



**Companion deck to:**

<https://awssecworkshops.com/workshops/identity-round-robin/permission-boundaries/>

# Identity Round Robin Workshop

## Permission Boundaries

Speaker

Title

Amazon Web Services (AWS)



# Agenda

- Intro
  - Permission boundary basics
  - Policy categories
  - Permission boundary mechanism
  - Resource restrictions
  - Q & A
- 
- Workshop
  - Final Q & A

# What are permission boundaries?

Mechanism to delegate the **permission to create users and roles** while **preventing privilege escalation** or unnecessarily broad permissions.

Method to safely grant actions like:

"iam:CreatePolicy"

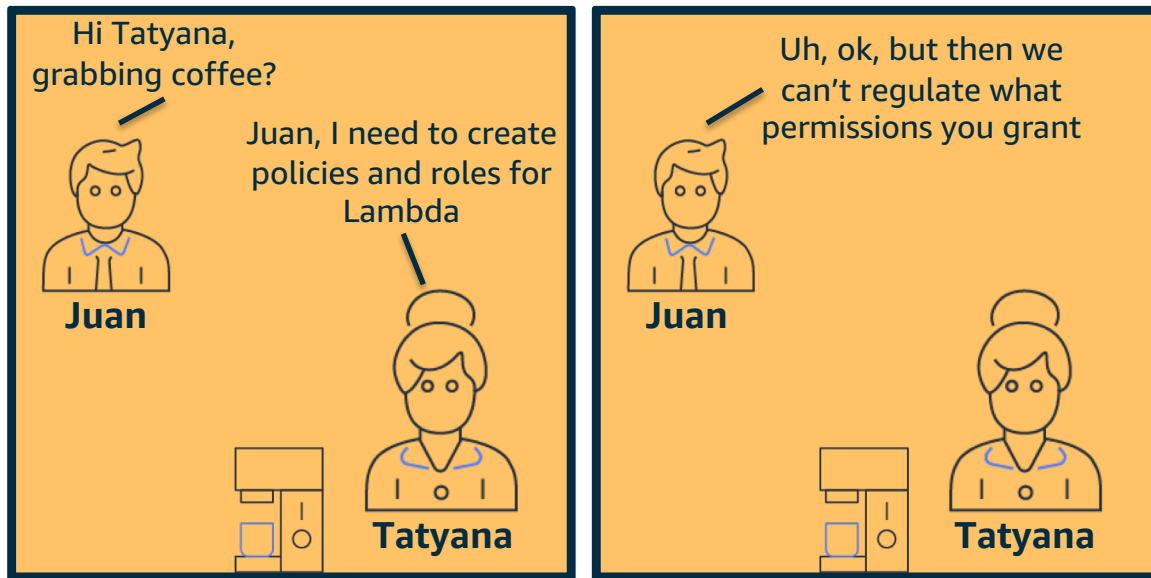
"iam:CreateRole"

"iam:PassRole"

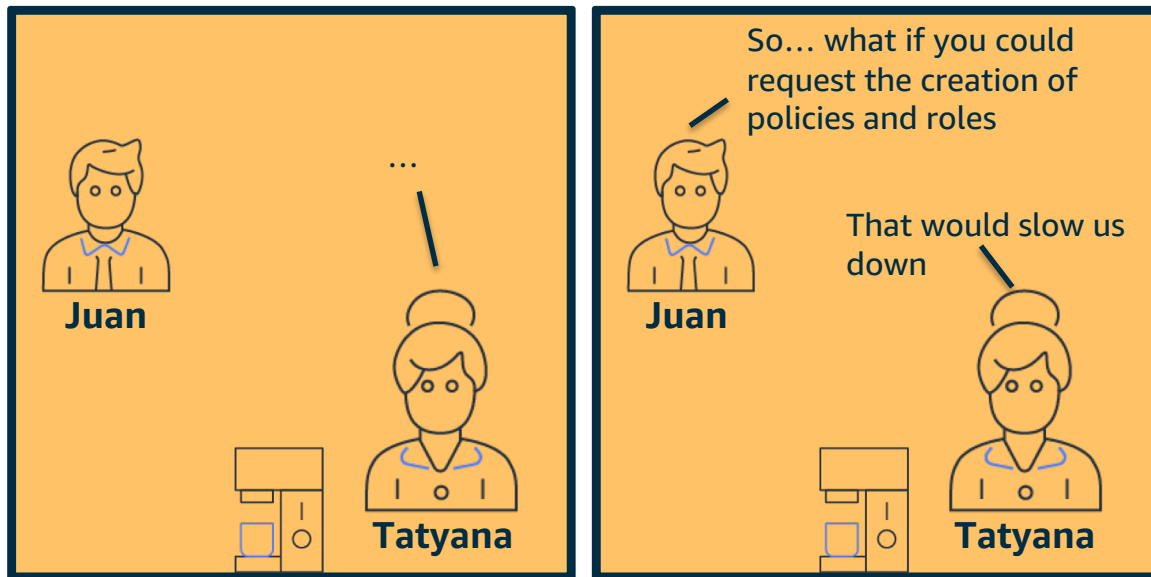
# Use cases

- Developers that need to create roles for Lambda functions
- Application owners that need to create roles for EC2 instances
- Admins that need to be able to create users for particular use cases
- Any others?

