Particular
Software

Microservices with Service Fabric

Easy... or is it?

Goals

Stateless and Stateful Services

Partitioning of Business Data

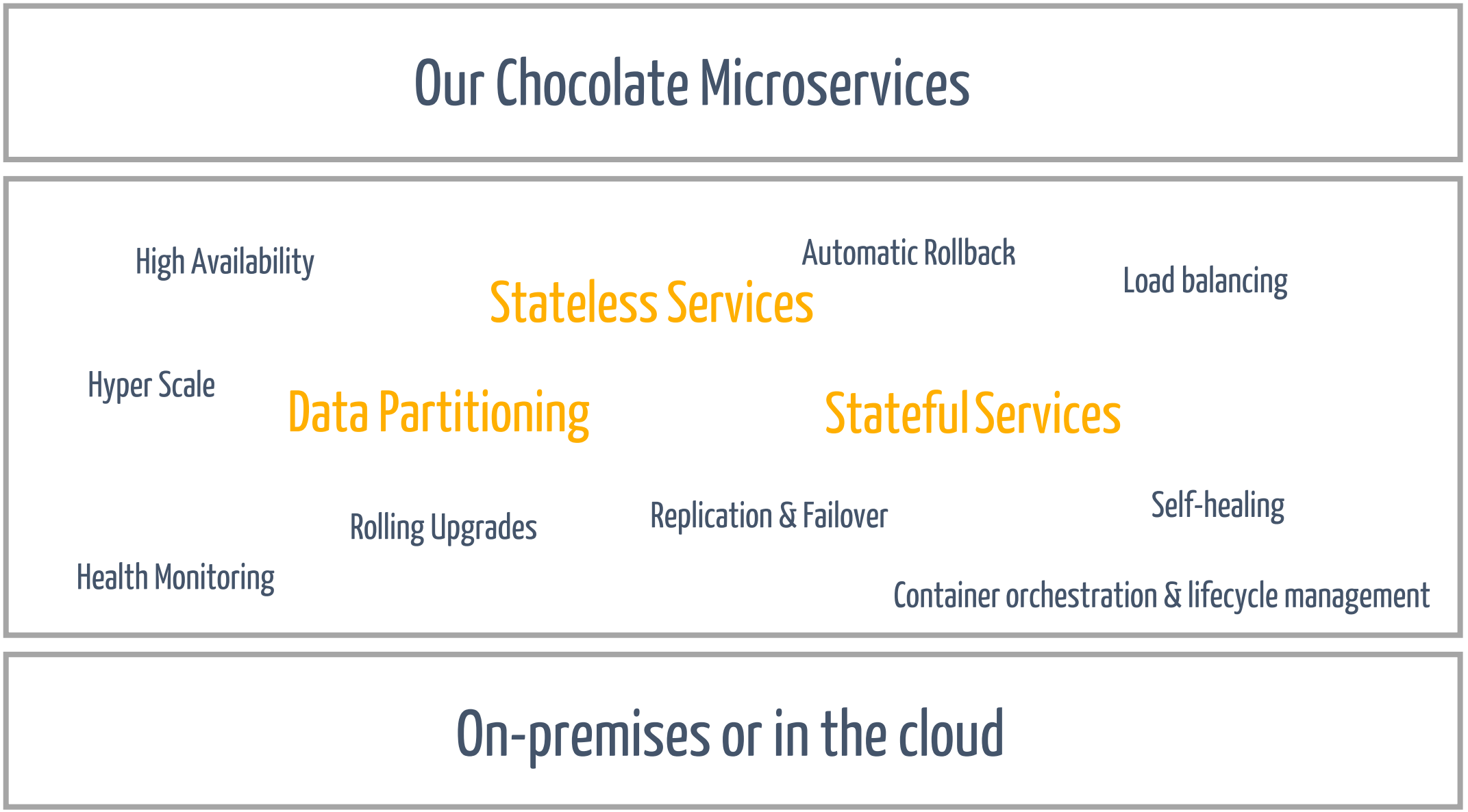
Message Patterns and Partitioning

Premise

A Tale of chocolate

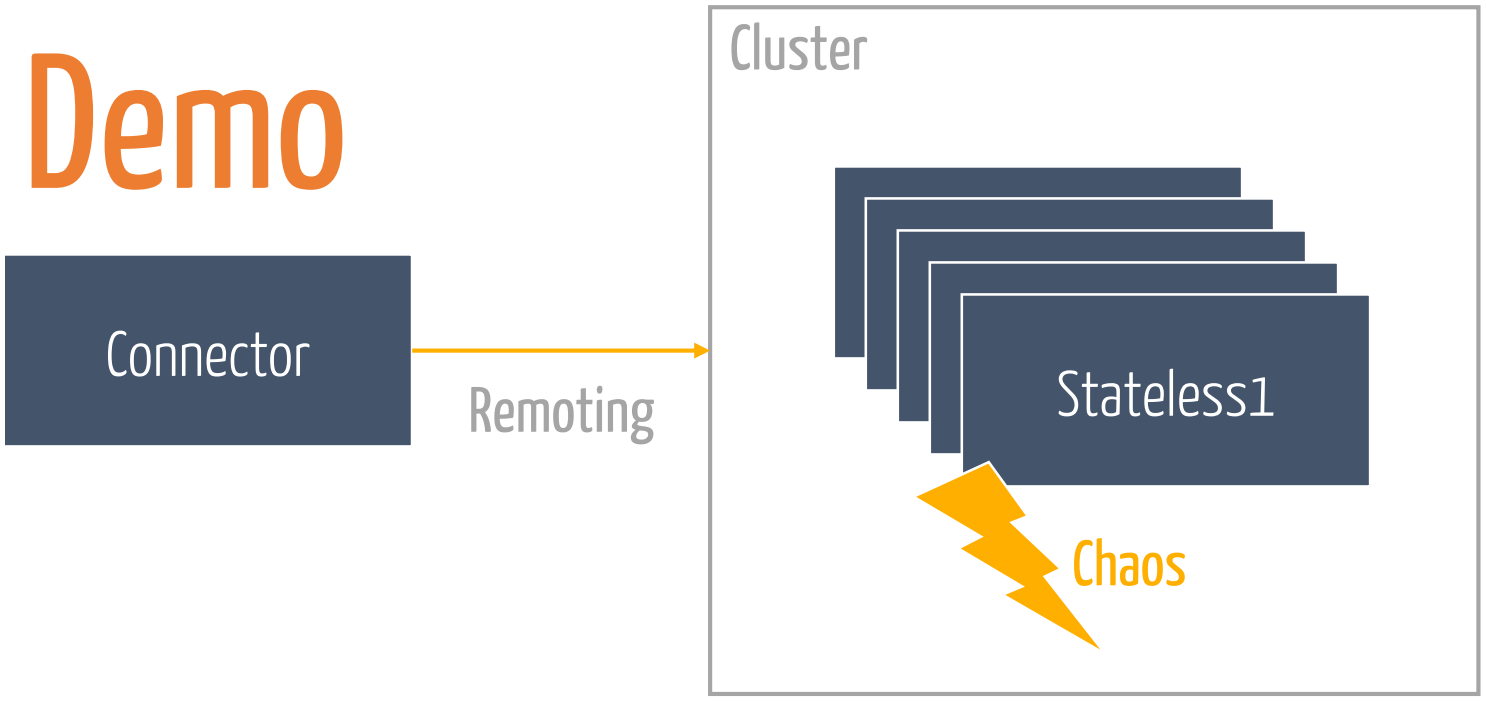
Horrible death of easter bunnies

