ScaleMePlease

Microservice Design Patterns

Michel Barret

Pierrick Rassat

Better, Faster, Stronger

SERVICES

Better, Faster, Stronger

Data API Storage

SERVICES

Protocols Behaviours

Better, Faster, Stronger

Reliability **Availability** **Scalability**

Data API Storage

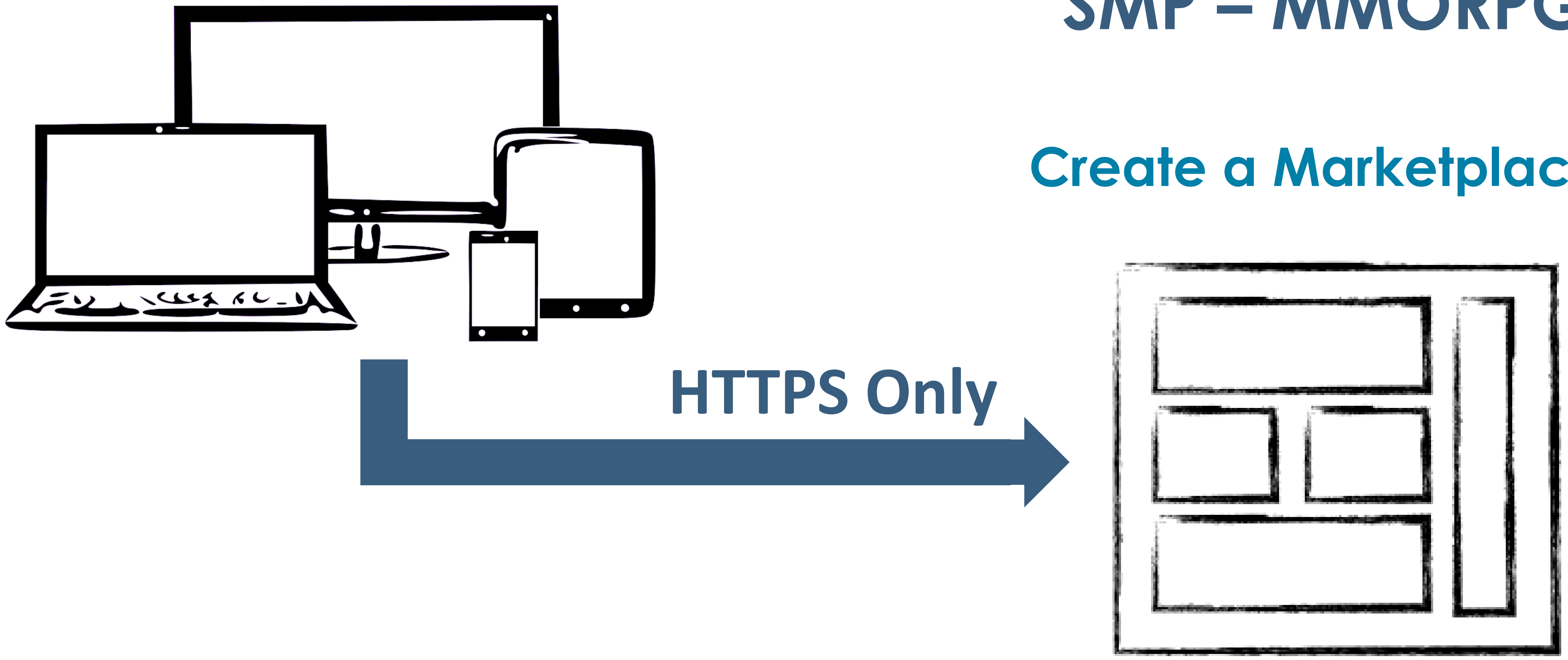
SERVICES

Protocols Behaviour

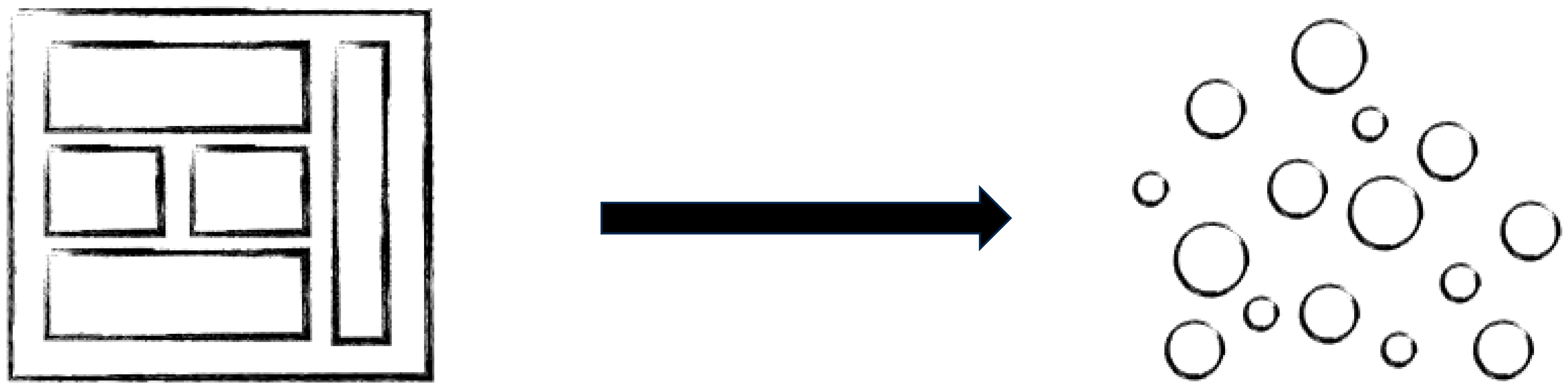
Resilience **Evoluability**

SMP – MMORPG

Create a Marketplace



From Monolith to microservices

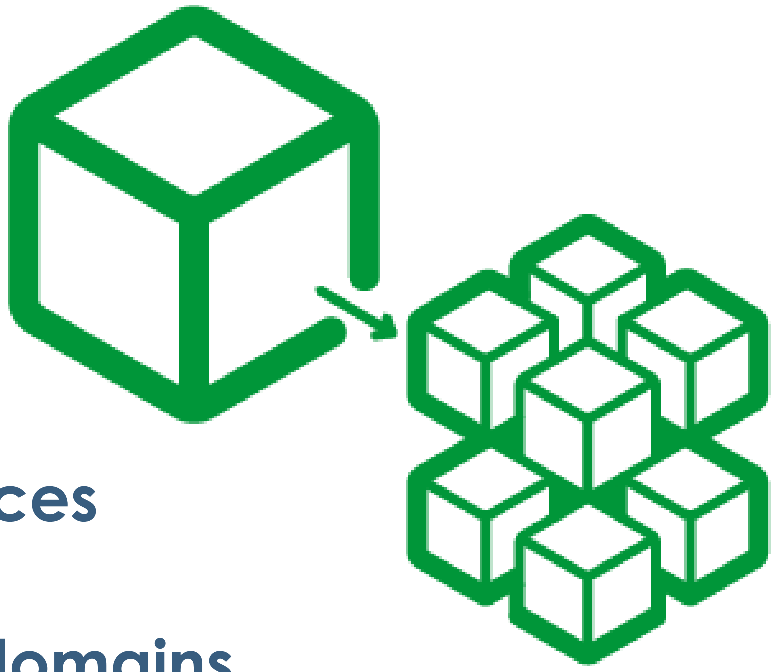


- More and more users
  - More requests, more data to process and store

- Improve reliability, availability…
  - Single point of failure, cascading errors

Breaking up into Microservices

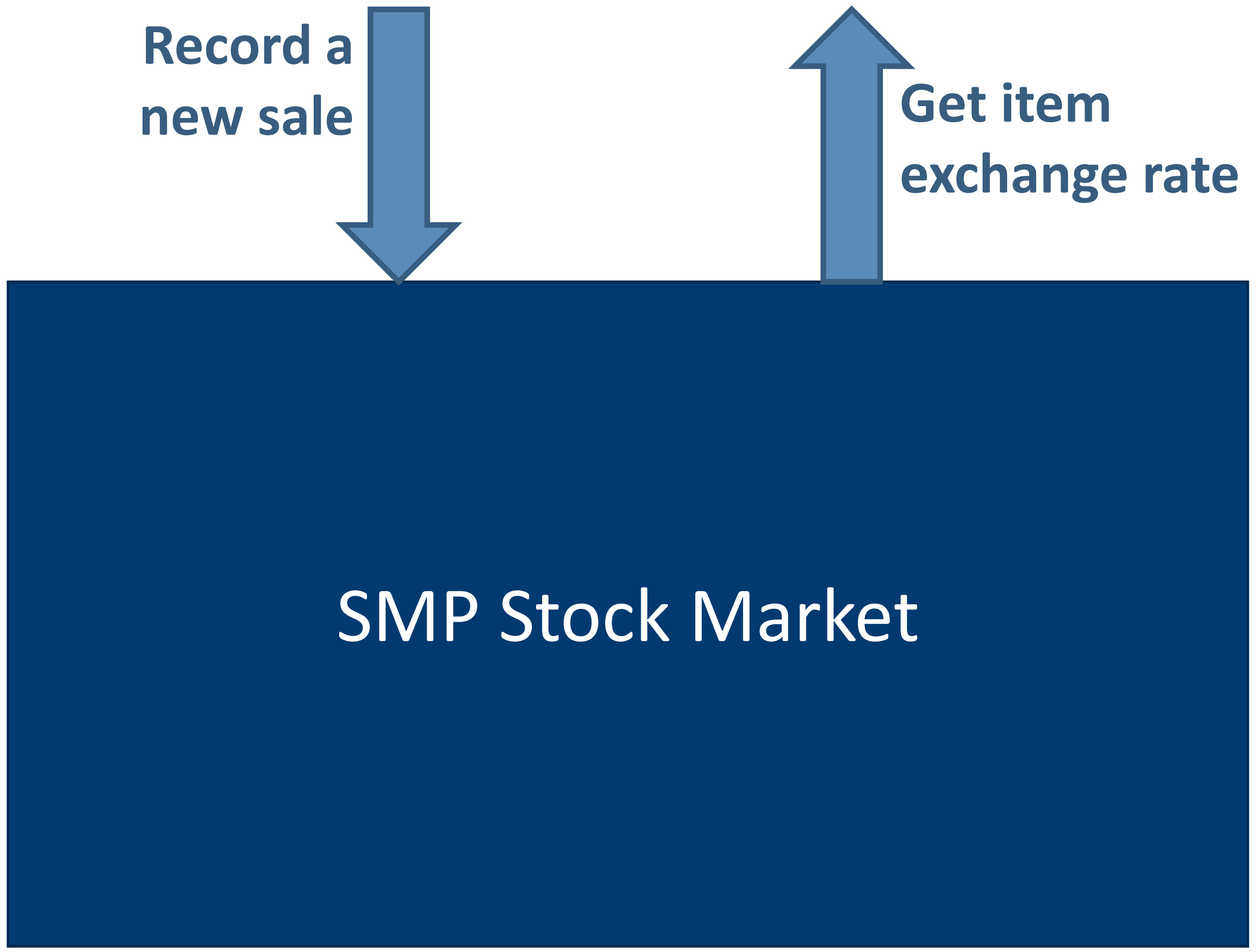
- Single Responsability Principle
- Reduce tight coupling between services
- Breaking up into functions or model domains

