From OOP to OOPS

Escaping the One-Paradigm Trap



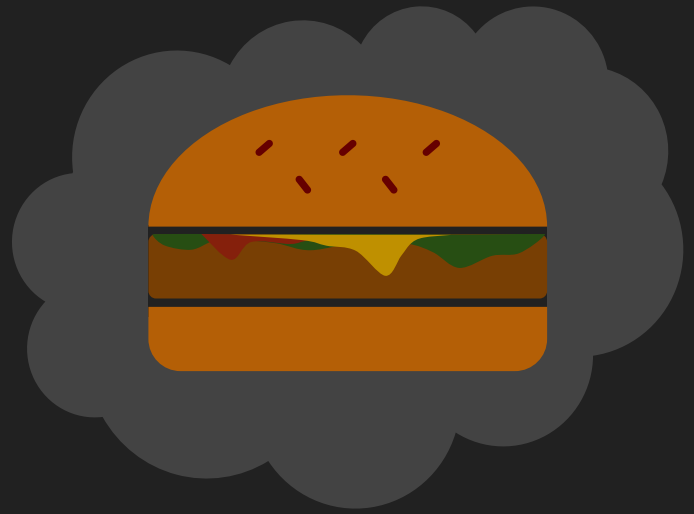
Masking Technology
hello@masking.tech
linkedin.com/company/maskingtechnology

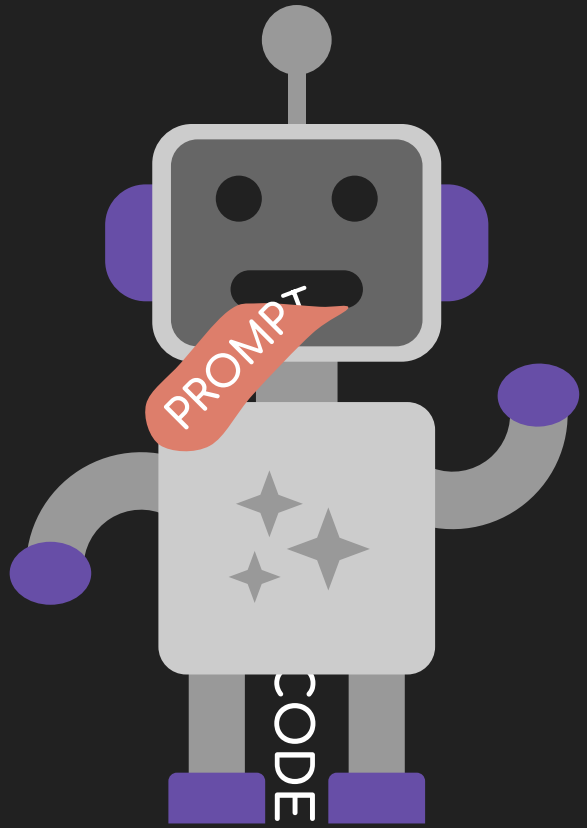
Masking Technology

Object-Oriented
Programming
is great!

But using it
for everything
is not.

Food for Thought



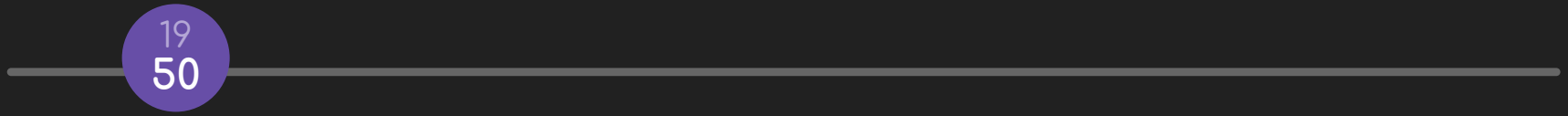


But...
WhAI?

Mainstream Paradigms

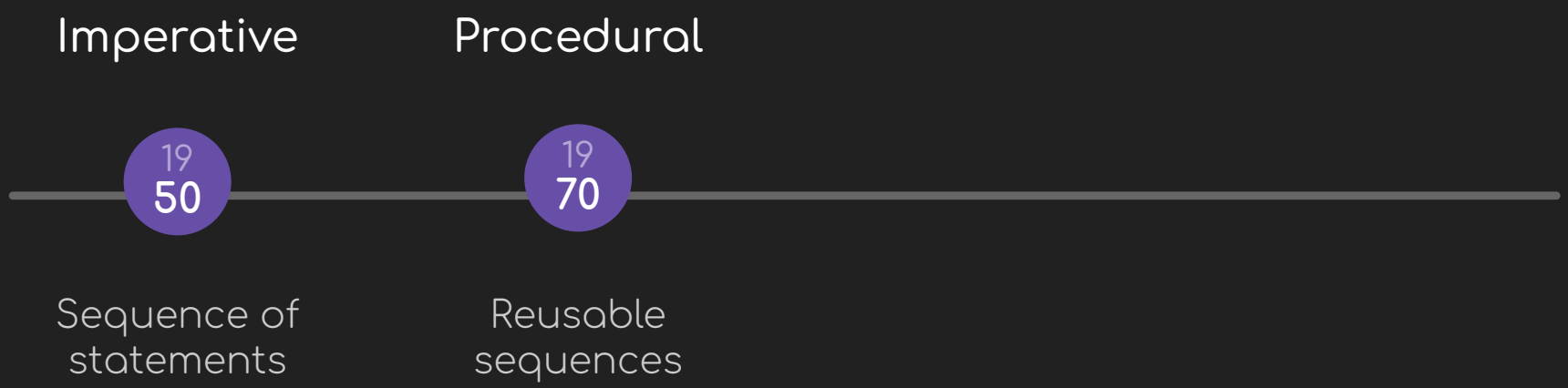
Mainstream Paradigms

Imperative

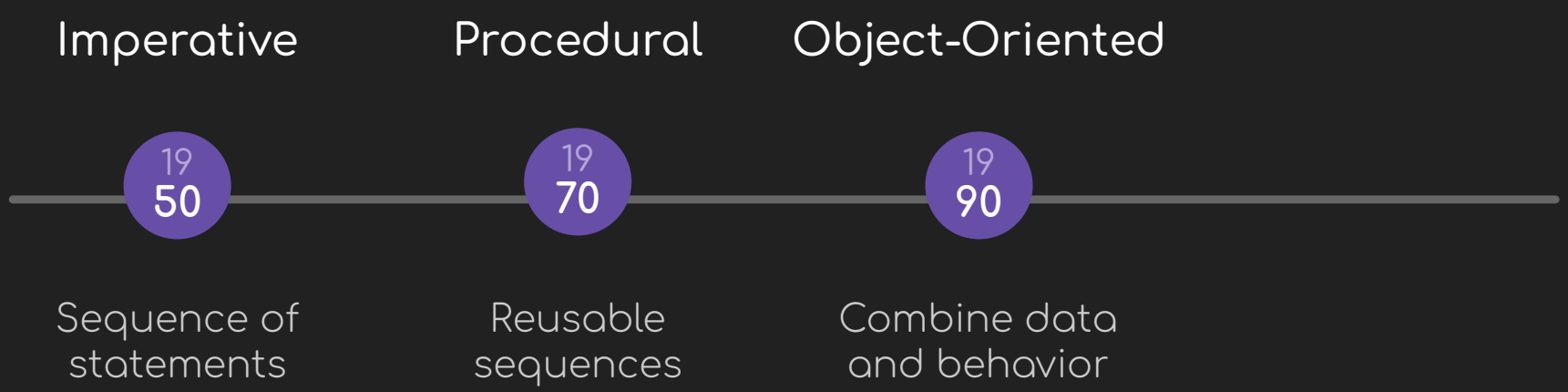


Sequence of statements

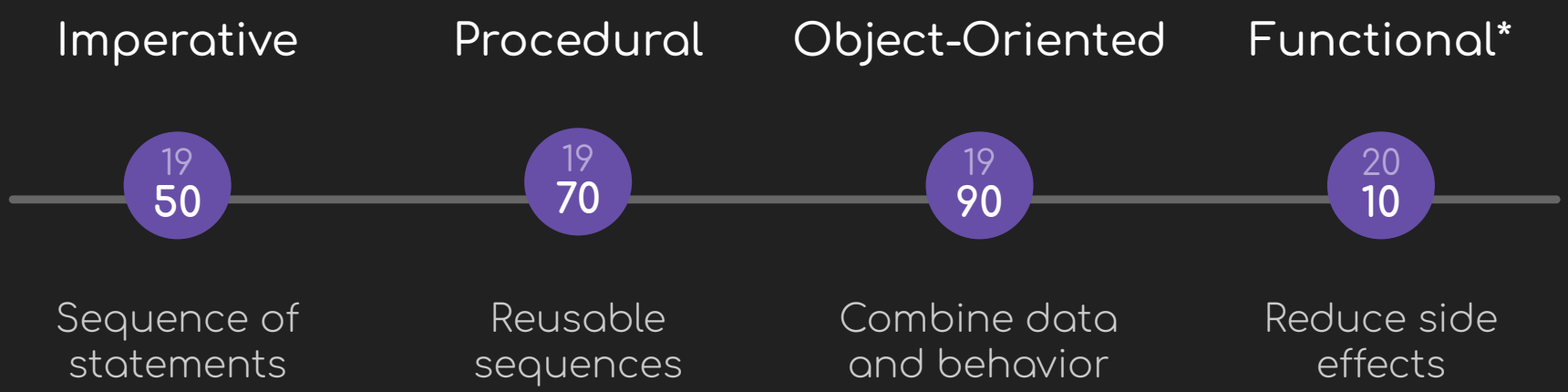
Mainstream Paradigms



Mainstream Paradigms



Mainstream Paradigms



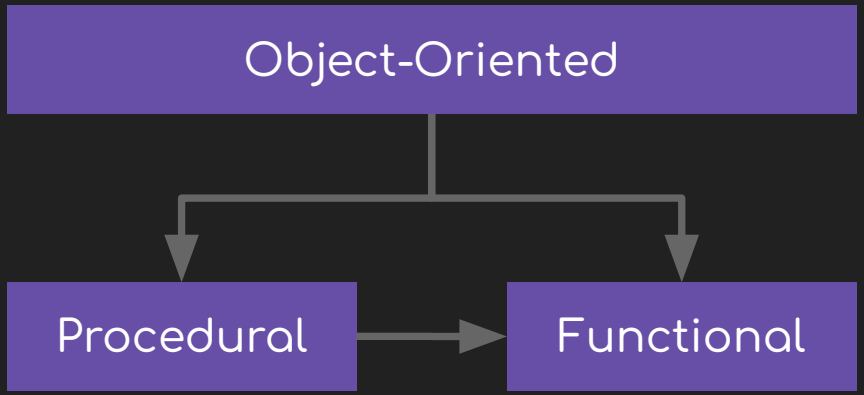
* Inline use in one of the other paradigms.

