Event-Driven Architectures in Elixir

Maciej Kaszubowski

ElixirConfEU 2018

mkaszubowski94

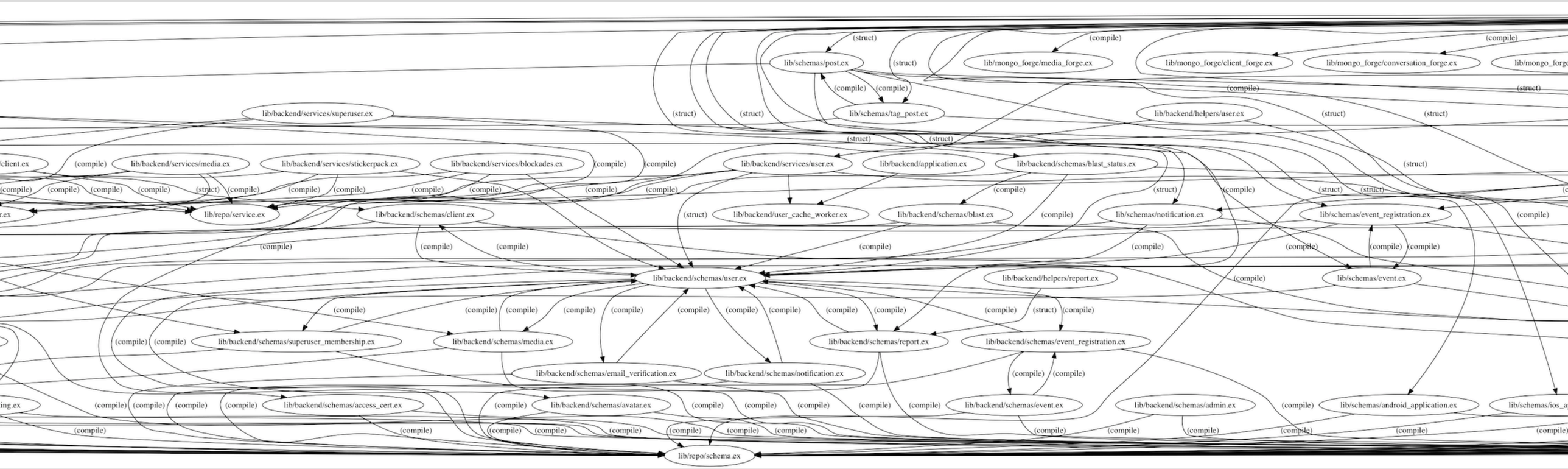
Event-Driven Architectures

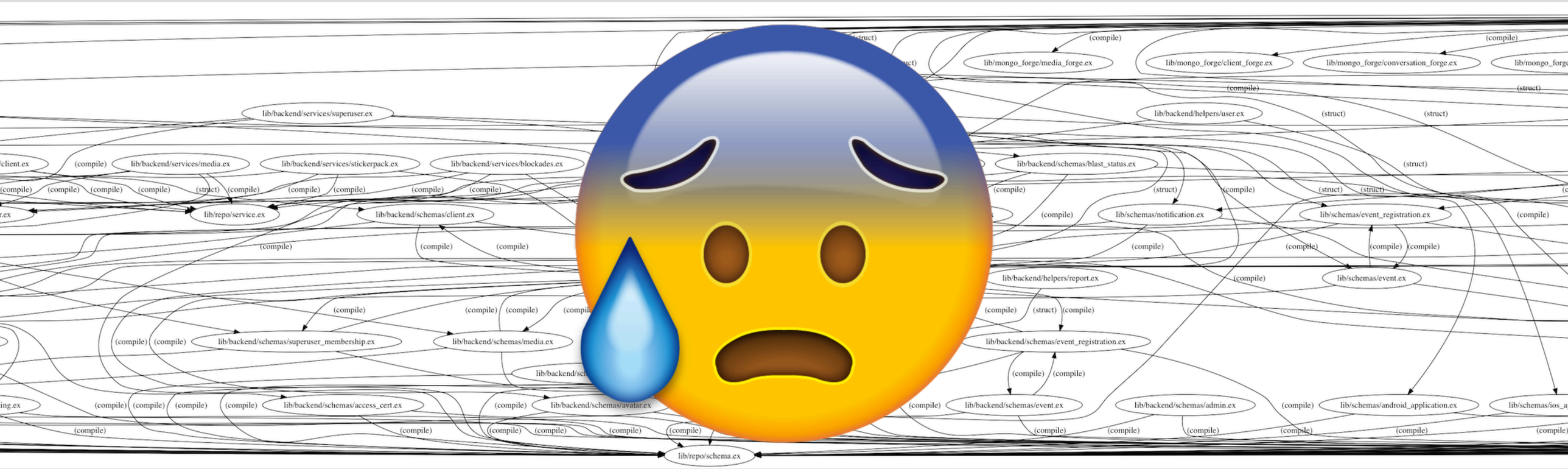
- Event Sourcing / CQRS
- Microservices
- Kafka, RabbitMQ, …
- Message passing / OTP

This is not the point

AppUnite

2+ years of Elixir in production


lib/schemas/post.ex
lib/mongo_forge/media_forge.ex
lib/mongo_forge/client_forge.ex
lib/mongo_forge/conversation_forge.ex
lib/schemas/tag_post.ex
lib/backend/services/superuser.ex
lib/backend/helpers/user.ex
lib/backend/services/media.ex
lib/backend/services/stickerpack.ex
lib/backend/services/blockades.ex
lib/backend/services/user.ex
lib/backend/application.ex
lib/backend/schemas/blast_status.ex
lib/repo/service.ex
lib/backend/schemas/client.ex
lib/backend/user_cache_worker.ex
lib/backend/schemas/blast.ex
lib/schemas/notification.ex
lib/schemas/event_registration.ex
lib/backend/schemas/user.ex
lib/backend/helpers/report.ex
lib/schemas/event.ex
lib/backend/schemas/superuser_membership.ex
lib/backend/schemas/media.ex
lib/backend/schemas/report.ex
lib/backend/schemas/event_registration.ex
lib/backend/schemas/email_verification.ex
lib/backend/schemas/notification.ex
lib/backend/schemas/access_cert.ex
lib/backend/schemas/avatar.ex
lib/backend/schemas/event.ex
lib/backend/schemas/admin.ex
lib/schemas/android_application.ex
lib/repo/schema.ex
(compile)
(struct)


lib/backend/services/superuser.ex
lib/backend/services/media.ex
lib/backend/services/stickerpack.ex
lib/backend/services/blockades.ex
lib/repo/service.ex
lib/backend/schemas/client.ex
lib/backend/schemas/superuser_membership.ex
lib/backend/schemas/media.ex
lib/backend/schemas/access_cert.ex
lib/backend/schemas/avatar.ex
lib/repo/schema.ex
lib/mongo_forge/media_forge.ex
lib/mongo_forge/client_forge.ex
lib/mongo_forge/conversation_forge.ex
lib/backend/helpers/user.ex
lib/backend/schemas/blast_status.ex
lib/schemas/notification.ex
lib/schemas/event_registration.ex
lib/backend/helpers/report.ex
lib/schemas/event.ex
lib/backend/schemas/event_registration.ex
lib/backend/schemas/event.ex
lib/backend/schemas/admin.ex
lib/schemas/android_application.ex
(compile)
(struct)

Building software is a complex task

Changing software is even harder

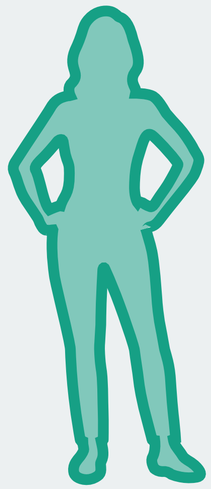
So, this is not a talk about:

- Event Sourcing / CQRS
- Microservices
- Kafka, RabbitMQ, …
- Message passing / OTP

This is a talk about building small, **independent** things

Our project

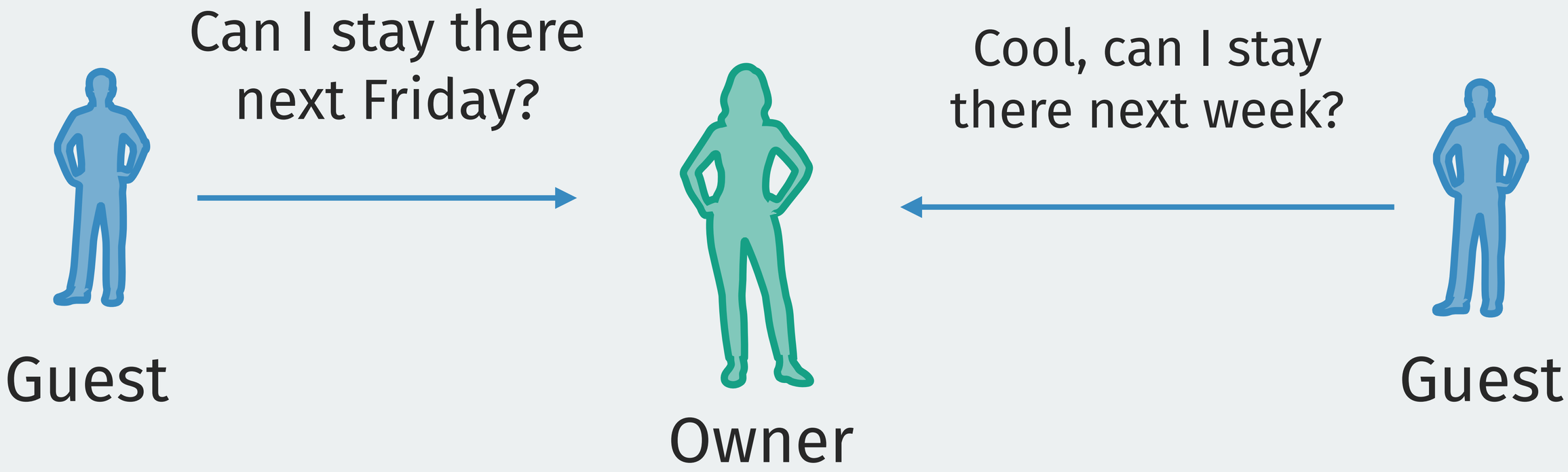
Guest

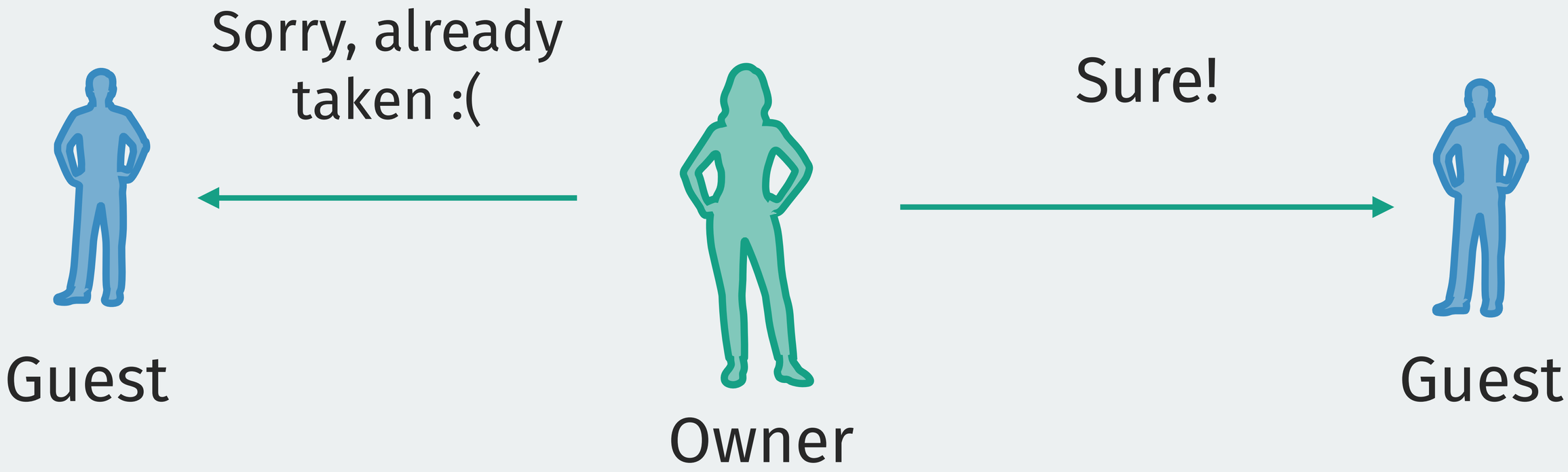


Hey, I have a free apartment!

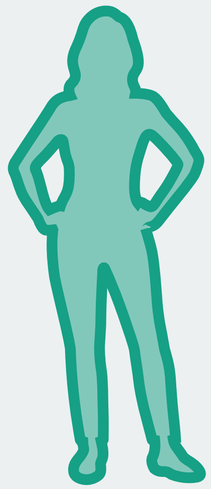
Owner

Guest


Can I stay there next Friday?
Cool, can I stay there next week?
Guest
Owner
Guest


Sorry, already taken :(
Sure!
Guest
Owner
Guest

Guest



Owner

$$$

Guest

Let's build this

```elixir
defmodule AirBnb.ApartmentsContext do
  def create_apartment(params) do
    changeset = Apartment.changeset(%Apartment{}, params)

    case Repo.insert(changeset) do
      {:ok, apartment} -> {:ok, apartmet}
      {:error, changeset} -> {:error, changeset}
    end
  end

  # ...

end
```

```elixir
defmodule AirBnb.ApartmentsContext do

  # ...


  def fetch_apartments(query_params) do
    Apartment
    |> query_by_location(query_params)
    |> query_by_price(query_params)
    |> ...
    |> Repo.all()
  end
end
```

