Extending the PHP Language with static analysis

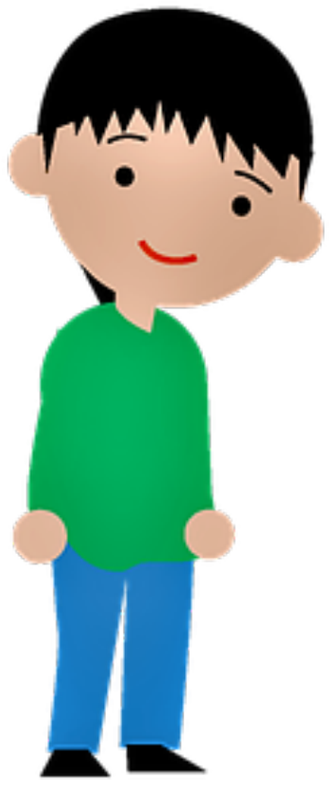
Dave Liddament

Lamp Bristol

@daveliddament

@DaveLiddament@phpc.social

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

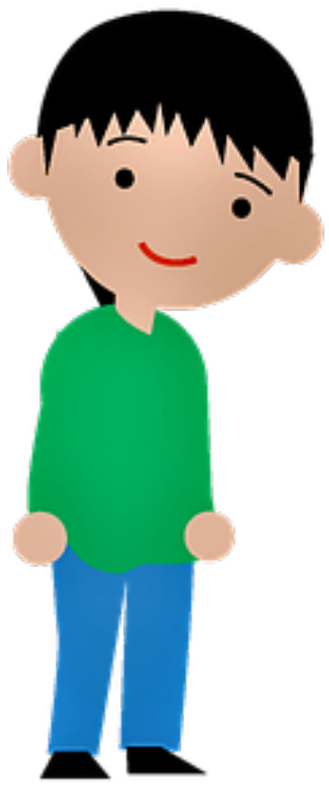


WOULDN'T IT BE GREAT IF PHP COULD
DO . . .

@daveliddament

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

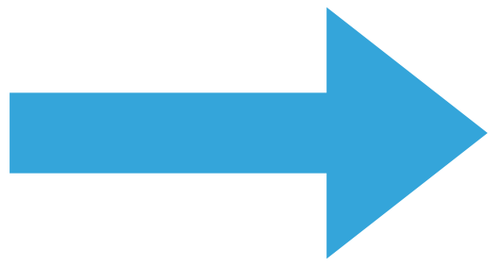
WOULDN'T IT BE GREAT IF PHP COULD
DO . . .



HOW I BUILT NEW LANGUAGE FEATURES

@daveliddament

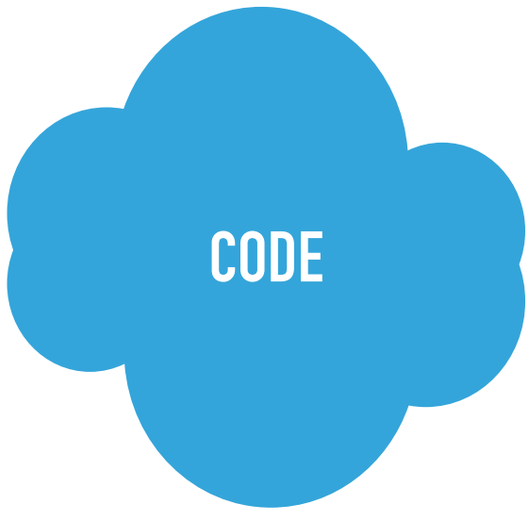
Very specific constraint



Generalised version that could be a useful on your project

Preconditions

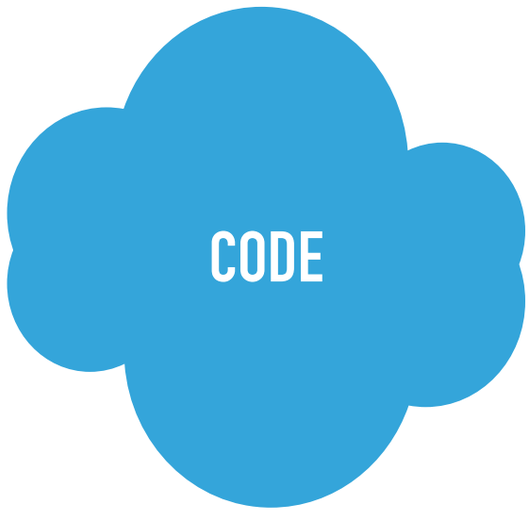
One of many examples…



TEXT MESSAGE SENDER

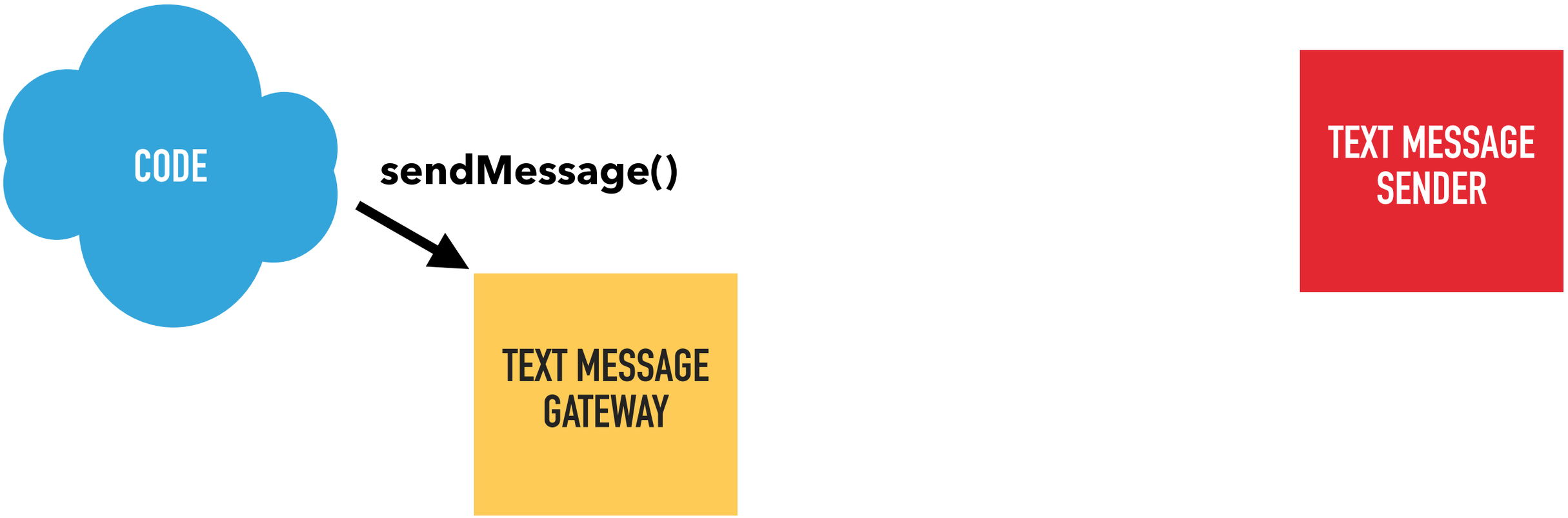
One of many examples…

One of many examples…

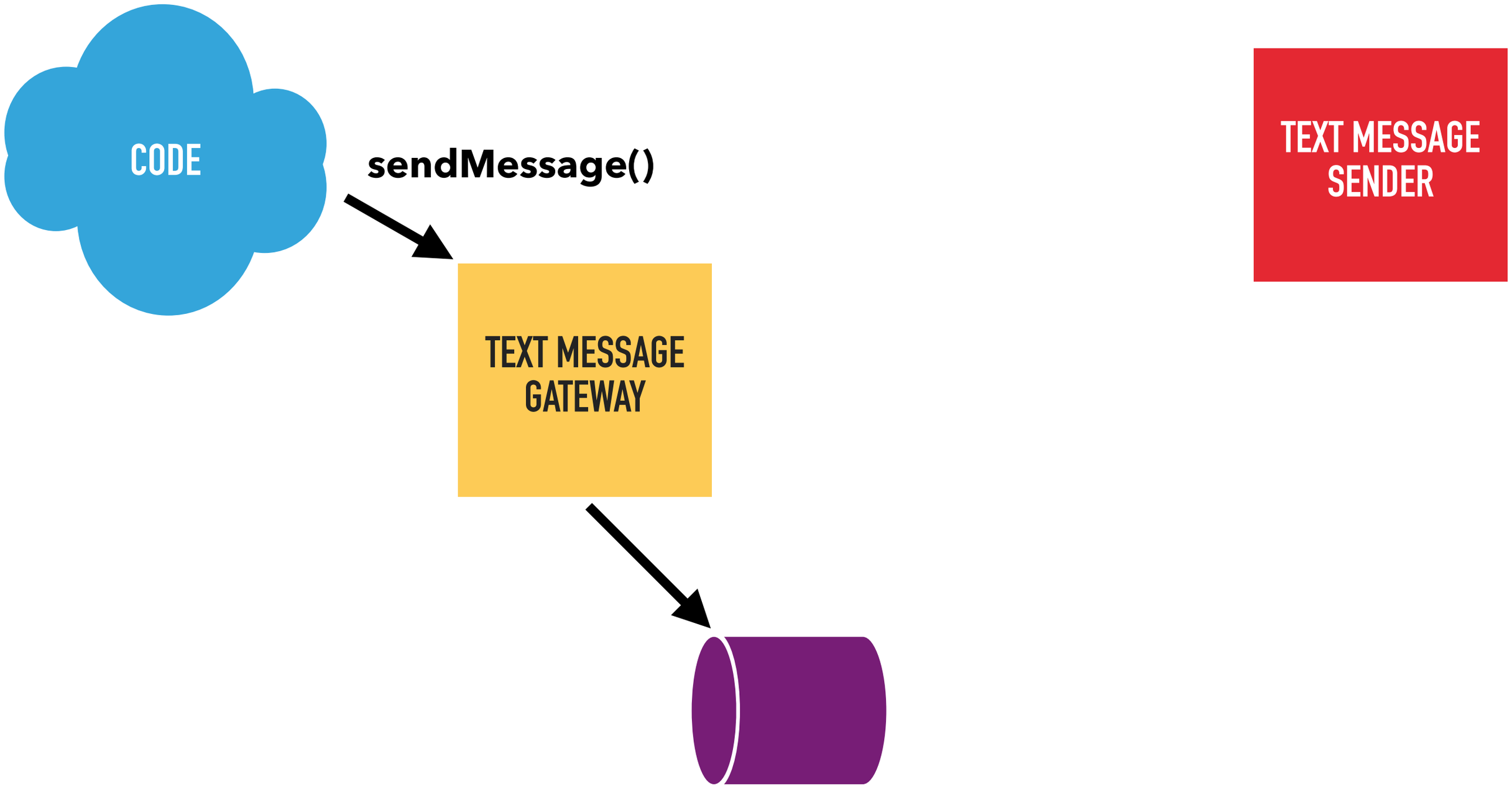


TEXT MESSAGE SENDER

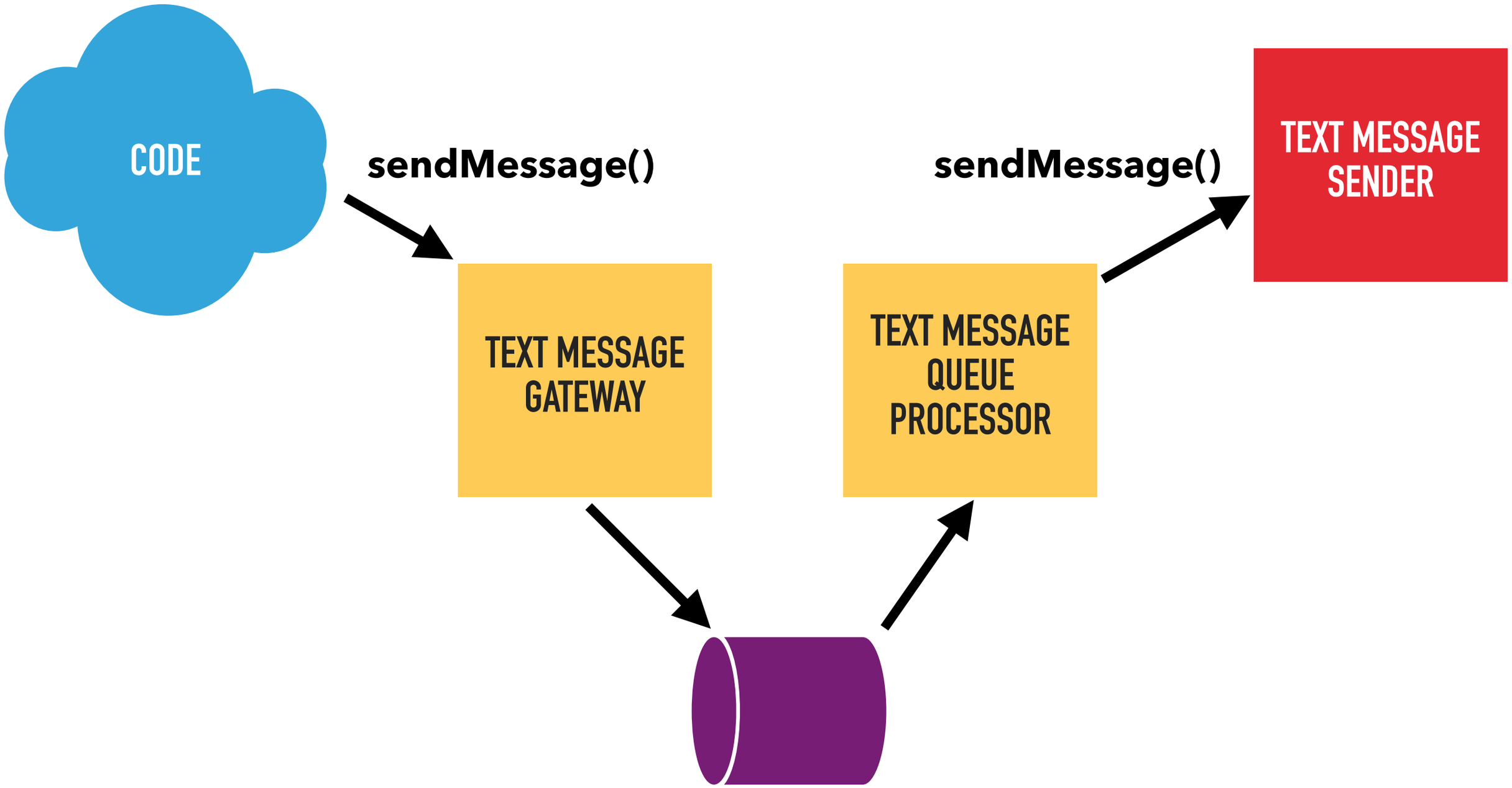
One of many examples…



One of many examples…



One of many examples…



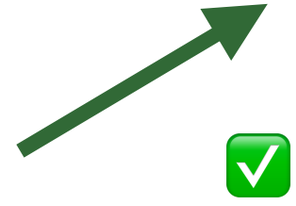
sendMessage()

TEXT MESSAGE SENDER

sendMessage()

TEXT MESSAGE SENDER

TEXT MESSAGE QUEUE PROCESSOR



✅

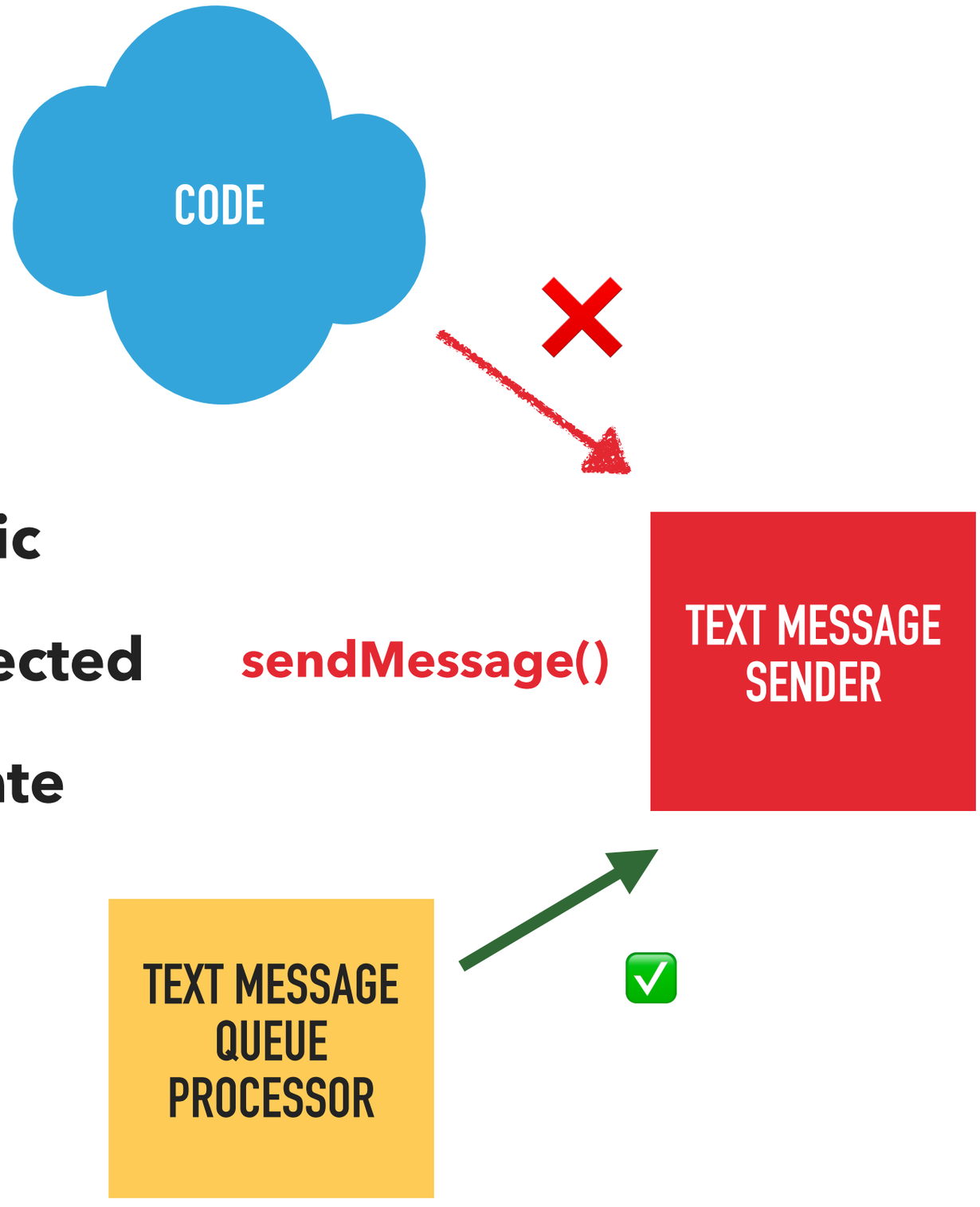
CODE

sendMessage()

