

From OOP to OOPS

Escaping the One-Paradigm Trap

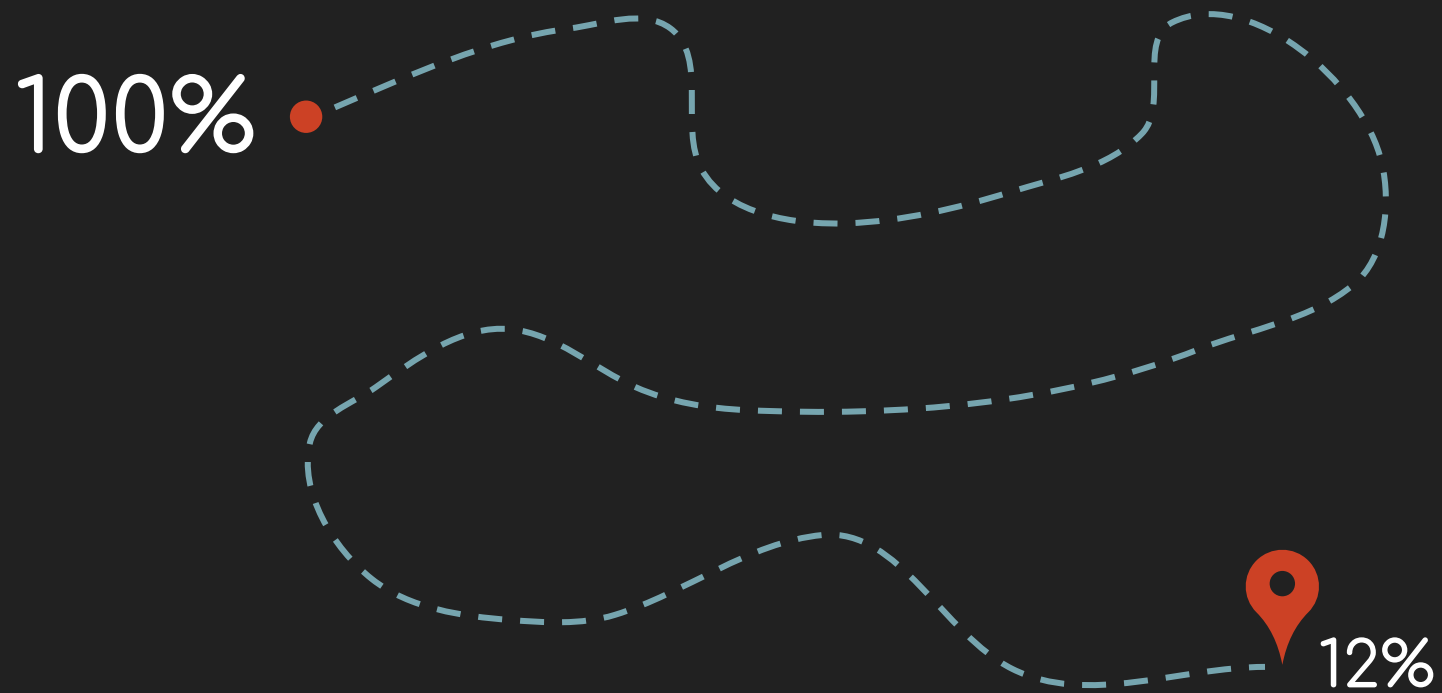


Masking Technology
hello@masking.tech
[linkedin.com/company/maskingtechnology](https://www.linkedin.com/company/maskingtechnology)



Object-Oriented
Programming
is great!

But using it
for everything
is not.



6' 8" | 203cm



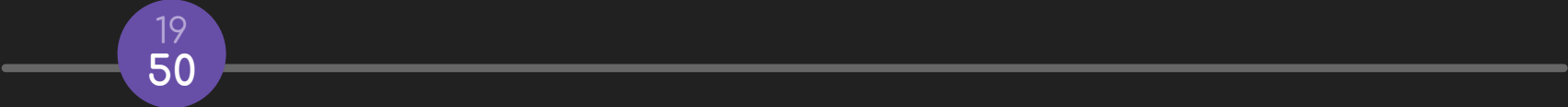
Peter van Vliet

- Co-founder Masking Technology
- Software architect
- Full-stack developer
- 20+ years of experience
- Java, C#, PHP, TypeScript
- Hobby illustrator (don't hire me)