Modern Multi-Paradigm Synthesis

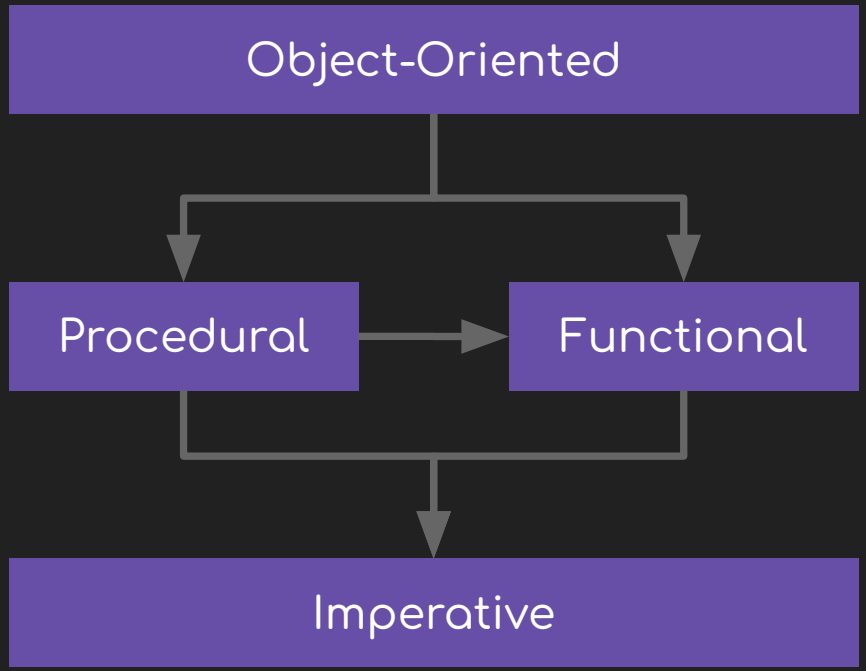
Modern Multi-Paradigm Synthesis

Object-Oriented

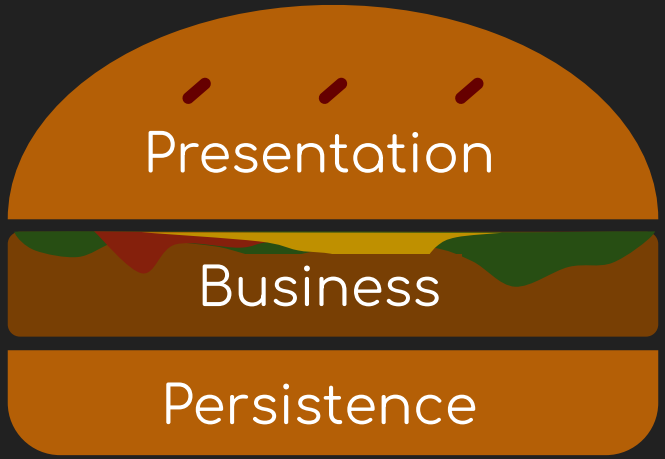
Modern Multi-Paradigm Synthesis



Modern Multi-Paradigm Synthesis



Common web architecture















- Presentation
- Business
- Persistence

- Presentation
- Business
- Persistence

Presentation burger.controller.ts

```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

```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.controller.ts

▼ **Business** burger.service.ts

```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

```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.controller.ts

▶ **Business** burger.service.ts

▼ **Persistence** burger.entity.ts

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

▶ **Presentation** BurgerController.php

▶ **Business** BurgerService.php

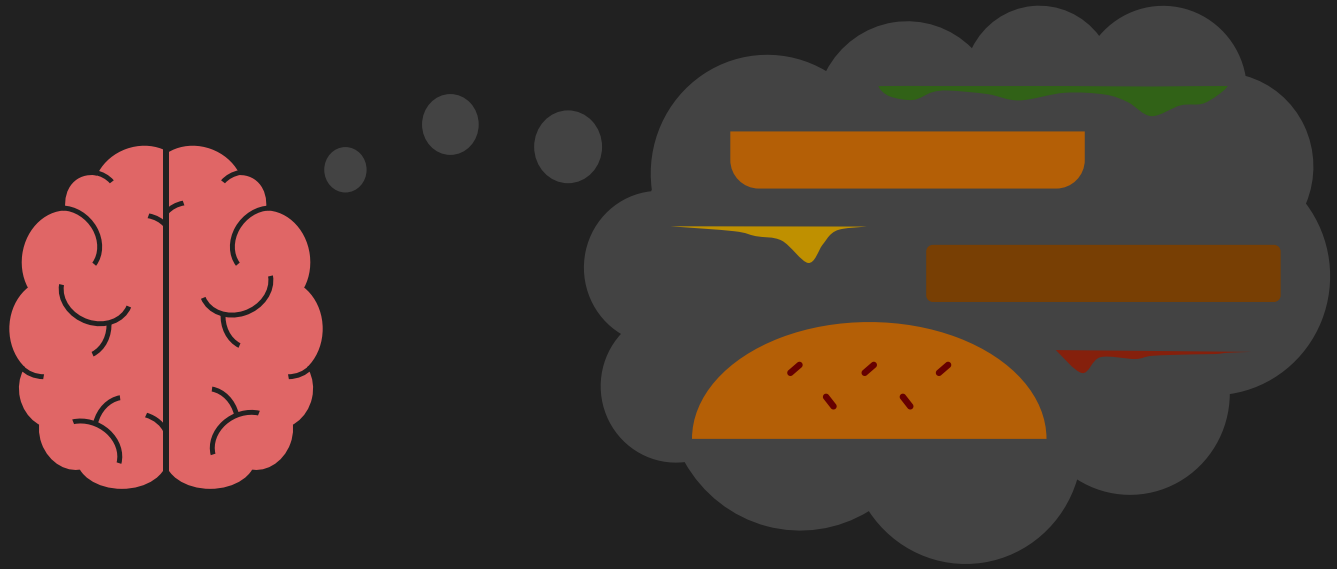
▼ **Persistence** Burger.php

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

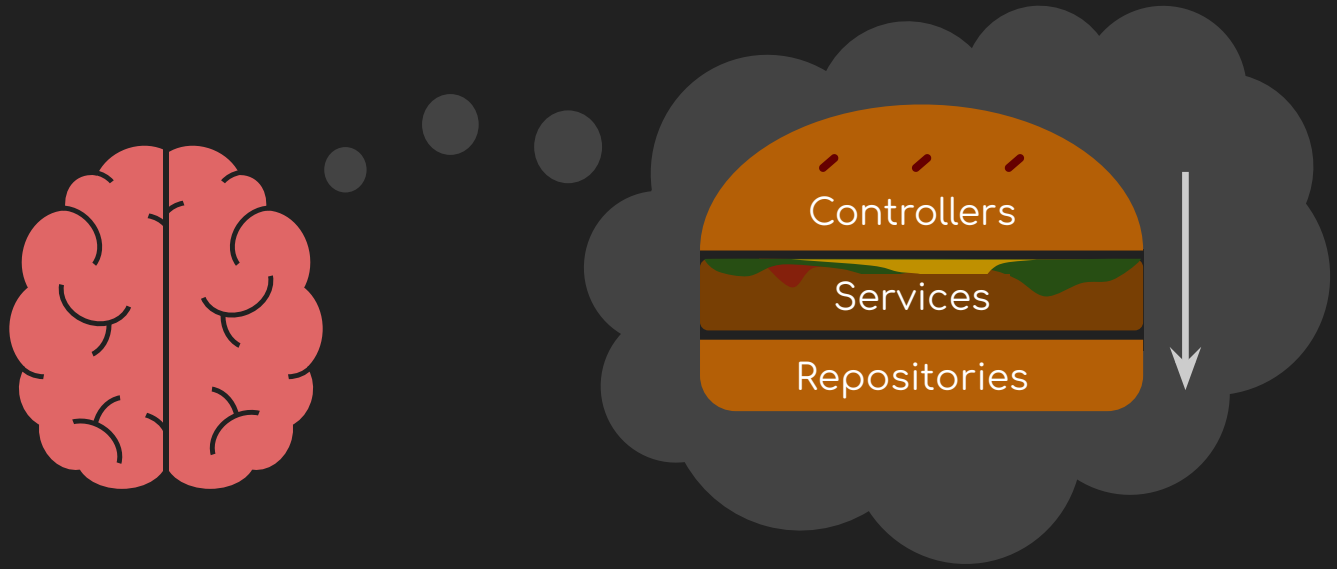
This is an established pattern that works.

But...