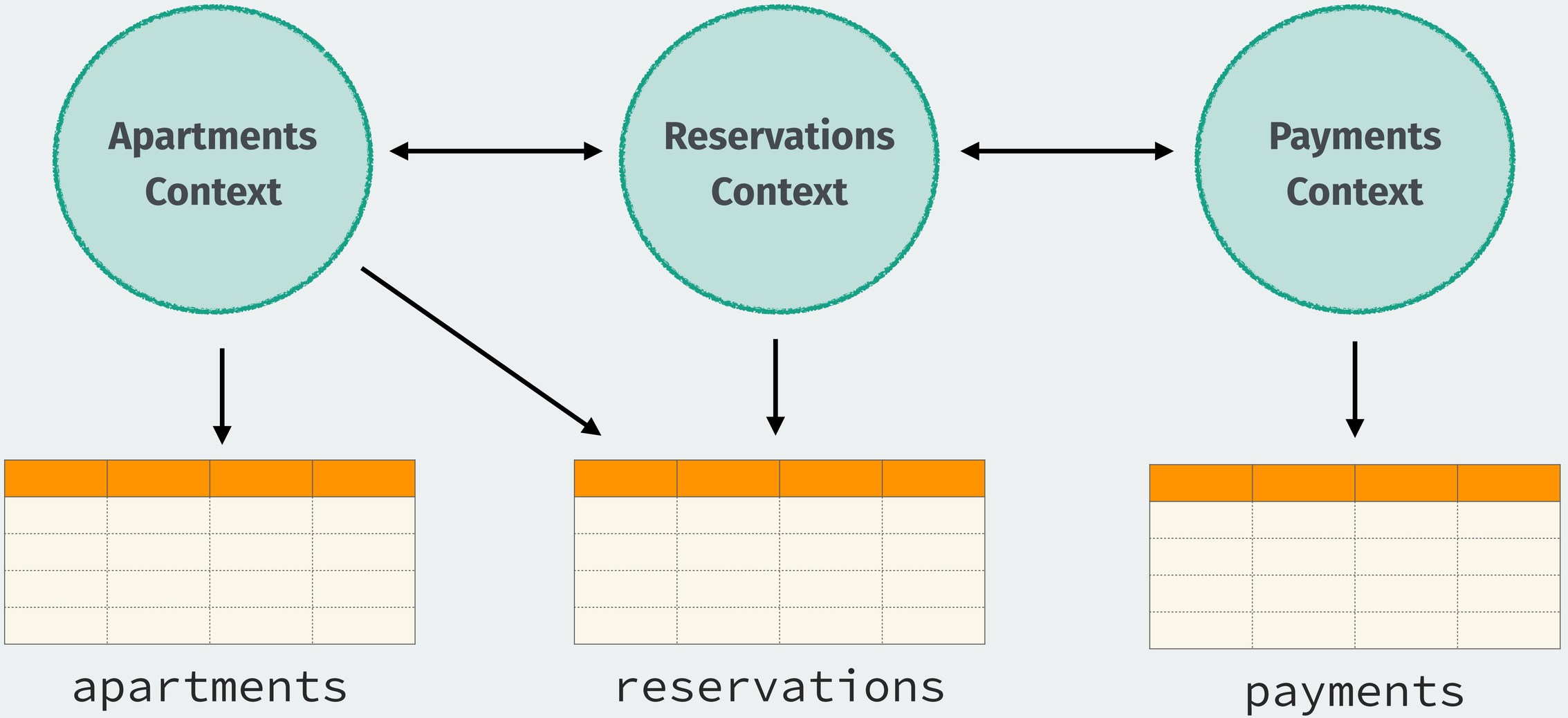
```elixir
defmodule Airbnb.BookingContext do
  def create_reservation(apartment_id, guest_id,
                         start_date, end_date) do
    params = %{
      apartment_id: apartment_id, guest_id: guest_id,
      start_date: start_date, end_date: end_date
    }

    %Reservation{}
    |> Reservation.changeset(params)
    |> Repo.insert()
  end
```

```elixir
defmodule Airbnb.BookingContext do

  def reject_reservation(owner_id, reservation_id) do
    reservation =
      Repo.get(Reservation, reservation_id)
      |> Repo.reload(:apartment)

    if reservation.apartment.owner_id == owner_id do
      Repo.delete(reservation)
    end
  end
```

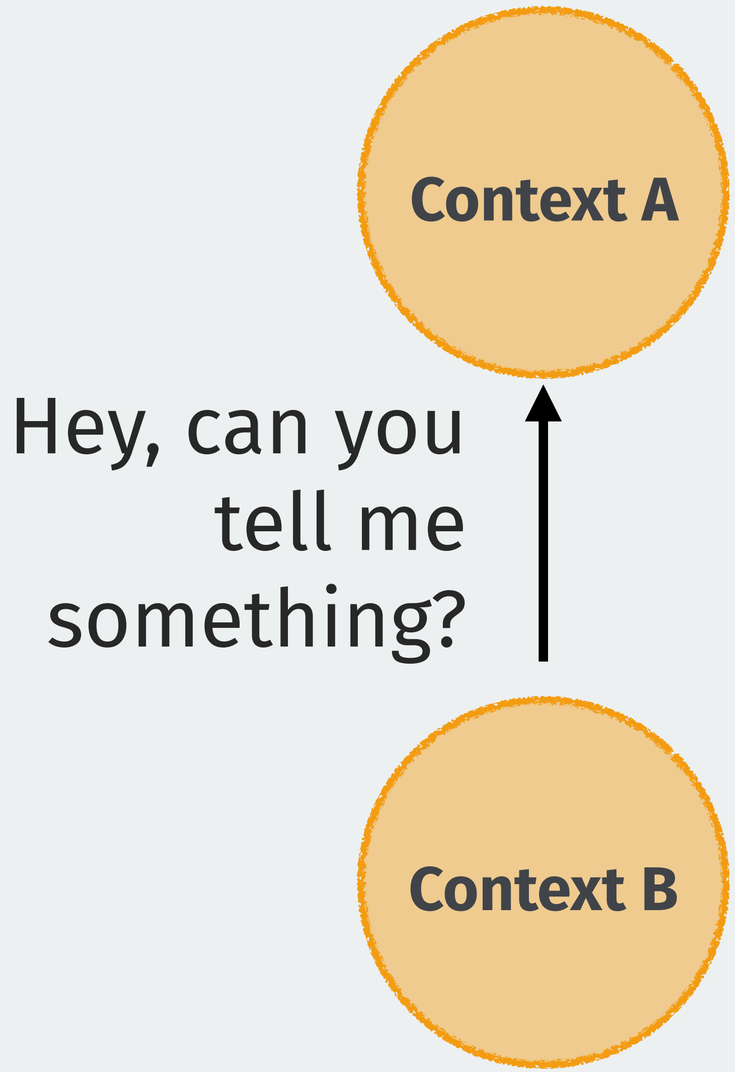
```elixir
defmodule Airbnb.BookingContext do

  def accept_reservation(owner_id, reservation_id) do
    reservation =
      Repo.get(Reservation, reservation_id)
      |> Repo.reload(:apartment)

    if reservation.apartment.owner_id == owner_id do
      reservation
      |> Reservation.changeset(%{accepted: true})
      |> Repo.update()
    end
  end
```

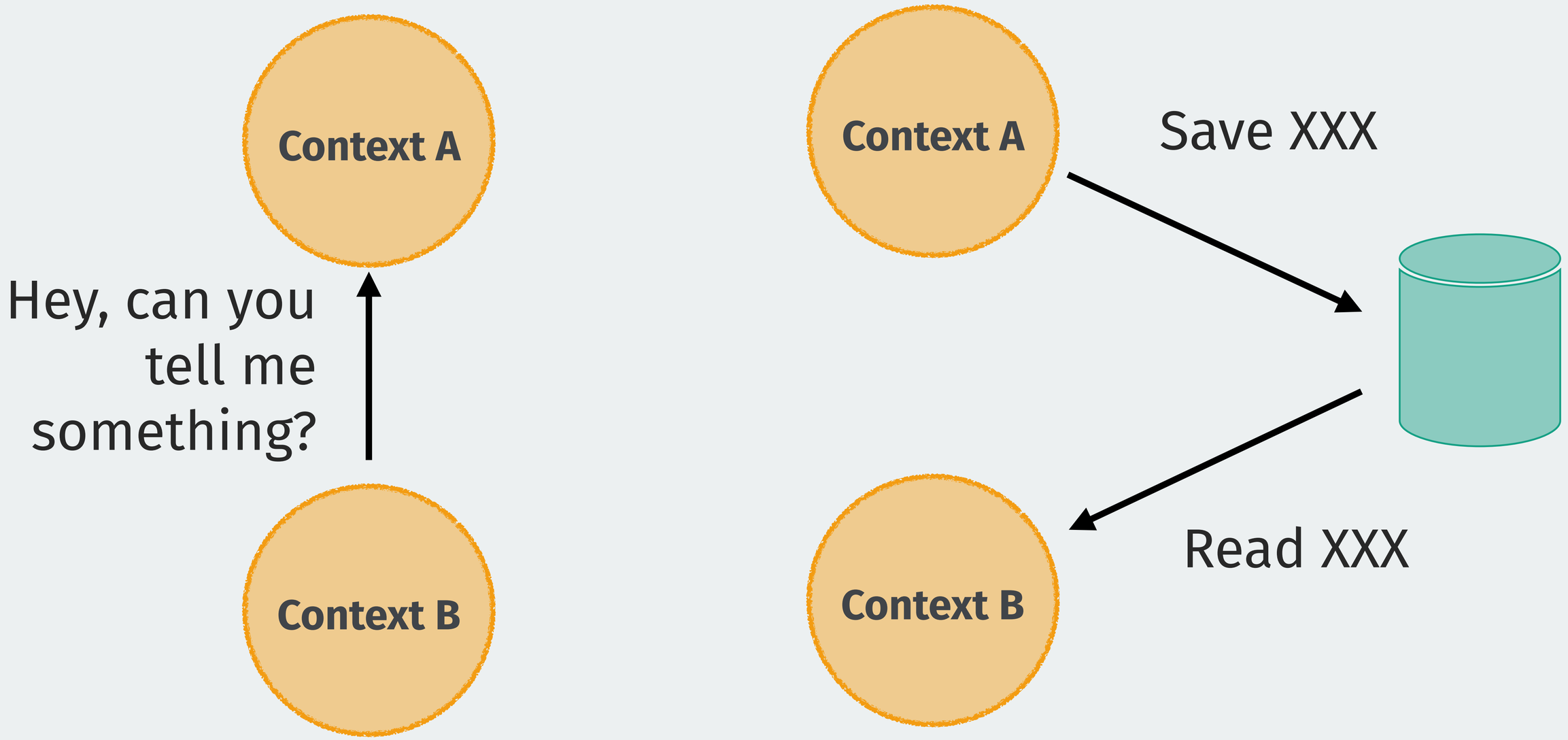
Accepting reservation

- Notify the guest
- Charge guest’s credit card
- Mark the apartment as taken

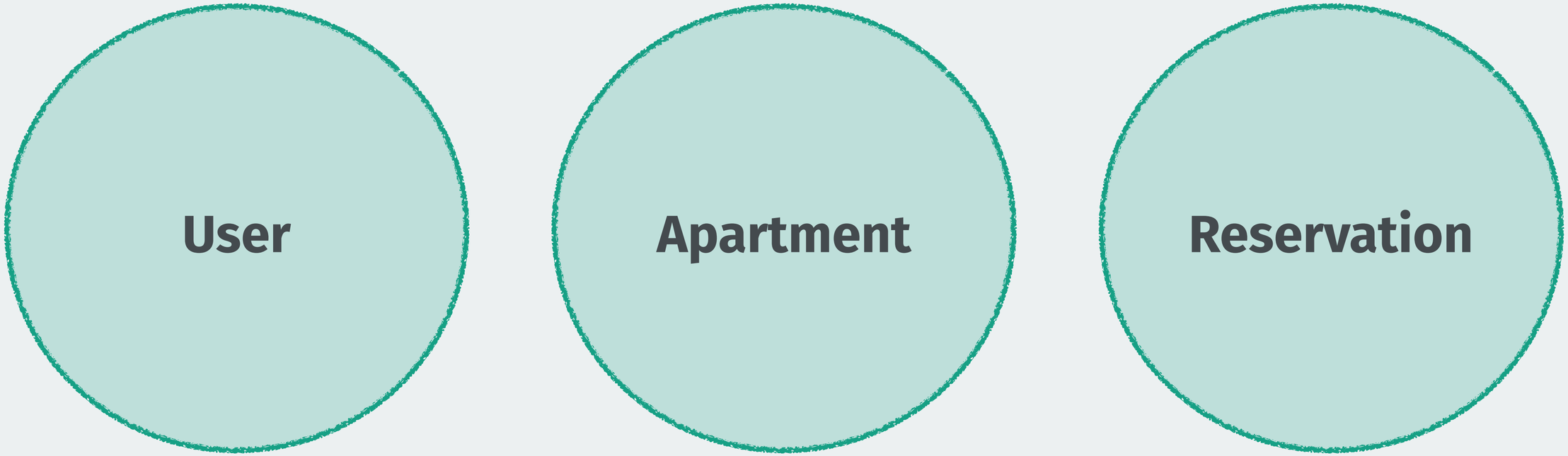

Apartments Context
Reservations Context
Payments Context
apartments
reservations
payments