Mainstream Paradigms

Mainstream Paradigms

Imperative



19
50

Sequence of
statements

Mainstream Paradigms

Imperative

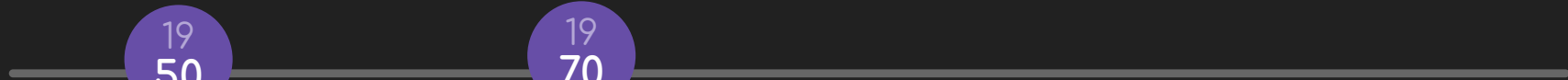
Procedural

19
50

19
70

Sequence of
statements

Reusable
sequences



Mainstream Paradigms

Imperative

Procedural

Object-Oriented

19
50

19
70

19
90

Sequence of
statements

Reusable
sequences

Combine data
and behavior

Mainstream Paradigms

Imperative

19
50

Sequence of
statements

Procedural

19
70

Reusable
sequences

Object-Oriented

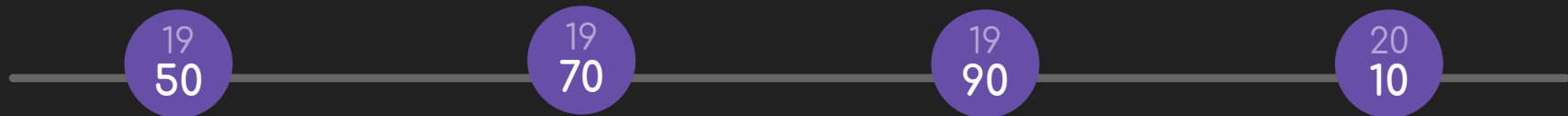
19
90

Combine data
and behavior

Functional

20
10

Reduce side
effects

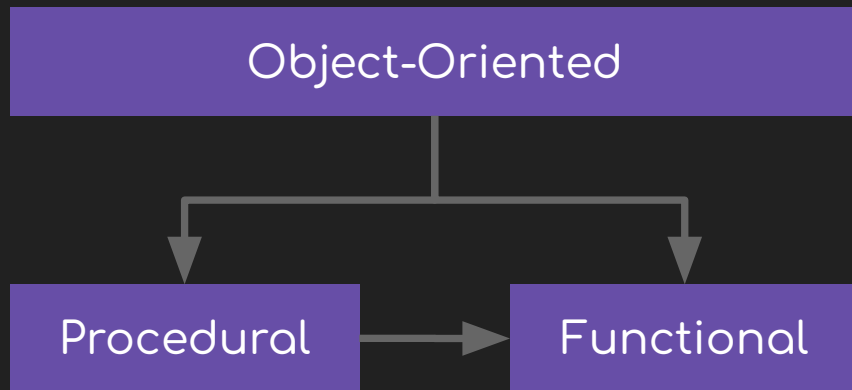


Modern Multi-Paradigm Synthesis

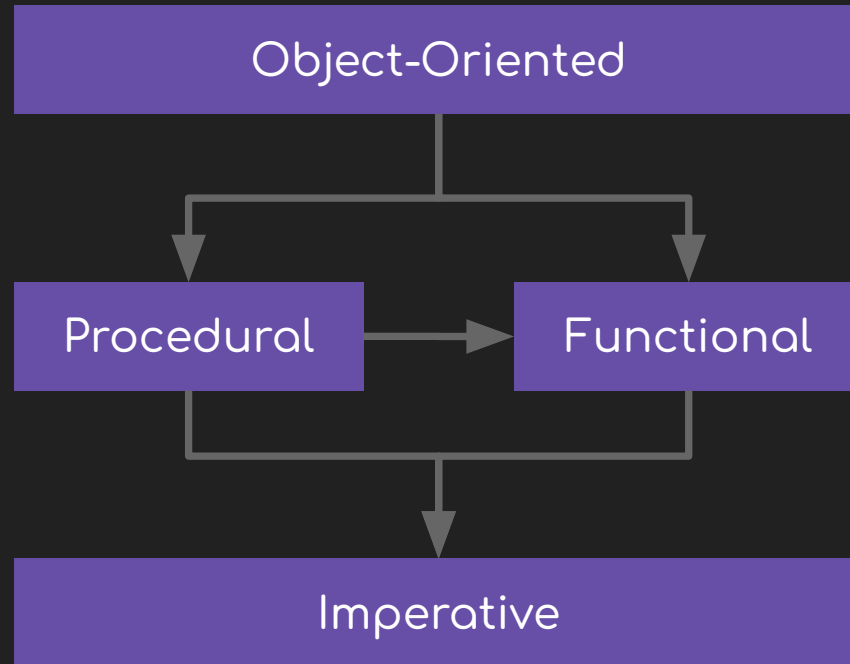
Modern Multi-Paradigm Synthesis

Object-Oriented

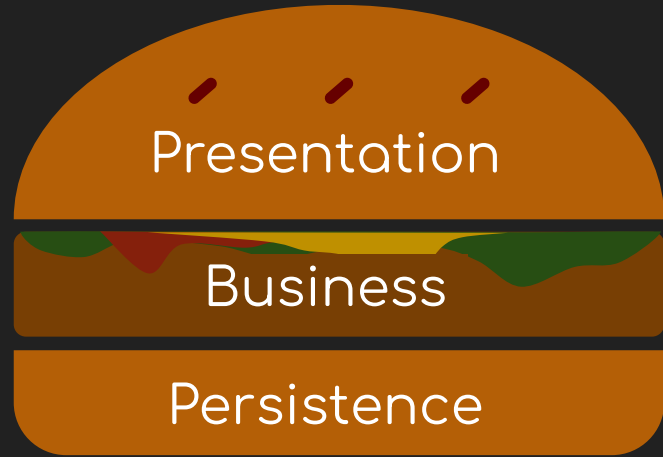
Modern Multi-Paradigm Synthesis



Modern Multi-Paradigm Synthesis



Common web architecture



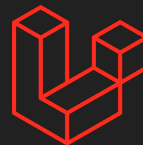


nest

► Presentation

► Business

► Persistence



Laravel

► Presentation

► Business

► Persistence

▼ Presentation burger.controllers.ts

```
@Controller('burgers')
export class BurgerController
{
    constructor(
        private service: BurgerService
    ) {}

    @Get('names')
    getnames(): string[]
    {
        return service.getNames();
    }
}
```

► Business burger.service.ts

► Persistence burger.entity.ts

▼ Presentation BurgerController.php

```
class BurgerController extends
Controller
{
    public function __construct(
        protected service $burgerService
    ) {}

    public function getNames(): array
    {
        return $this->service->getNames();
    }
}

Route::get('/burgers/names', [...]);
```

► Business BurgerService.php

► Persistence Burger.php

► **Presentation** burger.controllers.ts

▼ **Business** burger.service.ts

```
@Injectable()
export class BurgerService
{
    constructor(
        private repository:
            Repository<Burger>
    ) {}

    getNames(): string[]
    {
        return this.repository.find()
            .map(burger => burger.name);
    }
}
```

► **Persistence** burger.entity.ts

► **Presentation** BurgerController.php

▼ **Business** BurgerService.php

```
class BurgerService
{
    public function __construct() {}

    public function getNames(): array
    {
        $burgers = Burger::all()->all();

        return array_map(
            fn ($burger) => $burger->name,
            $burgers
        );
    }
}
```

► **Persistence** Burger.php

► **Presentation** burger.controllers.ts

► **Business** burger.service.ts

▼ **Persistence** burger.entity.ts

```
@Entity()
export class Burger
{
  @Column()
  name: string;
}
```

► **Presentation** BurgerController.php

► **Business** BurgerService.php

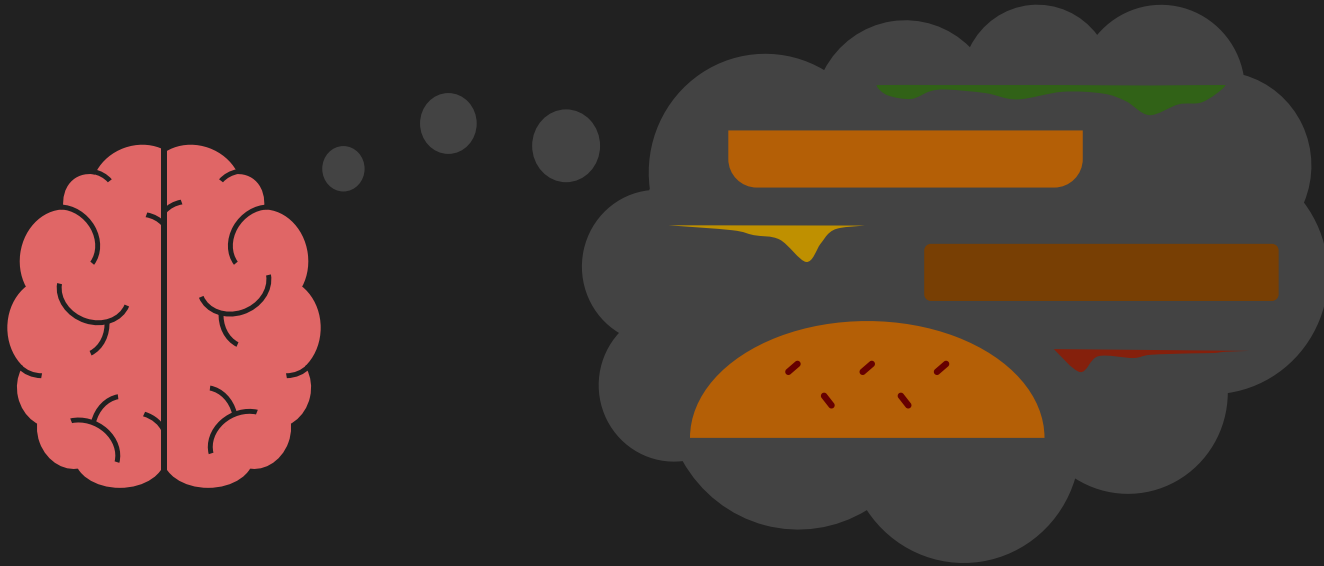
▼ **Persistence** Burger.php

```
class Burger extends Model
{
    protected $fillable = ['name'];
}
```

