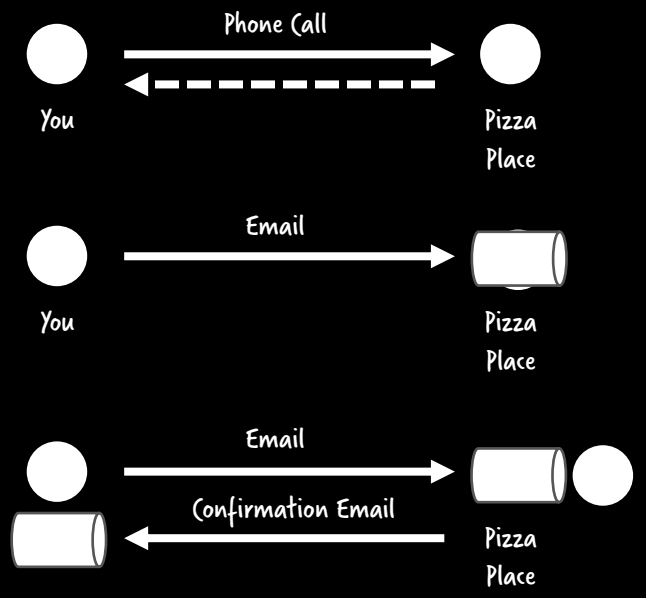
Loosely or Lousily Coupled?

Understanding Communication Patterns in Microservices Architectures

@berndruecker

Let's talk about food

How does ordering Pizza work?

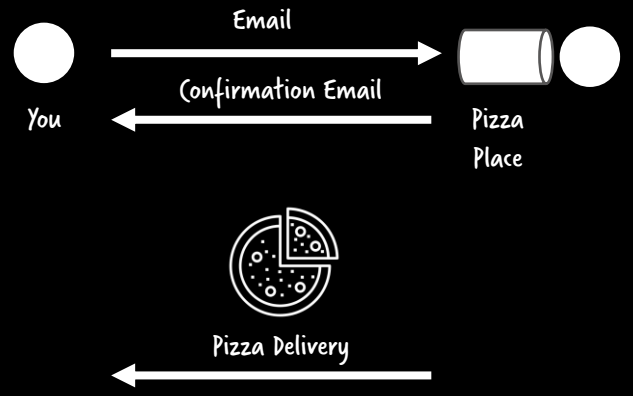


Synchronous blocking communication
Feedback loop (ack, confirmation or rejection)
Temporal coupling (e.g. busy, not answering)

Asynchronous non-blocking communication
No temporal coupling

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

Feedback loop != result



Feedback (ACK, confirmation, rejection)

Result

Synchronous blocking behavior for the result?

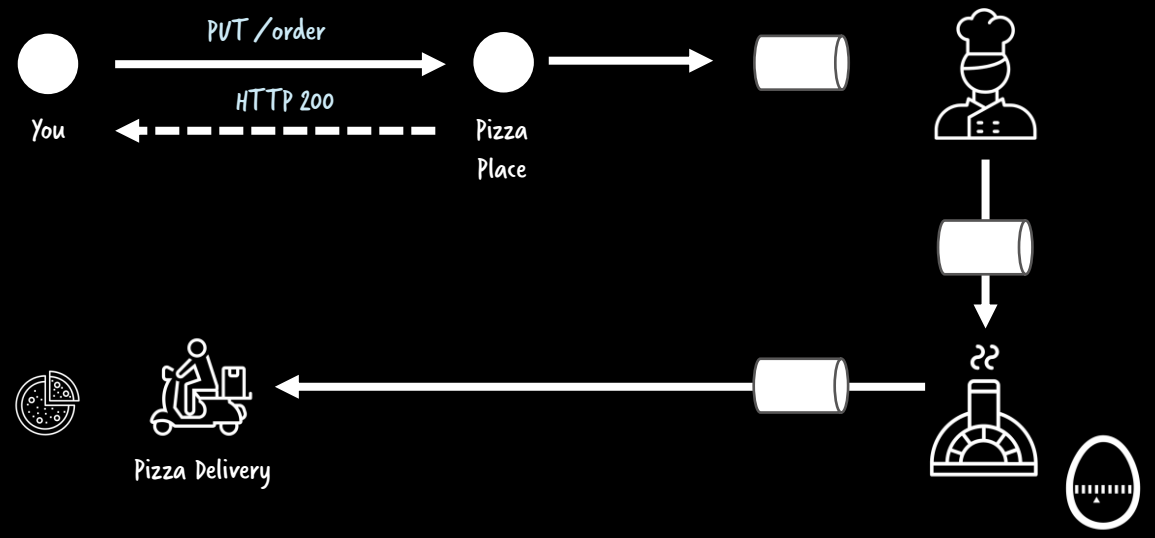
Bad user experience
Does not scale well

Scalable Coffee Making

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

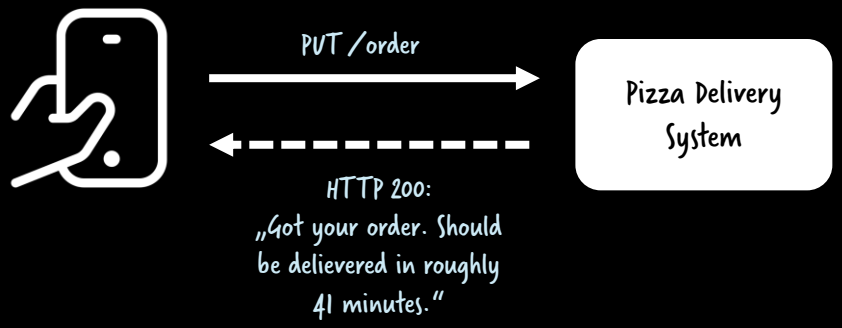
Photo by John Ingle

only the first communication step is synchronous

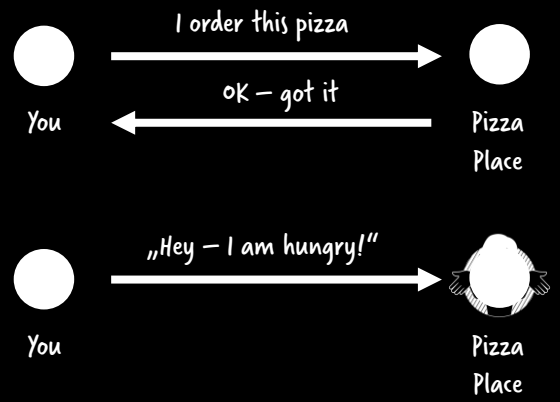


The task of
Pizza making is
long running

Example: Build a pizza ordering app



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


@berndruecker
Orchestrator
Command

Example: Build a pizza ordering app using events

PUT /order

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

Pizza Delivery System

Hey – somebody ordered

Hey – Pizza is ready

I have a Pizza for you

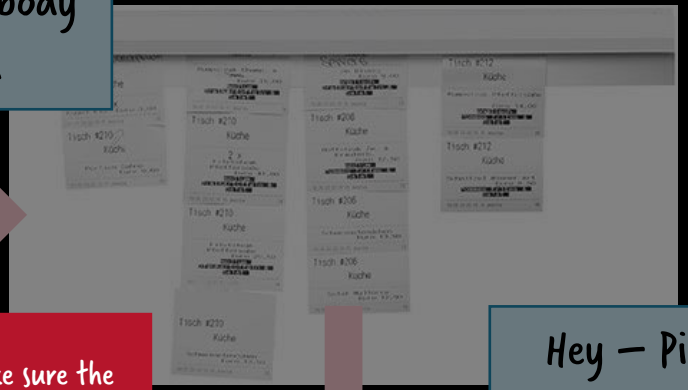
Example: Build a pizza ordering app using events

