Unleash the Power of Symfony Messenger

Kris Wallsmith

SymfonyCon Amsterdam 2025

Jeremy Mikola

Ryan Weaver

1984-2025

Unleash the Power of Symfony Messenger

Kris Wallsmith

SymfonyCon Amsterdam 2025

About Me

- VP of Engineering at Vacatia

Paris, 2015

About Me

- VP of Engineering at Vacatia
- Live in Portland, Oregon

Portland, 2013

About Me

- VP of Engineering at Vacatia
- Live in Portland, Oregon
- CTO at Pickathon Music Festival

San Francisco, 2011

About Me

- VP of Engineering at Vacatia
- Live in Portland, Oregon
- CTO at Pickathon Music Festival
- Former Symfony Core Team

Paris, 2010

Today

- Vacatia 101
- Symfony Messenger 101
- Cool stuff

Vacatia

Vacatia

TECH STACK

Reservations

Vacatia

TECH STACK

Reservations

salesforce

Owners & Contracts

Vacatia

TECH STACK

Reservations

salesforce

Owners & Contracts

Billing

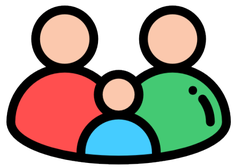
Vacatia

TECH STACK

Reservations

Owners & Contracts

Billing



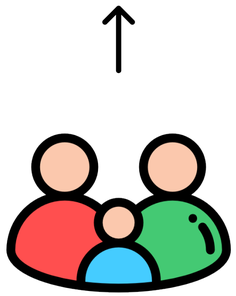
Vacatia

TECH STACK

Reservations

Owners & Contracts

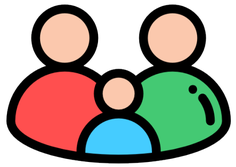
Billing



Vacatia

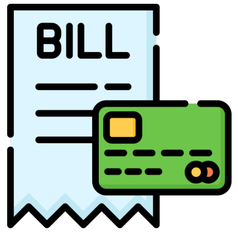
TECH STACK

Reservations



Owners & Contracts

Billing



