Complex event flows in distributed systems

@berndruecker

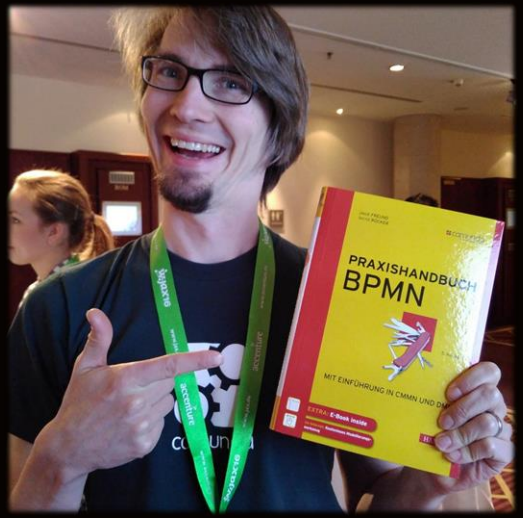
3 common hypotheses I check today:

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

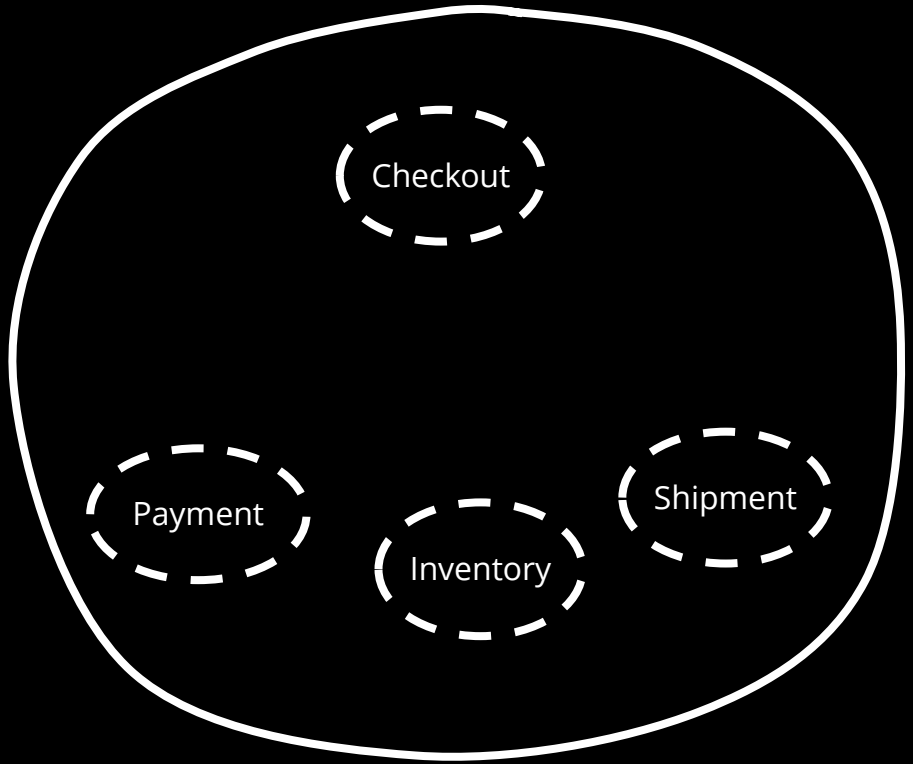
mail@berndruecker.io
@berndruecker


ORCHESTRATING A HIGHLY-SCALABLE FULFILLMENT PROCESS
JÖRN HORSTMANN
LUKAS NIEMEIER
2017-09-19
THE SHUTTLE
TECH INNOVATION LAB

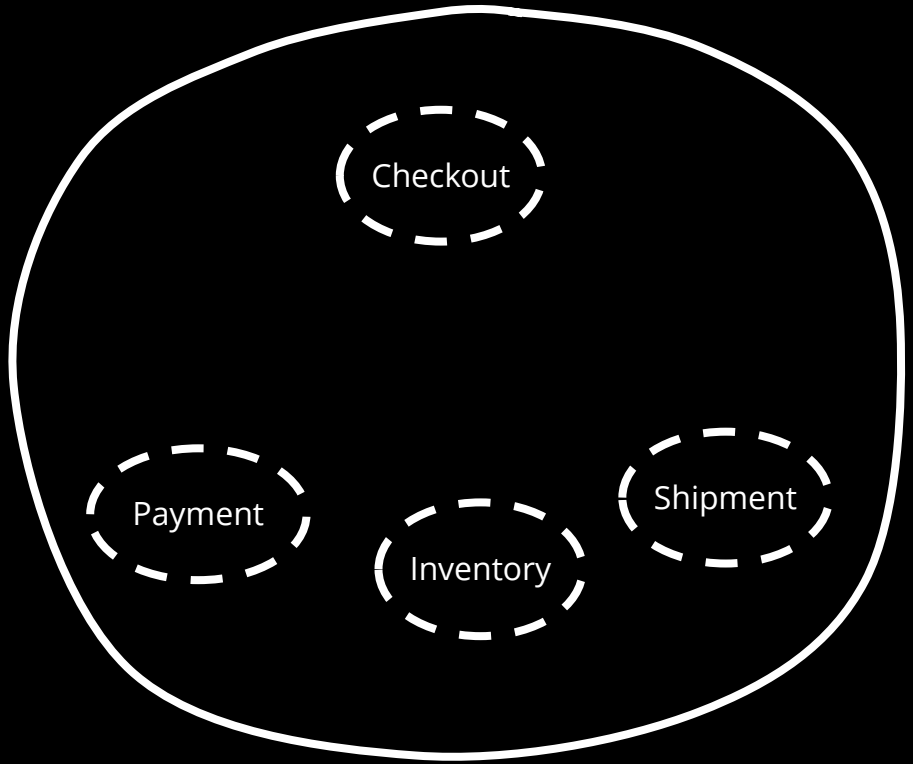
Simplified example: dash button

Three steps...

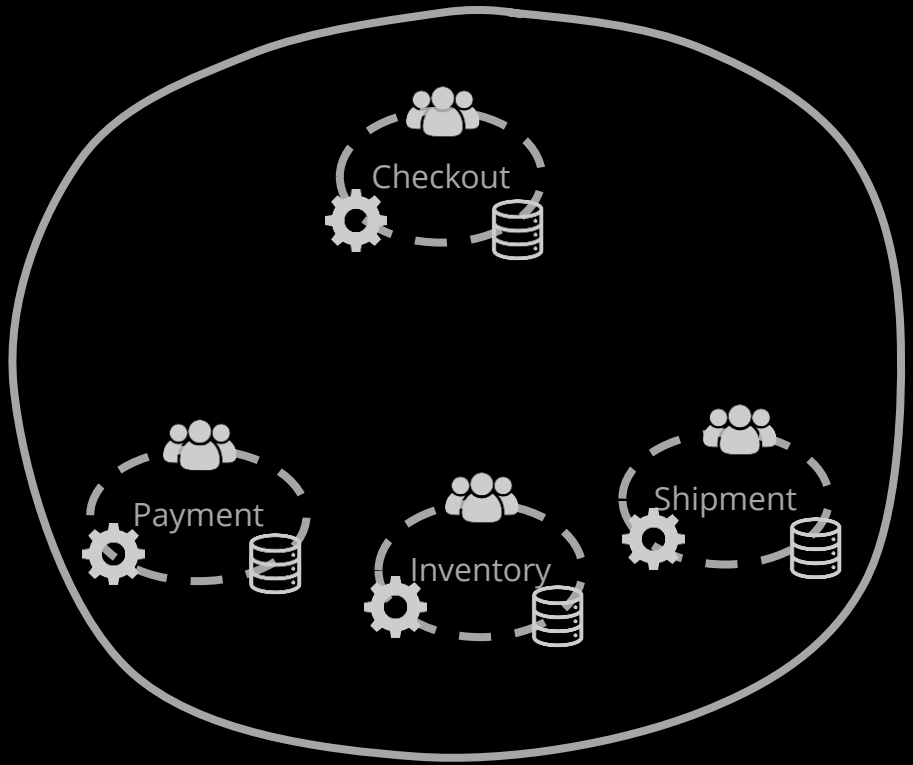
Who is involved? Some bounded contexts...



(Micro-)services



Autonomous (micro-)services



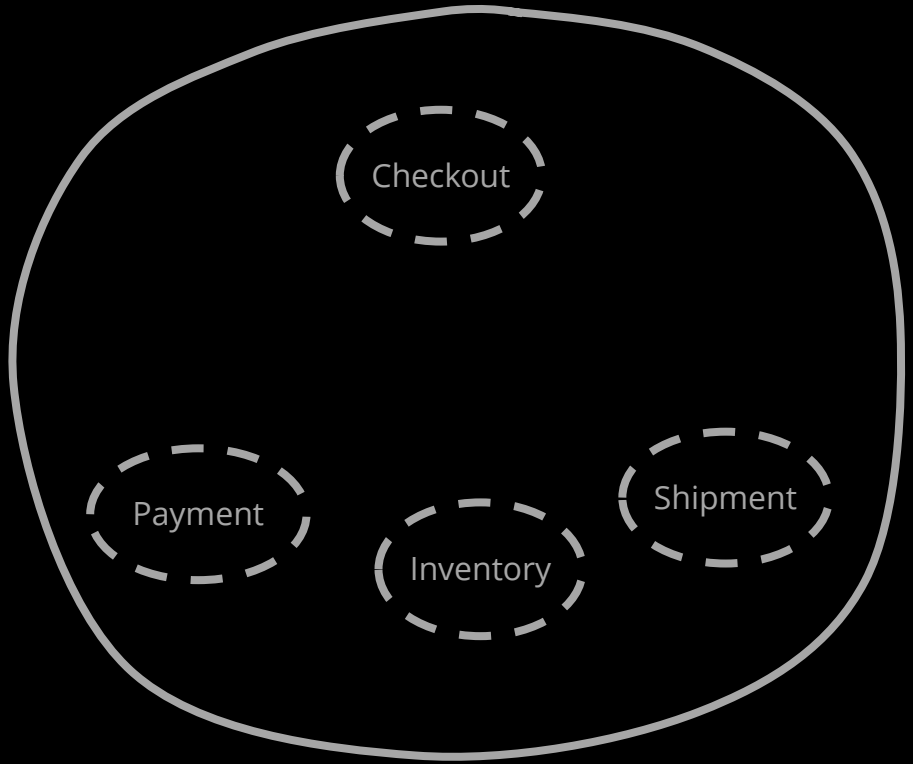
Dedicated Application Processes

Dedicated infrastructure

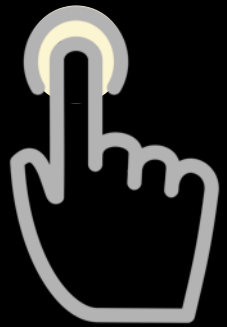
Dedicated Development Teams

Events decrease coupling

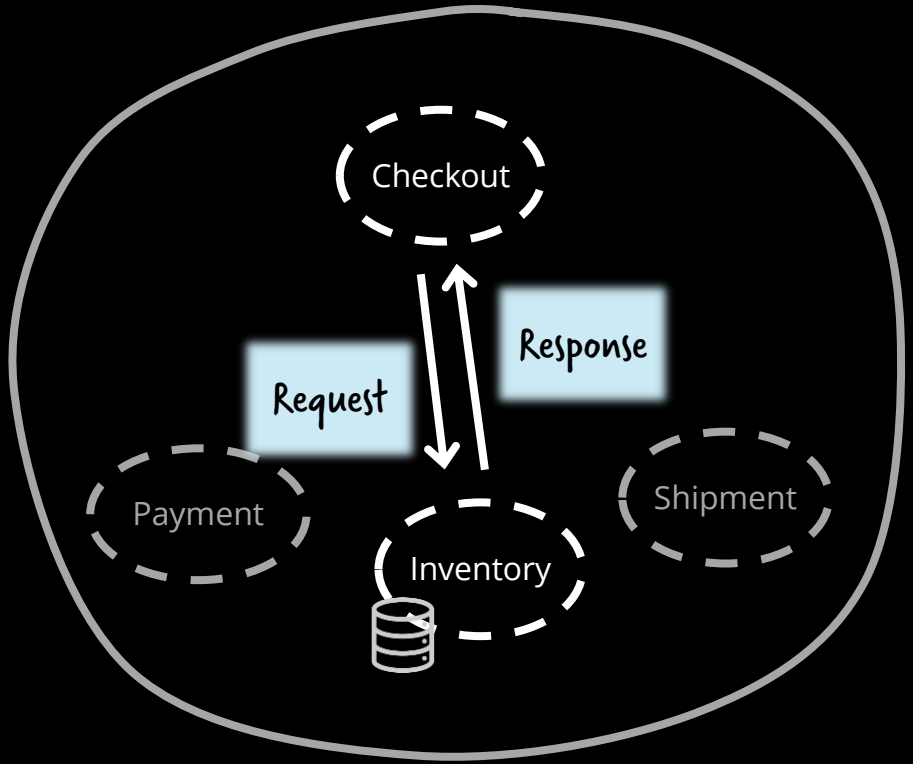
Example



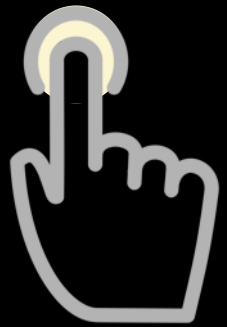
The button blinks if we can ship within 24 hours



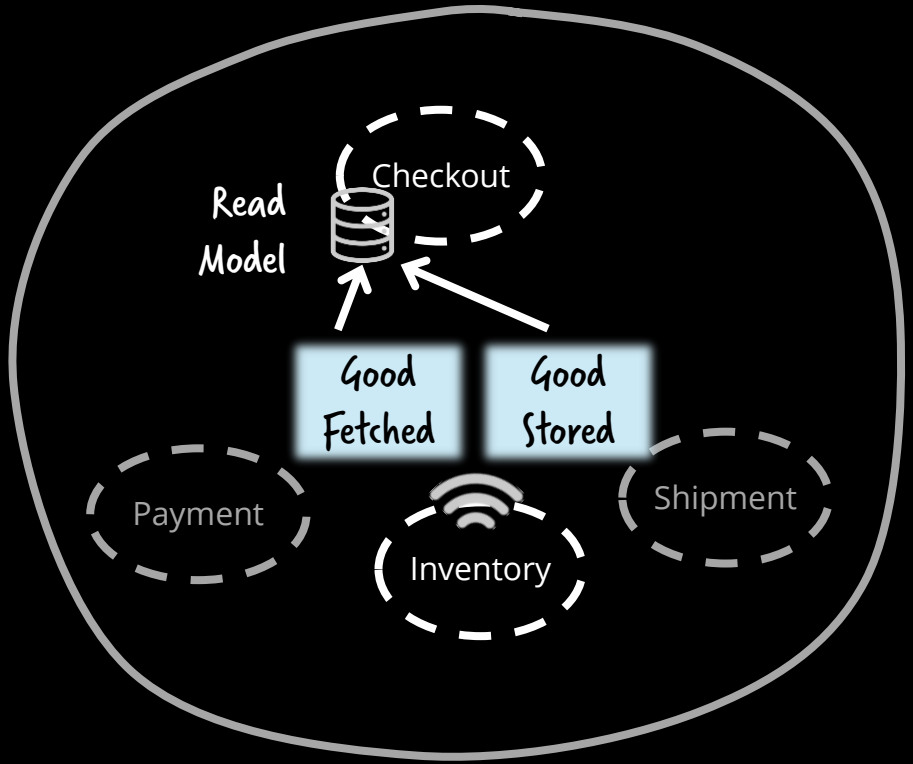
Request/response: temporal coupling



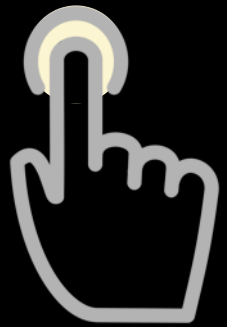
The button blinks if we can ship within 24 hours



