云智未来 9th

第九届中国系统架构师大会

SYSTEM ARCHITECT CONFERENCE CHINA 2017

Event Sourcing & CQRS

architecting a cloud based micro-service system

About me

- Oracle Certified Expert J2EE
- Software Engineer
- Software Architect

HypoVereinsbank
Member of UniCredit

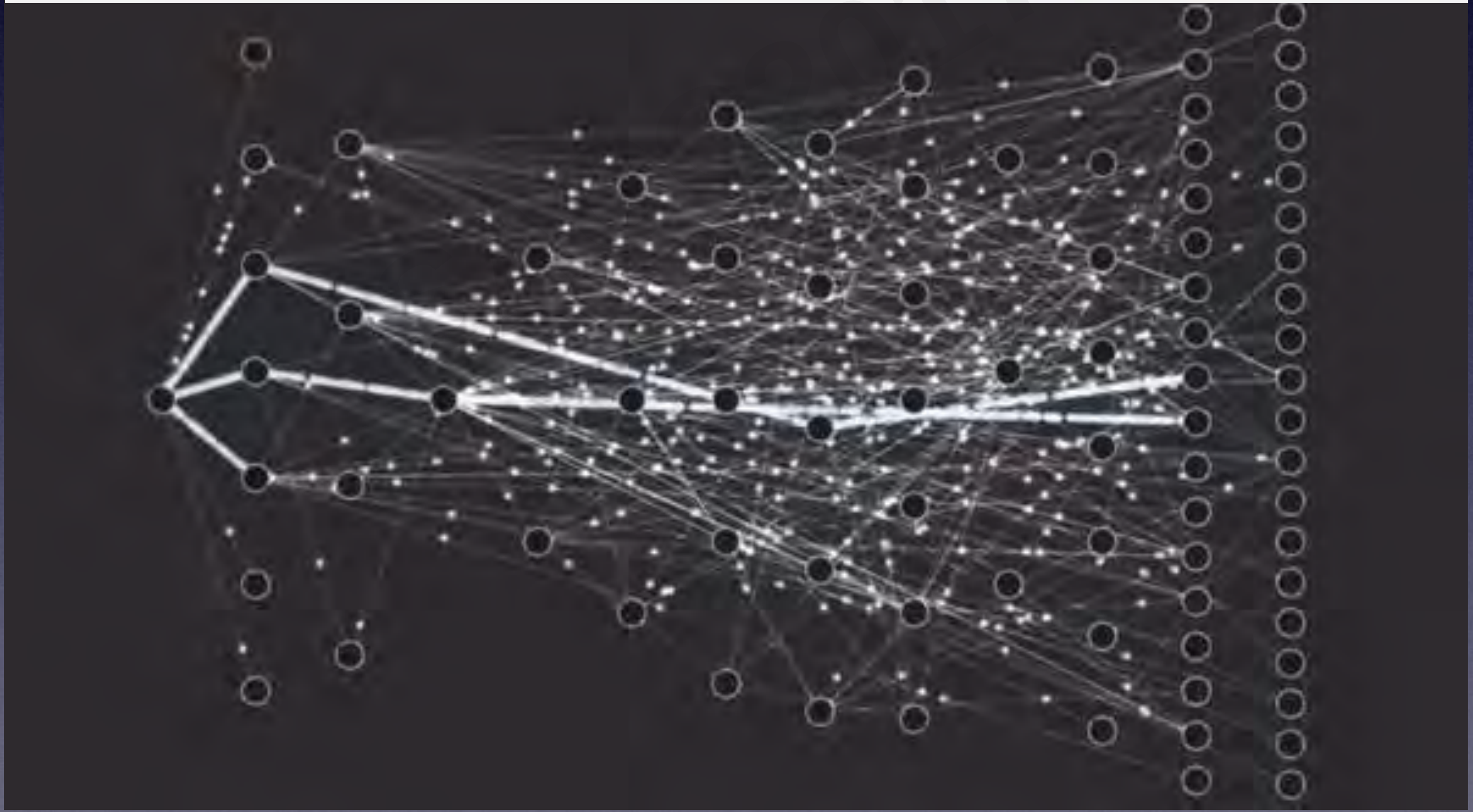
accenture
High performance. Delivered.

Datalex
You get a better view of people from here.

Agenda

- Event Driven Architecture
- Event Sourcing
- CQRS
- Lagom in a nut shell
- Demo
- Q&A

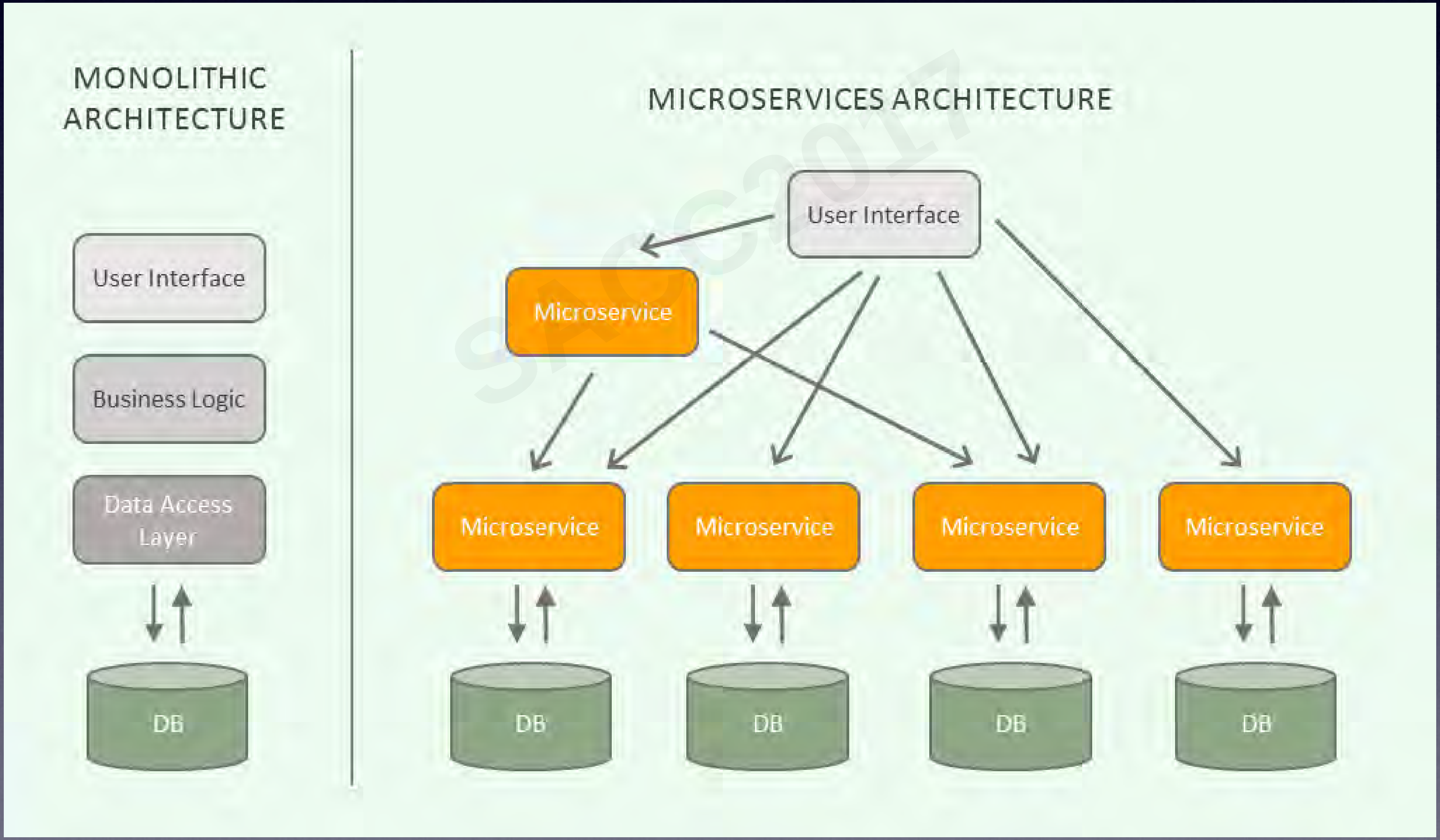
NETFLIX



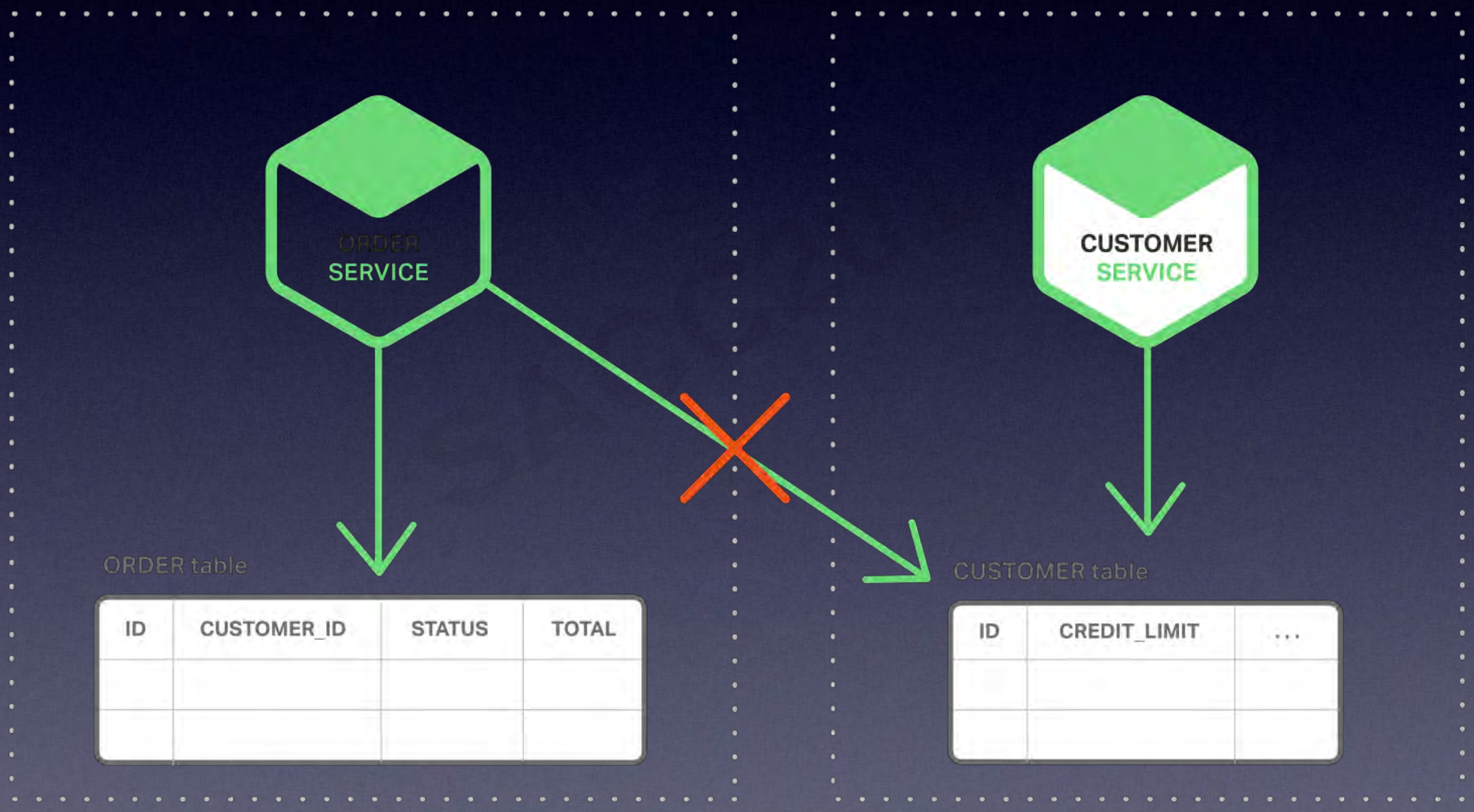
What we need?

