

# Complex event flows in distributed systems

@berndruecker



3 common hypotheses I check today:

# Events decrease coupling

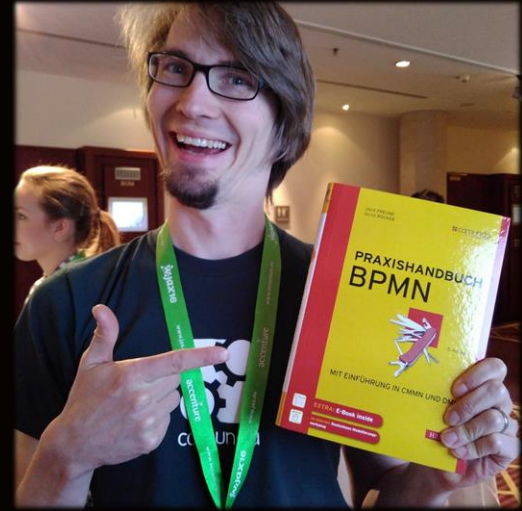
# orchestration needs to be avoided

# Workflow engines are painful

**Warning:  
Contains Opinion**



Bernd Ruecker  
Co-founder and  
Chief Technologist of  
Camunda



Berlin, Germany



[mail@berndruecker.io](mailto:mail@berndruecker.io)  
[@berndruecker](https://twitter.com/berndruecker)

ORCHESTRATING A  
HIGHLY-SCALABLE  
FULFILLMENT  
PROCESS

JÖRN HORSTMANN  
LUCAS NIEMEIER

2017-09-19



**THE SHUTTLE**  
TECH INNOVATION LAB

# Simplified example: dash button



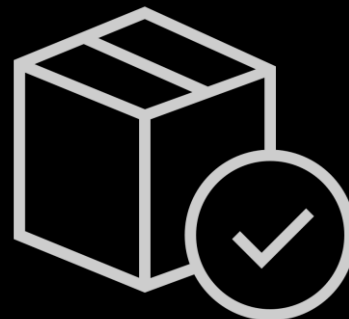
Three steps...



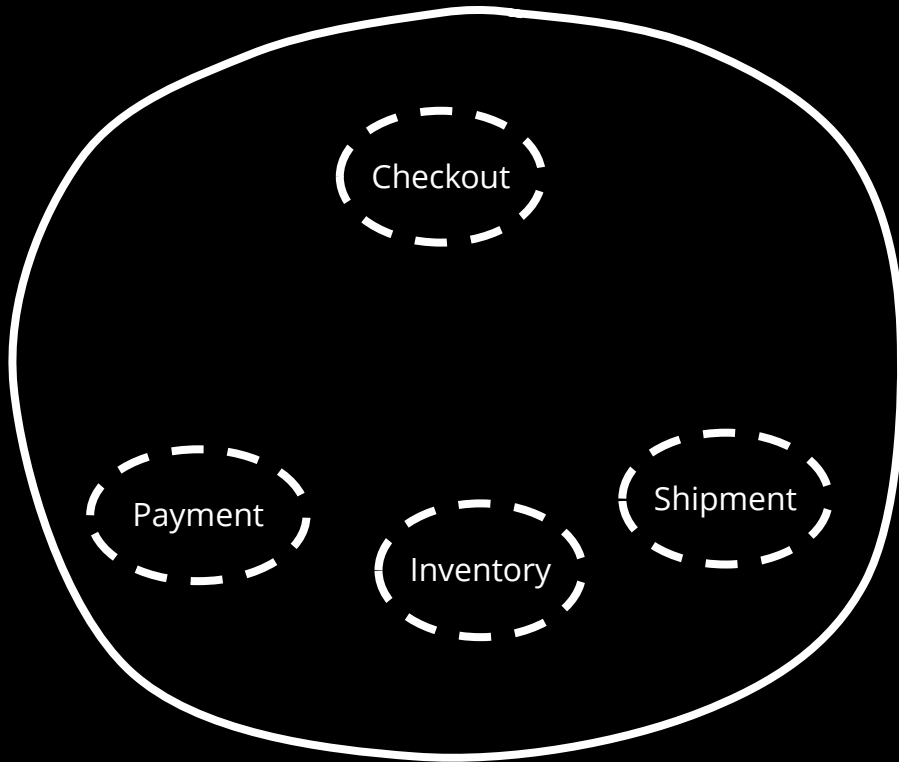
*Pay item*

*Fetch item*

*Ship item*

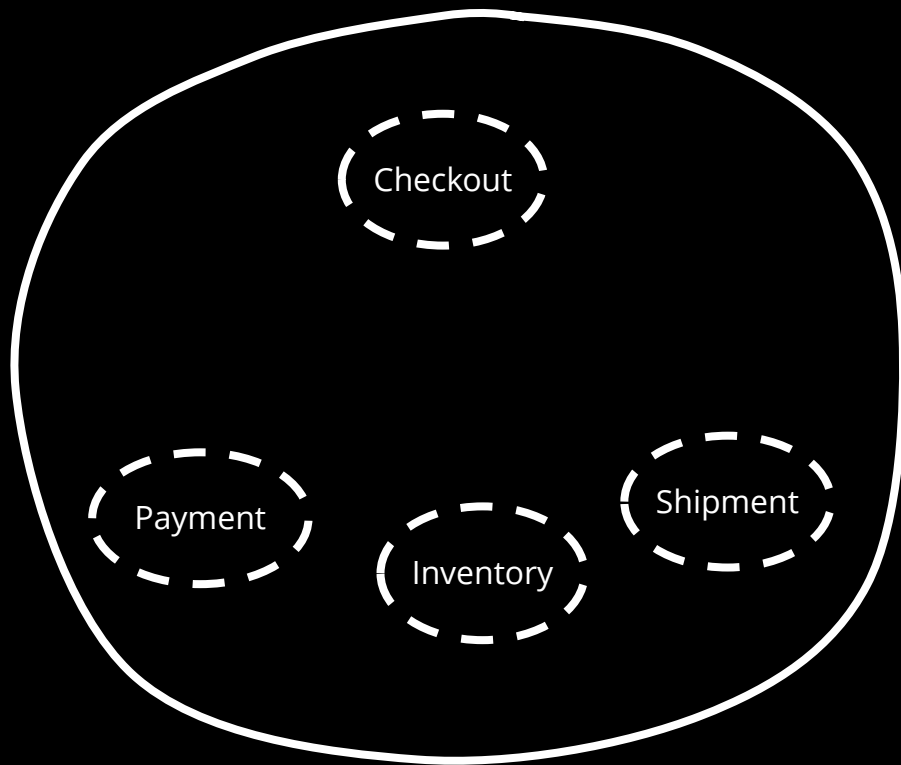


Who is involved? Some bounded contexts...

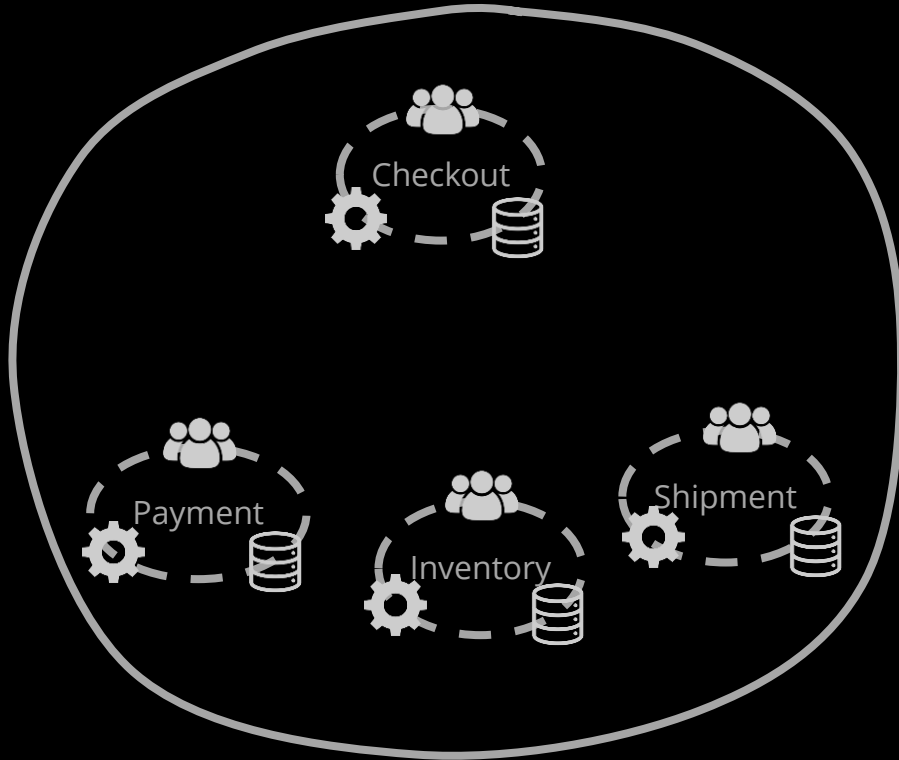




# (Micro-)services



# Autonomous (micro-)services



Dedicated Application Processes



Dedicated infrastructure

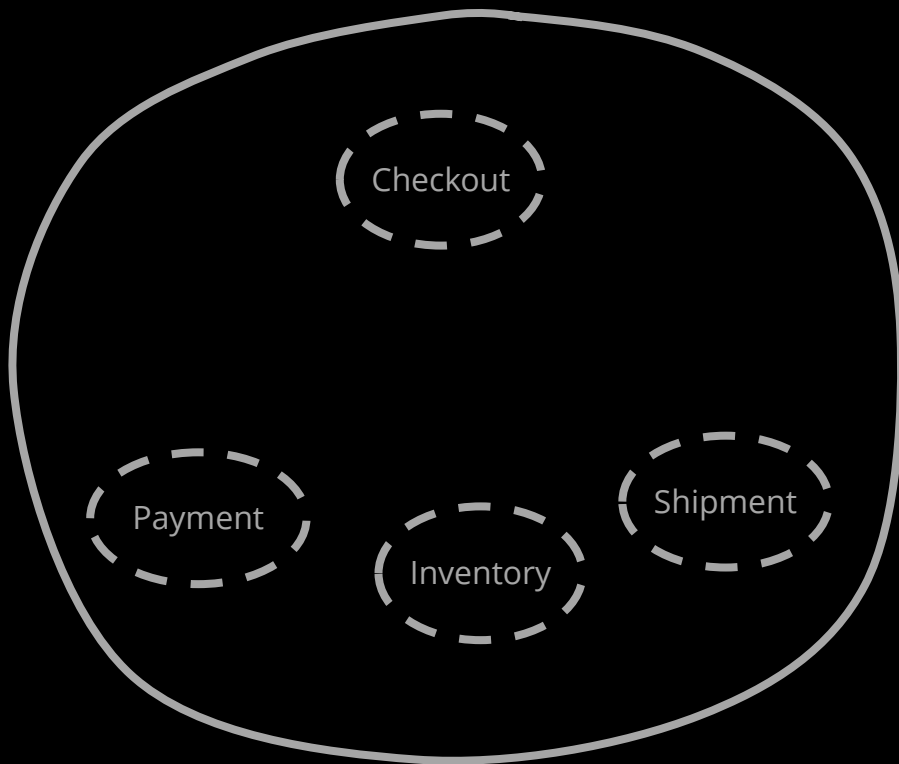


Dedicated Development Teams



Events decrease coupling

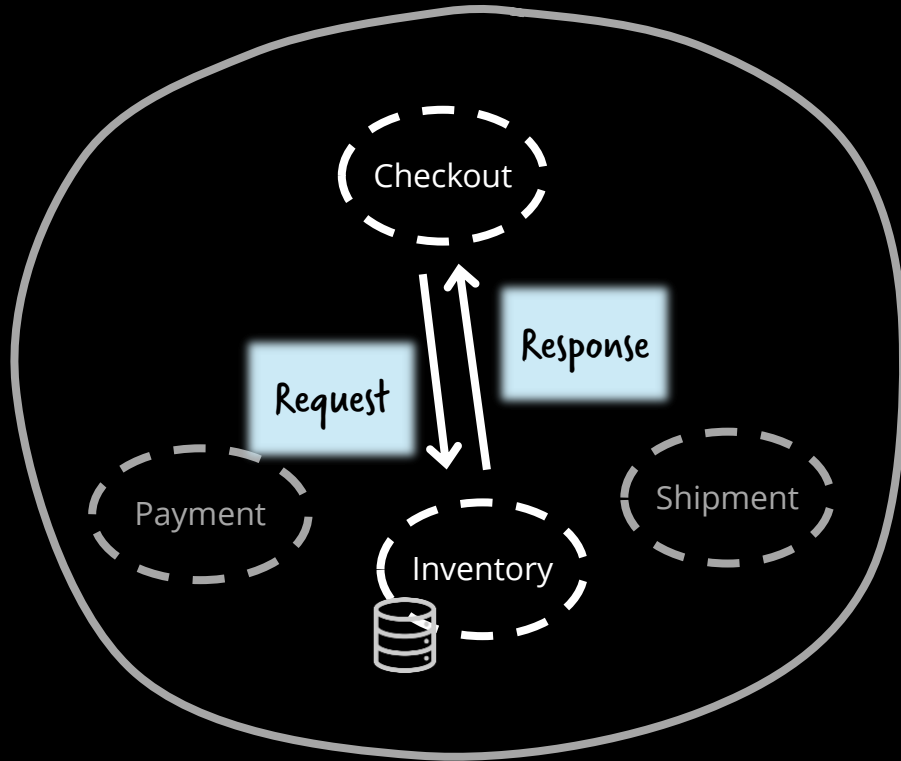
# Example



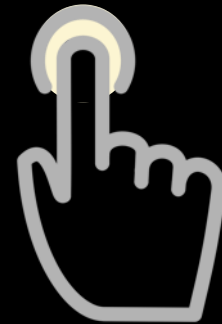
The button blinks if we can ship within 24 hours



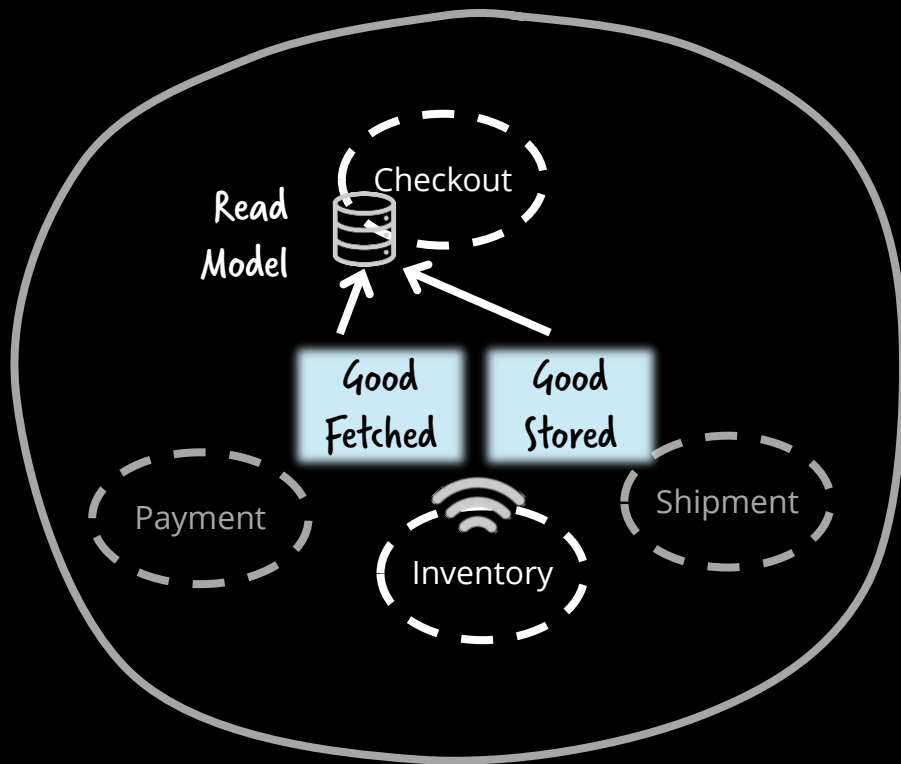
# Request/response: temporal coupling



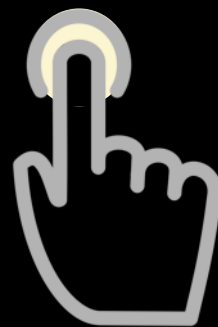
The button blinks if we can ship within 24 hours



