C# Design Patterns: State

THE STATE DESIGN PATTERN

Marc Gilbert
FRIVOLOUSTWIST, LLC

@frivoloustwist   www.frivoloustwist.com

What is state?

State

State is the condition of something variable.

States of Matter

Solid

Liquid

Gas

Questions of State

Is an order in an order processing application:

- New?
- Processing?
- Canceled?
- Complete?

Can a user edit a canceled order?

Can a completed order be canceled?

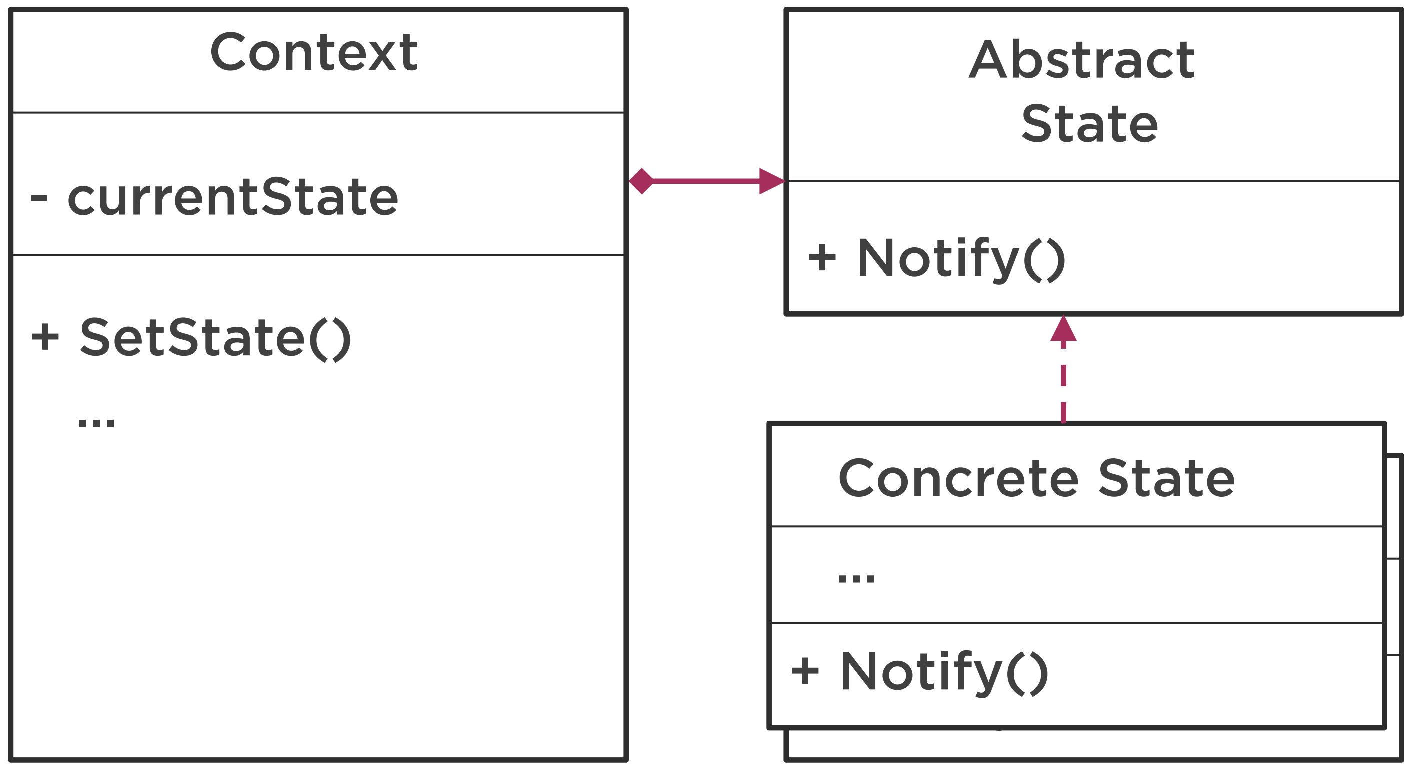
The State Design Pattern

Design Challenges

How can an object change its behavior when its internal state changes?

How can state specific behaviors be defined so that states can be added without altering the behavior of existing states?

