Aspect Oriented Programming on PHP

Repensez votre PHP

Certains exemples de code ont étés maltraités durant la réalisation de cette présentation.

Who's who

https://github.com/AOP-PHP

Gérald Croës	Julien Salleyron
www.croes.org/gerald/	Tests
Documentation / Api / Tests	Code Monkey
@geraldcroes	@juguul
Object Oriented Geek *	Lead Developper
gerald@php.net	juliens@php.net

Les slides seront ici

In PHP ? Dude ?
Are you serious ?

You may follow us…
we don't talk at all

C’est parti !

Qu’est-ce que l’AOP ?

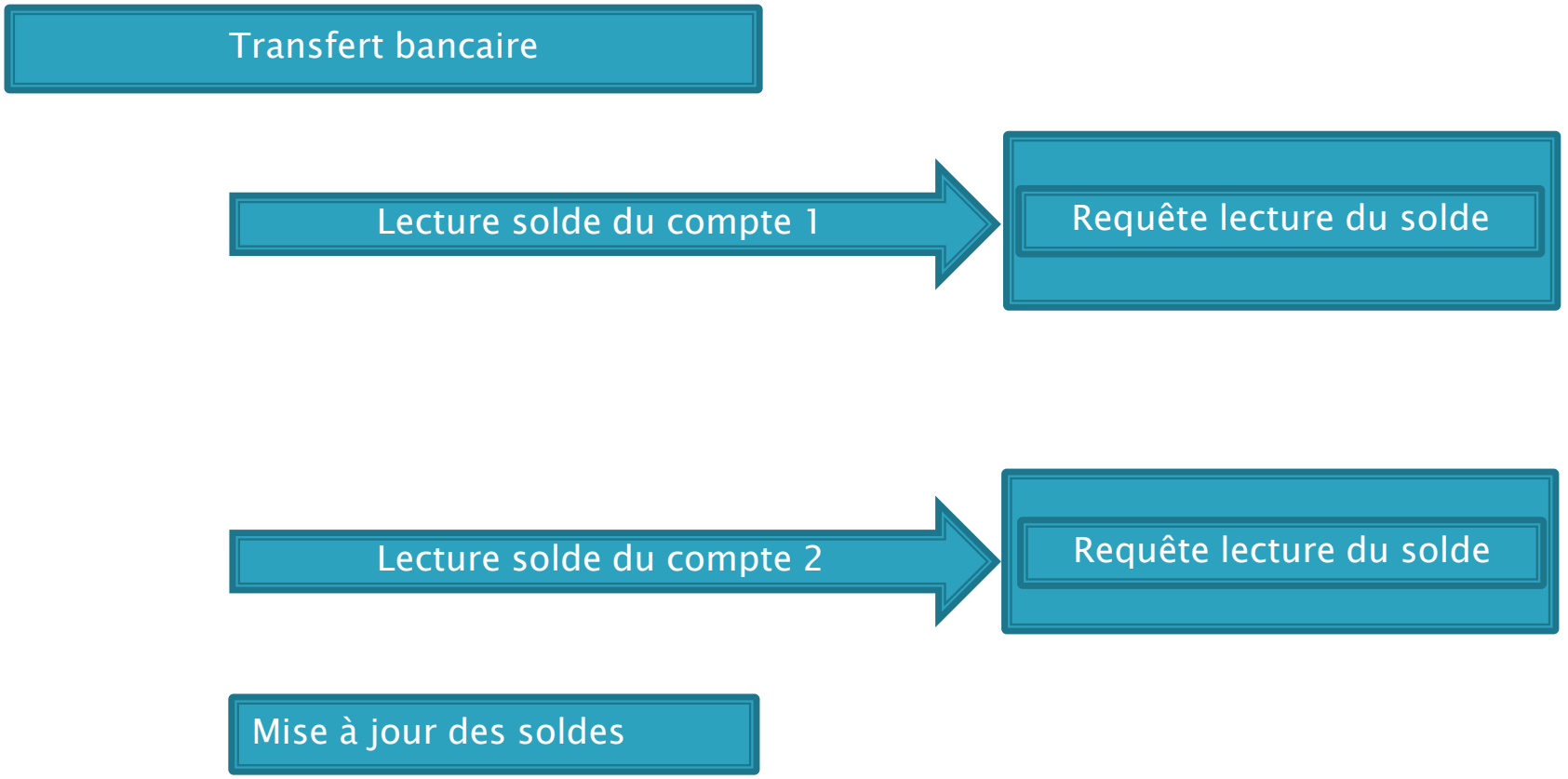
Les problématiques transverses

- Validation des données
- Persistance des données
- Gestion transactionnelle
- Internationalisation
- Sécurité
- Gestion du cache
- Gestion des logs
- Supervision
- Optimisations