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

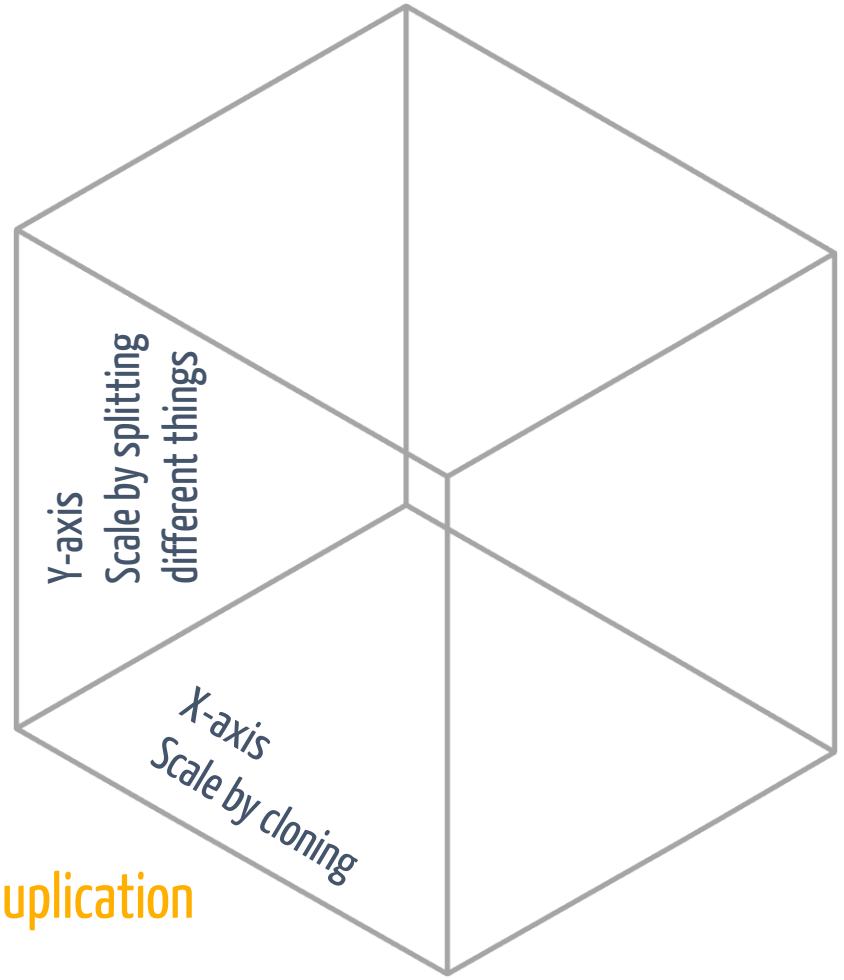
Demo



Demo

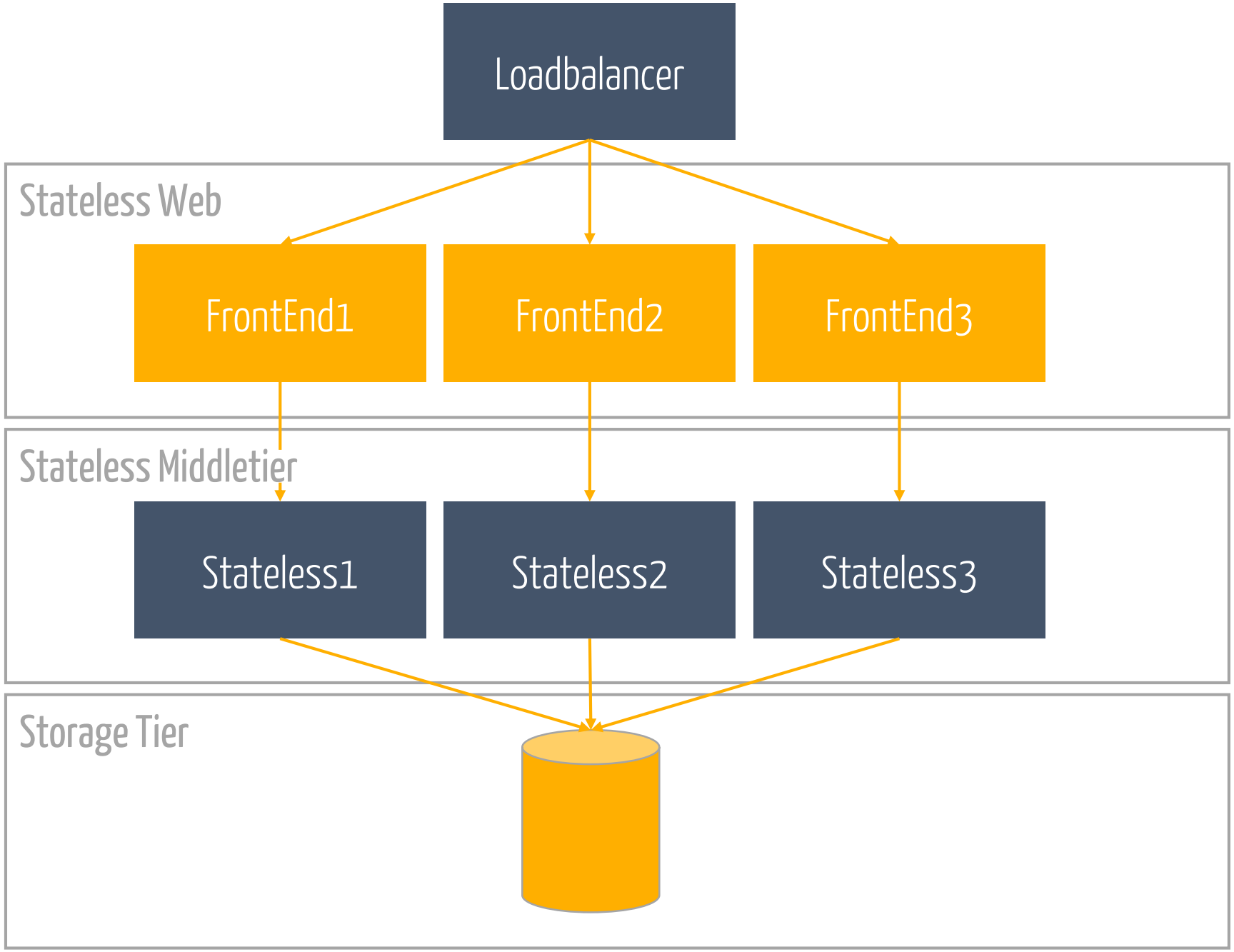
Let's scale chocolate

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

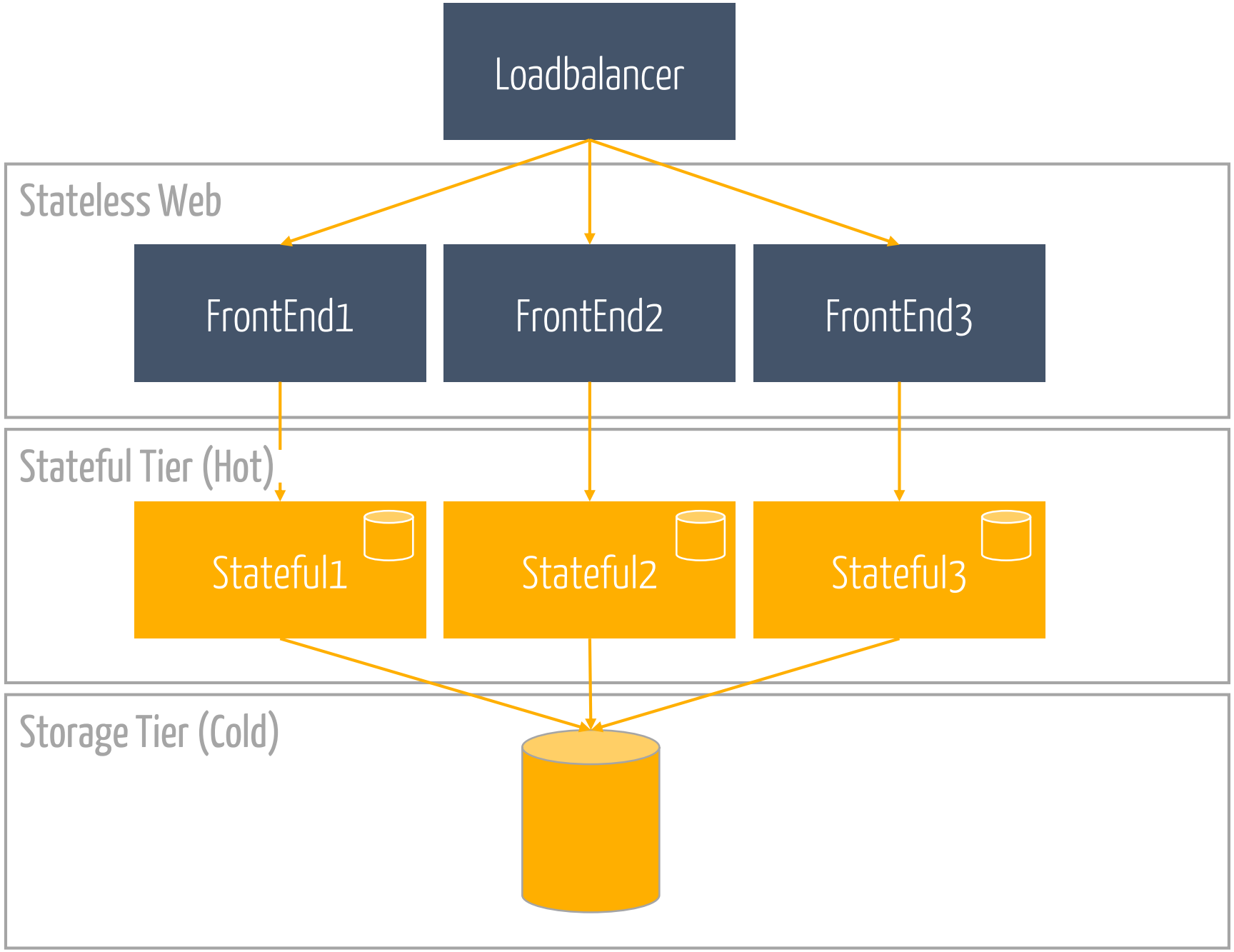
Mandy spoke up

and then

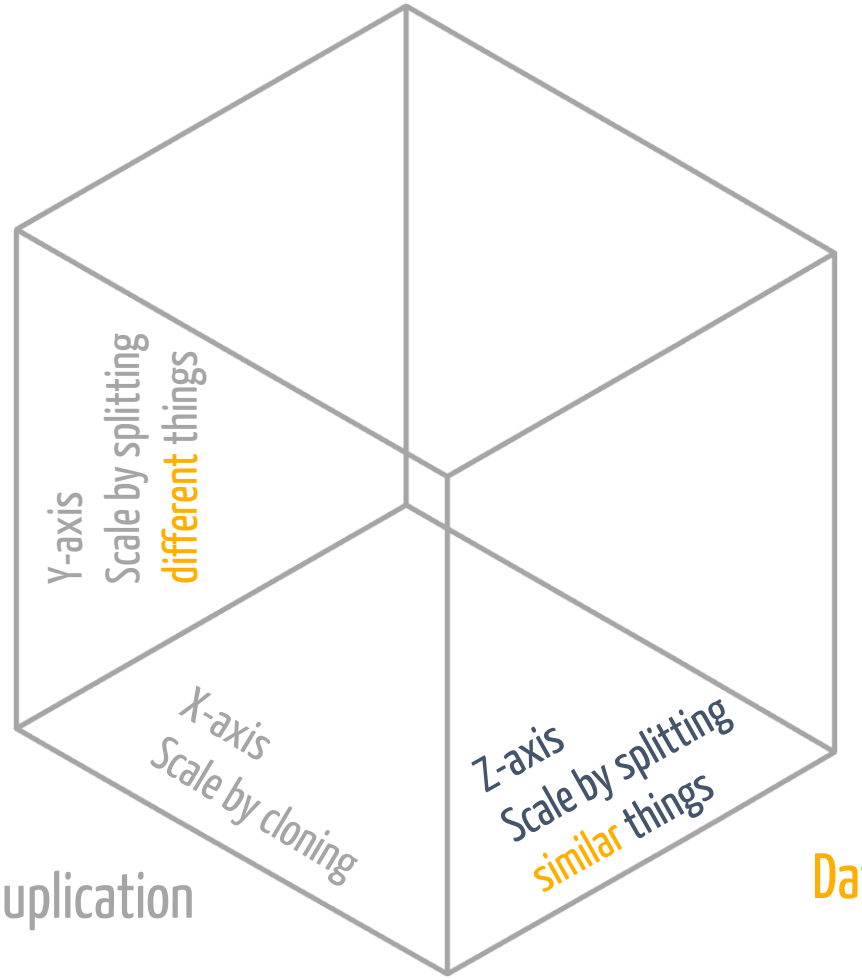