TEXT MESSAGE SENDER

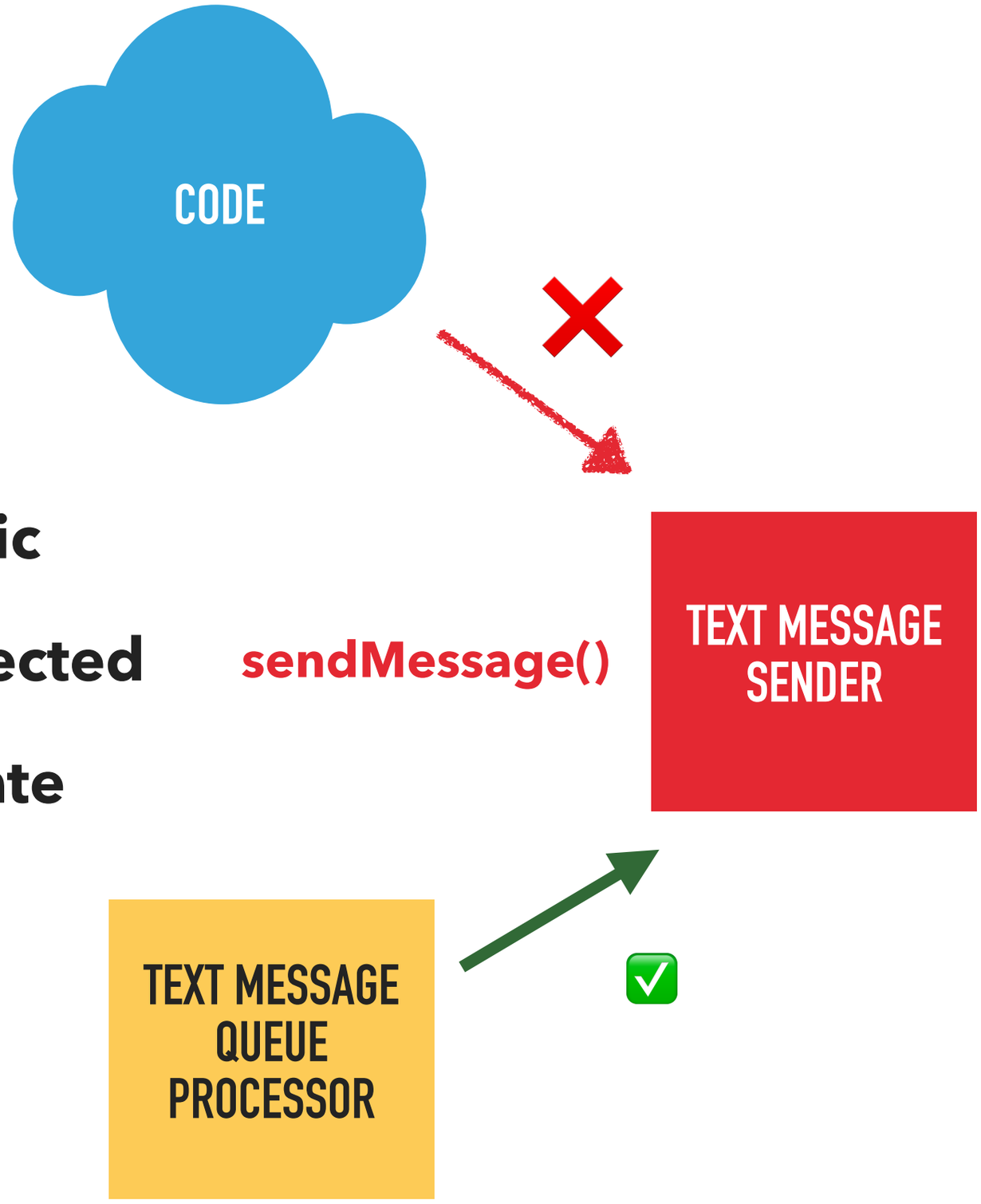
TEXT MESSAGE QUEUE PROCESSOR

✅


CODE
public
protected
private
sendMessage()
TEXT MESSAGE SENDER
TEXT MESSAGE QUEUE PROCESSOR
✅

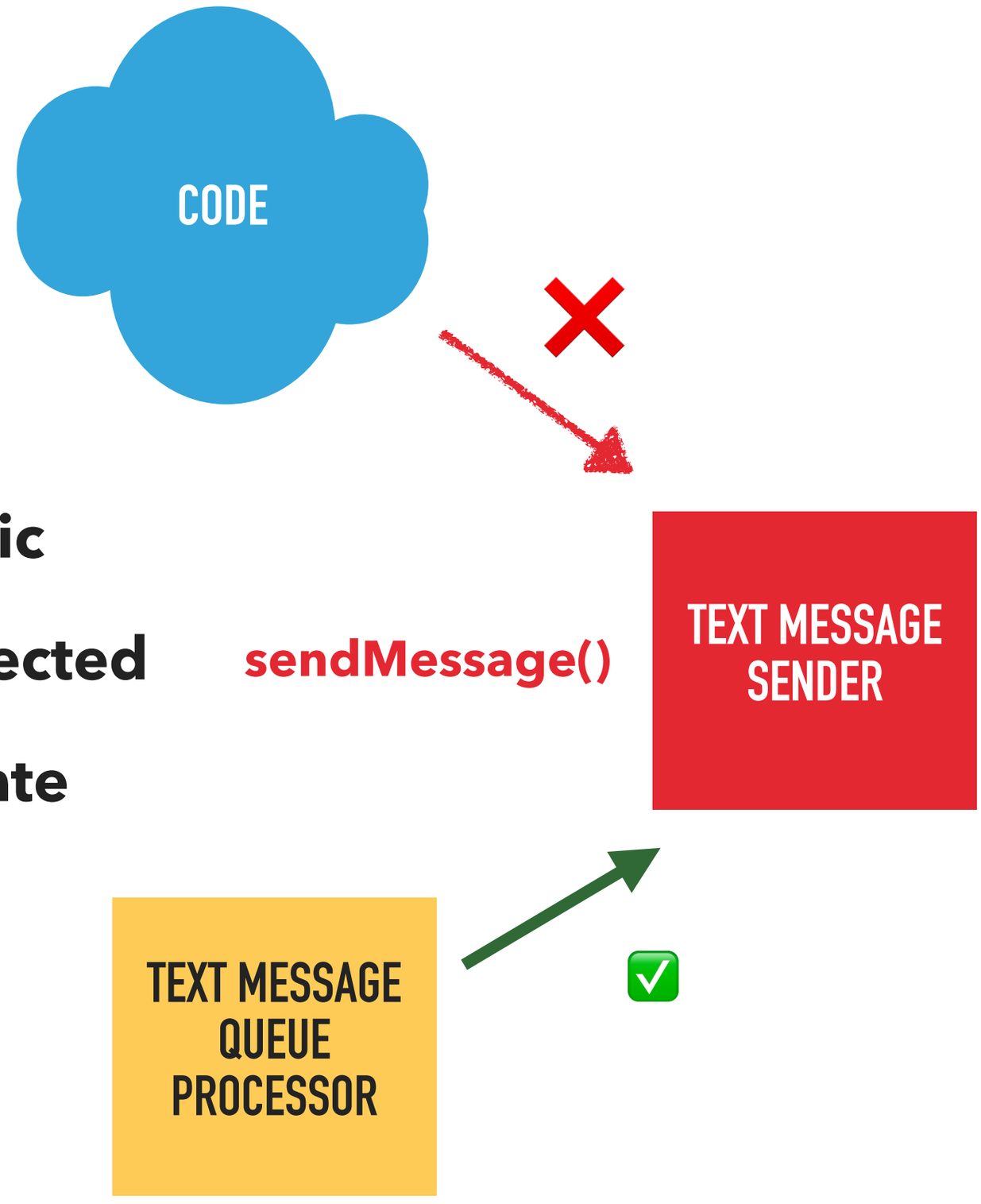
Existing visibility modifiers are not fine grained enough.

We need more control.



Existing visibility modifiers are not fine grained enough.

We need more control.



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

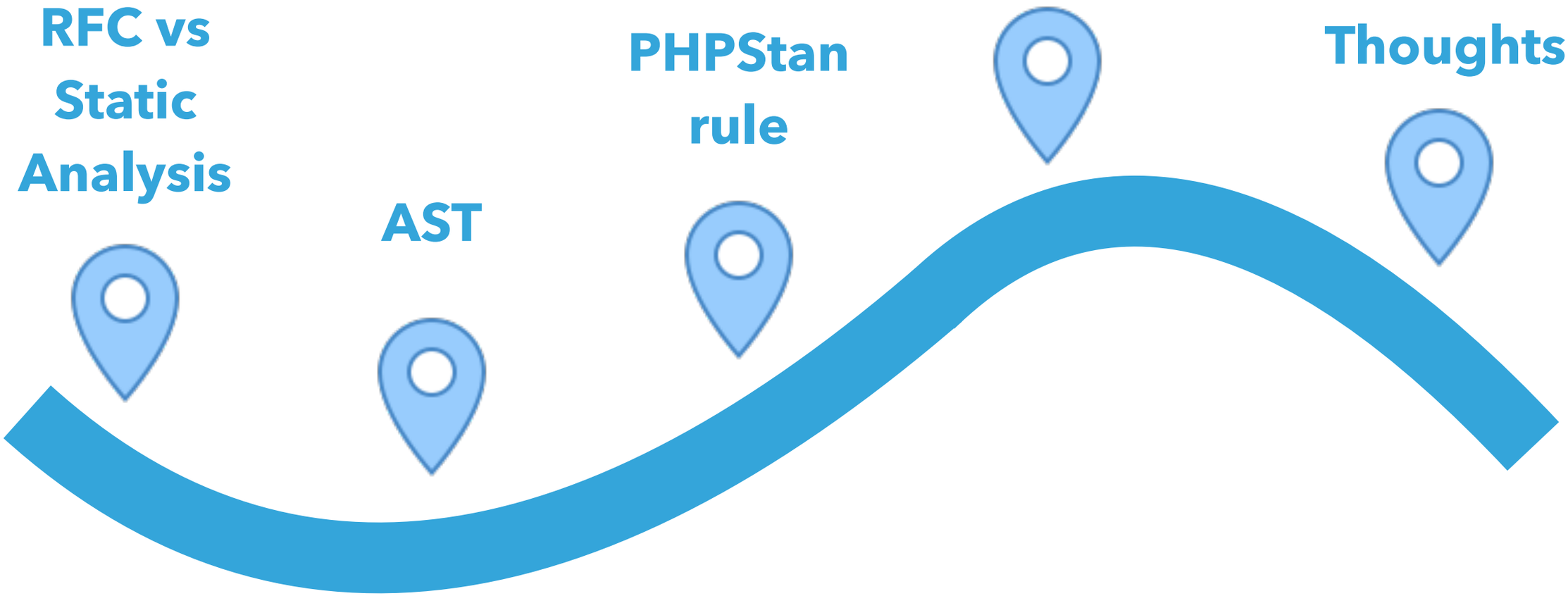
RFC vs Static Analysis

PHPStan rule

AST

PHP Extension Library

Thoughts



@daveliddament

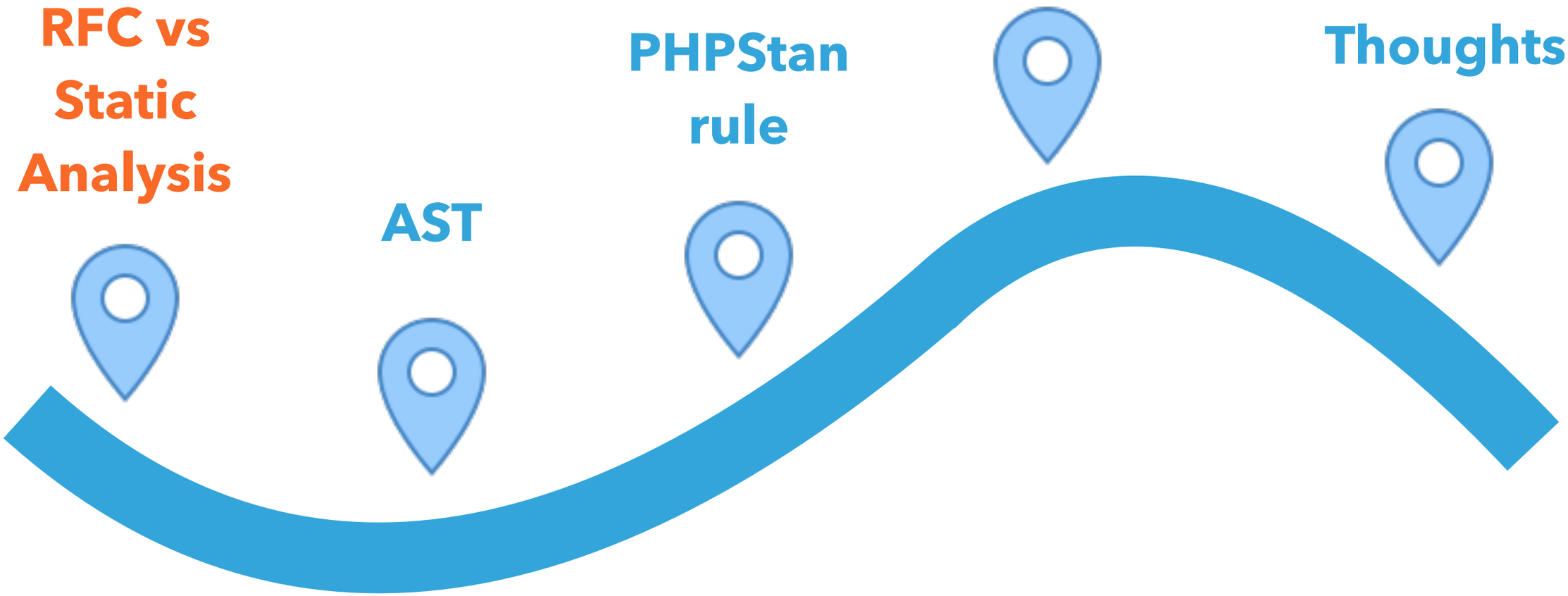
RFC vs Static Analysis

PHPStan rule

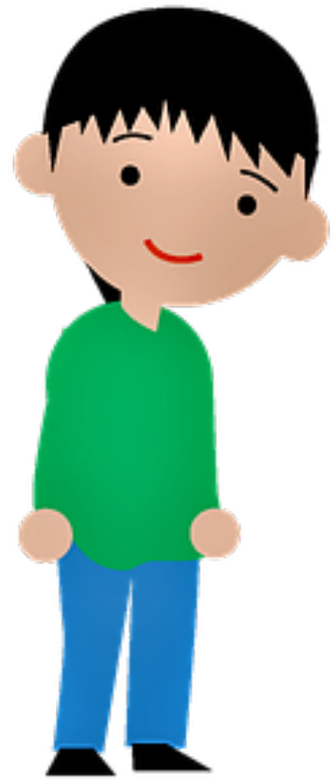
AST

PHP Extension Library

Thoughts



TELL ME ABOUT PHP'S RFC PROCESS



@daveliddament

TELL ME ABOUT PHP'S RFC PROCESS

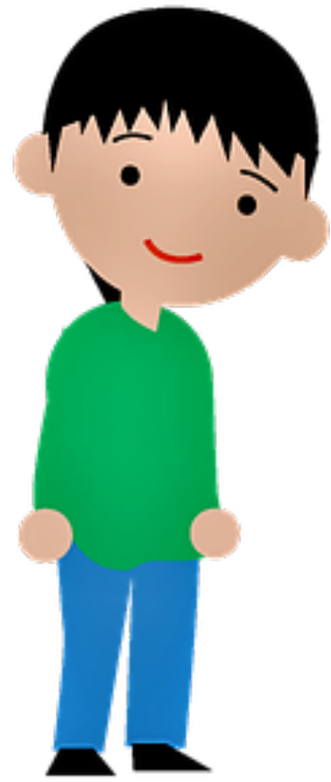
FIRST YOUR WRITE AN RFC

@daveliddament

TELL ME ABOUT PHP'S RFC PROCESS

FIRST YOUR WRITE AN RFC

TALK



@daveliddament

TELL ME ABOUT PHP'S RFC PROCESS

FIRST YOUR WRITE AN RFC

TALK

VOTE

@daveliddament

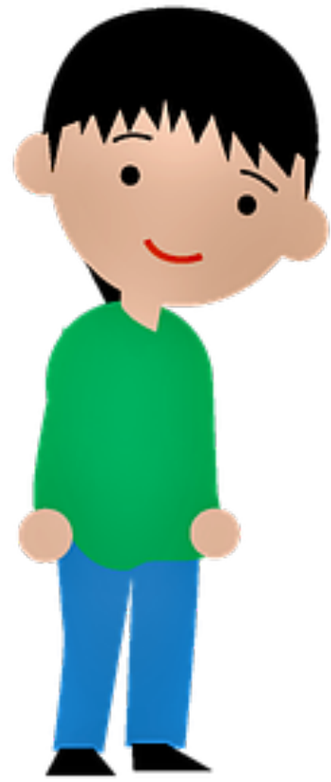
TELL ME ABOUT PHP'S RFC PROCESS

FIRST YOUR WRITE AN RFC

TALK

VOTE

IMPLEMENT



@daveliddament

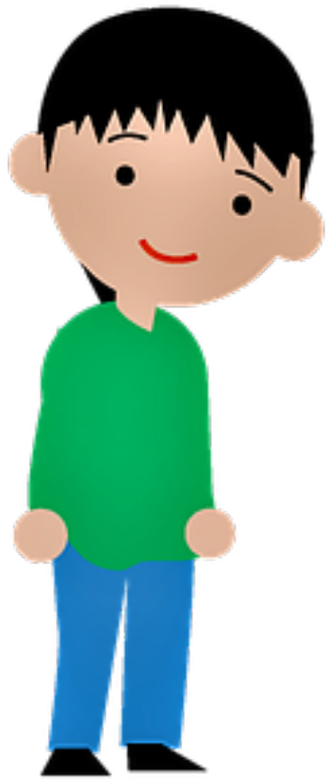
TELL ME ABOUT PHP'S RFC PROCESS

FIRST YOUR WRITE AN RFC

TALK

VOTE

IMPLEMENT



A LONG AND DIFFICULT PROCESS!

@daveliddament

THERE IS ANOTHER WAY . . .

THERE IS ANOTHER WAY . . .