Temporal decoupling with events and read models

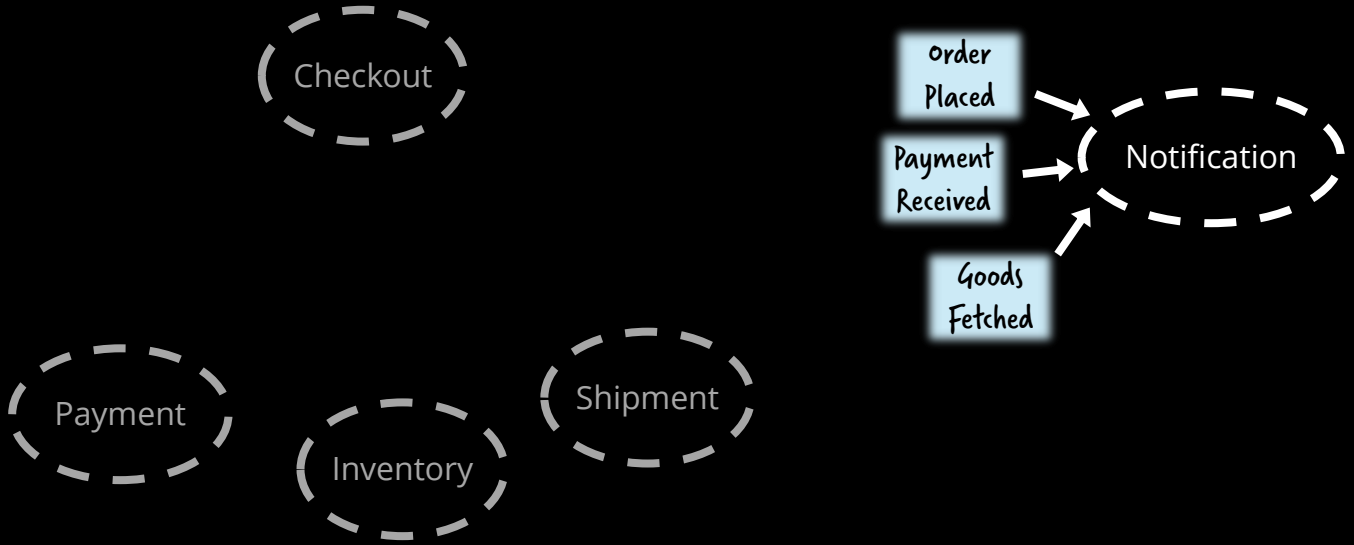


The button blinks if we can ship within 24 hours



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

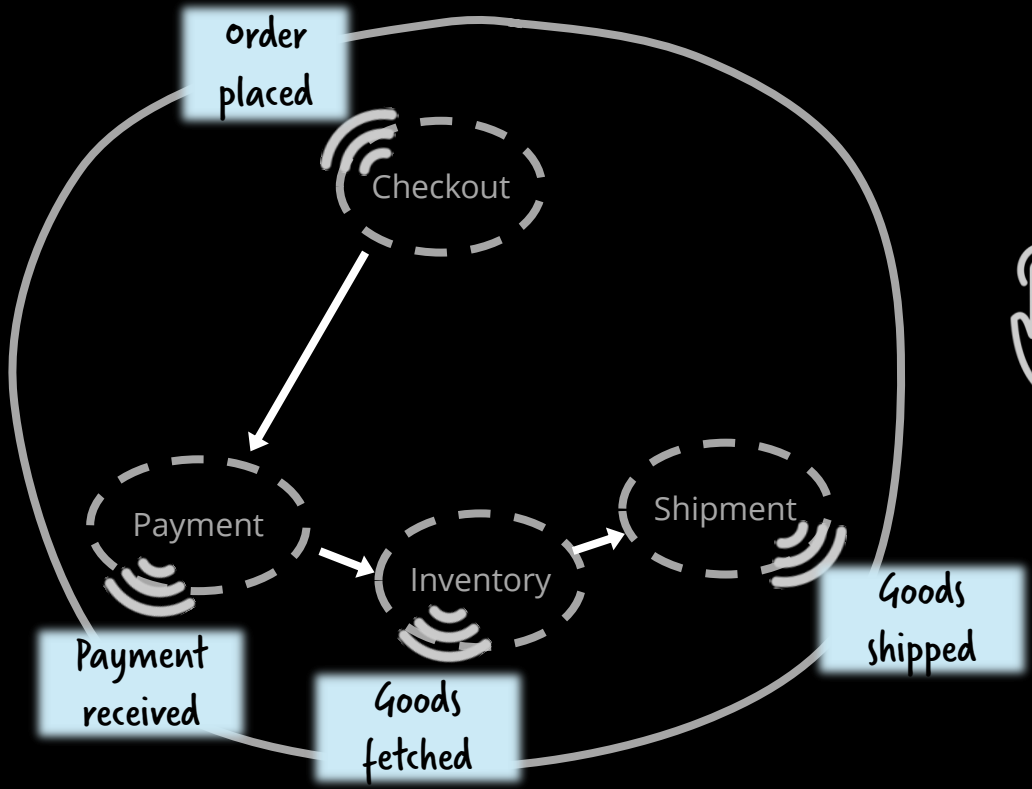
Event-driven architecture



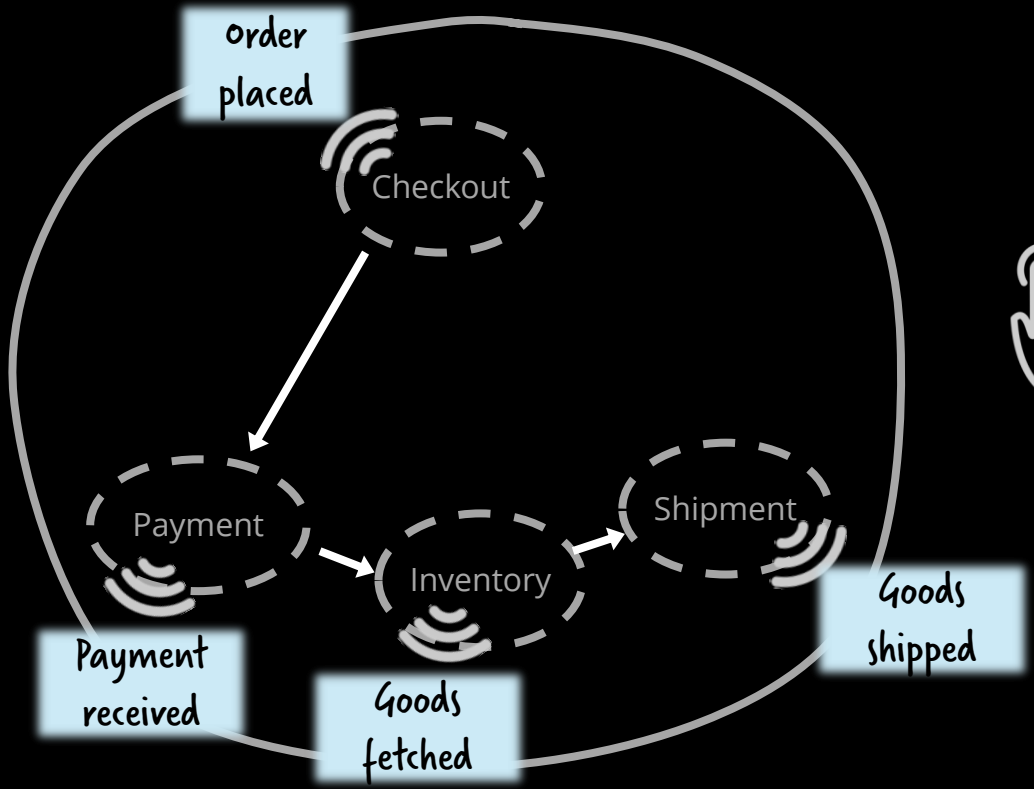
Events can decrease coupling*

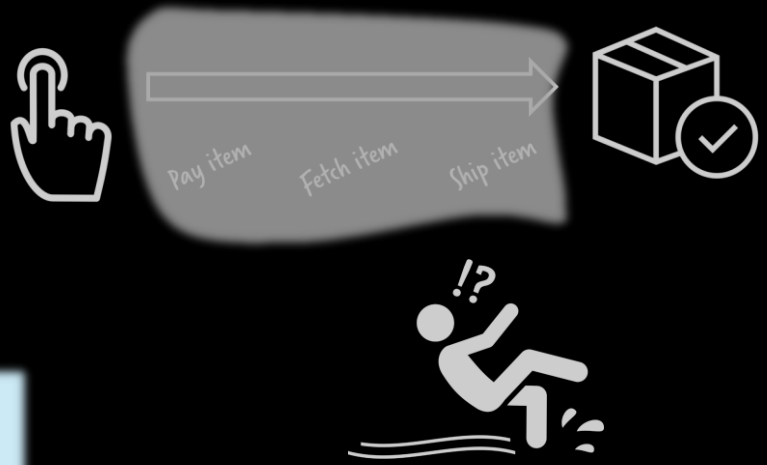
*e.g. decentral data-management, read models, extract cross-cutting aspects

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.

https://martinfowler.com/articles/201701-event-driven.html

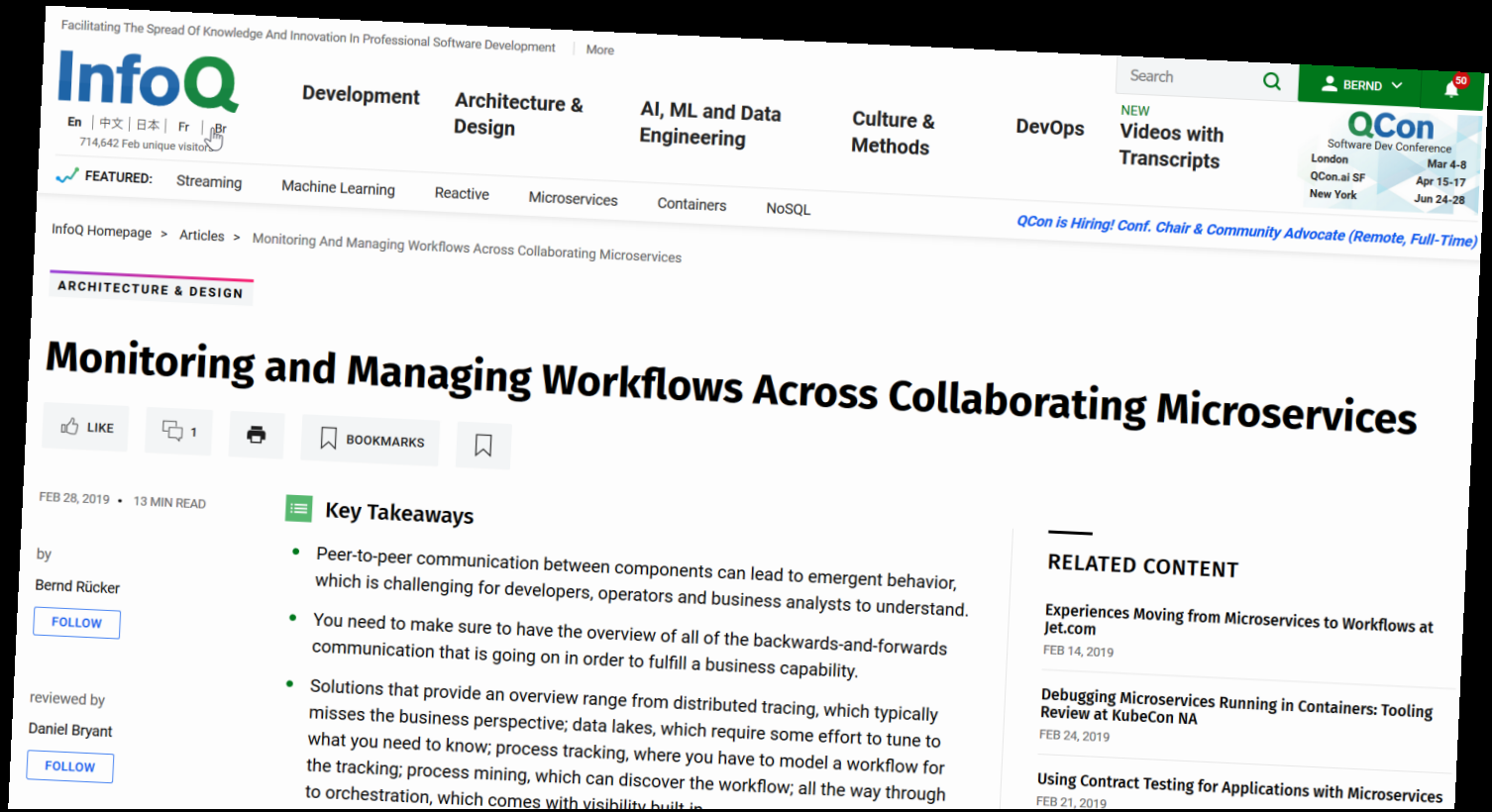
The danger is that it's very easy to make nicely decoupled systems with event notification, without realizing that you're losing sight of that larger-scale flow, and thus set yourself up for trouble in future years.

https://martinfowler.com/articles/201701-event-driven.html

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.

https://martinfowler.com/articles/201701-event-driven.html

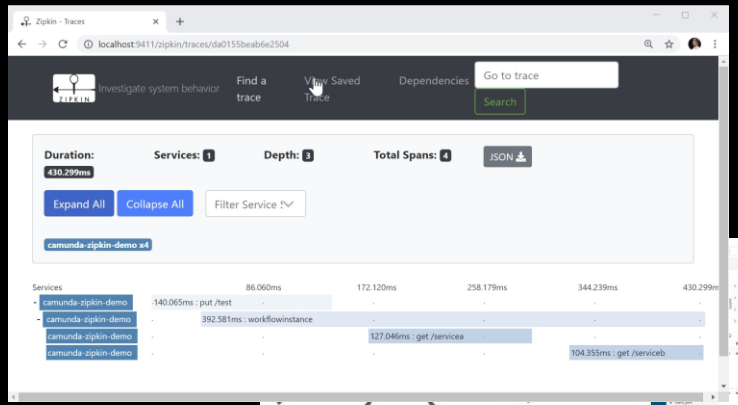
Monitoring Workflows Across Microservices



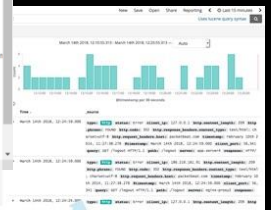
https://www.infoq.com/articles/monitor-workflow-collaborating-microservices

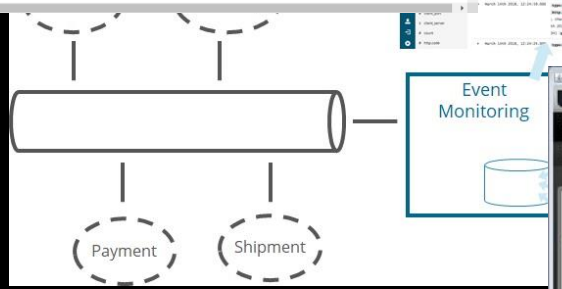
Typical approaches

Distributed Tracing

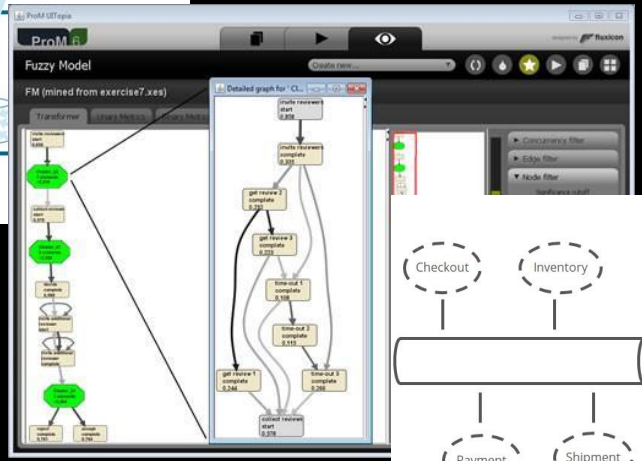


Data Lake / Event Monitoring

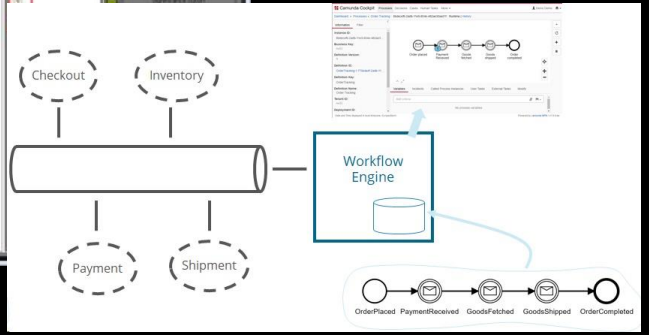




Process Mining



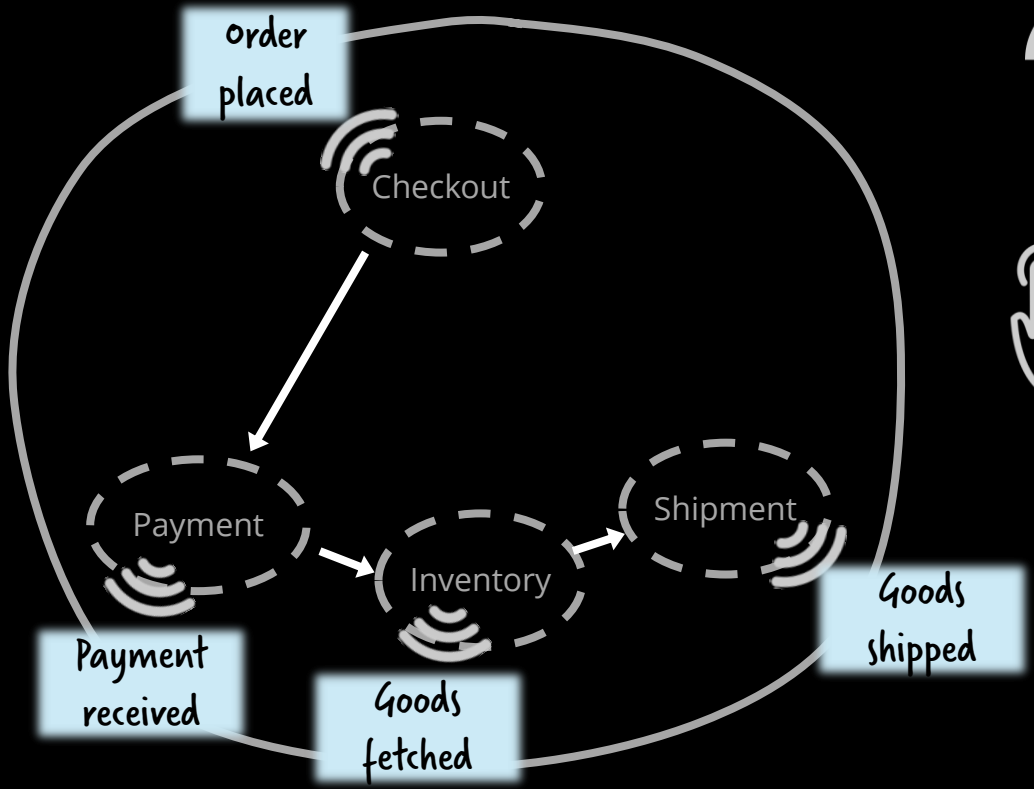
Process Tracking



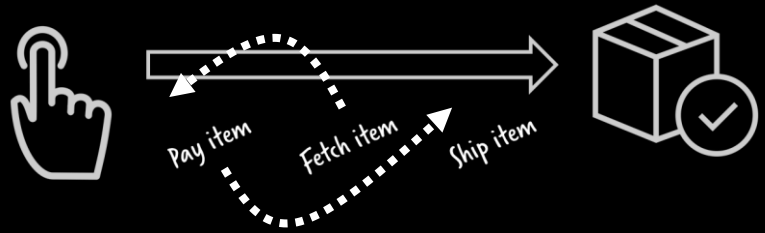
- Emerging behavior (a.k.a. “what the hell just happened?”)

