Decoupling an Application with Symfony Messenger

ConFoo, Montreal, Canada - February 23th, 2024

David Buchmann - david@liip.ch

PHP Engineer, Liip SA, Switzerland

We built a data kraken

Product API
massive central index of product data

- Constantly evolving since 2013
- Collecting data from ~40 source systems
  Legacy and future sources, always in the process of migrating to new services
- Cache all data locally in MySQL for quick access
- Aggregate data in Elasticsearch

First approach

- Cronjob looping over all changed products
- Incremental imports: last-modified
- Record command runs and use the timestamp

- Does not scale
- Error during processing: Start over
- Update products many times for different things

Encapsulate Logic

Extract the logic out of commands and controllers.

=> Refactored into `Importer` classes that can handle a single item at a time.

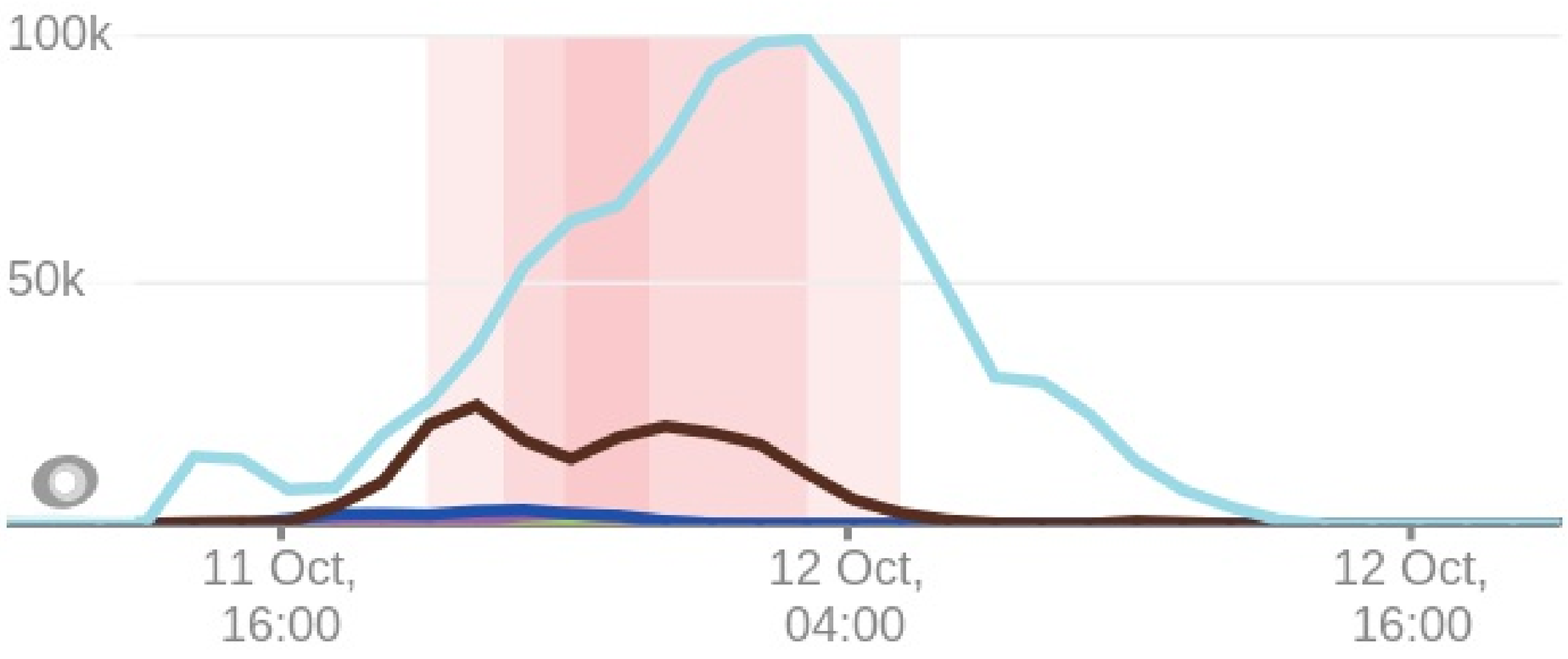
Decouple with message queue

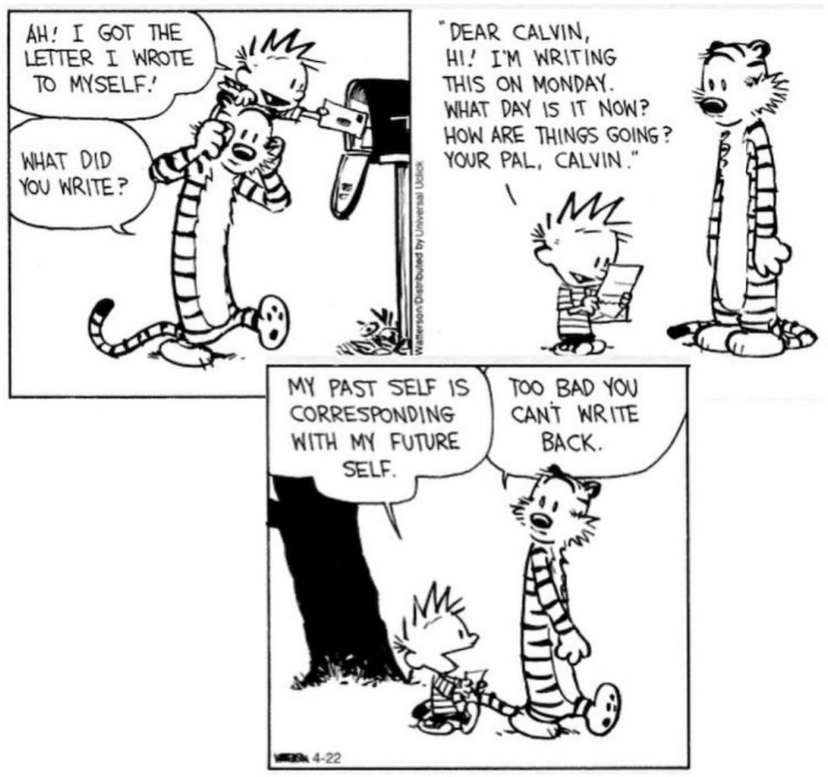
- `Importer` can be called from command line but also from message processor
- Importing triggers events on data changes
- Event listeners dispatch further messages as needed for next steps
- Perfect if source system can send messages
- Otherwise cronjob to only pull data and store it, then dispatch messages

Reaping the benefits

- Parallel processing of data
- Dynamically scale the number of workers
- Failure only affects single items, does not block
  - easy retry
