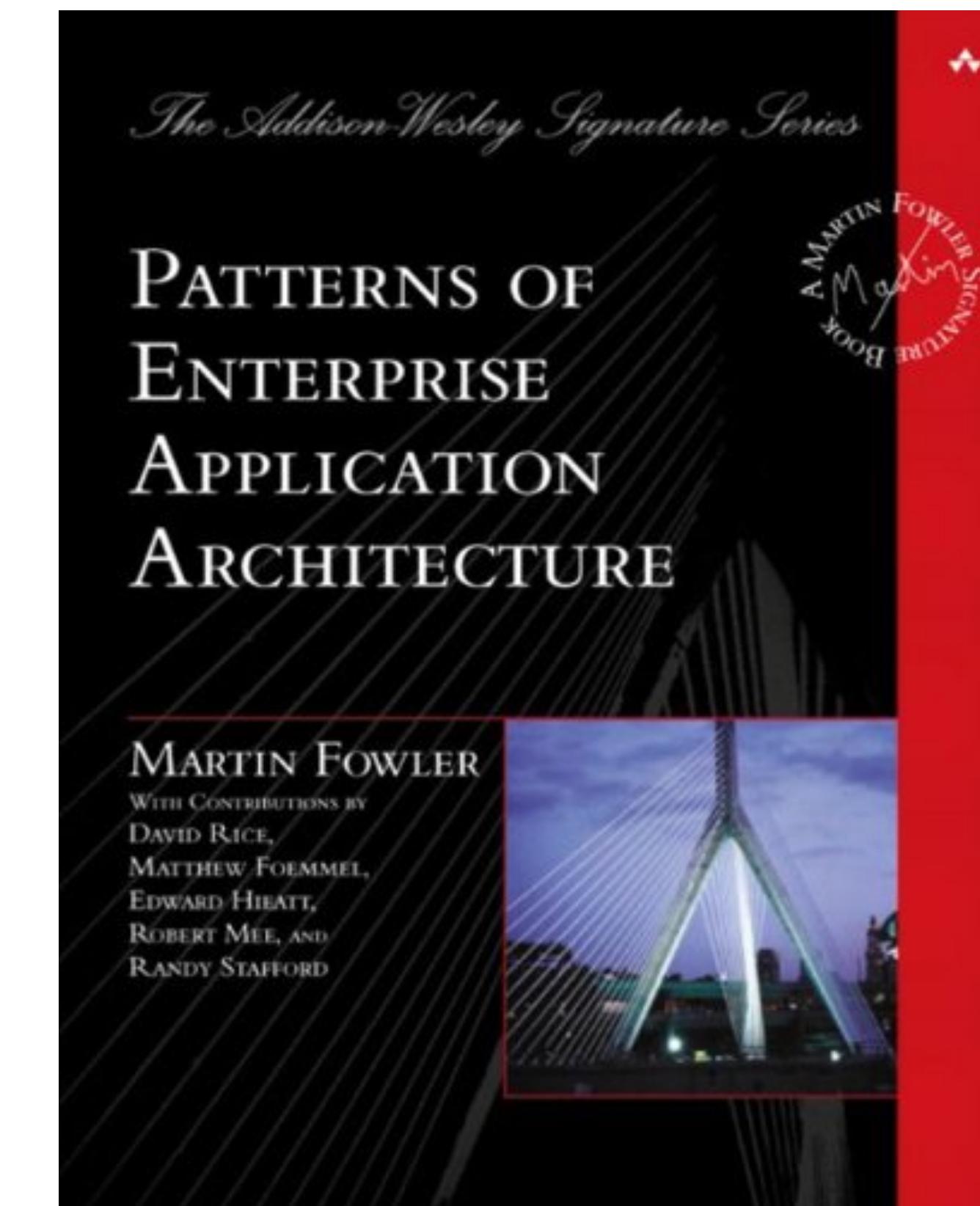
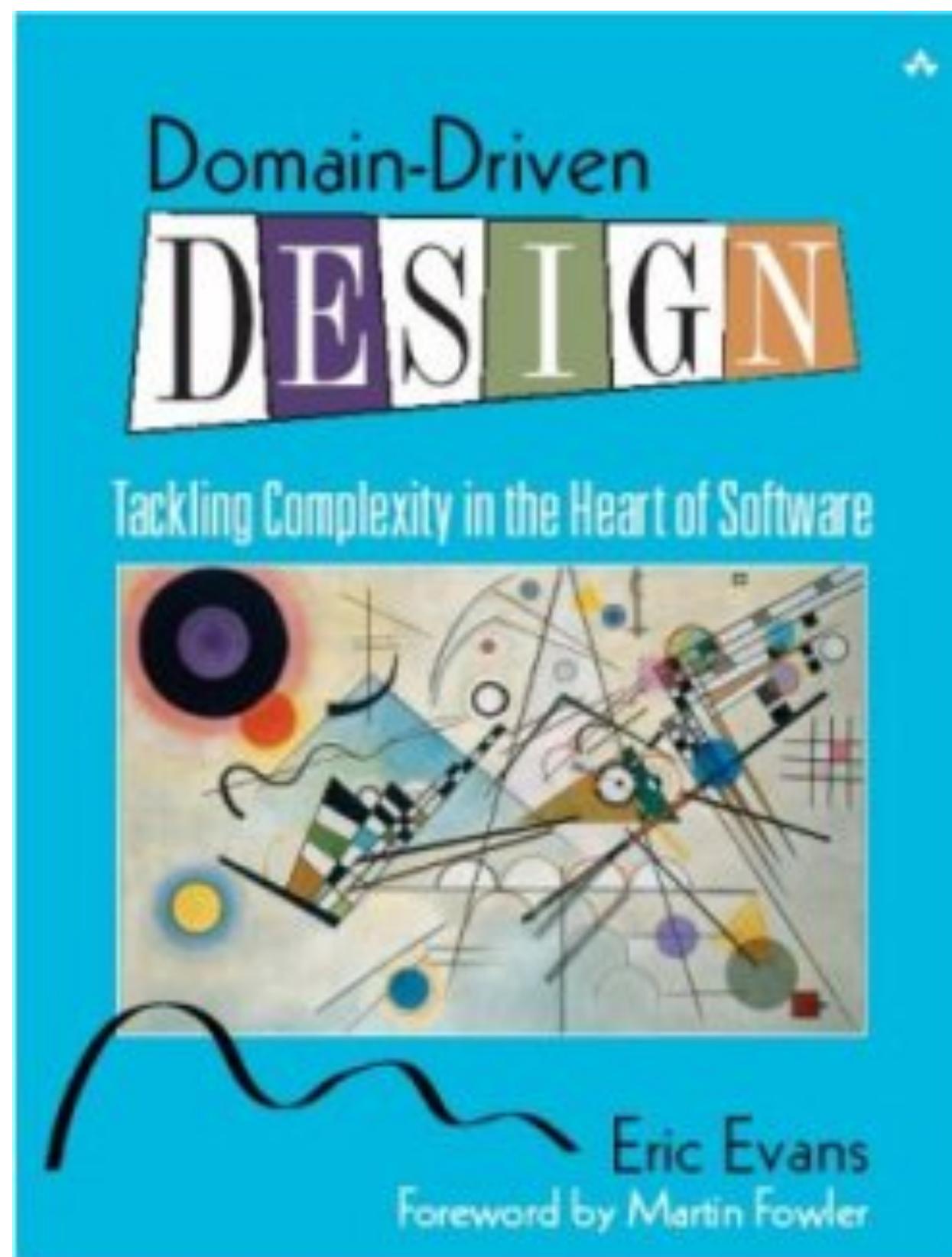