Stefan Tilkov: Microservice Patterns & Antipatterns - MicroXchg 2018

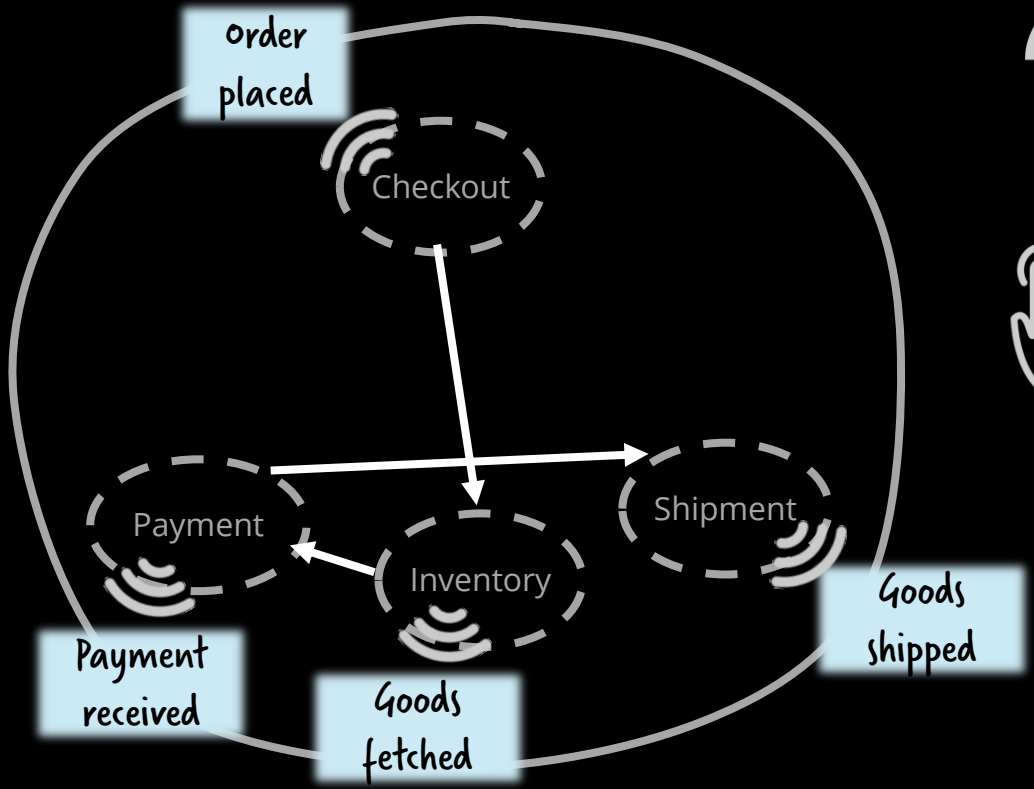
Peer-to-peer event chains



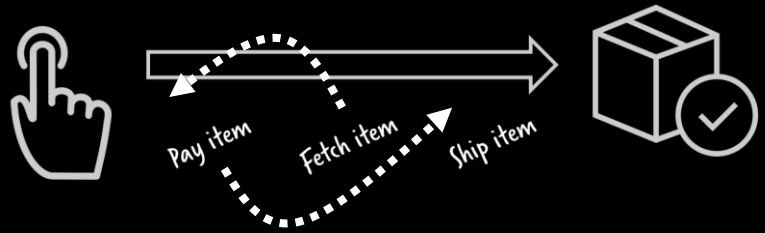
Fetch the goods **before** the payment



Peer-to-peer event chains



Fetch the goods **before** the payment

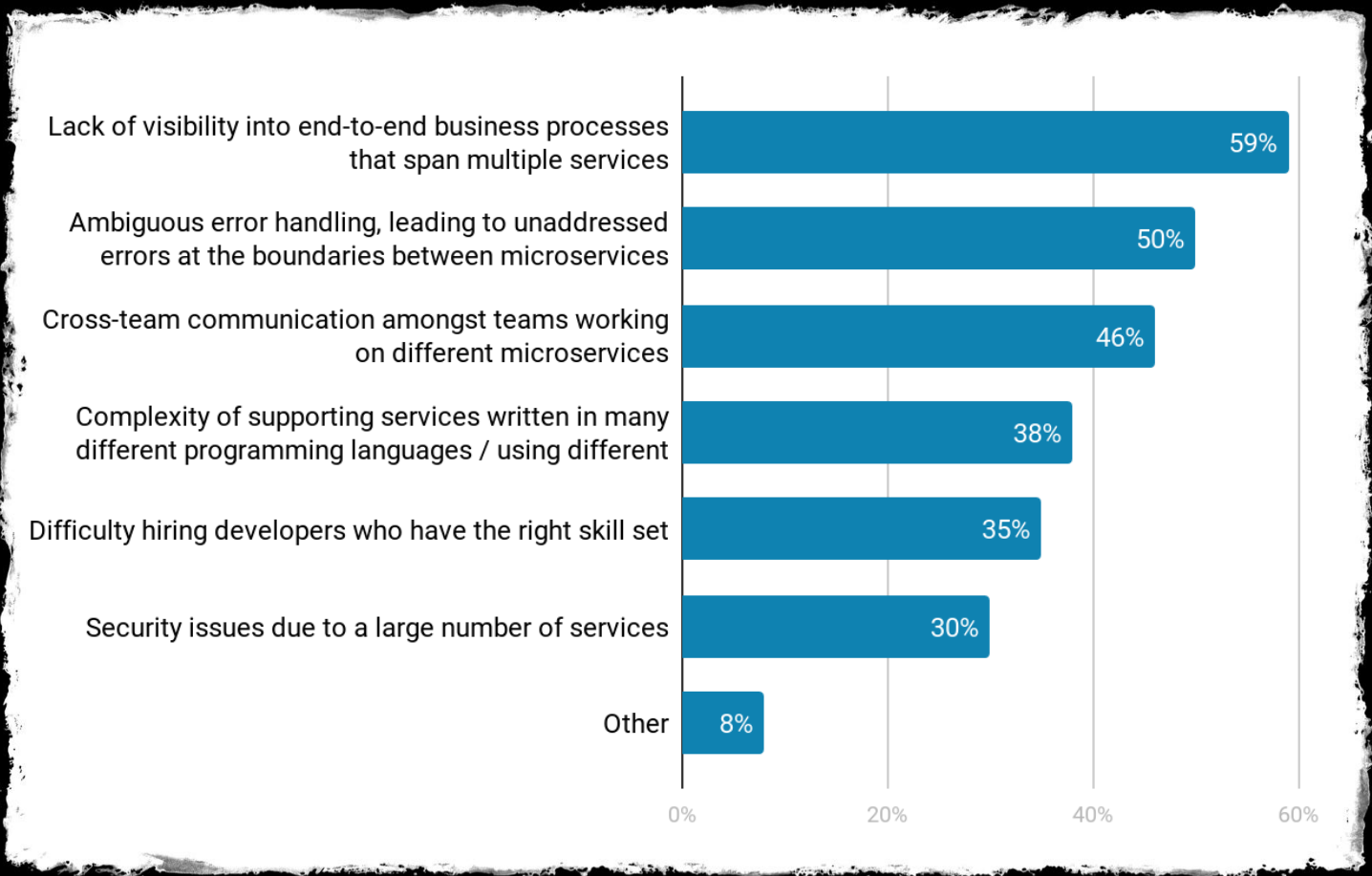


What we wanted

vs. what we got

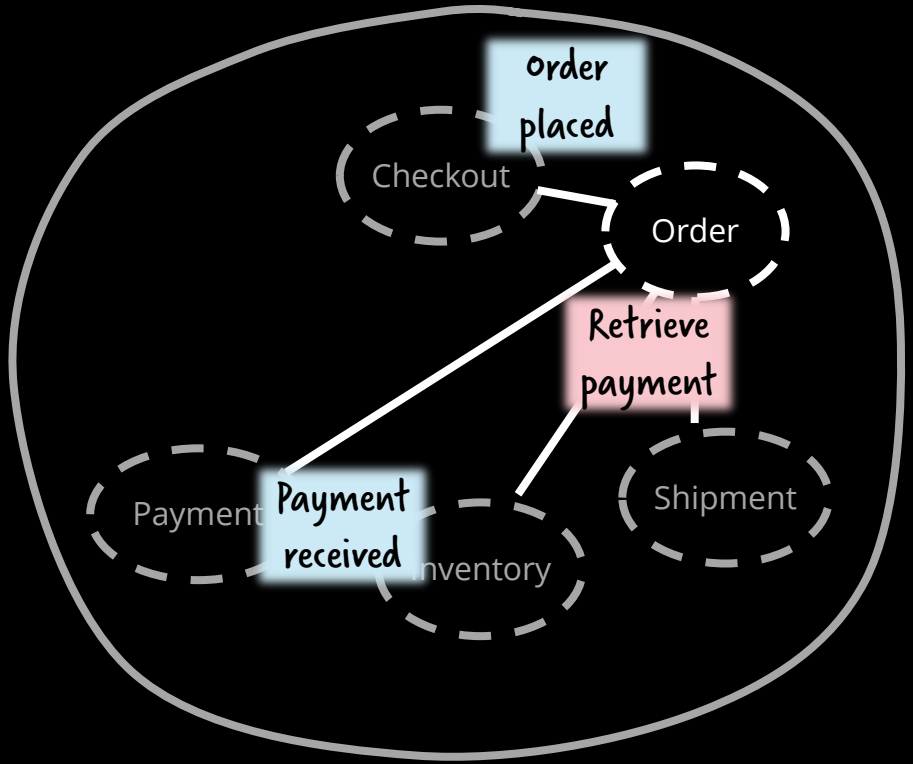
„Challenges?"

https://camunda.com/microservices-orchestration-survey-results-2018/



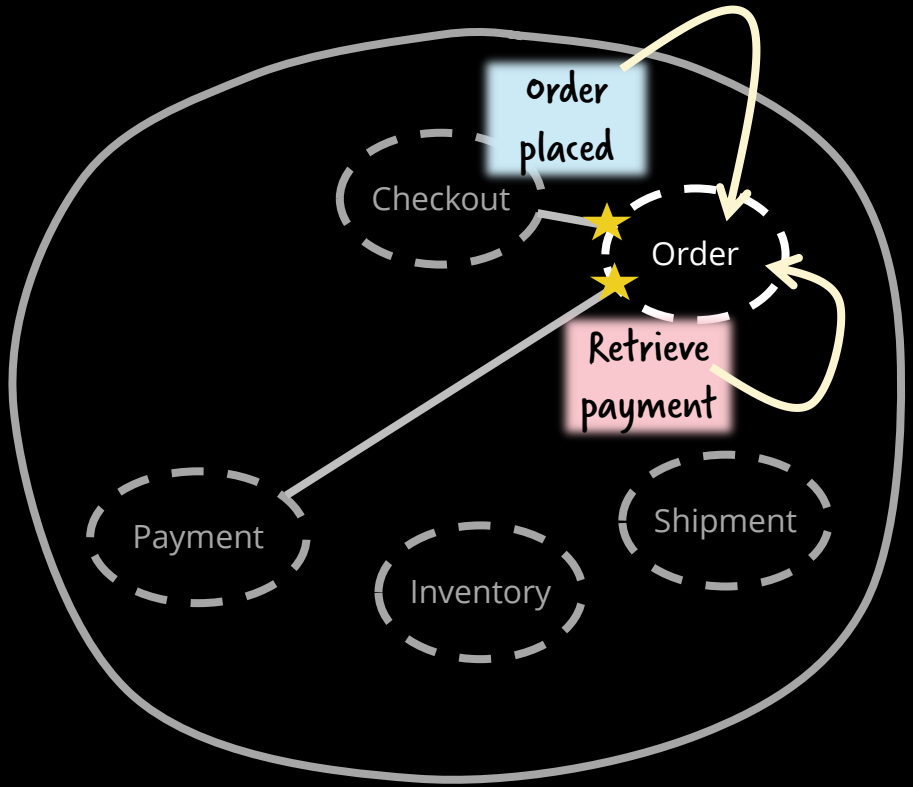
Source:
Microservices orchestration survey,
July 2018, 354 responses

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!

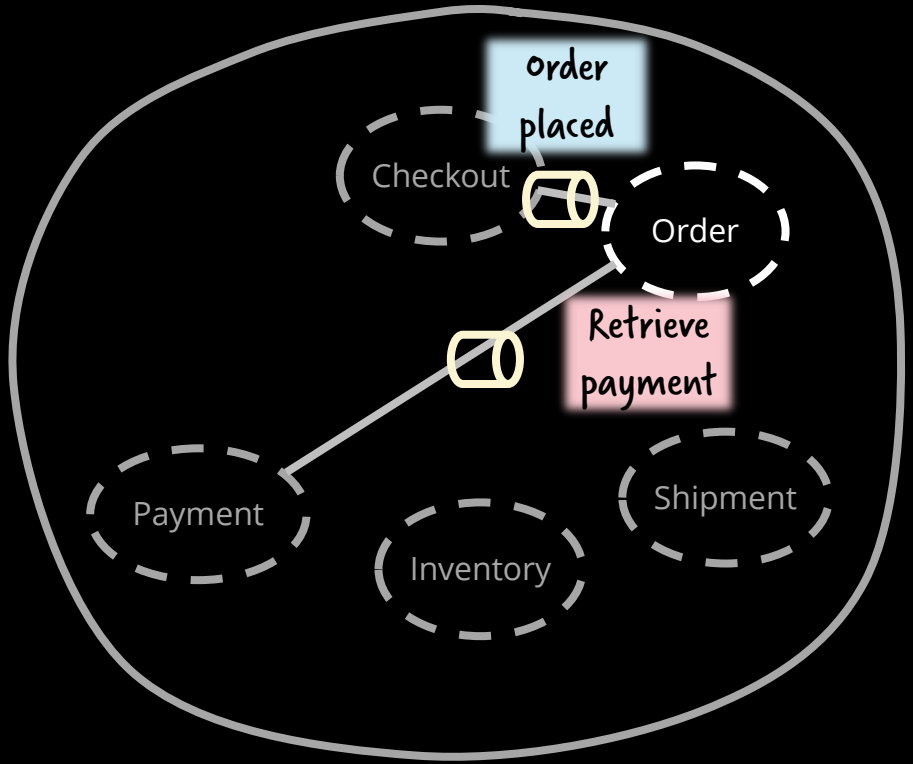


Order decides

. to listen to the event

. to issue the command

It is about where to decide about the coupling!

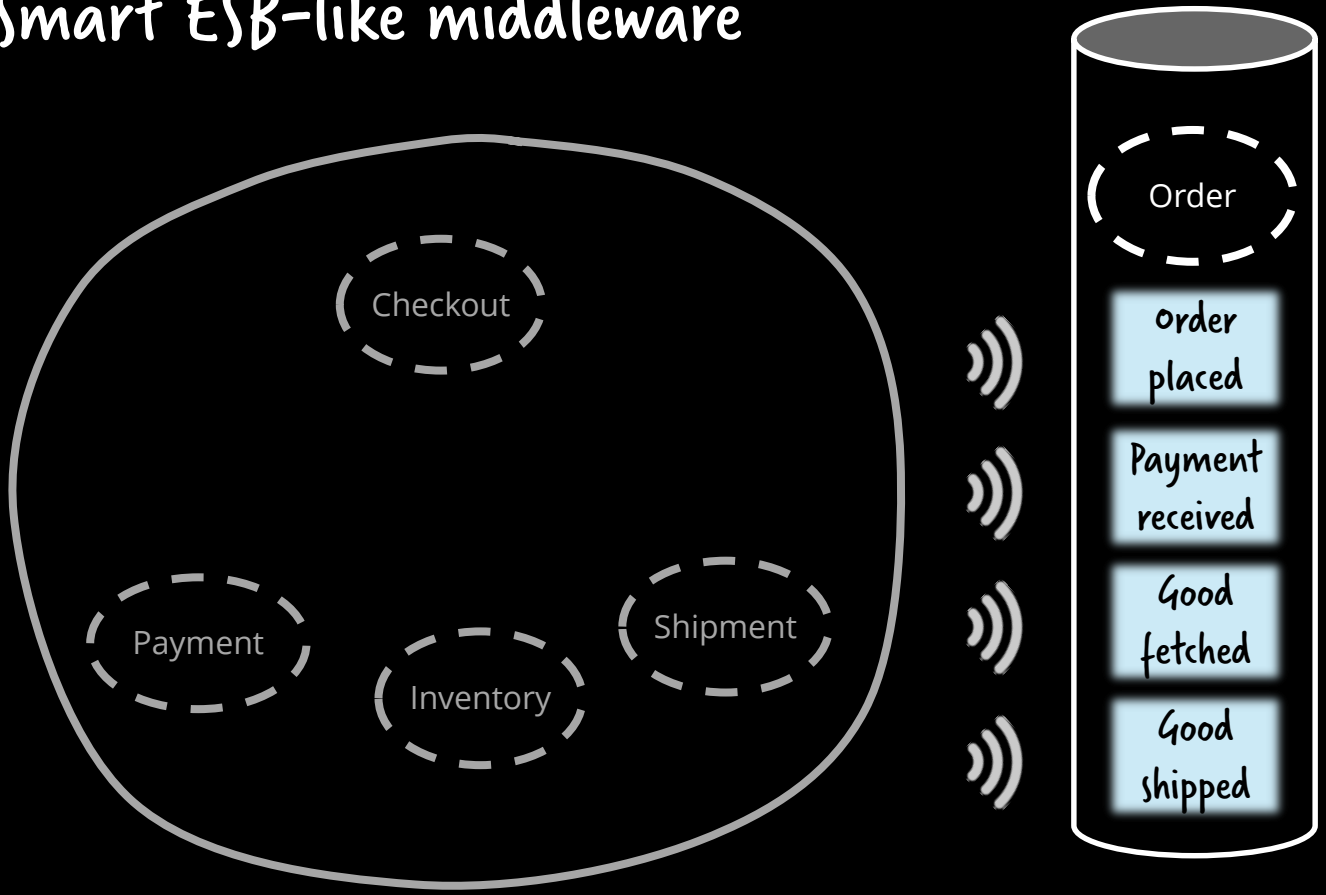


It can still be messaging!