Mandy spoke up

and then

Mandy spoke up


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

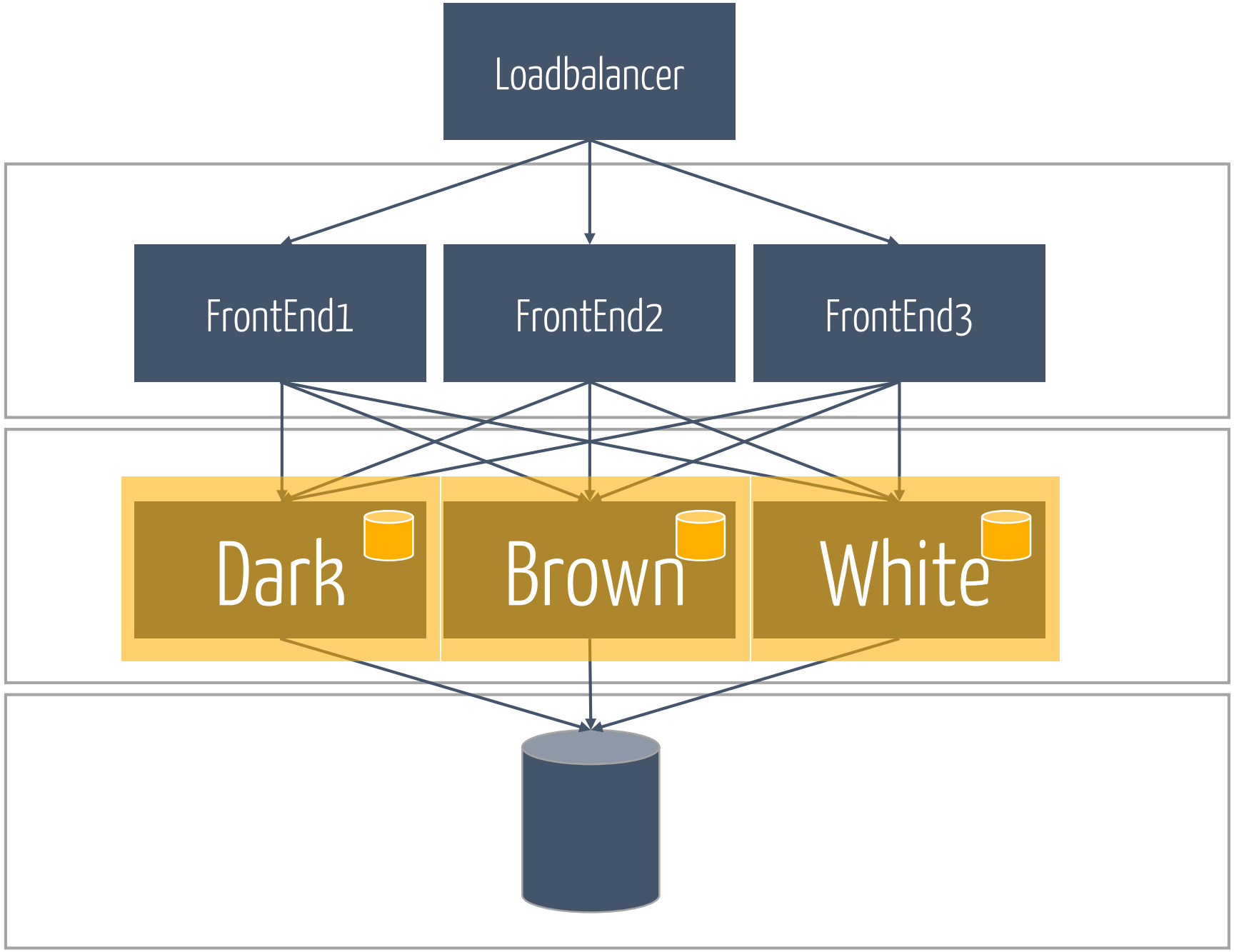
Functional decomposition

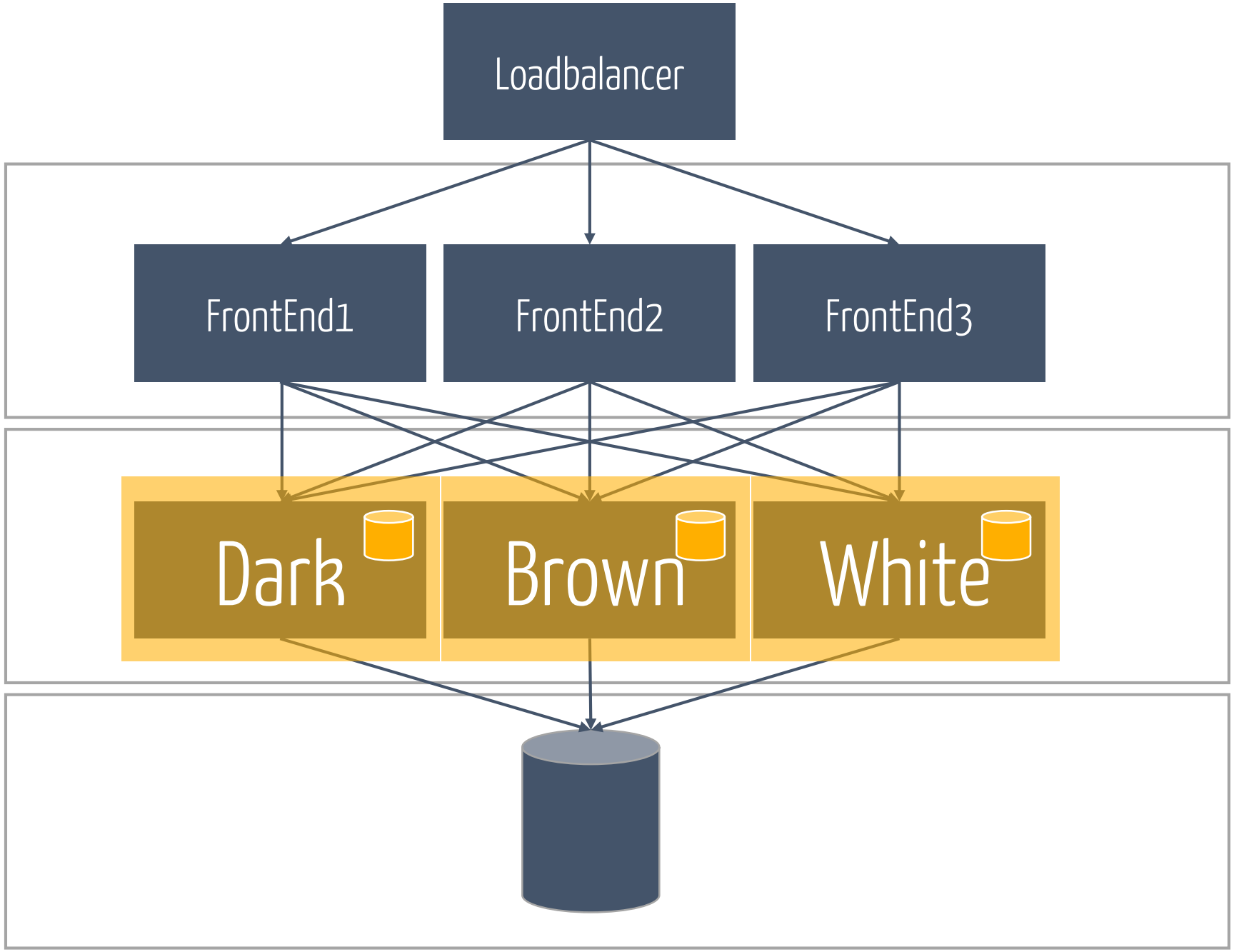


Horizontal duplication

Data Partitioning

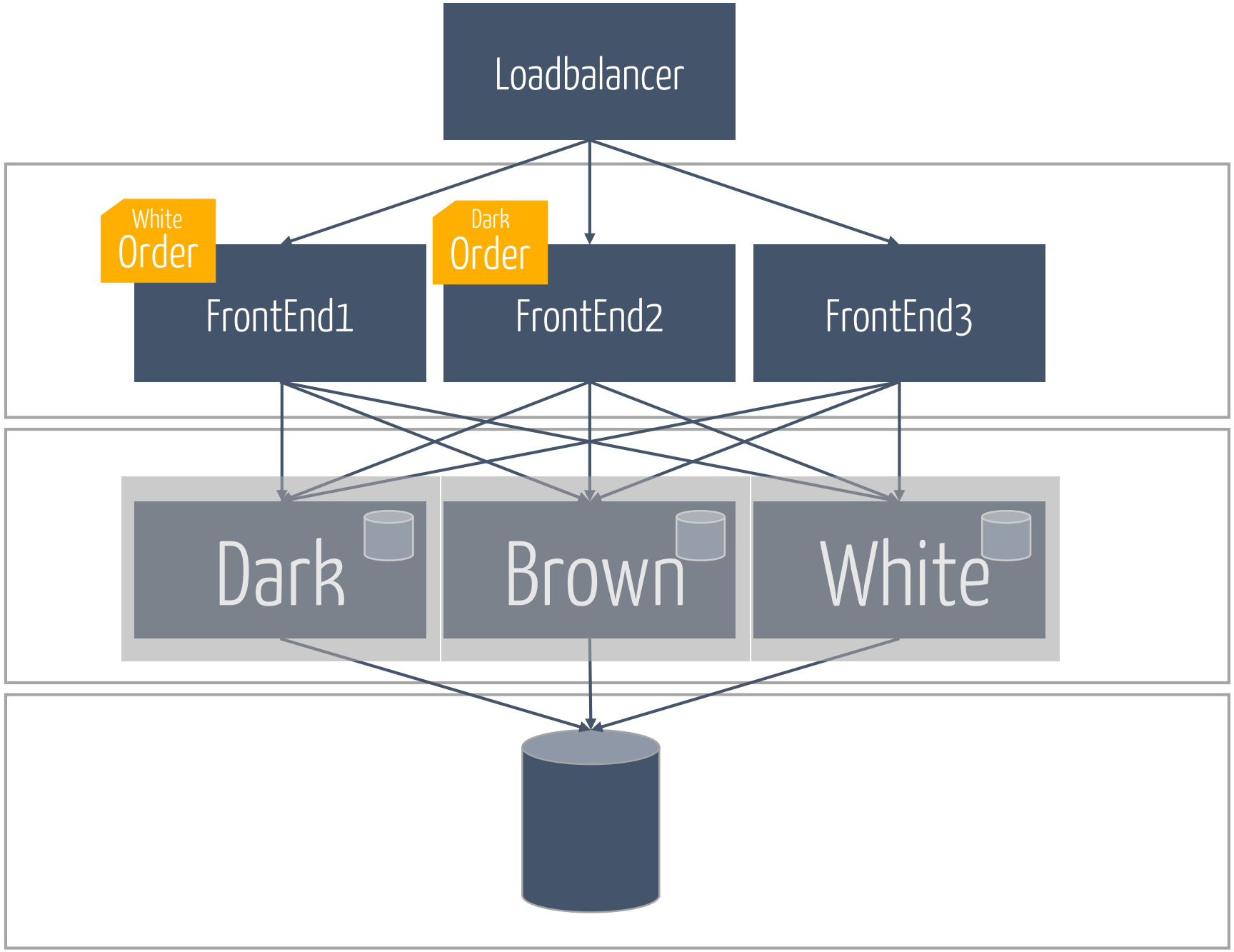
http://microservices.io/articles/scalecube.html


