Particular
Software

Microservices with Service Fabric

Easy... or is it?

Goals

Stateless and Stateful Services

Partitioning of Business Data

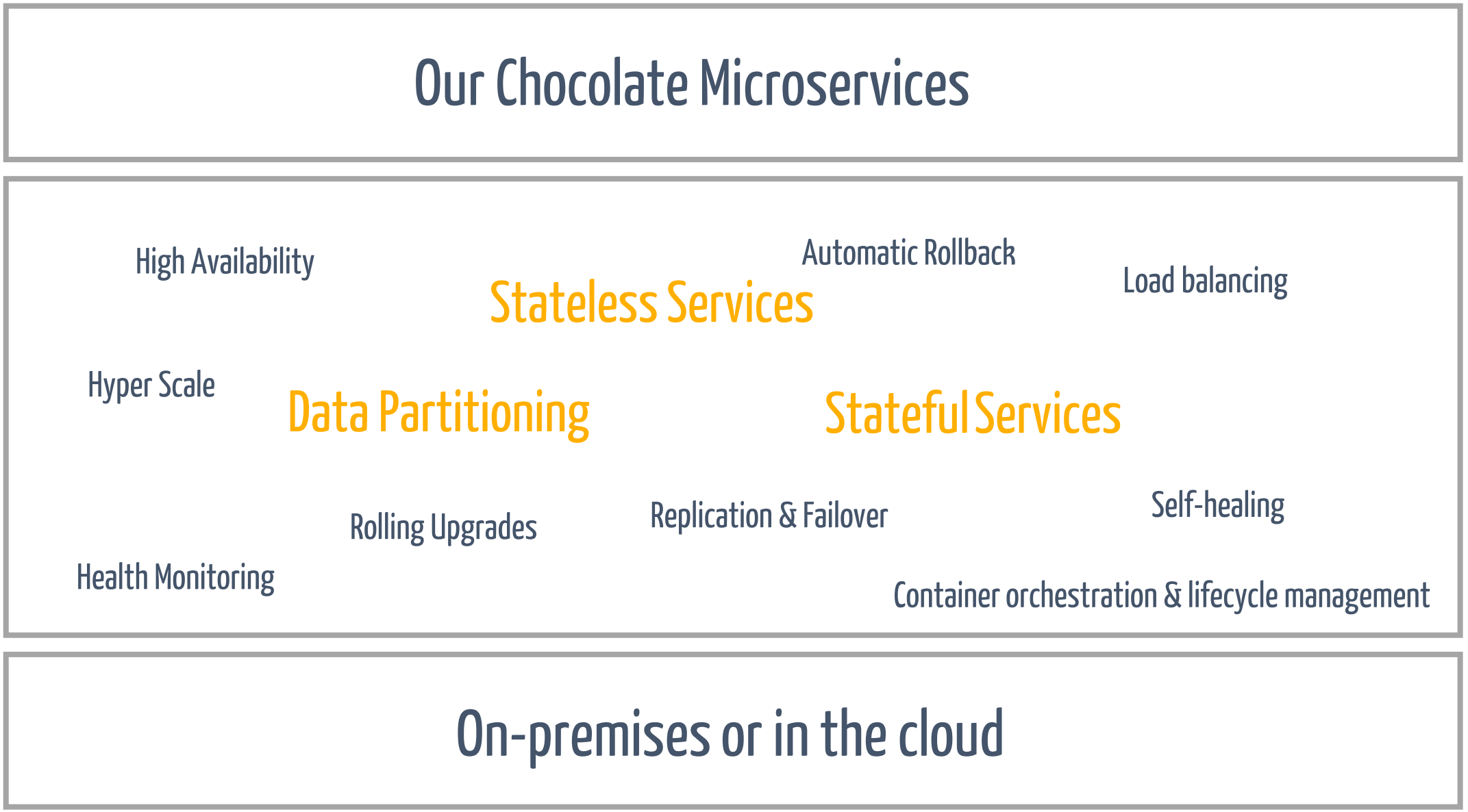
Message Patterns and Partitioning

Premise

A Tale of chocolate

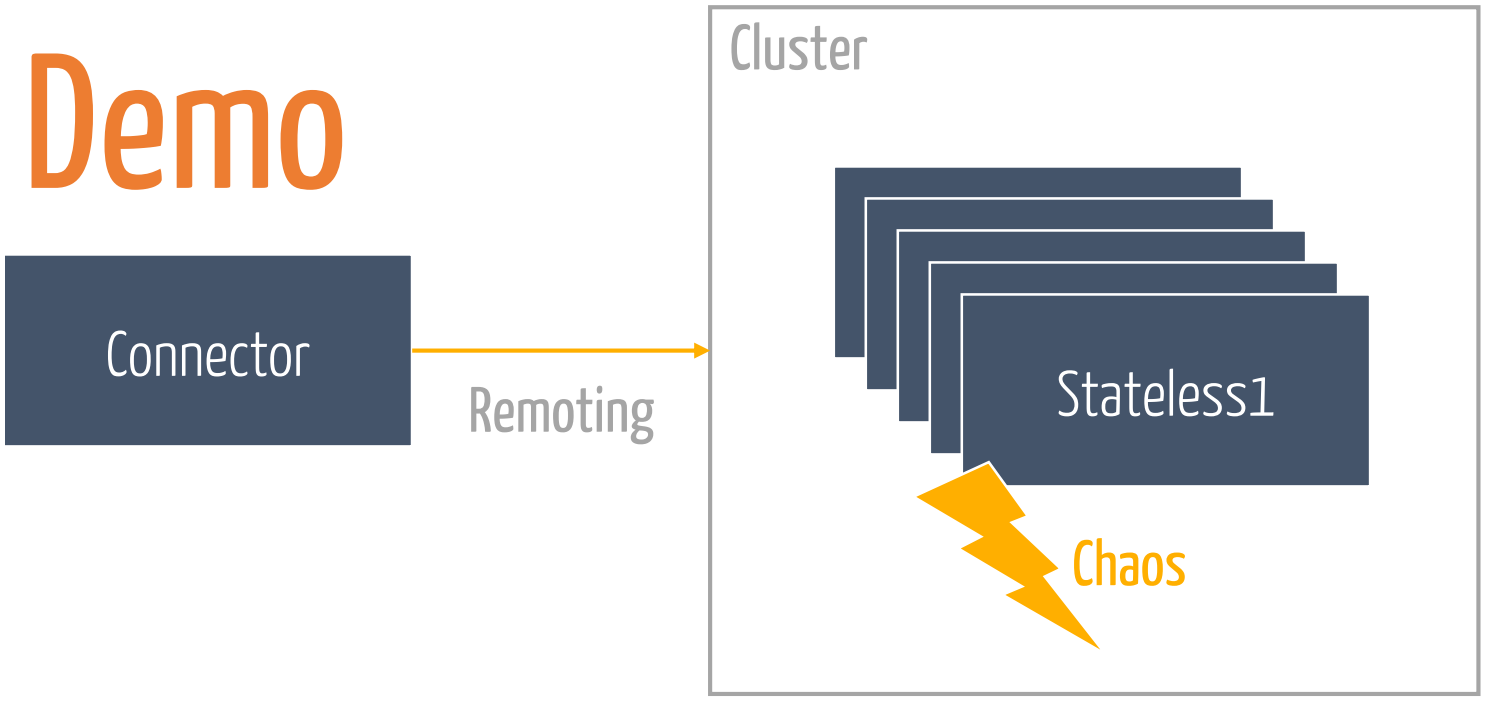
Horrible death of easter bunnies

Karl's, Sales Pitch



https://channel9.msdn.com/Blogs/Azure/Azure-Service-Fabric
https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-overview