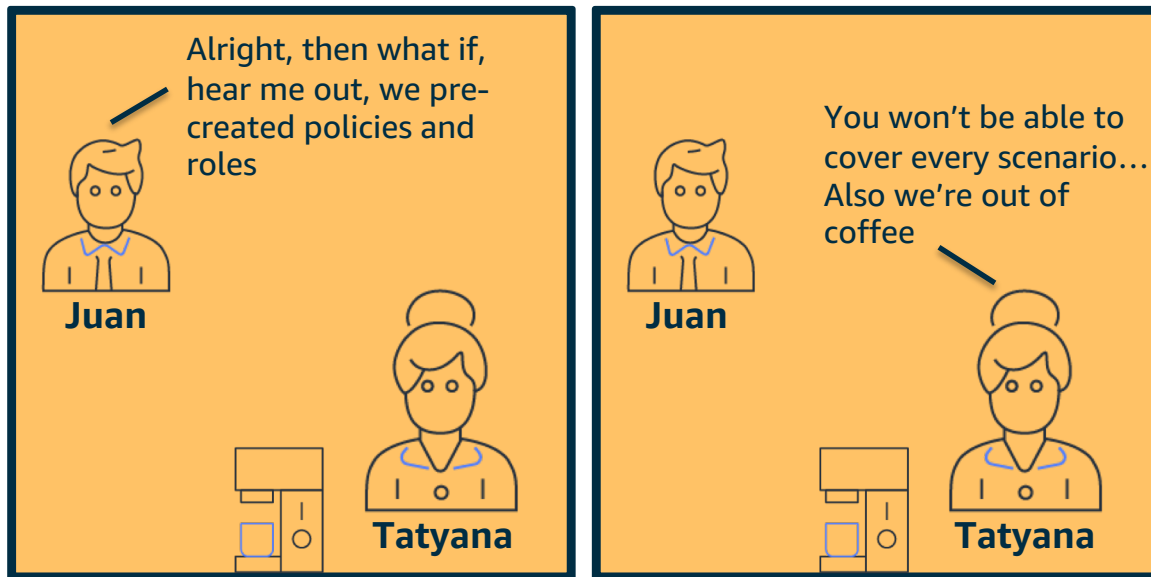
# Two viewpoints – the problem



# Two viewpoints – old solution 1



# Two viewpoints – old solution 2



# Permission boundary basics



# Before and After Permission Boundaries

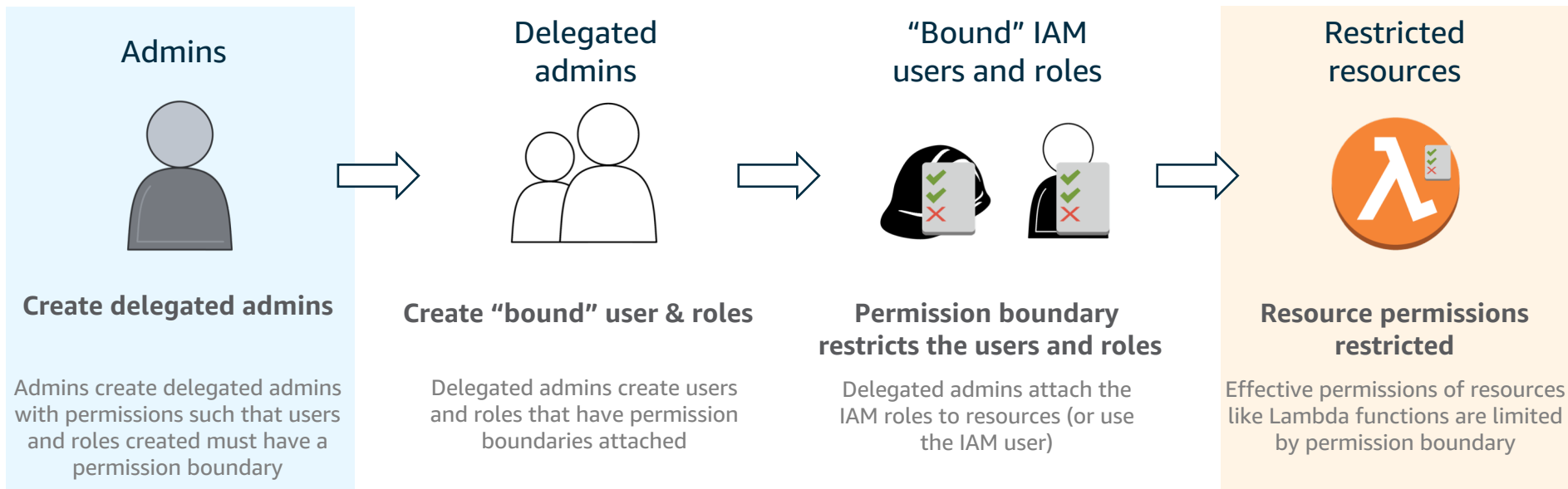
## Before

- Certain IAM policy actions (e.g. PutUserPolicy, AttachRolePolicy) are **essentially full admin-like permissions**.
- Doing any form of self-service permissions management **was non-trivial**.

## Now

- Administrator can grant full admin-like permissions, but **specify a "permissions boundary."**
- **Allow developers to create principals** for their applications and attach policies, but only **within the boundary**.

# Permission Boundaries – mechanism



# A condition

```
"Condition": {"StringEquals":  
  {"iam:PermissionsBoundary":  
    "arn:aws:iam::ACCOUNT_ID:policy/permissionboundary"  
  }  
}
```

# A condition applied to principal creation actions (users and roles)

```
"Effect": "Allow",
"Action": ["iam:CreateRole"],
"Resource": ["arn:aws:iam::ACCOUNT_ID:role/path/"],
"Condition": {"StringEquals":
  {"iam:PermissionsBoundary":
    "arn:aws:iam::ACCOUNT_ID:policy/permissionboundary"
  }
}
```

# Mechanism

Developer creates a role for a Lambda function

## # Step 1: Create role

```
$ aws iam create-role --role-name roleforlambda  
--assume-role-policy-document file://Role_Trust_Policy_Text.json  
--permissions-boundary arn:aws:iam::<ACCOUNT_NUMBER>:policy/department_a/boundary_1
```

