

# Decouple your Applications

## with Symfony Messenger

Alexander M. Turek – nfq Summit 2023

# about:me

Software Developer – Freelancer

♥ php + Symfony + Doctrine ♥

🔥 Open Source 🔥

❤️ Legacy Code ❤️

💕 Father of Four 💕



# Symfony

a **framework** for PHP web applications  
and  
a set of reusable **components**



# Symfony

## Register new account

Email

Password

Repeat password



I accept the terms and conditions of this service.

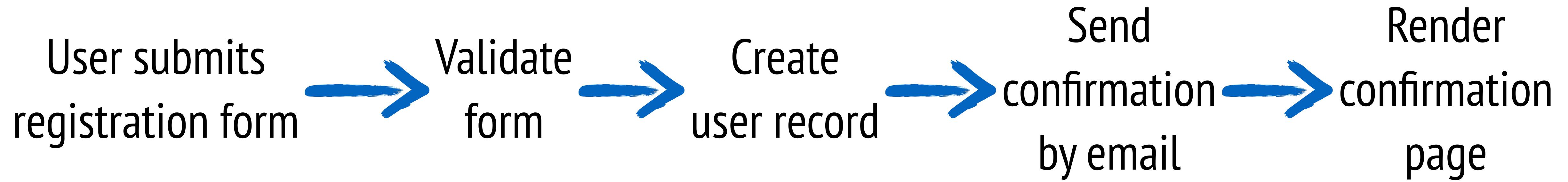
Submit

Thank you!

Thank you for registering.

An email with instructions for completing the registration will be sent to you shortly.

# Scenario: User Registration



# Sending emails is hard

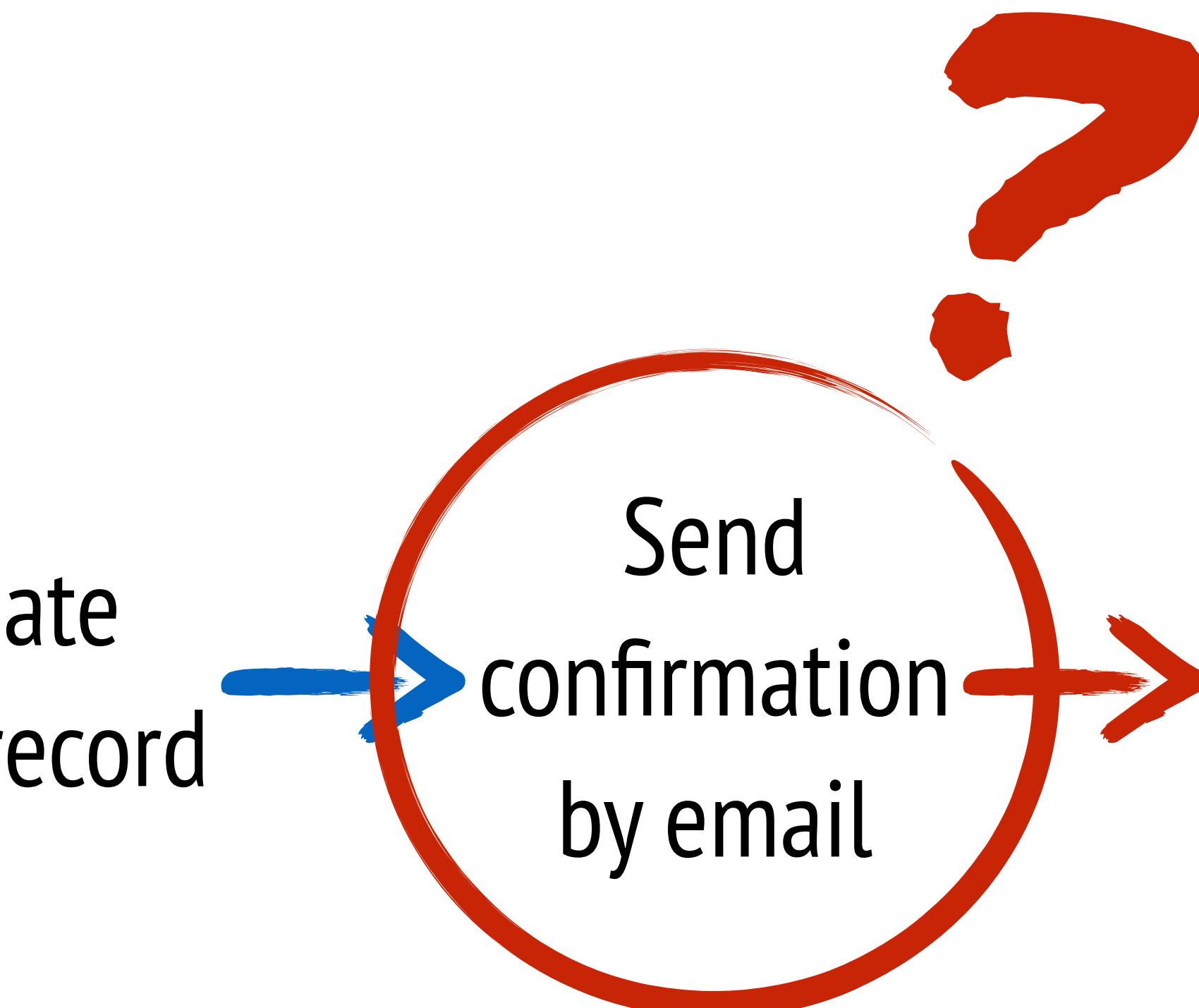
Sending an email could fail  
or take time.