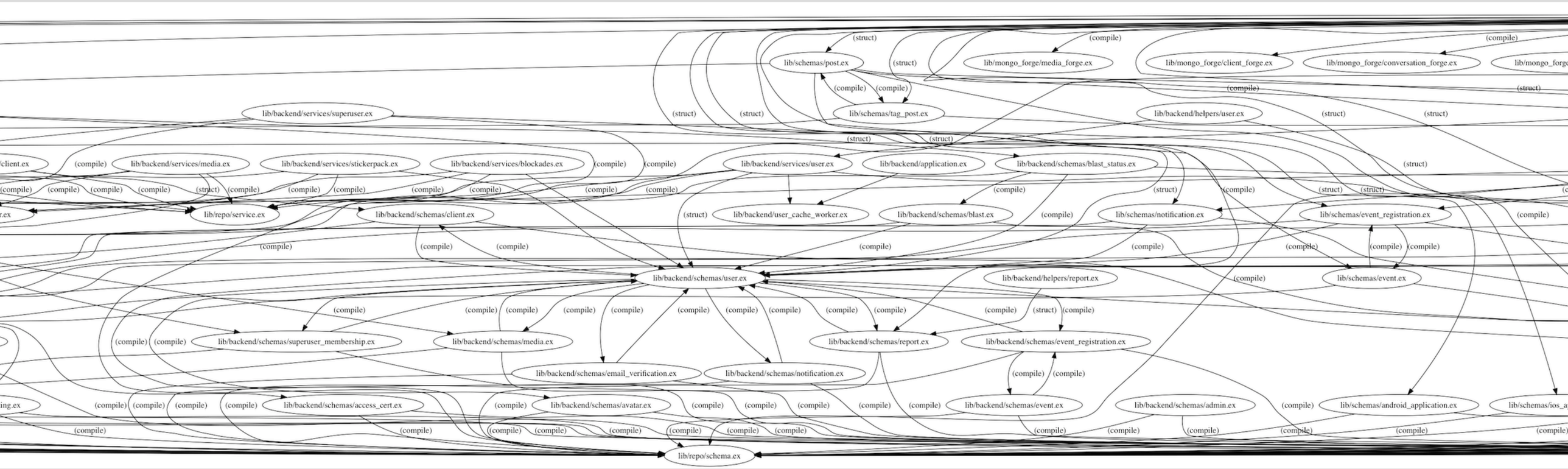
Coupling

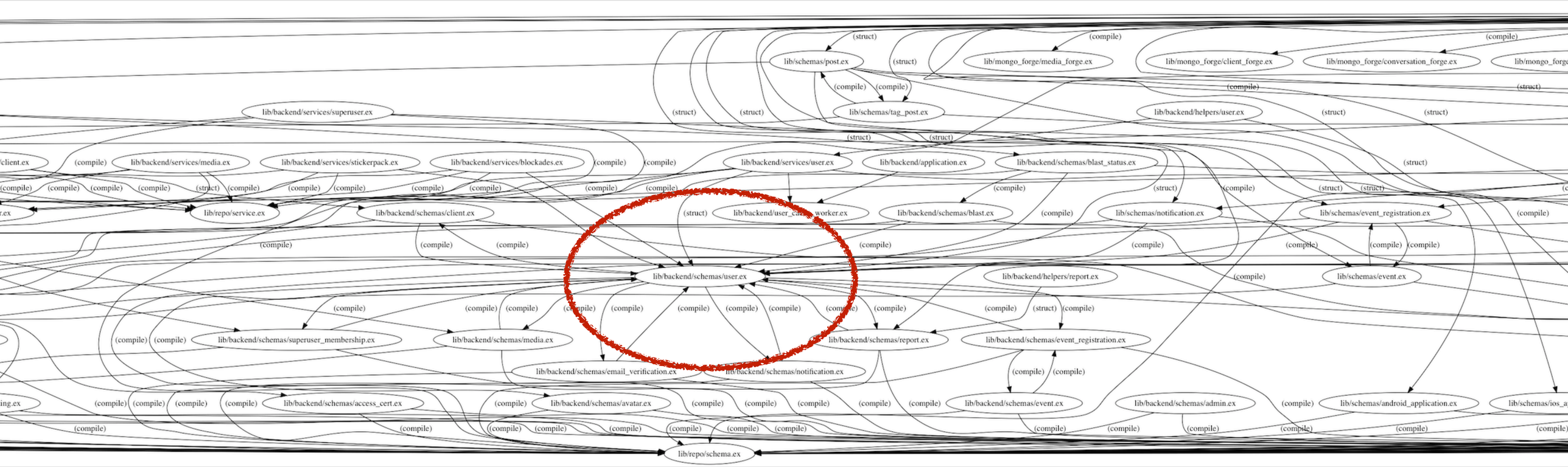

Context A
Hey, can you tell me something?
Context B


Context A
Hey, can you tell me something?
Context B
Context A
Save XXX
Read XXX
Context B

Few entities,
A lot of use cases


User
Apartment
Reservation


lib/schemas/post.ex
lib/mongo_forge/media_forge.ex
lib/mongo_forge/client_forge.ex
lib/mongo_forge/conversation_forge.ex
lib/schemas/tag_post.ex
lib/backend/services/superuser.ex
lib/backend/helpers/user.ex
lib/backend/services/media.ex
lib/backend/services/stickerpack.ex
lib/backend/services/blockades.ex
lib/backend/services/user.ex
lib/backend/application.ex
lib/backend/schemas/blast_status.ex
lib/repo/service.ex
lib/backend/schemas/client.ex
lib/backend/user_cache_worker.ex
lib/backend/schemas/blast.ex
lib/schemas/notification.ex
lib/schemas/event_registration.ex
lib/backend/schemas/user.ex
lib/backend/helpers/report.ex
lib/schemas/event.ex
lib/backend/schemas/superuser_membership.ex
lib/backend/schemas/media.ex
lib/backend/schemas/report.ex
lib/backend/schemas/event_registration.ex
lib/backend/schemas/email_verification.ex
lib/backend/schemas/notification.ex
lib/backend/schemas/access_cert.ex
lib/backend/schemas/avatar.ex
lib/backend/schemas/event.ex
lib/backend/schemas/admin.ex
lib/schemas/android_application.ex
lib/repo/schema.ex
(compile)
(struct)


lib/schemas/post.ex
lib/mongo_forge/media_forge.ex
lib/mongo_forge/client_forge.ex
lib/mongo_forge/conversation_forge.ex
lib/backend/services/superuser.ex
lib/schemas/tag_post.ex
lib/backend/helpers/user.ex
lib/backend/services/media.ex
lib/backend/services/stickerpack.ex
lib/backend/services/blockades.ex
lib/backend/services/user.ex
lib/backend/application.ex
lib/backend/schemas/blast_status.ex
lib/repo/service.ex
lib/backend/schemas/client.ex
lib/backend/schemas/blast.ex
lib/schemas/notification.ex
lib/schemas/event_registration.ex
lib/backend/schemas/user.ex
lib/backend/helpers/report.ex
lib/schemas/event.ex
lib/backend/schemas/superuser_membership.ex
lib/backend/schemas/media.ex
lib/backend/schemas/report.ex
lib/backend/schemas/event_registration.ex
lib/backend/schemas/email_verification.ex
lib/backend/schemas/notification.ex
lib/backend/schemas/access_cert.ex
lib/backend/schemas/avatar.ex
lib/backend/schemas/event.ex
lib/backend/schemas/admin.ex
lib/schemas/android_application.ex
lib/repo/schema.ex
(compile)
(struct)

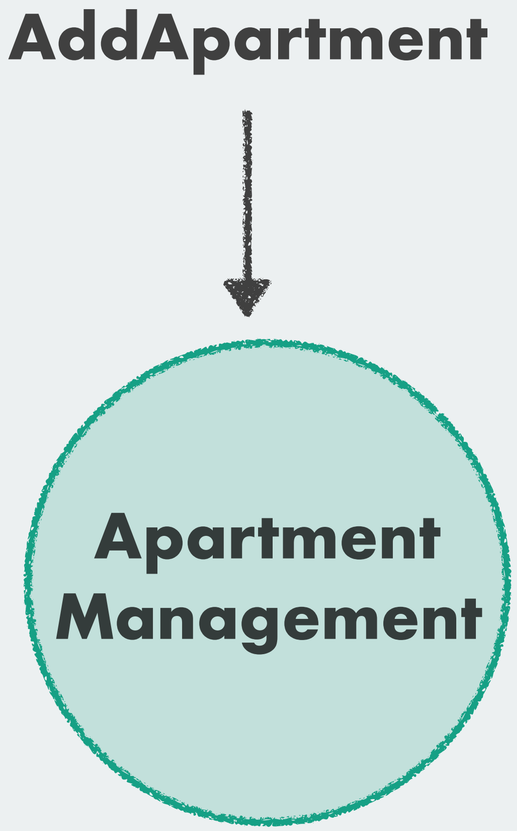
Let's take a step back

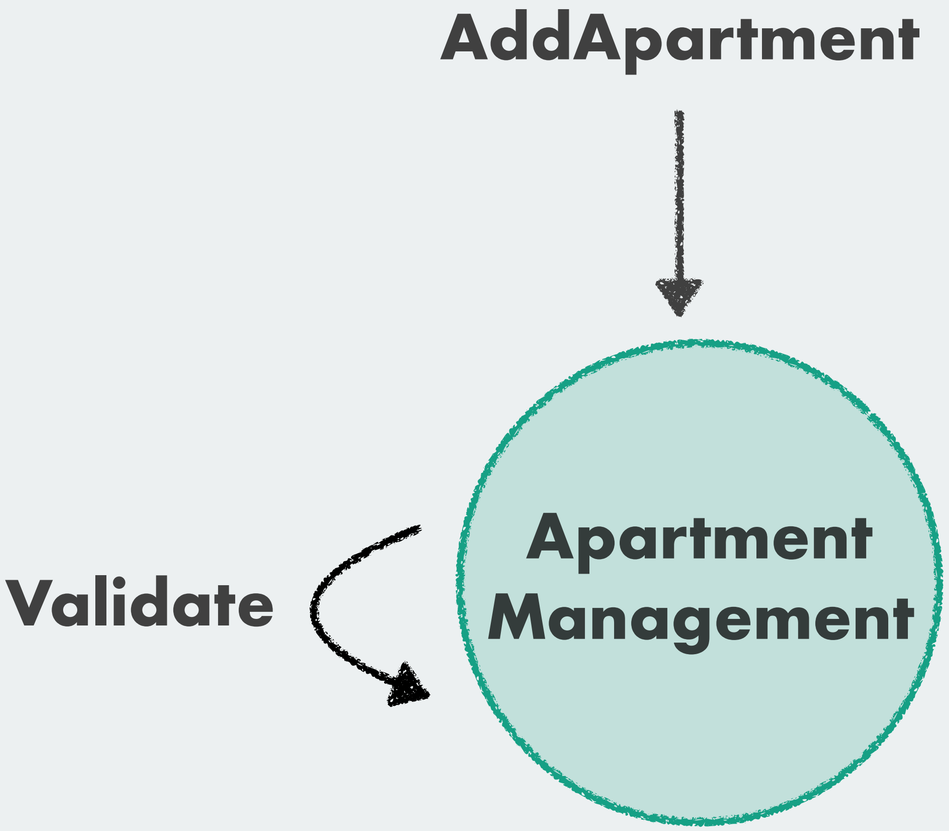
What's really important?