## # Step 2: Create policy

No change

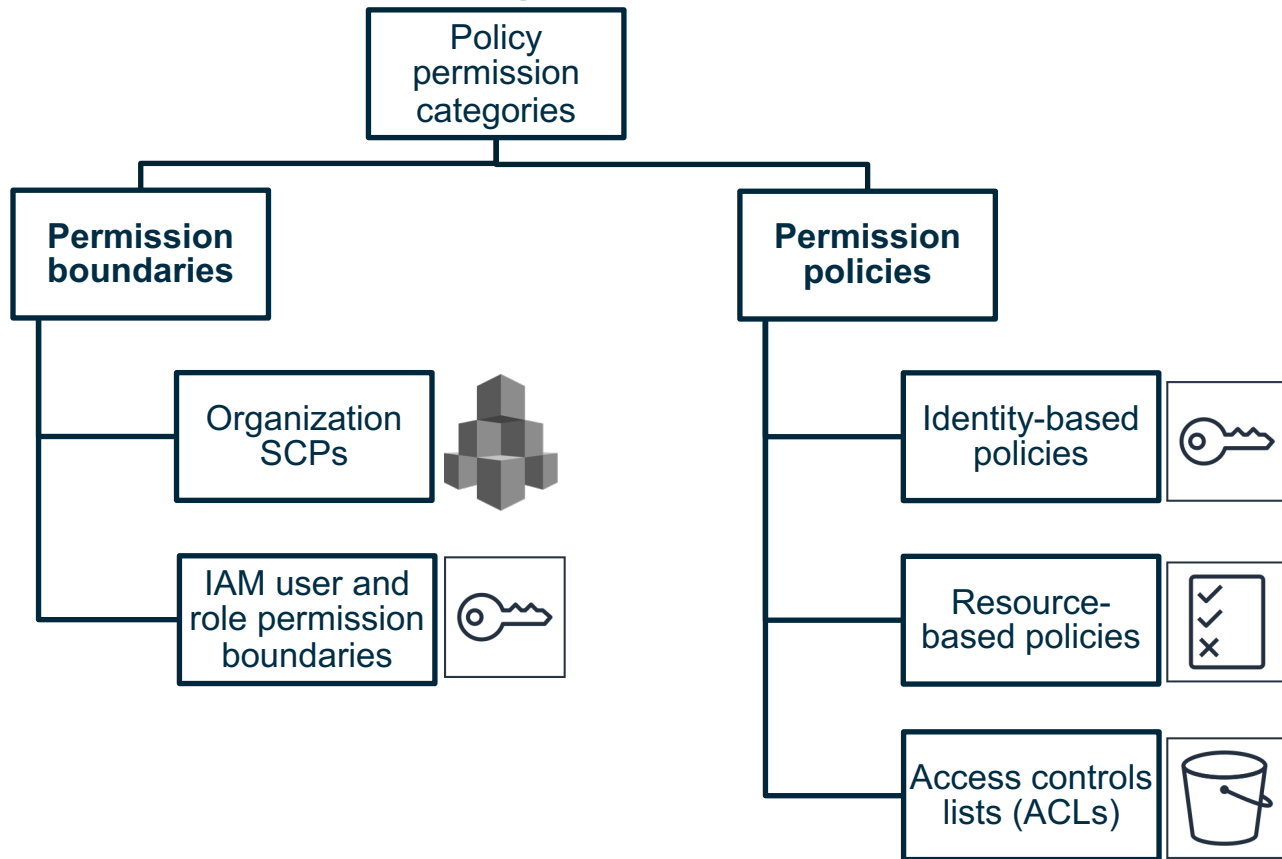
## # Step 3: Attach policy

No change

# Demo

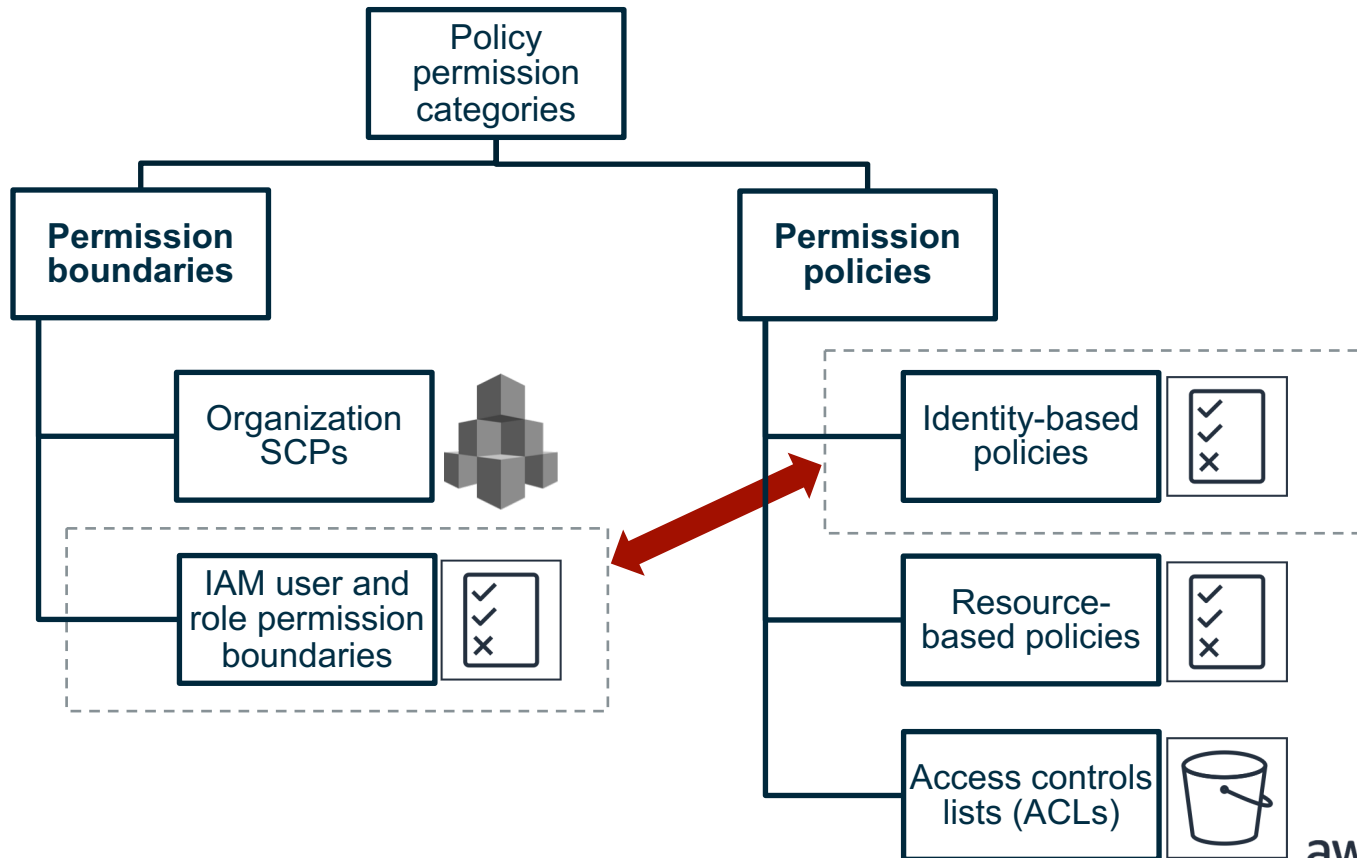
# Policy categories

# Policy permission categories

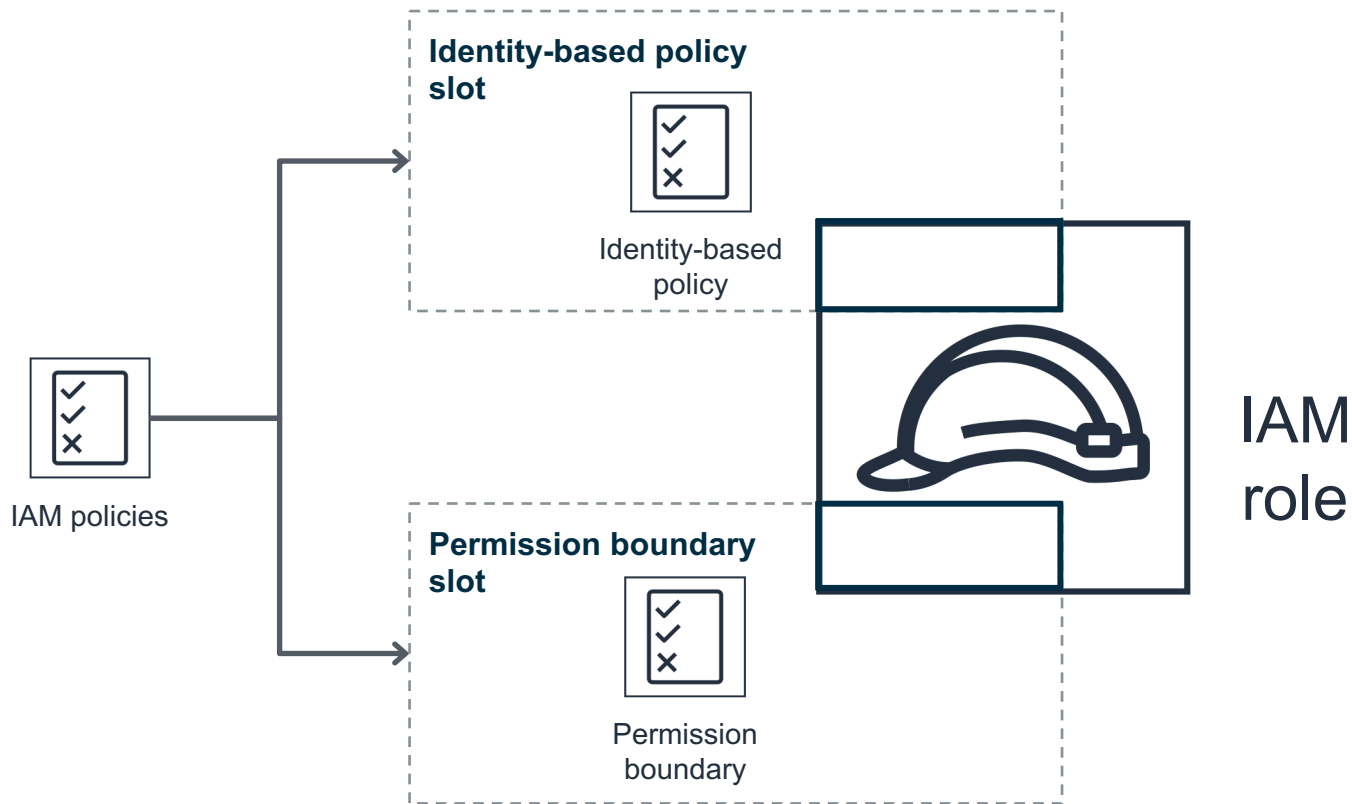




# Are these two things the same?



# But, it's just an IAM policy right?



# But, it's just an IAM policy right?

▼ Attach permissions policies

Choose one or more policies to attach to your new role.

Create policy

Identity-based policy slot

Filter policies ▼

Showing 582 results

	Policy name ▼	Used as	Description
<input type="checkbox"/>	▶ AdministratorAccess	Permissions policy (9)	Provides full access to AWS services an...
<input type="checkbox"/>	▶ AlexaForBusinessDeviceSetup	None	Provide device setup access to AlexaFor...
<input type="checkbox"/>	▶ AlexaForBusinessFullAccess	None	Grants full access to AlexaForBusiness r...
<input type="checkbox"/>	▶ AlexaForBusinessGatewayExecution	None	Provide gateway execution access to AI...
<input type="checkbox"/>	▶ AlexaForBusinessReadOnlyAccess	None	Provide read only access to AlexaForBu...
<input type="checkbox"/>	▶ AllowAssumeDeleteDDBRole	None	
<input type="checkbox"/>	▶ AllowDeleteofDDBTable	Permissions policy (1)	
<input type="checkbox"/>	▶ AmazonAPIGatewayAdministrator	None	Provides full access to create/edit/delete...

▼ Set permissions boundary

Set a permissions boundary to control the maximum permissions this role can have. This is an advanced feature used to delegate permission management to others. [Learn more](#)

