Balancing Choreography & orchestration

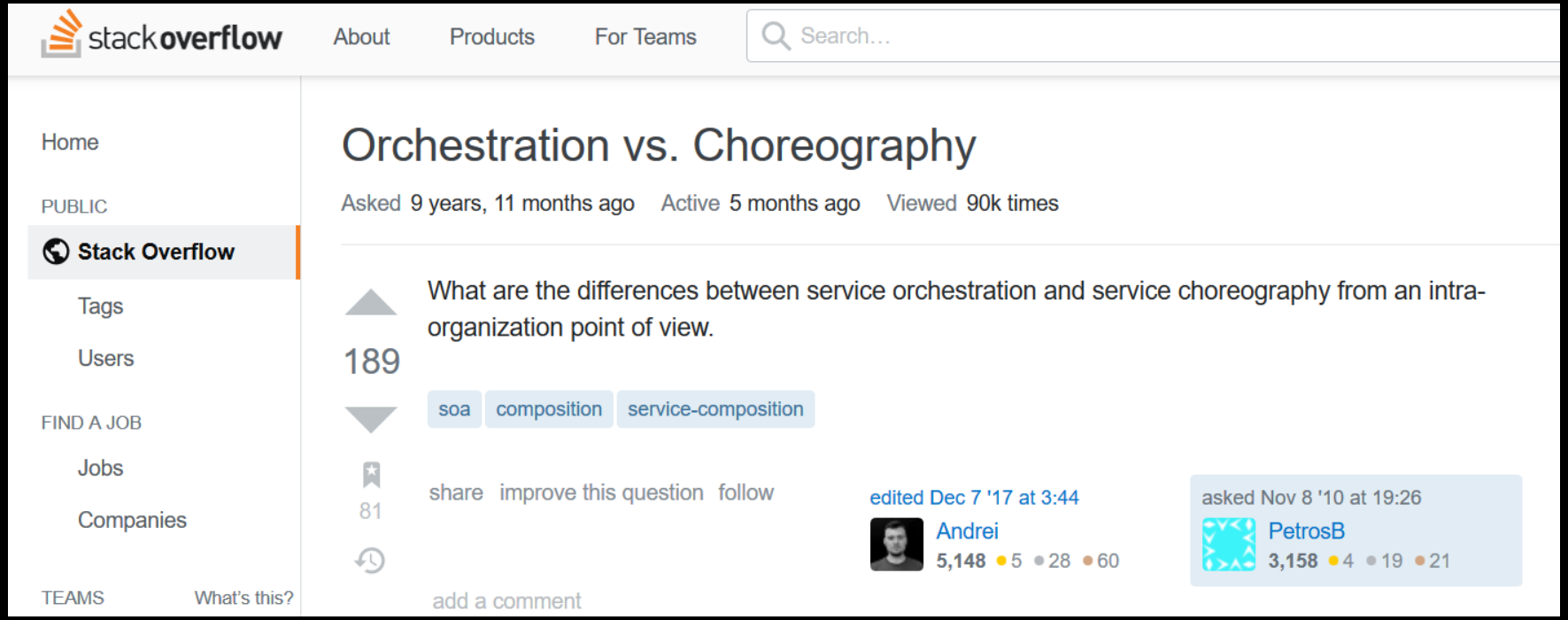
@berndruecker

Choreography is great!

What we wanted

vs. what we got

opinions...

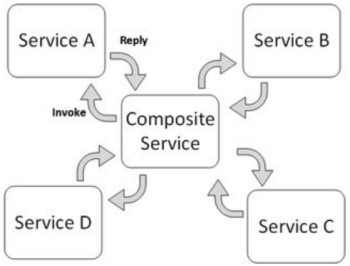


https://stackoverflow.com/questions/4127241/orchestration-vs-choreography

Service orchestration

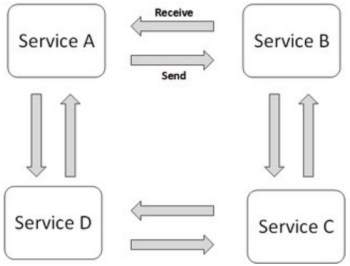
Service orchestration represents a single centralized executable business process (the orchestrator) that coordinates the interaction among different services. The orchestrator is responsible for invoking and combining the services.

The relationship between all the participating services are described by a single endpoint (i.e., the composite service). The orchestration includes the management of transactions between individual services. Orchestration employs a centralized approach for service composition.



Service Choreography

Service choreography is a global description of the participating services, which is defined by exchange of messages, rules of interaction and agreements between two or more endpoints. Choreography employs a decentralized approach for service composition.

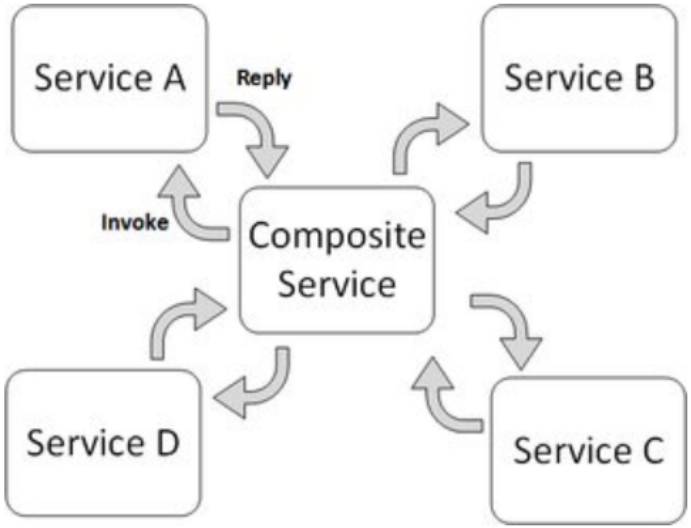


The choreography describes the interactions between multiple services, where as orchestration represents control from one party's perspective. This means that a **choreography** *differs* from an **orchestration** with respect to where the logic that controls the interactions between the services involved should reside.

Service orchestration

Service orchestration represents a single centralized executable business process (the orchestrator) that coordinates the interaction among different services. The orchestrator is responsible for invoking and combining the services.

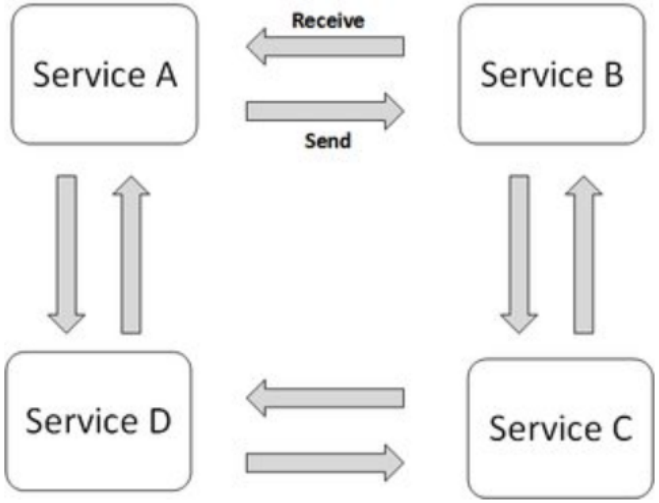
The relationship between all the participating services are described by a single endpoint (i.e., the composite service). The orchestration includes the management of transactions between individual services. Orchestration employs a centralized approach for service composition.



I don't agree!

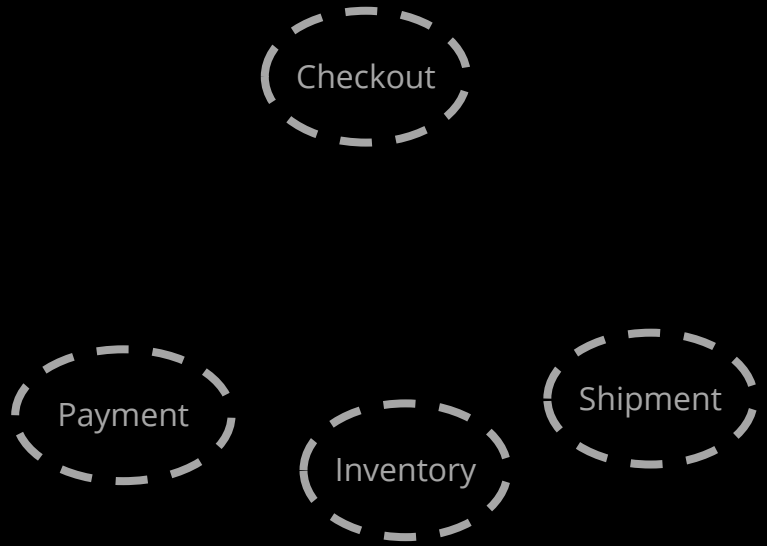
Service Choreography

Service choreography is a global description of the participating services, which is defined by exchange of messages, rules of interaction and agreements between two or more endpoints. Choreography employs a decentralized approach for service composition.



The choreography describes the interactions between multiple services, where as orchestration represents control from one party's perspective. This means that a **choreography** *differs* from an **orchestration** with respect to where the logic that controls the interactions between the services involved should reside.

