One Year Business Events

Escaping the Monolith & Lessons Learned

Stefan Beier
FlixMobility Tech GmbH

TEAM MCDRIVER - WE CONNECT OUR BUSSES

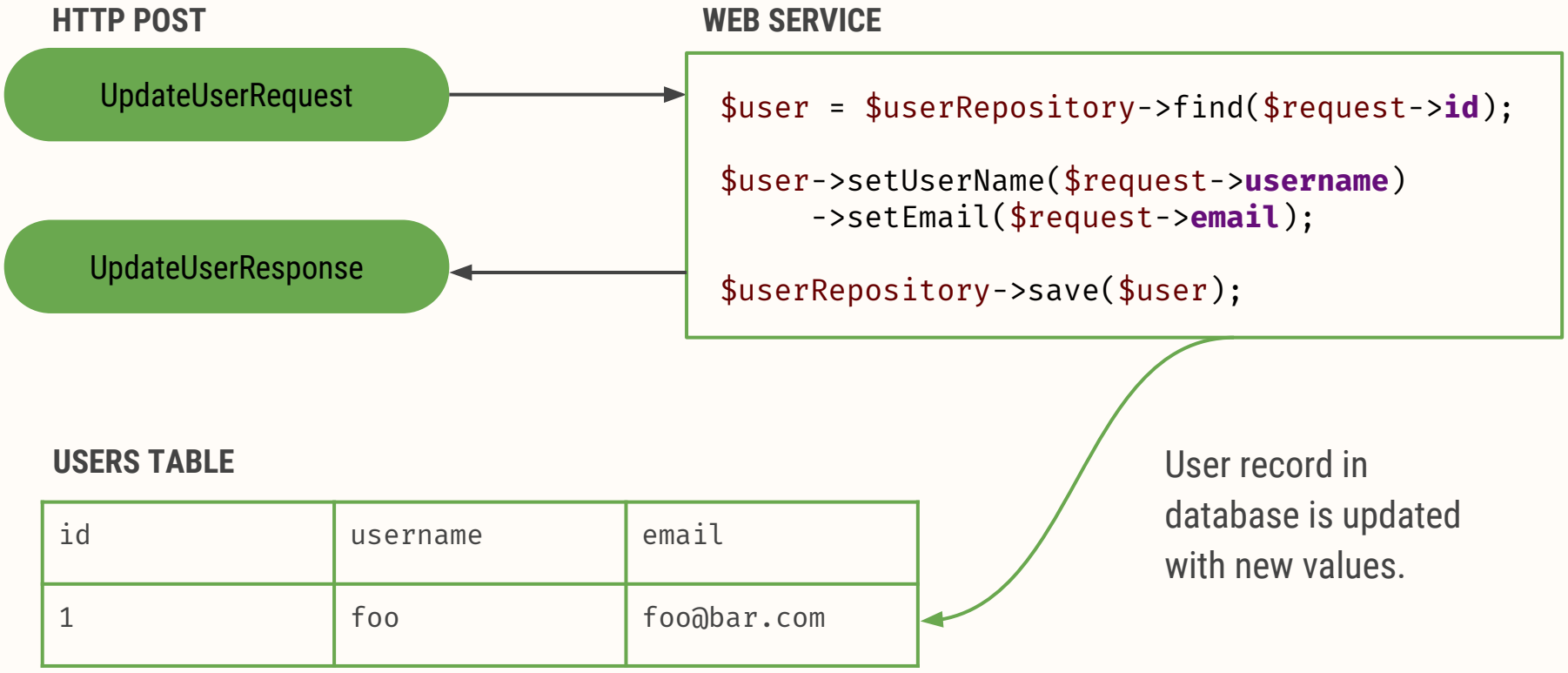
ANATOMY OF A BUSINESS EVENT

- Something important has happened event
- Aggregate root
- Version
- Date
- Payload / Data

