Developing an Actor to Represent an IoT Temperature Sensor Device

Jason Roberts
.NET MVP

@robertsjason dontcodetired.com

Overview

Design overview

The request-response pattern

Create TemperatureSensor actor class

Create message classes

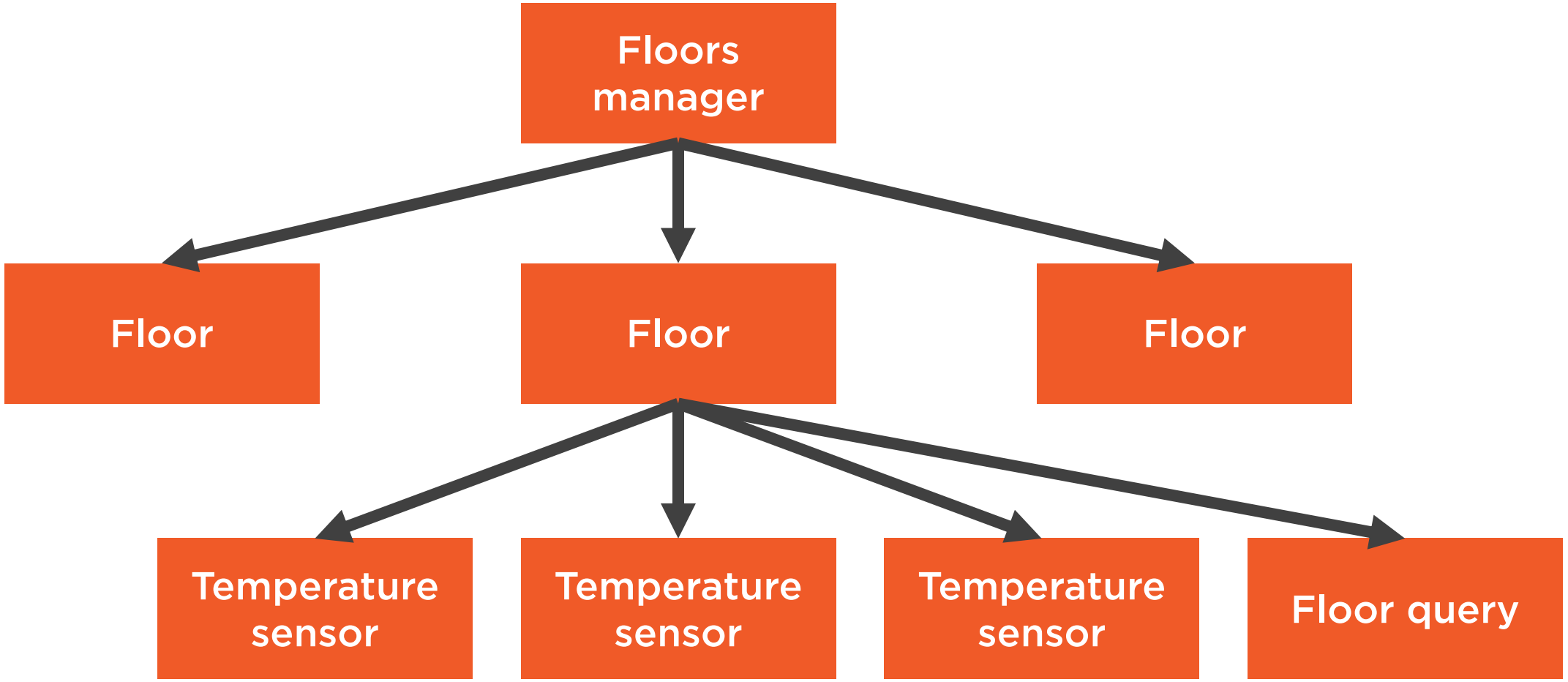
Develop using TDD

- Initializing sensor data
- Updating temperature readings
- Registering new sensors

