

Rediscovering PHP

Modern Practices Beyond Legacy

Horacio Gonzalez

2026-02-12



@LostInBrittany



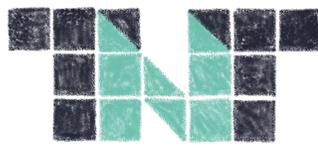


Who are we?

Introducing myself and
introducing Clever Cloud



Horacio Gonzalez



@LostInBrittany

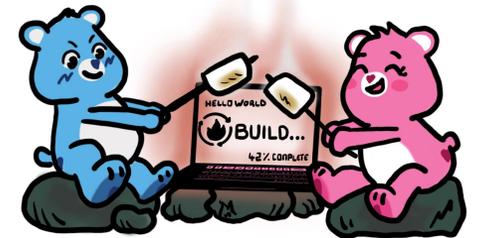
Spaniard Lost in Brittany

Old(ish) Developer

Head of DevRel



clever cloud





Clever Cloud

From Code to Product



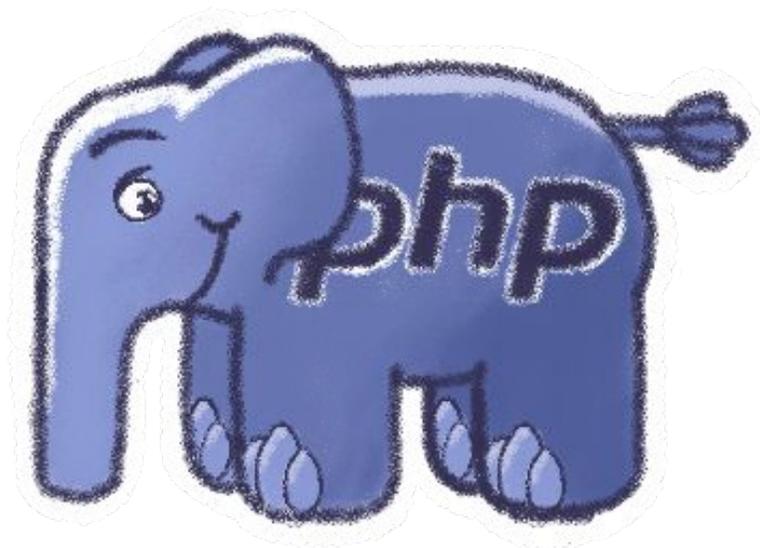
clever cloud





You're going to talk about PHP ?!?!

The elephant IS in the room

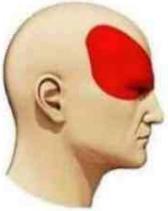


So yeah, I know both PHP and its reputation...

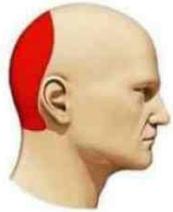


Types of Headache

Migraine



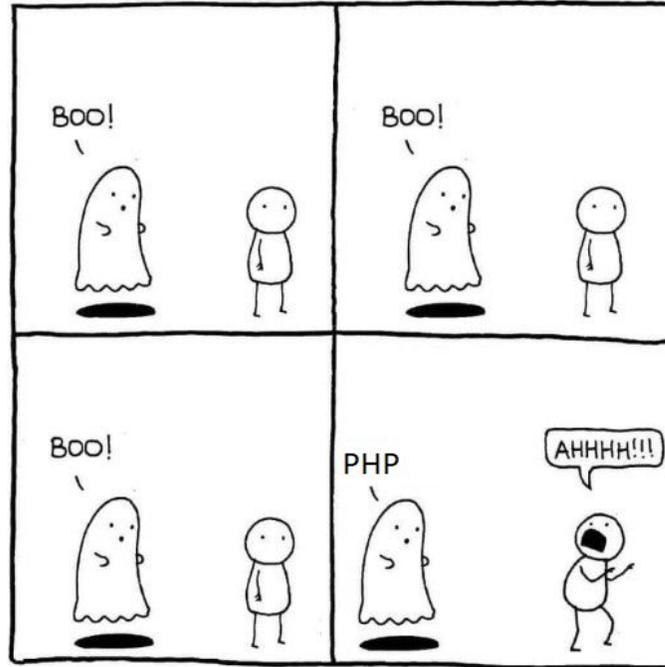
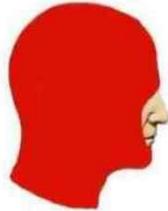
Hypertension



Stress



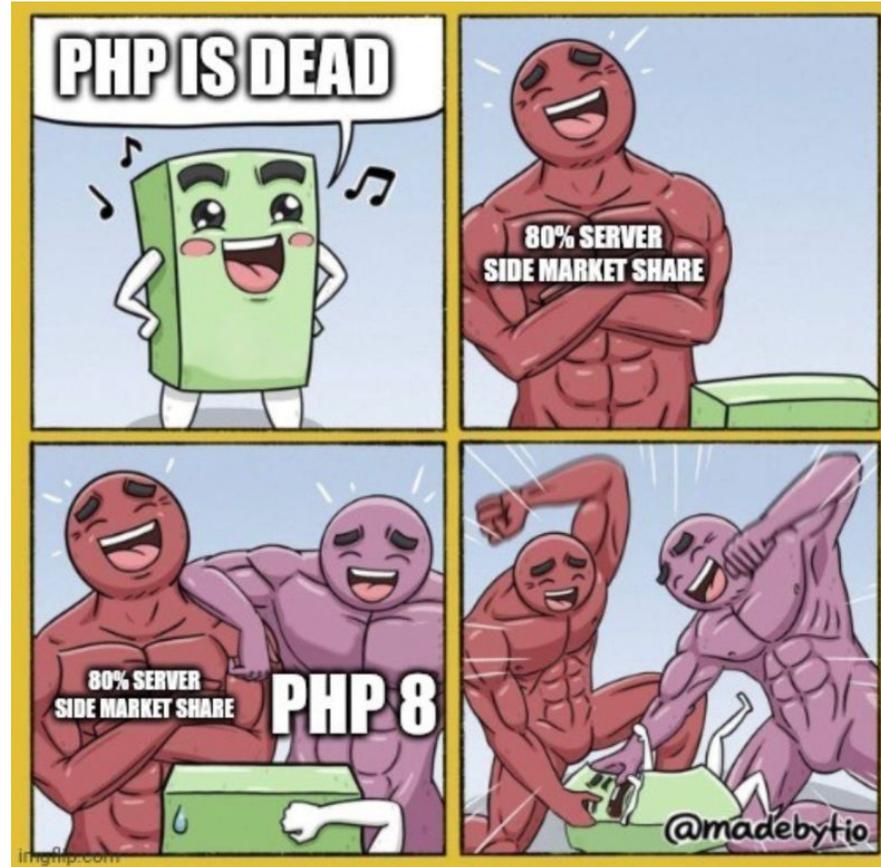
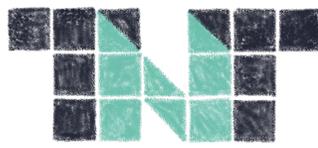
PHP



And in 2004 it was quite accurate, PHP felt like the Wild West



If you think PHP is inconsistent, insecure, and strictly procedural...



You were right 15 years ago !

