



*php***USERgroup**
Frankfurt am Main

Extending the PHP language with static analysis

Dave Liddament

Lamp Bristol

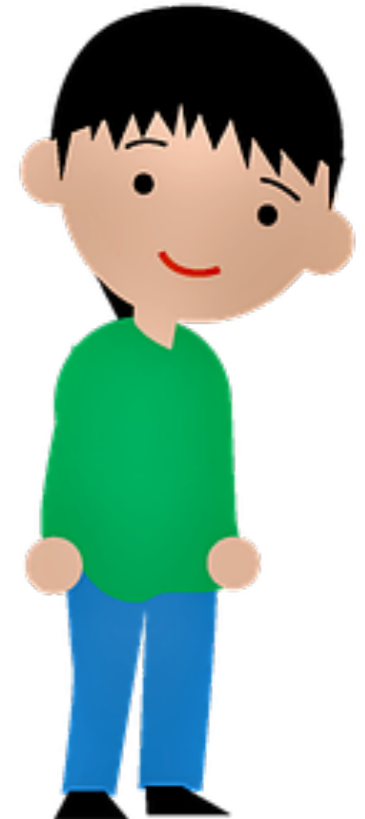
@DaveLiddament

@DaveLiddament@phpc.social



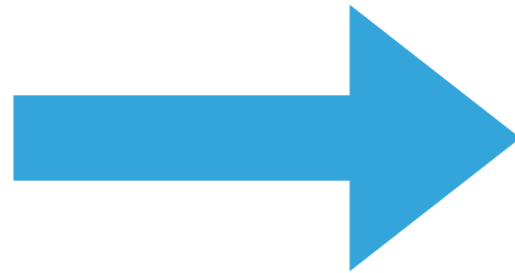
IF ONLY PHP HAD FEATURE ...
SIMILAR TO LANGUAGE ...

WOULDN'T IT BE GREAT IF PHP COULD
DO ...



HOW I BUILT NEW LANGUAGE FEATURES

Very
specific
constraint

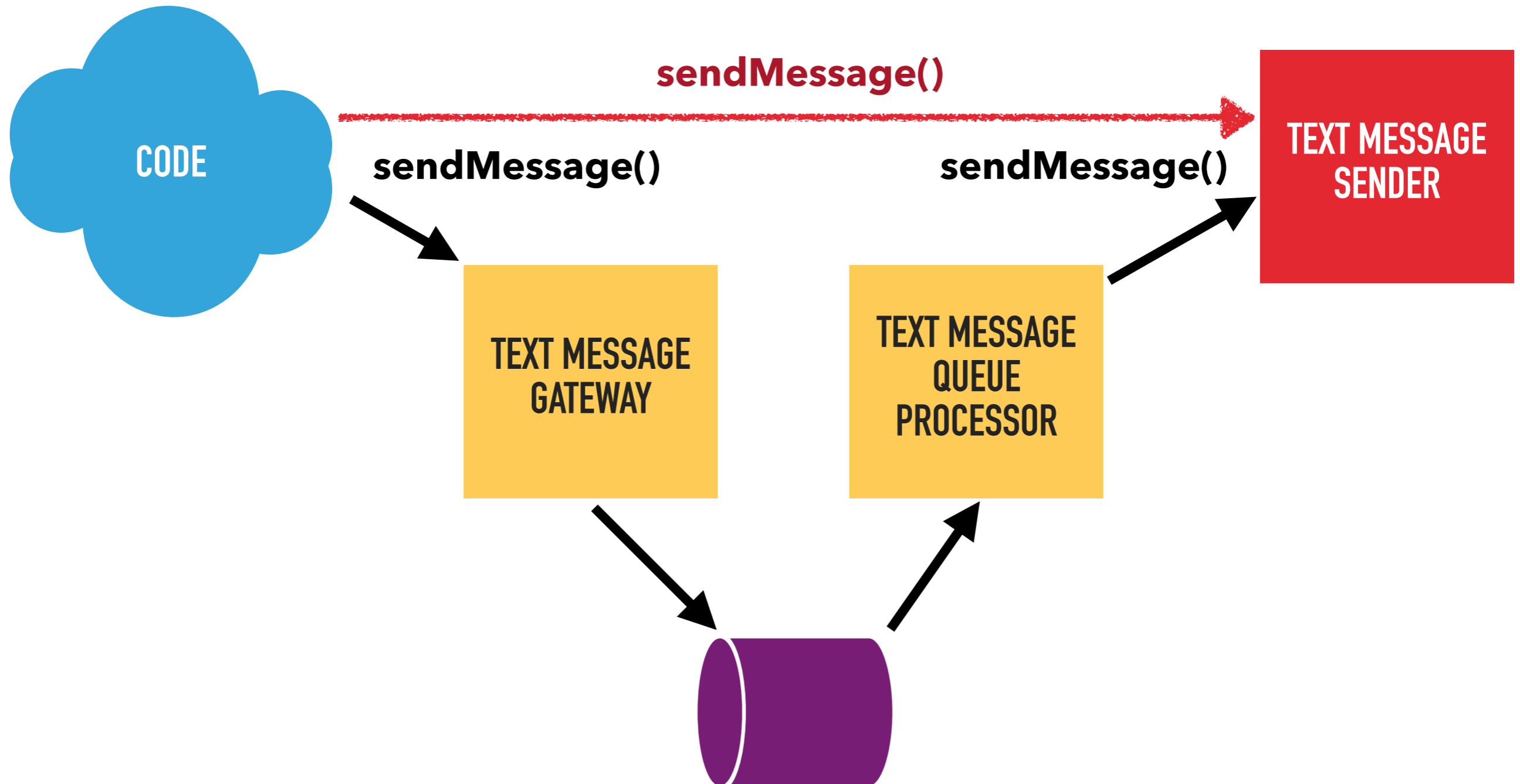


Generalised
version that
could be a
useful on your
project

Preconditions

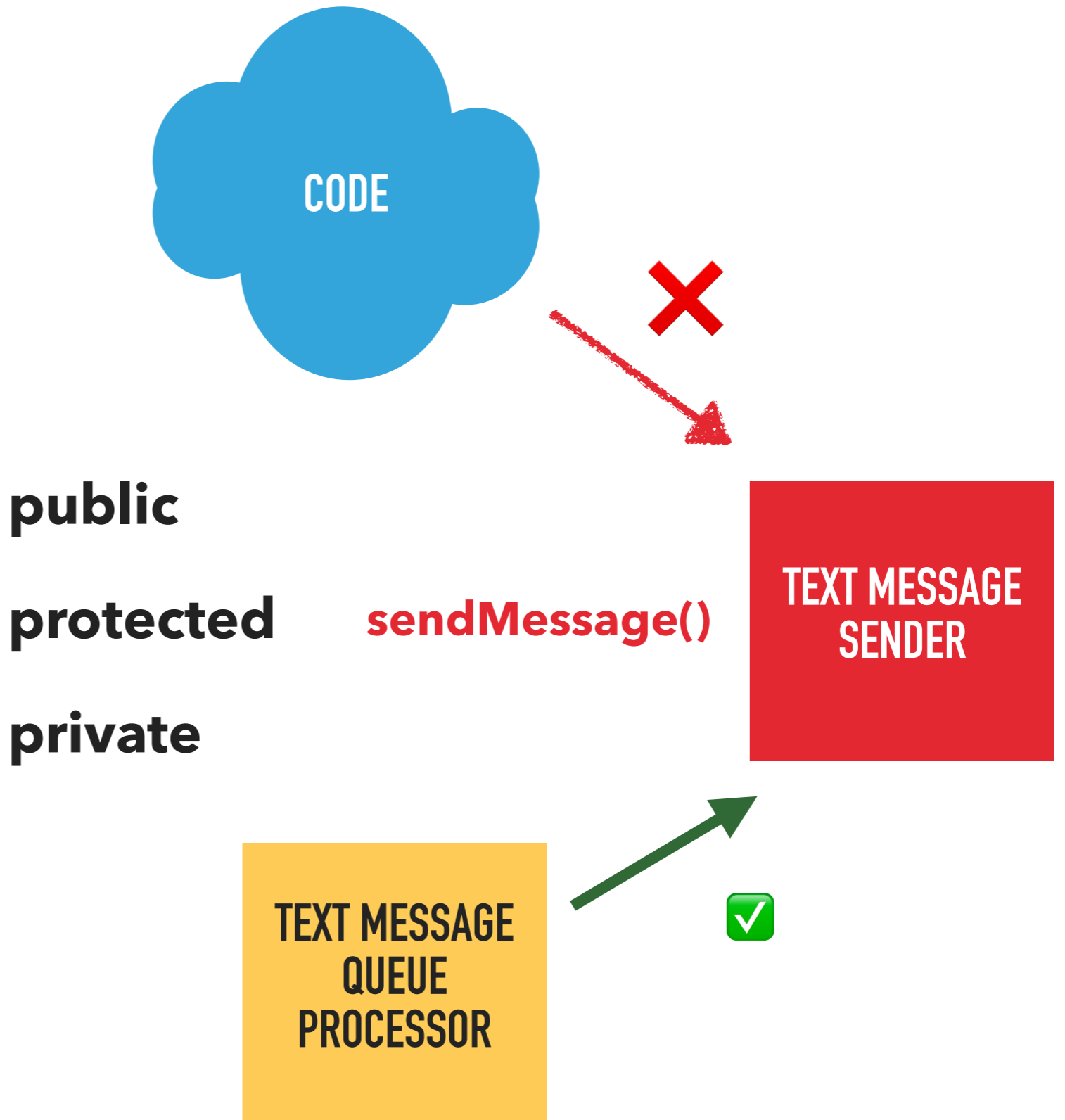


One of many examples...

