AWS re:Invent

SEC316

Become an IAM Policy Master in 60 Minutes or Less

Brigid Johnson
Senior Manager of Product Management
AWS Identity

Agenda

- Recap of IAM policy language
- Policy types and how they work together
- Deep dive on policy with specific use cases
  - Set permission guardrails across accounts
  - Control creation of resources to specific regions
  - Enable developers to create roles safely
  - Use tags to scale permissions management

Breakout repeats

Tuesday, Nov 27

Become an IAM Policy Master in 60 Minutes or Less

7pm | Aria West, Level 3, Ironwood 5

Related breakouts

Tuesday, November 27th

Sec301 - The Theory and Math Behind Data Privacy and Security Assurance
10:00 am- 11:00 am Venetian, Level 2, Titian 2204

Wednesday, Nov 28, 11:30 AM - 12:30 PM

IAM for Enterprises: How Vanguard Matured IAM Controls to Support Micro AccountsTime
11:30 AM - 12:30 PM Mirage, Grand Ballroom F

Recap of IAM policy language

Quick recap of IAM policies

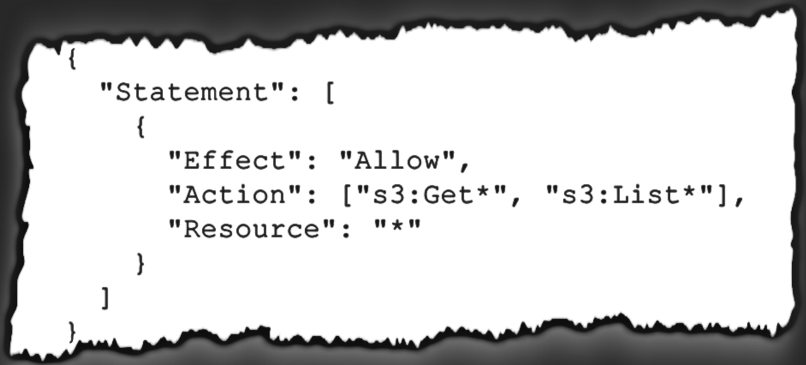
- What are IAM policies?
- IAM policy structure
- IAM policy evaluation rules
- How to think about IAM policy evaluation (new!)

What are IAM policies?

Policies provide authorization to AWS services and resources

Two parts:

- **Specification**: *Defining* access policies
- **Enforcement**: *Evaluating* policies



```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:Get*", "s3:List*"],
      "Resource": "*"
    }
  ]
}
```

When you *define* access policies. You specify which IAM principals are allowed to perform which actions on specific AWS resources and under which conditions.

IAM enforces this access by *evaluating* the AWS request and the policies you defined and returns either yes or no answer.

IAM policy structure

```
{
 "Statement":[{
  "Effect":"effect",
  "Principal":"principal",
  "Action":"action",
  "Resource":"arn",
  "Condition":{
   "condition":{
    "key":"value" }
    }
   }
  ]
 }
```

Principal – The entity that is allowed or denied access

"Principal":"AWS":"arn:aws:iam::123456789012:user/username"

Action - Type of access that is allowed or denied access

"Action":"s3:GetObject"

Resource – The Amazon resource(s) the action will act on

"Resource":"arn:aws:sqs:us-west-2:123456789012:queue1"

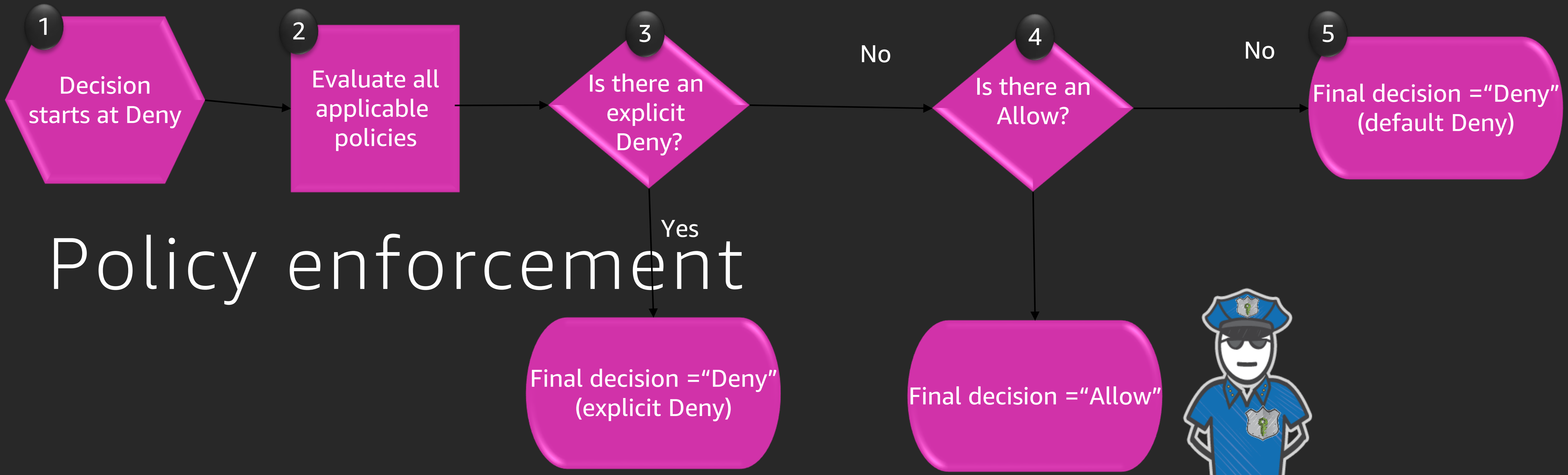
Condition – The conditions under the access defined is valid