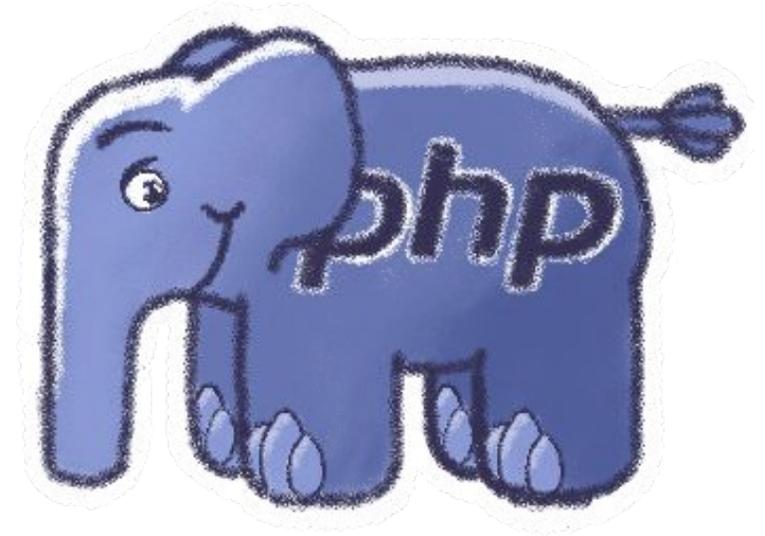


I've been a Java developer since '97



And I have done tons of frontend in HTML/CSS/JS

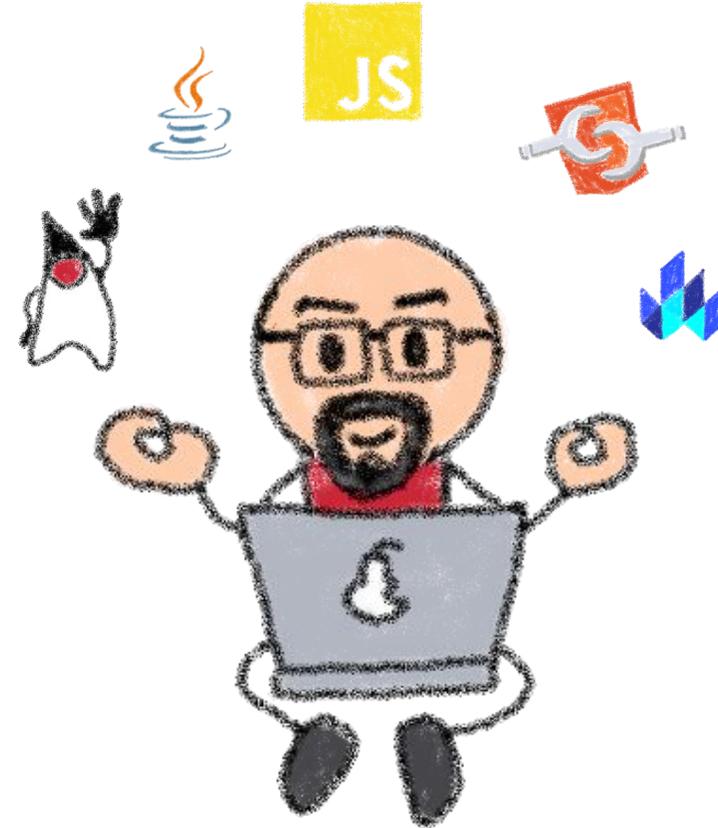
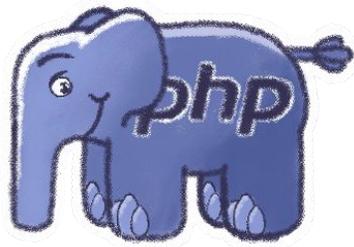
But I've also written and debugged lots of PHP



I even made a living of it!



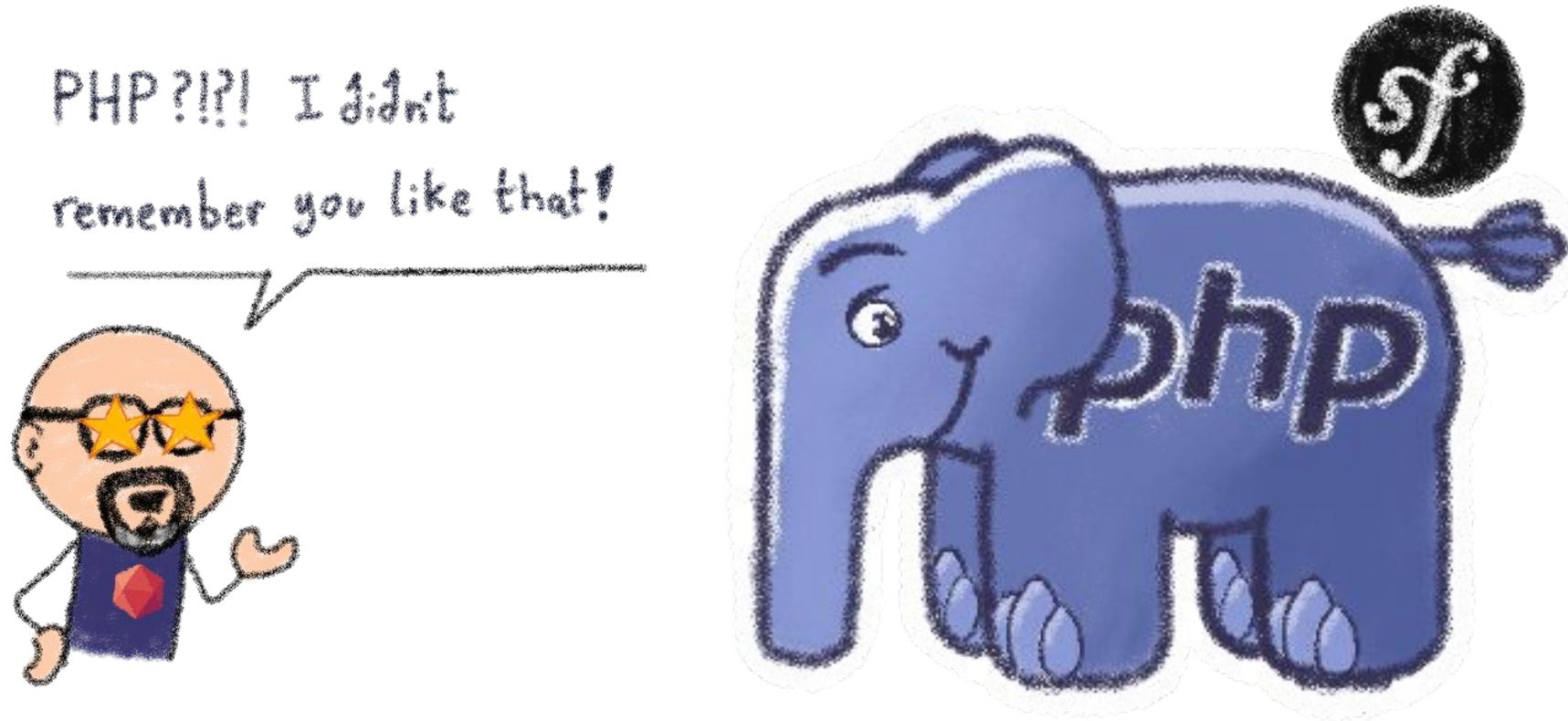
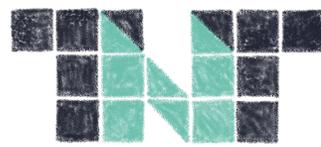
For years I kept away from PHP



I reckon I was a bit snobbish about it...



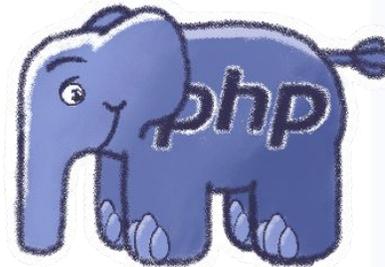
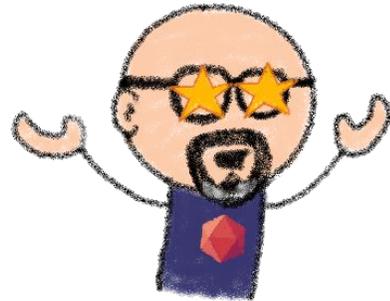
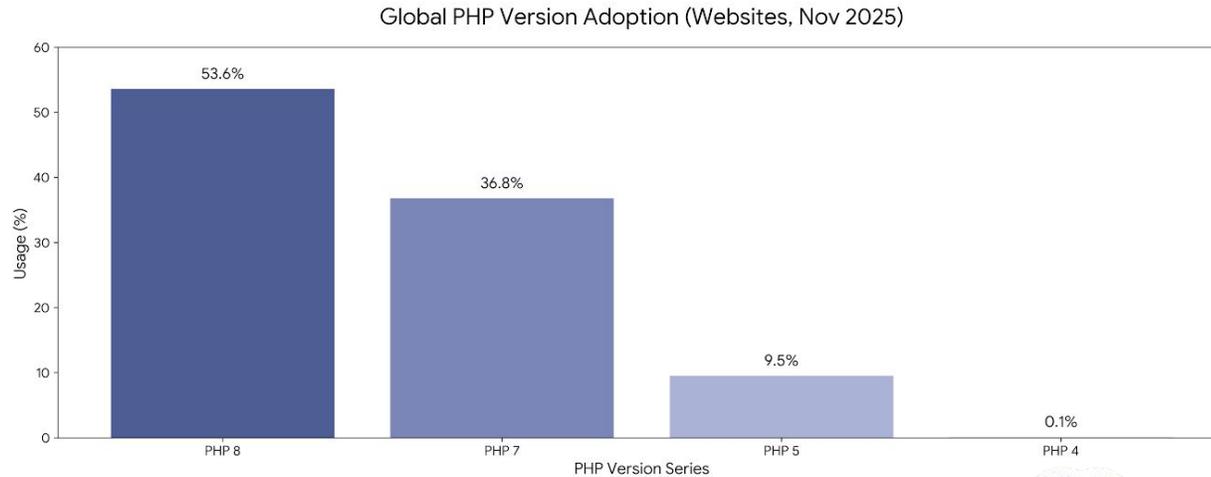
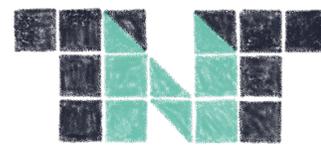
Until I joined Clever Cloud and rediscovered it



And, mate, it was so much better!



PHP is not like PHP 4 anymore



Version	Release Date	Key Note
PHP 3.0	June 6, 1998	The version that established the "PHP" name.
PHP 4.0	May 22, 2000	Introduced the Zend Engine.
PHP 5.0	July 13, 2004	Introduced robust Object-Oriented Programming (OOP).
PHP 5.3	June 30, 2009	Added Namespaces and Closures.
PHP 5.6	Aug 28, 2014	Last version of the PHP 5 series.
PHP 7.0	Dec 3, 2015	Major performance boost (2x speed); PHP 6 was skipped.
PHP 7.4	Nov 28, 2019	Introduced Typed Properties; last of the 7.x series.
PHP 8.0	Nov 26, 2020	Introduced JIT Compiler and Union Types.
PHP 8.1	Nov 25, 2021	Enums and Readonly properties.
PHP 8.2	Dec 8, 2022	Readonly classes.
PHP 8.3	Nov 23, 2023	Typed class constants.
PHP 8.4	Nov 21, 2024	Property hooks and asymmetric visibility.

