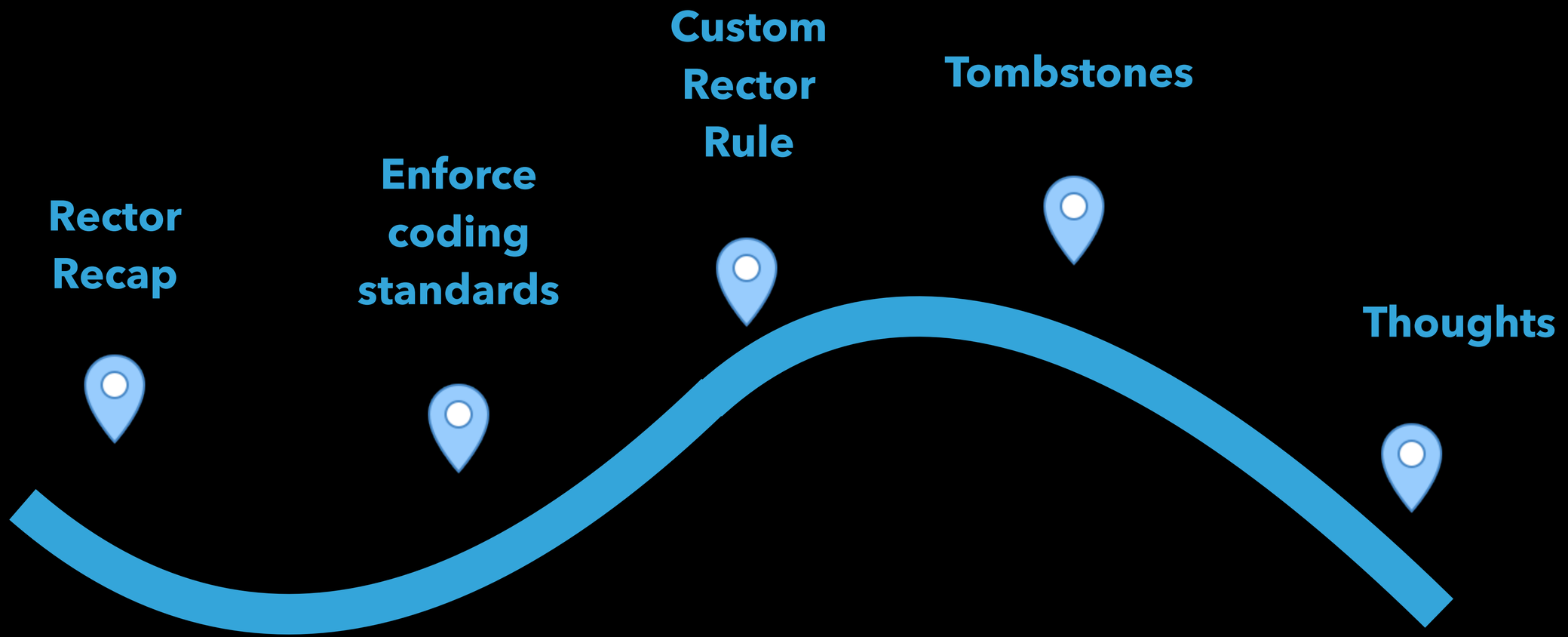
Speaker & session announcement Dutch PHP Conference

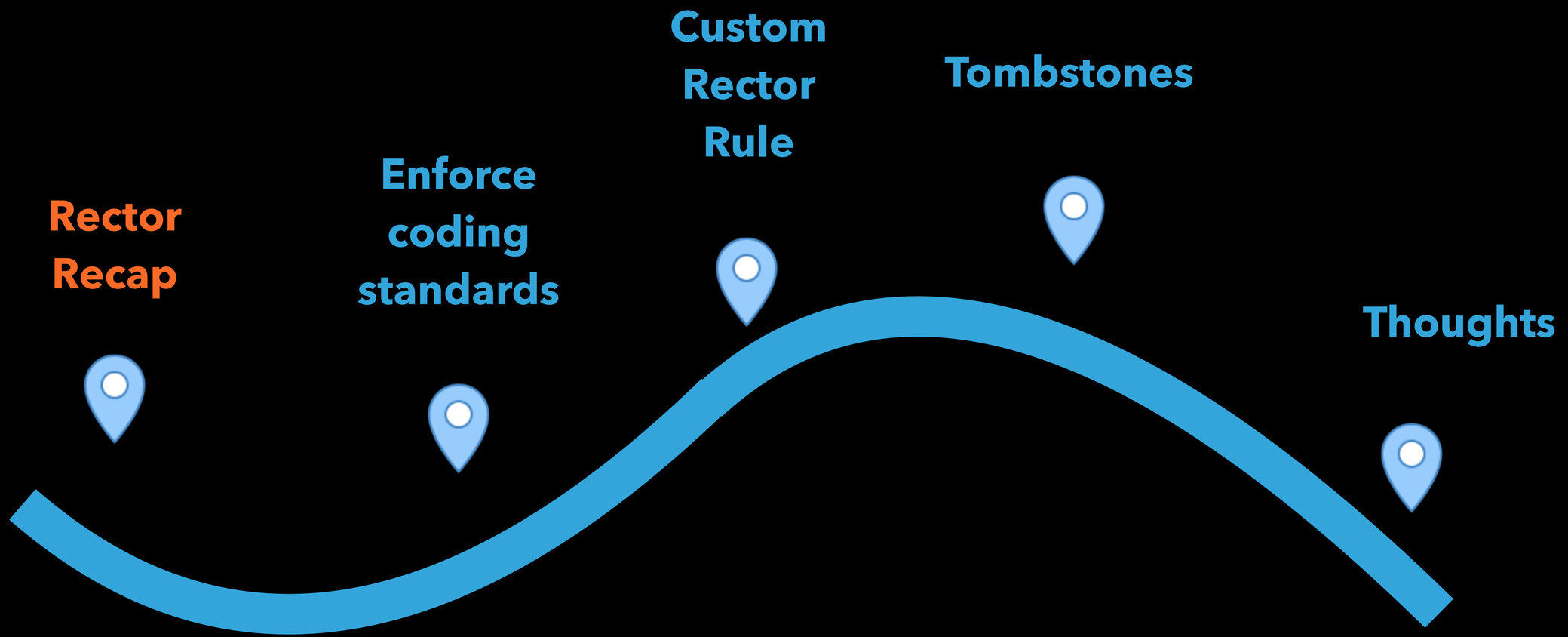
# Rector Beyond Upgrades: Transforming Your Workflow

Amsterdam, The Netherlands

Dave Liddament

https://getrector.com/


Rector Recap
Enforce coding standards
Custom Rector Rule
Tombstones
Thoughts


Rector Recap
Enforce coding standards
Custom Rector Rule
Tombstones
Thoughts

# 824 Rules (and counting)

## RenameMethodRector

Turns method names to new ones.

