Sagas

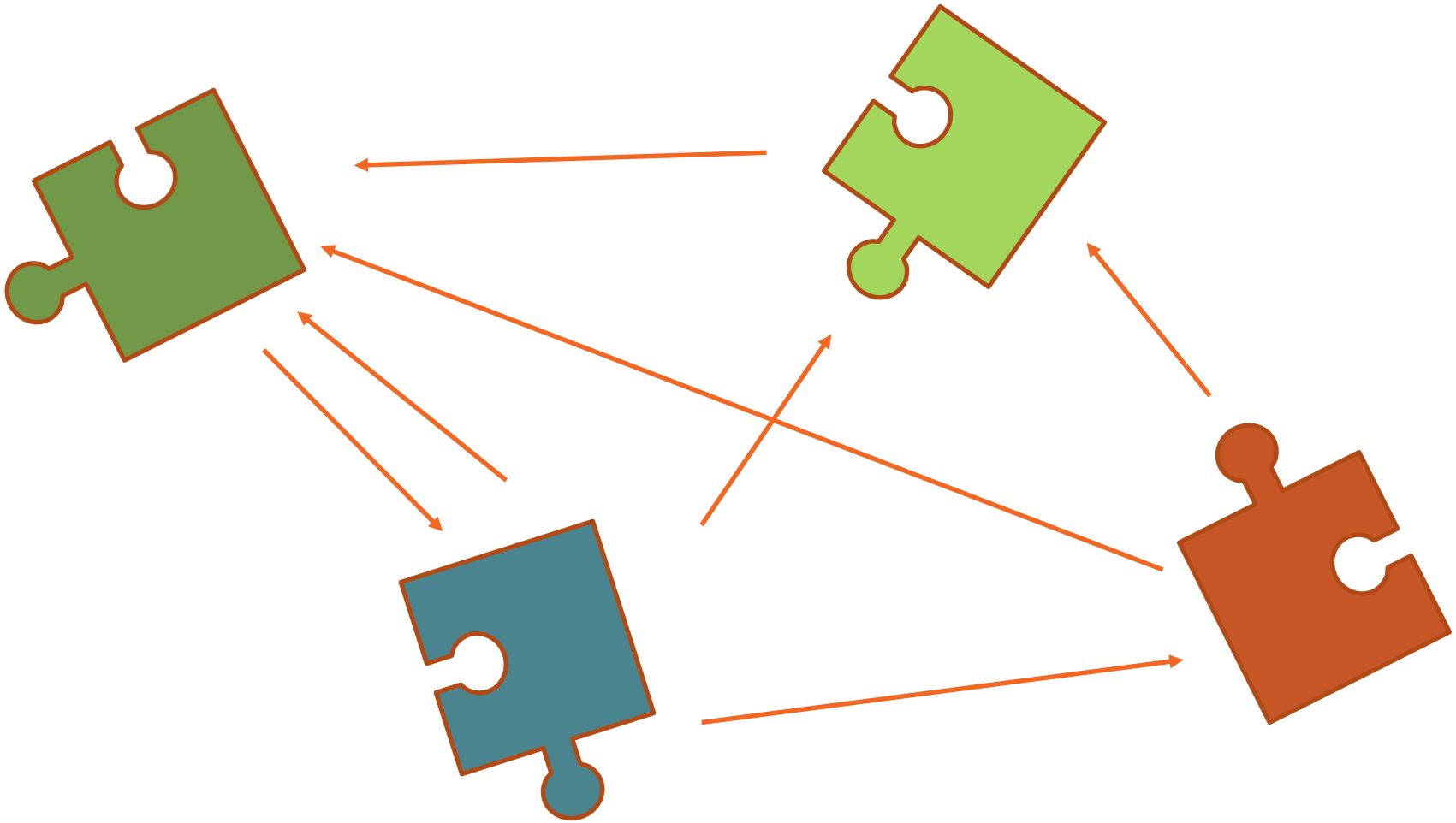
Roland Guijt

@rolandguijt | www.rmgsolutions.nl

Overview

- Introduction to sagas
- Defining a saga
- Designing sagas
- Timeouts
- Persistence
- Unit testing

Why Sagas?

