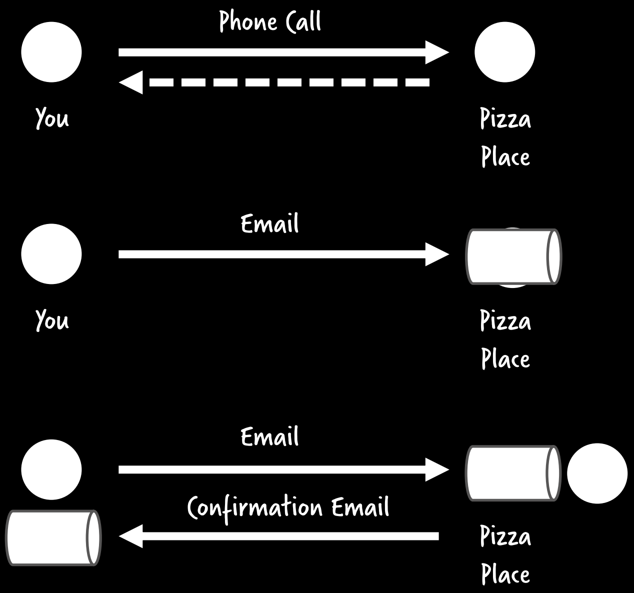
Loosely or Lousily Coupled?

Understanding Communication Patterns in Microservices Architectures

@berndruecker

Let's talk about food

How does ordering Pizza work?

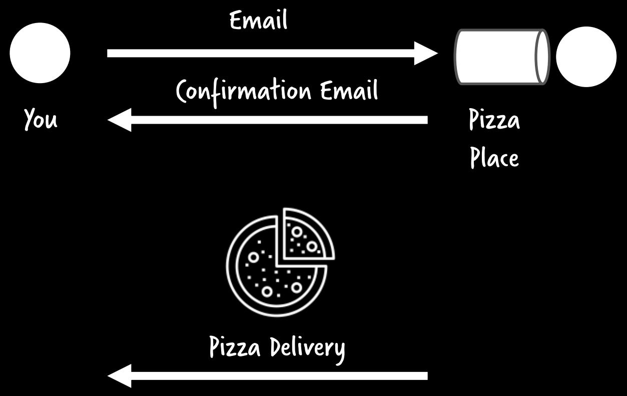


Synchronous blocking communication
Feedback loop (ack, confirmation or rejection)
Temporal coupling (e.g. busy, not answering)

Asynchronous non-blocking communication
No temporal coupling

A feedback loop might make sense
(ack, confirmation or rejection)

Feedback loop != result



Email

Confirmation Email

You

Pizza Place

Pizza Delivery

Feedback (ACK, confirmation, rejection)

Result

Synchronous blocking behavior for the result?

Bad user experience

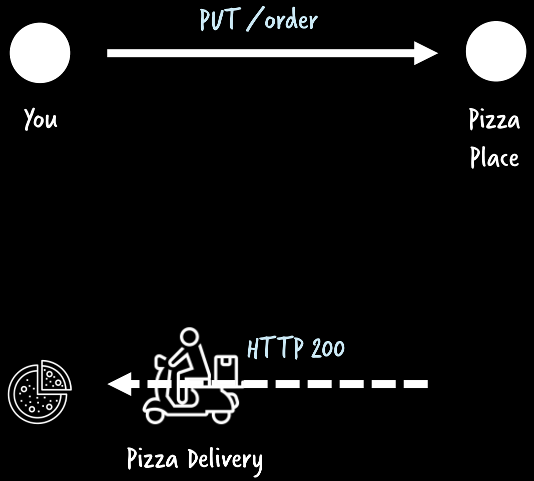
Does not scale well

Scalable Coffee Making

https://www.enterpriseintegrationpatterns.com/ramblings/18_starbucks.html

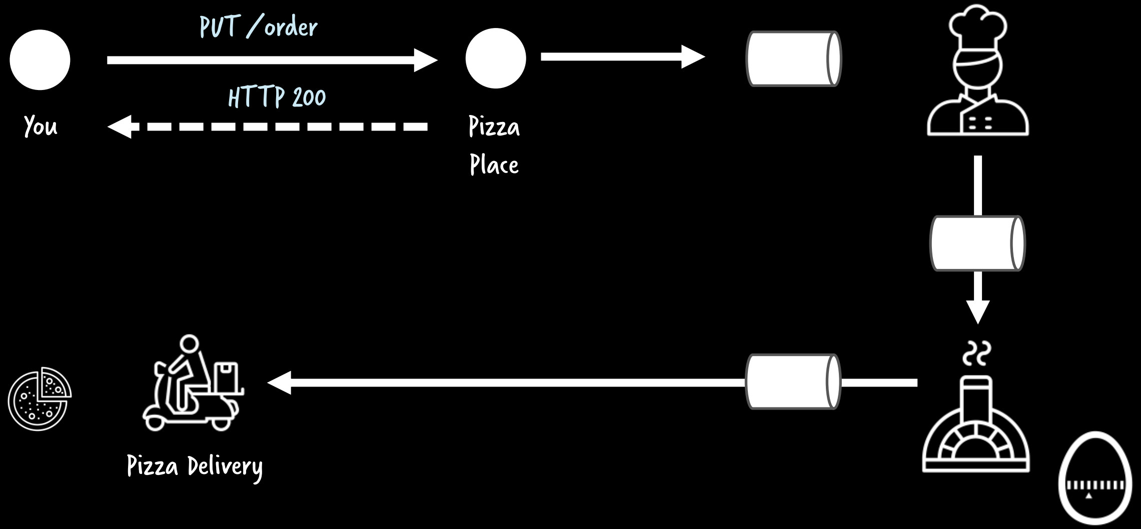
Photo by John Ingle

Synchronous results?



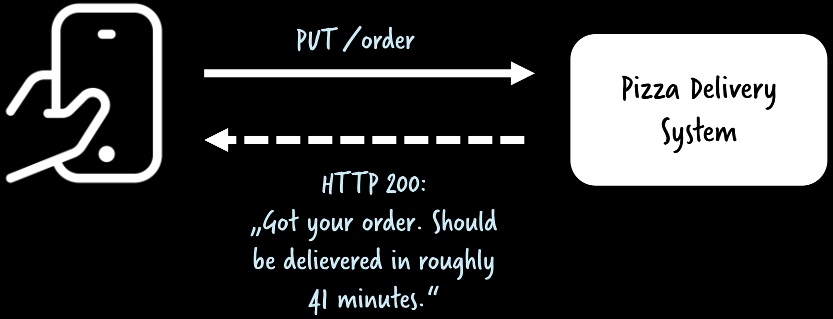
The task of Pizza making is long running

only the first communication step is synchronous



The task of Pizza making is long running

Example: Build a pizza ordering app

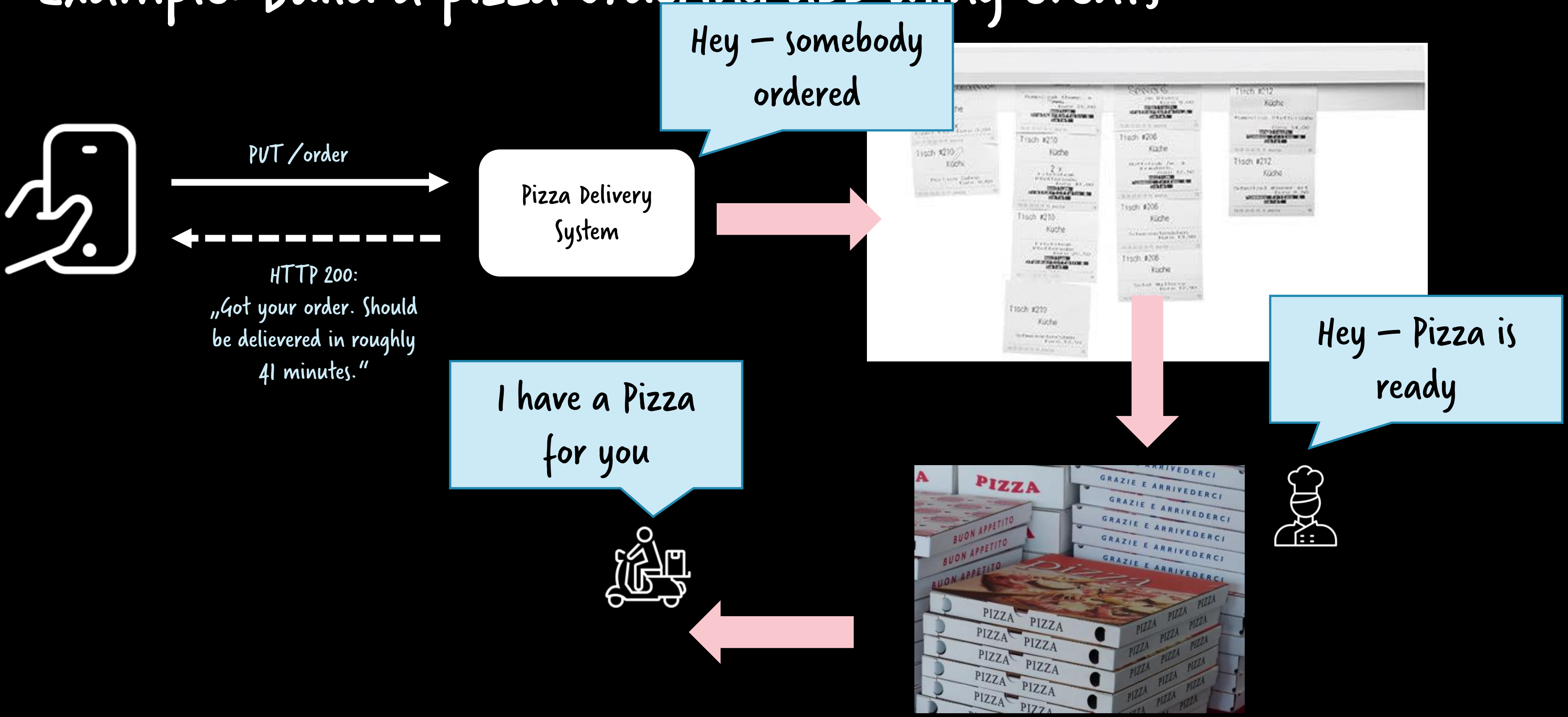


PUT /order

Pizza Delivery System

HTTP 200:
„Got your order. Should be delievered in roughly 41 minutes."