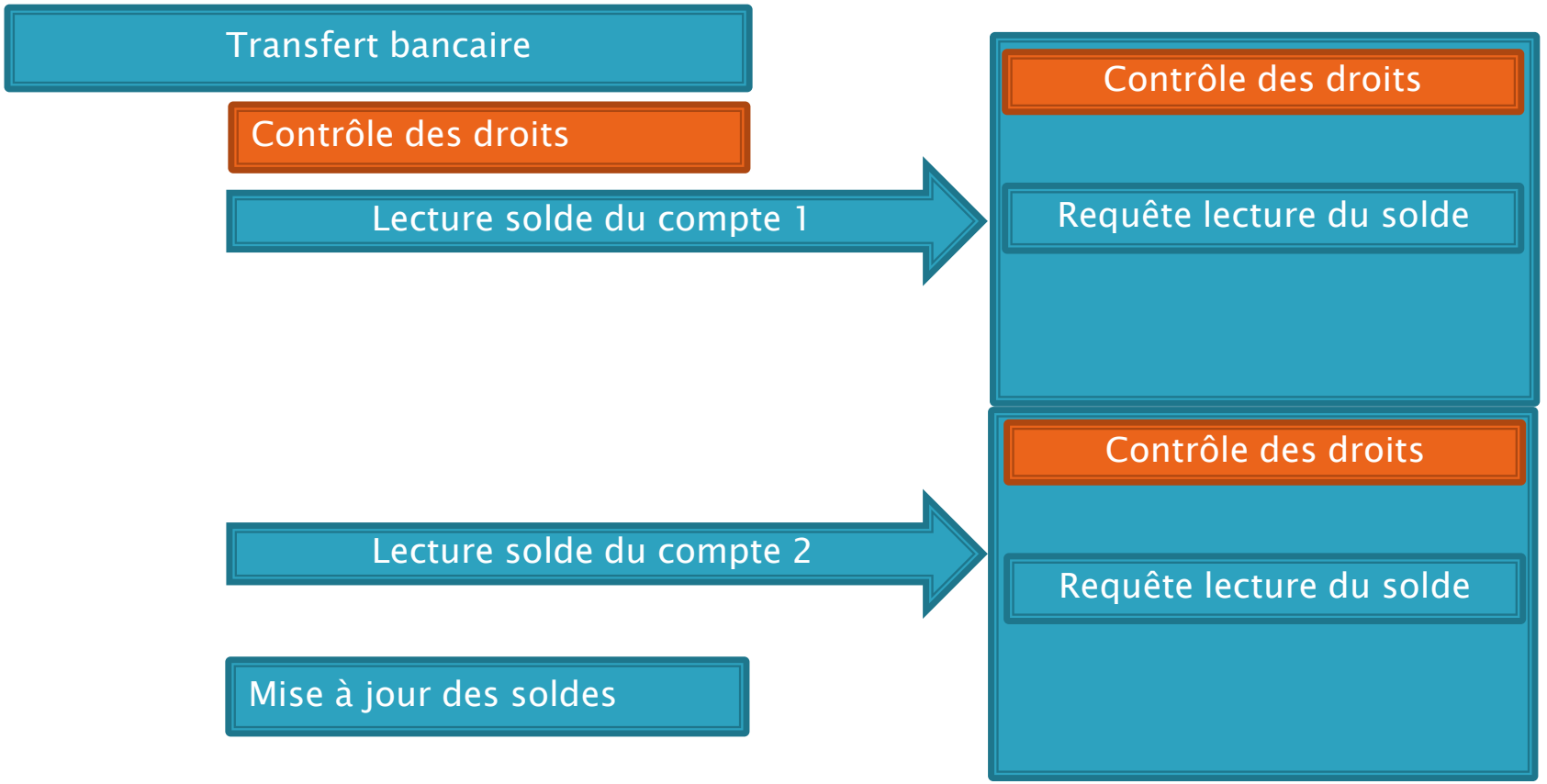
Les solutions

l'en-dur-ance / La pattern-ité / l'AOP

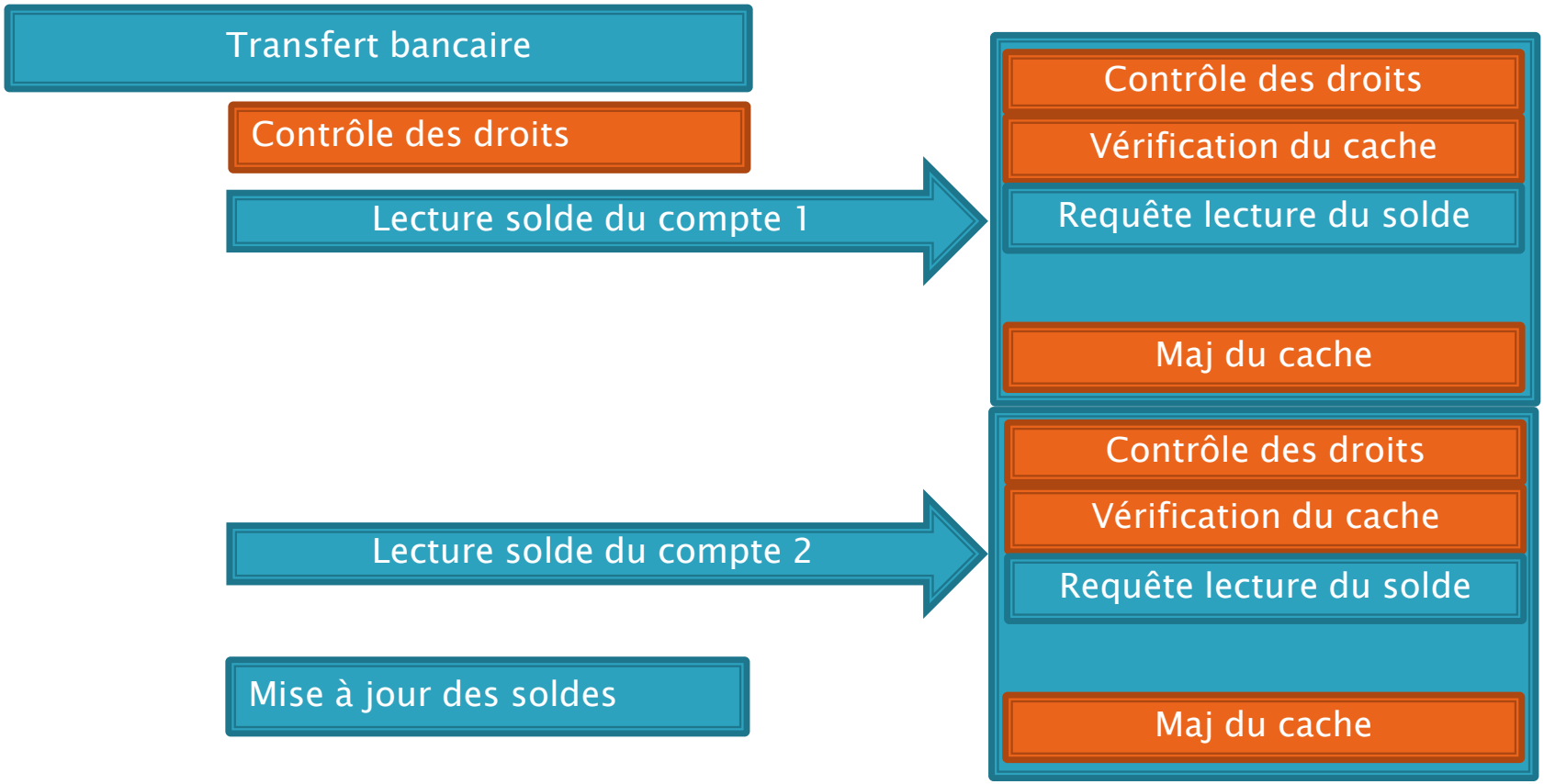
Le code métier



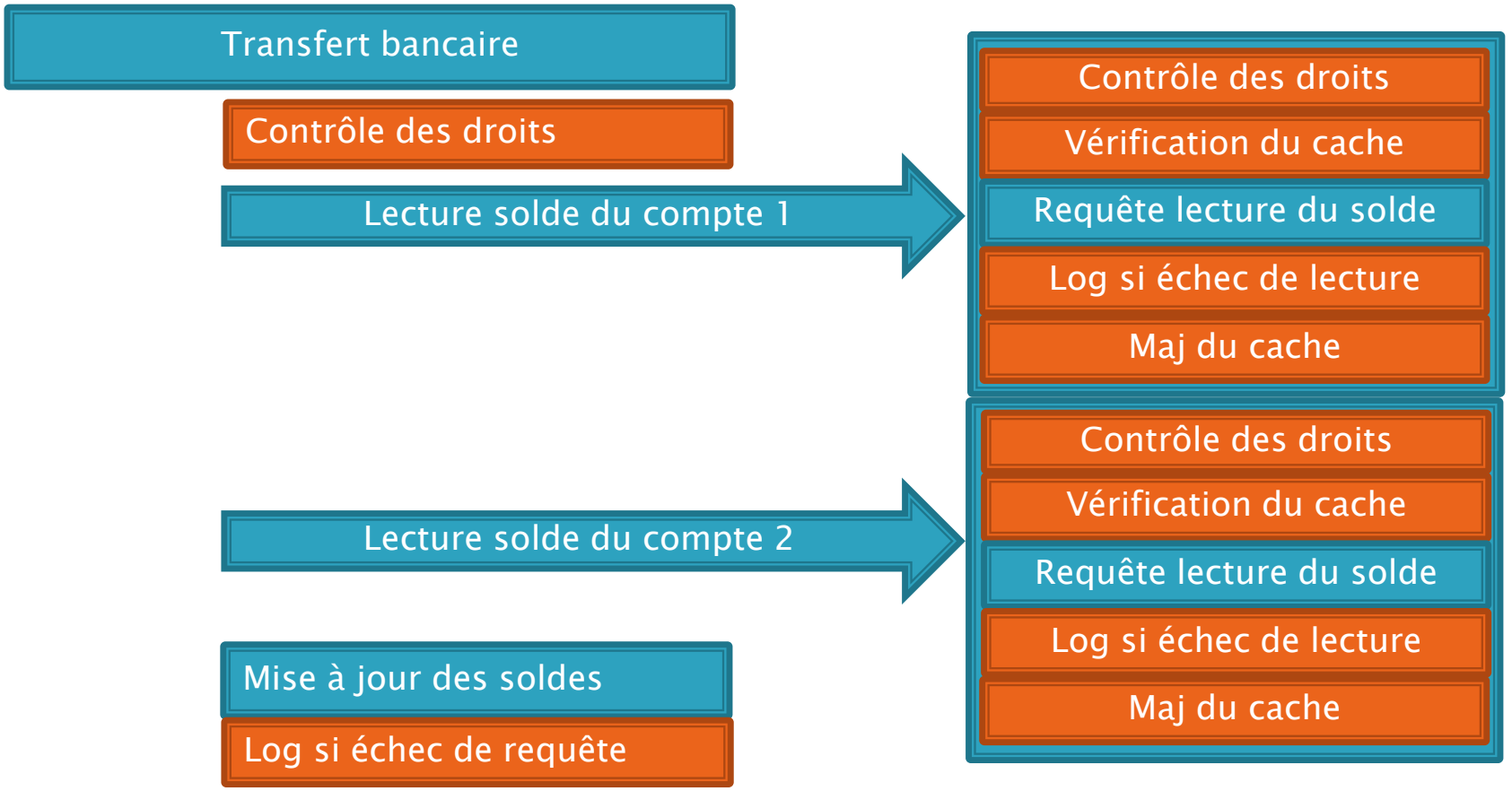
Métier + Gestion des droits



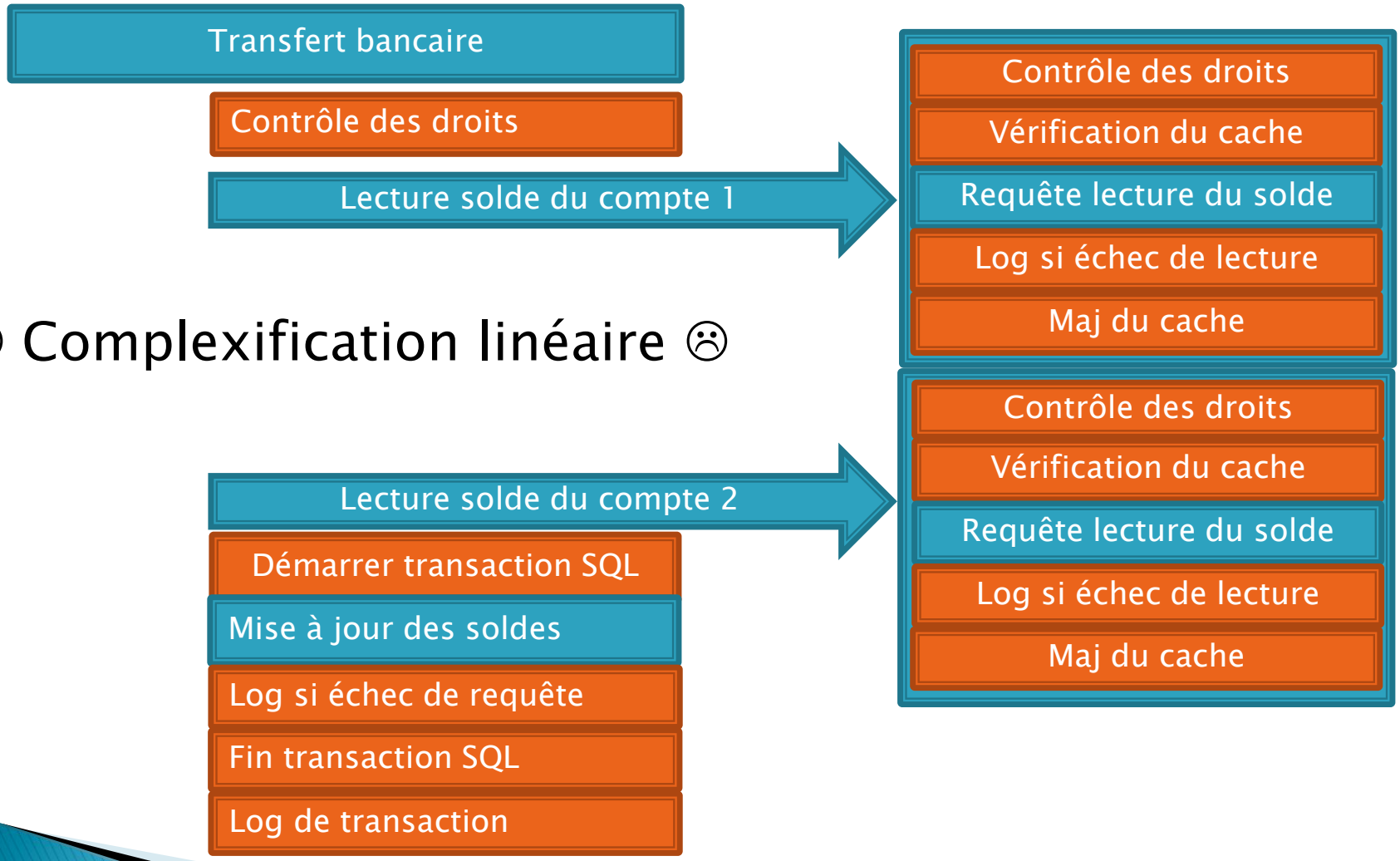
… + Gestion du cache



… + Gestion des logs



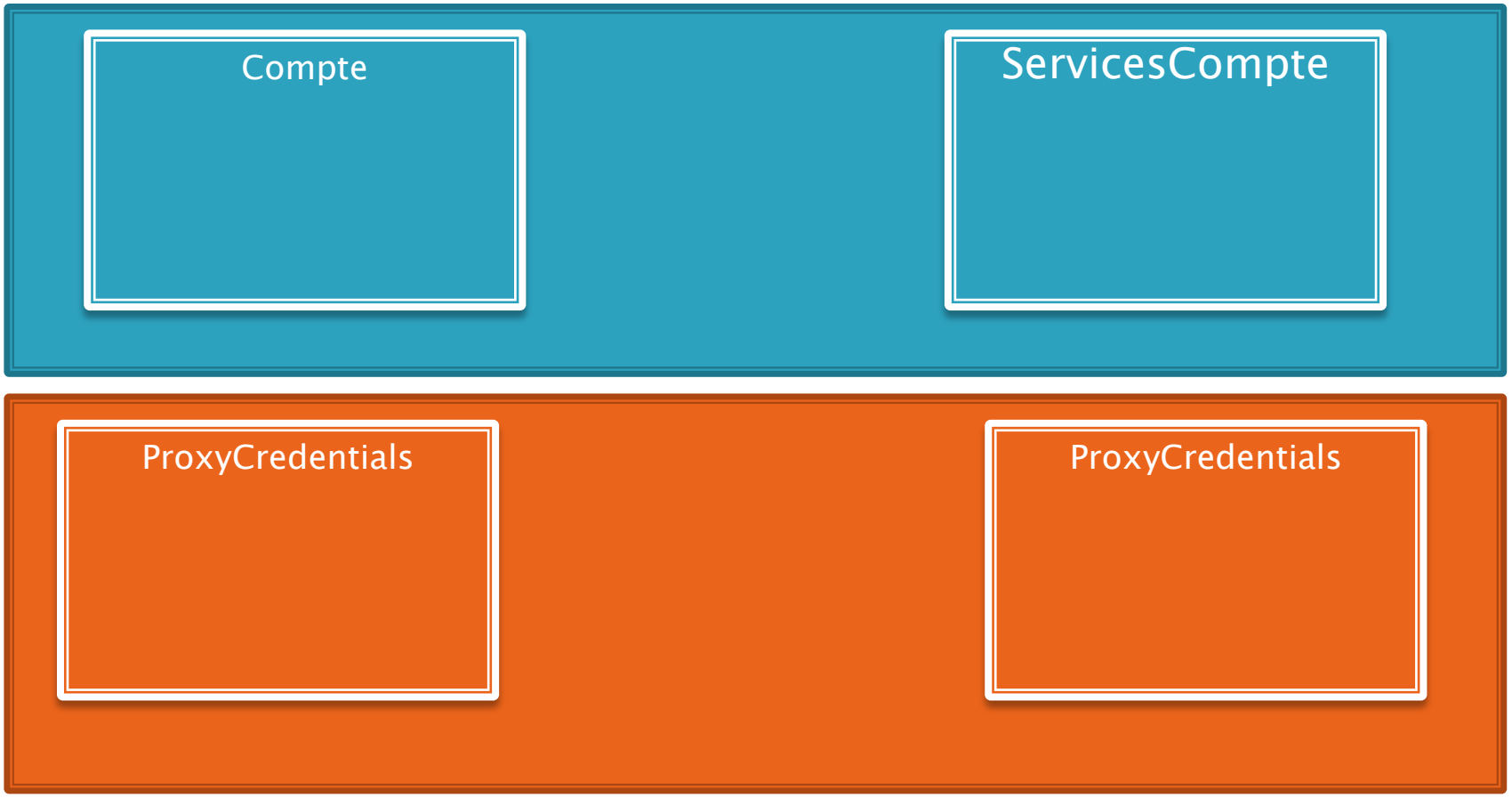
… + Gestion transactionnelle



☹ Complexification linéaire ☹

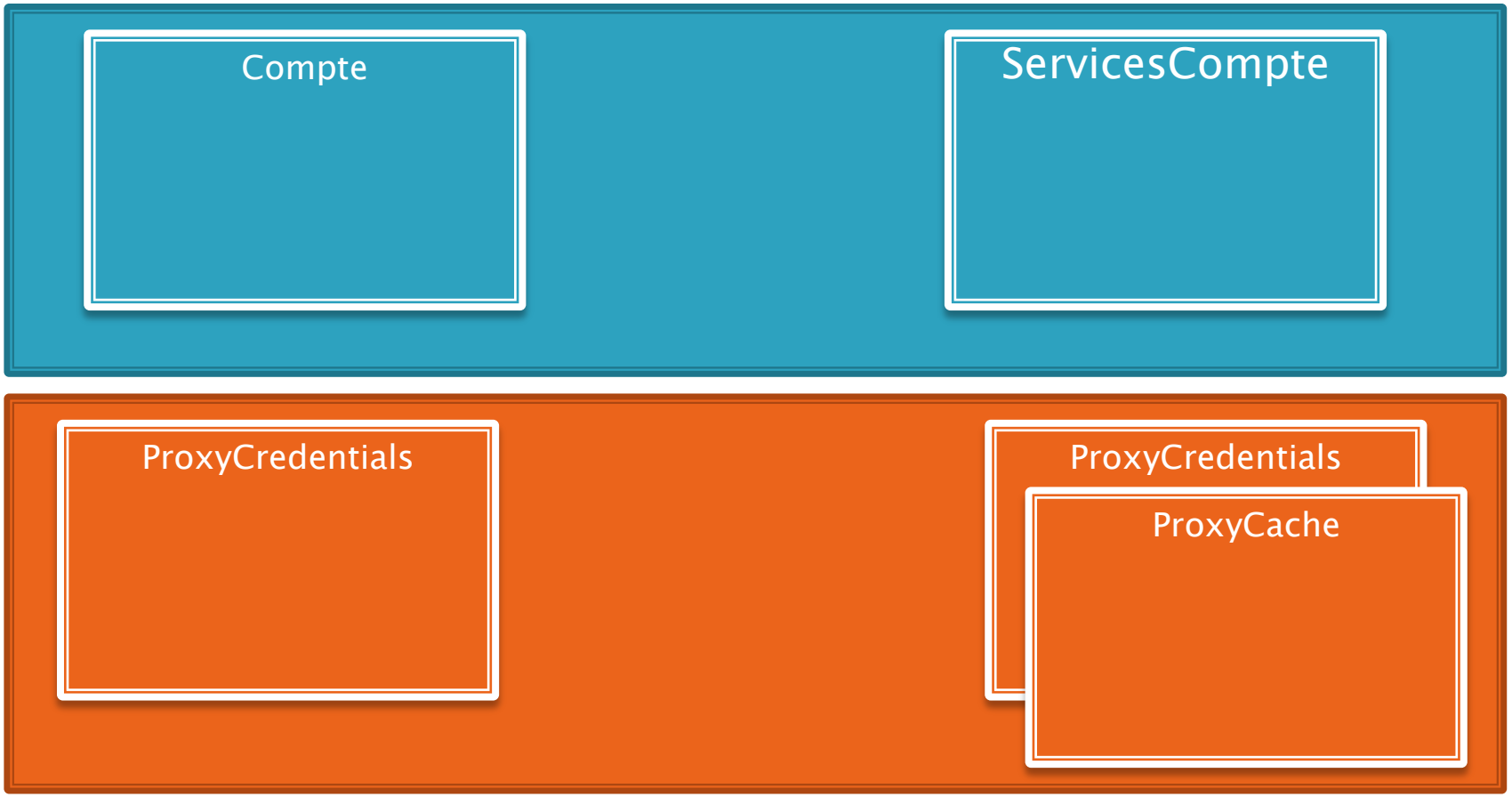
Le modèle objet du métier

… + Gestion des droits

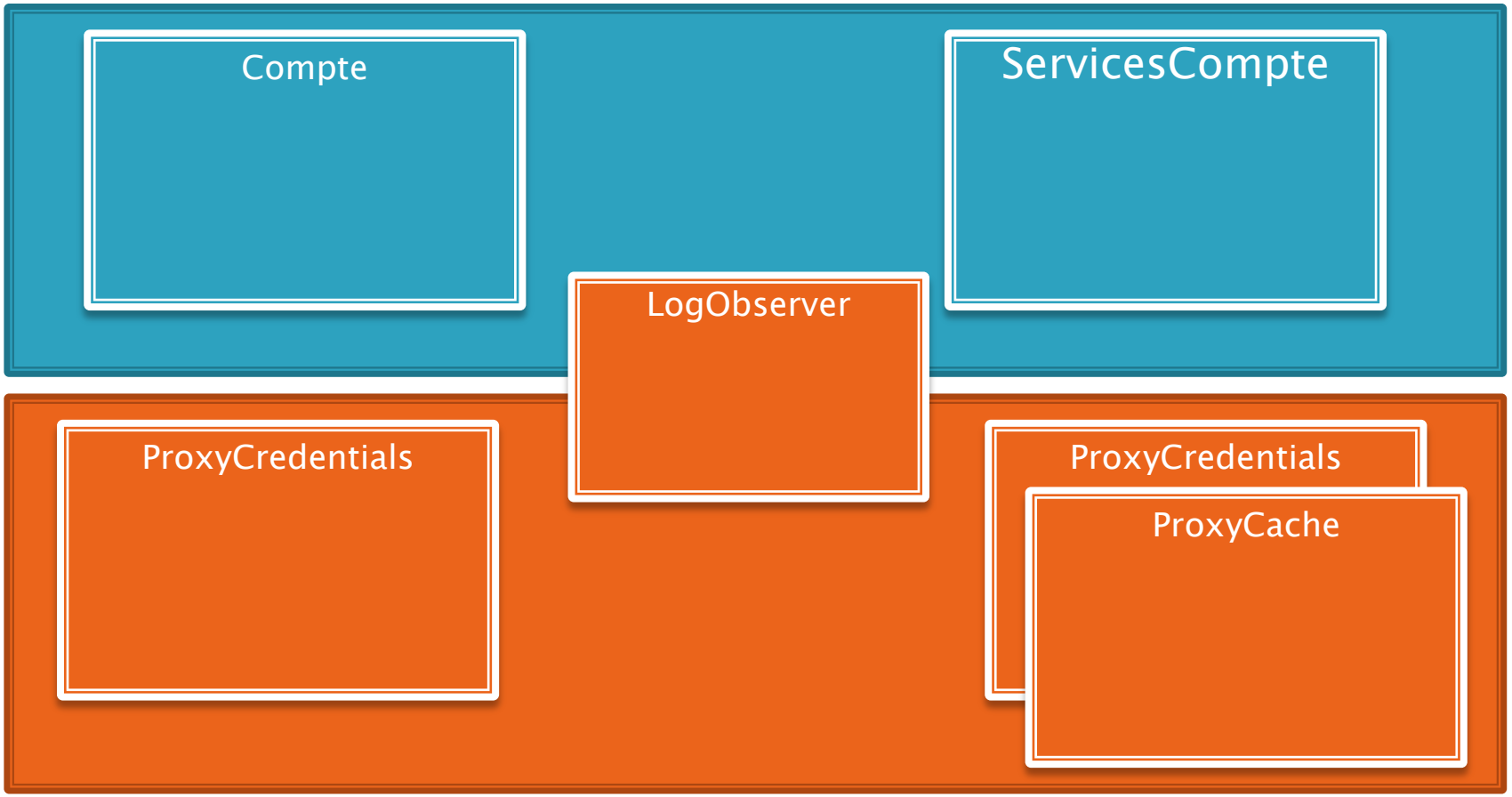


Proxy ? Se comporte comme l'objet distant aux yeux du code client

… + Gestion du cache

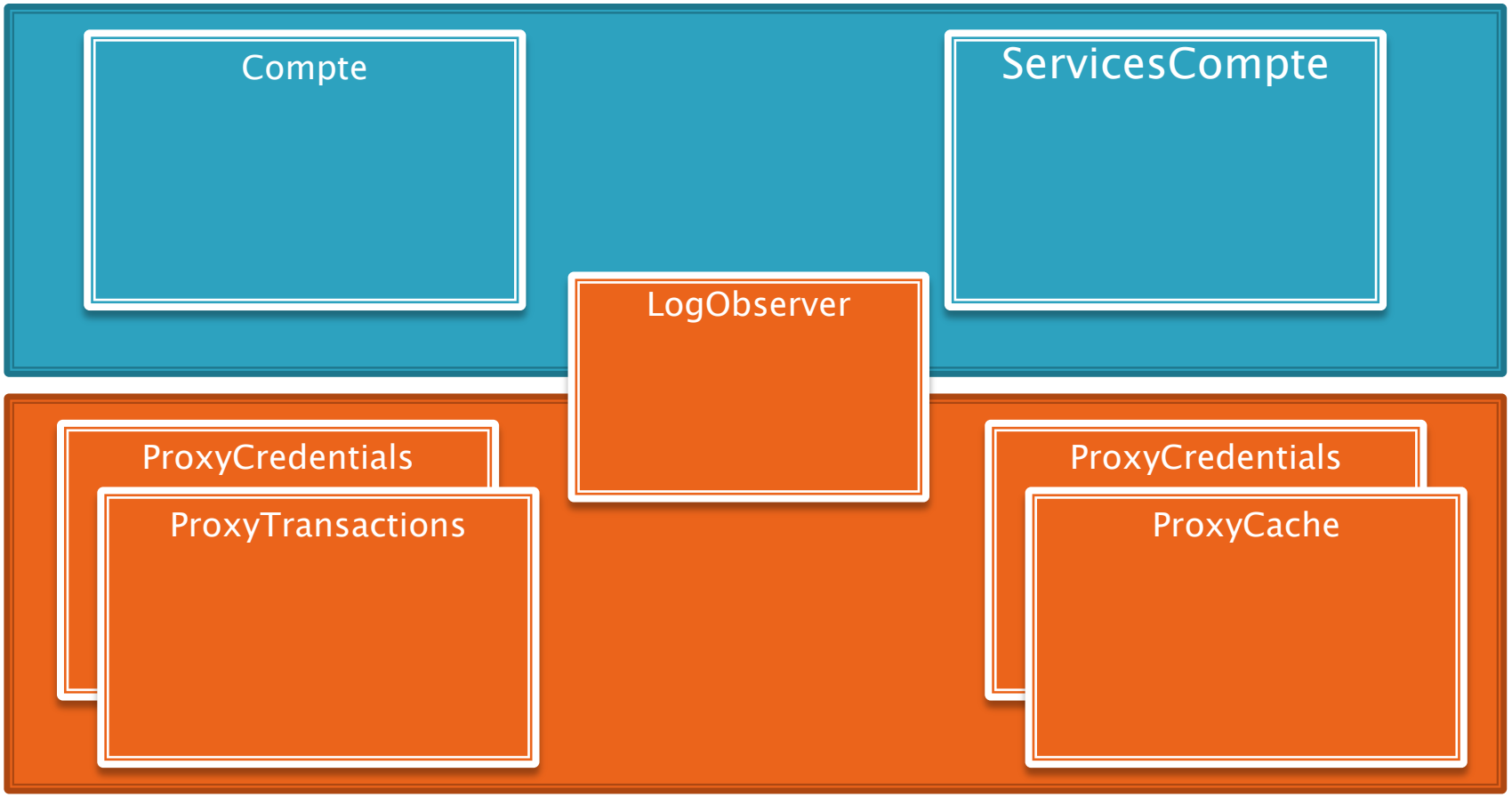


… + Gestion des logs



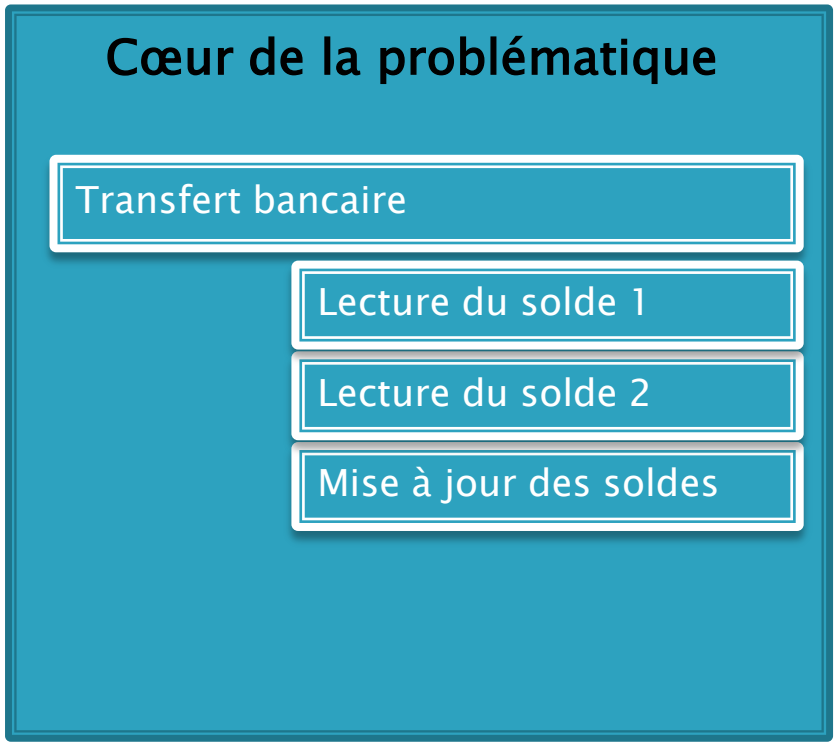
Observer ? Réagit aux événements d'un objet observé

… + Gestion transactionnelle



Complexification du modèle objet
Le code client est impacté

Un métier simple



Le métier doit rester intact