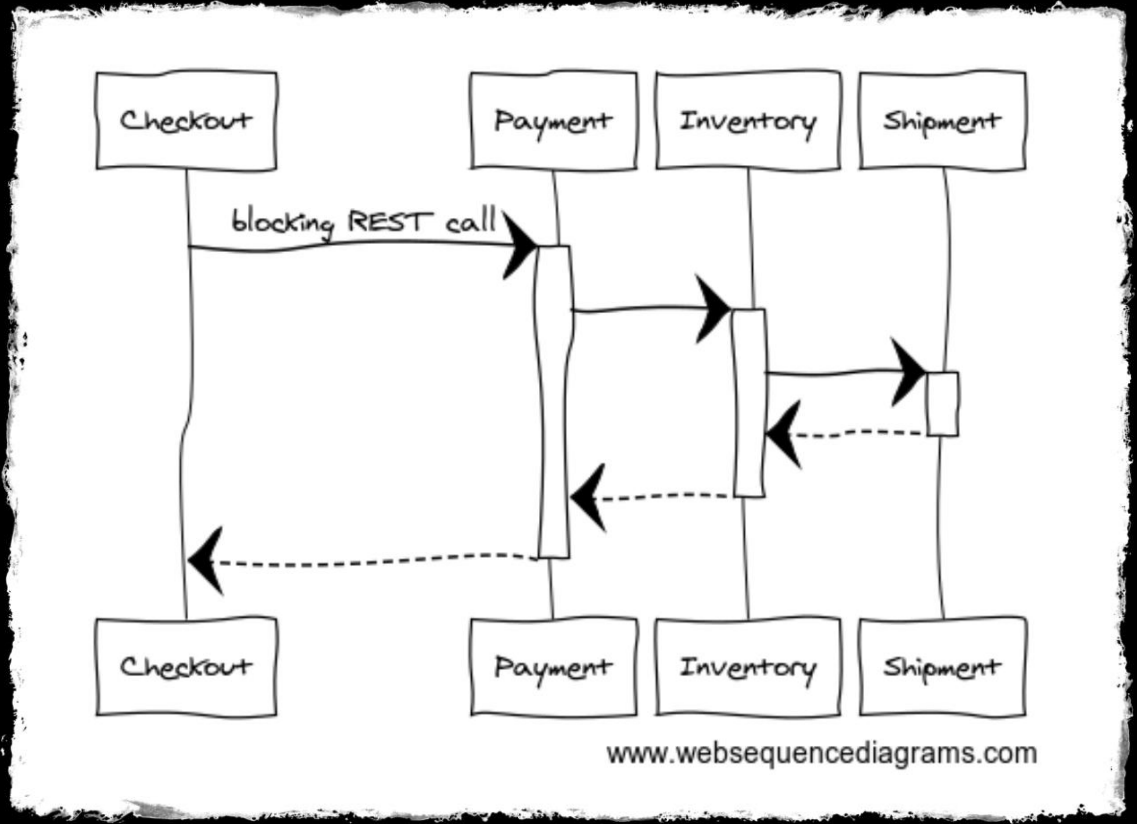
Example



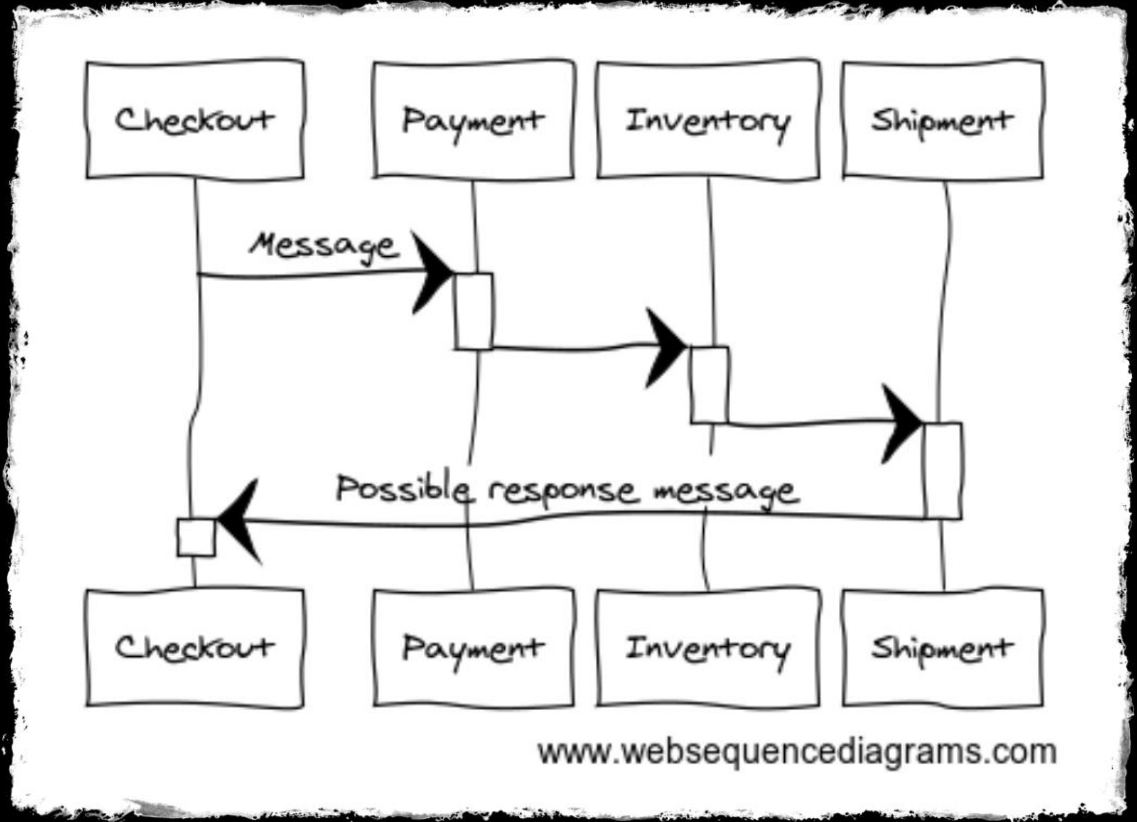
Synchronous call chains

„Synchronous call chains are evil!“

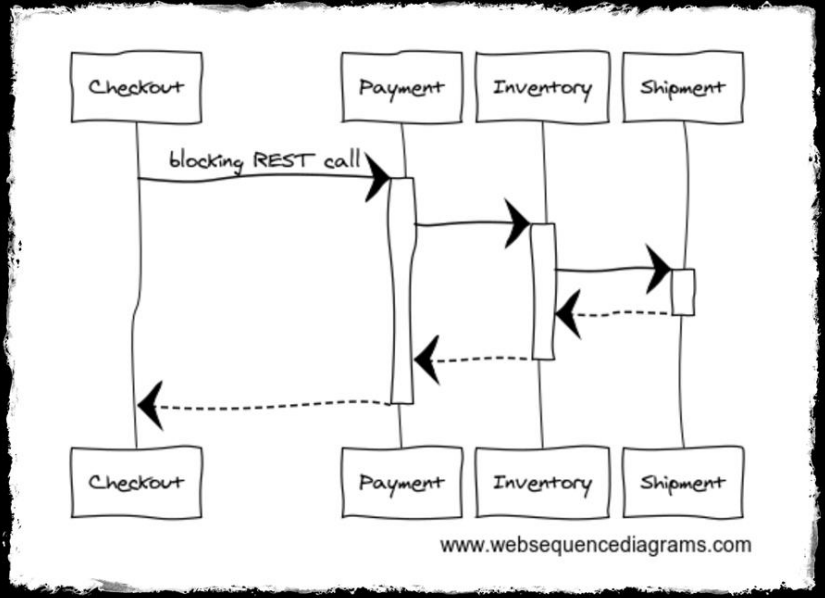
added latency, low availability, resource utilization, ...

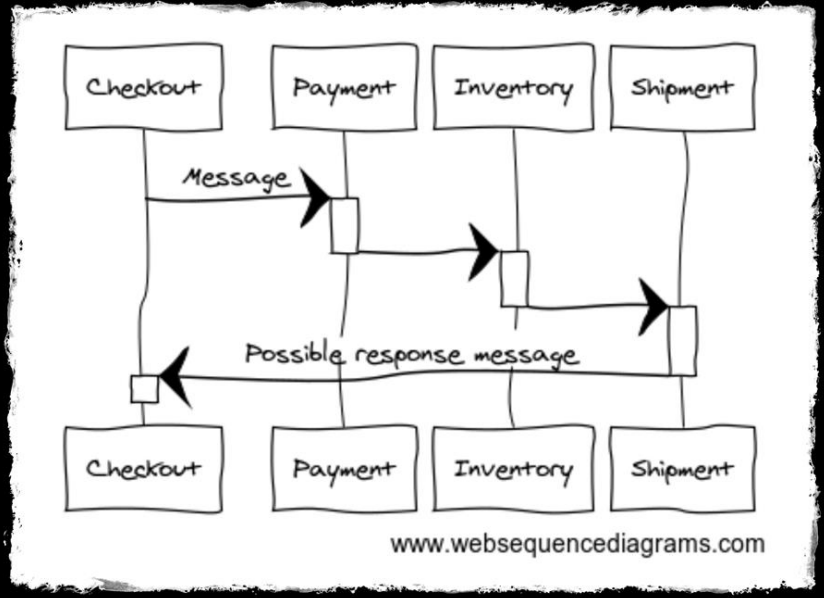


An asynchronous call chain



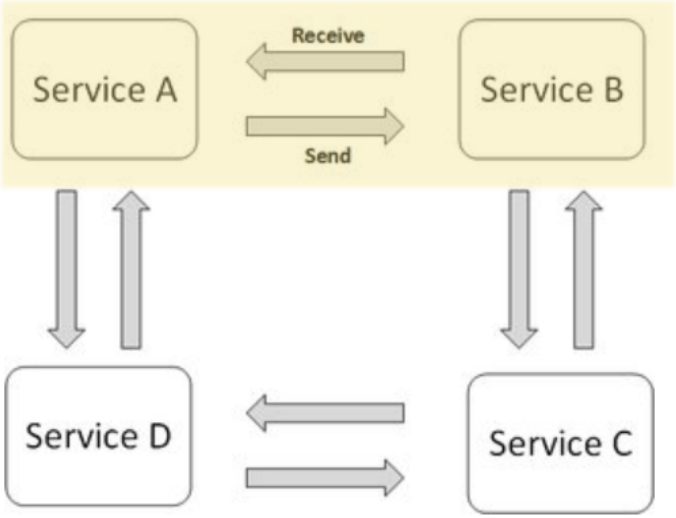
Choreography or orchestration?





Service Choreography

Service choreography is a global description of the participating services, which is defined by exchange of messages, rules of interaction and agreements between two or more endpoints. Choreography employs a decentralized approach for service composition.

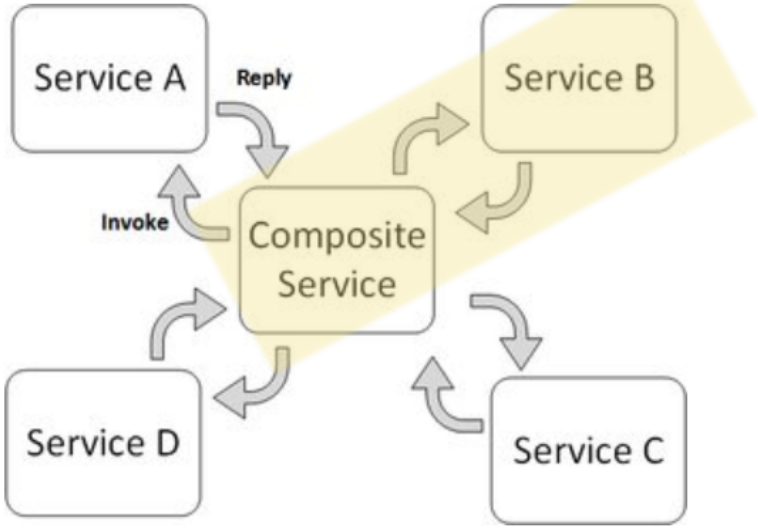


The choreography describes the interactions between multiple services, where as orchestration represents control from one party's perspective. This means that a **choreography** *differs* from an **orchestration** with respect to where the logic that controls the interactions between the services involved should reside.

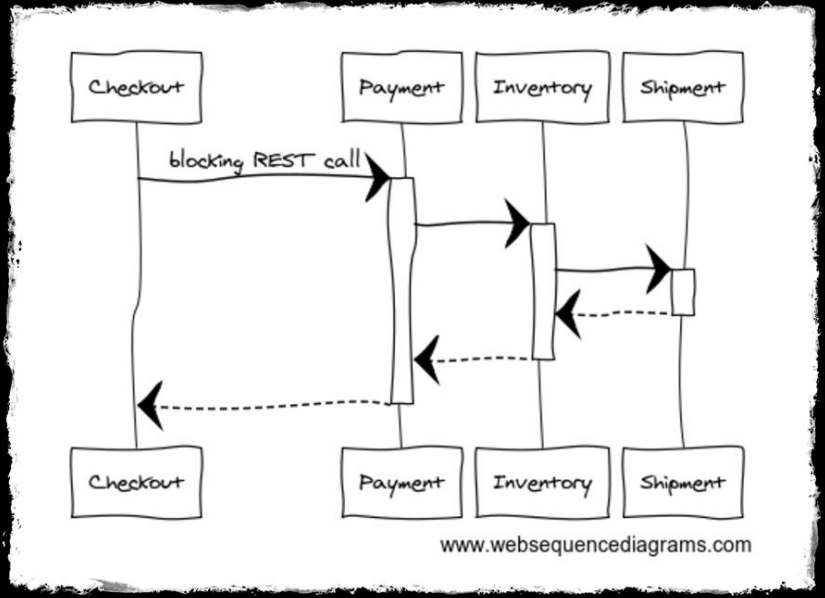
Service orchestration

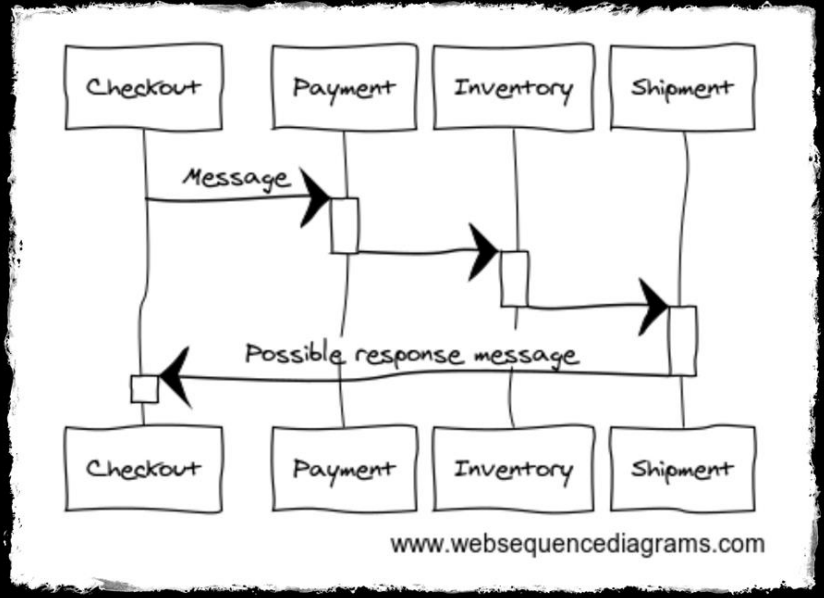
Service orchestration represents a single centralized executable business process (the orchestrator) that coordinates the interaction among different services. The orchestrator is responsible for invoking and combining the services.

The relationship between all the participating services are described by a single endpoint (i.e., the composite service). The orchestration includes the management of transactions between individual services. Orchestration employs a centralized approach for service composition.

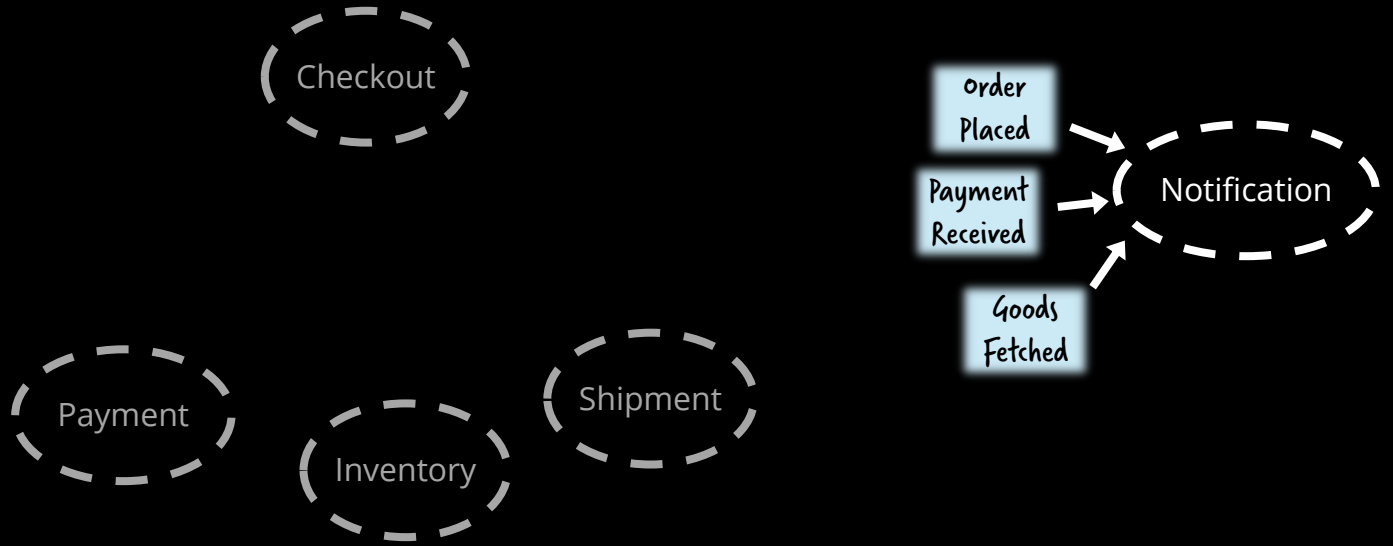


Choreography or orchestration?