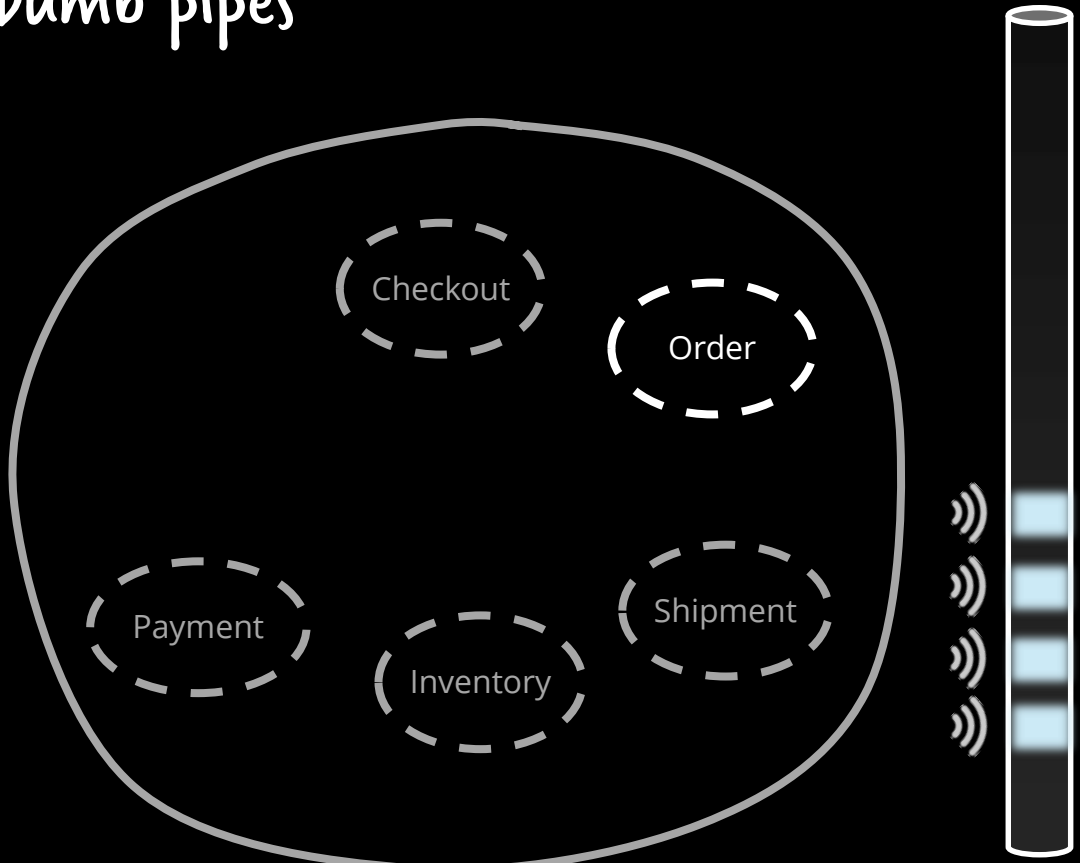
Commands help to avoid (complex) peer-to-peer event chains

orchestration needs to be avoided

Smart ESB-like middleware



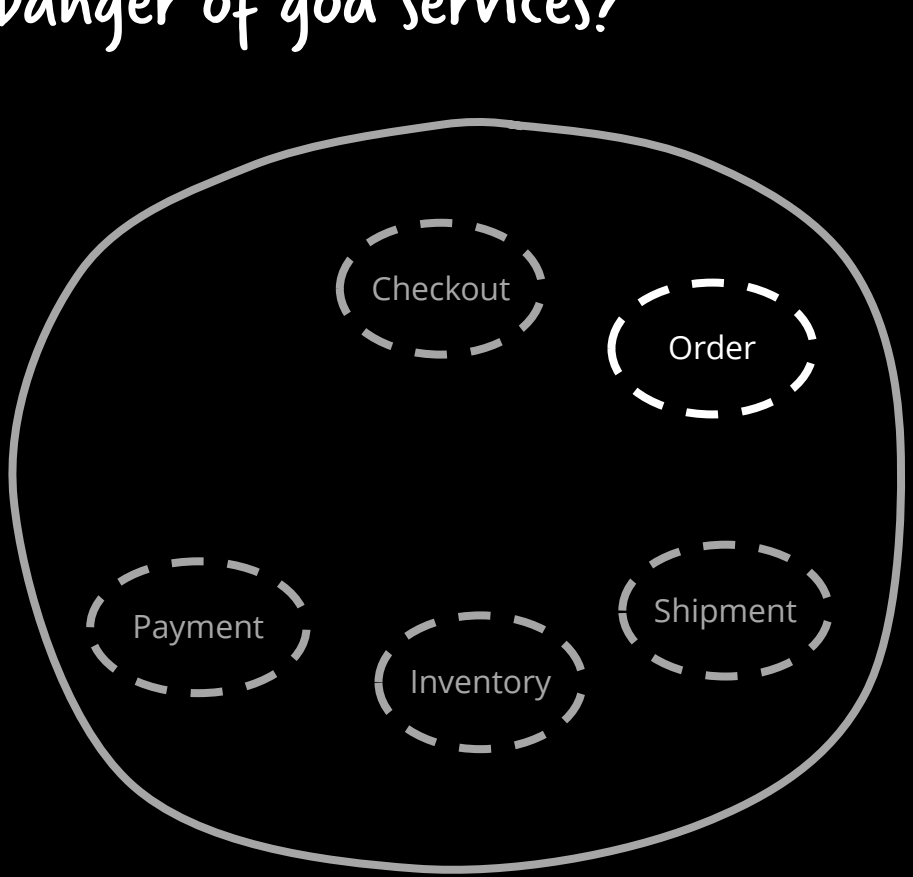
Dumb pipes



Martin Fowler

Smart endpoints
and **dumb pipes**

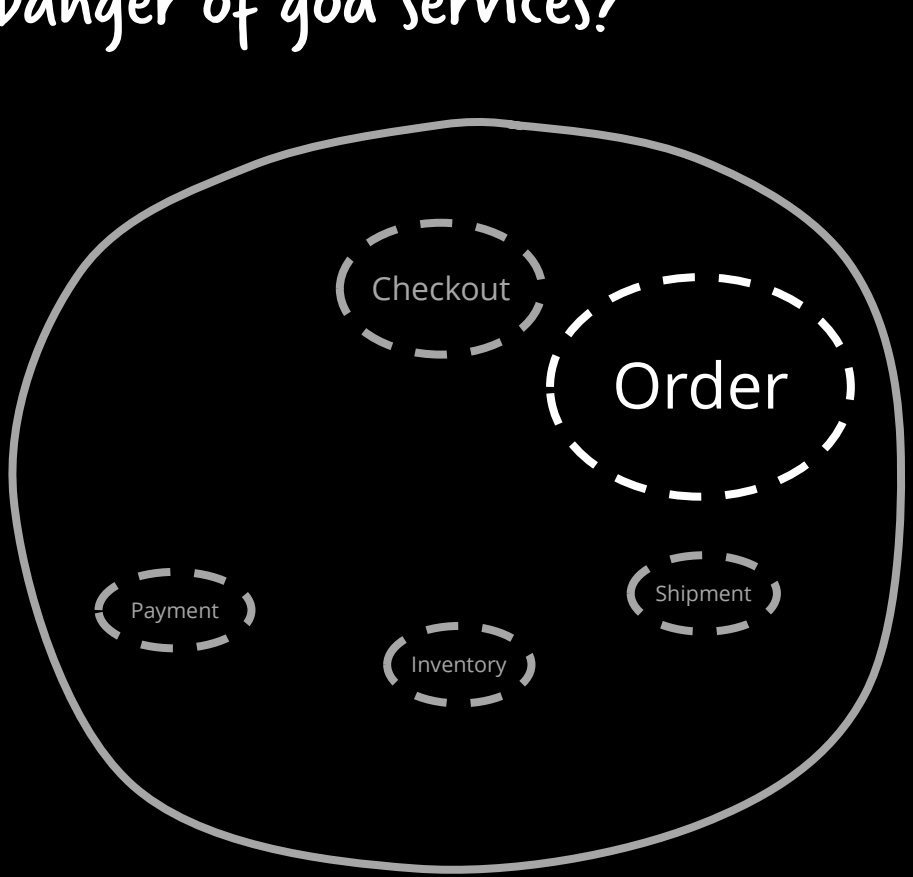
Danger of god services?



Sam Newmann

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

Danger of god services?



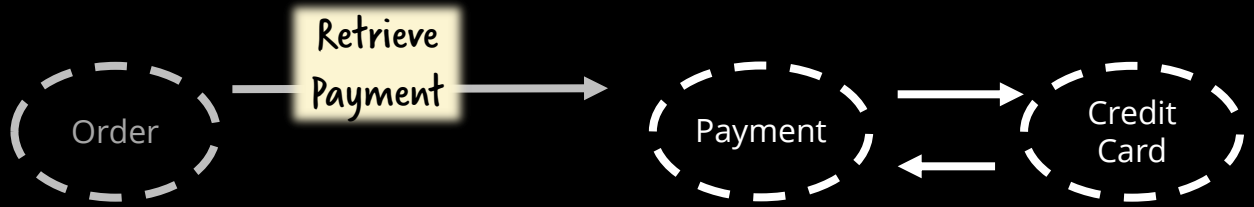
Sam Newmann

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

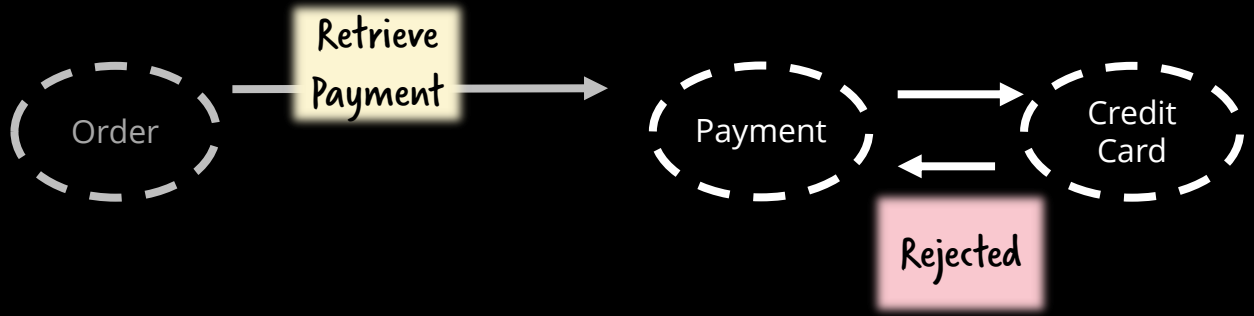
A god service is only created by bad API design!

Example

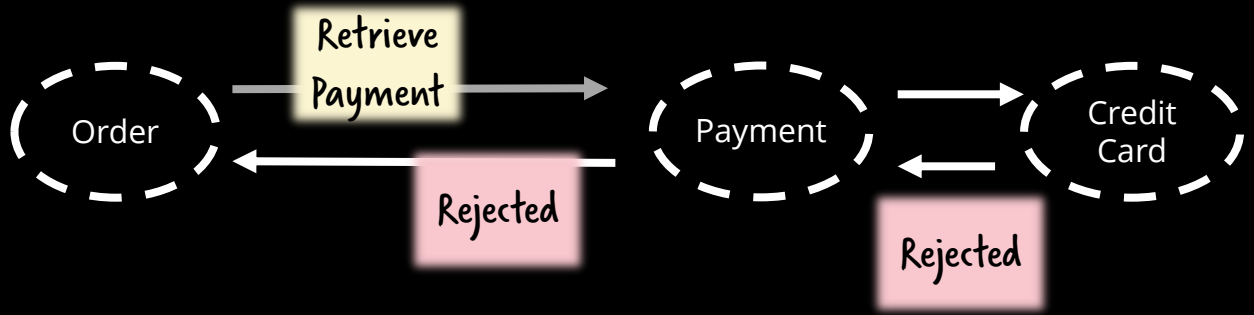
Example



Example

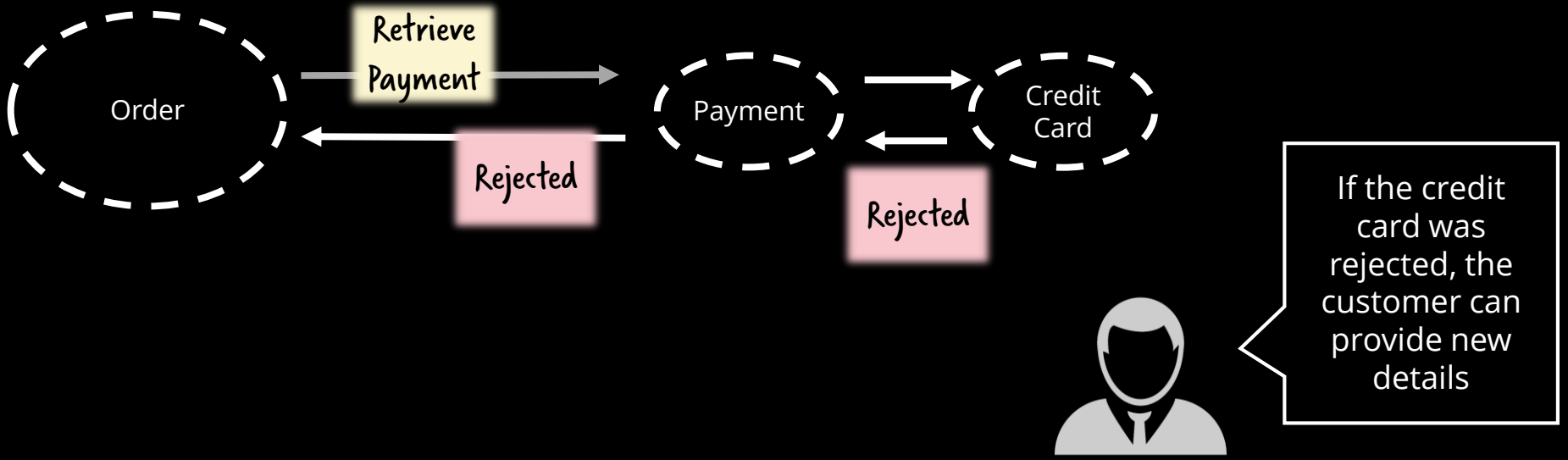


Example



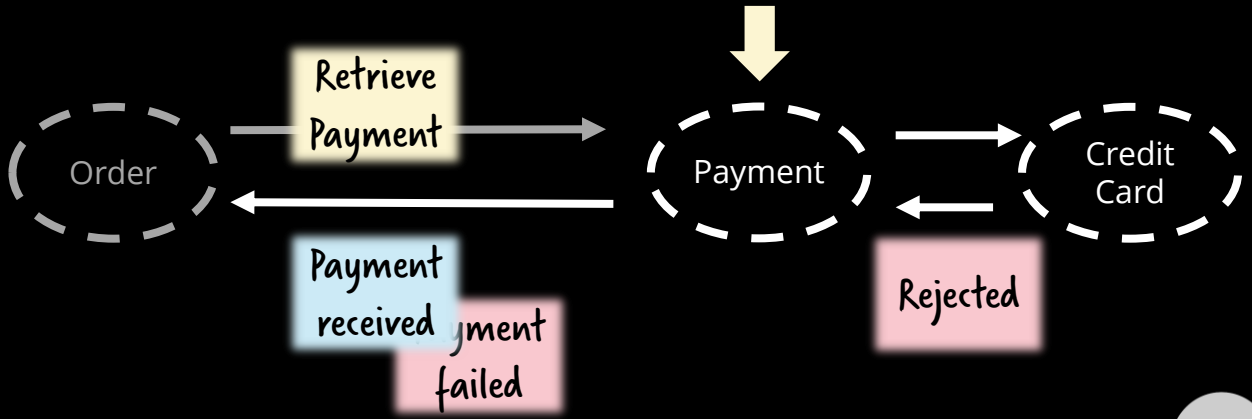
If the credit card was rejected, the customer can provide new details

Example



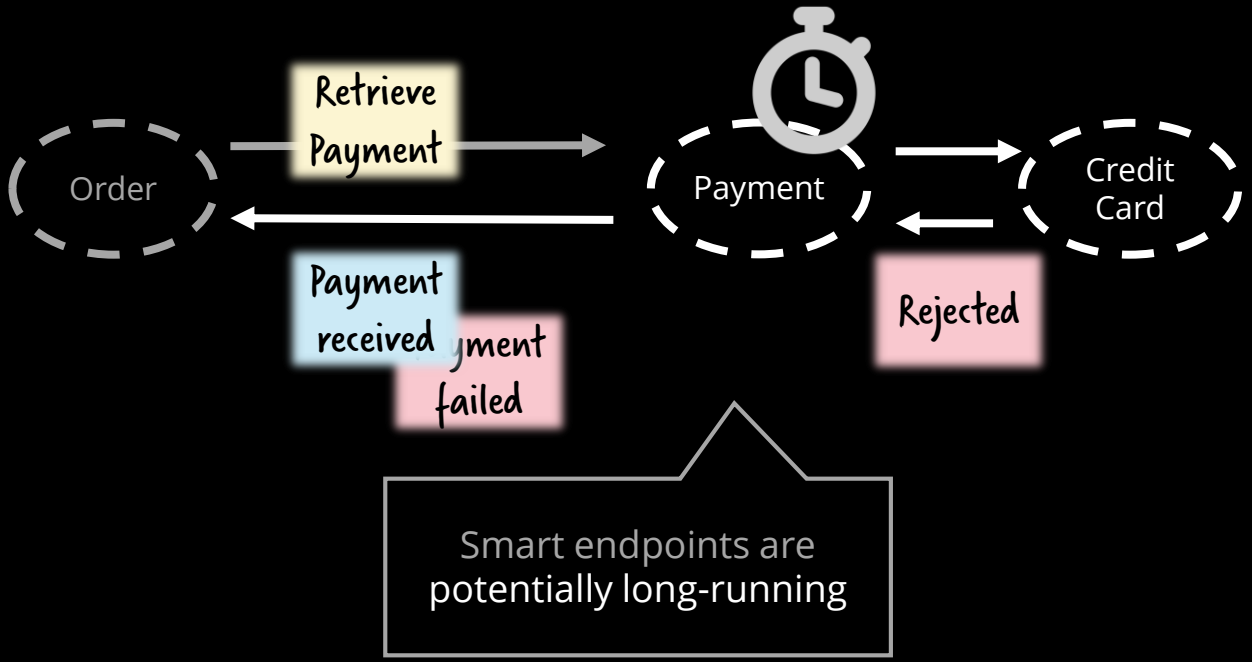
Client of **dumb endpoints** easily become a god services.

