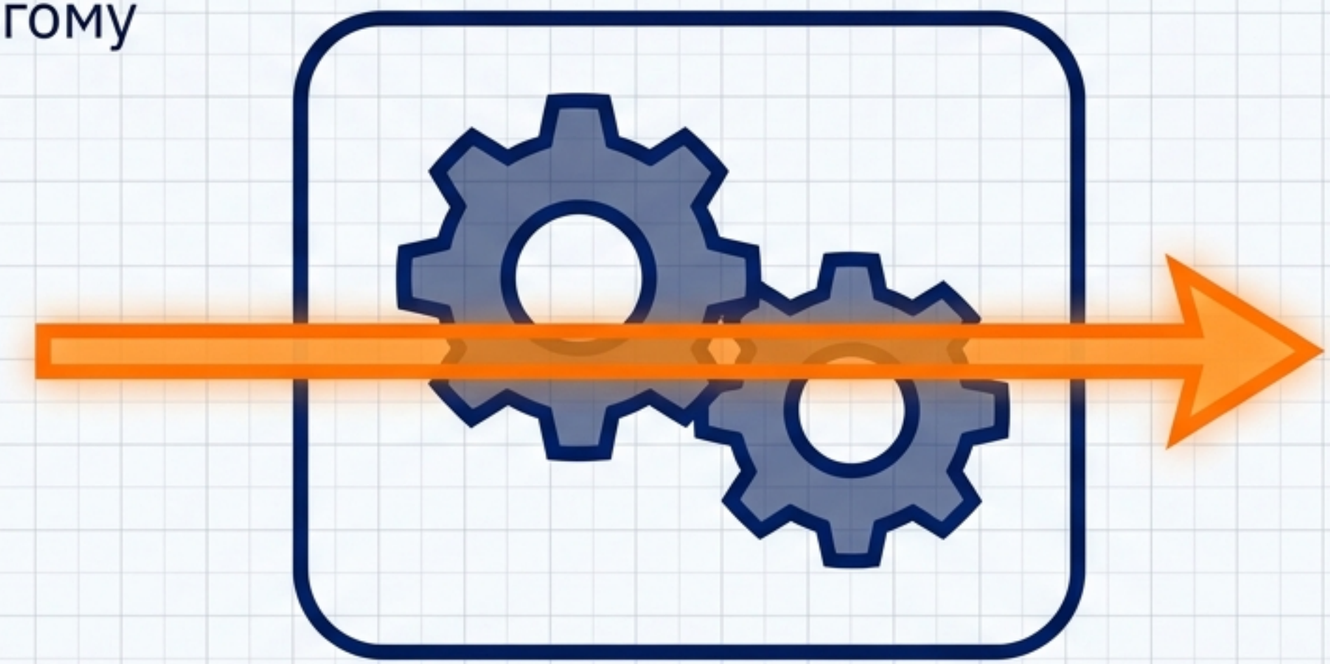
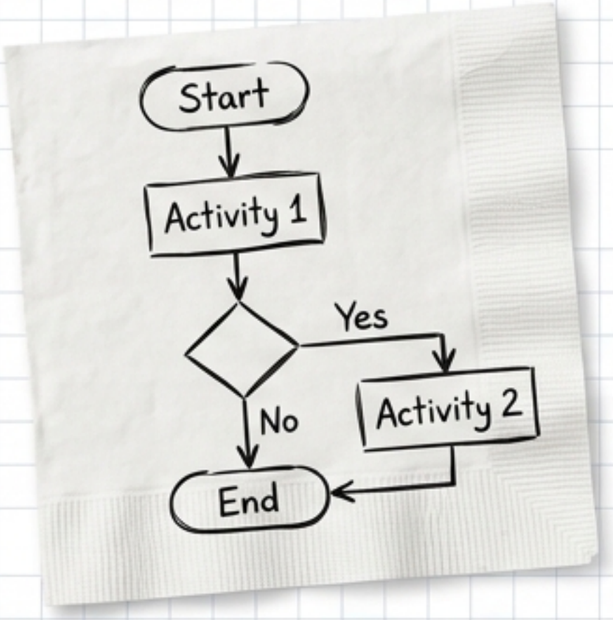
Архитектура бизнес-процессов: Исполняемый чертёж BPMN 2.0

От визуальных метафор к строгому синтаксису и оркестрации распределенных систем.



