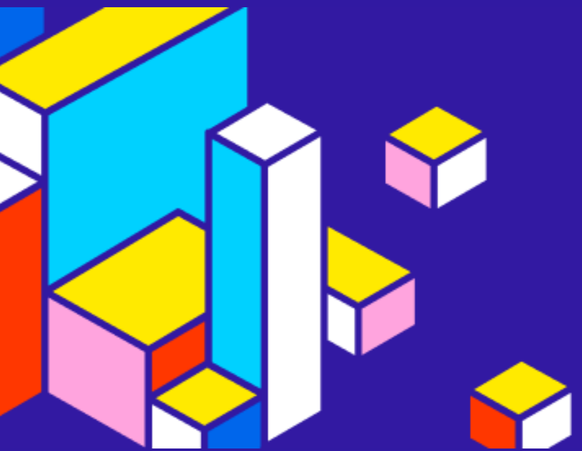
PHP GENERICS TODAY

DAVE LIDDAMENT

www.daveliddament.co.uk



LARAVEL LIVE DENMARK

Copenhagen · 20-21 August 2026

```
function process(User $user): void { … }
```

Type declaration

- **Communicate intent**

```
function process(User $user): void { … }
```

Type declaration

- **Communicate intent**
- **Run time checks**

```
process(“Bob”);
```

Bug. A string is not a User object.

```
function process(User $user): void { … }
```

Type declaration

- **Communicate intent**
- **Run time checks**
- **Static analysis checks**

```
process(“Bob”);
```

Bug. A string is not a User object.

```
function process(User $user): void { … }
```

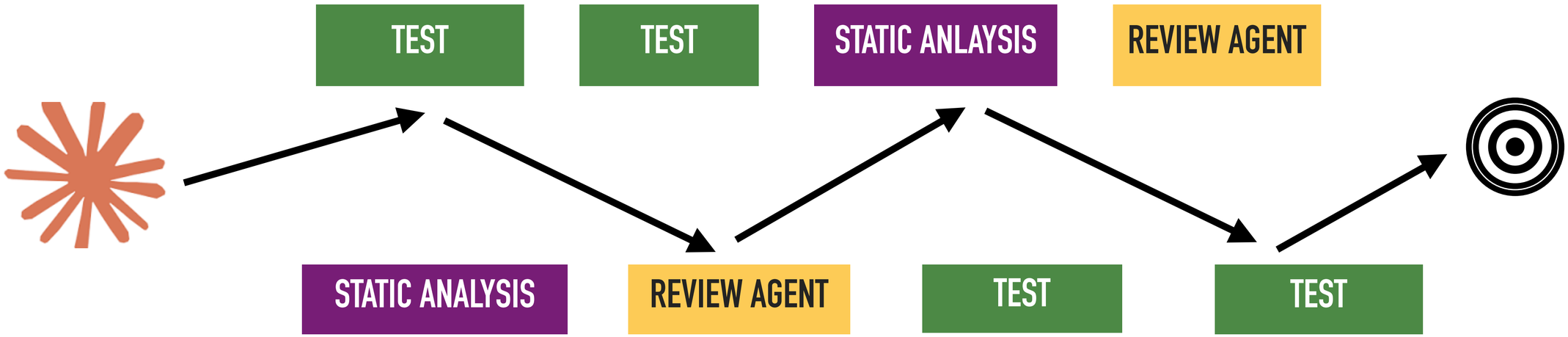
Type declaration

```
process(“Bob”);
```

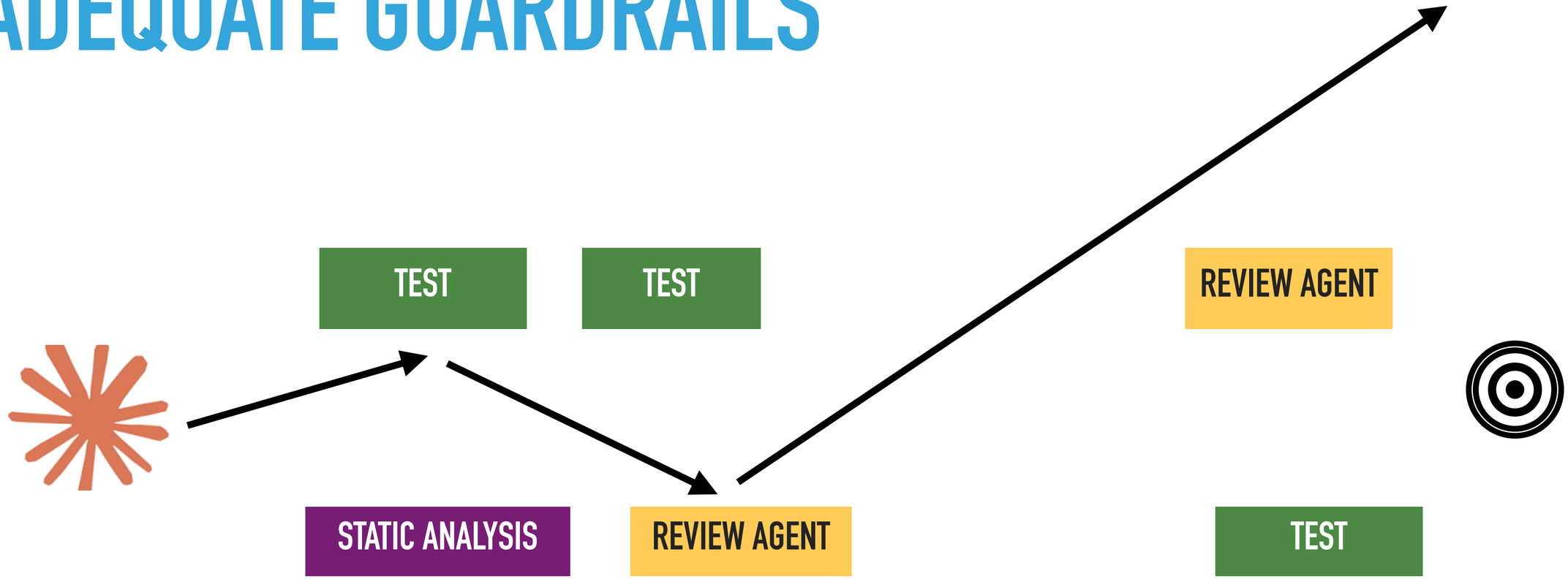
Bug. A string is not a User object.

- **Communicate intent**
- **Run time checks**
- **Static analysis checks**
- **Enable refactoring**
- **AI Assisted Development**

AI ASSISTED CODING WITH GUARDRAILS



INADEQUATE GUARDRAILS



Higher type coverage = more accurate static analysis

- Communicate intent
- Run time checks
- Static analysis checks
- Enable refactoring
- AI Assisted Development

```php
class PhpTypeSystemImprovements {
 public const string PHP_83 = 'typed class constants';
 private string $php74;
 public function php70(int $n): string {}
 public function php71(?string $s): void {}
 public function php72(object $o): object {}
 public function php80(int|string $id, mixed $anything): static {}
 public function php81(Countable&Traversable $c): never {}
 public function php82((Countable&Traversable)|null $c): true {}
}
```

BUT...