PUT /order

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

Pizza Delivery System

Hey – somebody ordered

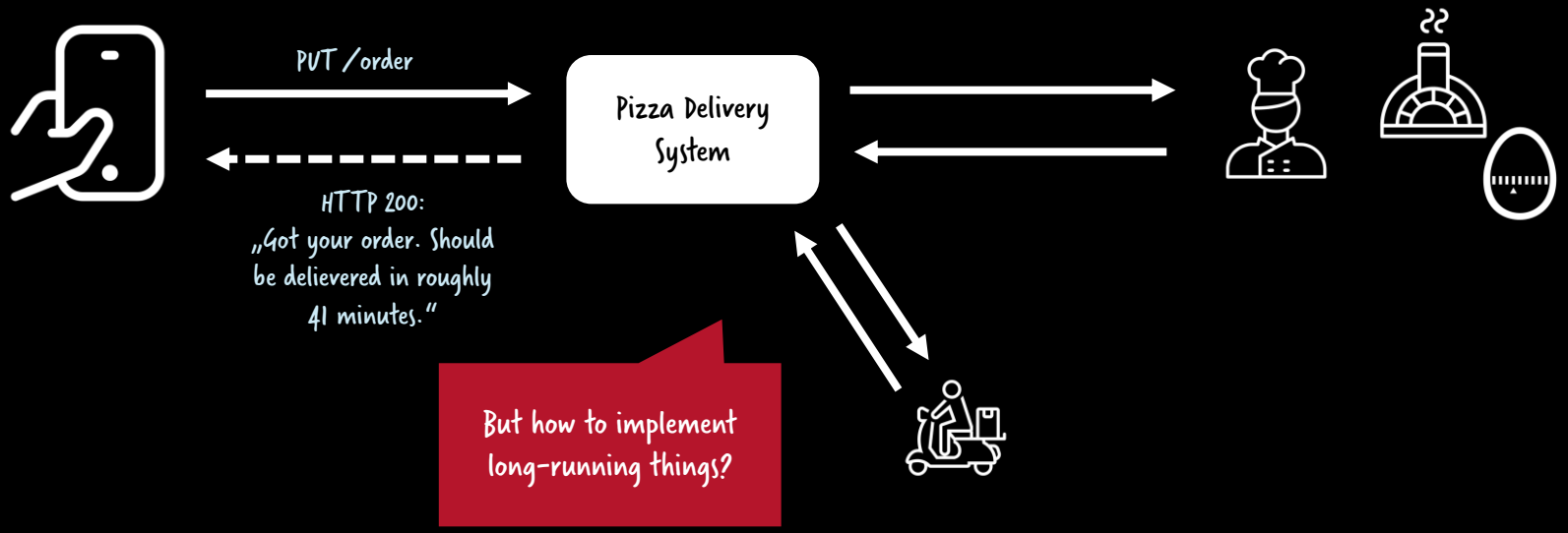


How do I make sure the Pizza is not forgotten?

Hey – Pizza is ready

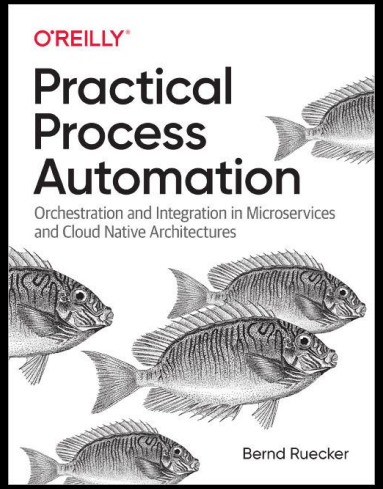
I have a Pizza for you

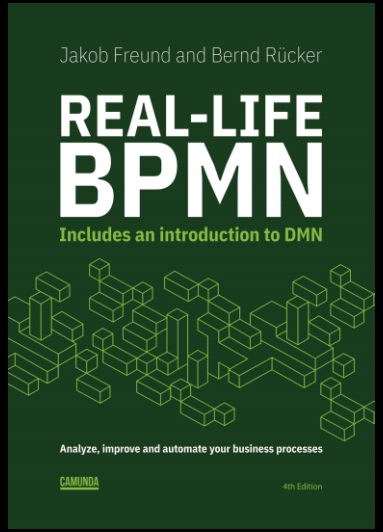
Example: Build a pizza ordering app via orchestration



Bernd Ruecker

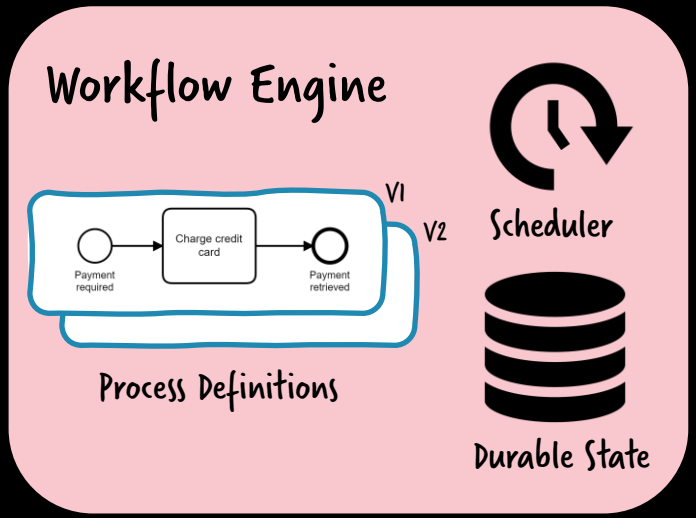
Co-founder and
Chief Technologist of
Camunda





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

An workflow engine provides long running capabilities



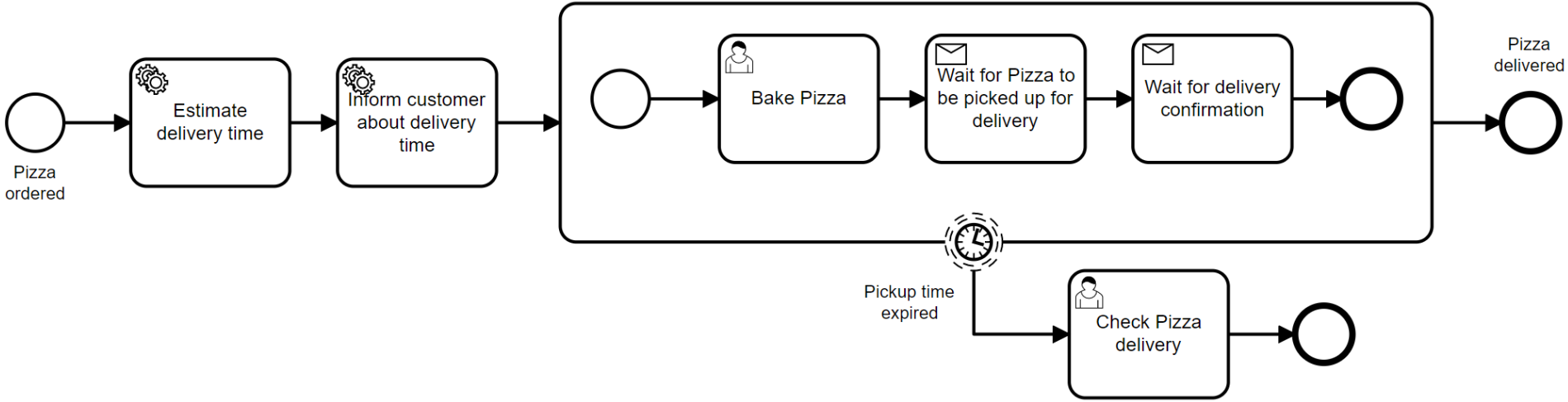
Workflow Engine:

Is stateful

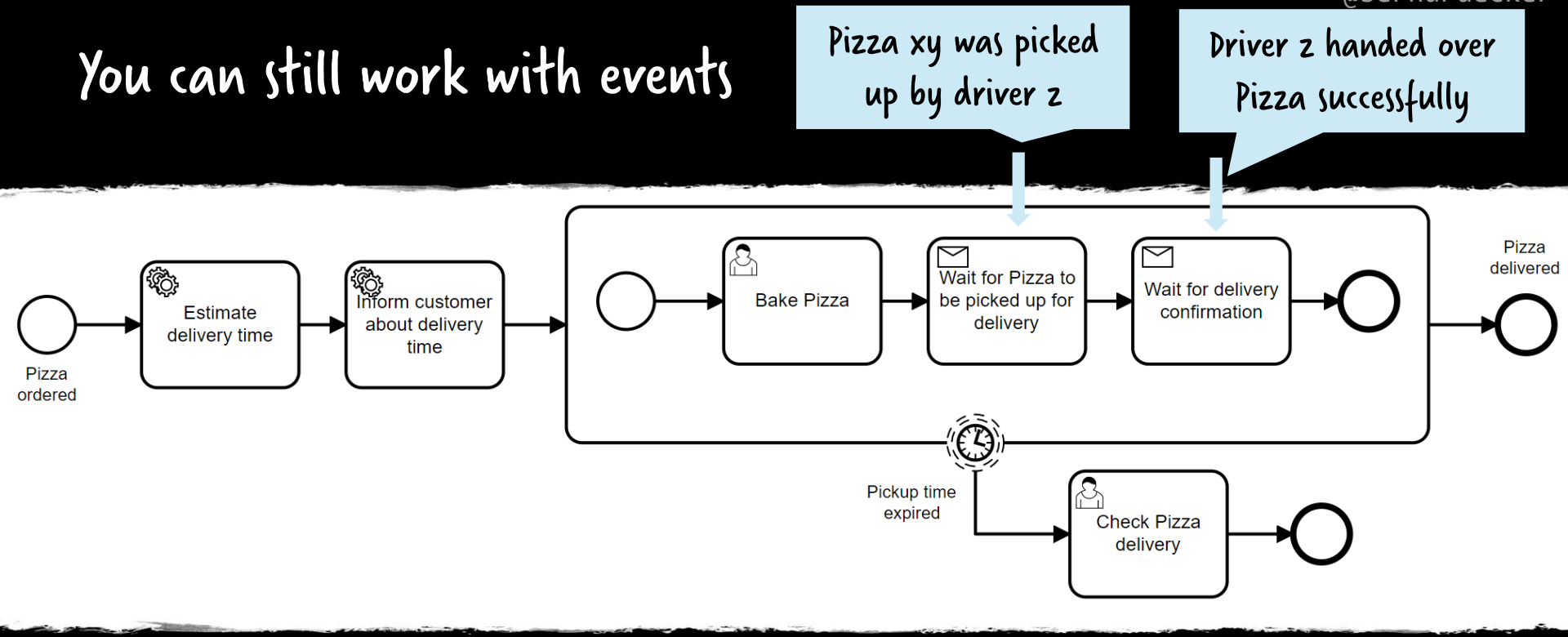
Can wait
Can retry
Can escalate
Can compensate

