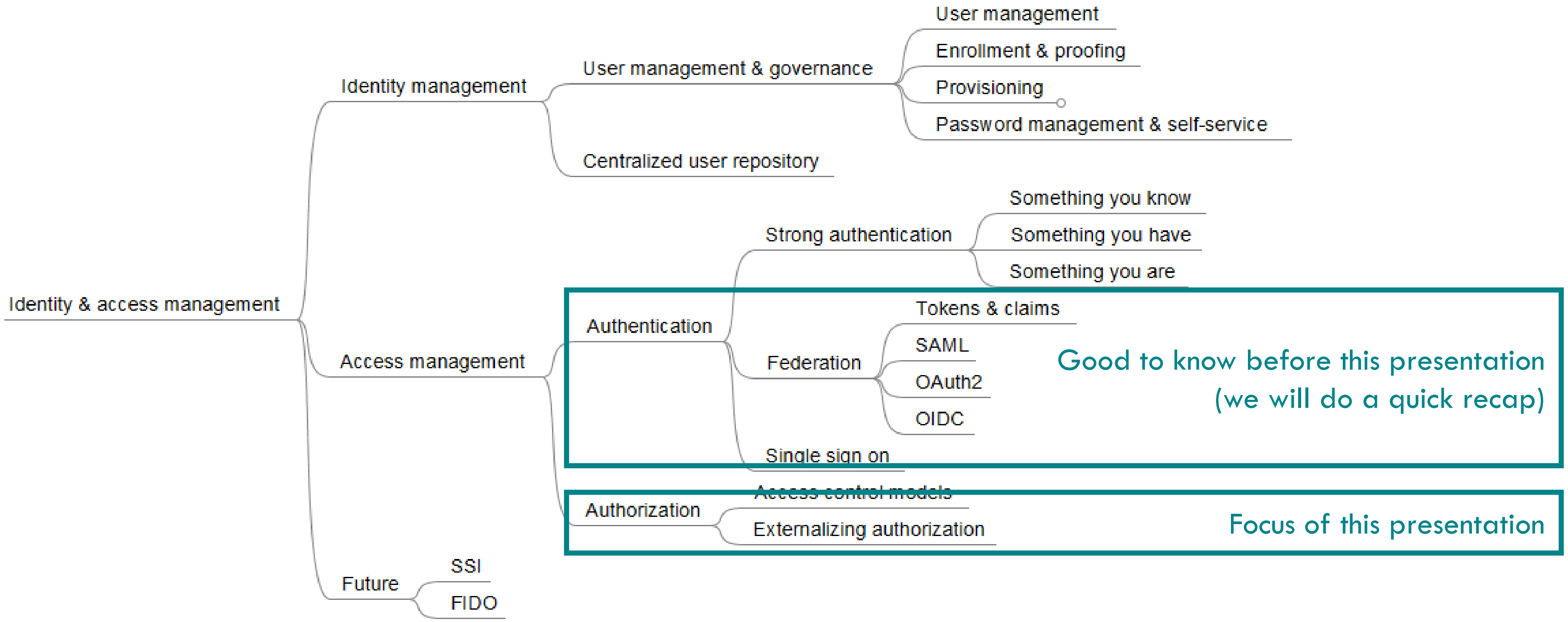
EXTERNALIZING AUTHORIZATION
USING OPEN POLICY AGENT



- Identity & access management
  - Identity management
    - User management & governance
      - User management
      - Enrollment & proofing
      - Provisioning
      - Password management & self-service
    - Centralized user repository
  - Access management
    - Authentication
      - Strong authentication
        - Something you know
        - Something you have
        - Something you are
      - Federation
        - Tokens & claims
        - SAML
        - OAuth2
        - OIDC
      - Single sign on
    - Authorization
      - Access control models
      - Externalizing authorization
  - Future
    - SSI
    - FIDO

Good to know before this presentation (we will do a quick recap)

Focus of this presentation

IMPORTANT CONCEPTS

- **Identification**: The process of discovering the identity (i.e., origin or initial history) of a person or item from the entire collection of similar persons or items.
- **Authentication**: Verifying the identity of an entity (user, process, or device), often as a prerequisite to allowing access to resources in an information system.
- **Authorization**: The process of verifying that a requested action or service is approved for a specific entity.
- **Externalization**: In the context of this presentation, ‘externalization’ means separating out specific (security) functionalities from an application into a component dedicated for such specific functionality. Often this naturally leads to ‘**centralization**’, as more and more applications externalize to the same dedicated component.

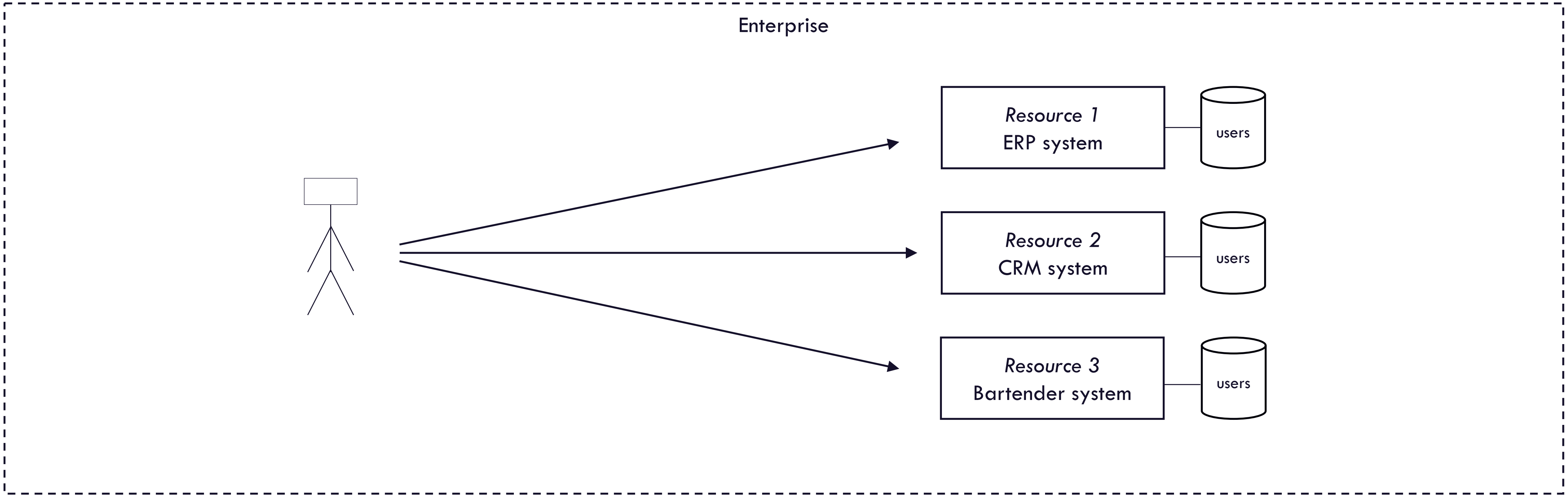
AGENDA

The problem we are trying to solve (≈ 5 slides)	Stage 0: The start Stage 1: Externalize the user directory Stage 2: Externalize authentication (**current state**) Stage 3: Externalize authorization (**focus of this presentation**)
How it's done in practice (≈ 4 slides)	From RBAC to ABAC in a .NET application
Demo time (≈ 2 slides)	A demo bartender system where authorization is externalized
Common questions (≈ 4 slides)	Relation to SSO Relation to Conditional Access Relation to ABAC Relation to XACML
Conclusion and questions (≈ 2 slides)	Conclusion and questions

WHICH PROBLEM ARE WE TRYING TO SOLVE?