Vacatia

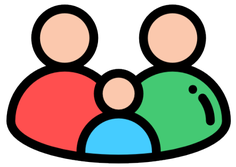
TECH STACK

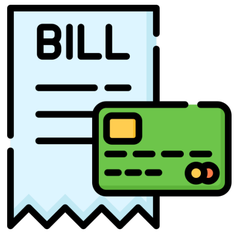
Reservations

salesforce

Owners & Contracts

Billing





Vacatia

TECH STACK

salesforce

Owners & Contracts

Reservations

Billing

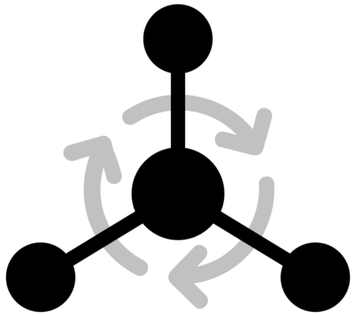
Vacatia

TECH STACK

salesforce

Owners & Contracts

Reservations

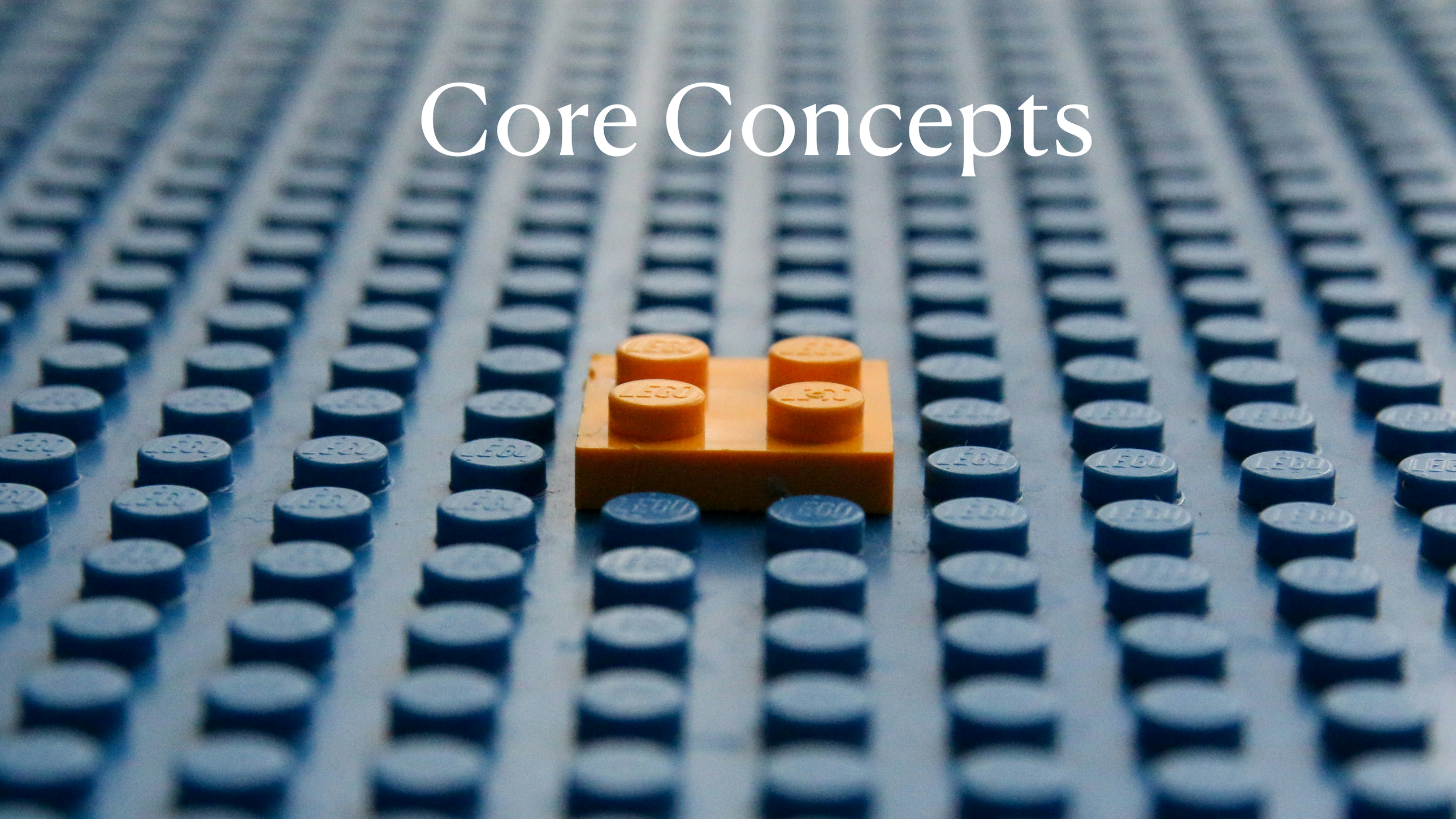


Integration Layer

Billing

Symfony Messenger

Core Concepts



Message Bus

“A bus … is a communication system that transfers data between components”

Wikipedia: Bus (computing)

```php
$envelope = $bus->dispatch($envelope);
```

Envelopes

```php
class Envelope
{
    function getMessage(): object;

    function with(StampInterface ...$stamps): static;
    function last(string $stampClass): ?StampInterface;

    /** @return StampInterface[]|StampInterface[][] */
    function all(?string $stampClass = null): array;

    // ...
}
```

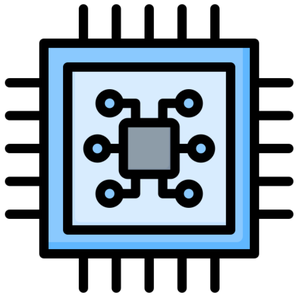
```php
/** Stamps must be serializable value objects for transport. */
interface StampInterface {}
```

Core Concepts

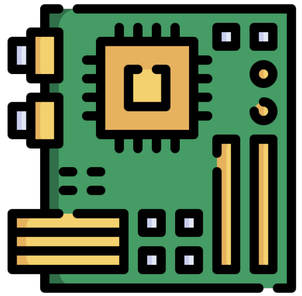
- Message Bus
- Envelopes
- Messages
- Stamps
- ...

What happens inside the bus?