Example: Build a pizza ordering app using events



PUT /order

Pizza Delivery System

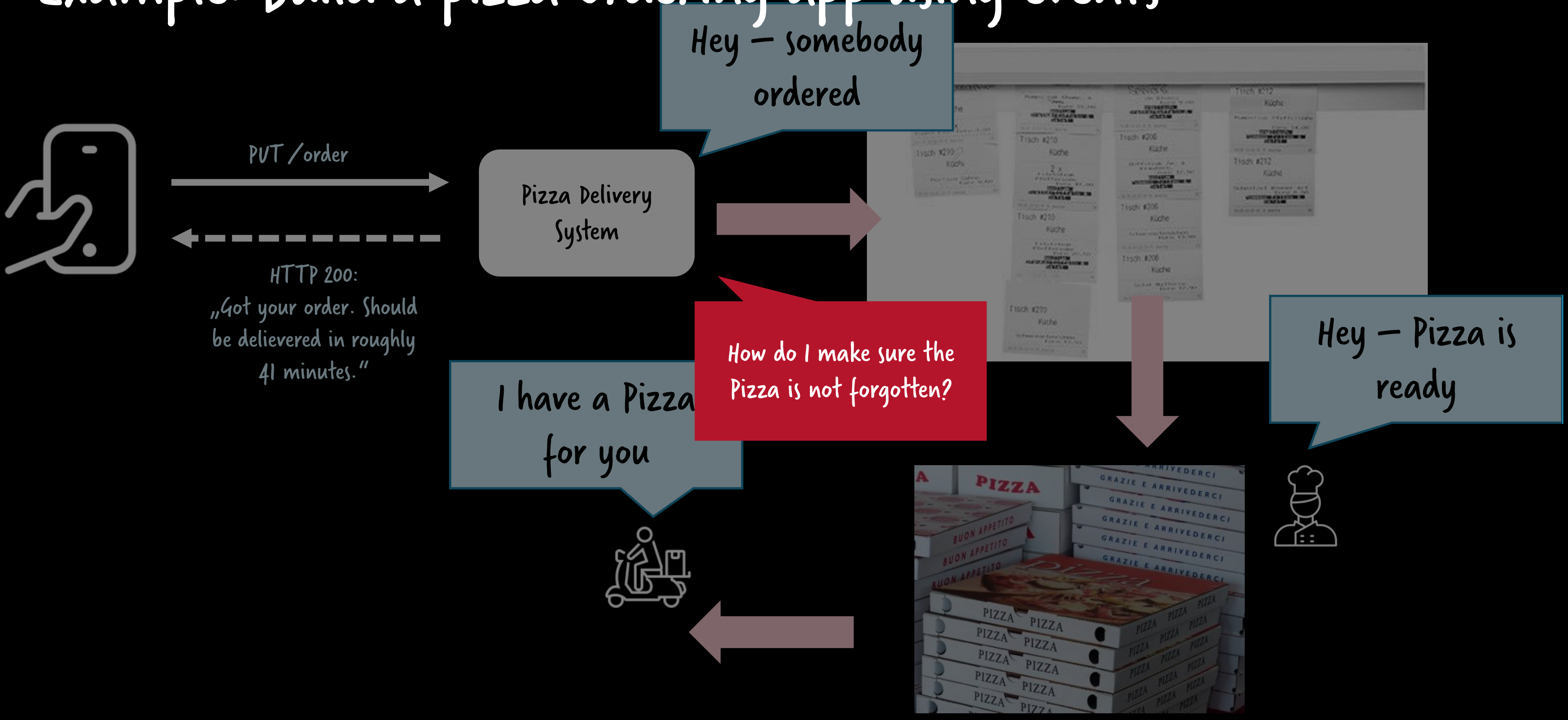
HTTP 200:
„Got your order. Should be delievered in roughly 41 minutes.“

Hey – somebody ordered

Hey – Pizza is ready

I have a Pizza for you

Example: Build a pizza ordering app using events



PUT /order

Pizza Delivery System

HTTP 200:
„Got your order. Should be delievered in roughly 41 minutes."

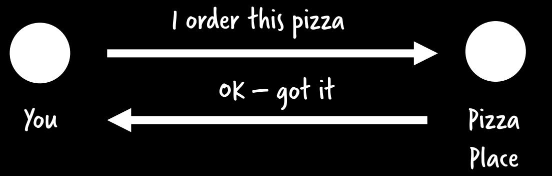
Hey – somebody ordered

Hey – Pizza is ready

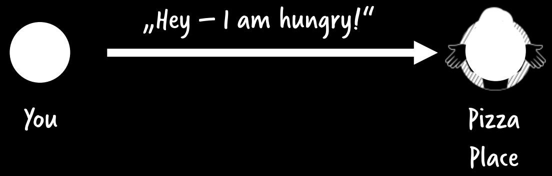
I have a Pizza for you

How do I make sure the Pizza is not forgotten?

Command vs. event-based communication



Command = Intent
Cannot be ignored
Independant of communication channel



Event = Fact
Sender can't control what happens

Definitions

Event = Something happened in the past. It is a fact.
Sender does not know who picks up the event.

Command = Sender wants s.th. to happen. It has an intent.
Recipient does not know who issued the command.

Events vs. Commands

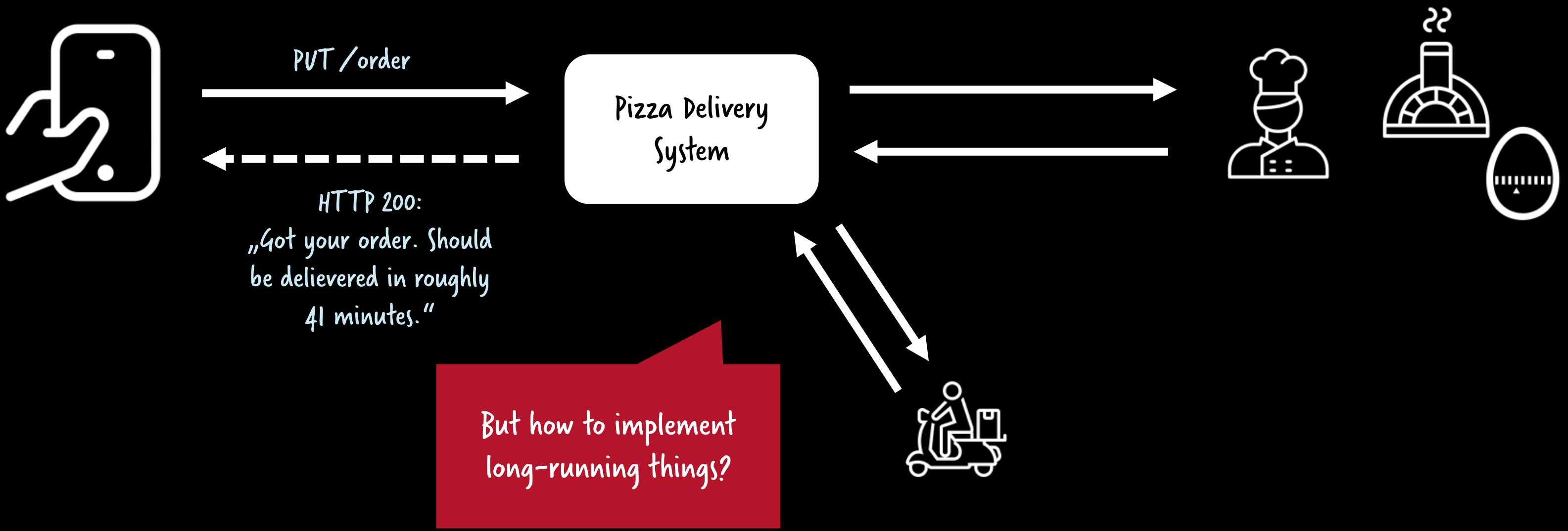
„Pizza Salmon is ready!"

I baked this pizza for Andrea. Please package it immediately and deliver it while it's hot!

Orchestrator

Command

Example: Build a pizza ordering app via orchestration

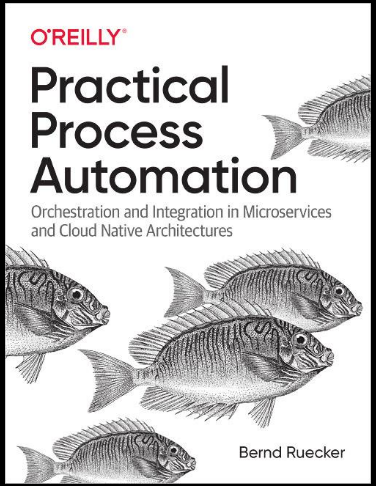


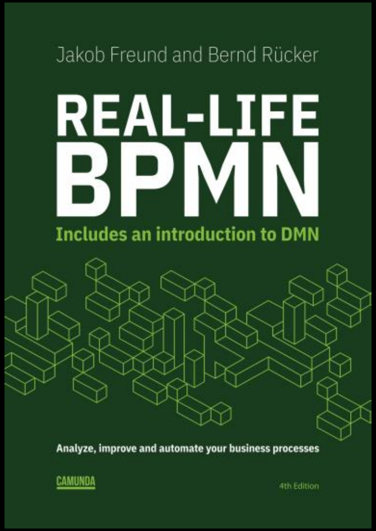
PUT /order
Pizza Delivery System
HTTP 200:
„Got your order. Should be delievered in roughly 41 minutes.“

But how to implement long-running things?

Bernd Ruecker

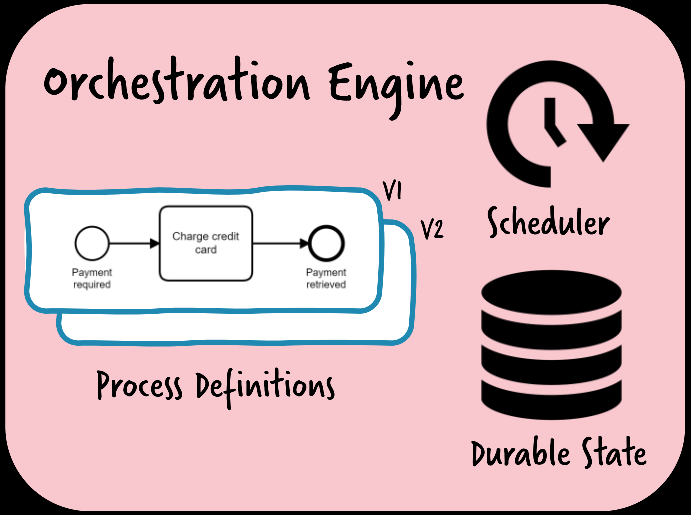
Co-founder and Chief Technologist of Camunda


O'REILLY
Practical Process Automation
Orchestration and Integration in Microservices and Cloud Native Architectures
Bernd Ruecker