Loadbalancer
FrontEnd1
FrontEnd2
FrontEnd3
Dark
Brown
White


Loadbalancer
FrontEnd1
FrontEnd2
FrontEnd3
Dark
Brown
White

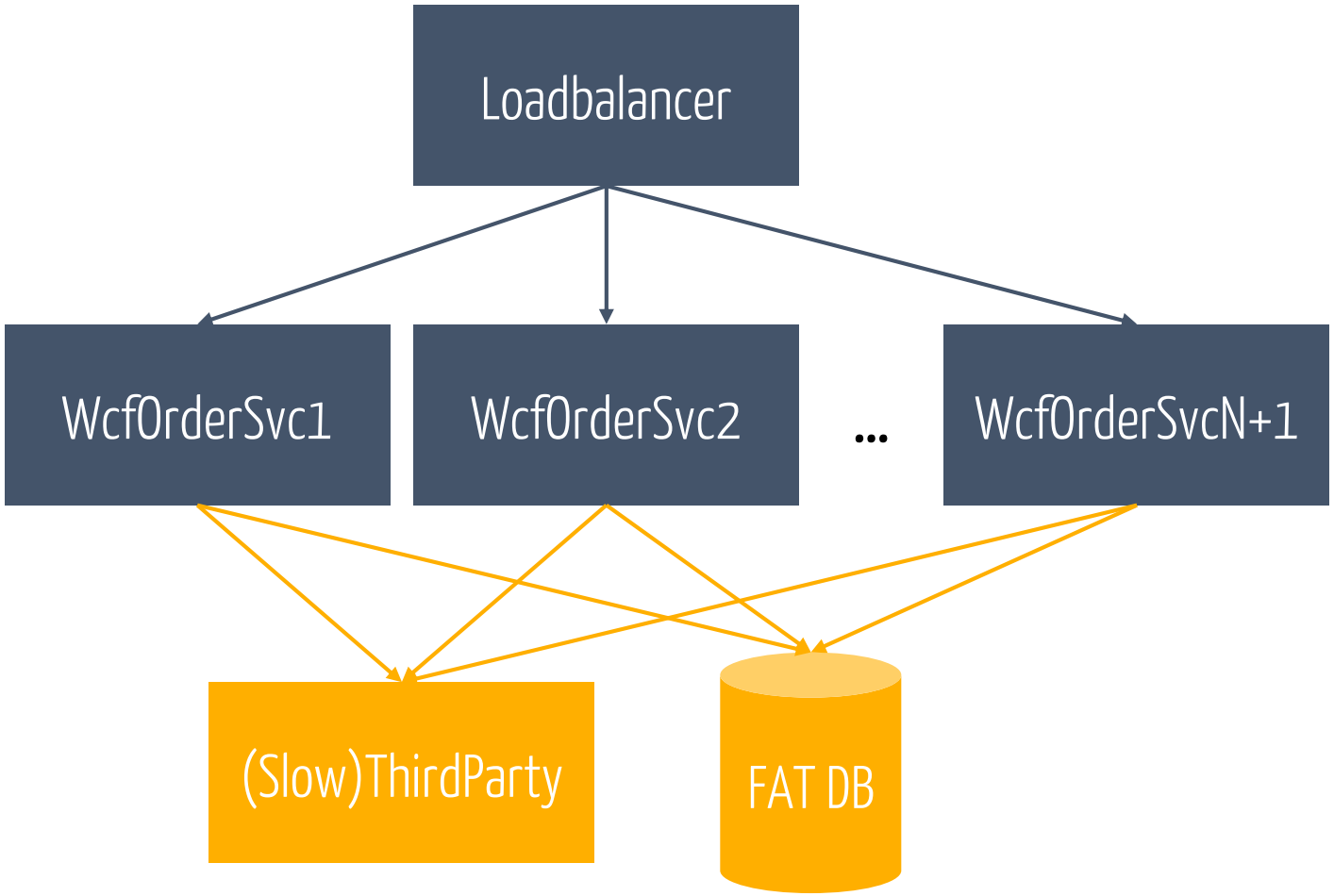
However

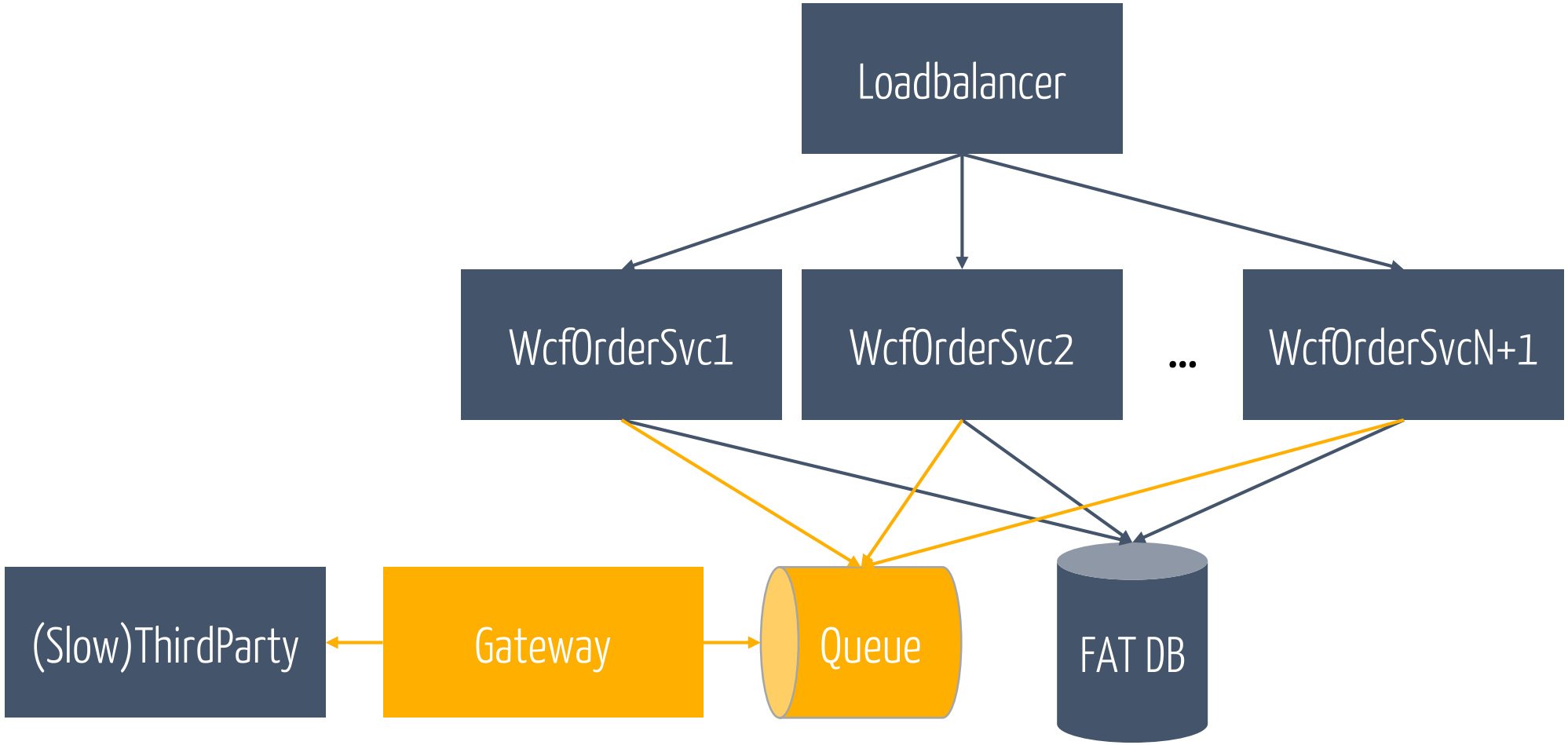
Joe couldn’t understand it

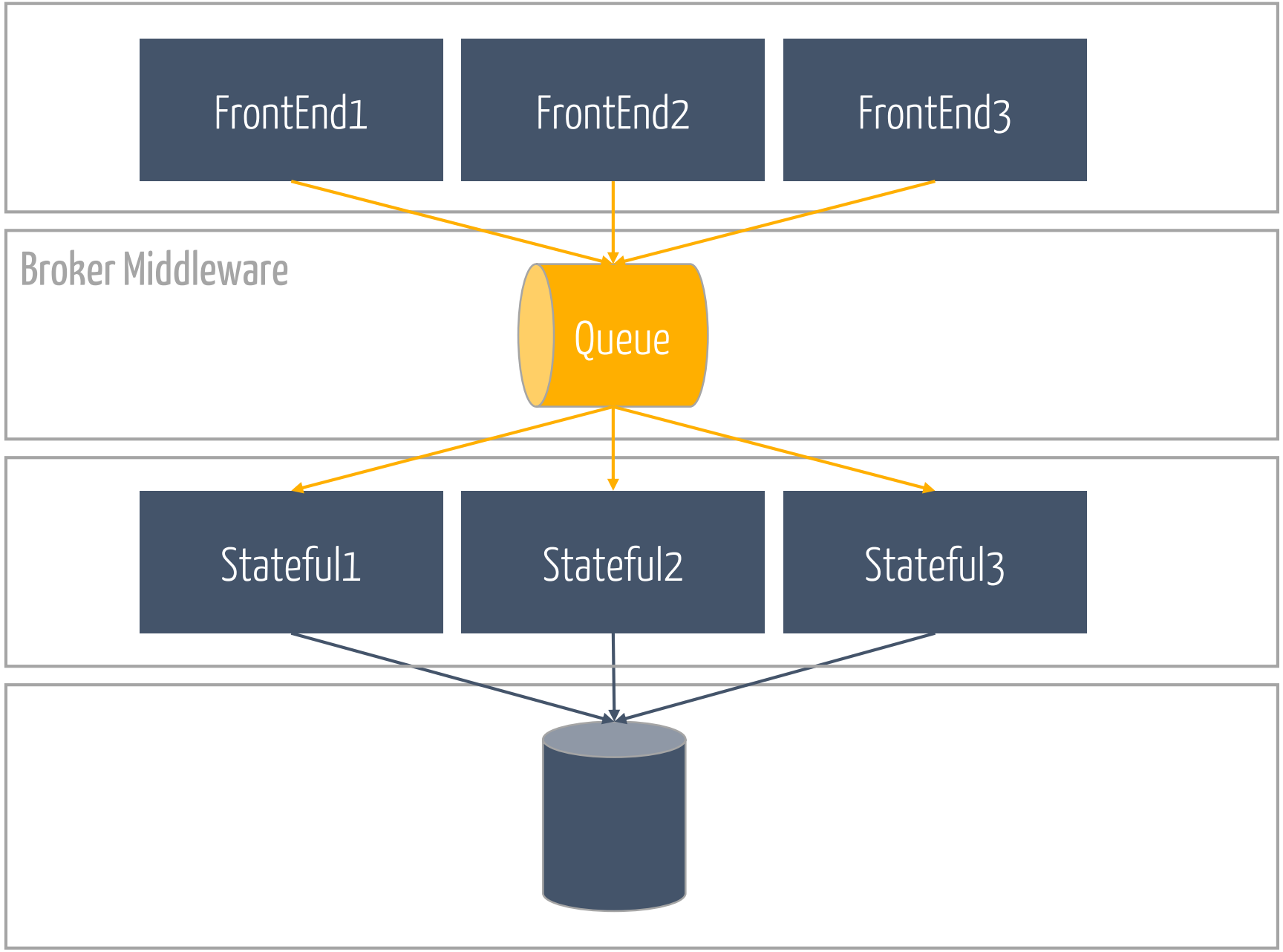