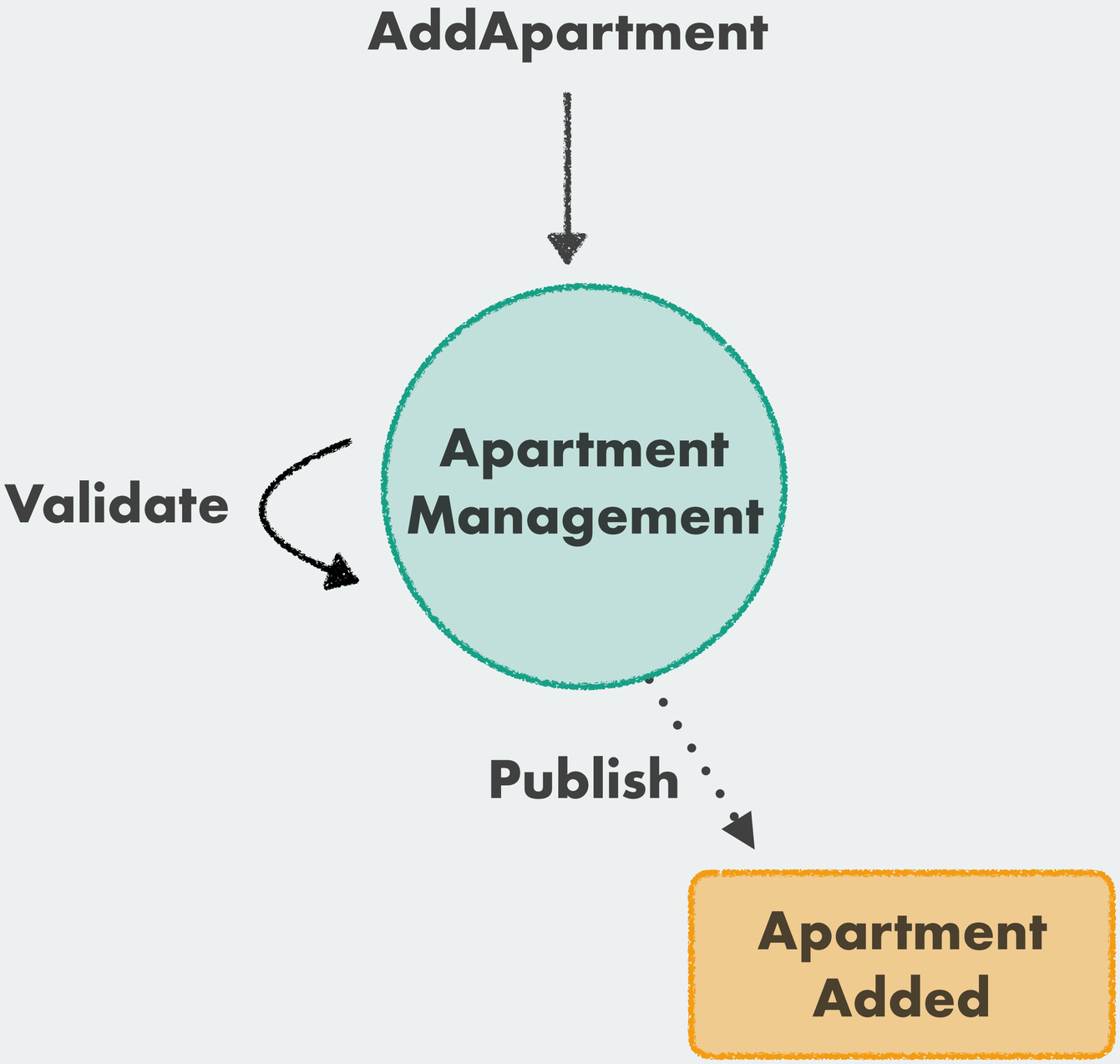
Start with use cases

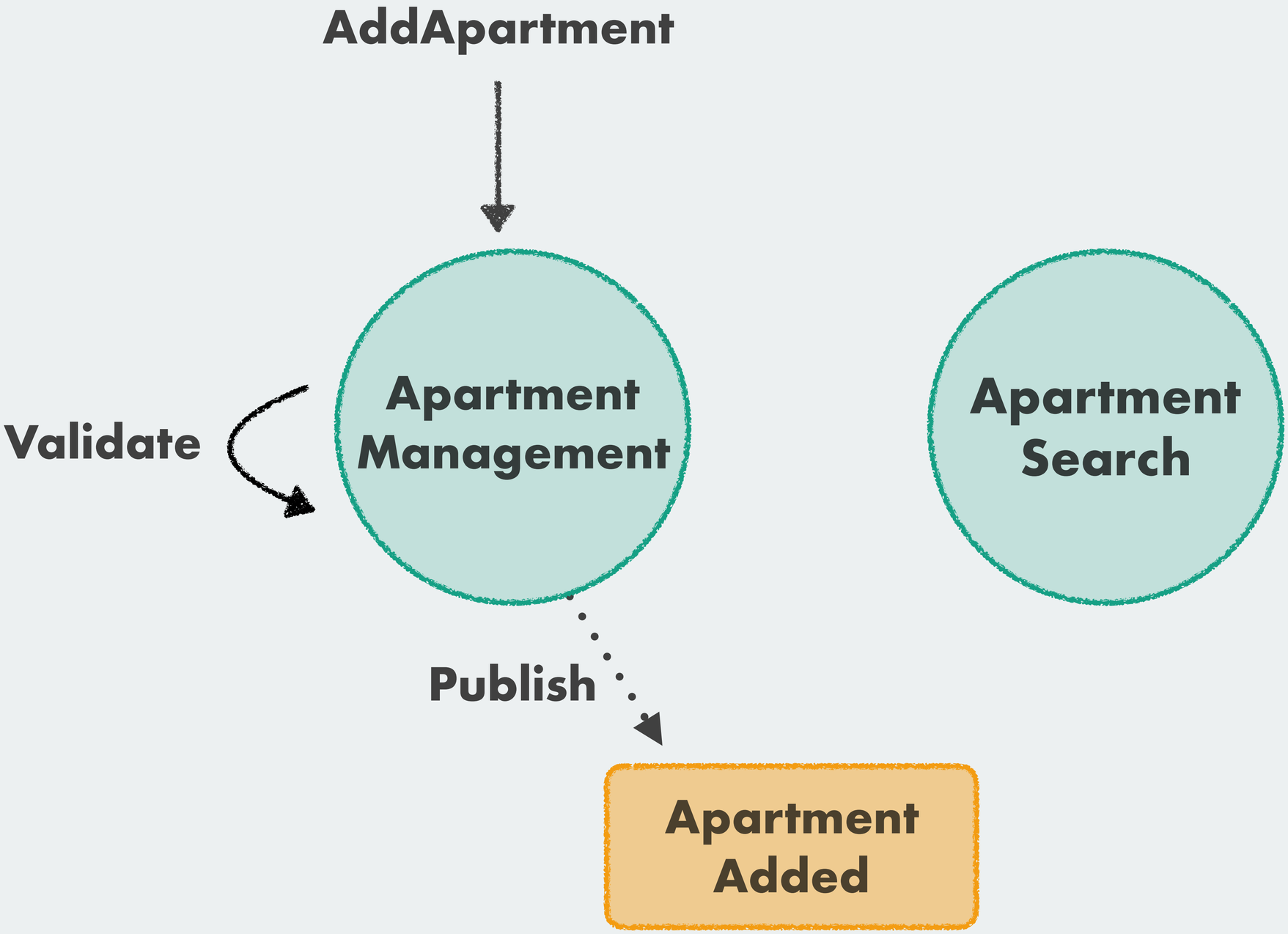
Adding and searching for apartments

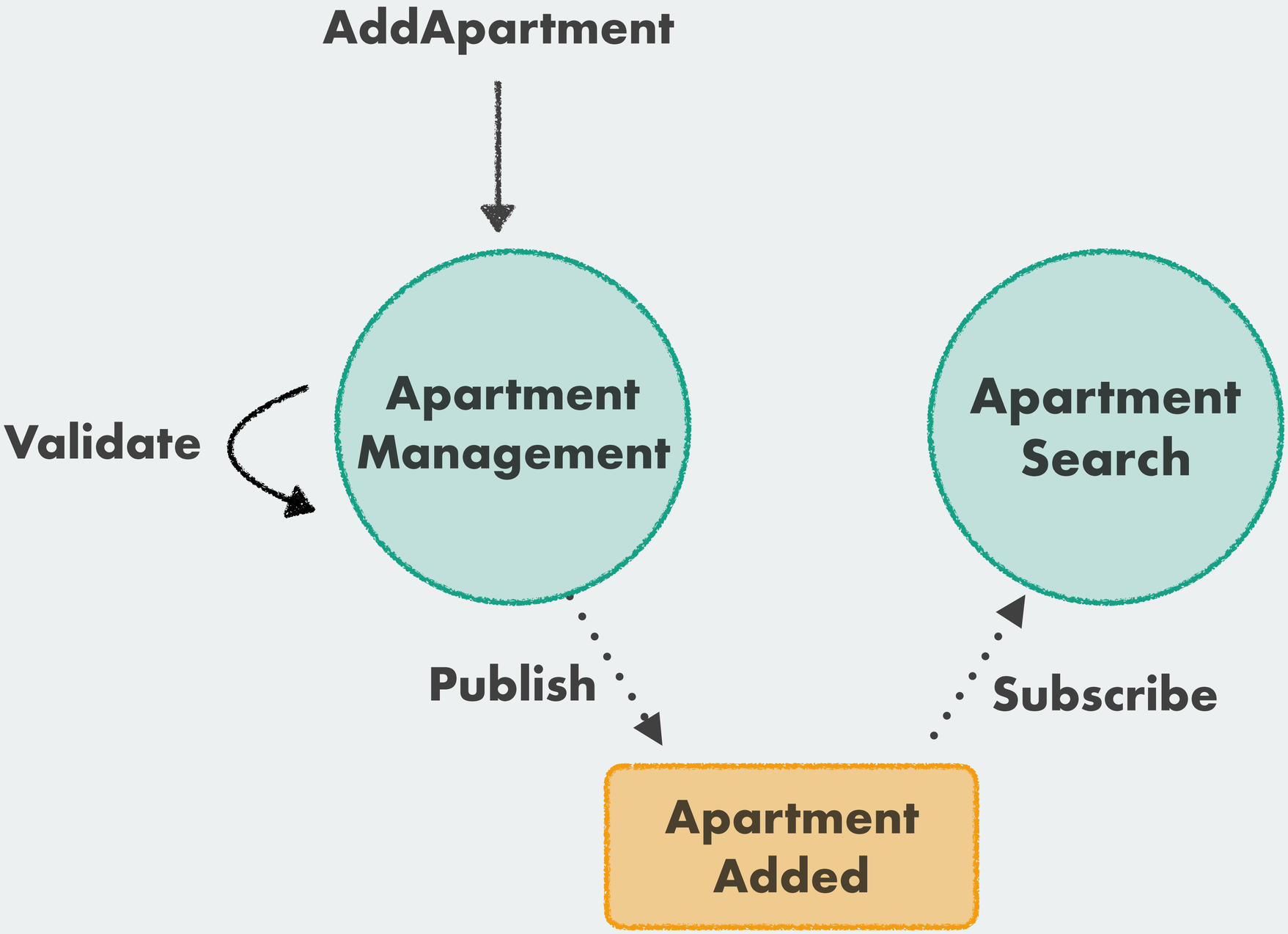
Apartment Management

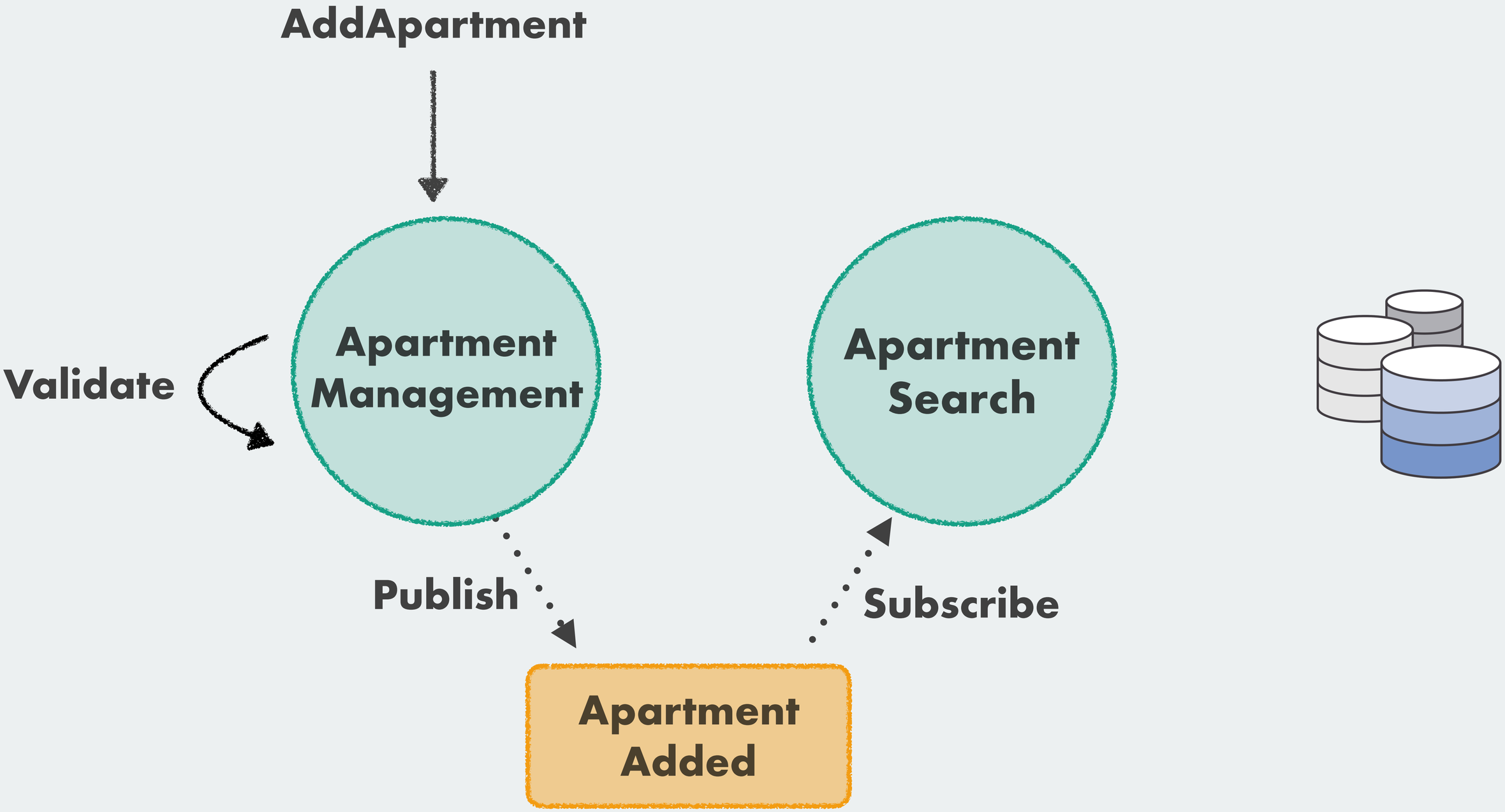

AddApartment
Apartment
Management

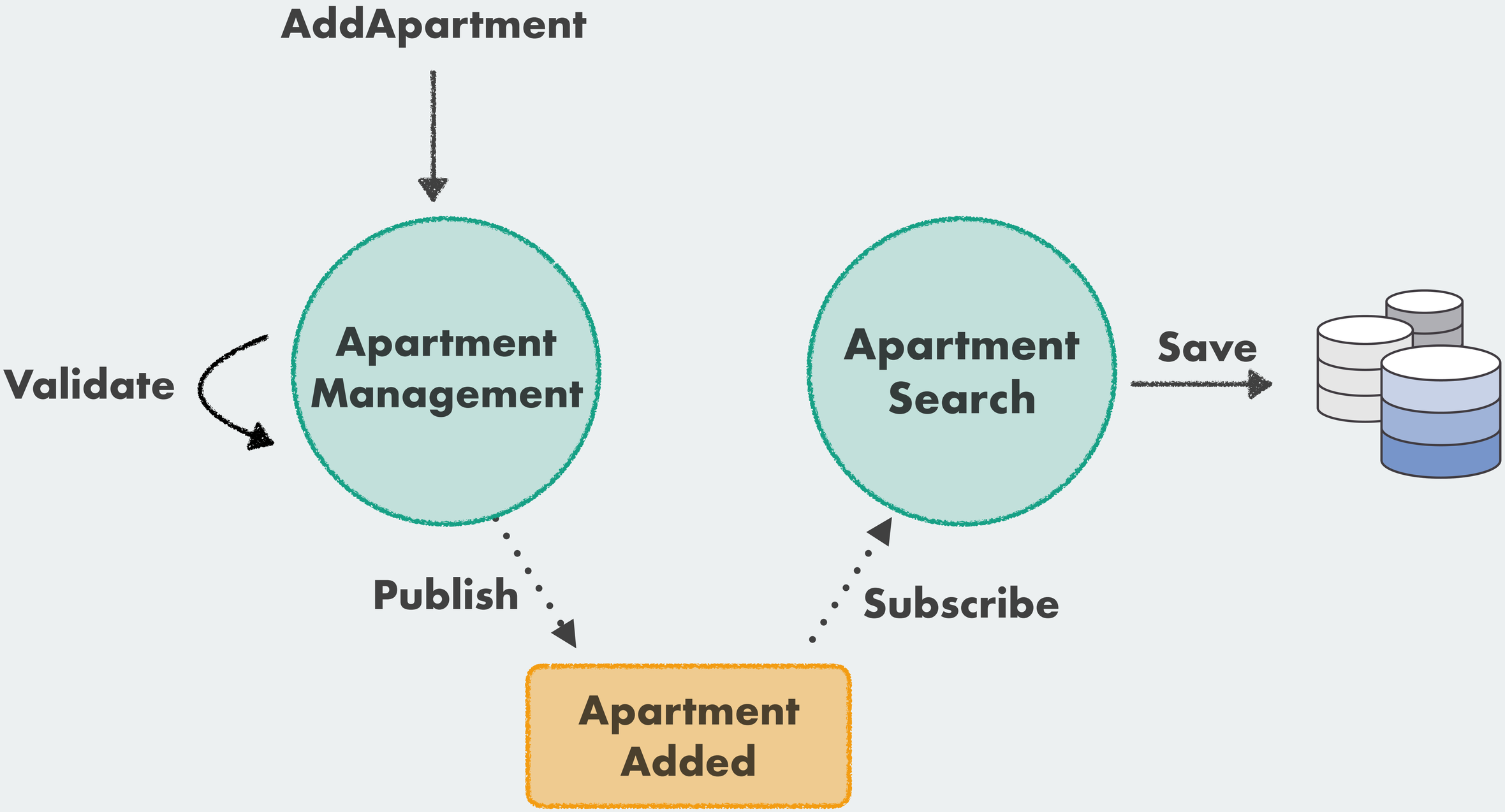

AddApartment
Validate
Apartment
Management


AddApartment
Apartment Management
Validate
Publish
Apartment Added


AddApartment
Validate
Apartment Management
Apartment Search
Publish
Apartment Added


AddApartment
Validate
Apartment Management
Apartment Search
Publish
Subscribe
Apartment Added


AddApartment
Validate
Apartment Management
Apartment Search
Publish
Subscribe
Apartment Added


AddApartment
Validate
Apartment Management
Apartment Search
Save
Publish
Subscribe
Apartment Added

Single responsibility

Validating apartments

Searching for apartments

What are events?