The State Pattern

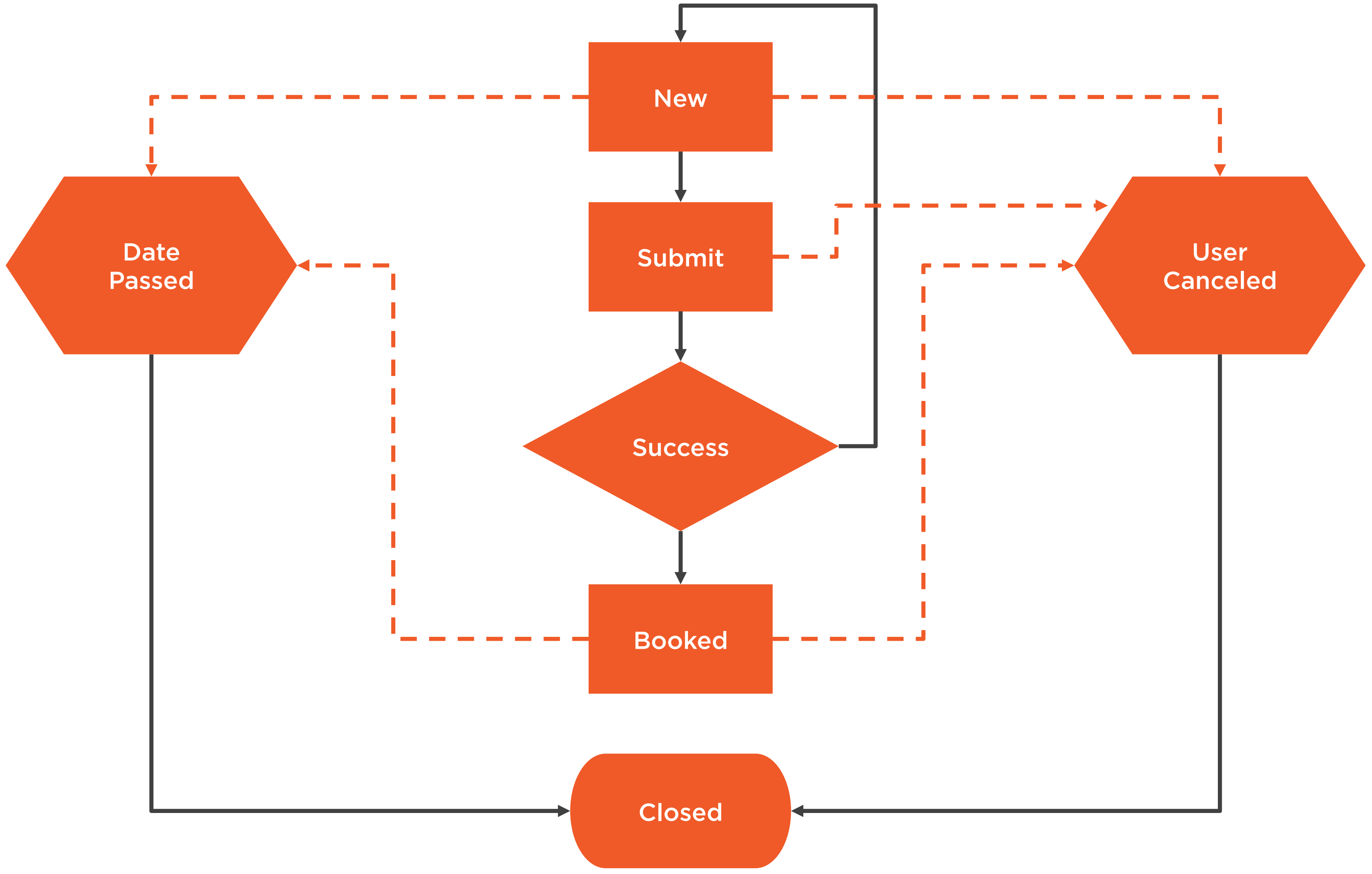


Coming Up

The Demo Project

StateDesignPattern.zip

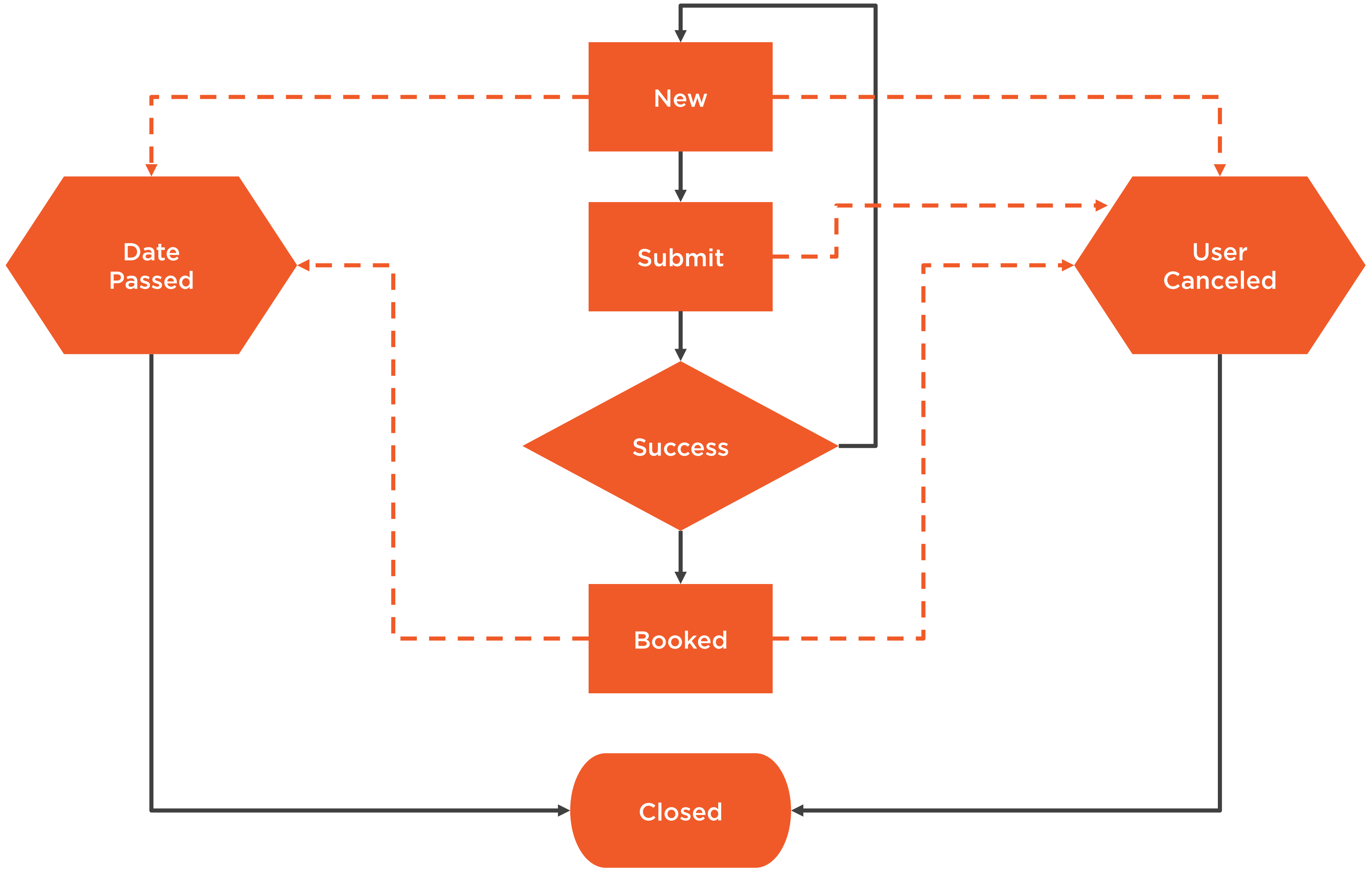
Event Booking Process



Coming Up

A Naïve Approach to Managing State

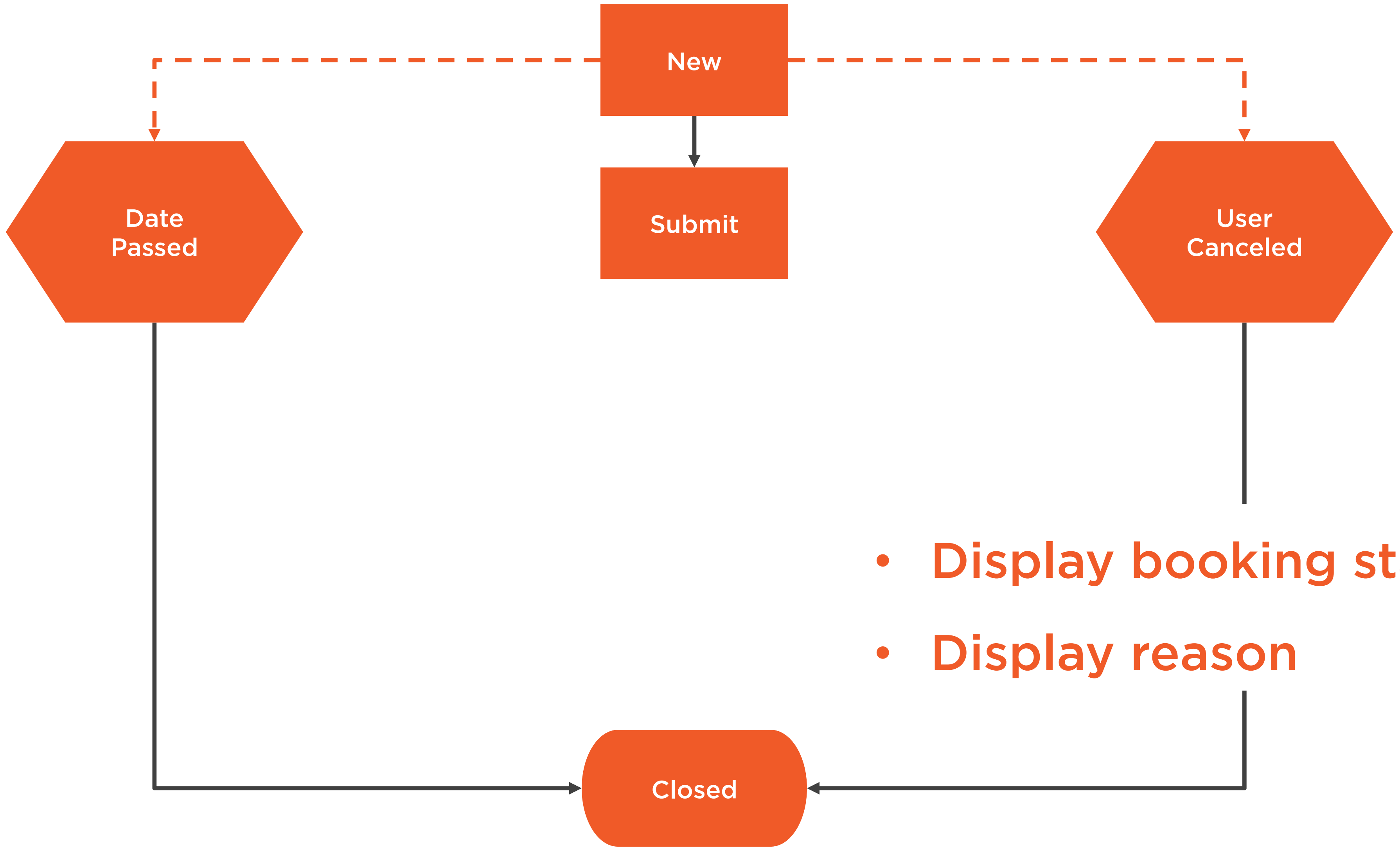
Event Booking Process



Event Booking Process

- Assign a booking ID
- Display booking status
- Provide for data entry

Event Booking Process

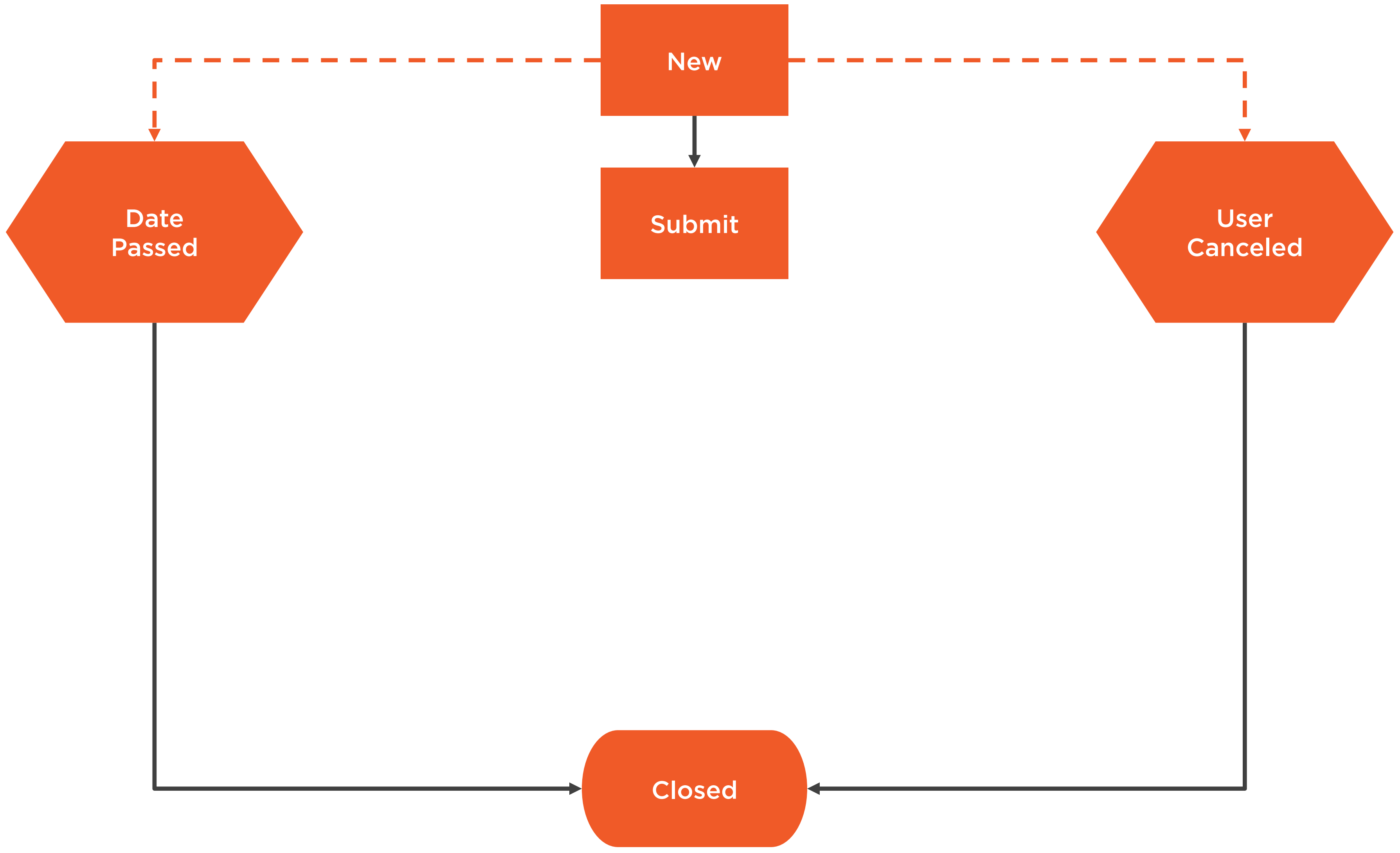


- Display booking state
- Display reason

