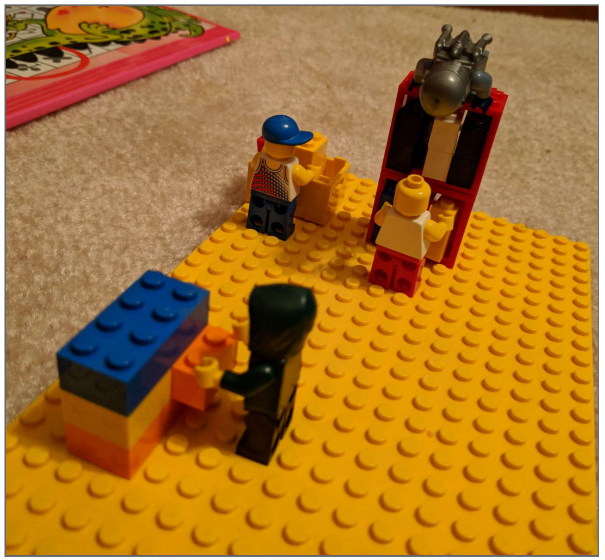
EVENT SOURCING A SMALL LIBRARY

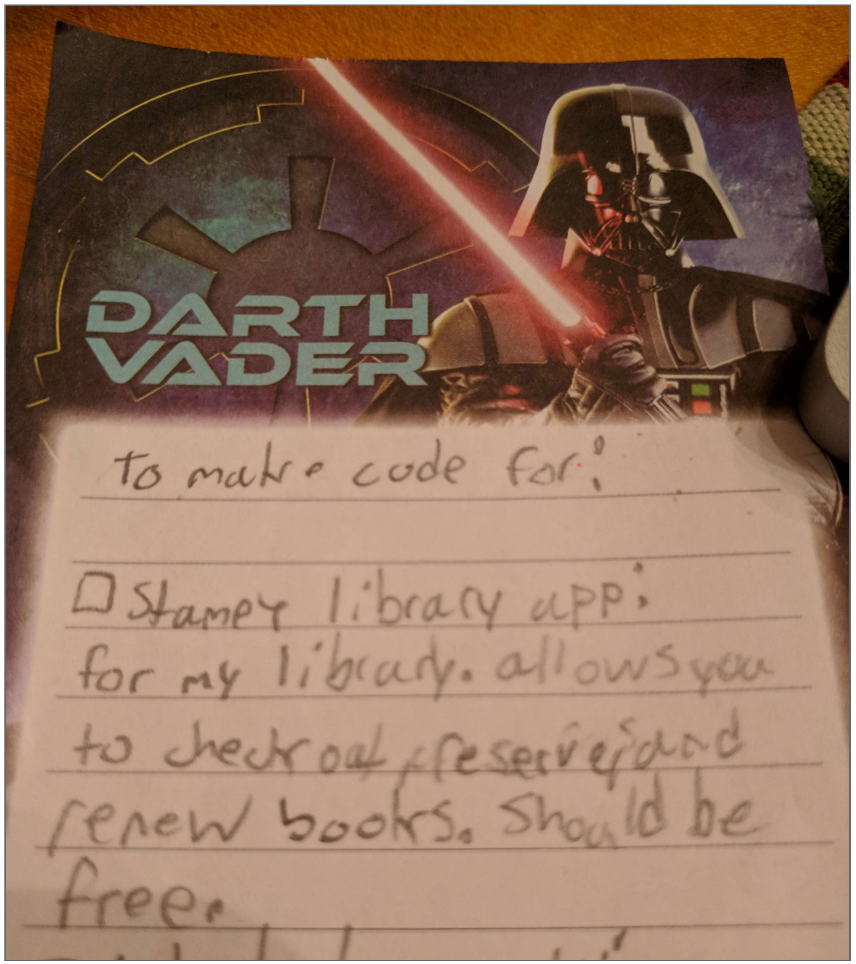
Emily Stamey
@elstamey



- PHP developer
- Organizer of Triangle PHP
- Director WWCode Raleigh-Durham
- Work at InQuest


LEMONADE STAND
ROBOT
LAWN SERVICE
LIBRARY APP
roomba
GAMES
Pew Pew

The Librarian


DARTH
VADER
to make code for:
☐ Stamey library app:
for my library. allows you
to check out, reserve, and
renew books. Should be
free.

REQUIREMENTS

- Check out books
- Reserve books
- Renew books

THE LIBRARIAN'S PROCESS

- Check in and Check out Books
- View a List of Books I Own
- See who has checked out books

CRUD

```php
<?php

namespace Library\Http\Controllers\Api\V1;

use Illuminate\Http\Request;
use Library\Book;
use Library\Http\Controllers\Controller;

class BooksController extends Controller
{
    public function index()
    {
        return Book::all();
    }

    public function show($id)
    {
        return Book::findOrFail($id);
    }

    public function update(Request $request, $id)
    {
        $Book = Book::findOrFail($id);
        $Book->update($request->all());
```

BOOK CHECK-IN/OUT

- Modify update method to change status
- Using a status table to show if the book is in or out
- Any of this would be fine

PAUSE FOR WARNING

Switch from CRUD to ES is tough

#NotAllApplications **need** to be Event-Sourced

LEARNING OBJECTIVES:

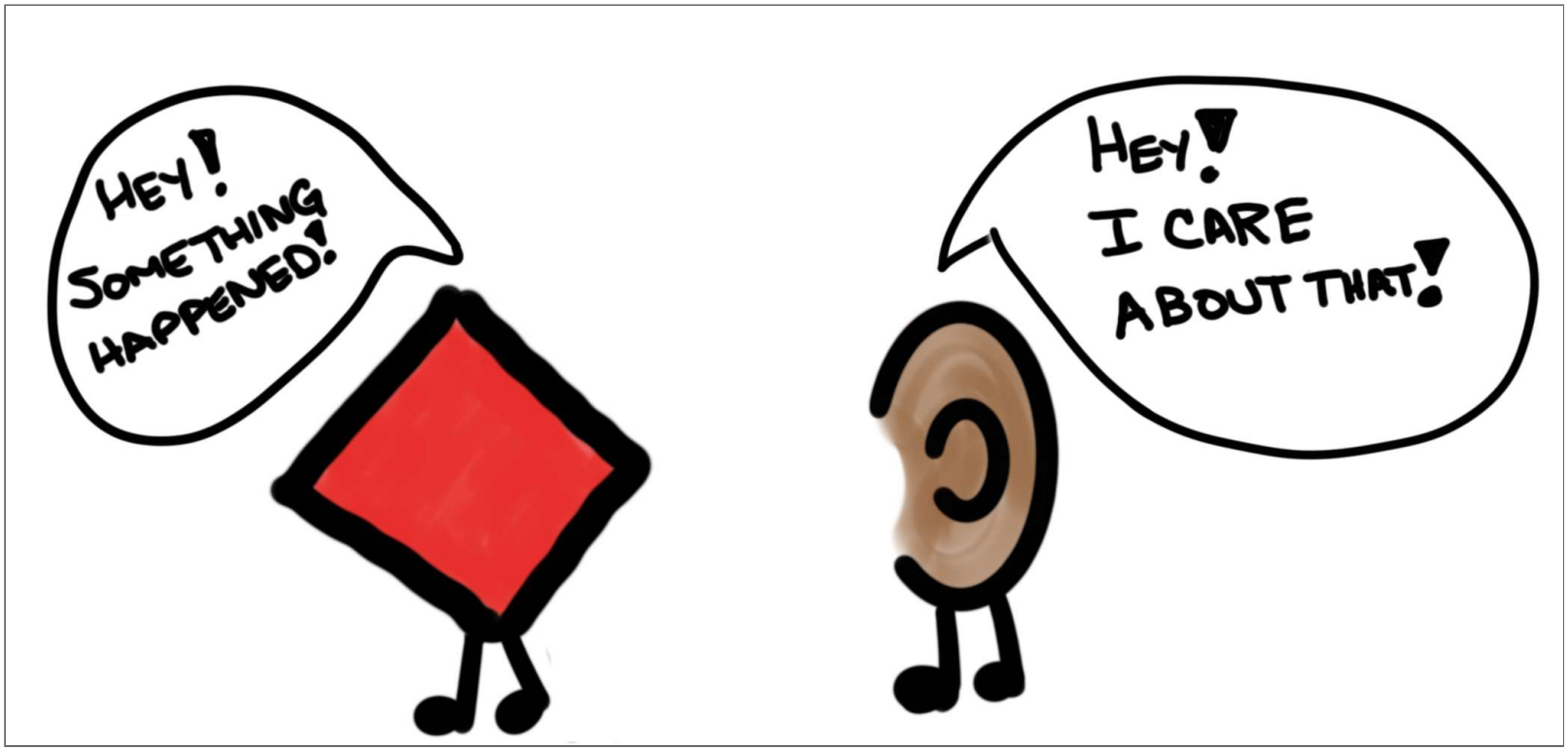
- What is Event Sourcing
  - Rules to Follow
  - Data Structures: Events, Projections, Read Models, CQRS(briefly)
- How our code will change
- Why would we want to use Event Sourcing

EVENT SOURCING

The fundamental idea of Event Sourcing is that of ensuring **every change to the state** of an application **is captured in an event object**, and that these event objects are themselves stored in the sequence they were applied for the same lifetime as the application state itself.

Martin Fowler

EVENTS AND LISTENERS



AN EVENT

What happened?
BookWasCheckedOut

What do I need to remember about it?
(book, patron, date)