- Availability
- Performance
- Scalability
- Resiliency
- Innovation

Monolith VS Micro-Service

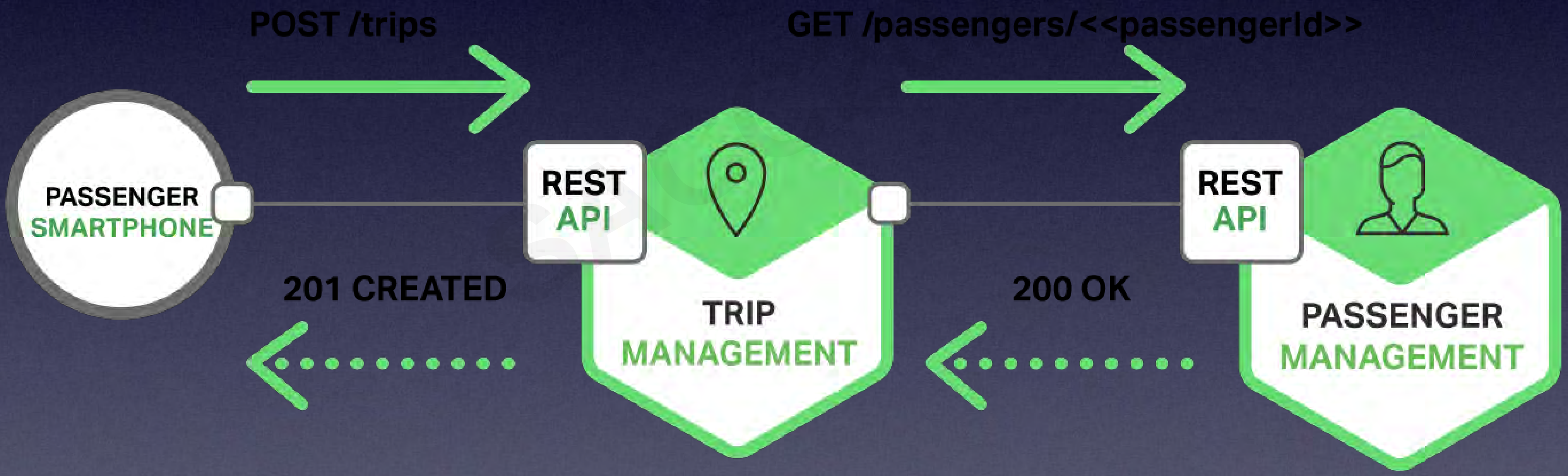


Inter-service Communication

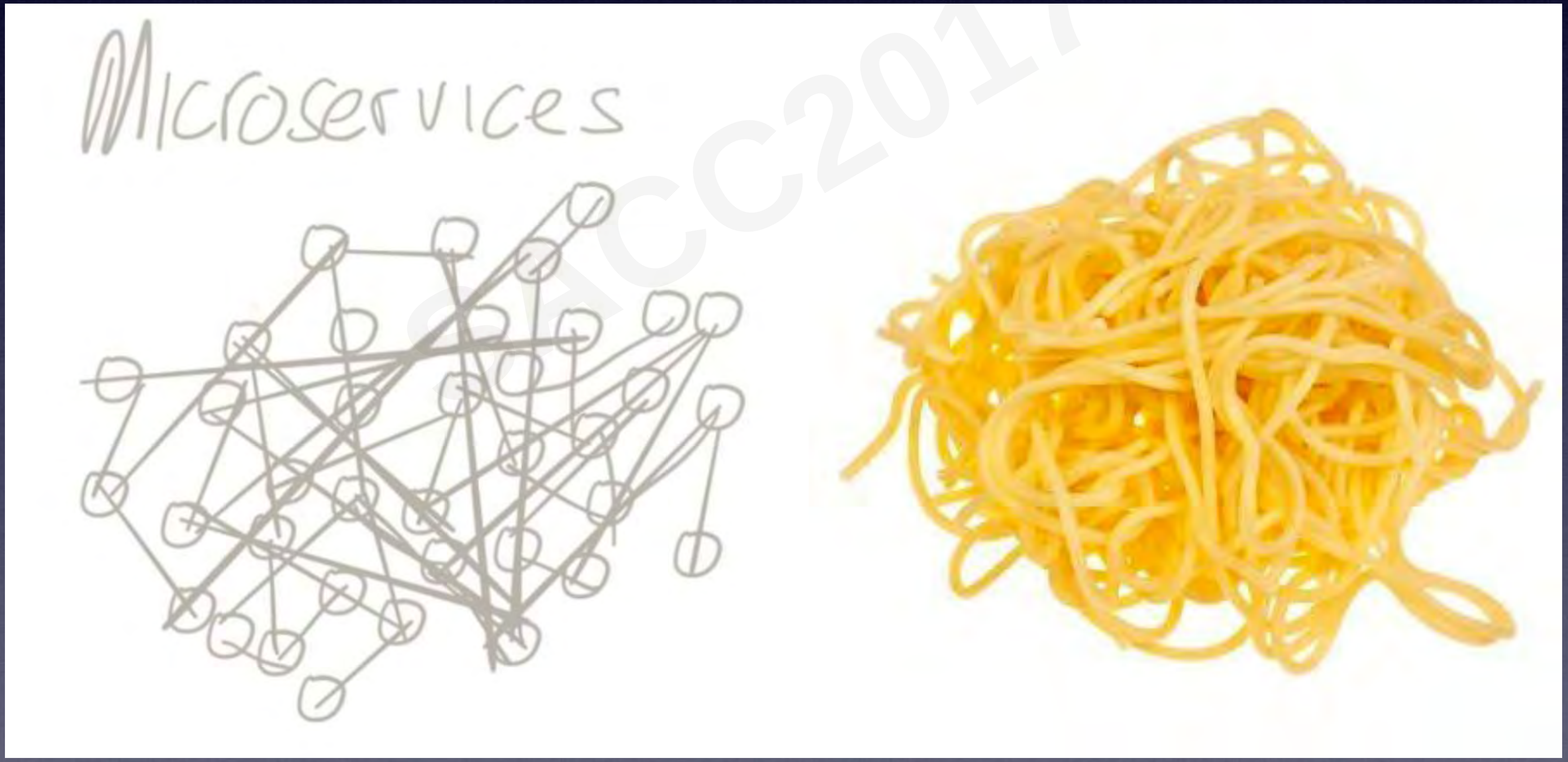


24Slides
NO.
A big NO NO !

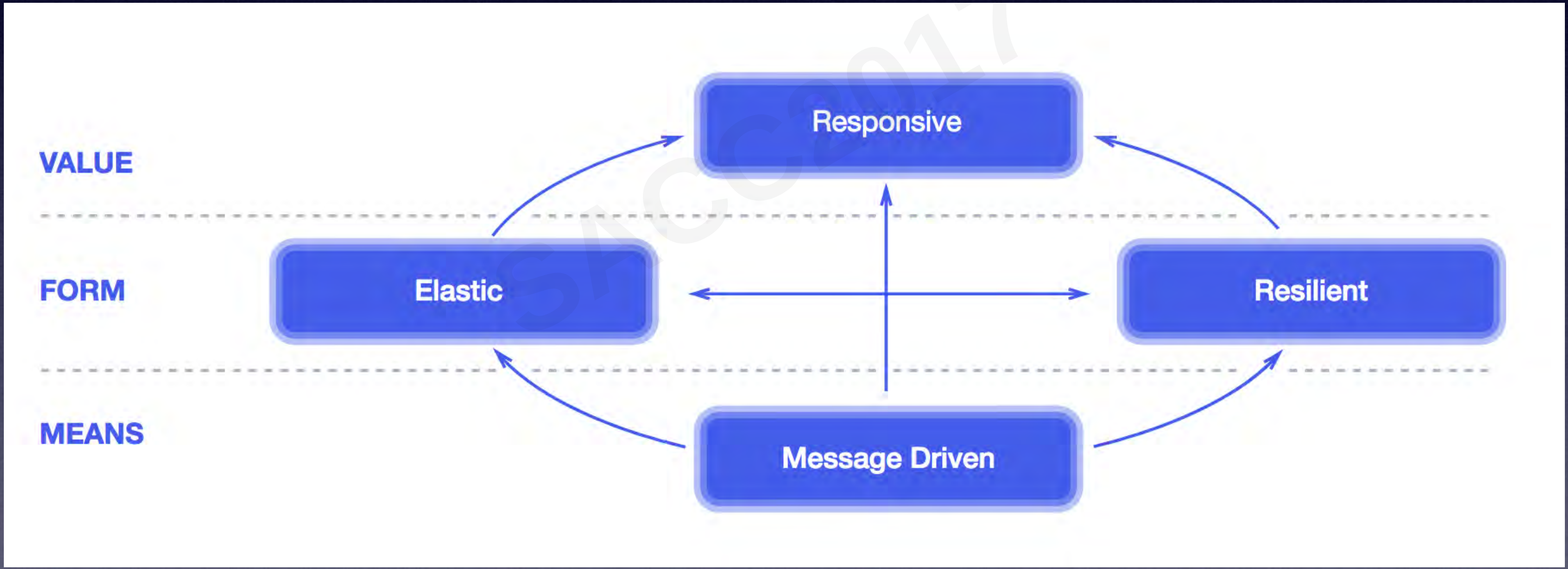
REST API Call



Micro service spaghetti



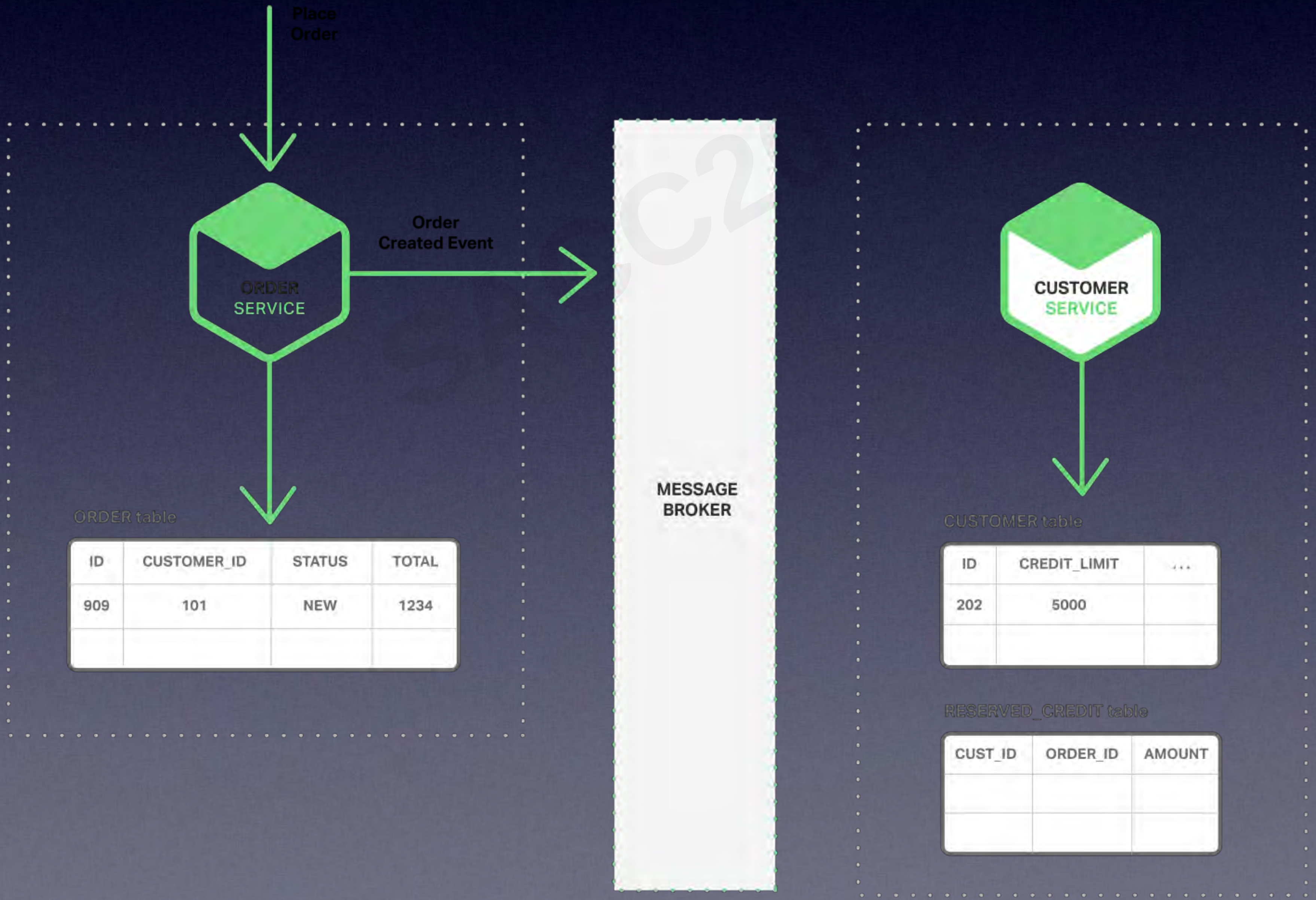
Reactive Manifesto



Event Driven Architecture

- Production, detection, consumption and reaction to events
- Services publish events when there is a state change
- Other services subscribe events to obtain state changes
- To achieve a loosely coupled distributed system

