BNY MELLON

Beyond Entitlements for Cloud Native

Scalable Responsibility Management with Spring Boot and Open Policy Agent

Chandra Guntur

August 2020

Bank of New York Mellon

Disclosure

BNY Mellon is the corporate brand of The Bank of New York Mellon Corporation and may be used as a generic term to reference the corporation as a whole and/or its various subsidiaries generally. Products and services may be provided under various brand names in various countries by duly authorized and regulated subsidiaries, affiliates, and joint ventures of The Bank of New York Mellon Corporation. Not all products and services are offered in all countries.

BNY Mellon will not be responsible for updating any information contained within this material and opinions and information contained herein are subject to change without notice.

BNY Mellon assumes no direct or consequential liability for any errors in or reliance upon this material. This material may not be reproduced or disseminated in any form without the express prior written permission of BNY Mellon.

About :: Chandra Guntur

- Technologist in the financial services industry since 2003 and is programming with Java since 1998.
- Chandra Guntur is a **Director** and **Sr. Principal Architect** in Java Platform Engineering, BNY Mellon.
- BNY Mellon representative in the **Java Community Process** (JCP) **Executive Committee**.
- Java Champion
- **JUG Leader**, at **NYJavaSIG** (New York Java Special Interest Group) and **NJ Java SIG**.
- Speaker at: **Oracle CodeOne**, **Oracle CodeNY**, **QCon NY**, **Devnexus, DawsCon** and **GIDS India**.

Agenda

- Responsibility Management
  - Scenarios
  - Common (existing) solutions for both Data and Logic
  - A Right Solution - described and visualized
- Architecture
  - Technology choices
  - Federated model
  - Distributed model
  - Comparison
- A Sample Policy in action
- IDE Productivity tool

BNY MELLON

Responsibility Management for the Enterprise

Background and existing solutions

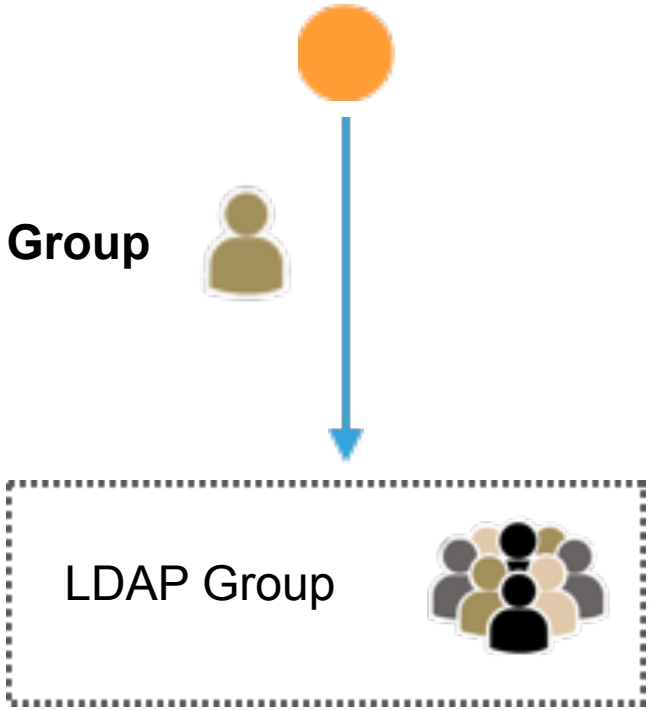
BNY MELLON

Responsibility Management System

Scenarios

Why Responsibility Management – Scenario 1

- Service A needs to know if a user is a member of an enterprise **LDAP Group**
  - Access may be granted based on membership.
  - Access may be denied based on membership.
  - Access may be granted based on lack of membership.
  - Access may be denied based on lack of membership.

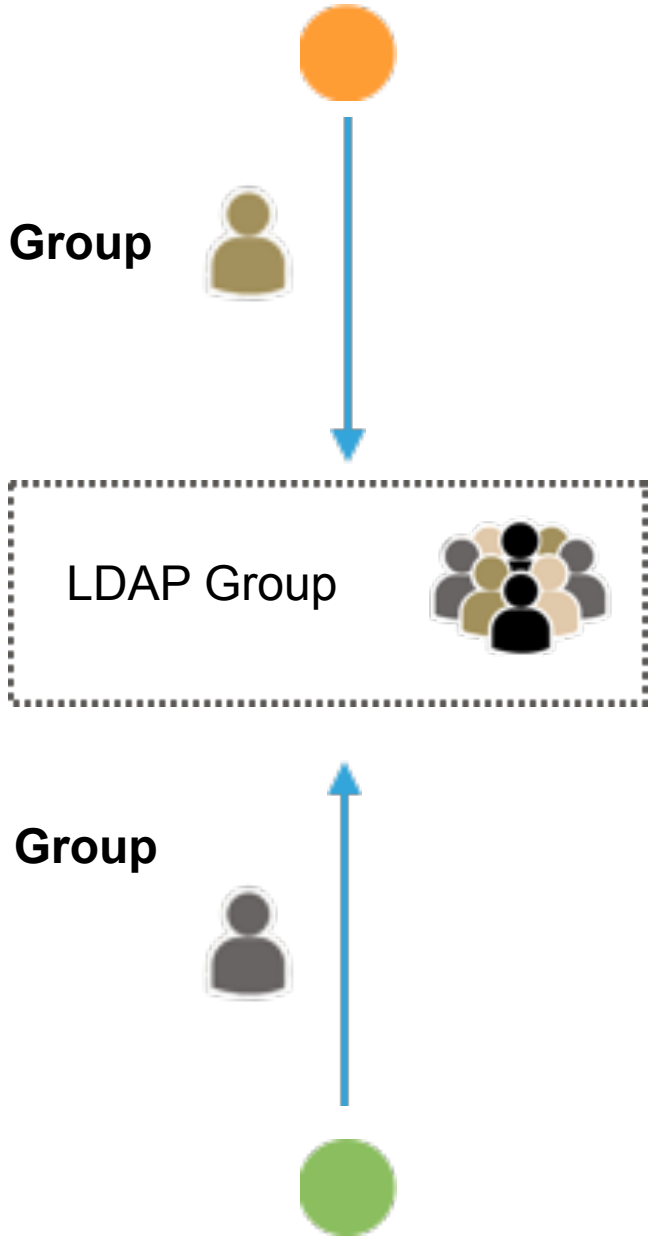


Why Responsibility Management – Scenario 1

- Service A needs to know if a user is a member of an enterprise **LDAP Group**
  - Access may be granted based on membership.
  - Access may be denied based on membership.
  - Access may be granted based on lack of membership.
  - Access may be denied based on lack of membership.

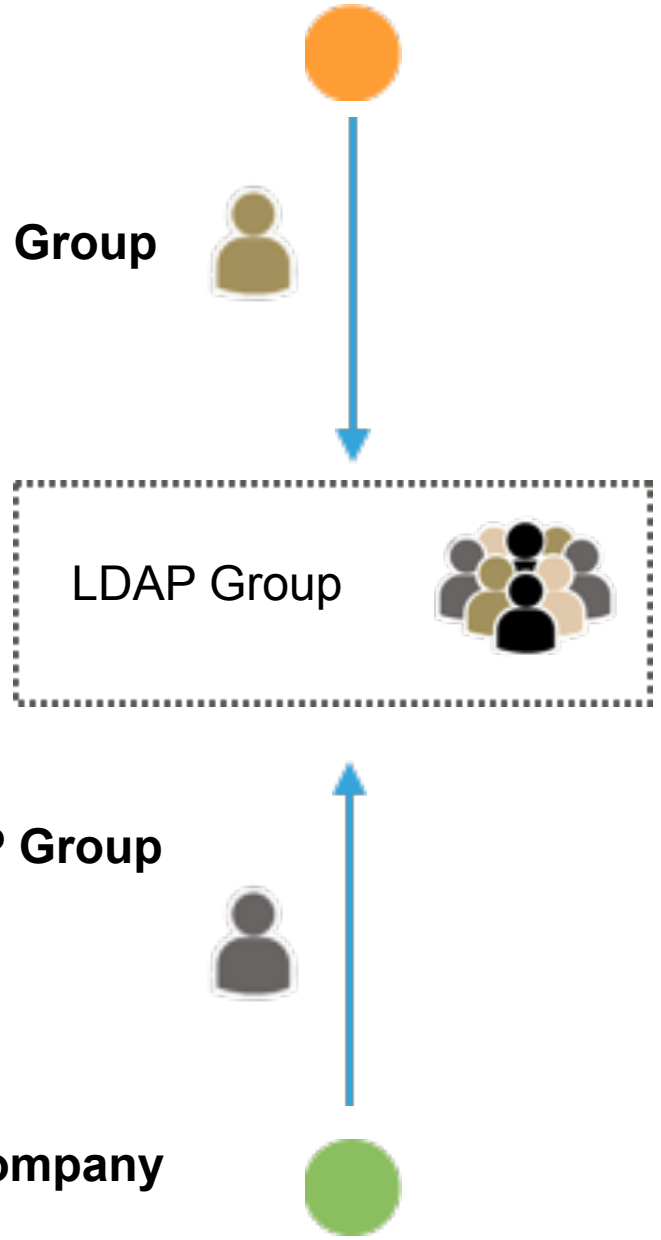
Then …

- Service B needs to know if a user is a member of an enterprise **LDAP Group**



Why Responsibility Management – Scenario 1

- Service A needs to know if a user is a member of an enterprise **LDAP Group**
  - Access may be granted based on membership.
  - Access may be denied based on membership.
  - Access may be granted based on lack of membership.
  - Access may be denied based on lack of membership.



Then …

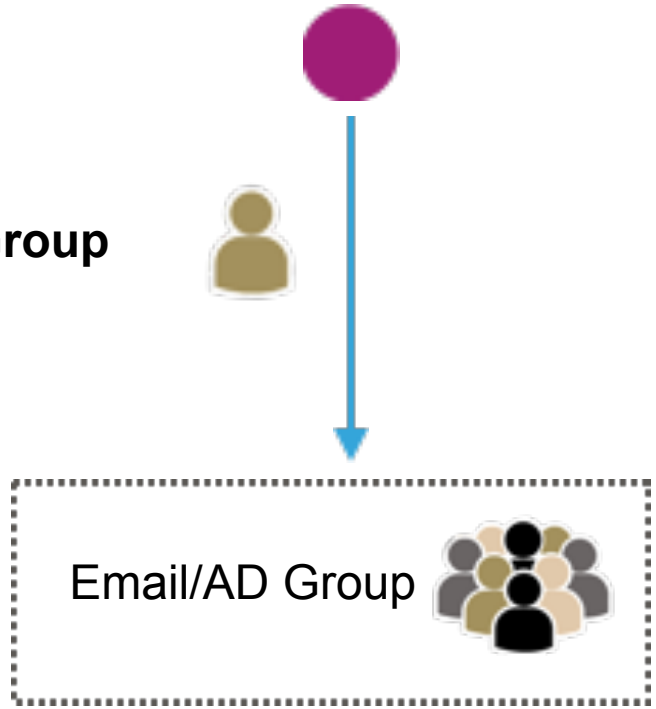
- Service B needs to know if a user is a member of an enterprise **LDAP Group**

Questions

- How about Service/Application C, D or E ?
- Who manages people who **move/leave/join the department/org/company** (Movers/Leavers/Joiners)