Provides visibility

A possible process for the Pizza ordering system



You can still work with events

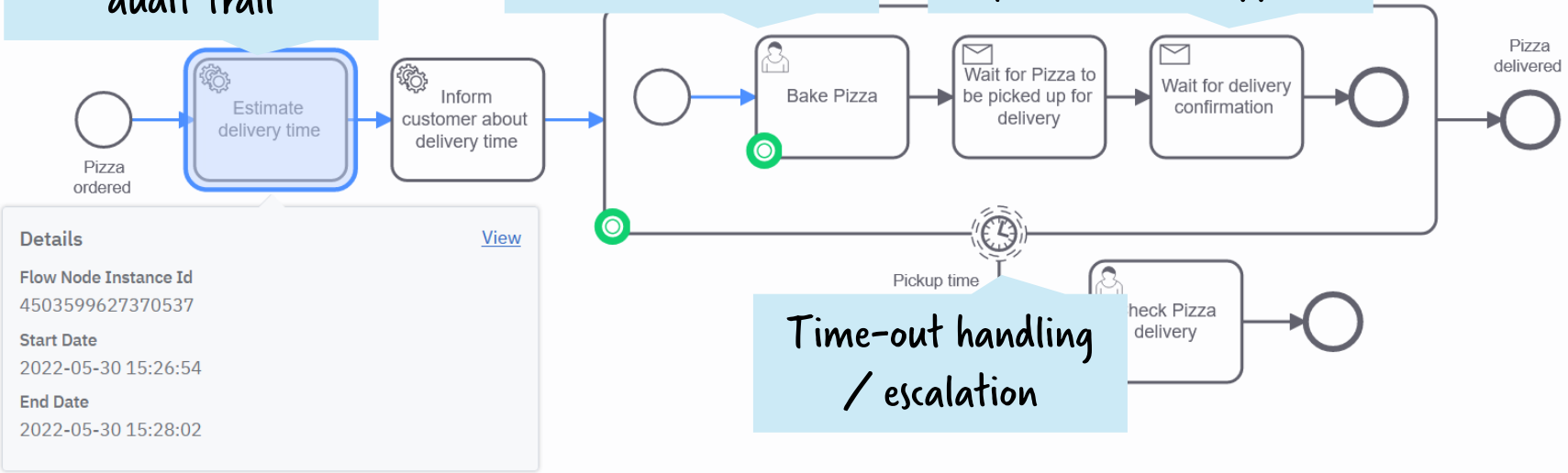


Advantages

Visibility: History and audit trail

Visibility: What's the current status?

Long running: Waiting for events to happen



Time-out handling / escalation

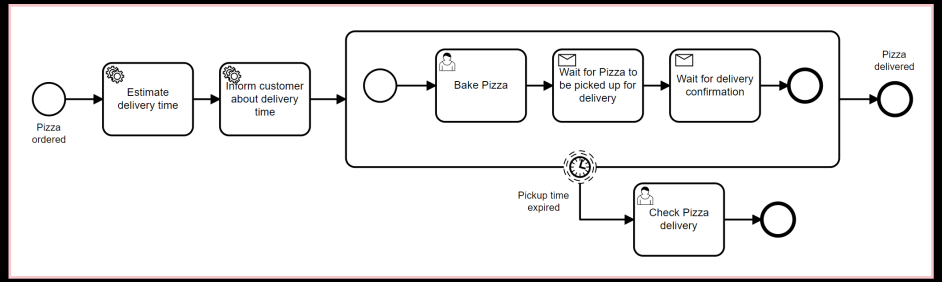
Developer-friendly workflow engines

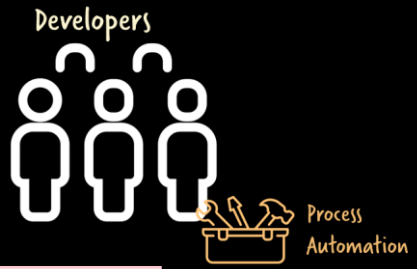
Your code to provide a REST endpoint

```
@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();
}
```





