Ayesh Karunaratne | https://aye.sh/talk/dpc2026-defensive-php

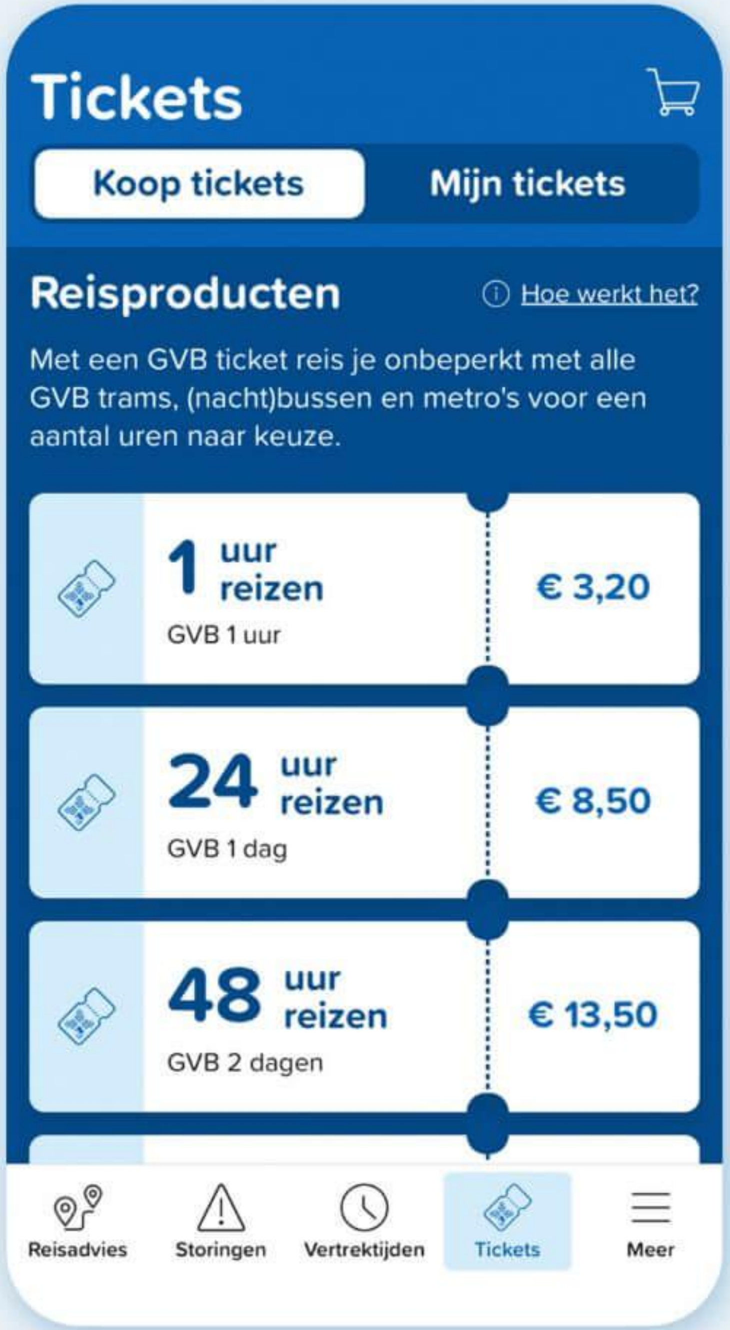
DEFENSIVE PHP

Code that refuses to fail silently

DEFENSIVE PHP

Code that refuses to fail silently

Ayesh Karunaratne | https://aye.sh/talk/dpc2026-defensive-php



Tickets

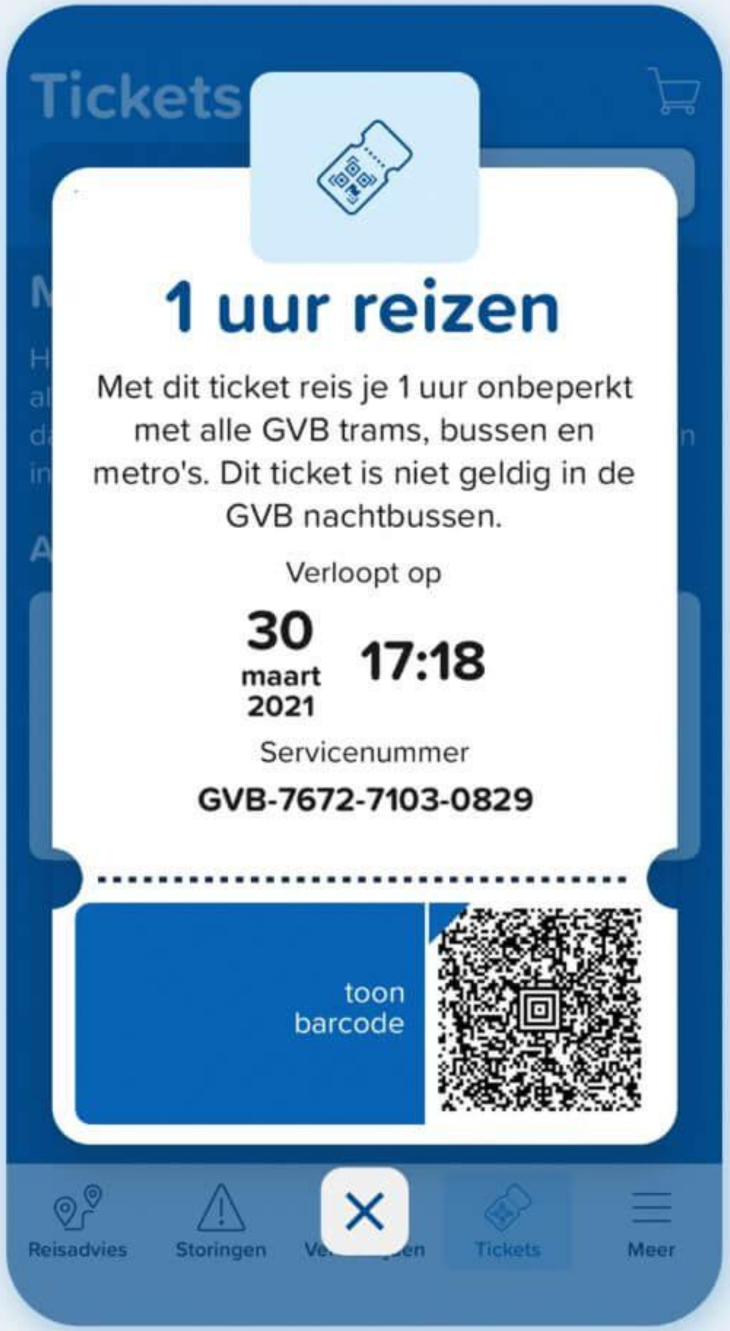
Koop tickets | Mijn tickets

Reisproducten

ⓘ Hoe werkt het?

Met een GVB ticket reis je onbeperkt met alle GVB trams, (nacht)bussen en metro's voor een aantal uren naar keuze.

1 uur reizen GVB 1 uur	€ 3,20
24 uur reizen GVB 1 dag	€ 8,50
48 uur reizen GVB 2 dagen	€ 13,50



1 uur reizen

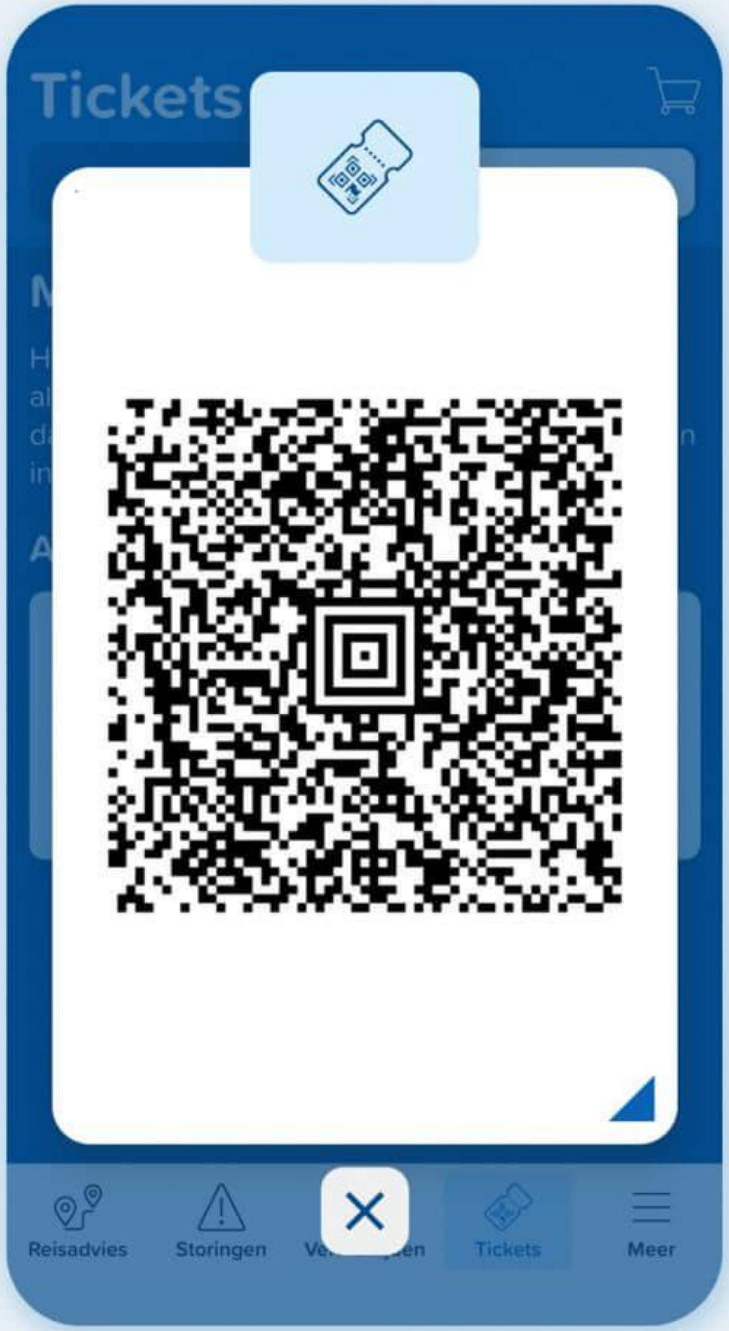
Met dit ticket reis je 1 uur onbeperkt met alle GVB trams, bussen en metro's. Dit ticket is niet geldig in de GVB nachtbussen.

Verloopt op

30 maart 2021 17:18

Servicenummer

GVB-7672-7103-0829



DBS
DBS

apples

apples

oranges

```
"10 apples"  +  "20 oranges"
```

```
"10 apples"  +  "20 oranges"
= 30
```

```
"10 apples"  +  "20 oranges"
= 30
```

```
"10 apples" + "20 oranges"
= 30
```

PHP 4, PHP 5, PHP 7.0	No warnings, no errors
PHP 7.1 – PHP 7.4	Notice A non well formed numeric value encountered
PHP 8.0+	Warning A non-numeric value encountered

```
"apples 10"  +  "oranges 20"
= 0
```

PHP 4, PHP 5, PHP 7.0 No warnings, no errors

PHP 7.1 – PHP 7.4 Notice A non well formed numeric value encountered

```
"apples 10"  +  "oranges 20"
```

PHP 4, PHP 5, PHP 7.0	No warnings, no errors
PHP 7.1 – PHP 7.4	Notice A non well formed numeric value encountered
PHP 8.0+	Fatal Error Unsupported operand types: string + string

```
$a + $b
```

```
(int) $a + (int) $b
```

```php
function add($a, $b) {
    return (int) $a + (int) $b
}
```

```php
function add(int $a, int $b) {
    return $a + $b;
}
```

```php
declare(strict_types=1);

function add(int $a, int $b) {
    return $a + $b;
}
```

```php
• add("10 apples", "20 oranges");
• add("10", "20");
```

TypeError Argument 1 ($a) must be of type int, string given

```php
function add(int|float $a, int|float $b) {
    return $a + $b;
}
```

```
function add(int|float $a, int|float $b): int|float {
    return $a + $b;
}
```

```php
function foo(int|float $a, int|float $b): int|float {
    is_int($a) || is_float($a);
    is_int($b) || is_float($b);



}
```

```
function foo(int|float $a, int|float $b): int|float {
    ~~is_int($a) || is_float($a);~~
    ~~is_int($b) || is_float($b);~~

    return false;

}
```

```
function foo(int|float $a, int|float $b): int|float {
    ~~is_int($a) || is_float($a);~~
    ~~is_int($b) || is_float($b);~~

    return false;

}
```

```php
function foo(int|float $a, int|float $b): int|float {

}
```

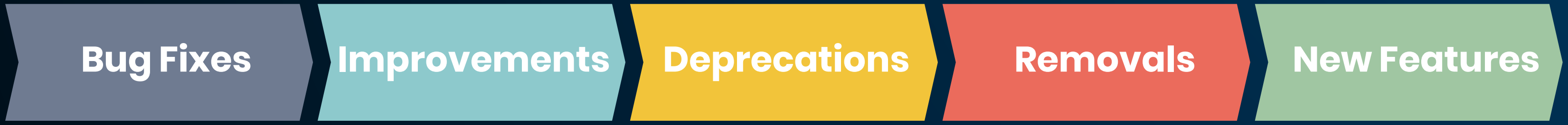
```php
function foo(int …$numbers): int|float

foo(12, 3.14, 45.9);
```

https://php.watch/versions/8.1/deprecate-implicit-conversion-incompatible-float-string

```
function foo(int …$numbers): int|float

foo(12, 3~~.14~~, 45~~.9~~);
```

PHP 8.1+ Deprecated Implicit conversion from float 3.14 to int loses precision

https://php.watch/versions/8.1/deprecate-implicit-conversion-incompatible-float-string



Prioritize Convenience
Polite
Forgiving

Lean & Composable
Accurate
Fail loudly

Namespaces
Autoloading
Traits
Interfaces
Visibility
Final classes

5.x

Scalar types
Return types
Strict types

7.0

Nullable types
`void` return type
Iterable type
Multi catch

7.1

LSP

7.2

Typed Properties
Arrow functions

7.4

Union Types
Named args
Const. properties
`match`
Throw exp.
Mixed type

8.0

`readonly` props
Intersection types
`never` return type
Enums
FCC

8.1

`readonly` classes
True type
DNF
Dynamic props deprecated

8.2

Typed class constants
`#[Override]` Attribute

8.3

Property hooks
Asymmetric Visibility
#[Deprecated] Attribute

8.4

`#[NoDiscard]` Attribute
Clone-with
Fatal error backtraces

8.5

Filter exceptions
More `ValueError`

8.6

Ayesh Karunaratne

PHP Core team, Full-time traveler, Author of PHP.Watch

linkorb

Sri Lanka › Vietnam › Netherlands

https://aye.sh https://php.watch

@Ayesh

ayesh@aye.sh

Defensive Programming

is ...

Defensive Programming is not about writing more checks.

Defensive Programming
is <u>**not**</u> about
writing more checks.

It's about
designing systems where
mistakes are impossible.

Containers

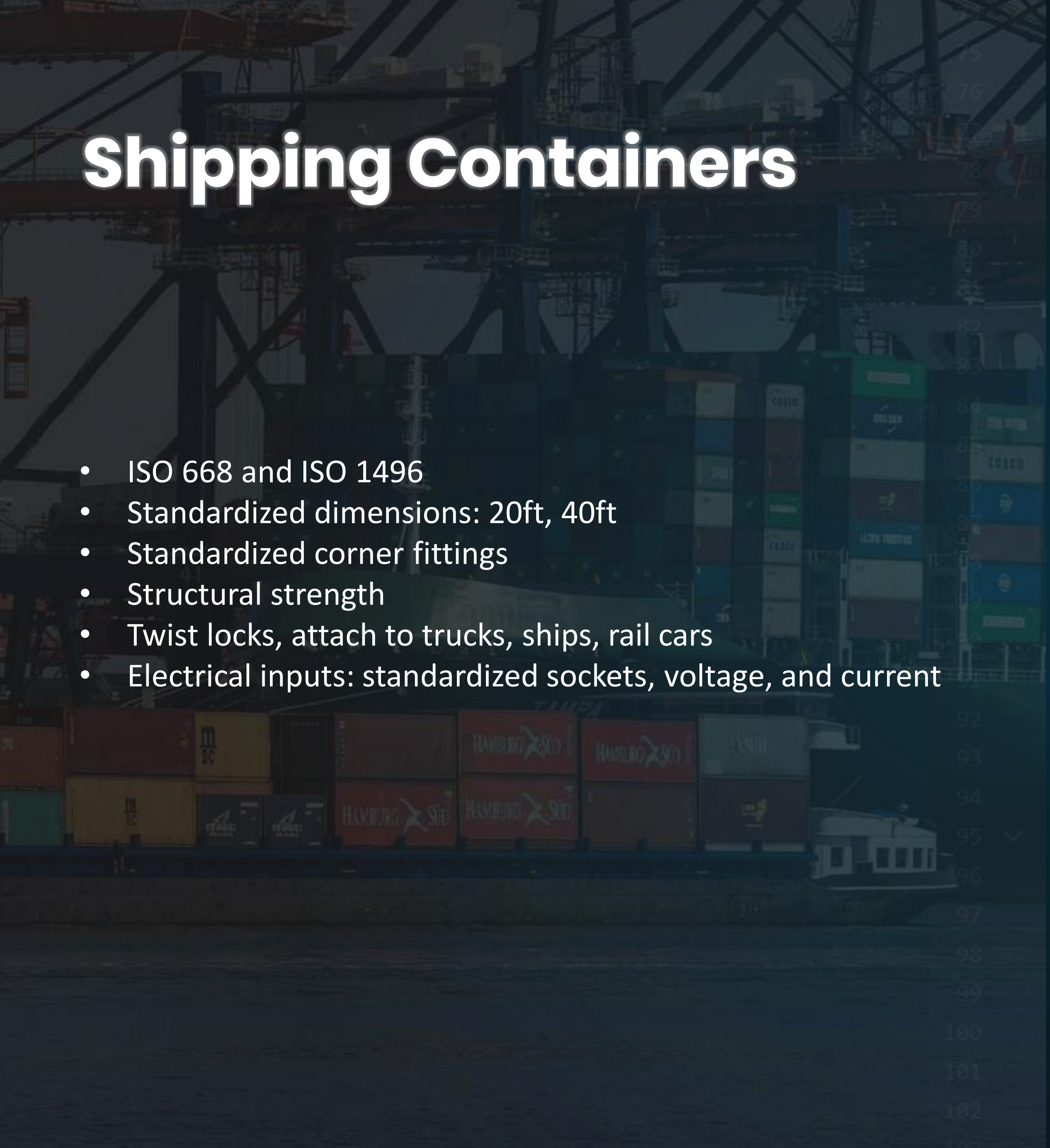
Shipping Containers

Shipping Containers

- ISO 668 and ISO 1496
- Standardized dimensions: 20ft, 40ft
- Standardized corner fittings
- Structural strength
- Twist locks, attach to trucks, ships, rail cars
- Electrical inputs: standardized sockets, voltage, and current

Shipping Containers

- ISO 668 and ISO 1496
- Standardized dimensions: 20ft, 40ft
- Standardized corner fittings
- Structural strength
- Twist locks, attach to trucks, ships, rail cars
- Electrical inputs: standardized sockets, voltage, and current

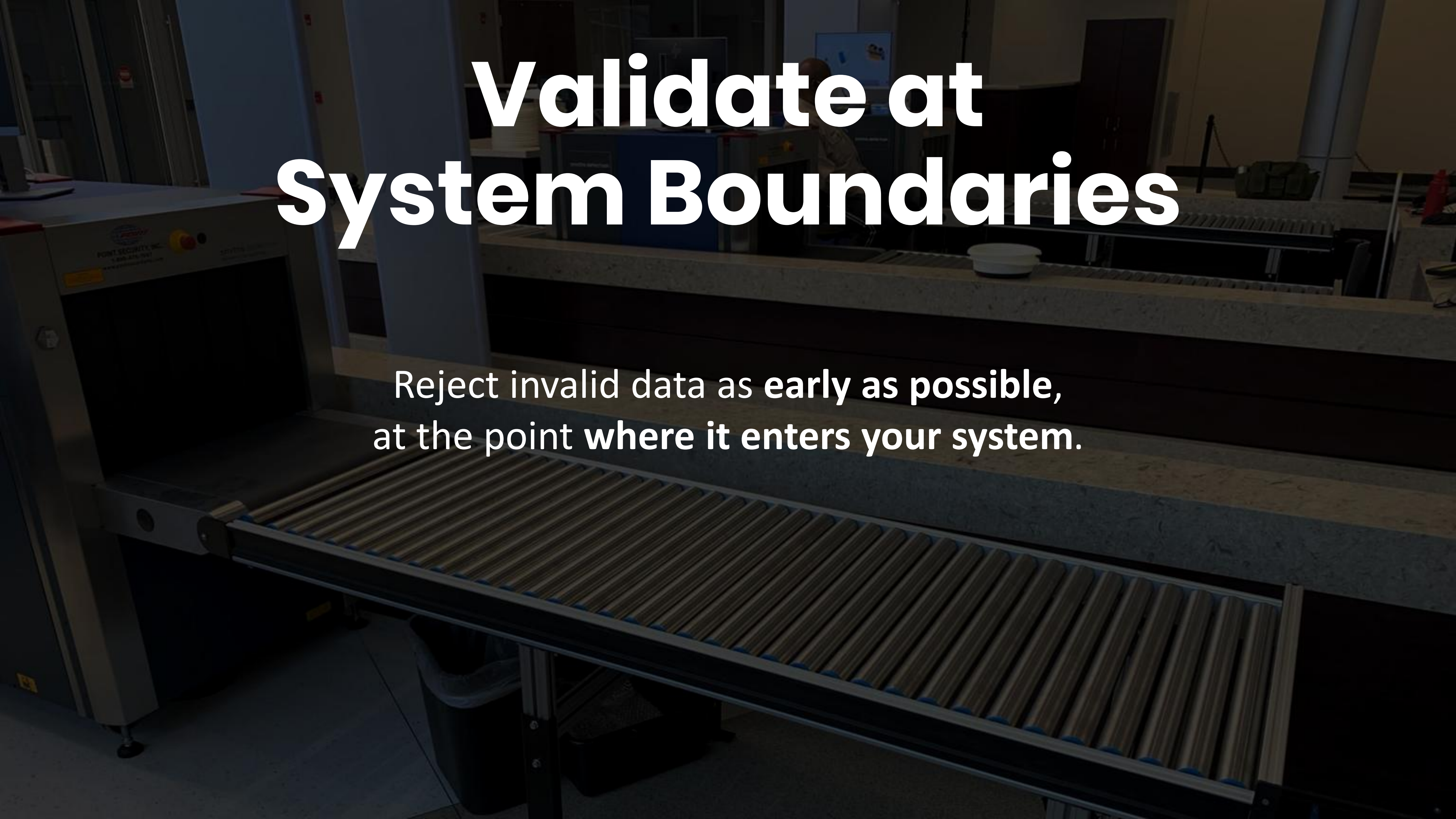


```
class Application implements ResetInterface
{
    private array $commands = [];
    private bool $wantHelps = false;
    private ?Command $runningCommand = null;
    private ?CommandLoaderInterface $commandLoader = null;
    private bool $catchExceptions = true;
    private bool $catchErrors = false;
    private bool $autoExit = true;
    private InputDefinition $definition;
    private HelperSet $helperSet;
    private ?EventDispatcherInter    e $dispatcher = null;
    private ?Arg      lverInt        gument          null;
    private Term       l;
    private str    $defau
    private boo    single        n    false
    private bo            zed      alse;
    private ?    alRegistry $signalRegistry =    l;
    private       $signalsToDispatchEvent =
    private ?int $alarmInterval = null;

    public function __construct(
        private string $name = 'UNKNOWN',
        private string $version = 'UNKNOWN',
    ) {
        $this->terminal = new Terminal();
        $this->defaultCommand = 'list';
        if (\defined('SIGINT') && SignalRegistry::isSupported()) {
            $this->signalRegistry = new SignalRegistry();
```

php

- Validate at system boundaries
- Use type system aggressively
- Invalid state impossible by design
- Immutability and Integrity
- Make behavior explicit
- Interface-defined behavior
- Small, focused, and predictable components