Event Driven Architecture

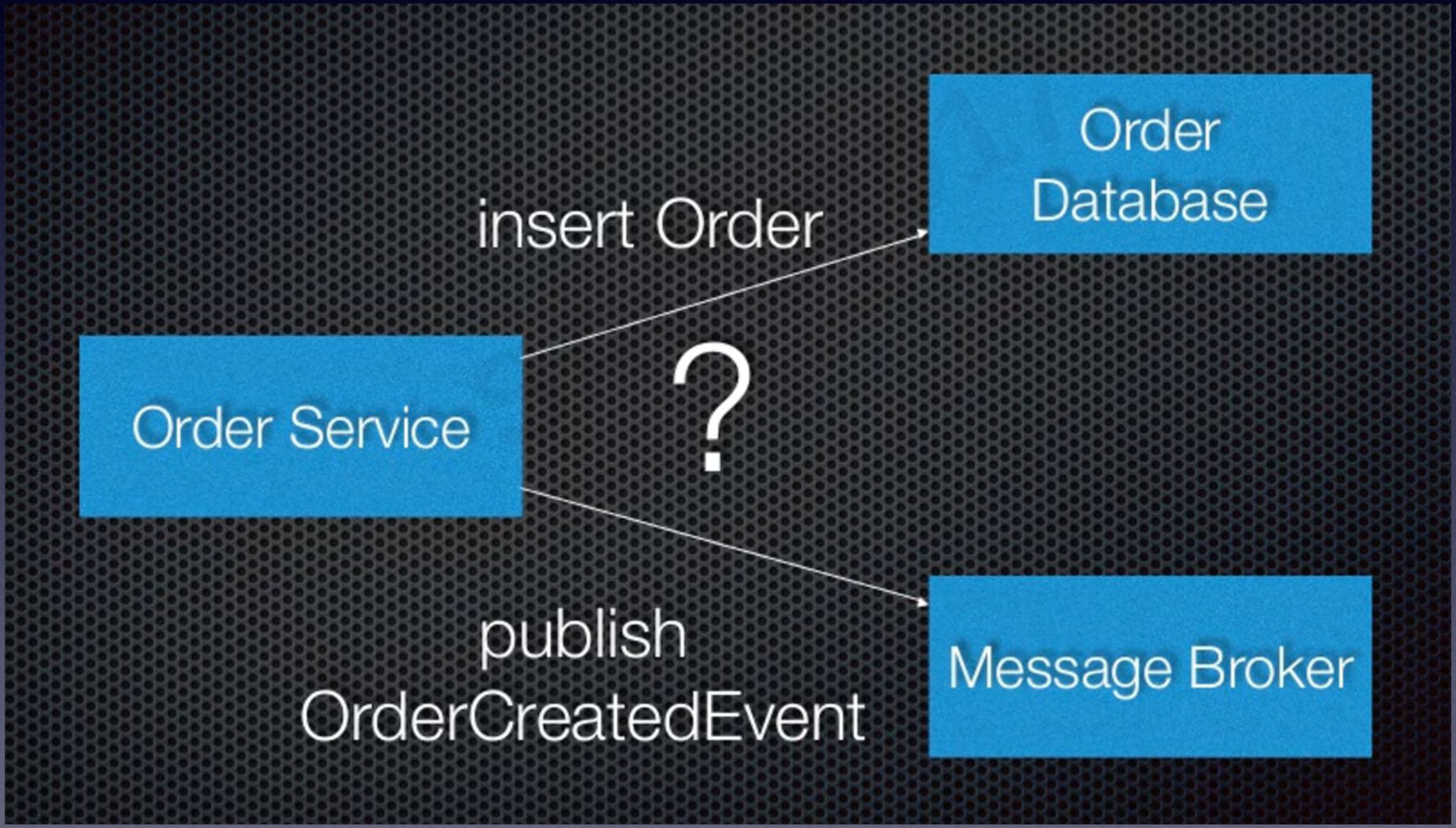


ID	CUSTOMER_ID	STATUS	TOTAL
909	101	NEW	1234

ID	CREDIT_LIMIT	...
202	5000	

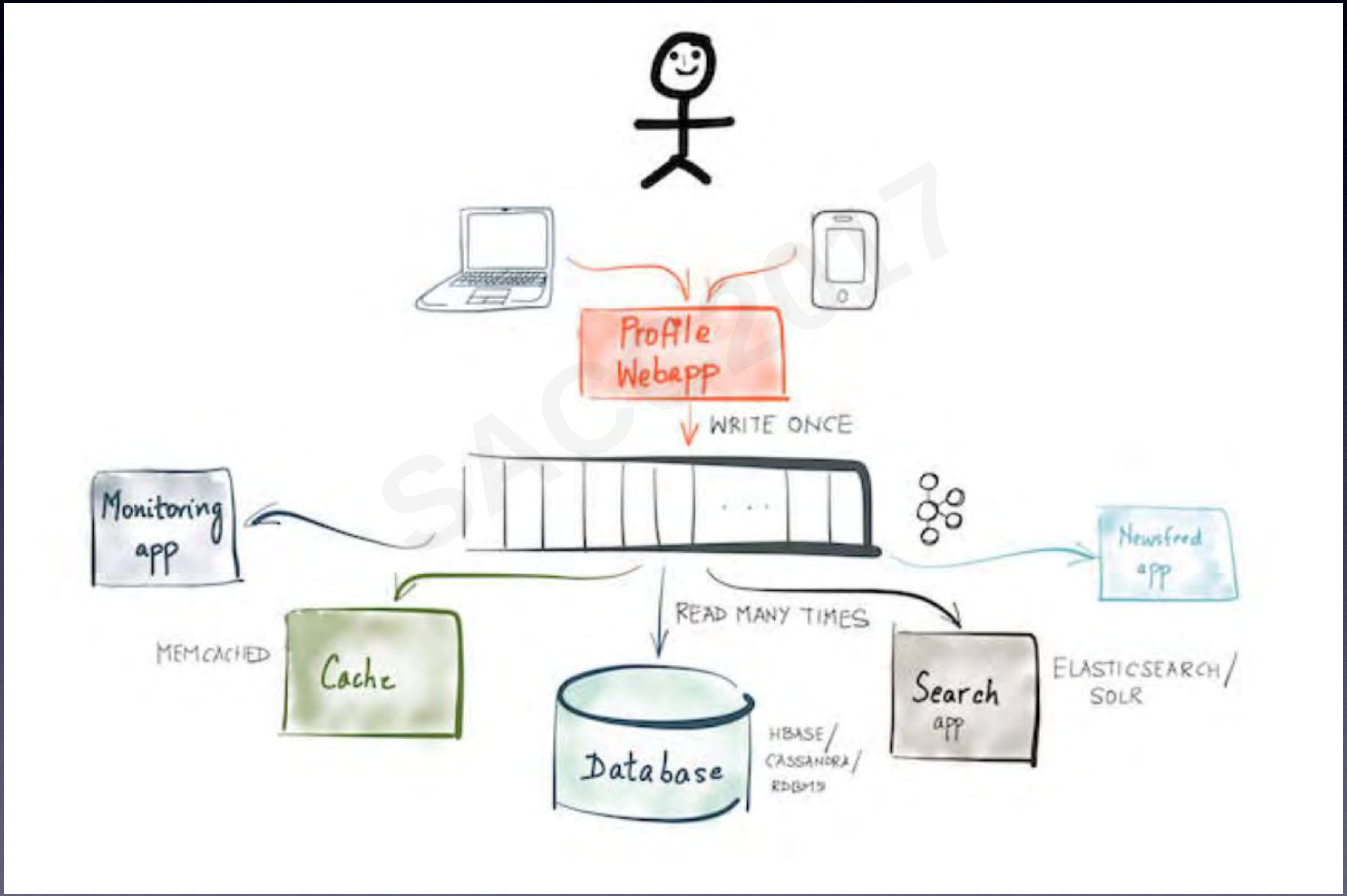
CUST_ID	ORDER_ID	AMOUNT

Problem: How Atomicity update database and publish event



https://www.slideshare.net/chris.e.richardson/developing-eventdriven-microservices-with-event-sourcing-and-cqrs-svcc-svcc2015