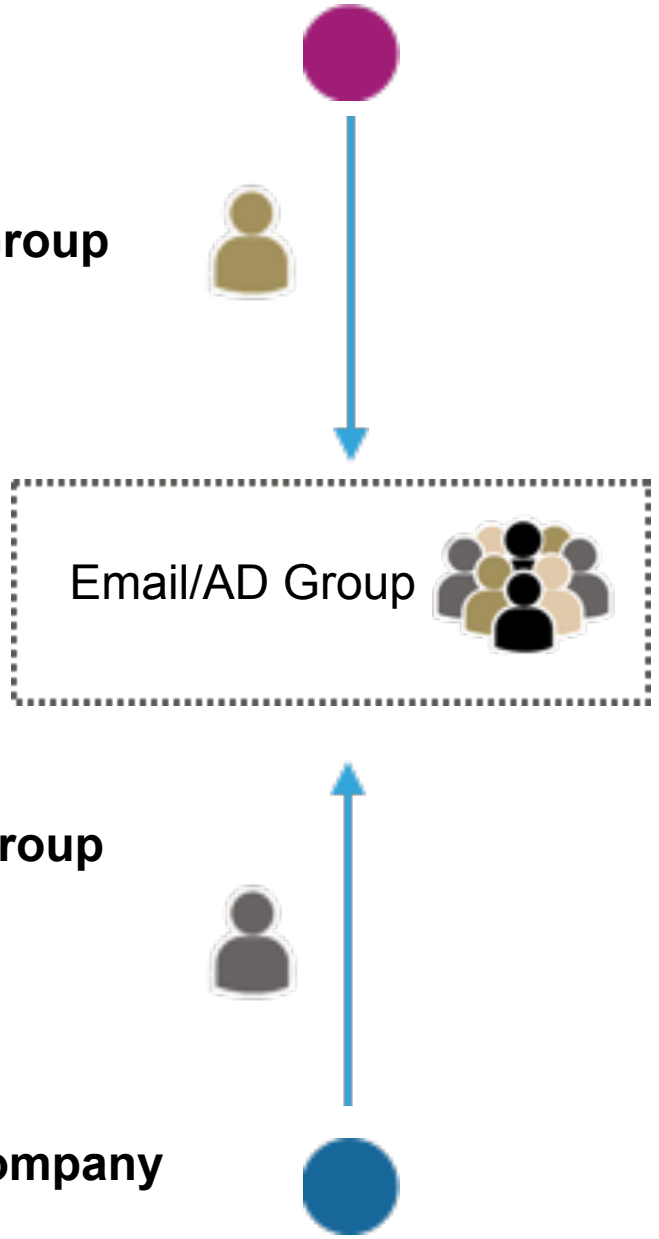
Why Responsibility Management – Scenario 2

- Service M needs to know if a user is a member of an enterprise **AD Group**
  - Access may be granted based on membership.
  - Access may be denied based on membership.
  - Access may be granted based on lack of membership.
  - Access may be denied based on lack of membership.



Why Responsibility Management – Scenario 2

- Service M needs to know if a user is a member of an enterprise **AD Group**
  - Access may be granted based on membership.
  - Access may be denied based on membership.
  - Access may be granted based on lack of membership.
  - Access may be denied based on lack of membership.



Then …

- Service N needs to know if a user is a member of an enterprise **AD Group**

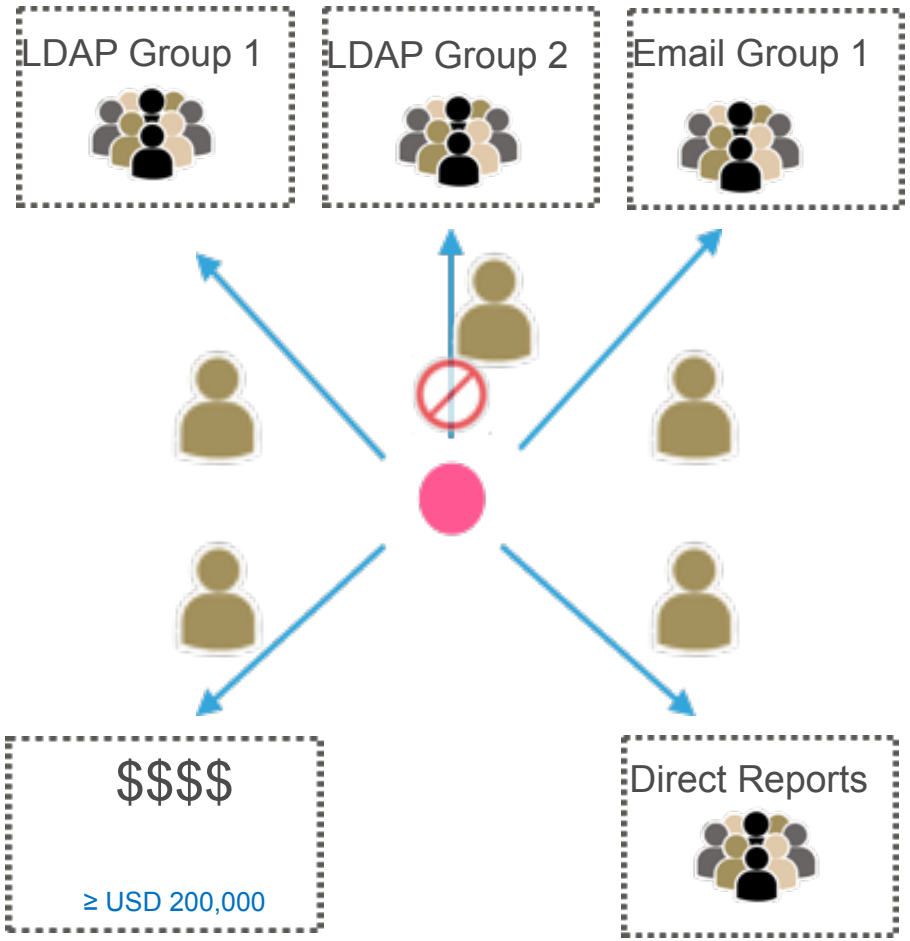
Questions

- How about Service/Application O, P or Q with different groups?
- Who manages people who **move/leave/join the department/org/company** (Movers/Leavers/Joiners)

Why Responsibility Management – Scenario 3

More complex evaluations occur as well.

Service X needs to check if all of the below are true for a user:

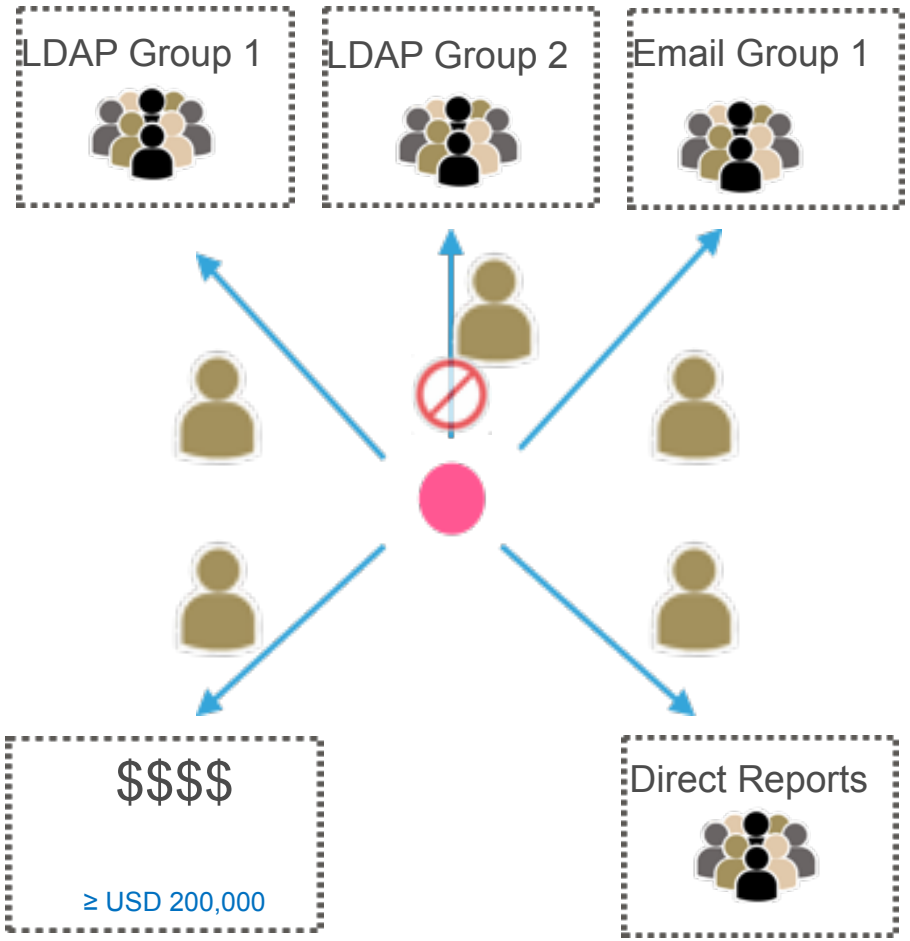


Why Responsibility Management – Scenario 3

More complex evaluations occur as well.

Service X needs to check if all of the below are true for a user:

- is member of LDAP Group 1

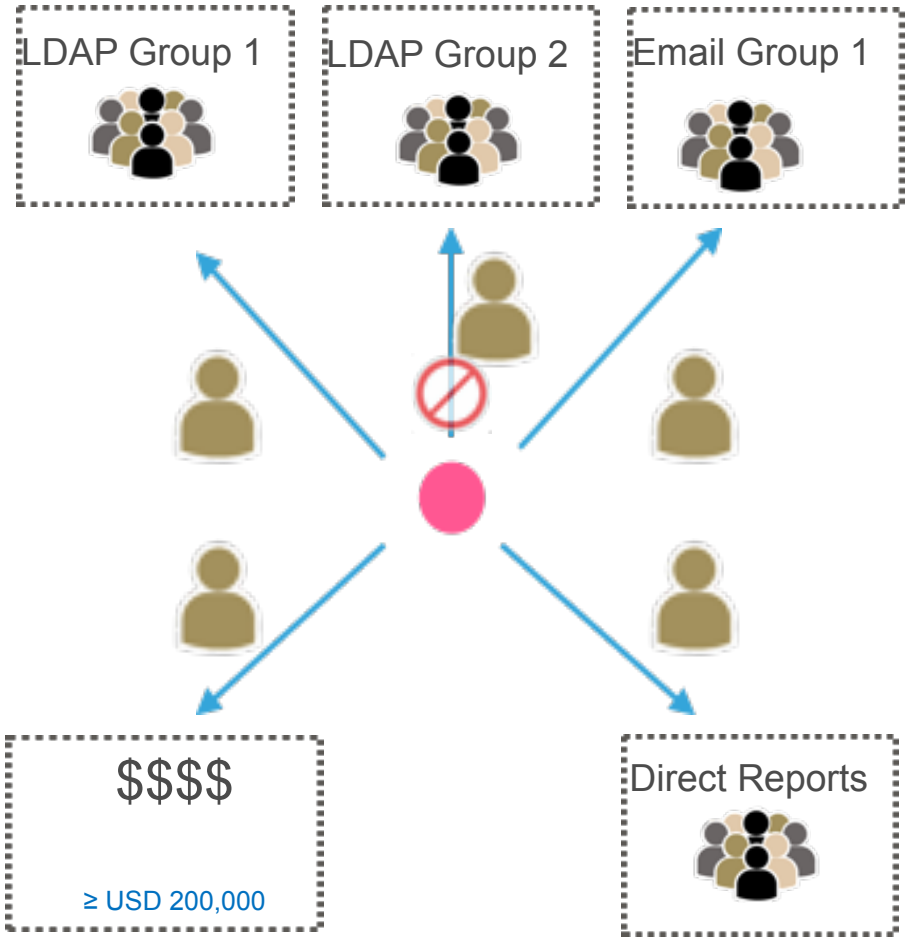


Why Responsibility Management – Scenario 3

More complex evaluations occur as well.