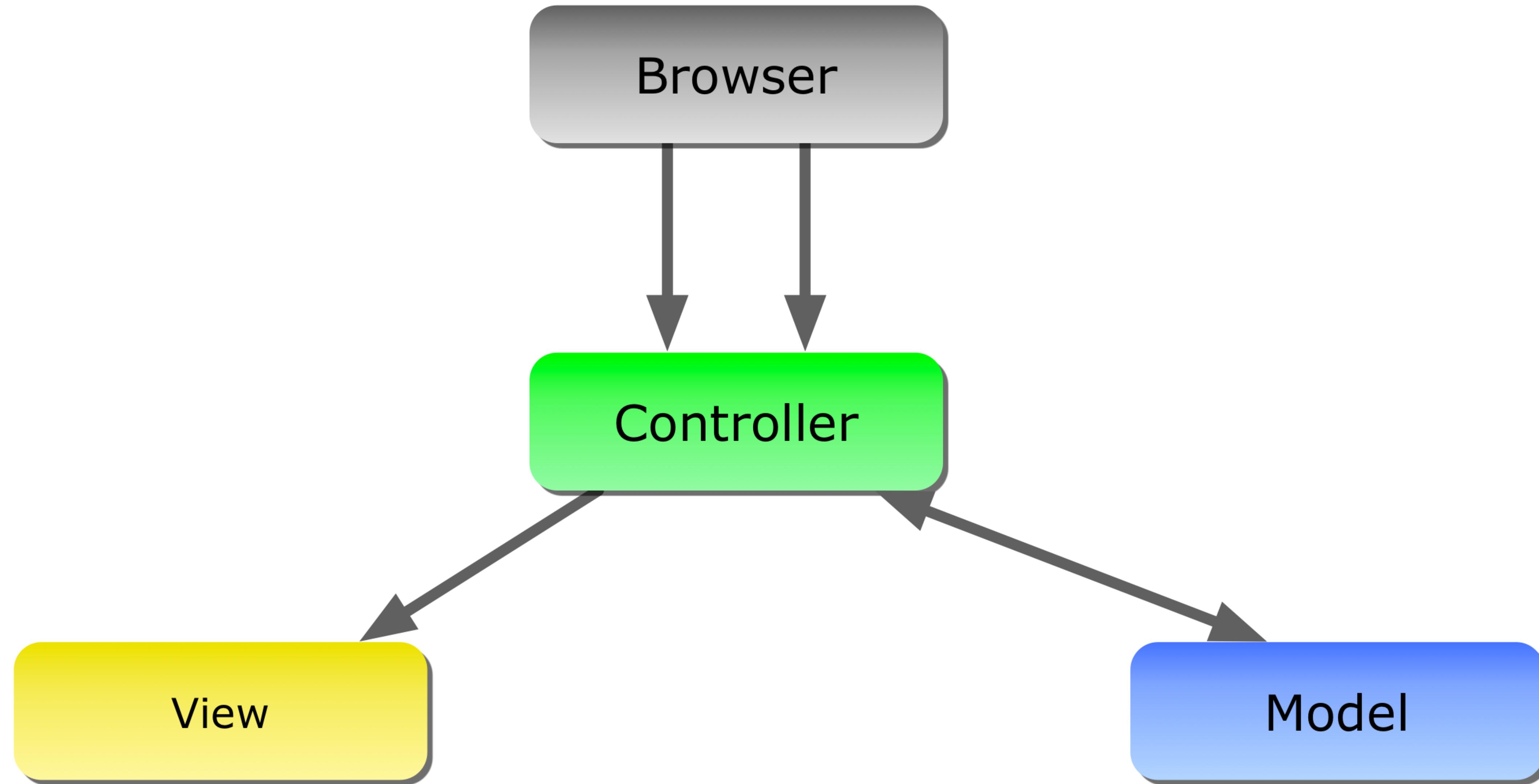
Creating Models

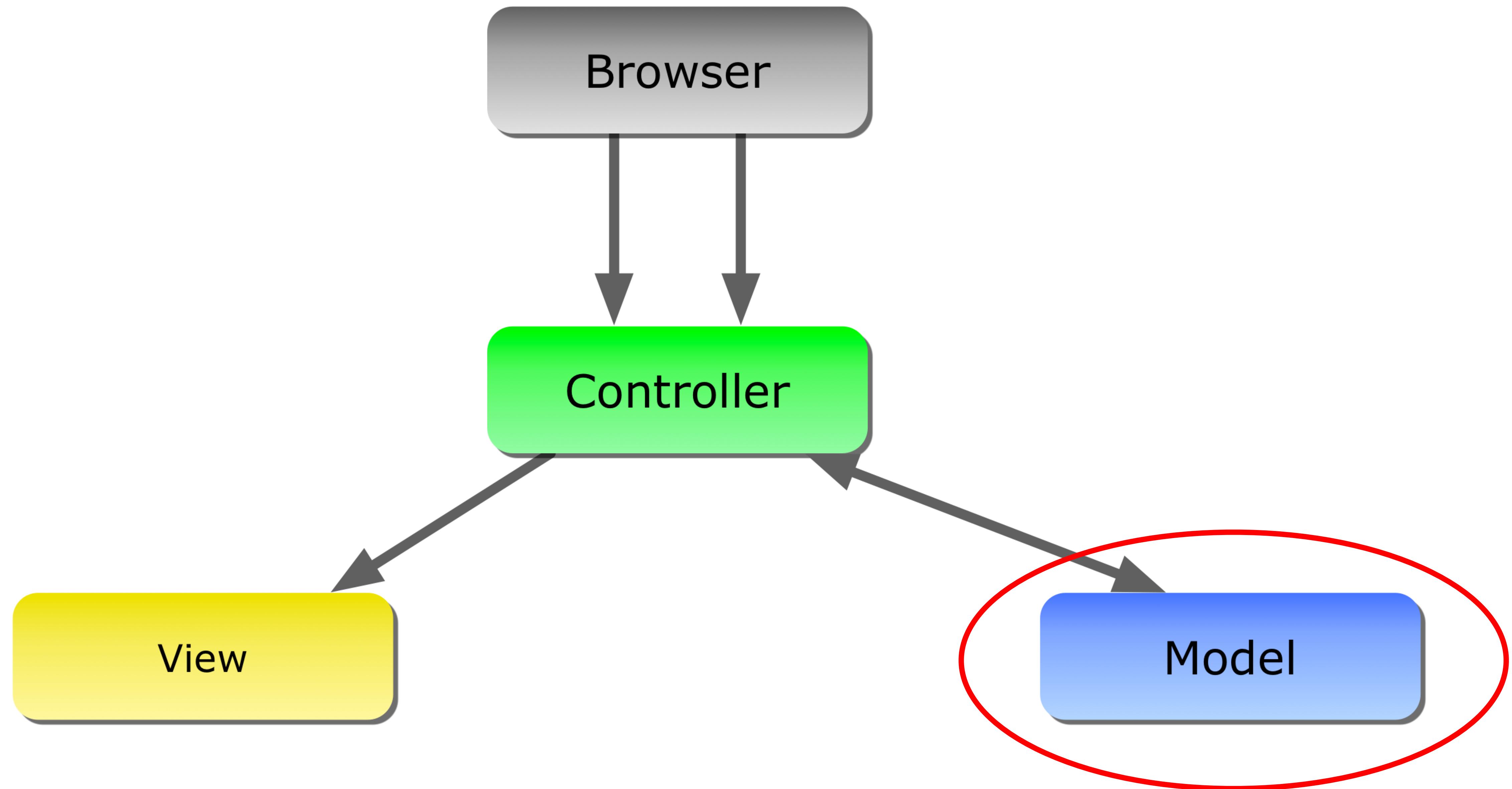
Rob Allen, July 2014

Have you read these?