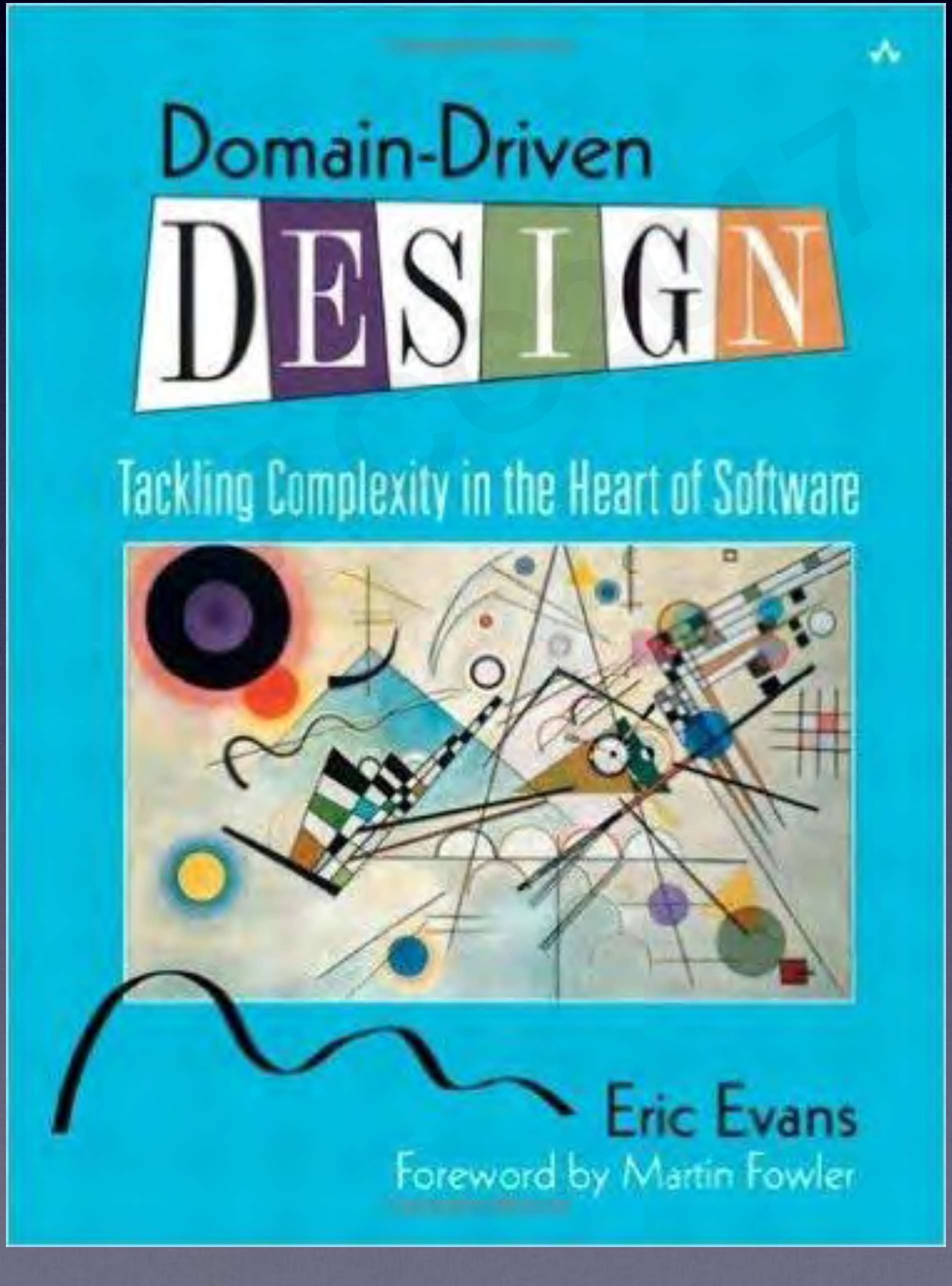
Kafka as a message bus



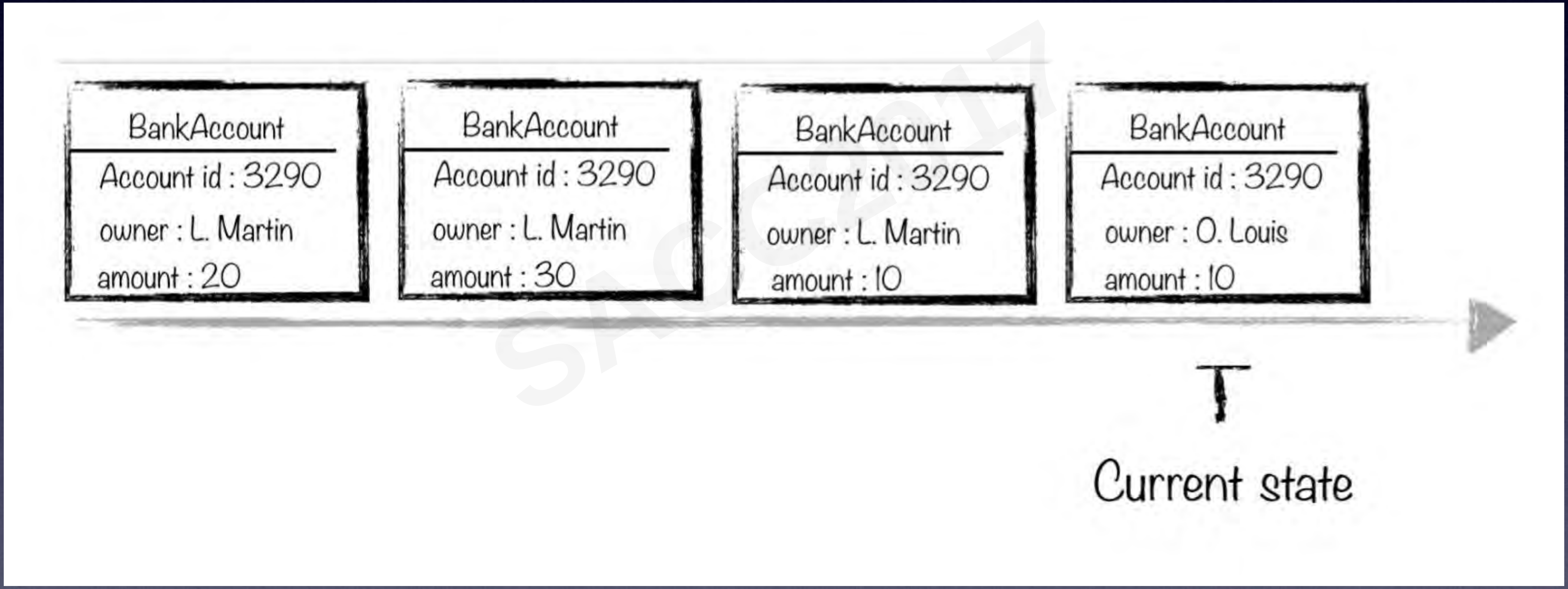
https://www.confluent.io/blog/event-sourcing-cqrs-stream-processing-apache-kafka-whats-connection/

Event Sourcing

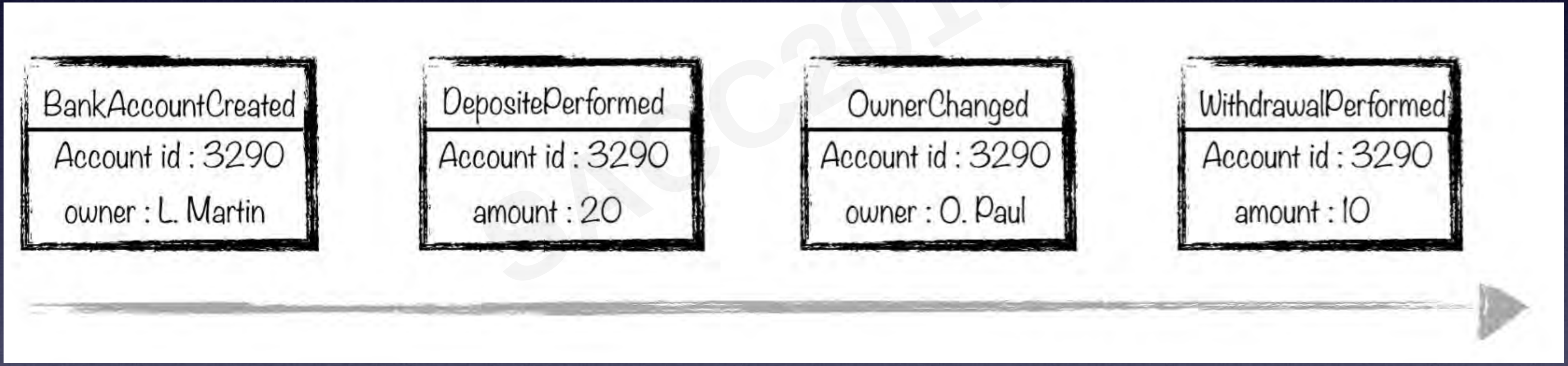
a core concept from DDD


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

Active current record



Actual event happened



no update, no delete
only append

Event sourcing like application

- bank statement
- git
- database transaction log

Events

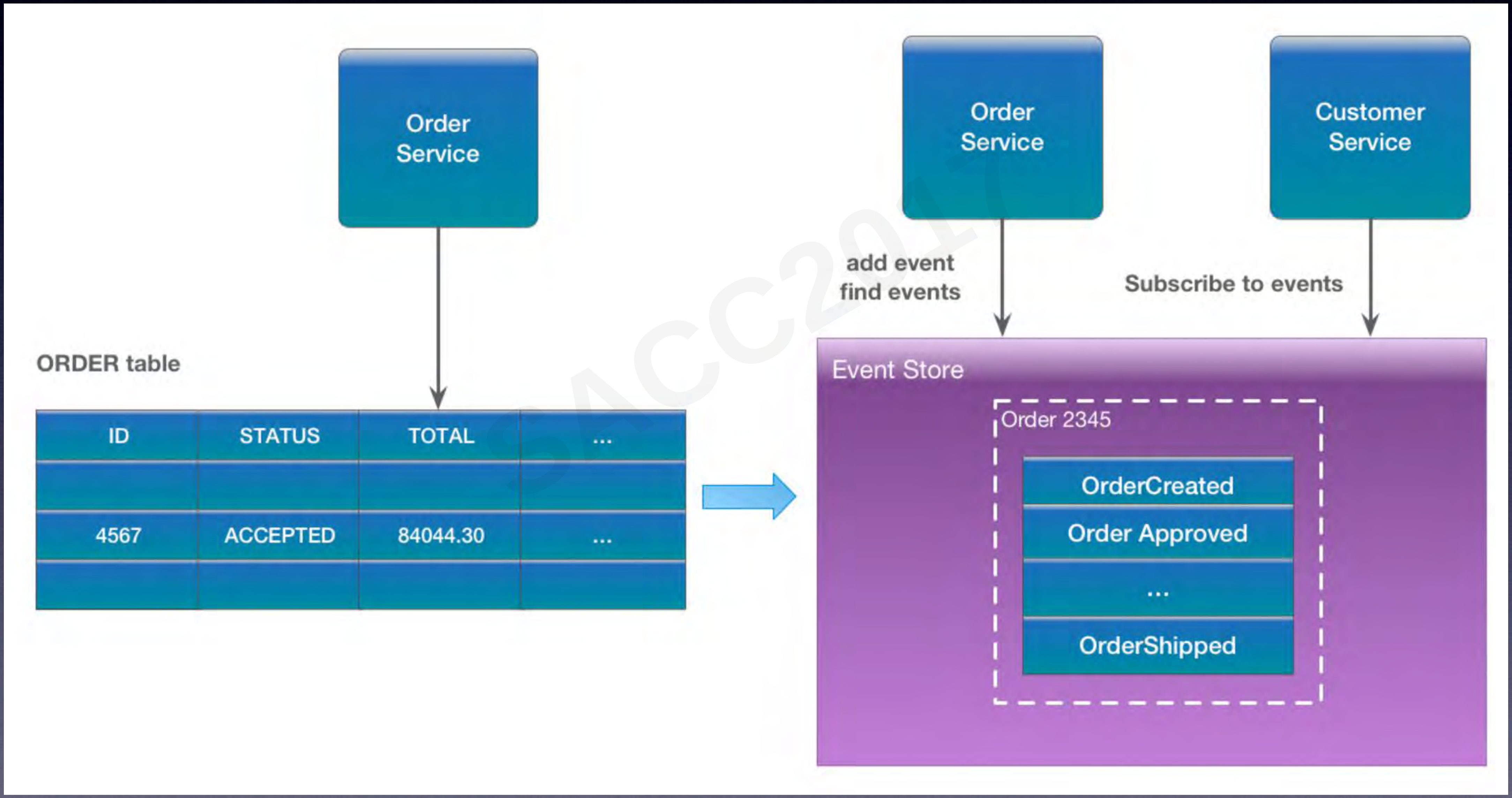
- History, can never be changed
- Immutable, we love immutability
- Generate corrective events
- Allow asynchronous communication

Event Sourcing

- Define the aggregates, or domain entities
- Identify domain events
- Capture all changes (commands) and convert them to domain events

- Examples:
  - Bank Account : accountCreated; depositPerformed; withdrawalPerformed
  - Order : orderCreated; orderUpdated; orderPaid; orderShipped

