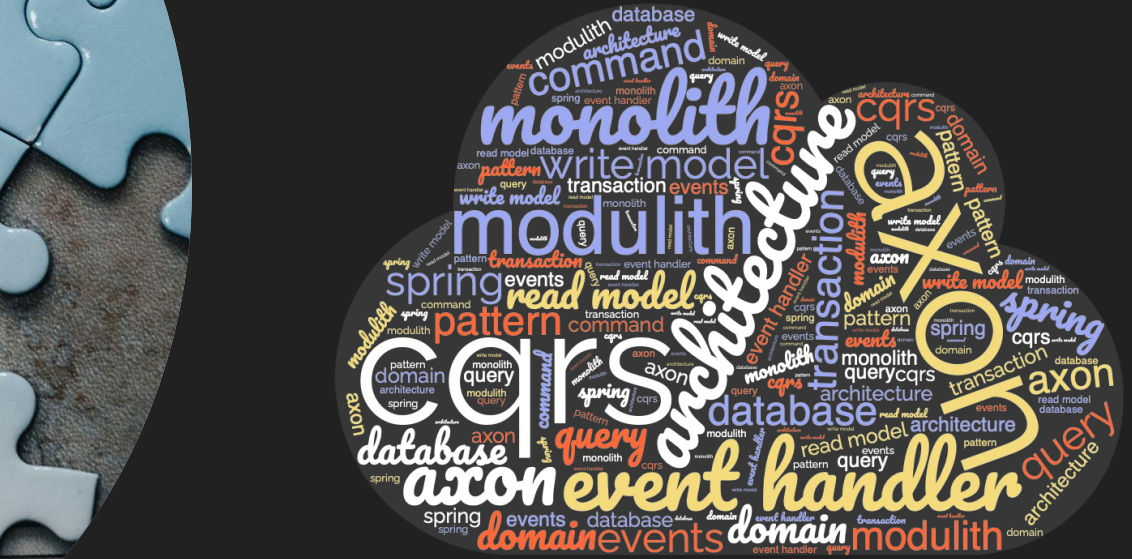
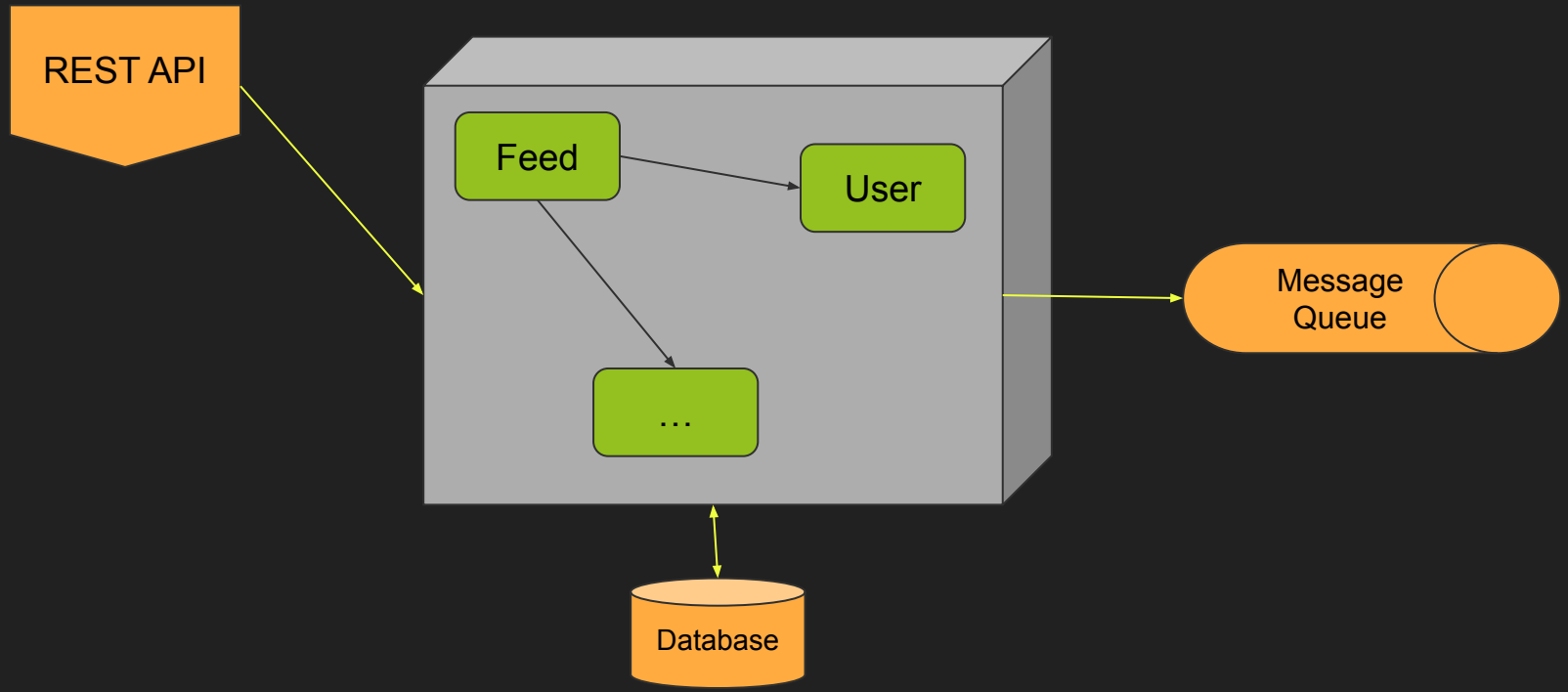
Ist Spring Modulith das nächste CQRS/ES-Framework?

