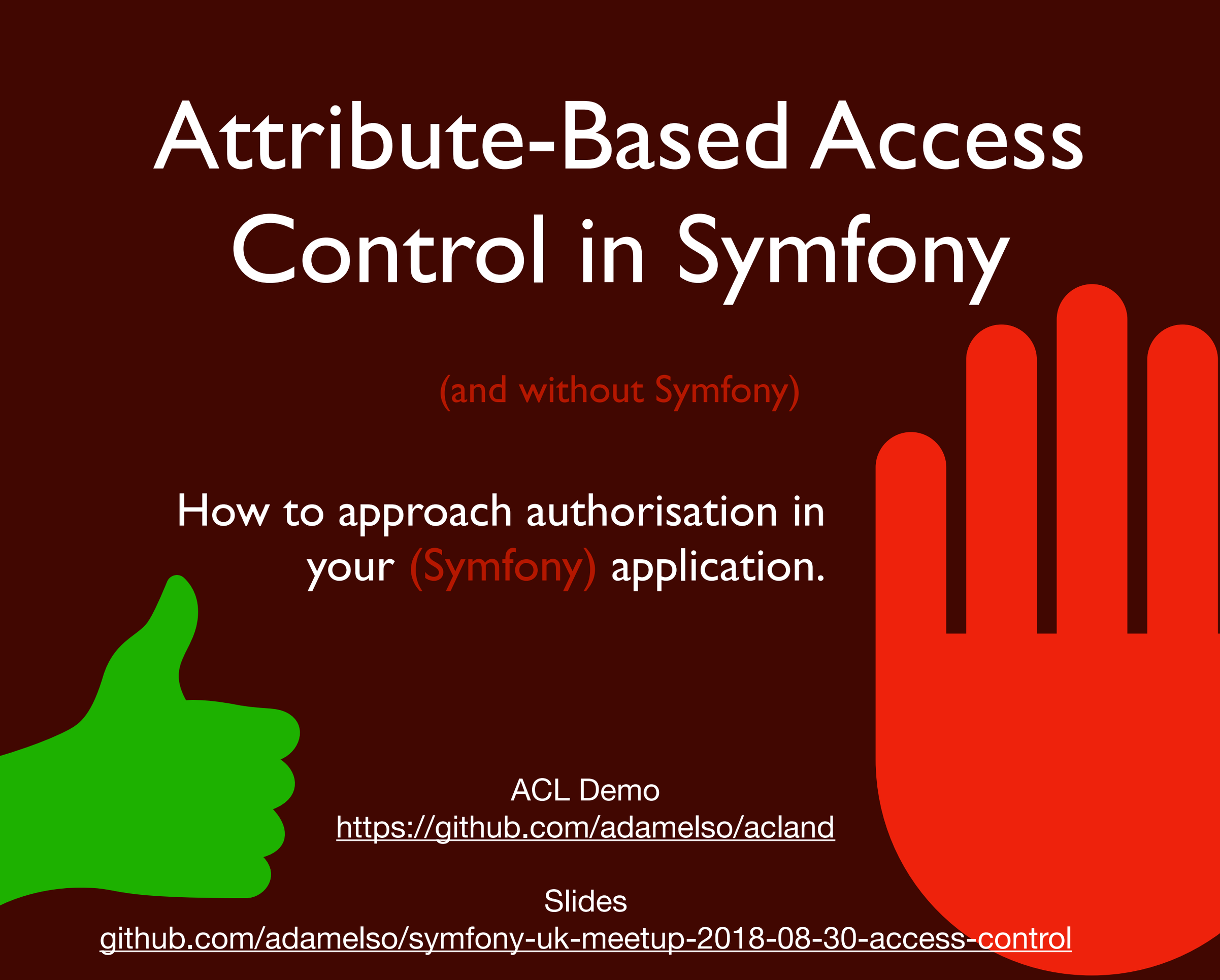
Attribute-Based Access Control in Symfony

(and without Symfony)



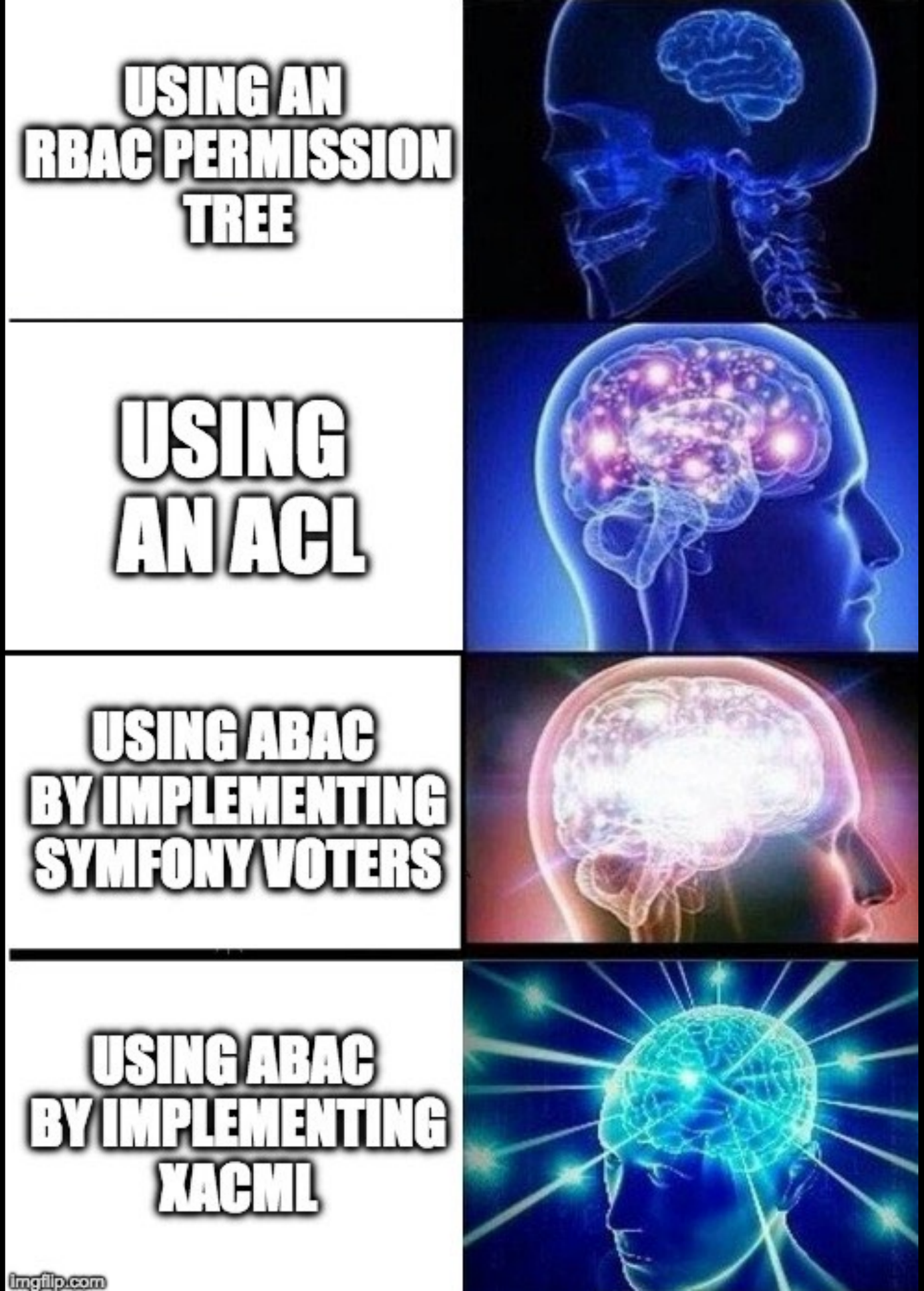
How to approach authorisation in your (Symfony) application.

ACL Demo
https://github.com/adamelso/acland

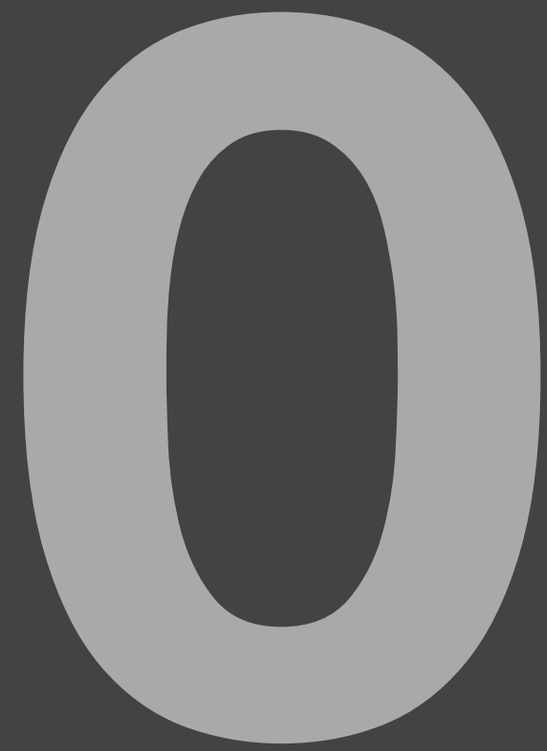
Slides
github.com/adamelso/symfony-uk-meetup-2018-08-30-access-control

This presentation is split into 4 parts

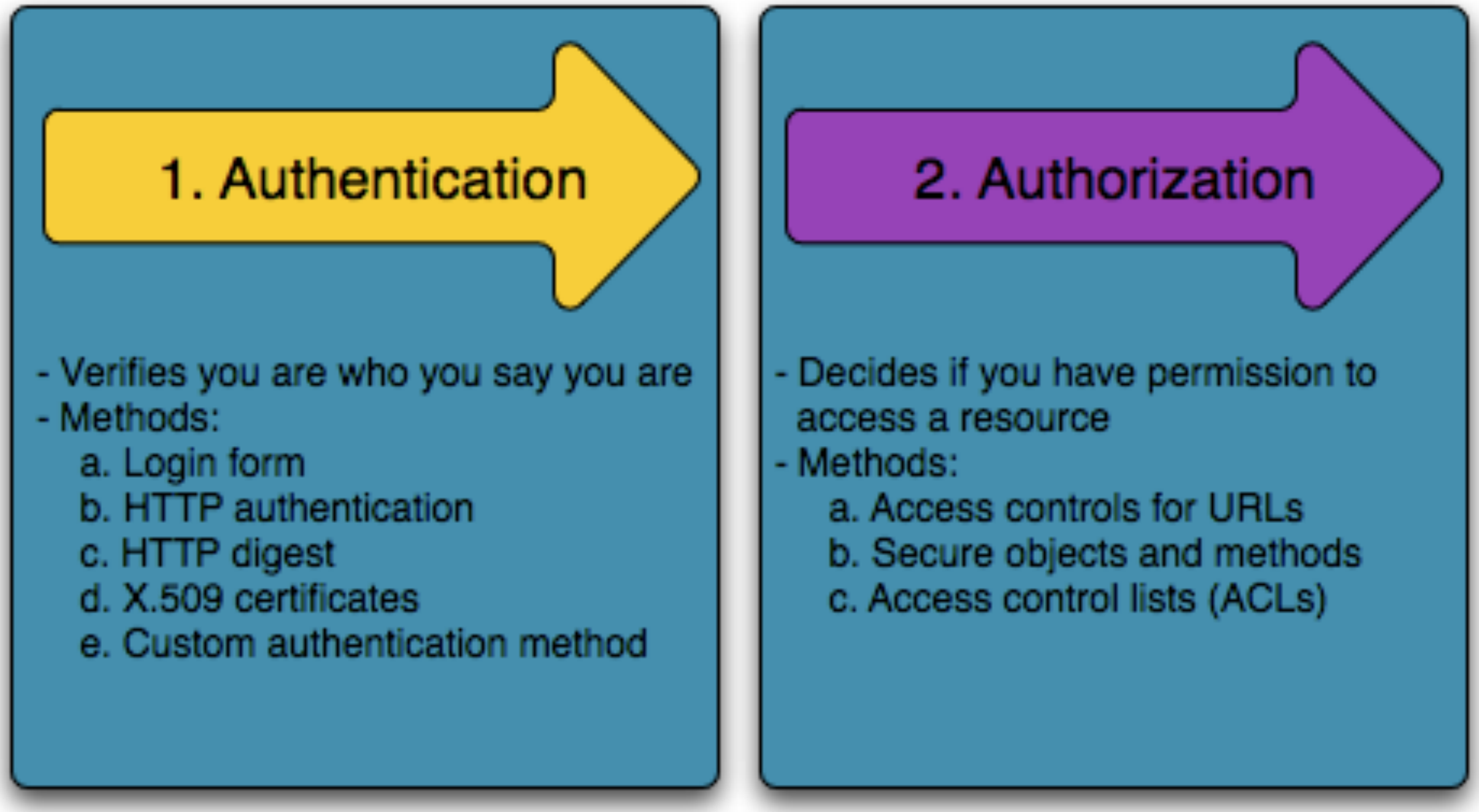
…maybe 5.