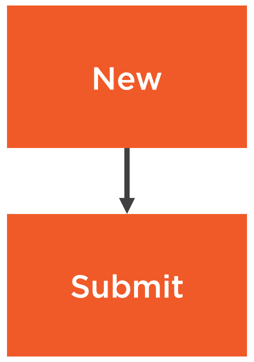
Coming Up

Completing the Naïve Implementation

Event Booking Process

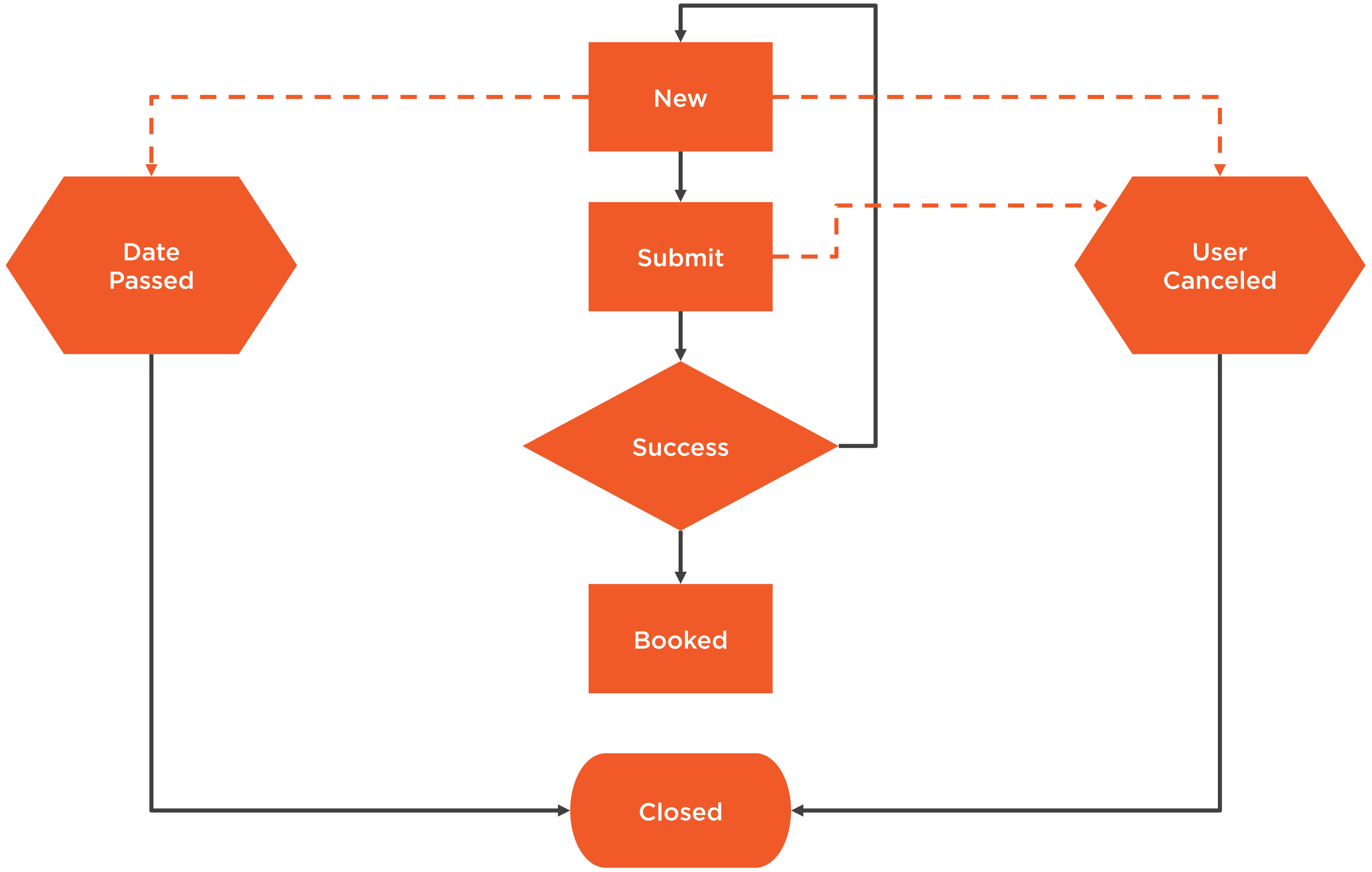


Event Booking Process

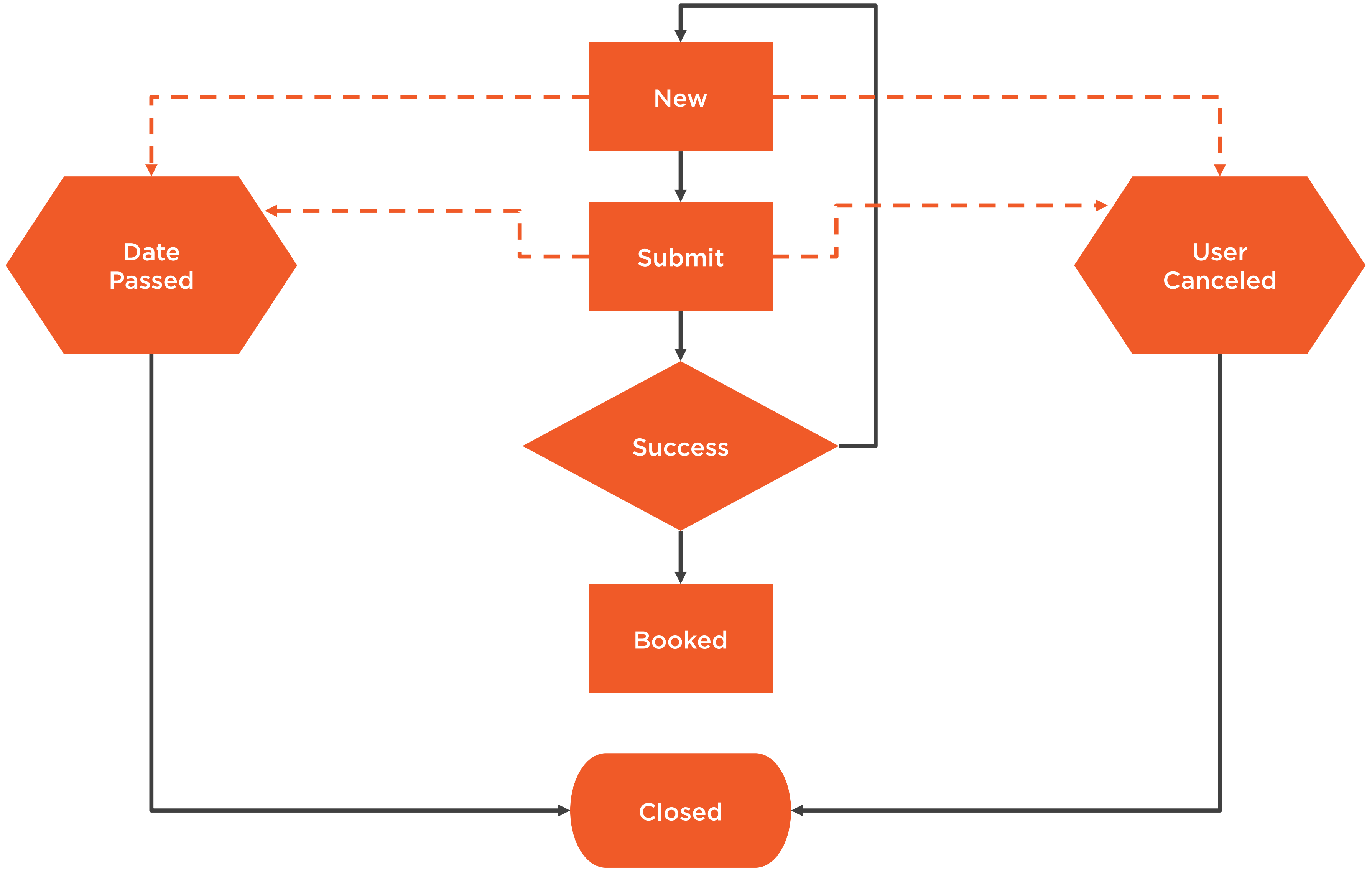


- Update booking data
- Display status
- Submit for processing
- Handle response

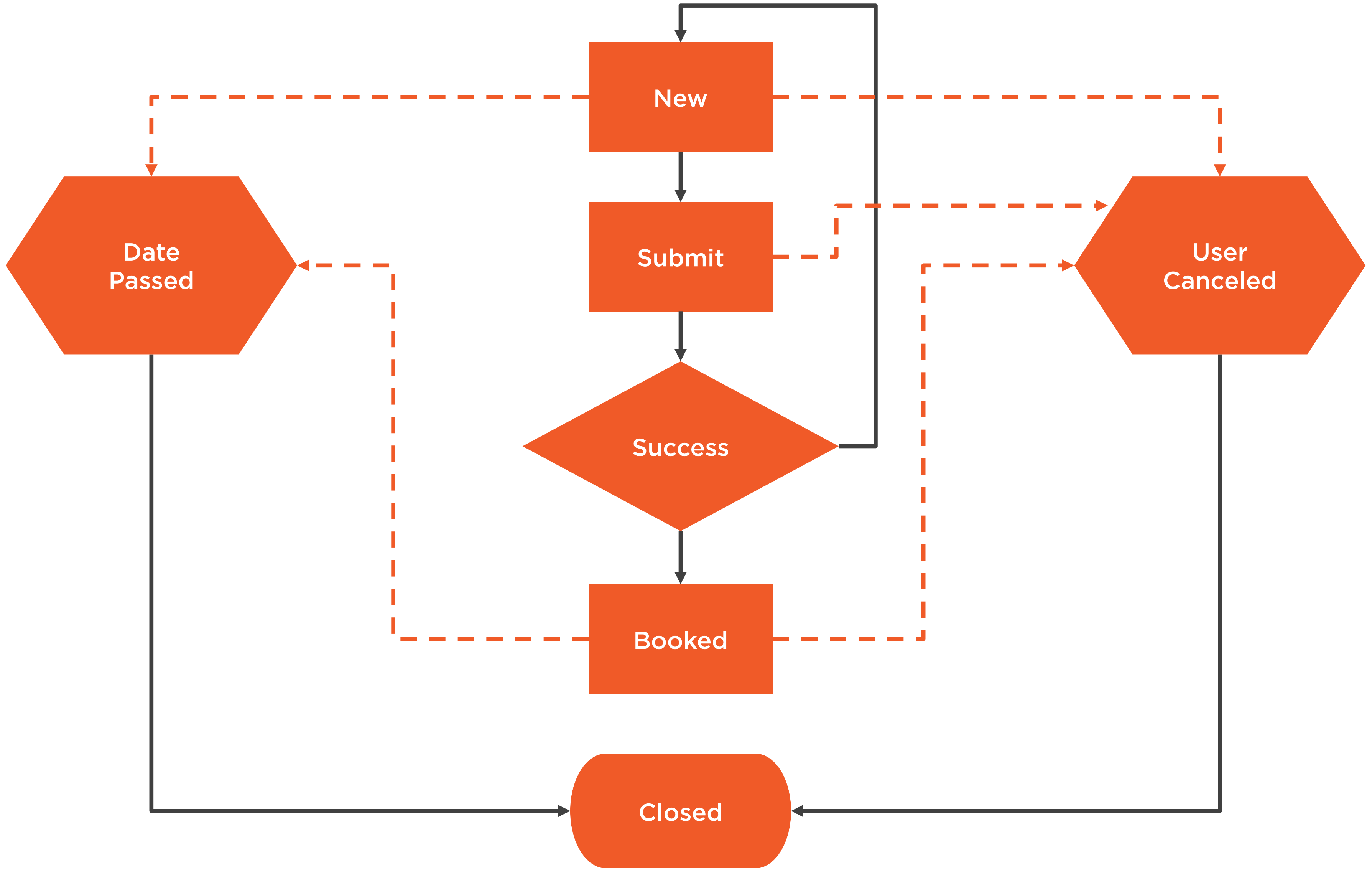
Event Booking Process



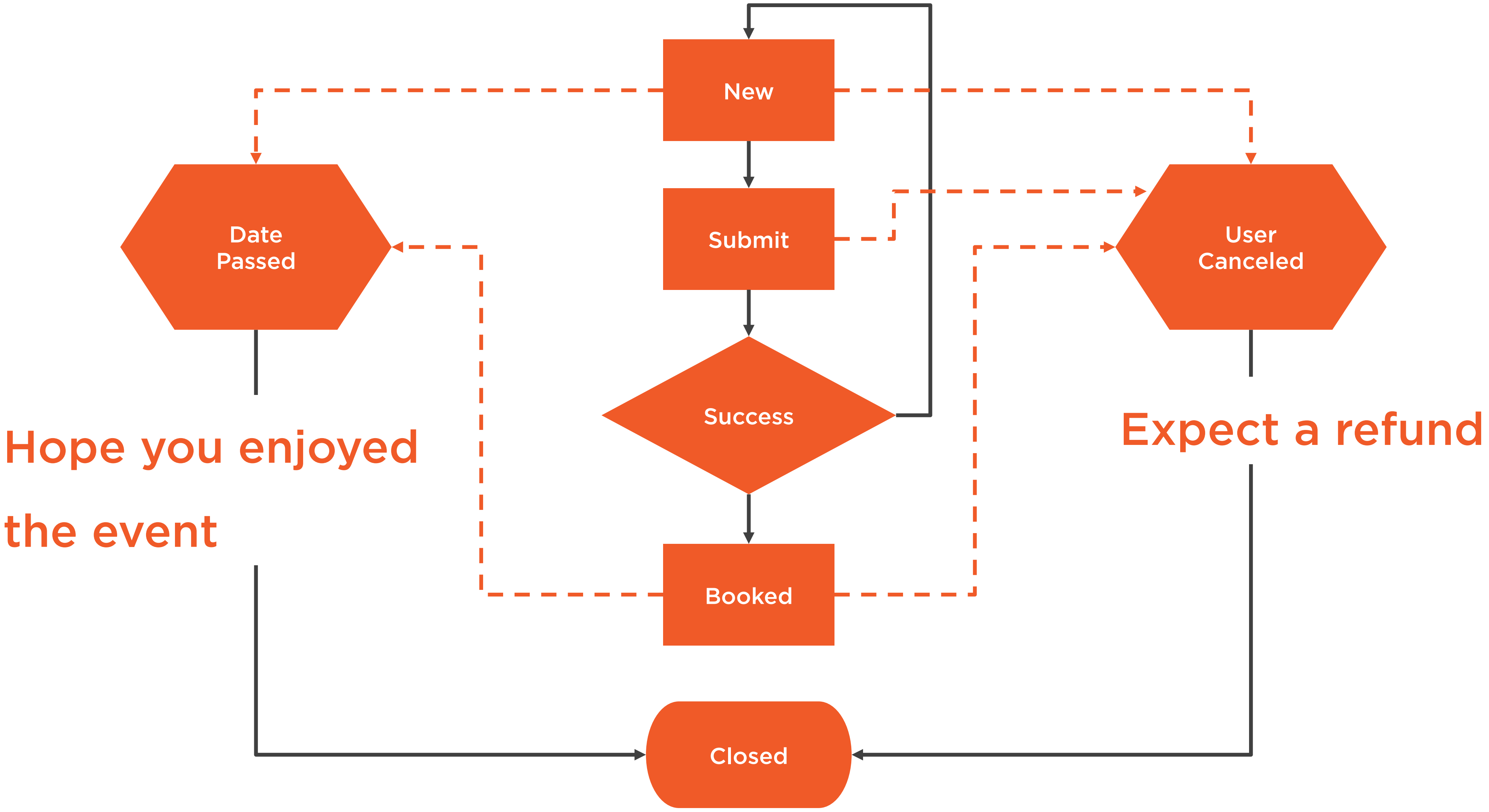
Event Booking Process



Event Booking Process



Event Booking Process



Coming Up

Why Use the State Design Pattern?