Who is responsible to deal with problems?



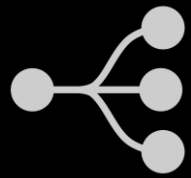
If the credit card was rejected, the customer can provide new details

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

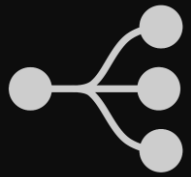
~~Complex, proprietary, heavyweight, central, developer adverse, ...~~

Avoid the wrong tools!

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

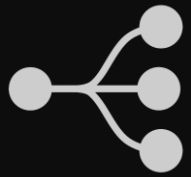
Death by properties panel

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



Workflow engines,
state machines

It is
relevant
in modern
architectures



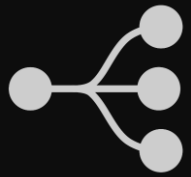
Workflow engines,
state machines

UBER CADENCE

AWS Step Functions

Silicon valley
has recognized

conductor
NETFLIX OSS



Workflow engines, state machines

UBER CADENCE

There are lightweight open source options

zeebe by Camunda

AWS Step Functions

camunda

jBPM

Activiti™

conductor NETFLIX OSS

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

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

   engine.getRuntimeServi
     "flow", Variables.put
 }
 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"));
   }
 }
```

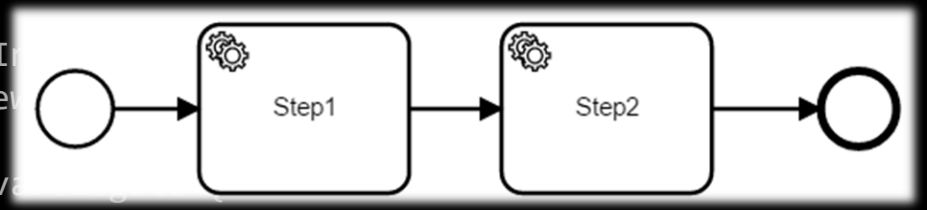
Build engine in one line of code (using in-memory H2)

```
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().startProcessI
     "flow", Variables.putValue("city", "Ne
 }
 public class SysoutDelegate implements Java
   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...

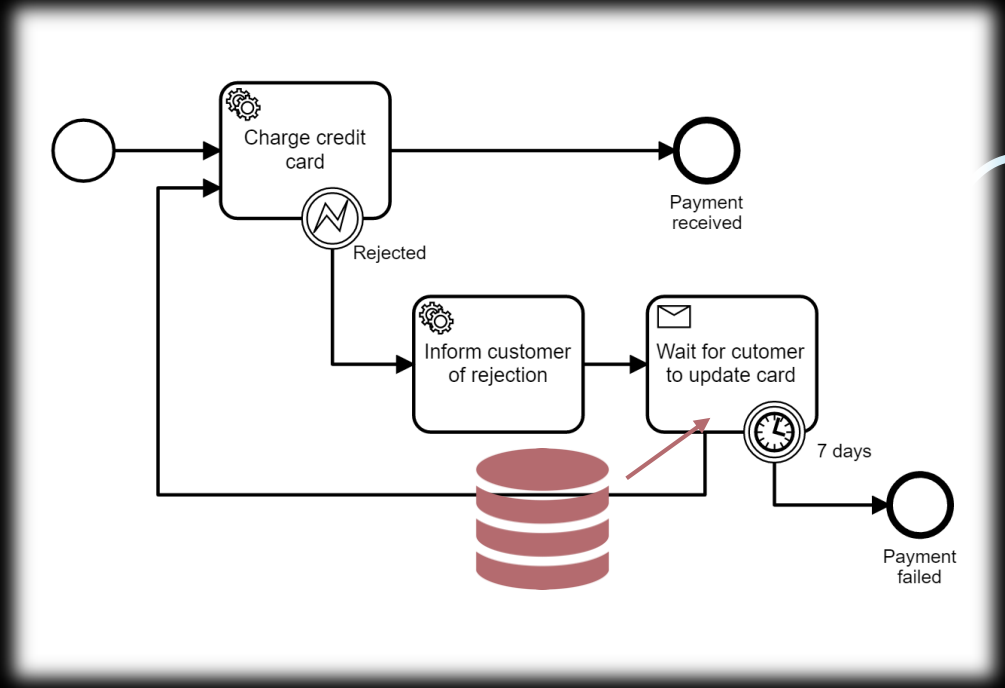
```
public static void main(String[] args) {
   ProcessEngine engine = new StandaloneInMemProcessEngineConfiguration()
     .buildProcessEngine();

   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

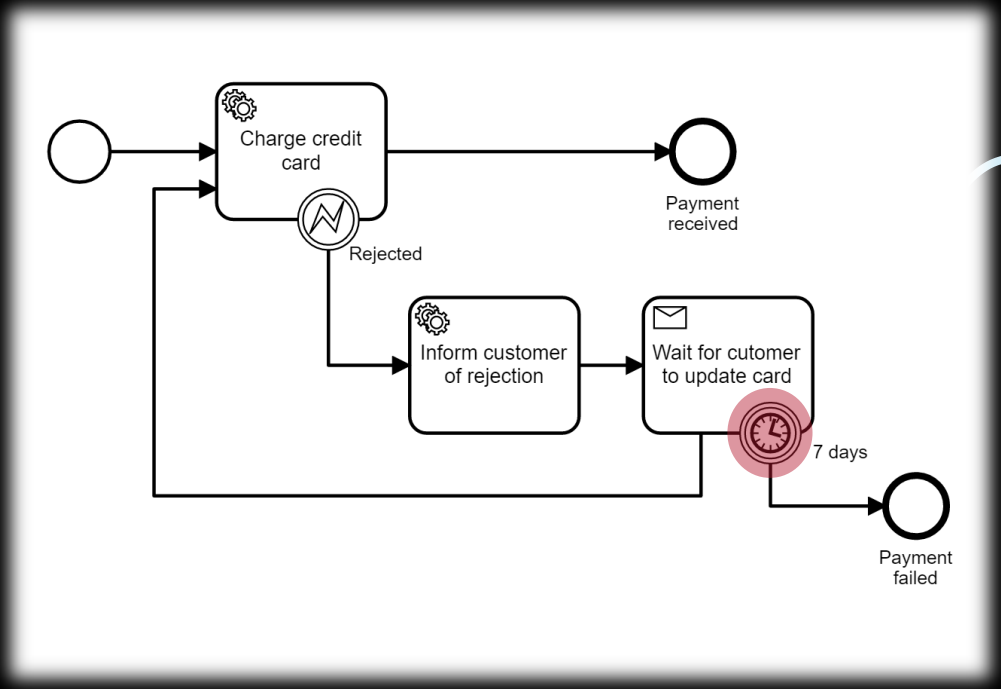
```
public static void main(String[] args) {
   ProcessEngine engine = new StandaloneInMemProcessEngineConfiguration()
     .buildProcessEngine();