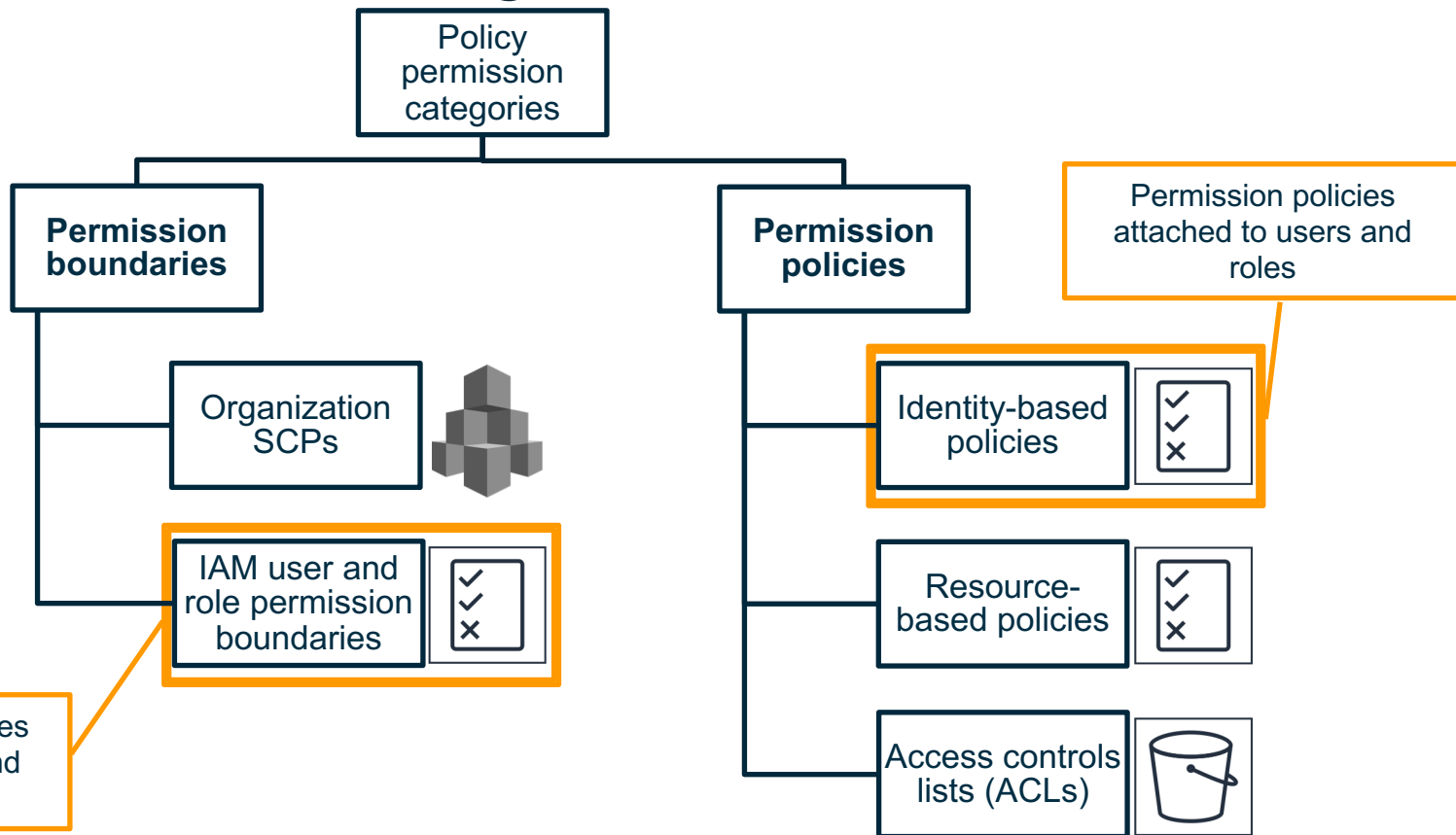
☒ Create role without a permissions boundary

☐ Use a permissions boundary to control the maximum role permissions

Permission boundary slot

# Permission boundary mechanism

# Policy permission categories



# Everything after authentication

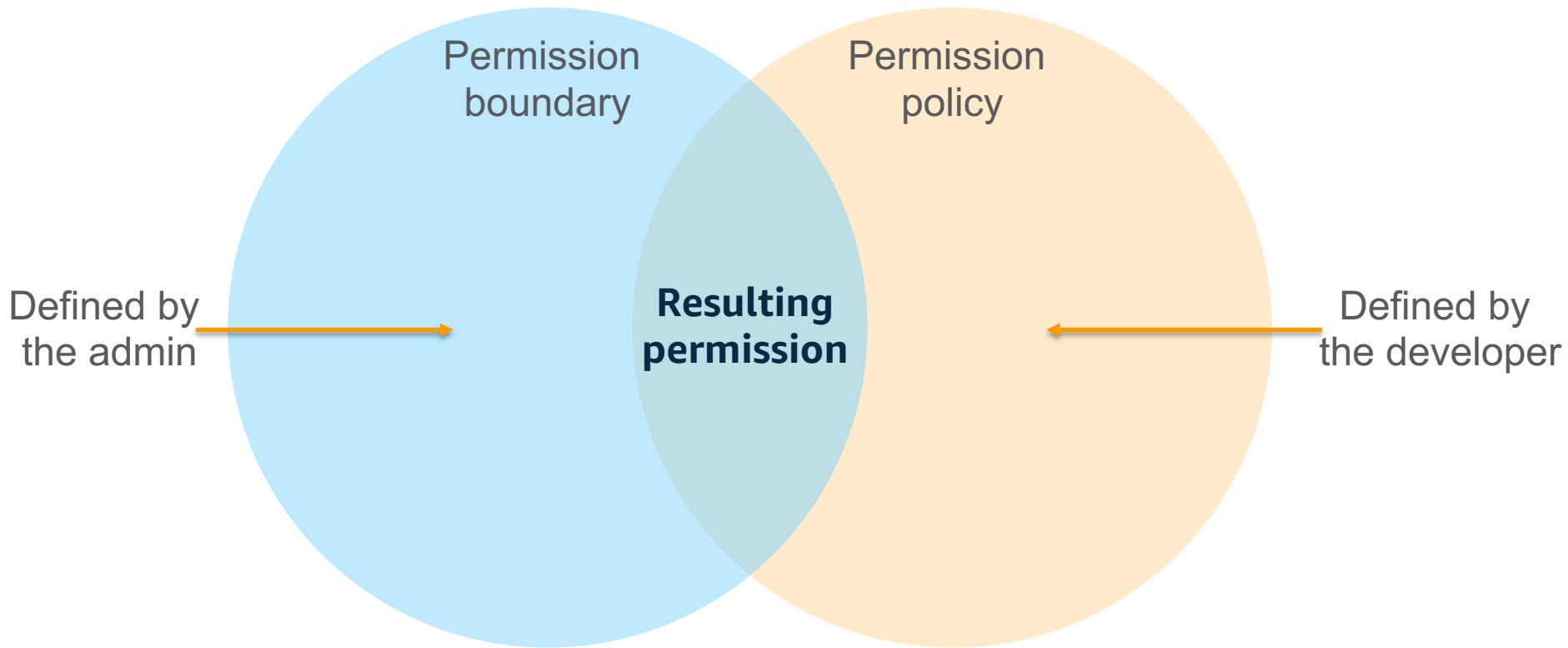
1. **Authenticate** the principal

2. Determine which **policies** apply to the request

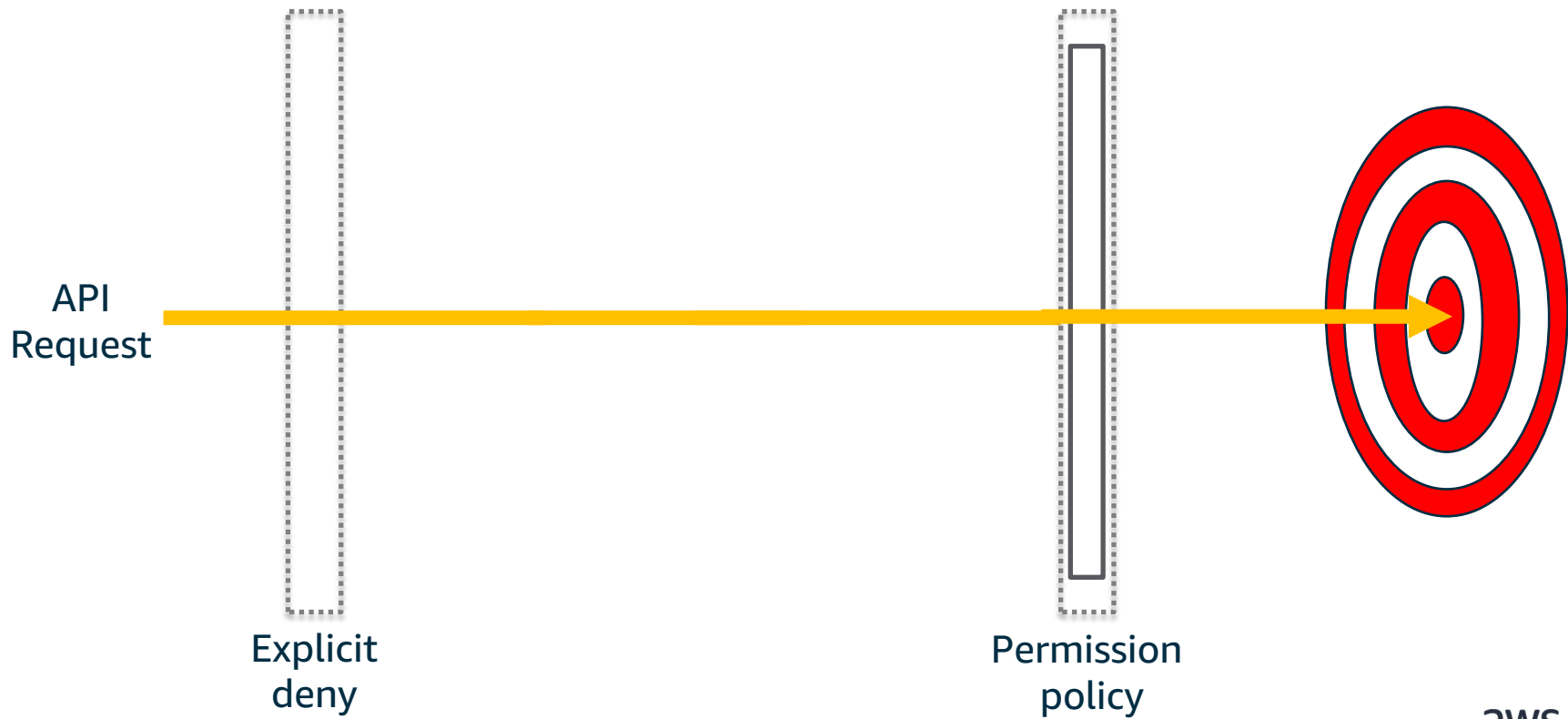
3. **Evaluate** the different policy types that apply which affect the order in which they are evaluated.