Validate at System Boundaries



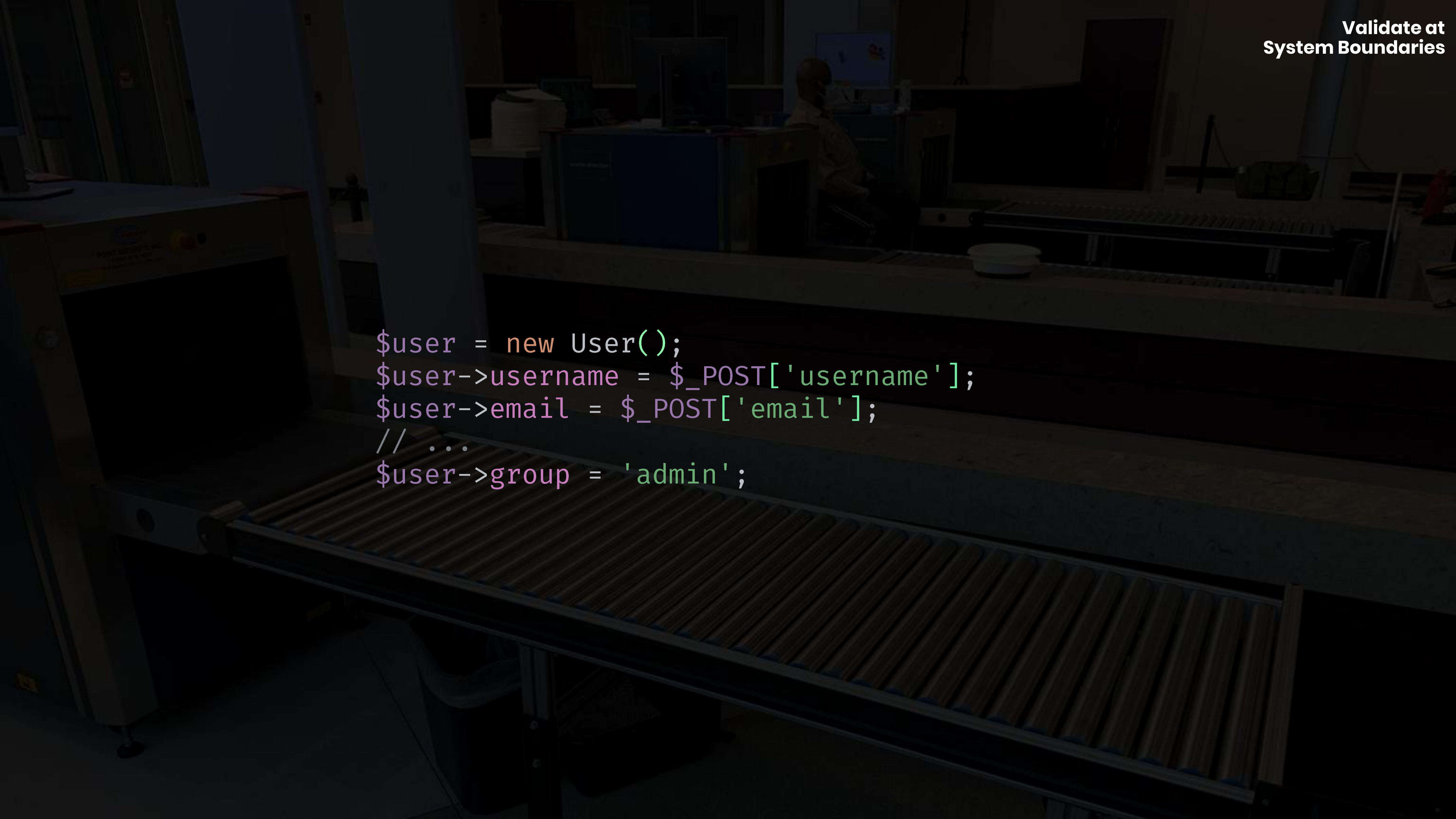
Validate at System Boundaries

Reject invalid data as **early as possible**,
at the point **where it enters your system**.

Validate at System Boundaries

Reject invalid data as **early as possible**,
at the point **where it enters your system**.

The goal is to prevent **bad input from propagating** through your domain model.



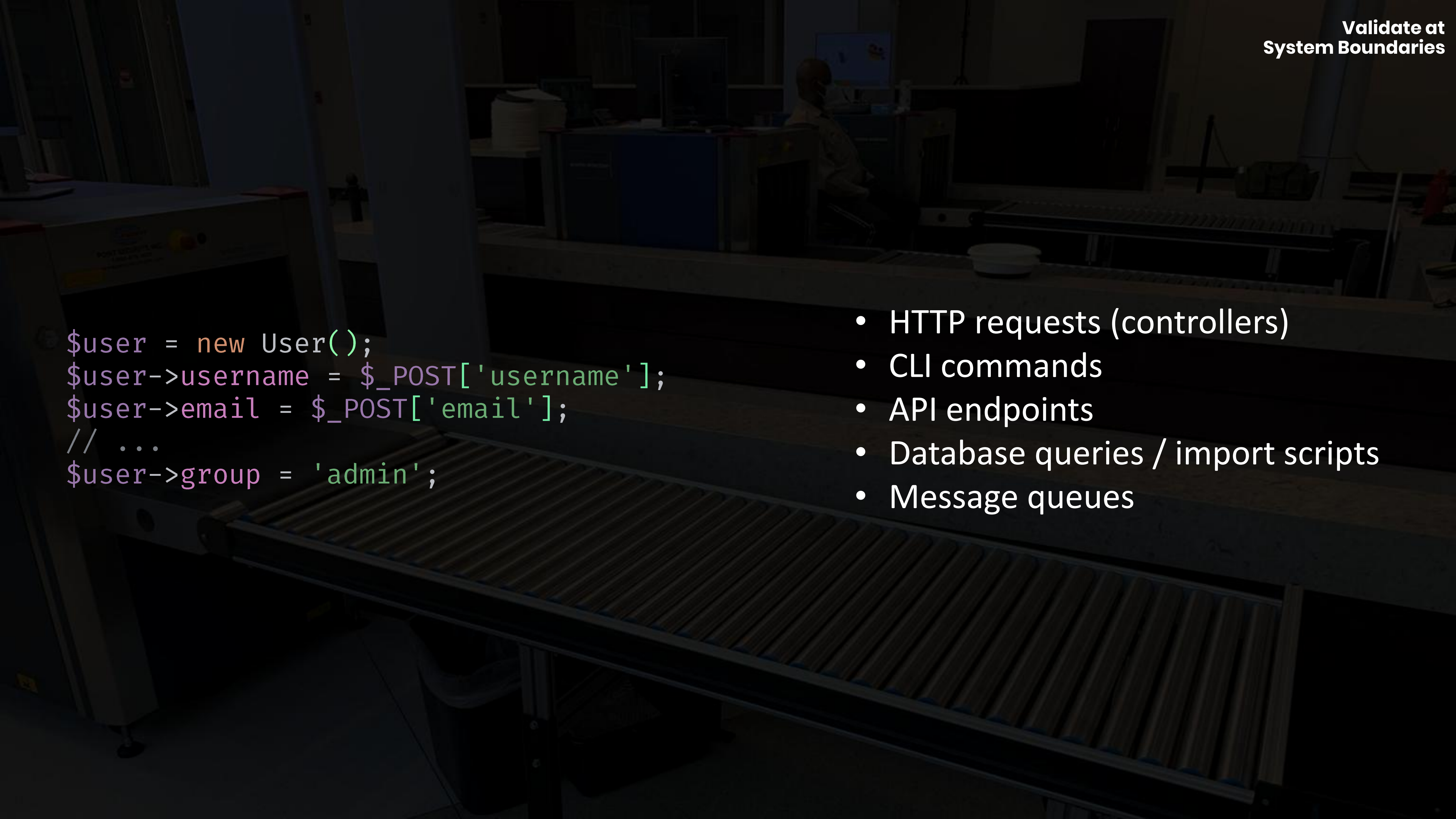
Validate at System Boundaries

```php
$user = new User();
$user->username = $_POST['username'];
$user->email = $_POST['email'];
// ...
$user->group = 'admin';
```

Validate at System Boundaries

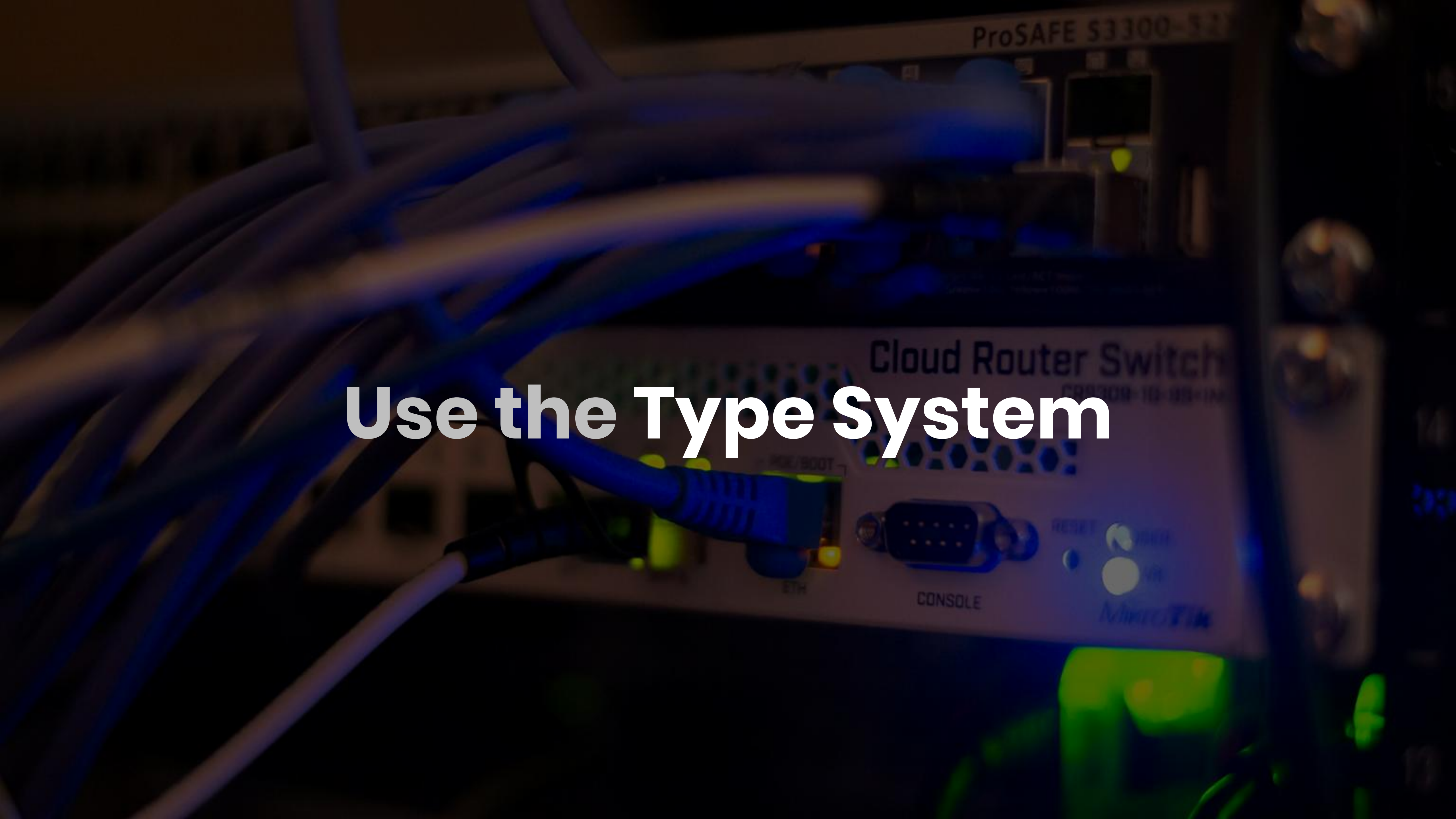
```
$user = new User();
$user->username = $_POST['username'];
$user->email = $_POST['email'];
// ...
$user->group = 'admin';
```

Validate at System Boundaries

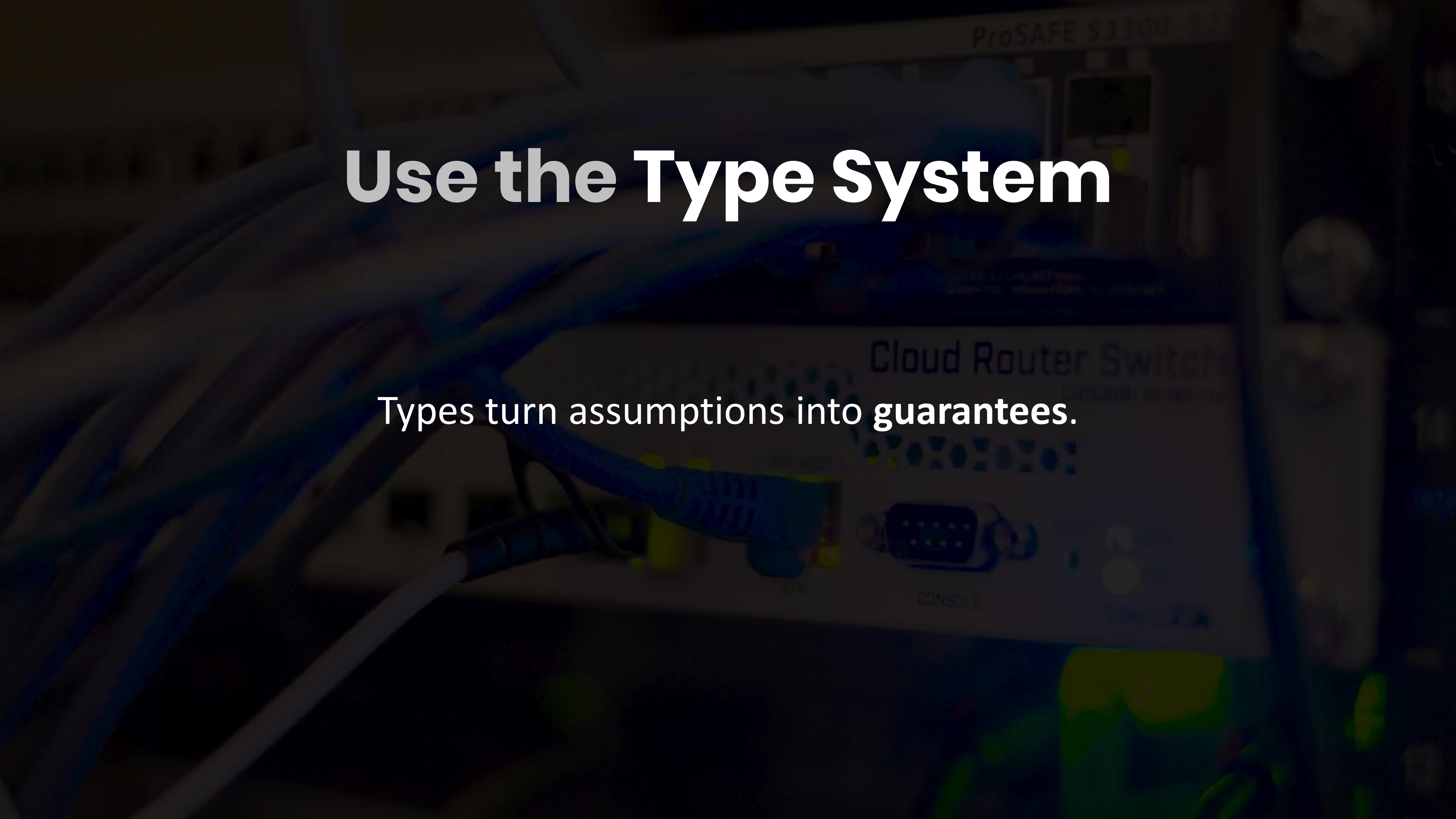


```php
$user = new User();
$user->username = $_POST['username'];
$user->email = $_POST['email'];
// ...
$user->group = 'admin';
```

- HTTP requests (controllers)
- CLI commands
- API endpoints
- Database queries / import scripts
- Message queues

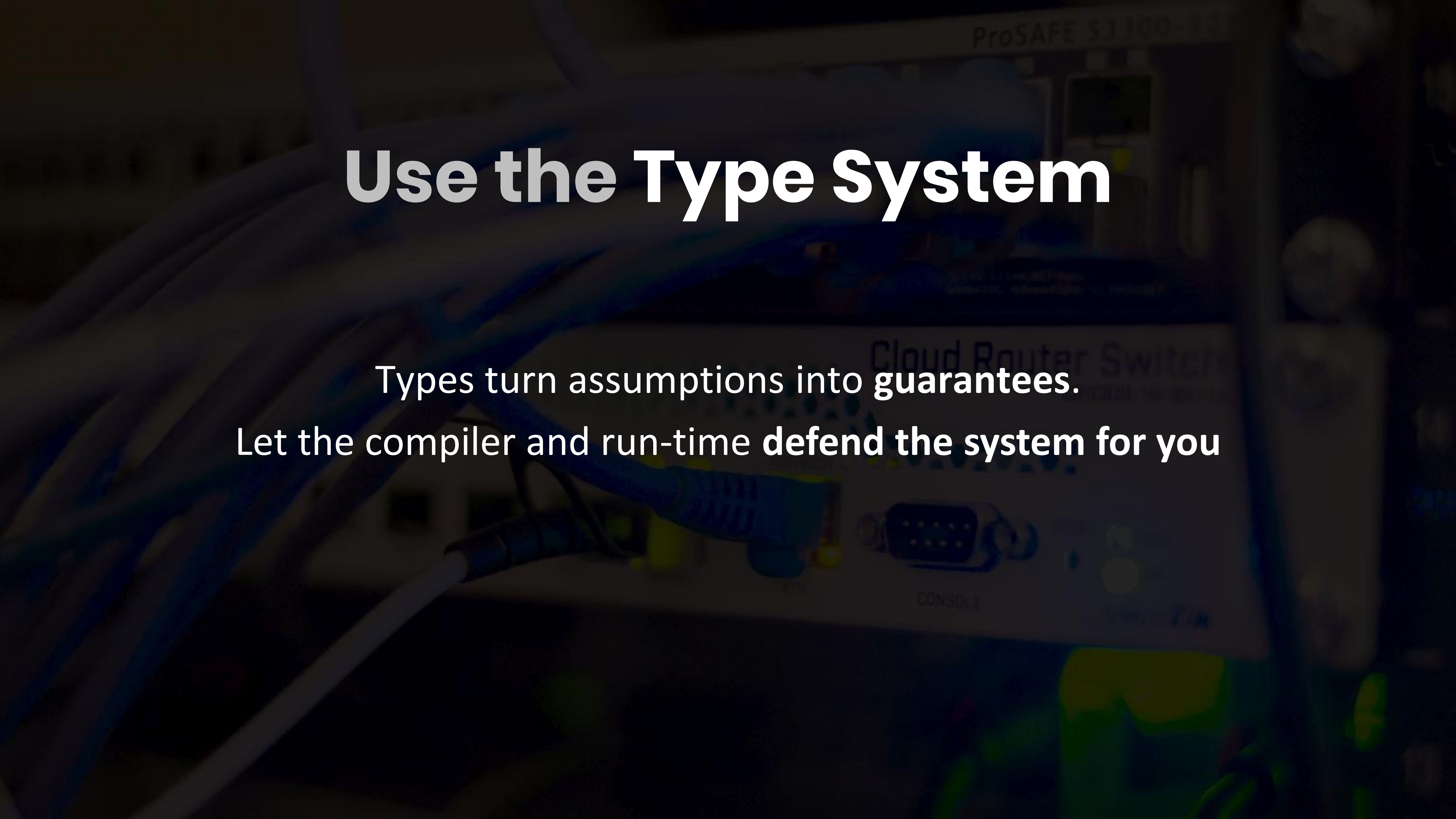


Use the Type System



Use the **Type System**

Types turn assumptions into **guarantees**.

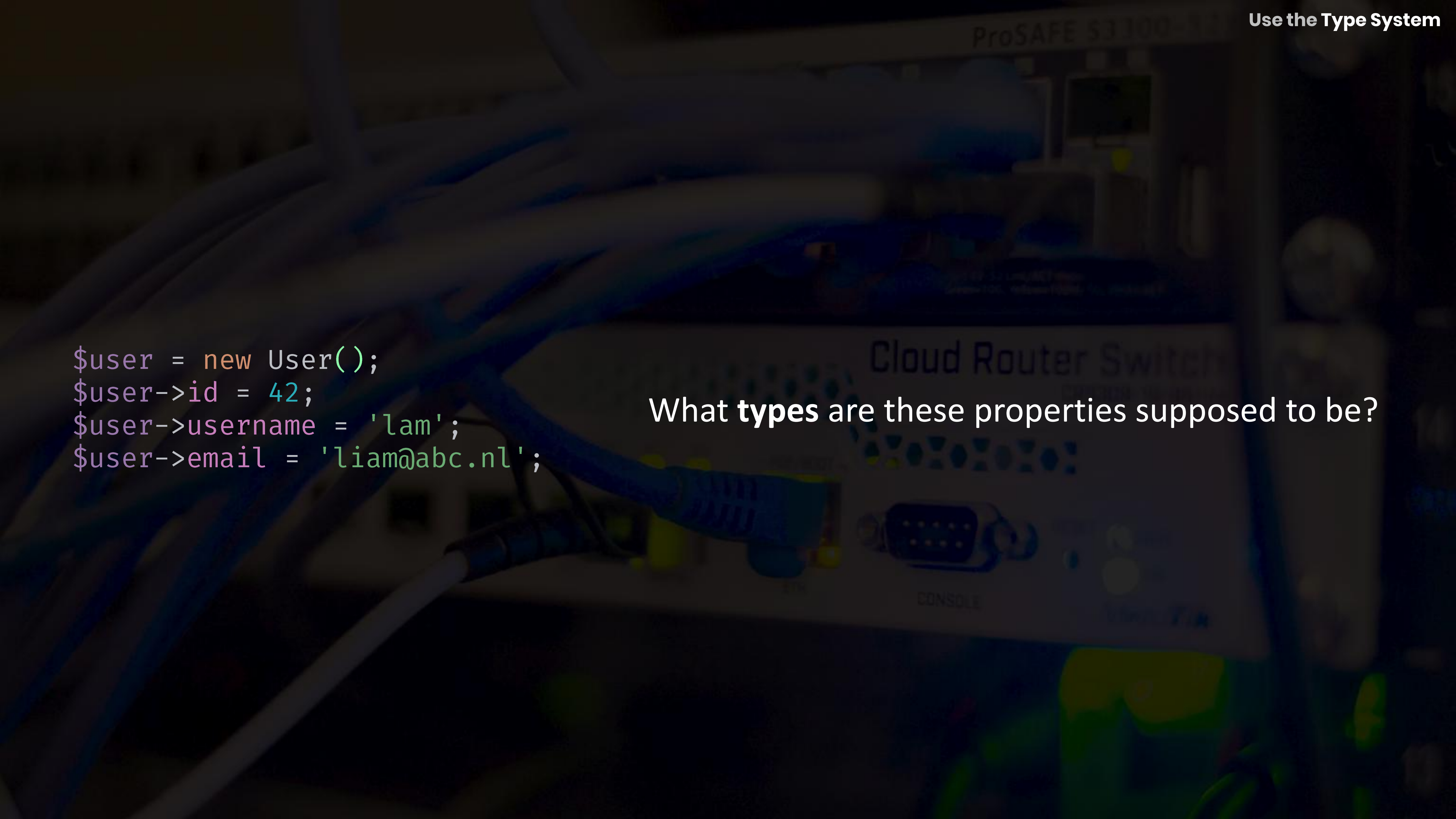


Use the **Type System**

Types turn assumptions into **guarantees**.