# Temporal decoupling with events and read models

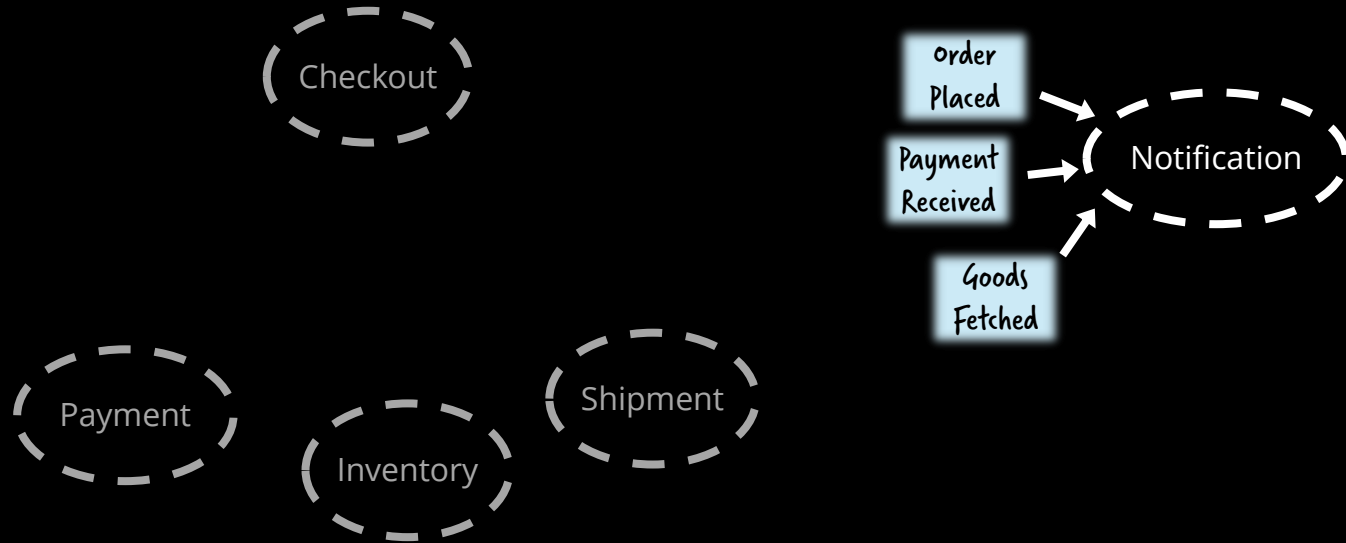


The button blinks if we can ship within 24 hours



**\*Events** are facts about what happened (in the past)

# Event-driven architecture

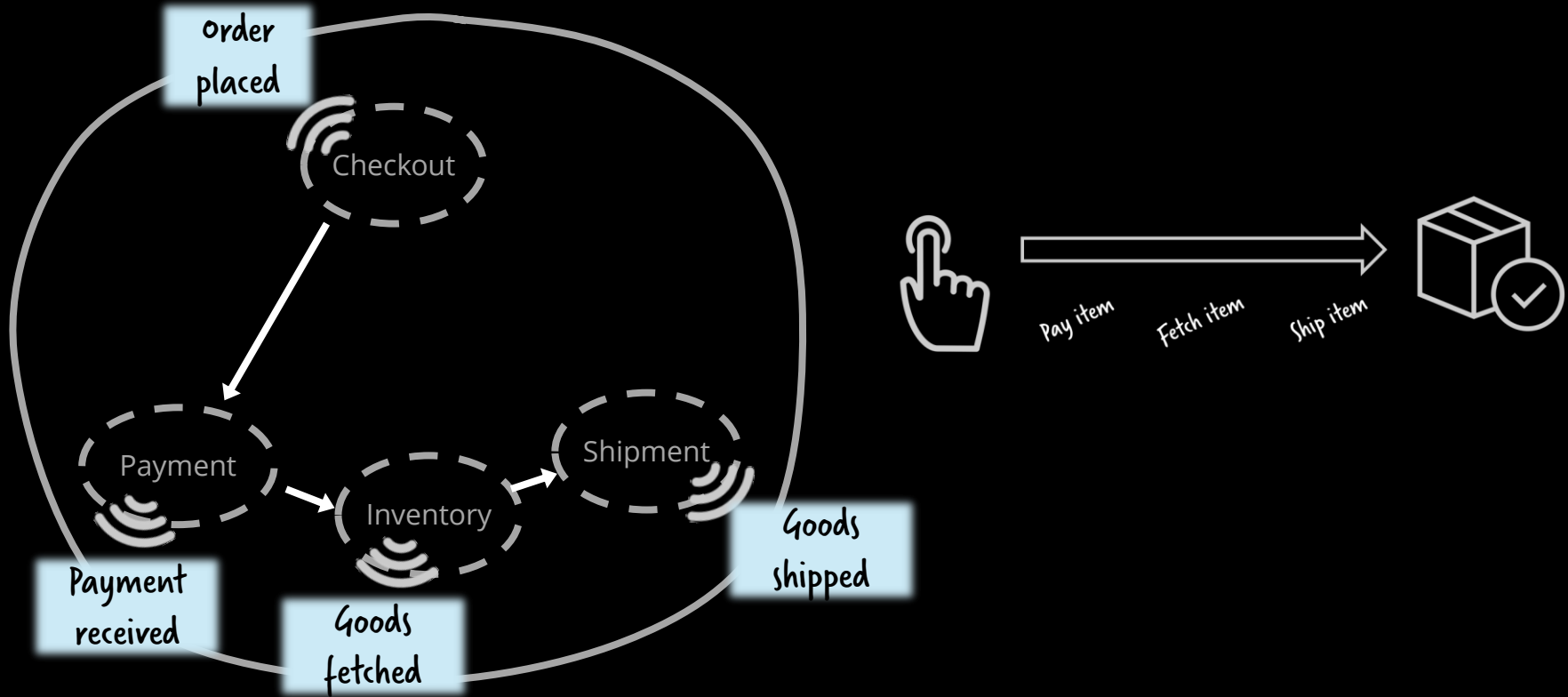


# Events can decrease coupling\*

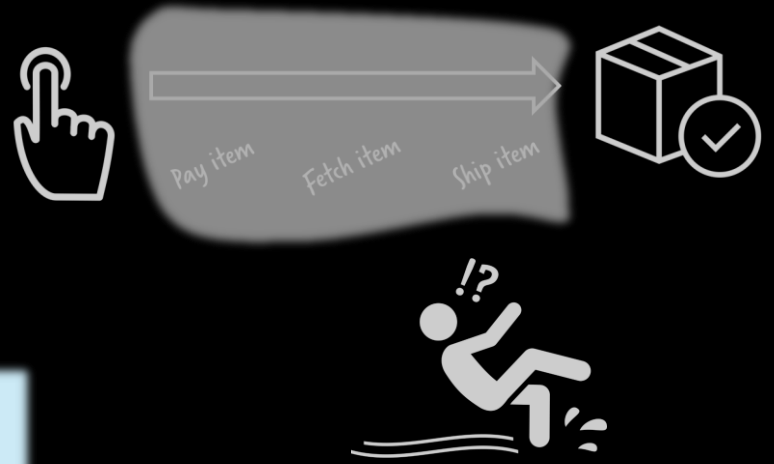
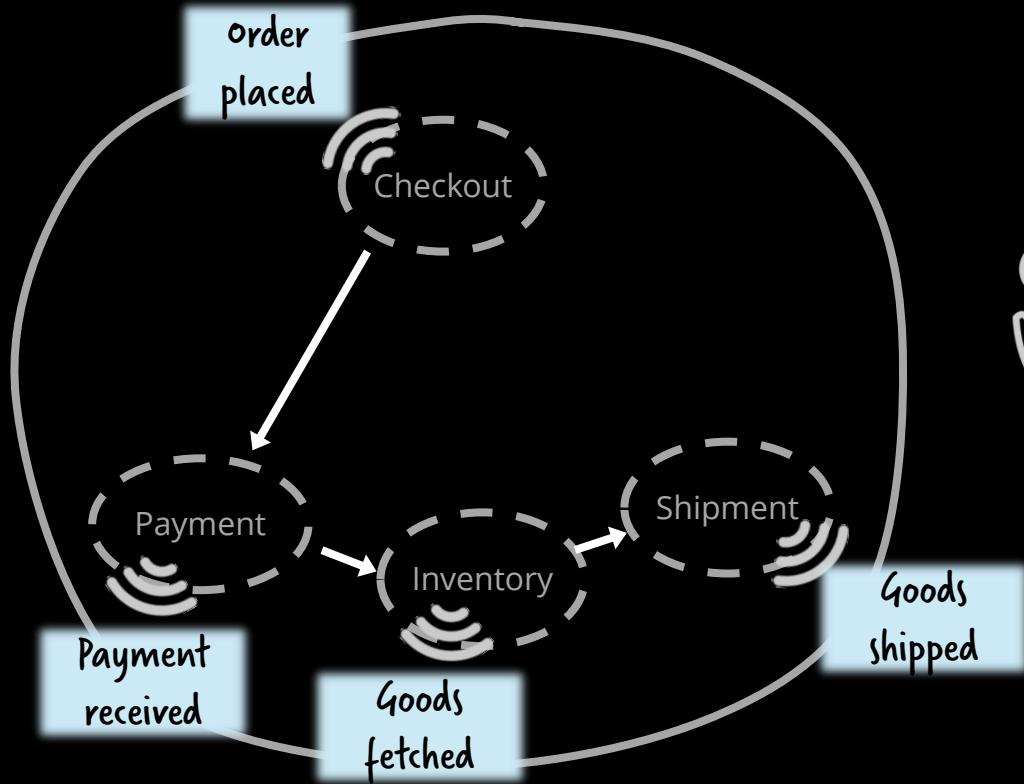
\*e.g. decentral data-management, read models,  
extract cross-cutting aspects



# Peer-to-peer event chains



# Peer-to-peer event chains





The danger is that it's very easy to make nicely decoupled systems with event notification, without realizing that you're losing sight of that larger-scale flow, and thus set yourself up for trouble in future years.



The danger is that it's very easy to make nicely decoupled systems with event notification, without realizing that you're losing sight of that larger-scale flow, and thus set yourself up for trouble in future years.



The danger is that it's very easy to make nicely decoupled systems with event notification, without realizing that you're losing sight of that larger-scale flow, and thus set yourself up for trouble in future years.

# Monitoring Workflows Across Microservices

The screenshot shows the InfoQ website interface. At the top, there's a navigation bar with categories like 'Development', 'Architecture & Design', 'AI, ML and Data Engineering', 'Culture & Methods', and 'DevOps'. A search bar and a user profile for 'BERND' are also visible. Below the navigation, there's a 'FEATURED' section with tags like 'Streaming', 'Machine Learning', 'Reactive', 'Microservices', 'Containers', and 'NoSQL'. The main article title is 'Monitoring and Managing Workflows Across Collaborating Microservices' under the 'ARCHITECTURE & DESIGN' category. The author is Bernd Rucker, and it was reviewed by Daniel Bryant. The article includes a 'Key Takeaways' section with three bullet points. On the right, there's a 'RELATED CONTENT' section with three related articles.

Facilitating The Spread Of Knowledge And Innovation In Professional Software Development | More

**InfoQ** Development Architecture & Design AI, ML and Data Engineering Culture & Methods DevOps

En | 中文 | 日本 | Fr | Br  
714,642 Feb unique visitors

Search

BERND

NEW Videos with Transcripts

QCon Software Dev Conference  
London Mar 4-8  
QCon.ai SF Apr 15-17  
New York Jun 24-28

QCon is Hiring! Conf. Chair & Community Advocate (Remote, Full-Time)

InfoQ Homepage > Articles > Monitoring And Managing Workflows Across Collaborating Microservices

ARCHITECTURE & DESIGN

## Monitoring and Managing Workflows Across Collaborating Microservices

LIKE 1 BOOKMARKS

FEB 28, 2019 • 13 MIN READ

by Bernd Rucker [FOLLOW](#)

reviewed by Daniel Bryant [FOLLOW](#)

### Key Takeaways

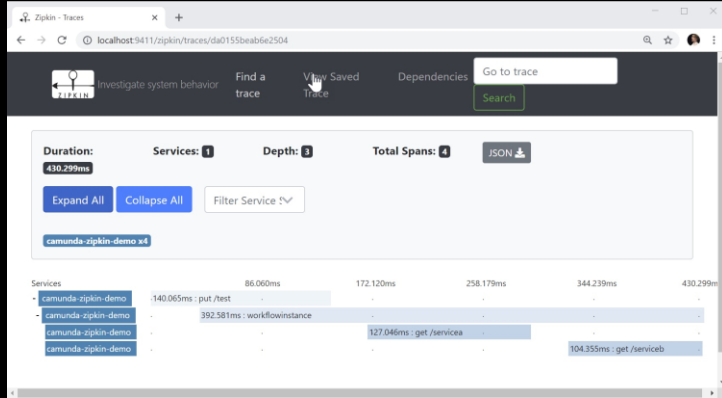
- Peer-to-peer communication between components can lead to emergent behavior, which is challenging for developers, operators and business analysts to understand.
- You need to make sure to have the overview of all of the backwards-and-forwards communication that is going on in order to fulfill a business capability.
- Solutions that provide an overview range from distributed tracing, which typically misses the business perspective; data lakes, which require some effort to tune to what you need to know; process tracking, where you have to model a workflow for the tracking; process mining, which can discover the workflow; all the way through to orchestration, which comes with visibility built in.

### RELATED CONTENT

Experiences Moving from Microservices to Workflows at Jet.com  
FEB 14, 2019

Debugging Microservices Running in Containers: Tooling Review at KubeCon NA  
FEB 24, 2019

Using Contract Testing for Applications with Microservices  
FEB 21, 2019



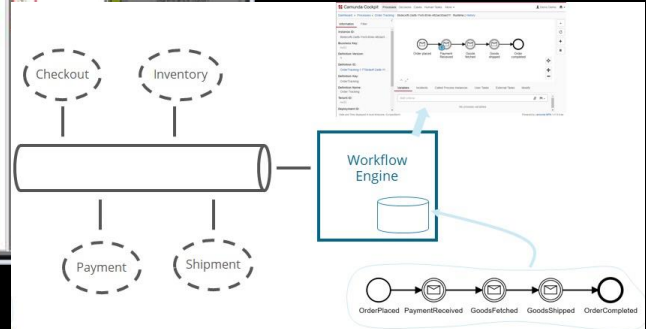
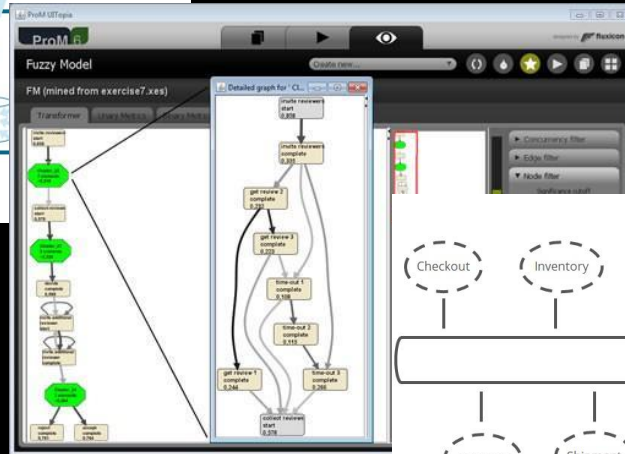
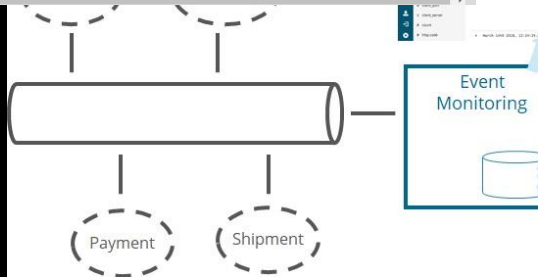
Distributed Tracing