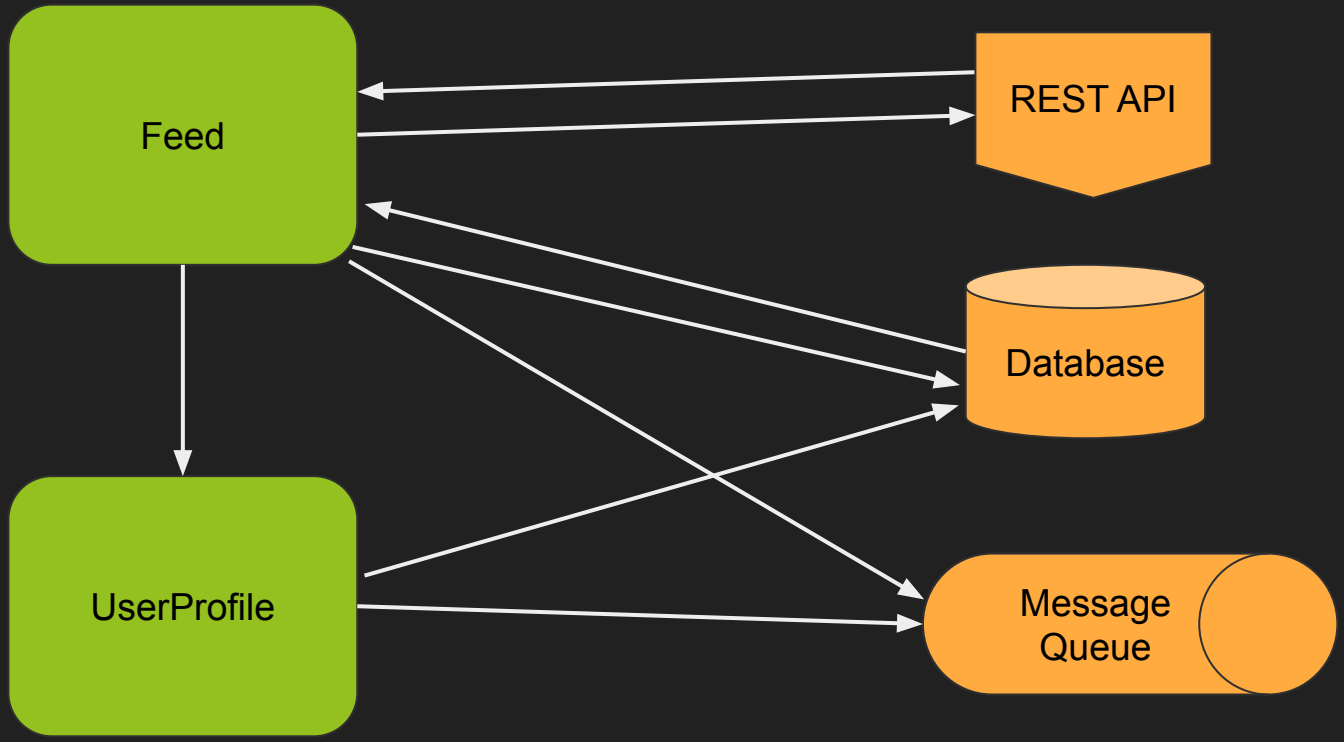


digital frontiers

Modulithic Architectures



Typical Challenges of Code Modularization



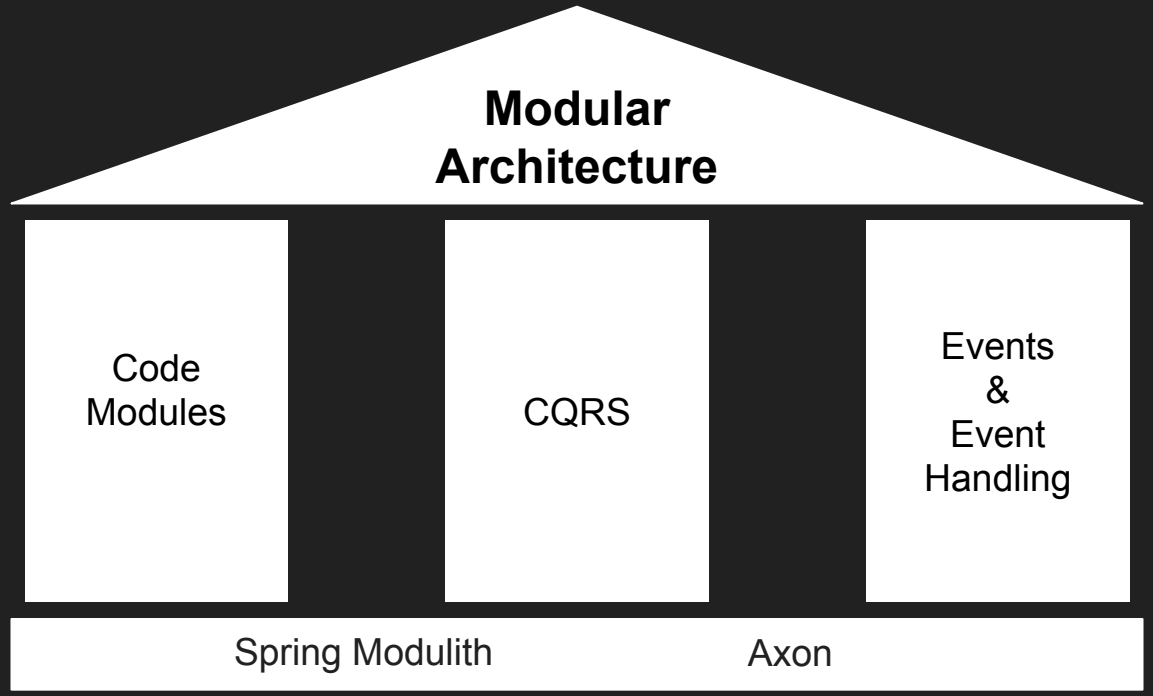
Typical Challenges of Code Modularization

1. Transactional Coupling
   a. Synchronous “accumulation”
   b. Resource Commit Ordering

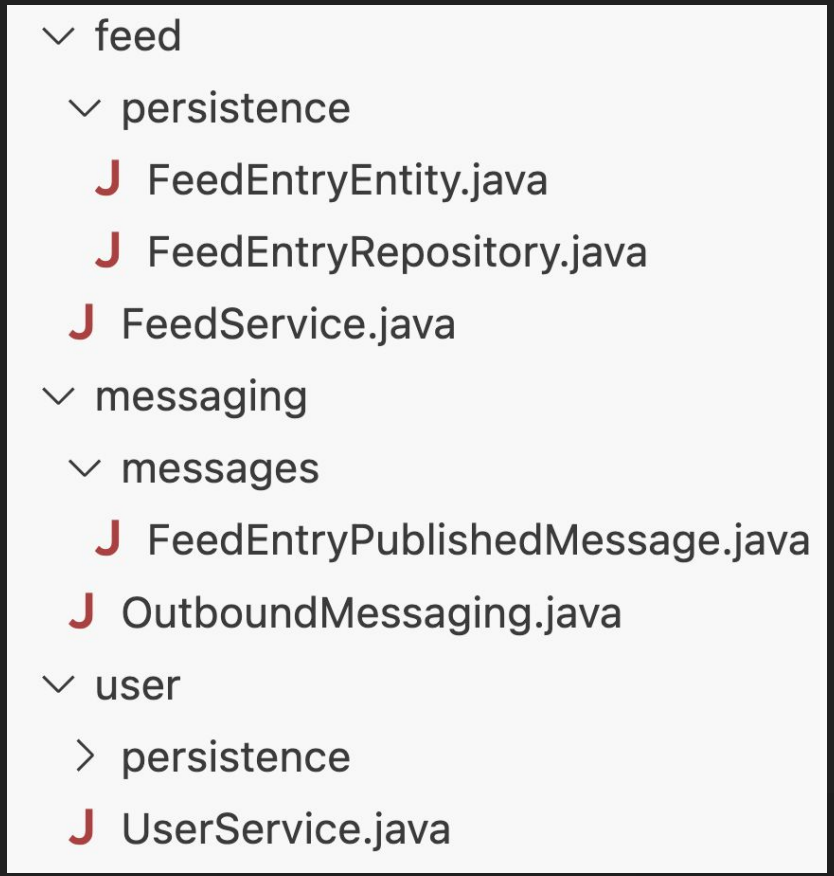
2. Technology Coupling
   a. Mixed domain logic and technological aspects
   b. Side effects between different technologies

3. Domain Coupling
   a. No enforced abstraction -> unclear side effects
   b. Lack of extensibility
   c. Circular dependencies

Solution Kit



Code-based Modularization



How to enforce modularization within the code?