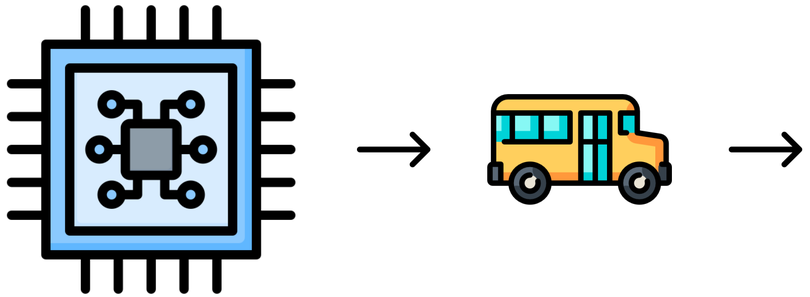
Middleware



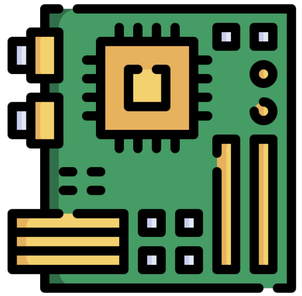
Component A



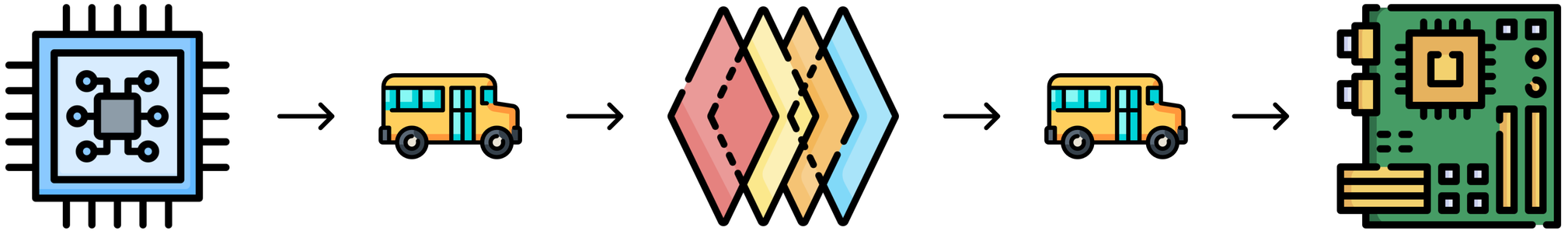
Component B



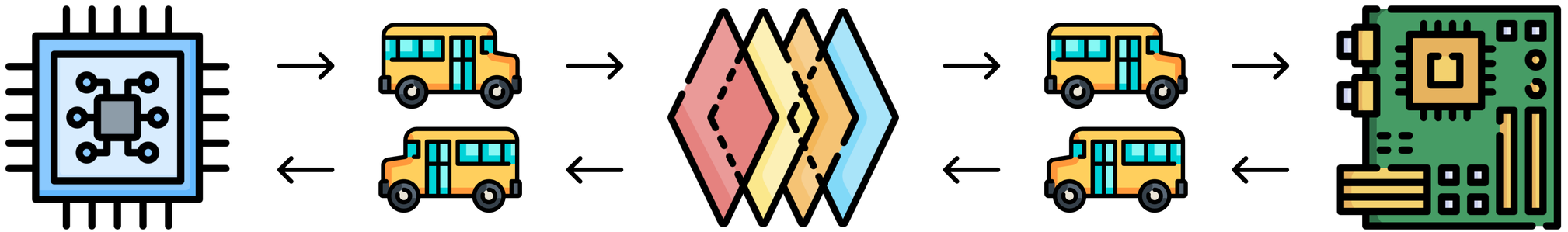
Component A



Component B



Component A Middleware Component B



Component A Middleware Component B

```php
interface MiddlewareInterface
{
    function handle(
        Envelope $envelope,
        StackInterface $stack,
    ): Envelope;
}
```

```
interface StackInterface
{
    function next(): MiddlewareInterface;
}
```

```php
class MyMiddleware implements MiddlewareInterface
{
    public function handle(
        Envelope $envelope,
        StackInterface $stack,
    ): Envelope {
        return $envelope;
    }
}
```

```php
class MyMiddleware implements MiddlewareInterface
{
    public function handle(
        Envelope $envelope,
        StackInterface $stack,
    ): Envelope {
        return $stack->next()->handle(
            $envelope,
            $stack,
        );
    }
}
```

```php
public function handle(Envelope $envelope, StackInterface $stack): Envelope
{
    // before
    try {
        $envelope = $stack->next()->handle($envelope, $stack);
    } catch (Throwable $e) {
        // on error
        throw $e;
    } finally {
        // always
    }

    // on success

    return $envelope;
}
```