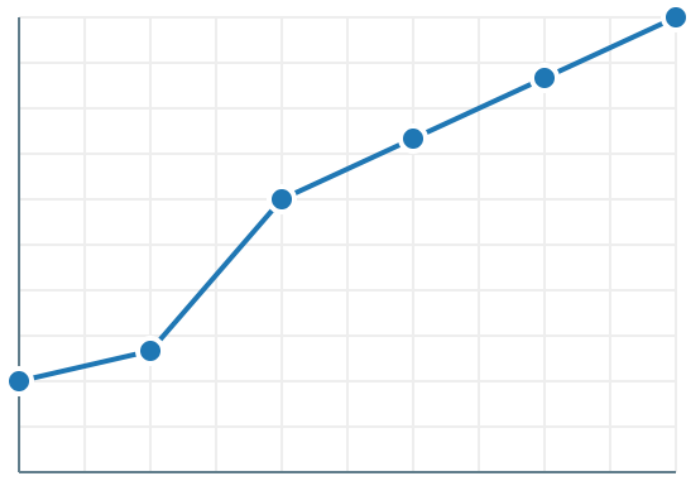


First Service

First Service

Community market stock exchange rate





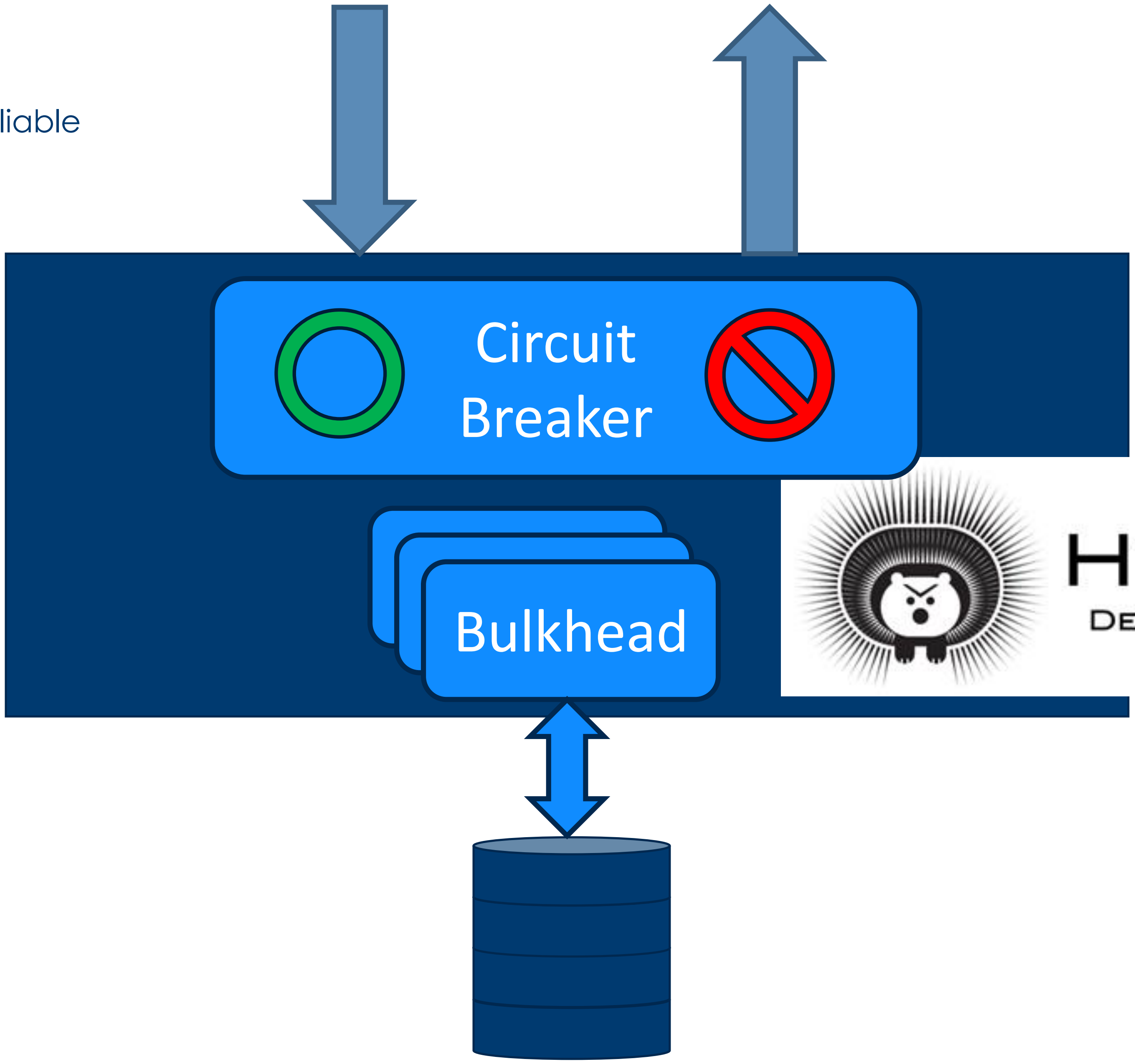
Service Responsability

Service API

…

First Service

Make the service reliable

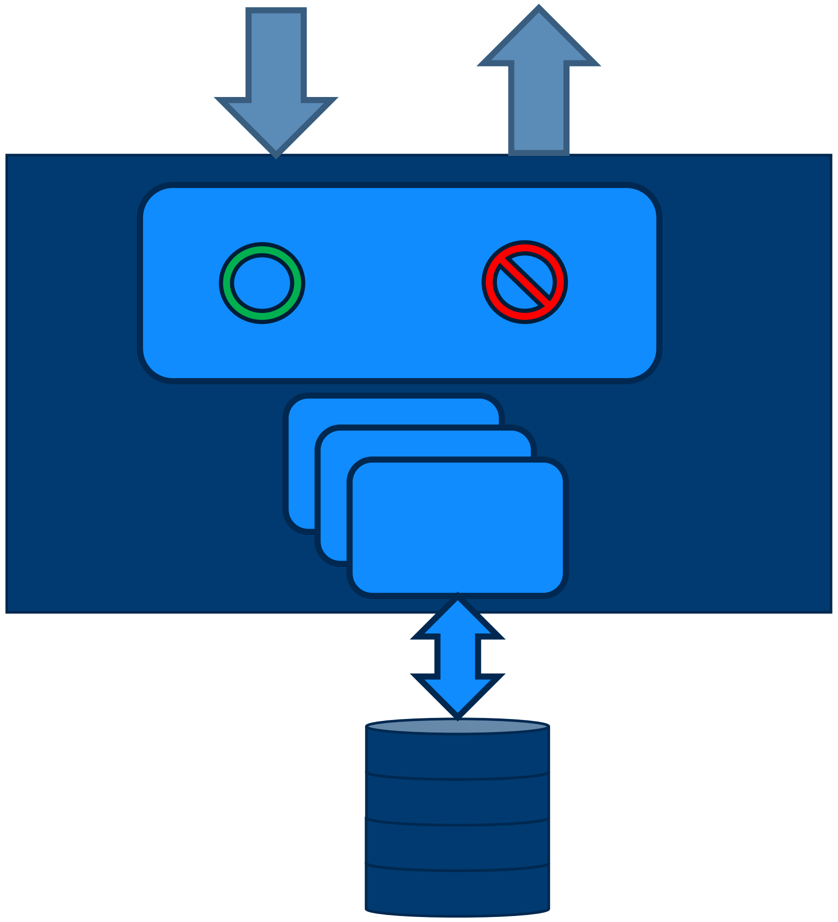


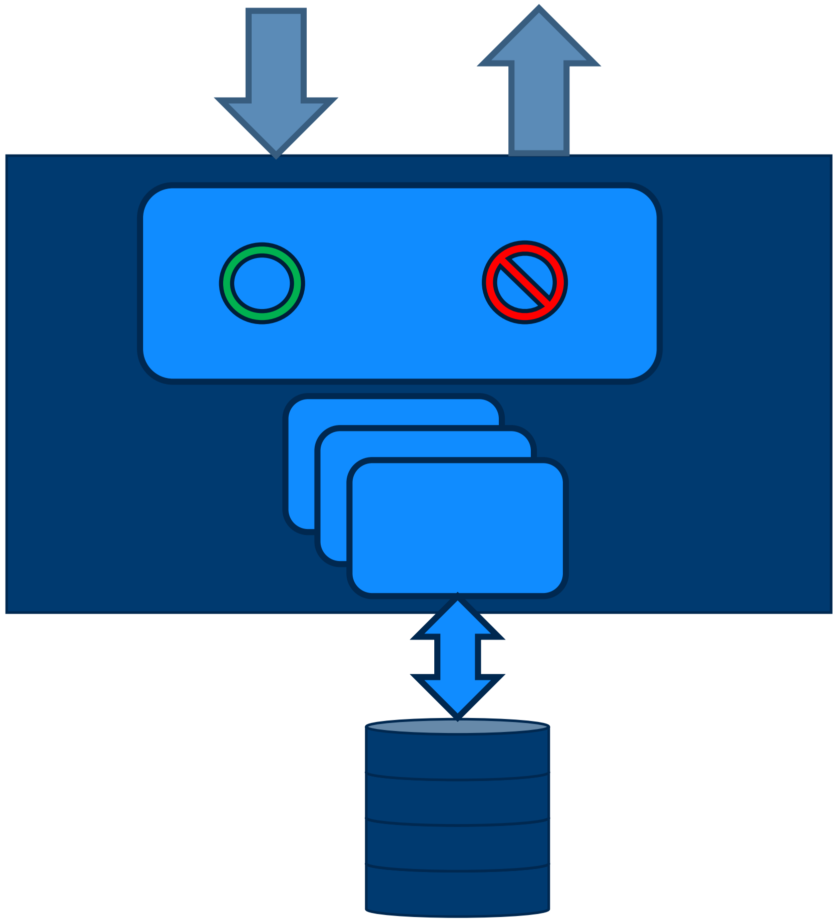
HYSTRIX

DEFEND YOUR APP

First Service

Avoid single point of failure



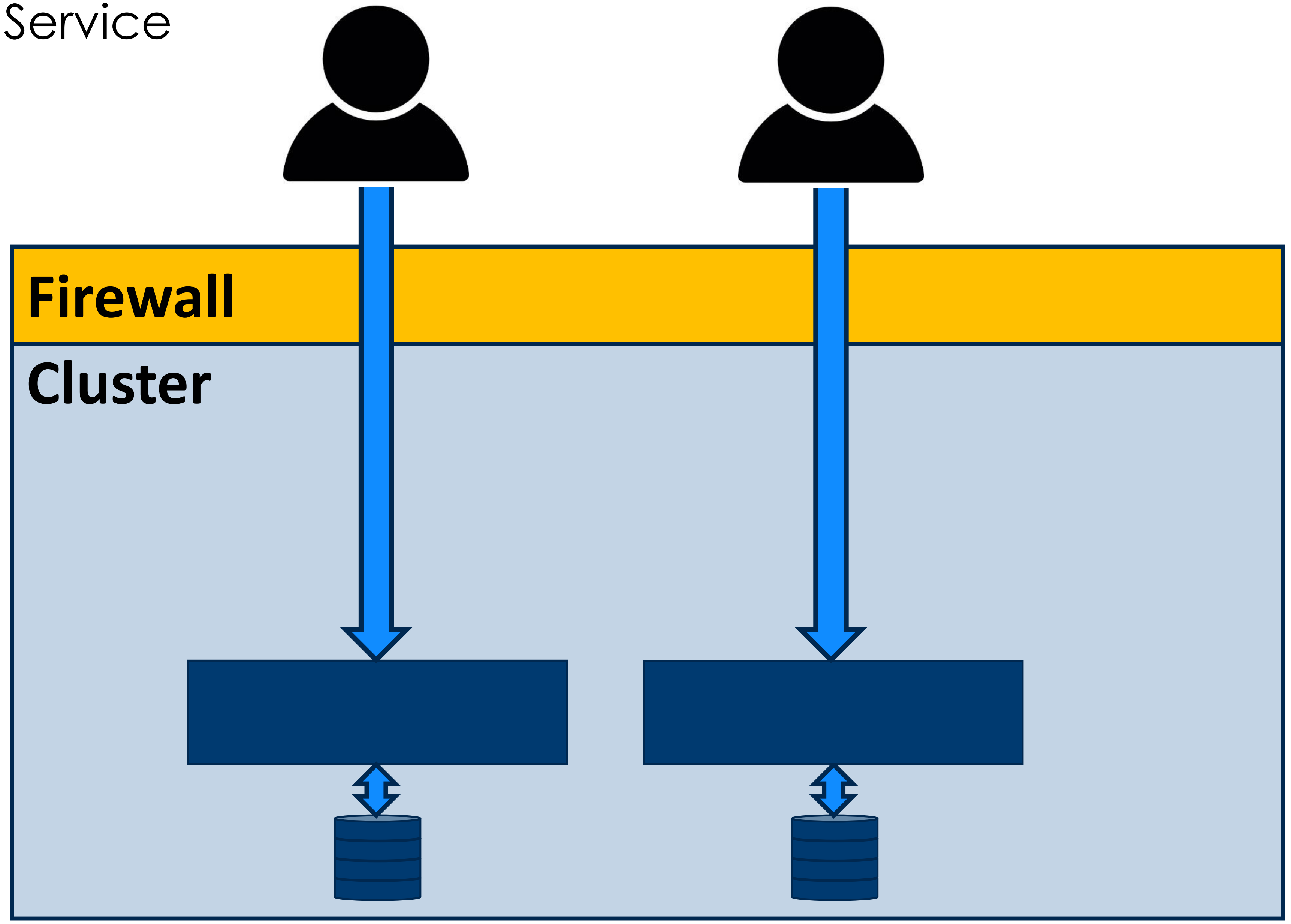


VISEO

Infrastructure

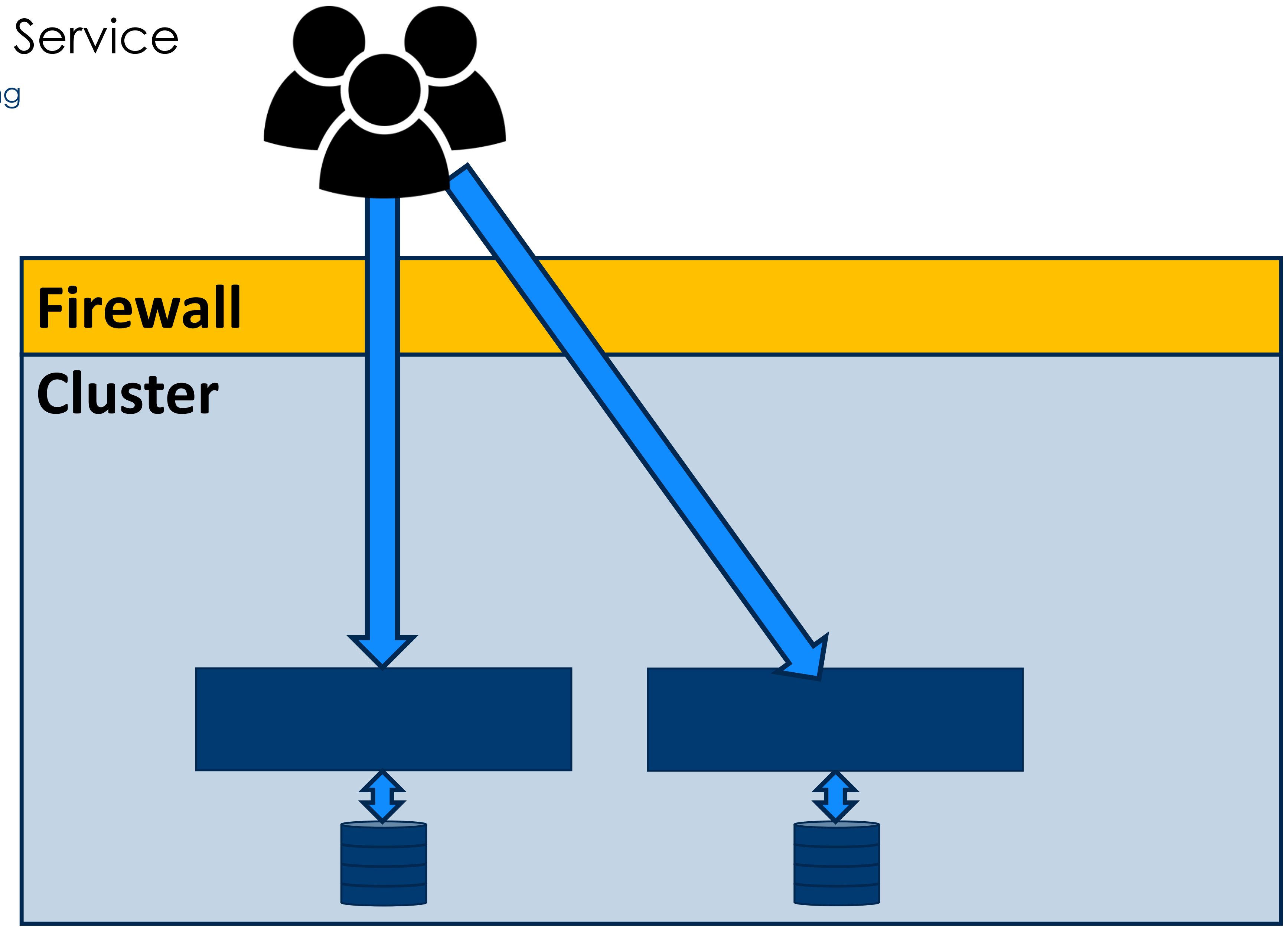
To connect the world!

