PHP Generics Today (Almost)

LONGHORN
PHP CONFERENCE

Dave Liddament
@daveliddament

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”);
```

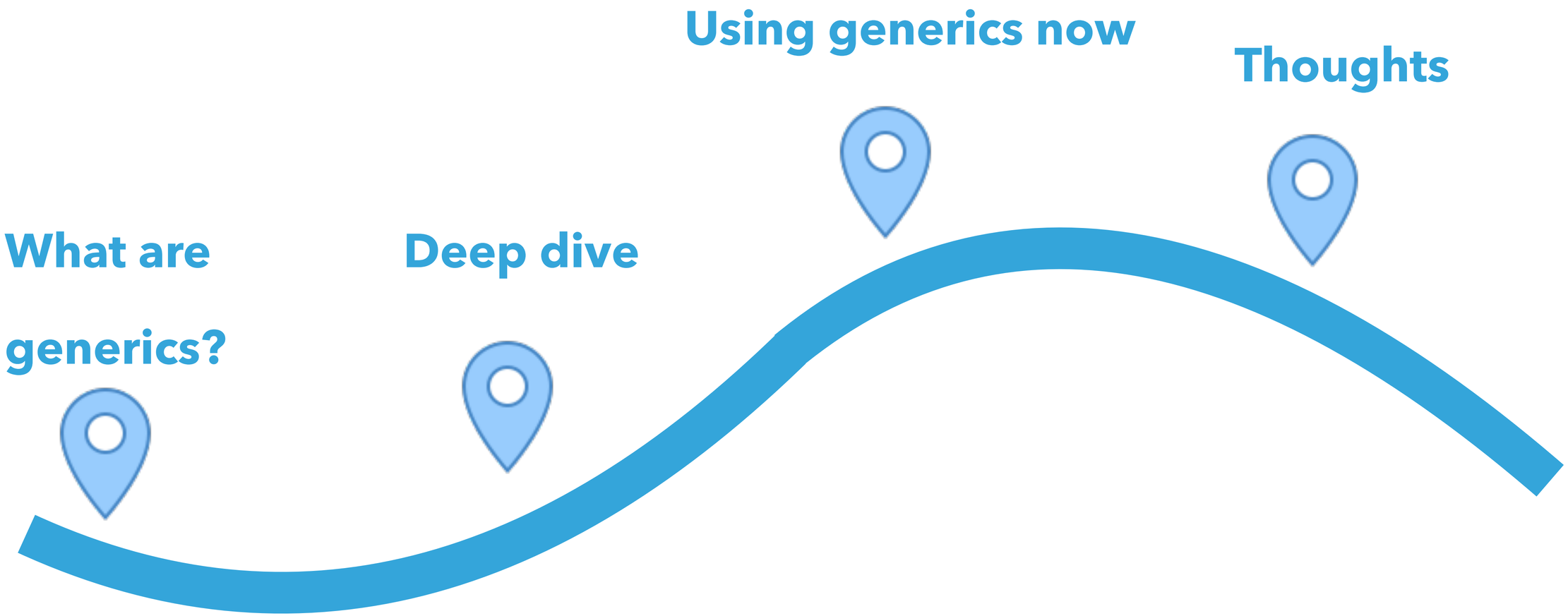
```
/** @template T of Animal */
interface AnimalProcessor {
  /** @return class-string<T> */
  public function supports(): string;

  …
```

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

process(“Bob”);
```

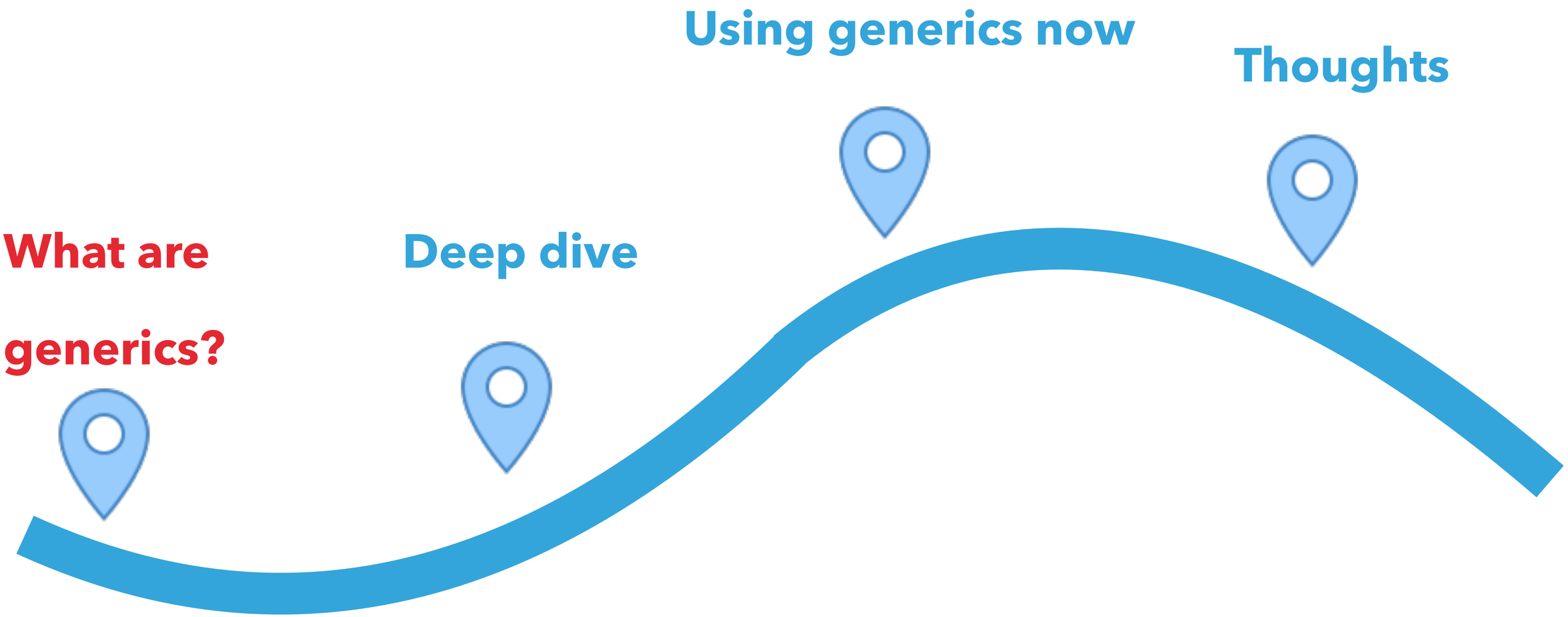
```
/** @template T of Animal */
interface AnimalProcessor {
  /** @return class-string<T> */
  public function supports(): string;

  …
```