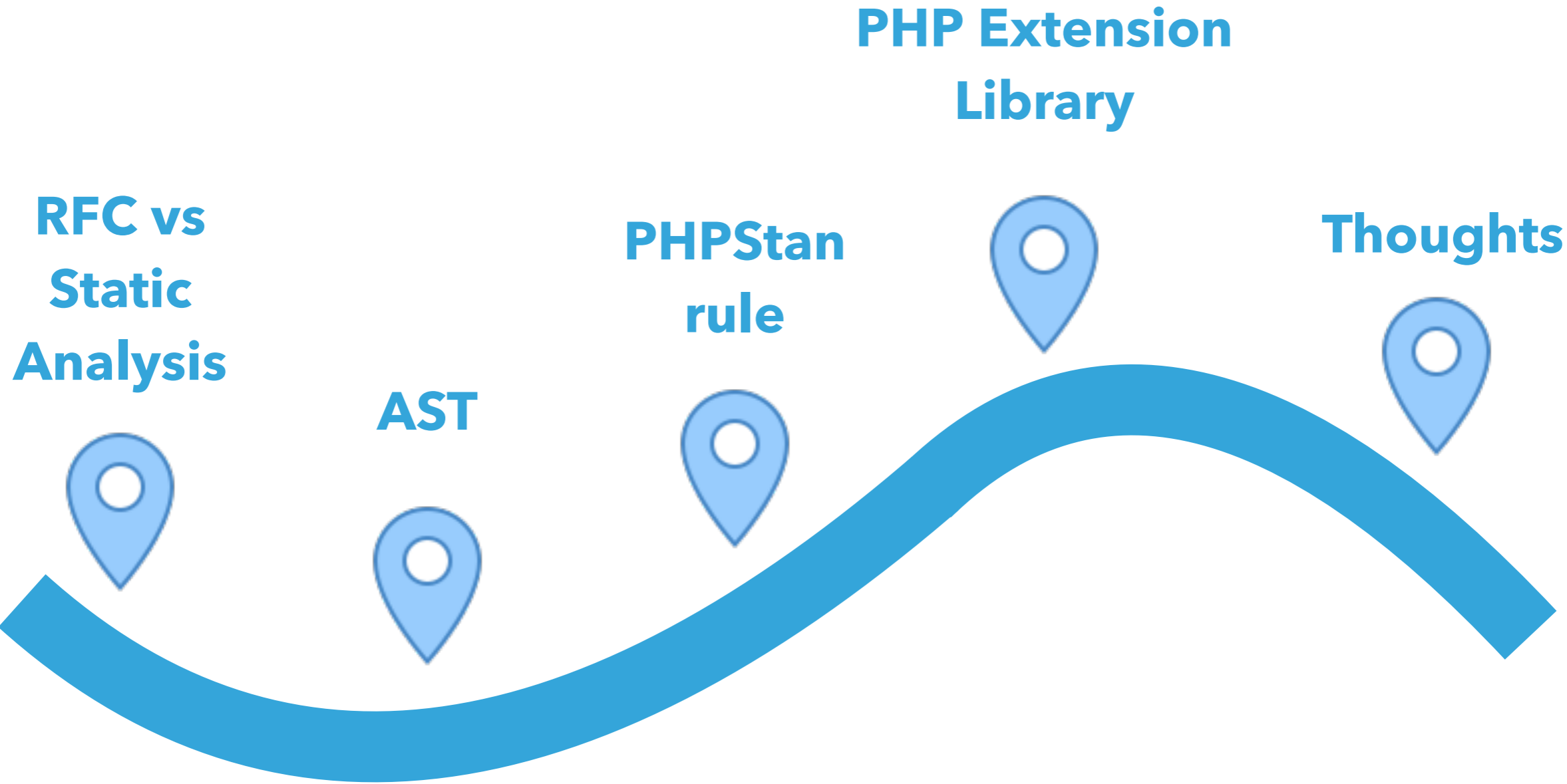


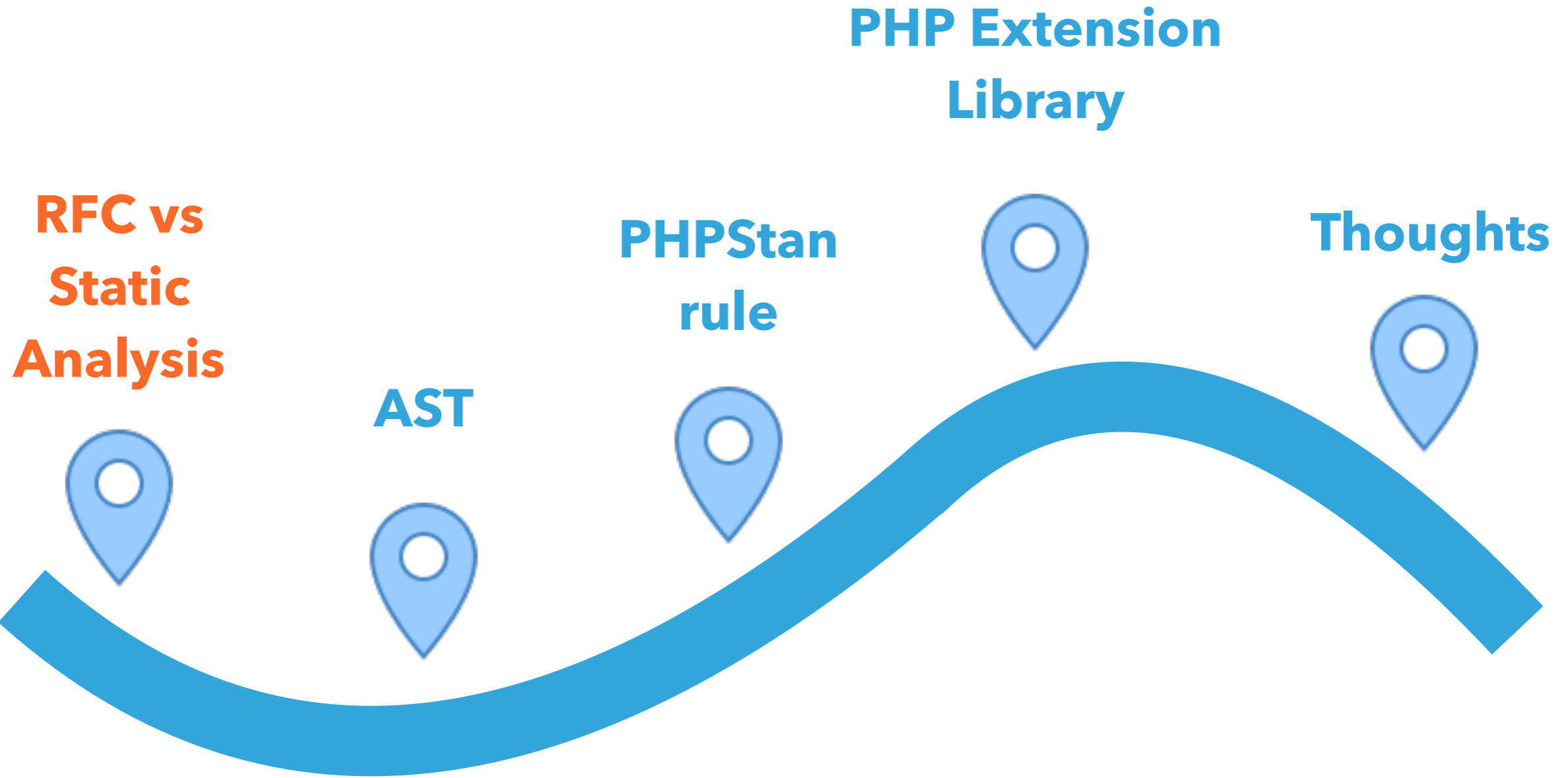
Existing visibility modifiers are not fine grained enough.

We need more control.



Automate checks to stop me, or other developers, breaking this constraint







TELL ME ABOUT PHP'S RFC PROCESS

FIRST YOU WRITE AN RFC

TALK

VOTE

IMPLEMENT

A LONG AND DIFFICULT PROCESS!

THERE IS ANOTHER WAY...

... FOR SOME FUNCTIONALITY

Run time

```
class Person
{
    private function update()
    {
        // Some code
    }
}
```

```
$person = new Person();  
$person->update();
```

Uncaught Error: Call to private method Person::update()

Static analysis

```
class Person
{
    private function update()
    {
        // Some code
    }
}
```

```
$person = new Person();  
$person->update(); ❌
```




Static analysis gives us generics now

```
/** @return Person[] */  
function getPeople():array {...}
```

```
function process(Car $car) {...}
```

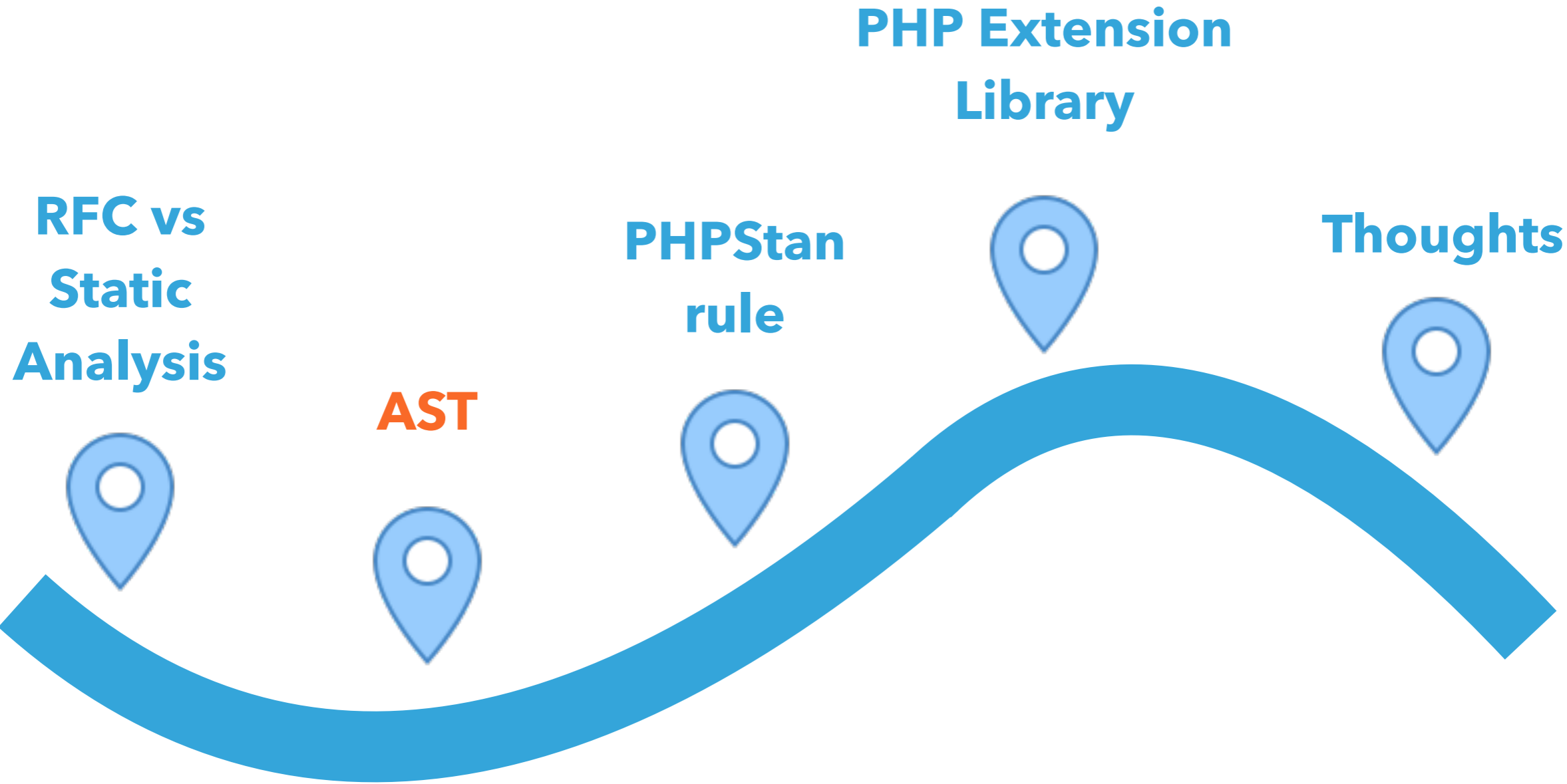
```
for (getPeople() as $person) {  
    process($person);  
}
```