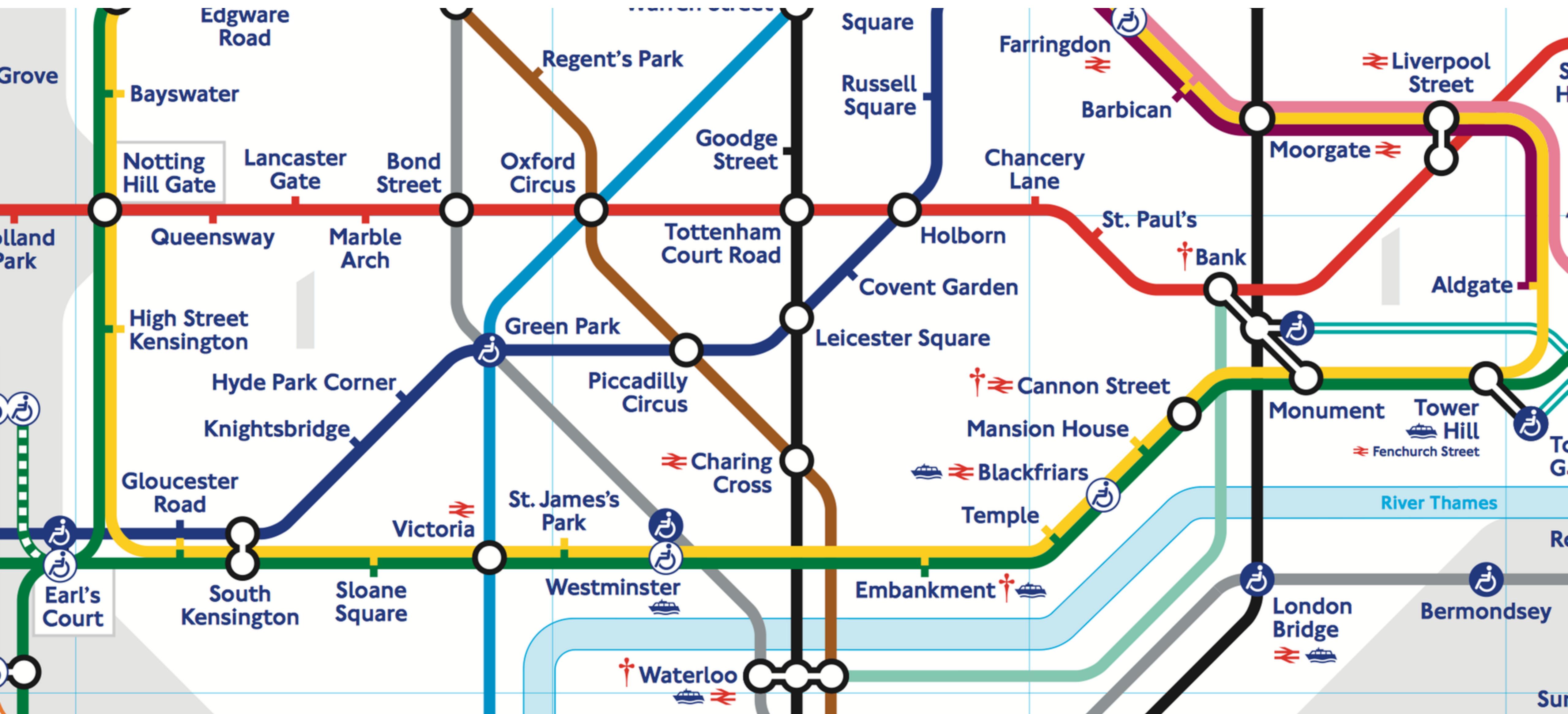
The business logic is the
hard part



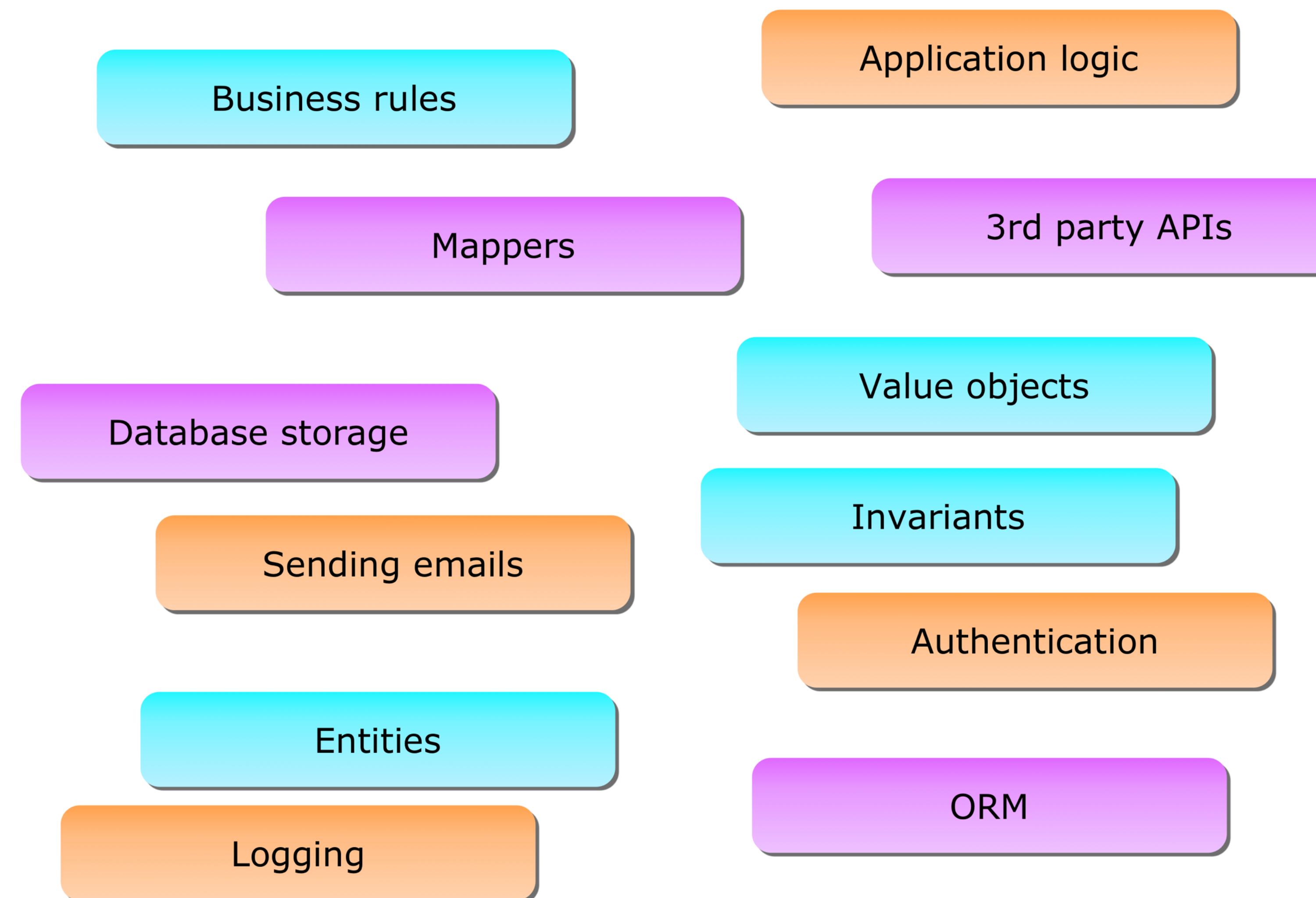


The model is the solution to