Design Challenges

How can an object change its behavior when its internal state changes?

How can state specific behaviors be defined so that states can be added without altering the behavior of existing states?

The State Design Pattern minimizes conditional complexity.

The Naive Approach

Interdependent logic

Time lost managing fields

Difficult to extend

Harder to debug and manage

Benefit of the State Pattern

More modular

Easier to read and maintain

Less difficult to debug

More extensible

Coming Up

The State Design Pattern

Addressing the Challenges

Encapsulates state-specific behaviors within separate state objects

A class delegates the execution of its state-specific behaviors to one state object at a time

Elements of the State Pattern

Context

Maintains an instance of a concrete state as the current state

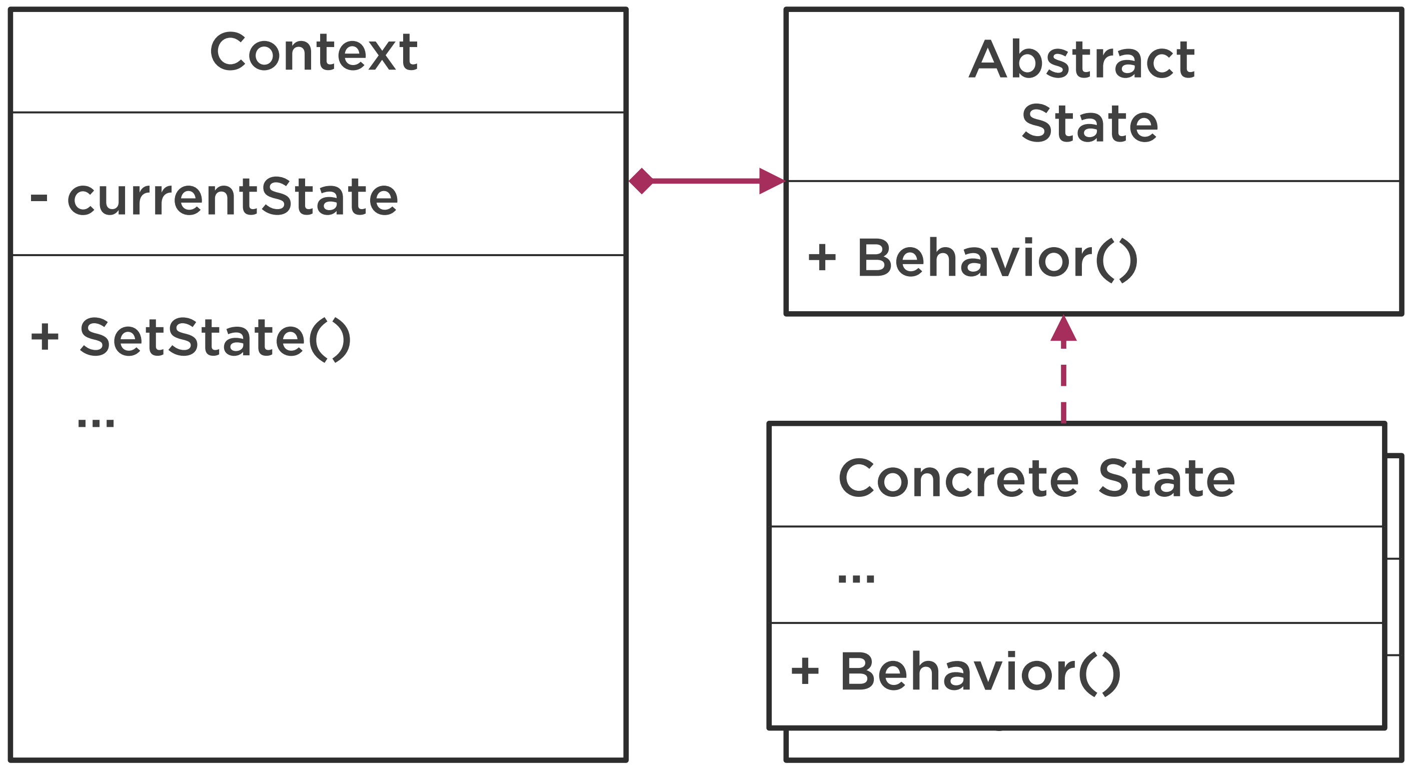
Abstract State

Defines an interface which encapsulates all state-specific behaviors

Concrete State

Implements behaviors specific to a particular state of context

Anatomy of the State Pattern



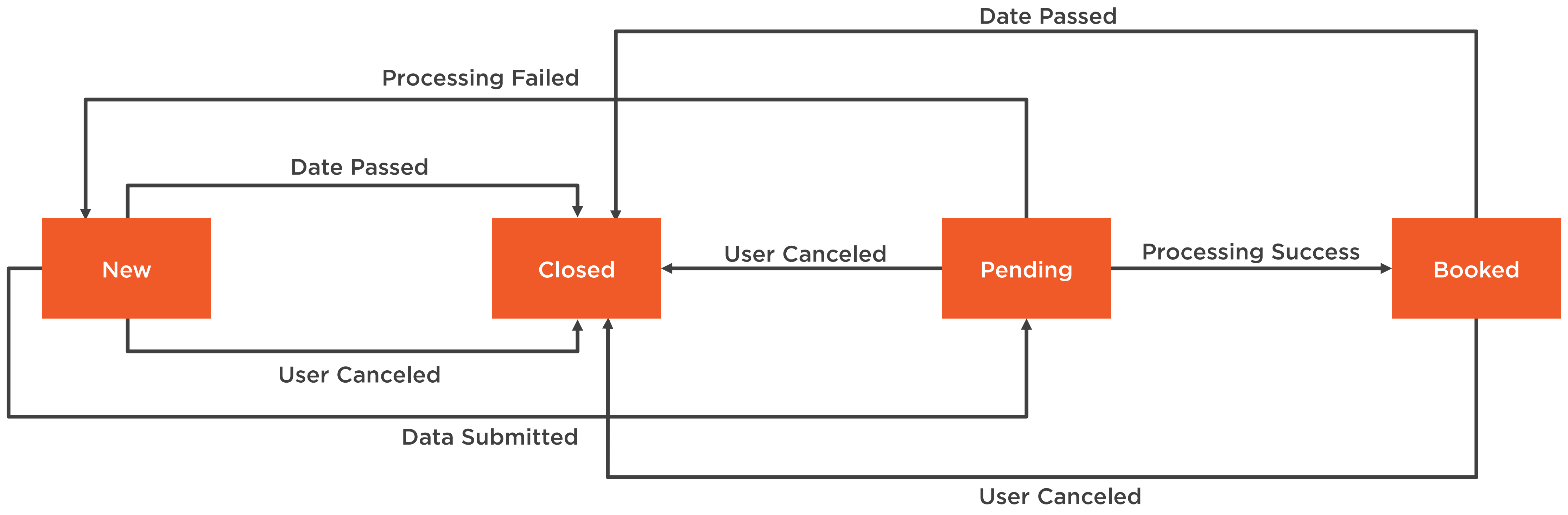
The State Pattern Approach

A list of possible states

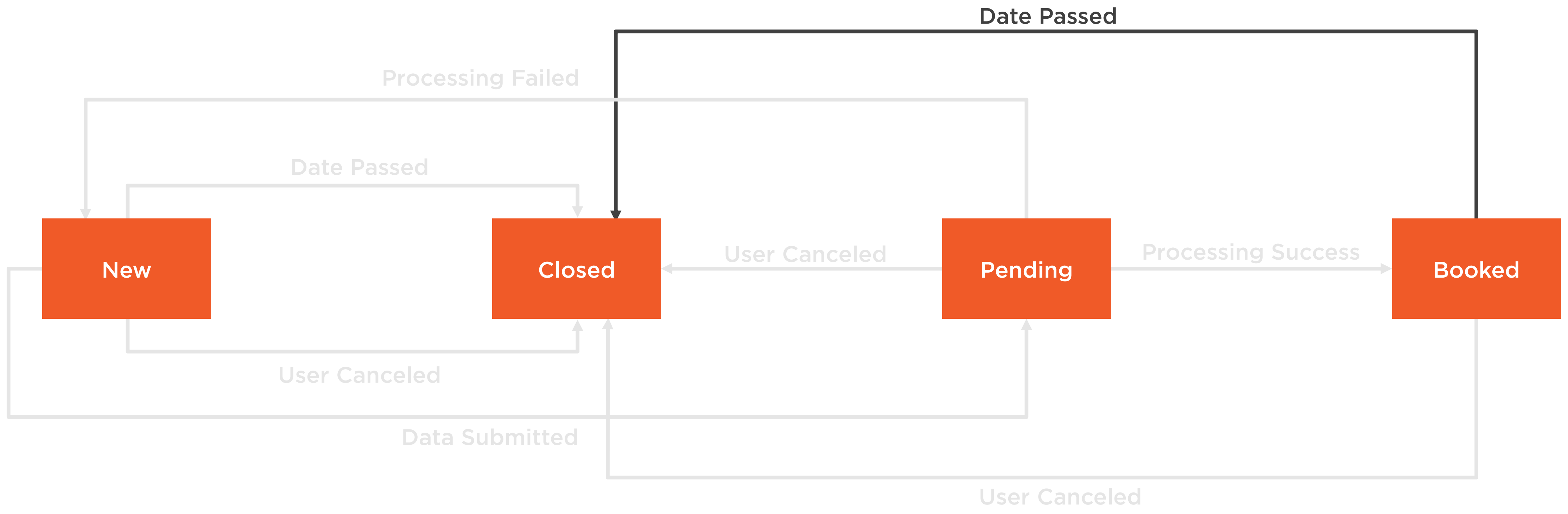
The conditions for transitioning between those states