Events

- facts
- things that happened (in the past)
- immutable
- **map closely to business logic**
- **reduce coupling**

Events

- facts (things that happened)
- multiple handlers
- cannot be rejected
- past tense

e.g. **ApartmentAdded**

Commands

- intentions, requests
- one handler
- can be rejected
- present tense

e.g. **AddApartment**

Let’s see the code

```elixir
defmodule Airbnb.ApartmentManagement.Logic do

  def add_apartment(%AddApartment{} = command) do
    case validate(command) do
      :ok ->
        event = event_from_command(command)
        [event]

      {:error, changeset} ->
        {:error, changeset}
    end
  end
  # …
```

```elixir
defmodule Airbnb.ApartmentManagement do
  @moduledoc "Public interface for managing apartments"

  def add_apartment(%AddApartment{} = command) do
    case ApartmentManagement.Logic.add_apartment(command) do
      {:error, changeset} -> {:error, changeset}

      events when is_list(events) ->
        Enum.each(events, fn event ->
          PubSub.publish("apartments", event)
        end)
    end
  end
```

```elixir
defmodule Airbnb.AddingApartments do
  @moduledoc "Public interface for adding apartments"


  def add_apartment(%AddApartment{} = command) do
    case AddingApartments.Logic.add_apartment(command) do
      {:error, changeset} -> {:error, changeset}

      events when is_list(events) ->
        Enum.each(events, fn event ->
          PubSub.publish("apartments", event)
        end)
    end

    :ok
  end
```

```elixir
defmodule Airbnb.AddingApartments do
  @moduledoc "Public interface for adding apartments"


  def add_apartment(%AddApartment{} = command) do
    case AddingApartments.Logic.add_apartment(command) do
      {:error, changeset} -> {:error, changeset}

      events when is_list(events) ->
        Enum.each(events, fn event ->
          PubSub.publish("apartments", event)
        end)
    end

    :ok
  end
```

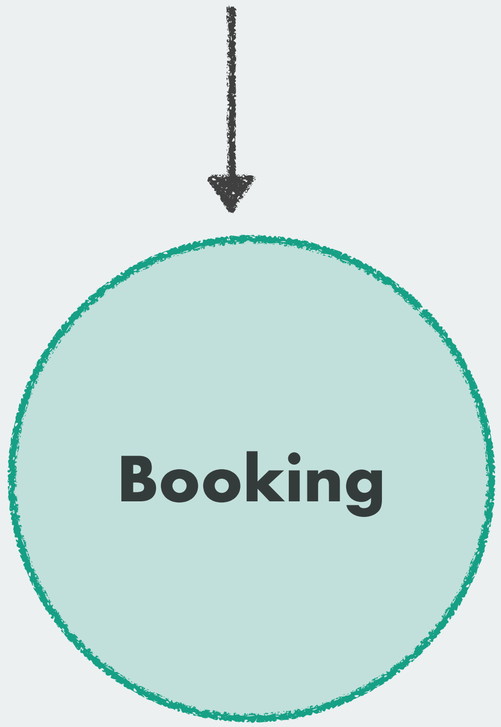
```elixir
defmodule Airbnb.ApartmentSearch do
  @behaviour Airbnb.PubSub.Handler

  def handle_event(%ApartmentAdded{} = event) do
    event
    |> Apartment.changeset_from_event()
    |> Airbnb.Repo.insert!(on_conflict: :nothing)

    :ok
  end
end
```

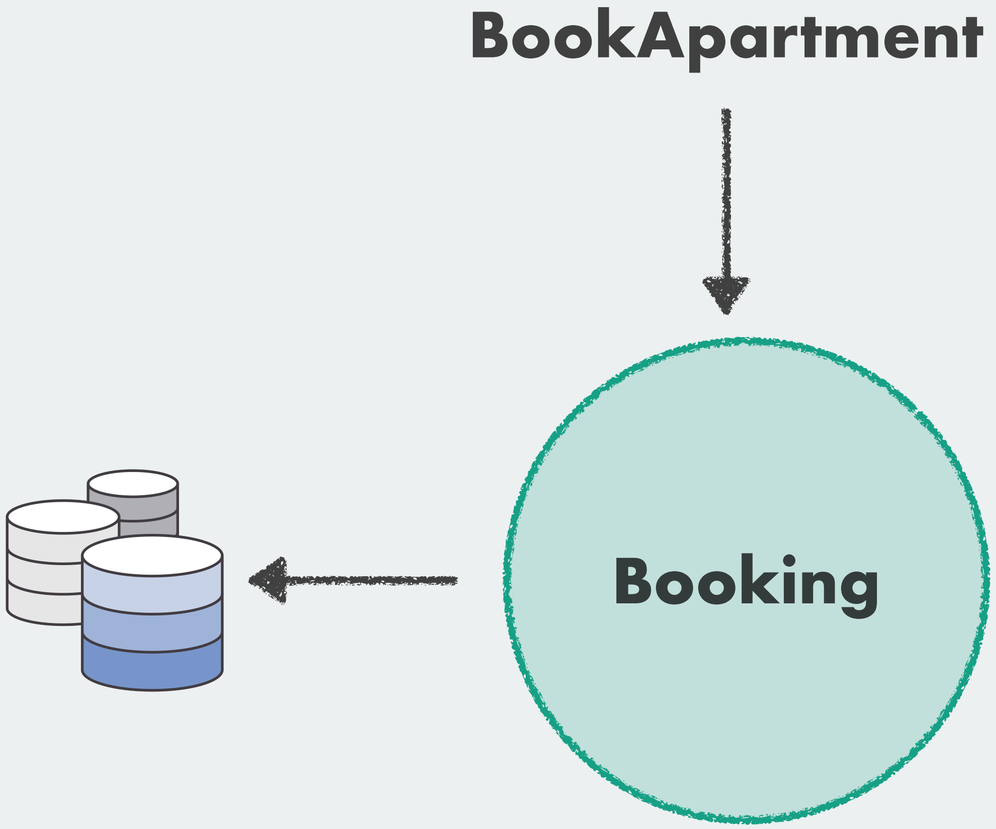
Pure functions expressing business rules

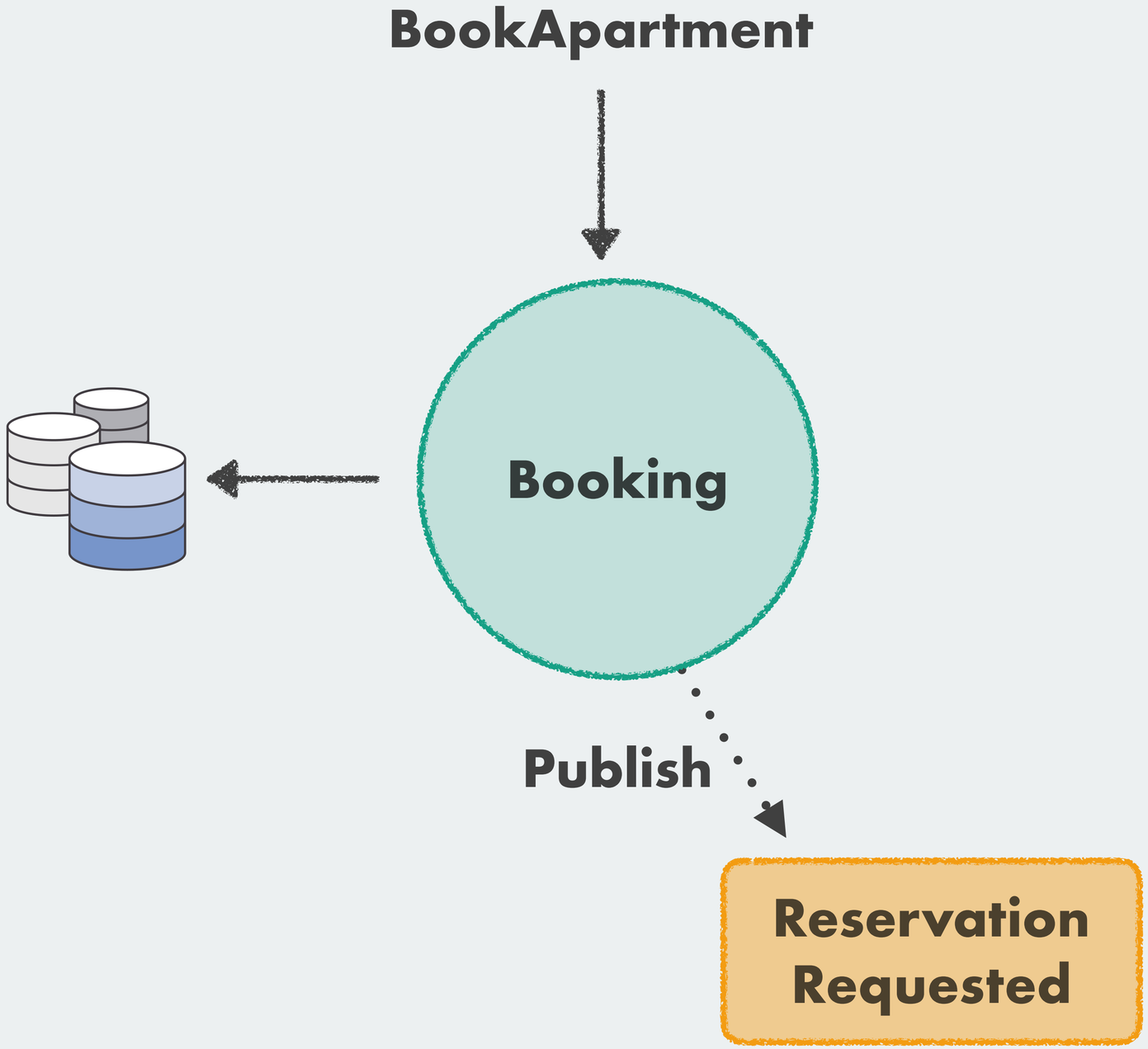
Side effects only caused by applying events