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

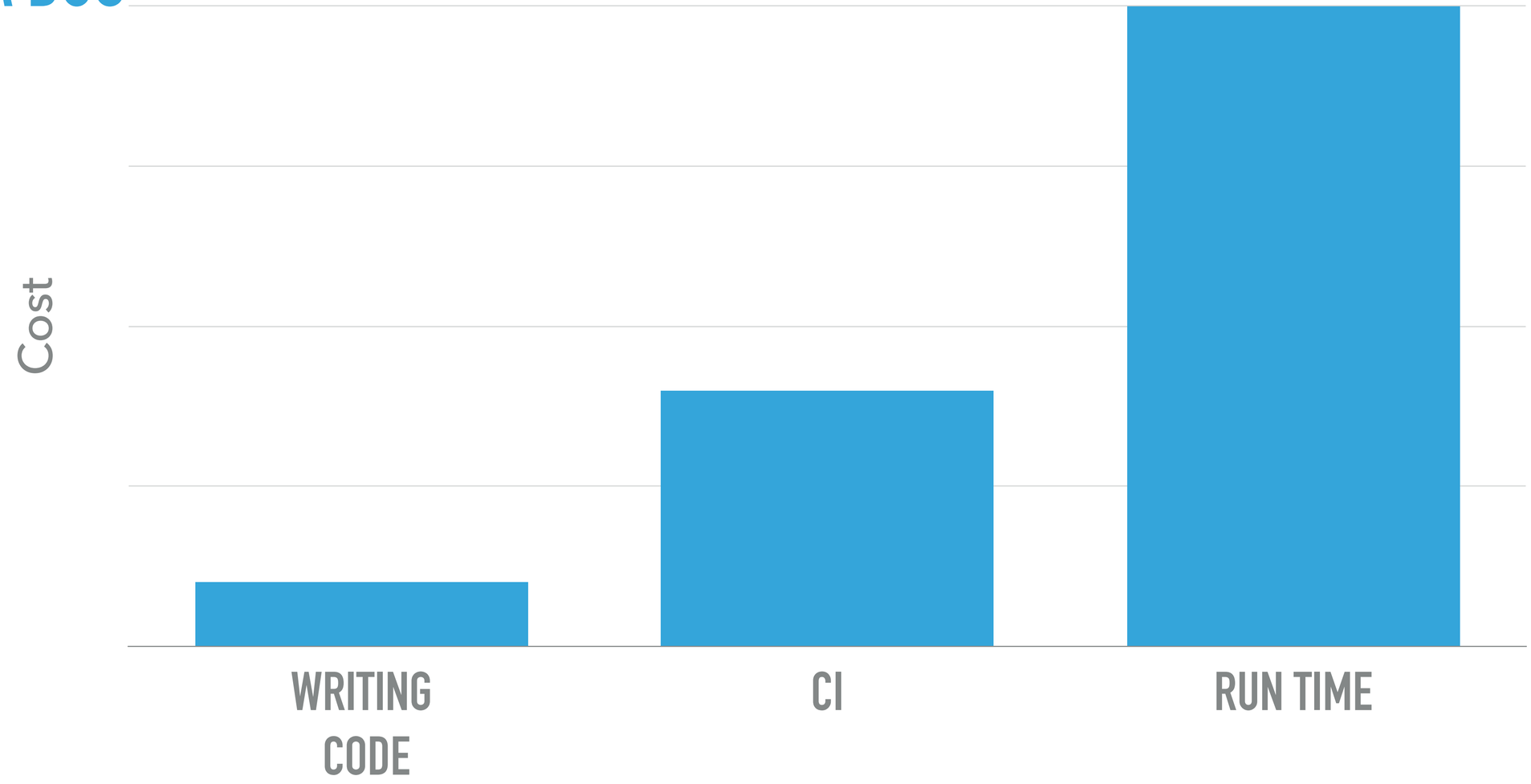
```
function process(User $user): void { … }

process(“Bob”);
```

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

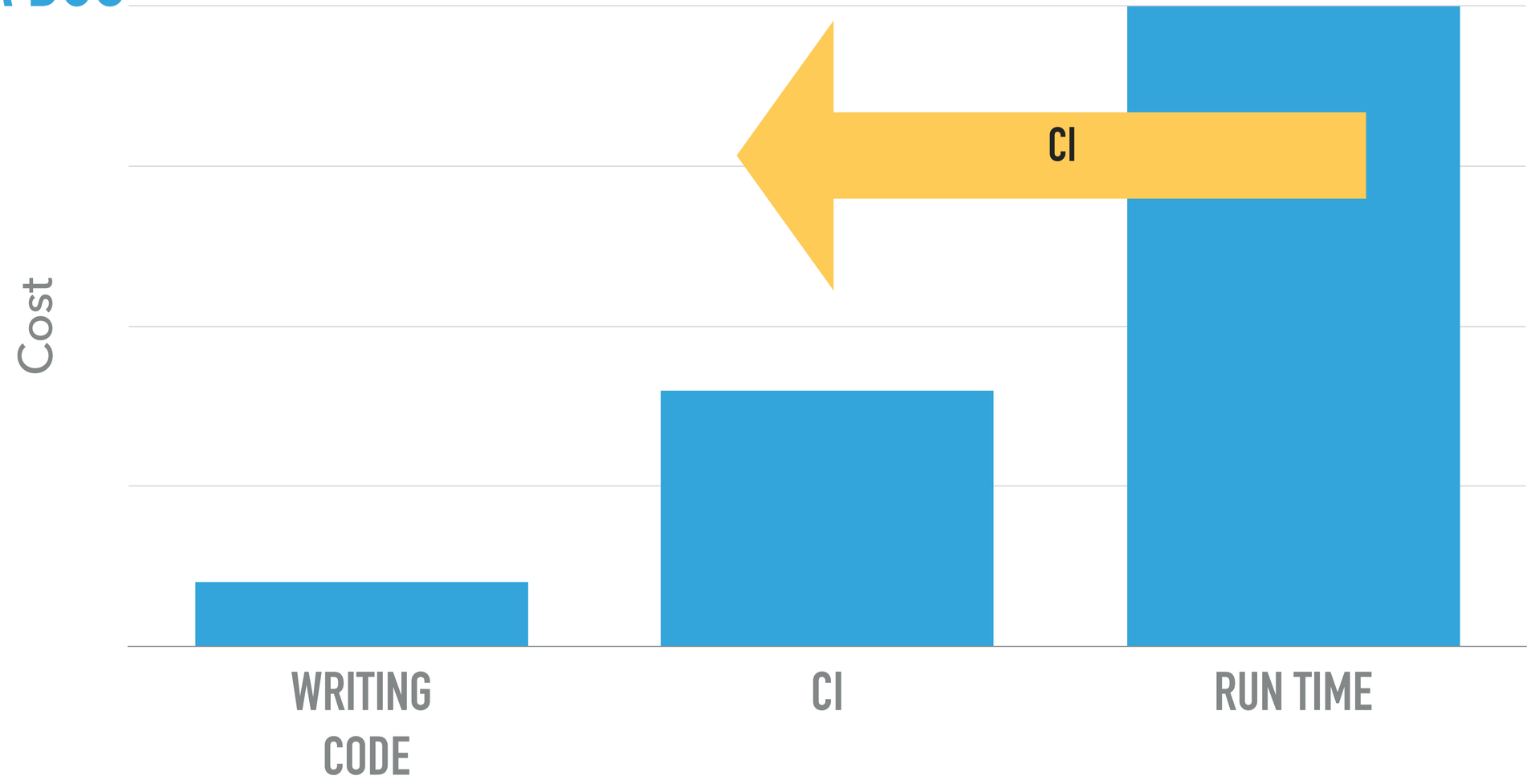
```
function process(User $user): void { … }

process(“Bob”);
```