Loadbalancer
White
Order
Dark
Order
FrontEnd1
FrontEnd2
FrontEnd3
Dark
Brown
White

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

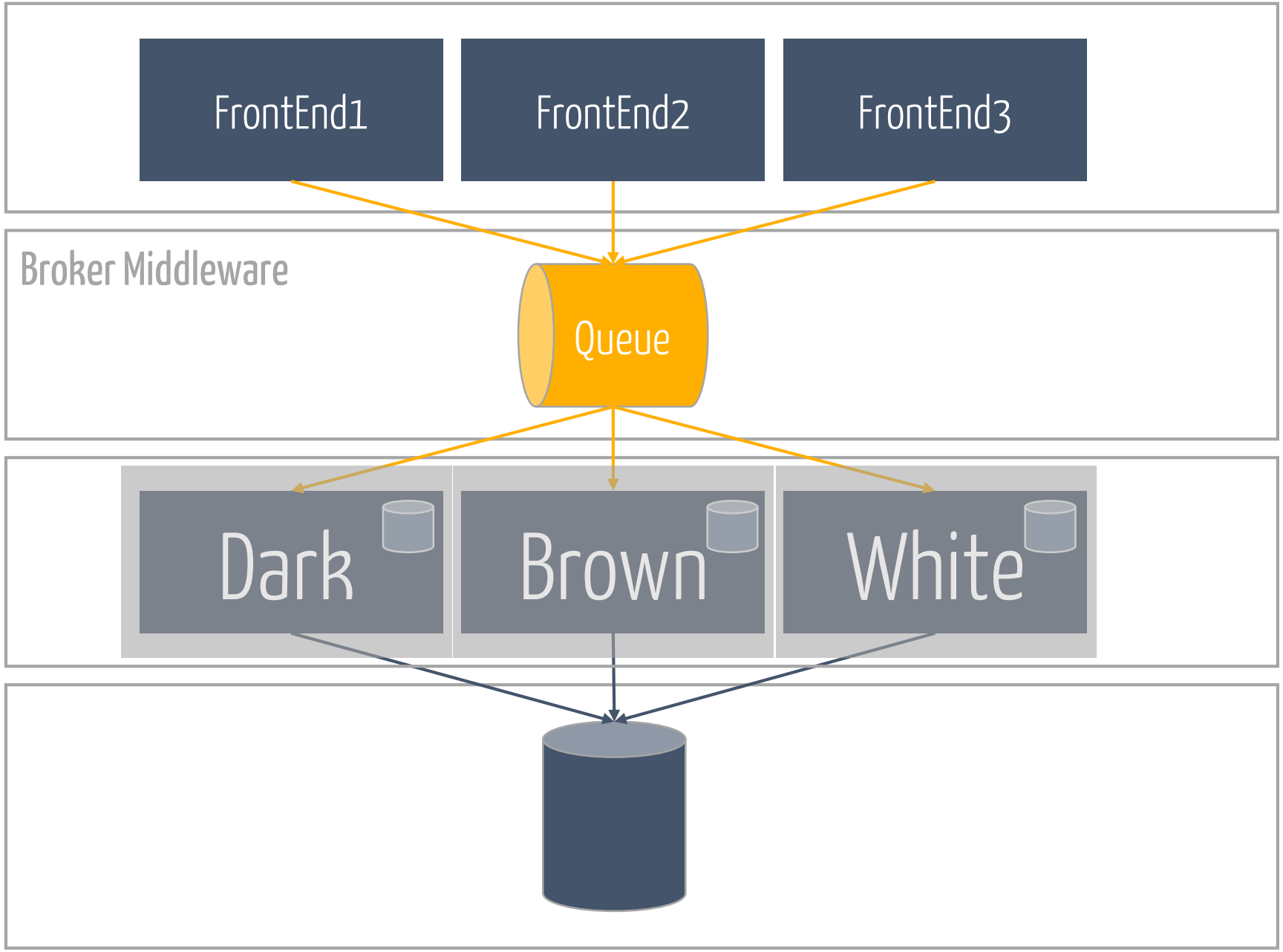

FrontEnd1
FrontEnd2
FrontEnd3
Broker Middleware
Queue
Stateful1
Stateful2
Stateful3