Cœur de la problématique

- Transfert bancaire
  - Lecture du solde 1
  - Lecture du solde 2
  - Mise à jour des soldes

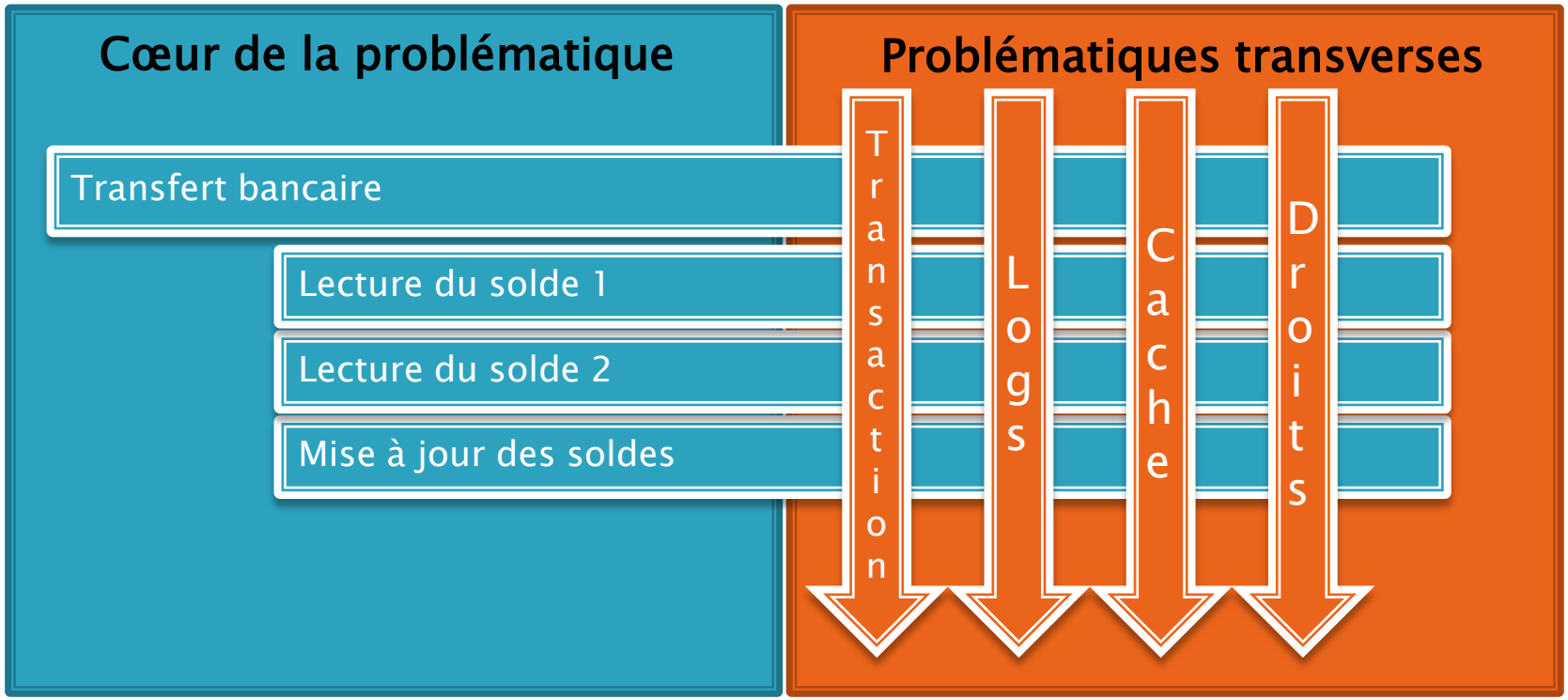
You Shall Not Pass !

Problématiques transverses

Problématiques transverses

Problématiques transverses

La réponse AOP

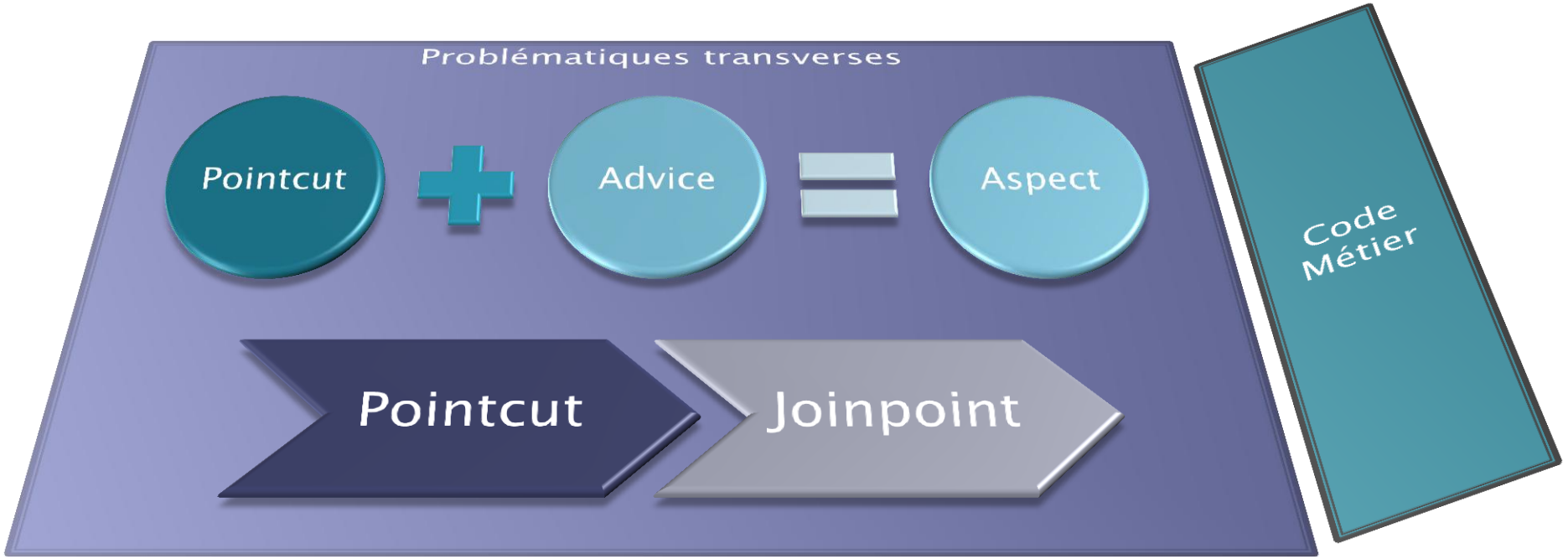


Un moyen de découpage supplémentaire

Ok… allons pour l'AOP

…et son vocabulaire si convivial

Le glossaire de l'AOP



One Weaver to bring them all, and in the darkness bind them

Un point de coupe ?

(ou pointcut) décrit le moment, dans le flot d'exécution, ou le programme peut être modifié.

Point de coupe

- Un langage de sélection
- Varie selon les implémentations

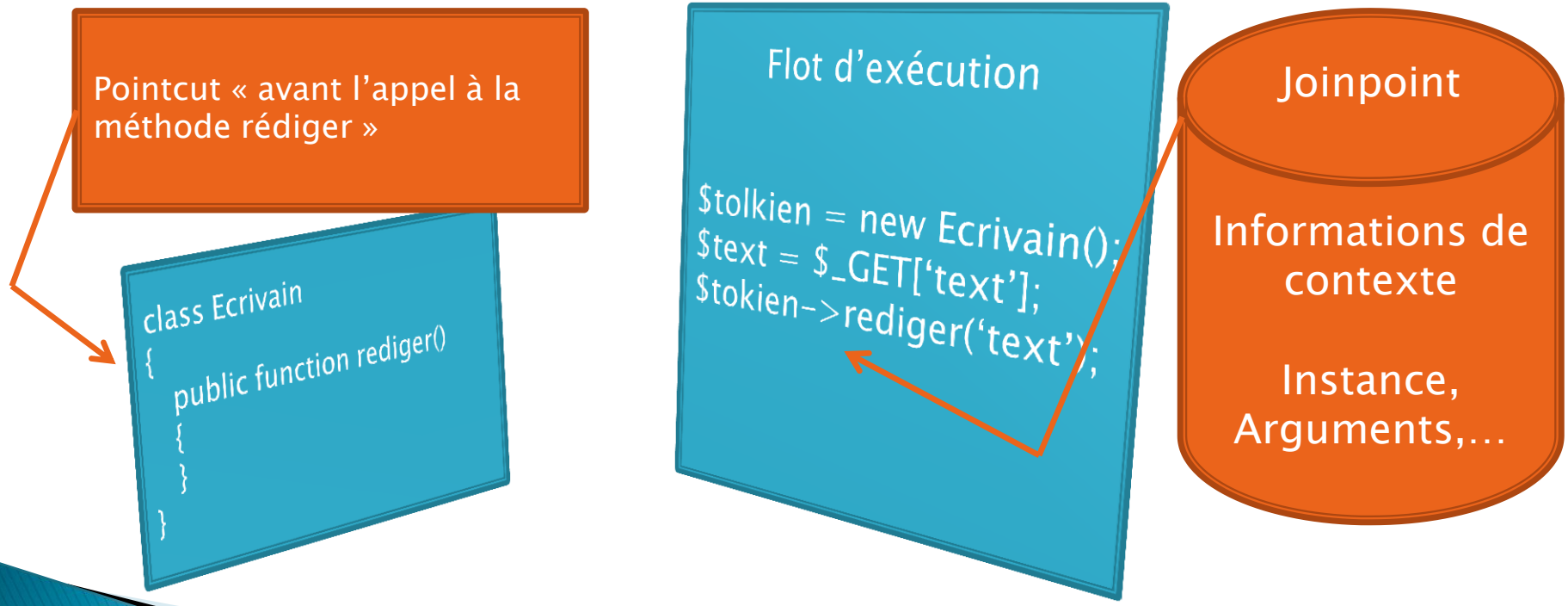
« Avant tous les appels aux méthodes adminQqChose »
« Après appel de la méthode écriture des instances de Ecrivain »
« Autour de l'appel à file_get_contents() »

Un point de jonction ?