4. **Allow or Deny** the request

# Effective Permissions - intersection



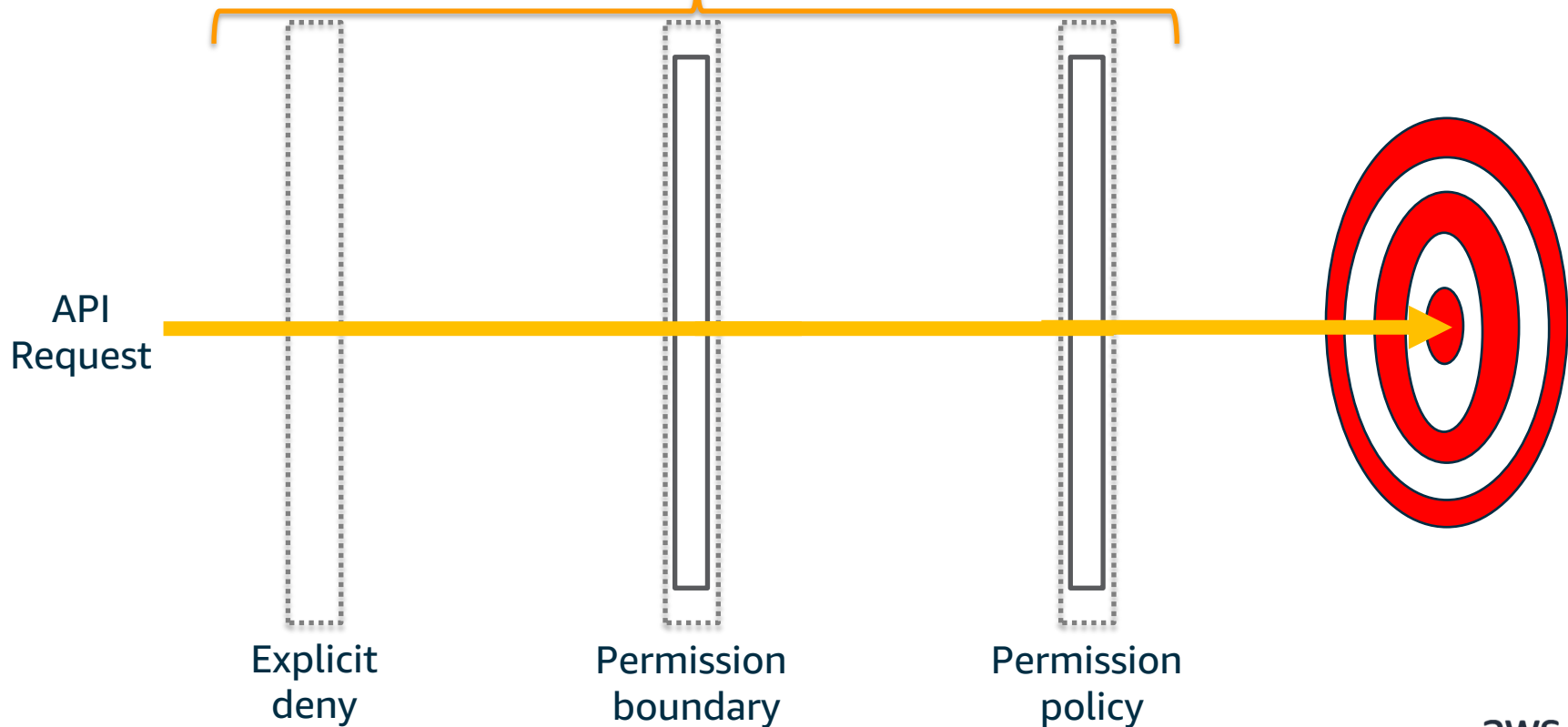
# Effective permissions – mechanism



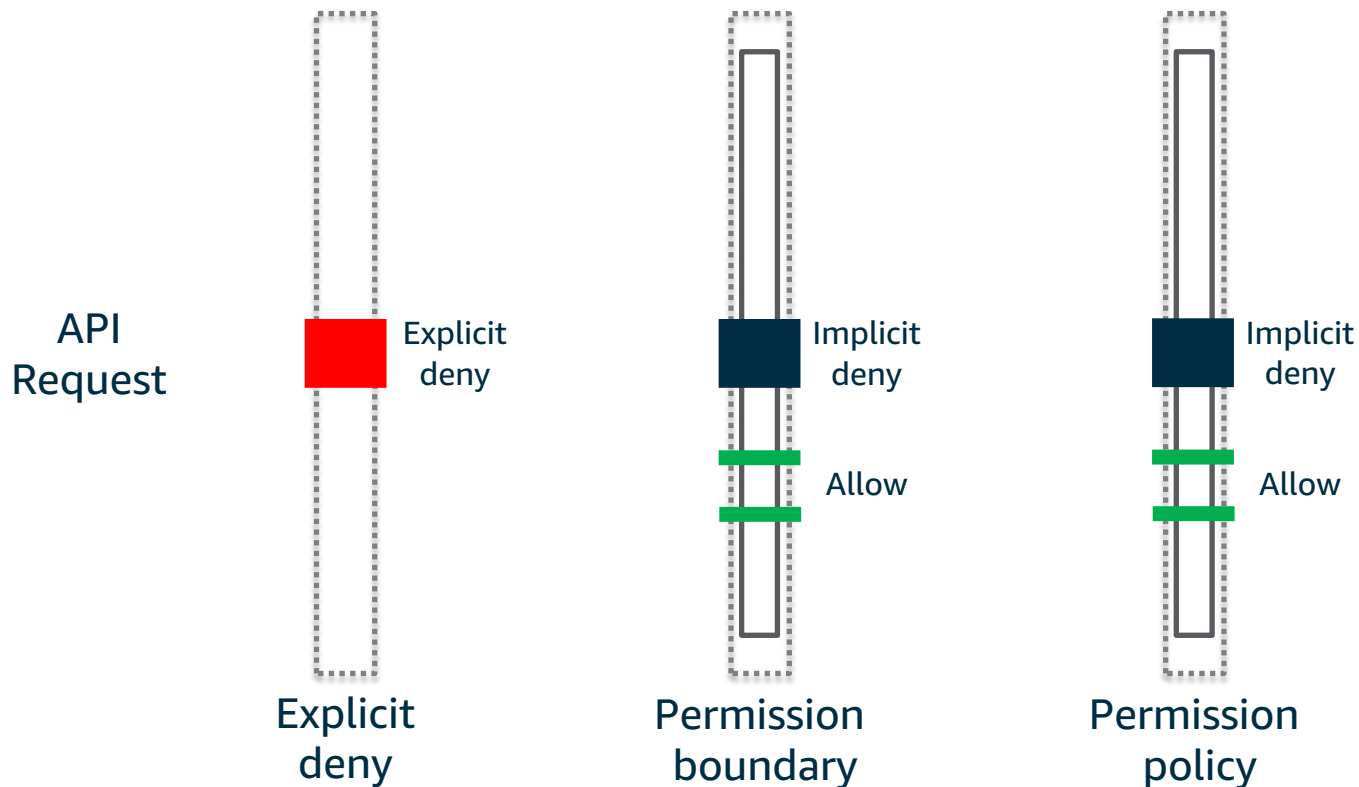


# Effective permissions – mechanism

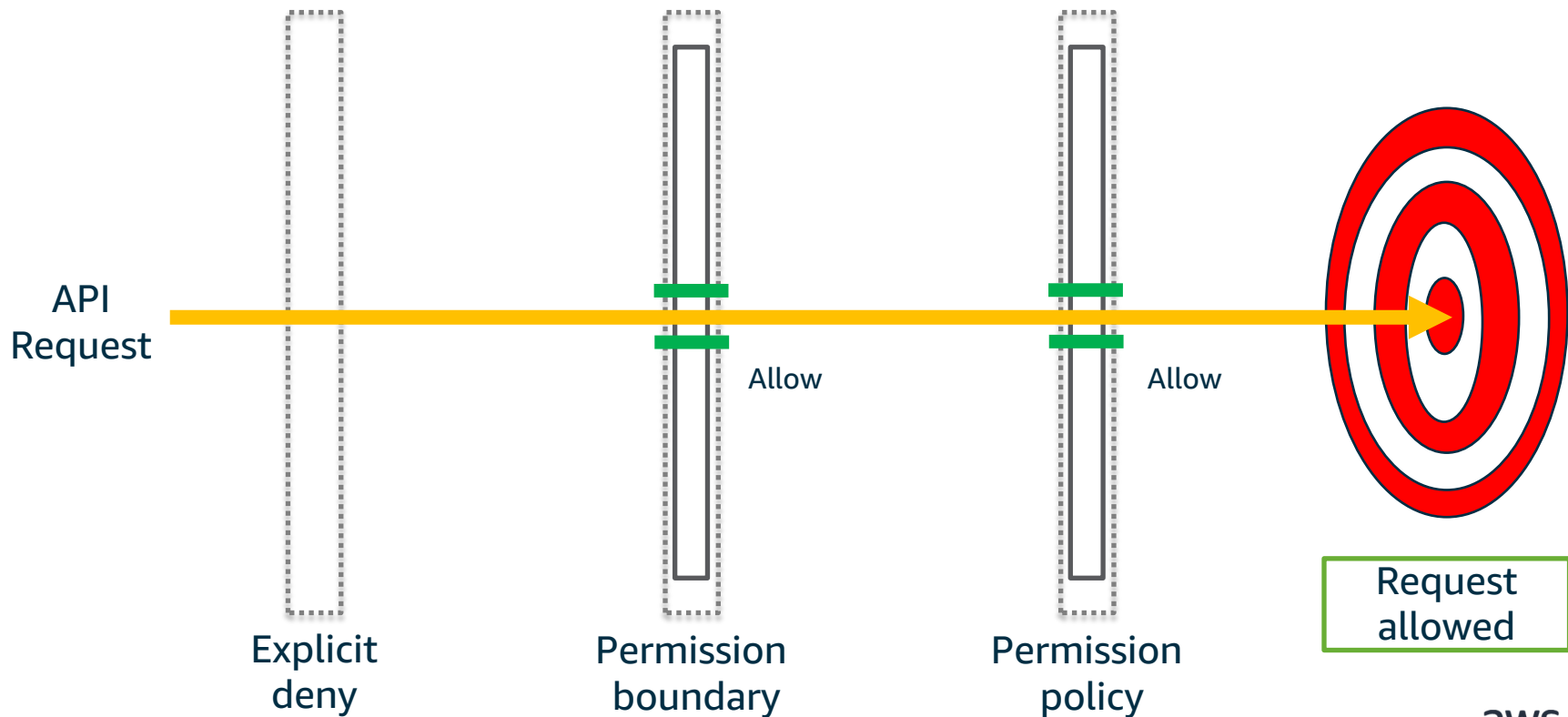
Resulting permission



# Effective permissions – mechanism



# Effective permissions – allow