a problem

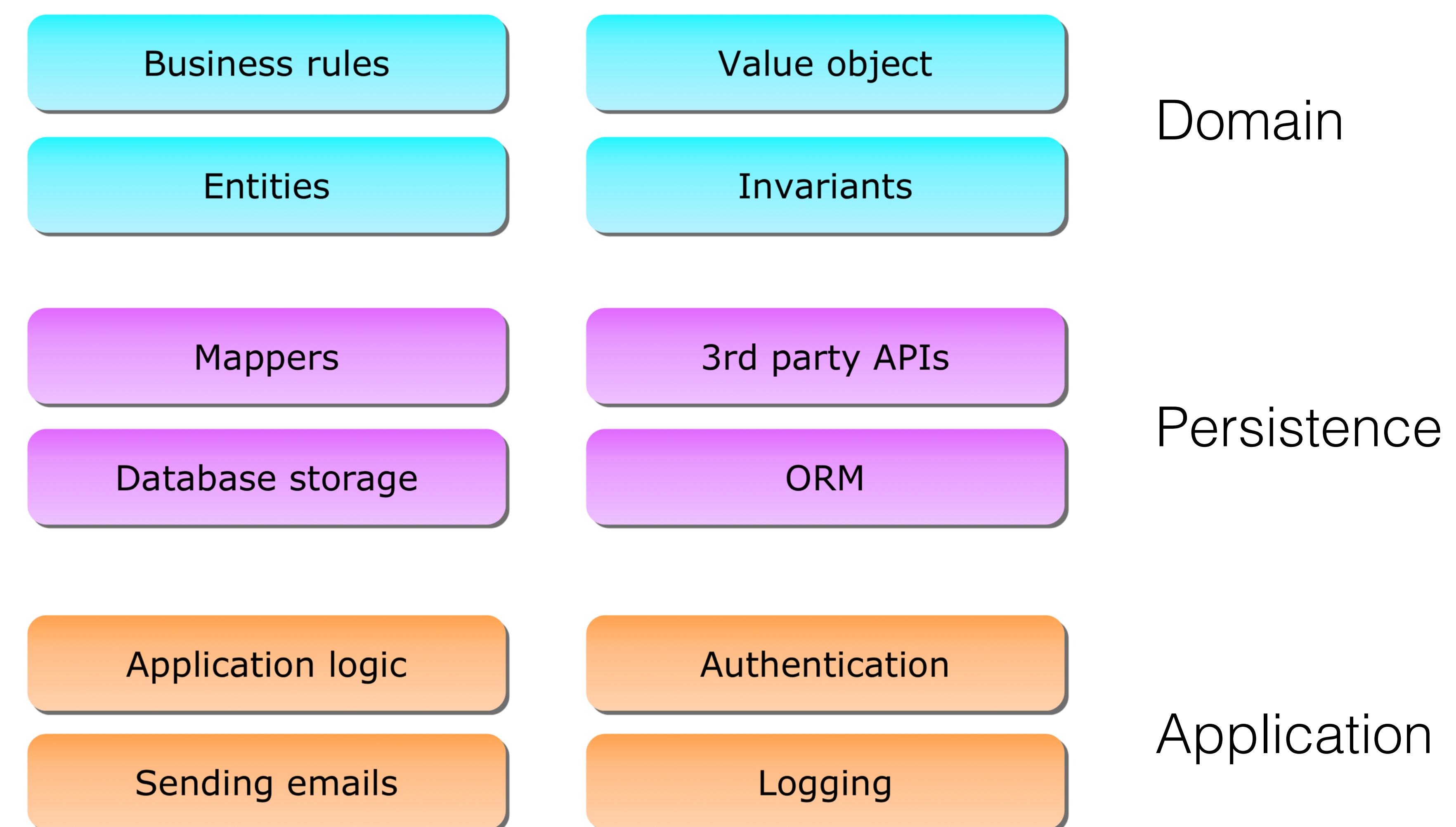
A fantastic solution!



What's in the model?



What's in the model?