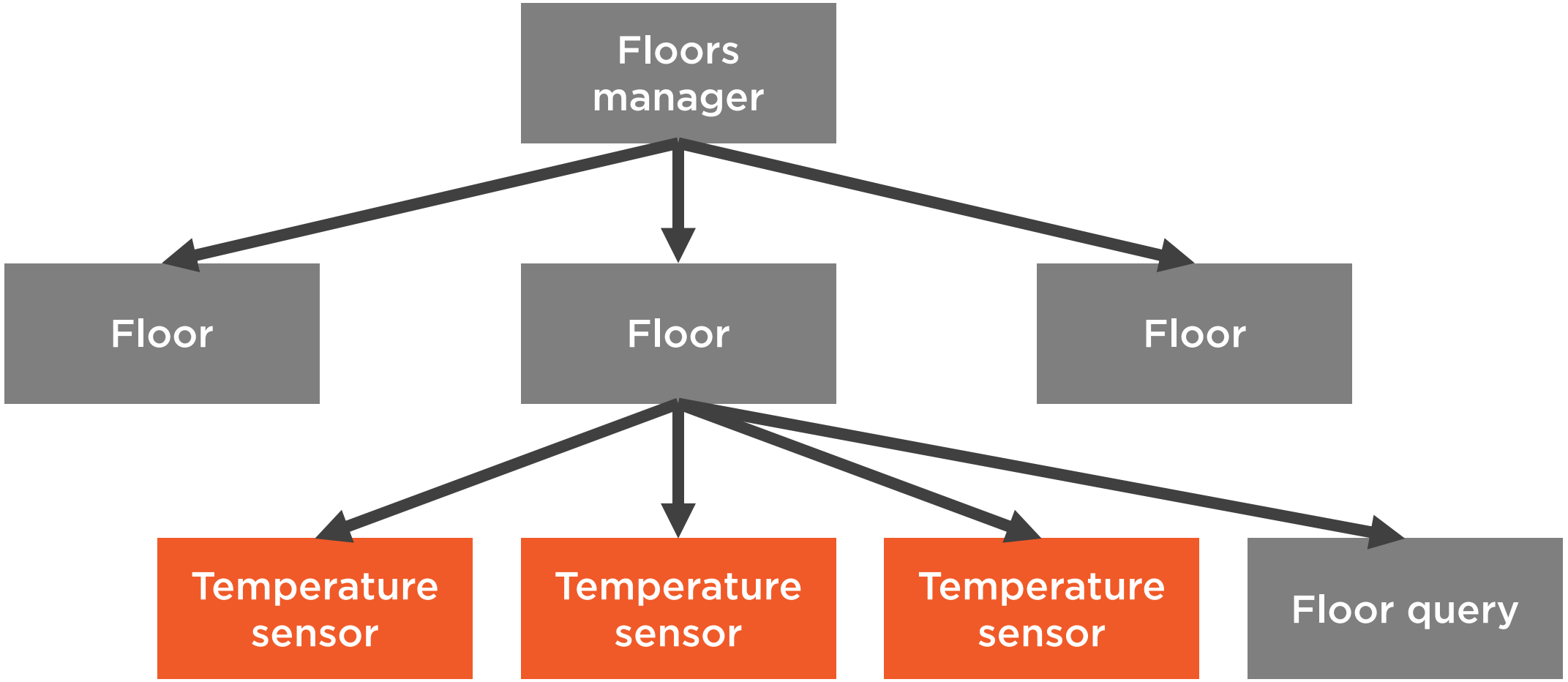
Working TemperatureSensor actor

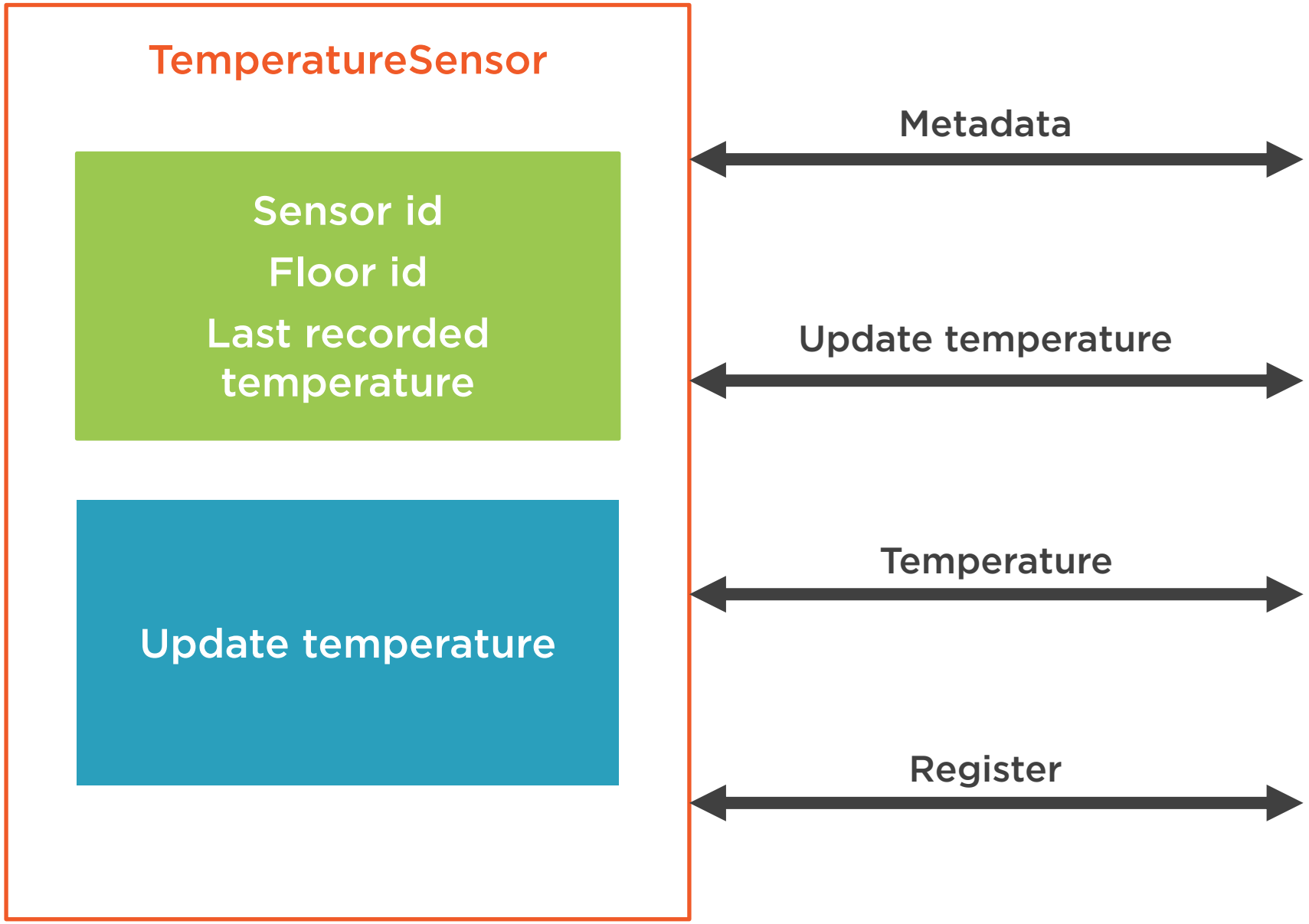
Passing automated tests

Supervision Hierarchy



Supervision Hierarchy




TemperatureSensor
Sensor id
Floor id
Last recorded temperature
Update temperature
Metadata
Update temperature
Temperature
Register

