How Process Orchestration Increases Agility Without Harming Architecture

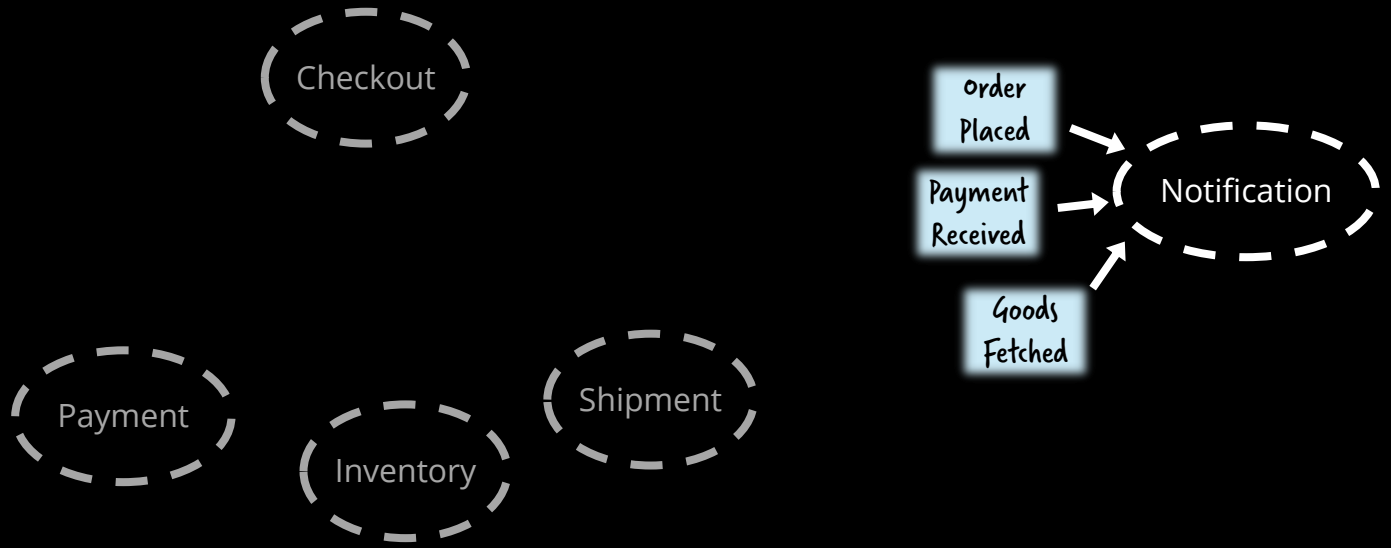
@berndruecker

Choreography is great!