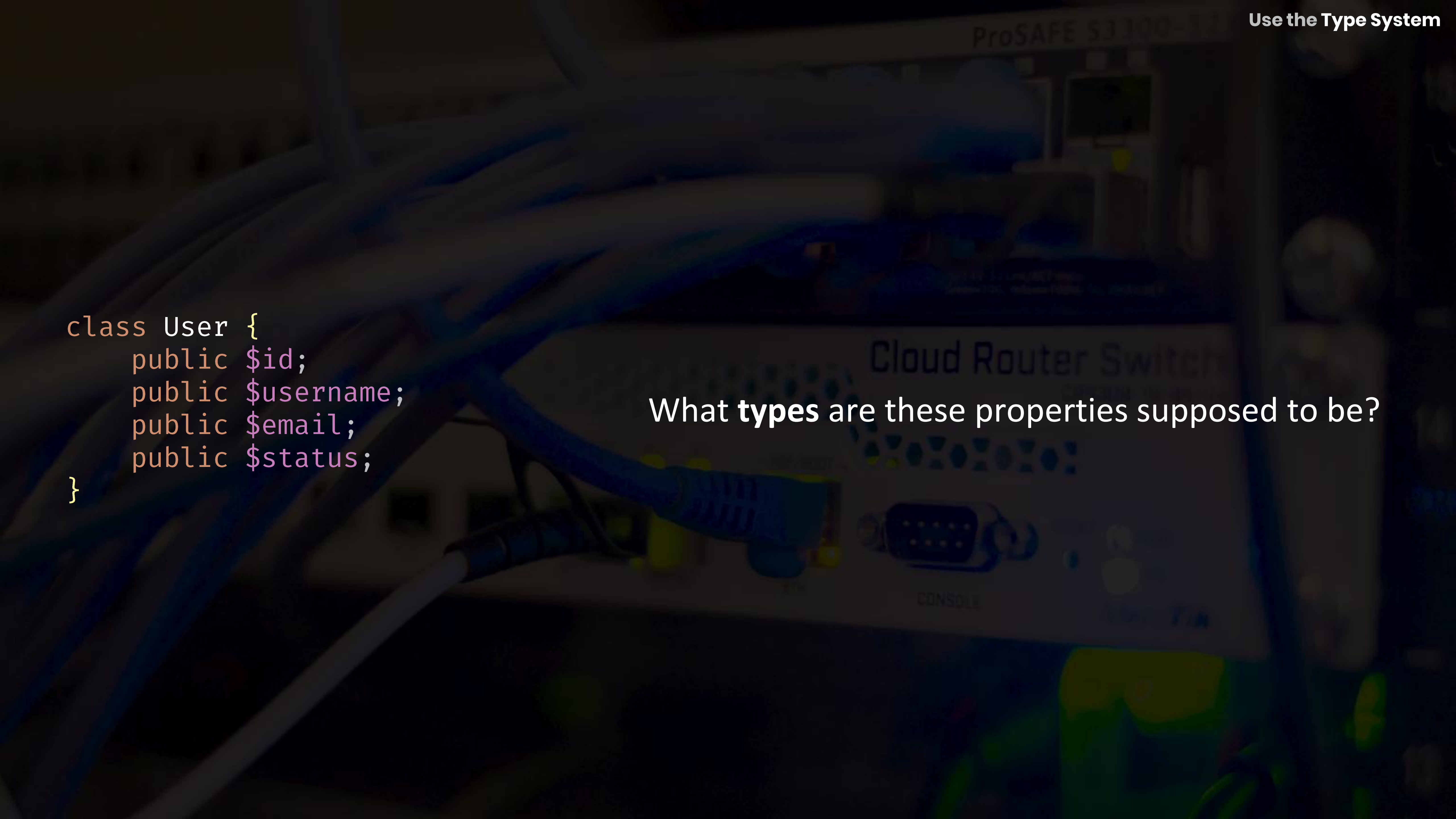
Let the compiler and run-time **defend the system for you**

```php
$user = new User();
$user->id = 42;
$user->username = 'lam';
$user->email = 'liam@abc.nl';
```



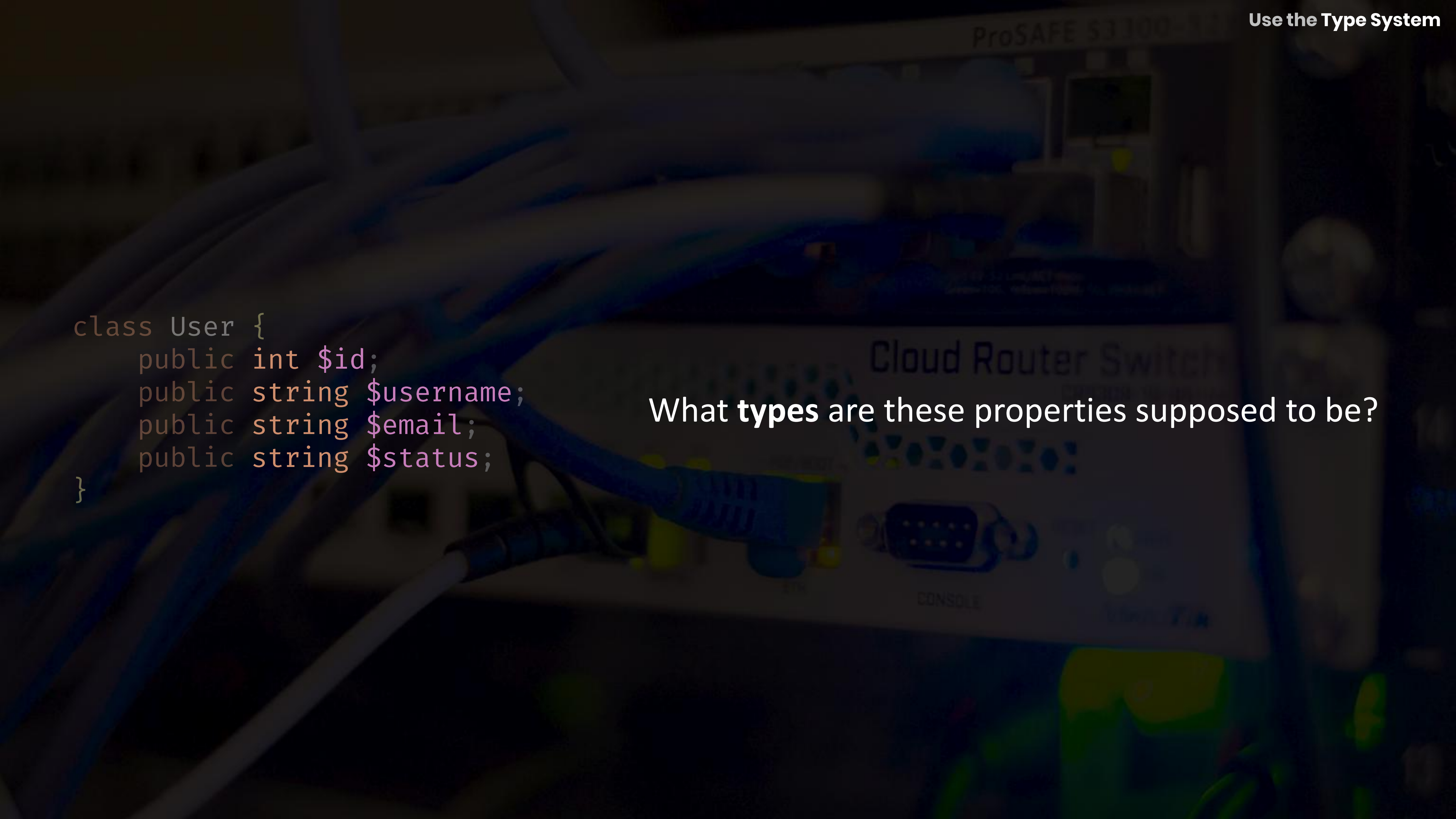
```php
$user = new User();
$user->id = 42;
$user->username = 'lam';
$user->email = 'liam@abc.nl';
```

What **types** are these properties supposed to be?



```
class User {
    public $id;
    public $username;
    public $email;
    public $status;
}
```

What **types** are these properties supposed to be?



```
class User {
    public int $id;
    public string $username;
    public string $email;
    public string $status;
}
```

What **types** are these properties supposed to be?

```
class User {
    public int $id;
    public string $username;
    public string $email;
    public string $status;
}


$user->id = '10 apples';
```

TypeError

What **types** are these properties supposed to be?

```
class User {
    public int $id;
    public string $username;
    public string $email;
    public string $status;
}
```

Avoid **stringly-types**

```php
class User {
    public int $id;
    public string $username;
    public string $email;
    public string $status;
}
```

Avoid **stringly-types**

```php
class User {
    public int $id;
    public string $username;
    public string $email;
    public string $status;
}
```

Avoid **stringly-types**

```php
$user->status = "active";
```

```php
class User {
    public int $id;
    public string $username;
    public string $email;
    public string $status;
}
```

Avoid **stringly-types**

```php
$user->status = "active";
$user->status = "blocked";
```

```
class User {
    public int $id;
    public string $username;
    public string $email;
    public string $status;
}
```

Avoid **stringly-types**

```
$user->status = "active";
$user->status = "blocked";
$user->status = "banana";
```

```
class User {
    public int $id;
    public string $username;
    public string $email;
    public UserStatus $status;
}
```

Use Enums (>= PHP 8.1)

```
enum UserStatus {
    case PENDING;
    case ACTIVE;
    case BLOCKED;
}
```

```php
class User {
    public int $id;
    public string $username;
    public string $email;
    public UserStatus $status;
}


enum UserStatus {
    case PENDING;
    case ACTIVE;
    case BLOCKED;
}


$user->status = UserStatus::ACTIVE;
$user->status = UserStatus::BLOCKED;
$user->status = UserStatus::PENDING;
```

Use Enums (>= PHP 8.1)

```
class User {
    public int $id;
    public string $username;
    public string $email;
    public UserStatus $status;
}
```

Extend with Value Objects

```php
class User {
    public int $id;
    public string $username;
    public string $email;
    public UserStatus $status;
}


$user->email = 'hi@example.com';
```

Extend with Value Objects

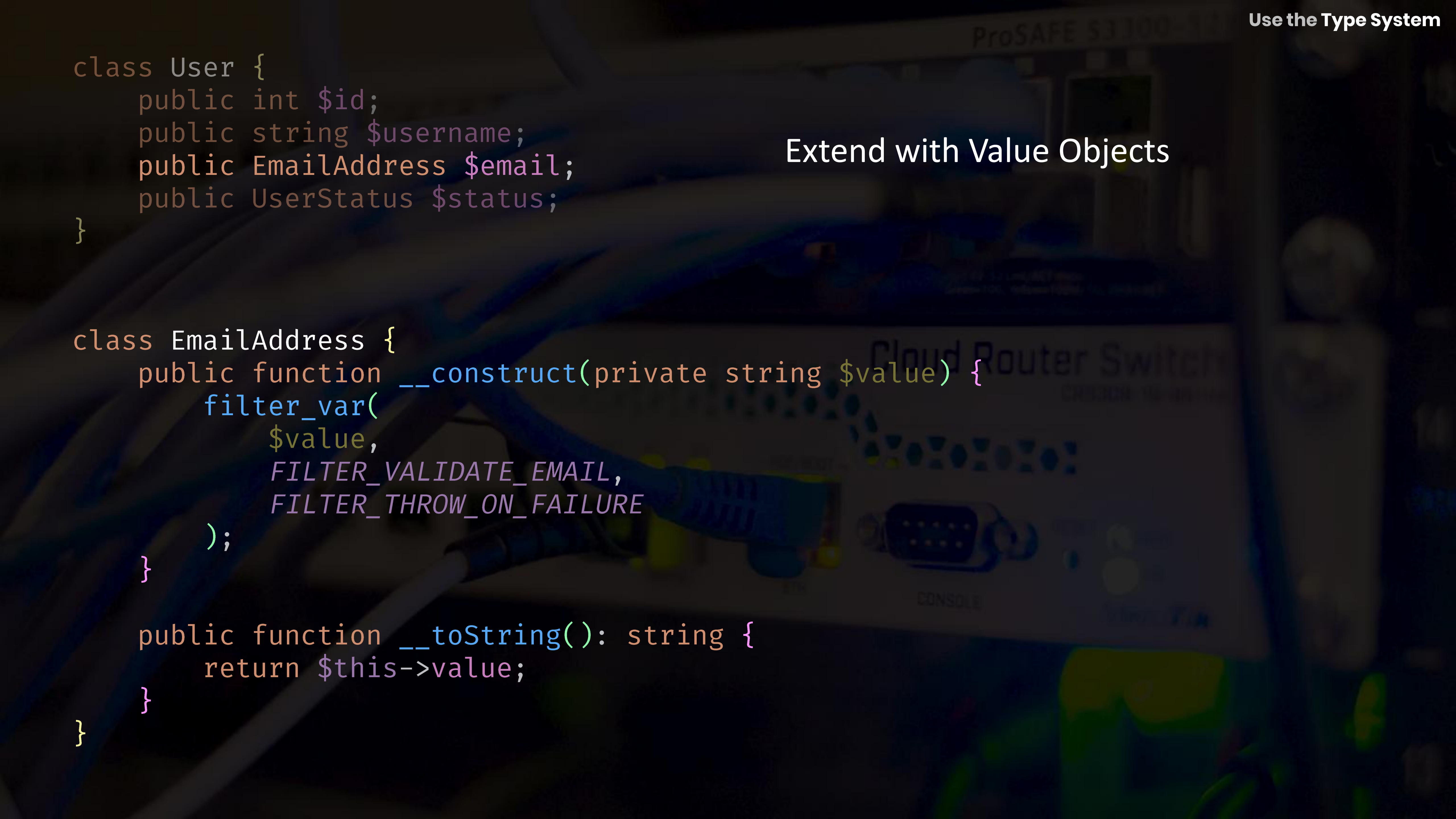
```
class User {
    public int $id;
    public string $username;
    public string $email;
    public UserStatus $status;
}


$user->email = 'hi@example.com';
$user->email = 'hi@localhost';
```

Extend with Value Objects

```php
class User {
    public int $id;
    public string $username;
    public string $email;
    public UserStatus $status;
}
```

Extend with Value Objects

```php
$user->email = 'hi@example.com';
$user->email = 'hi@localhost';
$user->email = '#DPC26';
```



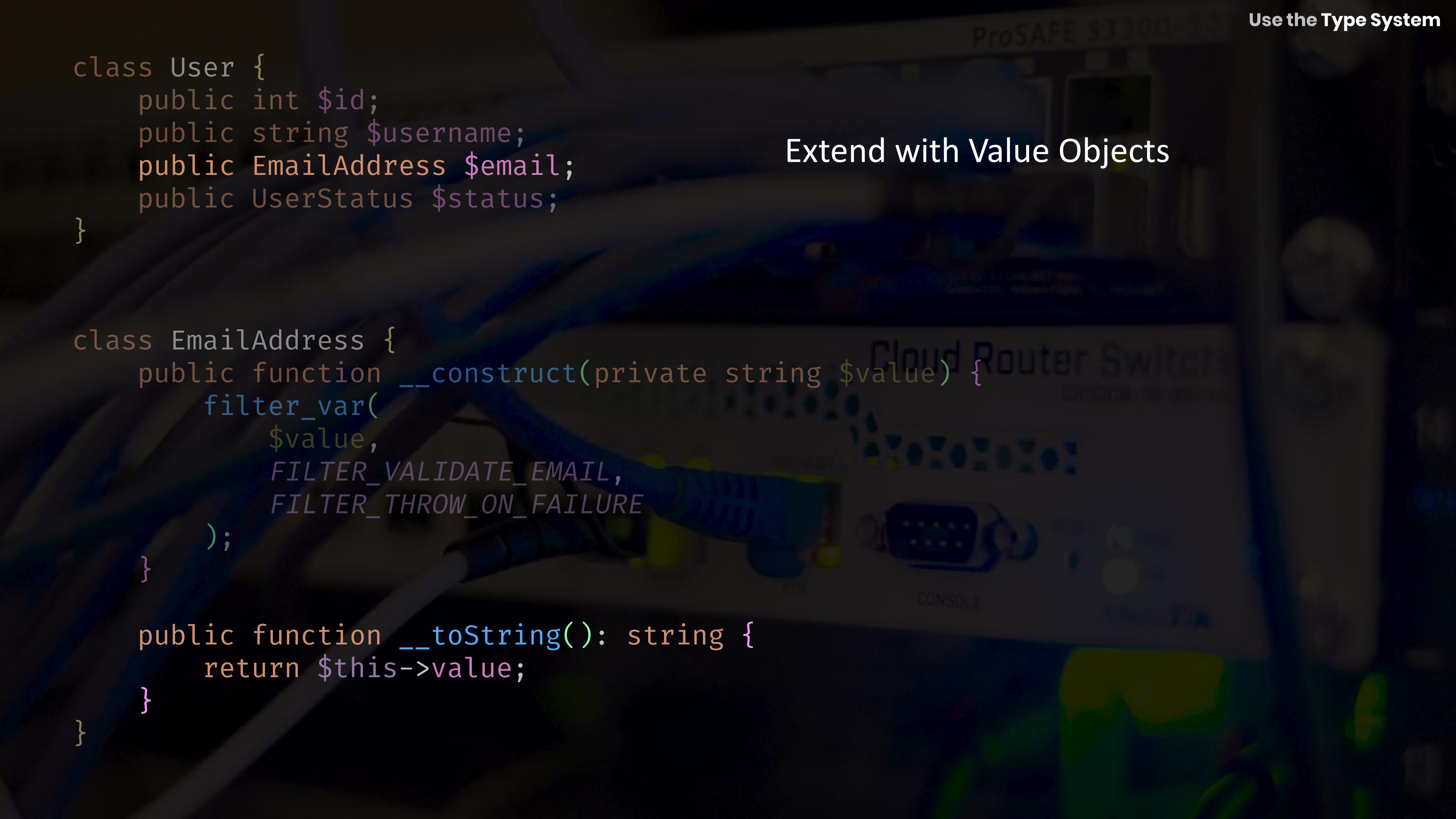
Extend with Value Objects

```php
class User {
    public int $id;
    public string $username;
    public EmailAddress $email;
    public UserStatus $status;
}



class EmailAddress {
    public function __construct(private string $value) {
        filter_var(
            $value,
            FILTER_VALIDATE_EMAIL,
            FILTER_THROW_ON_FAILURE
        );
    }

    public function __toString(): string {
        return $this->value;
    }
}
```

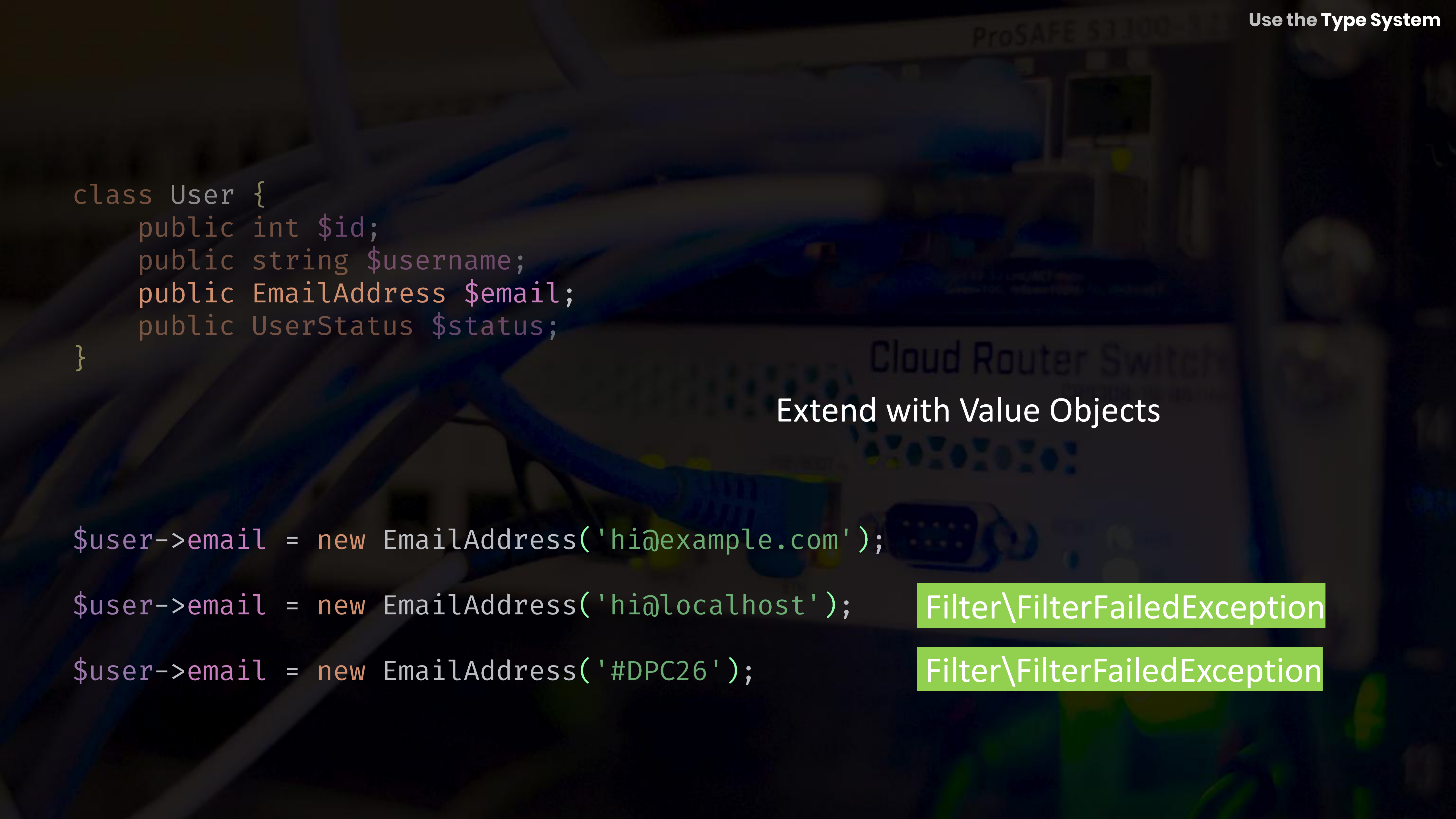
Extend with Value Objects

```
class User {
    public int $id;
    public string $username;
    public EmailAddress $email;
    public UserStatus $status;
}


class EmailAddress {
    public function __construct(private string $value) {
        filter_var(
            $value,
            FILTER_VALIDATE_EMAIL,
            FILTER_THROW_ON_FAILURE
        );
    }

    public function __toString(): string {
        return $this->value;
    }
}
```