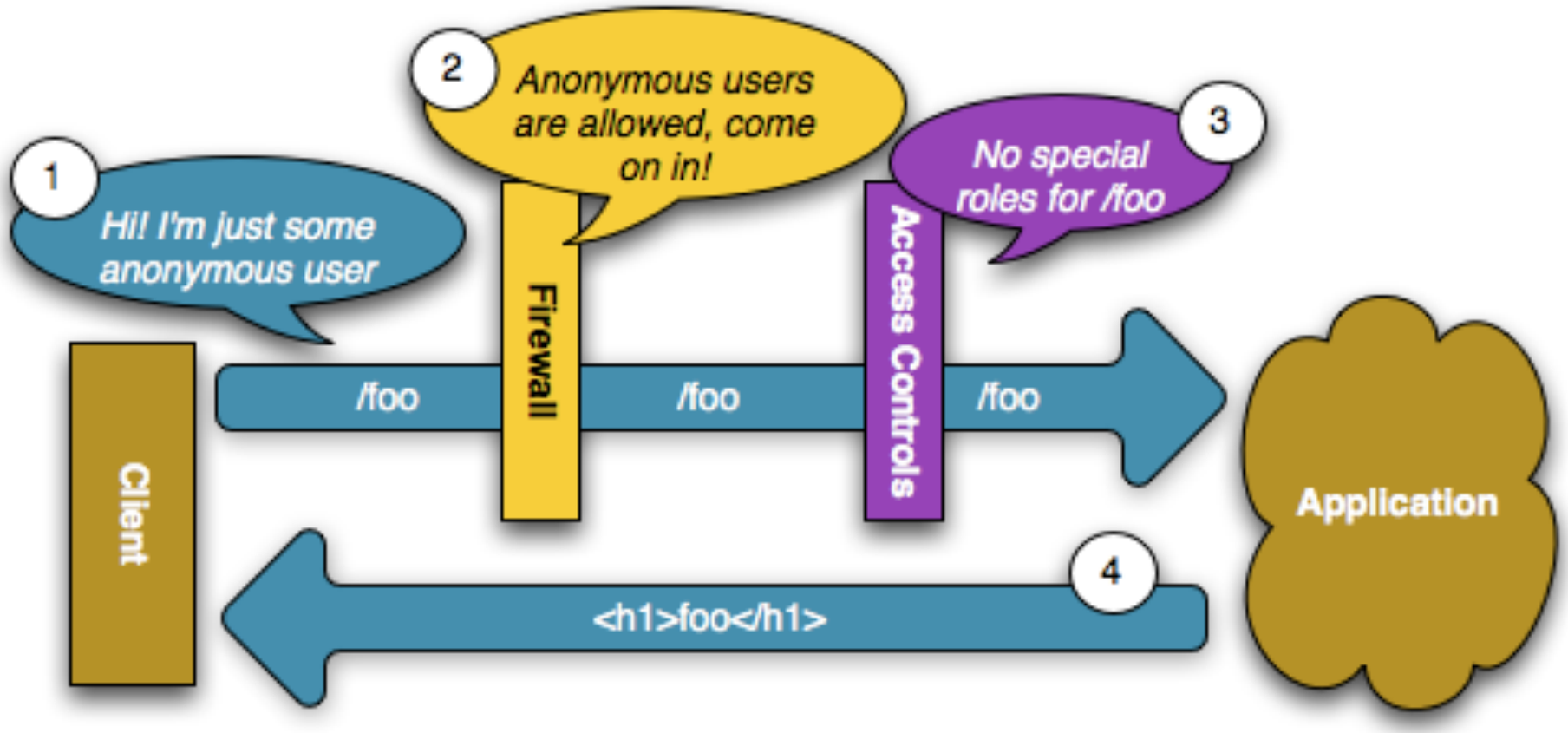


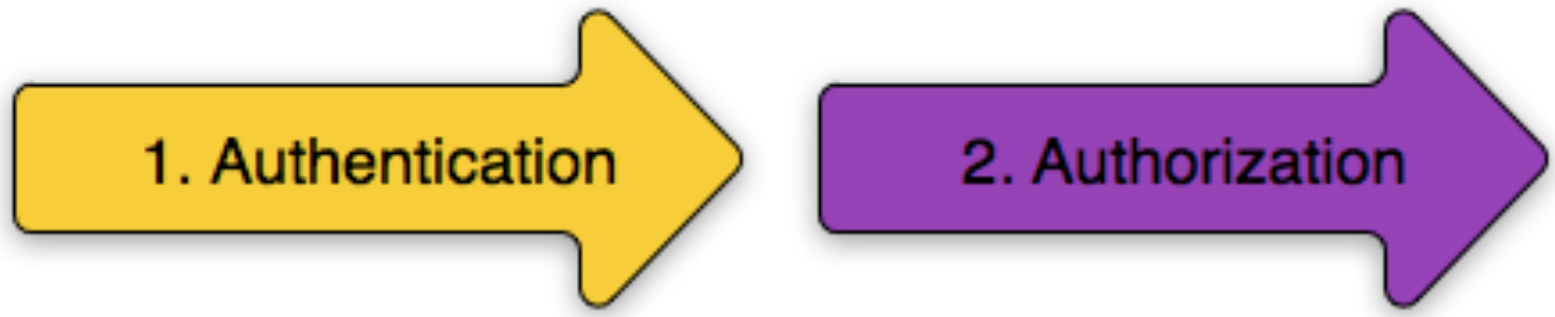
Out-of-the-box
Symfony
SecurityBundle
Access Control



There are 2 steps to securing a resource.





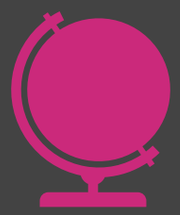


Authentication is enforced with **Firewalls**

Authorisation is enforced with **Access Controls**

That’s easy!

```yaml
# Easy way to control access for large sections of your site
# Note: Only the *first* access control that matches will be used
access_control:
    - { path: ^/admin, roles: ROLE_ADMIN }
    - { path: ^/profile, roles: ROLE_USER }

role_hierarchy:
    ROLE_ADMIN: [ROLE_USER, ROLE_ALLOWED_TO_SWITCH]
```



Path

String, Regular Expression

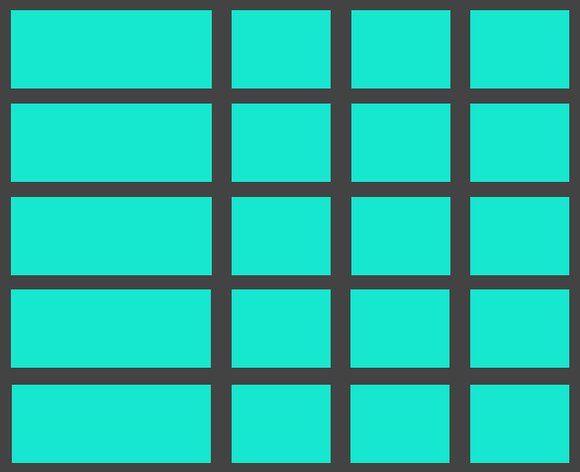
Role

String, RoleInterface, Hierarchical

…but not finely-grained.

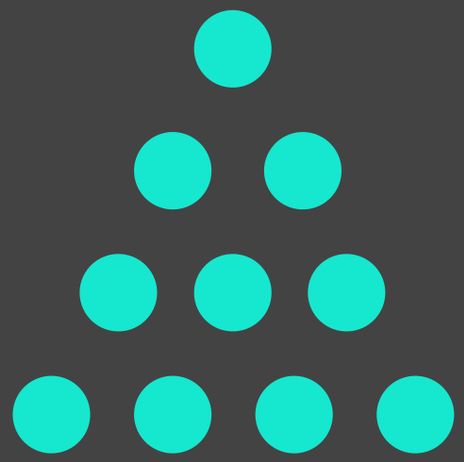
```
role_hierarchy:
    ROLE_ADMIN_FOR_ALL_JOURNALS:  [ROLE_ADMIN_FOR_JOURNAL_1, ROLE_ADMIN_FOR_JOURNAL_2, ROLE_ADMIN
    ROLE_ADMIN_FOR_SOME_JOURNALS: [ROLE_ADMIN_FOR_JOURNAL_2, ROLE_ADMIN_FOR_JOURNAL_4]

access_control:
    - { path: ^/admin/journal/1, role: ROLE_ADMIN_FOR_JOURNAL_1 }
    - { path: ^/admin/journal/2, role: ROLE_ADMIN_FOR_JOURNAL_2 }
    - { path: ^/admin/journal/3, role: ROLE_ADMIN_FOR_JOURNAL_3 }
    - { path: ^/admin/journal/4, role: ROLE_ADMIN_FOR_JOURNAL_4 }
```

There are many types of access control depending on your needs



Access Control Lists

ACL

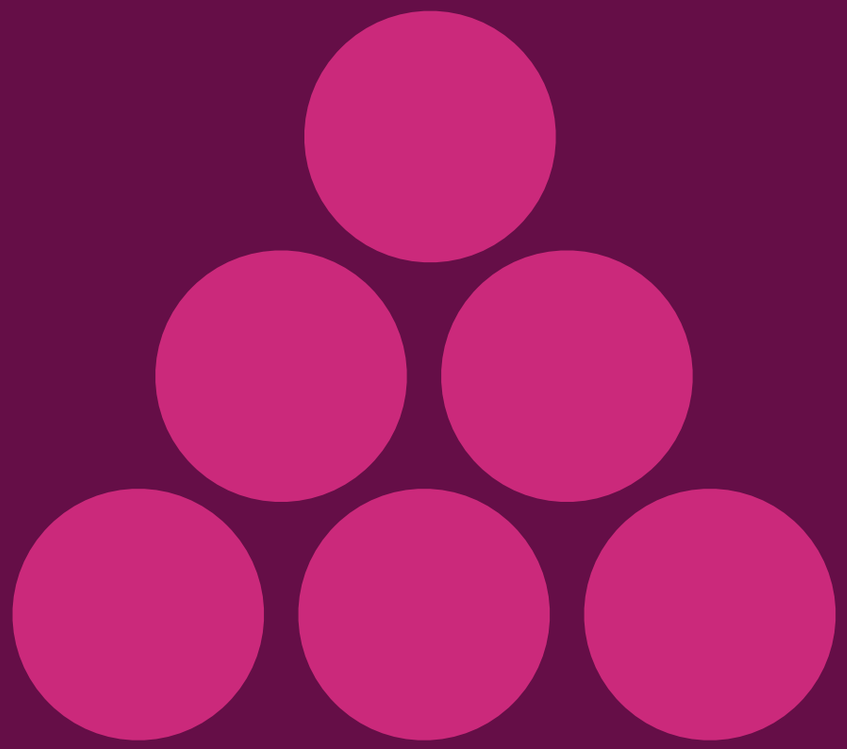


Role-Based
Access Control

RBAC

Attribute-Based
Access Control

ABAC



1

RBAC

Implementing RBAC:

> “ Probably the most common variant of authorization is **role-based access control (RBAC)**. As the name implies,
>
> • Users are assigned **roles**
>
> • Roles are assigned **permissions**.
>
> • Users inherit the permission for any roles they have been assigned.
>
> • Actions are validated for permissions.

https://martinfowler.com/articles/web-security-basics.html

Users have roles

Bob

USER

Associate Editor

ROLE

Roles have permissions

Users inherit the permissions for any roles they have been assigned

Actions are validated for permissions

Reject Article Submission

Leave Feedback

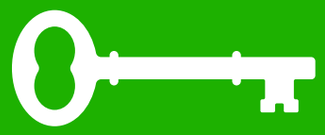
Approve Article Submission

Bob

USER

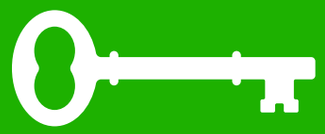
Associate Editor

ROLE



Reject Article Submission

PERMISSION



Approve Article Submission

PERMISSION

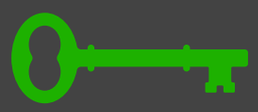
Reject Article Submission

Leave Feedback

Approve Article Submission

Action

Code



Permission

String

Role

String, RoleInterface, Hierarchical

```php
public function approve(Submission $submission, $comment)
{
    if (! $this->authorizationChecker->isGranted('veruscript.submission.approve')) {
        throw new AccessDeniedException();
    }

    $submission->approve();
    $submission->addFeedback($comment);
}
```

System Admin

ROLE

Journal Admin

ROLE

In some cases, roles inherit the permissions from other roles via a hierarchy…

Author

ROLE

Editor-in-Chief

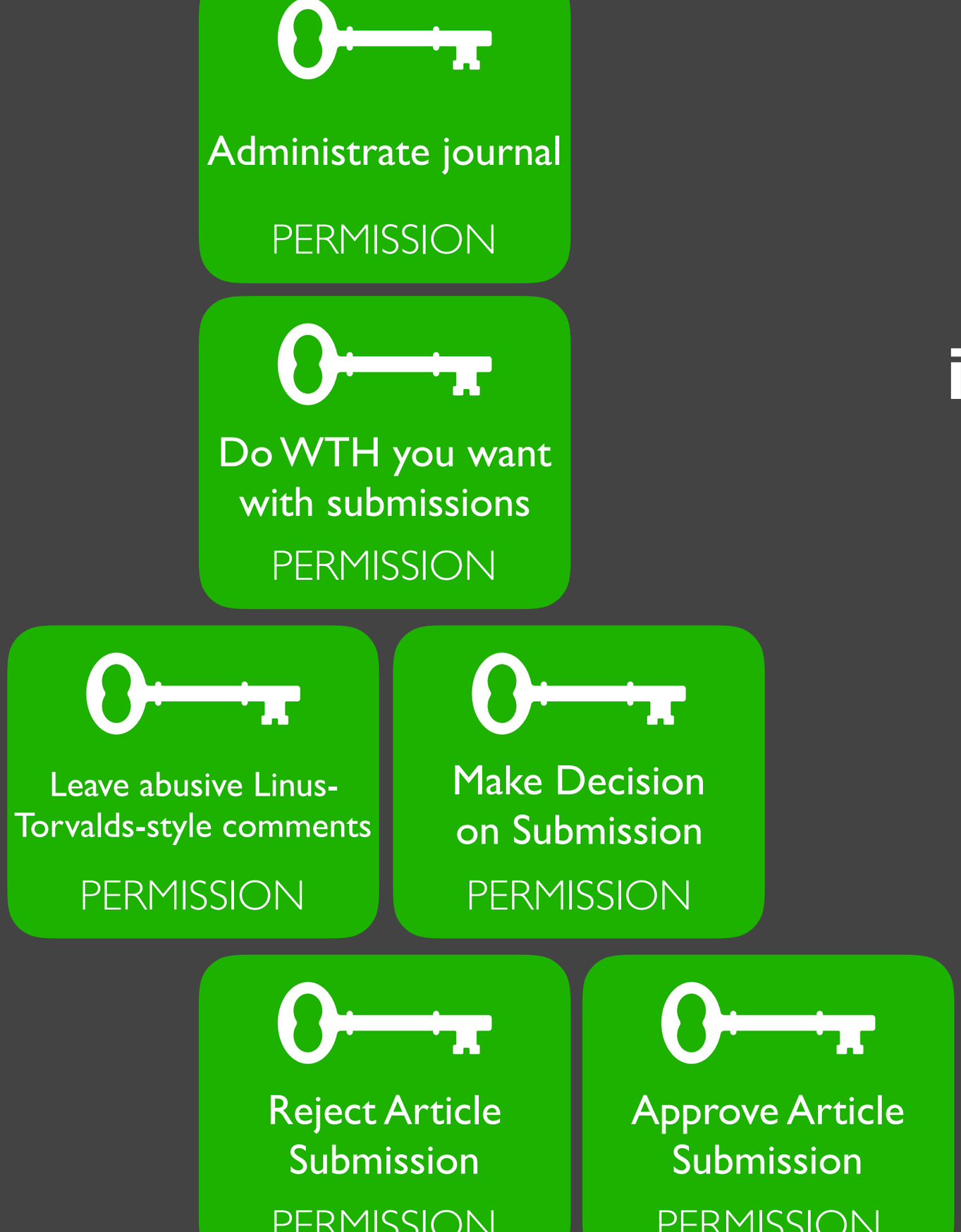
ROLE

Associate Editor

ROLE

Reviewer

ROLE



…and/or permissions inherit the permissions from other roles via a hierarchy.