. . . FOR SOME FUNCTIONALITY

Run time

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

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

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

```
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);
}
```

Static analysis gives us generics now

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

function process(Car $car) {…}

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

Static analysis gives us generics now

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

function process(Car $car) {…}

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

Static analysis gives us generics now

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

function process(Car $car) {…}

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

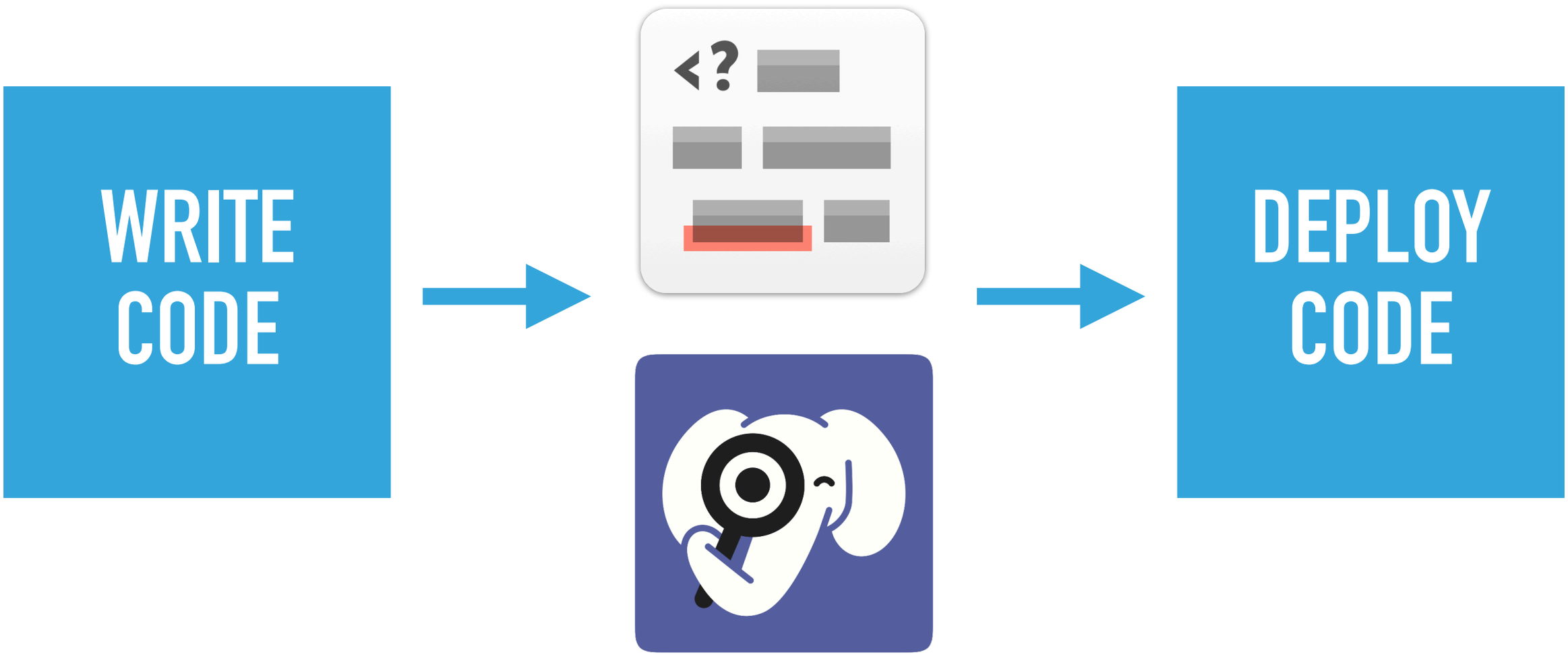
Static analysis gives us generics now

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

function process(Car $car) {…}

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

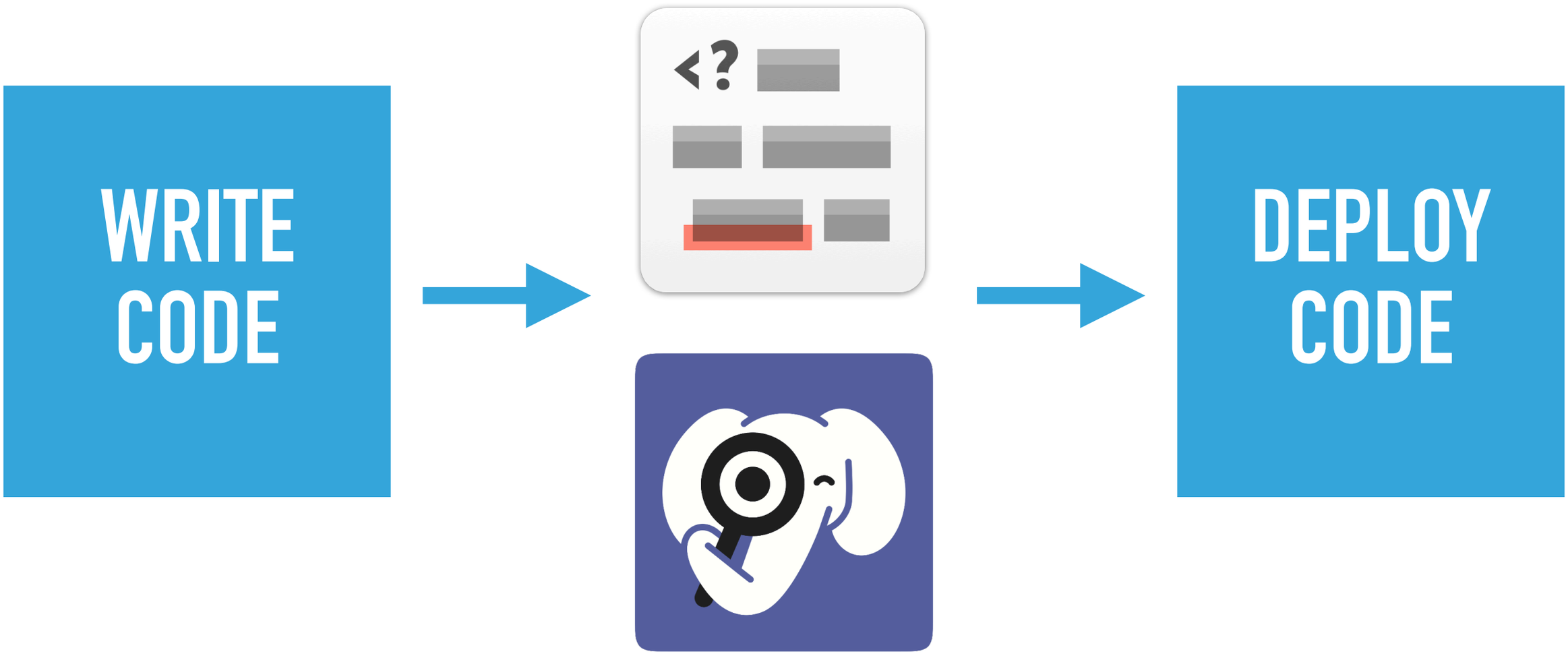
Static analysis gives us generics now

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

function process(Car $car) {…}

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

Add static analysis to dev process



Add static analysis to dev process



Create custom rules to emulate new language features

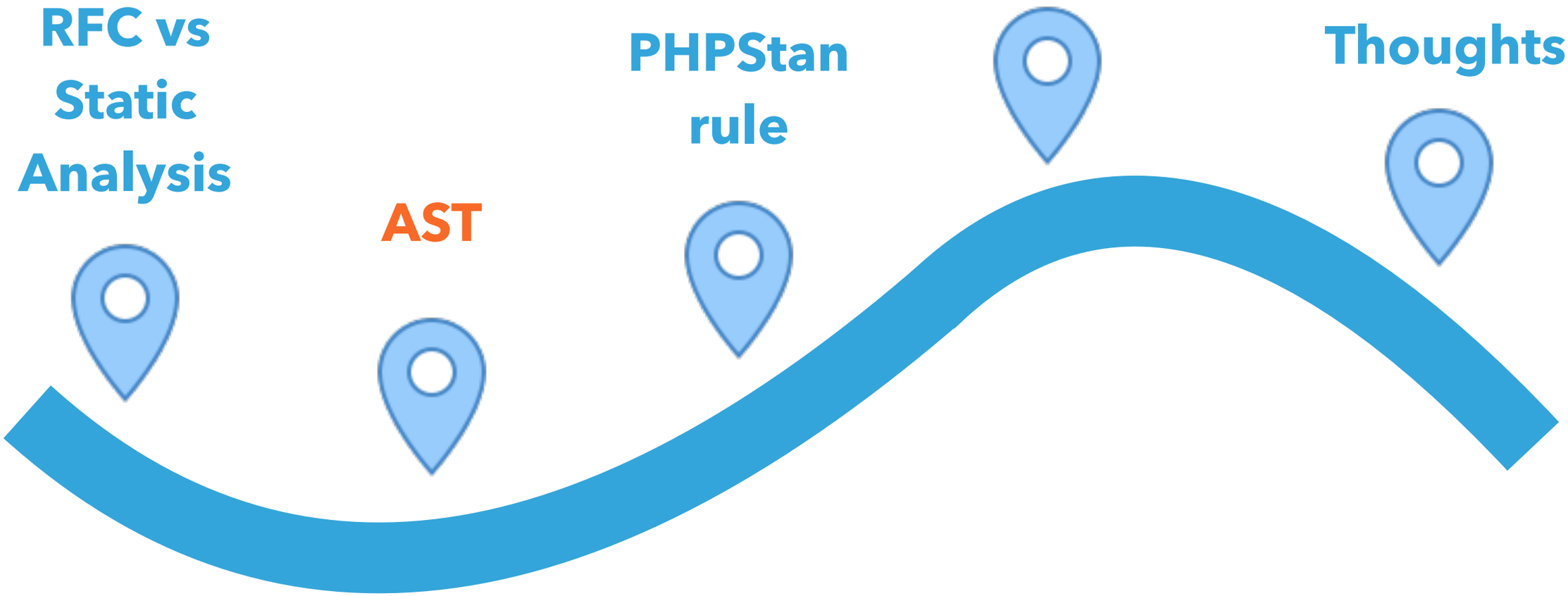
PHP Extension Library

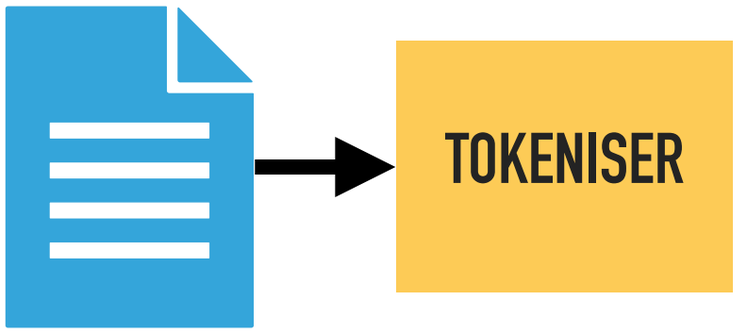
RFC vs Static Analysis

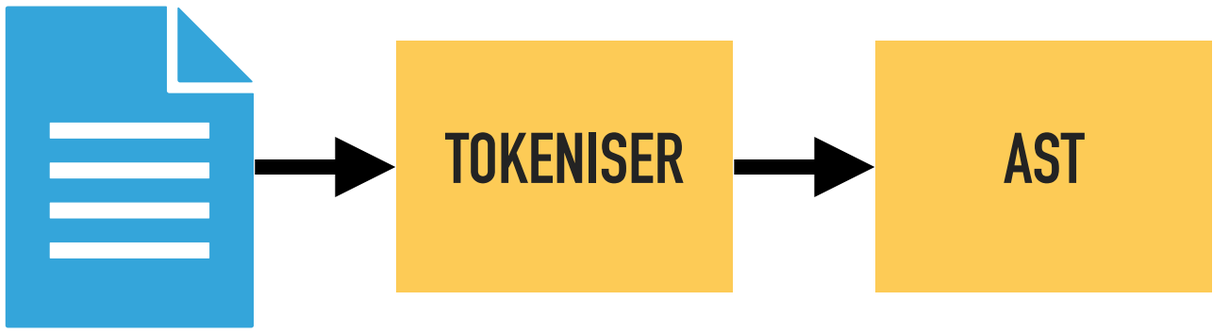
PHPStan rule

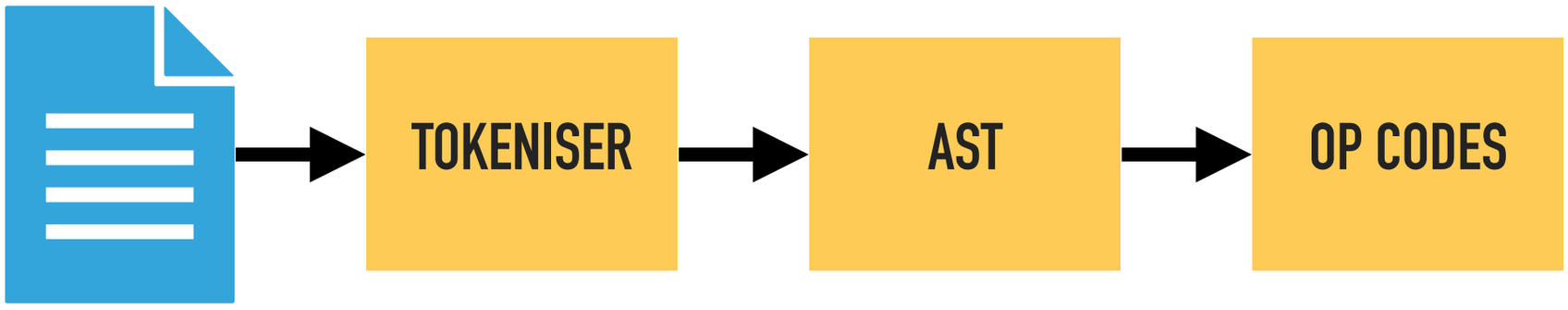
Thoughts

AST

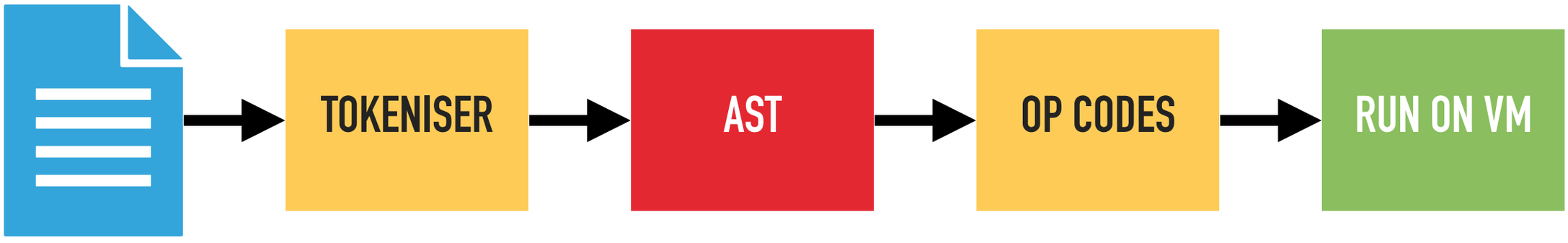



TOKENISER


TOKENISER
AST


TOKENISER
AST
OP CODES


TOKENISER
AST
OP CODES
RUN ON VM

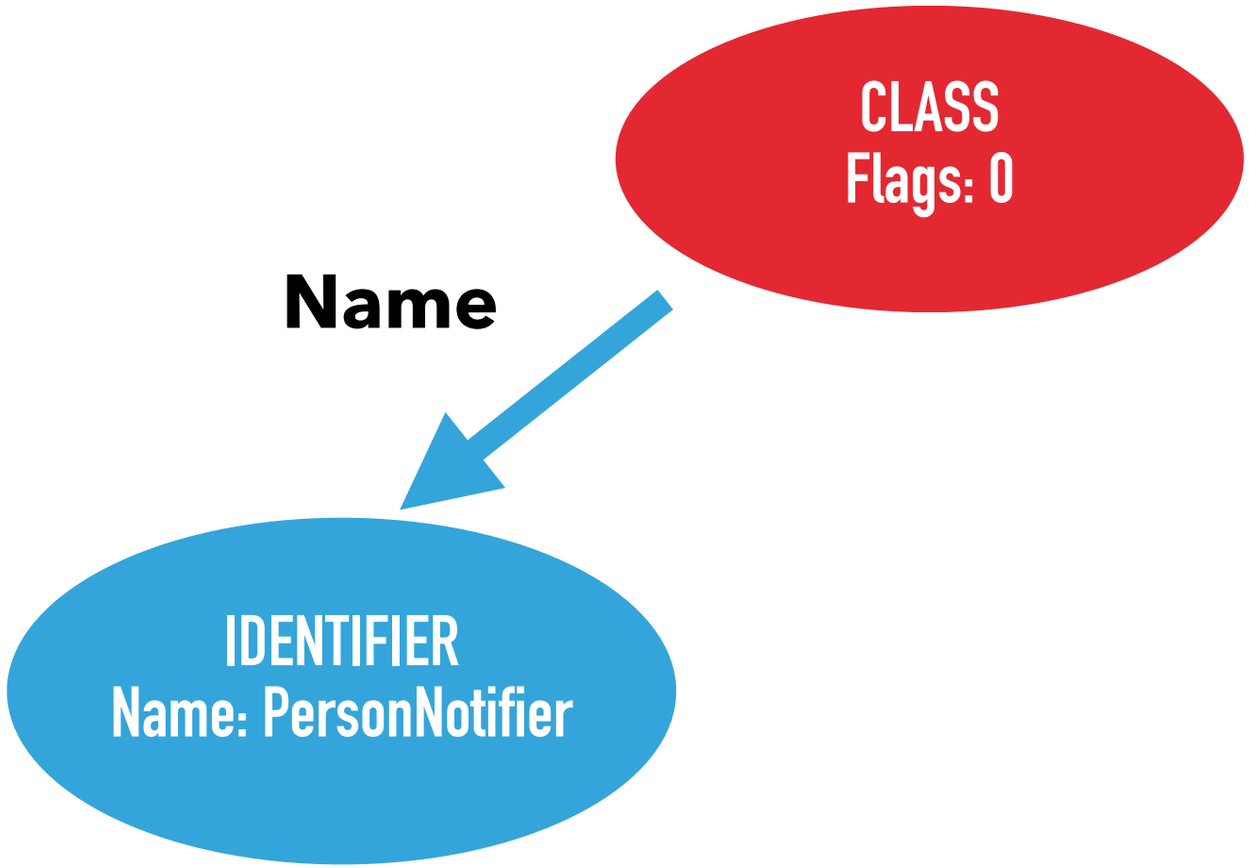

TOKENISER
AST
OP CODES
RUN ON VM

```
class PersonNotifier
{
  private TextMessageSender $sender;
  public function __construct() {…}
  public function notifyPlayer() {…}
}
```