Should we make the user wait?

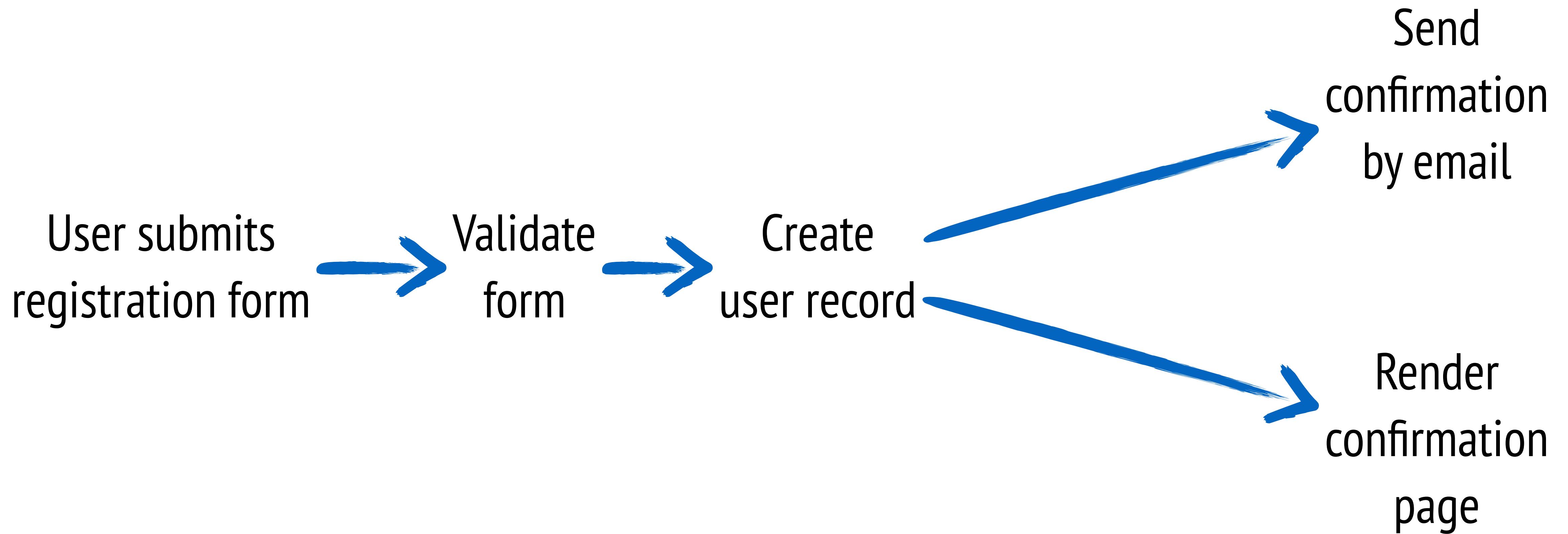
Create  
user record

Send  
confirmation  
by email

Render  
confirmation  
page



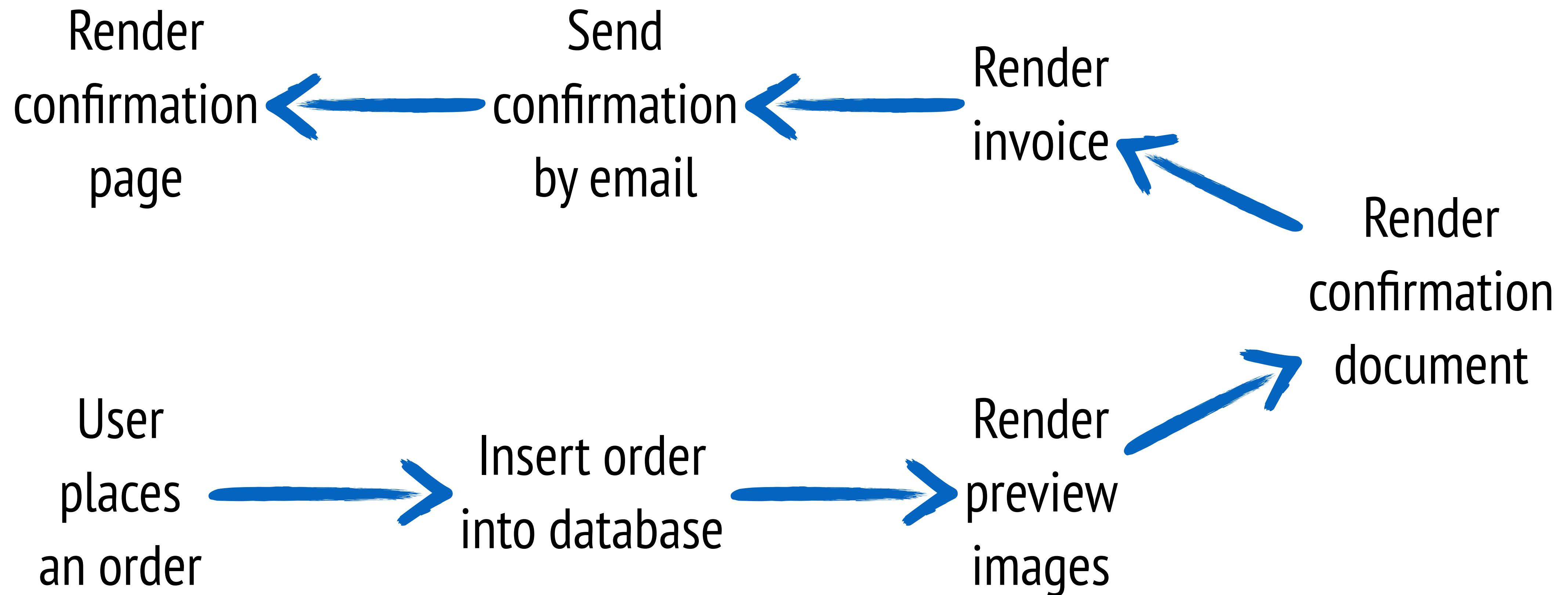
# Scenario: User Registration



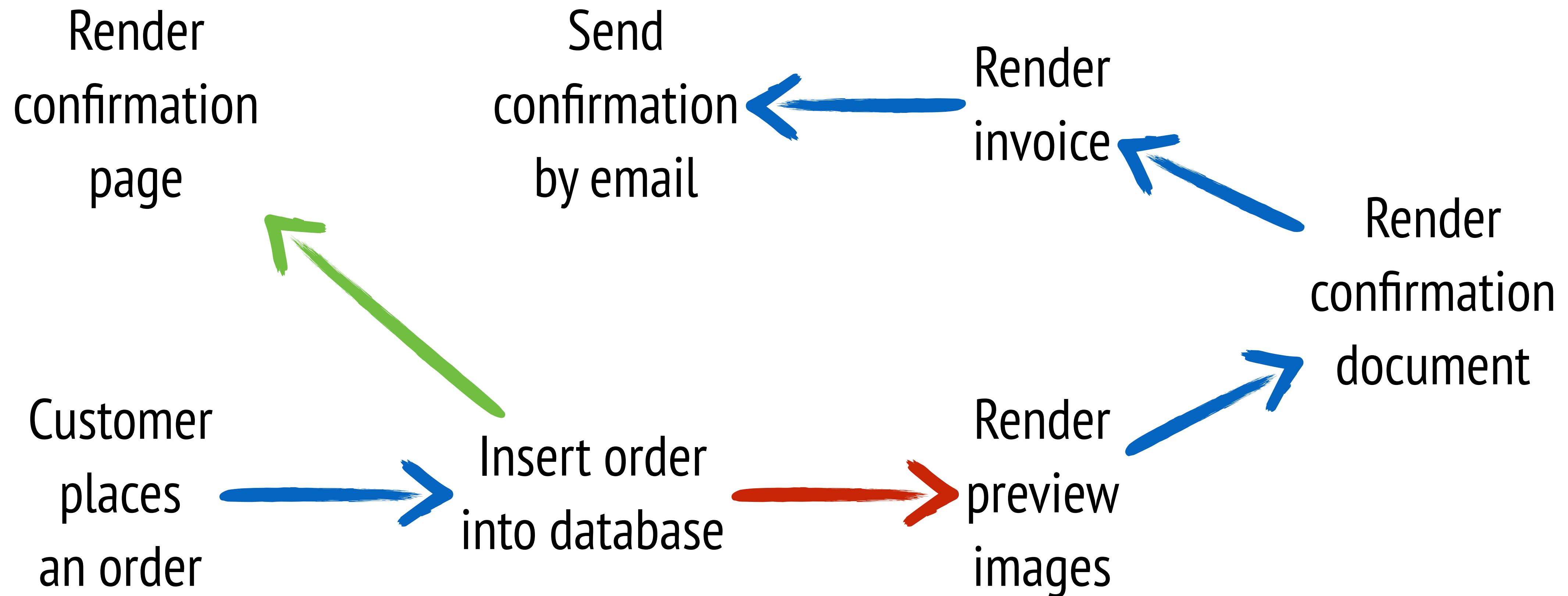
# Scenario: Customizable Products

- User can customize products with an online configurator tool.
- When placing an order, they receive a confirmation mail containing a PDF with preview images of their order along with an invoice.

# Scenario: Customizable Products



# Scenario: Customizable Products



# Process the order synchronously

```
[Route('/checkout/process', name: 'checkout.process', methods: ['POST'])]
public function placeOrderAction(Request $request): Response
{