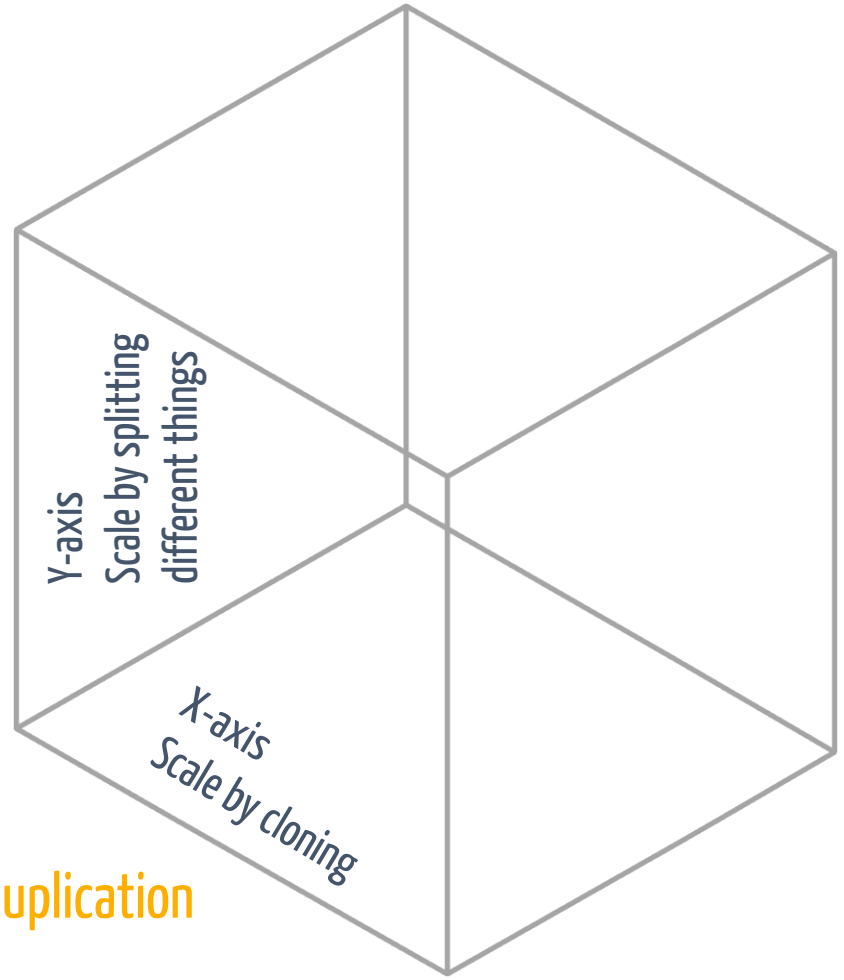
Demo



Demo

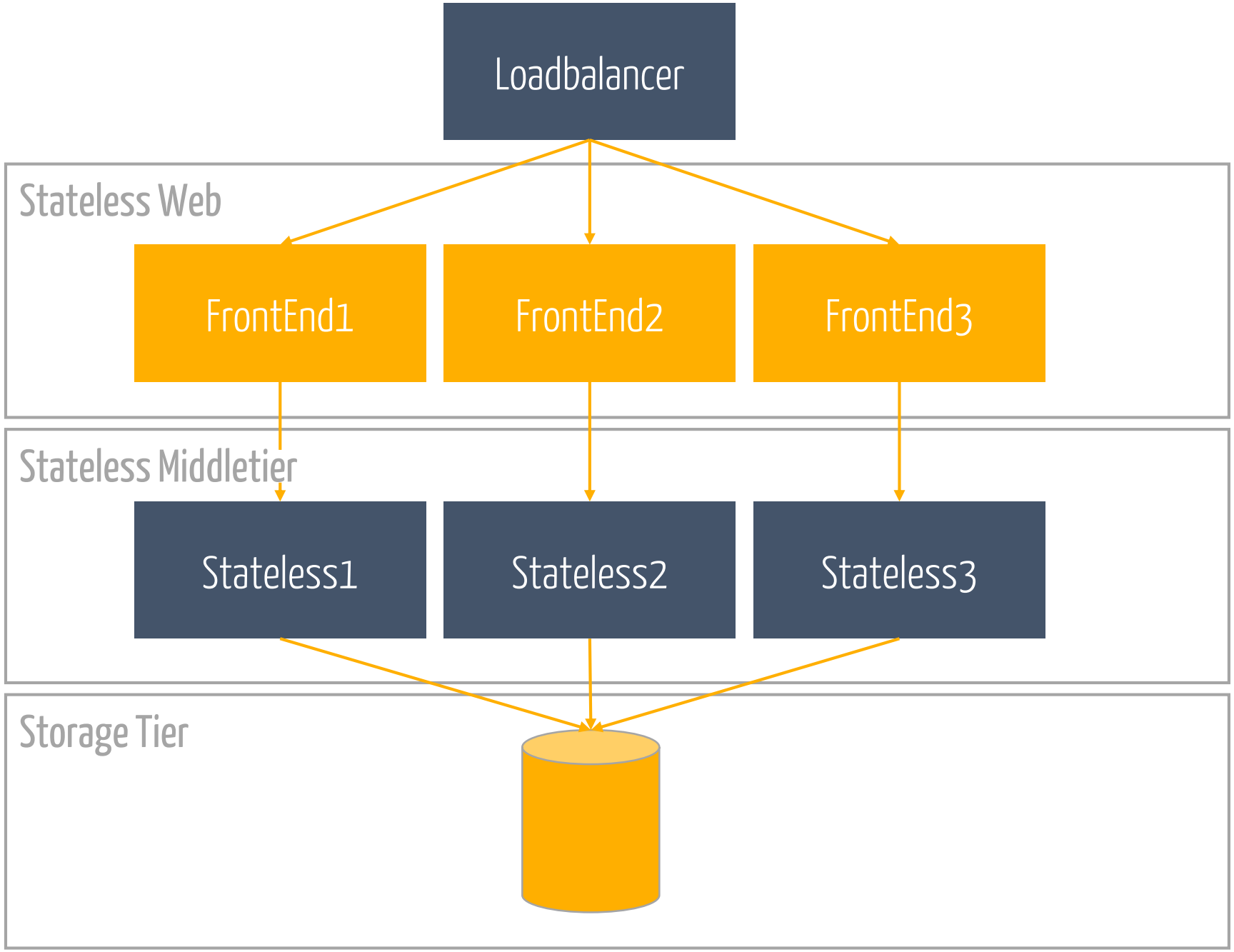
Let's scale chocolate

Functional decomposition



Horizontal duplication

http://microservices.io/articles/scalecube.html


Loadbalancer
Stateless Web
FrontEnd1
FrontEnd2
FrontEnd3
Stateless Middletier
Stateless1
Stateless2
Stateless3
Storage Tier

and then

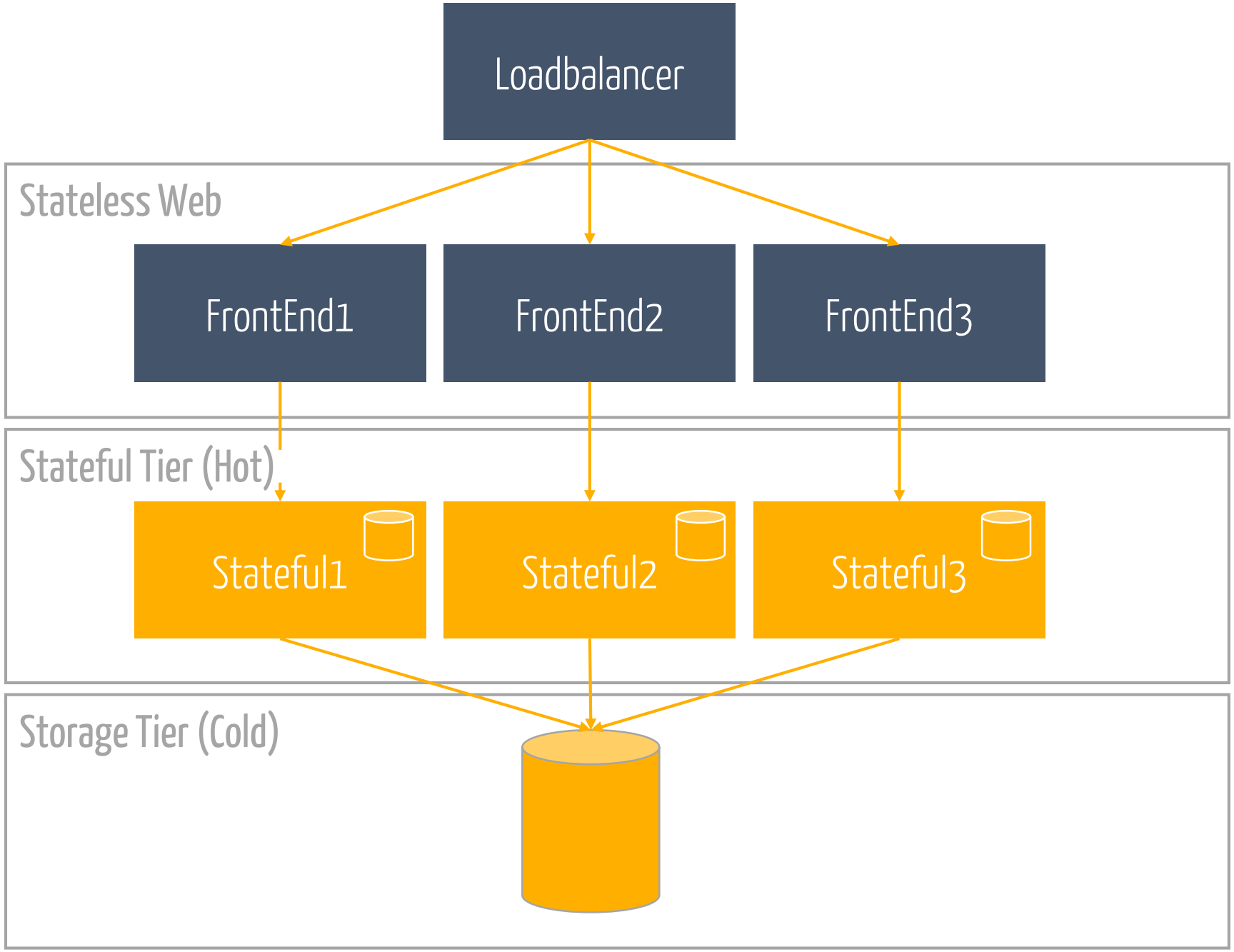
Mandy spoke up

and then

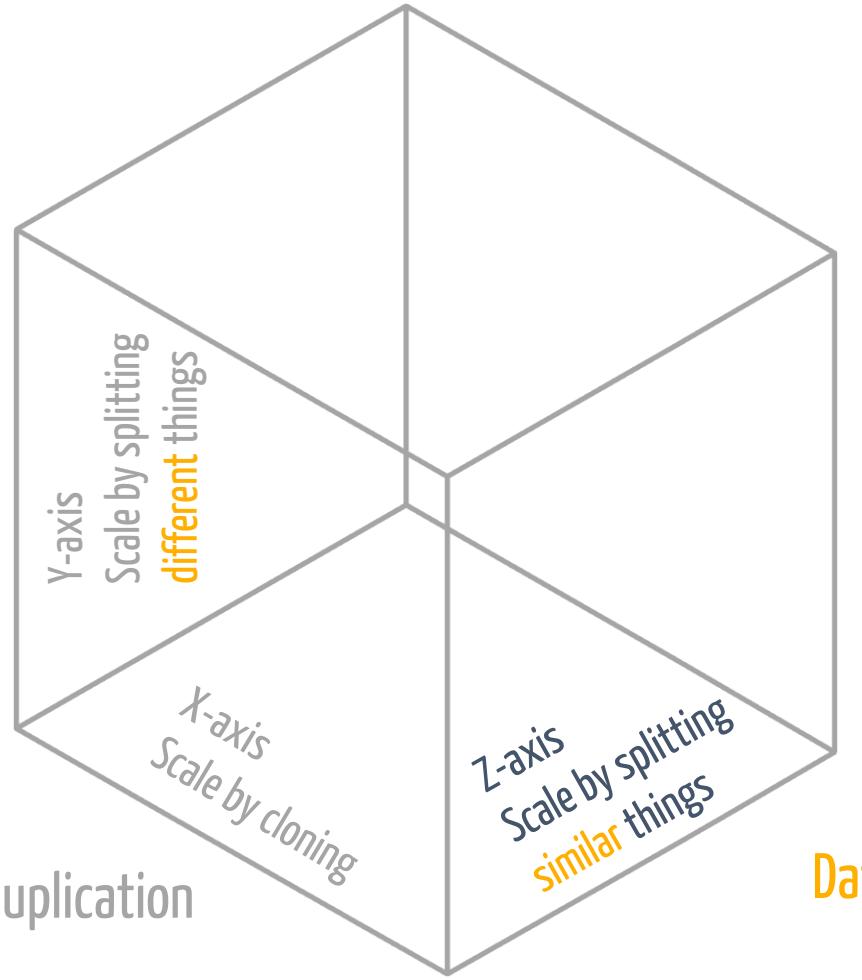
Mandy spoke up

and then

Mandy spoke up


Loadbalancer
Stateless Web
FrontEnd1
FrontEnd2
FrontEnd3
Stateful Tier (Hot)
Stateful1
Stateful2
Stateful3
Storage Tier (Cold)