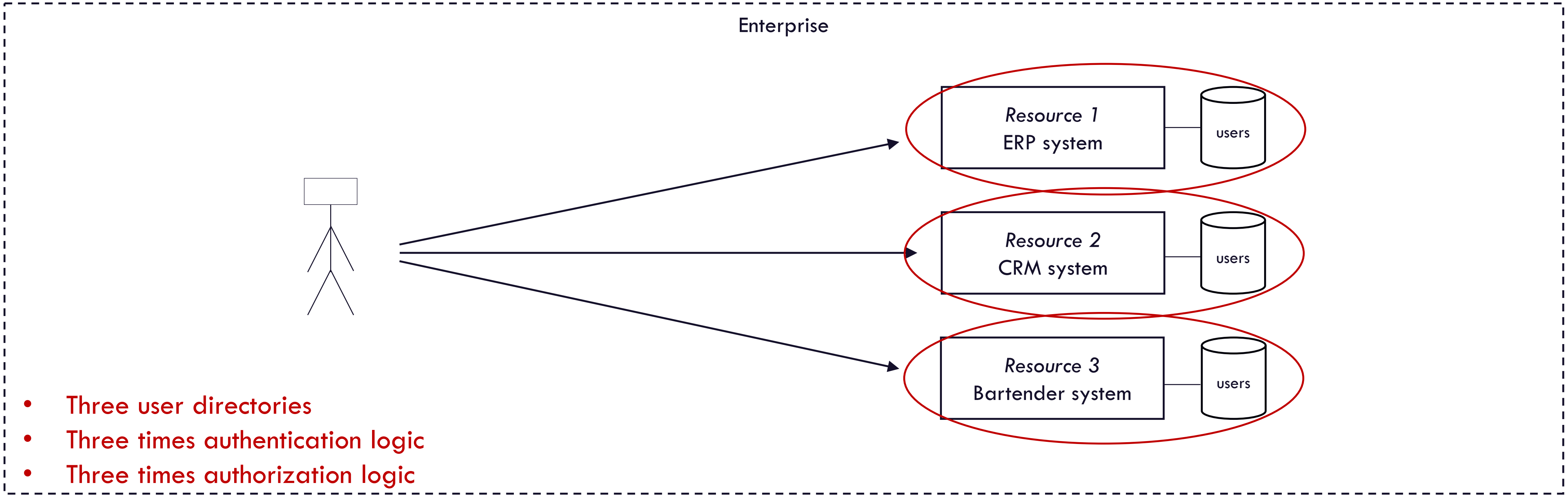
SPLYNTER

STAGE 0: THE STARTING PROBLEM



People need access to systems.

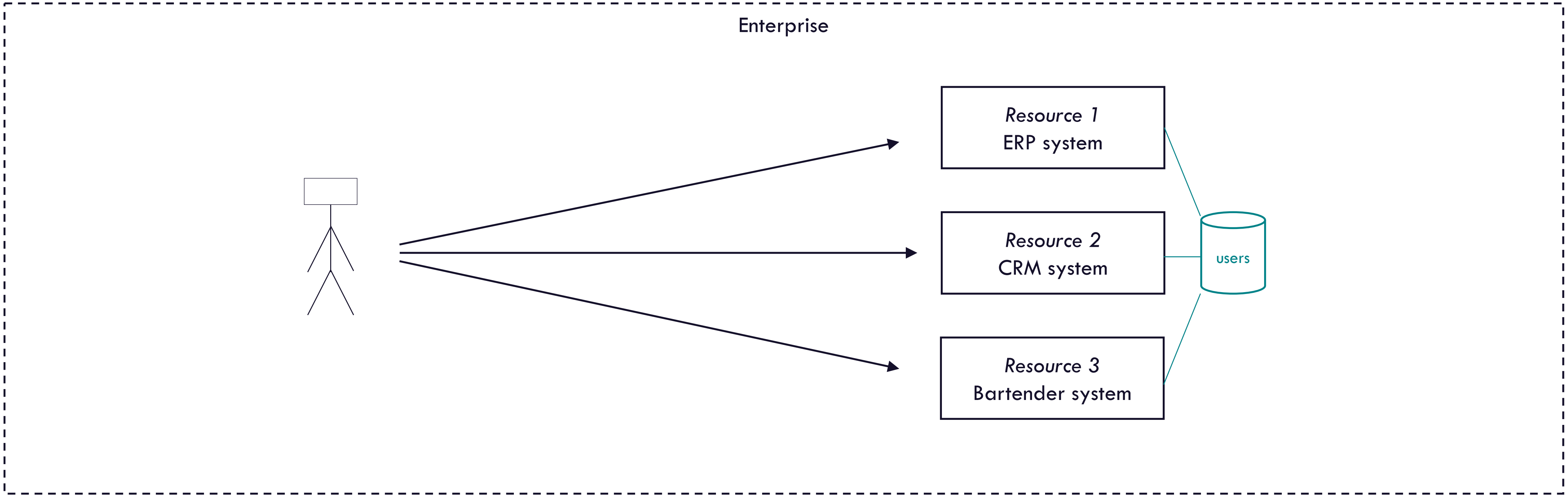
STAGE 0: THE STARTING PROBLEM



- Three user directories
- Three times authentication logic
- Three times authorization logic

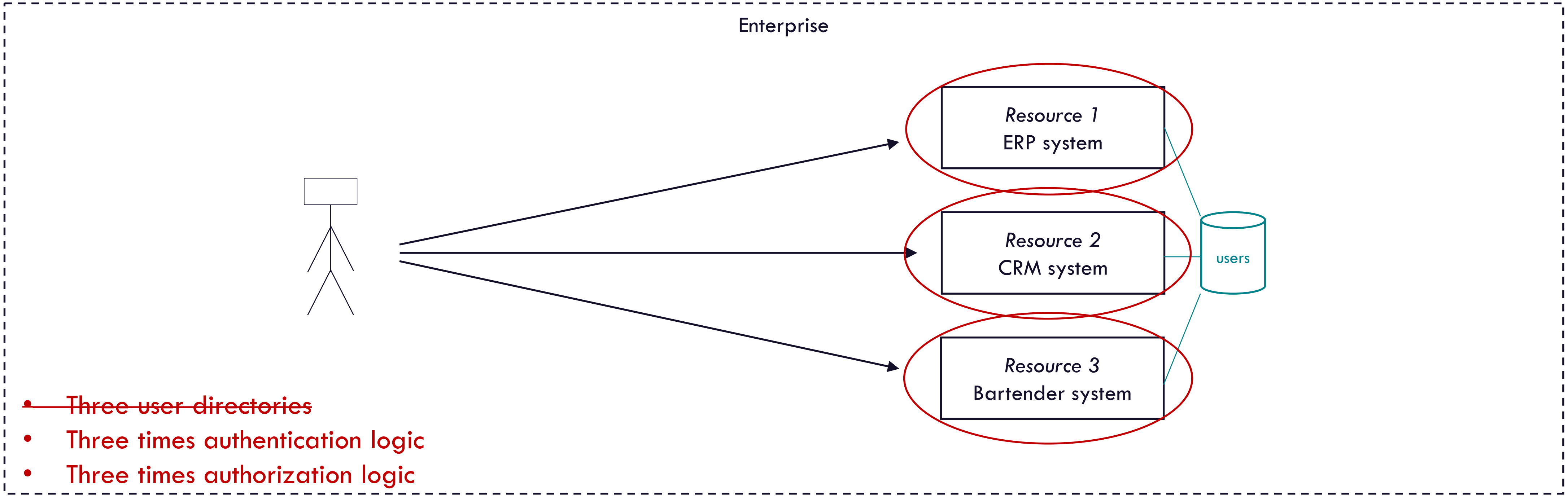
Every application reinvented the wheel.

STAGE 1: EXTERNALIZE THE USER DIRECTORY



This was a lot better! Think Kerberos and Active Directory.

