Testable Database Interaction

Sebastian Bergmann

- **Sebastian** since 1978
- **Computers** since 1990
- **PHP** since 1998
- **Open Source** since 1998
- **PHPUnit** since 2000
- **thePHP.cc** since 2009

thePHP.cc

sharing experience

Expectation Management

I will not cover techniques specific to Doctrine DBAL, Doctrine ORM, Eloquent, or similar standard solutions.

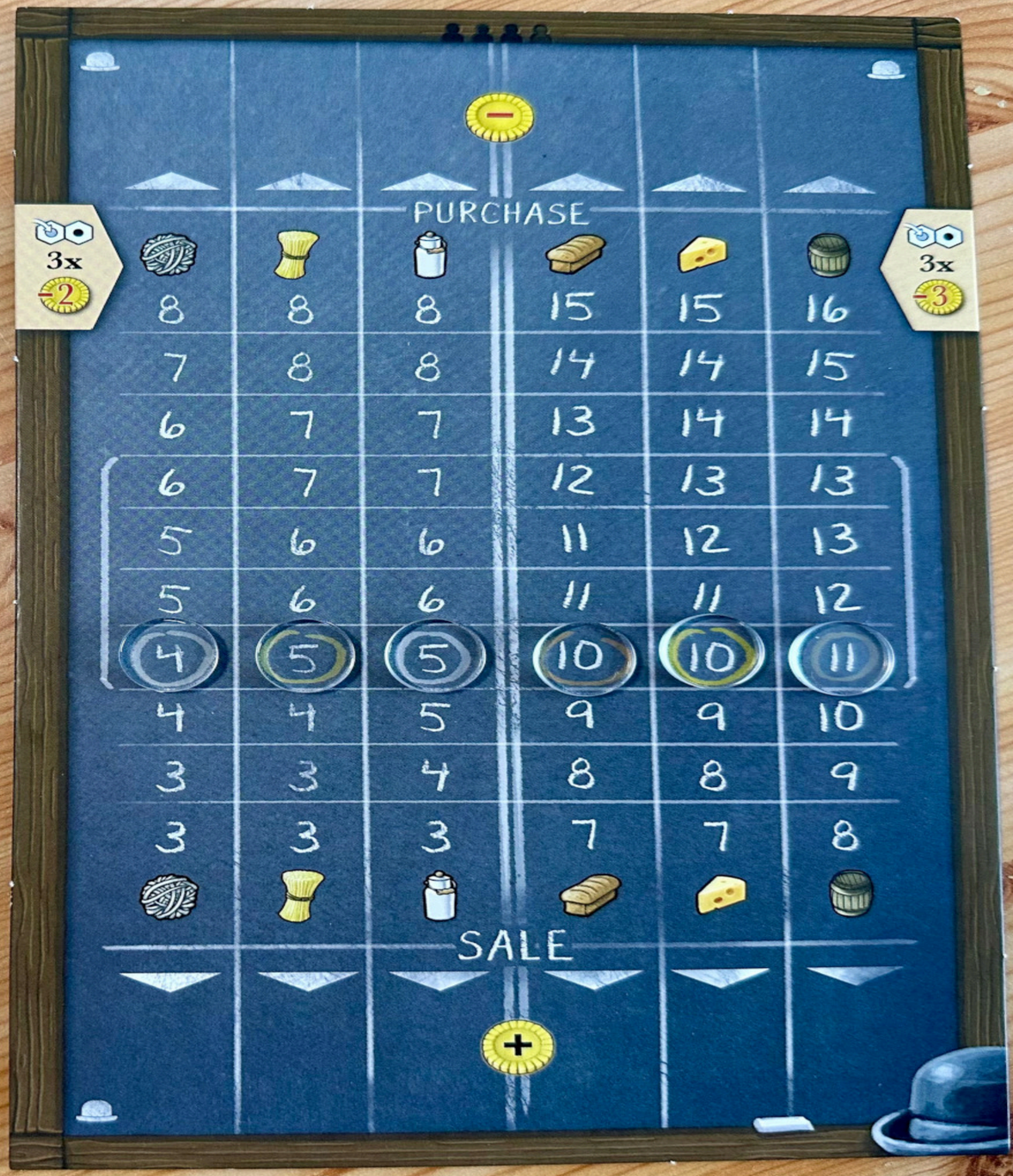
I will not cover optimisations such as running a relational database from a RAM disk, or using containerisation tricks to improve the performance of tests that interact with the database.

You will see **best practices** and **design patterns** in action for interacting with your database.

You will learn how to **implement testable database interaction** – and how to **test code that interacts with the database** without getting frustrated.

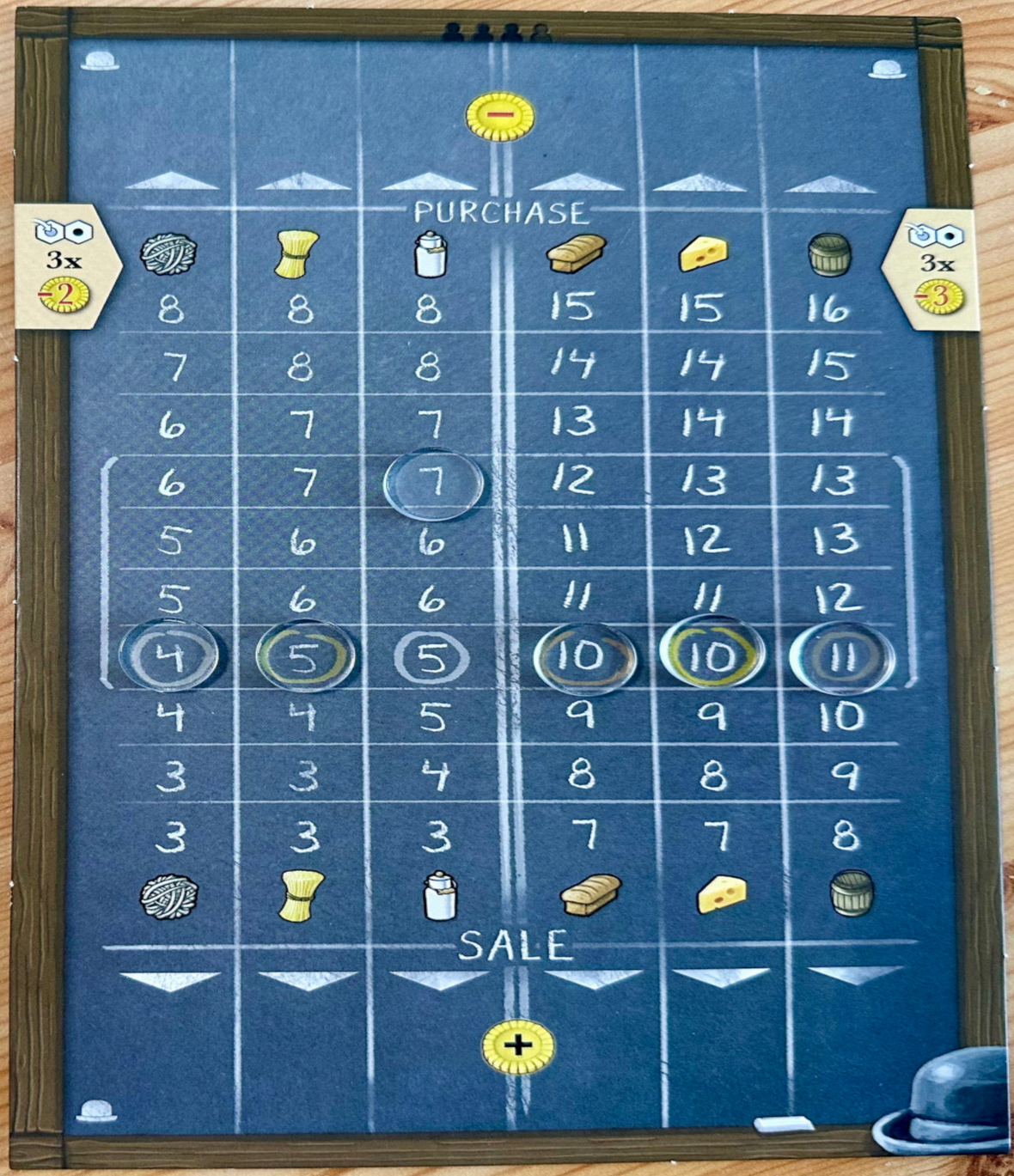
We focus on **concepts**, **best practices**, and **design patterns** rather than off-the-shelf frameworks and libraries.