ANATOMY OF A BUSINESS EVENT


```
{
   "id": "34d33a71-d4f4-4451-9df0-aa4b6ab48c86",
   "name": "flixtech.bus_driver_service.passenger.checked_in",
   "version": 1,
   "metadata": {
       "user_id": xxx,
       "user_agent": "samsung/xcover3ltexx/xcover3lte:..."
   },
   "created_at": "2018-06-15 10:16:27.000000",
   "payload": {
       "passenger_id": "xxx-xxx",
       "ride": {
           "ride_id": "xxx-xxx",
           "station_id": "xxx-xxx",
           "status": "checked",
           "status_timestamp": 1529057786
       }
   }
}
```

THE CRUD WAY


HTTP POST
UpdateUserRequest
UpdateUserResponse
WEB SERVICE
$user = $userRepository->find($request->id);
$user->setUserName($request->username)
->setEmail($request->email);
$userRepository->save($user);
USERS TABLE
id
username
email
1
foo
foo@bar.com
User record in database is updated with new values.

THE EVENT SOURCING WAY

HTTP POST

ChangeUserEmailRequest

UpdateUserResponse

WEB SERVICE

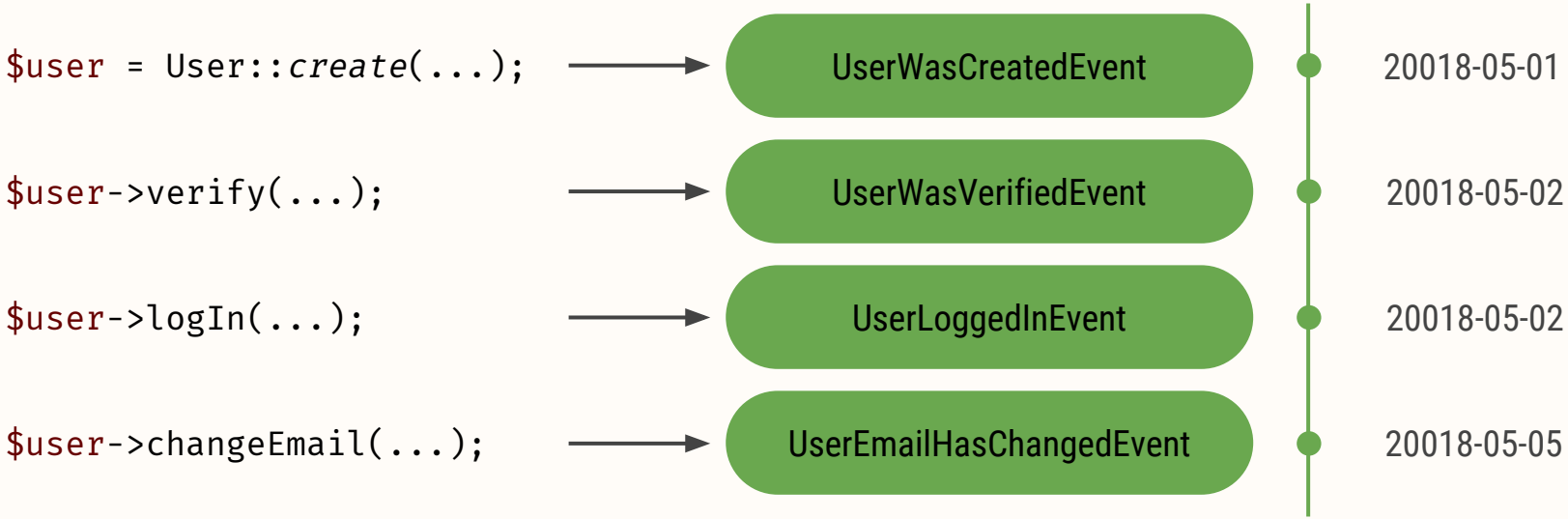
```
$user = $userRepository->find($request->id);

$user->changeEmail($request->email);

$userRepository->save($user);
```