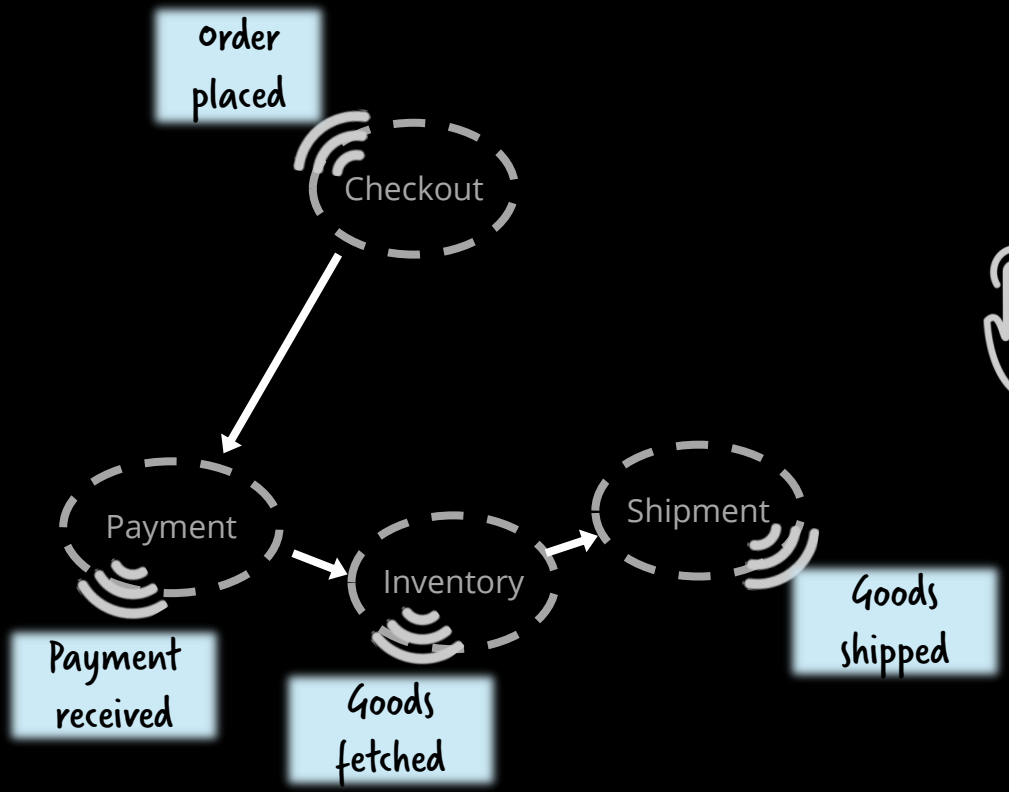
What we wanted

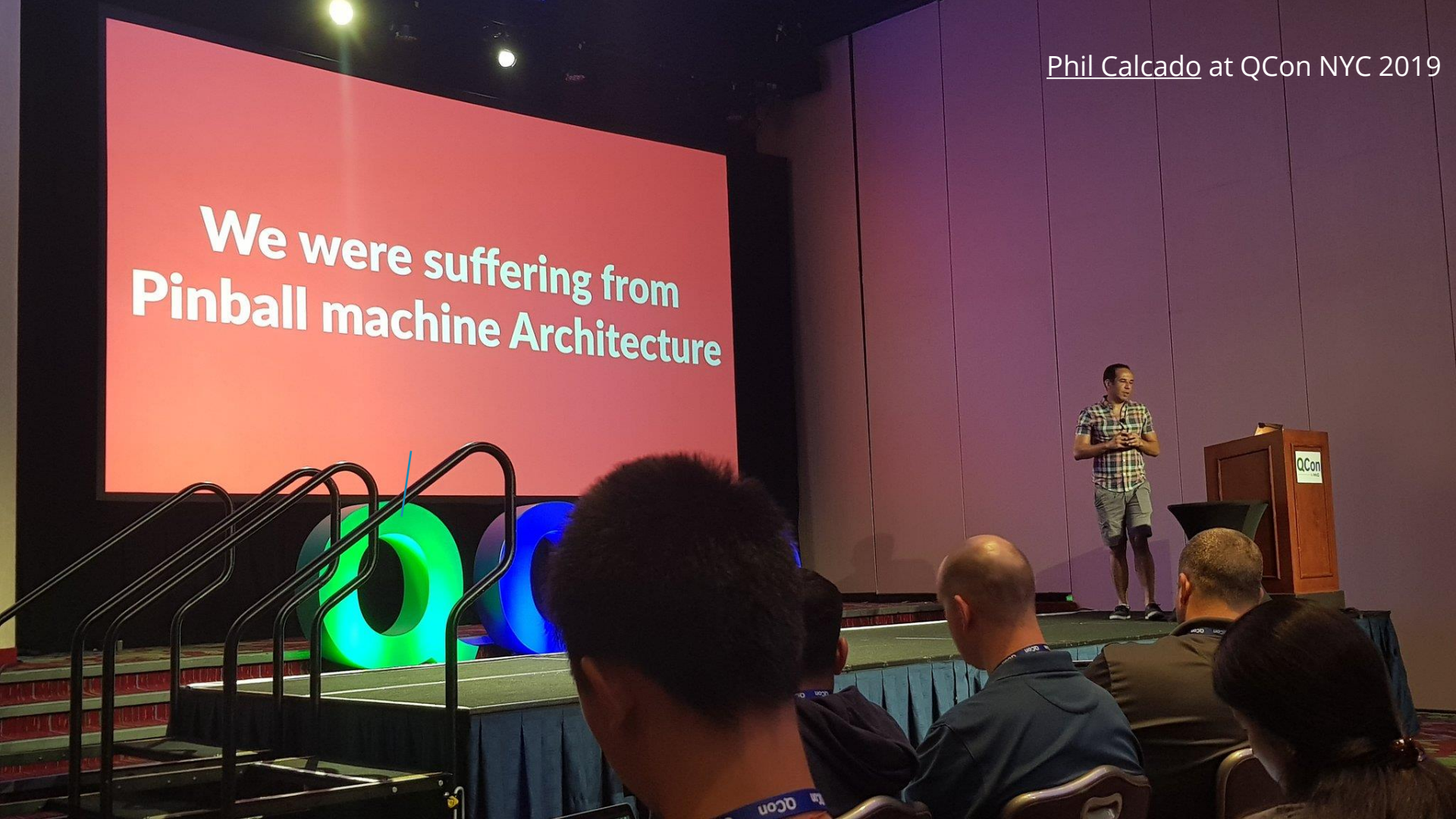
vs. what we got

Event-driven



Peer-to-peer event chains

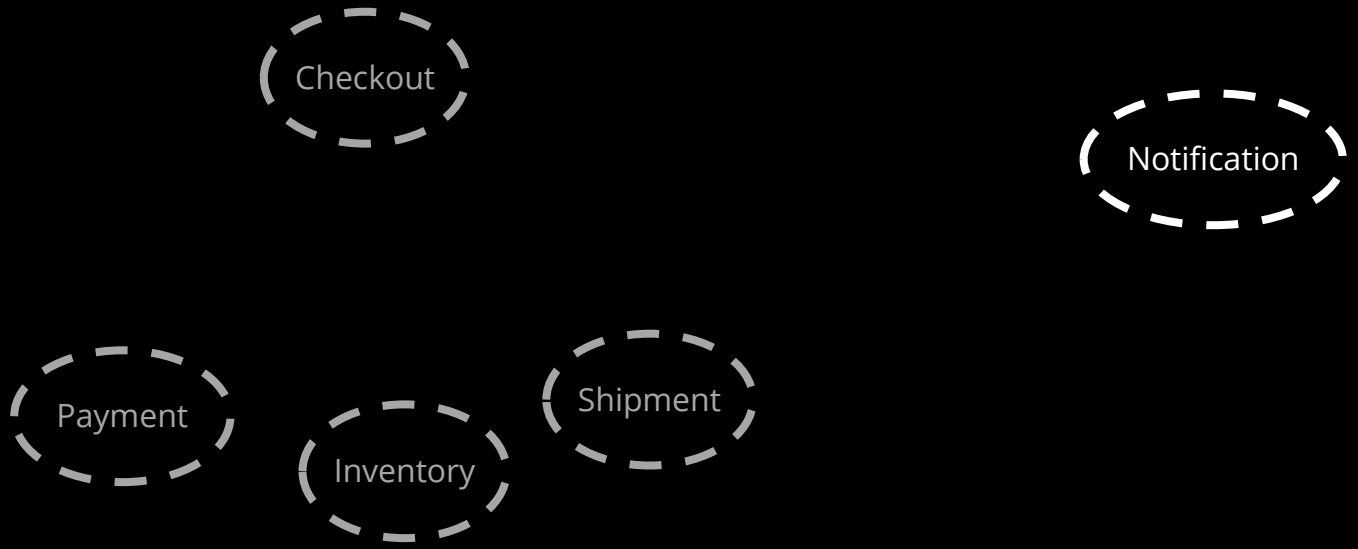




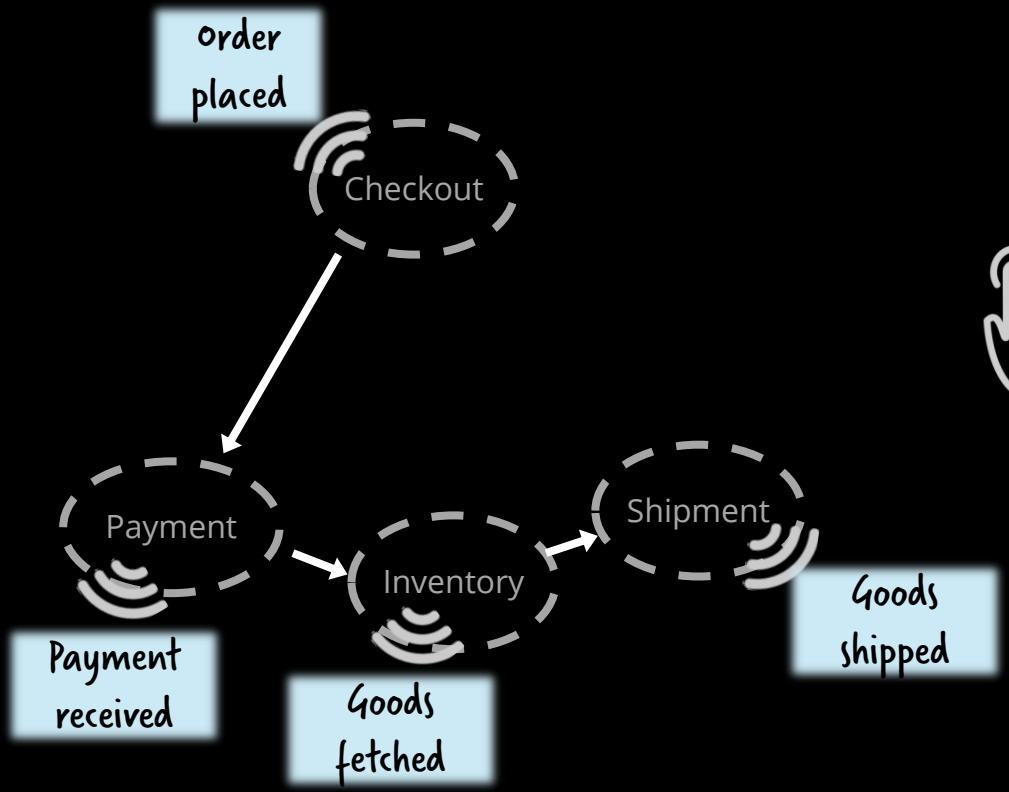
Phil Calcado at QCon NYC 2019

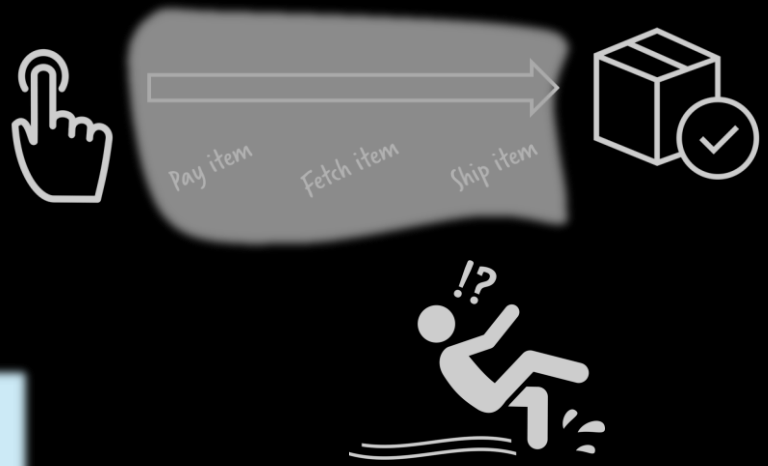
Pinball Machine Archite

„What the hell just happened?"

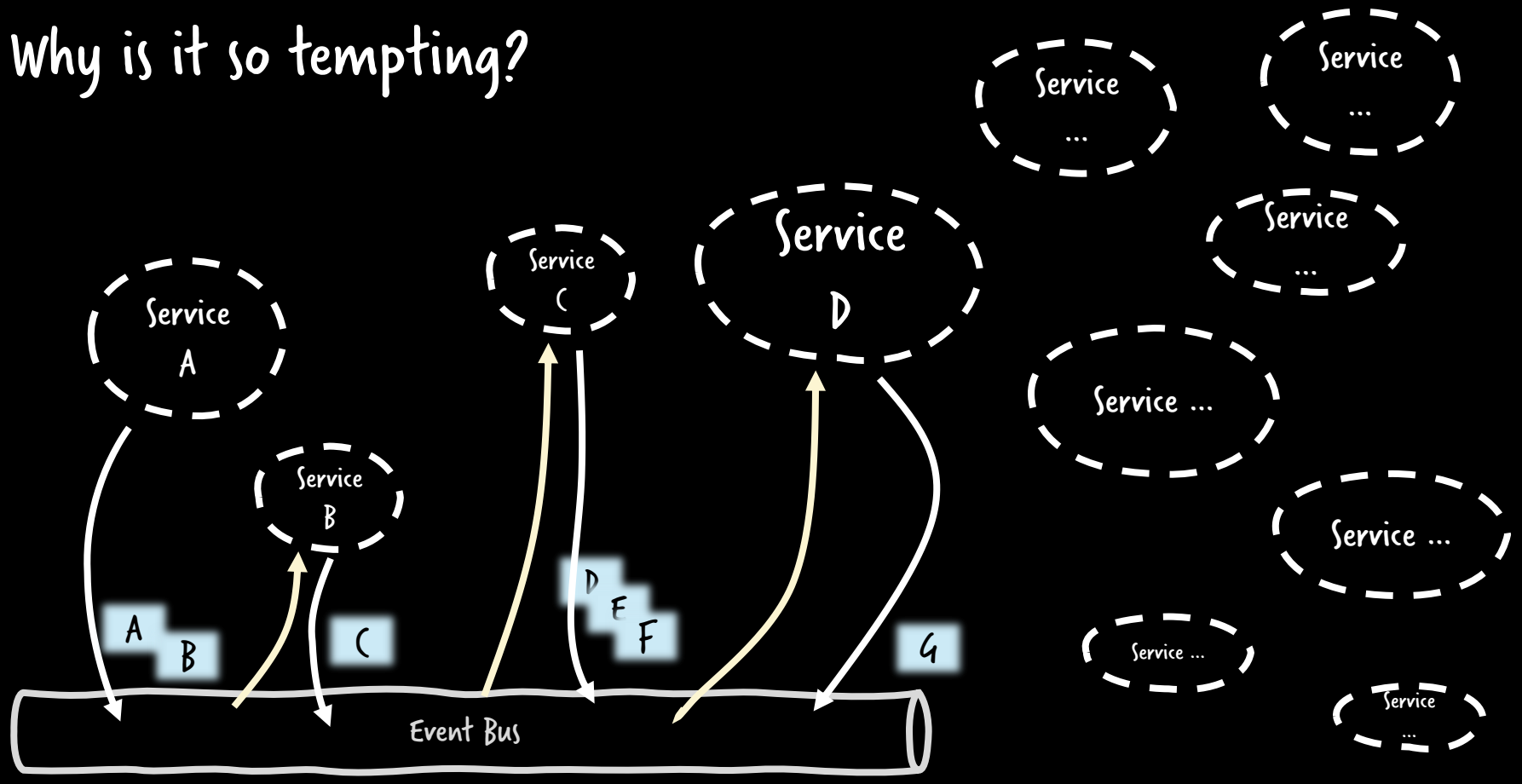


Peer-to-peer event chains

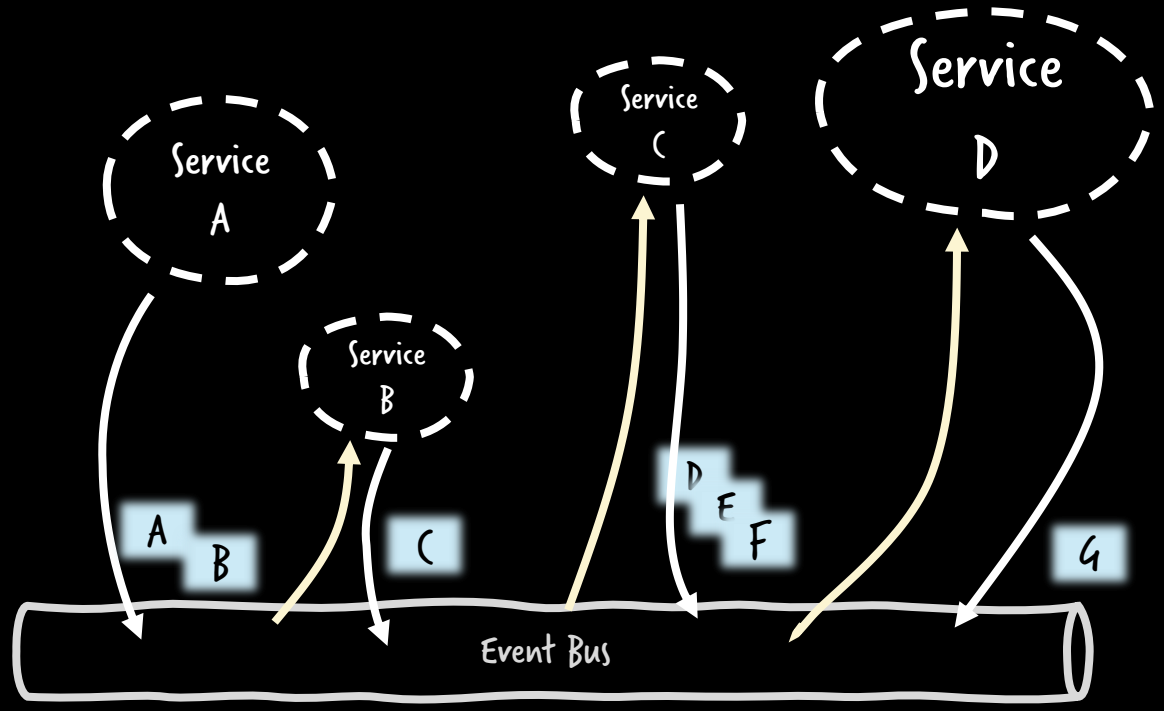




Why is it so tempting?



Why is it so tempting?



Adding is easy!

You can „buy" a shorter initial time-to-value by choreography.

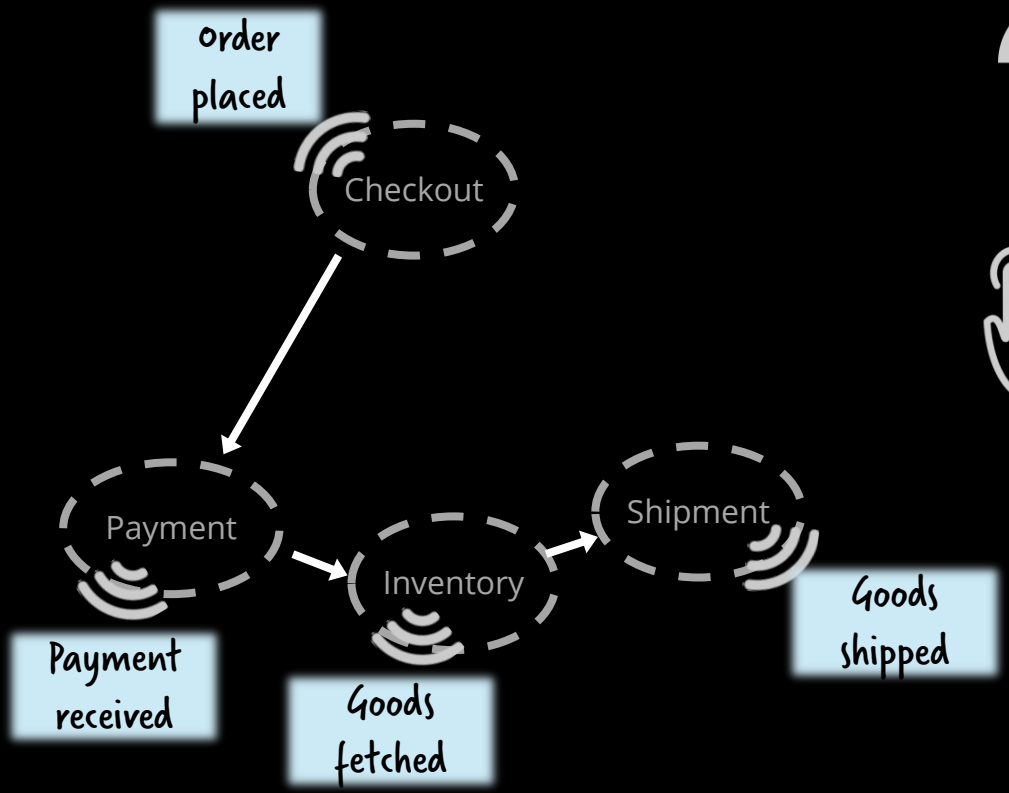
It yields in technical debt.

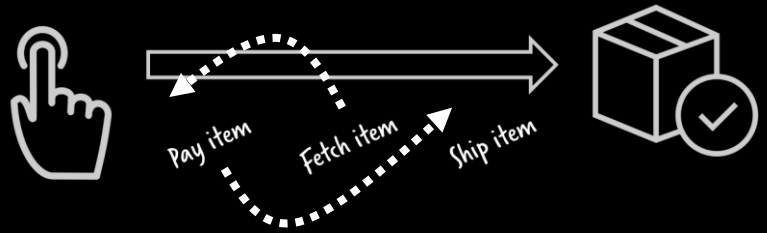
Service ...