Typical approaches

Data Lake / Event Monitoring

Process Mining

Process Tracking

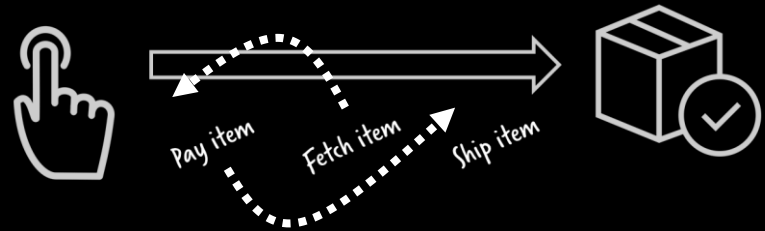
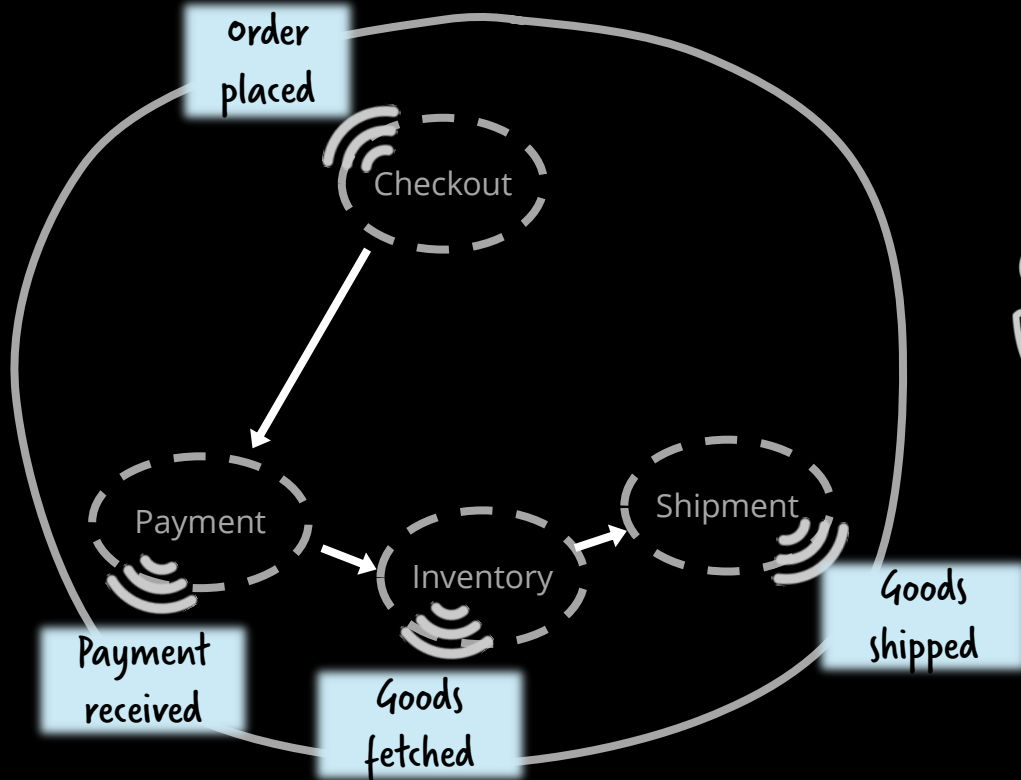


- **Emerging behavior (a.k.a. "what the hell just happened?")**





# Peer-to-peer event chains



# Peer-to-peer event chains

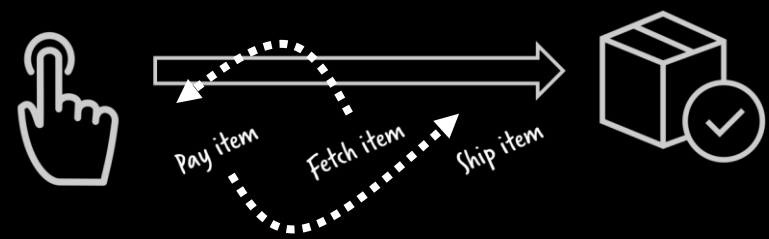
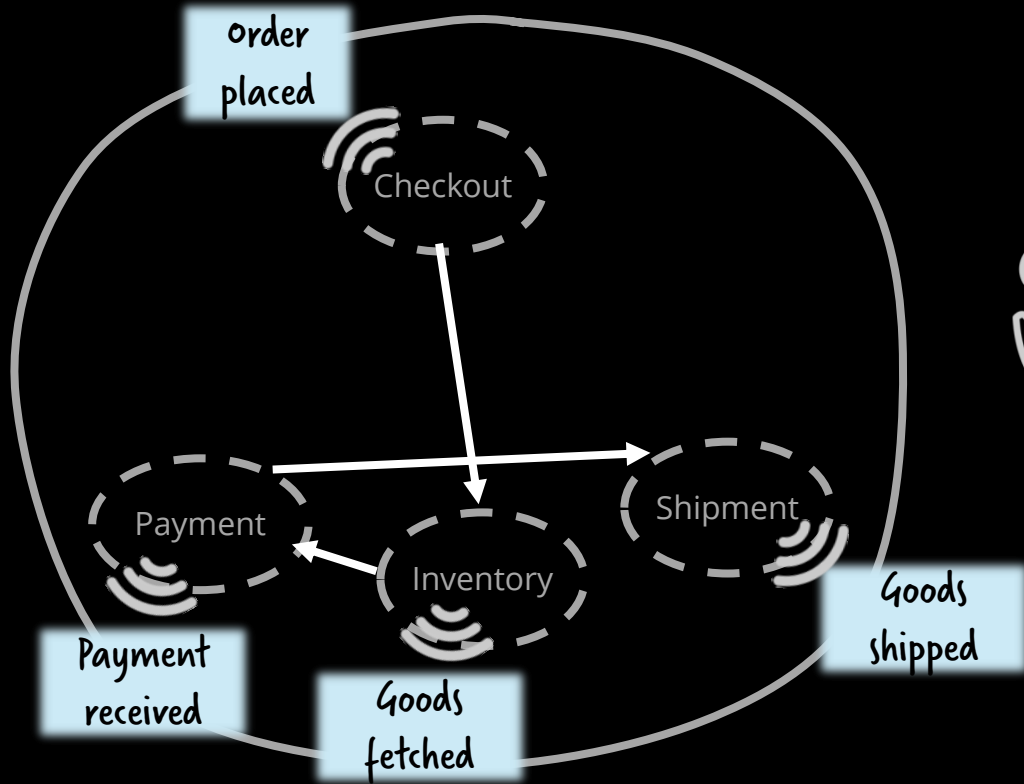




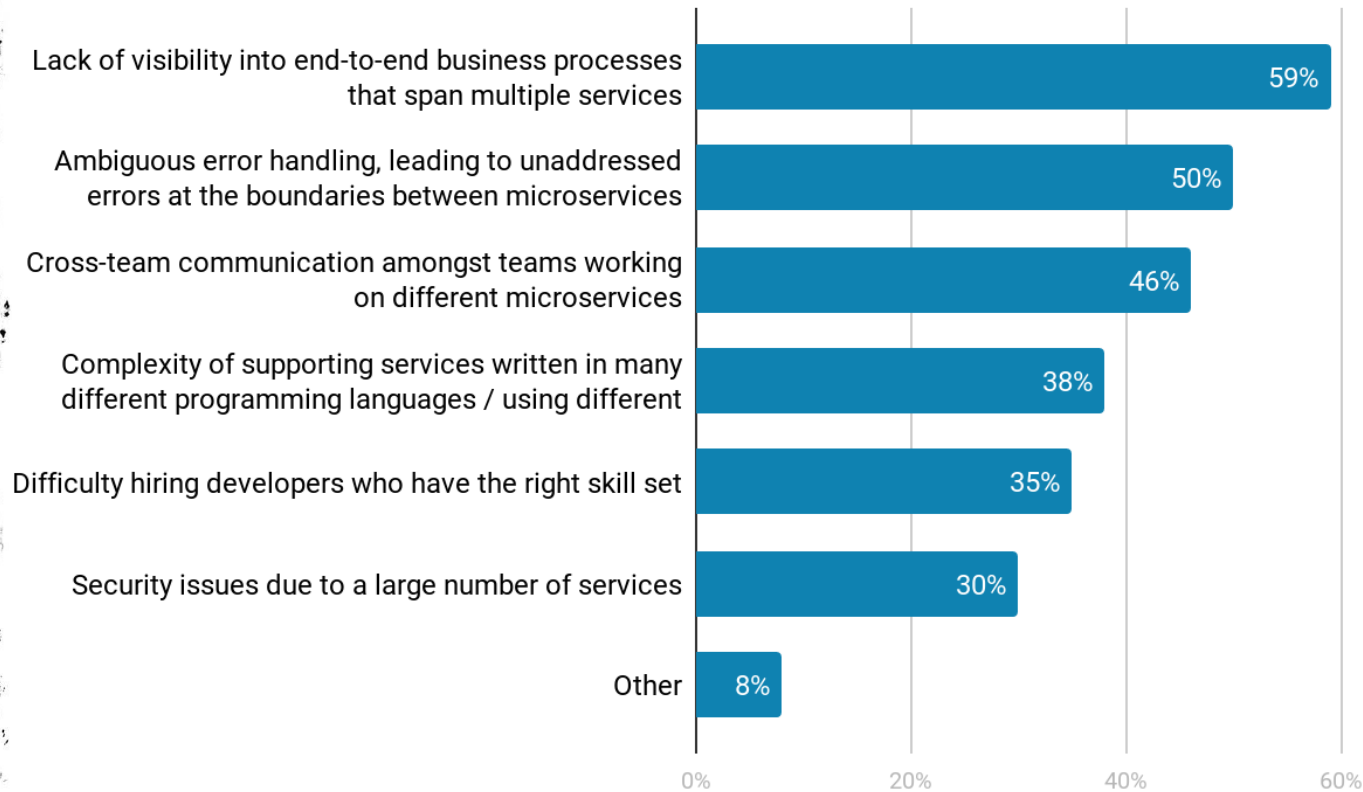
Photo by born1945, available under [Creative Commons BY 2.0 license](https://creativecommons.org/licenses/by/2.0/).

What we wanted



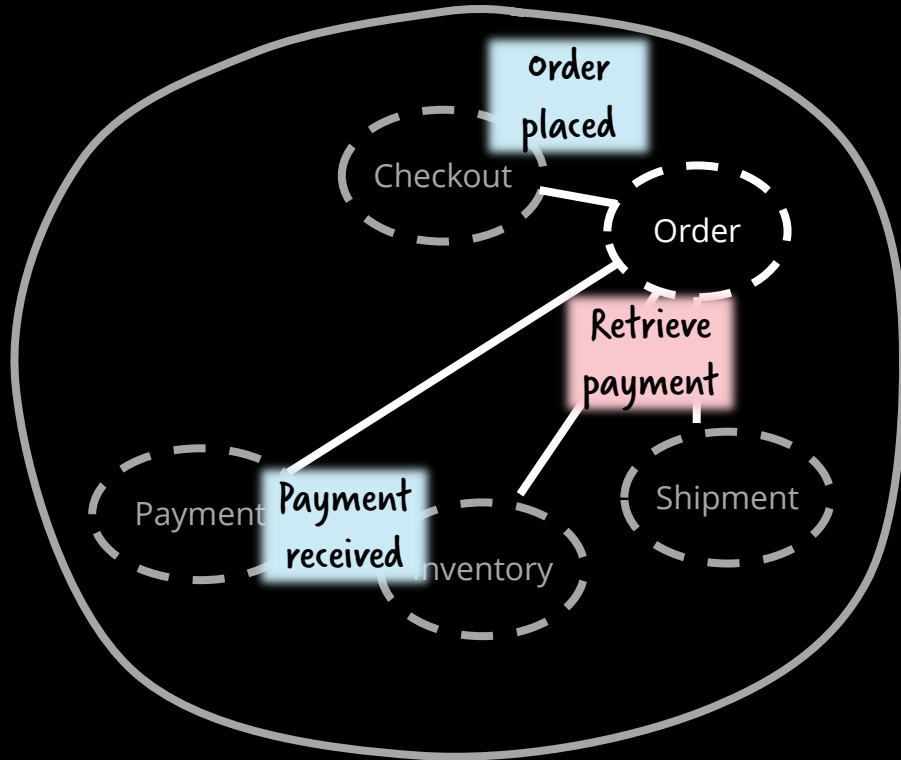
vs. what we got

# „Challenges?“



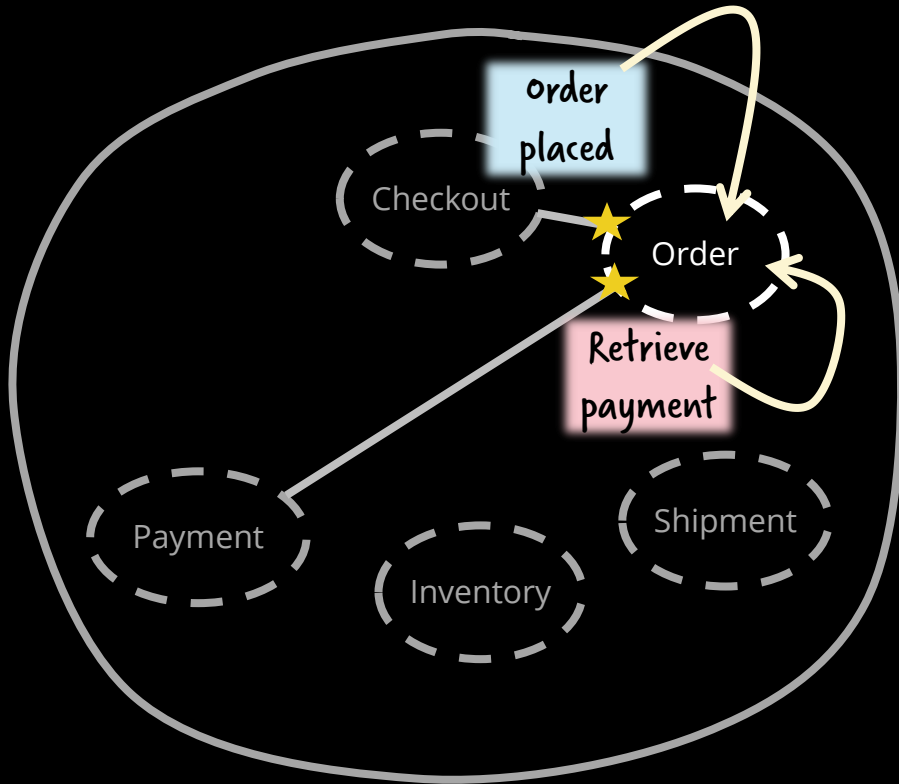
Source:  
Microservices orchestration survey,  
July 2018, 354 responses

# Extract the end-to-end responsibility



**\*Commands** have an intent about what needs to happen in the future

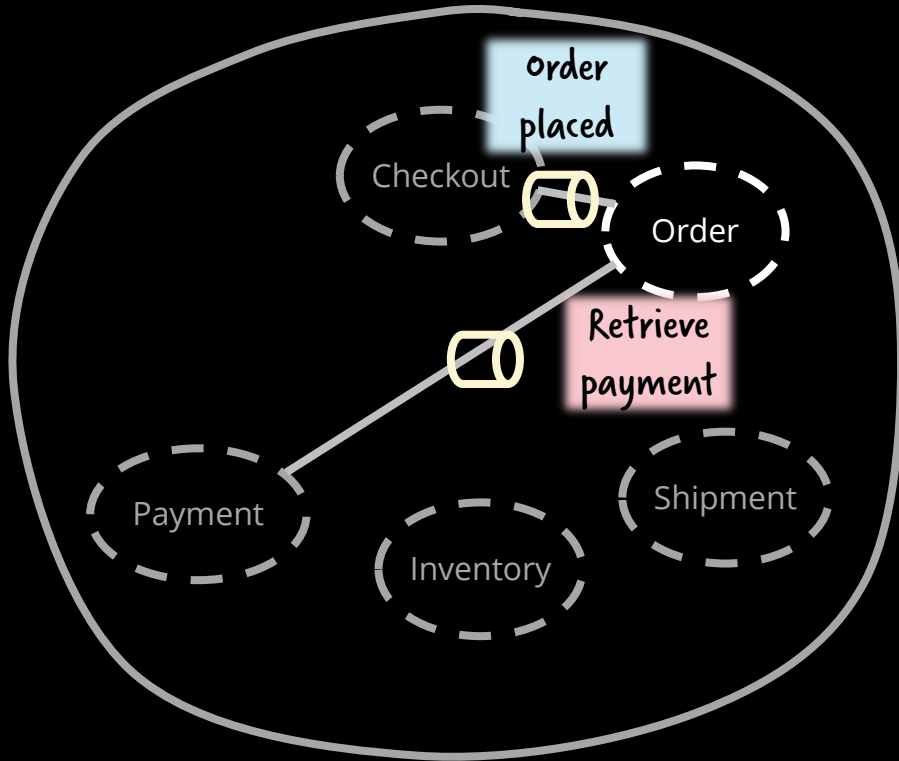
It is about where to decide about the coupling!



order decides

- . to listen to the event
- . to issue the command

It is about where to decide about the coupling!



It can still be messaging!