The domain

A problem

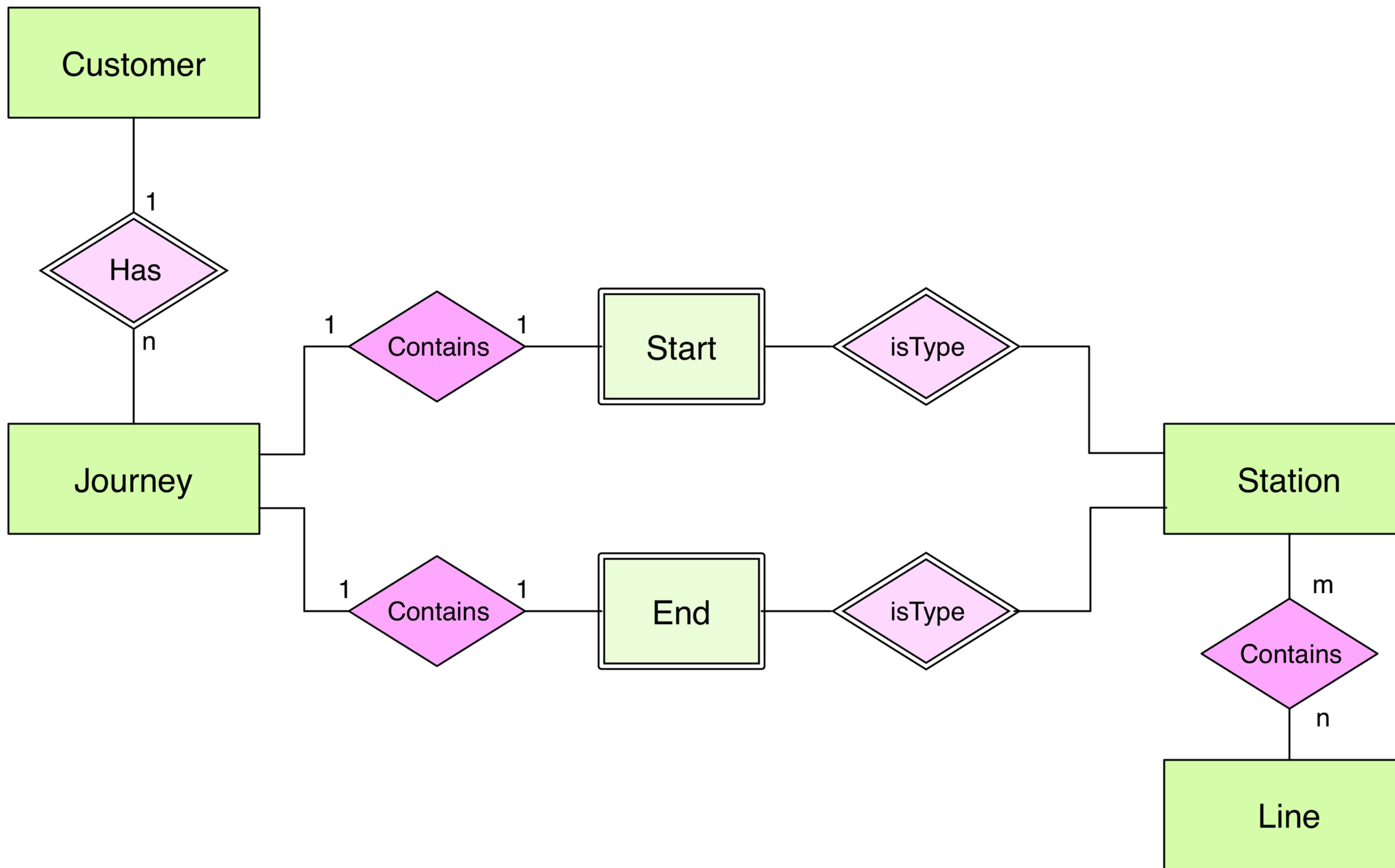
A customer wants to plan a journey between two stations.

How do we model this?

Identify the key objects

- Customer
- Journey
- Station
- Line

Entity–relationship diagram



Entities represent things in the
problem-space

Entity

- Means something to the customer
- An object defined primarily by its identity
- Mutable
- Persisted
- Has a life cycle

Identity

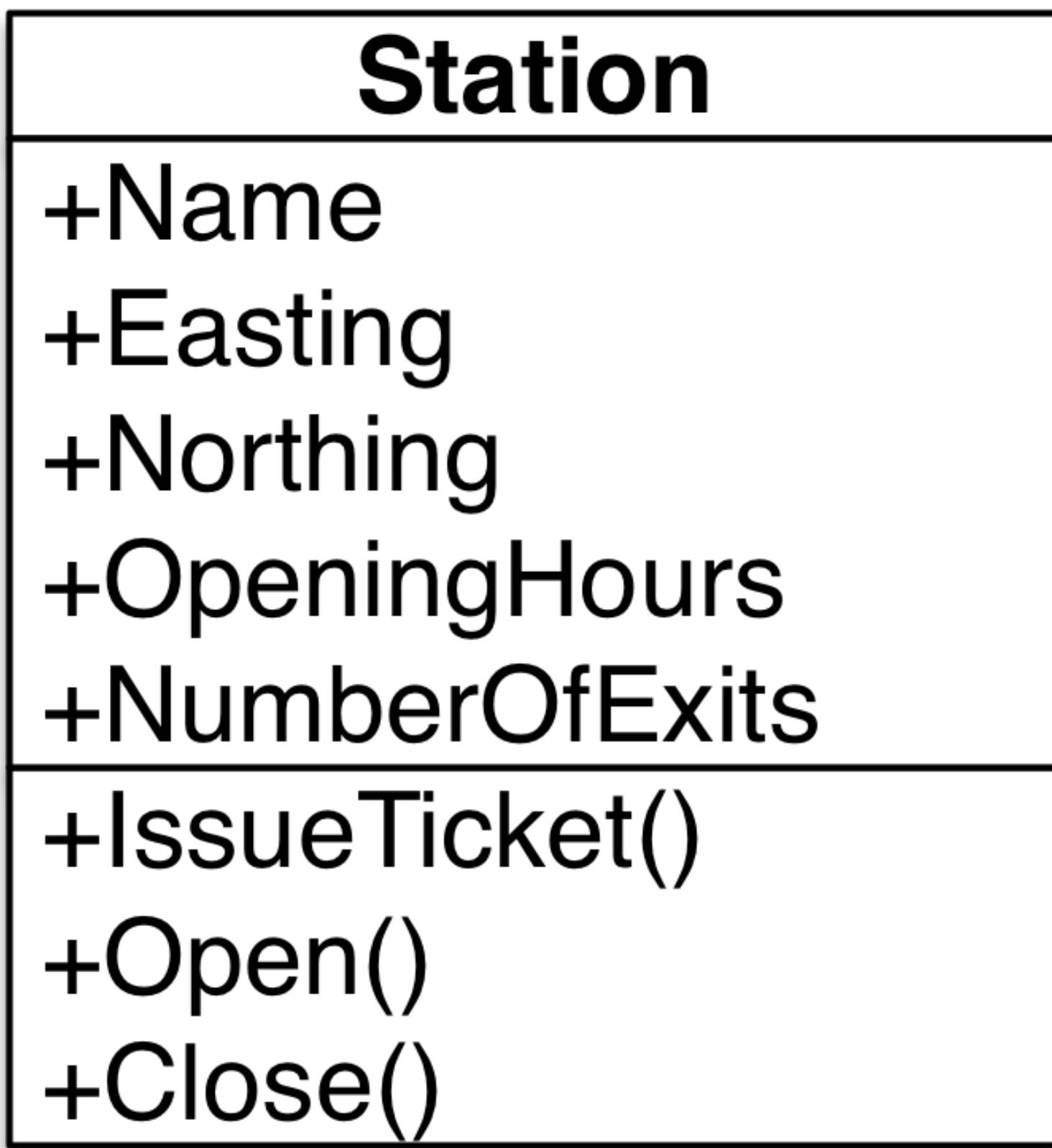
- The identity of an object is what it means to the customer
- Unique within the scope of the domain

For a tube station, this is its *name*, not its database id.