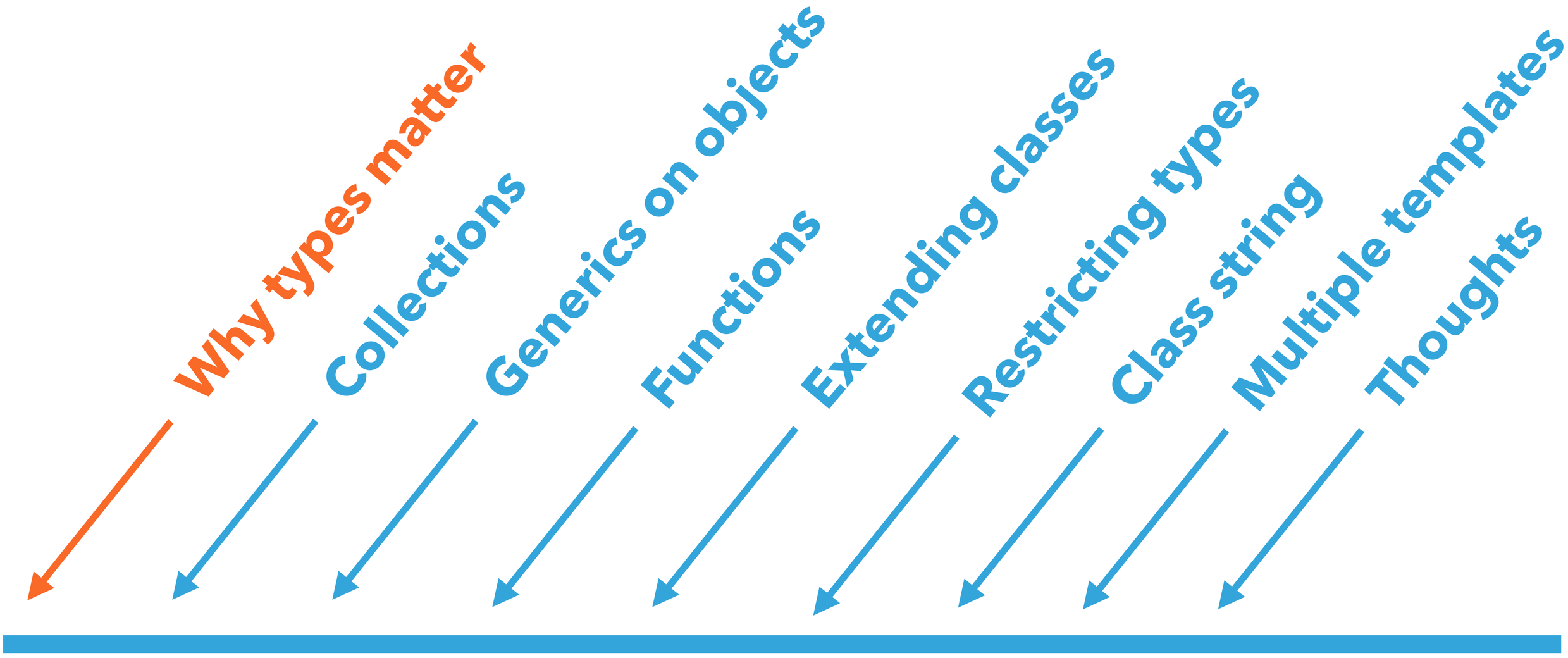
```
function process(array $users): void { … }
```

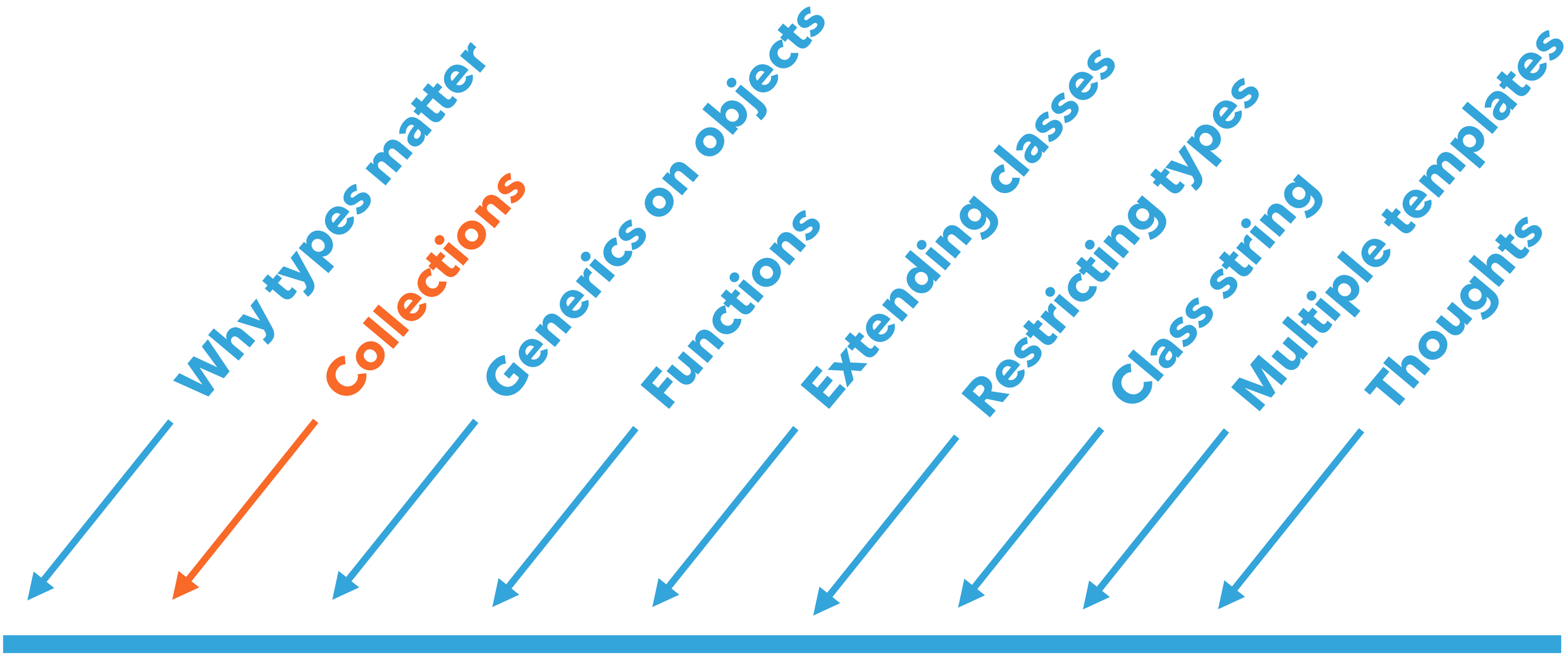
What is stored in the array?

AND...

```
class Queue {

    public function add(??? $item): void {…}

    public function getNext():??? {…}
}
```

What are these types?


Why types matter
Collections
Generics on objects
Functions
Extending classes
Restricting types
Class string
Multiple templates
Thoughts


Why types matter
Collections
Generics on objects
Functions
Extending classes
Restricting types
Class string
Multiple templates
Thoughts

```php
/** @return User[] */
function getUsers(): array {…}
```

```php
foreach(getUsers() as $user) {
  processUser($user);
}
```

```php
function processUser(User $user): void {…}
```

Type Hint

```
/** @return User[] */
function getUsers(): array {…}
```

Type Declaration

Psalm

PHPStan

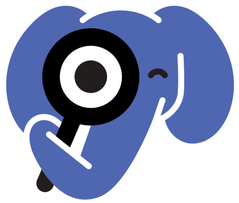
```
/** @return User[] $users */
function getUsers(): array
{
    return [
        new User(“Jane”),
        “james”,
    ];
}
```

```php
<?php declare(strict_types = 1);

class User {
    public function __construct(public string $name){}
}


/** @return User[] */
function getUsers(): array
{
    return [
        new User("jane"),
        "bob",
    ];
}
```

Share | Level 10 | Options

PHP 8.0 – 8.5 (1 error) | PHP 7.2 – 7.4 (2 errors)

Line	Error
11	Function getUsers() should return array<User> but returns array<int, string|User>.