example



# Effective permissions – scenario 1

Request: s3:GetObject / bucket name: example1

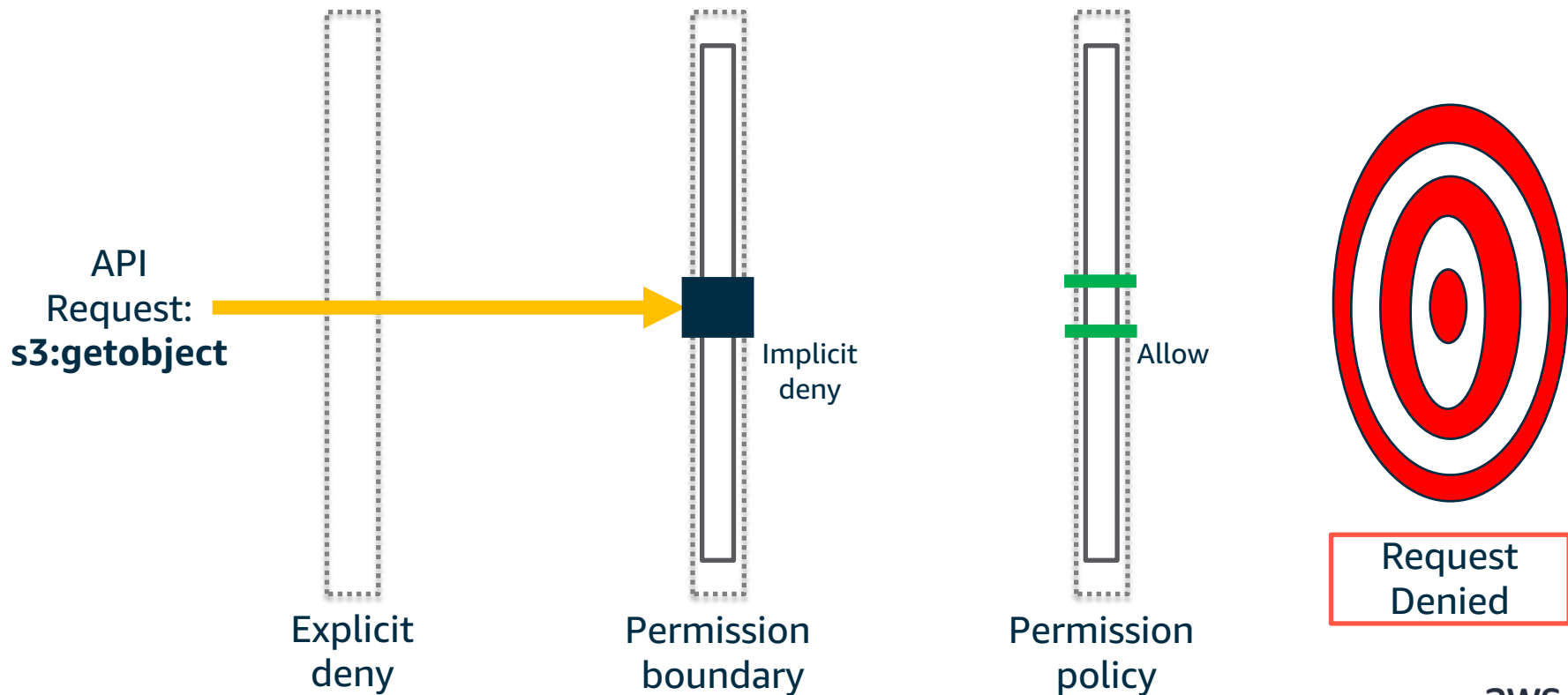
## Permission Boundary

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "arn:aws:logs:*:*:*"
    }
  ]
}
```

## Permission Policy

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "s3:*"
      ],
      "Resource": "*"
    }
  ]
}
```

