



WIND HIM UP

Mit SAGA service-übergreifende Transaktionen
verwalten

MIT SAGA SERVICE-ÜBERGREIFENDE TRANSAKTIONEN VERWALTEN

ÜBER MICH

Thomas Müller

sidion GmbH

Senior Software-Developer

thomas.mueller@sidion.de



@zaroselectro





AGENDA

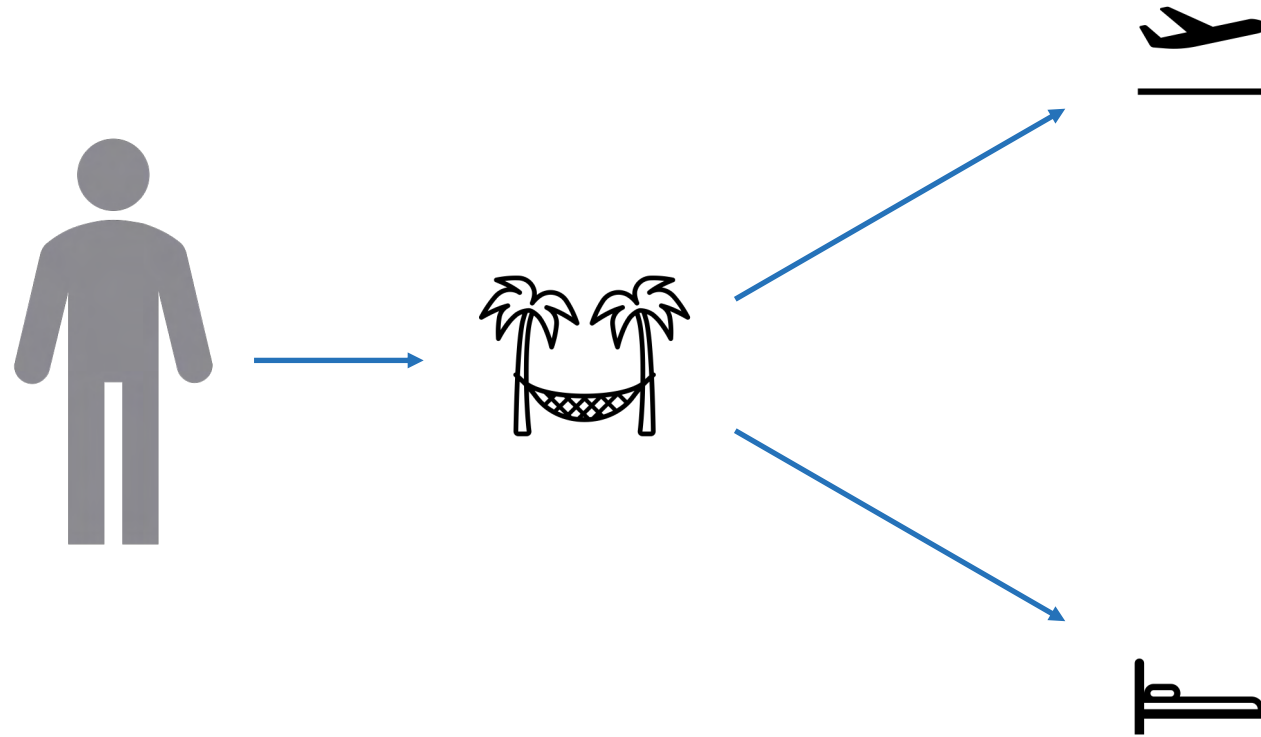
1. Szenario / Problemstellung
2. Distributet Transactions
3. Saga-Pattern
4. Choreography-based Saga
5. Orchestration-based Saga
6. Edge Cases
7. Coding



