PHP GENERICS TODAY (ALMOST)

DAVE LIDDAMENT

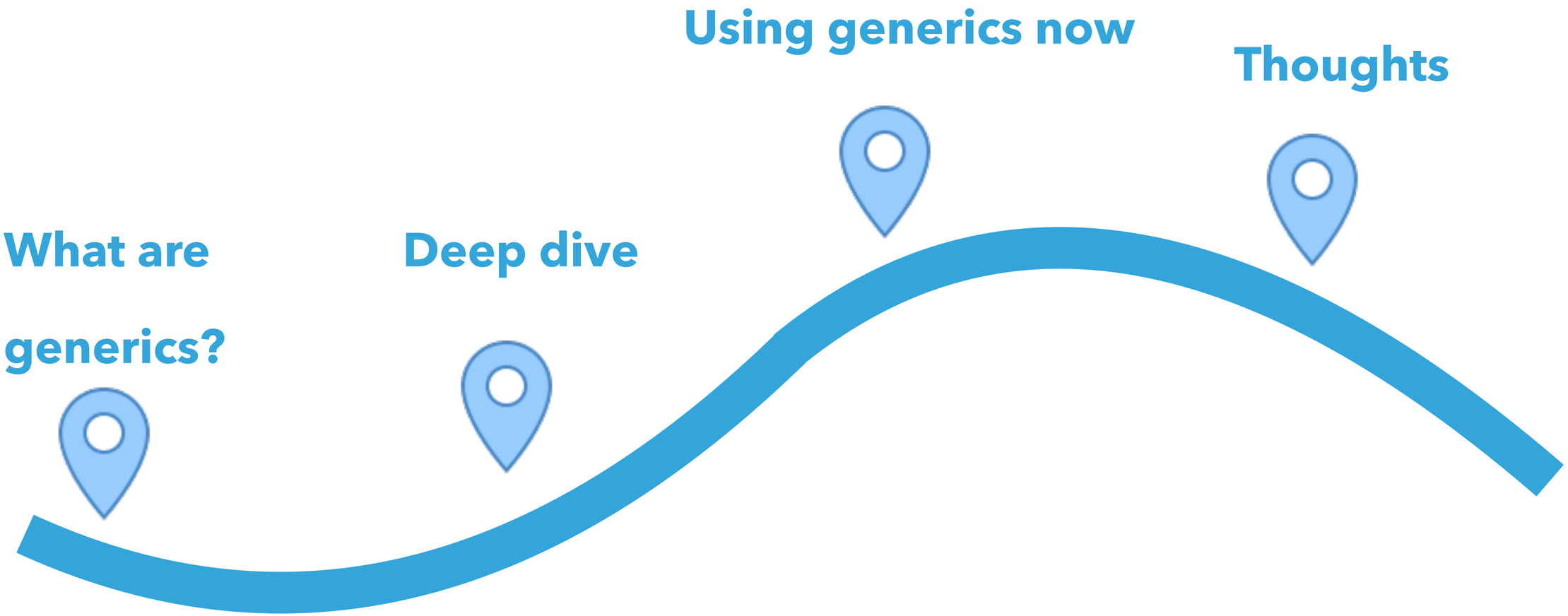
Lamp Bristol

@daveliddament

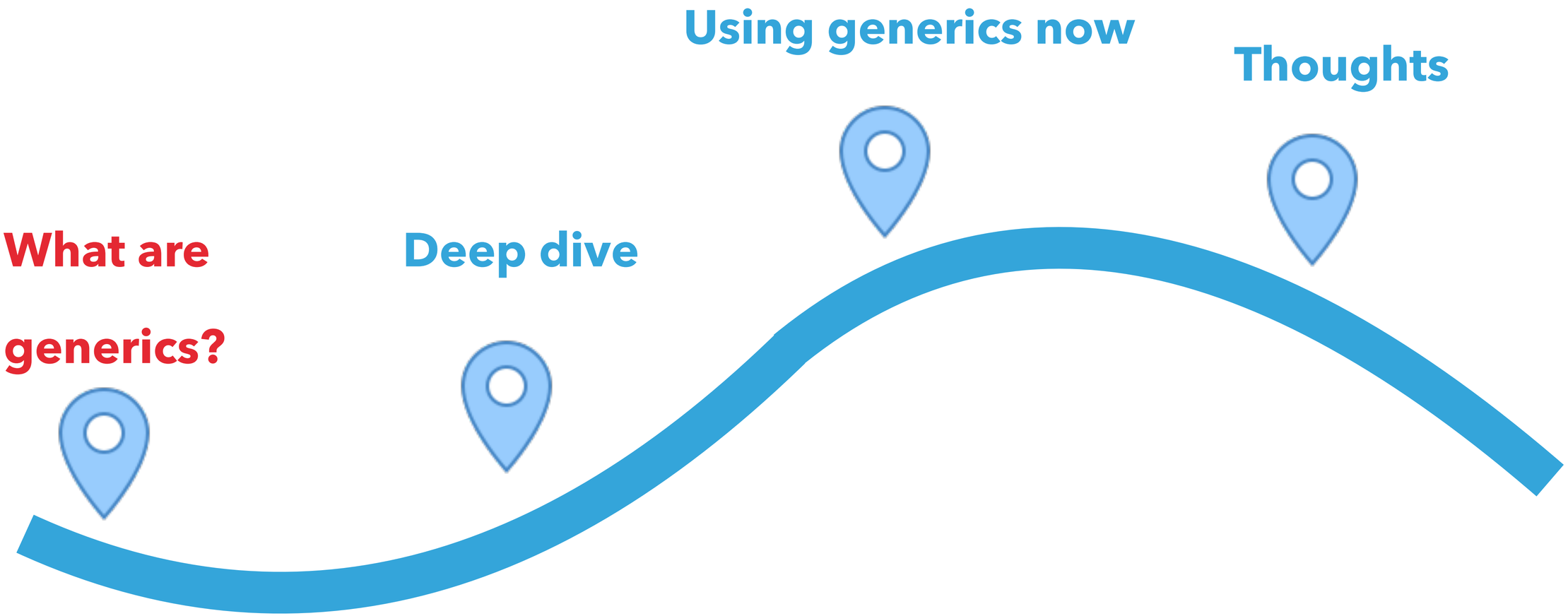
USING GENERICS CAN HELP US WRITE MORE UNDERSTANDABLE, ROBUST AND RELIABLE CODE.

DEMONSTRATE HOW EXISTING TOOLS CAN (ALMOST) GIVE US THE BENEFITS OF GENERICS NOW.

AGENDA



AGENDA



```
processUser(1);

function processUser(User $user) {
  // some implementation
}
```

```
processUser(1);

function processUser(User $user) {
  // some implementation
}
```

```
class Queue {

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

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

```
class Queue {

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

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

```
$queue = getQueue(); // Returns Queue

$user = $queue->getNext();
processUser($user); // Hope it's a User

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

```
$queue = getQueue(); // Returns Queue
```

```
$user = $queue->getNext();
processUser($user); // Hope it's a User
```

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

```
$queue = getQueue(); // Returns Queue

$user = $queue->getNext();
processUser($user); // Hope it's a User

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

```
$queue = getQueue(); // Returns Queue

$user = $queue->getNext();
processUser($user); // Hope it's a User


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

```
class TypedQueue {
 private string $type;
 private Queue $queue;
 public function __construct(string $type) {
   $this-> type = $type;
   $this->queue = new Queue();
 }
 public function add($item): void {
  if (!is_a($item, $this->type, true)) throw TypeError();
  $this->queue->add($item);
 }
  public function getNext() {
    return $this->queue->getNext();
  }
}
```

```
class TypedQueue {
  private string $type;
  private Queue $queue;
  public function __construct(string $type) {
    $this-> type = $type;
    $this->queue = new Queue();
  }
  public function add($item): void {
   if (!is_a($item, $this->type, true)) throw TypeError();
   $this->queue->add($item);
  }
   public function getNext() {
     return $this->queue->getNext();
   }
}
```

```
class TypedQueue {
  private string $type;
  private Queue $queue;
  public function __construct(string $type) {
    $this-> type = $type;
    $this->queue = new Queue();
  }
  public function add($item): void {
   if (!is_a($item, $this->type, true)) throw TypeError();
   $this->queue->add($item);
  }
   public function getNext() {
     return $this->queue->getNext();
   }
}
```

@daveliddament

```
class TypedQueue {
 private string $type;
 private Queue $queue;
 public function __construct(string $type) {
   $this-> type = $type;
   $this->queue = new Queue();
 }
 public function add($item): void {
  if (!is_a($item, $this->type, true)) throw TypeError();
  $this->queue->add($item);
 }
  public function getNext() {
    return $this->queue->getNext();
  }
}
```

```
class TypedQueue {
 private string $type;
 private Queue $queue;
 public function __construct(string $type) {
   $this-> type = $type;
   $this->queue = new Queue();
 }
 public function add($item): void {
  if (!is_a($item, $this->type, true)) throw TypeError();
  $this->queue->add($item);
 }
  public function getNext() {
    return $this->queue->getNext();
  }
}
```

@daveliddament

```
$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;
 public function __construct() {
   $this->queue = new Queue();
 }
 public function add(User $item): void {
  $this->queue->add($item);
 }
 public function getNext(): User {
   return $this->queue->getNext();
 }
}
```

```
class UserQueue {
  private Queue $queue;
  public function __construct() {
    $this->queue = new Queue();
  }
  public function add(User $item): void {
   $this->queue->add($item);
  }
  public function getNext(): User {
    return $this->queue->getNext();
  }
}
```