Add static analysis to dev process



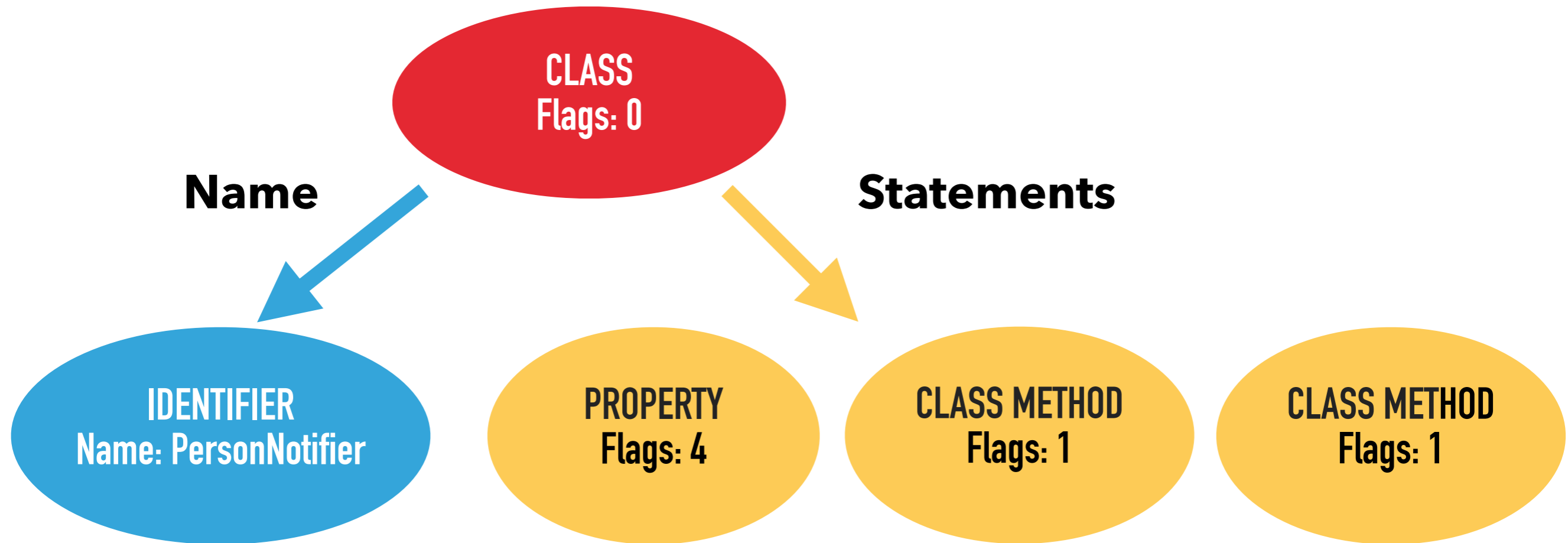
Create custom rules to emulate new language features



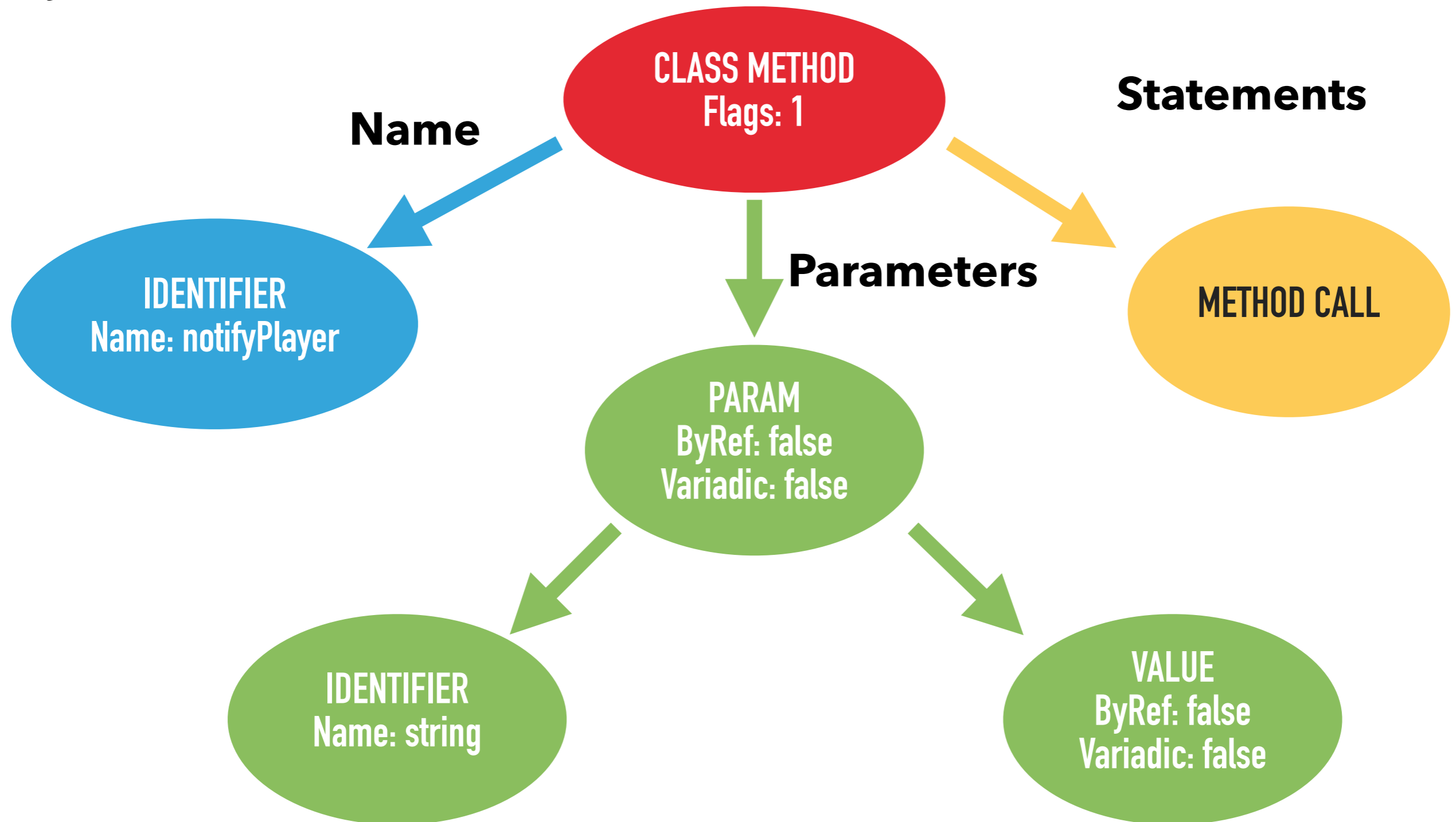



```
class PersonNotifier
```

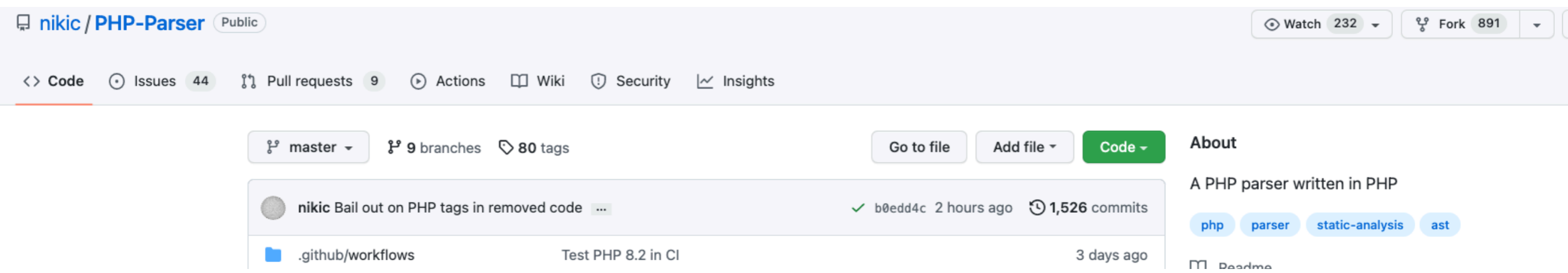
```
{  
  private TextMessageSender $sender;  
  public function __construct() {...}  
  public function notifyPlayer() {...}  
}
```



```
public function notifyPlayer (string $msg)
{
    $this->sender->sendMessage ($msg) ;
}
```