Extend with Value Objects



```
class User {
    public int $id;
    public string $username;
    public EmailAddress $email;
    public UserStatus $status;
}


class EmailAddress {
    public function __construct(private string $value) {
        filter_var(
            $value,
            FILTER_VALIDATE_EMAIL,
            FILTER_THROW_ON_FAILURE
        );
    }

    public function __toString(): string {
        return $this->value;
    }
}
```

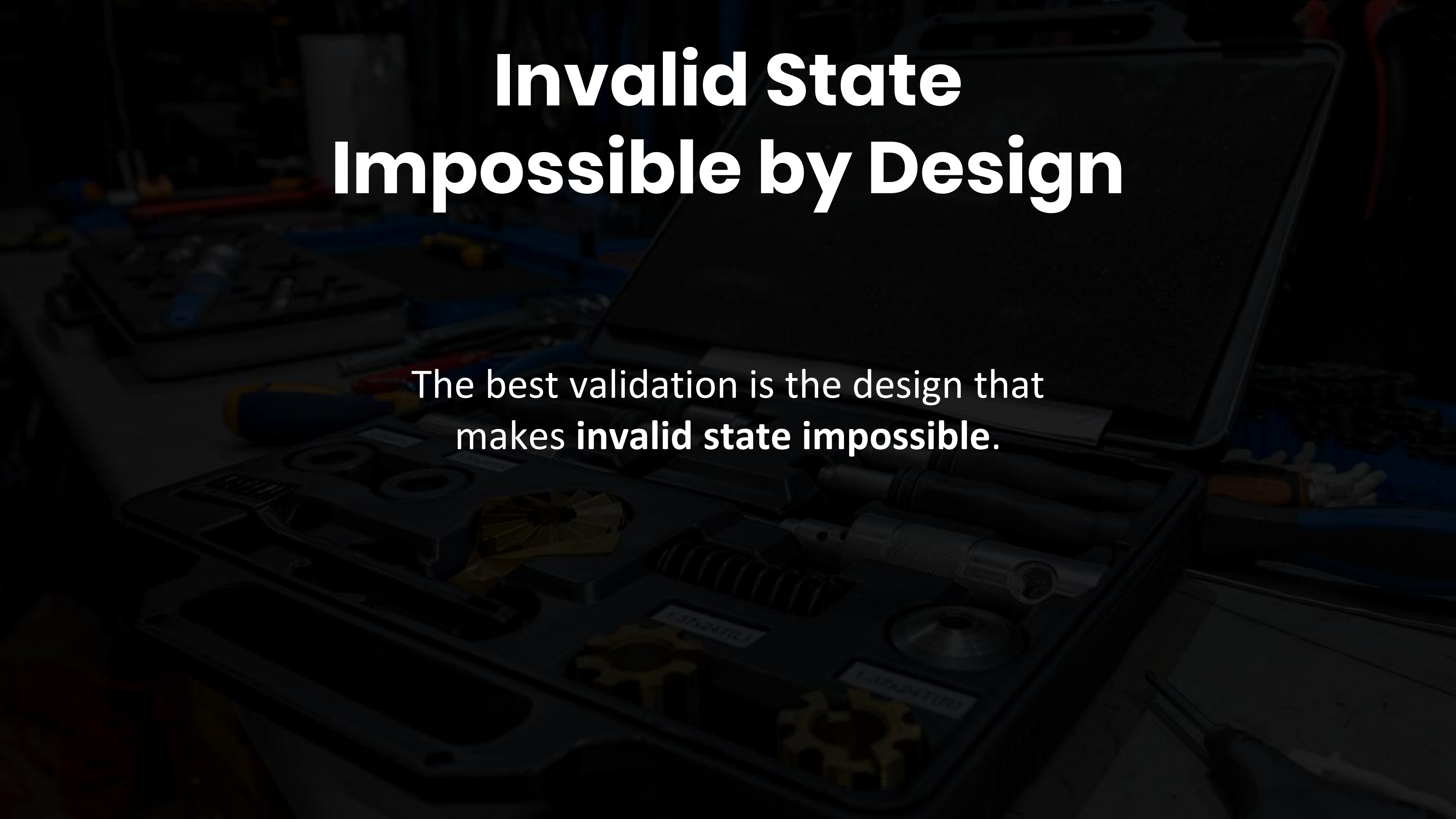


```
class User {
    public int $id;
    public string $username;
    public EmailAddress $email;
    public UserStatus $status;
}
```

Extend with Value Objects

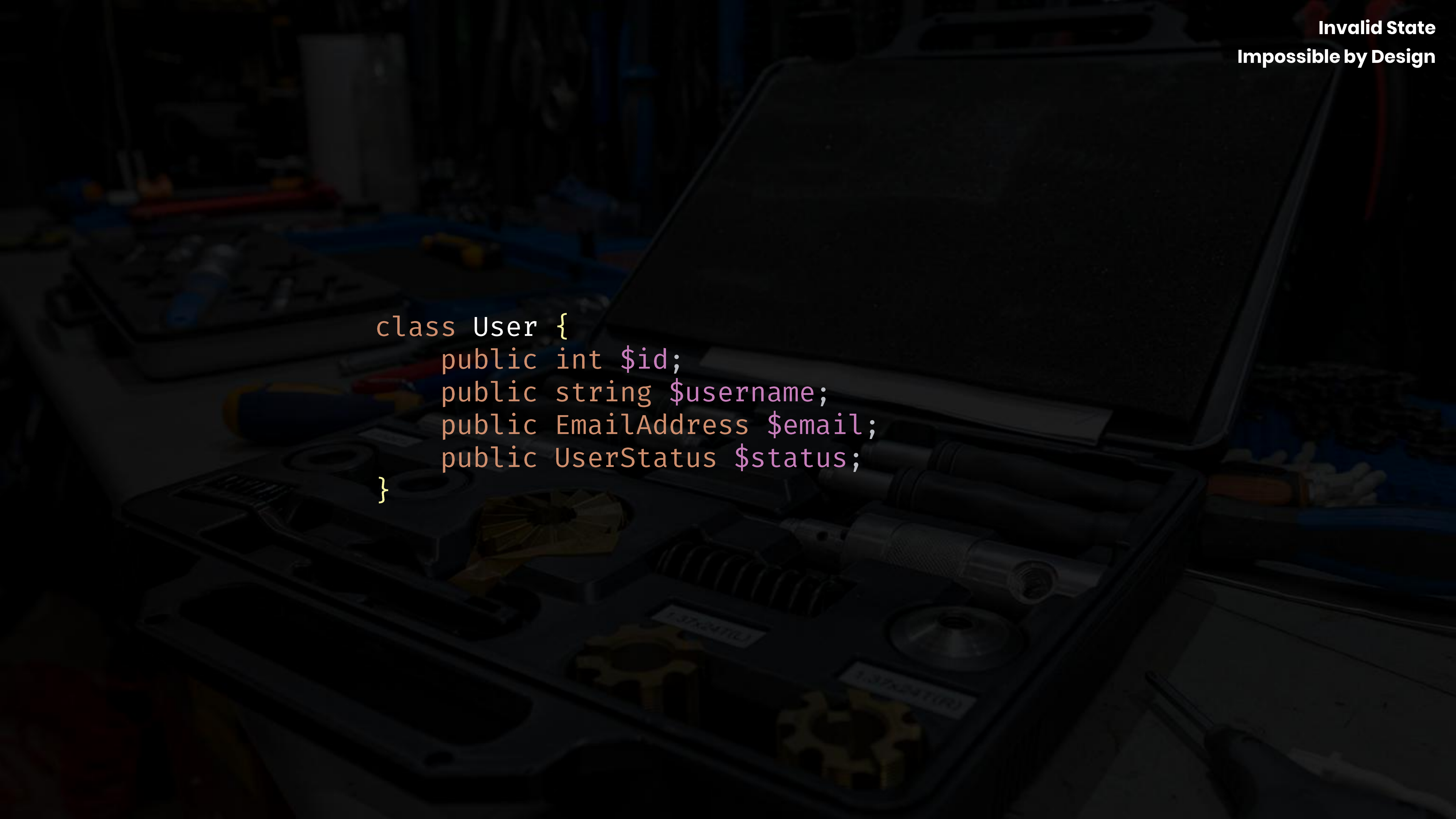
```
$user->email = new EmailAddress('hi@example.com');

$user->email = new EmailAddress('hi@localhost');      Filter\FilterFailedException

$user->email = new EmailAddress('#DPC26');            Filter\FilterFailedException
```

Invalid State Impossible by Design

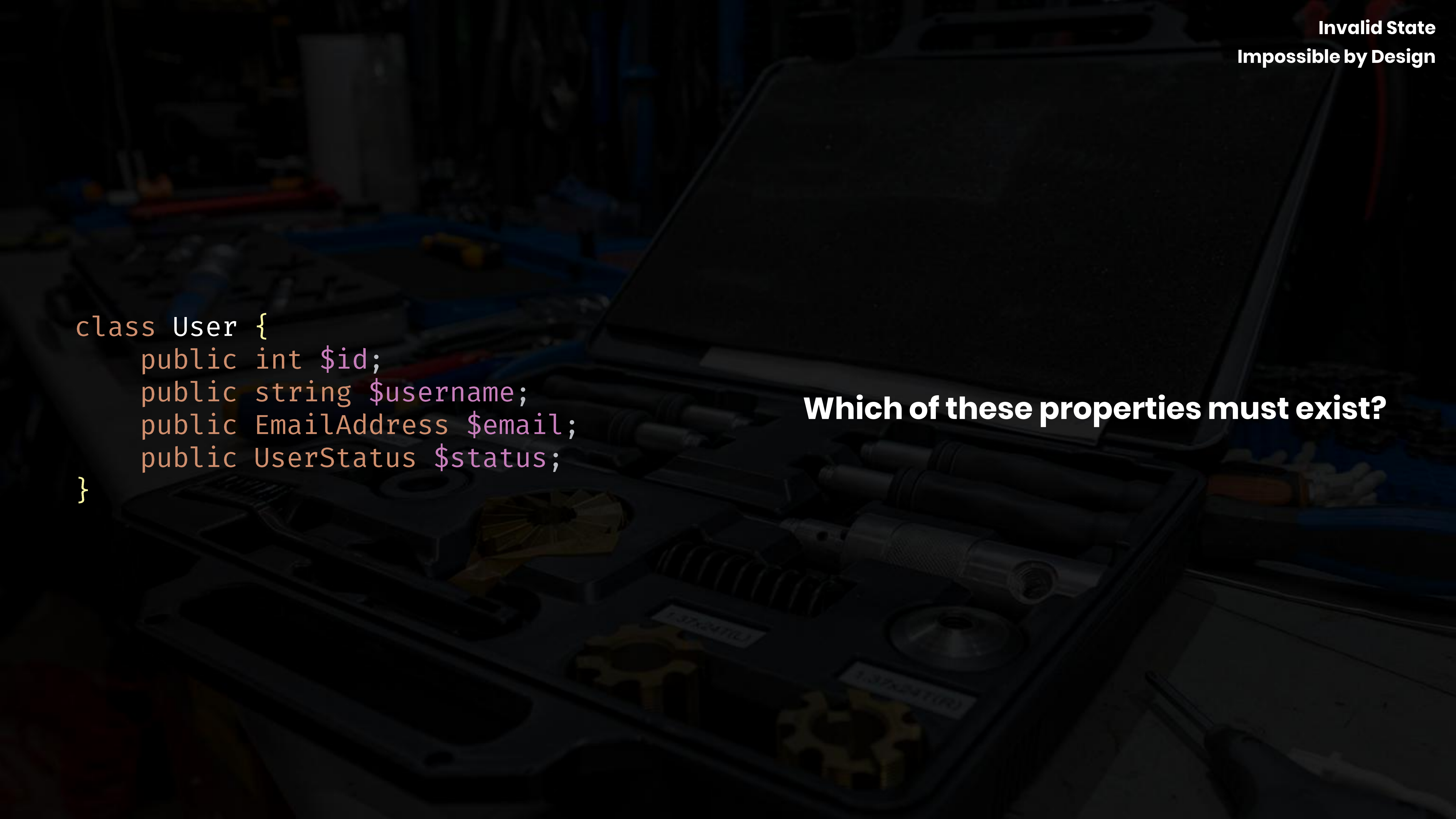


Invalid State Impossible by Design

The best validation is the design that makes **invalid state impossible**.



```
class User {
    public int $id;
    public string $username;
    public EmailAddress $email;
    public UserStatus $status;
}
```



```
class User {
    public int $id;
    public string $username;
    public EmailAddress $email;
    public UserStatus $status;
}
```

Which of these properties must exist?

```
class User {
    public int $id;
    public string $username;
    public EmailAddress $email;
    public UserStatus $status = UserStatus::ACTIVE;

    public function __construct(
        int $id,
        string $username,
        EmailAddress $email,
    ) {

    }
}
```

Which of these properties must exist?

They must be initialized at the constructor

Make state transitions explicit

Invalid State Impossible by Design

```
class User {
    public int $id;
    public string $username;
    public EmailAddress $email;
    public UserStatus $status = UserStatus::ACTIVE;

    public function __construct(...) {}

    public function block(): void {
        $this->status = UserStatus::BLOCKED;
    }
}
```

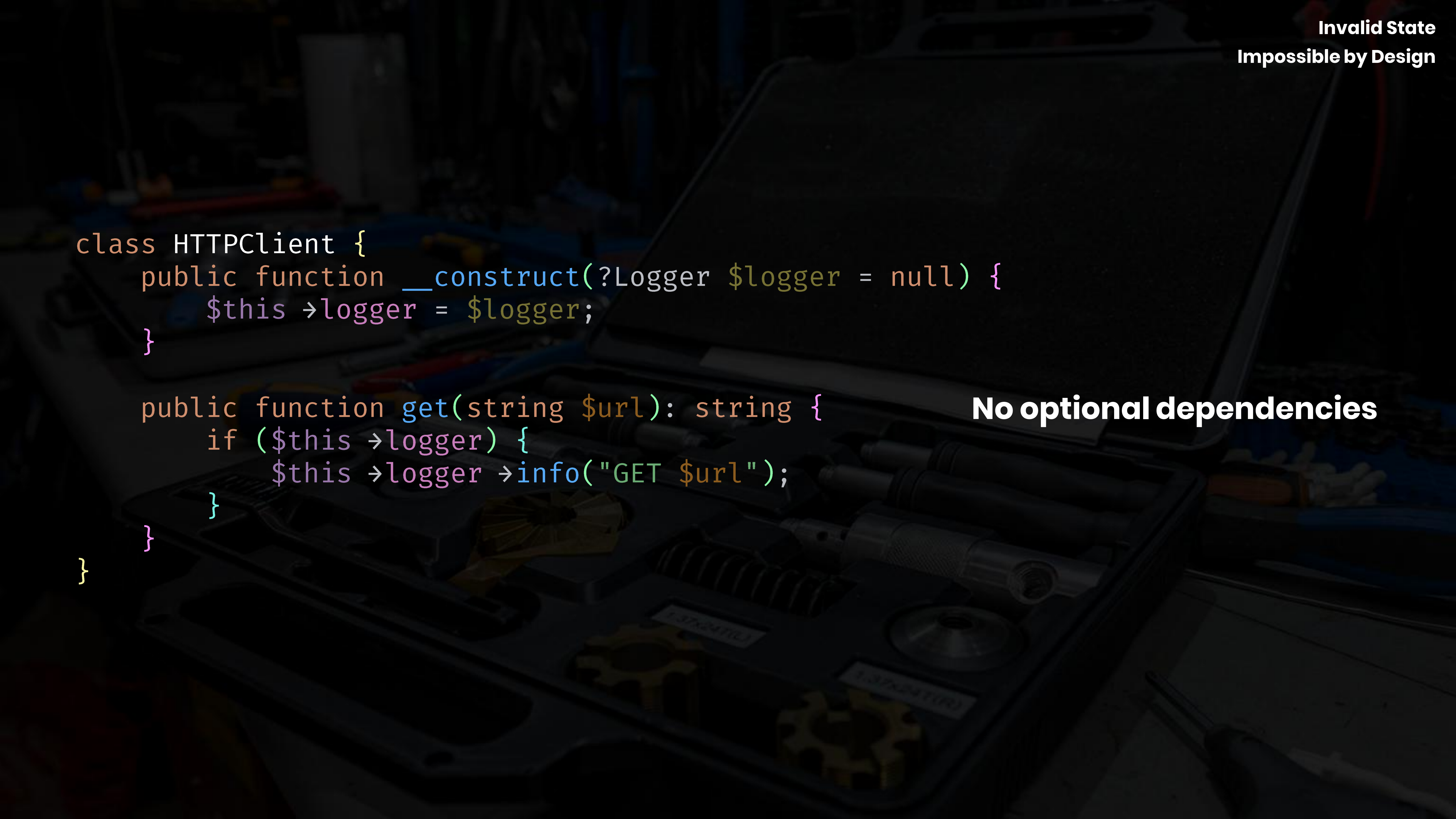
Make state transitions explicit

Invalid State
Impossible by Design

```
class User {
    public int $id;
    public string $username;
    public EmailAddress $email;
    private UserStatus $status = UserStatus::ACTIVE;

    public function __construct(...) {}

    public function block(): void {
        $this->status = UserStatus::BLOCKED;
    }
}
```

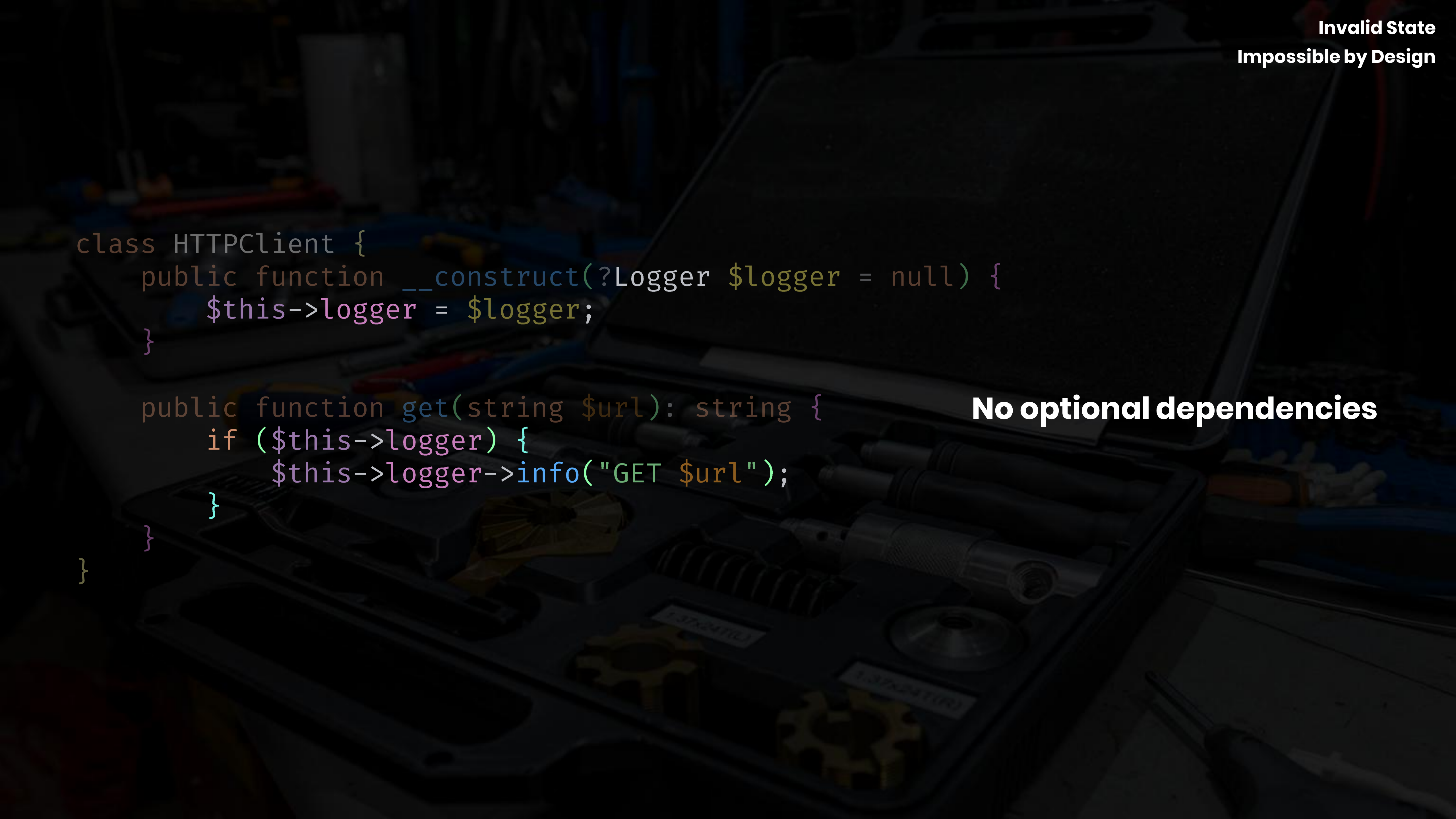
Make state transitions explicit

No optional dependencies



```php
class HTTPClient {
    public function __construct(?Logger $logger = null) {
        $this->logger = $logger;
    }

    public function get(string $url): string {
        if ($this->logger) {
            $this->logger->info("GET $url");
        }
    }
}
```

No optional dependencies



```php
class HTTPClient {
    public function __construct(?Logger $logger = null) {
        $this->logger = $logger;
    }

    public function get(string $url): string {
        if ($this->logger) {
            $this->logger->info("GET $url");
        }
    }
}
```

No optional dependencies

```php
class HTTPClient {
    public function __construct(Logger $logger) {
        $this->logger = $logger;
    }

    public function get(string $url): string {
        $this->logger->info("GET $url");
    }
}

$client = new HTTPClient(new NullLogger());
```

No optional dependencies

Immutability and Integrity

Immutability and Integrity

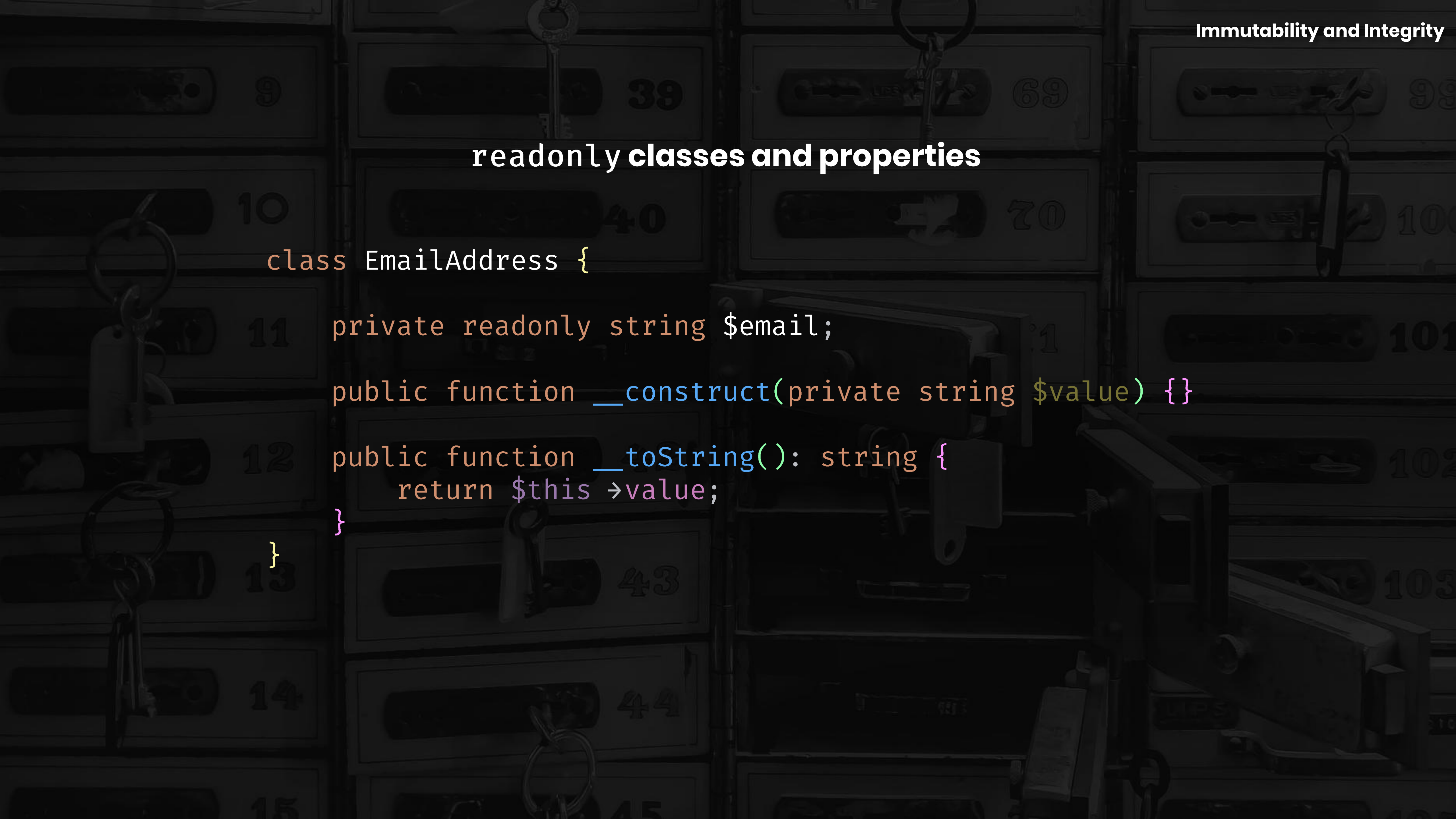
Validation at creation is not enough
if the object can be corrupted later.