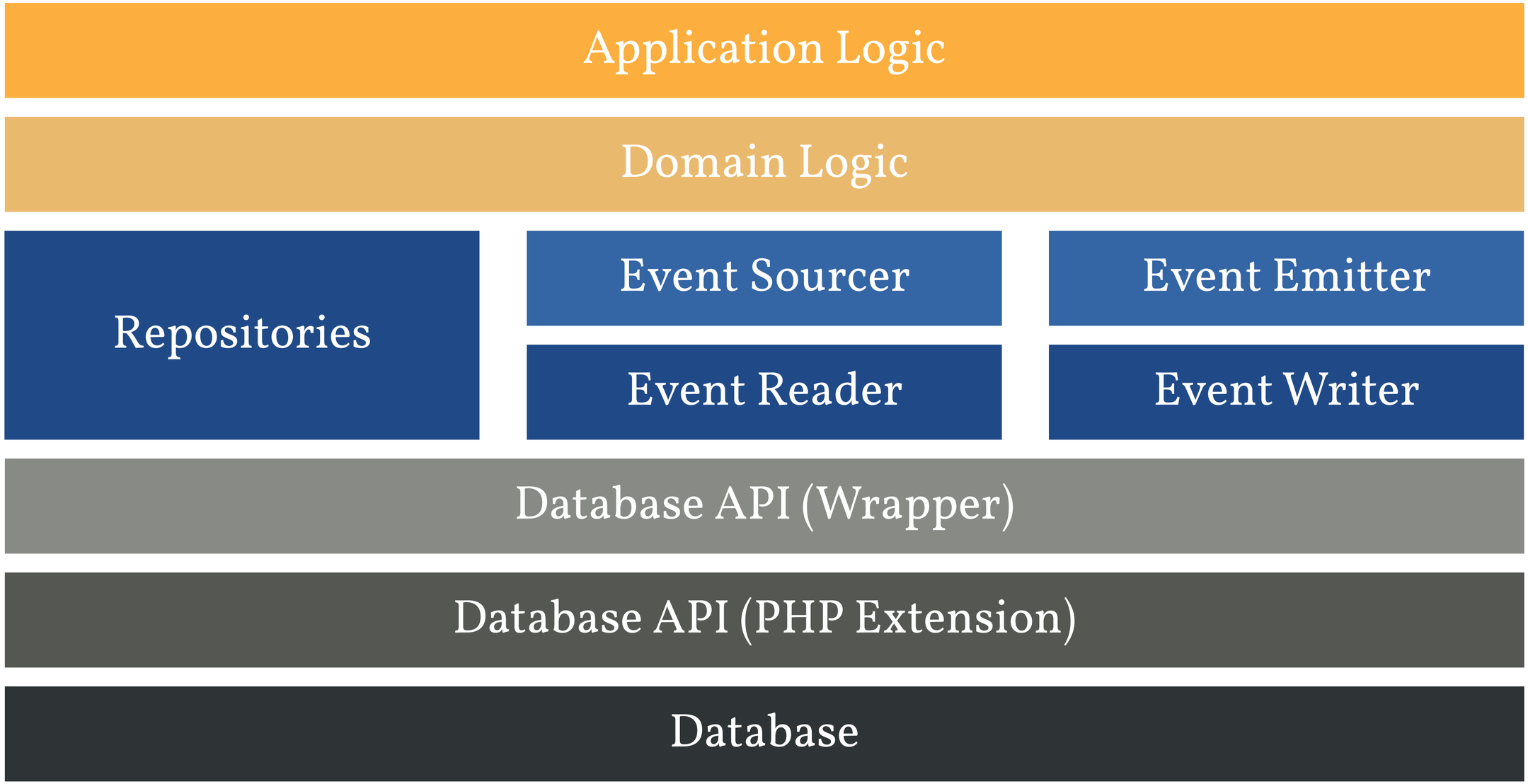
Once you understand these, you can apply them to your preferred framework or library with confidence and ease.


PURCHASE
3x
-2
3x
-3
8 8 8 15 15 16
7 8 8 14 14 15
6 7 7 13 14 14
6 7 7 12 13 13
5 6 6 11 12 13
5 6 6 11 11 12
4 5 5 10 10 11
4 4 5 9 9 10
3 3 4 8 8 9
3 3 3 7 7 8
SALE


PURCHASE
3x
-2
3x
-3
8 8 8 15 15 16
7 8 8 14 14 15
6 7 7 13 14 14
6 7 7 12 13 13
5 6 6 11 12 13
5 6 6 11 11 12
4 5 5 10 10 11
4 4 5 9 9 10
3 3 4 8 8 9
3 3 3 7 7 8
SALE


PURCHASE
3x
-2
3x
-3
8 8 8 15 15 16
7 8 8 14 14 15
6 7 7 13 14 14
6 7 7 12 13 13
5 6 6 11 12 13
5 6 6 11 11 12
4 5 5 10 10 11
4 4 5 9 9 10
3 3 4 8 8 9
3 3 3 7 7 8
SALE


PURCHASE
3x
-2
3x
-3
8 8 8 15 15 16
7 8 8 14 14 15
6 7 7 13 14 14
6 7 7 12 13 13
5 6 6 11 12 13
5 6 6 11 11 12
4 5 5 10 10 11
4 4 5 9 9 10
3 3 4 8 8 9
3 3 3 7 7 8
SALE


Application Logic
Domain Logic
Repositories
Event Sourcer
Event Emitter
Event Reader
Event Writer
Database API (Wrapper)
Database API (PHP Extension)
Database

```php
<?php declare(strict_types=1);
namespace example\framework\http;

final readonly class Kernel
{
    private PostRequestRouter $postRequestRouter;

    public function __construct(PostRequestRouter $postRequestRouter, /* ... */)
    {
        $this->getRequestRouter  = $getRequestRouter;
        $this->postRequestRouter = $postRequestRouter;
    }

    public function run(Request $request): Response
    {
        if ($request->isGetRequest()) { /* ... * / }

        if ($request->isPostRequest()) {
            $command = $this->postRequestRouter->route($request);

            return $command->execute();
        }

        // ...
    }
}
```