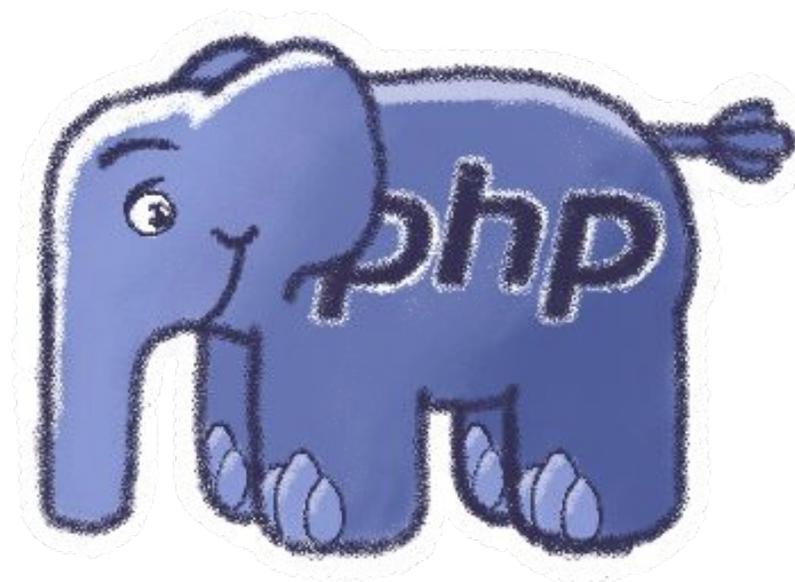
PHP powers 75%+ of the web...
and 90% of that usage looks nothing like WordPress 3.0



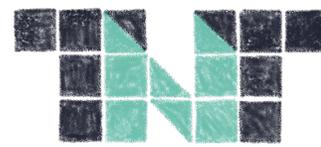


PHP - The Language

Syntax for the Java Mind



Code Organization: Namespaces & Autoloading



```
The Good Old Days
```

```
include 'config.php';
include 'database.php';
include 'functions.php';
include 'models/user.php';
include 'models/admin.php';
include 'controllers/user_controller.php';
include 'lib/email.php';
include 'lib/validator.php';
// ... 50 more includes ...// Everything is global
$user = new User();
$admin = new Admin();
// Name collision disasters waiting to happen:
function send_email() { } // From email.php
function send_email() { } // Oops, redeclared in another file!
```

```
The Modern Way
```

```
namespace App\Http\Controllers;
use App\Services\UserService;
use App\Models\User;
use App\Mail>WelcomeEmail;

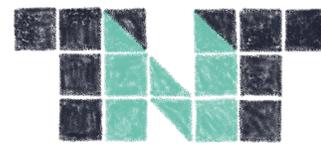
class UserController
{
    public function __construct(
        private UserService $userService
    ) {}

    ...
}
```

PSR-4 Autoloading: clean namespace hierarchy



Typing



The Good Old Days



```
function createUser($name, $age, $active) {
    // Is $age an int? "20"? 20.5?
    // Is $active a boolean? 0? 1? "yes"?

    if ($active == 1) {
        return "User: " . $name . " is " . $age;
    }

    // Returns null implicitly if condition fails
    // Inconsistent return types!
}

$u = createUser("Horacio", "45", "true");
// Works, but relies on "Magic" casting.
```



The Modern Way

```
declare(strict_types=1); // <--- The "Java Switch"
    // Contract-based programming. If you violate the contract,
    // the application stops immediately. No guessing.

final class UserFactory
{
    // Typed Arguments, Return Type, Visibility
    public function create(
        string $name, int $age, bool $isActive
    ): string {
        if ($isActive) {
            return sprintf("User: %s is %d", $name, $age);
        }

        throw new InactiveUserException();
    }
}
```

From "Anything Goes" to Strictness



The Type System Flex (Union & Intersection Types)



The Modern Way

```
public function handle(User|Guest $entity): Response
{
    // Type system knows it's one of these
    // IDE autocomplete works
    // Static analysis validates it

    return match(true) {
        $entity instanceof User => $this->handleUser($entity),
        $entity instanceof Guest => $this->handleGuest($entity),
    };
}

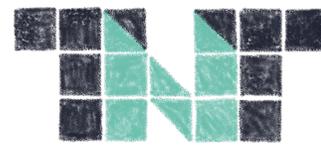
// Return type unions: much cleaner than Optional
public function findUser(int $id): User|null
{
    return $this->repository->find($id);
}

// Must satisfy BOTH interfaces
public function log(Loggable&Serializable $event): void
```

No need for overloading boilerplate.



Constructor Property Promotion (PHP 8.0+)



The Good Old Days



```
class User
{
    private string $name;
    private string $email;
    private int $age;

    public function __construct(string $name, string $email, int $age)
    {
        $this->name = $name;
        $this->email = $email;
        $this->age = $age;
    }

    public function getName(): string { return $this->name; }
    public function getEmail(): string { return $this->email; }
    public function getAge(): int { return $this->age; }
}
```



The Modern Way

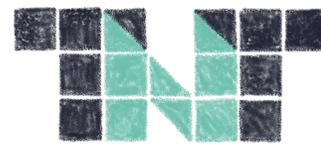
```
class UserProfile
{
    public function __construct(
        // Visibility + Type + Name = Property Created & Assigned
        private string $name,
        private string $email,
        private int $age,
    ) {}

    // That's it. Properties are declared, assigned, and accessible.
    // Getters are optional - you decide based on your encapsulation needs.
}

// Want to make it immutable like a Java Record?
// Just add 'readonly' (PHP 8.2+):
class ImmutableUser
{
    public function __construct(
        public readonly string $name,
        public readonly string $email,
    ) {}
}
```



Attributes: the End of "Code in Comments"



```
The Good Old Days

class UserController
{
    /**
     * @Route("/api/users", methods={"GET"})
     * @IsGranted("ROLE_ADMIN")
     *
     * This is technically a comment.
     * If I typo "@Rout", nothing crashes...
     * until the app runs and the route is missing.
     */
    public function list()
    {
        // ...
    }
}
```

```
The Modern Way

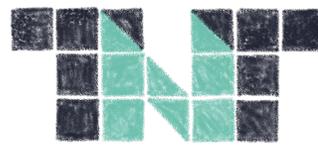
use Symfony\Component\Routing\Attribute\Route;
use Symfony\Component\Security\Http\Attribute\IsGranted;

class UserController
{
    // Native syntax. "Route" is a real class found via "use"
    #[Route('/api/users', methods: ['GET'])]
    #[IsGranted('ROLE_ADMIN')]
    public function list(): Response
    {
        // ...
    }
}

// How to make your own Attribute? It's just a class!
#[Attribute]
class MyCustomMetadata
{
    public function __construct(public string $info) {}
}
```



Enums (The Real Deal)



The Good Old Days



```
class BlogPost
{
    // These are just strings.
    // Nothing prevents me from passing "garbage" to a function expecting a
status.
    const STATUS_DRAFT = 'draft';
    const STATUS_PUBLISHED = 'published';
    const STATUS_ARCHIVED = 'archived';

    public function setStatus(string $status) {
        // I have to manually validate if $status is one of the allowed
constants
        if (!in_array($status, [self::STATUS_DRAFT, ...])) {
            throw new Exception();
        }
        $this->status = $status;
    }
}
```

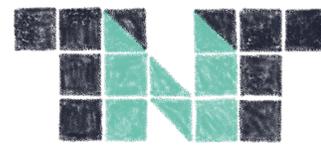


The Modern Way

```
/ 1. Defined using 'enum' keyword
enum Status: string
{
    case Draft = 'draft';
    case Published = 'published';
    case Archived = 'archived';
    // 2. THEY CAN HAVE METHODS! (Just like Java)
    public function color(): string
    {
        return match($this) {
            self::Draft => 'gray',
            self::Published => 'green',
            self::Archived => 'red',
        };
    }
}
// 3. Type Safety is absolute
function updateStatus(Status $newStatus): void {
    // Impossible to pass "foo" or "draft" string here.
}
// 4. Accessing values
echo Status::Published->value; // "published"
echo Status::Published->color(); // "green"
```



Match Expressions



The Good Old Days



```
// The problem:
// 1. Loose comparison ('200' string matches 200 int)
// 2. Requires 'break' (easy to forget = fallthrough bugs)
// 3. Verbose assignment logic

$status = 200;
$message = null;

switch ($status) {
    case 200:
        $message = 'OK';
        break;
    case 300:
    case 301:
        $message = 'Redirect';
        break;
    case 404:
        $message = 'Not Found';
        break;
    default:
        $message = 'Unknown';
}
```