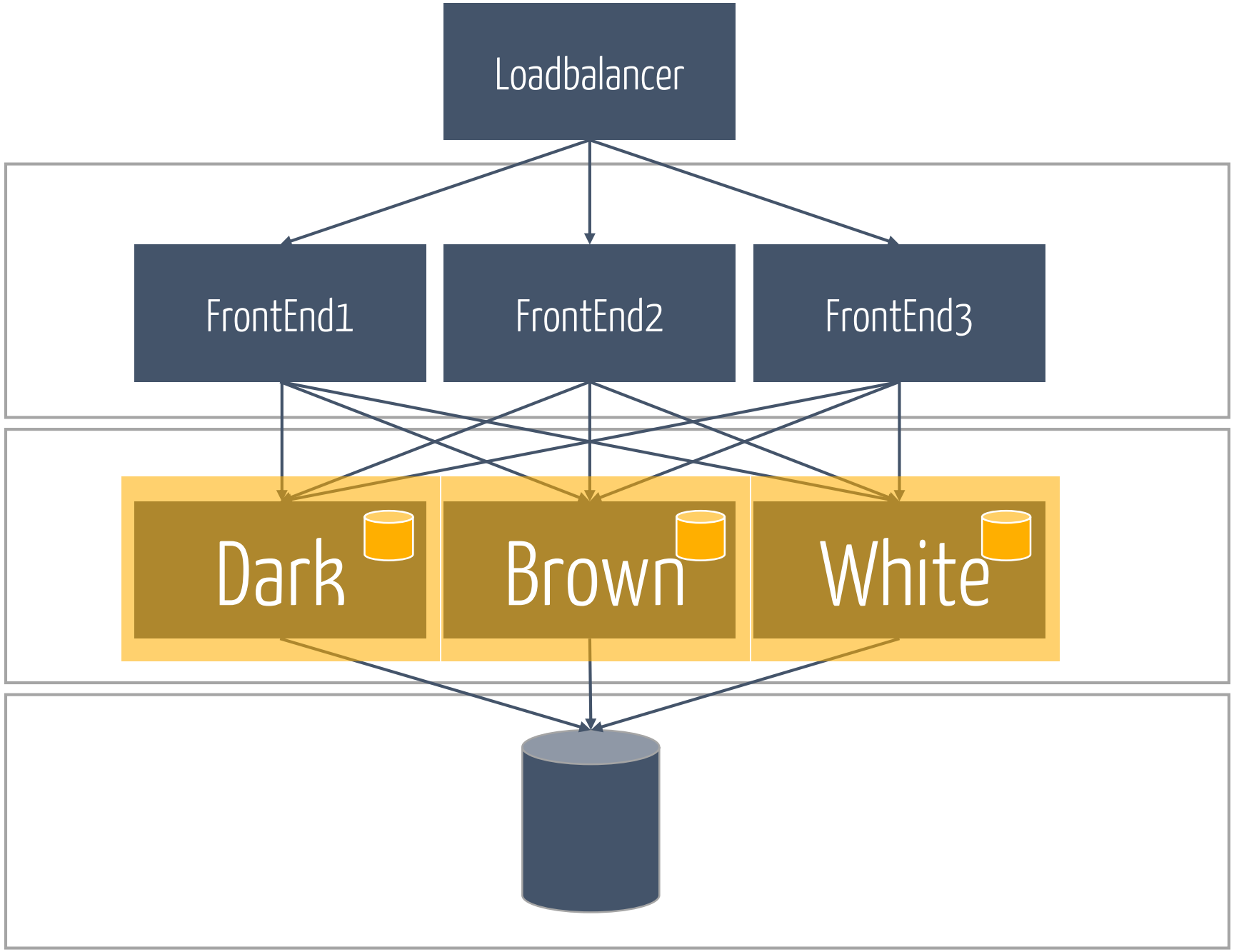
Functional decomposition

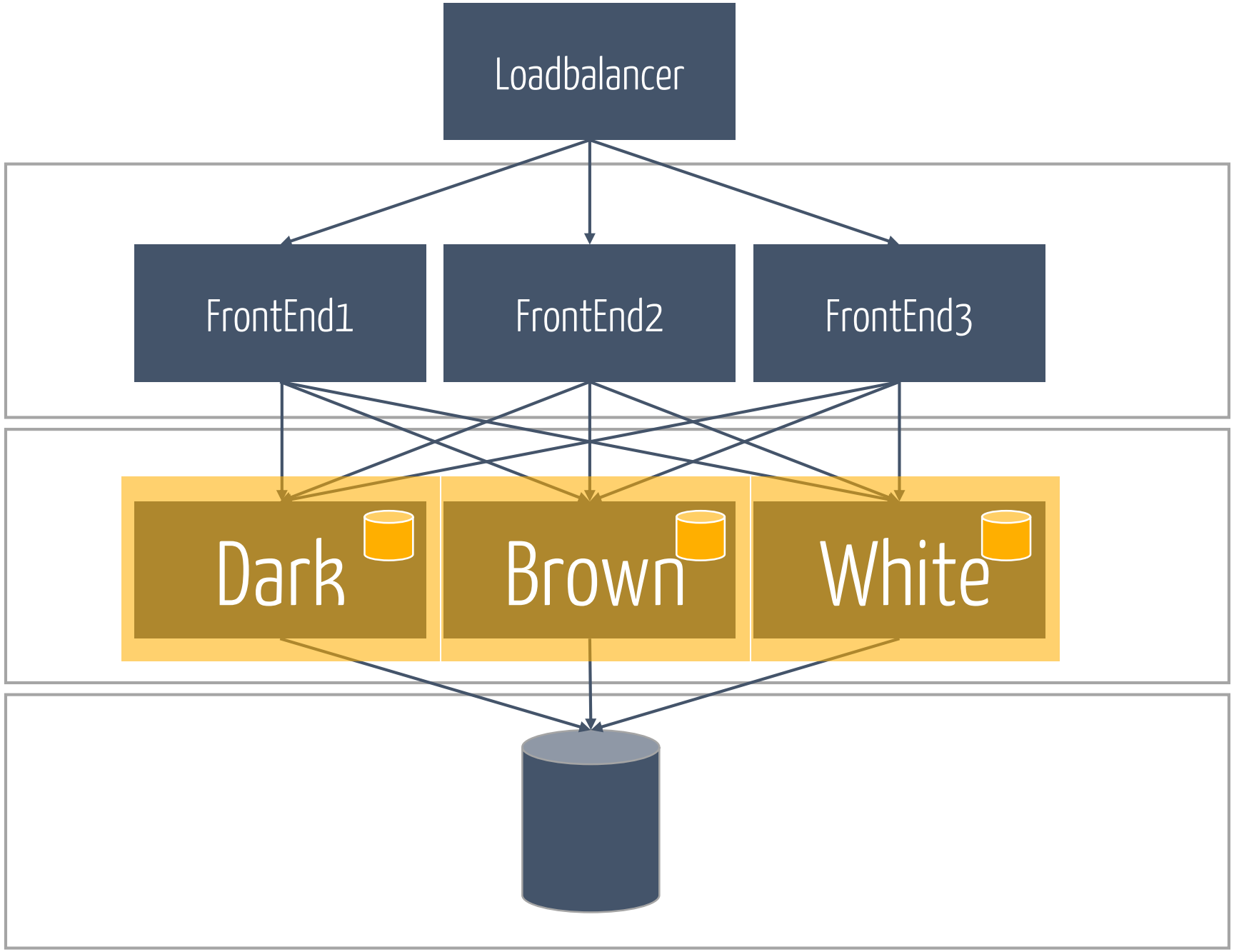


Horizontal duplication

Data Partitioning

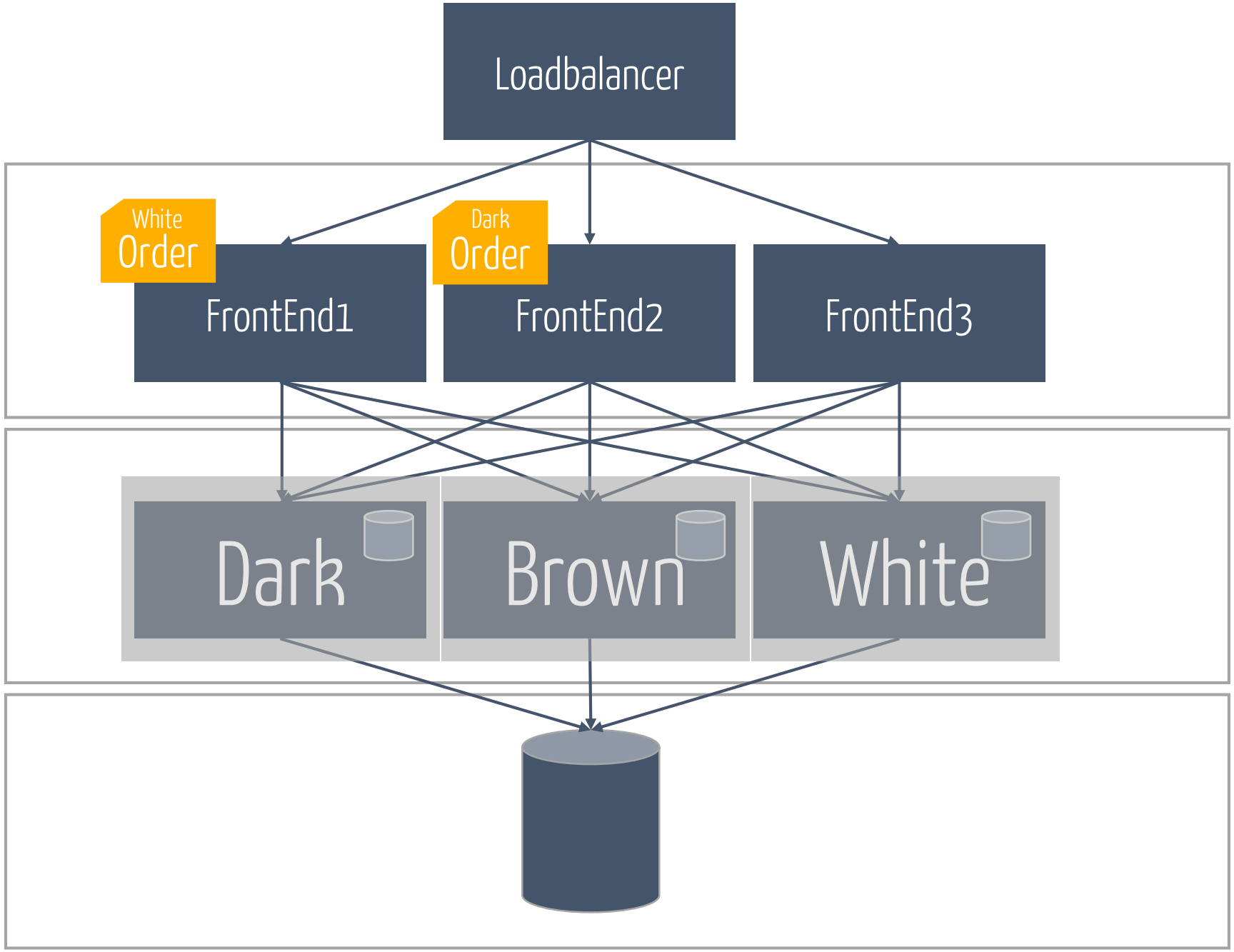
http://microservices.io/articles/scalecube.html