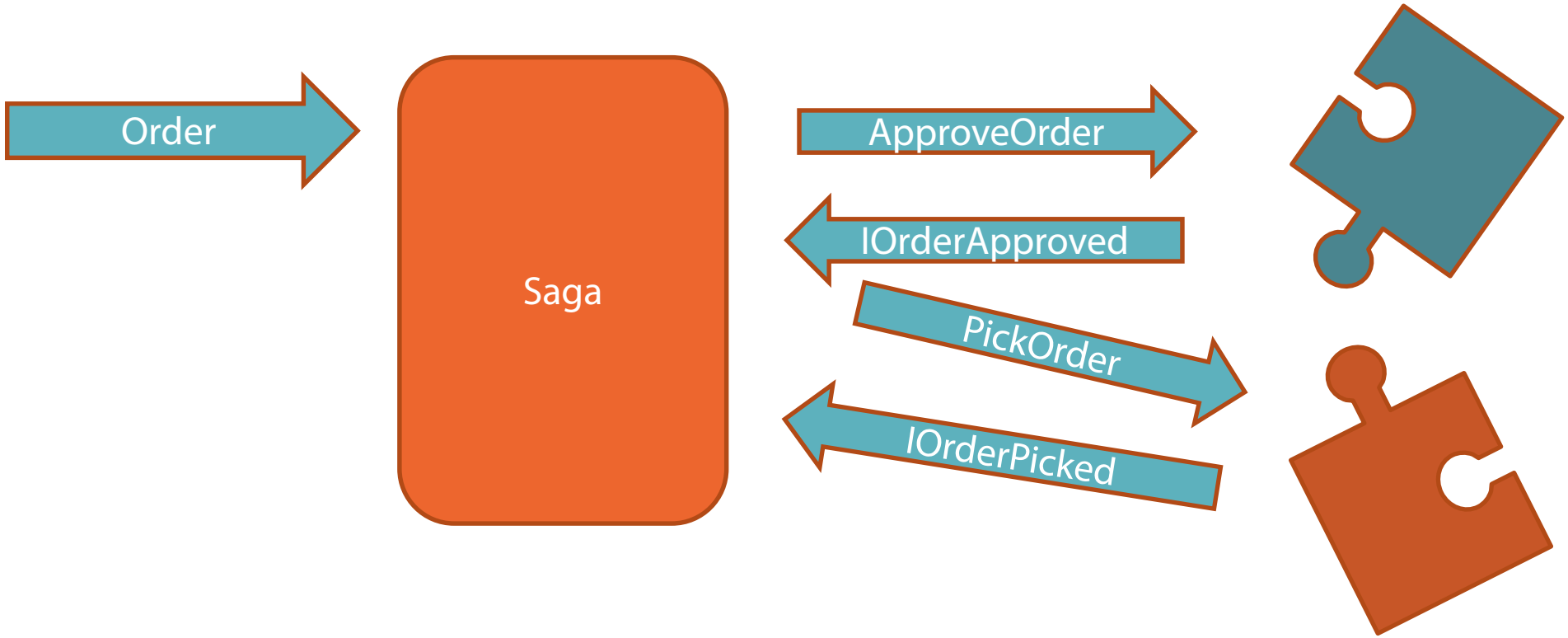


Why Sagas?

What Are Sagas?

- Long-running business processes
- Workflows with state
- Coordinate message flow
- Persisted while running

What Are Sagas?



When to Use Sagas

- Processes with more than one message round-trip
- Time-related requirements

Defining a Saga

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

Ending a Saga

```
public void Handle(CompleteOrder message)
{
    // 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);
}
```

Bus.Reply

```
public void Handle(RequestDataMessage message)
{
var response = new DataResponseMessage
    {
        OrderId = message.OrderId,
        String = message.String
    };
    Bus.Reply(response);
}
```

Bus.Reply

```
public void Handle(RequestDataMessage message)
{
var response = new DataResponseMessage
    {
        OrderId = message.OrderId,
        String = message.String
    };
    Bus.Reply(response);
}
```