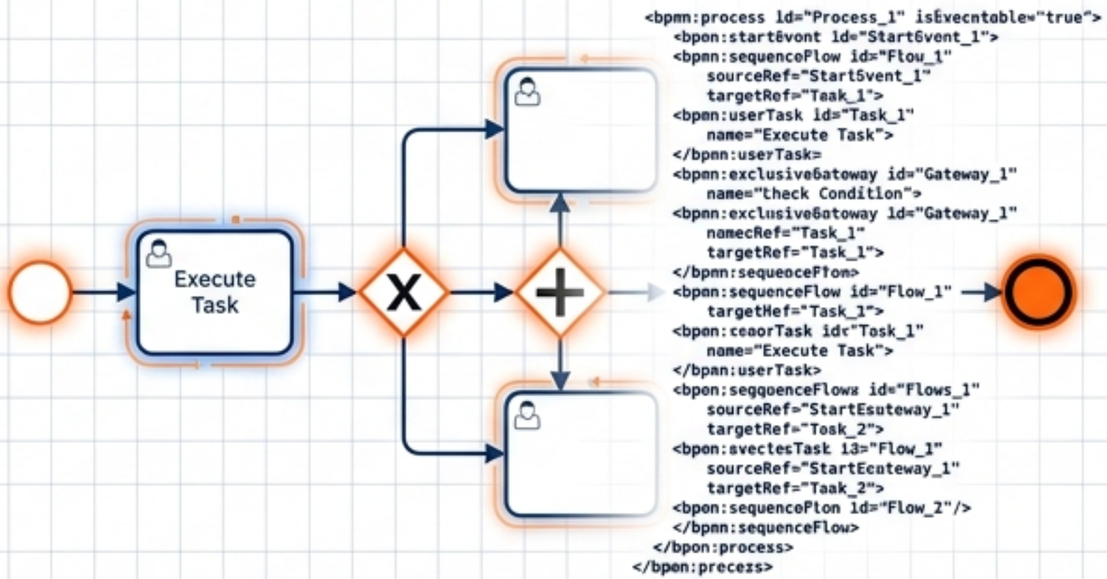
Смена парадигмы: От картинки к коду

UML Activity Diagram (1997)



- **Фокус:** Визуализация алгоритма.
- **Шлюзы:** Все условия одним ромбом.
- **События:** Только 3 (Start, Stop, Flow Final).
- **Формат:** Картинка (не предназначена для исполнения).

BPMN 2.0 (ISO 19510, 2011)