Cognitive Load



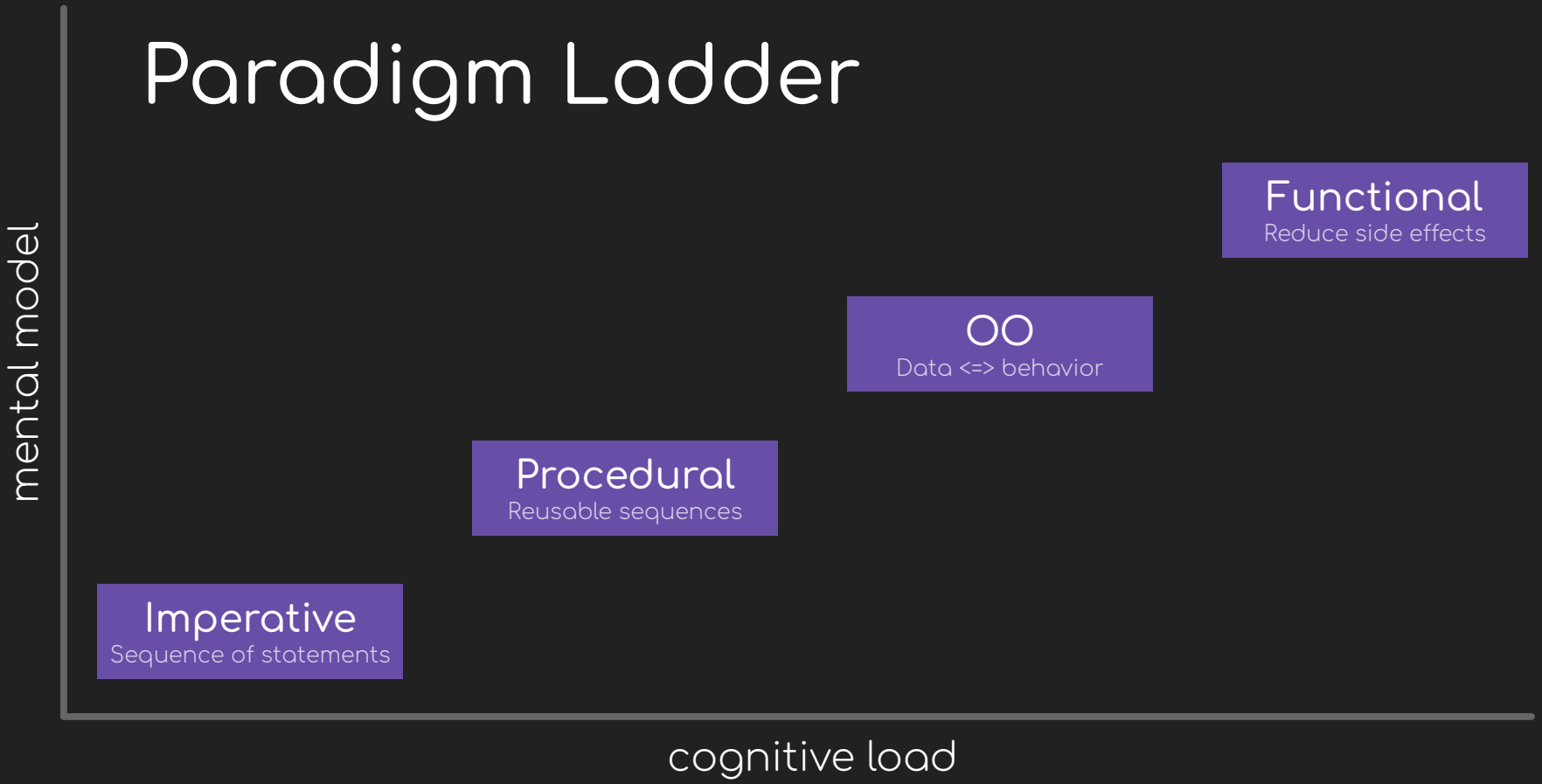
Mental Model



Paradigm Ladder



Paradigm Ladder



Paradigm Ladder



Most complex paradigm

Functional
Reduce side effects

OO
Data <=> behavior

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

Procedural
Reusable sequences

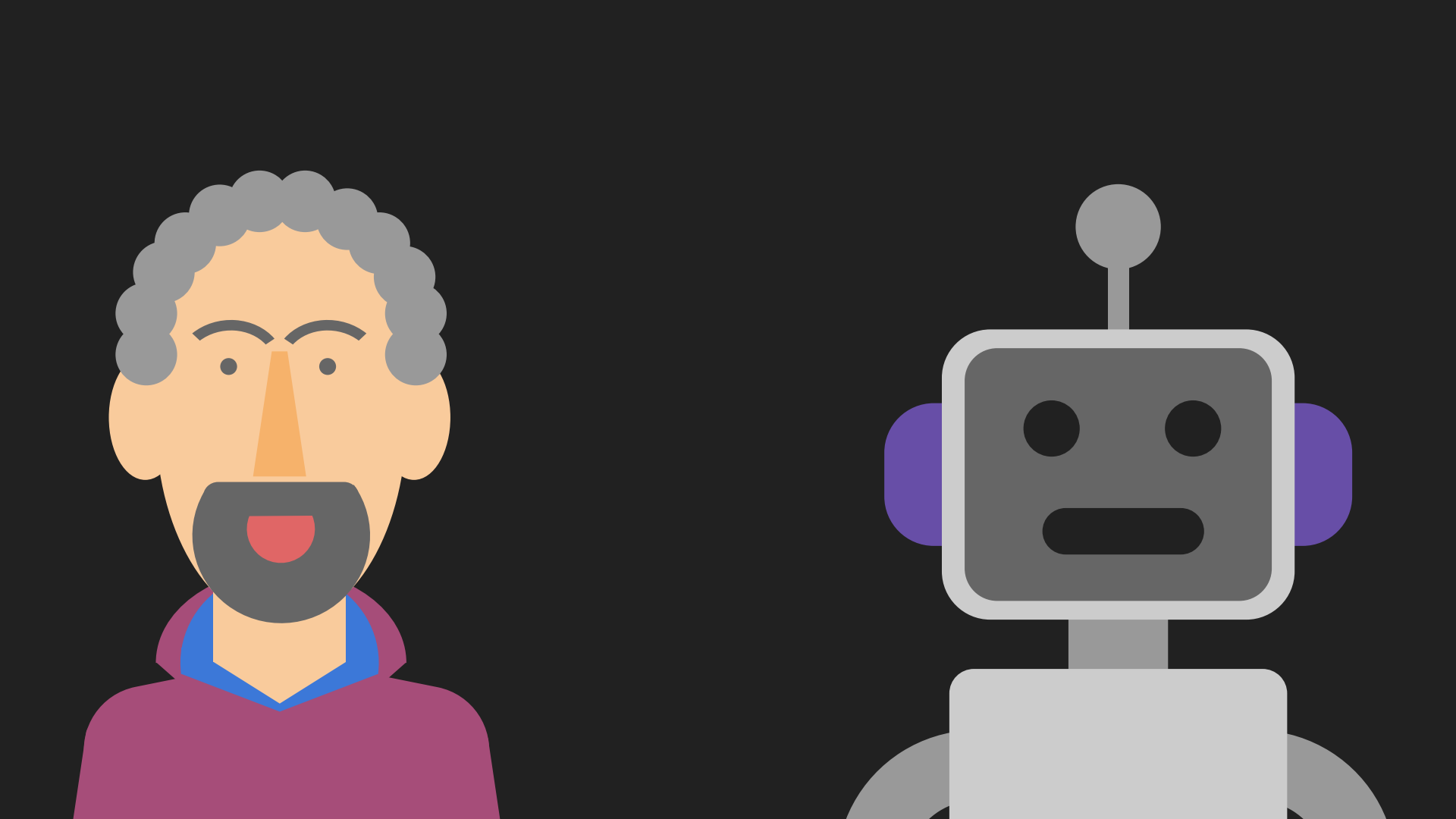
Imperative
Sequence of statements

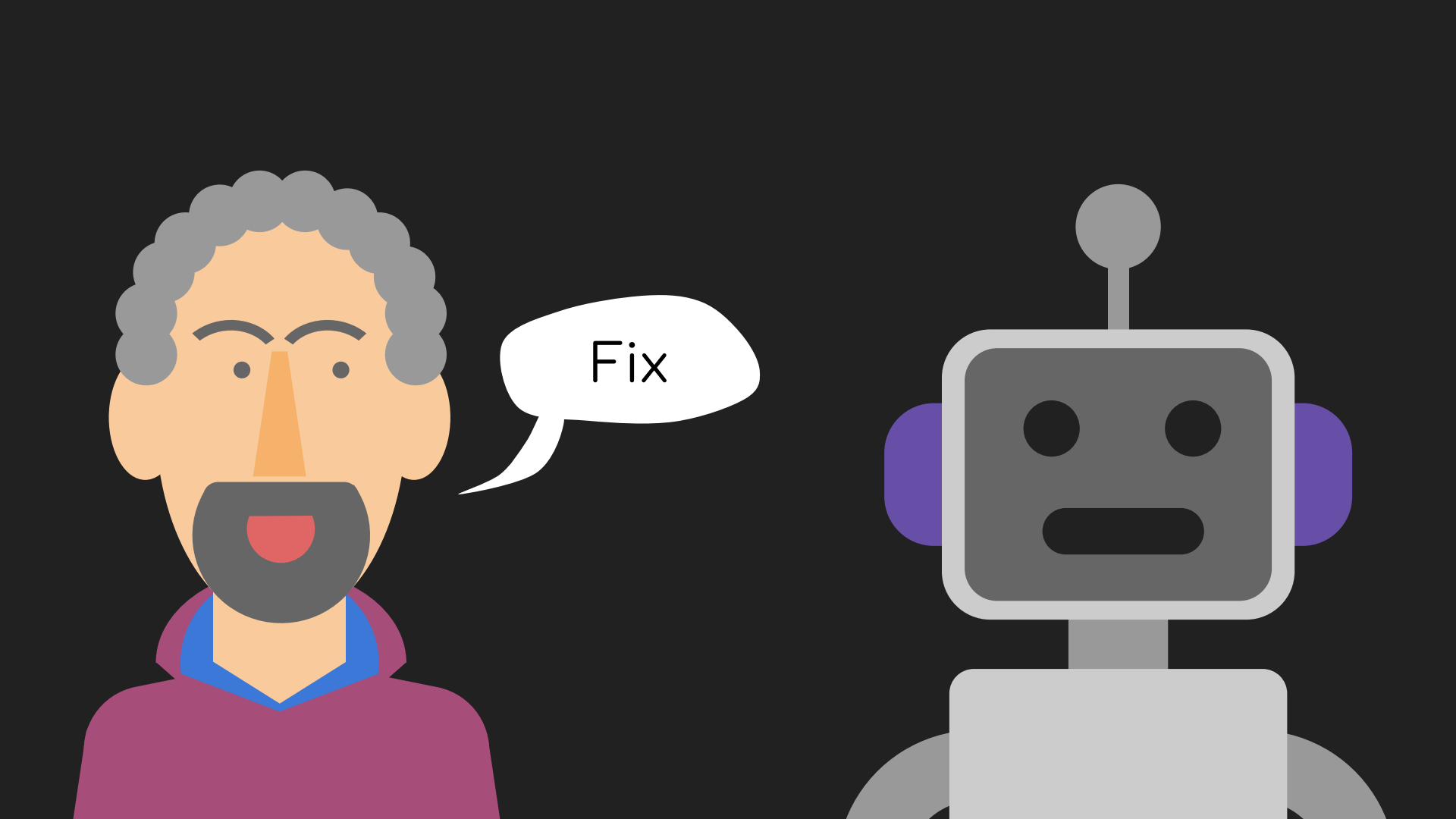
mental model

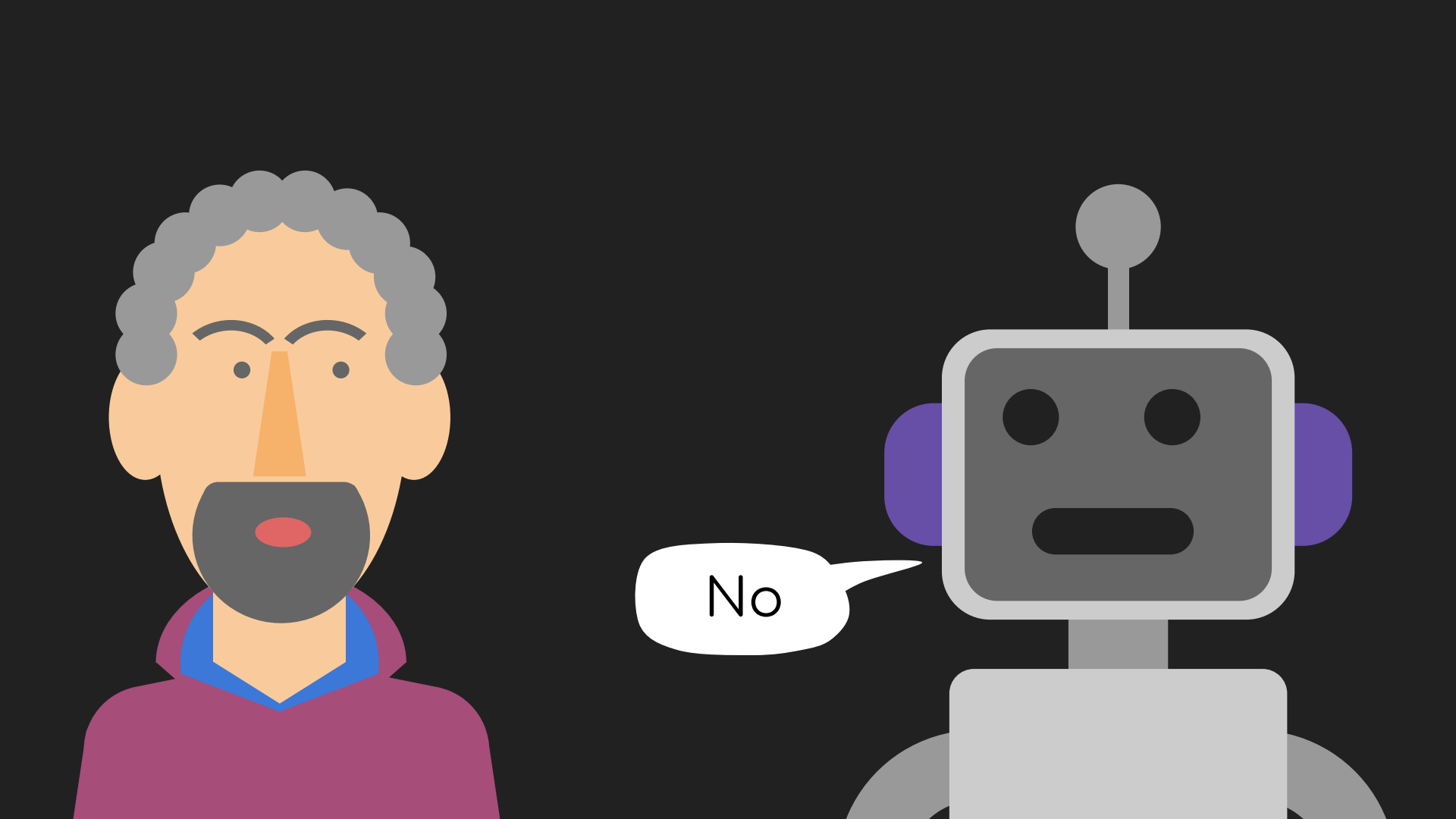
cognitive load

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

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




Fix


No

Semantic Understanding

Complexity-Aware Evaluation of LLM Comprehension (2026, September 29)

Semantic Understanding



Complexity-Aware Evaluation of LLM Comprehension (2026, September 29)

Semantic Understanding



Complexity-Aware Evaluation of LLM Comprehension (2026, September 29)

Semantic Understanding



Complexity-Aware Evaluation of LLM Comprehension (2026, September 29)