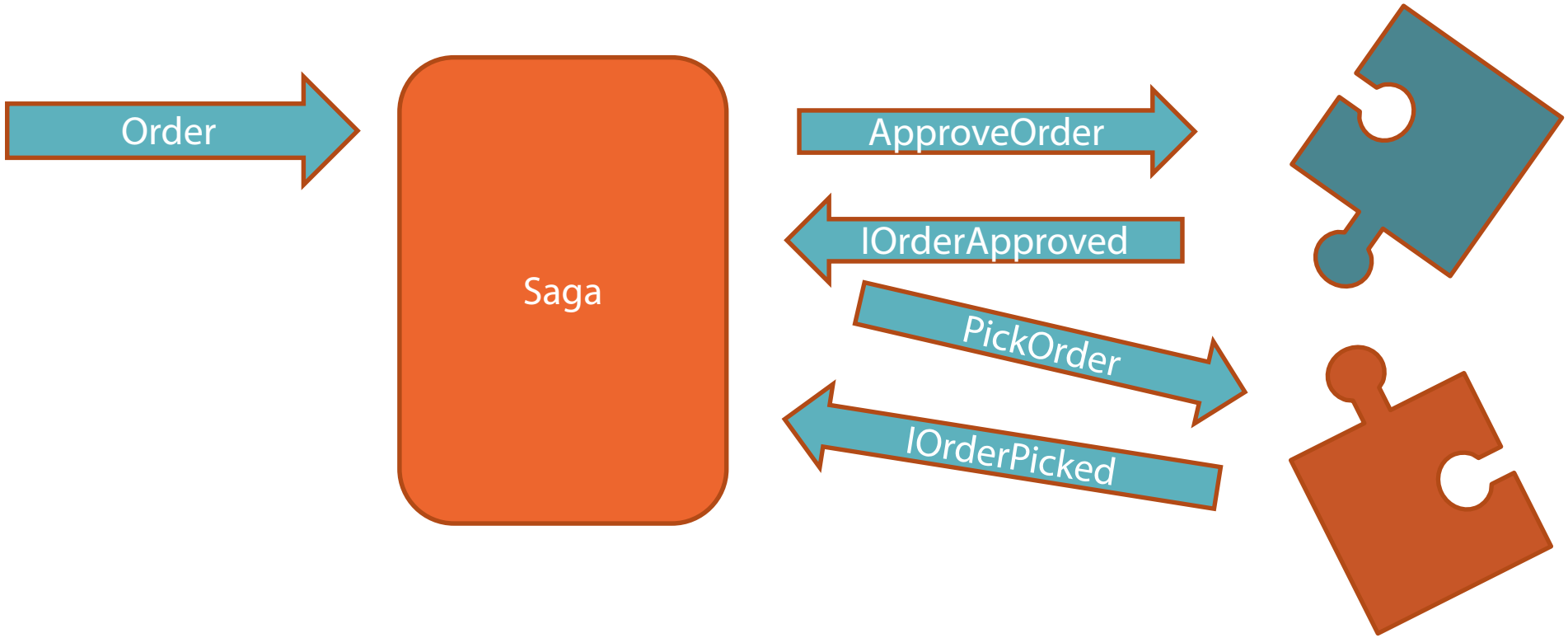
Bus.ReplyToOriginator

```
public void Handle(StartMessage message)
{
    Data.OrderId = message.OrderId;
    ReplyToOriginator(new AlmostDoneMessage
    {
        OrderId = Data.OrderId
    });
}
```

