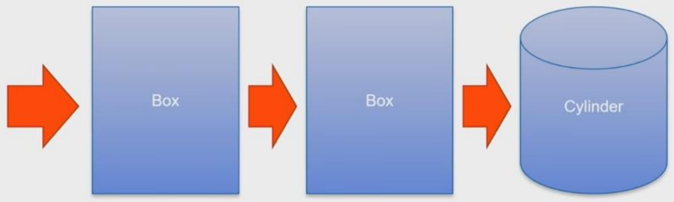
Event Driven Microservices

The sense, the non-sense and a way forward

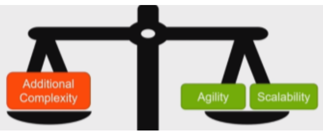
Once upon a time...

"Universal 'BBC' architecture"

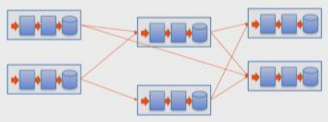


Microservices to the rescue!

Why microservices?



"Universal Microservices architecture"



Noun Driven Design

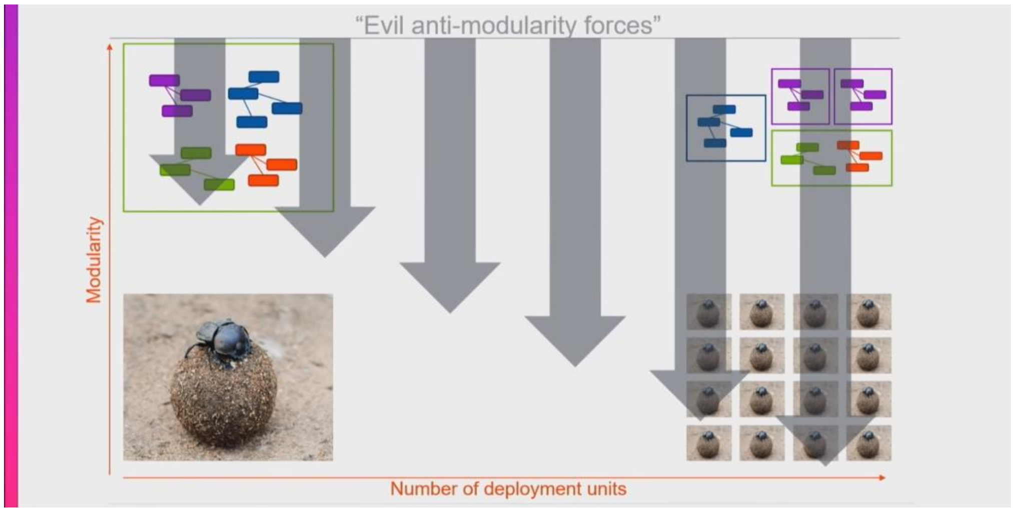
Noun? → Service!