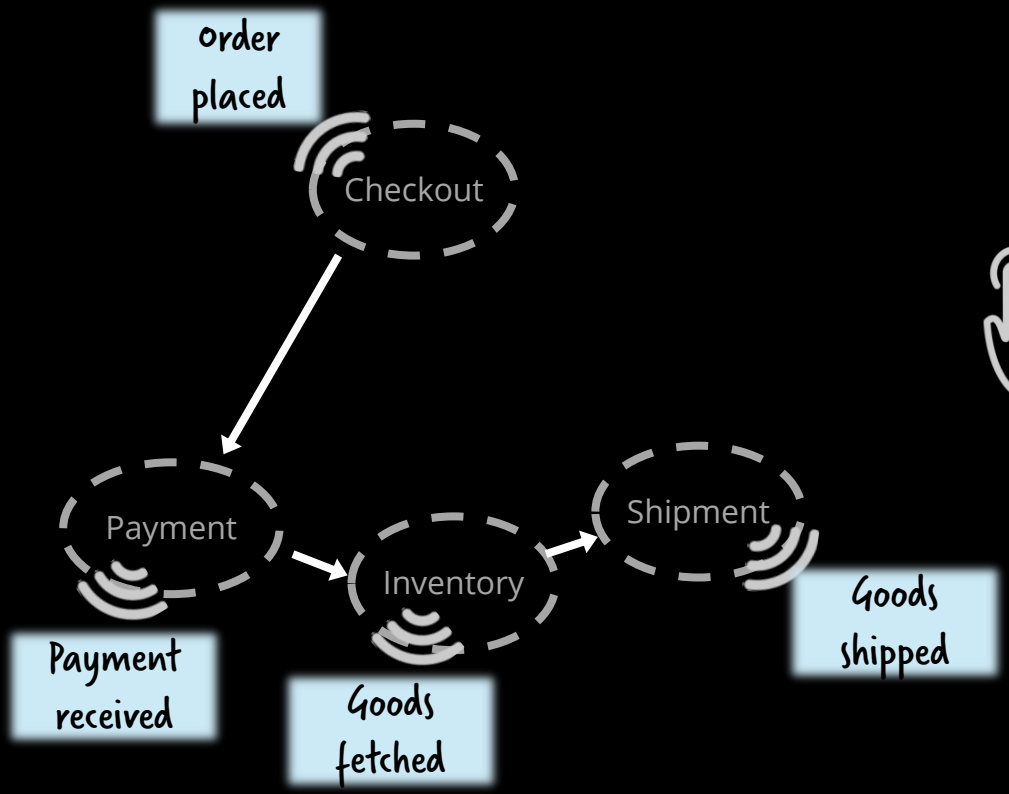
@berndruecker

orchestration vs. Choreography

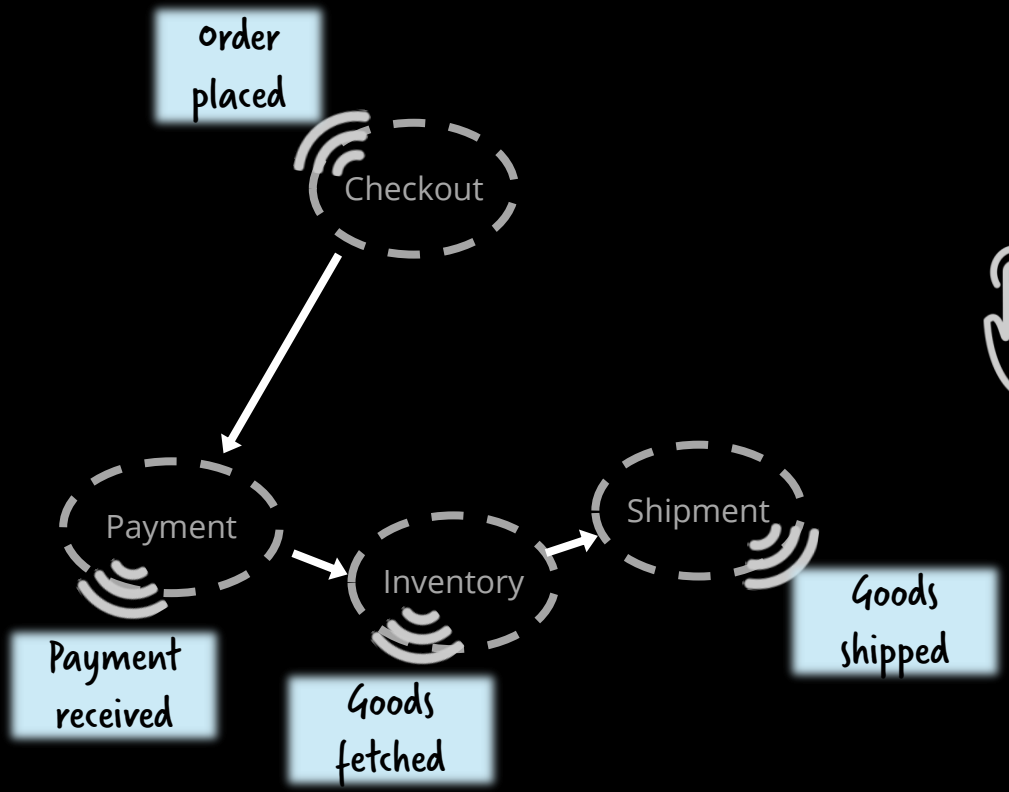
Definition

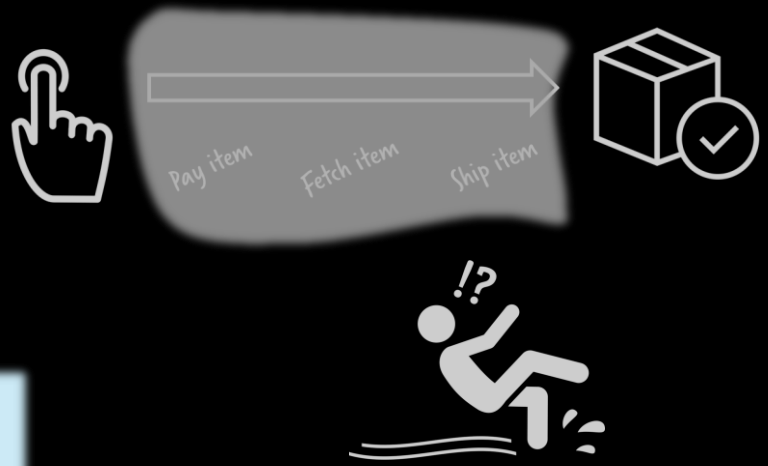
orchestration	=	command-driven communication
Choreography	=	event-driven communication

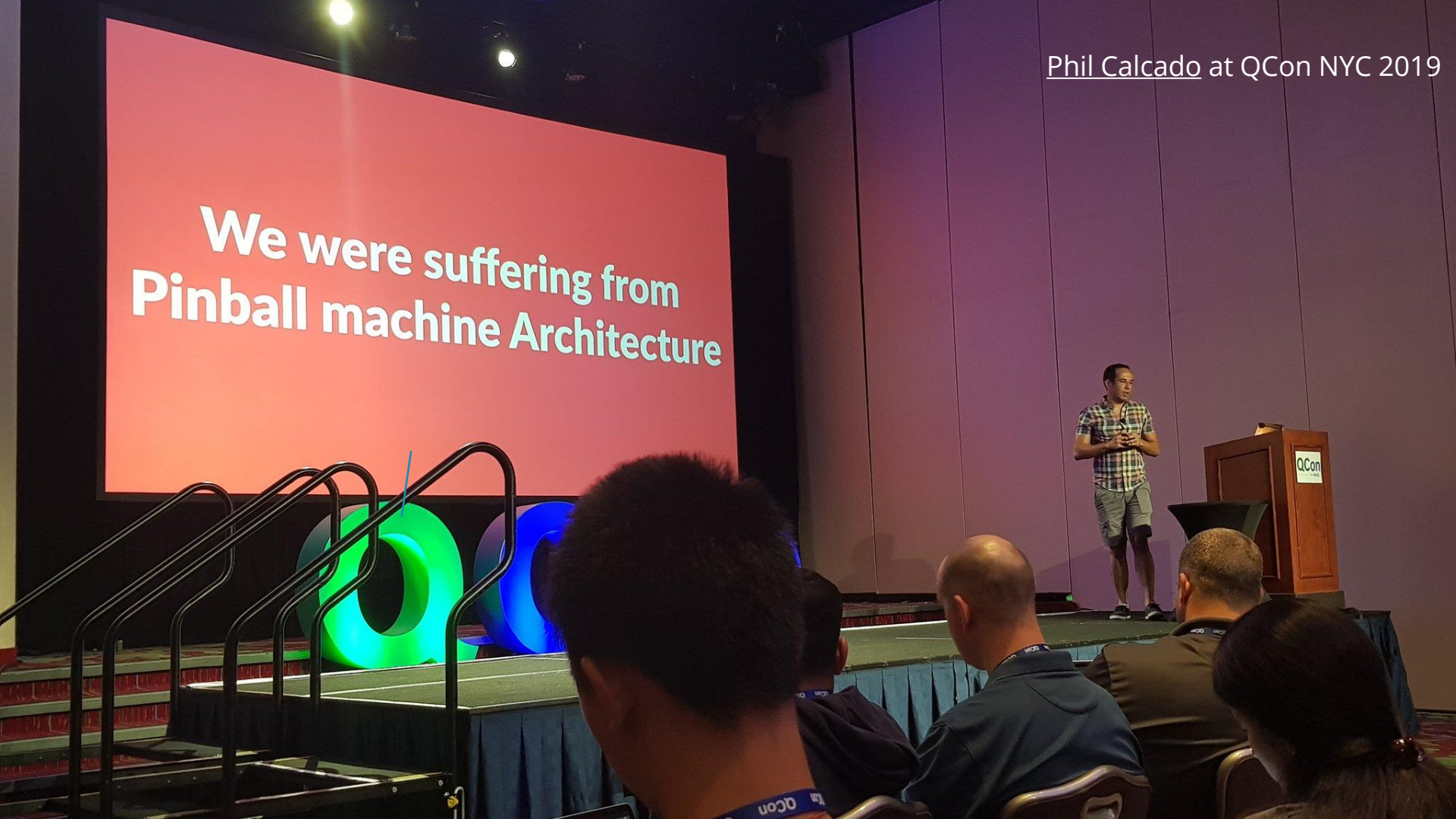
Let's switch examples: order fulfillment