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

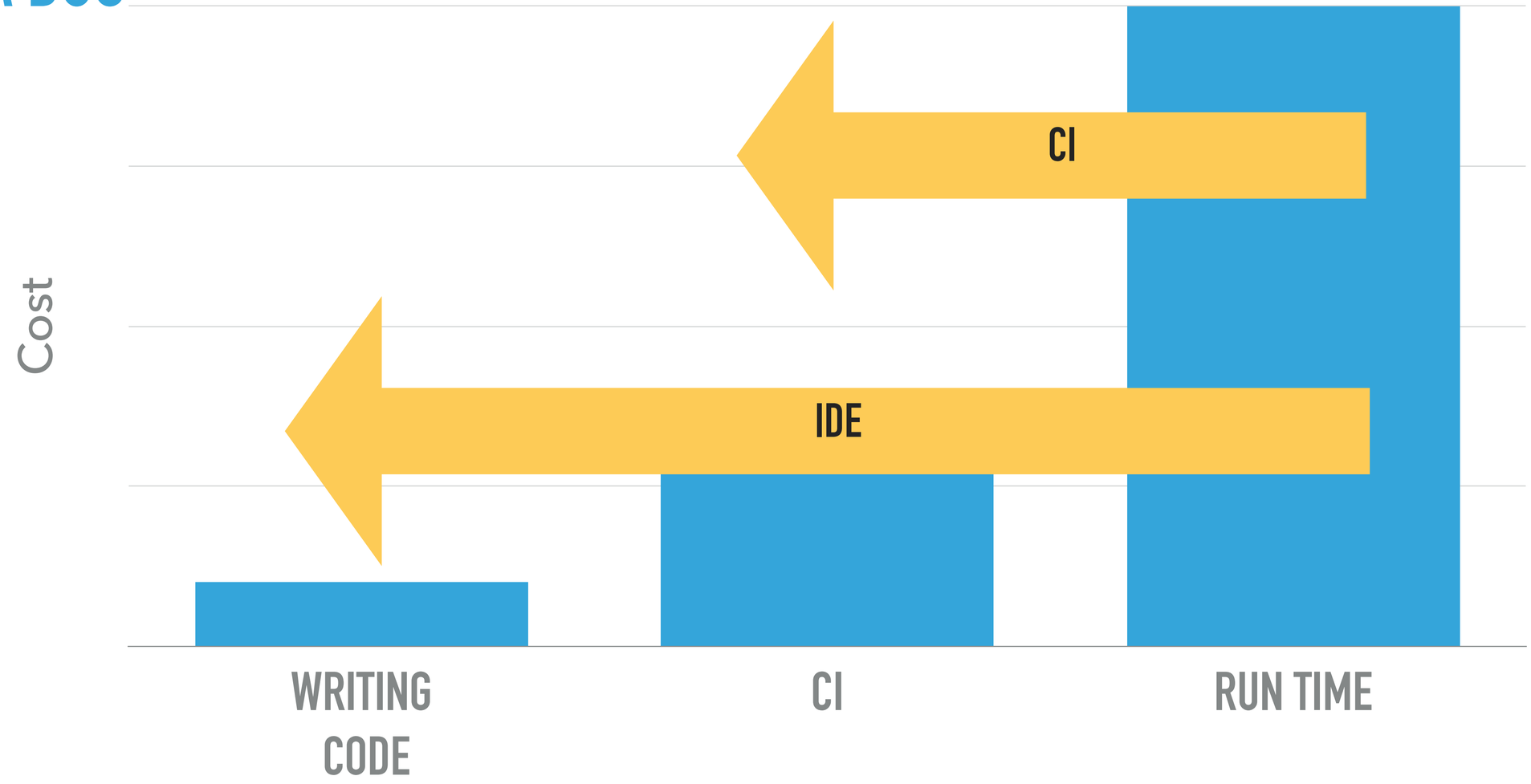
COST OF A BUG



COST OF A BUG



COST OF A BUG



```
class 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

Phan,

Static Analyzer for PHP

PHPStan

```php
<?php

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

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

ERROR: InvalidReturnStatement - 10:10 - The inferred type 'array{User, 'james'}' does not match the declared return type 'array<array-key, User>' for getUsers

ERROR: InvalidReturnType - 7:13 - The declared return type 'array<array-key, User>' for getUsers is incorrect, got 'array{User, 'james'}'

```php
<?php

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

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

```
ERROR: InvalidReturnStatement - 10:10 - The inferred type 'array{User, 'james'}' does not match the declared return type 'array<array-key, User>' for getUsers

ERROR: InvalidReturnType - 7:13 - The declared return type 'array<array-key, User>' for getUsers is incorrect, got 'array{User, 'james'}'
```

```
function getUsers(): array
{
    return [
        new User("Jane"),
        "james",
    ];
}
```

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();
```

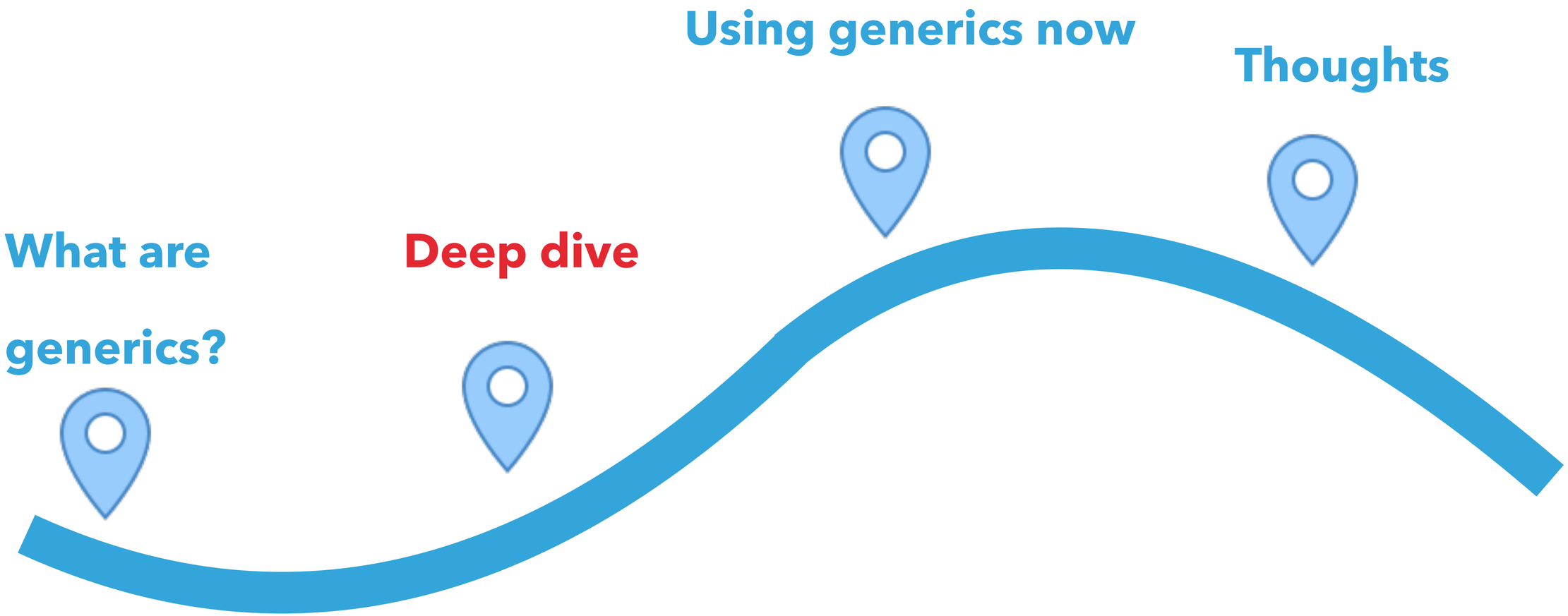
```
/** @template T */
class Queue() { … }

/** @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 { … }
```