OrderService

CustomerService

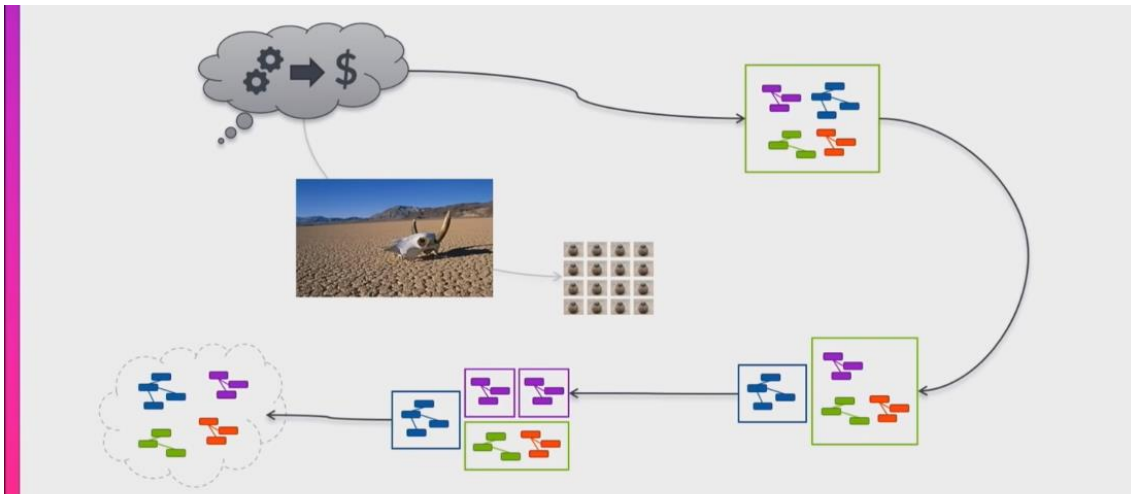
ProductService

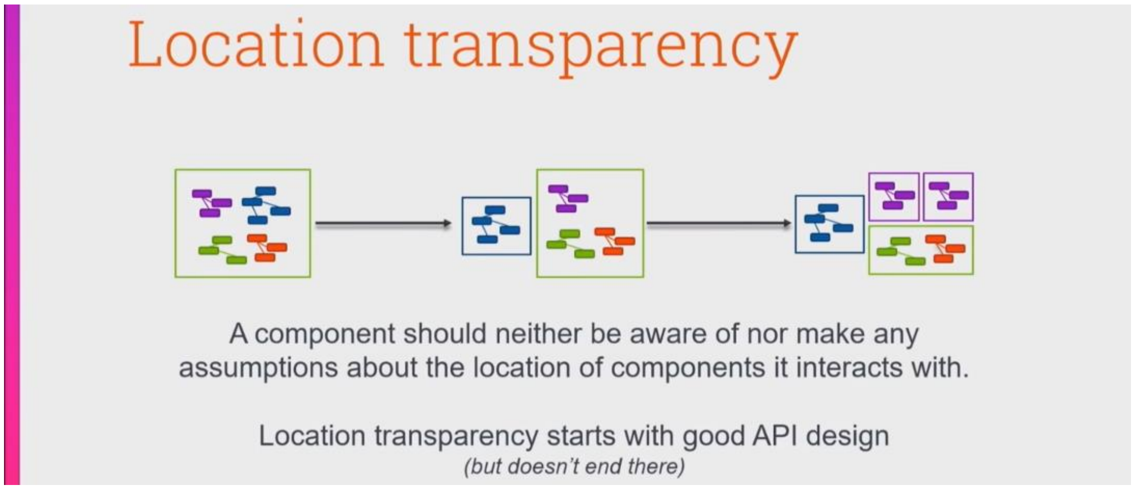
InventoryService



Want to build microservices?

Learn to build a decent monolith first!





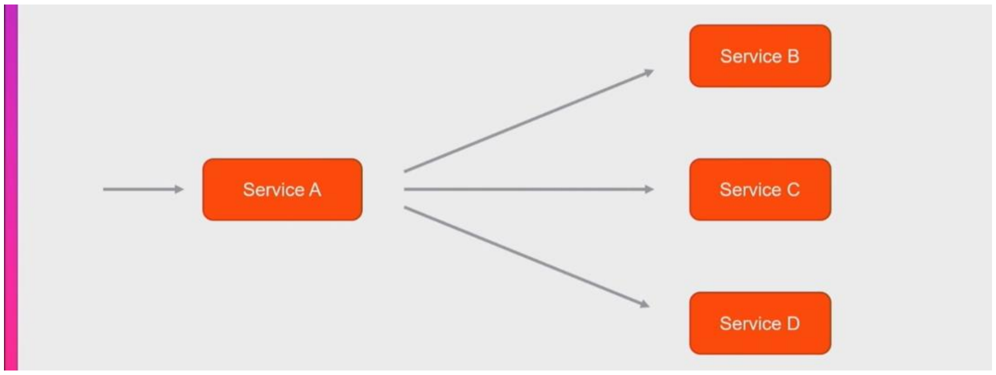
Location transparency

A component should neither be aware of nor make any assumptions about the location of components it interacts with.

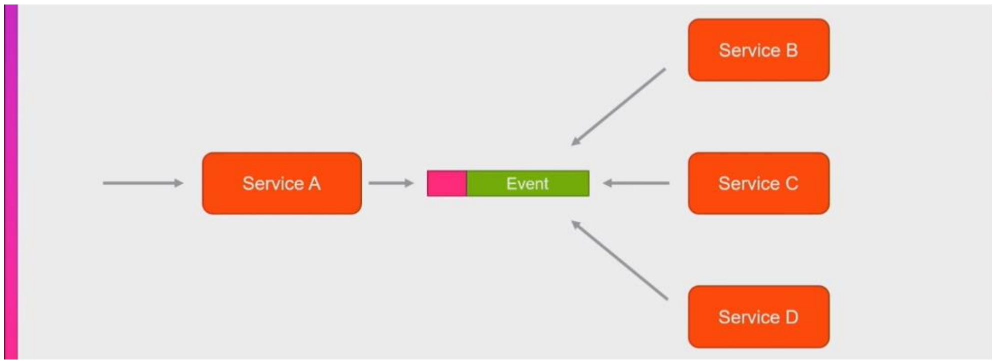
Location transparency starts with good API design
(but doesn't end there)

Make sure the services don't need to know where the other services are located or deployed

Events

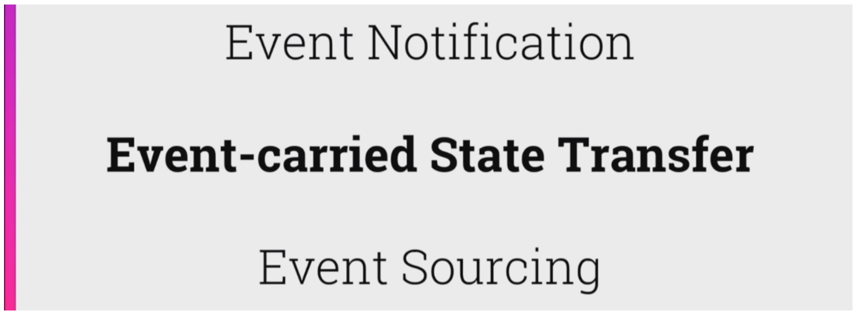


Services should not need to know where others are, just raise an event and watch for events you want.