Psalm

Type Hint

(Only checked by static analysis)

```
/** @return User[] */
function getUsers(): array {…}
```

Type Declaration

(Checked by PHP at runtime and by static analysis)

```
class Business {
    /** @return Employee[] */
    public function getEmployees(): array {…}
}

function promote(Employee $employee): void {…}
function welcome(string $name): void {…}

foreach($business->getEmployees() as $name => $employee) {
    welcome($name);
    promote($employee);
}
```

```
19 foreach($business->getEmployees() as $name => $employee) {
20     promote($employee);
21     welcome($name);
22 }
```

```
Psalm output (using commit add7c14):

INFO: MixedArgument - 21:12 - Argument 1 of welcome cannot be mixed, expecting string
```

```
class Business {
    /** @return array<string,Employee> */
    public function getEmployees(): array {…}
}

function promote(Employee $employee): void {…}
function welcome(string $name): void {…}


foreach($business->getEmployees() as $name => $employee)
{
    welcome($name);
    promote($employee);
}
```

```
/** @var array<V> */
```

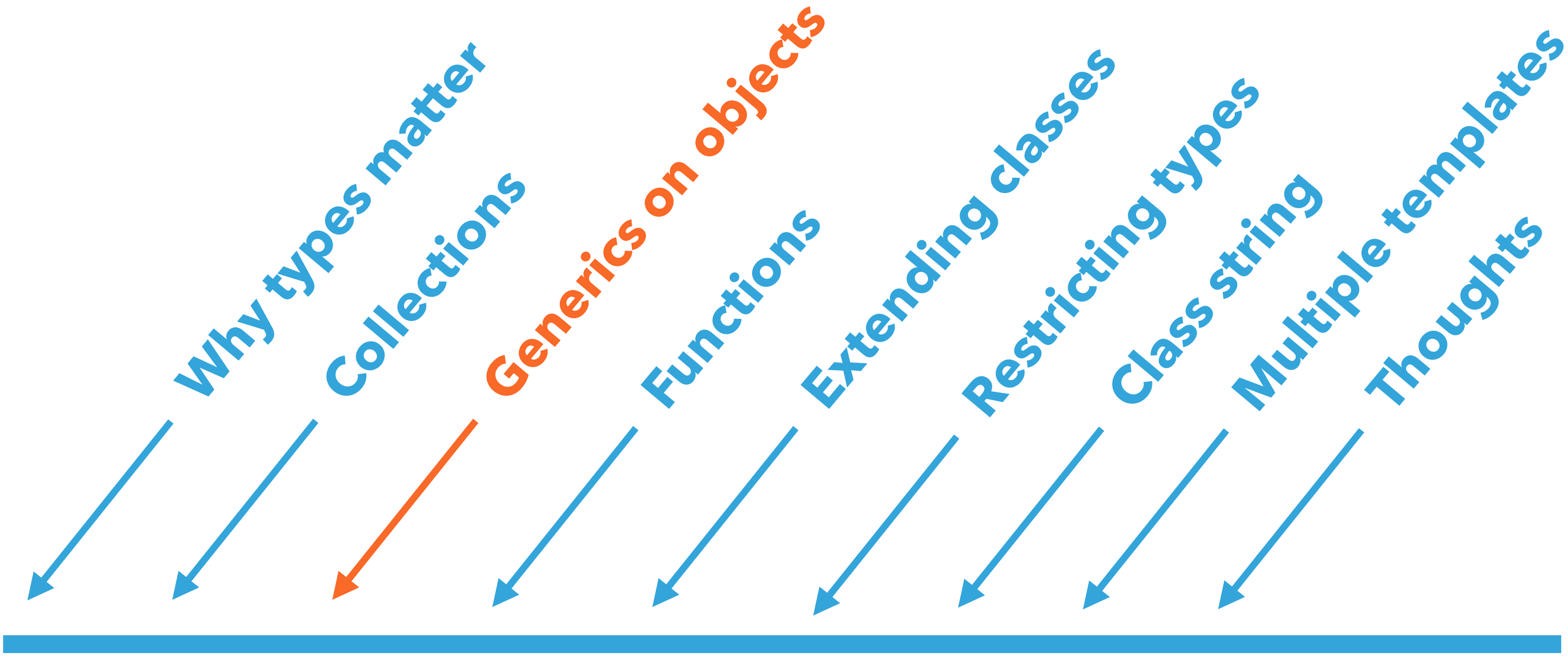
```
/** @var array<Person> */
$people = [ … ];
```

```
/** @var array<K, V> */
```

```
/** @var array<string, Person> */
$people = [ … ];
```

```
/** @var Collection<K, V> */
```

```
/** @var Collection<string, Person> */
$people = new Collection();
```


Why types matter
Collections
Generics on objects
Functions
Extending classes
Restricting types
Class string
Multiple templates
Thoughts

```
class Queue {

    public function add(??? $item): void {…}

    public function getNext():??? {…}
}
```

What are these types?

```
class Queue <T> {

  public function add(T $item): void {…}

  public function getNext():T {…}
}
```

```php
/** @template T */
class Queue <T> {

    /** @param T $item */
    public function add(T $item): void {…}

    /** @return T */
    public function getNext(): T{…}
}
```

```
/** @var Queue<User> $userQueue */
$userQueue = new Queue();
```

```php
/** @var Queue<User> $userQueue */
$userQueue = new Queue();
```

```php
/** @template User */
class Queue
{
    /** @param User $item */
    public function add(  $item): void {…}

    /** @return User */
    public function getNext()   {…}
}
```

```php
/** @var Queue<User> $userQueue */
$userQueue = new Queue();

$userQueue->add(new User()); ✅

$userQueue->add("Bob"); ❌
```

WITHOUT GENERICS

```
$userQueue = new Queue();

$userQueue->add(new User());   ✅

$userQueue->add("Bob"); ✅
```