Service ...

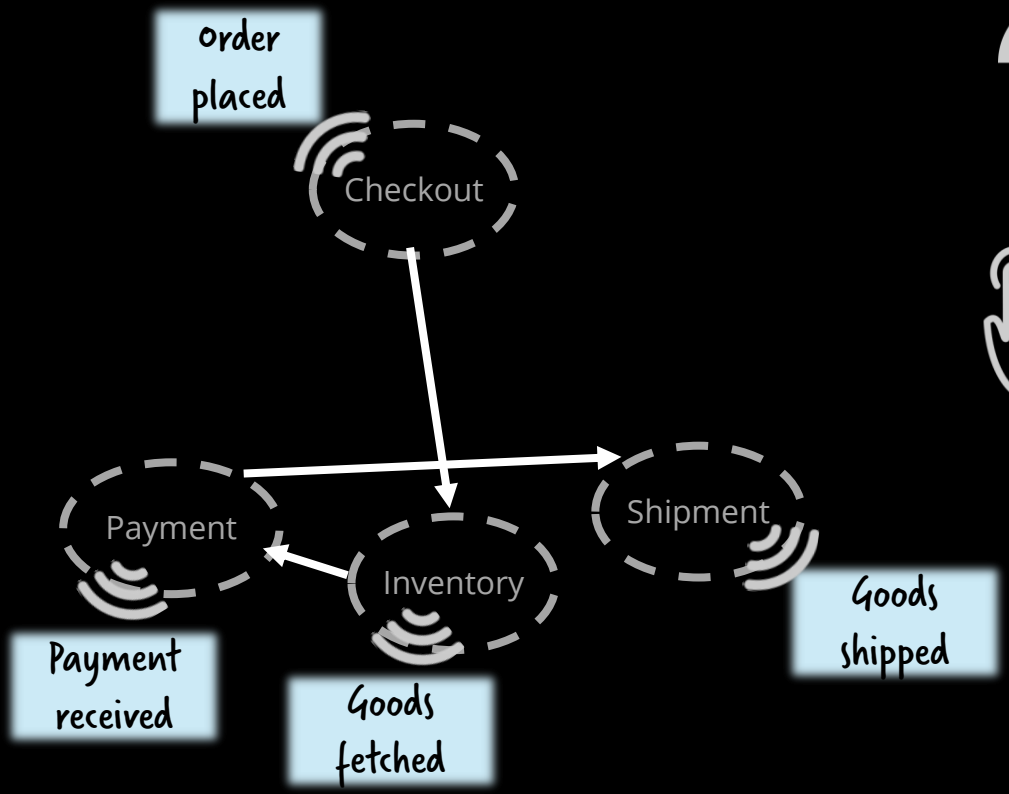
Service ...

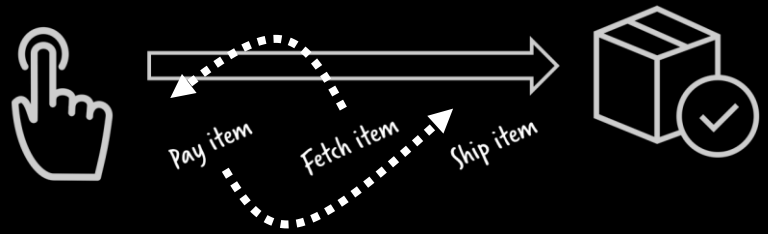
Peer-to-peer event chains





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

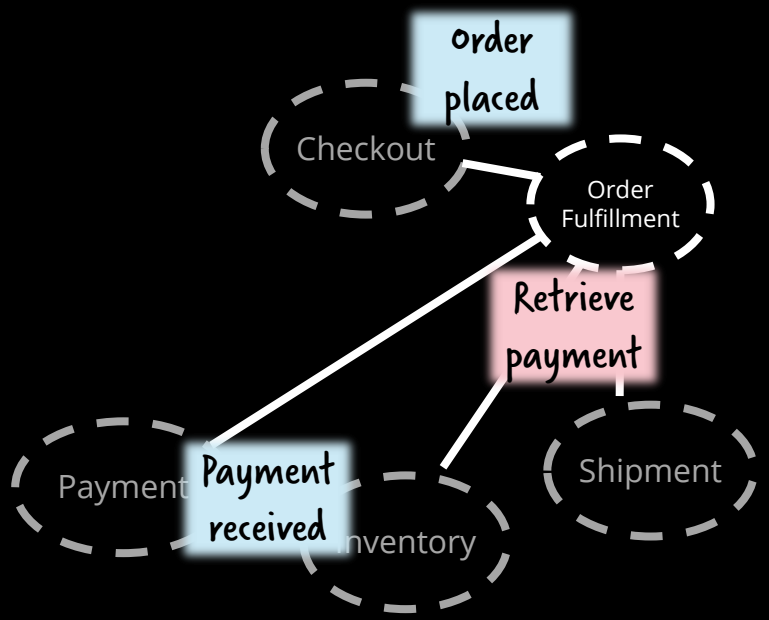
The danger is that it's very easy to make nicely decoupled systems with event notification, without realizing that you're losing sight of that larger-scale flow, and thus set yourself up for trouble in future years.

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

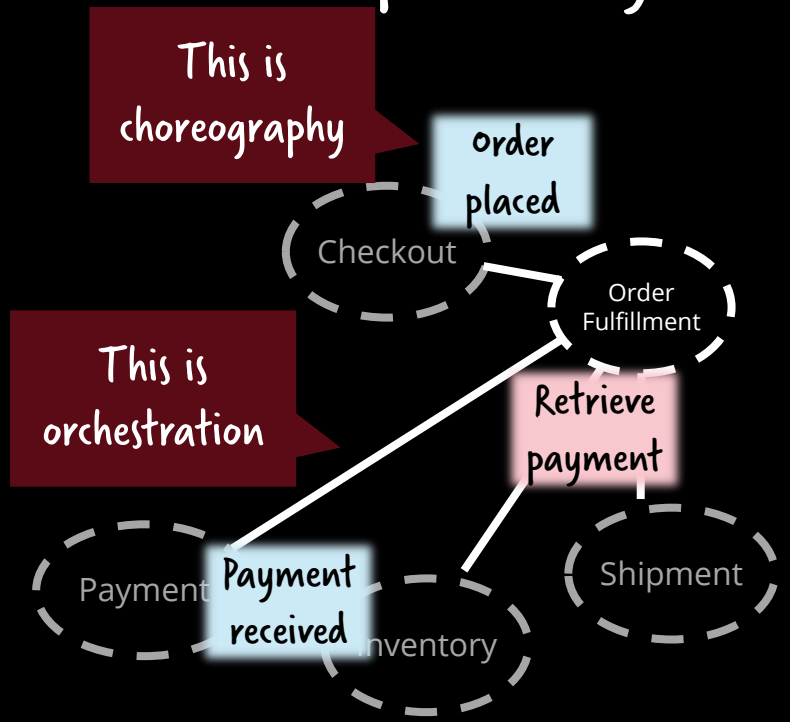
The danger is that it's very easy to make nicely decoupled systems with event notification, without realizing that you're losing sight of that larger-scale flow, and thus **set yourself up for trouble in future years**.

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

Extract the domain logic around order fulfillment



Decide about responsibility



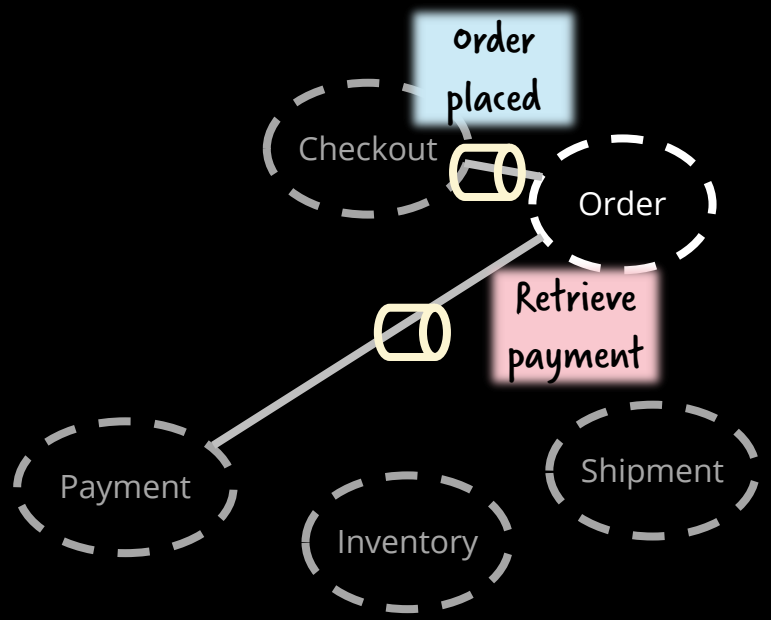
My definition

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

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.

It is not about the protocol!



It can still be messaging!

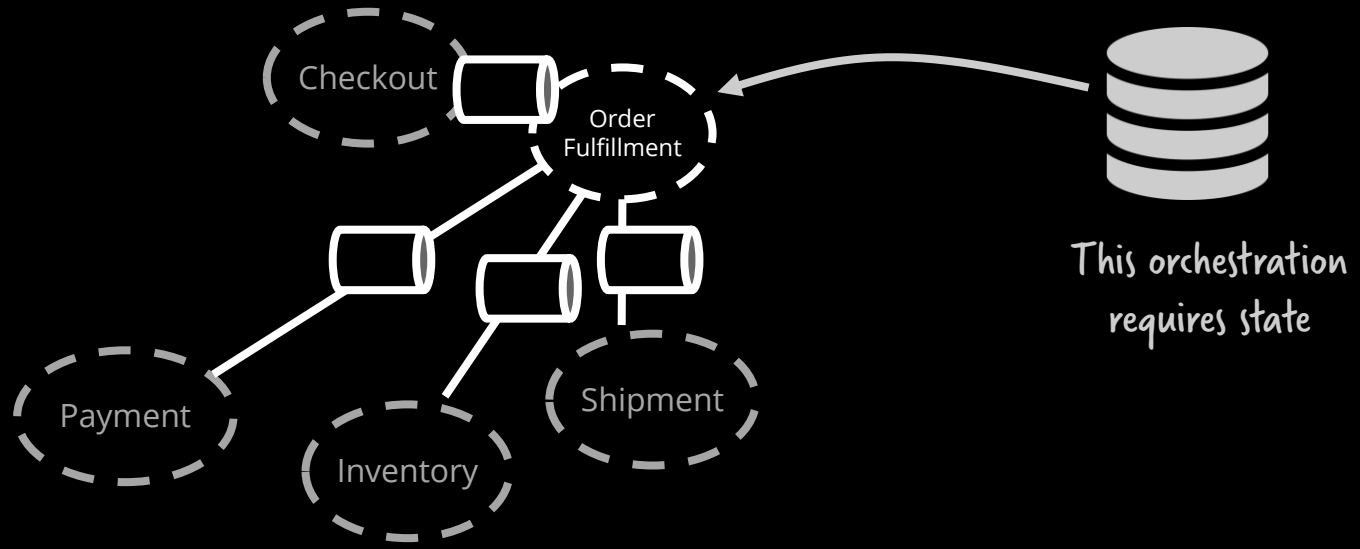
Communication vs. Collaboration Style

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!