thePHP.cc

```php
<?php declare(strict_types=1);
namespace example\framework\http;

final readonly class PostRequestRouter
{
    /** @var list<PostRequestRoute> */
    private array $routes;

    public function __construct(PostRequestRoute ...$routes)
    {
        $this->routes = $routes;
    }

    /** @throws RequestCannotBeRoutedException */
    public function route(PostRequest $request): Command
    {
        foreach ($this->routes as $route) {
            $command = $route->route($request);

            if ($command !== false) {
                return $command;
            }
        }

        throw new RequestCannotBeRoutedException;
    }
}
```

```php
<?php declare(strict_types=1);
namespace example\caledonia\application;

use example\caledonia\domain\PurchaseGoodCommand as DomainCommand;
// ...

final readonly class PurchaseGoodRoute implements PostRequestRoute
{
    // properties, constructor

    public function route(PostRequest $request): false|PurchaseGoodCommand
    {
        if ($request->uri() !== '/purchase-good') {
            return false;
        }

        $data = json_decode($request->body(), true);

        return new PurchaseGoodCommand(
            $this->factory->createPurchaseGoodCommandProcessor(),
            new DomainCommand(
                Good::fromString($data['good']),
                (int) $data['amount'],
            ),
        );
    }
}
```

```php
<?php declare(strict_types=1);
namespace example\caledonia\application;

use example\caledonia\domain\PurchaseGoodCommand as DomainCommand;
use example\framework\http\Command;
use example\framework\http\Response;

final readonly class PurchaseCommand implements Command
{
    private PurchaseGoodCommandProcessor $processor;
    private DomainCommand $command;

    public function __construct(PurchaseGoodCommandProcessor $processor, DomainCommand $command)
    {
        $this->processor = $processor;
        $this->command   = $command;
    }

    public function execute(): Response
    {
        $this->processor->process($this->command);

        return new Response;
    }
}
```

```php
<?php declare(strict_types=1);
namespace example\caledonia\application;

use example\caledonia\domain\PurchaseGoodCommand;

final readonly class ProcessingPurchaseGoodCommandProcessor implements PurchaseGoodCommandProcessor
{
    // properties, constructor

    public function process(PurchaseGoodCommand $command): void
    {
        $market = $this->sourcer->source();

        $oldPrice = $market->priceFor($command->good());

        $market = $market->purchase($command->good(), $command->amount());

        $this->emitter->goodPurchased($command->good(), $oldPrice, $command->amount());

        $newPrice = $market->priceFor($command->good());

        if (!$newPrice->equals($oldPrice)) {
            $this->emitter->priceChanged($command->good(), $oldPrice, $newPrice);
        }
    }
}
```

thePHP.cc

```php
<?php declare(strict_types=1);
namespace example\caledonia\application;

// use statements

final readonly class MarketEventSourcer implements MarketSourcer
{
    // properties, constructor

    public function source(): Market
    {
        $market = Market::create();

        foreach ($this->reader->topic('market.good-purchased', 'market.good-sold') as $event) {
           if ($event instanceof GoodPurchasedEvent) {
                $market = $market->purchase($event->good(), $event->amount());

                continue;
            }

            $market = $market->sell($event->good(), $event->amount());
        }

        return $market;
    }
}
```

```php
<?php declare(strict_types=1);
namespace example\framework\event;

use SebastianBergmann\MysqliWrapper\ReadingDatabaseConnection;

final readonly class DatabaseEventReader implements EventReader
{
    private ReadingDatabaseConnection $connection;
    private EventJsonMapper $mapper;

    public function __construct(ReadingDatabaseConnection $connection, EventJsonMapper $mapper)
    {
        $this->connection = $connection;
        $this->mapper     = $mapper;
    }

   public function topic(string ...$topics): EventCollection
    {
        $result = (new ReadEventsStatement($topics))->execute($this->connection);
        $events = [];

        foreach ($result as $row) {
            $events[] = $this->mapper->fromJson($row['payload']);
        }

        return EventCollection::fromArray($events);
    }
}
```

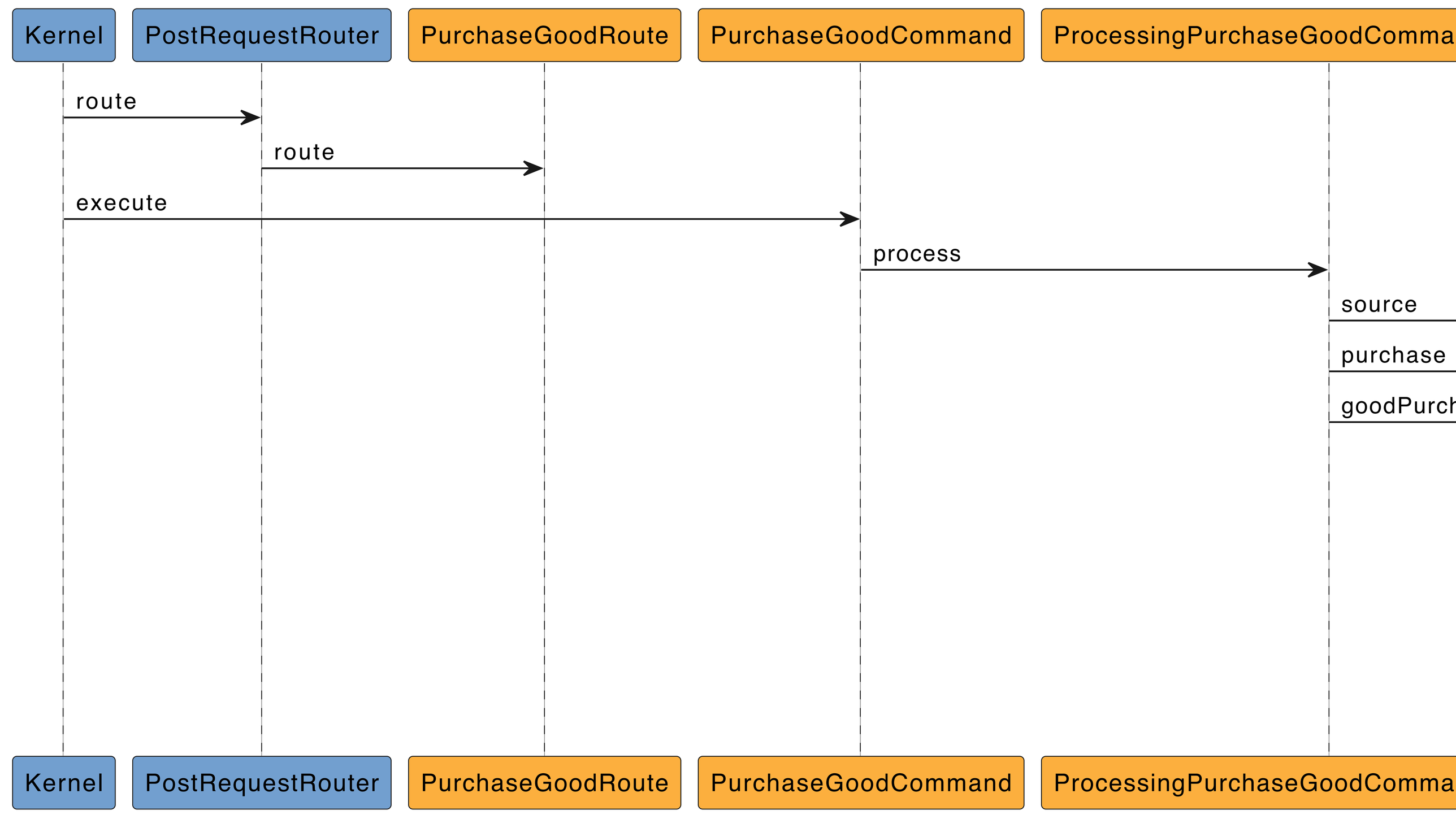
```php
<?php declare(strict_types=1);
namespace example\framework\event;

// use statements

final readonly class ReadEventsStatement implements ReadStatement
{
    // properties, constructor

    /** @return list<array{payload: non-empty-string}> */
    public function execute(ReadingDatabaseConnection $connection): array
    {
        return $connection->query(
            sprintf(
                'SELECT payload
                   FROM event
                  WHERE topic IN (%s)
                  ORDER BY id;',
                implode(
                    ', ',
                    array_fill(0, count($this->topics), '?'),
                ),
            ),
            ...$this->topics,
        );
    }
}
```