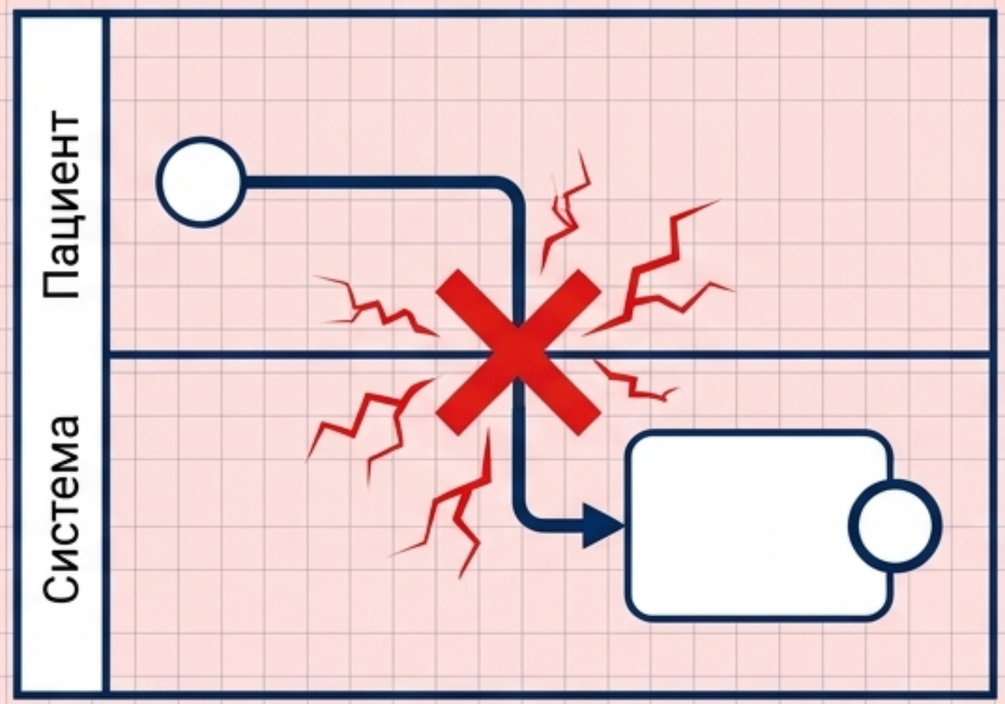


- **Фокус:** Исполняемая модель процесса (Camunda).
- **Шлюзы:** 5 строго различимых типов.
- **События:** ~50+ комбинаций с маркерами.
- **Формат:** Строгая XML-сериализация.

Ключевой вывод: BPMN — это техническая карта с точными параметрами. BPM-движок не допускает вольных отклонений.

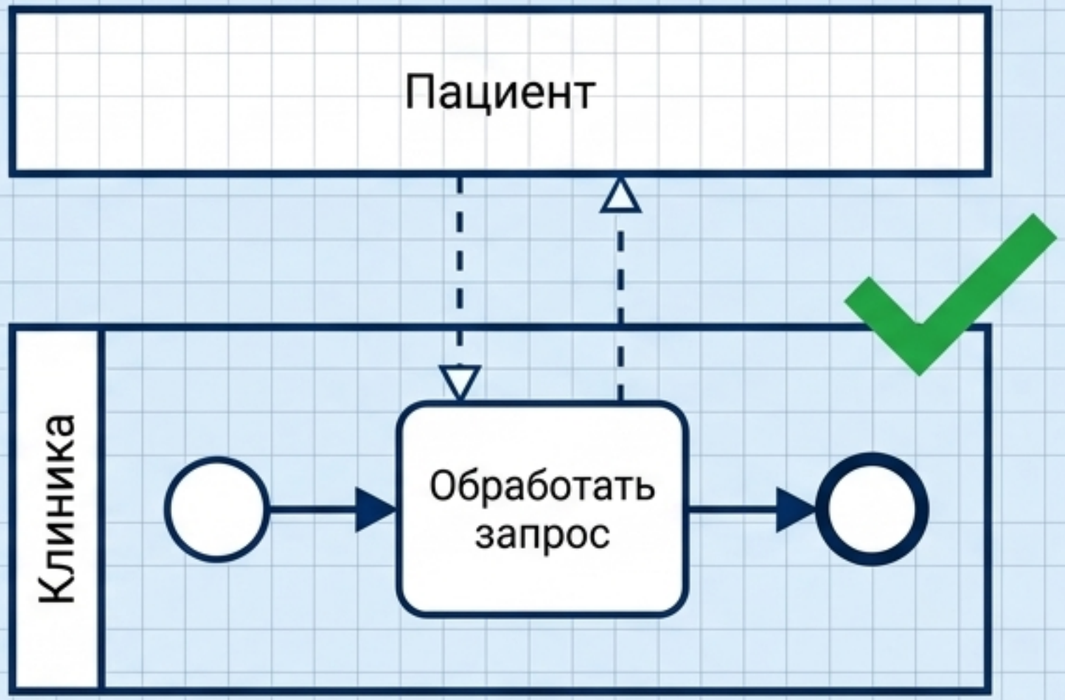
⚠️ Критическое правило: Клиент ≠ Lane

DON'T



❌ **ОШИБКА:** Пациент не владеет процессом клиники. Sequence Flow не может управлять внешним клиентом.

DO



✅ **ПРАВИЛЬНО:** Внешний участник — это отдельный пул. Связь между ними только через пунктирный Message Flow.

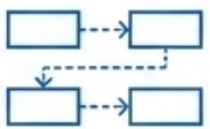
Алгоритм маршрутизации участников: Pool или Lane?