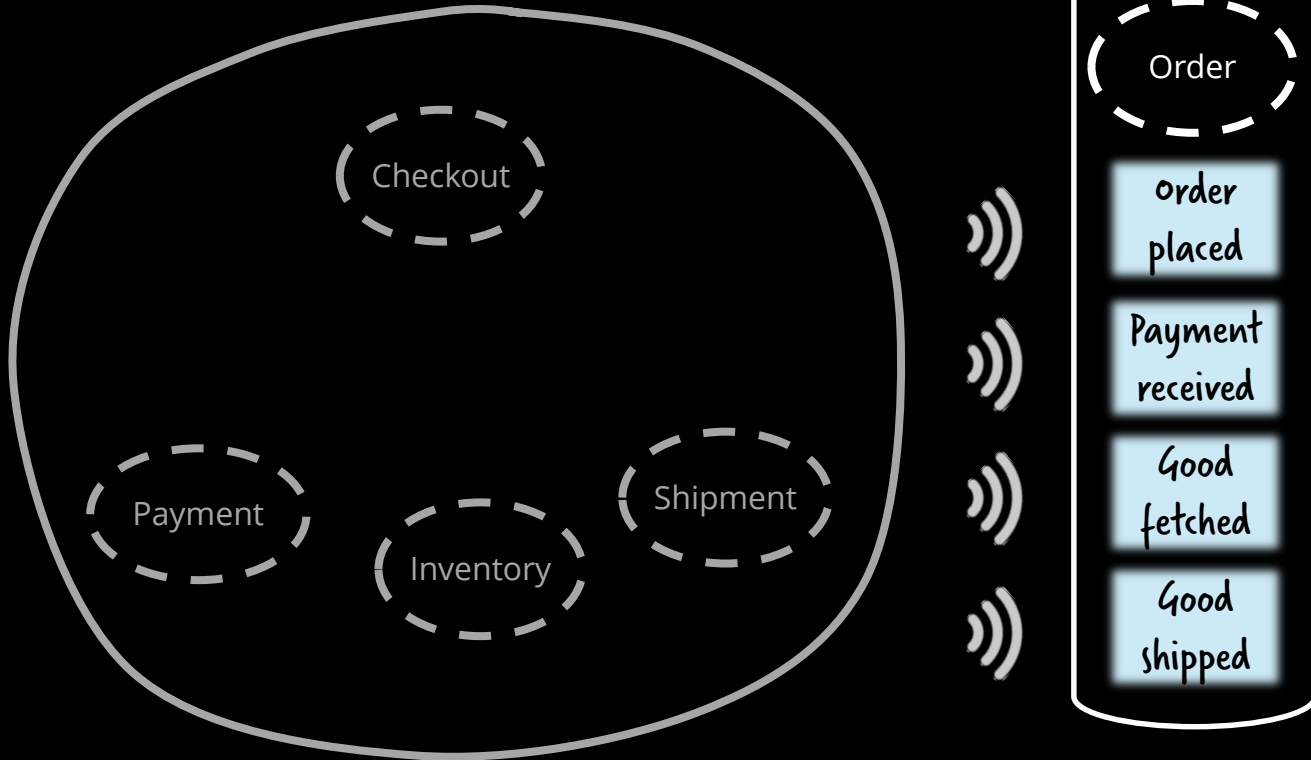


Commands help to avoid (complex)  
peer-to-peer event chains

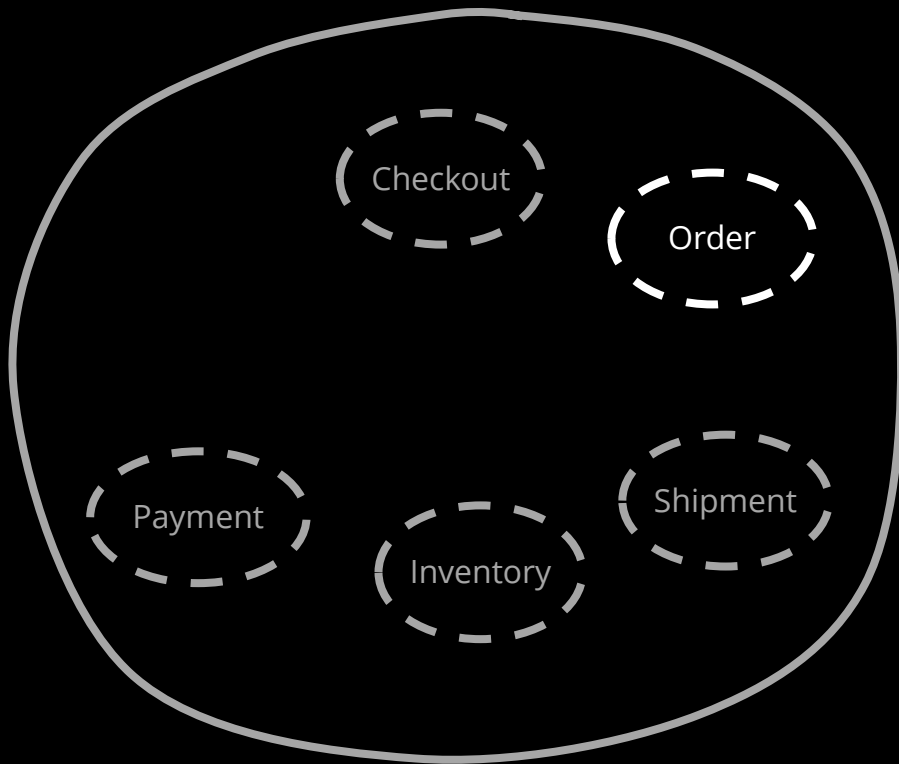


orchestration needs to be avoided

# Smart ESB-like middleware



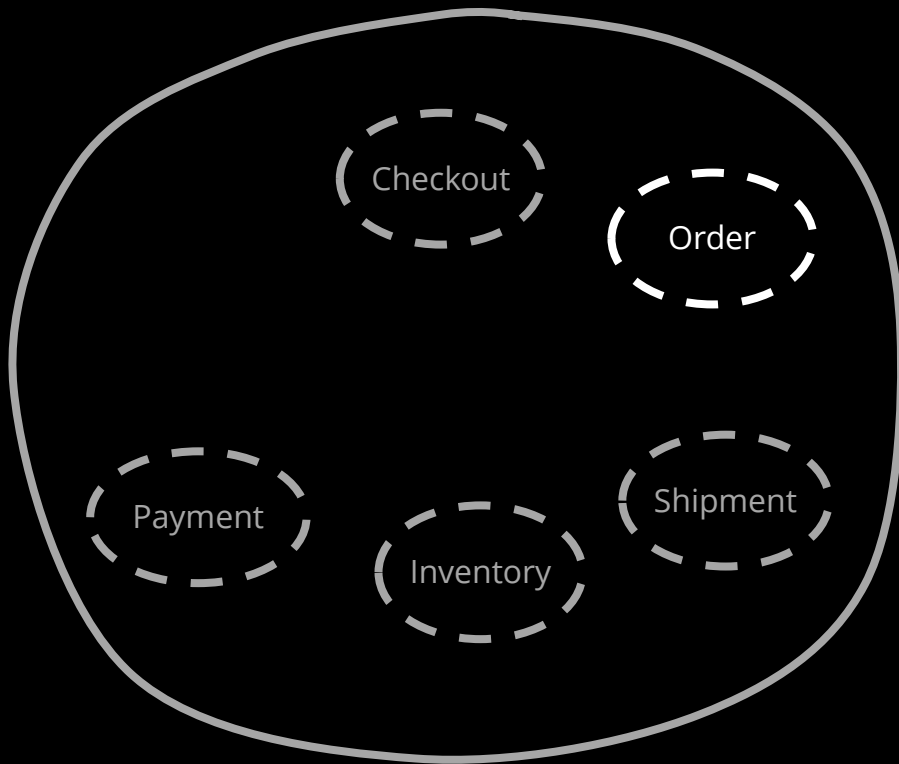
# Dumb pipes



Martin Fowler

Smart endpoints  
and **dumb pipes**

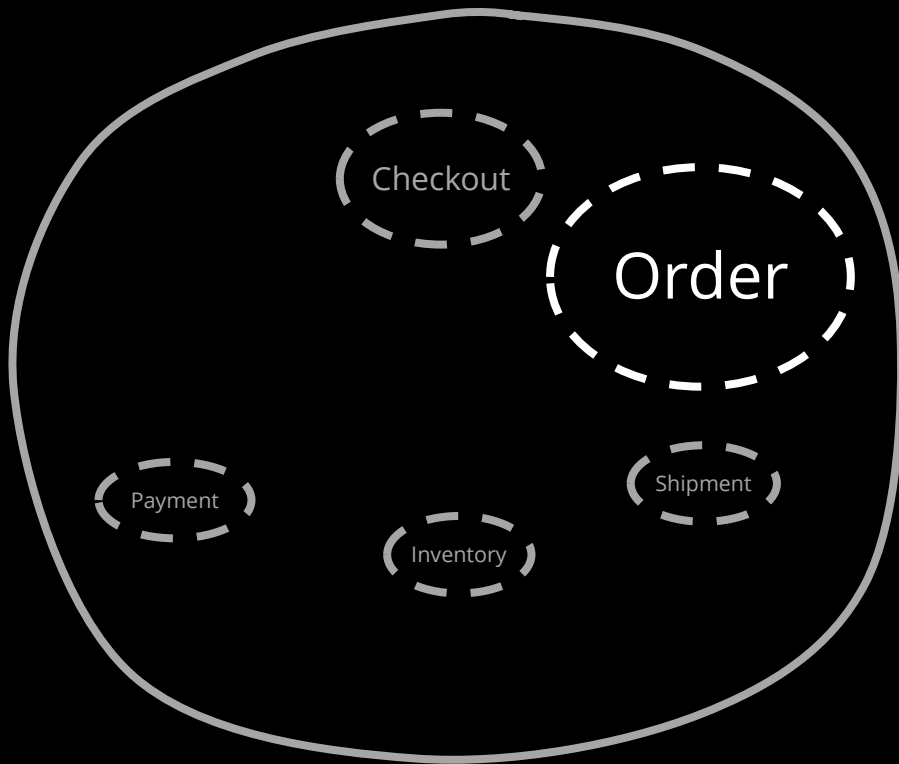
# Danger of god services?



Sam Newmann

A few smart god services tell anemic CRUD services what to do

# Danger of god services?



Sam Newmann

A few  
smart god services  
tell  
anemic CRUD services  
what to do

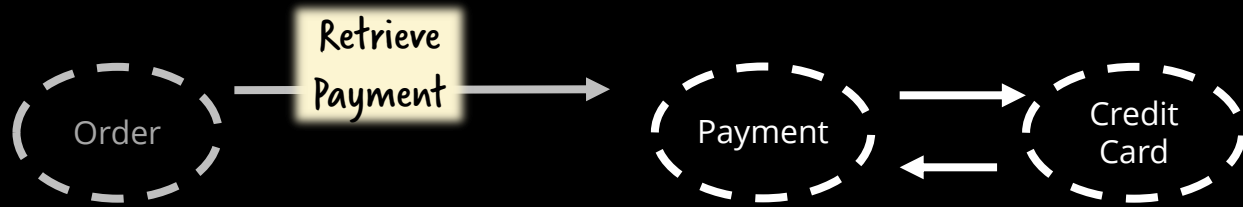
A god service is only created  
by bad API design!