First Service



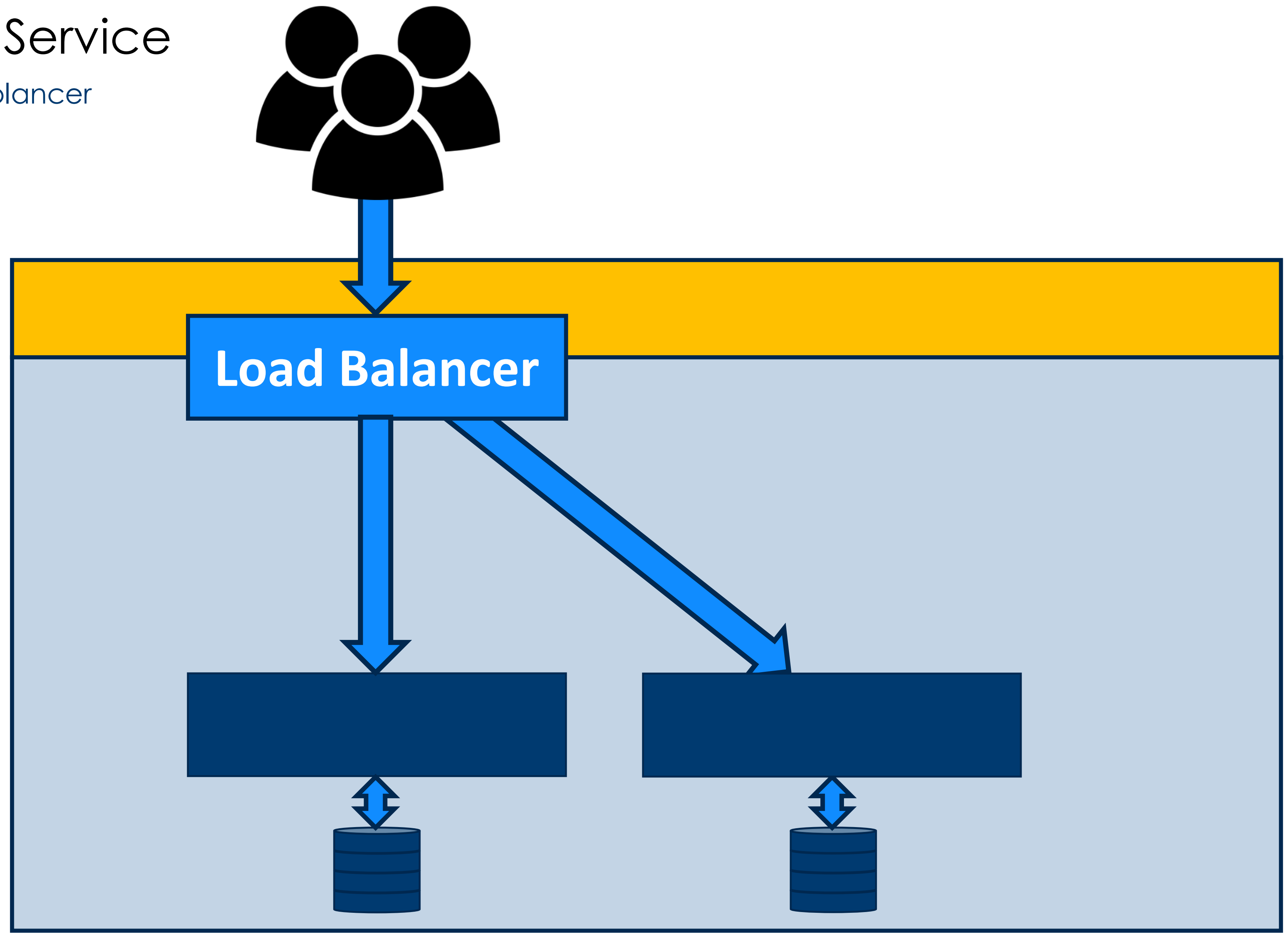
First Service

Routing



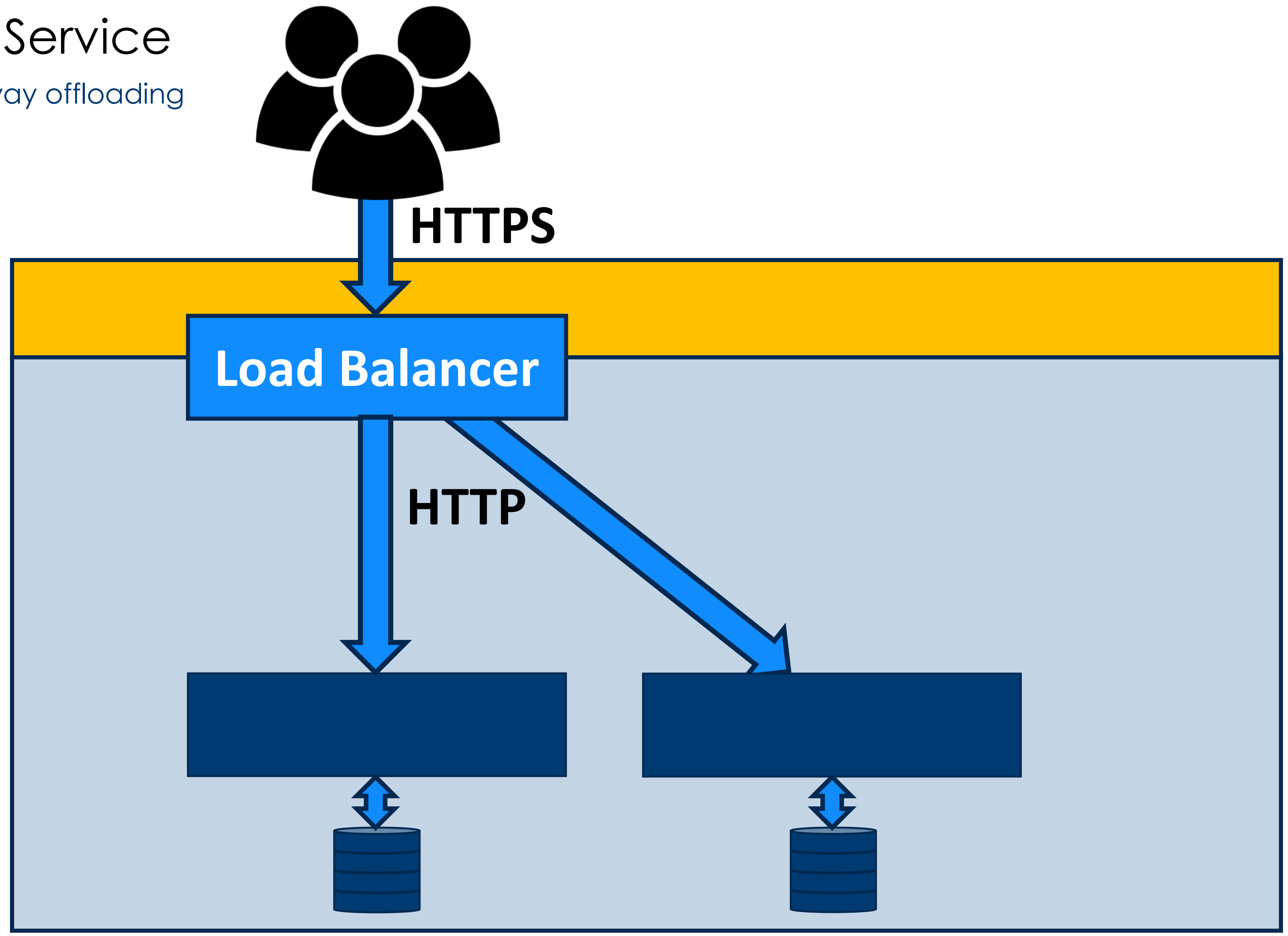
First Service

Load blancer



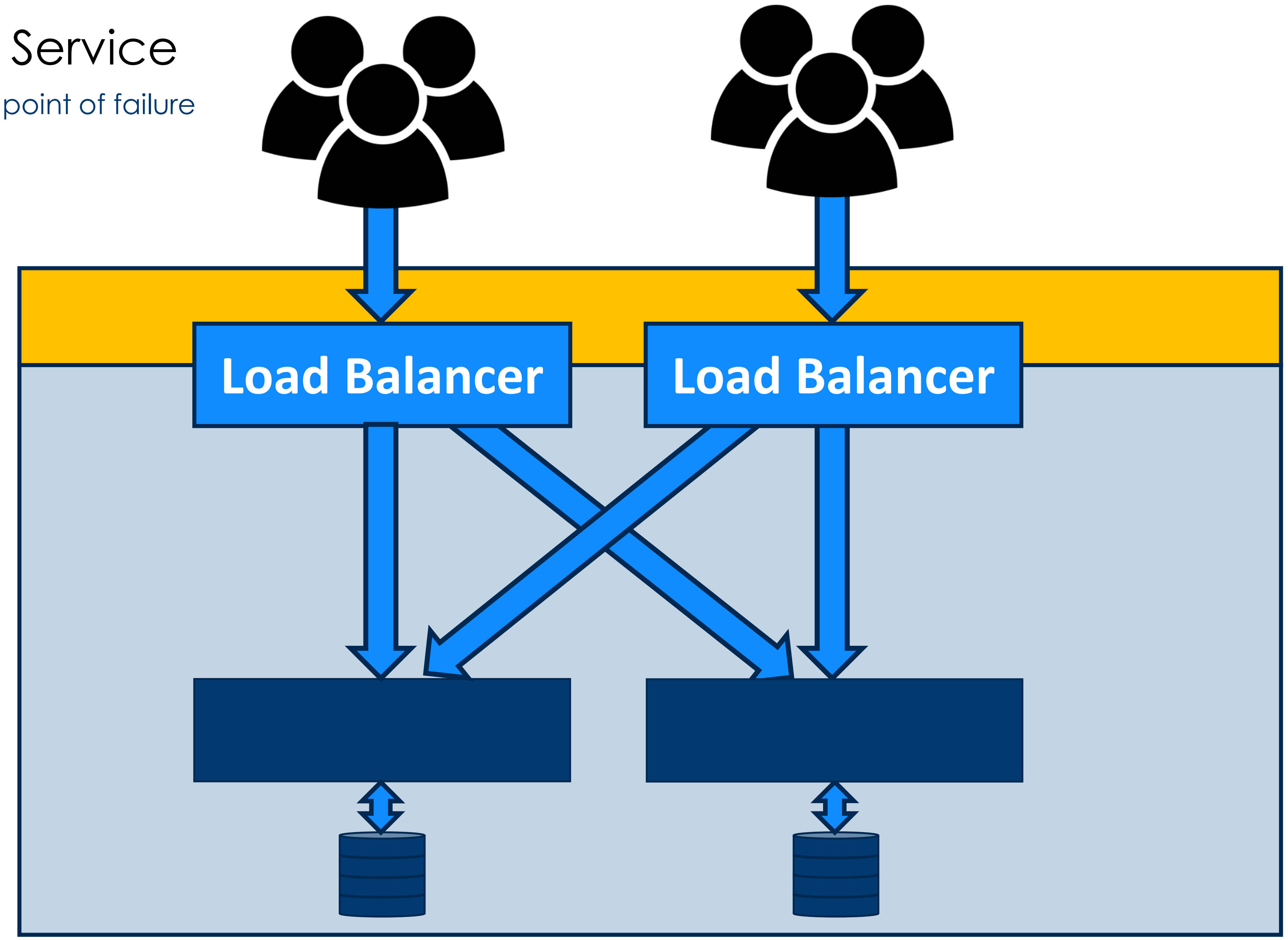
First Service

Gateway offloading



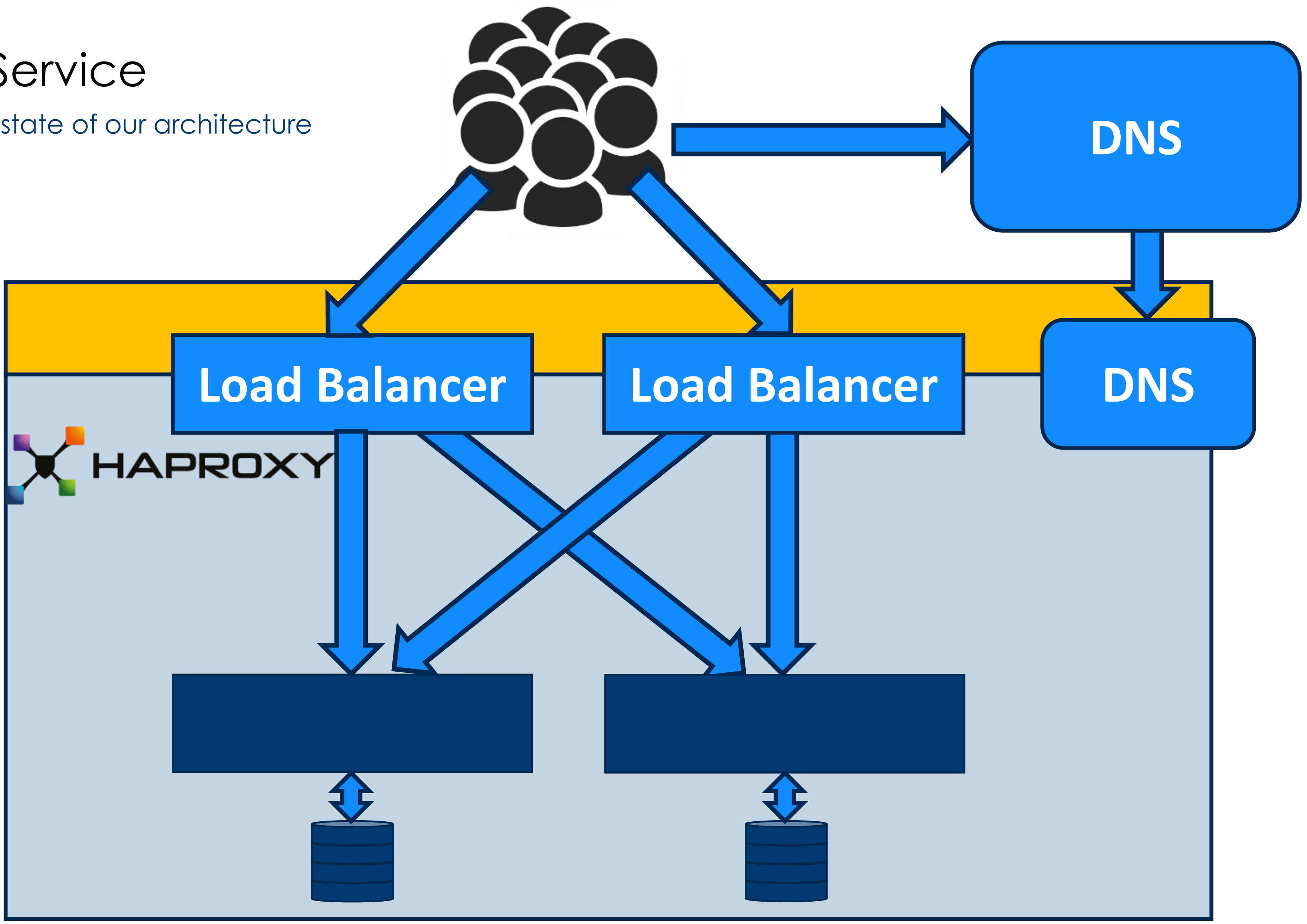
First Service

Single point of failure

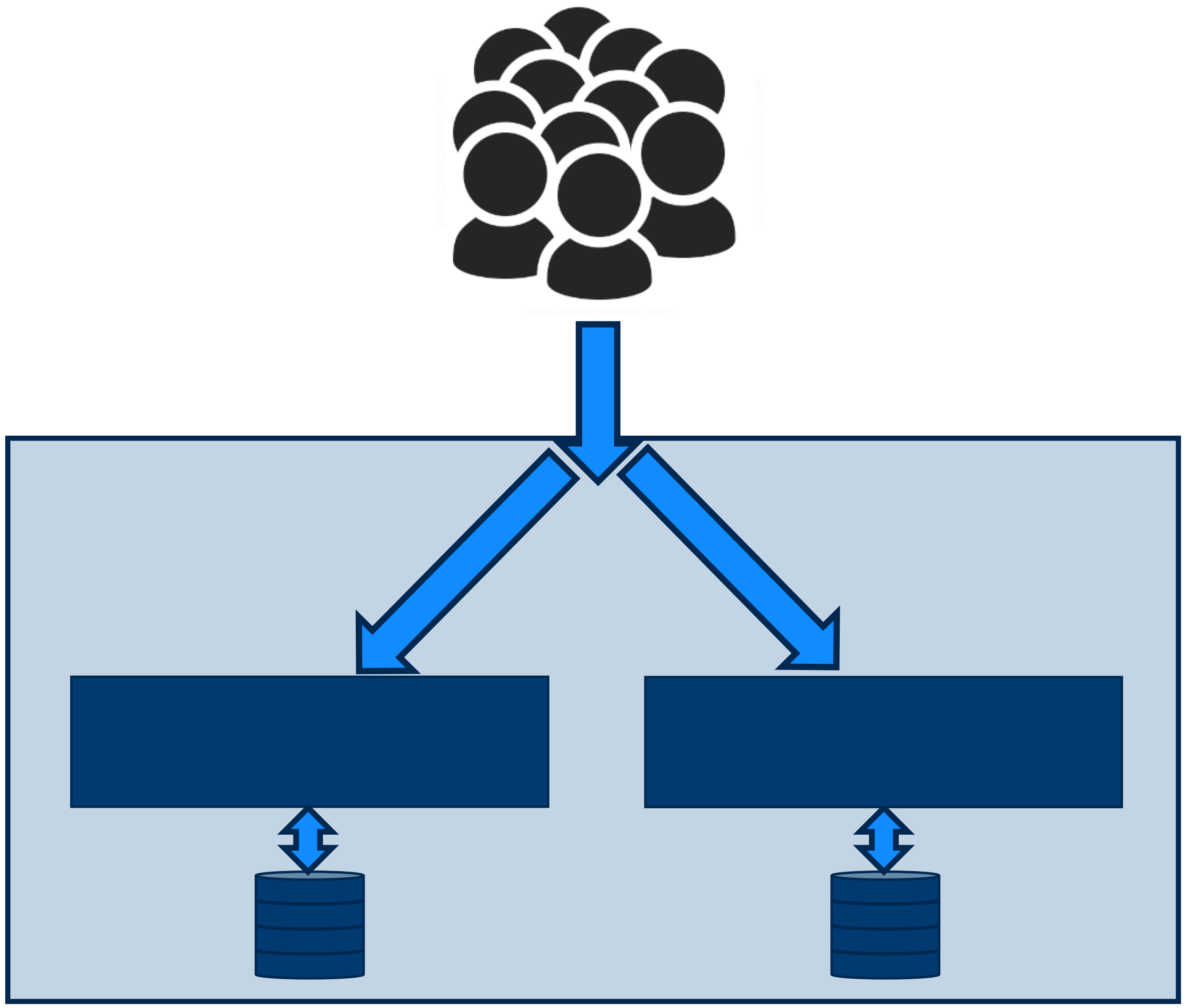


First Service

Current state of our architecture



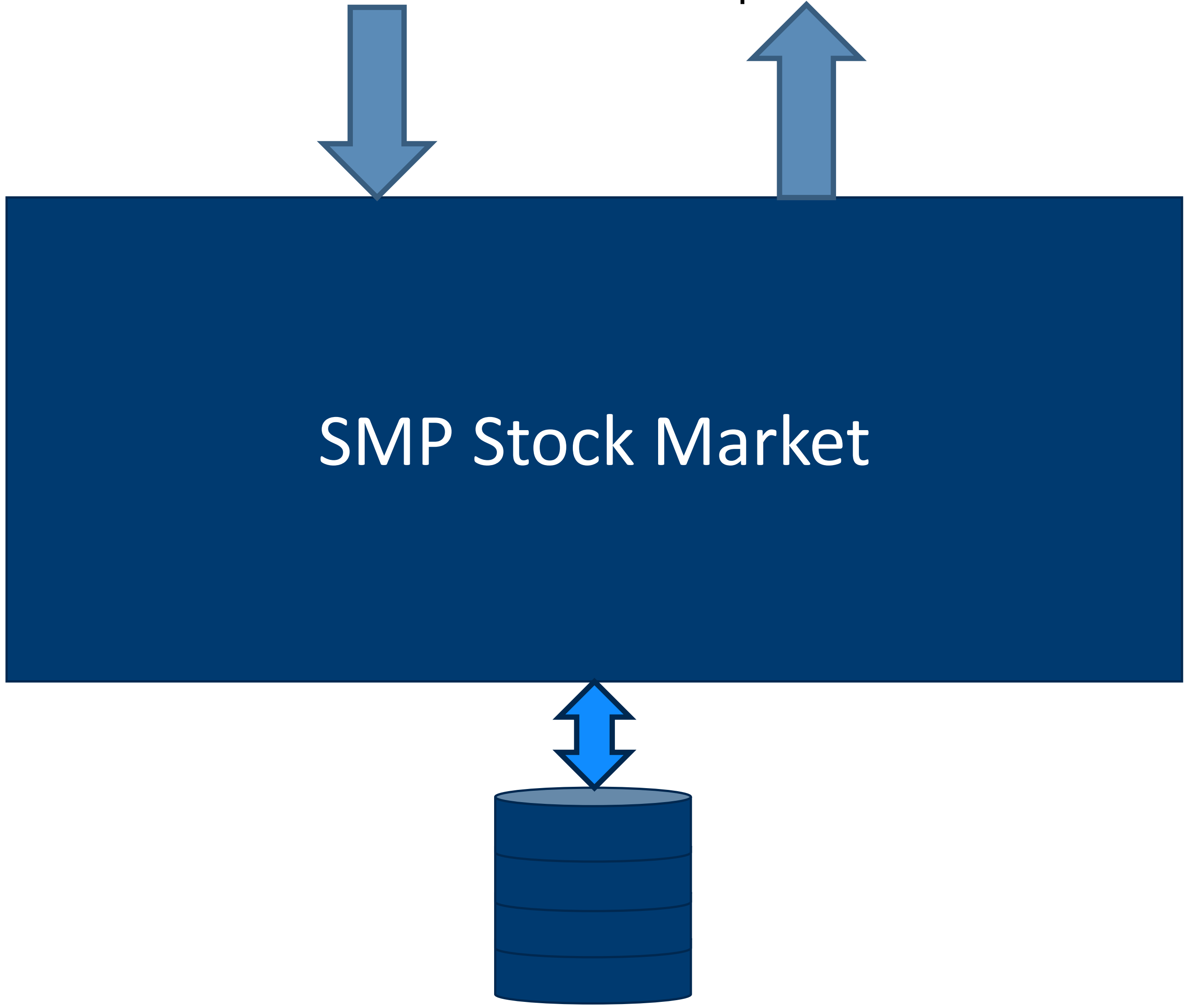
Cloud services make it magic !