Append “EmailHasChangedEvent” event

USER_EVENT_STORE

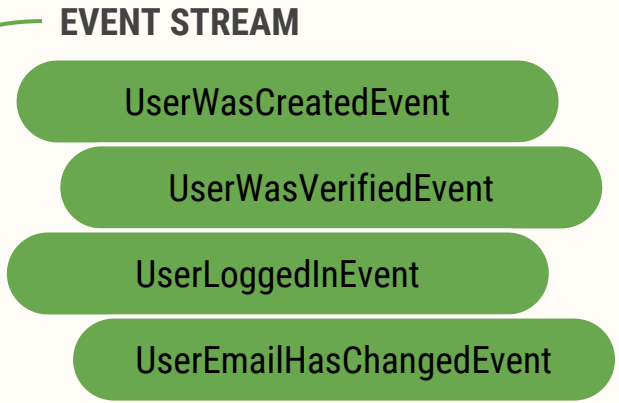
event_id	aggregate_id	aggregate_name	event_name	payload	version
1	yyyy	User	UserWasCreatedEvent	{...}	1
2	yyyy	User	EmailHasChangedEvent	{ "email": "foo" }	2

EVENT SOURCING



REPLAY EVENTS

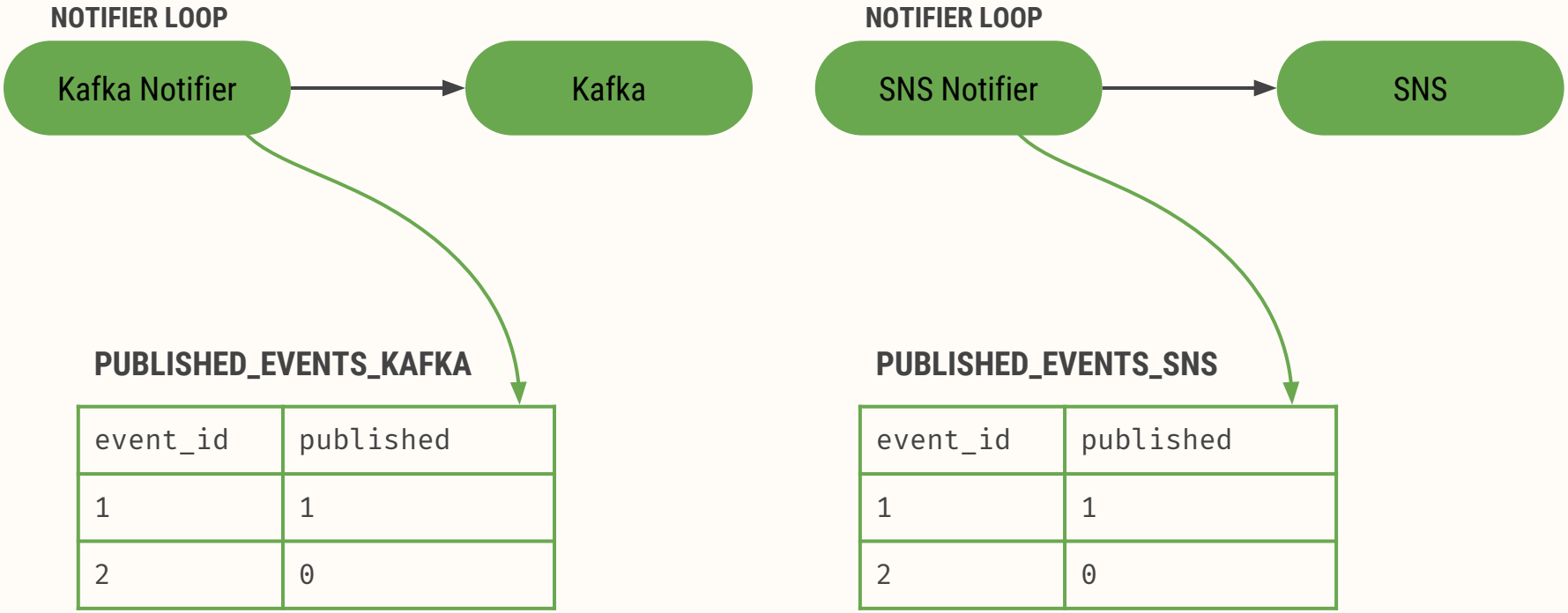
```php
// UserRepository.php
public function find(UserId $userId): ?User
{
    $eventStream = $this->eventStore->load($userId);

    $user = User::fromEventStream($eventStream);

    return $user;
}
```




```php
// User.php
public static function applyUserWasVerifiedEvent(
    UserWasVerifiedEvent $event,
    User $user
): User {
    $user->verified = true;
    $user->version = $event->version();

    return $user;
}
```