Service X needs to check if all of the below are true for a user:

- is member of LDAP Group 1
- is **not** member of LDAP Group 2

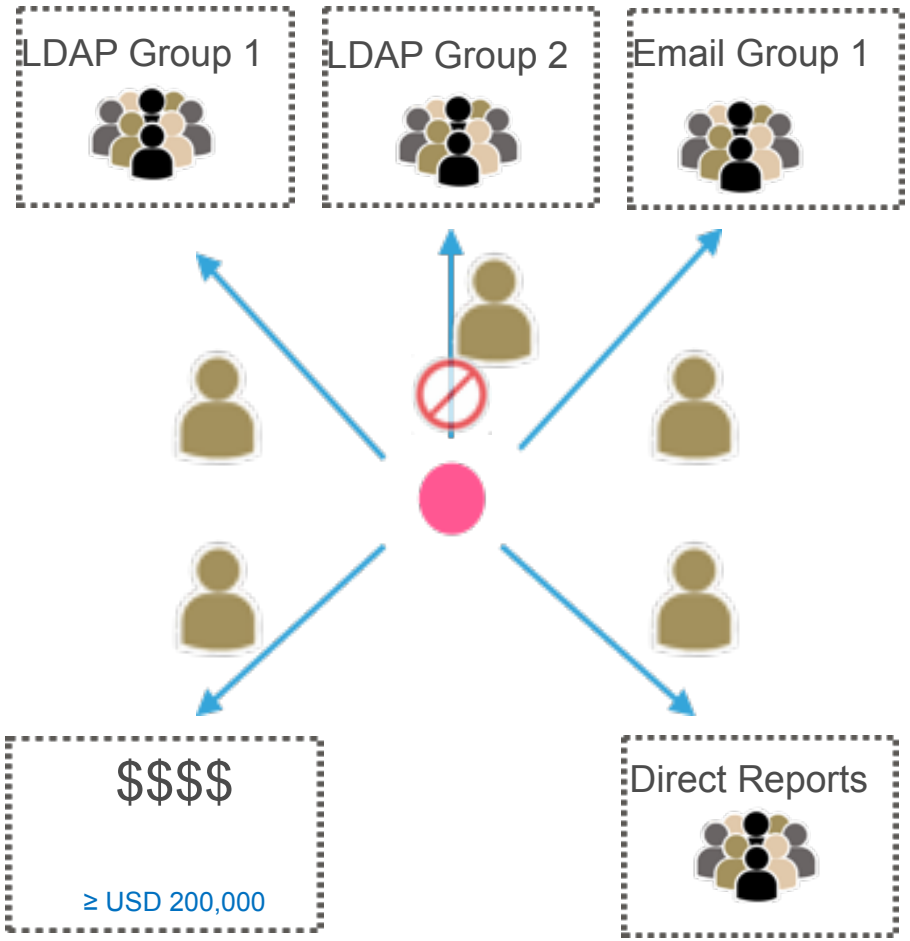


Why Responsibility Management – Scenario 3

More complex evaluations occur as well.

Service X needs to check if all of the below are true for a user:

- is member of LDAP Group 1
- is **not** member of LDAP Group 2
- is member of Email Group 1

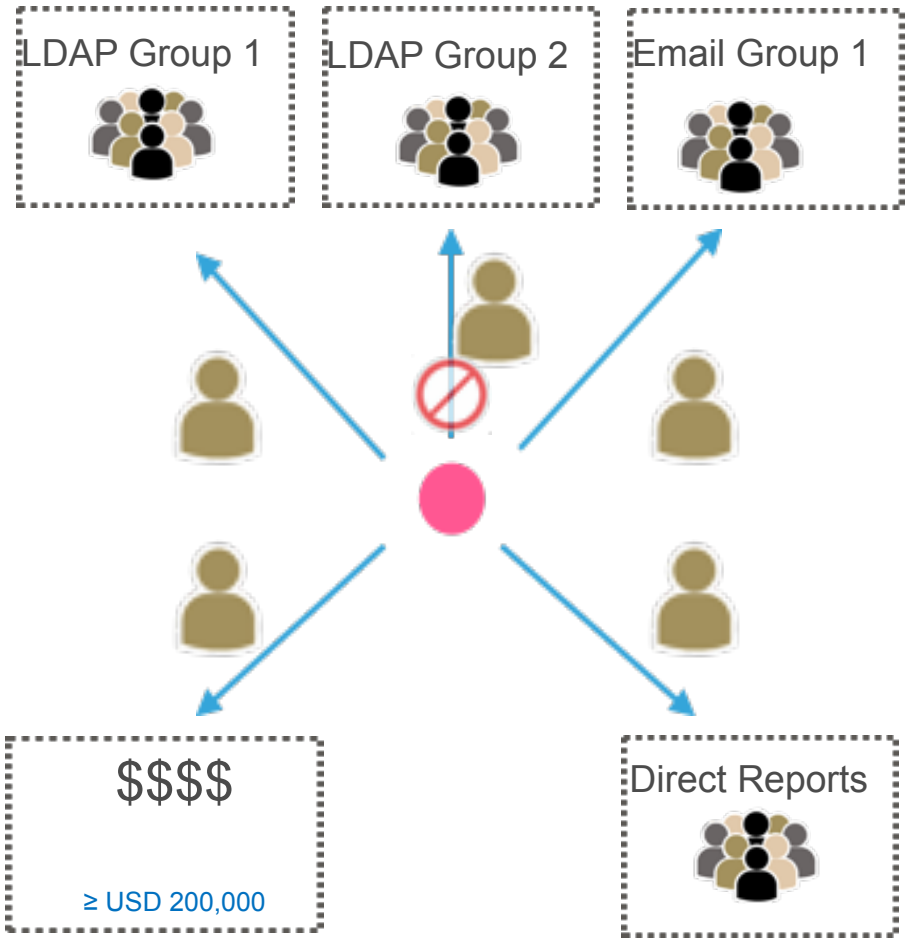


Why Responsibility Management – Scenario 3

More complex evaluations occur as well.

Service X needs to check if all of the below are true for a user:

- is member of LDAP Group 1
- is **not** member of LDAP Group 2
- is member of Email Group 1
- is allowed procurement amount USD 200,000

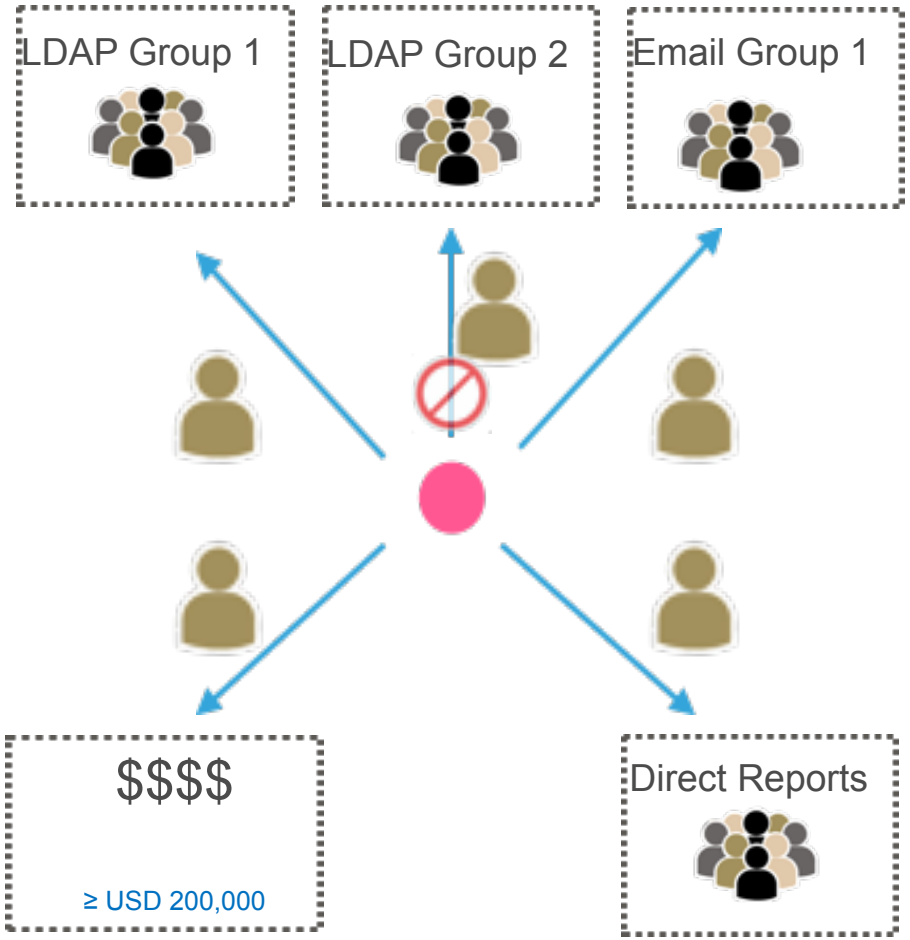


Why Responsibility Management – Scenario 3

More complex evaluations occur as well.

Service X needs to check if all of the below are true for a user:

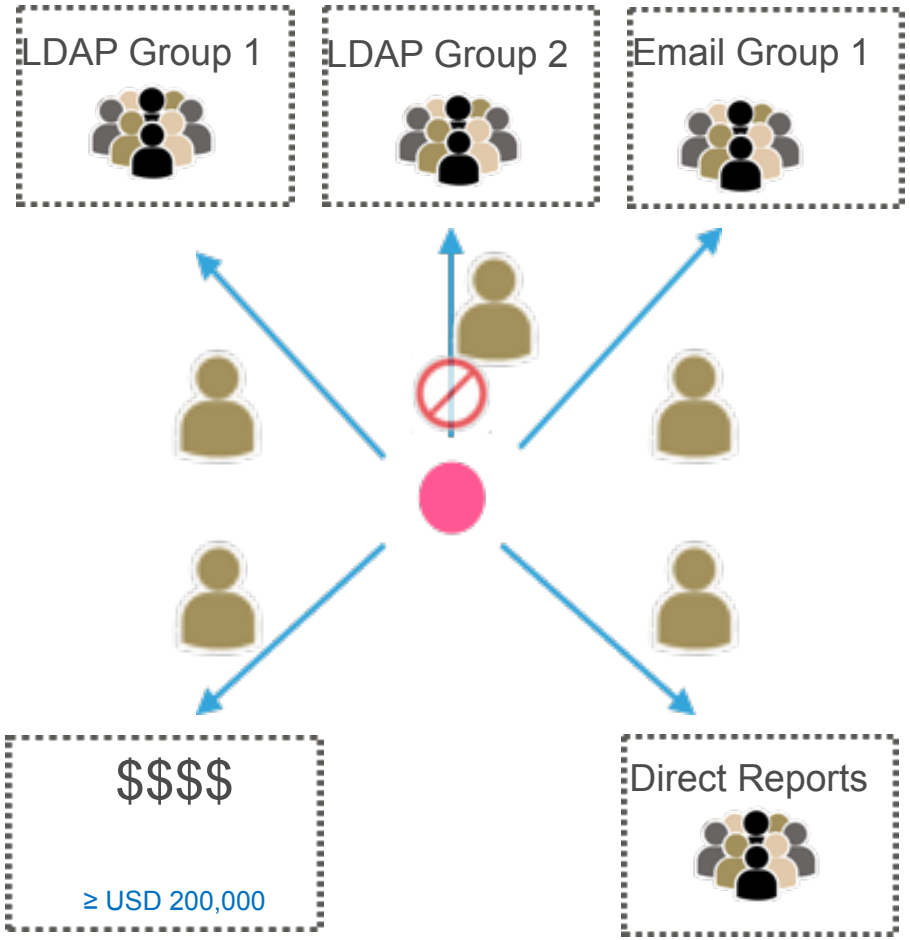
- is member of LDAP Group 1
- is **not** member of LDAP Group 2
- is member of Email Group 1
- is allowed procurement amount USD 200,000
- has at least two direct reports



Why Responsibility Management – Scenario 3 - Continued

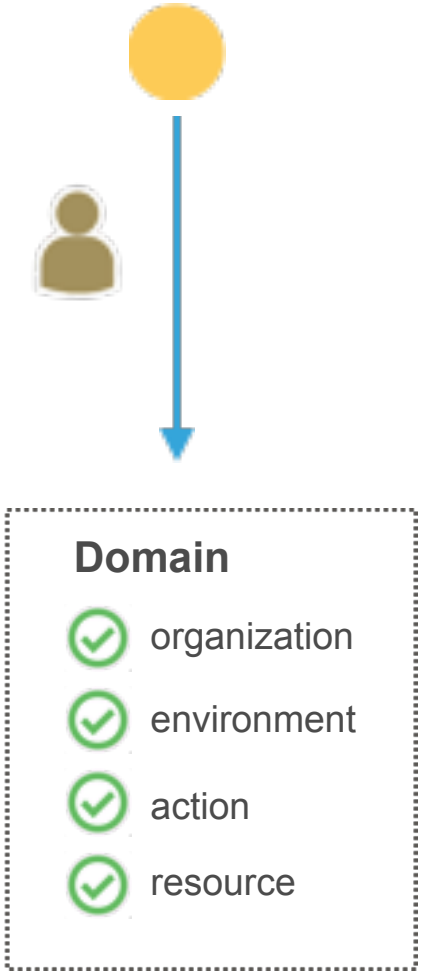
Questions

- What if each request is for different sets of groups and/or amounts?
- What if other services have similar functional constraints with different values?
- Where are such “policies” maintained?
  Are they auditable?
  Follow **SDLC** and **Config Management**?
- Who manages Mover/Leaver/Joiner logic?