Like Sylius RBAC

```php
use Sylius\Component\Rbac\Model\Role;
use Sylius\Component\Rbac\Model\Permission;

// Assume that in your application you want to have two roles and some permissions to manage cu

// Let's start with creating root permission.
$manageCustomer = new Permission();
$manageCustomer->setCode('sylius.manage.customer');
$manageCustomer->setDescription('Manage customers');

// Next create more specific permissions.
$deleteCustomer = new Permission();
$deleteCustomer->setCode('sylius.delete.customer');
$deleteCustomer->setDescription('Delete customer');

$createCustomer = new Permission();
$createCustomer->setCode('sylius.create.customer');
$createCustomer->setDescription('Create customer');

// Now take care of permission inheritance.
$manageCustomer->addChild($deleteCustomer);

$manageCustomer->addChild($createCustomer);
//Great! Now we have Customer Manager permission which inherits above permissions.

// Roles are defined as tree structure as well, it is the same rule as with permissions.
// Let's start with our root role.
$administrator = new Role();
$administrator->setCode('administrator');
$administrator->setName('Administrator');

$customerManager = new Role();
$customerManager->setCode('customer_manager');
$customerManager->setName('Customer Manager');

// Take care of role inheritance.
$administrator->addChild($customerManager);

$customerManager->addPermission($manageCustomer);
```

Consider RBAC When

- Permissions are relatively static.
- Roles in your policies actually map reasonably to roles within your domain, rather than feeling like contrived aggregations of permissions.
- There isn't a terribly large number of permutations of permission, and therefore roles that will have to be maintained.
- You have no compelling reason to use one of the other options.

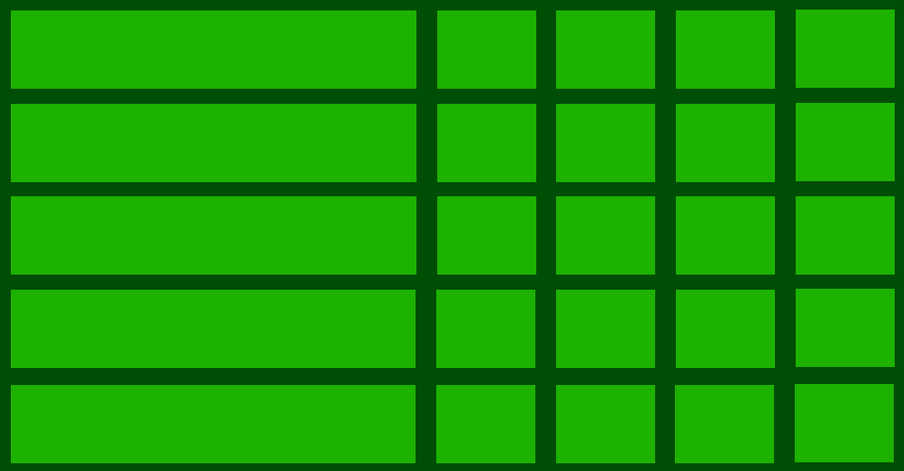
https://martinfowler.com/articles/web-security-basics.html

Shortcomings of RBAC

1. Cannot grant permissions per-resource, only by resource type.

2. Does not scope resource properties.

```php
public function canReinstateShipments(Order $order)
{
    return
        $this->authorizationChecker->isGranted('reiss.order.abort_shipping')
        && $order->getState() === Order::STATE_SHIPPED
        && $order->getShippingState() === Shipment::STATE_CANCELLED
        && $order->isShipped();
}
```



ACL

(Symfony ACL)

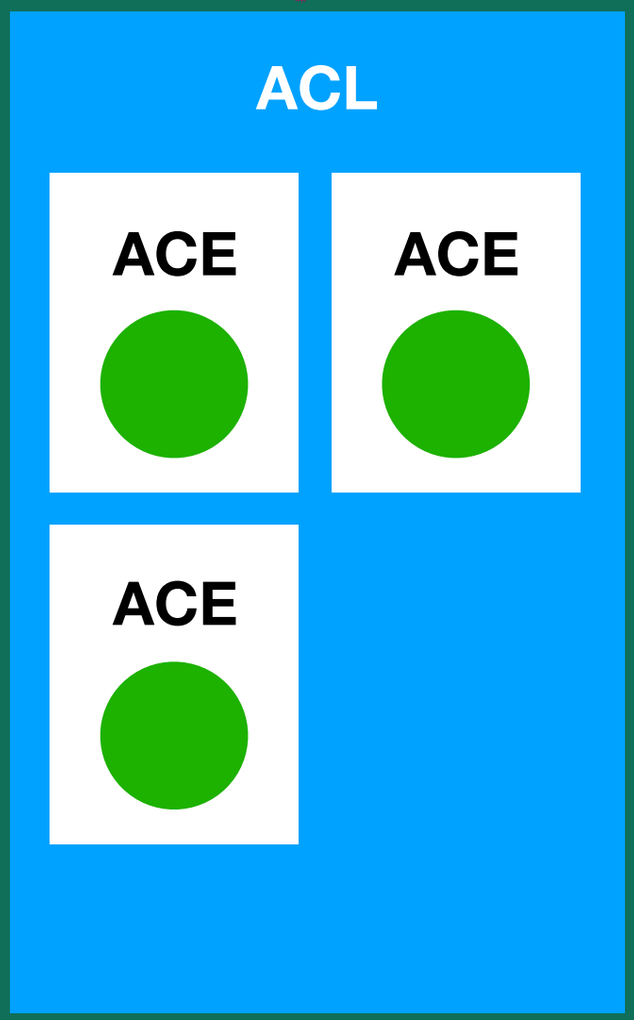
“ How to Use Access Control Lists (ACLs):

In complex applications, you will often face the problem that access decisions cannot only be based on the person (`Token`) who is requesting access, but also involve a domain object that access is being requested for. This is where the ACL system comes in.

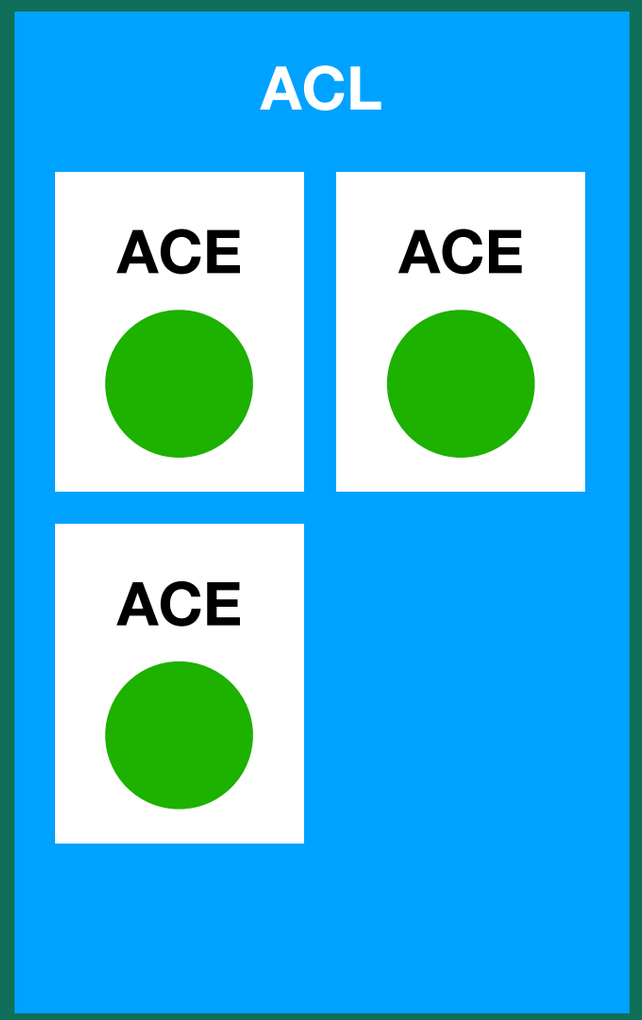
https://symfony.com/doc/3.4/security/acl.html

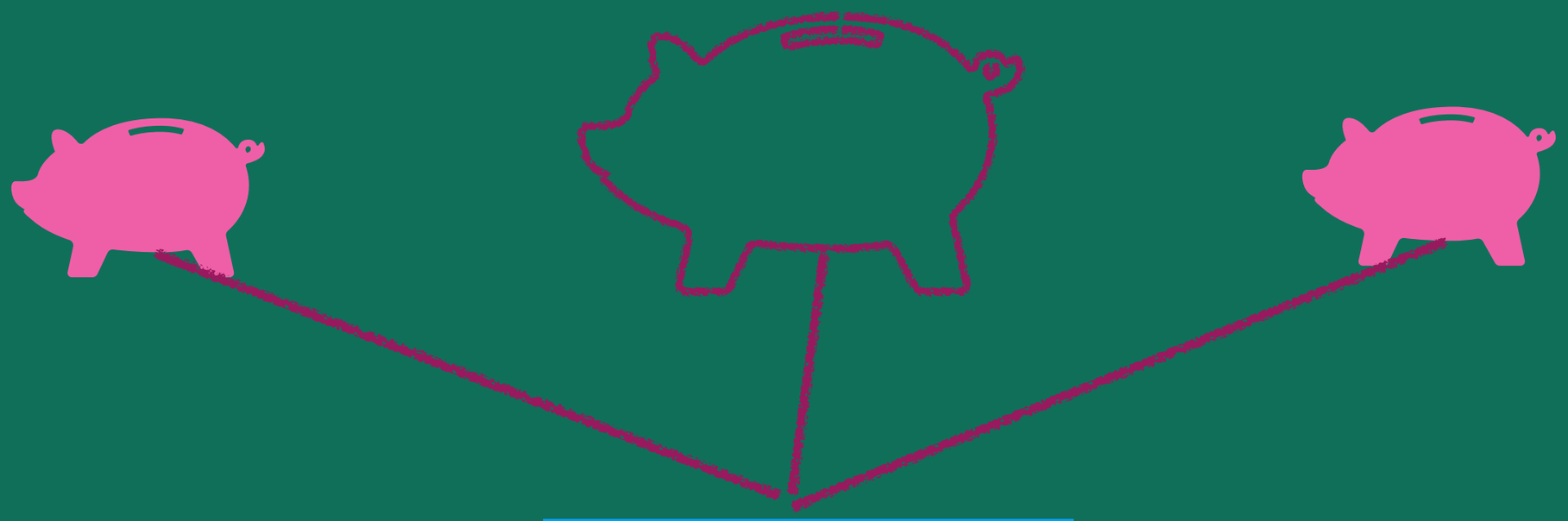
Access Control Lists (ACL)

First, check if the domain object requested has an associated ACL.

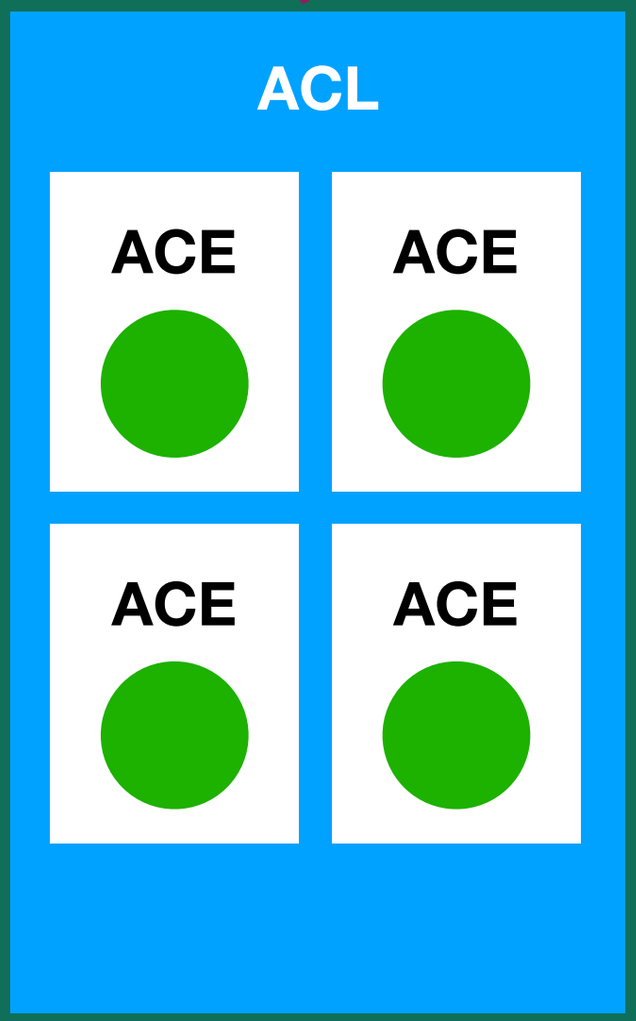


Each ACL contains one or more Access Control Entries (ACEs) that defines specific permissions for the ACL's resource.





Second, check the domain as a whole.



ACLs can be associated with both objects (entities) and domains (classnames).

Otherwise, deny access.

Using the Symfony ACL

1. Install Bundle

```
$ composer require symfony/acl-bundle
```

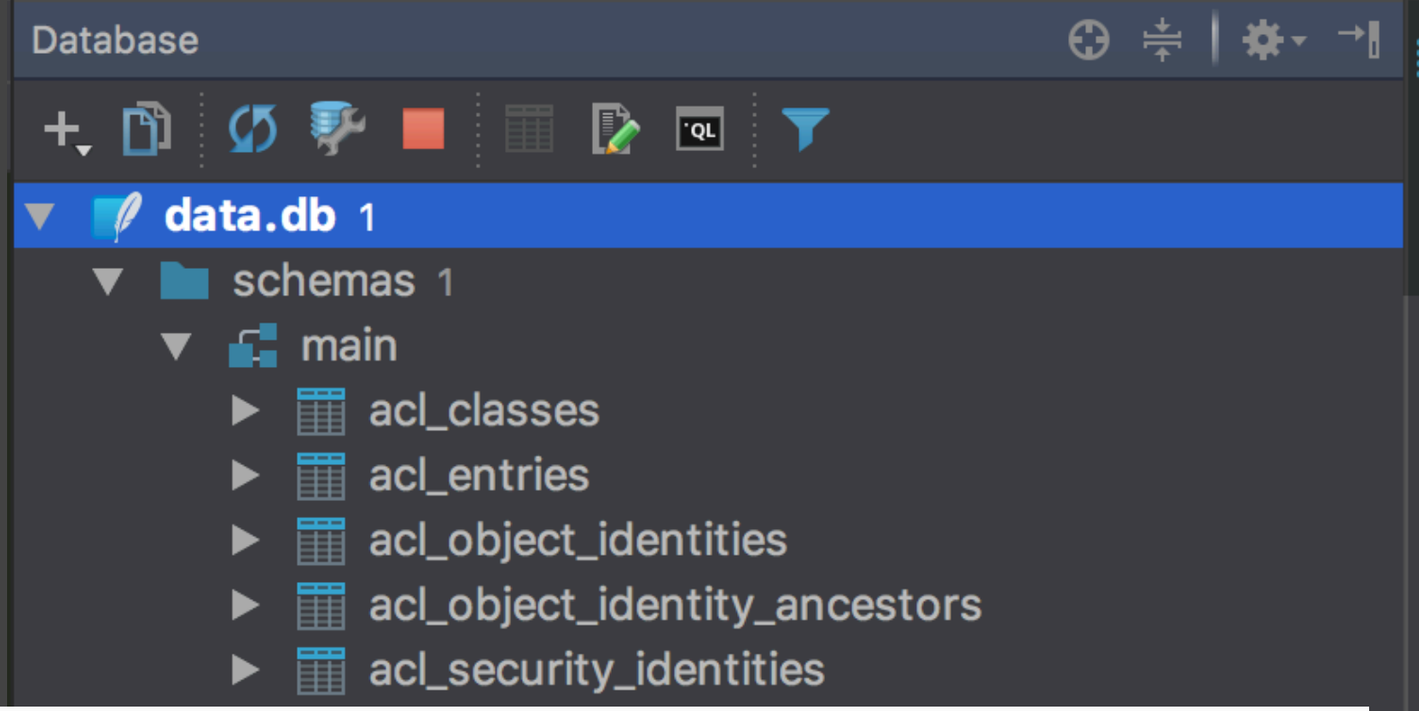
2. Configure

acl.yaml

```yaml
acl:
    connection: default

```

3. Initialise

```
adam@AF16 ~/Sites/acland
[$ bin/console acl:init
ACL tables have been initialized successfully.
```



Database Table Structure ¶

The default implementation uses five database tables as listed below. The tables are ordered from least rows to most rows in a typical application:

- *acl_security_identities*: This table records all security identities (SID) which hold ACEs. The default implementation ships with two security identities: `RoleSecurityIdentity` and `UserSecurityIdentity`.
- *acl_classes*: This table maps class names to a unique ID which can be referenced from other tables.
- *acl_object_identities*: Each row in this table represents a single domain object instance.
- *acl_object_identity_ancestors*: This table allows all the ancestors of an ACL to be determined in a very efficient way.
- *acl_entries*: This table contains all ACEs. This is typically the table with the most rows. It can contain tens of millions without significantly impacting performance.

acl_entries table

acl_entries [data.db]

9 rows | Tx: Auto | Tab-se...d (TSV) | DDL | View Query

	id	class_id	object_identity_id	security_identity_id	field_name	ace_order	mask	granting	granting_strategy	audit_success	audit_failure
1	1	1	1	1	<null>	0	128	☑	all	☐	☐
2	2	1	2	2	<null>	0	128	☑	all	☐	☐
3	3	1	3	1	<null>	3	128	☑	all	☐	☐
4	4	1	4	2	<null>	0	128	☑	all	☐	☐
5	5	1	3	2	<null>	2	29	☑	all	☐	☐
6	6	1	3	2	<null>	1	29	☑	all	☐	☐
7	7	1	3	2	<null>	0	29	☑	all	☐	☐
8	8	1	<null>	2	<null>	1	29	☑	all	☐	☐
9	9	1	<null>	2	<null>	0	29	☑	all	☐	☐

- id
- class
- object identity
- security identity
- field name
- ACE order
- mask
- is granting
- granting strategy
- audit success
- audit failure

```php
/**
 * @Route("/", name="chat")
 */
class ChatController extends Controller
{
    public function __invoke(Request $request)
    {
        $em = $this->getDoctrine()
            ->getManagerForClass( class: Message::class);

        $message = new Message();

        $form = $this->createFormBuilder($message)
            ->add( child: 'content', type: TextareaType::class, [
                'label' => 'Write your message'
            ])
            ->add( child: 'submit', type: SubmitType::class)
            ->getForm();

        if ($request->isMethod( method: 'POST')) {
            $message->setAuthor($this->getUser());
            $message->setCreatedAt(new \DateTime());

            $form->handleRequest($request);

            if ($form->isSubmitted() && $form->isValid()) {
                $this->postMessage($message);
            }
        }

        $repository = $em->getRepository( className: Message::class);

        return $this->render( view: 'chat', [
            'messages'  => $repository->findAll(),
            'chatbox'   => $form->createView(),
        ]);
    }
```

```php
private function postMessage(Message $message)
{
    $em = $this->getDoctrine()->getManagerForClass( class: Message::class);

    $em->persist($message);
    $em->flush();

    // creating the ACL

    /** @var MutableAclProviderInterface $aclProvider */
    $aclProvider     = $this->get('security.acl.provider');
    $objectIdentity = ObjectIdentity::fromDomainObject($message);
    $acl            = $aclProvider->createAcl($objectIdentity);

    // retrieving the security identity of the currently logged-in user

    $tokenStorage     = $this->get('security.token_storage');
    $user             = $tokenStorage->getToken()->getUser();
    $securityIdentity = UserSecurityIdentity::fromAccount($user);

    // grant owner access

    $acl->insertObjectAce($securityIdentity,  mask: MaskBuilder::MASK_OWNER);
    $aclProvider->updateAcl($acl);
}
```

Attribute	Intended Meaning	Integer Bitmasks
VIEW	Whether someone is allowed to view the domain object.	VIEW, EDIT, OPERATOR, MASTER, or OWNER
EDIT	Whether someone is allowed to make changes to the domain object.	EDIT, OPERATOR, MASTER, or OWNER
CREATE	Whether someone is allowed to create the domain object.	CREATE, OPERATOR, MASTER, or OWNER
DELETE	Whether someone is allowed to delete the domain object.	DELETE, OPERATOR, MASTER, or OWNER
UNDELETE	Whether someone is allowed to restore a previously deleted domain object.	UNDELETE, OPERATOR, MASTER, or OWNER
OPERATOR	Whether someone is allowed to perform all of the above actions.	OPERATOR, MASTER, or OWNER
MASTER	Whether someone is allowed to perform all of the above actions, and in addition is allowed to grant any of the above permissions to others.	MASTER, or OWNER
OWNER	Whether someone owns the domain object. An owner can perform any of the above actions *and* grant master and owner permissions.	OWNER

```php
/**
 * @Route("/message/{id}", name="edit")
 * @ParamConverter("message", class="App\Entity\Message")
 */
class EditMessageController extends Controller
{
    public function __invoke(Message $message)
    {
        /** @var AuthorizationCheckerInterface $authorizationChecker */
        $authorizationChecker = $this->get('security.authorization_checker');

        // check for edit access
        if (false === $authorizationChecker->isGranted( attributes: 'EDIT', $message)) {
            throw new AccessDeniedException();
        }

        return new Response( content: 'ok');
    }
}
```

Chat

[Edit] *20:02* | adamelso

Hello there

[Edit] *20:03* | boss

Hi can I help you

[Edit] *20:04* | adamelso

yeah can I have an 8-ball?

[Edit] *20:04* | boss

it aint that kinda place, mate

Write your message

Submit

Chat

[Edit] *20:02* | adamelso

Hello there

[Edit] *20:03* | boss

Hi can I help you

[Edit] *20:04* | adamelso

yeah can I have an 8-ball?

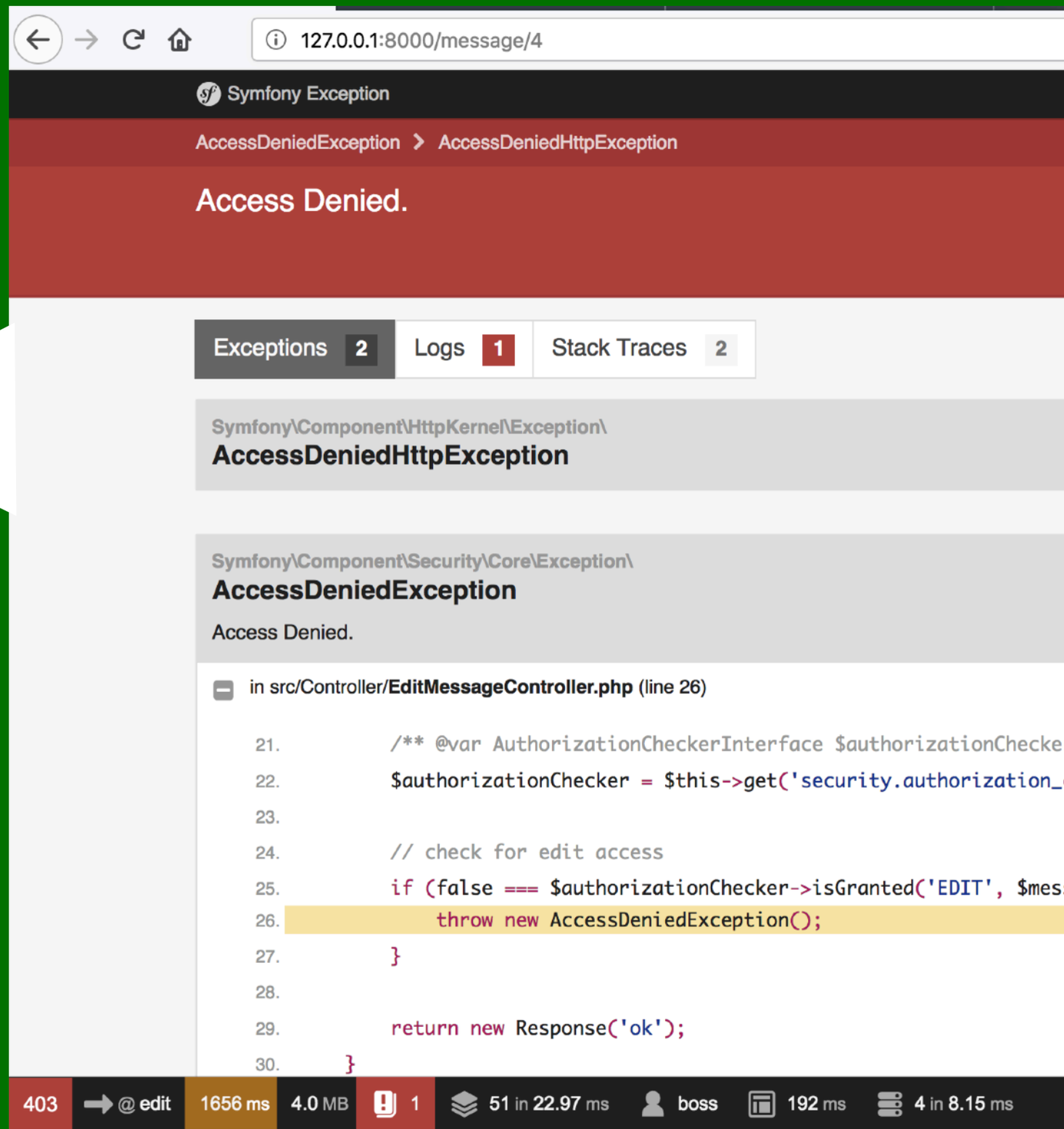
[Edit] *20:04* | boss

it aint that kinda place, mate

Write your message

Submit

As the boss of this website
I should be able to edit a particular message posted
In order to moderate the content



```php
protected function execute(InputInterface $i, OutputInterface $o)
{
    $username = $i->getArgument( name: 'username');
    $messageId = $i->getArgument( name: 'message-id');

    $user = $this->userManager->findUserByUsernameOrEmail($username);

    if (! $user) {
        $o->writeln( messages: "<error>Who the hell is {$username}?!</error>");
        return 1;
    }

    $repository = $this->doctrine->getRepository( persistentObject: Message::class);
    $message = $repository->find($messageId);

    if (! $message) {
        $o->writeln( messages: "<error>No message with ID {$username}.</error>");
        return 1;
    }

    $this->grantAllPermissionsOnMessageForUser($user, $message);

    $o->writeln( messages: "<info>{$username} has now been granted all permissions on message #{$messageId}.</info>");

    return 0;
}
```

```php
private function grantAllPermissionsOnMessageForUser(UserInterface $user, Message $message)
{
    $bitmask = $this->buildBossPermissionsMask();

    $objectIdentity        = ObjectIdentity::fromDomainObject($message);
    $userSecurityIdentity = UserSecurityIdentity::fromAccount($user);

    try {
        /** @var Acl $acl */
        $acl = $this->aclProvider->findAcl($objectIdentity);

    } catch (AclNotFoundException $e) {
        $acl = $this->aclProvider->createAcl($objectIdentity);
    }

    $acl->insertObjectAce($userSecurityIdentity, $bitmask);
    $this->aclProvider->updateAcl($acl);
}

private function buildBossPermissionsMask(): int
{
    $builder = new MaskBuilder();

    $builder->add('view');
    $builder->add('edit');
    $builder->add('delete');
    $builder->add('undelete');

    return $builder->get();
}
}
```

```
adam@AF16 ~/Sites/acland
[$ bin/console app:grant-permissions-on-message boss 4
boss has now been granted all permissions on message #4.
```

Chat

[Edit] *20:02* | adamelso

Hello there

[Edit] *20:03* | boss

Hi can I help you

[Edit] *20:04* | adamelso

yeah can I have an 8-ball?