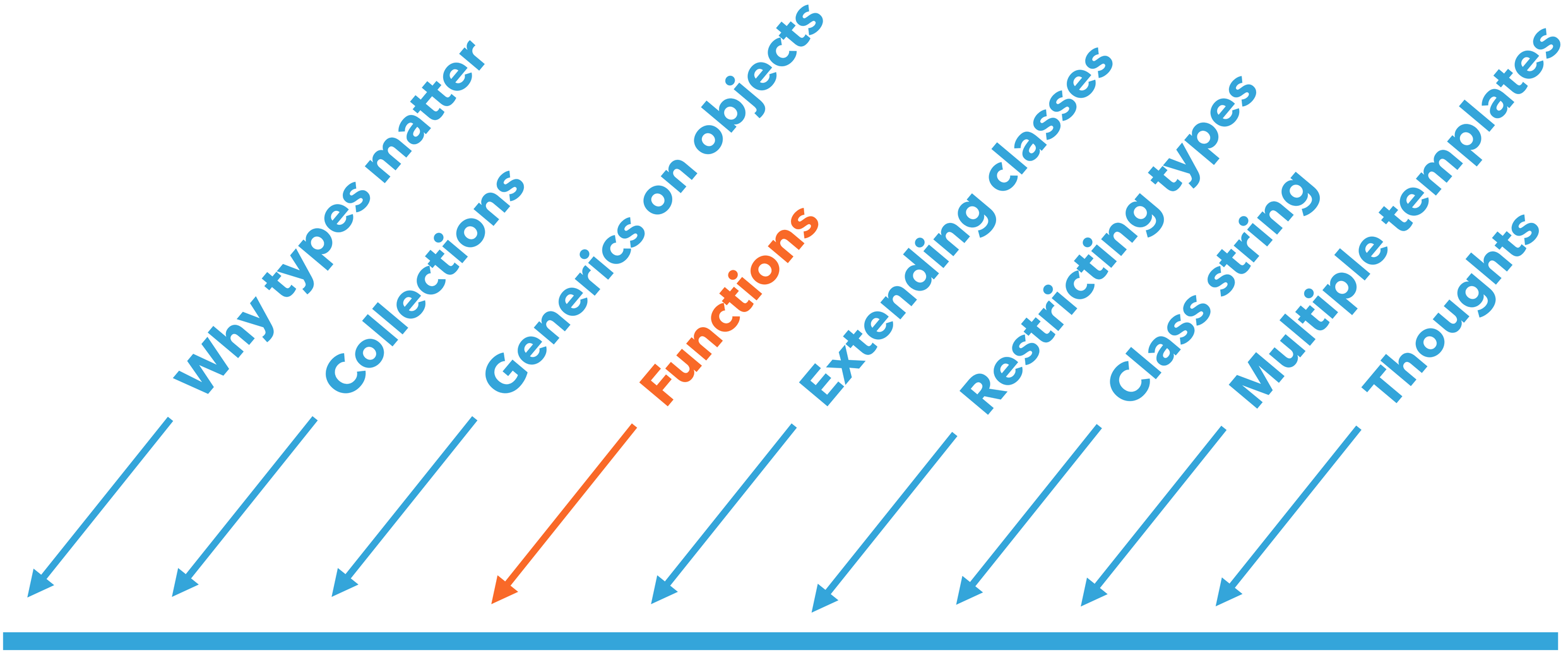
```php
/** @var Queue<Book> $userQueue */
$userQueue = new Queue();
```

```php
/** @template Book */
class Queue    {

  /** @param Book $item */
  public function add(  $item): void {…}

  /** @return Book */
  public function getNext()   {…}
}
```

```php
/** @var Queue<Book> $readingList */
$readingList = new Queue();

$readingList->add(new Book()); ✅

$readingList->add(new User()); ❌
```

```php
/** @template T */
class Queue() { … }

/** @var Queue<User> $userQueue */
$userQueue = new Queue();
```

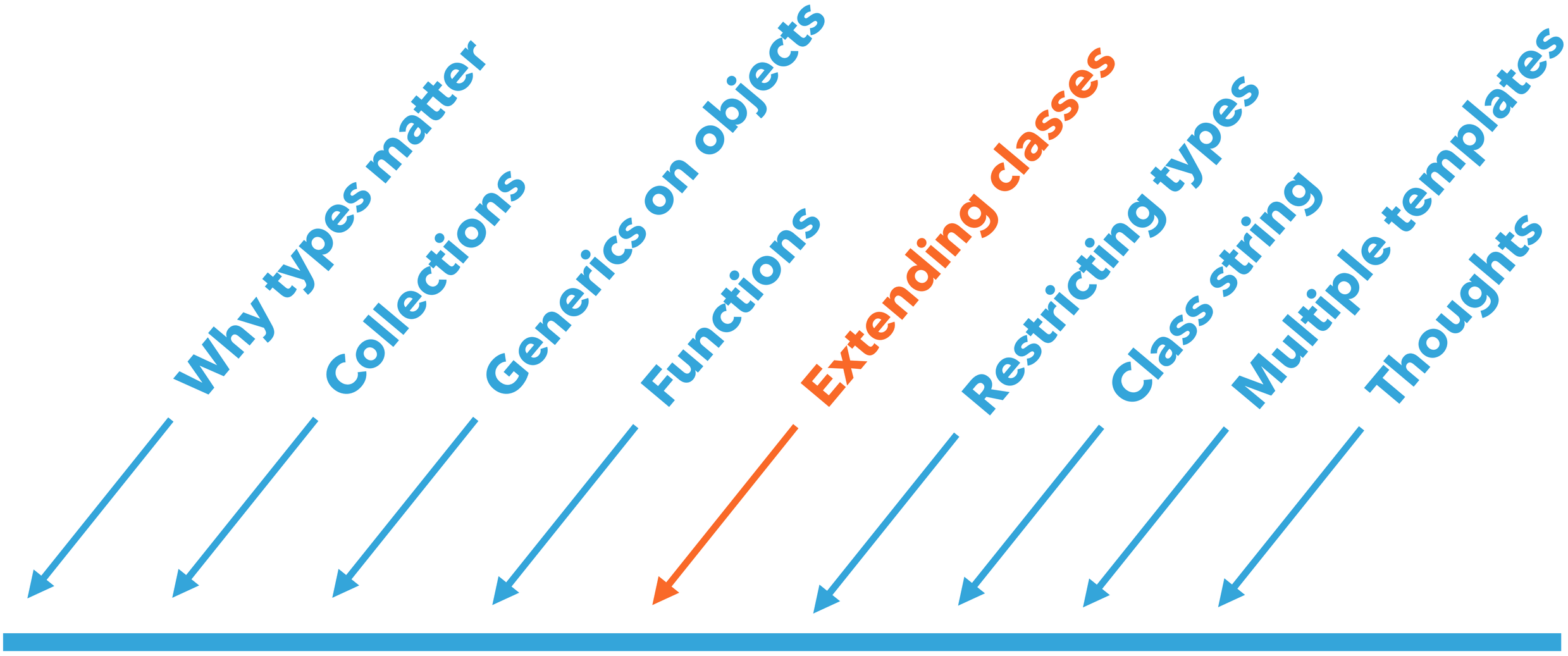
- Communicate intent
- ~~Run time checks~~
- Static analysis checks
- Enable refactoring
- AI Assisted Development


Why types matter
Collections
Generics on objects
Functions
Extending classes
Restricting types
Class string
Multiple templates
Thoughts

```
/**
 * @template T
 * @param T $value
 * @return T
 */
function mirror($value) { return $value; }
```

```php
/**
 * @template T
 * @param T $input
 * @return T
 */
function mirror($input) { return $input; }
$value = mirror(5);
```

```
/**
 * @template T
 * @param T $value
 * @return array<T>
 */
function asArray($value) { return [$value]; }
$values = asArray(5);
```


Why types matter
Collections
Generics on objects
Functions
Extending classes
Restricting types
Class string
Multiple templates
Thoughts

```
class Animal { … }

class Dog extends Animal {
  public function bark(): void {…}
}

class Cat extends Animal {
  public function meow(): void {…}
}
```

```
/** @template T */
interface AnimalGame {

    /** @param T $animal */
    public function play($animal): void;
}
```

```
/** @implements AnimalGame<Dog> */
class DogGame implements AnimalGame {…}
```

```php
/** @implements AnimalGame<Dog> */
class DogGame implements AnimalGame {…}
```