(ou joinpoint) est un moment précis dans le flot d'exécution.

Point de fonction/ Joinpoint

- Contient un contexte
- Le résultat d'un point de coupe

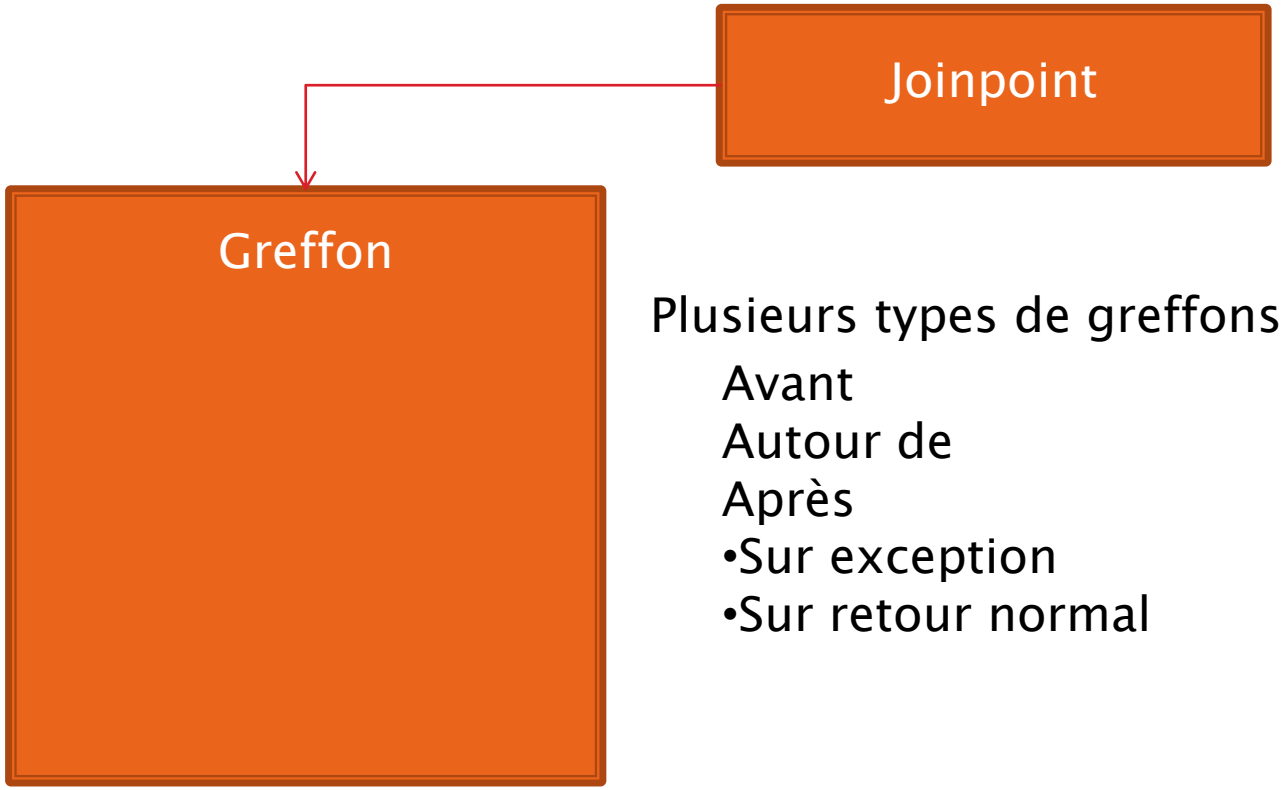


Un greffon ?

(ou advice) est l'action à mener à un point de jonction

Greffons

- La problématique à implémenter



Plusieurs types de greffons

- Avant
- Autour de
- Après
- •Sur exception
- •Sur retour normal

Aspect

(ou aspect… ☺) est la rencontre entre un point de coupe et un greffon

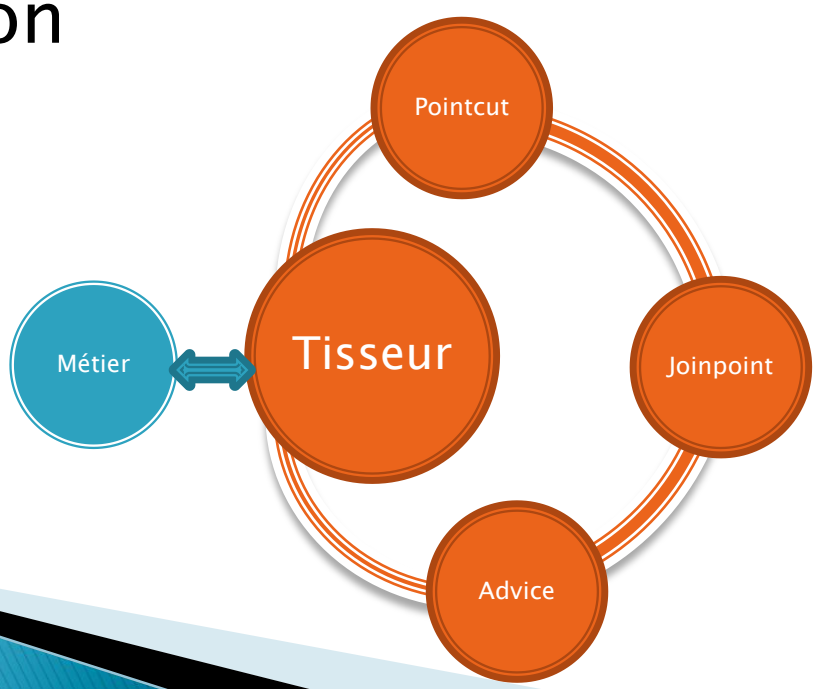
Un tisseur ?

(ou weaver) est l'outils qui permet de lier les aspects au flot d'exécution

Plusieurs types de tisseurs

- Compilation

- Exécution



Sérieusement…

Avec des noms pareils, des gens font de l'AOP ?

L'AOP aujourd'hui

- Java
  - Spring
  - AspectJ

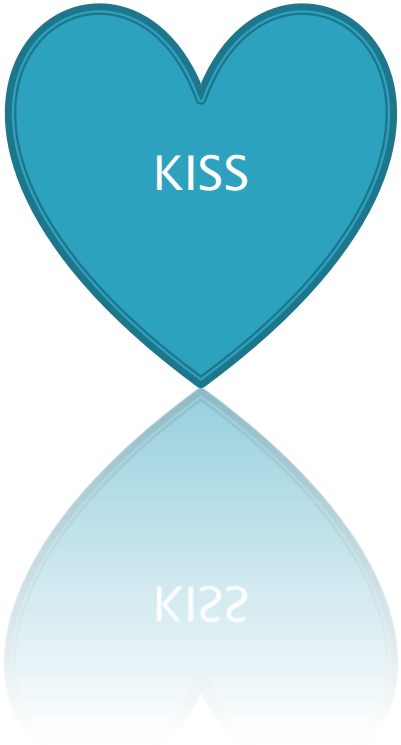
- PHP
  - Flow3
  - Symfony2 JMSAOP
  - … runkit ?

Et si j'aime pas les frameworks ?

Il y a l'extension : `sudo pecl install aop-beta`

Philosophie de l'extension

- Simplicité de compréhension
- Simplicité dans la syntaxe
- Simplicité dans la mise en œuvre



Objet

Procédural

Framework

…ou pas

Pure PHP

« DDTDD »

Prouvez-le !

Sans vous commander…

Lancer une exception sur erreur de lecture de fichier

Advice

```
$checkFGC = function (AopJoinpoint $jp) {
 if ($jp->getReturnedValue() === false) {
  $arguments = $jp->getArguments();
  $file      = $arguments[0];
  throw new Exception(Could not read from $file);
 }
};
```

Métier

```
$a = file_get_contents('file.txt');
```

```
aop_add_after('file_get_contents()', $checkFGC);
```

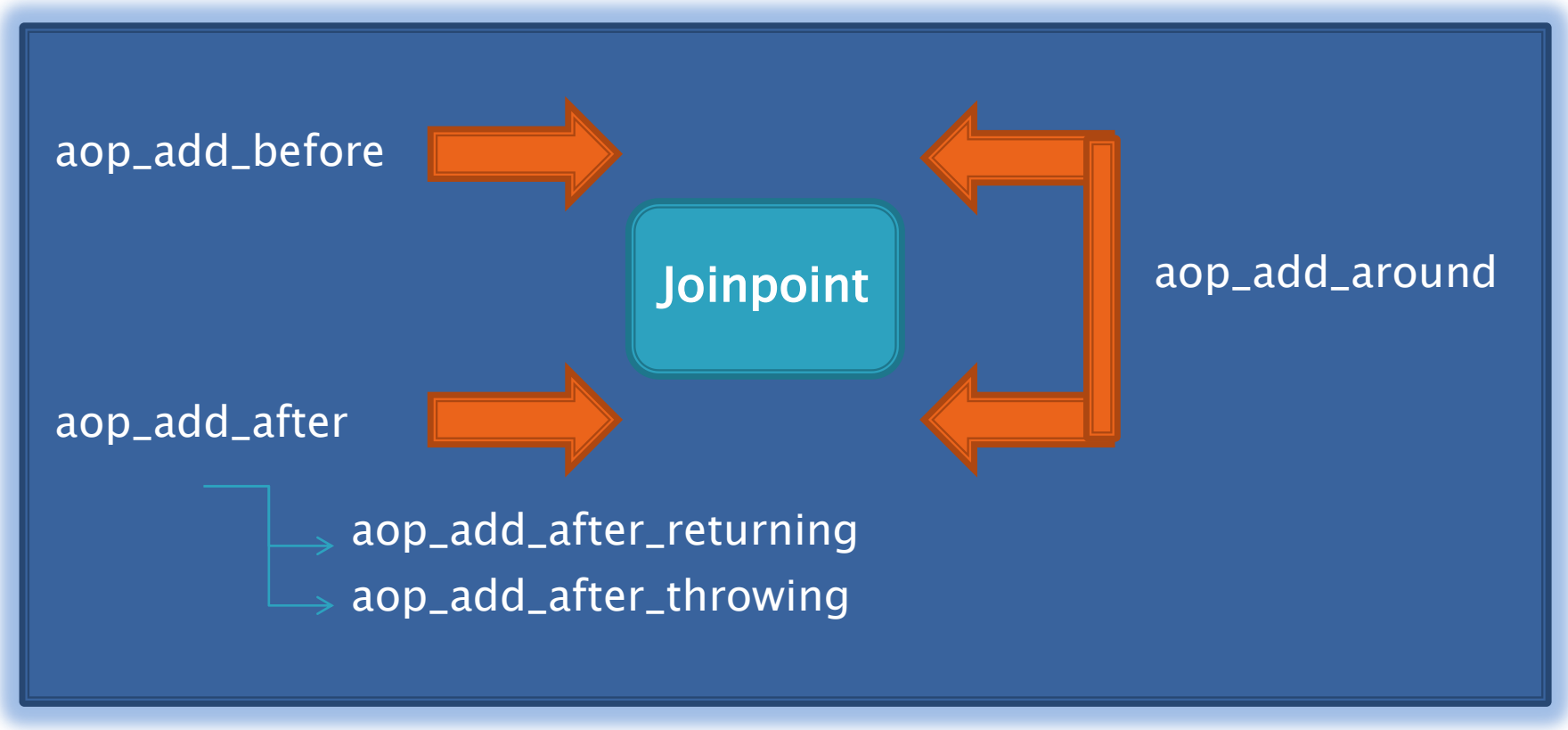
Aspect

Très peu de vocabulaire nécessaire pour profiter de l'extension.
Pas de compilation…

Ok, c’est plutôt facile…

Y’a quoi d’autre dans l’extension ?

Les 3 fonctions de l'API (+2)



Super, 3 fonctions… pas trop fatigués ?

En même temps, c’est facile à retenir.
On regarde comment les utiliser ?

Construction d'un aspect

Callback

```
aop_add_[type] ( Pointcut , Advice );
```

```
nom_fonction()
nomClasse->nomMethode()
Namespace\Classe->methode();
Classe->propriete;
```

Les pointcut alors… ils se décrivent comment ?

Avec une chaine de caractère

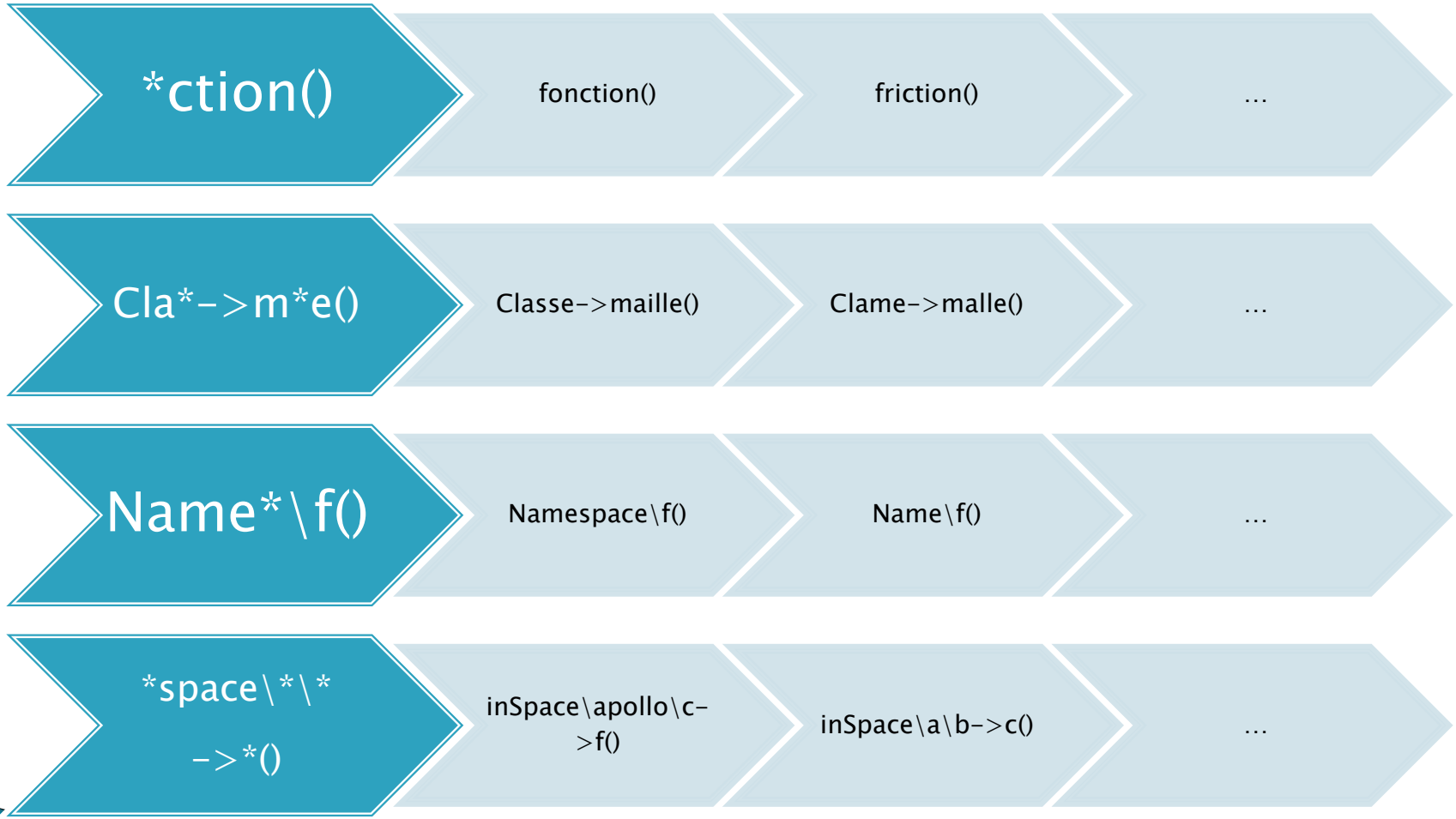
La syntaxe « Pointcut »

- fonction()
- Classe->methode()
- Namespace\fonction()
- Namespace\Classe->methode()

On doit tout taper à chaque fois ??