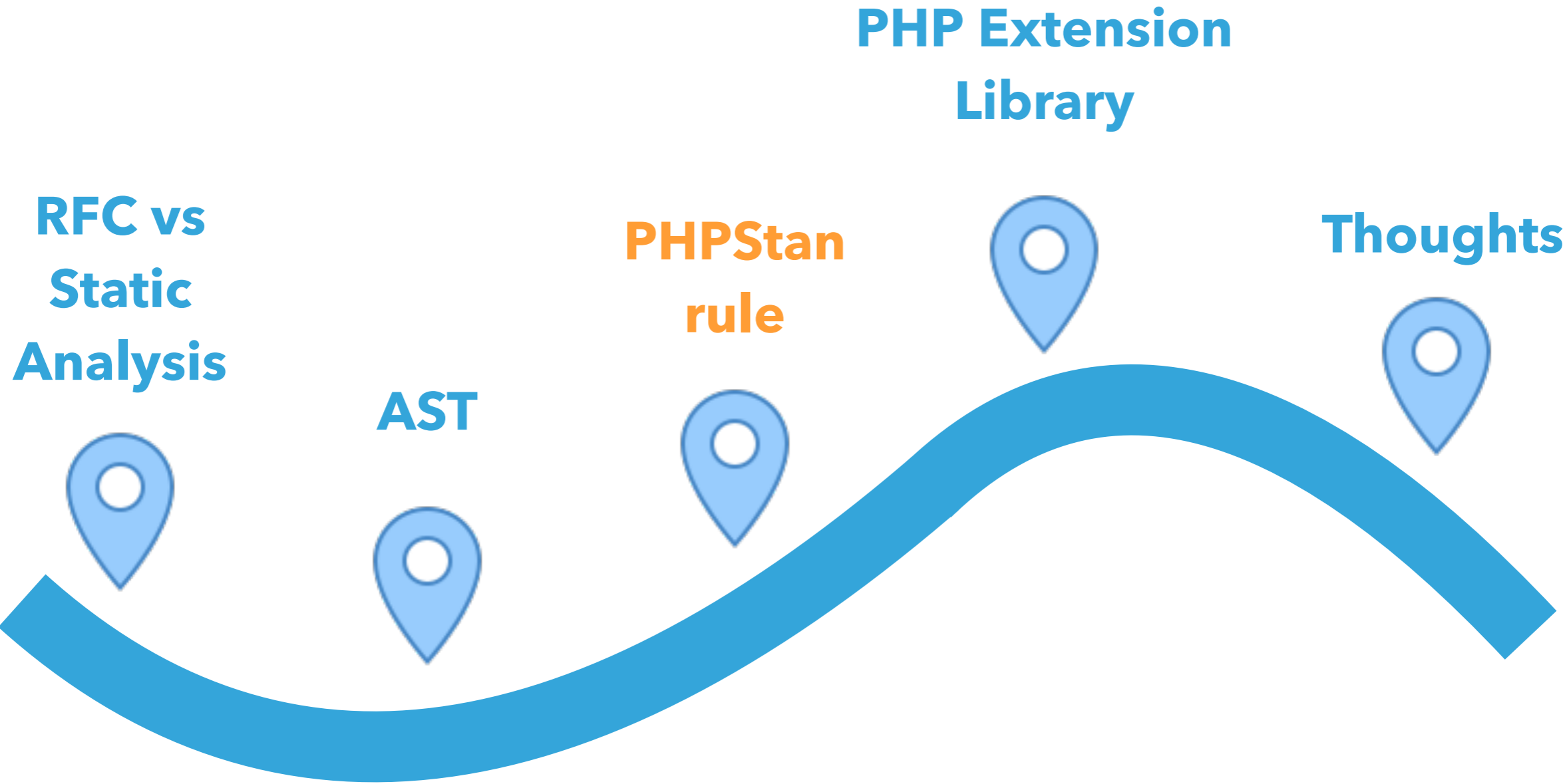


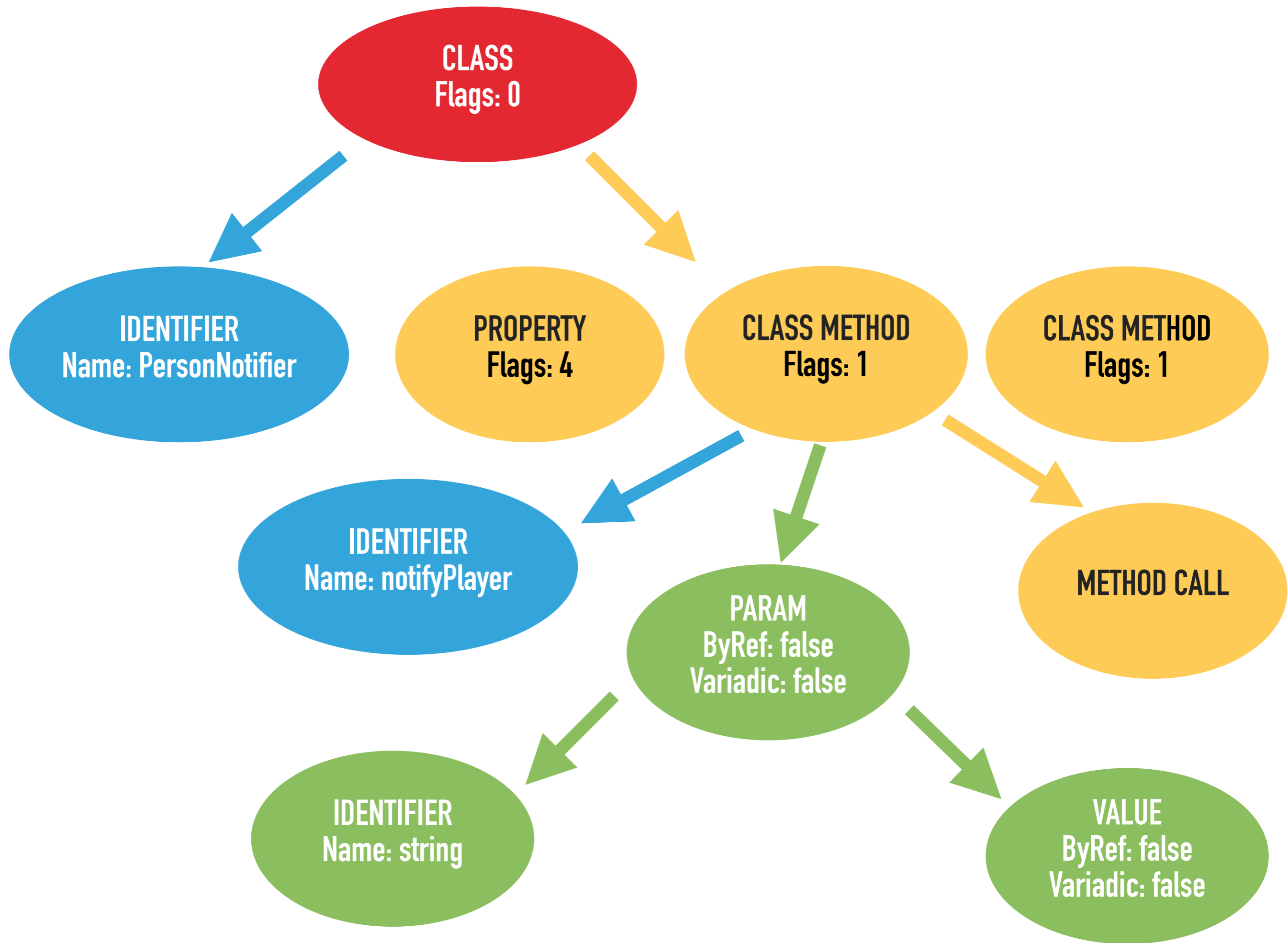
https://github.com/nikic/PHP-Parser



The screenshot shows the GitHub repository page for 'nikic/PHP-Parser'. The repository is public and has 232 watchers and 891 forks. The main navigation bar includes links for Code, Issues (44), Pull requests (9), Actions, Wiki, Security, and Insights. The repository is currently on the 'master' branch, with 9 branches and 80 tags. A recent commit by 'nikic' is shown with the message 'Bail out on PHP tags in removed code ...' and commit hash 'b0edd4c', made 2 hours ago. There are 1,526 commits in total. A workflow file '.github/workflows/Test PHP 8.2 in CI' is listed, updated 3 days ago. The 'About' section describes it as 'A PHP parser written in PHP' and lists tags: php, parser, static-analysis, and ast. A 'Readme' link is also visible.

- ▶ PHP code can be represented by an AST
- ▶ Different types of Node
- ▶ Nodes contain information
- ▶ Each type of node has different information



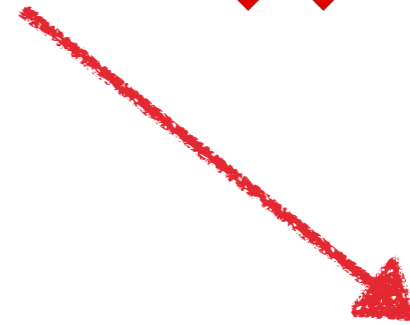
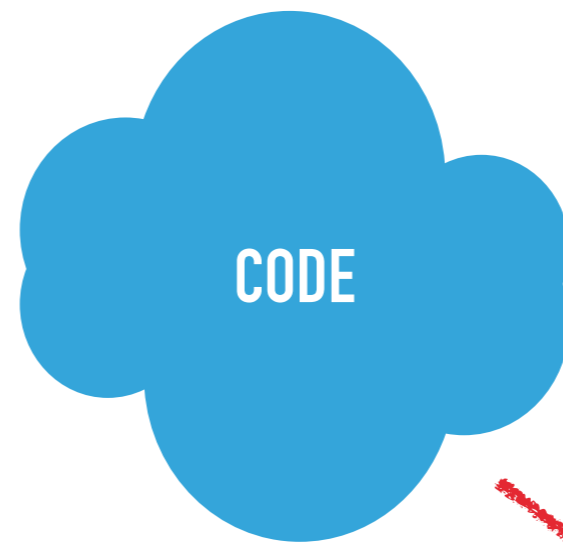


```
interface Rule
{

    public function getNodeTypes() : string;

    /**
     * @return (string|RuleError)[] errors
     */
    public function processNode(
        \PhpParser\Node $node,
        \PHPStan\Analyser\Scope $scope
) : array;

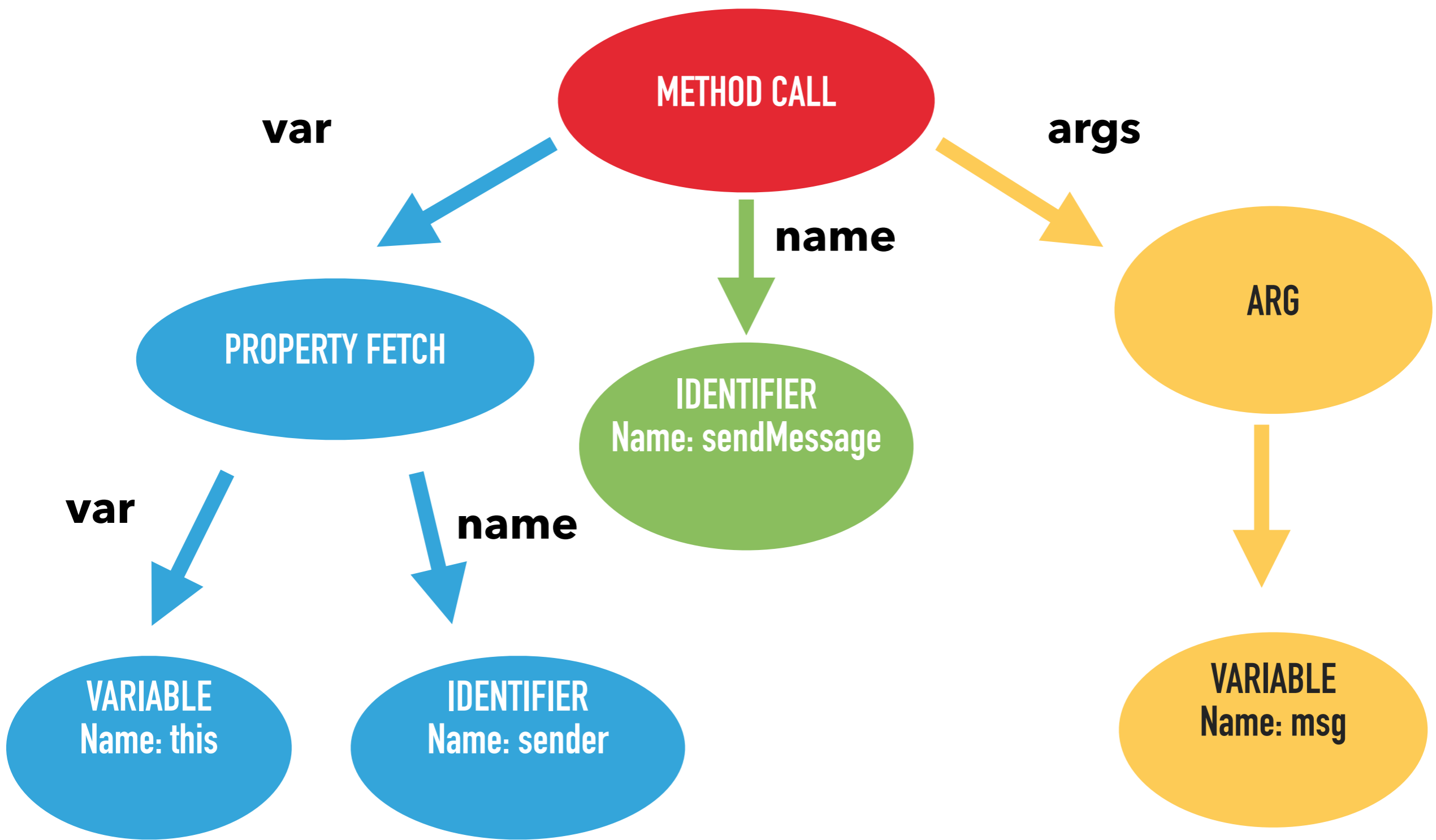
}
```