"StringEqualsIfExists": {"aws:RequestTag/project": [“Pickles“]}

AWS re:Invent

aws

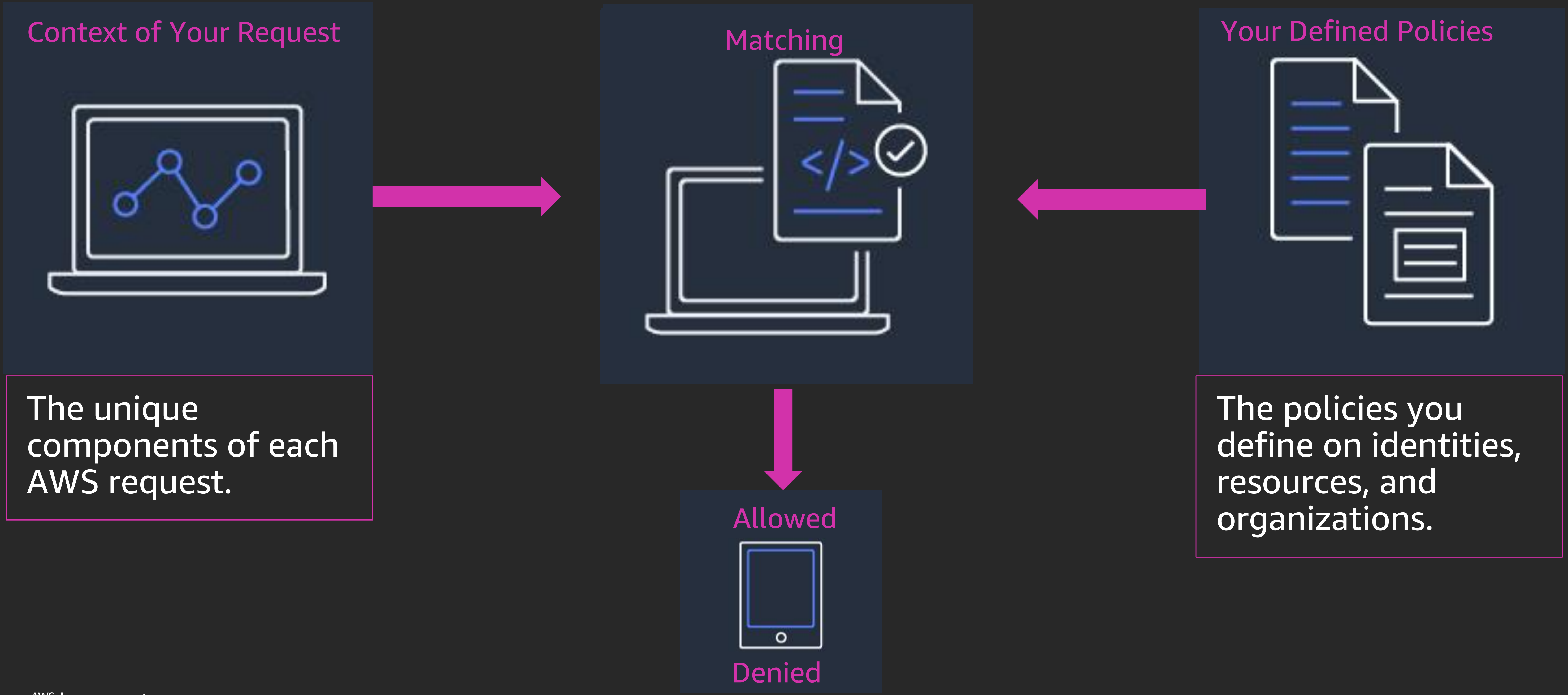
IAM policy evaluation



1
Decision starts at Deny
2
Evaluate all applicable policies
3
Is there an explicit Deny?
No
4
Is there an Allow?
No
5
Final decision ="Deny" (default Deny)
Yes
Final decision ="Deny" (explicit Deny)
Final decision ="Allow"

Policy enforcement

Context and policies – *a new way to think about evaluation*



The unique components of each AWS request.

The policies you define on identities, resources, and organizations.

Policy types and how they work together

Policy types and core use cases

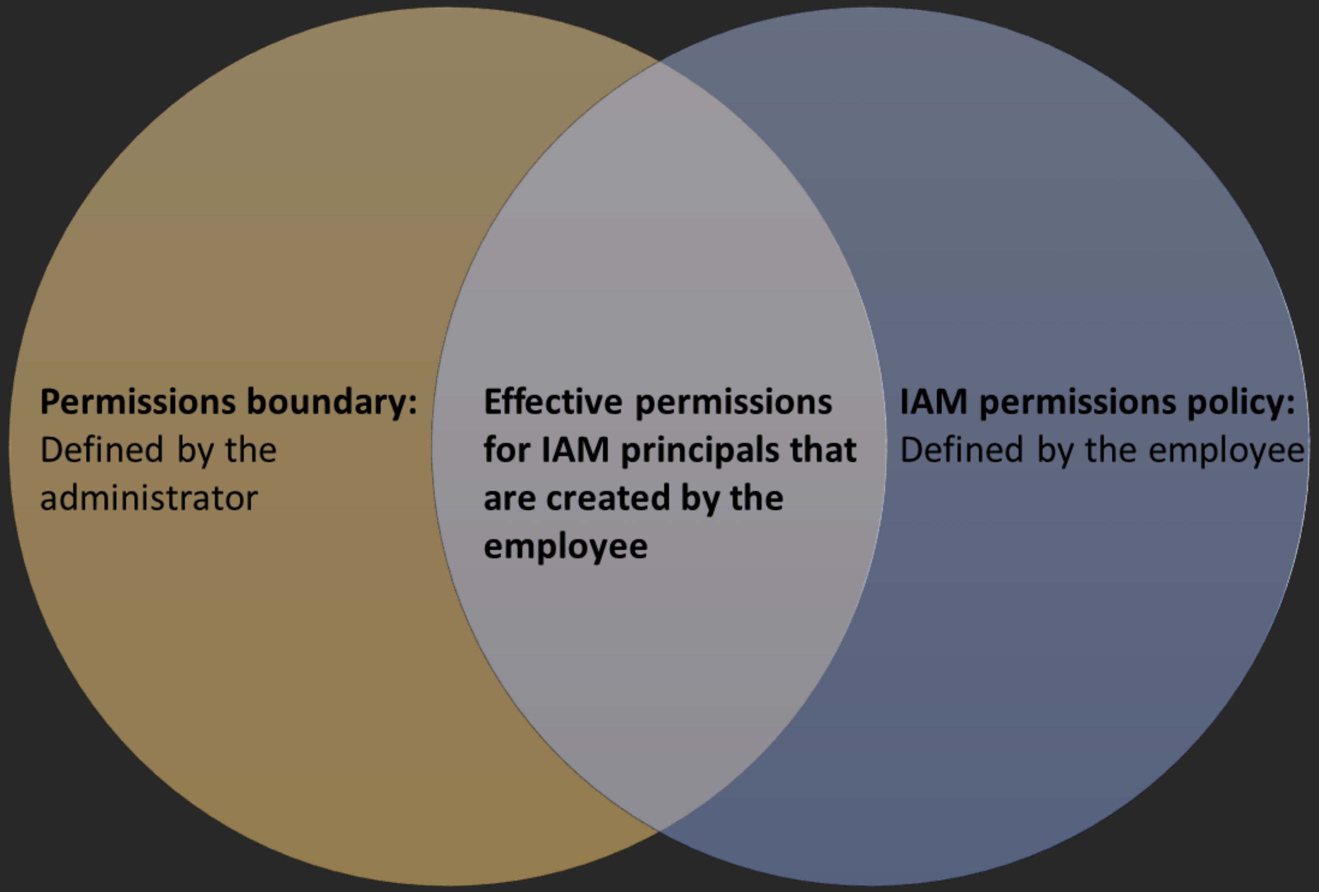
AWS Organizations Service control policies (SCPs)	*Guardrails to disable service access on the principals in the account*
AWS Identity and Access Management (IAM) As **Permission Policies** and **Permission Boundaries**	*Grant granular permissions on IAM principals (users and roles) and control the maximum permissions they can set*
AWS Security Token Service (AWS STS) Scoped-down policies	*Reduce general shared permissions further*
Specific AWS services Resource-based policies	*Cross-account access and to control access from the resource*
VPC Endpoints Endpoint Policies	*Controls access to the service with a VPC endpoint*