Шаг 1: Этот участник — часть нашей организации?

НЕТ

Отдельный Pool

(Пример: Пациент, Шлюз Stripe, Курьер)



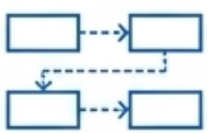
ДА

Шаг 2: Участник владеет своим независимым процессом?

ДА

Отдельный Pool

(Пример: Пациент, Шлюз Stripe, Курьер)



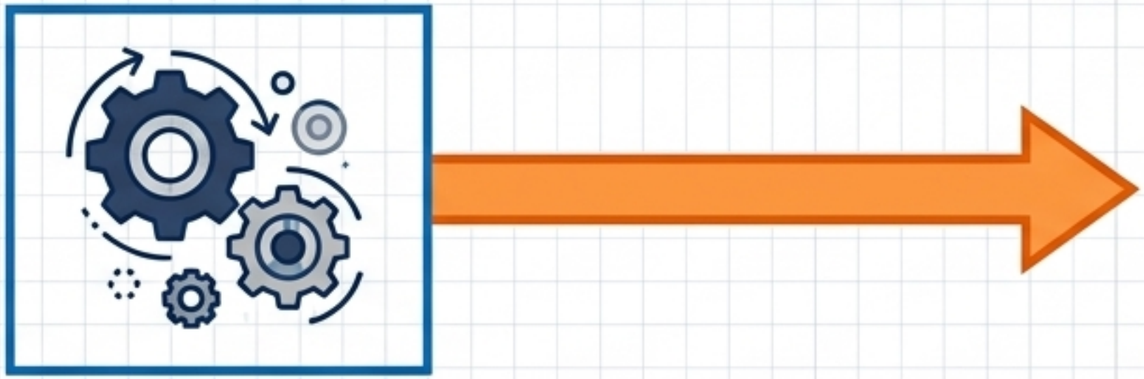
НЕТ

Lane внутри нашего Pool

(Пример: Регистратура, Склад, Бухгалтерия)

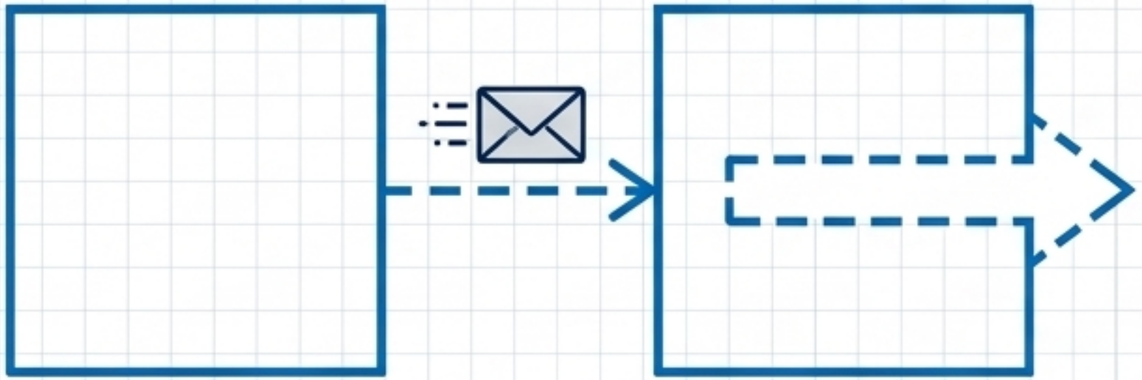
Физика соединений: Управление против Данных

Sequence Flow (Поток управления)



- **Правило:** Строго внутри одного пула. Пересекать границу ЗАПРЕЩЕНО.
- **Логика:** Порядок выполнения. Может содержать условия ветвления.
- **Код:** `[Сумма > 1000]`

Message Flow (Поток данных)



- **Правило:** Только между разными пулами. Внутри пула ЗАПРЕЩЕНО.
- **Логика:** Обмен сообщениями. Не содержит условий.
- **Смысл:** Передача данных, а не контроля.