Jakob Freund and Bernd Rücker
REAL-LIFE BPMN
Includes an introduction to DMN
Analyze, improve and automate your business processes
CAMUNDA
4th Edition

bernd.ruecker@camunda.com
@berndruecker
http://berndruecker.io/

An orchestration engine provides long running capabilities



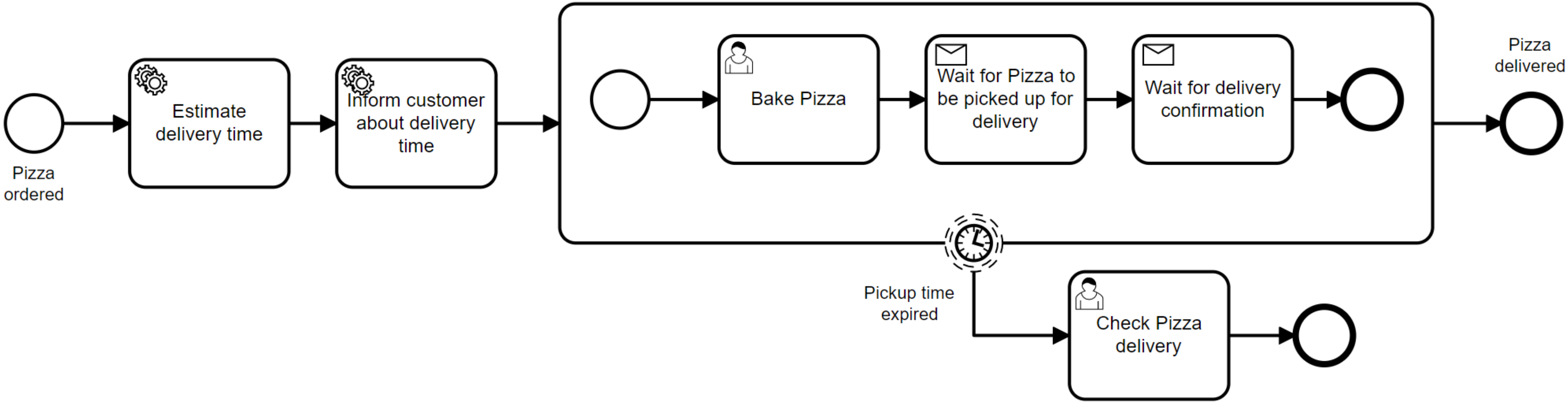
Orchestration Engine:

Is stateful

Can wait
Can retry
Can escalate
Can compensate

Provides visibility

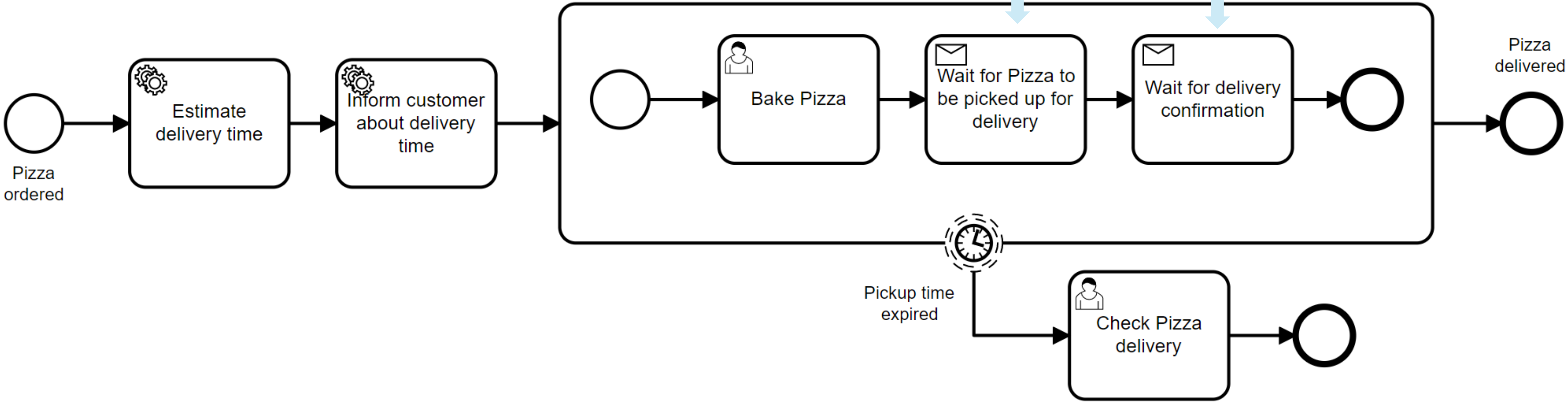
A possible process for the Pizza ordering system


Pizza ordered
Estimate delivery time
Inform customer about delivery time
Bake Pizza
Wait for Pizza to be picked up for delivery
Wait for delivery confirmation
Pizza delivered
Pickup time expired
Check Pizza delivery

You can still work with events

Pizza xy was picked up by driver z

Driver z handed over Pizza successfully


Pizza ordered
Estimate delivery time
Inform customer about delivery time
Bake Pizza
Wait for Pizza to be picked up for delivery
Wait for delivery confirmation
Pizza delivered
Pickup time expired
Check Pizza delivery

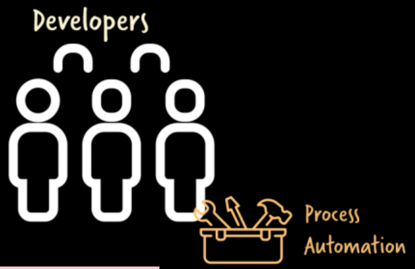
Developer-friendly orchestration engines

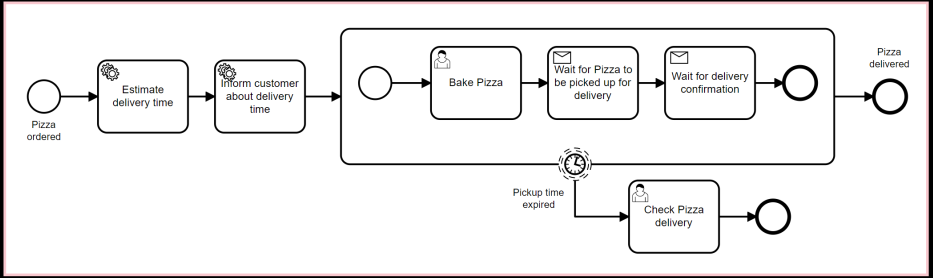
Your code to provide a REST endpoint

```
@PutMapping("/pizza-order")
public ResponseEntity<PizzaOrderResponse pizzaOrderReceived(...) {
  HashMap<String, Object> variables = new HashMap<String, Object>();
  variables.put("orderId", orderId);

  ProcessInstanceEvent processInstance = camunda.newCreateInstanceCommand()
            .bpmnProcessId("pizza-order")
            .latestVersion()
            .variables(variables)
            .send().join();

  return ResponseEntity.status(HttpStatus.ACCEPTED).build();
}
```




Pizza ordered
Estimate delivery time
Inform customer about delivery time
Bake Pizza
Wait for Pizza to be picked up for delivery
Wait for delivery confirmation
Pizza delivered
Pickup time expired
Check Pizza delivery

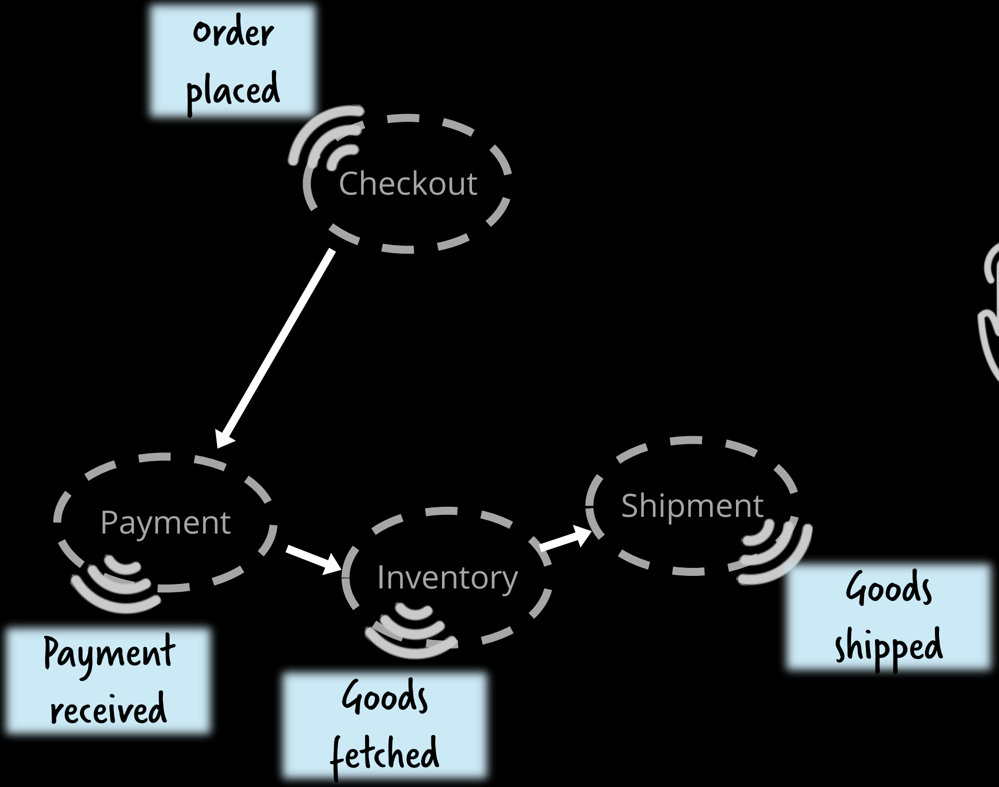
Orchestration vs. Choreography

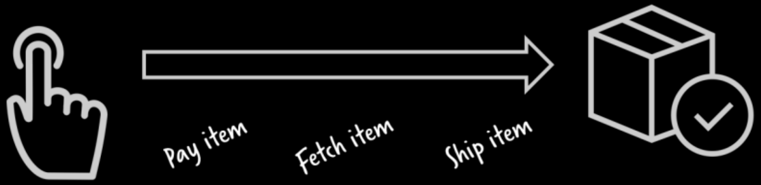
Definition

Orchestration = command-driven communication

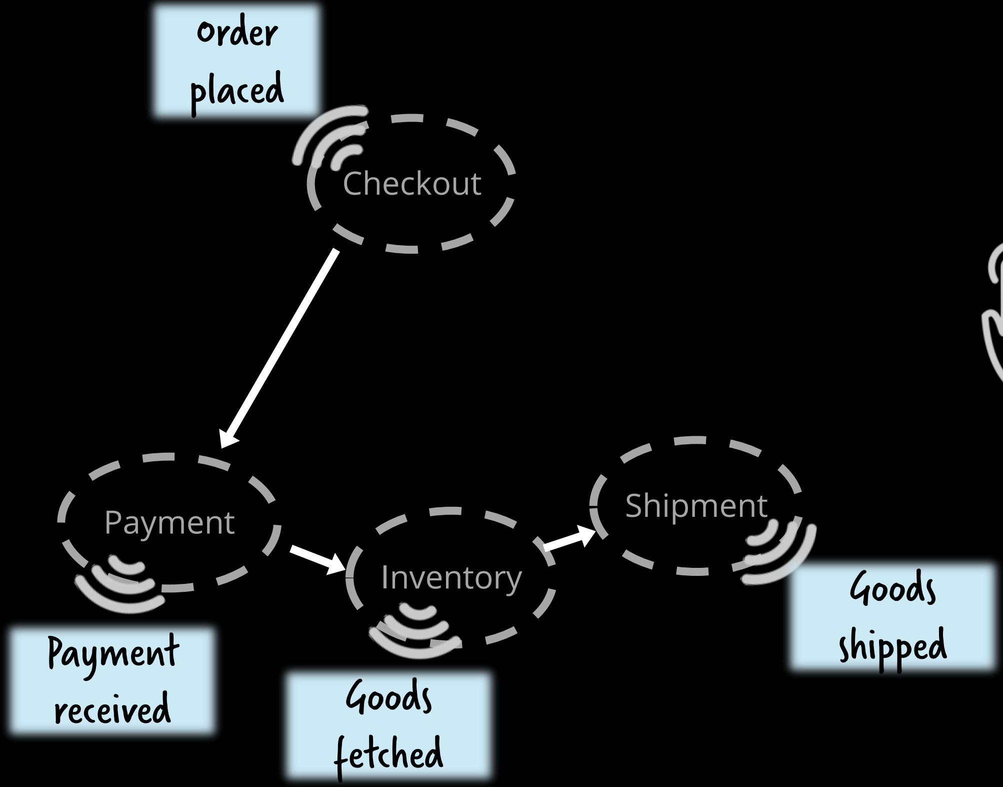
Choreography = event-driven communication