FeedService

```
@Transactional
public void publishFeedEntry(UUID userId, String category, String entry) {
   UserEntity user = userRepository.findByUserId(userId);

   UUID feedEntryId = UUID.randomUUID();
   Instant publishedAt = Instant.now();

   feedEntryRepository.save(
           new FeedEntryEntity(feedEntryId, userId, category, entry, publishedAt)
   );

   outboundMessaging.publishFeedEntryPublishedMessage(
           new FeedEntryPublishedMessage(feedEntryId, userId, category, entry, publishedAt)
   );

   user.setLastUpdatedAt(publishedAt);
   userRepository.save(user);
}
```

Enforced using Spring Modulith


```java
@Test
public void isModularized() {
    ApplicationModules.of(MySpringBootApplication.class).verify();
}
```

```
Violations:
- Module 'feed' depends on non-exposed type UserRepository within module 'user'!
…
- Module 'feed' depends on non-exposed type UserEntity within module 'user'!
…
- Module 'feed' depends on non-exposed type FeedEntryPublishedMessage within module 'messaging'!
```

Modularization – Fixed Violations

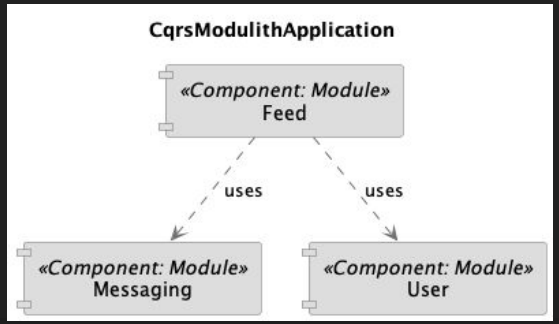
FeedService

```
@Transactional
public void publishFeedEntry(UUID userId, String category, String entry) {
    if (!userService.isValidUser(userId)) throw new RuntimeException();

    UUID feedEntryId = UUID.randomUUID();
    Instant publishedAt = Instant.now();

    feedEntryRepository.save(
            new FeedEntryEntity(feedEntryId, userId, category, entry, publishedAt)
    );

    outboundMessaging.publishFeedEntryPublishedMessage(feedEntryId, userId, category, entry, publishedAt);

    userService.updateLastUserActivity(userId, publishedAt);
}
```

Modularization – Generated Docs with Spring Modulith

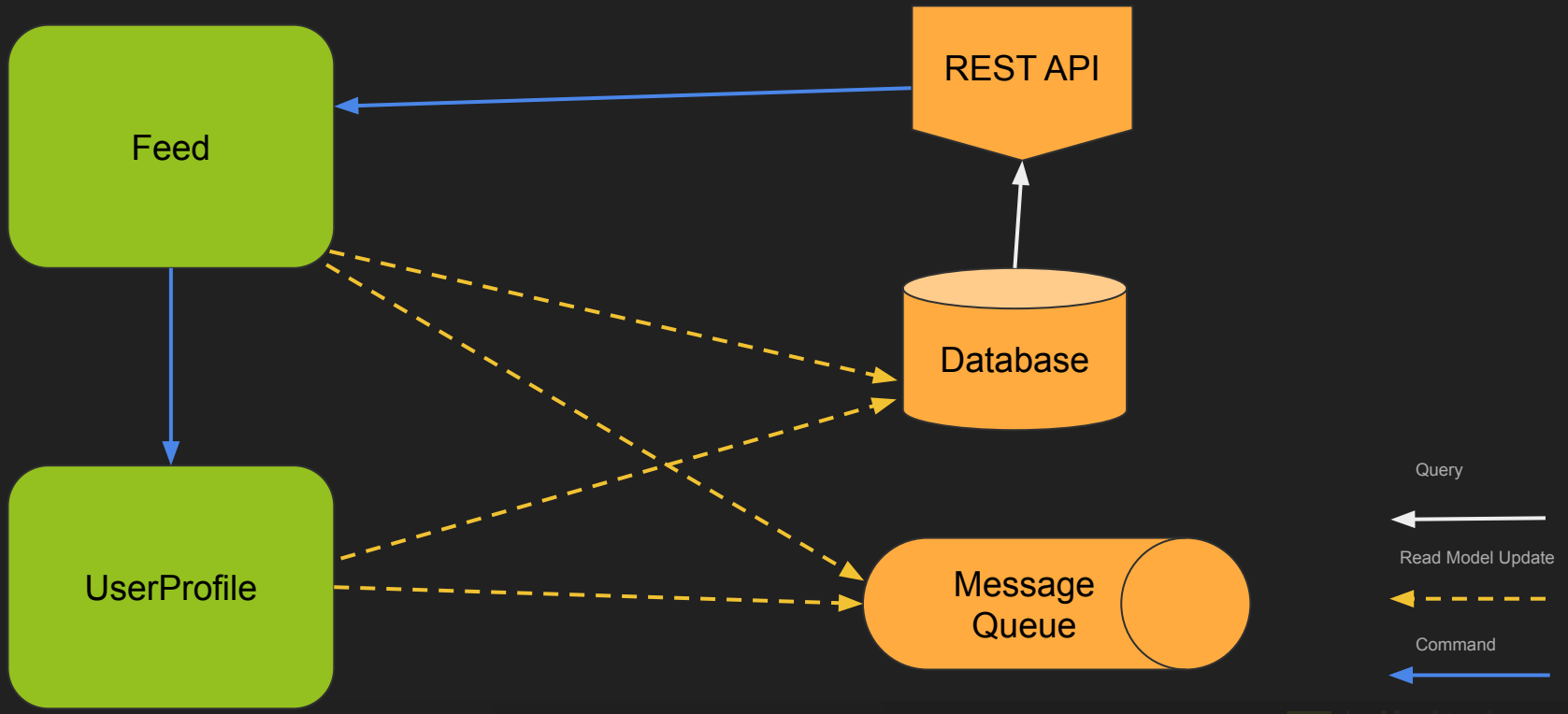

```java
@Test
public void renderDocumentation() {
   var canvasOptions = Documenter.CanvasOptions.defaults().revealInternals();

   var diagramOptions =
       Documenter.DiagramOptions.defaults().withStyle(Documenter.DiagramOptions.DiagramStyle.UML);

   new Documenter(ApplicationModules.of(CqrsModulithApplication.class))
           .writeDocumentation(diagramOptions, canvasOptions);
}
```