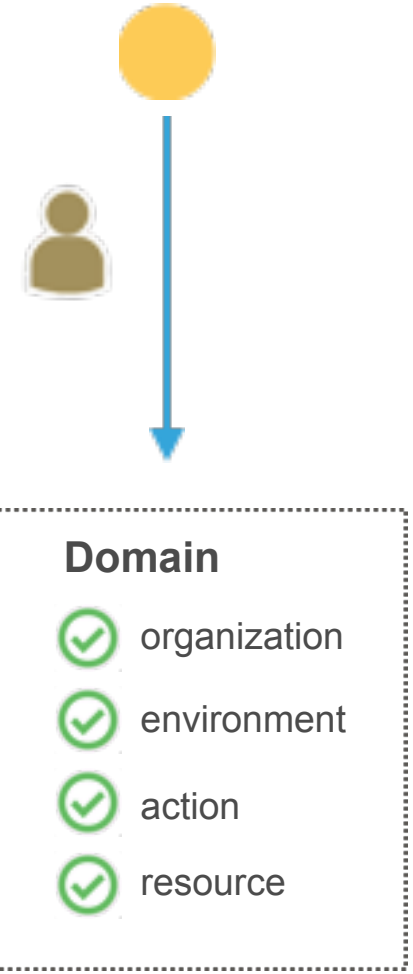


Why Responsibility Management – Scenario 4

Service Y needs to check responsibility privileges for a user/subject:



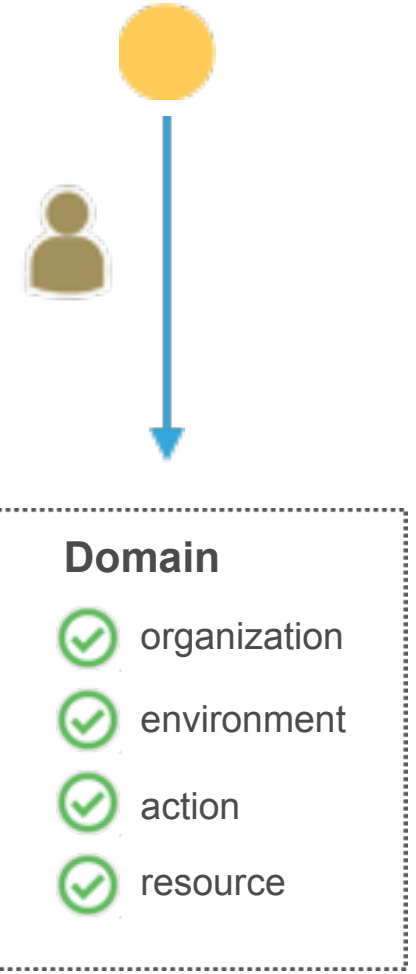
Why Responsibility Management – Scenario 4



Service Y needs to check responsibility privileges for a user/subject:

- in a given **domain** (Infra or Shared - service or tool)

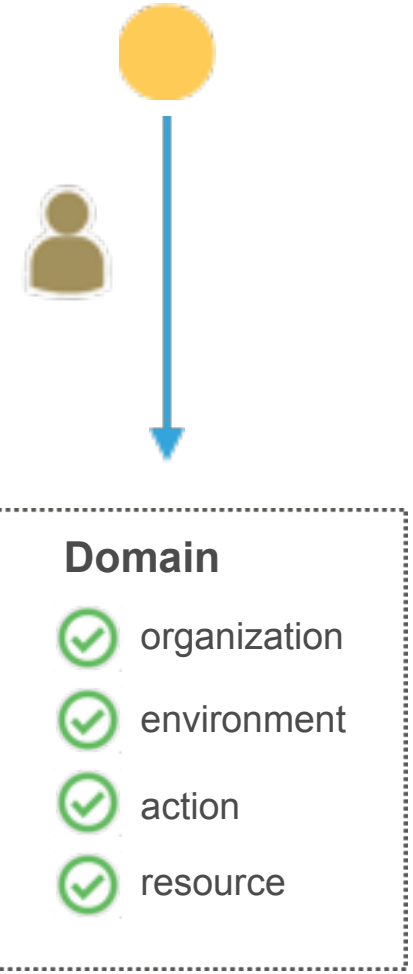
Why Responsibility Management – Scenario 4



Service Y needs to check responsibility privileges for a user/subject:

- in a given **domain** (Infra or Shared - service or tool)
- for a given **cost code identifier** or org. business unit ($)

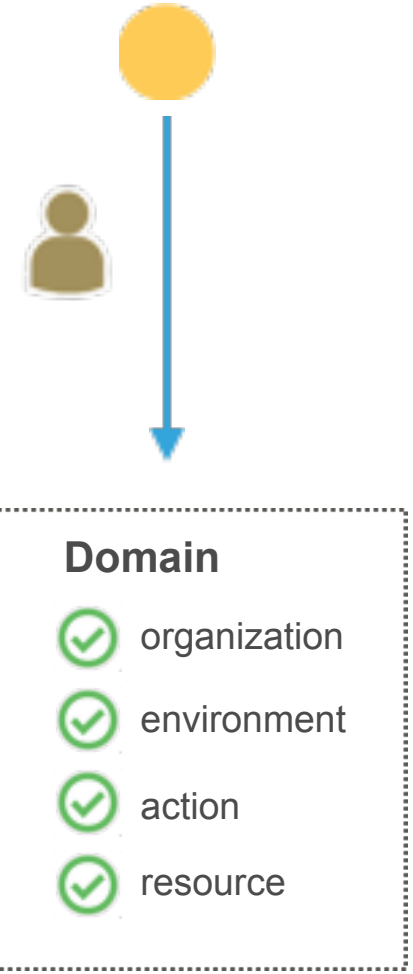
Why Responsibility Management – Scenario 4



Service Y needs to check responsibility privileges for a user/subject:

- in a given **domain** (Infra or Shared - service or tool)
- for a given **cost code identifier** or org. business unit ($)
- for a given **environment** (e.g. ‘PROD’, ‘QA’, ‘DEV’ …)

Why Responsibility Management – Scenario 4



Service Y needs to check responsibility privileges for a user/subject:

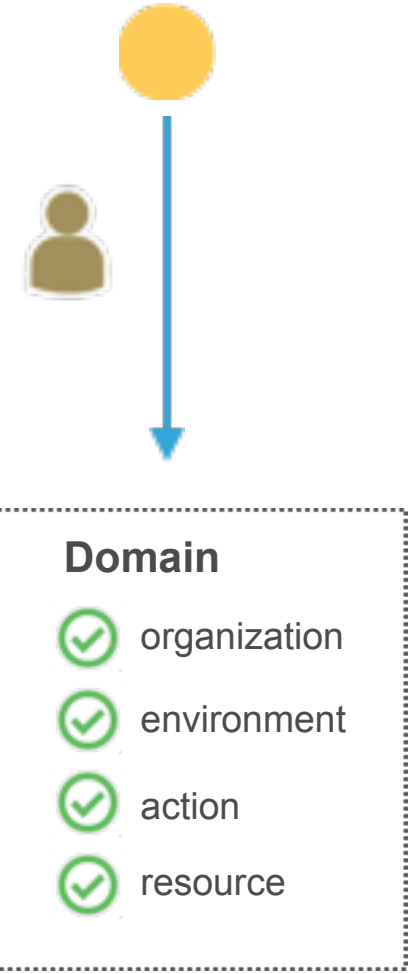
- in a given **domain** (Infra or Shared - service or tool)
- for a given **cost code identifier** or org. business unit ($)
- for a given **environment** (e.g. ‘PROD’, ‘QA’, ‘DEV’ …)
- for a given **action** (e.g. EDIT, DELETE, CREATE …)

Why Responsibility Management – Scenario 4

Service Y needs to check responsibility privileges for a user/subject:

- in a given **domain** (Infra or Shared - service or tool)
- for a given **cost code identifier** or org. business unit ($)
- for a given **environment** (e.g. ‘PROD’, ‘QA’, ‘DEV’ …)
- for a given **action** (e.g. EDIT, DELETE, CREATE …)
- for a given **resource** (e.g. org.databases.prod.instance1.schema1)

Why Responsibility Management – Scenario 4 - Continued



Questions

- What if each request is for different sets of values for the given domain?
- What if other services have similar functional constraints with different values?
- Who manages **Role-Responsibility** per domain and **User-Role** Mappings?
- Who manages Mover/Leaver/Joiner logic ?

BNY MELLON

Responsibility Management System

Common Solutions

Responsibility Management – Common Solutions – For Data

DATA - Reference or Static Data for evaluation

- **LDAP/Active directory** queried by the application/service via **direct connections**.
- **User approver/manager** is queried via **proprietary corporate directory services**.
- **Role-Responsibility mappings** usually stored in **local persistence**, **proprietary systems**.
- **User-Role mappings** usually stored in any of: **local persistence**, **proprietary systems**.

Responsibility Management – Common Solutions – For Functions

LOGIC - Calculations / Functions (Policies)

- Complex functions/calculations were **coded into the application/service**.
- Newer applications/services may separate such as an **independent [micro]service**.
- Some applications/services utilize **embedded rule engines** such as Drools.
- Some applications/services utilize **proprietary entitlement systems** for evaluations.

BNY MELLON

Responsibility Management System

A solution to manage dynamic privileges and entitlements

BNY MELLON

Responsibility Management System

A Right Solution

Responsibility Management System (RMS) – The Right Solution

A Responsibility Management System that:

- federates the calls to LDAP, Active Directory, and other services as **integrated services**
- provides appropriate **mapping of roles and responsibilities**, per domain
- provides for **user to role mapping, per organization per domain**
- provides **proper SDLC and audit mechanism** for policies per domain, to author and deploy