Immutability and Integrity

Validation at creation is not enough if the object can be corrupted later.

Once created, its core properties should not be freely mutable.

`readonly` classes and properties



`readonly` classes and properties

```php
class EmailAddress {

    private readonly string $email;

    public function __construct(private string $value) {}

    public function __toString(): string {
        return $this->value;
    }
}
```

`readonly` classes and properties

```php
class EmailAddress {

    private readonly string $email;

    public function __construct(private string $value) {}

    public function __toString(): string {
        return $this->value;
    }
}
```

readonly classes and properties

```
class EmailAddress {

    private readonly string $email;

    public function __construct(private string $value) {}

    public function __toString(): string {
        return $this->value;
    }
}
```

`readonly` classes and properties

```
readonly class EmailAddress {

    private string $email;

    public function __construct(private string $value) {}

    public function __toString(): string {
        return $this->value;
    }
}
```

Asymmetric Visibility

Asymmetric Visibility

```php
class Event {
    public private(set) int $name;

    public function __construct(string $name) {
        $this->name = 'DPC26';
    }
}
```

Asymmetric Visibility

```
class Event {
    public private(set) int $name;

    public function __construct(string $name) {
        $this->name = 'DPC26';
    }
}
```

Read as if it is a public property

Asymmetric Visibility

```
class Event {
    public private(set) int $name;

    public function __construct(string $name) {
        $this->name = 'DPC26';
    }
}
```

Read as if it is a public property

Prevent writing from outside

Asymmetric Visibility

```
class Event {
    public private(set) int $name;

    public function __construct(string $name) {
        $this->name = 'DPC26';
    }
}



$event = new Event();

echo $event->name; "DPC26"
```

Read as if it is a public property

Prevent writing from outside

Asymmetric Visibility

```php
class Event {
    public private(set) int $name;

    public function __construct(string $name) {
        $this->name = 'DPC26';
    }
}



$event = new Event();

echo $event →name;

$event →name = 'DPC26';
```

Read as if it is a public property

Prevent writing from outside

Error: : Cannot modify private(set) property Event::$name from global scope

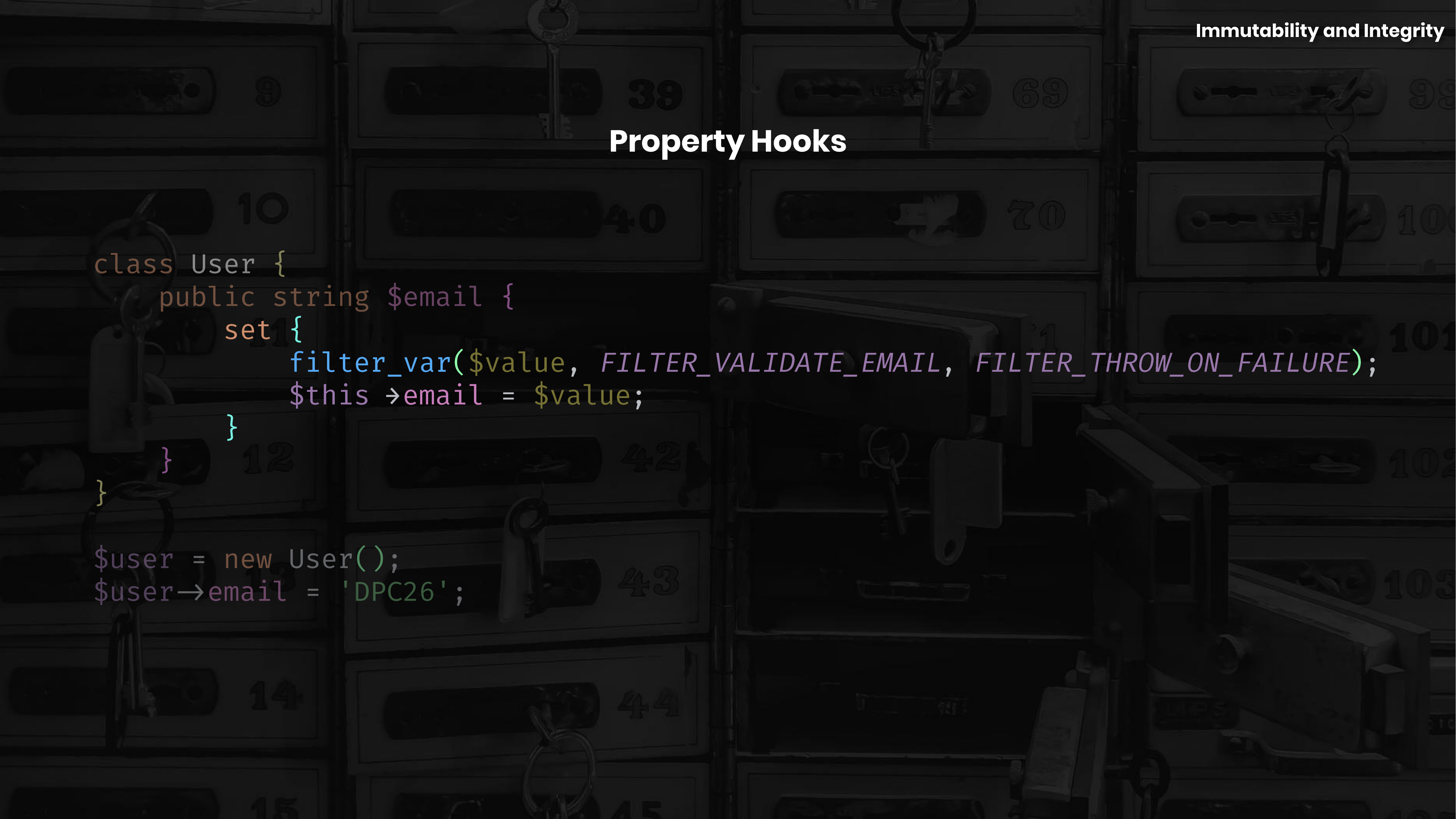
Property Hooks

Property Hooks

- Add additional logic when setting and getting properties
- Virtual property support
- Retroactively add validation to existing classes

Property Hooks

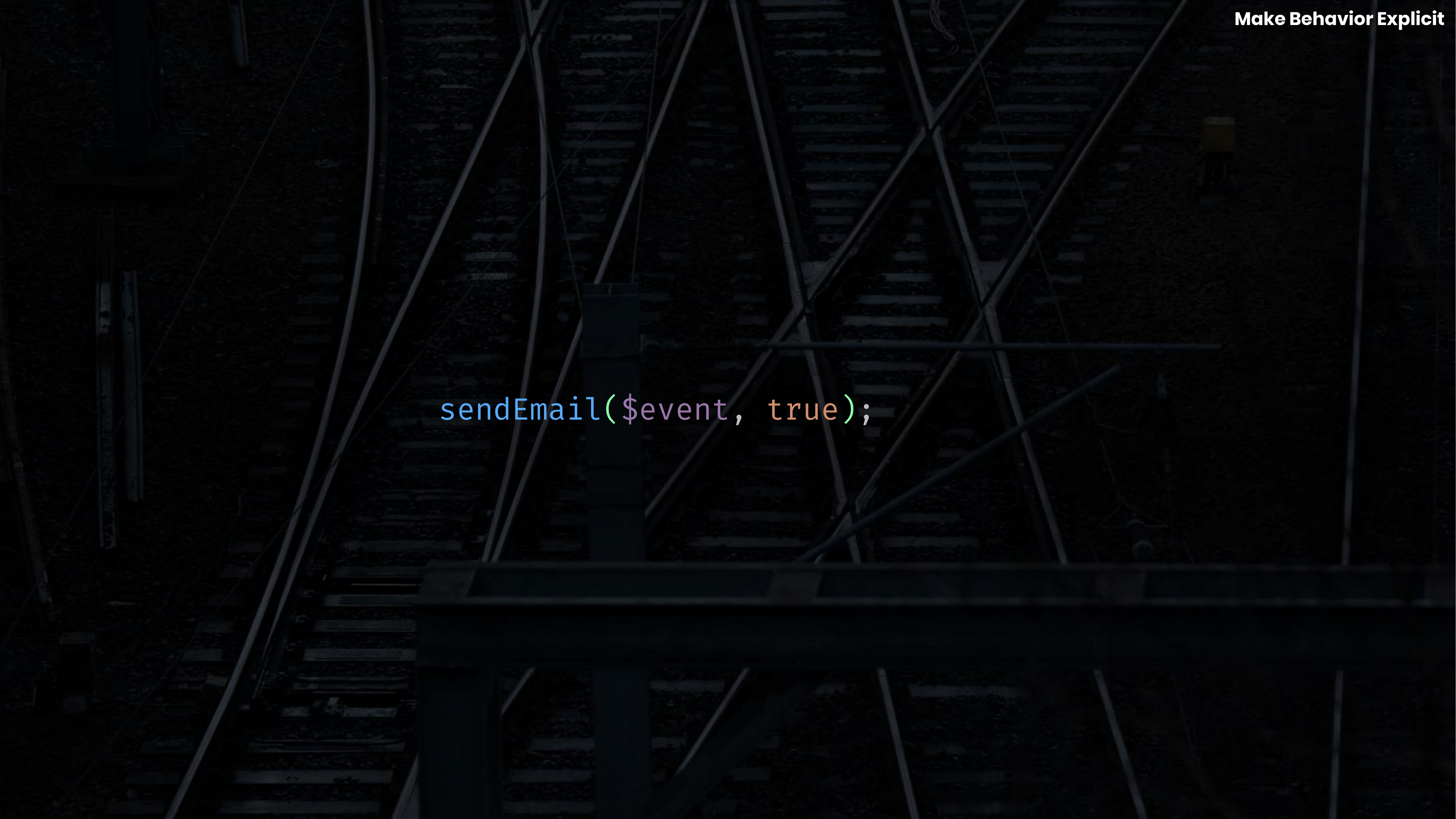
```php
class User {
    public string $email {
        set {
            filter_var($value, FILTER_VALIDATE_EMAIL, FILTER_THROW_ON_FAILURE);
            $this->email = $value;
        }
    }
}

$user = new User();
$user->email = 'DPC26';
```



Property Hooks

```
class User {
    public string $email {
        set {
            filter_var($value, FILTER_VALIDATE_EMAIL, FILTER_THROW_ON_FAILURE);
            $this ->email = $value;
        }
    }
}

$user = new User();
$user->email = 'DPC26';
```

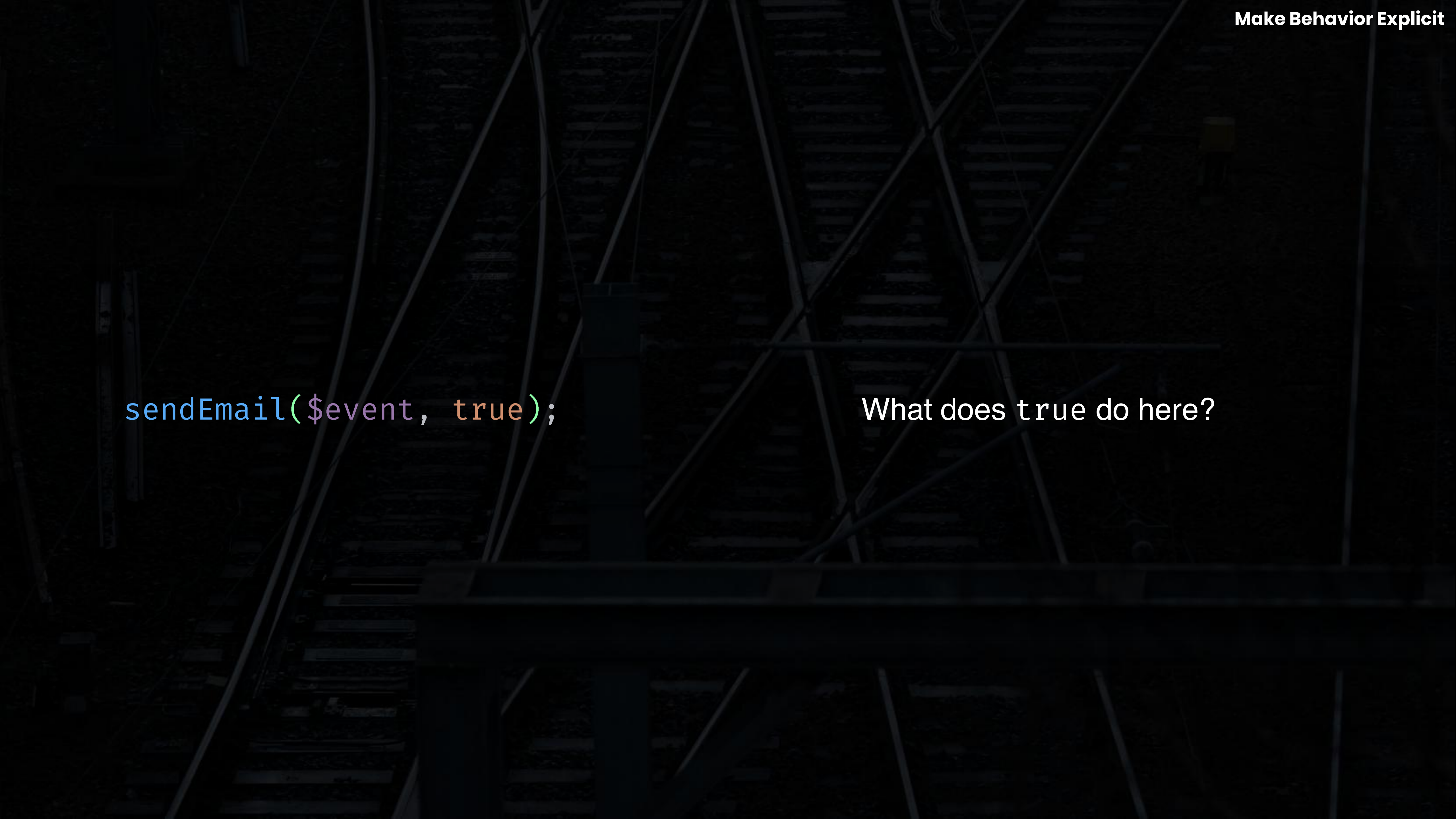
Make Behavior Explicit



Make Behavior Explicit

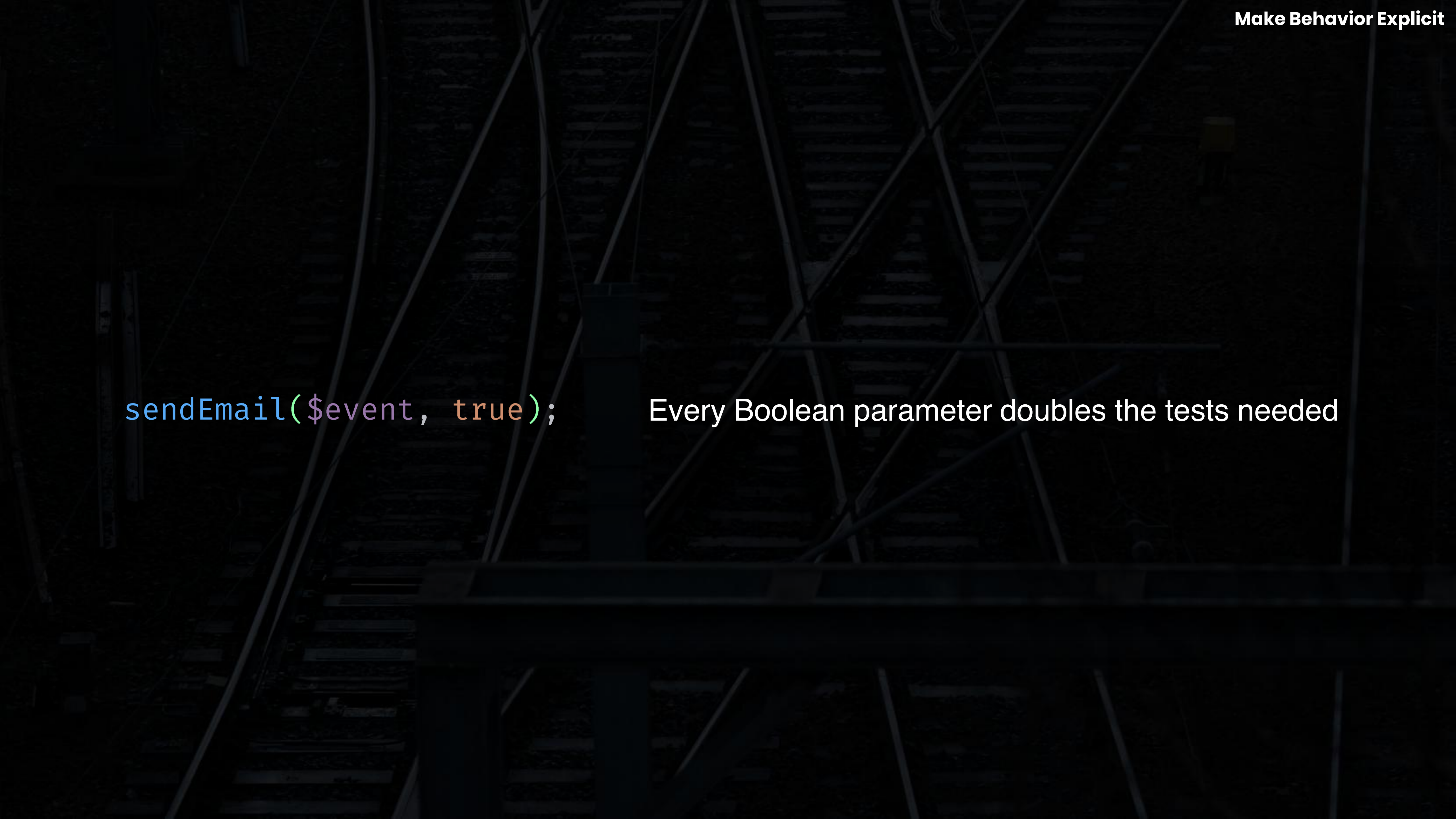
```php
sendEmail($event, true);
```

Make Behavior Explicit



```
sendEmail($event, true);
```

What does `true` do here?



```
sendEmail($event, true);
```

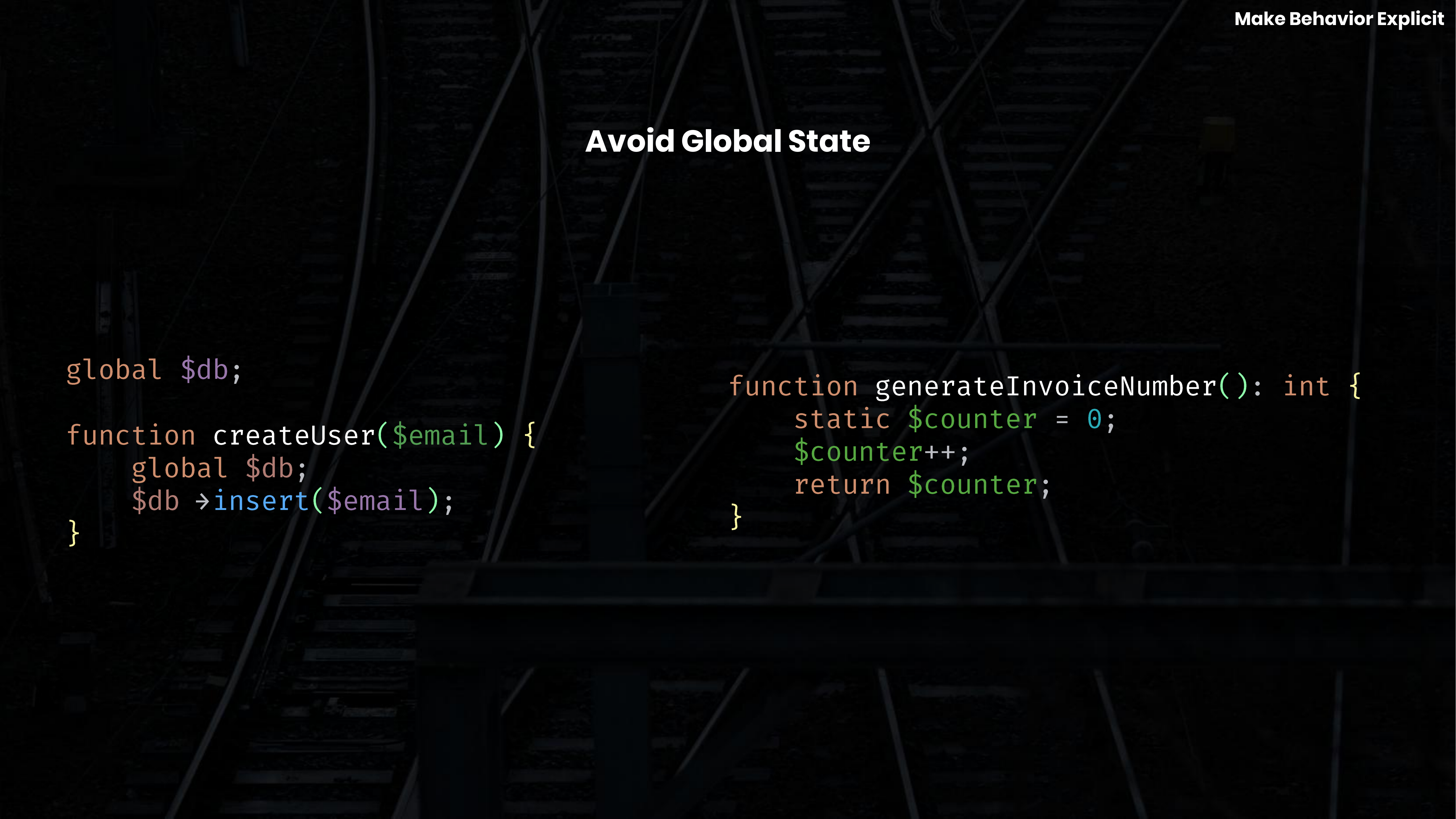
Every Boolean parameter doubles the tests needed