❌

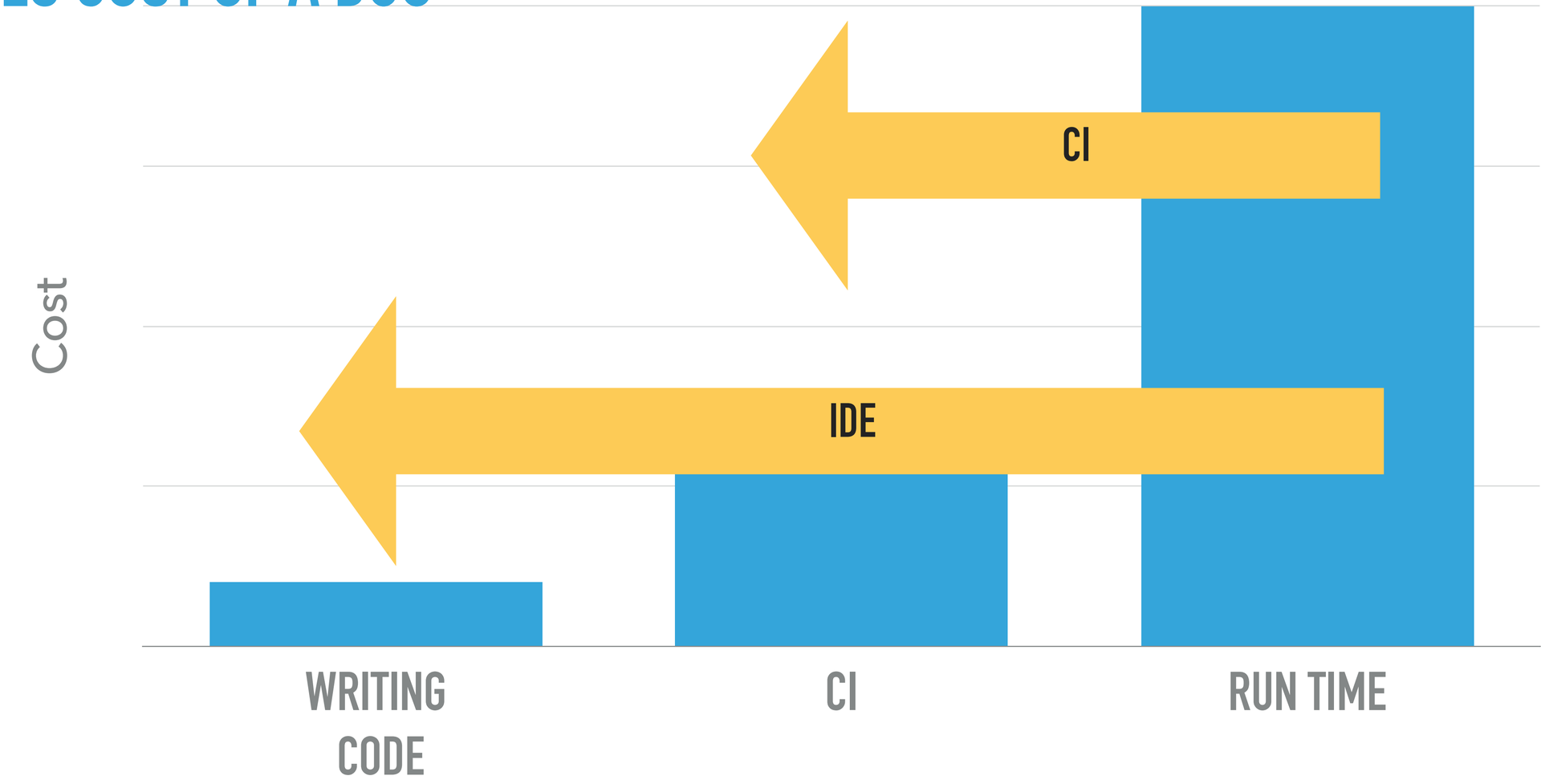
✅

HOW DOES THIS HELP US?

1. COMMUNICATES ADDITIONAL TYPE INFORMATION

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

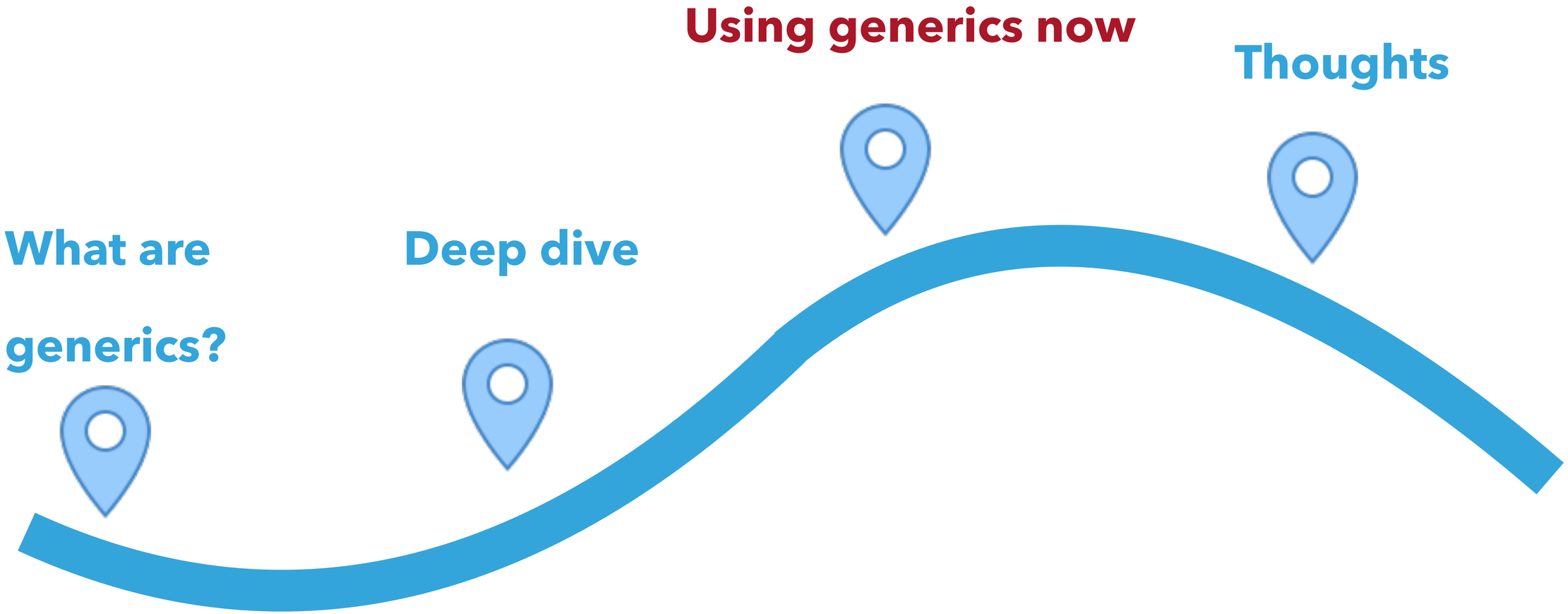
2. REDUCES COST OF A BUG



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

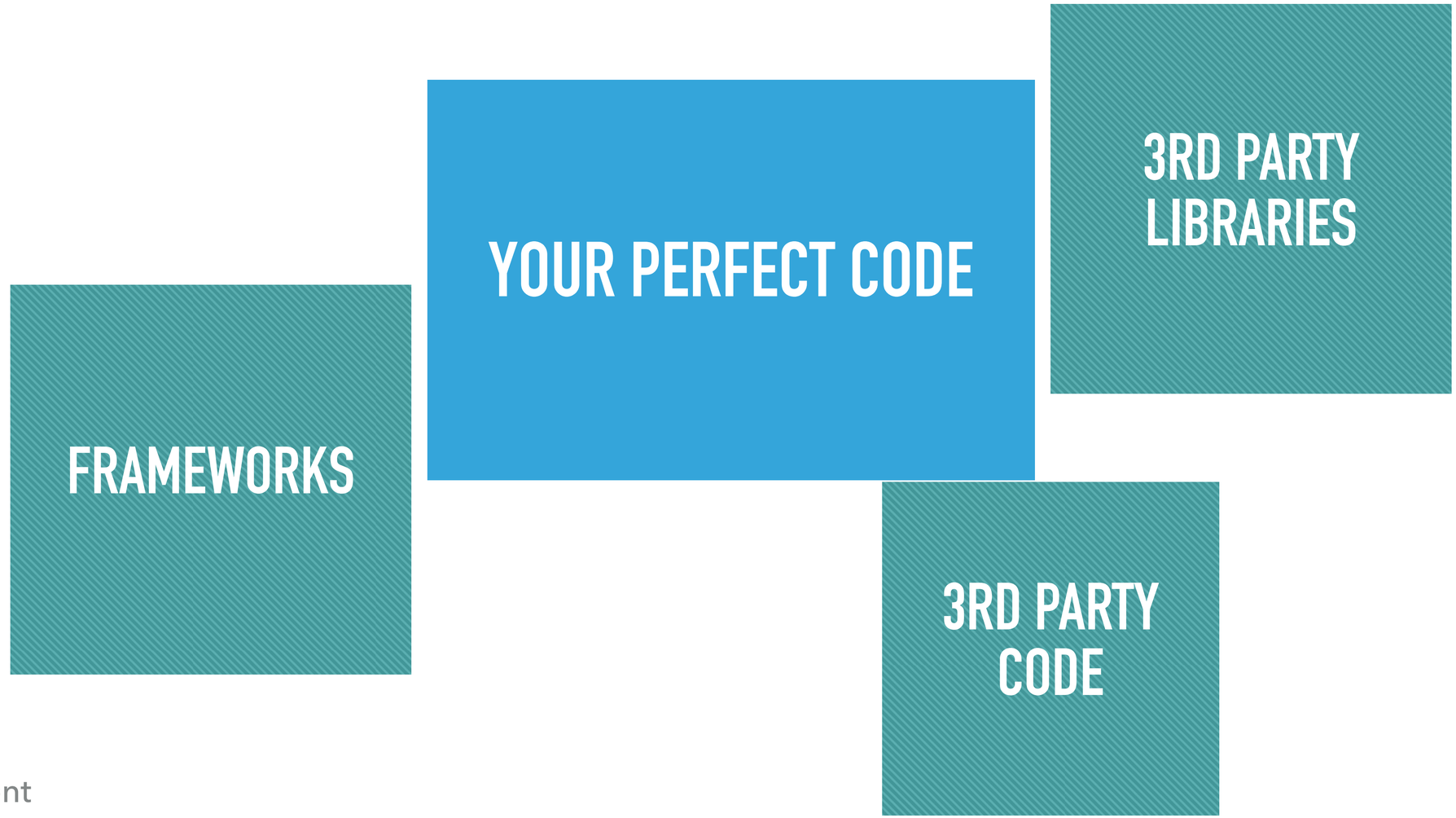
USING GENERICS NOW

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

```
interface Hasher {
    /**
     * @return string
     */
    public function encode();
}

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

ADAPTORS FOR 3RD PARTY LIBRARIES: PROBLEM