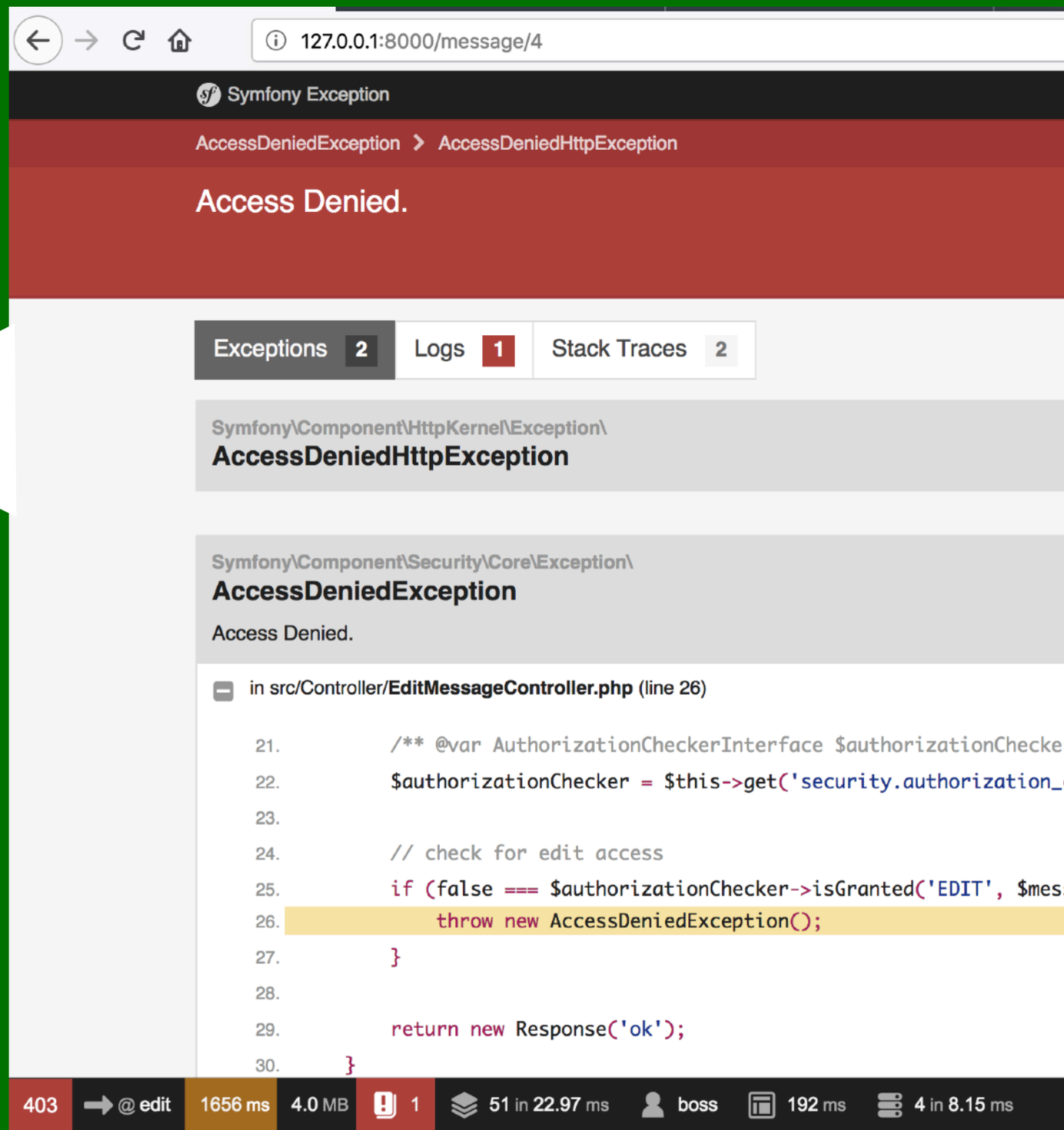
[Edit] *20:04* | boss

it aint that kinda place, mate

Write your message

Submit

As the boss of this website
I should be able to edit
~~a particular message~~ **all messages posted**
In order to moderate the content



```php
class PermissionsAdministrationPoint
{
    /**
     * @var PermissionsBuilder
     */
    private $permissionsBuilder;

    /**
     * @var MutableAclProviderInterface
     */
    private $aclProvider;

    public function __construct(PermissionsBuilder $permissionsBuilder, MutableAclProviderInterface $aclProvider)
    {
        $this->permissionsBuilder = $permissionsBuilder;
        $this->aclProvider = $aclProvider;
    }

    public function grantAllOnEveryMessageTo(UserInterface $user)
    {
        $bitmask = $this->permissionsBuilder->grantAll();

        // Notice we use 'class', and not an entity ID.
        $objectIdentity       = new ObjectIdentity( identifier: 'class',  type: Message::class);
        $userSecurityIdentity = UserSecurityIdentity::fromAccount($user);

        try {
            /** @var Acl $acl */
            $acl = $this->aclProvider->findAcl($objectIdentity);

        } catch (AclNotFoundException $e) {
            $acl = $this->aclProvider->createAcl($objectIdentity);
        }

        // Notice we use 'insertClassAce', instead of 'insertObjectAce'.
        $acl->insertClassAce($userSecurityIdentity, $bitmask);

        $this->aclProvider->updateAcl($acl);
    }
}
```

```php
    public function __construct(UserManagerInterface $userManager, RegistryInterface $doctrine, PermissionsAdministrationPoint
    {
        parent::__construct( name: 'app:grant-permissions-on-everything');
        $this->userManager = $userManager;
        $this->doctrine = $doctrine;
        $this->permissionsAdministrationPoint = $permissionsAdministration;
    }

    protected function configure()
    {
        $this->addArgument( name: 'username',  mode: InputArgument::REQUIRED);
    }

    protected function execute(InputInterface $i, OutputInterface $o)
    {
        $username = $i->getArgument( name: 'username');

        $user = $this->userManager->findUserByUsernameOrEmail($username);

        if (! $user) {
            $o->writeln( messages: "<error>Who the hell is {$username}?!</error>");
            return 1;
        }

        $this->permissionsAdministrationPoint->grantAllOnEveryMessageTo($user);

        $o->writeln( messages: "<info>{$username} has now been granted all permissions to every message.</info>");

        return 0;
    }
}
```

```
adam@AF16 ~/Sites/acland
[$ bin/console app:grant-permissions-on-everything boss
boss has now been granted all permissions to every message.
```

“ Alternatives to ACLs

Using [ACLs] isn't trivial, and for simpler use cases, it may be overkill. If your permission logic could be described by just writing some code (e.g. to check if a Blog is owned by the current User), then consider using *voters*. A voter is passed the object being voted on, which you can use to make complex decisions and effectively implement your own ACL. Enforcing authorization (e.g. the `isGranted()` part) will look similar to what you see in this article, but your voter class will handle the logic behind the scenes, instead of the ACL system.

https://symfony.com/doc/3.4/security/acl.html

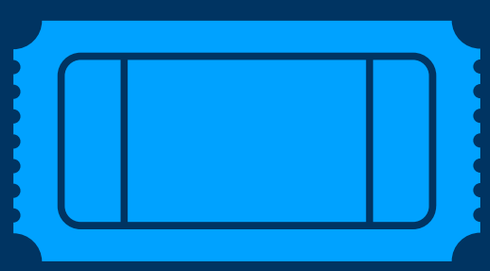
ABAC

using Symfony Voters

> “ Security Voters provide a mechanism to set up fine-grained restrictions in Symfony applications. The main advantage over ACLs is that they are an order of magnitude easier to set up, configure and use.

http://symfony.com/blog/new-in-symfony-2-6-simpler-security-voters

In Symfony, an authorisation decision will always be based on the following:



TOKEN

When a user is authenticated (identified) they will receive a token from the firewall to hand over to the access control in the authorisation step.

We can get the user's identity from the token.

SET OF ATTRIBUTES

Each attribute stands for a certain right the user should have.

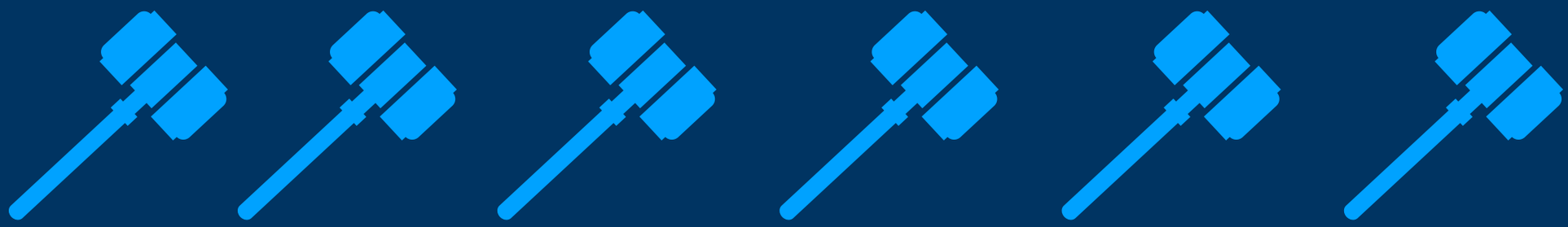
Eg. Role, Order Number, Email Address, Time of Day

RESOURCE

Any object for which access control needs to be checked, like an article or a comment object (or a piggy bank object containing bitcoins)

Access Decision Manager

Contains all voters. Some might be supported based on the attributes to vote on.



Voter 1

Voter 2

Voter 3

Voter 4

Voter 5

Voter 6

Access Decision Manager

Access Decision Manager

Affirmative Strategy

grant access as soon as there is one voter granting access

PERMIT

Voter 1	Voter 2	Voter 3	Voter 4	Voter 5	Voter 6
PERMIT	DENY	Not Supported	PERMIT	PERMIT	ABSTAIN
					

Access Decision Manager

Consensus Strategy

grant access if there are more voters granting access than there are denying

PERMIT

PERMIT	DENY	Not Supported	PERMIT	PERMIT	ABSTAIN
					
Voter 1	Voter 2	Voter 3	Voter 4	Voter 5	Voter 6

Access Decision Manager

Unanimous Strategy

grant access only if none of the voters have denied access

DENY

Voter 1	Voter 2	Voter 3	Voter 4	Voter 5	Voter 6
PERMIT	DENY	Not Supported	PERMIT	PERMIT	ABSTAIN
					

```php
use Symfony\Component\Security\Core\Authorization\AccessDecisionManager;
use Symfony\Component\Security\Core\Authorization\Voter\VoterInterface;

// instances of VoterInterface
$voters = [ '...' ];

// one of "affirmative", "consensus", "unanimous"
$strategy = AccessDecisionManager::STRATEGY_AFFIRMATIVE;

// whether or not to grant access when all voters abstain
$allowIfAllAbstainDecisions = false

// whether or not to grant access when there is no majority
// (applies only to the "consensus" strategy)
$allowIfEqualGrantedDeniedDecisions = false

$accessDecisionManager = new AccessDecisionManager(
    $voters,
    $strategy,
    $allowIfAllAbstainDecisions,
    $allowIfEqualGrantedDeniedDecisions
);
```

Built-in Symfony Voters

All are in the Symfony\Component\Security\Core\Authorization\Voter namespace

RoleVoter

```php
$roleVoter->vote($token, $object, array('ROLE_ADMIN'));
```

RoleHierarchyVoter

```php
use Symfony\Component\Security\Core\Authorization\Voter\RoleHierarchyVoter;
use Symfony\Component\Security\Core\Role\RoleHierarchy;

$hierarchy = array(
    'ROLE_SUPER_ADMIN' => array('ROLE_ADMIN', 'ROLE_USER'),
);

$roleHierarchy = new RoleHierarchy($hierarchy);

$roleHierarchyVoter = new RoleHierarchyVoter($roleHierarchy);
```

Built-in Symfony Voters

AuthenticatedVoter

```php
$vote = $authenticatedVoter->vote($token, $object, array('IS_AUTHENTICATED_FULLY'));
```

ExpressionVoter

```php
$vote = $expressionVoter->vote($token, $journalArticle, [
    new Expression( expression: 'subject.journal in user.journals')
]);
```

Creating custom voters

First, define what attributes you want to check.

```php
use ...

class OrderActionsVoter extends Voter
{
    const ACCOUNT   = 'account';
    const ADMIN     = 'admin';
    const GUEST     = 'guest';
    const SESSION   = 'session';

    const SUPPORTED_ATTRIBUTES = [
        self::ACCOUNT,
        self::ADMIN,
        self::GUEST,
        self::SESSION,
    ];
```

Second, check if your voter should vote on the given subject or attributes.

```php
    /**
     * {@inheritdoc}
     */
    protected function supports($attribute, $subject)
    {
        if (in_array($attribute, haystack: self::SUPPORTED_ATTRIBUTES, strict: true)) {
            return $subject instanceof Order;
        }

        return false;
    }
```

Third, cast the vote.

```php
     */
    protected function voteOnAttribute($attribute, $order, TokenInterface $token)
    {
        switch ($attribute) {
            case self::ADMIN:
                return $this->canManageAllOrders();

            case self::GUEST:
                return $this->isCustomerOfOrder($order, $token)
                    && $this->isGuestWithMatchingOrderDetails($order, $token);

            case self::ACCOUNT:
                return $this->isCustomerOfOrder($order, $token)
                    && $this->isRegisteredCustomer($token);

            case self::SESSION:
                return $this->hasMatchingCartOrPreviousOrderInSession($order);
        }
```

Finally, declare the service and it is ready to use.

```yaml
reiss.voter.order_summary:
    class: Reiss\WebBundle\Security\OrderActionsVoter
    arguments:
        - "@sylius.authorization_checker"
        - "@sylius.storage.session"
    tags:
        - { name: security.voter }
    public: false
```

```php
$this->denyAccessUnlessGranted( attributes: OrderActionsVoter::GUEST, $authorizedOrder);
```

Shortcomings of Symfony Voters

1. Still requires writing code, unless you implement a Voter that loads its logic from the database.

ABAC via XACML*

*Pronounced “X-akamull”, “X-A-C-M-L” or “zakamull”

[What is XACML?]

XACML (eXtensible Access Control Markup Language) offers a standardized way to achieve externalized and dynamic authorization. This means that authorization decisions are made by an authorization service **at run-time** based on policies which determine what actions a user or service can perform on a given information asset and in a specific context.

https://www.axiomatics.com/100-pure-xacml/

eXtensible Access Control Markup Language (XACML) Version 3.0

OASIS Standard

22 January 2013

Specification URIs

This version:

http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.doc (Authoritative)
http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html
http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf

Previous version:

http://www.oasis-open.org/committees/download.php/43799/xacml-3.0-core-spec-csprd03-en.zip

Latest version:

http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-en.doc (Authoritative)
http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-en.html
http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-en.pdf

Technical Committee:

OASIS eXtensible Access Control Markup Language (XACML) TC

Chairs:

Bill Parducci (bill@parducci.net), Individual
Hal Lockhart (hal.lockhart@oracle.com), Oracle

Editor:

Erik Rissanen (erik@axiomatics.com), Axiomatics AB

Additional artifacts:

This prose specification is one component of a Work Product which also includes:

- XML schema: http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd

Related work:

This specification replaces or supersedes:

- *eXtensible Access Control Markup Language (XACML) Version 2.0*. 01 February 2005. OASIS Standard. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf.

Declared XML namespace:

- urn:oasis:names:tc:xacml:3.0:core:schema:wd-17

Abstract:

This specification defines Version 3.0 of the eXtensible Access Control Markup Language.