# Example

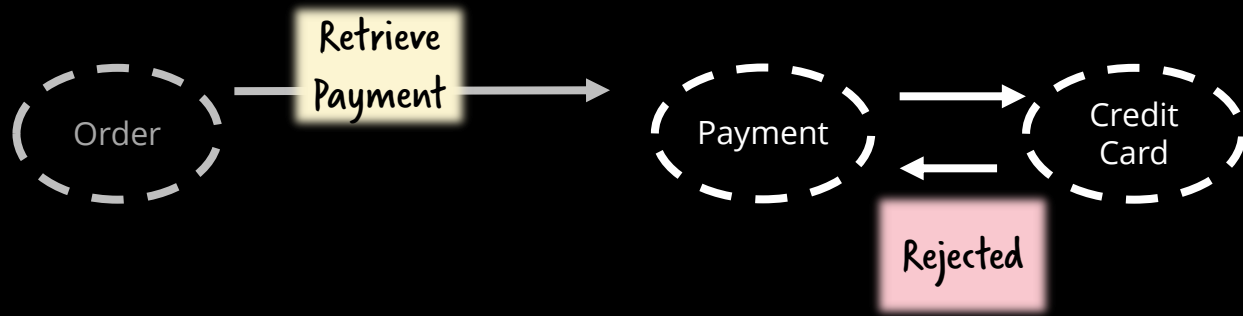




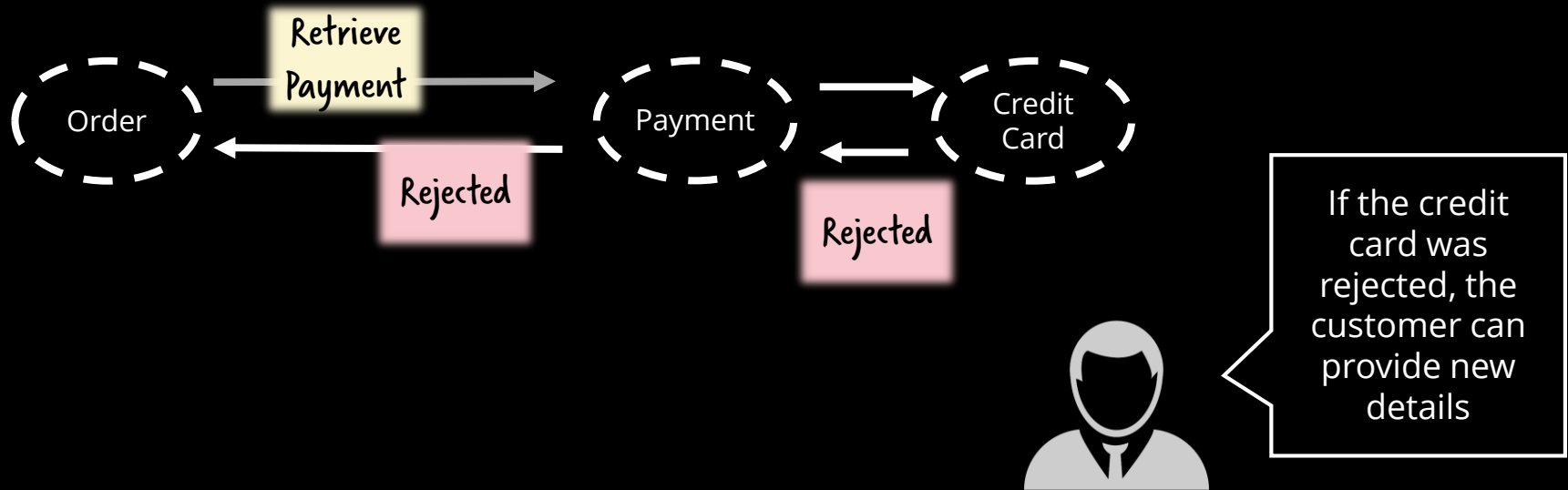
# Example



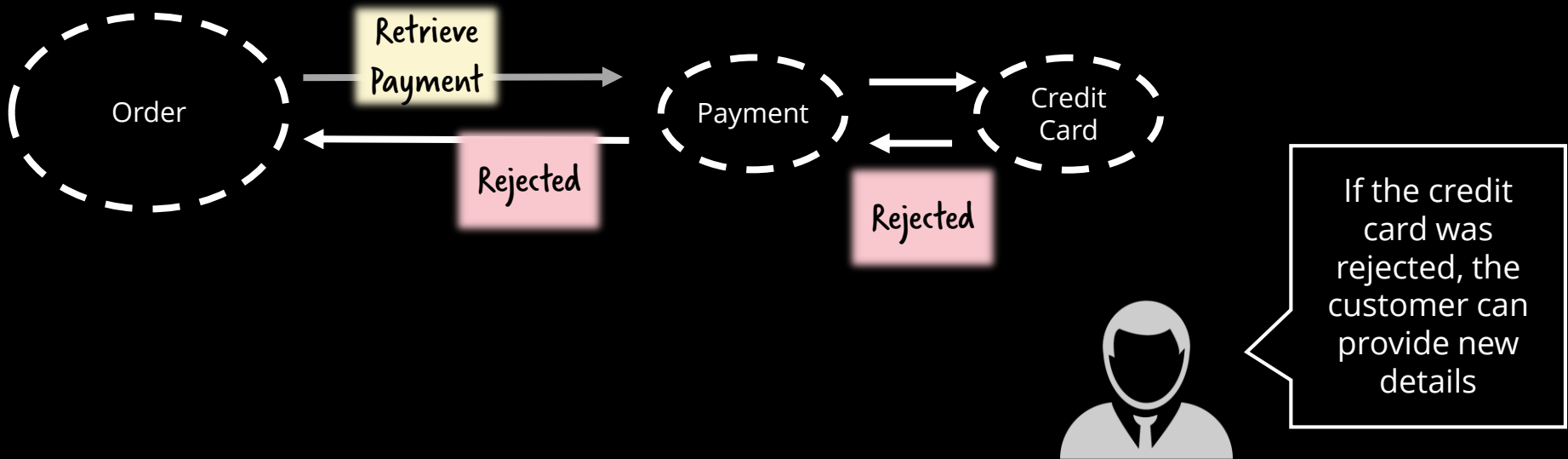
# Example



# Example

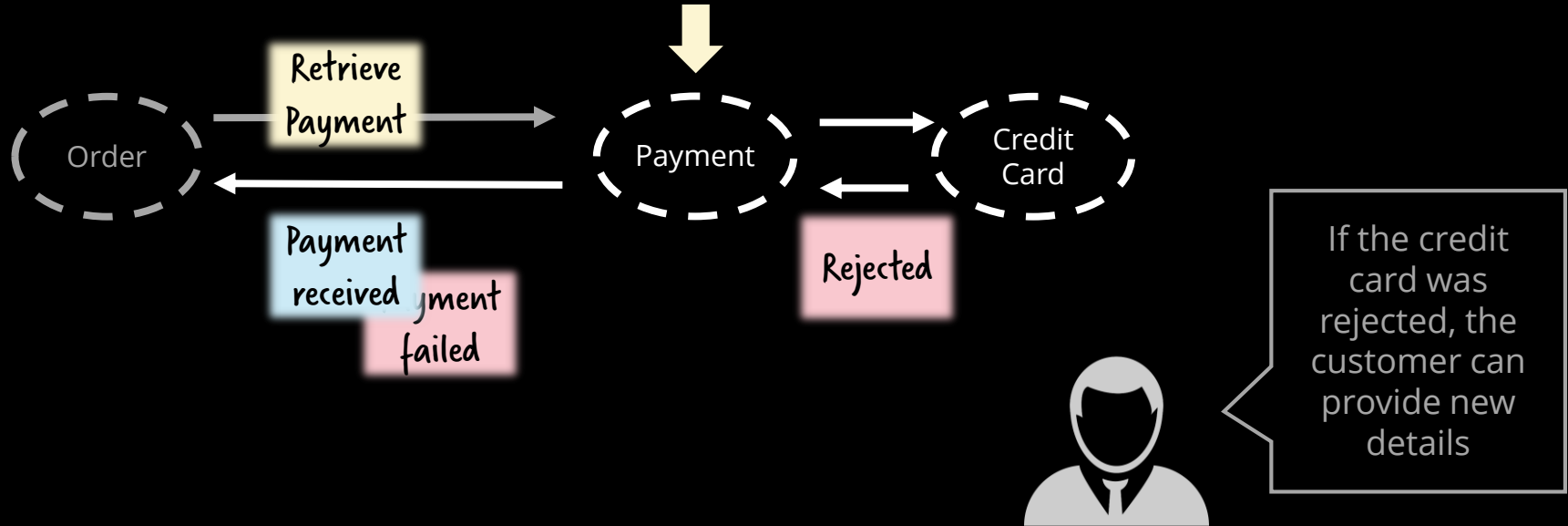


# Example

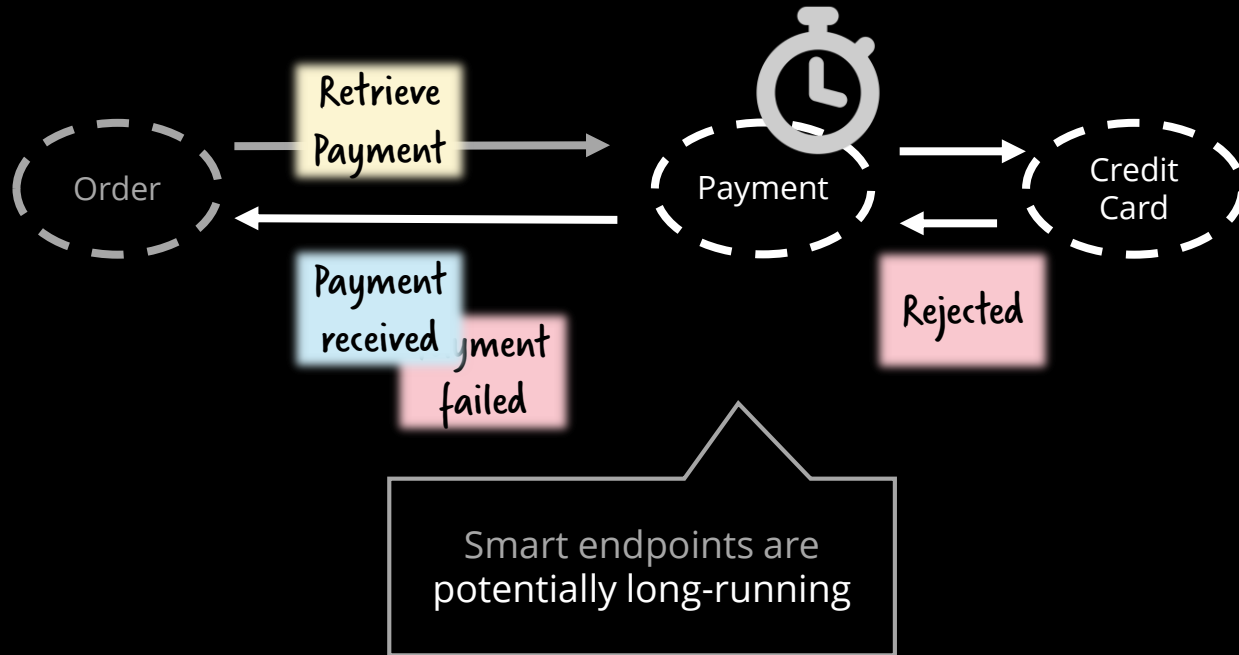


Client of **dumb endpoints** easily become a god services.

# Who is responsible to deal with problems?



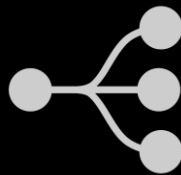
# Long running services



# Handling State



Persist thing  
(Entity, Actor, ...)



State machine or  
workflow engine

Typical  
concerns

DIY = effort,  
accidental  
complexity



Scheduling, Versioning,  
operating, visibility,  
scalability, ...



# Workflow engines are painful

~~Complex, proprietary, heavyweight, central, developer adverse, ...~~



# Avoid the wrong tools!

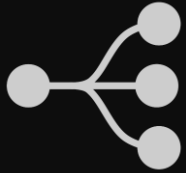


Low-code is great!  
(You can get rid  
of your developers!)



Death by properties panel

Complex, proprietary, heavyweight, central, developer adverse, ...

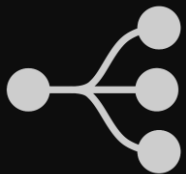


Workflow engines,  
state machines



AWS Step  
Functions

It is  
**relevant**  
in modern  
architectures

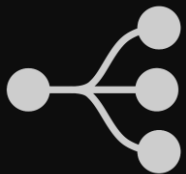


Workflow engines,  
state machines



Silicon valley  
has recognized





Workflow engines,  
state machines

UBER CADENCE

There are  
lightweight open source  
options



```
public static void main(String[] args) {
    ProcessEngine engine = new StandaloneInMemProcessEngineConfiguration()
        .buildProcessEngine();

    engine.getRepositoryService().createDeployment() //
        .addModelInstance("flow.bpmn", Bpmn.createExecutableProcess("flow") //
            .startEvent()
            .serviceTask("Step 1")
            .serviceTask("Step 2")
            .endEvent()
            .done()
        ).deploy();

    engine.getRuntimeService().start("flow", Variables.putValue("city", "New York"));
}

public class SysoutDelegate implements JavaDelegate {
    public void execute(DelegateExecution execution) throws Exception {
        System.out.println("Hello " + execution.getVariable("city"));
    }
}
```

What do I mean by  
„leightweight?“



```
public static void main(String[] args) {
    ProcessEngine engine = new StandaloneInMemProcessEngineConfiguration()
        .buildProcessEngine();

    engine.getRepositoryService().createDeployment() //
        .addModelInstance("flow.bpmn", Bpmn.createExecutableProcess("flow") //
            .startEvent()
            .serviceTask("Step1").camundaClass(SysoutDelegate.class)
            .serviceTask("Step2").camundaClass(SysoutDelegate.class)
            .endEvent()
            .done()
        ).deploy();

    engine.getRuntimeService().startProcessInstanceByKey(
        "flow", Variables.putValue("city", "New York"));
}
public class SysoutDelegate implements JavaDelegate {
    public void execute(DelegateExecution execution) throws Exception {
        System.out.println("Hello " + execution.getVariable("city"));
    }
}
```

Build engine  
in one line of  
code  
(using in-  
memory H2)

```
public static void main(String[] args) {
    ProcessEngine engine = new StandaloneInMemProcessEngineConfiguration()
        .buildProcessEngine();

    engine.getRepositoryService().createDeployment() //
        .addModelInstance("flow.bpmn", Bpmn.createExecutableProcess("flow")
            .startEvent()
            .serviceTask("Step1").camundaClass(SysoutDelegate.class)
            .serviceTask("Step2").camundaClass(SysoutDelegate.class)
            .endEvent()
            .done()
        ).deploy();

    engine.getRuntimeService().startProcessInstanceByKey(
        "flow", Variables.putValue("city", "New York"));
}

public class SysoutDelegate implements JavaDelegate {
    public void execute(DelegateExecution execution) throws Exception {
        System.out.println("Hello " + execution.getVariable("city"));
    }
}
```

Define flow  
e.g. in Java  
DSL

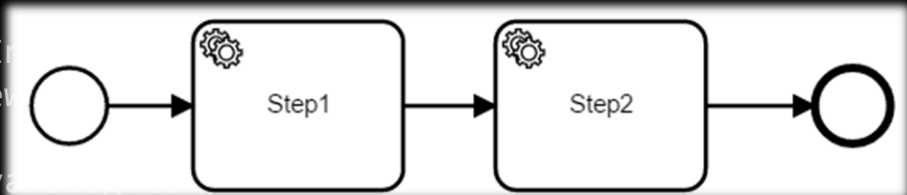
```
public static void main(String[] args) {
    ProcessEngine engine = new StandaloneInMemProcessEngineConfiguration()
        .buildProcessEngine();

    engine.getRepositoryService().createDeployment() //
        .addModelInstance("flow.bpmn", Bpmn.createExecutableProcess("flow")
            .startEvent()
            .serviceTask("Step1").camundaClass(SysoutDelegate.class)
            .serviceTask("Step2").camundaClass(SysoutDelegate.class)
            .endEvent()
            .done()
        ).deploy();

    engine.getRuntimeService().startProcessInstanceByKey(
        "flow", Variables.putValue("city", "New York"));
}

public class SysoutDelegate implements JavaDelegate {
    public void execute(DelegateExecution execution) throws Exception {
        System.out.println("Hello " + execution.getVariable("city"));
    }
}
```

Define flow  
e.g. in Java  
DSL





# BPMN

Business Process  
Model and Notation

ISO Standard



```
public static void main(String[] args) {
    ProcessEngine engine = new StandaloneInMemProcessEngineConfiguration()
        .buildProcessEngine();

    engine.getRepositoryService().createDeployment() //
        .addModelInstance("flow.bpmn", Bpmn.createExecutableProcess("flow")
            .startEvent()
            .serviceTask("Step1").camundaClass(SysoutDelegate.class)
            .serviceTask("Step2").camundaClass(SysoutDelegate.class)
            .endEvent()
            .done()
        ).deploy();

    engine.getRuntimeService().startProcessInstanceByKey(
        "flow", Variables.putValue("city", "New York"));
}

public class SysoutDelegate implements JavaDelegate {
    public void execute(DelegateExecution execution) throws Exception {
        System.out.println("Hello " + execution.getVariable("city"));
    }
}
```

We can attach  
code...

```
public static void main(String[] args) {
    ProcessEngine engine = new StandaloneInMemProcessEngineConfiguration()
        .buildProcessEngine();

    engine.getRepositoryService().createDeployment() //
        .addModelInstance("flow.bpmn", Bpmn.createExecutableProcess("flow")
            .startEvent()
            .serviceTask("Step1").camundaClass(SysoutDelegate.class)
            .serviceTask("Step2").camundaClass(SysoutDelegate.class)
            .endEvent()
            .done()
        ).deploy();

    engine.getRuntimeService().startProcessInstanceByKey(
        "flow", Variables.putValue("city", "New York"));
}

public class SysoutDelegate implements JavaDelegate {
    public void execute(DelegateExecution execution) throws Exception {
        System.out.println("Hello " + execution.getVariable("city"));
    }
}
```

...that is  
called when  
workflow  
instances pass  
through

```
public static void main(String[] args) {
    ProcessEngine engine = new StandaloneInMemProcessEngineConfiguration()
        .buildProcessEngine();

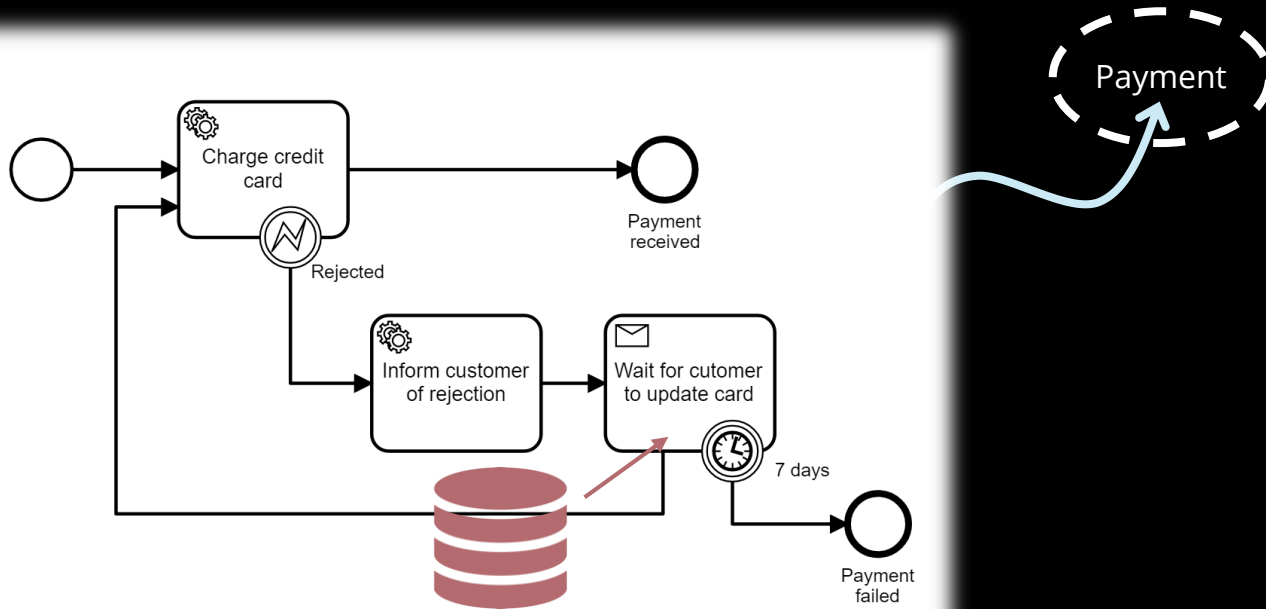
    engine.getRepositoryService().createDeployment() //
        .addModelInstance("flow.bpmn", Bpmn.createExecutableProcess("flow")
            .startEvent()
            .serviceTask("Step1").camundaClass(SysoutDelegate.class)
            .serviceTask("Step2").camundaClass(SysoutDelegate.class)
            .endEvent()
            .done()
        ).deploy();

    engine.getRuntimeService().startProcessInstanceByKey(
        "flow", Variables.putValue("city", "New York"));
}

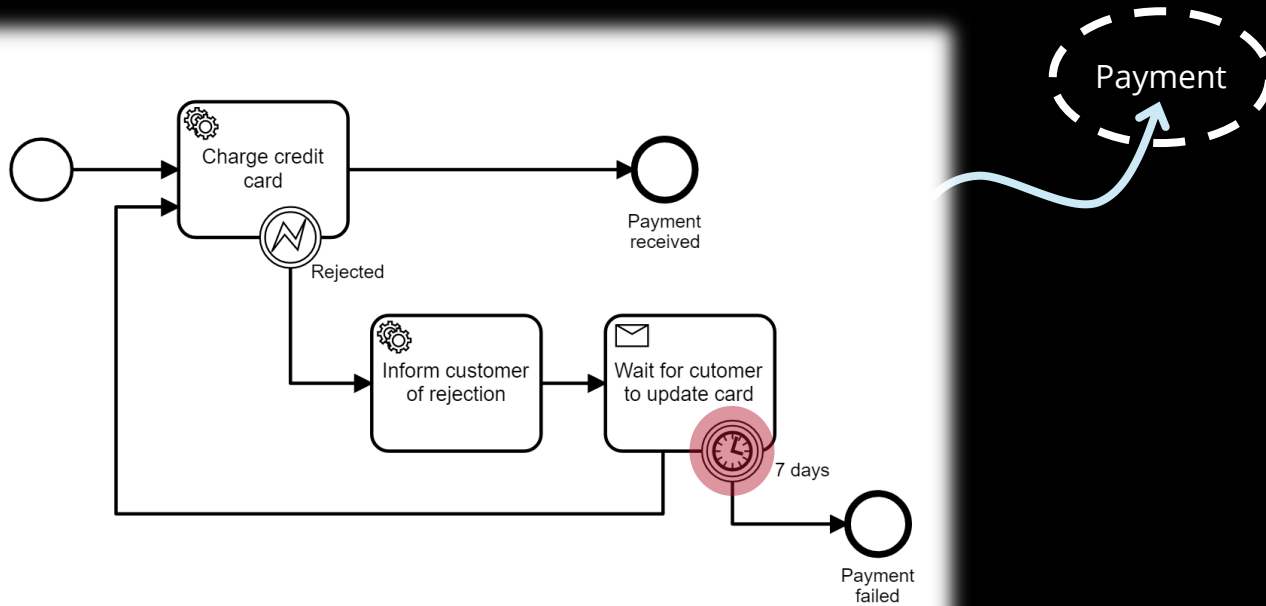
public class SysoutDelegate implements JavaDelegate {
    public void execute(DelegateExecution execution) throws Exception {
        System.out.println("Hello " + execution.getVariable("city"));
    }
}
```