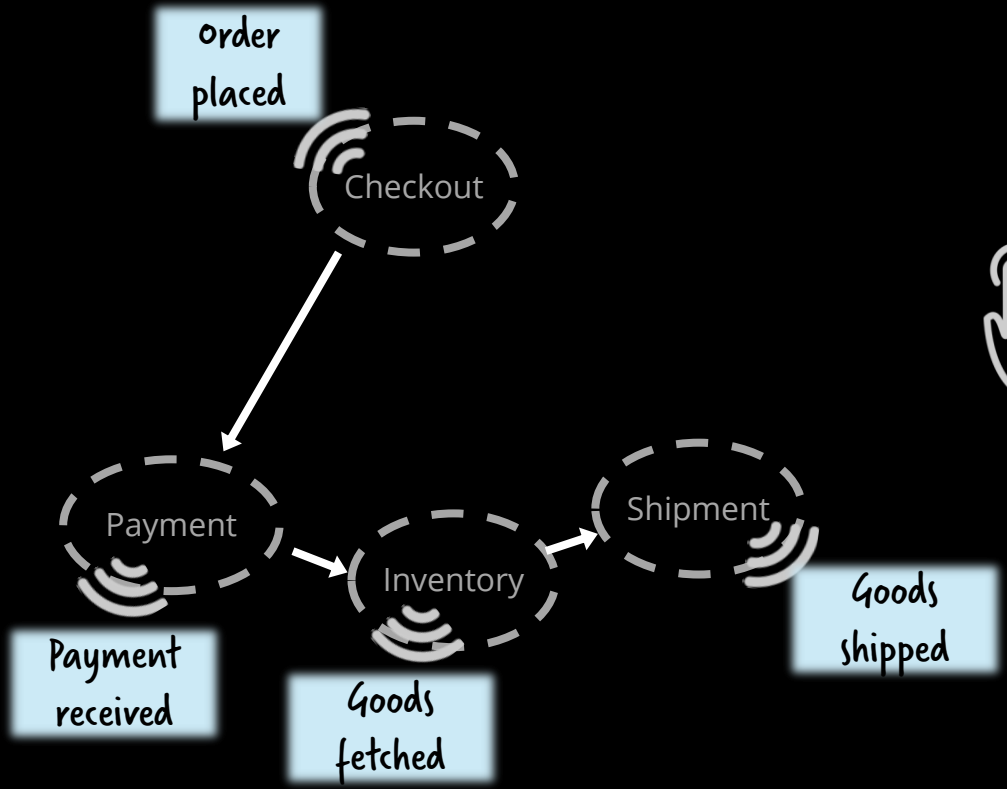


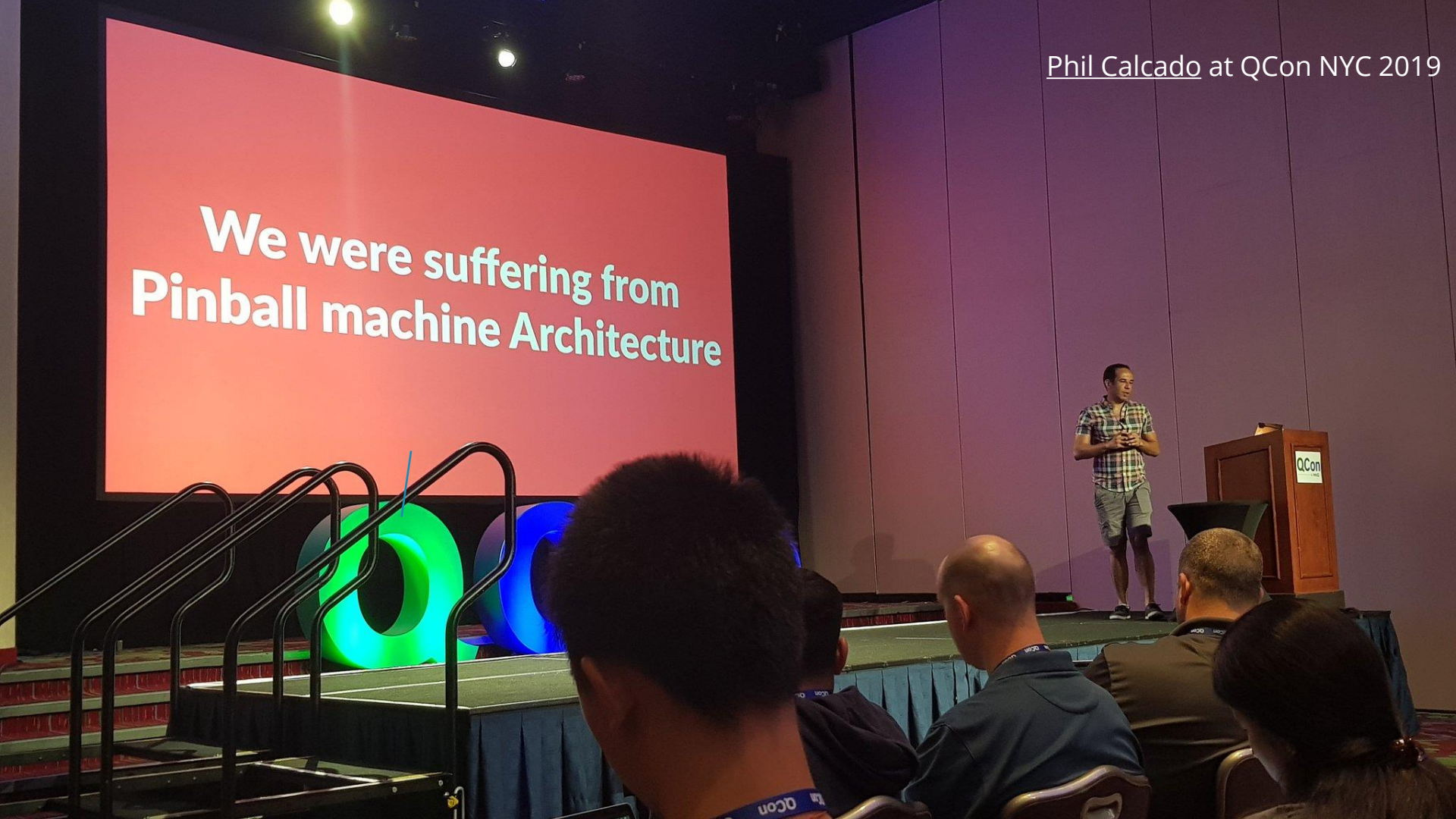


Event-driven



Peer-to-peer event chains



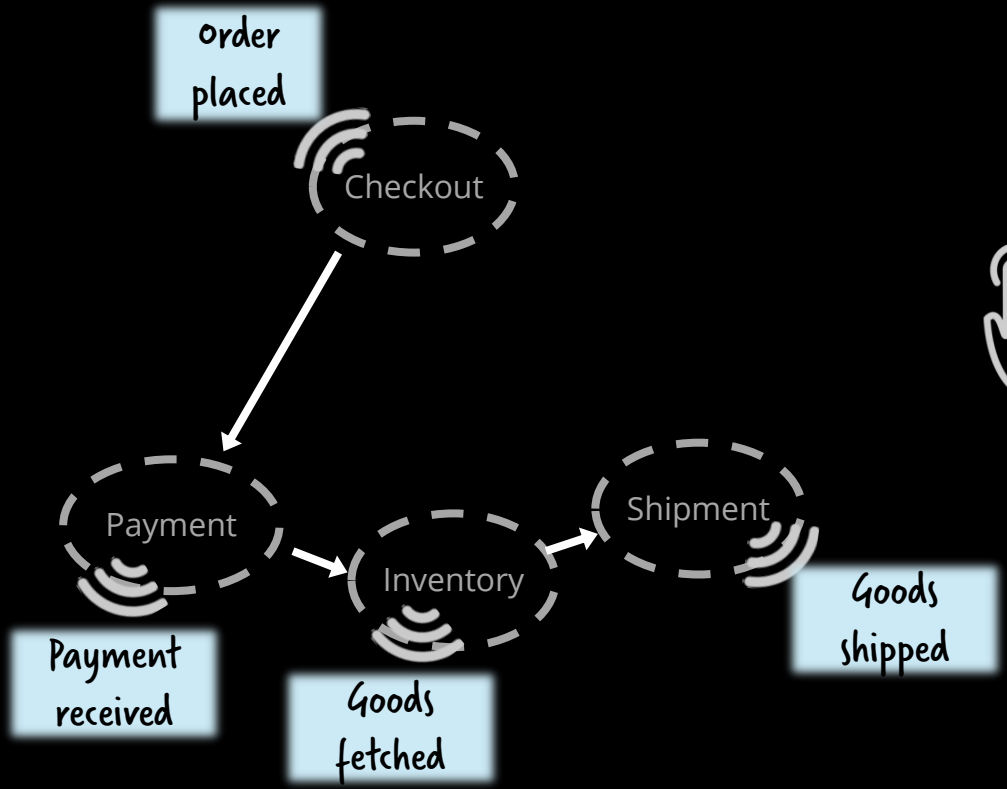


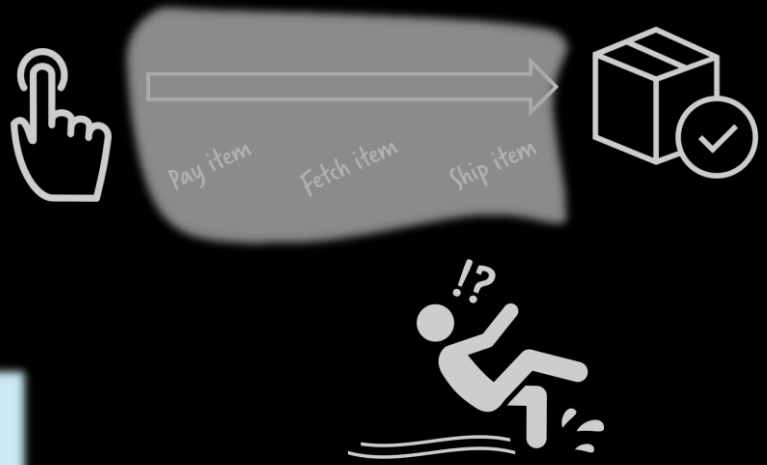
Phil Calcado at QCon NYC 2019

Pinball Machine Archite

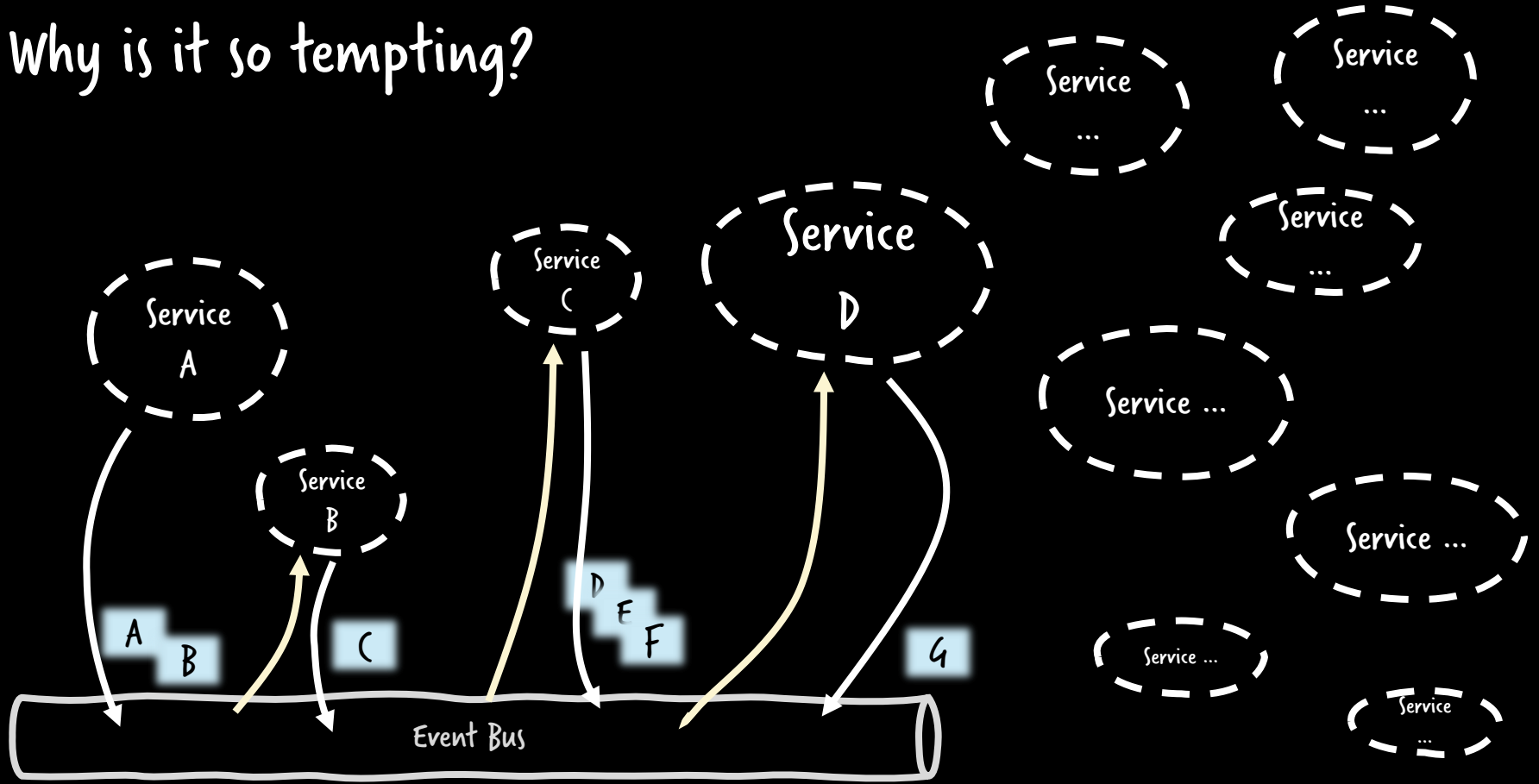
„What the hell just happened?“

Peer-to-peer event chains