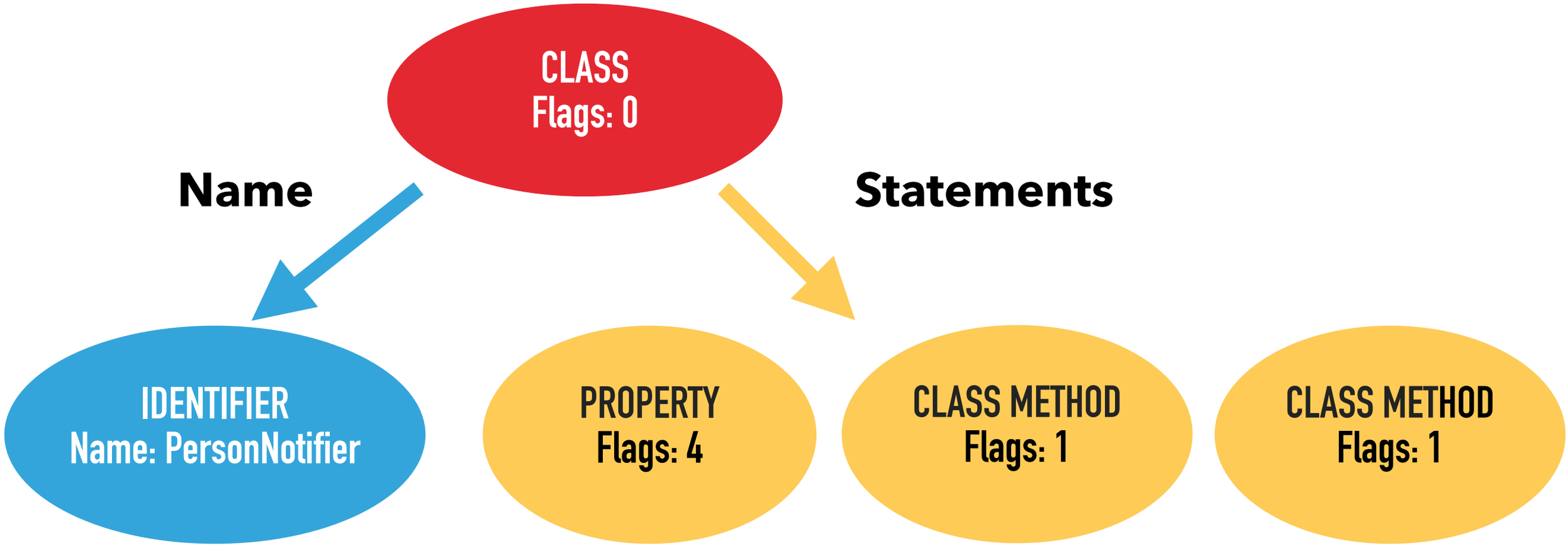
```
class PersonNotifier
{
  private TextMessageSender $sender;
  public function __construct() {…}
  public function notifyPlayer() {…}
}
```

CLASS
Flags: 0

```
class PersonNotifier
{
  private TextMessageSender $sender;
  public function __construct() {…}
  public function notifyPlayer() {…}
}
```



```
class PersonNotifier
{
    private TextMessageSender $sender;
    public function __construct() {…}
    public function notifyPlayer() {…}
}
```

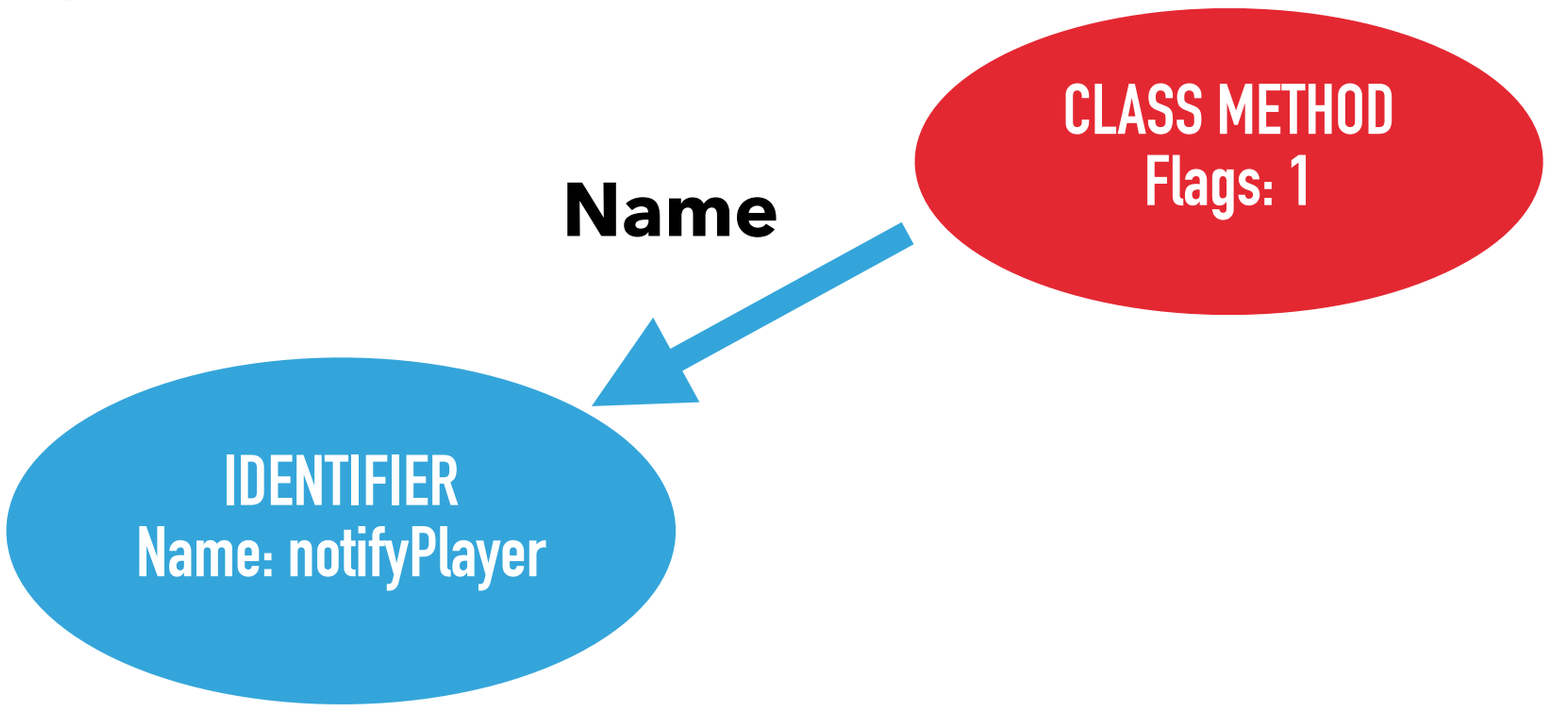


```
public function notifyPlayer (string $msg)
{
   $this->sender->sendMessage($msg);
}
```

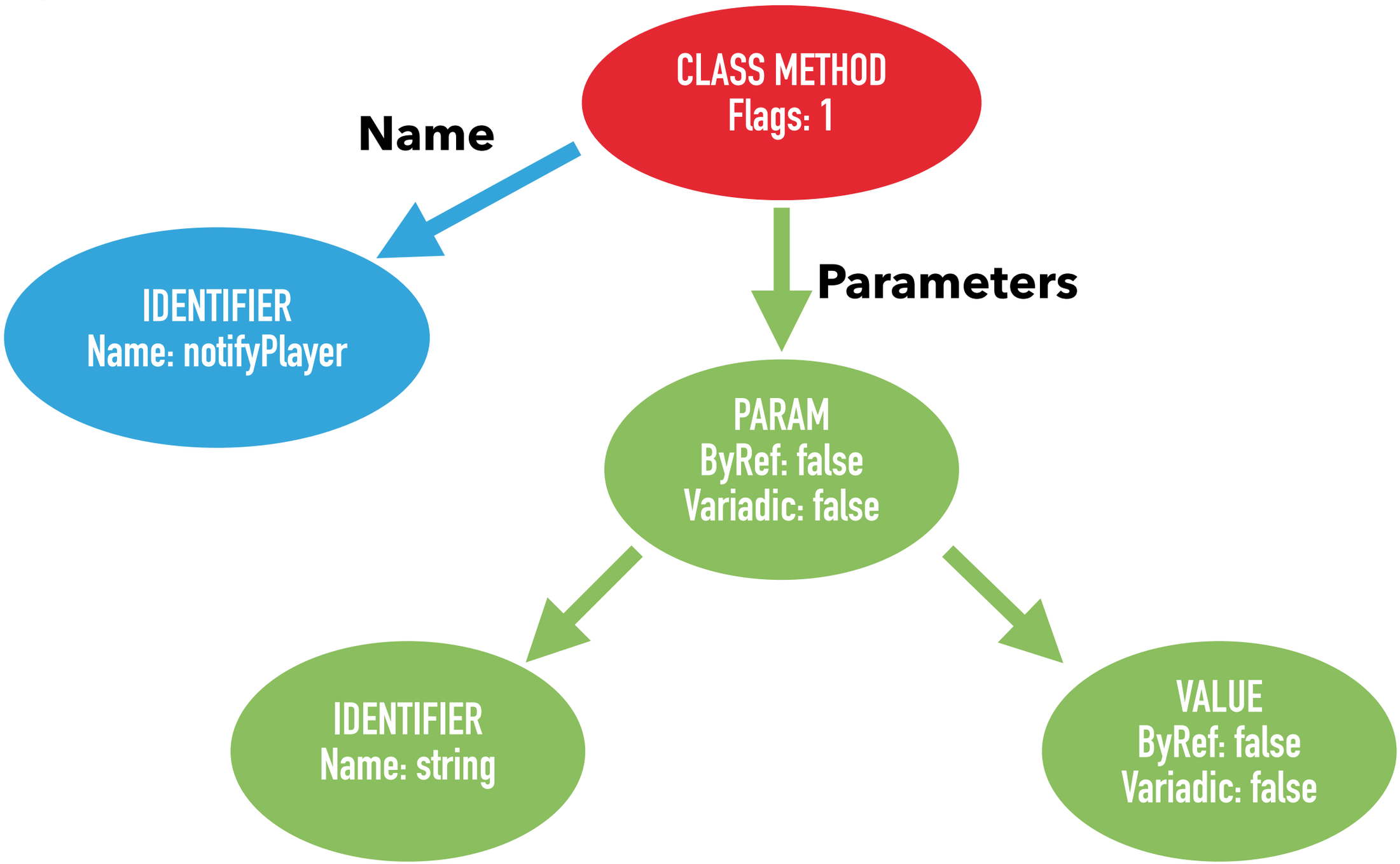
```
public function notifyPlayer (string $msg)
{
   $this->sender->sendMessage($msg);
}
```

CLASS METHOD
Flags: 1

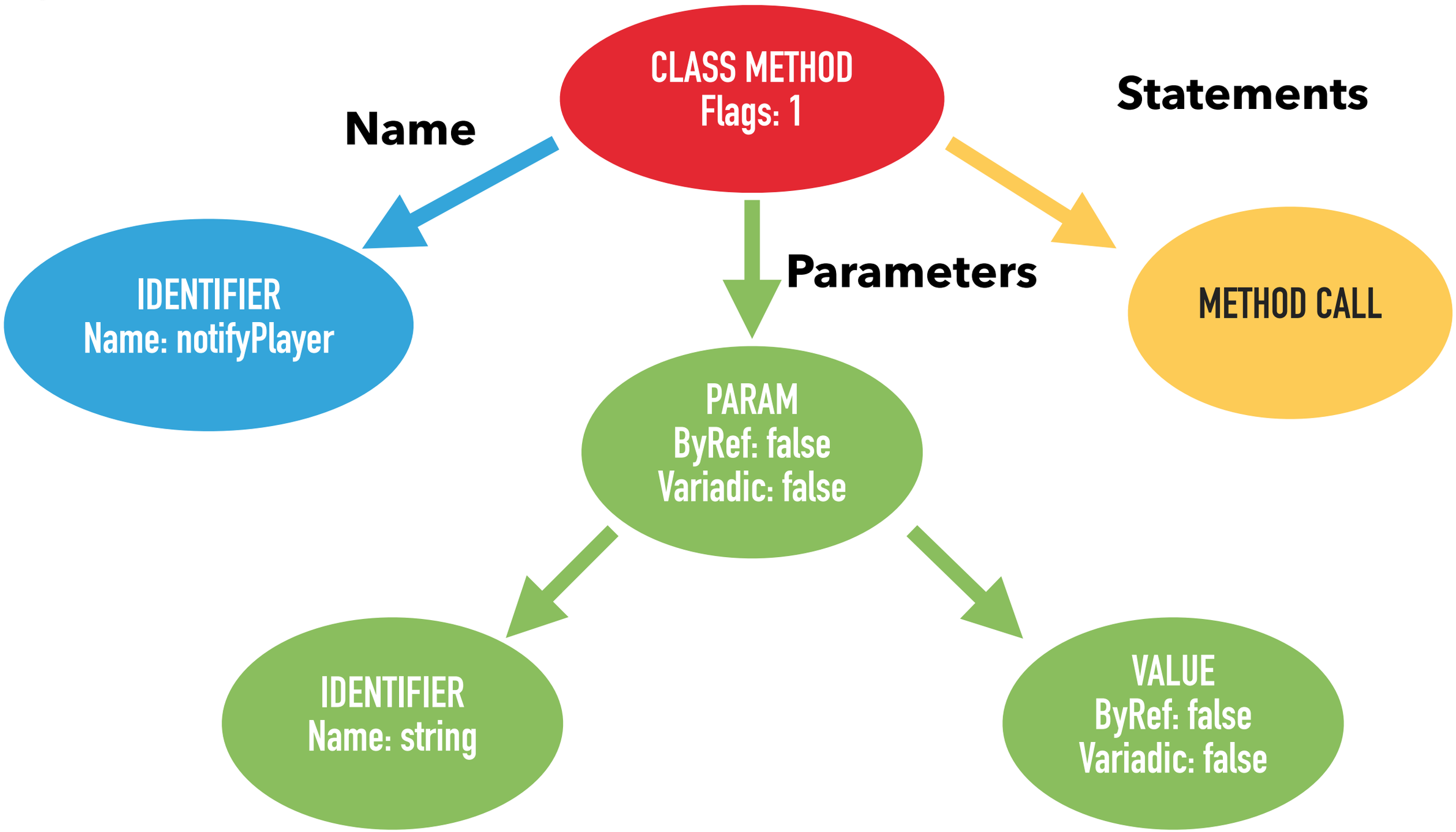
```
public function notifyPlayer (string $msg)
{
   $this->sender->sendMessage($msg);
}
```



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



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



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

nikic / PHP-Parser Public

Watch 232 | Fork 891

Code | Issues 44 | Pull requests 9 | Actions | Wiki | Security | Insights

master | 9 branches | 80 tags | Go to file | Add file | Code

nikic Bail out on PHP tags in removed code ... b0edd4c 2 hours ago 1,526 commits

.github/workflows	Test PHP 8.2 in CI	3 days ago
bin	Add --version flag to php-parse	17 days ago
doc	Fix pretty printing example	17 days ago
grammar	Support readonly before DNF type	3 days ago
lib/PhpParser	Bail out on PHP tags in removed code	2 hours ago
test	Bail out on PHP tags in removed code	2 hours ago
test_old	Avoid repeatedly downloading archive in run-php-src.sh	2 months ago
tools	Add tools/ directory	10 days ago
.editorconfig	[PHP 8.1] Add support for enums (#758)	17 months ago
.gitattributes	Add CONTRIBUTING.md	23 days ago
.gitignore	gitignore: add phpunit test cache	3 years ago
.php-cs-fixer.dist.php	Also format the grammar directory	23 days ago
CHANGELOG.md	Release PHP-Parser 5.0.0-alpha1	17 days ago
CONTRIBUTING.md	Add CONTRIBUTING.md	23 days ago
LICENSE	Corrected license text	2 years ago
README.md	Partial documentation update	17 days ago
UPGRADE-1.0.md	Fix typos	8 years ago