Maintainability Tug of War



A Large-Scale Empirical Study of AI-Generated Code in Real-World Repositories (2026, April 3)

Maintainability Tug of War

Humans

Artificial Intelligence

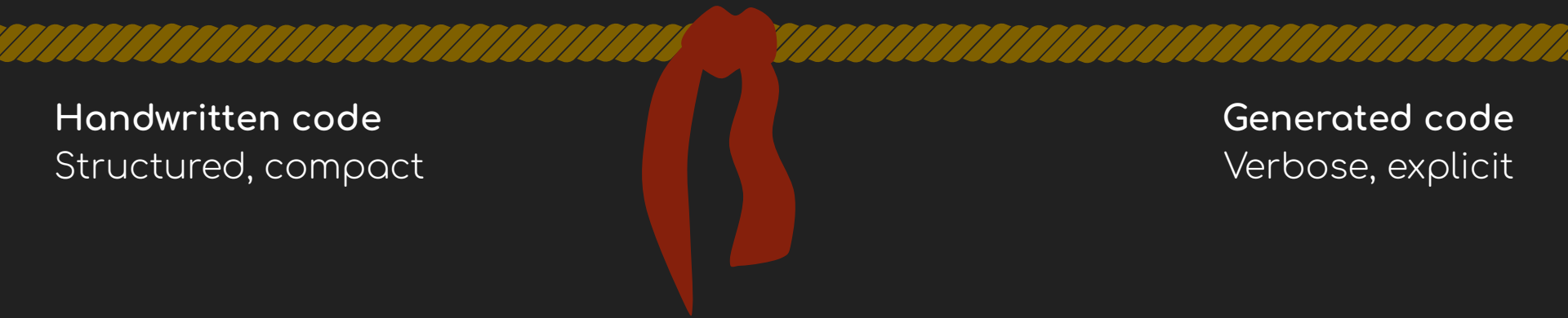


A Large-Scale Empirical Study of AI-Generated Code in Real-World Repositories (2026, April 3)

Maintainability Tug of War

Humans

Artificial Intelligence



Handwritten code
Structured, compact

Generated code
Verbose, explicit

A Large-Scale Empirical Study of AI-Generated Code in Real-World Repositories (2026, April 3)

Maintainability Tug of War

Humans

Handwritten code
Structured, compact

Long-term
maintainability



Artificial Intelligence

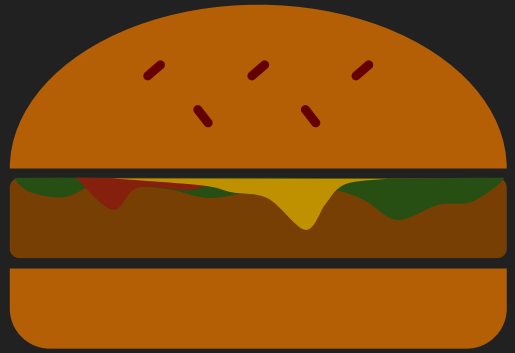
Generated code
Verbose, explicit

Immediate
correctness

A Large-Scale Empirical Study of AI-Generated Code in Real-World Repositories (2026, April 3)

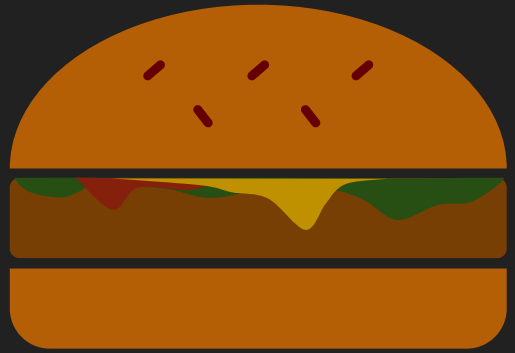
Essential or Accidental Complexity?