🔧 **configure it!**

- class: `Rector\Renaming\Rector\MethodCall\RenameMethodRector`

```
 $someObject = new SomeExampleClass;
-$someObject->oldMethod();
+$someObject->newMethod();
```

```
composer require --dev rector/rector
```

```
vendor/bin/rector
```

```
No "rector.php" config found. Should we
generate it for you? [yes]:
 > yes
```

```
 [OK] The config is added now. Re-run command
to make Rector do the work!
```

```php
<?php

use Rector\Config\RectorConfig;
use Rector\Renaming\Rector\MethodCall\RenameMethodRector;
use Rector\Renaming\ValueObject\MethodCallRename;

return RectorConfig::configure()

    ->withPaths([
        __DIR__ . '/src',
    ])

    ->withConfiguredRule(
        RenameMethodRector::class,
        [
            new MethodCallRename(
                App\Framework\Model::class,
                'belongsTo',
                'belongs')
        ]
    );
```

```
 7/7 [▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓] 100%
2 files with changes
====================

1) src/Framework/Model.php:5

    ---------- begin diff ----------
@@ @@

 abstract class Model
 {
-    public function belongsTo(string $model): BelongsTo
+    public function belongs(string $model): BelongsTo
     {
         return new BelongsTo($model);
     }
    ----------- end diff -----------

Applied rules:
 * RenameMethodRector
```

## ChangeIfElseValueAssignToEarlyReturnRector

Change if/else value to early return

- class: `Rector\EarlyReturn\Rector\If_\ChangeIfElseValueAssignToEarlyReturnRector`

```
 class SomeClass
 {
     public function run()
     {
         if ($this->hasDocBlock($tokens, $index)) {
-            $docToken = $tokens[$this->getDocBlockIndex($tokens, $index)];
-        } else {
-            $docToken = null;
+            return $tokens[$this->getDocBlockIndex($tokens, $index)];
         }
-
-        return $docToken;
+        return null;
     }
 }
```

# Rulesets:  A set of rules

Rector: PHP Language upgrades

https://github.com/rectorphp/rector-symfony

https://github.com/driftingly/rector-laravel

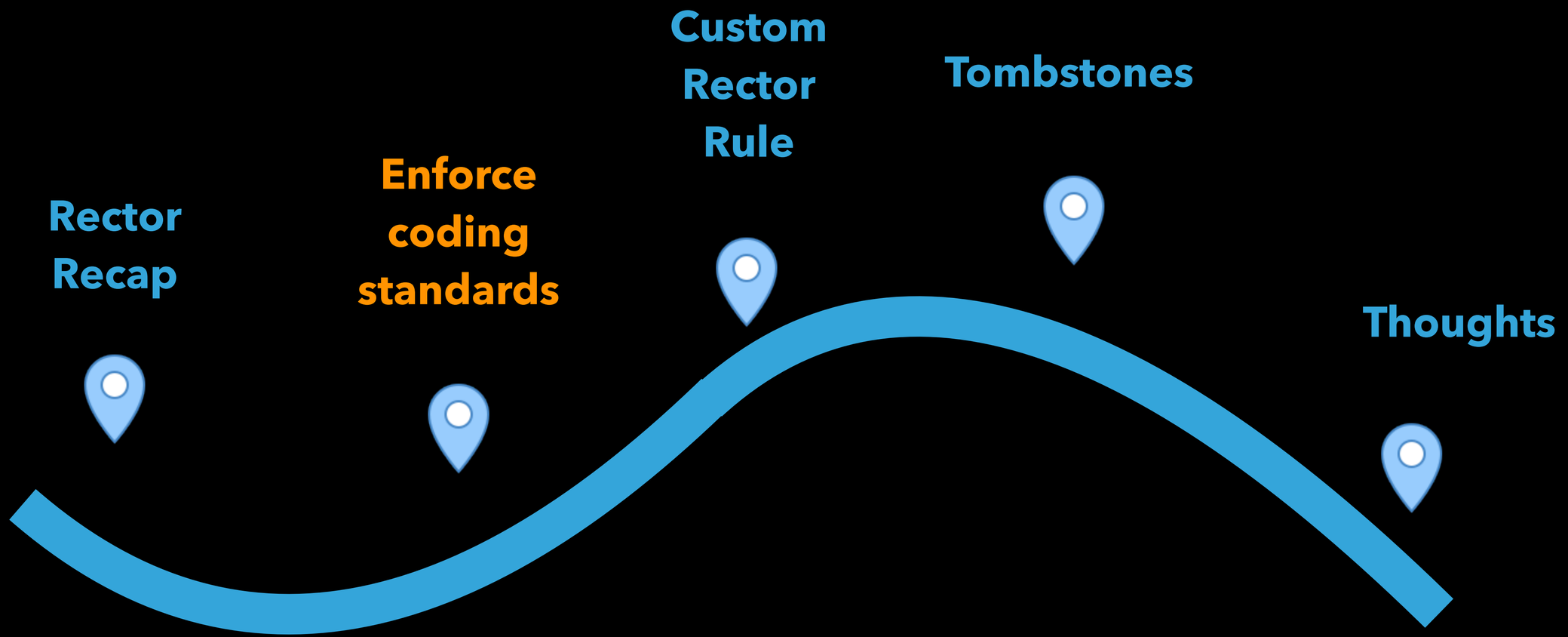
# rector.php

```php
<?php

declare(strict_types=1);

use Rector\Config\RectorConfig;
use Rector\Set\ValueObject\SetList;

return RectorConfig::configure()
    ->withPaths(…)
    ->withSets([
        SetList::PHP_85,
    ]);
```

# Levels: Step by step

```
return RectorConfig::configure()
    ->withPaths(…)
    ->withTypeCoverageLevel(10)
    ->withCodeQuality(5)

    ->and others…
);
```

- Upgrades
- One off Refactors
- Downgrades
- Automate code review
- Investigation work


Rector Recap
Enforce coding standards
Custom Rector Rule
Tombstones
Thoughts

# Coding standards

- PSR 12
- Use Early Return pattern
- Use named arguments for #[Route]

# Coding standards

- PSR 12  

- Use Early Return pattern 
- Use named arguments for #[Route] 