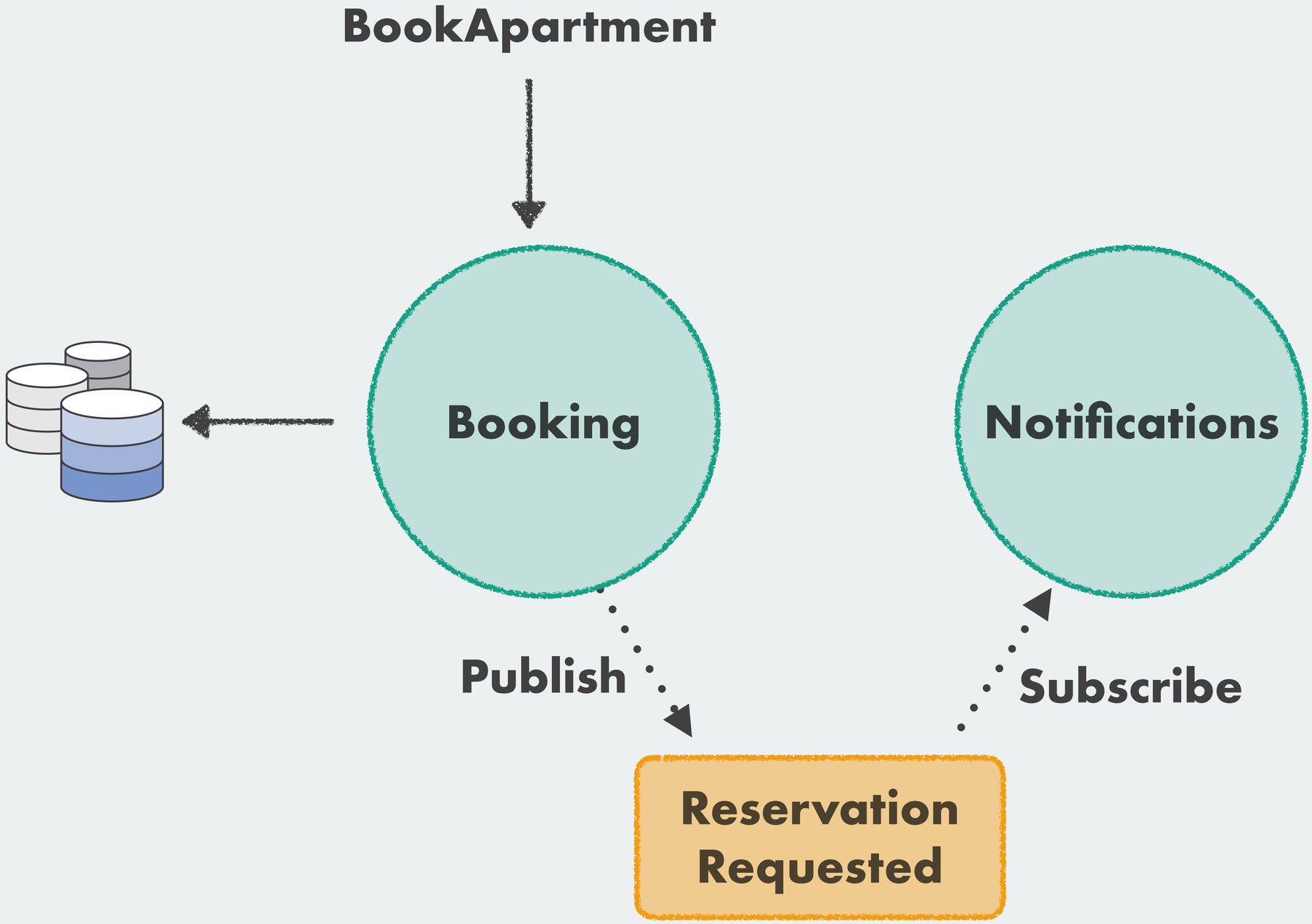

Booking

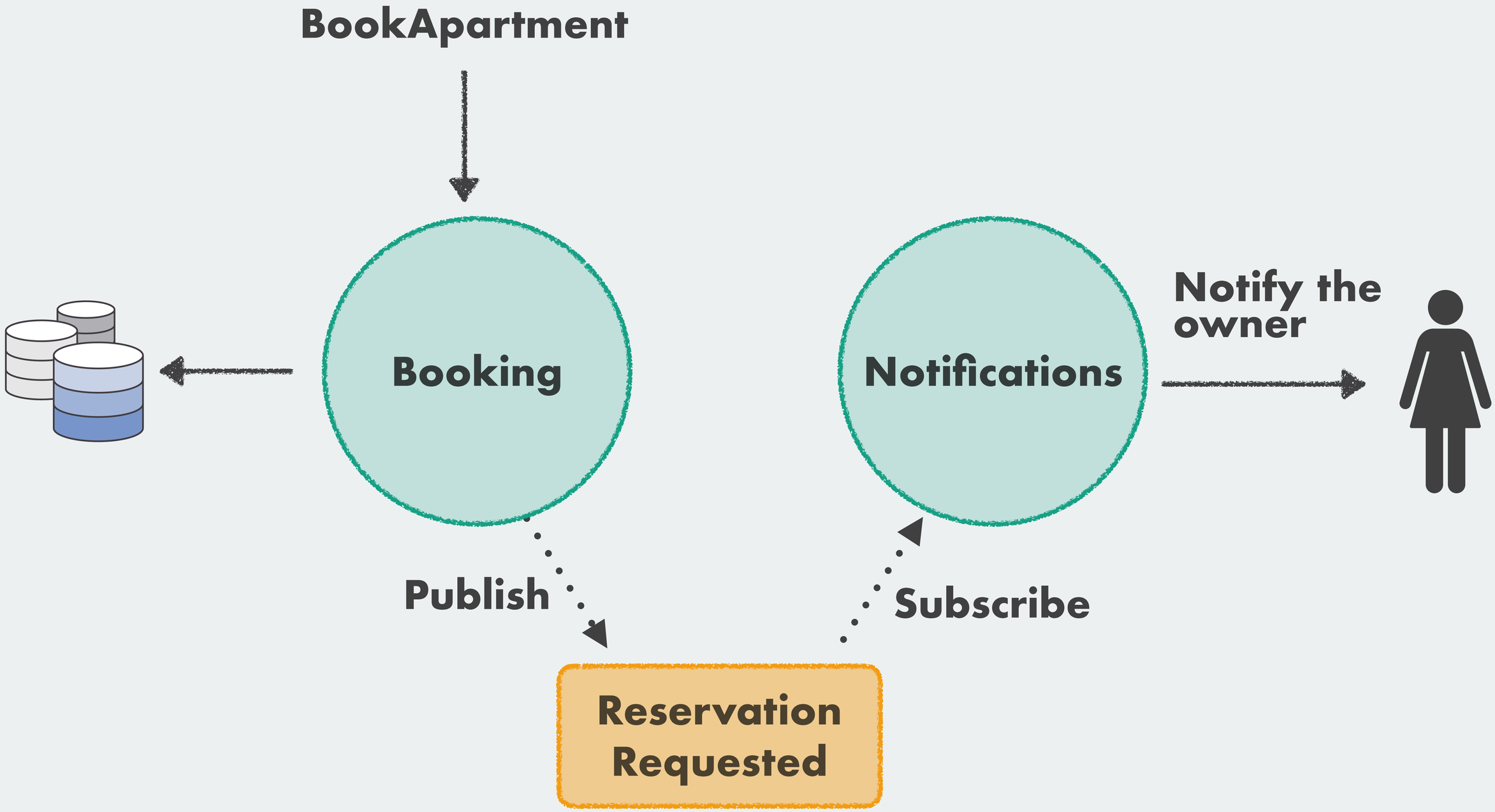
BookApartment

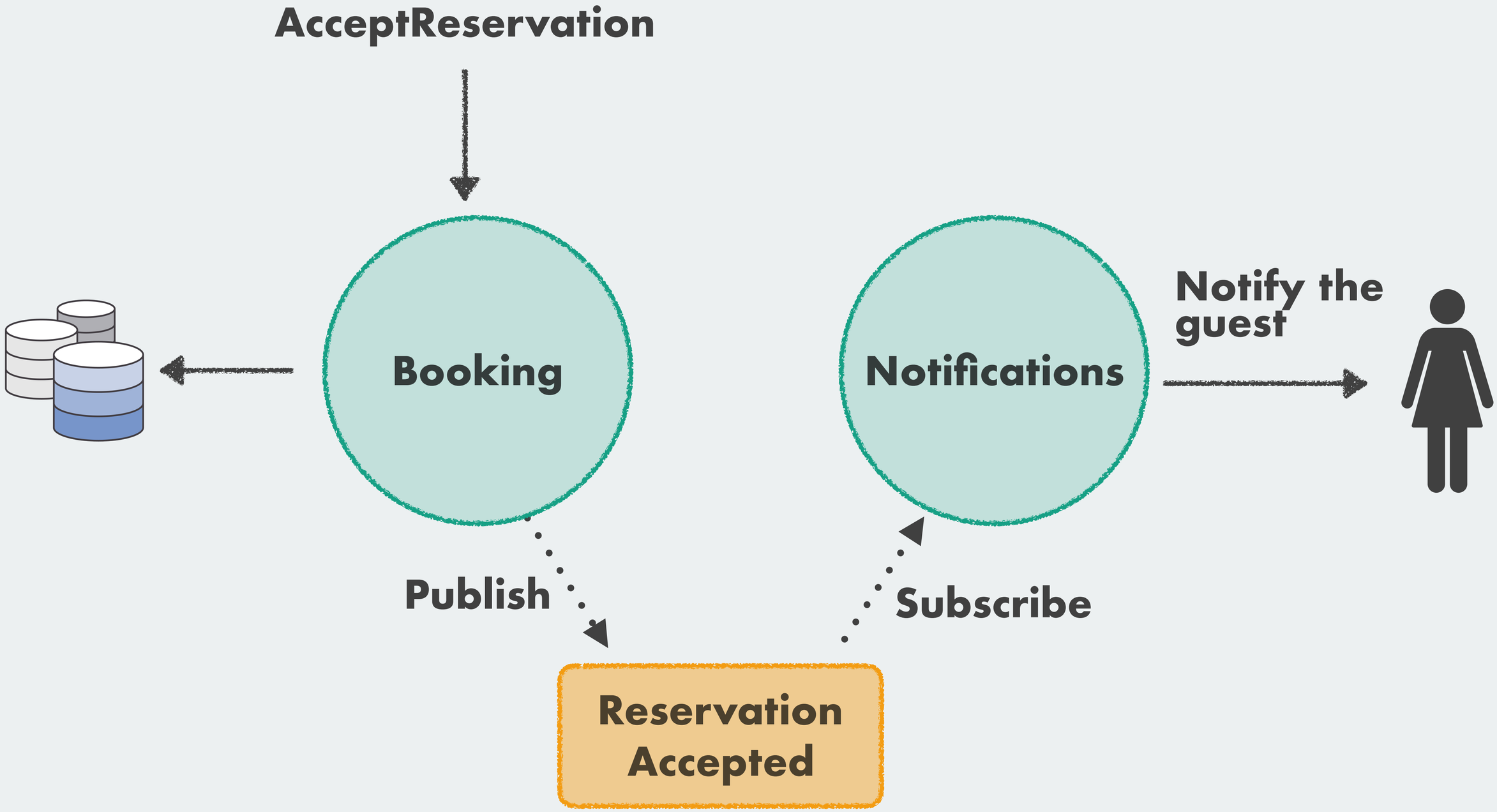


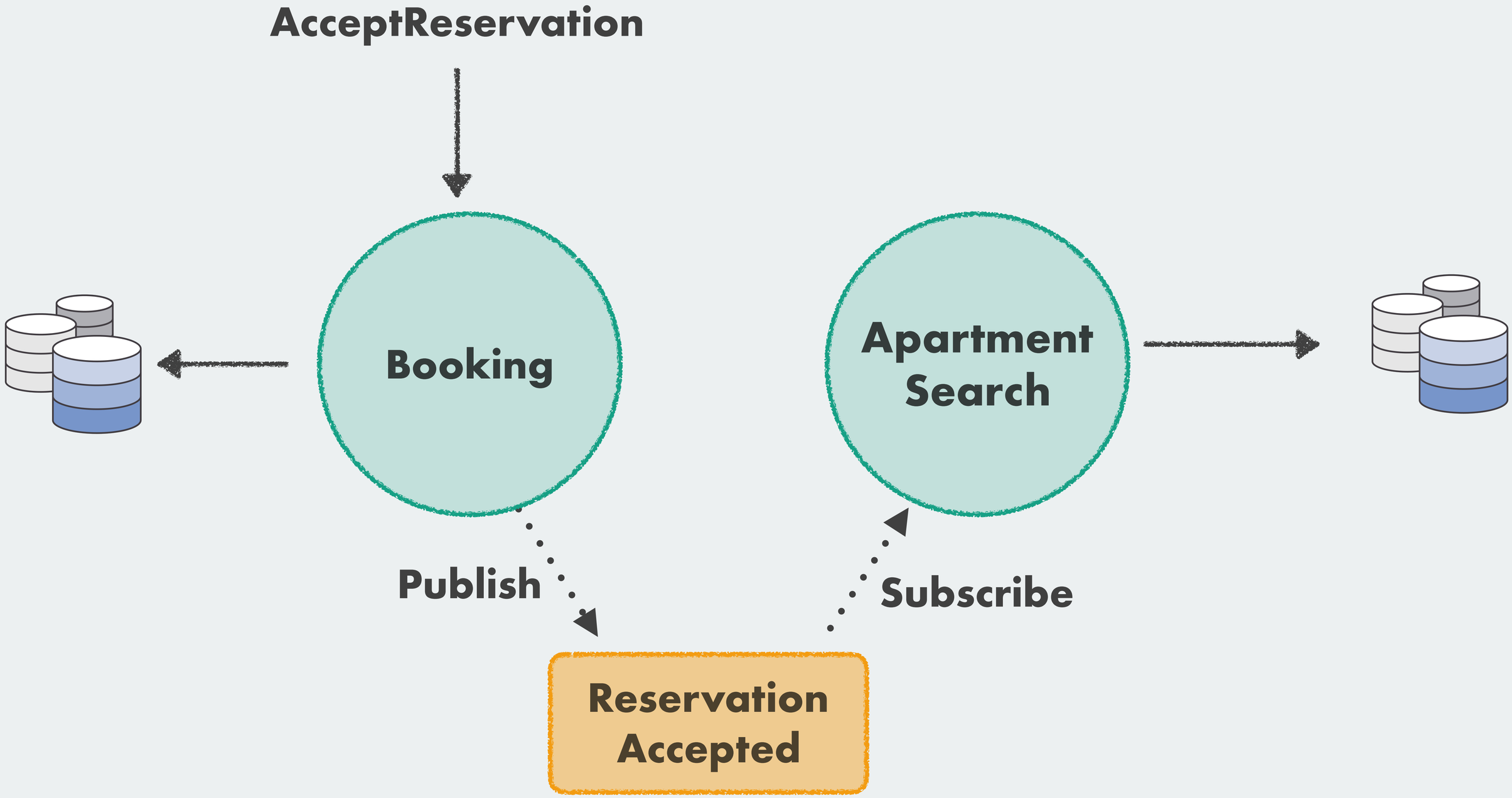

BookApartment
Booking

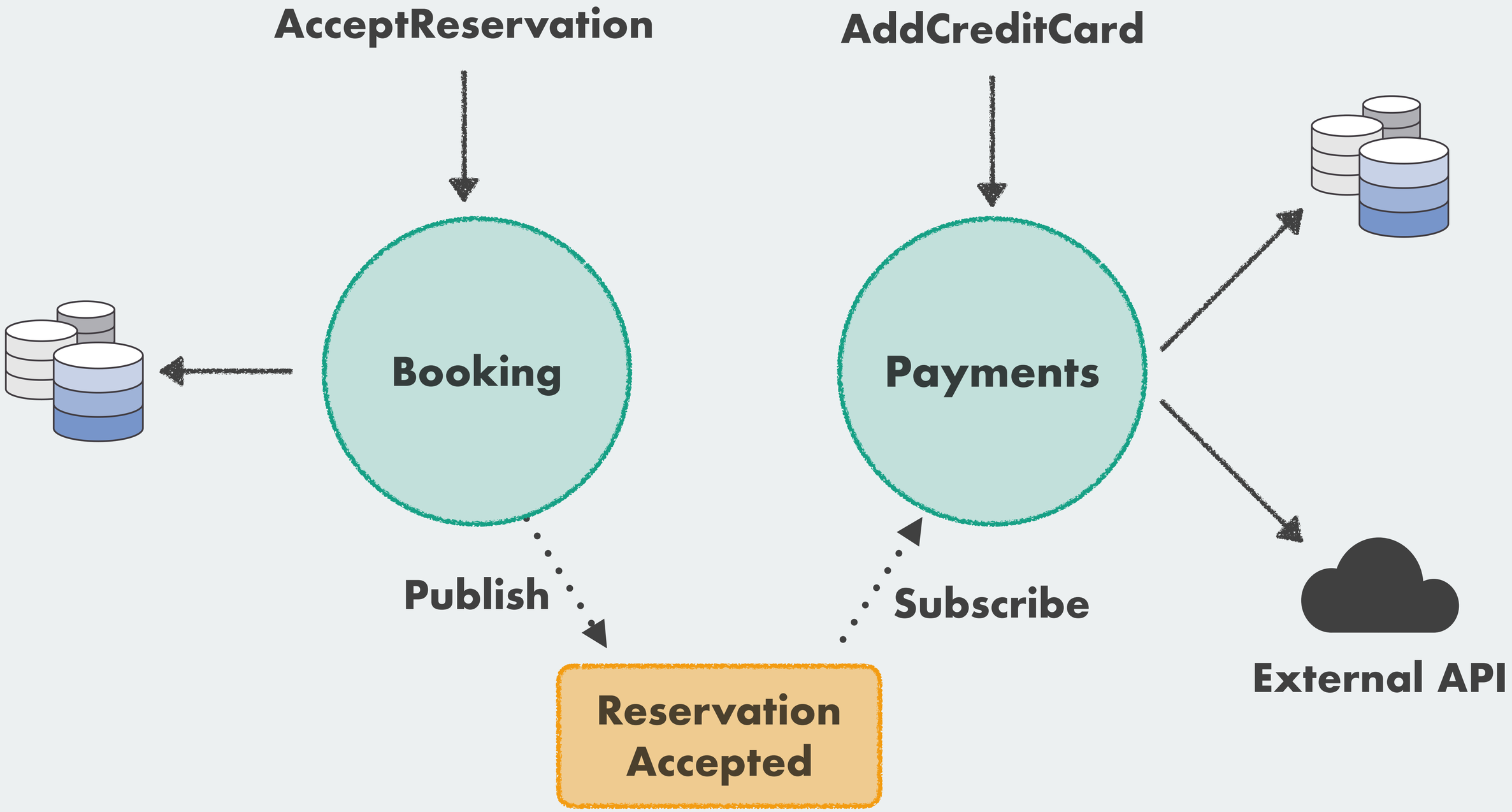

BookApartment
Booking
Publish
Reservation Requested

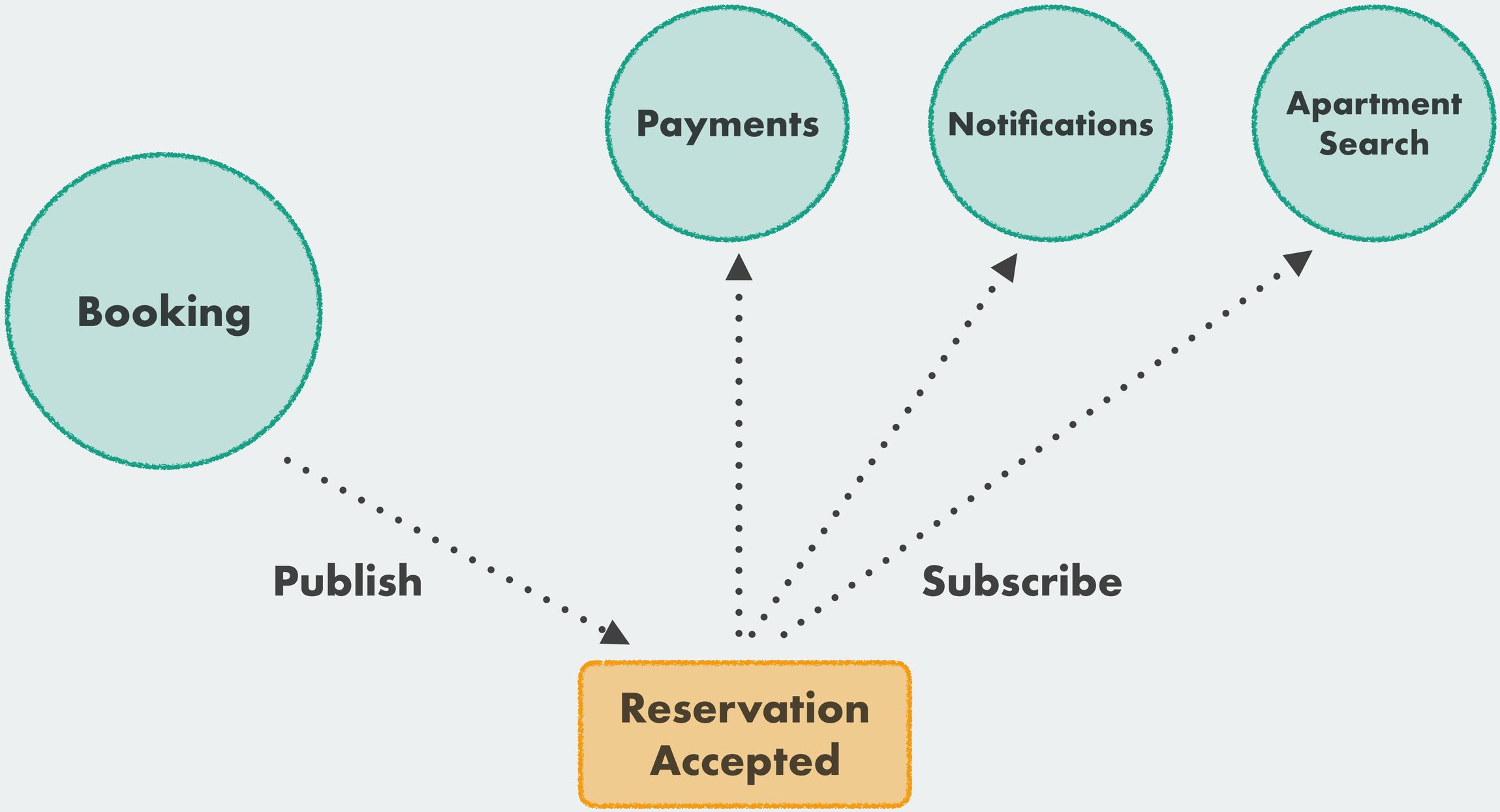

BookApartment
Booking
Notifications
Publish
Subscribe
Reservation Requested


BookApartment
Booking
Notifications
Notify the owner
Publish
Subscribe
Reservation Requested


AcceptReservation
Booking
Notifications
Notify the guest
Publish
Subscribe
Reservation Accepted


AcceptReservation
Booking
Apartment Search
Publish
Subscribe
Reservation Accepted


AcceptReservation
AddCreditCard
Booking
Payments
Publish
Subscribe
Reservation Accepted
External API


Payments
Notifications
Apartment Search
Booking
Publish
Subscribe
Reservation Accepted

Small, decoupled, autonomous parts

Each part of the system

1. What do I really have to do?
2. What do I need to know to do this?
3. How can I know this?

Apartment Management

Responsibility	Data	Data Source
Validating commands (business rules)	Business rules	Directly from the code

Apartment Search

Responsibility	Data	Data Source
Searching for apartments	Available apartments	Apartment-related events

Payments

Responsibility	Data	Data Source
Charging the guests	Billing info	Data submitted by the guest
	Accepted requests	RequestAccepted event

So, what about PubSub?

```elixir
defmodule Airbnb.PubSub do
  def publish(topic, event) do
    handlers = Application.get_env(:airbnb, :handlers)[topic]

    Enum.each(handlers || [], fn handler ->
      :ok = handler.handle_event(event)
    end)

    :ok
  end
end
```

Using as a dispatcher