All use the same policy language

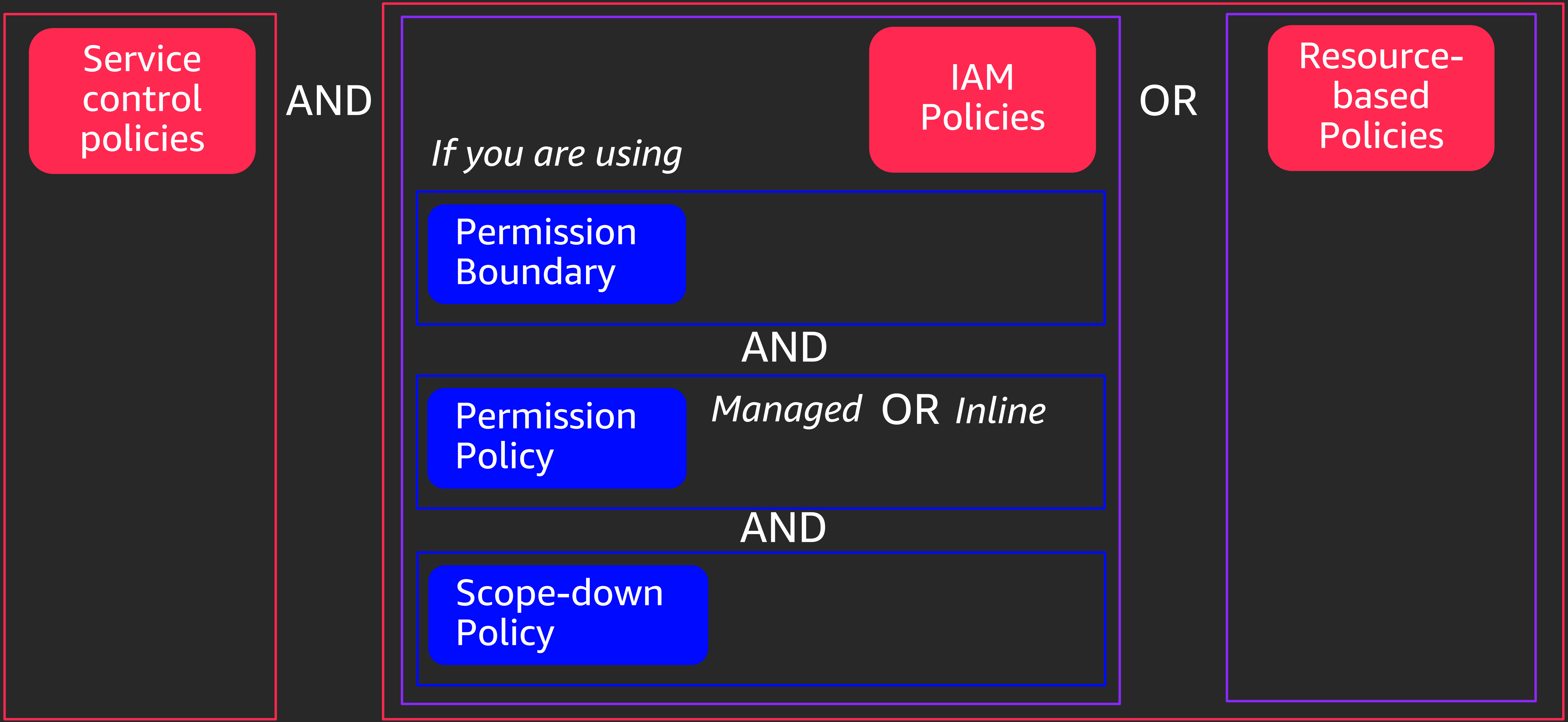
Permission boundaries – new this year!