Core Middleware

- SendMessageMiddleware
- HandleMessageMiddleware

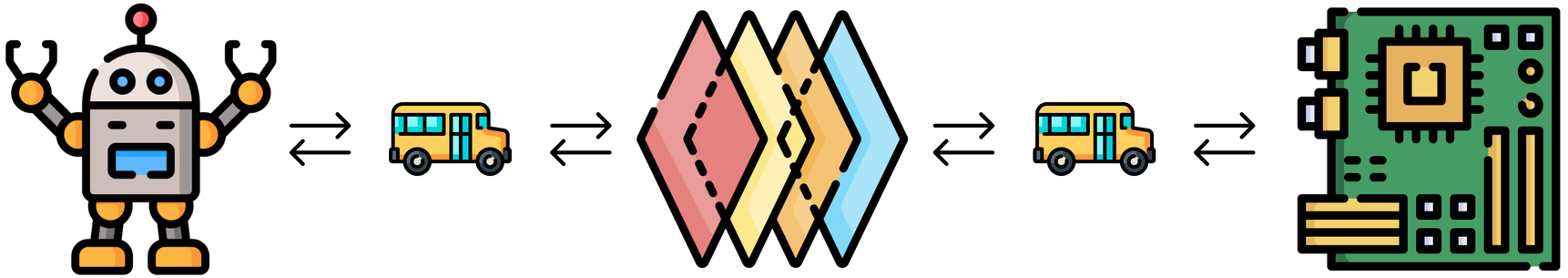
Transports

↓ Send

Transport

↓ Receive

Worker



Core Concepts

- Message Bus
- Envelopes
- Messages
- Stamps
- Middleware
- Transports (send & receive)
- Workers
- Handlers


Unleashed!

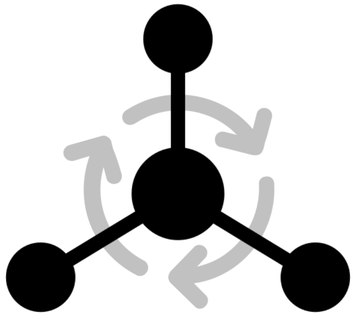
Vacatia

TECH STACK

salesforce

Owners & Contracts

Reservations



Integration Layer

Billing

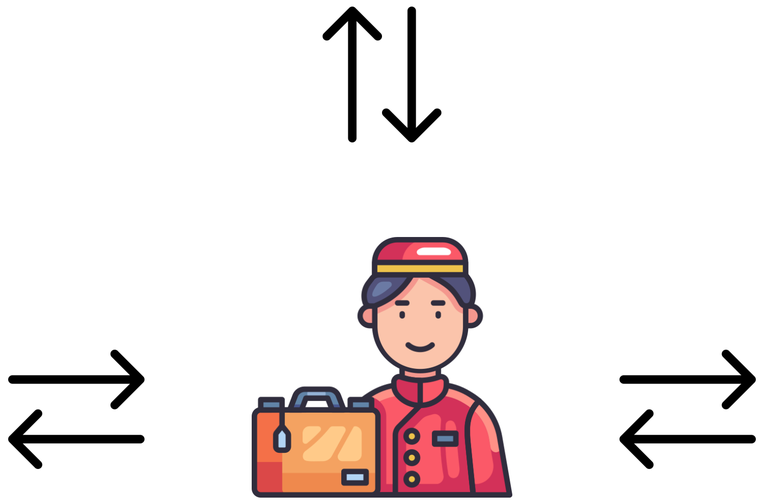
Vacatia

TECH STACK

salesforce

Owners & Contracts

Reservations



“Porter”

Billing

```php
readonly class IdStamp implements StampInterface
{
    public function __construct(
        public Ulid $id = new Ulid(),
    ) {}
}
```

