Modeling Workflows with NServiceBus Sagas

Roland Guijt
MVP | CONSULTANT | TRAINER

@rolandguijt   rolandguijt.com

Overview

Introduction to sagas

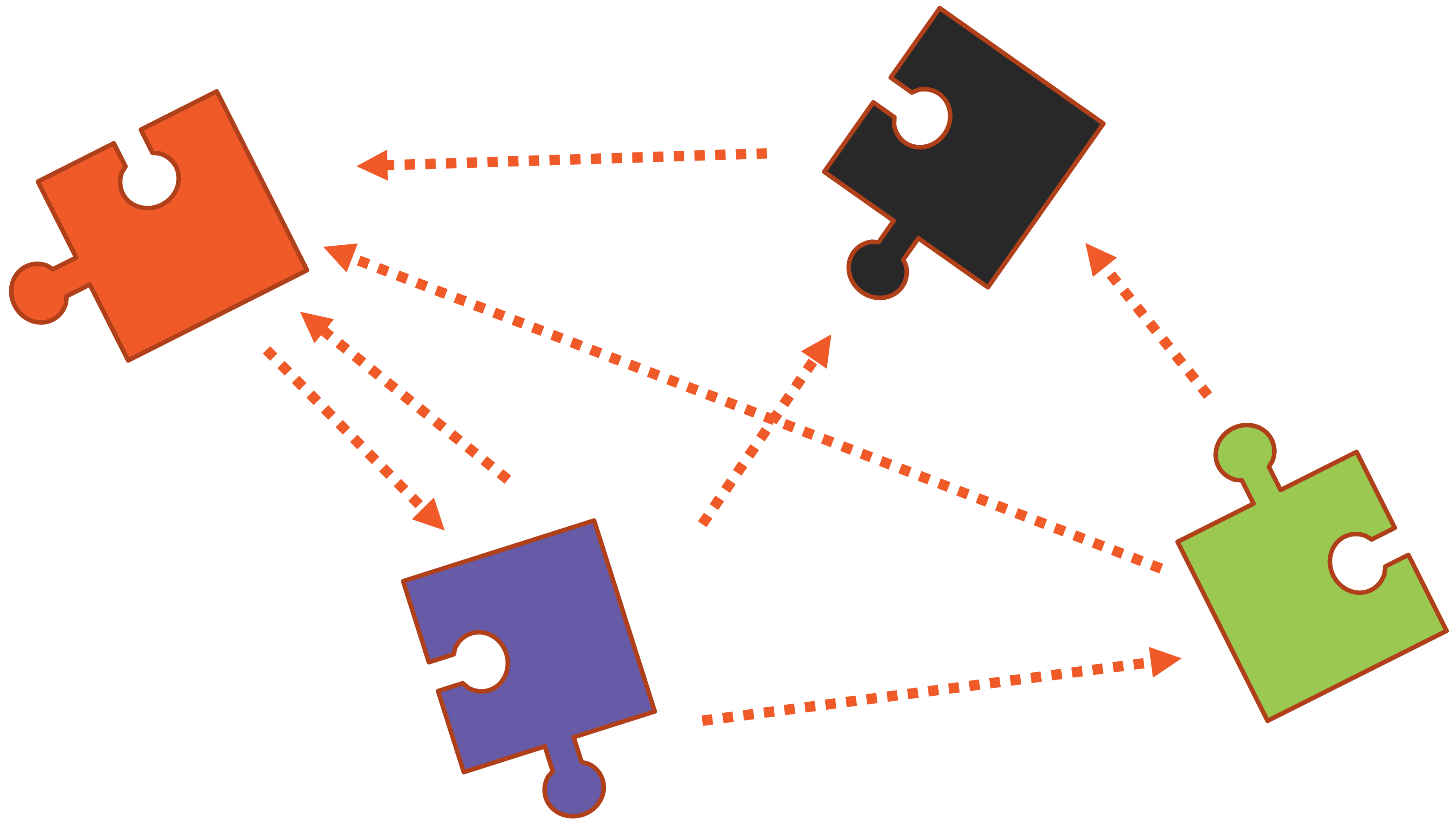
Defining a saga

Designing sagas

Timeouts

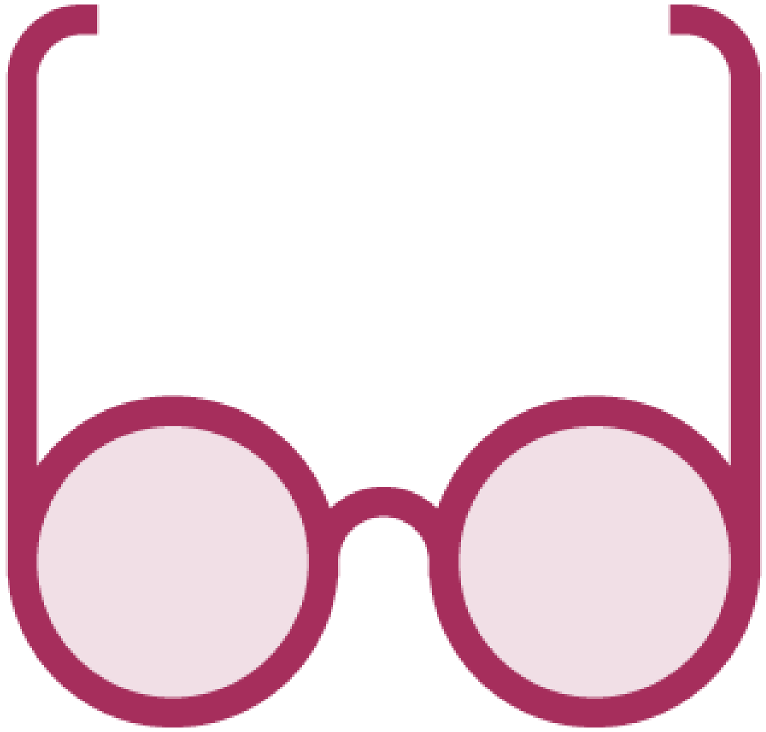
Persistence

Why Sagas?



Why Sagas?

What Are Sagas?