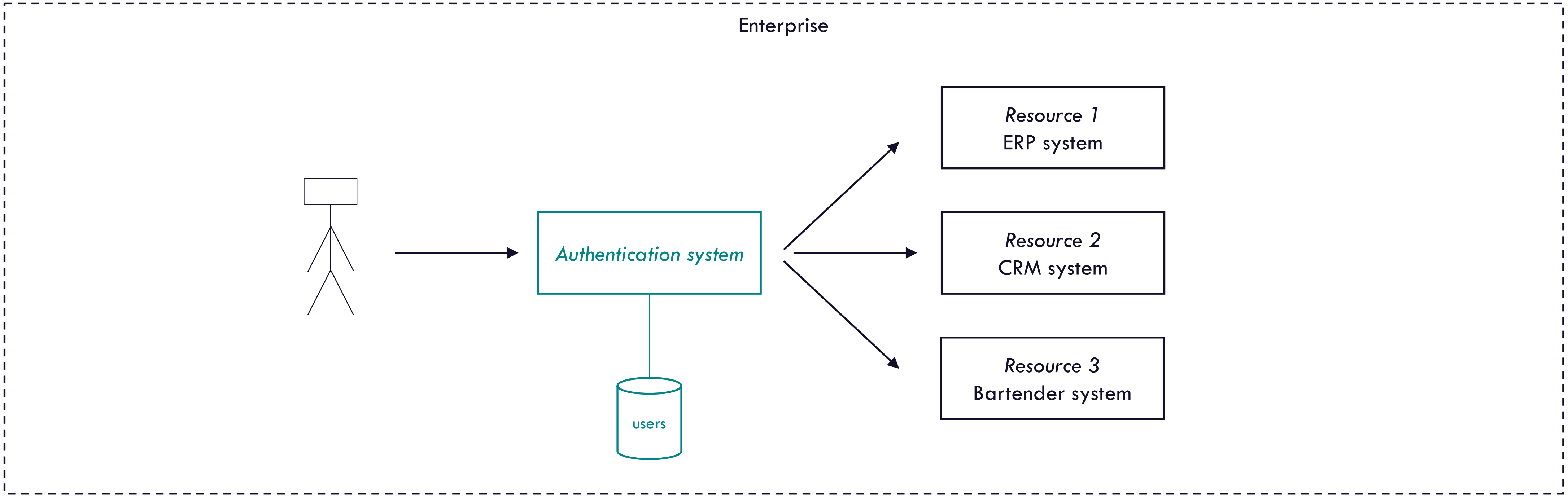
STAGE 1: EXTERNALIZE THE USER DIRECTORY



- ~~Three user directories~~
- Three times authentication logic
- Three times authorization logic

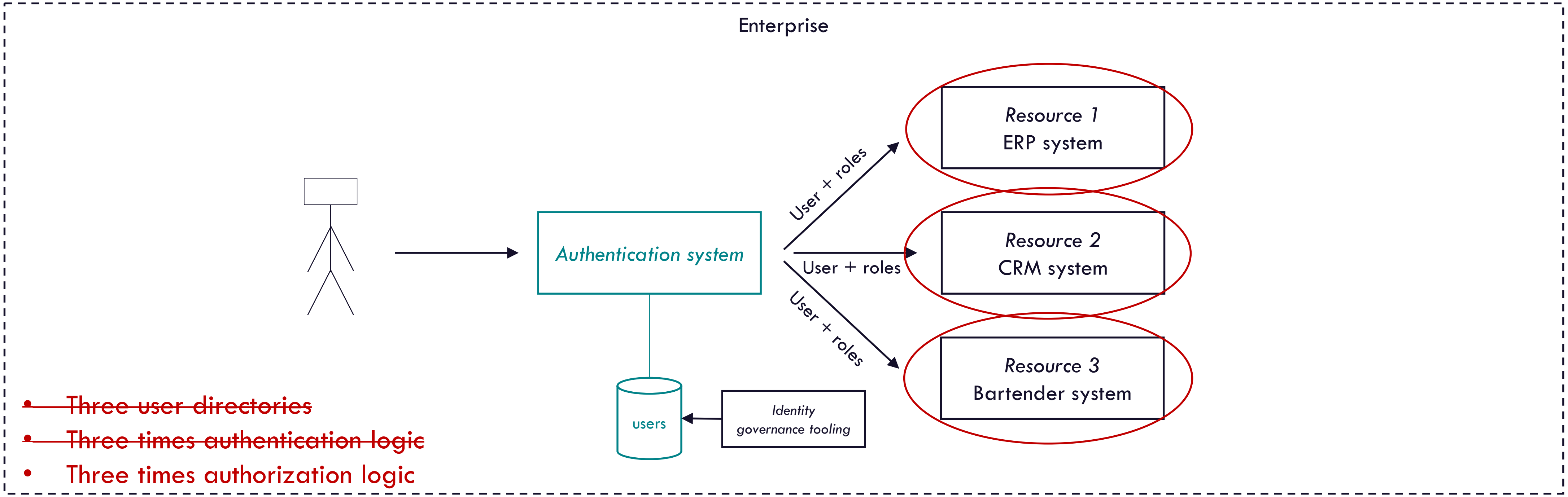
Still, the authentication logic was not shared

STAGE 2: EXTERNALIZE AUTHENTICATION



This currently is the state of the industry. Think SAML, OAuth2 and OIDC.

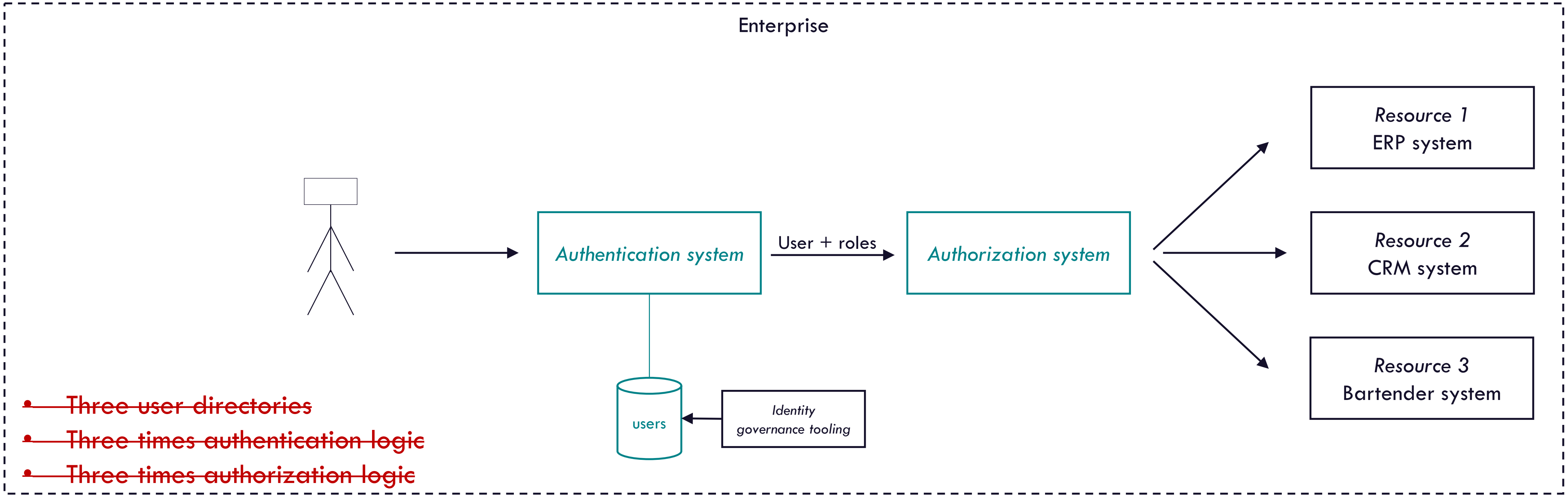
STAGE 2: EXTERNALIZE AUTHENTICATION



- ~~Three user directories~~
- ~~Three times authentication logic~~
- Three times authorization logic

*But, what about authorization? Sure, roles are at best governed centrally. But **deciding what permissions are linked to these roles is still managed inside each system separately.***

