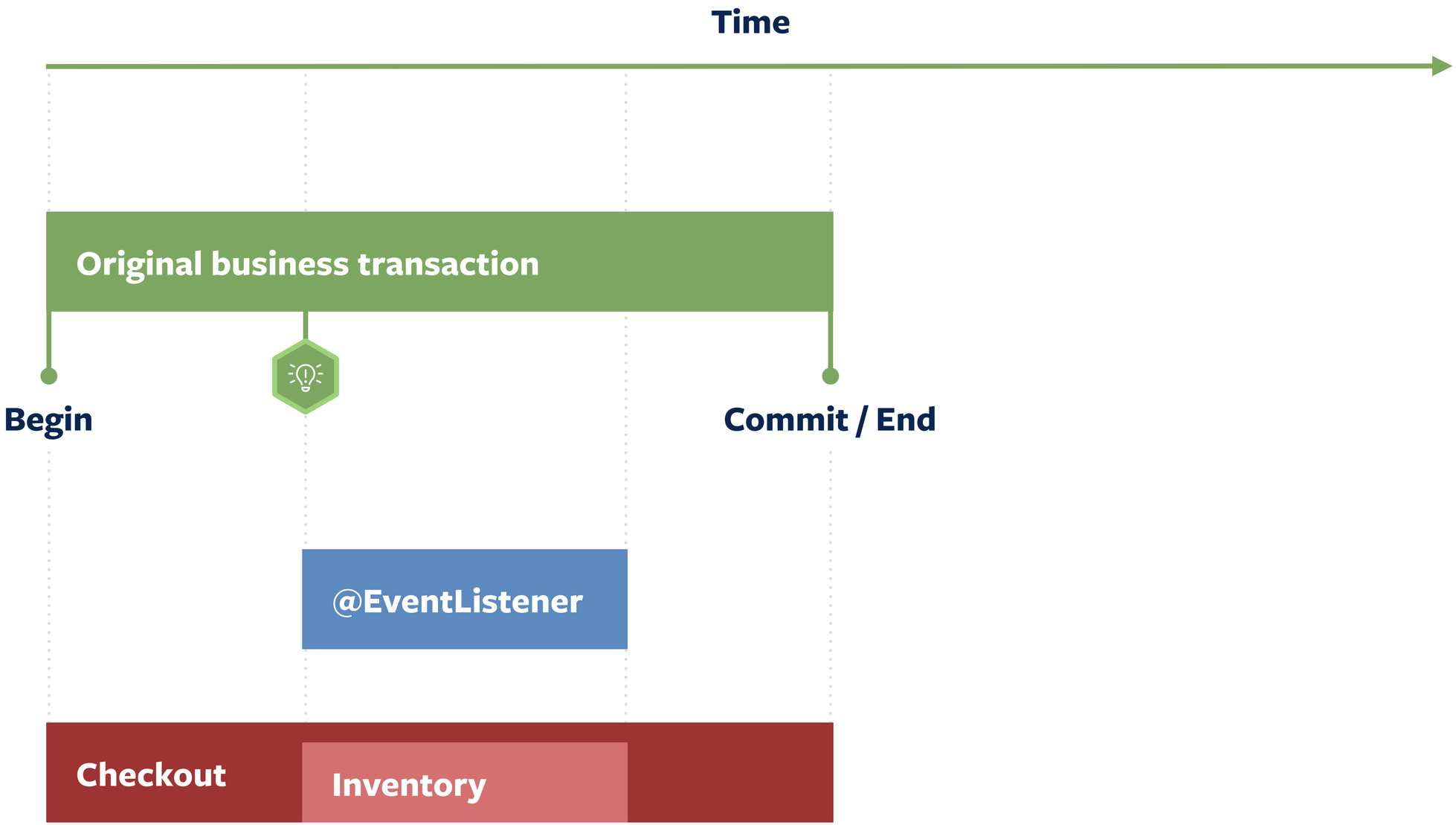
```java
@Service
@RequiredArgsConstructor
class Checkout {

  private final OrderRepository orders;
  private final ApplicationEventPublisher events;

  @Transactional
  void complete(Order order) {

    orders.save(order.complete());

    events.publishEvent(
        OrderCompleted.of(order.getId()));
  }
}
```