# Enforcing Coding standards


Increasing human input

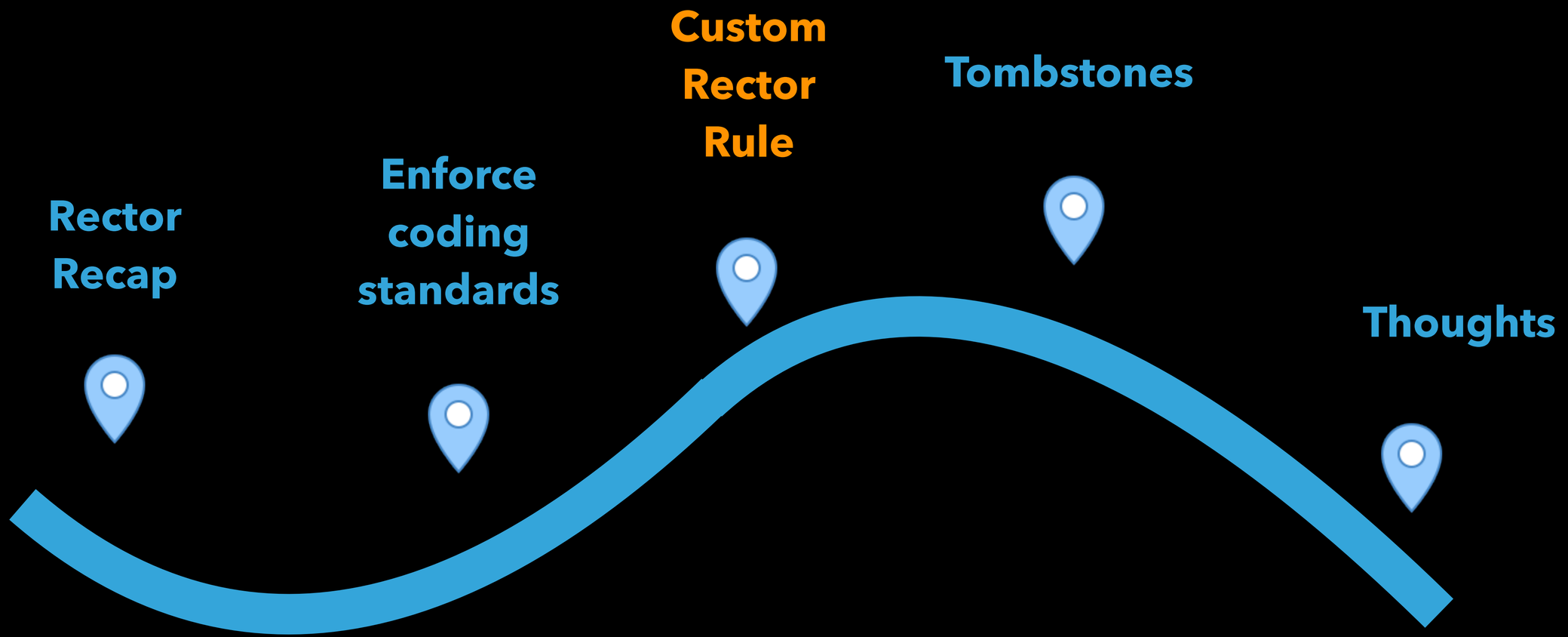
# rector-ci.php

```
return RectorConfig::configure()

    ->withPaths([
        __DIR__ . '/src',
    ])

    ->withRules([
        ChangeIfElseValueAssignToEarlyReturnRector::class,
         … any other rules …
    ]);
```

---

```
vendor/bin/rector --config rector-ci.php
```

Notes:
1.Run cs fixer after rector
2.On CI run with --dry-run


Rector Recap
Enforce coding standards
Custom Rector Rule
Tombstones
Thoughts

# Coding standards

- PSR 12
- Use Early Return pattern
- Use named arguments for #[Route]

Old

```
class PersonController
{
  #[Route('/people', 'GET')]
  public function users(): Response
  {
  }
}
```

New

```
class PersonController
{
  #[Route(url: '/people', method: 'GET')]
  public function users(): Response
  {
  }
}
```

# No Change: 1

```
class PersonController
{
  #[Route(method: ‘GET’, url: ’/people’)]
  public function users(): Response
  {
  }
}
```

# No Change: 2

```
class PersonController
{
  #[IsGranted('view')]
  public function users(): Response
  {
  }
}
```

```
$ vendor/bin/rector custom-rule


 What is the name of the rule class (e.g. "LegacyCallToDbalMethodCall")?:
 > RouterUseNamedArguments

Generated files
===============

 * utils/rector/src/Rector/FixModelMappingsRector.php
 * utils/rector/tests/Rector/FixModelMappingsRector/Fixture/some_class.php.inc
 * utils/rector/tests/Rector/FixModelMappingsRector/config/configured_rule.php
 * utils/rector/tests/Rector/FixModelMappingsRector/FixModelMappingsRectorTest.php


 [OK] Base for the "RouterUserNamedArguments" rule was created. Now you can fill the
missing parts


 [OK] We also update composer.json autoload-dev, to load Rector rules. Now run
"composer dump-autoload" to update paths
```

Old

```
class PersonController
{
  #[Route('/people', 'GET')]
  public function users(): Response
  {
  }
}
```

---

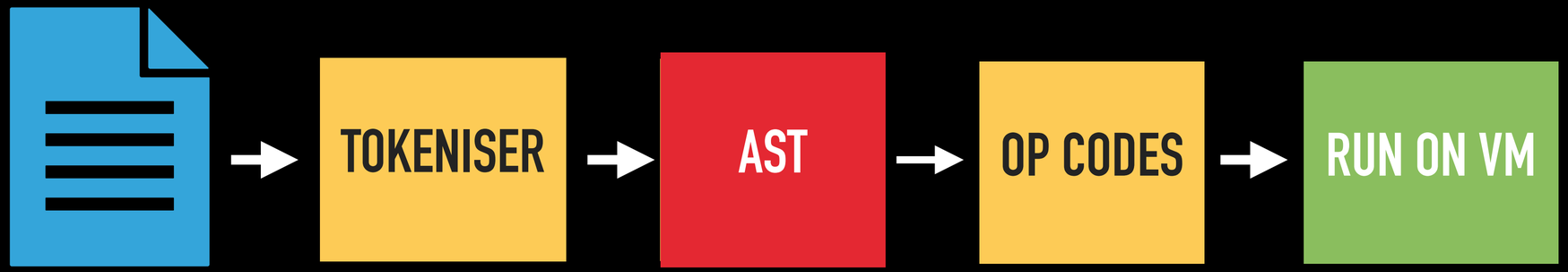
New

```
class PersonController
{
  #[Route(url: '/people', method: 'GET')]
  public function users(): Response
  {
  }
}
```

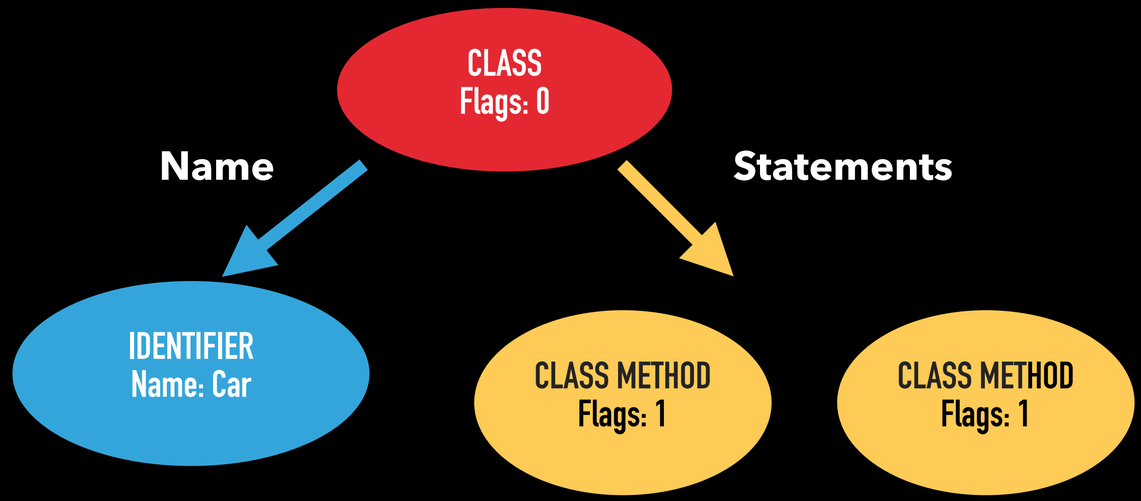
No Change

```
class PersonController
{
    #[Route(method: 'GET', url: '/people')]
    public function users(): Response
    {
    }
}
```

```
RouterUseNamedArguments extends AbstractRector
{

    public function getNodeTypes(): array

    public function refactor(Node $node): ?Node
}
```