```
class UserQueue {
  private Queue $queue;
  public function __construct() {
    $this->queue = new Queue();
  }
  public function add(User $item): void {
   $this->queue->add($item);
  }
  public function getNext(): User {
    return $this->queue->getNext();
  }
}
```

@daveliddament

```
class UserQueue {
 private Queue $queue;
 public function __construct() {
   $this->queue = new Queue();
 }
 public function add(User $item): void {
  $this->queue->add($item);
 }
 public function getNext(): User {
   return $this->queue->getNext();
 }
}
```

@daveliddament

```
class UserQueue {
 private Queue $queue;
 public function __construct() {
   $this->queue = new Queue();
 }
 public function add(User $item): void {
  $this->queue->add($item);
 }
 public function getNext(): User {
   return $this->queue->getNext();
 }
}
```

@daveliddament

```
$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"); ❌
```

```
$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 {…}
```

```
/** @param User[] $users */
function processUsers(array $users): void {…}


processUsers([
  new User(“Jane”),
  “james”,
]);
```

```
/** @param User[] $users */
function processUsers(array $users): void {…}
```

```
processUsers([
  new User("Jane"),
  "james",
]);
```

```
/** @param User[] $users */
function processUsers(array $users): void {…}
```

```
processUsers([
  new User(“Jane”),
  “james”,
]);
```

```
/** @param User[] $users */
function processUsers(array $users): void {…}


processUsers([
  new User("Jane"),
  "james",
]);
```

CAN WE ENFORCE THESE CHECKS?

CAN WE ENFORCE THESE CHECKS?

Psalm

Phan,

Static Analyzer for PHP

PHPStan

Psalm

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

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

/** @param User[] $users */
function processUsers(array $users): void {
    var_export($users);
}

processUsers([
  new User('Jane'),
  'james',
]);
```

```
Psalm output (using commit 39a8227):

ERROR: InvalidArgument - 12:14 - Argument 1 of processUsers expects array<array-key, User>, array{0: User, 1:
string(james)} provided
```

Psalm

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

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

/** @param User[] $users */
function processUsers(array $users): void {
    var_export($users);
}

processUsers([
  new User('Jane'),
  'james',
]);
```

Psalm output (using commit 39a8227):

```
ERROR: InvalidArgument - 12:14 - Argument 1 of processUsers expects array<array-key, User>, array{0: User, 1: string(james)} provided
```

```
processUsers([
  new User('Jane'),
  'james',
]);
```

@daveliddament

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 {…}
}
```

HOW DO WE TELL STATIC ANALYSERS ABOUT GENERICS?

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

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


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

HOW DO WE TELL STATIC ANALYSERS ABOUT GENERICS?

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

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

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

INSTANTIATING A GENERIC CLASS

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

RECAP

```
class Queue() { … }
```

```
class Queue() { … }

$userQueue = new Queue();
```

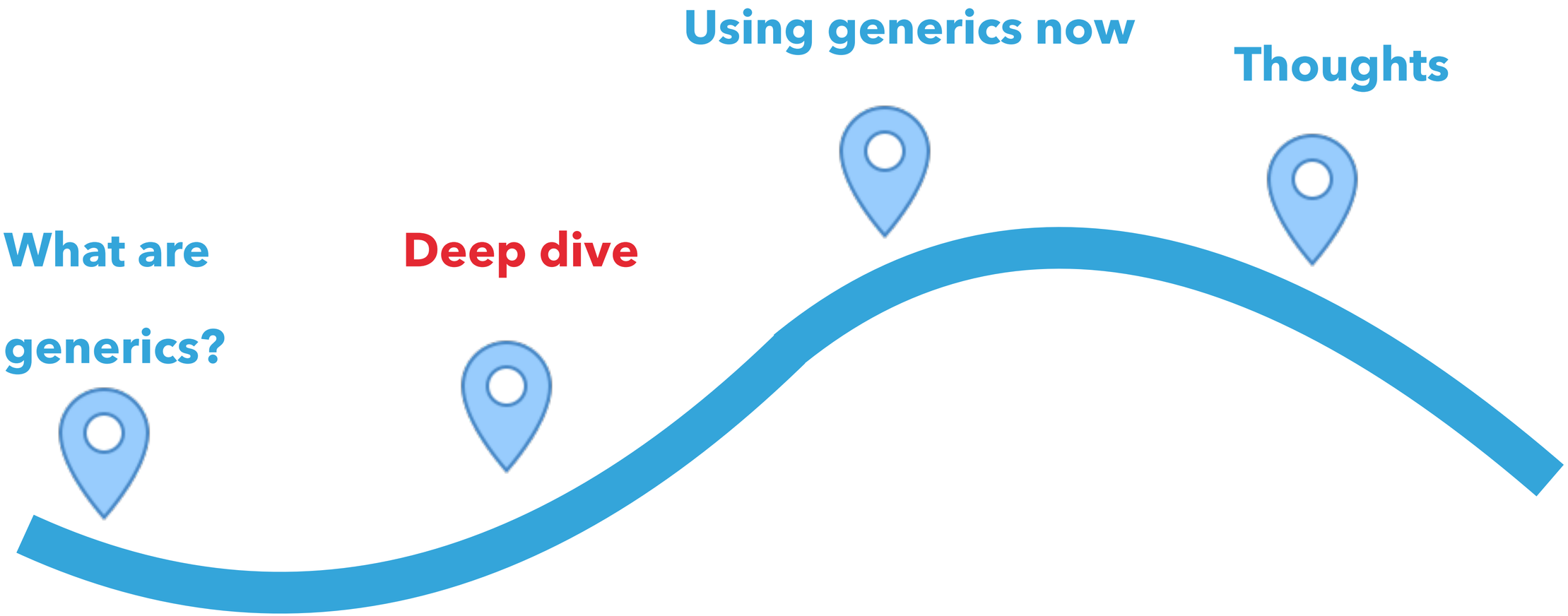
```
/** @template T */
class Queue() { … }

/** @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



```
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);
}
```

```
18
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);
}
```

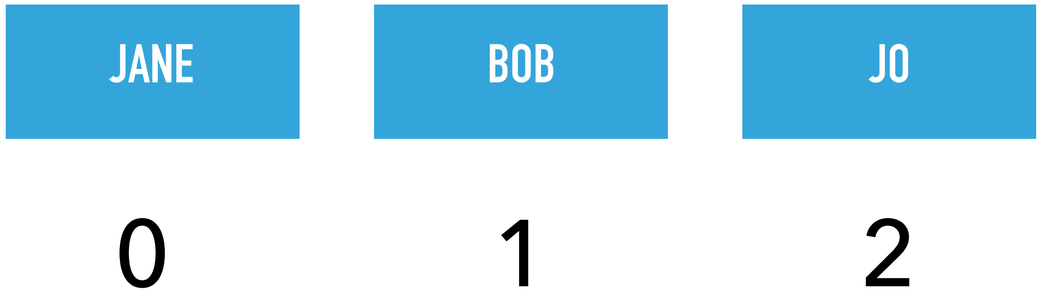
```
class Business {
    /** @return array<string,Employee> */
    public function getEmployees(): array {…}
}
function promote(Employee $employee): void {…}
function welcome(string $name): void {…}

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);
}
```

ARRAYS IN PHP ARE OVERLOADED

- List
- Map or Dictionary
- Shapes or Struts

ARRAY AS A LIST



ARRAY AS A LIST

JANE	BOB	JO
0	1	2

- Keys are of type int
- Zero indexed
- Keys are consecutively numbered

ARRAY AS A LIST

JANE	BOB	JO
0	1	2

- Keys are of type int
- Zero indexed
- Keys are consecutively numbered

`string[]`

ARRAY AS A LIST

JANE	BOB	JO
0	1	2

- Keys are of type int
- Zero indexed
- Keys are consecutively numbered

```
string[]
array<string>
```

ARRAY AS A LIST

JANE	BOB	JO
0	1	2

- Keys are of type int
- Zero indexed
- Keys are consecutively numbered

```
string[]
array<string>
array<int,string>
```

ARRAY AS A LIST

JANE	BOB	JO
0	1	2

- Keys are of type int
- Zero indexed
- Keys are consecutively numbered

```
string[]
array<string>
array<int,string>
list<string>
```

ARRAY AS A LIST

JANE	BOB	JO
0	1	2

- Keys are of type int
- Zero indexed
- Keys are consecutively numbered

```
string[]
array<string>
array<int,string>
list<string>
```

✅

ARRAY AS A MAPS OR DICTIONARIES

"Jane" => JANE

"Bob" => BOB

"Jo" => JO

ARRAY AS A MAPS OR DICTIONARIES

"Jane" => JANE

"Bob" => BOB

"Jo" => JO

```
array<string, User>
```

ARRAY AS A SHAPES OR STRUTS

```
$person = [
   'name' => 'Dave',
   'age' => 21,
   'addresses' => [
      '1 some street, some town',
      '2 another street, another town',
    ],
];
```

ARRAY AS A SHAPES OR STRUTS

```
$person = [
   'name' => 'Dave',
   'age' => 21,
   'addresses' =>
      '1 some street, some town',
      '2 another street, another town',
    ],
];
```

```
class Person
{
   public string $name;
   public int $age;

   /** @var string[] */
   public array $addresses;
}
```

GENERIC FUNCTIONS

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

GENERIC FUNCTIONS

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

GENERIC FUNCTIONS

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

GENERIC FUNCTIONS

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

GENERIC FUNCTIONS

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

INFER TYPE

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

INFER TYPE

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

INFER TYPE

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

INFER TYPE

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

INFER TYPE

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

INFER TYPE

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