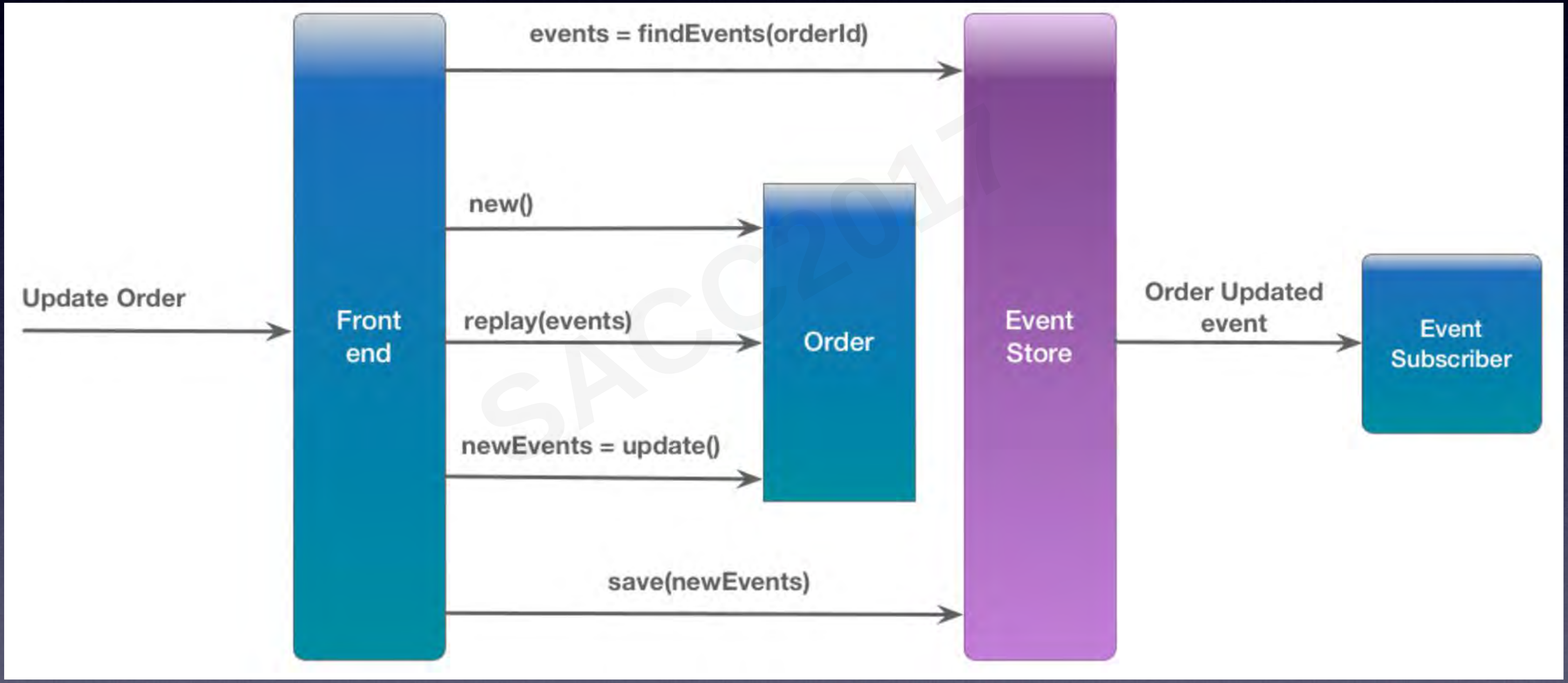
Event Sourcing



ID	STATUS	TOTAL	...
4567	ACCEPTED	84044.30	...

http://eventuate.io/whyeventsourcing.html

Update an order



http://eventuate.io/whyeventsourcing.html

Benefits

- 100% accurate audit logging
- easy debugging
- real time stream processing
- writing is blazing fast, since it is append only
- no impedance mismatch
- building a forward-compatible application architecture

But!

- What’s the total amount of all transaction from this month?
- Search all products with the brand Gucci and cost less than 1000 euro for female?

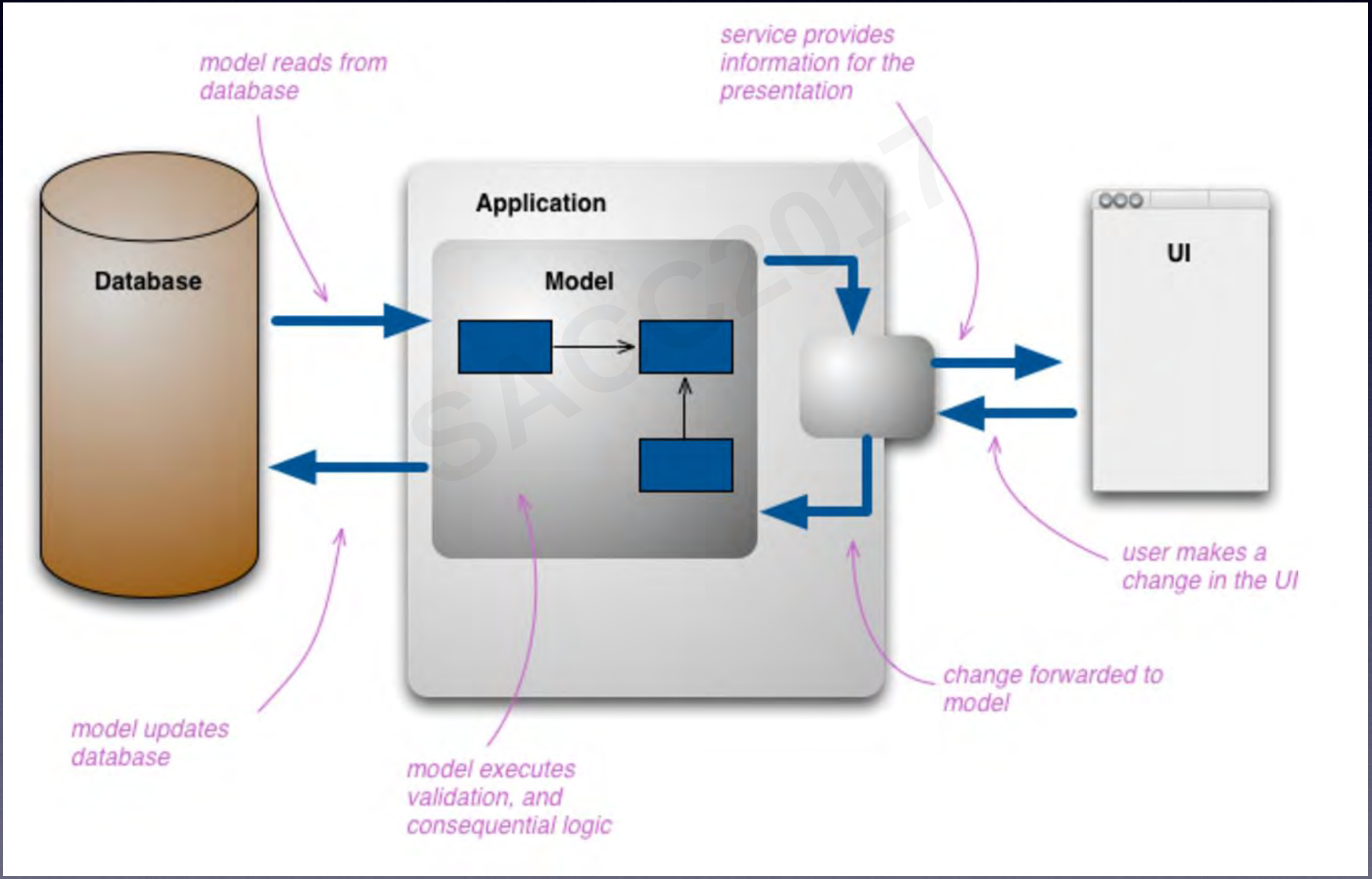
Oh dear !

CQRS

CQRS

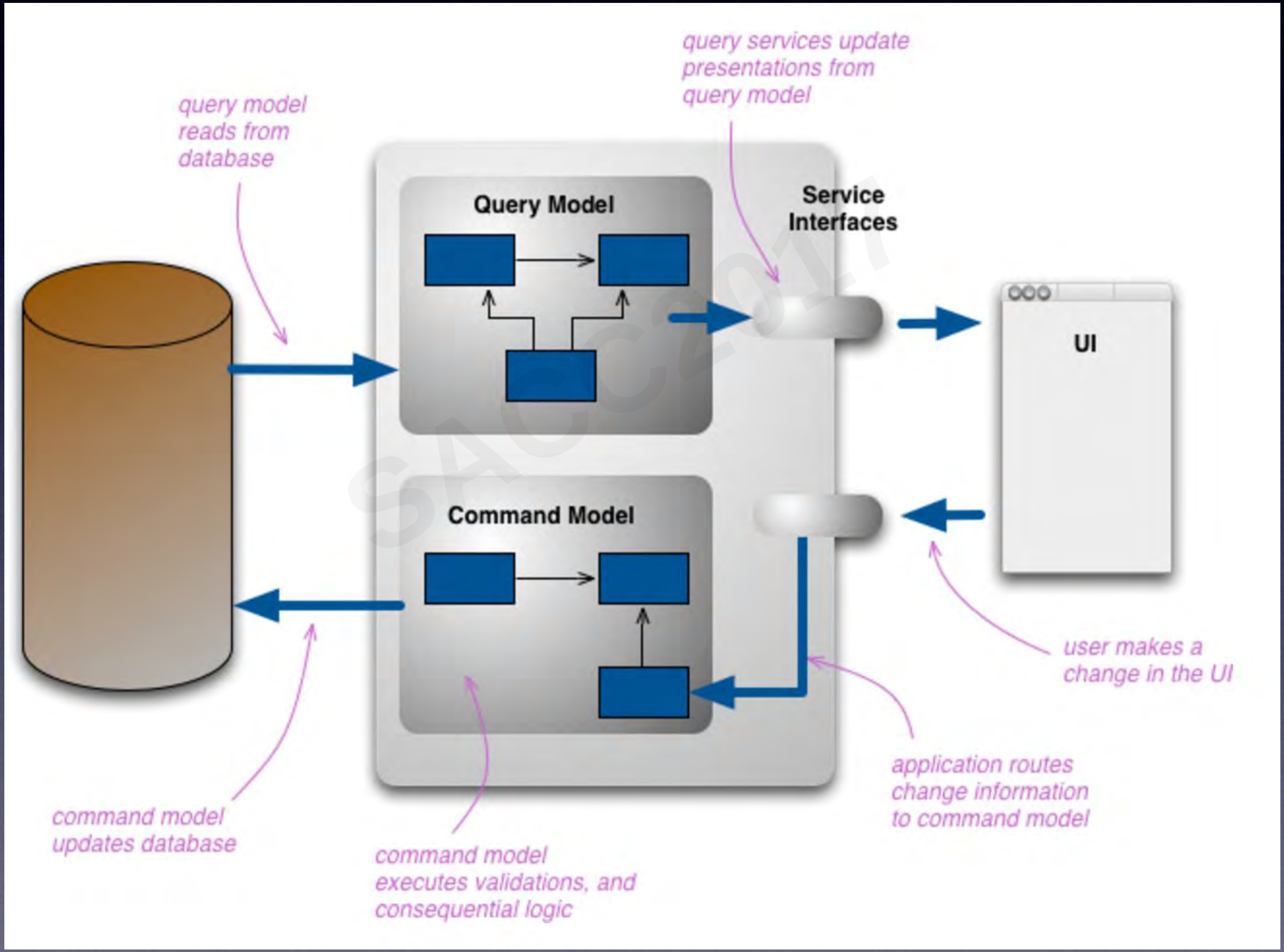
C : command -> write/append
Q : query -> read
R : responsibility
S : segregation

Traditional approach


model reads from database
service provides information for the presentation
Application
Model
Database
UI
user makes a change in the UI
change forwarded to model
model updates database
model executes validation, and consequential logic