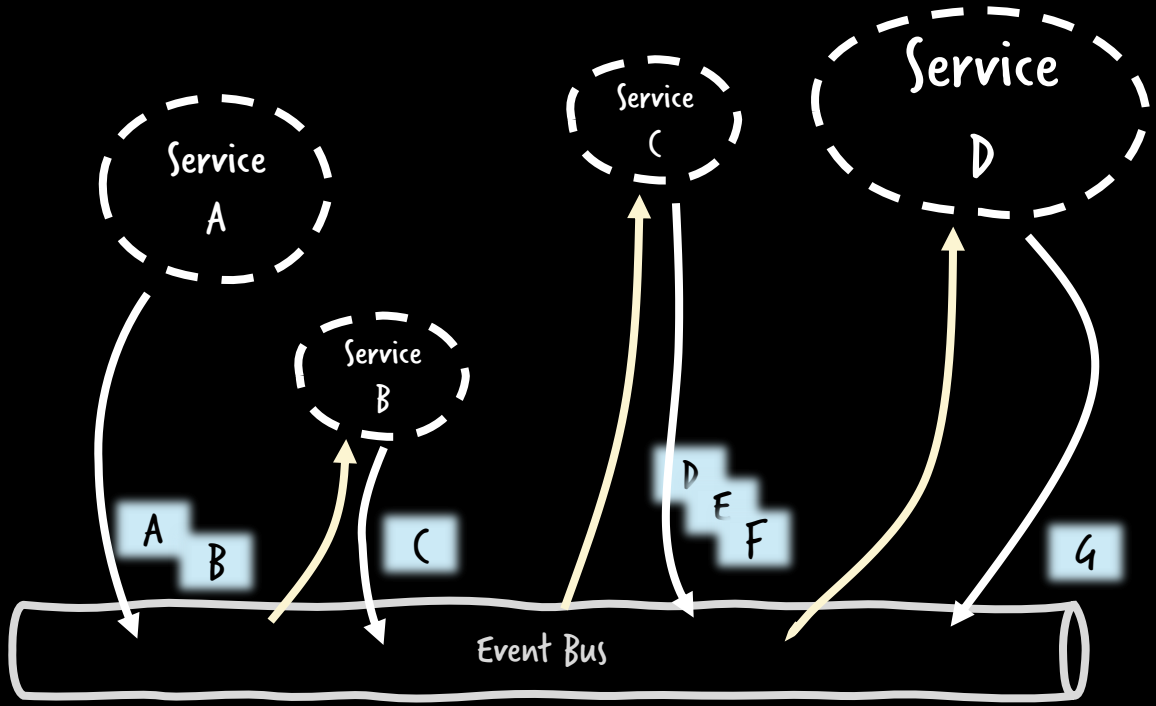




Why is it so tempting?



Why is it so tempting?



Adding is easy!

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

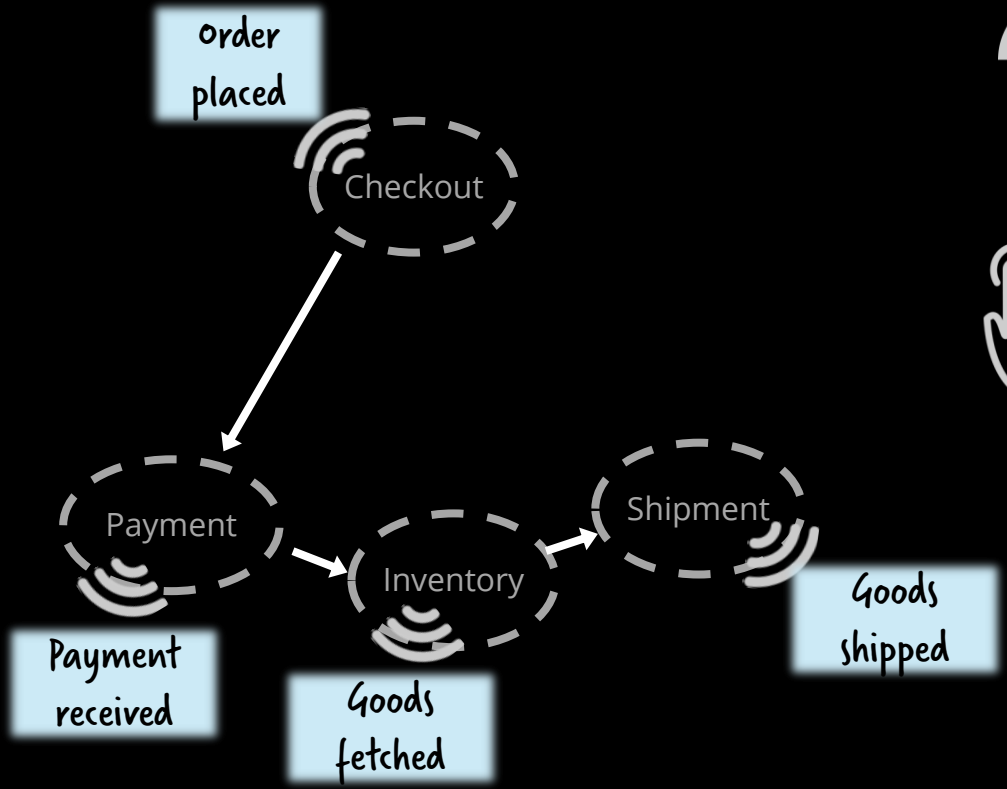
It yields in technical debt.

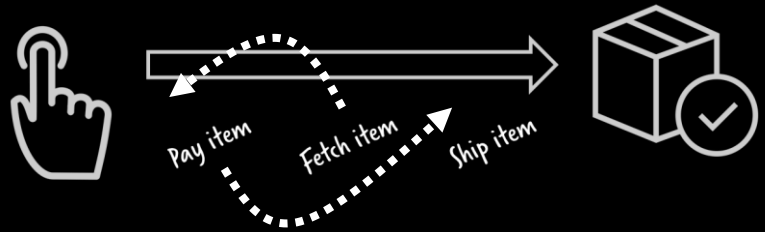
Service ...

Service ...

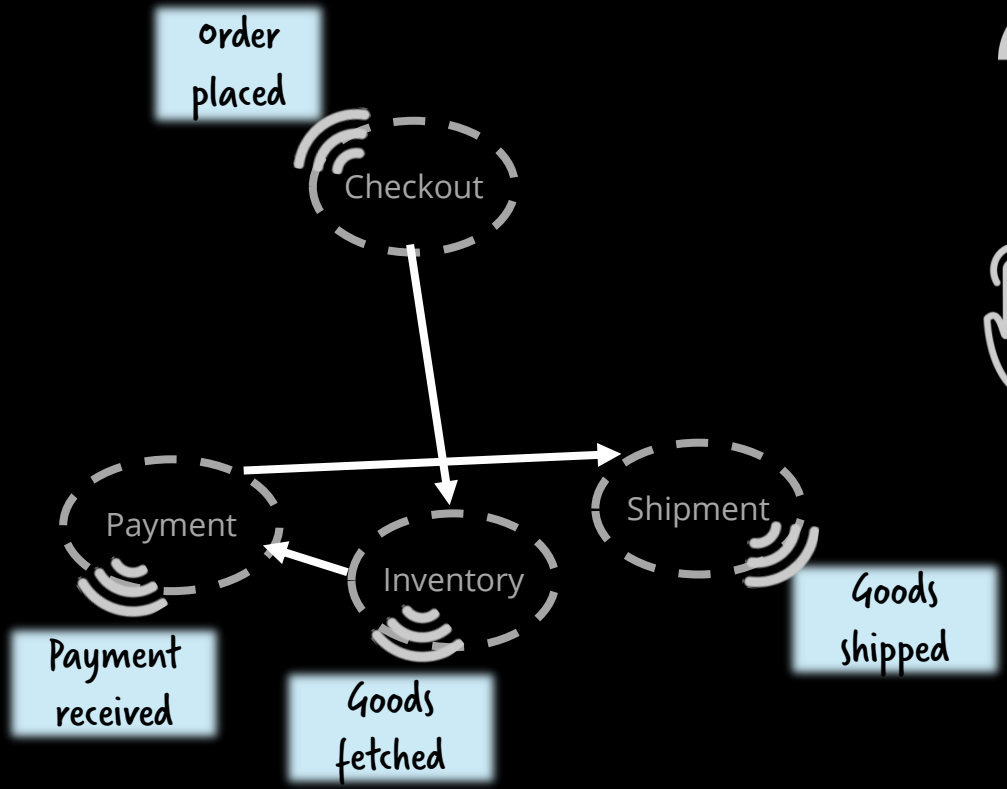
Service ...