```
$users = [new User(“Jane”), new User(“Bob”)]; // User[]
$user = getFirst($users);


/**
 * @template T
 * @param array<T> $items
 * @return T
 */
function getFirst(array $items) {…}
```

```
$users = [new User("Jane"), new User("Bob")]; // User[]
$user = getFirst($users);


/**
 * @template T
 * @param array<T> $items
 * @return T
 */
function getFirst(array $items) {…}
```

```
$users = [new User("Jane"), new User("Bob")]; // User[]
$user = getFirst($users);


/**
 * @template T
 * @param array<T> $items
 * @return T
 */
function getFirst(array $items) {…}
```

```
$users = [new User("Jane"), new User("Bob")]; // User[]
$user = getFirst($users);


/**
 * @template T
 * @param array<T> $items
 * @return T
 */
function getFirst(array $items) {…}
```

```
$users = [new User("Jane"), new User("Bob")]; // User[]
$user = getFirst($users);

/**
 * @template T
 * @param array<T> $items
 * @return T
 */
function getFirst(array $items) {…}
```

```
/** @template T */
class Collection {
 /** @param T[] $items */
 public function __construct($items) {…}
 /** @return T */
 public function next() {…}
}

$collection = new Collection([new User(…), new User(…)]);
$user = $collection->next();
```

```
/** @template T */
class Collection {
 /** @param T[] $items */
 public function __construct($items) {…}
 /** @return T */
 public function next() {…}
}

$collection = new Collection([new User(…), new User(…)]);
$user = $collection->next();
```

```
/** @template T */
class Collection {
 /** @param T[] $items */
 public function __construct($items) {…}
 /** @return T */
 public function next() {…}
}

$collection = new Collection([new User(…), new User(…)]);
$user = $collection->next();
```

```
/** @template T */
class Collection {
 /** @param T[] $items */
 public function __construct($items) {…}
 /** @return T */
 public function next() {…}
}

$collection = new Collection([new User(…), new User(…)]);
$user = $collection->next();
```

```
/** @template T */
class Collection {
  /** @param T[] $items */
  public function __construct($items) {…}
  /** @return T */
  public function next() {…}
}
// No need for docblock
$collection = new Collection([new User(…), new User(…)]);
$user = $collection->next();
```

```
/** @template T */
class Collection {
 /** @param T[] $items */
 public function __construct($items) {…}
 /** @return T */
 public function next() {…}
}
// No need for docblock
$collection = new Collection([new User(…), new User(…)]);
$user = $collection->next();
```

CLASS-STRING

```
class Person {…}

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

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

CLASS-STRING

```
class Person {…}

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

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

CLASS-STRING

```
class Person {…}

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

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

CLASS-STRING

```
class Person {…}

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

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

CLASS-STRING

```
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-STRING

```
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-STRING

```
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-STRING

```
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-STRING

```
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-STRING

```
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 Repository {

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

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

EXTENDING TEMPLATES

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

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

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

EXTENDING TEMPLATES

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

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

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

EXTENDING TEMPLATES

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

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

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

```
/** @extends Repository<Person> */
class PersonRepository extends Repository {
   … Additional Person specific methods …
}
/** @template T */
abstract class Repository {
    /** @return T|null */
    public function findById(int $id) {…}
}
$person = $this->person->findById(2);
```

```
/** @extends Repository<Person> */
class PersonRepository extends Repository {
    … Additional Person specific methods …
}
/** @template T */
abstract class Repository {
    /** @return T|null */
    public function findById(int $id) {…}
}
$person = $this->person->findById(2);
```

```
/** @extends Repository<Person> */
class PersonRepository extends Repository {
    … Additional Person specific methods …
}
/** @template T */
abstract class Repository {
    /** @return T|null */
    public function findById(int $id) {…}
}
$person = $this->person->findById(2);
```

```
/** @extends Repository<Person> */
class PersonRepository extends Repository {
    … Additional Person specific methods …
}
/** @template T */
abstract class Repository {
    /** @return T|null */
    public function findById(int $id) {…}
}
$person = $this->person->findById(2);
```

```
/** @extends Repository<Person> */
class PersonRepository extends Repository {
    … Additional Person specific methods …
}
/** @template T */
abstract class Repository {
    /** @return T|null */
    public function findById(int $id) {…}
}
$person = $this->person->findById(2);
```

```
/** @extends Repository<Person> */
class PersonRepository extends Repository {
    … Additional Person specific methods …
}
/** @template T */
abstract class Repository {
    /** @return T|null */
    public function findById(int $id) {…}
}
$person = $this->person->findById(2);
```

```
/** @extends Repository<Person> */
class PersonRepository extends Repository {
    … Additional Person specific methods …
}
/** @template T */
abstract class Repository {
    /** @return T|null */
    public function findById(int $id) {…}
}
$person = $this->person->findById(2);
```

```
/** @extends Repository<Person> */
class PersonRepository extends Repository {
    … Additional Person specific methods …
}
/** @template T */
abstract class Repository {
    /** @return T|null */
    public function findById(int $id) {…}
}
$person = $this->person->findById(2);
```

```
class Animal { … }

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

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

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

  /** @return class-string<T> */
  public abstract function supports(): string;

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

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

  /** @return class-string<T> */
  public abstract function supports(): string;

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

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

  /** @return class-string<T> */
  public abstract function supports(): string;

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

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

  /** @return class-string<T> */
  public abstract function supports(): string;

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

```
/** @extends AnimalProcessor<Dog> */
class DogProcessor extends AnimalProcessor {
  public function supports(): string {
    return Dog::class;
  }
  public function process($animal): void {
    $animal->bark(); // We know $animal is a dog
  }
}
```

```
/** @extends AnimalProcessor<Dog> */
class DogProcessor extends AnimalProcessor {
  public function supports(): string {
    return Dog::class;
  }
  public function process($animal): void {
    $animal->bark(); // We know $animal is a dog
  }
}
```

```
/** @extends AnimalProcessor<Dog> */
class DogProcessor extends AnimalProcessor {
  public function supports(): string {
    return Dog::class;
  }
  public function process($animal): void {
    $animal->bark(); // We know $animal is a dog
  }
}
```

```
/** @extends AnimalProcessor<Dog> */
class DogProcessor extends AnimalProcessor {
  public function supports(): string {
    return Dog::class;
  }
  public function process($animal): void {
    $animal->bark(); // We know $animal is a dog
  }
}
```

```
/** @extends AnimalProcessor<Dog> */
class DogProcessor extends AnimalProcessor {
  public function supports(): string {
    return Dog::class;
  }
  public function process($animal): void {
    $animal->bark(); // We know $animal is a dog
  }
}
```

```
/** @extends AnimalProcessor<Dog> */
class DogProcessor extends AnimalProcessor {
  public function supports(): string {
    return Cat::class; // Cats aren't Dogs!
  }
  public function process($animal): void {
    $animal->meow(); // Dogs can't meow!
  }
}
```

```
/** @extends AnimalProcessor<Dog> */
class DogProcessor extends AnimalProcessor {
  public function supports(): string {
    return Cat::class; // Cats aren't Dogs!
  }
  public function process($animal): void {
    $animal->meow(); // Dogs can't meow!
  }
}
```

```
/** @extends AnimalProcessor<Dog> */
class DogProcessor extends AnimalProcessor {
  public function supports(): string {
    return Cat::class; // Cats aren't Dogs!
  }
  public function process($animal): void {
    $animal->meow(); // Dogs can't meow!
  }
}
```

```
/** @extends AnimalProcessor<Dog> */
class DogProcessor extends AnimalProcessor {
  public function supports(): string {
    return Cat::class; // Cats aren't Dogs!
  }
  public function process($animal): void {
    $animal->meow(); // Dogs can't meow!
  }
}
```

```
/** @template T */
class AnimalProcessor { … }

/** @extends AnimalProcessor<Car> */
class CarProcessor extends AnimalProcessor { … }
```

```
/** @template T */
class AnimalProcessor { … }

/** @extends AnimalProcessor<Car> */
class CarProcessor extends AnimalProcessor { … }
```

```
/** @template T */
class AnimalProcessor { … }

/** @extends AnimalProcessor<Car> */
class CarProcessor extends AnimalProcessor { … }
```

```
/** @template T of Animal */
class AnimalProcessor { … }

/** @extends AnimalProcessor<Car> */
class CarProcessor extends AnimalProcessor { … }

/** @extends AnimalProcessor<Cat> */
class CatProcessor extends AnimalProcessor { … }
```

```
/** @template T of Animal */
class AnimalProcessor { … }

/** @extends AnimalProcessor<Car> */
class CarProcessor extends AnimalProcessor { … }

/** @extends AnimalProcessor<Cat> */
class CatProcessor extends AnimalProcessor { … }
```

```
/** @template T of Animal */
class AnimalProcessor { … }

/** @extends AnimalProcessor<Car> */
class CarProcessor extends AnimalProcessor { … }

/** @extends AnimalProcessor<Cat> */
class CatProcessor extends AnimalProcessor { … }
```