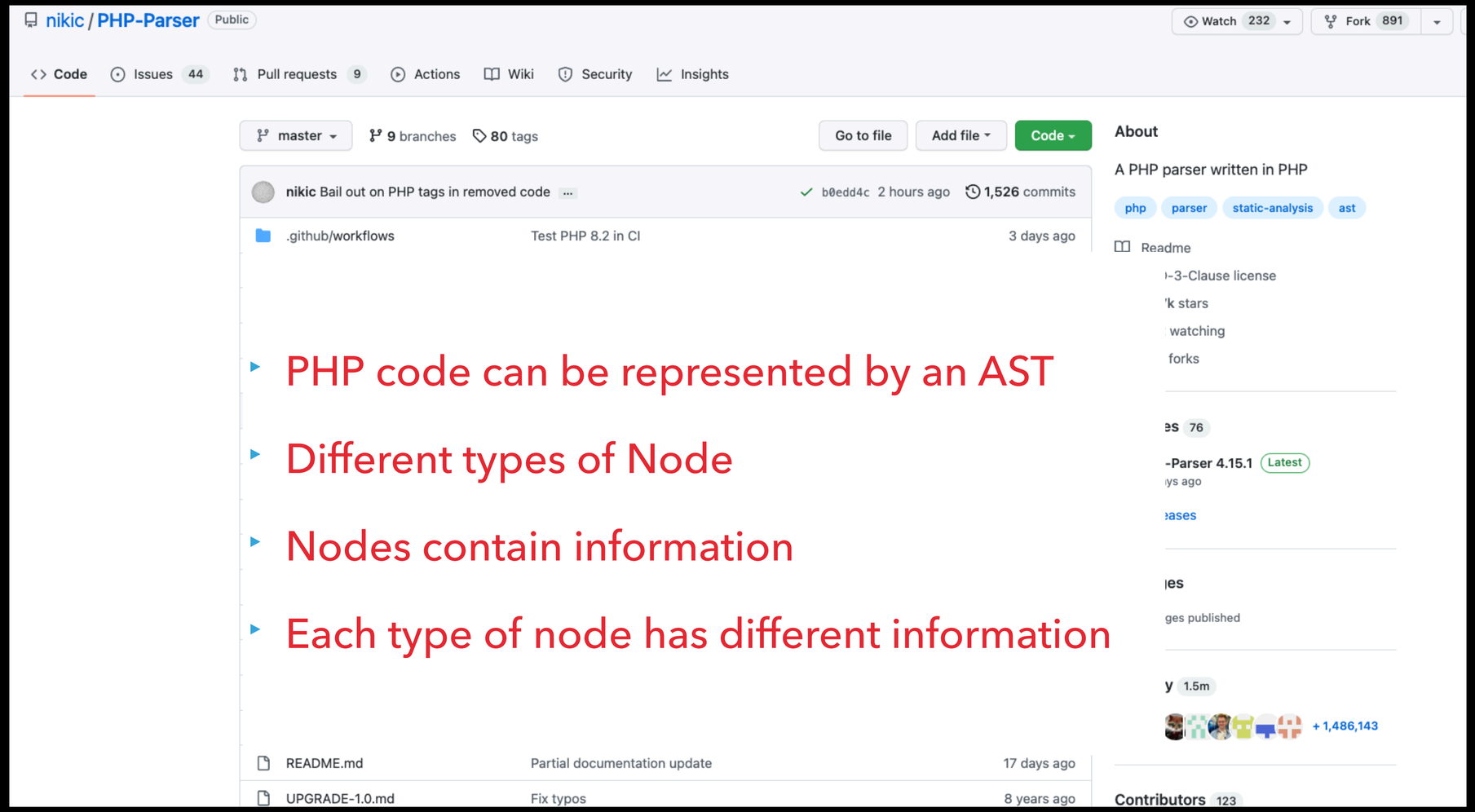

TOKENISER
AST
OP CODES
RUN ON VM

```
class Car
{
    public function users() {…}

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



```
public function user ()
{
    return $this->belongsTo('User');
}
```

# https://github.com/nikic/PHP-Parser



# www.getrector.com/ast

RECTOR Try Rector Docs Hire Team Find Rule Blog Contact AST

## 1. Write short PHP code you want to understand

```php
<?php

if ($condition === 'demo') {
    return true;
}
```

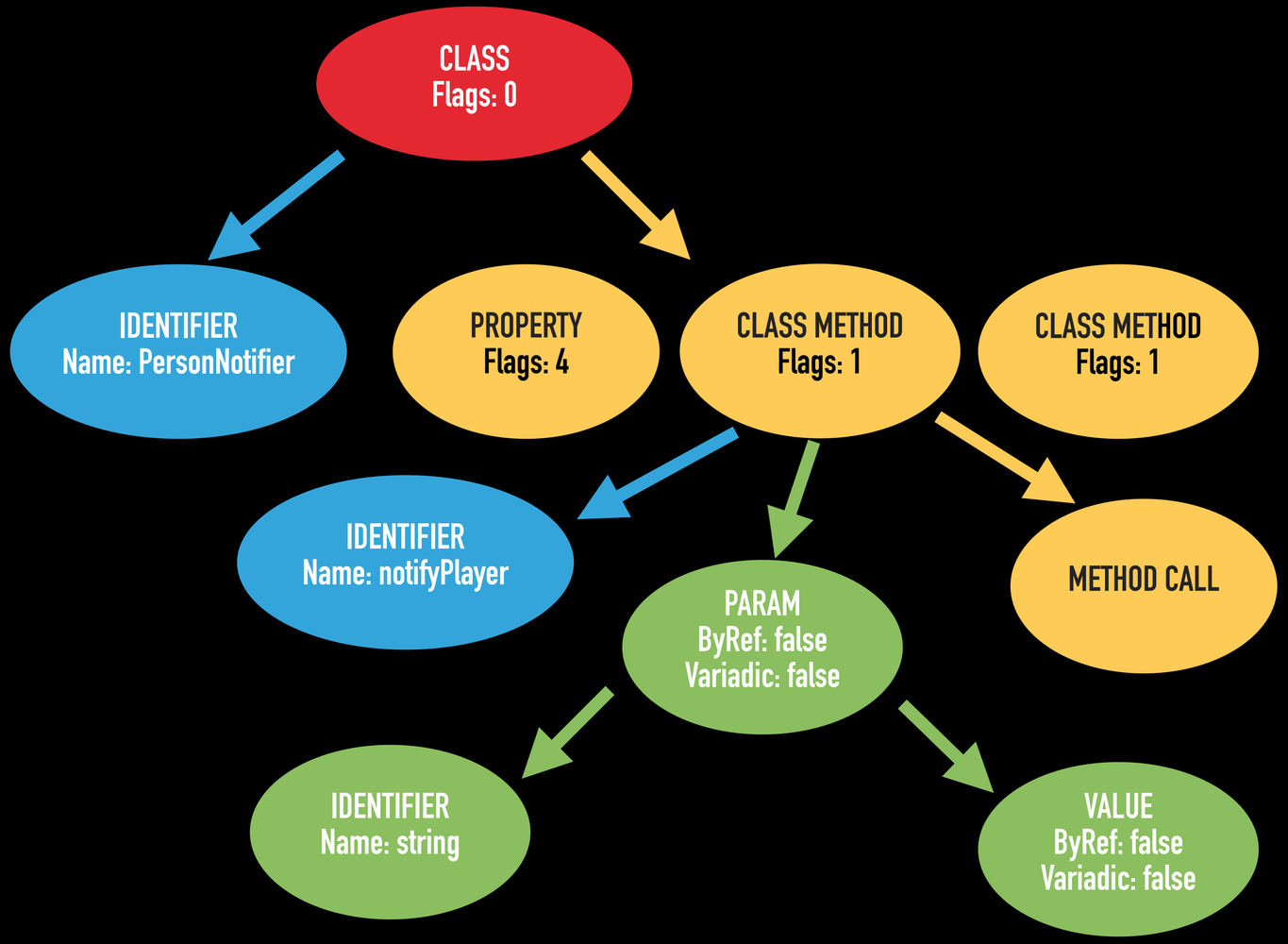
Parse

## 2. Click on any part of the code