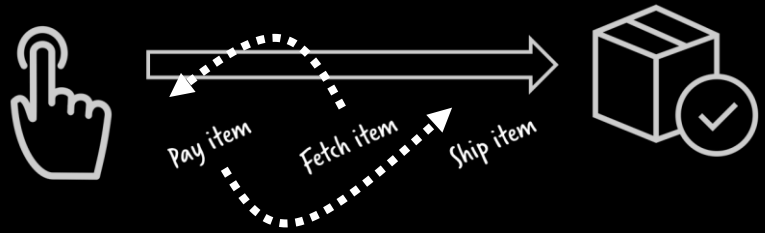
Peer-to-peer event chains



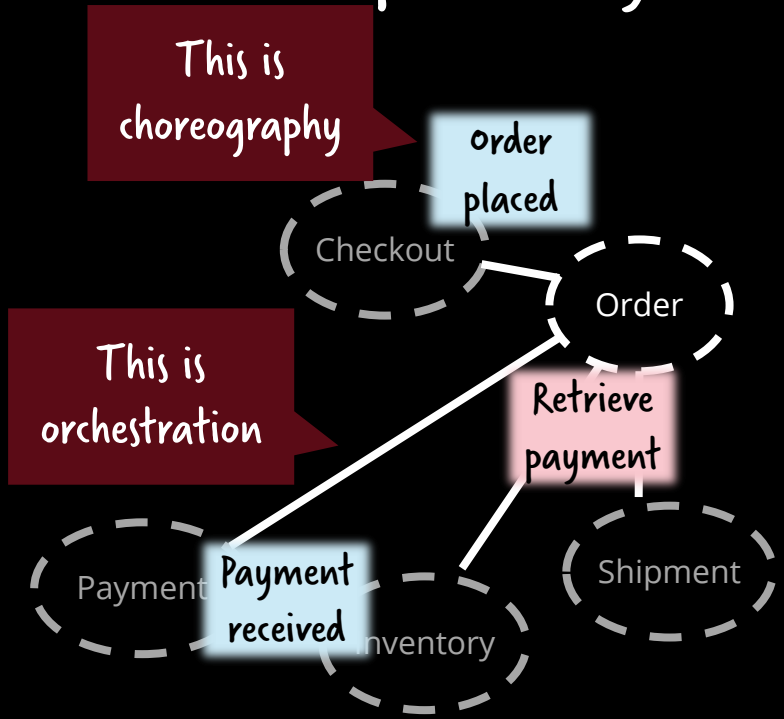


Peer-to-peer event chains





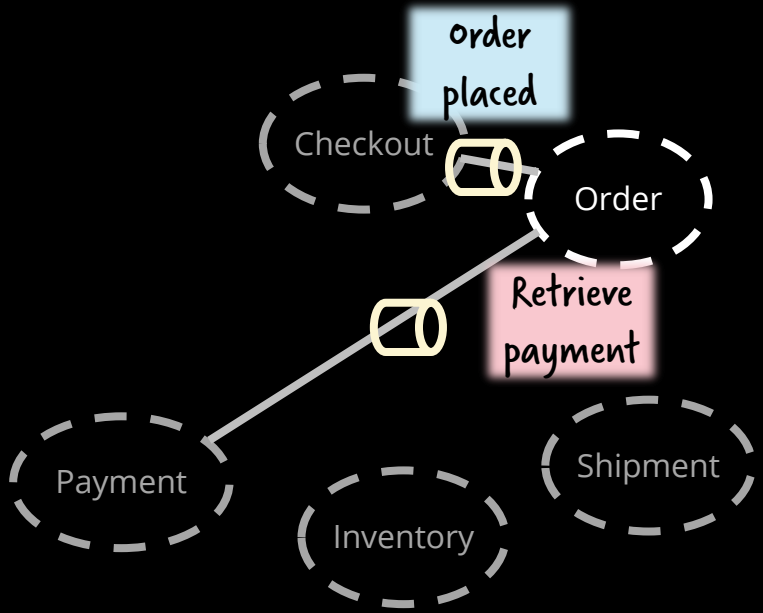
Decide about responsibility



My definition

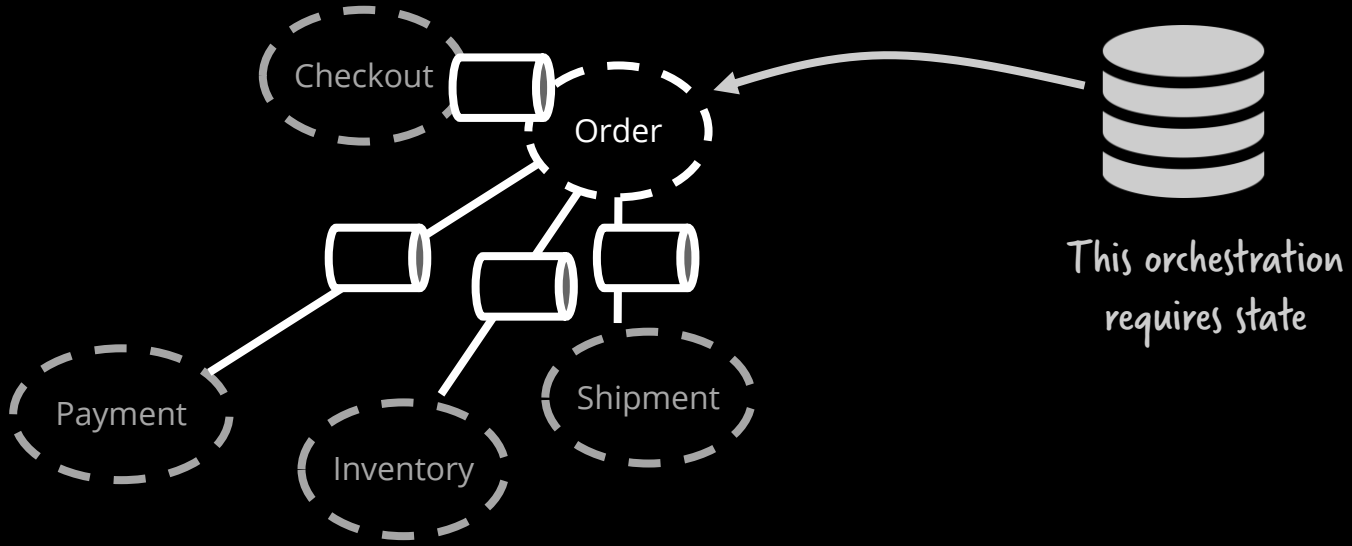
Orchestration	=	command-driven communication
Choreography	=	event-driven communication

Decide about responsibility



It can still be messaging!

Stateful orchestration

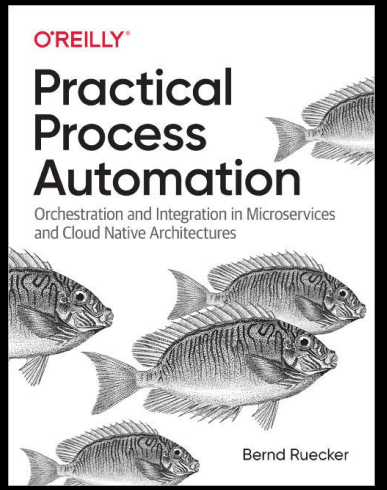


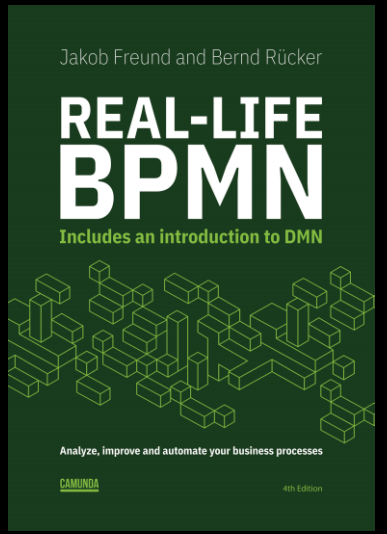
Warning:
Contains Opinion

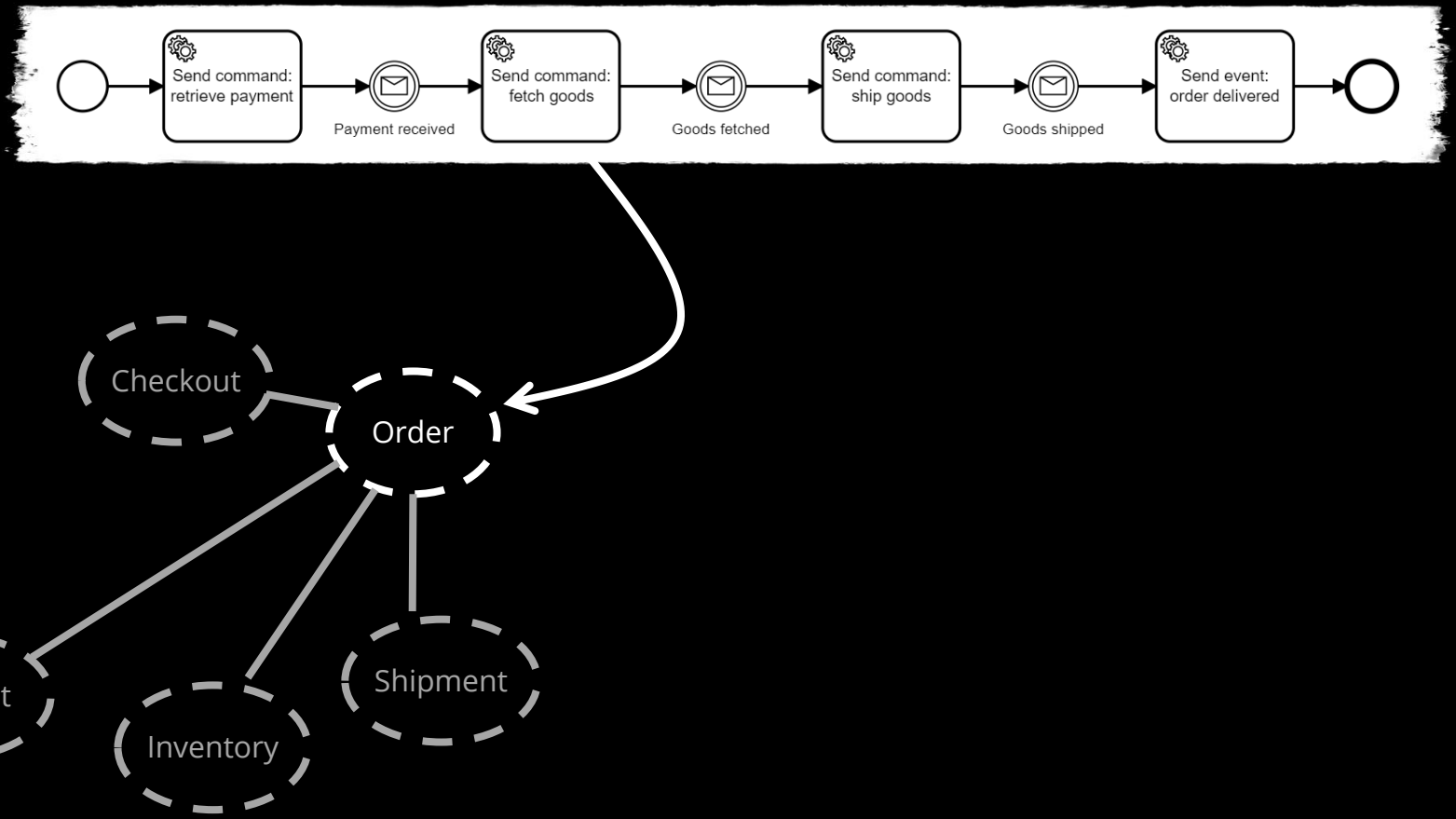
Bernd Ruecker

Co-founder and
Chief Technologist of
Camunda

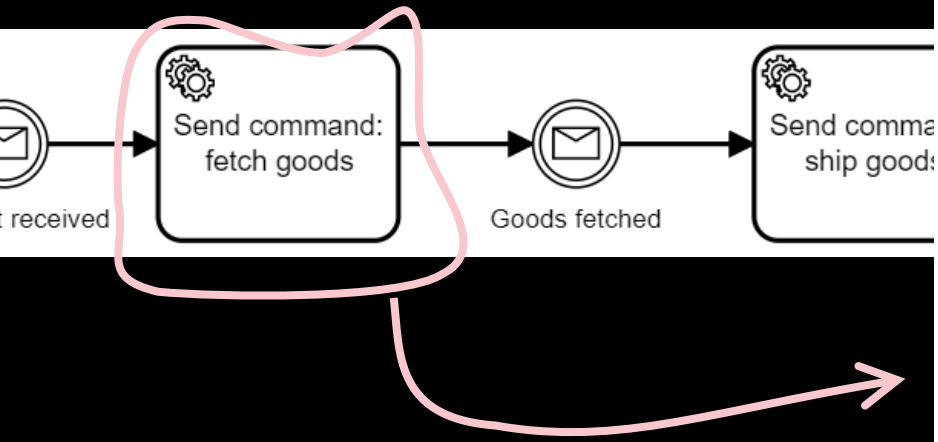
mail@berndruecker.io
@berndruecker
http://berndruecker.io/






Send command: retrieve payment
Payment received
Send command: fetch goods
Goods fetched
Send command: ship goods
Goods shipped
Send event: order delivered
Checkout
Order
Payment
Inventory
Shipment

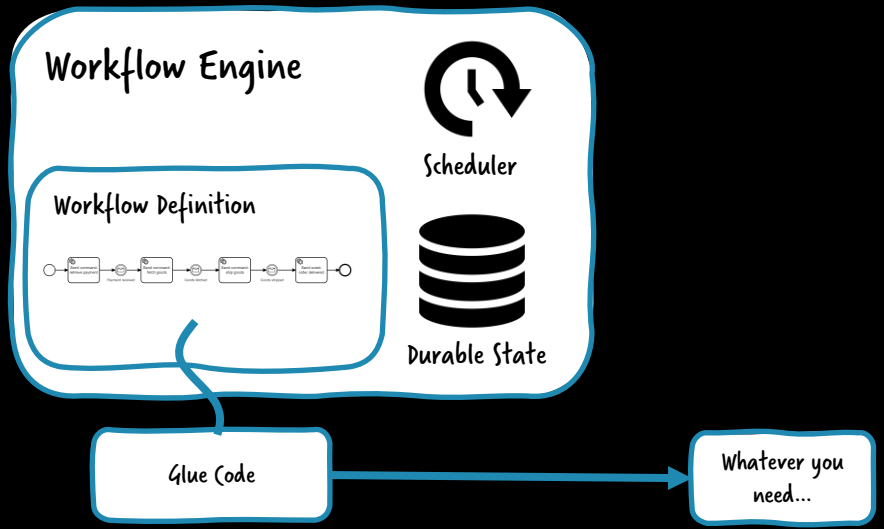
Glue code (e.g. Java)