See https://www.slideshare.net/BerndRuecker/jcon-2021-loosely-or-lousily-coupled

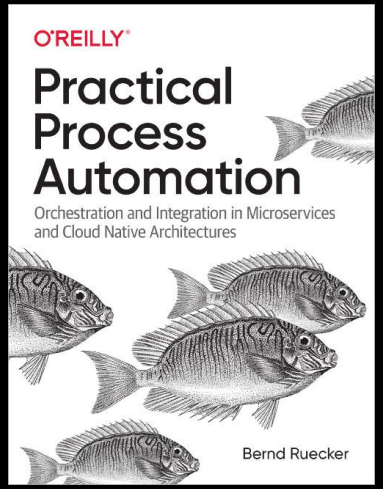
orchestration can be stateful / long running

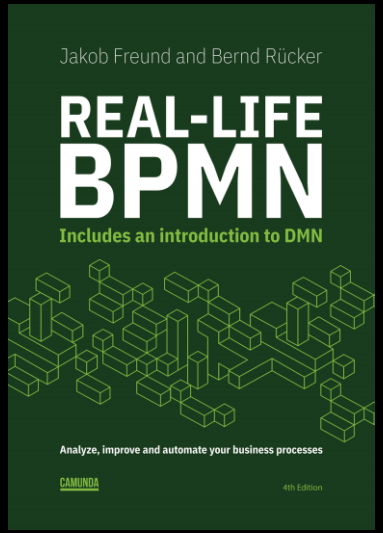


Warning:
Contains Opinion

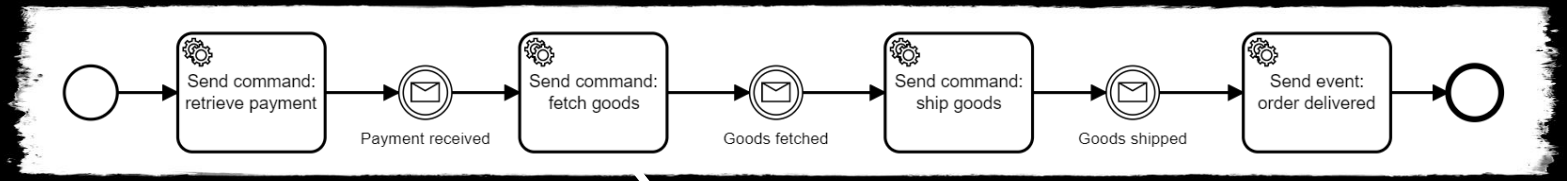
Bernd Ruecker

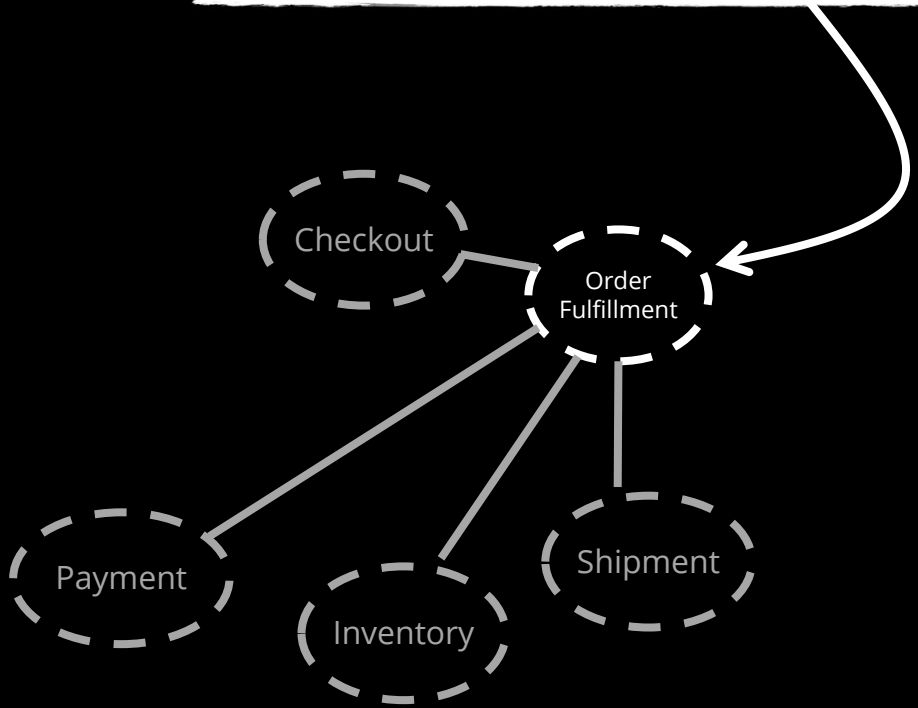
Co-founder and Chief Technologist of Camunda





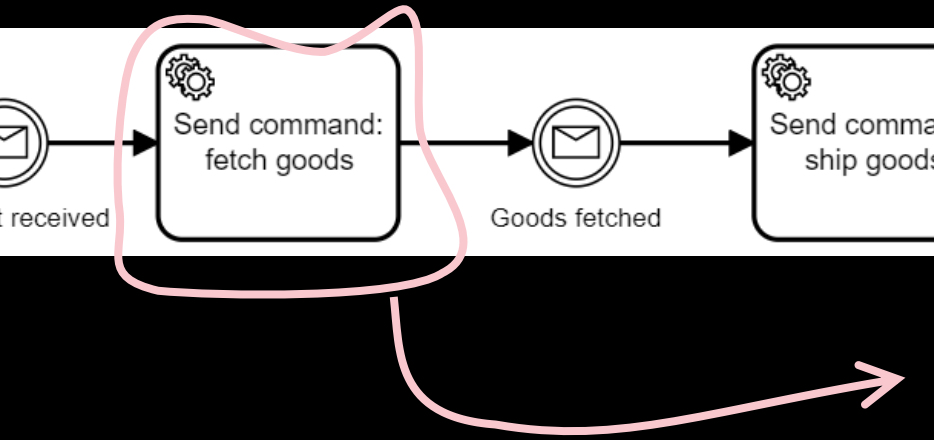
mail@berndruecker.io
@berndruecker
http://berndruecker.io/





@berndruecker

Glue code (e.g. Java)




```java
@Component
public class FetchGoodsAdapter {

  @Autowired
  private MessageSender messageSender;

  @Autowired
  private OrderRepository orderRepository;

  @ZeebeWorker(type = "fetch-goods")
  public void handle(JobClient client, ActivatedJob job) {
    OrderFlowContext context = OrderFlowContext.fromMap(job.getVariablesAsMap());
    Order order = orderRepository.findById( context.getOrderId() ).get();

    // generate an UUID for this communication
    String correlationId = UUID.randomUUID().toString();

    messageSender.send(new Message<FetchGoodsCommandPayload>( //
            "FetchGoodsCommand", //
            context.getTraceId(), //
            new FetchGoodsCommandPayload() //
              .setRefId(order.getId()) //
              .setItems(order.getItems())) //
        .setCorrelationid(correlationId));

    client.newCompleteCommand(job.getKey()) //
      .variables(Collections.singletonMap("CorrelationId_FetchGoods", correlationId))
      .send().join();
  }

}
```

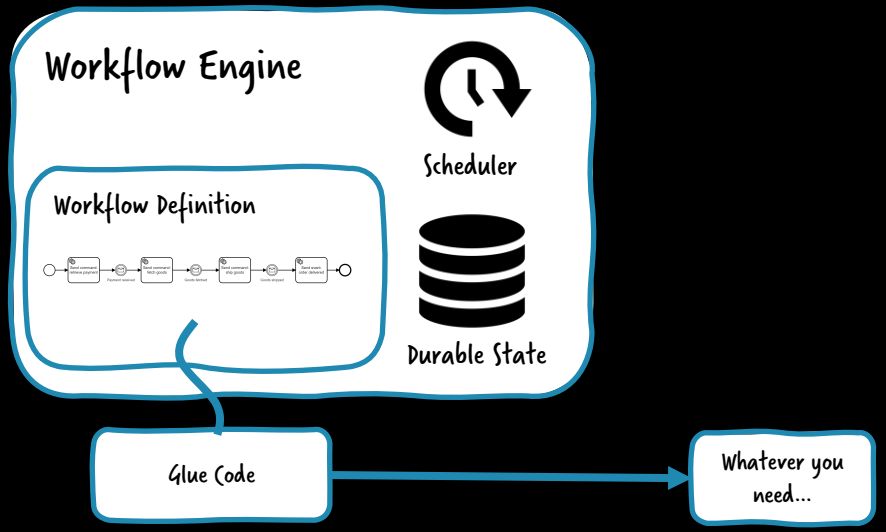
https://github.com/berndruecker/flowing-retail/blob/master/kafka/java/order-zeebe/src/main/java/io/flowing/retail/kafka/order/flow/FetchGoodsAdapter.java

Out-of-the-box Connectors

- REST API
- SendGrid
- Slack
- Microsoft Teams
- Google Drive
- Automation Anywhere
- Kafka Producer
- Amazon SQS
- AWS Lambda
- Amazon SNS
- RabbitMQ
- GraphQL
- GitLab
- OpenAI
- Camunda Operate
- MessageBird
- Twilio
- EasyPost
- GitHub
- Asana
- Google Maps
- UiPath
- Microsoft Power Automate

... and more

Using a workflow engine



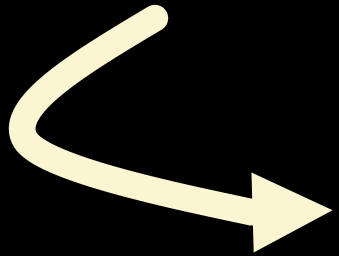
Workflow Engine:

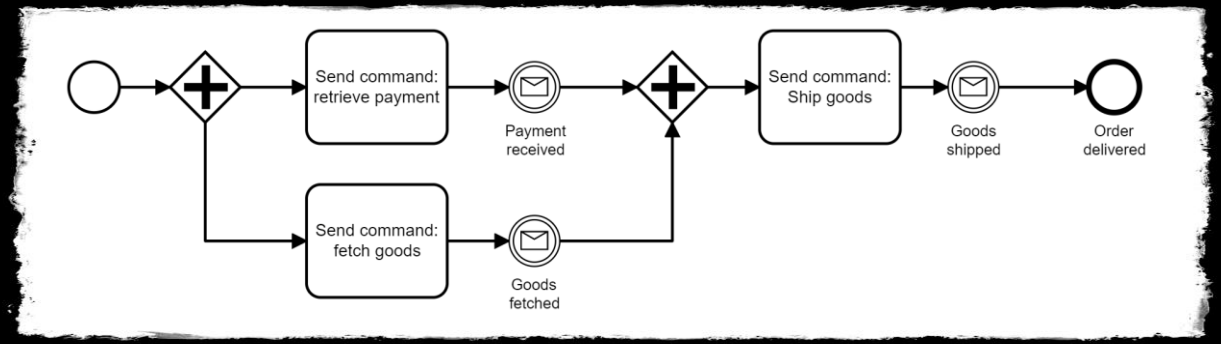
Is stateful

Can wait
Can retry
Can escalate
Can compensate

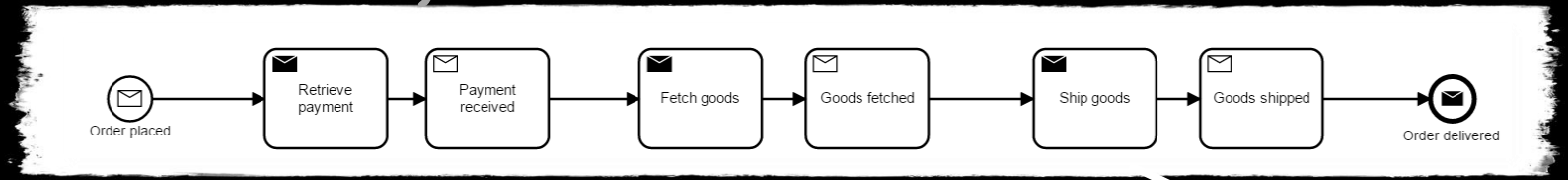
Provides visibility

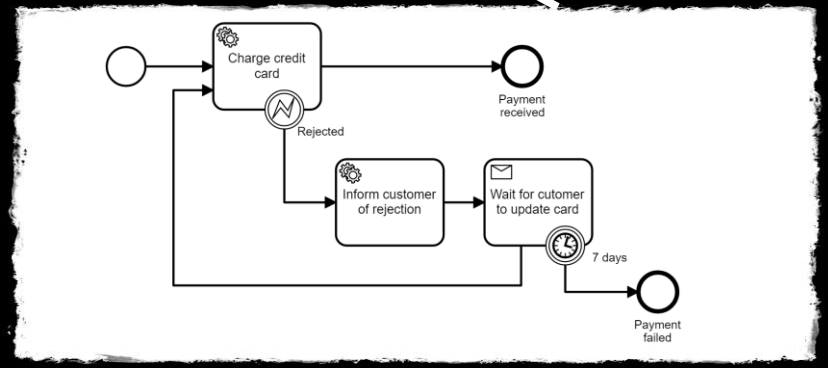
Now it is easy to change the orchestration logic