Scale and delegate permission management to developers safely

Control the maximum permissions employees can grant


Permissions boundary: Defined by the administrator
Effective permissions for IAM principals that are created by the employee
IAM permissions policy: Defined by the employee

How policies work together *within an account*



Test your knowledge

Note: Within an account

Service control policies

Allow S3:PutObject

Denied

If you are using

IAM Policies

Permission Boundary

Not Present

Allow EC2 Read

Permission Policy

Managed OR *Inline*

Allow S3:PutObject

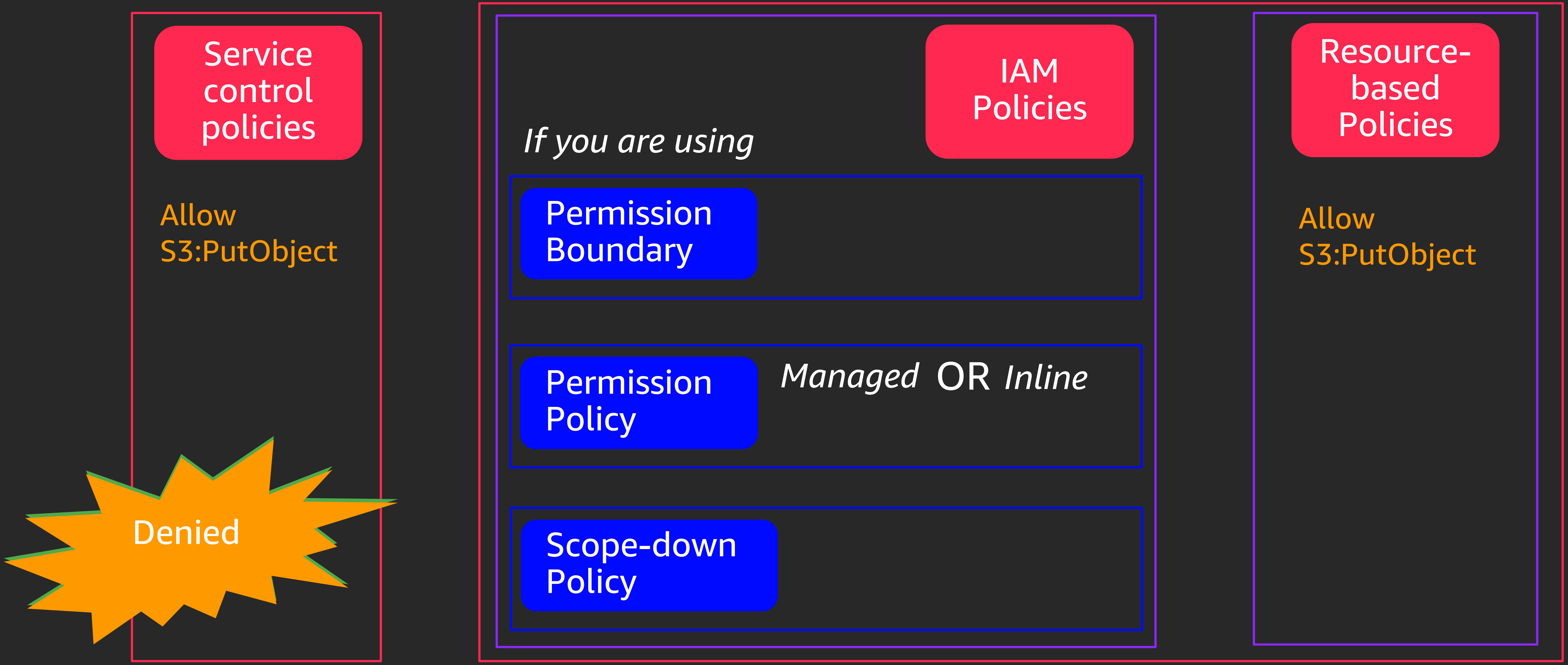
Scope-down Policy

Not Present

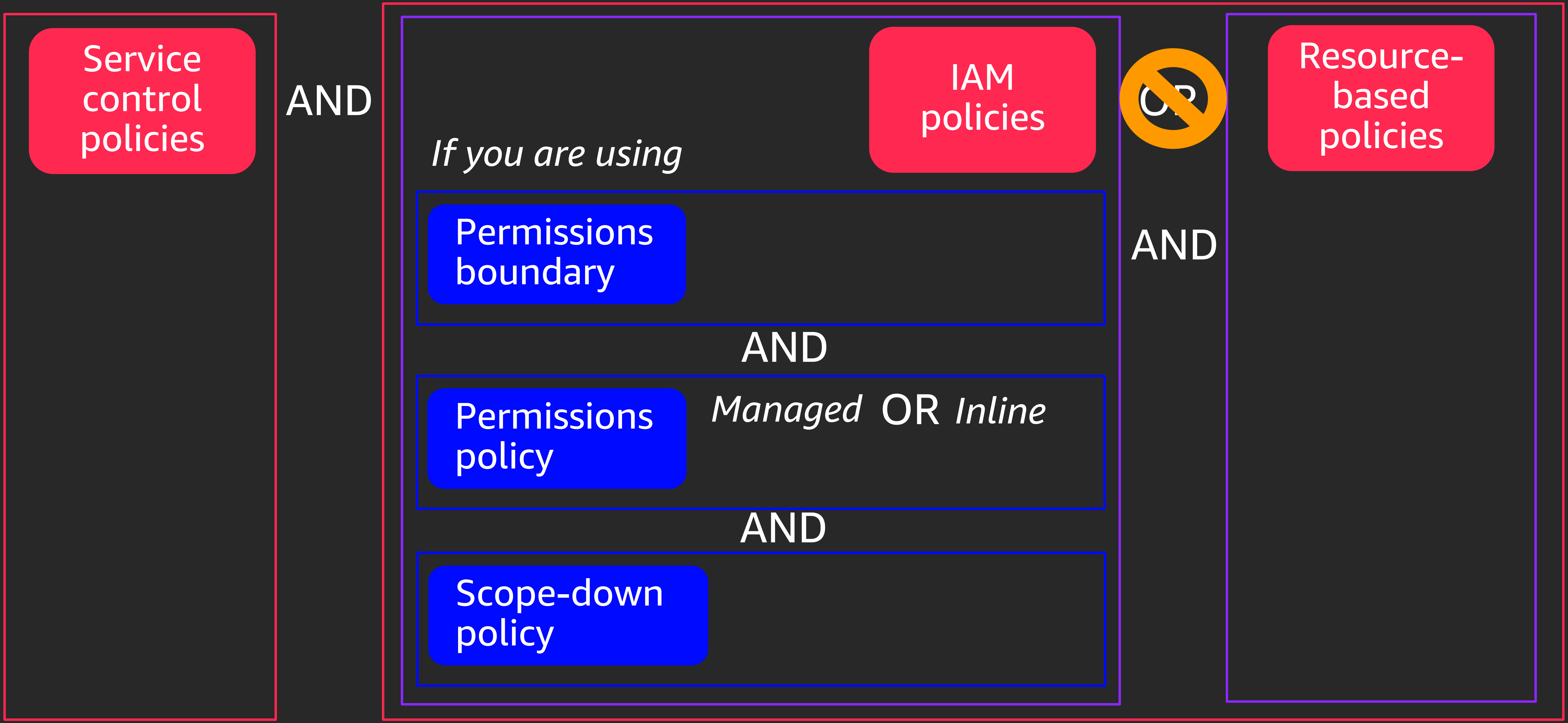
Resource-based Policies

Test your knowledge

Note: Within an account



How policies work together *across accounts*



Test your knowledge

Note: Across accounts

Service control policies

Allow S3:PutObject

If you are using **IAM policies**

Permissions boundary — *Not Present*

Permissions policy — *Managed* OR *Inline*

Allow S3:PutObject

Scope-down policy — *Not Present*

Resource-based policies