SZENARIO / PROBLEMSTELLUNG

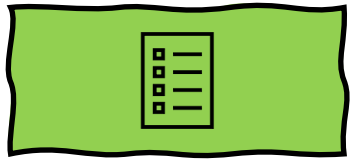
MIT SAGA SERVICE-ÜBERGREIFENDE TRANSAKTIONEN VERWALTEN

SZENARIO



MIT SAGA SERVICE-ÜBERGREIFENDE TRANSAKTIONEN VERWALTEN

SERVICES



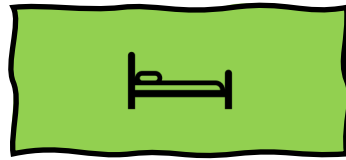
BookingService

- REST-API für Frontend
- Koordinator der Buchung



FlightBooking Service

- handelt Flugbuchung



HotelBooking Service

- handelt Hotelbuchung



PaymentService

- Validiert Kreditkarte
- handelt Zahlung

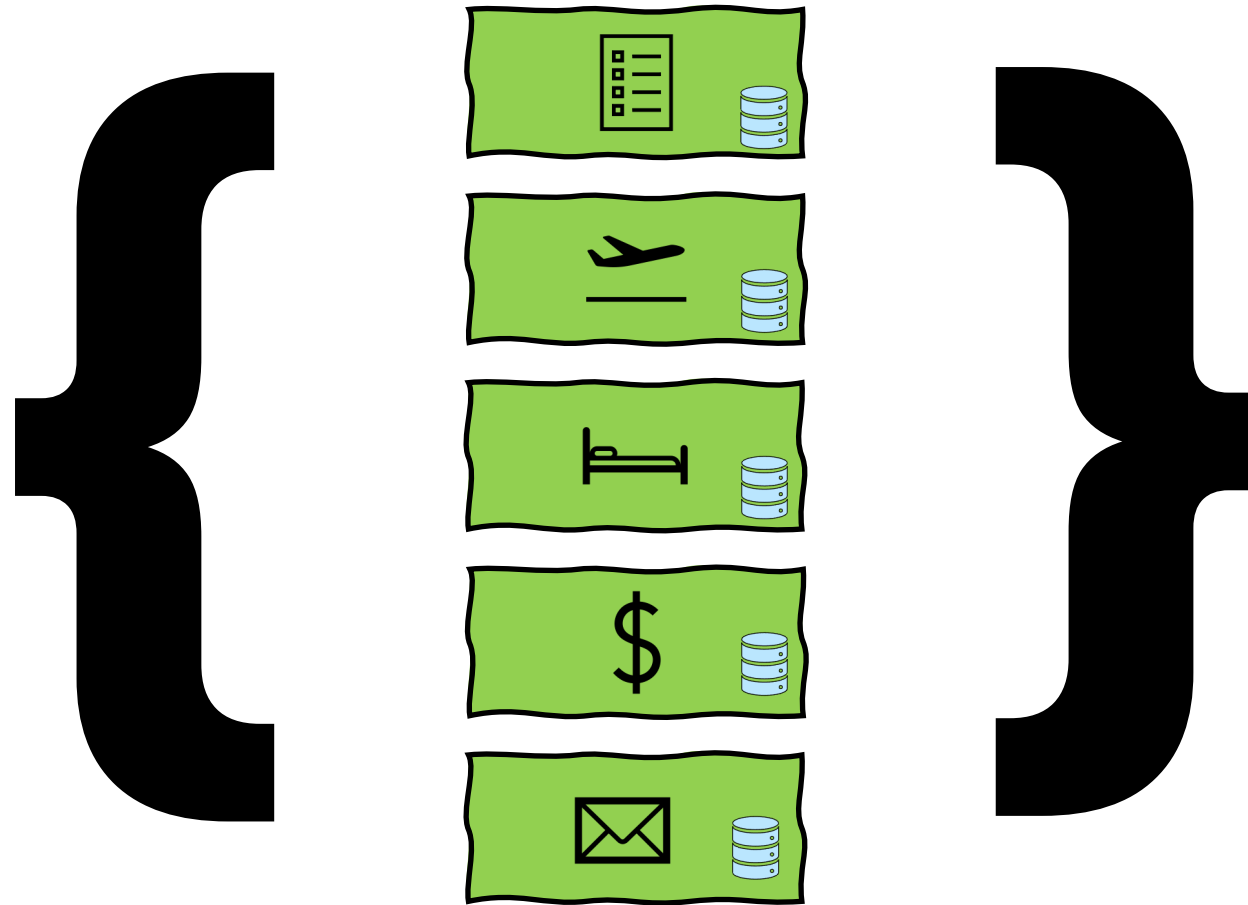


Confirmation Service

- Bestätigungs-Email

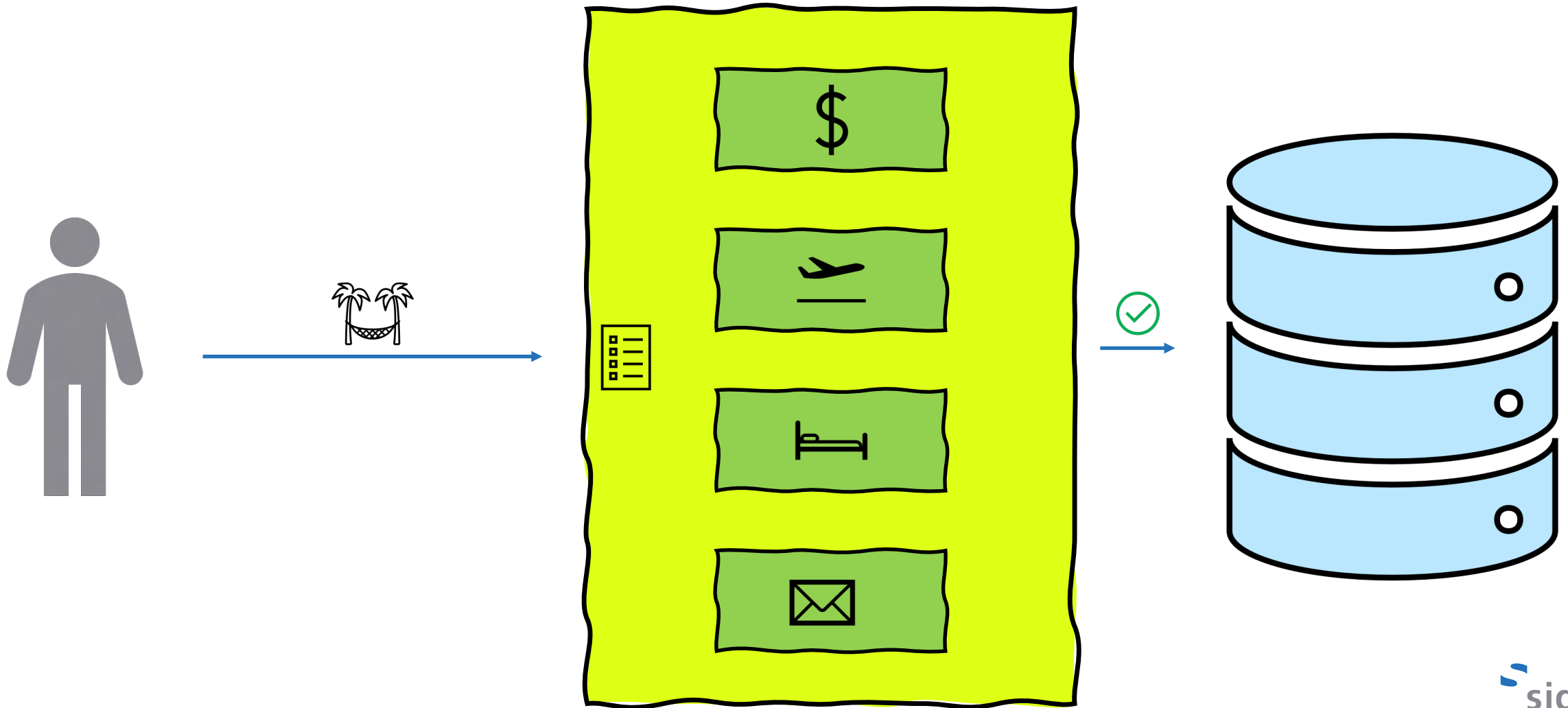
MIT SAGA SERVICE-ÜBERGREIFENDE TRANSAKTIONEN VERWALTEN

TRANSAKTION



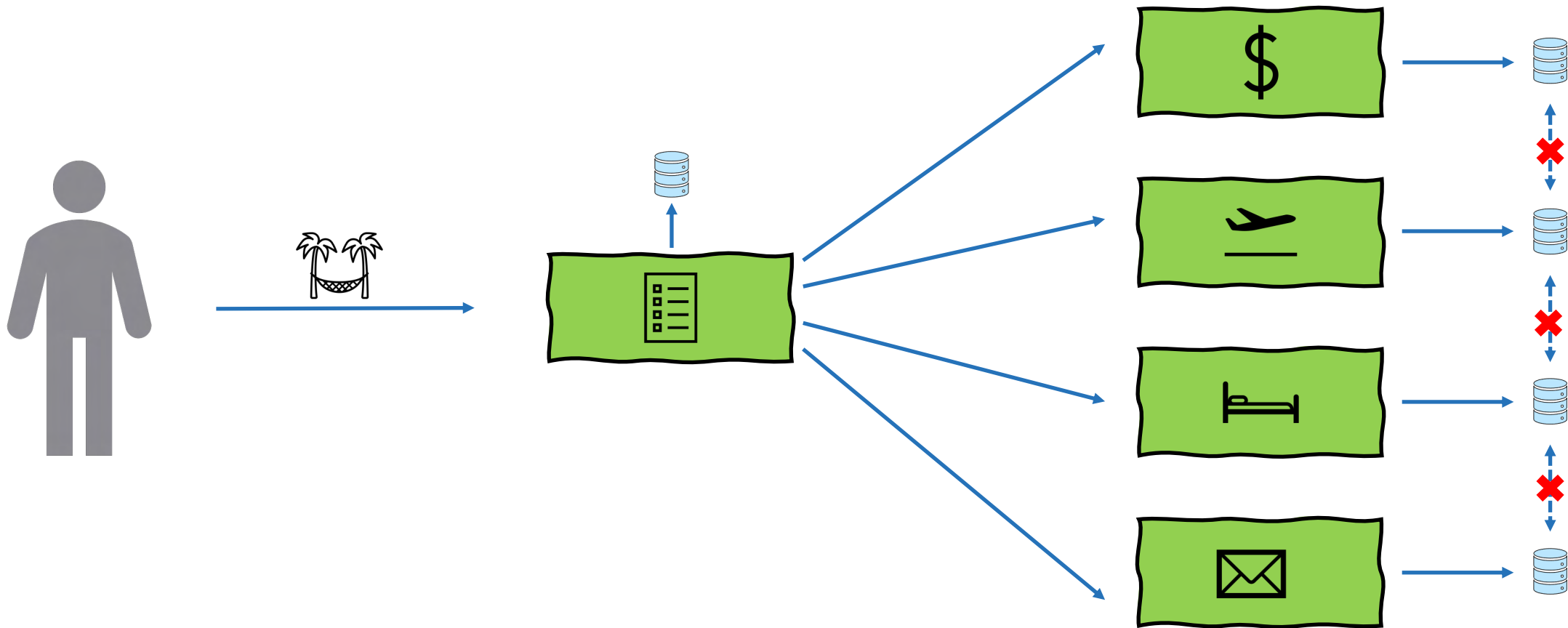
MIT SAGA SERVICE-ÜBERGREIFENDE TRANSAKTIONEN VERWALTEN

MONOLITH



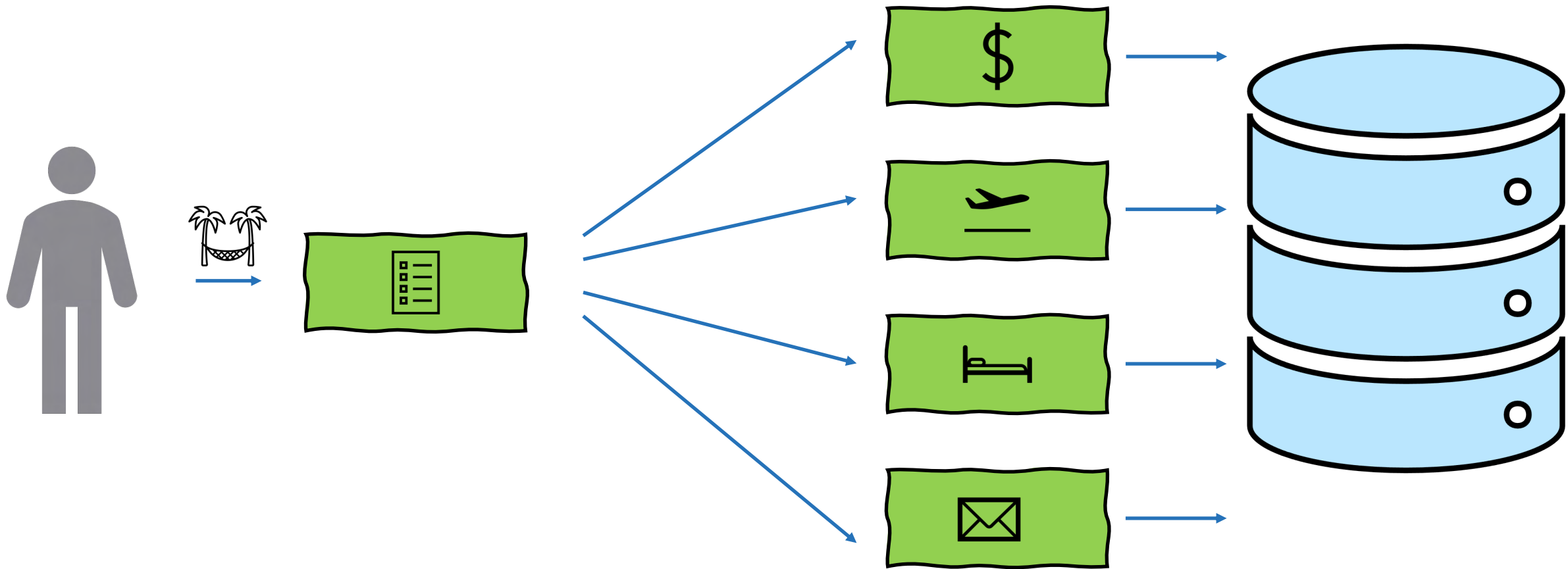
MIT SAGA SERVICE-ÜBERGREIFENDE TRANSAKTIONEN VERWALTEN