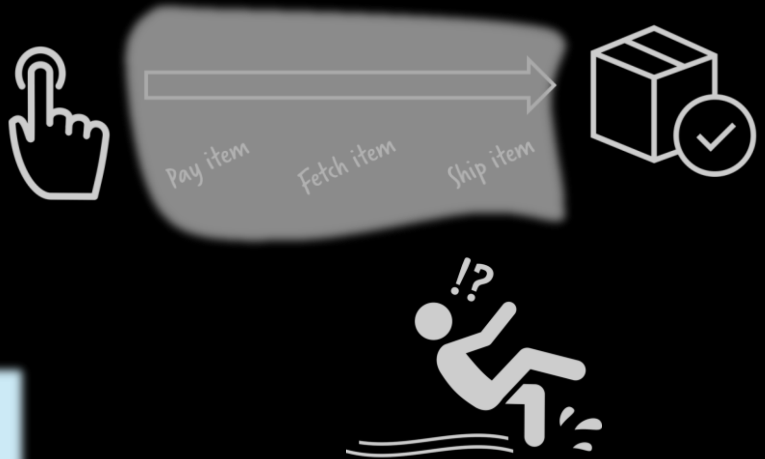
Let's switch examples: order fulfillment

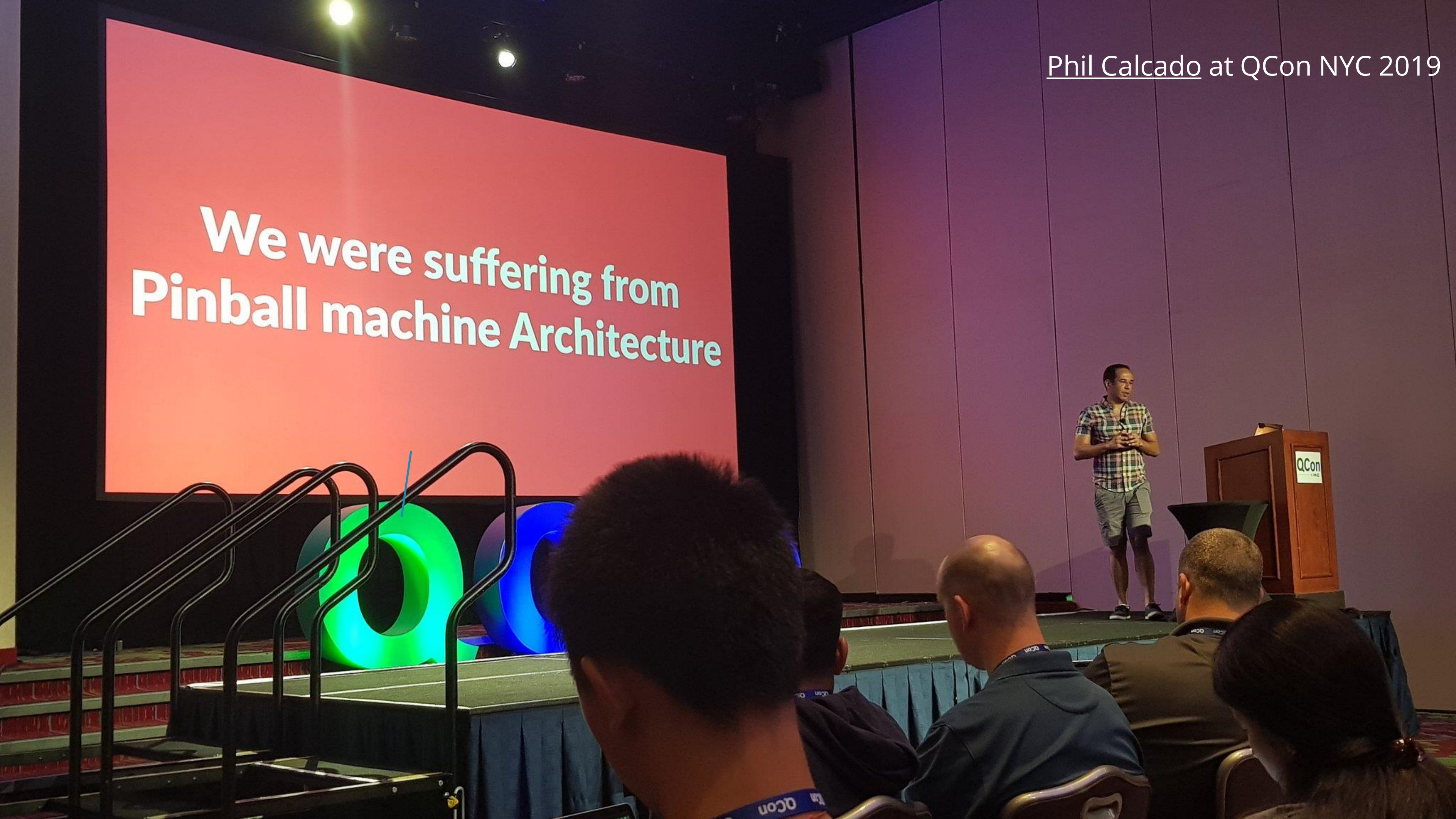




Event chains



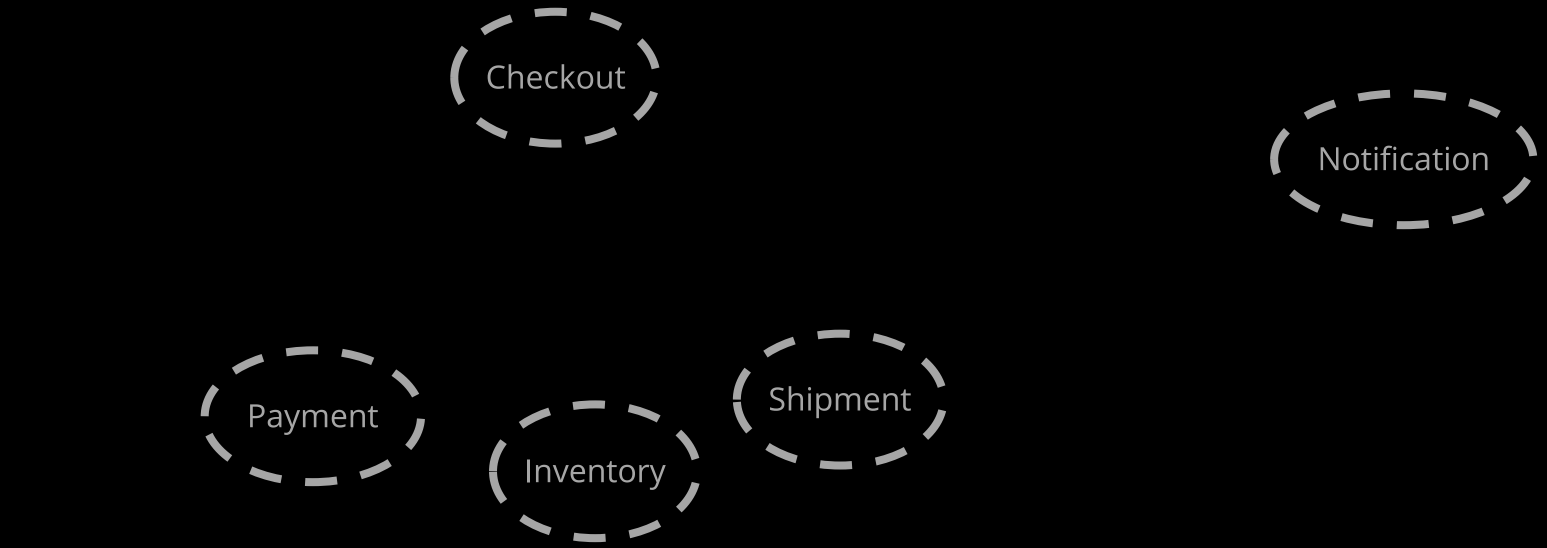




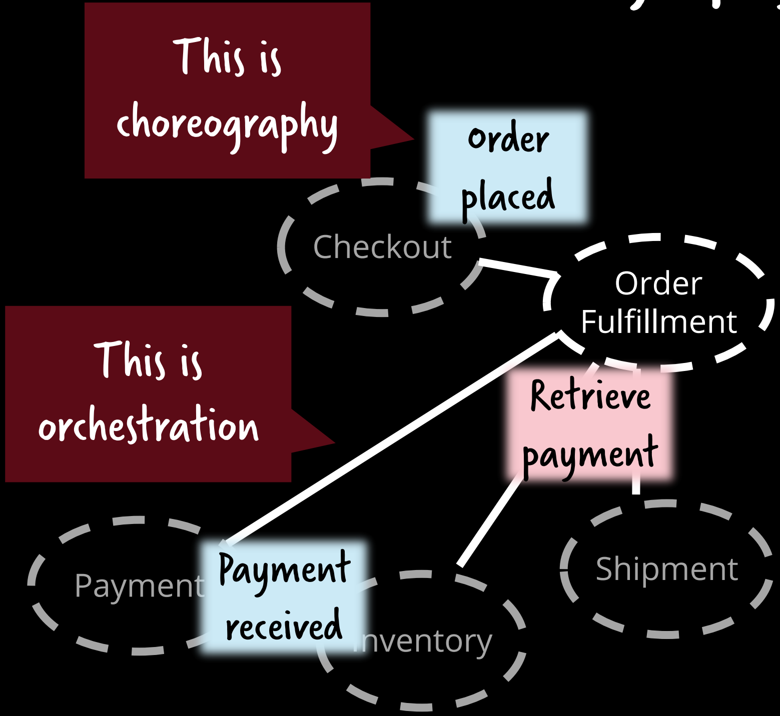
We were suffering from Pinball machine Architecture