Анатомия задач: Человек против Машины

User Task

- Исполнитель: Человек
- Время: Минуты — дни
- Интерфейс: UI форма (React)

```
camunda:assignee="${manager}"
```

JSON-Контракт для Service Task

Вход (Input):
`paymentToken, amount, orderId`

Выход (Output):
`chargeId, status`

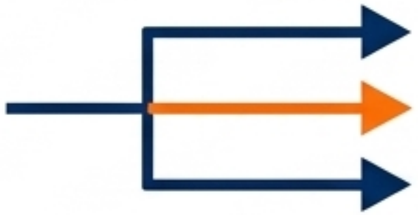
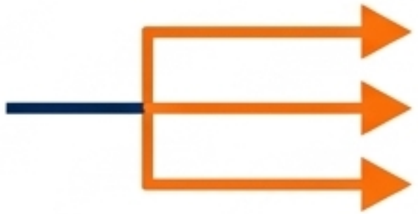
Сбой (Failure):
Тайм-аут 30 сек → Цикл Retry

Service Task

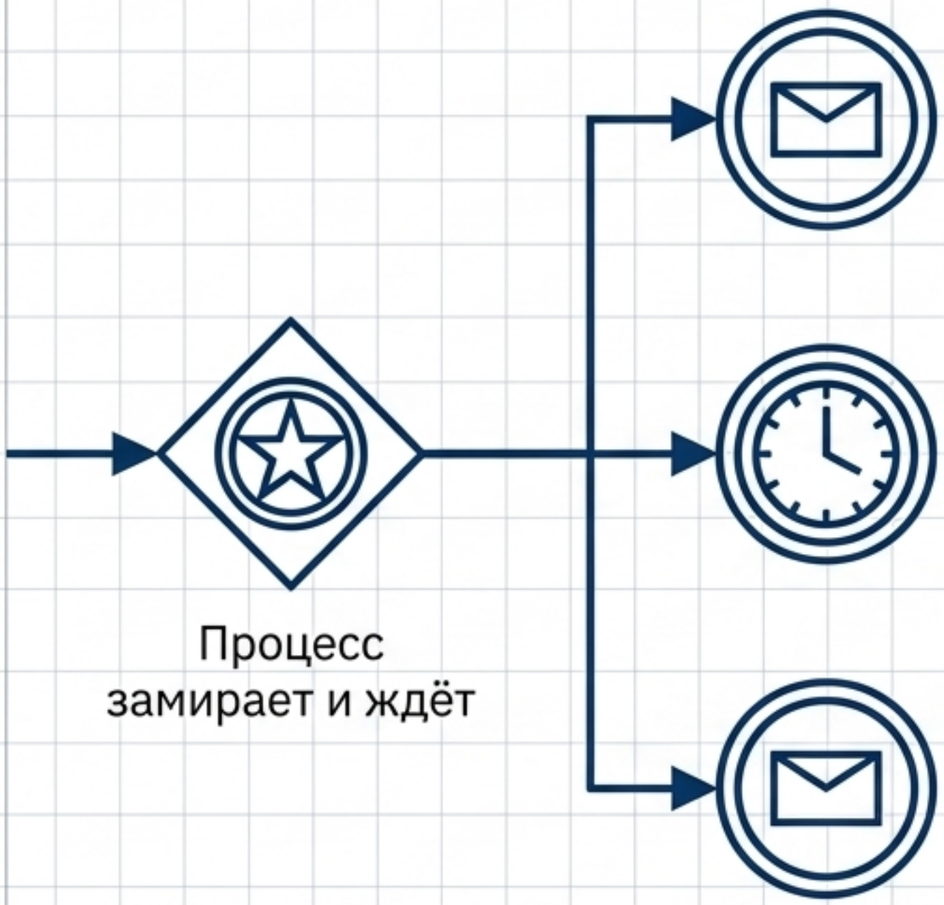
- Исполнитель: Машина
- Время: Миллисекунды
- Интерфейс: API / Микросервис

```
camunda:delegateExpression="${paymentService}"
```

Матрица маршрутизации (Шлюзы по данным)

Шлюз	Логика маршрутизации	Правило	Код
 Exclusive (XOR)	 Ровно один путь.	Обязательно наличие Default Flow.	`switch / if-else`
 Parallel (AND)	 Все пути одновременно.	На входе работает как синхронизатор (ждет все ветки).	`fork-join / Promise.all()`
 Inclusive (OR)	 Один или более путей.	Выполняются все истинные условия.	несколько независимых `if`

Маршрутизация по событиям (Event-based Gateway)



Процесс замирает и ждёт

Событие 1: Объявили посадку на рейс (Message).