Make Behavior Explicit

```
~~sendEmail($event, true);~~


sendEmail($event);
sendEmailAndPushNotification($event);
```

Make Behavior Explicit

```
~~sendEmail($event, true);~~


~~sendEmail($event);~~
~~sendEmailAndPushNotification($event);~~


enum NotificationChannel {
    case EMAIL;
    case PUSH_NOTIFICATION;
    case SMS;

}


sendNotification(NotificationChannel::EMAIL);
```



Avoid Global State

```php
global $db;

function createUser($email) {
    global $db;
    $db->insert($email);
}
```

```php
function generateInvoiceNumber(): int {
    static $counter = 0;
    $counter++;
    return $counter;
}
```

Avoid Global State

```
global $db;

function createUser($email) {
    global $db;
    $db->insert($email);
}
```

```
function generateInvoiceNumber(): int {
    static $counter = 0;
    $counter++;
    return $counter;
}
```

Avoid Global State

```
global $db;

function createUser($email) {
    global $db;
    $db →insert($email);
}
```

```
function generateInvoiceNumber(): int {
    static $counter = 0;
    $counter++;
    return $counter;
}
```

- Hard to test
- Not deterministic
- State is shared globally
- Difficult to reset

Go *extra* on Exceptions

Many PHP APIs still fail silently unless you opt into exceptions

```php
$pdo = new PDO($dsn, $user, $password);

$pdo->setAttribute(
    PDO::ATTR_ERRMODE,
    PDO::ERRMODE_EXCEPTION
);
```

Go *extra* on Exceptions

Many PHP APIs still fail silently unless you opt into exceptions

```php
$data = json_decode(
    $json,
    true,
    flags: JSON_THROW_ON_ERROR
);
```

Make Behavior Explicit

- #[Deprecated] Attribute for deprecated functionality
- #[Override] to ensure the parent method exists
- #[NoDiscard] to make sure the return value is not discarded
- `match` expression over `switch`
- Named parameters for ambiguous parameters
- Meaningful parameter names
- Strongly typed parameter, return values, and properties
- static lambda functions: static fn() ⇒ 42
- Unset for/foreach/while look varables

Small, focused, and predictable components

Small, focused, and predictable components

Complexity and hidden behavior are the enemy of defensive code

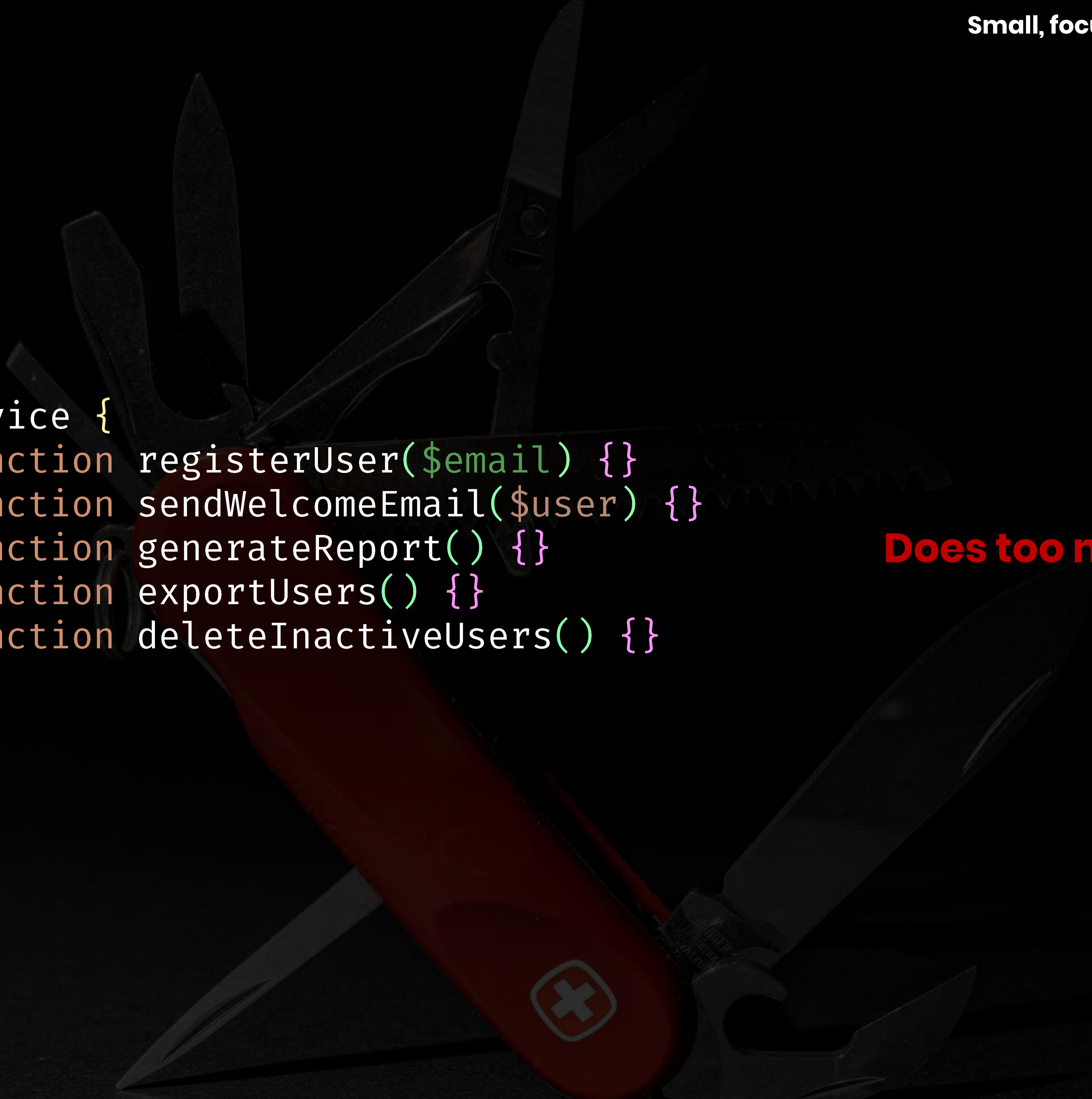
Small, focused, and predictable components

Complexity and hidden behavior are the enemy of defensive code

Small components make bugs easier to find — and harder to create

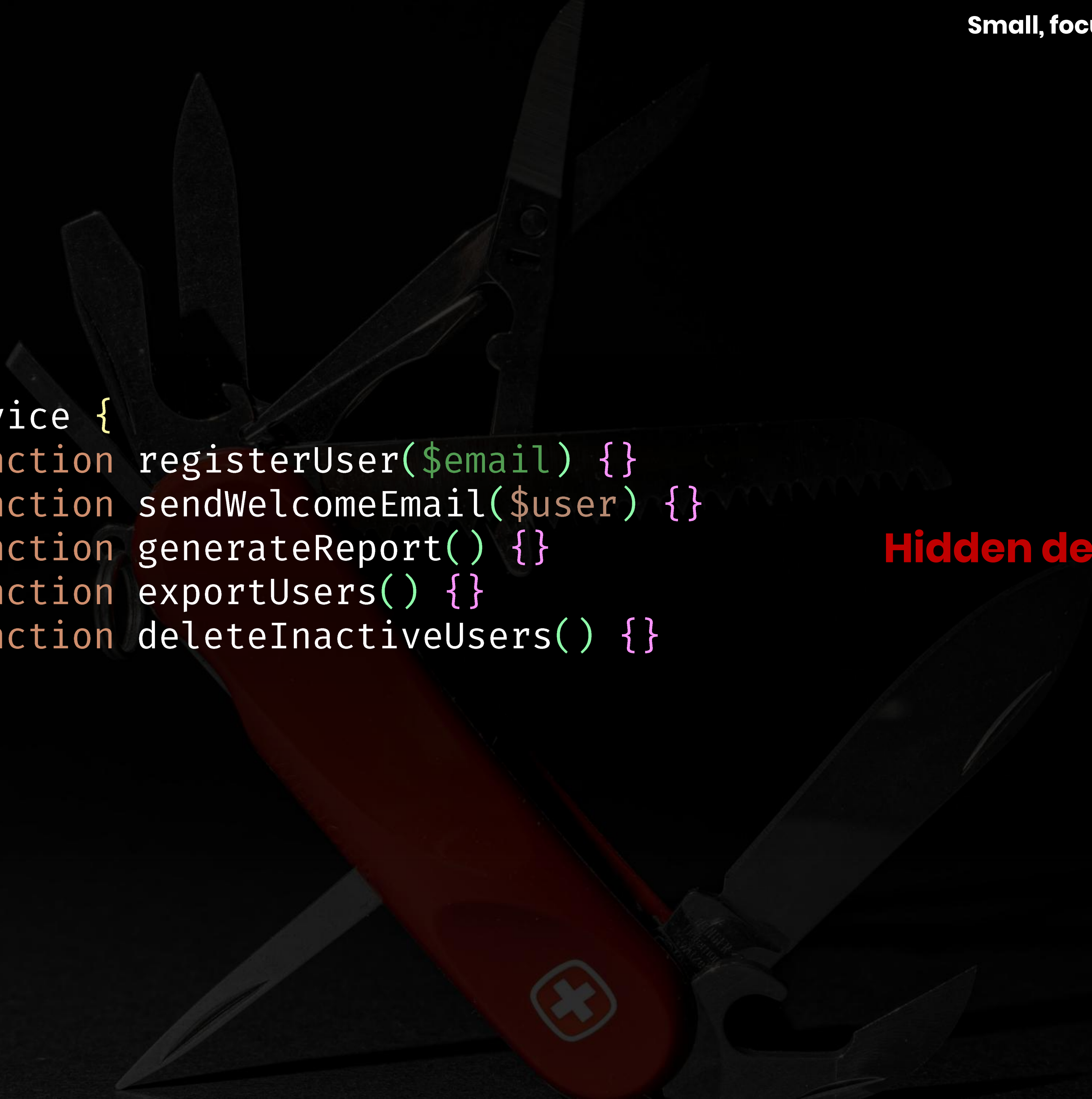
Small, focused, and predictable components

```php
class UserService {
    public function registerUser($email) {}
    public function sendWelcomeEmail($user) {}
    public function generateReport() {}
    public function exportUsers() {}
    public function deleteInactiveUsers() {}
}
```



```
class UserService {
    public function registerUser($email) {}
    public function sendWelcomeEmail($user) {}
    public function generateReport() {}
    public function exportUsers() {}
    public function deleteInactiveUsers() {}
}
```

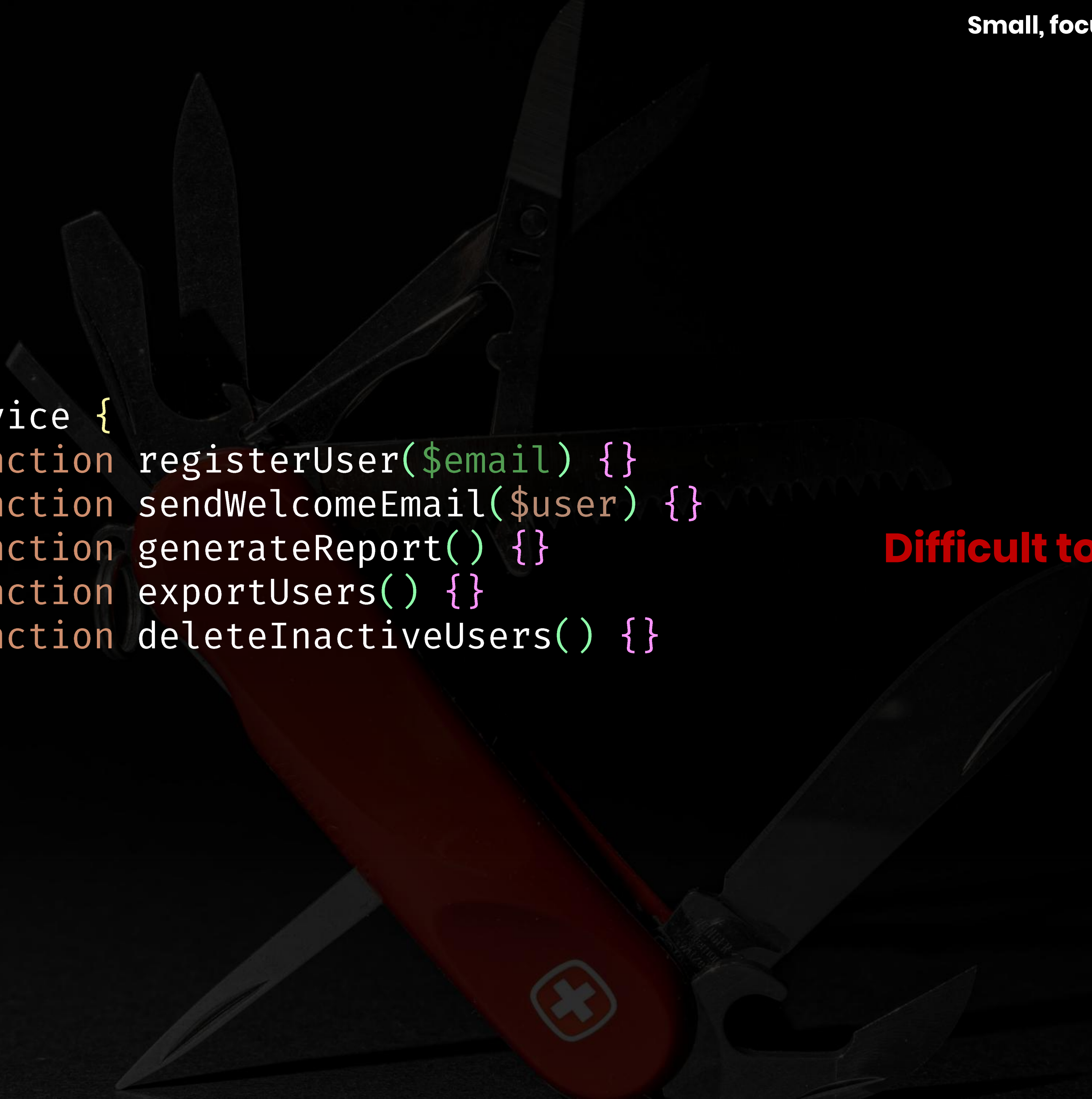
Does too many things



```
class UserService {
    public function registerUser($email) {}
    public function sendWelcomeEmail($user) {}
    public function generateReport() {}
    public function exportUsers() {}
    public function deleteInactiveUsers() {}
}
```

Hidden dependencies

Small, focused, and predictable components



```
class UserService {
    public function registerUser($email) {}
    public function sendWelcomeEmail($user) {}
    public function generateReport() {}
    public function exportUsers() {}
    public function deleteInactiveUsers() {}
}
```

Difficult to test

Small, focused, and predictable components

```
class UserService {
    public function registerUser($email) {}
    public function sendWelcomeEmail($user) {}
    public function generateReport() {}
    public function exportUsers() {}
    public function deleteInactiveUsers() {}
}
```

Grows endlessly

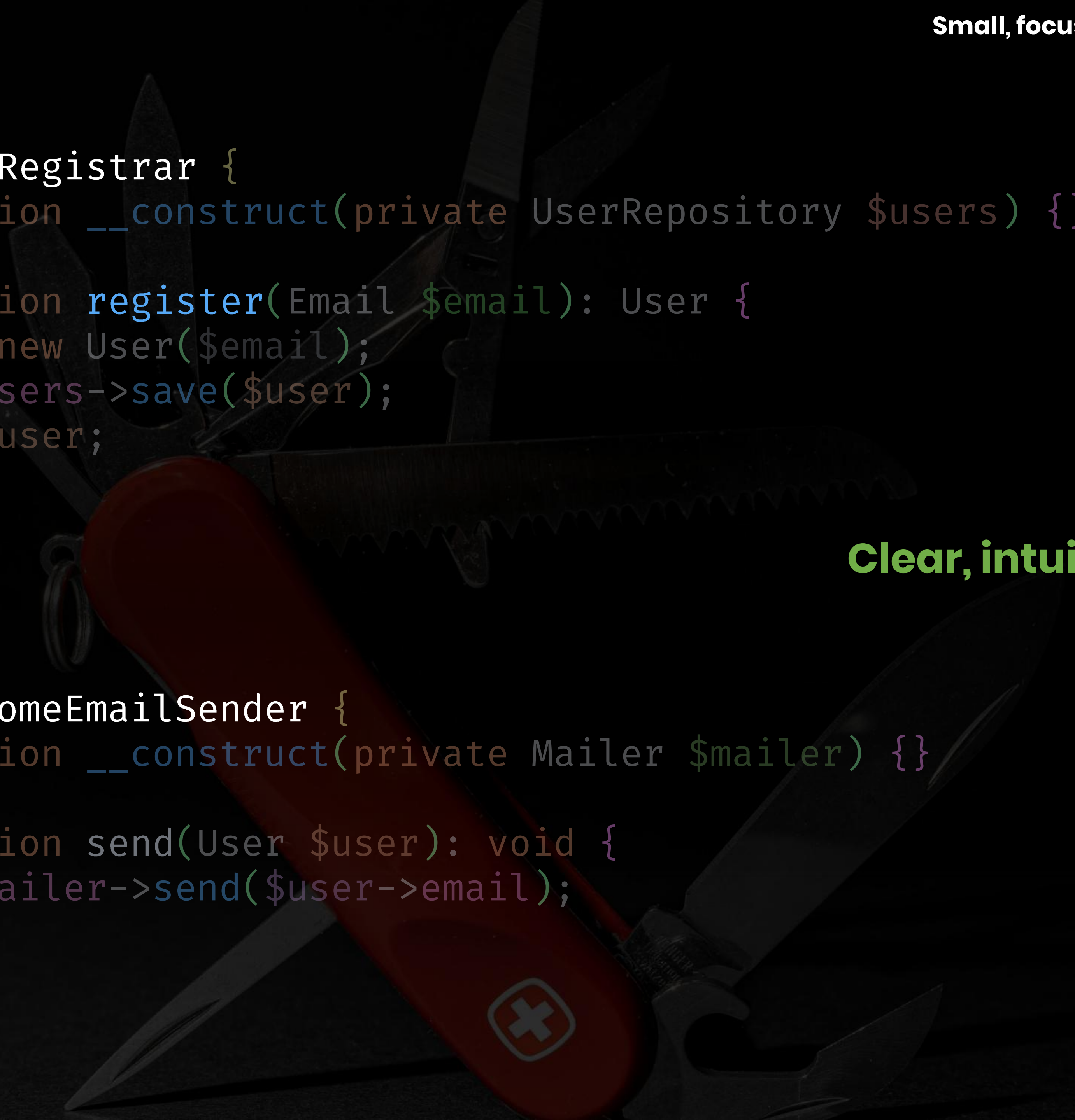
Small, focused, and predictable components

```php
final class UserRegistrar {
    public function __construct(private UserRepository $users) {}

    public function register(Email $email): User {
        $user = new User($email);
        $this->users->save($user);
        return $user;
    }
}



final class WelcomeEmailSender {
    public function __construct(private Mailer $mailer) {}

    public function send(User $user): void {
        $this->mailer->send($user->email);
    }
}
```

```
final class UserRegistrar {
    public function __construct(private UserRepository $users) {}

    public function register(Email $email): User {
        $user = new User($email);
        $this->users->save($user);
        return $user;
    }
}
```

Single Responsibility

```
final class WelcomeEmailSender {
    public function __construct(private Mailer $mailer) {}

    public function send(User $user): void {
        $this->mailer->send($user->email);
    }
}
```