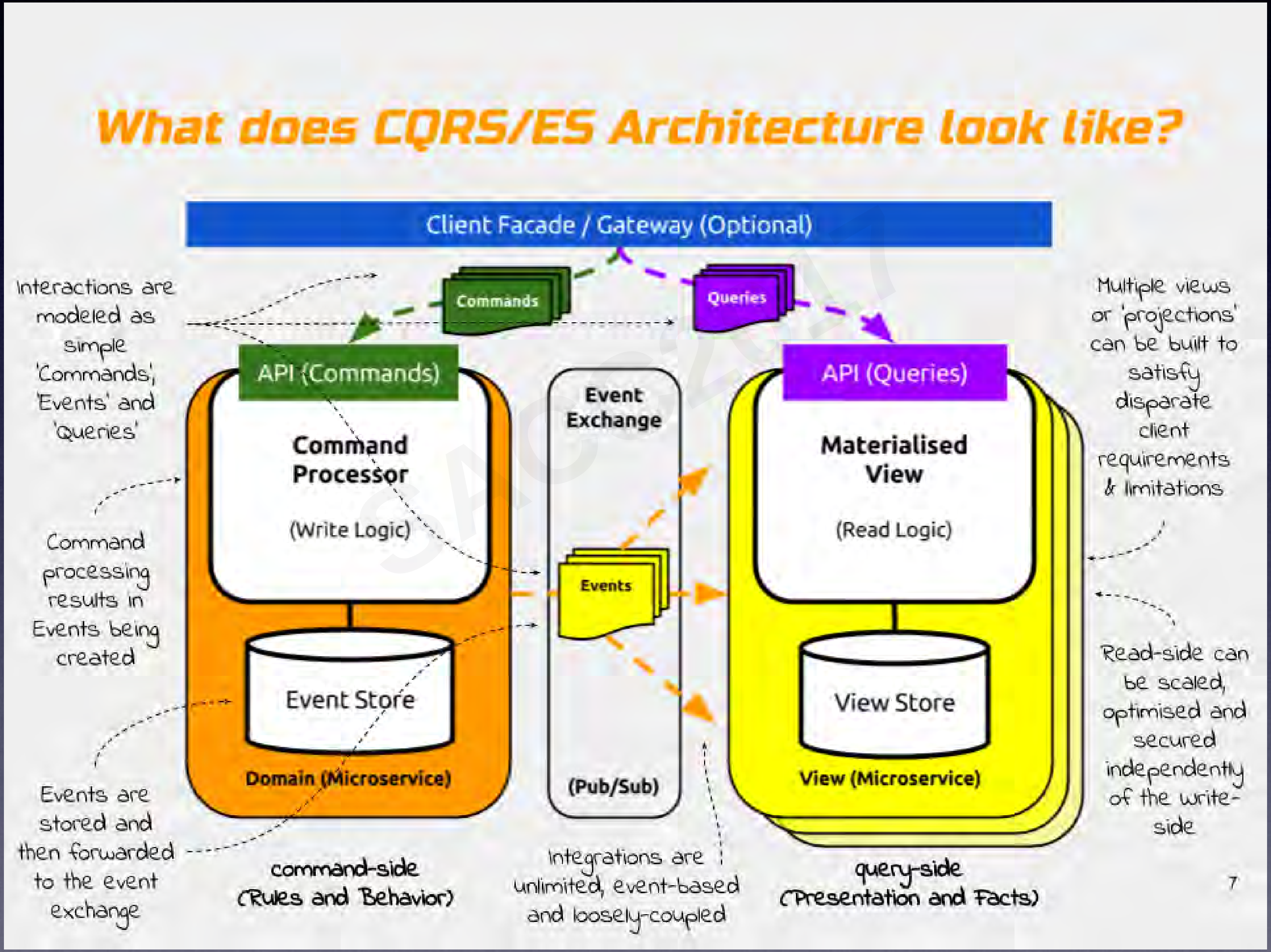
https://martinfowler.com/bliki/CQRS.html

CQRS


query model reads from database
query services update presentations from query model
Query Model
Service Interfaces
UI
Command Model
user makes a change in the UI
application routes change information to command model
command model updates database
command model executes validations, and consequential logic
SACC2017

https://martinfowler.com/bliki/CQRS.html

ES + CQRS



What does CQRS/ES Architecture look like?
Client Facade / Gateway (Optional)
Commands
Queries
API (Commands)
Command Processor
(Write Logic)
Event Store
Domain (Microservice)
Event Exchange
Events
(Pub/Sub)
API (Queries)
Materialised View
(Read Logic)
View Store
View (Microservice)
Interactions are modeled as simple 'Commands', 'Events' and 'Queries'
Command processing results in Events being created
Events are stored and then forwarded to the event exchange
command-side (Rules and Behavior)
Integrations are unlimited, event-based and loosely-coupled
query-side (Presentation and Facts)
Multiple views or 'projections' can be built to satisfy disparate client requirements & limitations
Read-side can be scaled, optimised and secured independently of the write-side
7

https://dzone.com/articles/microservices-with-spring-boot-axon-cqrses-anddock

Lagom

Lagom

- From Lightbend, a Scala company
- Opinionated Microservices Framework
- Early adoption phase
- Aims for micro-service system
- Based on actor model

Lagom Technology Stack

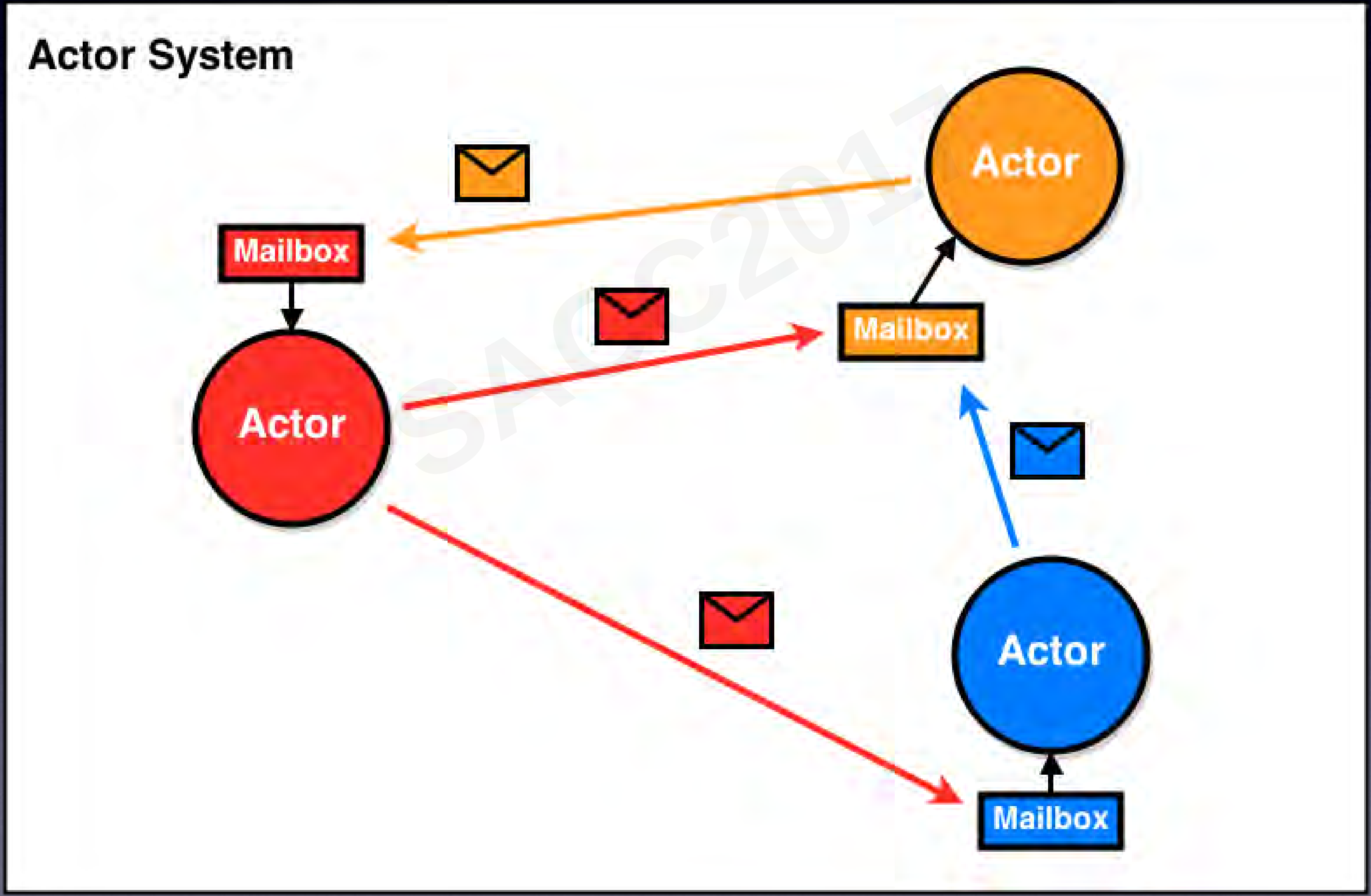
Scala Java play

akka kafka cassandra

API Overview

- Service API
  - Asynchronous streaming between services
  - Synchronous request-response calls
- Persistence API
    - Event-sourced persistent, CQRS read-side support
    - Persistent entities managed automatically across a cluster of nodes
  - Persistent entity is an actor
    - Cassandra
- Message Broker API
  - Publish-subscribe model
  - Share data via topics
  - Push and pull, back pressure
  - Kafka

