Particular
Software

Knock. Knock. Who's there?

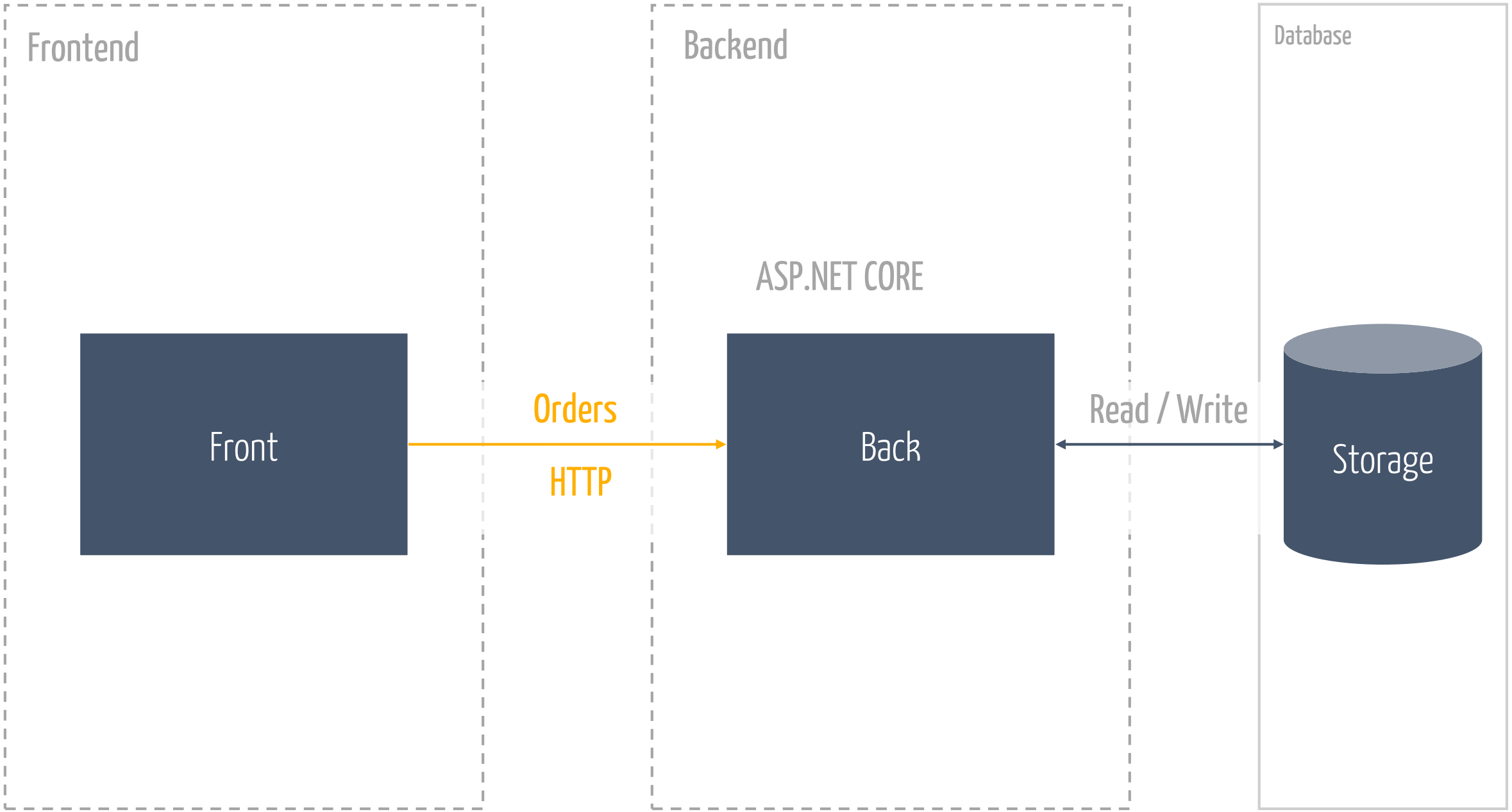
A message from the future


EXIT

Hiding Inconsistencies

Multi user collaboration

Multi user collaboration


Frontend
Backend
Database
ASP.NET CORE
Front
Orders
HTTP
Back
Read / Write
Storage

```
public IActionResult Post([FromBody] Order order)
{
    var totalOfAllOrdersOfLastWeek = Database
        .OrdersOfLastWeekFor(order.CustomerId)
        .Sum(o => o.Total);

    var discount = 0m;
    if (totalOfAllOrdersOfLastWeek >= 500)
    {
        discount = 0.1m;
    }

    order.Total -= order.Total * discount;

    Database.Save(order);

    return Ok(order);
}
```