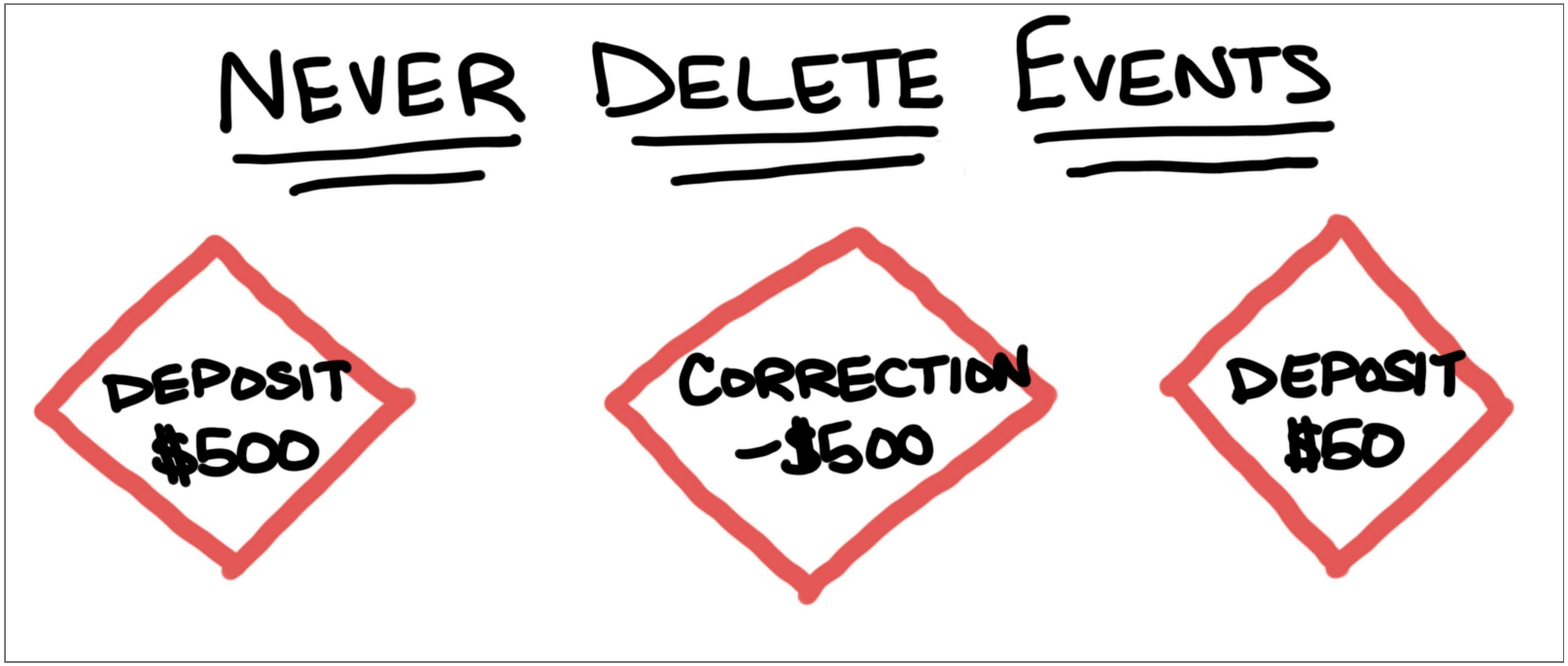
ATTRIBUTES

Save only what you need to preserve, The rest can be looked up

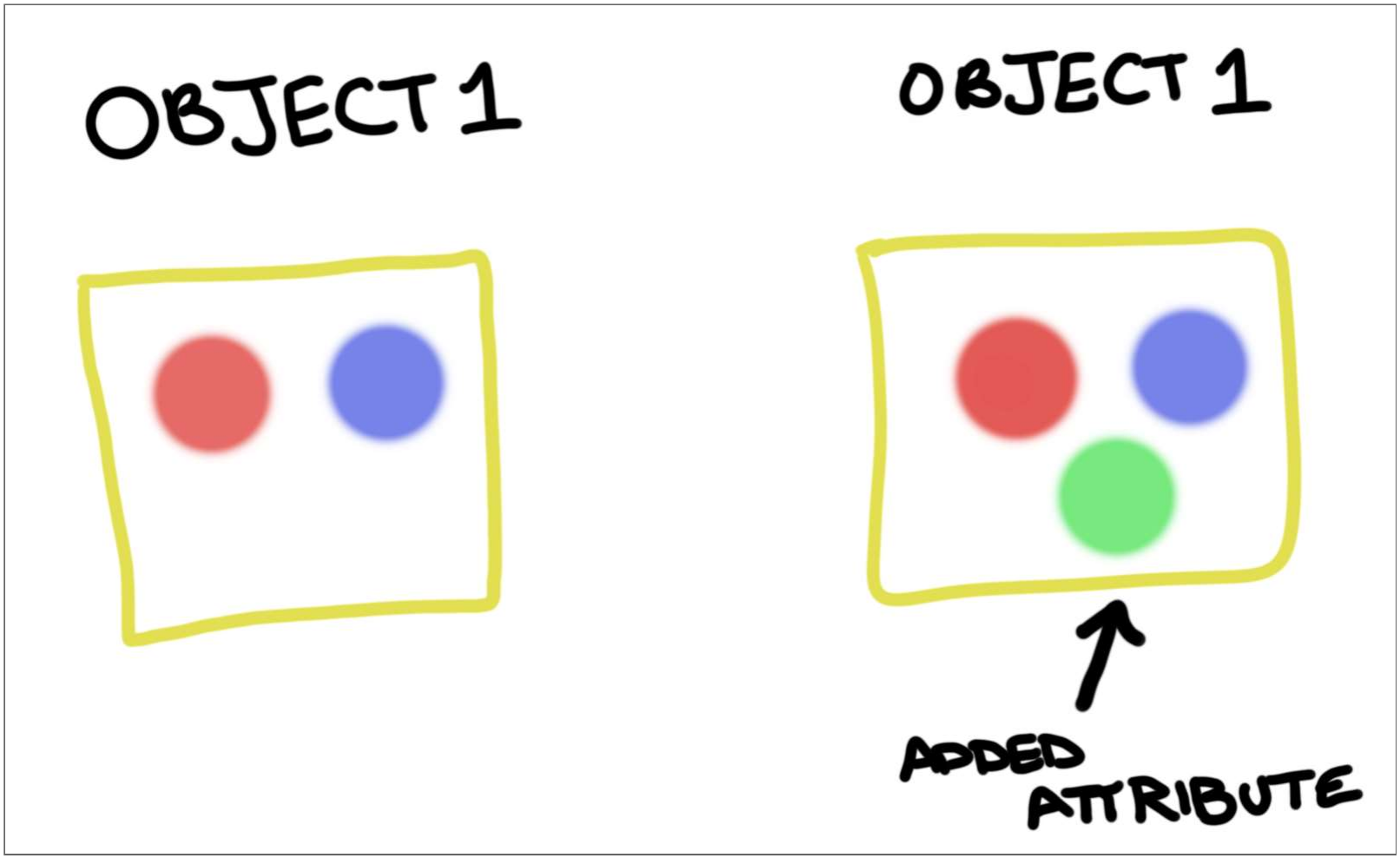
(book id, patron id, date)

RULES TO FOLLOW

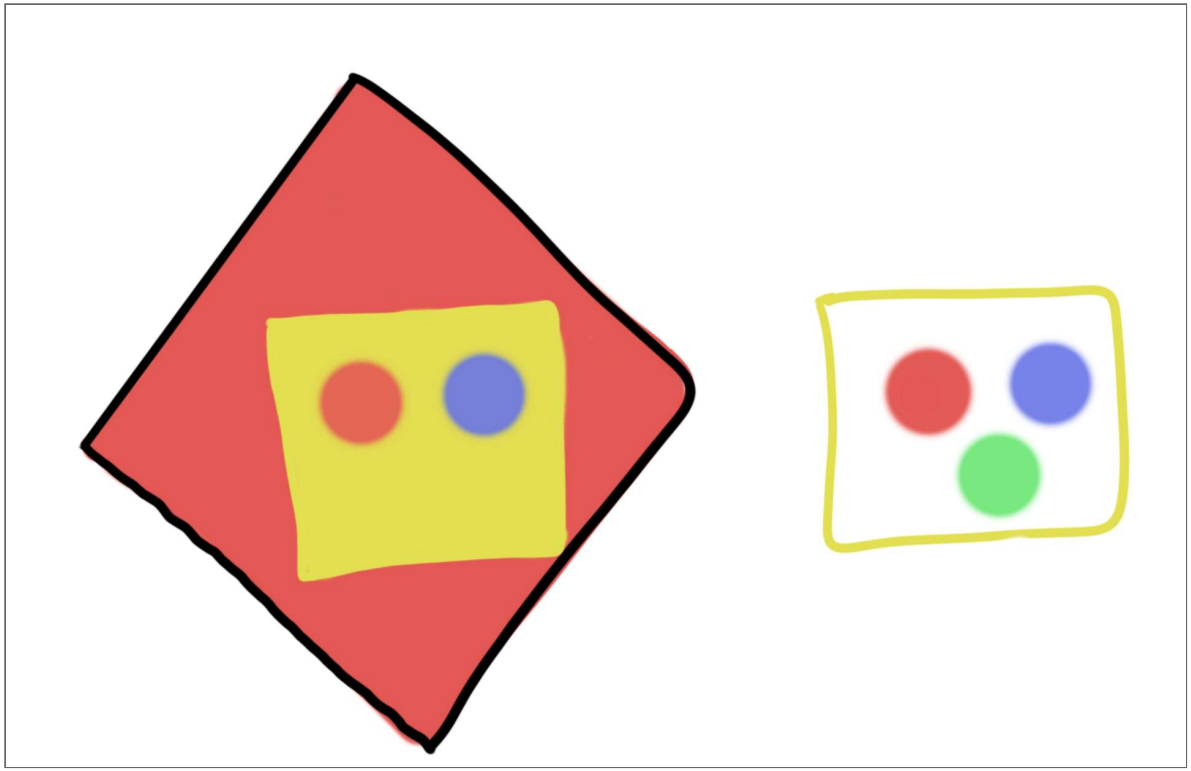
- Usually named as past-tense verbs
- **RARELY** changed
- **Never** deleted
- Has attributes that are values
  - not model, object, collection, or aggregate root