The state its in when initialized, or its initial state

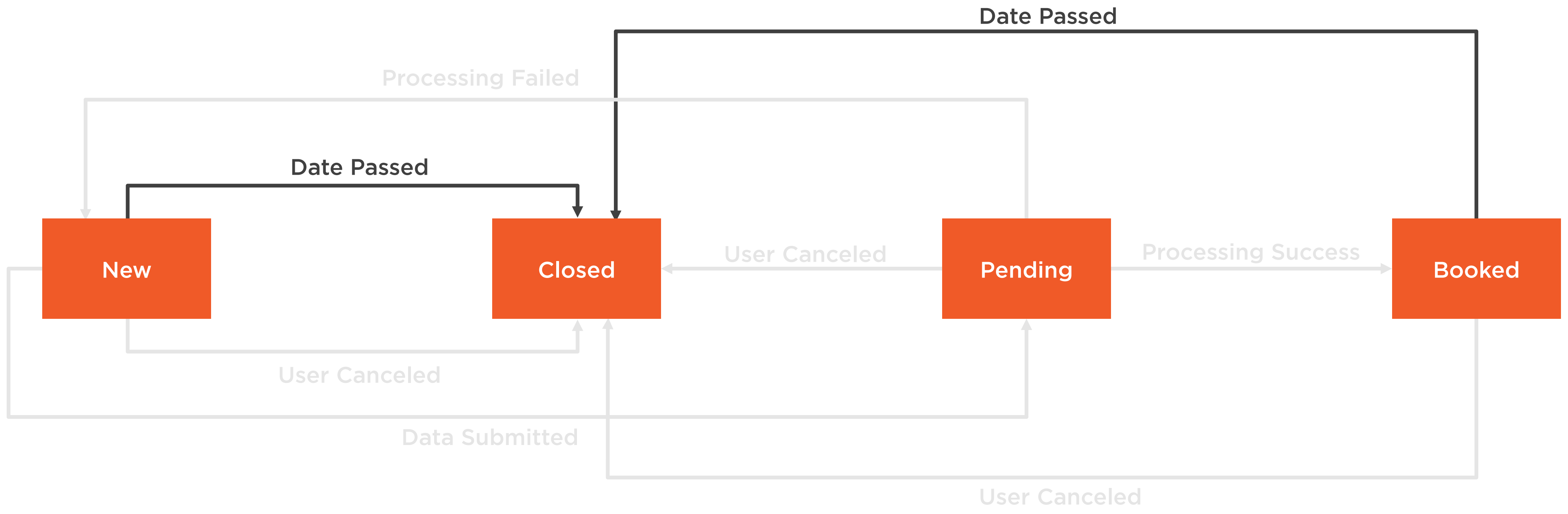
A State Pattern for the Booking Process



A State Pattern for the Booking Process



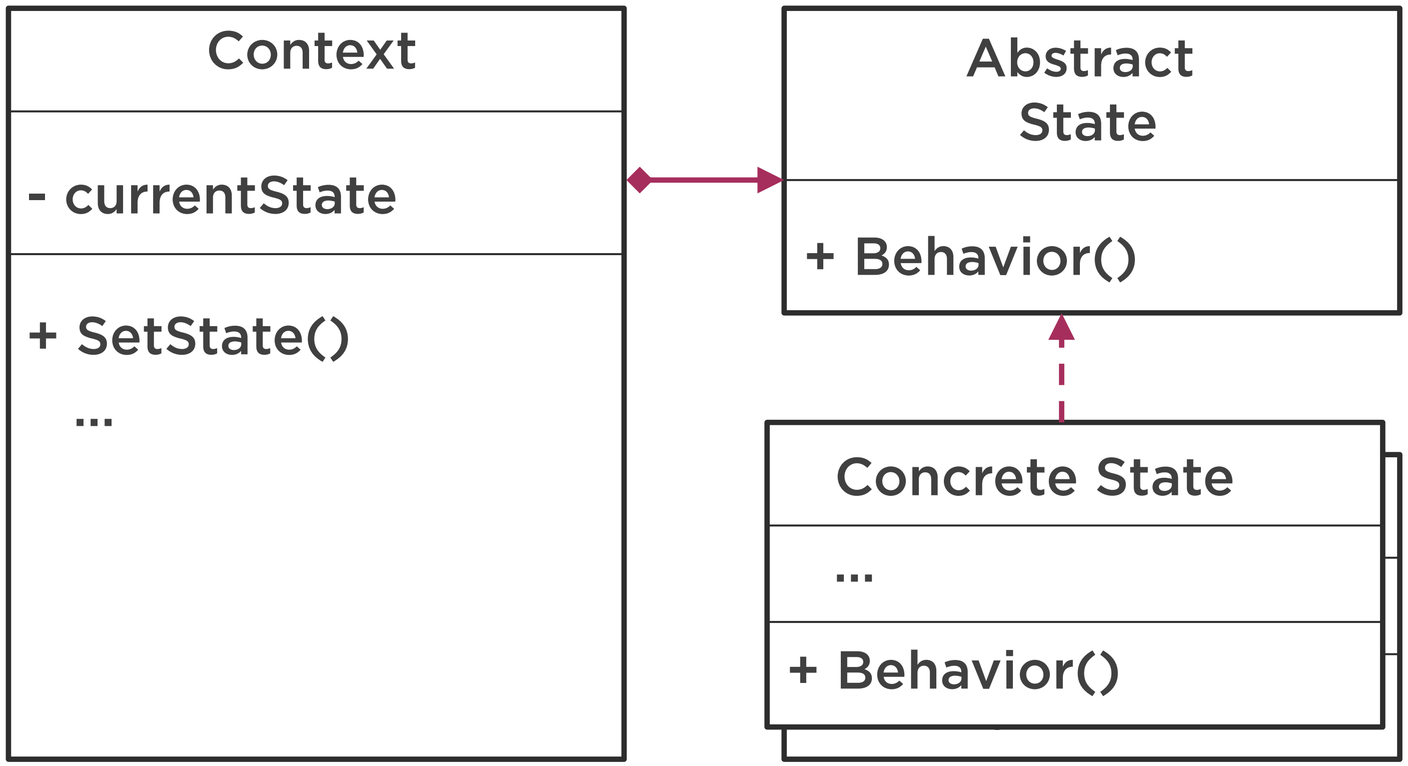
A State Pattern for the Booking Process



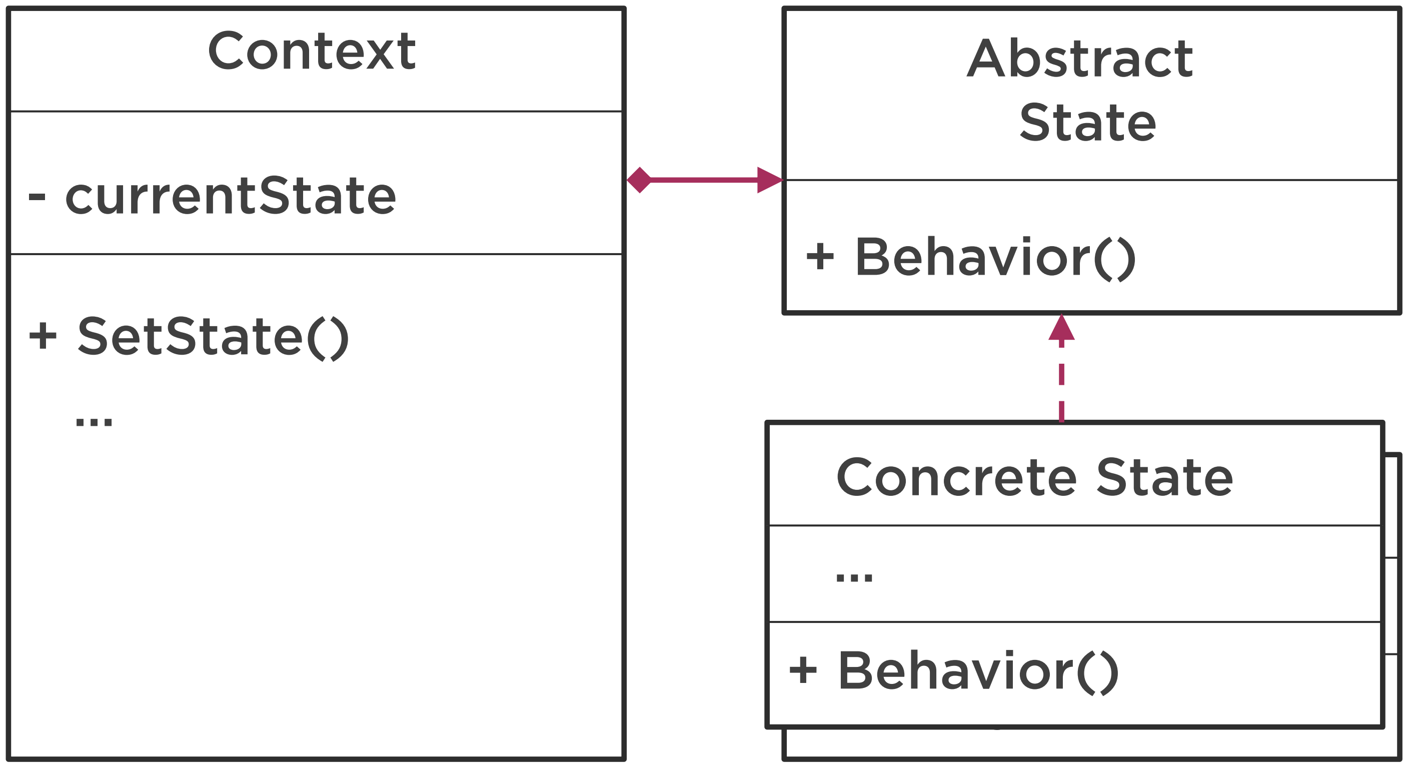
Coming Up

The Abstract State

Anatomy of the State Pattern



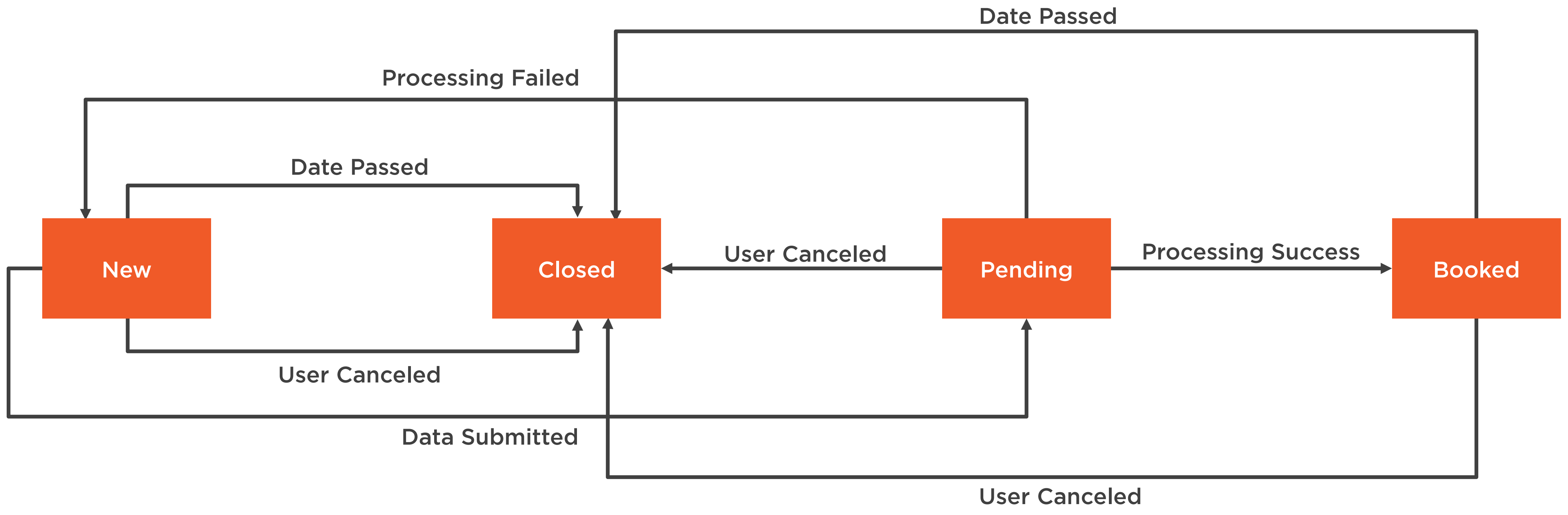
Anatomy of the State Pattern



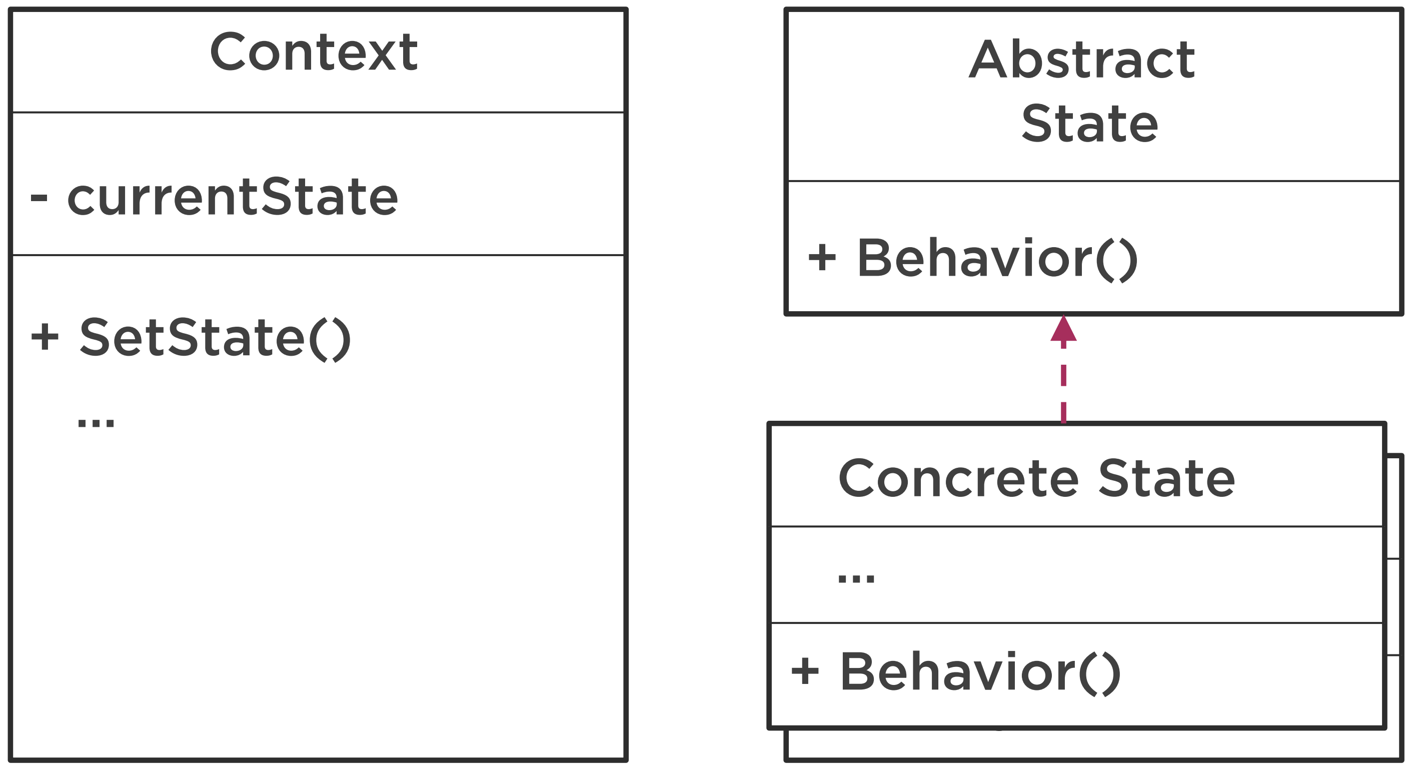
Coming Up

Concrete States

A State Pattern for the Booking Process



Anatomy of the State Pattern



Benefit of Finite State Machines

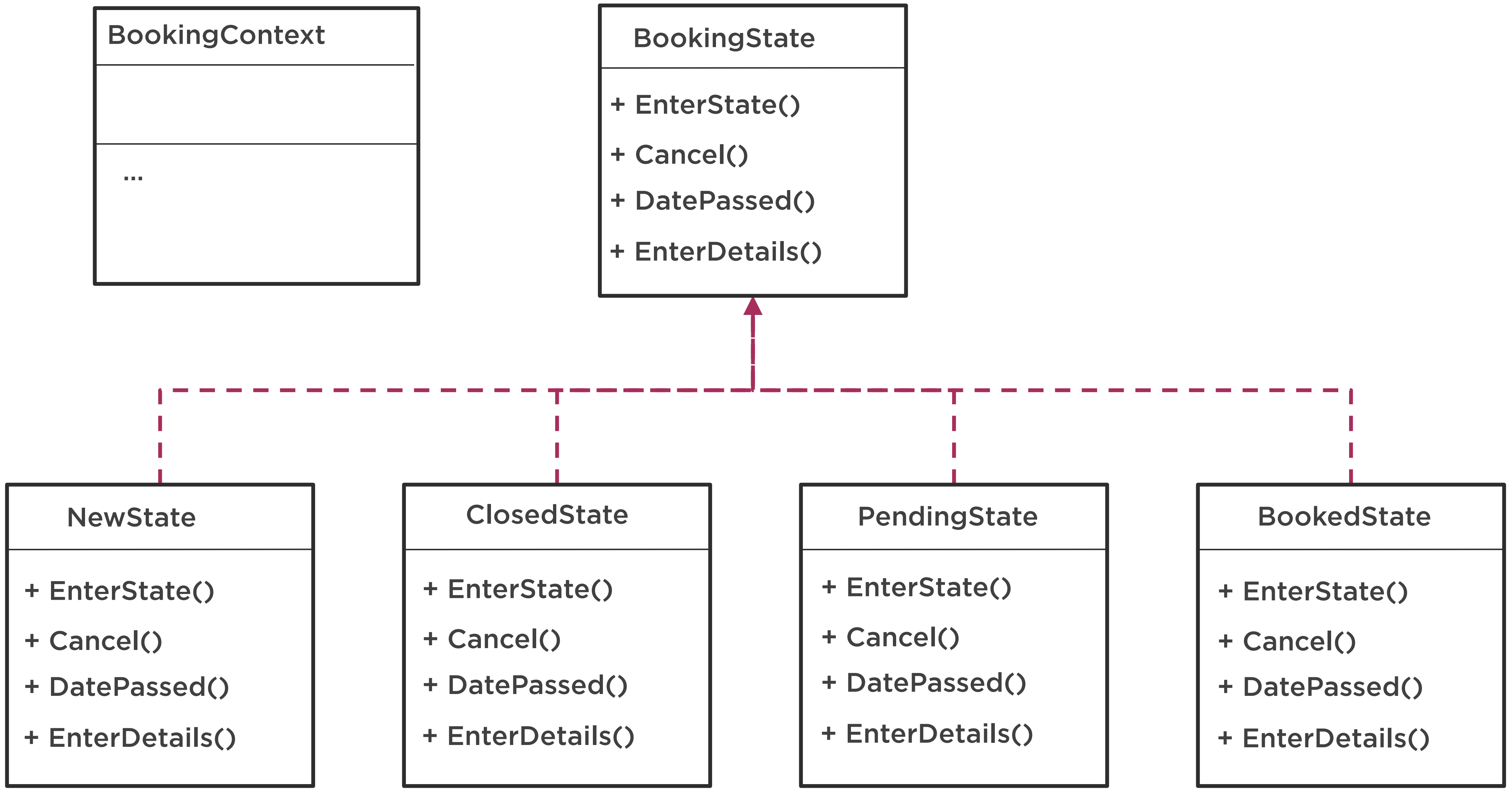
Readability

Maintainability

Easier to debug

Extensibility

