I'm speaking at PHP Tek 2026

Type Safe PHP: Leveraging Static Analysis for Robust Code

Wed, 5/20/2026 1:00 pm — Chicago, Illinois, United States

Dave Liddament

https://phptek.io

PHP's type system is improving with every release....

But there is still a missing part: Generics.

Using generics can help us write more understandable, robust and reliable code.

Demonstrate how existing tools can (almost) give us the benefits of generics now.

IS THIS TALK FOR YOU?

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

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

process(“Bob”);
```

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

process(“Bob”);
```

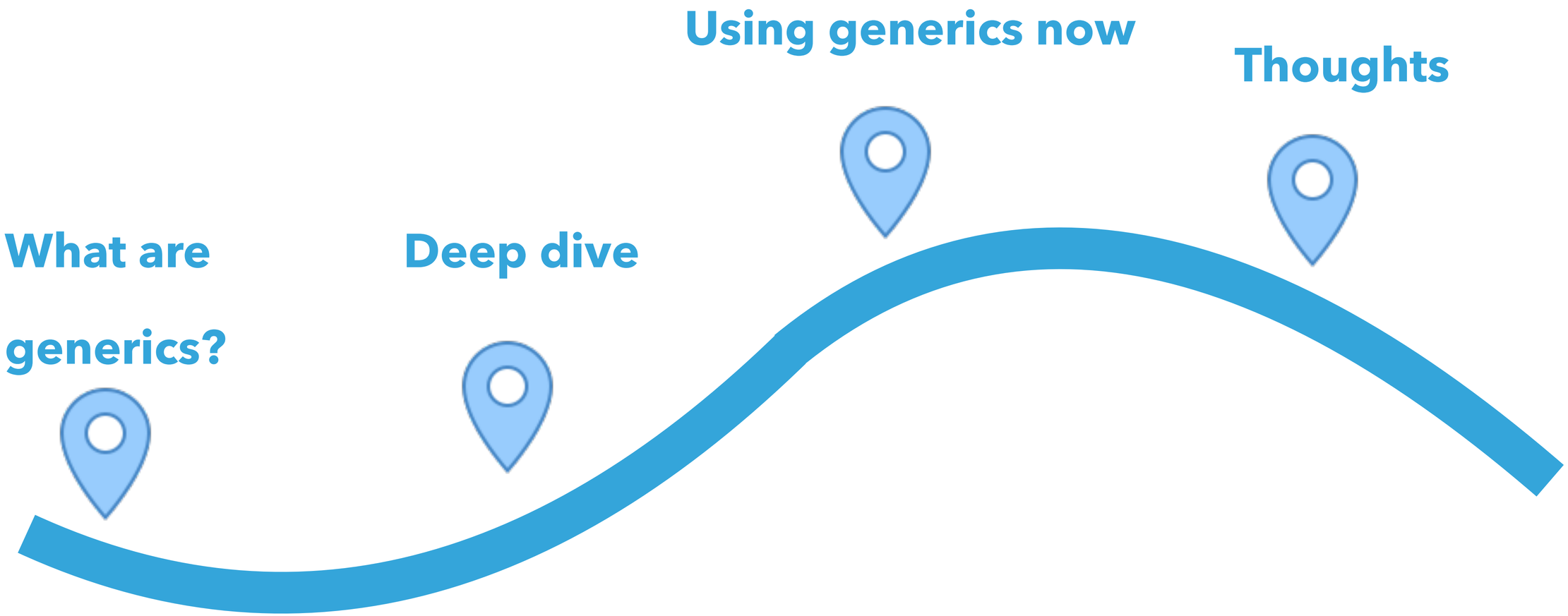
```
/** @template T of Animal */
interface AnimalProcessor {
  /** @return class-string<T> */
  public function supports(): string;

  …
```

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

process(“Bob”);
```

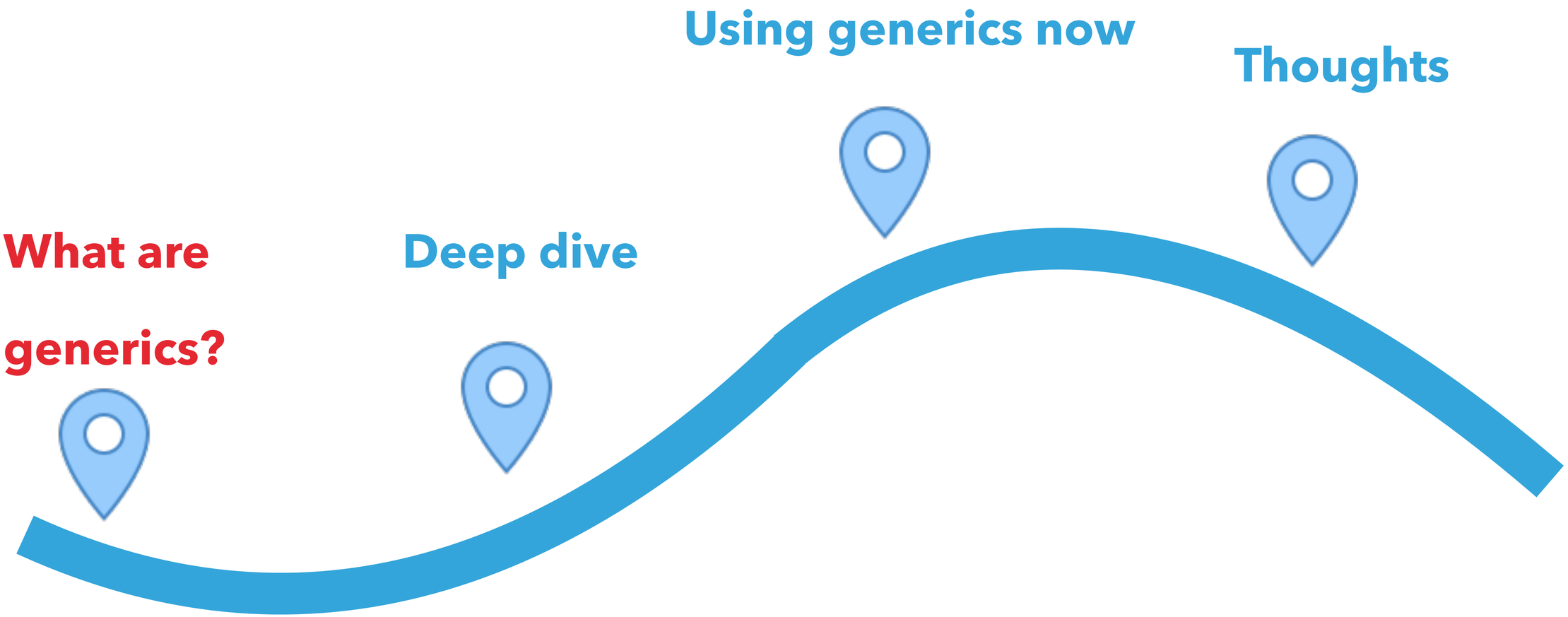
```
/** @template T of Animal */
interface AnimalProcessor {
  /** @return class-string<T> */
  public function supports(): string;

  …
```

Psalm

AGENDA



AGENDA



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

process(“Bob”);
```

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

process(“Bob”);
```

Clear, unambiguous type information

Run time check

Static analysis check

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

process(“Bob”);
```

✅ **Clear, unambiguous type information**

Run time check

Static analysis check

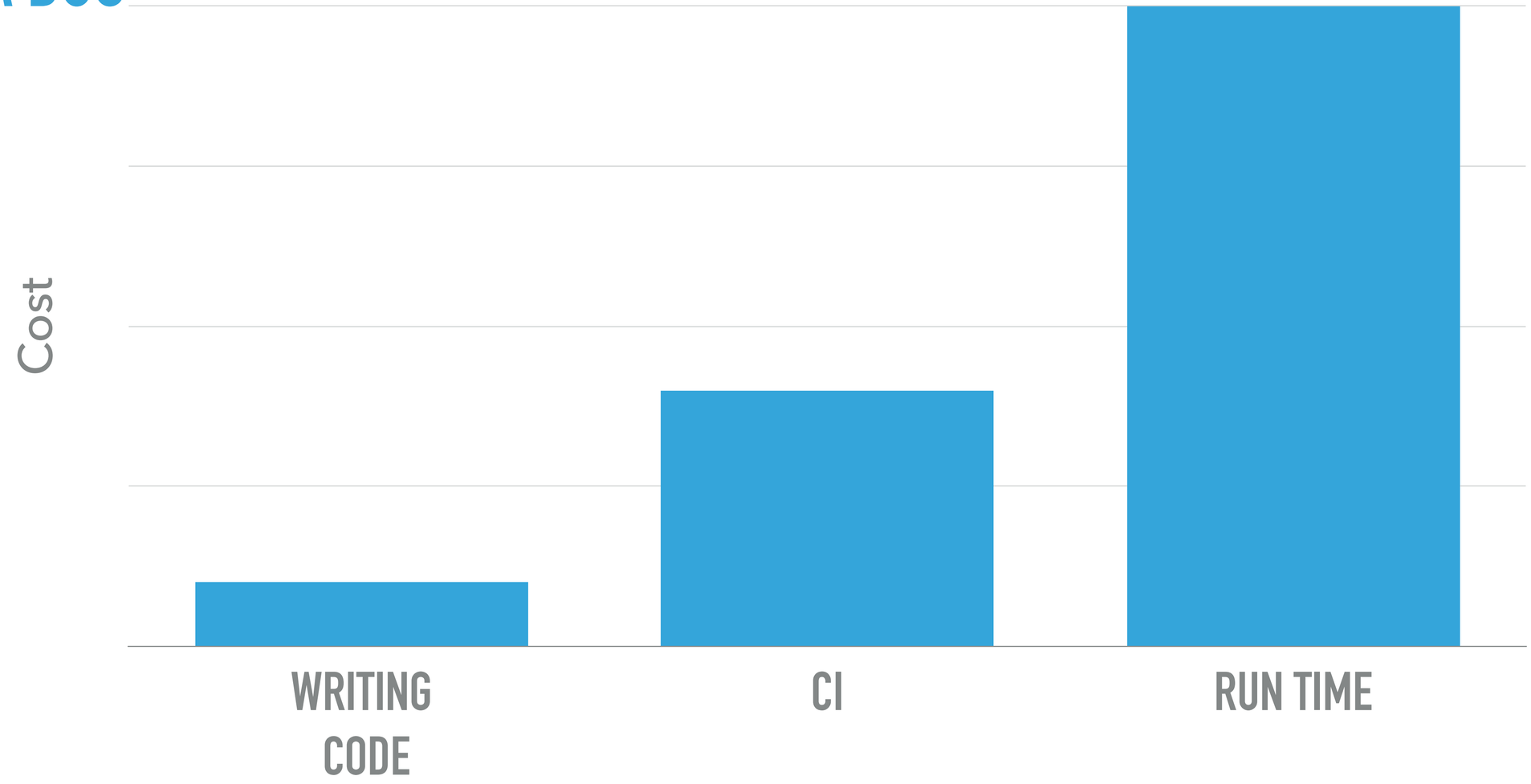
```
function process(User $user): void { … }

process(“Bob”);
```

- ✅ **Clear, unambiguous type information**
- ✅ **Run time check**
- **Static analysis check**