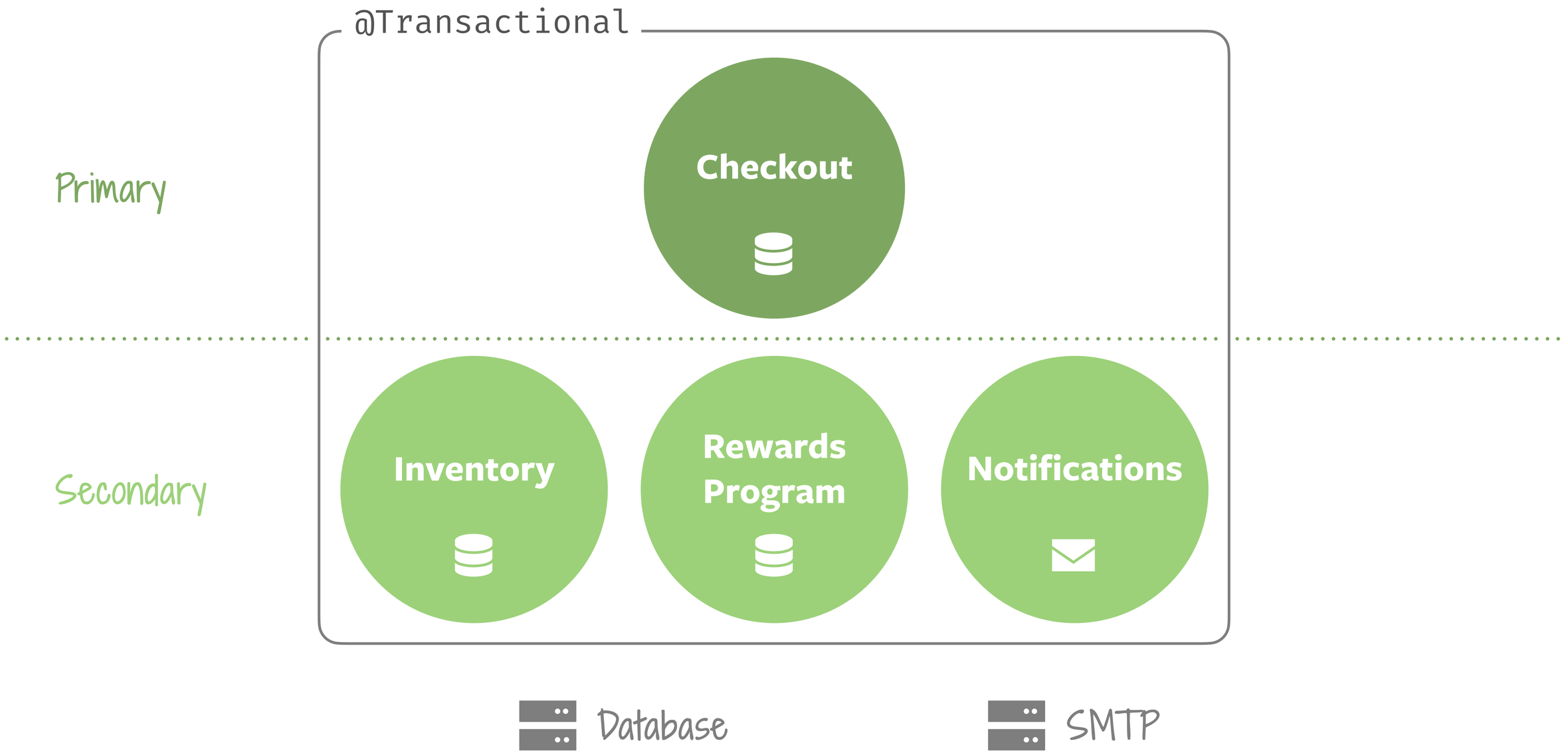
Primary

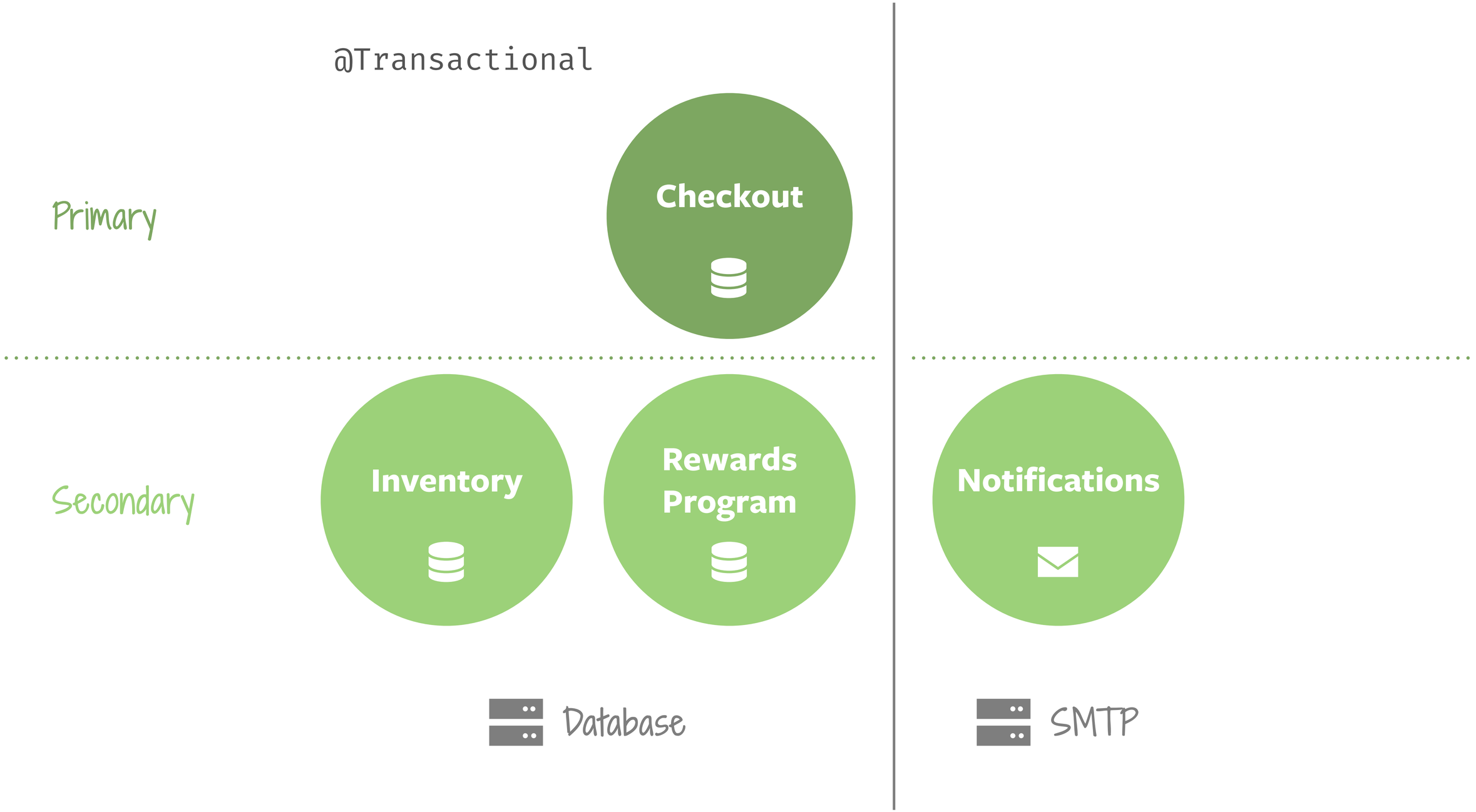
Secondary

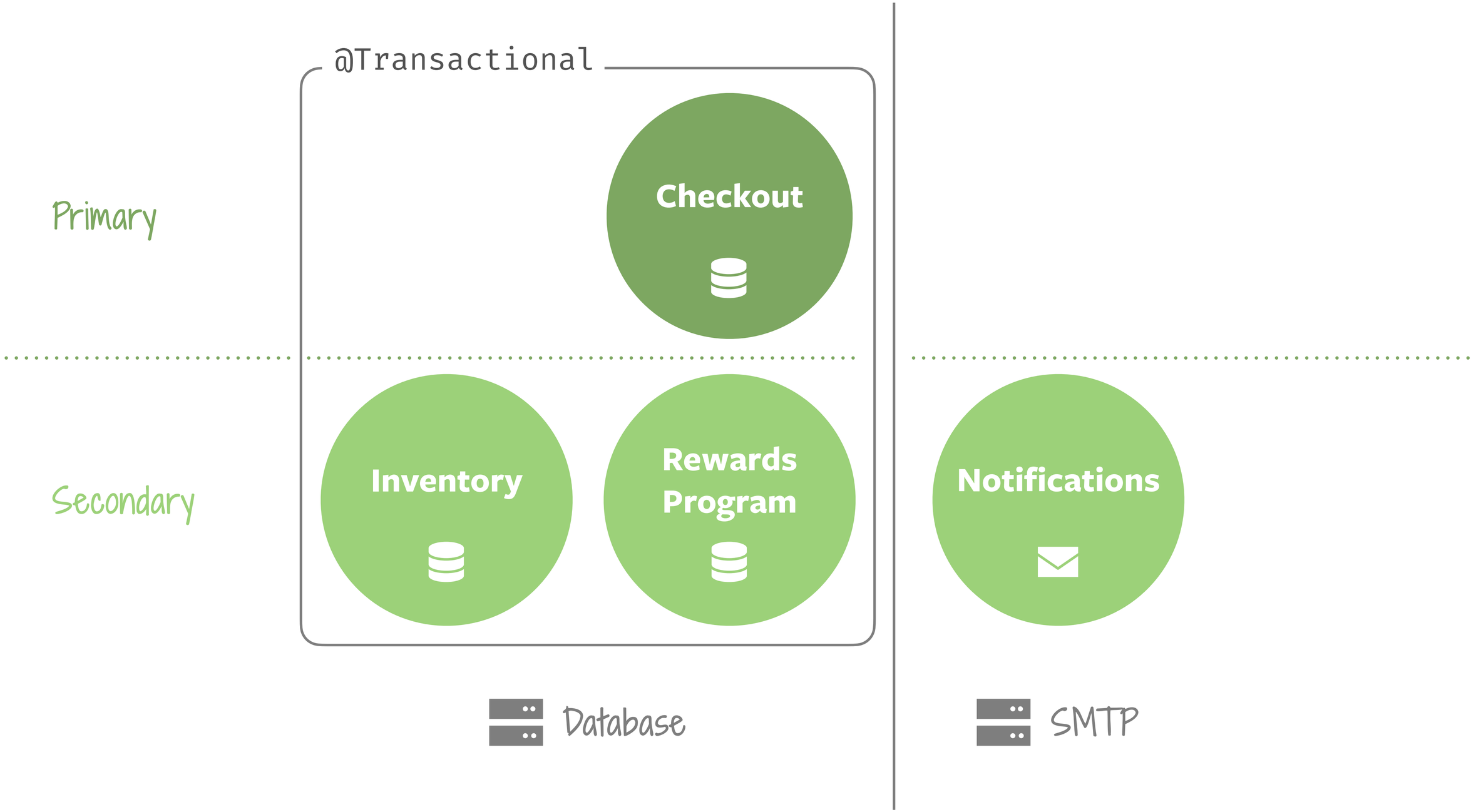
```java
@Service
class Notifications {

  @EventListener
  void on(OrderCompleted event) {
    // Interact with SMTP server
  }
}
```