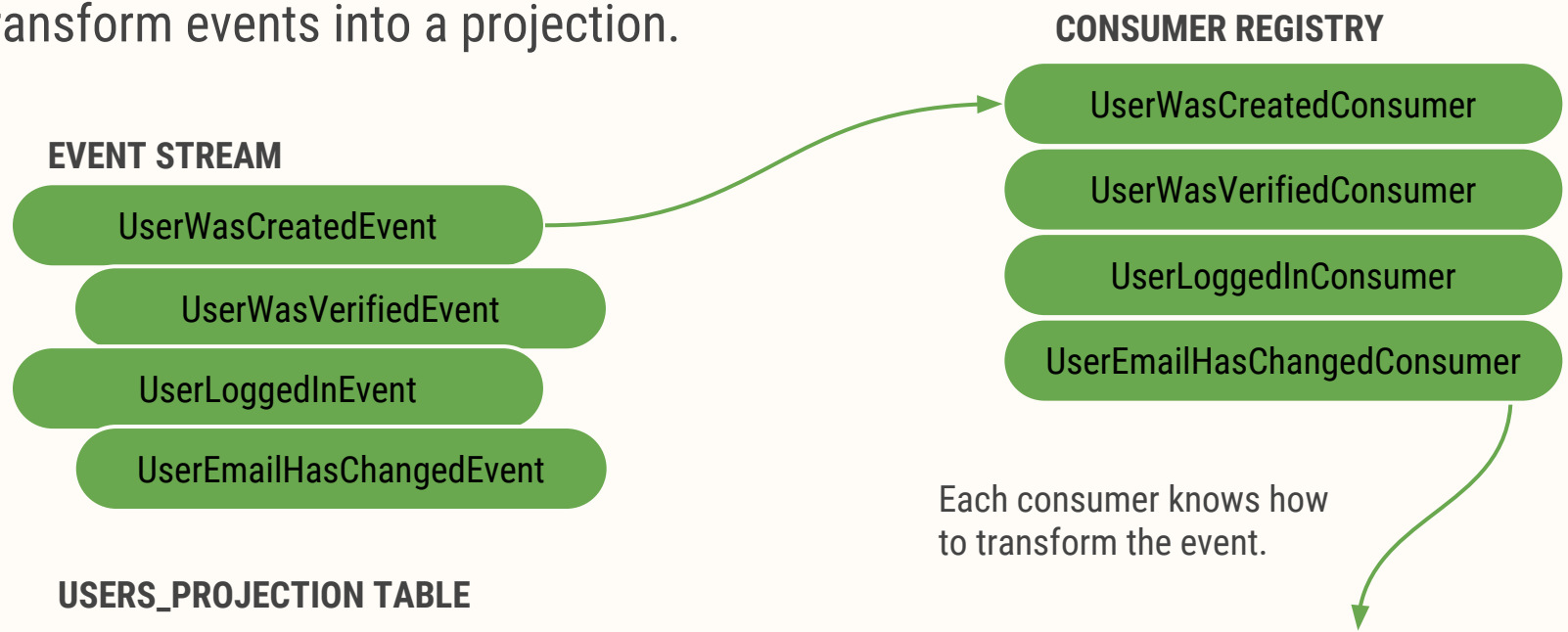
PUBLISHING EVENTS

Keep track of each event which should be published to a messaging service



CONSUMING EVENTS

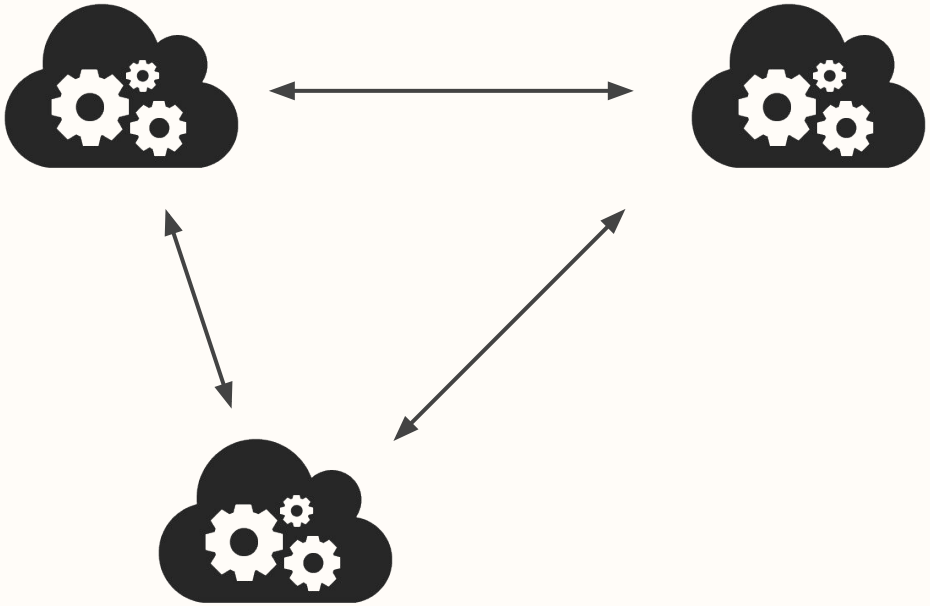
Transform events into a projection.