We can only call methods
in **TextMessageSender** from
TextMessageQueueProcessor



```
$this->sender -> sendMessage ($msg) ;
```




```
class MethodCall extends \PhpParser\Node\Expr\CallLike
{
```

```
/** @var Expr Variable holding object */
public $var;
```

```
/** @var Identifier|Expr Method name */
public $name;
```

```
/** @var array<Arg|VariadicPlaceholder> Arguments */
public $args;
```

```
// Rest of class ...
```

```
$this->sender -> sendMessage ($msg) ;
```

```
class TextMessageSenderCallerRule  
  implements Rule
```

```
{
```

```
  public function getNodeTypes() : string
```

```
{
```

```
    return MethodCall::class;
```

```
}
```

```
public function processNode(Node $node, Scope $scope): array
{
    $callingClass = $scope->getClassReflection()->getName();

    if ($callingClass === TextMessageQueueProcessor::class) {
        return [];
    }

    $type = $scope->getType($node->var);

    foreach ($type->getReferencedClasses() as $targetClass) {
        if ($targetClass === TextMessageSender::class) {
            $msg = "Cant call TextMessageSender from here";
            return [RuleErrorBuilder::message($message)->build()];
        }
    }

    return [];
}
```

```
public function processNode(Node $node, Scope $scope): array
{
    $callingClass = $scope->getClassReflection()->getName();

    if ($callingClass === TextMessageQueueProcessor::class) {
        return [];
    }

    $type = $scope->getType($node->var);

    foreach ($type->getReferencedClasses() as $targetClass) {
        if ($targetClass === TextMessageSender::class) {
            $msg = "Cant call TextMessageSender from here";
            return [RuleErrorBuilder::message($message)->build()];
        }
    }

    return [];
}
```

```
class PersonNotifier
```

```
{
```

```
    private TextMessageSender $sender;
```

```
    public function notifyPlayer(string $msg): void
```

```
{
```

```
    $this->sender->sendMessage($msg);
```

```
}
```

```
}
```

`$this->sender` is called from `PersonNotifier`

```
public function processNode(Node $node, Scope $scope): array
{
    $callingClass = $scope->getClassReflection()->getName();

    if ($callingClass === TextMessageQueueProcessor::class) {
        return [];
    }

    $type = $scope->getType($node->var);

    foreach ($type->getReferencedClasses() as $targetClass) {
        if ($targetClass === TextMessageSender::class) {
            $msg = "Cant call TextMessageSender from here";
            return [RuleErrorBuilder::message($message)->build()];
        }
    }

    return [];
}
```

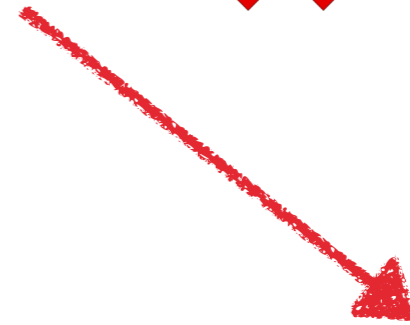
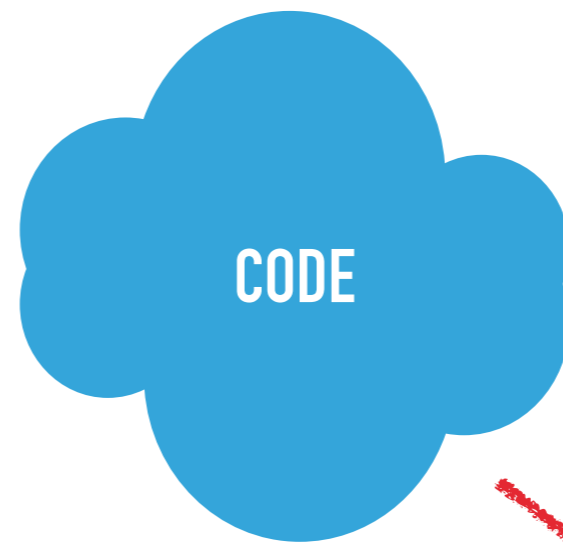
```
public function processNode(Node $node, Scope $scope): array
{
    $callingClass = $scope->getClassReflection()->getName();

    if ($callingClass === TextMessageQueueProcessor::class) {
        return [];
    }

    $type = $scope->getType($node->var);

    foreach ($type->getReferencedClasses() as $targetClass) {
        if ($targetClass === TextMessageSender::class) {
            $msg = "Cant call TextMessageSender from here";
            return [RuleErrorBuilder::message($message)->build()];
        }
    }

    return [];
}
```



We can only call methods
in **TextMessageSender** from
TextMessageQueueProcessor




```
public function processNode(Node $node, Scope $scope): array
{
    $callingClass = $scope->getClassReflection()->getName();

    if ($callingClass === TextMessageQueueProcessor::class) {
        return [];
    }

    $type = $scope->getType($node->var);

    foreach ($type->getReferencedClasses() as $targetClass) {
        if ($targetClass === TextMessageSender::class) {
            $msg = "Cant call TextMessageSender from here";
            return [RuleErrorBuilder::message($message)->build()];
        }
    }

    return [];
}
```

```
public function processNode(Node $node, Scope $scope): array
{
    $callingClass = $scope->getClassReflection()->getName();

    if ($callingClass === TextMessageQueueProcessor::class) {
        return [];
    }

    $type = $scope->getType($node->var);

    foreach ($type->getReferencedClasses() as $targetClass) {
        if ($targetClass === TextMessageSender::class) {
            $msg = "Cant call TextMessageSender from here";
            return [RuleErrorBuilder::message($message)->build()];
        }
    }

    return [];
}
```

```
$this->sender->notifyPlayer($msg);
```



var



name



args

```
class PersonNotifier
```

```
{
```

```
private TextMessageSender $sender;
```

```
public function notifyPlayer(string $msg): void
```

```
{
```

```
$this->sender->sendMessage($msg);
```

```
}
```

```
}
```

`$this->sender` is of type `TextMessageSender`

```
public function processNode(Node $node, Scope $scope): array
{
    $callingClass = $scope->getClassReflection()->getName();

    if ($callingClass === TextMessageQueueProcessor::class) {
        return [];
    }

    $type = $scope->getType($node->var);

    foreach ($type->getReferencedClasses() as $targetClass) {
        if ($targetClass === TextMessageSender::class) {
            $msg = "Cant call TextMessageSender from here";
            return [RuleErrorBuilder::message($message)->build()];
        }
    }

    return [];
}
```

```
public function processNode(Node $node, Scope $scope): array
{
    $callingClass = $scope->getClassReflection()->getName();

    if ($callingClass === TextMessageQueueProcessor::class) {
        return [];
    }

    $type = $scope->getType($node->var);

    foreach ($type->getReferencedClasses() as $targetClass) {

        if ($targetClass === TextMessageSender::class) {
            $msg = "Cant call TextMessageSender from here";
            return [RuleErrorBuilder::message($message)->build()];
        }
    }

    return [];
}
```

```
class PersonNotifier
```

```
{
```

```
private TextMessageSender | WhatsappSender $sender;
```

```
public function notifyPlayer(string $msg): void
```

```
{
```

```
$this->sender->sendMessage($msg);
```

```
}
```

```
}
```

`$this->sender` is of type `TextMessageSender`

or `WhatsappSender`

```
public function processNode(Node $node, Scope $scope): array
{
    $callingClass = $scope->getClassReflection()->getName();

    if ($callingClass === TextMessageQueueProcessor::class) {
        return [];
    }

    $type = $scope->getType($node->var);

    foreach ($type->getReferencedClasses() as $targetClass) {

        if ($targetClass === TextMessageSender::class) {
            $msg = "Cant call TextMessageSender from here";
            return [RuleErrorBuilder::message($message)->build()];
        }
    }

    return [];
}
```



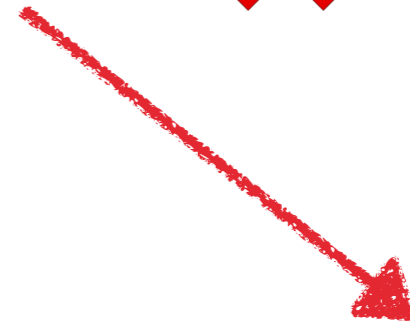
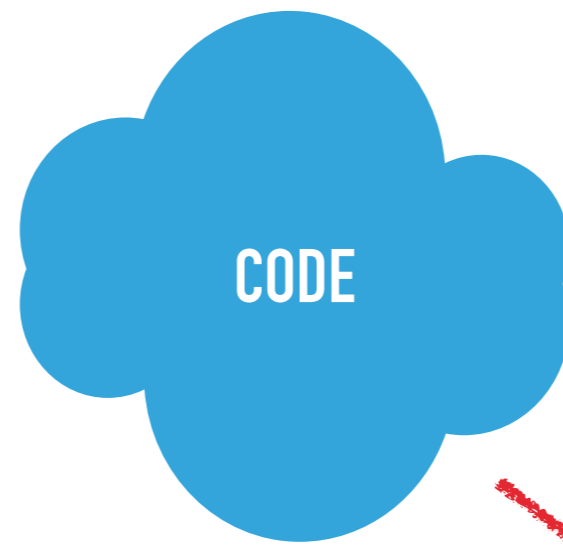
```
public function processNode(Node $node, Scope $scope): array
{
    $callingClass = $scope->getClassReflection()->getName();

    if ($callingClass === TextMessageQueueProcessor::class) {
        return [];
    }

    $type = $scope->getType($node->var);

    foreach ($type->getReferencedClasses() as $targetClass) {
        if ($targetClass === TextMessageSender::class) {
            $msg = "Cant call TextMessageSender from here";
            return [RuleErrorBuilder::message($message)->build()];
        }
    }

    return [];
}
```



We can only call methods
in **TextMessageSender** from
TextMessageQueueProcessor



```
public function processNode(Node $node, Scope $scope): array
{
    $callingClass = $scope->getClassReflection()->getName();

    if ($callingClass === TextMessageQueueProcessor::class) {
        return [];
    }

    $type = $scope->getType($node->var);

    foreach ($type->getReferencedClasses() as $targetClass) {

        if ($targetClass === TextMessageSender::class) {
            $msg = "Cant call TextMessageSender from here";
            return [RuleErrorBuilder::message($message)->build()];
        }
    }

    return [];
}
```

```
public function processNode(Node $node, Scope $scope): array
{
    $callingClass = $scope->getClassReflection()->getName();

    if ($callingClass === TextMessageQueueProcessor::class) {
        return [];
    }

    $type = $scope->getType($node->var);

    foreach ($type->getReferencedClasses() as $targetClass) {

        if ($targetClass === TextMessageSender::class) {
            $msg = "Cant call TextMessageSender from here";
            return [RuleErrorBuilder::message($message)->build()];
        }
    }

    return [];
}
```

```
public function processNode(Node $node, Scope $scope): array
{
    // Find the class that method call is in
    $callingClass = $scope->getClassReflection()->getName();

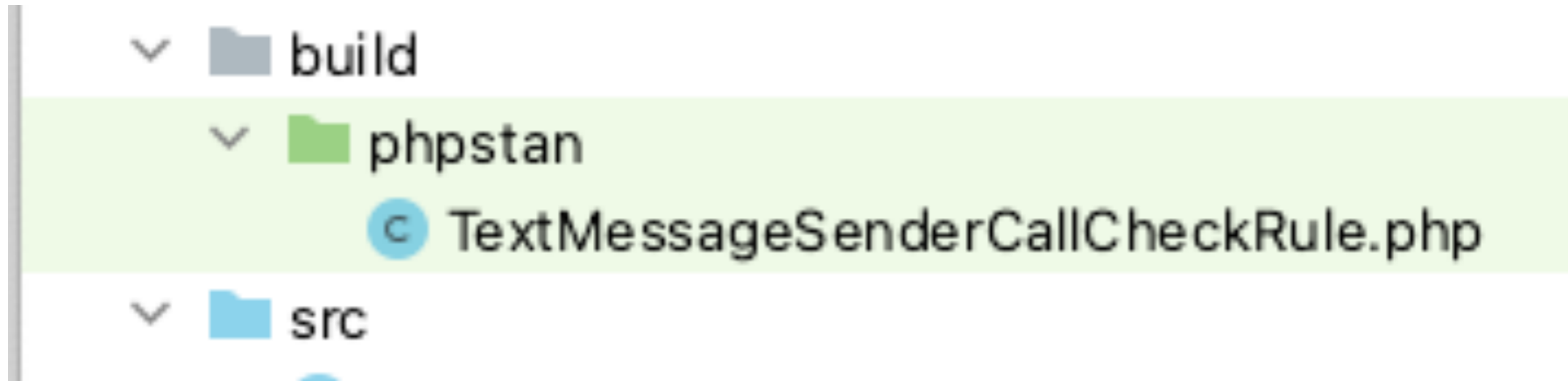
    // If in TextMessageQueueProcessor everything is OK
    if ($callingClass === TextMessageQueueProcessor::class) {
        return [];
    }

    // Get type of the class of the method call
    $type = $scope->getType($node->var);

    // Iterate through all the possible classes
    foreach ($type->getReferencedClasses() as $targetClass) {

        // Trying to call a method in TextMessageSender? Report error
        if ($targetClass === TextMessageSender::class) {
            $msg = "Cant call TextMessageSender from here";
            return [RuleErrorBuilder::message($message)->build()];
        }
    }

    // If we've got this far then there are no errors
    return [];
}
```



```
"autoload-dev": {  
    "psr-4": {  
        "DaveLiddament\\PhpstanRules\\": "build/phpstan/"  
    }  
},
```