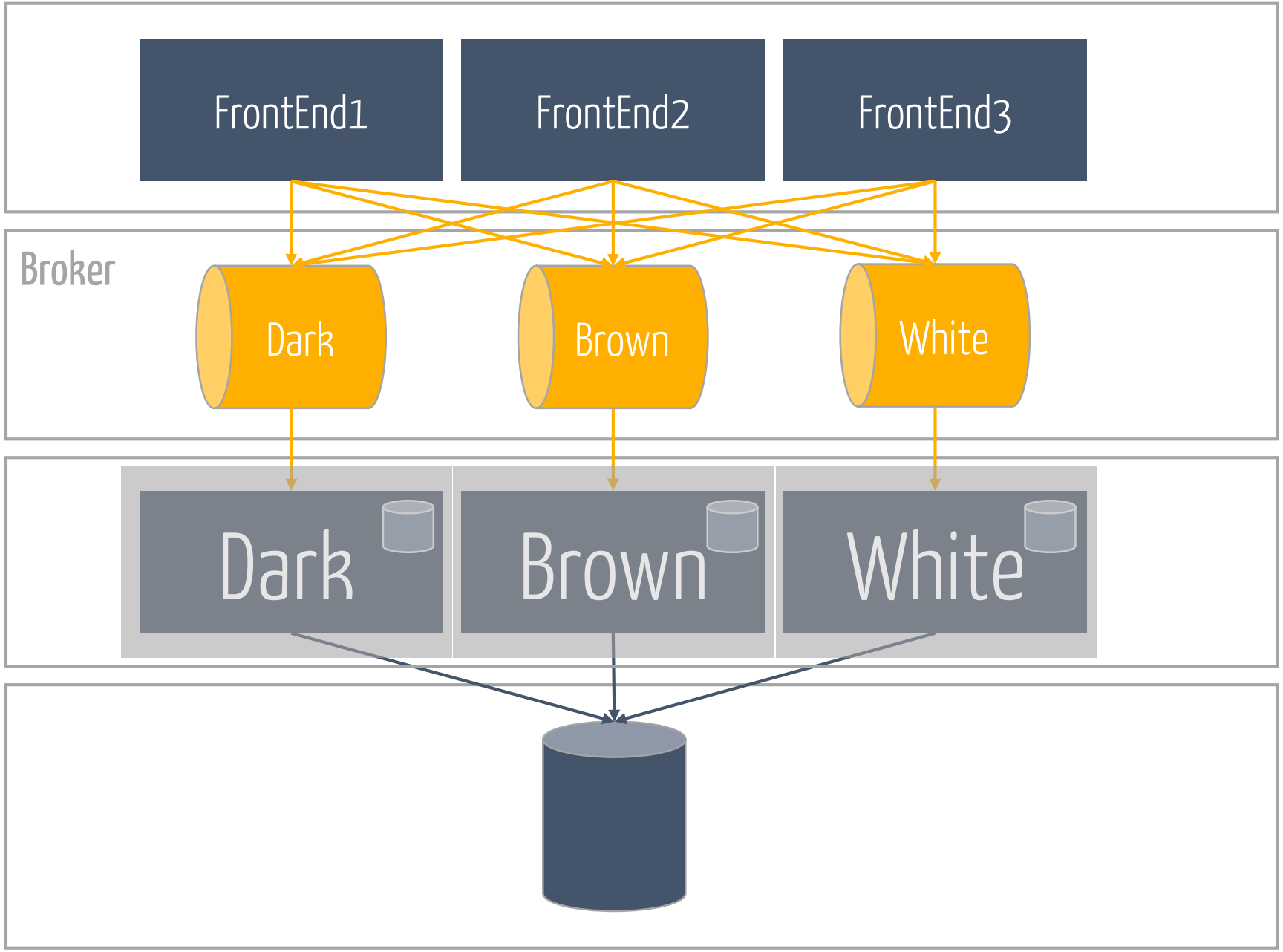
Peter snatches

the whiteboard markers

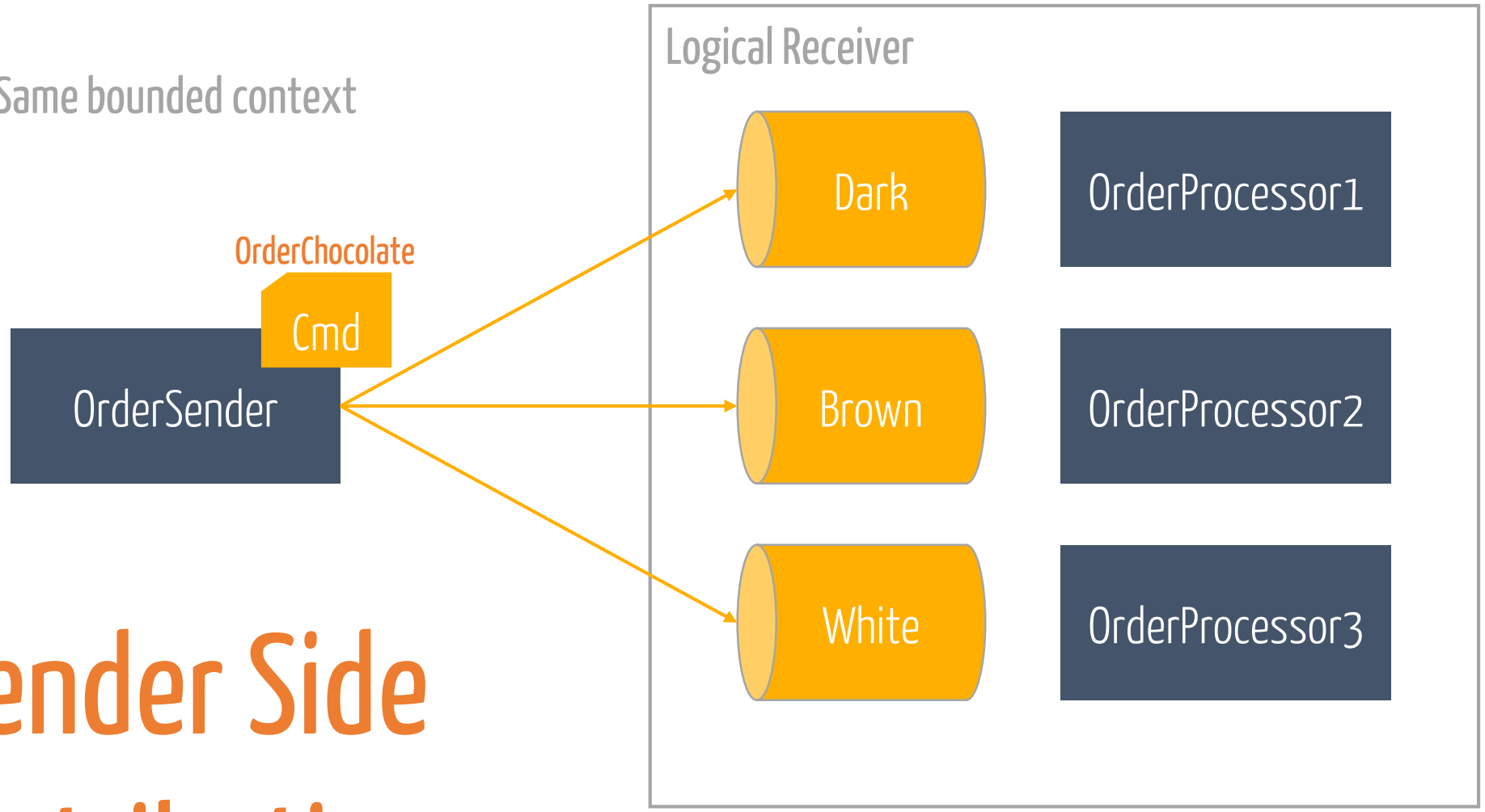
and furiously

screams


FrontEnd1
FrontEnd2
FrontEnd3
Broker Middleware
Queue
Dark
Brown
White


FrontEnd1
FrontEnd2
FrontEnd3
Broker
Dark
Brown
White
Dark
Brown
White

Commands

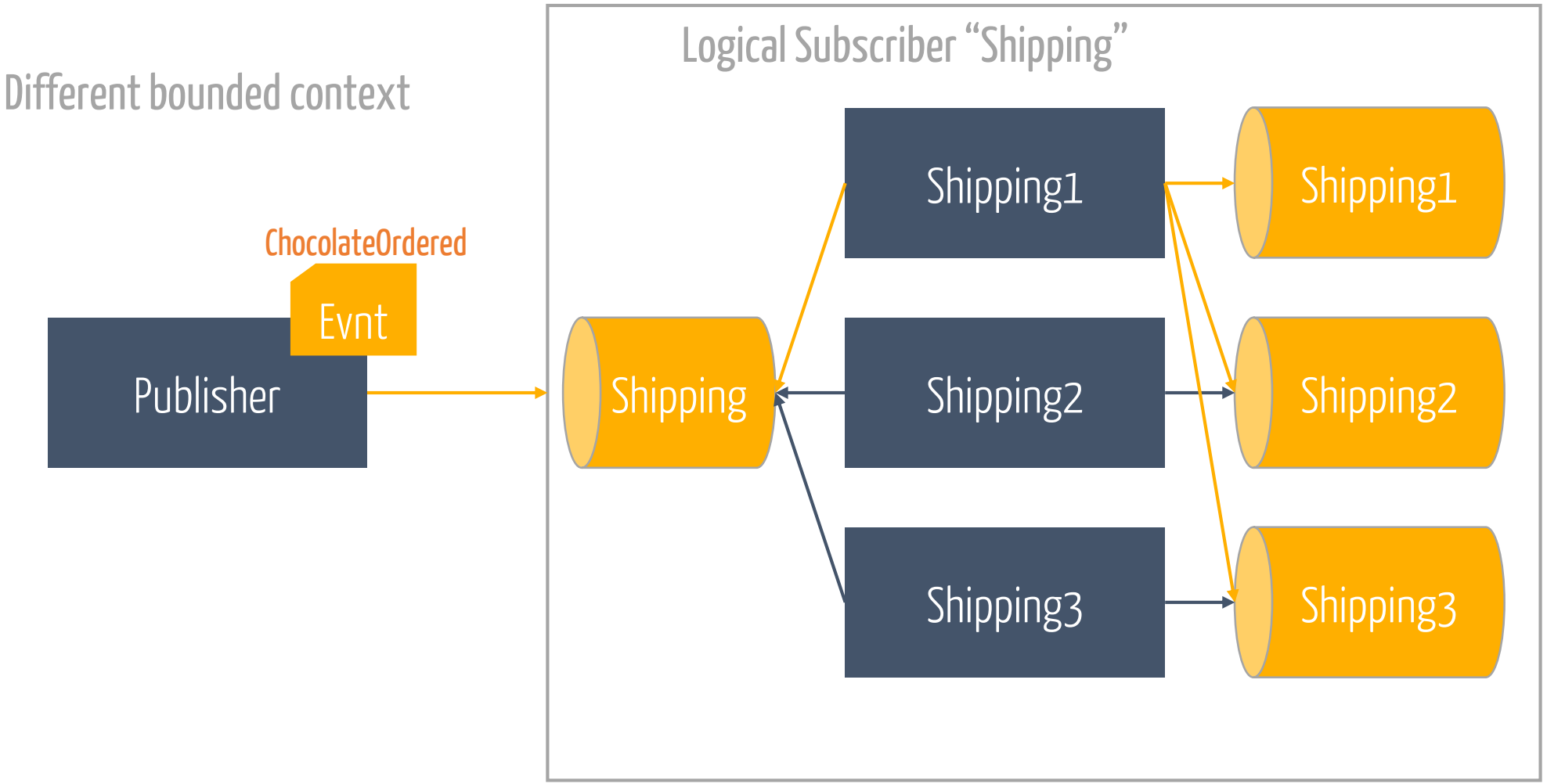


Sender Side Distribution

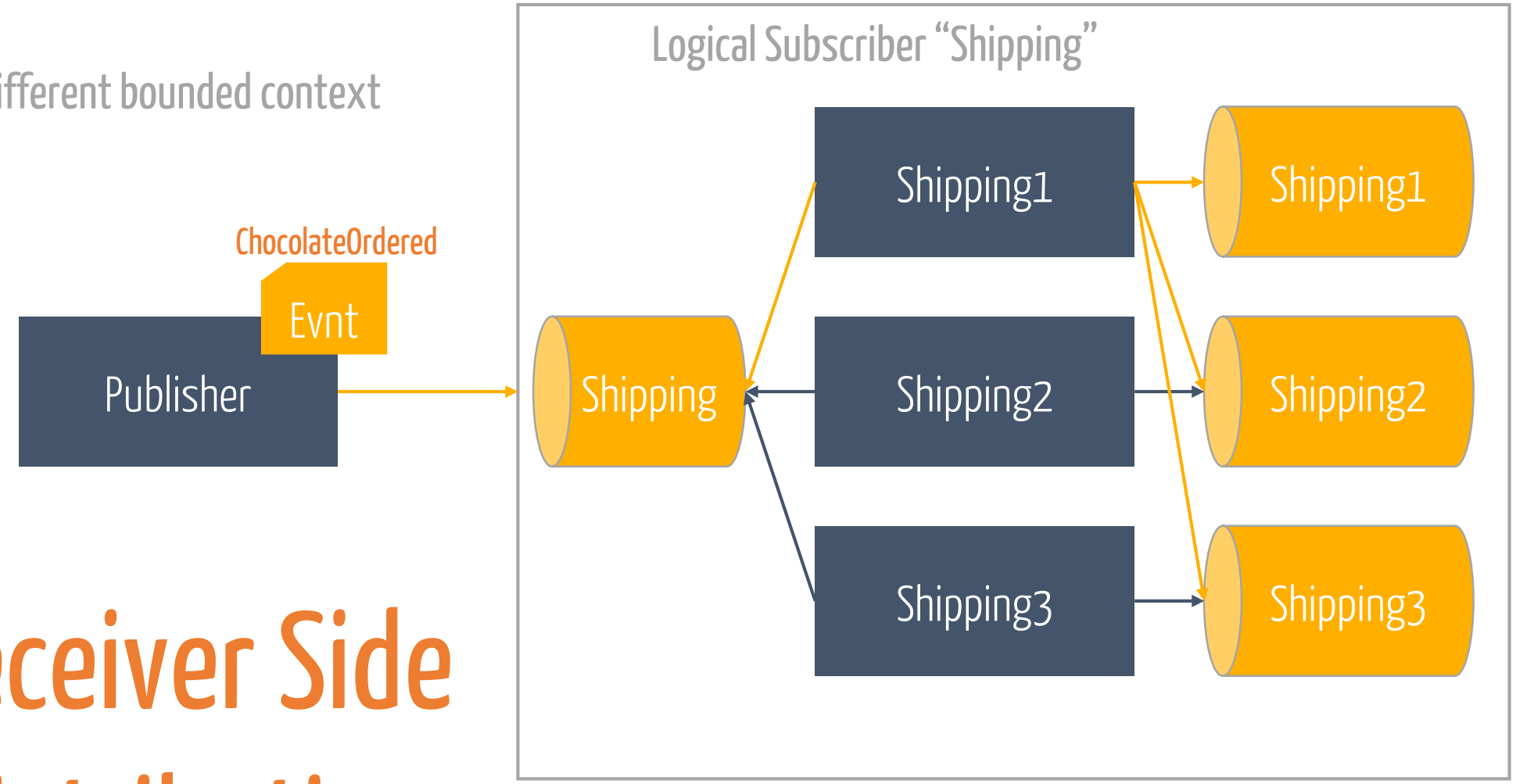
The PhD dude

acts like a smart ass

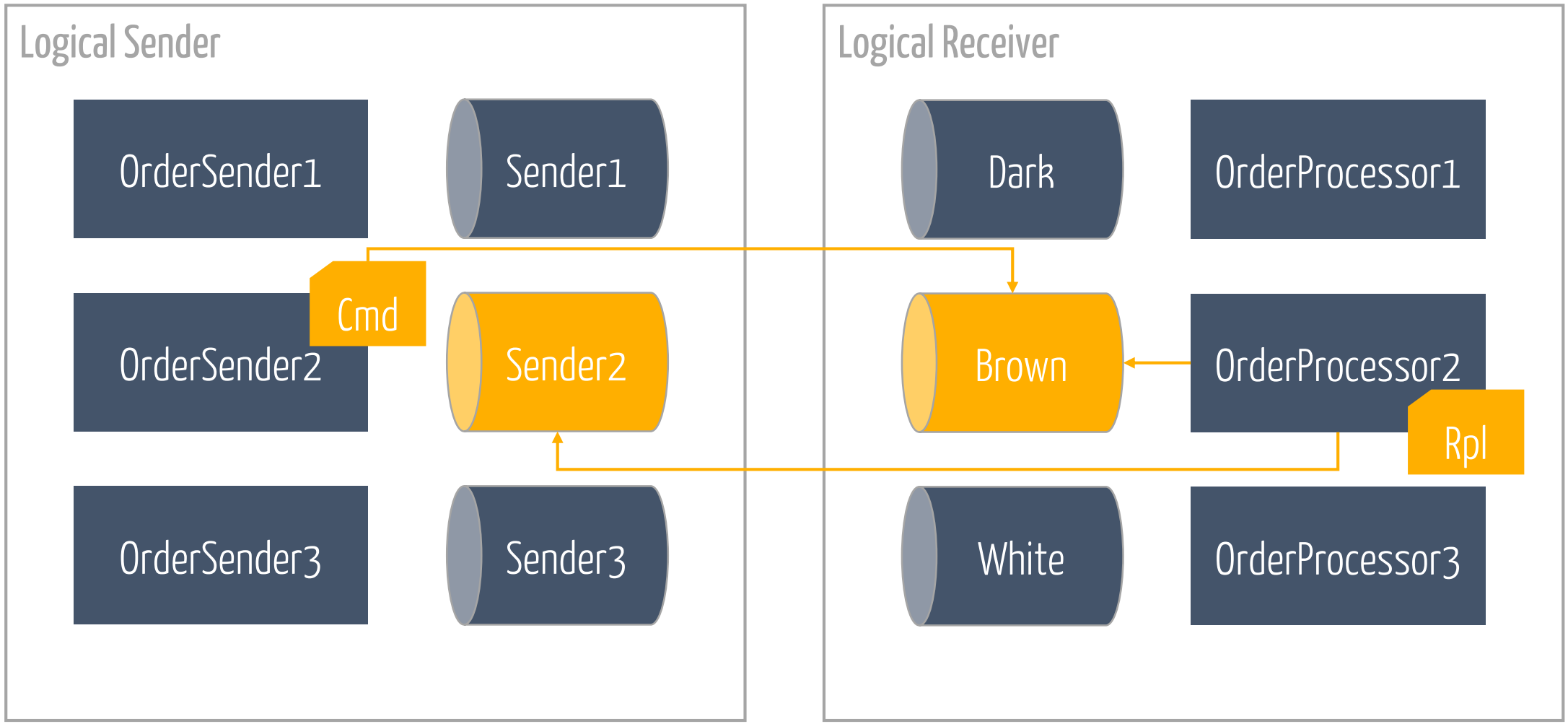
Events


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

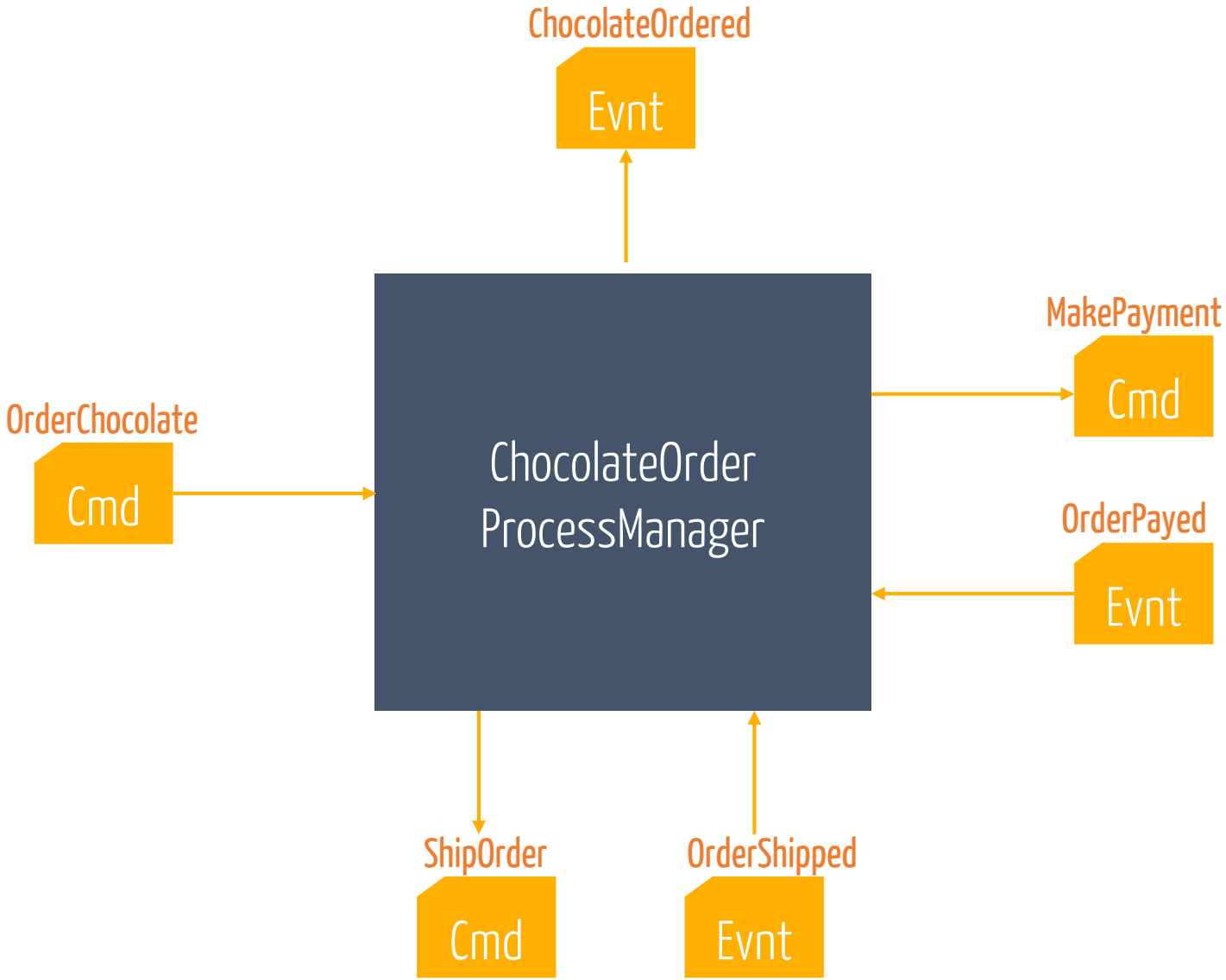
Receiver Side Distribution

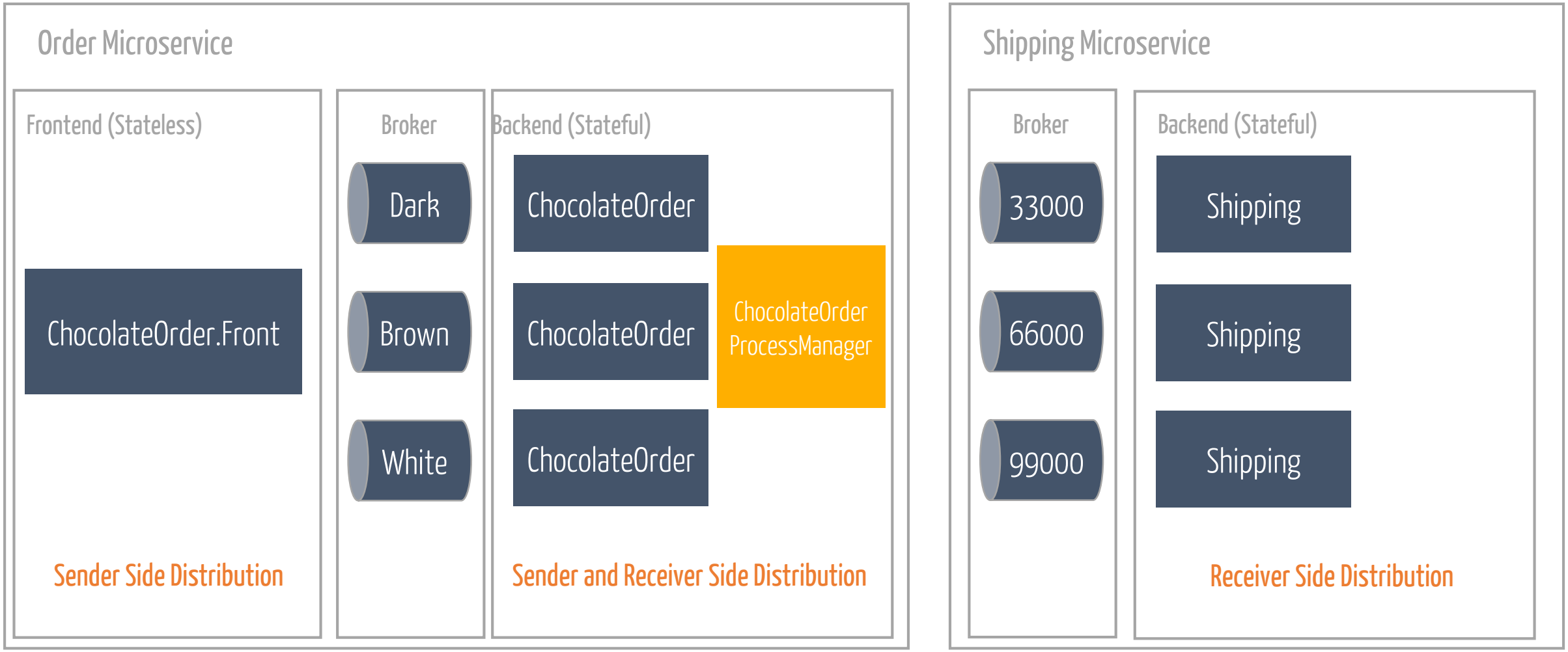


Request / Reply


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

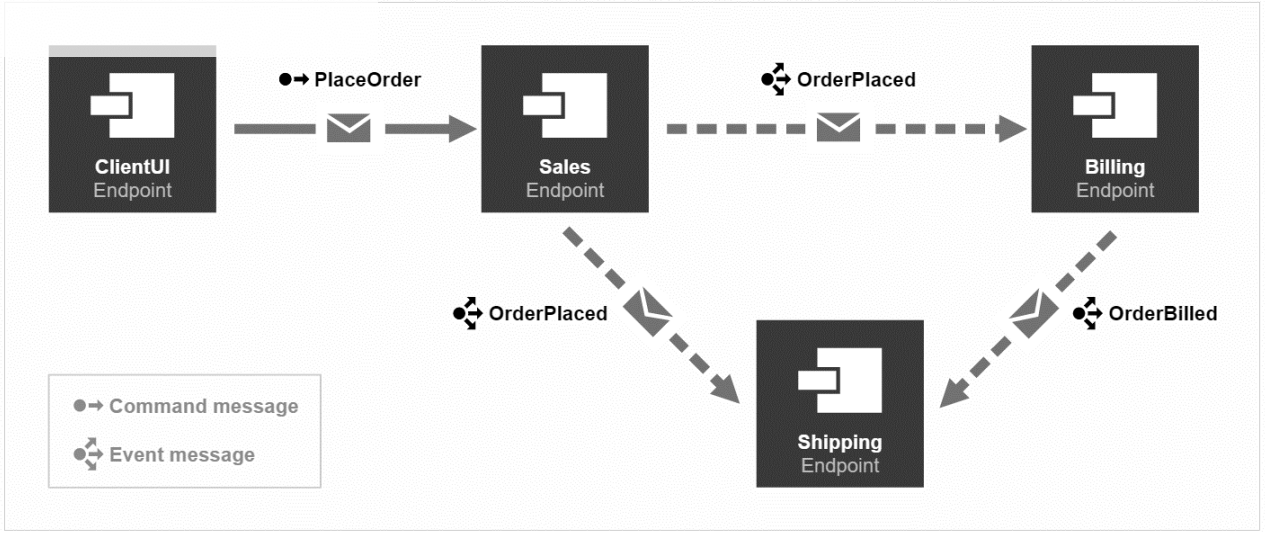
Introduction to NServiceBus

The best way to get started with NServiceBus is to use it to build something realistic. In doing so you'll learn the architectural concepts behind the software, and start to learn its capabilities. In