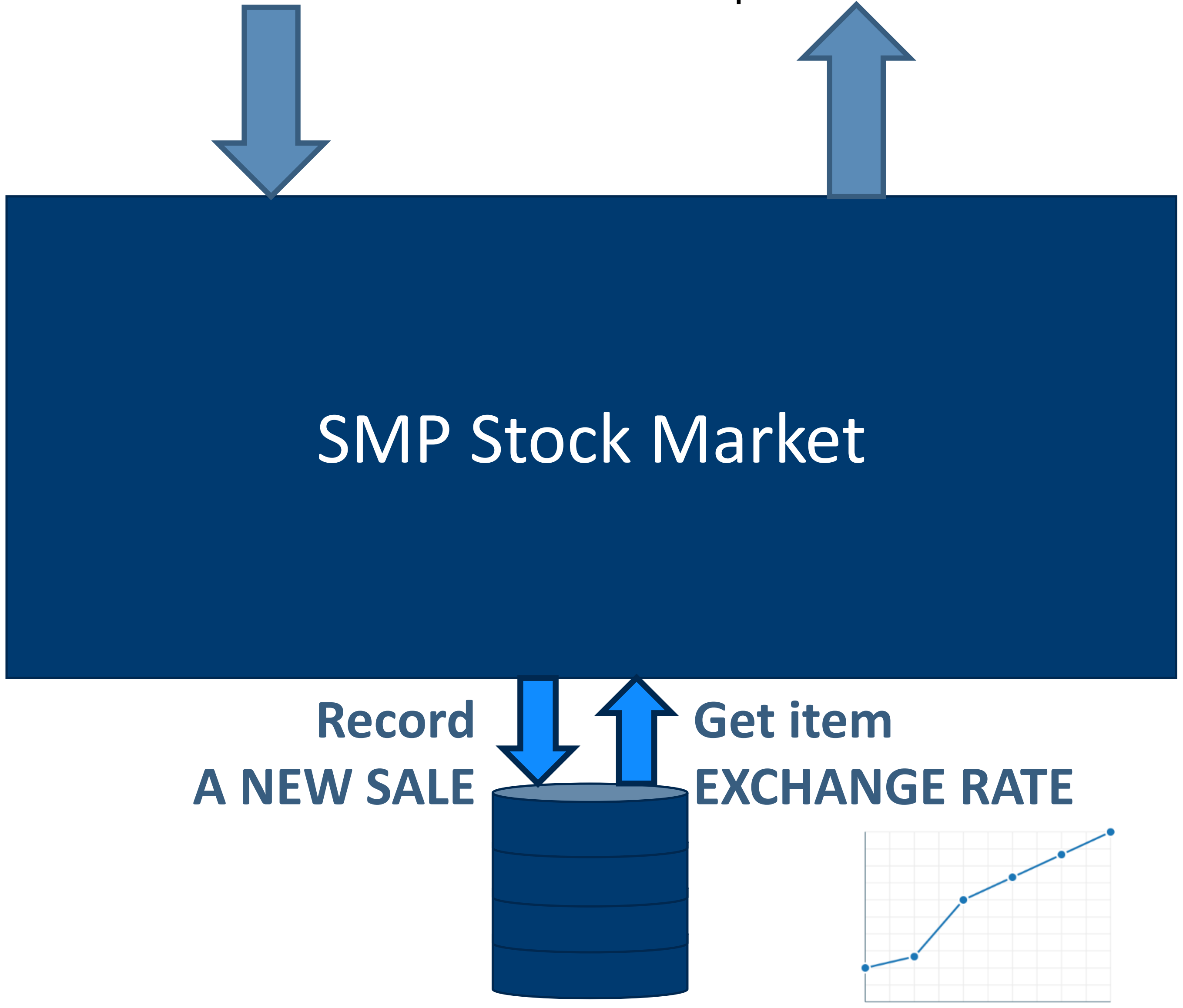
VISEO

Event Based System Design

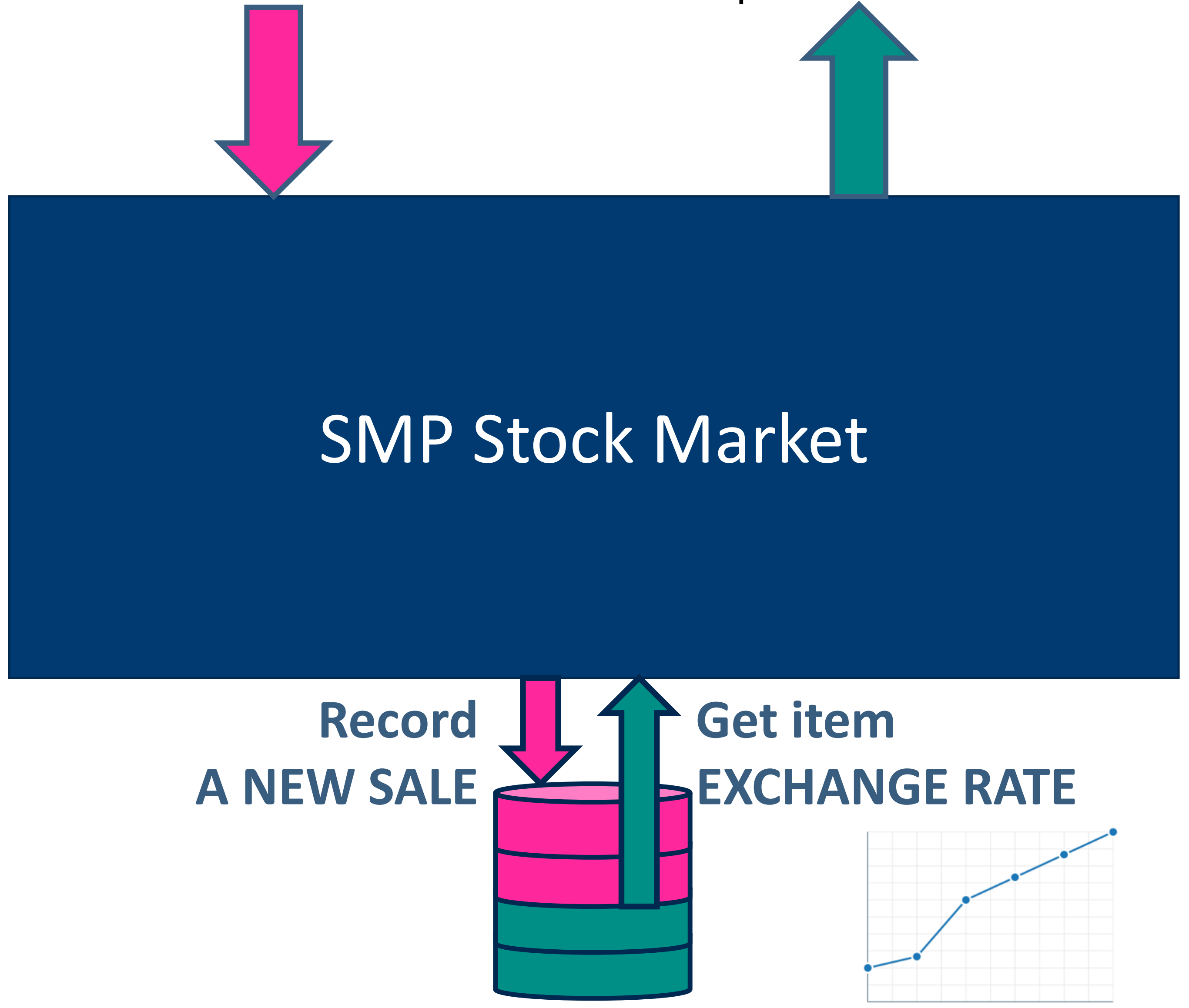
Mismatch Between Read and Write Representations



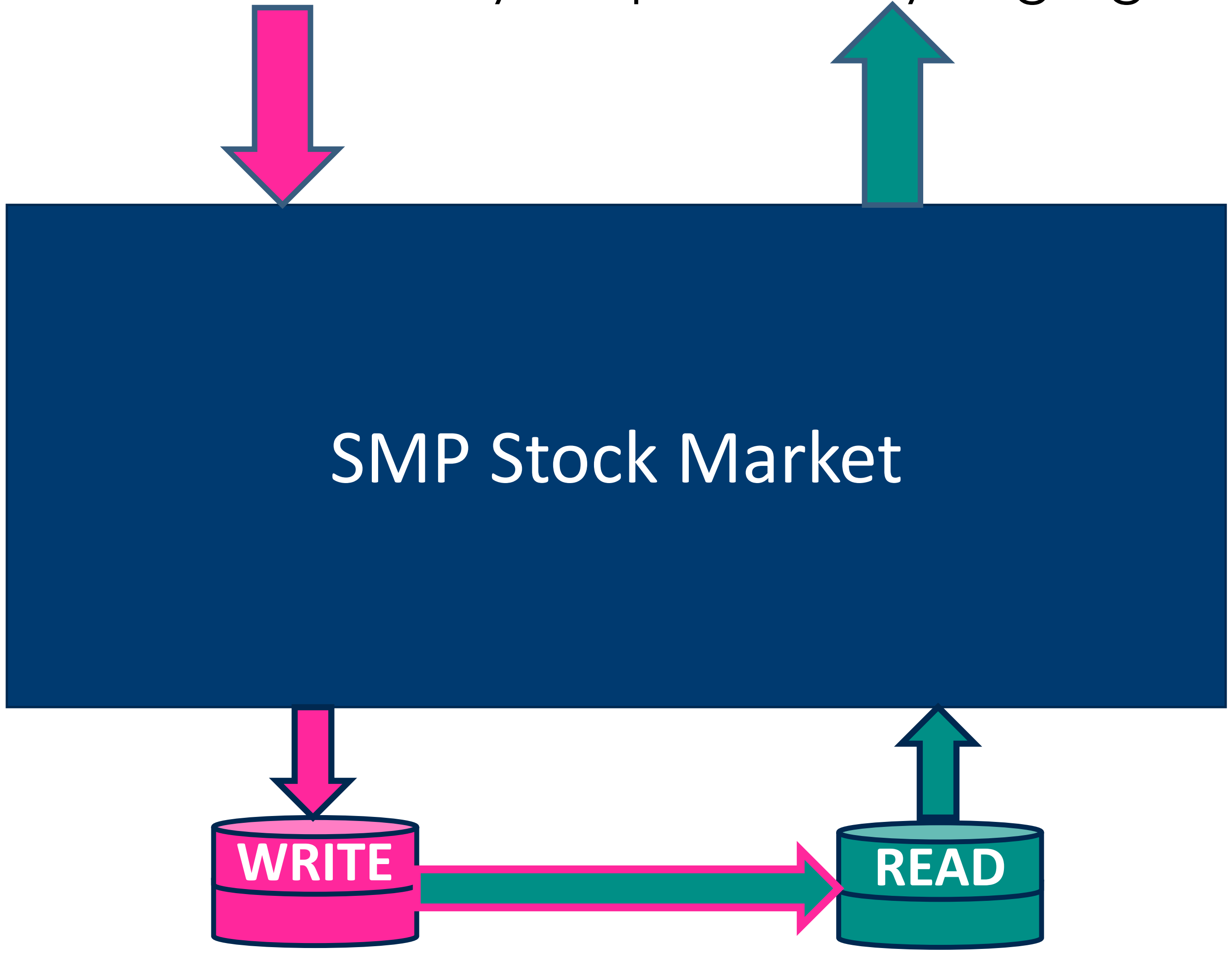
Mismatch Between Read and Write Representations



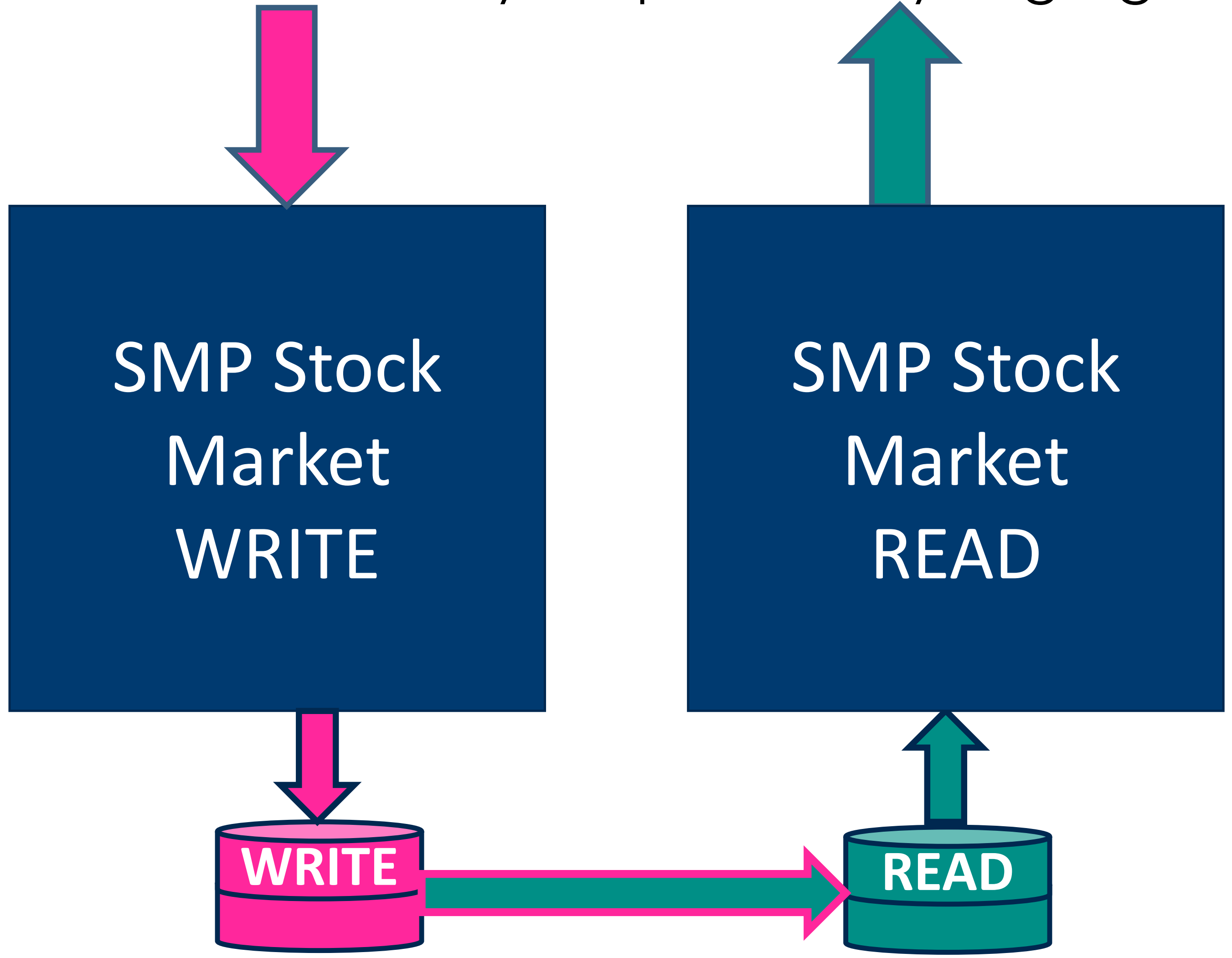
Mismatch Between Read and Write Representations