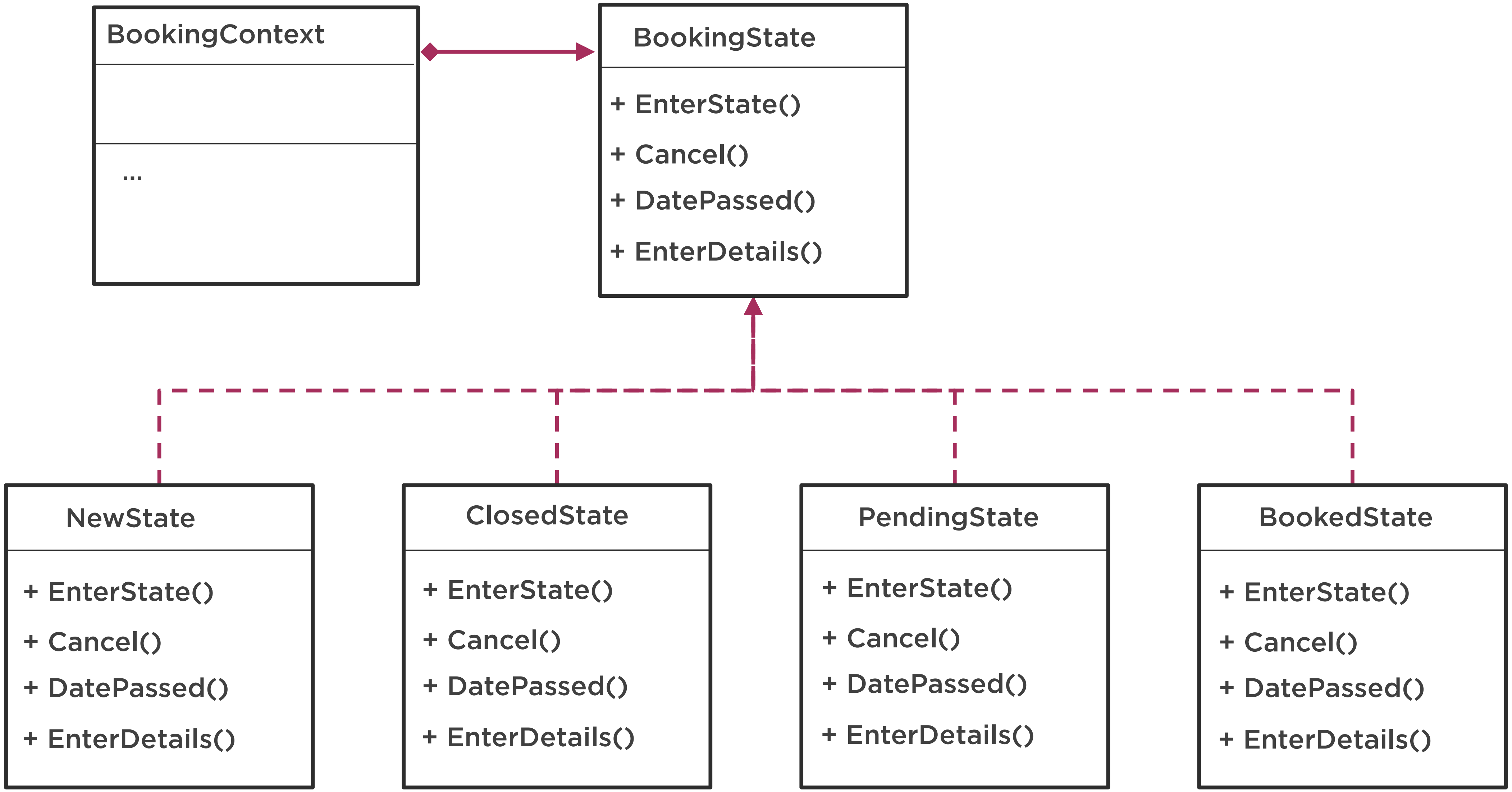
Booking State Pattern



Coming Up

Context and State

Booking State Pattern



Elements of the State Pattern

Context

Maintains an instance of a concrete state as the current state

Abstract State

Defines an interface which encapsulates all state-specific behaviors

Concrete State

Implements behaviors specific to a particular state of context

Coming Up

Implementing the Pattern

A State Pattern for the Booking Process

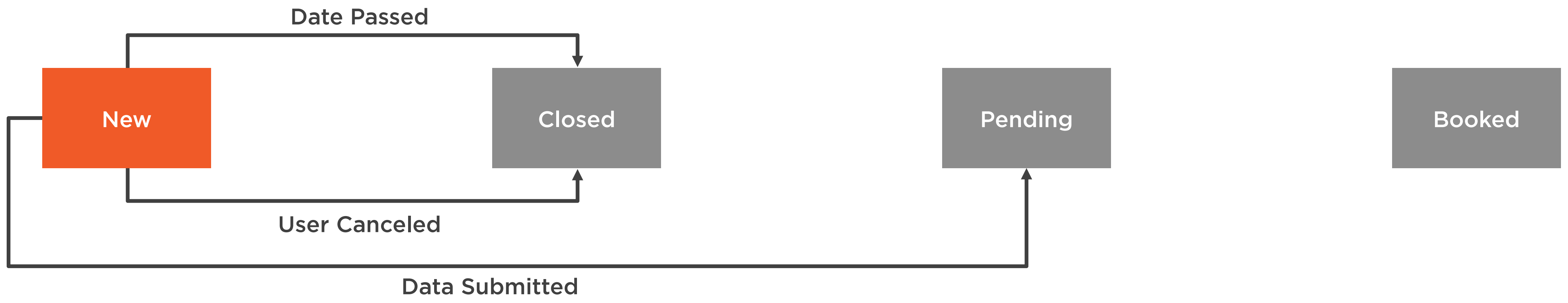
New
Closed
Pending
Booked

A State Pattern for the Booking Process

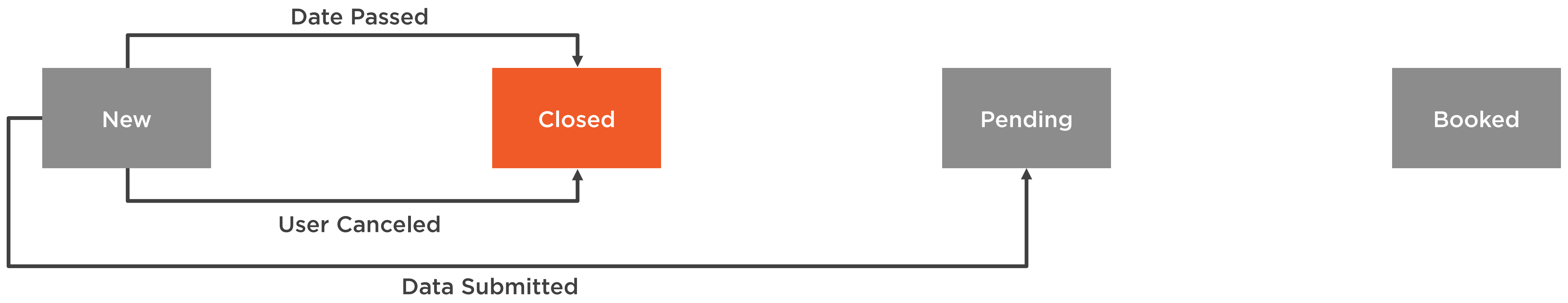
New
Closed
Pending
Booked

- Assign a booking ID
- Display booking status
- Provide for data entry
- Submit for processing