About

A PHP parser written in PHP

php | parser | static-analysis | ast

Readme

BSD-3-Clause license

15.7k stars

232 watching

891 forks

Releases 76

PHP-Parser 4.15.1 Latest

18 days ago

+ 75 releases

Packages

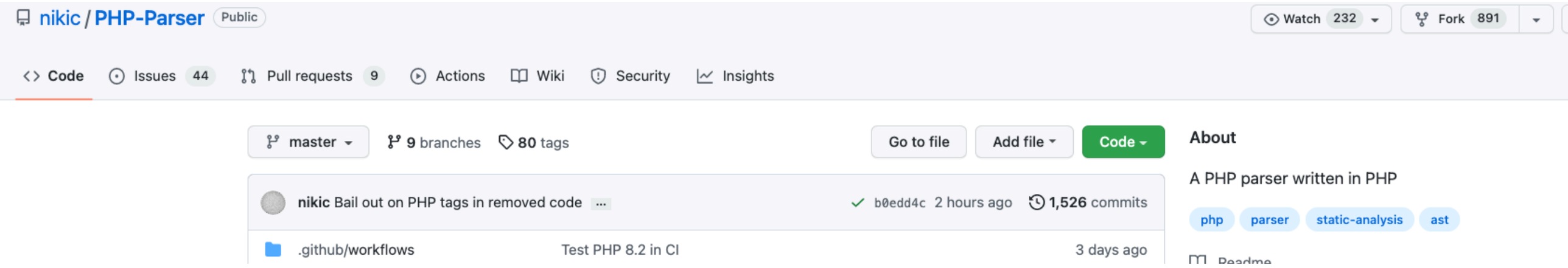
No packages published

Used by 1.5m

+ 1,486,143

Contributors 123

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



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

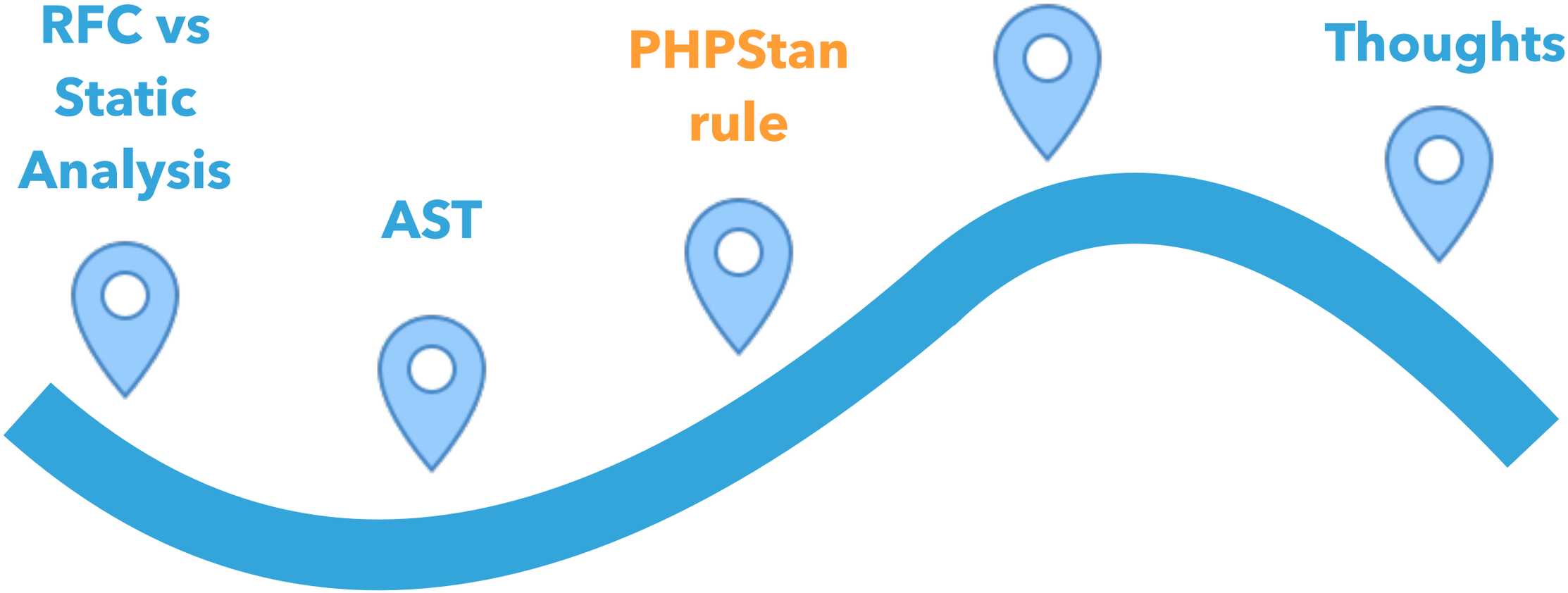
RFC vs Static Analysis

PHPStan rule

AST

PHP Extension Library

Thoughts



@daveliddament

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

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

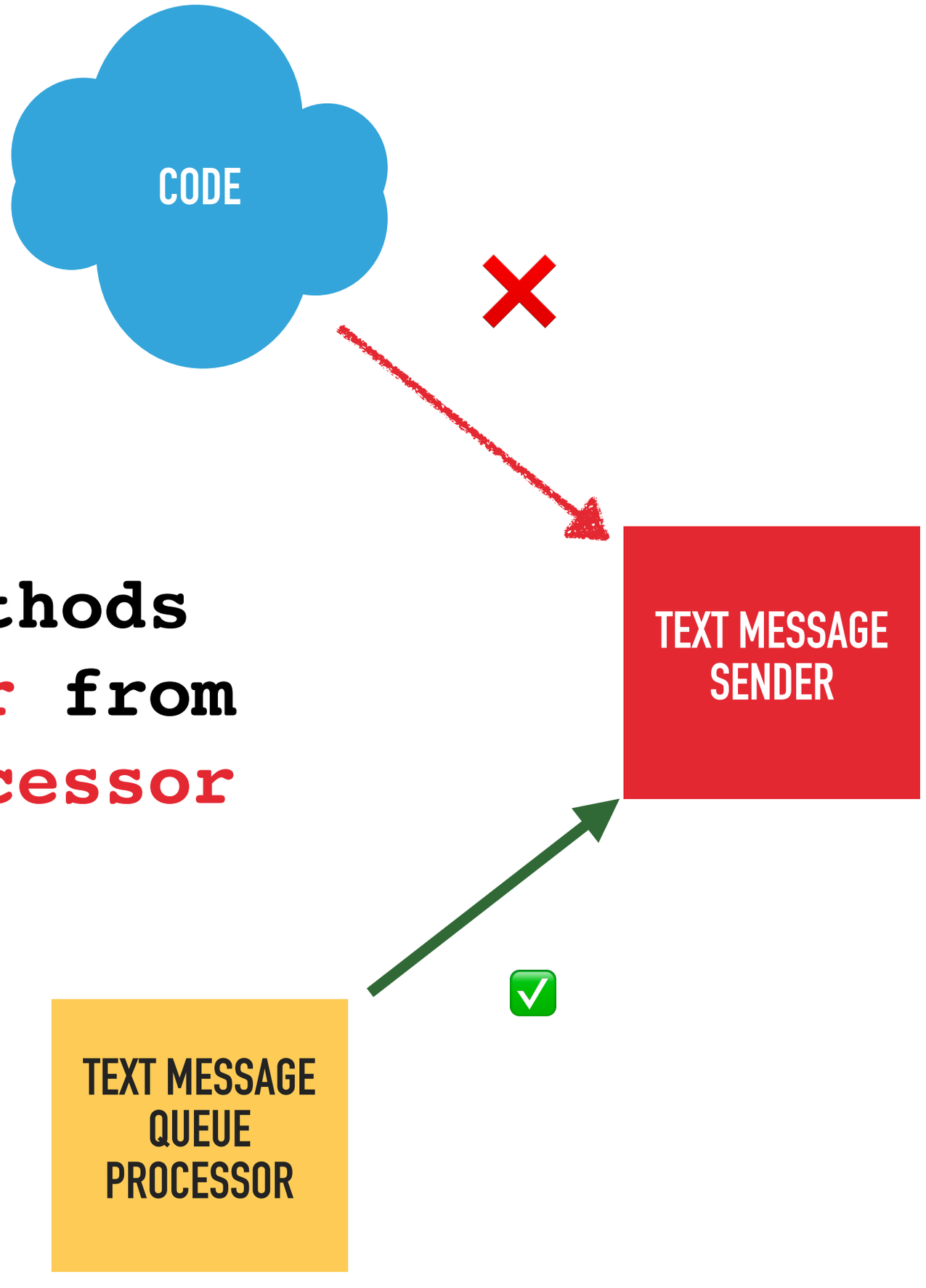
METHOD CALL

IDENTIFIER
Name: string

VALUE
ByRef: false
Variadic: false

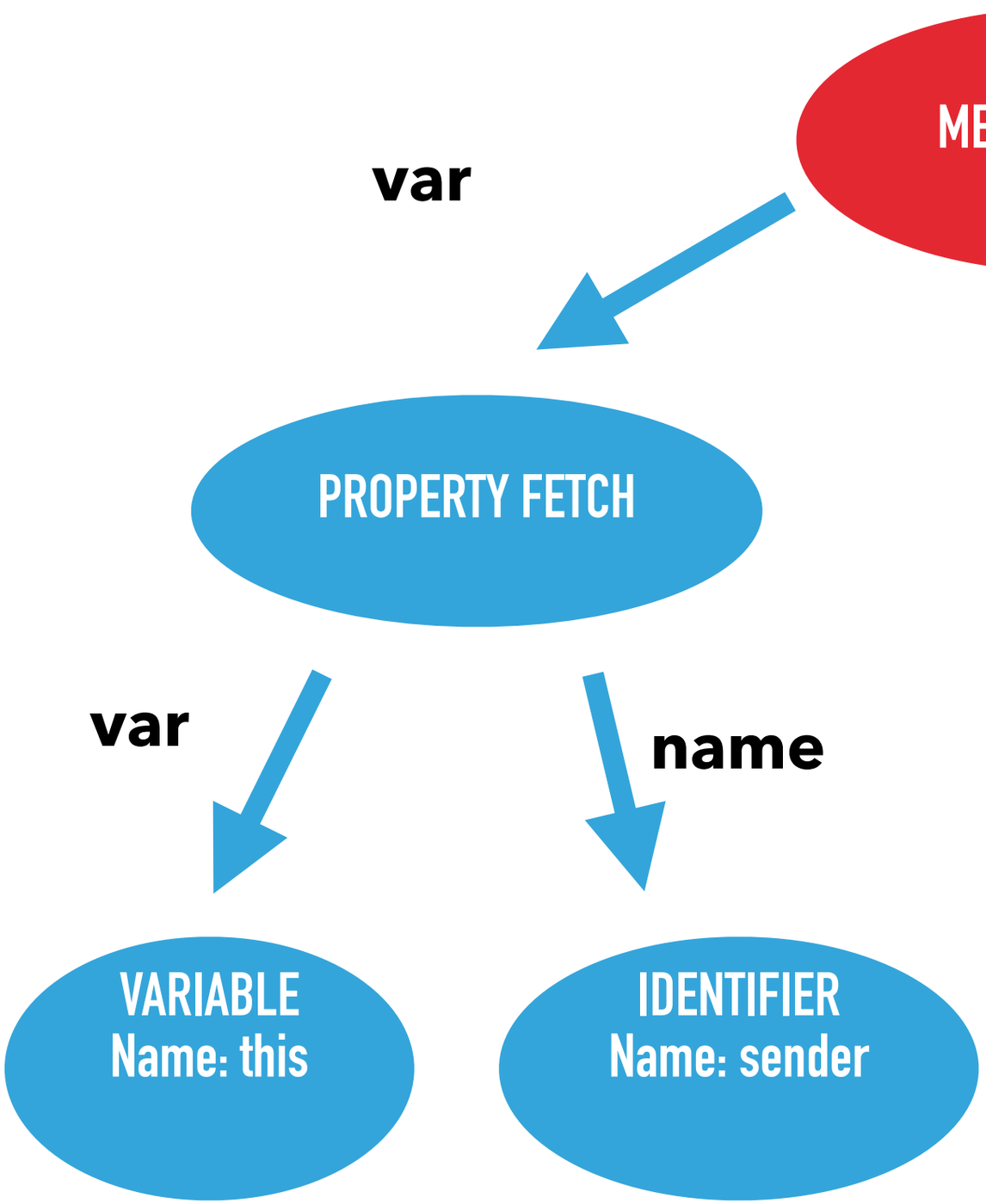
```
interface Rule
{

  public function getNodeType() : 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

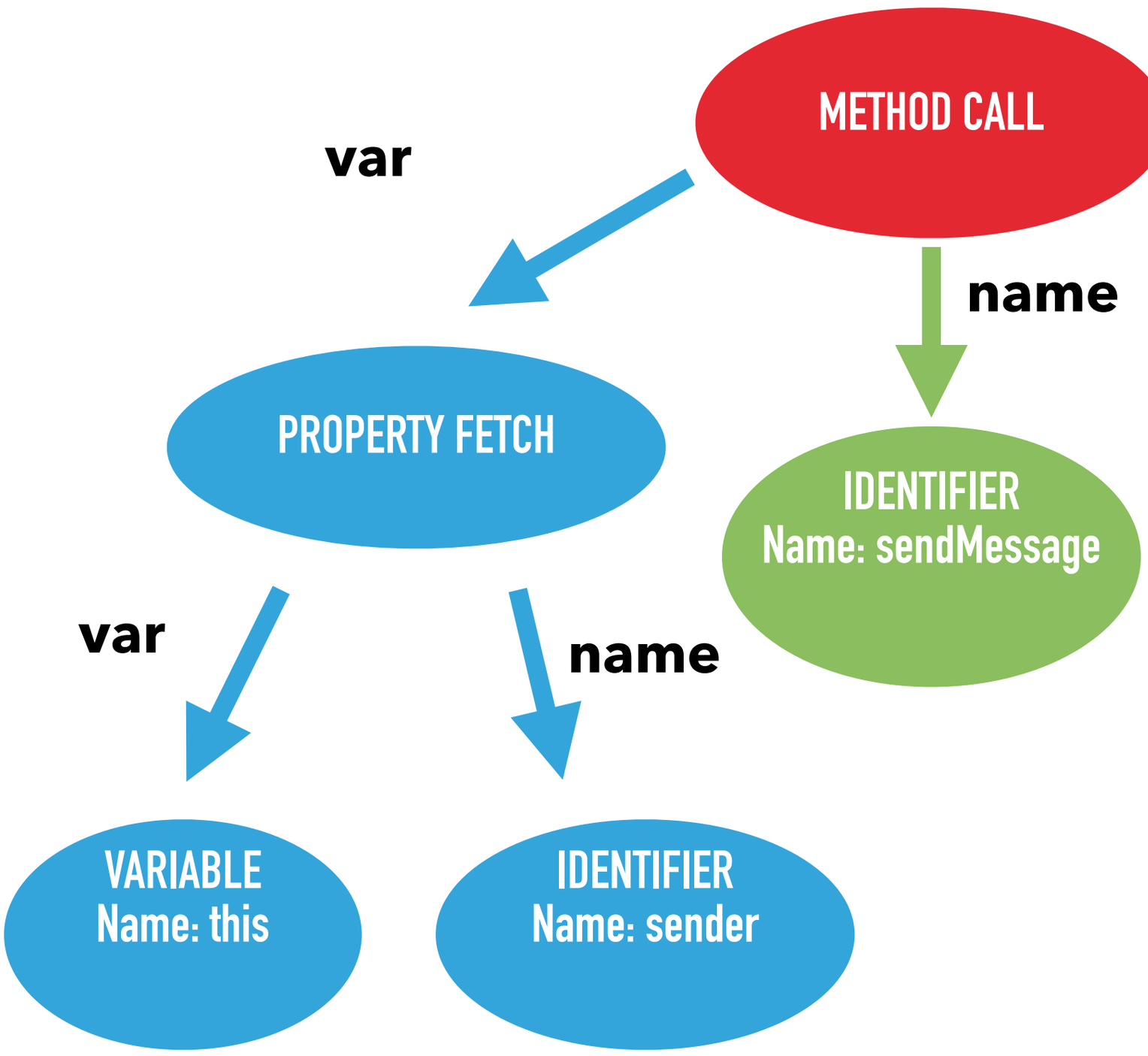


```
$this->sender -> sendMessage ($msg) ;
```

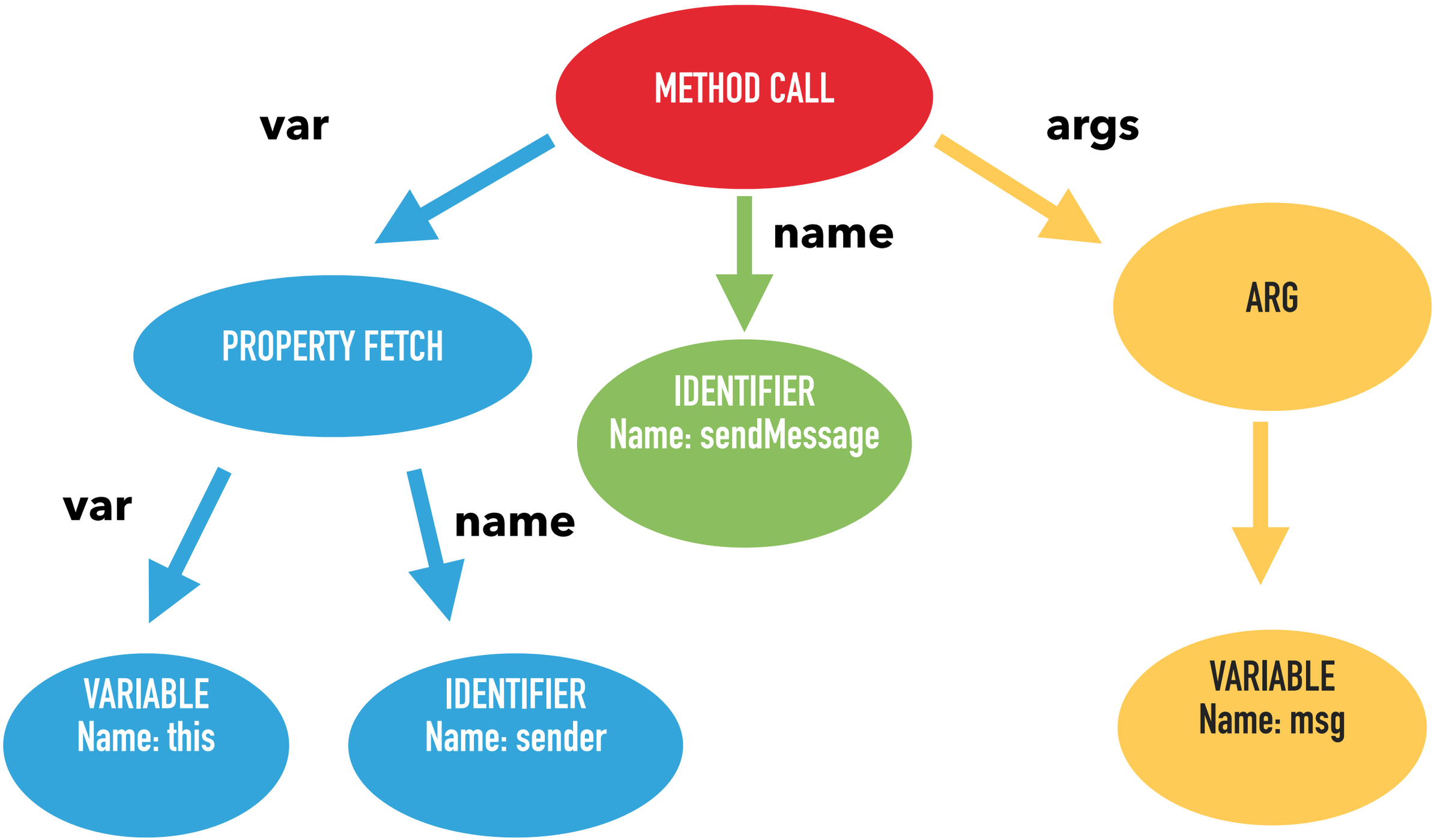
```
$this->sender -> sendMessage ($msg) ;
```



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



```
$this->sender->sendMessage($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 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 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 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 getNodeType() : string
    {
        return MethodCall::class;
    }
```

```
class TextMessageSenderCallerRule
    implements Rule
{

    public function getNodeType() : string
    {
        return MethodCall::class;
    }
```