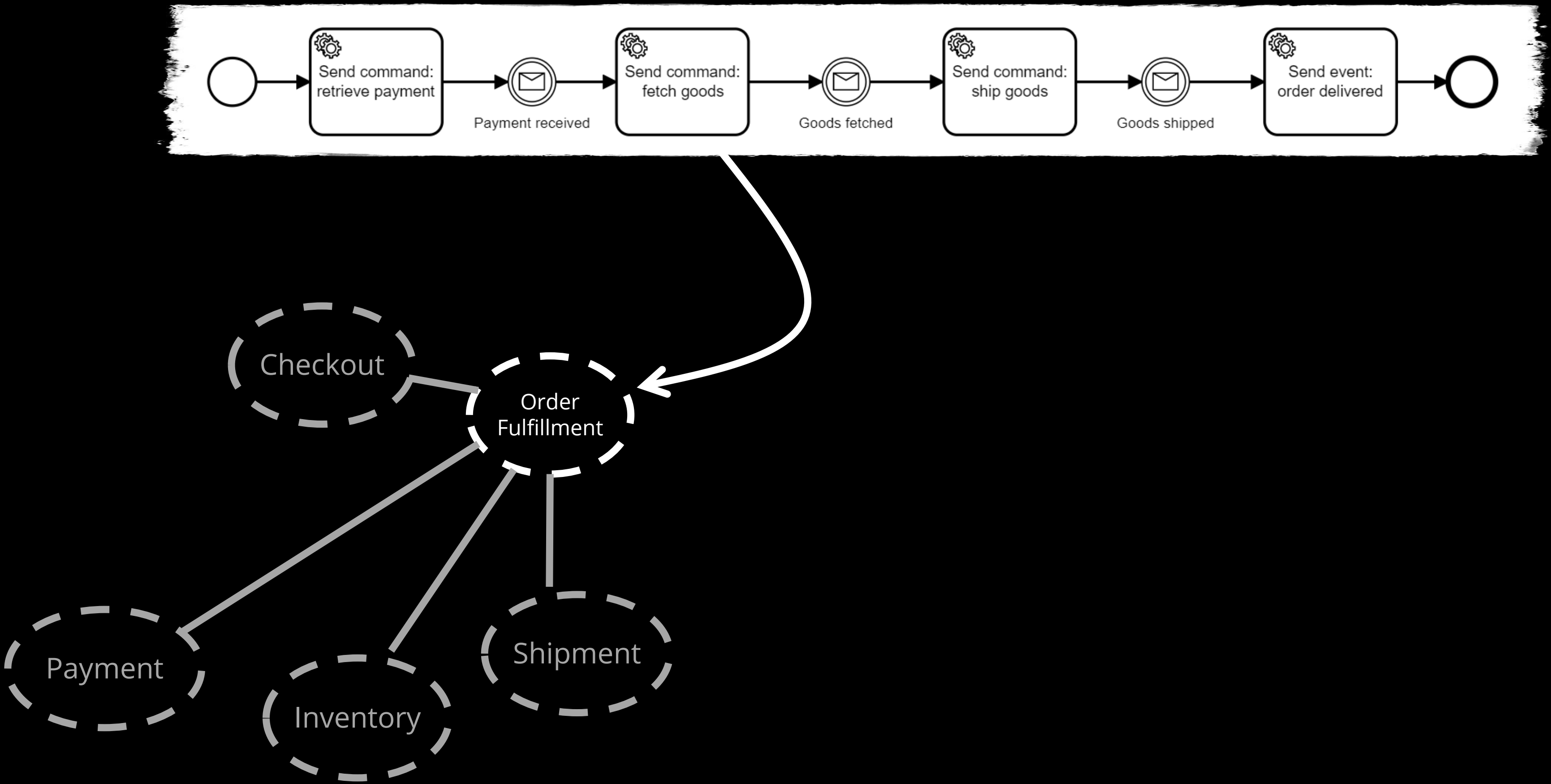
Phil Calcado at QCon NYC 2019

Pinball Machine Architecture



Orchestration and Choreography





Send command: retrieve payment

Payment received

Send command: fetch goods

Goods fetched

Send command: ship goods

Goods shipped

Send event: order delivered

Checkout

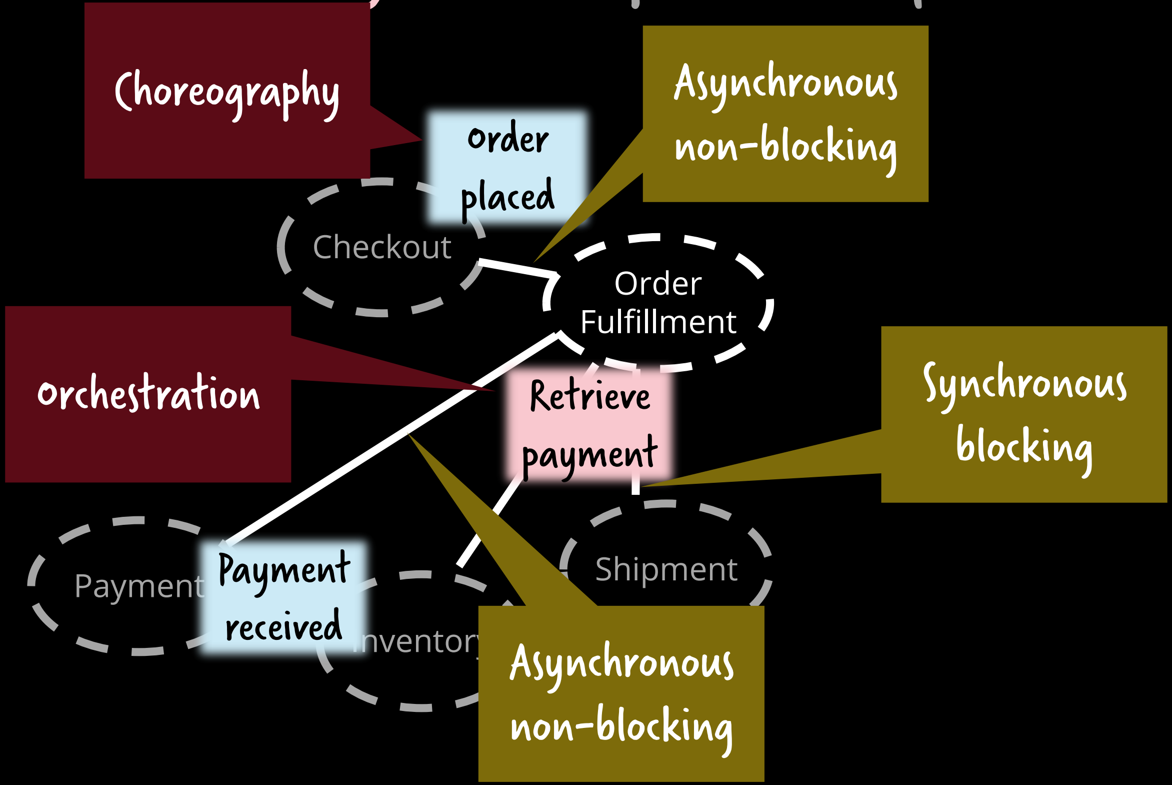
Order Fulfillment

Payment

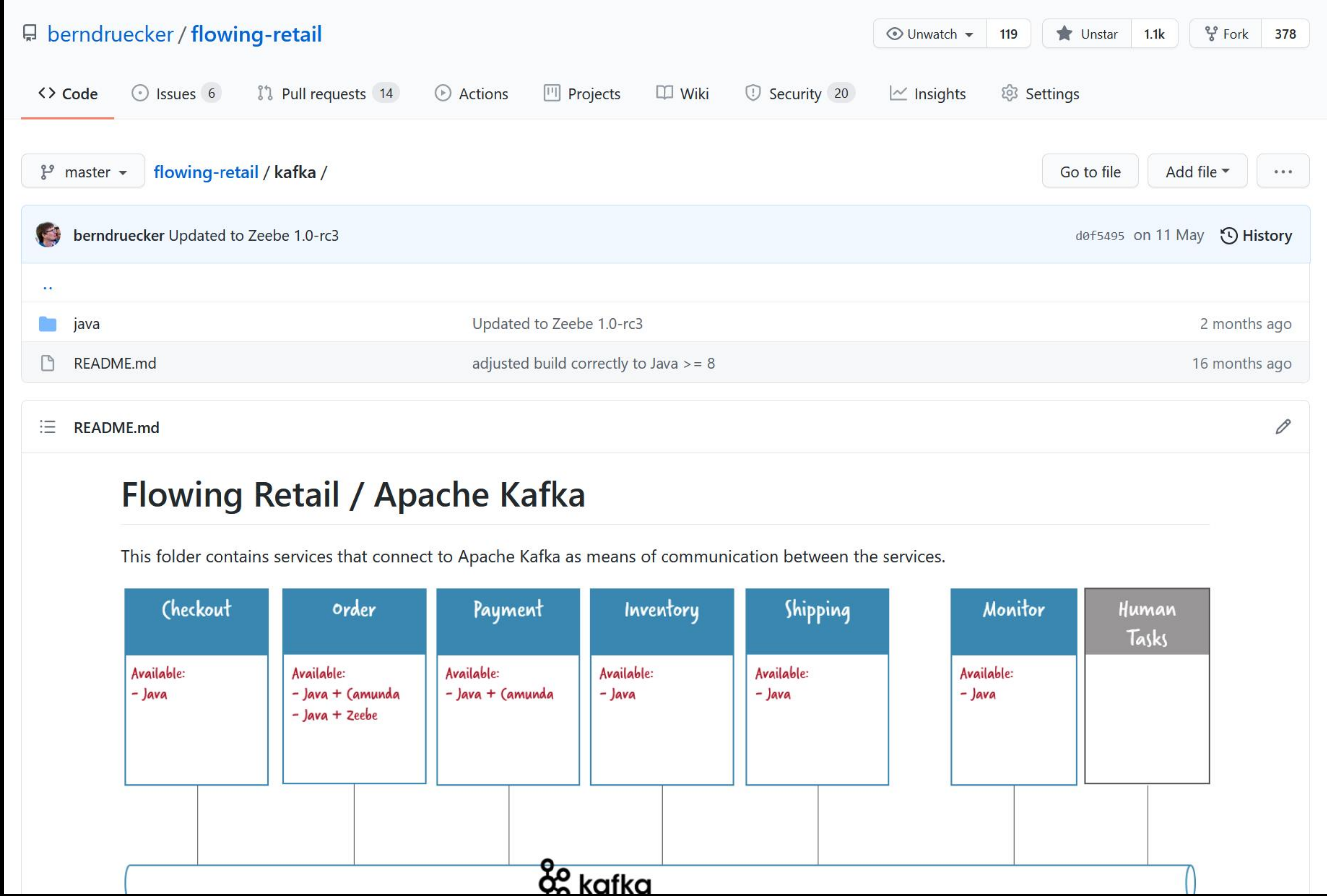
Inventory

Shipment

Collaboration style is independant of communication style



Some code?