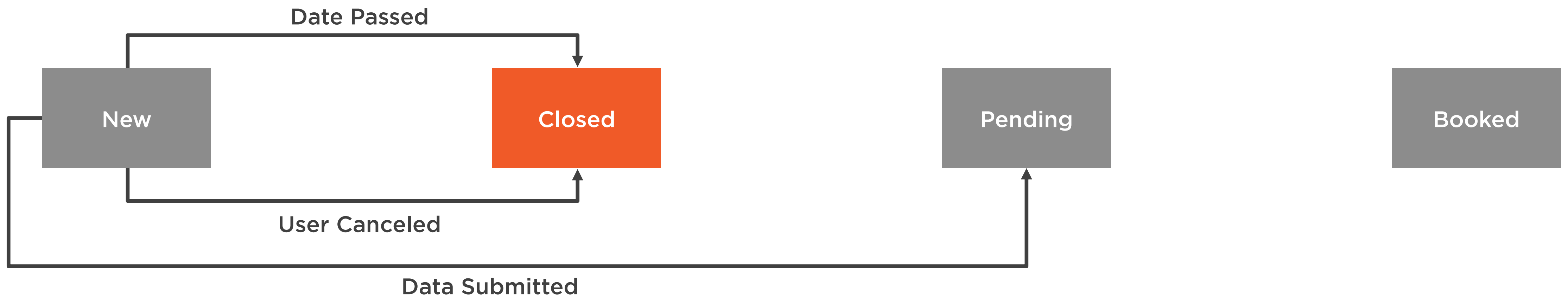
A State Pattern for the Booking Process



A State Pattern for the Booking Process



A State Pattern for the Booking Process

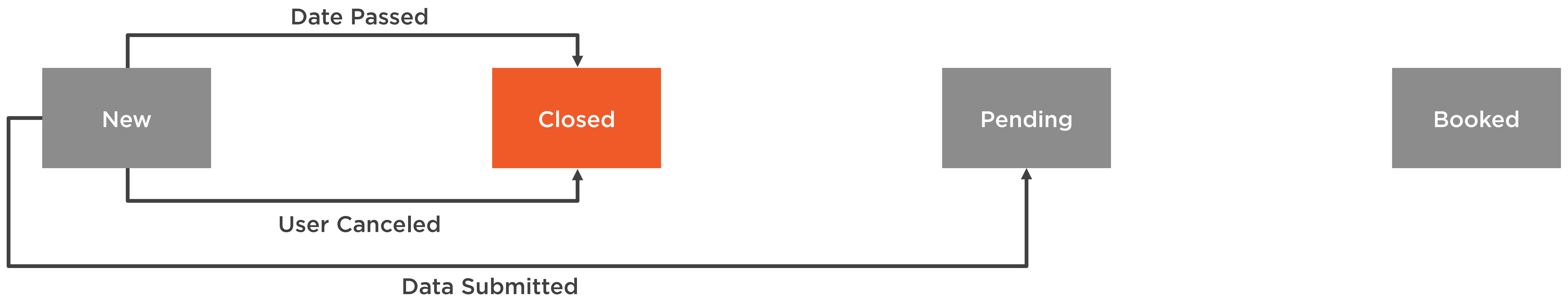


- Display booking state
- Display reason

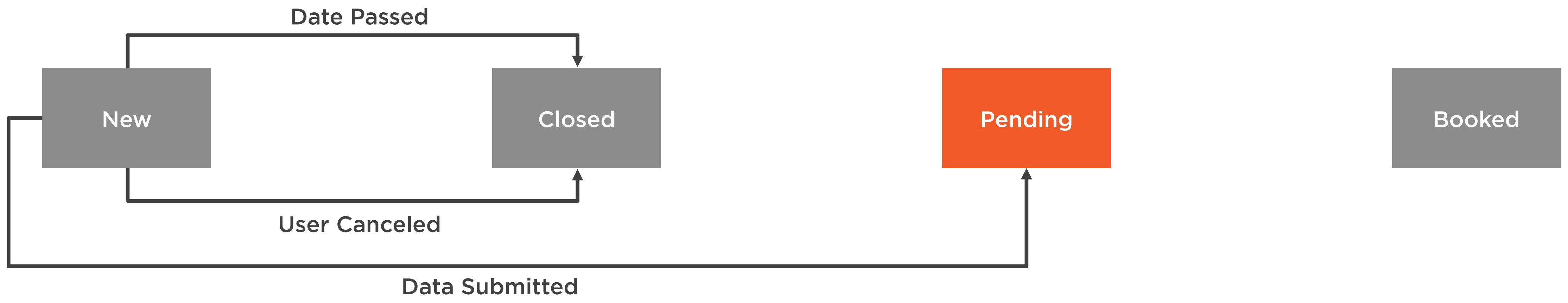
A State Pattern for the Booking Process

New
Closed
Pending
Booked

A State Pattern for the Booking Process

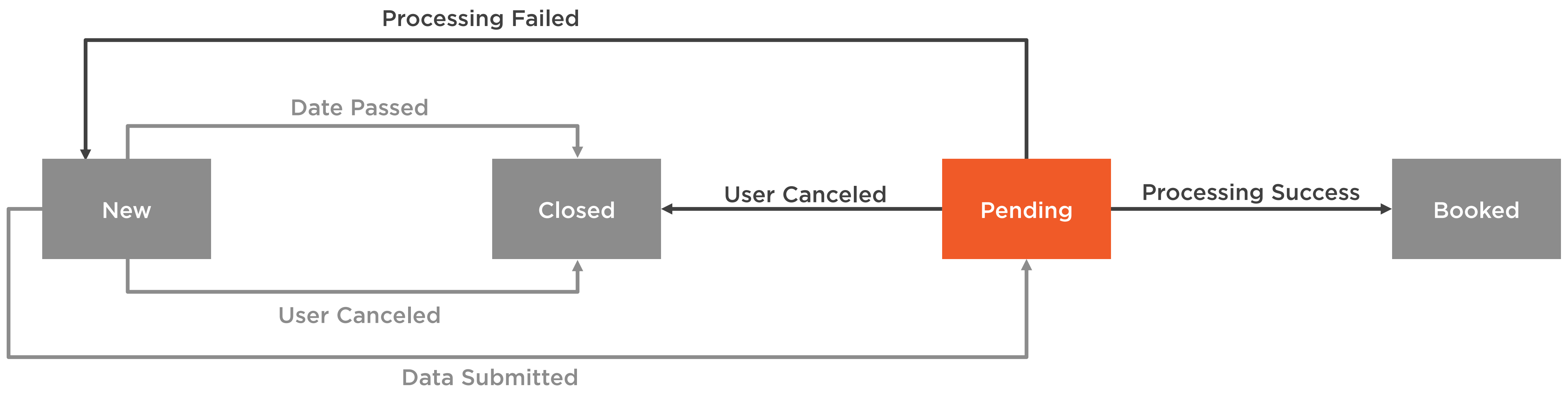


A State Pattern for the Booking Process



- **Display status**
- **Submit for processing**
- **Handle response**

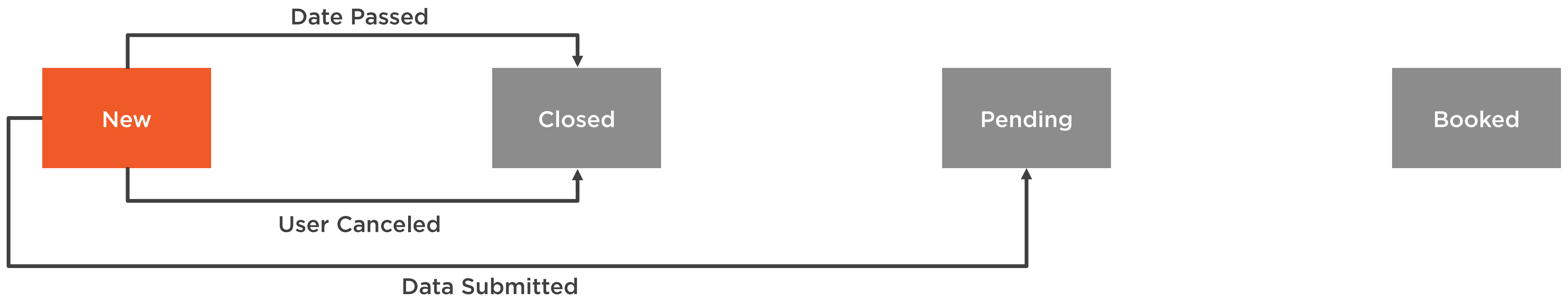
A State Pattern for the Booking Process



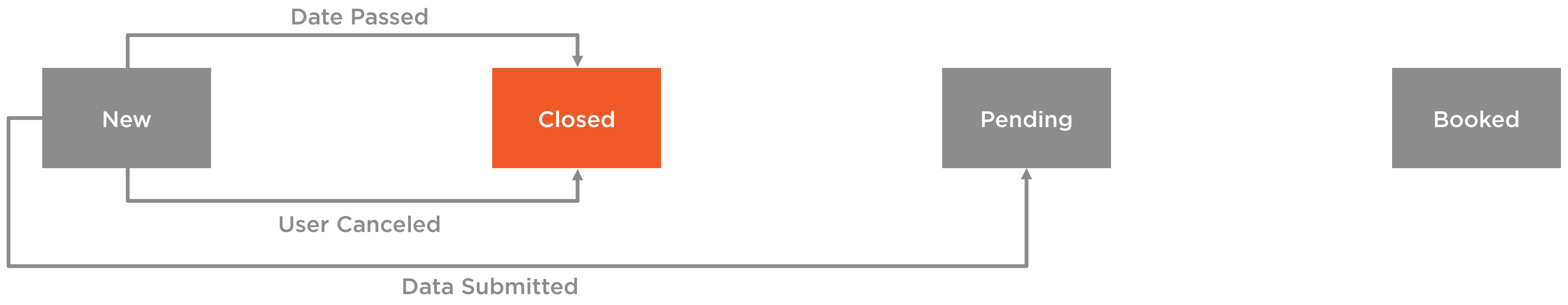
A State Pattern for the Booking Process

New
Closed
Pending
Booked

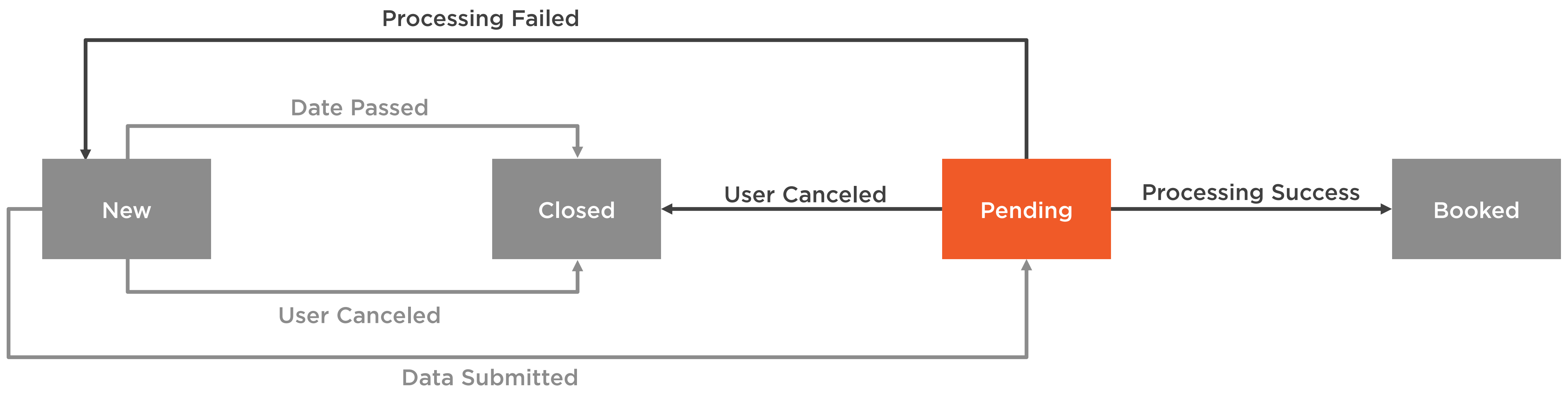
A State Pattern for the Booking Process