Start  
instances

# Now you have a state machine!



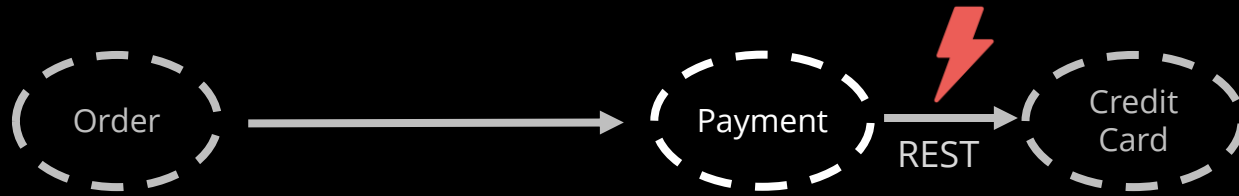
# Easy to handle time



# Distributed systems

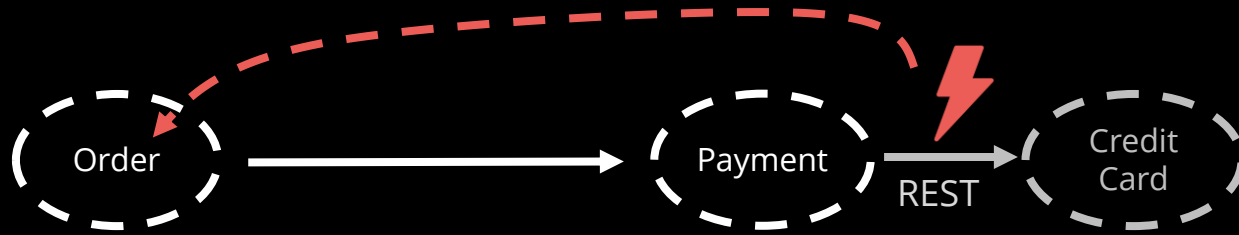


# Example with synchronous communication

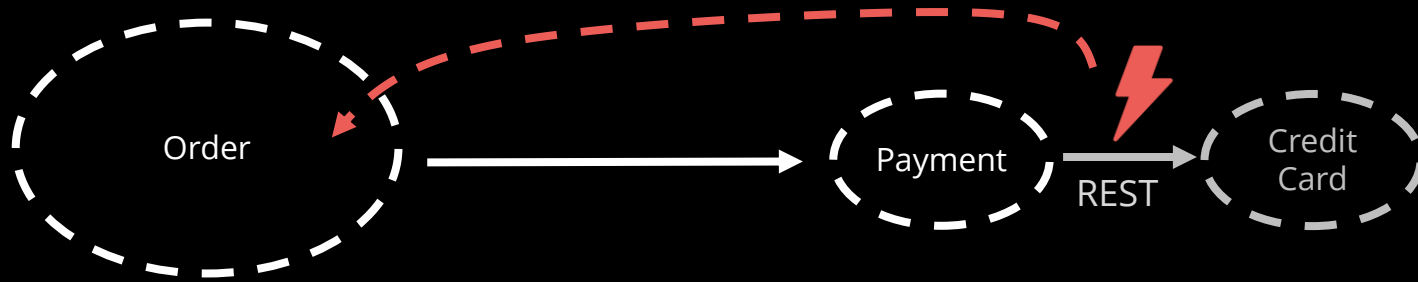




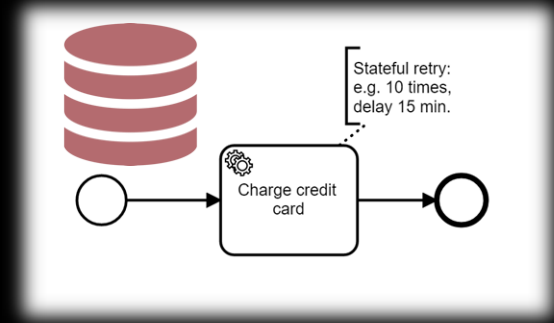
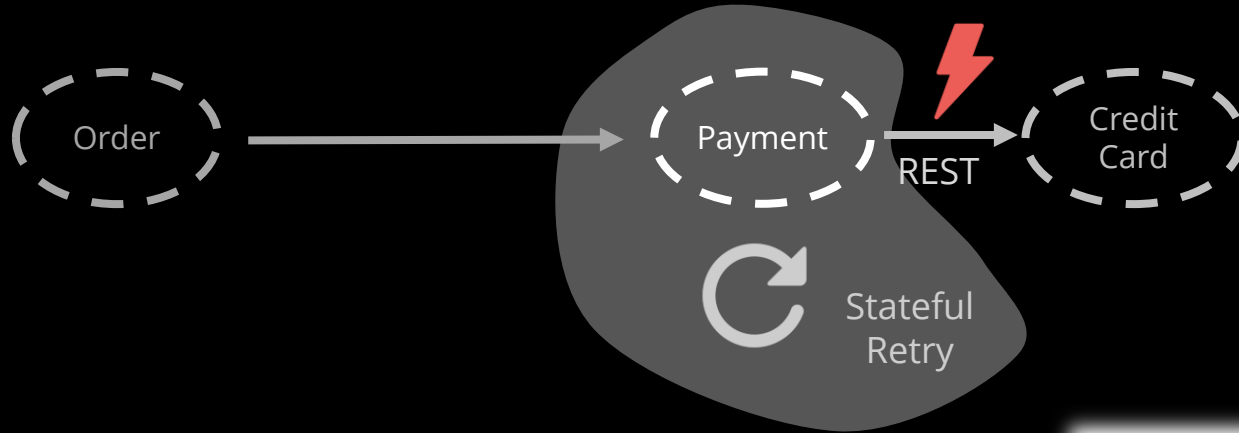
# Example with synchronous communication



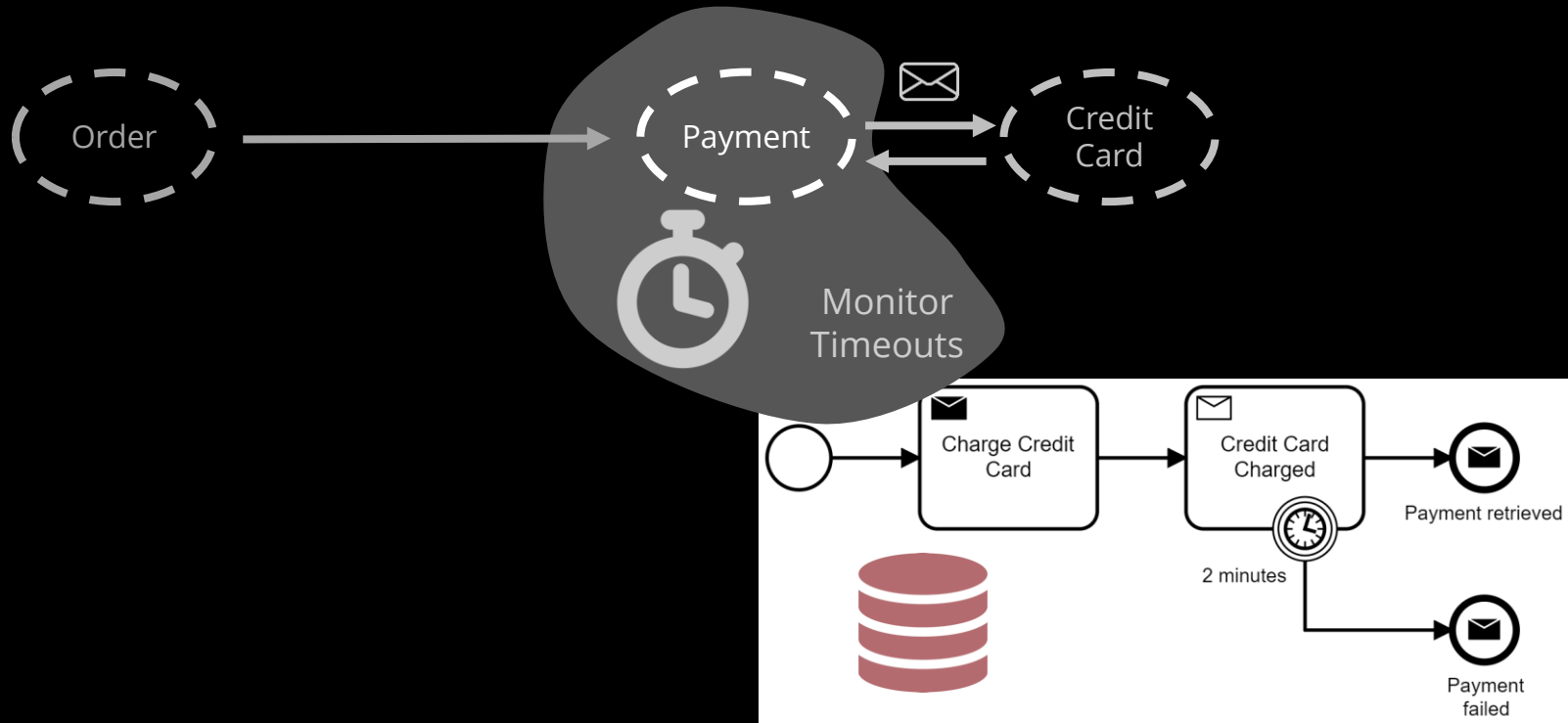
# Example with synchronous communication



# Example with synchronous communication



# Works also for asynchronous communication

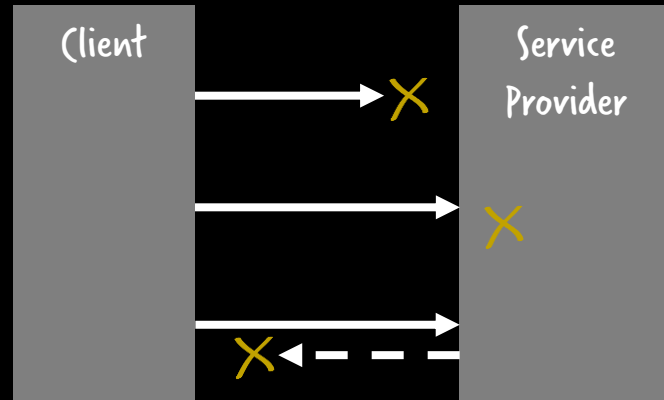


# Distributed systems

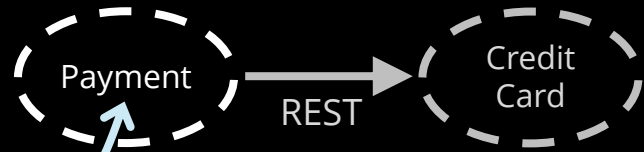
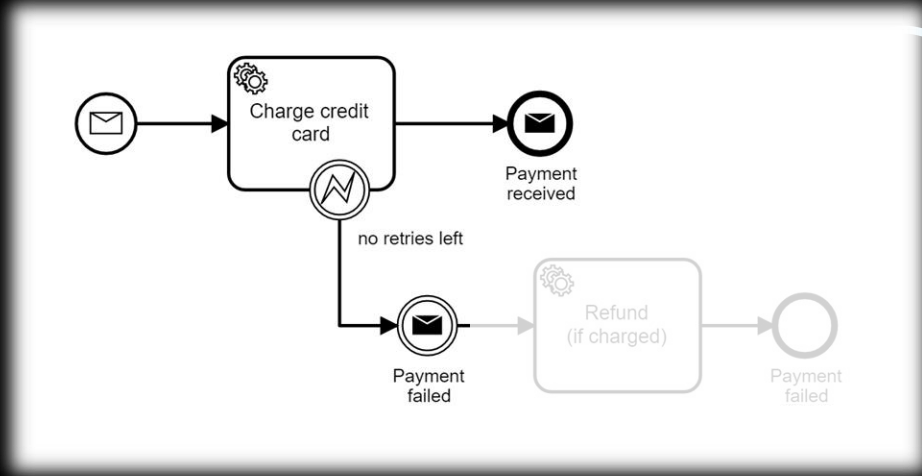


It is impossible to  
differentiate certain  
failure scenarios:

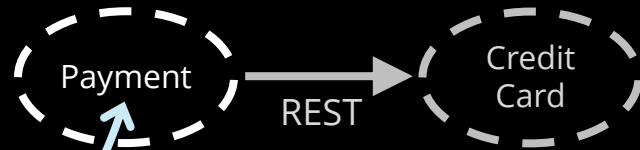
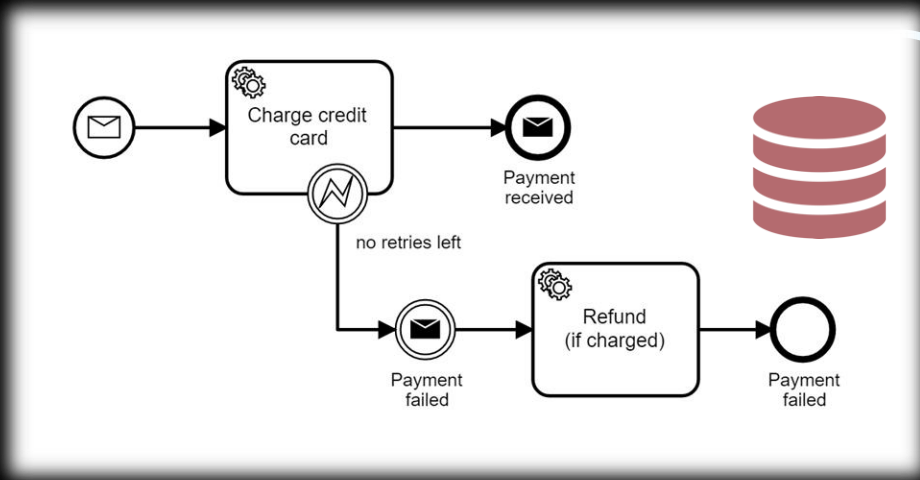
Independant of  
communication style!



# Distributed systems introduce complexity you have to tackle!



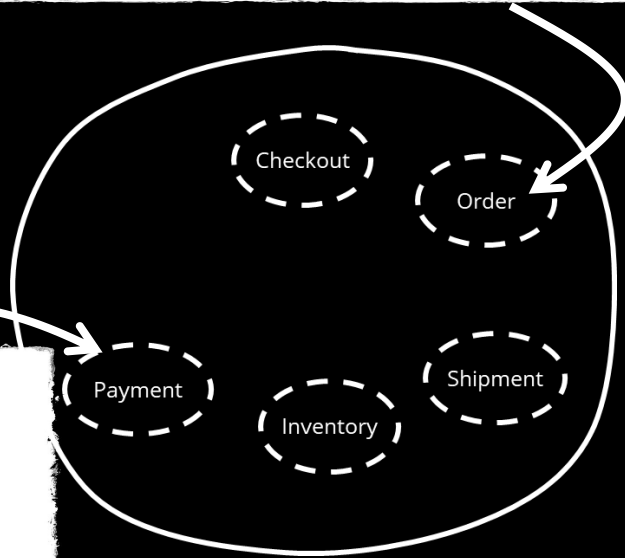
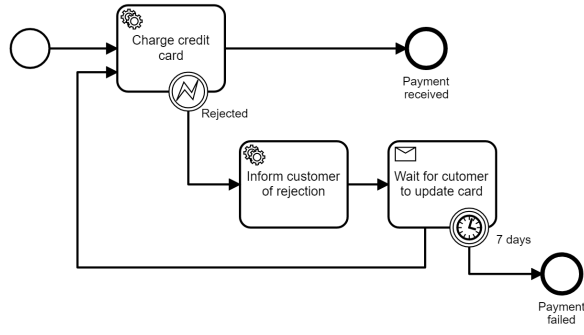
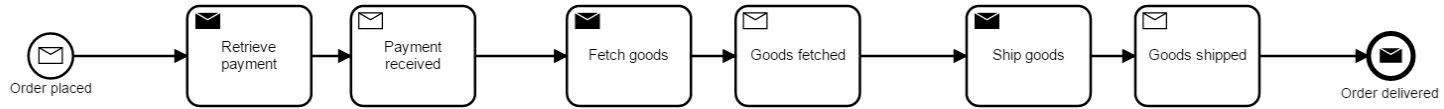
# Distributed systems introduce complexity you have to tackle!