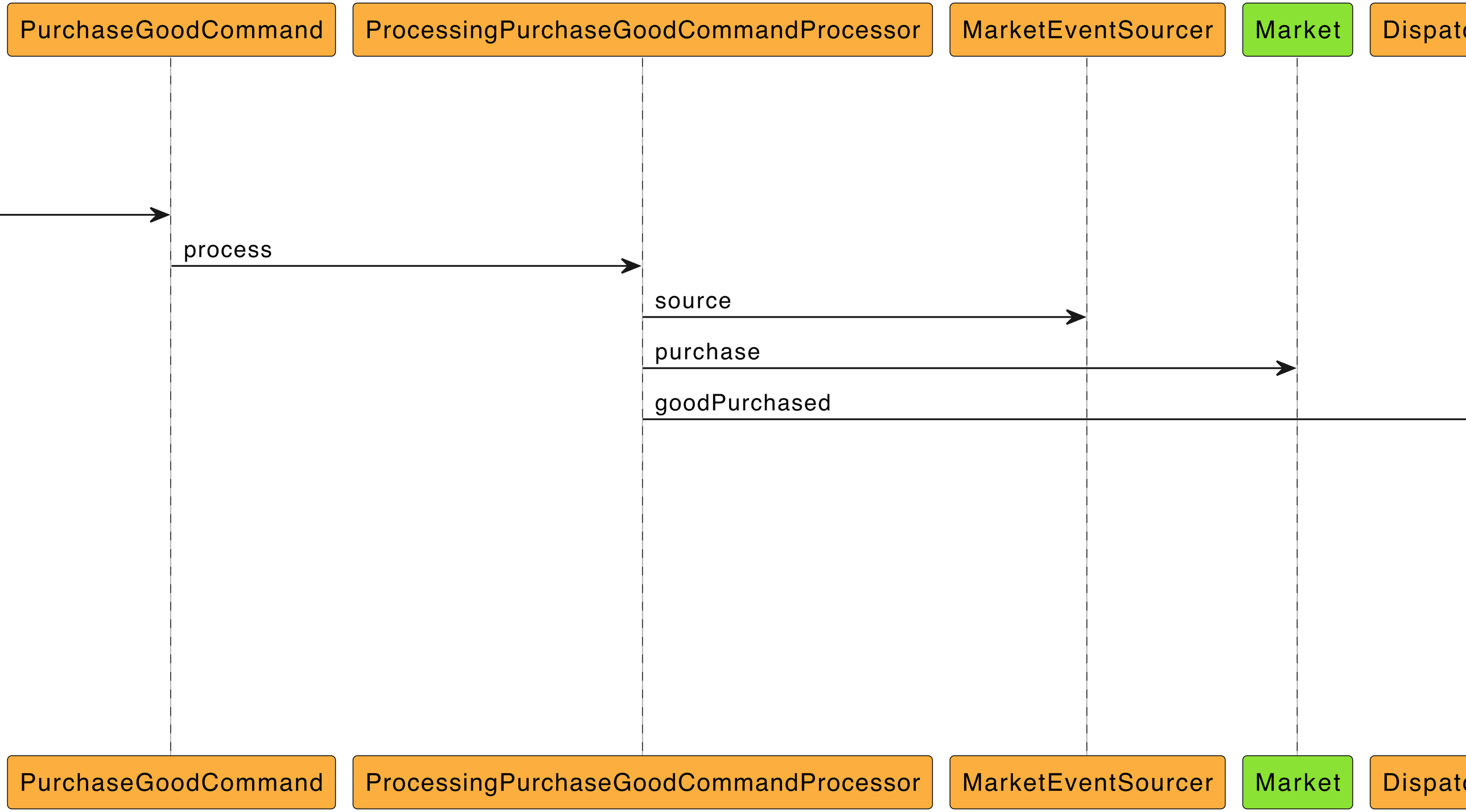
```php
<?php declare(strict_types=1);
namespace example\framework\event;

use SebastianBergmann\MysqliWrapper\WritingDatabaseConnection;

final readonly class DatabaseEventWriter implements EventWriter
{
    private WritingDatabaseConnection $connection;
    private EventJsonMapper $mapper;

    public function __construct(WritingDatabaseConnection $connection, EventJsonMapper $mapper)
    {
        $this->connection = $connection;
        $this->mapper     = $mapper;
    }

    /** @throws CannotMapEventException */
    public function write(Event $event): void
    {
        $statement = new WriteEventStatement(
            $event->id()->asString(),
            $event->topic(),
            $this->mapper->toJson($event),
        );

        $statement->execute($this->connection);
    }
}
```

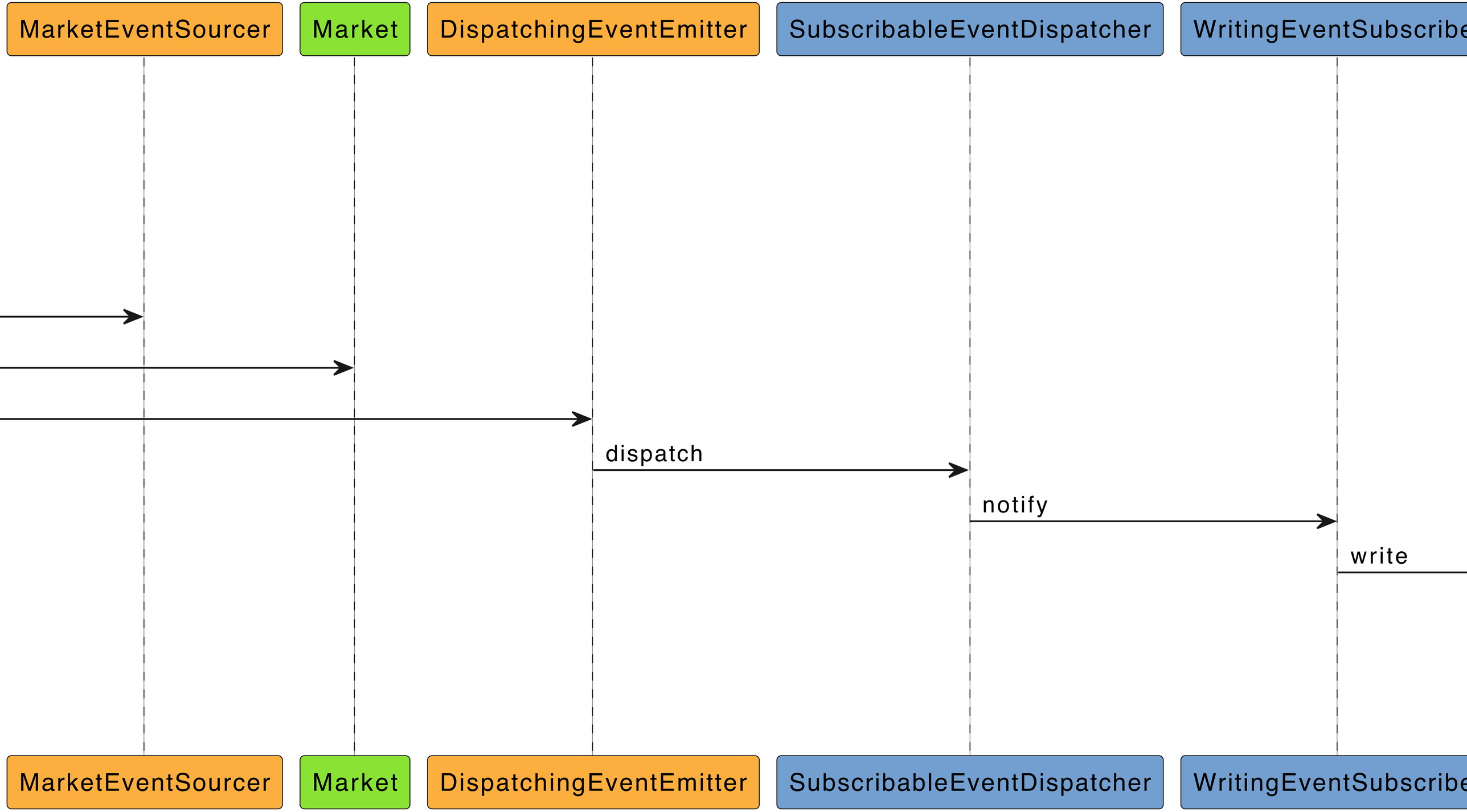
```php
<?php declare(strict_types=1);
namespace example\framework\event;

use SebastianBergmann\MysqliWrapper\WriteStatement;
use SebastianBergmann\MysqliWrapper\WritingDatabaseConnection;

final readonly class WriteEventStatement implements WriteStatement
{
    // properties, constructor

    public function execute(WritingDatabaseConnection $connection): void
    {
        $connection->execute(
            'INSERT INTO event
                        (event_id, topic, payload)
                VALUES (?, ?, ?);',
            $this->id,
            $this->topic,
            $this->payload,
        );
    }
}
```