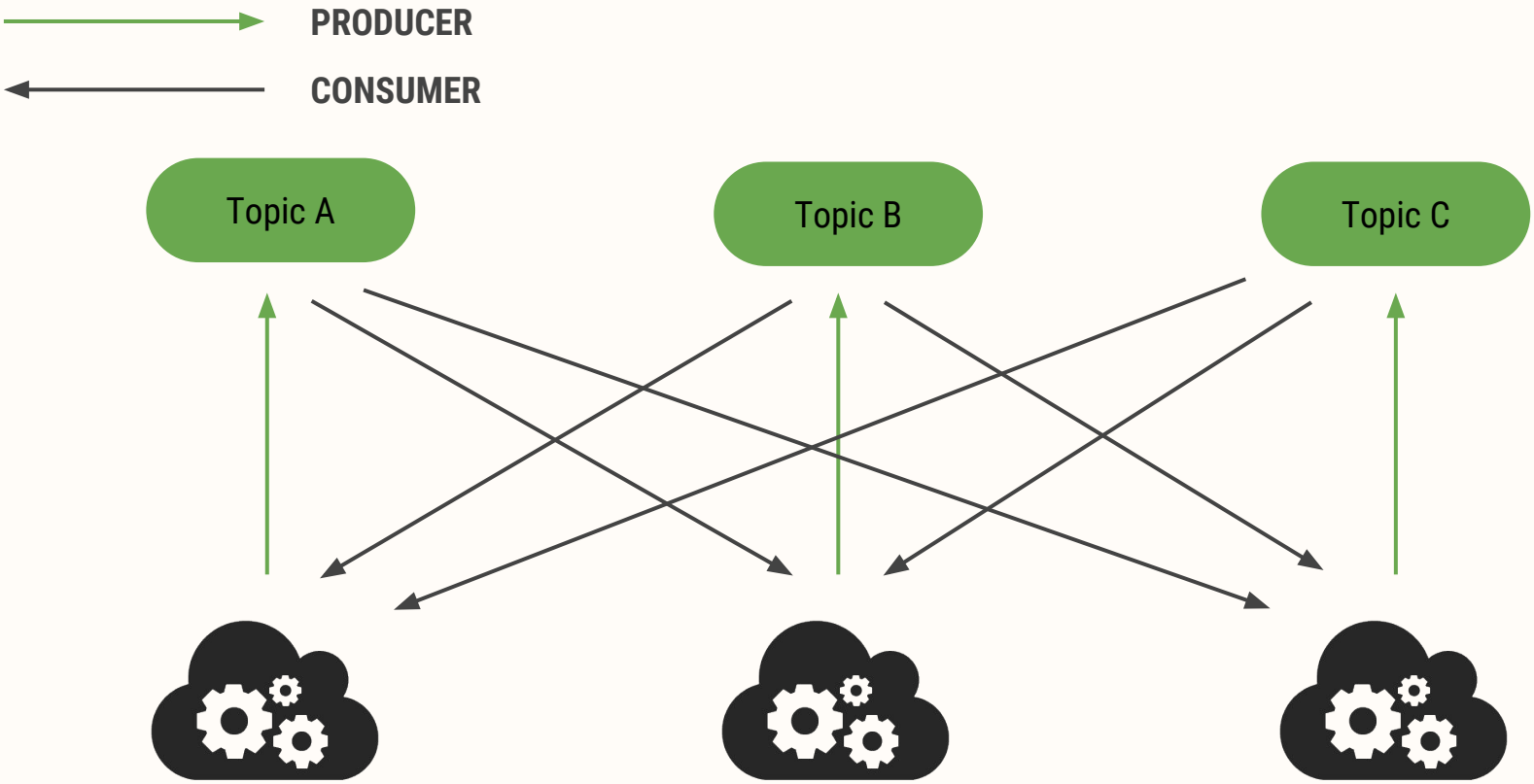
USERS_PROJECTION TABLE

id	username	email	verified	last_login
1	foo	foo@bar.com	true	2018-06-01 00:00:00

EVENT SOURCING AS DECOUPLING STRATEGY



EVENT SOURCING AS DECOUPLING STRATEGY



EVENT SOURCING AS DECOUPLING STRATEGY

Identify data ownership and extract small parts.

OLD SERVICE

```
class Station
{
    private $name;
    private $address;
    private $zip;
    private $city;