    $order = $this→createOrderFromRequest($request);
    $this→entityManager→persist($order);
    $this→entityManager→flush();

    $this→renderPreviewImages($order);
    $this→renderConfirmationDocument($order);
    $this→renderInvoice($order);
    $this→sendConfirmationMail($order);

    return $this→redirectToRoute('checkout.thank_you');
}
```

# Process the order asynchronously

```
#Route('/checkout/process', name: 'checkout.process', methods: ['POST'])
public function placeOrderAction(Request $request): Response
{
    $order = $this->createOrderFromRequest($request);
    $this->entityManager->persist($order);
    $this->entityManager->flush();

    $this->messageBus->dispatch(new ProcessOrder(
        orderNo: $order->orderNo,
    ));

    return $this->redirectToRoute('checkout.thank_you');
}
```

```
#[AsMessageHandler]
final readonly class ProcessOrderHandler
{
    private function __construct(
        private EntityManagerInterface $entityManager,
        // ...
    ) {
    }

    public function __invoke(ProcessOrder $message): void
    {
        $order = $this->entityManager
            ->find(ProcessOrder::class, $message->orderNo);

        $this->renderPreviewImages($order);
        $this->renderConfirmationDocument($order);
        $this->renderInvoice($order);
        $this->sendConfirmationMail($order);
    }

    // ...
}
```

# Symfony Messenger

Messenger provides

**a message bus**

with the ability to send messages and  
then **handle them immediately**  
in your application or  
**send them through transports**  
to be handled later.