NEVER DELETE EVENTS
DEPOSIT $500
CORRECTION -$500
DEPOSIT $50

DON'T STORE OBJECTS



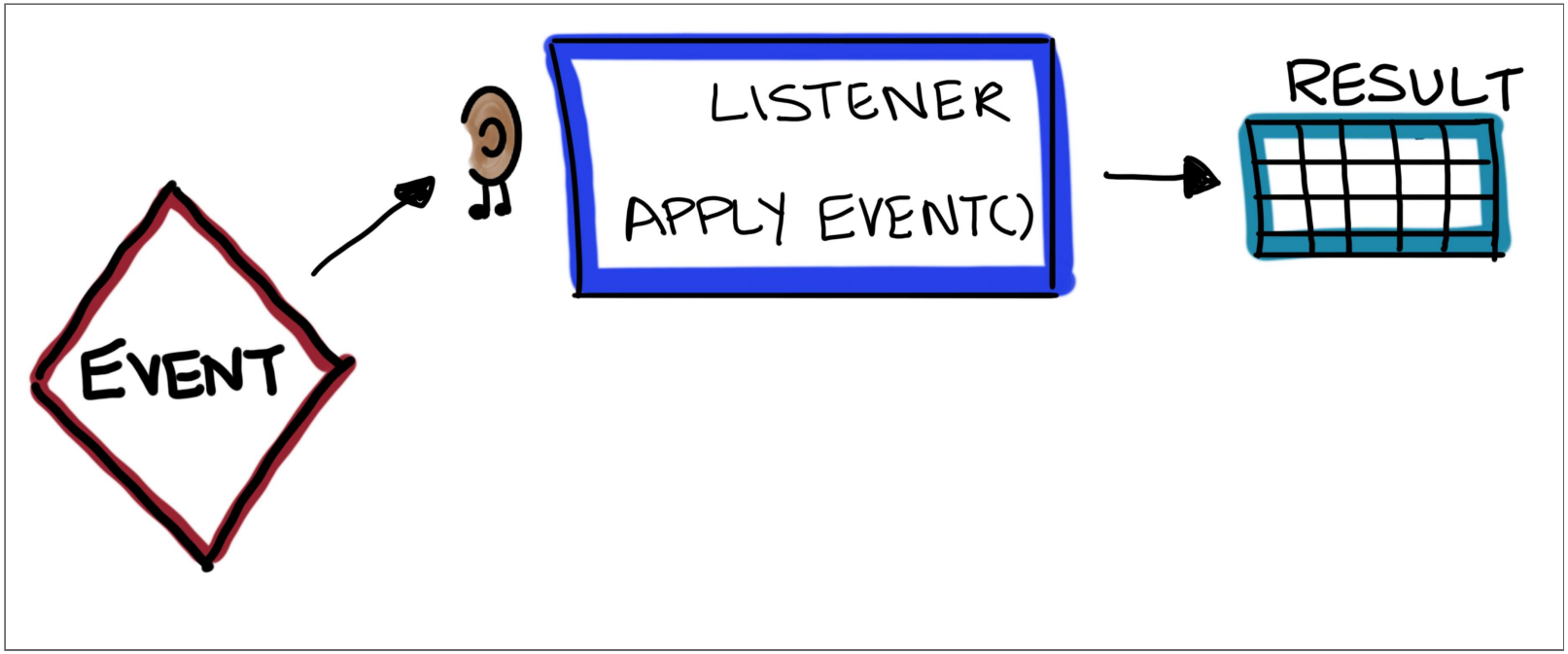
IF WE STORED IT IN AN EVENT...