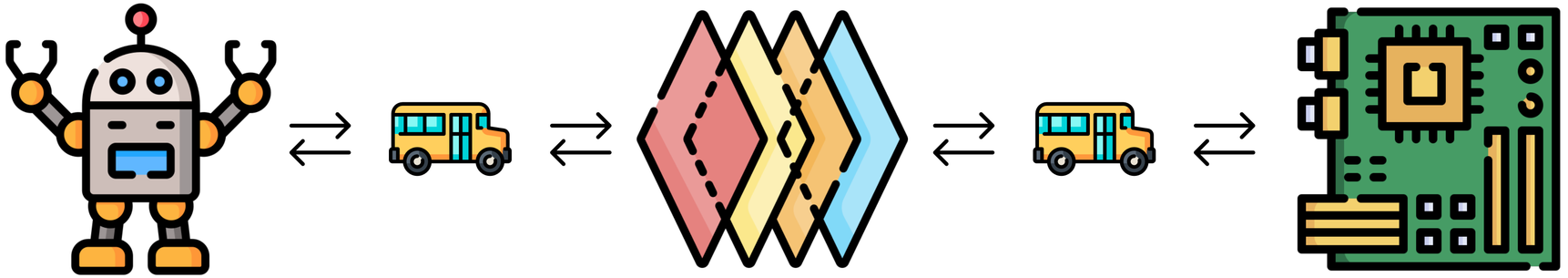
```php
class IdMiddleware implements MiddlewareInterface
{
    public function handle(
        Envelope $envelope,
        StackInterface $stack,
    ): Envelope {
        $envelope = $envelope->with(new IdStamp());

        return $stack->next()->handle($envelope, $stack);
    }
}
```

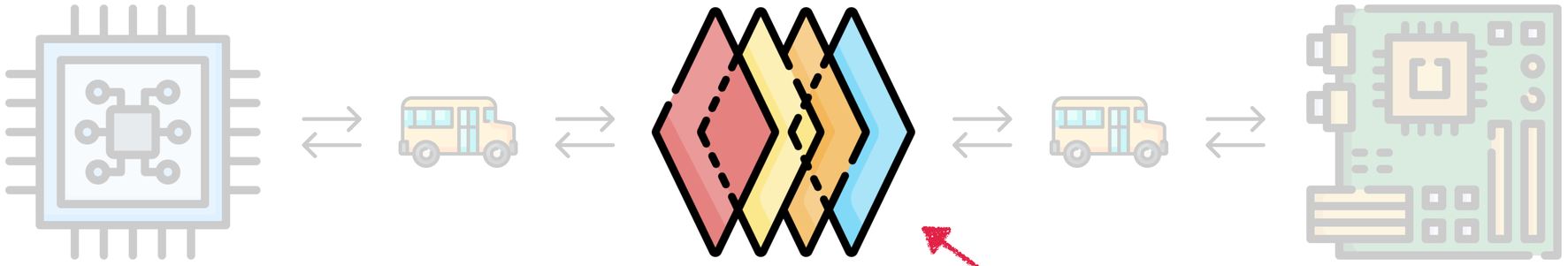
↓ Send

Transport

↓ Receive

Worker





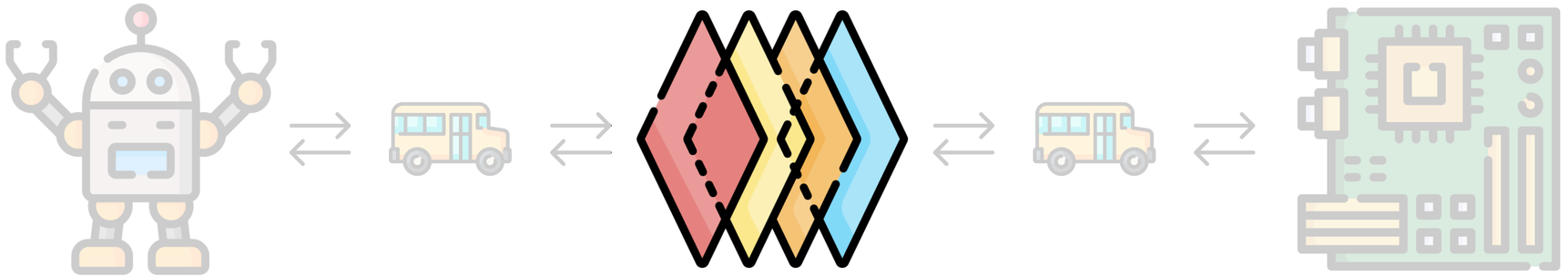
Send

Transport

Envelopes can pass through middleware multiple times

Receive

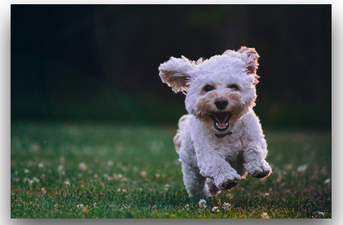
Worker



```php
class IdMiddleware implements MiddlewareInterface
{
    public function handle(
        Envelope $envelope,
        StackInterface $stack,
    ): Envelope {
        $envelope = $envelope->with(new IdStamp());

        return $stack->next()->handle($envelope, $stack);
    }
}
```

```php
class IdMiddleware implements MiddlewareInterface
{
    public function handle(
        Envelope $envelope,
        StackInterface $stack,
    ): Envelope {
        if (!$envelope->last(IdStamp::class)) {
            $envelope = $envelope->with(new IdStamp());
        }

        return $stack->next()->handle($envelope, $stack);
    }
}
```

Now What?