Non, on a prévu des petits aménagements !

Wildcard *s'arrête sur namespace*



Super Wildcard *Traverse les namespaces **

**ction()	fonction()	n\friction()
Name**\f()	Namespace\foo\f()	Name\f()
space->*()	DL\Space\M\c->f()	inSpace\a\b->c()

* Même \Chuck\Norris\

On décrit que des fonctions et des classes ?

Oui, avec bien sûr les plus habituels…

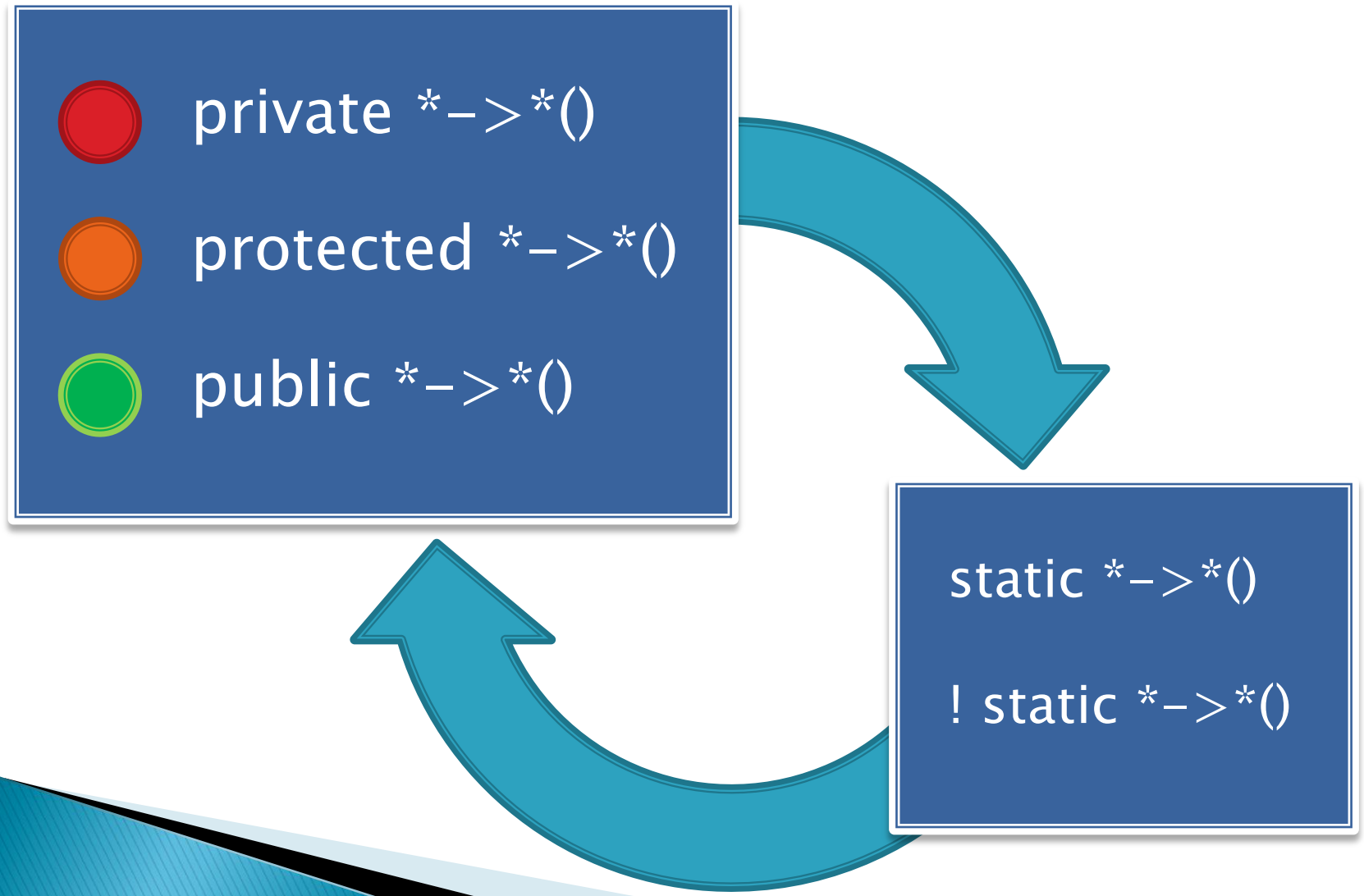
Interface / Trait

- Trait::methode
- Interface::methode
- Class::methode

Des cas particuliers ?

Pas grand-chose…

Encapsulation / Mode



Et donc les advices ?

Comme on est en PHP, c'est du « callable »

Advice – callable

```
Aop_add_before('pc',
array($object, 'methode');
```

```
Aop_add_before('pc',
'nom_de_fonction');
```

```
Aop_add_before('pc',
$variable_callable);
```

```
Aop_add_before('pc',
function ($joinpoint) {});
```

http://php.net/manual/fr/language.types.callable.php

Un joinpoint en paramètre… pour quoi faire ?

Pour récupérer le contexte, et un peu plus

Ils savent d'où ils viennent

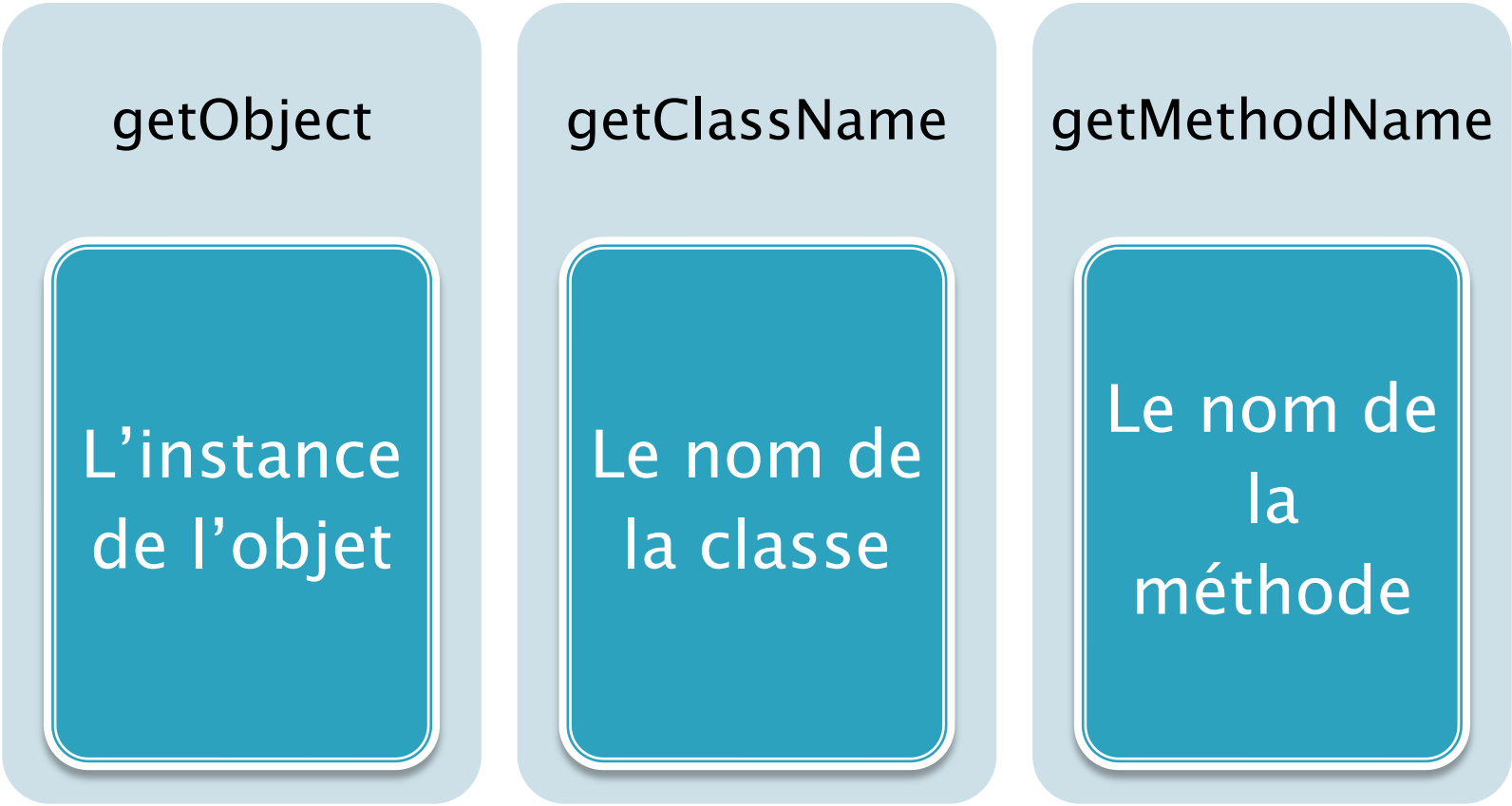
getKindOfAdvice

- AOP_KIND_AROUND
- AOP_KIND_BEFORE
- AOP_KIND_AFTER
- …

getPointcut

Le selecteur que vous avez utilisé

D’un appel de méthode



Et si c’est un appel statique ?

Ou d'un appel de fonction

C'est pas un peu du snobisme de dissocier fonction et méthode ?

Et les arguments ? Les retours ?

Spécifique en fonction du type de greffon

Avant exécution (aop_add_before)

getArguments

Tableau d'arguments

setArguments

Tableau des nouveaux arguments

Après exécution (aop_add_after)

getReturnedValue	setReturnedValue	getException
La valeur de retour	La nouvelle valeur de retour	L'exception soulevé dans le joinpoint

Et mes arguments ?

Et autours (aop_add_around)

Before

- getArguments
- setArguments

After

- getReturnedValue
- setReturnedValue

process

- Lance l'execution du joinpoint (bypass si omis)

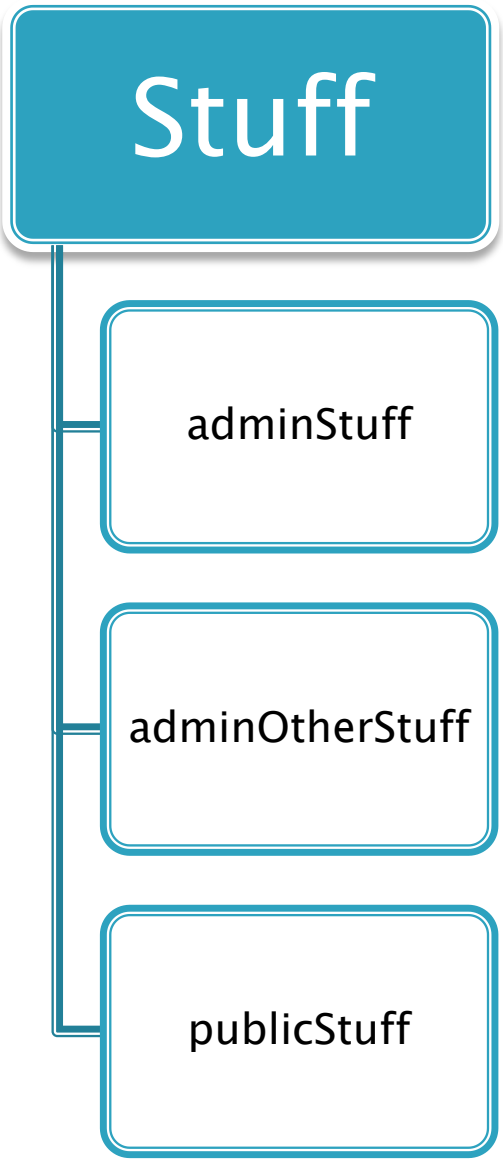
Un cas pratique ?

Un petit contrôle des droits !

La classe à protéger

```
<?php
class Stuff {
    public function adminStuff () {
        return ‘adminStuff’;
    }
    public function adminOtherStuff () {
        return ‘adminStuff’;
    }
    public function publicStuff () {
    }
}
```

Une classe bien stuffée

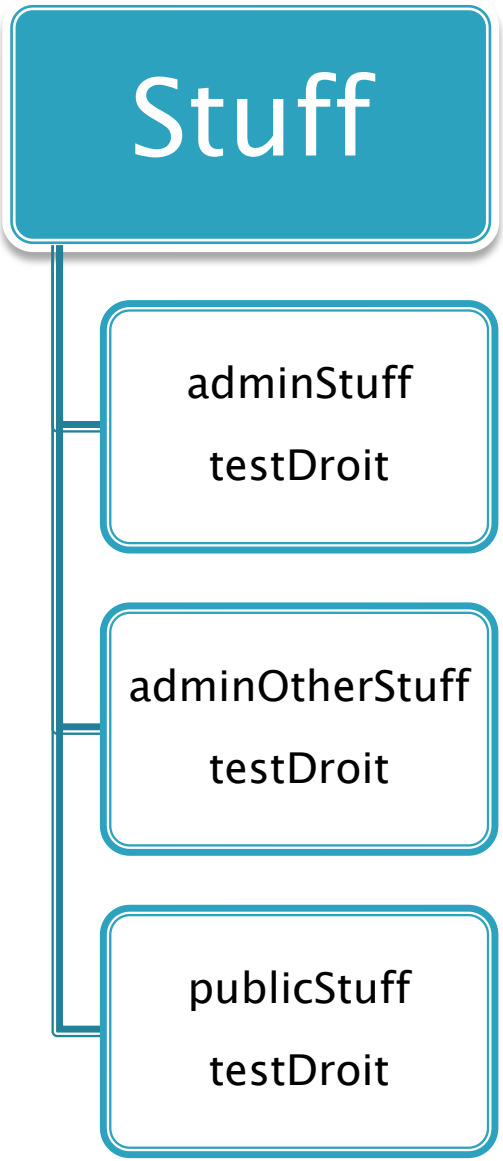


L’en-dur-oh

Après tout, c’est rapide

Les droits sont dans le code métier

```
<?php
class Stuff {
   public function adminStuff() {
      if (isset($_SESSION['user'])
          && $_SESSION['user']=='admin') {
         return 'adminStuff';
      } else {
         throw new \Exception ('You need to be admin');
      }
   }

   public function adminOtherStuff() {
   //,,,
   }
}
```



Le code métier est noyé !