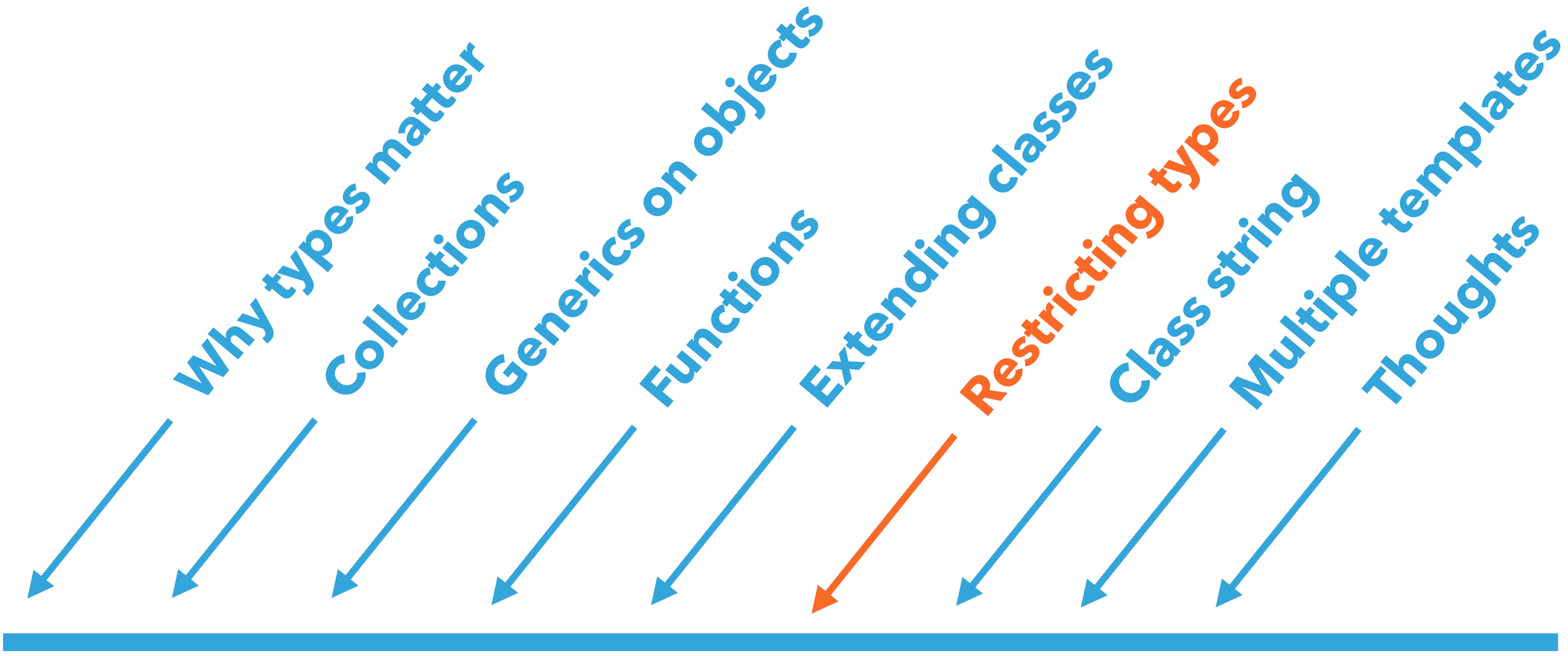
```php
/** @template Dog */
interface AnimalGame {

  /** @param Dog $animal */
  public function play($animal): void;
}
```

```
/** @implements AnimalGame<Dog> */
class DogGame implements AnimalGame {

    public function play($animal): void {
        $animal->bark(); // We know $animal is a dog
    }
}
```

```
/** @implements AnimalGame<Dog> */
class DogGame implements AnimalGame {

  public function play($animal): void {
    $animal->meow(); // Dogs can't meow
  }
}
```

```
/** @implements AnimalGame<Dog> */
class DogGame implements AnimalGame {…}
```

```
/** @extends AnimalGame<Dog> */
class DogGame extends AnimalGame {…}
```

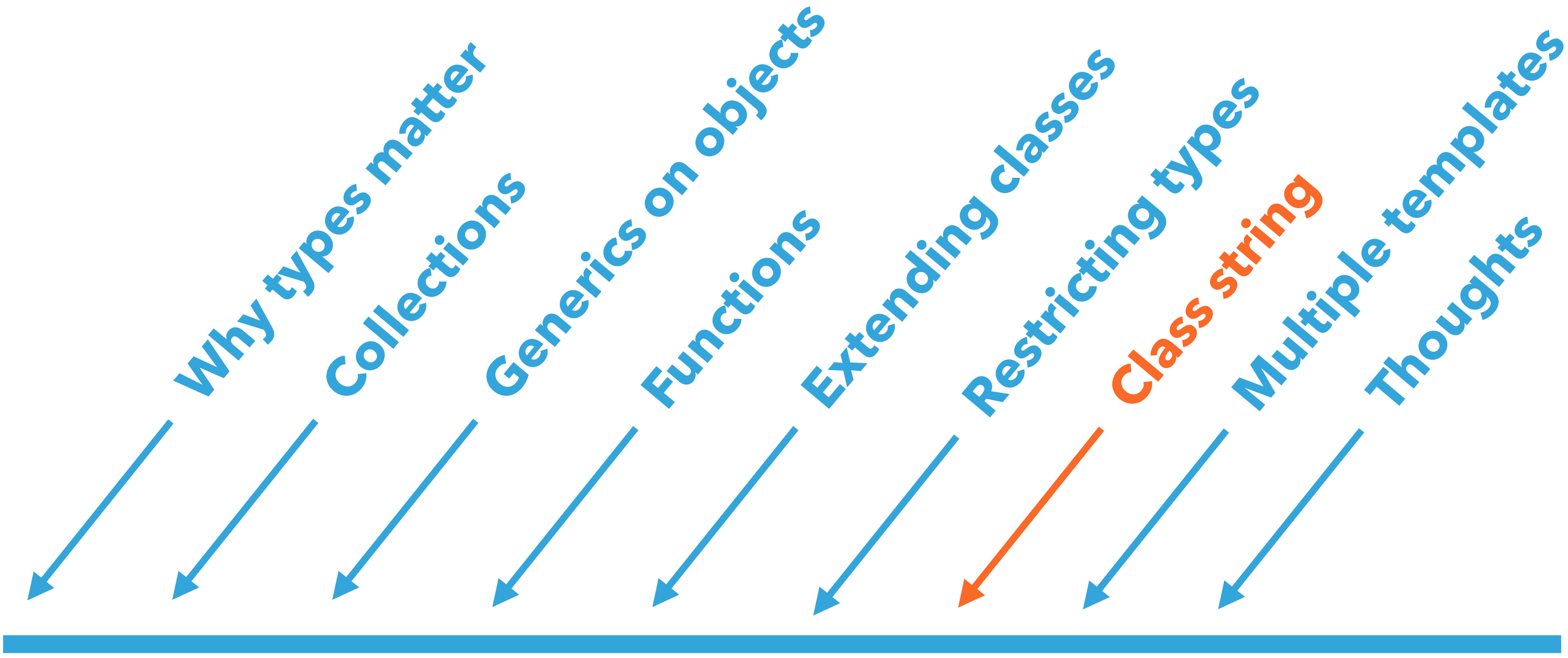

Why types matter
Collections
Generics on objects
Functions
Extending classes
Restricting types
Class string
Multiple templates
Thoughts

```
/** @implements AnimalGame<Car> */
class CarGame implements AnimalGame { … }
```

```
/** @template T of Animal */
interface AnimalGame { … }

/** @implements AnimalGame<Car> */
class CarGame implements AnimalGame { … }

/** @implements AnimalGame<Cat> */
class CatGame implements AnimalGame { … }
```

❌

✅


Why types matter
Collections
Generics on objects
Functions
Extending classes
Restricting types
Class string
Multiple templates
Thoughts