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

}
```

Using a workflow engine



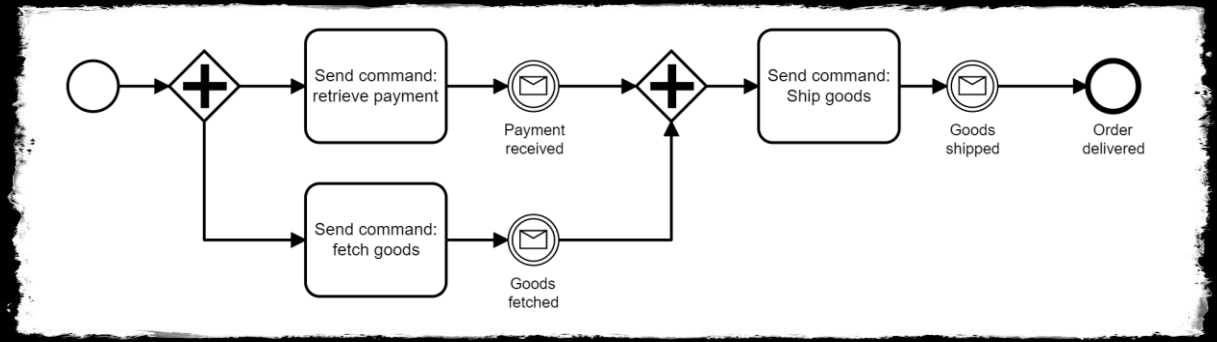
Workflow Engine:

Is stateful

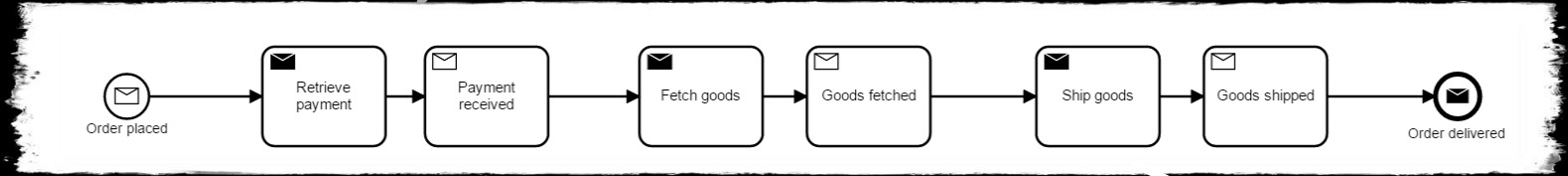
Can wait
Can retry
Can escalate
Can compensate

Provides visibility

Now it is easy to change the process flow



Processes are domain logic and live inside service boundaries



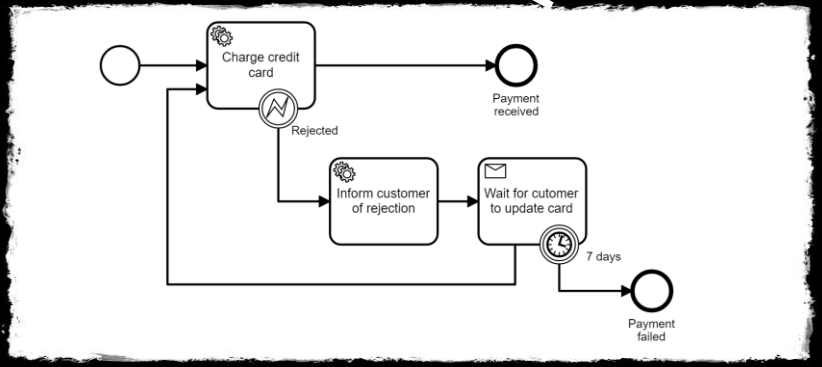
Checkout

Order

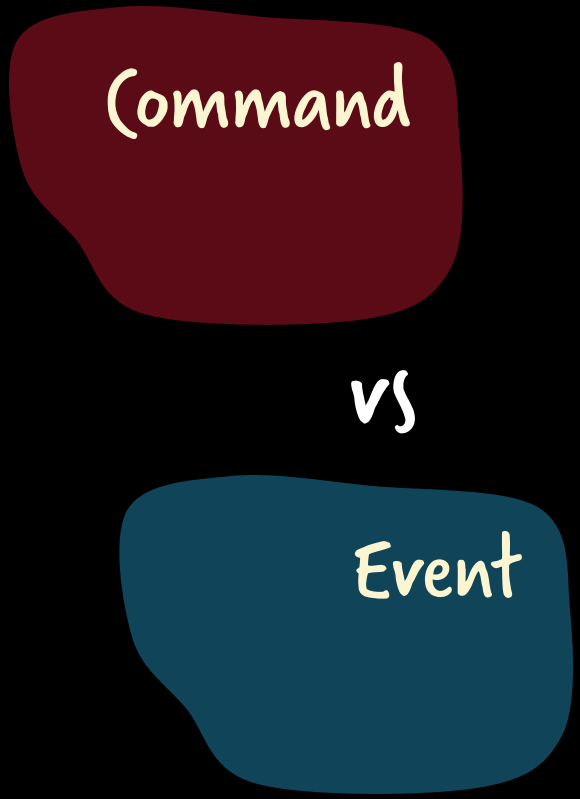
Payment

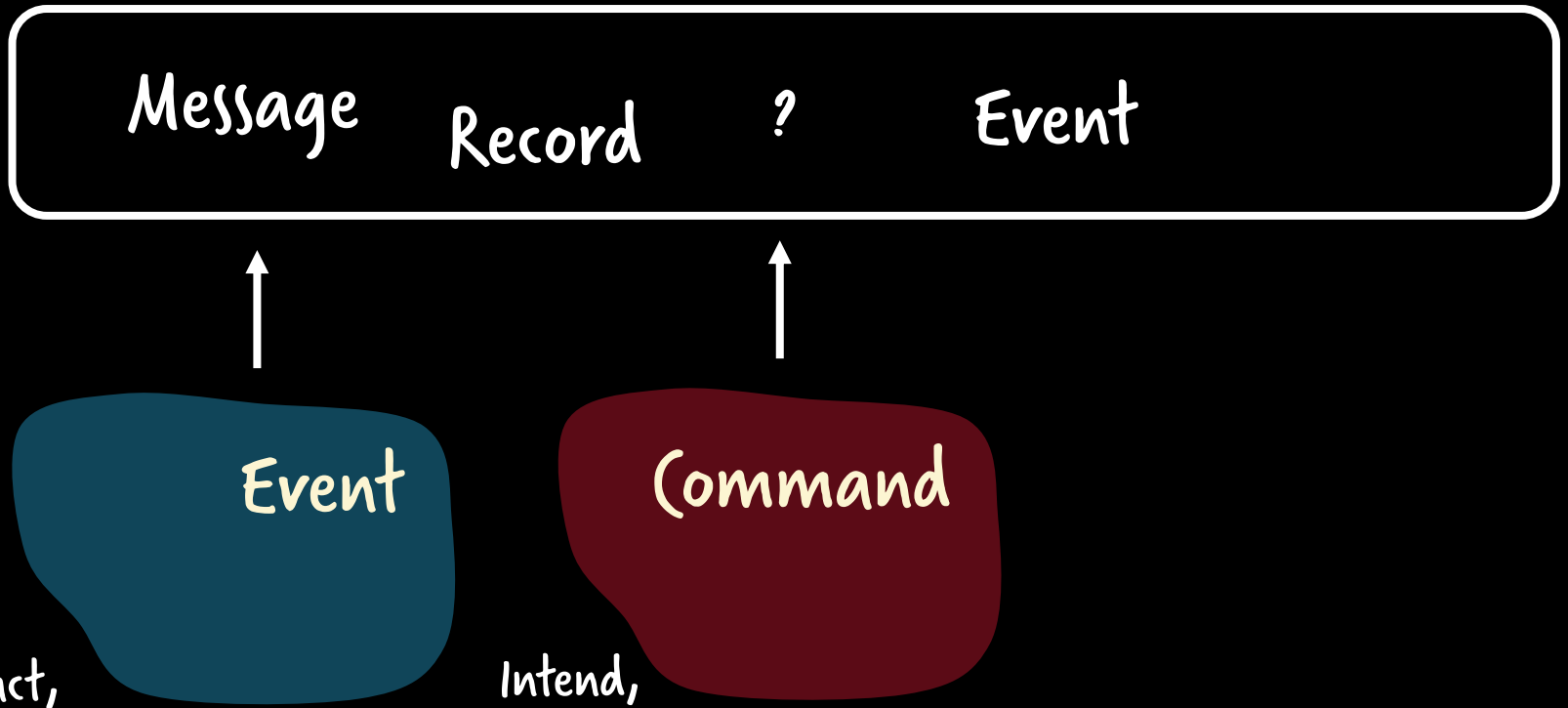
Inventory

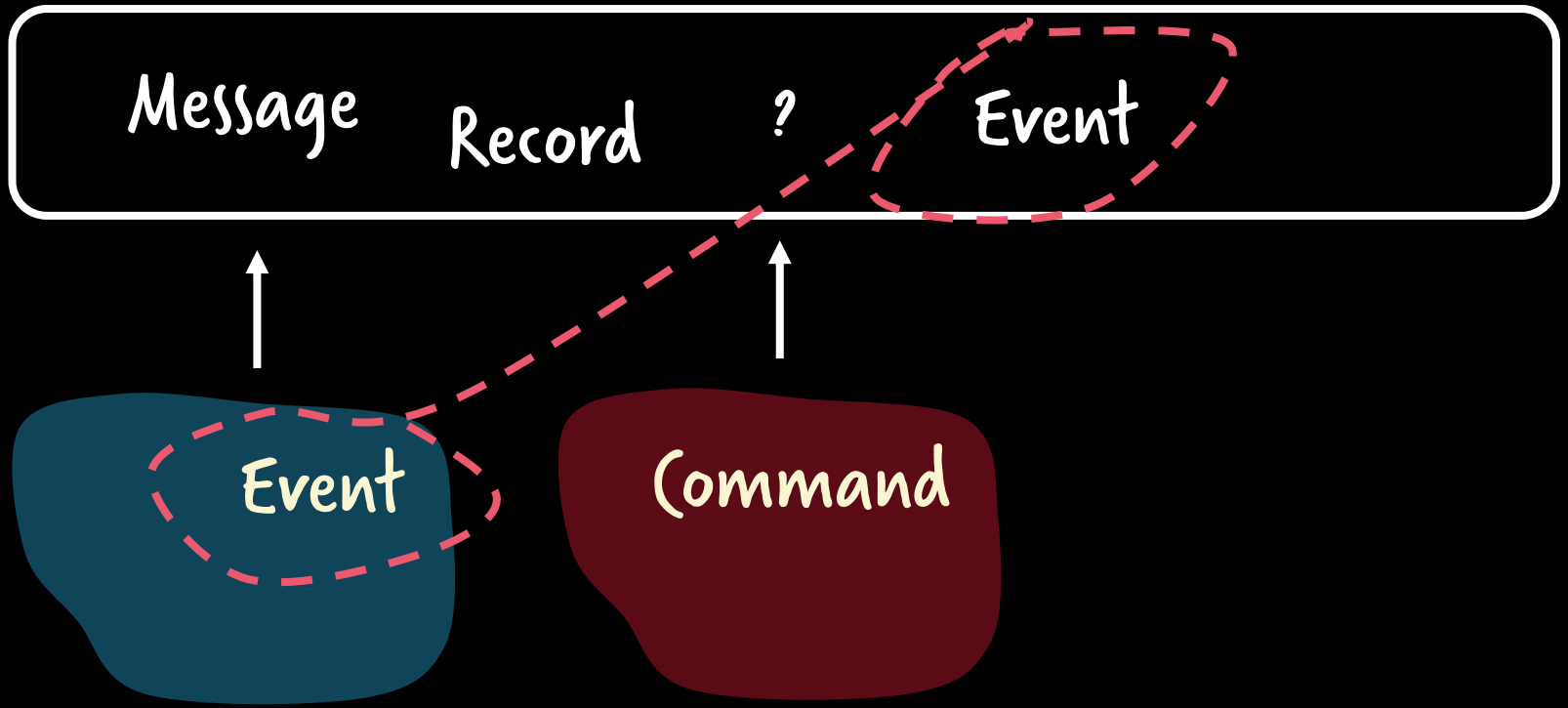
Shipment



Challenge: Command vs. Event




Message
Record
?
Event
Event
Command