MICROSERVICE-ANSATZ



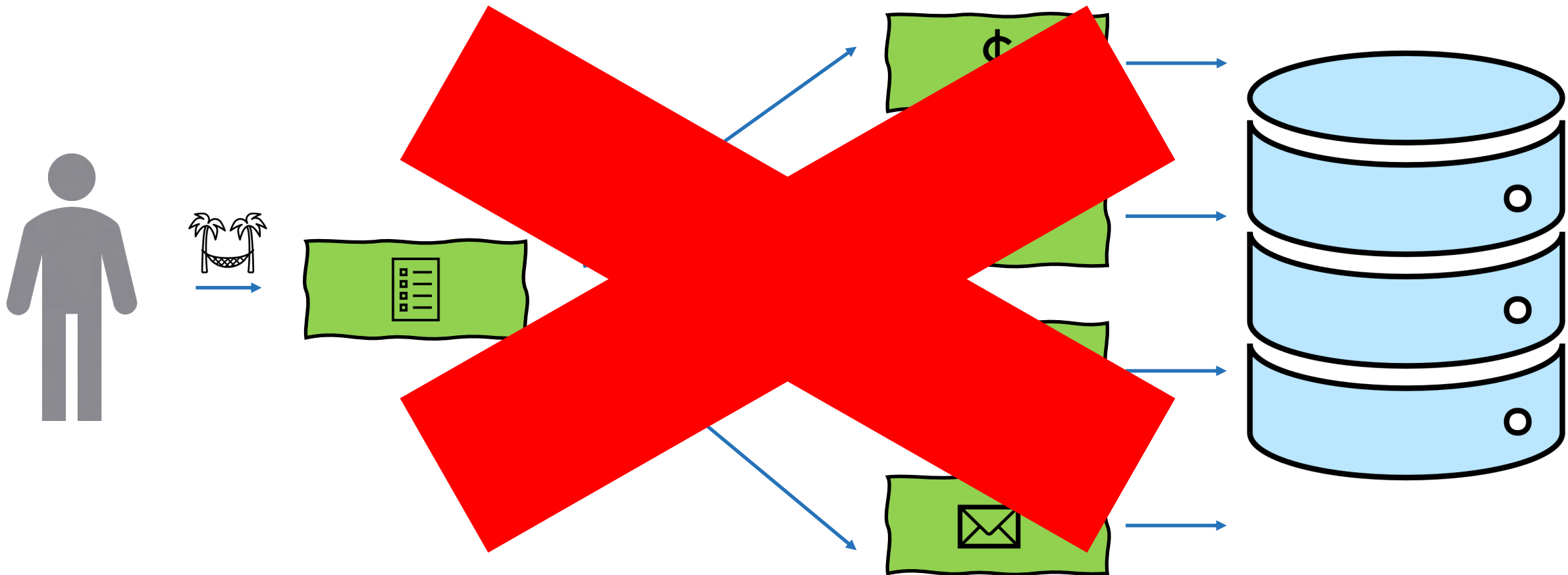
MIT SAGA SERVICE-ÜBERGREIFENDE TRANSAKTIONEN VERWALTEN

MICROSERVICE-ANSATZ



MIT SAGA SERVICE-ÜBERGREIFENDE TRANSAKTIONEN VERWALTEN

MICROSERVICE-ANSATZ



A - C - I - D

A tomicity

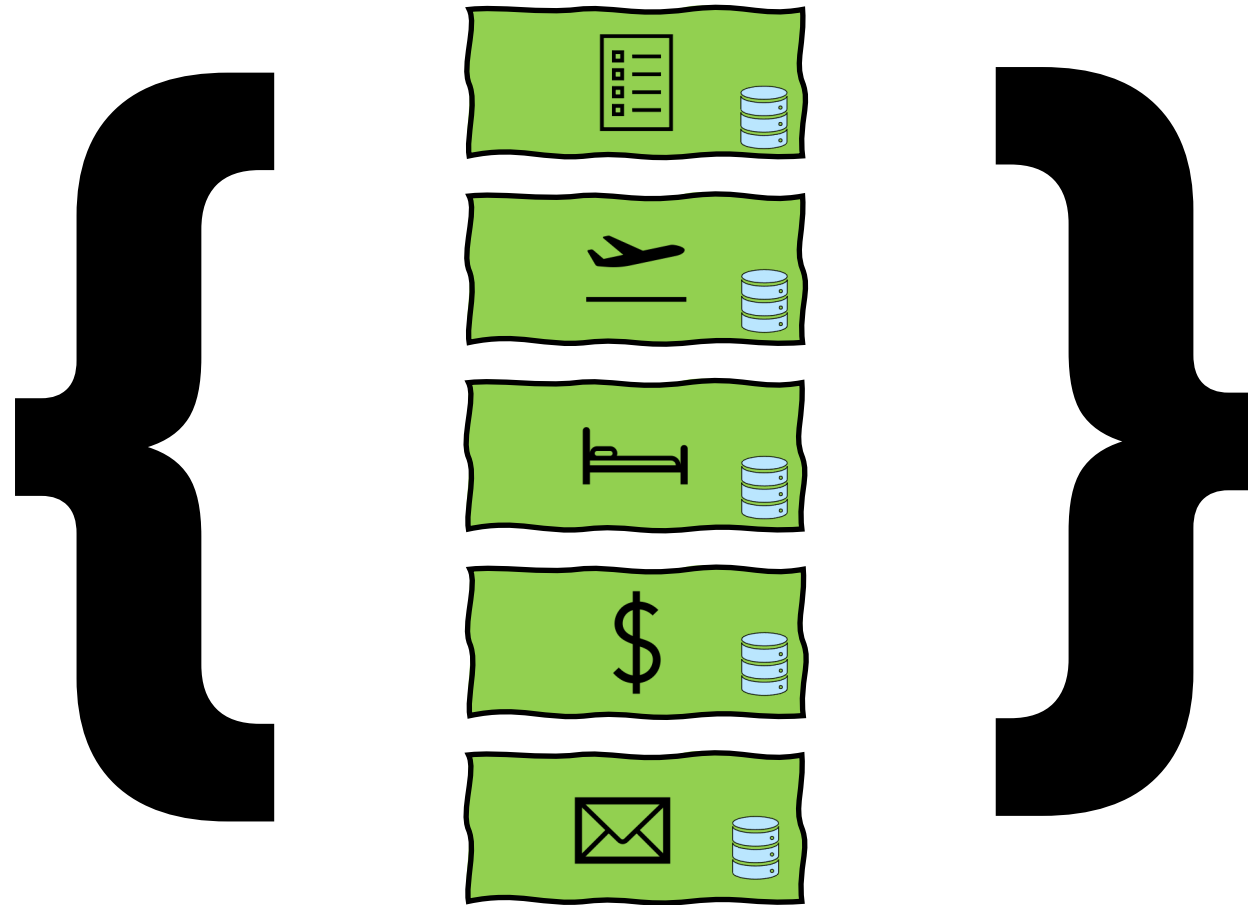
C onsistency

I solation

D urability

MIT SAGA SERVICE-ÜBERGREIFENDE TRANSAKTIONEN VERWALTEN

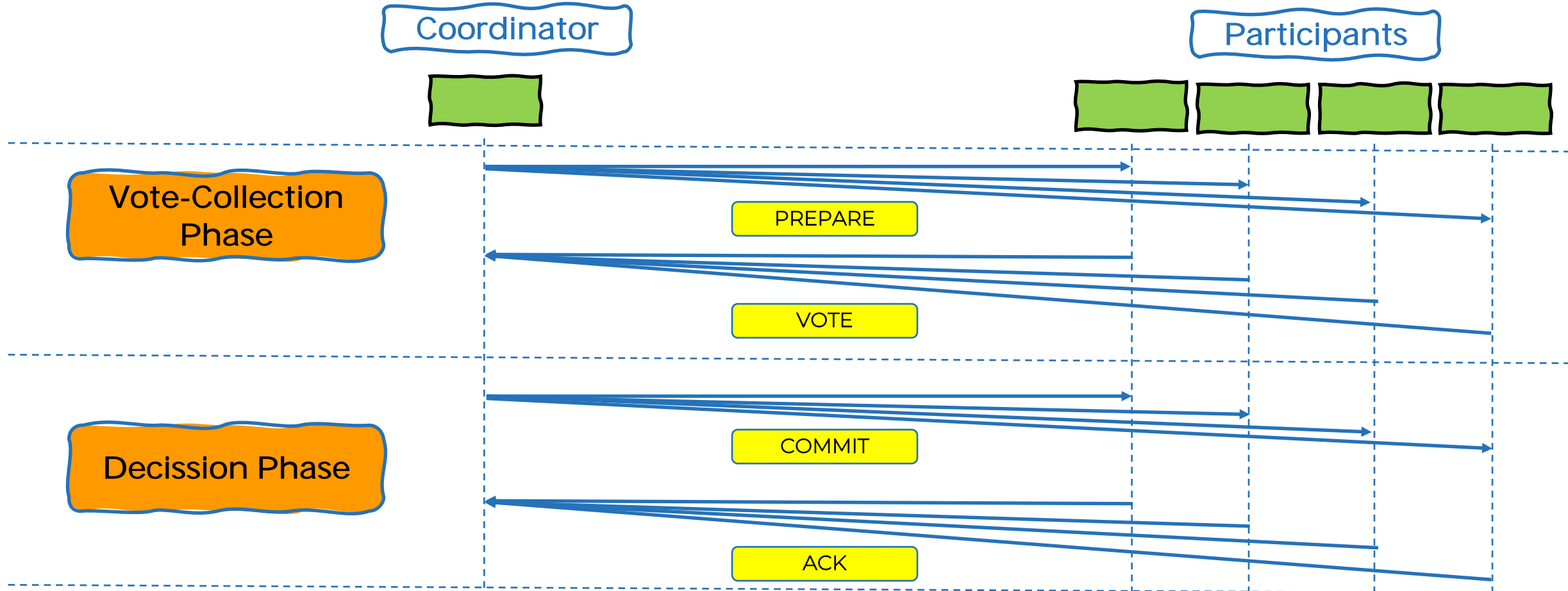
TRANSAKTION





DISTRIBUTED TRANSACTIONS

2-PHASE-COMMIT (2PC)



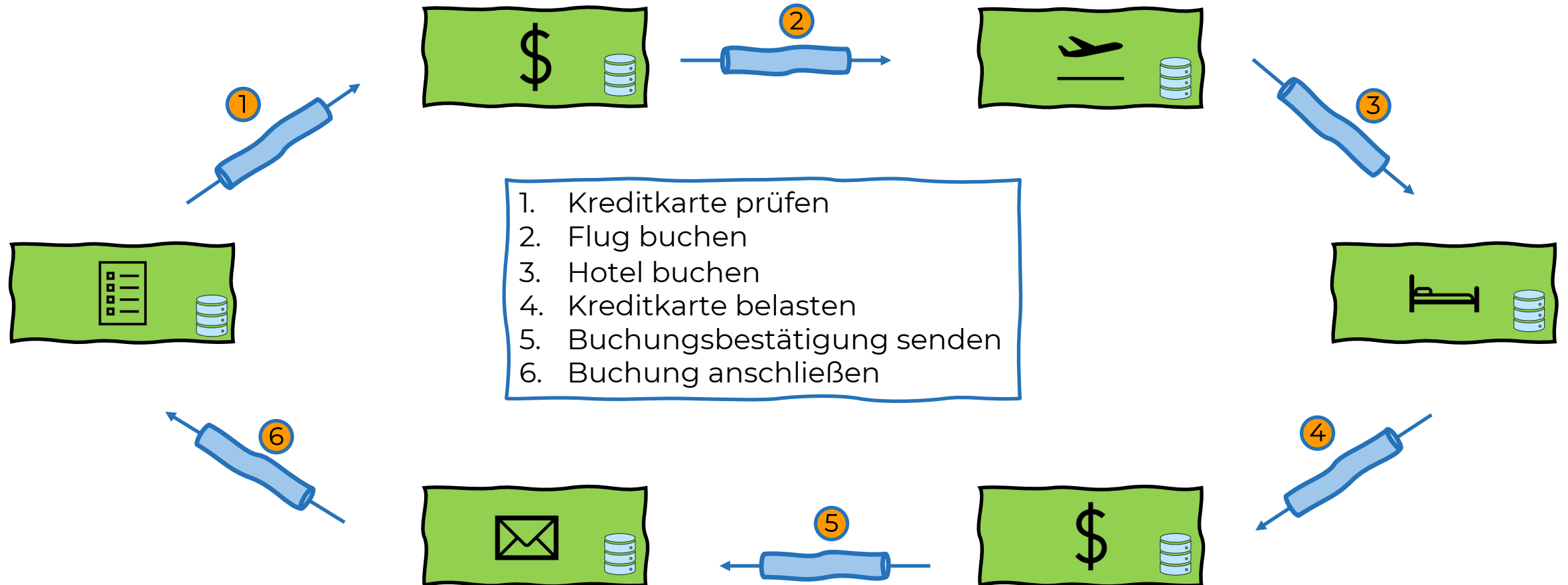


SAGA-PATTERN

SAGA PATTERN

- ✔ SAGA is a sequence of local transactions that are coordinate using messaging
- ✔ Each local transaction updates data in a single service
- ✔ On failure due to the violation of a business rule, it must execute compensation transactions to explicitly undo changes