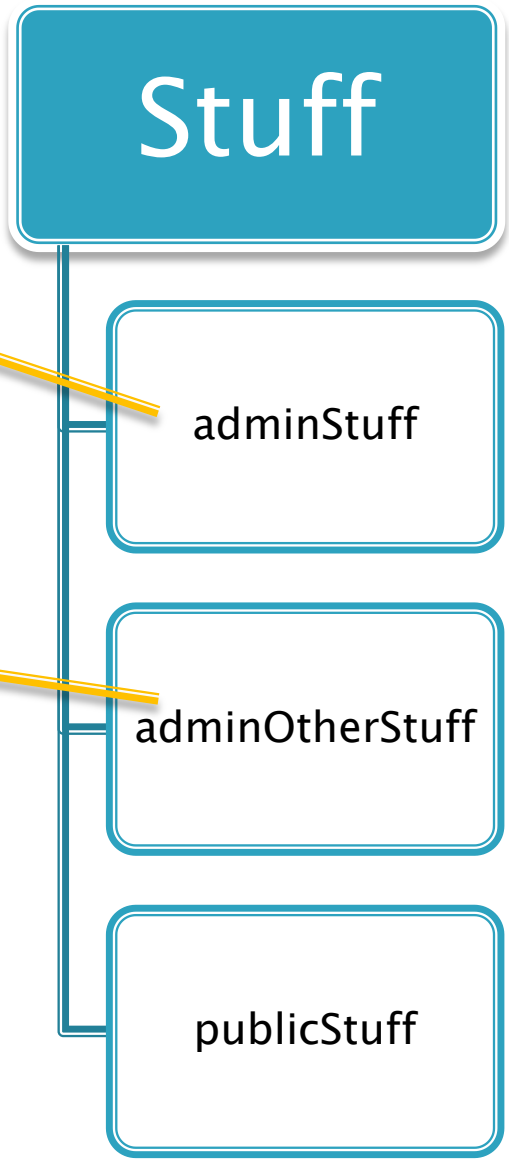
```
<?php
class Stuff {
   public function adminStuff() {
      if (isset($_SESSION['user'])
          && $_SESSION['user']=='admin') {
         return 'adminStuff';
      } else {
         throw new \Exception ('You need to be admin');
      }
   }

   public function adminOtherStuff() {
   //,,,
   }
}
```

Needle in haystack

Testons avant l'appel dans le code client

```
$stuff = new Stuff();
If (isset ($_SESSION['user']) &&
$_SESSION['user']=='admin') {
   $retour = $stuff->adminStuff();
} else {
   throw new \Exception ('You need to be admin');
}
```

```
$stuff = new Stuff();
If (isset ($_SESSION['user']) &&
$_SESSION['user']=='admin') {
   $retour = $stuff->adminOtherStuff();
} else {
   throw new \Exception ('You need to be admin');
}
```

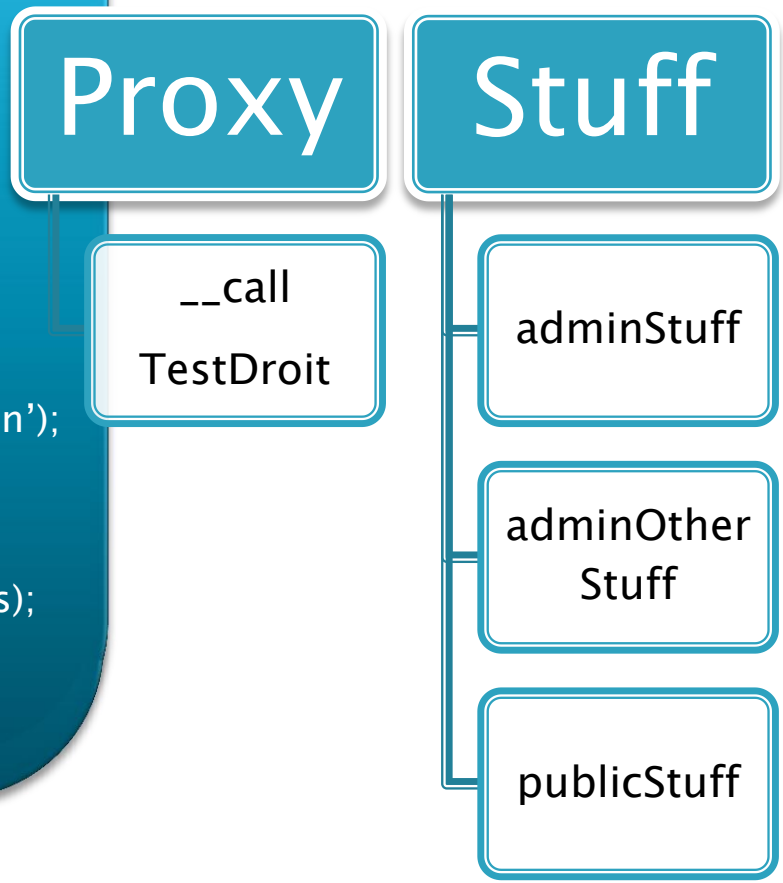


Avec des objets, ce serait plus « beau »

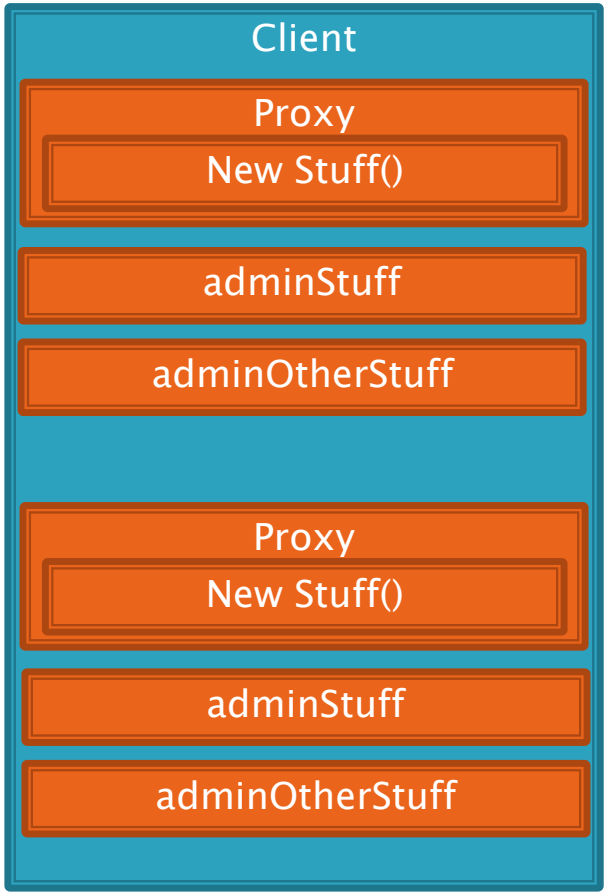
Pour être beau, c’est bien plus beau

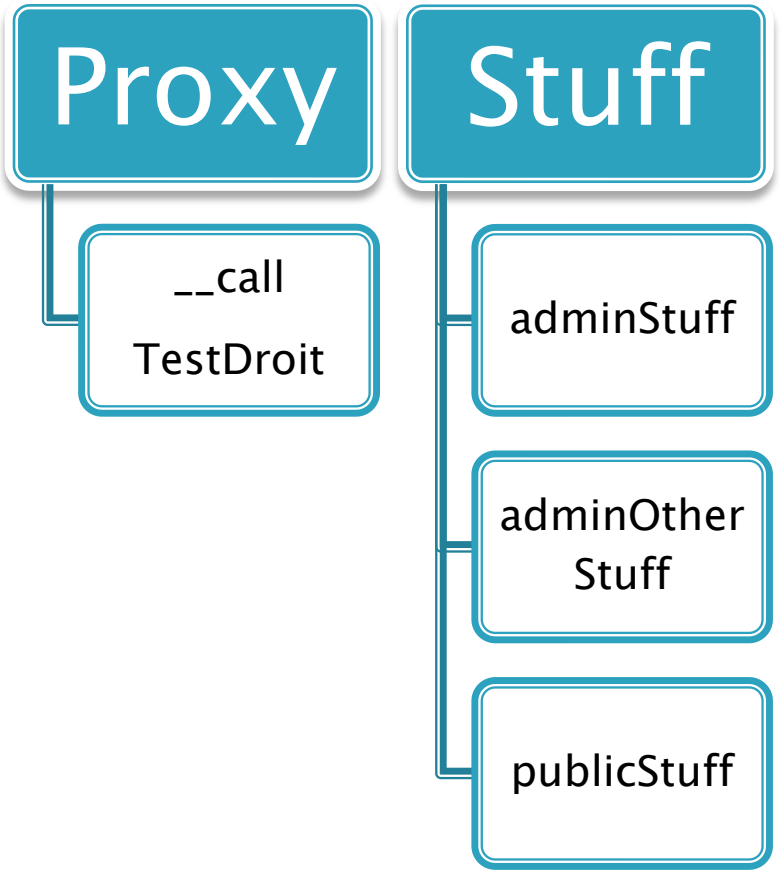
OOP Une version avec Proxy

```
Class ProxyCredentialStuff {
    private $stuff = null;
    public function __construct($stuff) {
        $this->stuff = $stuff;
    }
    public function __call ($pName, $pArgs) {
        if (substr($pName,0,5)=='admin') {
            if (!isset($_SESSION['usr'])
             || $_SESSION['usr']!='admin') {
                throw new \Exception ('You need to be admin');
            }
        }
        return call_user_func_array(array($this->stuff,
                                          $pName), $pArgs);
    }
}
```



Mais il faut penser à tous les appels clients





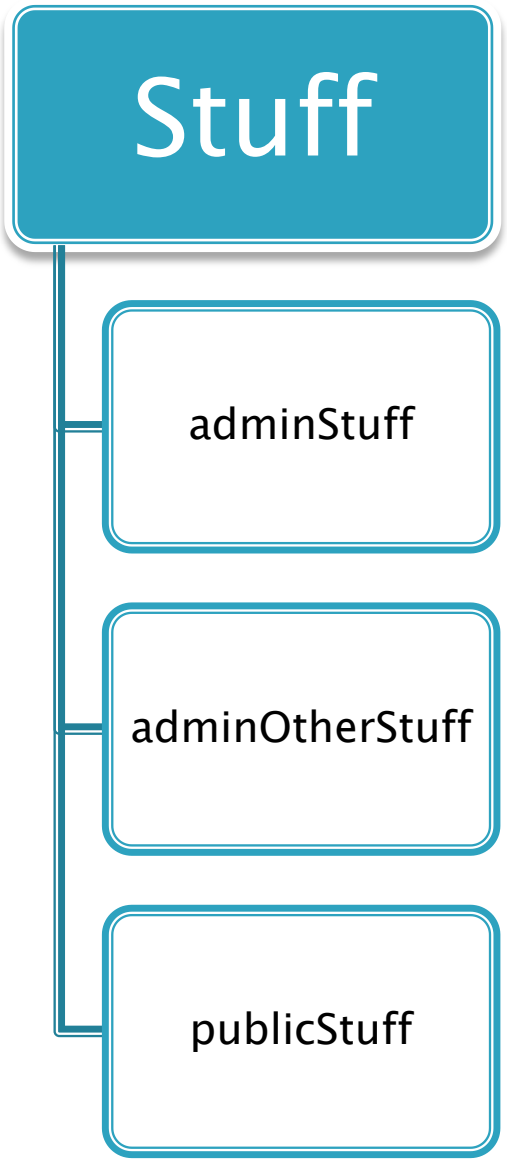
Je m'en fiche, moi j'utilise un framework d'injection de dépendances

Et donc, la version avec AOP ?

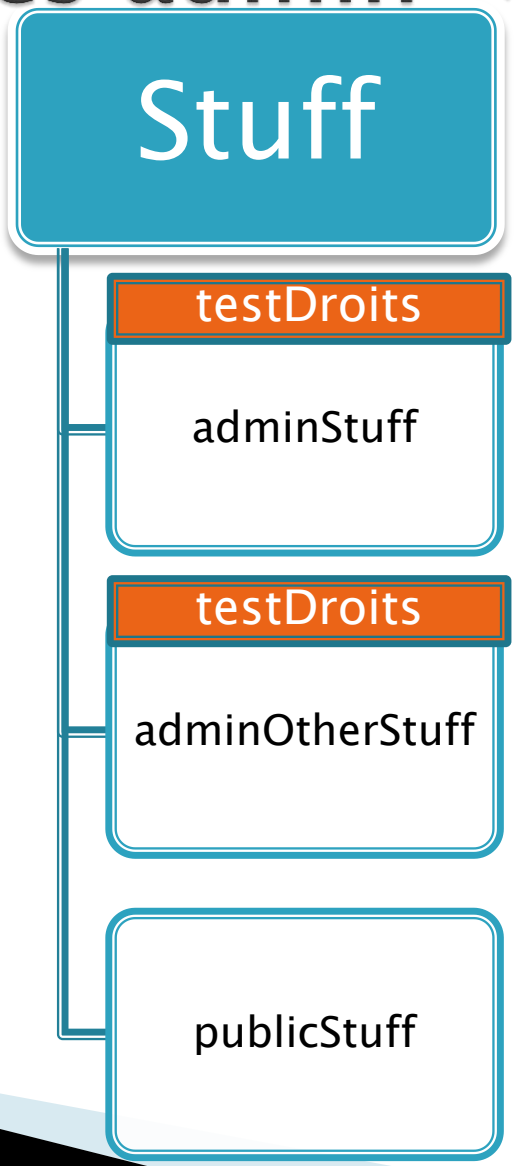
Une autre forme de découpage

« Ce qui serait bien »

```php
<?php
class Stuff {
    public function adminStuff() {
        return 'adminStuff';
    }
    public function adminOtherStuff() {
        return 'adminStuff';
    }
    public function publicStuff () {
    }
}
```



« c'est de tester les droits avant les méthodes admin* »



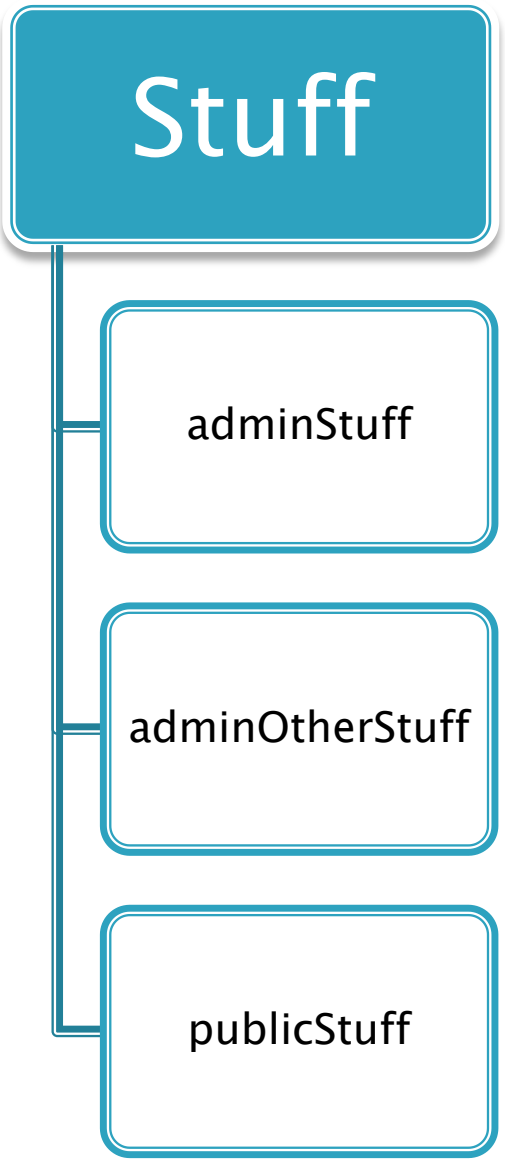
Mais c'est génial !

```
<?php
$advice = function () {
    if (!isset($_SESSION['user']
        ||$_SESSION['user']!='admin') {
      throw new \Exception ('You need to be admin');
     }
};
```