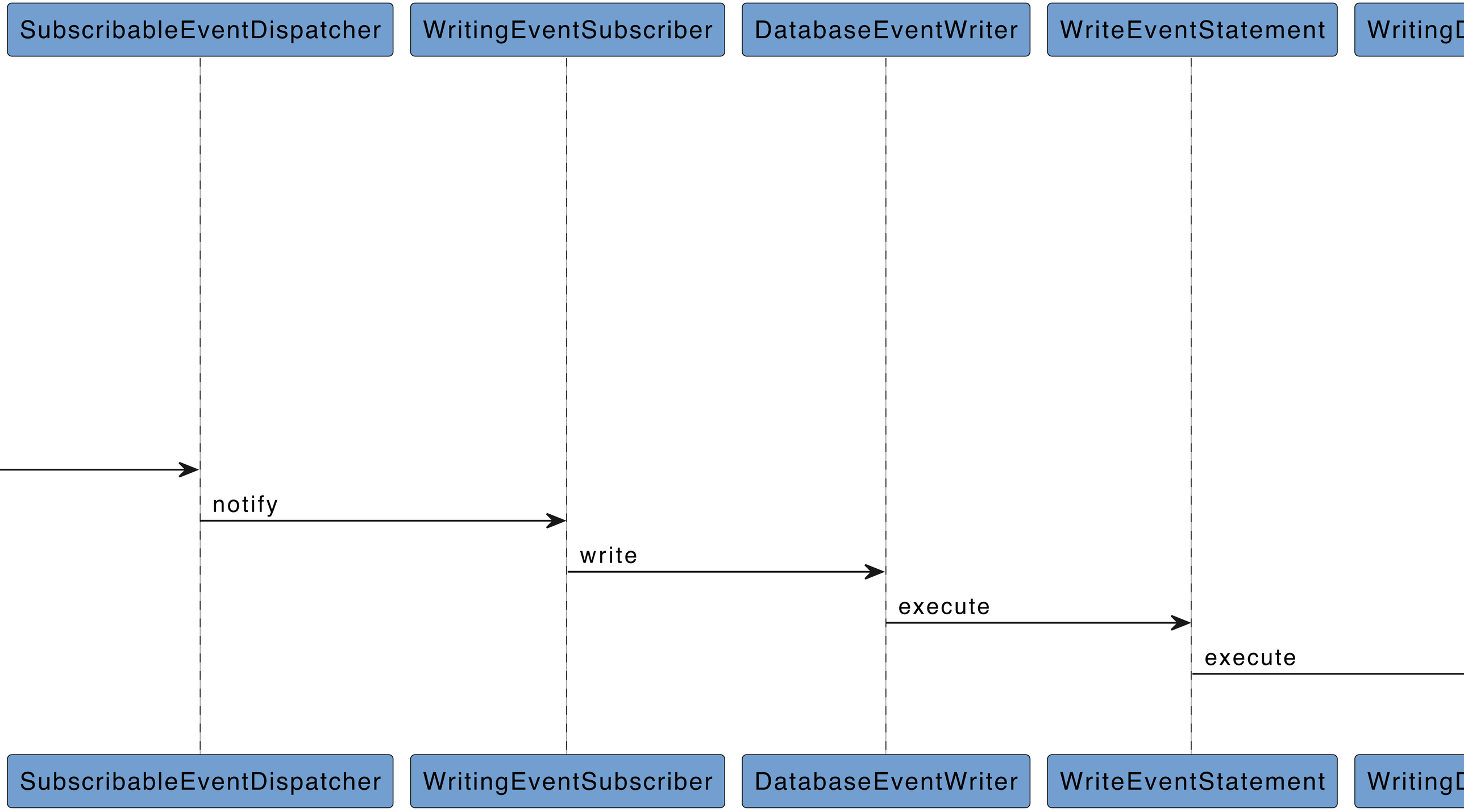
```php
<?php declare(strict_types=1);
namespace SebastianBergmann\MysqliWrapper;

interface ReadingDatabaseConnection
{
    /**
     * @param non-empty-string $sql
     *
     * @return list<array<non-empty-string, float|int|string>>
     */
    public function query(string $sql, float|int|string ...$parameters): array;
}



interface ReadStatement
{
    /**
     * @return list<array<non-empty-string, float|int|string>>
     */
    public function execute(ReadingDatabaseConnection $connection): array;
}
```

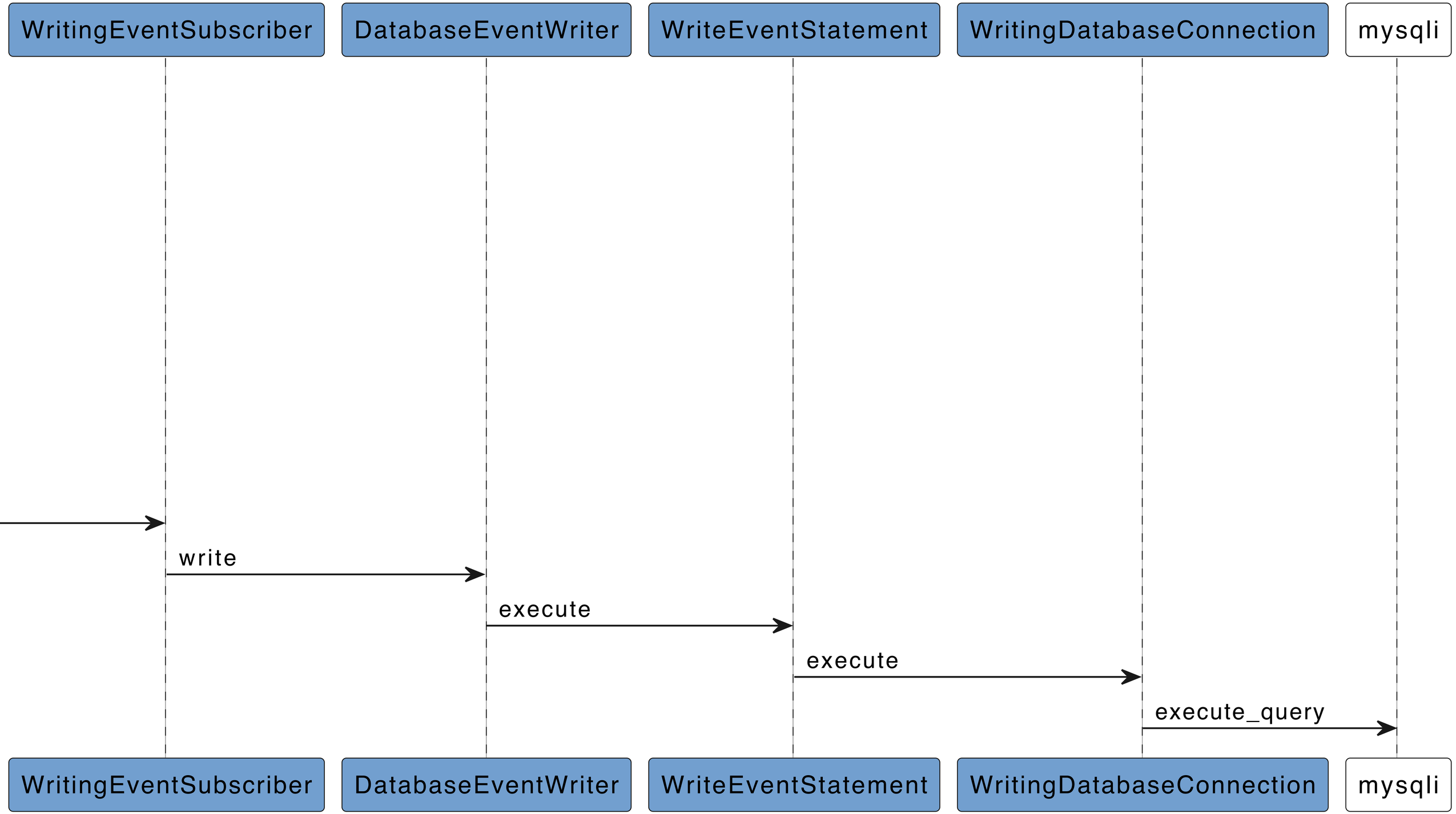
```php
<?php declare(strict_types=1);
namespace SebastianBergmann\MysqliWrapper;

interface WritingDatabaseConnection
{
    /**
     * @param non-empty-string $sql
     */
    public function execute(string $sql, float|int|string ...$parameters): true;
}



interface WriteStatement
{
    public function execute(WritingDatabaseConnection $connection): void;
}
```