Loadbalancer
FrontEnd1
FrontEnd2
FrontEnd3
Dark
Brown
White


Loadbalancer
FrontEnd1
FrontEnd2
FrontEnd3
Dark
Brown
White

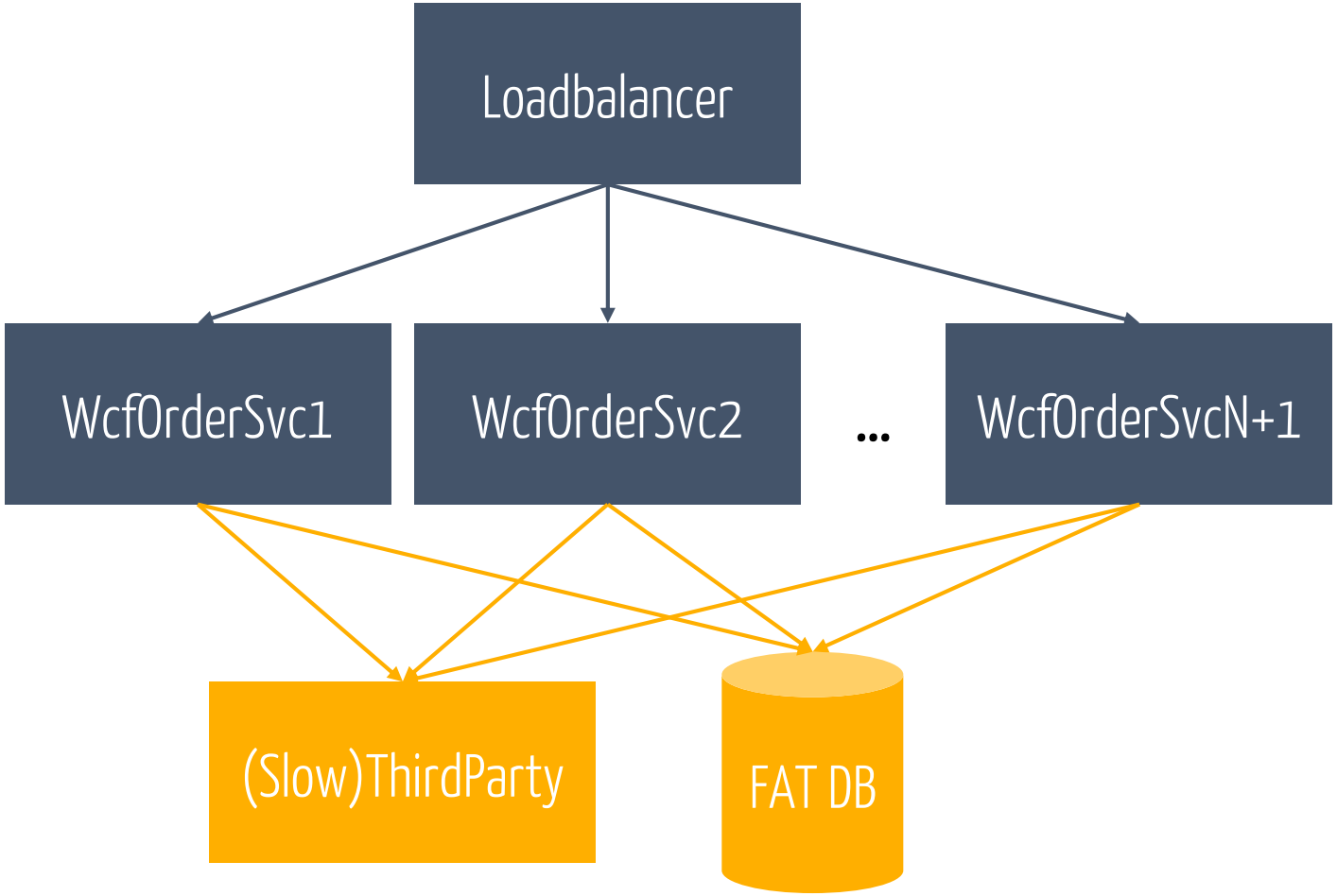
However

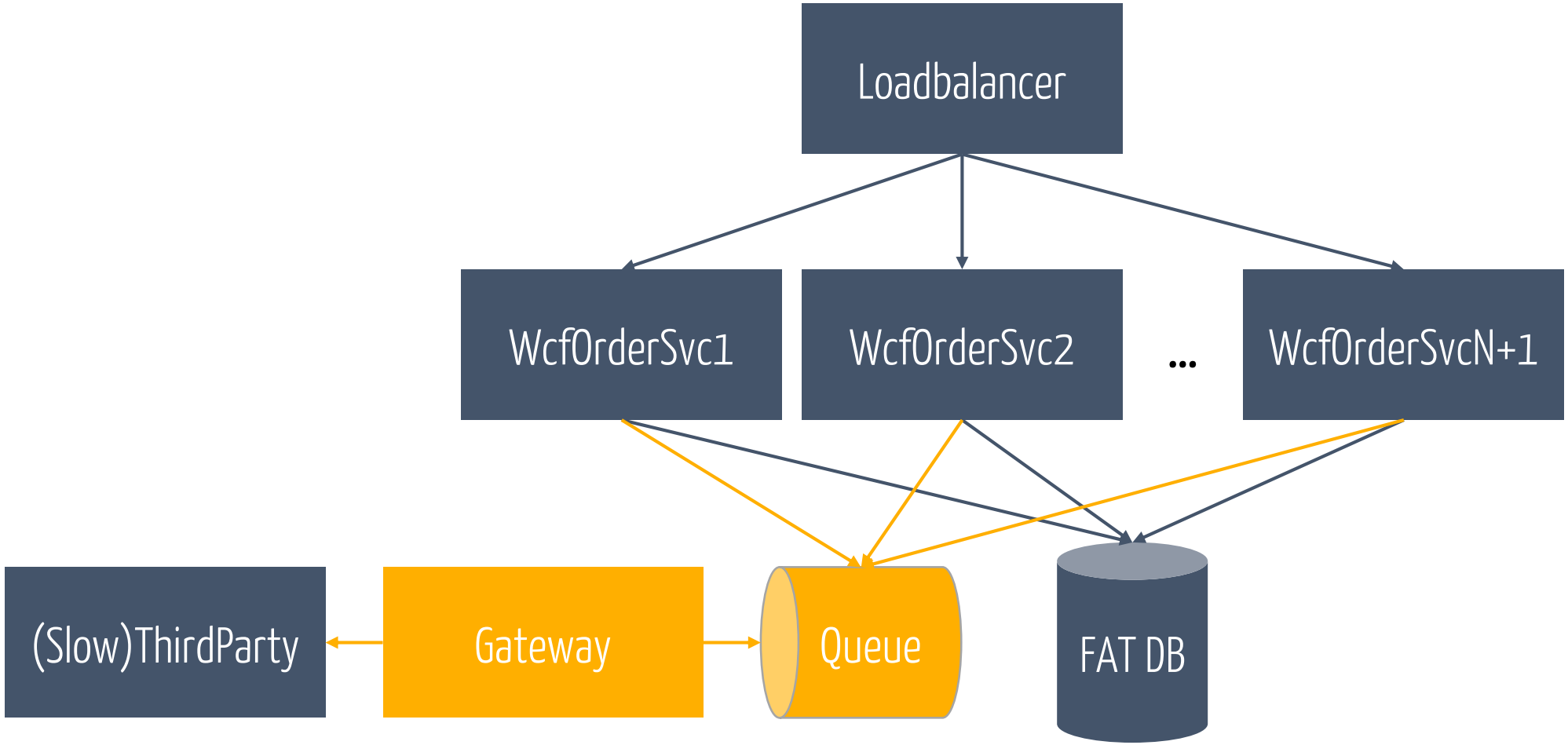
Joe couldn’t understand it

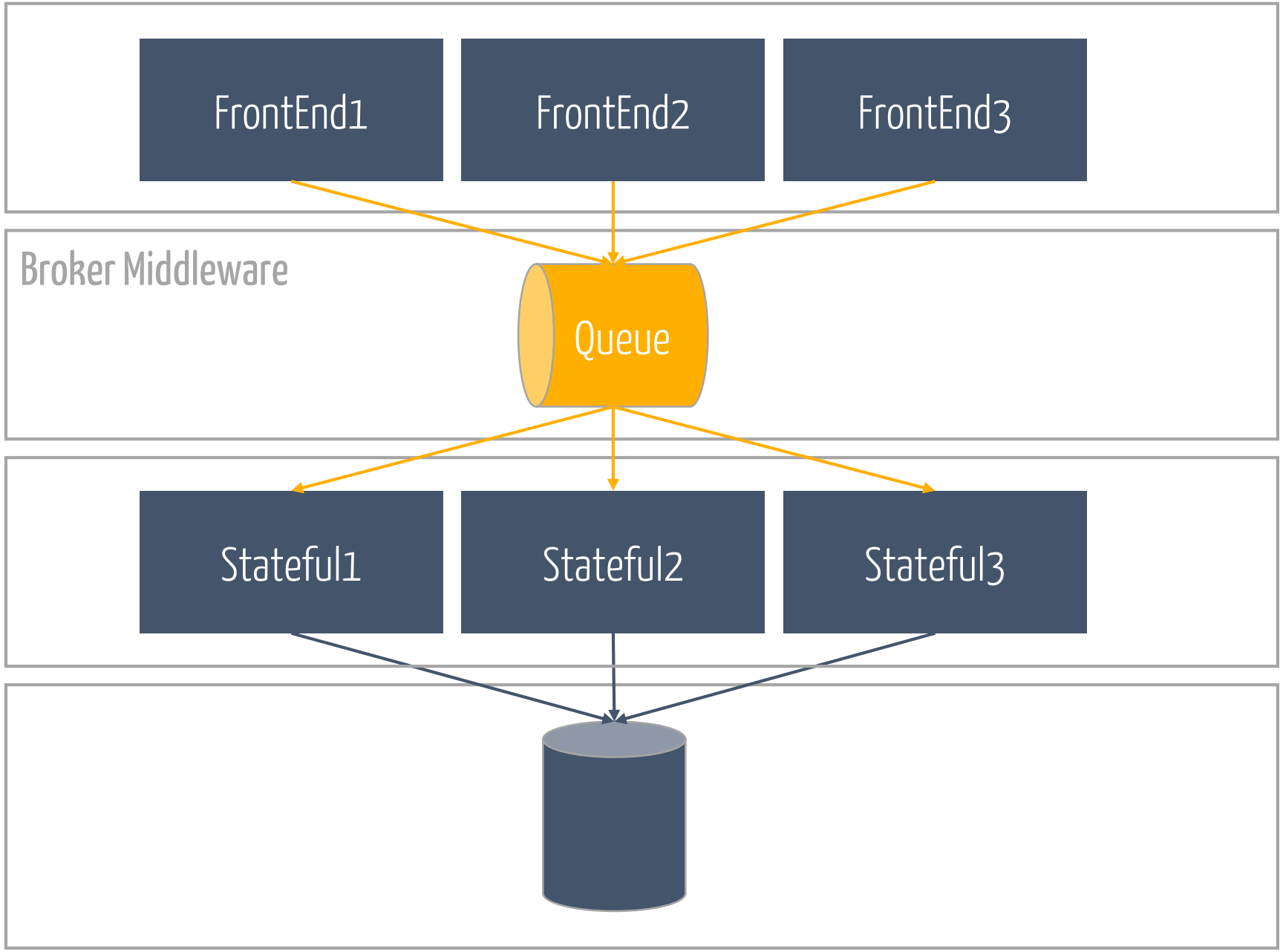

Loadbalancer
White
Order
Dark
Order
FrontEnd1
FrontEnd2
FrontEnd3
Dark
Brown
White

And then

Sophia threw a grenade


Loadbalancer
WcfOrderSvc1
WcfOrderSvc2
...
WcfOrderSvcN+1
(Slow)ThirdParty
FAT DB


Loadbalancer
WcfOrderSvc1
WcfOrderSvc2
...
WcfOrderSvcN+1
(Slow)ThirdParty
Gateway
Queue
FAT DB

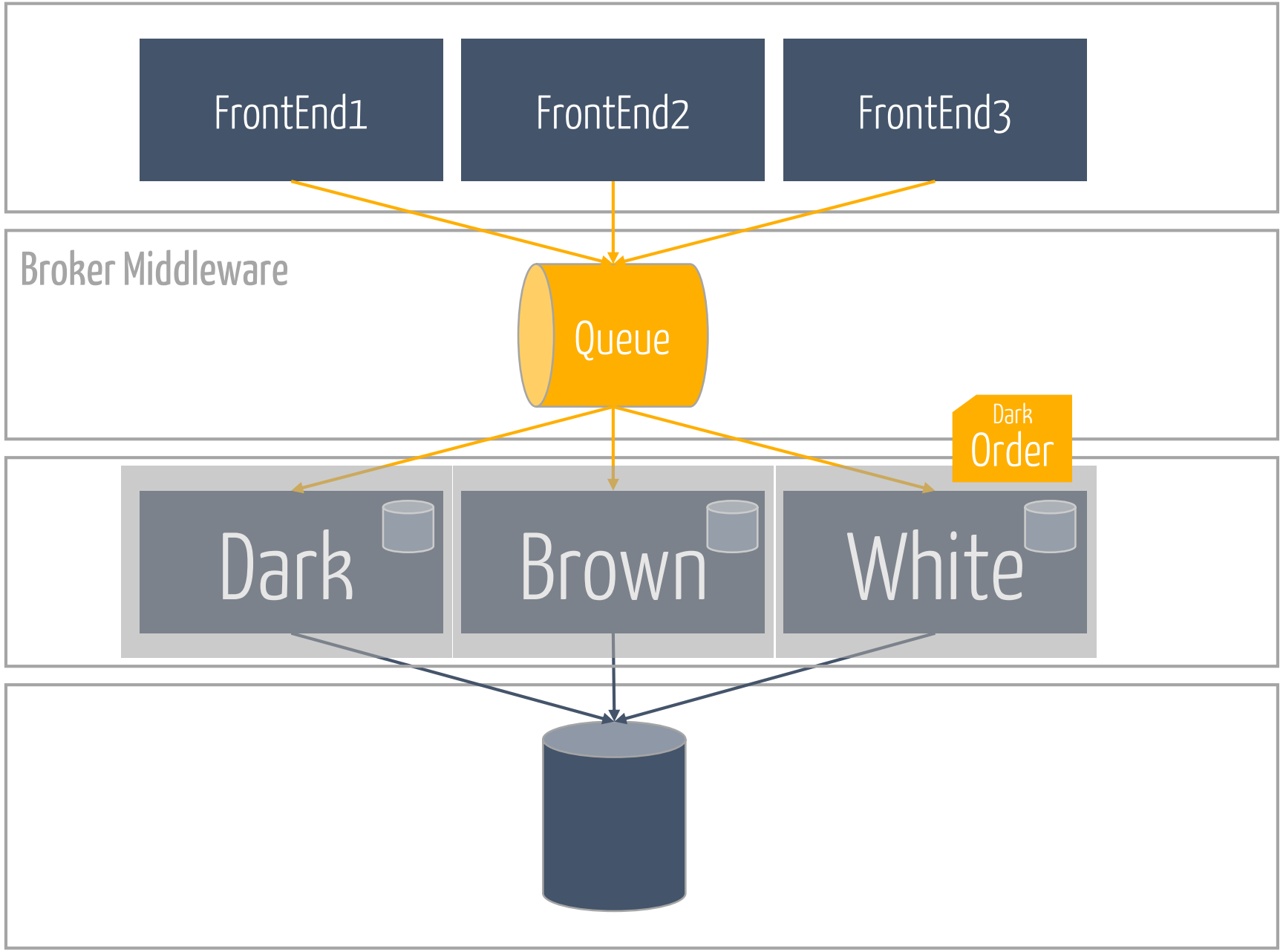

FrontEnd1
FrontEnd2
FrontEnd3
Broker Middleware
Queue
Stateful1
Stateful2
Stateful3