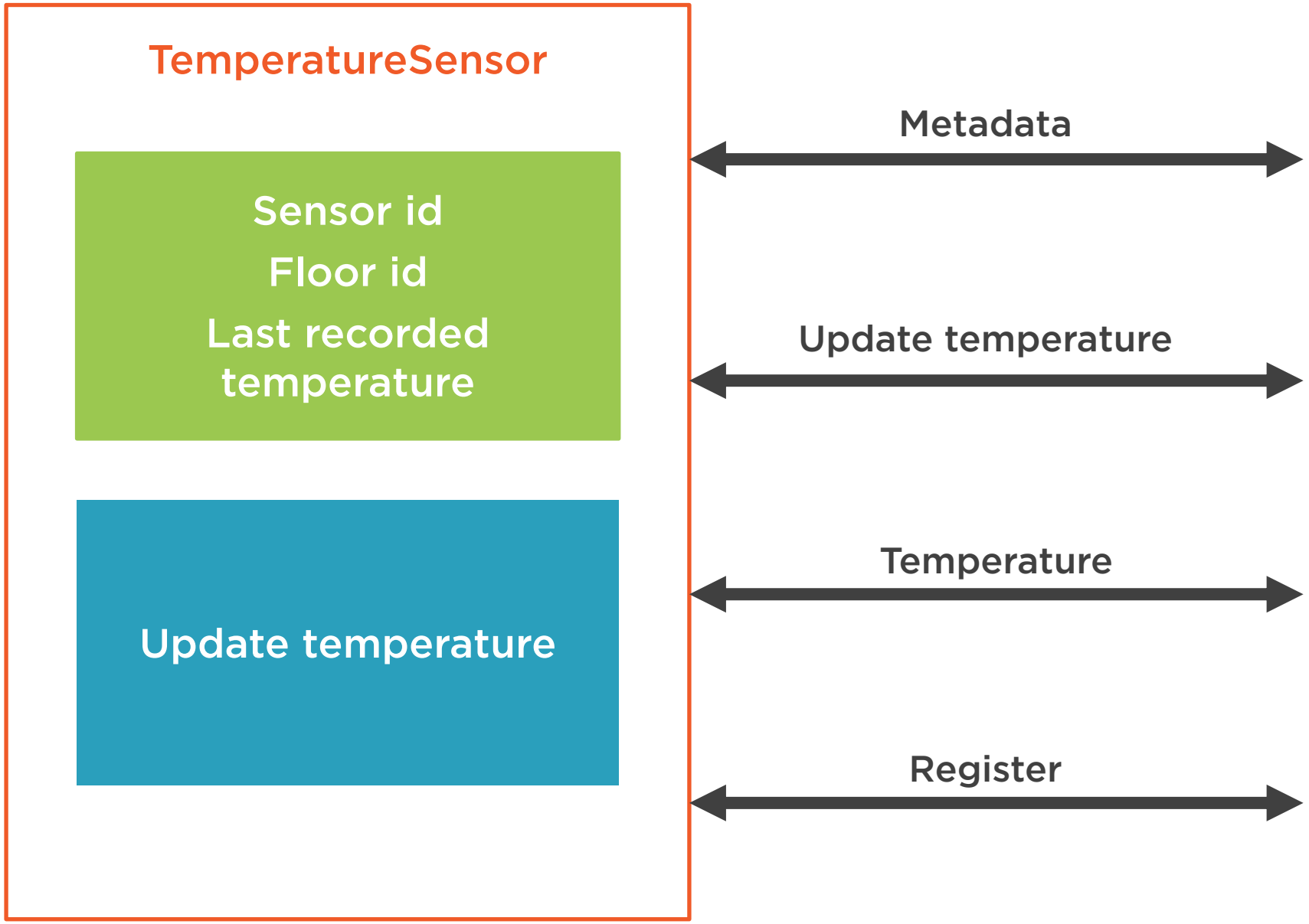
By default, out of the box Akka.NET has at-most-once delivery of messages.