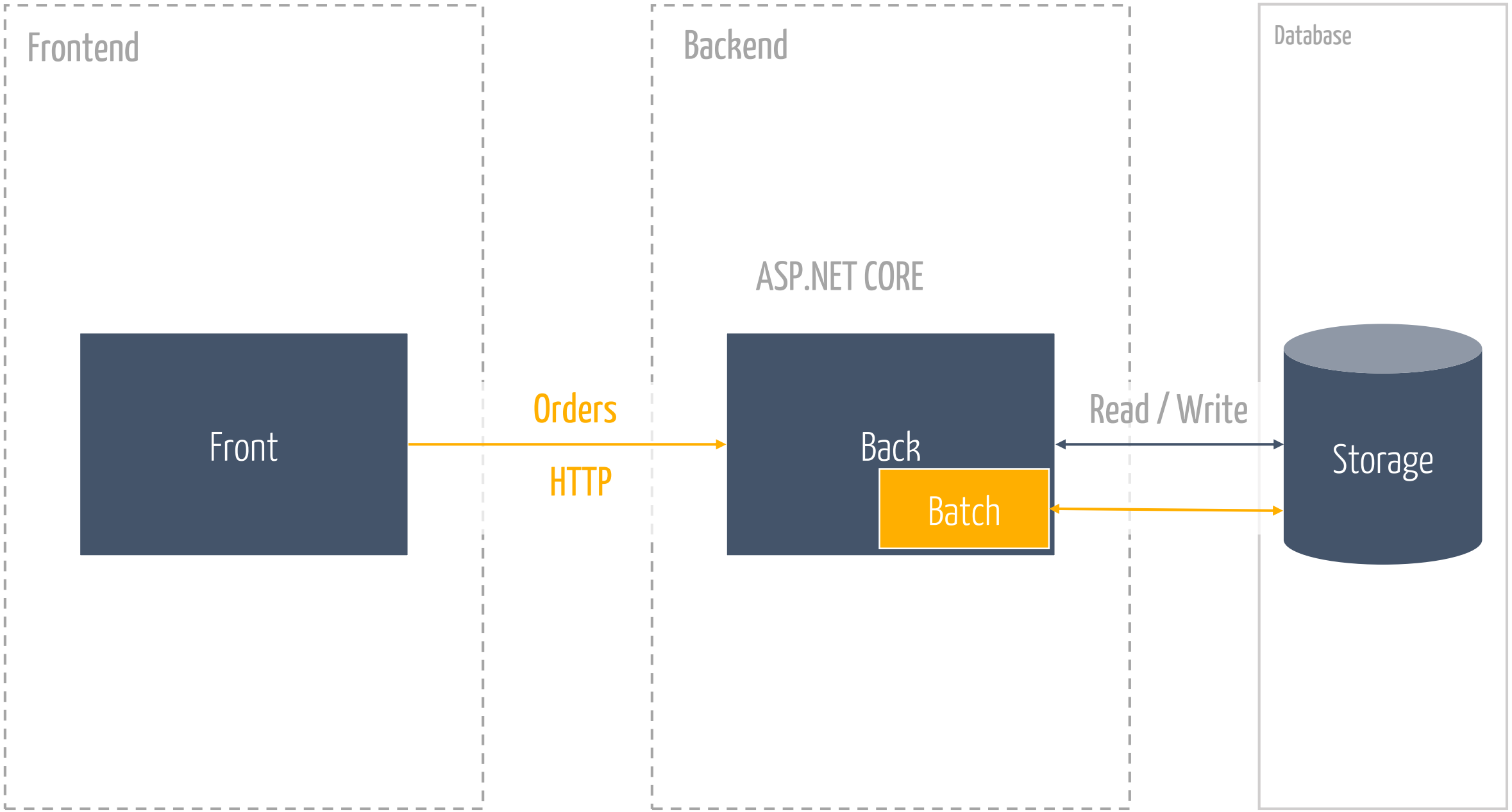
```
Daniel@DESKTOP-SUIHDT7 c:\p\KnockKnock\demo1
> docker-compose up
```

And then the customer walks into your office


4GIFs.com

Concurrency and latency in collaborative domains can lead to incorrect application of business rules even in safe architectures

Batch Jobs

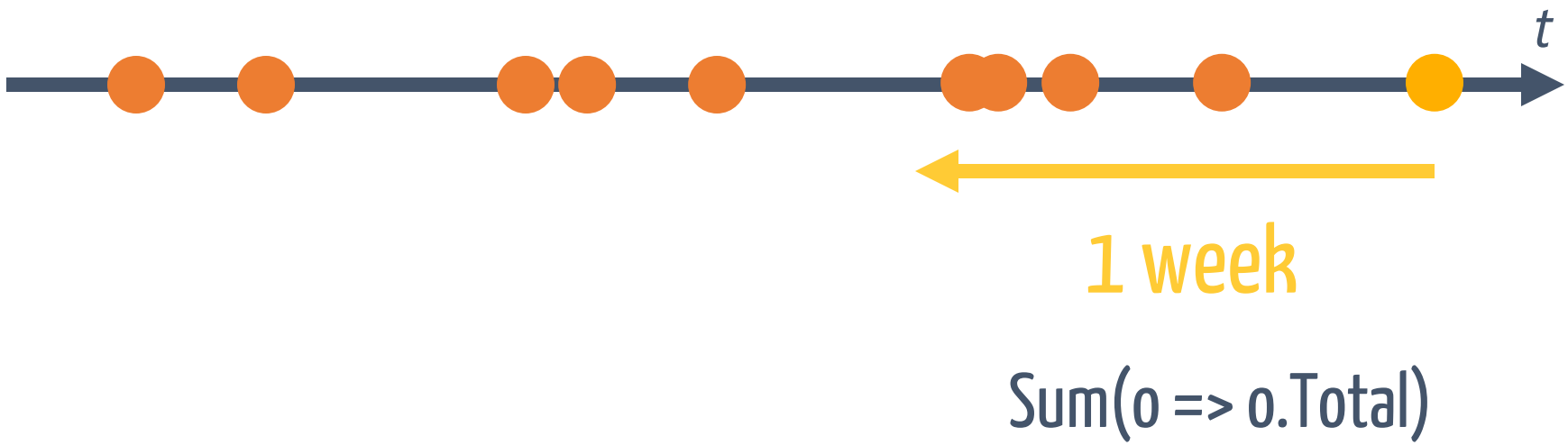

Frontend
Backend
Database
ASP.NET CORE
Front
Orders
HTTP
Back
Batch
Read / Write
Storage

```
Daniel@DESKTOP-SUIHDT7 c:\p\KnockKnock\demo2
> docker-compose up
```