`Registry` has a dispatch mechanism that allows developers to implement custom dispatch logic triggered from the caller. For example, let's say we have a duplicate registry started as so:

```
{:ok, _} = Registry.start_link(keys: :duplicate, name: Registry.DispatcherTest)
```

By calling `register/3`, different processes can register under a given key and associate any value under that key. In this case, let's register the current process under the key `"hello"` and attach the `{IO, :inspect}` tuple to it:

```
{:ok, _} = Registry.register(Registry.DispatcherTest, "hello", {IO, :inspect})
```

Now, an entity interested in dispatching events for a given key may call `dispatch/3` passing in the key and a callback. This callback will be invoked with a list of all the values registered under the requested key, alongside the pid of the process that registered each value, in the form of `{pid, value}` tuples. In our example, `value` will be the `{module, function}` tuple in the code above:

```
Registry.dispatch(Registry.DispatcherTest, "hello", fn entries ->
  for {pid, {module, function}} <- entries, do: apply(module, function, [pid])
end)
# Prints #PID<...> where the pid is for the process that called register/3 above
#=> :ok
```

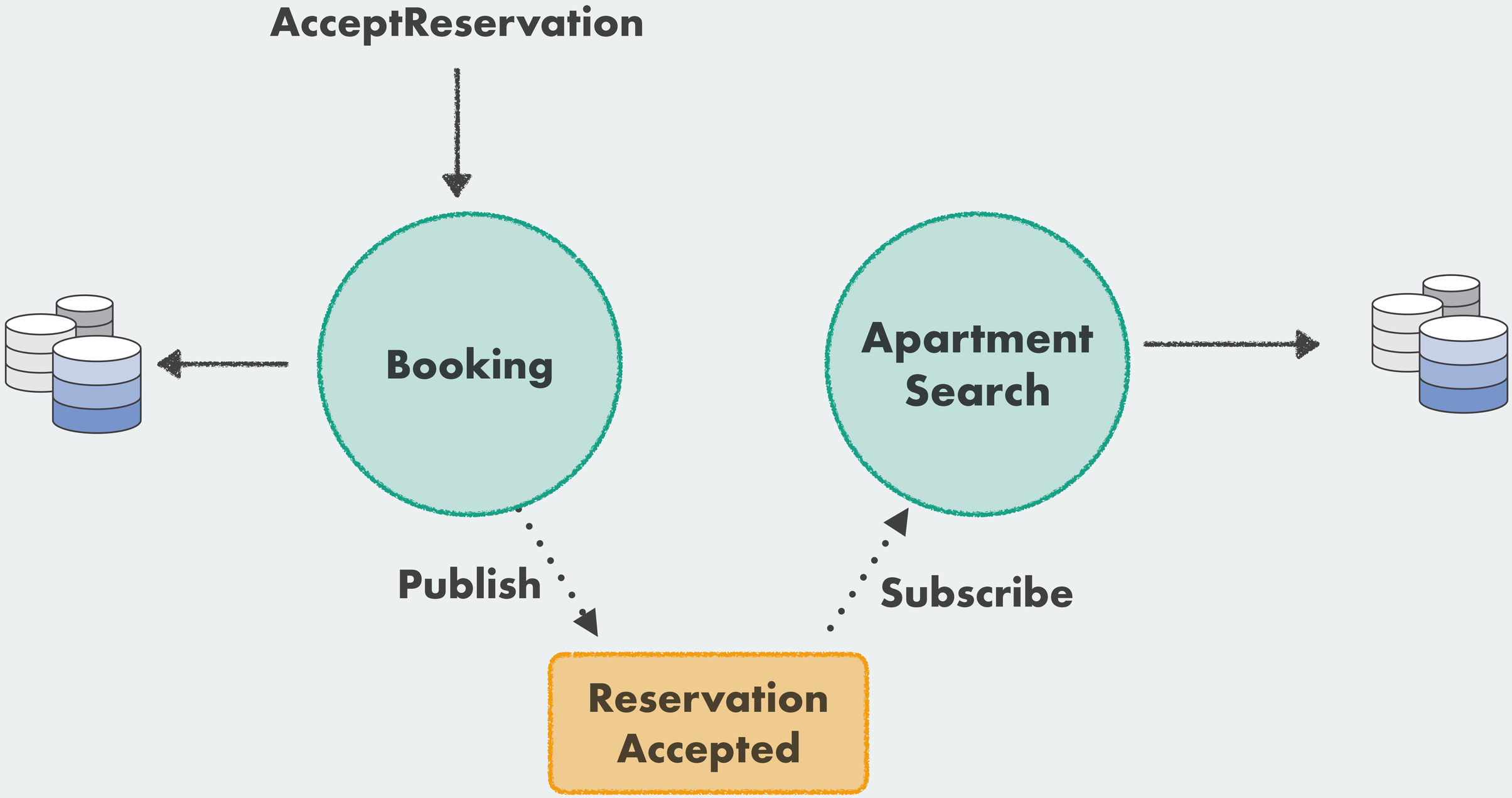
Dispatching happens in the process that calls `dispatch/3` either serially or concurrently in case of multiple partitions (via spawned tasks). The registered processes are not involved in dispatching unless involving

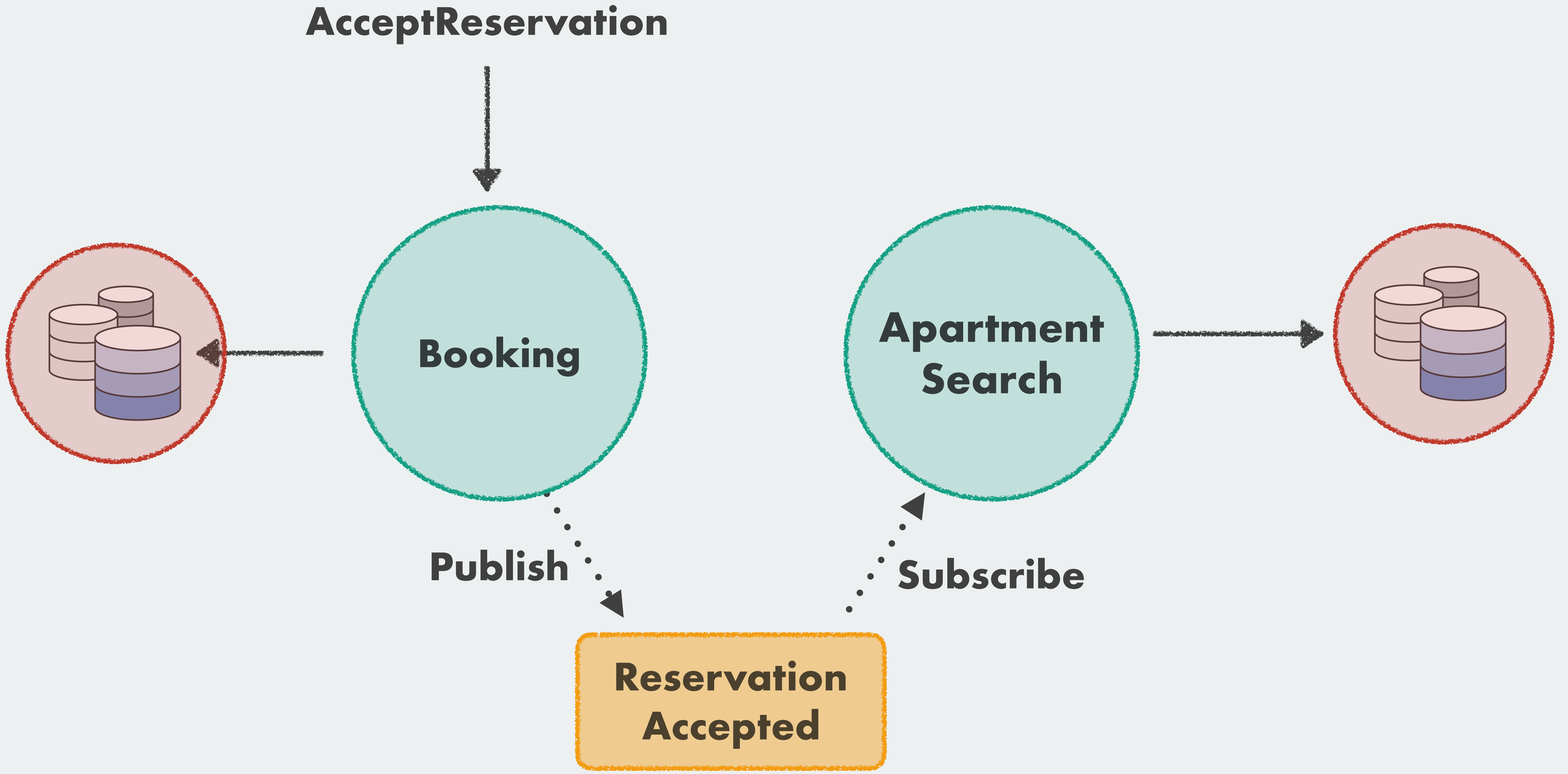
Doesn’t really matter

Let's talk about DRY

Don't Repeat Yourself

"Every piece of knowledge must have a single, unambiguous, authoritative representation within a system"


AcceptReservation
Booking
Apartment Search
Publish
Subscribe
Reservation Accepted

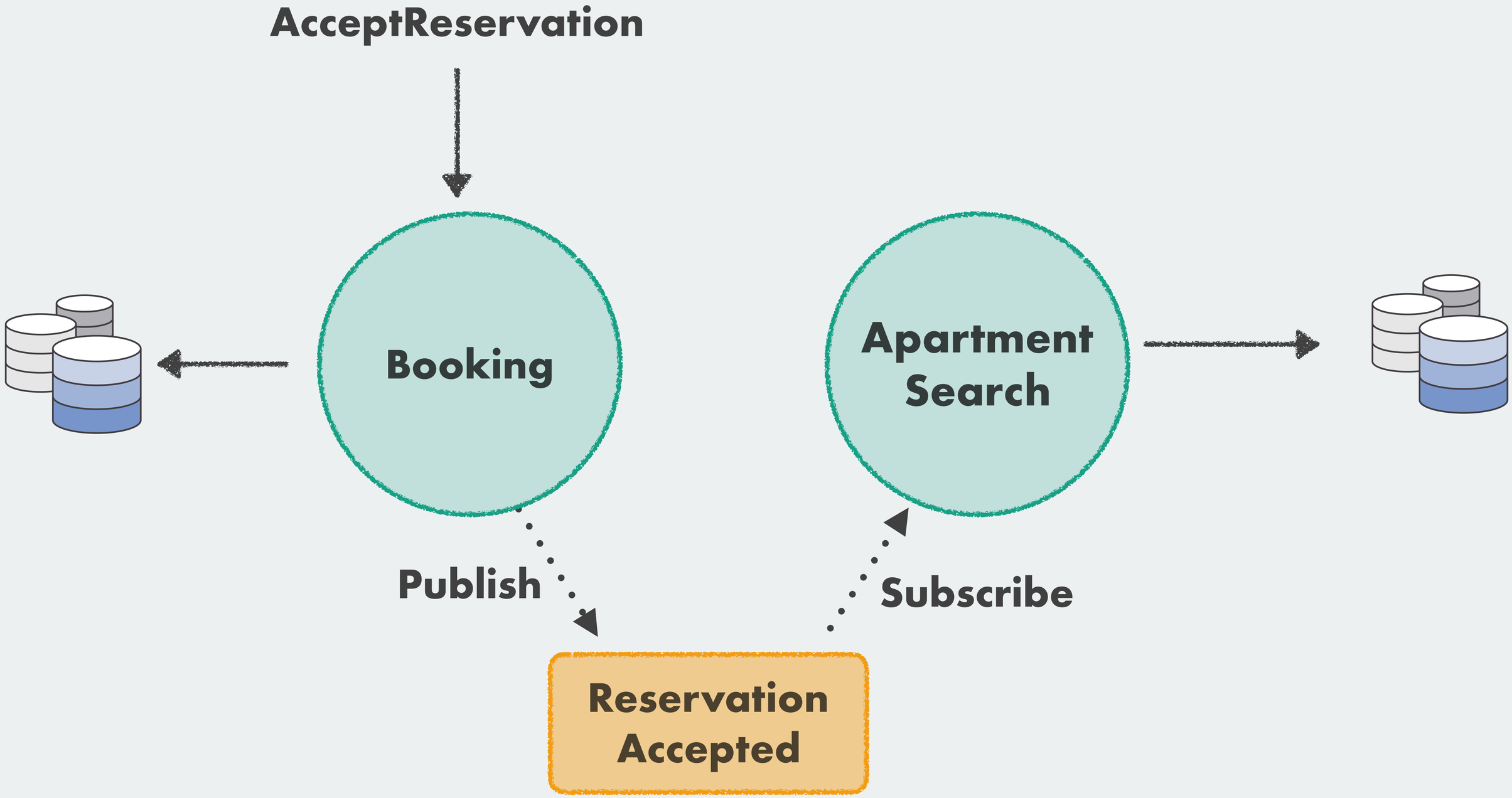

AcceptReservation
Booking
Apartment Search
Publish
Subscribe
Reservation Accepted