The Modern Way

```
$status = 200;

// 1. Assign result directly to variable
// 2. Strict comparison ('200' string will NOT match 200 int)
// 3. Comma-separated values for multiple matches

$message = match ($status) {
    200 => 'OK',
    300, 301 => 'Redirect',
    404 => 'Not Found',
    default => 'Unknown',
};

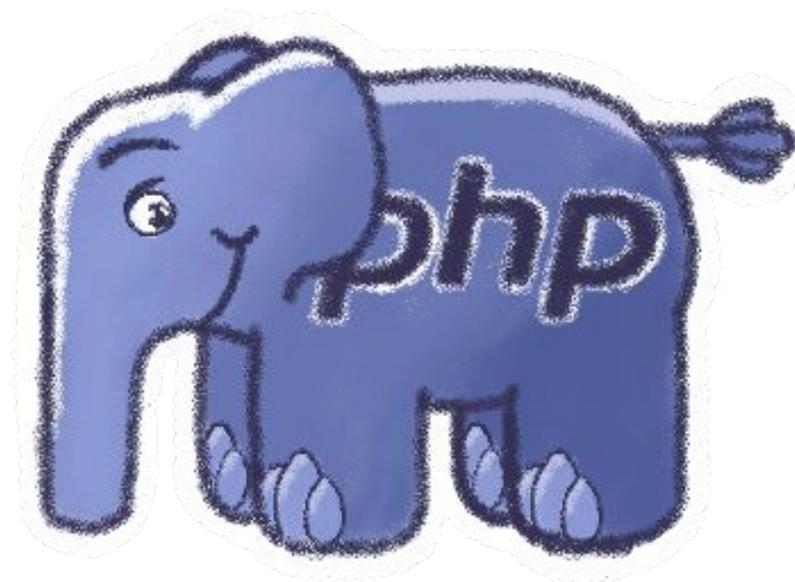
// Bonus: Pattern matching in PHP 8 allows logic inside match!
/*
$result = match (true) {
    $age >= 18 => 'Adult',
    $age < 18 => 'Minor',
};
*/
```



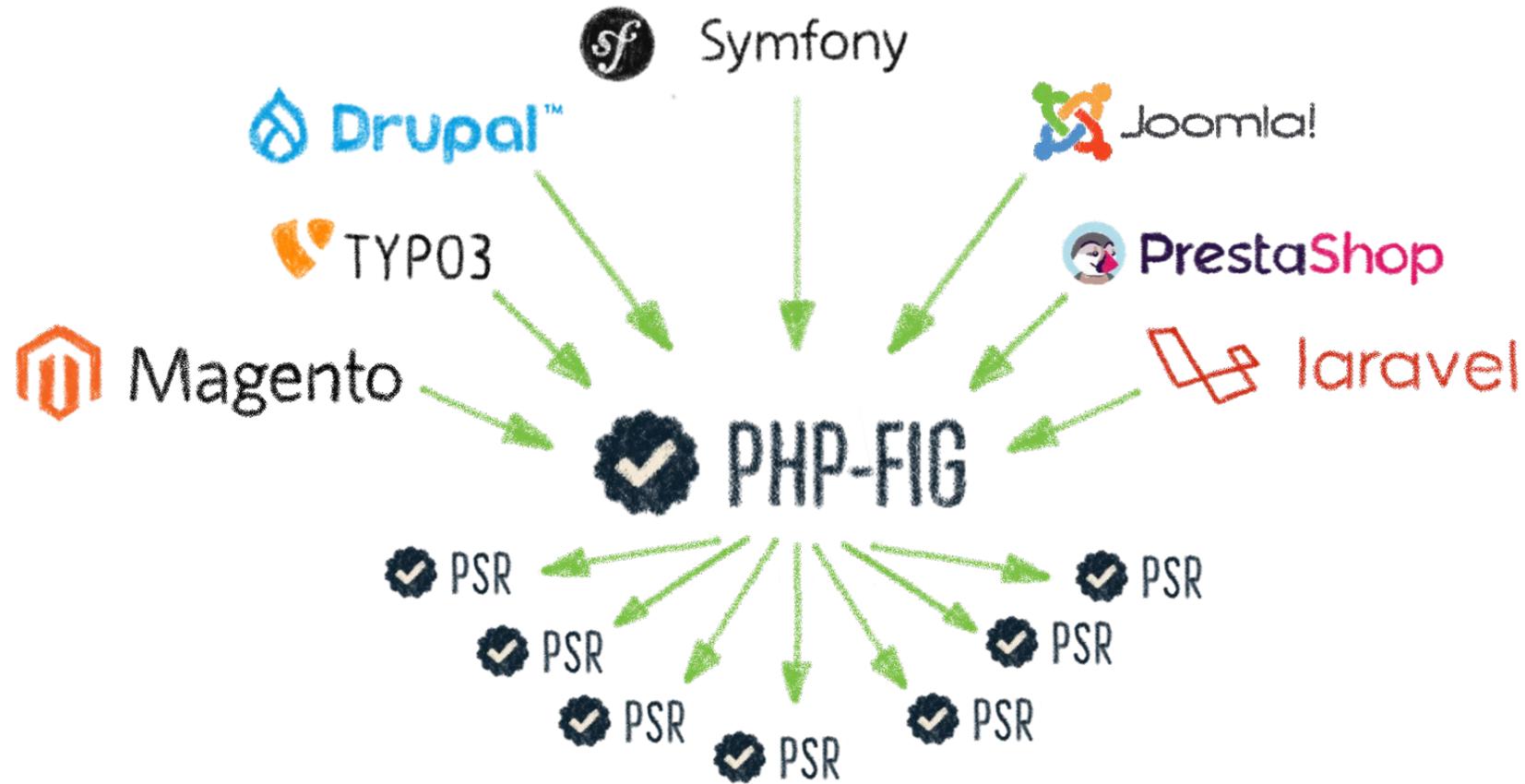
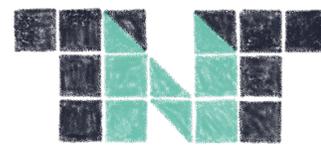


PHP - The Ecosystem

Professionalism & Tooling



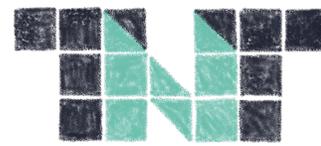
The PSR Standards (PHP-FIG)



PHP-FIG: Agree on Interfaces, not Implementations
PSRs (PHP Standard Recommendations).



PSR: Dependency Inversion



```
The Good Old Days

// Tightly coupled to "Monolog"
use Monolog\Logger;

class UserManager
{
    public function __construct(
        private Logger $logger // <--- HARD DEPENDENCY
    ) {}

    public function create() {
        // Tied to Monolog's specific method names
        $this->logger->addInfo('User created');
    }
}
```

```
// Decoupled. Relies on PSR-3 Standard.
use Psr\Log\LoggerInterface;

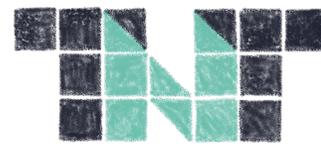
class UserManager
{
    public function __construct(
        private LoggerInterface $logger // <--- INTERFACE ONLY
    ) {}

    public function create() {
        // Guaranteed to exist on ANY PSR-3 compliant logger
        $this->logger->info('User created');
    }
}
```

True Dependency Inversion at a community scale



Dependency Management: Composer



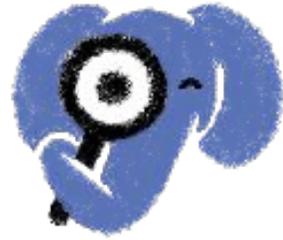
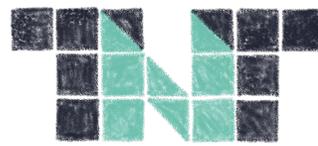
```
Java - Maven pom.xml
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
  <version>2.0.7</version>
</dependency>
<!-- And then refresh your IDE... -->
```

```
PHP - Composer composer.json
{
  "require": {
    "monolog/monolog": "^3.0"
  }
}
// Terminal command: composer require monolog/monolog
```

Dependency Management: Solved since 2012



PHPStan: The Compiler Substitute



PHPStan

```
PHP

class UserManager
{
    /**
     * @param array<User> $users <-- PHPStan reads this "Generic" definition
     */
    public function activateAll(array $users): void
    {
        foreach ($users as $user) {
            // Runtime: If $user is NOT a User object, this crashes HERE.
            $user->activate();
        }
    }
}

// The Bug:
$managers = [new User(), new User(), "oops, a string"];
(new UserManager())->activateAll($managers);
```

```
$ vendor/bin/phpstan analyse src --level=9

-----
1/1 [████████████████████████████████████████] 100%

-----

Line   src/UserManager.php
-----

18     Parameter #1 $users of method UserManager::activateAll()
       expects array<User>, array{User, User, string} given.
       Type string is not subtype of User.

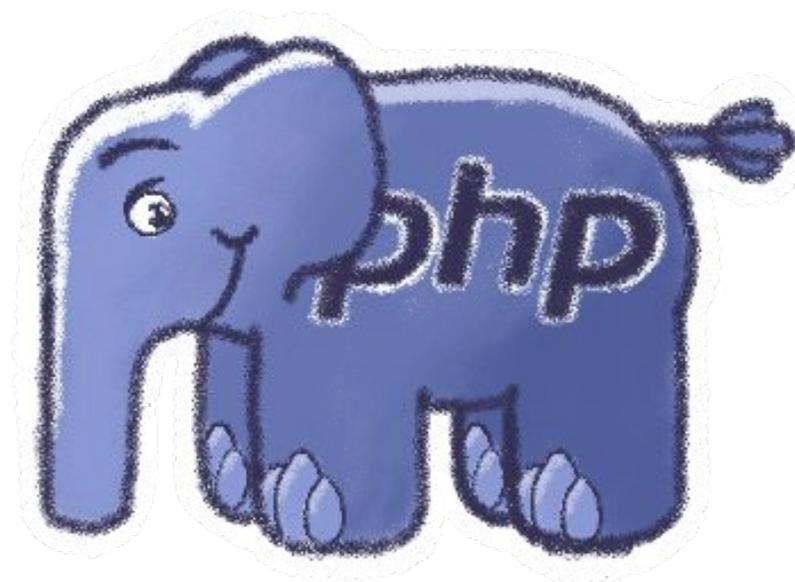
-----

[ERROR] Found 1 error
```