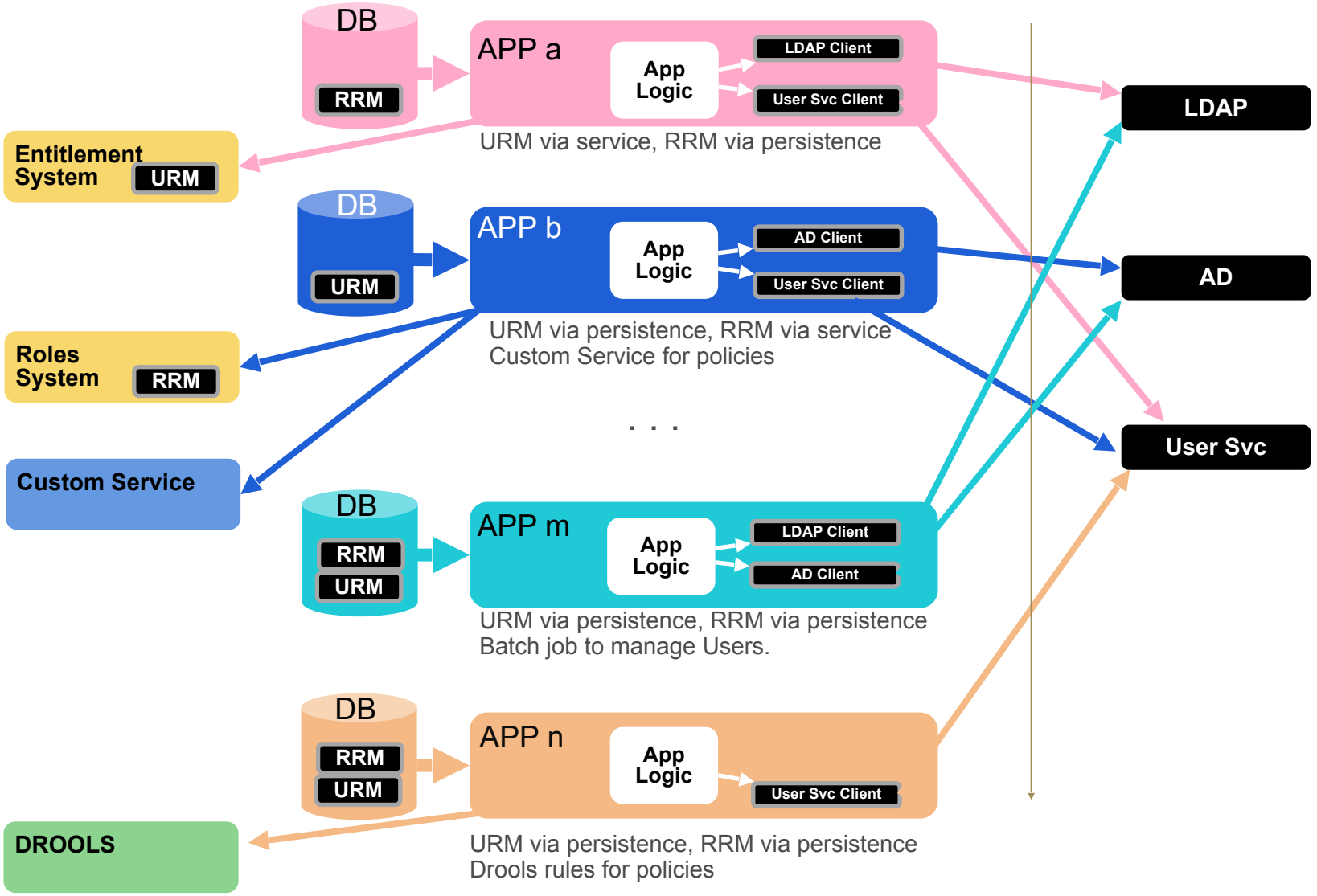
. . .

Responsibility Management System (RMS) – The Right Solution - Continued

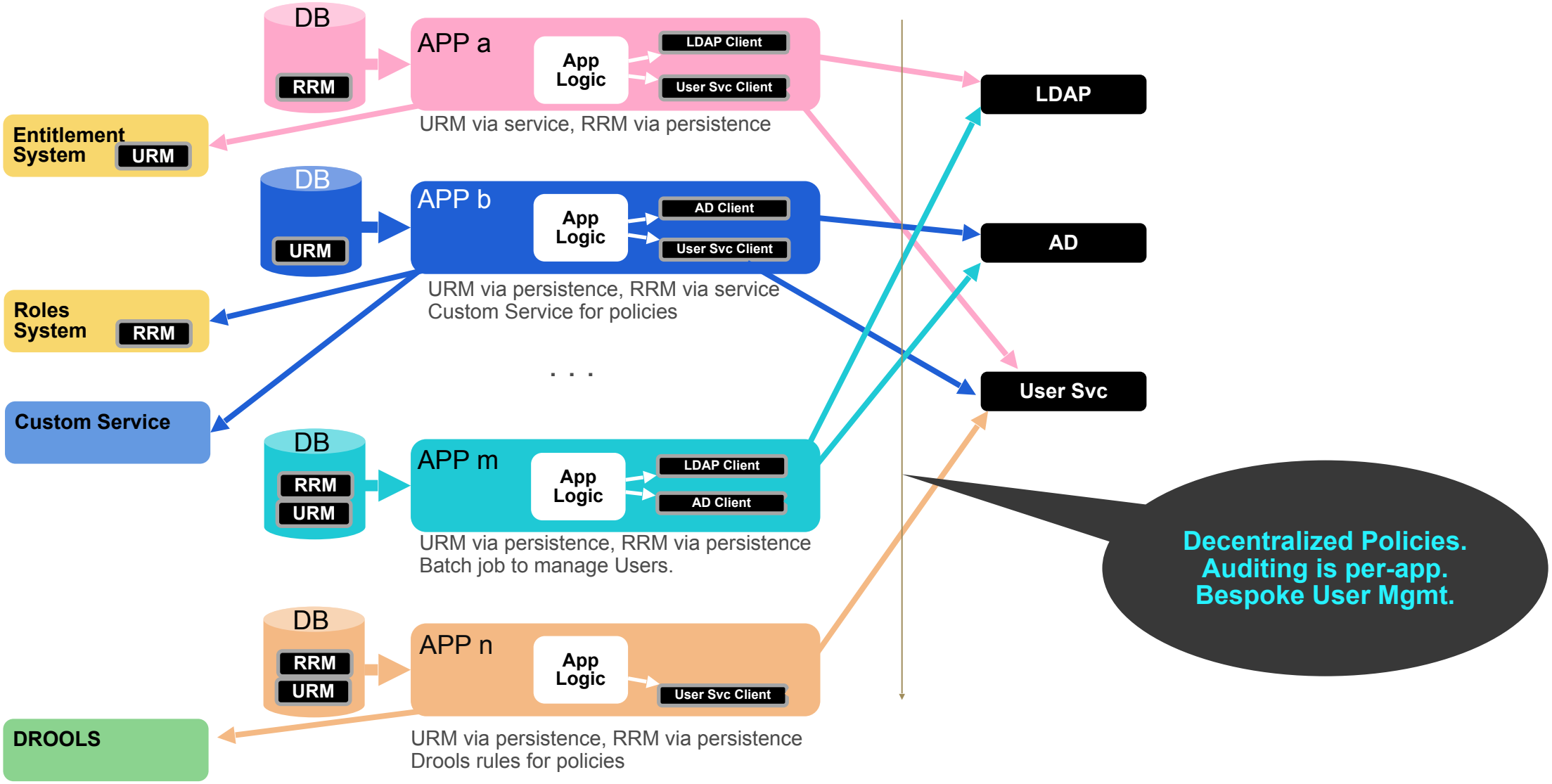
A Responsibility Management System that:

- provides a **policy engine** to evaluate complex calculations/functions using:
  - data provided as inputs by service-consumer
  - data queried from integrated services
  - policies provided by the domains
- caters to **applying a mover/leaver/joiner logic** to all controlled datasets
- provides **horizontal scaling** and thus, **high availability** for varying request volumes