  - abort after several failed attempts

	Messages			Message rates		
State	▼ **Ready**	**Unacked**	**Total**	**incoming**	**deliver / get**	**ack**
running	504,680	99	504,779	2.2/s	31/s	31/s
running	103,692	25	103,717	2.4/s	2.4/s	2.4/s




AH! I GOT THE LETTER I WROTE TO MYSELF!
WHAT DID YOU WRITE?
Watterson Distributed by Universal Uclick
"DEAR CALVIN, HI! I'M WRITING THIS ON MONDAY. WHAT DAY IS IT NOW? HOW ARE THINGS GOING? YOUR PAL, CALVIN."
MY PAST SELF IS CORRESPONDING WITH MY FUTURE SELF.
TOO BAD YOU CAN'T WRITE BACK.
4-22

Messages within the application

- Also split up heavy tasks in the application
- Multiple message-driven steps
- Easy to trigger the next step
- Using Symfony event system to decouple
- Event listeners can dispatch messages: e.g. we have 2 sources of product data, decide based on category which is relevant

e.g. Compile data into Elasticsearch

- SAP data must be processed as fast as possible, data delivery process is blocking in SAP
- Dispatch message with data as payload
- First consumer stores data locally
- And dispatches an event, which makes another listener queue this product for reindexing
- Different priorities to process urgent changes fast (e.g. prices)

Flexible on the outside, consistent on the inside

- Some source systems send messages on changes
- Other systems are polled for changes
- Daily import of CSV files with full data
- Temporal data with time limited validity
- We process everything as message


Any questions
so far?