flowing-retail / kafka /

berndruecker Updated to Zeebe 1.0-rc3 d0f5495 on 11 May History

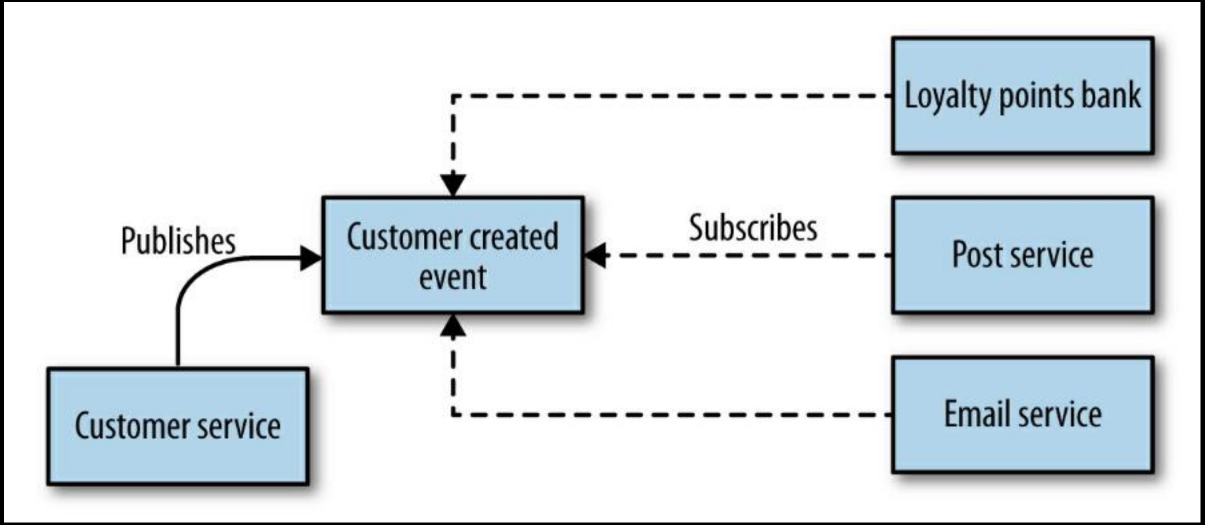
..		
java	Updated to Zeebe 1.0-rc3	2 months ago
README.md	adjusted build correctly to Java >= 8	16 months ago

README.md

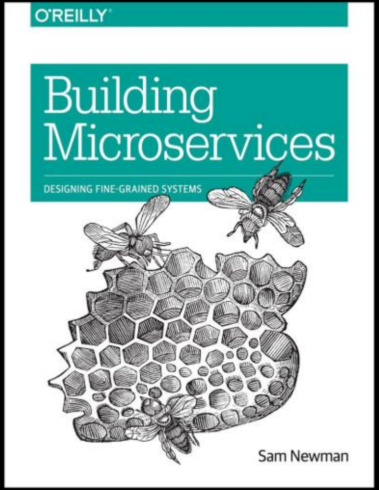
Flowing Retail / Apache Kafka

This folder contains services that connect to Apache Kafka as means of communication between the services.

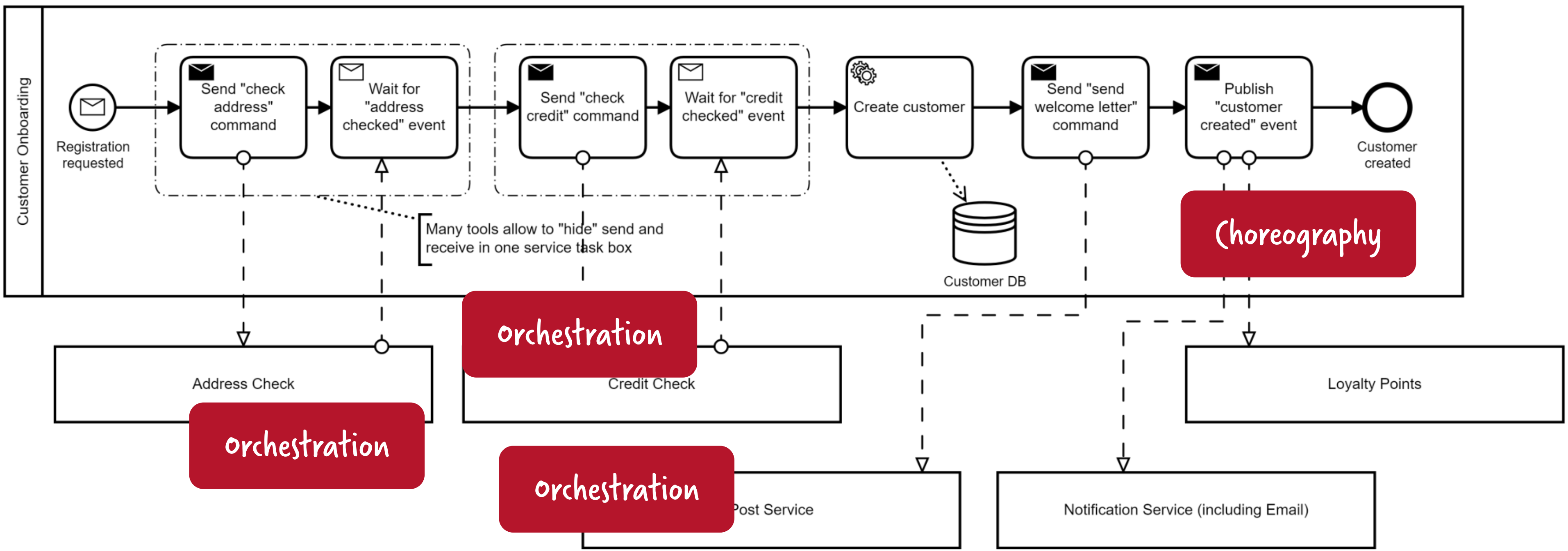
https://github.com/berndruecker/flowing-retail/tree/master/kafka



Sam Newman: Building Microservices

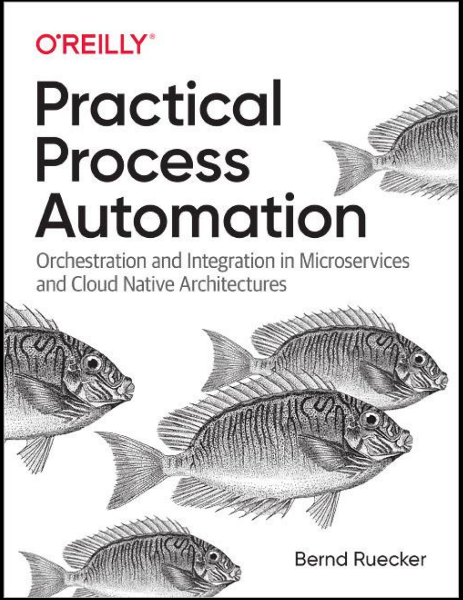


Mix orchestration and choreography


Customer Onboarding
Registration requested
Send "check address" command
Wait for "address checked" event
Send "check credit" command
Wait for "credit checked" event
Create customer
Send "send welcome letter" command
Publish "customer created" event
Customer created
Customer DB
Many tools allow to "hide" send and receive in one service task box
Choreography
Orchestration
Address Check
Credit Check
Orchestration
Orchestration
Post Service
Notification Service (including Email)
Loyalty Points

Want to learn more about choreography vs. orchestration?

Recording from QCon: https://drive.google.com/file/d/1IRWoQCX-gTPs7RVP5VrXaF1JozYWVbJv/view?usp=sharing
Slides: https://www.slideshare.net/BerndRuecker/gotopia-2020-balancing-choreography-and-orchestration

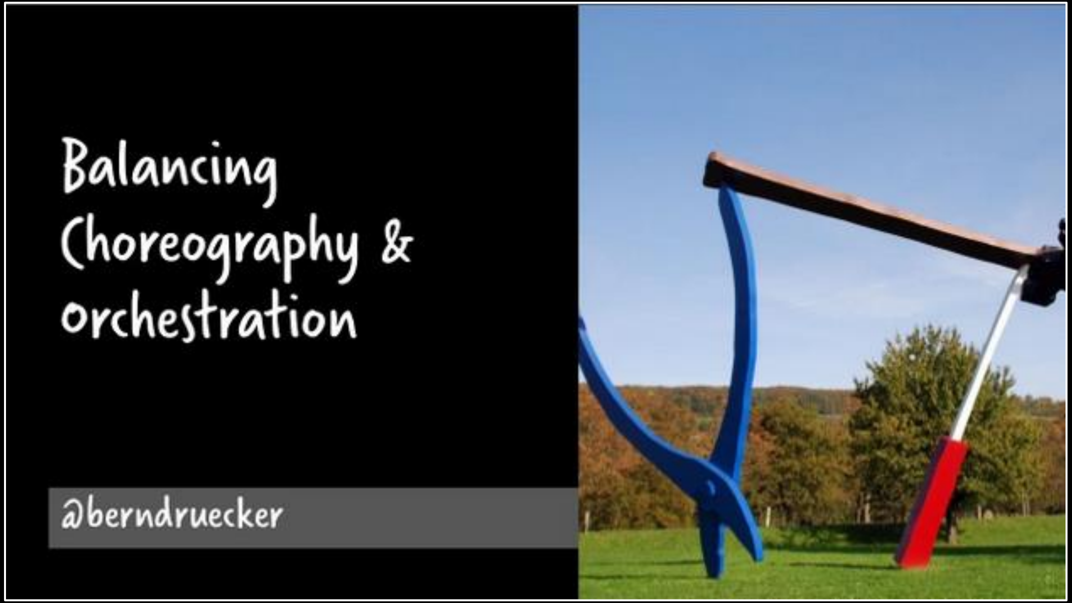


O'REILLY

Practical Process Automation

Orchestration and Integration in Microservices and Cloud Native Architectures

Bernd Ruecker



Balancing Choreography & orchestration

@berndruecker

https://learning.oreilly.com/library/view/practical-process-automation/9781492061441/
30 days trial: https://learning.oreilly.com/get-learning/?code=PPAER20

Communication options – Quick Summary

Communication Style	Synchronous Blocking	Asynchronous Non-Blocking	
Collaboration Style	Command-Driven		Event-Driven
Example	REST	Messaging (Queues)	Messaging (Topics)
Feedback Loop	HTTP Response	Response Message	-
Pizza Ordering via	Phone Call	E-Mail	Twitter

This is not the same!