# Symfony

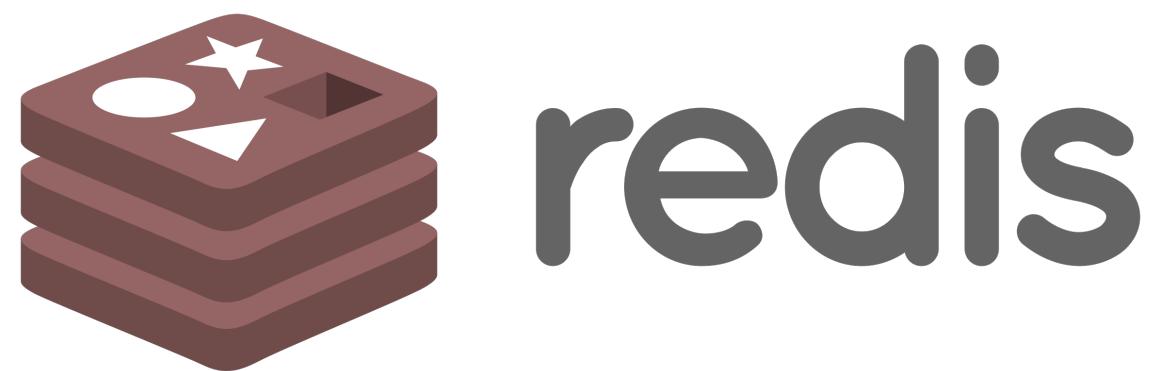
# Transports



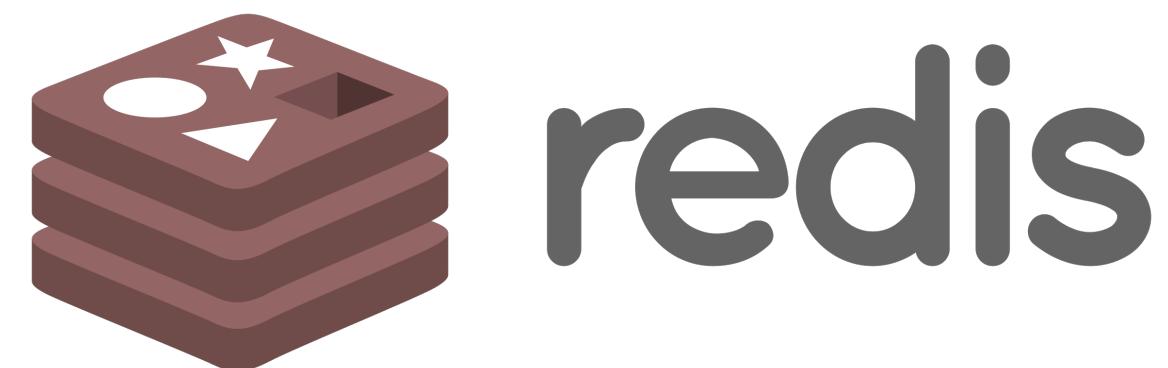
# Transports



# Transports



# Transports



# Transports



# Design Goal: Portability

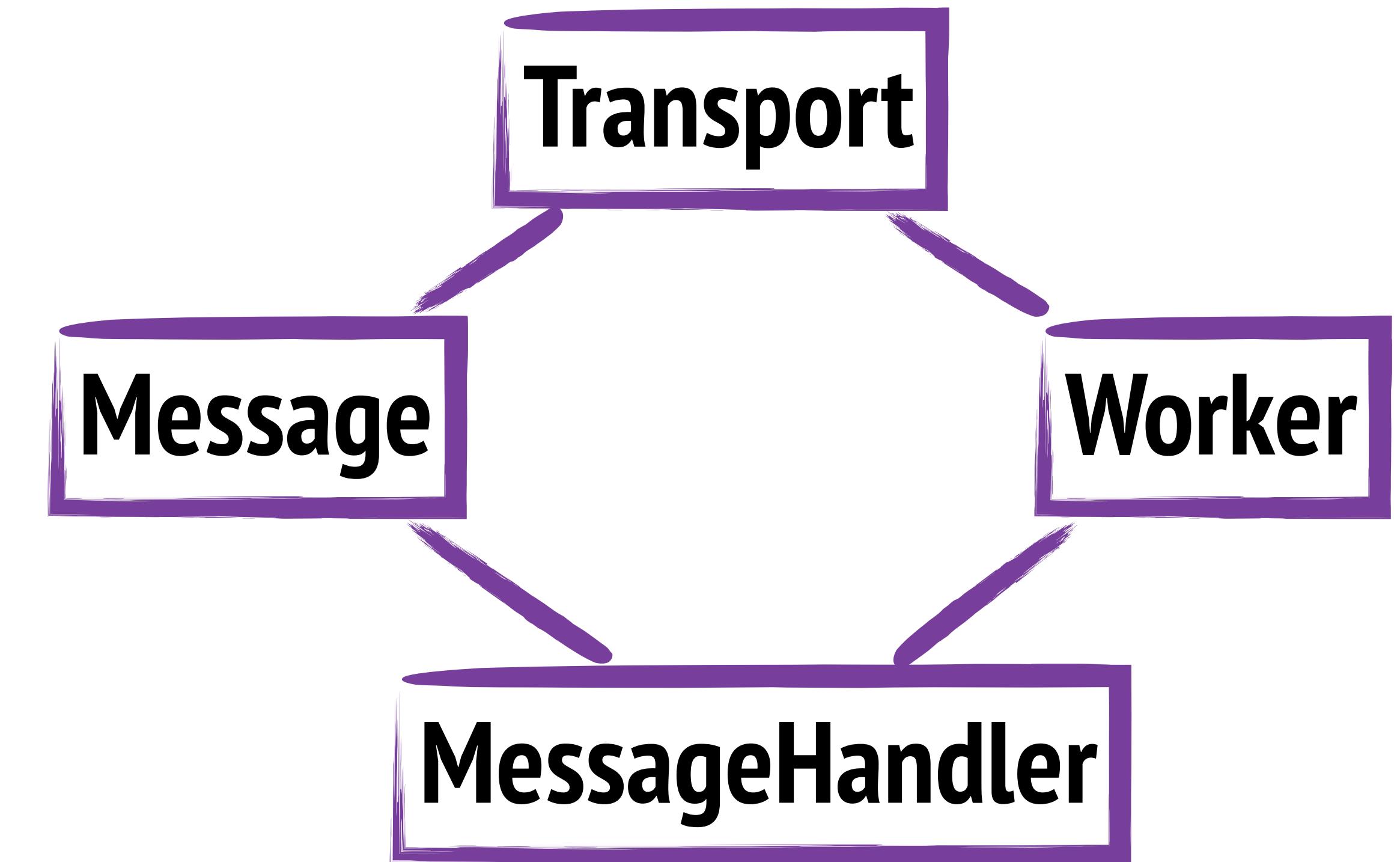
```
$messageBus→dispatch(  
    new MyMessage(/* ... */)  
)
```

```
class MyMessage  
{  
    // ...  
}
```

```
#[AsMessageHandler]  
class MyMessageHandler  
{  
    public function __invoke(MyMessage $message): void  
    {  
        // ...  
    }  
}
```

# Symfony Messenger: The Actors

- A **message handler** can handle one or multiple types of **messages**.
- A **message** is routed through a **transport**.
- A **worker** listens to one or multiple **transports**.
- The **worker** calls the **message handler**.



# Configuration

```
framework:  
    messenger:  
        transports:  
            async: '%env(MESSENGER_TRANSPORT_DSN)%'  
            failed: 'doctrine://default?queue_name=failed'  
            sync: 'sync://'  
  
        routing:  
            App\Message\ProcessOrder: async  
            Symfony\Component\Mailer\Messenger\SendEmailMessage: async  
  
        failure_transport: failed
```