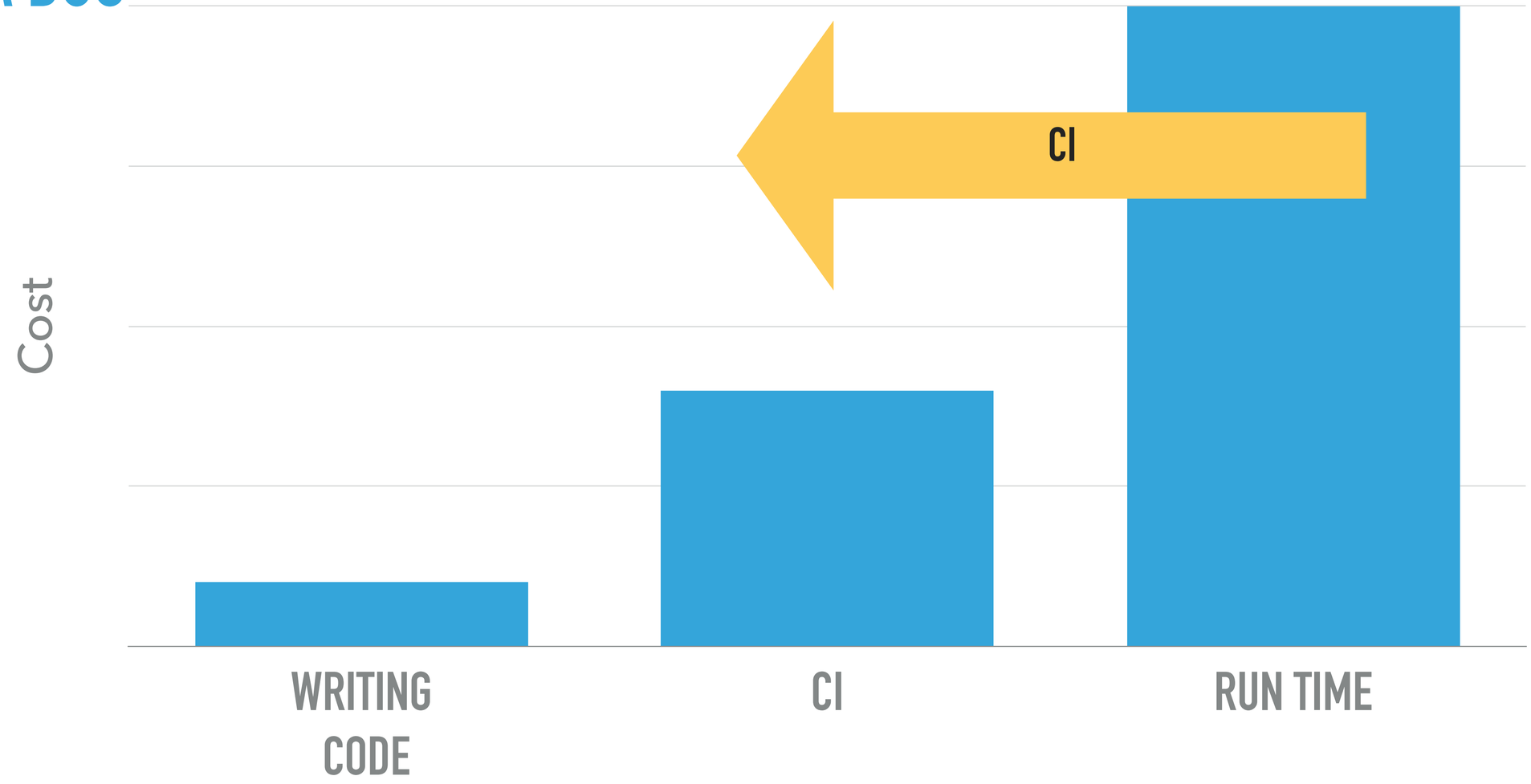
```
function process(User $user): void { … }

process(“Bob”);
```

- ✅ **Clear, unambiguous type information**
- ✅ **Run time check**
- ✅ **Static analysis check**

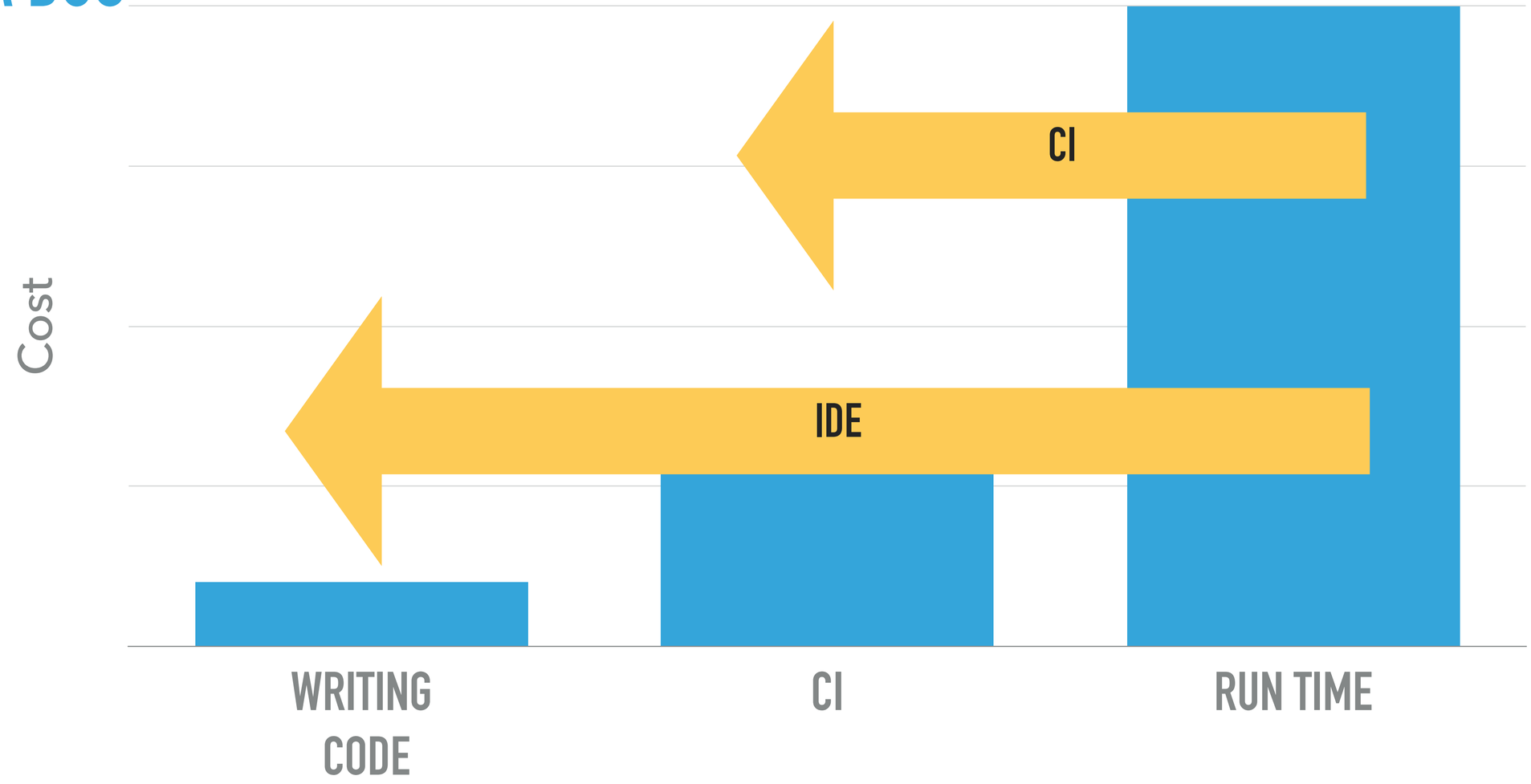
COST OF A BUG



COST OF A BUG



COST OF A BUG



```
class User {
    public function getName(): string {…}
}

class Company {
    public function getName(): string {…}
}
```

```
class User {
    public function getName(): string {…}
    public function getUserName(): string {…}
}

class Company {
    public function getName(): string {…}
}
```

```
function process($person, $retailer): void {
    $person->getName();
    $retailer->getName();
}
```

```
function process($person, $retailer): void {
    $person->getName();
    $retailer->getName();
}
```

```
function process($person, $retailer): void {
    $person->getName();
    $retailer->getName();
}
```

```
function process(User $person, Company $retailer): void
{
    $person->getName();

    $retailer->getName();
}
```

```
function process(User $person, Company $retailer): void
{
    $person->getName();

    $retailer->getName();
}
```

```
function process(User $person, Company $retailer): void
{
    $person->getName();
    $retailer->getName();
}
```

```
function process(User $person, Company $retailer): void
{
    $person->getName();
    $retailer->getName();
}
```

```
function process(User $person, Company $retailer): void
{
    $person->getName();

    $person->getUserName();

    $retailer->getName();
}
```

BENEFITS TYPES: REFACTORING

Clear, unambiguous type information

Run time check

Static analysis check

Safe refactoring

✅ **Clear, unambiguous type information**

Run time check

Static analysis check

Safe refactoring

✅ **Clear, unambiguous type information**

✅ **Run time check**

Static analysis check

Safe refactoring

- ✅ **Clear, unambiguous type information**
- ✅ **Run time check**
- ✅ **Static analysis check**
- **Safe refactoring**

- ✅ **Clear, unambiguous type information**
- ✅ **Run time check**
- ✅ **Static analysis check**
- ✅ **Safe refactoring**

Where are PHP's limitations?

```
class Queue {

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

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

```
class Queue {

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

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

```
function getQueue(): Queue { … }

$queue = getQueue();
```

```
function getQueue(): Queue { … }
$queue = getQueue();
```

Type of entities in the queue is known

Run time check

Static analysis check

```
function getQueue(): Queue { … }
$queue = getQueue();
```

Type of entities in the queue is known

Run time check

Static analysis check

```
function getQueue(): Queue { … }
$queue = getQueue();
```

❌ **Type of entities in the queue is known**

❌ **Run time check**

Static analysis check

```
function getQueue(): Queue { … }
$queue = getQueue();
```

Type of entities in the queue is known

Run time check

Static analysis check

```
class TypedQueue {
 private string $type;
 private Queue $queue;

 public function __construct(string $type) {
   $this->type = $type;
   $this->queue = new Queue();
 }
```

```
class TypedQueue {
  private string $type;
  private Queue $queue;

  public function __construct(string $type) {
    $this->type = $type;
    $this->queue = new Queue();
  }
```

```
class TypedQueue {
 private string $type;
 private Queue $queue;

 public function __construct(string $type) {
  $this->type = $type;
   $this->queue = new Queue();
 }
```

```
class TypedQueue {
 private string $type;
 private Queue $queue;

 public function __construct(string $type) {
   $this->type = $type;
   $this->queue = new Queue();
 }
```

```
public function add($item) {
   if (!$item instanceof $this->type) {
     throw new TypeError();
   }
   $this->queue->add($item);
 }
public function getNext() {
   return $this->queue->getNext();
}
```

```
public function add($item) {
    if (!$item instanceof $this->type) {
      throw new TypeError();
    }
    $this->queue->add($item);
  }
public function getNext() {
    return $this->queue->getNext();
}
```

```
public function add($item) {
    if (!$item instanceof $this->type) {
      throw new TypeError();
    }
    $this->queue->add($item);
  }
public function getNext() {
    return $this->queue->getNext();
}
```

```
public function add($item) {
    if (!$item instanceof $this->type) {
      throw new TypeError();
    }
    $this->queue->add($item);
  }
public function getNext() {
    return $this->queue->getNext();
}
```

```
public function add($item) {
   if (!$item instanceof $this->type) {
     throw new TypeError();
   }
   $this->queue->add($item);
 }
public function getNext() {
   return $this->queue->getNext();
}
```

```
$userQueue = new TypedQueue(User::class);
```

```
$userQueue = new TypedQueue(User::class);

$userQueue->add(new User(“Jane”)); ✅
```

```
$userQueue = new TypedQueue(User::class);

$userQueue->add(new User("Jane")); ✅

$userQueue->add("bob");            ❌
```

```
$personQueue->add(new User("bob"));
```