```
interface Hasher {
    /**
     * @return string
     */
    public function encode();
}

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

ADAPTORS FOR 3RD PARTY LIBRARIES: PROBLEM

```
interface Hasher {
    /**
     * @return string
     */
    public function encode();
}

$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

Stubs/ThirdParty/DI.php

```
namespace ThirdParty\DI;

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

Stubs/ThirdParty/DI.php

```
namespace ThirdParty\DI;

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

Stubs/ThirdParty/DI.php

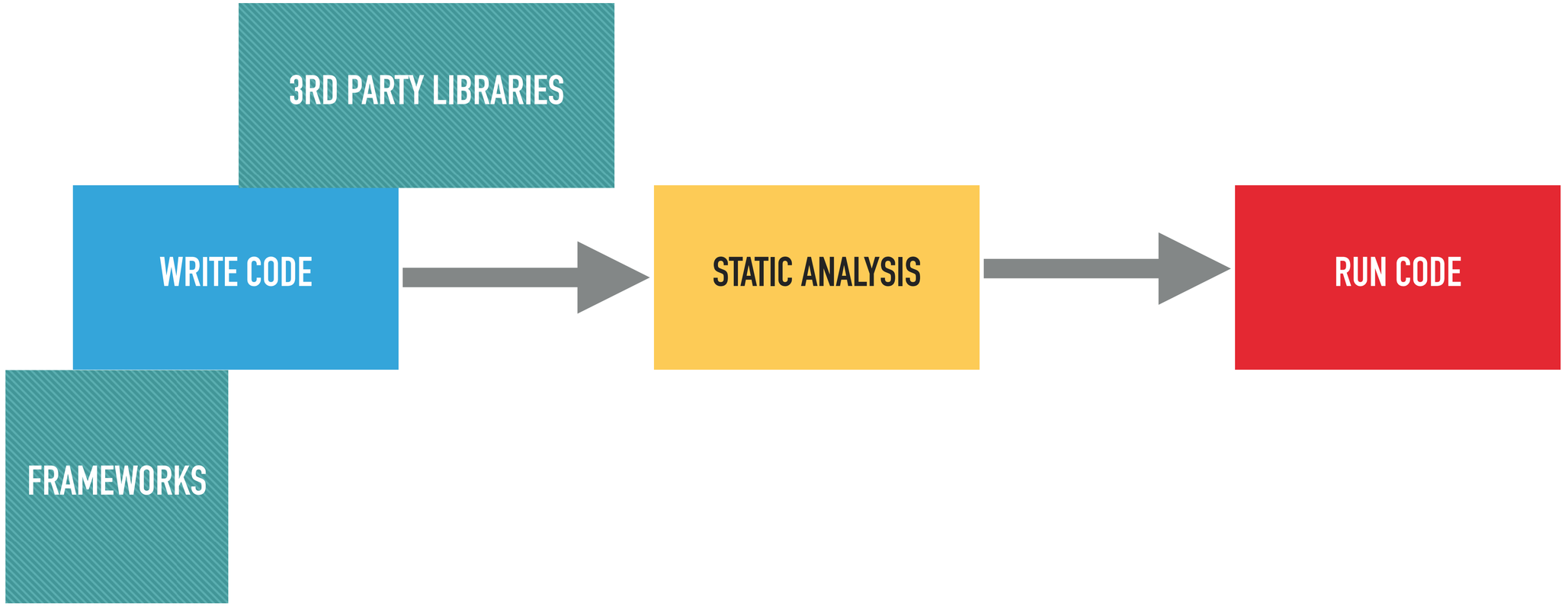
```php
namespace ThirdParty\DI;

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

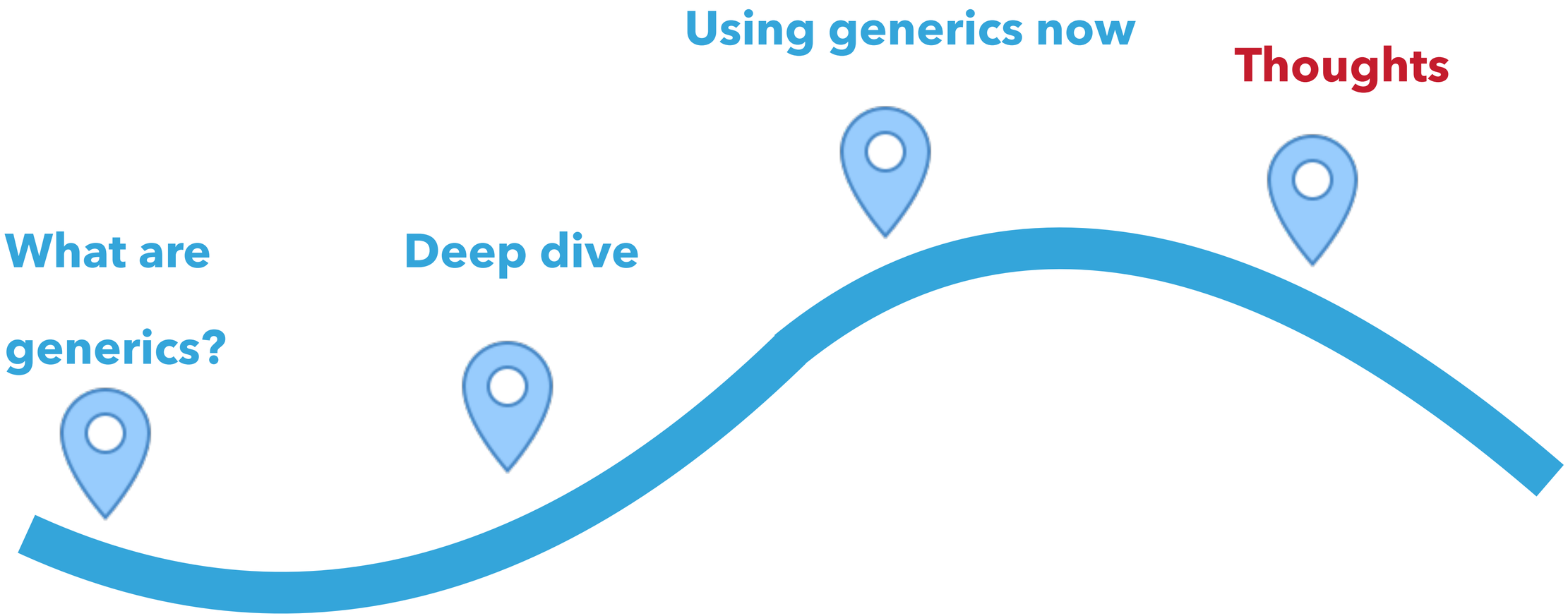
USING GENERICS NOW

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)

DO WE NEED A FORMAL STANDARD?