# Effective permissions – result



# Effective permissions – scenario 2

Request: s3:GetObject / bucket name: example1

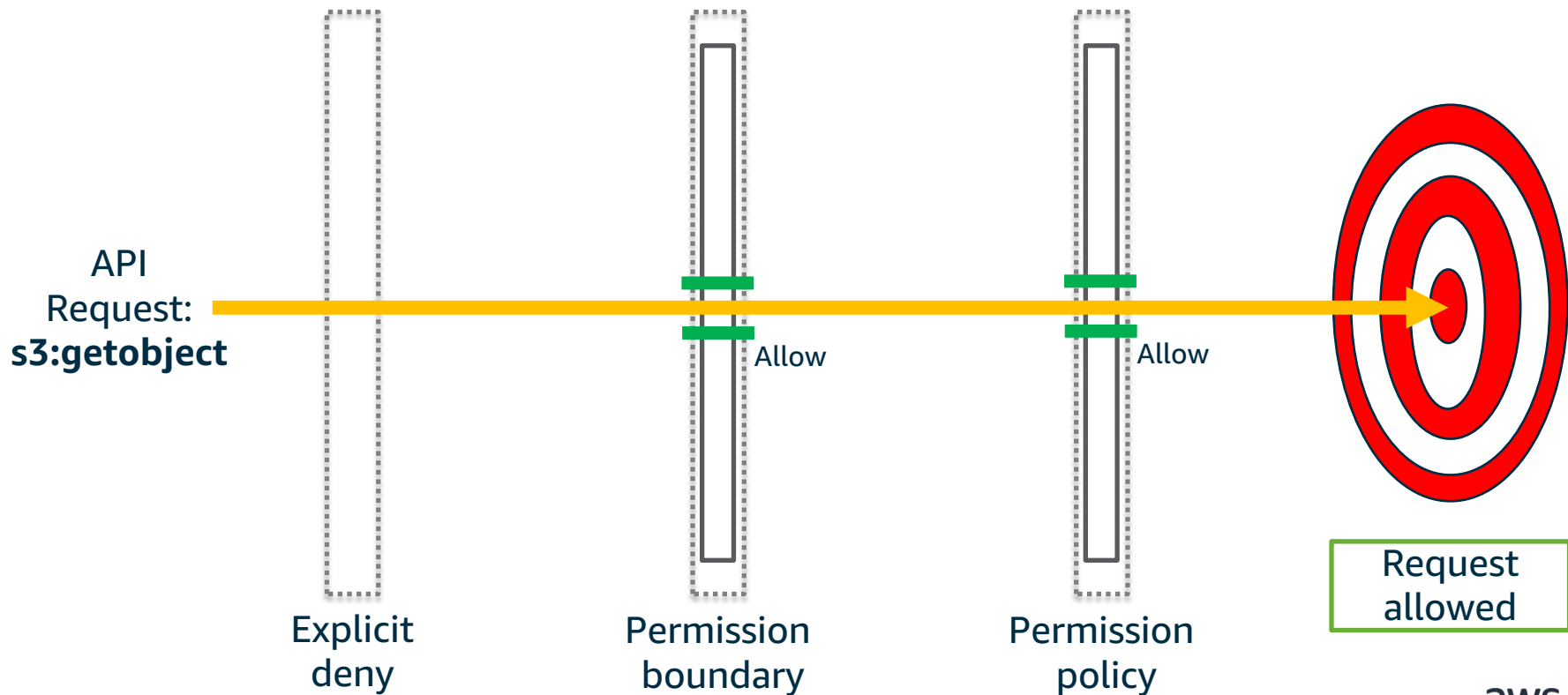
## Permission Boundary

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "arn:aws:logs:*:*:*"
    },
    {
      "Effect": "Allow",
      "Action": ["s3:GetObject"],
      "Resource": "arn:aws:s3:::example1/*"
    }
  ]
}
```

## Permission Policy

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "s3:*"
      ],
      "Resource": "*"
    }
  ]
}
```

# Effective permissions – result



# Effective permissions – scenario 3

Request: s3:GetObject / bucket name: example1

## Permission Boundary

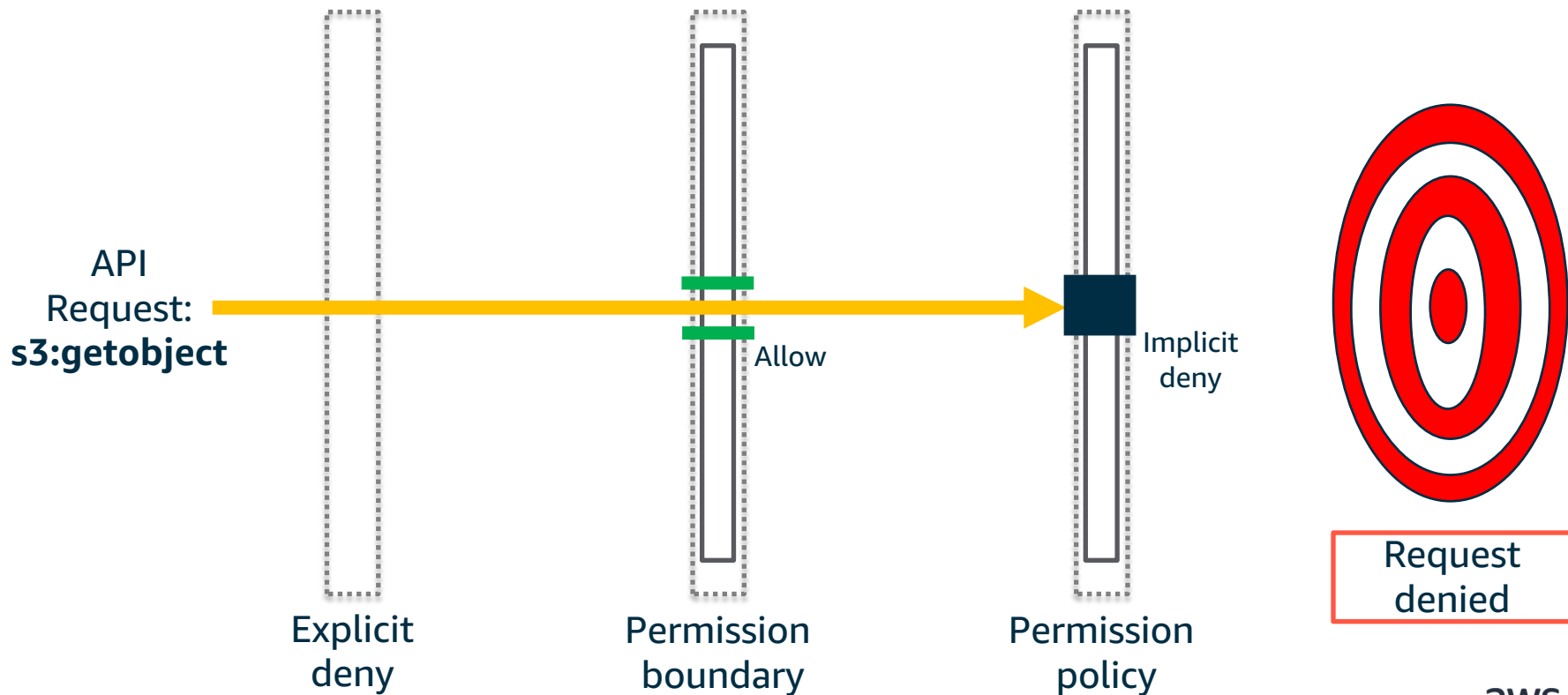
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "arn:aws:logs:*:*:*"
    },
    {
      "Effect": "Allow",
      "Action": ["s3:GetObject"],
      "Resource": "arn:aws:s3:::example1/*"
    }
  ]
}
```

## Permission Policy

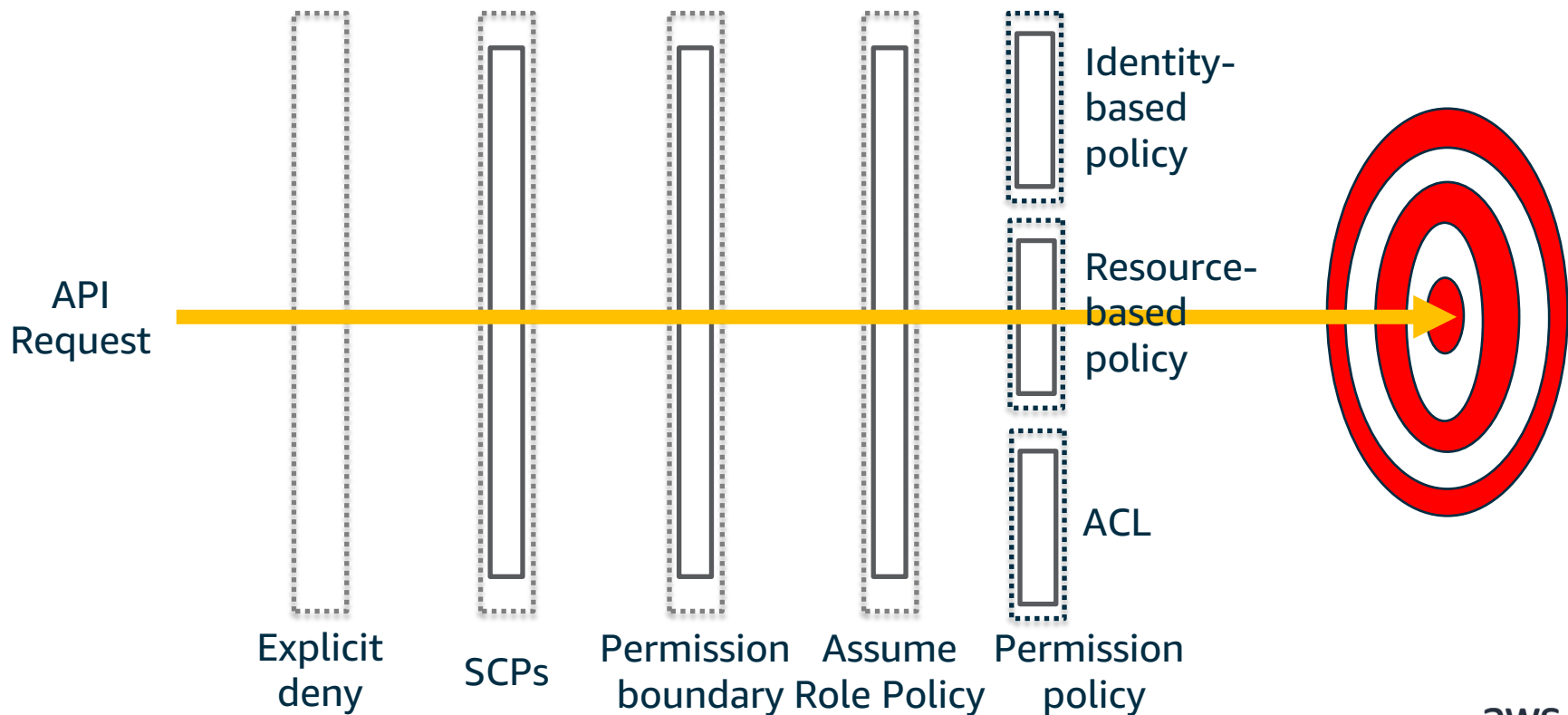
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "*"
    }
  ]
}
```