Allow S3:PutObject

Allowed

Deep dive on policy with specific use cases

Congratulations – you just got a new job!

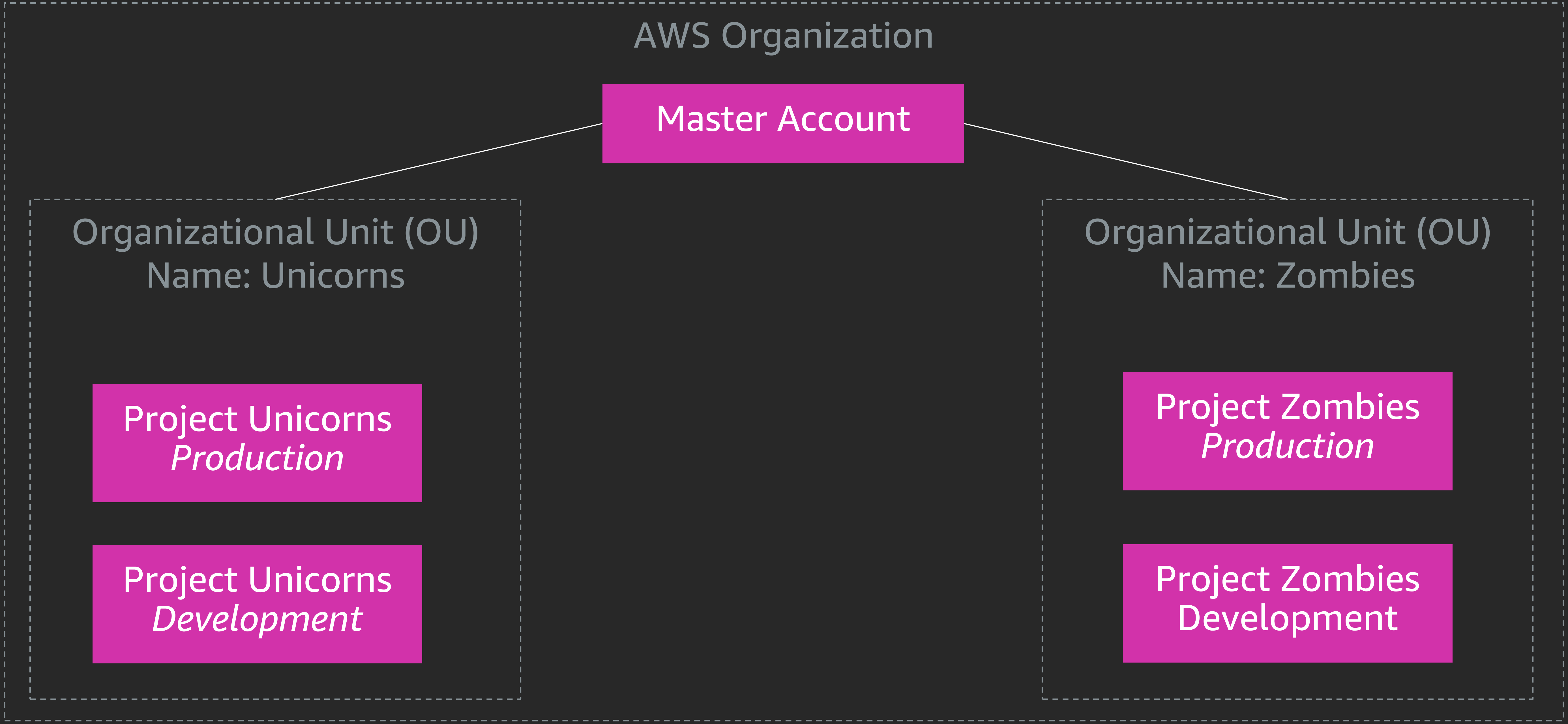
Your new position

Let's imagine you are now the lead of a central security team

Your first mission

Prevent developers from reverting setting in your AWS accounts and onboard a new two new teams!

AWS Organization and account structure


AWS Organization
Master Account
Organizational Unit (OU)
Name: Unicorns
Project Unicorns *Production*
Project Unicorns *Development*
Organizational Unit (OU)
Name: Zombies
Project Zombies *Production*
Project Zombies Development

Services your organization uses

Amazon EC2 Amazon EC2 to run workloads

AWS Lambda AWS Lambda for serverless applications

AWS Secrets Manager AWS Secrets Manager to store secrets for database access and third party API keys

Amazon S3 Amazon S3 to store content objects

#1 - Set permission guardrails across accounts

#1 - Situation

Your team has gone through and set up AWS CloudTrail in all accounts. Your company also requires users to authenticate with their existing identity provider.