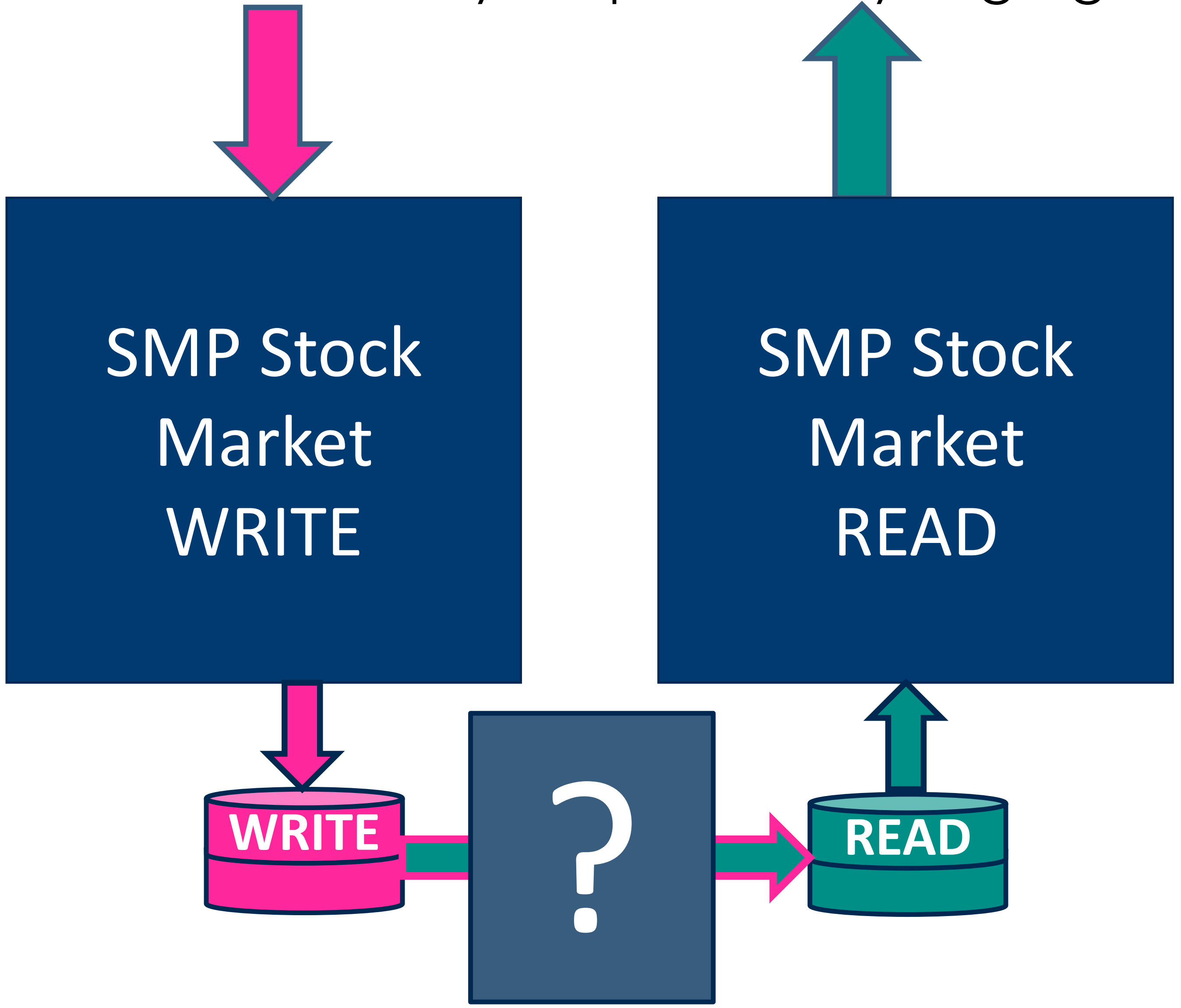
CQRS - Command and Query Responsability Segregation



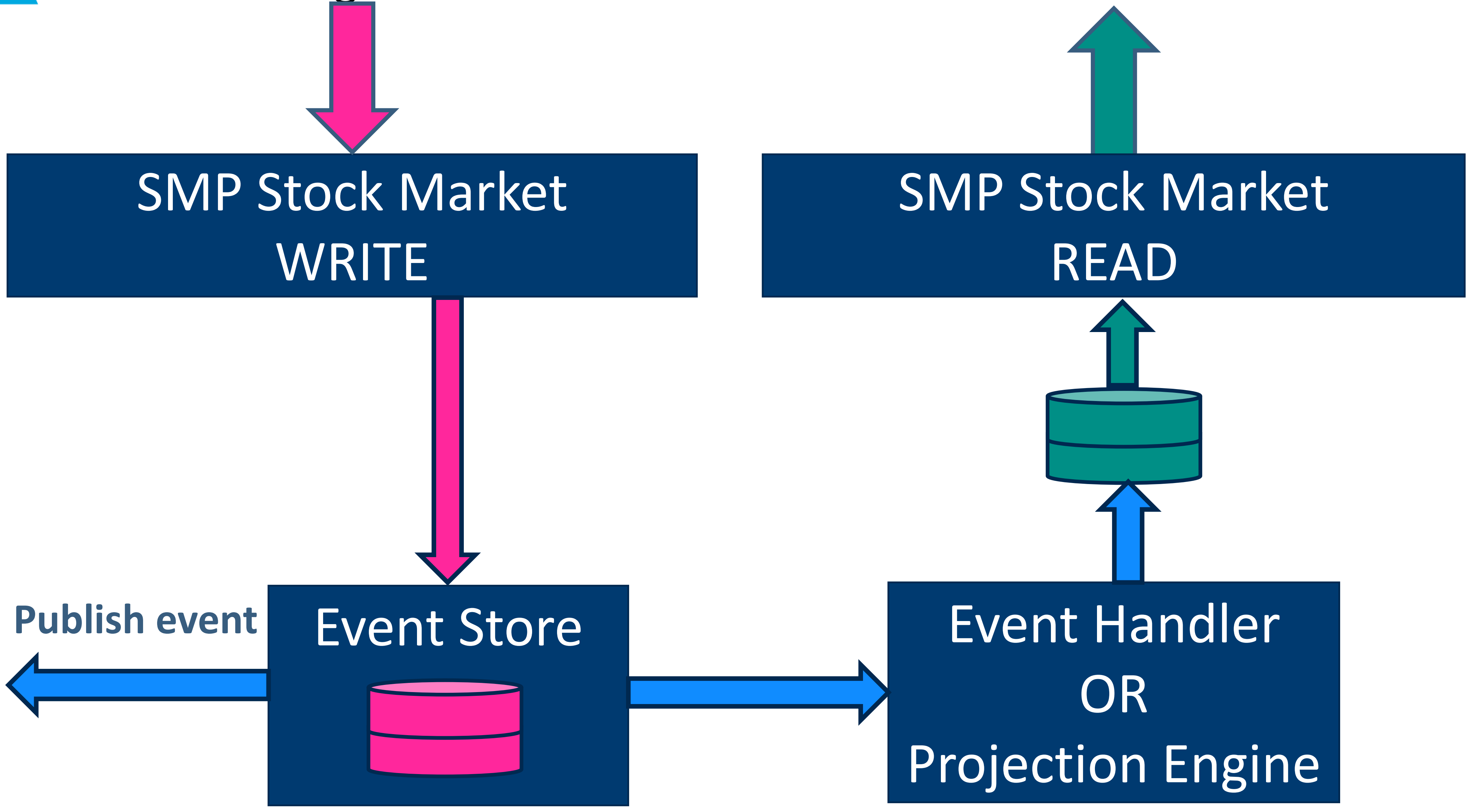
CQRS - Command and Query Responsability Segregation



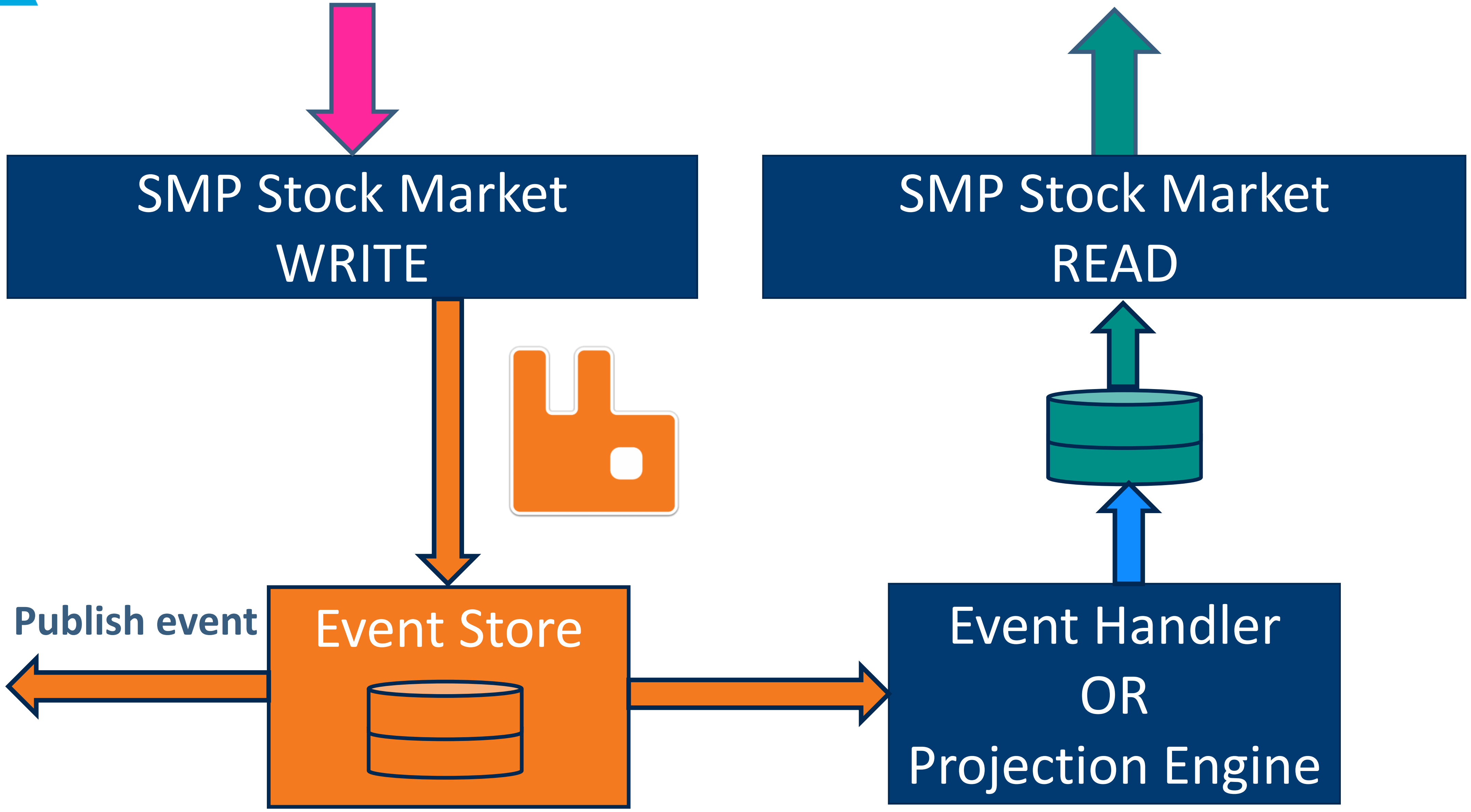
CQRS - Command and Query Responsability Segregation



Event Sourcing to the Rescue

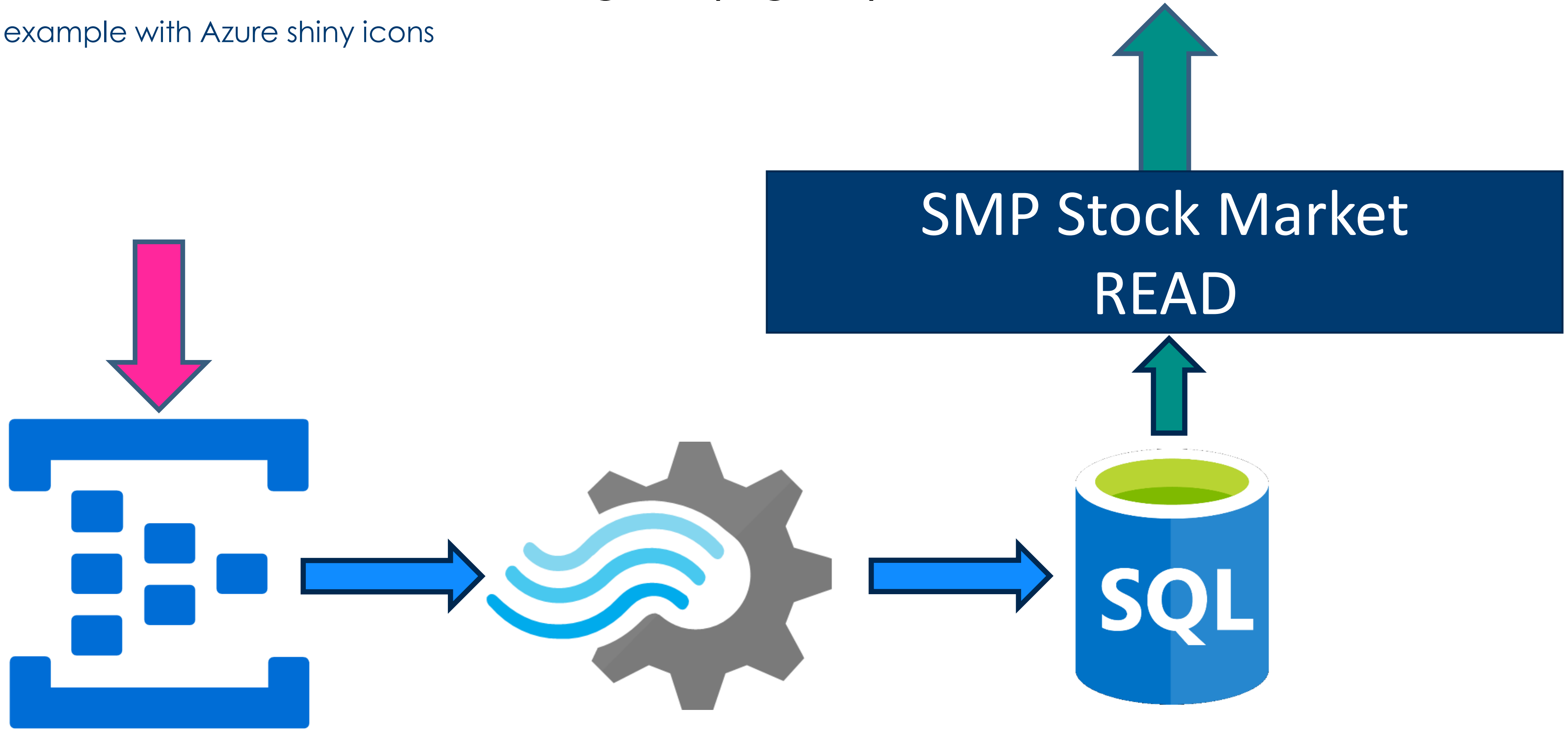


RabbitMQ


SMP Stock Market
WRITE
SMP Stock Market
READ
Publish event
Event Store
Event Handler
OR
Projection Engine

Cloud services make it magic ! (again)