   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"));
   }
 }
```

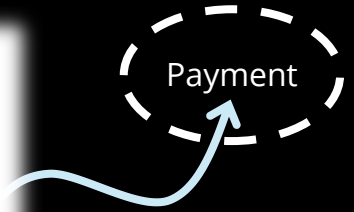
Start instances

Now you have a state machine!



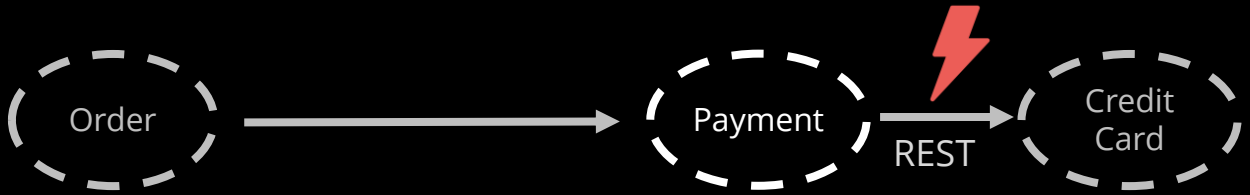
Easy to handle time



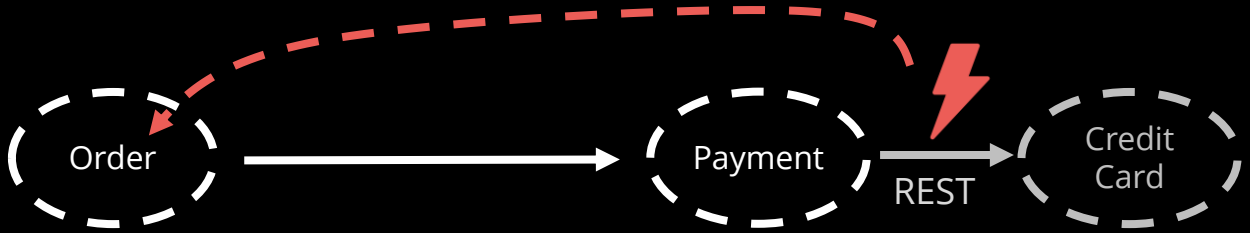


Distributed systems

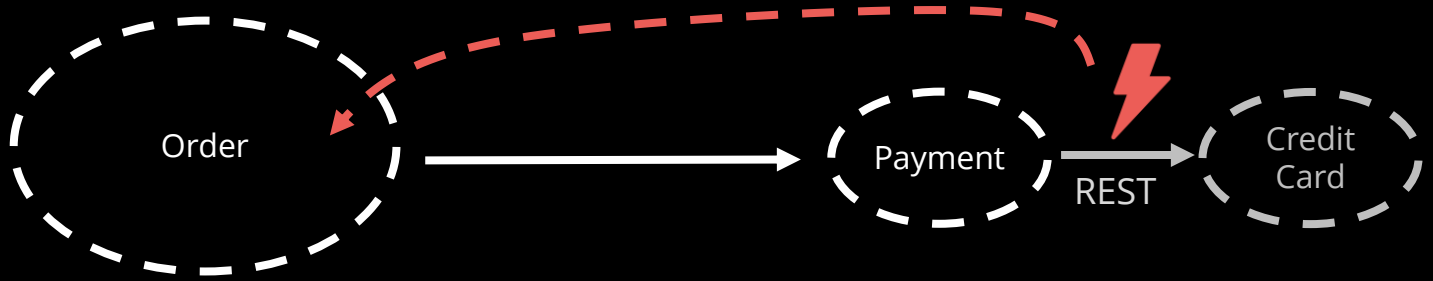
Example with synchronous communication



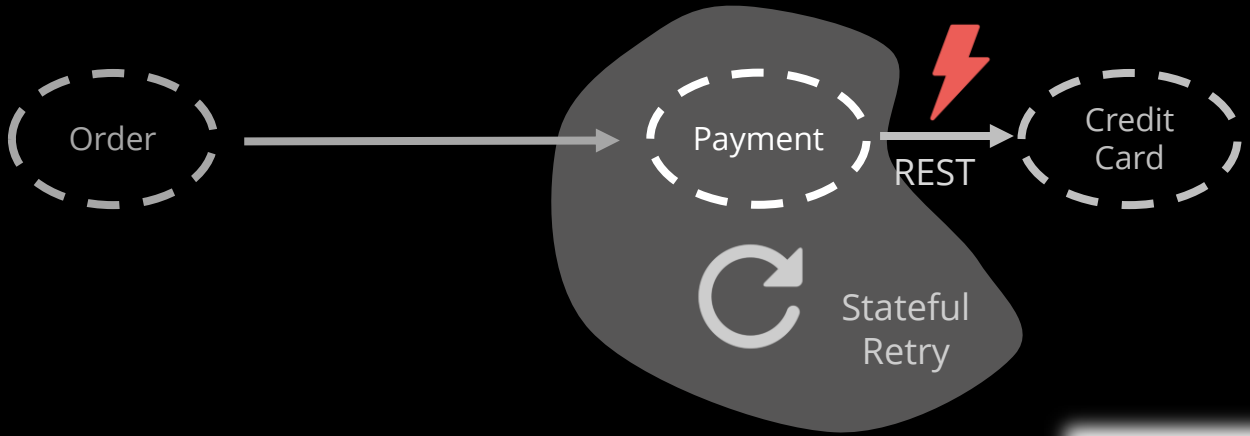
Example with synchronous communication

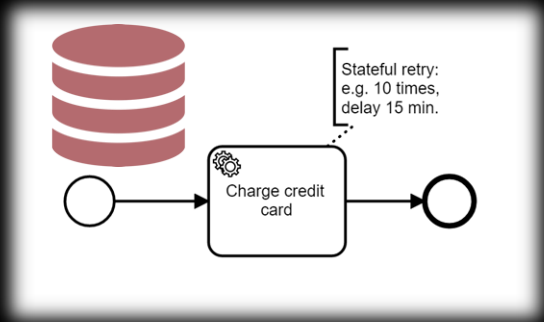


Example with synchronous communication

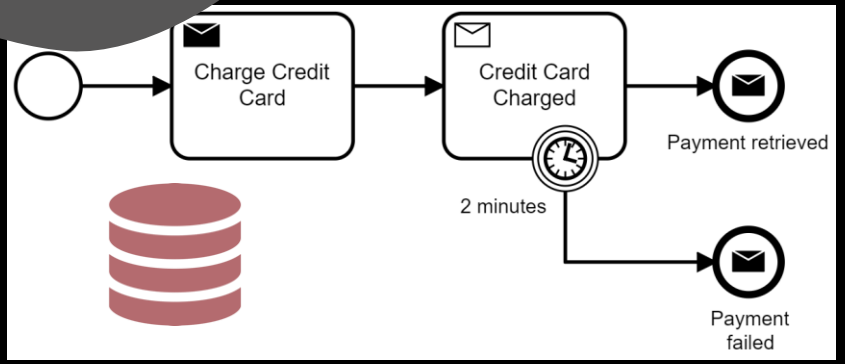


Example with synchronous communication





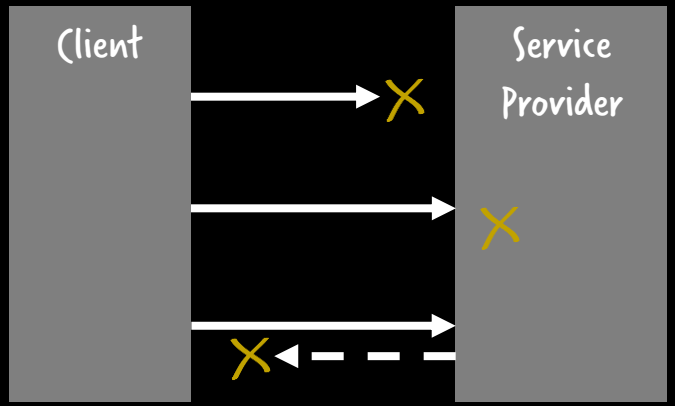
Works also for asynchronous communication



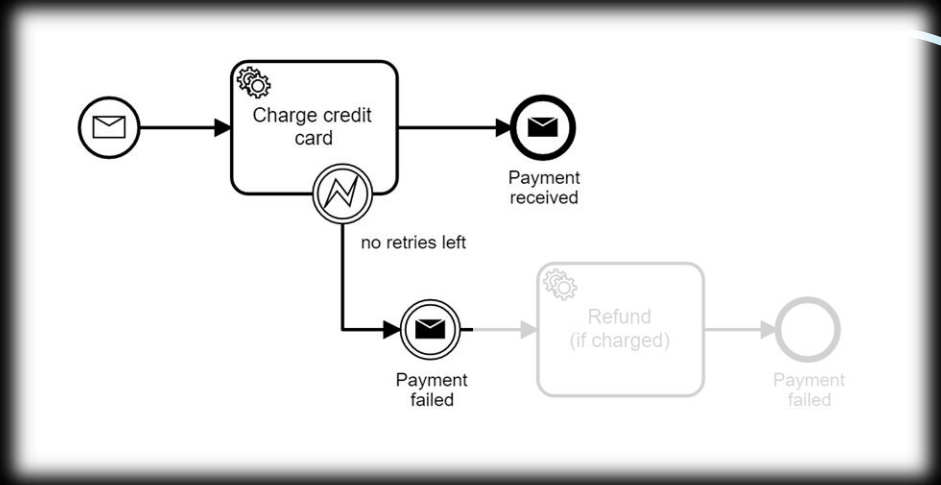
Distributed systems

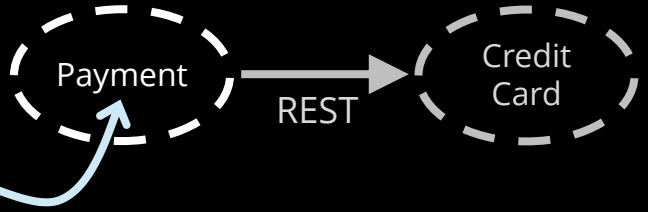
It is impossible to differentiate certain failure scenarios:

Independant of communication style!

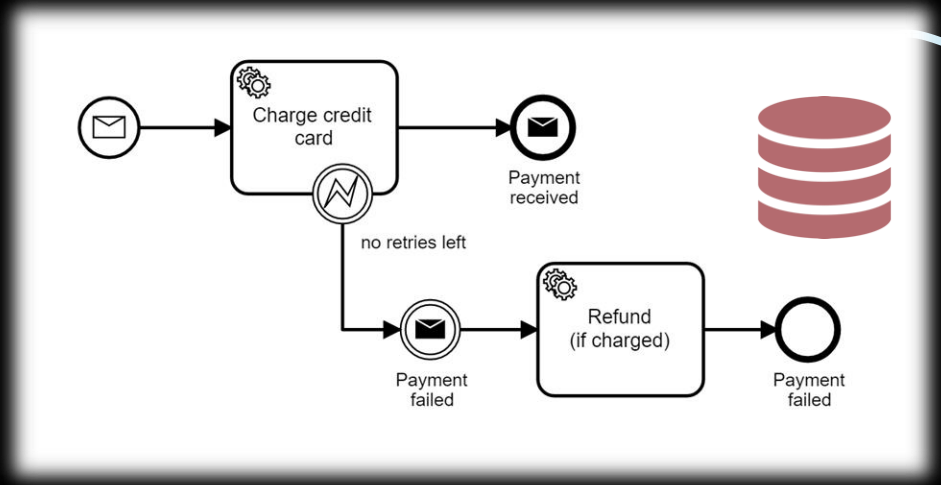


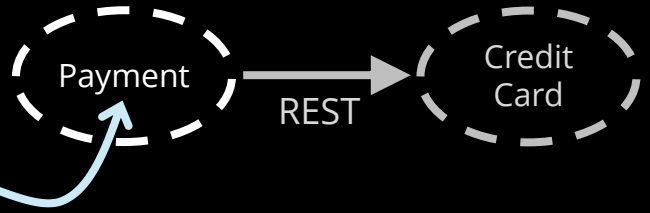
Distributed systems introduce complexity you have to tackle!





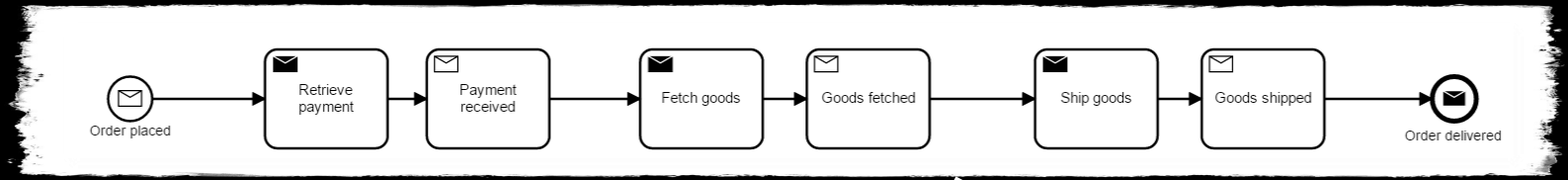
Distributed systems introduce complexity you have to tackle!

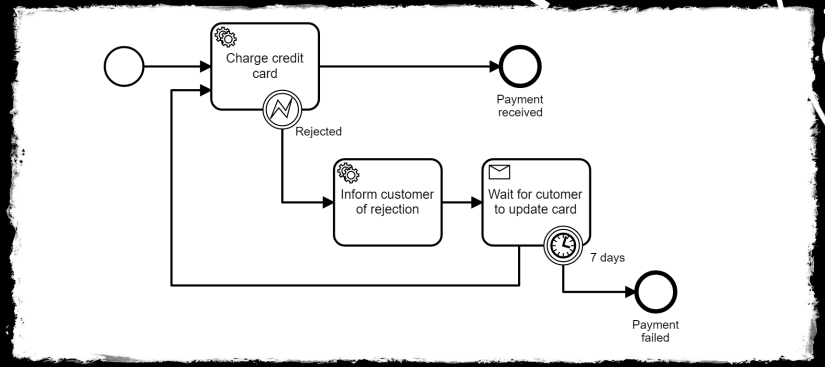


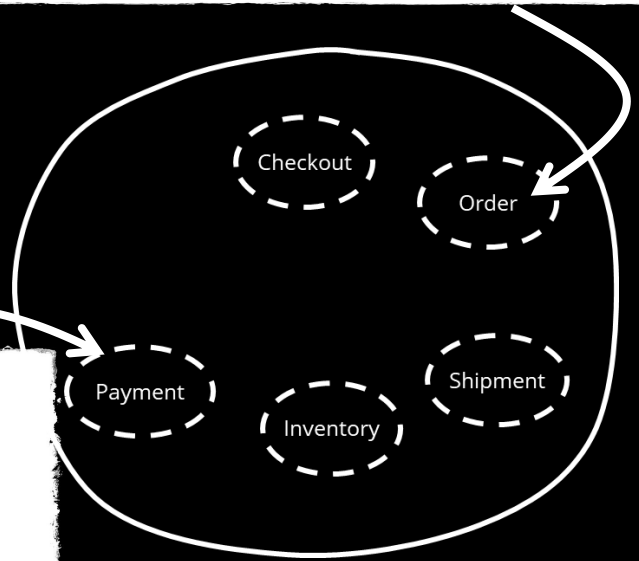


The service can be
long running.
You get a better API and less gods

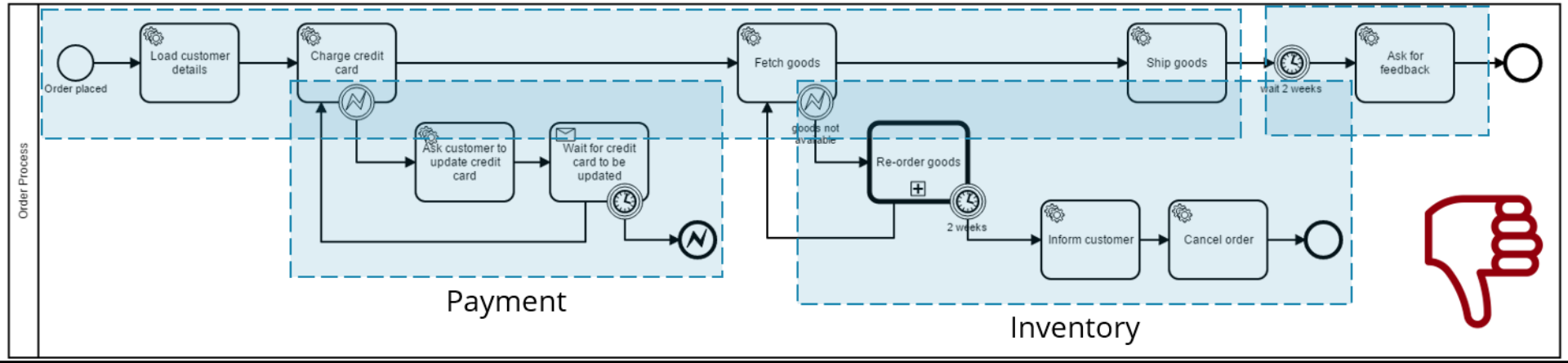
Workflows live inside service boundaries







No BPM(N) monoliths



https://blog.bernd-ruecker.com/avoiding-the-bpm-monolith-when-using-bounded-contexts-d86be6308d8

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