App Assessment



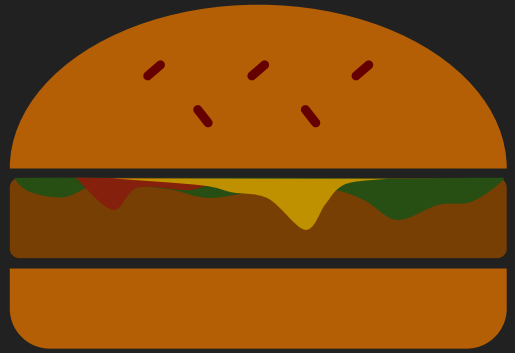
App Assessment

Stateless
Singletons



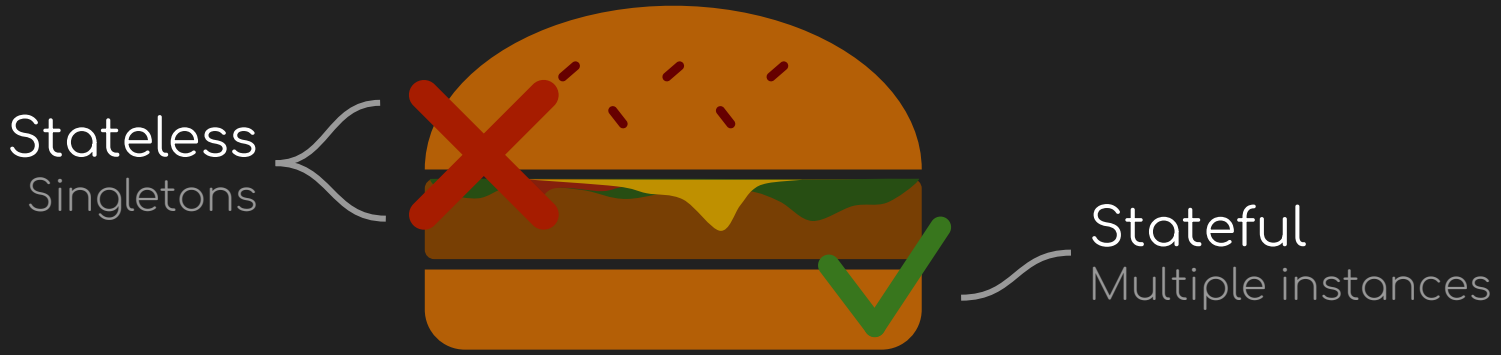
App Assessment

Stateless
Singletons

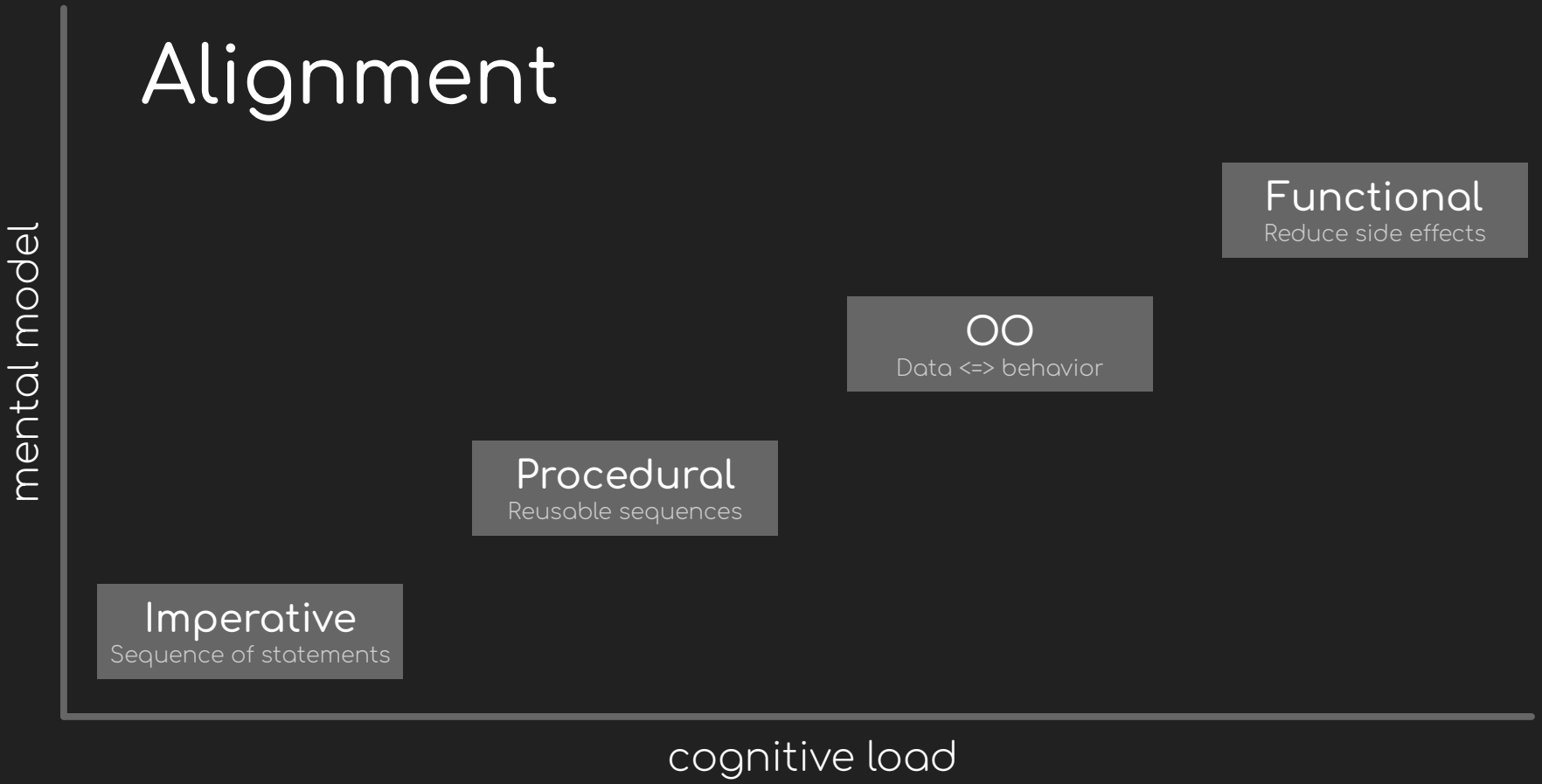


Stateful
Multiple instances

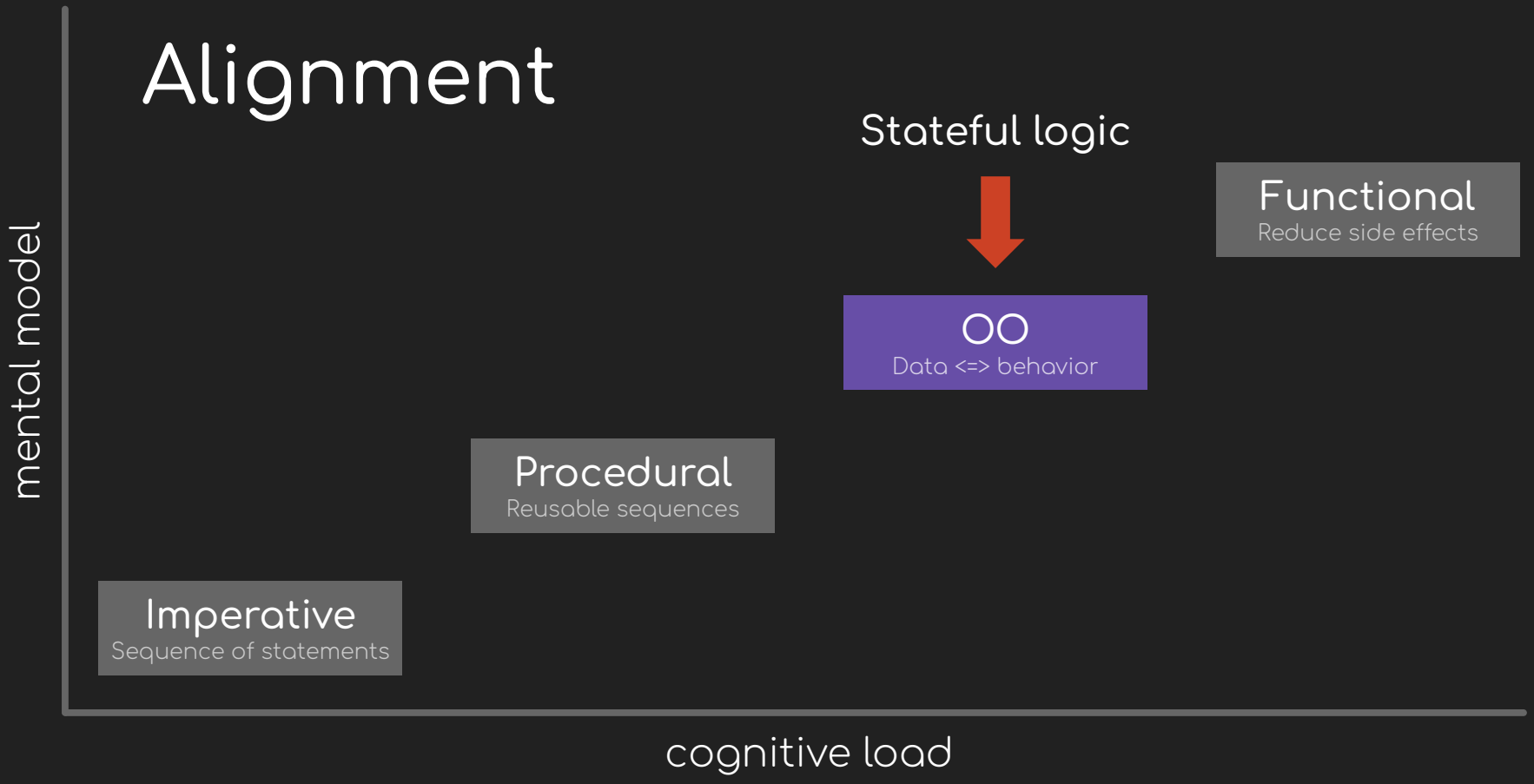
Partial Misalignment

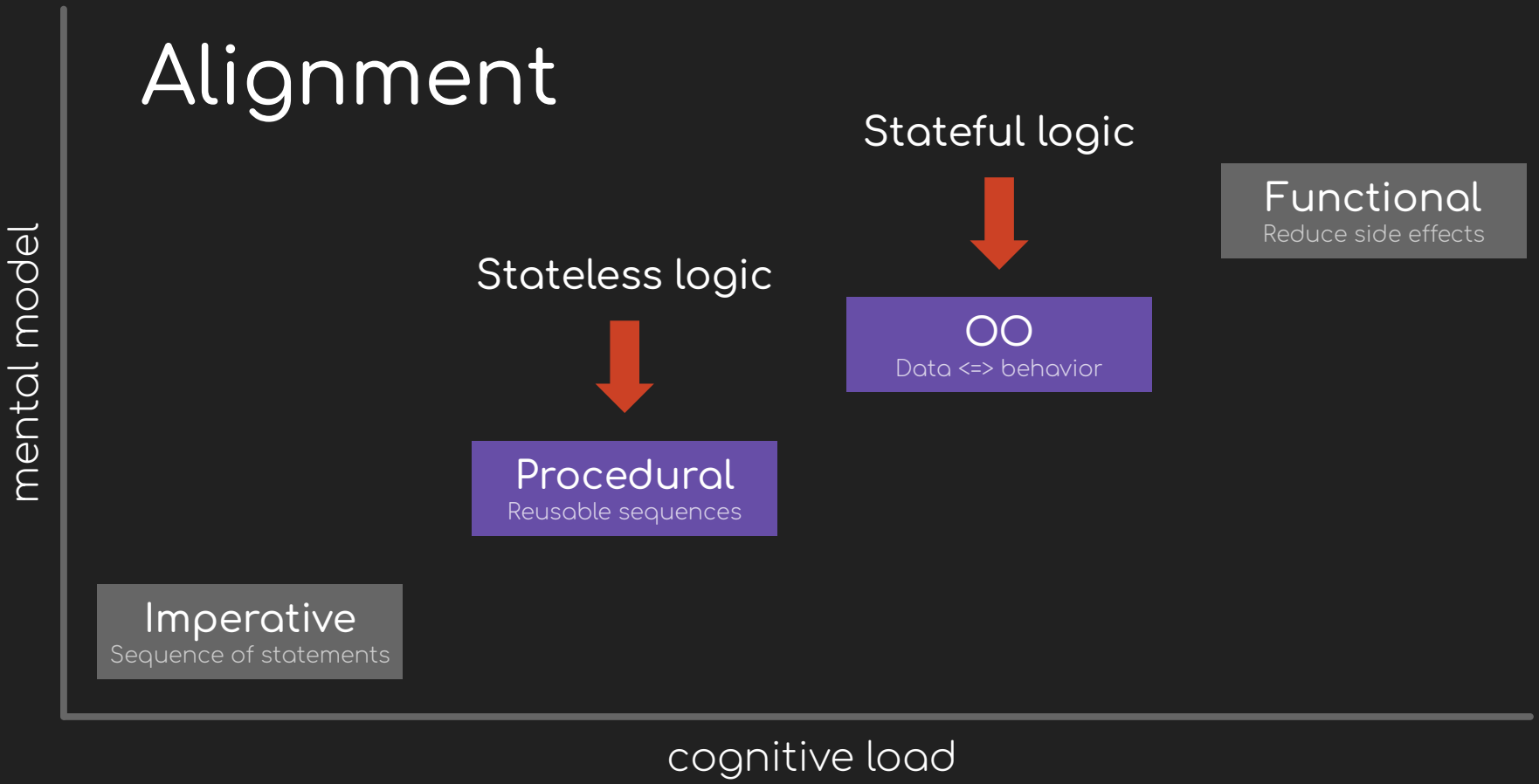


Alignment


mental model
cognitive load
Functional
Reduce side effects
OO
Data <=> behavior
Procedural
Reusable sequences
Imperative
Sequence of statements