Logging

```php
class IdMiddleware implements MiddlewareInterface
{
    public function __construct(
        private IdContext $context,
    ) {}

    public function handle(Envelope $envelope, StackInterface $stack): Envelope
    {
        if (!$stamp = $envelope->last(IdStamp::class)) {
            $envelope = $envelope->with($stamp = new IdStamp());
        }

        $this->context->push($stamp->id);
        try {
            return $stack->next()->handle($envelope, $stack);
        } finally {
            $this->context->pop();
        }
    }
}
```

```php
class IdProcessor implements ProcessorInterface
{
    public function __construct(
        private IdContext $context,
    ) {}

    public function __invoke(LogRecord $record): LogRecord
    {
        if ($id = $this->context->head()) {
            $record['extra']['message_id'] = "$id";
        }

        return $record;
    }
}
```

```php
public function handle(Envelope $envelope, StackInterface $stack): Envelope
{
    if (!$stamp = $envelope->last(IdStamp::class)) {
        $envelope = $envelope->with($stamp = new IdStamp());
        $this->logDispatched($envelope);
    }

    $this->context->push($stamp->id);
    try {
        $envelope = $stack->next()->handle($envelope, $stack);
        $envelope->last(HandledStamp::class) && $this->logHandled($envelope);
        return $envelope;
    } catch (Throwable $e) {
        $this->logFailed($envelope, $e);
        throw $e;
    } finally {
        $this->context->pop();
    }
}
```

Nested Messages

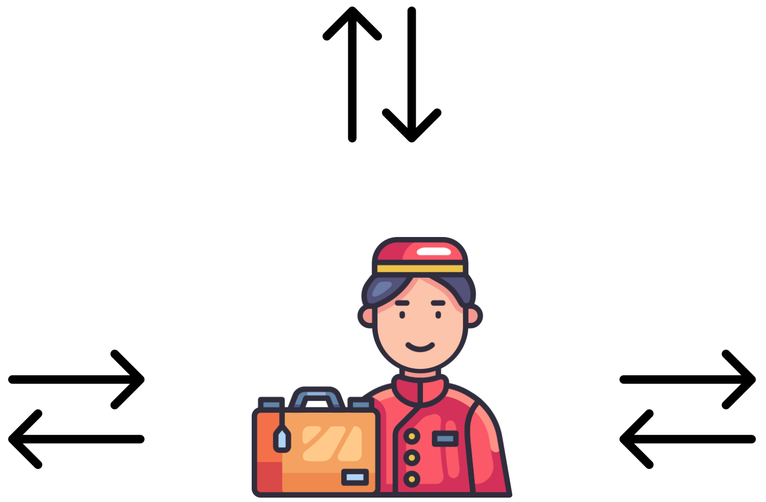
Vacatia

TECH STACK

salesforce

Owners & Contracts

Reservations



“Porter”

Billing

```php
class Envelope
{
    function with(StampInterface ...$stamps): static;
    function last(string $stampClass): ?StampInterface;

    /** @return StampInterface[]|StampInterface[][] */
    function all(?string $stampClass = null): array;

    // ...
}
```

```php
public function handle(Envelope $envelope, StackInterface $stack): Envelope
{
    if (!$envelope->last(IdStamp::class)) {
        $envelope = $envelope->with(...array_map(
            idToStamp(...),
            $this->context->all(),
        ))->with(new IdStamp());

        $this->logDispatched($envelope);
    }

    $this->context->push(array_map(
        stampToId(...),
        $envelope->all(IdStamp::class),
    ));

    //...
}
```

```
CREATE TABLE messenger_log (
    id CHAR(26) NOT NULL,
    path JSON NOT NULL,
    headers JSON,
    body TEXT,
    dispatched_at DATETIME NOT NULL,
    handled_at DATETIME,
    failed_at DATETIME,
    error TEXT,
    PRIMARY KEY (id)
)
```