Same code works for any type

Run time check

Static analysis check

```
$personQueue->add(new User("bob"));
```

✅ **Same code works for any type**

Run time check

Static analysis check

```
$personQueue->add(new User("bob"));
```

✅ **Same code works for any type**

✅ **Run time check**

Static analysis check

```
$personQueue->add(new User("bob"));
```

✅ **Same code works for any type**

✅ **Run time check**

❌ **Static analysis check**

```
class UserQueue {
 private Queue $queue; // Setup in constructor
 public function add(User $item): void {
  $this->queue->add($item);
 }
 public function getNext(): User {
   return $this->queue->getNext();
 }
}
```

```
class UserQueue {
 private Queue $queue; // Setup in constructor
 public function add(User $item): void {
  $this->queue->add($item);
 }
 public function getNext(): User {
   return $this->queue->getNext();
 }
}
```

```
class UserQueue {
 private Queue $queue; // Setup in constructor
 public function add(User $item): void {
  $this->queue->add($item);
 }
 public function getNext(): User {
    return $this->queue->getNext();
 }
}
```

```
class UserQueue {
 private Queue $queue; // Setup in constructor
 public function add(User $item): void {
  $this->queue->add($item);
 }
 public function getNext(): User {
   return $this->queue->getNext();
 }
}
```

```
class UserQueue {
 private Queue $queue; // Setup in constructor
 public function add(User $item): void {
  $this->queue->add($item);
 }
 public function getNext(): User {
   return $this->queue->getNext();
 }
}
```

```
$userQueue = new UserQueue();
```

```
$userQueue = new UserQueue();

$userQueue->add(new User("Jane")); ✅

$userQueue->add("bob");            ❌
```

```
$personQueue->add(new Person("bob"));
```

Same code works for any type

Run time check

Static analysis check

```
$personQueue->add(new Person("bob"));
```

Same code works for any type

Run time check

Static analysis check

```
$personQueue->add(new Person("bob"));
```

❌ **Same code works for any type**

✅ **Run time check**

Static analysis check

```
$personQueue->add(new Person("bob"));
```

❌ **Same code works for any type**

✅ **Run time check**

✅ **Static analysis check**

```
class Queue    {

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

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

```
class Queue <T> {

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

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

```
class Queue <T> {

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

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

```
class Queue <T> {

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

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

```
$userQueue = new Queue();
```

```
$userQueue = new Queue<User>();
```

```
$userQueue = new Queue<User>();

$userQueue->add(new User("Alice")); ✅

$userQueue->add("bob"); ❌
```

```
$personQueue->add(new Person("bob"));
```

Same code works for any type

Run time check

Static analysis check

```
$personQueue->add(new Person("bob"));
```

✅ **Same code works for any type**

Run time check

Static analysis check

```
$personQueue->add(new Person("bob"));
```

✅ **Same code works for any type**

✅ **Run time check**

Static analysis check

```
$personQueue->add(new Person("bob"));
```

- ✅ **Same code works for any type**
- ✅ **Run time check**
- ✅ **Static analysis check**

```
$userQueue = new TypedQueue(User::class);

$userQueue = new UserQueue();

$userQueue = new Queue<User>();
```

DEJA VU?

```
/** @return User[] */
function getUsers(): array;


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


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

```
/** @return User[] */
function getUsers(): array;


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


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

```
/** @return User[] */
function getUsers(): array;

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

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

```
/** @return User[] */
function getUsers(): array;

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

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

```
/** @return User[] */
function getUsers(): array;
```

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

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

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

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

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

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

Psalm

PHPStan

```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>.

```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>.

HOW DO WE TELL STATIC ANALYSERS ABOUT GENERICS?

```
class Queue <T> {

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

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

HOW DO WE TELL STATIC ANALYSERS ABOUT GENERICS?

```
/** @template T */
class Queue    {


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


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

```
/** @template T */
class Queue    {

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


  public function getNext(): T{…}

}
```

```
/** @template T */
class Queue    {

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

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

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

RECAP

```
class Queue() { … }
```

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

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


$userQueue = new Queue();
```

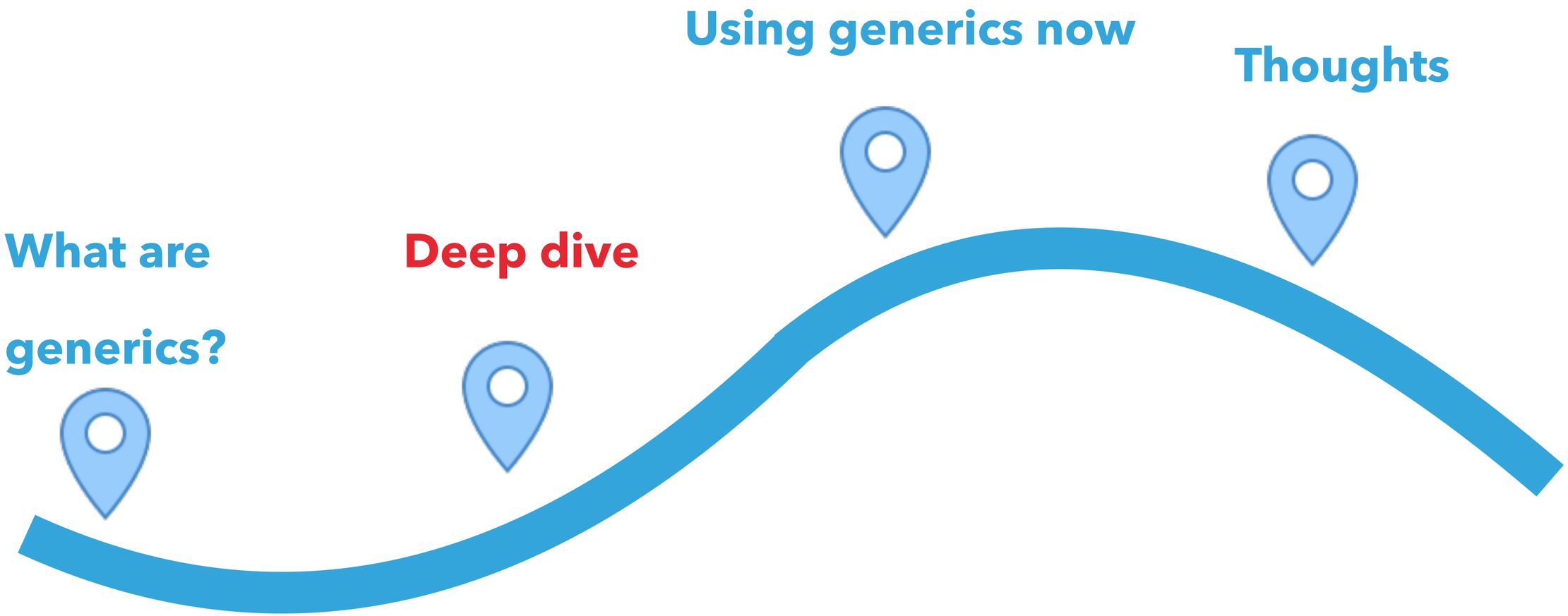
```
/** @template T */
class Queue() { … }

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

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

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

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

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

Psalm

AGENDA



COLLECTIONS

```
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);
}
```

```
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);
}
```

```
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);
}
```

```
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);
}
```

```
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);
}
```

```
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);
}
```

```
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);
}
```

```
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);
}
```

```
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);
}
```

```
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> */
$people = [ … ];
```

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

```
/** @var ArrayCollection<K, V> */
$people = new ArrayCollection();
```

FUNCTIONS

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

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

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

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

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

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

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

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

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

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

```
/**
 * @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);
```

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

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

CLASS STRING

```
App\Entities\Person
```

```
Person::class
```

```
class Person {…}

class DIContainer
{
 /**
  *
  * @param string $className
  * @return object
  */
 public function make(string $className): object {…}

}

$person = $this->diContainer->make(Person::class);
```

```
class Person {…}

class DIContainer
{
 /**
  *
  * @param string $className
  * @return object
  */
 public function make(string $className): object {…}

}

$person = $this->diContainer->make(Person::class);
```

```
class Person {…}

class DIContainer
{
    /**
     *
     * @param string $className
     * @return object
     */
    public function make(string $className): object {…}
}

$person = $this->diContainer->make(Person::class);
```

```
class Person {…}

class DIContainer
{
 /**
  *
  * @param string $className
  * @return object
  */
 public function make(string $className): object {…}

}
```

```
$person = $this->diContainer->make(Person::class);
```

```
class Person {…}

class DIContainer
{
 /**
  * @template T
  * @param class-string<T> $className
  * @return T
  */
 public function make(string $className): object {…}

}

$person = $this->diContainer->make(Person::class);
```

```
class Person {…}

class DIContainer
{
 /**
  * @template T
  * @param class-string<T> $className
  * @return T
  */
 public function make(string $className): object {…}

}

$person = $this->diContainer->make(Person::class);
```

```
class Person {…}

class DIContainer
{
 /**
  * @template T
  * @param class-string<T> $className
  * @return T
  */
 public function make(string $className): object {…}

}

$person = $this->diContainer->make(Person::class);
```

```
class Person {…}

class DIContainer
{
 /**
  * @template T
  * @param class-string<T> $className
  * @return T
  */
 public function make(string $className): object {…}

}

$person = $this->diContainer->make(Person::class);
```

```
class Person {…}

class DIContainer
{
 /**
  * @template T
  * @param class-string<T> $className
  * @return T
  */
 public function make(string $className): object {…}

}
```

```
$person = $this->diContainer->make(Person::class);
```

```
class Person {…}

class DIContainer
{
 /**
  * @template T
  * @param class-string<T> $className
  * @return T
  */
 public function make(string $className): object {…}

}

$person = $this->diContainer->make(Person::class);
```

EXTENDING TEMPLATES

```
/** @template T */
abstract class Repository {

  /** @return array<T> */
  public function findAll(): array {…}

  /** @return T|null */
  public function findById(int $id) {…}
}
```

```
/** @template T */
abstract class Repository {

    /** @return array<T> */
    public function findAll(): array {…}

    /** @return T|null */
    public function findById(int $id) {…}
}
```

```
/** @template T */
abstract class Repository {

    /** @return array<T> */
    public function findAll(): array {…}

    /** @return T|null */
    public function findById(int $id) {…}
}
```

```
/** @template T */
abstract class Repository {

  /** @return array<T> */
  public function findAll(): array {…}

  /** @return T|null */
  public function findById(int $id) {…}
}
```

```
/** @template T */
abstract class Repository { … }

/** @extends Repository<User> */
class UserRepository extends Repository {…}

$user = $userRepository->findById(1);
```

```
/** @template T */
abstract class Repository { … }
```

```
/** @extends Repository<User> */
class UserRepository extends Repository {…}
```