Before RMS



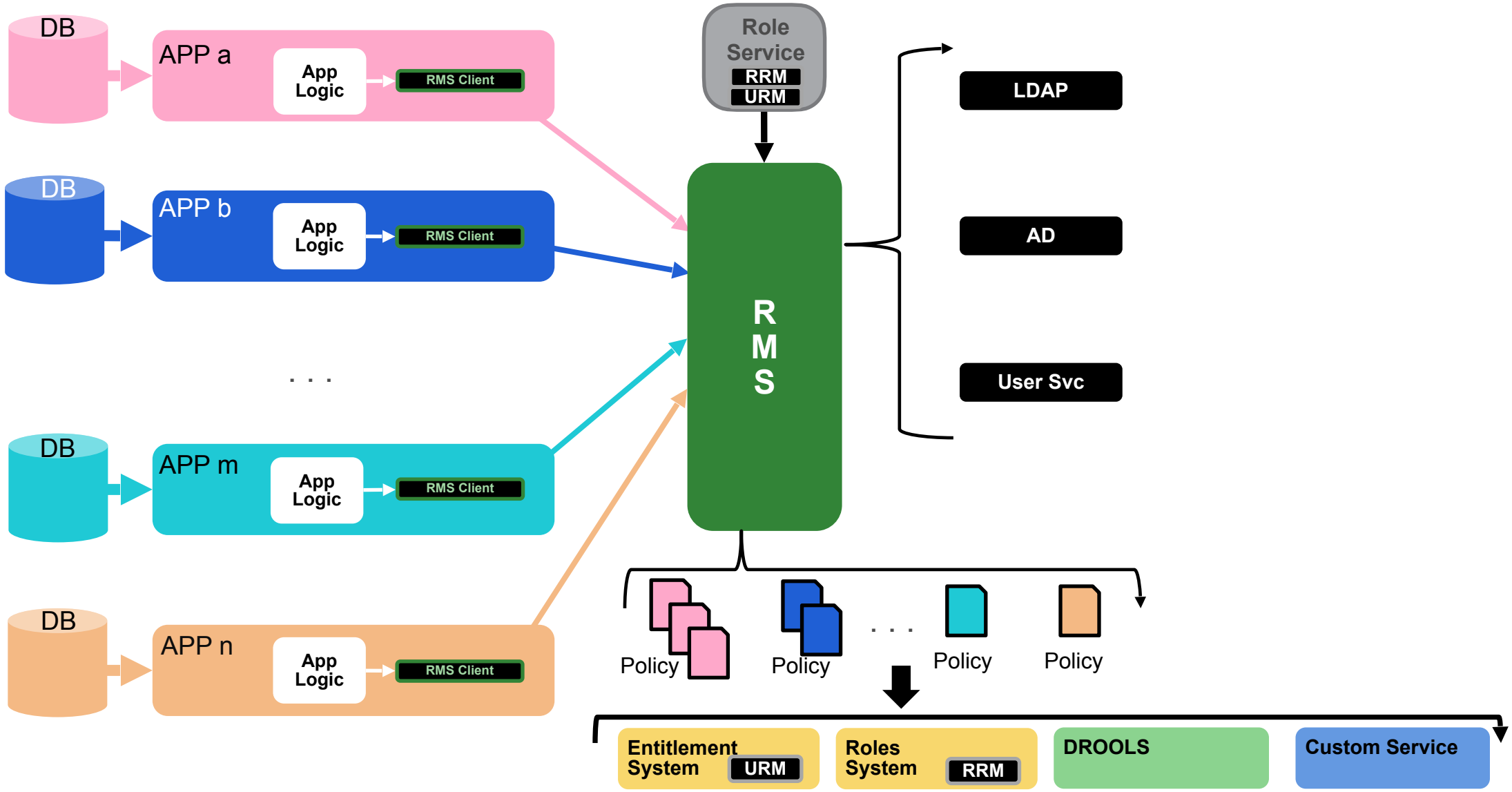
Before RMS



Post-RMS visualized

RRM Role Responsibility Mapping

URM User Role Mapping

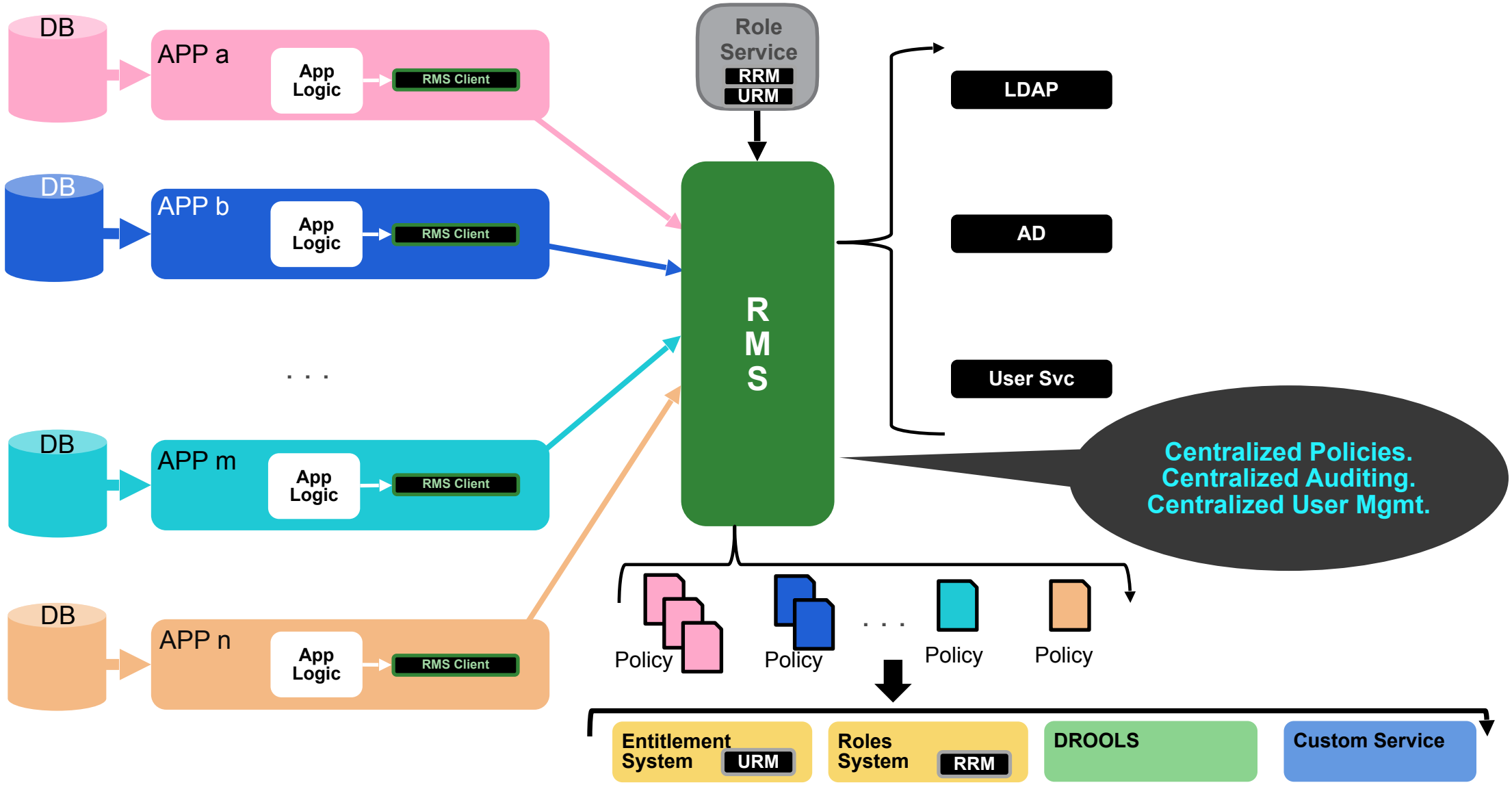


BNY MELLON

Post-RMS visualized

RRM Role Responsibility Mapping

URM User Role Mapping



BNY MELLON

BNY MELLON

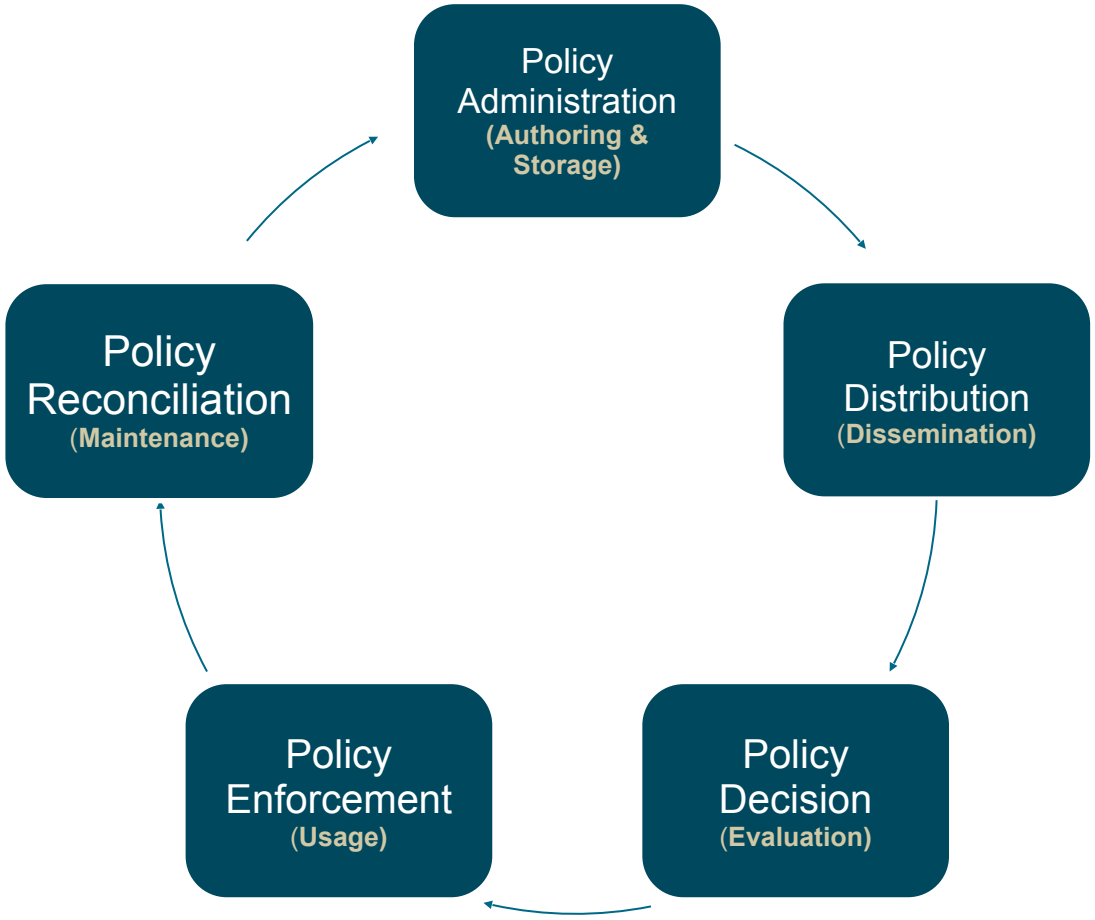
Architecting the Responsibility Management System

A platform solution for Responsibility Management

Responsibility Management - Policy Lifecycle

Responsibility Management is performed via policies

Policies have a lifecycle



More detailed flow: Appendix

BNY MELLON

Technology Choices

Technology choices for building the Responsibility Management System

Payload format: HOCON: Use case

A Case for using **H**uman-**O**ptimized **C**onfiguration **O**bject **N**otation

- Intent is to expose query-only operations.
- Policy decisions should be queried (non-mutating), thus logically GET operations.

☐ GET operations do not support a request body.

☐ GET operations may be exposed to character limits, large content not possible.

☑ POST operations allow for a request body but do not support meaningful caching.

☐ JSON and individual query parameters are quite verbose.

☑ HOCON * trims the parameter verbosity by a significant amount.

https://github.com/lightbend/config/blob/master/HOCON.md

Payload format: HOCON: Benefits

Benefits of using **H**uman-**O**ptimized **C**onfiguration **O**bject **N**otation

HOCON (see link below)

- syntax is quite simple and has low ambiguity.
- is a superset of JSON. JSON is parsed properly by HOCON parsers.
- allows the use of comments.
- allows multi-line strings.
- allows for includes and substitutions (next slide has more on this).
- has built-in durations (5d or 100ms)

https://github.com/lightbend/config/blob/master/HOCON.md

Payload format: HOCON: Features

Human-**O**ptimized **C**onfiguration **O**bject **N**otation - using *includes* and *substitutions*

generic.conf

```
{x: 10, y: ${x}, z: 5s}
```

my.conf

```
{a : { include "generic.conf" } }
```

Substitution

Inclusion

a.x = 10

a.y = 10

a.z = 5s

https://github.com/lightbend/config/blob/master/HOCON.md

Payload format: HOCON: Comparing to JSON

Human-**O**ptimized **C**onfiguration **O**bject **N**otation - sample comparisons to JSON

Sample JSON

```
foo : {
  bar : {
    baz: myvalue
  }
}
employee: {
  firstname: ”Jane"
  lastname: ”Doe"
  nested: {
    loginTimeoutInMilliSeconds: 5000
  }
  fullname: “Jane Doe”
}
standard-policy: {
  developer: "yes"
  operator: false
}
```

Sample HOCON

```
foo.bar.baz: myvalue
        ----    Or    ----
foo { bar { baz: myvalue}}
employee {
  firstname: ”Jane"
  lastname: ”Doe"
  nested {
    loginTimeout: 5s
  }
  fullname: ${employee.firstname} ${employee.lastname}
}
standard-policy {
  developer: "yes"
  operator: false
}
```

Java Collections Library: Eclipse Collections

Key highlights

Eclipse Collections (see link below)

- Rich, concise and readable APIs.
- Clear mutable and immutable hierarchies for collection types.
- Memory efficient containers.
- Optimized eager APIs instead of Java Collection Framework's lazy APIs.
- Improved code readability.
- Ease of learning thanks to several Code Katas.

https://www.eclipse.org/collections/

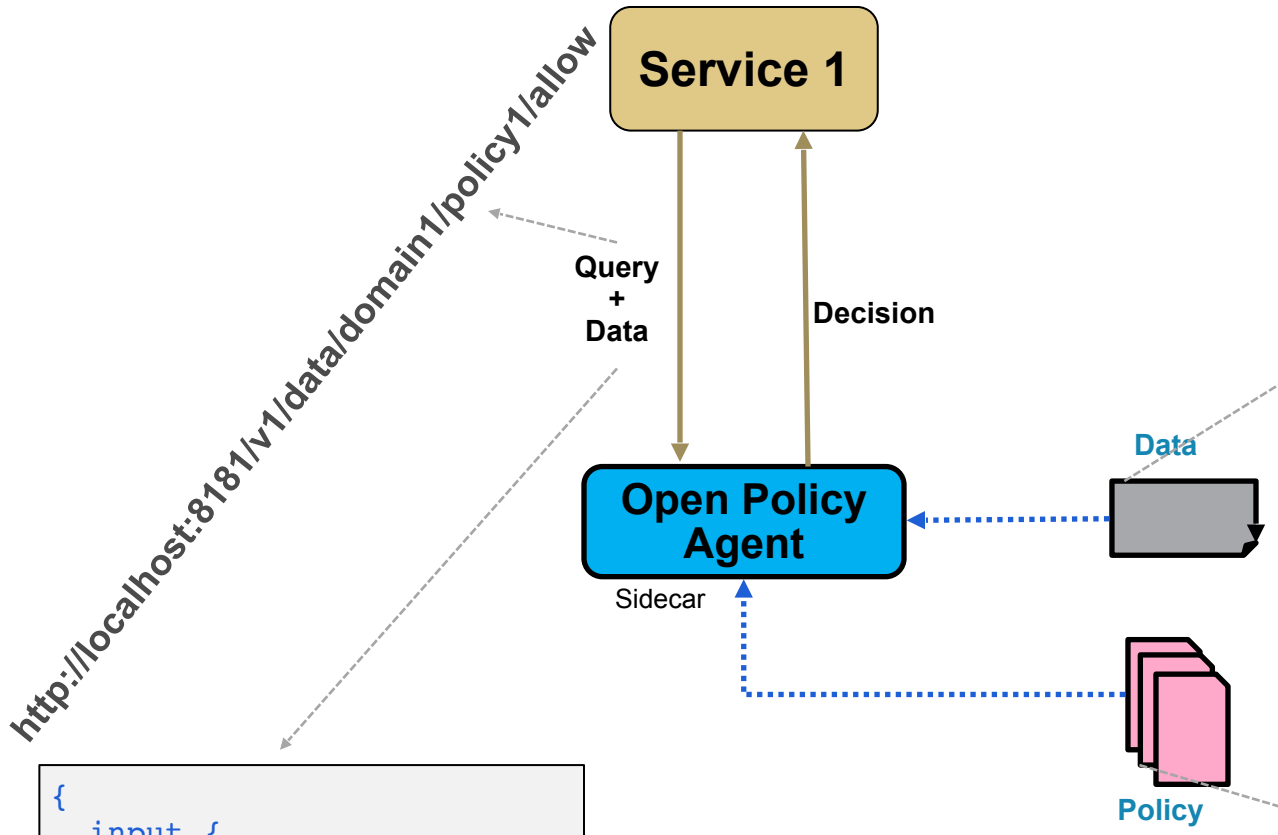
Policy Engine: Open Policy Agent

Key highlights

- Open Policy Agent (OPA) (see link below) is an open source general purpose policy engine.
- Uses “rego” (inspired by Datalog) as a declarative native query language.
- Policies are written as rulesets (similar to functions).
- Policies can be queried as RESTful POST operations.
- Data and policy publishing is via RESTful PUT operations.
- Can be launched as a **library** for a **service**, an **independent daemon** or **sidecar**.
- Decision in RMS was to use OPA as a **sidecar**.