Coupling

Types of Coupling

Type of coupling	Description	Example	Recommendation
Implementation Coupling	Service knows internals of other services	Joined database	

Types of Coupling

Type of coupling	Description	Example	Recommendation
Implementation Coupling	Service knows internals of other services	Joined database	**Avoid**

Types of Coupling

Type of coupling	Description	Example	Recommendation
Implementation Coupling	Service knows internals of other services	Joined database	**Avoid**
Temporal Coupling	Service depends on availability of other services	Synchronous blocking communication	

Types of Coupling

Type of coupling	Description	Example	Recommendation
Implementation Coupling	Service knows internals of other services	Joined database	**Avoid**
Temporal Coupling	Service depends on availability of other services	Synchronous blocking communication	**Reduce or manage**

Types of Coupling

Type of coupling	**Description**	**Example**	**Recommendation**
Implementation Coupling	Service knows internals of other services	Joined database	**Avoid**
Temporal Coupling	Service depends on availability of other services	Synchronous blocking communication	**Reduce or manage**
Deployment Coupling	Multiple services can only be deployed together	Release train	

Types of Coupling

Type of coupling	Description	Example	Recommendation
Implementation Coupling	Service knows internals of other services	Joined database	**Avoid**
Temporal Coupling	Service depends on availability of other services	Synchronous blocking communication	**Reduce or manage**
Deployment Coupling	Multiple services can only be deployed together	Release train	Typically **avoid**, but depends

Types of Coupling

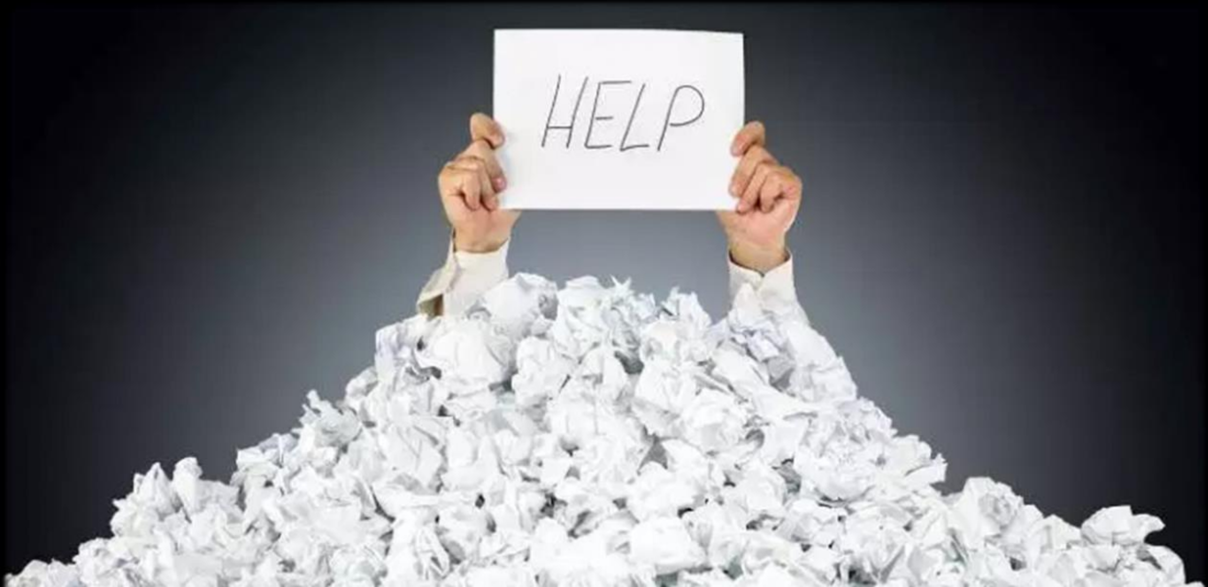
Type of coupling	**Description**	**Example**	**Recommendation**
Implementation Coupling	Service knows internals of other services	Joined database	**Avoid**
Temporal Coupling	Service depends on availability of other services	Synchronous blocking communication	**Reduce or manage**
Deployment Coupling	Multiple services can only be deployed together	Release train	Typically **avoid**, but depends
Domain Coupling	Business capabilities require multiple services	Order fulfillment requires payment, inventory and shipping	

Types of Coupling

This is influenced with the communication or collaboration style

Type of coupling	Description	Example	Recommendation
Implementation Coupling	Service knows internals of other services	Joined database	**Avoid**
Temporal Coupling	Service depends on availability of other services	Synchronous blocking communication	**Reduce or manage**
Deployment Coupling	Multiple services can only be deployed together	Release train	Typically **avoid**, but depends
Domain Coupling	Business capabilities require multiple services	Order fulfillment requires payment, inventory and shipping	**Unavoidable** unless you change business requirements or service boundaries

Messaging?



Patterns To Survive Remote Communication

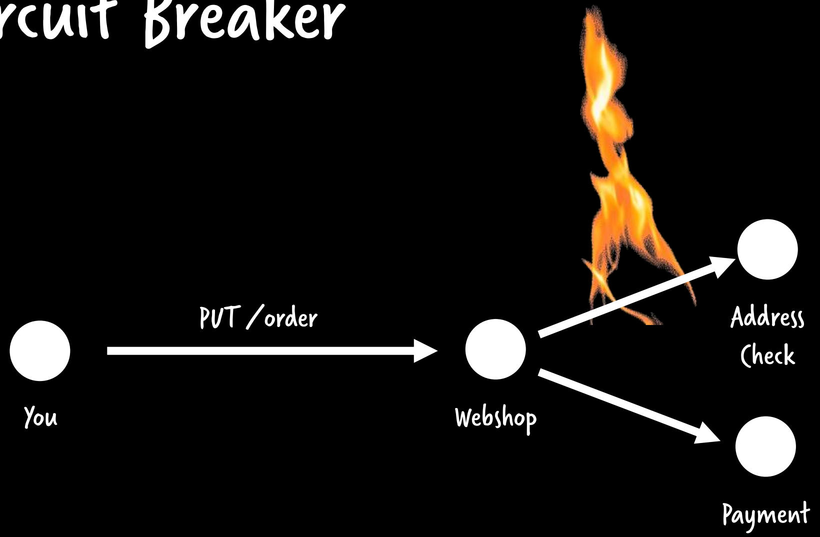
Service Consumer	Pattern/Concept	Use With	Service Provider
X	Service Discovery	Sync	(X)
X	Circuit Breaker	Sync	
X	Bulkhead	Sync	
(X)	Load Balancing	Sync	X
X	Retry	Sync / Async	
X	Idempotency	Sync / Async	X
	De-duplication	Async	X
(X)	Back Pressure & Rate Limiting	Sync / (Async)	X
X	Await feedback	Async	
X	Sagas	Sync / Async	(X)

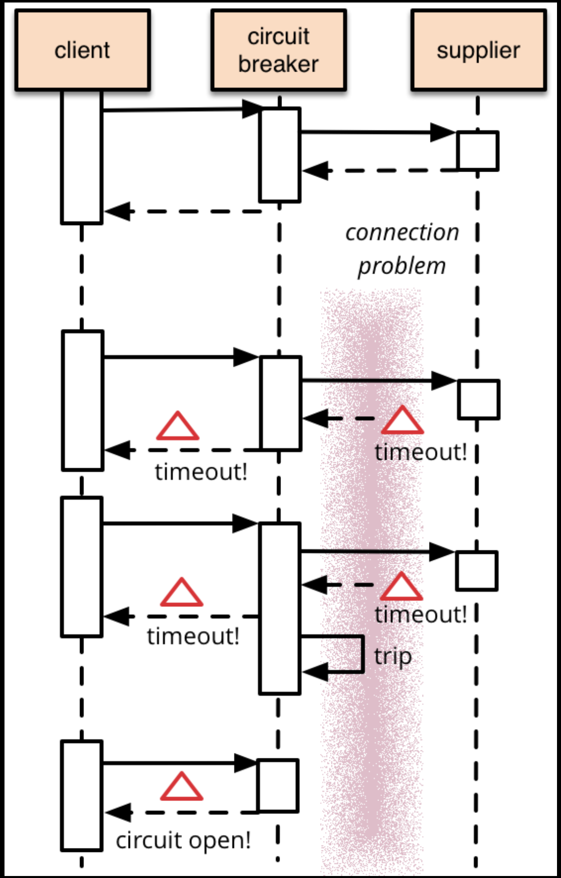
…

Circuit Breaker

Photo by CITYEDV, available under Creative Commons CC0 1.0 license.