Base package	`de.dxfrontiers.cqrsmodulith.feed`
Spring components	*Services* • `d.d.c.f.FeedService` *Others* • `d.d.c.f.FeedCommandHandlers`
Bean references	• `d.d.c.u.UserService` (in User) • `d.d.c.m.OutboundMessaging` (in Messaging)



digital frontiers

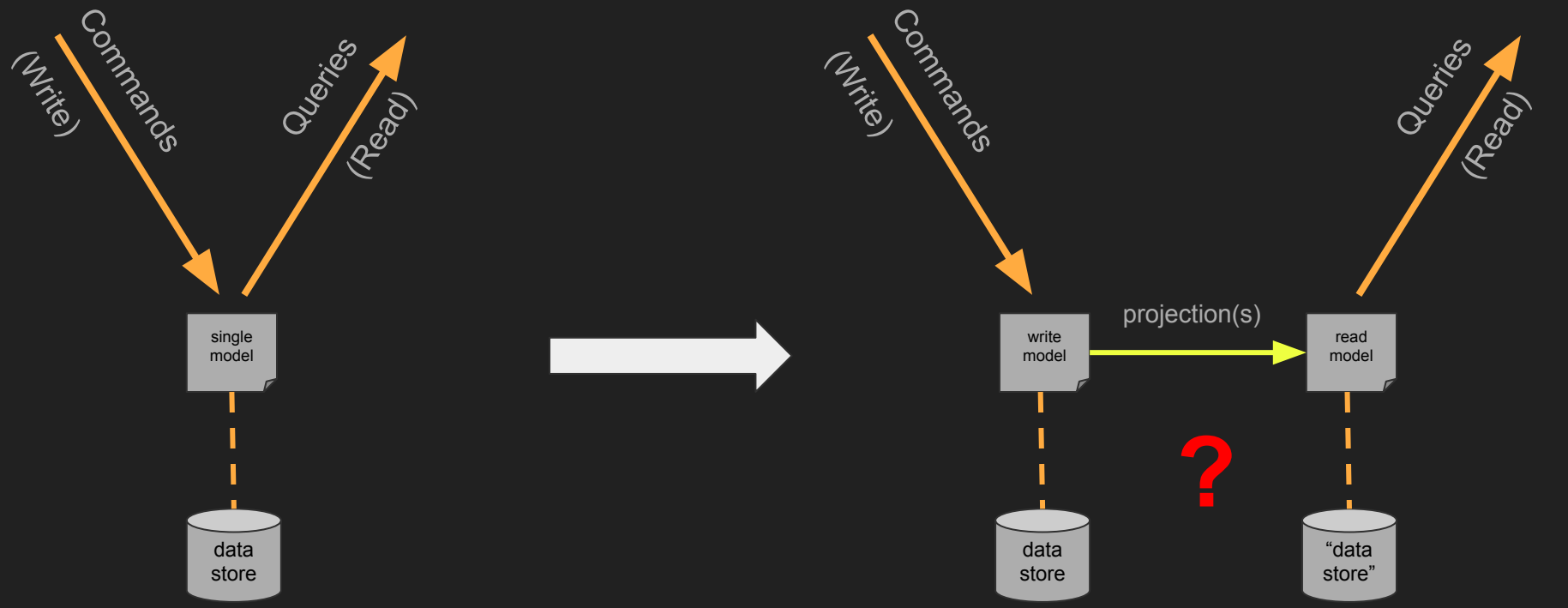
Command Query Responsibility Segregation (CQRS)