"Event" all the things!

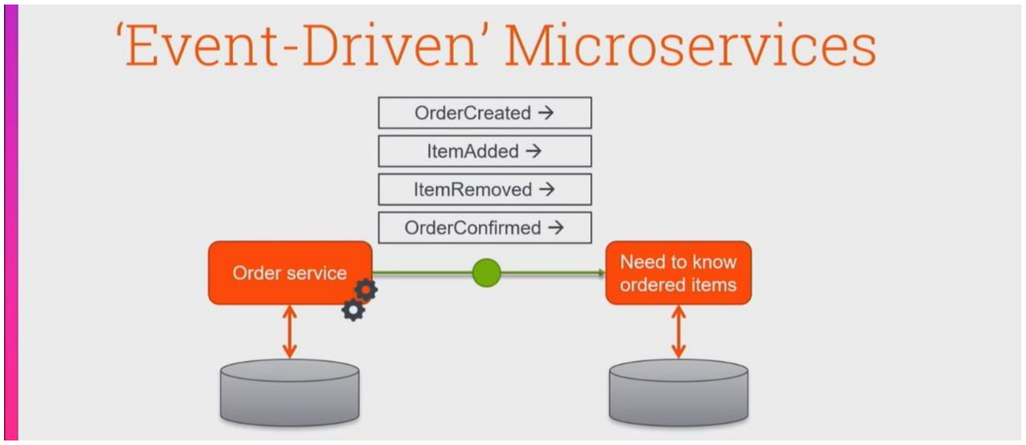
Maslow's Hammer

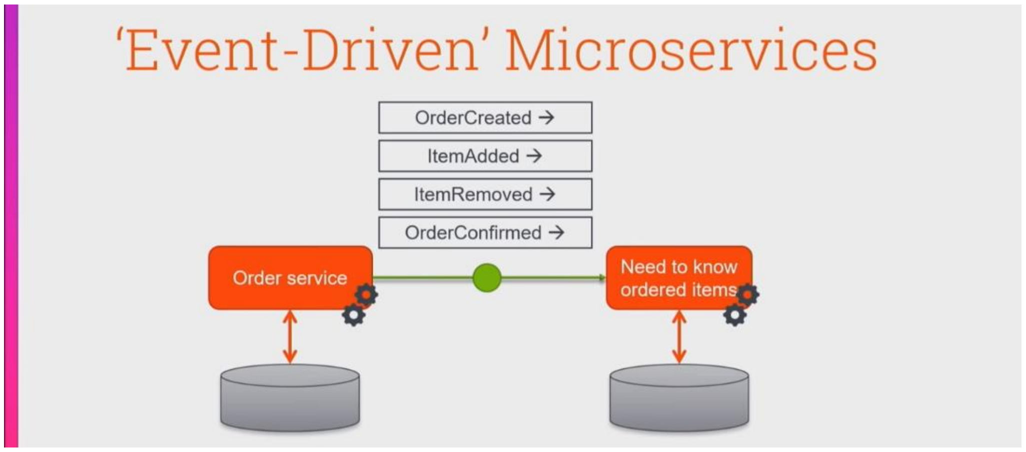


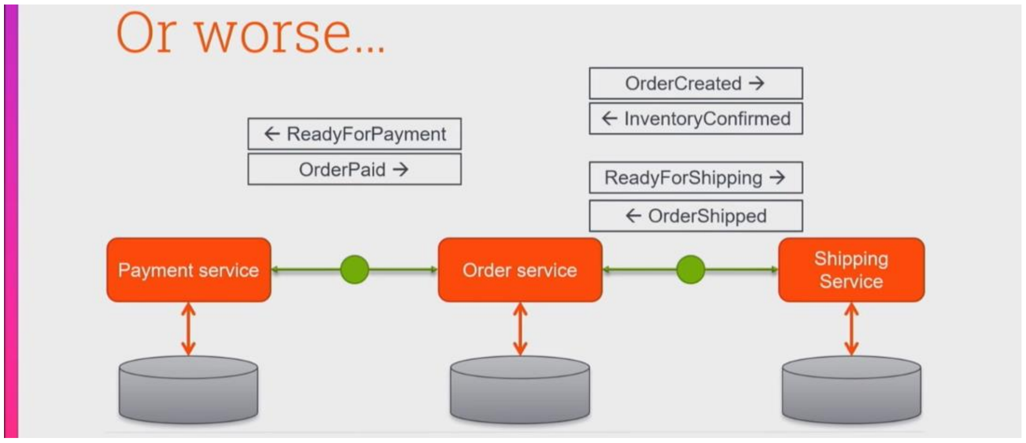
Event Notification

Event-carried State Transfer

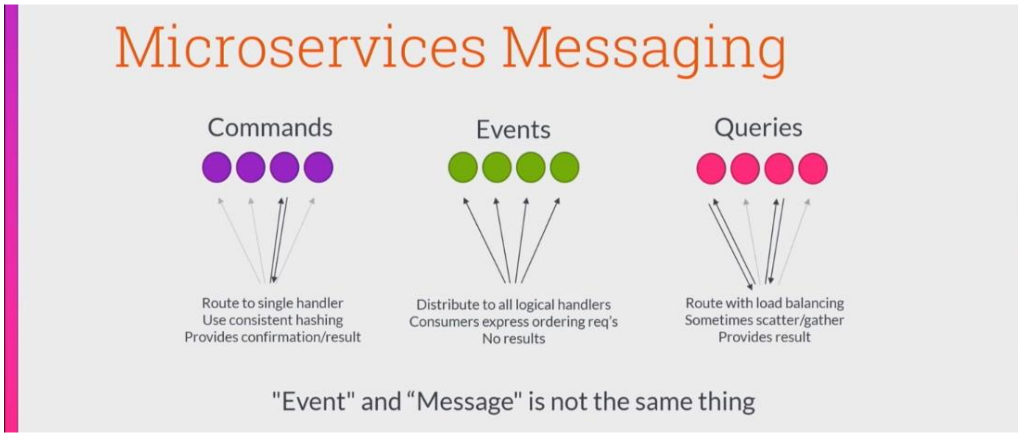
Event Sourcing







This style of communication is weird, the Order service can just communicate directly with the Shipping