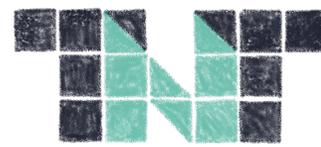




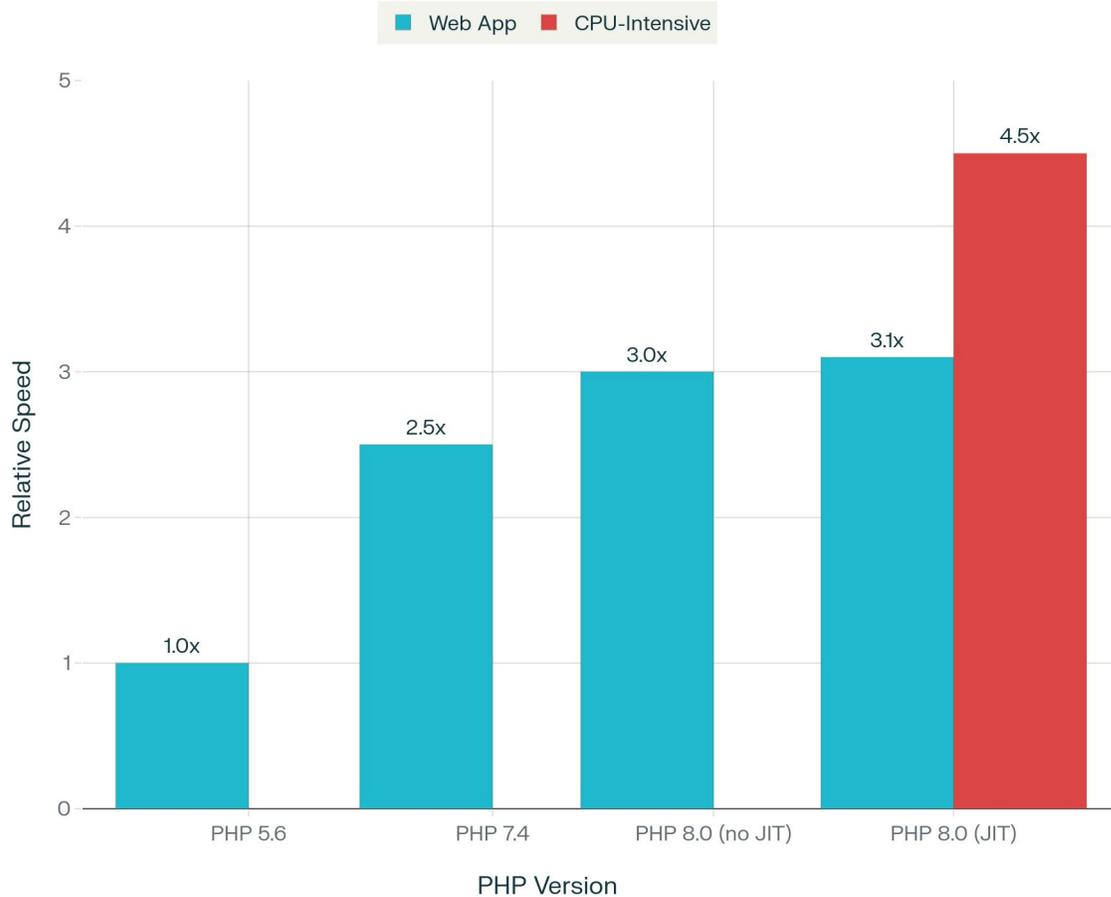
PHP - Performance & Architecture



JIT & Performance



PHP Performance Evolution



Sources: Kinsta, Tideways, ICDSOft, PHP manual, various public benchmarks

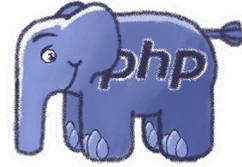
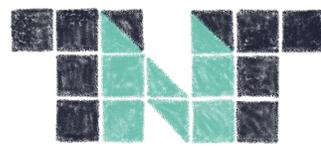
CPU-Intensive Tasks: PHP 8.3 vs Python 3.11



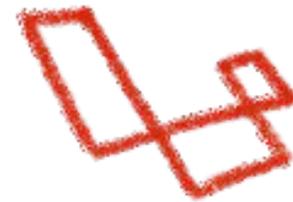
Source: The Computer Language Benchmarks Game (2024 Data)



The Frameworks: Symfony & Laravel



Symfony



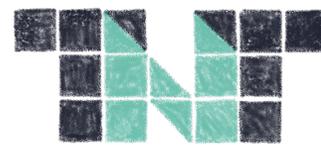
laravel

Symfony is Spring Boot. Laravel is Rails.

We aren't scripting anymore; we are architecting



Symfony: The Enterprise Standard



```
Java - Spring

@Service
public class ReportGenerator {
    private final Mailer mailer;

    @Autowired
    public ReportGenerator(Mailer mailer) {
        this.mailer = mailer;
    }
}
```

```
PHP - Symfony

use Symfony\Component\DependencyInjection\Attribute\Autowire;

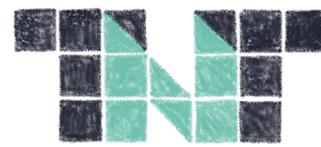
class ReportGenerator
{
    // Constructor Injection is the standard.
    // The container automatically injects the implementation o
    public function __construct(
        private MailerInterface $mailer
    ) {}
}
```

If you know Spring, you know Symfony

- Dependency Injection Container
- Decoupling
- Stability



Laravel: The "Developer Happiness" Framework



If you love the speed of Express or Rails, Laravel is that, but typed.

It's not just a framework; it's a platform.

- **Eloquent ORM:** (ActiveRecord)
Incredibly expressive
- **Queue Workers:** (Laravel Horizon)
Redis-backed queues out of the box.
- **Real-time:** (Laravel Reverb)
WebSockets without Node.js.
- **Serverless:** (Laravel Vapor)
AWS Lambda deployment helper.
- ...

```
PHP - Laravel
<?php
// Find all active users and email them... in 3 lines.
User::query()
    ->where('active', true)
    ->get()
    ->each(fn(User $user) =>
        Mail::to($user)->send(new WelcomeEmail()));
```



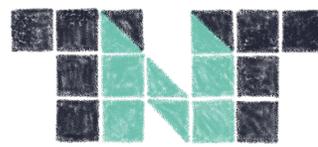
New Runtimes: Async & Long-Running Processes



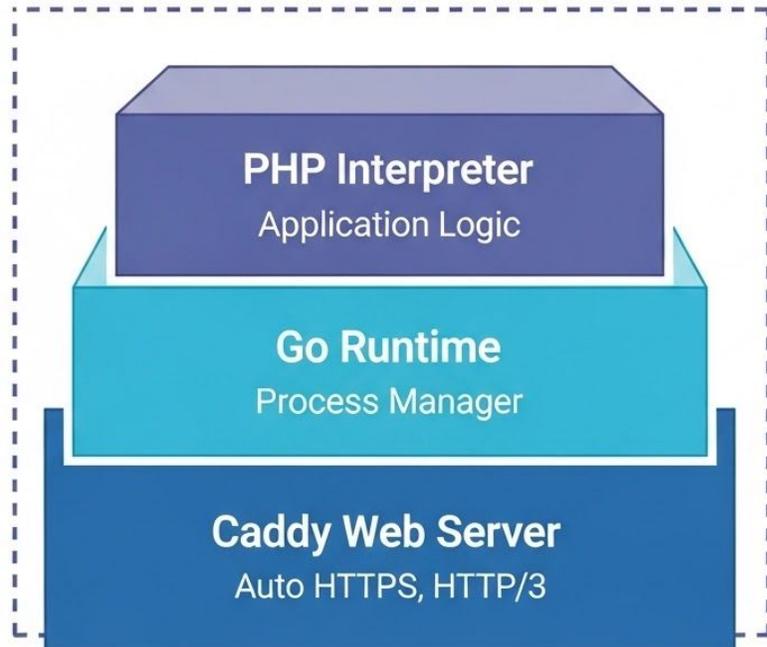
PHP used to die after every request. Not anymore.