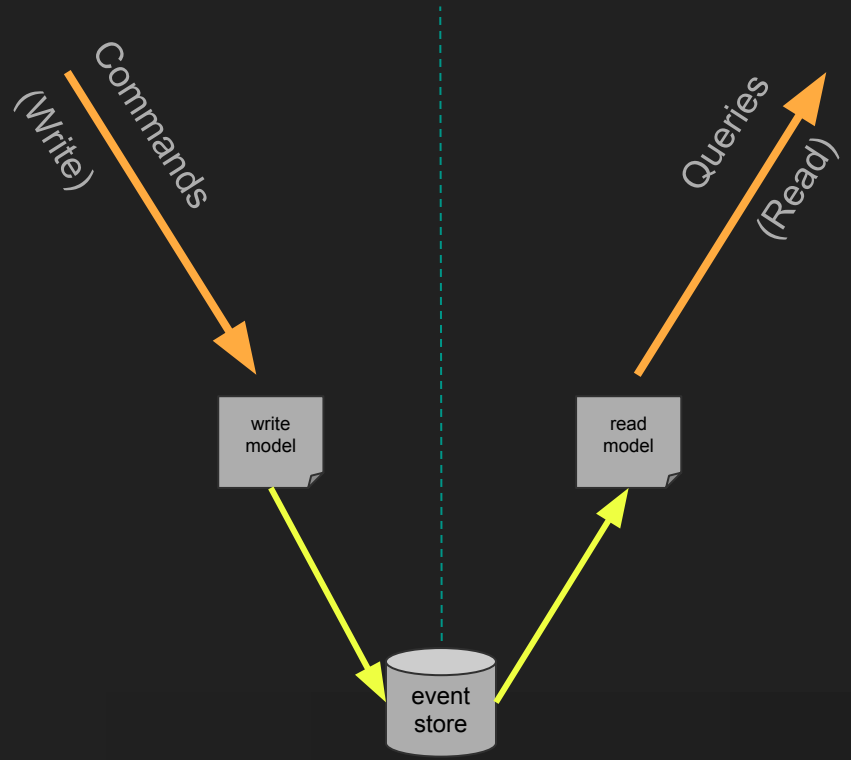
CQRS in a Nutshell

- Commands trigger changes
  - Validate inputs against current state (write model)
  - Update write model
  - “Trigger” read model updates
- Read Models reflect changes applied
  - Dedicated views per consumer
  - Extensible
- Queries deliver updated read model views

CQRS Read Model Projections



CQRS & Events



CQRS Example using Axon – Command Handler

FeedCommandHandlers

```
private final EventGateway eventGateway;

@CommandHandler
public void handle(PublishFeedEntryCommand command) {
    if (!userService.isValidUser(command.userId())) throw new RuntimeException();

    UUID feedEntryId = UUID.randomUUID();
    Instant publishedAt = Instant.now();

    feedEntryRepository.save(
        new FeedEntryEntity(feedEntryId, command.userId(), command.category(), command.entry(), publishedAt)
    );

    eventGateway.publish(
        new FeedEntryPublishedEvent(feedEntryId, command.userId(), command.category(), command.entry(), publishedAt)
    );
}
```

CQRS Example using Axon – Command Execution

FeedApiController

```
private final CommandGateway commandGateway;

@PostMapping
public void publishFeedEntry(@RequestBody PublishFeedEntryBody body) {
    commandGateway.sendAndWait(
        new PublishFeedEntryCommand(body.userId(), body.category(), body.entry())
    );
}
```

CQRS Example using Axon – Event Handling

OutboundMessaging

```
@EventHandler
public void on(FeedEntryPublishedEvent event) {
    FeedEntryPublishedMessage message = new FeedEntryPublishedMessage(
        event.feedEntryId(), event.userId(), event.category(), event.entry(), event.publishedAt()
    );

    publish("feed-topic", message);
}
```