- Notation
- Test suite

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

FULL LANGUAGE SUPPORT

```
class Queue <T>
{
  public function add( T $item ): void {…}

  public function next(): T {…}
}

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

FULL LANGUAGE SUPPORT

```
class Queue<T>
{
  public function add( T $item ): void {…}

  public function next(): T {…}
}

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

FULL LANGUAGE SUPPORT

```
class Queue<T>
{
  public function add(T $item): void {…}

  public function next(): T {…}
}

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

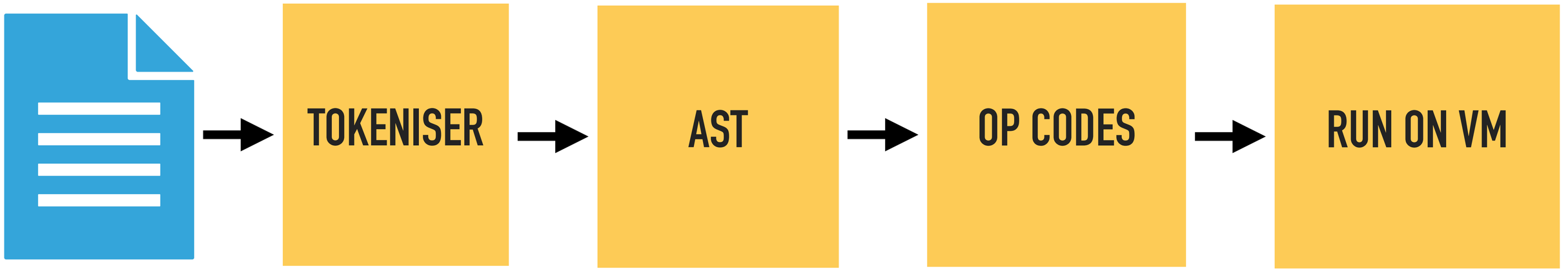
FULL LANGUAGE SUPPORT

```
class Queue<T>
{
  public function add(T $item): void {…}

  public function next():T {…}
}

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

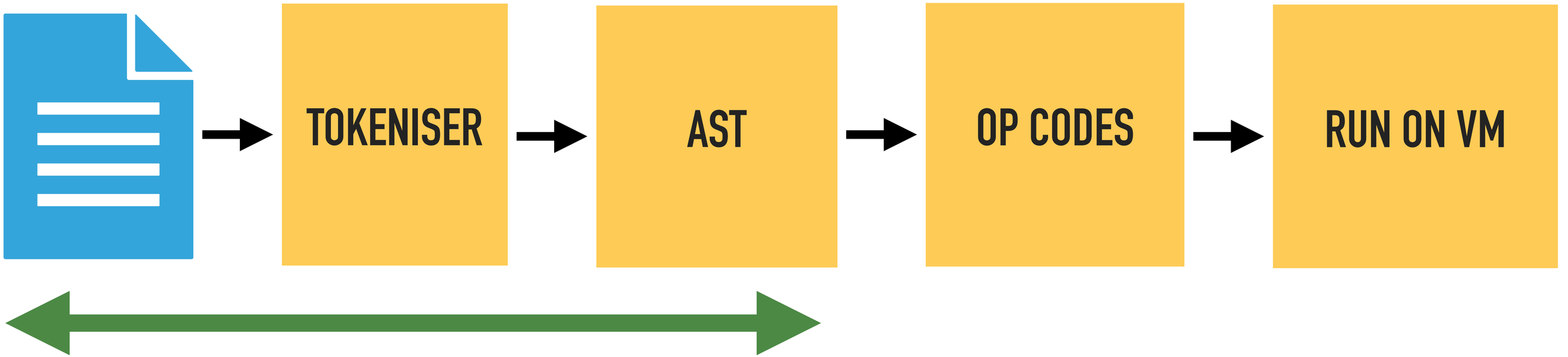
FULL LANGUAGE SUPPORT

```
class Queue<T>
{
  public function add(T $item): void {…}

  public function next():T {…}
}

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

PARTIAL LANGUAGE SUPPORT

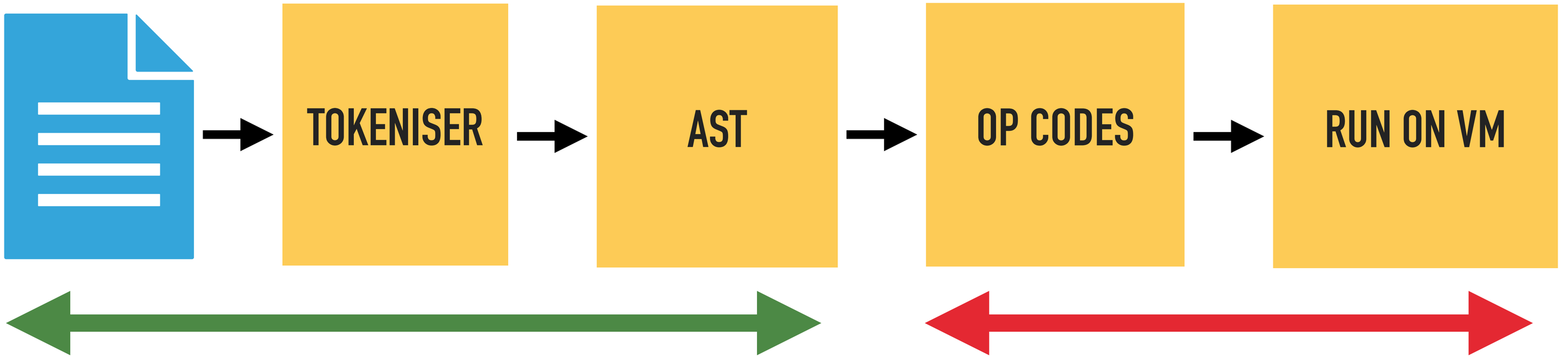


PARTIAL LANGUAGE SUPPORT

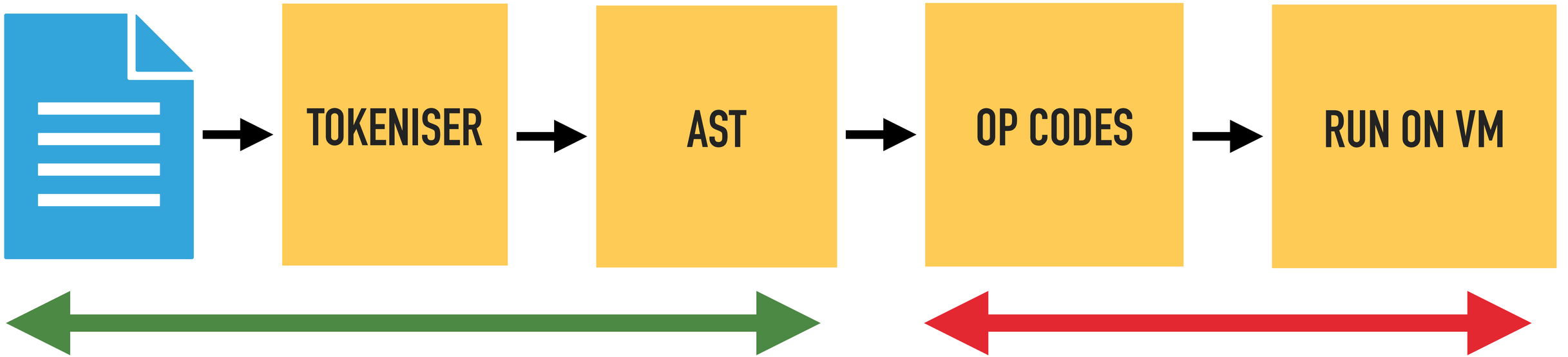


Understanding of generics

PARTIAL LANGUAGE SUPPORT



PARTIAL LANGUAGE SUPPORT



Validated by Static Analysis, not at run time.

WHAT WOULD STANDARD LOOK LIKE?

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

WHAT WOULD STANDARD LOOK LIKE?

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

```
#[Template("T")]
#[Type("array<int,T>")]
function asArray(
    #[Type("T")] $value,
) {
    return [$value];
}
```

https://github.com/DaveLiddament/php-generics-standard

https://www.daveliddament.co.uk/articles/php-generics-standard/

tl;dr: How about using generics in PHP attributes?