```
$user = $userRepository->findById(1);
```

```
/** @template T */
abstract class Repository { … }

/** @extends Repository<User> */
class UserRepository extends Repository {…}

$user = $userRepository->findById(1);
```

```
/** @template T */
abstract class Repository { … }

/** @extends Repository<User> */
class UserRepository extends Repository {…}

$user = $userRepository->findById(1);
```

```
/** @template T */
abstract class Repository { … }

/** @extends Repository<User> */
class UserRepository extends Repository {…}

$user = $userRepository->findById(1);
```

RESTRICTING TYPES

```
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;
}
```

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

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

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

    /** @param T $animal */
    public function play($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->bark(); // We know $animal is a dog
  }
}
```

```
/** @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->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 {

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

```
/** @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 { … }
```

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

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

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

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

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

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

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

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

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

❌

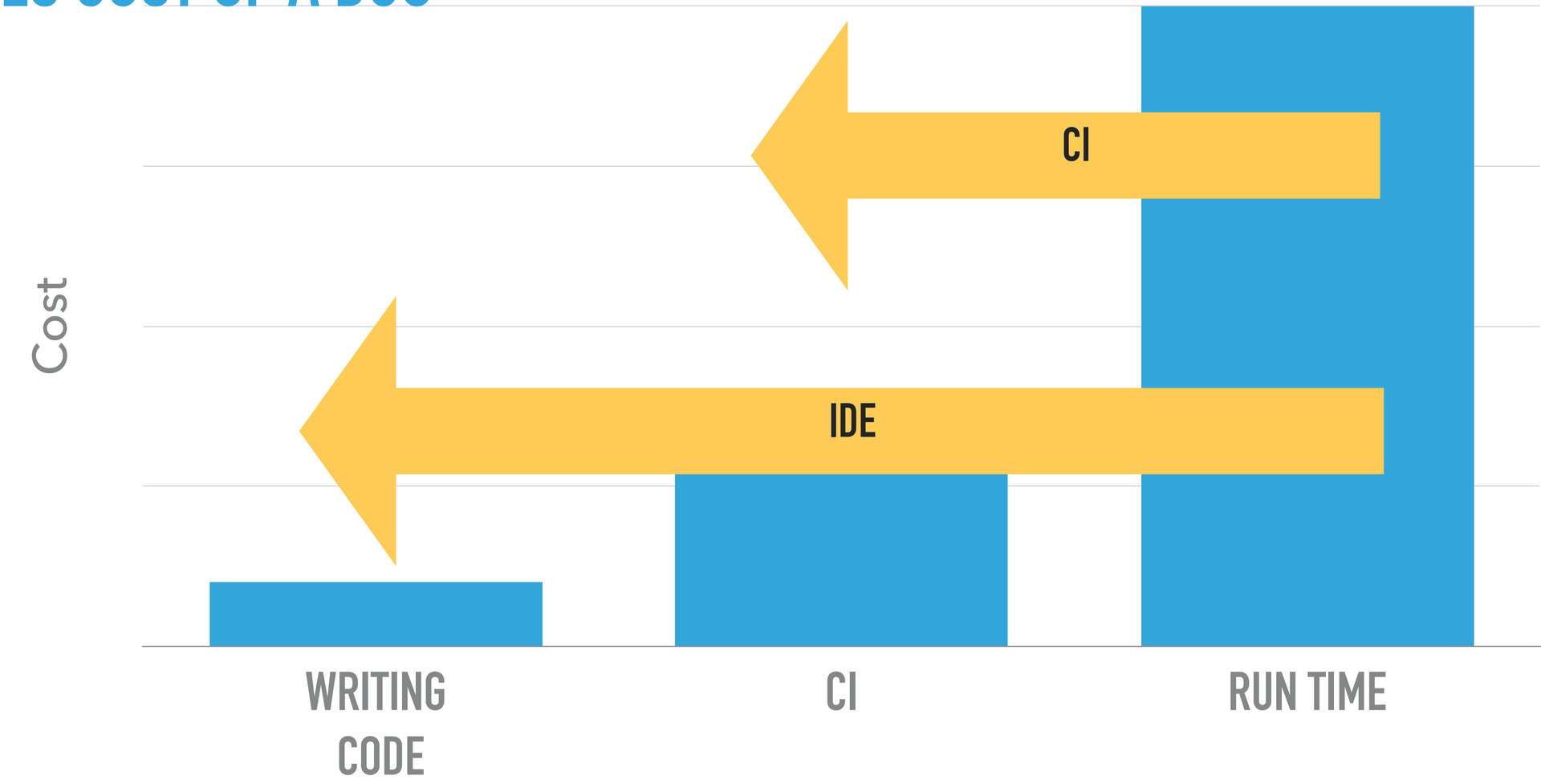
✅

HOW DOES THIS HELP US?

1. COMMUNICATES ADDITIONAL TYPE INFORMATION

```
/** @param array<string,Translation> $translations */
function storeTranslations(array $translations): void;
```

2. REDUCES COST OF A BUG

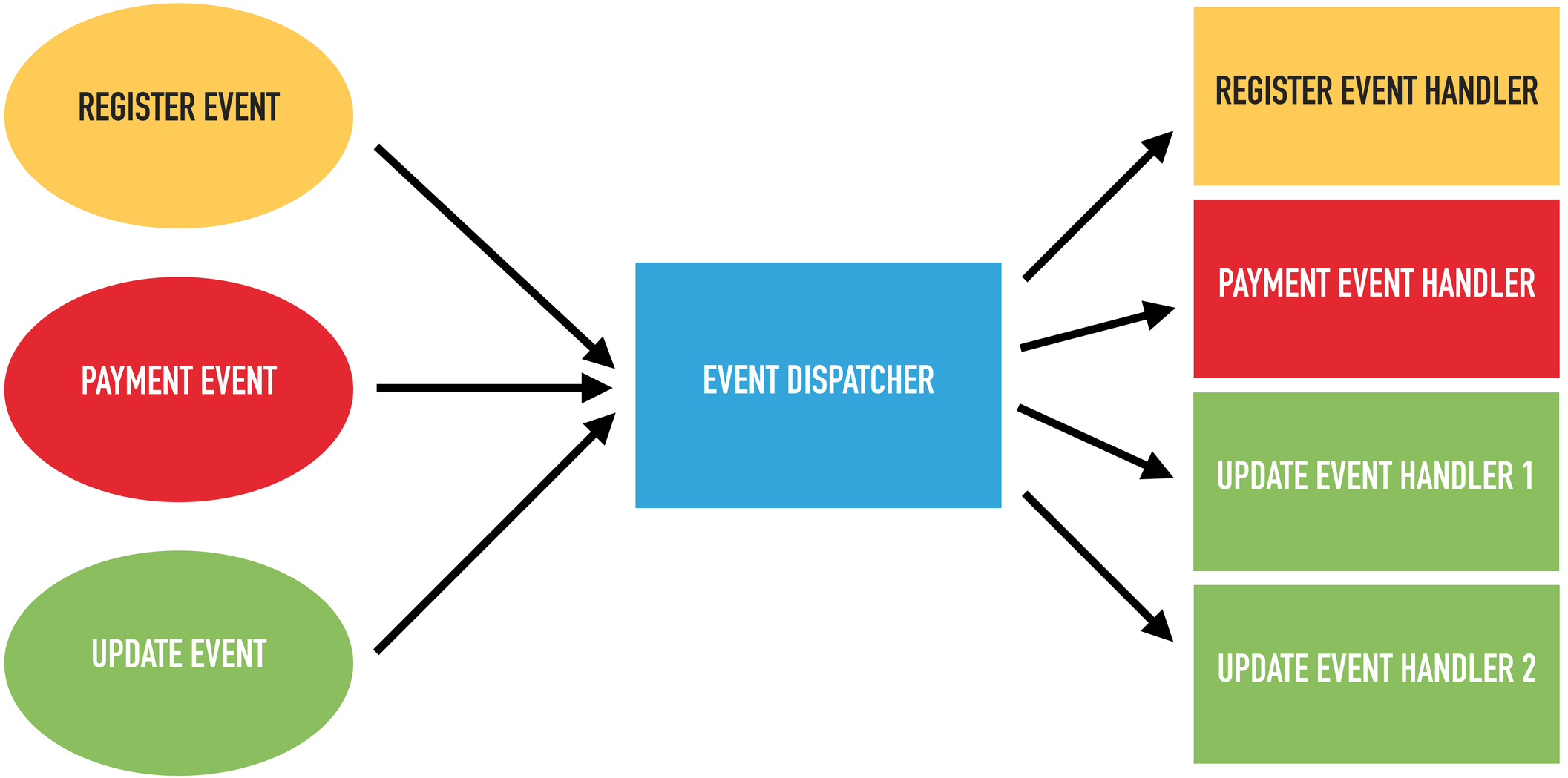


3. ENABLES REFACTORING

```
/** @param array<User> $users */
function processUsers(array $users): void {
  foreach($users as $user) {
    doSomething($user->getName);
    doSomething($user->getUserName());
  }
}
```

HOW ABOUT A REAL EXAMPLE?

A REAL EXAMPLE



```
interface Event {…}


final class RegisterEvent implements Event {
    public function name(): string {…}
}


final class PaymentEvent implements Event {
    public function amount(): int {…}
}
```

```
interface Event {…}

final class RegisterEvent implements Event {
    public function name(): string {…}
}

final class PaymentEvent implements Event {
    public function amount(): int {…}
}
```

```
interface Event {…}

final class RegisterEvent implements Event {
    public function name(): string {…}
}

final class PaymentEvent implements Event {
    public function amount(): int {…}
}
```

```
interface Event {…}


final class RegisterEvent implements Event {
    public function name(): string {…}
}


final class PaymentEvent implements Event {
    public function amount(): int {…}
}
```

```
/** @template E of Event */
interface EventHandler {
    /** @return class-string<E> */
    public function supports(): string;

    /** @param E $event */
    public function handle(Event $event): void;
}
```

```
/** @template E of Event */
interface EventHandler {
    /** @return class-string<E> */
    public function supports(): string;

    /** @param E $event */
    public function handle(Event $event): void;
}
```

```
/** @template E of Event */
interface EventHandler {
    /** @return class-string<E> */
    public function supports(): string;

    /** @param E $event */
    public function handle(Event $event): void;
}
```

```
/** @template E of Event */
interface EventHandler {
    /** @return class-string<E> */
    public function supports(): string;

    /** @param E $event */
    public function handle(Event $event): void;
}
```

```
/** @template E of Event */
interface EventHandler {
    /** @return class-string<E> */
    public function supports(): string;

    /** @param E $event */
    public function handle(Event $event): void;
}
```

```
/** @implements EventHandler<RegisterEvent> */
class RegisterEventHandler implements EventHandler {
```

```
/** @implements EventHandler<RegisterEvent> */
class RegisterEventHandler implements EventHandler {
```

```
/** @implements EventHandler<RegisterEvent> */
class RegisterEventHandler implements EventHandler {
```

```
/** @implements EventHandler<RegisterEvent> */
class RegisterEventHandler implements EventHandler {
```

```
/** @template E of Event */
interface EventHandler {
  … methods …
}
```

A REAL EXAMPLE

```
/** @implements EventHandler<RegisterEvent> */

class RegisterEventHandler implements EventHandler {
    public function supports(): string

    {

        return RegisterEvent::class;

    }
```

```
/** @implements EventHandler<RegisterEvent> */
class RegisterEventHandler implements EventHandler {
    public function supports(): string
    {
        return RegisterEvent::class;
    }
```

A REAL EXAMPLE

```
/** @implements EventHandler<RegisterEvent> */

class RegisterEventHandler implements EventHandler {
  public function supports(): string

  {
    return RegisterEvent::class;
  }
```

```
/** @implements EventHandler<RegisterEvent> */

class RegisterEventHandler implements EventHandler {
  public function supports(): string