service via a direct command since it knows about the Shipping service

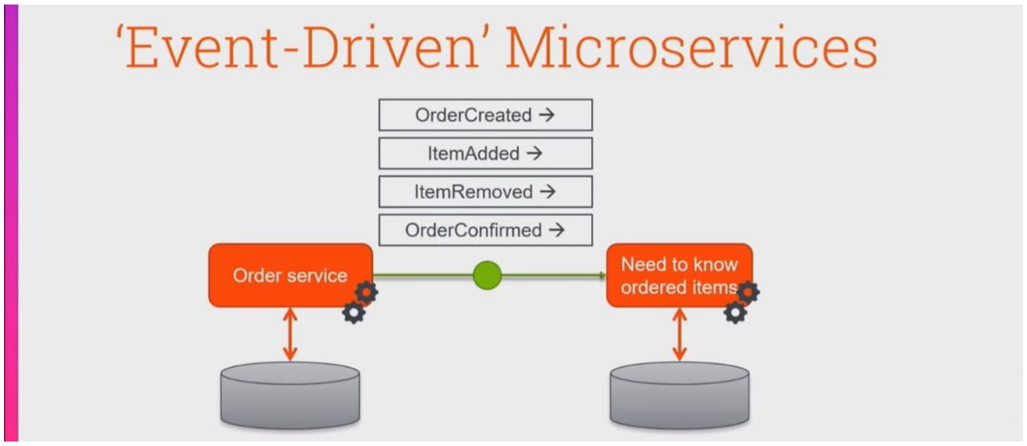


Route to single handler
Use consistent hashing
Provides confirmation/result

Distribute to all logical handlers
Consumers express ordering req's
No results

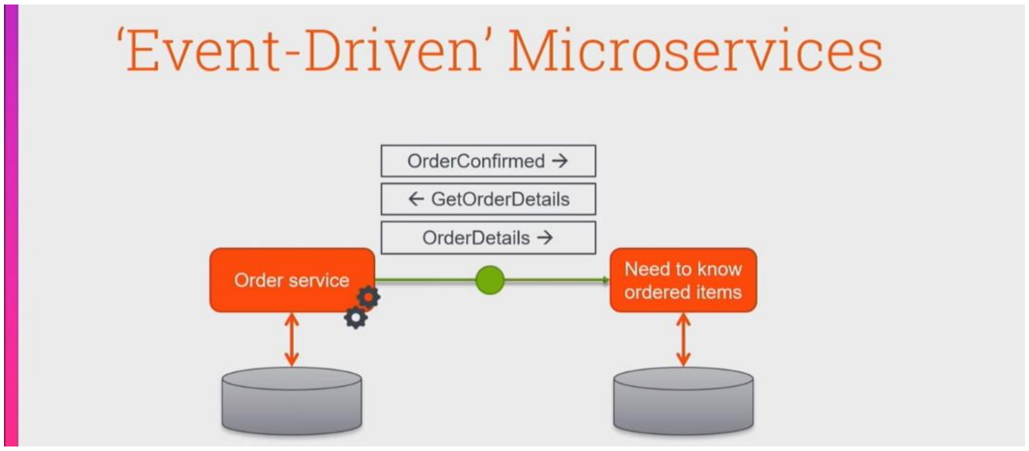
Route with load balancing
Sometimes scatter/gather
Provides result