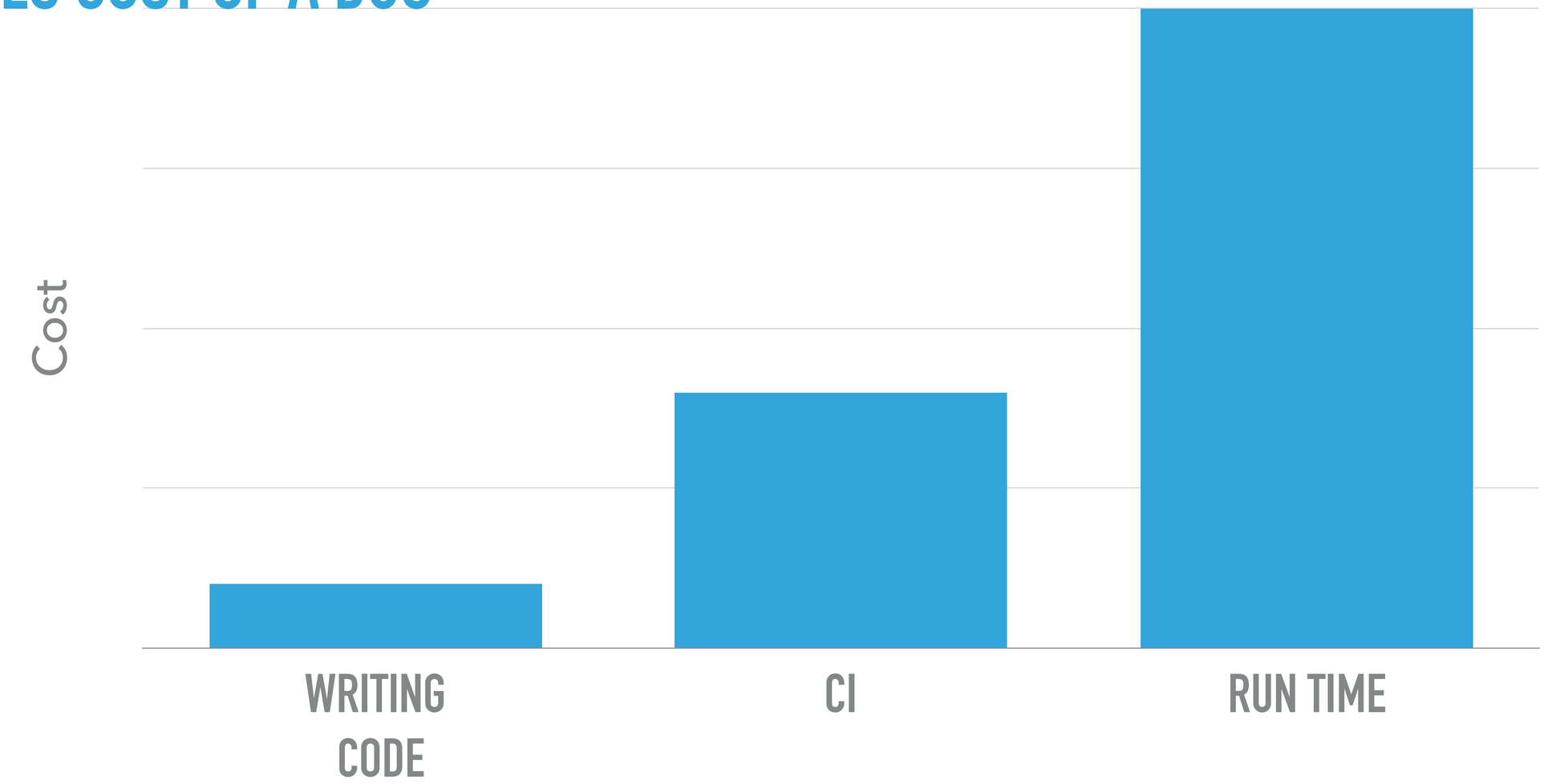
```
/** @template T of Animal */
class AnimalProcessor { … }

/** @extends AnimalProcessor<Car> */
class CarProcessor extends AnimalProcessor { … }

/** @extends AnimalProcessor<Cat> */
class CatProcessor extends AnimalProcessor { … }
```

HOW DOES THIS HELP US?

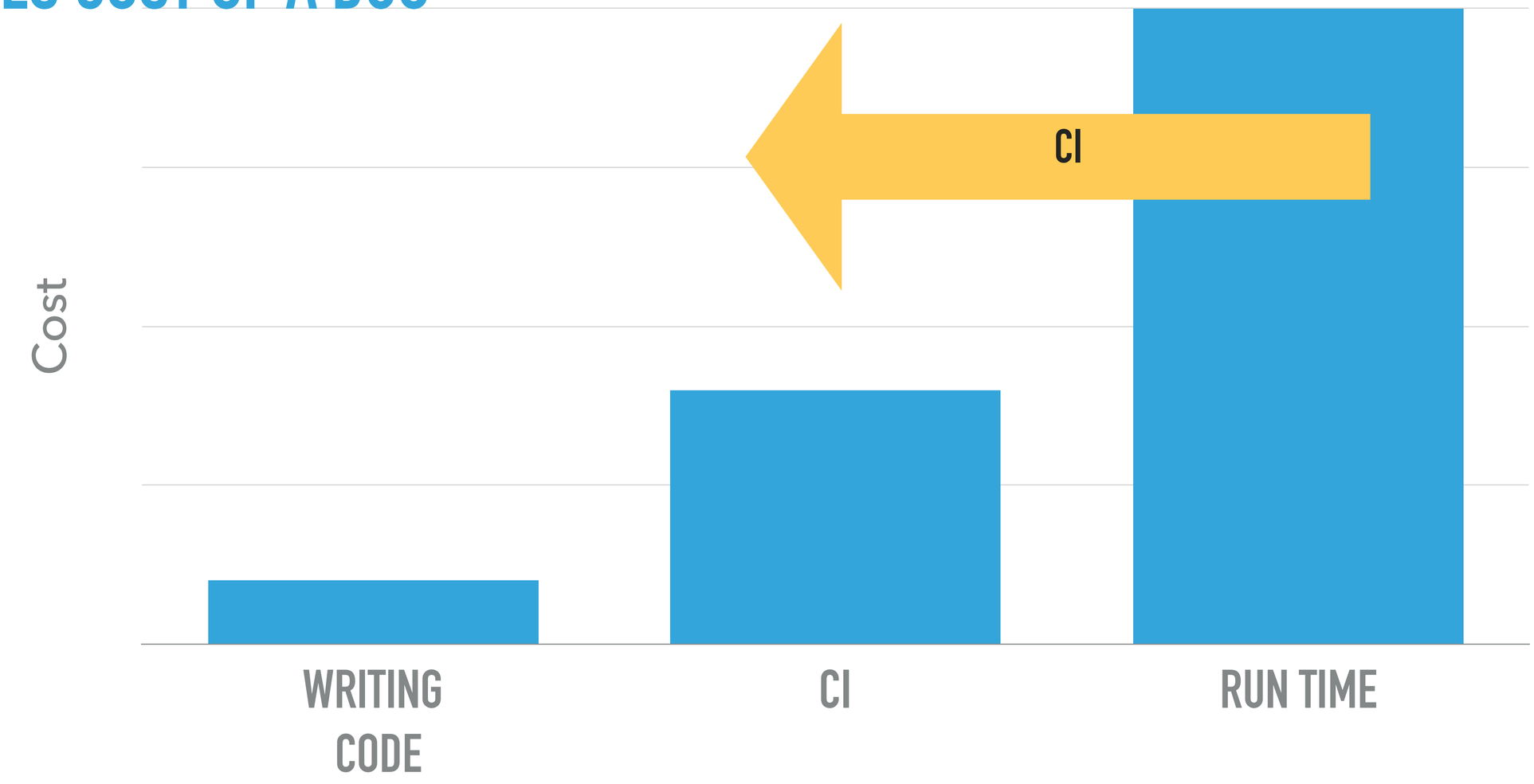
1. COMMUNICATES ADDITIONAL TYPE INFORMATION