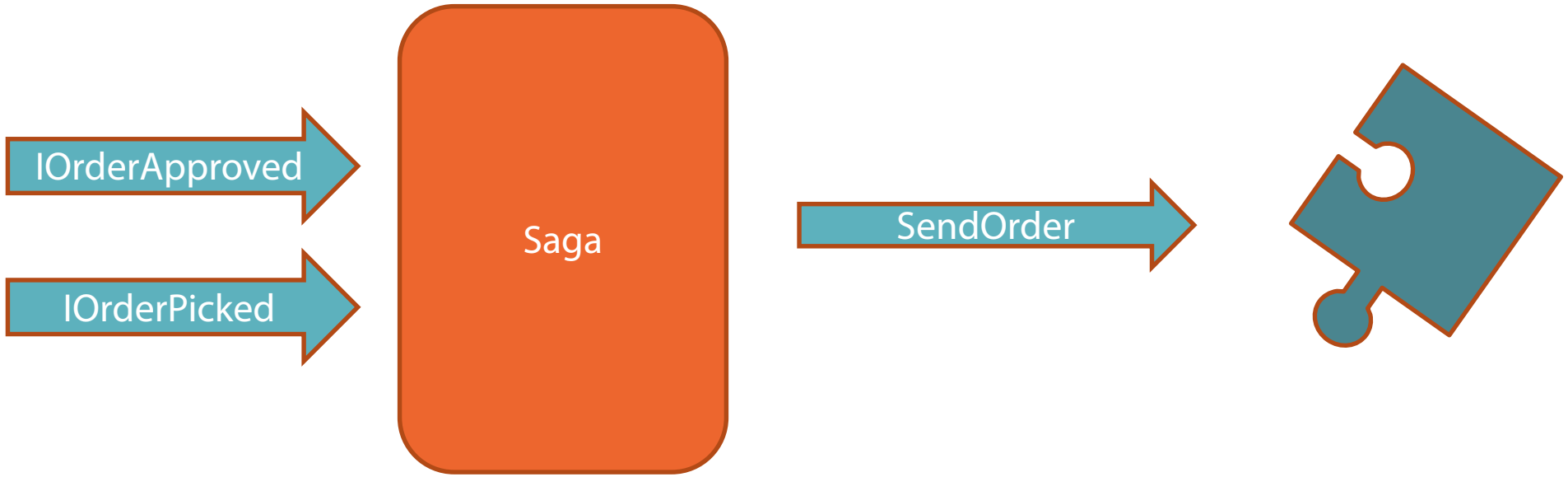
Designing Sagas

- Coordinate only
- Saga starting messages
- Message order
- Patterns

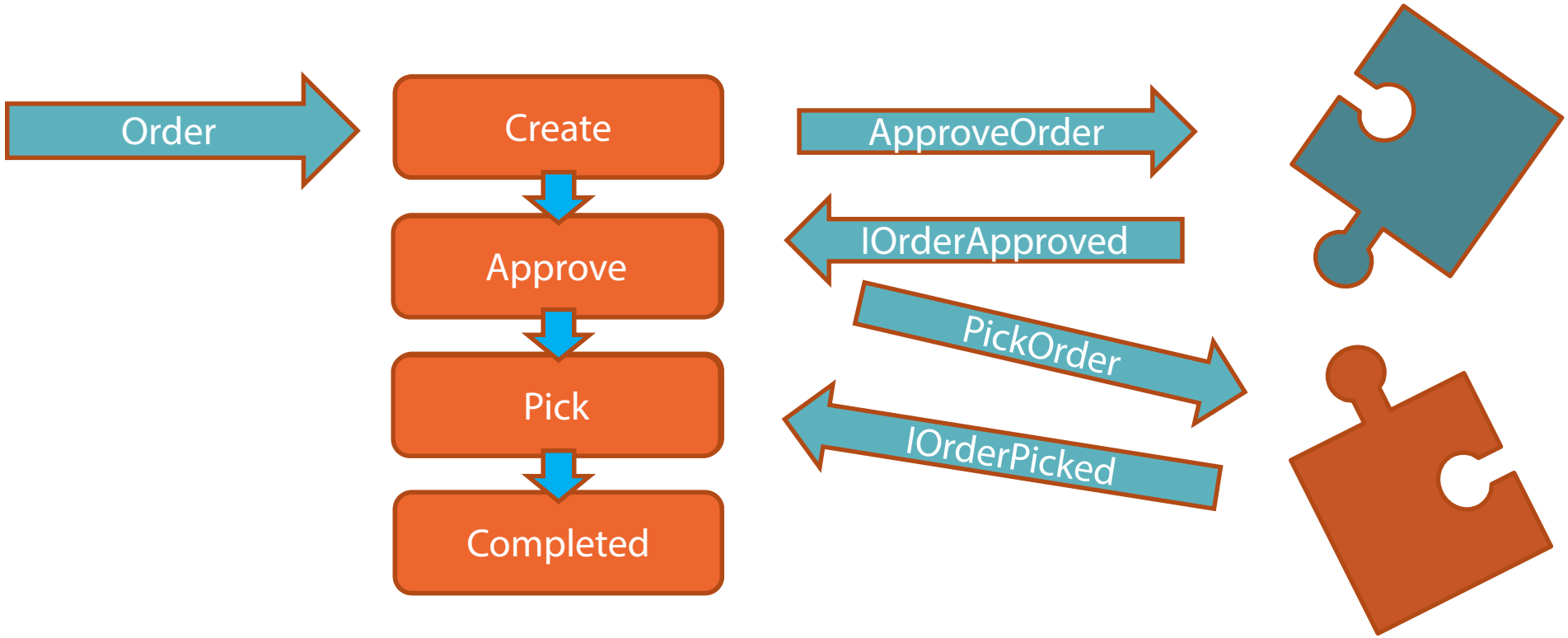
Command Pattern



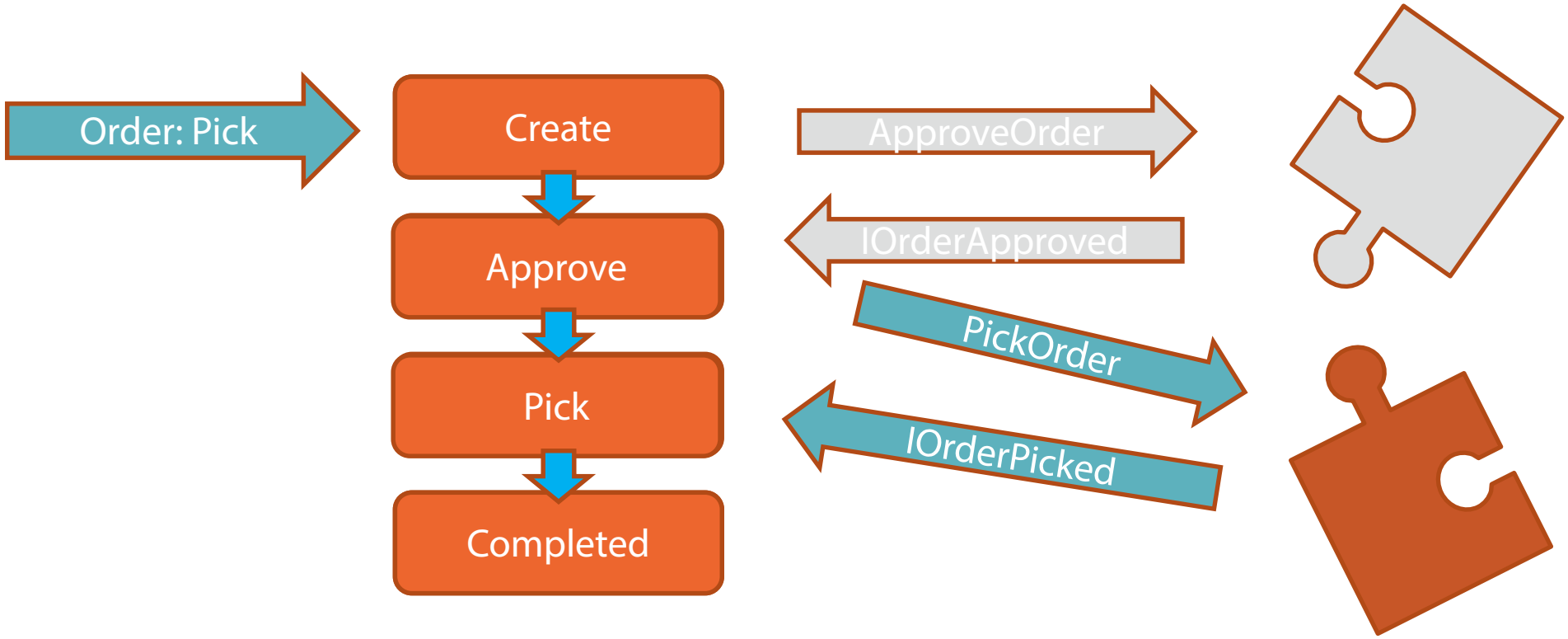
Observer Pattern