CLASS STRING

```
App\Entities\Person
```

```
Person::class
```

```
class Person {…}
```

```
/**
 *
 * @param string $className
 * @return object
 */
function app(string $className): object {…}
```

```
$person = app(Person::class);
```

```
class Person {…}
```

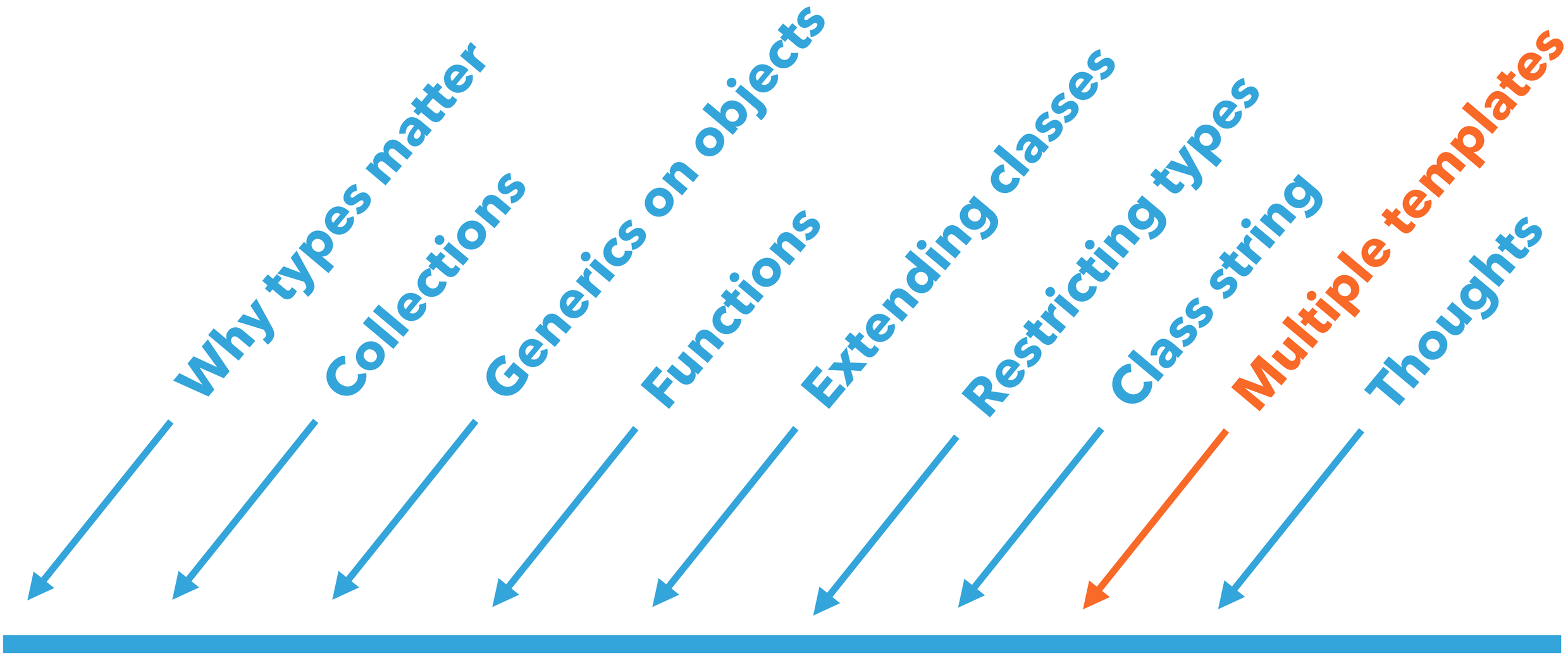
```
 /**
  *
  * @param string $className
  * @return object
  */
 function app(string $className): object {…}
```

```
 $person = app(Person::class);
```

```
class Person {…}
```

```
/**
 * @template T of object
 * @param class-string<T> $className
 * @return T
 */
function app(string $className): object {…}
```

```
$person = app(Person::class);
```


Why types matter
Collections
Generics on objects
Functions
Extending classes
Restricting types
Class string
Multiple templates
Thoughts

```
/**
 * @template T
 * @template U
 */
final readonly class Tuple {

    /**
     * @param T $a
     * @param U $b
     */
    public function __construct(
            public $a,
            public $b,
        ) {}
 }
```

```php
/**
 * @template T
 * @template U
 */
final readonly class Tuple {

 /**
  * @param T $a
  * @param U $b
  */
 public function __construct(
        public $a,
        public $b,
     ) {}
 }
```

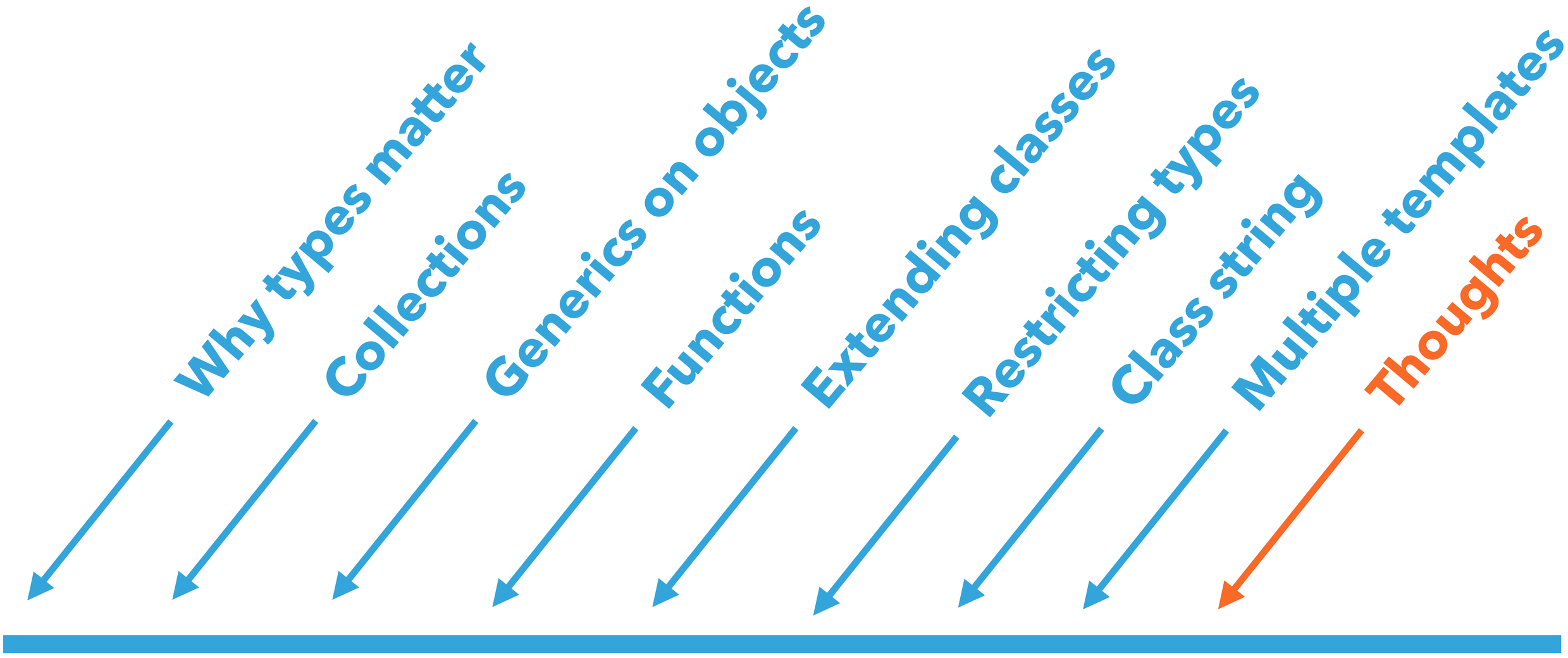
```php
/**
 * @param Tuple<User,Book> $result
 */
function process(Tuple $result)
{

  // $user is a User object
  $user = $result->a;

  // $book is a Book object
  $book = $result->b;
}
```

```php
/**
 * @template T
 * @template U
 */
final readonly class Tuple {

 /**
  * @param T $a
  * @param U $b
  */
 public function __construct(
        public $a,
        public $b,
     ) {}
 }
```

```php
/**
 * @param Tuple<User,Book> $result
 */
function process(Tuple $result)
{

  // $user is a User object
  $user = $result->a;

  // $book is a Book object
  $book = $result->b;
}
```

```php
/**
 * @template T
 * @template U
 */
final readonly class Tuple {

 /**
  * @param T $a
  * @param U $b
  */
 public function __construct(
        public $a,
        public $b,
    ) {}
 }
```

```php
/**
 * @param Tuple<User,Book> $result
 */
function process(Tuple $result)
{

  // $user is a User object
  $user = $result->a;

  // $book is a Book object
  $book = $result->b;
}
```