```
class TextMessageSenderCallerRule
   implements Rule
{

    public function getNodeType() : 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`**

```
class PersonNotifier
{
 private TextMessageSender $sender;

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

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

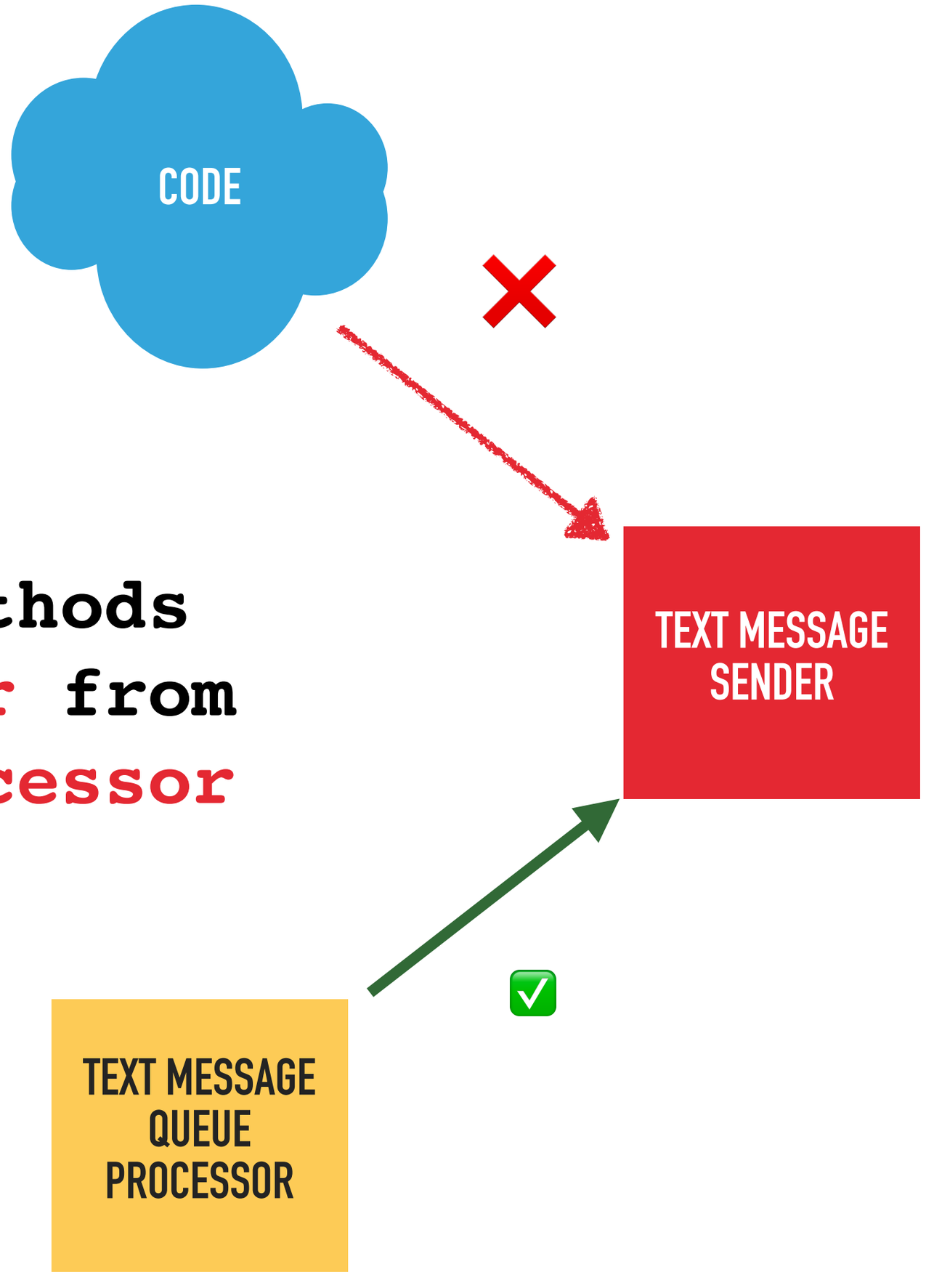
```
class PersonNotifier
{
    private TextMessageSender $sender;

    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 [];
}
```

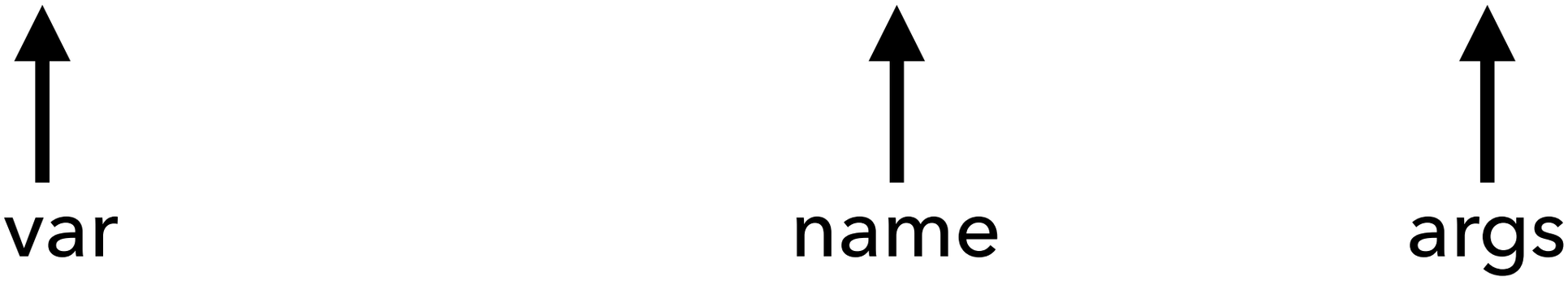
We can only call methods in TextMessageSender from TextMessageQueueProcessor



```
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 [];
}
```

```php
$this->sender->notfiyPlayer($msg);
```



```
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);
    }
}
```

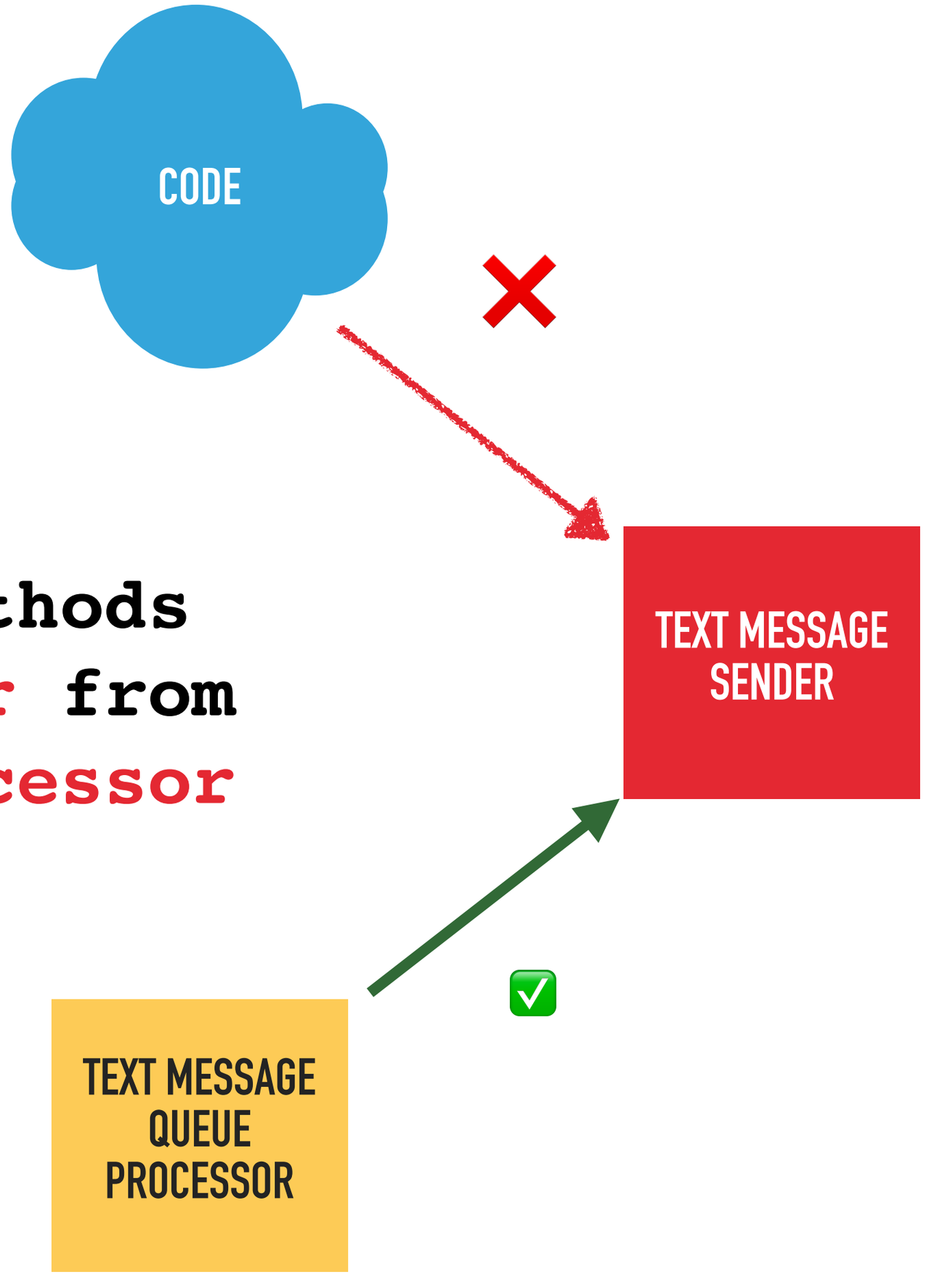
`$this->sender` is of type `TextMessageSender`

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 [];
}
```

```php
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 [];
}
```

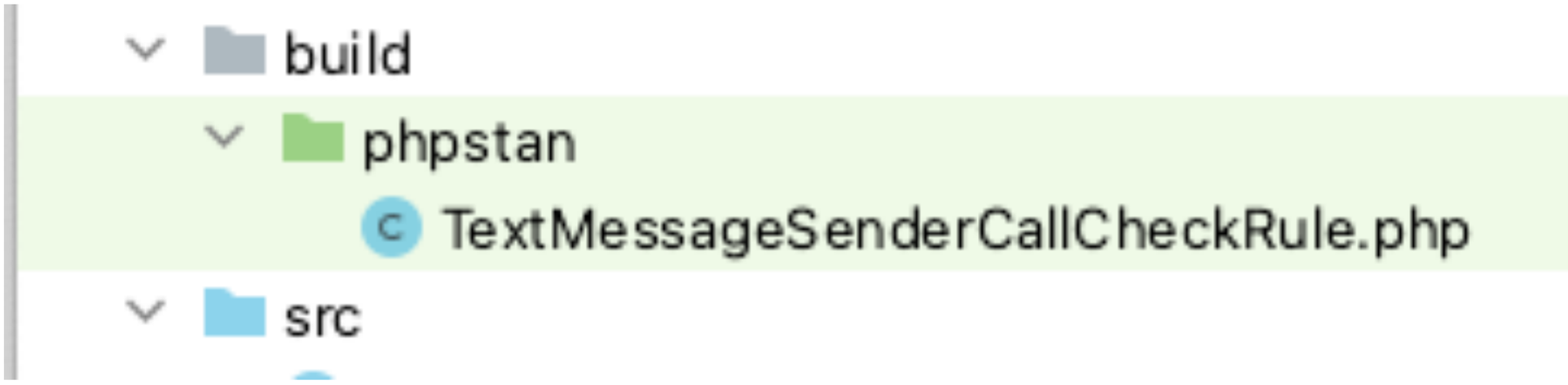
We can only call methods in TextMessageSender from TextMessageQueueProcessor

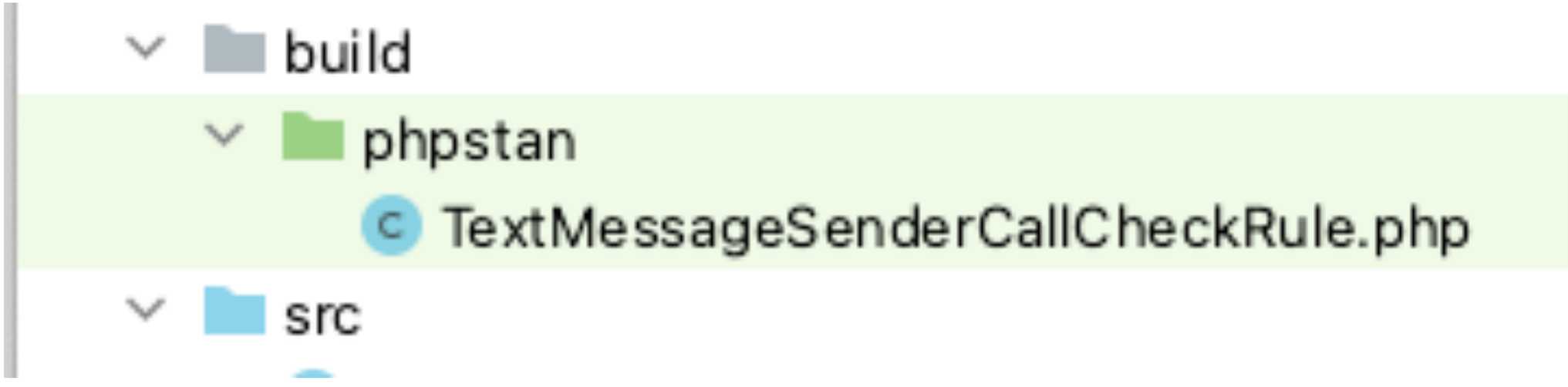


```php
public function processNode(Node $node, Scope $scope): array
{

  $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 [];
}
```

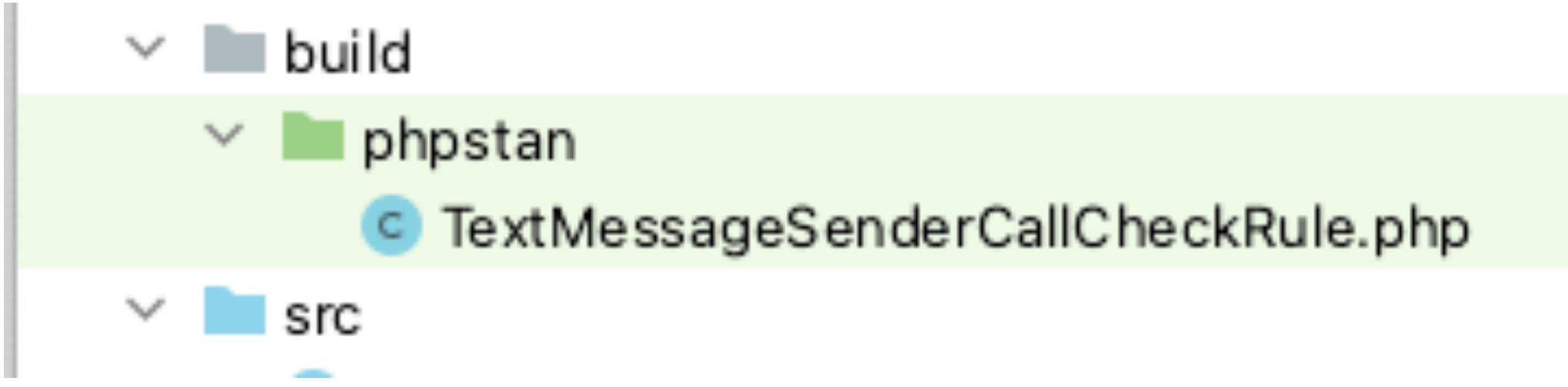
```
public function processNode(Node $node, Scope $scope): array
{

  $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 [];
}
```

```php
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 [];
}
```


build
phpstan
TextMessageSenderCallCheckRule.php
src



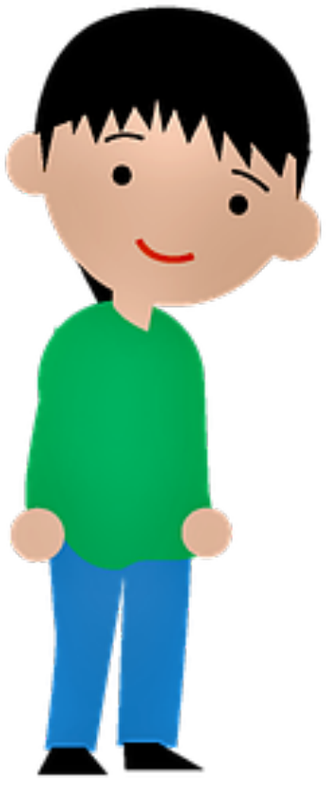

```
"autoload-dev": {
    "psr-4": {
        "DaveLiddament\\PhpstanRules\\": "build/phpstan/"
    }
},
```




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

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

AMAZING!



@daveliddament

AMAZING!

WE CAN DO BETTER...

@daveliddament

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,
    ) {}
```

```php
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 [];
}
```

```php
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 [];
}
```

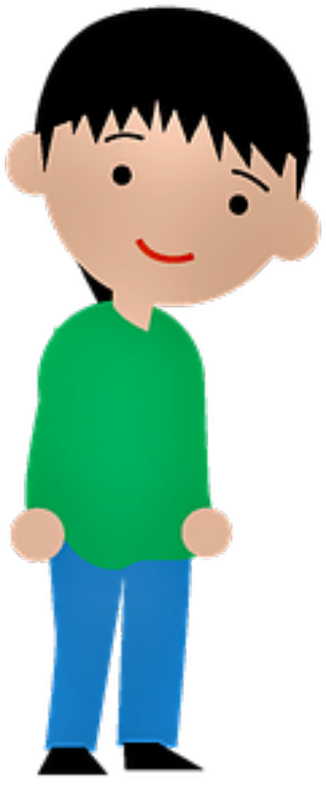
```php
public function processNode(Node $node, Scope $scope): array
{

  $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
```

```
services:
-
  class: PhpstanRules\TextMessageSenderCallCheckRule
  tags:
  - phpstan.rules.rule
  arguments:
    allowedCallingClass: RuleDemo\TextMessageQueueProcessor
    targetClass: RuleDemo\TextMessageSender
```

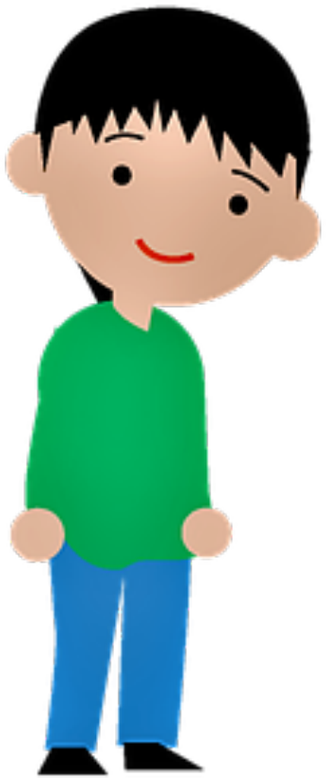
```
services:
-
  class: PhpstanRules\TextMessageSenderCallCheckRule
  tags:
  - phpstan.rules.rule
  arguments:
    allowedCallingClass: RuleDemo\TextMessageQueueProcessor
    targetClass: RuleDemo\TextMessageSender

 class: PhpstanRules\TextMessageSenderCallCheckRule
tags:
- phpstan.rules.rule
arguments:
    allowedCallingClass: Foo
    targetClass: Bar
```

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



@daveliddament

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



WE CAN DO BETTER...

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

}
```