    private $latitude;
    private $longitude;
    private $slug;
    private $timezone;
    private $navigationLatitude;
    private $navigationLongitude;
}
```

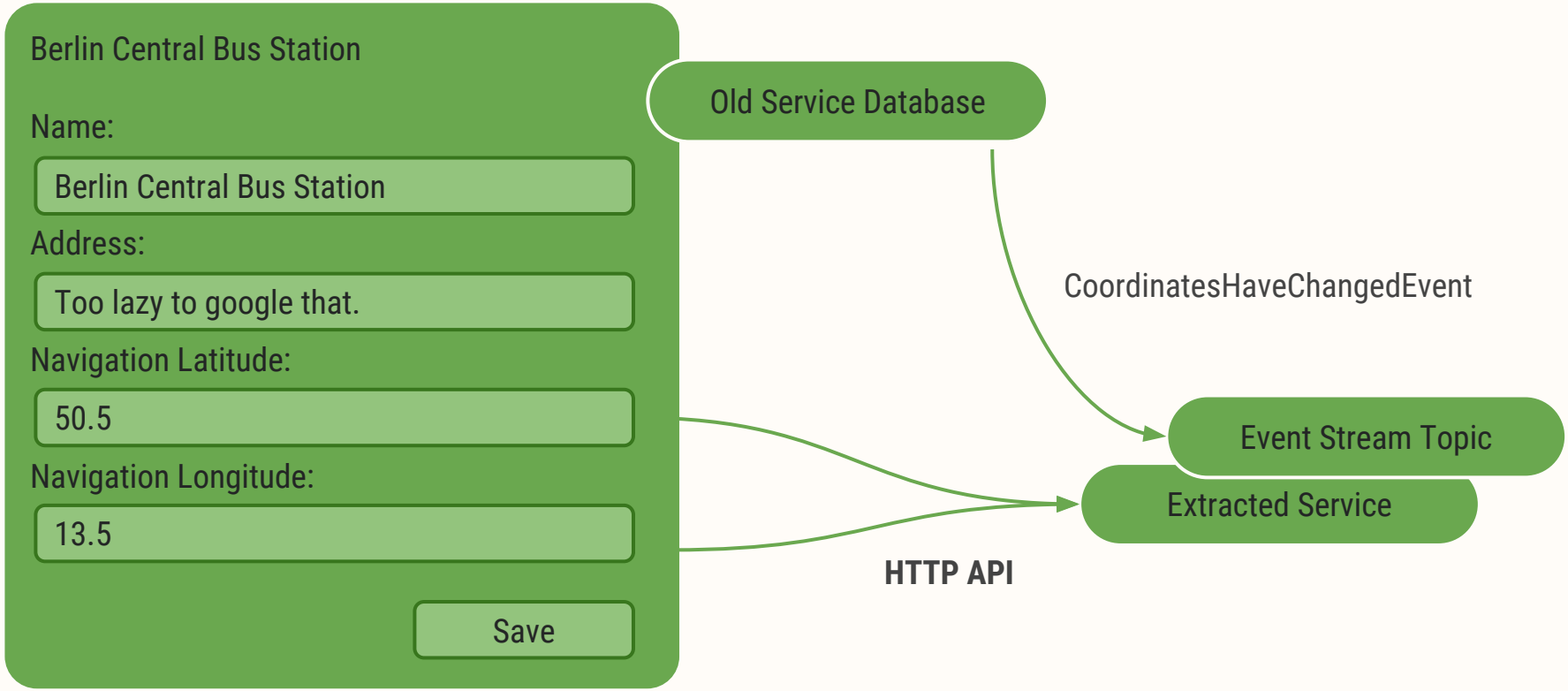
EXTRACTED SERVICE

```
class Station
{
    private $id;
    private $navigationLatitude;
    private $navigationLongitude;
    private $version;

    public function changeNavigationCoordinates(...): void {}
}
```