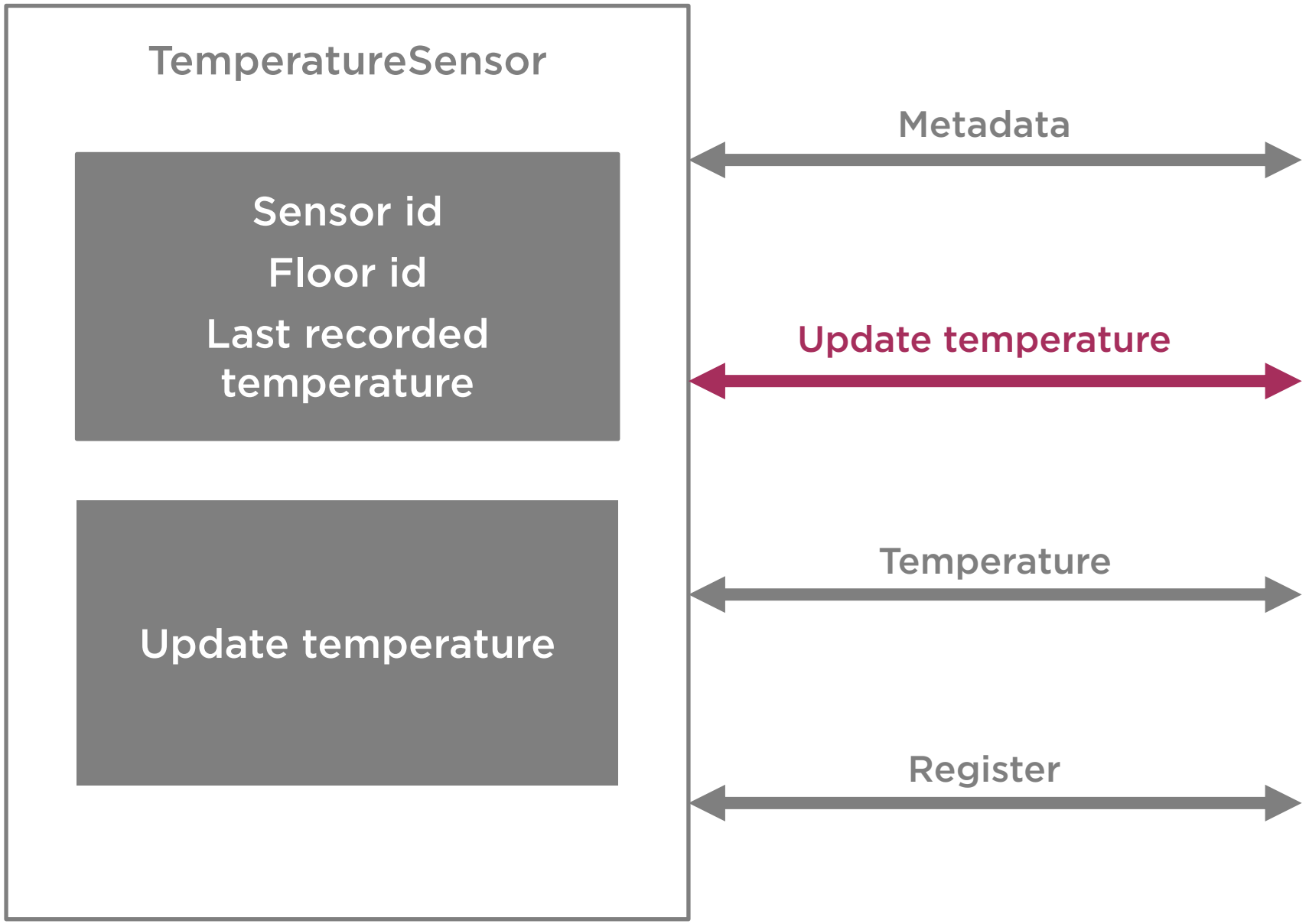
No guarantee of message delivery.

