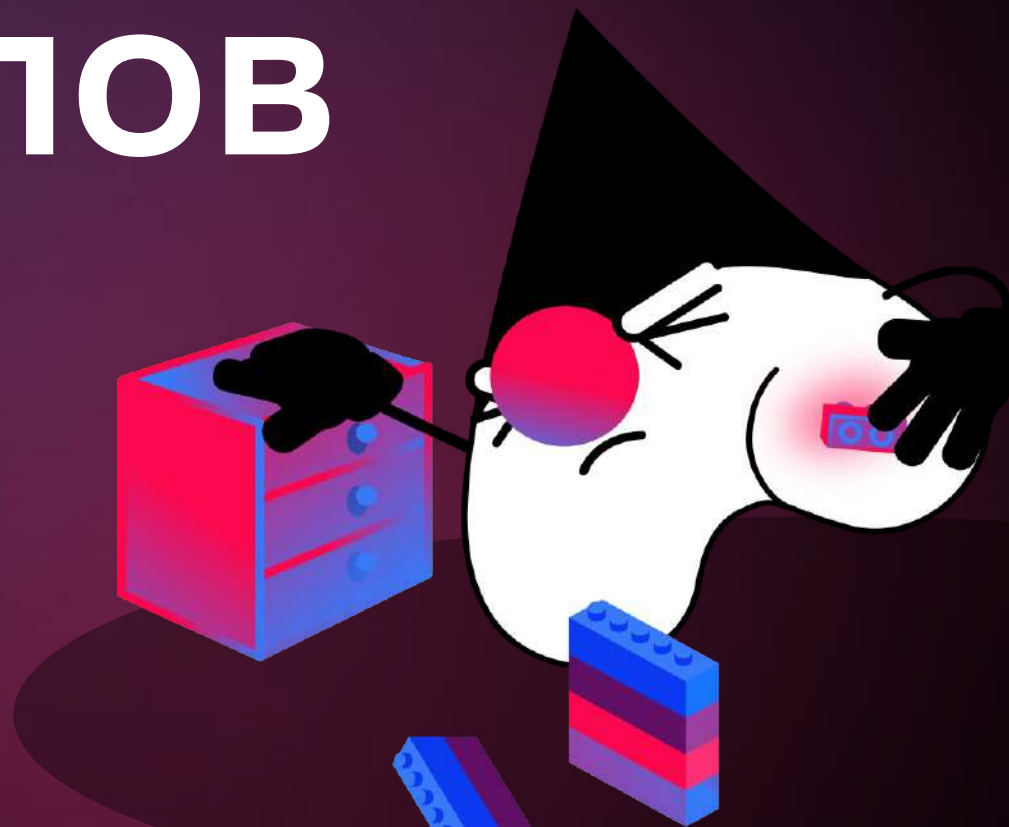
Ночь провалов

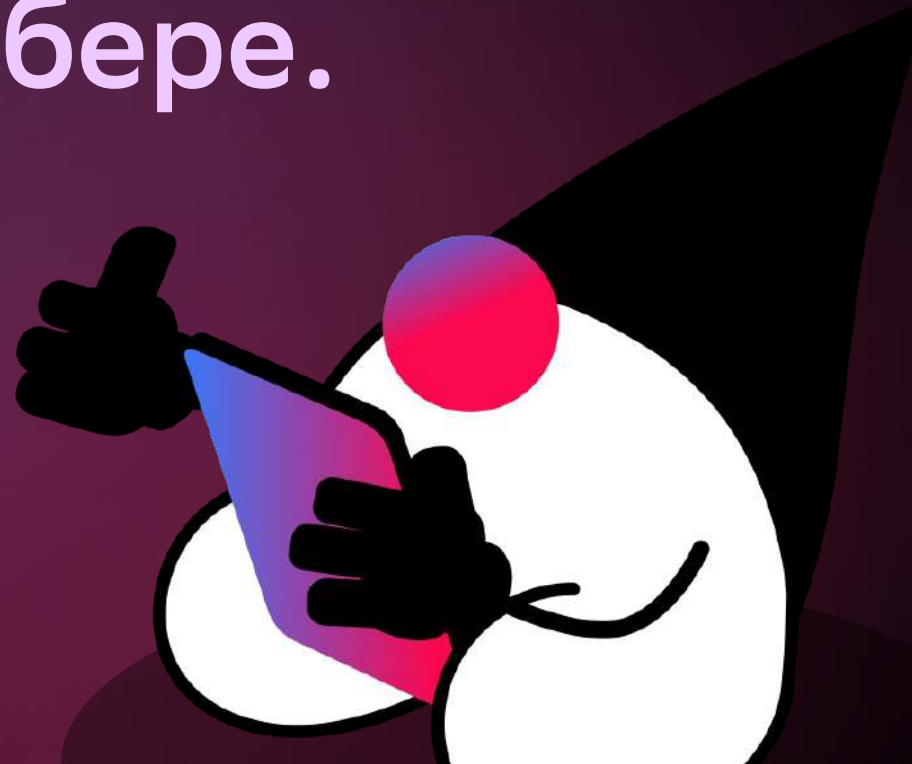


java
community
#SBERTEAM

Как продукт воровал деньги у людей и пароходов, и ему за это ничего не было (почти). Это происходило не в Сбере.

Вячеслав Чернышов

java
community
#SBERTEAM



Вячеслав Чернышов

Backend-разработчик
СберТех, Platform V Flow

Случилось страшное:
приложение начало воровать деньги у
честных покупателей.

У кого приложение воровало деньги?