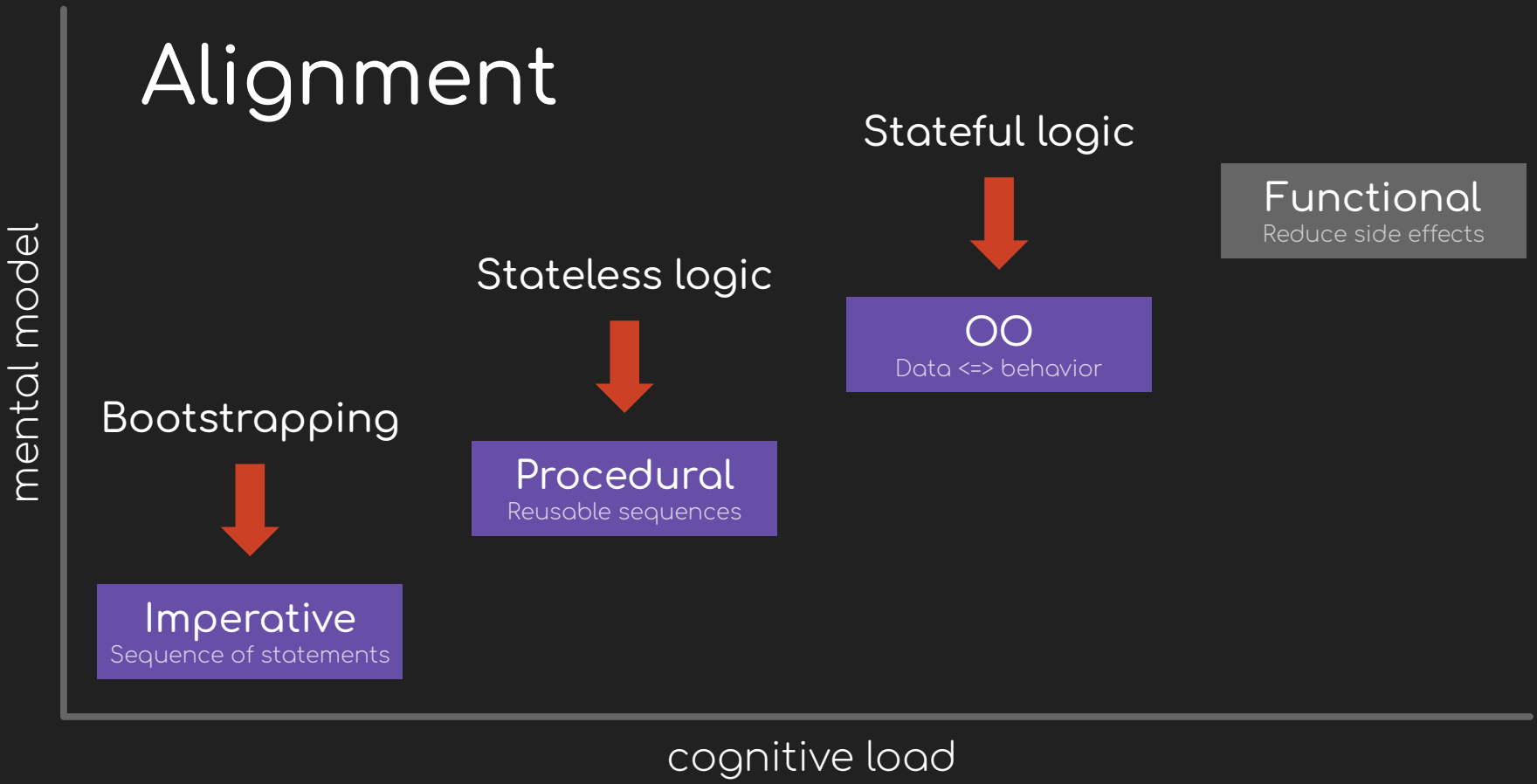
Alignment


mental model
cognitive load
Stateful logic
Functional
Reduce side effects
OO
Data <=> behavior
Procedural
Reusable sequences
Imperative
Sequence of statements


Alignment
Stateful logic
Functional
Reduce side effects
Stateless logic
OO
Data <=> behavior
Procedural
Reusable sequences
Imperative
Sequence of statements
mental model
cognitive load

But what about bootstrapping?




Alignment
Bootstrapping
Imperative
Sequence of statements
Stateless logic
Procedural
Reusable sequences
Stateful logic
OO
Data <=> behavior
Functional
Reduce side effects
mental model
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 <=> 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 <=> 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



Slim Framework

- Presentation
- Business
- Persistence

- 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

```ts
function getBurgerNames(): string[]
{
  const burgers = retrieveBurgers();

  return burgers
    .map(burger => burger.name);
}
```