The Game Changer: FrankenPHP



Single Binary Executable



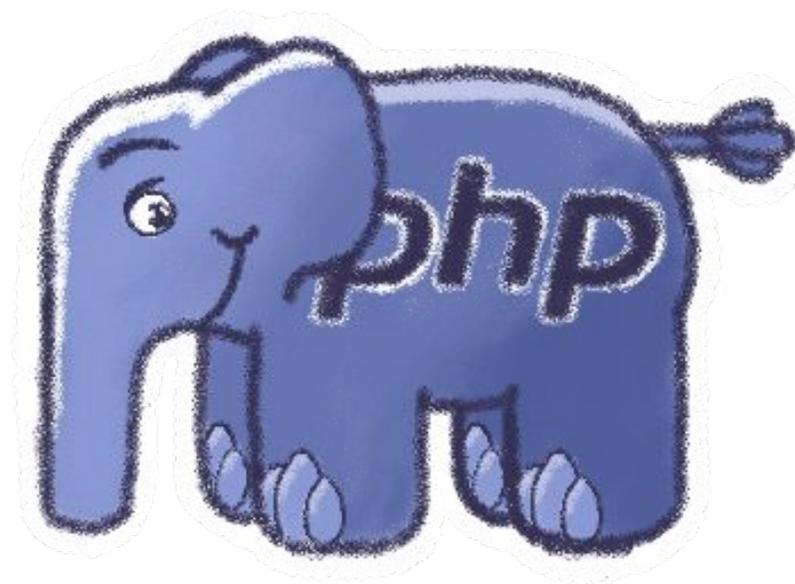
A **modern application server** written in **Go**, built on top of **Caddy**, that embeds the **PHP interpreter** and most popular **extensions**

- Worker Mode
- Early Hints (HTTP 103)
- Real-Time Mercure hub
- Static Binary

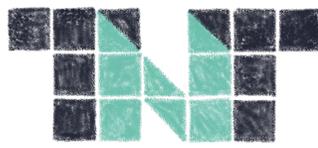




PHP - Myths, Misconceptions, and Reality Checks



"PHP is slow"



CPU-Intensive Tasks: PHP 8.3 vs Python 3.11

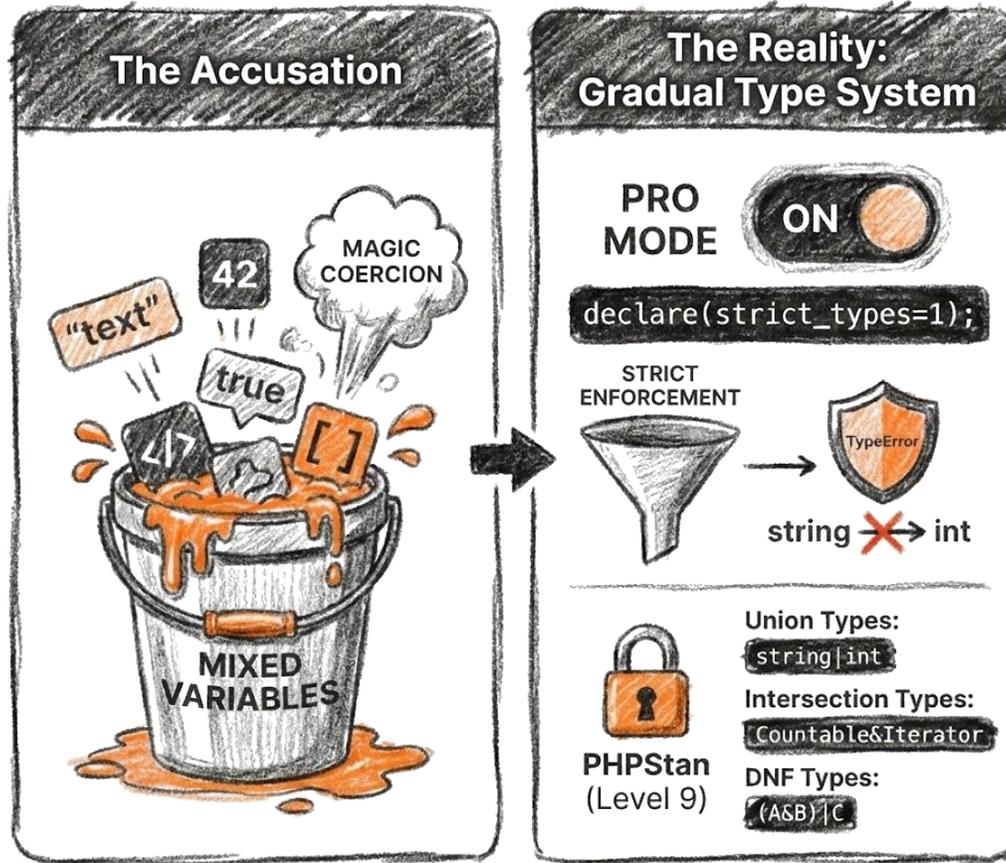


Source: The Computer Language Benchmarks Game (2024 Data)

Myth busted!



"PHP has no types"



Myth busted!



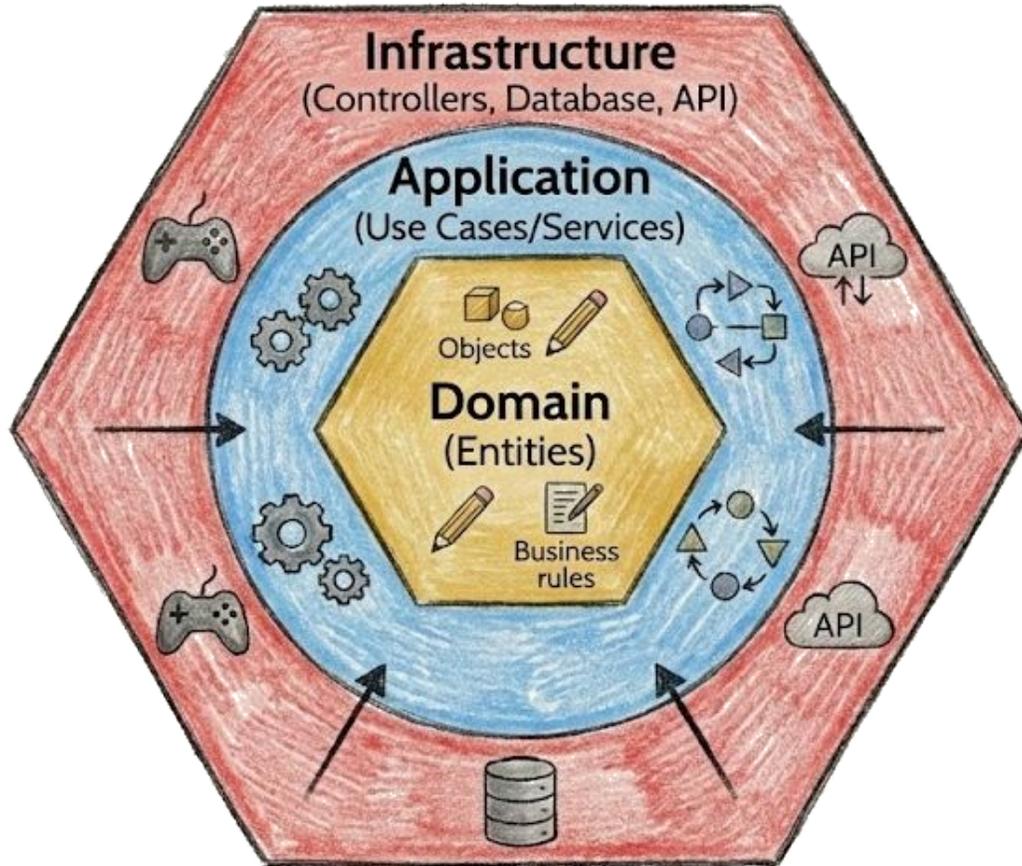
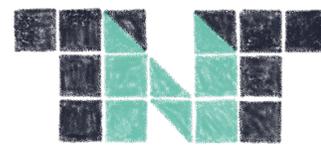
"PHP apps don't scale"



Myth busted!



"PHP code is spaghetti"



Modern PHP uses **Strict MVC** or **Clean Architecture**

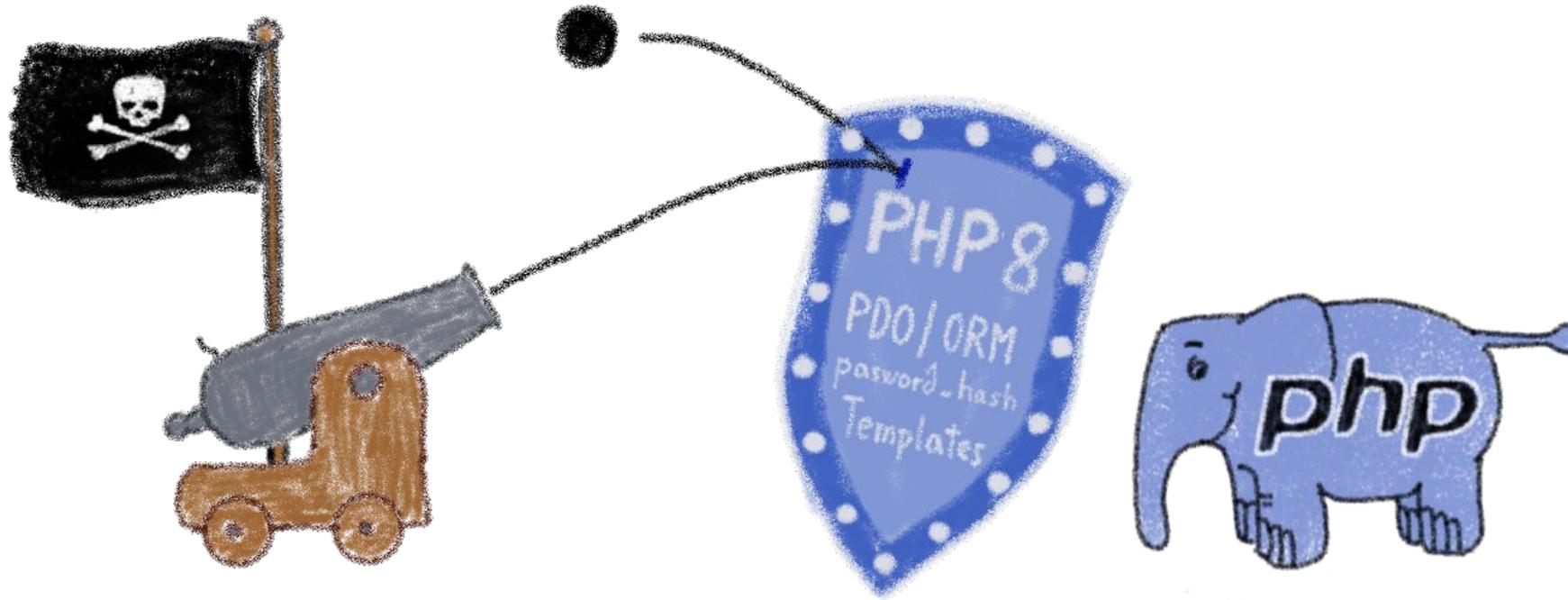
If you write 1000 lines of code in a single file in Java, it's spaghetti. If you do it in PHP, it's spaghetti.