Событие 2: Прошло 30 минут, страховка от вечного ожидания (Timer).

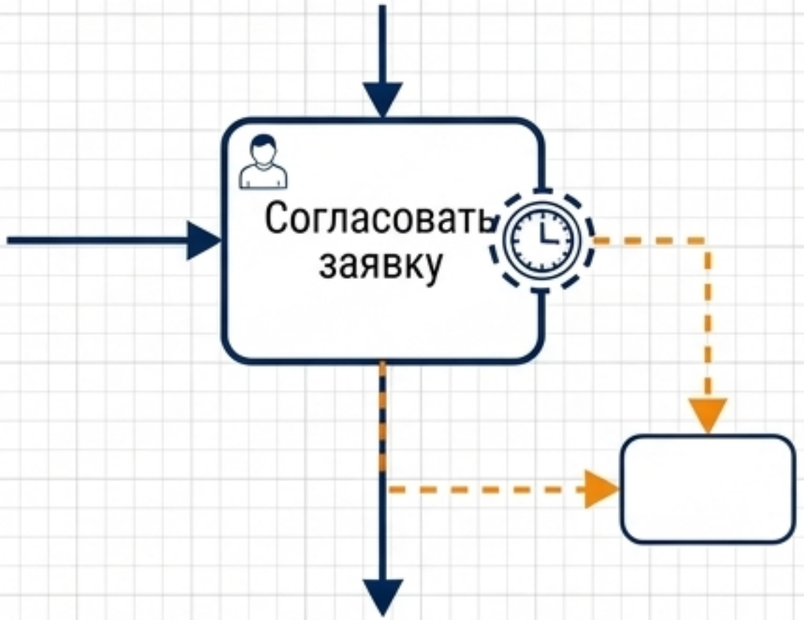
Событие 3: SMS об отмене рейса (Message).

Выбор пути определяет **СОБЫТИЕ** (кто наступит первым), а не данные. Аналог `Promise.race()`

СТРОГОЕ ПРАВИЛО: За Event-based шлюзом следуют ТОЛЬКО ловящие события (Catch Events). Задачи ставить нельзя.

Управление временем и исключениями: Boundary Events

Non-Interrupting (Пунктирная линия)



Сценарий: 24h Напоминание.
Процесс идёт параллельно. Задача остаётся активной у руководителя.

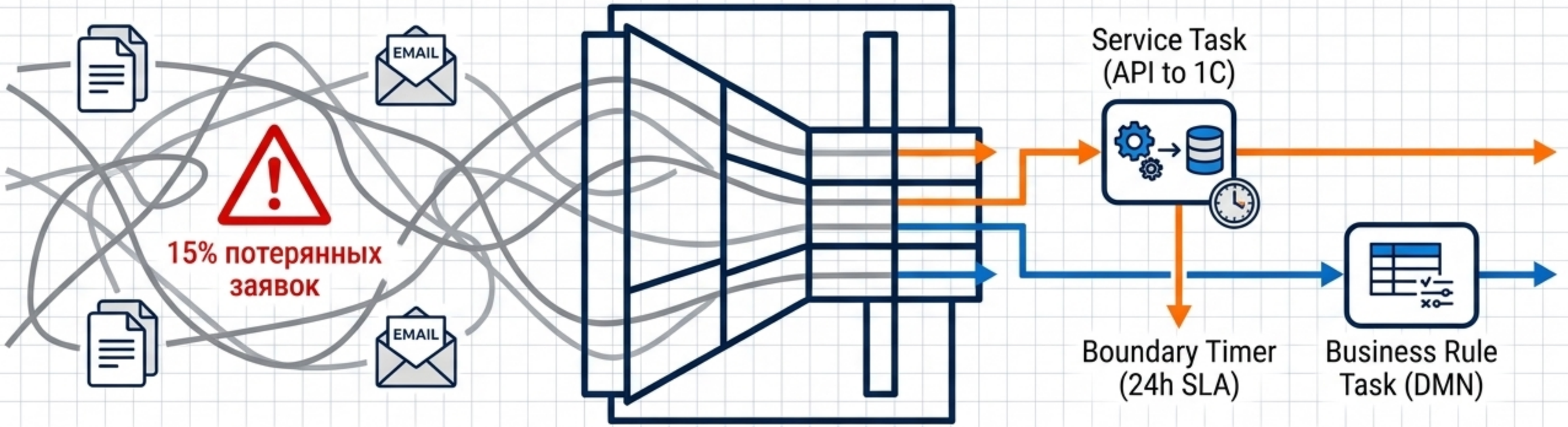
Interrupting (Сплошная линия)