this tutorial, you'll be building a back end for a retail e-commerce system. You'll learn how to send asynchronous messages between processes, how to use the Publish/Subscribe pattern to decouple business processes, and the advantages of using reliable messaging to enable automatic retries after processing failures.

The tutorial is divided into five lessons, each of which can be accomplished in a half hour or less — perfect for your lunch break.

- **Lesson 1: Getting started** (10-15 minutes) - learn how to set up your development environment and create your very first messaging endpoint.
- **Lesson 2: Sending a command** (15-20 minutes) - learn how to define messages and message handlers, and send your first message.
- **Lesson 3: Multiple endpoints** (15-20 minutes) - learn how to create multiple endpoints and send messages between them.
- **Lesson 4: Publishing events** (25-30 minutes) - learn about the Publish/Subscribe pattern, how to publish events to multiple subscribers, and about the benefits of using this pattern to decouple business processes.
- **Lesson 5: Retrying errors** (25-30 minutes) - learn how to use the Particular Service Platform tools to gracefully recover from exceptions in your code, allowing you to build systems that are resistant to failure.

When you've completed all the exercises, your solution will look like this:



docs.particular.net/
tutorials/intro-to-nservicebus

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.

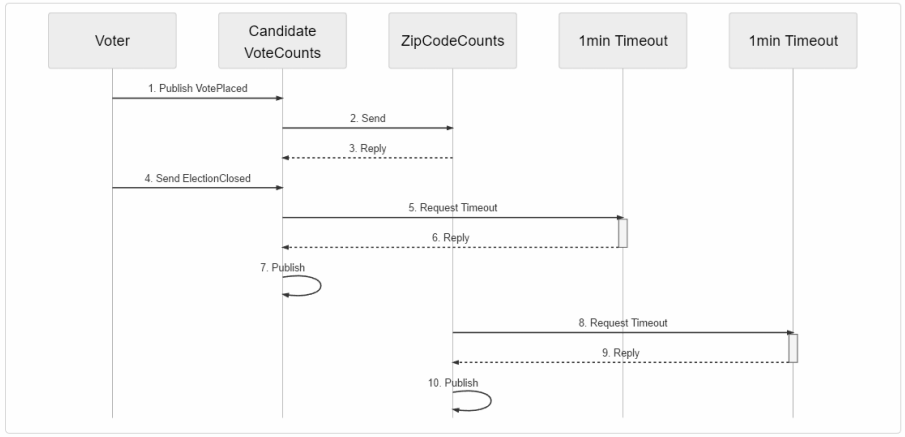
When the election is closed, the candidate vote counting endpoint will `publish` the results per candidate and report them using Service Fabric diagnostics infrastructure (ETW/ Event Viewer windows).

[illegible] timeout [illegible] zip code counting endpoint `sends a local command` to report the [illegible] zip code.

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.

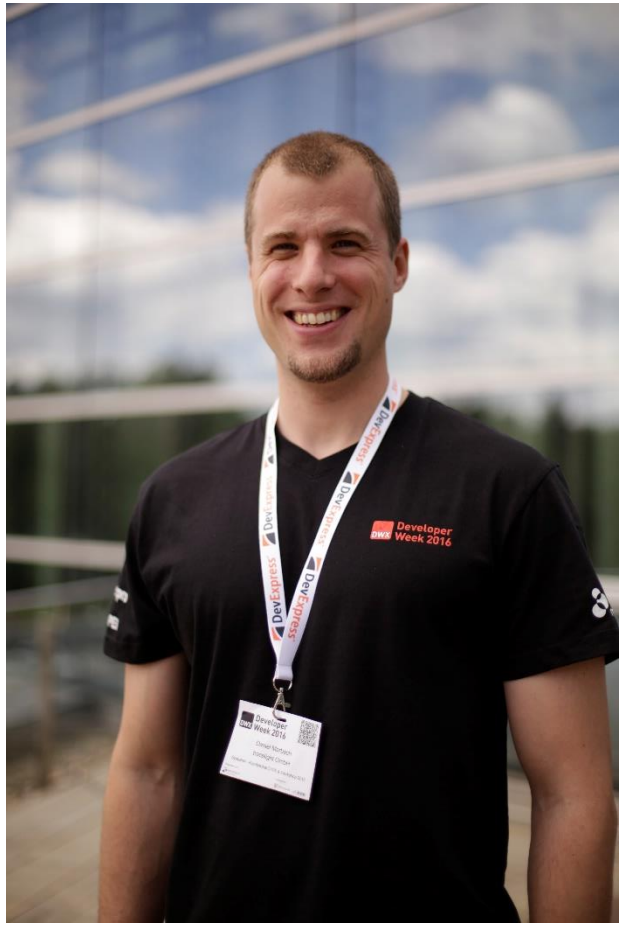


Slides, Links...

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