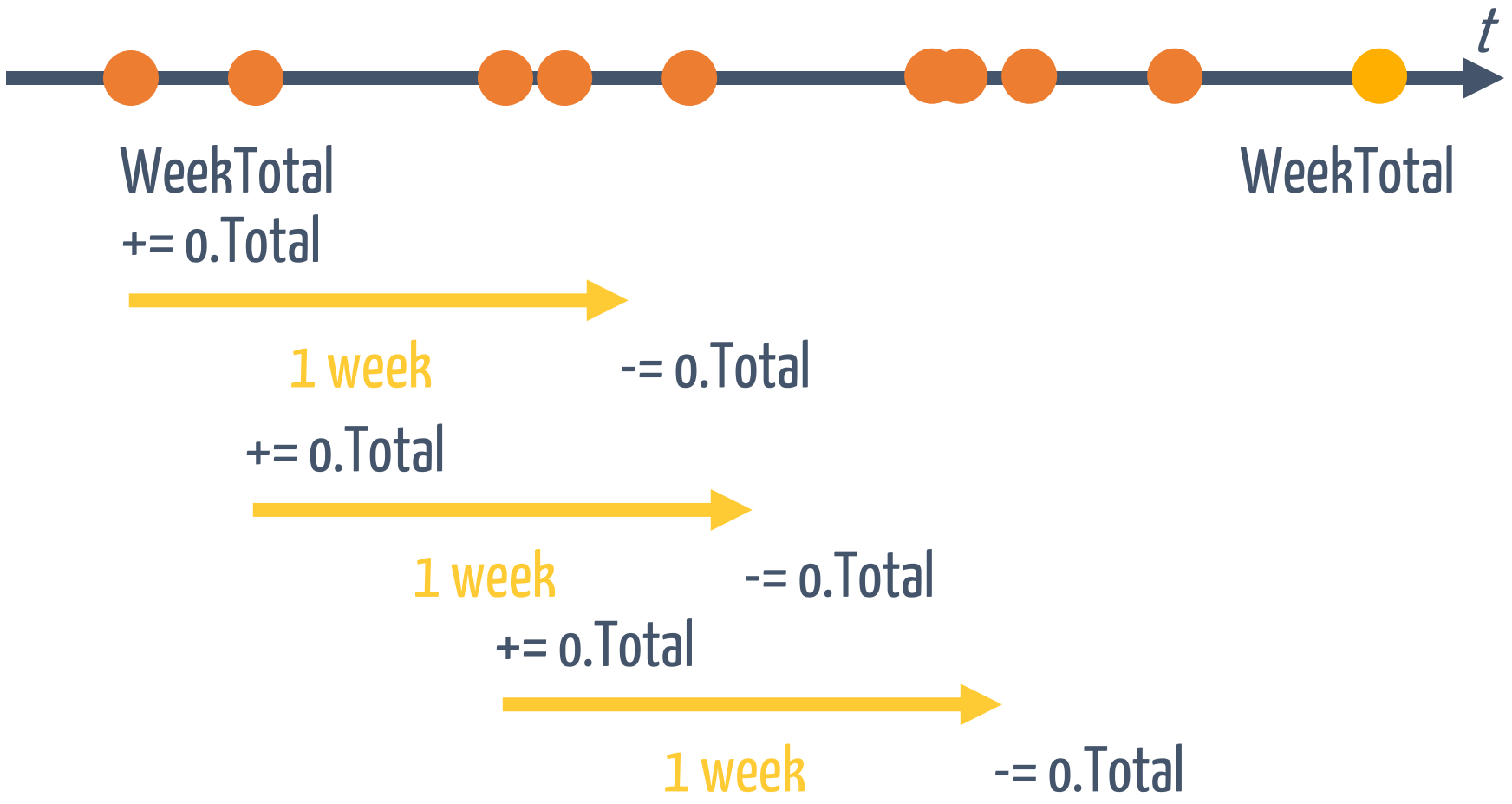
Batch Jobs increase the window of consistency problems

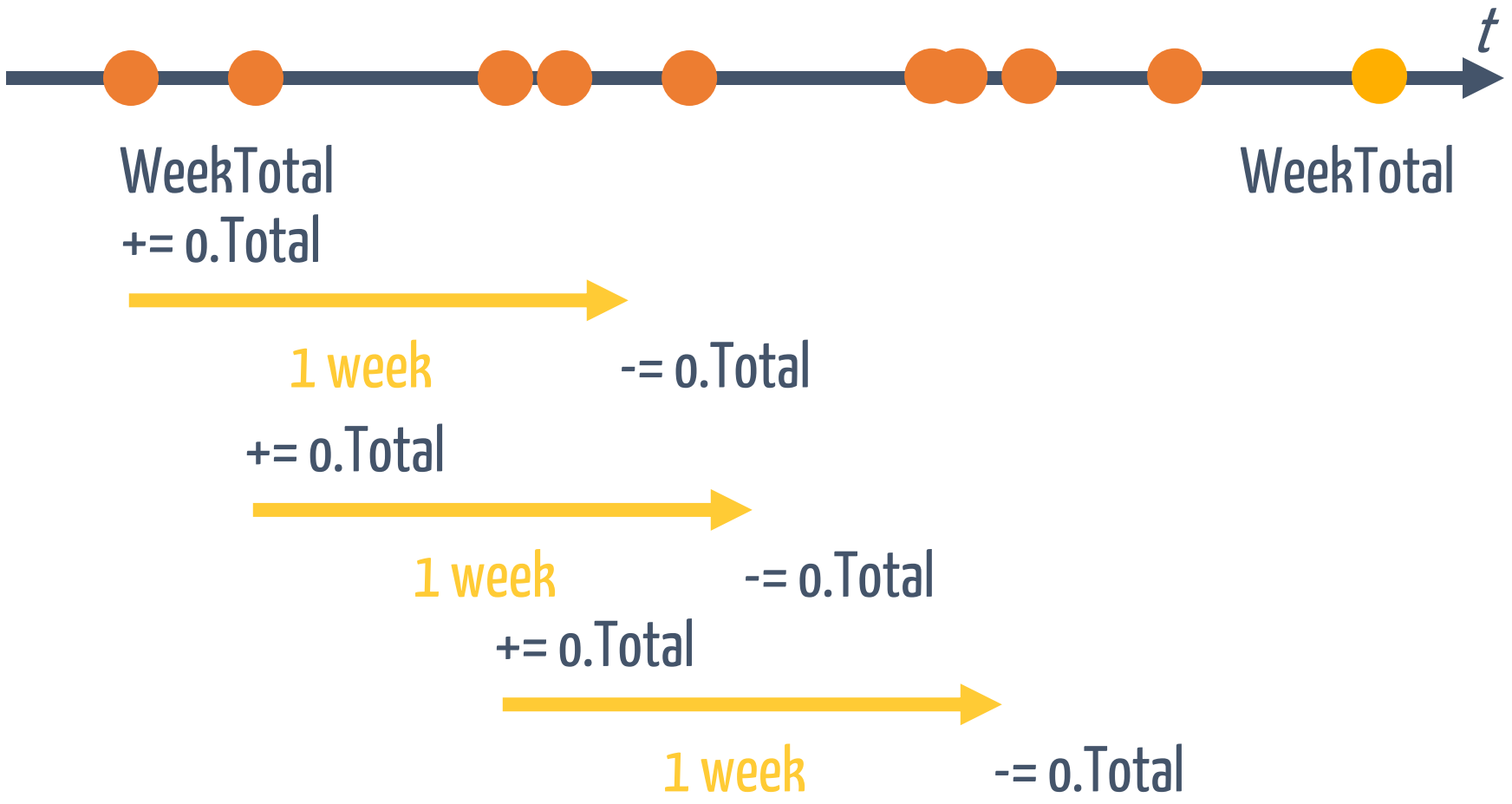
Batch Jobs for time based business rules are the worst enemy for growth