What if this succeeds but this rolls back?


Time
Original business transaction
Begin
Commit / End
@EventListener
Checkout
Inventory


@Transactional
Primary
Checkout
Secondary
Inventory
Rewards Program
Notifications
Database
SMTP


@Transactional
Primary
Checkout
Secondary
Inventory
Rewards Program
Notifications
Database
SMTP


@Transactional
Primary
Checkout
Secondary
Inventory
Rewards Program
Notifications
Database
SMTP

Primary

Secondary

```java
@Service
@RequiredArgsConstructor
class Checkout {

  private final OrderRepository orders;
  private final ApplicationEventPublisher events;

  @Transactional
  void complete(Order order) {

    orders.save(order.complete());

    events.publishEvent(
        OrderCompleted.of(order.getId()));
  }
}
```

```java
@Service
class Notifications {

  @EventListener
  void on(OrderCompleted event) {
    // Interact with SMTP server
  }
}
```

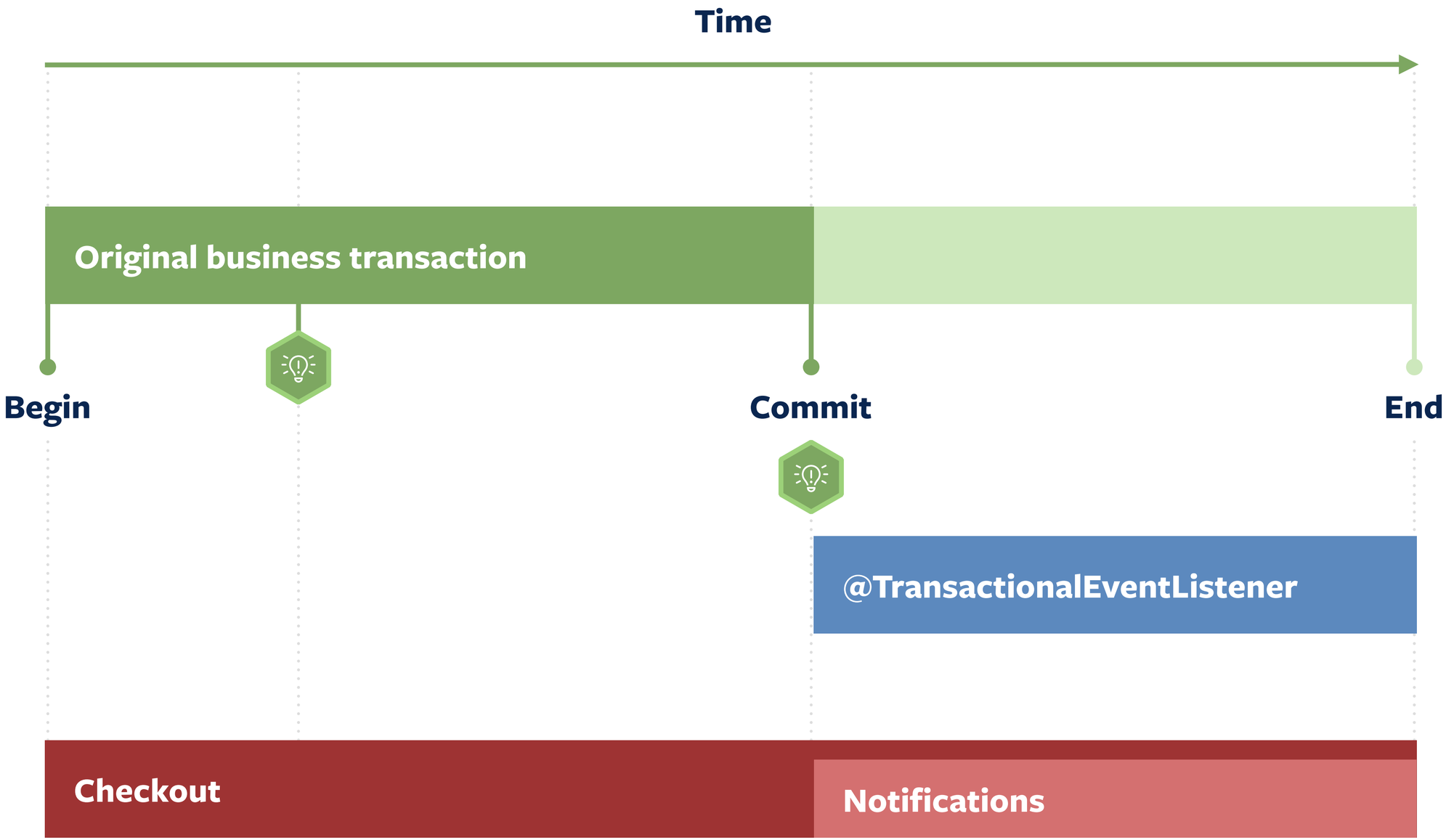
Triggered on transaction commit

Primary

Secondary

```java
@Service
@RequiredArgsConstructor
class Checkout {

  private final OrderRepository orders;
  private final ApplicationEventPublisher events;

  @Transactional
  void complete(Order order) {

    orders.save(order.complete());

    events.publishEvent(
        OrderCompleted.of(order.getId()));
  }
}
```

```java
@Service
class Notifications {

  @TransactionalEventListener
  void on(OrderCompleted event) {
    // Interact with SMTP server
  }
}
```