Kernel
PostRequestRouter
PurchaseGoodRoute
PurchaseGoodCommand
ProcessingPurchaseGoodComma
route
route
execute
process
source
purchase
goodPurch
Kernel
PostRequestRouter
PurchaseGoodRoute
PurchaseGoodCommand
ProcessingPurchaseGoodComma


PurchaseGoodCommand
ProcessingPurchaseGoodCommandProcessor
MarketEventSourcer
Market
Dispat
process
source
purchase
goodPurchased
PurchaseGoodCommand
ProcessingPurchaseGoodCommandProcessor
MarketEventSourcer
Market
Dispat


MarketEventSourcer
Market
DispatchingEventEmitter
SubscribableEventDispatcher
WritingEventSubscribe
dispatch
notify
write
MarketEventSourcer
Market
DispatchingEventEmitter
SubscribableEventDispatcher
WritingEventSubscribe

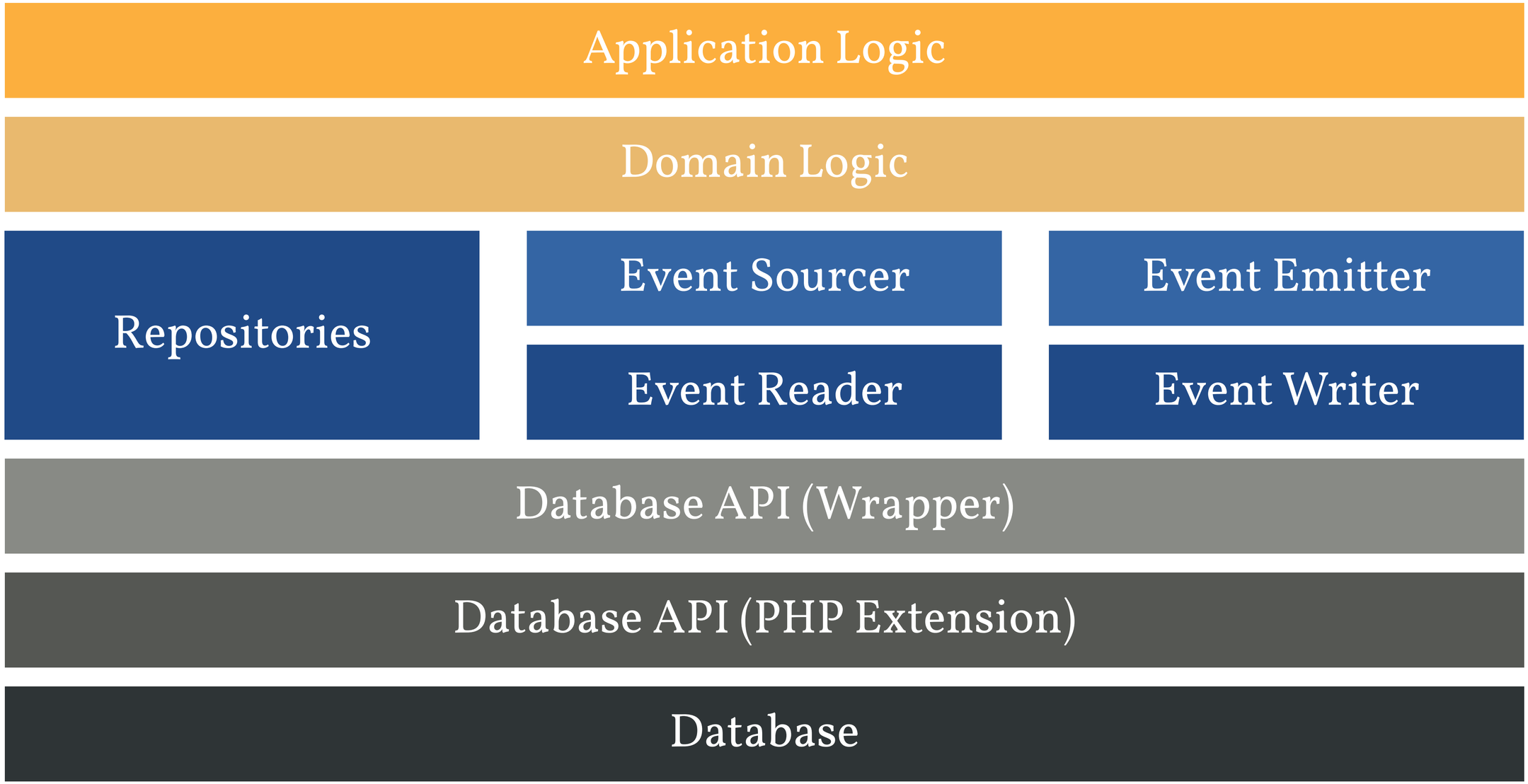

SubscribableEventDispatcher
WritingEventSubscriber
DatabaseEventWriter
WriteEventStatement
WritingD
notify
write
execute
execute
SubscribableEventDispatcher
WritingEventSubscriber
DatabaseEventWriter
WriteEventStatement
WritingD