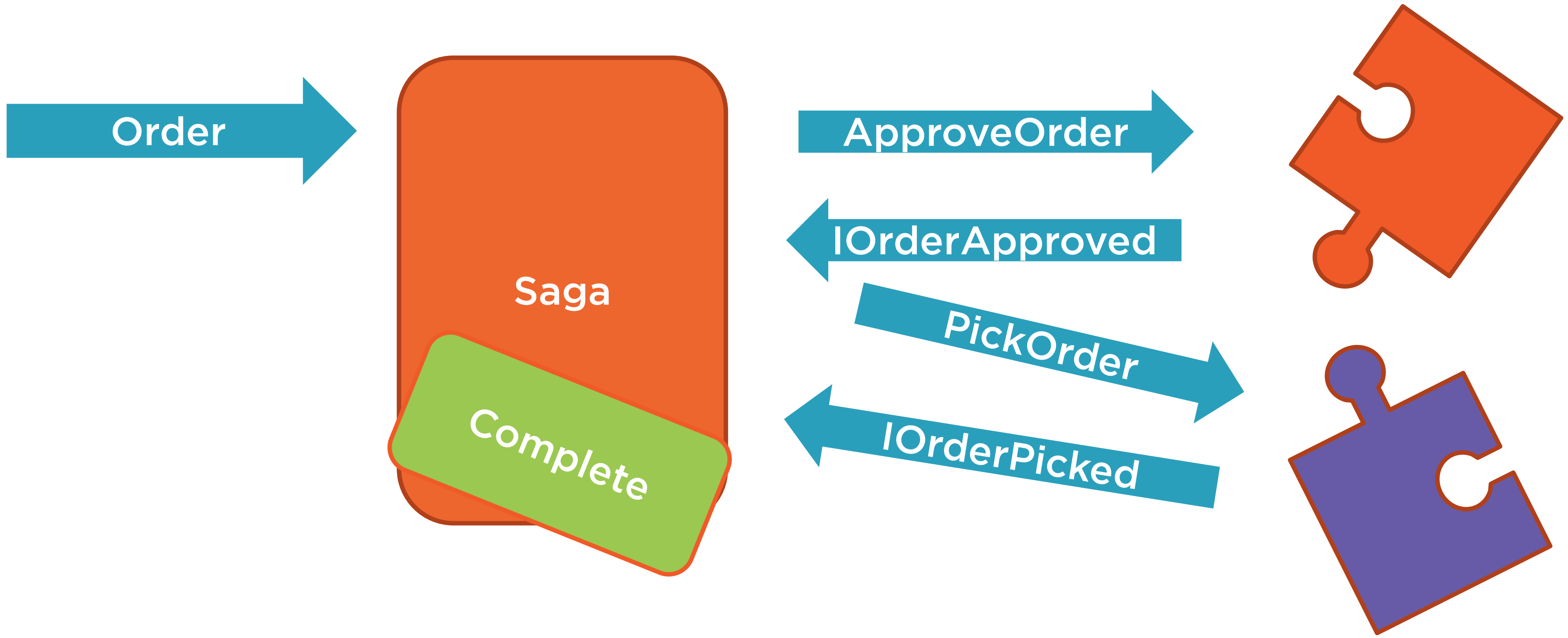
Long running business processes

Workflows with state

Coordinate message flow

Persisted while running

What Are Sagas?



When to Use a Saga?

Processes with more than one message roundtrip

Time related process requirements

Defining a Saga

```
public class OrderSaga : Saga<OrderSagaData>,
                        IAmStartedByMessages<StartOrder>,
                        IHandleMessages<CompleteOrder>
{
//
}
```

Ending a Saga

```
public async Task Handle(CompleteOrder message,
      IMessageHandlerContext context)
{
    // code to handle order completion
    MarkAsComplete();
}
```

Configuring How to Find a Saga

```
protected override void ConfigureHowToFindSaga
  (SagaPropertyMapper<OrderSagaData> mapper)
{
    mapper.ConfigureMapping<CompleteOrder>(s =>s.OrderId)
        .ToSaga(m => m.OrderId);
}
```

Reply

```
public void Handle(RequestDataMessage message,
      IMessageHandlerContext context)
{
var response = new DataResponseMessage
    {
        OrderId = message.OrderId,
        String = message.String
    };
    await context.Reply(response).ConfigureAwait(false);
}
```

ReplyToOriginator

```
public void Handle(StartMessage message,
        IMessageHandlerContext context)
{
    Data.OrderId = message.OrderId;
    await ReplyToOriginator(new AlmostDoneMessage
    {
        OrderId = Data.OrderId
    }).ConfigureAwait(false);
}
```