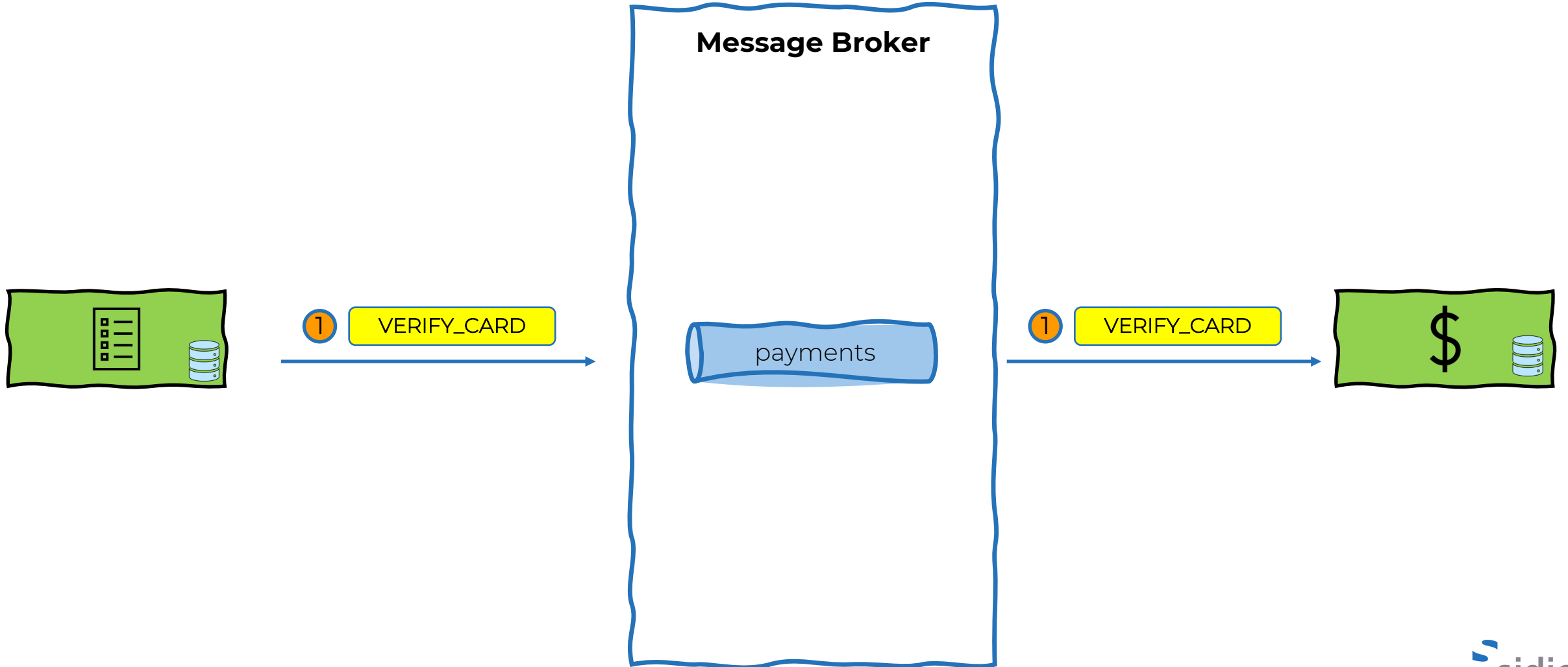
SAGA PATTERN



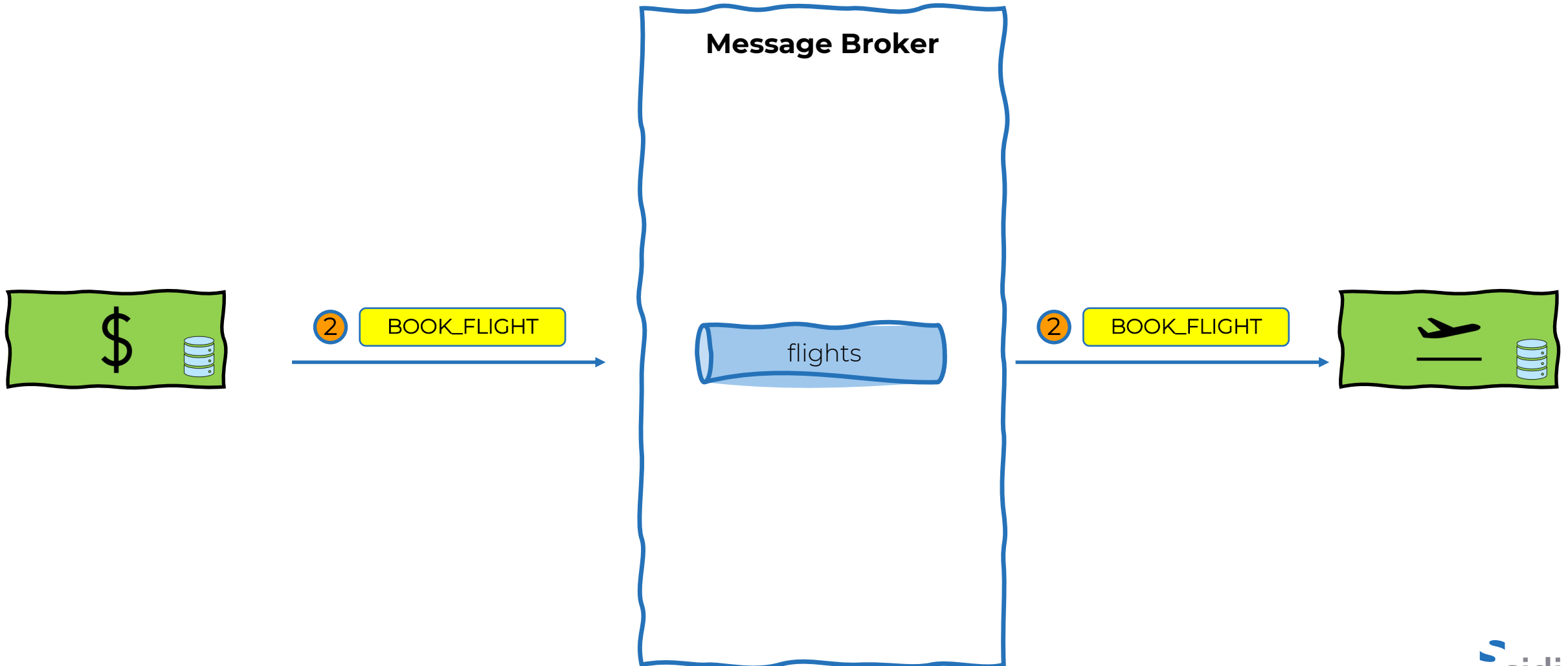


**CHOREOGRAPHY
BASED
SAGA**

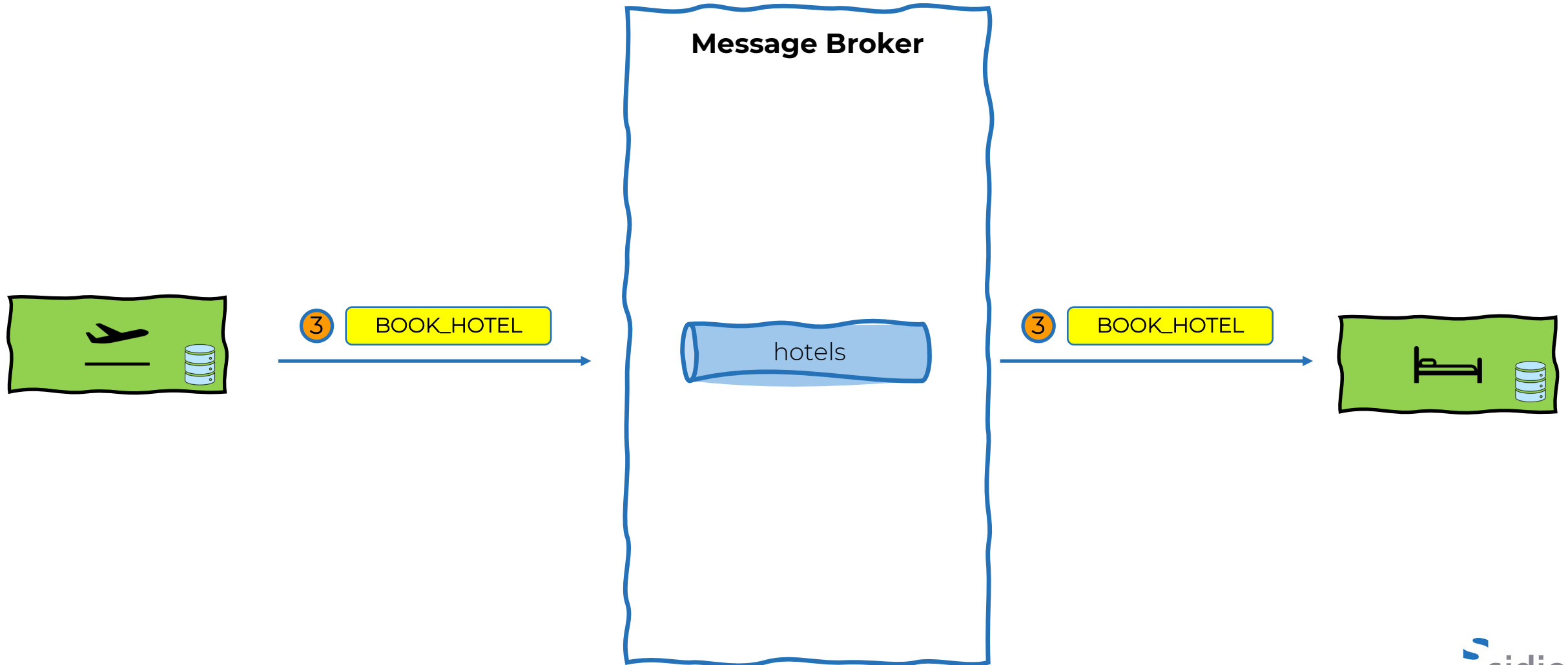
1. VERIFY_CARD



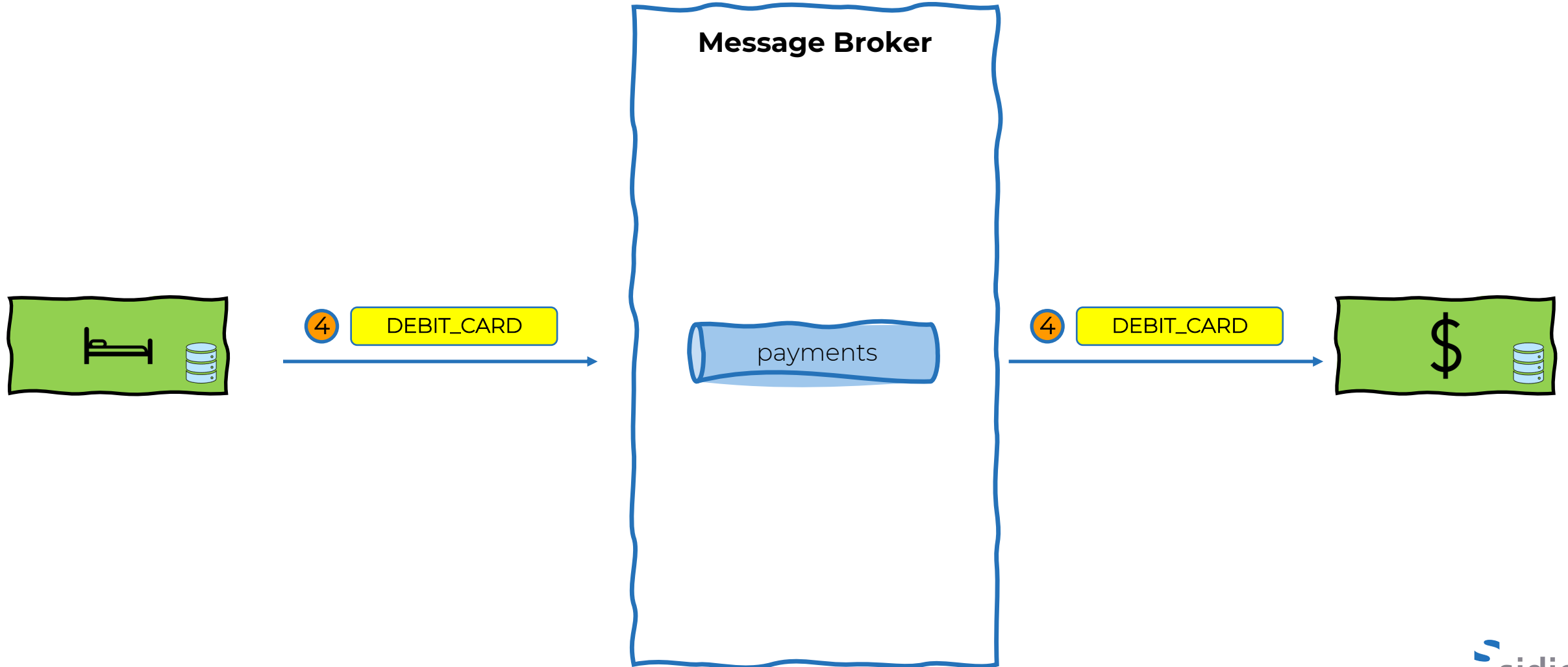
2. BOOK_FLIGHT



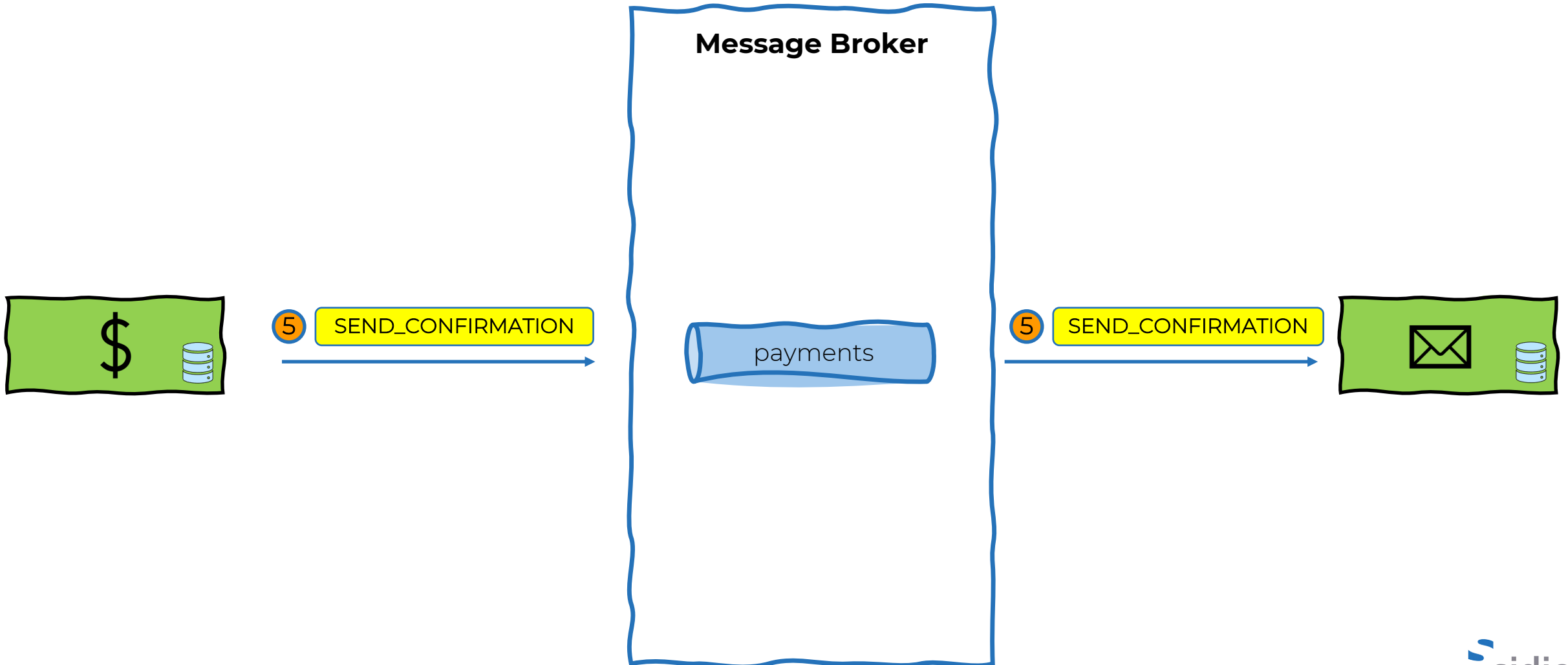
3. BOOK_HOTEL



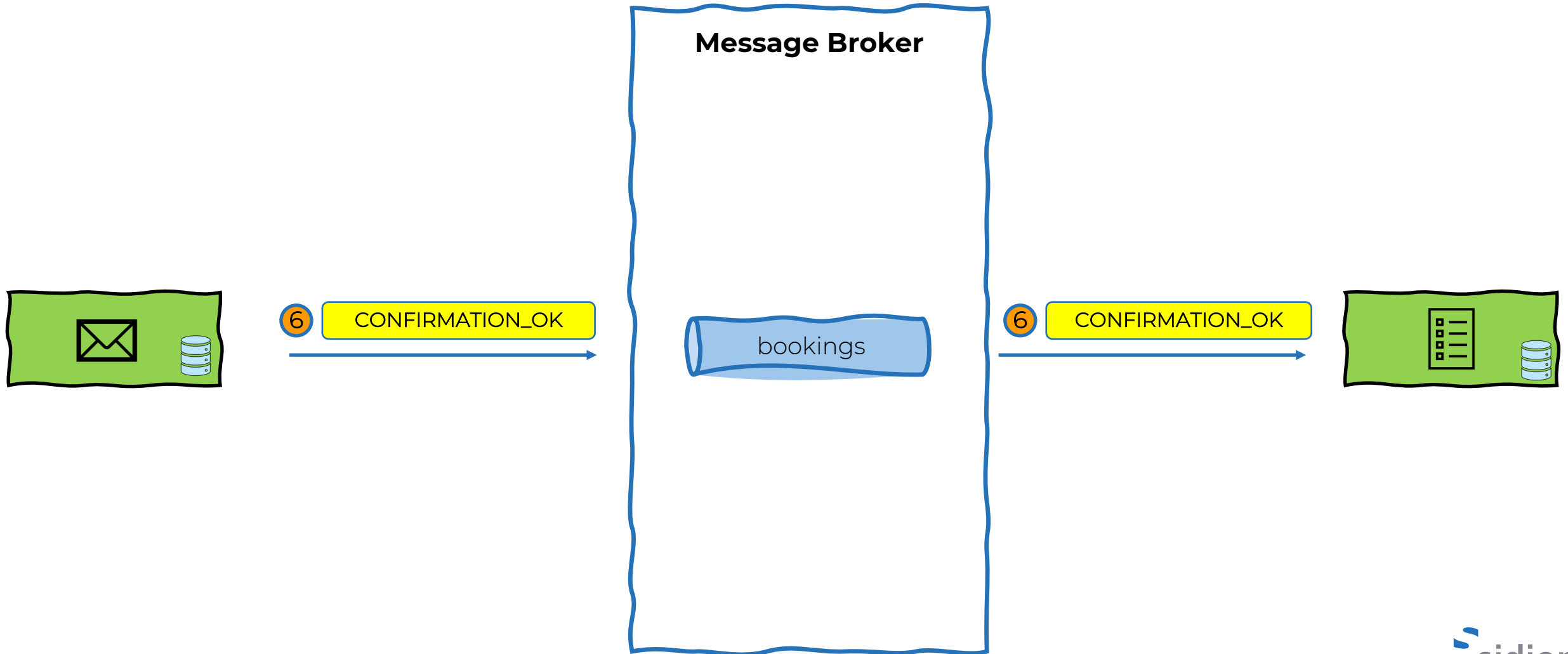
4. DEBIT_CARD



5. SEND_CONFIRMATION

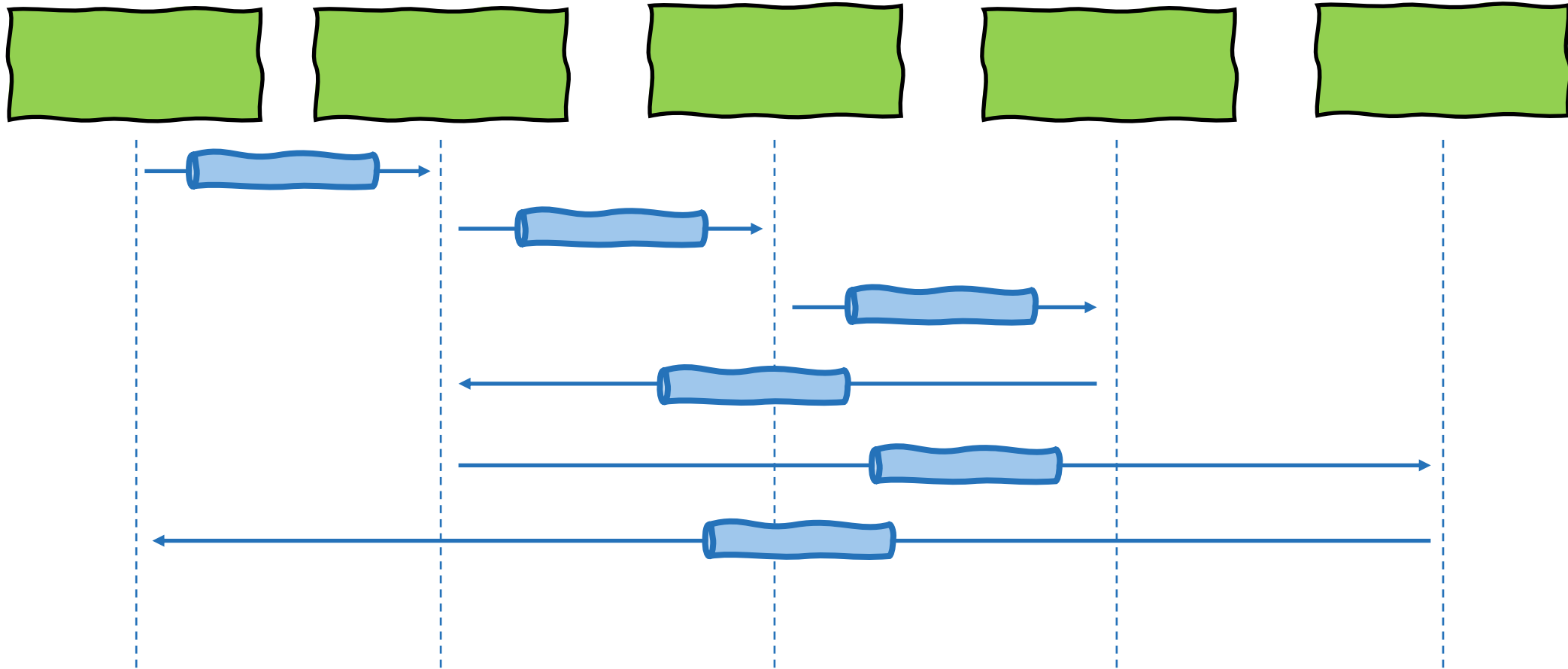