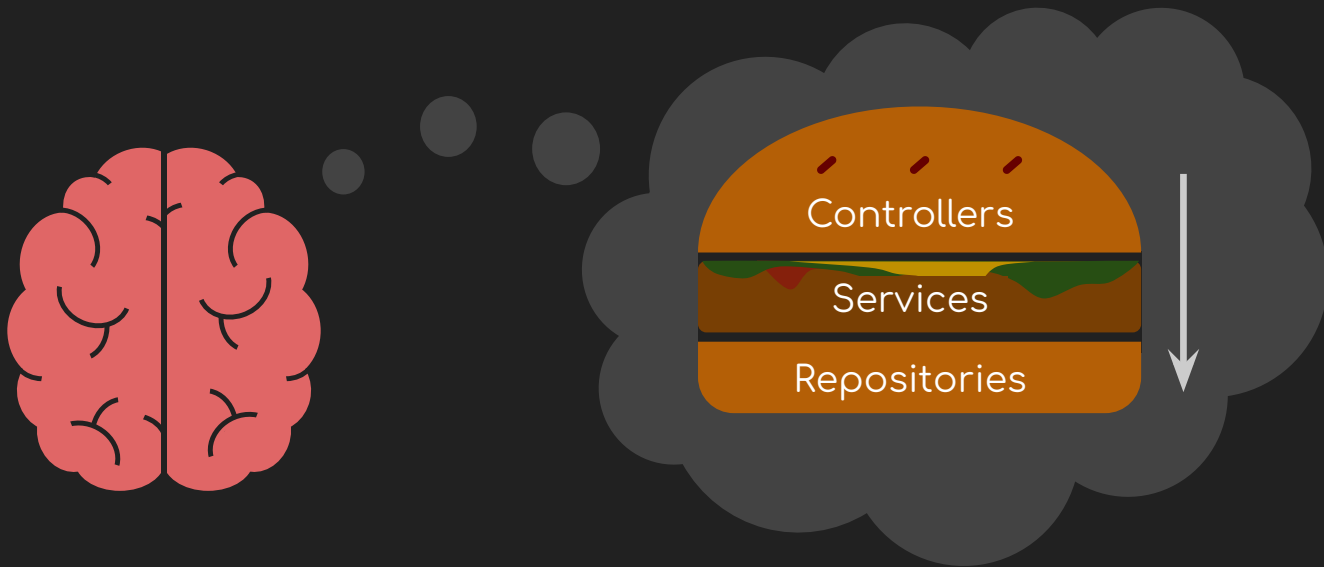
This is an established
pattern that works.

But...

Cognitive Load



Mental Model



Paradigm Ladder

mental model

cognitive load

Paradigm Ladder

mental model

Imperative
Sequence of statements

Procedural
Reusable sequences

OO
Data $\leftrightarrow$ behavior

Functional
Reduce side effects

cognitive load

Paradigm Ladder

mental model

Imperative
Sequence of statements

Procedural
Reusable sequences



Most complex
paradigm

OO
Data $\leftrightarrow$ behavior

- Classes & objects
- State management
- Abstractions
- Encapsulation
- Inheritance
- Polymorphism

Functional
Reduce side effects

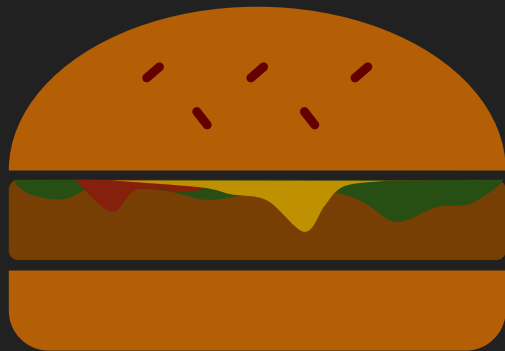
cognitive load

*"Complexity is a measure of
understandability, and lack of
understandability leads to errors."*

*On the Relationship between Software Complexity
and Maintenance Costs (2014)*

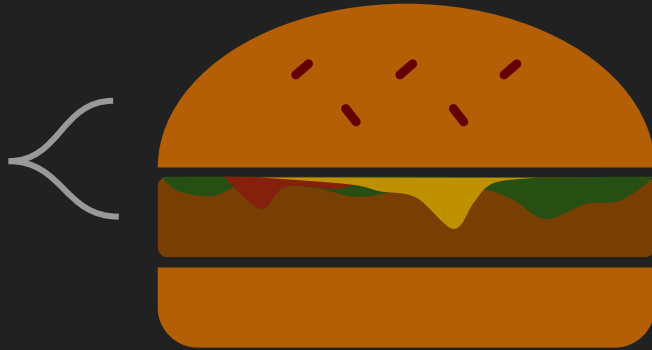
Essential or Accidental
Complexity?