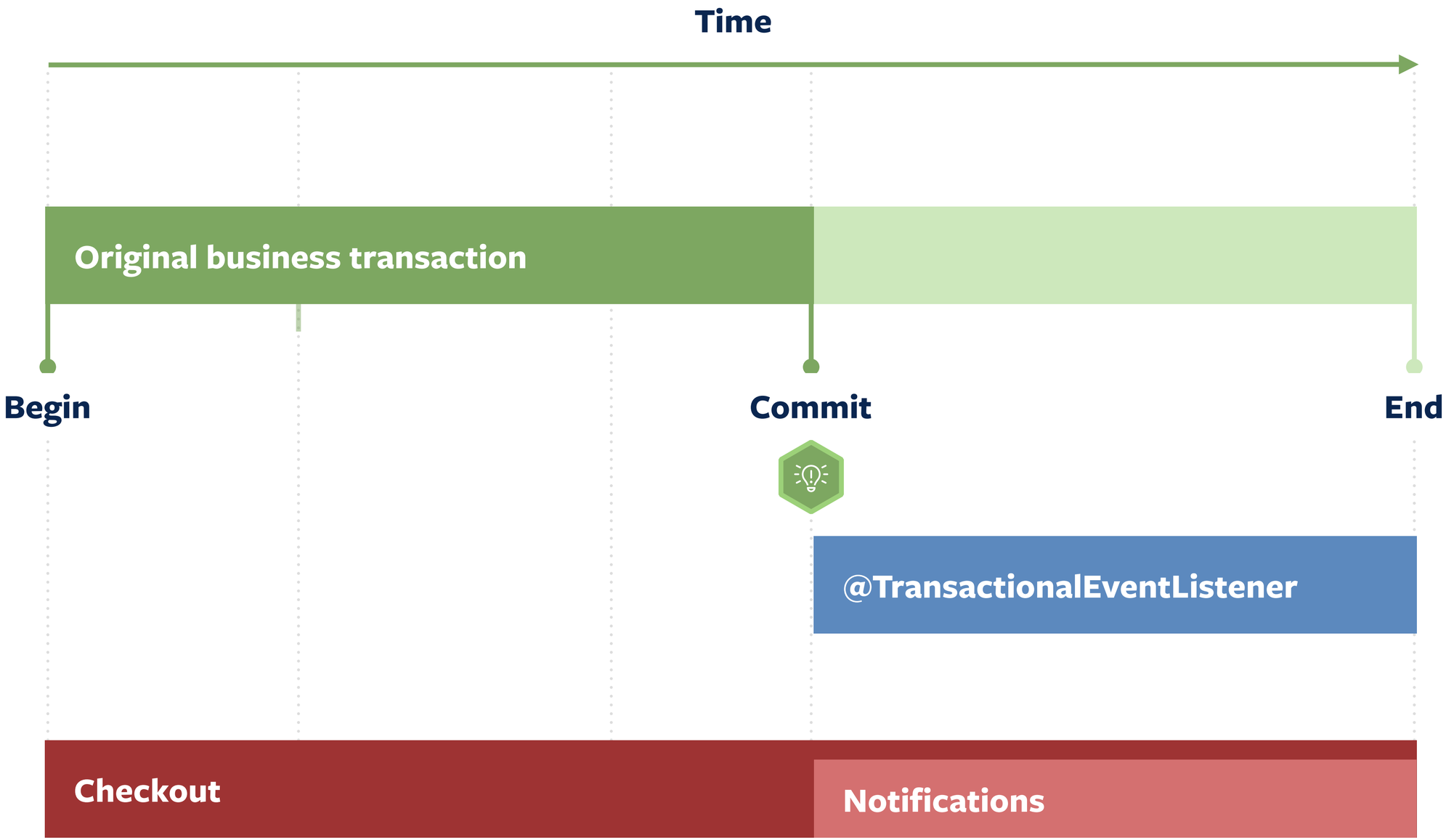

Time
Original business transaction
Begin
Commit
End
@TransactionalEventListener
Checkout
Notifications

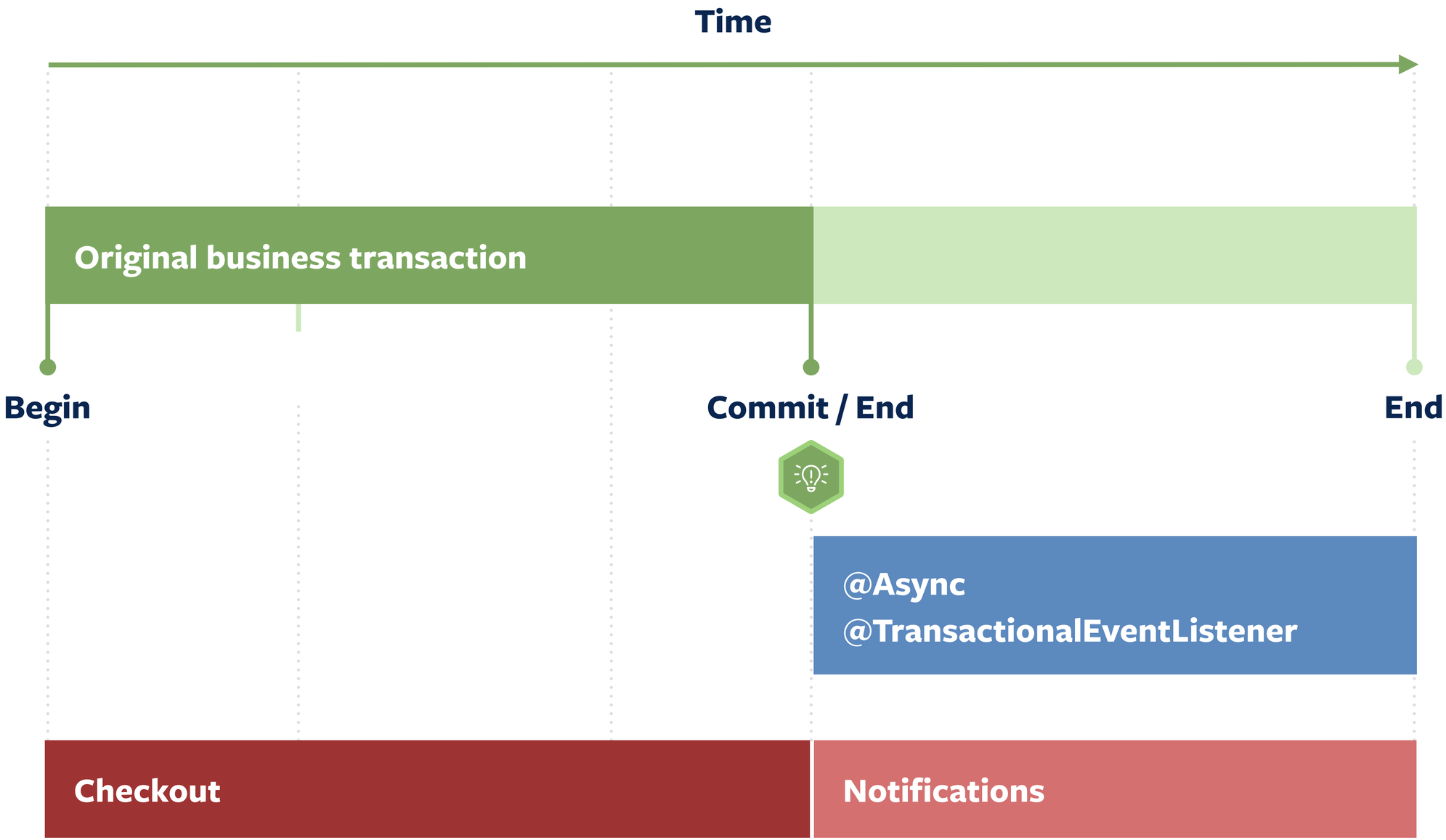
```java
@Service
@RequiredArgsConstructor
class Checkout {

  private final OrderRepository orders;
  private final ApplicationEventPublisher events;

  @Transactional
  void complete(Order order) {

    orders.save(order.complete());

    events.publishEvent(
        OrderCompleted.of(order.getId()));
  }
}
```

```java
@Service
class Notifications {

  @TransactionalEventListener
  void on(OrderCompleted event) {
    // Interact with SMTP server
  }
}
```