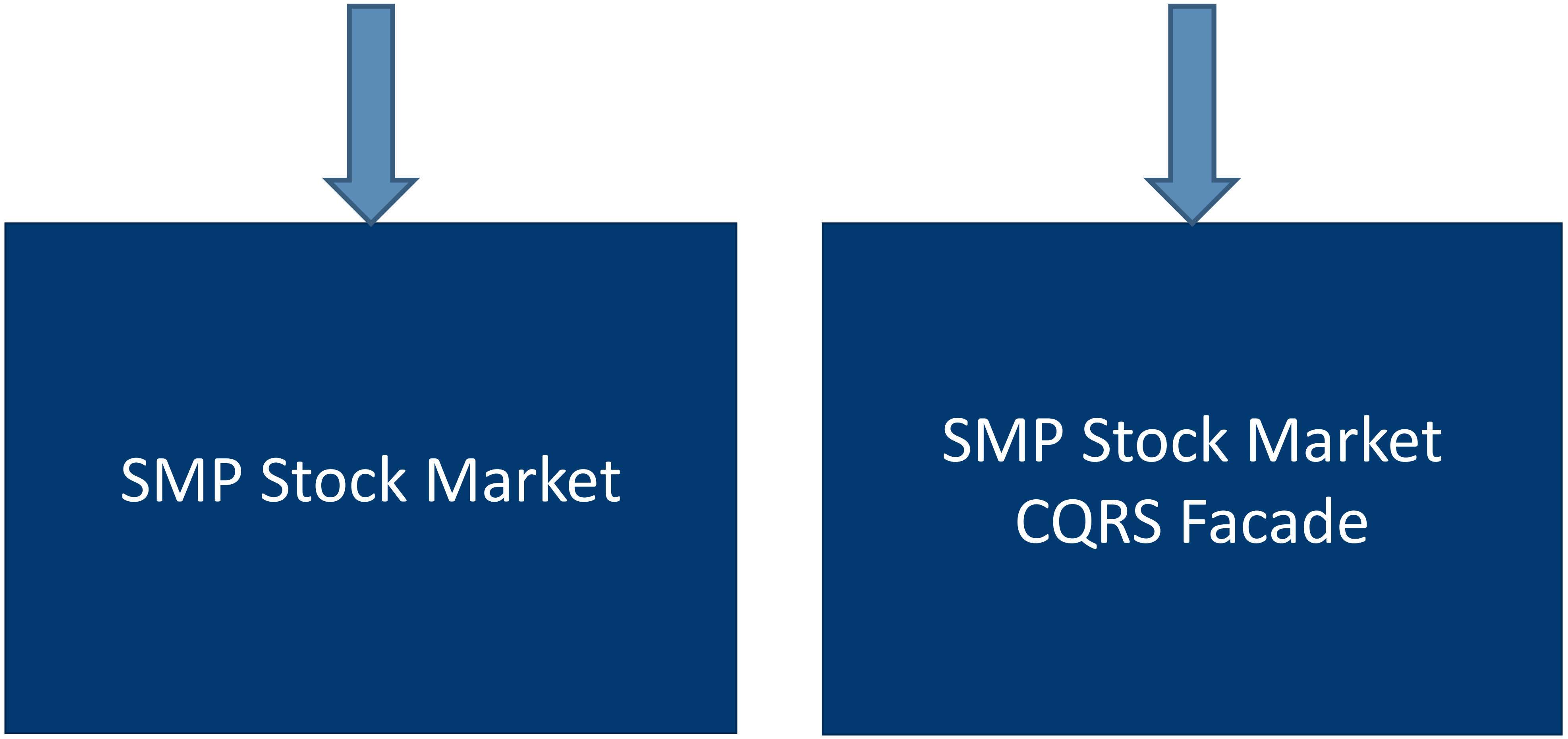
An example with Azure shiny icons



Migrate Legacy Service

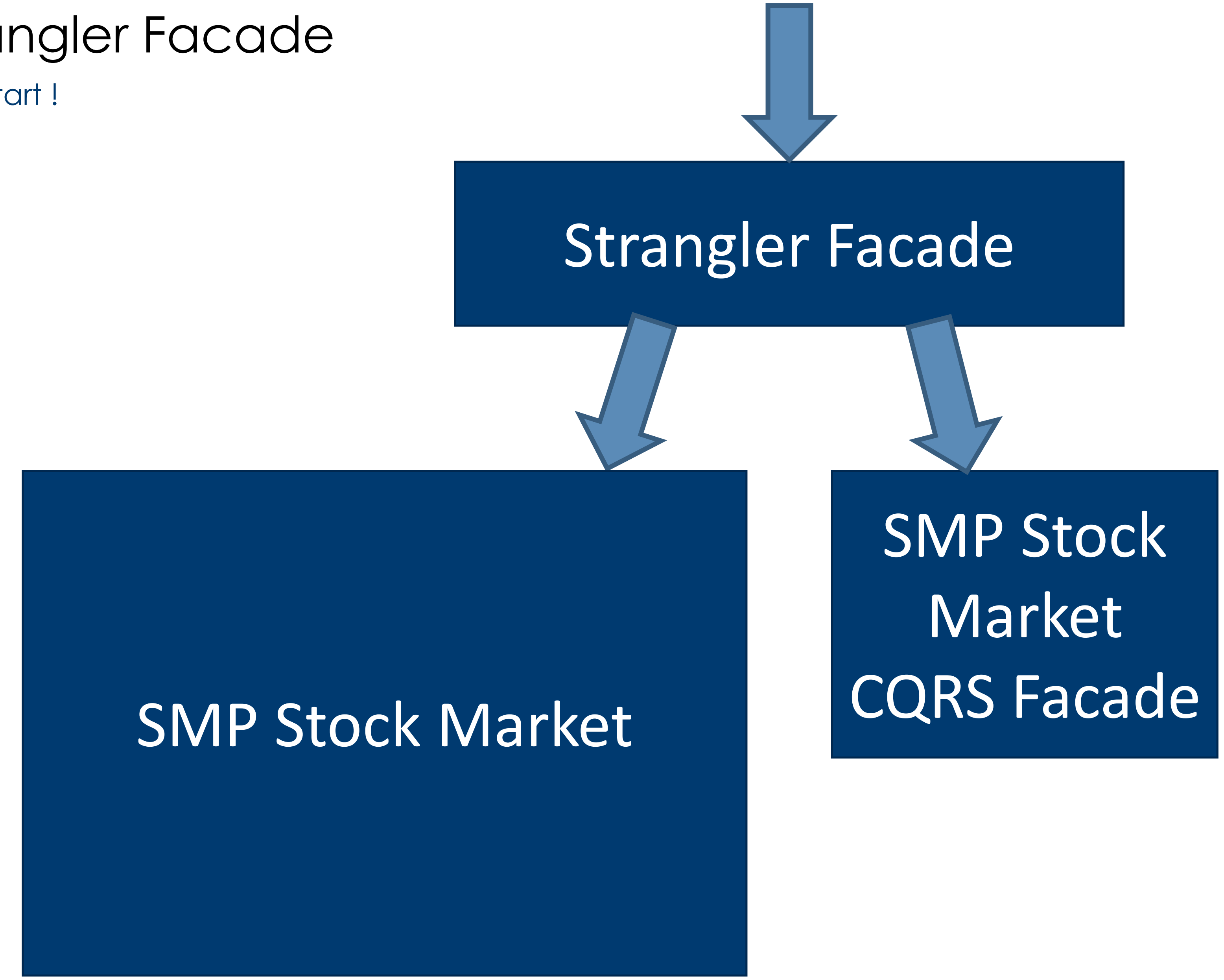
Strangler Facade

Let's start !



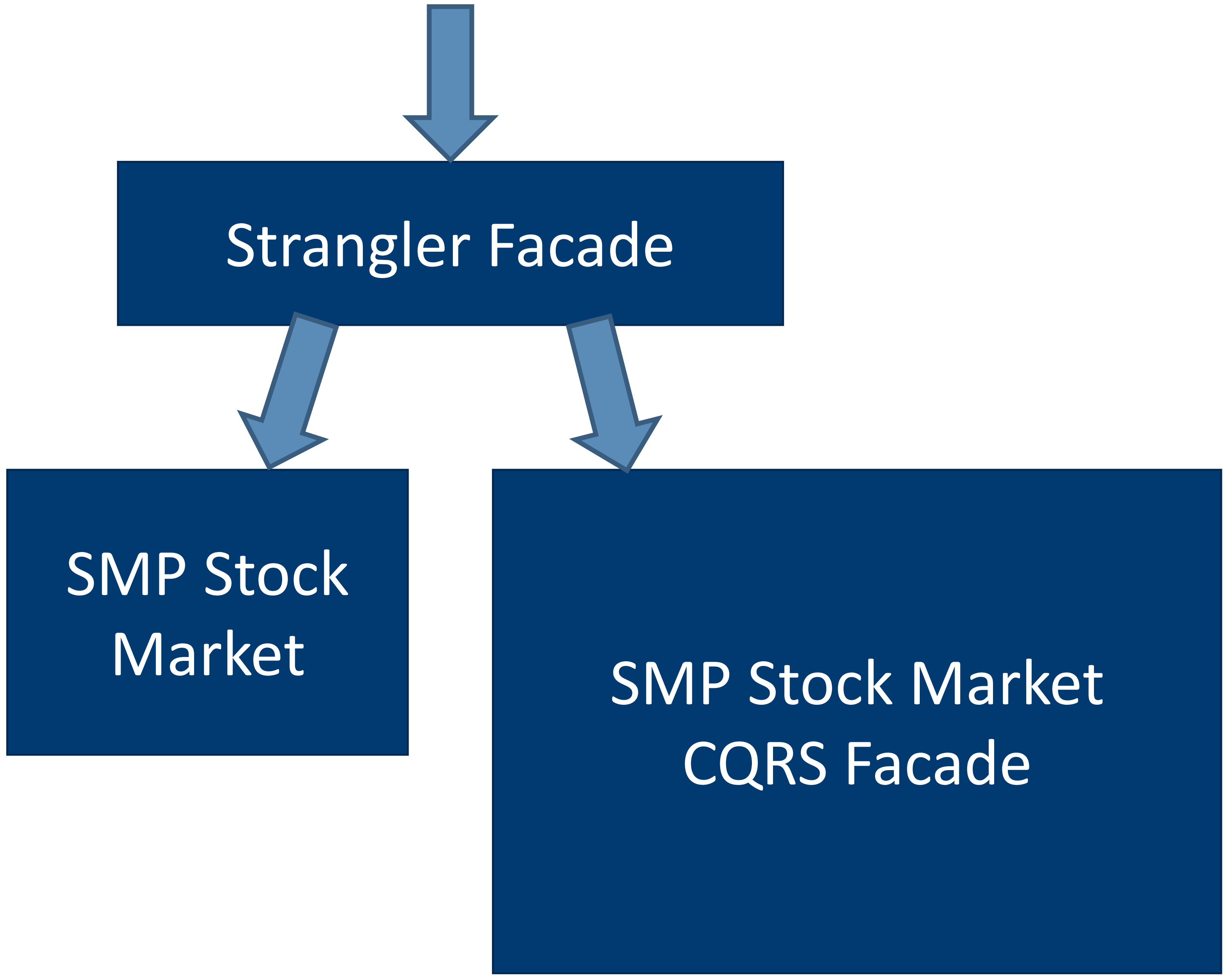
Strangler Facade

Let's start !