  {
    return RegisterEvent::class;
  }
```

```
/** @template E of Event */
interface EventHandler {
  /** @return class-string<E> */
  public function supports(): string;
```

```
/** @implements EventHandler<RegisterEvent> */
class RegisterEventHandler implements EventHandler {
 public function handleEvent(Event $event): void
 {
    $event->name();
 }
```

```
/** @implements EventHandler<RegisterEvent> */
class RegisterEventHandler implements EventHandler {
 public function handleEvent(Event $event): void
 {
     $event->name();
 }
```

```
/** @implements EventHandler<RegisterEvent> */
class RegisterEventHandler implements EventHandler {
 public function handleEvent(Event $event): void
 {
    $event->name();
 }
```

```
/** @implements EventHandler<RegisterEvent> */
class RegisterEventHandler implements EventHandler {
 public function handleEvent(Event $event): void
 {
    $event->name();
 }
        final class RegisterEvent implements Event {
            public function name(): string {…}
        }
```

```
final readonly class EventDispatcher {


 /** @param array<EventHandler<Event>> $eventHandlers */
 public function __construct(private array $eventHandlers) {}


 public function handle(Event $event): void {
     foreach($this->eventProcessors as $eventProcessor) {
        if (get_class($event) === $eventProcessor->supports()) {
           $eventProcessor->process($event);
        }
     }
 }
```

```
final readonly class EventDispatcher {

    /** @param array<EventHandler<Event>> $eventHandlers */
    public function __construct(private array $eventHandlers) {}

    public function handle(Event $event): void {
        foreach($this->eventProcessors as $eventProcessor) {
            if (get_class($event) === $eventProcessor->supports()) {
                $eventProcessor->process($event);
            }
        }
    }
```

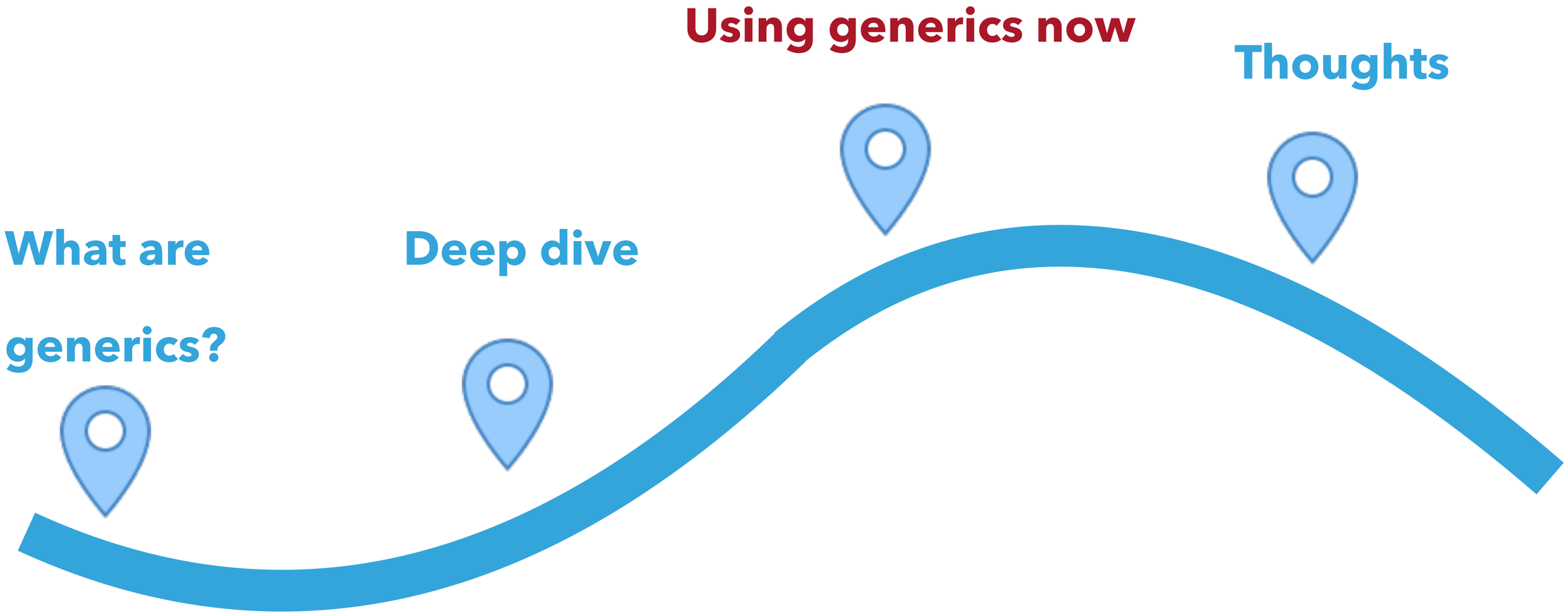
```
final readonly class EventDispatcher {

    /** @param array<EventHandler<Event>> $eventHandlers */
    public function __construct(private array $eventHandlers) {}

    public function handle(Event $event): void {
        foreach($this->eventProcessors as $eventProcessor) {
            if (get_class($event) === $eventProcessor->supports()) {
                $eventProcessor->process($event);
            }
        }
    }
```

```
final readonly class EventDispatcher {


 /** @param array<EventHandler<Event>> $eventHandlers */
 public function __construct(private array $eventHandlers) {}


 public function handle(Event $event): void {
     foreach($this->eventProcessors as $eventProcessor) {
        if (get_class($event) === $eventProcessor->supports()) {
           $eventProcessor->process($event);
        }
     }
 }
```

Using generics can help us write more understandable, robust and reliable code.

Demonstrate how existing tools can (almost) give us the benefits of generics now.

AGENDA



USING GENERICS NOW

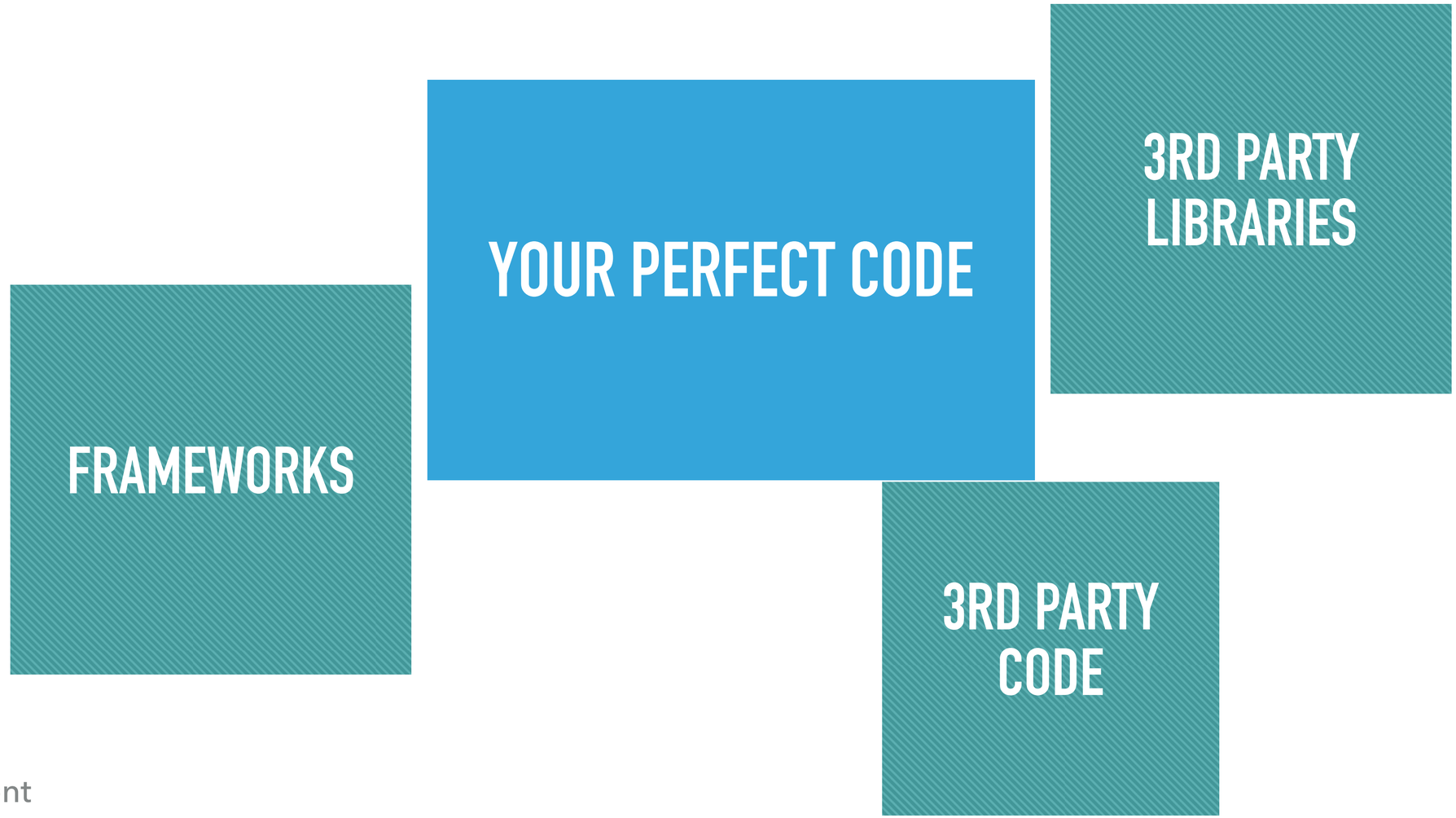
USING GENERICS NOW

Provide type information for everything including generics

Psalm

YOUR PERFECT CODE

INTEGRATING WITH 3RD PARTY CODE



GET THIRD PARTY LIBRARIES ON BOARD

- E.g. Doctrine, PHPUnit, Webmozart Assertion
- Engage with maintainers
- 2 steps
  - Adding additional annotations
  - Introduce static analysers to build process

ADAPTORS FOR 3RD PARTY LIBRARIES: PROBLEM

```
class Hasher {

    public function encode():string
    {…}
}

$hash = $this->hasher->encode($id);
```

ADAPTORS FOR 3RD PARTY LIBRARIES: PROBLEM

```
class Hasher {

    public function encode():string
    {…}
}

$hash = $this->hasher->encode($id);
```

ADAPTORS FOR 3RD PARTY LIBRARIES: PROBLEM

```
class Hasher {

    public function encode():string
    {…}
}

$hash = $this->hasher->encode($id);
```

```
class CleanHasher {

  public function __construct(private Hasher $hasher){}

  public function encode(int $id): string {
      return $this->hasher->encode($id);
  }
}


… in our code …
$hash = $this->cleanHasher->encode($id);
```

```
class CleanHasher {

    public function __construct(private Hasher $hasher){}

    public function encode(int $id): string {
        return $this->hasher->encode($id);
    }
}


… in our code …
$hash = $this->cleanHasher->encode($id);
```

```
class CleanHasher {

  public function __construct(private Hasher $hasher){}

  public function encode(int $id): string {
      return $this->hasher->encode($id);
  }
}


… in our code …
$hash = $this->cleanHasher->encode($id);
```

```
class CleanHasher {

  public function __construct(private Hasher $hasher){}

  public function encode(int $id): string {
      return $this->hasher->encode($id);
  }
}


… in our code …
$hash = $this->cleanHasher->encode($id);
```

```
class CleanHasher {

  public function __construct(private Hasher $hasher){}

  public function encode(int $id): string {
      return $this->hasher->encode($id);
  }
}