6. CONFIRMATION_OK



MIT SAGA SERVICE-ÜBERGREIFENDE TRANSAKTIONEN VERWALTEN

CHOREOGRAPHY-BASED SAGA

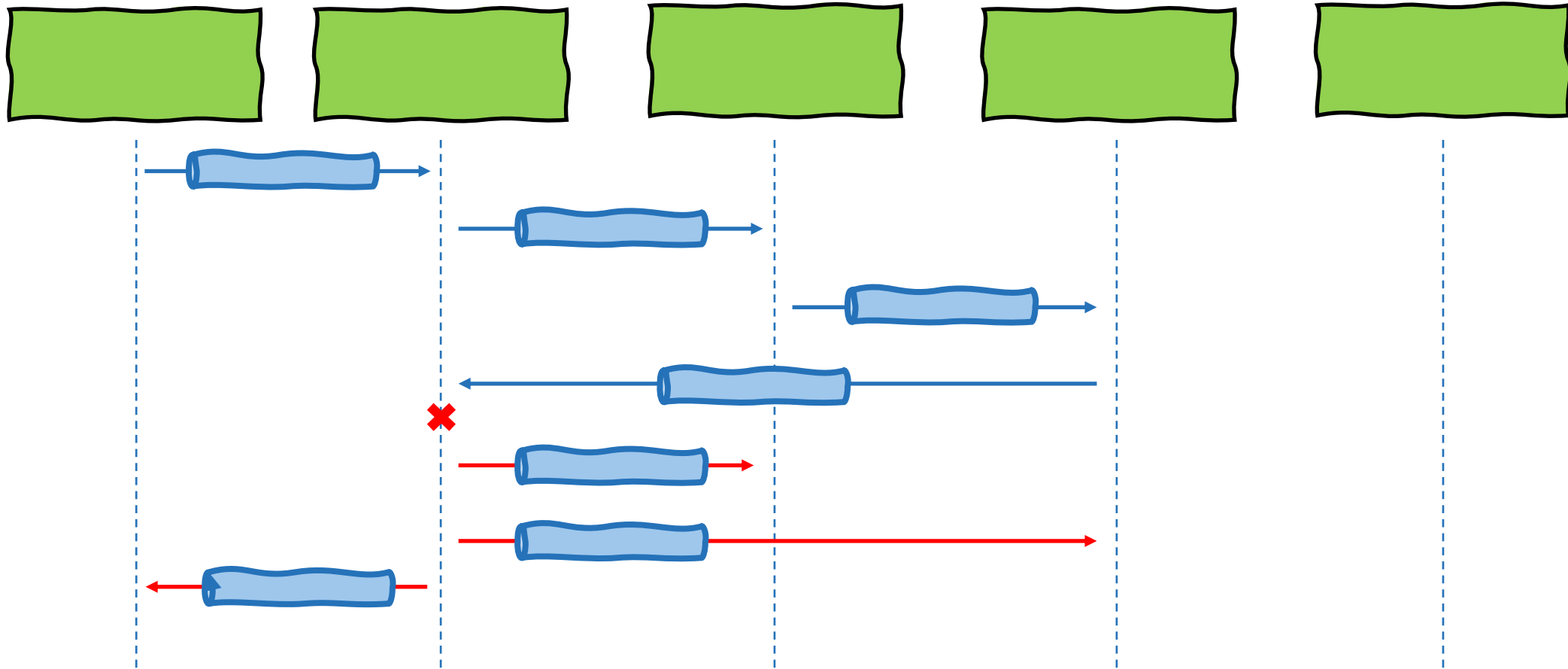


SAGA PATTERN

- ✔ SAGA is a sequence of local transactions that are coordinate using messaging
- ✔ Each local transaction updates data in a single service
- ✔ On failure due to the violation of a business rule, it must execute compensation transactions to explicitly undo changes