▶ **Persistence** Burger.ts

▶ **Presentation** route.php

▼ **Business** getBurgerNames.php

```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

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

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

▶ **Presentation** route.php

▶ **Business** getBurgerNames.php

▼ **Persistence** retrieveBurgers.php

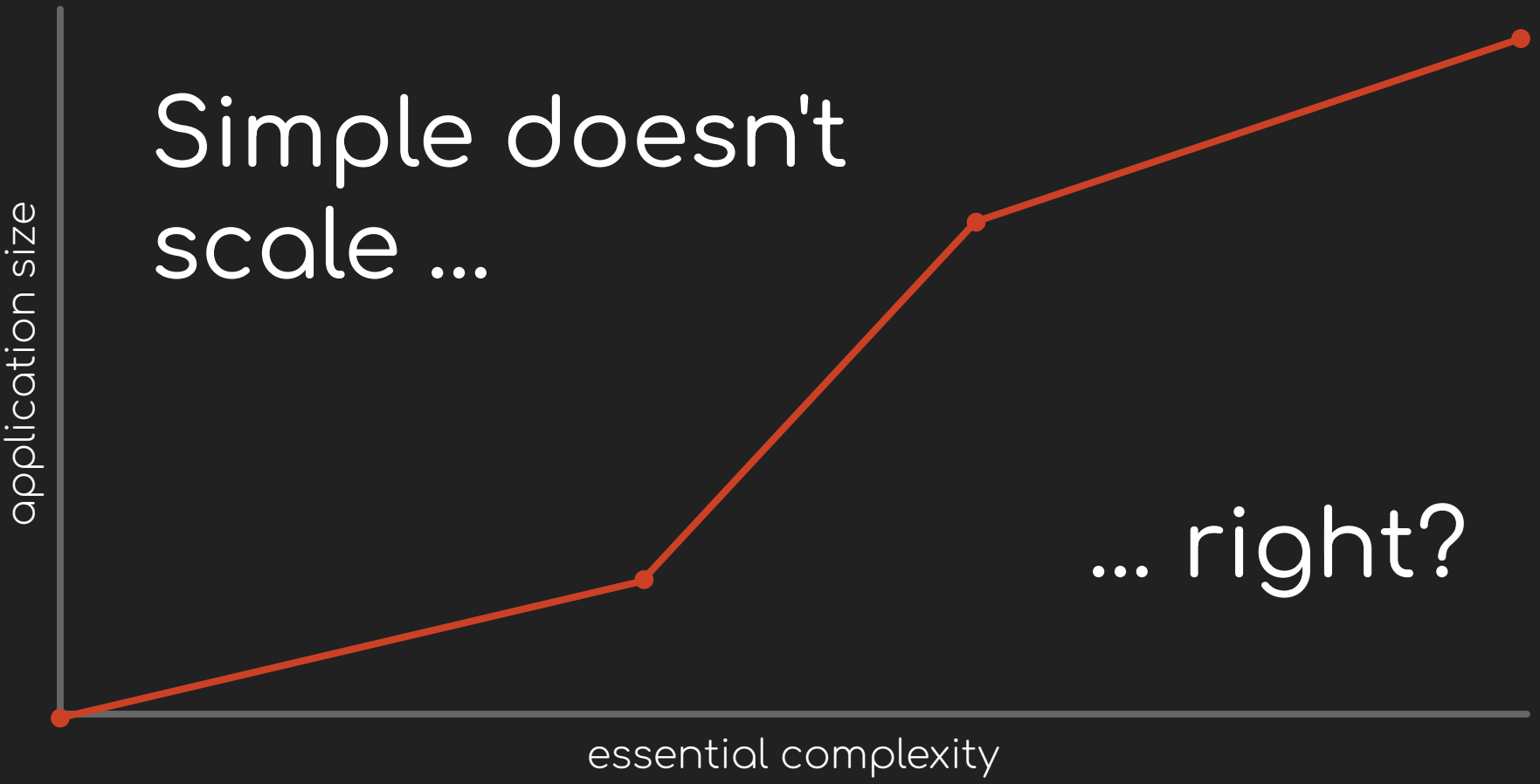
```php
function retrieveBurgers(): array
{
  // Query the database
  // Map the results
}
```

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

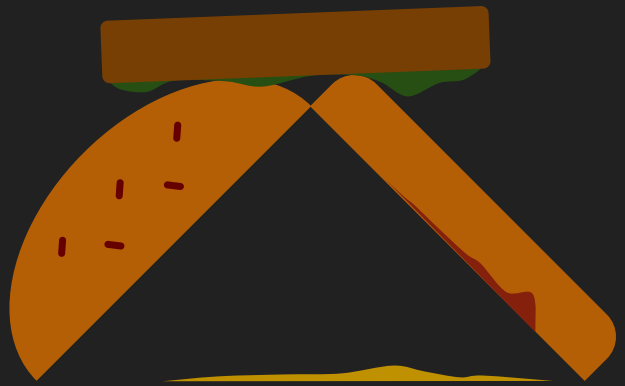
Simplicity!



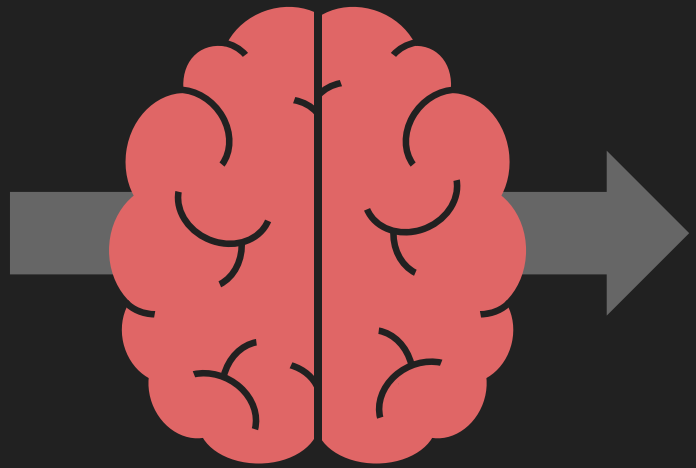
But...


Simple doesn't scale ...
... right?
application size
essential complexity

It's all about proper Application Structure.

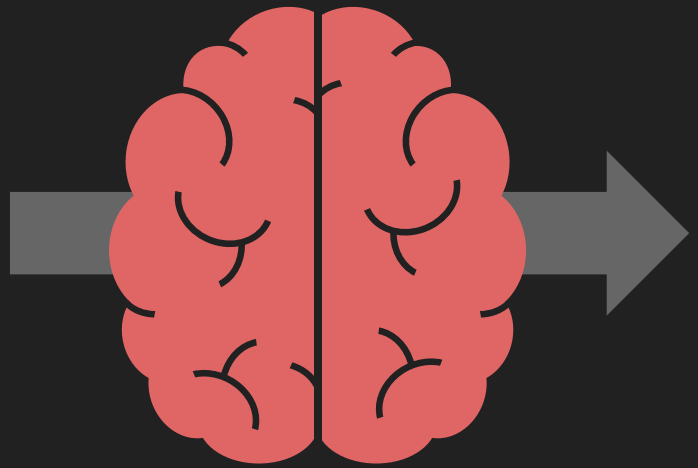


Mindset Shift



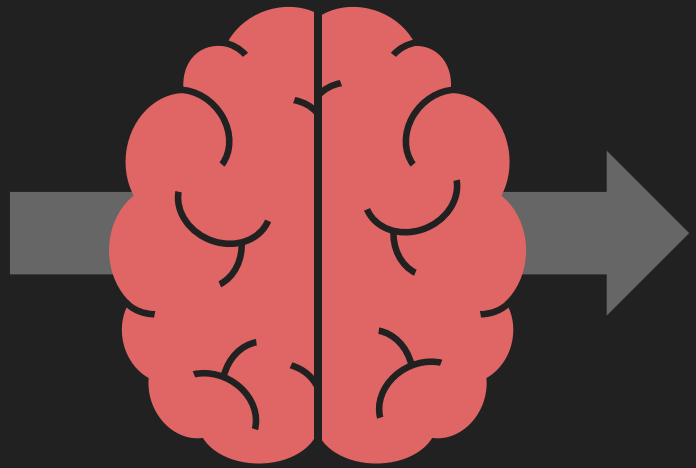
Mindset Shift

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

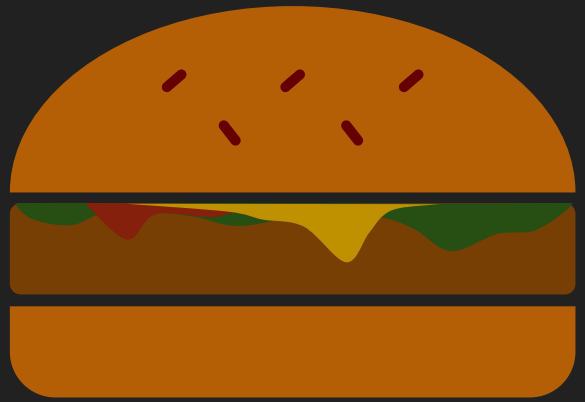
Taxonomy

- app root
  - controllers layer
    - BurgerController
  - services layer
    - BurgerService
  - repositories layer
    - BurgerRepository
  - models layer
    - Burger

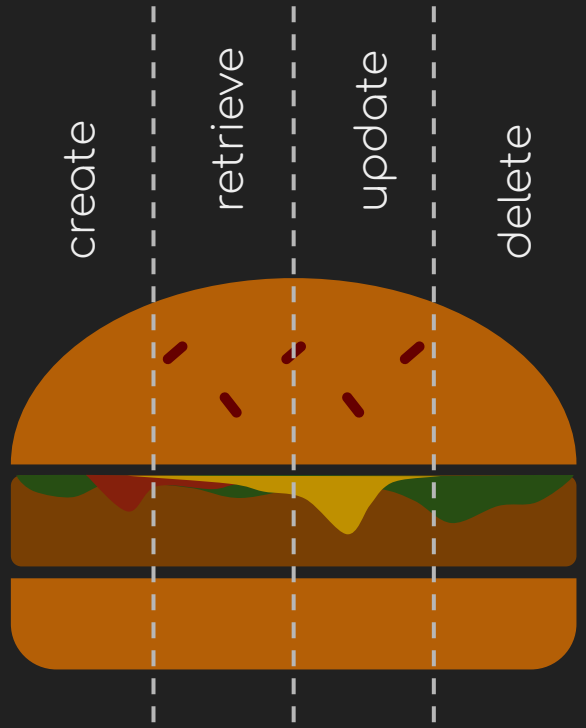
Topography

- app root
  - burger business capability
    - getNames intent
      - Controller
      - Service
      - Repository
      - Model

Vertical Slice Architecture



Vertical Slice Architecture



Object-Oriented

- app root
  - burger feature
    - getNames use-case
      - Controller object-oriented
      - Service object-oriented
      - Repository object-oriented
      - Model object-oriented

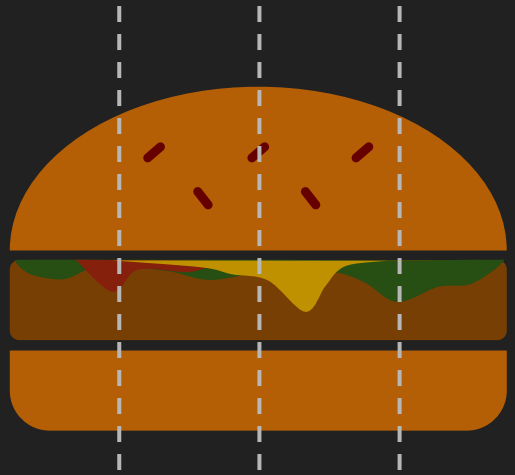
Object-Oriented

- app root
  - burger feature
    - getNames use-case
      - Controller object-oriented
      - Service object-oriented
      - Repository object-oriented
      - Model object-oriented

Blended Approach

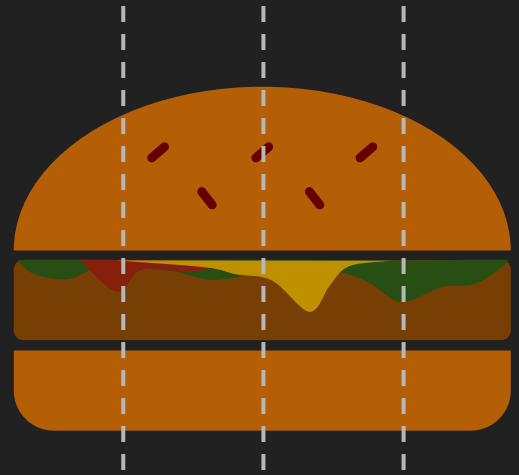
- app root
  - burger feature
    - getNames use-case
      - route imperative
      - getNames procedural
      - retrieveBurgers procedural
      - Burger object(-oriented)

Cognitive Load Decrease

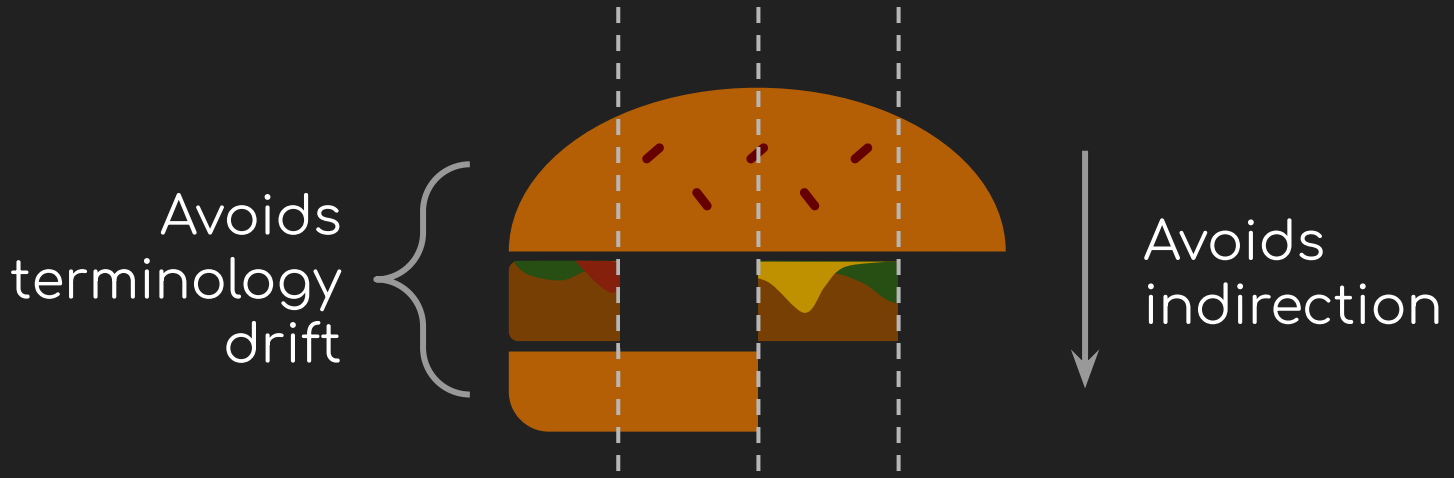


Cognitive Load Decrease

Avoids terminology drift



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

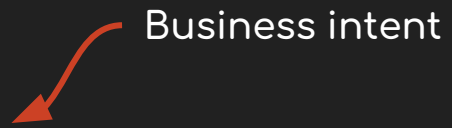
Imperative	**End-to-End** run and verify.
Procedural	**Unit & Integration** per process / flow.
OO	**Unit & Integration** per class / composition.

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.
OO	**Unit & Integration** per class / composition.
Functional	**Mostly implicit** tested in the containing procedure or class.



Use-Case getNames.spec.ts

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

import { getNames } from '...';

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

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

Use-Case GetNamesTest.php

```php
use PHPUnit\Framework\TestCase;