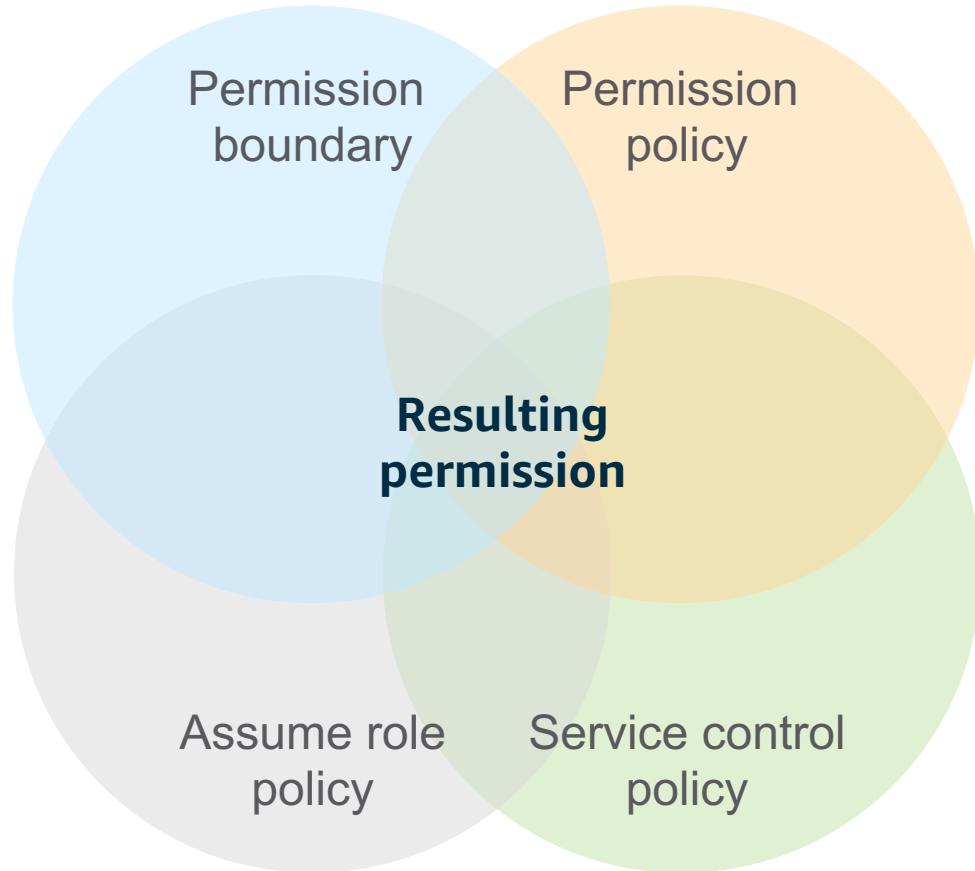
# Effective permissions – result



# Effective permissions – mechanism expanded



# Effective Permissions – intersection expanded



# Resource Restrictions

Goal: carve out a space for the delegated admins to be able to modify resources without impacting other resources.

Paths are preferred by request the CLI. Naming (name\*) can also be used.

<https://docs.aws.amazon.com/general/latest/gr/aws-arns-and-namespaces.html#arns-paths>

[https://docs.aws.amazon.com/IAM/latest/UserGuide/reference\\_identifiers.html#identifiers-arns](https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_identifiers.html#identifiers-arns)

# Resource Restrictions

- We are granting admin-like permissions (create and delete policies and roles)
- Permission boundary is one part of the delegation
- The other part is restricting the policies and roles they can impact then they could

```
"Effect": "Allow",  
"Action": [  
    "iam:CreatePolicy",  
    "iam:DeletePolicy",  
    "iam:CreatePolicyVersion",  
    "iam:DeletePolicyVersion",  
    "iam:SetDefaultPolicyVersion"  
],  
"Resource": "*/"
```

# Resource Restrictions - examples

## Resource restriction using paths:

arn:aws:iam::123456789012:role/**department1/\***

**Example role:** arn:aws:iam::123456789012:role/**department1/role1**

## Resource restriction using names:

arn:aws:iam::123456789012:policy/**development-users\***

**Example policy:** arn:aws:iam::123456789012:policy/**development-users-policy1**

# Resource Restrictions - policies

- Examine permissions assigned to a delegated admin to create policies
- If there is not a resource restriction then the delegated admins could modify any customer managed policies

```
"Effect": "Allow",  
"Action": [  
    "iam:CreatePolicy",  
    "iam:DeletePolicy",  
    "iam:CreatePolicyVersion",  
    "iam:DeletePolicyVersion",  
    "iam:SetDefaultPolicyVersion"  
],  
"Resource": ""
```

VS

```
"Effect": "Allow",  
"Action": [  
    "iam:CreatePolicy",  
    "iam:DeletePolicy",  
    "iam:CreatePolicyVersion",  
    "iam:DeletePolicyVersion",  
    "iam:SetDefaultPolicyVersion"  
],  
"Resource":  
    "arn:aws:iam::ACCOUNT_ID:policy/path/name"
```

# Resource Restrictions - roles

- Just like with policies we want to carve out a safe space for roles.
- Permission boundaries play a part here, but not all actions support the condition
- In addition different teams could be using the same permission boundaries

```
"Effect": "Allow",  
"Action": [  
    "iam:UpdateRole",  
    "iam:DeleteRole"  
],  
"Resource": ""
```

VS

```
"Effect": "Allow",  
"Action": [  
    "iam:UpdateRole",  
    "iam:DeleteRole"  
],  
"Resource":  
    "arn:aws:iam::ACCOUNT_ID:role/path/name*"
```



# Presentation Q & A

# Presentation Q & A

- How does a permission boundary differ from a standard IAM policy?
- What would happen if we delegated permissions without resource restrictions?
- The scenario where you have user in an account that need to be able to create IAM polices, roles and Lambda functions is common. How was this situation handled before permission boundaries came along?

# Final Q & A

# Final Q & A

- What is the fundamental mechanism of permission boundaries?
- Why do we not allow the web admins to attach any role to the Lambda functions?
- There are two ways of doing resource restrictions (naming and pathing.) Which option allows you to create policies using both the AWS Console and CLI?
- Can an IAM policy be used as both a permission boundary and a permission policy?  
Is this a good practice?
- Is there an advantage to using one over the other?