Rethink time

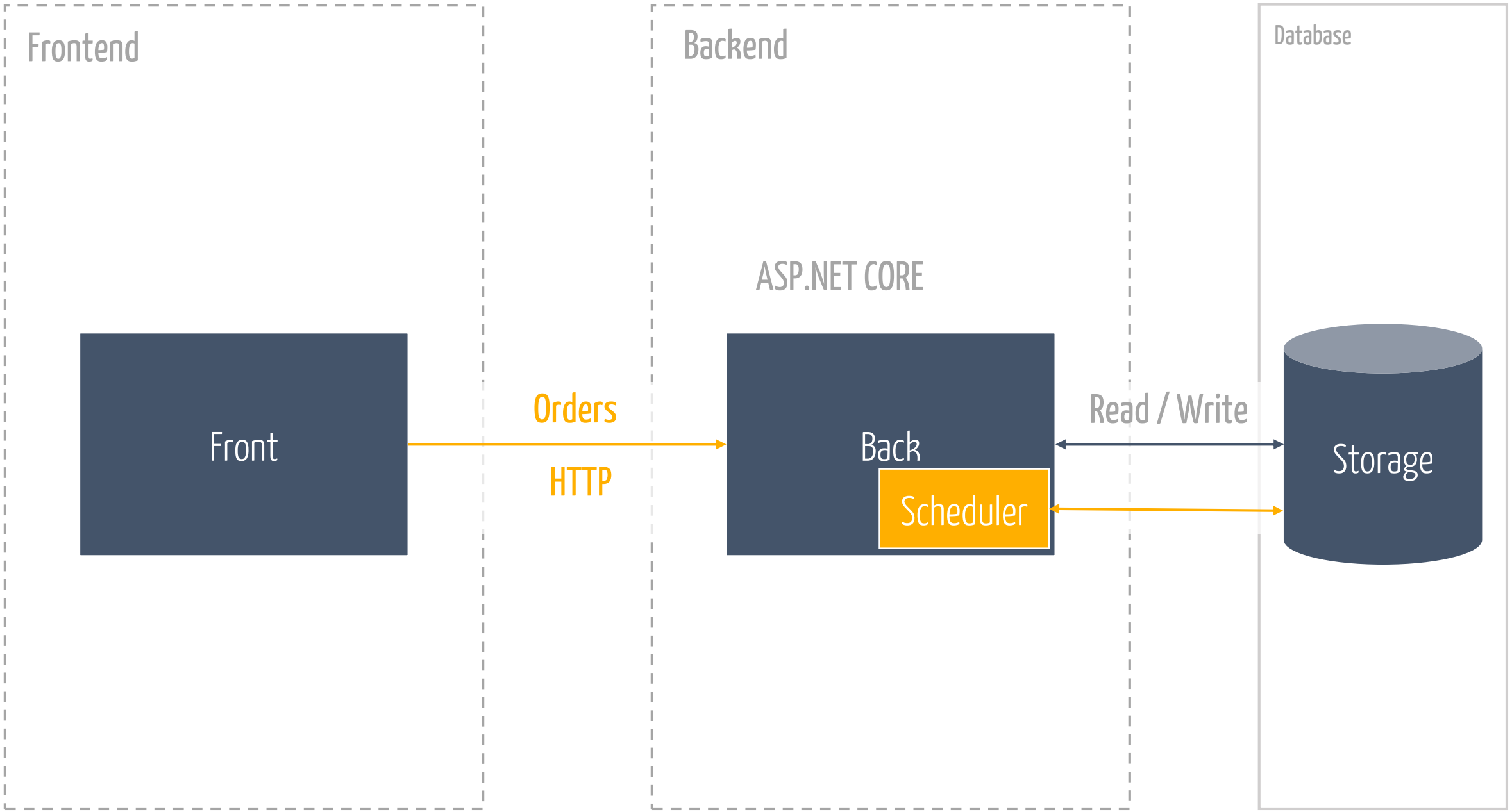


Sum(o => o.Total)


t
WeekTotal
+= o.Total
WeekTotal
1 week
-= o.Total
+= o.Total
1 week
-= o.Total
+= o.Total
1 week
-= o.Total


t
WeekTotal
+= o.Total
WeekTotal
1 week
-= o.Total
+= o.Total
1 week
-= o.Total
+= o.Total
1 week
-= o.Total

Program time


Frontend
Backend
Database
ASP.NET CORE
Front
Orders
HTTP
Back
Scheduler
Read / Write
Storage

```
User@BF108680 c:\p\KnockKnock\demo3
$ docker-compose up
```

```
User@BF108680 c:\p\KnockKnock\demo3
$ docker-compose up
Starting demo3_orders.backend.future_1 ... done
Starting demo3_orders.frontend.future_1 ... done
Attaching to demo3_orders.backend.future_1, demo3_orders.frontend.future_1
orders.backend.future_1    | Hosting environment: PRODUCTION
orders.backend.future_1    | Content root path: /app
orders.backend.future_1    | Now listening on: http://[::]:8080
orders.backend.future_1    | Application started. Press Ctrl+C to shut down.
orders.frontend.future_1   | Ready
orders.frontend.future_1   |
orders.frontend.future_1   | Order #1: Value 300 for customer 6521547
orders.frontend.future_1   | Order #1: No discount
orders.frontend.future_1   | Order #2: Value 300 for customer 6521547
orders.frontend.future_1   | Order #2: No discount
orders.frontend.future_1   | Order #3: Value 300 for customer 6521547
orders.frontend.future_1   | Order #3: Got a discount of 30.00
orders.frontend.future_1   | Order #4: Value 300 for customer 6521547
orders.frontend.future_1   | Order #4: Got a discount of 30.00
orders.frontend.future_1   | Order #5: Value 300 for customer 6521547
orders.frontend.future_1   | Order #5: Got a discount of 30.00
```