Why types matter
Collections
Generics on objects
Functions
Extending classes
Restricting types
Class string
Multiple templates
Thoughts

- Communicate intent
- Run time checks
- Static analysis checks
- Enable refactoring
- AI Assisted Development

Type Hint

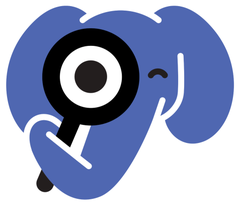
(Only checked by static analysis)

```
/** @return User[] */
function getUsers(): array {…}
```

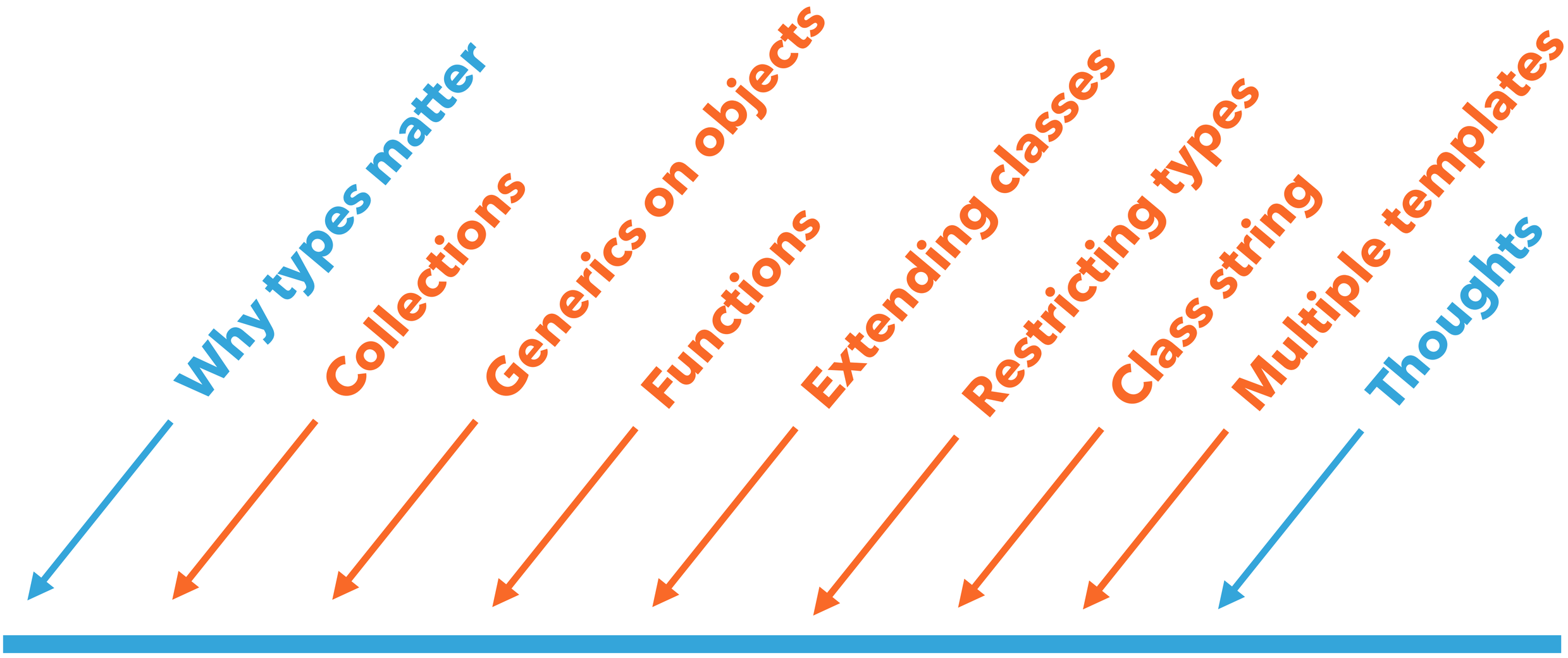
Type Declaration

(Checked by PHP at runtime and by static analysis)

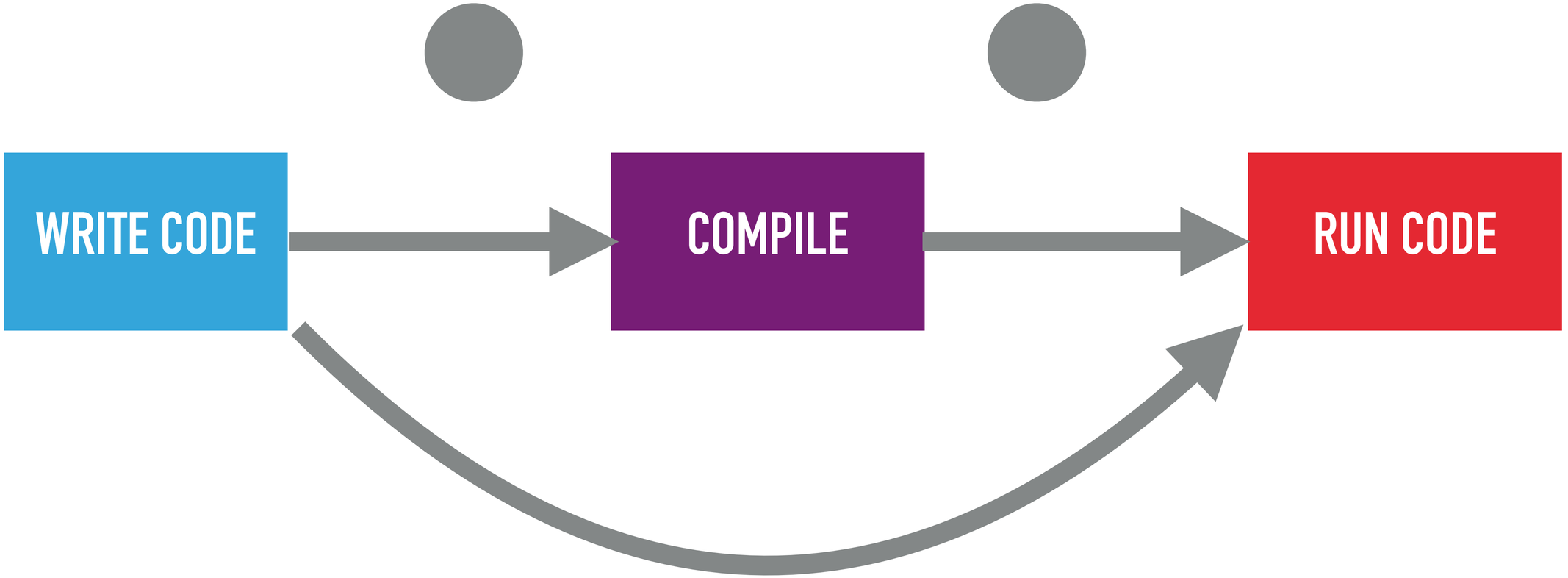
- Communicate intent
- ~~Run time checks~~
- Static analysis checks
- Enable refactoring
- AI Assisted Development