STAGE 3: SOLUTION - EXTERNALIZE AUTHORIZATION

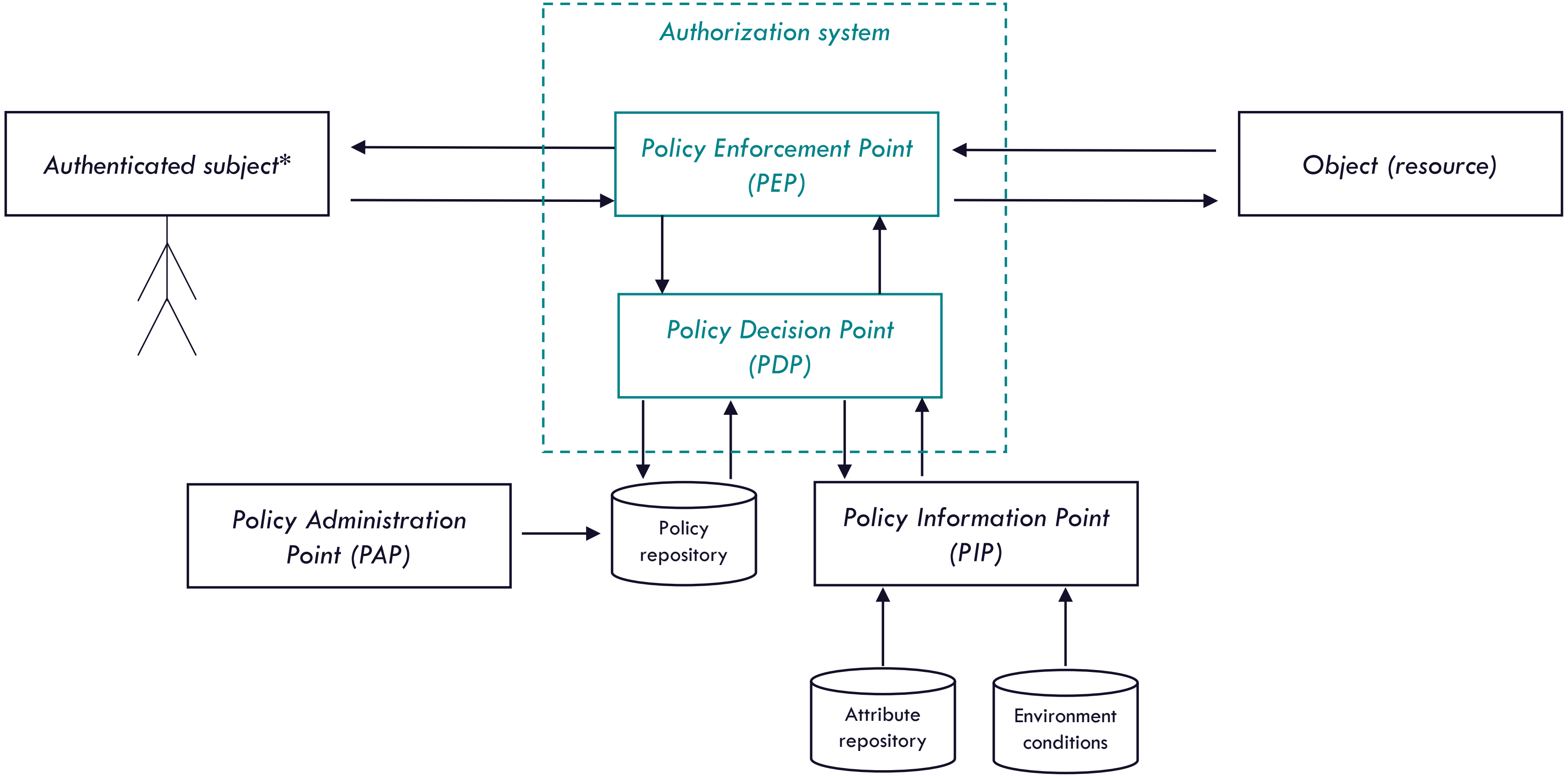


- ~~Three user directories~~
- ~~Three times authentication logic~~
- ~~Three times authorization logic~~

Externalizing authorization provides increased control and thereby assurance, easier auditing and compliance reporting, and a centralized management point for all authorization policies.

SPLYNTER

STAGE 3: SOLUTION - EXTERNALIZE AUTHORIZATION



Authorization system
Authenticated subject*
Policy Enforcement Point (PEP)
Object (resource)
Policy Decision Point (PDP)
Policy Administration Point (PAP)
Policy repository
Policy Information Point (PIP)
Attribute repository
Environment conditions

* We leave the authentication problem behind us, and assume the subject is already authenticated.

HOW IT’S DONE IN PRACTICE

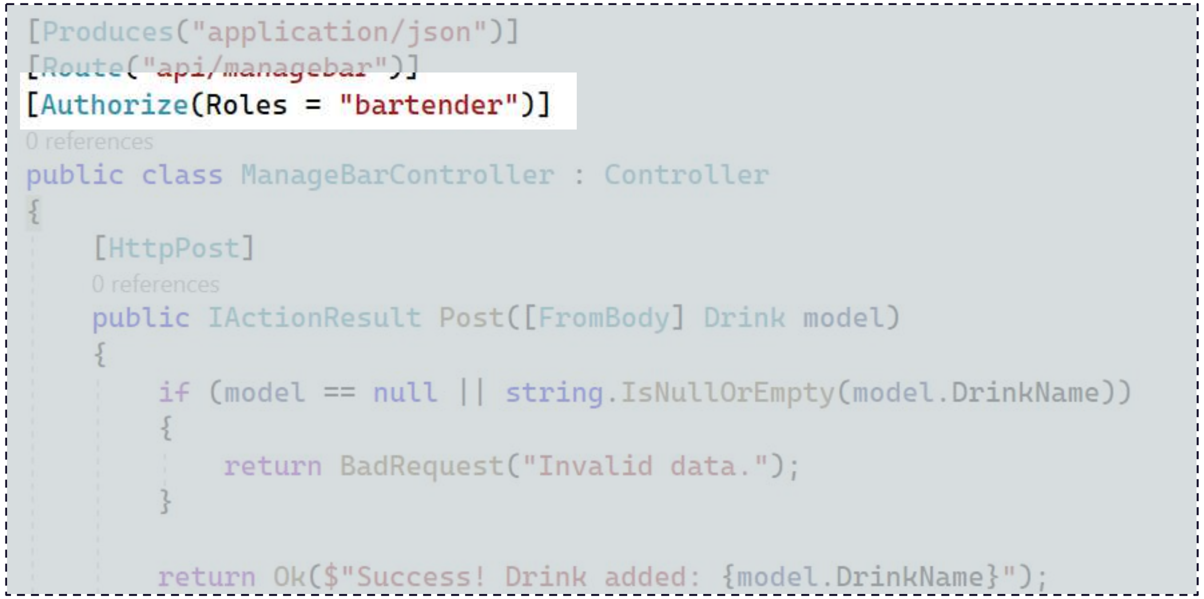
LET'S BUILD A BARTENDER SYSTEM

Requirements:

- Every customer can order non-alcoholic beverages
- Only customers >=18 can order alcoholic beverages
- Bartenders can add new drinks to the system

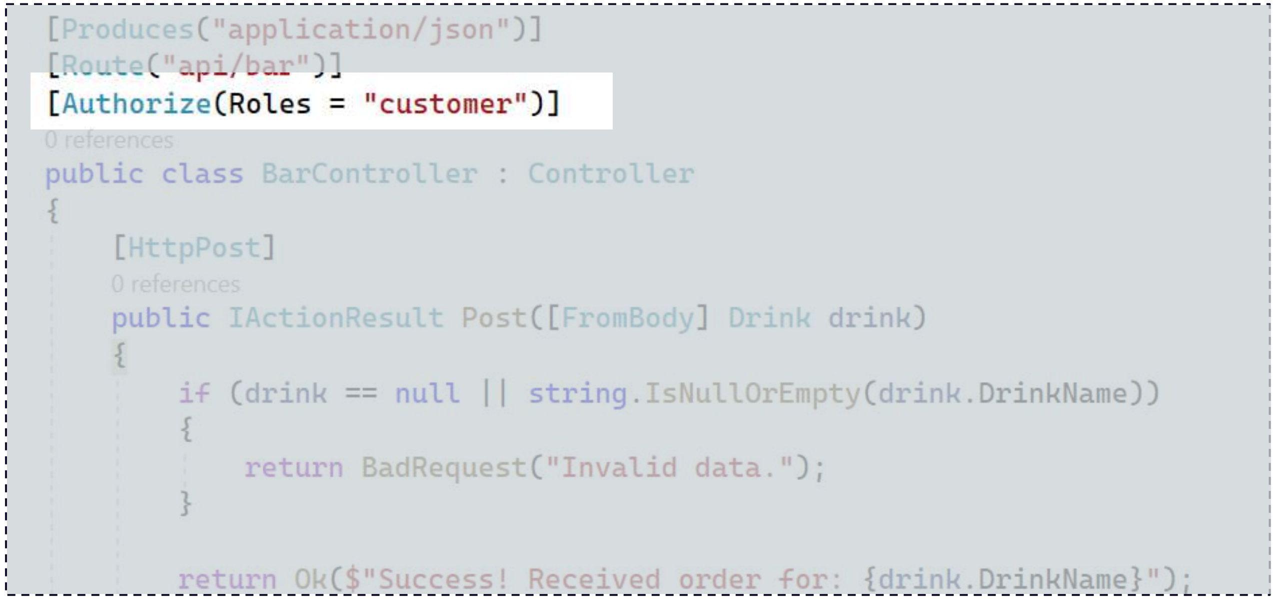
WITHOUT EXTERNALIZATION – PURE RBAC

- The application is responsible to validate what a certain role is permitted to do.
- The developer decides - or forgets (**Broken Access Control is OWASP's number 1 risk**).
- It is ~~difficult~~ impossible to get a view on all permissions linked to a role: Can a customer do something else? And in a different application?



```
[Produces("application/json")]
[Route("api/managebar")]
[Authorize(Roles = "bartender")]
0 references
public class ManageBarController : Controller
{
    [HttpPost]
    0 references
    public IActionResult Post([FromBody] Drink model)
    {
        if (model == null || string.IsNullOrEmpty(model.DrinkName))
        {
            return BadRequest("Invalid data.");
        }

        return Ok($"Success! Drink added: {model.DrinkName}");
```