#1 - Challenge

Ensure developers cannot turn off CloudTrail, create IAM users, or set up AWS Directory Service.

#2 - Control creation of resources to specific regions

#2 - Situation

You've learned that you can trust your development team to create resources in AWS, however your leadership is concerned about creating resources in unapproved regions.

#2 - Challenge

Ensure your developers can create resources, but only in approved regions.

#3 - Enable developers to create roles safely

#3 - Situation

Your developers know their stuff! They mentioned they can build on AWS more quickly if they can create their own roles without going through your central security team.

#3 - Challenge

Enable your developers to create IAM roles to pass to Amazon EC2 and AWS Lambda, but ensure they cannot exceed their own permissions.

#4 - Use tags to scale permissions management

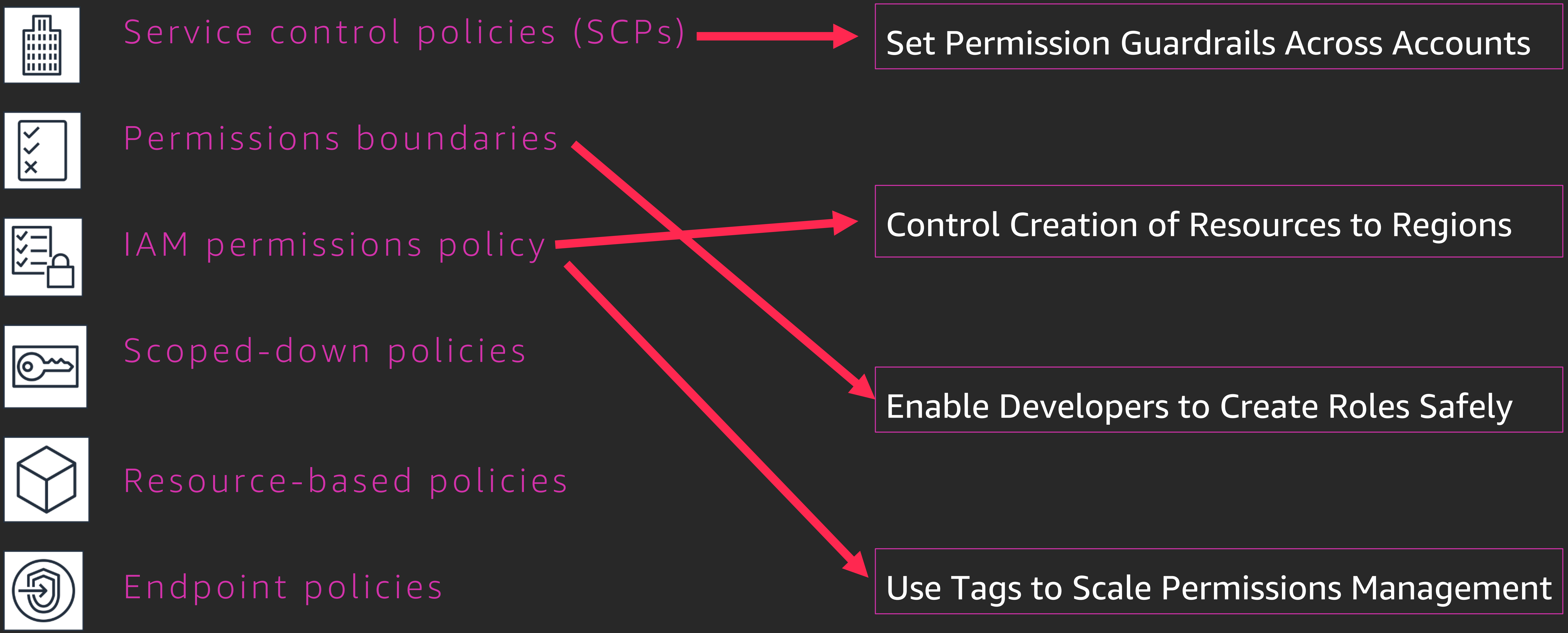
#4 - Situation

The Unicorns project has been split into two projects. Dorky Unicorns and Sneaky Unicorns. They still share an account and keep stepping on each other toes.

#4 - Challenge

Update permissions to enable developers working on Dorky Unicorns and Sneaky Unicorns to manage their own resources without managing the other project’s.

Match the tool to use for each challenge


Service control policies (SCPs)
Permissions boundaries
IAM permissions policy
Scoped-down policies
Resource-based policies
Endpoint policies
Set Permission Guardrails Across Accounts
Control Creation of Resources to Regions
Enable Developers to Create Roles Safely
Use Tags to Scale Permissions Management

Challenge #1

Ensure developers cannot turn off CloudTrail, create IAM users, or set up AWS Directory Service.

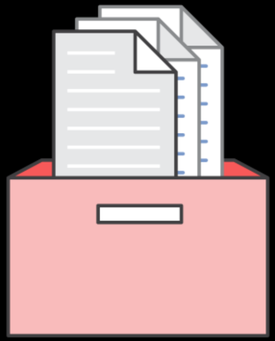
Pro Tip: Rely on deny statements when restricting access to accounts to reduce blast radius.

SCP for challenge #1

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyUnapprovedAction",
            "Effect": "Deny",
            "Action": [
                "ds:*",
                "iam:CreateUser",
                "cloudtrail:StopLogging"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Don’t Forget!

*We also have an Allow *.* policy attached to this OU*



AWS re:Invent

Let's see SCPs in action

- Show SCP to deny access to modify AWS CloudTrail, Create IAM users, and AWS Directory Services
- Use the CLI as an administrator in the Unicorn-dev account and try to:
  - Create an IAM user
  - List roles
  - Stop logging in AWS CloudTrail

AWS re:Invent

aws

Challenge #2

Ensure your developers can create resources, but only in approved regions.