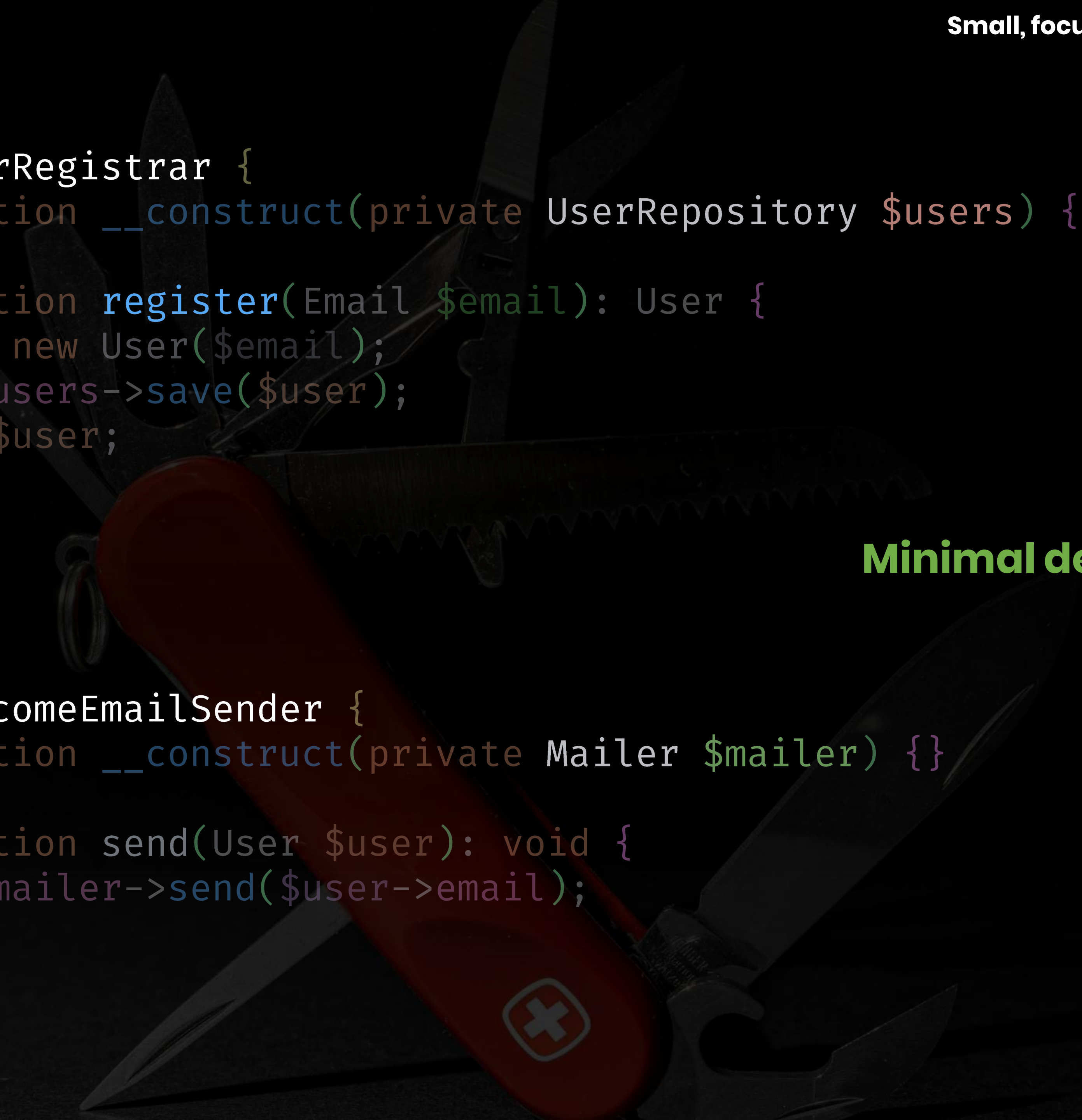


```
final class UserRegistrar {
    public function __construct(private UserRepository $users) {}

    public function register(Email $email): User {
        $user = new User($email);
        $this->users->save($user);
        return $user;
    }
}
```

Clear, intuitive behavior

```
final class WelcomeEmailSender {
    public function __construct(private Mailer $mailer) {}

    public function send(User $user): void {
        $this->mailer->send($user->email);
    }
}
```

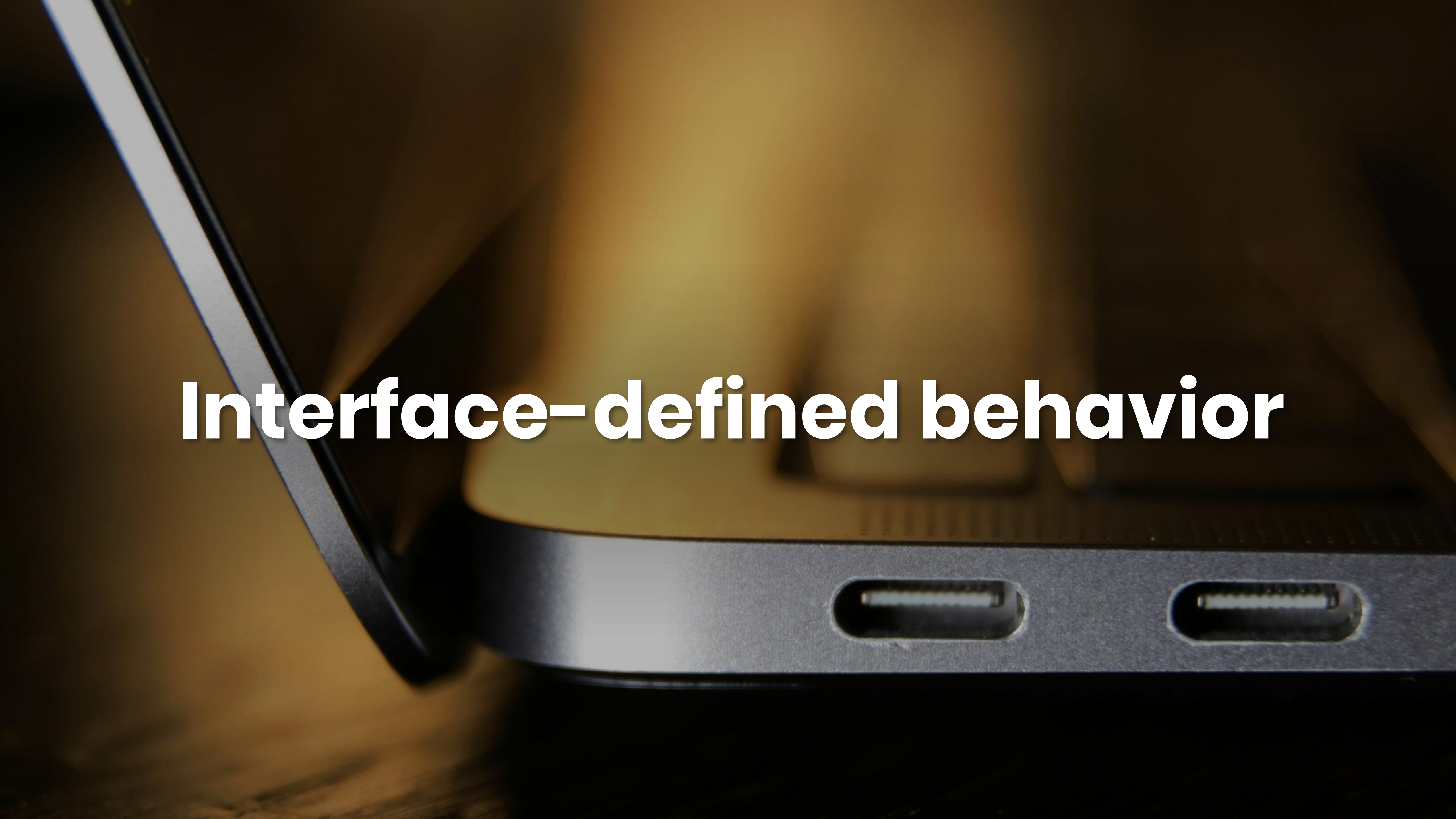
Small, focused, and predictable components



```php
final class UserRegistrar {
    public function __construct(private UserRepository $users) {}

    public function register(Email $email): User {
        $user = new User($email);
        $this->users->save($user);
        return $user;
    }
}
```

Minimal dependencies

```php
final class WelcomeEmailSender {
    public function __construct(private Mailer $mailer) {}

    public function send(User $user): void {
        $this->mailer->send($user->email);
    }
}
```



Interface-defined behavior

Interface-defined behavior

Interfaces define **what**, *not how*

Interface-defined behavior

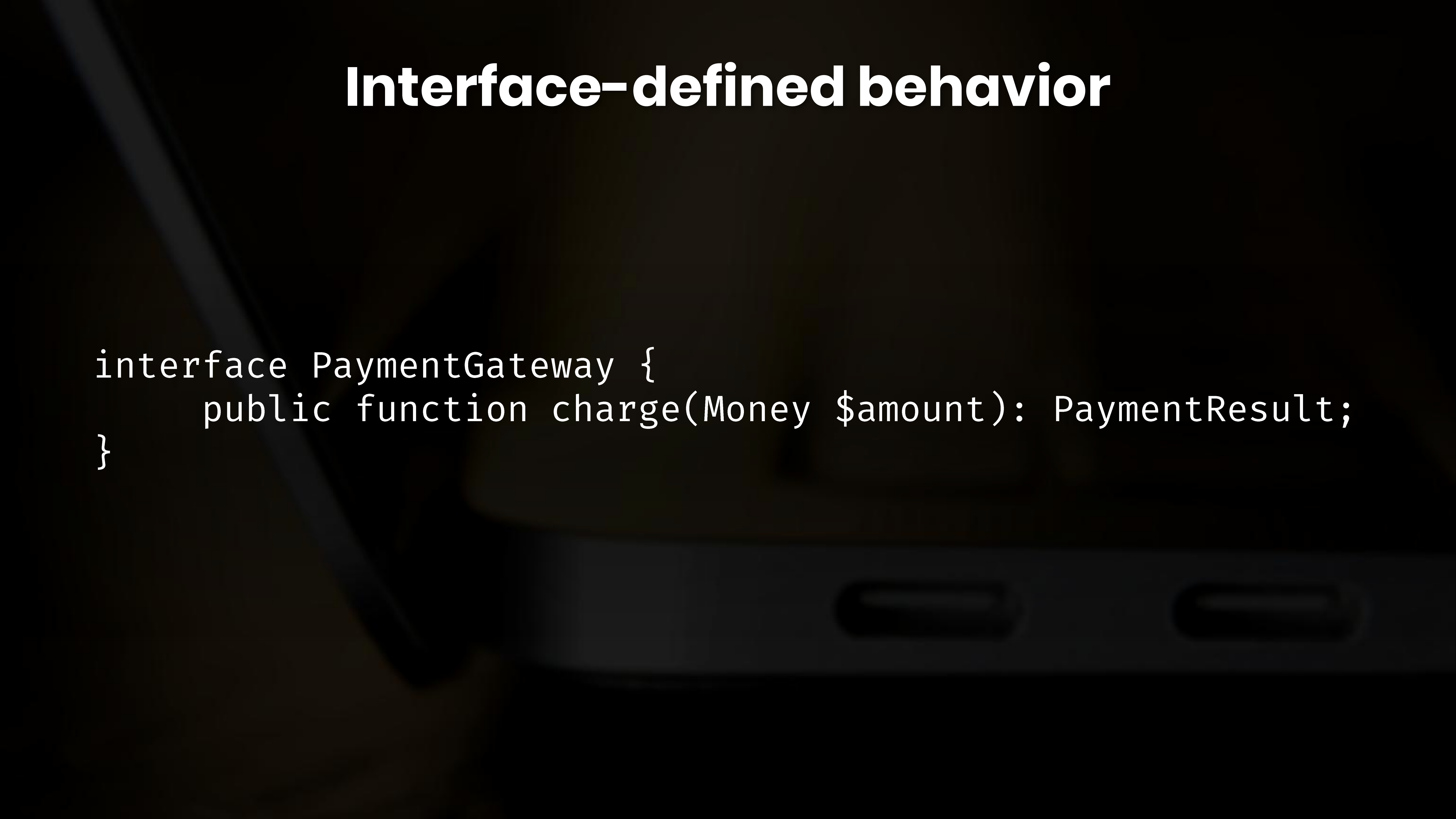
Interfaces define **what**, *not how*
Standards enable **interoperability**

Interface-defined behavior

Interfaces define **what**, *not how*

Standards enable **interoperability**

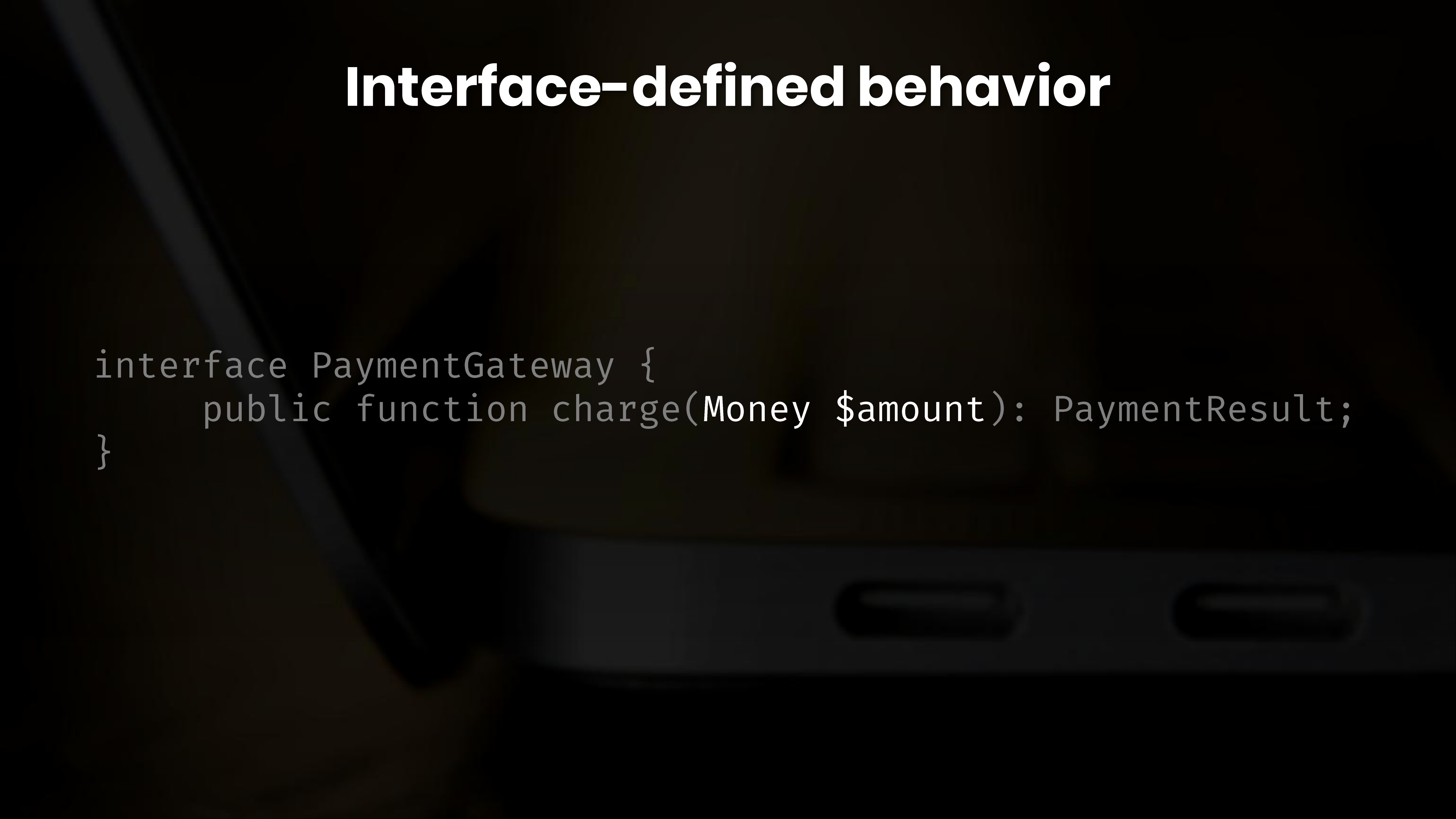
Expose **behaviors**, *not data structures*



Interface-defined behavior

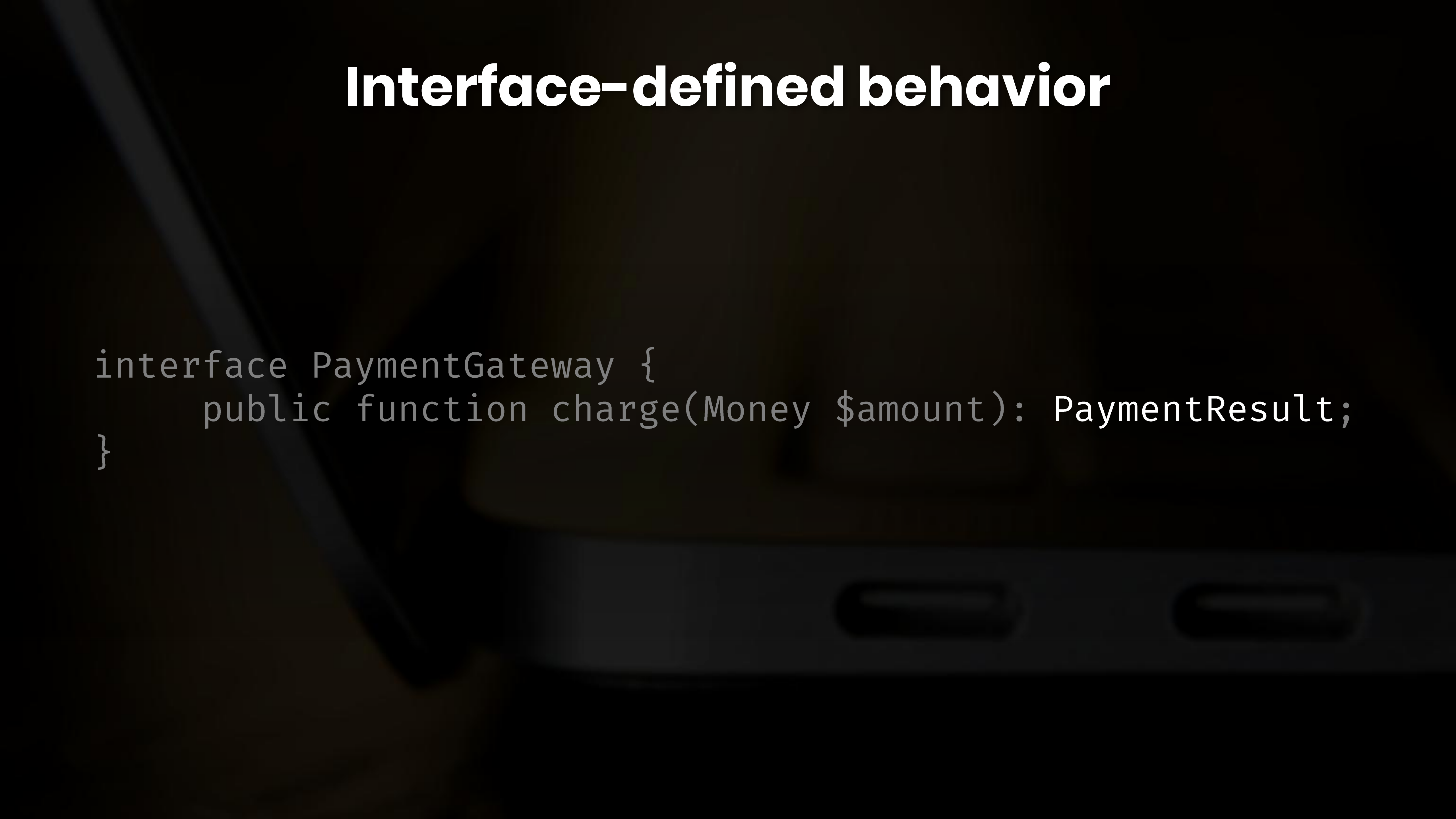
```
interface PaymentGateway {
     public function charge(Money $amount): PaymentResult;
}
```

Interface-defined behavior



```
interface PaymentGateway {
     public function charge(Money $amount): PaymentResult;
}
```

Interface-defined behavior



```
interface PaymentGateway {
    public function charge(Money $amount): PaymentResult;
}
```

Interface-defined behavior

```
interface PaymentGateway {
      public function charge(Money $amount): PaymentResult;
}


      class AdyenPaymentGateway implements PaymentGateway {
          public function __construct(
              private readonly string $apiKey,
              private readonly string $apiSecret
          ) {}

          public function charge(Money $amount): PaymentResult {
              // Adyen-specific API calls
          }
      }
```

Interface-defined behavior

```php
interface PaymentGateway {
      public function charge(Money $amount): PaymentResult;
}



      class AdyenPaymentGateway implements PaymentGateway {
          public function __construct(
              private readonly string $apiKey,
              private readonly string $apiSecret
          ) {}

          public function charge(Money $amount): PaymentResult {
              // Adyen-specific API calls
          }
      }
```

Interface-defined behavior

```
interface PaymentGateway {
        public function charge(Money $amount): PaymentResult;
}
```

```
        class AdyenPaymentGateway implements PaymentGateway {
            public function __construct(
                private readonly string $apiKey,
                private readonly string $apiSecret
            ) {}

            public function charge(Money $amount): PaymentResult {
                // Adyen-specific API calls
            }
        }
```

Interface-defined behavior

```
interface PaymentGateway {
        public function charge(Money $amount): PaymentResult;
}

class AdyenPaymentGateway implements PaymentGateway {

}
```

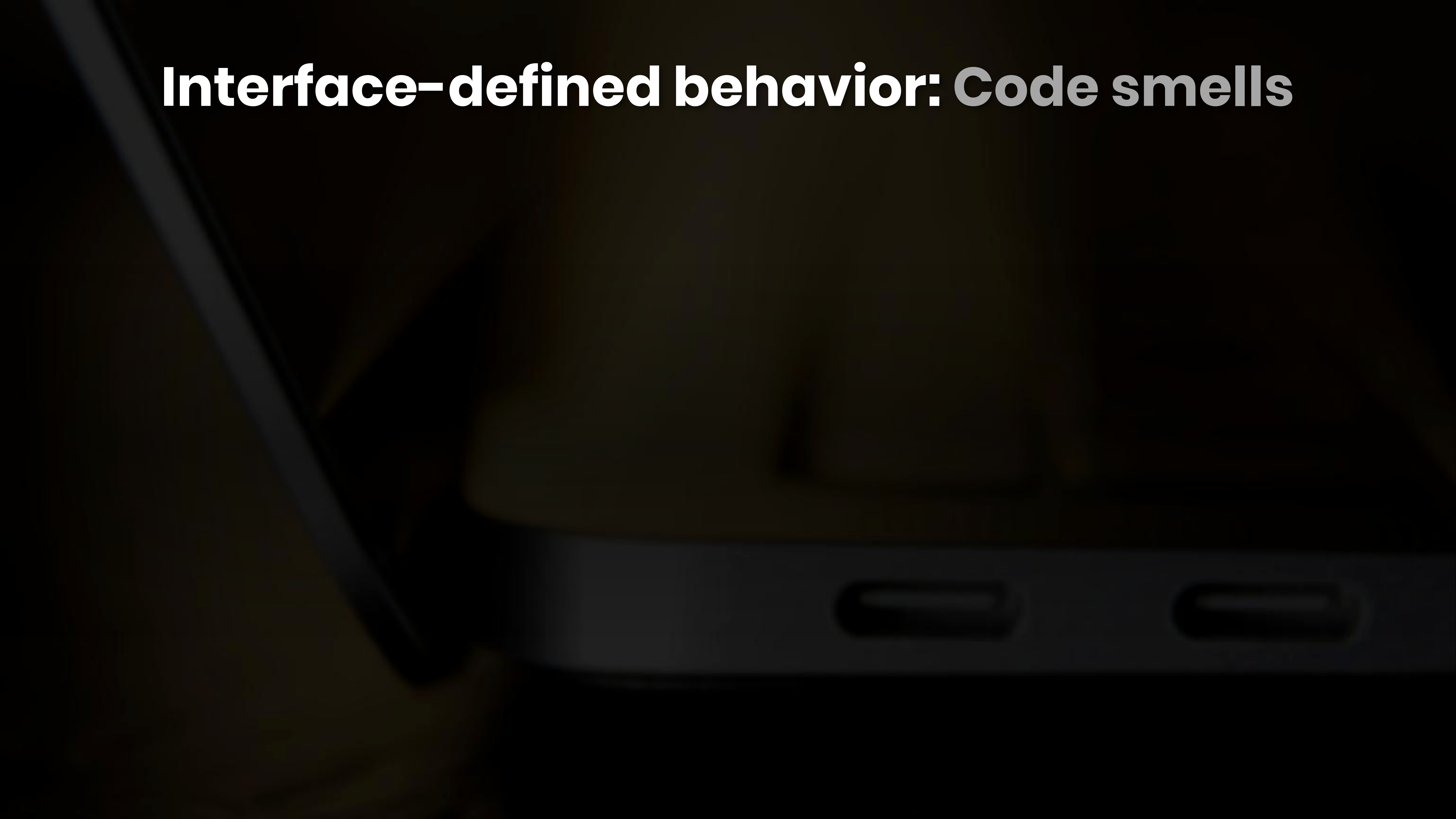
```
class ShoppingCart {
    public function __construct(
        private PaymentGateway $paymentGateway
    ) {}
}
```