Bartenders can add new drinks to the system

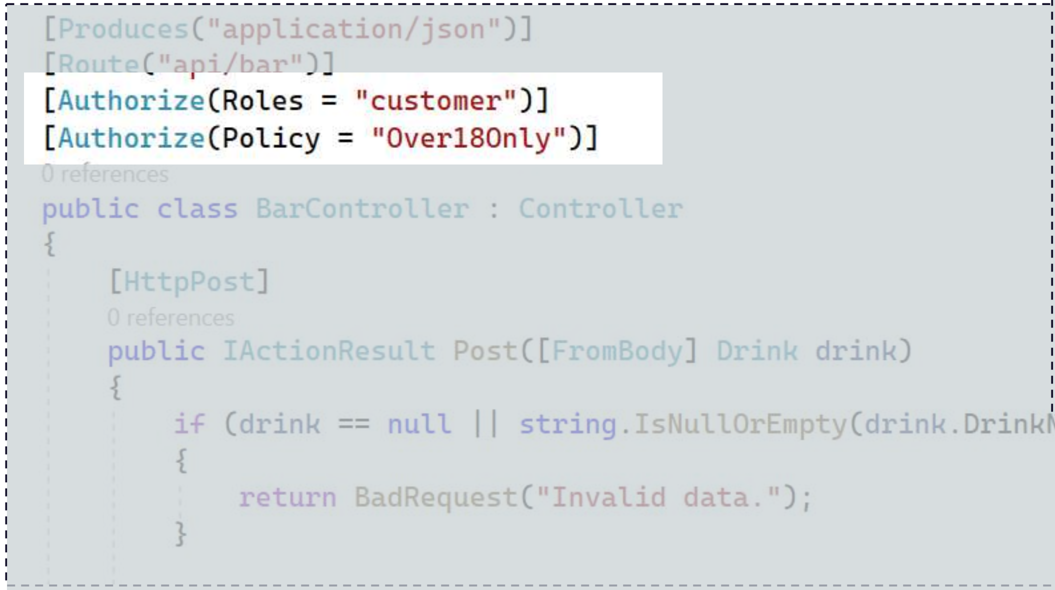


```
[Produces("application/json")]
[Route("api/bar")]
[Authorize(Roles = "customer")]
0 references
public class BarController : Controller
{
    [HttpPost]
    0 references
    public IActionResult Post([FromBody] Drink drink)
    {
        if (drink == null || string.IsNullOrEmpty(drink.DrinkName))
        {
            return BadRequest("Invalid data.");
        }

        return Ok($"Success! Received order for: {drink.DrinkName}");
```

Every customer can order non-alcoholic beverages

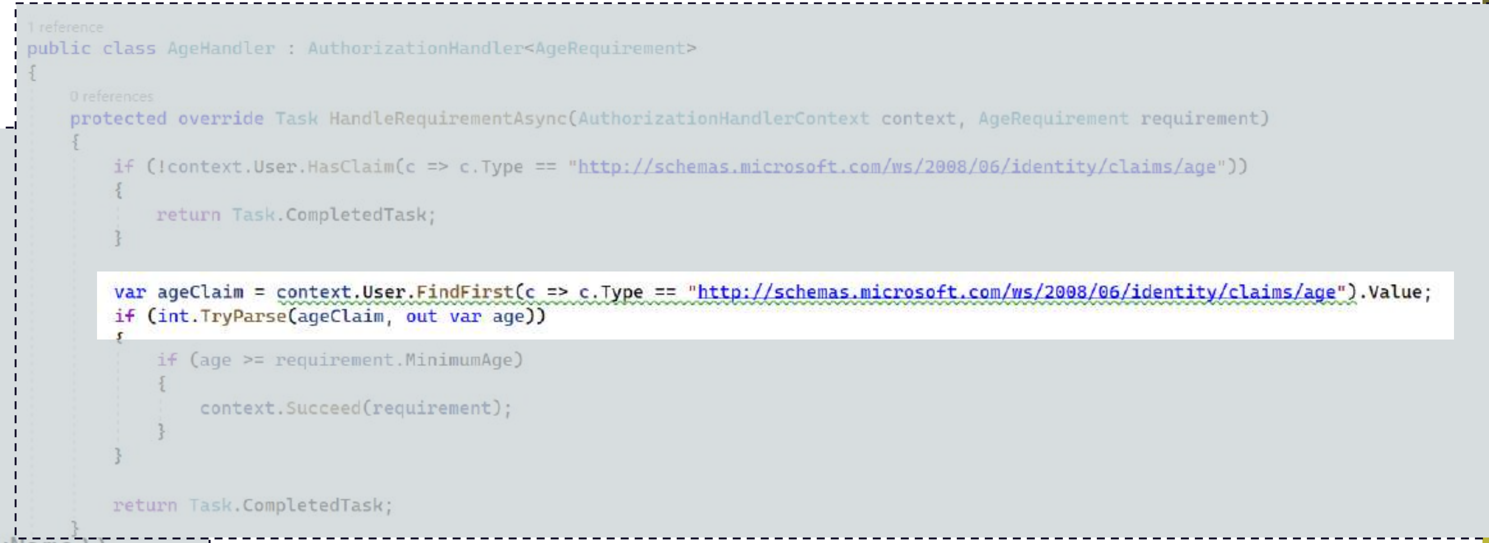
WITHOUT EXTERNALIZATION – ADDING MORE COMPLICATED POLICIES

- The more complex the requirement, the more complex the code.
- **All repeated, for every single method in every single application.** (In a small application, there are easily hundreds of methods.)



```
[Produces("application/json")]
[Route("api/bar")]
[Authorize(Roles = "customer")]
[Authorize(Policy = "Over18Only")]
0 references
public class BarController : Controller
{
    [HttpPost]
    0 references
    public IActionResult Post([FromBody] Drink drink)
    {
        if (drink == null || string.IsNullOrEmpty(drink.DrinkName))
        {
            return BadRequest("Invalid data.");
        }
```

A customer >= 18 can order alcoholic beverages



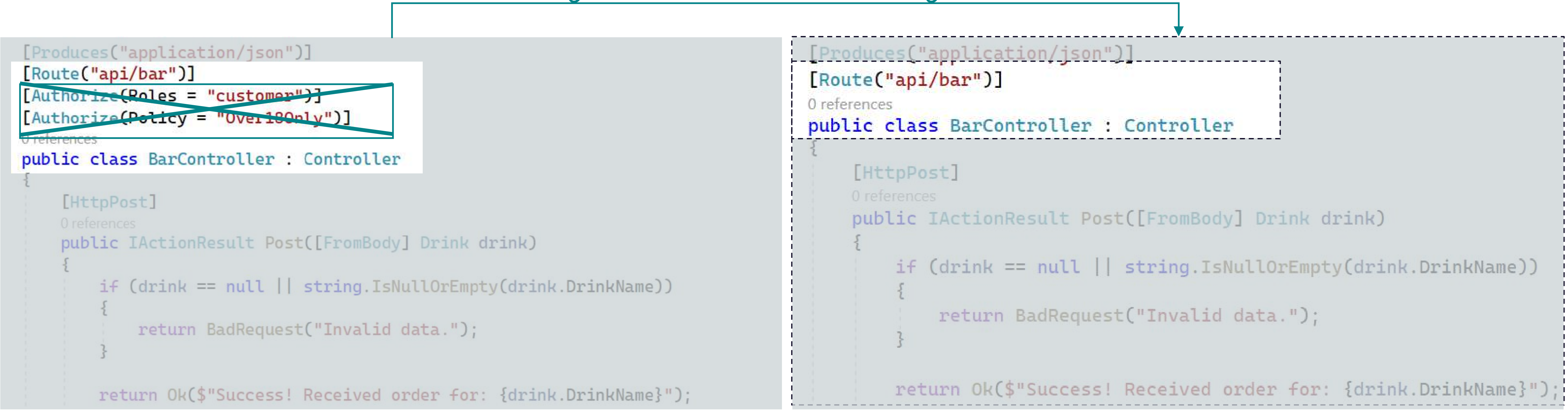
```
1 reference
public class AgeHandler : AuthorizationHandler<AgeRequirement>
{
    0 references
    protected override Task HandleRequirementAsync(AuthorizationHandlerContext context, AgeRequirement requirement)
    {
        if (!context.User.HasClaim(c => c.Type == "http://schemas.microsoft.com/ws/2008/06/identity/claims/age"))
        {
            return Task.CompletedTask;
        }

        var ageClaim = context.User.FindFirst(c => c.Type == "http://schemas.microsoft.com/ws/2008/06/identity/claims/age").Value;
        if (int.TryParse(ageClaim, out var age))
        {
            if (age >= requirement.MinimumAge)
            {
                context.Succeed(requirement);
            }
        }

        return Task.CompletedTask;
    }
```

Detailed policy to get the age claim from the token

HELLO EXTERNALIZED AUTHORIZATION

- Get rid of authorization logic in the application.
- Only one requirement remains: the application must pass via the authorization system.