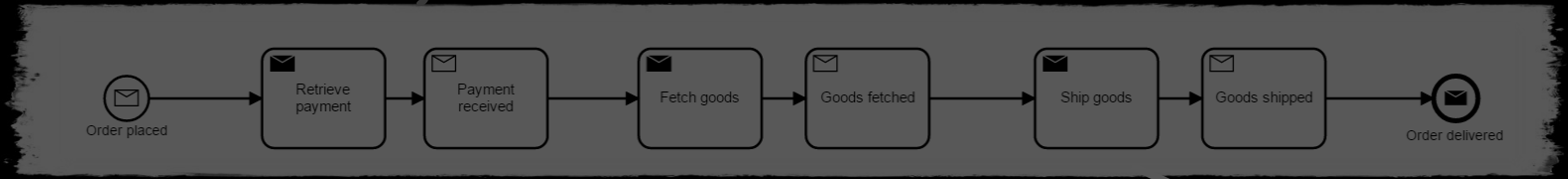


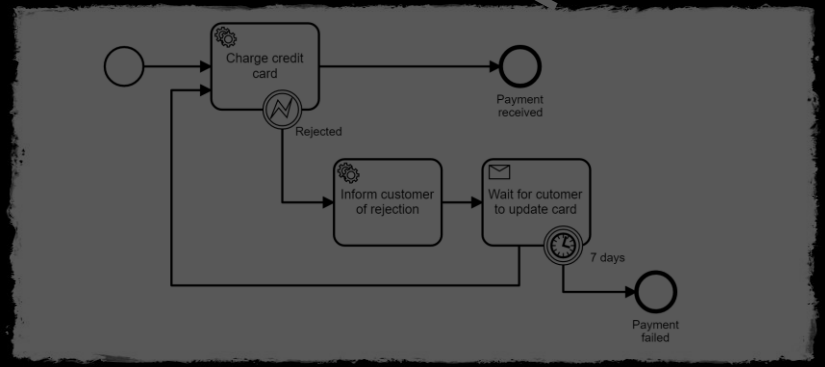
Processes are domain logic and live inside service boundaries





Processes are domain logic and live inside service boundaries





orchestration does **not** need to be central

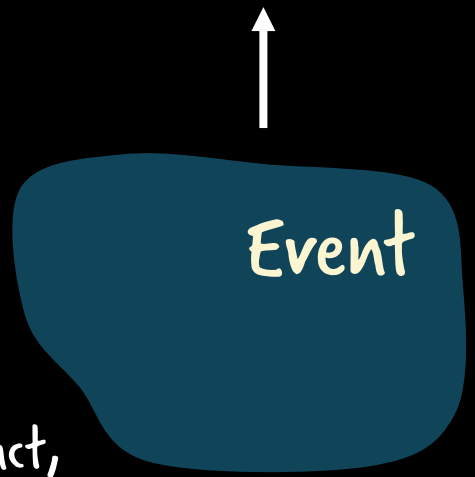
Challenge: Command vs. Event

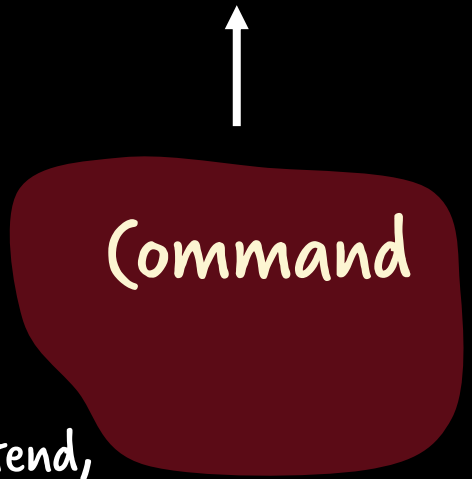
Command

vs

Event

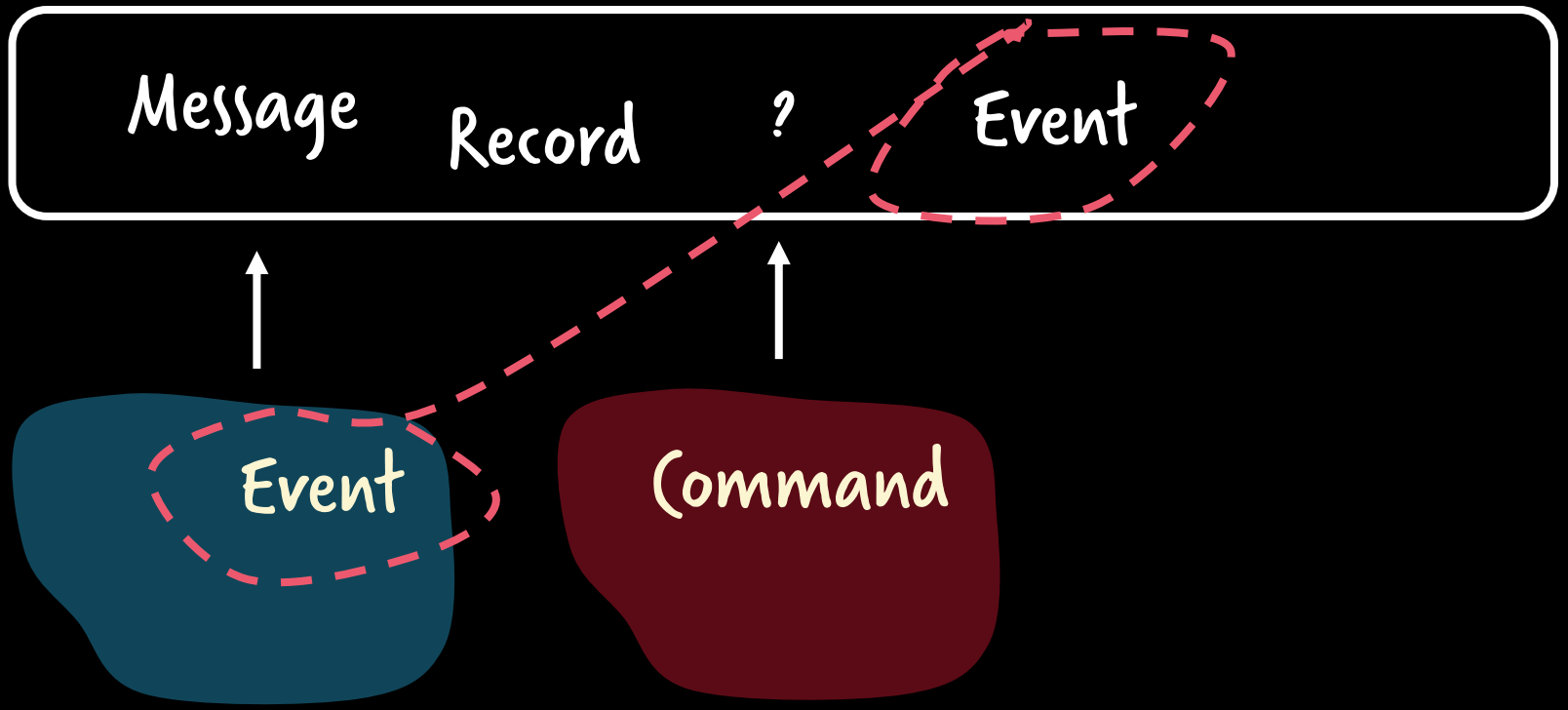
Message Record ? Event





Fact,
happened in the past,
immutable

Intend,
Want s.th. to happen,
The intention itself is a fact


Message
Record
?
Event
Event
Command

Commands in disguise

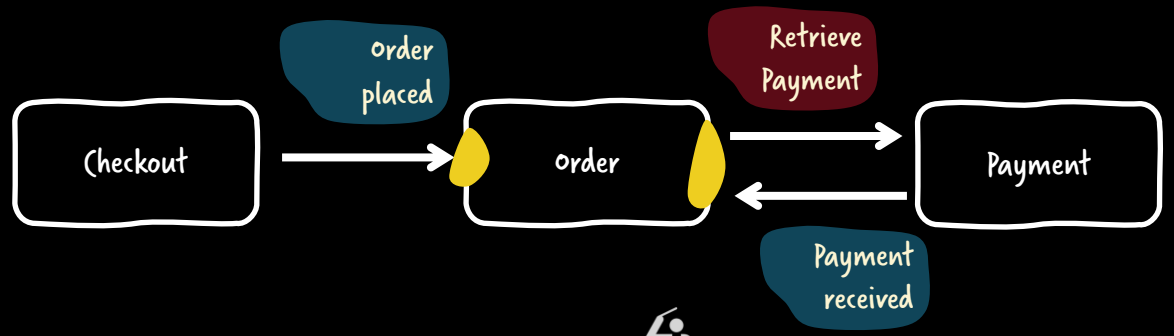
Wording of recipient

Send Message

The Customer Needs To Be Sent A Message To Confirm Address Change Event

Wording of Sender

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


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

This is influenced with the communication or collaboration style

Can be also reduced by other means than asynchronous messaging!

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

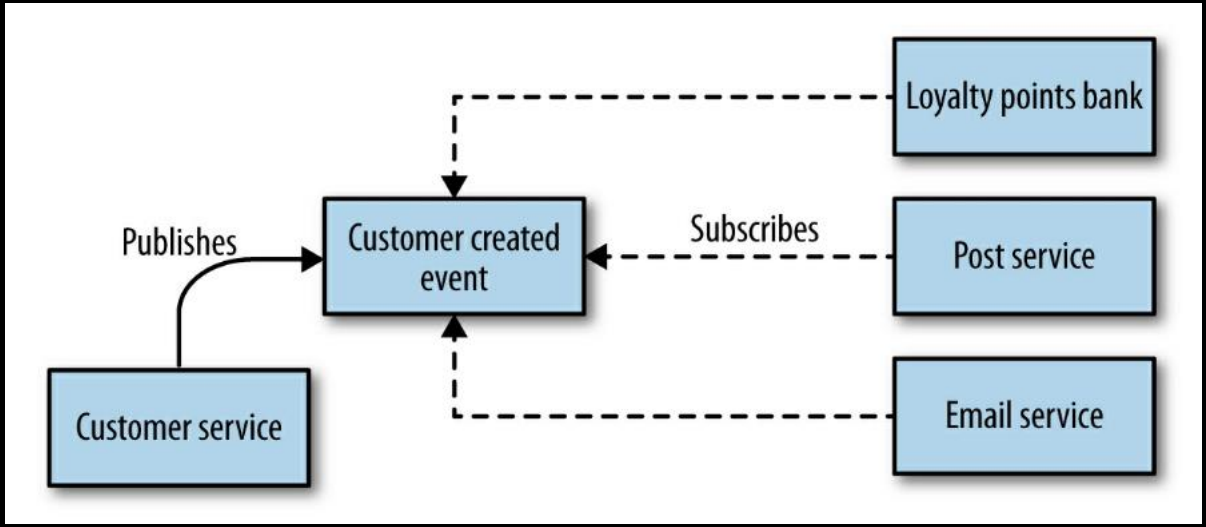
Types of Coupling

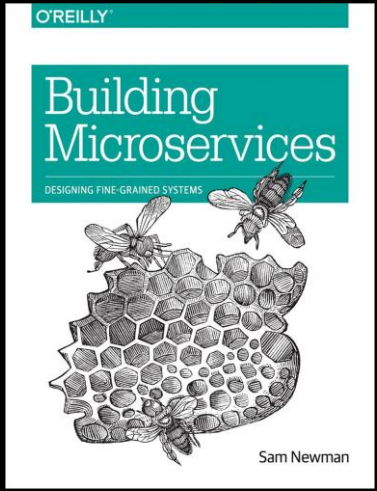
Type of coupling	Description	Example	Recommendation
Implementation Coupling	Service knows internals of other services	Joined database	**Avoid**
Temporal Coupling	Service depends on availability of other services	Synchronous blocking communication	**Reduce or manage**
Deployment Coupling	Multiple services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