… in our code …
$hash = $this->cleanHasher->encode($id);
```

```
class CleanHasher {

   public function __construct(private Hasher $hasher){}

   public function encode(int $id): string {
       return $this->hasher->encode($id);
   }
}


… in our code …
$hash = $this->cleanHasher->encode($id);
```

```
class CleanHasher {

   public function __construct(private Hasher $hasher){}

   public function encode(int $id): string {
        return $this->hasher->encode($id);
   }
}


… in our code …
$hash = $this->cleanHasher->encode($id);
```

```
class CleanHasher {

   public function __construct(private Hasher $hasher){}

   public function encode(int $id): string {
        return $this->hasher->encode($id);
   }
}


… in our code …
$hash = $this->cleanHasher->encode($id);
```

USING STUBS

```
namespace ThirdParty\DI;

class DependencyInjection
{

 public function make(string $className): object
 {…}

}
```

Stubs/ThirdParty/DI.php.stub

Stubs/ThirdParty/DI.php.stub

```
namespace ThirdParty\DI;

class DependencyInjection
{
 /**
  * @template T
  * @param class-string<T> $className
  * @return T
  */
 public function make(string $className): object;
}
```

Stubs/ThirdParty/DI.php.stub

```
namespace ThirdParty\DI;

class DependencyInjection
{
 /**
  * @template T
  * @param class-string<T> $className
  * @return T
  */
 public function make(string $className): object;
}
```

Stubs/ThirdParty/DI.php.stub

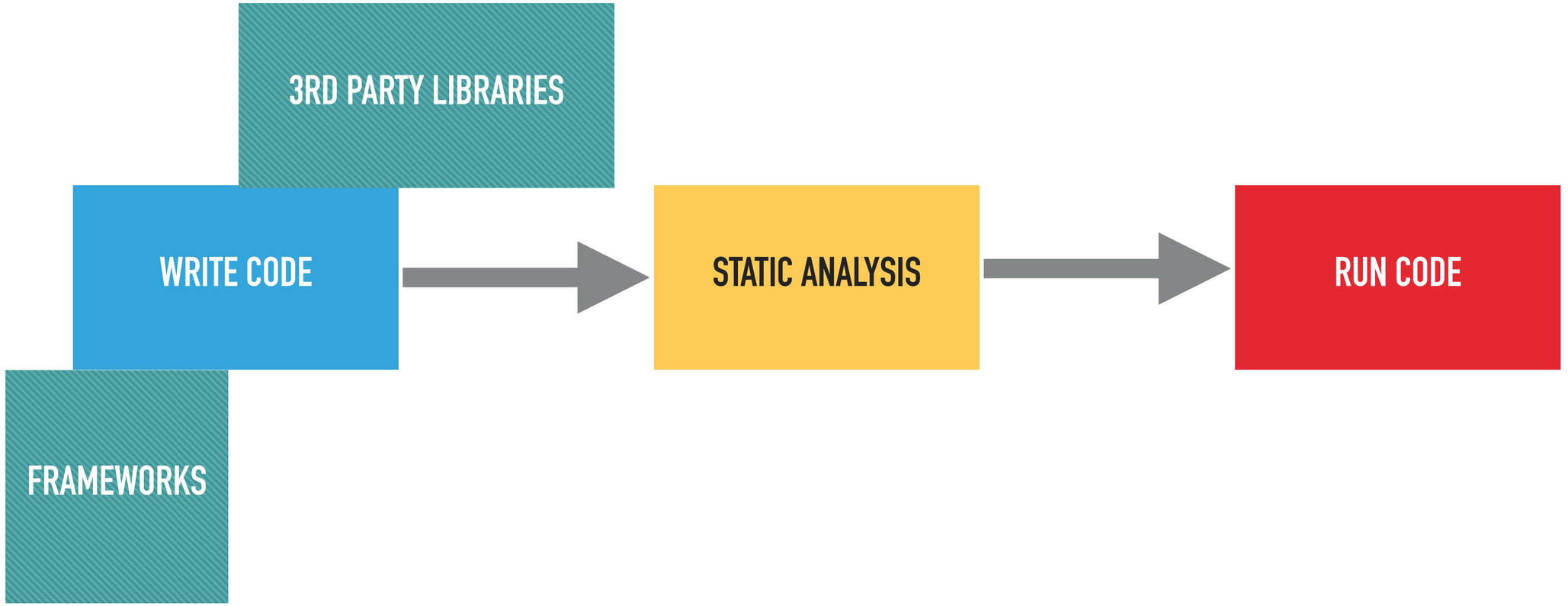
```
namespace ThirdParty\DI;

class DependencyInjection
{
 /**
  * @template T
  * @param class-string<T> $className
  * @return T
  */
 public function make(string $className): object;
}
```

STATIC ANALYSER PLUGINS

- Needed where lots of “magic” is going on
- Specific to static analysis tool
- Harder to write

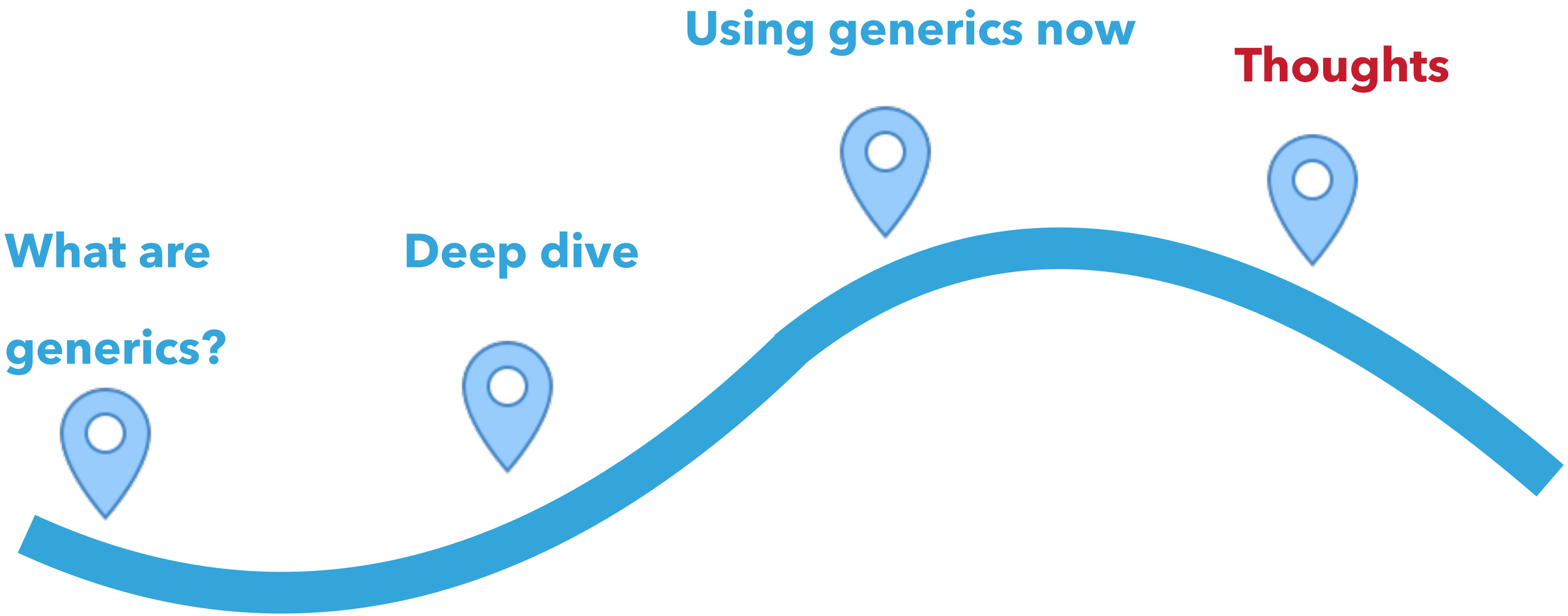
USING GENERICS NOW

Static analyser needs to know the types of everything



Static analyser needs to know the types of everything

AGENDA



PHP GENERICS TODAY (ALMOST)

PHP GENERICS TODAY (ALMOST)

IMPLEMENTING A STANDARD

IMPLEMENTING A STANDARD

- Full language support

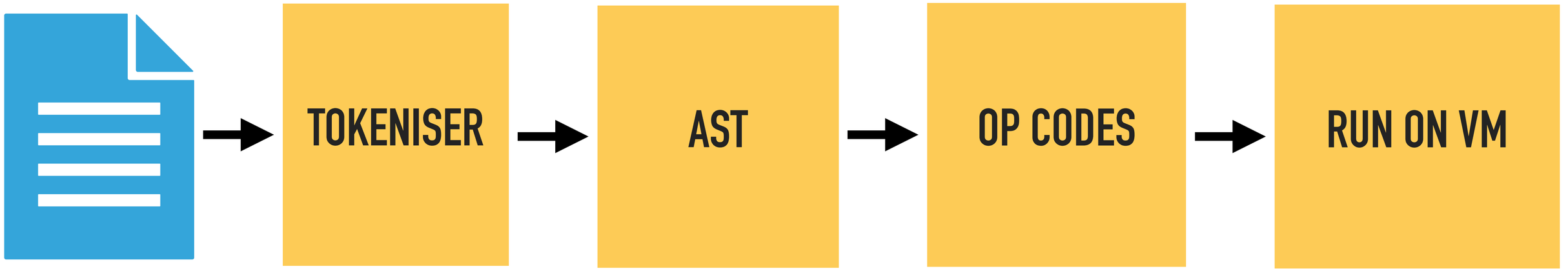
IMPLEMENTING A STANDARD

- Full language support
- Partial language support
  - Valid syntax
  - Ignored at run time

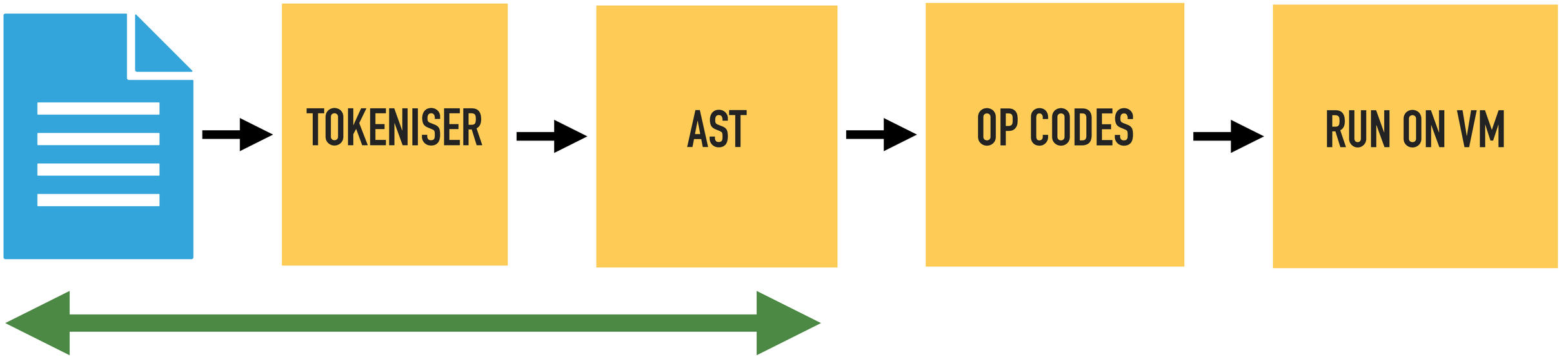
IMPLEMENTING A STANDARD

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

PARTIAL LANGUAGE SUPPORT

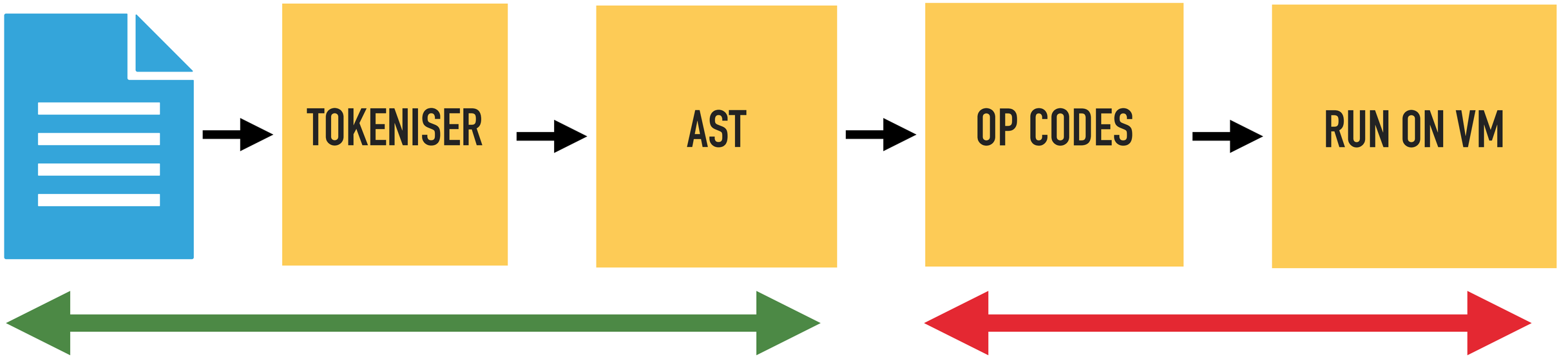


PARTIAL LANGUAGE SUPPORT

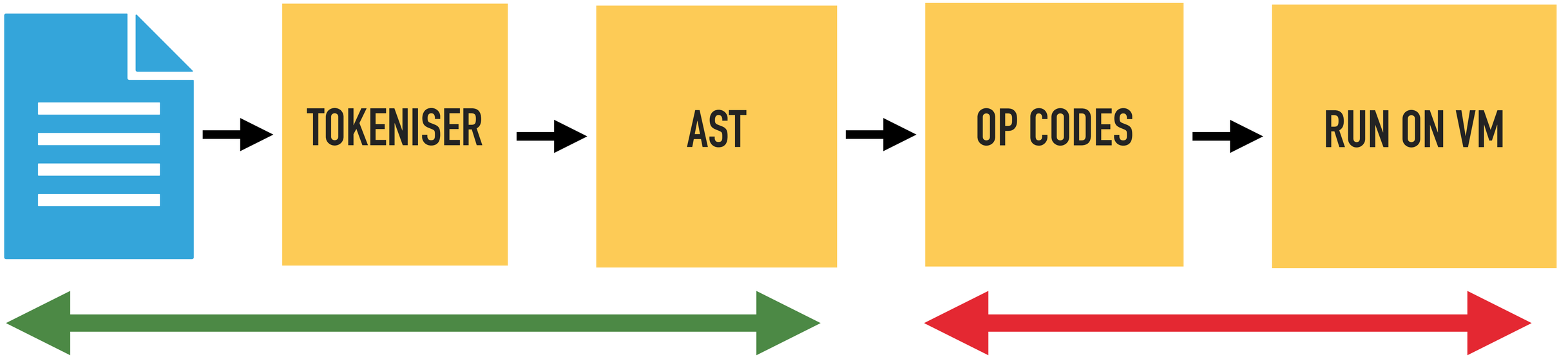


Understanding of generics

PARTIAL LANGUAGE SUPPORT



PARTIAL LANGUAGE SUPPORT



Validated by Static Analysis, not at run time.

HOW ABOUT #<>