# The worker

```
bin/console messenger:consume
```

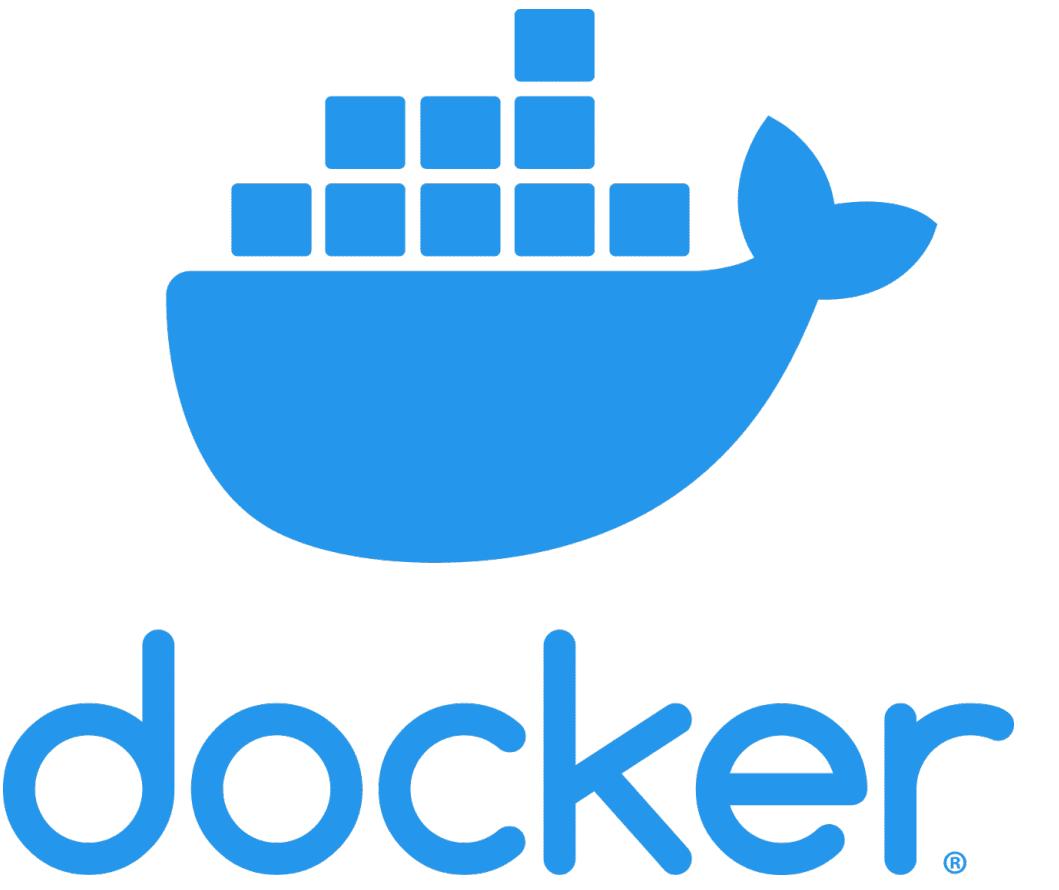
The worker is a long running PHP process that...

- receives messages,
- dispatches them to their corresponding handlers,
- monitors the success of the handling and
- shuts down after X messages/seconds

```
bin/console \
messenger:consume \
--limit=100 \
--time-limit=3600 \
my_high_prio_queue \
my_normal_queue \
my_low_prio_queue
```

# Running the worker

- Run multiple workers in parallel.
- The worker(s) will regularly shut down.
- A worker might crash.
- Keep workers running via SupervisorD, Docker, Kubernetes, ...
- Monitor your worker status!



# The failure queue

- Collects messages that could not be handled.
- Typically a database table.
- No message is lost!
- Allows for manual intervention.
- Monitor that queue!

# Limitations

Symfony Messenger is **not** a complete abstraction layer for message queues.

There are aspects that are explicitly not covered!

- Management of queued messages.
- Querying the status of a message (polling).
- Queue monitoring.

# Summary

- A message bus allows us to **execute logic asynchronously.**
- A web application can **delegate tasks into the background** that don't need immediate execution.
- Symfony Messenger **hides the complexity of queue handling** from your business logic.

# Thank you!

spam me:

[me@derrabus.de](mailto:me@derrabus.de)

follow me:

[@derrabus](https://twitter.com/derrabus)

hire me:

<https://derrabus.de>