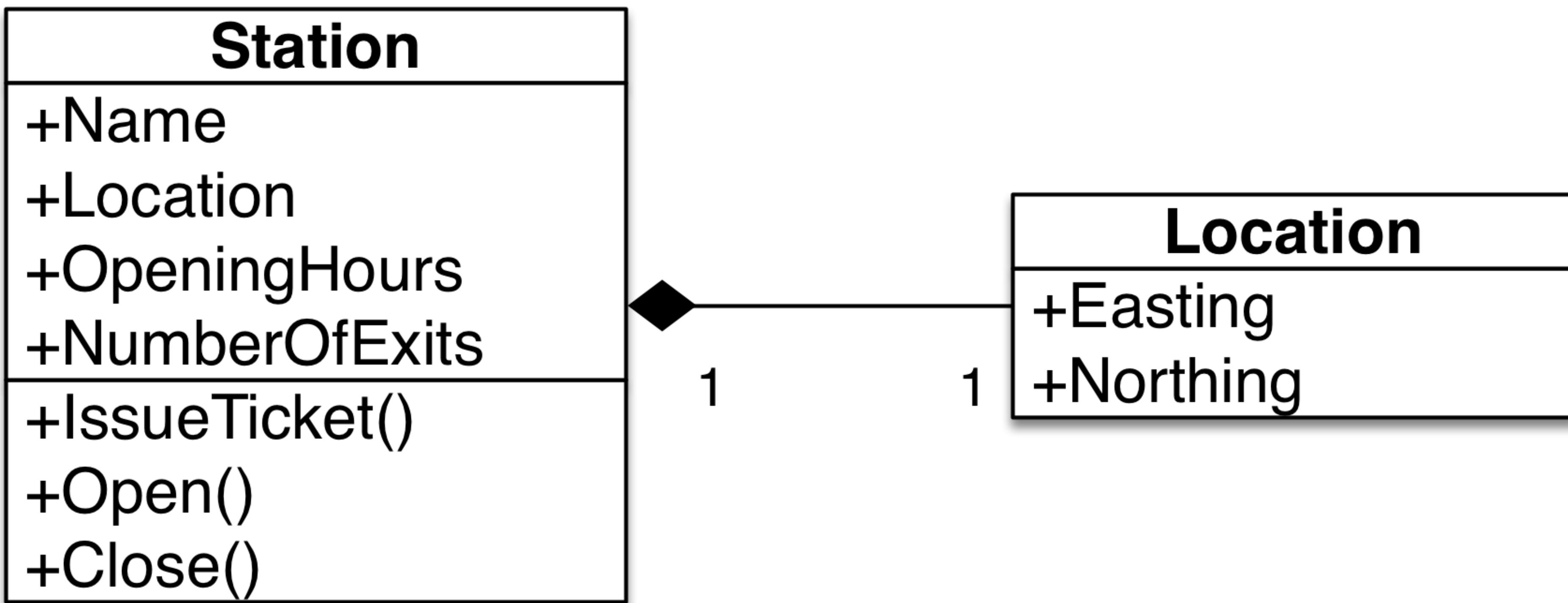
Value objects

- Defined primarily by its attributes
- Immutable
- Simple!
- Do not exist without an entity

A station has a location



A station has a location



Domain services

If a SERVICE were devised to make appropriate debits and credits for a funds transfer, that capability would belong in the domain layer.

–Eric Evans

Domain services

- We map the business processes to services
- Represents behaviour in the domain
- A service does not have internal state
- Usually a point of connection for many objects

Let's look at some code

Some code

```
class Journey {  
    function getStart() {}  
    function setStart(Station $start) {}  
    function getStop() {}  
    function setStop(Station $stop) {}  
    function setRoute() {}  
    function getRoute() {}  
}  
class RoutingService {  
    function route(Station $start, Station $stop) {}  
}
```

Some code

```
class Journey {  
    function getStart() {}  
    function setStart(Station $start) {}  
    function getStop() {}  
    function setStop(Station $stop) {}  
    function setRoute() {}  
    function getRoute() {}  
}  
  
class RoutingService {  
    function route(Station $start, Station $stop) {}  
}
```

Some code

```
class Journey {  
    function getStart() {}  
    function setStart(Station $start) {}  
    function getStop() {}  
    function setStop(Station $stop) {}  
    function setRoute() {}  
    function getRoute() {}  
}  
  
class RoutingService {  
    function route(Station $start, Station $stop) {}  
}
```

Anaemic domain model

You realize that there is hardly any behavior on these objects, making them little more than bags of getters and setters.

Instead there are a set of service objects which capture all the domain logic.

–Martin Fowler

Entity with behaviour

```
class Journey {  
    function getStart() {}  
    function setStart(Station $start) {}  
  
    function getStop() {}  
    function setStop(Station $stop) {}  
  
    function route() {}  
}
```

Entity with behaviour

```
class Journey {  
    function getStart() {}  
    function setStart(Station $start) {}  
  
    function getStop() {}  
    function setStop(Station $stop) {}  
  
    function route() {}  
}
```

Entity with behaviour

```
class Journey {  
    function getStart() {}  
    function setStart(Station $start) {}  
  
    function getStop() {}  
    function setStop(Station $stop) {}  
  
    function route() {}  
}
```

What happens if route() is
complex?

Double dispatch

The entity calls the helper domain service, passing a reference to itself.

```
// Helper service
class JourneyRouter {
    function route(Journey $journey) {}
}

// Journey class
function route() {
    $router = $new JourneyRouter();
    $this->route = $router->route($this);
}
```

Double dispatch

The entity calls the helper domain service, passing a reference to itself.

```
// Helper service
class JourneyRouter {
    function route(Journey $journey) {}
}

// Journey class
function route() {
    $router = $new JourneyRouter();
    $this->route = $router->route($this);
}
```

Double dispatch

The entity calls the helper domain service, passing a reference to itself.

```
// Helper service
class JourneyRouter {
    function route(Journey $journey) {}
}

// Journey class
function route() {
    $router = $new JourneyRouter();
    $this->route = $router->route($this);
}
```

Persistence

Persistence options

A simple domain model can use ActiveRecord/TDG;
a complex one will require mapping.