Сценарий: 48h Эскалация.
Время вышло. Задача немедленно прерывается и передаётся финдиректору.

Важно: Error Boundary (❌) ВСЕГДА Interrupting.
Системный сбой нельзя игнорировать.

Gap-анализ: От ручного хаоса к прямой обработке

AS-IS → TO-BE



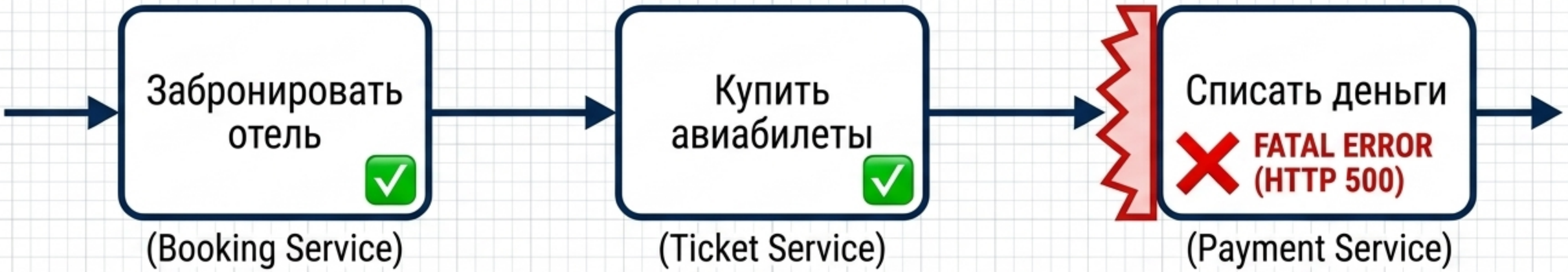
- Бумажный документооборот
- Ручной ввод в 1С (ошибки)
- Отсутствие SLA (висит неделю)

Исполняемая модель BPMN

- API-интеграция с 1С
- Эскалация по SLA таймерам
- Автоматическая DMN-маршрутизация

Метрики трансформации: Автоматизация 70% операций. Время согласования снижено с 5 дней до 4 часов.