MIT SAGA SERVICE-ÜBERGREIFENDE TRANSAKTIONEN VERWALTEN

CHOREOGRAPHY-BASED SAGA



VORTEILE / NACHTEILE



Einfachheit



Loose Koppelung



Schwieriger zu verstehen



Zyklische Abhängigkeiten zwischen den Services



Risiko der zu engen Koppelung

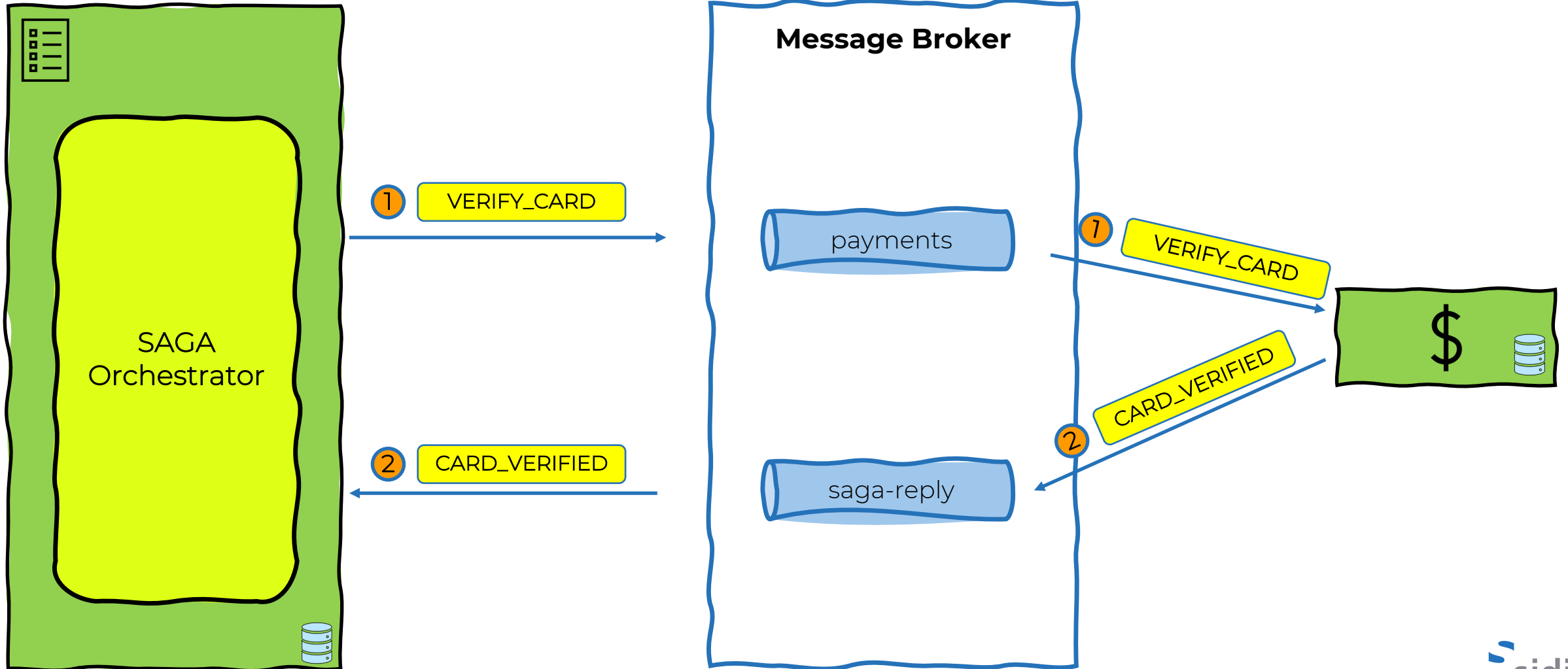


**ORCHESTRATION
BASED
SAGA**

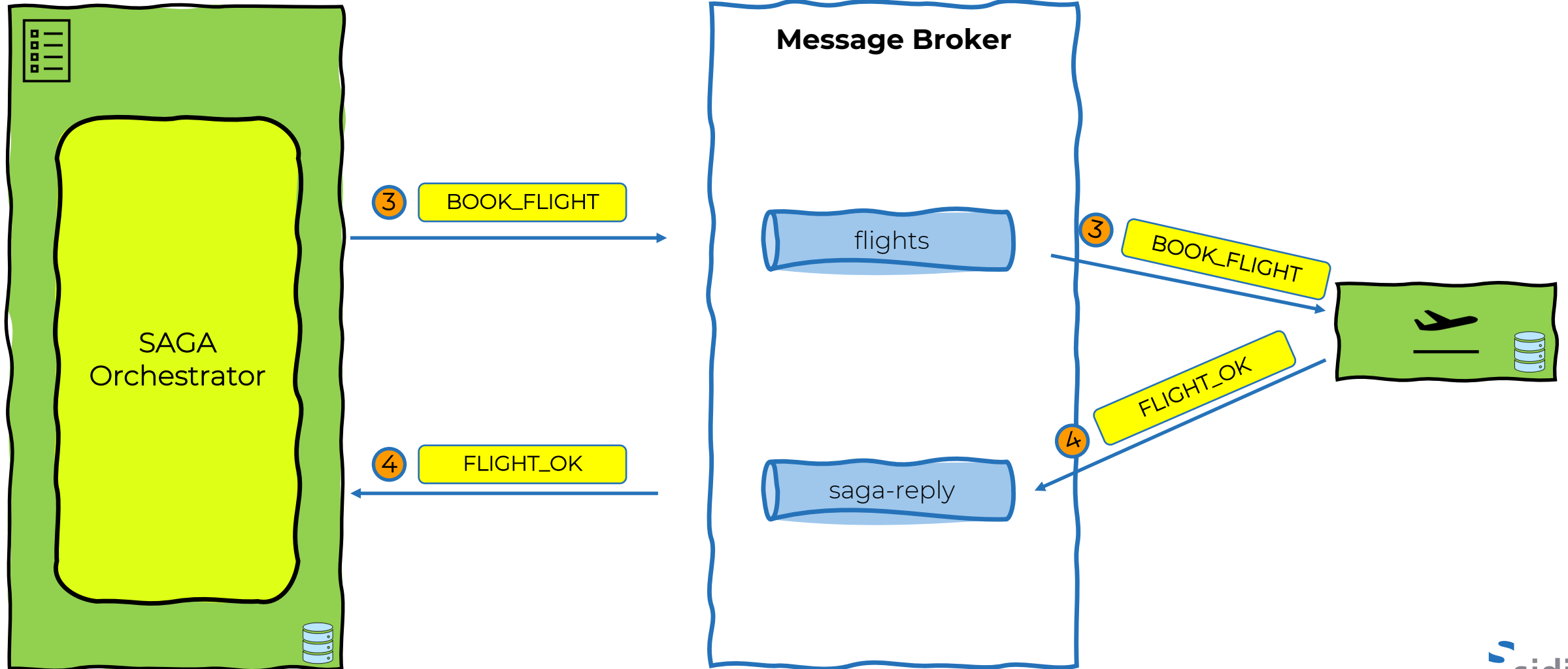
ORCHESTRATION-BASED SAGA

- ✓ eigenständige Orchestrator-Klasse mit der einzigen Verantwortung die Saga-Teilnehmer ihre Aufgaben zuzuweisen
- ✓ Command / Async-Reply-Style-Interaction