Demo Time!

github.com/dbu/messenger-demo

https://md.davidbu.ch

Neuromancer, William Gibson

Symfony Messenger

ConFoo.CA

L//P

Event	Message
Blocking	Asynchronous
Can alter behaviour	No influence
Deterministic order	Undefined order (of consumers)
Not scalable	Scalable

ConFoo.CA

L//P

Message System Components

- Message (payload application specific)
- Sender (Publisher, Producer)
- Receiver (Subscriber, Consumer)
- Channels (Consume only relevant messages)

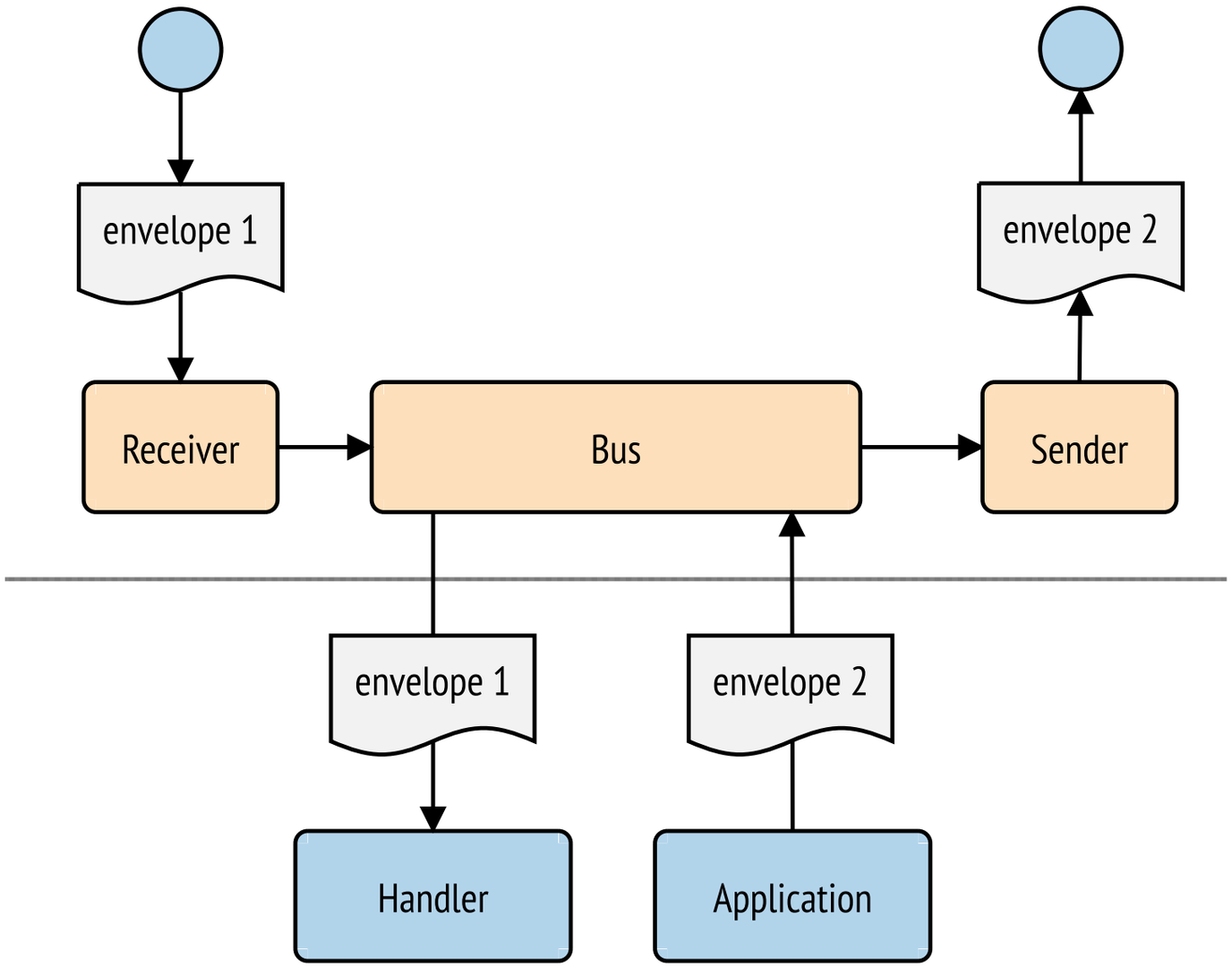