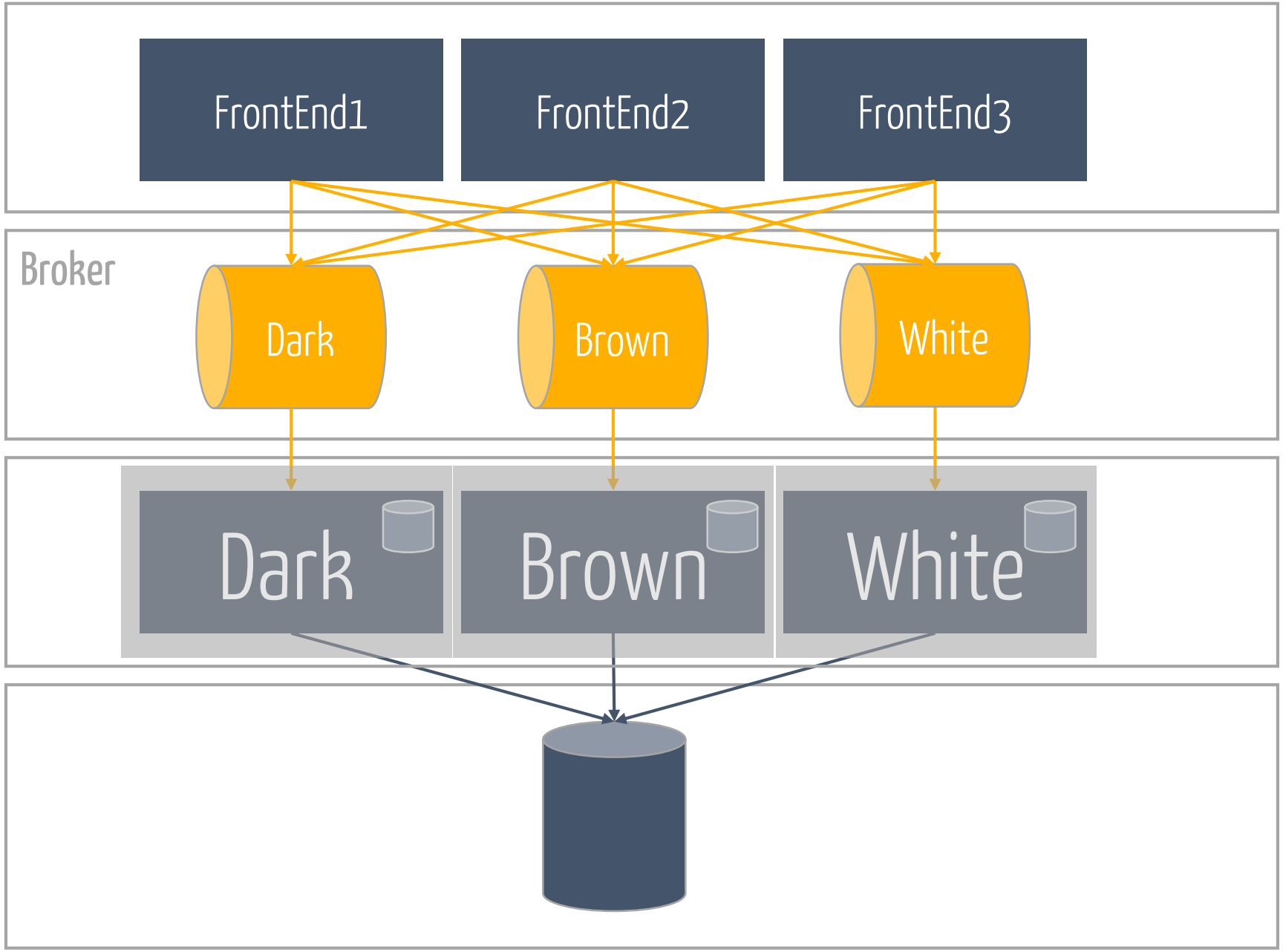
Peter snatches

the whiteboard markers

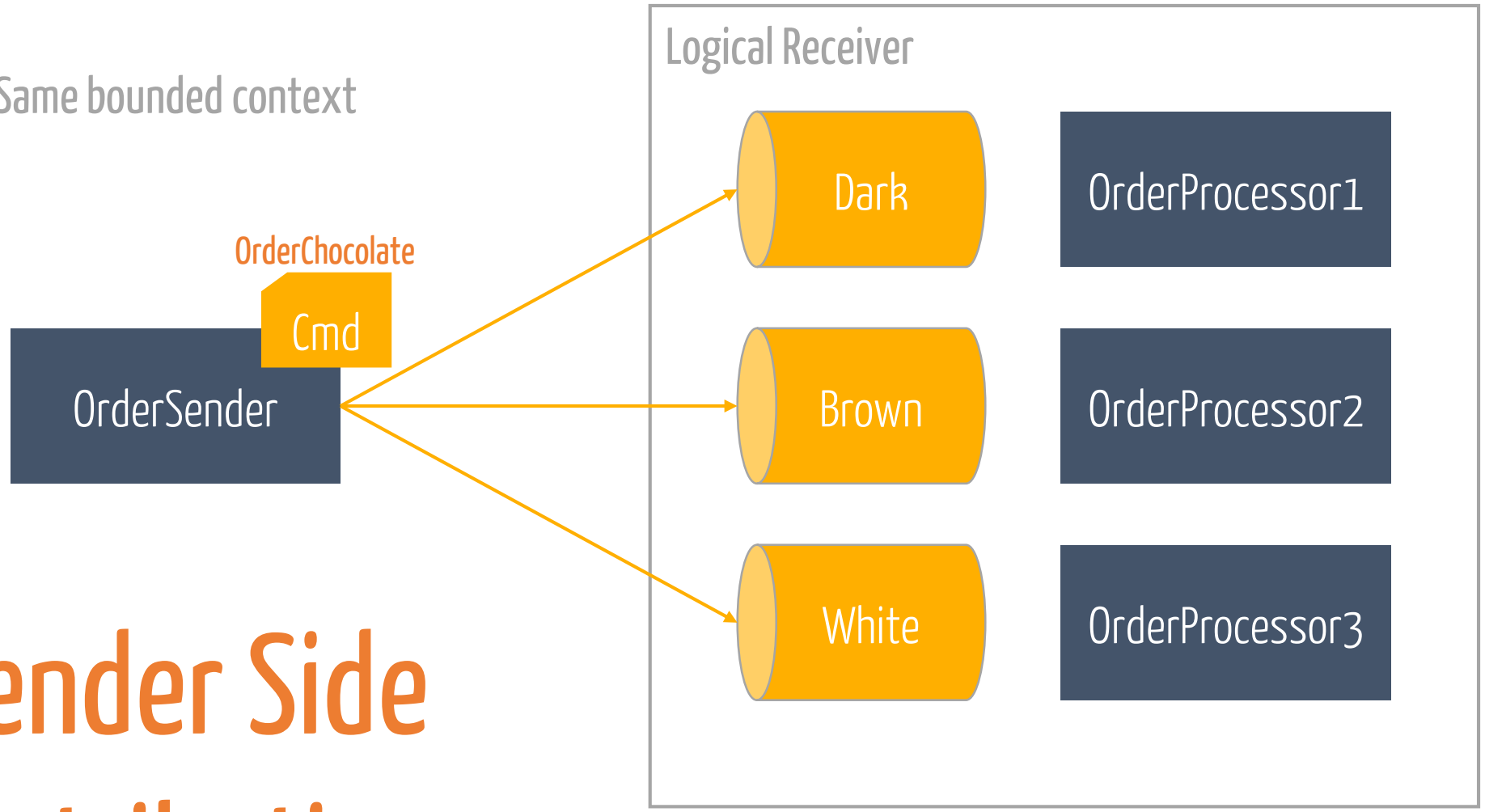
and furiously

screams


FrontEnd1
FrontEnd2
FrontEnd3
Broker Middleware
Queue
Dark
Order
Dark
Brown
White