Менеджер проекта

Бизнес-аналитик

Системный аналитик

Тестировщик

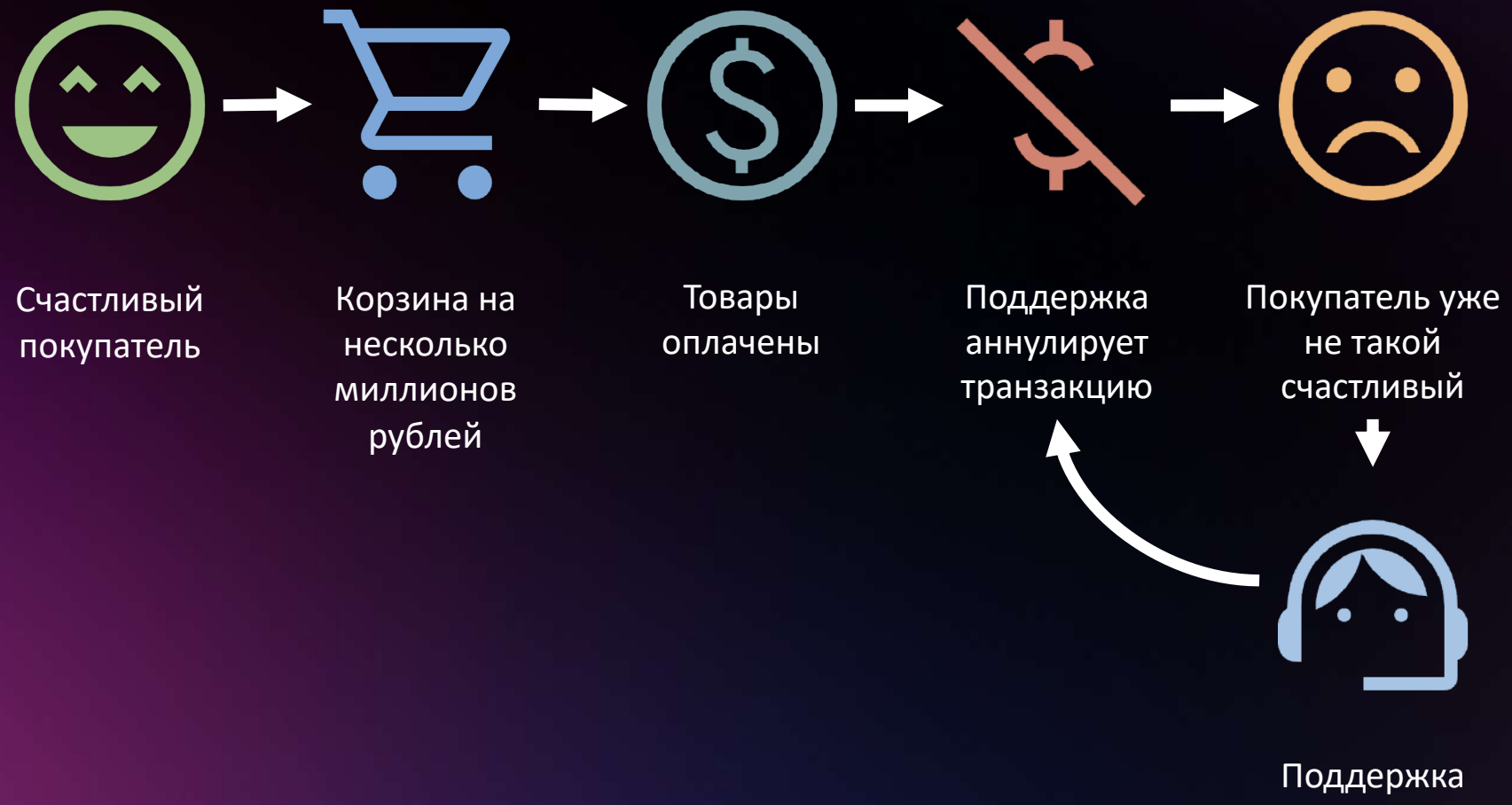
Поддержка

Backend-разработчик

Приложение позволяло совершить очень дорогую покупку.

Кража денег: как это было?

Кража денег: как это было?



Кража денег: как это было?

Кража денег: как это было?

Итог:

+ 1 сорванная сделка

+ 1 недовольный покупатель

Этапы проведения сделки

Создание задания на проведение сделки

Запуск события по расписанию

Списание средств со счёта покупателя

Генерация платёжного документа

Сохранение документа в файловое хранилище

```java
@Service
@RequiredArgsConstructor
public class DealServiceImpl implements DealService {

    private final TaskRepository repository;
    private final PaymentService paymentService;
    private final DocumentService documentService;

    @Transactional
    @Scheduled(cron = "0/30 * * * * ?") every 30 seconds
    @Override
    public void processAllTasks() {
        var tasksToProcess = repository.findAll(TaskStatus.TO_PROCESS);
        tasksToProcess
                .stream()
                .peek(task -> repository.updateStatus(task.id(), TaskStatus.IN_PROCESS))
                .peek(task -> paymentService.pay(Payment.of(task)))
                .peek(task -> documentService.generate(Document.of(task)))
                .forEach(task -> repository.updateStatus(task.id(), TaskStatus.CLOSED));
    }
}
```

Каждые 30 секунд сервис просыпается и проверяет задания на проведение сделок

```java
@Service
@RequiredArgsConstructor
public class DealServiceImpl implements DealService {

    private final TaskRepository repository;
    private final PaymentService paymentService;
    private final DocumentService documentService;

    @Transactional
    @Scheduled(cron = "0/30 * * * * ?") every 30 seconds
    @Override
    public void processAllTasks() {
        var tasksToProcess = repository.findAll(TaskStatus.TO_PROCESS);
        tasksToProcess
                .stream()
                .peek(task -> repository.updateStatus(task.id(), TaskStatus.IN_PROCESS))