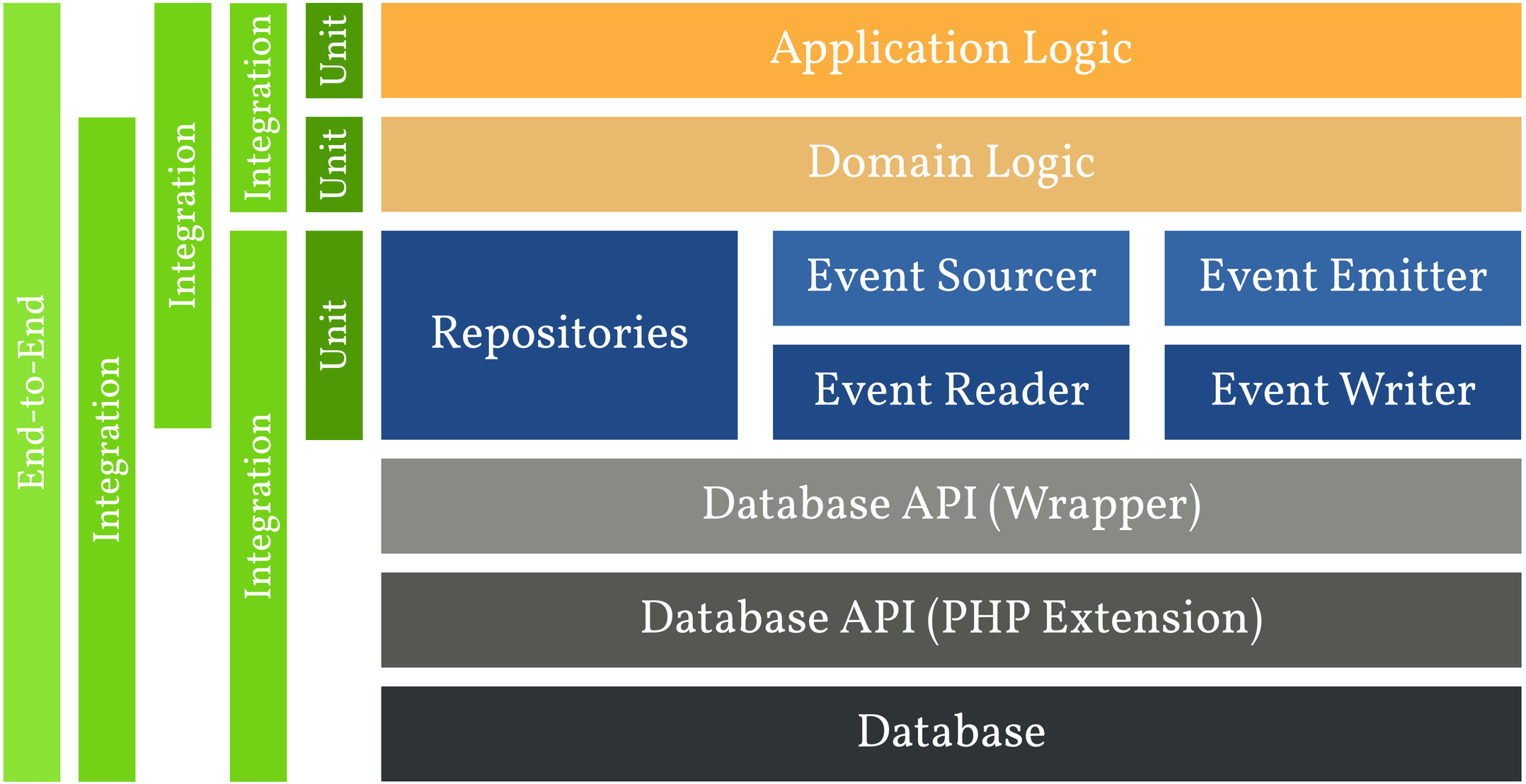

WritingEventSubscriber
DatabaseEventWriter
WriteEventStatement
WritingDatabaseConnection
mysqli
write
execute
execute
execute_query
WritingEventSubscriber
DatabaseEventWriter
WriteEventStatement
WritingDatabaseConnection
mysqli

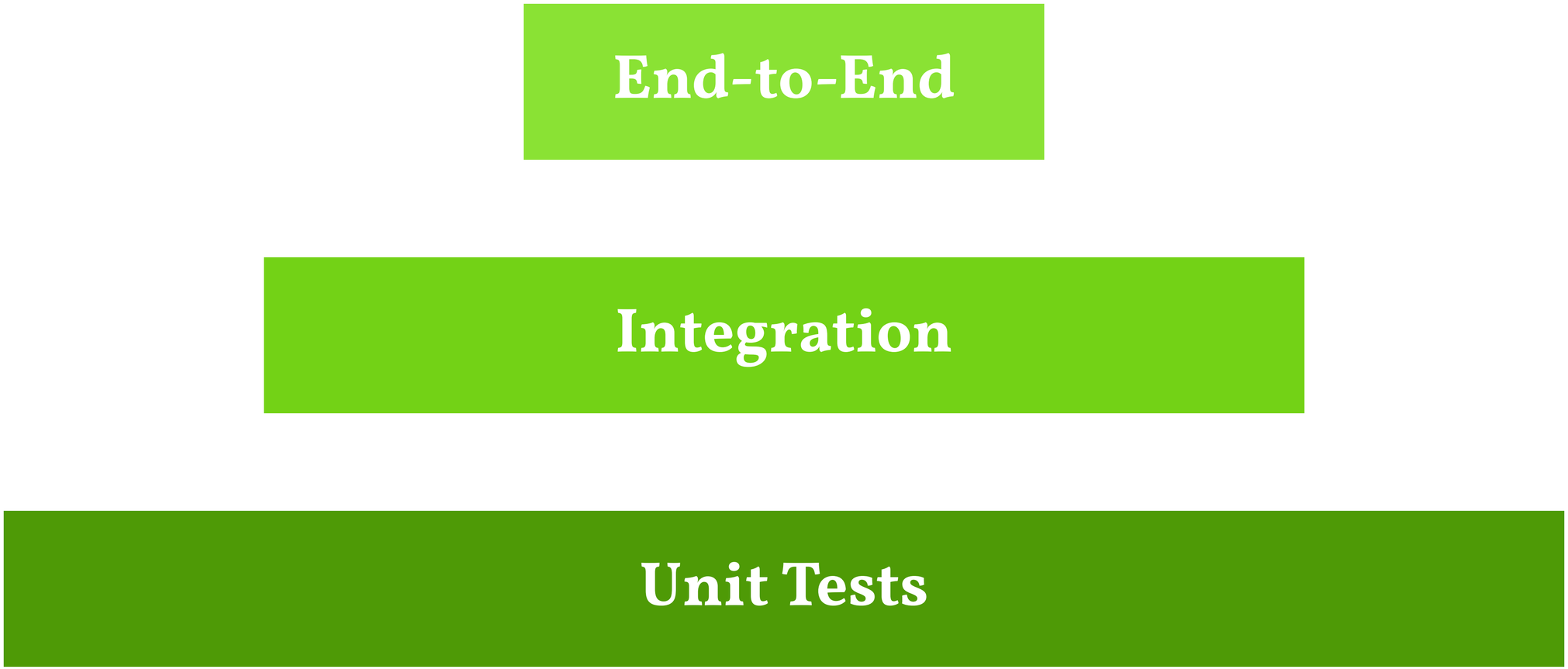
```php
<?php declare(strict_types=1);
namespace example\framework\event;

// use statements

final class DatabaseEventWriterTest extends DatabaseTestCase
{
    public function testWritesEventToDatabase(): void
    {
        $this->emptyTable('event');

        $this->writer()->write($this->event());

        $this->assertTableEqualsCsvFile(
            __DIR__ . '/../../../expectation/event.csv',
            'event',
            $this->eventSchema(),
        );
    }
}
```