App Assessment



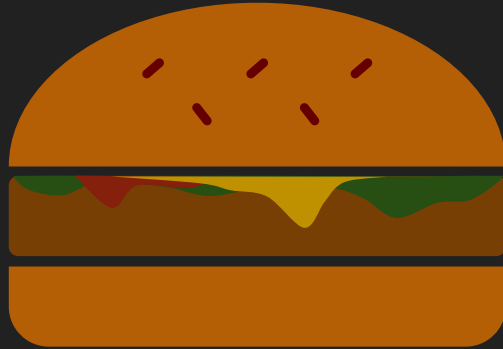
App Assessment

Stateless
Singletons



App Assessment

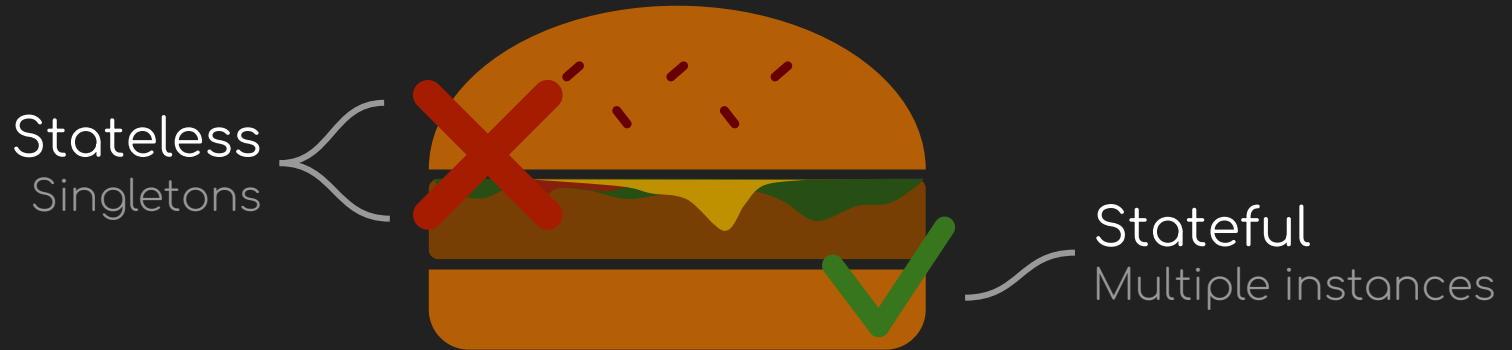
Stateless
Singletons



Stateful
Multiple instances



Partial Misalignment



Alignment

mental model

Imperative
Sequence of statements

Procedural
Reusable sequences

OO
Data $\leftrightarrow$ behavior

Functional
Reduce side effects

cognitive load

Alignment

mental model

Stateful logic



OO

Data $\leftrightarrow$ behavior

Functional

Reduce side effects

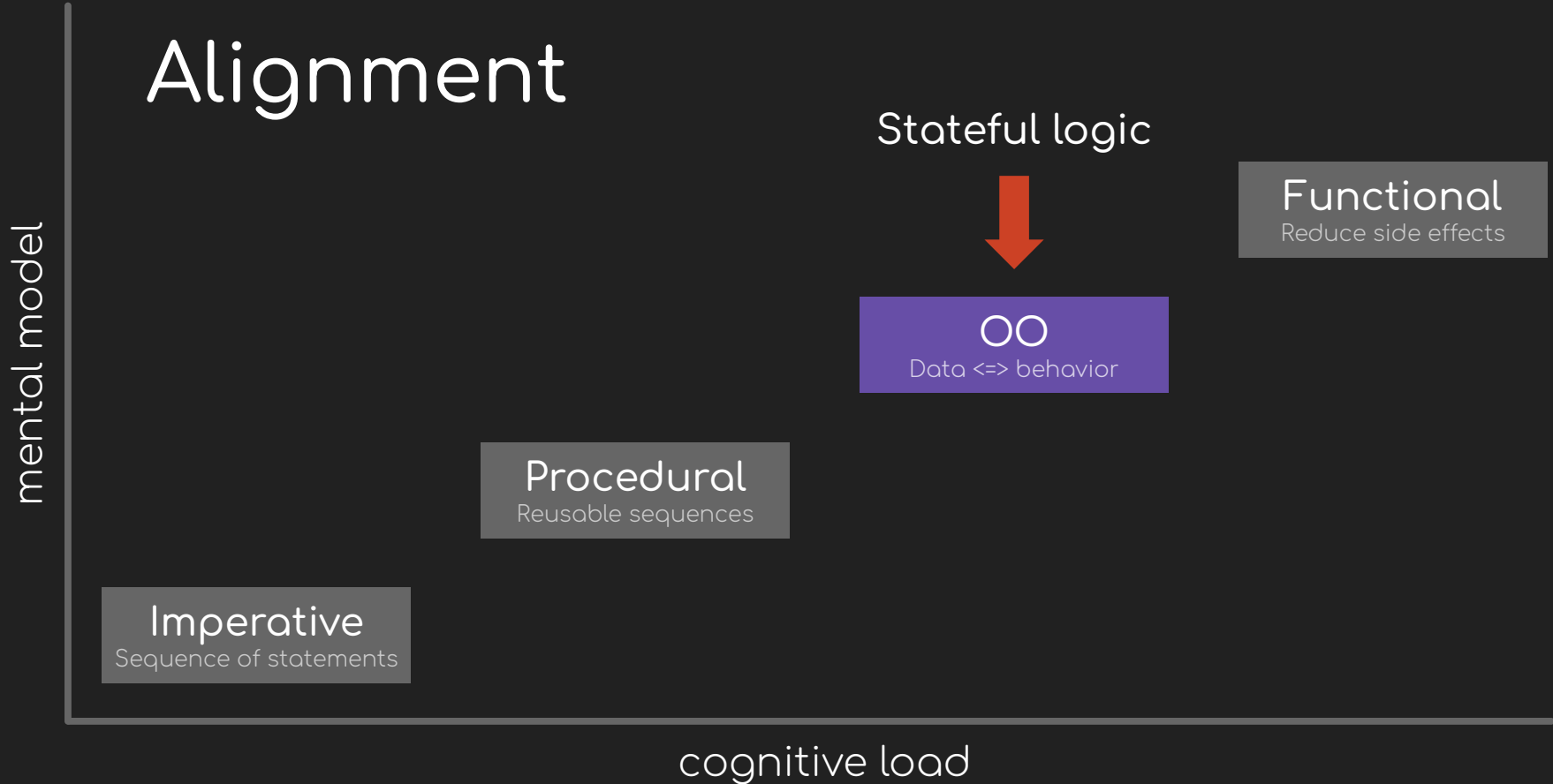
Procedural

Reusable sequences

Imperative

Sequence of statements

cognitive load



Alignment

mental model

Stateless logic



Procedural
Reusable sequences

Stateful logic

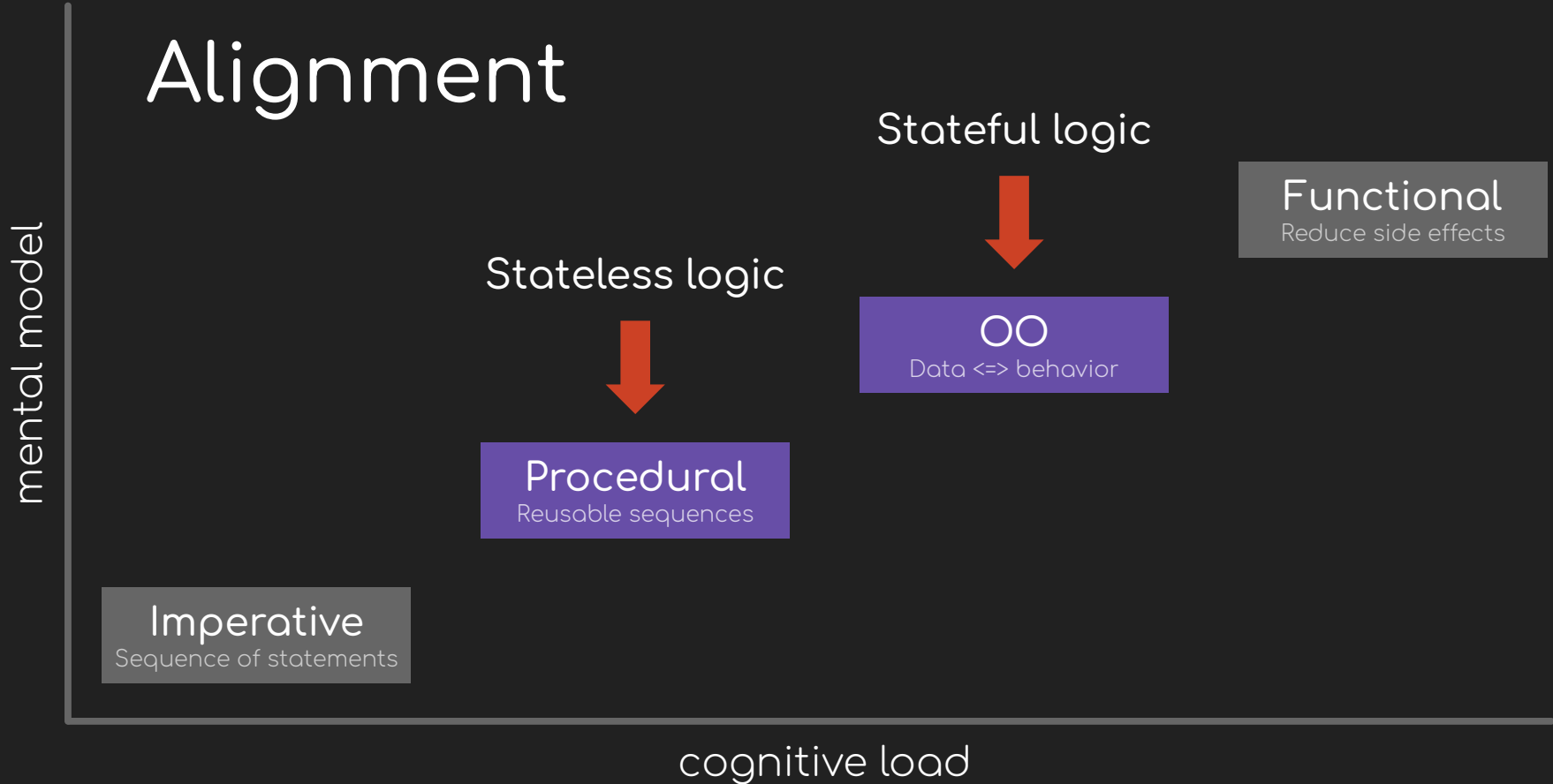


OO
Data $\Leftrightarrow$ behavior

Functional
Reduce side effects

Imperative
Sequence of statements

cognitive load



But what about bootstrapping?