We can get rid of ALL authorization logic in the code.



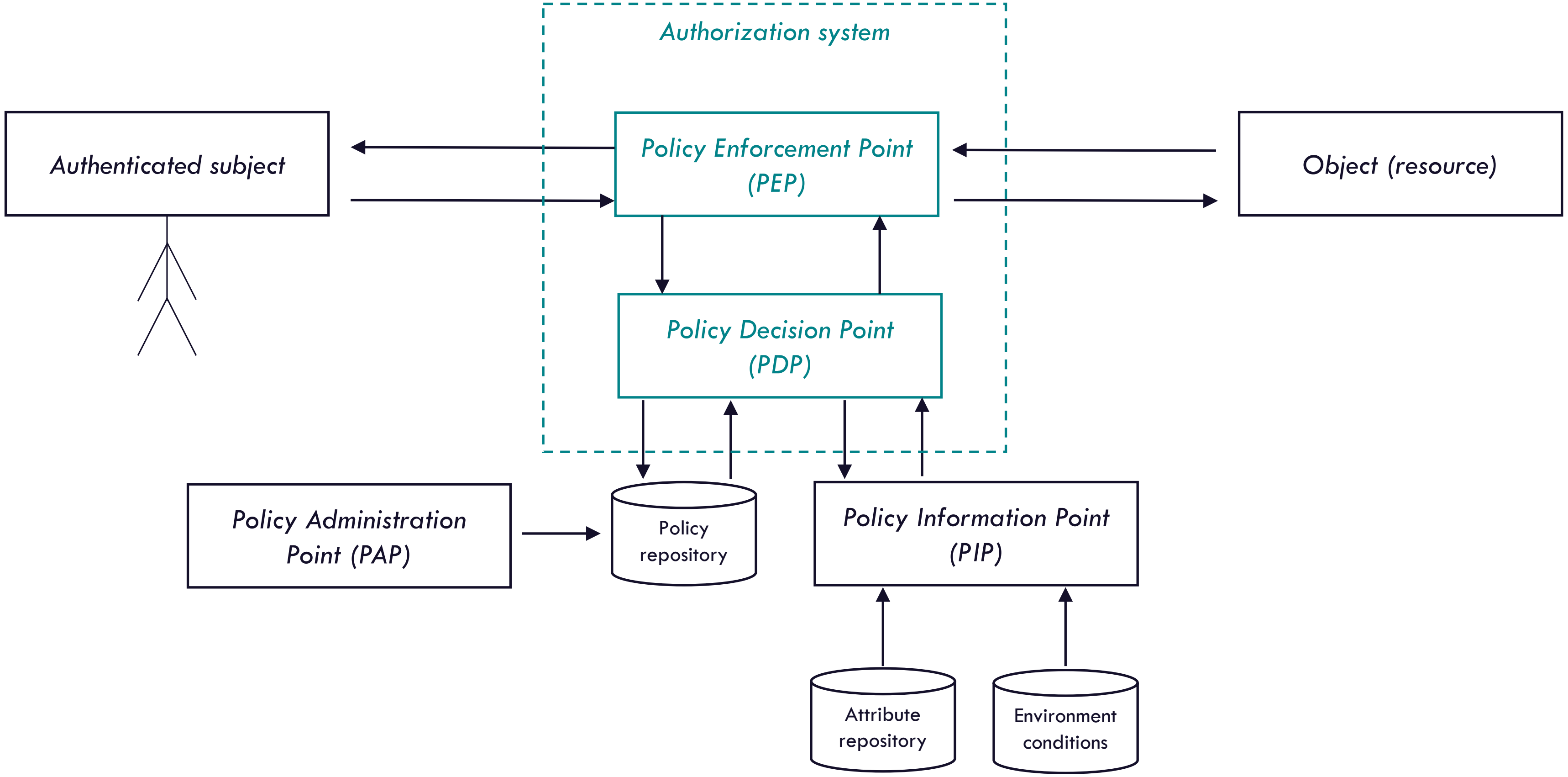
```
[Produces("application/json")]
[Route("api/bar")]
~~[Authorize(Roles = "customer")]~~
~~[Authorize(Policy = "Over18Only")]~~
0 references
public class BarController : Controller
{
    [HttpPost]
    0 references
    public IActionResult Post([FromBody] Drink drink)
    {
        if (drink == null || string.IsNullOrEmpty(drink.DrinkName))
        {
            return BadRequest("Invalid data.");
        }

        return Ok($"Success! Received order for: {drink.DrinkName}");
```

```
[Produces("application/json")]
[Route("api/bar")]
0 references
public class BarController : Controller
{
    [HttpPost]
    0 references
    public IActionResult Post([FromBody] Drink drink)
    {
        if (drink == null || string.IsNullOrEmpty(drink.DrinkName))
        {
            return BadRequest("Invalid data.");
        }

        return Ok($"Success! Received order for: {drink.DrinkName}");
```

DEMO TIME

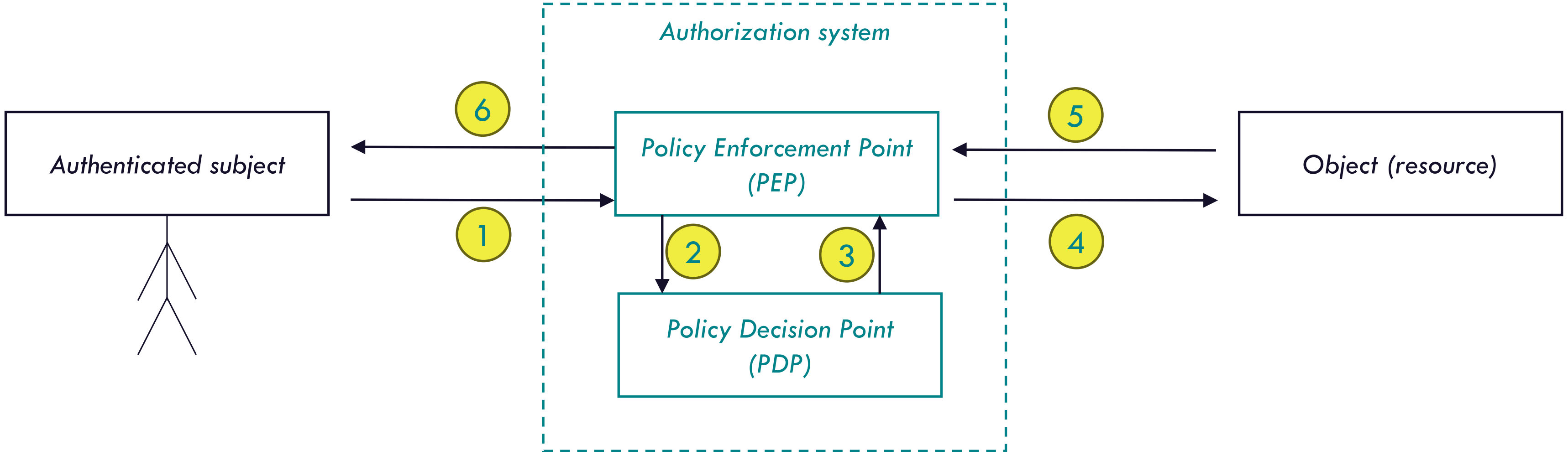
SPLYNTER

DEMO: EXTERNALIZE AUTHORIZATION


Authorization system
Authenticated subject
Policy Enforcement Point (PEP)
Object (resource)
Policy Decision Point (PDP)
Policy Administration Point (PAP)
Policy repository
Policy Information Point (PIP)
Attribute repository
Environment conditions