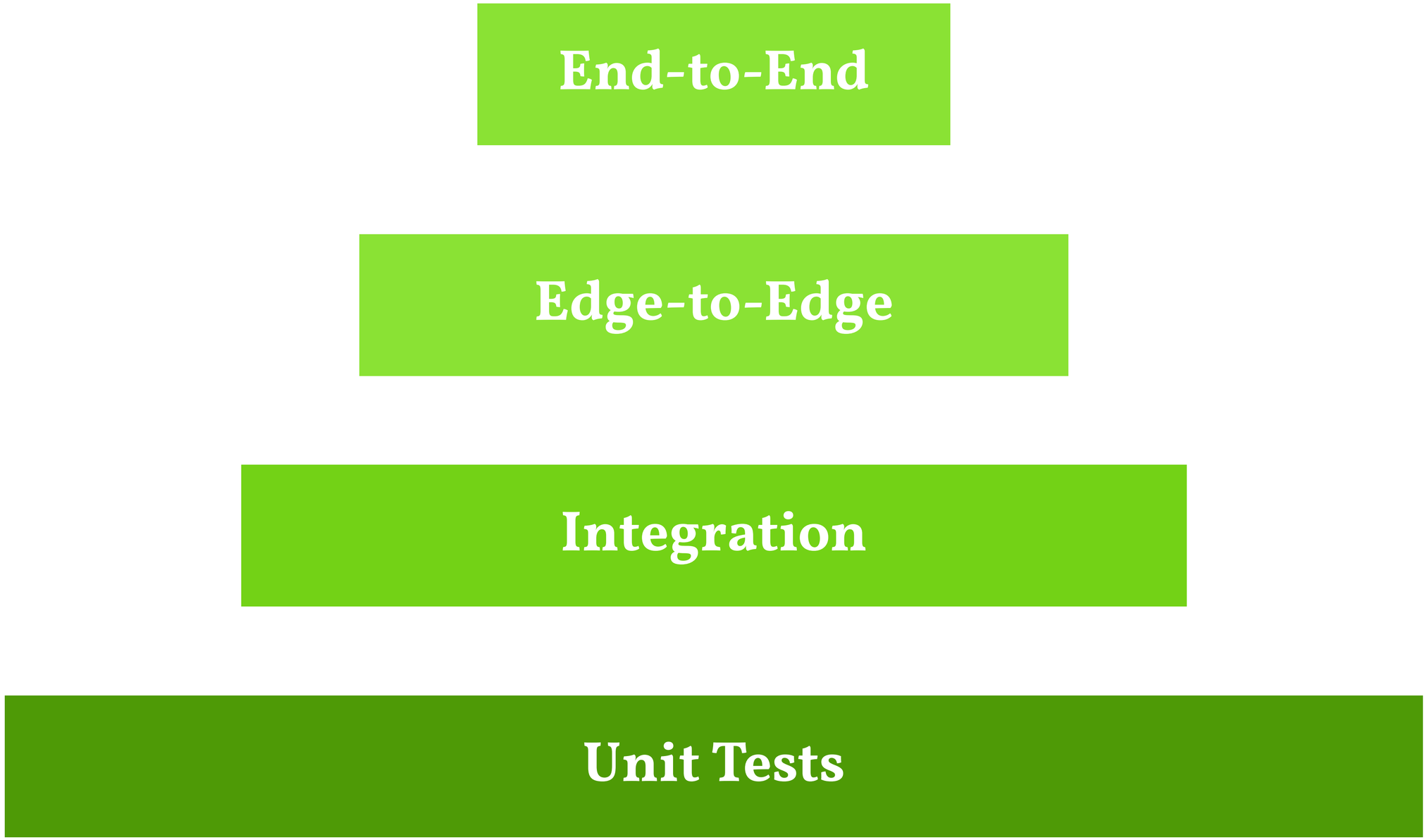
```php
<?php declare(strict_types=1);
namespace example\framework\event;

// use statements

final class WriteEventStatementIntegrationTest extends DatabaseTestCase
{
    public function testWritesEventToDatabase(): void
    {
        $this->emptyTable('event');

        $statement = new WriteEventStatement(
            'b5578a2a-3188-470c-a2b7-3a249faed6fb',
            'the-topic',
            'payload',
        );

        $statement->execute($this->connectionForWritingEvents());

        $this->assertTableEqualsCsvFile(
            __DIR__ . '/../../../expectation/write_event.csv',
            'event',
            $this->eventSchema(),
        );
    }
}
```

```php
<?php declare(strict_types=1);
namespace example\framework\event;

// use statements

final class WriteEventStatementTest extends TestCase
{
    public function testExecutesStatementToWriteEvent(): void
    {
        $connection = $this->createMock(WritingDatabaseConnection::class);

        $connection
            ->expects($this->once())
            ->method('execute')
            ->with(
                'INSERT INTO event
                        (event_id, topic, payload)
                  VALUES (?, ?, ?);',
                'id',
                'topic',
                'payload',
            );

        $statement = new WriteEventStatement('id', 'topic', 'payload');

        $statement->execute($connection);
    }
}
```


Application Logic
Domain Logic
Repositories
Event Sourcer
Event Emitter
Event Reader
Event Writer
Database API (Wrapper)
Database API (PHP Extension)
Database


End-to-End
Integration
Integration
Integration
Unit
Unit
Unit
Application Logic
Domain Logic
Repositories
Event Sourcer
Event Emitter
Event Reader
Event Writer
Database API (Wrapper)
Database API (PHP Extension)
Database


End-to-End
Integration
Unit Tests


End-to-End
Edge-to-Edge
Integration
Unit Tests

thePHP.cc

For a class to be easy to unit-test, the class must have **explicit dependencies** that can easily be substituted and **clear responsibilities** that can easily be invoked and verified.

In software-engineering terms, that means that the code must be **loosely coupled** and **highly cohesive** - in other words, well-designed.

http://www.growing-object-oriented-software.com/

The **Single Responsibility Principle** states that each class (and method) should have exactly one responsibility.

https://thephp.cc/topics/solid

Queries [r]eturn a result and [are free of side effects].

Commands [c]hange the state of a system but do not return a value.

https://martinfowler.com/bliki/CommandQuerySeparation.html

A **Repository** mediates between the domain and data mapping layers, acting like an in-memory domain object collection.

https://martinfowler.com/eaaCatalog/repository.html

The **Data Mapper** is a layer of software that separates the in-memory objects from the database. Its responsibility is to transfer data between the two and also to isolate them from each other.

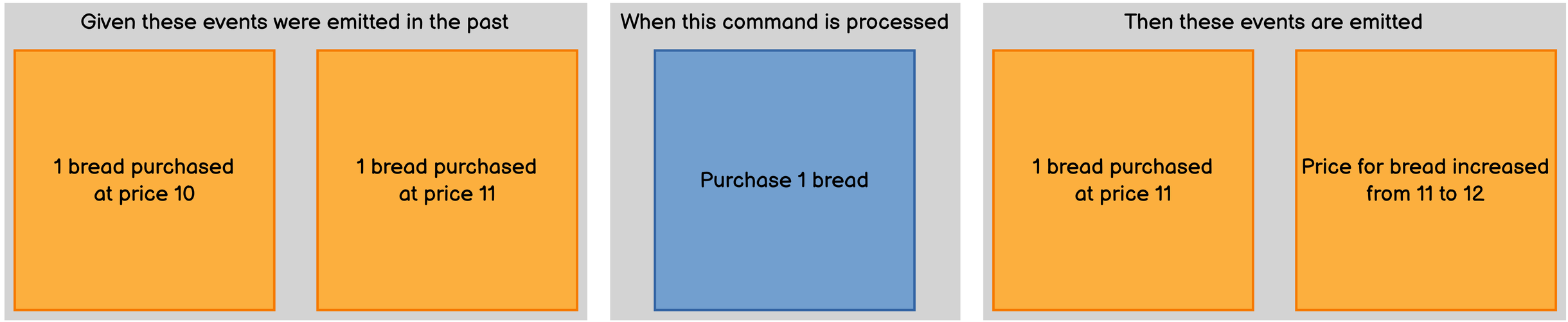
https://martinfowler.com/eaaCatalog/dataMapper.html

Event Sourcing ensures that all changes to application state are stored as a sequence of events. Not just can we query these events, we can also use the event log to reconstruct past states.

https://martinfowler.com/eaaDev/EventSourcing.html

```php
<?php declare(strict_types=1);
namespace example\caledonia\application;

// use statements

final class PurchaseGoodCommandProcessorTest extends EventTestCase
{
   public function testEmitsGoodPurchasedAndPriceChangedEvents(): void
    {
        $this->given(
            $this->goodPurchased(Good::Bread, Price::from(10), 1),
            $this->goodPurchased(Good::Bread, Price::from(11), 1),
        );

        $this->when(new PurchaseGoodCommand(Good::Bread, 1));

        $this->then(
            $this->goodPurchased(Good::Bread, Price::from(11), 1),
            $this->priceChanged(Good::Bread, Price::from(11), Price::from(12)),
        );
    }
}
```

thePHP.cc


Given these events were emitted in the past
1 bread purchased at price 10
1 bread purchased at price 11
When this command is processed
Purchase 1 bread
Then these events are emitted
1 bread purchased at price 11
Price for bread increased from 11 to 12

Thank you!

https://thePHP.cc/

sebastian@thephp.cc

@sebastian@phpc.social

Image Credits

- https://www.pexels.com/de-de/foto/brandenburgh-gate-deutschland-1114892
- https://www.pexels.com/de-de/foto/die-gluhbirne-577514