Durable scheduling introduces reliability on the server side

Resilient HTTP introduces some reliability
but doesn't survive restarts

Client side retries help to resolve transient failures but increase the latency

Orders might be lost when clients give up on retries

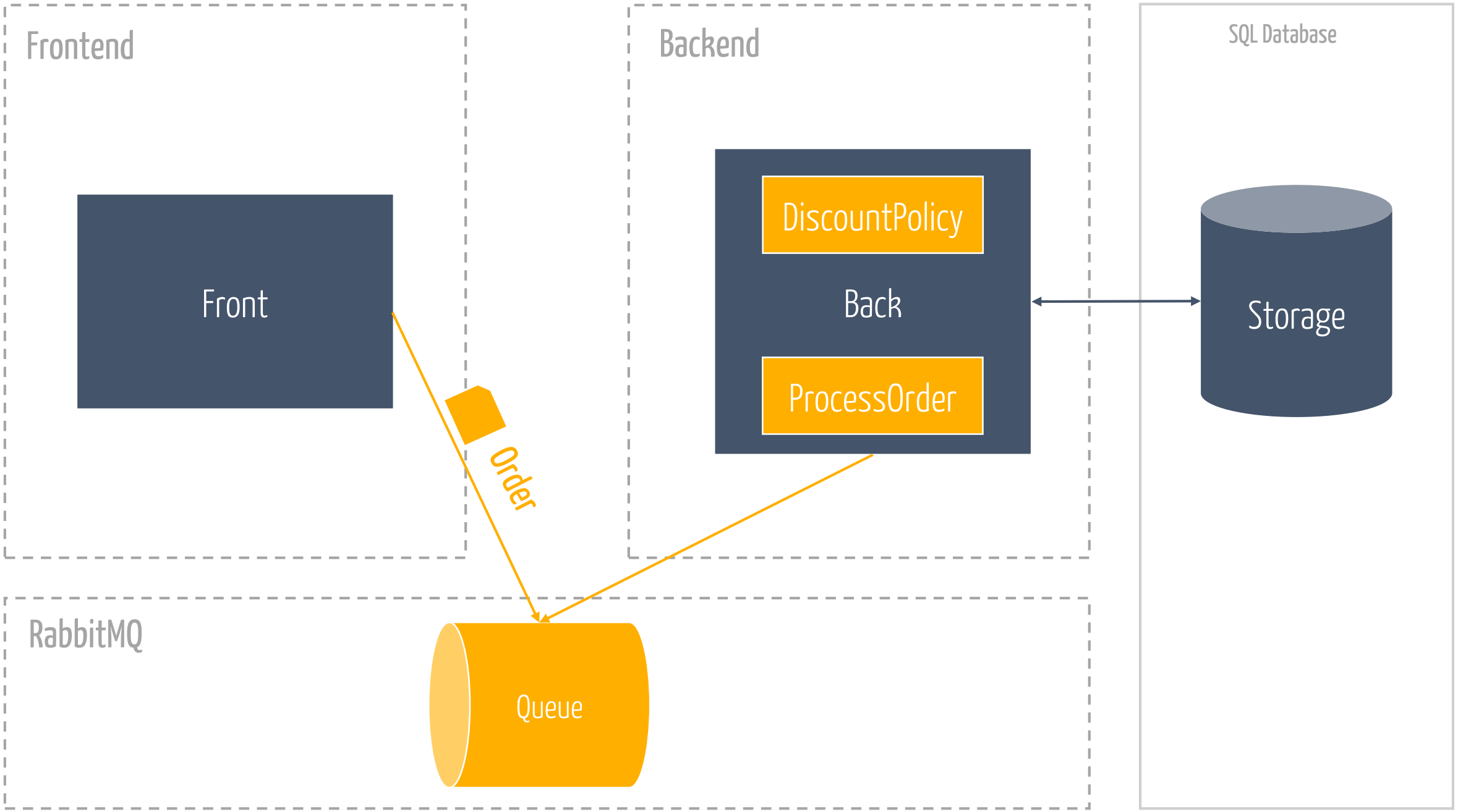
Orders are ideal to

Fire & forget

from the frontend perspective

Sagas

Sagas


Frontend
Front
Order
Backend
DiscountPolicy
Back
ProcessOrder
SQL Database
Storage
RabbitMQ
Queue

```
0 references
public class DiscountPolicy : Saga<DiscountPolicyData>,
    IAmStartedByMessages<SubmitOrder>,
    IHandleTimeouts<SubmitOrder>
{
    0 references
    public async Task Handle(SubmitOrder message, IMessageHandlerContext context)
    {
        Data.CustomerId = message.CustomerId;
        var totalOfAllOrdersOfLastWeek = Data.RunningTotal;
        Data.RunningTotal += message.Total;

        var discount = 0m;
        if (totalOfAllOrdersOfLastWeek >= 500)
        {
            discount = 0.1m;
        }

        await context.SendLocal(new ProcessOrder {
            DiscountedTotal = message.Total - (message.Total * discount),
            CustomerId = message.CustomerId,
            OrderNumber = message.OrderNumber,
            Total = message.Total,
        });

        await RequestTimeout(context, Schedule.InAWeek, message);
    }

    0 references
    public Task Timeout(SubmitOrder state, IMessageHandlerContext context)
    {
        Data.RunningTotal -= state.Total;
        #region
        Console.WriteLine($"Decreased running total of {state.CustomerId.Short()} by {state.Total}");
        #endregion
        return Task.CompletedTask;
    }

    0 references
    protected override void ConfigureHowToFindSaga(SagaPropertyMapper<DiscountPolicyData> mapper)
    {
        mapper.ConfigureMapping<SubmitOrder>(m => m.CustomerId).ToSaga(s => s.CustomerId);
    }
}
```

```
0 references
public class DiscountPolicy : Saga<DiscountPolicyData>,
    IAmStartedByMessages<SubmitOrder>,
    IHandleTimeouts<SubmitOrder>
{
    0 references
    public async Task Handle(SubmitOrder message, IMessageHandlerContext context)
    {
        Data.CustomerId = message.CustomerId;
        var totalOfAllOrdersOfLastWeek = Data.RunningTotal;
        Data.RunningTotal += message.Total;

        var discount = 0m;
        if (totalOfAllOrdersOfLastWeek >= 500)
        {
            discount = 0.1m;
        }

        await context.SendLocal(new ProcessOrder {
            DiscountedTotal = message.Total - (message.Total * discount),
            CustomerId = message.CustomerId,
            OrderNumber = message.OrderNumber,
            Total = message.Total,
        });

        await RequestTimeout(context, Schedule.InAWeek, message);
    }
```