```php
<?php

if ($condition === 'demo') {
    return \true;
}
```

## 3. See Abstract Syntax Tree created by php-parser for full file

```
PhpParser\Node\Stmt\If_(
    cond: PhpParser\Node\Expr\BinaryOp\Identical(
        left: PhpParser\Node\Expr\Variable( name: "condition" )
        right: PhpParser\Node\Scalar\String_( value: "demo" )
    )
    stmts: [
        0: PhpParser\Node\Stmt\Return_(
            expr: PhpParser\Node\Expr\ConstFetch(
                name: PhpParser\Node\Name\FullyQualified( parts: ["true"] )
            )
        )
    ]
    elseifs: []
    else: null
)
```


CLASS
Flags: 0
IDENTIFIER
Name: PersonNotifier
PROPERTY
Flags: 4
CLASS METHOD
Flags: 1
CLASS METHOD
Flags: 1
IDENTIFIER
Name: notifyPlayer
PARAM
ByRef: false
Variadic: false
METHOD CALL
IDENTIFIER
Name: string
VALUE
ByRef: false
Variadic: false

```
#[Route('/people', 'GET')]
#[Route(url: '/people', method: 'GET')]
```

```
class Attribute extends \PhpParser\NodeAbstract
{
    public Name $name

    /** @list<Arg> */
    public array $args;

    // Rest of class …
```

---

```
    #[ Route ( '/people', 'GET' )]
```

```
public function getNodes(): array
{
  return [Attribute::class];
}
```

```
public function refactor(Node $node): ?Node
{

}
```

```
#[Route('/people', 'GET')]
```


- **name** is **Route**
- **arg** does not have a name

```
public function refactor(Node $node): ?Node
{
    // Is name Route?

    if (!$this->isName($node->name, Route::class)) {
        return null;
    }
```

```
$updated = false;

 // Does the first argument use a name?

 $arg1 = $node->args[0] ?? null;

if (($arg1 !== null) && ($arg1->name === null)) {
  $arg1->name = new Identifier('url');
  $updated = true;
 }
```

```
$updated = false;

// Does the first argument use a name?
$arg1 = $node->args[0] ?? null;
if (($arg1 !== null) && ($arg1->name === null)) {
  $arg1->name = new Identifier('url');
  $updated = true;
 }
```

```
$updated = false;


// Does the first argument use a name?

$arg1 = $node->args[0] ?? null;

if (($arg1 !== null) && ($arg1->name === null)) {

$arg1->name = new Identifier('url');
  $updated = true;
 }
```

```
class Arg extends NodeAbstract
{
    /**
     * @var Identifier|null Parameter name
     * (for named parameters)
     */
    public ?\PhpParser\Node\Identifier $name;

    /** @var Expr Value to pass */
    public \PhpParser\Node\Expr $value;

    /** @var bool Whether to pass by ref */
    public bool $byRef;

    /** @var bool Whether to unpack the argument */
    public bool $unpack;
```

```
$updated = false;


// Does the first argument use a name?

$arg1 = $node->args[0] ?? null;

if (($arg1 !== null) && ($arg1->name === null)) {

$arg1->name = new Identifier('url');
  $updated = true;
 }
```

```
$updated = false;


// Does the first argument use a name?

$arg1 = $node->args[0] ?? null;

if (($arg1 !== null) && ($arg1->name === null)) {

$arg1->name = new Identifier('url');
  $updated = true;
}
```

```
// Does the second argument use a name?

$arg2 = $node->args[1] ?? null;

if (($arg2 !== null) && ($arg2->name === null)) {
  $arg2->name = new Identifier('method');
  $updated = true;
}
```

```
  // Update if needed

  if (!$updated) {
    return null;
   }

  return $node;
}
```

```php
public function refactor(Node $node): ?Node
{
    // Is name Route?
    if (!$this->isName($node->name, Route::class)) {
        return null;
    }

    $updated = false;

    // Does the first argument use a name?
    $arg1 = $node->args[0] ?? null;
    if (($arg1 !== null) && ($arg1->name === null)) {
        $arg1->name = new Identifier('url');
        $updated = true;
    }

    // Does the second argument use a name?
    $arg2 = $node->args[1] ?? null;
    if (($arg2 !== null) && ($arg2->name === null)) {
        $arg2->name = new Identifier('method');
        $updated = true;
    }

    // Update if needed
    if (!$updated) {
        return null;
    }

    return $node;
}
```

# Improvement 1

```
public function refactor(Node $node): ?Node
{
 // Is name Route?
 if (!$this->isName($node->name, Route::class)) {
    return null;
 }

 // Get information on parameters

 $parameters = new ReflectionClass(Route::class)
                          ->getConstructor()
                          ->getParameters();
```

```
$updated = false;

// Add name to each argument if needed
foreach($node->args as $index => $arg) {

  $parameterName = $parameters[$index]?->name;
  if ($parameterName === null) {
    continue;
  }

  if ($arg->name === null) {
    $arg->name = new Node\Identifier($parameterName);
    $updated = true;
  }
}
```

```
$updated = false;

// Add name to each argument if needed
foreach($node->args as $index => $arg) {
    $parameterName = $parameters[$index]?->name;
    if ($parameterName === null) {
        continue;
    }

    if ($arg->name === null) {
        $arg->name = new Node\Identifier($parameterName);
        $updated = true;
    }
}
```

```php
$updated = false;