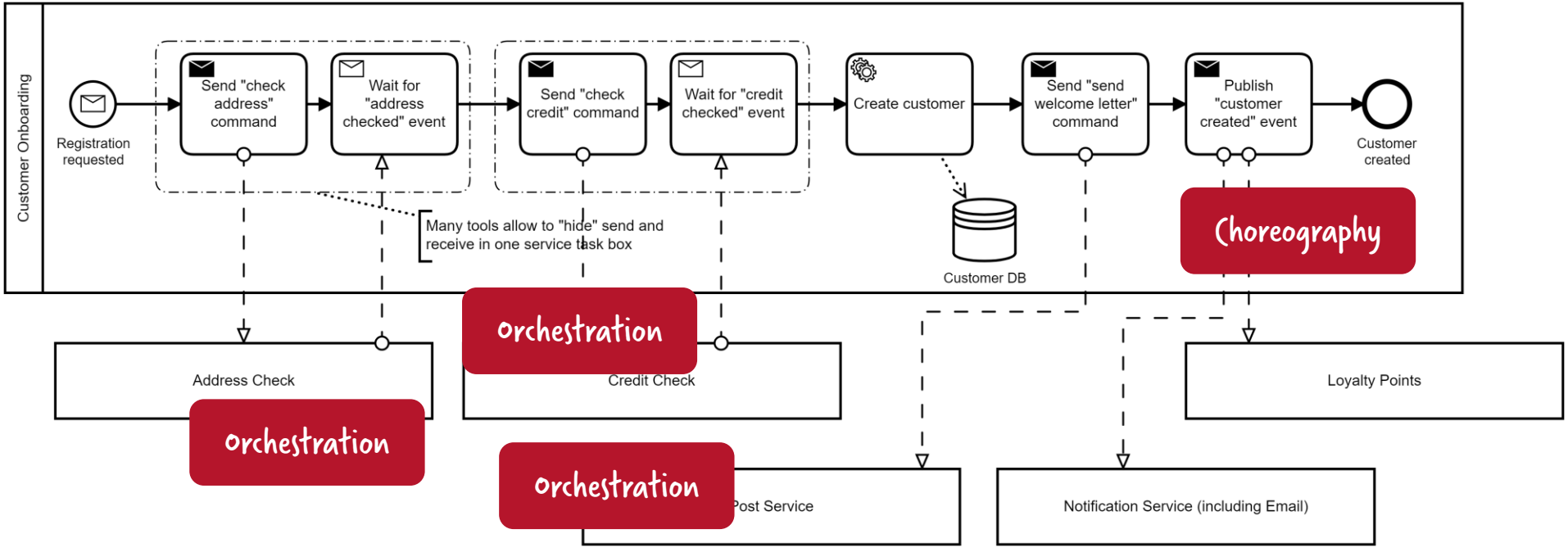
Customer Created



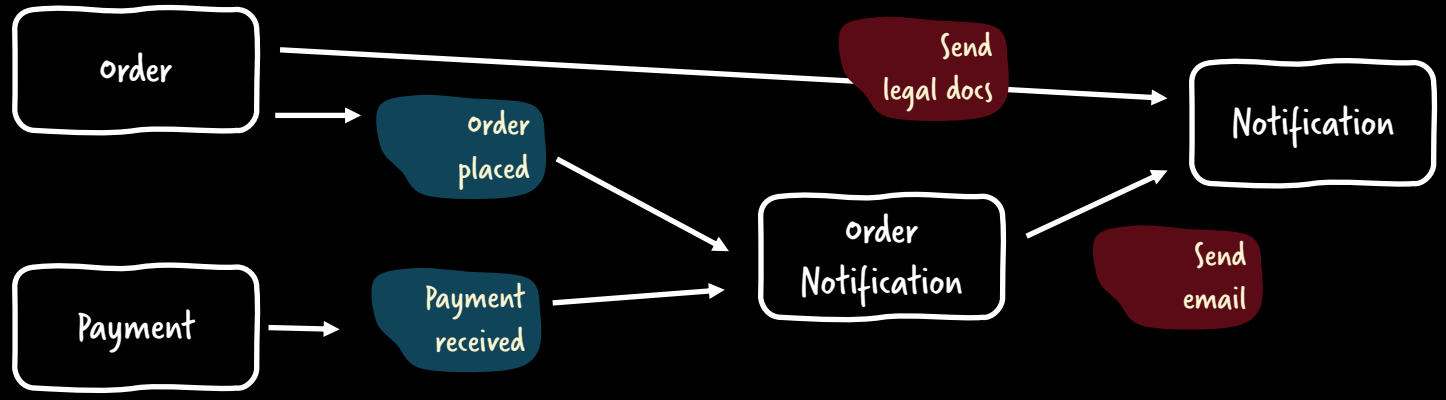


Sam Newman: Building Microservices

Customer onboarding is a mix!



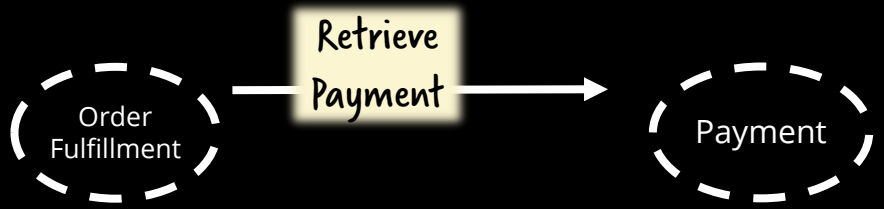
It is all about responsibility!



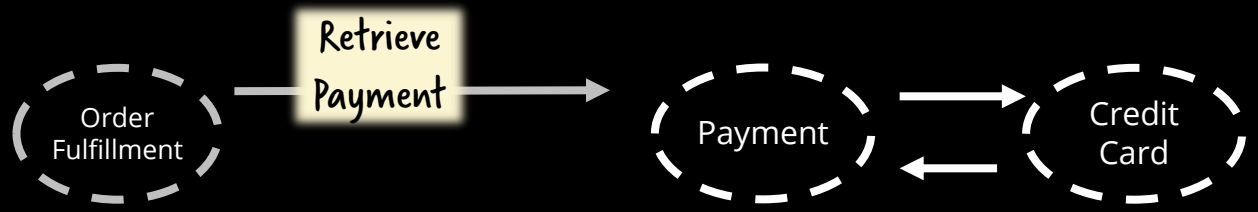
Long running capabilities are essential to design good service boundaries

(= a good architecture)

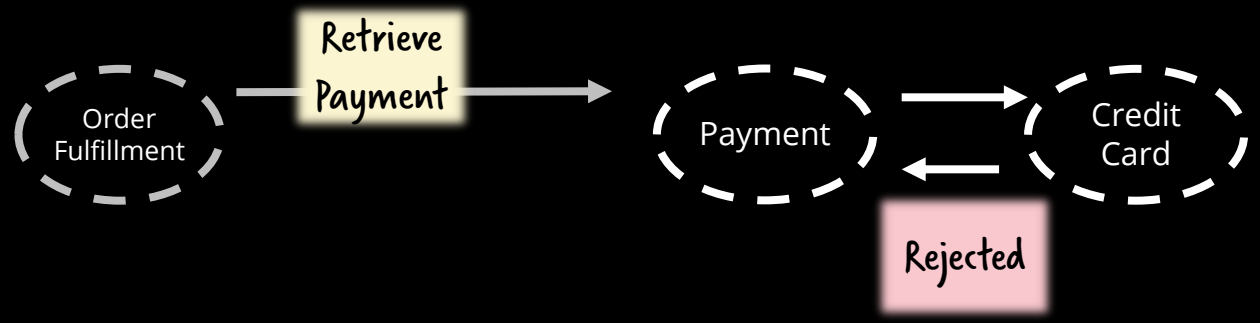
Example



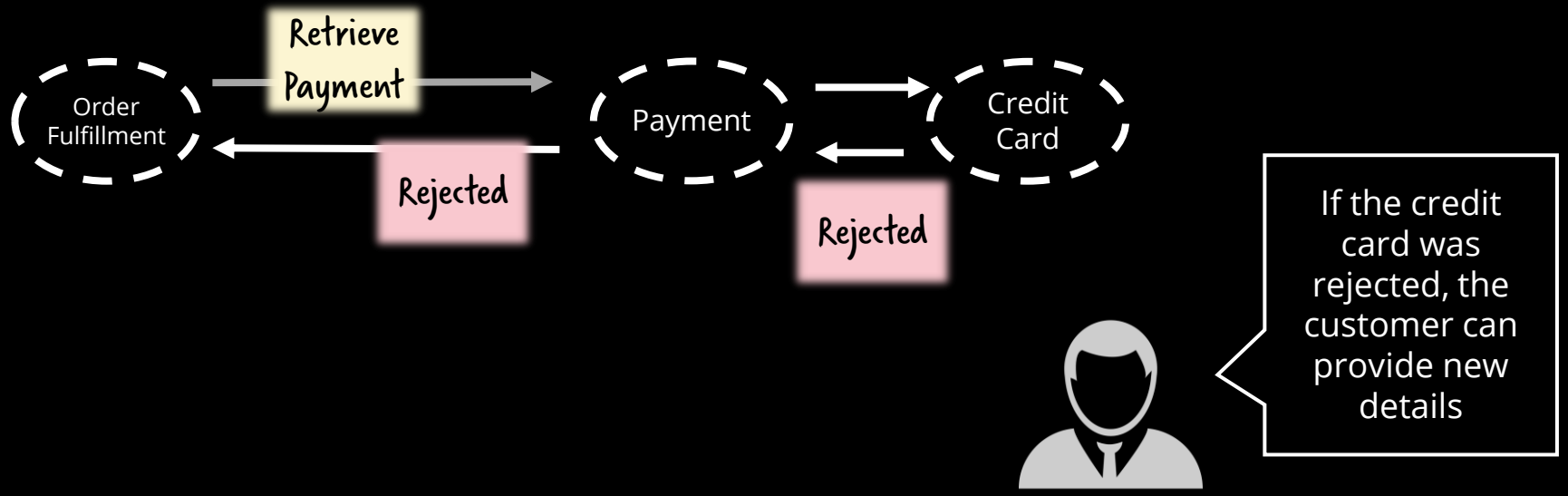
Example



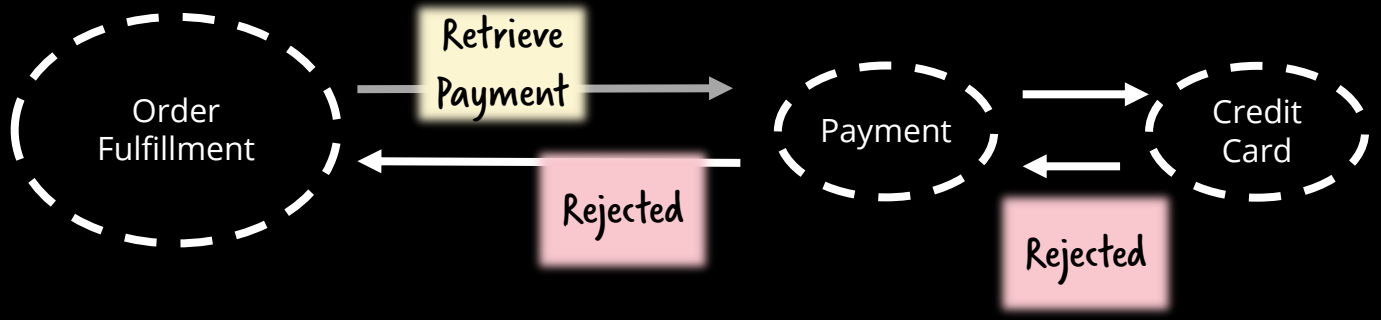
Example


Order Fulfillment
Retrieve Payment
Payment
Credit Card
Rejected

Example


Order Fulfillment
Retrieve Payment
Payment
Credit Card
Rejected
Rejected
If the credit card was rejected, the customer can provide new details