Bad code is language-agnostic

Myth busted!



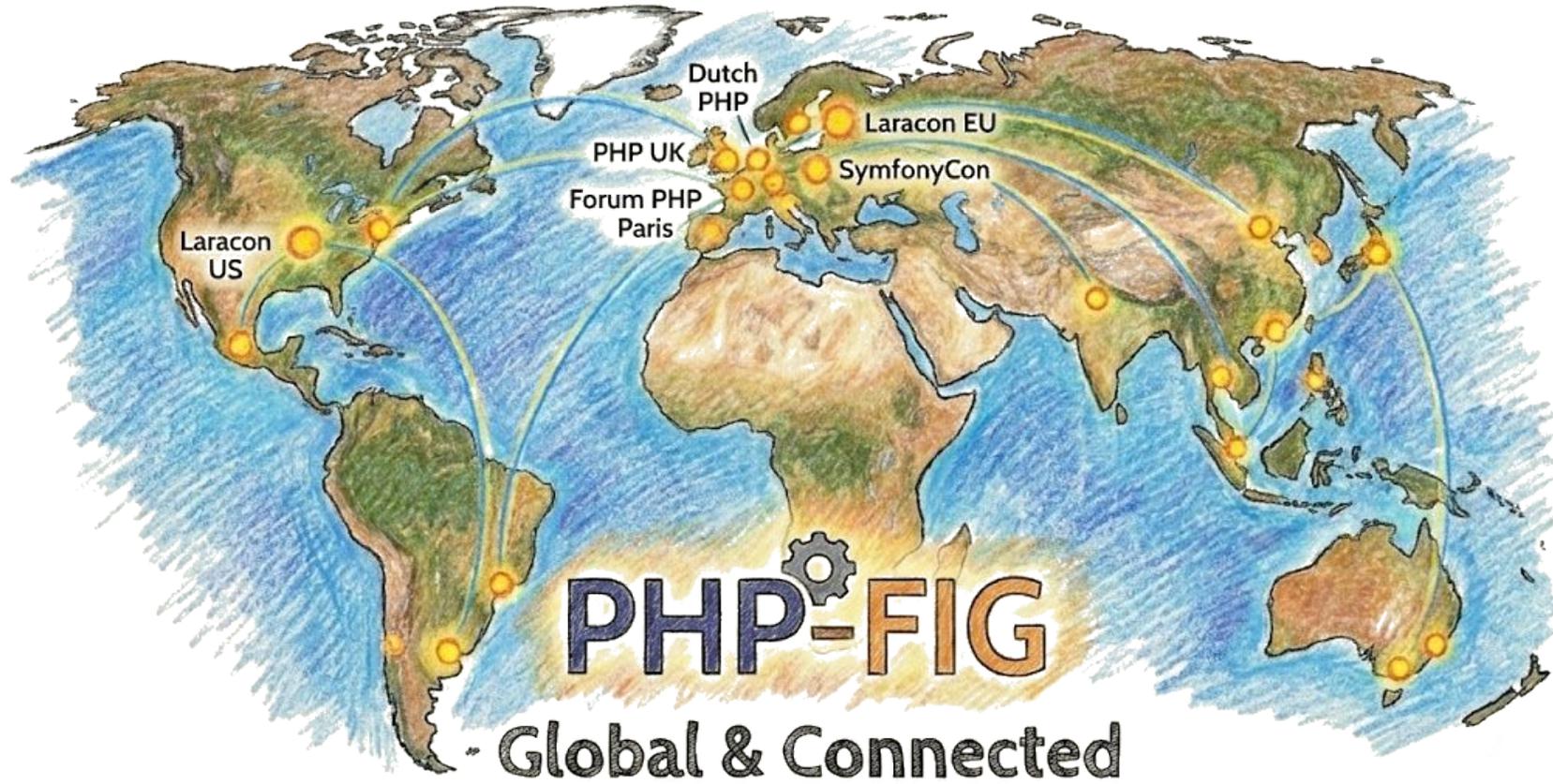
"PHP is insecure"



Myth busted!



"PHP developers are isolated"



Myth busted!

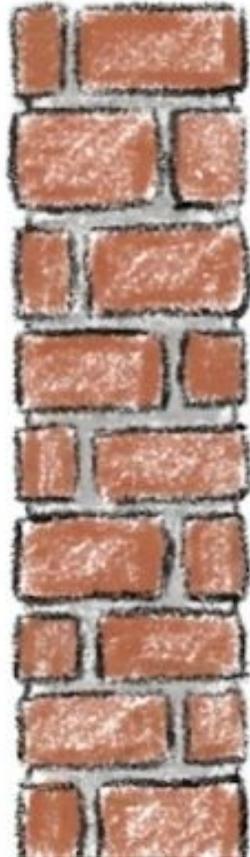


WordPress Question: "Why Does It Still Look Old?"



WORDPRESS

Chose backward compatibility



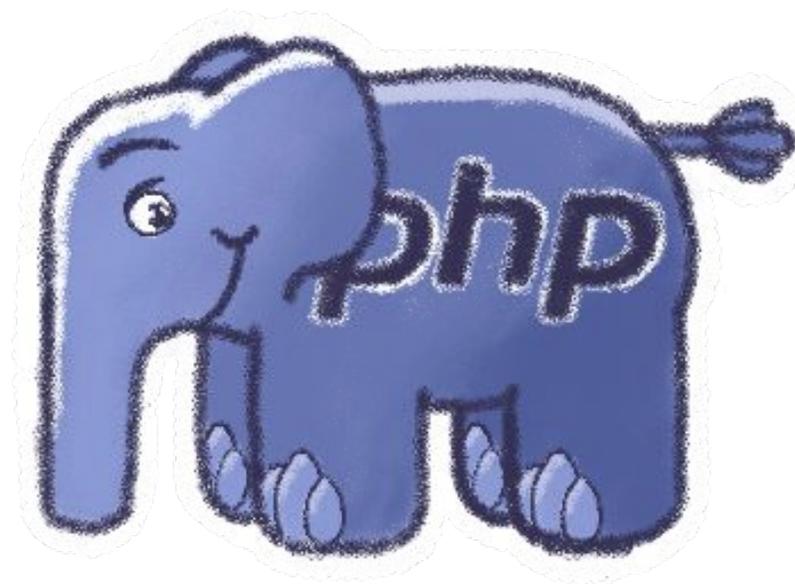
Chose modernization

Legacy Deployment ≠ Language Limitation

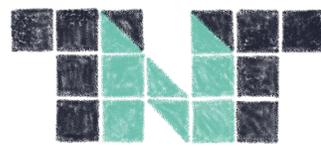




PHP - The Polyglot Conclusion



Reality Check: When PHP Isn't the Answer



Use PHP

- ~~~~~*
- ~~~~~*
- ~~~~~*
- ~~~~~*
- ~~~~-*

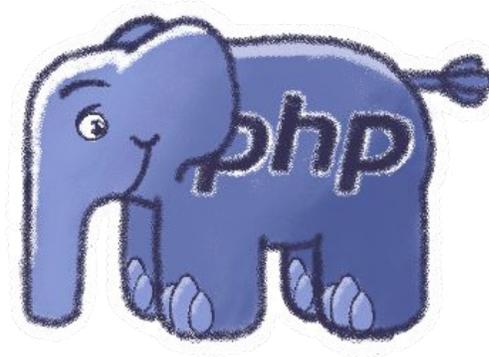
Consider Alternatives

- ~~~~~*
- ~~~~~*
- ~~~~~*
- ~~~~~*
- ~~~~~*

Honest Assessment: Choose the Right Tool



PHP isn't the language you remember



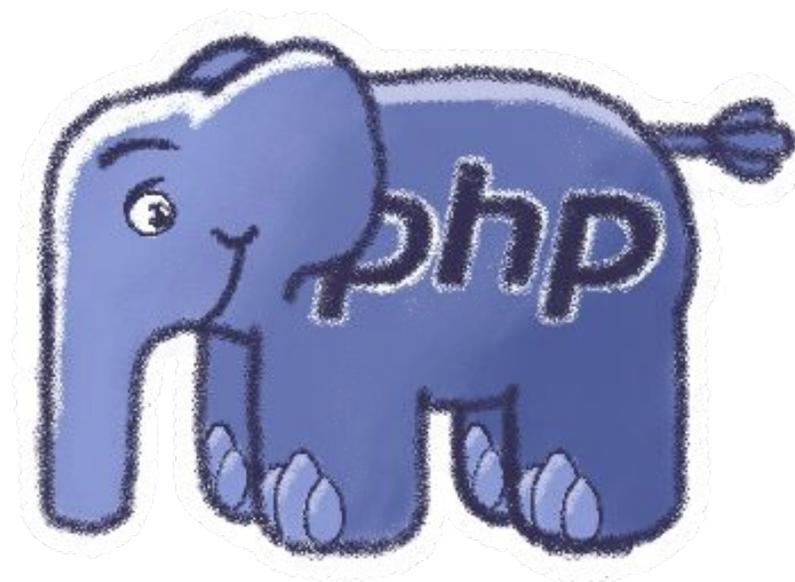
It's a pragmatic, high-performance tool that respects your time.