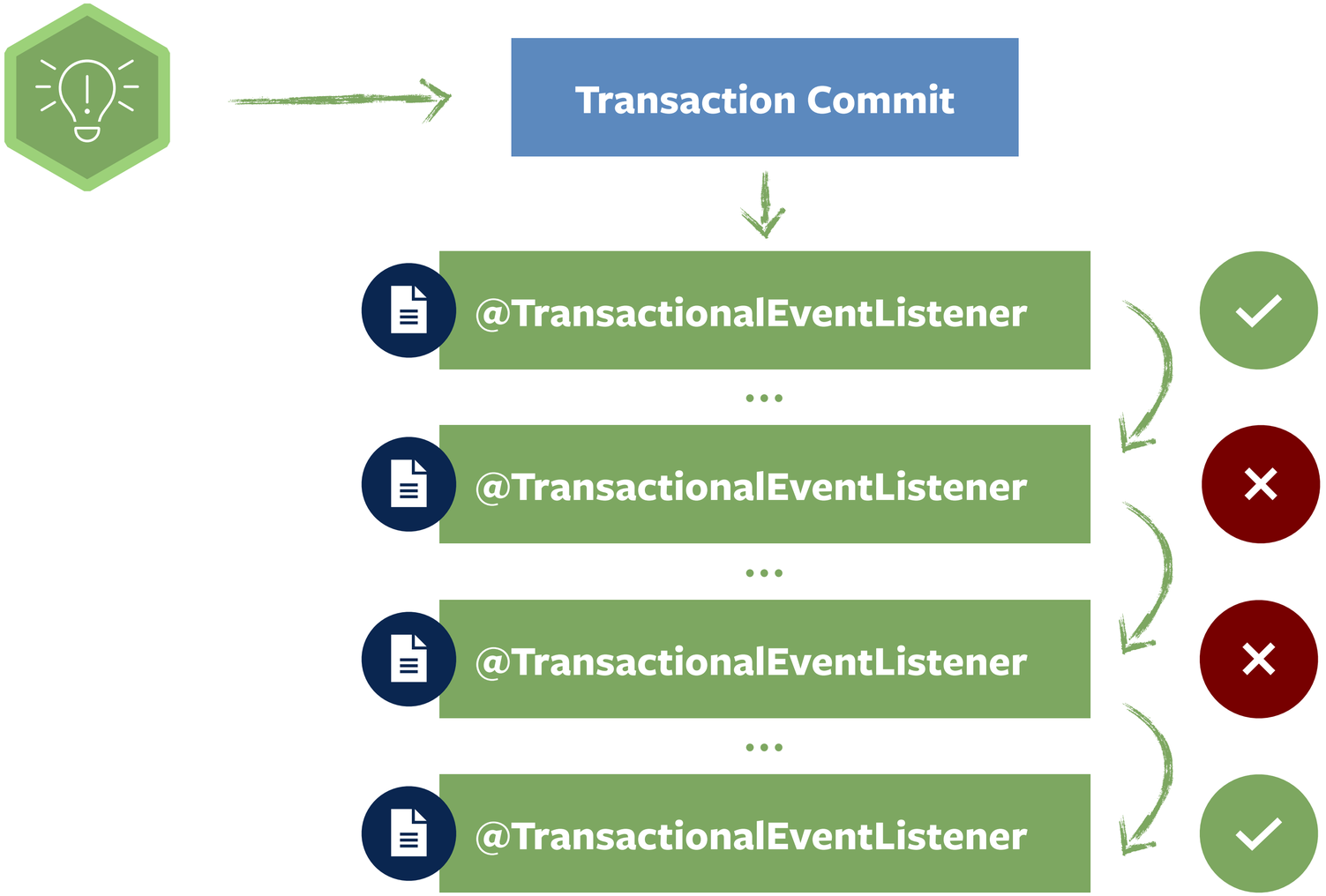
```java
@Service
@RequiredArgsConstructor
class Checkout {

  private final OrderRepository orders;
  private final ApplicationEventPublisher events;

  @Transactional
  void complete(Order order) {

    orders.save(order.complete());

    events.publishEvent(
        OrderCompleted.of(order.getId()));
  }
}
```

```java
@Service
class Notifications {

  @Async
  @TransactionalEventListener
  void on(OrderCompleted event) {
    // Interact with SMTP server
  }
}
```