FrontEnd1
FrontEnd2
FrontEnd3
Broker
Dark
Brown
White
Dark
Brown
White

Commands

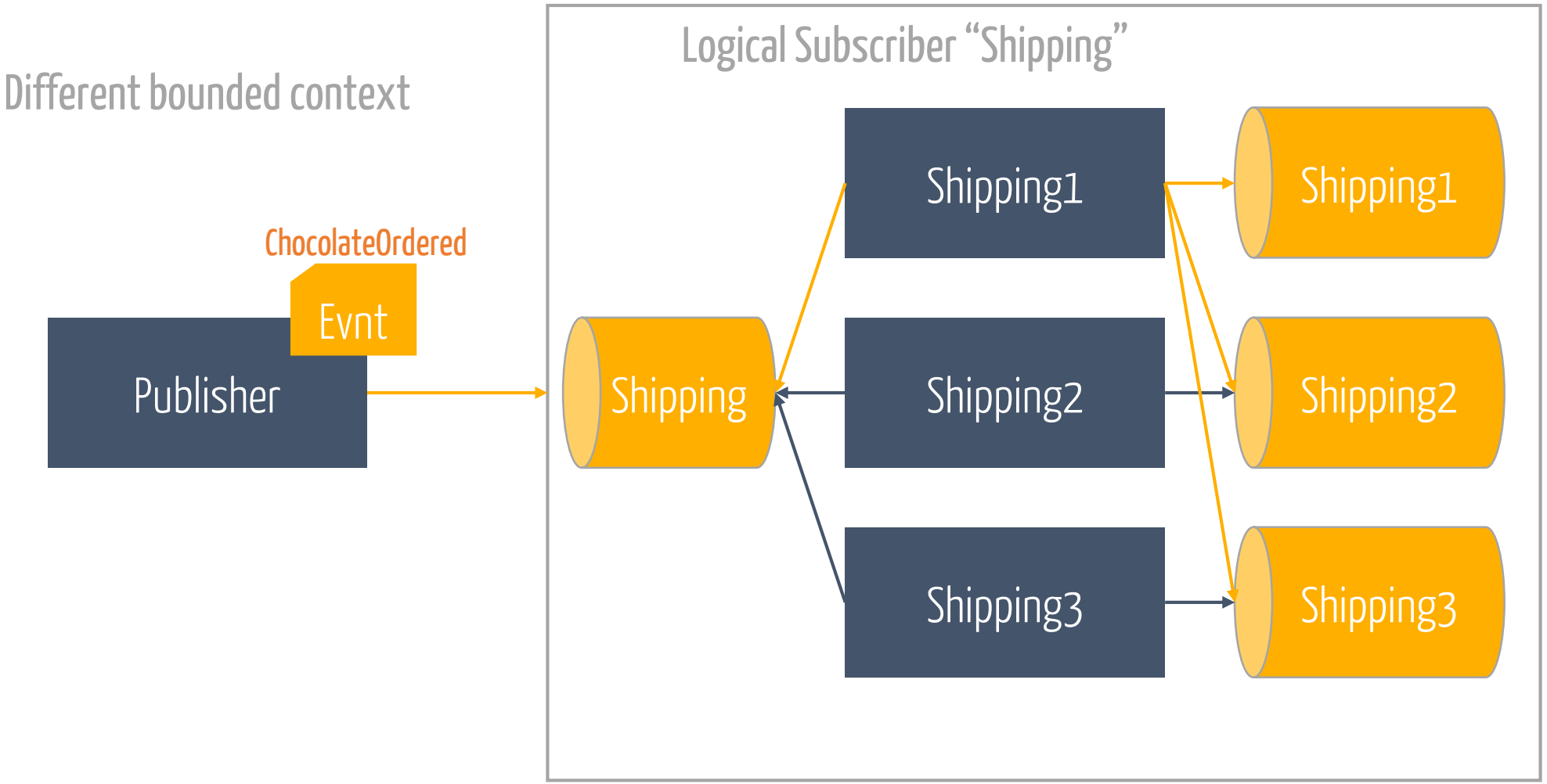


Sender Side Distribution

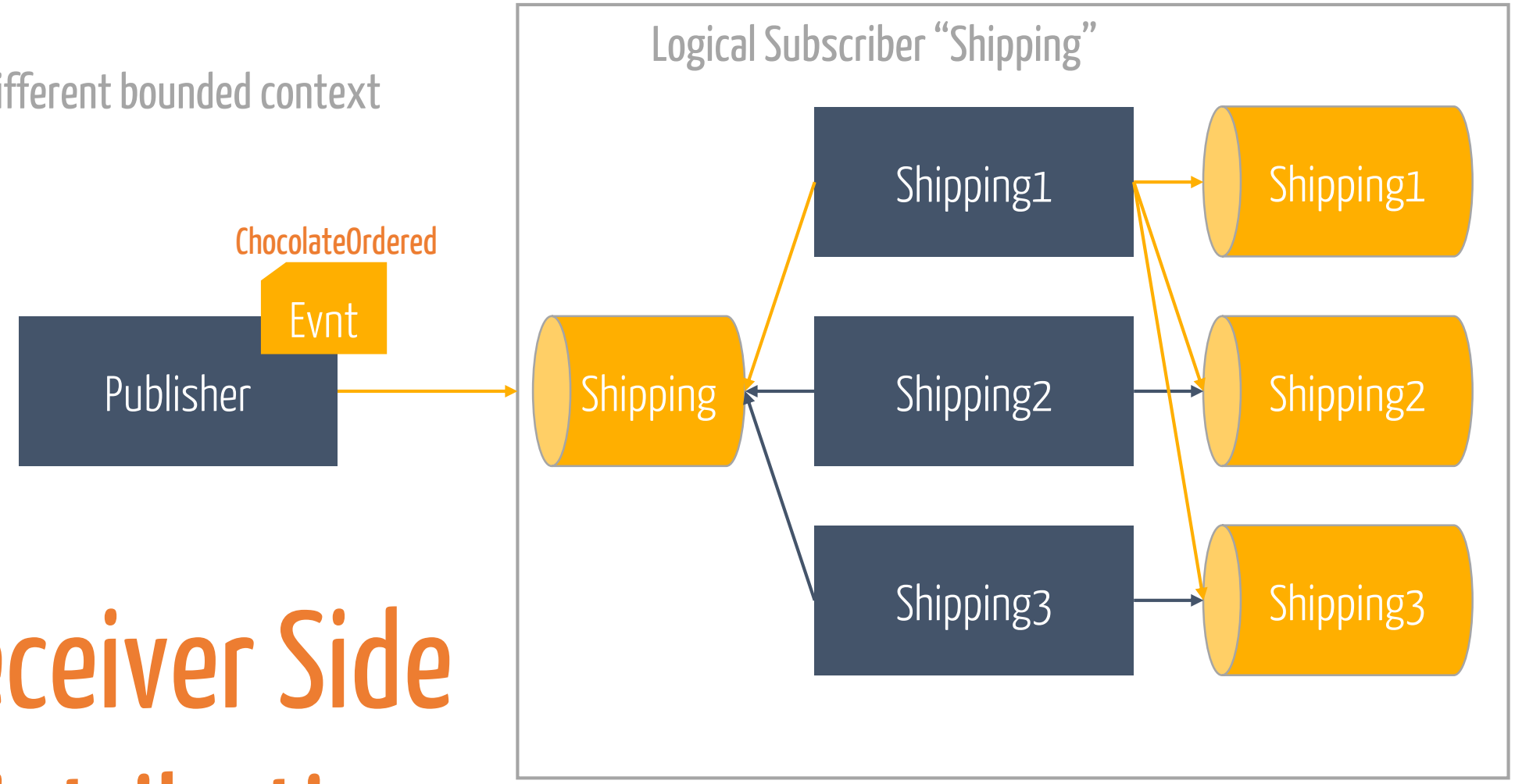
The PhD dude

acts like a smart ass

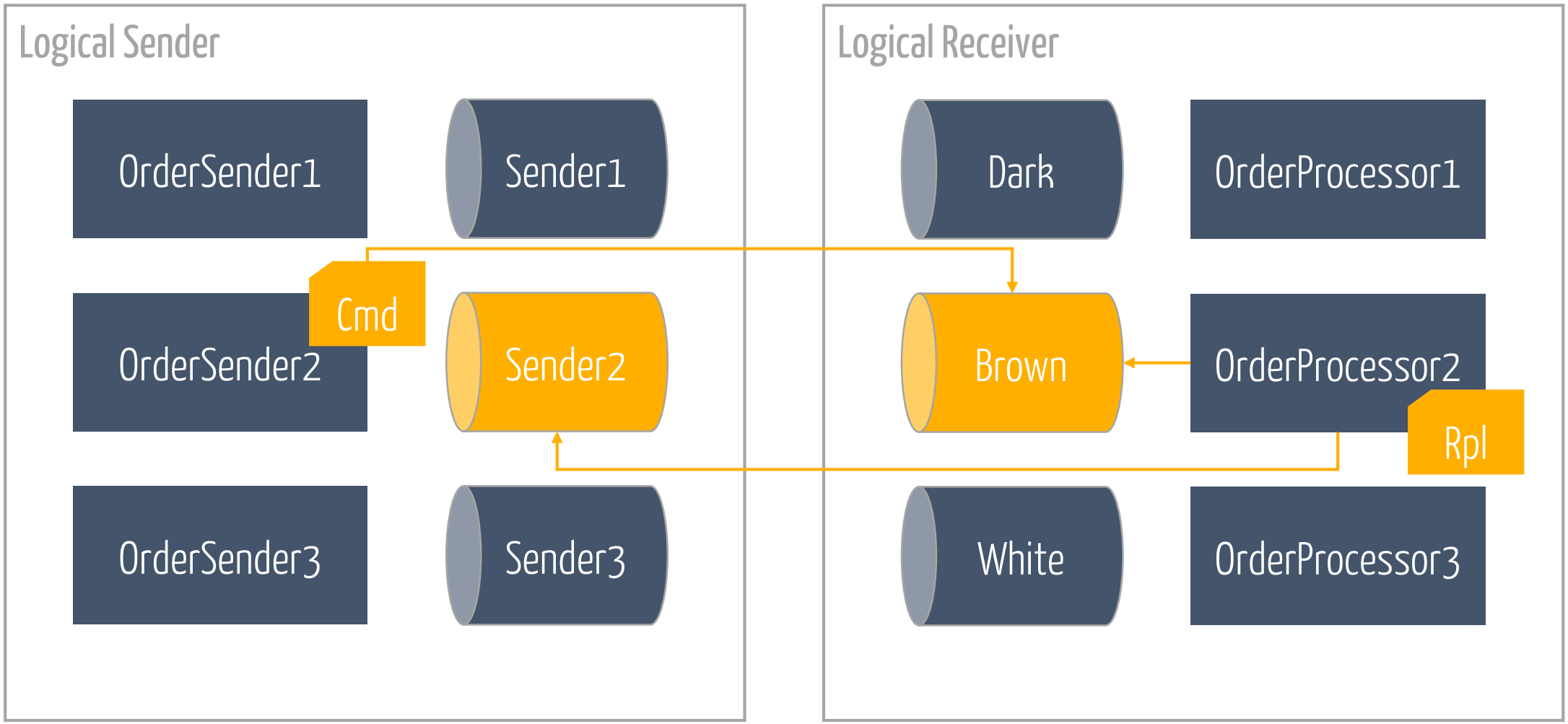
Events


Different bounded context
Logical Subscriber "Shipping"
ChocolateOrdered
Evnt
Publisher