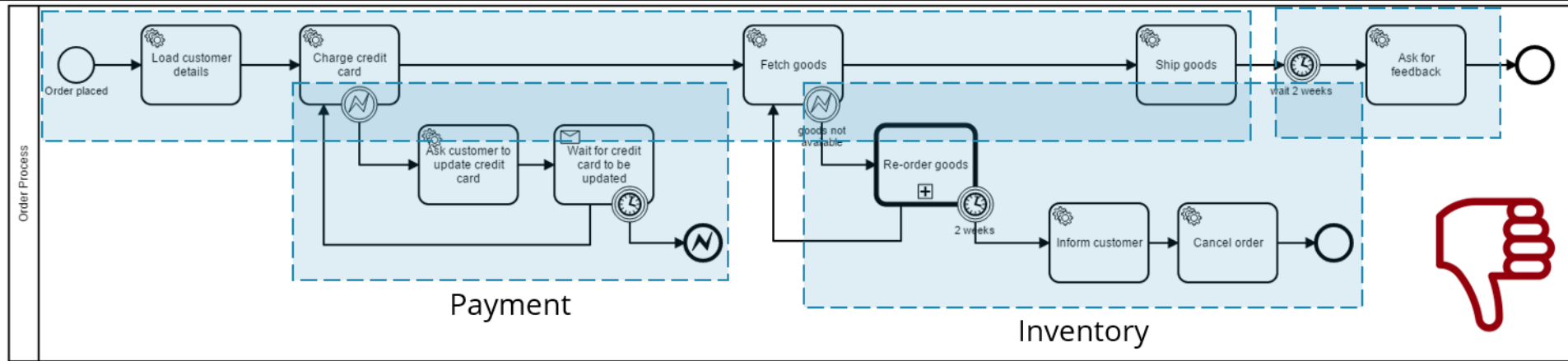
The service can be long running.  
You get a better API and less gods



# Workflows live inside service boundaries



# No BPM(N) monoliths





Pat Helland

Distributed Systems Guru  
Worked at Amazon,  
Microsoft & Salesforce

## Life beyond Distributed Transactions: an Apostate's Opinion

Position Paper

Pat Helland

Amazon.Com  
705 Fifth Ave South  
Seattle, WA 98104  
USA

[PHelland@Amazon.com](mailto:PHelland@Amazon.com)

The positions expressed in this paper are personal opinions and do not in any way reflect the positions of my employer Amazon.com.

### ABSTRACT

Many decades of work have been invested in the area of distributed transactions including protocols such as 2PC, Paxos, and various approaches to quorum. These protocols provide the application programmer a façade of global serializability. Personally, I have invested a non-trivial portion of my career as a strong advocate for the implementation and use of platforms

Instead, applications are built using different techniques which do not provide the same transactional guarantees but still meet the needs of their businesses.

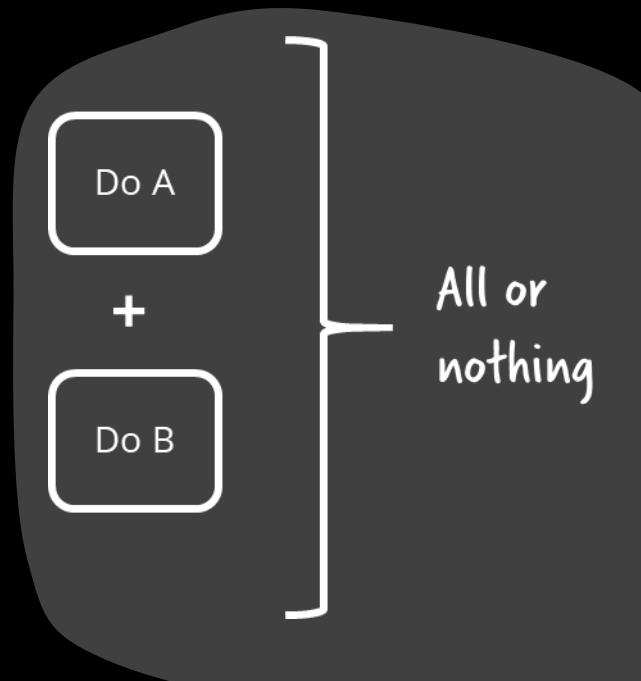
This paper explores and names some of the practical approaches used in the implementations of large-scale mission-critical applications in a world which rejects distributed transactions. We discuss the management of fine-grained pieces of application data which may be repartitioned over time as the application grows. We also discuss the design patterns used in sending messages between these repartitionable pieces of data.

# "Grown-Ups Don't Use Distributed Transactions"



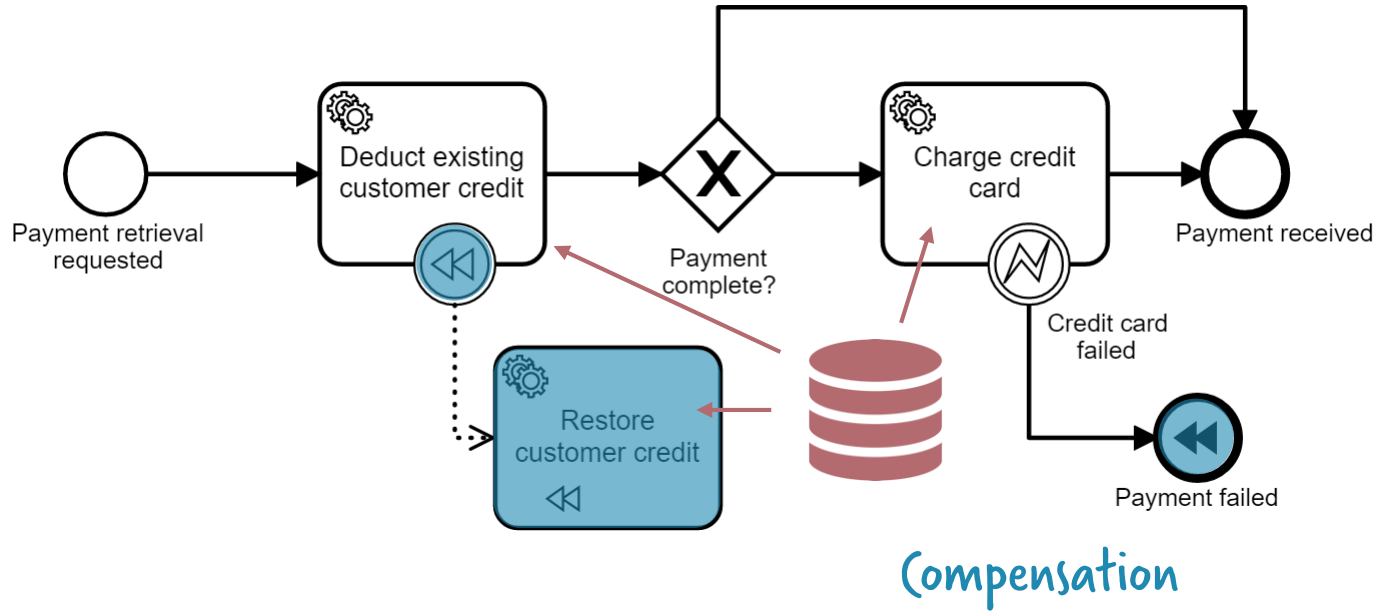
Pat Helland

Distributed Systems Guru  
Worked at Amazon,  
Microsoft & Salesforce

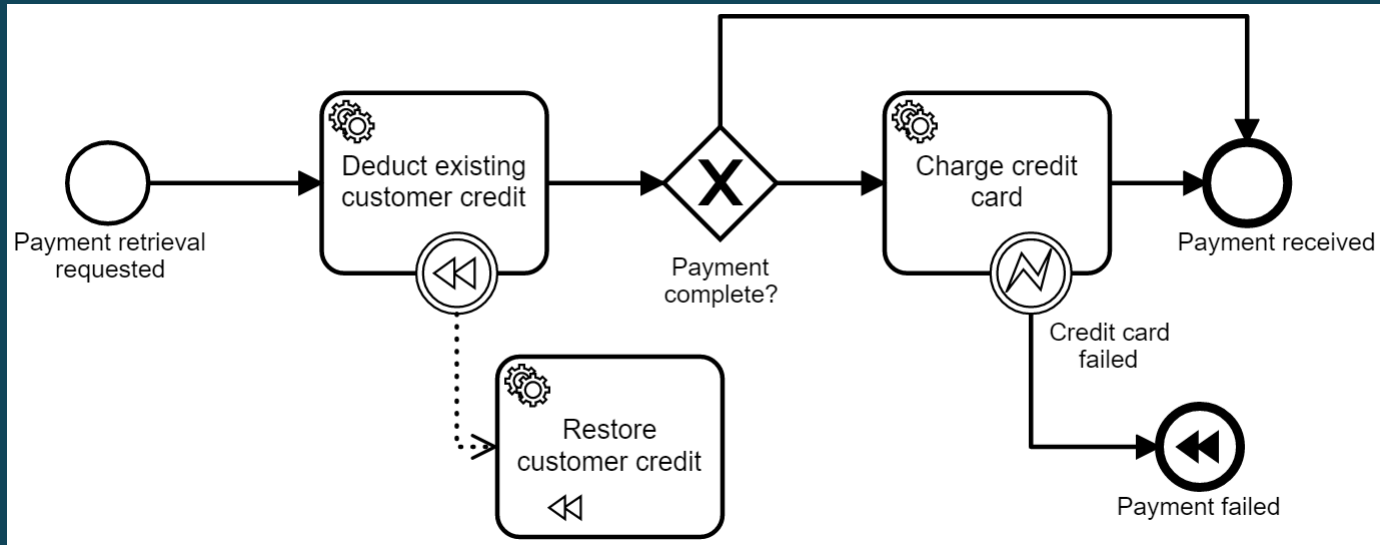


# Distributed transactions using compensation \*

\* aka Saga pattern



Homework:  
Try to do this purely event-driven!



Send to: [mail@berndruecker.io](mailto:mail@berndruecker.io)

# Biz Dev ops

improve  
communication

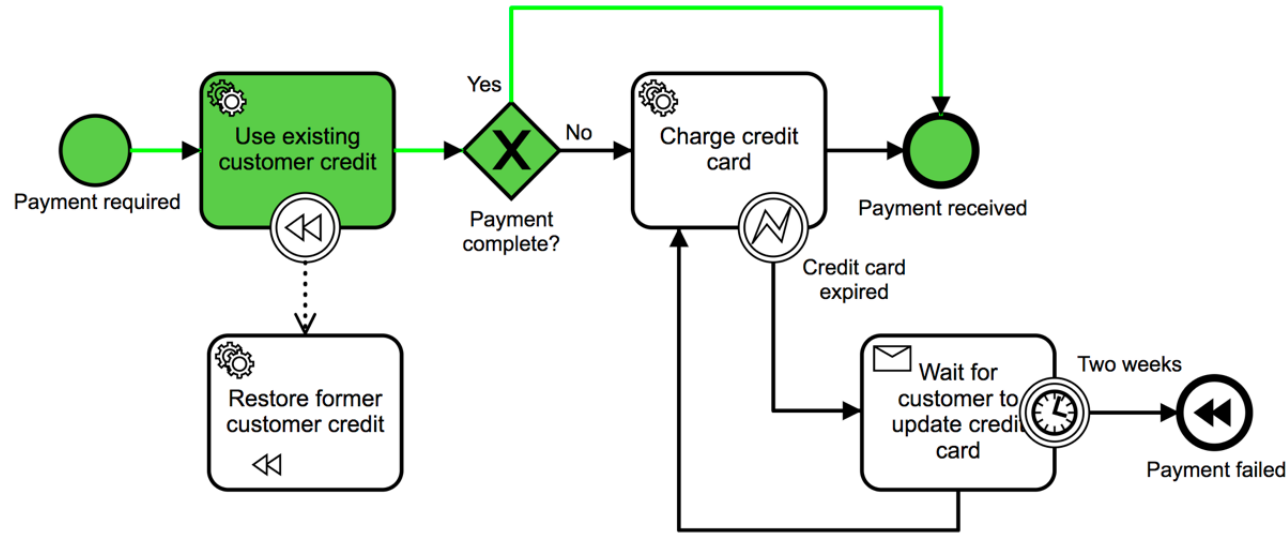
improve  
communication

Leverage  
state machine &  
workflow engine

Living  
documentation

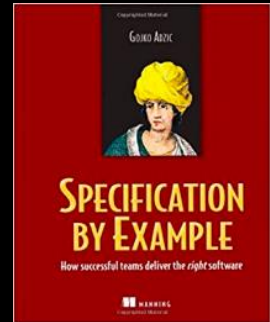
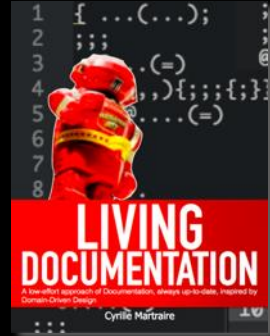
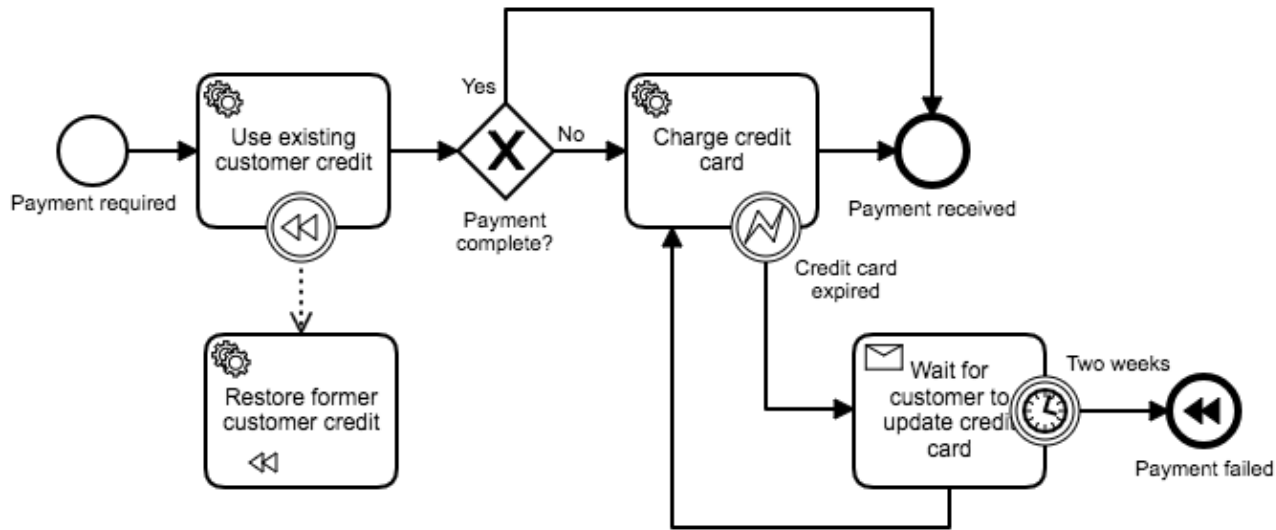
Visibility in  
testing

# Visual HTML reports for test cases





# Living documentation for long-running behaviour



# Proper operations

## Visibility + Context

**Camunda Cockpit** Processes Decisions Cases Human Tasks More ▾

Dashboard > Processes > paymentV5 : Runtime | History

Definition Version: 2  
Version Tag: null  
Definition ID: paymentV5:2:45aea93a-1ad9-11e8-8-...  
Definition Key: paymentV5  
Definition Name: null  
History Time To Live: null  
Tenant ID: null

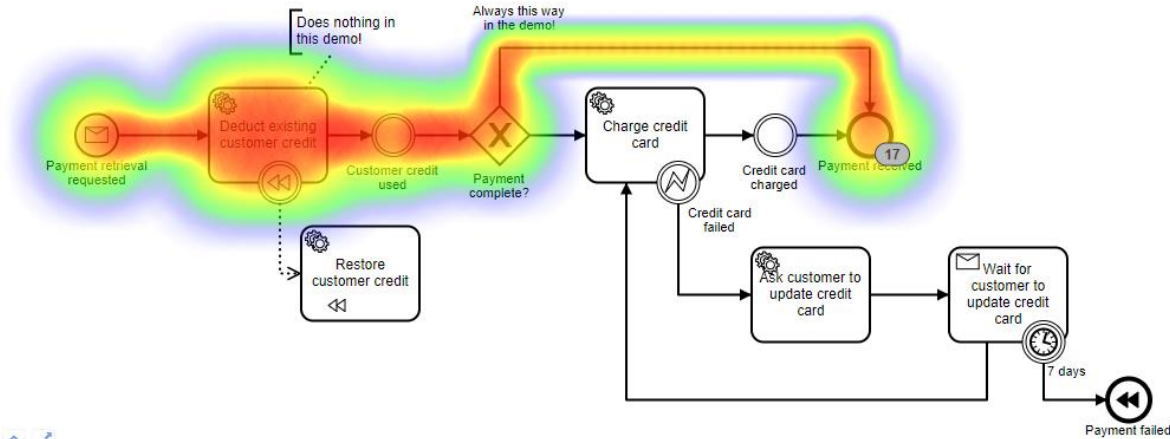
**Process Flow:**

```
graph LR; Start((Payment retrieval requested)) --> Deduct[Deduct existing customer credit]; Deduct --> Decision{Payment complete?}; Decision --> Charge[Charge credit card]; Charge --> End((Payment received)); Decision --> End; Charge --> Deduct; Charge --> Failed[Credit card failed]; Failed --> Ask[Ask customer to update credit card]; Ask --> Wait[Wait for customer to update credit card]; Wait --> Failed; Failed --> Restore[Restore customer credit]; Restore --> Deduct;
```