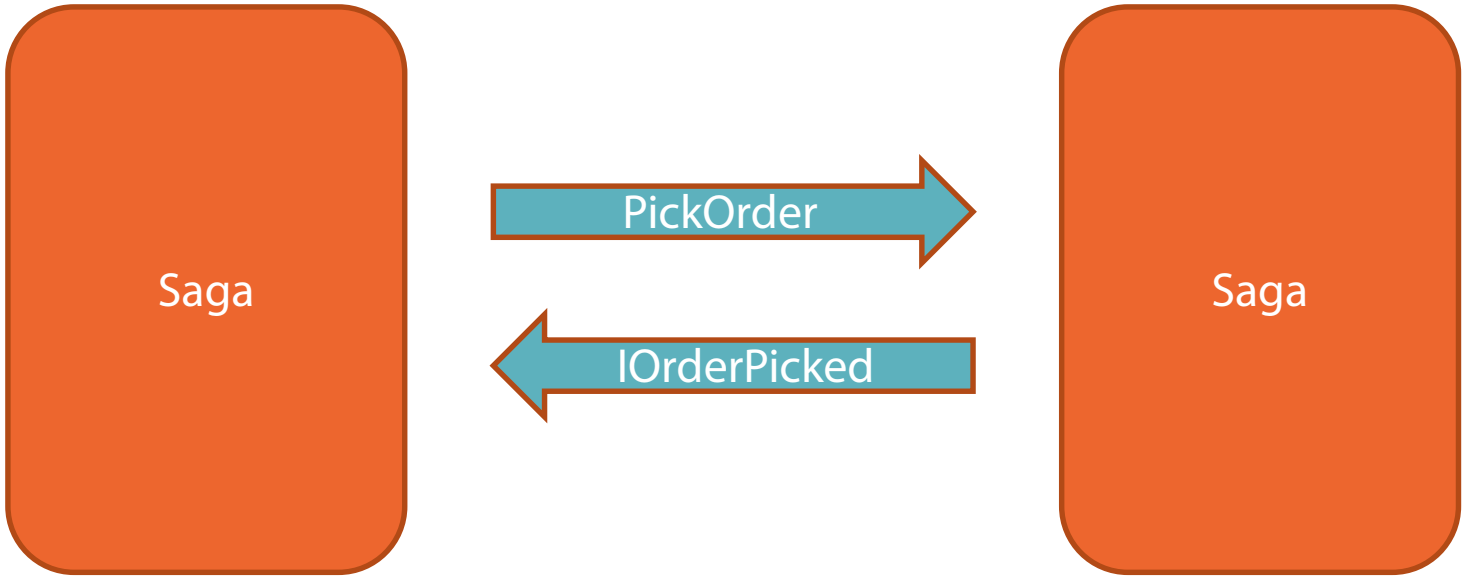
Using Steps



Routing Slip Pattern

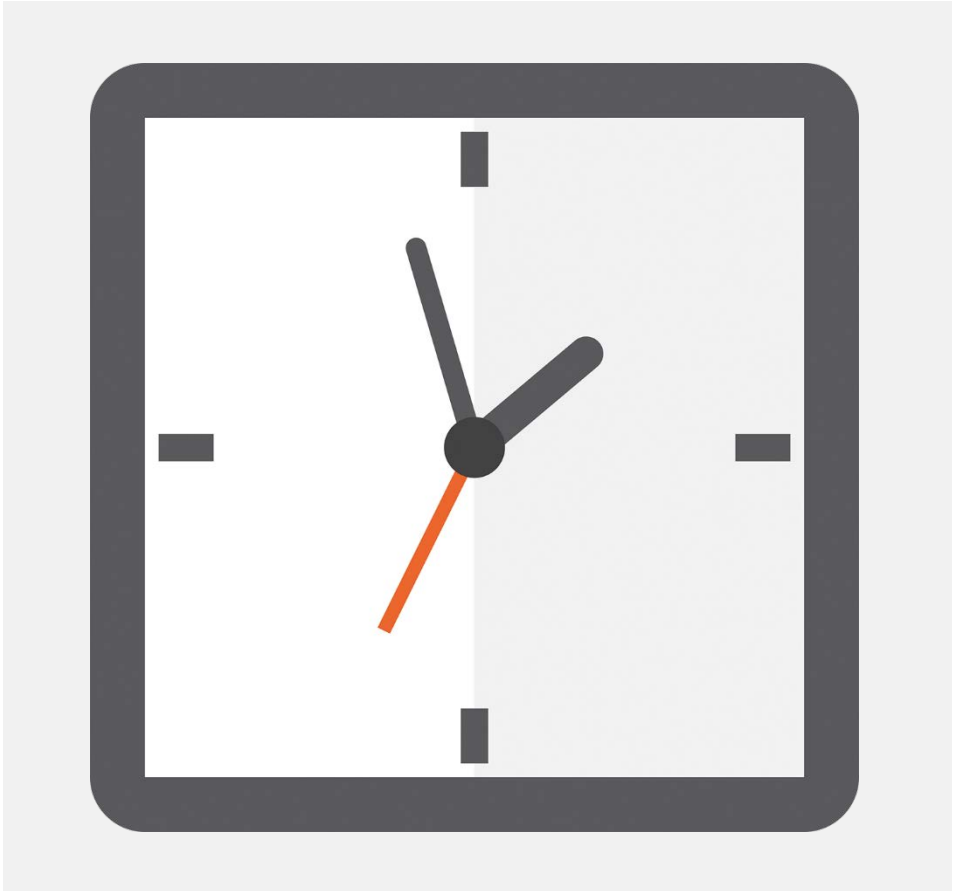


Multiple Sagas



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

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