```
function getPeople(): array#<int, Person>(){
    // Some code
}
```

https://gist.github.com/DaveLiddament/40130a7a107478bf6f92fcbb0b01a2fc

```
class Queue #<T of object>
{
  private array#<int,T> $queue = [];

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

  public function next(): #<T> {…}
}

$personQueue = new Queue#<Person>();

interface Repository #<T> {…}

class PersonRepository implements Repository#<Person> {…}
```

```
class Queue #<T of object>
{
  private array#<int,T> $queue = [];

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

  public function next(): #<T> {…}
}

$personQueue = new Queue#<Person>();

interface Repository #<T> {…}

class PersonRepository implements Repository#<Person> {…}
```

```
//
```

```
/**
 *
 */
```

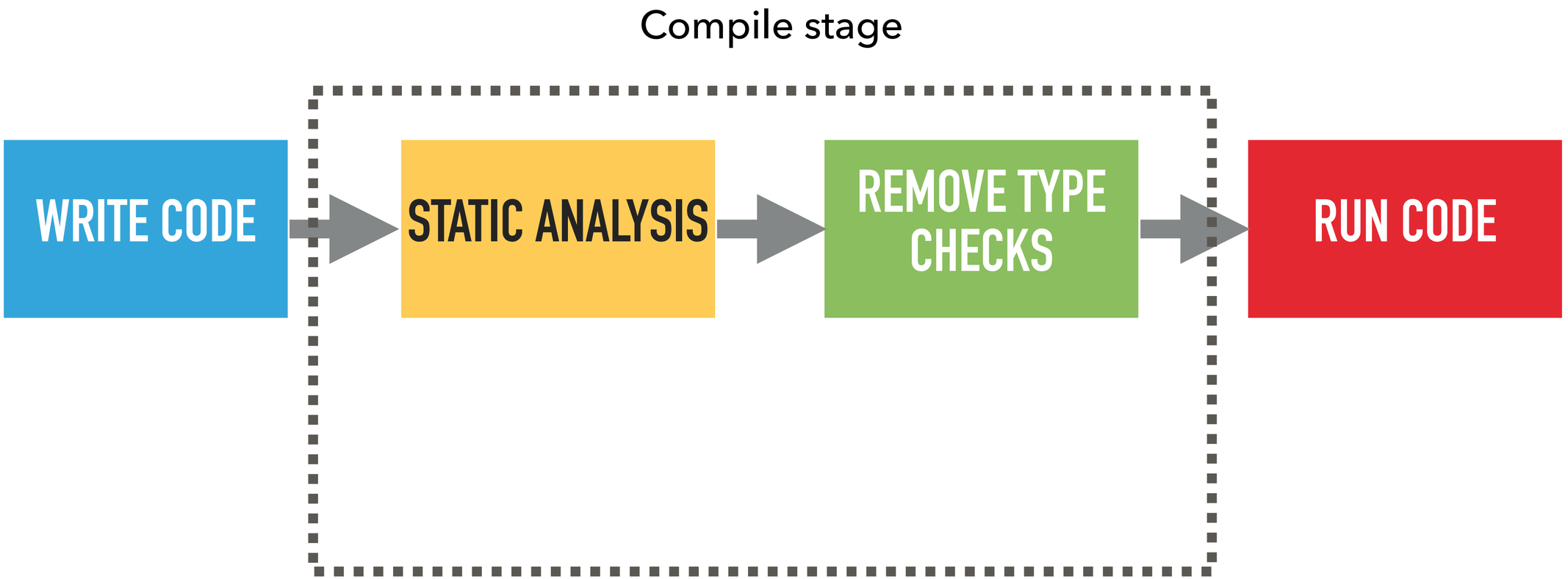
HUMANS

```
#[]
#<>
```

HUMANS + COMPUTERS

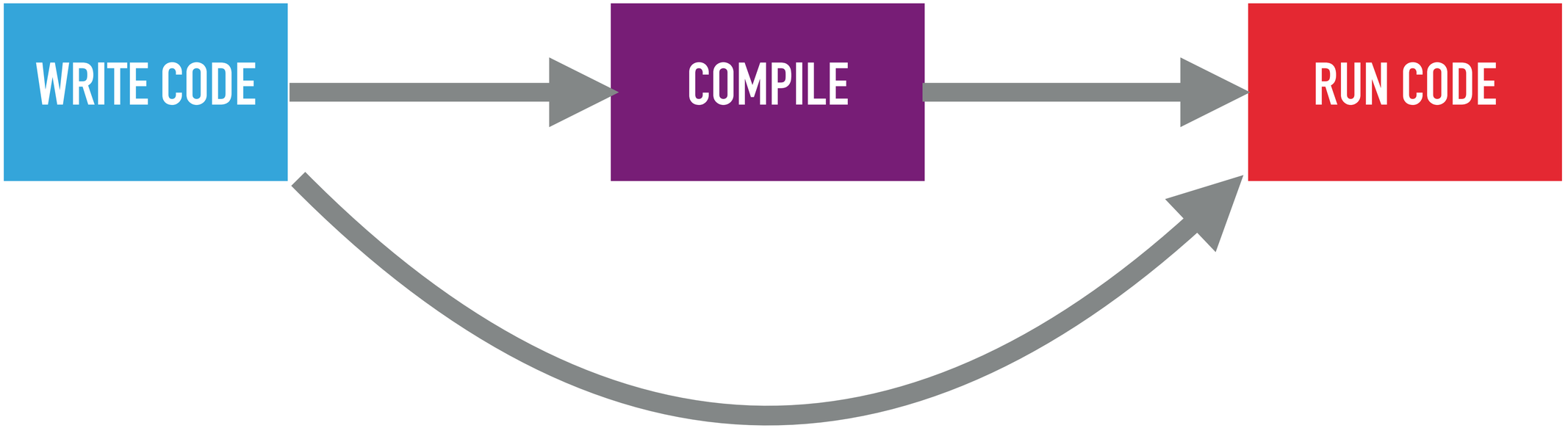
https://wiki.php.net/rfc/bound_erased_generic_types

THE END OF RUN TIME CHECKS?

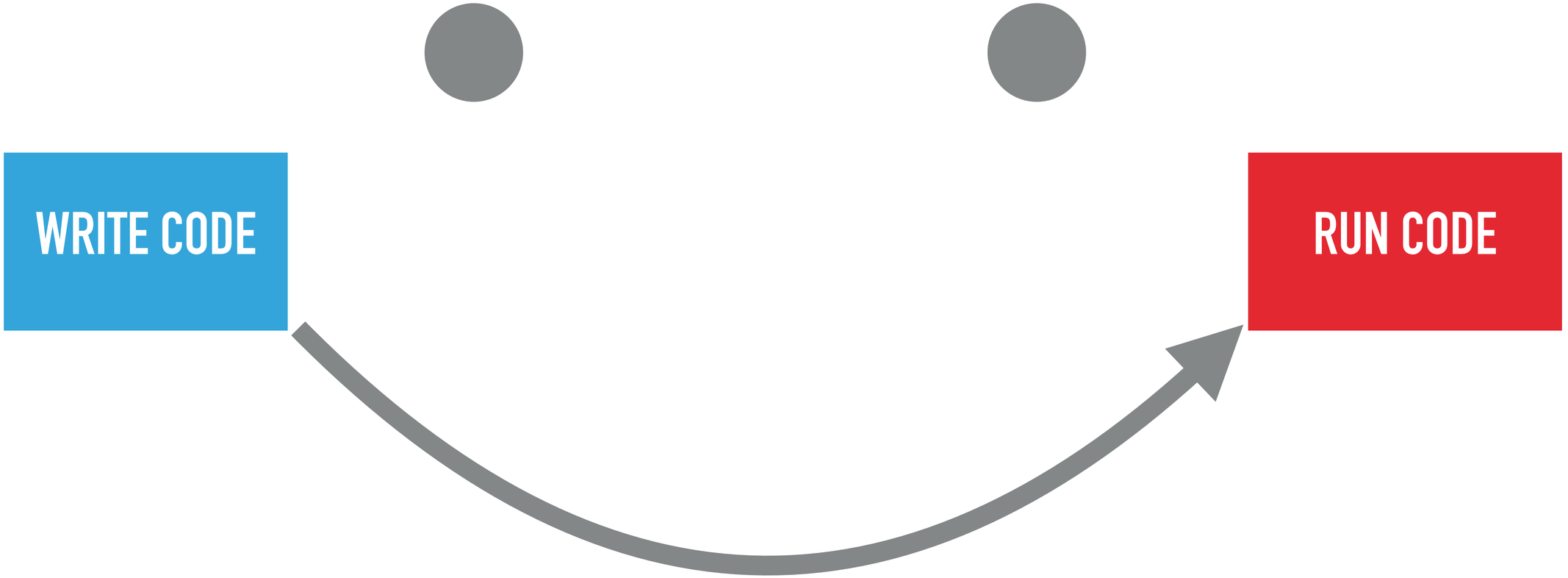


WHY NOT JUST USE JAVA?

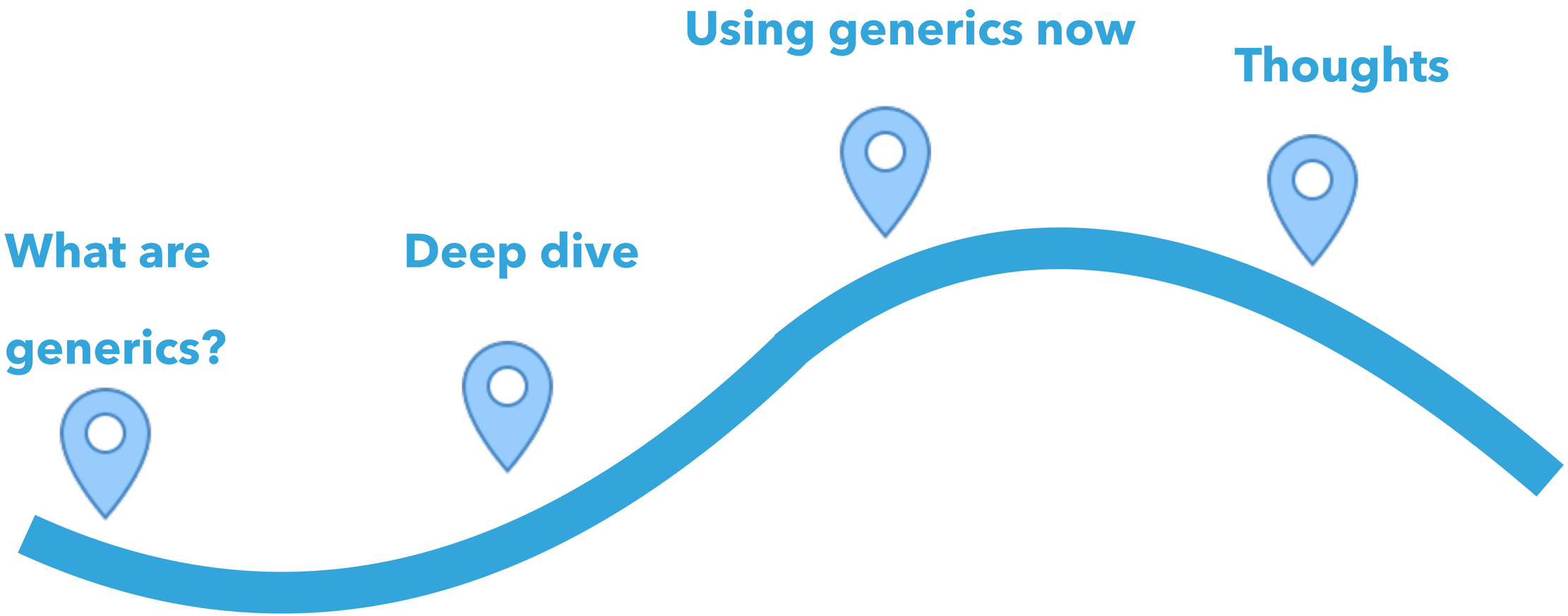
WHY NOT JUST USE JAVA?



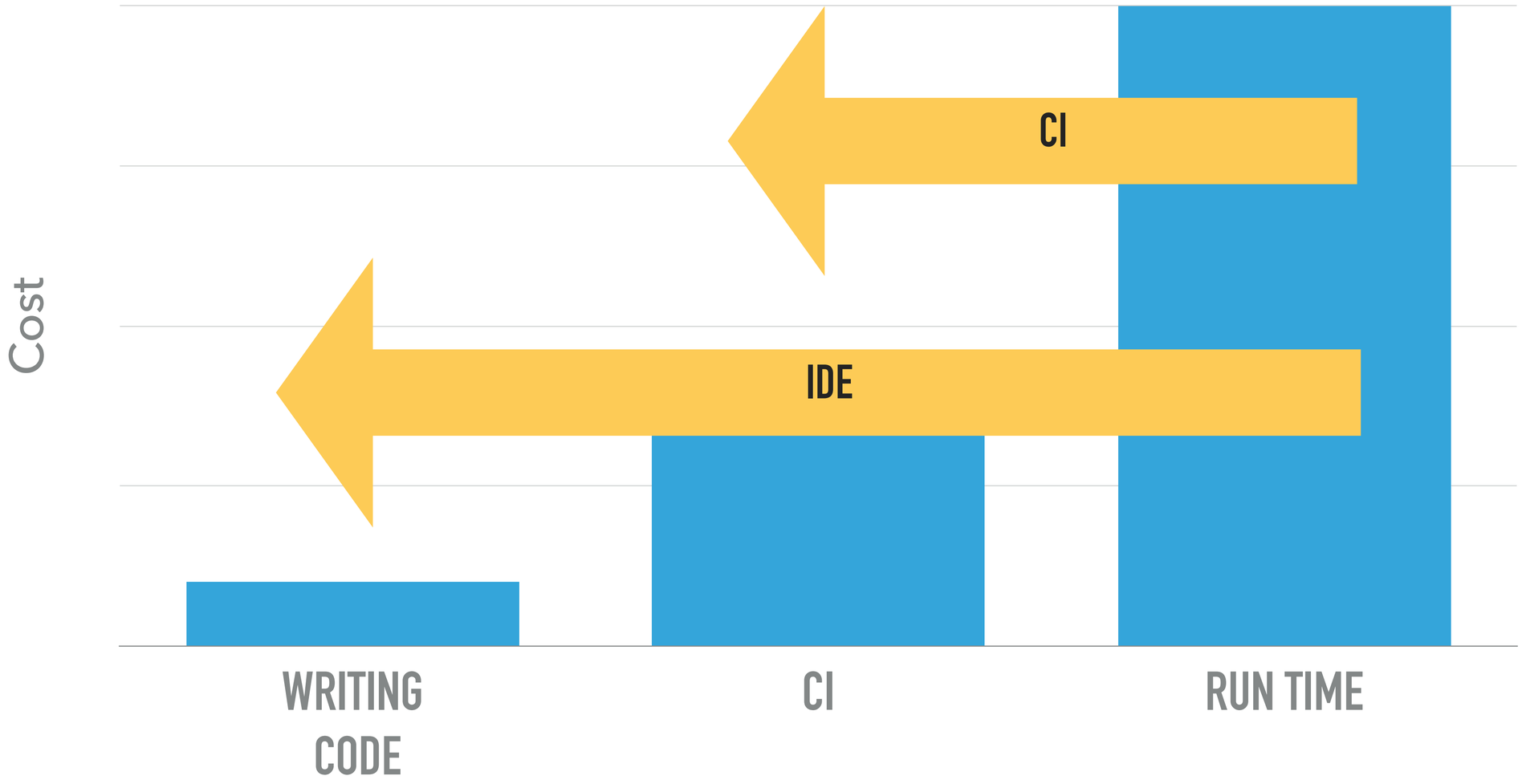
WHY NOT JUST USE JAVA?



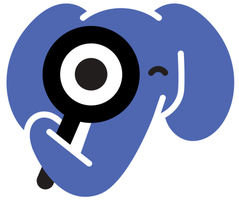
AGENDA



ADD CLARITY TO CODE. FIND SOME BUGS EARLIER. ENABLE REFACTORING

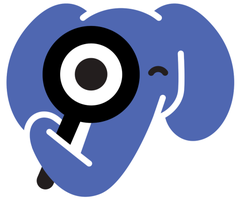


USING GENERICS NOW



Psalm

USING GENERICS NOW



Psalm

WE ALREADY HAVE GENERICS!

I'VE BEEN DOING THIS FOR 6 YEARS.

NOW YOU CAN TOO!

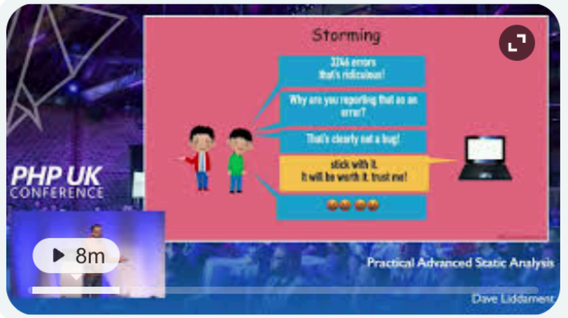
static analysis dave

All Images Videos Shopping Short videos News More Tools

AI Overview

Dave Liddament is a prominent expert in PHP static analysis, known for advocating for tools like PHPStan and Psalm to improve code quality and safety. He created the Static Analysis Results Baseliner (SARB), a tool designed to integrate static analysis into legacy projects by managing existing issues. Dave Liddament +2