SPLYNTER

DEMO: EXTERNALIZE AUTHORIZATION



Authorization system

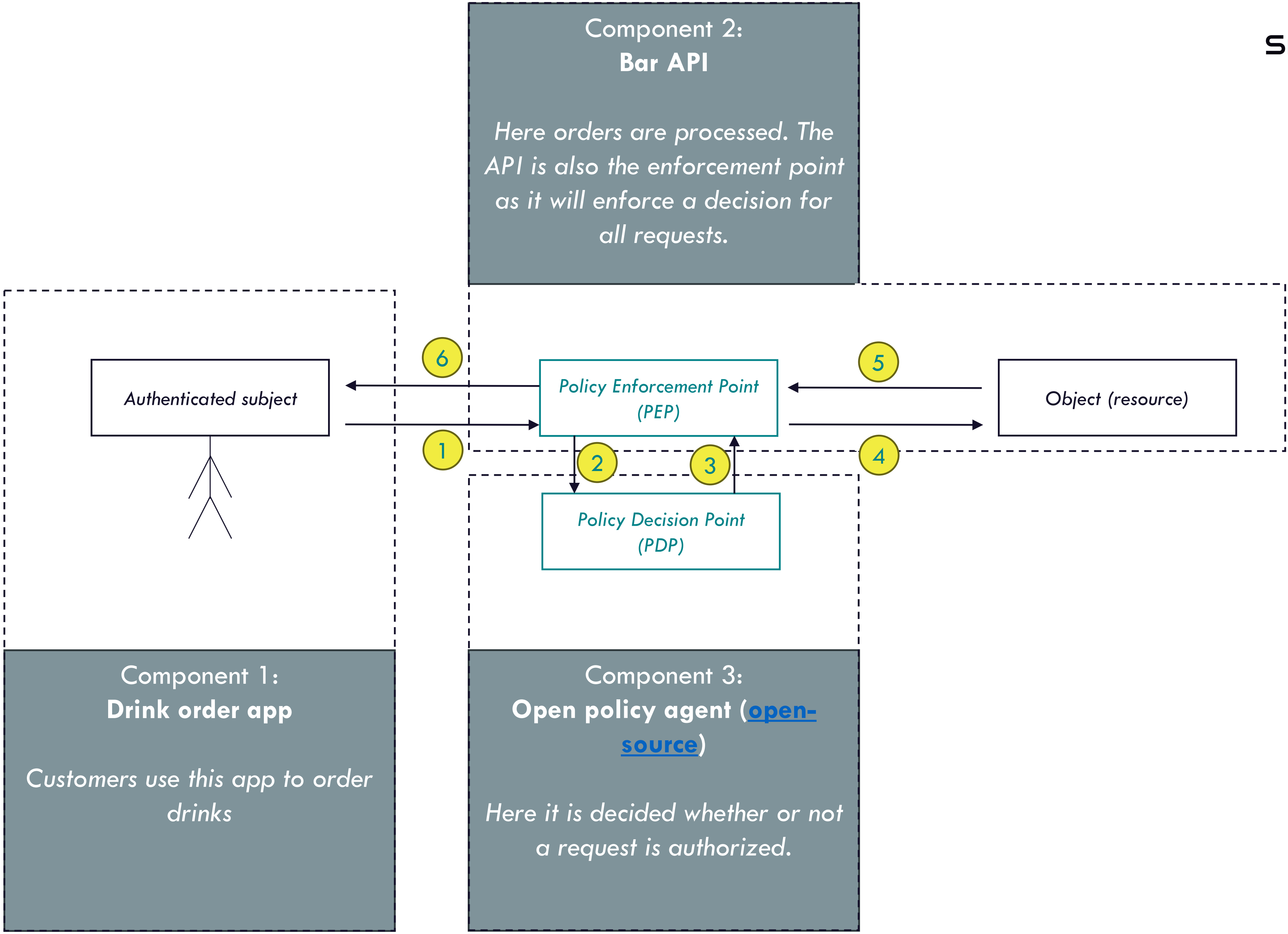
Authenticated subject

Policy Enforcement Point (PEP)

Policy Decision Point (PDP)

Object (resource)

SPLYNTER



Component 2:
Bar API

Here orders are processed. The API is also the enforcement point as it will enforce a decision for all requests.

Authenticated subject

Policy Enforcement Point (PEP)

Object (resource)

Policy Decision Point (PDP)

Component 1:
Drink order app

Customers use this app to order drinks

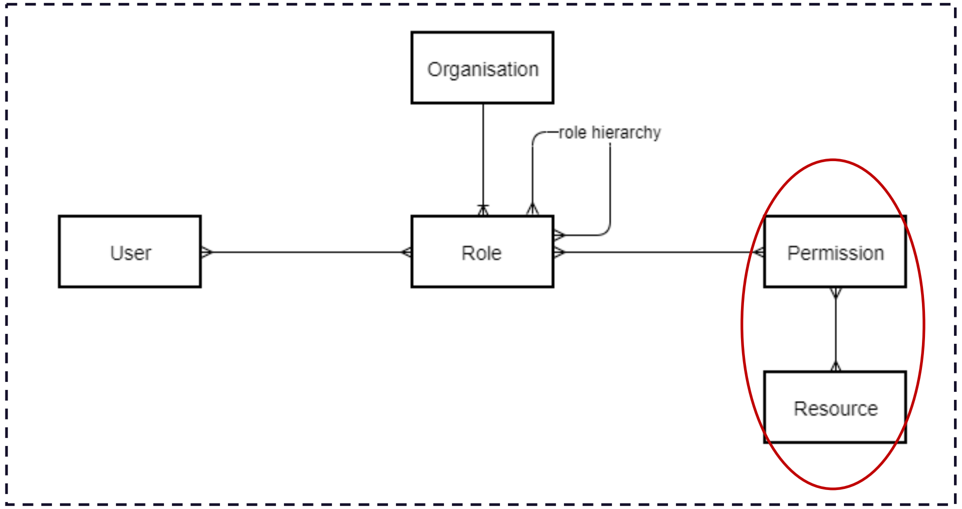
Component 3:
Open policy agent ([open-source](#))

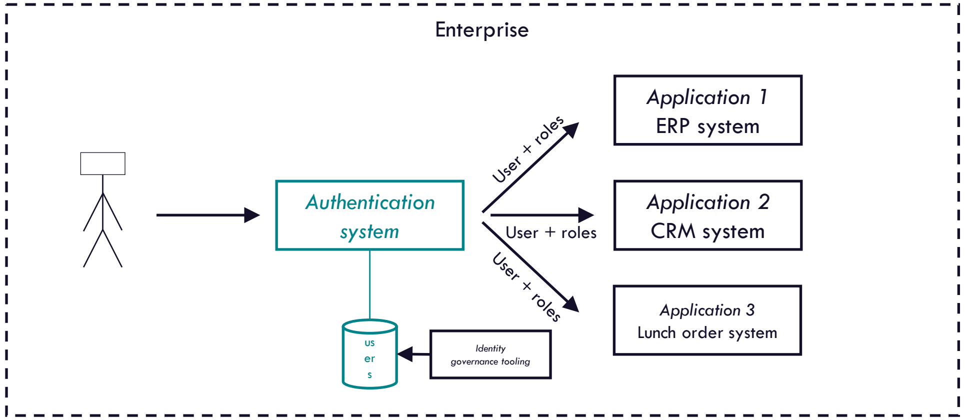
Here it is decided whether or not a request is authorized.

COMMON QUESTIONS

I'M USING SSO, DON'T I ALREADY HAVE CONTROL?