CQRS Example using Axon – Event Handling

Read Model Update

user.FeedEventHandlers

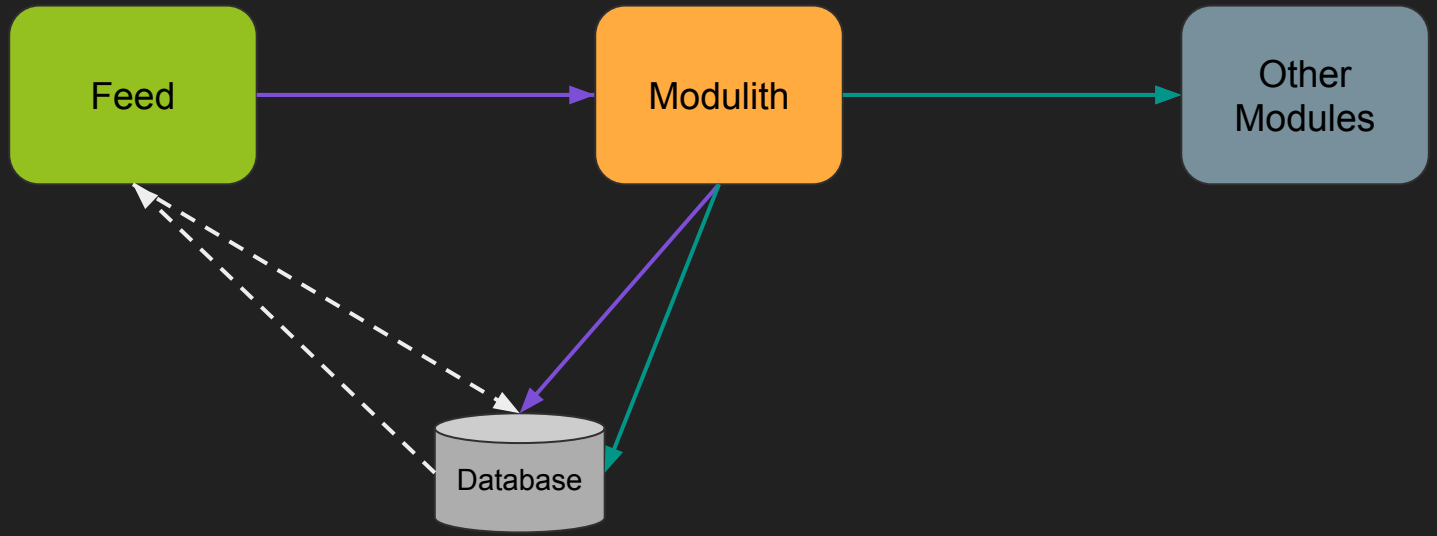
```
@EventHandler
public void on(FeedEntryPublishedEvent event) {
    UserEntity user = this.userRepository.findByUserId(event.userId());

    user.setLastUpdatedAt(event.publishedAt());

    userRepository.save(user);
}
```

Aggregate Update

feed.FeedEventHandlers

```
@EventHandler
public void on(FeedEntryPublishedEvent event) {
    commandGateway.sendAndWait(
        new UpdateUserLastActiveCommand(…)
    );
}
```

Leveraging Events using Spring Modulith



Spring Modulith – Publishing Events

FeedCommandHandlers

```
private final ApplicationEventPublisher eventPublisher;

@Transactional
public void handle(PublishFeedEntryCommand command) {
    if (!userService.isValidUser(command.userId())) throw new RuntimeException();

    UUID feedEntryId = UUID.randomUUID();
    Instant publishedAt = Instant.now();

    feedEntryRepository.save(
            new FeedEntryEntity(feedEntryId, command.userId(), command.category(), command.entry(), publishedAt)
    );

    eventPublisher.publishEvent(
            new FeedEntryPublishedEvent(feedEntryId, command.userId(), command.category(), command.entry(), publishedAt)
    );
}
```

Spring Modulith – REST Controller

FeedApiController

```
private final FeedCommandHandlers feedCommandHandlers;

@PostMapping
public void publishFeedEntry(@RequestBody PublishFeedEntryBody body) {
    feedCommandHandlers.handle(
            new PublishFeedEntryCommand(body.userId(), body.category(), body.entry())
    );
}
```

Spring – Handling Events (no Modulith yet)

Read Model Update

user.FeedEventHandlers

```
@TransactionalEventListener
@Transactional(propagation = Propagation.REQUIRES_NEW)
@Async
public void on(FeedEntryPublishedEvent event) {
    UserEntity user = this.userRepository.findByUserId(event.userId());

    user.setLastUpdatedAt(event.publishedAt());

    userRepository.save(user);
}
```

Spring Modulith – Handling Events

Read Model Update

FeedEventHandlers

```
@ApplicationModuleListener
public void on(FeedEntryPublishedEvent event) {
   UserEntity user = this.userRepository.findByUserId(event.userId());

   user.setLastUpdatedAt(event.publishedAt());

   userRepository.save(user);
}
```

Spring Modulith – Externalizing Events

FeedEntryPublishedEvent

```
@Externalized("feed-topic")
public record FeedEntryPublishedEvent(
        UUID feedEntryId,
        UUID userId
        …
) { }
```

Supported Externalizations:

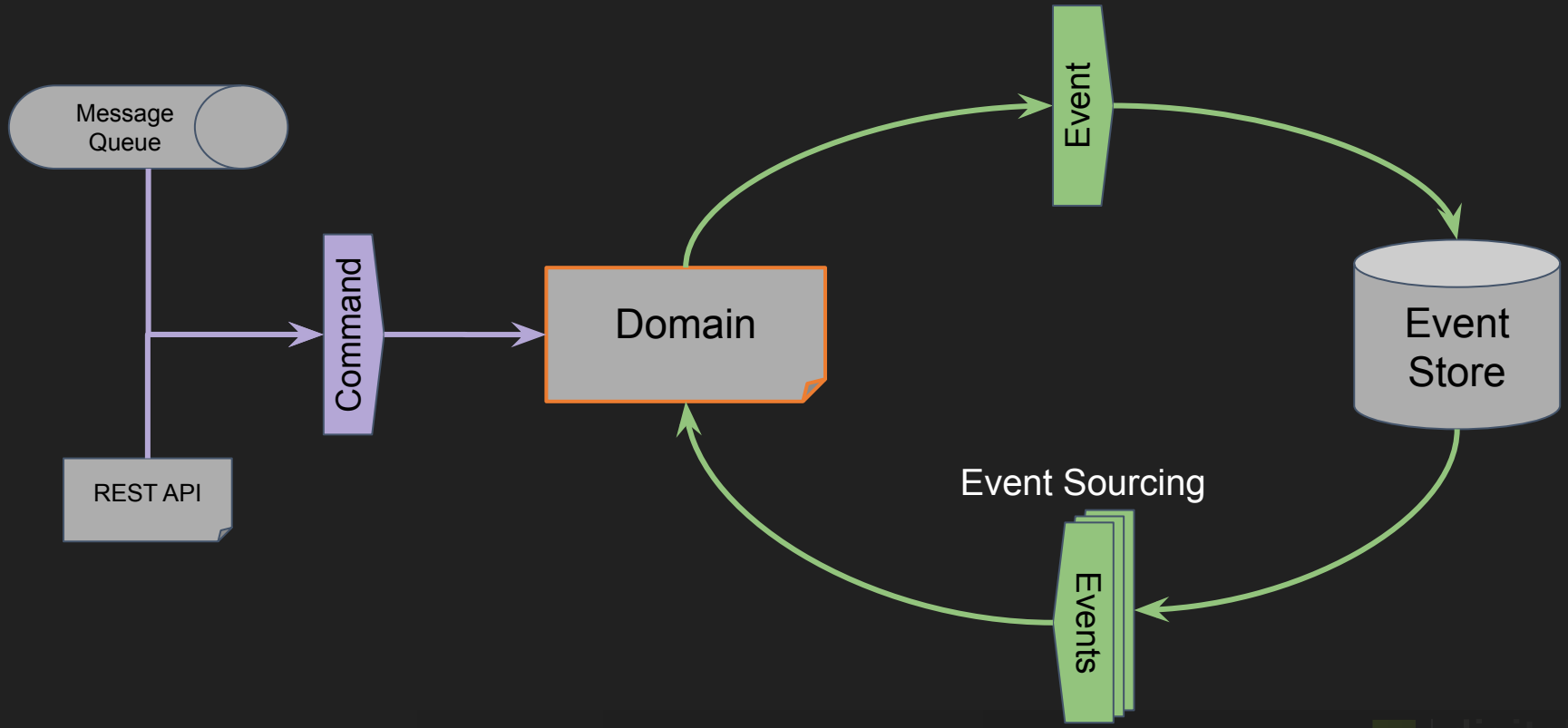
- Kafka
- AMQP (RabbitMQ)
- JMS
- AWS-SQS
- AWS-SNS

Recap

	MODULITH	AXON
Modularization Support	✔	
Easy Testing of Modules	✔	✔
Read / Write Models	✔	✔
CQRS		✔
Minimal Overhead	✔	✘
Generatable Documentation	✔	✘

Is Spring Modulith the next CQRS/ES-Framework?

CQRS & Event Sourcing (ES)



Axon - Event Sourcing

FeedAggregate

```
@Aggregate
public class FeedAggregate {

    @AggregateIdentifier
    private UUID id;

    @CommandHandler
    public FeedAggregate(PublishFeedEntryCommand command) {
        AggregateLifecycle.apply(new FeedEntryPublishedEvent());
    }

    @EventSourcingHandler
    public void on(FeedEntryPublishedEvent event) {
        this.id = event.feedEntryId();
    }

    @CommandHandler
    public void handle(ModifyFeedEntryCommand command) { }
}
```

Summary

	MODULITH	AXON
Modularization Support	✔	
Easy Testing of Modules	✔	✔
Read / Write Models	✔	✔
CQRS		✔
Minimal Overhead	✔	✘
Generatable Documentation	✔	✘
Event Sourcing	✘	✔

Ist Spring Modulith das nächste CQRS/ES-Framework?

Nikolai Neugebauer

@nik101010
https://www.linkedin.com/in/nikolai-neugebauer
nikolai.neugebauer@digitalfrontiers.de

https://blog.digitalfrontiers.de

https://github.com/dxfrontiers

@dxfrontiers

https://www.linkedin.com/company/digital-frontiers-gmbh-co-kg

Frank Scheffler

@maverick_1601
https://www.linkedin.com/in/frank-scheffler-64b918148
frank.scheffler@digitalfrontiers.de

digital frontiers