Designing Sagas

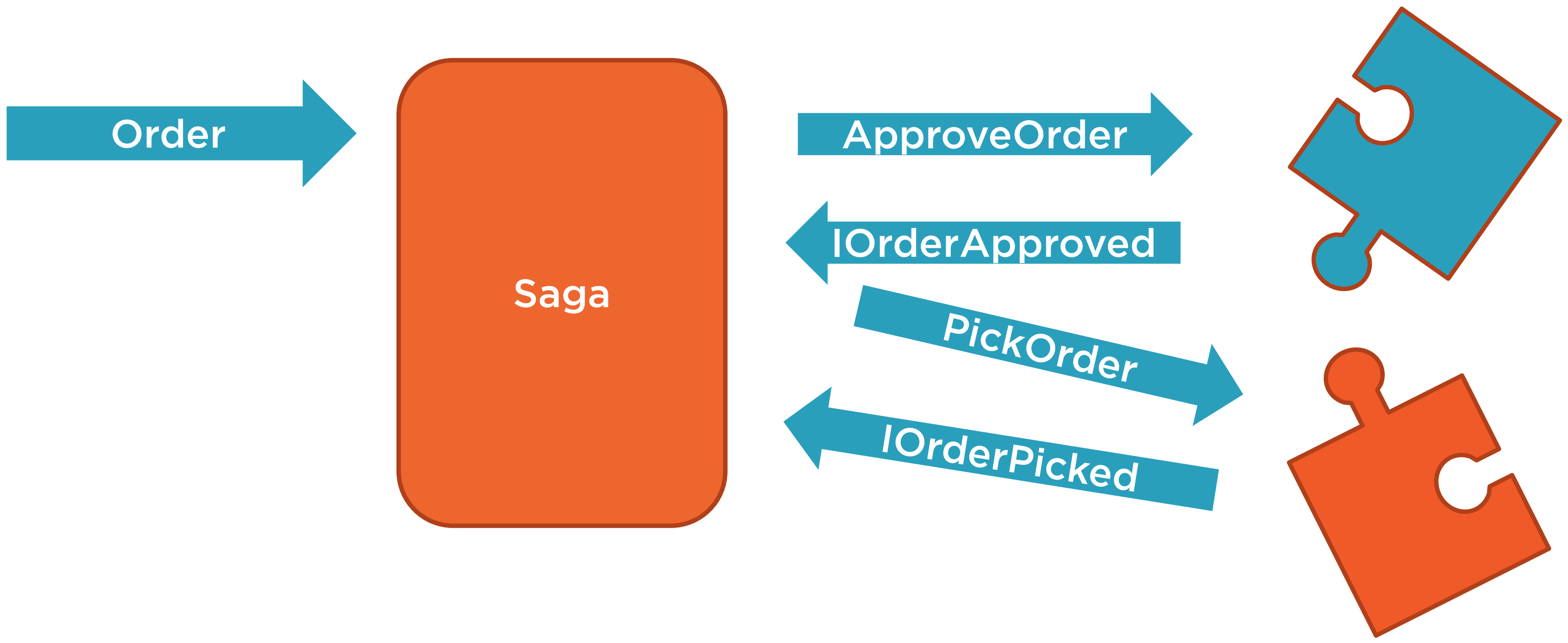
Coordinate only

Saga starting messages