I don't really care what you choose!

I lied.

Don't use ActiveRecord!

It integrates the database code
into your domain model

Table Data Gateway

- Operates on a single database table
- Contains all the SQL for accessing the table
- Doesn't know anything about the entity.
- Simple to implement

Table Data Gateway

```
class JourneyGateway
{
    function __construct($dsn, $username, $password) {}

    function find($id) {}
    function findForStartingStation($stationId) {}

    function insert($startId, $stopId) {}
    function update($id, $startId, $stopId)
}
```

Table Data Gateway

```
class JourneyGateway
{
    function __construct($dsn, $username, $password) {}

    function find($id) {}
    function findForStartingStation($stationId) {}

    function insert($startId, $stopId) {}
    function update($id, $startId, $stopId)
}
```

Table Data Gateway

```
class JourneyGateway
{
    function __construct($dsn, $username, $password) {}

    function find($id) {}
    function findForStartingStation($stationId) {}

    function insert($startId, $stopId) {}
    function update($id, $startId, $stopId) {}

}
```

Table Data Gateway

```
class JourneyGateway
{
    function __construct($dsn, $username, $password) {}

    function find($id) {}
    function findForStartingStation($stationId) {}

    function insert($startId, $stopId) {}
    function update($id, $startId, $stopId)
}
```

Data Mapper

- Class to transfer data from objects to the database and back.
- Entity aware
- Isolates the domain model from the database
- Not limited to a single table

Data Mapper

```
class JourneyMapper
{
    function __construct($dsn, $username, $password) {}

    function find($id) {}
    function findForStartingStation($stationId) {}

    public function save(Journey $journey) {}
}
```

Data Mapper

```
class JourneyMapper
{
    function __construct($dsn, $username, $password) {}

    function find($id) {}
    function findForStartingStation($stationId) {}

    public function save(Journey $journey) {}
}
```

Data Mapper

```
class JourneyMapper
{
    function __construct($dsn, $username, $password) {}

    function find($id) {}
    function findForStartingStation($stationId) {}

    public function save(Journey $journey) {}

}
```

Data Mapper

```
class JourneyMapper
{
    function __construct($dsn, $username, $password) {}

    function find($id) {}
    function findForStartingStation($stationId) {}

    public function save(Journey $journey) {}
}
```

Increased scope: ORM

Persistence layer is more complicated:

- Identity map to hold loaded objects
- Storage of entire object graphs to the database
- Unit of Work to track changed objects for saving

If you need this, then use a pre-written ORM library!

Web services

- Still within your persistence layer
- Data mappers work really well

Integrating our model into the
application

The service layer*

It does not contain business rules or knowledge, but only coordinates tasks and delegates work to collaborations of domain objects in the next layer down.

–Eric Evans

* (Known as application layer in DDD)

Service layer

We can sub-divide:

- Application services
- Infrastructural services

Application services

If the banking application can convert and export our transactions into a spreadsheet file for us to analyze, this is an application SERVICE.

–Eric Evans

Infrastructural services

A bank might have an application that sends out an email to a customer when an account balance falls below a specific threshold. The interface that encapsulates the email system, and perhaps alternate means of notification, is a SERVICE in the infrastructure layer.

–Eric Evans

Application service

```
class JourneyService {  
    function createJourney($customer, $start, $stop)  
    {  
        $journey = $customer->createJourney($start, $stop);  
        $journey->route();  
  
        $this->entityManager->flush();  
        $this->mailer->newJourneyNotification($journey);  
        $this->auditor->log('newJourney', $journey);  
    }  
}
```

Application service

```
class JourneyService {  
    function createJourney($customer, $start, $stop)  
    {  
        $journey = $customer->createJourney($start, $stop);  
        $journey->route();  
  
        $this->entityManager->flush();  
        $this->mailer->newJourneyNotification($journey);  
        $this->auditor->log('newJourney', $journey);  
    }  
}
```

Application service

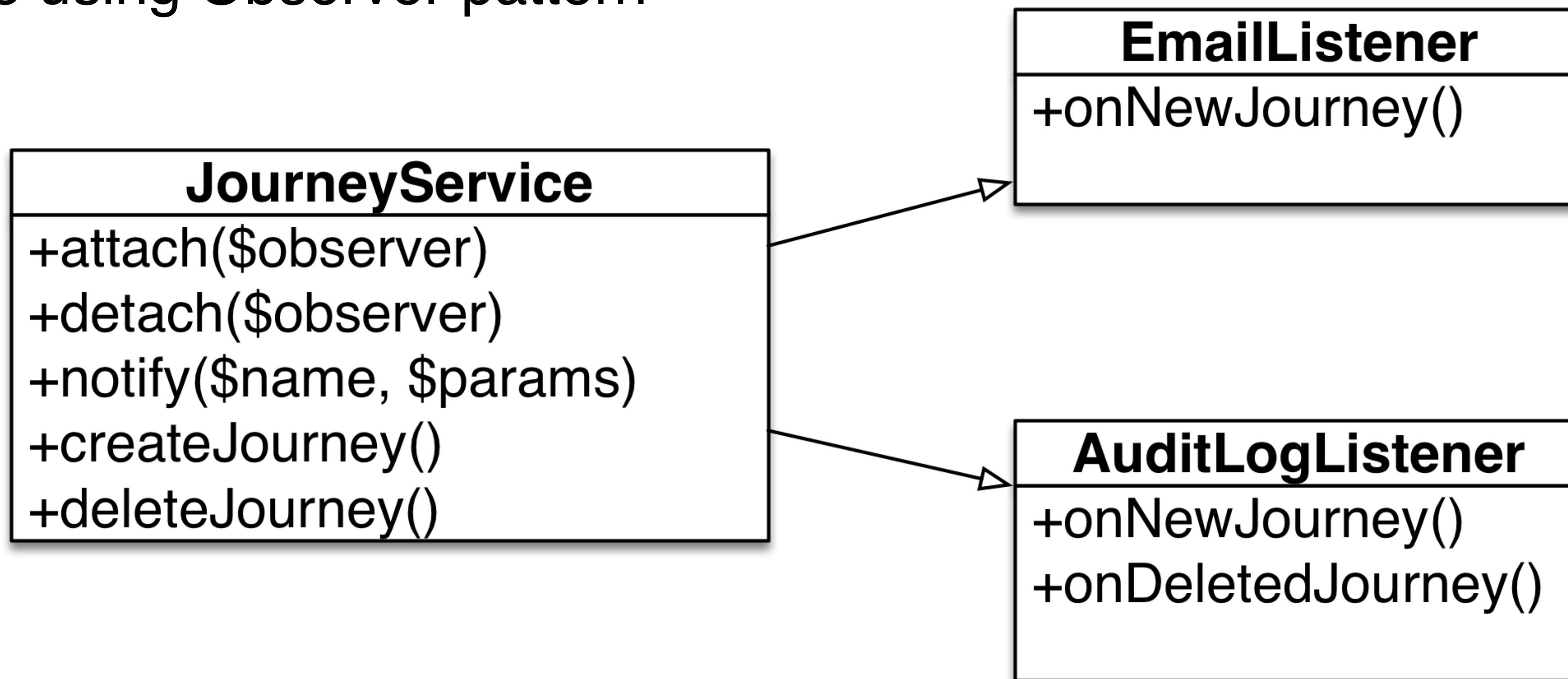
```
class JourneyService {  
    function createJourney($customer, $start, $stop)  
    {  
        $journey = $customer->createJourney($start, $stop);  
        $journey->route();  
  
        $this->entityManager->flush();  
        $this->mailer->newJourneyNotification($journey);  
        $this->auditor->log('newJourney', $journey);  
    }  
}
```