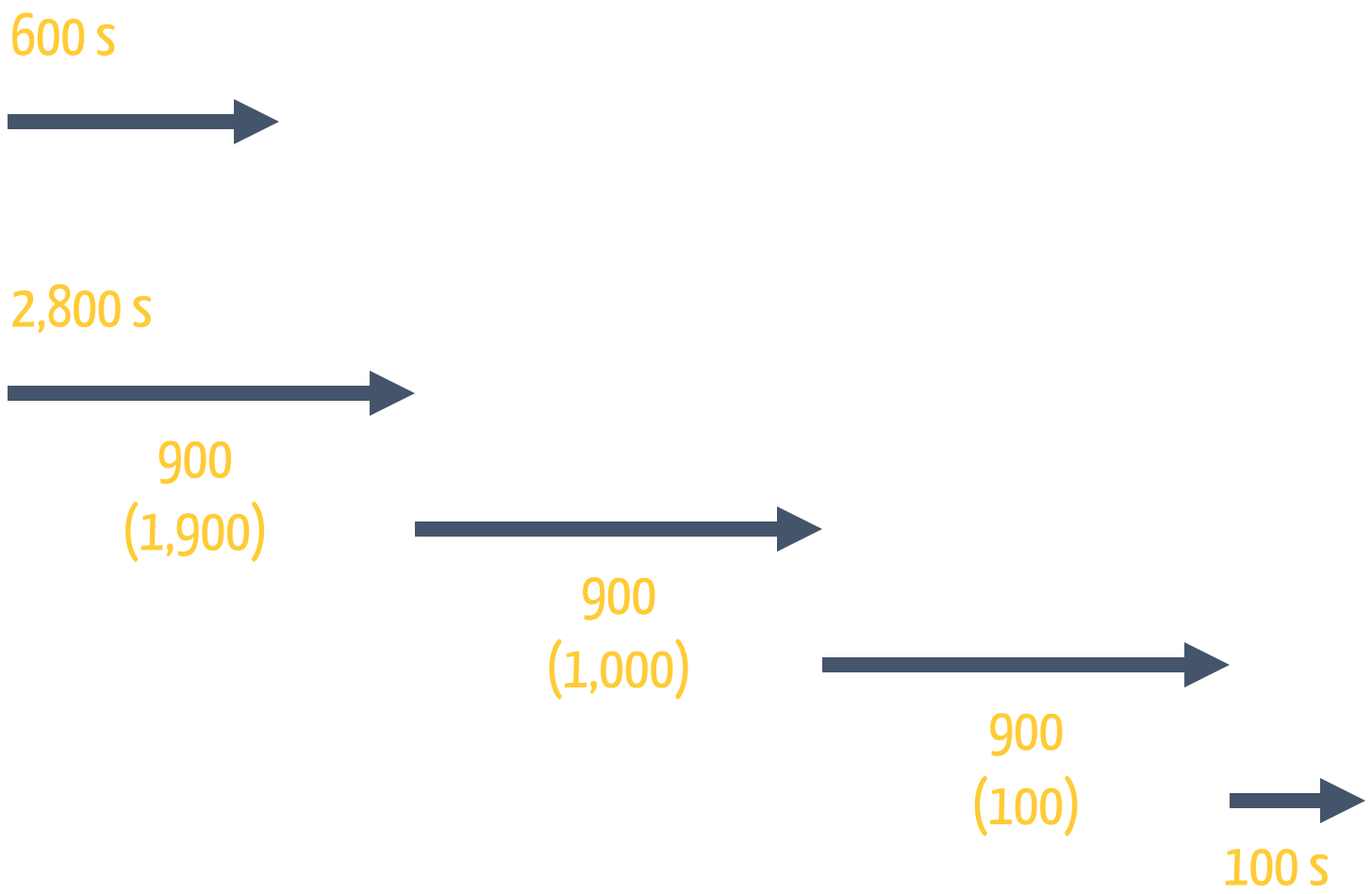
```
    0 references
    public Task Timeout(SubmitOrder state, IMessageHandlerContext context)
    {
        Data.RunningTotal -= state.Total;
        #region
        Console.WriteLine($"Decreased running total of {state.CustomerId.Short()} by {state.Total}");
        #endregion
        return Task.CompletedTask;
    }

    0 references
    protected override void ConfigureHowToFindSaga(SagaPropertyMapper<DiscountPolicyData> mapper)
    {
        mapper.ConfigureMapping<SubmitOrder>(m => m.CustomerId).ToSaga(s => s.CustomerId);
    }
}
```

```
Daniel@DESKTOP-SUIHDT7 c:\p\KnockKnock\demo4
> docker-compose up
Recreating demo4_orders.rabbitmq.nsb_1 ... done
Starting demo4_orders.backend.db.nsb_1 ... done
Recreating demo4_orders.backend.nsb_1  ... done
Recreating demo4_orders.frontend.nsb_1 ... done
Attaching to demo4_orders.backend.db.nsb_1, demo4_orders.rabbitmq.nsb_1, demo4_orders.backend.nsb_1, demo4_orders.frontend.nsb_1
orders.backend.db.nsb_1  | WARNING: no logs are available with the 'none' log driver
orders.rabbitmq.nsb_1    | WARNING: no logs are available with the 'none' log driver
```

```
Daniel@DESKTOP-SUIHDT7 c:\p\KnockKnock\demo4
> docker-compose up
Recreating demo4_orders.rabbitmq.nsb_1 ... done
Starting demo4_orders.backend.db.nsb_1 ... done
Recreating demo4_orders.backend.nsb_1  ... done
Recreating demo4_orders.frontend.nsb_1 ... done
Attaching to demo4_orders.backend.db.nsb_1, demo4_orders.rabbitmq.nsb_1, demo4_orders.backend.nsb_1, demo4_orders.frontend.nsb_1
orders.backend.db.nsb_1   | WARNING: no logs are available with the 'none' log driver
orders.rabbitmq.nsb_1     | WARNING: no logs are available with the 'none' log driver
orders.frontend.nsb_1     | Ready
orders.frontend.nsb_1     |
orders.frontend.nsb_1     | Order #1: Value 300 for customer 19103a6
orders.frontend.nsb_1     | Order #2: Value 300 for customer 19103a6
orders.frontend.nsb_1     | Order #3: Value 300 for customer 19103a6
orders.frontend.nsb_1     | Order #4: Value 300 for customer 19103a6
orders.frontend.nsb_1     | Order #5: Value 300 for customer 19103a6
orders.backend.nsb_1      | --> Order #2: Value 300.0 for customer 19103a6 will be retried due to concurrency conflict.
orders.backend.nsb_1      | Order #1: No discount
orders.backend.nsb_1      | Order #2: No discount
orders.backend.nsb_1      | Order #3: Got a discount of 30.00
orders.backend.nsb_1      | Order #4: Got a discount of 30.00
orders.backend.nsb_1      | Order #5: Got a discount of 30.00
```