Message Chains

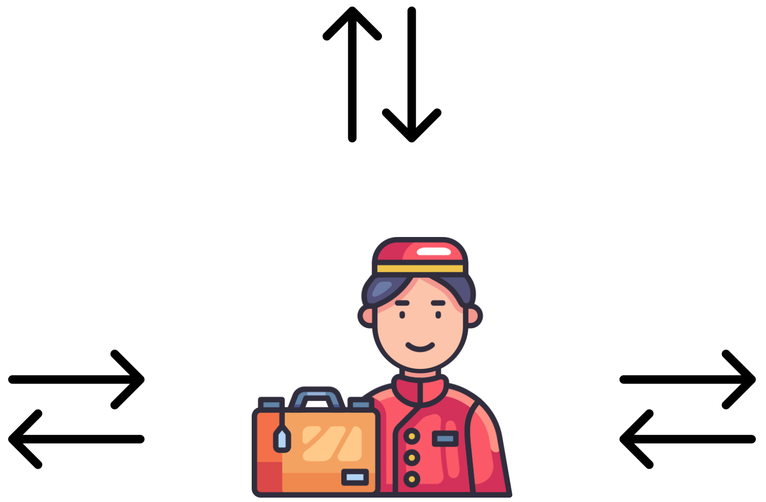
Vacatia

TECH STACK

salesforce

Owners & Contracts

Reservations



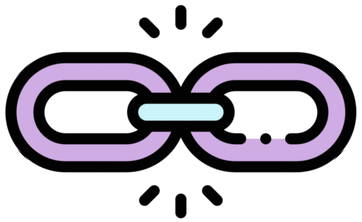
“Porter”

Billing

Vacatia

TECH STACK

Create Reservation



Update Reservation

Update Guest

Message Sequencing

```php
class ChainStamp implements StampInterface
{
    public function __construct(
        public Envelope $envelope,
    ) {}
}
```

```php
class ChainMiddleware implements MiddlewareInterface
{
    public function handle(Envelope $envelope, StackInterface $stack): Envelope
    {
        $envelope = $stack->next()->handle($envelope, $stack);
        if (!$envelope->last(HandledStamp::class)) {
            return $envelope;
        }

        foreach ($envelope->all(ChainStamp::class) as $chainStamp) {
            $this->bus->dispatch($chainStamp->envelope);
        }

        return $envelope->withoutAll(ChainStamp::class);
    }
}
```

```php
$bus->dispatch($createReservation, [
    new ChainStamp($sendConfirmation),
]);
```

Message Dependencies

```php
interface ChainInterface
{
    function chain(
        object $prevMessage,
        mixed $prevResult,
    ): ?object;
}
```

```php
class UpdateReservation implements ChainInterface
{
    public string $reservationId;

    public function __construct(
        public array $updates,
    ) {}

    public function chain(object $prevMessage, mixed $prevResult): ?object
    {
        $this->reservationId = $prevResult['id'];

        return $this;
    }
}
```

```php
$bus->dispatch($createReservation, [
    new ChainStamp($updateReservation),
]);
```

```php
public function handle(
    Envelope $envelope,
    StackInterface $stack,
): Envelope {
    // ...

    foreach ($this->nextEnvelopes(
        $envelope,
        $handledStamp,
    ) as $nextEnvelope) {
        $this->bus->dispatch($nextEnvelope);
    }

    // ...
}
```

```php
private function nextEnvelopes(
    Envelope $envelope,
    HandledStamp $handledStamp,
): iterable {
    foreach ($envelope->all(ChainStamp::class) as $chainStamp) {
        $nextMessage = $chainStamp->envelope->getMessage();

        if (!$nextMessage instanceof ChainInterface) {
            yield $chainStamp->envelope;
        } elseif ($nextMessage = $nextMessage->chain(
            $envelope->getMessage(),
            $handledStamp->getResult(),
        )) {
            yield Envelope::wrap($nextMessage);
        }
    }
}
```

Unleash the Power of Symfony Messenger

- Message Bus
- Envelopes
- Messages
- Stamps
- Middleware
- Transports (send & receive)
- Workers
- Handlers
- Message IDs
- Message Log
- Nested Messages
- Message Sequencing
- Message Dependencies

Thank you, Symfony!