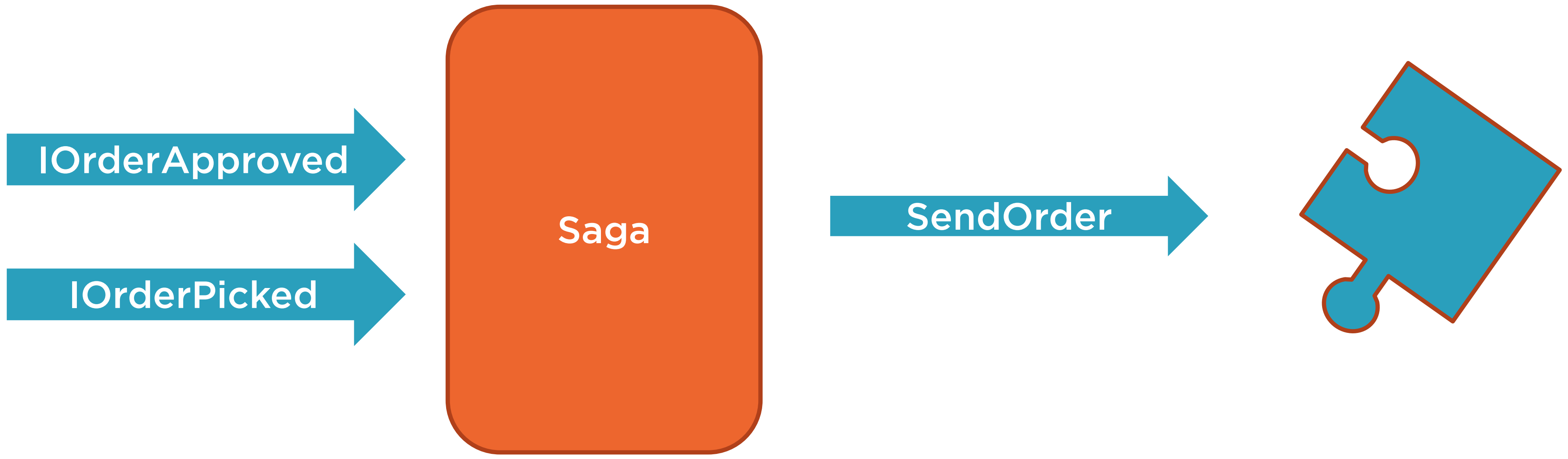
Message order

Patterns

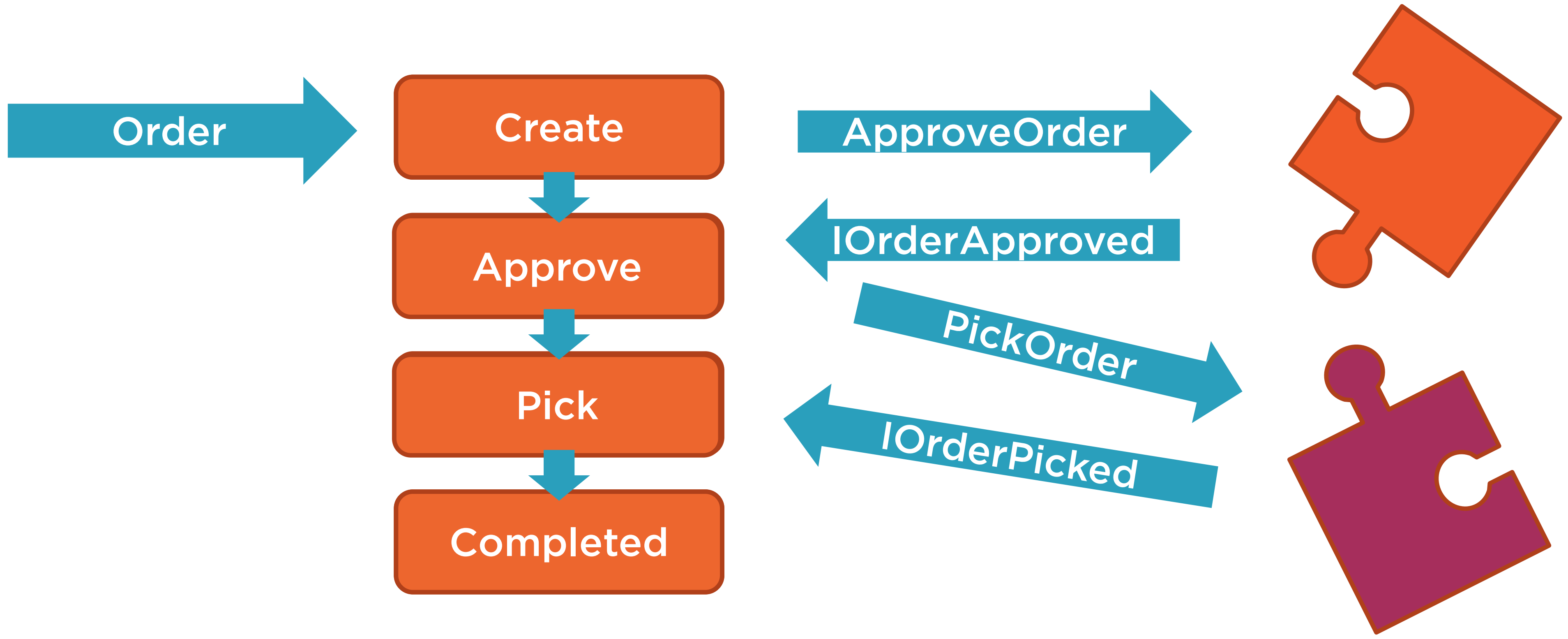