```php
#[<T>]
class Stack
{
    public function push(#[<T>] mixed $item): void
    {
    }

    public function pop(): #[<T>] mixed
    {
    }
}
```

https://pronskiy.com/blog/generics-via-attributes-in-php/

HOW ABOUT #<>

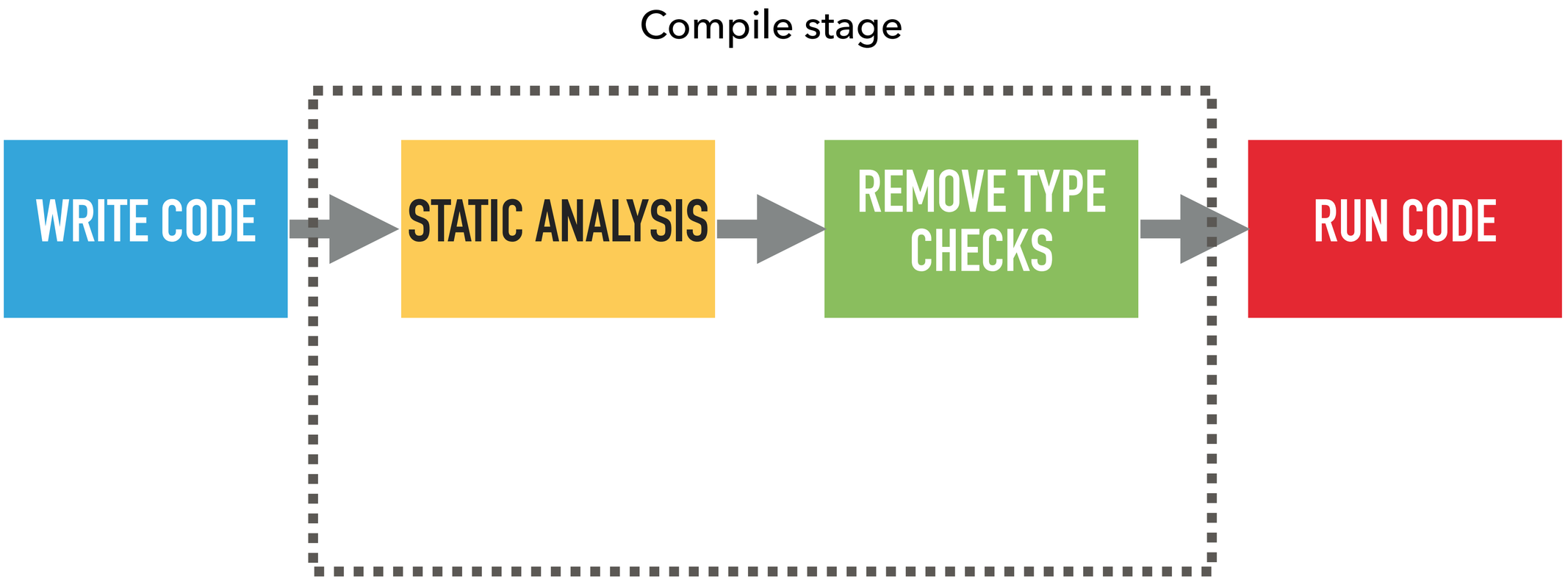
```
function getPeople(): array#<int, Person>(){
    // Some code
}
```

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> {…}
```