Pro Tip: Use the RequestedRegion AWS condition

Policy for challenge #2

```
{
            "Effect": "Allow",
            "Action": [
                "secretsmanager:*",
                "lambda:*",
                "s3:PutObject",
                "s3:GetObject",
                "s3:DeleteObject"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestedRegion": [
                        "us-west-1",
                        "us-west-2"
                    ]
                }
            }
        }
```

Policy for challenge #2

```
{
            "Effect": "Allow",
            "Action": "ec2:RunInstances",
            "Resource": [
                "arn:aws:ec2:*:*:subnet/*",
                "arn:aws:ec2:*:*:key-pair/*",
                "arn:aws:ec2:*:*:instance/*",
                "arn:aws:ec2:*::snapshot/*",
                "arn:aws:ec2:*:*:launch-template/*",
                "arn:aws:ec2:*:*:volume/*",
                "arn:aws:ec2:*:*:security-group/*",
                "arn:aws:ec2:*:*:placement-group/*",
                "arn:aws:ec2:*:*:network-interface/*",
                "arn:aws:ec2:*::image/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:RequestedRegion": ["us-west-1","us-west-2"]
                }
            }
}
```

Policy for challenge #2

```
{
            "Effect": "Allow",
            "Action": [
                "ec2:Describe*",
                "ec2:Get*",
                "s3:ListBucket",
                "s3:ListAllMyBuckets",
                "iam:list*"
            ],
            "Resource": "*"
        }
```

Let's see region control in action

Using the developer role for challenge 2:

- ➢ Create a secret in the west region

- ➢ Create a secret in the London region

AWS re:Invent

aws

Challenge #3

Enable your developers to create IAM roles to pass to EC2 and Lambda, but ensure they cannot exceed their own permissions.

Pro Tip: Require and use role naming conventions to control the roles developers can manage.

Four parts required for permission boundaries

- Allow create managed policies
- Allow create role, but only with a specific permission boundary

  This is a condition with a pointer to an existing managed policy
- Allow attach managed policies, but only to roles with a specific boundary
- Allow passRole for these roles using a naming requirement

Policy for Challenge #3

Allow create managed policies

```
 {
        "Effect": "Allow",
        "Action": [
                "iam:CreatePolicy",
                "iam:CreatePolicyVersion",
                "iam:DeletePolicyVersion"
        ],
        "Resource": "arn:aws:iam::128609111811:policy/unicorns-*"
}
```

Policy for Challenge #3

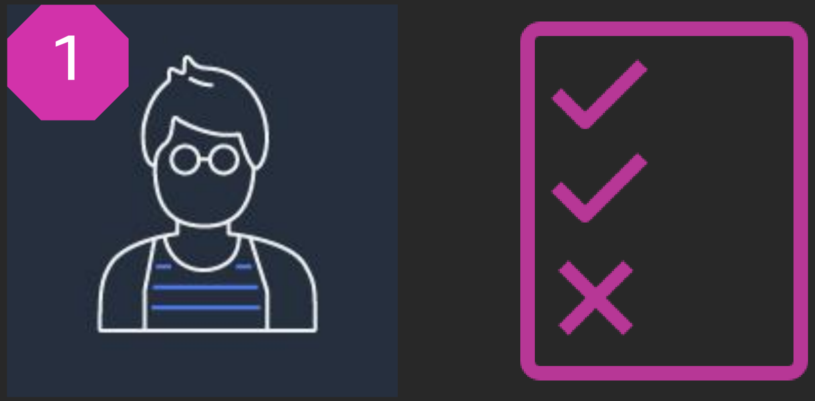
Allow create role, but only with a specific permission boundary

Allow attach managed policies, but only to roles with a specific boundary

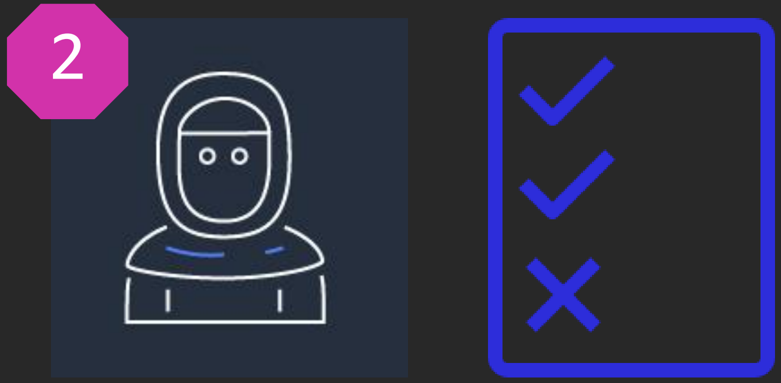
```
{
        "Effect": "Allow",
        "Action": [
                "iam:DetachRolePolicy",
                "iam:CreateRole",
                "iam:AttachRolePolicy"
        ],
        "Resource": "arn:aws:iam::128609111811:role/unicorns-*",
        "Condition": {
                "StringEquals": {
                        "iam:PermissionsBoundary":
        "arn:aws:iam::128609111811:policy/region-restriction"
                }
        }
}
```

AWS re:Invent

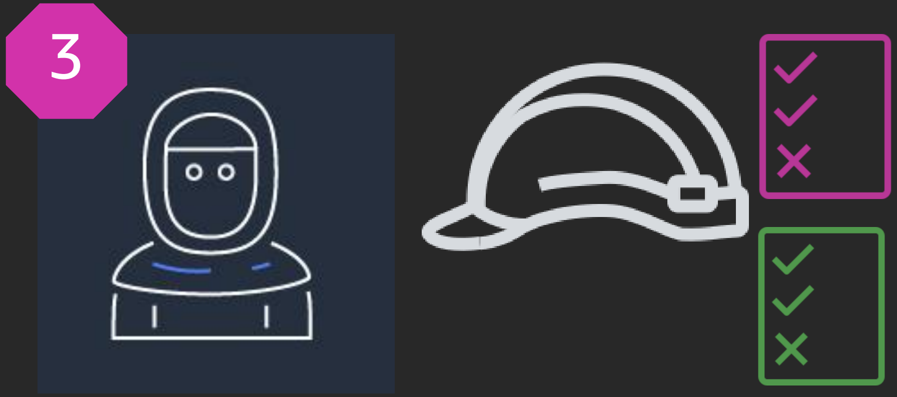
Permission boundary workflows



Admin creates
maximum permissions



Admin allows developers to create role with maximum permissions



Developer creates role with maximum permissions and specific permissions

Developers passes the role to application resources

Let’s see permission boundaries in action

- ➢ Using the developer role, create a role with a permission boundary
- ➢ Use a role with a permission boundary to put data in S3 for approved regions and for unapproved regions.