                .peek(task -> paymentService.pay(Payment.of(task)))
                .peek(task -> documentService.generate(Document.of(task)))
                .forEach(task -> repository.updateStatus(task.id(), TaskStatus.CLOSED));
    }
}
```

Получение всех заданий на исполнение

```java
@Service
@RequiredArgsConstructor
public class DealServiceImpl implements DealService {

    private final TaskRepository repository;
    private final PaymentService paymentService;
    private final DocumentService documentService;

    @Transactional
    @Scheduled(cron = "0/30 * * * * ?") every 30 seconds
    @Override
    public void processAllTasks() {
        var tasksToProcess = repository.findAll(TaskStatus.TO_PROCESS);
        tasksToProcess
                .stream()
                .peek(task -> repository.updateStatus(task.id(), TaskStatus.IN_PROCESS))
                .peek(task -> paymentService.pay(Payment.of(task)))
                .peek(task -> documentService.generate(Document.of(task)))
                .forEach(task -> repository.updateStatus(task.id(), TaskStatus.CLOSED));
    }
}
```

Найденные элементы сервис помечает особым статусом, чтобы не найти их заново

```java
@Service
@RequiredArgsConstructor
public class DealServiceImpl implements DealService {

    private final TaskRepository repository;
    private final PaymentService paymentService;
    private final DocumentService documentService;

    @Transactional
    @Scheduled(cron = "0/30 * * * * ?") every 30 seconds
    @Override
    public void processAllTasks() {
        var tasksToProcess = repository.findAll(TaskStatus.TO_PROCESS);
        tasksToProcess
                .stream()
                .peek(task -> repository.updateStatus(task.id(), TaskStatus.IN_PROCESS))
                .peek(task -> paymentService.pay(Payment.of(task)))
                .peek(task -> documentService.generate(Document.of(task)))
                .forEach(task -> repository.updateStatus(task.id(), TaskStatus.CLOSED));
    }
}
```