Command Pattern



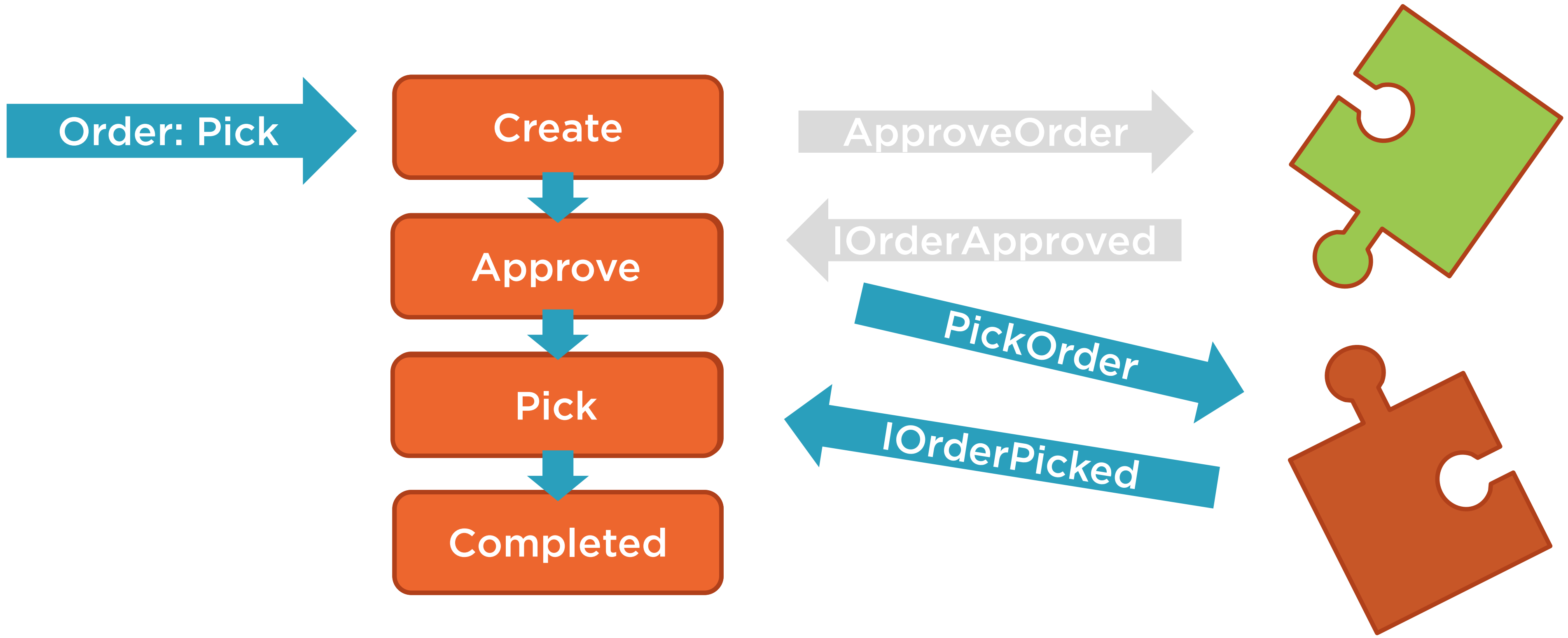
Observer Pattern



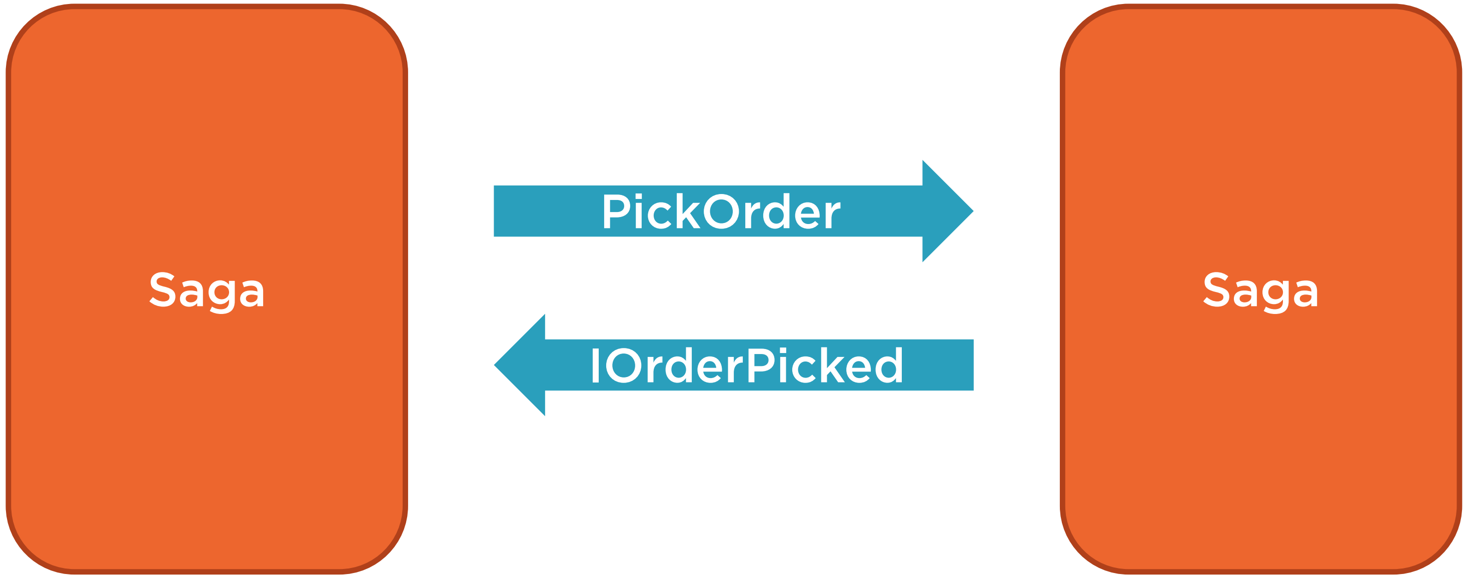
Using Steps


Order
Create
Approve
Pick
Completed
ApproveOrder
IOrderApproved
PickOrder
IOrderPicked