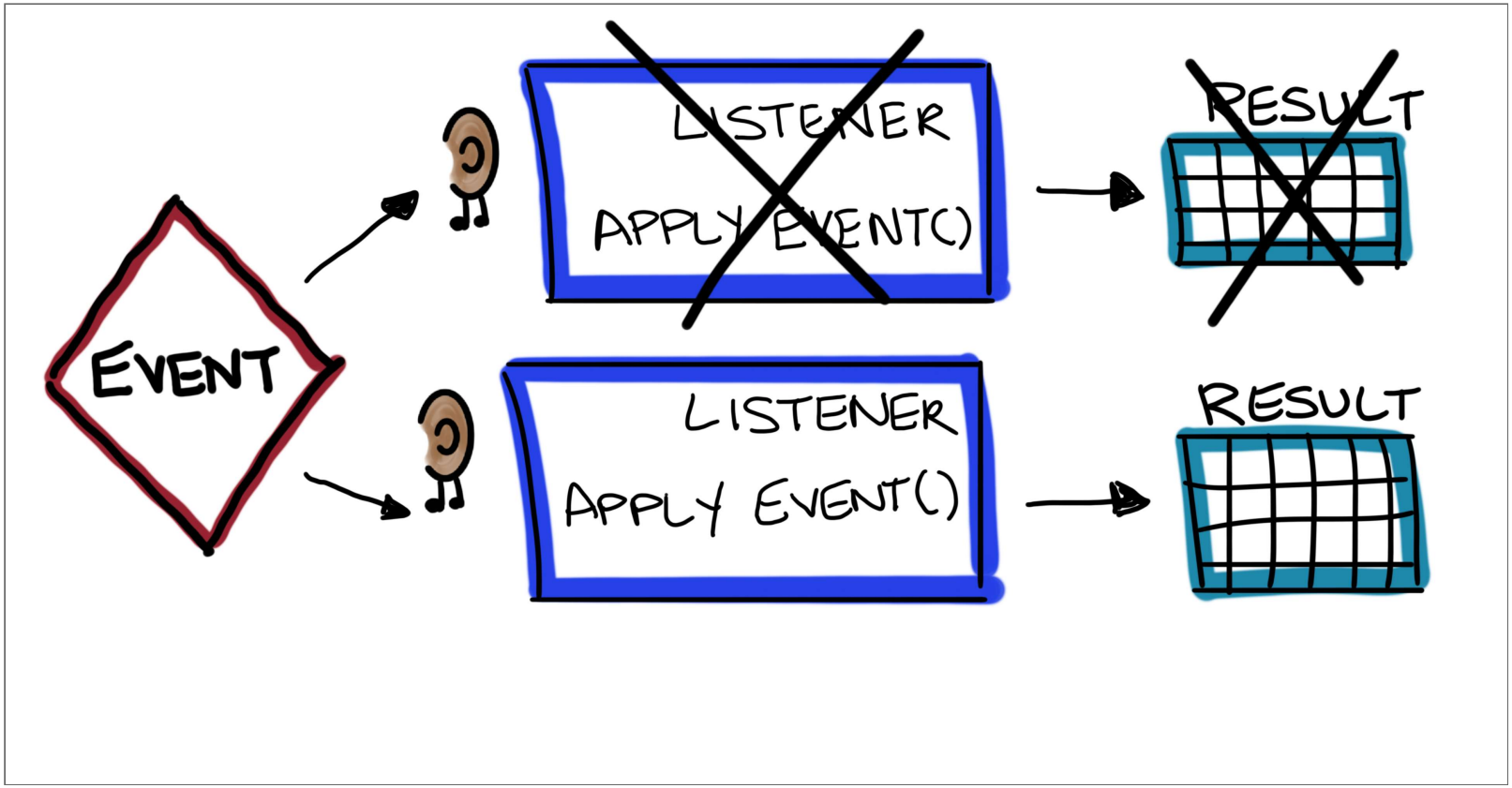


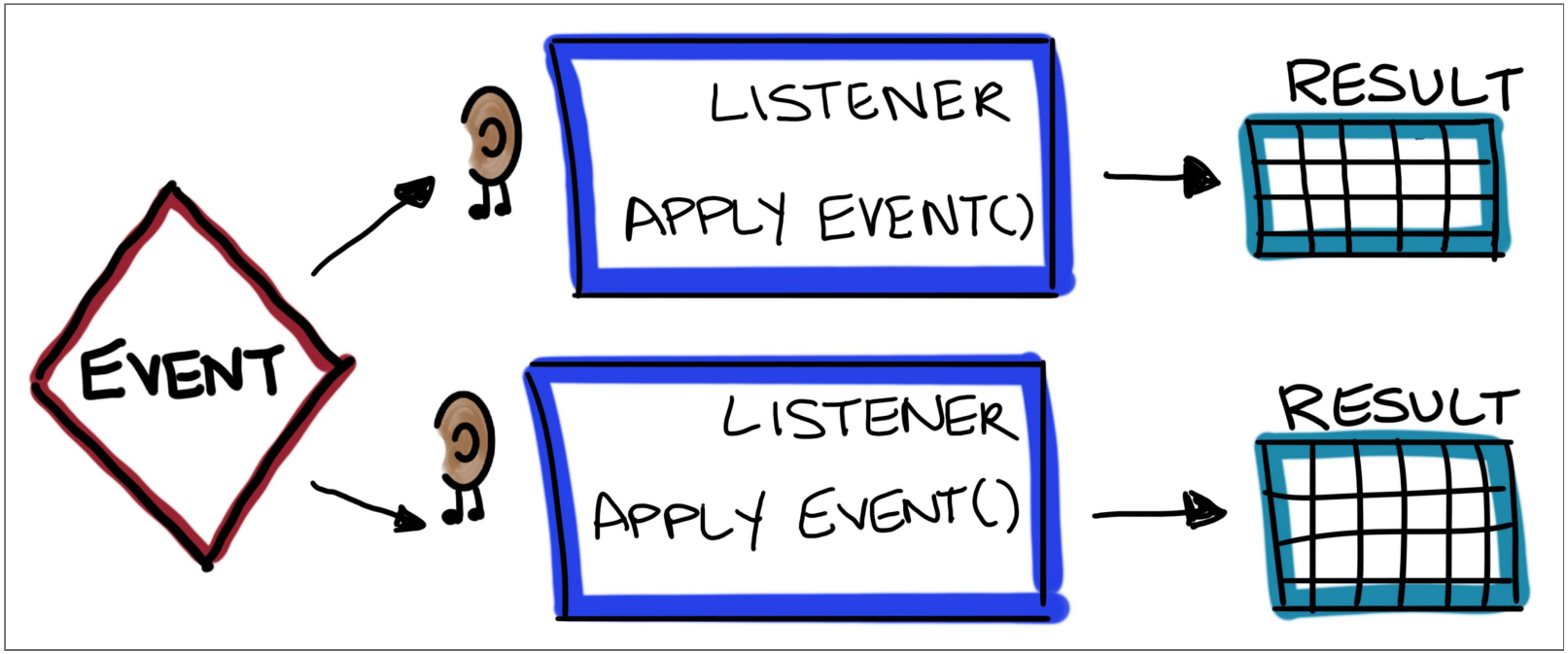
EVENTS ARE NOT BIONIC

EVENTS RARELY CHANGE

- The part of the code that will change is most likely the **result that follows that event**.
- The **structure of the resulting data** is more likely to change than the thing that happened


EVENT
LISTENER
APPLY EVENT()
RESULT


EVENT
LISTENER
APPLY EVENT()
RESULT
LISTENER
APPLY EVENT()
RESULT


EVENT
LISTENER
APPLY EVENT()
RESULT
LISTENER
APPLY EVENT()
RESULT

REASONS TO USE EVENTS

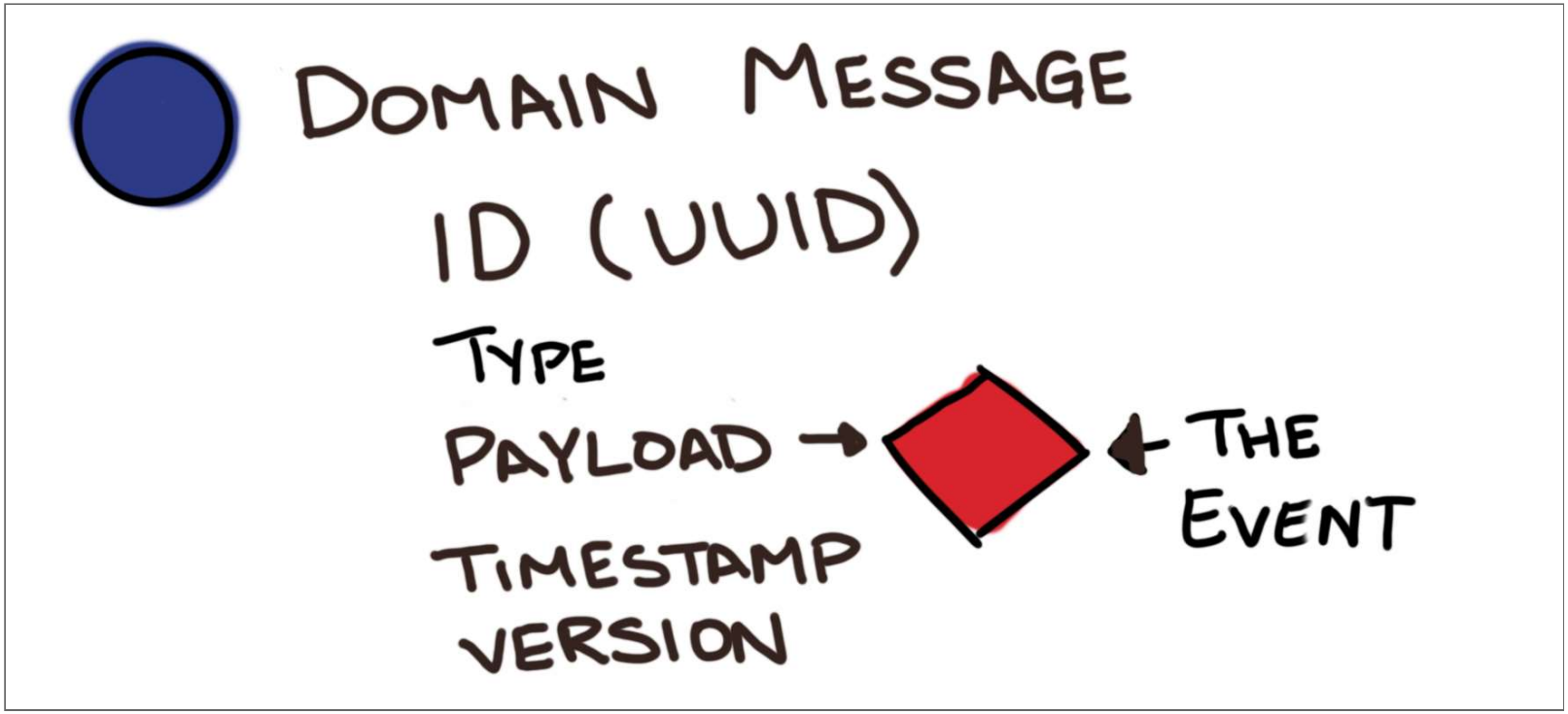
- State transitions are important
- We need an audit log, proof of the state we are currently in
- The history of what happened is more important than the current state
- Events are replayable if behavior in your application changes