This video explains how to use static analysis tools like PHPStan and Psalm:



Practical Advanced Static Analysis - Dave Liddament - PHP ...

PHP UK Conference
YouTube• 30 Mar 2022

Key aspects of Dave Liddament's work include:

- **SARB (Static Analysis Results Baseliner):** A PHP tool used to create a baseline of existing issues in a codebase, allowing developers to focus on new, incoming issues.
- **Practical Static Analysis:** Focuses on using advanced static analysis to find bugs, ensure code quality, and enable safe refactoring.
- **Legacy Code Solutions:** Specializes in introducing modern static analysis tools to old projects without needing to fix thousands of existing issues immediately.
- **Presentations:** Frequent speaker at PHP conferences on topics including static analysis, custom rules, and testing. GitHub +5

DaveLiddament/sarb: Static Analysis Results Baseliner · GitHub
4 Jan 2025 — Static Analysis Baseliner (SARB) is a PHP script that creates a baseline of static analysis…
GitHub

Squash Bugs with Static Analysis | Dave Liddament | IPC 2018
28 Nov 2018 — Squash Bugs with Static Analysis | Dave Liddament | IPC 2018 - YouTube. ... This conte…
YouTube · International PHP Confer…

Introducing Static Analysis Results Baseliner (SARB)
Dave Liddament's technical blog. ... SARB is written in PHP, however it can be used to baseline results for any language and any s...
Dave Liddament

Show all

Dave Liddament

github.com/DaveLiddament/php-language-extensions
github.com/DaveLiddament/phpstan-rule-test-helper
github.com/DaveLiddament/test-splitter
github.com/DaveLiddament/sarb

www.linkedin.com/in/daveliddament/
@daveliddament@phpc.social
@daveliddament.bsky.social
@daveliddament

Me when I’m not coding!