"Event" and "Message" is not the same thing

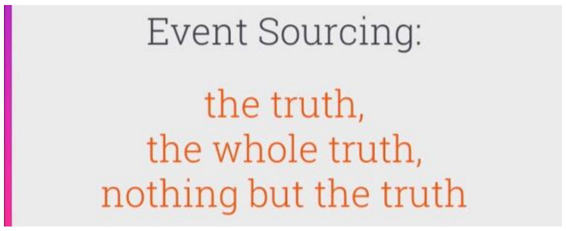


'Event-Driven' Microservices

We can implement this better as below

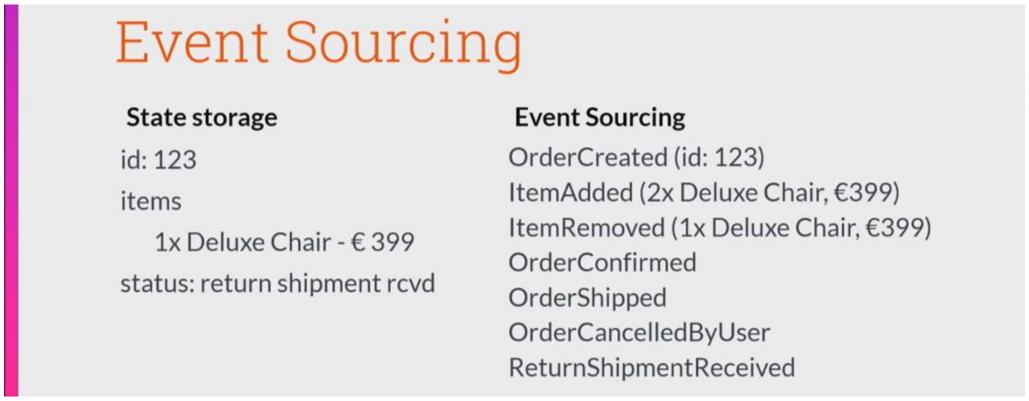


'Event-Driven' Microservices



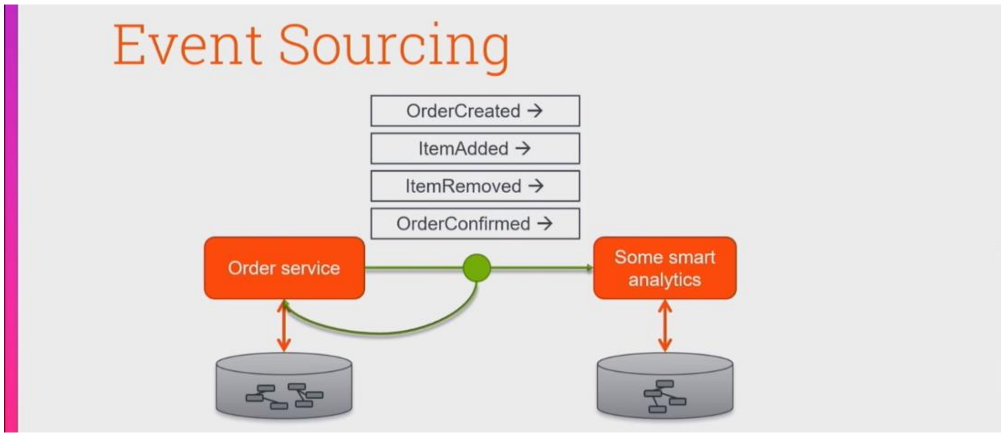
Event Sourcing:

the truth,
the whole truth,
nothing but the truth

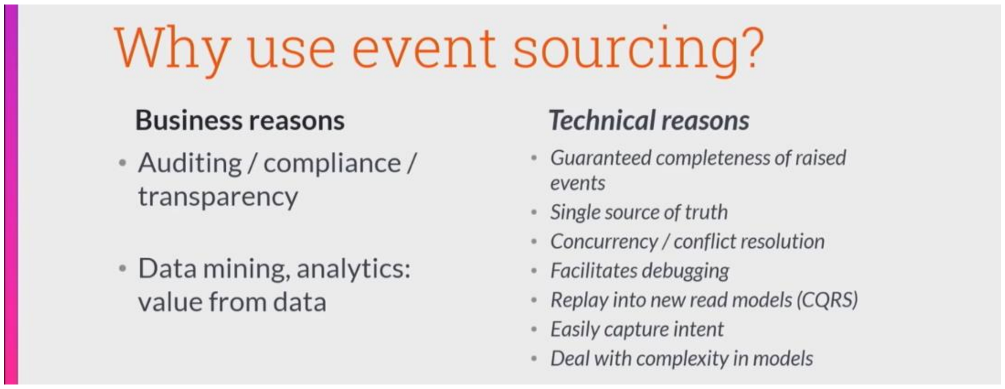


Event Sourcing

State storage	Event Sourcing
id: 123	OrderCreated (id: 123)
items	ItemAdded (2x Deluxe Chair, €399)
1x Deluxe Chair - € 399	ItemRemoved (1x Deluxe Chair, €399)
status: return shipment rcvd	OrderConfirmed
	OrderShipped
	OrderCancelledByUser
	ReturnShipmentReceived