// Add name to each argument if needed
foreach($node->args as $index => $arg) {
    $parameterName = $parameters[$index]?->name;
    if ($parameterName === null) {
        continue;
    }
    if ($arg->name === null) {
        $arg->name = new Node\Identifier($parameterName);
        $updated = true;
    }
}
```

```
$updated = false;

// Add name to each argument if needed
foreach($node->args as $index => $arg) {

    $parameterName = $parameters[$index]?->name;
    if ($parameterName === null) {
        continue;
    }

    if ($arg->name === null) {
        $arg->name = new Node\Identifier($parameterName);
        $updated = true;
    }
}
```

```php
public function refactor(Node $node): ?Node
{
    // Is name Route?
    if (!$this->isName($node->name, Route::class)) {
        return null;
    }

    // Get information on parameters
    $parameters = new ReflectionClass(Route::class)
      ->getConstructor()->getParameters();

    $updated = false;

    // Add name to each argument if needed
    foreach($node->args as $index => $arg) {
     $parameterName = $parameters[$index]?->name;
     if ($parameterName === null) {
         continue;
     }

     if ($arg->name === null) {
         $arg->name = new Node\Identifier($parameterName);
         $updated = true;
     }
   }

   // Update if needed
   return $updated ? $node : null;
}
```

# Improvement 2

```
public function refactor(Node $node): ?Node
{
  if (!$this->isName($node->name, $this->className)) {
      return null;
   }

   // Get information on parameters
   $parameters = new ReflectionClass($this->className)
     ->getConstructor()->getParameters();

  … Rest of class as before …
```

# RECTOR

## The Power of Automated Refactoring

2024 Edition
Rector 1.0

Matthias Noback
Tomas Votruba

# All Value Objects must be final and read only

Old

```
#[ValueObject]
class Person
{
}
```

---

New

```
#[ValueObject]
final readonly class Person
{
}
```

```
final class MakeValueObjectsFinalRector
     extends AbstractRector
{

public function getNodeTypes(): array
 {
   return [Class_::class];
 }
```

```
public function __construct(
  private PhpAttributeAnalyzer $attributeAnalyzer,
 ) {}
```

```
public function refactor(Node $node): ?Node
{
    if (!$this->attributeAnalyze->hasPhpAttribute(
         $node, ValueObject::class)) {
        return null;
    }

    if ($node->isFinal() && ($node->isReadonly())) {
        return null;
    }

    $node->flags |= Modifiers::FINAL | Modifiers::READONLY;

    return $node;
}
```

```
public function refactor(Node $node): ?Node
{

if (!$this->attributeAnalyze->hasPhpAttribute(
     $node, ValueObject::class)) {
    return null;
 }

 if ($node->isFinal() && ($node->isReadonly())) {
   return null;
 }

 $node->flags |= Modifiers::FINAL | Modifiers::READONLY;

 return $node;
}
```

```
public function refactor(Node $node): ?Node
{

if (!$this->attributeAnalyze->hasPhpAttribute(
     $node, ValueObject::class)) {
    return null;
 }

 if ($node->isFinal() && ($node->isReadonly())) {
   return null;
 }

 $node->flags |= Modifiers::FINAL | Modifiers::READONLY;

 return $node;
}
```

# Previously working code breaks

```
#[ValueObject]
class Person {…}

class SomeClass extends Person
{…}
```

Before

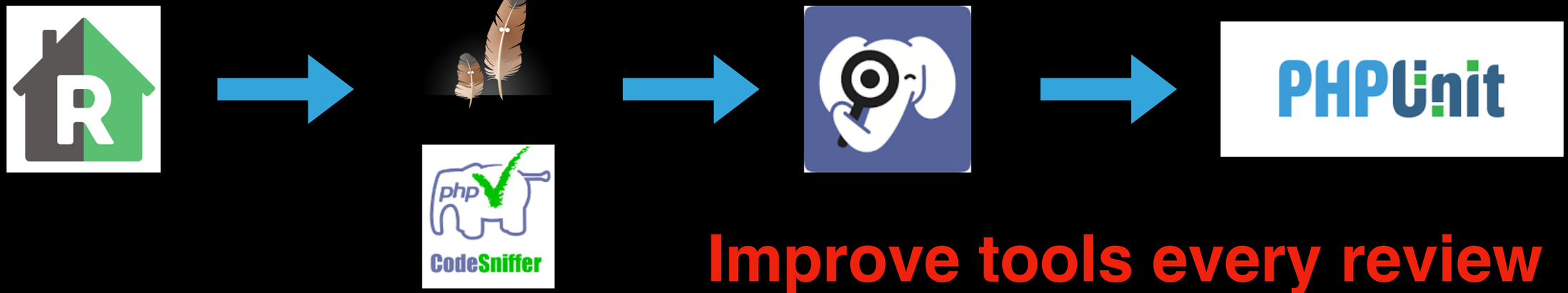
---

```
#[ValueObject]
final readonly class Person {…}

class SomeClass extends Person
{…}
```

After

# Coding standards

- PSR 12
- Use Early Return pattern
- Use named arguments for #[Route]
- Value objects must be final and readonly




**Improve tools every review**

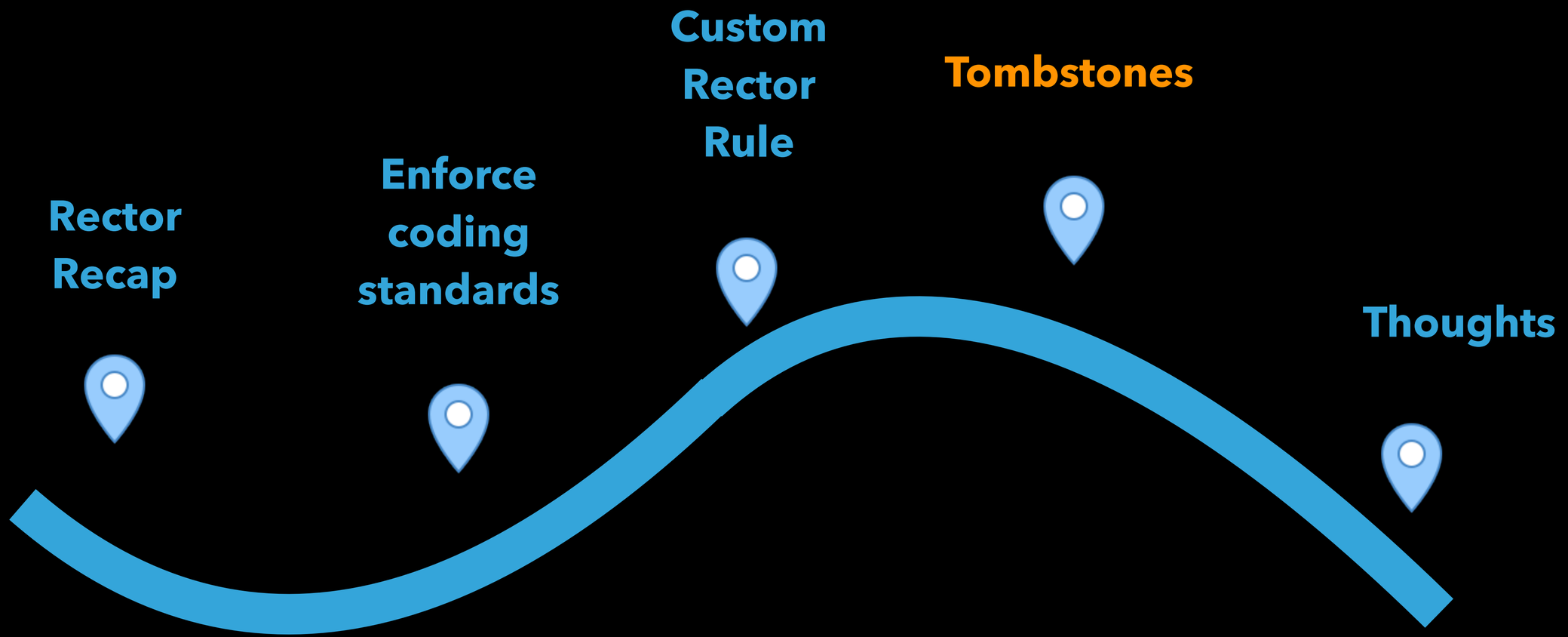

Rector Recap
Enforce coding standards
Custom Rector Rule
Tombstones
Thoughts


Photo by Antoine GIRET on Unsplash

# Add tombstones