Оркестровка микросервисов: Проблема распределенных транзакций



Опасность: Данные неконсистентны

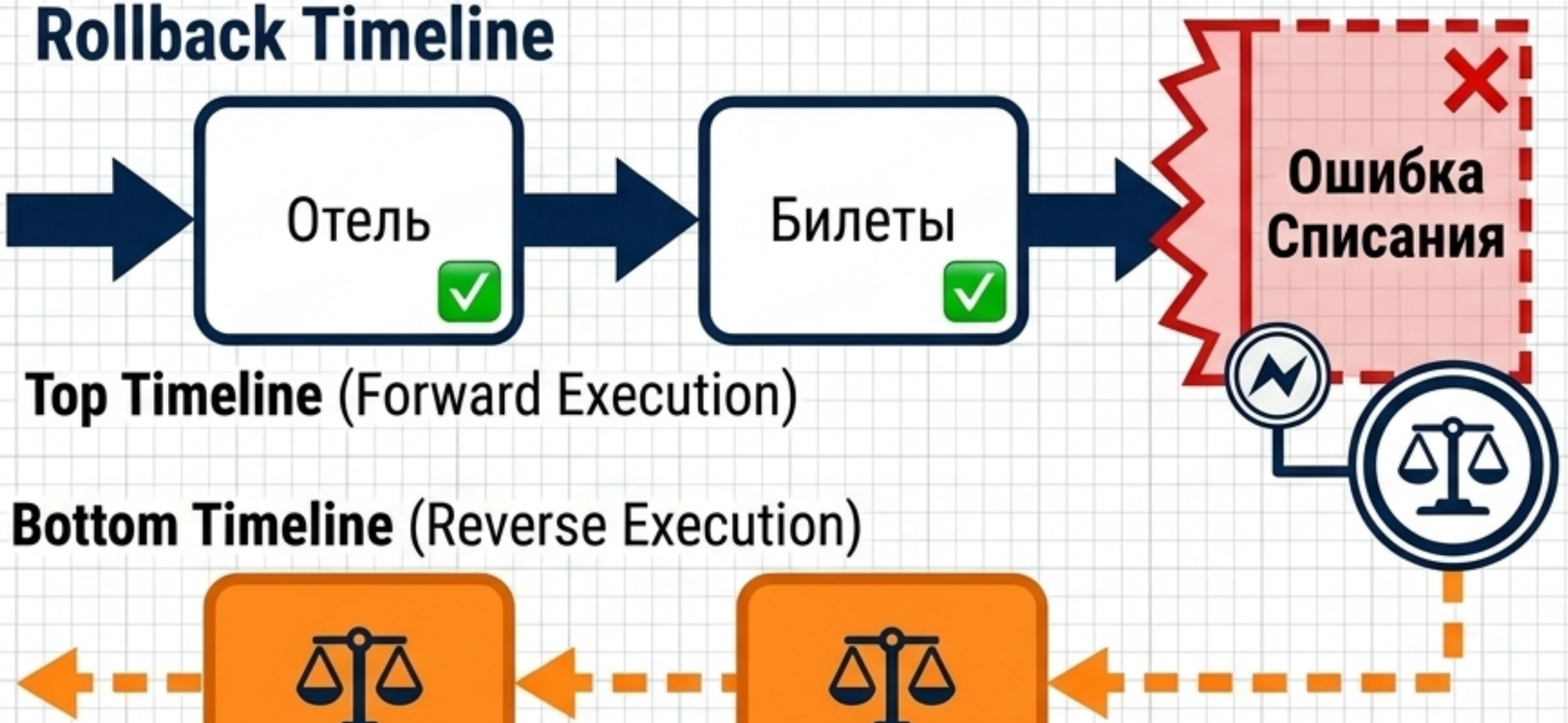
Отель забронирован, билеты куплены, но деньги не списаны.

Обычный **SQL-откат** (ROLLBACK) здесь **НЕ работает**, так как системы независимы.

Требуемая Архитектура: BPMN-движок должен выступить в роли оркестратора, чтобы физически «отмотать» реальные действия во внешних системах.

Паттерн Saga: Восстановление через Компенсацию (Compensation)

Rollback Timeline



Требования к Аналитику

- **Идемпотентность:** Повторный вызов отмены не ломает систему.
- **Retry Policy:** 3 попытки с экспоненциальной задержкой.
- **Временные окна:** Проверка SLA (Например: можно ли сдать билет через 24ч?).

Чек-лист Архитектора: Валидация BPMN-модели

Structural Integrity (Структура)

- ☐ **Sequence Flow** строго внутри пулов?
- ☐ Внешние участники вынесены в отдельные пулы (**Message Flow**)?
- ☐ У процесса есть явные **Start Event** и минимум один **End Event** на каждую ветку?

Routing Logic (Маршрутизация)

- ☐ У всех шлюзов **XOR** задан **Default Flow** (страховка от Incident)?
- ☐ За **Event-based** шлюзом следуют **ТОЛЬКО** ловящие события?
- ☐ У **Event-based** шлюза есть ветка с **Timer** (страховка от зависания)?

Technical Contract (ИТ-Контракт)

- ☐ Для **Service Task** описаны **JSON**-входы, выходы и обработка **HTTP 5xx**?
- ☐ Различаются ли прерывающие (**Interrupting**) и непрерывающие (**Non-interrupting**) **Boundary Events**?

Идеальная BPMN-модель — это не просто рисунок. Это исполняемый код, готовый к запуску в BPM-движке.