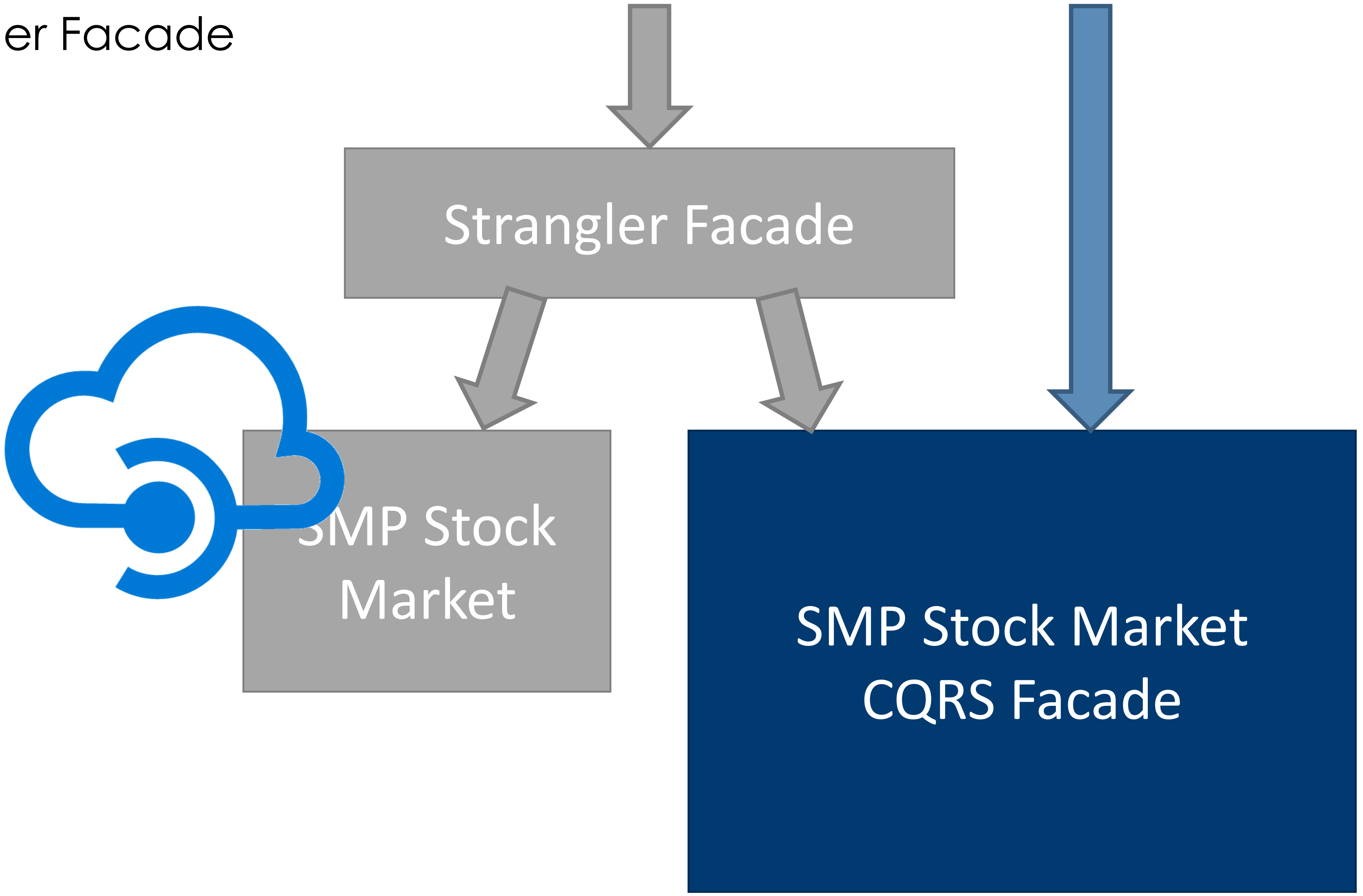
Strangler Facade

Next step



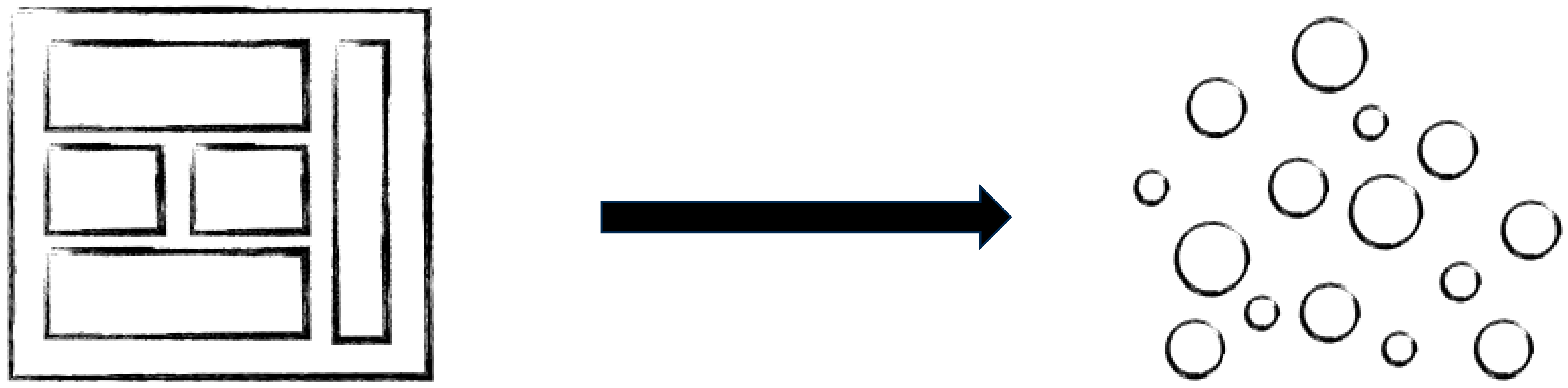
Strangler Facade

Next step



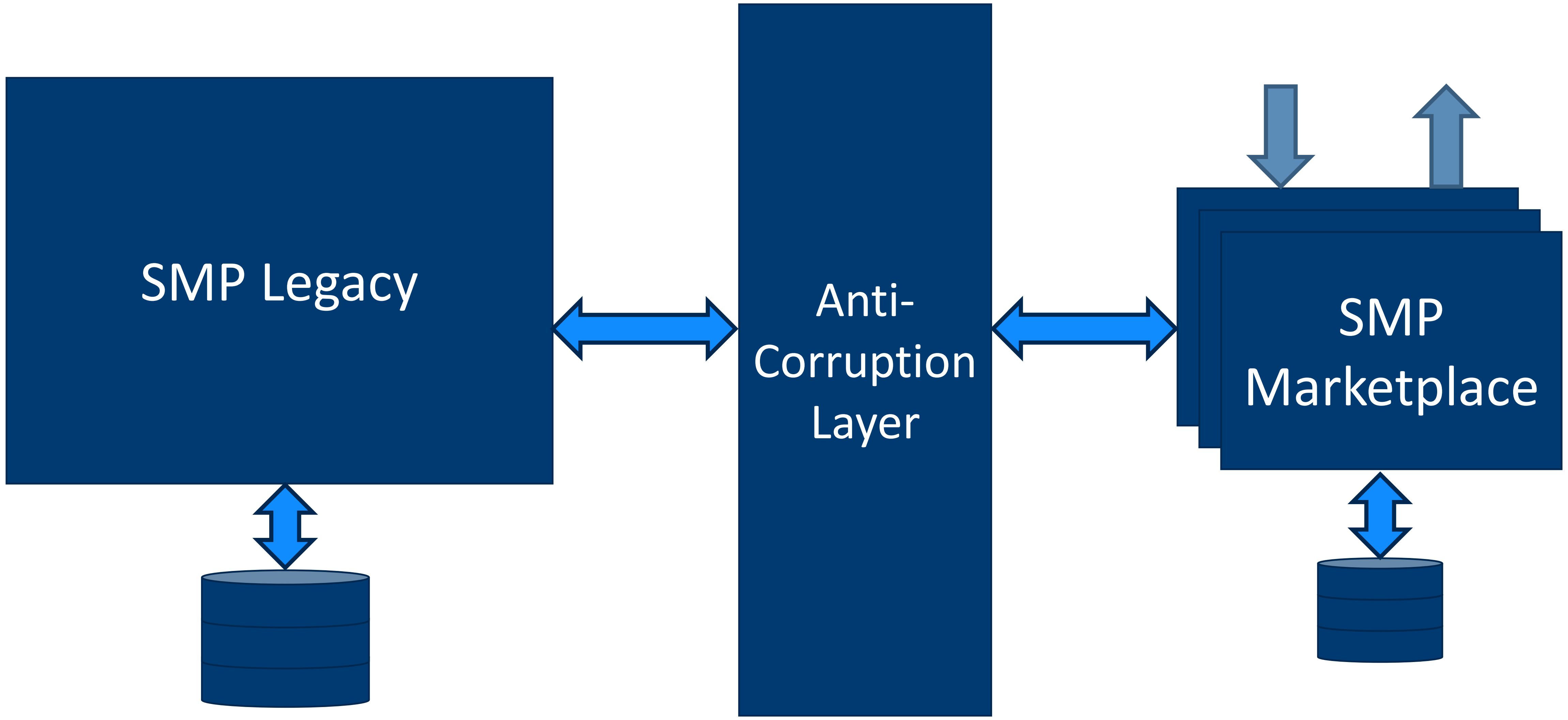
Monolith to Microservices

When new services are highly dependent on legacy…



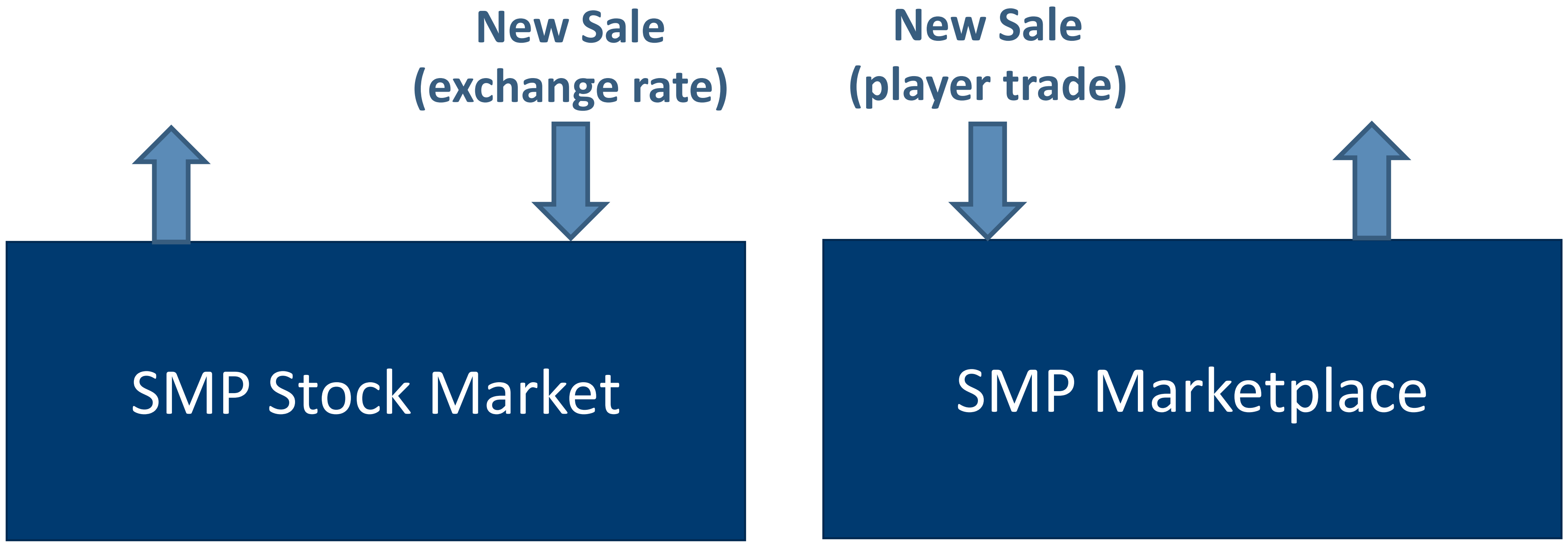
Anti-Corruption Layer

When new services are highly dependent on legacy… And you want to protect them

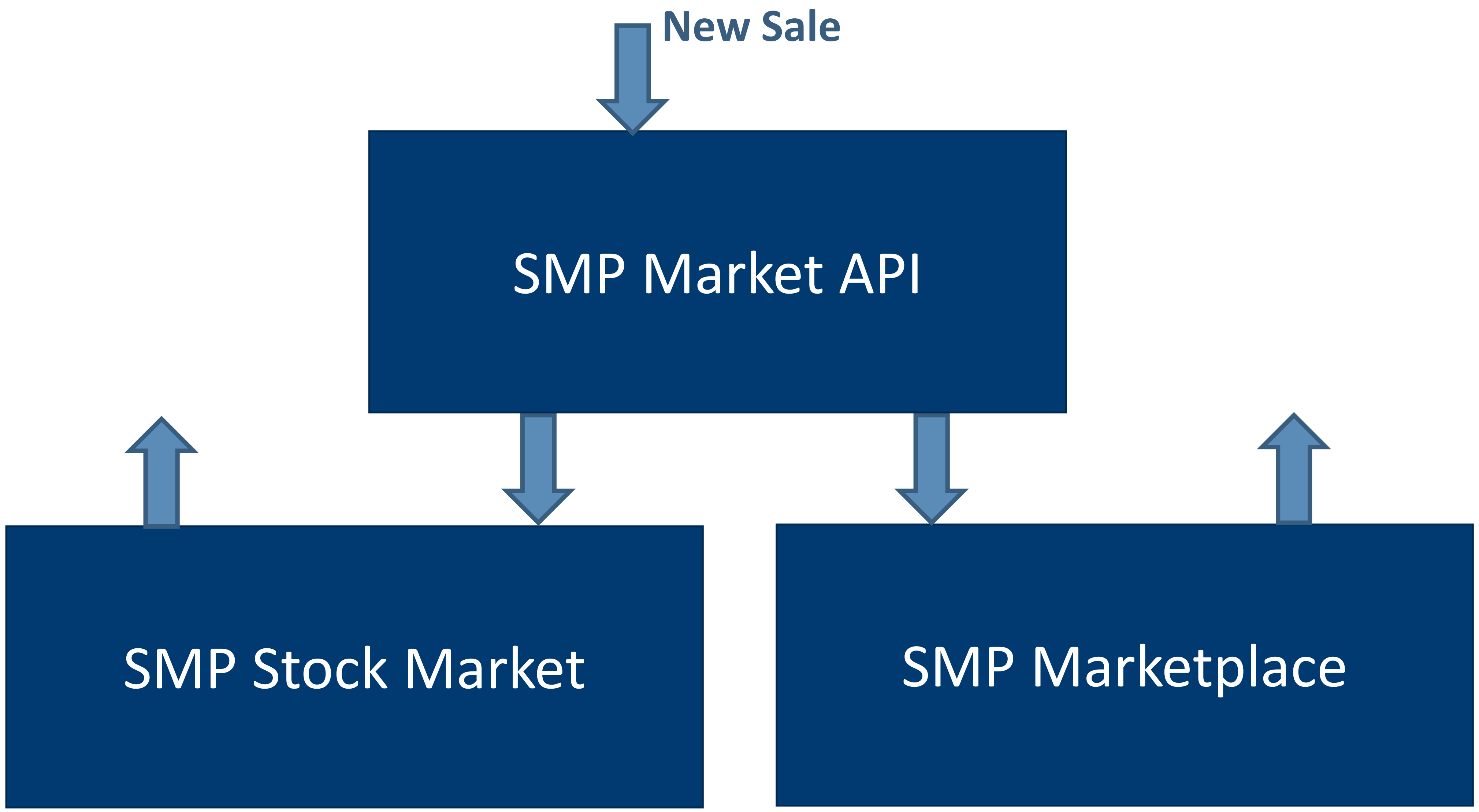


Improve Client-side API

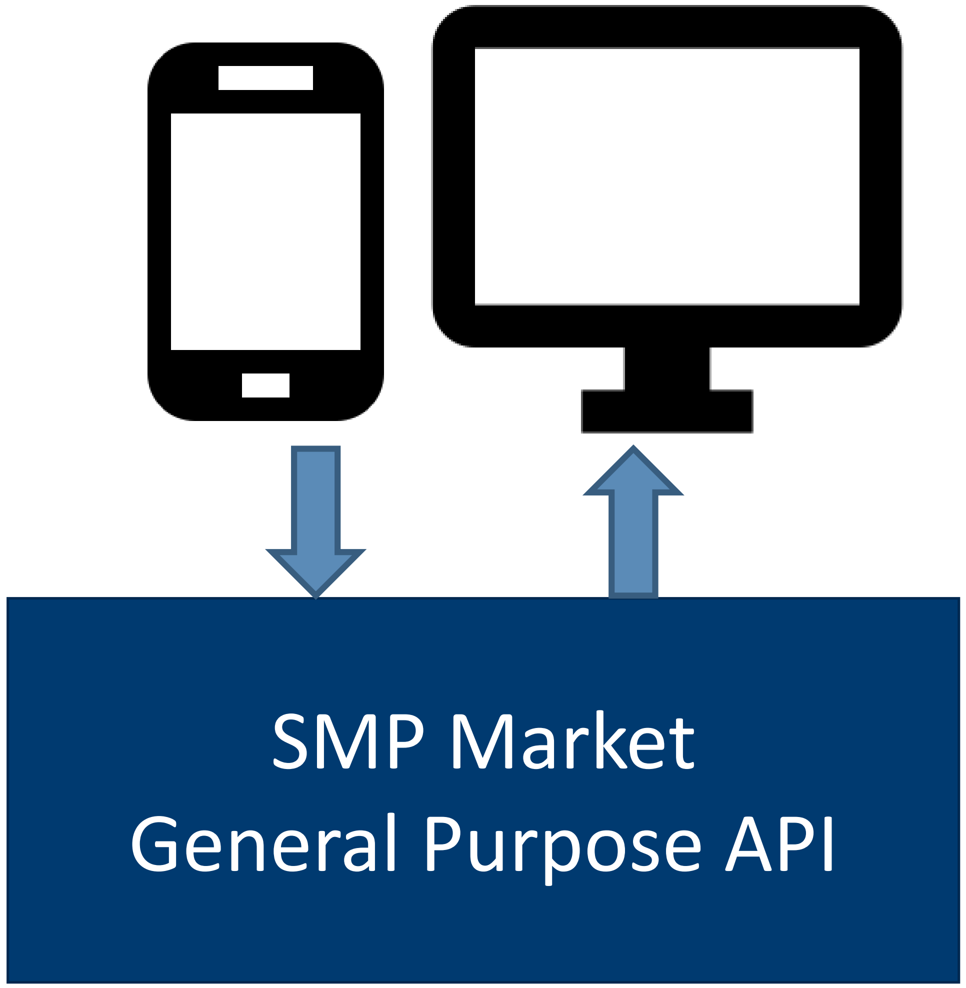
Gateway Aggregation



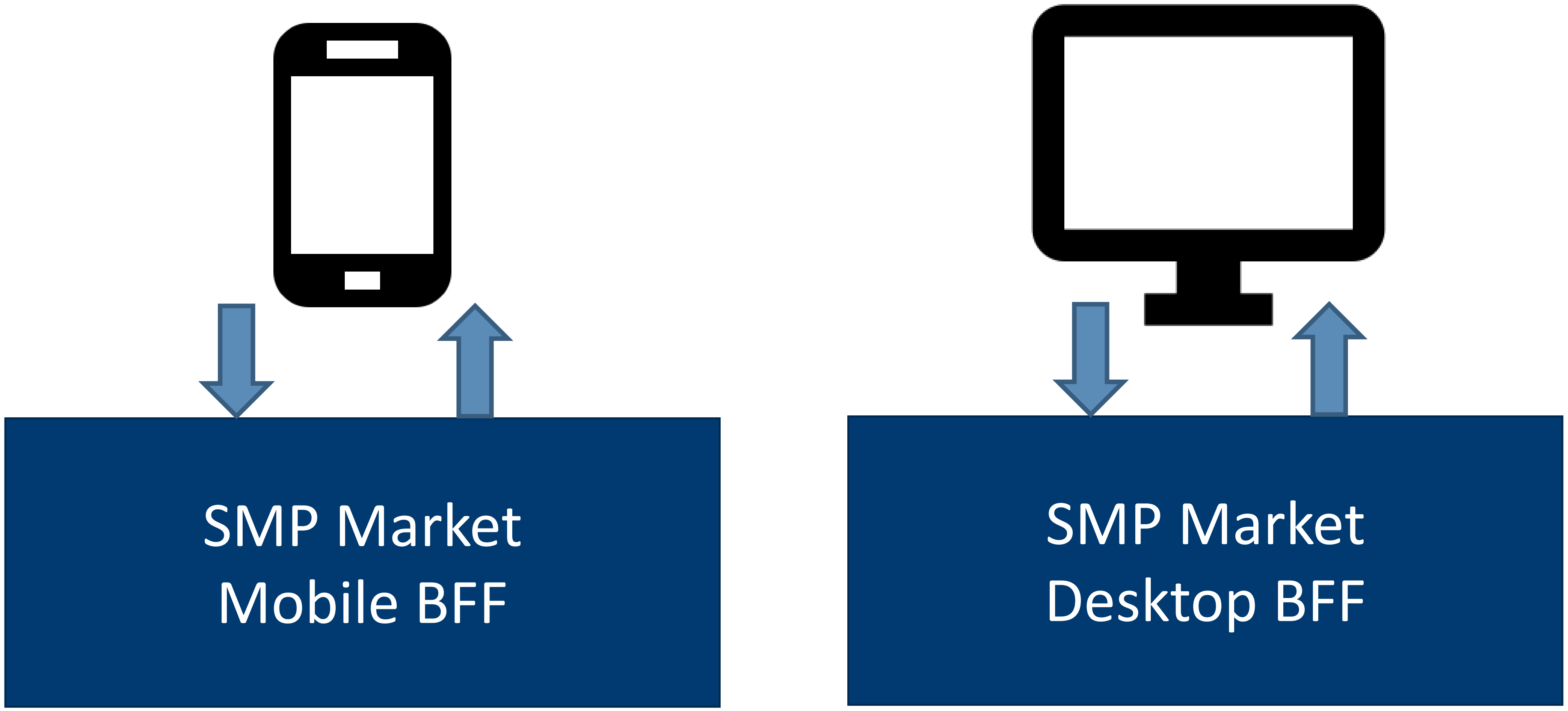
Gateway Aggregation



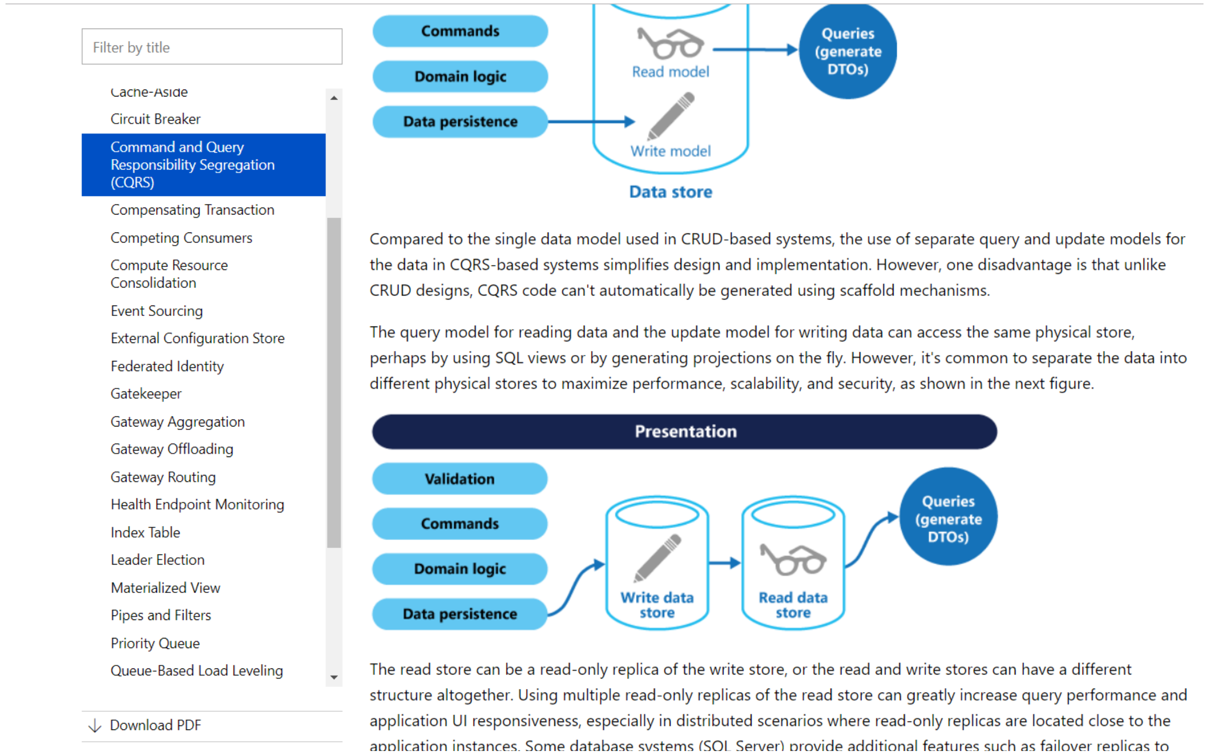
Backend For Frontend



Backend For Frontend

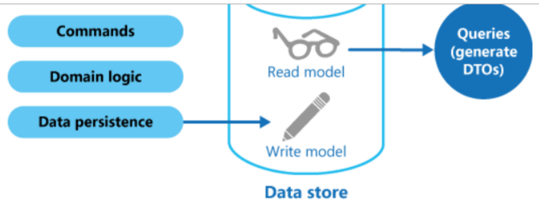


docs.microsoft.com/en-us/azure/architecture/patterns



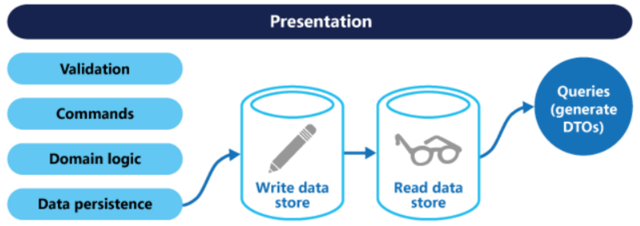
Filter by title

Download PDF



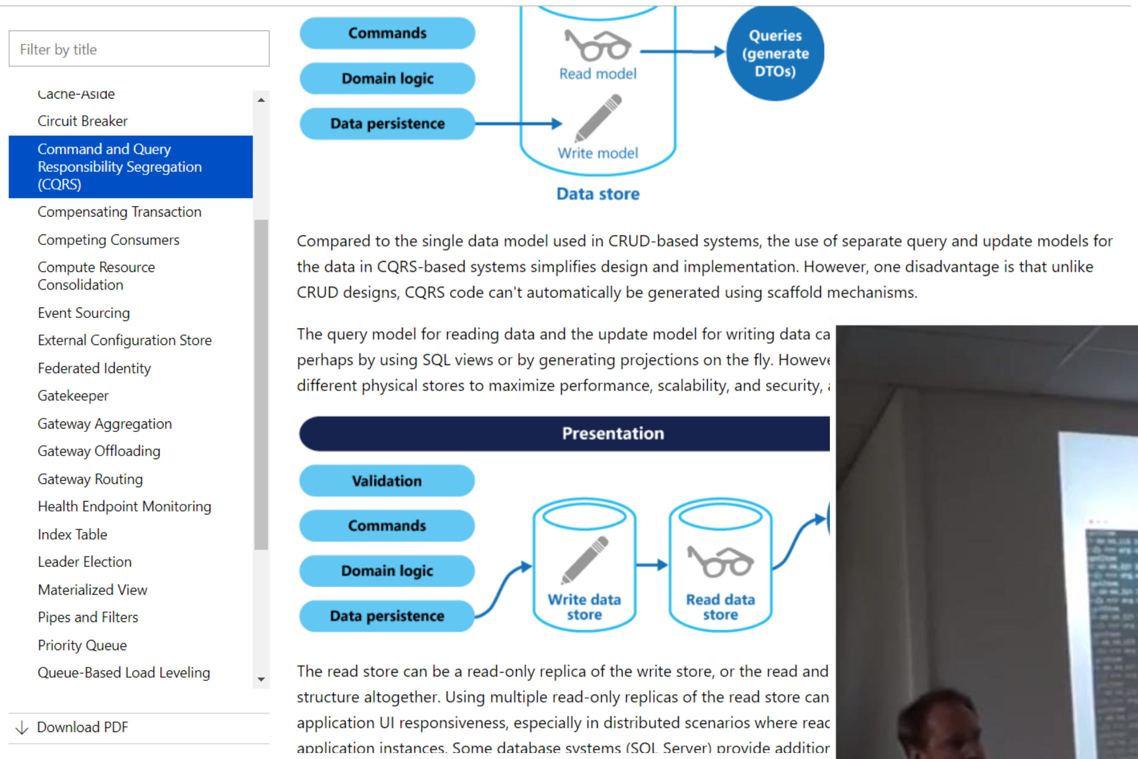
Compared to the single data model used in CRUD-based systems, the use of separate query and update models for the data in CQRS-based systems simplifies design and implementation. However, one disadvantage is that unlike CRUD designs, CQRS code can't automatically be generated using scaffold mechanisms.

The query model for reading data and the update model for writing data can access the same physical store, perhaps by using SQL views or by generating projections on the fly. However, it's common to separate the data into different physical stores to maximize performance, scalability, and security, as shown in the next figure.



The read store can be a read-only replica of the write store, or the read and write stores can have a different structure altogether. Using multiple read-only replicas of the read store can greatly increase query performance and application UI responsiveness, especially in distributed scenarios where read-only replicas are located close to the application instances. Some database systems (SQL Server) provide additional features such as failover replicas to