Event chains

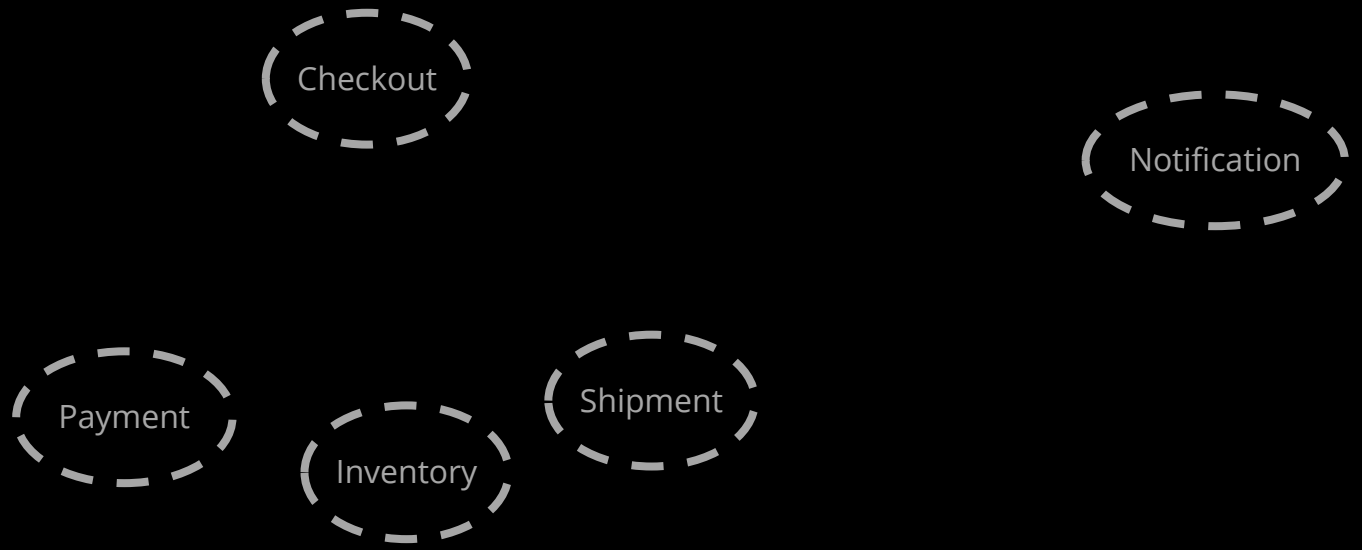




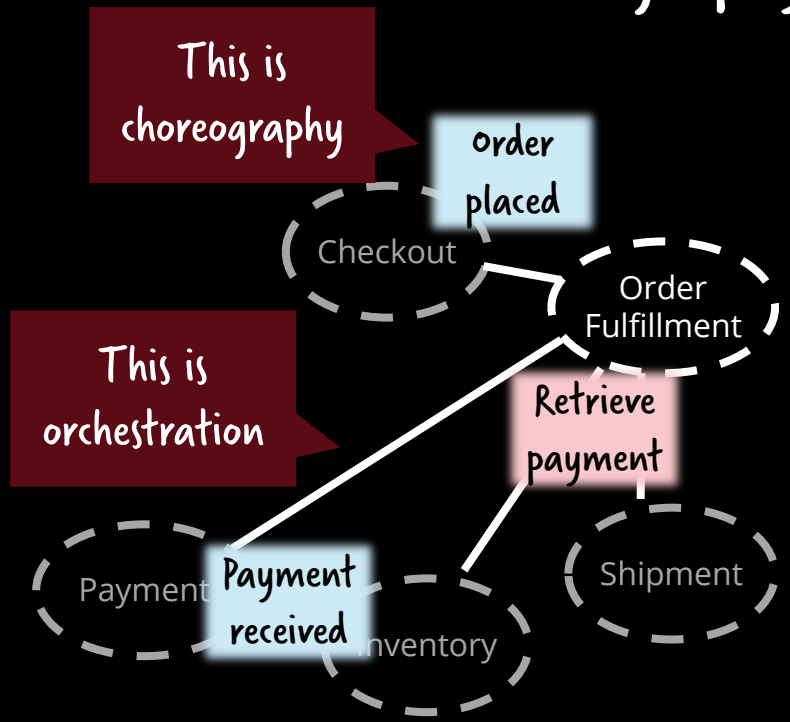


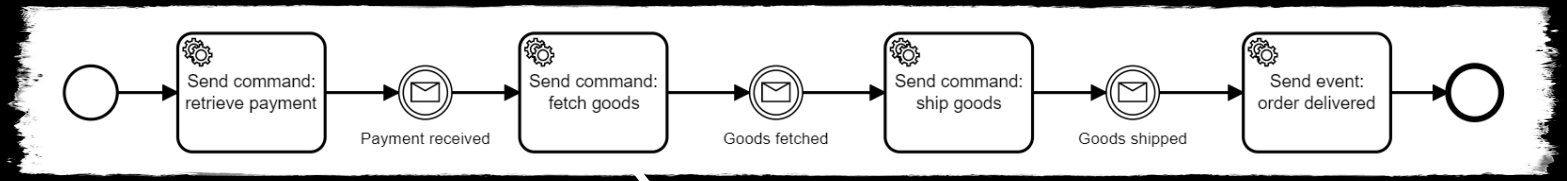
Phil Calcado at QCon NYC 2019

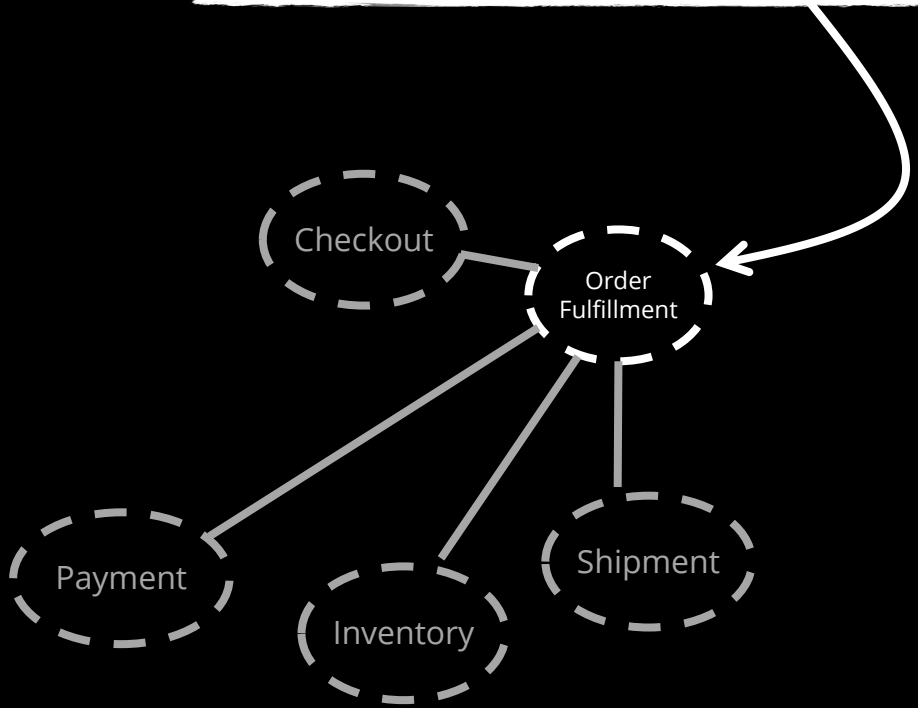
Pinball Machine Architecture



Orchestration and Choreography

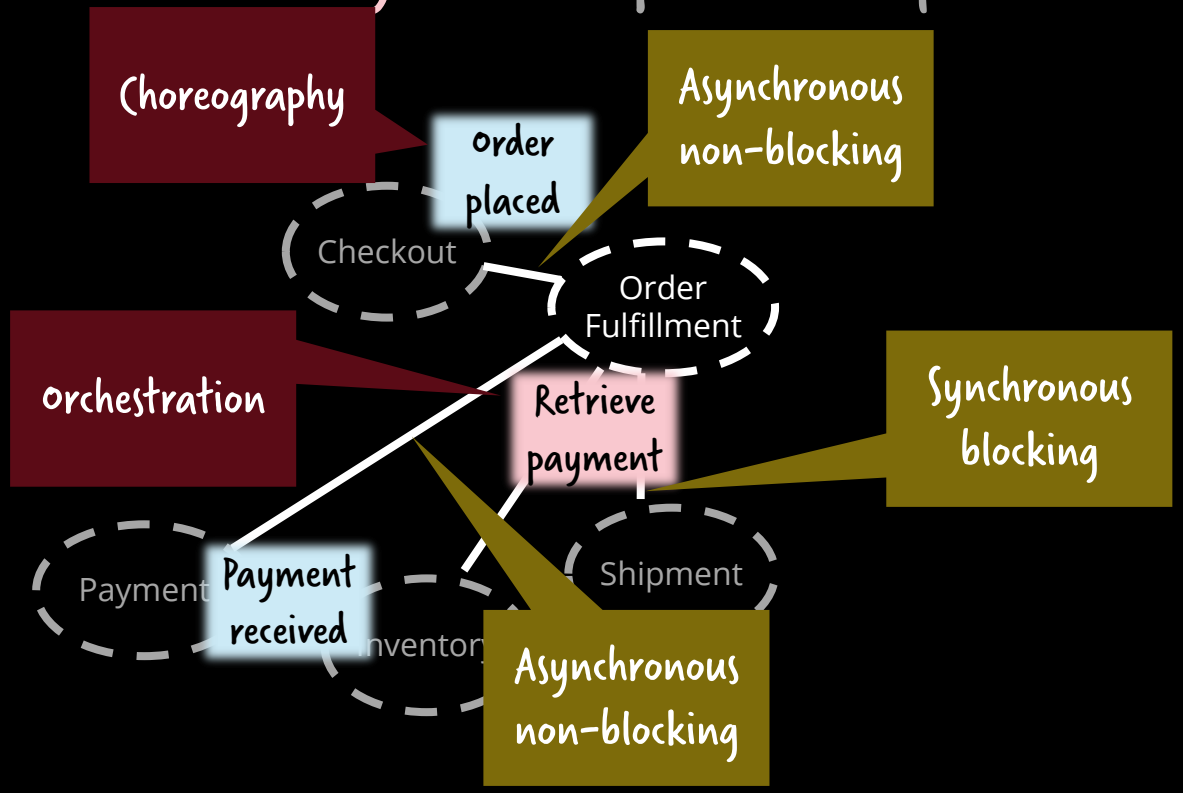




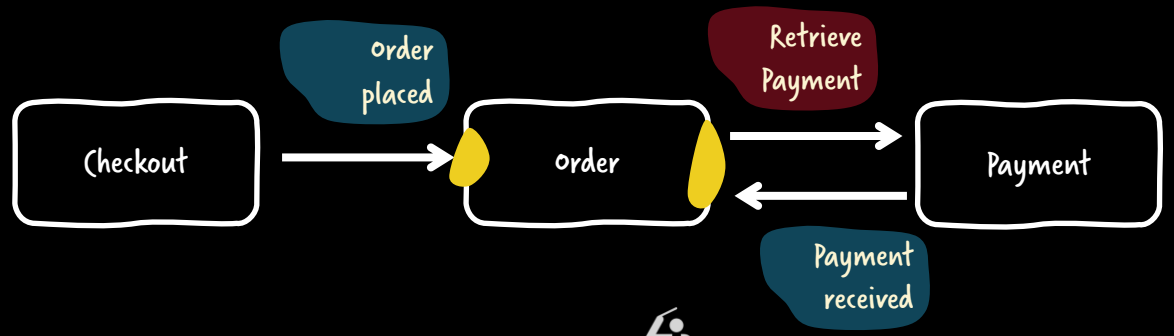


@berndruecker

Collaboration style is independant of communication style



Direction of dependency

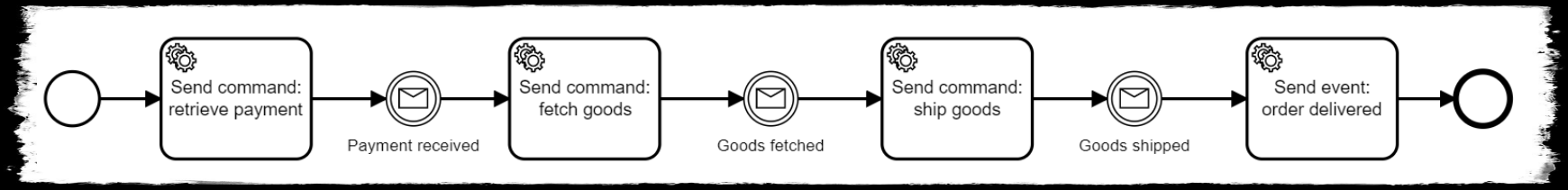


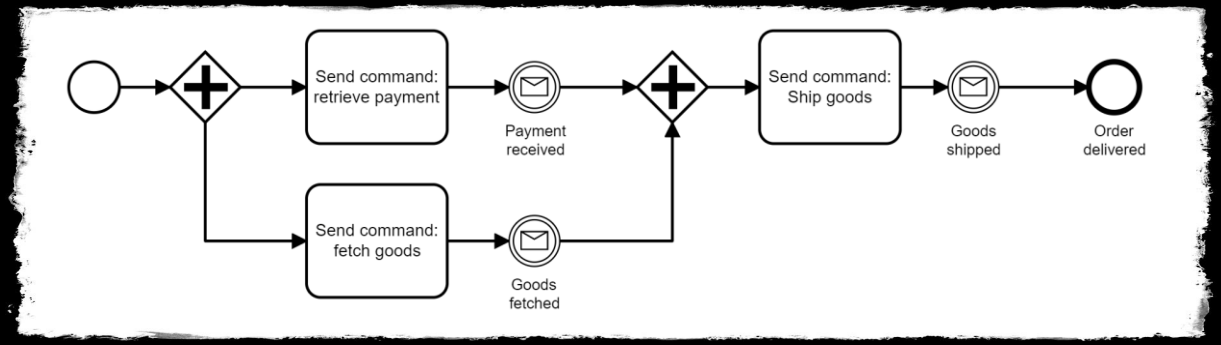
Event-driven:
Decision to couple is on the **receiving** side

Command-driven
Decision to couple is on the **sending** side

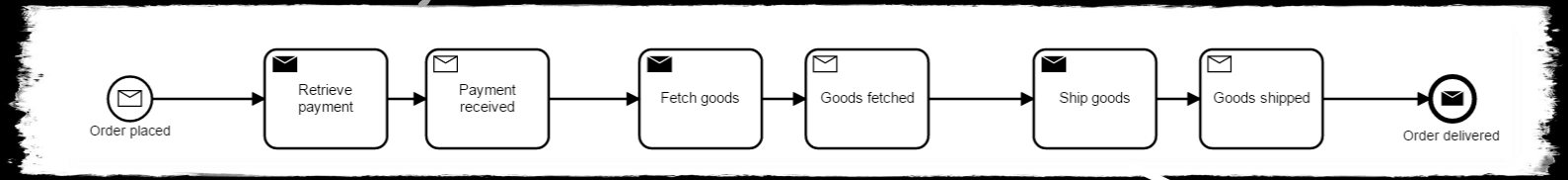
Direction of dependency

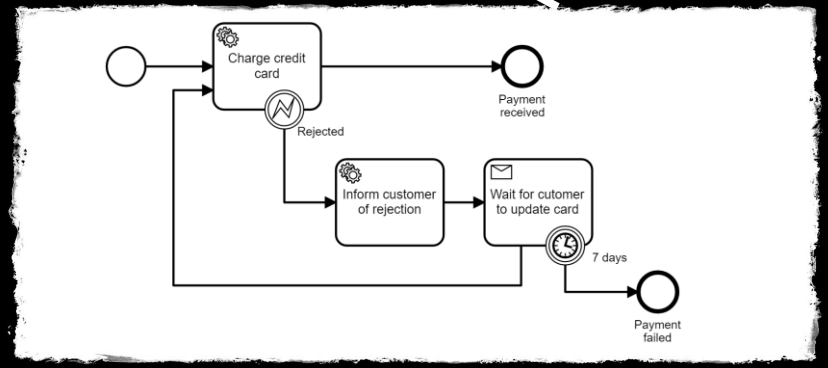
Now it is easy to change the process flow





Processes are domain logic and live inside service boundaries

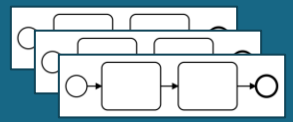




orchestration is not centralized

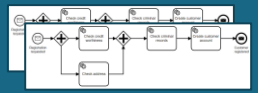
Every microservice (process solution) owns its process model, glue code, and any additional artifacts

Microservice A