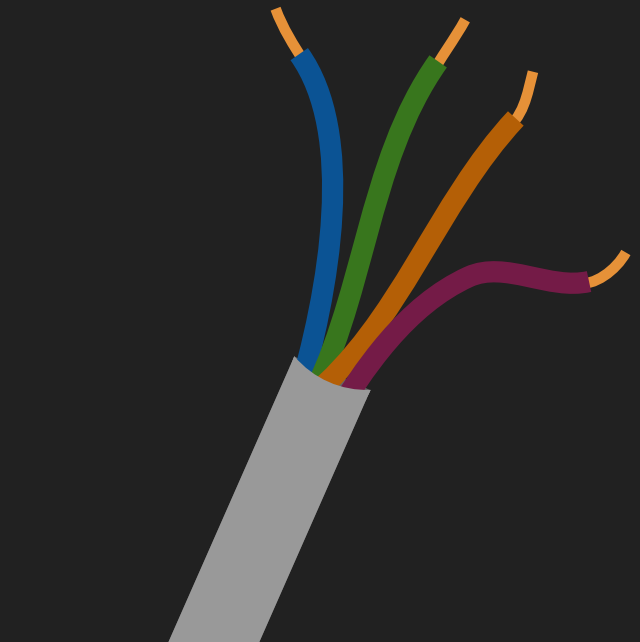
use function App\Burger\getNames;

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

    // reset database
  }

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

Dependency Wiring



Wiring database.ts

```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;
```

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

Wiring database.php

```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
```

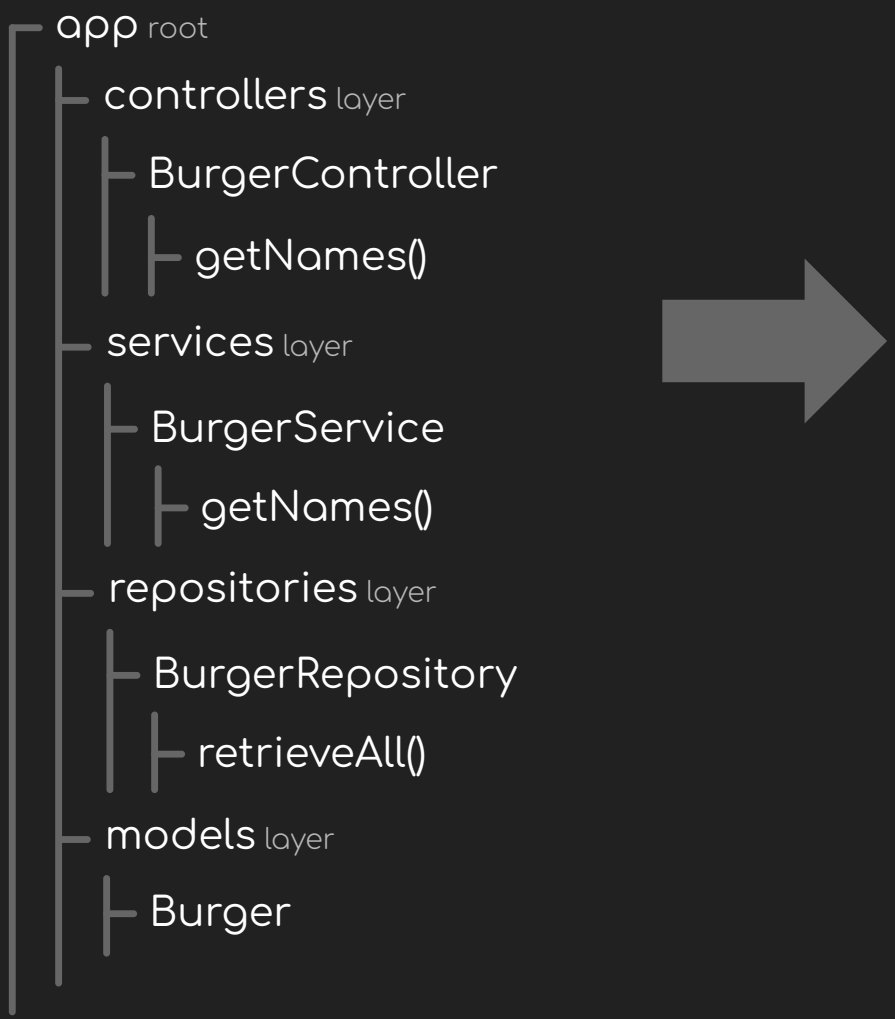
```php
$database = require_once '../db.php';
```

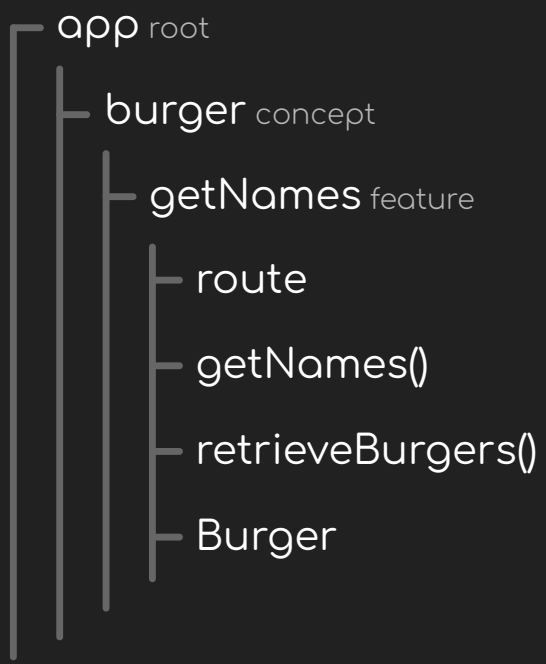
Burger Time!




app root
controllers layer
BurgerController
getNames()
services layer
BurgerService
getNames()
repositories layer
BurgerRepository
retrieveAll()
models layer
Burger


app root
controllers layer
BurgerController
getNames()
services layer
BurgerService
getNames()
repositories layer
BurgerRepository
retrieveAll()
models layer
Burger
app root
burger concept
getNames feature
route
getNames()
retrieveBurgers()
Burger


app root
controllers layer
BurgerController
getNames()
services layer
BurgerService
getNames()
repositories layer
BurgerRepository
retrieveAll()
models layer
Burger


app root
burger concept
getNames feature
route
getNames()
retrieveBurgers()
Burger

75%

less OOP

EXAMPLE

COMIFY

Comic based social media platform.

TS



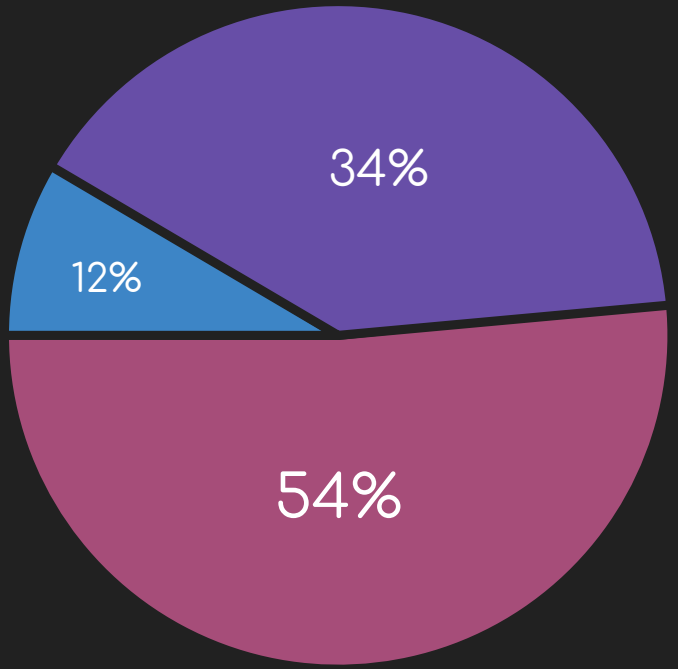
View on GitHub

COMIFY

- Object-Oriented
- Procedural
- Imperative

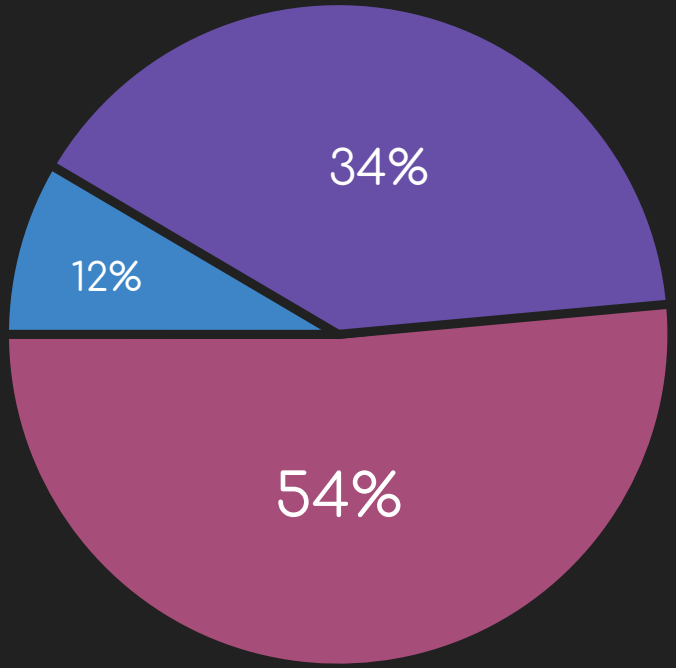


COMIFY



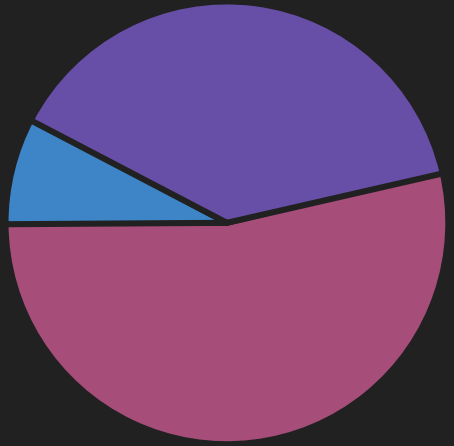
- Object-Oriented
- Procedural
- Imperative

COMIFY



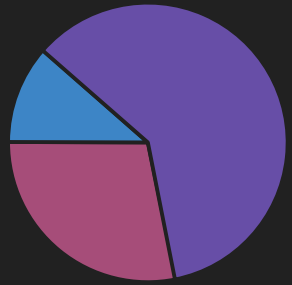
Domain

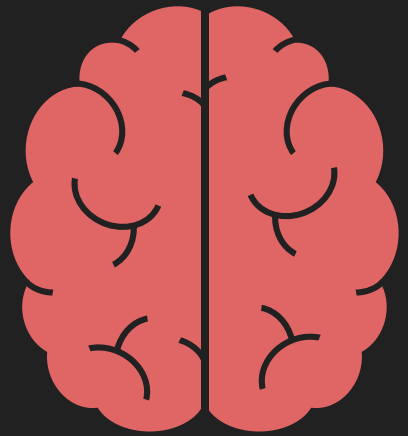
92 use-cases



Integration

11 implementations





Use paradigms
where they fit best.

Thanks!



https://masking.tech