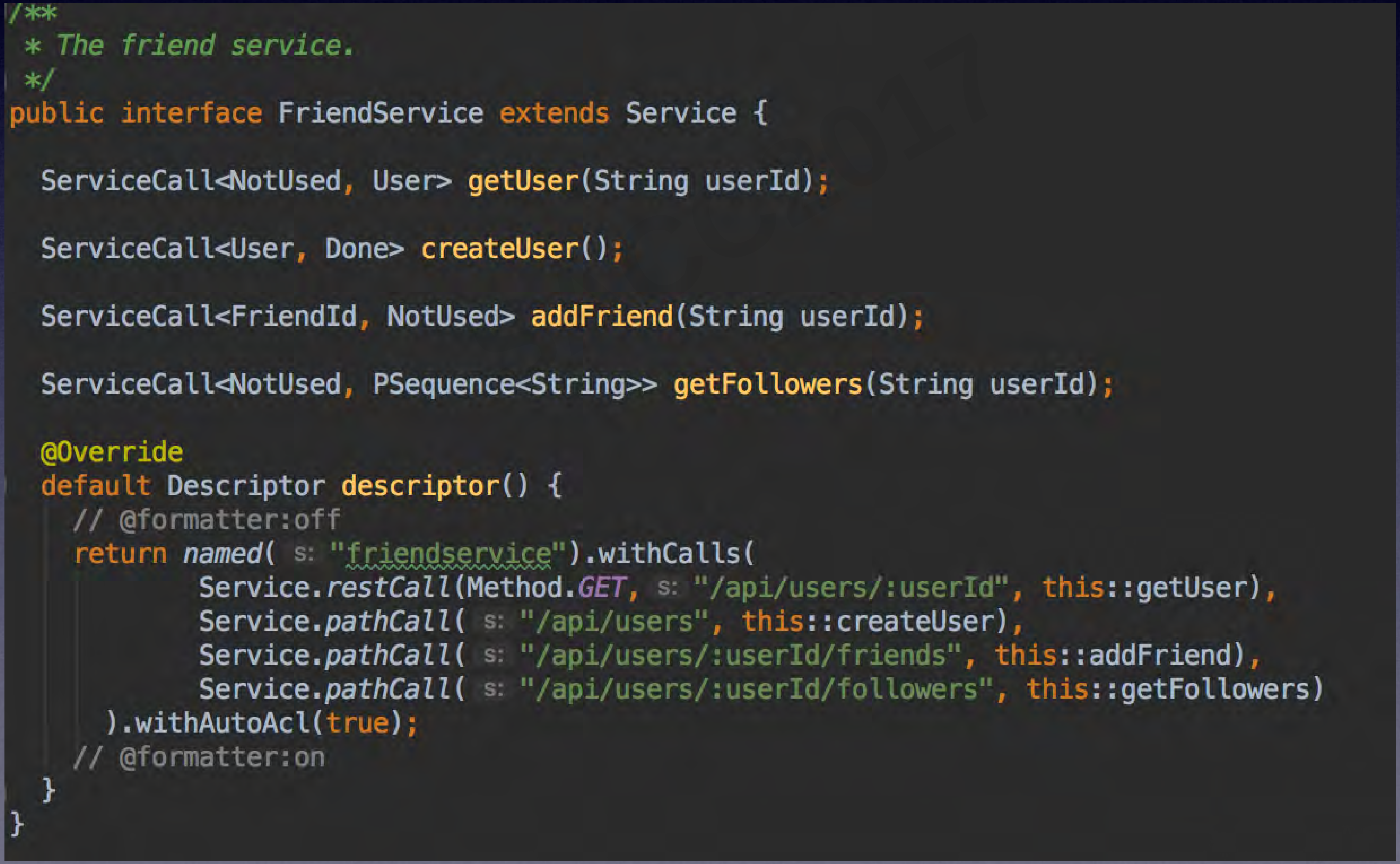
Actor Model



Let’s Demo

Service API

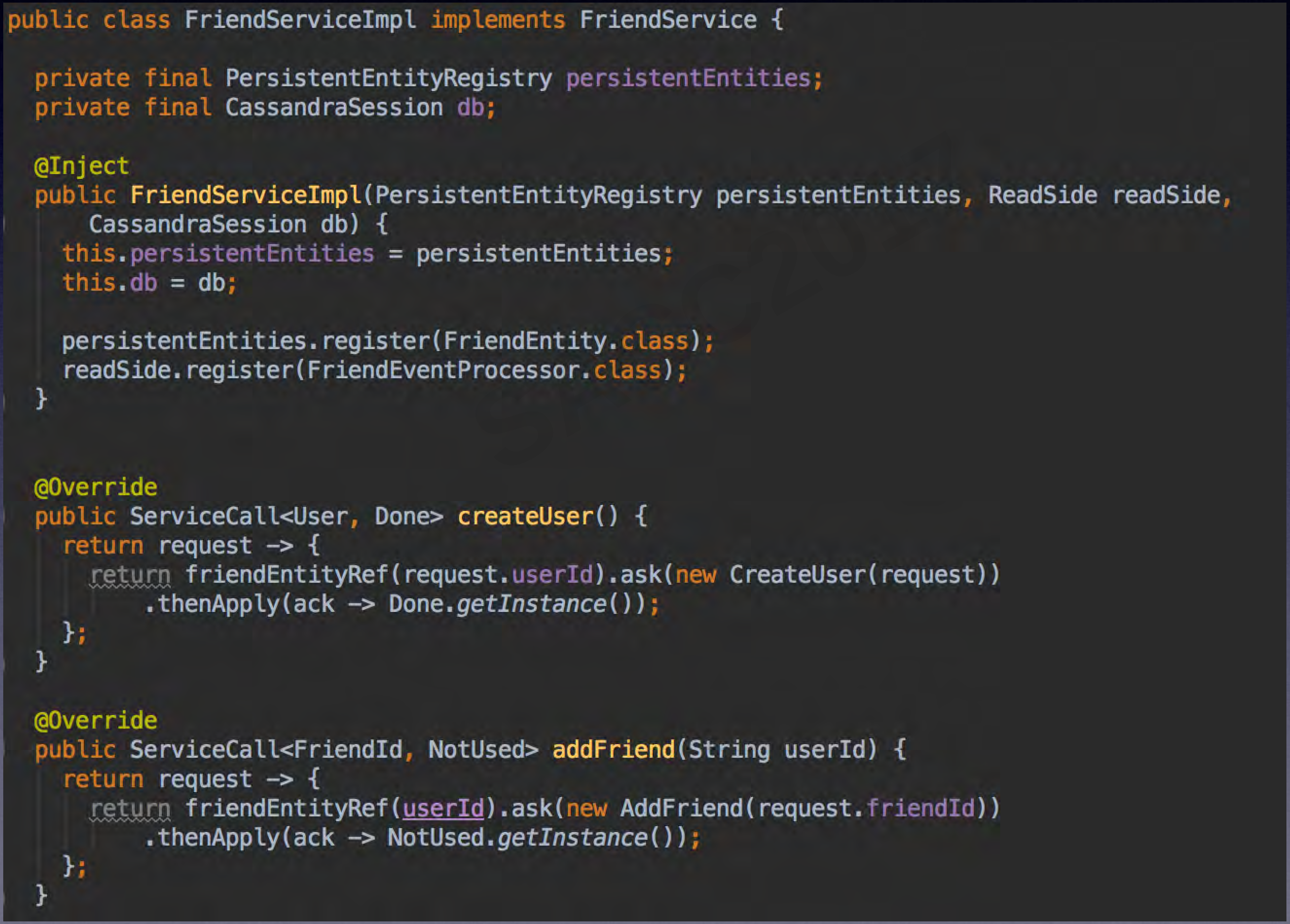
Declaration of Service Descriptors



```
/**
 * The friend service.
 */
public interface FriendService extends Service {

  ServiceCall<NotUsed, User> getUser(String userId);

  ServiceCall<User, Done> createUser();

  ServiceCall<FriendId, NotUsed> addFriend(String userId);

  ServiceCall<NotUsed, PSequence<String>> getFollowers(String userId);

  @Override
  default Descriptor descriptor() {
    // @formatter:off
    return named( s: "friendservice").withCalls(
            Service.restCall(Method.GET, s: "/api/users/:userId", this::getUser),
            Service.pathCall( s: "/api/users", this::createUser),
            Service.pathCall( s: "/api/users/:userId/friends", this::addFriend),
            Service.pathCall( s: "/api/users/:userId/followers", this::getFollowers)
      ).withAutoAcl(true);
    // @formatter:on
  }
}
```

Service API

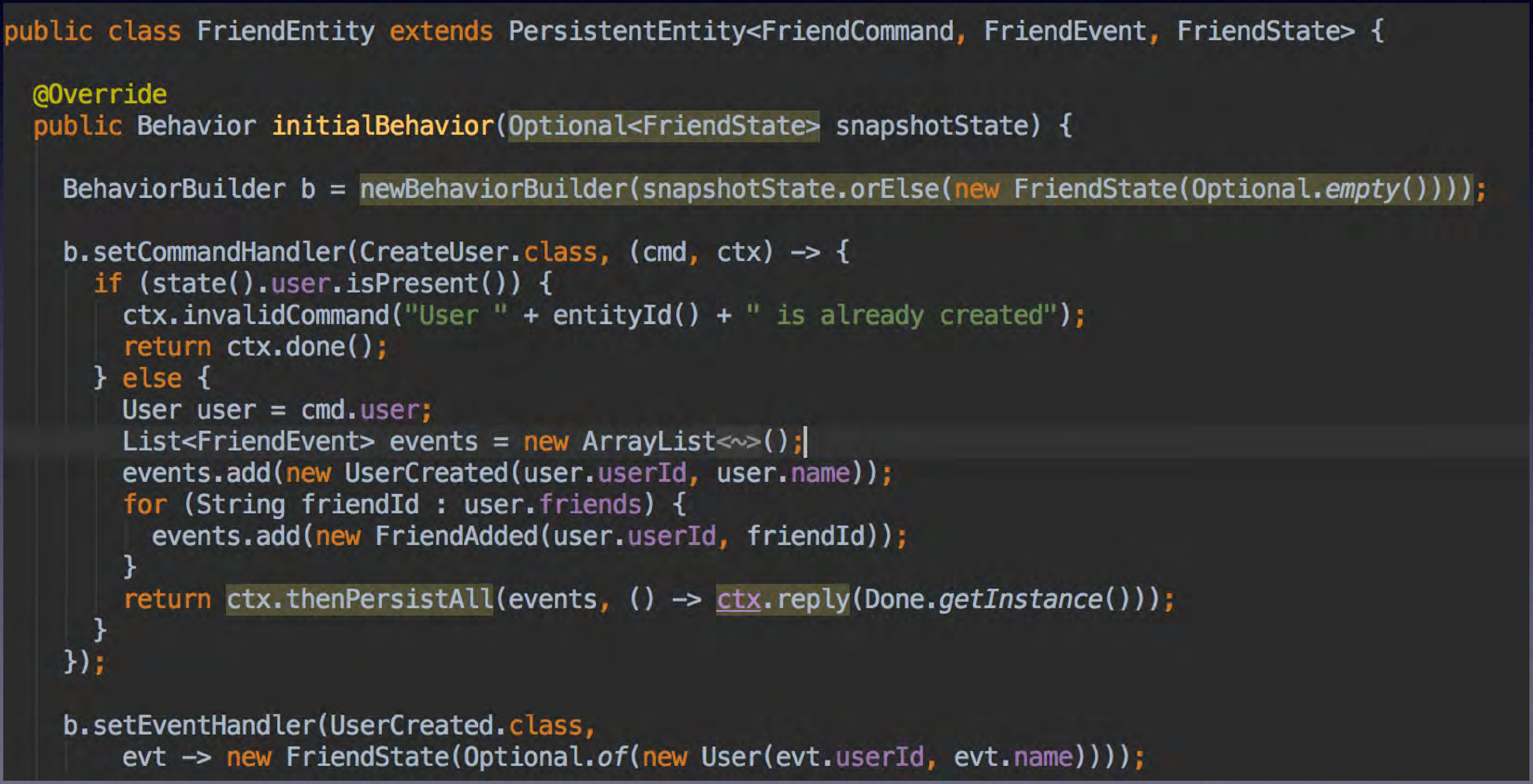
Implementation of HelloService



```java
public class FriendServiceImpl implements FriendService {

  private final PersistentEntityRegistry persistentEntities;
  private final CassandraSession db;

  @Inject
  public FriendServiceImpl(PersistentEntityRegistry persistentEntities, ReadSide readSide,
      CassandraSession db) {
    this.persistentEntities = persistentEntities;
    this.db = db;

    persistentEntities.register(FriendEntity.class);
    readSide.register(FriendEventProcessor.class);
  }


  @Override
  public ServiceCall<User, Done> createUser() {
    return request -> {
      return friendEntityRef(request.userId).ask(new CreateUser(request))
          .thenApply(ack -> Done.getInstance());
    };
  }

  @Override
  public ServiceCall<FriendId, NotUsed> addFriend(String userId) {
    return request -> {
      return friendEntityRef(userId).ask(new AddFriend(request.friendId))
          .thenApply(ack -> NotUsed.getInstance());
    };
  }
```

Persistence API

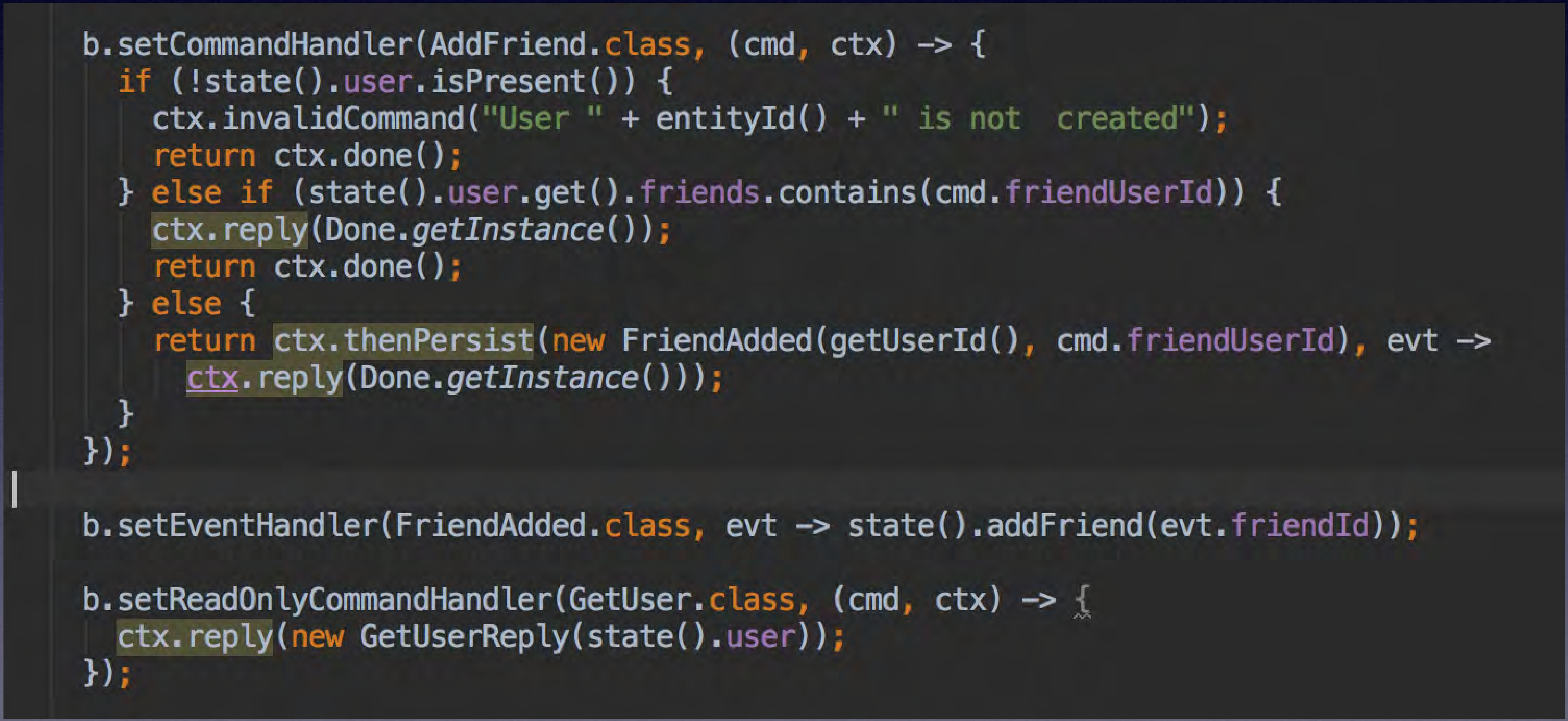
CreateUser Command handler



```java
public class FriendEntity extends PersistentEntity<FriendCommand, FriendEvent, FriendState> {

  @Override
  public Behavior initialBehavior(Optional<FriendState> snapshotState) {

    BehaviorBuilder b = newBehaviorBuilder(snapshotState.orElse(new FriendState(Optional.empty())));

    b.setCommandHandler(CreateUser.class, (cmd, ctx) -> {
      if (state().user.isPresent()) {
        ctx.invalidCommand("User " + entityId() + " is already created");
        return ctx.done();
      } else {
        User user = cmd.user;
        List<FriendEvent> events = new ArrayList<~>();
        events.add(new UserCreated(user.userId, user.name));
        for (String friendId : user.friends) {
          events.add(new FriendAdded(user.userId, friendId));
        }
        return ctx.thenPersistAll(events, () -> ctx.reply(Done.getInstance()));
      }
    });

    b.setEventHandler(UserCreated.class,
        evt -> new FriendState(Optional.of(new User(evt.userId, evt.name))));
```