Event Sourcing



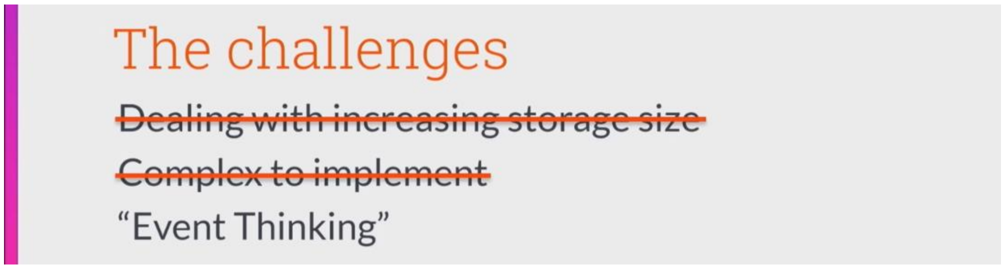
Why use event sourcing?

Business reasons

- Auditing / compliance / transparency
- Data mining, analytics: value from data

Technical reasons

- *Guaranteed completeness of raised events*
- *Single source of truth*
- *Concurrency / conflict resolution*
- *Facilitates debugging*
- *Replay into new read models (CQRS)*
- *Easily capture intent*
- *Deal with complexity in models*

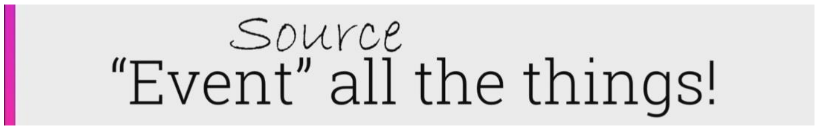


The challenges

~~Dealing with increasing storage size~~

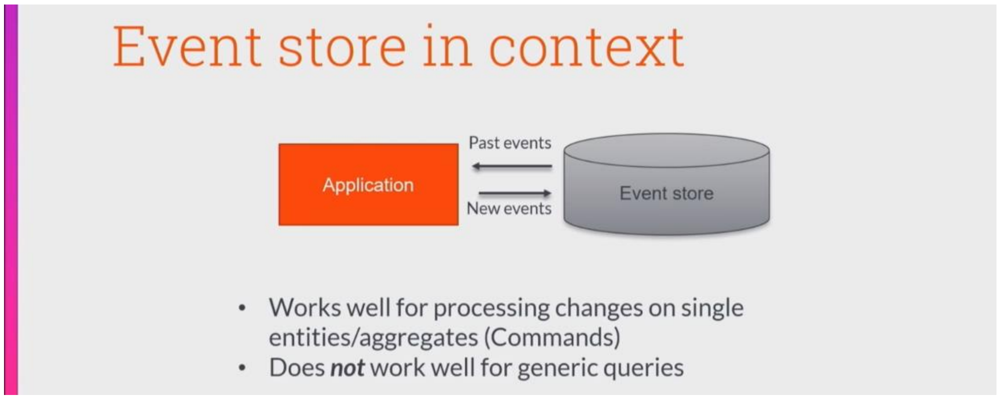
~~Complex to implement~~

“Event Thinking”



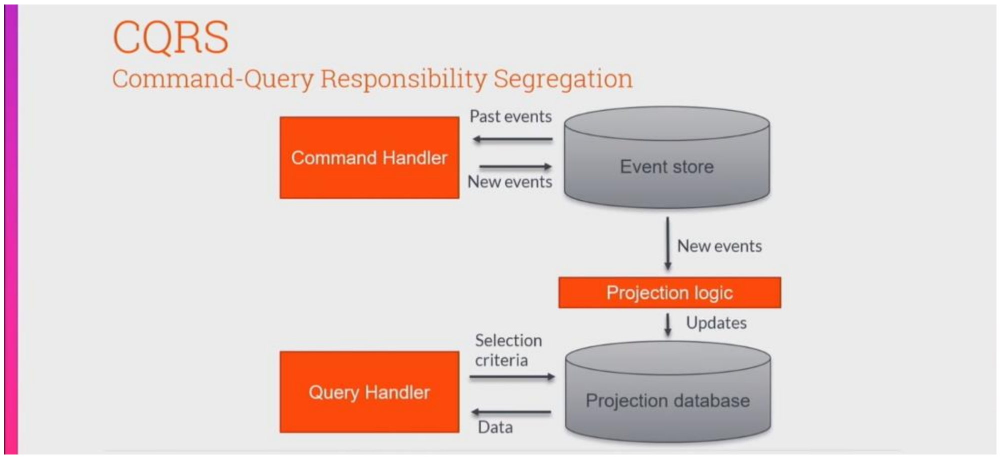
Source

“Event” all the things!