Microservice B

Microservice C



Camunda

Camunda

Camunda

Ideally SaaS
(probably self-managed)

Self-service control plane

Console | Clusters | Modeler

Bernd Ruecker

Clusters

Create New Cluster

Name	Region	Generation	Status
1.3.1 Patch Release	Integration Worker	Zeebe 1.3.1 - update available	Healthy
Version 1.3.2 tests	Integration Worker	Zeebe 1.3.2 - update available	Healthy
Simon-Bernd G3-L Test	Integration Worker	Zeebe 8.0.0 - update available	Healthy
New_cluster_Geetha	Integration Worker	Zeebe 1.2.2 - update available	Healthy
QA Optimize Test3	Integration Worker	Zeebe 1.2.2 - update available	Healthy
Menski - Deleting Stuff	Integration Worker	Zeebe 8.0.0 - update available	Healthy
Acess-api-aut-test-feb	Integration Worker	Zeebe SNAPSHOT	Healthy

Some code?

berndruecker / flowing-retail Public

Unpin | Unwatch 116 | Fork 420 | Starred 1.2k

Code | Issues 6 | Pull requests 14 | Discussions | Actions | Projects | Wiki | Security 56 | Insights

master | flowing-retail / kafka / java / | Go to file | Add file

berndruecker Added payment microservice alternative using Zeebe (related to #73) 27bc0ee 7 days ago History