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

}
```

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

}
```

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

```php
#[Attribute(Attribute::TARGET_CLASS)]
class Friend
{

  /** @param class-string $friend */
  public function __construct(
    public string $friend,
  ) {}
}
```

```php
#[Attribute(Attribute::TARGET_CLASS)]
class Friend
{

  /** @param class-string $friend */
  public function __construct(
    public string $friend,
  ) {}
}


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

}
```

```php
#[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 [];
}
```

```php
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 [];
}
```

```php
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) {

 // 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) {

 // 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) {

 // 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) {

 // 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];
```

```php
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()];

  }

}
```

```
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()];

  }

}
```

```php
#[Attribute(Attribute::TARGET_CLASS)]
class Friend
{

  /** @param class-string $friend */
  public function __construct(
    public string $friend,
  ) {}
}
```

```php
class TextMessageSenderCallCheckRule implements Rule
{
  public function __construct(
    private ReflectionProvider $reflectionProvider,
  ) {}

  public function getNodeType(): string
  {
    return MethodCall::class;
  }

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

    foreach ($type->getReferencedClasses() as $targetClass) {
      $nativeReflection = $this->reflectionProvider->getClass($targetClass)->getNativeReflection();
      $friendAttributes = $nativeReflection->getAttributes(Friend::class);
      if (count($friendAttributes) !== 1) {
        continue;
      }

      $friendAttribute = $friendAttributes[0];
      $friendArguments = $friendAttribute->getArguments();
      if (count($friendArguments) !== 1) {
        continue;
      }

      $friend = $friendArguments[0];
      if ($callingClass !== $friend) {
        $msg = sprintf("Can not call %s from %s”, $targetClass, $callingClass);
        return [RuleErrorBuilder::message($msg)->build()];
      }
    }

    return [];
  }
}
```

```php
#[Attribute(Attribute::TARGET_CLASS)]
class Friend
{

  /** @param class-string $friend */
  public function __construct(
    public string $friend,
  ) {}
}
```

```php
class TextMessageSenderCallCheckRule implements Rule
{
  public function __construct(
    private ReflectionProvider $reflectionProvider,
  ) {}

  public function getNodeType(): string
  {
    return MethodCall::class;
  }

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

    foreach ($type->getReferencedClasses() as $targetClass) {
      $nativeReflection = $this->reflectionProvider->getClass($targetClass)->getNativeReflection();
      $friendAttributes = $nativeReflection->getAttributes(Friend::class);
      if (count($friendAttributes) !== 1) {
        continue;
      }

      $friendAttribute = $friendAttributes[0];
      $friendArguments = $friendAttribute->getArguments();
      if (count($friendArguments) !== 1) {
        continue;
      }

      $friend = $friendArguments[0];
      if ($callingClass !== $friend) {
        $msg = sprintf("Can not call %s from %s”, $targetClass, $callingClass);
        return [RuleErrorBuilder::message($msg)->build()];
      }
    }
  }
}
```

```php
#[Attribute(Attribute::TARGET_CLASS)]
class Friend
{

  /** @param class-string $friend */
  public function __construct(
    public string $friend,
  ) {}
}
```

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

```
class TextMessageSenderCallCheckRule implements Rule
{
  public function __construct(
    private ReflectionProvi
  ) {}

  public function get
  {
    return MethodCall::c
  }

  public function processNode(Node $node, Scope $scope): array
  {
    $c     ass = $scope->getClassReflection()->getName();
    $      ope->getType($node->var);

    i      ype->getReferencedClas
           lection = $this
    $      Attribut
           ount($friendA
           tinue;


           dAttribute = $frie
           ndArguments = $friendAtt
           ount($friendArguments) !== 1) {
           tinue;


    $friend = $friendArguments[0];
    if ($callingClass !== $friend) {
      $msg = sprintf("Can not call %s from %s", $targetClass, $callingClass);
      return [RuleErrorBuilder::message($msg)->build()];
    }
  }
}
```

```
#[Attribute(Attribute::TARGET_CLASS)]
    Friend

    ss-string $friend */
    n __construct(
    ing $friend,

    )->getNativeR
```

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

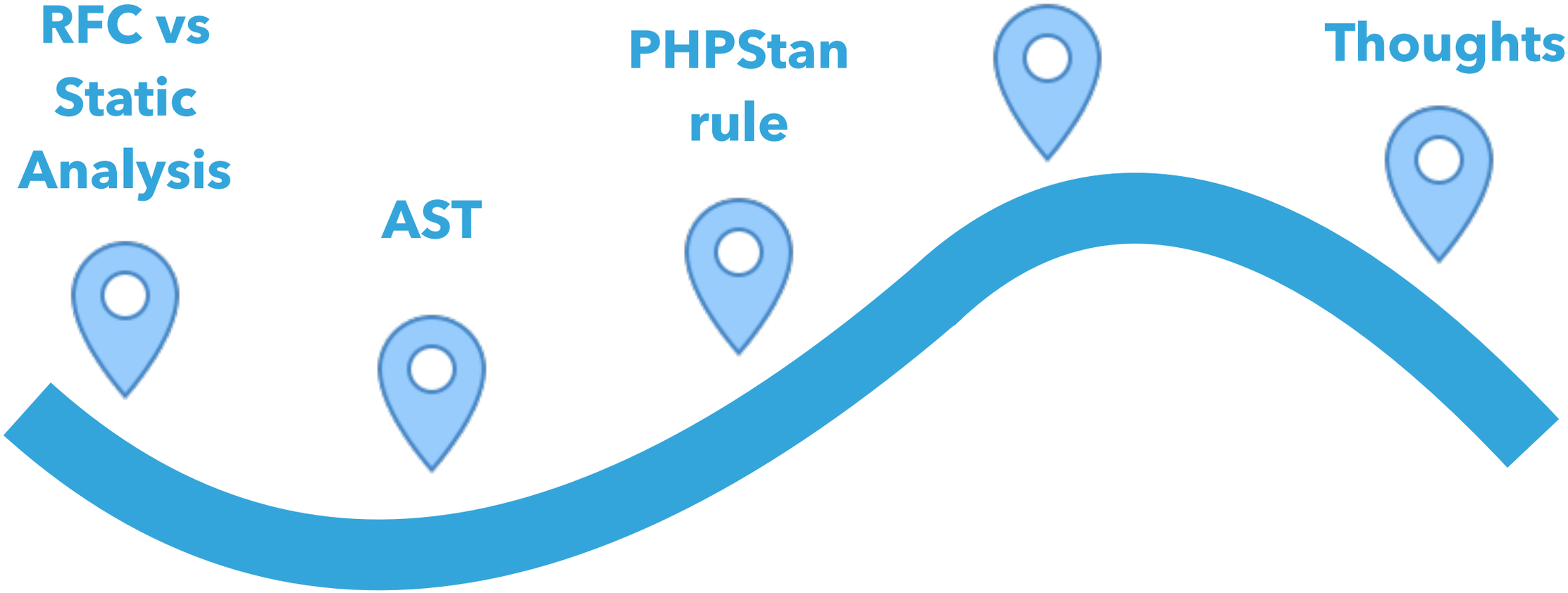
RFC vs Static Analysis

PHPStan rule

AST

PHP Extension Library

Thoughts



@daveliddament

DaveLiddament / **php-language-extensions** Public

Unpin | Unwatch 2 | Fork 1 | Star 16

Code | Issues 2 | Pull requests | Discussions | Actions | Security | Insights | Settings

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

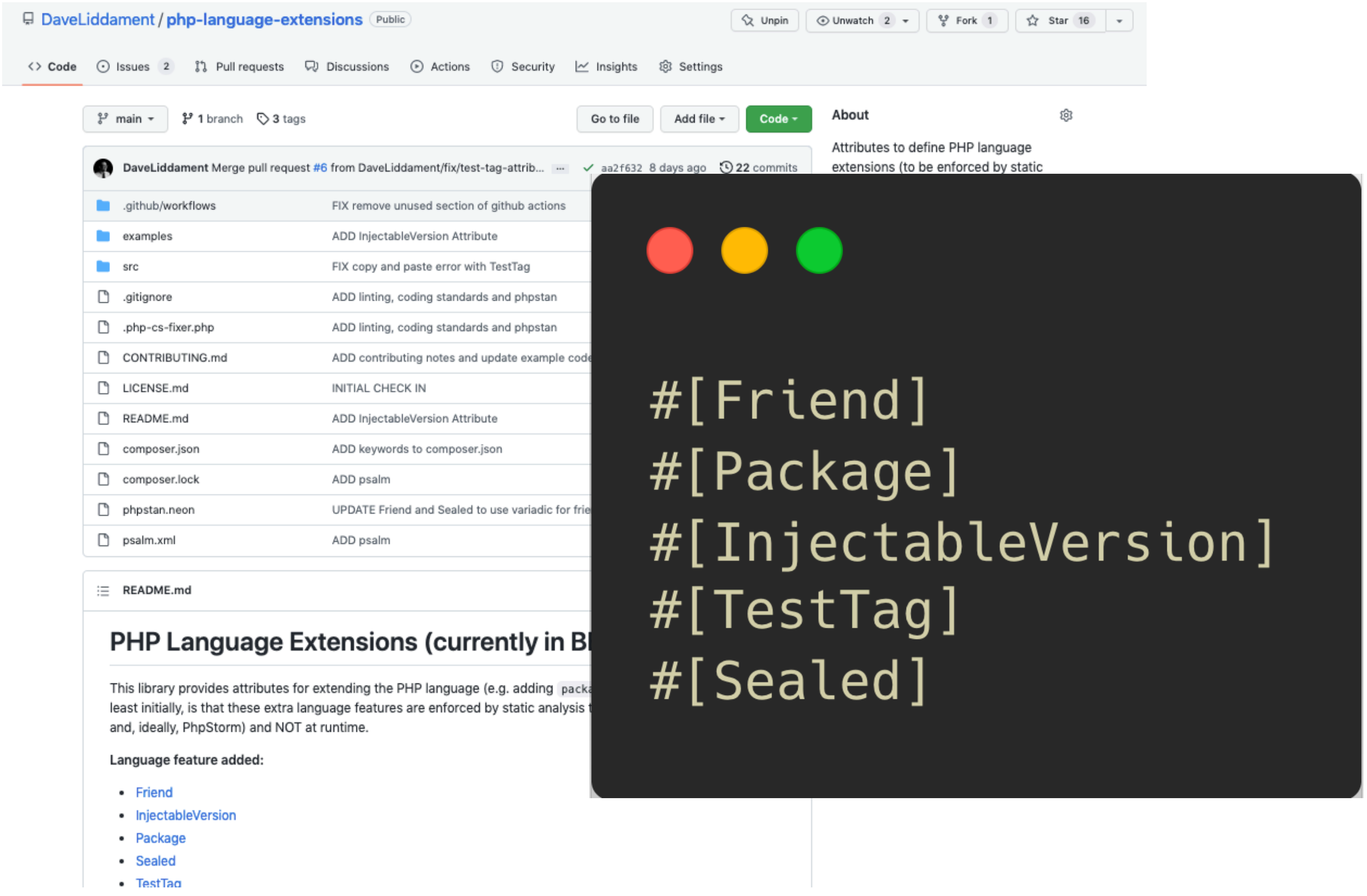
Contributors 2

DaveLiddament Dave Liddament
ruudk Ruud Kamphuis

Languages

PHP 100.0%

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



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

Definition

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

Definition

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

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

Definition

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

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

#[Friend]

Only allowed to call method from specified classes

@daveliddament

```php
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

@daveliddament

```php
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

```php
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

@daveliddament

```php
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

```php
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

@daveliddament

```php
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

@daveliddament

```php
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

```php
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]

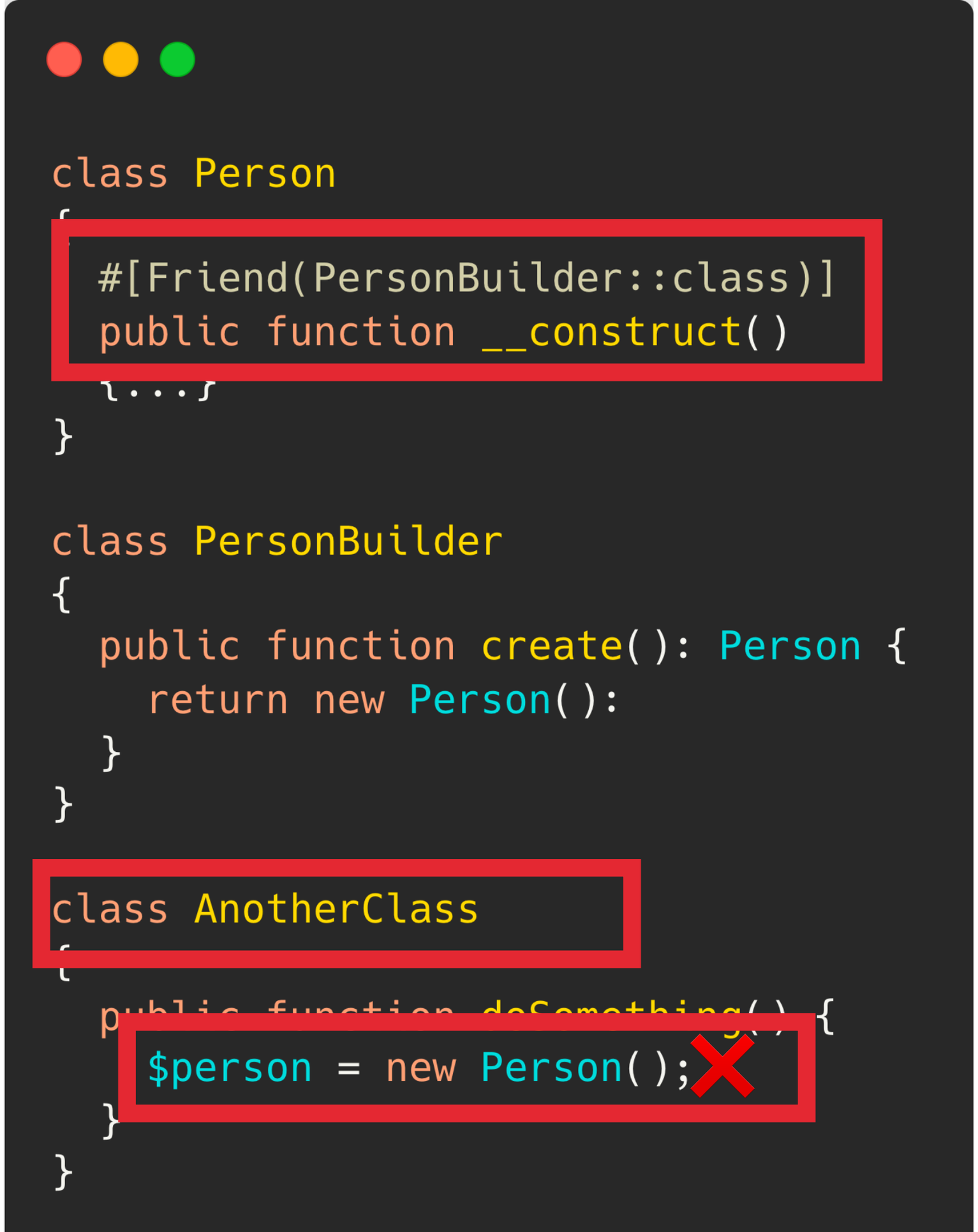
Only allowed to call method from specified classes

@daveliddament

```php
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



#[Package]

Only allowed to call methods from the same namespace

```
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;
  }
}
```

#[Package]

Only allowed to call methods from the same namespace

```
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;
  }
}
```

#[Package]

Only allowed to call methods from the same namespace

```php
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;
  }
}
```

#[Package]

Only allowed to call methods from the same namespace

```
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;
  }
}
```

#[Package]

Only allowed to call methods from the same namespace

```php
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;
  }
}
```

```php
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;
  }
}
```

```php
namespace SomeNamespace;

class SomeClass {
  public function SomeMethod() {
    // Some code
    DiscountCaluclator::getDiscount($items);
  }
}
```

@daveliddament

```php
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;
  }
}
```

```php
namespace SomeNamespace;

class SomeClass {
  public function SomeMethod() {
    // Some code
    DiscountCaluclator::getDiscount($items);
  }
}
```

```php
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;
  }
}
```

```php
namespace SomeNamespace;

class SomeClass {
  public function SomeMethod() {
    // Some code
    DiscountCaluclator::getDiscount($items);
  }
}
```

@daveliddament

```php
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;
  }
}
```

```php
namespace SomeNamespace;

class SomeClass {
  public function SomeMethod() {
    // Some code
    DiscountCaluclator::getDiscount($items);
  }
}
```

@daveliddament

```php
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;
  }
}
```

```php
namespace SomeNamespace;

class SomeClass {
  public function SomeMethod() {
    // Some code
    DiscountCaluclator::getDiscount($items);
  }
}
```

@daveliddament

```php
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;
  }
}
```

```php
namespace SomeNamespace;

class SomeClass {
  public function SomeMethod() {
    // Some code
    DiscountCaluclator::getDiscount($items);
  }
}
```

@daveliddament

#[TestTag]

Only allowed to call method from test code

```php
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

```php
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

```php
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

```php
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

```php
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

```php
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

```php
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

```php
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

```php
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 maybe injected into constructors

@daveliddament

```php
#[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

@daveliddament

```php
#[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

@daveliddament

```php
#[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

@daveliddament

```php
#[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

@daveliddament

```php
#[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

@daveliddament

```php
#[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

@daveliddament

```php
#[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,
  ) {}
}
```