Interface-defined behavior

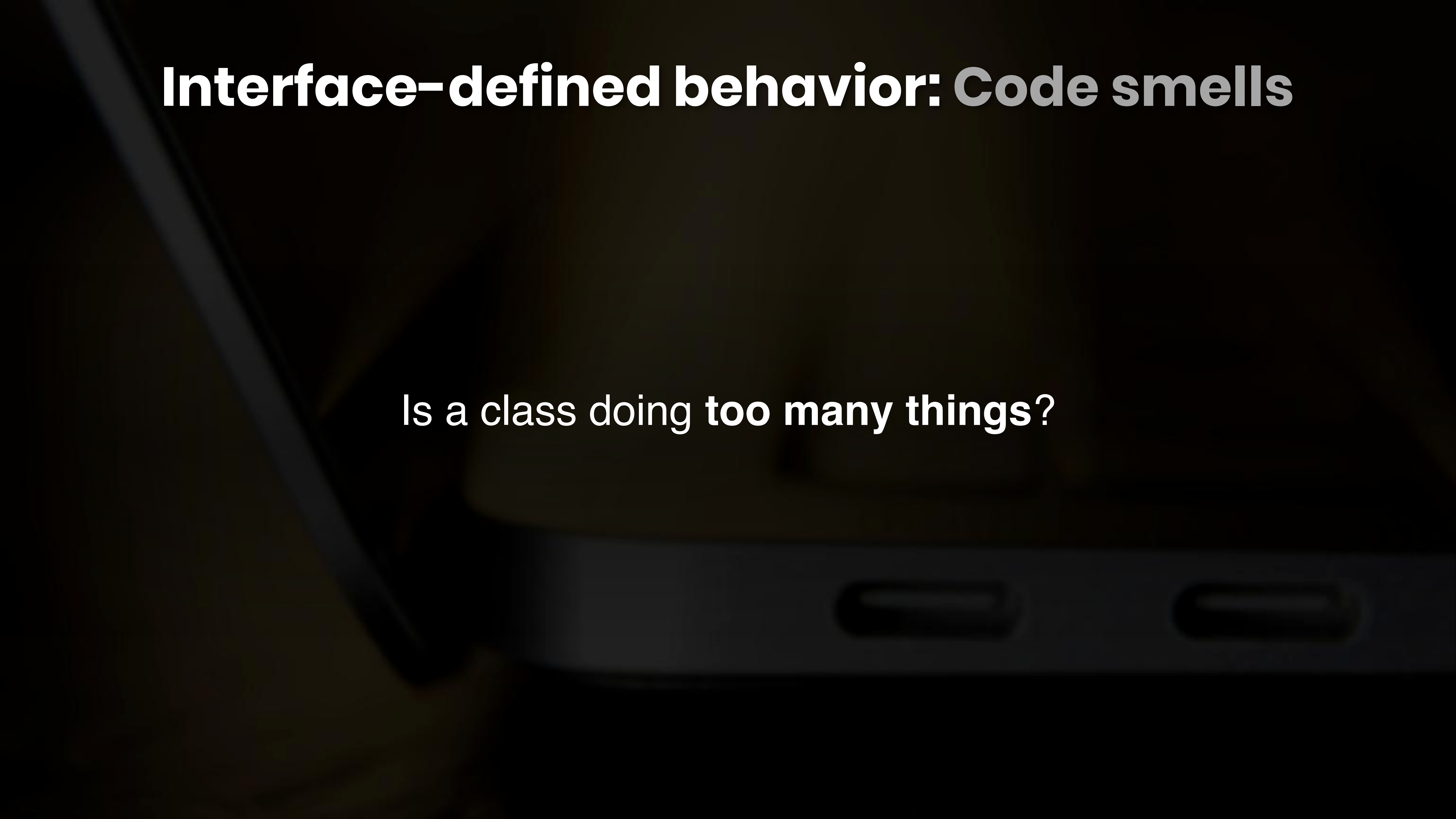
```
interface PaymentGateway {
        public function charge(Money $amount): PaymentResult;
}

class AdyenPaymentGateway implements PaymentGateway {

}
```

```
class ShoppingCart {
    public function __construct(
        private PaymentGateway $paymentGateway
    ) {}
}
```

Interface-defined behavior: Code smells

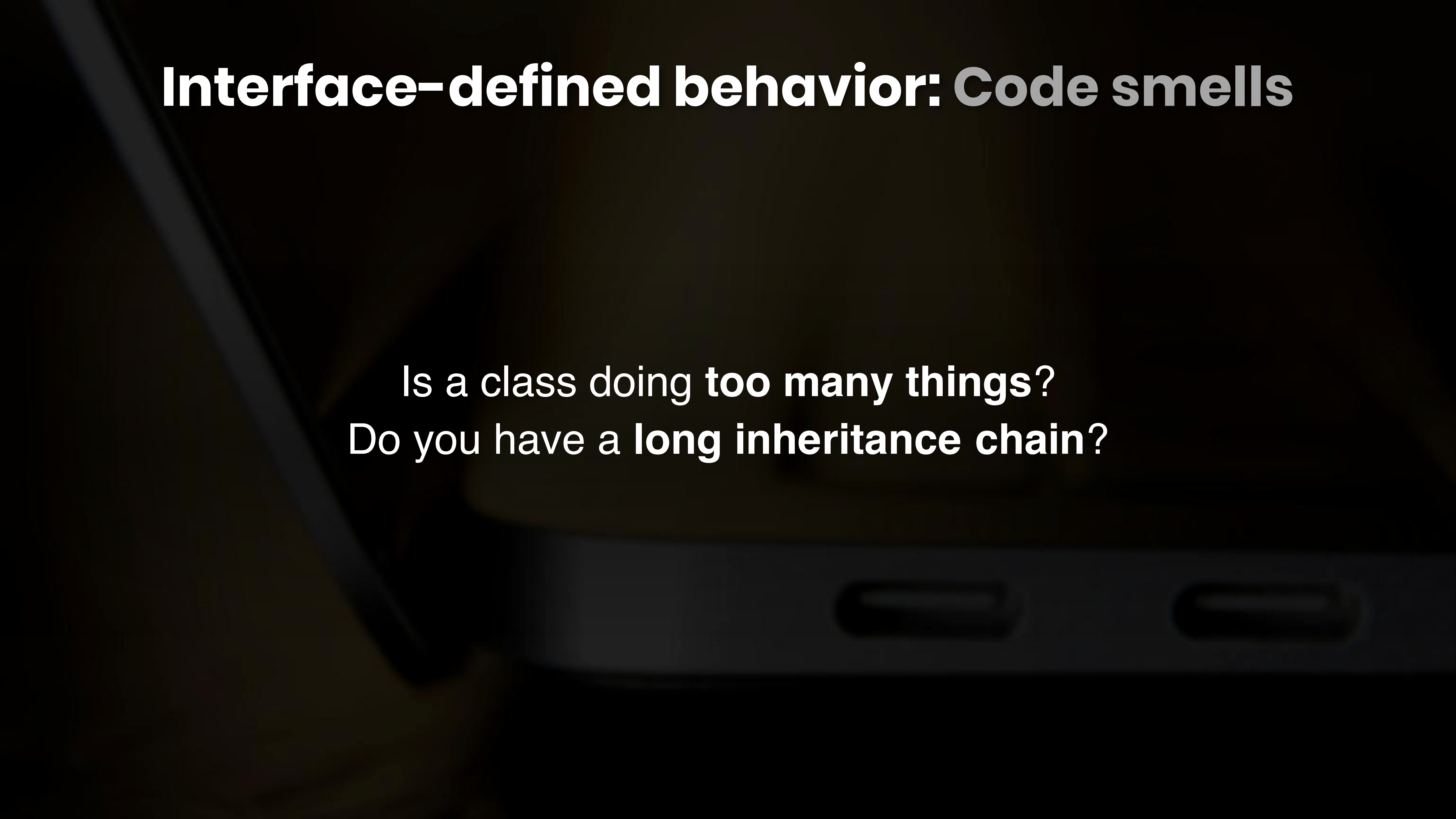


Interface-defined behavior: Code smells



Is a class doing **too many things**?

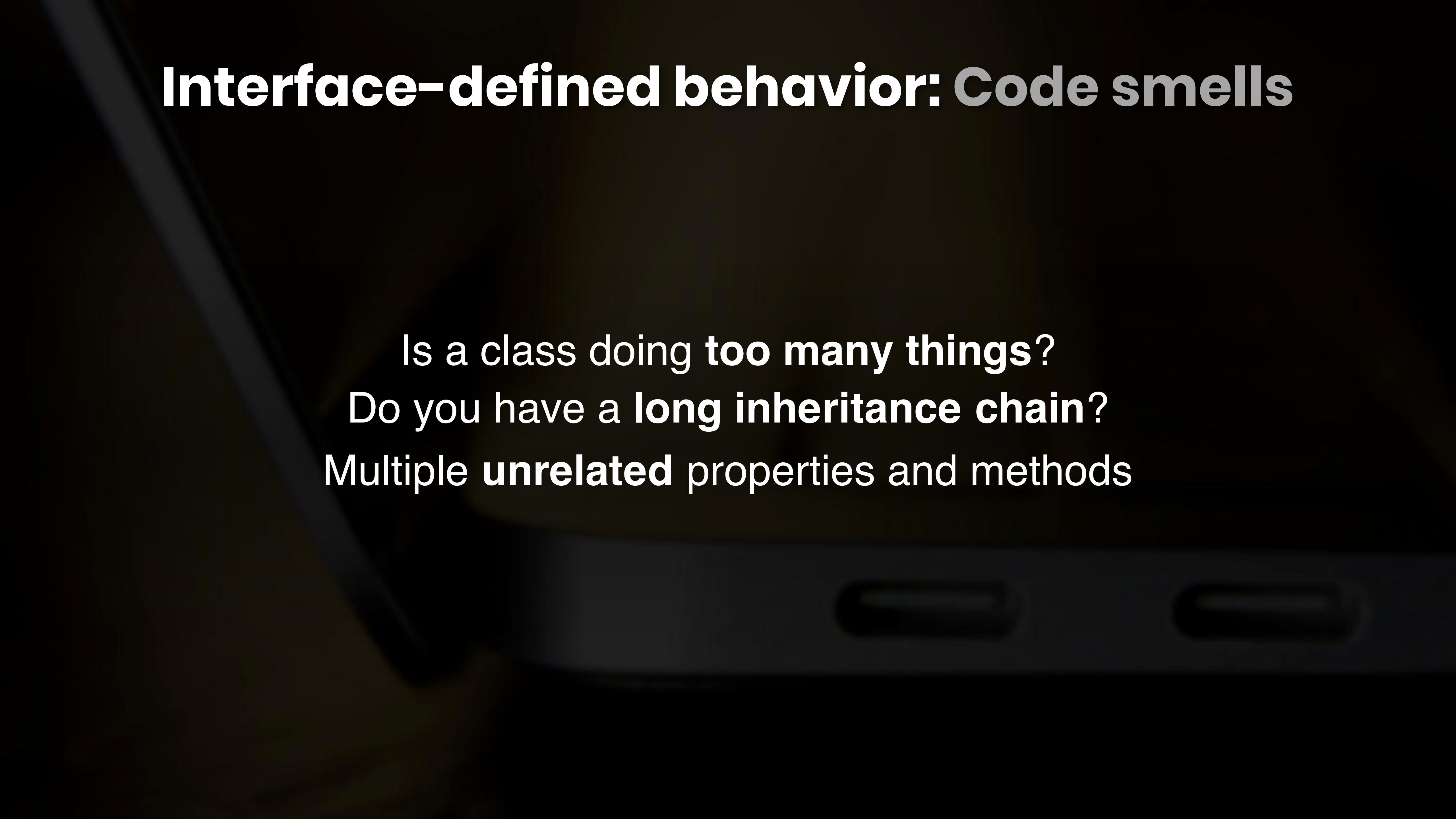
Interface-defined behavior: Code smells



Is a class doing **too many things**?

Do you have a **long inheritance chain**?

Interface-defined behavior: Code smells

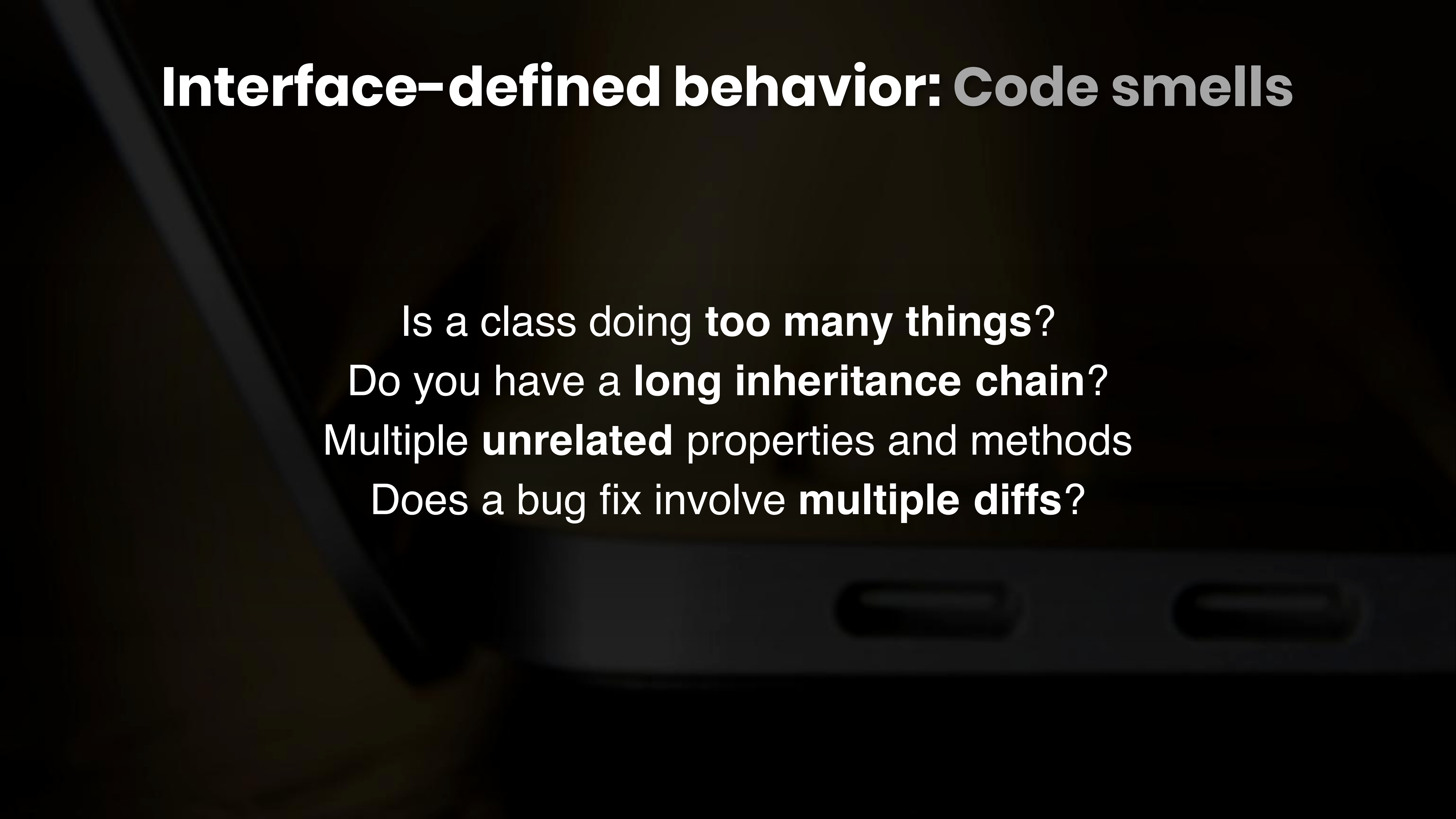


Is a class doing **too many things**?

Do you have a **long inheritance chain**?

Multiple **unrelated** properties and methods

Interface-defined behavior: Code smells

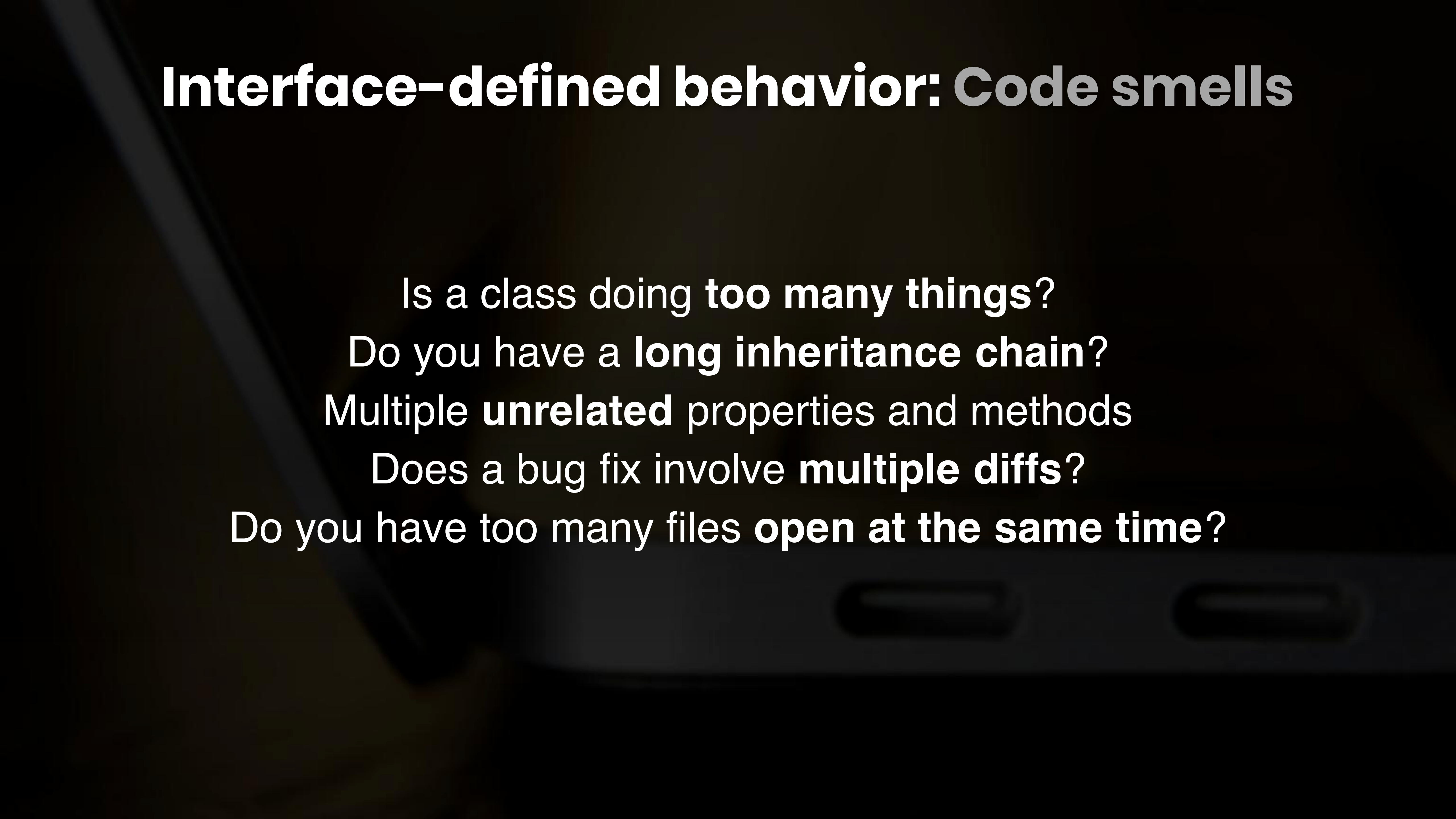


Is a class doing **too many things**?

Do you have a **long inheritance chain**?

Multiple **unrelated** properties and methods

Does a bug fix involve **multiple diffs**?



Interface-defined behavior: Code smells

Is a class doing **too many things**?

Do you have a **long inheritance chain**?

Multiple **unrelated** properties and methods

Does a bug fix involve **multiple diffs**?

Do you have too many files **open at the same time**?

Further Resources

- https://aye.sh/talk/dpc2026-defensive-php
- https://www.youtube.com/watch?v=8d2AtAGJPno | Extremely Defensive PHP – Marco Pivetta
- https://phpdelusions.net/articles/error_reporting
- https://getrector.com/
- https://phpstan.org/
- https://php.watch/versions

Questions?

No question is too smol

#DPC26 ayesh@php.watch

https://aye.sh/talk/dpc2026-defensive-php

Ďakujem dziękuję tak

arigatô paldies

mahalo kop khun

diolch dankie děkuji

cảm ơn bạn хвала شكرا لك köszönöm

tänan

a dank gràcies ngiyabonga Баярлалаа dhanyavād

Дякую Благодарам

ευχαριστώ

THANK YOU

спасибо благодаря

tack

grazie M̀h’gōi Dank u Благодаря ти gracias

multumesc 多謝 ස්තුතියි ačiū நன்றி תודה.

takk

danke

faleminderit 谢谢

teşekkür ederim choukrane

obrigado kiitos

Շնորհակալություն

terima kasih hvala grazzi

arigatô paldies dziękuję Ďakujem tak

děkuji mahalo

kop khun

diolch dankie хвала Баярлалаа شكرا لك

tänan köszönöm

cảm ơn bạn ngiyabonga

a dank gràcies dhanyavād

THANK YOU

Дякую ευχαριστώ Благодарам

спасибо

Dank u wel

tack благодаря

grazie M̀h’gōi

Dank u Благодаря ти gracias

mulţumesc 多謝 ačiū

danke takk ස්තුතියි நன்றி תודה.

faleminderit

teşekkür ederim choukrane

obrigado kiitos 谢谢

Շնորհակալություն terima kasih hvala grazzi arigatô paldies děkuji mahalo شكرا لك Ďakujem tak

dankie dziękuję Баярлалаа kop khun

diolch хвала tänan tack köszönöm

cảm ơn bạn ngiyabonga

dhanyavād

a dank gràcies

THANK YOU

Благодарам

Дякую ευχαριστώ

спасибо благодаря

Dank u wel

grazie Ṁh’gōi 多謝 ačiū תודה.

Defensive PHP: Code that refuses to fail silently

multumesc Dank u Благодаря ти

danke takk ස්තූතියි faleminderit gracias

teşekkür ederim choukrane obrigado நன்றி 谢谢

Շնորհակալություն terima kasih hvala kiitos grazzi

THANK YOU

Sabrina Schipper from **Travel Counsellors**
for arranging me flights *four* times
when pretty much all the flights got cancelled/

I wouldn't have made it without the miracles you pulled.

travel counsellors

Ayesh Karunaratne | https://aye.sh/talk/dpc2026-defensive-php

DEFENSIVE PHP

Code that refuses to fail silently

Defensive PHP: Code that refuses to fail silently