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

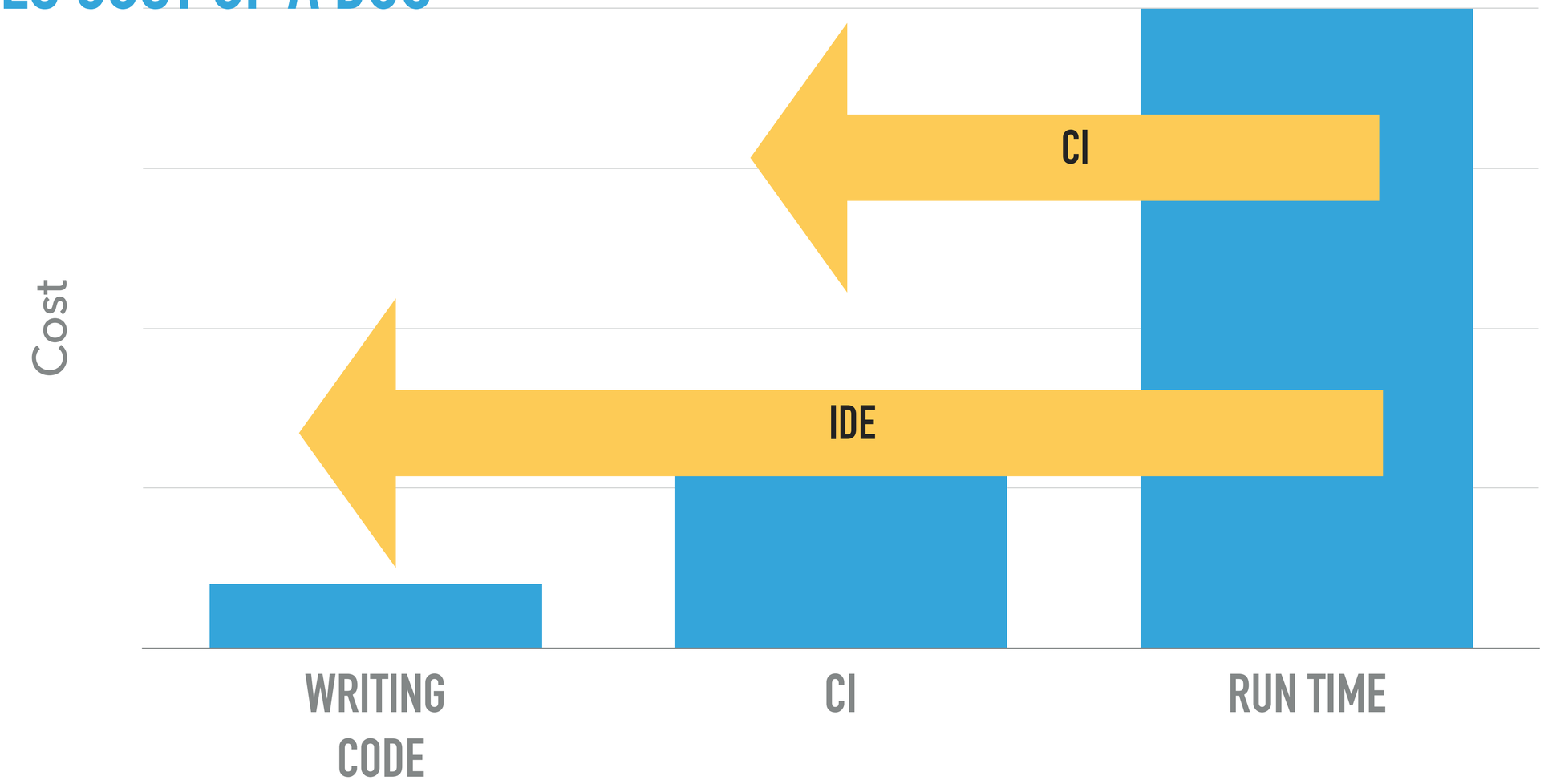
2. REDUCES COST OF A BUG



2. REDUCES COST OF A BUG



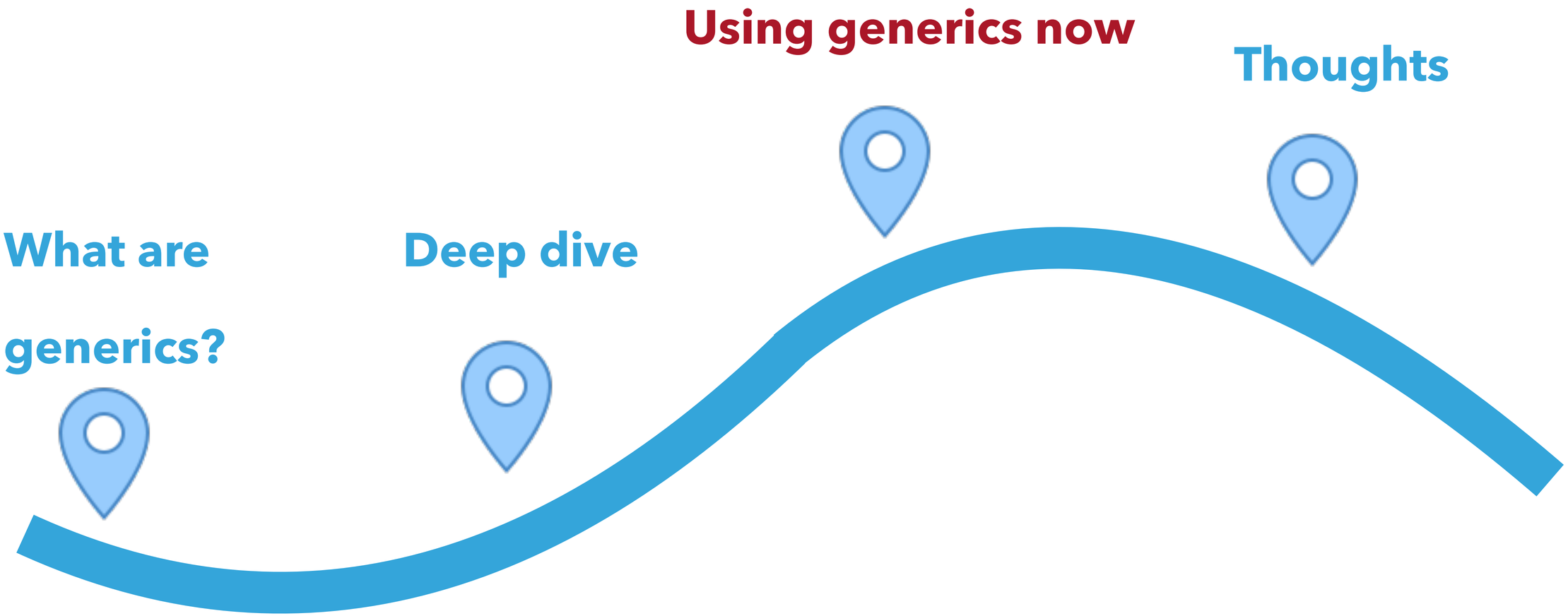
2. REDUCES COST OF A BUG



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

Psalm

Provide type information for everything including generics

Psalm

Provide type information for everything including generics

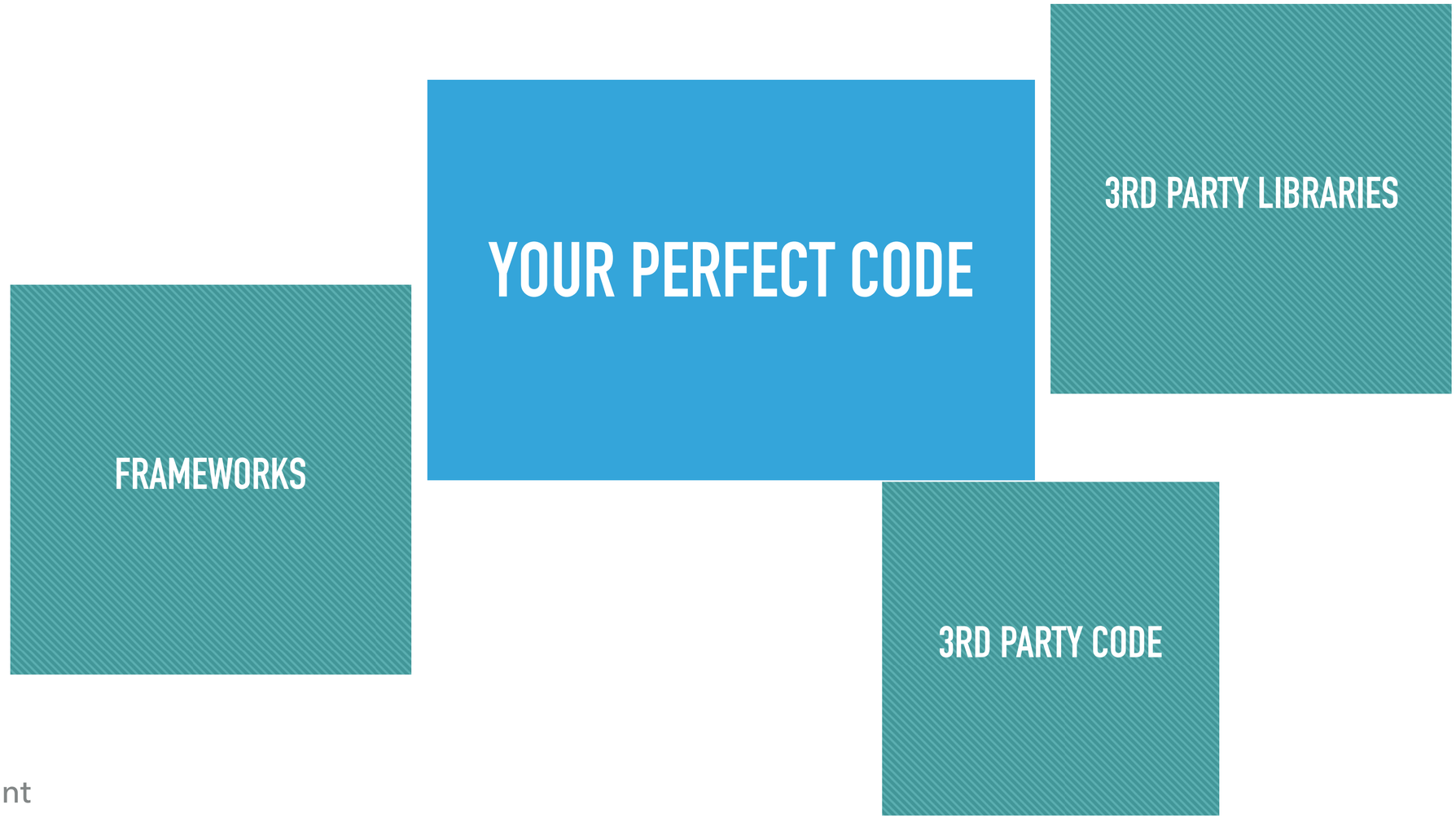
Psalm

Totally Typed mode

INTEGRATING WITH 3RD PARTY CODE

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();

}

… in our code …

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

ADAPTORS FOR 3RD PARTY LIBRARIES: PROBLEM

```
interface Hasher {

    /**
     * @return string
     */
    public function encode();

}

… in our code …

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

ADAPTORS FOR 3RD PARTY LIBRARIES: PROBLEM