https://www.openpolicyagent.org/

Open Policy Agent: Usage Pattern



Reference Data (available from Policy Information Point)

```
[
  {
    "name": "bucket1",
    "clients": [
      {
        "name": ”client1",
        "access": ["READ”, “WRITE”]
      },
      {
        "name": ”client2",
        "access": ["WRITE"]
      }
    ]
  },
  {
    "name": "bucket2",
    "clients": [
      {
        "name": ”client1",
        "access": [”READ"]
      }
    ]
  }
]
```

data.json

```
{
  input {
    bucket: "bucket2",
    client: ”client1",
    access: "READ"
  }
}
```

Query Payload (Inputs from invoker per request)

policy.rego

```
package domain1.policy1

import data.domain1.policy1.buckets

default allow = false

allow {
     buckets[i].name == input.bucket
     buckets[i].clients[j].name == input.client
     buckets[i].clients[j].access[k] == input.access
}
```

Policy Logic (available from Policy Distribution Point)

BNY MELLON

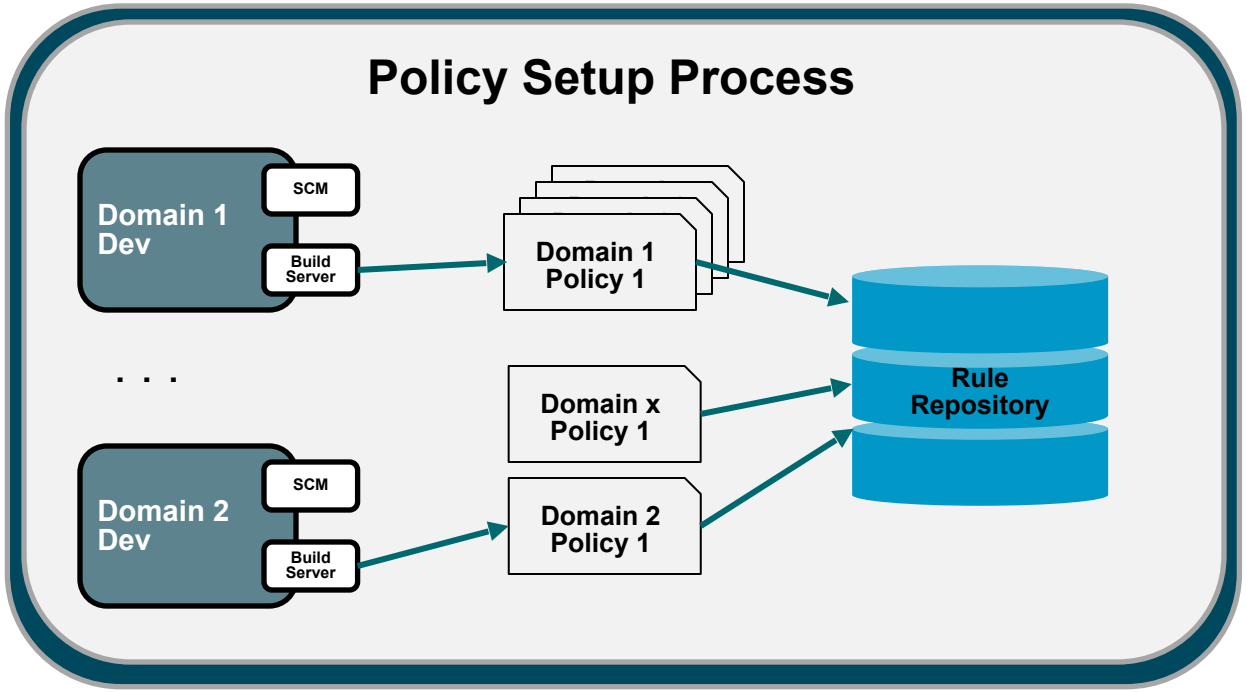
Responsibility Management System

Architecture (Version 1)

A ***Federated*** Responsibility Management Service

RMS Architecture - Version 1: Federated

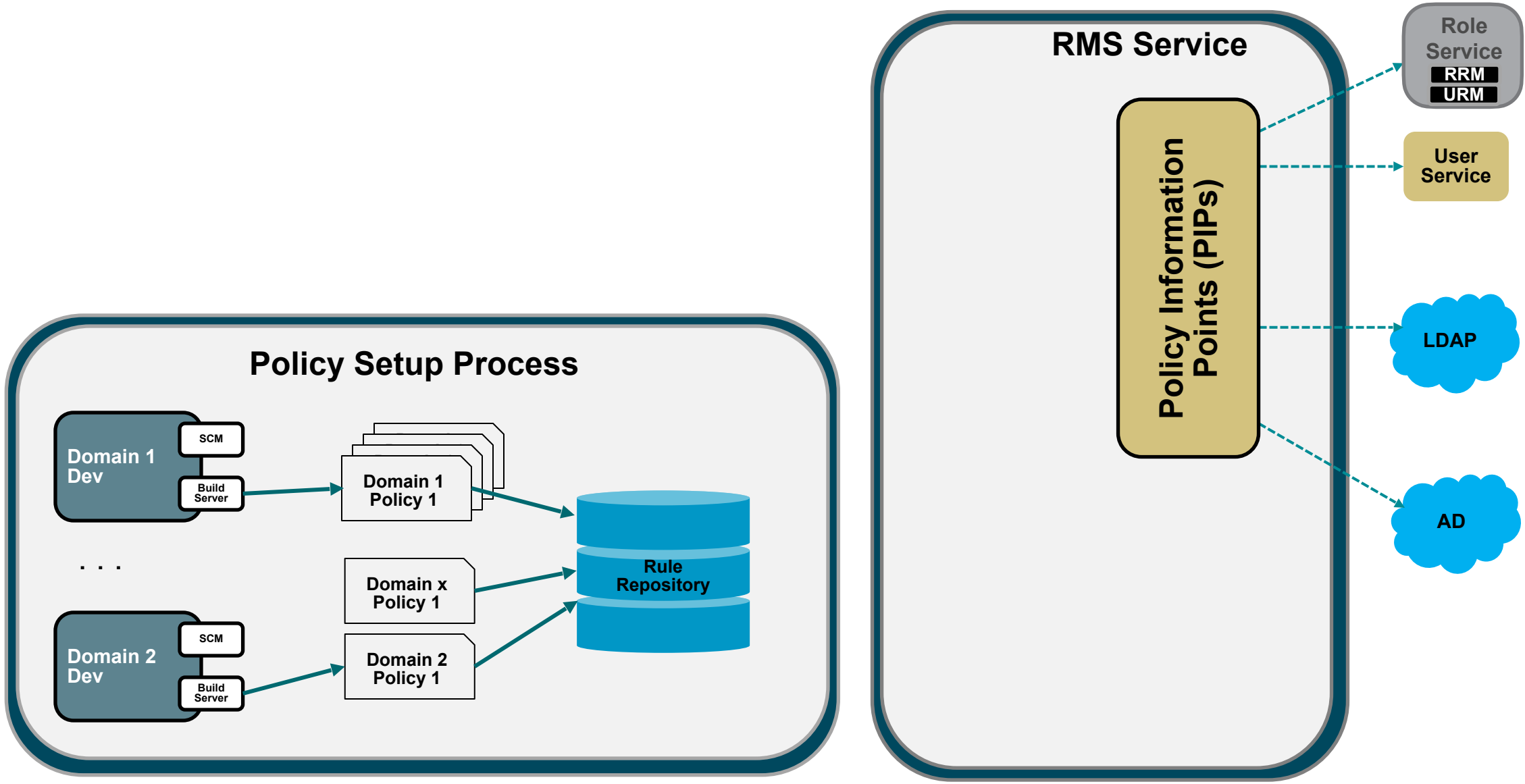
RRM	Role Responsibility Mapping
URM	User Role Mapping