Alignment

mental model

Bootstrapping



Imperative

Sequence of statements

Stateless logic



Procedural

Reusable sequences

Stateful logic



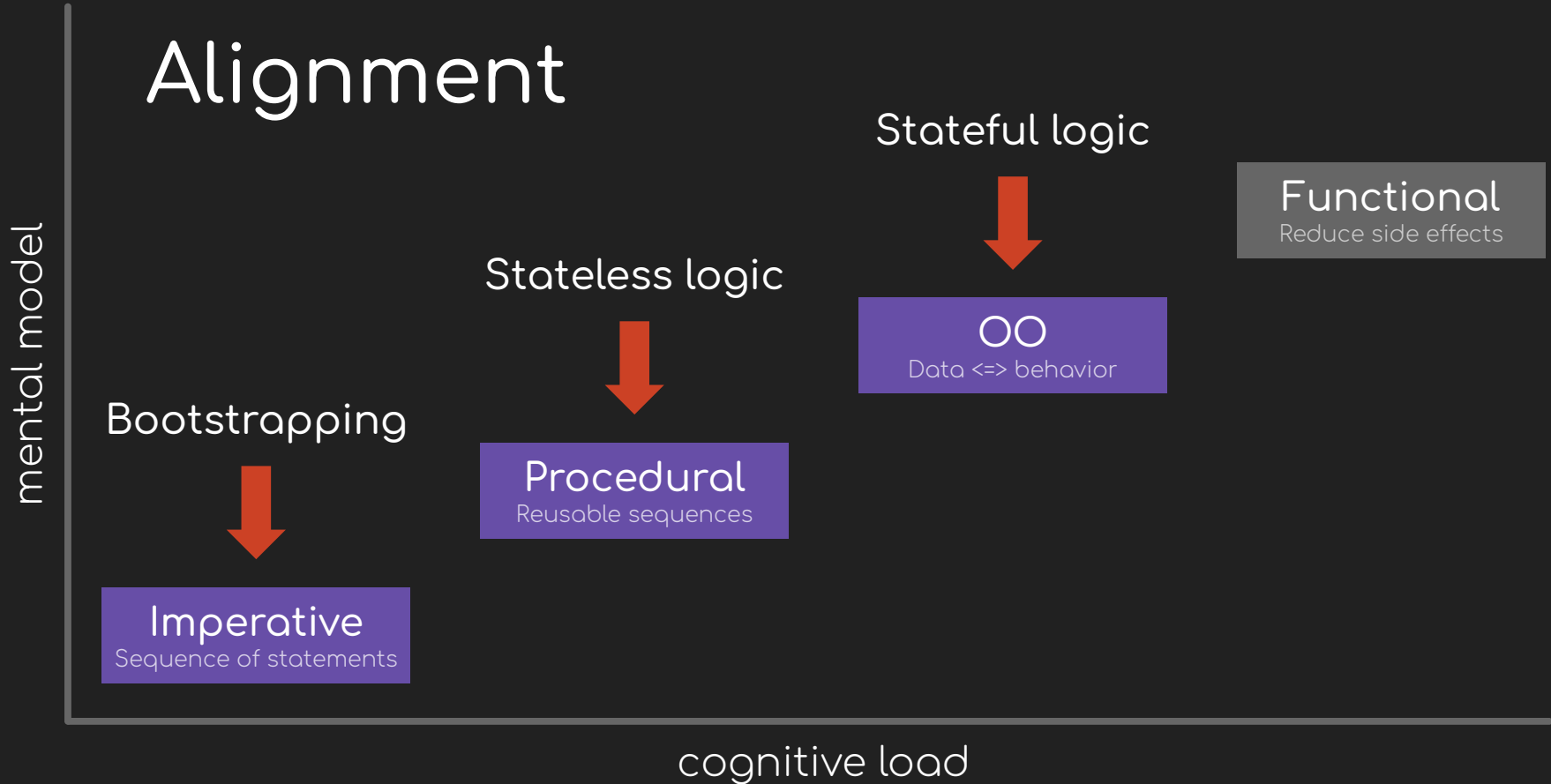
OO

Data $\Leftrightarrow$ behavior

Functional

Reduce side effects

cognitive load



Paradigm Fit

Paradigm Fit

Imperative

Simple tasks bootstrapping, configuration, migration, etc.

Paradigm Fit

Imperative

Simple tasks bootstrapping, configuration, migration, etc.

Procedural

Processes and flows application, business and persistence logic.

Paradigm Fit

Imperative

Simple tasks bootstrapping, configuration, migration, etc.

Procedural

Processes and flows application, business and persistence logic.

OO

State $\Leftrightarrow$ behaviour UI components, complex data models, etc.

Paradigm Fit

Imperative

Simple tasks bootstrapping, configuration, migration, etc.

Procedural

Processes and flows application, business and persistence logic.

OO

State $\Leftrightarrow$ behaviour UI components, complex data models, etc.

Functional

Helpers data processing, calculations, concurrency, etc.



Top-level
availability
required

Language support is limited.





Time for
Refactoring

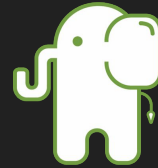


express

► Presentation

► Business

► Persistence



Slim Framework

► Presentation

► Business

► Persistence

▼ Presentation route.ts

```
app.get('/burgers/names', (
  request: Request, response: Response)
=> {
  const names = getBurgerNames();

  response.json(names);
});
```

► Business getBurgerNames.ts

► Persistence Burger.ts

▼ Presentation route.php

```
$app->get('/burgers/names', function (
  Request $request, Response $response
){
  $names = getBurgerNames();

  $response->getBody()
    ->write(json_encode($names));

  return $response
    ->withHeader('Content-Type',
      'application/json')
    ->withStatus(200);
});
```

► Business getBurgerNames.php

► Persistence Burger.php

► Presentation route.ts

▼ Business getBurgerNames.ts

```
function getBurgerNames(): string[]  
{  
  const burgers = retrieveBurgers();  
  
  return burgers  
    .map(burger => burger.name);  
}
```

► Persistence Burger.ts

► Presentation route.php

▼ Business getBurgerNames.php

```
function getBurgerNames(): array  
{  
  $burgers = retrieveBurgers();  
  
  return array_map(  
    fn ($burger) => $burger->name,  
    $burgers  
  );  
}
```

► Persistence Burger.php

► **Presentation** route.ts

► **Business** getBurgerNames.ts

▼ **Persistence** retrieveBurgers.ts

```
function retrieveBurgers(): Burger[]
{
    // Query the database
    // Map the results
}
```

```
class Burger
{
    name: string;
}
```

► **Presentation** route.php

► **Business** getBurgerNames.php

▼ **Persistence** retrieveBurgers.php

```
function retrieveBurgers(): array
{
    // Query the database
    // Map the results
}
```

```
class Burger
{
    public string $name;
}
```

Simplicity!



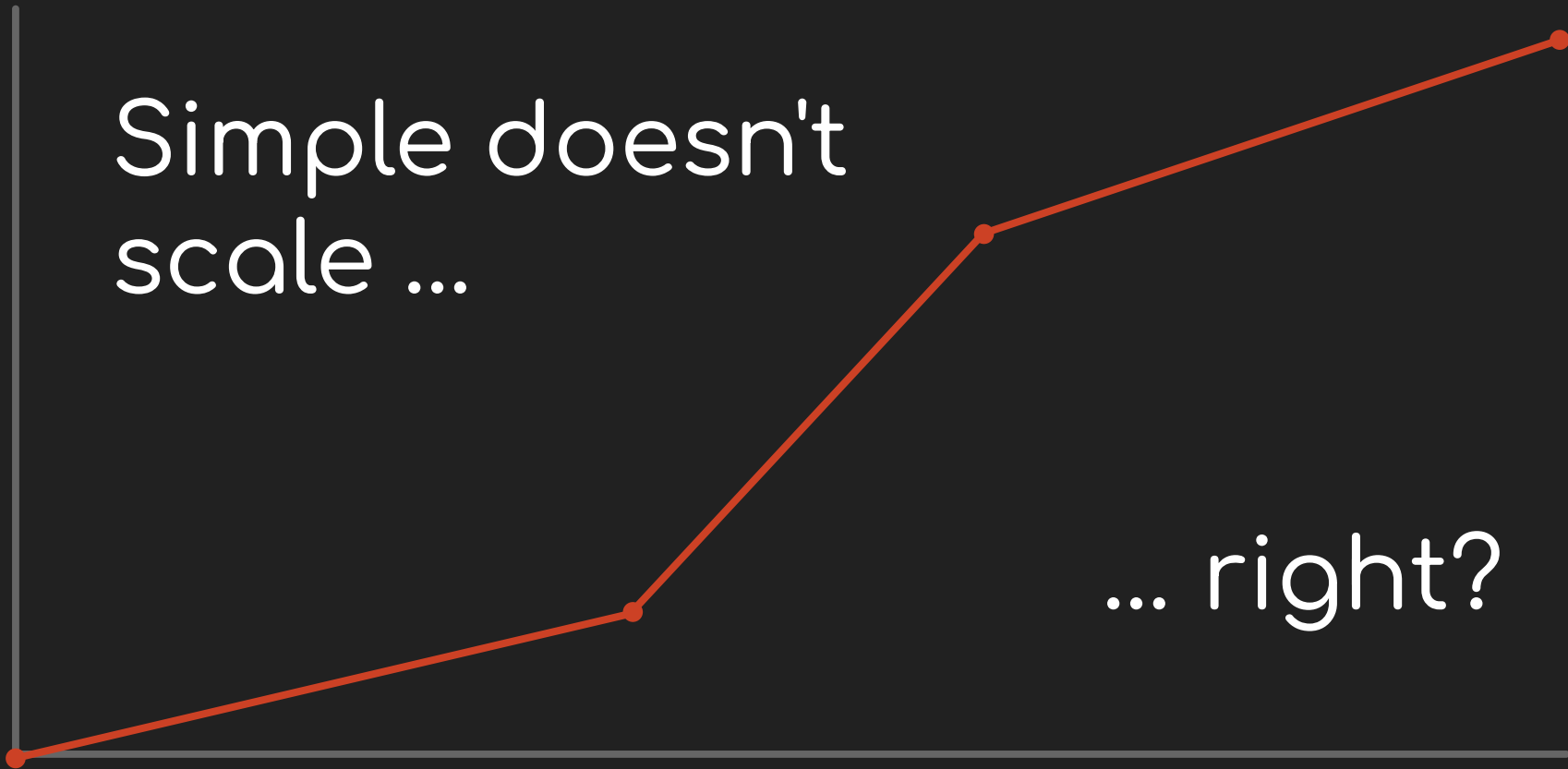
But...