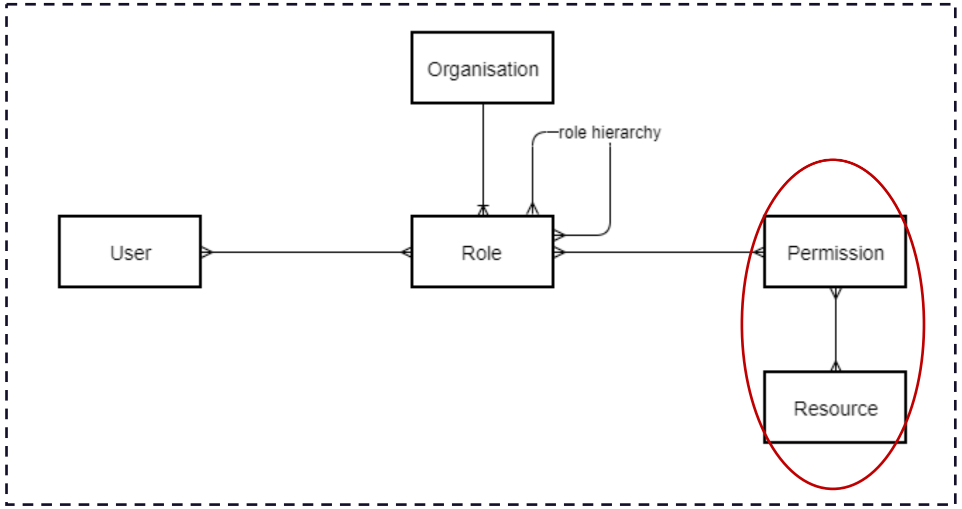
- ✓ You have control over your users.
- ✓ You have control over which users have which role.
- ✗ You have NO control over which users have which permissions in your applications.

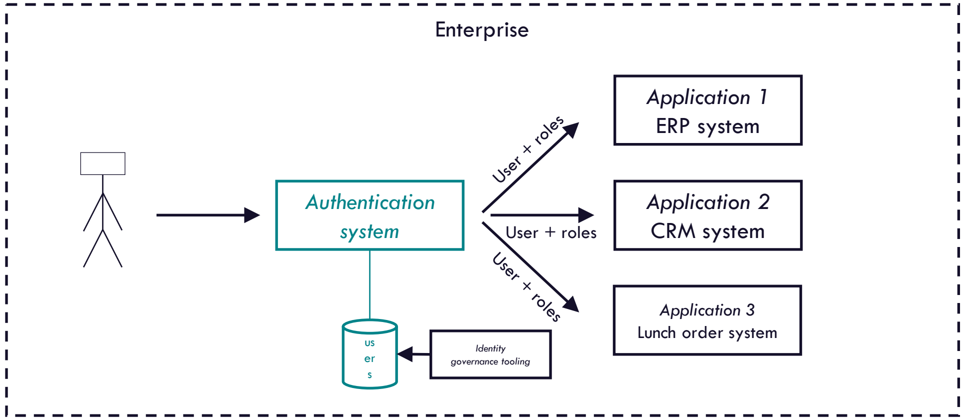




I'M USING CONDITIONAL ACCESS (E.G. MS ENTRA), DON'T I ALREADY HAVE CONTROL?

- ✓ You have control over your users.
- ✓ You have control over which users have which role.
- ✗ You have NO control over which users have which permissions in your applications.
- ✓ You have more control over your authentication: from which location is a user authenticating, at what time, etc.
- ✗ You have NO control over your authorization: has the user moved location after authentication, has the user copy pasted the authentication cookie onto another device, etc.?

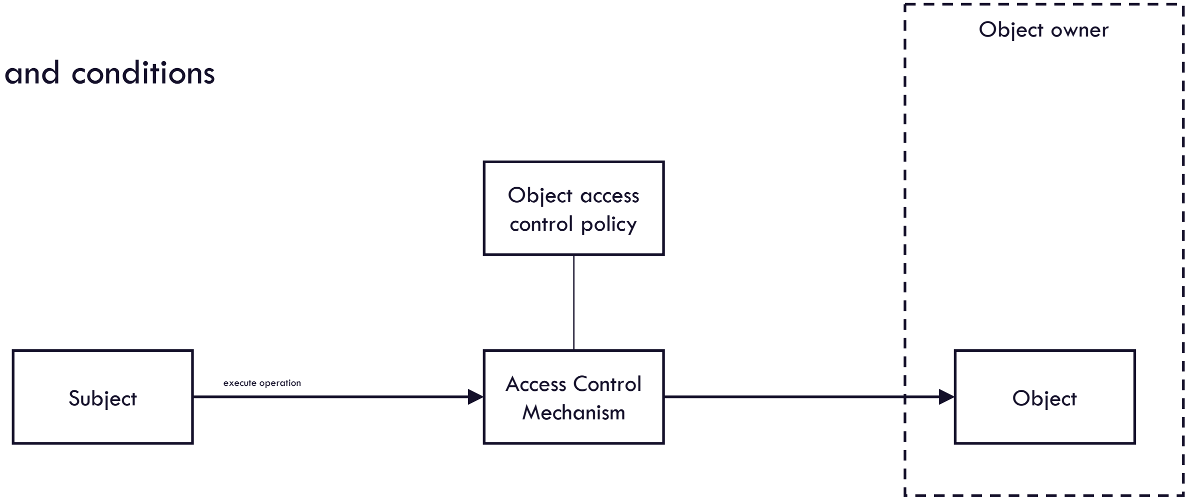




WHERE DOES ABAC FIT IN?

*Externalizing authorization works with any model (RBAC, ABAC) but makes **the transition to ABAC much simpler.***

- RBAC: An access control method where access is granted or denied based on role.
  - Permissions are grouped into roles
  - Users are assigned to a role
  - **Role explosion** is a real problem in many organizations.
- Attribute Based Access Control (ABAC)*: An access control method where access is granted or denied based on
  - assigned attributes of the subject
  - assigned attributes of the object
  - environment conditions
  - and a set of policies that are specified in terms of those attributes and conditions



* *"PBAC" is an acceptable synonym.*

WHAT ABOUT XACML?

- XACML has long been the only standard way to implement ABAC;
- However, **XACML is not an implementation,** it is a policy definition language which is implemented by vendors such as Axiomatics or Forgerock
- **OPA is an implementation,** which uses REGO as policy definition language.

	Open Policy Agent (OPA) with REGO	XACML
Language	REGO (declarative, query-based)	XML-based policy language (a JSON profile is available)
Ease of Use	Generally considered more straightforward and flexible	Often seen as more complex due to XML verbosity and reliance on vendor implementations
Integration	Easily integrates with modern cloud-native environments, Kubernetes, microservices	Integrates with enterprise systems; often used in traditional IT environments
Community and Support	Growing community, with increasing support and adoption in cloud-native ecosystems	Established in enterprise environments with robust support but less prevalent in newer tech stacks
Standards Compliance	Not a standard; more of a tool/framework	Standardized by OASIS, ensuring consistent implementation across different platforms
Learning Curve	Moderate, with a need to learn REGO	Steeper, due to the complexity of XACML policies

CONCLUSION

BENEFITS OF EXTERNALIZING AUTHORIZATION

+

- Consistency in access decisions (and therefore less risk of introducing **Broken Access Control** vulnerabilities);
- Reusability of policies across multiple applications and services in different technologies (Kubernetes, Terraform, Kafka, Java, .Net, …);
- Flexibility and scalability since changes in authorization policies do not require a redeploy of applications;
- Modularity since business logic and security logic are decoupled (although the line between the two is sometimes very thin);
- Compliance through centralized policy monitoring and better audit trails;
- Future-readiness given the possibility to transition to ABAC easily (if required);

-

- Requires a high maturity in the access control domain;
- May introduce delays, although these can be limited using smart caching strategies.

*Note that **centralized** policy management does not necessarily mean introducing a single point of failure. OPA allows for having multiple PDPs, **managed** centrally.*

WHO ARE WE

Michael Boeynaems

Jasper Rots

Founder of Splynter
A cyber security company specialized in vendor-independent and risk-based consultancy, where we combine high-quality enterprise security architecture with in-depth technical cyber security expertise.

Lector at AP Hogeschool Antwerpen
Teaching software security and cyber security advanced.

Certifications
CISSP, OSCP, CISM, CEH, Archimate practitioner

Contact
michael.boeynaems@splynter.be

Security Architect at Splynter
IT security manager and SSDLC process owner, true to Splynter's value of making the complex simple through deep understanding, structure and passion.

Educator and trainer
Former KU Leuven TA and SE teacher
Nowadays teaching software security, cyber security regulation, network security and cryptography.

Contact
jasper.rots@splynter.be

Feel free to contact us for gaining access to the code used in this demo, or for additional information.

QUESTIONS?