Далее, сервис списывает деньги

```java
@Service
@RequiredArgsConstructor
public class DealServiceImpl implements DealService {

    private final TaskRepository repository;
    private final PaymentService paymentService;
    private final DocumentService documentService;

    @Transactional
    @Scheduled(cron = "0/30 * * * * ?") every 30 seconds
    @Override
    public void processAllTasks() {
        var tasksToProcess = repository.findAll(TaskStatus.TO_PROCESS);
        tasksToProcess
                .stream()
                .peek(task -> repository.updateStatus(task.id(), TaskStatus.IN_PROCESS))
                .peek(task -> paymentService.pay(Payment.of(task)))
                .peek(task -> documentService.generate(Document.of(task)))
                .forEach(task -> repository.updateStatus(task.id(), TaskStatus.CLOSED));
    }
}
```

И генерирует платёжный документ. Эта долгая операция длится порядка полутора минут.

```java
@Service
@RequiredArgsConstructor
public class DealServiceImpl implements DealService {

    private final TaskRepository repository;
    private final PaymentService paymentService;
    private final DocumentService documentService;

    @Transactional
    @Scheduled(cron = "0/30 * * * * ?") every 30 seconds
    @Override
    public void processAllTasks() {
        var tasksToProcess = repository.findAll(TaskStatus.TO_PROCESS);
        tasksToProcess
                .stream()
                .peek(task -> repository.updateStatus(task.id(), TaskStatus.IN_PROCESS))
                .peek(task -> paymentService.pay(Payment.of(task)))
                .peek(task -> documentService.generate(Document.of(task)))
                .forEach(task -> repository.updateStatus(task.id(), TaskStatus.CLOSED));
    }
}
```

В конце, задание окончательно помечается как закрытое.

Что не так?

```java
@Service
@RequiredArgsConstructor
public class DealServiceImpl implements DealService {

    private final TaskRepository repository;
    private final PaymentService paymentService;
    private final DocumentService documentService;

    @Transactional
    @Scheduled(cron = "0/30 * * * * ?") every 30 seconds
    @Override
    public void processAllTasks() {
        var tasksToProcess = repository.findAll(TaskStatus.TO_PROCESS);
        tasksToProcess
                .stream()
                .peek(task -> repository.updateStatus(task.id(), TaskStatus.IN_PROCESS))
                .peek(task -> paymentService.pay(Payment.of(task)))
                .peek(task -> documentService.generate(Document.of(task)))
                .forEach(task -> repository.updateStatus(task.id(), TaskStatus.CLOSED));
    }
}
```

Если что-то пойдёт не так, транзакция откатится, конечно.

Уровни изоляции транзакций

READ UNCOMMITED

Позволяет считывать не закоммиченные изменения.

READ COMMITED

Позволяет считывать только закоммиченные изменения.

REPEATABLE READ

Читающая транзакция блокирует изменение читаемых данных.

SERIALIZABLE

Последовательное выполнение транзакций.

Что не так?