Shipping
Shipping1
Shipping2
Shipping3
Shipping1
Shipping2
Shipping3

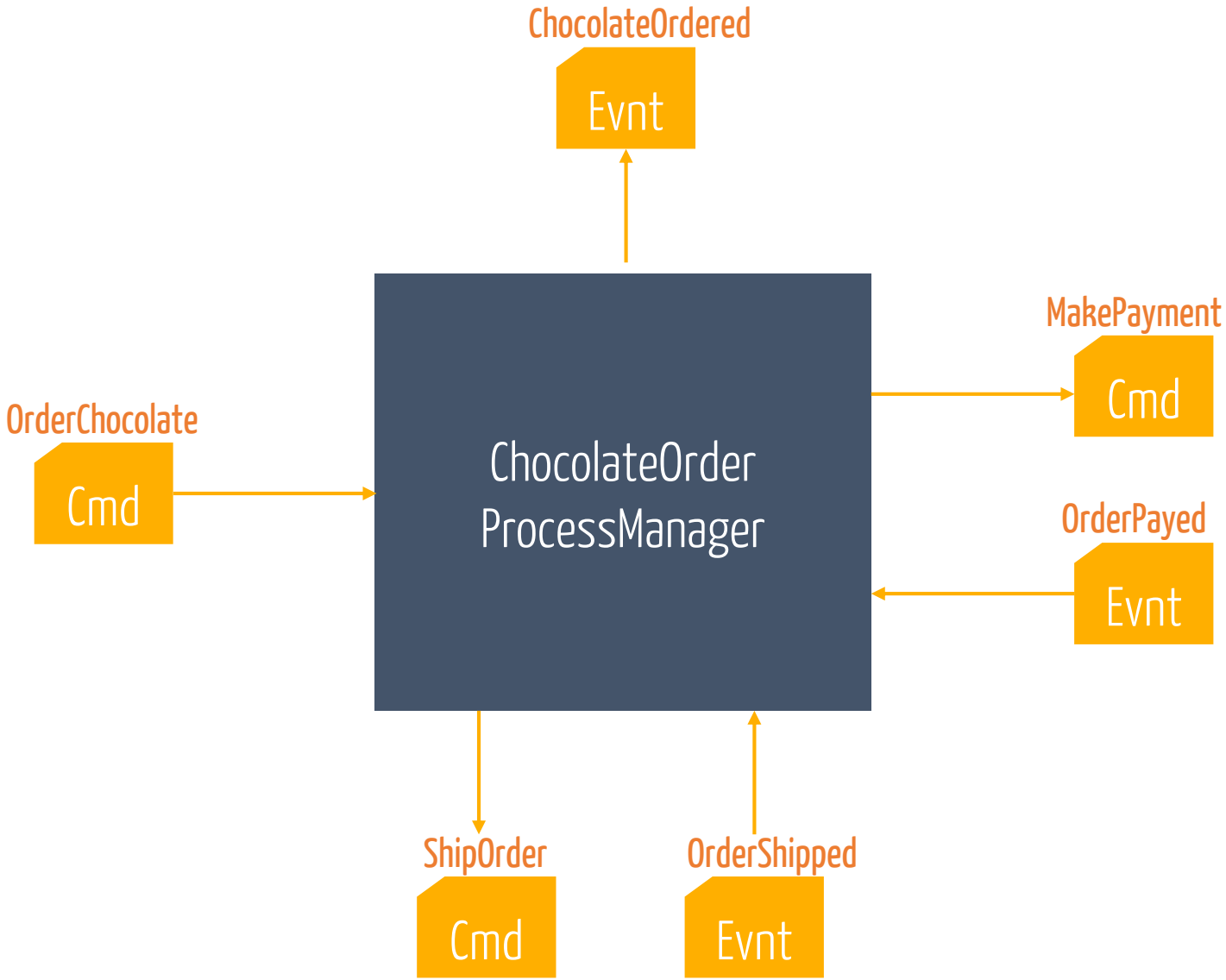
Receiver Side Distribution

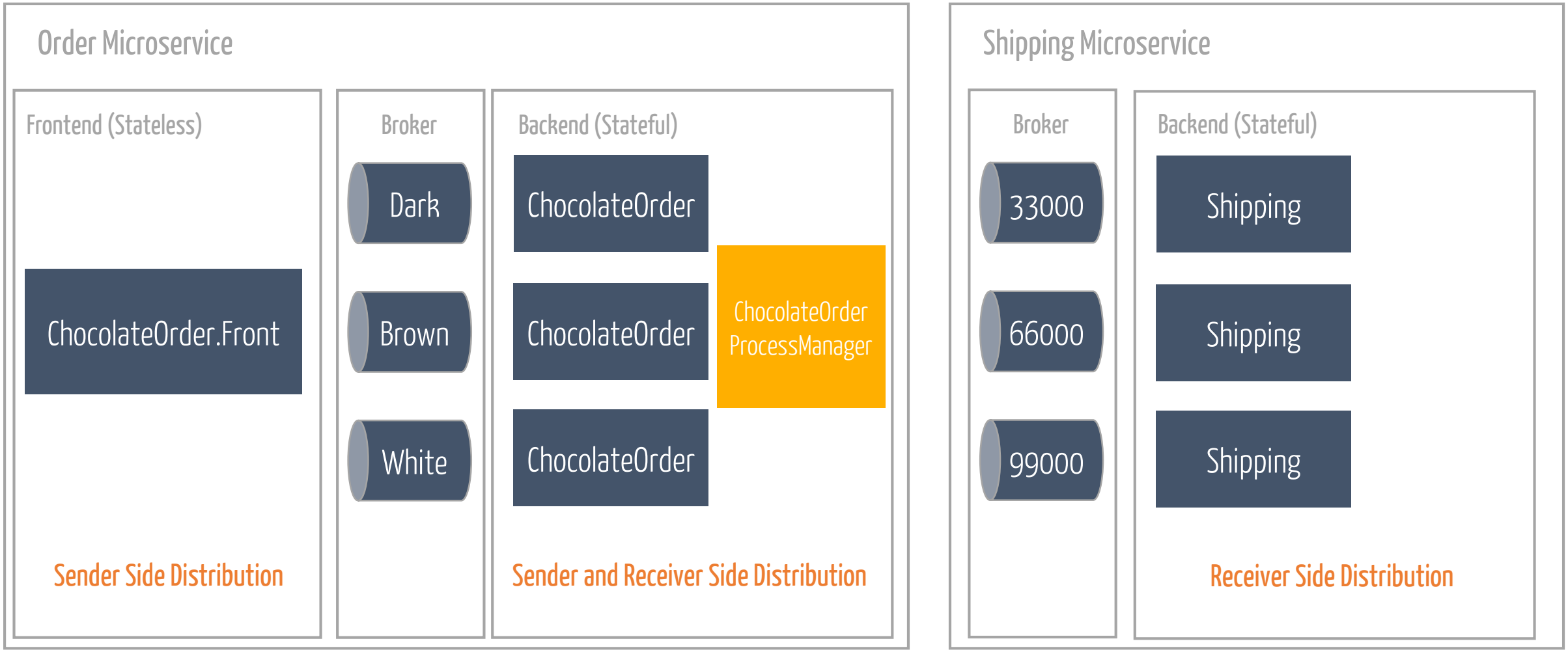


Request / Reply


Logical Sender
OrderSender1
Sender1
Cmd
OrderSender2
Sender2
OrderSender3
Sender3
Logical Receiver
Dark
OrderProcessor1
Brown
OrderProcessor2
Rpl
White
OrderProcessor3