Roads? Where we're going, we don't need roads.

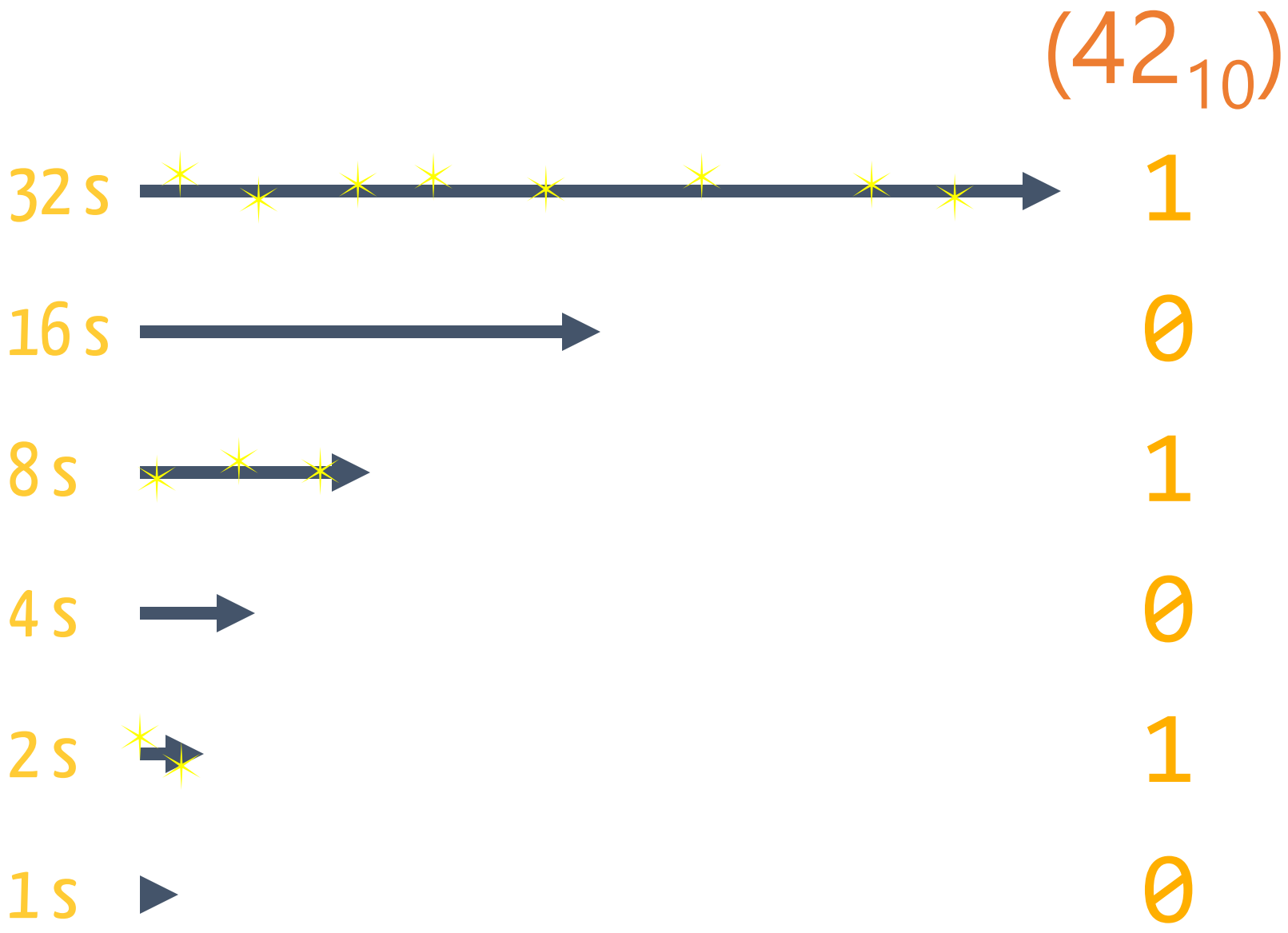
Amazon SQS

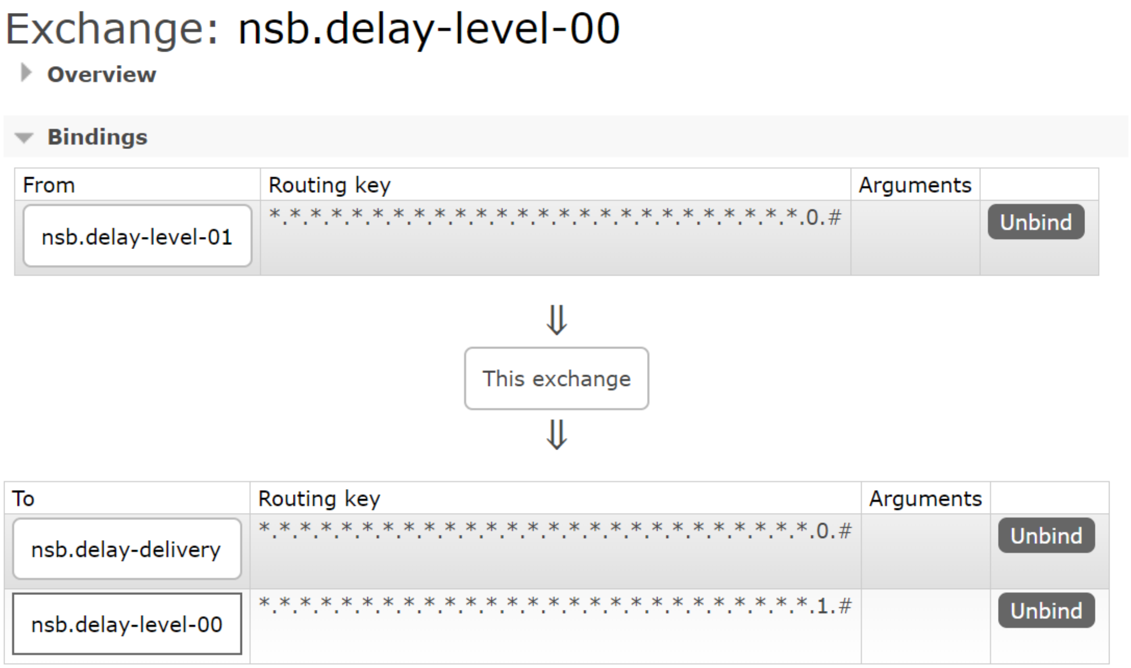

600 s
2,800 s
900
(1,900)
900
(1,000)
900
(100)
100 s

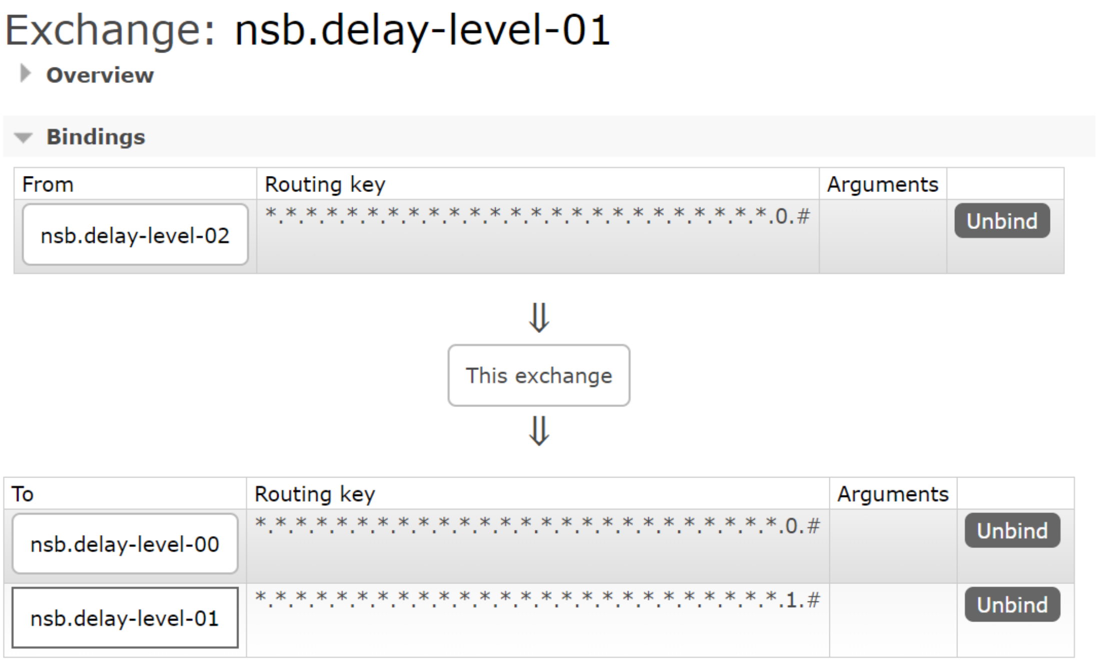
RabbitMQ

42_{10} seconds

101010_{2} seconds


(42_{10})
32 s
1
16 s
0
8 s
1
4 s
0
2 s
1
1 s
0





N.N.N.N.N.N.N.N.N.N.N.N.N.N.N.N.N.N.N.N.N.N.N.N.N.N.N.N.destination

28 queues

2^{28} seconds

268,435,456 seconds

8.5 years

docs.particular.net/transports/sqs/delayed-delivery

docs.particular.net/transports/rabbitmq/delayed-delivery

N.N.N.N.N.N.N.N.N.N.N.N.N.N.N.N.N.N.N.N.N.N.N.N.N.N.N.N.destination

Messaging introduces reliability

Retries resolve consistency issues automatically

Sagas on top of a robust middleware allow to focus on the business logic and stay reactive

For ultra high contention domains different approaches might be necessary

Ask the collaborative domain question first

Business consistency rarely ever needs to be addressed with technical solutions

Favor simplicity over complex design wherever you can

Particular
Software

Thanks


PARSEC
go.particular.net/dnd19-parsec

Long Running Processes

Get free access to a section of the Advanced Distributed Systems Design course

Enroll for free

go.particular.net/dnd19-knock-knock

Slides, Links...

github.com/danielmarbach/KnockKnock

Particular
Software

Q&A

Particular Software

Software Engineer
Enthusiastic Software Engineer
Microsoft MVP

@danielmarbach
particular.net/blog
planetgeek.ch

Particular
Software

Thanks