```
interface Hasher {

    /**
     * @return string
     */
    public function encode();

}

… in our code …

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

ADAPTORS FOR 3RD PARTY LIBRARIES: A SOLUTION

```
class CleanHasher {

  /** @var Hasher $hasher */
  private $hasher;

  … constructor to inject Hasher …

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


… in our code …

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

ADAPTORS FOR 3RD PARTY LIBRARIES: A SOLUTION

```
class CleanHasher {

  /** @var Hasher $hasher */
  private $hasher;

  … constructor to inject Hasher …

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


… in our code …

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

ADAPTORS FOR 3RD PARTY LIBRARIES: A SOLUTION

```
class CleanHasher {

  /** @var Hasher $hasher */
  private $hasher;

  … constructor to inject Hasher …

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


… in our code …

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

ADAPTORS FOR 3RD PARTY LIBRARIES: A SOLUTION

```
class CleanHasher {

  /** @var Hasher $hasher */
  private $hasher;

  … constructor to inject Hasher …

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


… in our code …

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

ADAPTORS FOR 3RD PARTY LIBRARIES: A SOLUTION

```
class CleanHasher {

  /** @var Hasher $hasher */
  private $hasher;

  … constructor to inject Hasher …

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


… in our code …

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

ADAPTORS FOR 3RD PARTY LIBRARIES: A SOLUTION

```
class CleanHasher {

  /** @var Hasher $hasher */
  private $hasher;

  … constructor to inject Hasher …

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


… in our code …

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

ADAPTORS FOR 3RD PARTY LIBRARIES: A SOLUTION

```
class CleanHasher {

  /** @var Hasher $hasher */
  private $hasher;

  … constructor to inject Hasher …

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


… in our code …

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

ADAPTORS FOR 3RD PARTY LIBRARIES: A SOLUTION

```
class CleanHasher {

  /** @var Hasher $hasher */
  private $hasher;

  … constructor to inject Hasher …

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


… in our code …

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

ADAPTORS FOR 3RD PARTY LIBRARIES: A SOLUTION

```
class CleanHasher {

  /** @var Hasher $hasher */
  private $hasher;

  … constructor to inject Hasher …

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


… in our code …

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

USING STUBS

USING STUBS

```
namespace ThirdParty\DI;

class DependencyInjection
{

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

USING STUBS

USING STUBS

Stubs/ThirdParty/DI.php

USING STUBS

Stubs/ThirdParty/DI.php

```
namespace ThirdParty\DI;

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

USING STUBS

Stubs/ThirdParty/DI.php

```
namespace ThirdParty\DI;

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

USING STUBS

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: DANGER IF YOU MAKE A MISTAKE

Stubs/ThirdParty/DI.php

```
namespace ThirdParty\DI;

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

STUBS: DANGER IF YOU MAKE A MISTAKE

Stubs/ThirdParty/DI.php

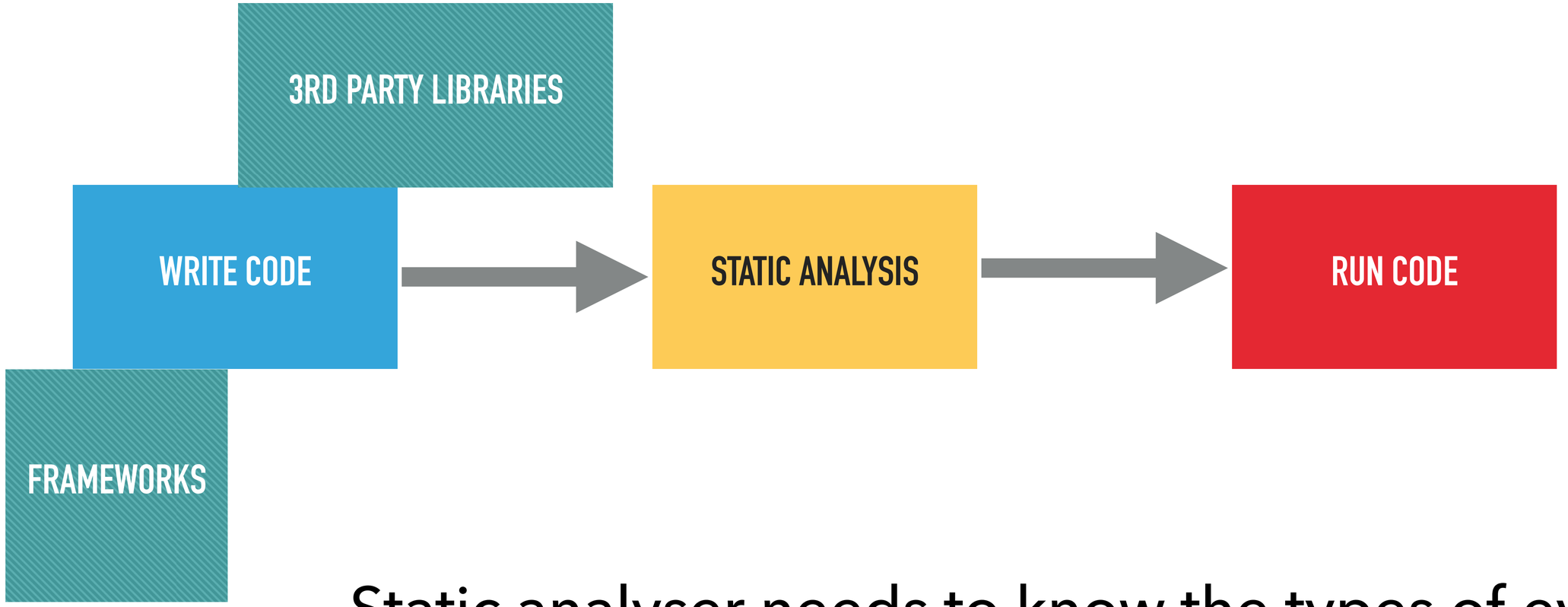
```
namespace ThirdParty\DI;

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

STATIC ANALYSER PLUGINS

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

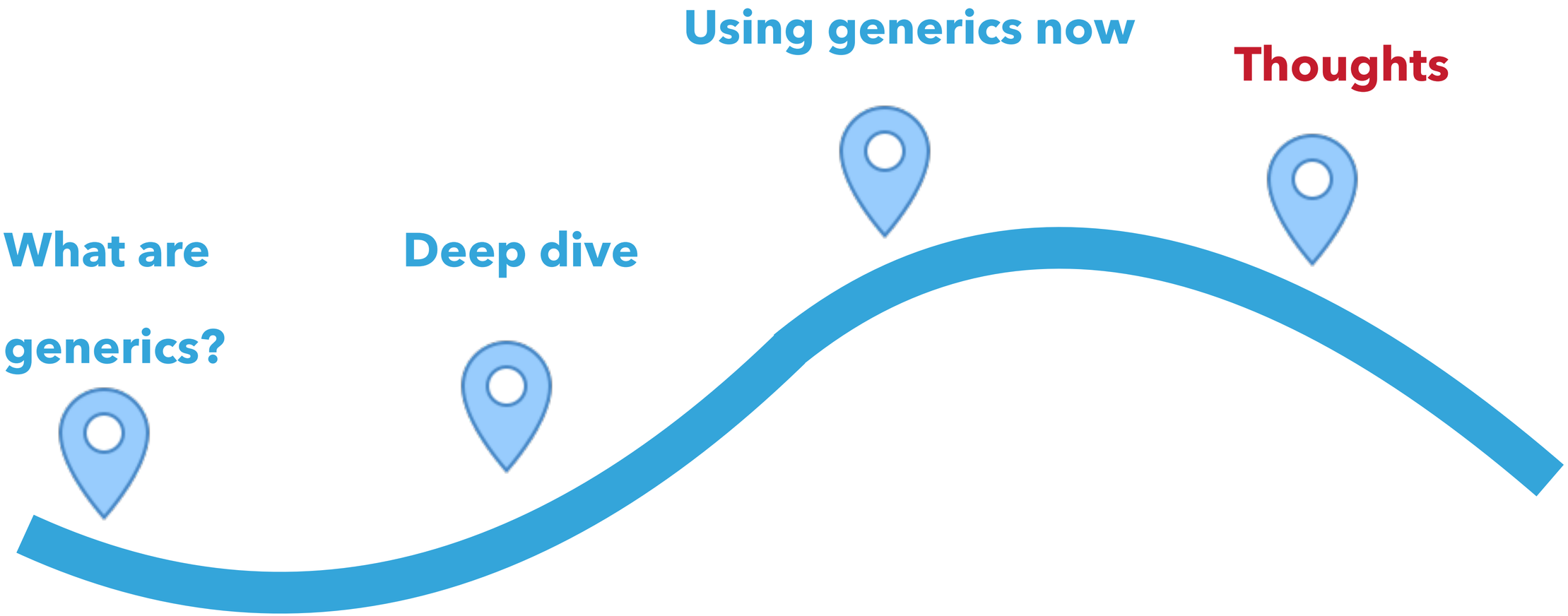
USING GENERICS NOW

Static analyser needs to know the types of everything



Static analyser needs to know the types of everything

AGENDA



PHP GENERICS NOW (ALMOST)

IDE SUPPORT

```
interface  Employee
{
    public function getName(): string;
}

/** @var array<string,Employee> $employees */
$employees = [];

foreach ($employees as $employee) {
    $employee->getName(
}
```

$employee mixed

Namespace:

IDE SUPPORT

```
interface  Employee
{
    public function getName(): string;
}

/** @var array<string,Employee> $employees */
$employees = [];

foreach ($employees as $employee) {
    $employee->getName(
}
```

$employee mixed

Namespace:

IDE SUPPORT

```
interface  Employee
{
    public function getName(): string;
}

/** @var array<string,Employee> $employees */
$employees = [];

foreach ($employees as $employee) {
    $employee->getName(
}
```

$employee mixed

Namespace:

MITIGATE AGAINST NON SUPPORT

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

MITIGATE AGAINST NON SUPPORT

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

MITIGATE AGAINST NON SUPPORT

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

WE NEED A STANDARD

WE NEED A STANDARD

WE NEED A STANDARD

IMPLEMENTING A STANDARD

IMPLEMENTING A STANDARD

- Full language support

IMPLEMENTING A STANDARD

- Full language support
- PSR

IMPLEMENTING A STANDARD

- Full language support
- PSR
- AST?