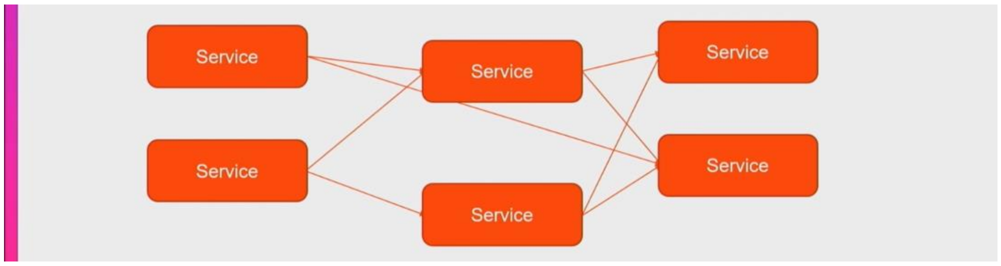
Event store in context

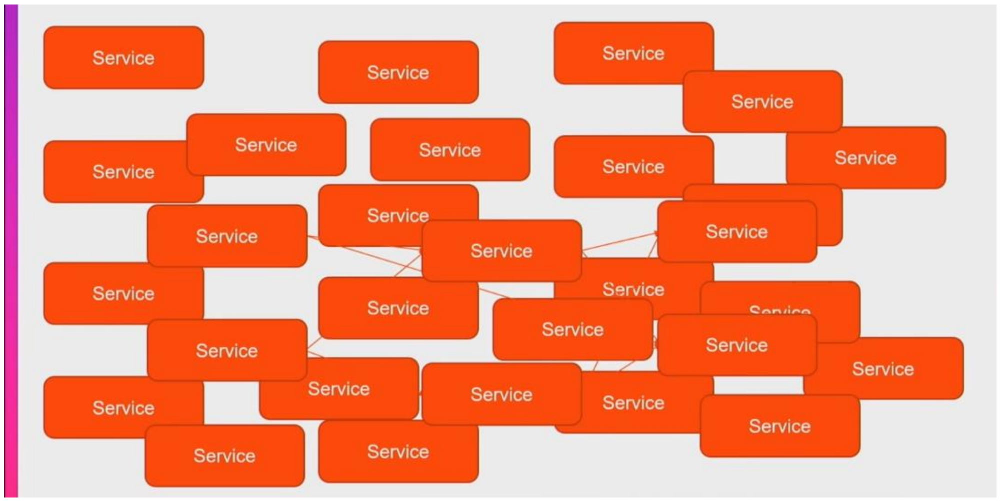
- Works well for processing changes on single entities/aggregates (Commands)
- Does ***not*** work well for generic queries



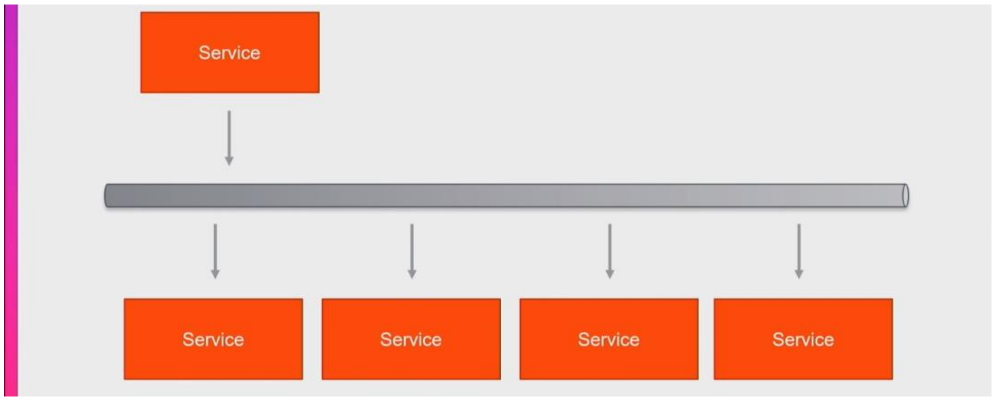
This is a pattern where we split the responsibility into 2, querying and commands to the event store using the relevant models for retrieving or saving the events.

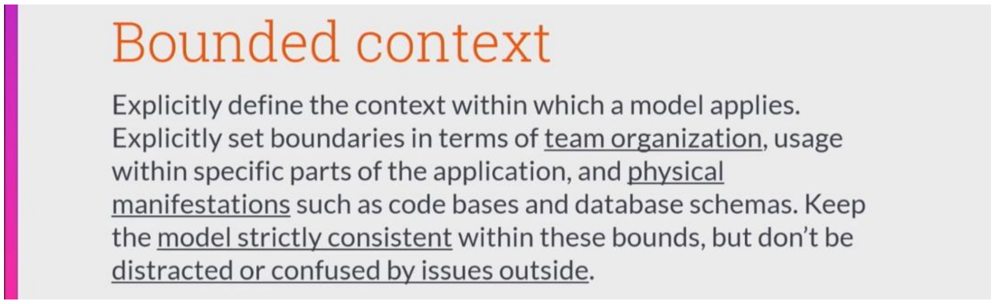
“CQRS” all the things?