**Runtime History:**

Start Time	Business Key
2018-02-26T10:40:59	
2018-02-26T10:40:18	

Powered by camunda BPM / v7.8.0-ee



# Biz Dev ops

improve  
communication

improve  
communication

Understand and discuss  
business processes

Leverage  
state machine &  
workflow engine

operate with visibility  
and context

Evaluate optimizations  
in-sync with  
implementation

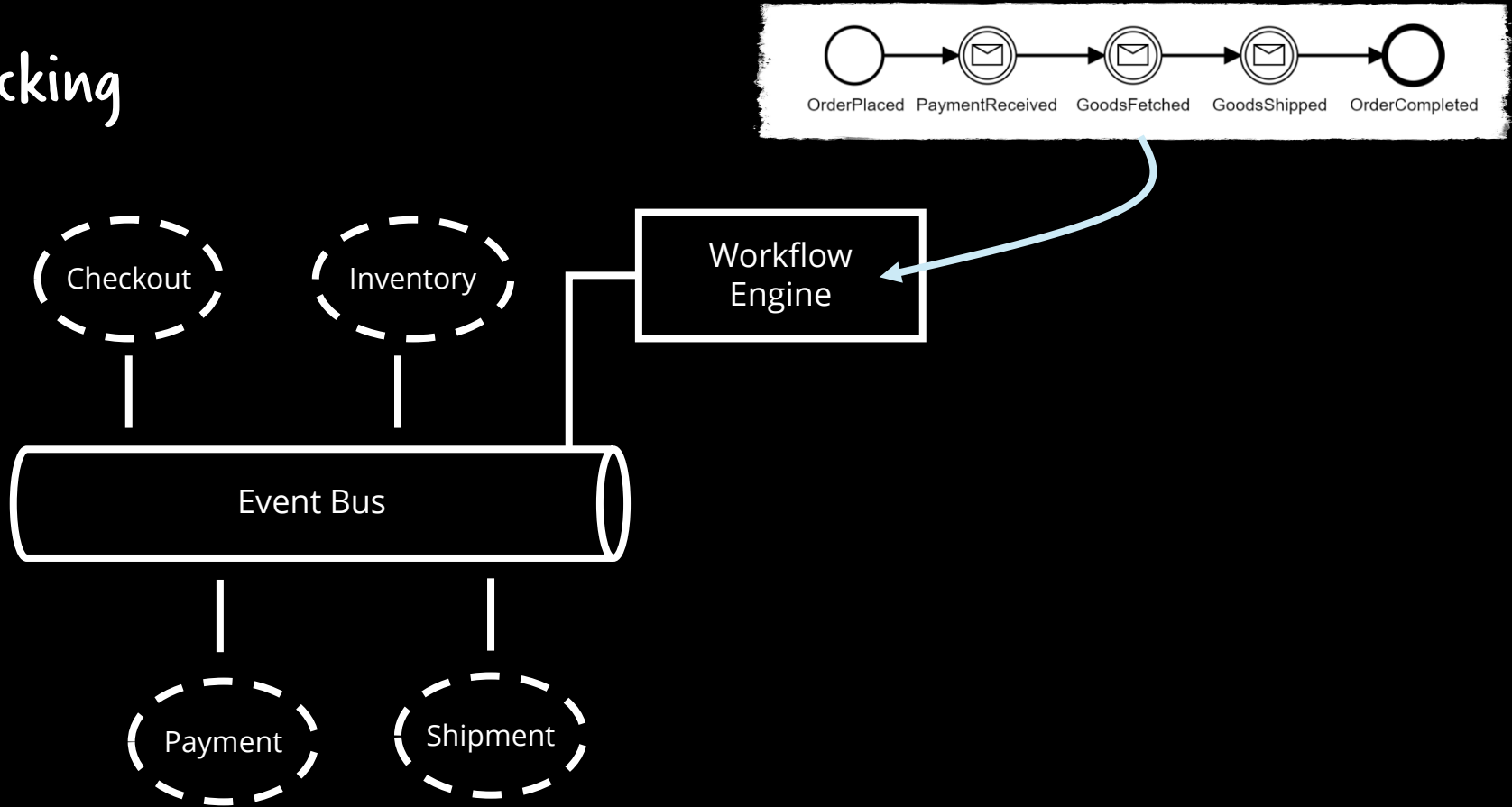
Living  
documentation

Visibility in  
testing

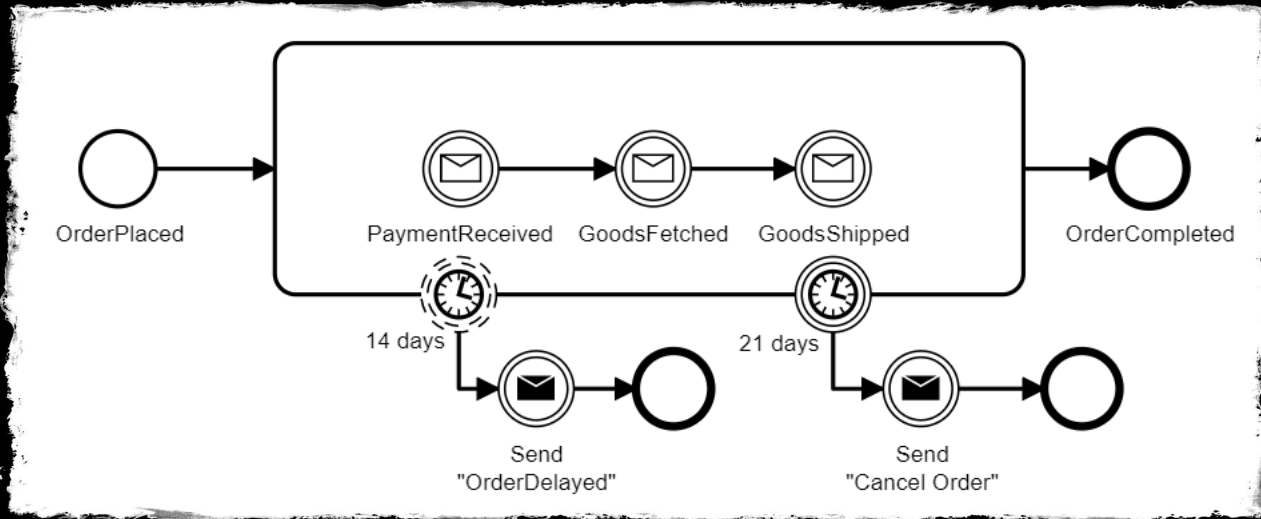
# Monitoring Workflows Across Microservices

The screenshot shows the InfoQ website interface. At the top, there's a navigation bar with categories like 'Development', 'Architecture & Design', 'AI, ML and Data Engineering', 'Culture & Methods', and 'DevOps'. A search bar and a user profile for 'BERND' are also visible. Below the navigation, there's a featured section with tags like 'Streaming', 'Machine Learning', 'Reactive', 'Microservices', 'Containers', and 'NoSQL'. The main article title is 'Monitoring and Managing Workflows Across Collaborating Microservices' under the 'ARCHITECTURE & DESIGN' category. The article is dated 'FEB 28, 2019' and has a '13 MIN READ' duration. It is written by Bernd Rucker and reviewed by Daniel Bryant. A 'Key Takeaways' section lists three points: 1. Peer-to-peer communication between components can lead to emergent behavior, which is challenging for developers, operators and business analysts to understand. 2. You need to make sure to have the overview of all of the backwards-and-forwards communication that is going on in order to fulfill a business capability. 3. Solutions that provide an overview range from distributed tracing, which typically misses the business perspective; data lakes, which require some effort to tune to what you need to know; process tracking, where you have to model a workflow for the tracking; process mining, which can discover the workflow; all the way through to orchestration, which comes with visibility built in. A 'RELATED CONTENT' section on the right lists other articles like 'Experiences Moving from Microservices to Workflows at Jet.com' and 'Debugging Microservices Running in Containers: Tooling Review at KubeCon NA'.

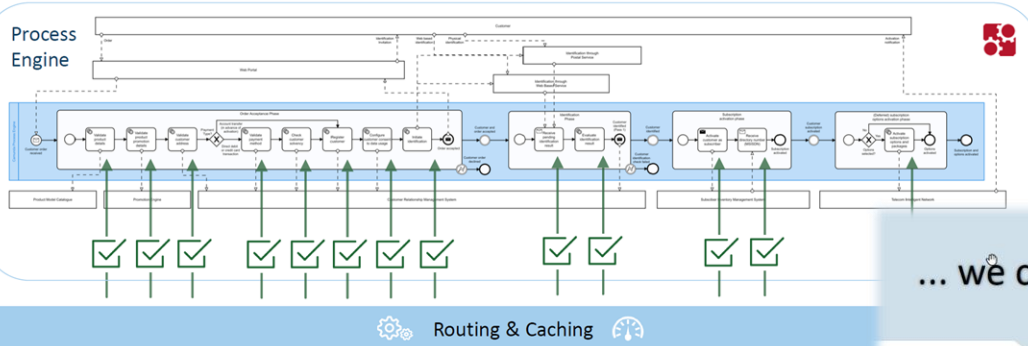
# Tracking



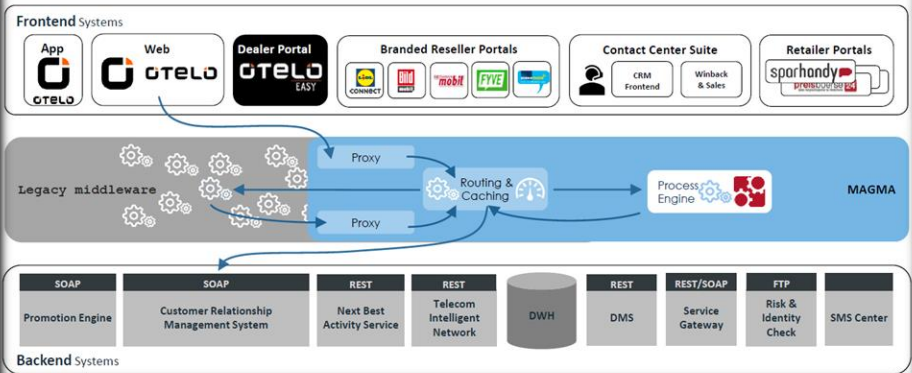
# Journey towards more orchestration



# Using our monitoring abilities for the migration ...



... we could activate the new process step by step



Vodafone, Liongate & WDW  
Presented at CamundaCon Berlin 2018



Not even a full day



”

Before mapping processes explicitly with BPMN, the truth was buried in the code and nobody knew what was going on.

Jimmy Floyd, 24 Hour Fitness





Josh Wulf

Credit Sense

...

It addresses one of the core issues in a distributed microservices architecture—where is the source of truth for the coordinated interaction of the entire system?

...

the system we are replacing uses a complex peer-to-peer choreography that requires reasoning across multiple codebases to understand.

Lightweight workflow engines are  
great – don't DIY\*

\*e.g. enabling potentially long-running services, solving hard  
developer problems, can run decentralized

Reality check

FULFILLMENT PROCESS



Sales-order & order-Fulfillment  
via Camunda  
for every order worldwide  
(Q2 2017: 22,2 Mio)

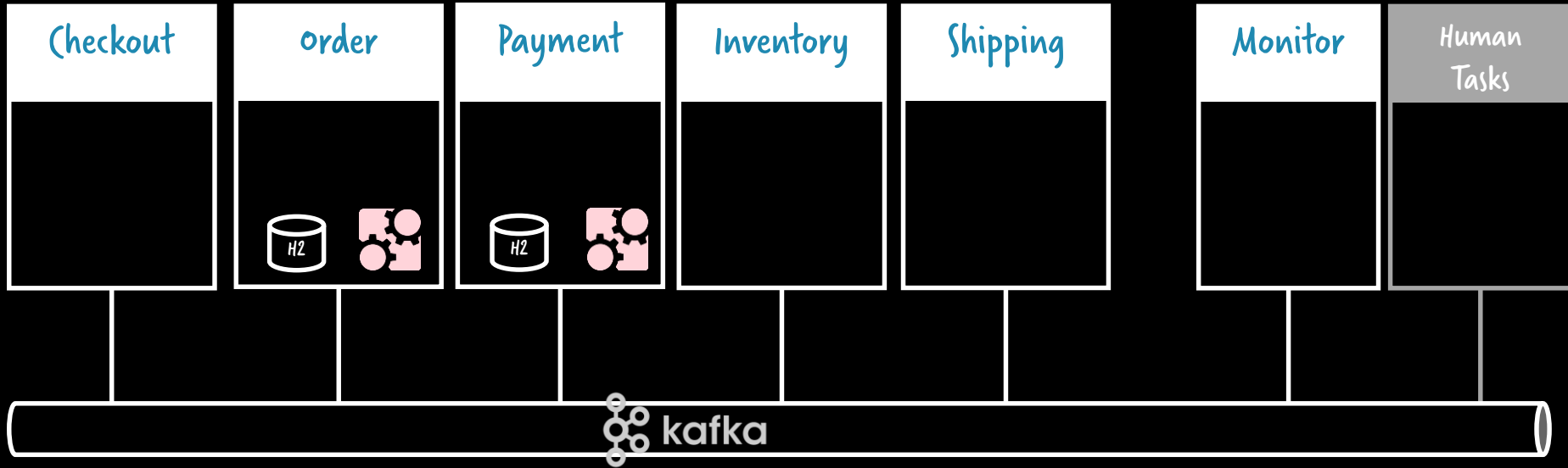


Code, code, code...

The screenshot shows the GitHub interface for the repository 'berndruecker / flowing-retail'. At the top, there are navigation tabs for 'Code', 'Issues', 'Pull requests', 'Projects', 'Wiki', 'Insights', and 'Settings'. The repository description states: 'Sample application demonstrating an order fulfillment system decomposed into multiple independent components (e.g. microservices). Showing concrete implementation alternatives using e.g. Java, Spring Boot, Apache Kafka, Camunda, Zeebe, ...'. Below the description, statistics show 106 commits, 1 branch, 0 releases, and 2 contributors. A progress bar indicates the license is Apache-2.0. A 'Branch: master' dropdown and a 'New pull request' button are visible. A commit history table is shown below, with columns for the commit message and the time since the latest commit.

Commit Message	Time since latest commit
berndruecker Started to add runners for windows-cmd and docker-compose (#12 and #13)	Latest commit 64909d8 3 hours ago
docs included .NET example in readme	5 months ago
kafka Started to add runners for windows-cmd and docker-compose (#12 and #13)	3 hours ago
rest fixed V6 and removed some dead code	5 days ago
runner Started to add runners for windows-cmd and docker-compose (#12 and #13)	3 hours ago
zeebe USe JPA entity instead of HashMap to allow restarts	5 months ago
.gignore changed docker names	4 days ago
.travis.yml Started to add runners for windows-cmd and docker-compose (#12 and #13)	

# Event-driven example



# Events decrease coupling: sometimes

read-models, but no complex peer-to-peer event chains!

# orchestration needs to be avoided: sometimes

no ESB, smart endpoints/dumb pipes, balance orchestration and choreography

# Workflow engines are painful: some of them

lightweight engines are easy to use and can run decentralized,  
they solve hard developer problems, don't DIY

Thank you!



Contact: [mail@berndruecker.io](mailto:mail@berndruecker.io)  
[@berndruecker](#)

Slides: <https://berndruecker.io>

Blog: <https://medium.com/berndruecker>

Code: <https://github.com/berndruecker>

**InfoWorld**  
FROM IDG

<https://www.infoworld.com/article/3254777/application-development/3-common-pitfalls-of-microservices-integration-and-how-to-avoid-them.html>

**InfoQ**  
neue

<https://www.infoq.com/articles/events-workflow-automation>

**THE NEW STACK**

<https://thenewstack.io/5-workflow-automation-use-cases-you-might-not-have-considered/>