Process Manager


ChocolateOrdered
Evnt
OrderChocolate
Cmd
ChocolateOrder
ProcessManager
MakePayment
Cmd
OrderPayed
Evnt
ShipOrder
Cmd
OrderShipped
Evnt


Order Microservice
Frontend (Stateless)
ChocolateOrder.Front
Sender Side Distribution
Broker
Dark
Brown
White
Backend (Stateful)
ChocolateOrder
ChocolateOrder
ChocolateOrder
ChocolateOrder ProcessManager
Sender and Receiver Side Distribution
Shipping Microservice
Broker
33000
66000
99000
Backend (Stateful)
Shipping
Shipping
Shipping
Receiver Side Distribution

Demo

Recap

It is always more difficult than Microsoft tells you ;)

Stateful computation with low latency requires smart routing

Service Fabric with stateless and stateful services combined with messaging gives you best of two worlds

docs.particular.net/
tutorials/quickstart

NServiceBus Quick Start

In this tutorial, we'll see why software systems built on asynchronous messaging using NServiceBus are superior to traditional synchronous HTTP-based web services. We'll also show how NServiceBus guarantees reliability and extensibility that can't be achieved with REST.

This tutorial skips over some concepts and implementation details in order to get up and running quickly. If you'd prefer to go more in-depth, check out our Introduction to NServiceBus tutorial. It will teach you the NServiceBus API and important concepts you need to learn to build successful message-based software systems.

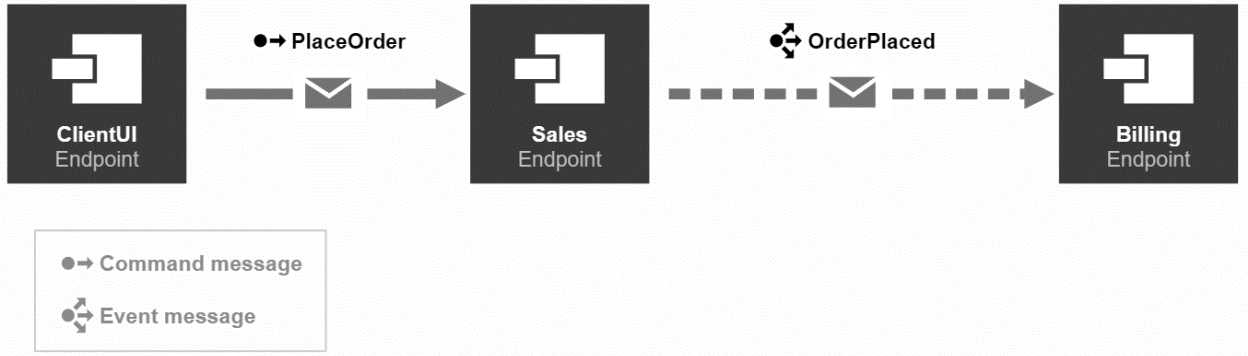
To get started, download the solution, extract the archive, and then open the **RetailDemo.sln** file with Visual Studio 2017.

Download the solution now

Project structure

The solution contains four projects. The **ClientUI**, **Sales**, and **Billing** projects are endpoints that communicate with each other using NServiceBus messages. The **ClientUI** endpoint mimics a web application and is an entry point in our system. The **Sales** and **Billing** endpoints contain business logic related to processing and fulfilling orders. Each endpoint references the **Messages** assembly, which contains the definitions of messages as POCO class files.

As shown in the diagram, the **ClientUI** endpoint sends a **PlaceOrder** command to the **Sales** endpoint. As a result, the **Sales** endpoint will publish an **OrderPlaced** event using the publish/subscribe pattern, which will be received by the **Billing** endpoint.



The solution mimics a real-life retail system, where the command to place an order is sent as a result of a customer interaction, and the actual processing occurs in the background. Publishing an event allows us to isolate the code to bill the credit card from the code to place the order, reducing coupling and making the system easier to maintain over the long term. Later in this tutorial, we'll see how to add a second subscriber in the **Shipping** endpoint which would begin the process of shipping the order.

docs.particular.net/

samples/azure/azure-service-fabric-routing/

Service Fabric Partition Aware Routing

Download | Nugets Used | Edit Code

Component: NServiceBus | Nuget: NServiceBus (Version: 6.x)

This sample currently makes use of a pre-release version of NServiceBus.Persistence.ServiceFabric.

The sample demonstrates how the NServiceBus API can be used to implement partition aware routing for services hosted inside a Service Fabric cluster. It takes advantage of routing system extensibility points and custom pipeline behaviors to support various types of NServiceBus communication patterns. It is assumed that the NServiceBus users are able to define mapping between message type and service partition for each message. It is also assumed that `send local`, `timeout` and `reply` messages are partition affine i.e. should be processed in the context of originating partition. The sample consists of services hosted inside and outside the Service Fabric and enables proper communication between the two.

Scenario

The scenario used in this sample covers a voting system. In this voting system the cast votes are counted by candidate. The endpoint responsible for counting candidate votes is subscribed to an event published when votes are cast.

Next to this the system also counts the total number of votes cast in each zip code. In order to achieve this the candidate voting endpoint issues a `request` to the zip code counting endpoint to track the zip code. The zip code counting endpoint will `reply` back with the intermediary results.

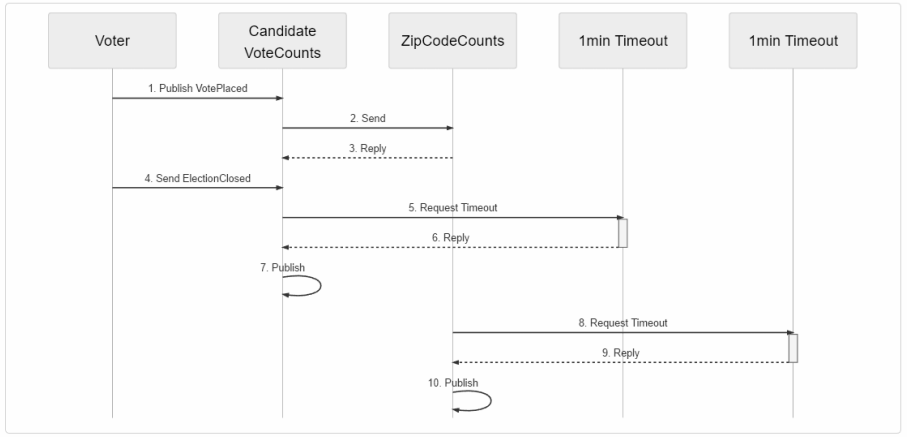
When the election is closed, the candidate vote counting endpoint will `publish` the results per candidate and report them using Service Fabric diagnostics infrastructure (ETW/ Event Viewer windows).

After receiving timeout message from each zip code counting endpoint `sends a local command` to report the [illegible] zip codes.

The sample assumes that:

- There are only 2 candidates in the election, called "John" and "Abby",
- Zip codes are integers in the range of 0 to 99000.

This simplifies partition id value calculation. In a real world scenario a hash function could be used to perform mapping from arbitrary input types.

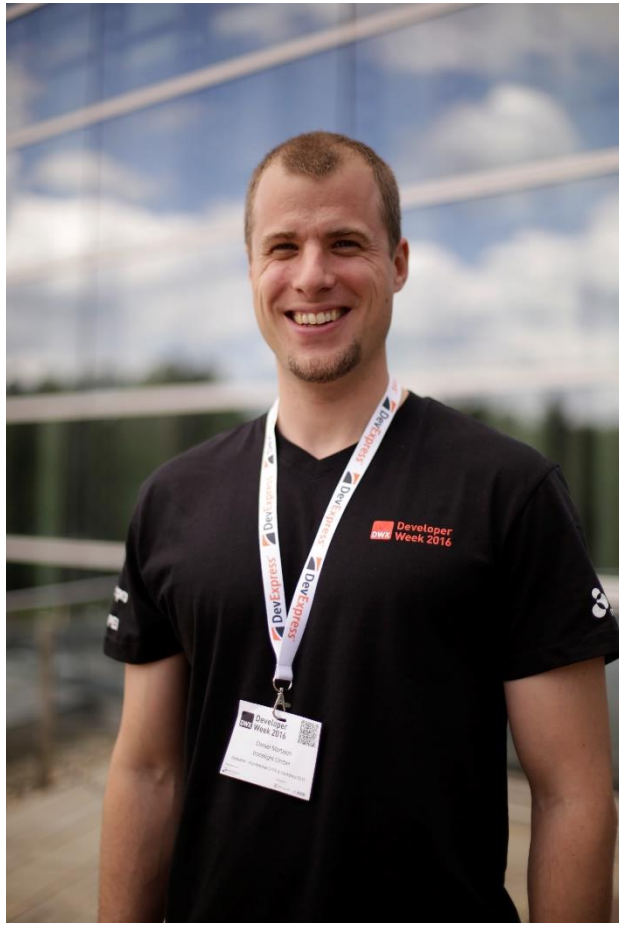


Slides, Links...

github.com/danielmarbach/Microservices.ServiceFabric

Particular
Software

Q&A



Particular Software

Software Engineer
Enthusiastic Software Engineer
Microsoft MVP

@danielmarbach
particular.net/blog
planetgeek.ch

Particular
Software

Thanks