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

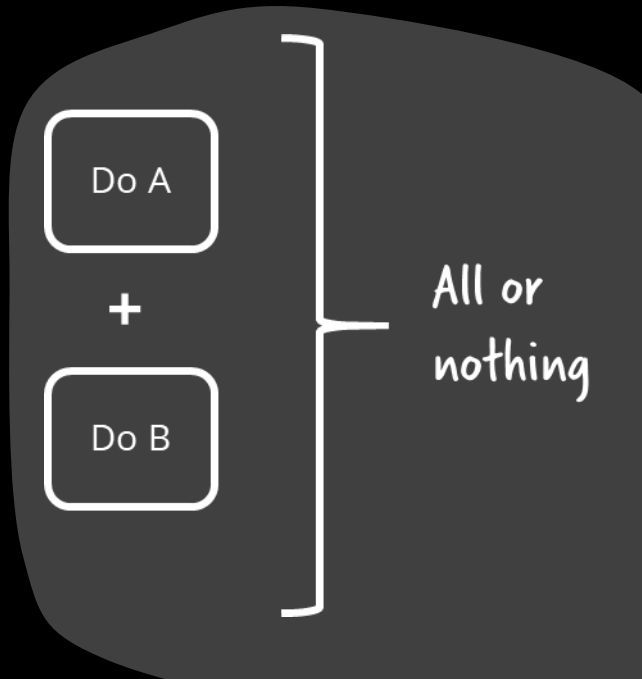
Instead, applications are built using different techniques which do not provide the same transactional guarantees but still meet the needs of their businesses.

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

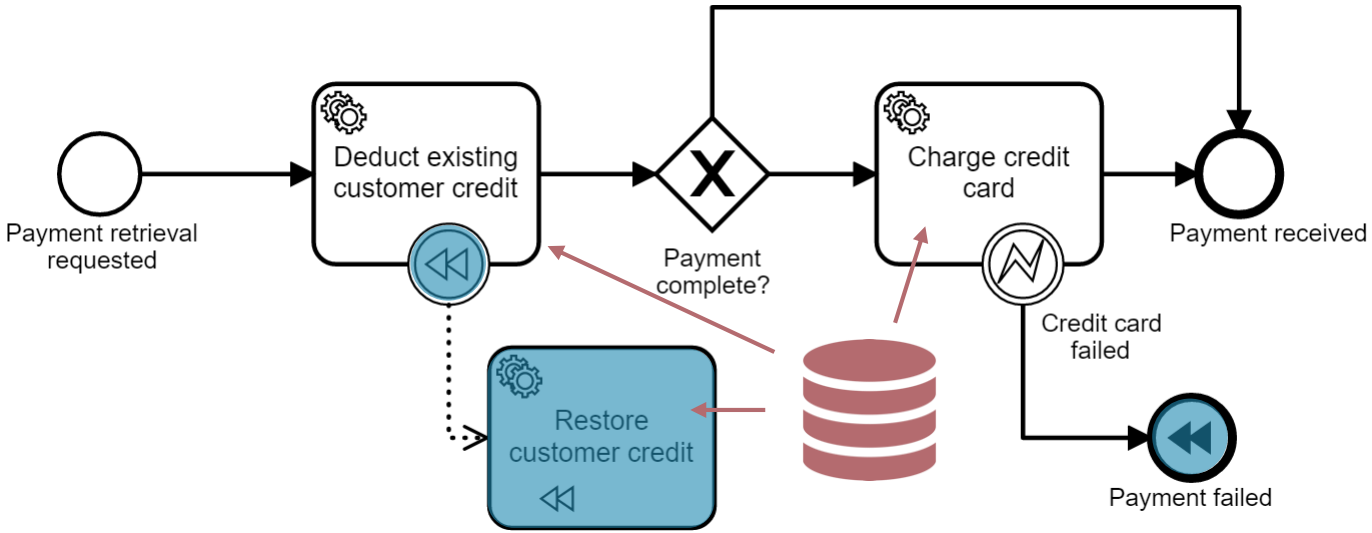
Pat Helland

Distributed Systems Guru
Worked at Amazon,
Microsoft & Salesforce

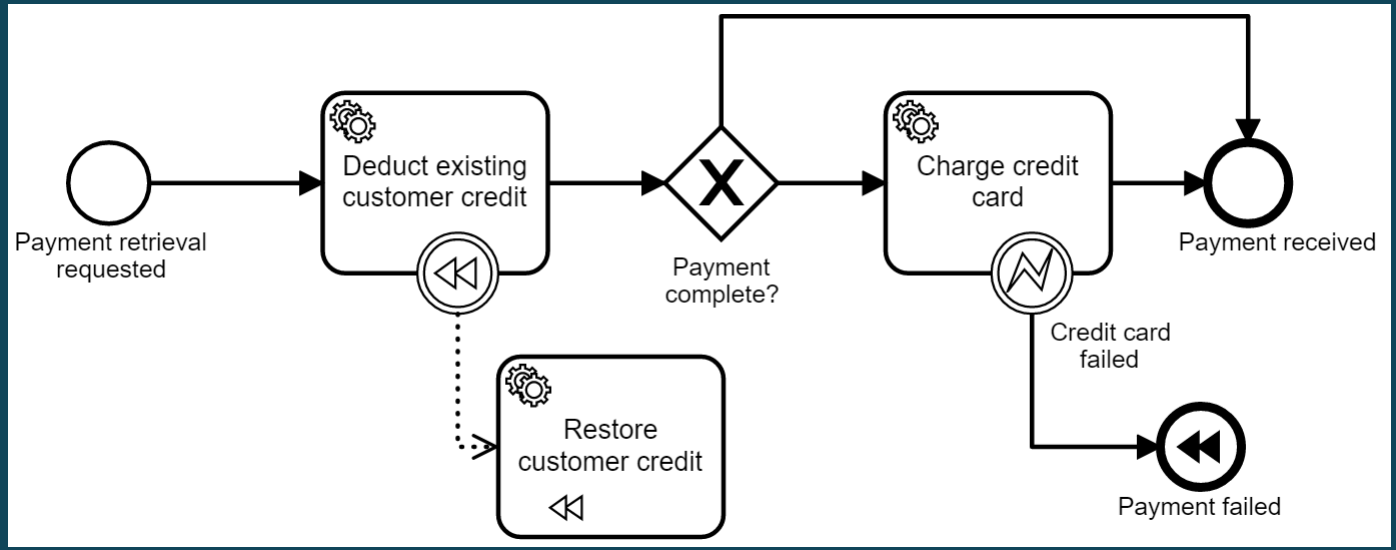


Distributed transactions using compensation *

*aka Saga pattern



Homework:
Try to do this purely event-driven!



Send to: mail@berndruecker.io

Biz Dev Ops

improve communication

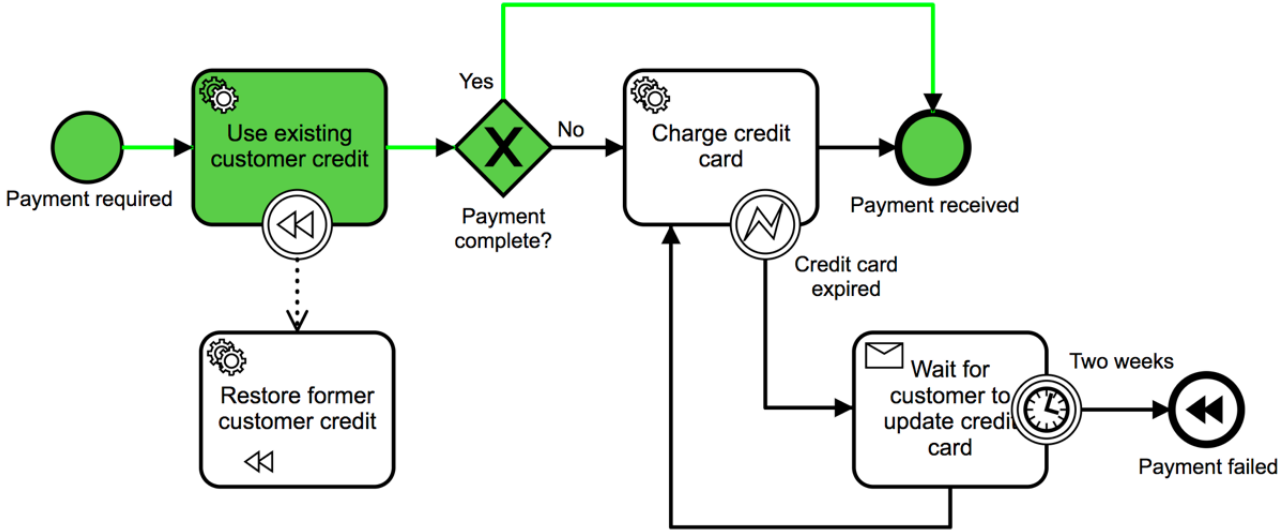
improve communication

Leverage state machine & workflow engine

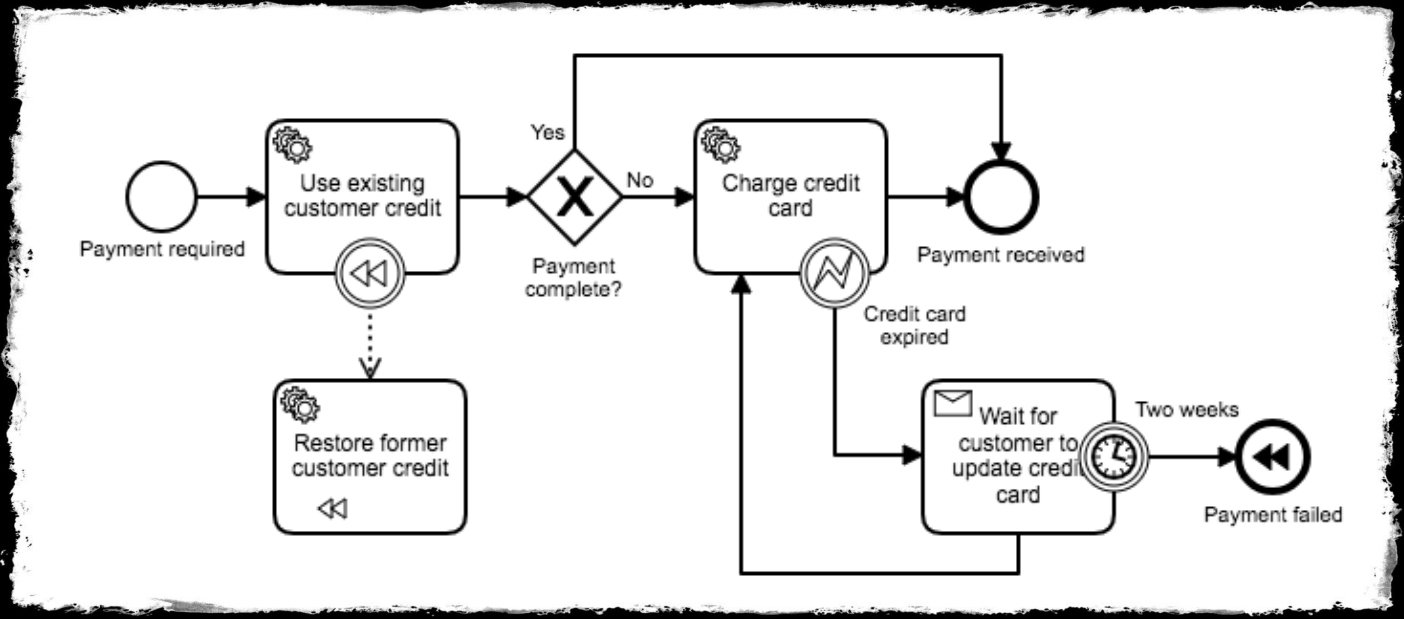
Living documentation

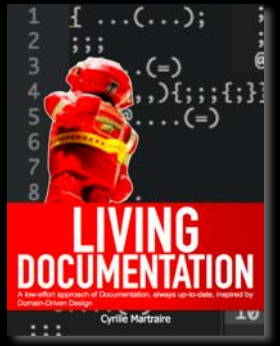
Visibility in testing

Visual HTML reports for test cases



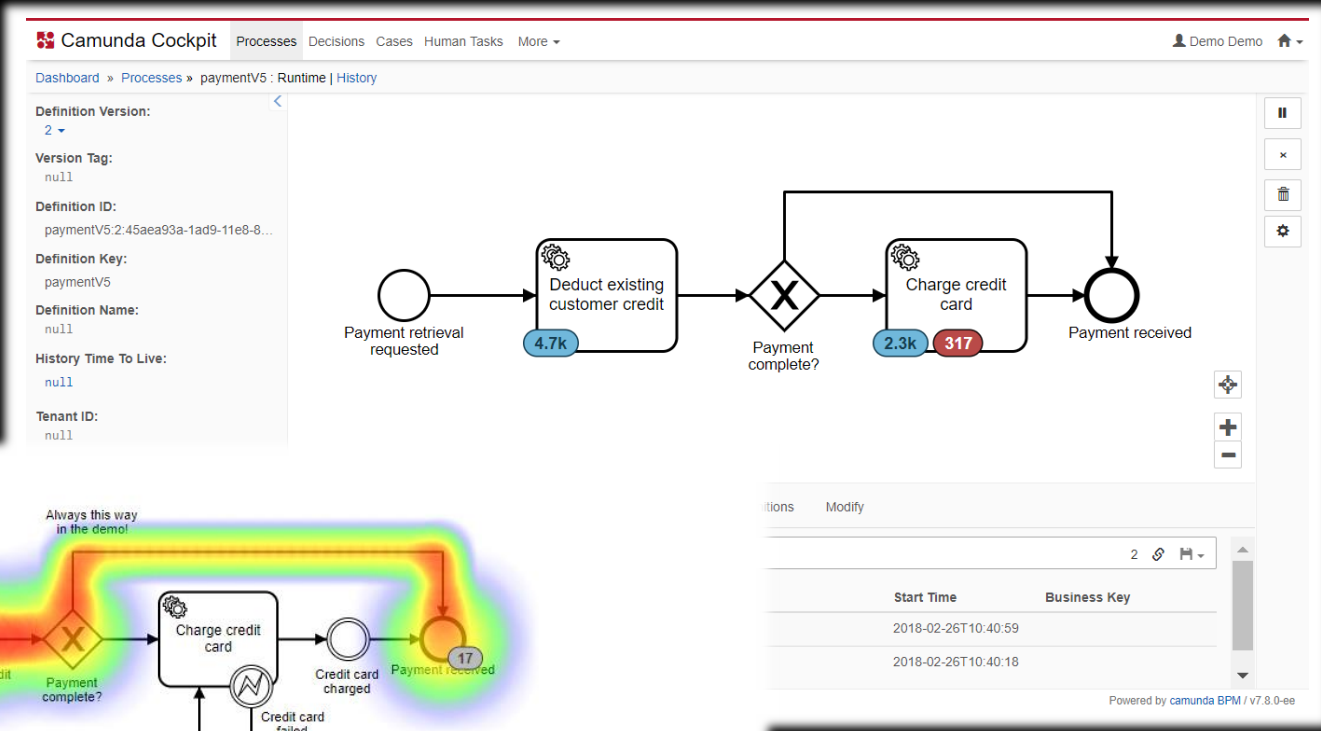
Living documentation for long-running behaviour

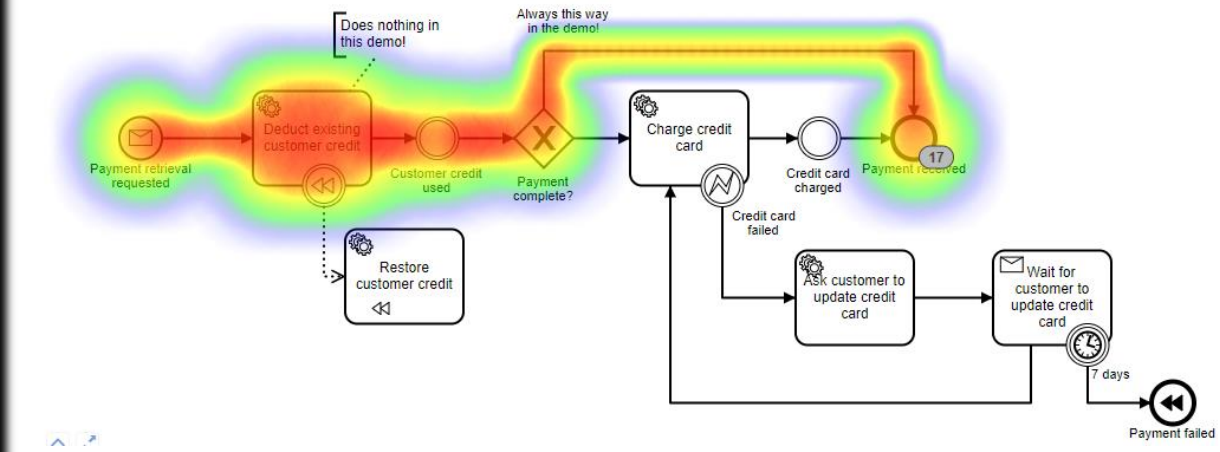


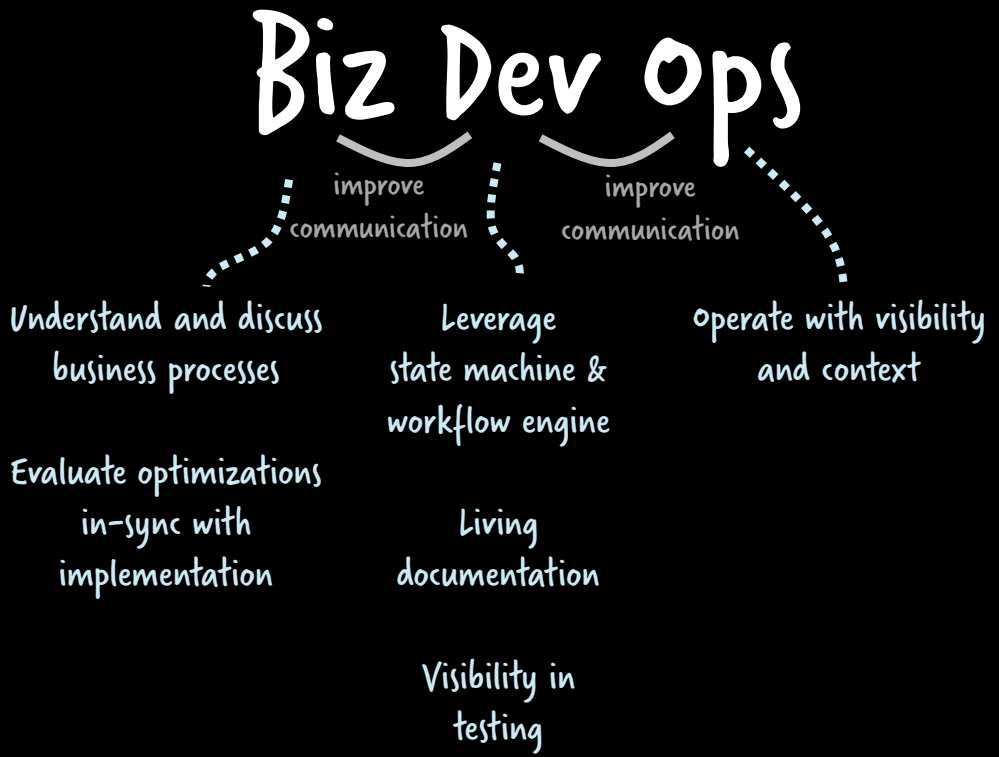


Proper operations

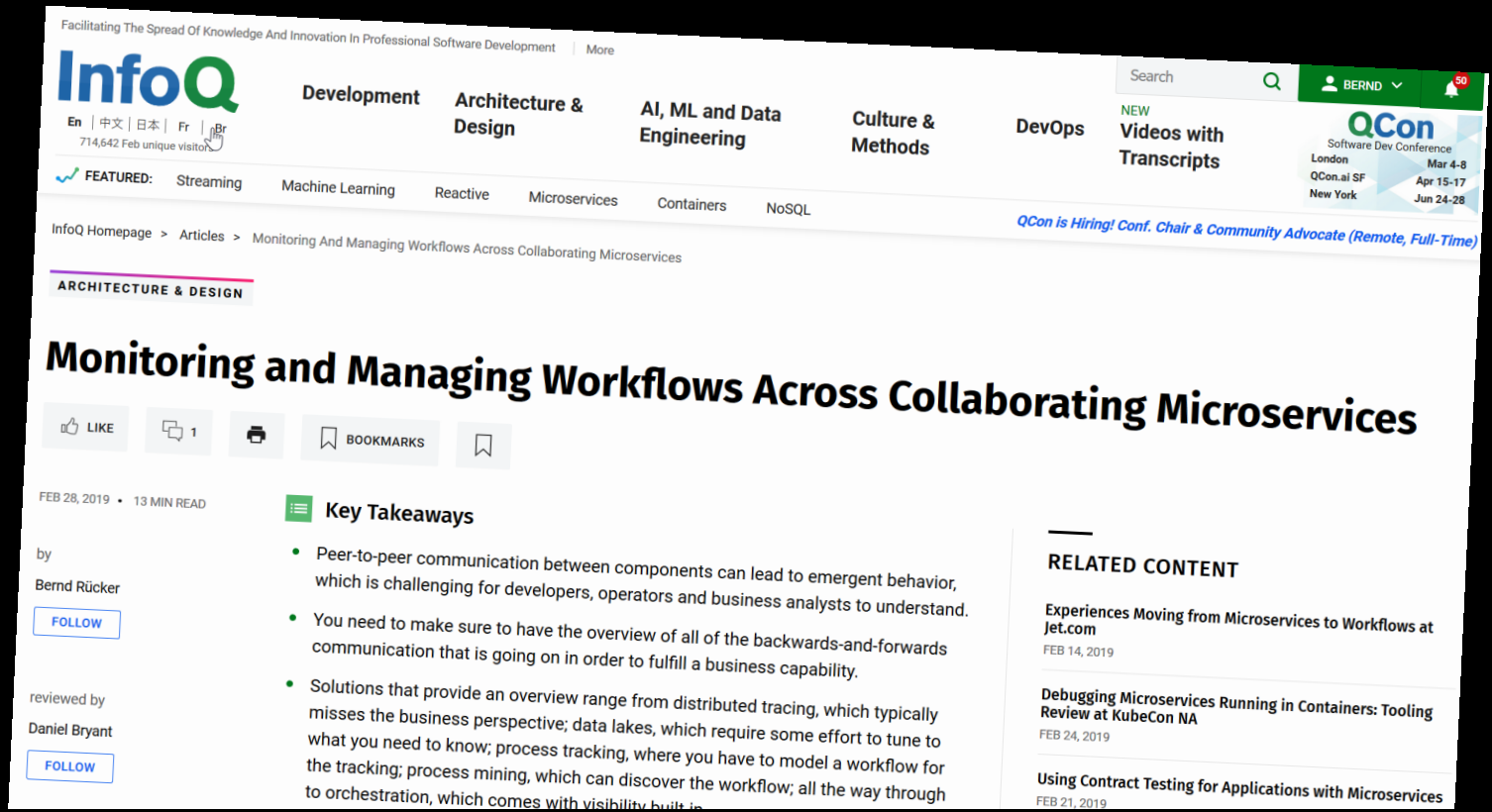
Visibility + Context





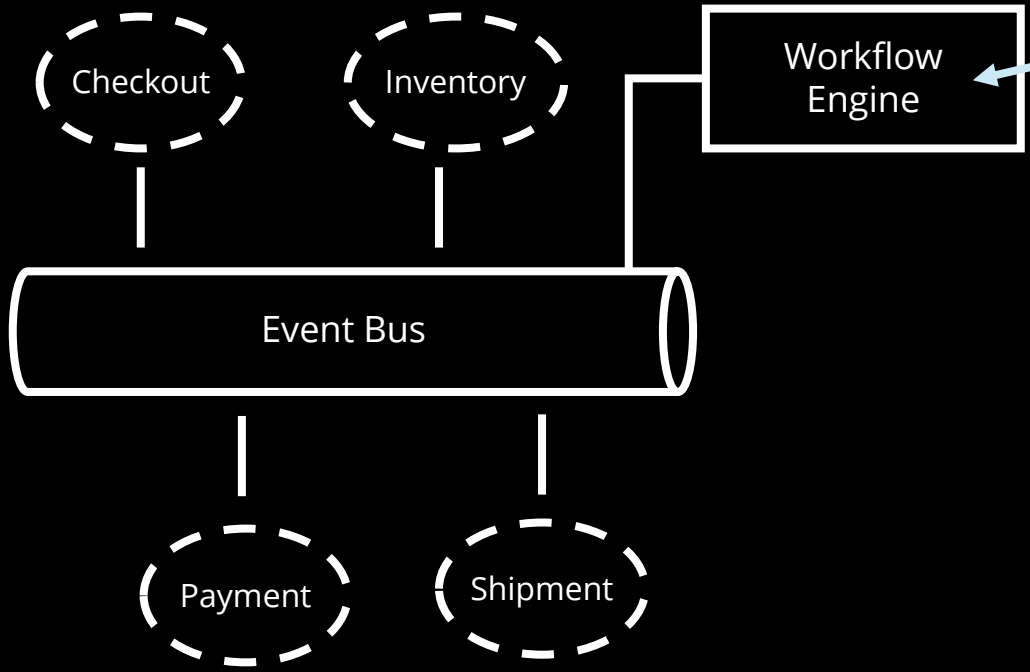

Biz Dev Ops
improve communication
improve communication
Understand and discuss business processes
Evaluate optimizations in-sync with implementation
Leverage state machine & workflow engine
Living documentation
Visibility in testing
Operate with visibility and context

Monitoring Workflows Across Microservices



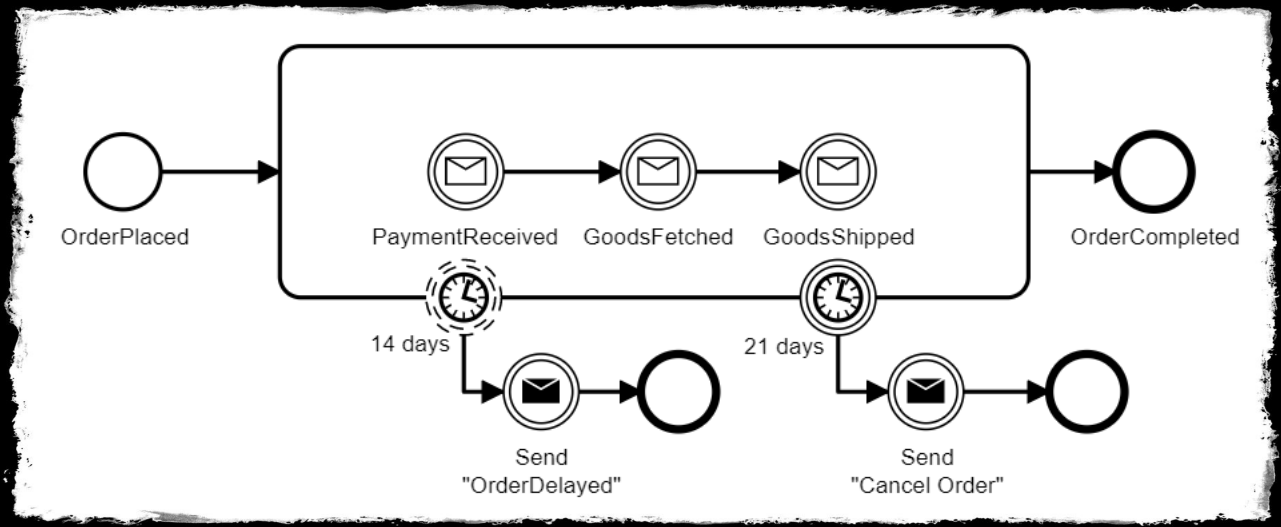
https://www.infoq.com/articles/monitor-workflow-collaborating-microservices