Simple doesn't
scale ...

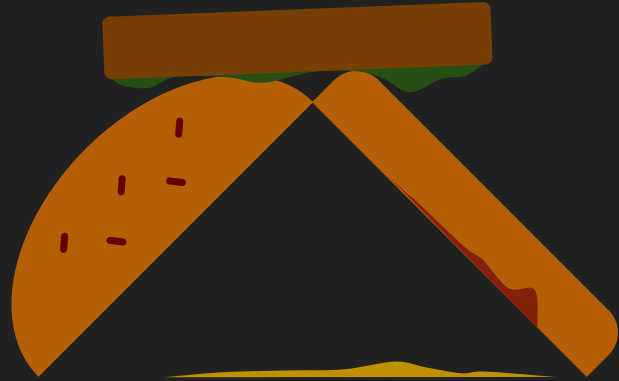
application size

... right?

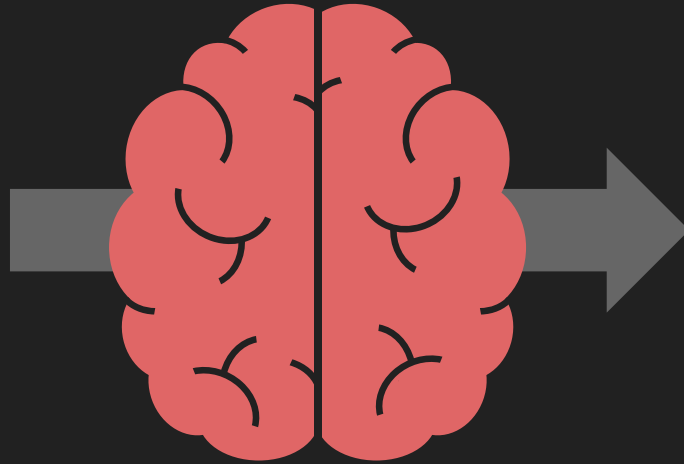
essential complexity



It's all about proper
Application
Structure.

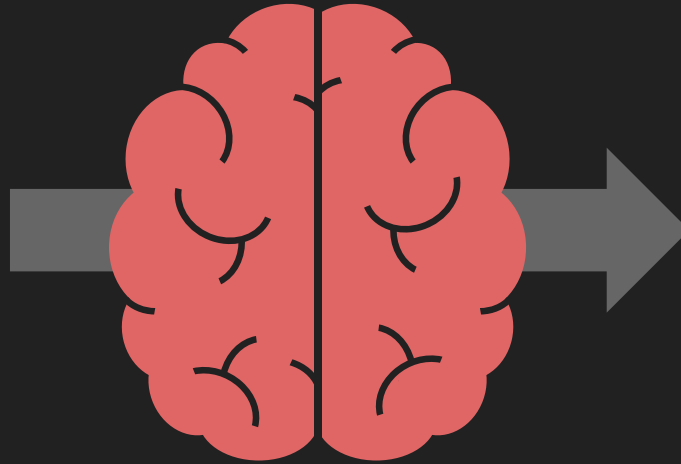


Mindset Shift



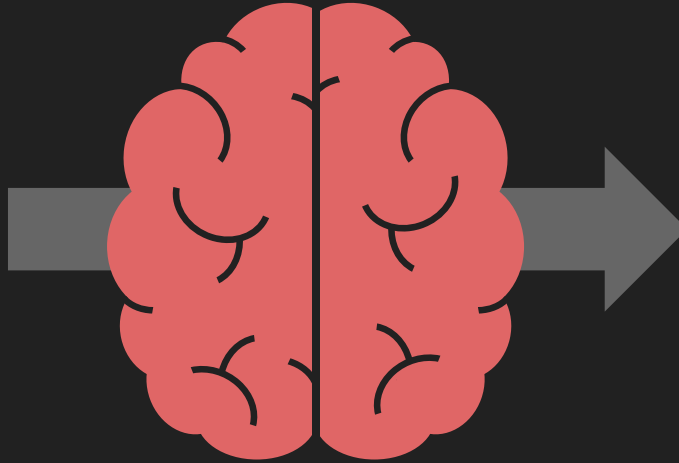
Mindset Shift

Taxonomy
(technical identity)



Mindset Shift

Taxonomy
(technical identity)



Topography
(business intent)