README.md	adjusted readme to latest version/ports	7 days ago
pom.xml	added build for event ingestion to CI	2 years ago

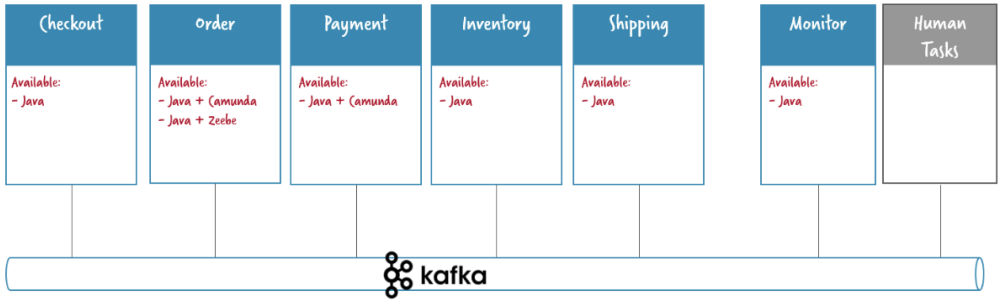
README.md

Flowing Retail / Apache Kafka / Java

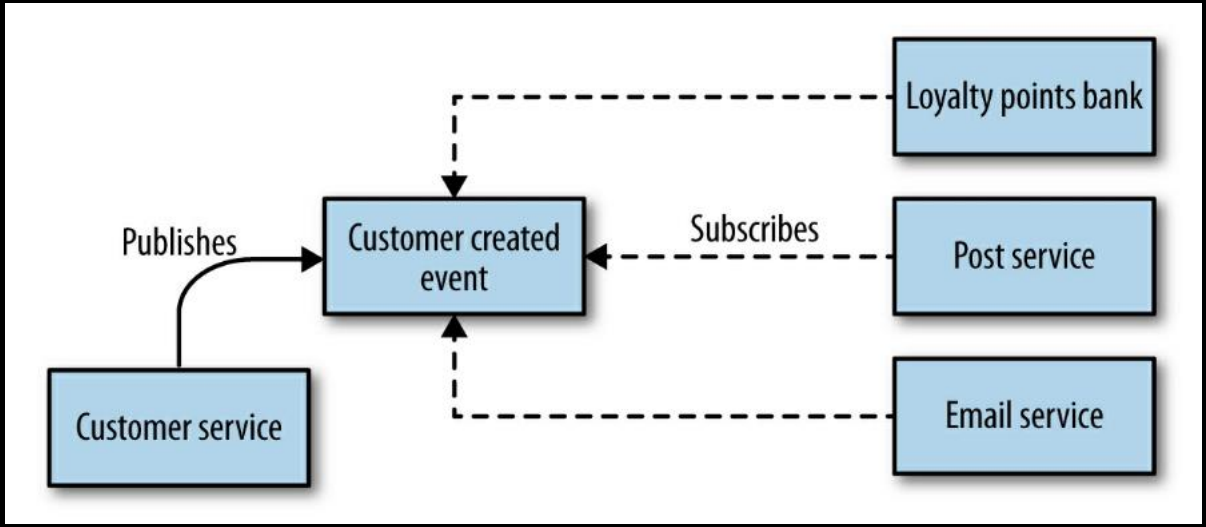
This folder contains services written in Java that connect to Apache Kafka as means of communication between the services.

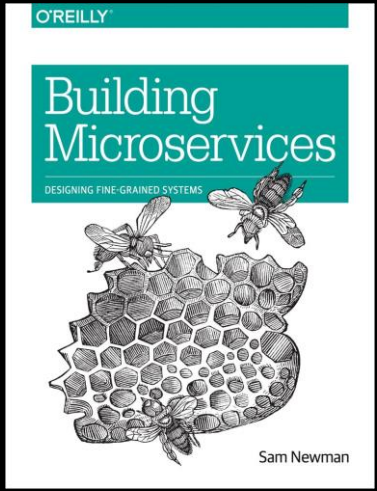
Tech stack:

- Java 8
- Spring Boot 2.6.x
- Apache Kafka (and Spring Kafka)
- Camunda Zeebe 8.x (and Spring Zeebe)



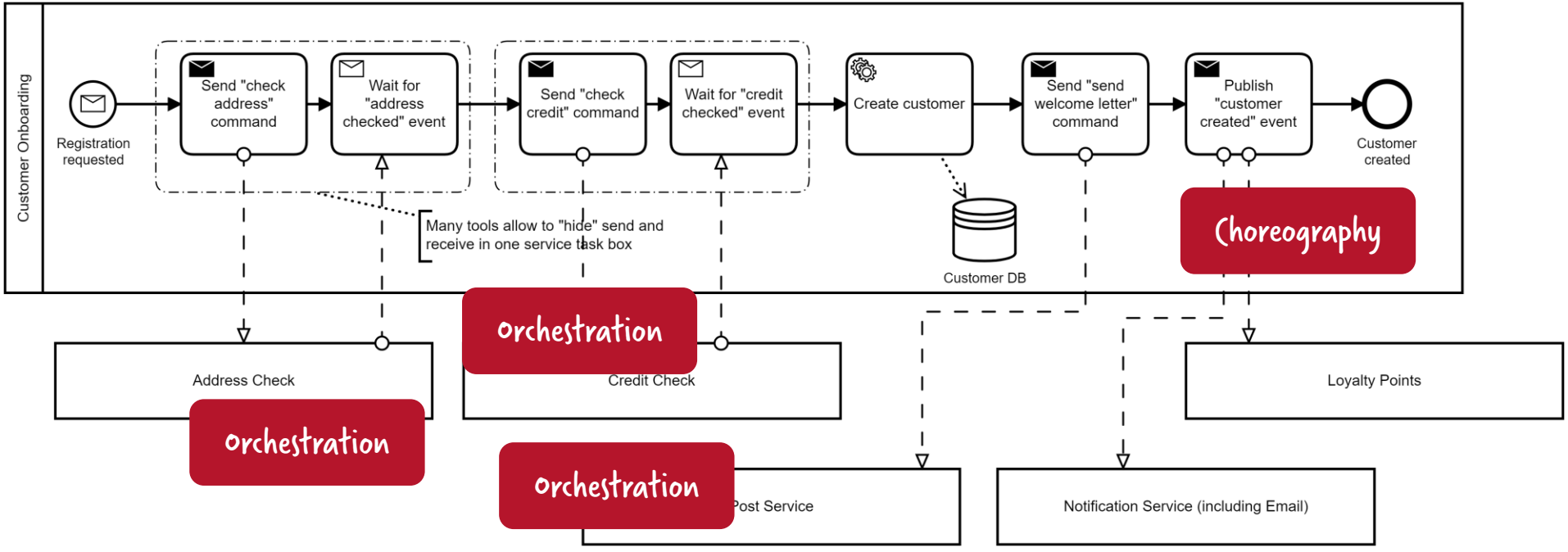
https://github.com/berndruecker/flowing-retail/tree/master/kafka