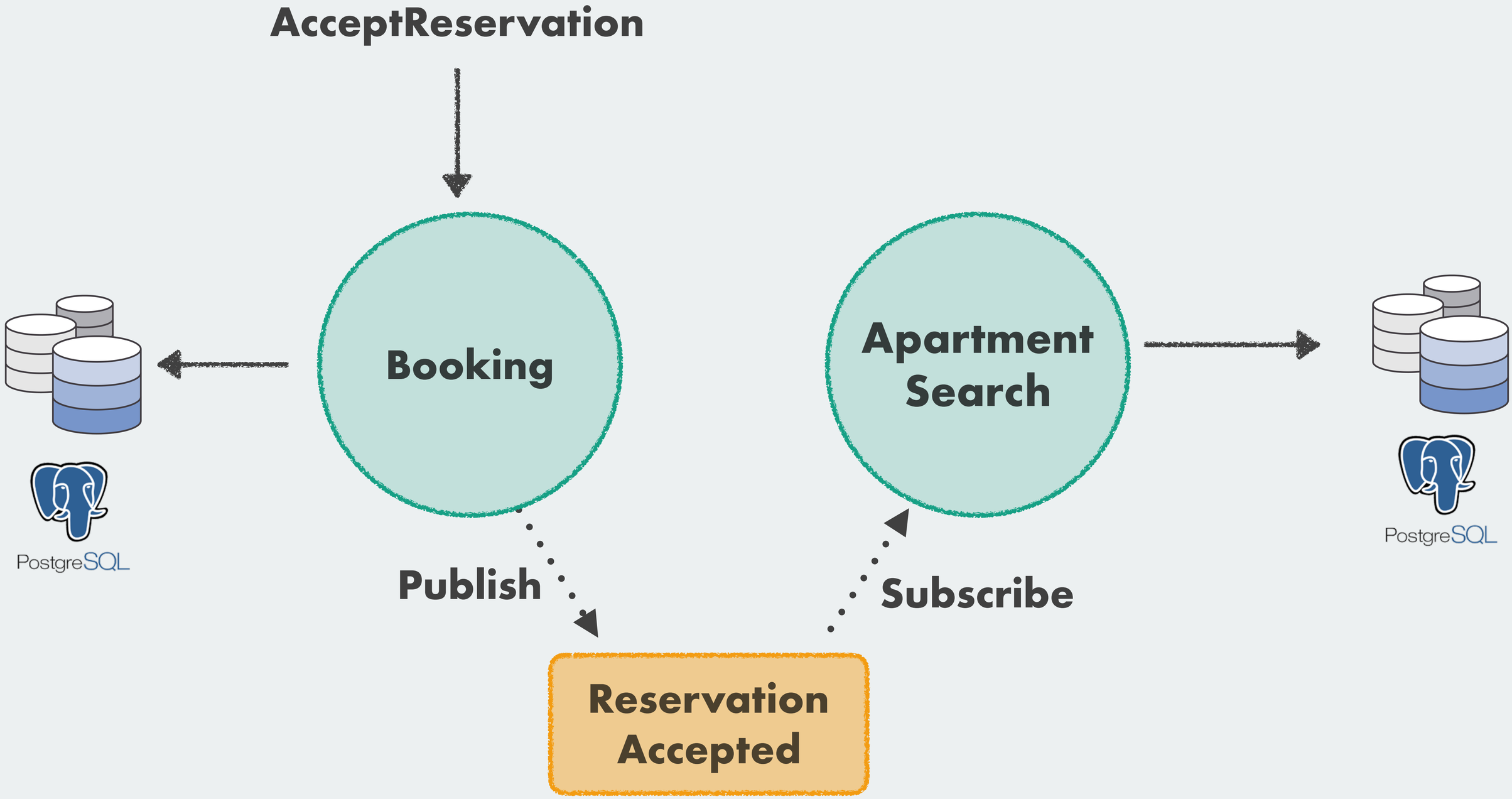
Why?

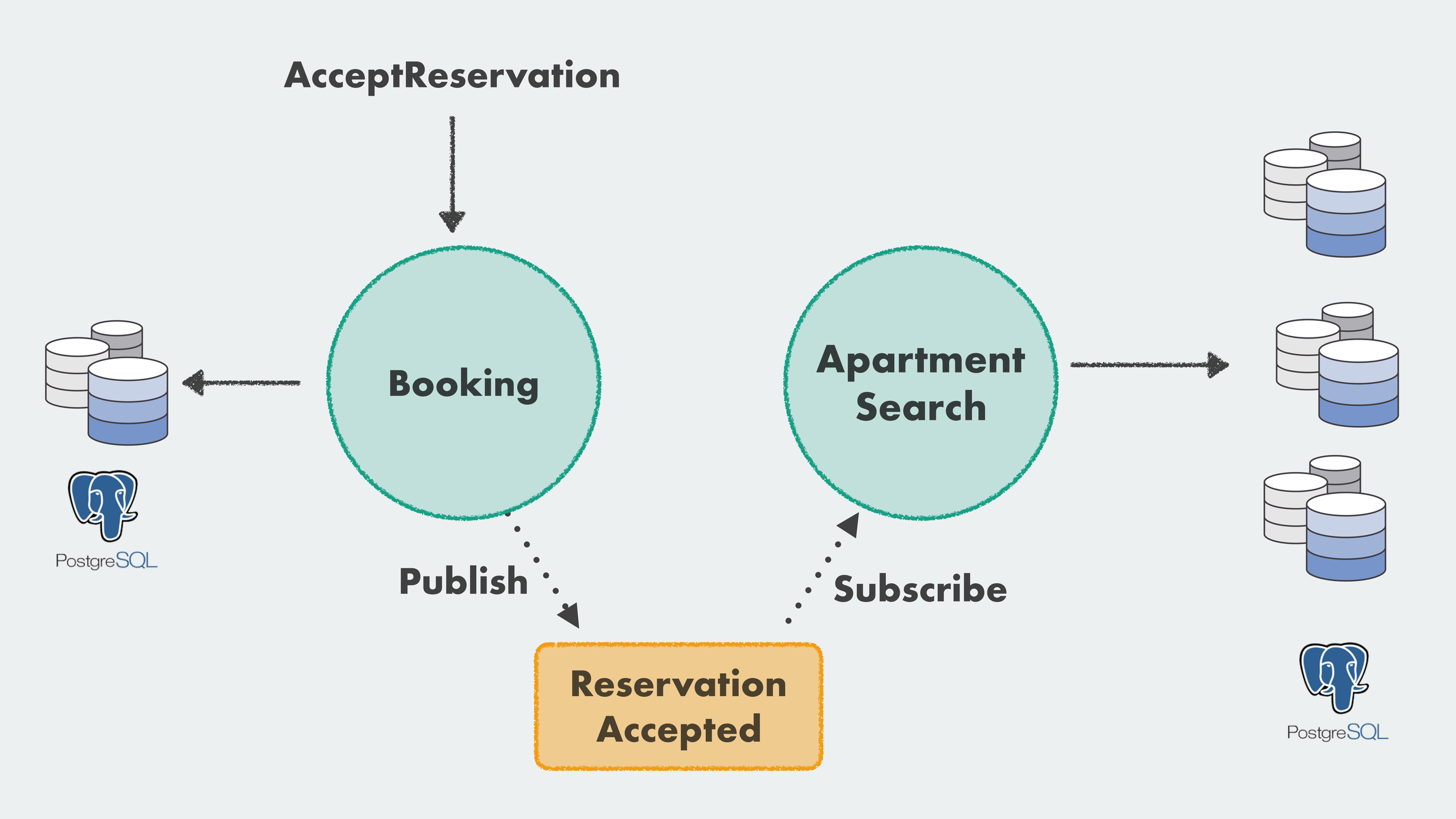
"When the DRY principle is applied successfully, a **modification of any single element of a system does not require a change in other logically unrelated elements.**"

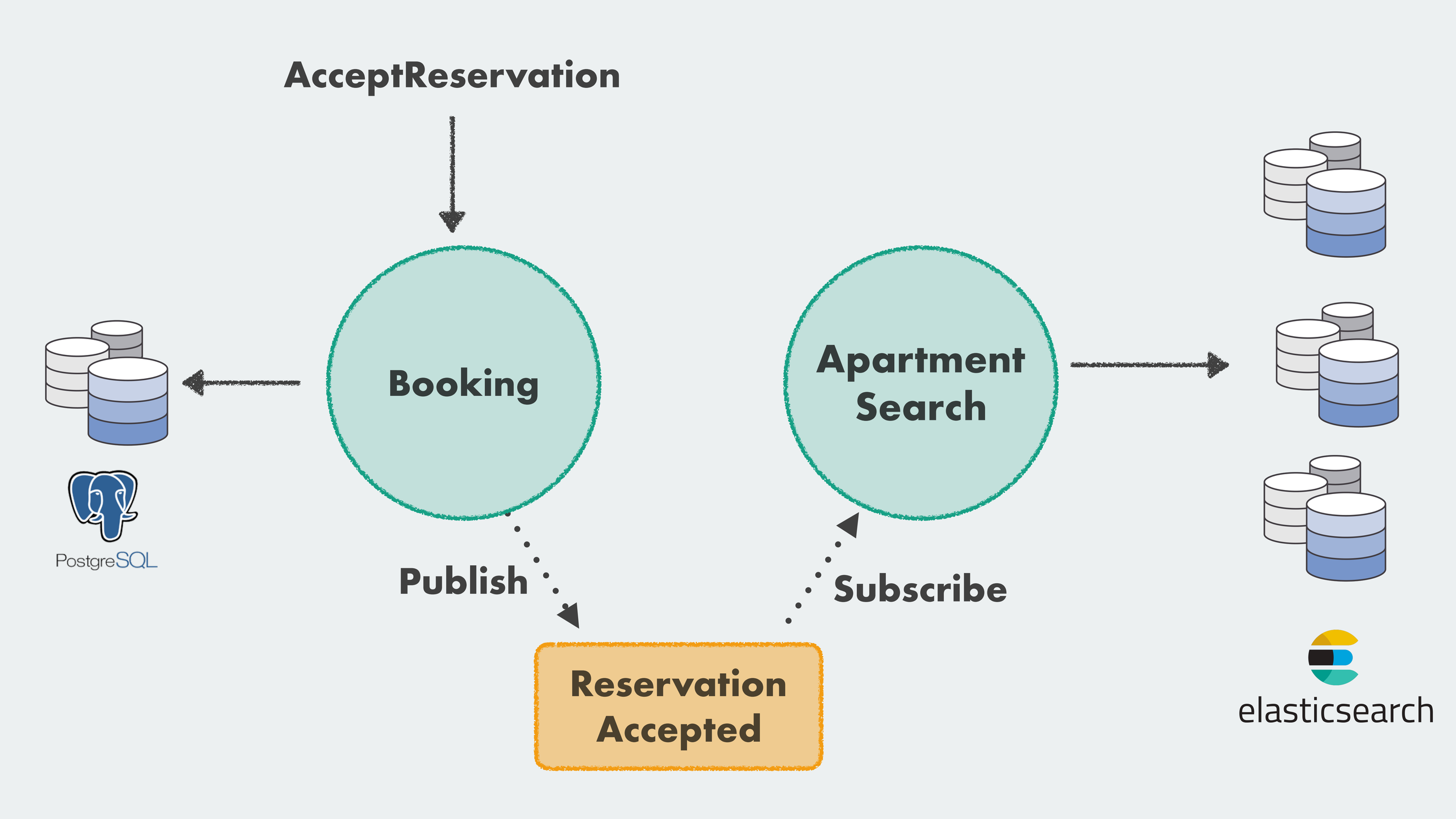
"**It's okay to have mechanical, textual duplication** (the equivalent of caching values: a repeatable, automatic derivation of one source file from some meta-level description), **as long as the authoritative source is well known**."

http://wiki.c2.com/?DontRepeatYourself


AcceptReservation
Booking
Apartment Search
Publish
Subscribe
Reservation Accepted


AcceptReservation
Booking
Apartment Search
PostgreSQL
PostgreSQL
Publish
Subscribe
Reservation Accepted


AcceptReservation
Booking
Apartment Search
PostgreSQL
Publish
Subscribe
Reservation Accepted
PostgreSQL


AcceptReservation
Booking
Apartment Search
PostgreSQL
Publish
Subscribe
Reservation Accepted
elasticsearch

Sometimes duplication allows independent changes

Summary

Benefits

- great isolation / loose coupling
- better domain model
- better boundaries
- small components
- scalable
- easy to change

Drawbacks

- more initial work (not always worth it)
- unfamiliar model
- (sometimes) eventual consistency / message loss / …
- versioning of events
- how to discover events?

More cool stuff

- Event Sourcing / CQRS
- Microservices
- Kafka, RabbitMQ, …
- Storage
- OTP / Message Passing

Think about behaviour

Think about behaviour

- what does this part do?
- what does it need to know to do its job?
- how to provide it with this information?
- **it's much important than internal data representation**

Use the tools wisely,
Don't focus on any "rules"

Thank you!

Maciej Kaszubowski

ElixirConfEU 2018

mkaszubowski94