Taxonomy

app root

controllers layer

BurgerController

services layer

BurgerService

repositories layer

BurgerRepository

models layer

Burger

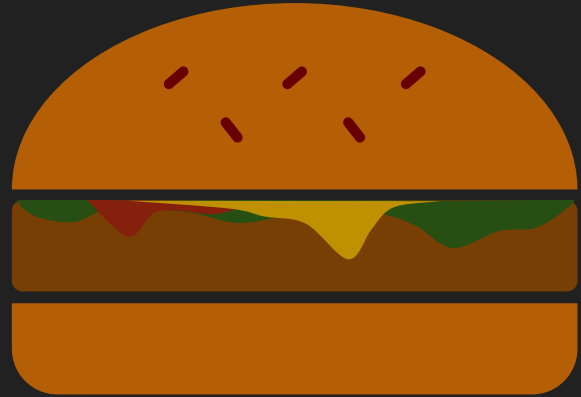
Taxonomy



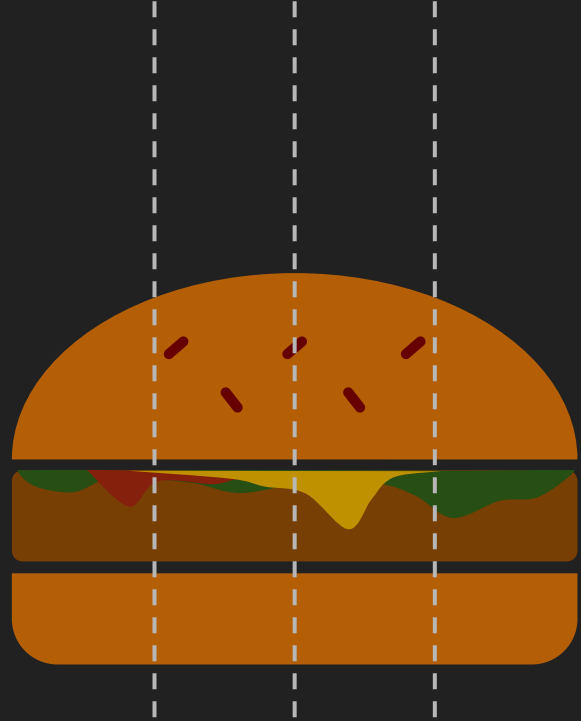
Topography



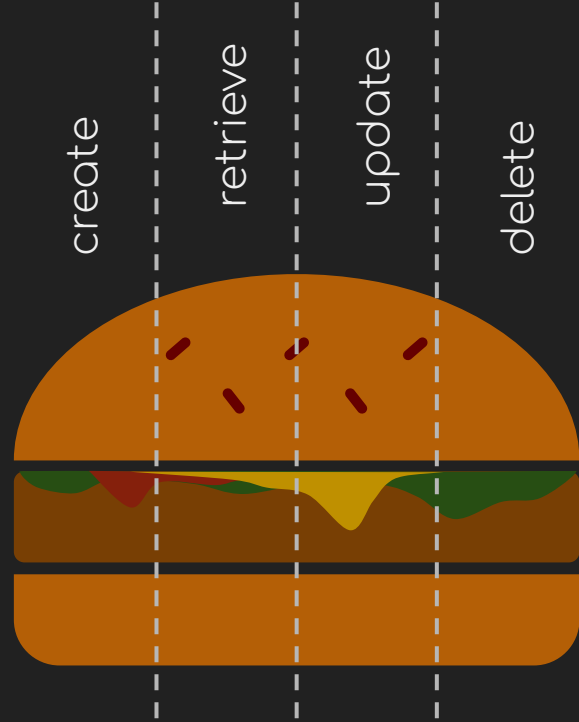
Vertical Slice Architecture



Vertical Slice Architecture



Vertical Slice Architecture



Object-Oriented



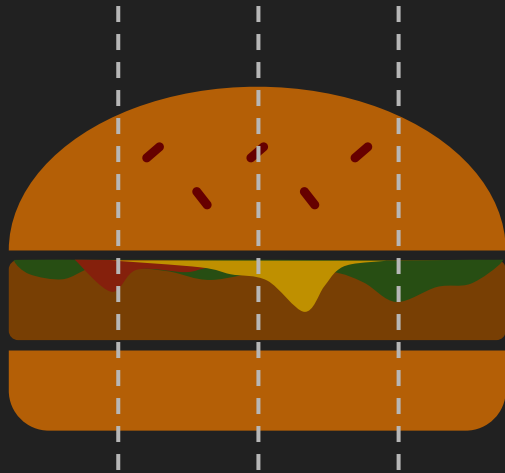
Object-Oriented



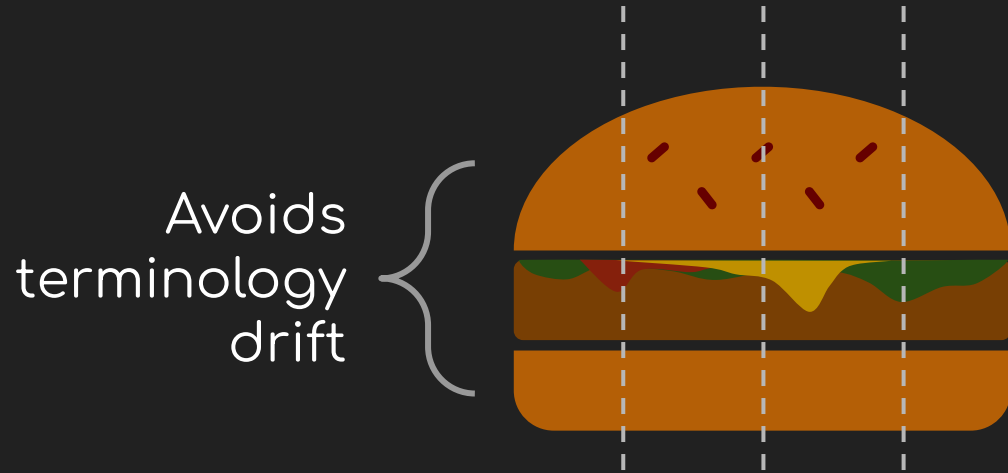
Blended Approach



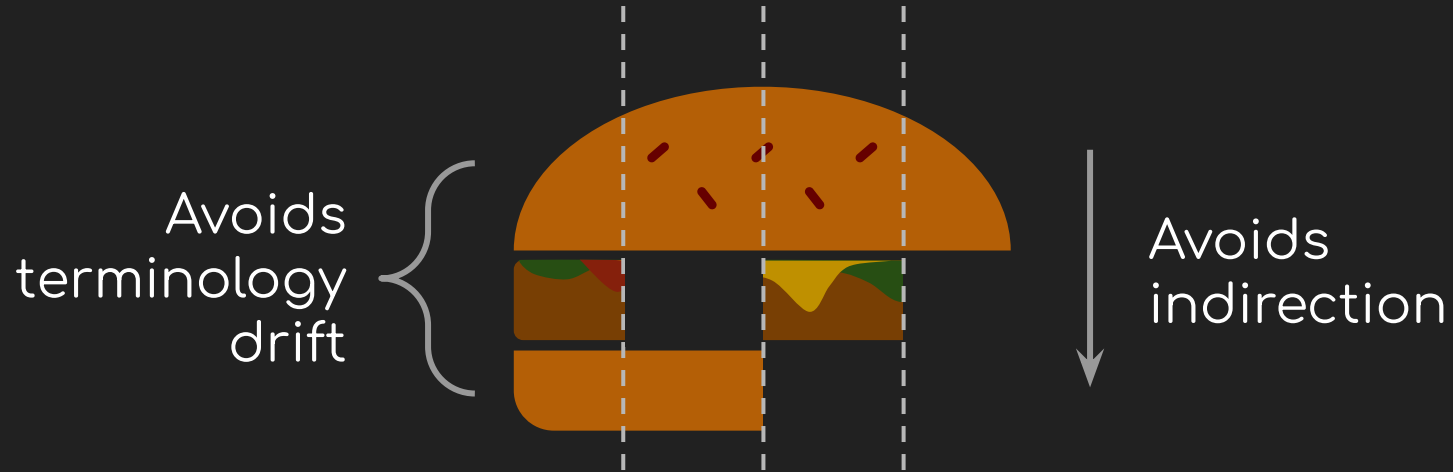
Cognitive Load Decrease



Cognitive Load Decrease



Cognitive Load Decrease



What
About
Testing?



Testing strategies

Testing strategies

Imperative

End-to-End run and verify.

Testing strategies

Imperative

End-to-End run and verify.

Procedural

Unit & Integration per process / flow.

Testing strategies

Imperative

End-to-End run and verify.

Procedural

Unit & Integration per process / flow.

OO

Unit & Integration per class / composition.

Testing strategies

Imperative

End-to-End run and verify.

Procedural

Unit & Integration per process / flow.

OO

Unit & Integration per class / composition.

Functional

Mostly implicit tested in the containing procedure or class.

Pragmatic approach

Imperative

End-to-End run and verify.

Procedural

Unit & Integration per process / flow.

Feature based



OO

Unit & Integration per class / composition.

Functional

Mostly implicit tested in the containing procedure or class.

▼ Feature getNames.spec.ts

```
import
  { beforeEach, describe, it }
from 'vitest';

import { getNames } from '...';

beforeEach(() =>
{
  // reset database
});

describe('getNames', () =>
{
  it('should work', () =>
  {
    // conduct test
  }
});
```

▼ Feature GetNamesTest.php

```
use PHPUnit\Framework\TestCase;

use function App\Burger\getNames;

class GetNamesTest extends TestCase
{
    protected function setUp(): void
    {
        parent::setUp();

        // reset database
    }

    public function testItWorks(): void
    {
        // conduct test
    }
}
```

Don't we need
Dependency
Injection?



Houston ...



But there is an
alternative.