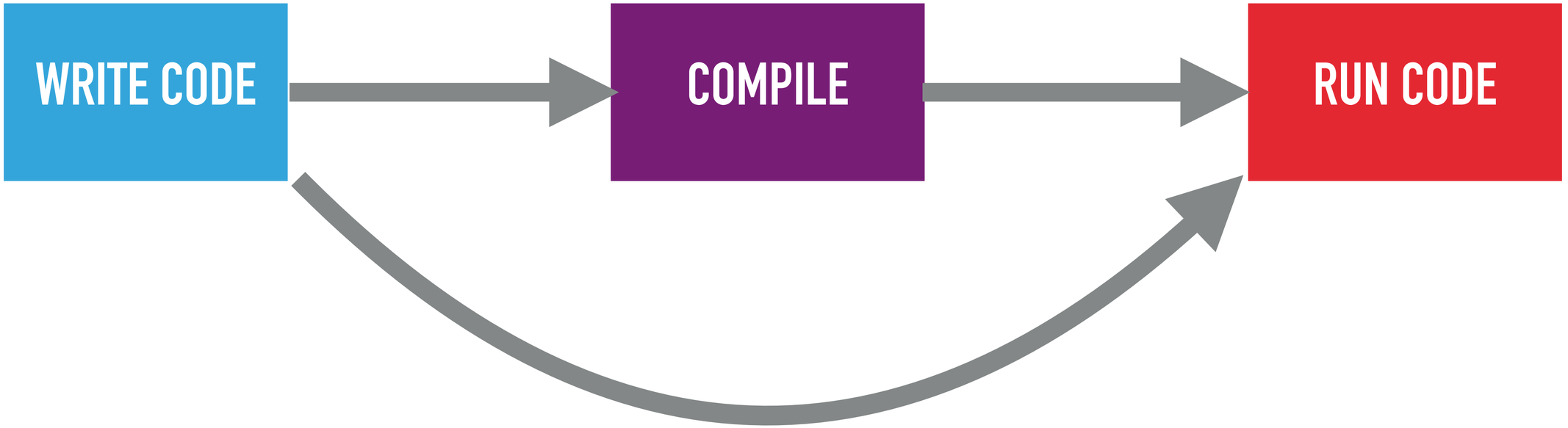
```
class Queue #<T of object>
{
  private array#<int,T> $queue = [];

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

  public function next(): #<T> {…}
}

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

interface Repository #<T> {…}

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

THE END OF RUN TIME CHECKS?

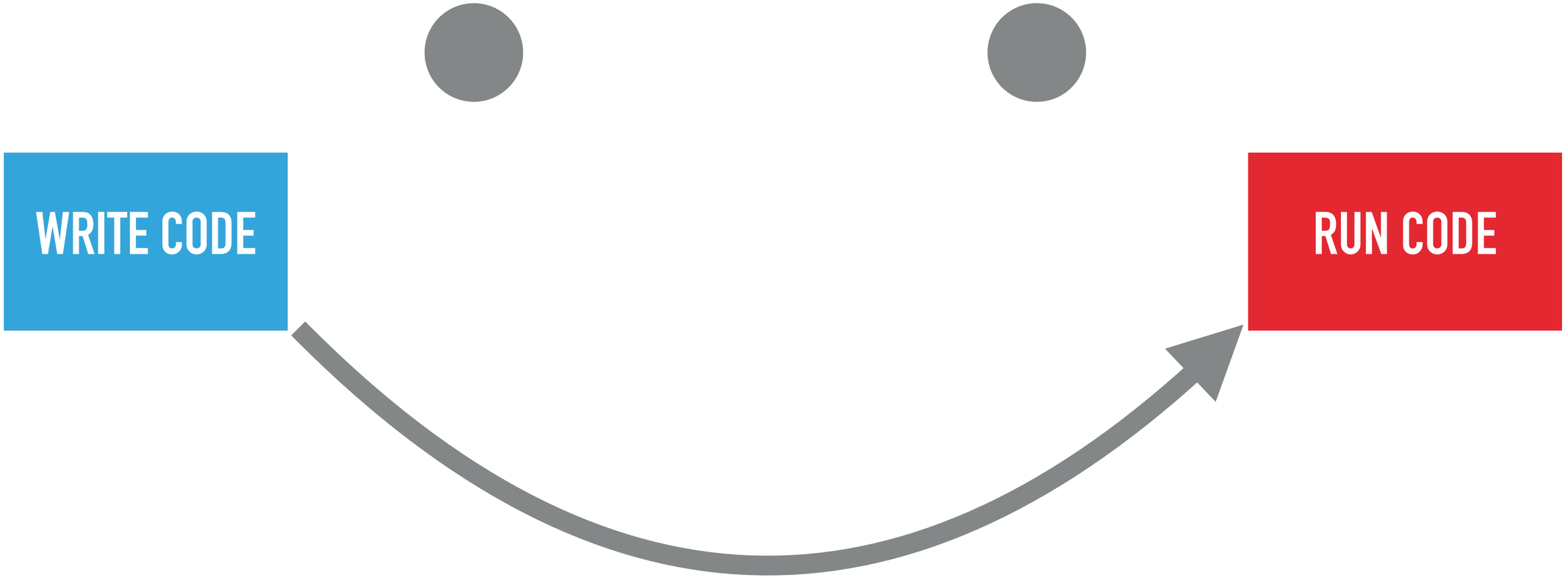


WHY NOT JUST USE JAVA?

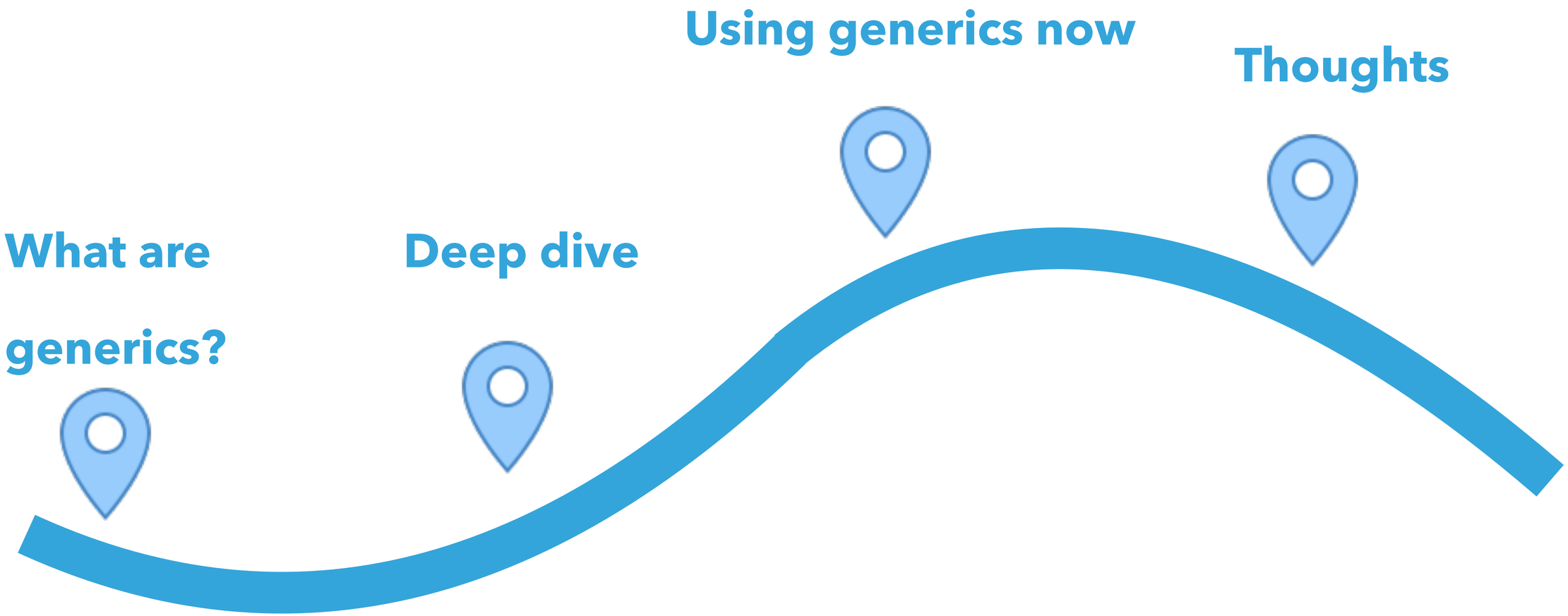
WHY NOT JUST USE JAVA?



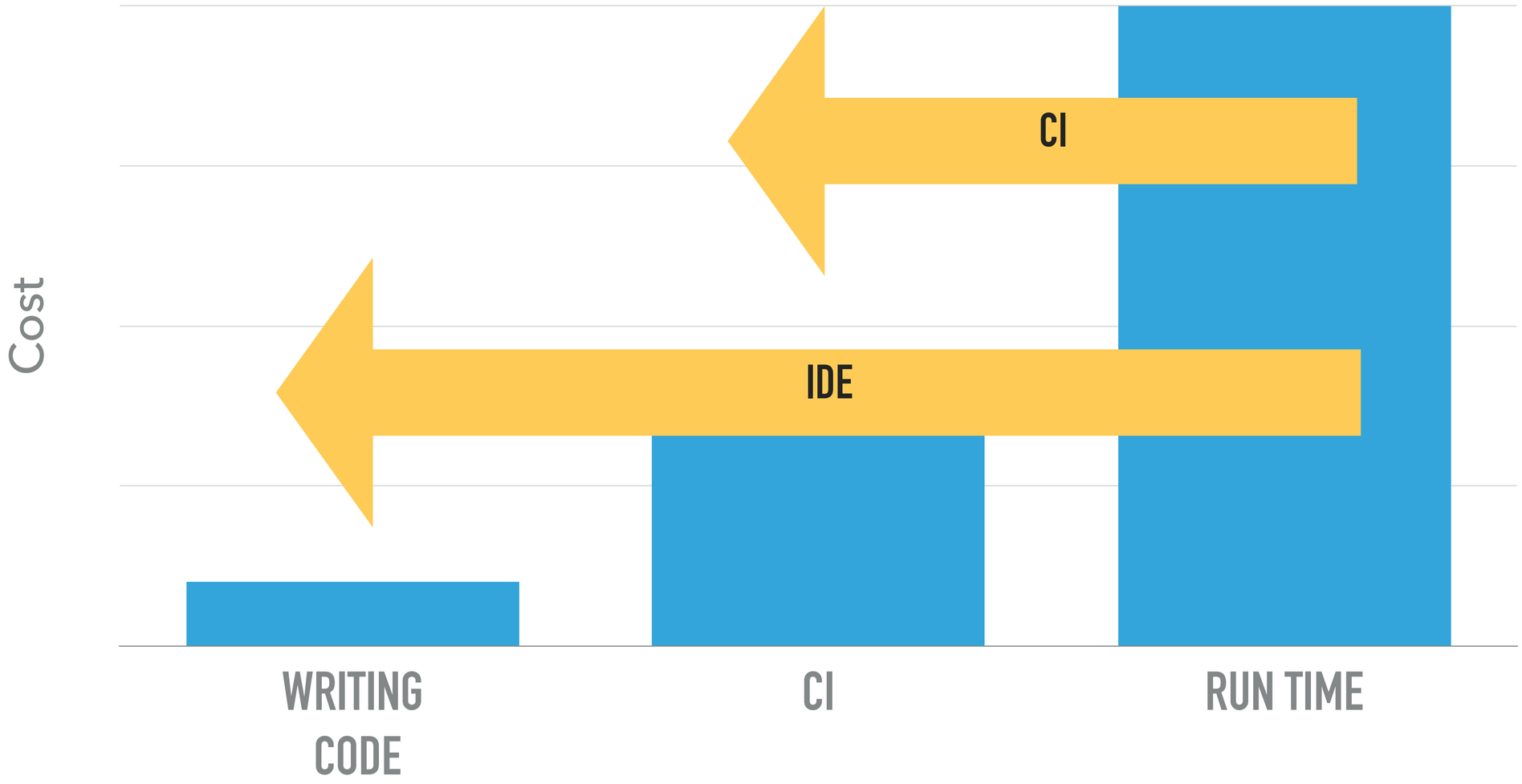
WHY NOT JUST USE JAVA?



AGENDA



ADD CLARITY TO CODE. FIND SOME BUGS EARLIER.



USING GENERICS NOW

Psalm

Dave Liddament

Lamp Bristol

Thank you for listening

Organise PHP-SW

Author of Static Analysis Results Baseliner (SARB)

20 years of writing software (C, Java, Python, PHP)

@daveliddament

FURTHER READING

- Slides: https://www.daveliddament.co.uk/talks/php-generics-today-almost
- Code: https://github.com/DaveLiddament/php-generics-today-almost
- Static Analysers:
  - Psalm: https://psalm.dev
  - PHPStan: https://phpstan.org
- RFC and notes:
  - https://wiki.php.net/rfc/generics
  - https://github.com/PHPGenerics/php-generics-rfc/issues/45
  - https://wiki.php.net/rfc/annotations_v2
- Thoughts on a standard
  - https://www.daveliddament.co.uk/articles/php-generics-standard/
  - https://github.com/DaveLiddament/php-generics-standard
  - https://pronskiy.com/blog/generics-via-attributes-in-php/
  - https://gist.github.com/DaveLiddament/40130a7a107478bf6f92fcbb0b01a2fc