Application service

```
class JourneyService {  
    function createJourney($customer, $start, $stop)  
    {  
        $journey = $customer->createJourney($start, $stop);  
        $journey->route();  
  
        $this->entityManager->flush();  
        $this->mailer->newJourneyNotification($journey);  
        $this->auditor->log('newJourney', $journey);  
    }  
}
```

Beware the fat service

Decouple using Observer pattern



CQRS: Separating reading
and writing

CQRS

Command Query Responsibility Segregation.

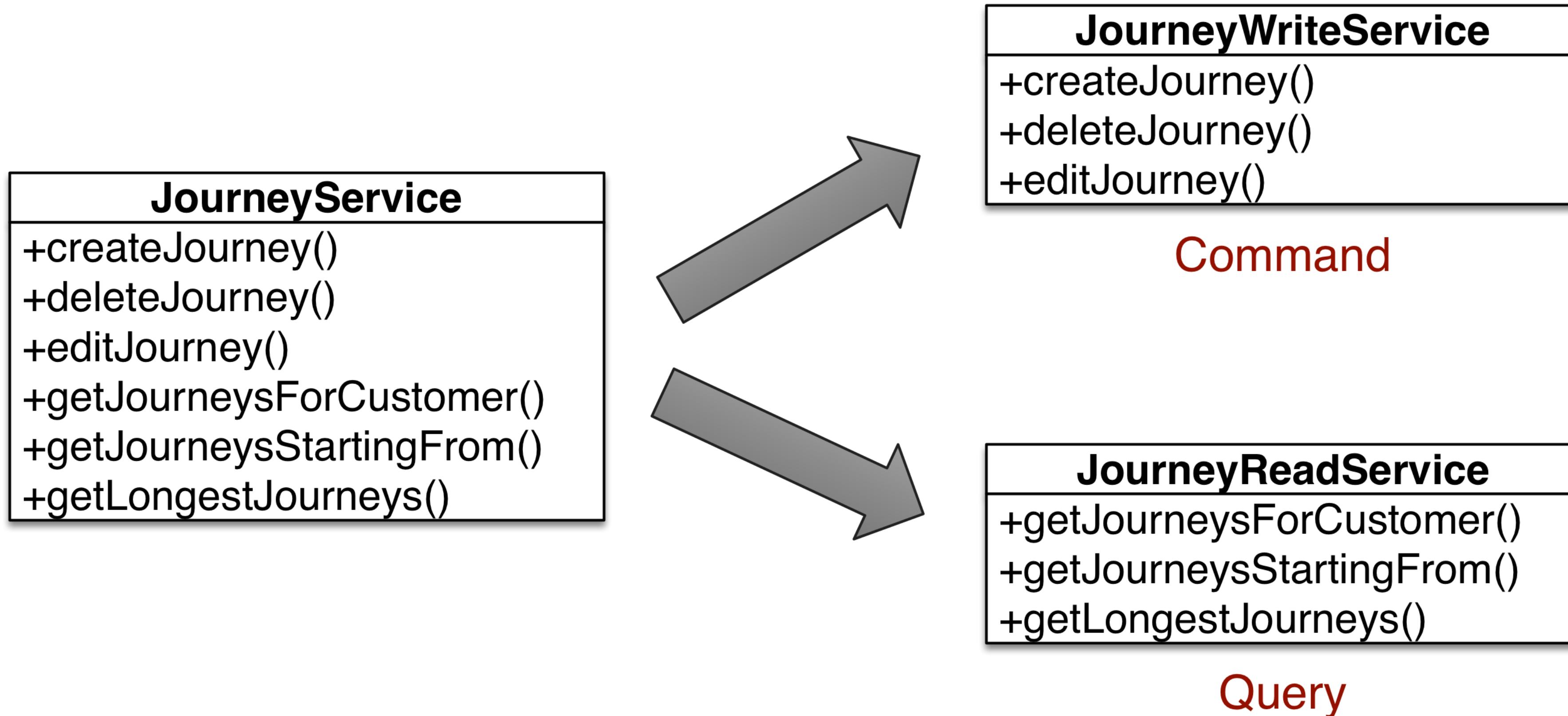
- *Commands* change data
- *Queries* read data

Most useful when:

- Separate hardware
- Optimise performance

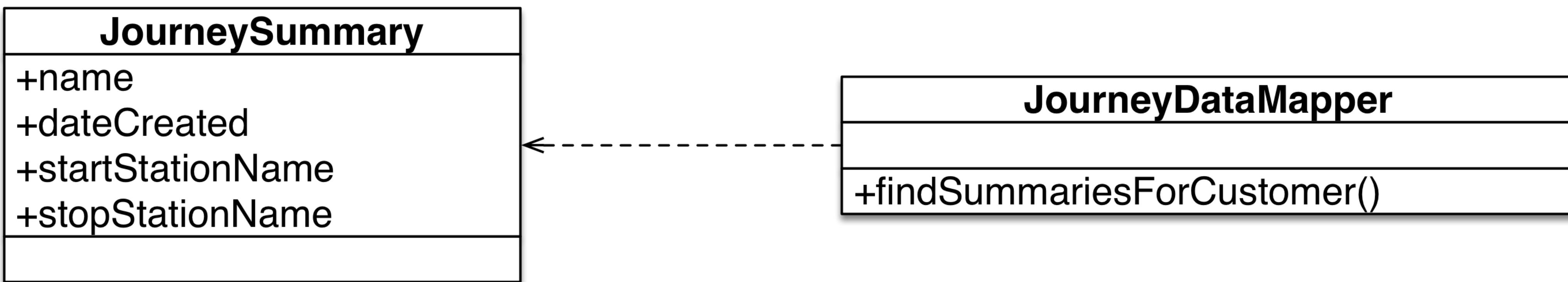
CQRS at its most basic

Two services where there was one



CQRS: useful case

Summary object for a read-only list



Final point

*The success of a design is not necessarily marked by its longevity.
It is the nature of software to change.*

–Eric Evans

Questions?

Thank you!

Feedback: <https://bit.ly/creatingmodels>

Rob Allen - <http://akrabat.com> - @akrabat