```
final class PriceCalculator
{
 public function calculate(): void
 {

   TombstoneReporter::trigger(
     'PriceCalculator', 'calculate');

  // Rest of method's code
 }
}
```

# TombstoneReporter::trigger

- Gets list of all triggered tombstones from database (once per request)
- If triggered tombstone not on list, create DB entry

# TombstoneReporter::dump

triggered_tombstones.json

```
[
  'PriceCalculator::calculate',
  'PriceController::index',
  'PriceRepository::getItems,
  ...
];
```

# Remove triggered tombstones

```
final class PriceCalculator
{
 public function calculate(): void
 {

   TombstoneReporter::trigger(
     'PriceCalculator', 'calculate');

  // Rest of method's code
 }
}
```

# Delete code

“REMOVE: Product listing page”

“REMOVE: 2 for 1 discount code”

Photo by Claudio Schwarz on Unsplash

# How do we add tombstones?

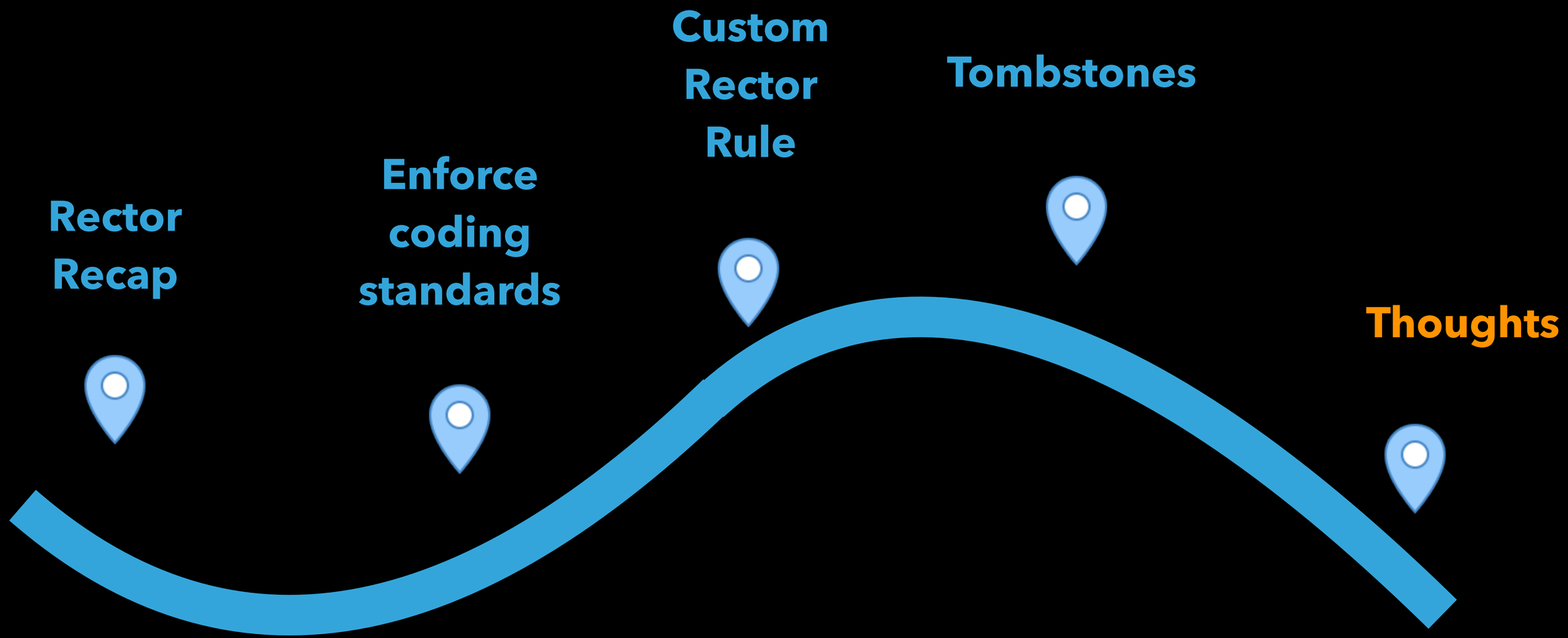
# How do we add tombstones?

```
final class AddTombstoneRector extends AbstractRector
{


    public function getNodeTypes(): array
    {
        return [ClassMethod::class];
    }
```

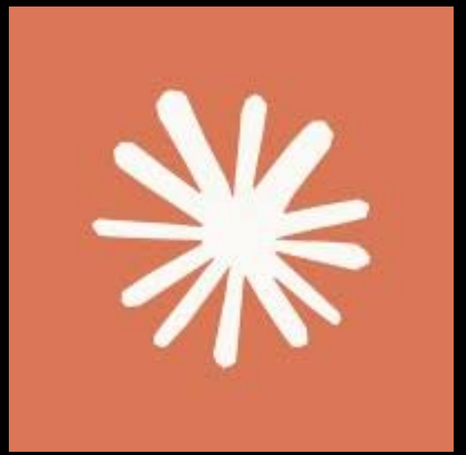
```
public function refactor(Node $node): ?Node
{
    $scope = ScopeFetcher::fetch($node);
    $className = $scope->getClassReflection()?->getName() ?? null;
    if ($className === null) {
      return null;
    }

    $staticCall = $this->nodeFactory->createStaticCall(
      Tombstone::class,
      'trigger',
      [
        $this->nodeFactory->createArg($className),
        $this->nodeFactory->createArg($node->name->name),
      ]
    );
    $staticCallStatement = new Node\Stmt\Expression($staticCall);

    $statements = $node->stmts ?? [];
    array_unshift($statements, $staticCallStatement);
    $node->stmts = $statements;
    return $node;
}
```

```php
public function refactor(Node $node): ?Node
{

  $scope = ScopeFetcher::fetch($node);
  $className = $scope->getClassReflection()?->getName() ?? null;
  if ($className === null) {
    return null;
  }