POD 1

POD 2

Транзакция

Фактически, статус задания остаётся *TO_PROCESS.*

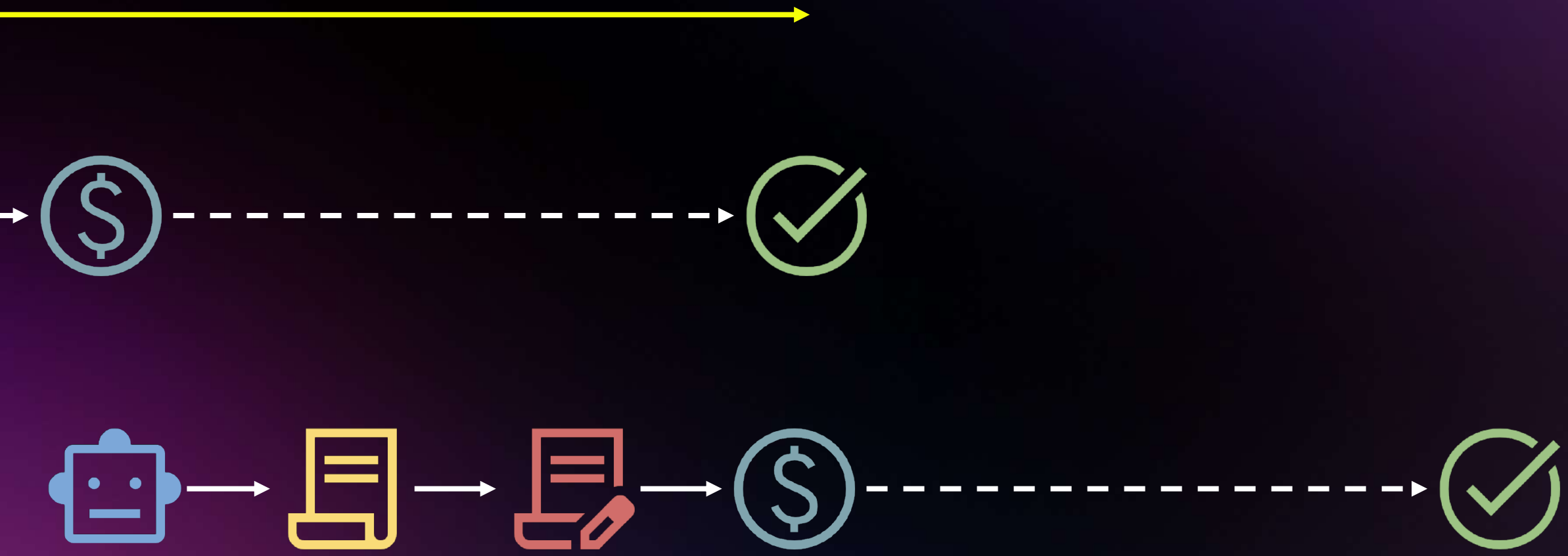
Фактически, статус задания остаётся *TO_PROCESS.*

И задание забирает другой pod.

Транзакция

Транзакция завершается

Транзакция



Но если очень хорошо присмотреться,
дело даже не во второй поде.

```java
@Service
@RequiredArgsConstructor
public class DealServiceImpl implements DealService {

    private final TaskRepository repository;
    private final PaymentService paymentService;
    private final DocumentService documentService;

    @Transactional
    @Scheduled(cron = "0/30 * * * * ?") every 30 seconds
    @Override
    public void processAllTasks() {
        var tasksToProcess = repository.findAll(TaskStatus.TO_PROCESS);
        tasksToProcess
                .stream()
                .peek(task -> repository.updateStatus(task.id(), TaskStatus.IN_PROCESS))
                .peek(task -> paymentService.pay(Payment.of(task)))
                .peek(task -> documentService.generate(Document.of(task)))
                .forEach(task -> repository.updateStatus(task.id(), TaskStatus.CLOSED));
    }
}
```

Статус не обновляется до применения коммита

Решения?

Решение 1

SelectForUpdate. Блокировка полученных записей до последующего изменения.

Решение 2

Понижение уровня транзакции до READ UNCOMMITED.