Example

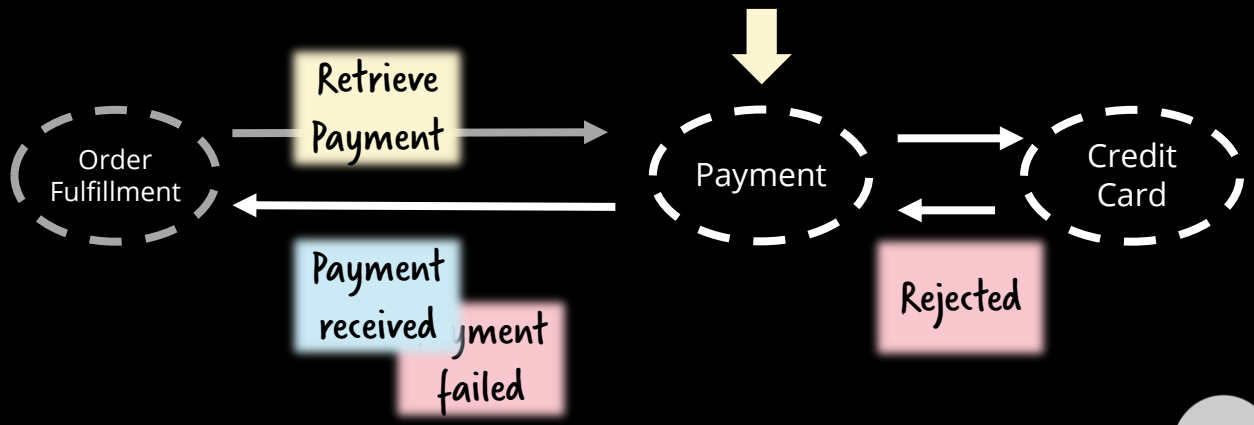


If the credit card was rejected, the customer can provide new details

Sam Newmann

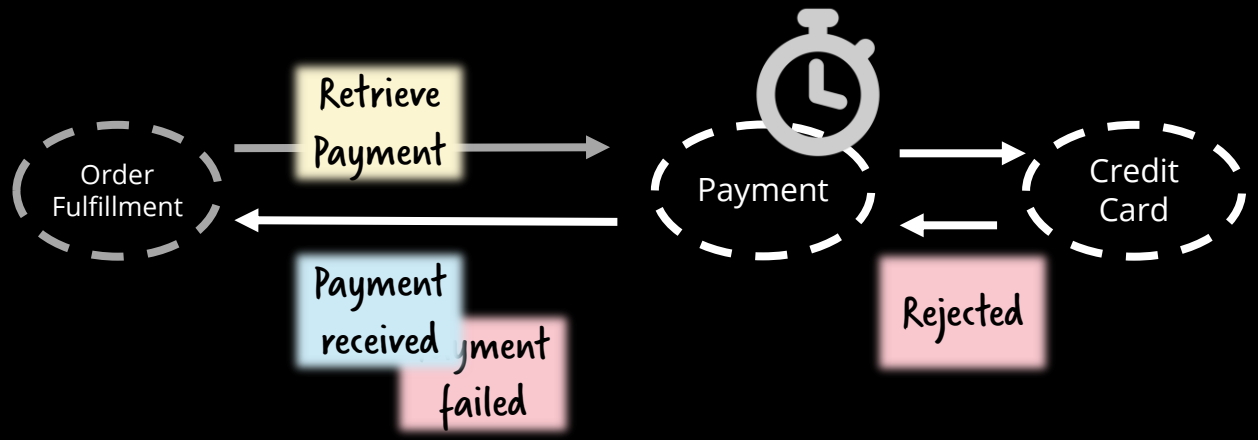
A few
smart god services
tell
anemic CRUD services
what to do

Who is responsible to deal with problems?

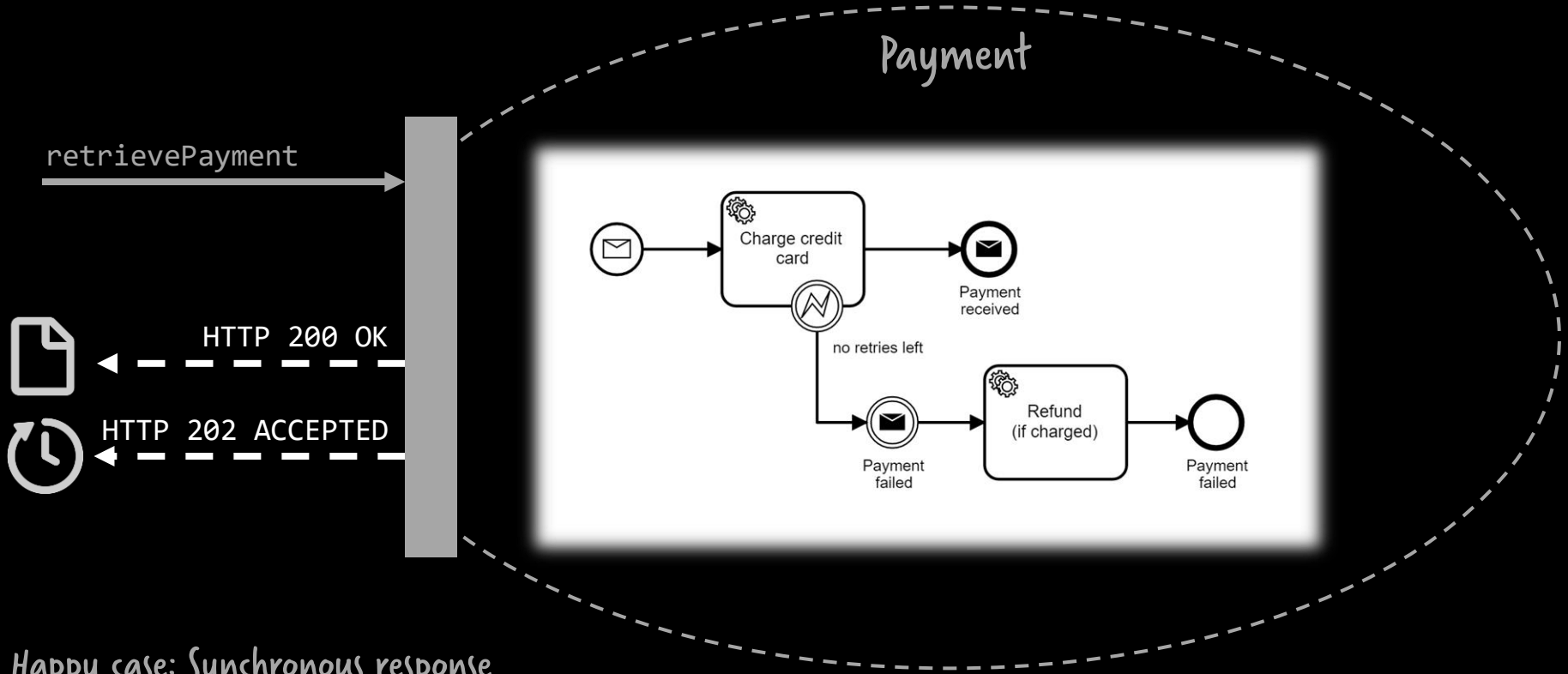


If the credit card was rejected, the customer can provide new details

(Potentially) long running services

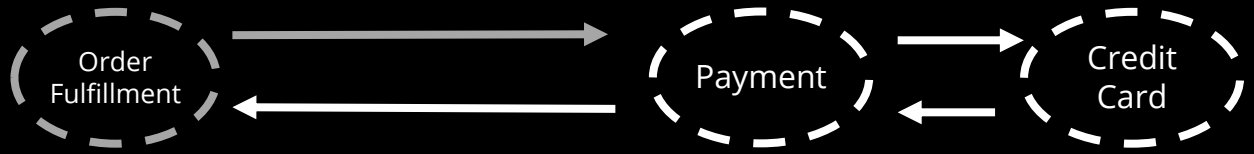


Embrace asynchronicity

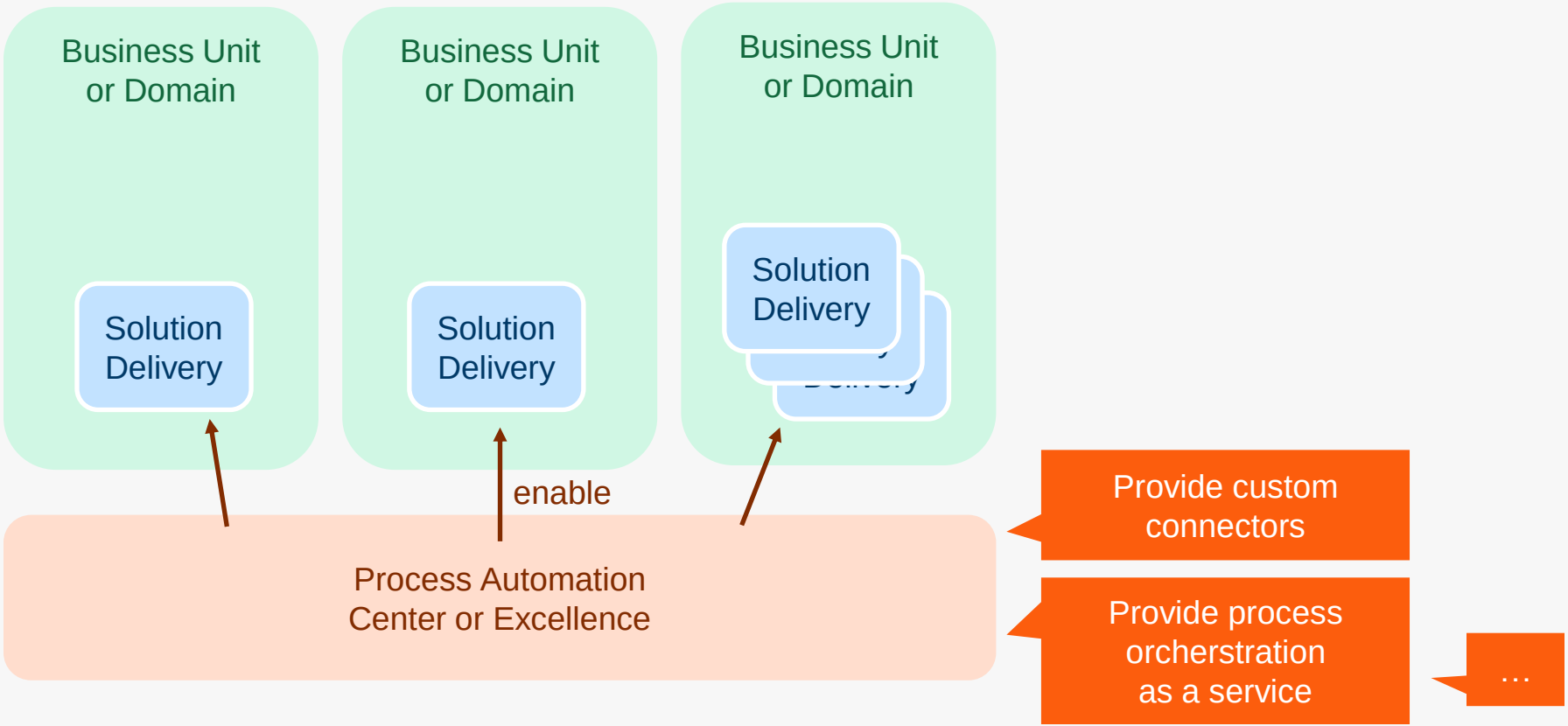


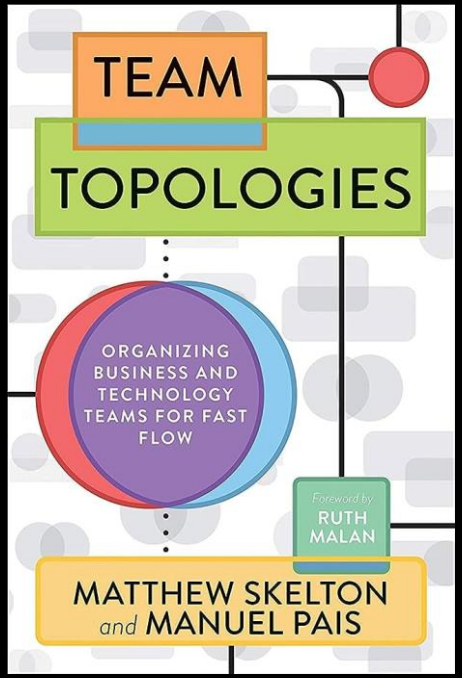
Happy case: Synchronous response
otherwise: asynchronous

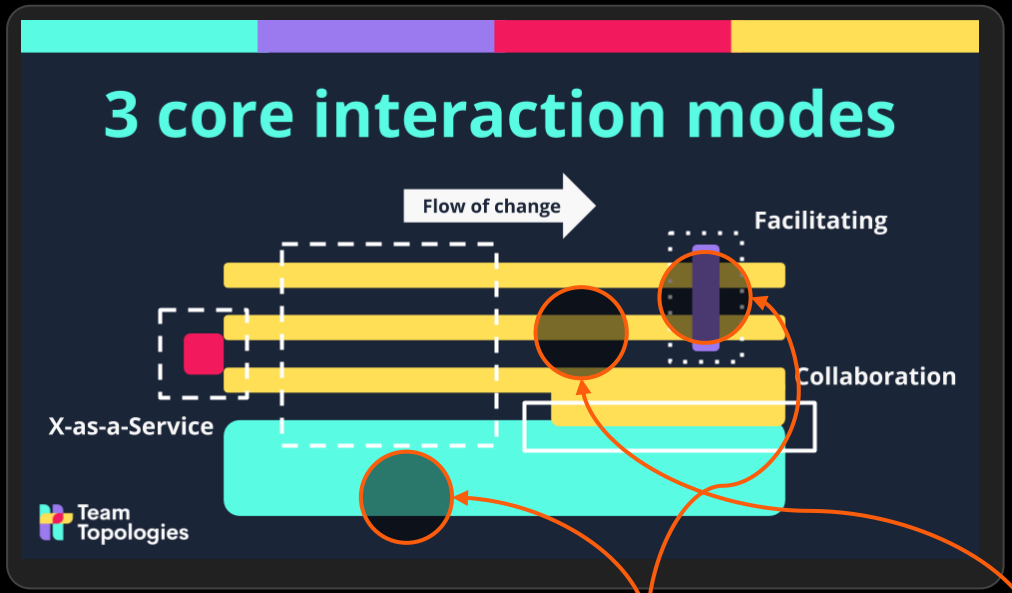
Designing good service boundaries



Scaling adoption

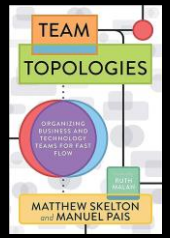



TEAM
TOPOLOGIES
ORGANIZING BUSINESS AND TECHNOLOGY TEAMS FOR FAST FLOW
Foreword by
RUTH MALAN
MATTHEW SKELTON
and MANUEL PAIS