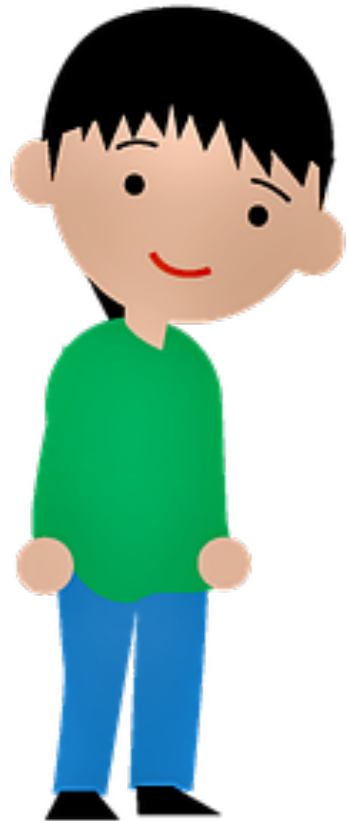
services:

-

```
class: DaveLiddament\\PhpstanRules\\TextMessageSenderCallCheckRule  
tags:  
- phpstan.rules.rule
```

AMAZING!

WE CAN DO BETTER...



**We can only call
methods in target class
from a specified
allowed calling class**


```
class TextMessageSenderCallerRule implements Rule  
{
```

```
    public function __construct(  
        private string $allowedCallingClass,  
        private string $targetClass,  
) {}
```

```
public function processNode(Node $node, Scope $scope): array
{
    $callingClass = $scope->getClassReflection()->getName();

    if ($callingClass === $this->allowedCallingClass) {
        return [];
    }

    $type = $scope->getType($node->var);

    foreach ($type->getReferencedClasses() as $targetClass) {

        if ($targetClass === $this->targetClass) {
            $msg = "Can not call {$this->targetClass} from here";
            return [RuleErrorBuilder::message($message)->build()];
        }
    }

    return [];
}
```

services:

-

class: PhpstanRules\TextMessageSenderCallCheckRule

tags:

- **phpstan.rules.rule**

arguments:

allowedCallingClass: RuleDemo\TextMessageQueueProcessor

targetClass: RuleDemo\TextMessageSender

class: PhpstanRules\TextMessageSenderCallCheckRule

tags:

- **phpstan.rules.rule**

arguments:

allowedCallingClass: Foo

targetClass: Bar

**I WAS AMAZED BEFORE. NOW I'M
EVEN MORE AMAZED!**



WE CAN DO BETTER...



```
/**
 * Can only be called from TextMessageQueueProcessor
 */
class TextMessageSender
{
}
}
```

- ▶ Remember to document
- ▶ Remember to setup some config
- ▶ What happens if we rename a class?

```
# [Attribute(Attribute::TARGET_CLASS)]  
class Friend  
{  
  
    /** @param class-string $friend */  
    public function __construct(  
        public string $friend,  
    ) {}  
}
```

```
# [Friend(TextMessageQueueProcessor::class)]  
class TextMessageSender  
{  
  
}
```

```
public function processNode(Node $node, Scope $scope): array
{
    $callingClass = $scope->getClassReflection()->getName();
    $type = $scope->getType($node->var);

    foreach ($type->getReferencedClasses() as $targetClass) {

        // TODO:
        // 1. Does $targetClass have a #[Friend] attribute
        // 2. Yes? Check $callingClass is a friend of $targetClass

    }

    return [];
}
```

```
class TextMessageSenderCallerRule implements Rule  
{
```

```
public function __construct(
```

```
    private ReflectionProvider $reflectionProvider,
```

```
) {}
```



```
foreach ($type->getReferencedClasses() as $targetClass) {
```

```
    // 1. Does $targetClass have a #[Friend] attribute
```

```
    $info = $this->reflectionProvider->getClass($targetClass);
```

```
    $nativeReflection = $info->getNativeReflection();
```

```
    $friendAttributes = $nativeReflection  
                        ->getAttributes(Friend::class);
```

```
    if (count($friendAttributes) !== 1) {  
        continue;  
    }
```

```
    $friendAttribute = $friendAttributes[0];  
    $friendArguments = $friendAttribute->getArguments();
```

```
    if (count($friendArguments) !== 1) {  
        continue;  
    }
```

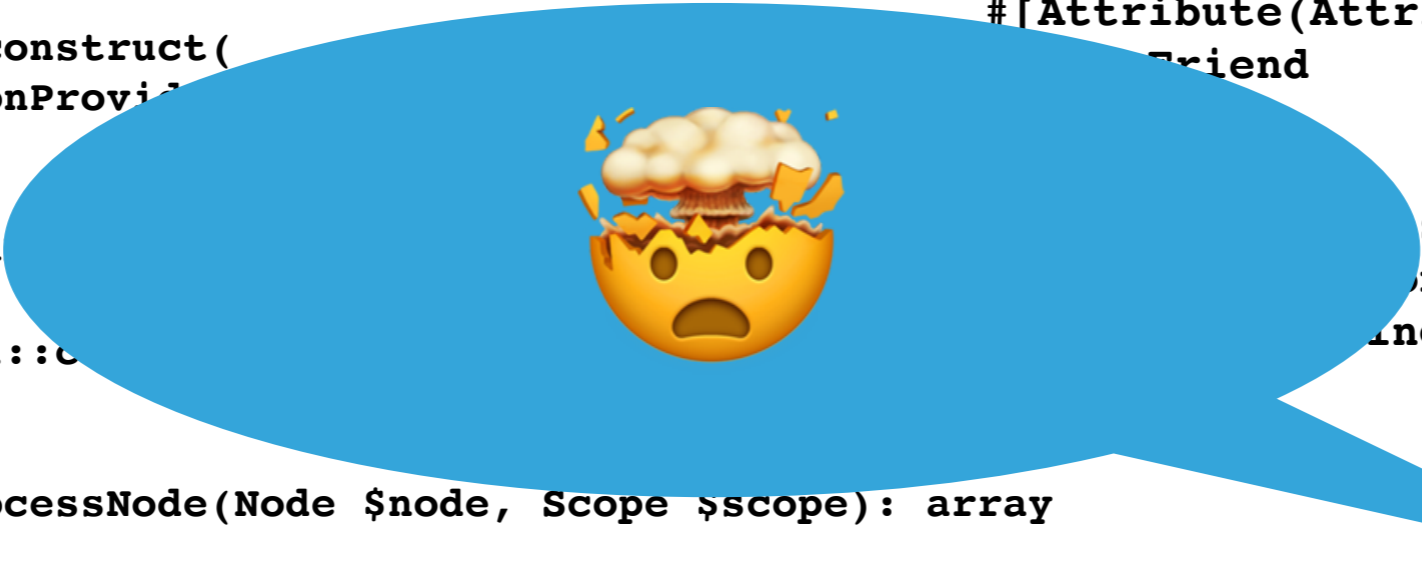
```
    $friendClass = $friendArguments[0];
```

```
foreach ($type->getReferencedClasses() as $targetClass) {  
    // Step 1 see previous slide  
  
    // 2. Yes? Check $callingClass is a friend of $targetClass  
  
    if ($callingClass !== $friendClass) {  
  
        $msg = sprintf(  
            "%s can only be called its friend %s and not from %s",  
            $targetClass,  
            $friendClass,  
            $callingClass);  
  
        return [RuleErrorBuilder::message($msg)->build()];  
  
    }  
  
}
```

```
class TextMessageSenderCallCheckRule implements Rule
{
    public function __construct(
        private ReflectionProvider $reflectionProvider,
        #[Attribute(Attribute::TARGET_CLASS)]
        Friend $friend
    ) {}

    public function getTargetClassString($friend) /*
    {
        return MethodCall::class;
    }

    public function processNode(Node $node, Scope $scope): array
    {
        $callingClass = $scope->getClassReflection()->getName();
        $nodeType = $scope->getType($node->var);
        $referencedClassReflection = $this->reflectionProvider->getNativeReflection($node->var->getNativeReflection());
        $attribute = $this->reflectionProvider->getNativeReflection($node->var->getNativeReflection());
        $attributeCount = count($friendArguments);
        if ($attributeCount !== 1) {
            continue;
        }
        $friend = $friendArguments[0];
        if ($callingClass !== $friend) {
            $msg = sprintf("Can not call %s from %s", $targetClass, $callingClass);
            return [RuleErrorBuilder::message($msg)->build()];
        }
    }
}
```



I KNOW, IT'S AMAZING.
2 HOURS OF WORK VS RFC

<https://github.com/DaveLiddament/phpstan-rule-demo>

Custom Static Analysis Rules

+

Attributes

=

New Language Features

PHP Extension Library

**RFC vs
Static
Analysis**



AST



**PHPStan
rule**



Thoughts



main 1 branch 3 tags

Go to file Add file Code

	DaveLiddament Merge pull request #6 from DaveLiddament/fix/test-tag-attrib... aa2f632 8 days ago 22 commits
	.github/workflows FIX remove unused section of github actions 2 months ago
	examples ADD InjectableVersion Attribute 9 days ago
	src FIX copy and paste error with TestTag 8 days ago
	.gitignore ADD linting, coding standards and phpstan 2 months ago
	.php-cs-fixer.php ADD linting, coding standards and phpstan 2 months ago
	CONTRIBUTING.md ADD contributing notes and update example code 2 months ago
	LICENSE.md INITIAL CHECK IN 2 months ago
	README.md ADD InjectableVersion Attribute 9 days ago
	composer.json ADD keywords to composer.json 2 months ago
	composer.lock ADD psalm 2 months ago
	phpstan.neon UPDATE Friend and Sealed to use variadic for friends/permitted 2 months ago
	psalm.xml ADD psalm 2 months ago

README.md

PHP Language Extensions (currently in BETA)

This library provides attributes for extending the PHP language (e.g. adding `package` visibility). The intention, at least initially, is that these extra language features are enforced by static analysis tools (such as Psalm, PHPStan and, ideally, PhpStorm) and NOT at runtime.