Status:

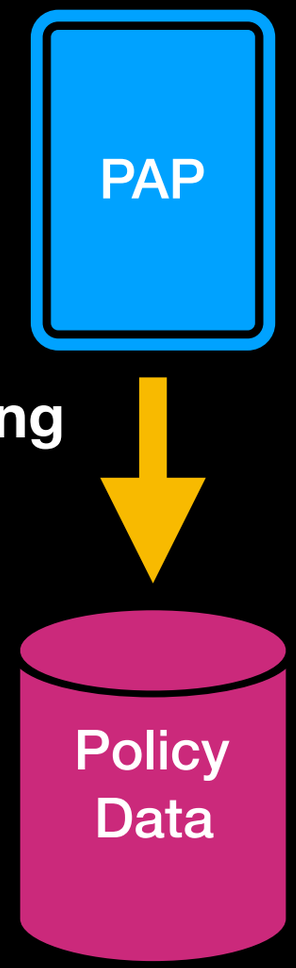
This document was last revised or approved by the membership of OASIS on the above date. The level of approval is also listed above. Check the "Latest version" location noted above for possible later revisions of this document.

http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

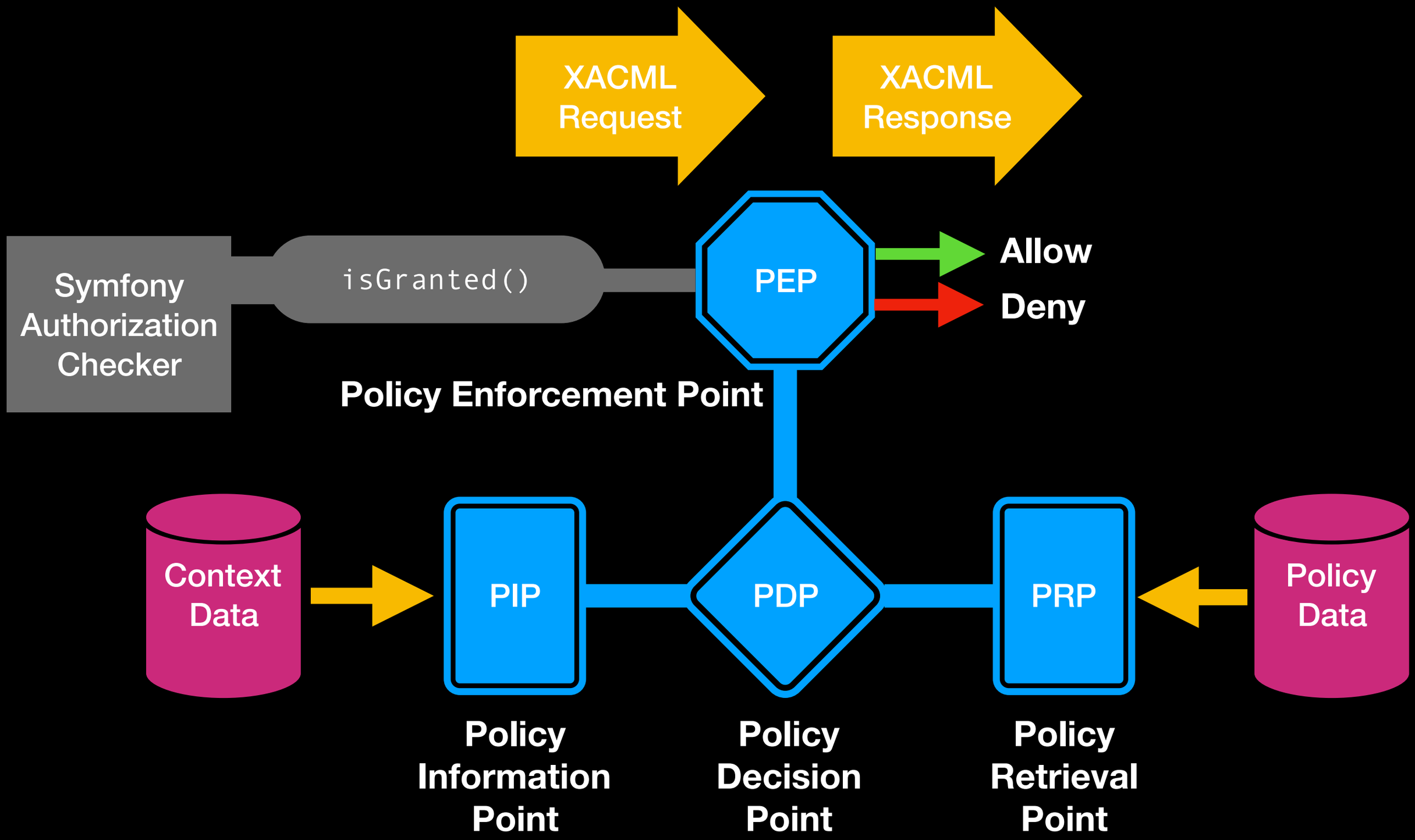
XACML Administration

Policy Administration Point (PAP)

- **Create, View, Delete policies**
- **Version policies on Update**
- **Evaluate policies before committing**



XACML Enforcement Flow



XACML 3.0 Policies

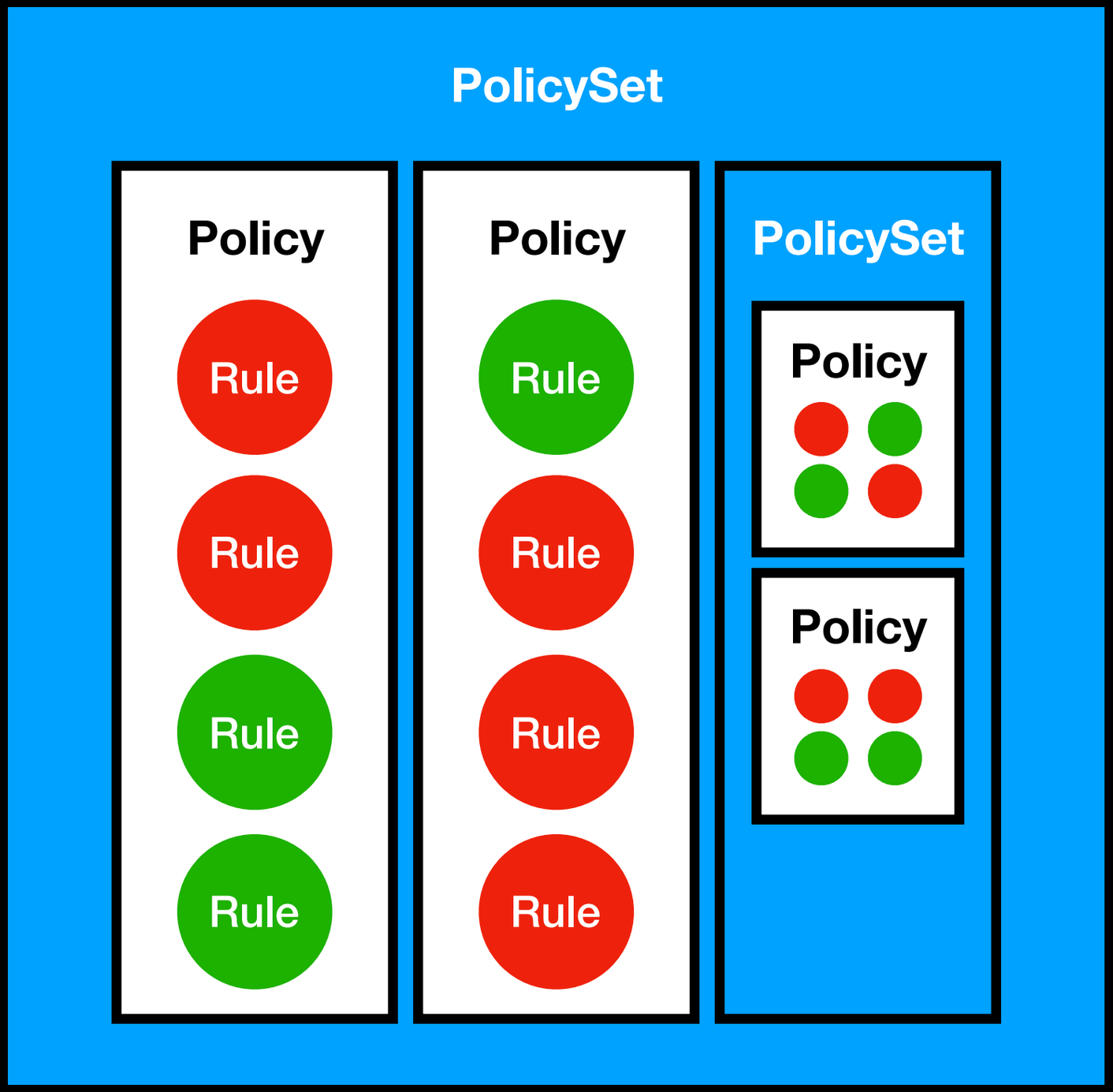
Policy Sets contain a collection of Policies.

They may also contain or reference other Policy Sets.

However, the Decision Point will only evaluate at Policy level.

Rules are never evaluated by themselves.

Neither are Policy Sets.



“ Targets and Rules

Part of what [the] XACML PDP [Policy Decision Point] needs to do is find a policy that applies to a given request. To do this, XACML provides another feature called a Target.

A Target is basically a set of simplified conditions for the Subject, Resource and Action that must be met for a PolicySet, Policy or Rule to apply to a given request.

If all the conditions of a Target are met, then its associated PolicySet, Policy, or Rule applies to the request.

In addition to being a way to check applicability, Target information also provides a way to index policies, which is useful if you need to store many policies and then quickly sift through them to find which ones apply.

A Request must be matched to a Policy

Request

Policy A

Policy B

Policy C

Policy D

Policy E

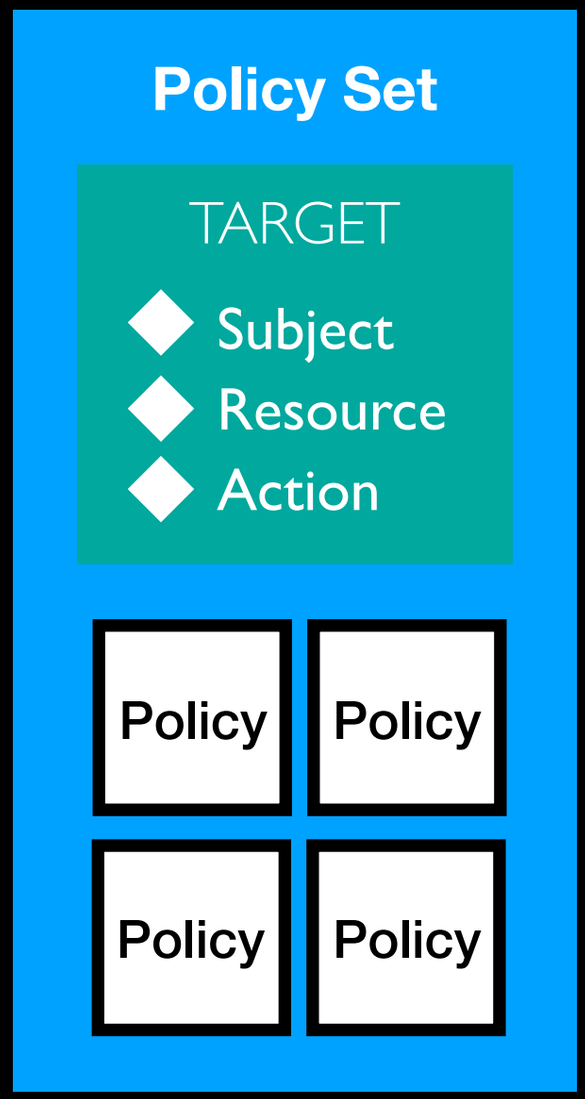
Policy F

Policy G

This is done using Targets

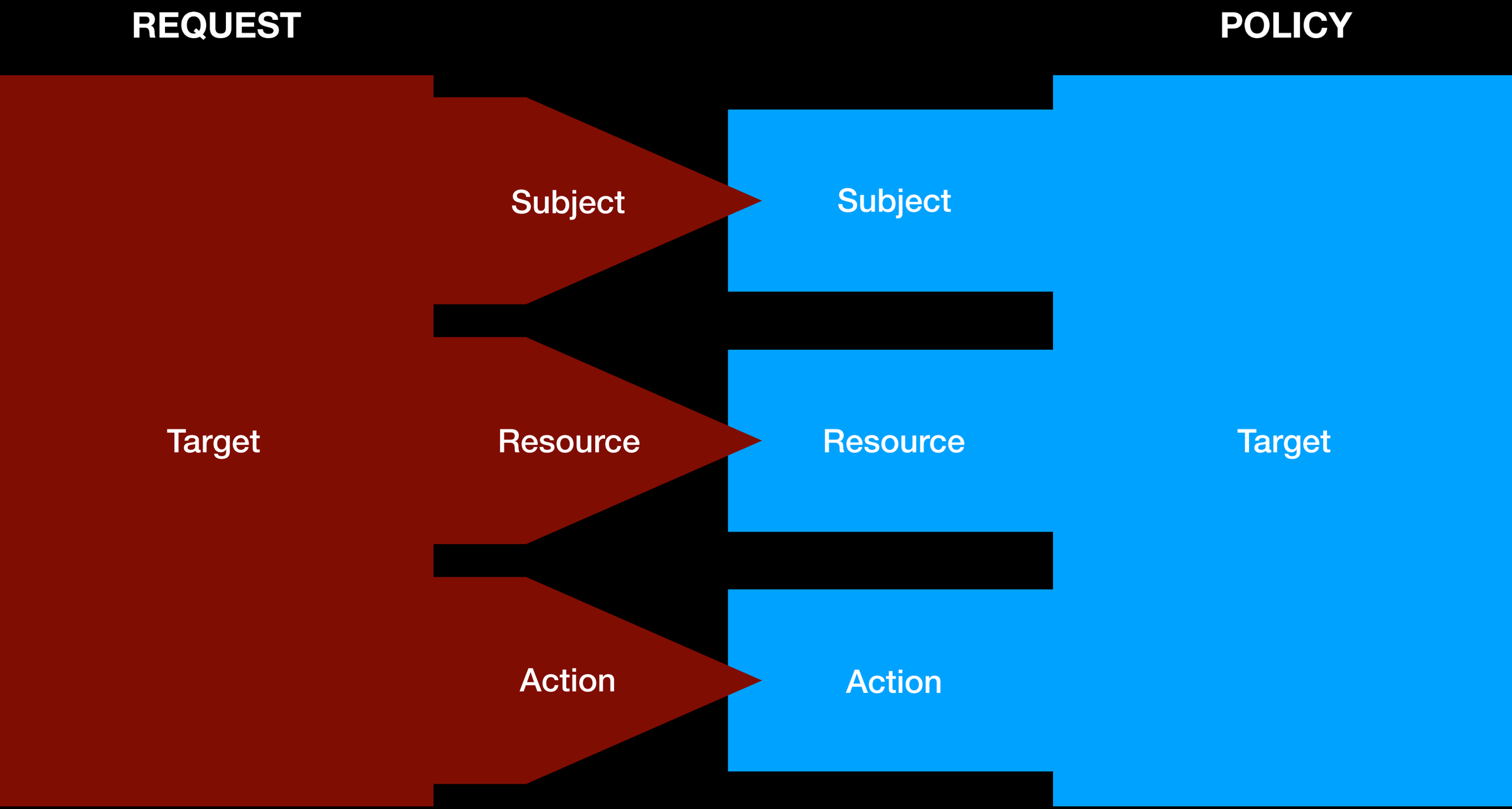
XACML 3.0 Targets

Policies, Policy Sets and Rules only apply if the Target matches.