Tracking

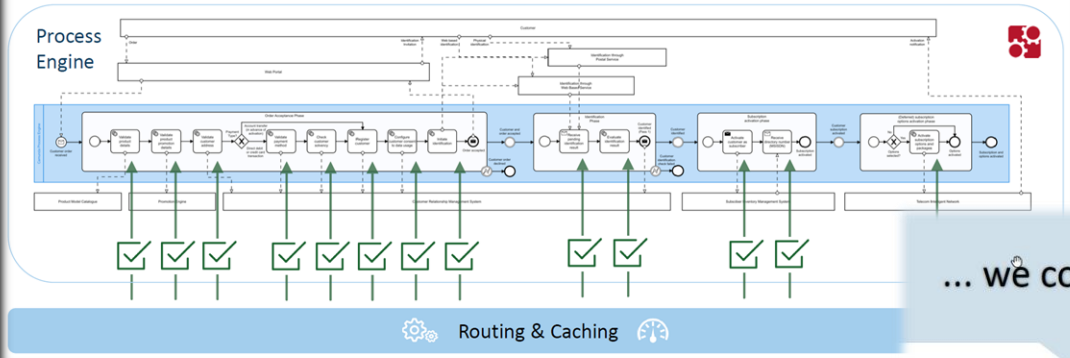


https://www.confluent.io/kafka-summit-sf18/the_big_picture

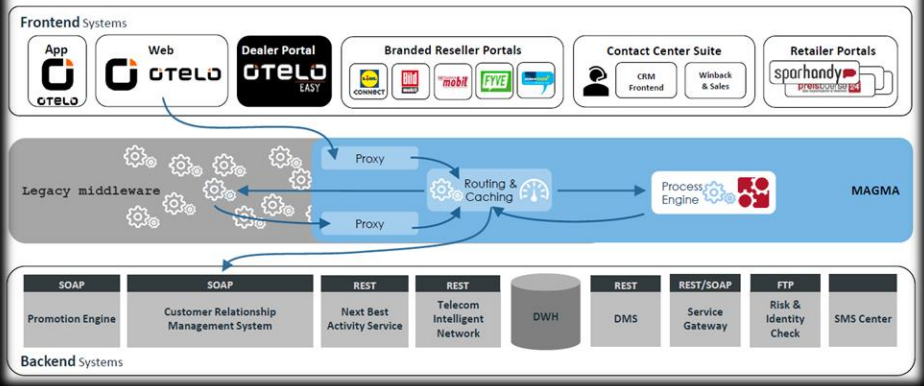
Journey towards more orchestration



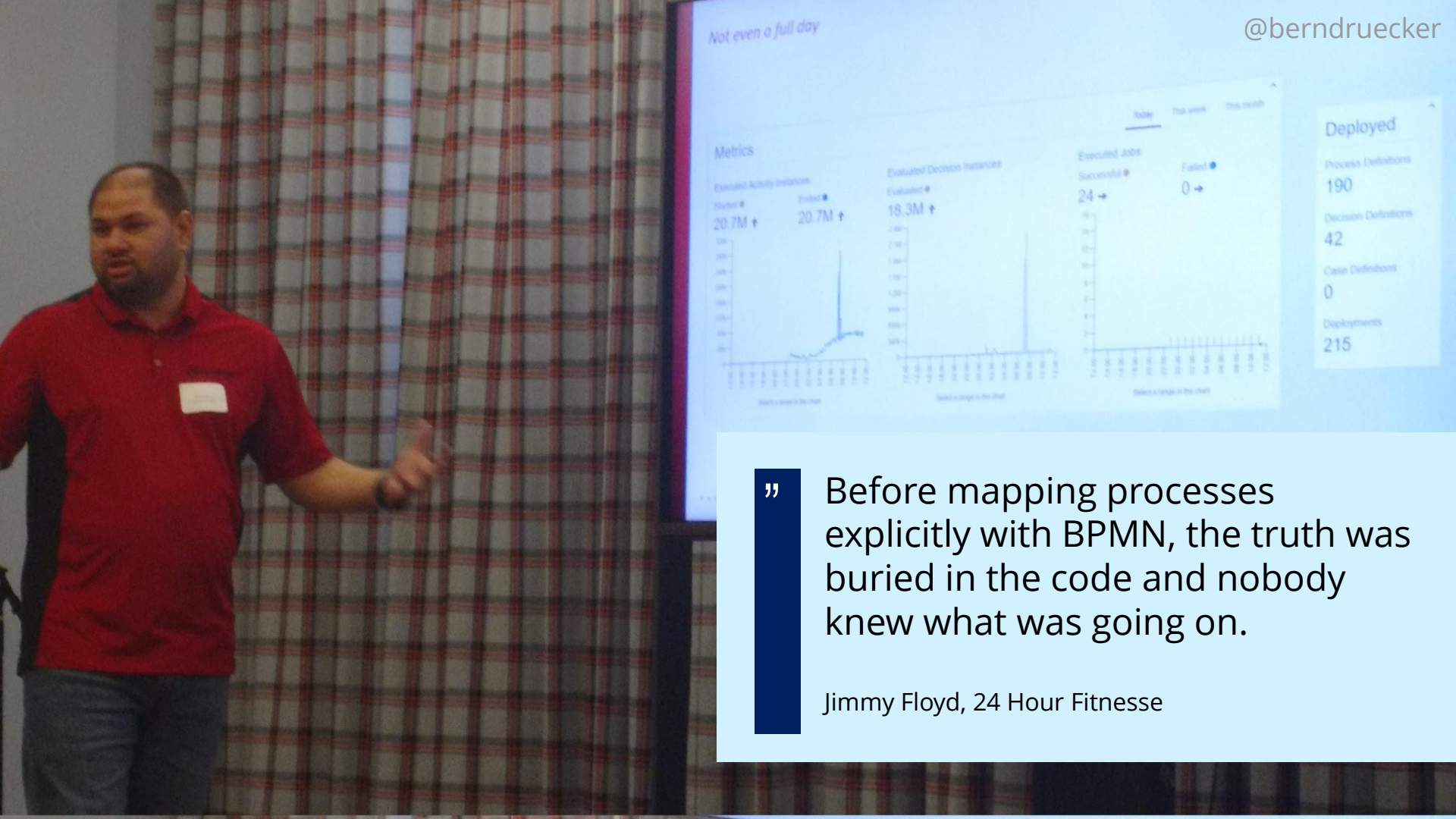
Using our monitoring abilities for the migration ...



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



Vodafone, Liongate & WDW
Presented at CamundaCon Berlin 2018



> Before mapping processes explicitly with BPMN, the truth was buried in the code and nobody knew what was going on.
>
> Jimmy Floyd, 24 Hour Fitnesse

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?

...

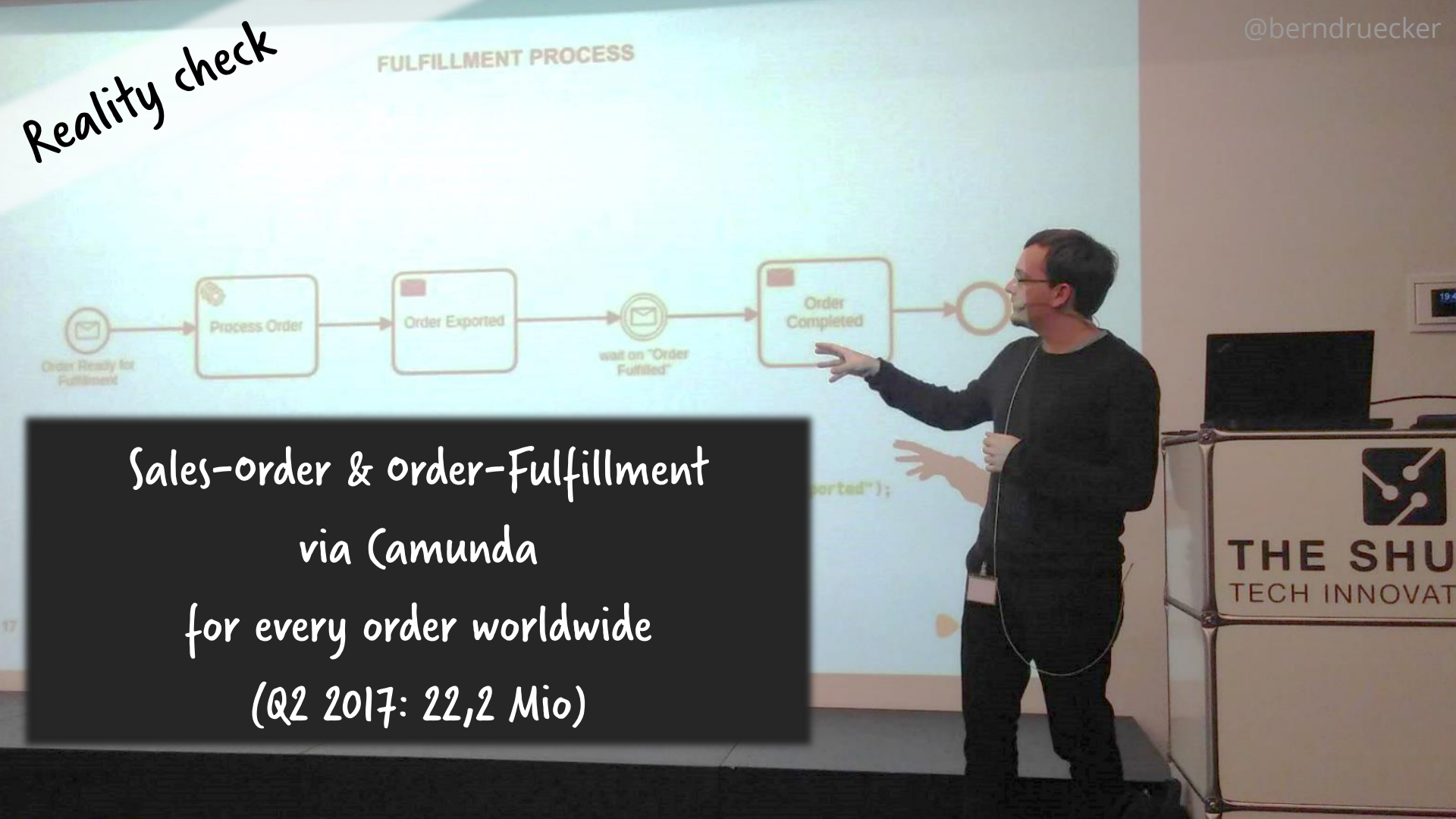
the system we are replacing uses a complex peer-to-peer choreography that requires reasoning across multiple codebases to understand.

https://medium.com/@sitapati/node-js-client-for-zeebe-microservices-orchestration-engine-72287e4c7d94

Lightweight workflow engines are great – don't DIY*

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

Reality check



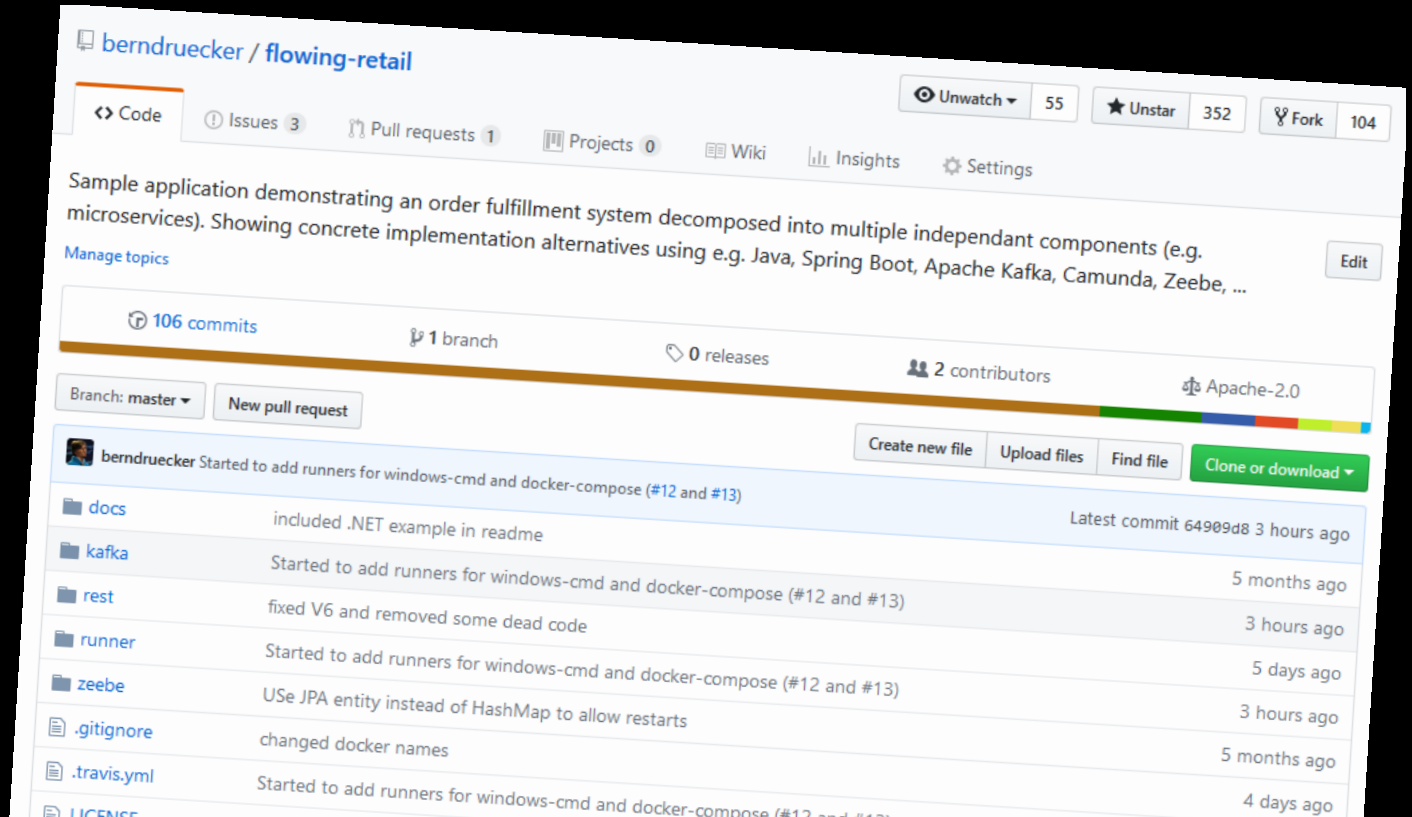
Sales-order & order-Fulfillment

via Camunda

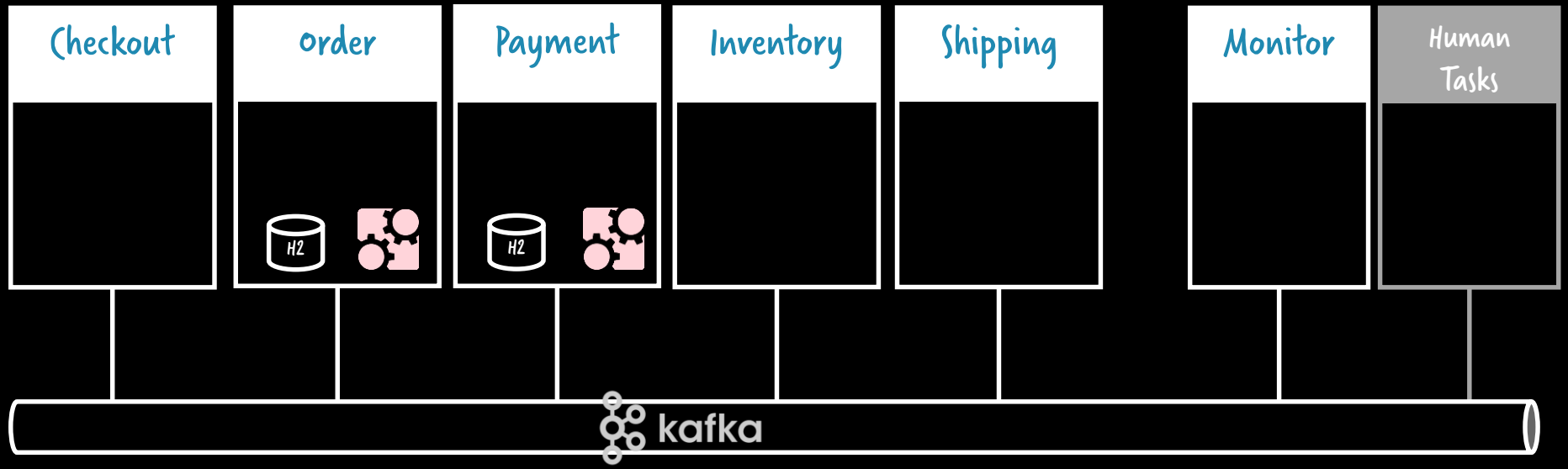
for every order worldwide

(Q2 2017: 22,2 Mio)

Code, code, code...



Event-driven example



https://github.com/berndruecker/flowing-retail/

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
@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-integrationand-how-to-avoid-them.html

InfoQ ueue

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

THENEWSTACK

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