Time
Original business transaction
Begin
Commit
End
@TransactionalEventListener
Checkout
Notifications


Time
Original business transaction
Begin
Commit / End
End
@Async
@TransactionalEventListener
Checkout
Notifications

What if a transactional event listener fails?


Transaction Commit
@TransactionalEventListener
...
@TransactionalEventListener
...
@TransactionalEventListener
...
@TransactionalEventListener

```java
@Service
@RequiredArgsConstructor
class Checkout {

  private final OrderRepository orders;
  private final ApplicationEventPublisher events;

  @Transactional
  void complete(Order order) {

    orders.save(order.complete());

    events.publishEvent(
        OrderCompleted.of(order.getId()));
  }
}
```

```java
@Service
class Inventory {

  @Async
  @TransactionalEventListener
  void on(OrderCompleted event) {
    // Update stock
  }
}
```

```java
@Service
@RequiredArgsConstructor
class Checkout {

  private final OrderRepository orders;
  private final ApplicationEventPublisher events;

  @Transactional
  void complete(Order order) {

    orders.save(order.complete());

    events.publishEvent(
        OrderCompleted.of(order.getId()));
  }
}
```

```java
@Service
class Inventory {

  @Async
  @Transactional
  @TransactionalEventListener
  void on(OrderCompleted event) {
    // Update stock
  }
}
```

```java
@Service
@RequiredArgsConstructor
class Checkout {

  private final OrderRepository orders;
  private final ApplicationEventPublisher events;

  @Transactional
  void complete(Order order) {

    orders.save(order.complete());

    events.publishEvent(
        OrderCompleted.of(order.getId()));
  }
}
```

```java
@Service
class Inventory {

  @ApplicationModuleListener
  void on(OrderCompleted event) {
    // Interact with SMTP server
  }
}
```

```java
@ApplicationModuleTest
class CheckoutTests {

  private final Checkout checkout;

  @Test
  void completesOrder(Scenario scenario) {

    var order = new Order(…);

    scenario.stimulate(() → checkout.complete(order))
      .andWaitForEventOfType(InventoryUpdated.class)
      .matchingMappedValue(InventoryUpdated::getOrderId, order.getId())
      .toArrive();
  }
}
```

Spring Modulith

Given / When / Then

	Classic	Event Listener	Application Module Listener
Module references	via Spring beans	via `@EventListener`	via `@ApplicationModuleListener` (`@Async` `@TransactionalEventListener`)
Integration style	synchronous	synchronous	Asynchronous / Transaction-bound
Tests			
Orientation	vertical / horizontal	vertical	vertical
Focus	Module interaction	Module operation outcome signaled by an event	Module operation outcome signaled by an event
Approach	via mocks / `@MockBean`	via `AssertablePublishedEvents`	via `Scenario`
Risk	Integration tests including multiple modules	Tests exclude code that could break transaction at runtime	Asynchronous module interaction
Consistency			
Boundaries	Potentially (accidentally) spanning multiple modules	Potentially (accidentally) spanning multiple modules	Aligned with module boundaries / Eventual Consistency /
Mechanism	Transaction	Transaction	Event Publication Registry
Module coupling			

Links

- **xMolecules**
  https://xmolecules.org
- **jMolecules**
  https://jmolecules.org
- **jMolecules Examples**
  https://github.com/xmolecules/jmolecules-examples

Resources

- **Software Architecture for Developers**
  Simon Brown – Books
- **Just Enough Software Architecture**
  George Fairbanks – Book
- **Architecture, Design, Implementation**
  Ammon H. Eden, Rick Kazman – Paper
- **Sustainable Software Architecture**
  Carola Lilienthal – Book

Shoutouts 🙇

- Peter Gafert – ArchUnit
- Rafael Winterhalter – ByteBuddy
- Bernd Dutkowski – IDEA plugin
- You?? 🙃 – ideas, discussions

Thank you!

Oliver Drotbohm / / odrotbohm info@odrotbohm.de