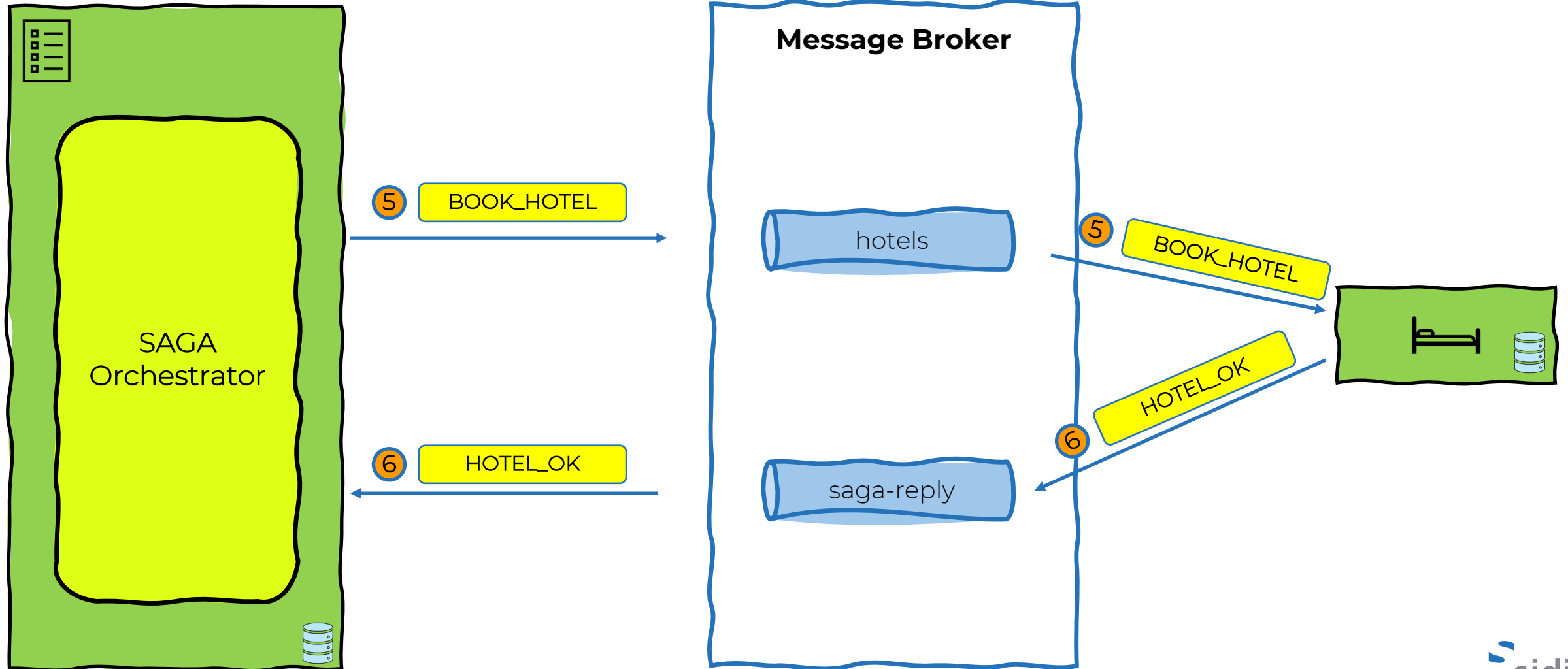
1. VERIFY_CARD



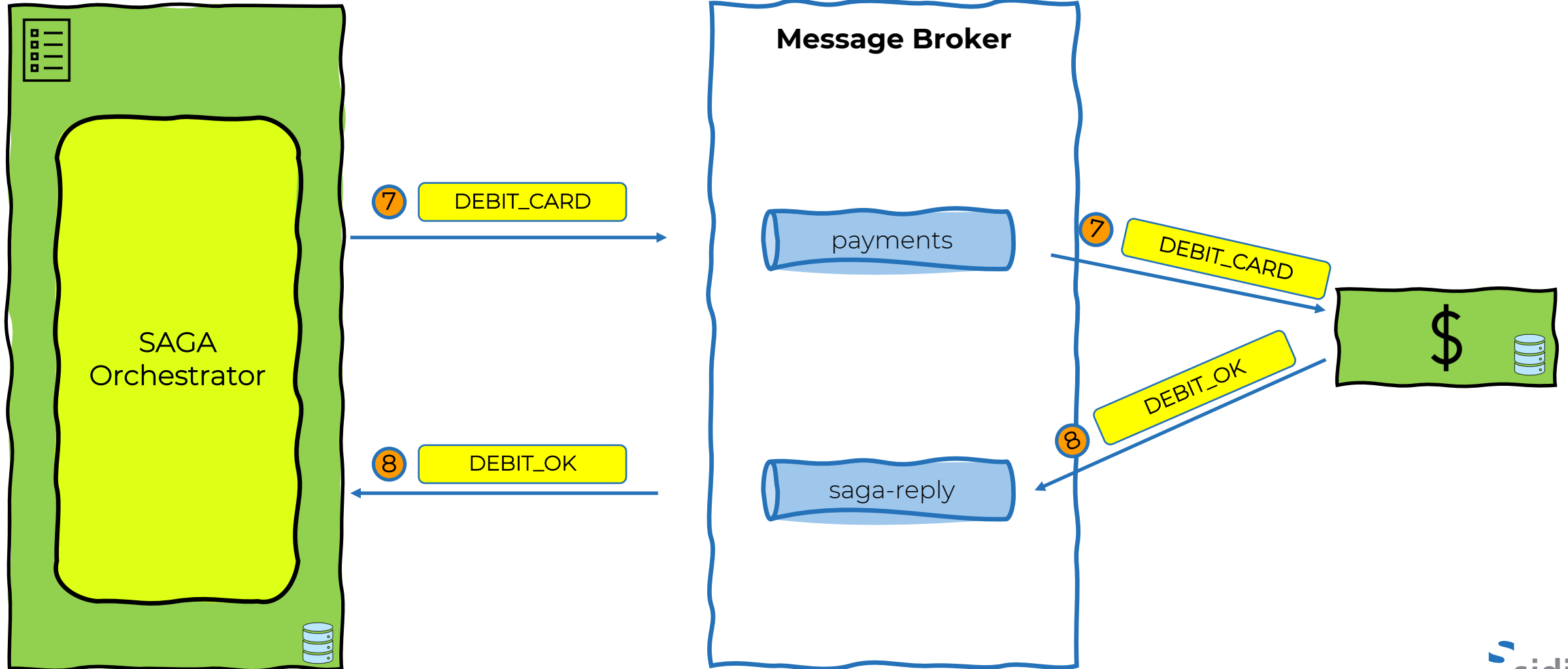
2. BOOK_FLIGHT



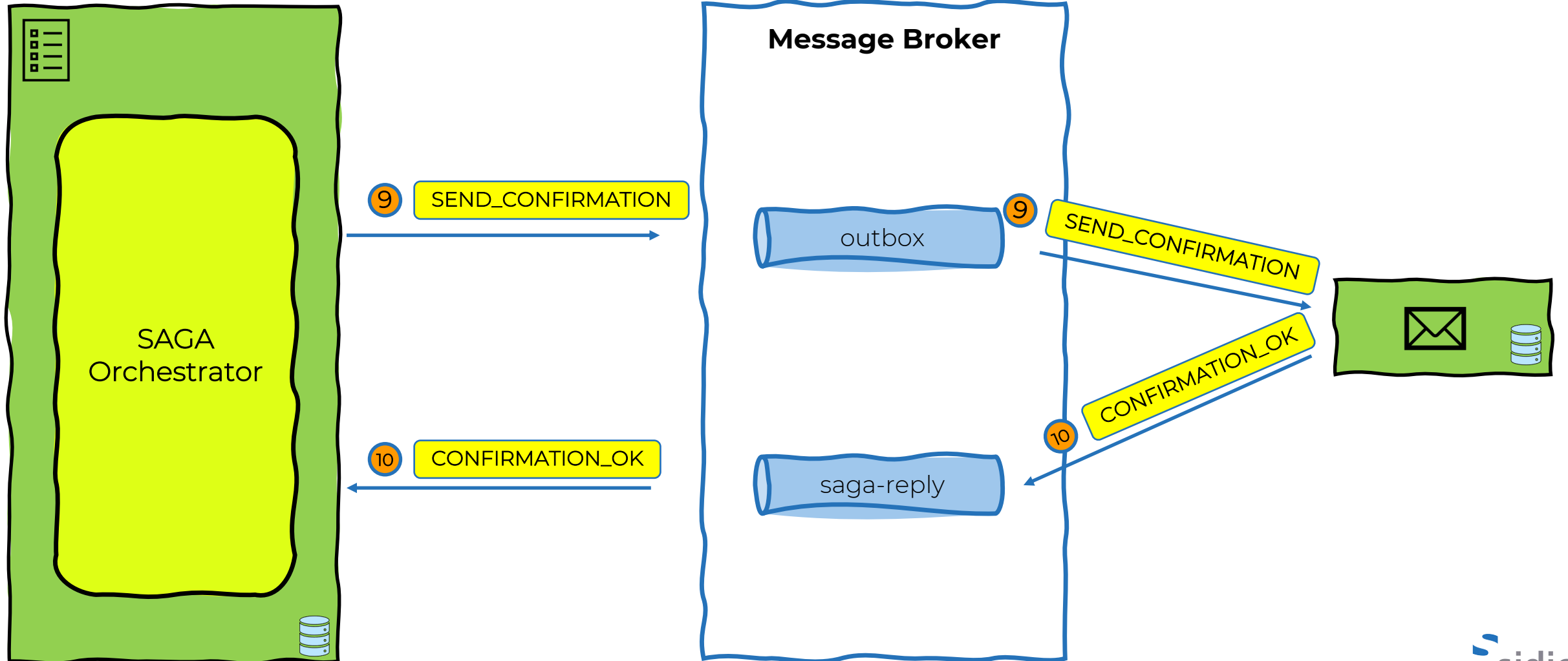
3. BOOK_HOTEL



4. DEBIT_CARD

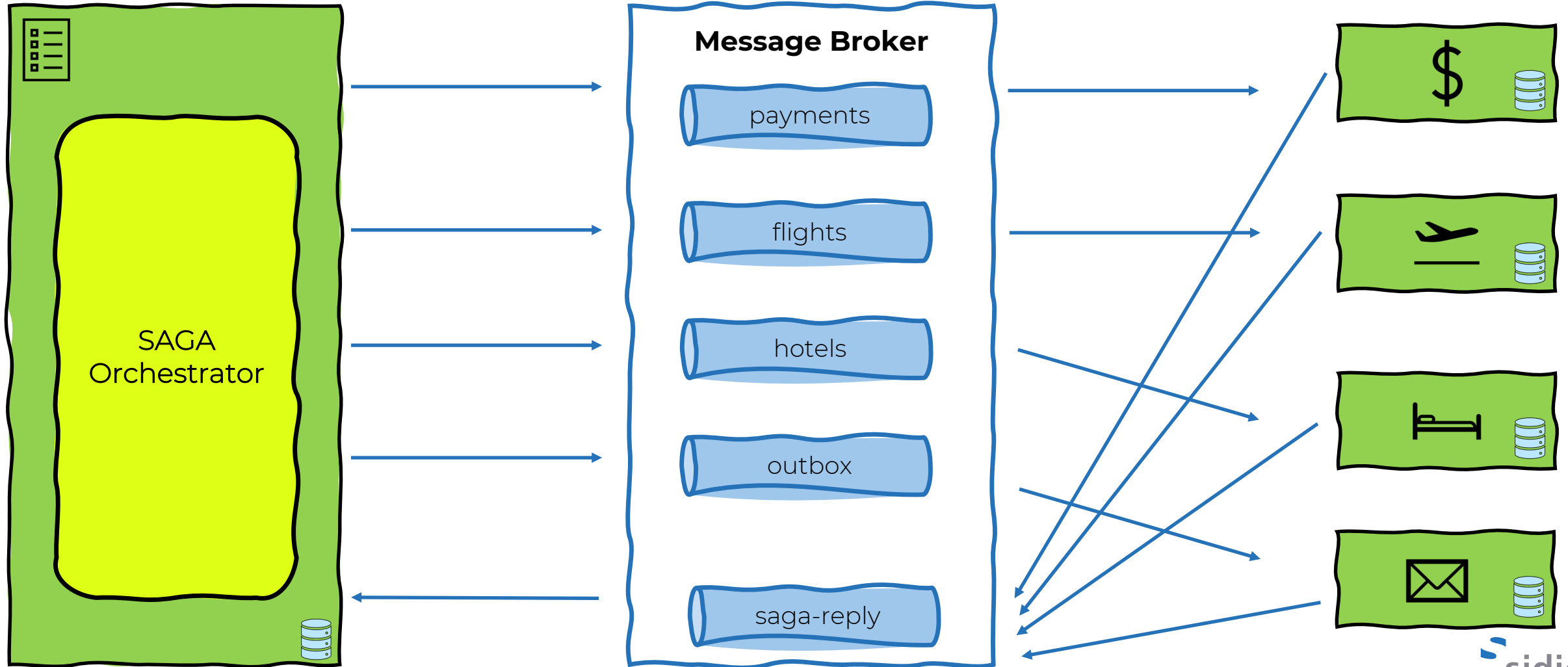


5. SEND_CONFIRMATION



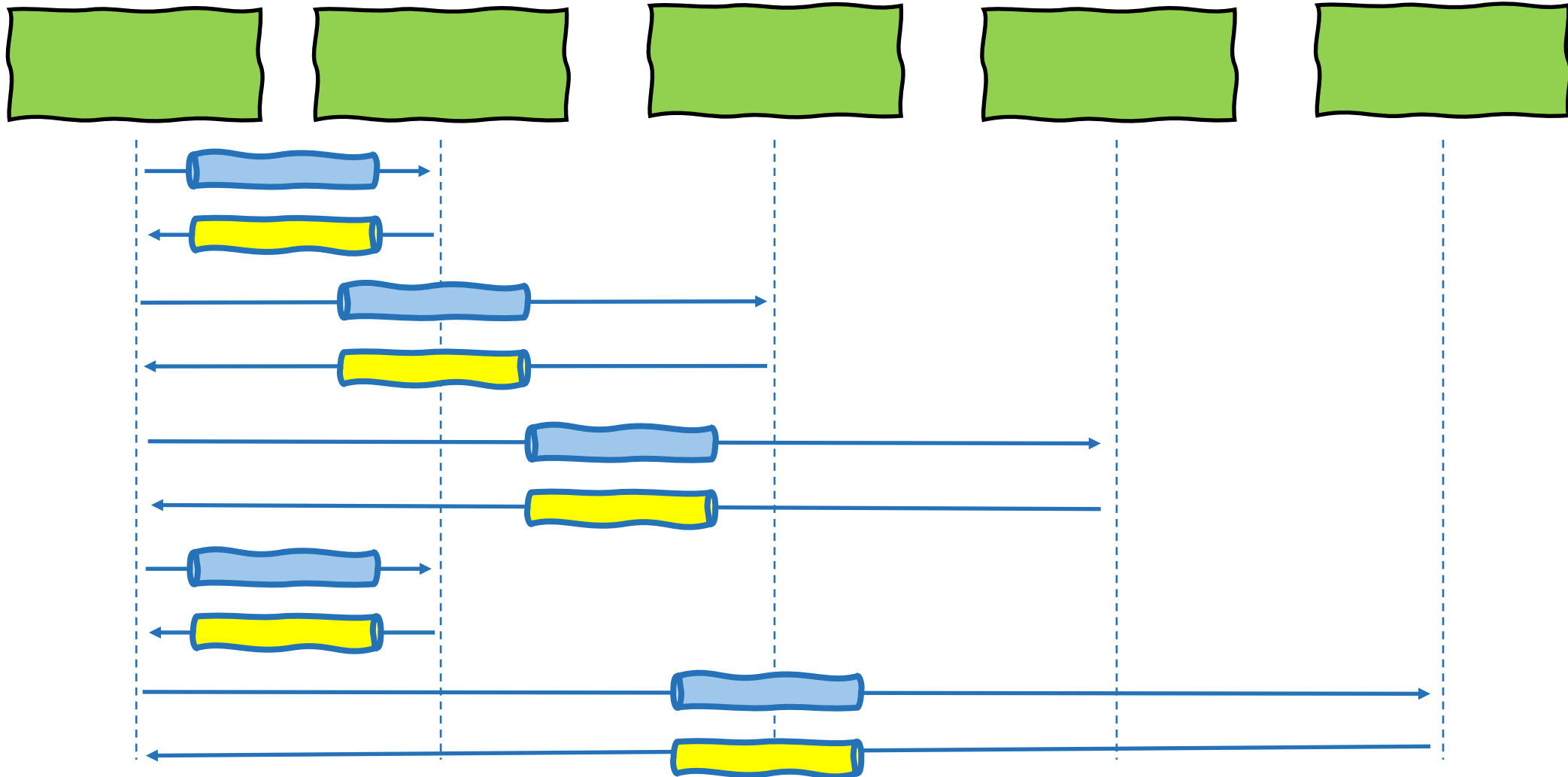
MIT SAGA SERVICE-ÜBERGREIFENDE TRANSAKTIONEN VERWALTEN

ORCHESTRATION-BASED SAGA



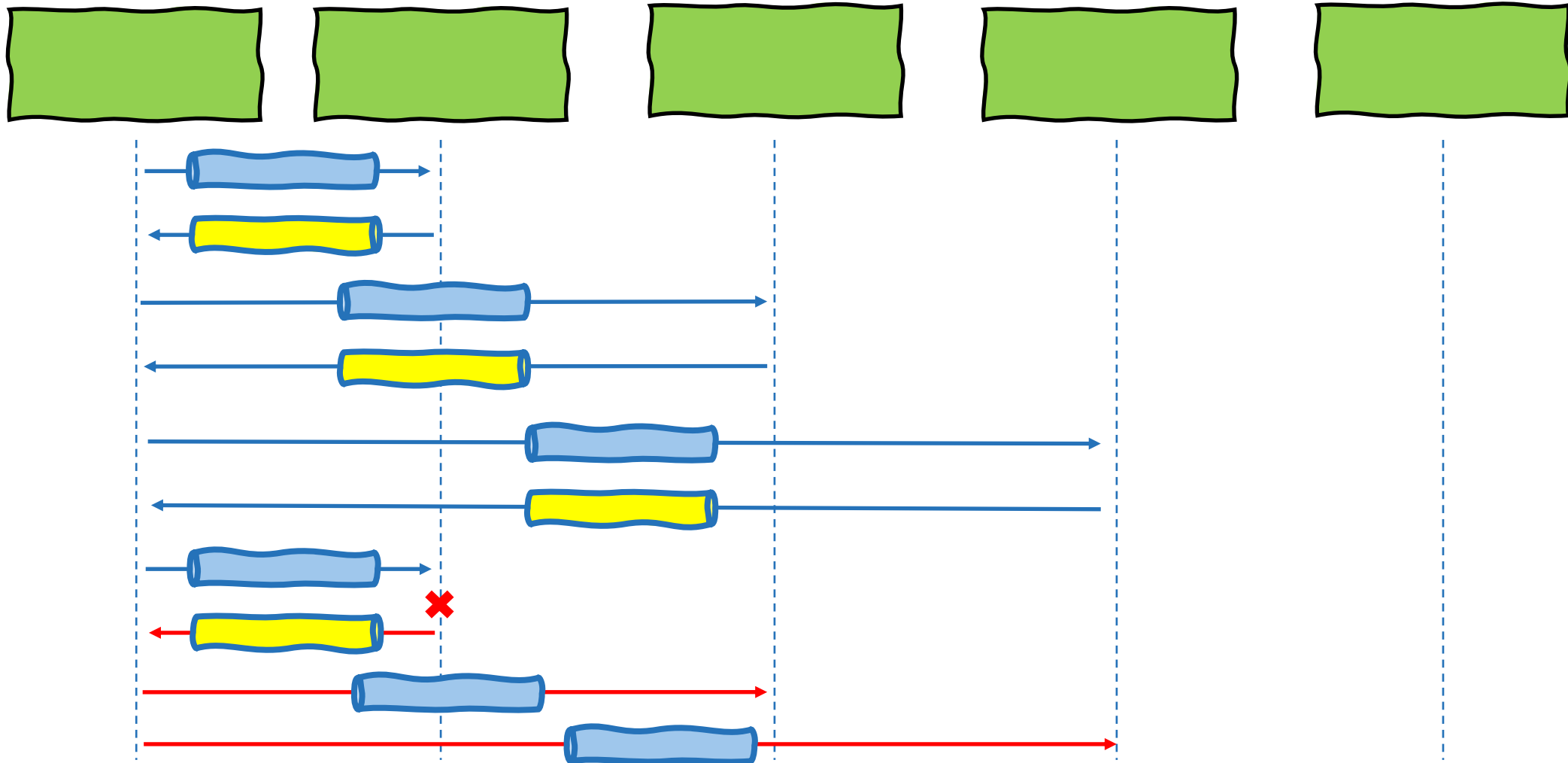
MIT SAGA SERVICE-ÜBERGREIFENDE TRANSAKTIONEN VERWALTEN

ORCHESTRATION-BASED SAGA



MIT SAGA SERVICE-ÜBERGREIFENDE TRANSAKTIONEN VERWALTEN

ORCHESTRATION-BASED SAGA



VORTEILE / NACHTEILE



Einfache Abhängigkeiten



Risiko der Zentralisierung



Loose Koppelung



Trennung der Verantwortlichkeit

SAGA-STRUKTUR

Step	Service	Transaction	Compensation Transaction
1	PaymentService	verifyCard()	---
2	FlightBookingService	bookFlight()	rejectFlight()
3	HotelBookingService	bookHotel()	rejectHotel()
4	PaymentService	debitCard()	---
5	ConfirmationService	sendConfirmation()	---

A Compensatable transactions

B Pivot transactions

C Retriable transactions



EDGE CASES

ACID IN SAGA

- ✓ **A** tomicity
- ✓ **C** onsistency
- ✗ **I** solation → Lack Of Isolation
- ✓ **D** urability

ANOMALIEN IN SAGA

- ✔ Lost Updates
- ✔ Dirty Reads
- ✔ Fuzy / Nonrepeatable Reads

COUNTERMEASURES

- ✓ Semantic Lock
- ✓ Commutative Updates
- ✓ Pessimistic View
- ✓ Reread Value

MIT SAGA SERVICE-ÜBERGREIFENDE TRANSAKTIONEN VERWALTEN

CODING

The screenshot shows the GitLab web interface for the project 'wind-him-up-with-saga'. The main content area displays the project name, ID (36986165), and statistics (2 Commits, 1 Branch, 0 Tags, 35.9 MB Project Storage). A text block explains the problem of data consistency in modern microservice architectures and the use of Saga patterns. Below this, there is an 'Auto DevOps' notification and a table of recent commits. The commit table has the following data:

Name	Last commit	Last update
akhq	add implementation	3 days ago
docker	add implementation	3 days ago
saga-services	add implementation	3 days ago
.gitignore	add implementation	3 days ago
README.md	add implementation	3 days ago

Below the table, the 'README.md' content is partially visible, starting with the title 'Wind Him Up - Mit Saga verteilte Transaktionen in einer Kafka-Architektur verwalten' and an 'Abstract' section.