  $staticCall = $this->nodeFactory->createStaticCall(
    Tombstone::class,
    'trigger',
    [
      $this->nodeFactory->createArg($className),
      $this->nodeFactory->createArg($node->name->name),
    ]
  );
  $staticCallStatement = new Node\Stmt\Expression($staticCall);

  $statements = $node->stmts ?? [];
  array_unshift($statements, $staticCallStatement);
  $node->stmts = $statements;
  return $node;
}
```

```
public function refactor(Node $node): ?Node
{

  $scope = ScopeFetcher::fetch($node);
  $className = $scope->getClassReflection()?->getName() ?? null;
  if ($className === null) {
    return null;
  }

  $staticCall = $this->nodeFactory->createStaticCall(
    Tombstone::class,
    'trigger',
    [
      $this->nodeFactory->createArg($className),
      $this->nodeFactory->createArg($node->name->name),
    ]
  );
  $staticCallStatement = new Node\Stmt\Expression($staticCall);

  $statements = $node->stmts ?? [];
  array_unshift($statements, $staticCallStatement);
  $node->stmts = $statements;
  return $node;
}
```


Rector Recap
Enforce coding standards
Custom Rector Rule
Tombstones
Thoughts

# Type coverage is good

# 0% type coverage

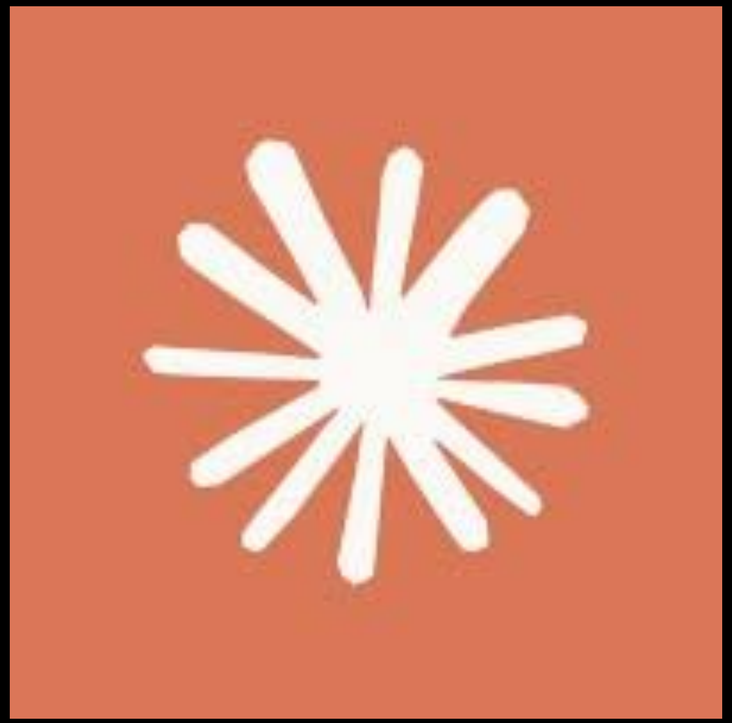
```
function getNames($users)
{

  $names = [];
  foreach($users as $user) {
    $names[] = $user->getUser();
  }

  return join(‘,’, $names);
}
```

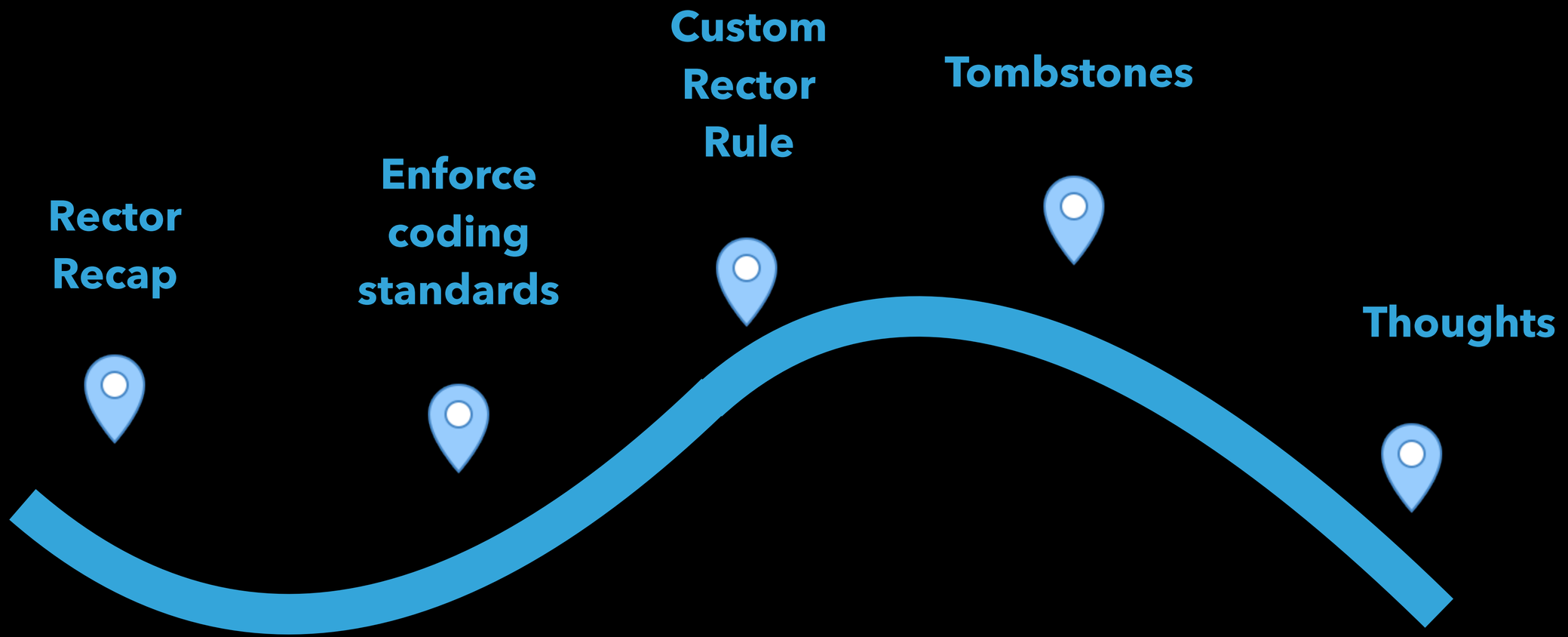
# 100% type coverage

```
/** @param array<int,User> $users */
function getNames(array $users): string
{

  $names = [];
  foreach($users as $user) {
    $names[] = $user->getUser();
  }
  return join(‘,’, $names);

}
```

# Could we track type information?

```
class Person
{
  public function process($name, $age)
  {
    TypeRecorder::record(
        method: “Person::process”
        argument: 1,
        value: $name);

    TypeRecorder::record(
        method: “Person::process”
        argument: 2,
        value: $age);

    … rest of method …
  }
```




Rector Recap
Enforce coding standards
Custom Rector Rule
Tombstones
Thoughts

- Upgrades
- One off Refactors
- Downgrades
- Automate code review
- Investigation work

# Dave Liddament

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


Me when I'm not coding!