Don't hate it, add it to your toolbox

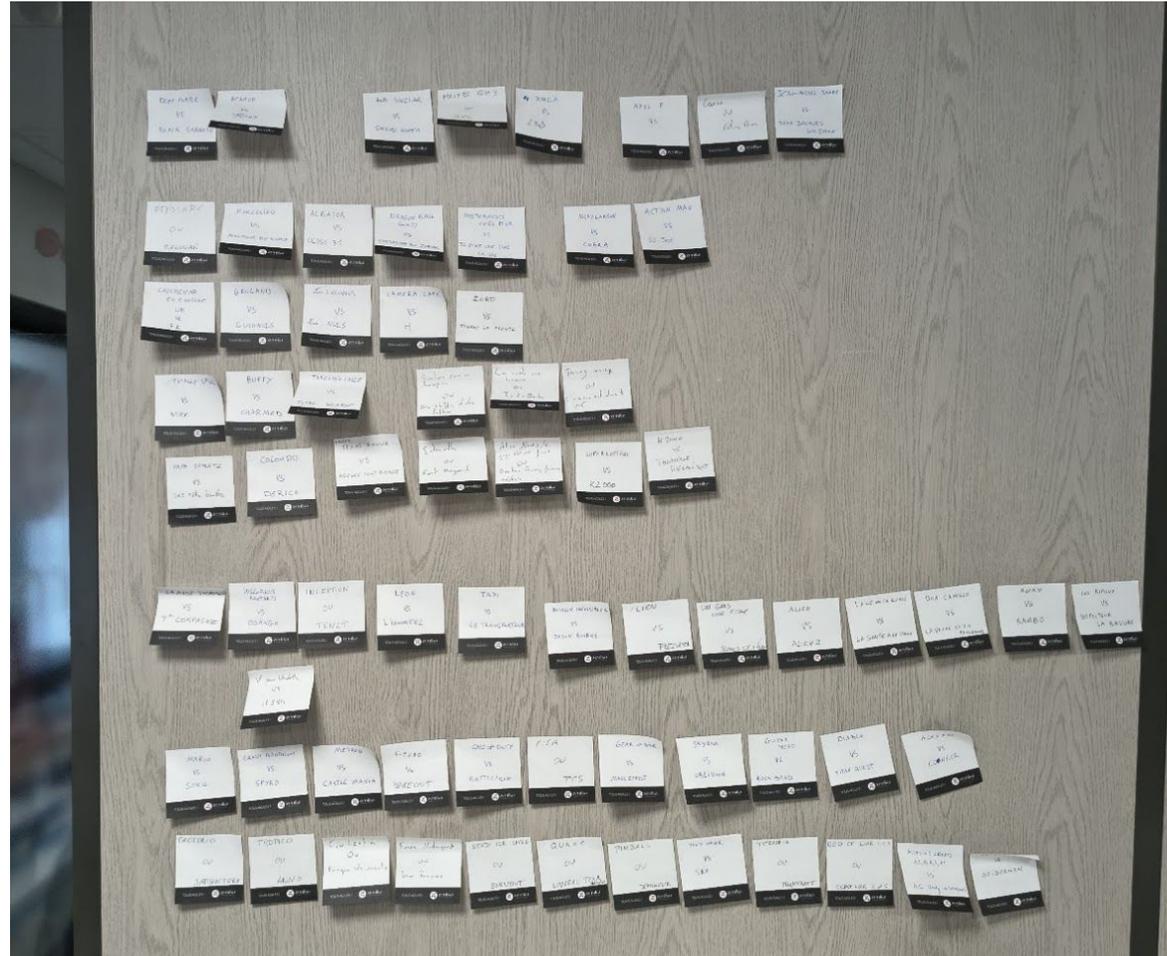
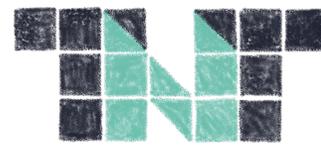




PHP - Live Demo / Code Walkthrough



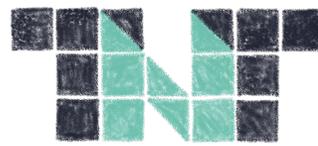
Meetup driven development



A challenge proposed by Zenika Clermont-Ferrand



Show me the code!



clash-of-pop-culture Public Pin Watch 0 Fork 0 Star 0

main 1 Branch 0 Tags Add file Code

LostInBrittany Screenshot and QRcode 7484859 · 11 minutes ago 4 Commits

assets	Screenshot and QRcode	11 minutes ago
bin	Initial commit	1 hour ago
config	Adding SQLite	1 hour ago
migrations	Screenshot and QRcode	11 minutes ago
public	Adding SQLite	34 minutes ago
src	Adding SQLite	1 hour ago
.editorconfig	Initial commit	1 hour ago
.env	Adding SQLite	1 hour ago

About

A demo application designed to showcase the capabilities and elegance of Modern PHP (8.3+) and the Symfony framework.

- Readme
- Activity
- 0 stars
- 0 watching
- 0 forks

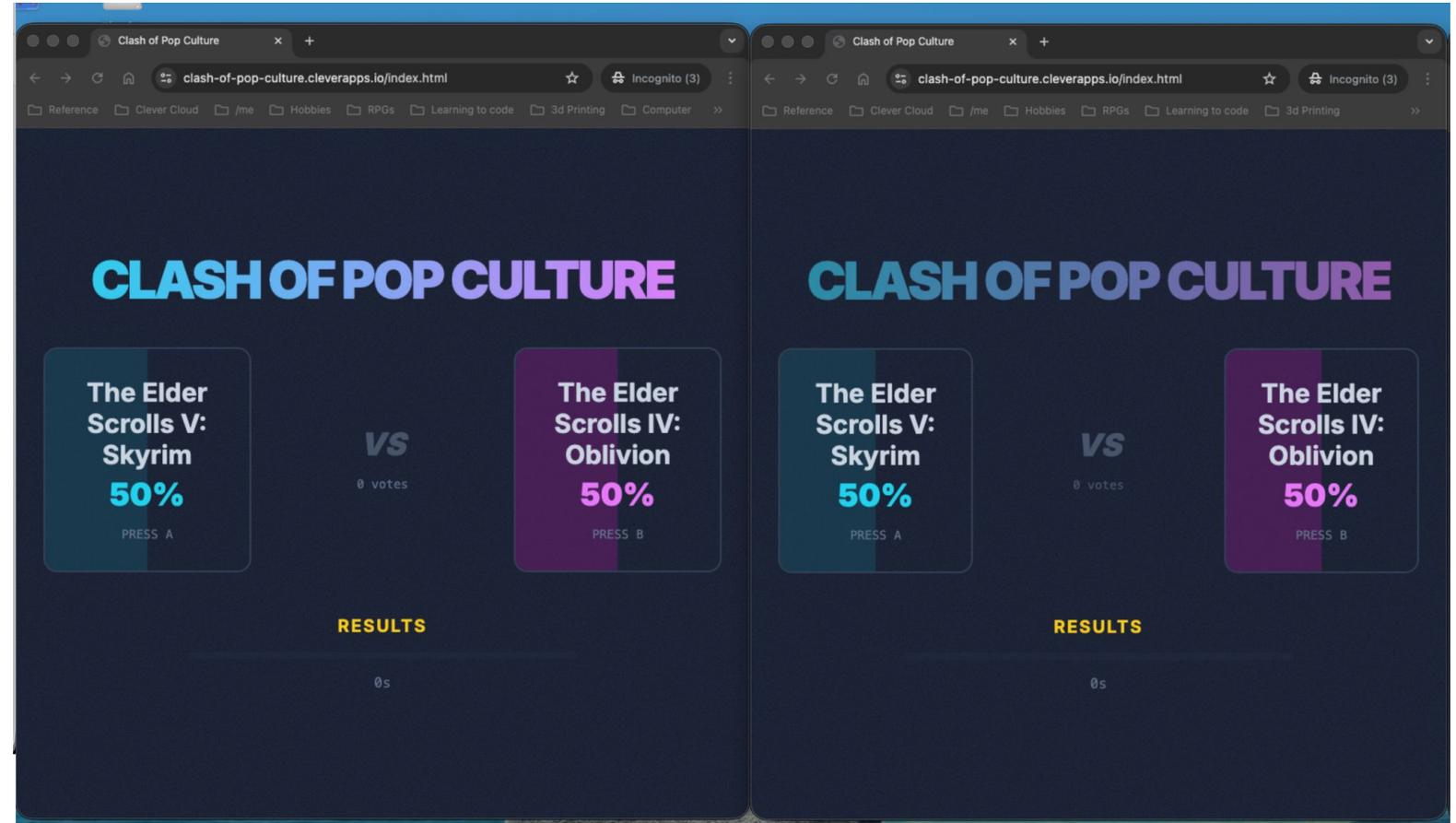
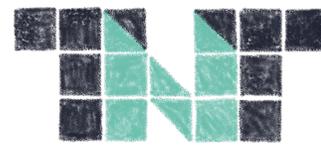
Releases

No releases published
[Create a new release](#)

<https://github.com/LostInBrittany/clash-of-pop-culture>



Clash of Pop Culture



<https://clash-of-pop-culture.cleverapps.io/index.html>





That's all, folks!

Thank you all!



*Please leave your
feedback!*