Advice

```
aop_add_before('Stuff::admin*', $advice);
```

Aspect

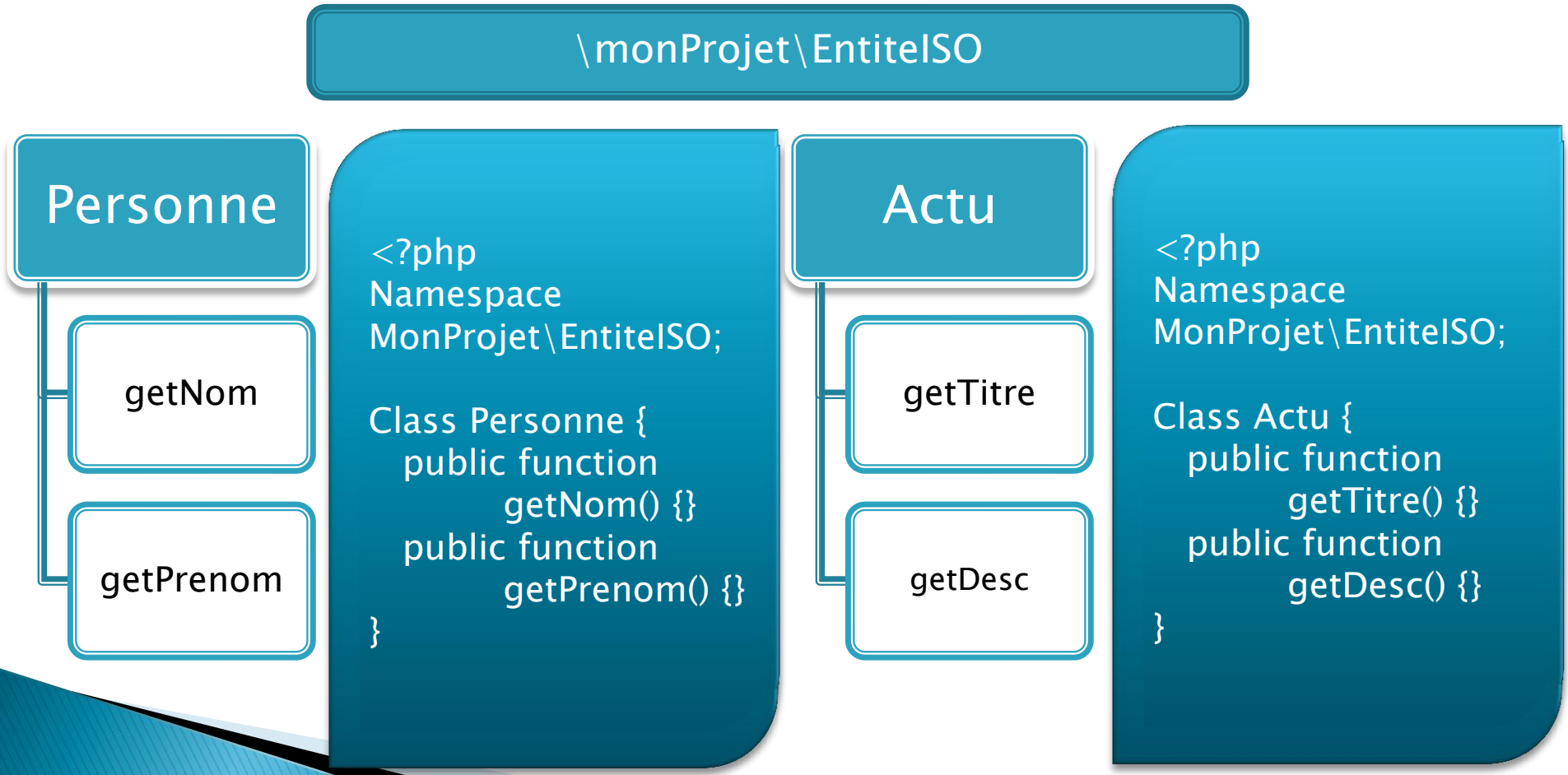


Et si on voyait quelques cas d'études ?

Ok, voici une gestion d'encodage avec « after »

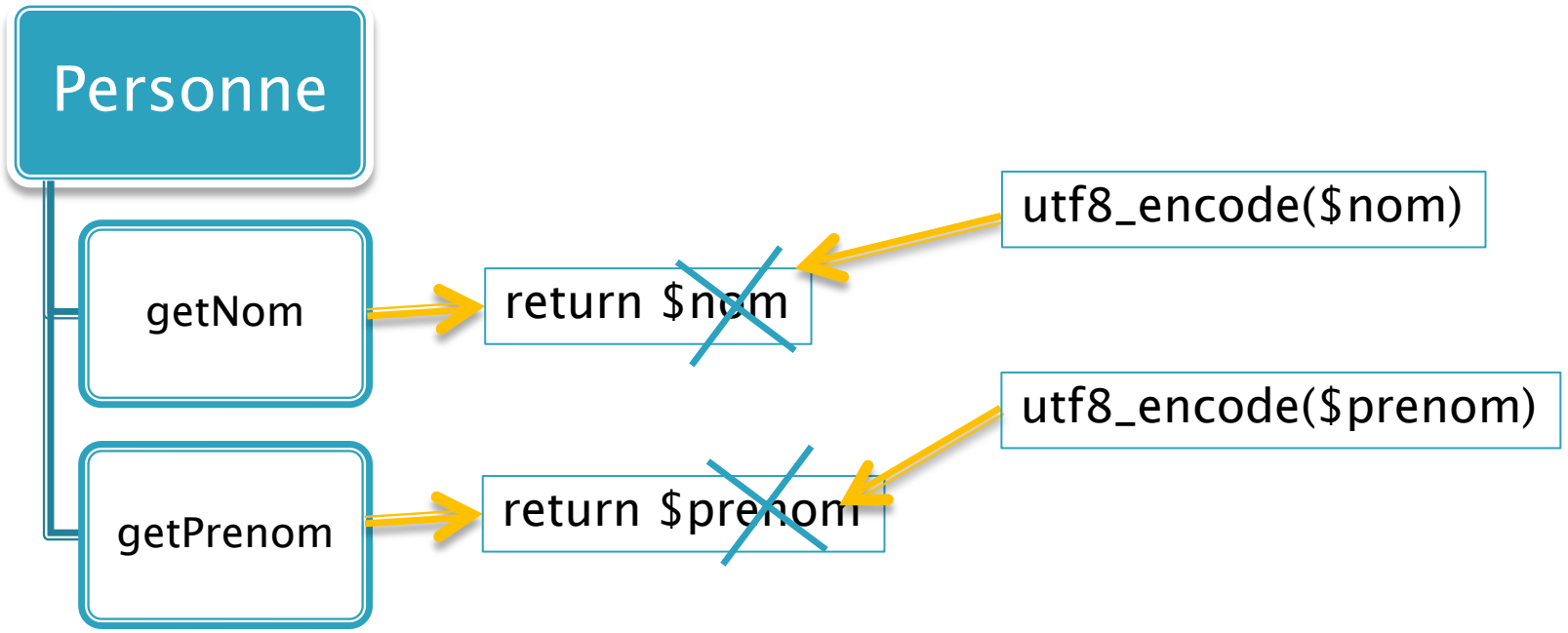
Des entités en ISO, on avait dit UTF8 !

- « ce qui serait bien »



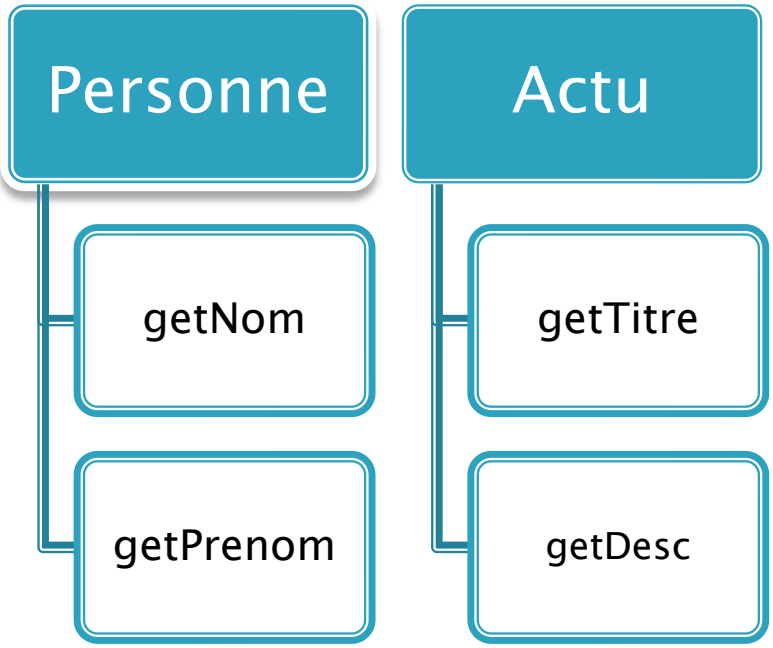
After peut modifier le retour

- « c'est que le retour des méthodes get soit utf8_encodées* »



*néologisme pour « ça marche même chez les Asteks »

Plus de concret, moins de théorie



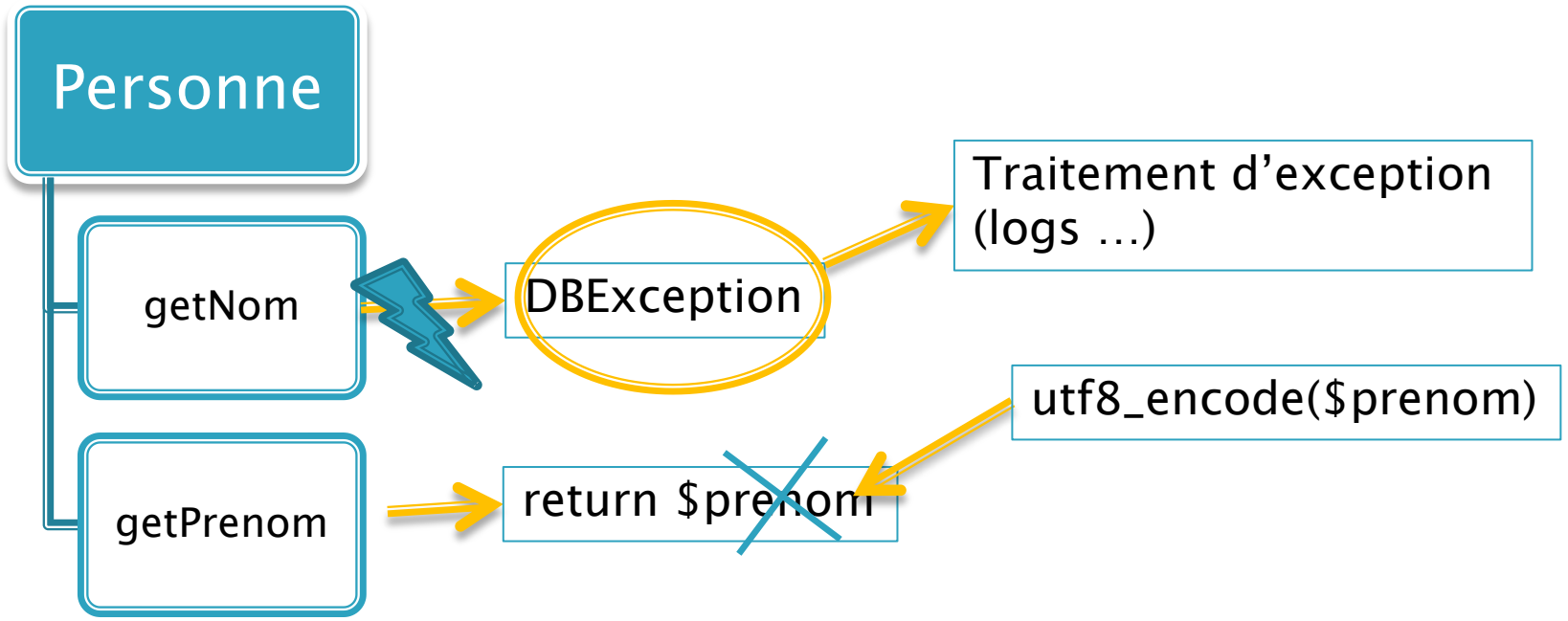

```
$advice_utf8 = function (AopJoinpoint $jp) {
   $retour = $jp->getReturnedValue();
   if (is_string($retour)) {
      return utf8_encode($retour);
   }
   return $retour;
}
```

Advice

```
aop_add_after ('MonProjet\EntiteISO\*->get*(),
                $advice_utf8);
```

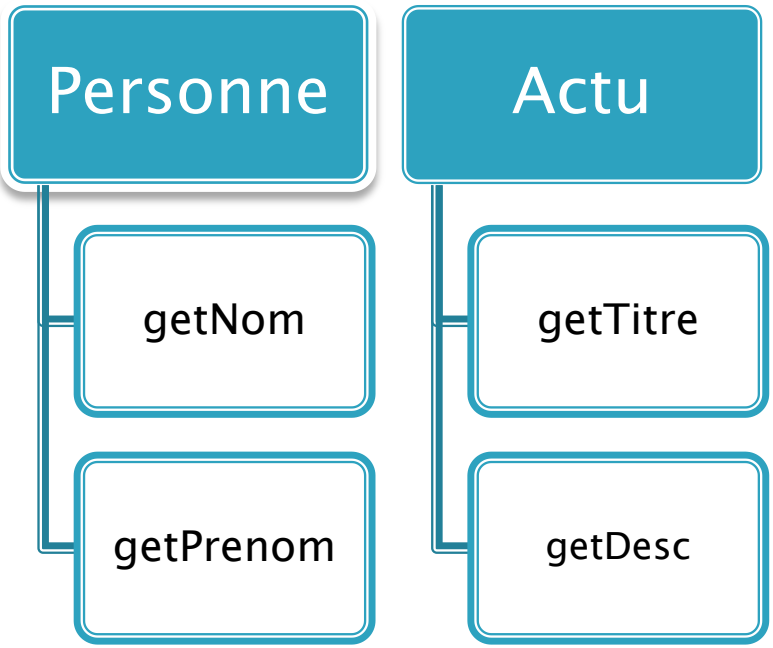
Aspect

Et si ca plante, il peut gérer l'exception ?



Traitement de l'exception

- Interrogation directe du Joinpoint




```
$advice_utf8 = function (AopJoinpoint $jp) {
   if ($jp->getException() !== null) {
      Mailer::doStuff();
   }
   //…
}
```

Advice

```
aop_add_after ('MonProjet\EntiteISO\*->get*(),
                $advice_utf8);
```

Aspect

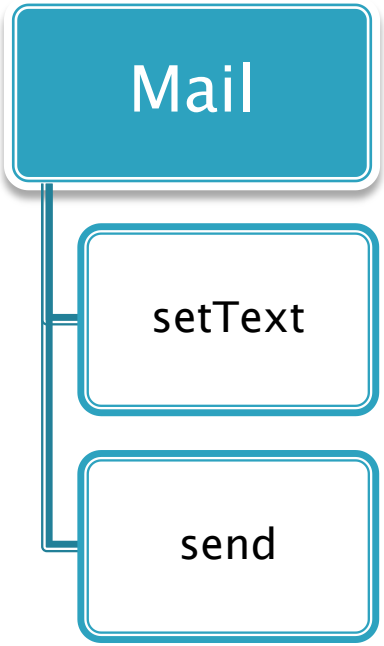
Et un cas avec before ?