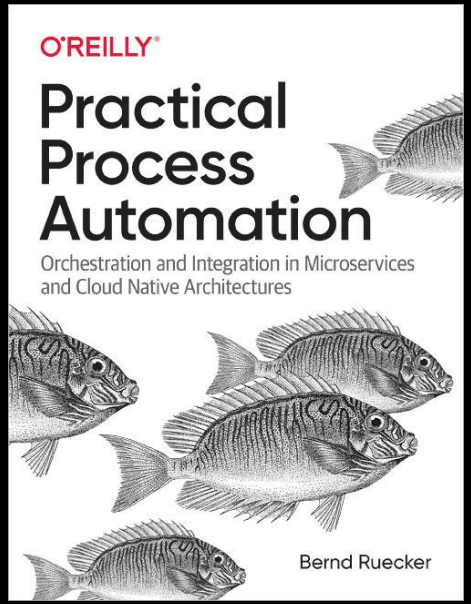
Sam Newman: Building Microservices

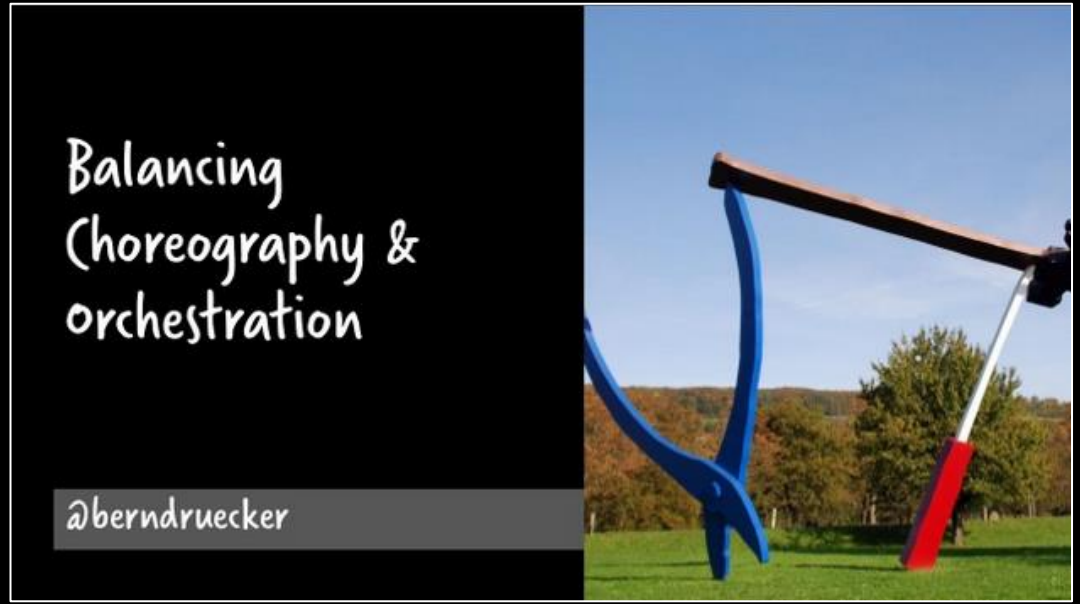
Mix orchestration and choreography



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





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

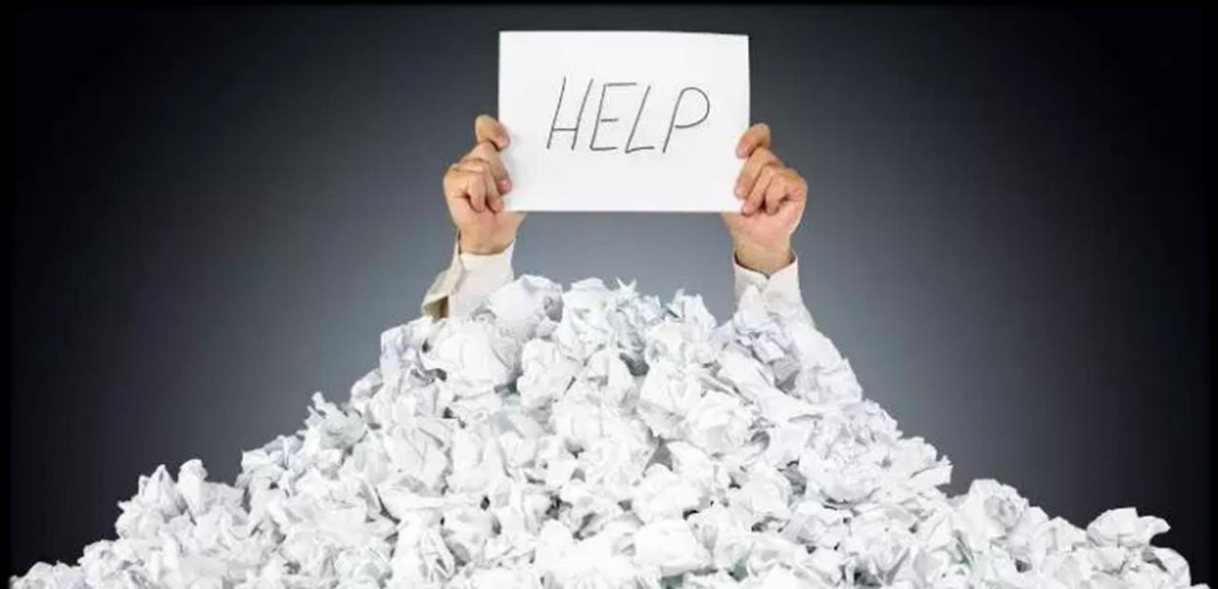
Type of coupling	Description	Example	Recommendation
Implementation Coupling	Service knows internals of other services	Joined database	**Avoid**
Temporal Coupling	Service depends on availability of other services	Synchronous blocking communication	**Reduce or manage**
Deployment Coupling	Multiple services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

Type of coupling	Recommendation
Implementation Coupling	**Avoid**
Temporal Coupling	**Reduce or manage**
Deployment Coupling	Typically **avoid**, but depends
Domain Coupling	**Unavoidable** unless you change business requirements or service boundaries

The **communication** style can reduce temporal coupling

Some people refer to this, when they say that event-driven systems decouple better.
But in reality, it just turns the direction of the dependency around.
The **collaboration** style does **not** decouple!

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)

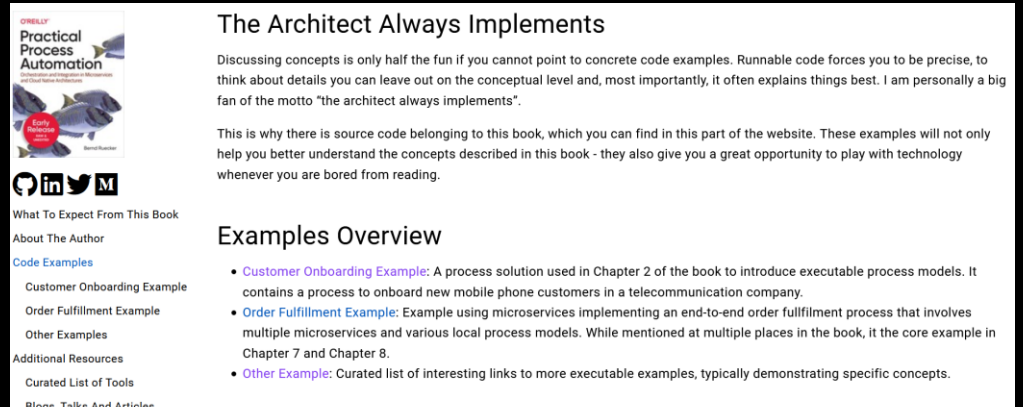
...

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/



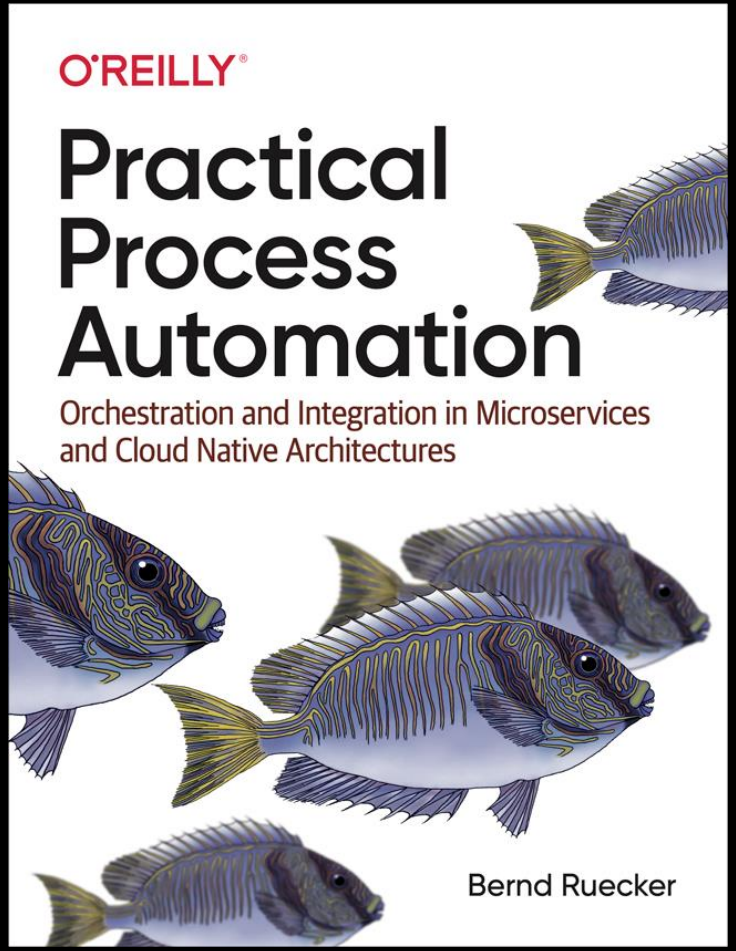
The Architect Always Implements

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.




Thank you!
@bernduecker