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

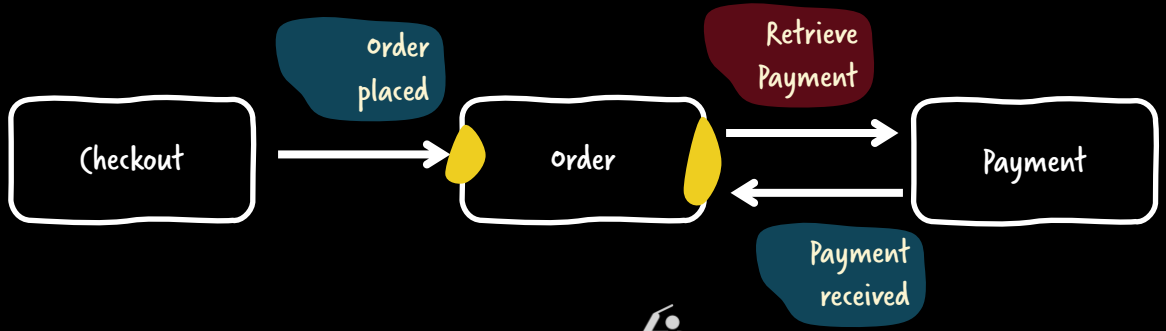
Wording of recipient

Send Message

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

Wording of Sender

Direction of dependency

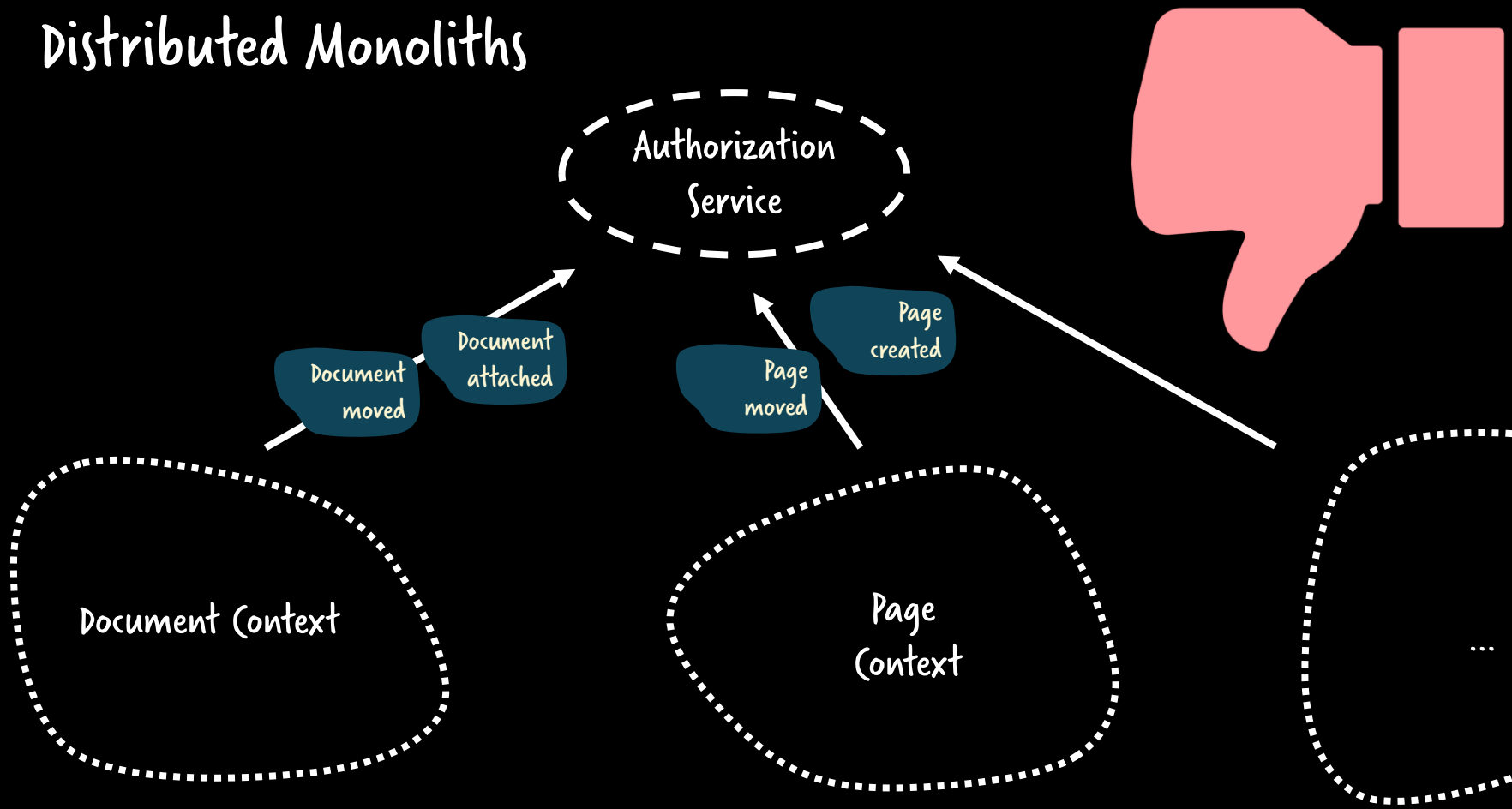


Event-driven:
Decision to couple is on the **receiving** side

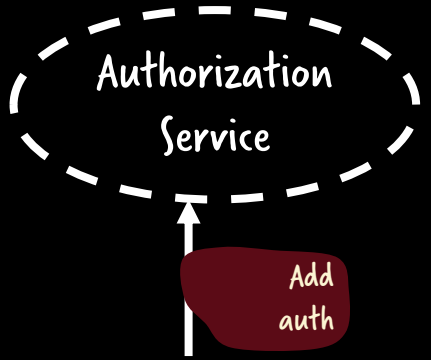
Command-driven
Decision to couple is on the **sending** side

Direction of dependency

Distributed Monoliths



Define stable contract/API instead



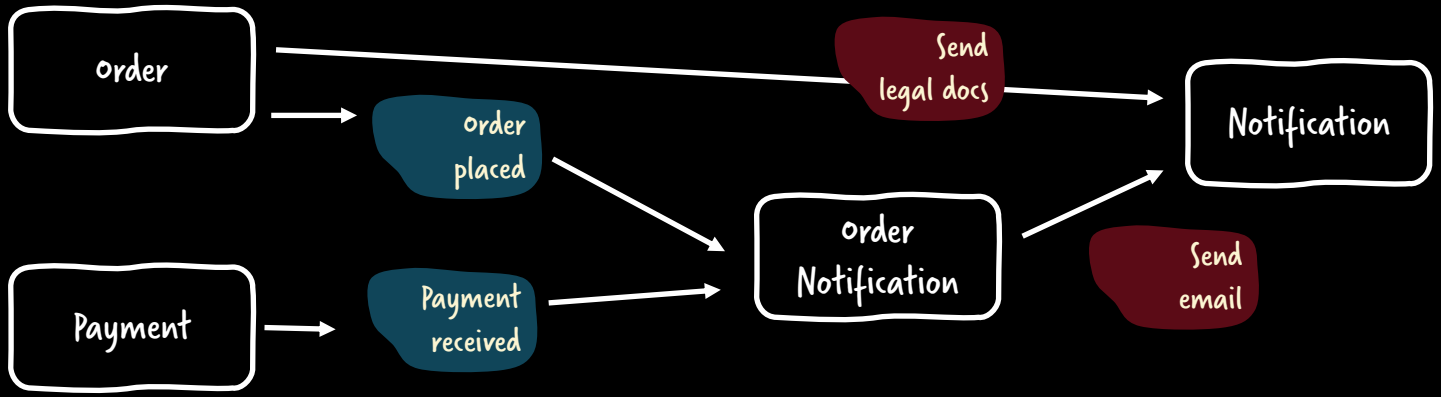
Document Context

Page Context

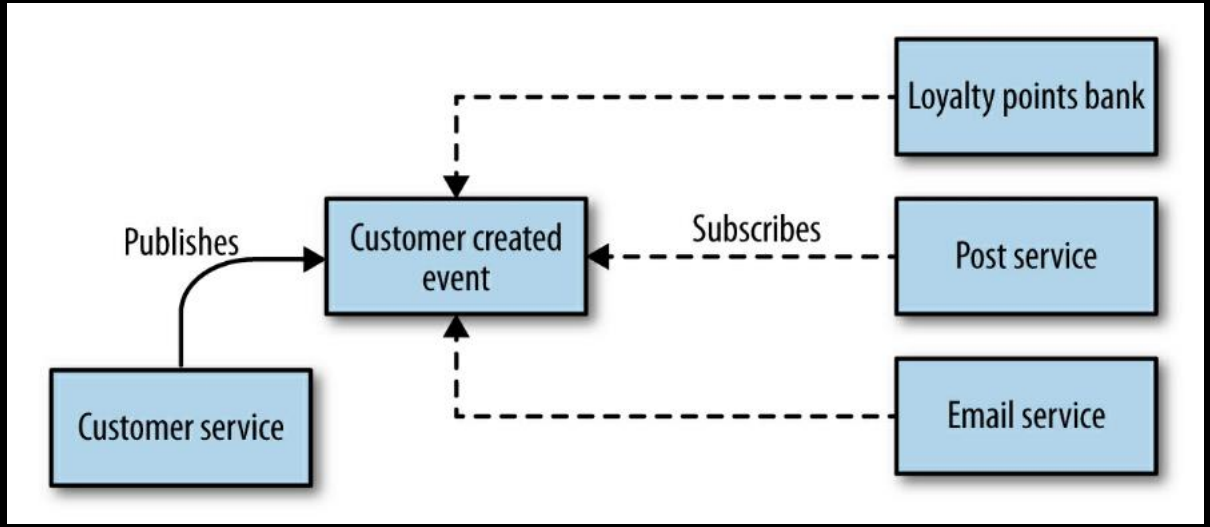
...

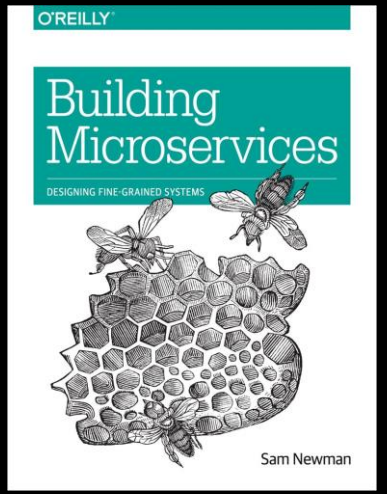

Who is responsible?