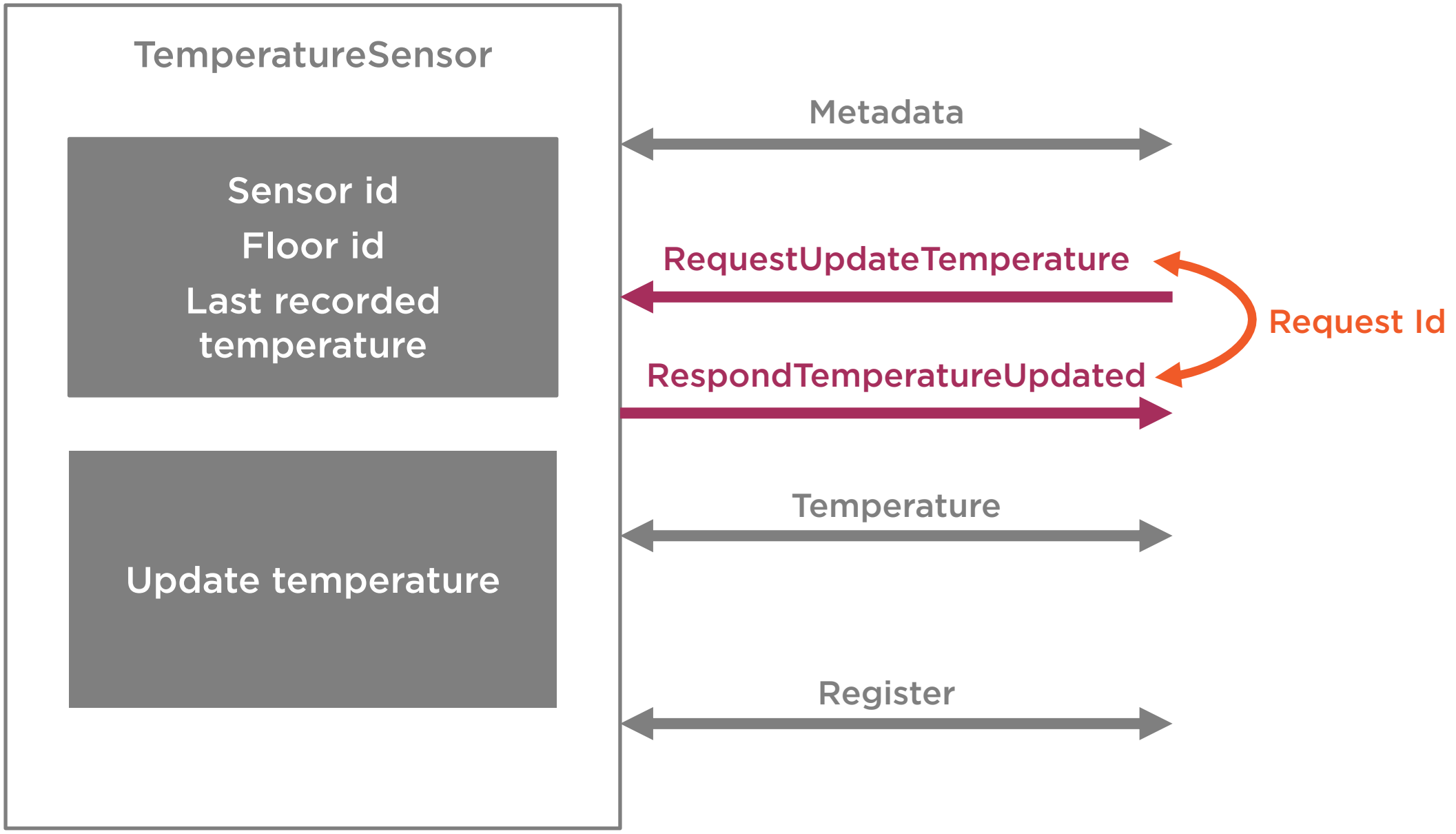
Sender-receiver pair ordering maintained.

“Akka.NET lifts the responsibilities of guarantees to the application itself, i.e. you have to implement them yourself. On the other hand, you are in full control of the guarantees that you want to provide.”

Akka.NET documentation [http://getakka.net]


TemperatureSensor
Sensor id
Floor id
Last recorded temperature
Update temperature
Metadata
Update temperature
Temperature
Register


TemperatureSensor
Sensor id
Floor id
Last recorded temperature
Update temperature
Metadata
Update temperature
Temperature
Register


TemperatureSensor
Sensor id
Floor id
Last recorded temperature
Update temperature
Metadata
RequestUpdateTemperature
RespondTemperatureUpdated
Request Id
Temperature
Register

Summary

- Design overview
- The request-response pattern
- Created TemperatureSensor actor class
- Created message classes
- Developed using TDD
- Completed a working TemperatureSensor actor class
- Passing automated tests

Next:

Creating, Grouping, and Supervising Temperature Sensor Actors