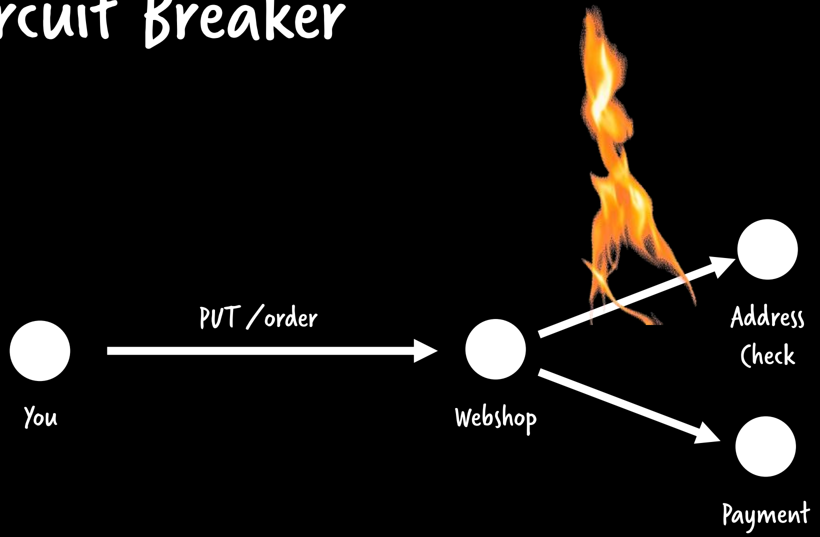
Circuit Breaker





from https://martinfowler.com/bliki/CircuitBreaker.html

Circuit Breaker

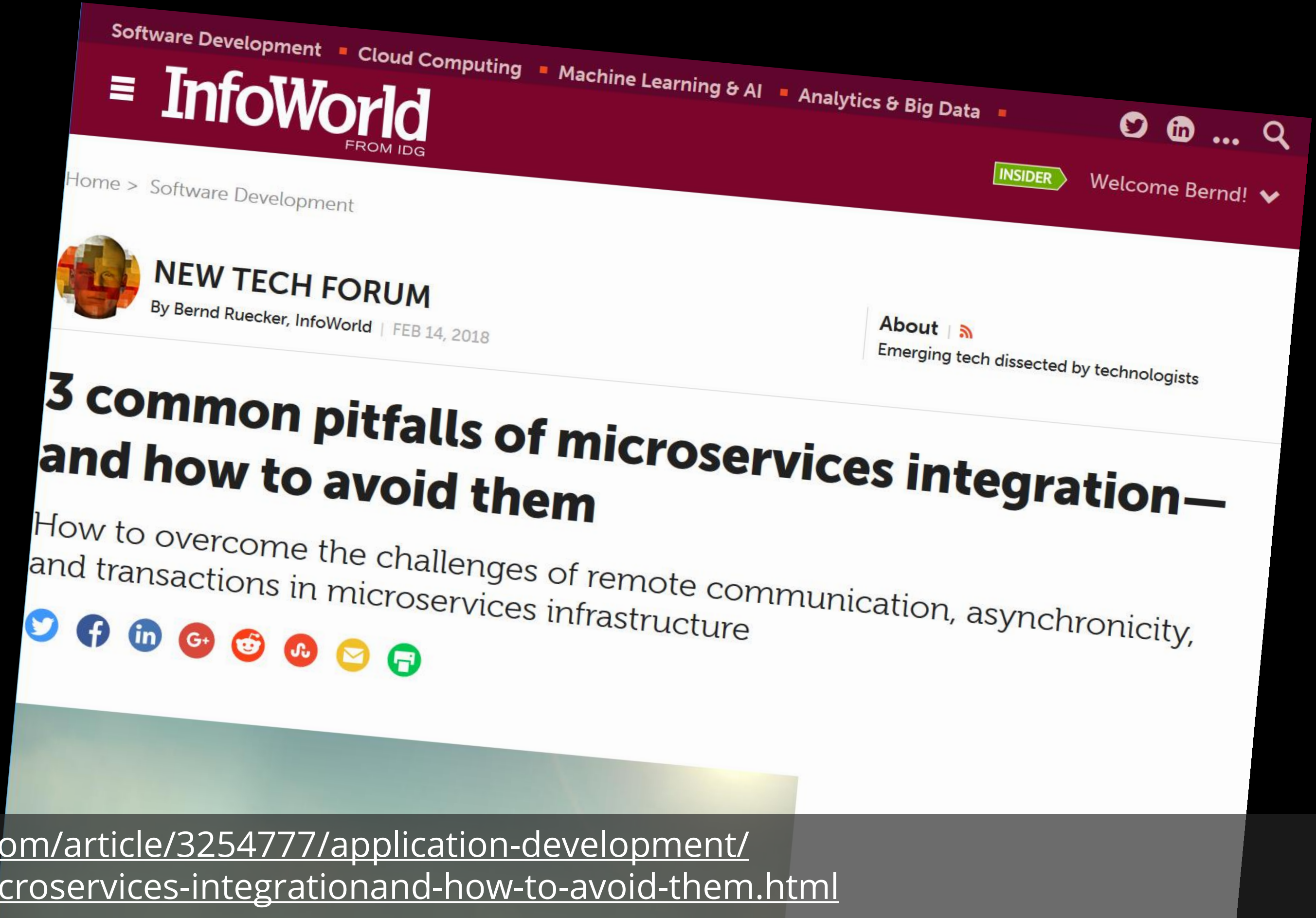


e.g. Resilience4J:

```
@CircuitBreaker(name = BACKEND, fallbackMethod =
"fallback")
public boolean addressValid(Address a) {
    return httpEndpoint.GET(...);
}

private boolean fallback(Address a) {
    return true;
}
```

```
resilience4j.circuitbreaker:
    instances:
        BACKEND:
            registerHealthIndicator: true
            slidingWindowSize: 100
            permittedNumberOfCallsInHalfOpenState: 3
            minimumNumberOfCalls: 20
            waitDurationInOpenState: 50s
            failureRateThreshold: 50
```



NEW TECH FORUM

By Bernd Ruecker, InfoWorld | FEB 14, 2018

About

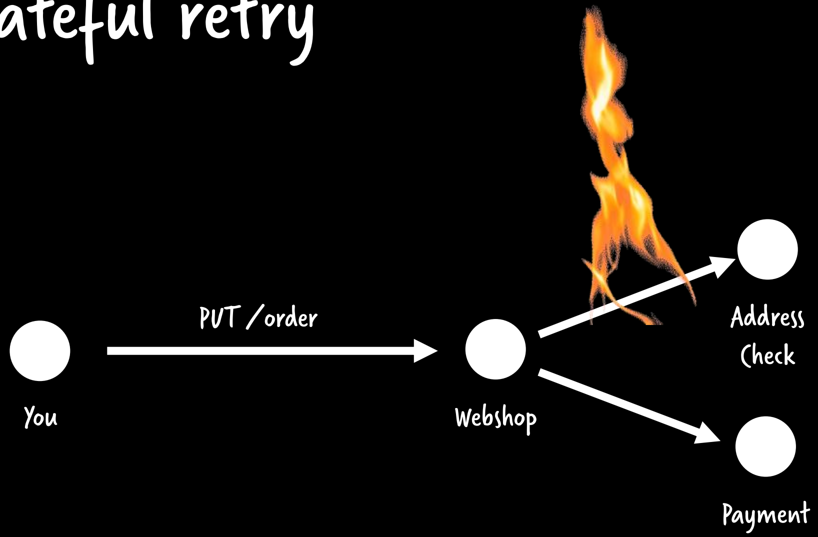
Emerging tech dissected by technologists

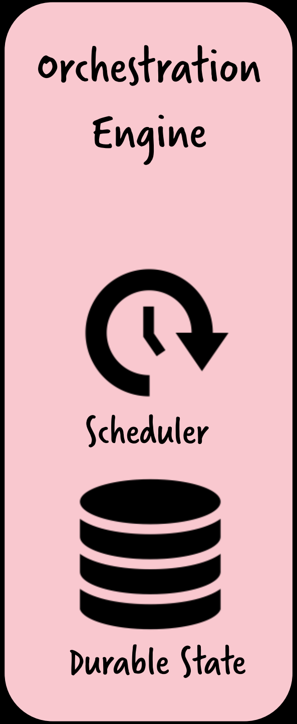
3 common pitfalls of microservices integration—and how to avoid them

How to overcome the challenges of remote communication, asynchronicity, and transactions in microservices infrastructure

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

Stateful retry





Patterns To Survive Remote Communication

Service Consumer	Pattern/Concept	Use With	Service Provider
X	Service Discovery	Sync	(X)
X	Circuit Breaker	Sync	
X	Bulkhead	Sync	
(X)	Load Balancing	Sync	X
X	Retry	Sync / Async	
X	Idempotency	Sync / Async	X
	De-duplication	Async	X
(X)	Back Pressure & Rate Limiting	Sync / (Async)	X
X	Await feedback	Async	
X	Sagas	Sync / Async	(X)

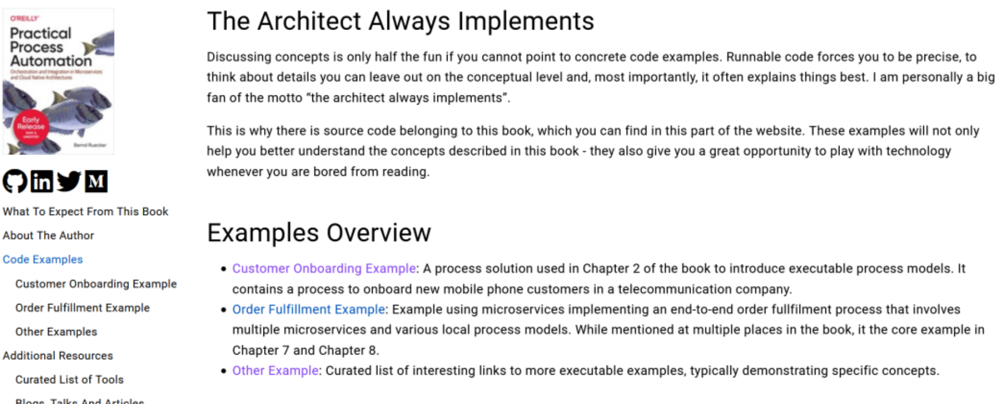
…

Summary

- Know
  - **communication styles (sync/async)**
  - **collaboration styles (command/event)**
- You can get rid of **temporal coupling** with asynchronous communication
  - Make sure you or **your team** can handle it
  - You will need **long running capabilities** (you might need it anyway)
  - Synchronous communication + correct patterns might also be OK
- **Domain coupling** does not go away!

Want to learn more…

https://ProcessAutomationBook.com/



What To Expect From This Book

About The Author

Code Examples

Customer Onboarding Example

Order Fulfillment Example

Other Examples

Additional Resources

Curated List of Tools

Blogs, Talks And Articles

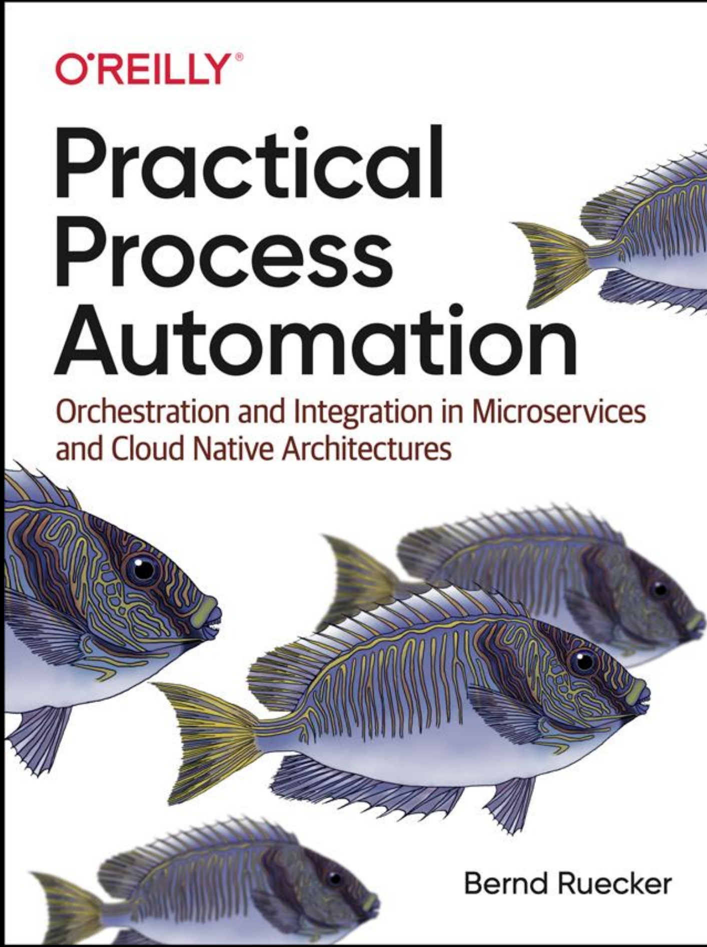
The Architect Always Implements

Discussing concepts is only half the fun if you cannot point to concrete code examples. Runnable code forces you to be precise, to think about details you can leave out on the conceptual level and, most importantly, it often explains things best. I am personally a big fan of the motto "the architect always implements".

This is why there is source code belonging to this book, which you can find in this part of the website. These examples will not only help you better understand the concepts described in this book - they also give you a great opportunity to play with technology whenever you are bored from reading.

Examples Overview

- Customer Onboarding Example: A process solution used in Chapter 2 of the book to introduce executable process models. It contains a process to onboard new mobile phone customers in a telecommunication company.
- Order Fulfillment Example: Example using microservices implementing an end-to-end order fullfilment process that involves multiple microservices and various local process models. While mentioned at multiple places in the book, it the core example in Chapter 7 and Chapter 8.
- Other Example: Curated list of interesting links to more executable examples, typically demonstrating specific concepts.



O'REILLY
Practical Process Automation
Orchestration and Integration in Microservices and Cloud Native Architectures
Bernd Ruecker

Thank you!