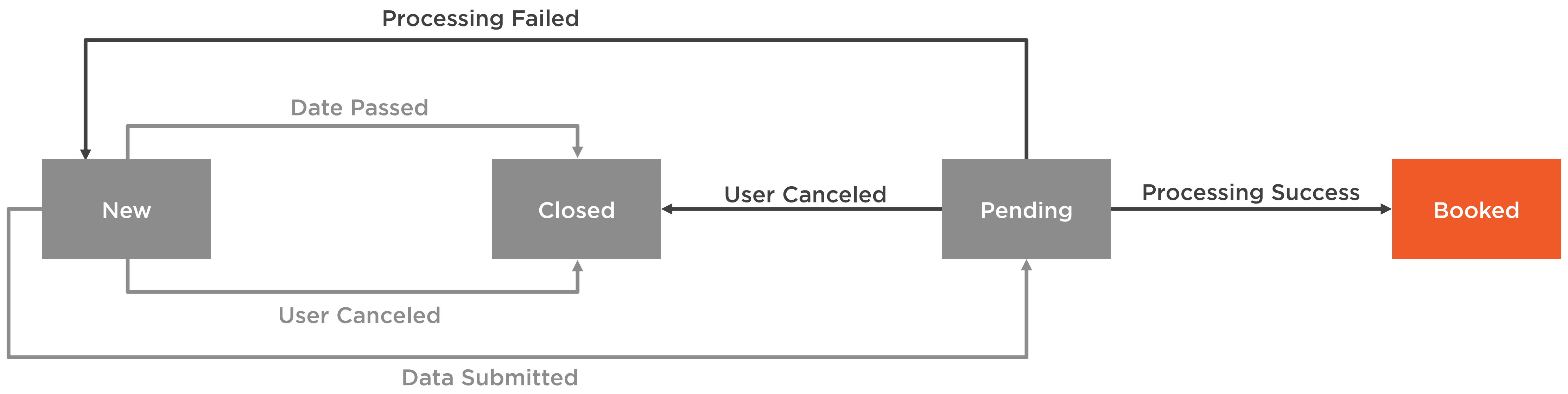
A State Pattern for the Booking Process



A State Pattern for the Booking Process

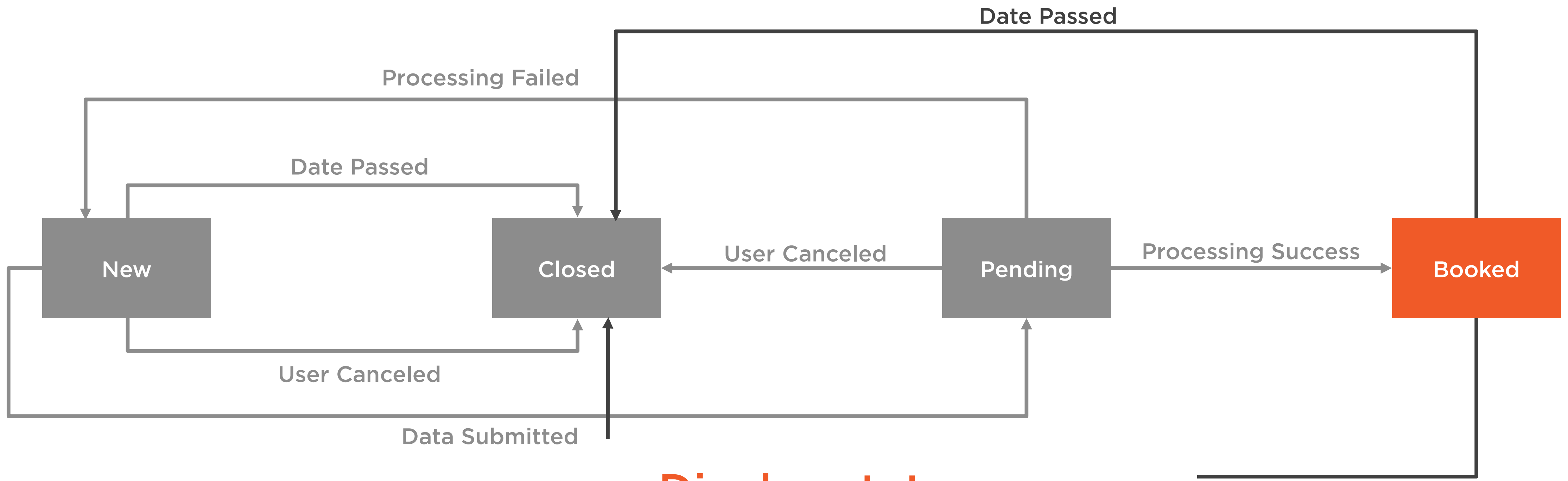


A State Pattern for the Booking Process



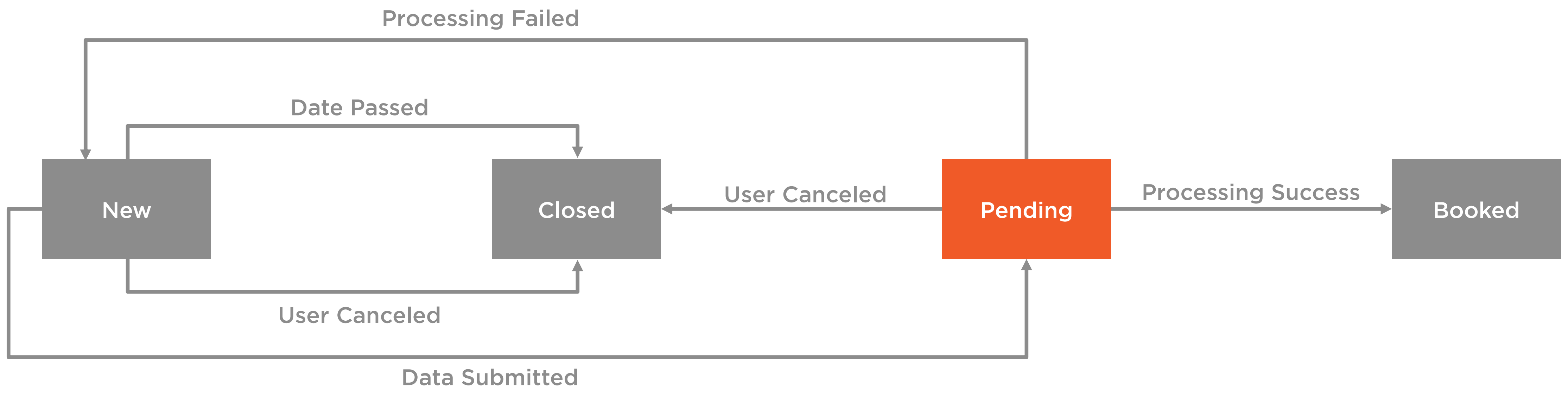
- Display status
- Enjoy the event

A State Pattern for the Booking Process



- Display status
- Enjoy the event

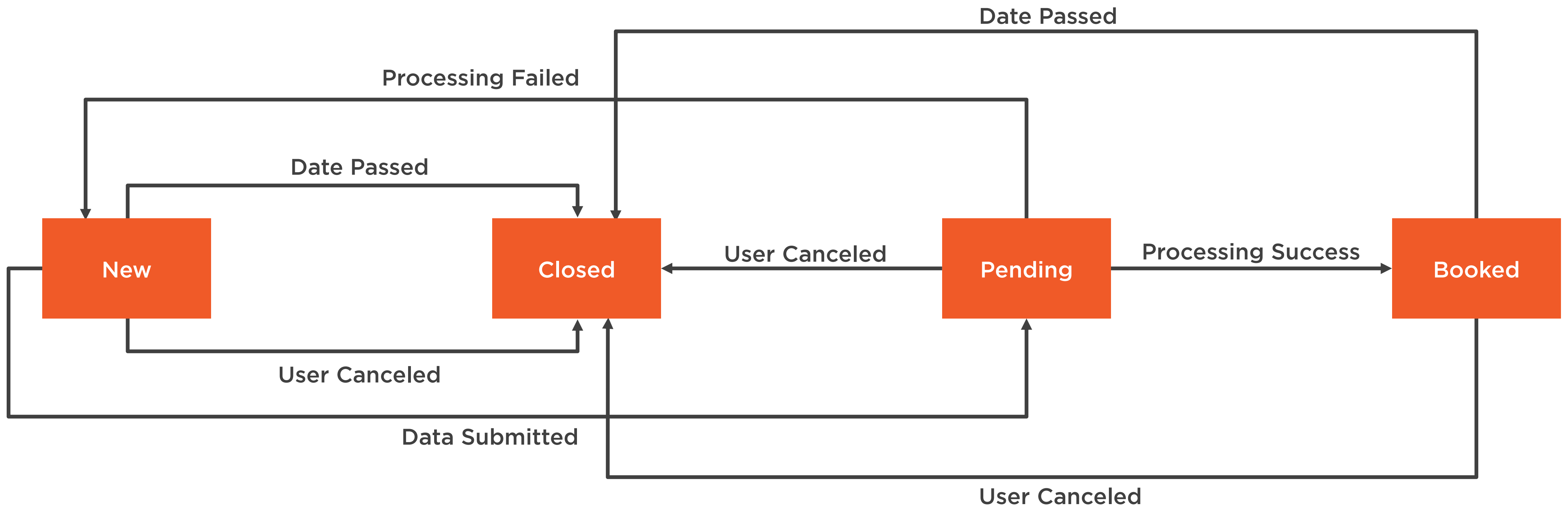
A State Pattern for the Booking Process



Coming Up

Module Conclusion

A State Pattern for the Booking Process



Benefits of the State Design Pattern

More modular

Easier to read and maintain

Less difficult to debug

More extensible

Disadvantages of the State Design pattern

Takes time to set up

More moving parts

Potentially less performant

The State Design Pattern is a great addition to your developer’s toolkit.

Thank You!