IMPLEMENTING A STANDARD

- Full language support
- PSR
- AST?

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

IMPLEMENTING A STANDARD

PHP CODE

- Full language support
- PSR
- AST?

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

IMPLEMENTING A STANDARD

- Full language support
- PSR
- AST?

PHP CODE

TOKENISER

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

IMPLEMENTING A STANDARD

- Full language support
- PSR
- AST?

PHP CODE

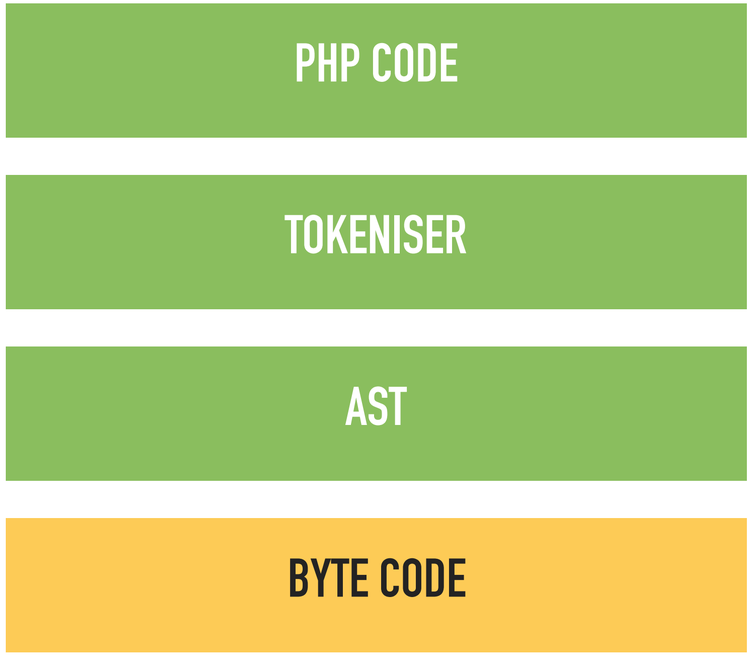
TOKENISER

AST

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

IMPLEMENTING A STANDARD

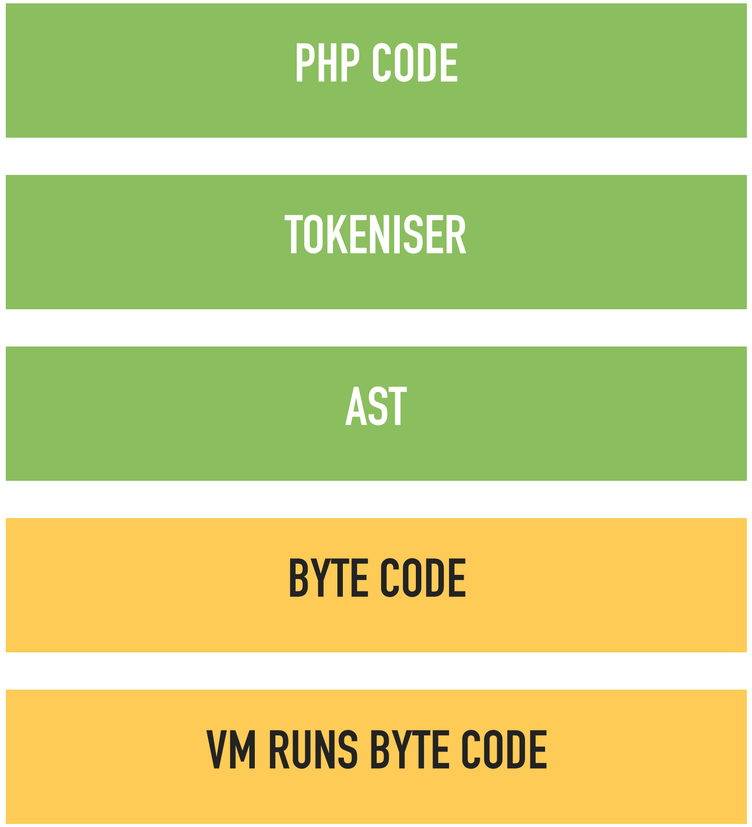
- Full language support
- PSR
- AST?




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

IMPLEMENTING A STANDARD

- Full language support
- PSR
- AST?