EVENT CLASS

```php
<?php

namespace Library\Events;

use Library\Support\Event;

class BookWasCheckedOut
{

    /**
     * @var DateTime
     */
    public $checkoutDate;

    /**
     * @var int
     */
    public $patronId;

    /**
     * @var int
     */
```

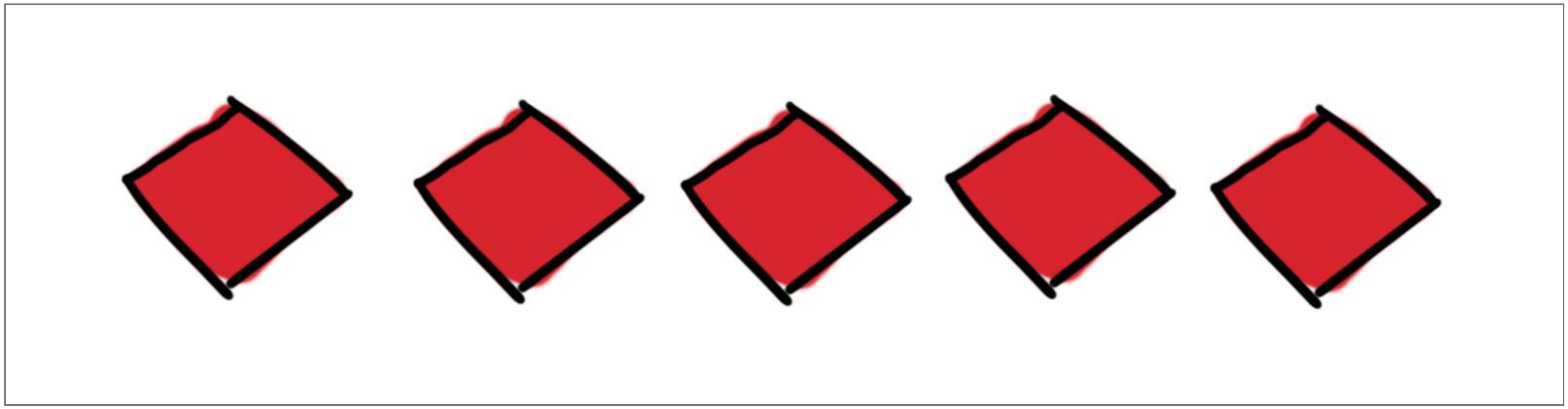
CONNECTING THE EVENTS

- An event is created only after validation
  - directly in a controller 'checkout' method
  - using a Check Out Book Command and Handler