▼ Wiring database.ts

```
import { Database } from 'some-orm';

function setUpPostgres(): Database {
  // set up postgres
}

function setUpInMemory(): Database {
  // set up in memory
}

const driver = process.env.DB_DRIVER;
const database = driver === 'postgres'
  ? setUpPostgres()
  : setUpInMemory();

export default database;
```

```
import database from '../db';
```

▼ Wiring database.php

```
use SomeOrm\Database;

function setUpPostgres(): Database {
  // set up postgres
}

function setUpInMemory(): Database {
  // set up in memory
}

$driver = $_ENV['DB_DRIVER'] ?? null;
$database = $driver === 'postgres'
  ? setUpPostgres()
  : setUpInMemory();

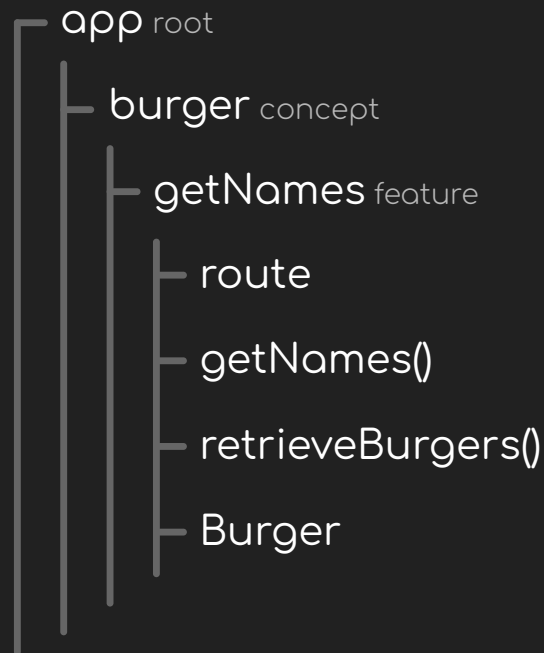
return $database; // avoid global scope
```

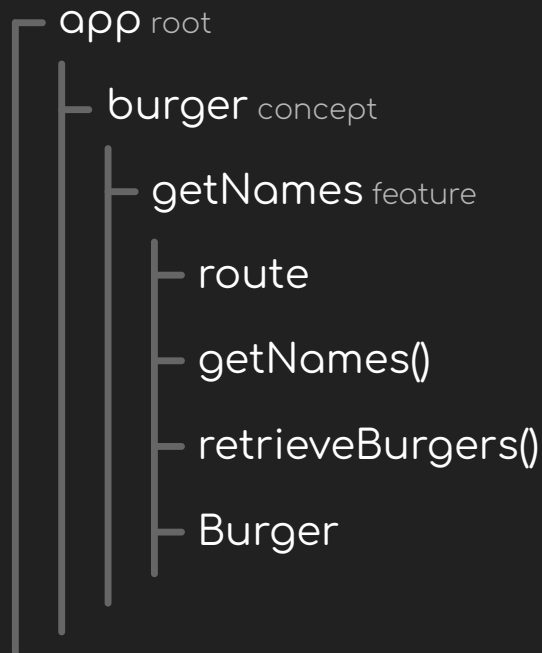
```
$database = require_once '../db.php';
```

Burger Time!









25%

Object-Oriented

EXAMPLE

COMIFY

Comic based social
media platform.



[View on GitHub](#)

COMIFY

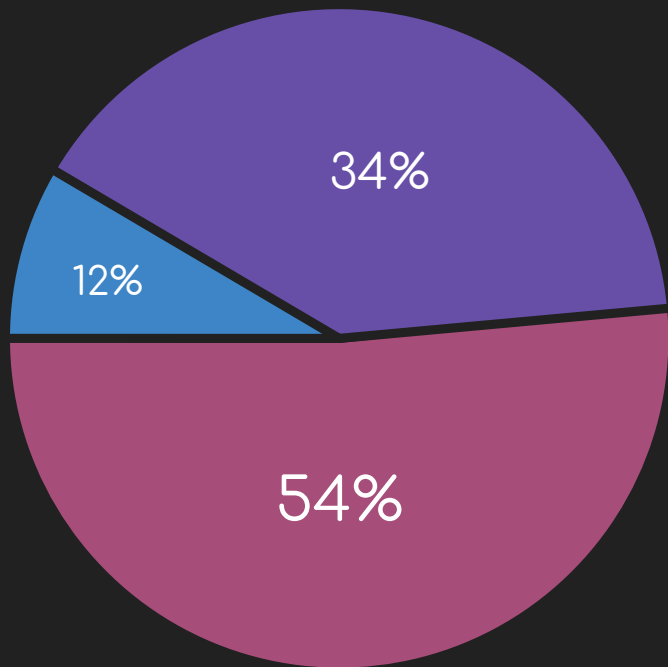
 Object-Oriented

 Procedural

 Imperative



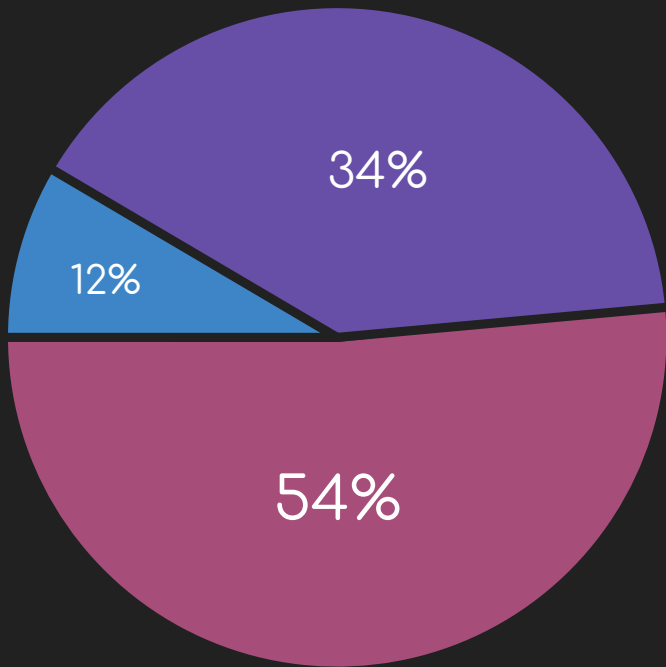
COMIFY



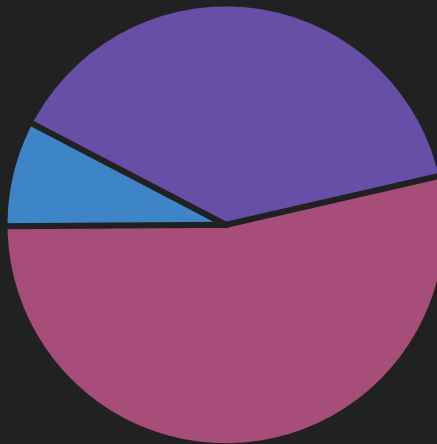
- Object-Oriented
- Procedural
- Imperative



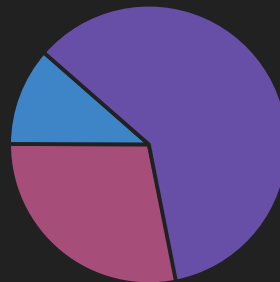
COMIFY

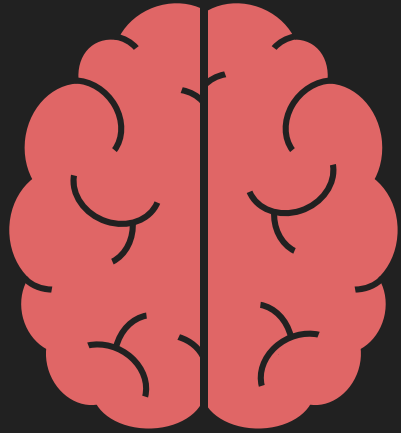


Domain
92 features



Integration
11 implementations





Use paradigms
where they fit best.

Thanks!



<https://masking.tech>