Challenge #4

Enable developers working on the Dorky project and the Sneaky project to manage their own resources without also managing the other project's.

Pro Tip: Carefully consider the tag keys you want to use for authorization

Three parts required for tag-based access control

- Allow users to create tags when creating resources, but require specific tags when users create resources
  - RequestTag condition to require specific tag value during create actions
- Control which existing resources and values developers can tag
  - Use a combination of RequestTag and ResourceTag control access
- Control resources users can manage based on tag values
  - ResourceTag to control access to resources based on a tag that exists on a resource

Policy for challenge #4

```
"Effect": "Allow",
           "Action": [
               "ec2:RunInstances"
           ],
           "Resource": [
               "arn:aws:ec2:*:*:subnet/*",
               "arn:aws:ec2:*:*:key-pair/*",
               "arn:aws:ec2:*::snapshot/*",
               "arn:aws:ec2:*:*:launch-template/*",
               "arn:aws:ec2:*:*:volume/*",
               "arn:aws:ec2:*:*:security-group/*",
               "arn:aws:ec2:*:*:placement-group/*",
               "arn:aws:ec2:*:*:network-interface/*",
               "arn:aws:ec2:*::image/*"
           ],
           "Condition": {
               "StringEquals": {
                   "aws:RequestedRegion": ["us-west-1", "us-west-2" ]
               }
           }
```

Policy for challenge #4

Allow for creation of tags when creating new resources, but…

```
"Effect": "Allow",

"Action": "ec2:CreateTags",

"Resource": "*",

"Condition": {

       "StringEquals": {

              "ec2:CreateAction": "RunInstances"

       }

}
```

Allows creation of tags

But only during RunInstances calls

Policy for challenge #4

…require specific tags when users create new resources

```
"Effect": "Allow",
"Action": [
  "ec2:RunInstances"
],
"Resource": [
  "arn:aws:ec2:*:*:instance/*"],
"Condition": {
  "ForAllValues:StringEquals": {
       "aws:TagKeys": ["project","name"]
  },
  "StringEquals": {
       "aws:RequestTag/project": ["dorky"],
       "aws:RequestedRegion": ["us-west-1","us-west-2"]
  }}
```

Allows project and/or name, but nothing else

Requires project tag and must be this value

Requires instance to be in approved region

Policy for challenge #4

Control which existing resources and values developers can tag

```
"Effect": "Allow",
"Action": "ec2:CreateTags",
"Resource": "*",
"Condition": {
        "StringEquals": {
                "ec2:ResourceTag/project": ["dorky"]
        },
        "ForAllValues:StringEquals": {
                "aws:TagKeys": ["project","name"]
        },
        "StringEqualsIfExists": {
                "aws:RequestTag/project": ["dorky"]
        }
}
```

Only tag resources with these tags

Tag with either of these keys

For *project*, you specify only these values

AWS re:Invent

aws

Policy for challenge #4

Control resources users can manage based on tag values

```
"Effect": "Allow",
"Action": [
       "ec2:StartInstances",
       "ec2:StopInstances"],
"Resource": "*",
"Condition": {
       "StringEquals": {
              "ec2:ResourceTag/project": "dorky"
       }
}
}
```

Only manage resources with these tags

Let’s see tag-based access control in action

- ➢ Launch instances for project dorky
- ➢ Try to launch instances for project sneaky
- ➢ Modify tags on existing instances (dorky and sneaky)
- ➢ Manage existing instances

Bonus challenge

New! You can tag IAM users and roles

Create a general policy that allows read access to secrets

tagged with a role tag.

Pro Tip: Any condition key can also be used as a variable as a condition value (the right-hand side)

```
["${aws:PrincipalTag/project}"]
```

Policy for challenge #5

…require specific tags when users create new resources

```
"Effect": "Allow",
"Action": [
  "ec2:RunInstances"
],
"Resource": [
  "arn:aws:ec2:*:*:instance/*"],
"Condition": {
  "ForAllValues:StringEquals": {
       "aws:TagKeys": ["project","name"]
  },
  "StringEquals": {
       "aws:RequestTag/project": ["${aws:PrincipalTag/project}"],
       "aws:RequestedRegion": ["us-west-1","us-west-2"]
  }}
```

Allows project and/or name, but nothing else

Requires project tag and must be my project tag

Requires instance to be in approved region

Policy for challenge #5

Control which existing resources and values developers can tag

```
"Effect": "Allow",
"Action": "ec2:CreateTags",
"Resource": "*",
"Condition": {
        "StringEquals": {
                "ec2:ResourceTag/project": ["${aws:PrincipalTag/project}"]
        },
        "ForAllValues:StringEquals": {
                "aws:TagKeys": ["project","name"]
        },
        "StringEqualsIfExists": {
                "aws:RequestTag/project": ["${aws:PrincipalTag/project}"]
        }
}
```

Only tag resources with my project ta

Tag with either of these keys

For *project*, you specify your project tag

Policy for challenge #5

Control resources users can manage based on tag values

```
"Effect": "Allow",
"Action": [
       "ec2:StartInstances",
       "ec2:StopInstances"],
"Resource": "*",
"Condition": {
       "StringEquals": {
              "ec2:ResourceTag/project": "${aws:PrincipalTag/project}"
       }
}
}
```

Only manage resources with my project tag

Items I didn't get to that I recommend

PrincipalOrgID

Use in a resource-based policy to only allow IAM principals from your organization to access resources.

Blog: An easier way to control access to AWS resources by using the AWS organization of IAM principals

Service Specific Permission Documentation

A central location of services, actions, resource-level permissions, and conditions supported across AWS.

Page: Actions, Resources, and Condition Keys for AWS Services

Thank you!

Brigid Johnson

Please complete the session survey in the mobile app.