DOMAIN MESSAGE
ID (UUID)
TYPE
PAYLOAD
TIMESTAMP
VERSION
THE EVENT

EVENT STORE

- Domain-specific database
- Based on a Publish-Subscribe message pattern

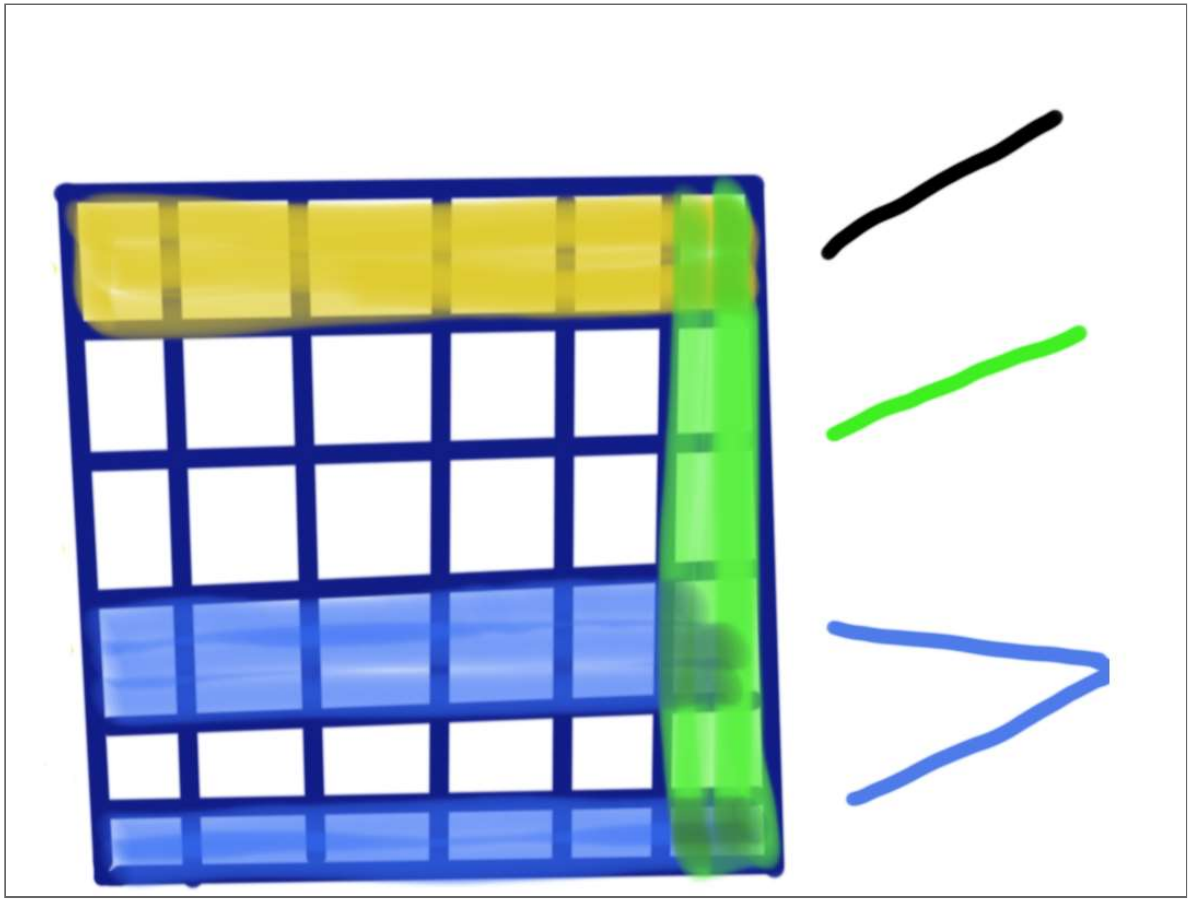


PROJECTOR

```php
<?php

namespace Library\ReadModel;

use Library\Events\BookWasCheckedIn;
use Library\Events\BookAddedToBookshelf;
use App\Support\ReadModel\Replayable;
use App\Support\ReadModel\SimpleProjector;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Connection;

class BookshelfProjector extends SimpleProjector implements Replayable
{

    /**
     * @var Connection
     */
    private $connection;

    /**
     * @var string table we're playing events into
     */
    private $table = 'proj_bookshelf';

    public function __construct(Connection $connection)
```

A set of event handlers that work together to build and maintain a table to be accessed by the read model.

READ MODEL



READ MODEL

```php
<?php

namespace Library\ReadModel;


use Carbon\Carbon;
use Illuminate\Database\Eloquent\Model;

/**
* @codeCoverageIgnore
*/
class Bookshelf extends Model
{
    protected  $table = 'proj_bookshelf';
    public $incrementing = false;
    public $timestamps = false;

    public static function lookupLoansFor($patronId)
    {
        return static::where('patron_id', $patronId)->get();
    }

    public function lookupAvailableBooks()
```

CRUD: UPDATE

```
public function update(Request $request, $id)
{
    // our default update method
        // validate inputs
    $Book = Book::findOrFail($id);
    $Book->update($request->all());

    return $Book;
}
```

CRUD CHECKOUT

```
public function checkOutBook(Request $request, $i
{
    // altered the update method
        // validate inputs
    $Book = Book::findOrFail($id);
    $Book->update('status' => 'checked out',
                  'patron' => $request->patronI

    return $Book;
}
```

CRUD CHECKOUT

```
public function checkOutBook(Request $request, $id)
{
    // altered the update method
    // validate book can be checked out

    $event = new BookWasCheckedOut($request->bookId,
                                   $request->patronId,
                                   time());

}
```

CQRS

Command and Query Response Segregation

- **Command** is any method that mutates state
- **Query** is any method that returns a value
- These methods become part of Services
- When parts of your application no longer fit a CRUD model
- Should only be used on specific portions of a system, not the system as a whole

CRUD TO CQRS

```
public function update(Request $request)
{
    // $request has book id, patron id

    try {

        $command = new CheckOutBook($request->bookId, $request->patronId);

        $this->bookLendingService->handleCheckOutBook($command);

    } catch (InvalidUserException $e) {
        return response()->json("Not authorized to request enrollment.", Response::HTTP_FOR
    } catch (BookUnavailableException $e) {
       return response()->json("Book was not available to be checked out", 400);


    return $Book;
}

<------ dynamic command handler  ------>

private function handle(Command $command)
{
```

COMMAND HANDLER

A command handler receives a command and brokers a result from the appropriate aggregate. "A result" is either a successful application of the command, or an exception.

should affect one and only one aggregate

COMMAND HANDLER

1. Validate the command on its own merits.
2. Validate the command on the current state of the aggregate.
3. If validation is successful, create an event(s)
4. Attempt to persist the new events. If there's a concurrency conflict during this step, retry or exit.

OPTIMIZE

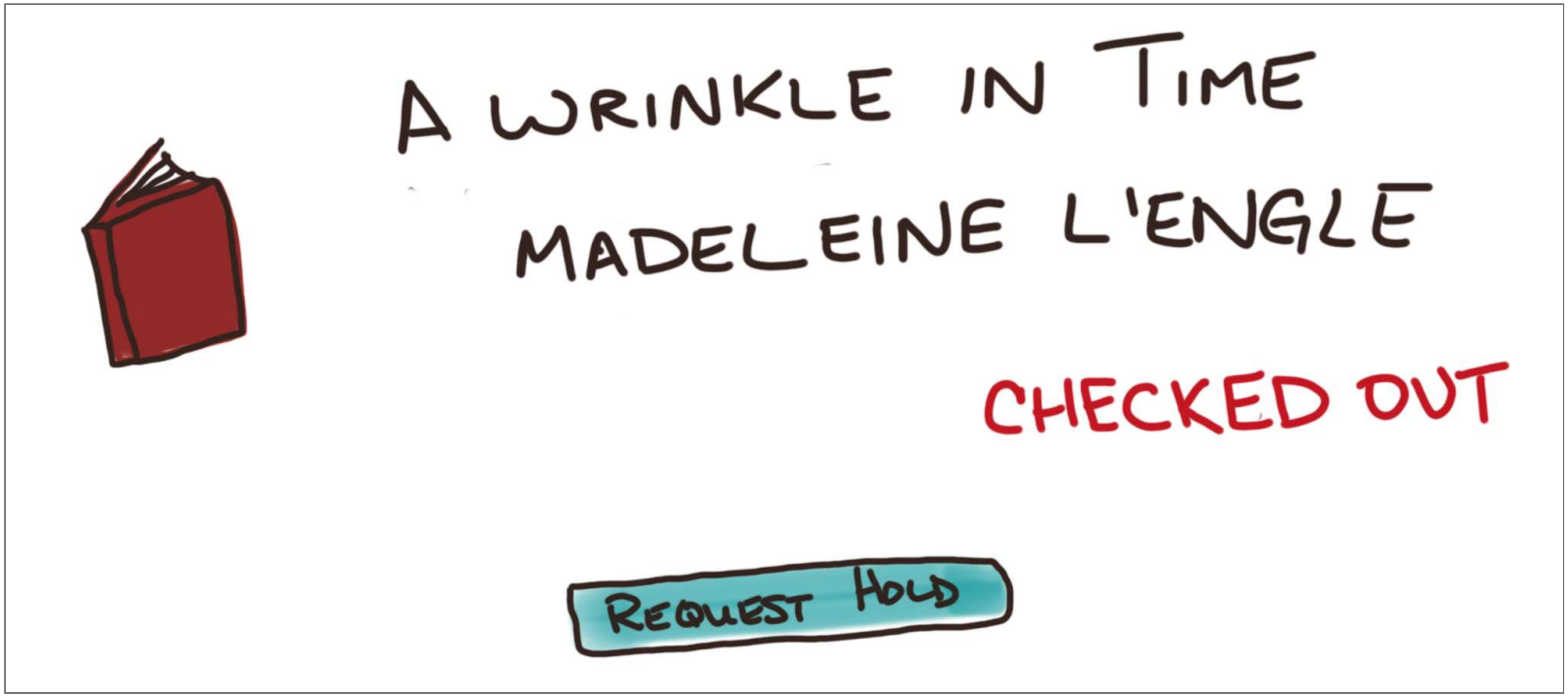
- all **commands** go into a WriteService
- all **queries** go into a ReadService
- to optimize your application for reads and writes
- Separate the load from reads and writes allowing you to scale each independently.
  - *If your application sees a big disparity between reads and writes this is very handy.*
- Uses a separate model for all queries