There is a contract between the 2 communicating components, a service can have contracts with many many services

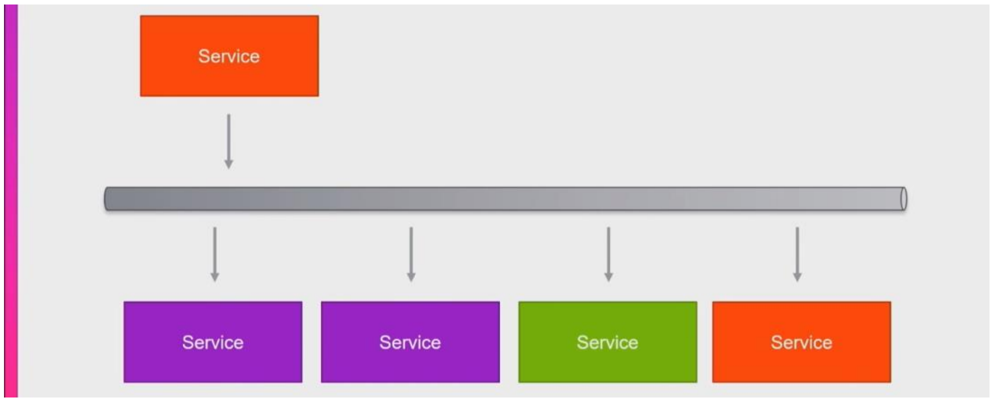




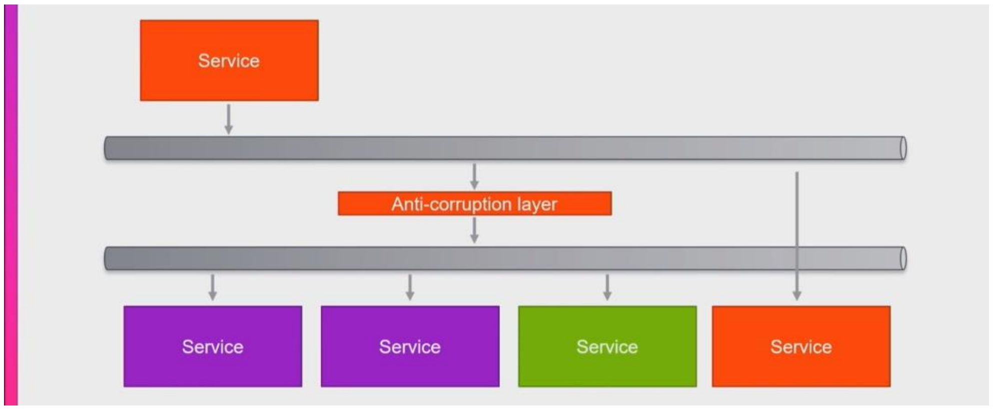
Bounded context

Explicitly define the context within which a model applies. Explicitly set boundaries in terms of team organization, usage within specific parts of the application, and physical manifestations such as code bases and database schemas. Keep the model strictly consistent within these bounds, but don't be distracted or confused by issues outside.

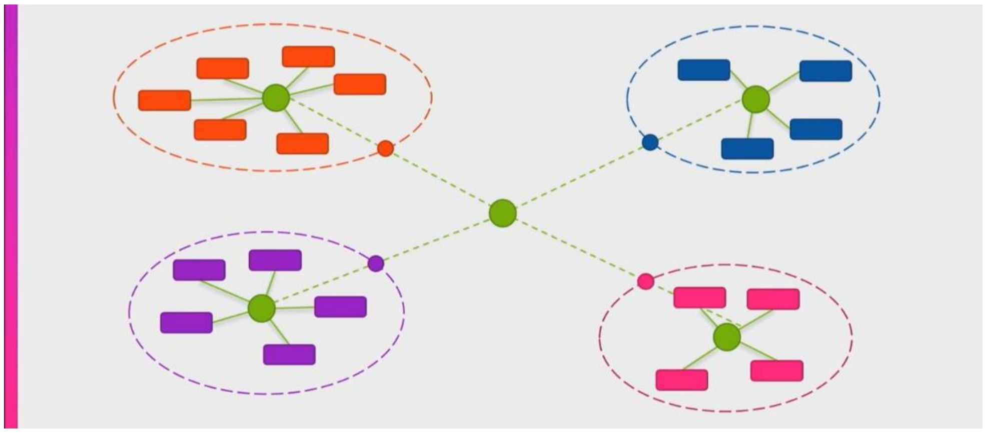
This refers to an area where certain words have a specific meaning.



We can assign all the services to bounded contexts, now we see 2 services in the same context



Services in the same context can read from the same stream. You might have to transform the data for services in different contexts in a sort of anti-corruption layer



In closing....

Consider *commands*
and *queries*
as much as *events*

Sharing is caring

Beware coupling
across
bounded contexts

"Microservice Journey"

Monolith first

wax-on , wax-off!