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

NEED TRUST TOOLS: NO RUN TIME CHECKS

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

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

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

NEED TRUST TOOLS: NO RUN TIME CHECKS

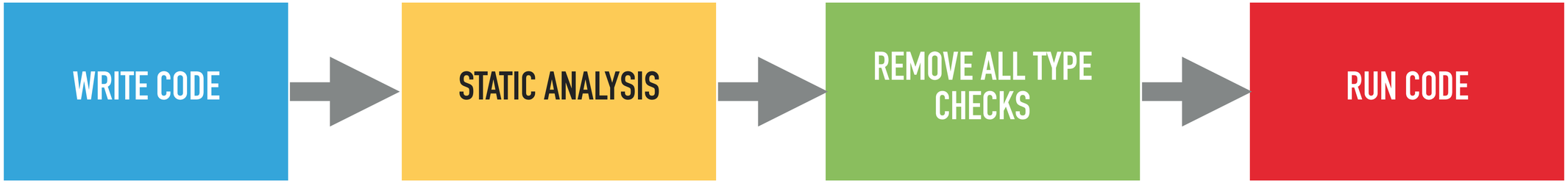
```
/** @template T */
class Queue     {

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

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

NEED TRUST TOOLS: NO RUN TIME CHECKS

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

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

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

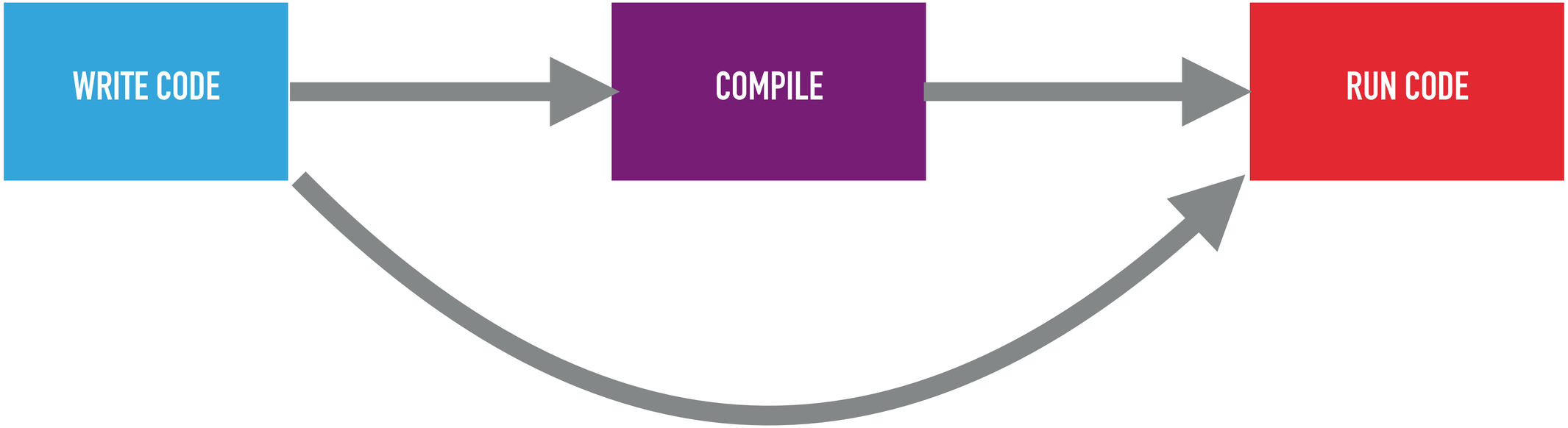
ALL BETS ARE OFF IF THERE ARE IS ANY MISSING OR INCORRECT TYPE INFORMATION

THE END OF RUN TIME CHECKS?

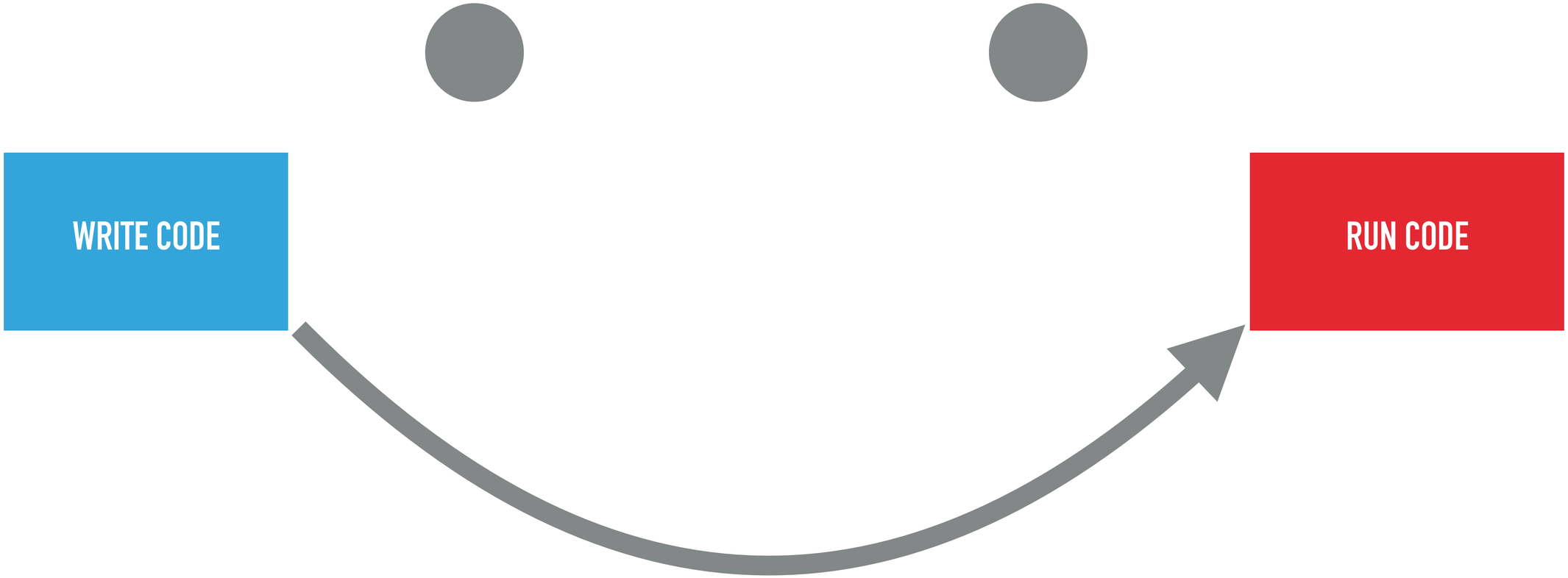


WHY NOT JUST USE JAVA?

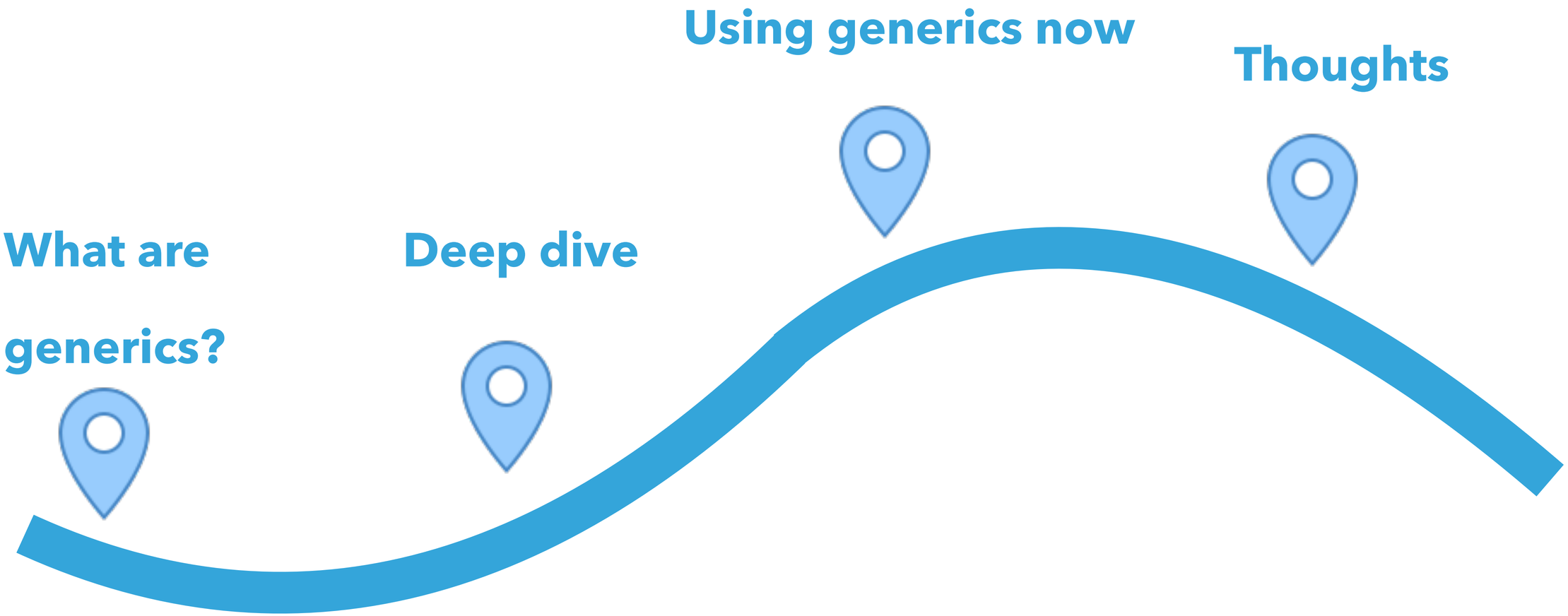
WHY NOT JUST USE JAVA?



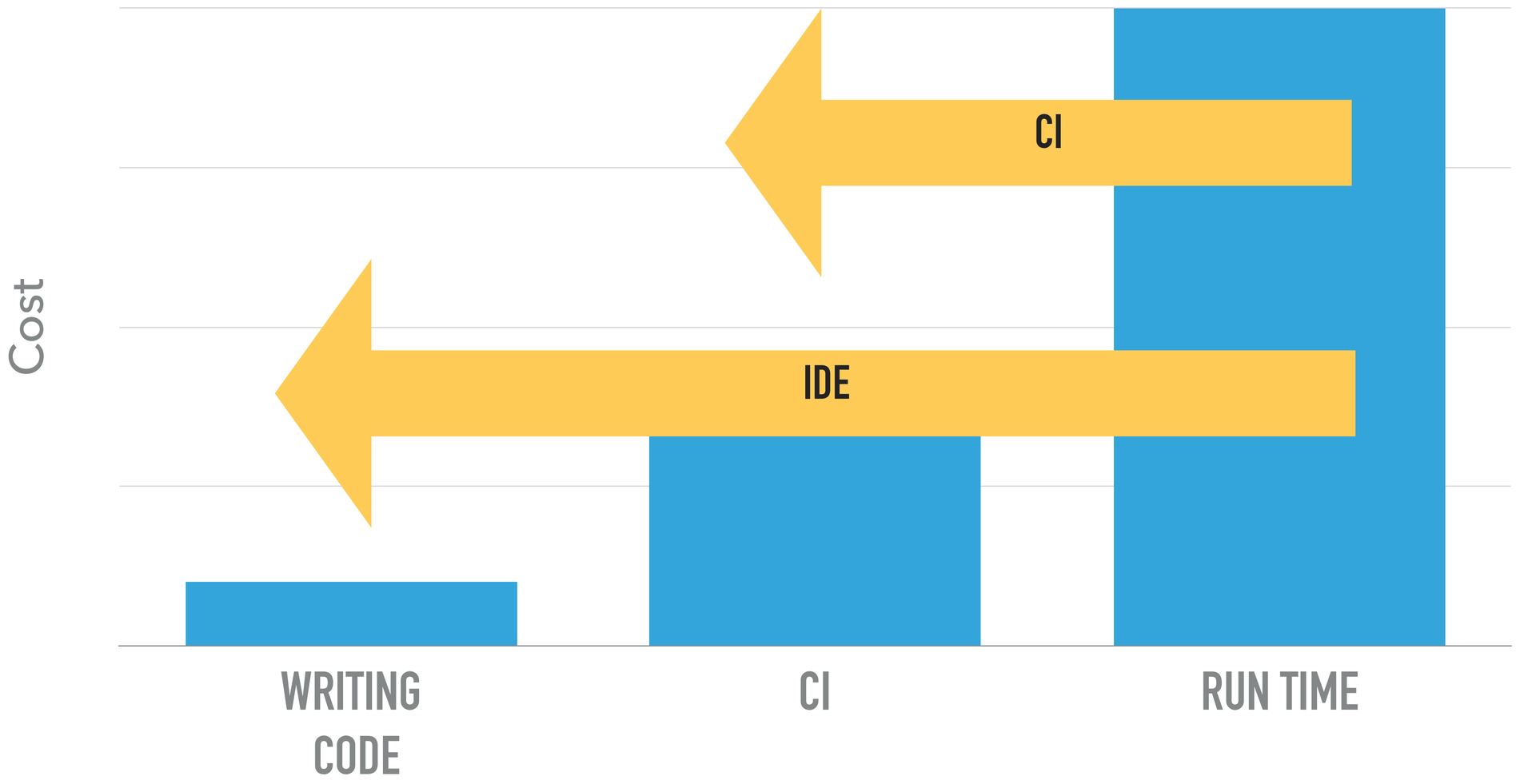
WHY NOT JUST USE JAVA?



AGENDA



ADD CLARITY TO CODE. FIND SOME BUGS EARLIER.



USING GENERICS NOW

Psalm

STANDARDS

Dave Liddament

Lamp Bristol

Organise PHP-SW and Bristol PHP Training

Author of Static Analysis Results Baseliner (SARB)

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

@daveliddament

Dave Liddament

Lamp Bristol

Thank you for listening

Organise PHP-SW and Bristol PHP Training

Author of Static Analysis Results Baseliner (SARB)

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

@daveliddament

QUESTIONS