Fields become read-only and will only update from the event stream.

Can be removed from old service once all usages are removed.

EVENT SOURCING AS DECOUPLING STRATEGY



LESSONS LEARNED

Use fully qualified names for your events.

Different teams can have the same name for something. Make a difference.

```
PassengerCheckedInEvent

mc_driver.bus_driver_service.passenger.checked_in


StationNavigationCoordinatesChangedEvent

mc_driver.bus_driver_service.station.navigation_coordinates_changed
```

LESSONS LEARNED

Consumers must be idempotent to guarantee consistency on multiple delivery.

```php
public function checkVersion(?Entity $entity = null, Version $eventVersion)
{
    if ($entity == null && $eventVersion->rawValue() == 1) {
        return;
    }

    if ($entity == null) {
        throw new EventShouldBePostponedException();
    }

    if ($eventVersion->rawValue() <= $entity->getVersion()) {
        throw new EventShouldBeSkippedException();
    }

    if ($entity->getVersion() + 1 == $eventVersion->rawValue()) {
        return;
    }

    throw new EventShouldBePostponedException();
}
```

LESSONS LEARNED

Consumers should monitor postponed and skipped events.

> [bus-driver-service] production - New Error: Exception: Event e7e5467c-11e5-4301-a232-02adb4d14a2a was postponed 117 times. Better check what is going on. https://rollbar.com/flixtech/bus-driver-service/items/11571/

```
try {
   $eventHandler->handle($event);

   call_user_func($processedCallback);

} catch (EventShouldBePostponedException $e) {
   call_user_func($postponedCallback);

   $this->logPostponed($event);

} catch (EventShouldBeSkippedException $e) {
   call_user_func($processedCallback);

   $this->logSkipped($event);
}
```

LESSONS LEARNED

Consumers must be able to handle unknown or new events to prevent version gaps.

New events can be added at any time and it shouldn’t break your consumers.

```
if (!$eventIsUnknown) {

    $this->unknownEventConsumer->handle($event);

    return;
}
```

LESSONS LEARNED

Use a standard format for events throughout the organisation.

```
{
    "id": "...",
    "name": "...",
    "aggregate_id": "...",
    "version": 1,
    "metadata": {},
    "created_at": "2018-06-15 10:16:27.000000",
    "payload": {}
}
```

FURTHER READINGS

- Eventual Consistency
- Domain Driven Design
- Command Query Responsibility Segregation (CQRS)
- Event storming