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

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

Handling a Timeout

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

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

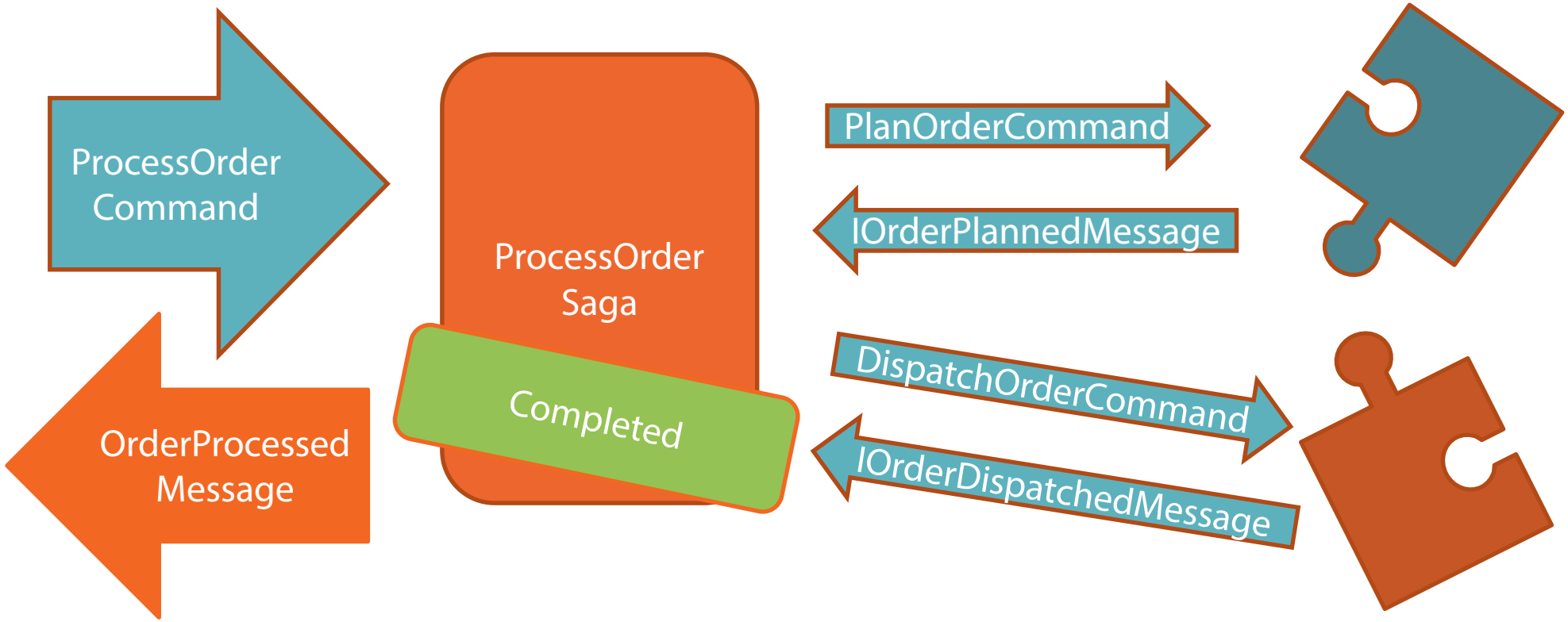
Demo: Sagas

Orders are handled inefficiently

Extra service: planner

Coordination needed

The New Architecture



Summary

Sagas are long-running business processes

Coordinate and decide - not implement

Time

Unit testing