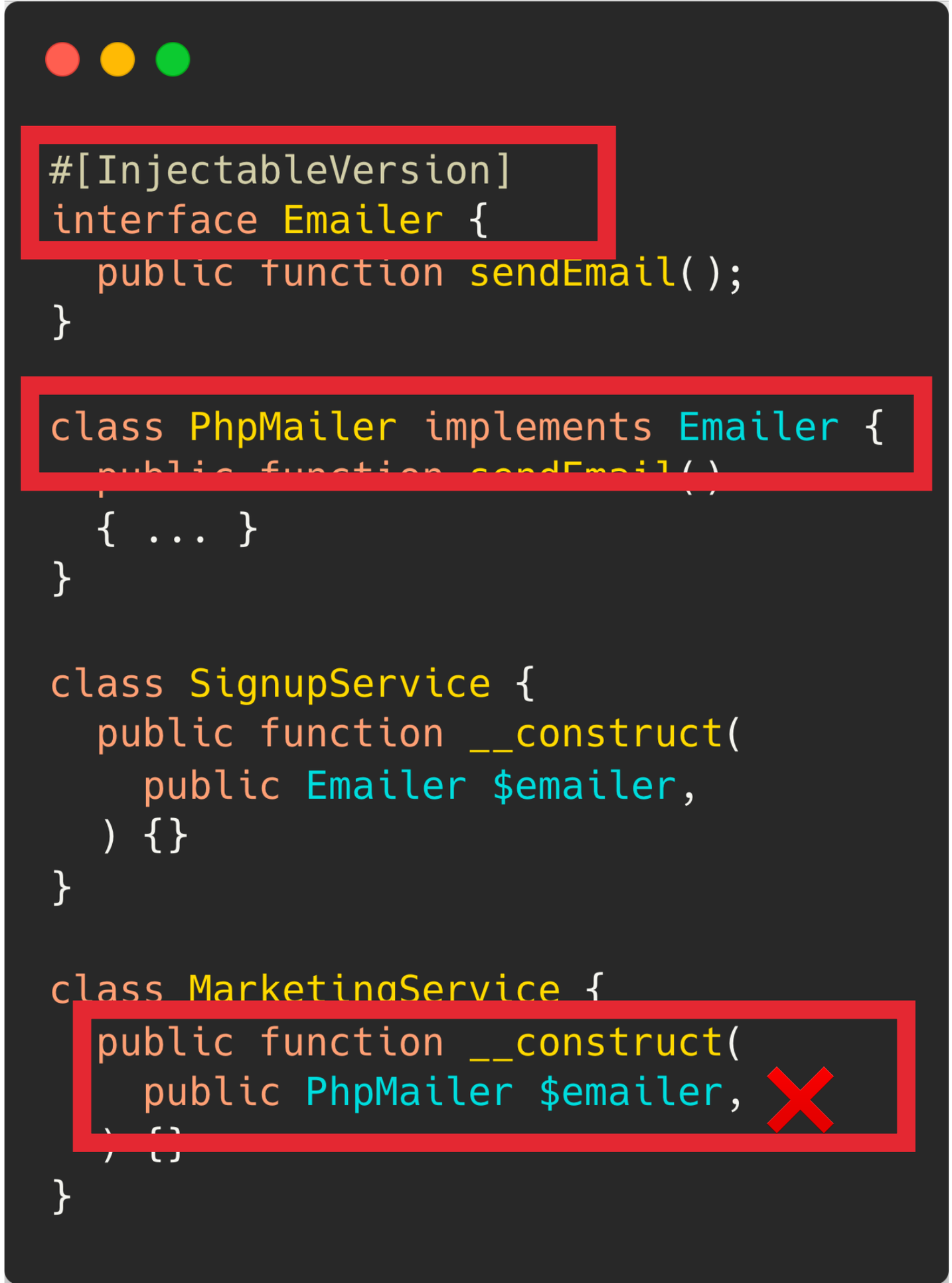
#[InjectableVersion]

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

```php
#[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



Attributes

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

PHPStan

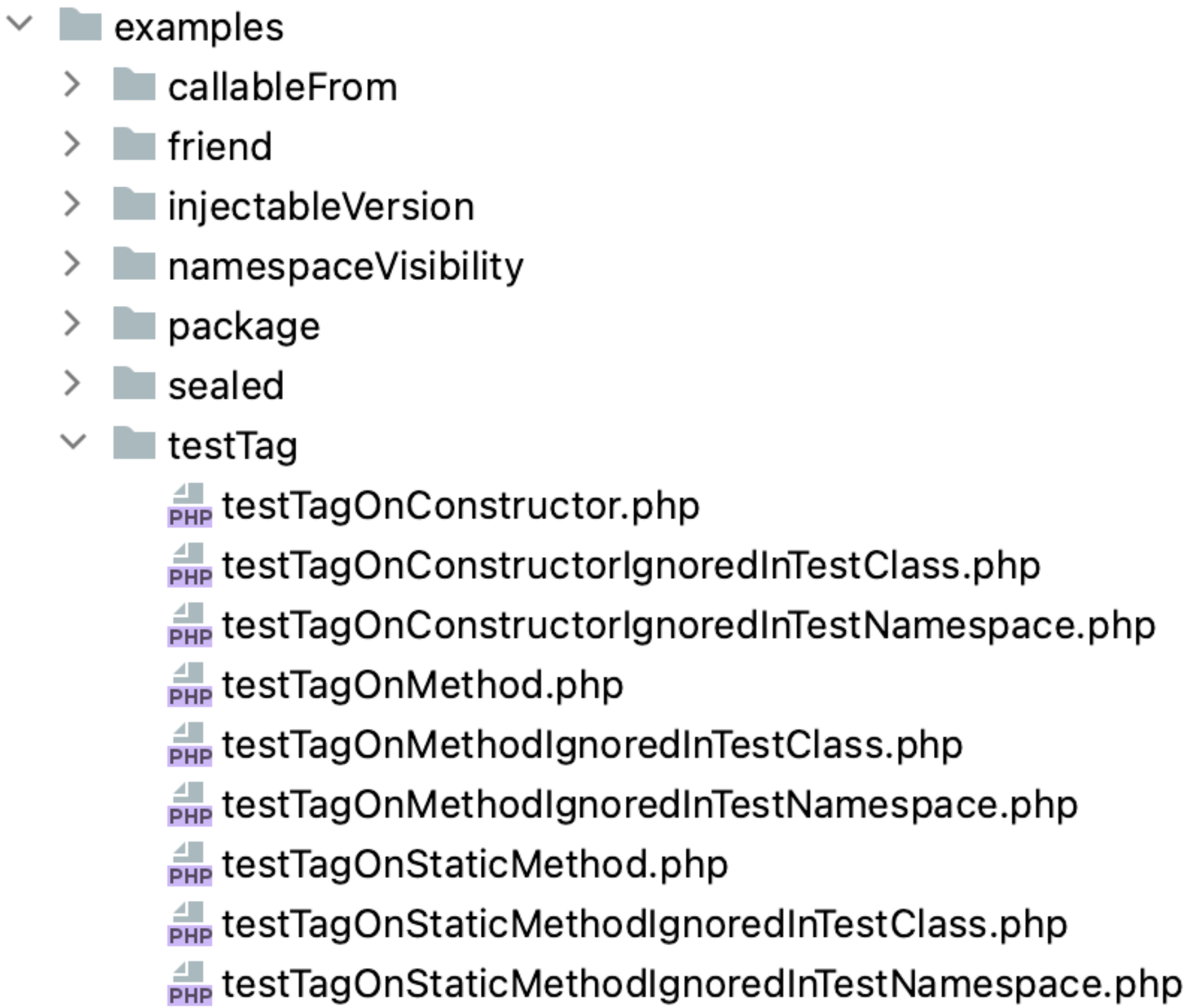
https://github.com/DaveLiddament/phpstan-php-language-extensions

Psalm and others

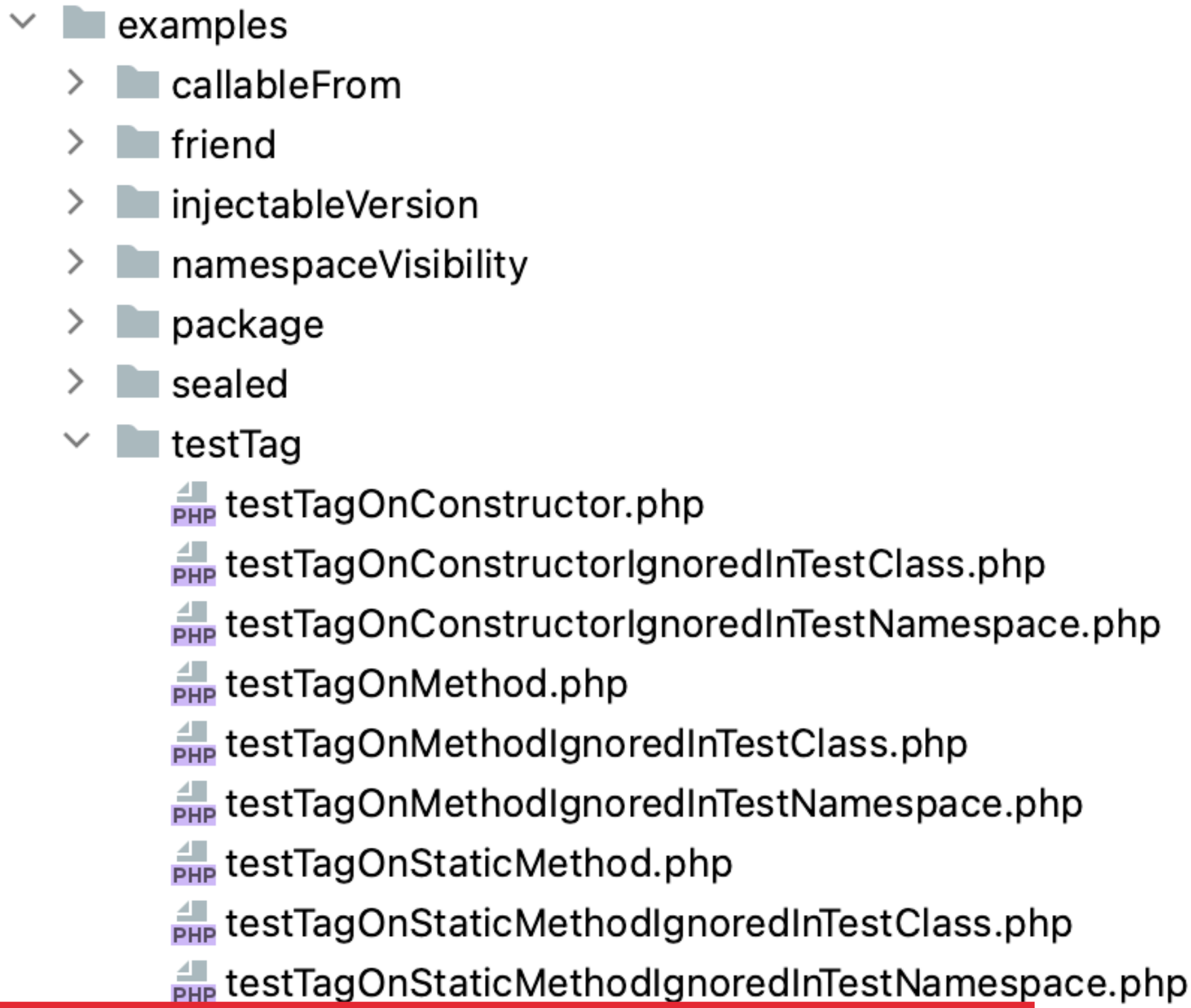
Contributions welcome

@daveliddament

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



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



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

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

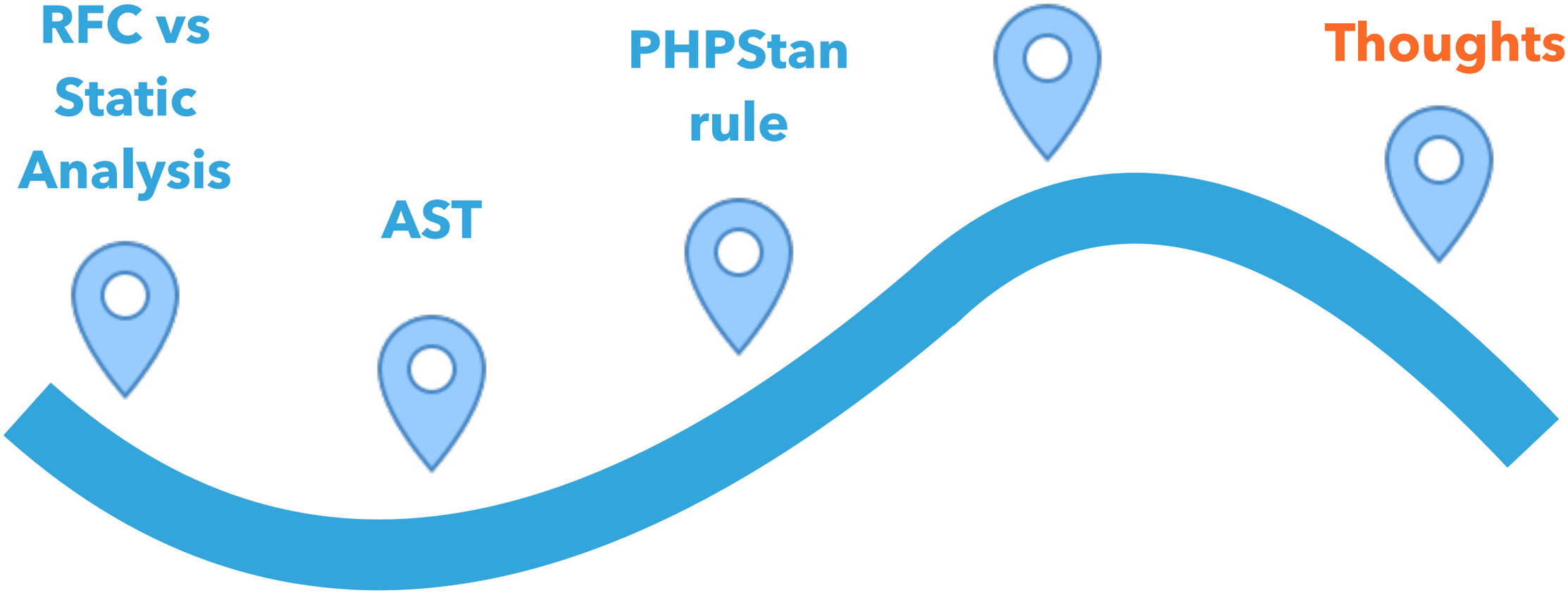
RFC vs Static Analysis

PHPStan rule

AST

PHP Extension Library

Thoughts



@daveliddament

SECURITY

SYNTACTIC SUGAR

SECURITY

SYNTACTIC SUGAR

SECURITY

PERFORMANCE

SYNTACTIC SUGAR

SECURITY

PERFORMANCE

NEW FEATURES

SYNTACTIC SUGAR

SECURITY

PERFORMANCE

NEW FEATURES

COMMUNICATING INTENTION

SYNTACTIC SUGAR

SECURITY

PERFORMANCE

NEW FEATURES

COMMUNICATING INTENTION

OVERLAP

SYNTACTIC SUGAR

SECURITY

PERFORMANCE

NEW FEATURES

COMMUNICATING INTENTION

public

protected

private

final

abstract

readonly

type declarations

type hints

public

protected

private

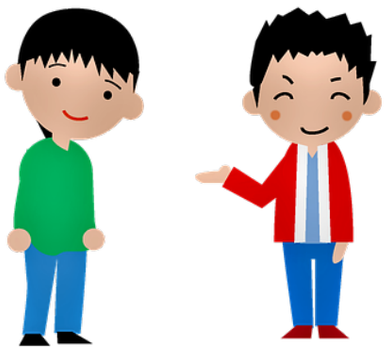
final

abstract

readonly

type declarations

type hints



public

protected

private

final

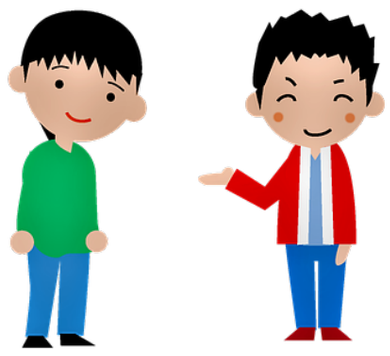
abstract

readonly

type declarations

Friend

type hints



Sealed

Package

InjectableVersion

public

protected

private

final

abstract

readonly

type declarations

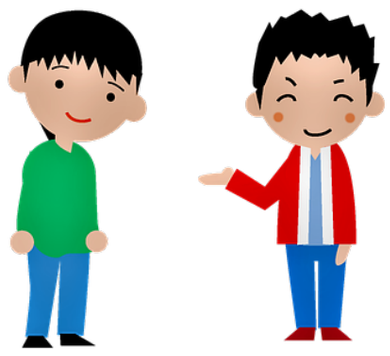
Friend

type hints

Sealed

Package

InjectableVersion



AUTOMATE CHECKS FOR VIOLATIONS

Try out ideas…

Today

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

}
```

Try out ideas...

Today

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

}
```

PHP 8.3+ ?

```
class TextMessageSender
    friend TextMessageQueueProcessor
{

}
```

Try out ideas...

Today

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

}
```

PHP 8.3+ ?

```
class TextMessageSender
    friend TextMessageQueueProcessor
{

}
```

never return type (added in PHP 8.1)

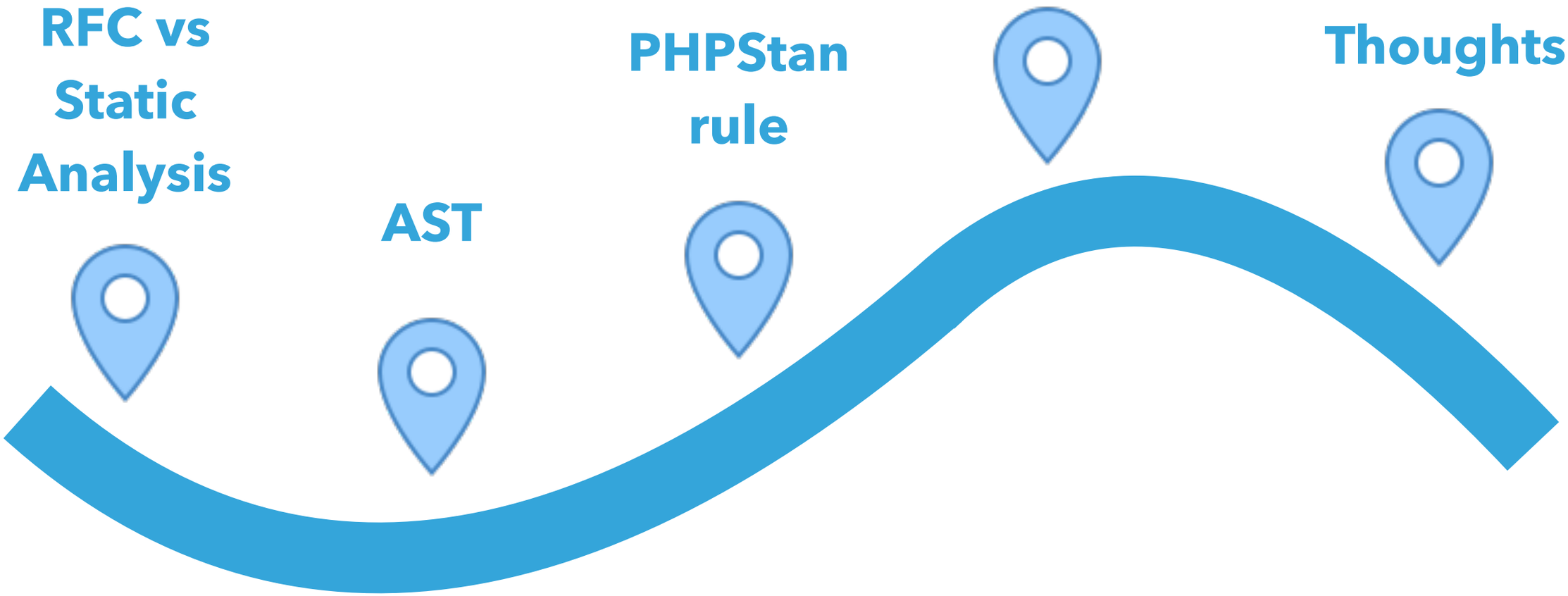
RFC vs Static Analysis

PHPStan rule

AST

PHP Extension Library

Thoughts



@daveliddament

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