Message System Advantages

- Decoupled (mix systems or languages)
- Scalable (add more producers or consumers)
- Buffering/load management (Async)
- Resilient (buffered when receivers are down)
- Reliable (acknowledgement and retry)
- Order (sequential if only 1 instance)
- Extensions (e.g. message priorities)

Why Symfony Messenger?

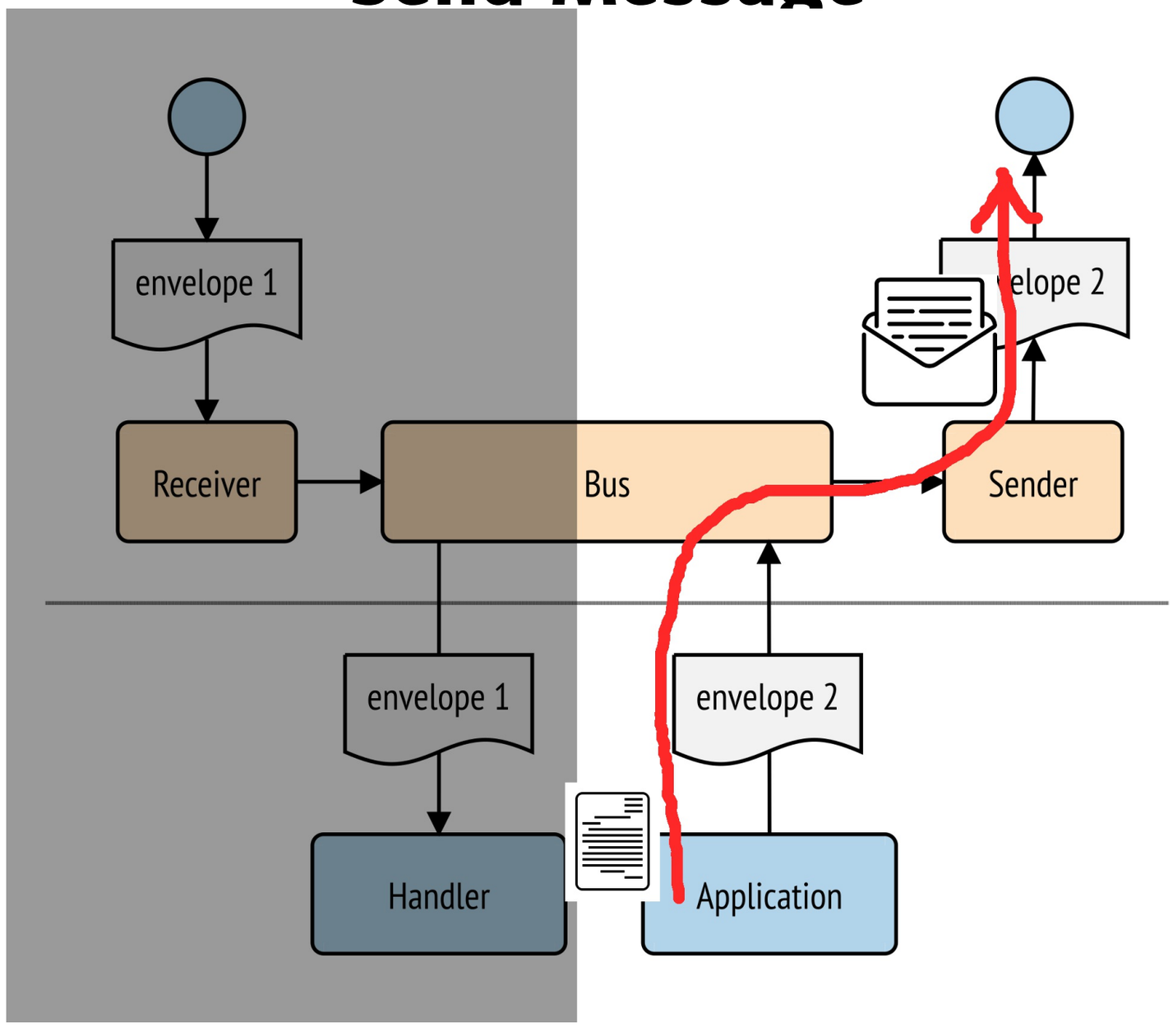
- High level functionality
- Abstract from specific message transport
- Nice and modern architecture