Routing Slip Pattern



Multiple Sagas



Saga Persistence

Each storage mechanism is inherently different

Know the following before choosing

RavenDB

Fetches document using an index specified by unique property

NHibernate

Child objects converted to string in one column

Collections result in extra tables

Danger of lock increases

Mark properties in data object as virtual

Azure

Uses table storage

Collections and child objects not supported

Simple types only

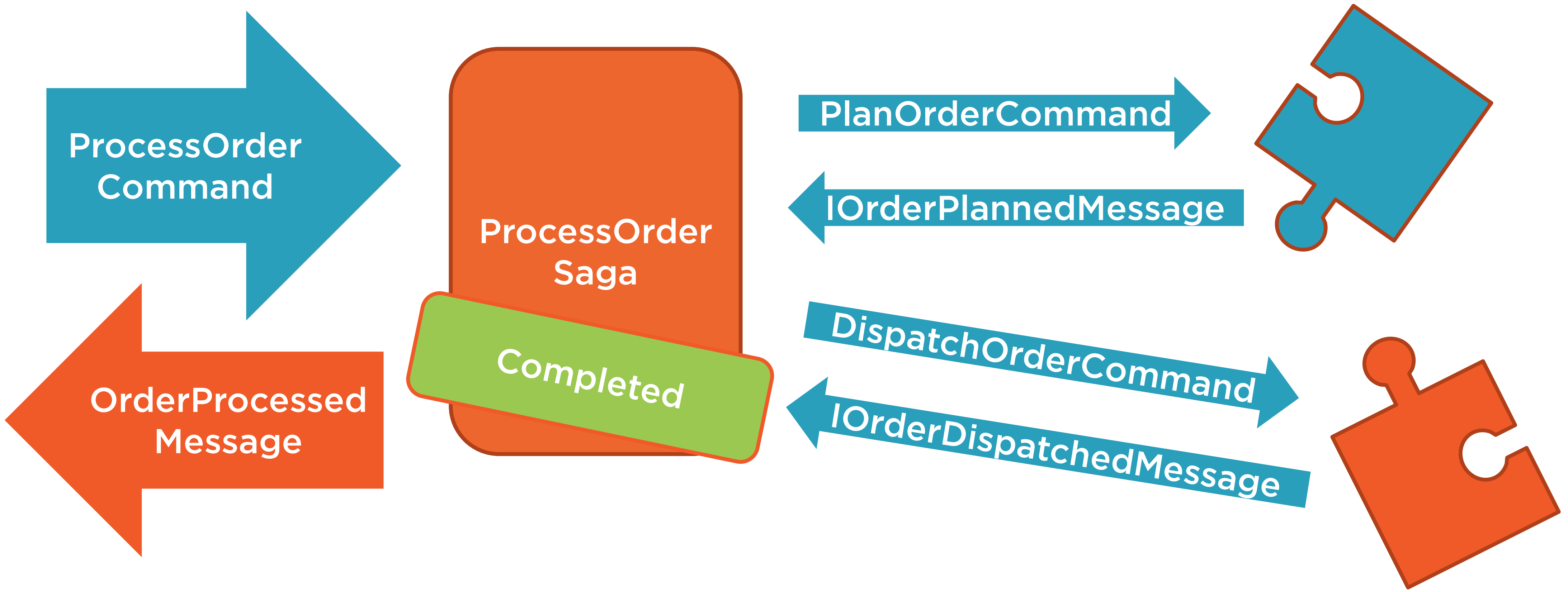
Demo

Orders are handled inefficiently

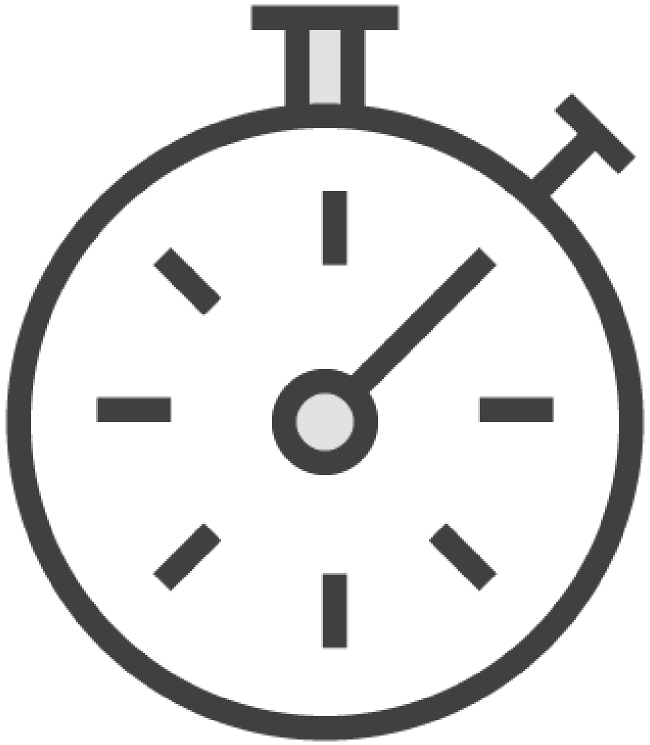
Extra service: planner

Coordination needed

The New Architecture



Timeouts



Saga sends message to timeout manager

When the specified time is up it sends the message back to the saga

When saga has completed, message is ignored

Setting a Timeout

```
await RequestTimeout<ApprovalTimeout>
   (DateTime.Now.AddDays(2));

await RequestTimeout(DateTime.Now.AddDays(2),
   new ApprovalTimeout { SomeState = state });

await RequestTimeout<ApprovalTimeout>
   (TimeSpan.FromDays(2), t => t.SomeState = state);
```

Handling a Timeout

```
public class OrderSaga : Saga<OrderSagaData>,
    IHandleTimeouts<ApprovalTimeout>
{
    public void Timeout(ApprovalTimeout state,
        IMessageHandlerContext context)
    {
    }
}
```

Summary

Sagas are long-running business processes

Coordinate, decide - not implement

Time

Persistence