Déjà vu avec la gestion des droits

Mais on va en faire un autre…

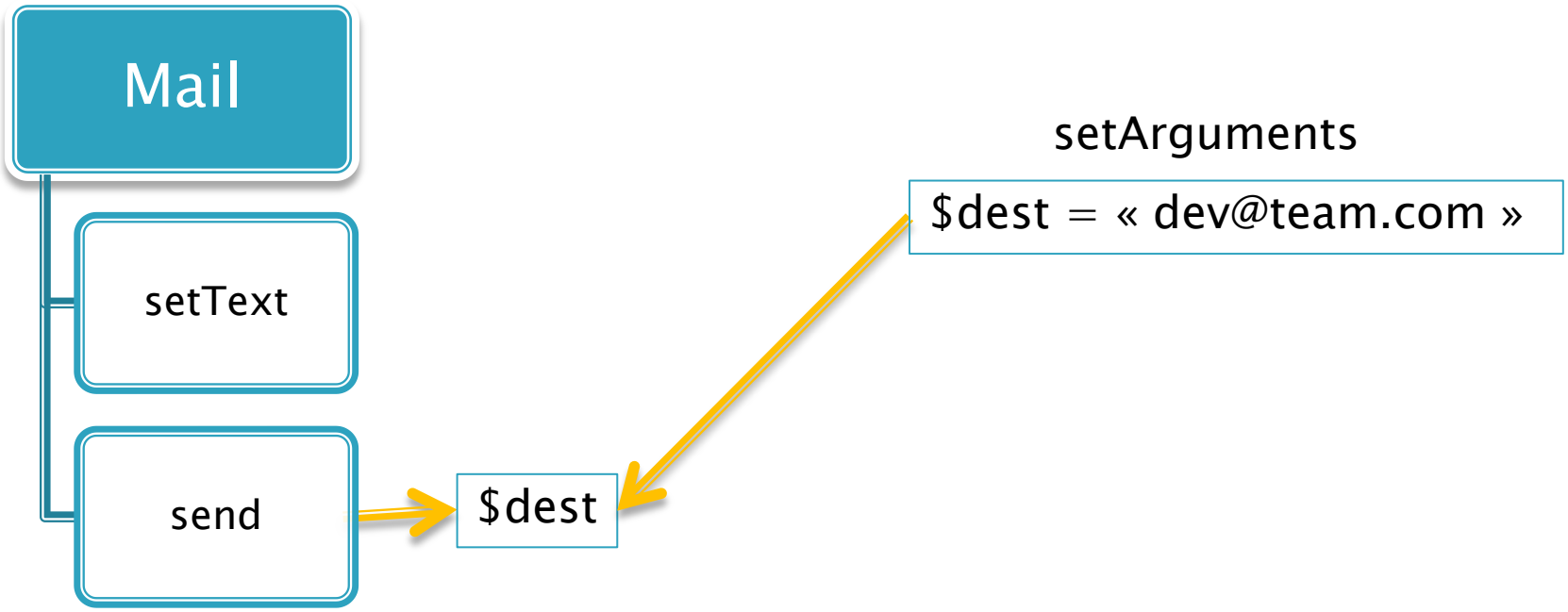
En recette, on ne veut pas envoyer de mails aux internautes

- « ce qui serait bien »

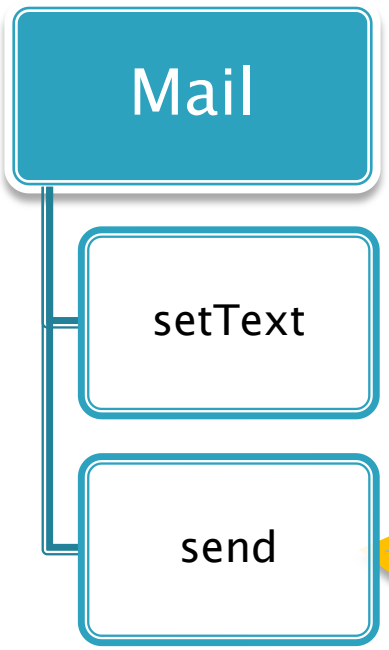


Avec setArguments()

- « c'est de changer le destinataire »



Le code parle de lui-même *




```
$advice_mail_recette = function ($jp) {
    $args = $jp->getArguments();
    $jp->setArguments(array ('dev@team.com'));
    $jp->getObject()->setText(
        'Normalement envoyé à :'.
        $args[0].  ''\n ''.$jp->getObject()->getText()
    );
};
```

Advice

```
aop_add_before ('Mail::send()', $advice_mail_recette);
```

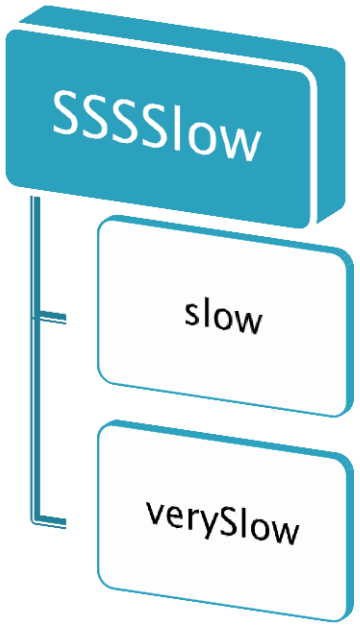
Aspect

* Il envoi même des mails

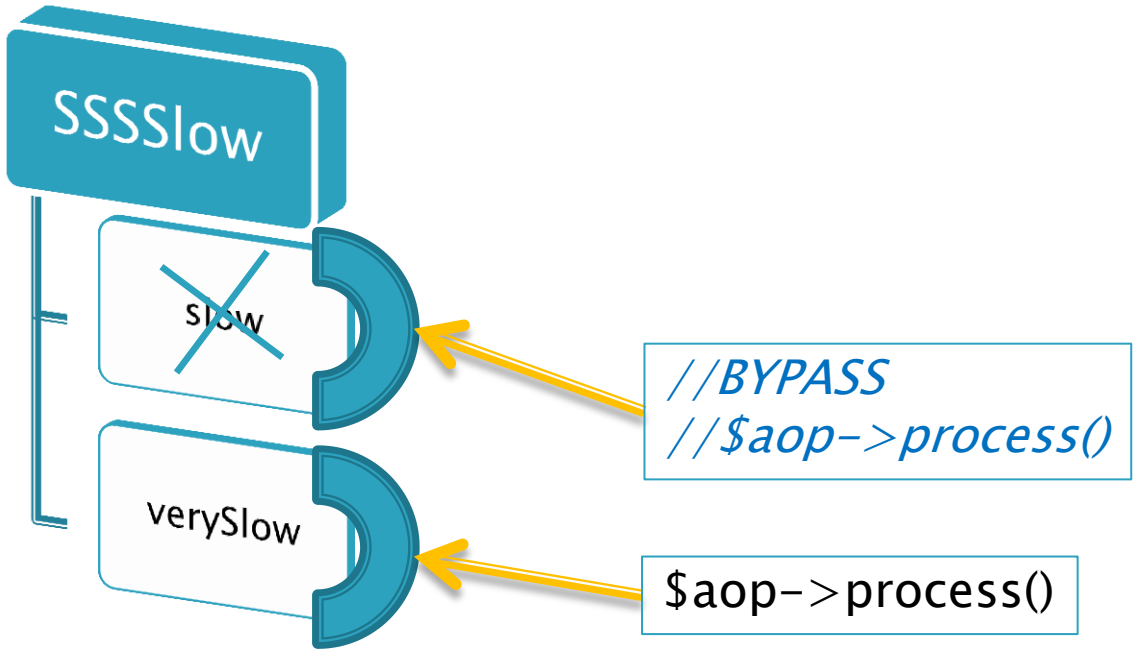
Un cas avec around ?

Par exemple une gestion de cache

Si c'est long, fait le qu'une fois !

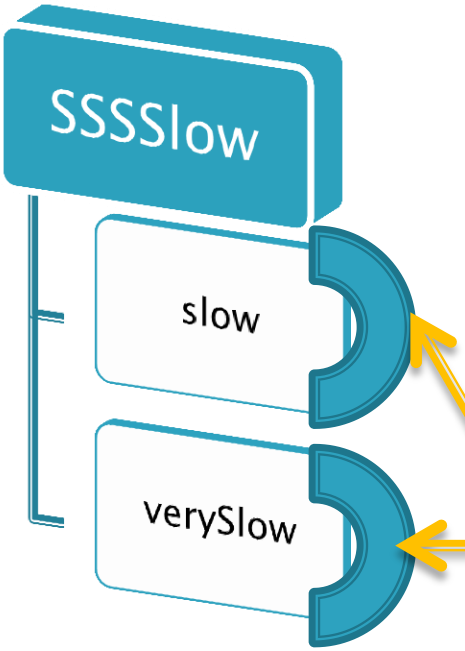


Around maîtrise l'exécution finale



Avec AOP !

- « Il faudrait mettre en cache les méthodes de SSSSlow »




```
$cache = function (AopJoinpoint $jp) {
    $cache_file = '/tmp/'.$jp->getMethod().'|'
                        .$jp->getArguments[0];
    if (file_exists($cache_file))
        return file_get_contents($cache_file);
    $toReturn = $jp->process();
    file_put_contents($cache_file, $toReturn);
}
```

Advice

```
aop_add_around ('SSSSlow::*()',  $cache);
```

Aspect

D'autres usages à l'AOP ?

Changement d'API, Débogage, Quickfix, …

Changement d'api

- « ce qui serait bien »

```
<?php

Aop_add_after('file_get_contents', function ($aop) {
        if ($aop->getReturnedValue()===false) {
                throw new \Exception ('File not exist');
        }
});
```

Debug

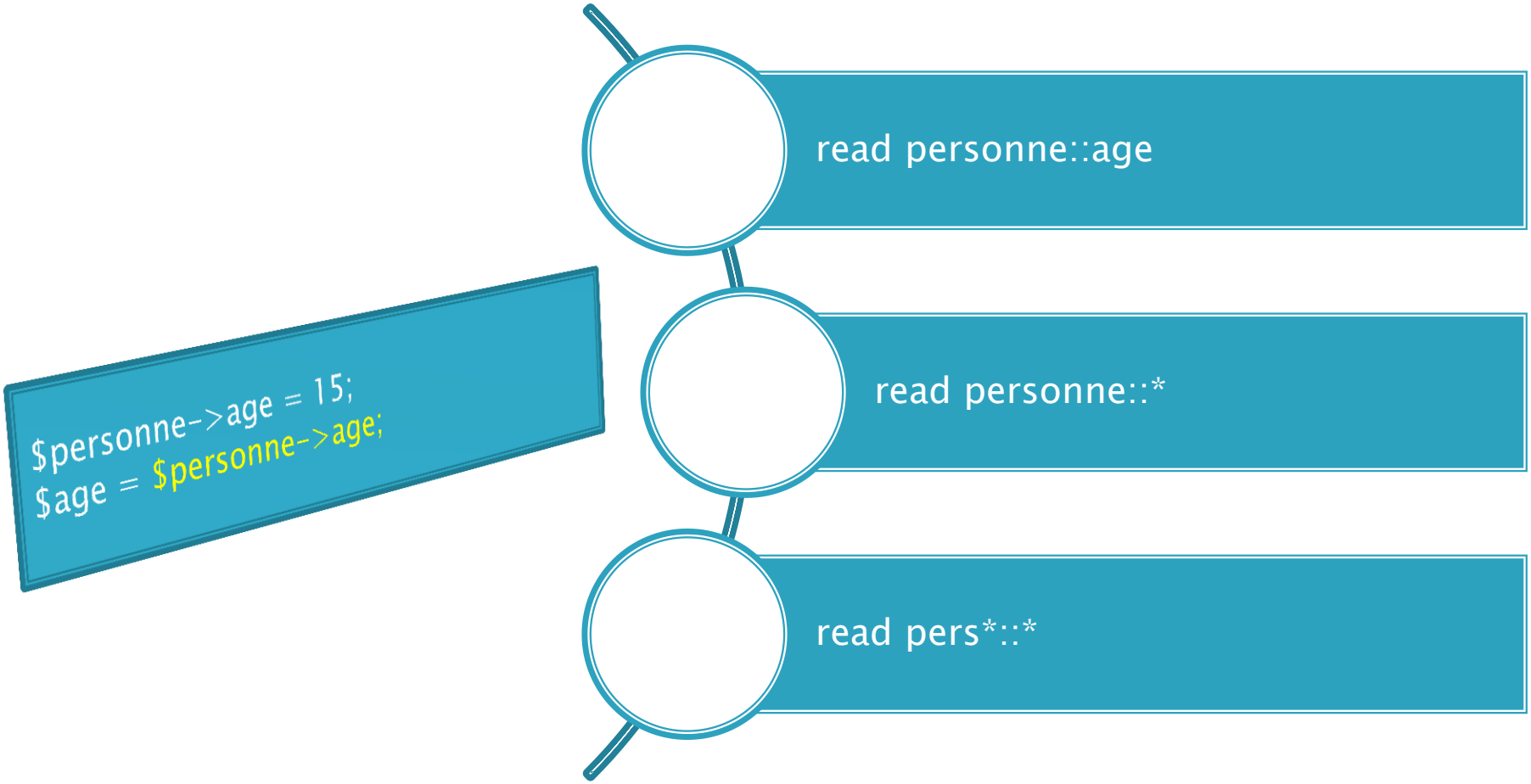
- Savoir si votre méthode est appelé

```
<?php
Aop_add_before (‘maClass::maMethode()’, function ($aop)
{
        echo « maMethode est appelé »;
});
```

On a tout vu ?

Non, on peut aussi gérer les propriétés ☺

Dis moi si qq'un te regarde

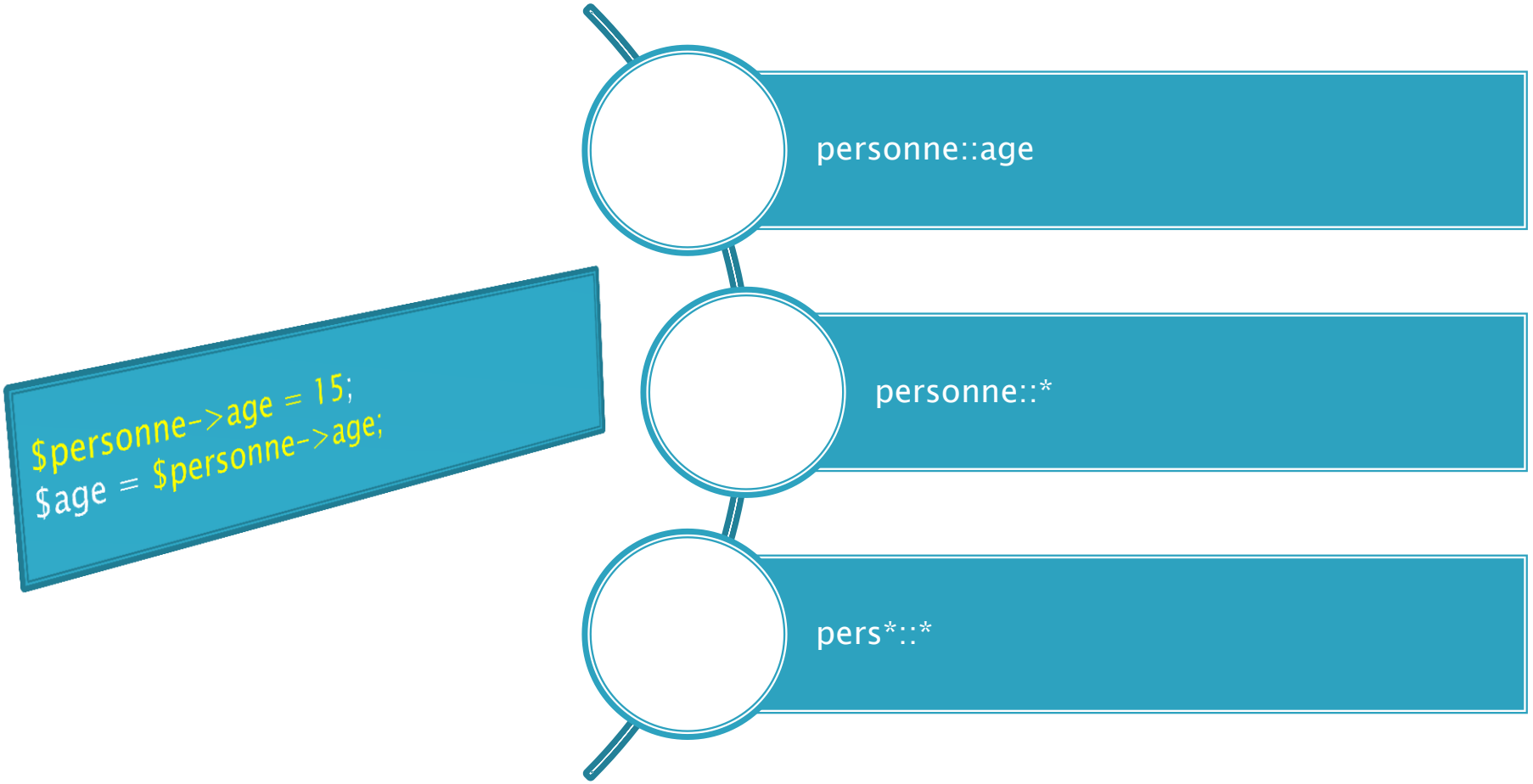


Qui a touché à ça ?

```
$personne->age = 15;
$age = $personne->age;
```

- write personne::age
- write personne::*
- write pers*::*

Je connais tous tes faits et gestes



Dans le futur ?

Stabilisation / Evolutions

Par ordre de priorité

- Documentation complète
- Phase de beta + RC
- Performances
- Extension aux Pointcut
- aop_add ?
- Introduction

Et toutes les idées que vous voudrez bien donner…

D’autres questions ?