Psalm


Why types matter
Collections
Generics on objects
Functions
Extending classes
Restricting types
Class string
Multiple templates
Thoughts

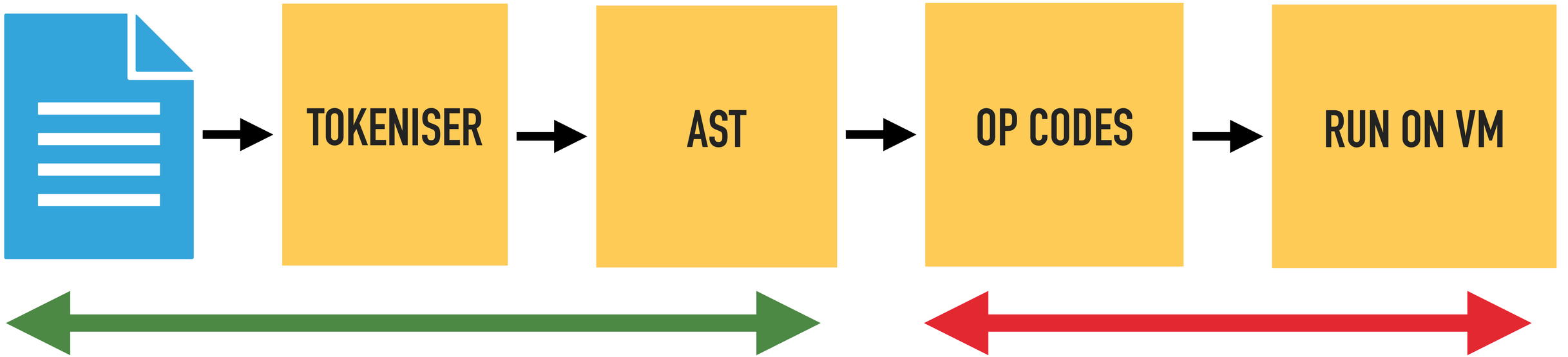
WHY NOT JUST USE JAVA?



IMPLEMENTING GENERICS

- Full language support
- Partial language support
  - Valid syntax
  - Ignored at run time
- PSR / similar

PARTIAL LANGUAGE SUPPORT



Validated by Static Analysis, not at run time.

https://wiki.php.net/rfc/bound_erased_generic_types

```
class Queue <T: Animal>
{
  private array<int,T> $queue = [];

  public function add(T $item): void {…}

  public function next(): T {…}
}

$personQueue = new Queue::<Dog>();
```

PHP GENERICS TODAY (ALMOST)

USING GENERICS NOW

Psalm

WE ALREADY HAVE GENERICS!

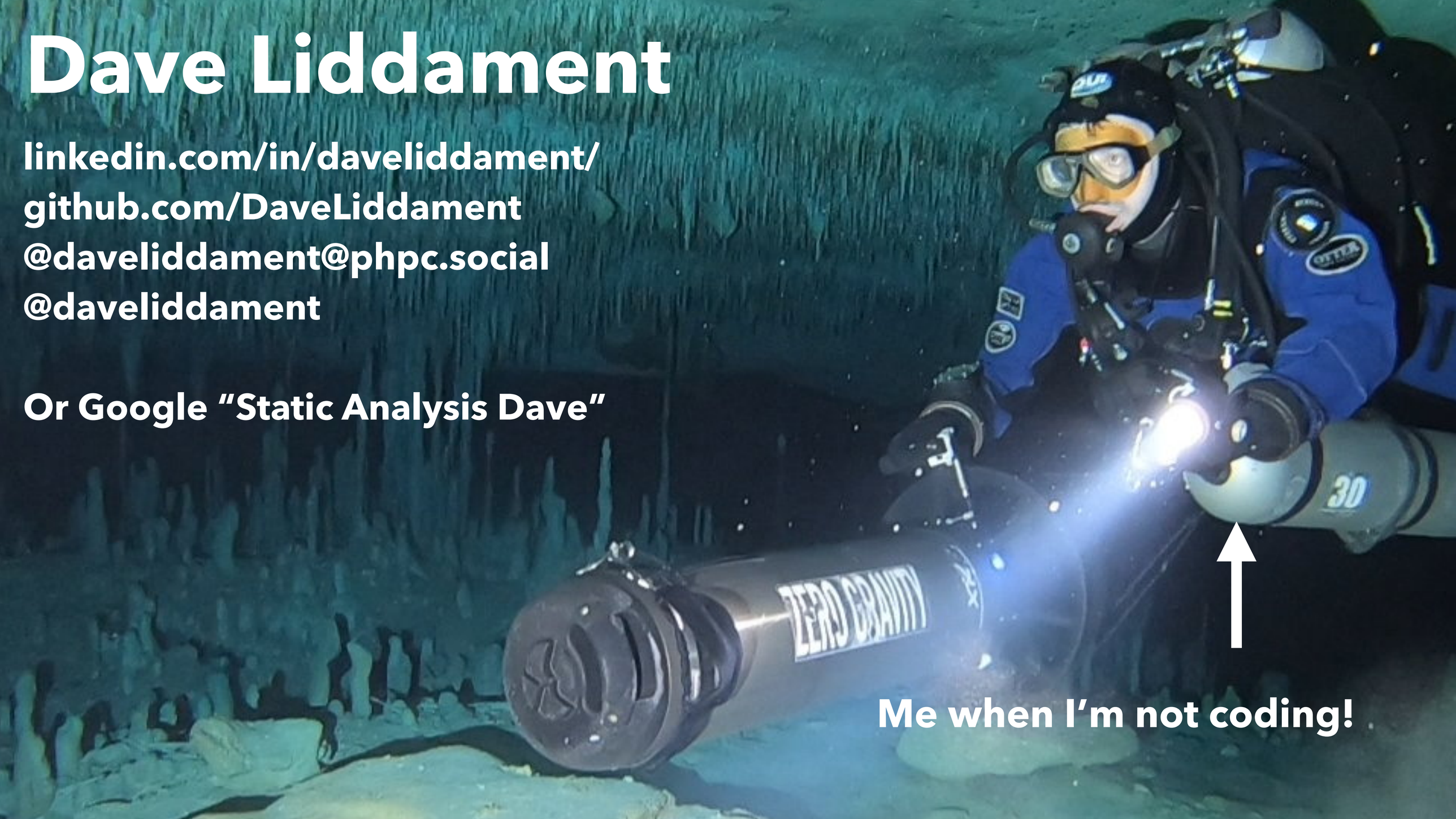
I'VE BEEN DOING THIS FOR 6 YEARS.

NOW YOU CAN TOO!

Dave Liddament

linkedin.com/in/daveliddament/
github.com/DaveLiddament
@daveliddament@phpc.social
@daveliddament

Or Google “Static Analysis Dave”



Me when I’m not coding!