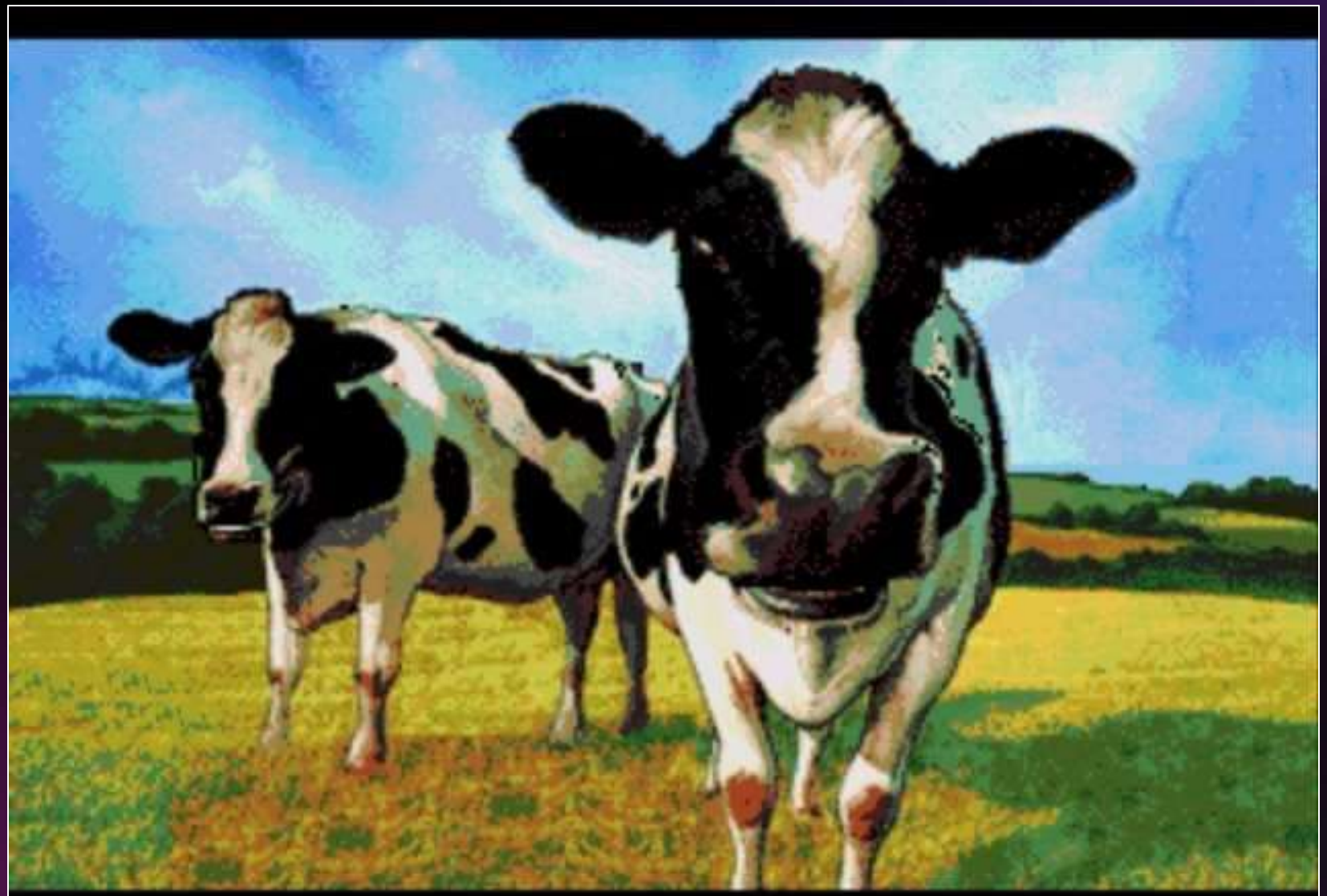
```java
@Service
@RequiredArgsConstructor
public class DealServiceImpl implements DealService {

    private final TaskRepository repository;
    private final PaymentService paymentService;
    private final DocumentService documentService;

    @Transactional(isolation = Isolation.READ_UNCOMMITTED)
    @Scheduled(cron = "0/30 * * * * ?") every 30 seconds
    @Override
    public void processAllTasks() {
        var tasksToProcess = repository.findAll(TaskStatus.TO_PROCESS);
        tasksToProcess
                .stream()
                .peek(task -> repository.updateStatus(task.id(), TaskStatus.IN_PROCESS))
                .peek(task -> paymentService.pay(Payment.of(task)))
                .peek(task -> documentService.generate(Document.of(task)))
                .forEach(task -> repository.updateStatus(task.id(), TaskStatus.CLOSED));
    }
}
```

Кража денег: как это было?

Счастливый покупатель → Корзина на несколько миллионов рублей → Товары оплачены → Покупатель безмерно счастлив → Дела компании идут в гору



Well done!

Как избежать повторения подобных ситуаций?

Больше тестов. Самых разных.

Модульное, интеграционное, функциональное, автоматизированное, нагрузочное. Помните: если вы не будете тестировать ваше приложение, его будут тестировать пользователи своими деньгами.

Окружение IFT должно полностью соответствовать окружению PROD.

Менеджмент должен любыми путями обеспечить соответствие IFT-стенда проду.

Менеджмент должен выделять любые ресурсы на поддержание репутации продукта.

Менеджмент должен был любыми путями обеспечить устранение бага в течение ближайших часов.

Покрывайте приложение тестами, и оно с меньшей вероятностью будет красть деньги у простых людей.

Профиль на Хабре

Канал в Telegram