The flow of Symfony Messenger

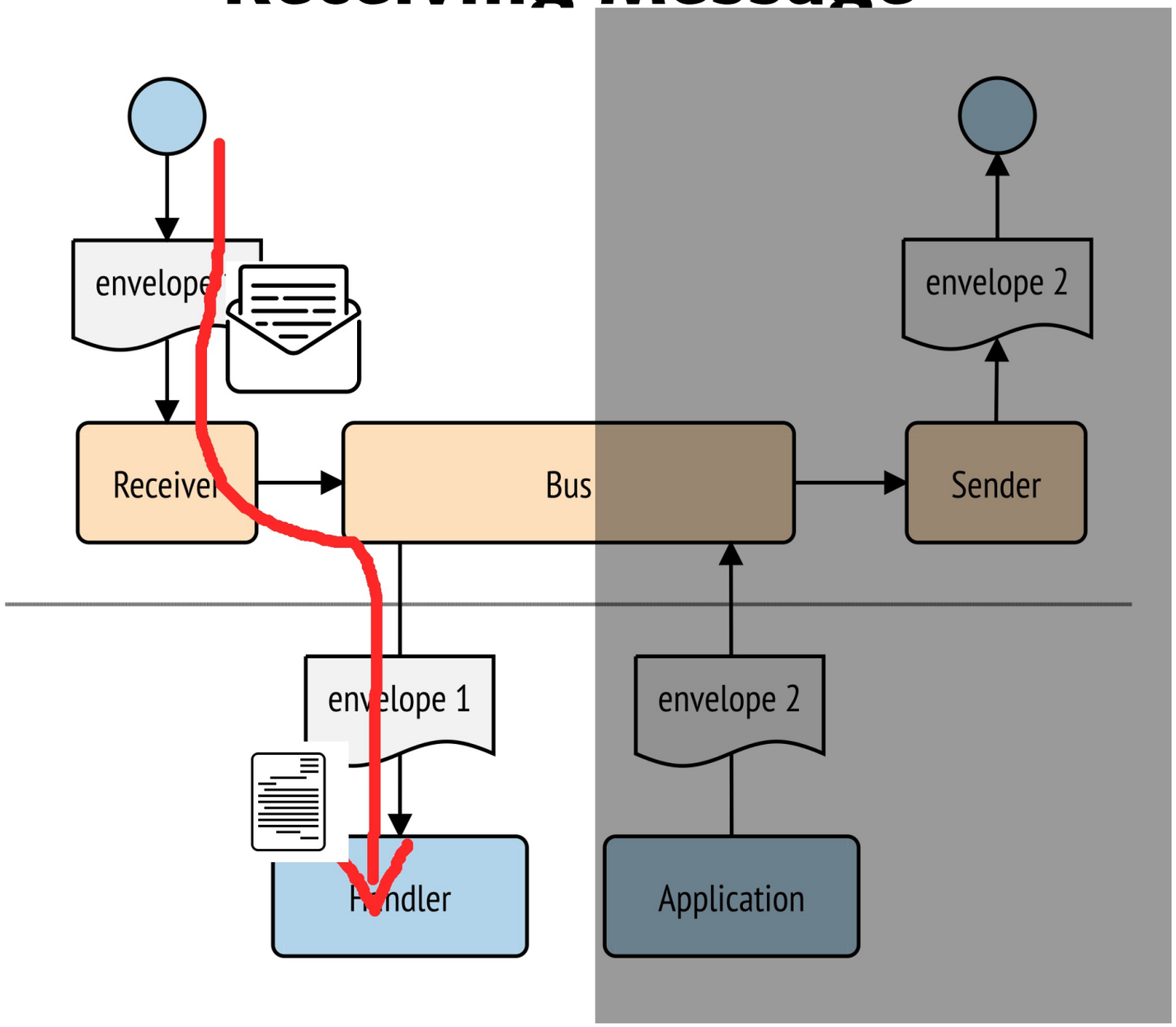


https://symfony.com/doc/current/components/messenger.html#concepts

Send Message



Receiving Message



Message

- This is your class. It is serialized by the messenger.
- Small (Entity ID rather than the full entity data)
- Specific

```php
class UpdatePromotion
{
    public function __construct(
        public string $id
    ) {}
}
```

Sending a message

```
1 private MessageBusInterface $bus;
2 ...
3 $this->bus->dispatch(new UpdatePromotion($id));
```

Stamps (Metadata)

```php
$stamps[] = new DelayStamp(20);
$this->bus->dispatch(
    new UpdatePromotion($id),
    $stamps
);
```

Routing

- What queue for which message?
- Control which workers to scale
- Better monitoring

```
framework:
  messenger:
    transports:
      neuromancer:
        dsn: '%env(MESSENGER_TRANSPORT_DSN)%&queue_name=neuromancer'

    routing:
      App\Message\NeuromancerMessage: neuromancer
```

Retry

- In a large system, things will go wrong
  - Software crashes, maschines reboot
  - Network issues
  - Service or system not available
  - Actual logic bugs
- Message goes back to queue
- Exponential backoff
- Failure transport

Keep workers running

- Long-running PHP processes
- Careful about memory leaks.
- supervisord or similar to restart
  - Rejuvenation: Restart after number of messsages, time or when memory consumption exceeds threshold
  - Restart workers on deployment

Autoscaler

- Own service to start and stop workers
- Application registers its workers with autoscaler
- Autoscaler monitors queue sizes
- Spawns workers up to the limit when queue grows

What is going on?

- Copy all messages to an archive queue with a TTL
- Import state log table, track all steps

Message Transport

- Without a configured transport, messages are processed immediately
- Doctrine (using a database table)
- AMQP with ext-amqp (rabbitmq, kafka)
- Redis with ext-redis
- Amazon SQS
- Plug your own, e.g. to call a custom API

Did i hear «microservices»?

ConFoo.CA

L//P

Une Bataille, ~1750, François-Joseph Casanova

Messaging != microservices

- Our application has ~30 types of message and their processors
- Messages and processors all in the same application
- I would not want to maintain 30 tiny Symfony applications just for one single worker each
- Some separate applications that have nothing directly to do with the main data api

Thank you!

github.com/dbu/messenger-demo/

@dbu@phpc.social

David Buchmann, Liip SA