TASK-BASED UI

- Track what the user is doing and push forward commands representing the intent of the user
- CQRS **does not require** a task based UI (DDD does)
- CQRS used in a CRUD interface, makes creating separated data models harder

LIBRARY

- May not be a purely task-based UI
- I don't need to optimize for load

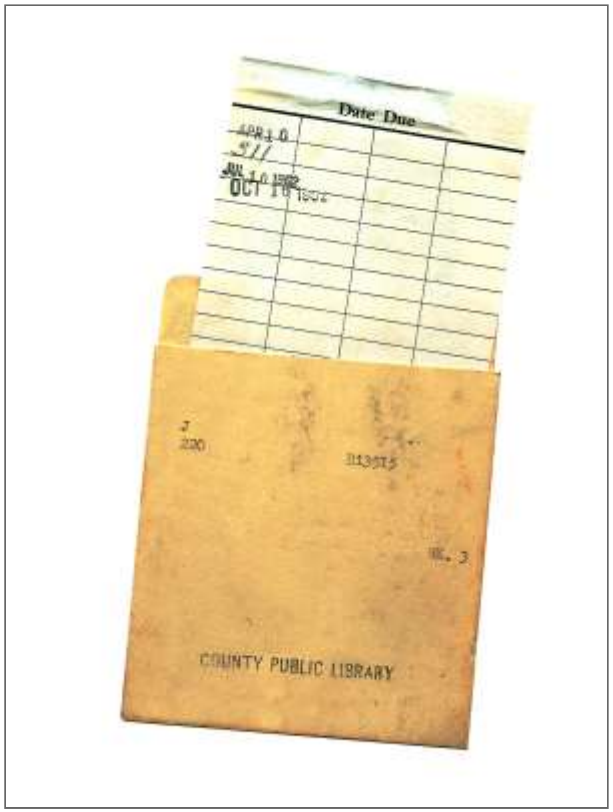

A WRINKLE IN TIME
MADELEINE L'ENGLE
CHECKED OUT
REQUEST HOLD

BOOKLENDINGSERVICE

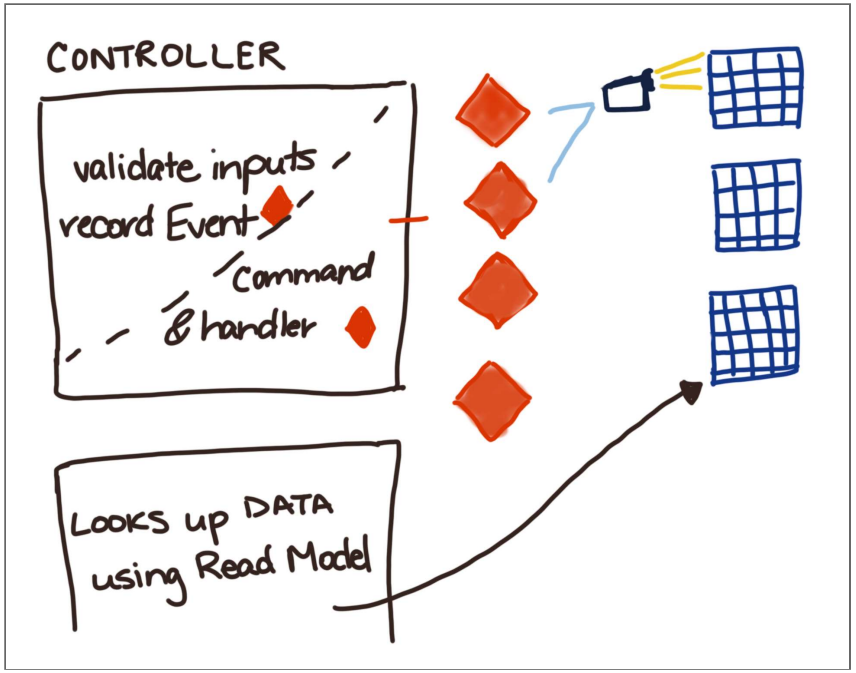
- CheckOutBook
- CheckInBook
- RequestBook

FLEXIBILITY GAINED LONG TERM

- Aren't locked into the current interpretation of events
- Can track the events and build more views of the data and add functionality later
- Financial reports, can show budget balance on any given day in the past, prove how you got the current balance
- Could display everyone who checked out a book, and style it like old cards in the book pockets


Date Due
APR 1 0
OCT 10
J
220
COUNTY PUBLIC LIBRARY

ADDING EVENT SOURCING TO YOUR LEGACY APP



RESOURCES

- Tools
  - **Event Sauce - Event-sourcing & Commands**
  - **Prooph - CQRS and ES**
  - **Broadway - framework for CQRS and ES**

RESOURCES

- **CQRS by Martin Fowler**
- **CQRS by Greg Young**
- videos
  - **Greg Young CQRS and Event Sourcing**
  - **Greg Young - long class**
  - **Greg Young - A Decade of DDD, CQRS, Event Sourcing**
- podcasts
  - **PHP Round Table - Event Sourcing** (+2 more)

THANK YOU!

EMILY STAMEY

@ELSTAMEY

HTTPS://WWW.ELSTAMEY.COM/

@elstamey