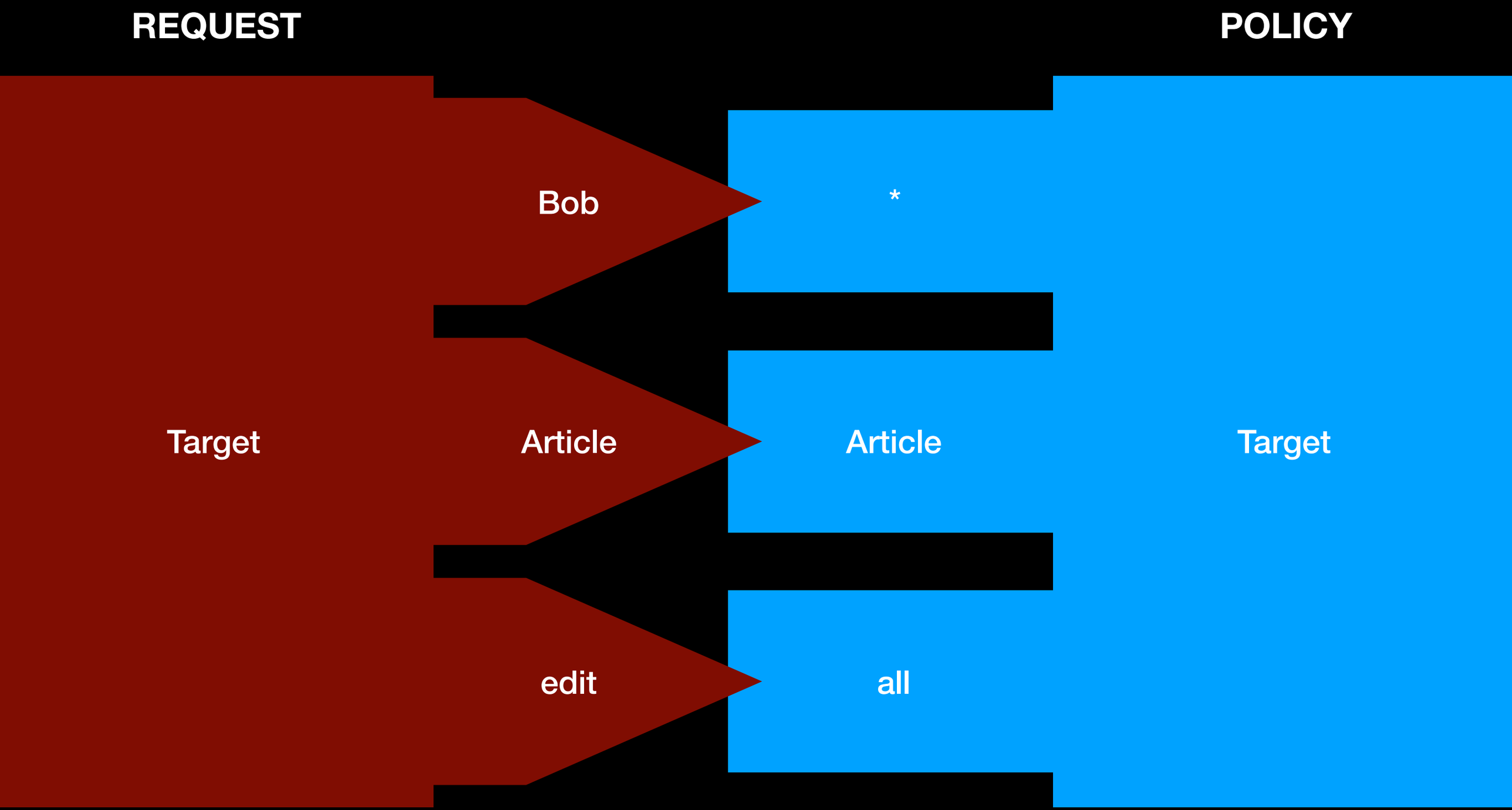
Targets consist of Subject, Resource and Action

behaves like Voter::supports() in Symfony

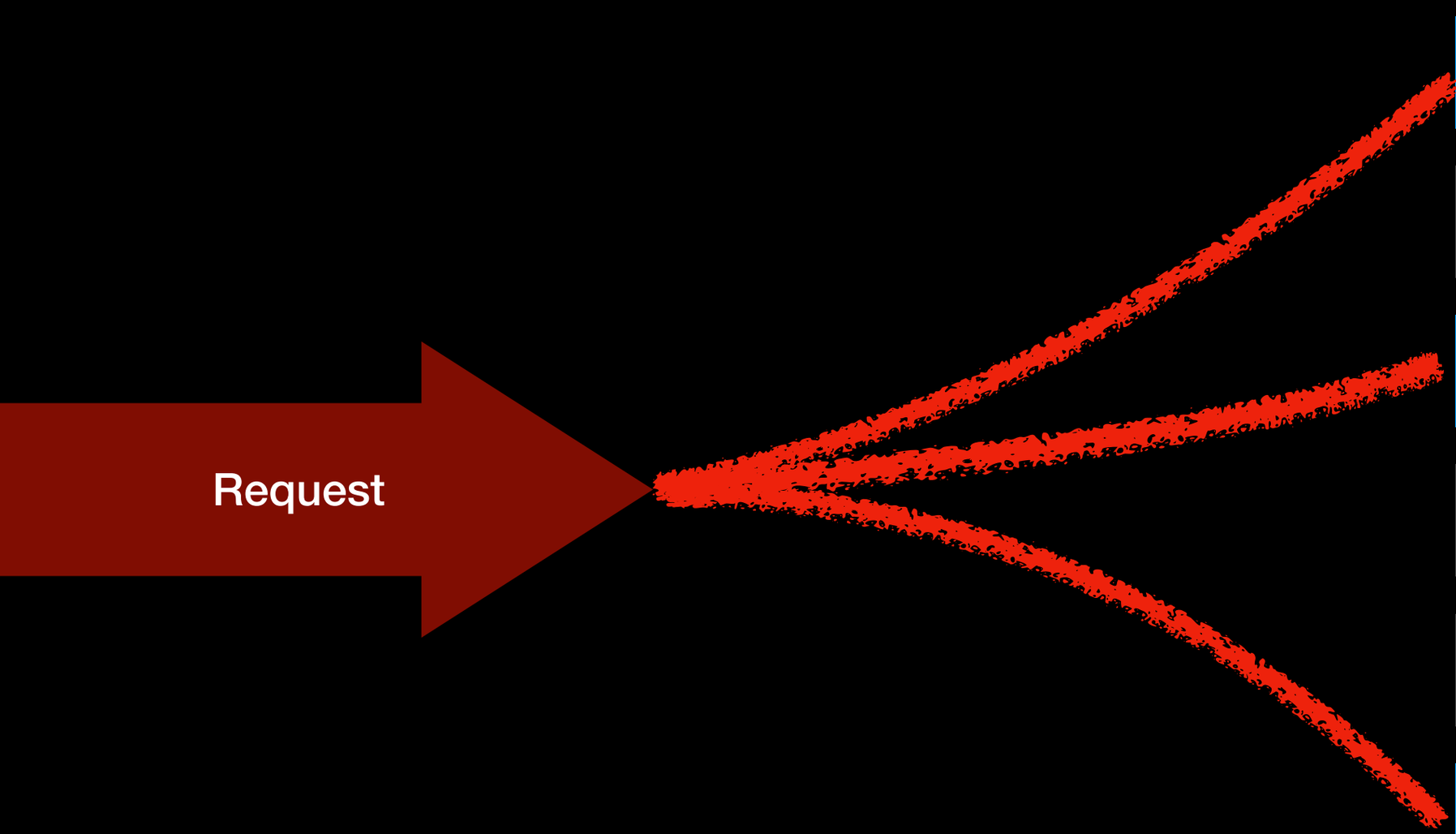


Targets consist of Subject, Resource and Action

behaves like Voter::supports() in Symfony



More than one policy may be matched



Policy A

Policy B

Policy C

Policy D

Policy E

Policy F

Policy G

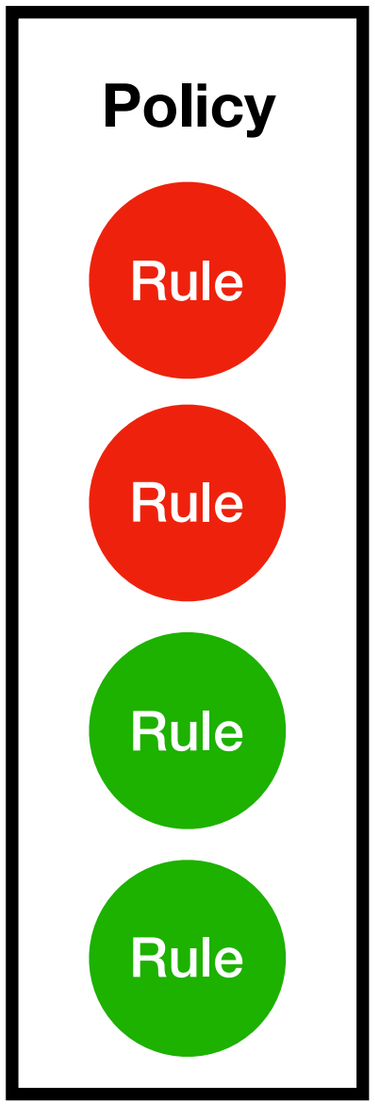
XACML 3.0 Rule Example

```
<Rule Id="R1" Effect="Permit">
    <Target>
        <Resources>
            <R>Journal Article Submission #5N9400</R>
        </Resources>
        <Actions>
            <A>Make Suggestion</A>
        </Actions>
    </Target>
    <Condition>
        <Apply Function="AND">
            Role: Editor-in-Chief
            Journal: CJES
            Date: Before 2018-09-01
        </Apply>
    </Condition>
</Rule>
```

* The XACML syntax is more verbose than what you see here.

“ Understanding XACML combining algorithms

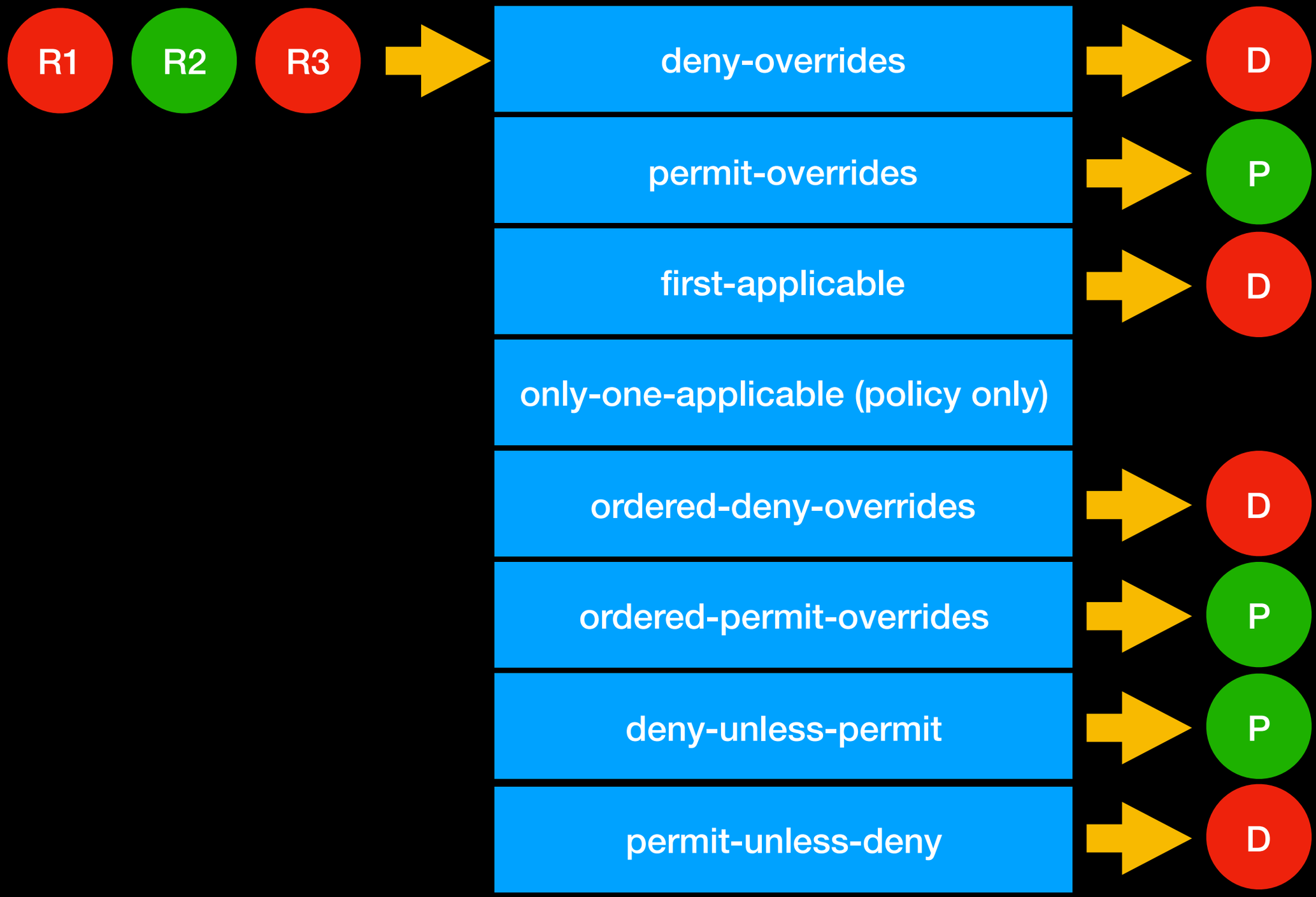
If a policy contains multiple rules, and the rules return different decisions e.g. **Permit** and **Deny**, what should the policy return? Permit? Deny? Neither?



https://www.axiomatics.com/blog/understanding-xacml-combining-algorithms/

XACML 3.0 Rule-Combining and Policy-Combining Algorithms

behaves like AccessDecisionManager Strategies in Symfony



XACML 3.0 Policy Example

```
<Policy>
    <Target>
        <Resources />
        <Subjects />
        <Actions />
    </Target>

    <RuleSet ruleCombiningAlgId="AllowOverrides">
        <Rule Id="R1" />
        <Rule Id="R2" />
        <Rule Id="R3" />
    </RuleSet>
</Policy>
```

* The XACML syntax is more verbose than what you see here.

XACML 3.0 Policy Example

```
<Policy>
    <Target>
        <Resources />
        <Subjects />
        <Actions />
    </Target>

    <RuleSet ruleCombiningAlgId="AllowOverrides">
        <Rule Id="R1" />
        <Rule Id="R2" />
        <Rule Id="R3" />
    </RuleSet>
</Policy>
```

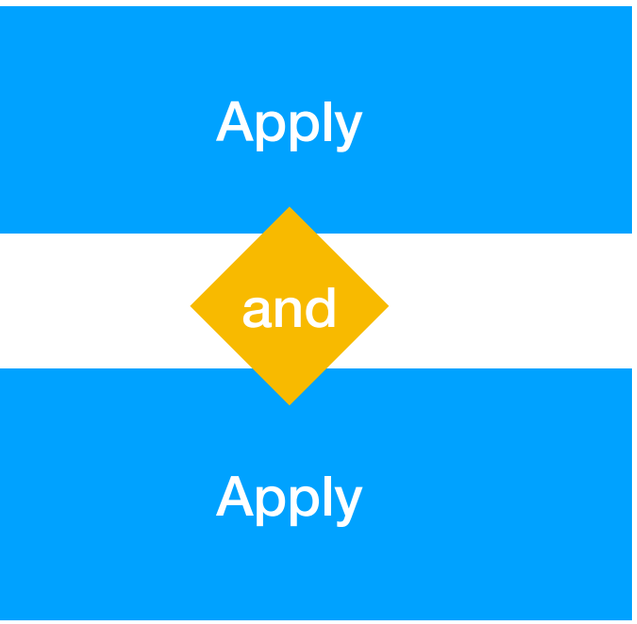
* The XACML syntax is more verbose than what you see here.