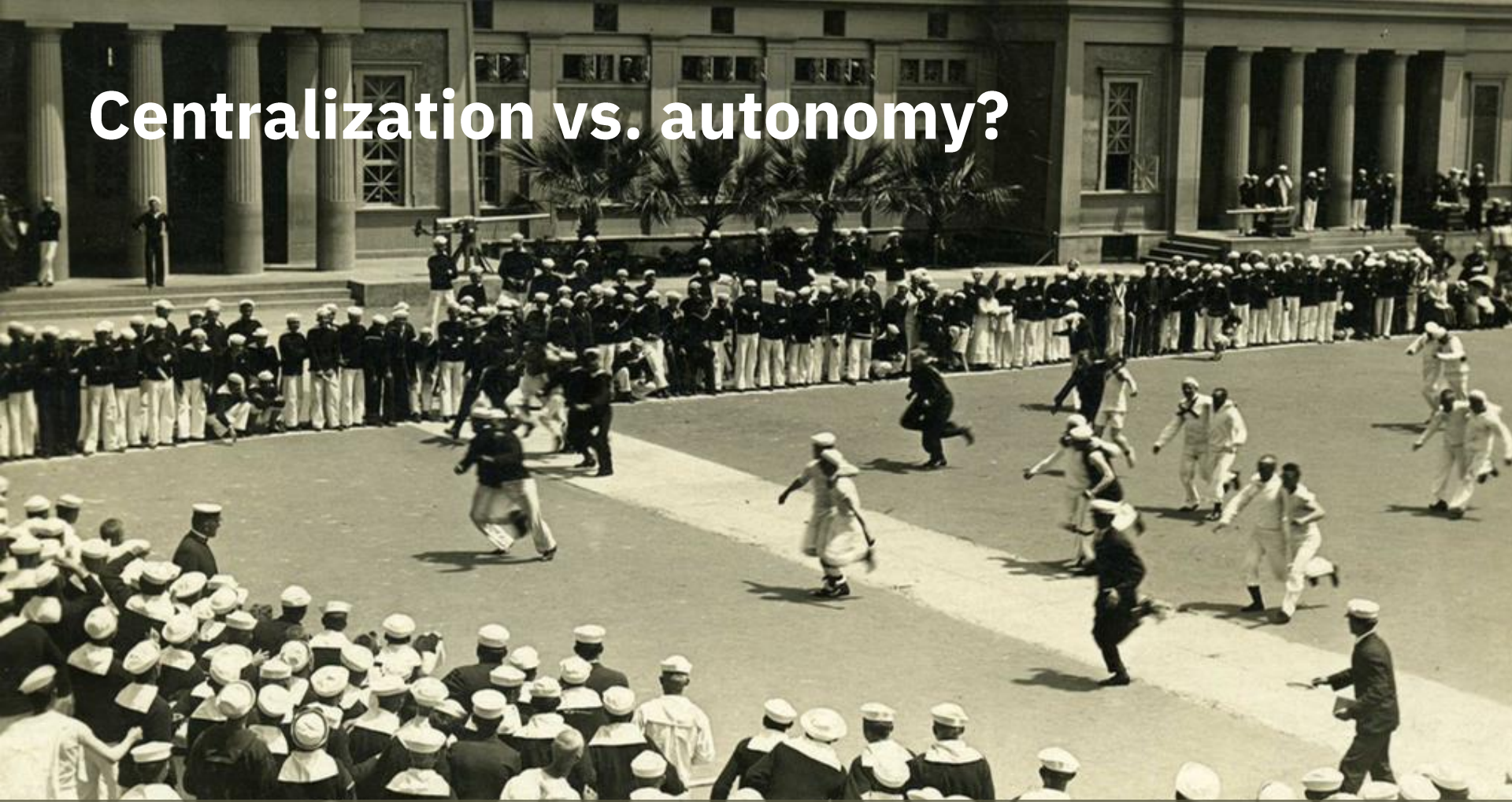
Center of Excellence

Domain



- Stream-aligned team
- Enabling team
- Complicated Subsystem team
- Platform team

Centralization vs. autonomy?



Spotify R&D

Golden Paths

“We found that
rumour-driven development
simply wasn’t scalable”

https://engineering.atspotify.com/2020/08/how-we-use-golden-paths-to-solve-fragmentation-in-our-software-ecosystem/

@berndruecker

The Speed Paradox

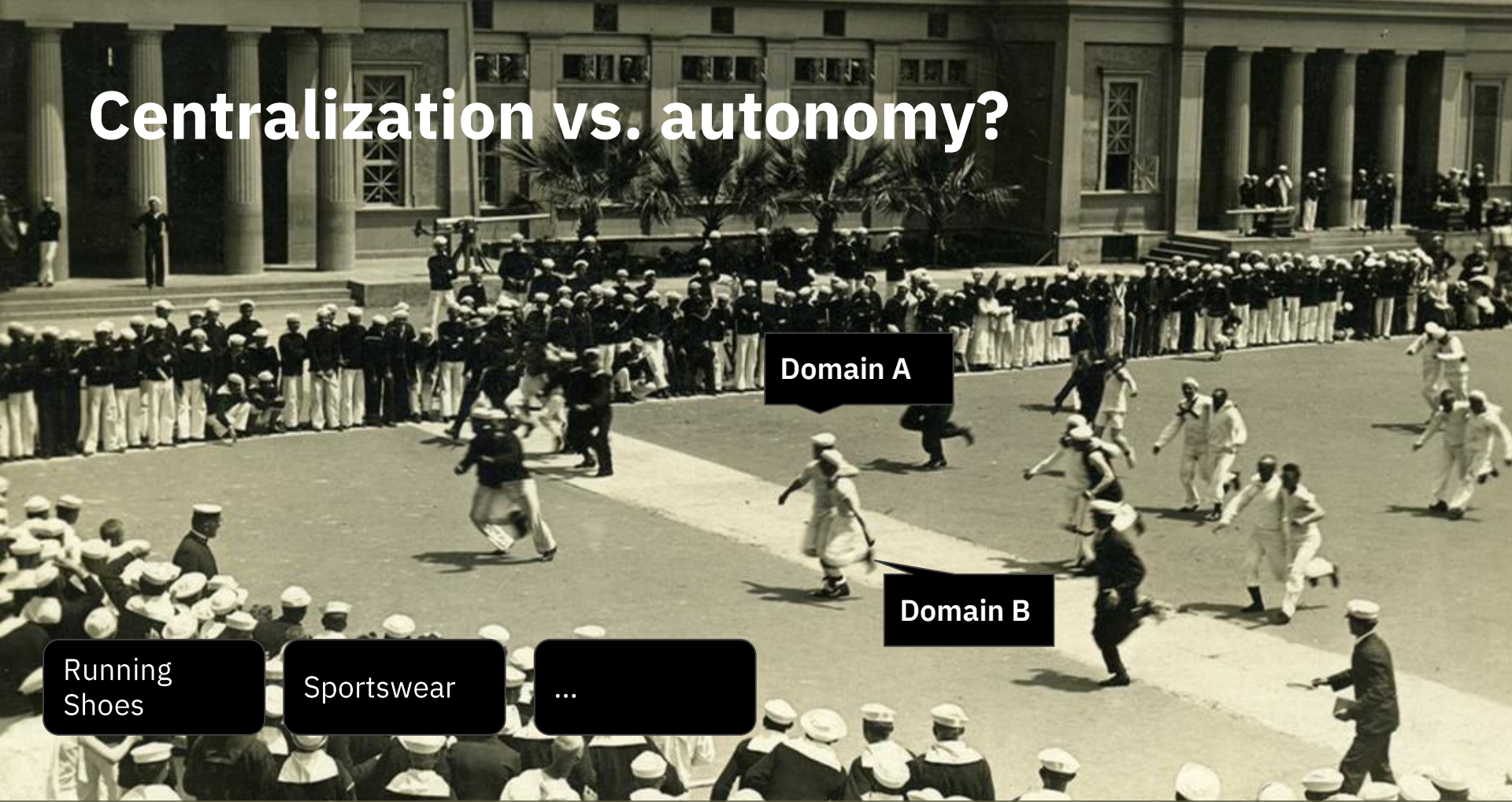
At Spotify, we've always believed in the speed and ingenuity that comes from having autonomous development teams. But as we learned firsthand, the faster you grow, the more fragmented and complex your software ecosystem becomes. And then everything slows down again.

The Standards Paradox

By centralizing services and standardizing your tooling, Backstage streamlines your development environment from end to end. Instead of restricting autonomy, standardization frees your engineers from infrastructure complexity. So you can return to building and scaling, quickly and safely.

https://backstage.io/ (OSS, Made with ♡ at Spotify)

Centralization vs. autonomy?



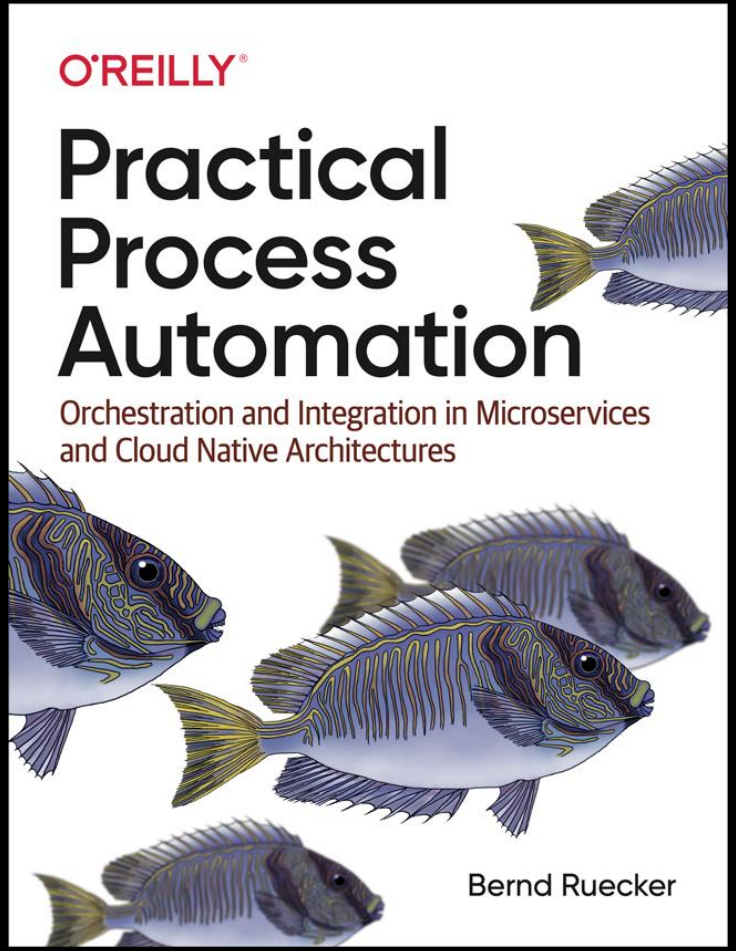
orchestration != central
Choreography != decoupled

orchestration = Command-driven
Choreography = Event-driven

You need to balance both!
It is about responsibility and the direction of coupling

You need long running capabilities to design good boundaries
Some central capability for providing infrastructure helps

Want To Know More?



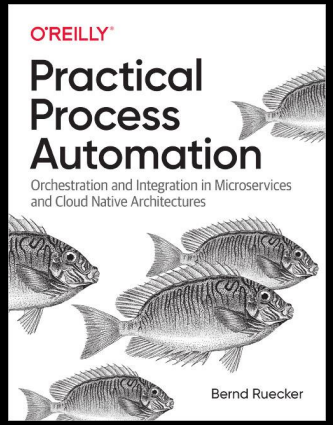

Thank you!
@berndruecker

Contact: bernd.ruecker@camunda.com
@berndruecker

Slides: https://berndruecker.io

Blog: https://blog.bernd-ruecker.com/

Code: https://github.com/berndruecker



https://ProcessAutomationBook.com/