The screenshot shows an IDE window displaying the 'BookingService.java' file. The code defines a 'BookingService' class with several dependencies and methods. The dependencies include 'BookingRepository', 'BookingEventToBookingEntityMapper', and 'BookingSagaOrchestratorService'. The methods include 'createBooking', 'persistBooking', and 'verifyCard'. The 'createBooking' method uses 'Thread.sleep' to simulate a delay. The 'persistBooking' method uses 'BookingRepository.save' to save the booking entity.

```
package de.sidion.saga.holidaybookingservice.services;

import ...

@Component
@S1f4j
public class BookingService {

    2 usages
    private final BookingRepository bookingRepository;
    2 usages
    private final BookingEventToBookingEntityMapper bookingEventToBookingEntityMapper;
    3 usages
    private final BookingSagaOrchestratorService bookingSagaOrchestratorService;

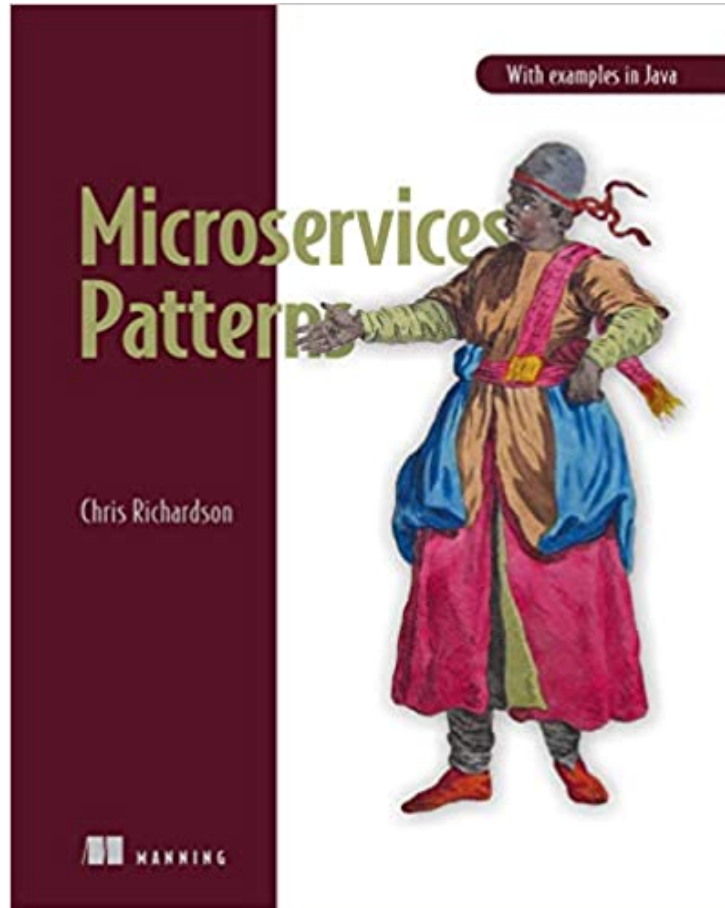
    1 usages
    public BookingService(BookingRepository bookingRepository,
        BookingEventToBookingEntityMapper bookingEventToBookingEntityMapper, BookingSagaOrchestratorService bookingSagaOrchestratorService) {
        this.bookingRepository = bookingRepository;
        this.bookingEventToBookingEntityMapper = bookingEventToBookingEntityMapper;
        this.bookingSagaOrchestratorService = bookingSagaOrchestratorService;
    }

    1 usages
    public void createBooking(final BookingEvent bookingEvent) {
        final BookingEntity bookingEntity = persistBooking(bookingEvent);
        bookingEvent.setBookingId(bookingEntity.getBookingId());
        bookingSagaOrchestratorService.sendBooking(bookingEvent);
        try {
            Thread.sleep( millis 1000L);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
        bookingSagaOrchestratorService.verifyCard(bookingEntity.getBookingId());
    }

    1 usages
    public BookingEntity persistBooking(BookingEvent booking) {
        final BookingEntity bookingEntity = bookingEventToBookingEntityMapper.createEntityFromEvent(booking, status: "PENDING");
        return bookingRepository.save(bookingEntity);
    }
}
```

MIT SAGA SERVICE-ÜBERGREIFENDE TRANSAKTIONEN VERWALTEN

BUCHVORSCHLAG



Microservices Patterns

von Chris Richardson

CODE

Die ausführlichen Codebeispiele findet Ihr unter:



<https://gitlab.com/sidion/vortraege/2022/javaforumstuttgart/wind-him-up-with-saga>

SIDION

Zuhören. Analysieren. Beraten.