Conditions

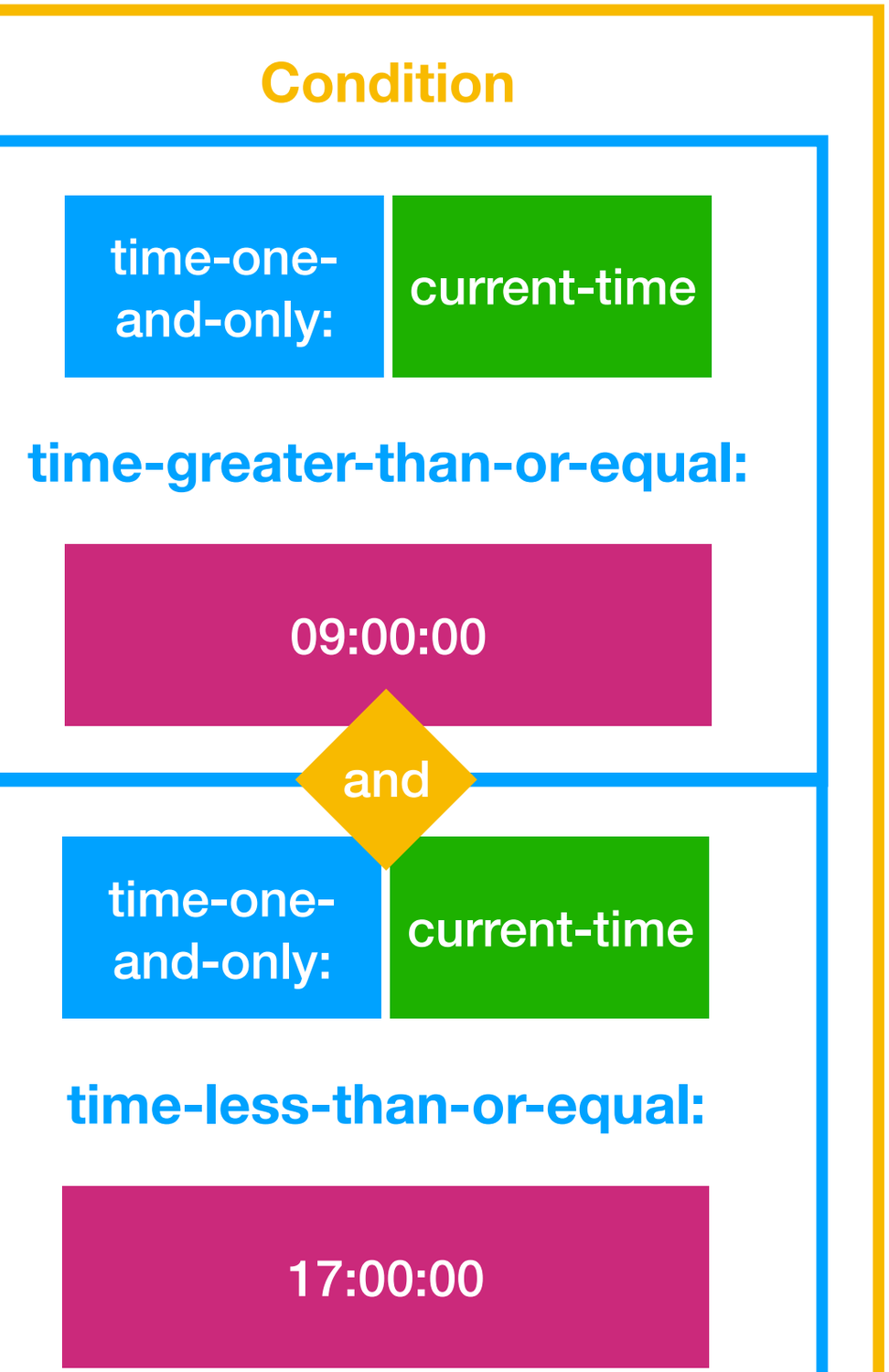
Allow only logins between 9am and 5pm.

```
<!-- Only allow logins from 9am to 5pm -->
<Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-greater-than-or-equal"
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
      <EnvironmentAttributeSelector DataType="http://www.w3.org/2001/XMLSchema#time"
                                    AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
    </Apply>
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">09:00:00</AttributeValue>
  </Apply>
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-less-than-or-equal"
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
      <EnvironmentAttributeSelector DataType="http://www.w3.org/2001/XMLSchema#time"
                                    AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
    </Apply>
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">17:00:00</AttributeValue>
  </Apply>
</Condition>
```

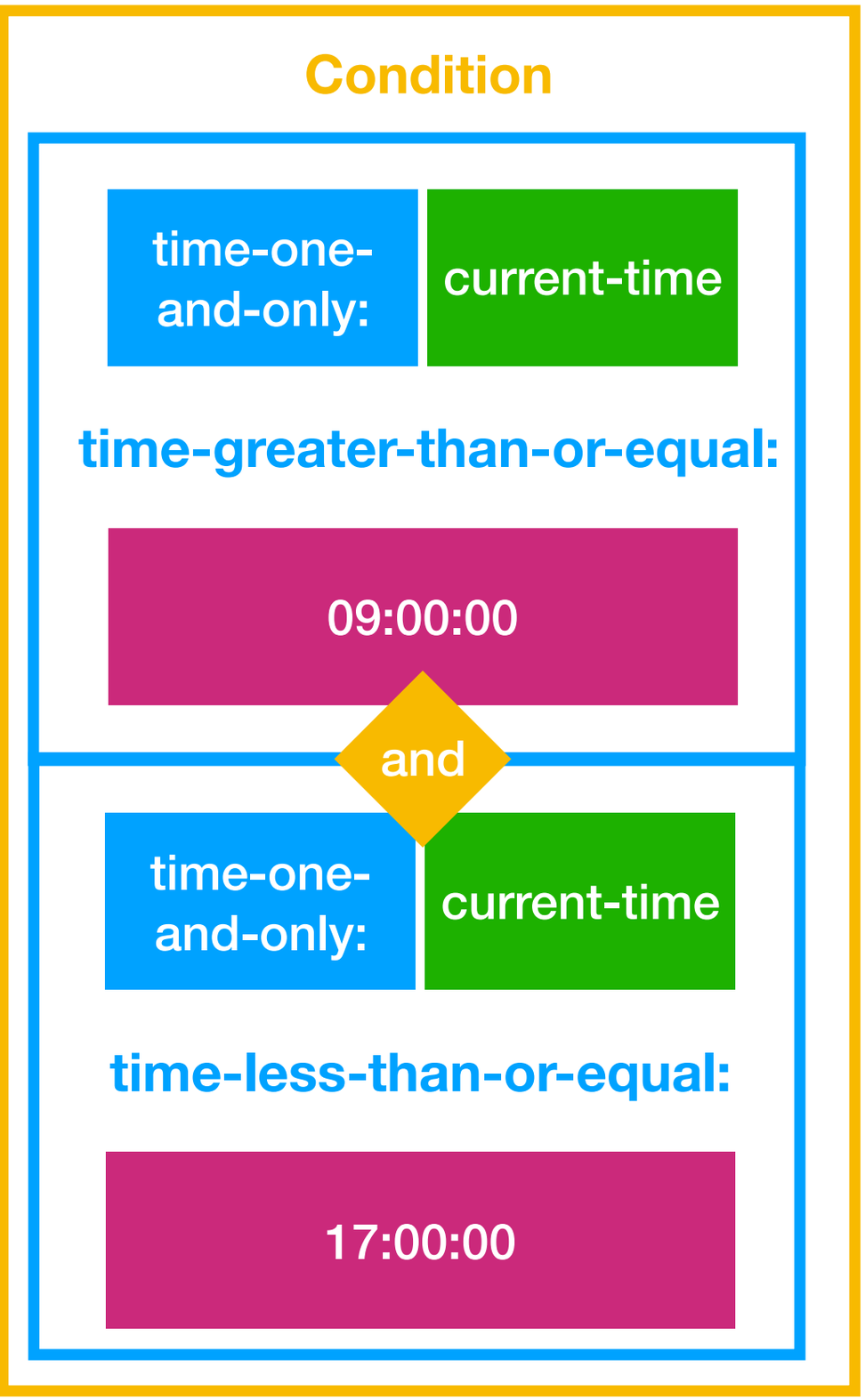
Condition




```
    <!-- Only allow logins from 9am to 5pm -->
    <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-greater-than-or-
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
          <EnvironmentAttributeSelector DataType="http://www.w3.org/2001/XMLSchema#t
                                        AttributeId="urn:oasis:names:tc:xacml:1.0:en
        </Apply>
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">09:00:00</A
      </Apply>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-less-than-or-equ
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
          <EnvironmentAttributeSelector DataType="http://www.w3.org/2001/XMLSchema#t
                                        AttributeId="urn:oasis:names:tc:xacml:1.0:en
        </Apply>
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">17:00:00</A
      </Apply>
    </Condition>
```




```
<!-- Only allow logins from 9am to 5pm -->
<Condition f="and">
  <Apply f="time-greater-than-or-equal"
    <Apply f="time-one-and-only">
      <EnvironmentAttributeSelector
       DataType="#time"
       AttributeId="environment:current-time"/>
    </Apply>
    <AttributeValue
     DataType="#time">09:00:00</AttributeValue>
  </Apply>
  <Apply f="time-less-than-or-equal"
    <Apply f="time-one-and-only">
      <EnvironmentAttributeSelector
       DataType=“#time"
       AttributeId="environment:current-time"/>
    </Apply>
    <AttributeValue
     DataType=#time">17:00:00</AttributeValue>
  </Apply>
</Condition>
```




```
$timeGreaterThanOrEq = function($x, $y): bool {
    return $x >= $y;
}

$timeLessThanOrEq = function($x, $y): bool {
    return $x <= $y;
}

$timeOneAndOnly = function($x): \DateTimeInterface {
    return new \DateTimeImmutable($x);
}

$condition = \Functional\true([
    $timeGreaterThanOrEq(
        $timeOneAndOnly($env->getCurrentTime()), '09:00:00'
    ),
    $timeLessThanOrEq(
        $timeOneAndOnly($env->getCurrentTime()), '17:00:00'
    ),
]);
```

“

What’s a XACML Obligation?

The XACML standard defines the concept of obligations which are elements which can be returned along with a XACML decision (either of Permit or Deny) in order to enrich that decision. **Obligations are triggered on either Permit or Deny.** The Policy Enforcement Point [PEP] must implement and enforce obligations. If it fails to do so, it must deny access to the requested resource (in the case of a Permit).

https://www.webfarmr.eu/2015/02/tgif-xacml-whats-a-xacml-obligation/

Examples of Obligations

- **Auditing -** Log when an action was performed on a resource.
- **Security Checkup -** Ask the user to review their 2FA details after a remembered login.
- **Security Lockdown -** If credentials entered incorrectly multiple times.
- **Break-the-Glass Scenario -** Medical records may need to be accessed in emergency situations, regardless of what permissions were granted.

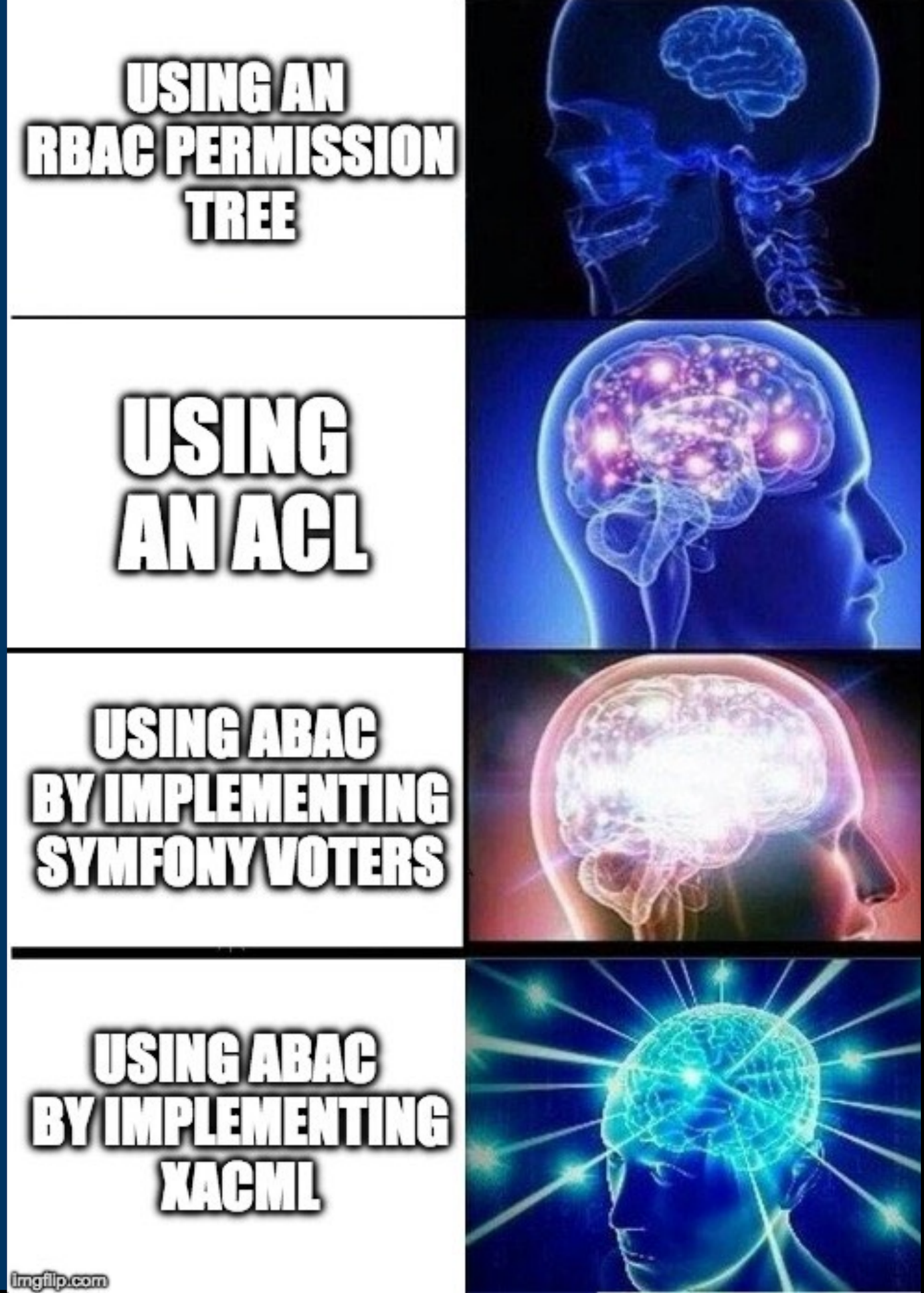
Shortcomings of XACML

- **XACML** syntax is very verbose.
- Is complex, but better describes business requirements than ACL.
- Somewhat limited resources, or non-concise.
- Overkill for most needs.

SUMMARY

Symfony Voters solve 80% of your requirements for 20% of the work.

XACML scales well and is enterprise-capable.



Thank you for listening

Adam Elsodaney

LEAD DEVELOPER

VERUSCRIPT

Publish high-quality, cost-effective journals with our publishing services

adam@veruscript.com

@ArchFizz

www.veruscript.com

@Veruscript

ACL Demo
https://github.com/adamelso/acland

Slides
github.com/adamelso/symfony-uk-meetup-2018-08-30-access-control