docs.microsoft.com/en-us/azure/architecture/patterns



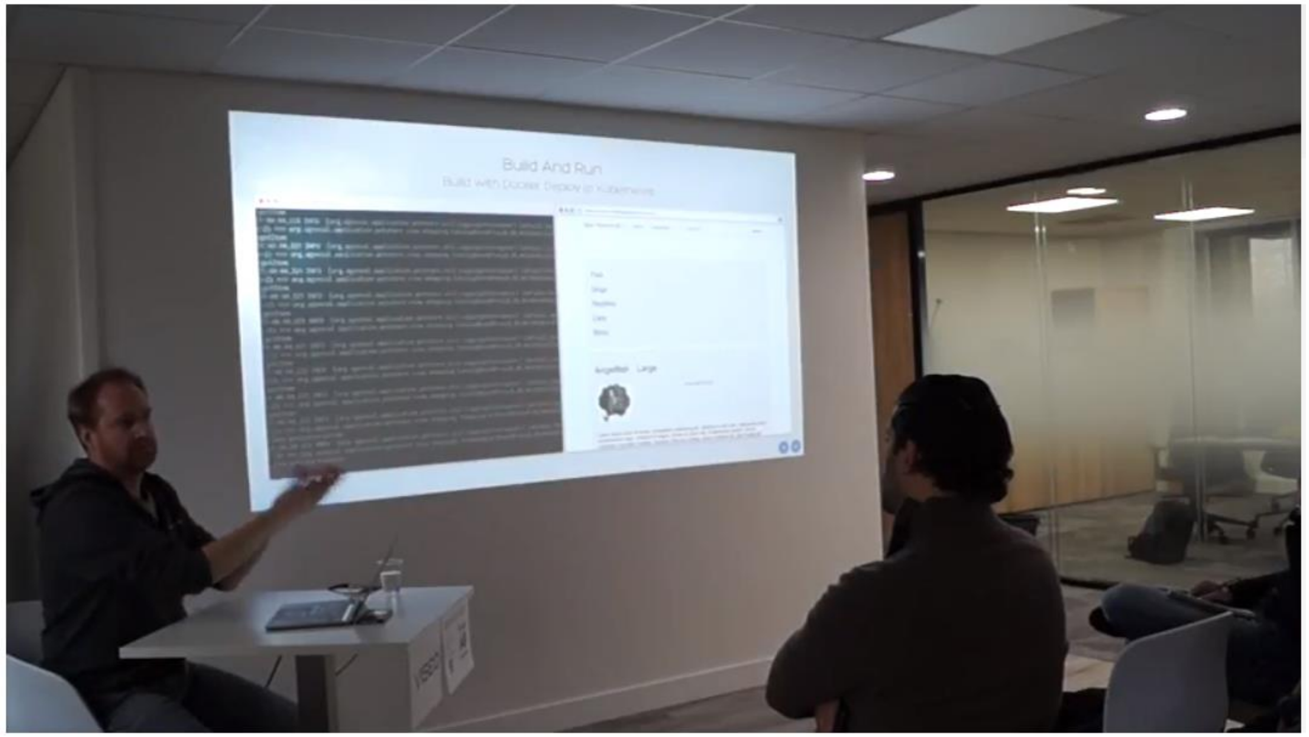
Filter by title

Download PDF

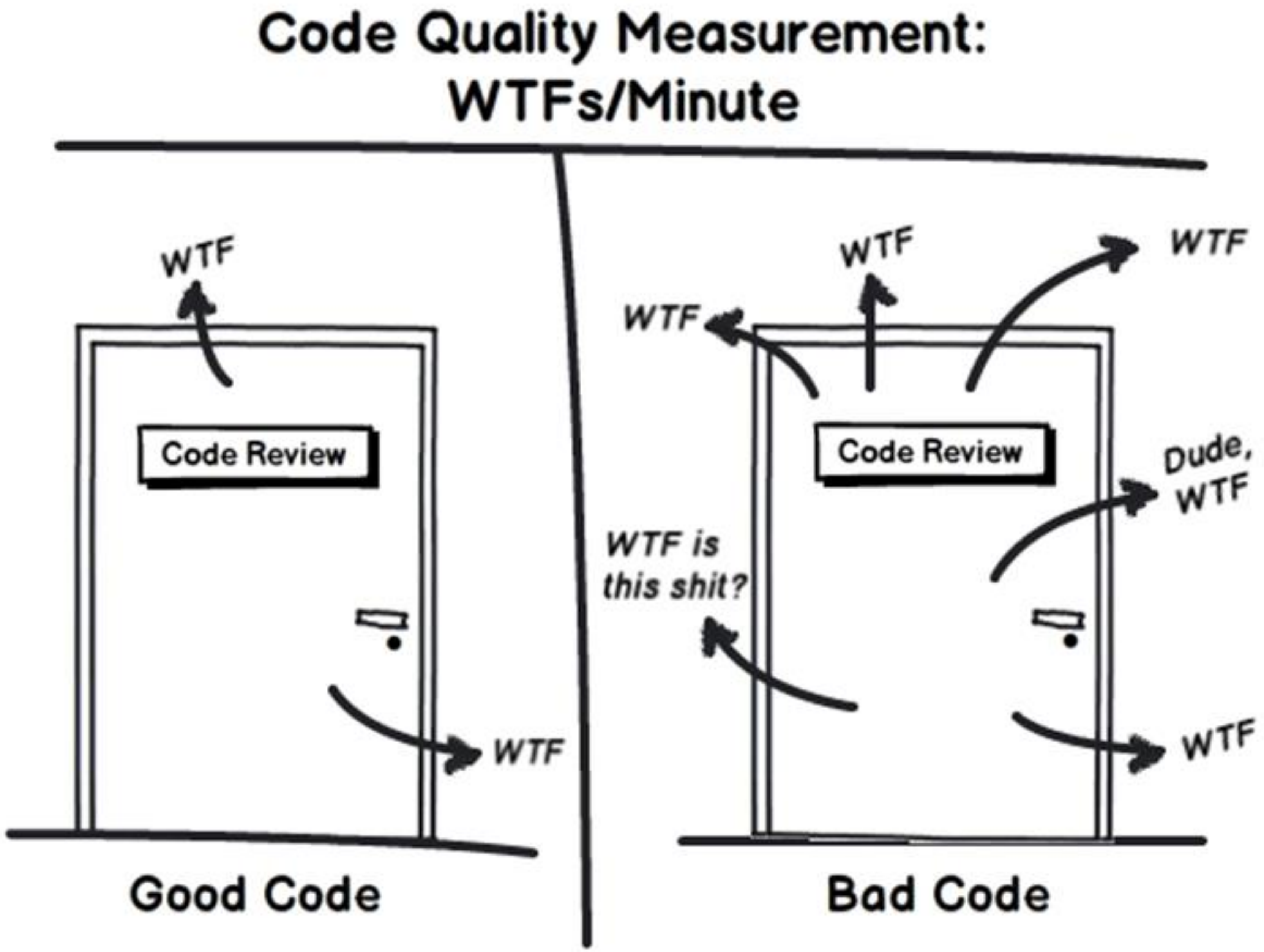
Compared to the single data model used in CRUD-based systems, the use of separate query and update models for the data in CQRS-based systems simplifies design and implementation. However, one disadvantage is that unlike CRUD designs, CQRS code can't automatically be generated using scaffold mechanisms.

The query model for reading data and the update model for writing data ca perhaps by using SQL views or by generating projections on the fly. Howeve different physical stores to maximize performance, scalability, and security,

The read store can be a read-only replica of the write store, or the read and structure altogether. Using multiple read-only replicas of the read store can application UI responsiveness, especially in distributed scenarios where reac application instances. Some database systems (SQL Server) provide additior



Transform a legacy application with Kubernetes & Istio (by David Gageot, Google)


Code Quality Measurement:
WTFs/Minute
WTF
Code Review
WTF
Good Code
WTF
WTF
WTF
Code Review
Dude, WTF
WTF is this shit?
WTF
Bad Code

Anti-CorruptionLayer

GatewayOffloading

GatewayAggregation

MicroServices

LoadBalancer

APIGateway

CompetingConsumer

CircuitBreaker

StranglerFacade

CQRS

EventSourcing

Sidecar

BulkRequest

BackendForFrontend

Michel Barret

Pierrick Rassat

Q/A : How to implement business transactions ? "Distributed Sagas" pattern to the rescue

https://www.youtube.com/watch?v=0UTOLRTwOX0