Language feature added:

- Friend
- InjectableVersion
- Package
- Sealed
- TestTag

About

Attributes to define PHP language extensions (to be enforced by static analysis)

Readme MIT license 16 stars 2 watching 1 fork

Releases 3

0.2.1 Latest 8 days ago

+ 2 releases

Packages

No packages published Publish your first package

Contributors 2

DaveLiddament Dave Liddament ruudk Ruud Kamphuis

Languages

PHP 100.0%

<https://github.com/DaveLiddament/php-language-extensions>

```
class Person
```

```
{
```

```
    #[Friend(PersonBuilder::class)]
```

```
    public function __construct()
```

```
    {
```

```
    }
```

```
}
```

```
class DiscountCalculator  
{  
  
    # [NamespaceVisibility]  
    public function calculate(): int  
    {  
    }  
  
}
```



```
class Person
```

```
{
```

```
    # [TestTag]
```

```
    public function setId(int $id)
```

```
    {
```

```
    }
```

```
}
```

```
# [Sealed(Success::class, Failure::class)]  
abstract class Result  
{  
}
```

```
# [InjectableVersion]
```

```
interface class PersonRepository {...}
```

```
class DoctrinePersonRepository  
    implements PersonRepository {...}
```

```
class PersonCreator {  
    public function __construct(  
        PersonRepository $personRepository,  
    ) {...}  
}
```

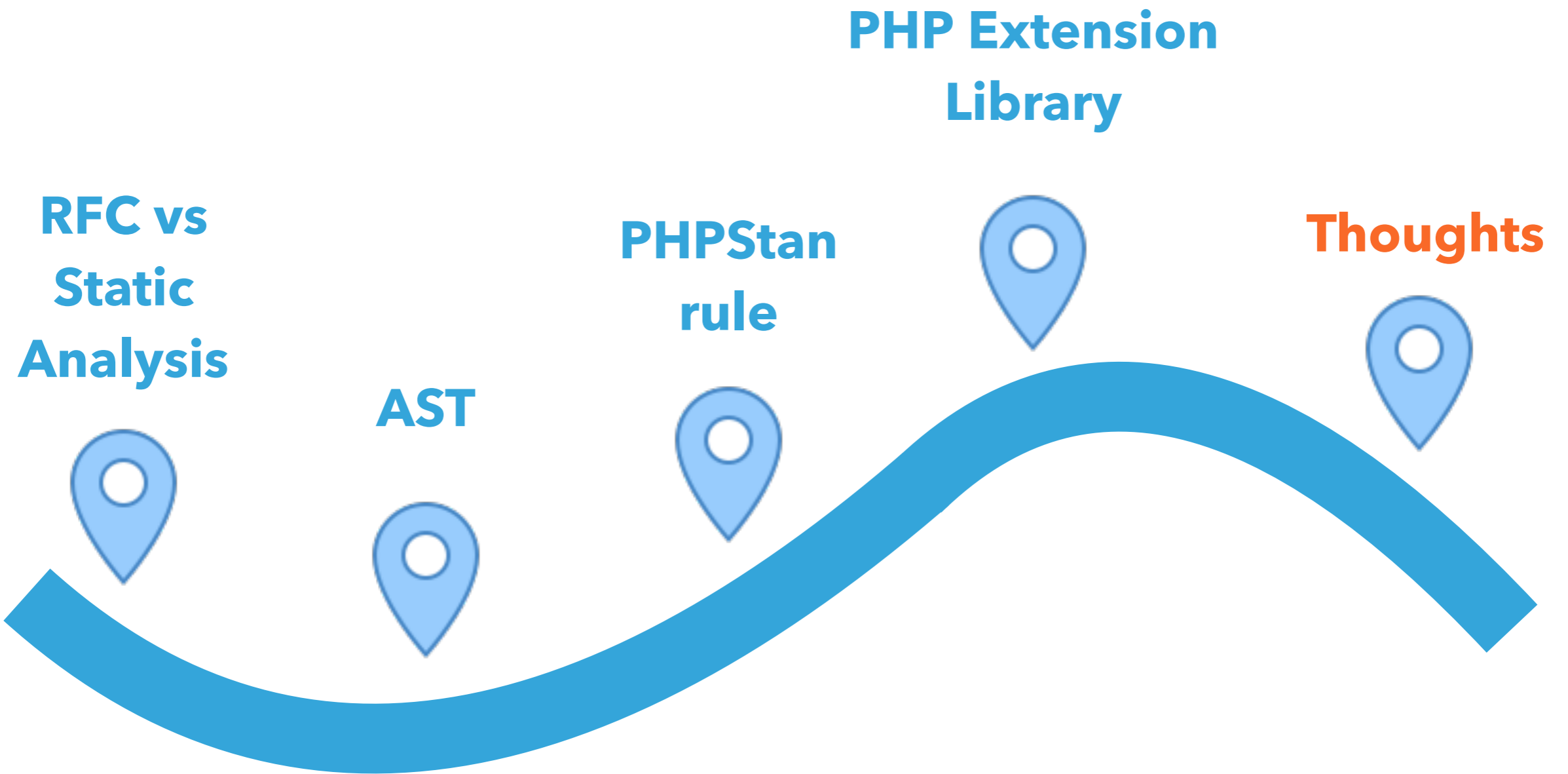
Definition

<https://github.com/DaveLiddament/php-language-extensions>



<https://github.com/DaveLiddament/phpstan-php-language-extensions>





SYNTACTIC SUGAR

SECURITY

PERFORMANCE

NEW FEATURES

**COMMUNICATING
INTENTION**

OVERLAP

SYNTACTIC SUGAR

SECURITY

PERFORMANCE

NEW FEATURES

**COMMUNICATING
INTENTION**

public

final

protected

abstract

private

readonly

type declarations

Friend

type hints

Sealed



NamespaceVisibility

InjectableVersion

AUTOMATE CHECKS FOR VIOLATIONS

Try out ideas...

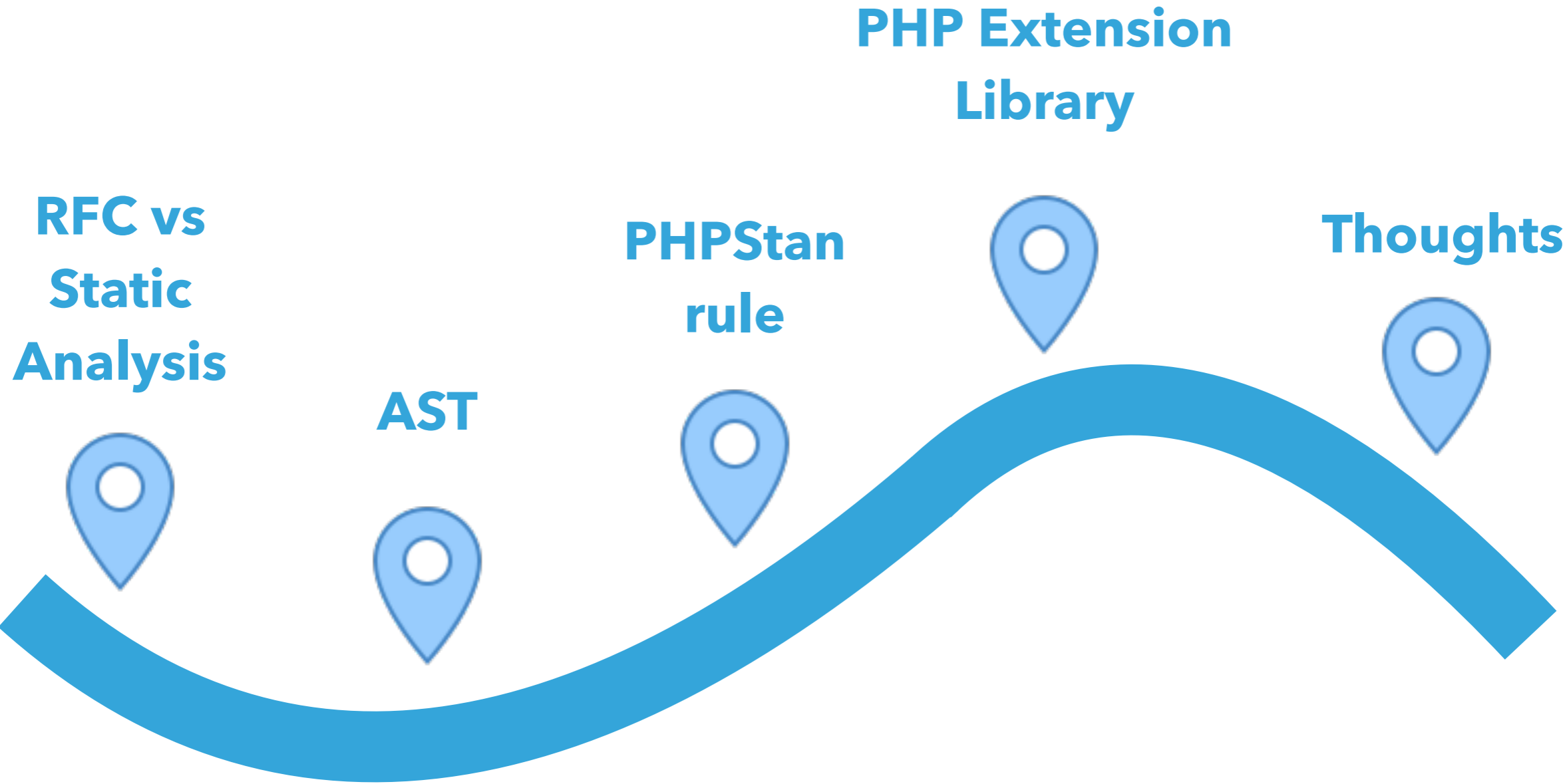
Today

```
#[Friend(TextMessageQueueProcessor::class)]  
class TextMessageSender  
{  
  
}
```

PHP 8.3+ ?

```
class TextMessageSender  
    friend TextMessageQueueProcessor  
{  
  
}
```

never return type (added in PHP 8.1)



Custom static analysis rules allows developers to:

- ▶ automatically enforce project conventions
- ▶ create new language features

Thank you for
listening

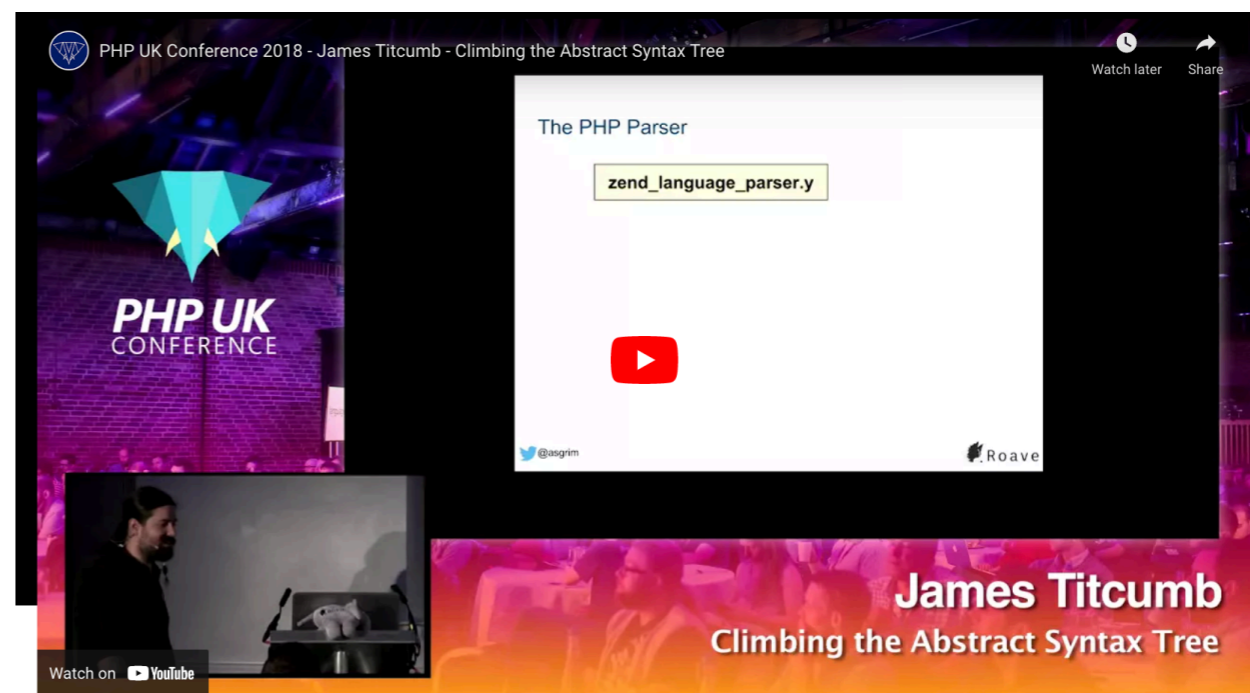
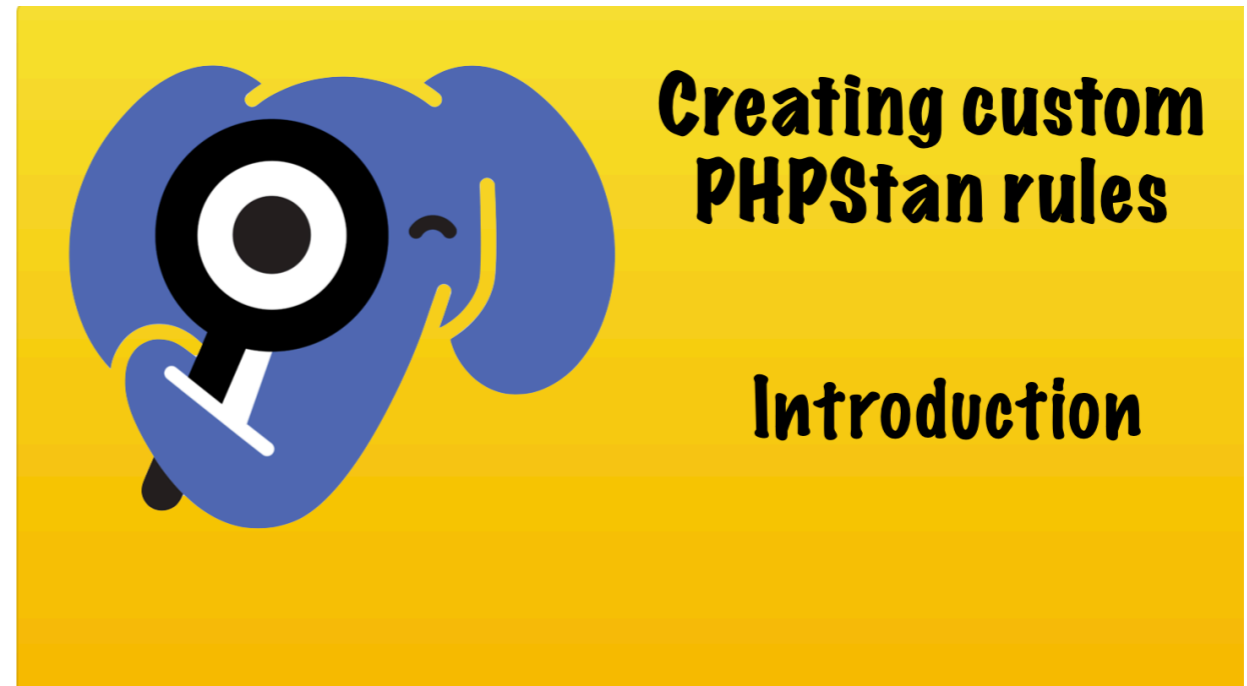
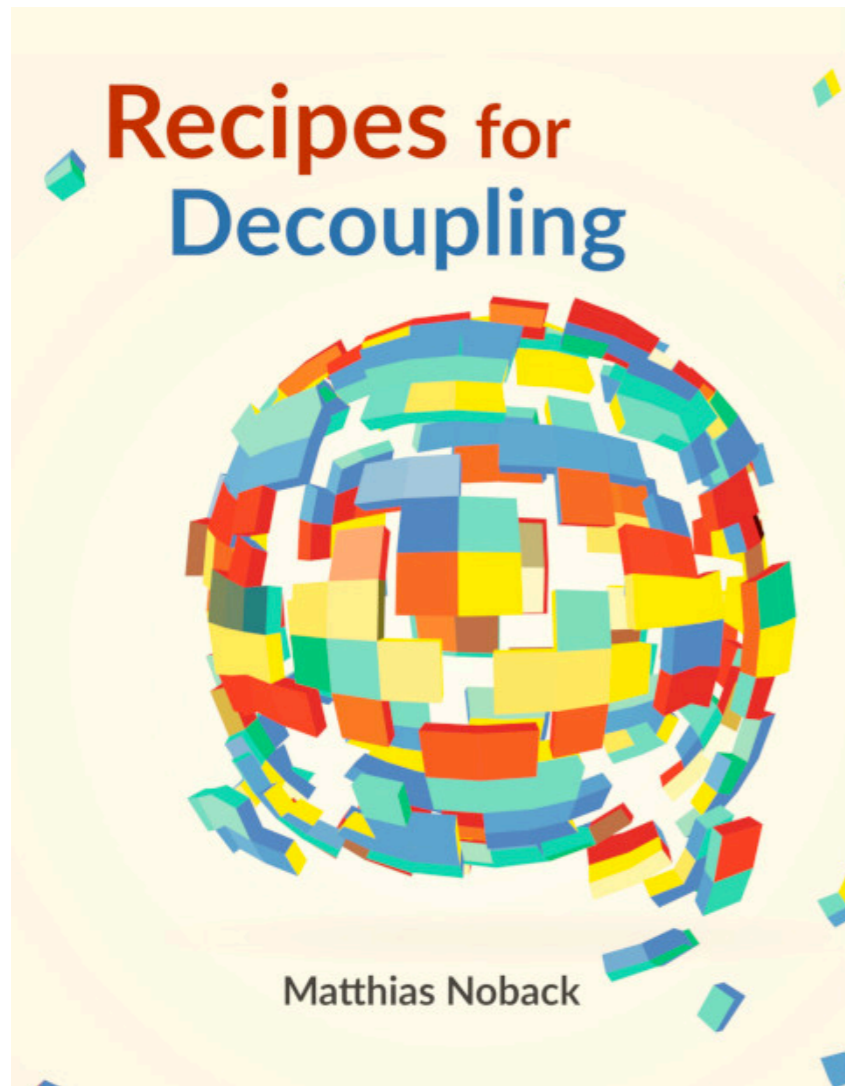


Dave Liddament

@daveliddament

Further information

<https://phpstan.org/developing-extensions/rules>



PHP LANGUAGE

EXTENSIONS

LIBRARY

#[Friend]

Only allowed to
call method from
specified classes

```
class Person
{
    #[Friend(PersonBuilder::class)]
    public function __construct()
    {
        ...
    }
}

class PersonBuilder
{
    public function create(): Person {
        return new Person(): ✓
    }
}

class AnotherClass
{
    public function doSomething() {
        $person = new Person();
    }
}
```

#[Friend]

Only allowed to
call method from
specified classes

```
class Person
{
    #[Friend(PersonBuilder::class)]
    public function __construct()
    {
        ...
    }
}

class PersonBuilder
{
    public function create(): Person {
        return new Person();
    }
}

class AnotherClass
{
    public function doSomething() {
        $person = new Person();
    }
}
```


#[Package]

Only allowed
to call methods
from the same
namespace

```
namespace ShoppingBasket;
```

```
class PriceCalculator {  
    public function calculatePrice(): Money {  
        // Some code
```

```
DiscountCalculator::getDiscount($items); ✓
```

```
class DiscountCalculator {  
    #[Package]  
    public static function getDiscount($items)  
        // Some code  
        return $discount;  
    }  
}
```

```
namespace ShoppingBasket;
```

```
class PriceCalculator {  
    public function calculatePrice(): Money {  
        // Some code  
        DiscountCaluclator::getDiscount($items);  
    }  
}
```

```
class DiscountCalculator {  
    #[Package]  
    public static function getDiscount($items) {  
        // Some code  
        return $discount;  
    }  
}
```

```
namespace SomeNamespace;
```

```
class SomeClass {  
    public function SomeMethod() {
```

```
        // some code
```

```
        DiscountCaluclator::getDiscount($items); ❌
```

#[TestTag]

Only allowed to
call method from
test code

```
class Person
{
    #[TestTag]
    public function setId(int $id) {
        $this->id = $id;
    }
}

function updatePerson(Person $person) {
    $person->setId(3);
}

// Tests
class MyTest
{
    public function testSomething() {
        $person = new Person();
        $person->setId(6); ✓
    }
}
```

#[TestTag]

Only allowed to
call method from
test code

```
class Person
{
    #[TestTag]
    public function setId(int $id) {
        $this->id = $id;
    }
}

function updatePerson(Person $person) {
    $person->setId(3); ❌
}

// Tests
class MyTest
{
    public function testSomething() {
        $person = new Person();
        $person->setId(6);
    }
}
```

#[InjectableVersion]

If the class hierarchy contains an interface or class with this attribute only that version may be injected into constructors

```
#[InjectableVersion]
interface EMailer {
    public function sendEmail();
}

class PhpMailer implements EMailer {
    public function sendEmail()
    { ... }
}

class SignupService {
    public function __construct(
        public EMailer $emailer, ✓
    ) {}
}

class MarketingService {
    public function __construct(
        public PhpMailer $emailer,
    ) {}
}
```

#[InjectableVersion]

If the class hierarchy contains an interface or class with this attribute only that version maybe injected into constructors

```
#[InjectableVersion]
interface EMailer {
    public function sendEmail();
}

class PhpMailer implements EMailer {
    public function sendEmail()
    { ... }
}

class SignupService {
    public function __construct(
        public EMailer $emailer,
    ) {}
}

class MarketingService {
    public function __construct(
        public PhpMailer $emailer,
    ) {}
}
```

Attributes

<https://github.com/DaveLiddament/php-language-extensions>

PHPStan

<https://github.com/DaveLiddament/phpstan-php-language-extensions>

Psalm and others

Contributions welcome

<https://github.com/DaveLiddament/php-language-extensions>

- examples
 - callableFrom
 - friend
 - injectableVersion
 - namespaceVisibility
 - package
 - sealed

- testTag
 - testTagOnConstructor.php
 - testTagOnConstructorIgnoredInTestClass.php
 - testTagOnConstructorIgnoredInTestNamespace.php
 - testTagOnMethod.php
 - testTagOnMethodIgnoredInTestClass.php
 - testTagOnMethodIgnoredInTestNamespace.php
 - testTagOnStaticMethod.php
 - testTagOnStaticMethodIgnoredInTestClass.php
 - testTagOnStaticMethodIgnoredInTestNamespace.php

- src
 - CallableFrom.php
 - CheckInjectableVersion.php