Persistence API

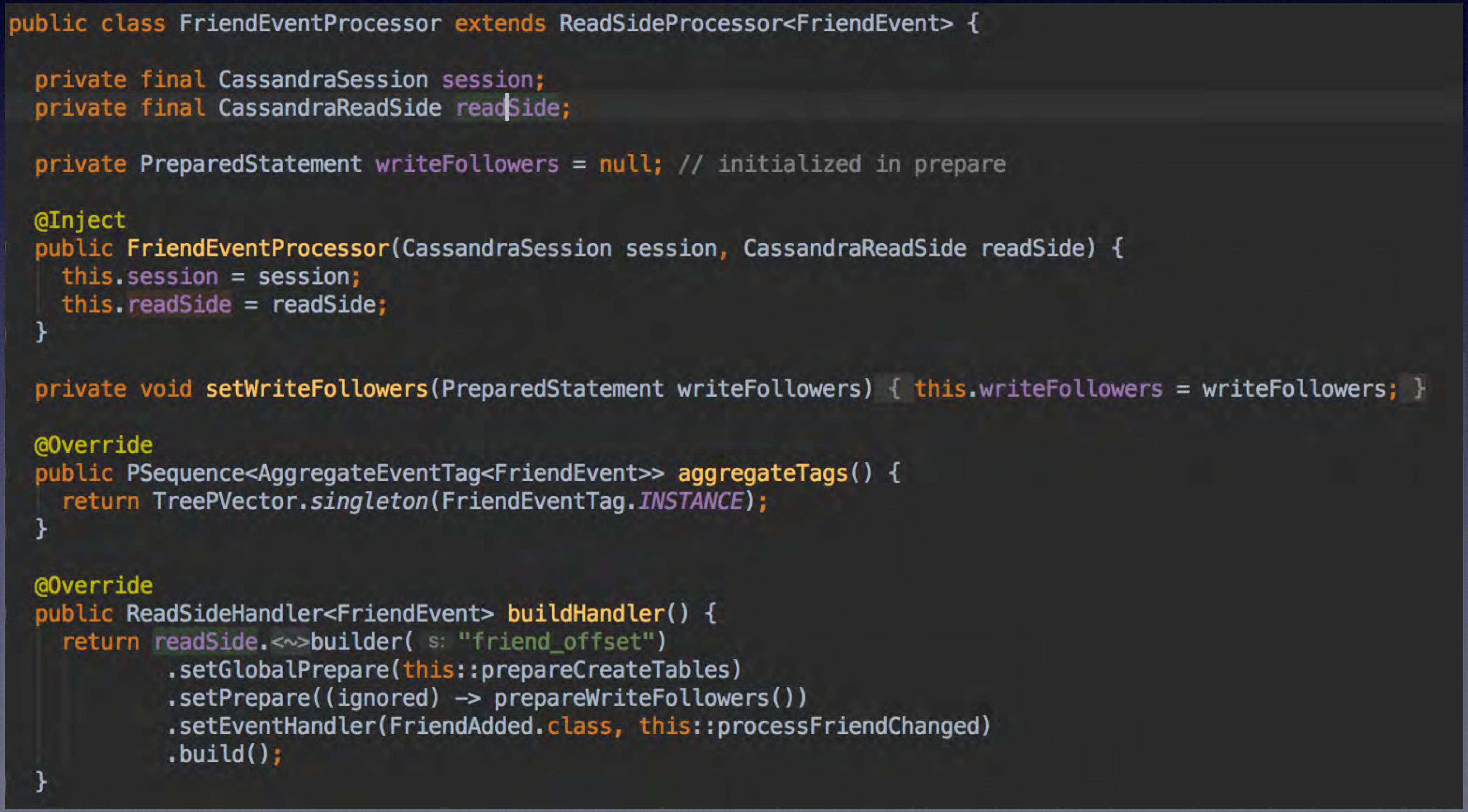
AddFriend Command and GetUser Command Handler



```
b.setCommandHandler(AddFriend.class, (cmd, ctx) -> {
  if (!state().user.isPresent()) {
    ctx.invalidCommand("User " + entityId() + " is not  created");
    return ctx.done();
  } else if (state().user.get().friends.contains(cmd.friendUserId)) {
    ctx.reply(Done.getInstance());
    return ctx.done();
  } else {
    return ctx.thenPersist(new FriendAdded(getUserId(), cmd.friendUserId), evt ->
      ctx.reply(Done.getInstance()));
  }
});

b.setEventHandler(FriendAdded.class, evt -> state().addFriend(evt.friendId));

b.setReadOnlyCommandHandler(GetUser.class, (cmd, ctx) -> {
  ctx.reply(new GetUserReply(state().user));
});
```

Persistence API

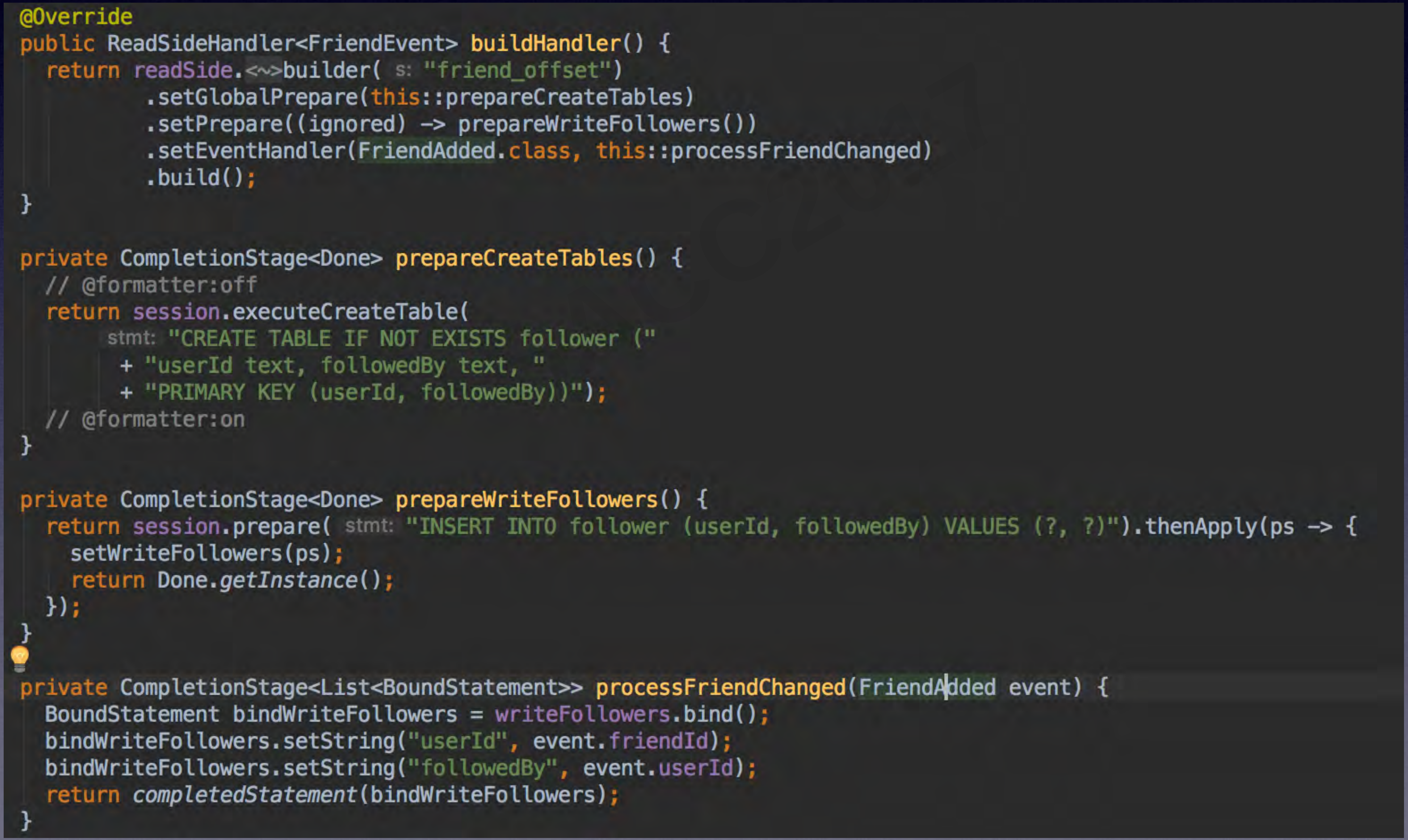
CQRS Read Side Support Definition



```java
public class FriendEventProcessor extends ReadSideProcessor<FriendEvent> {

  private final CassandraSession session;
  private final CassandraReadSide readSide;

  private PreparedStatement writeFollowers = null; // initialized in prepare

  @Inject
  public FriendEventProcessor(CassandraSession session, CassandraReadSide readSide) {
    this.session = session;
    this.readSide = readSide;
  }

  private void setWriteFollowers(PreparedStatement writeFollowers) { this.writeFollowers = writeFollowers; }

  @Override
  public PSequence<AggregateEventTag<FriendEvent>> aggregateTags() {
    return TreePVector.singleton(FriendEventTag.INSTANCE);
  }

  @Override
  public ReadSideHandler<FriendEvent> buildHandler() {
    return readSide.<~>builder( s: "friend_offset")
            .setGlobalPrepare(this::prepareCreateTables)
            .setPrepare((ignored) -> prepareWriteFollowers())
            .setEventHandler(FriendAdded.class, this::processFriendChanged)
            .build();
  }
```

Persistence API

View update based on FriendAdded event



```java
@Override
public ReadSideHandler<FriendEvent> buildHandler() {
  return readSide.<~>builder( s: "friend_offset")
          .setGlobalPrepare(this::prepareCreateTables)
          .setPrepare((ignored) -> prepareWriteFollowers())
          .setEventHandler(FriendAdded.class, this::processFriendChanged)
          .build();
}

private CompletionStage<Done> prepareCreateTables() {
  // @formatter:off
  return session.executeCreateTable(
      stmt: "CREATE TABLE IF NOT EXISTS follower ("
        + "userId text, followedBy text, "
        + "PRIMARY KEY (userId, followedBy))");
  // @formatter:on
}

private CompletionStage<Done> prepareWriteFollowers() {
  return session.prepare( stmt: "INSERT INTO follower (userId, followedBy) VALUES (?, ?)").thenApply(ps -> {
    setWriteFollowers(ps);
    return Done.getInstance();
  });
}

private CompletionStage<List<BoundStatement>> processFriendChanged(FriendAdded event) {
  BoundStatement bindWriteFollowers = writeFollowers.bind();
  bindWriteFollowers.setString("userId", event.friendId);
  bindWriteFollowers.setString("followedBy", event.userId);
  return completedStatement(bindWriteFollowers);
}
```

QUESTIONS?

轱辘轱辘转
Dublin, Ireland
Scan the QR code to add me on WeChat