It is all about responsibility!



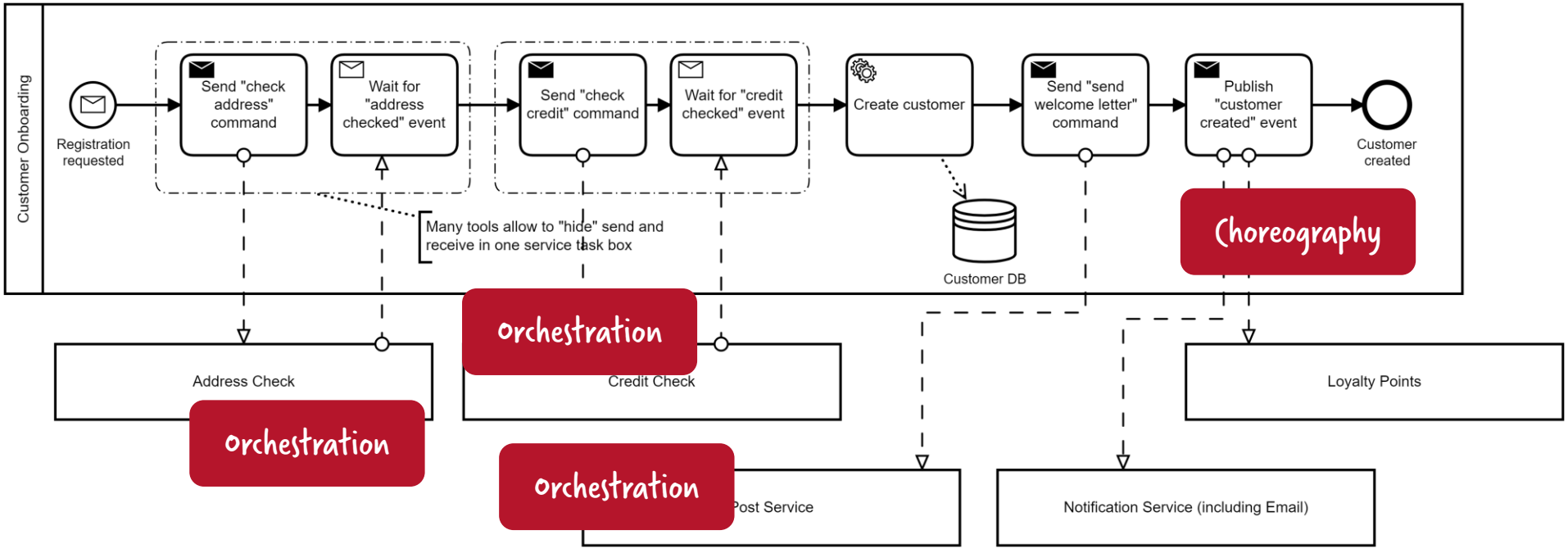
Customer Created

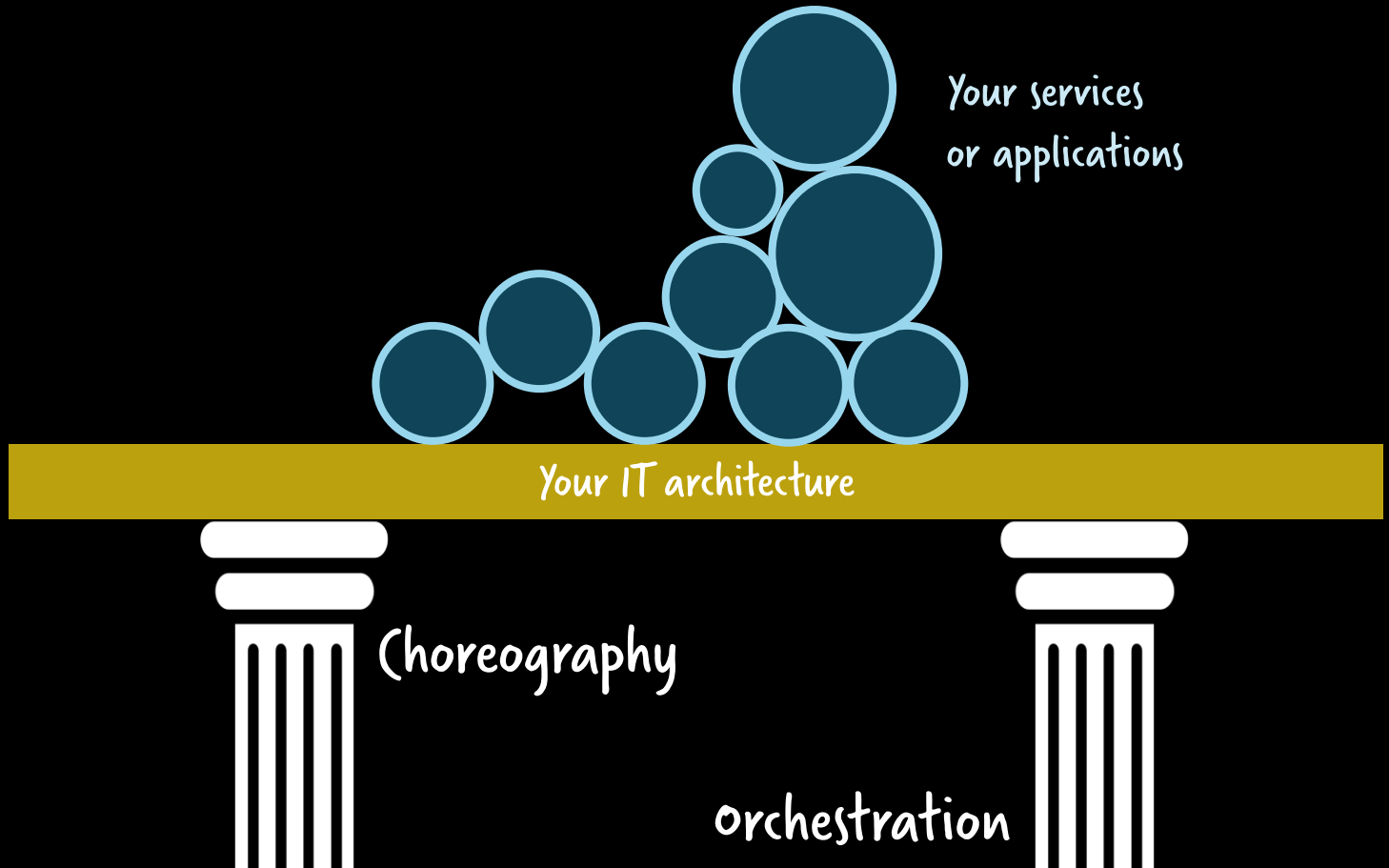


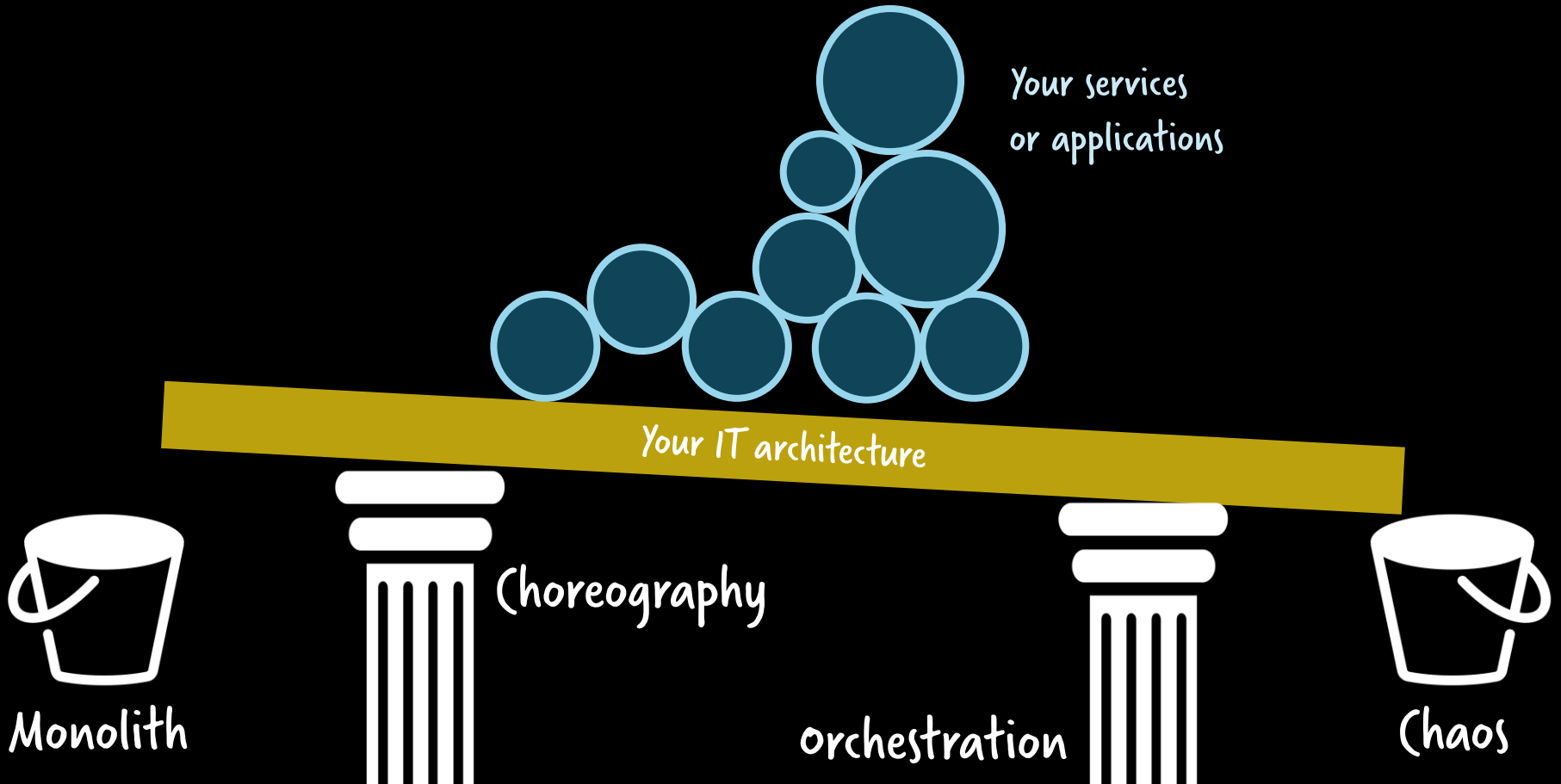


Sam Newman: Building Microservices

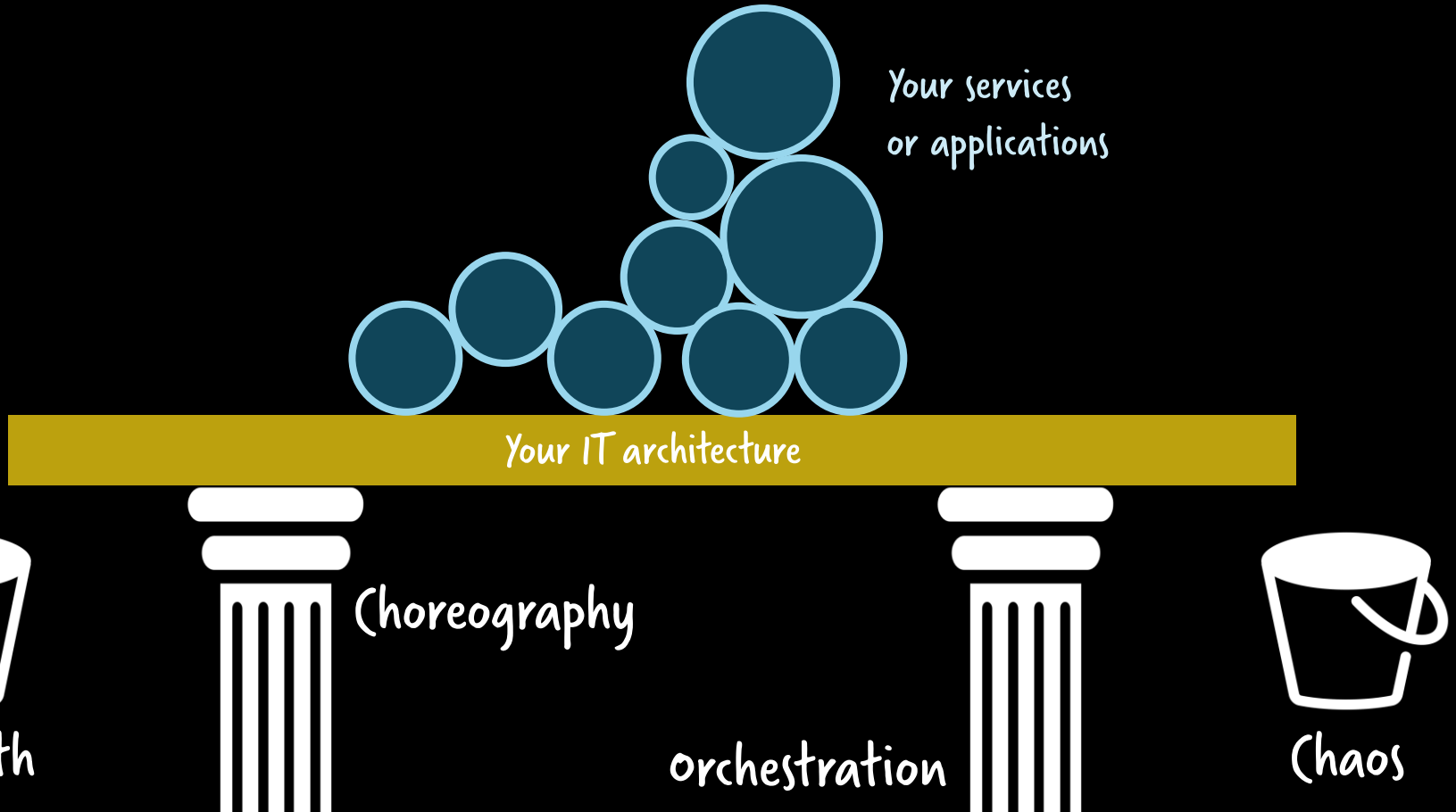
Customer onboarding is a mix!







Balance choreography and orchestration

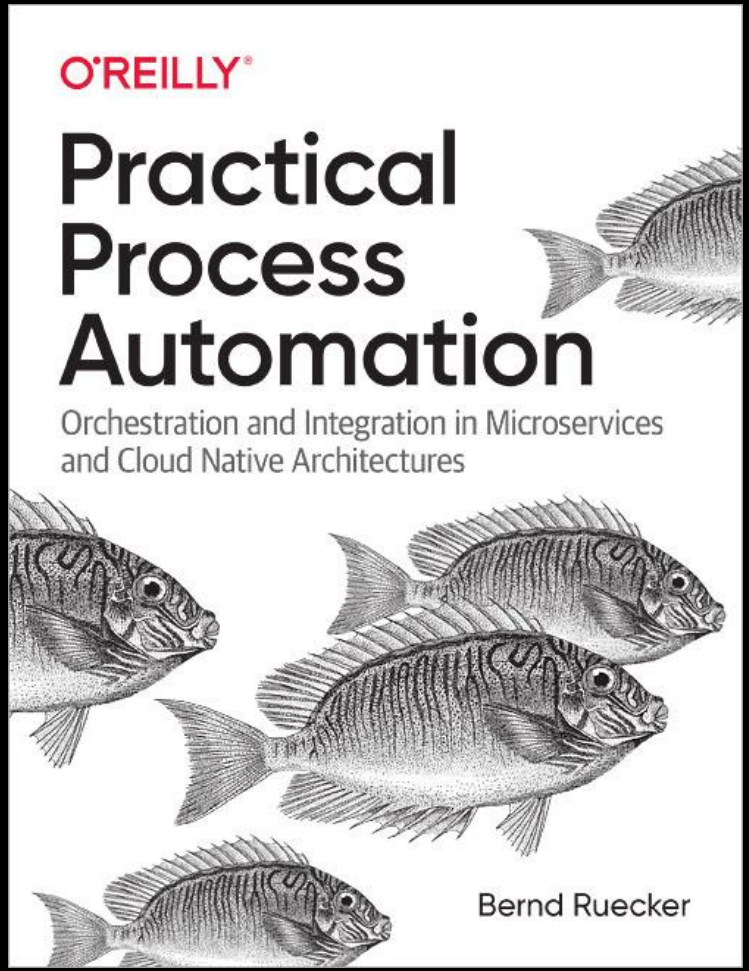


orchestration != central
Choreography != decoupled

orchestration = Command-driven
Choreography = Event-driven

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

Want to learn more?



https://learning.oreilly.com/get-learning/?code=PPAER20

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/