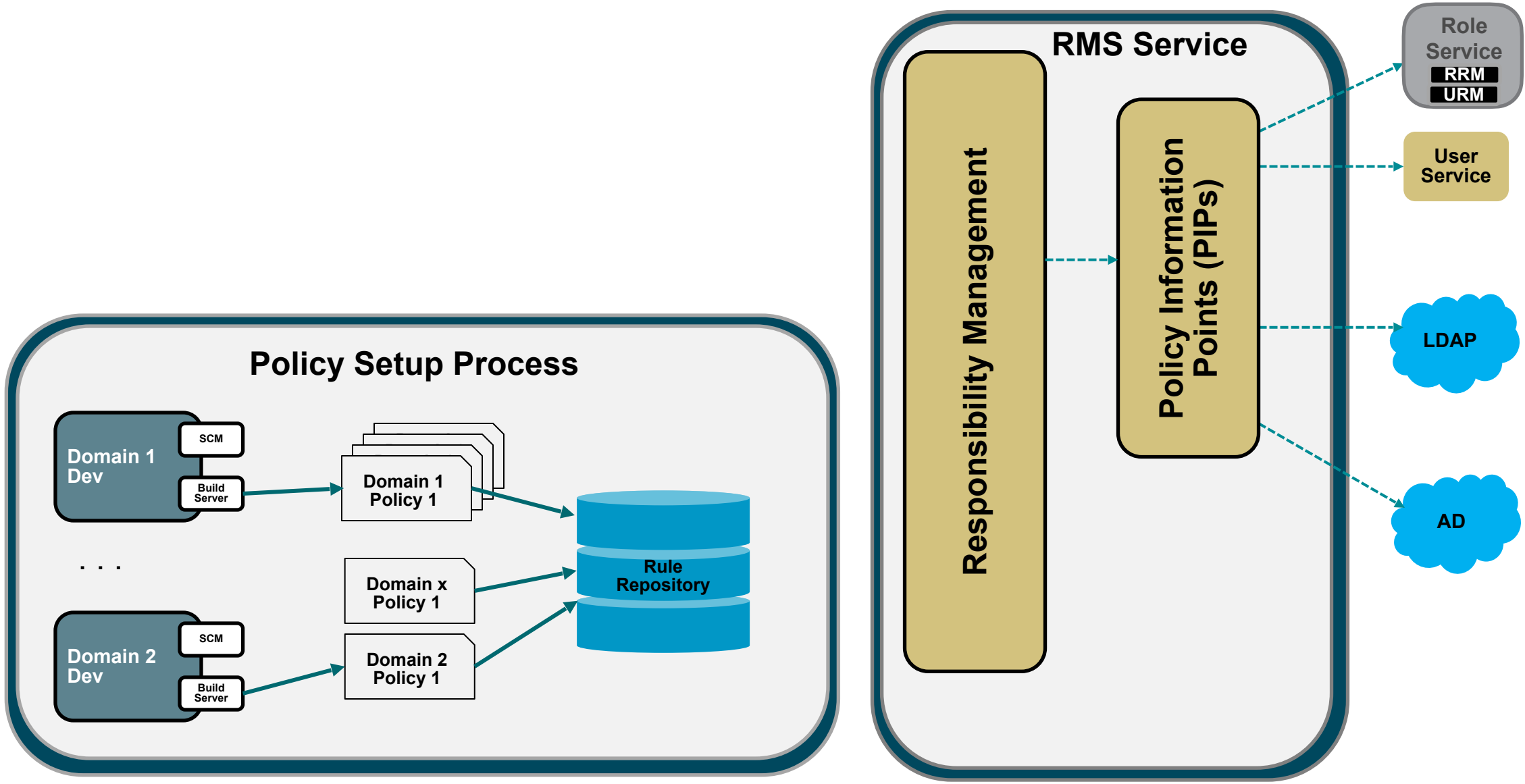
RMS Architecture - Version 1: Federated

RRM Role Responsibility Mapping

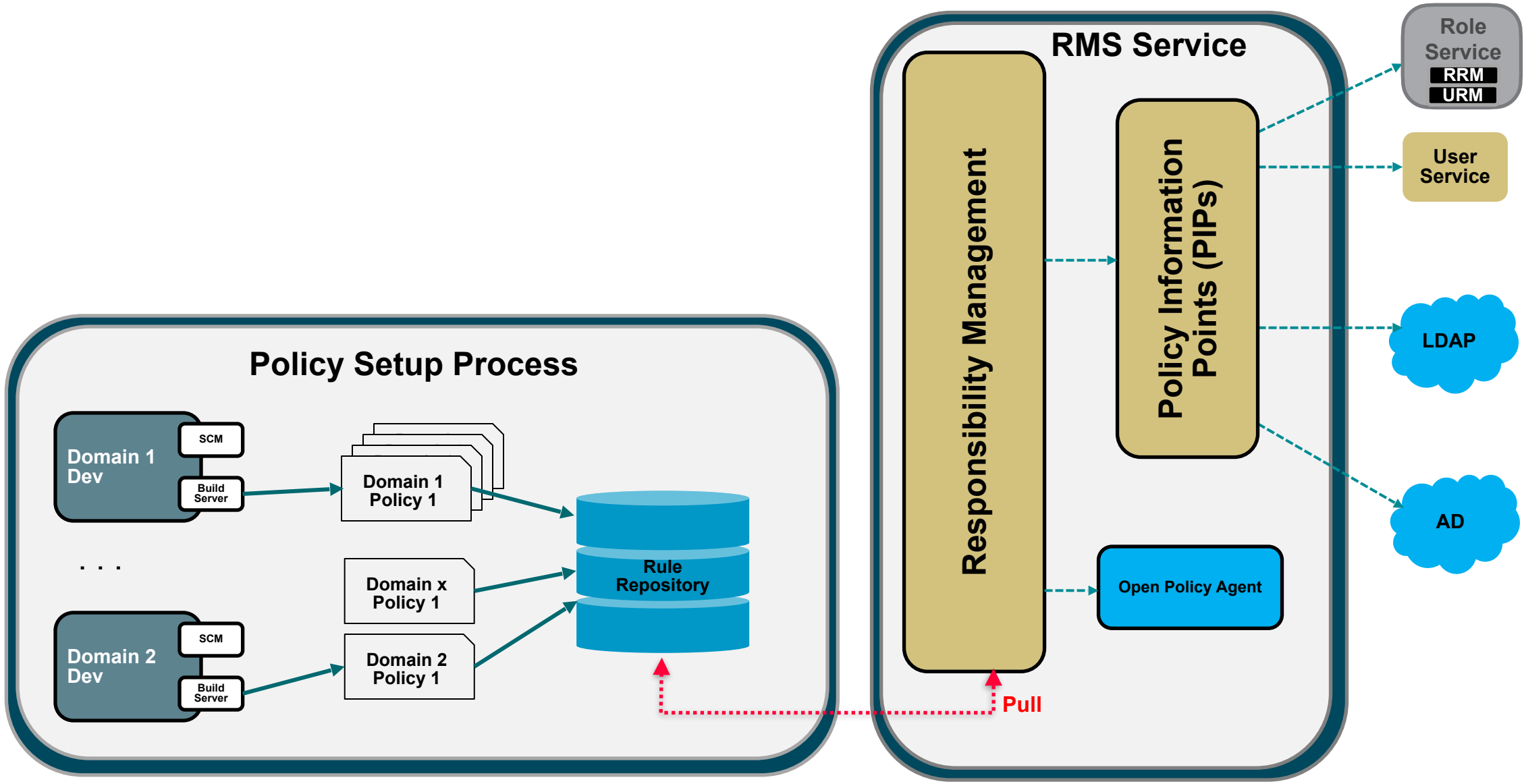
URM User Role Mapping



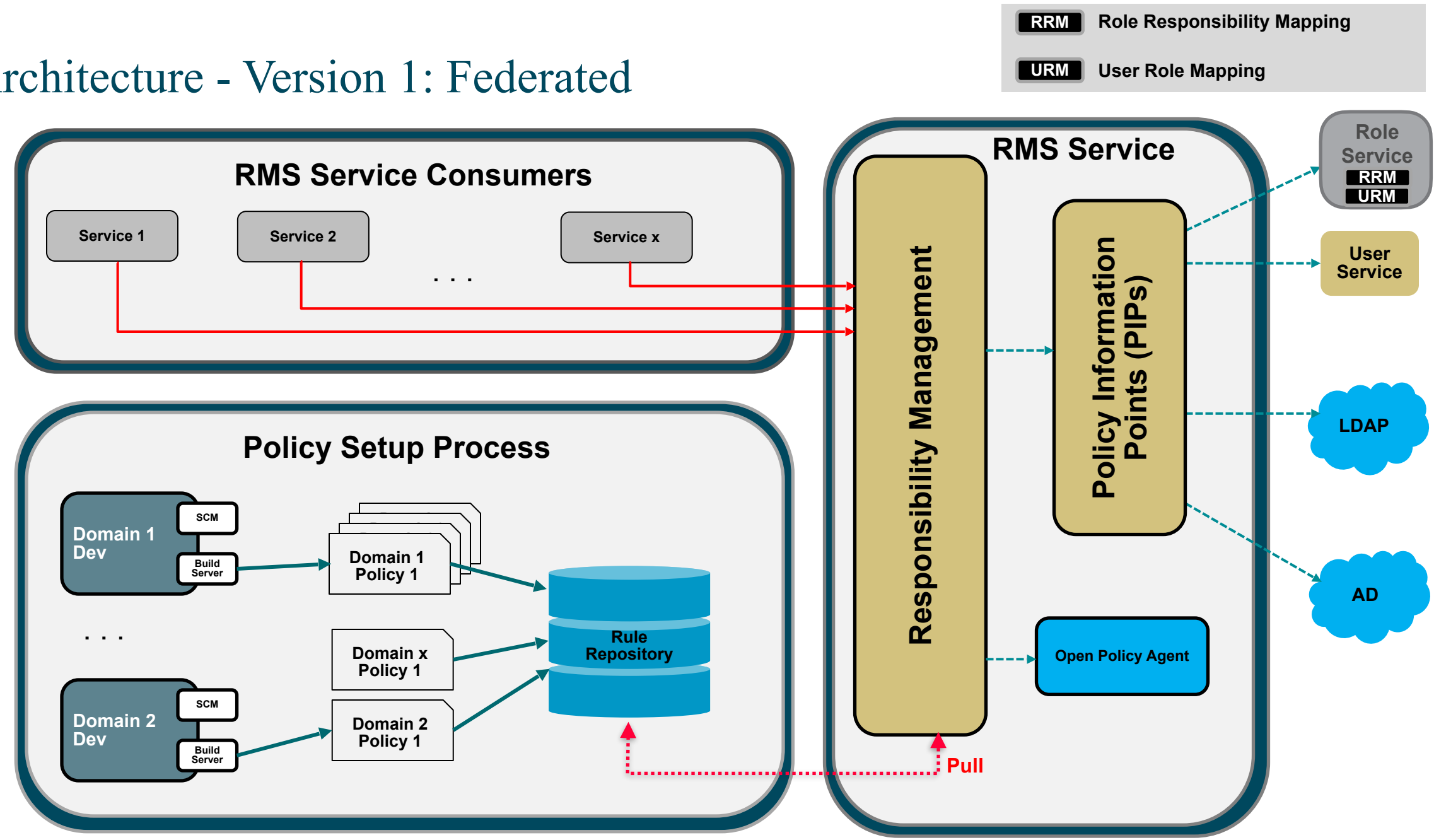
RMS Architecture - Version 1: Federated



RMS Architecture - Version 1: Federated



RMS Architecture - Version 1: Federated



Federated Architecture: Issues Faced

Key issues

- Segregation and information-barrier needs **implied more work**.
- A rogue policy script could lead to **loss of service for all domains**.
- RM Service **became the gatekeeper for testing and coverage**.
- RM Service **had to establish a release-train model** to pick up new policies.
- Out-of-band policy changes lead to **intermittent service-unavailability**.
- **Observation**: Policy changes were more frequent when a new domain onboards.

BNY MELLON

Responsibility Management System

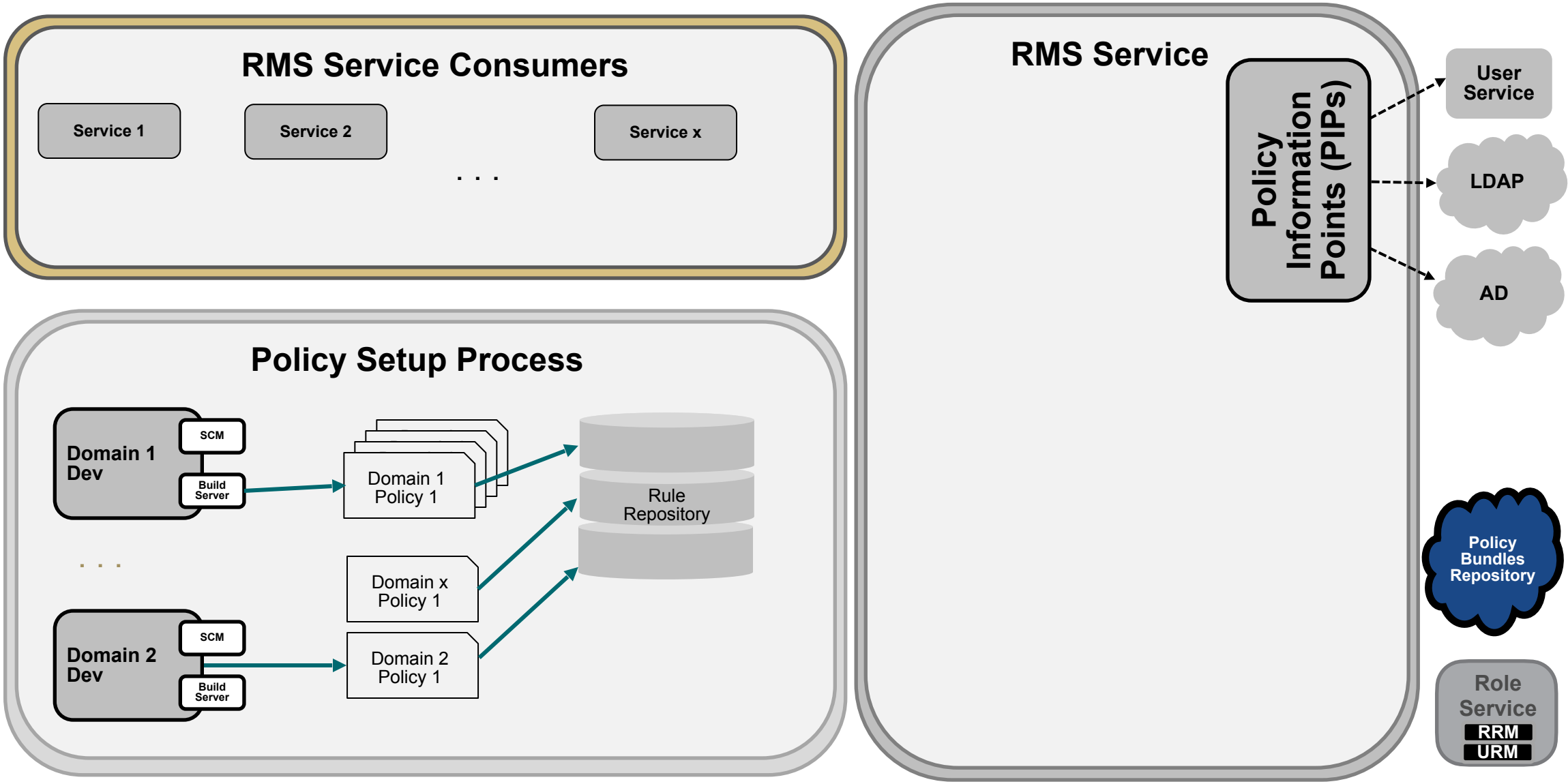
Architecture (Version 2)

A ***Distributed*** Responsibility Management Service

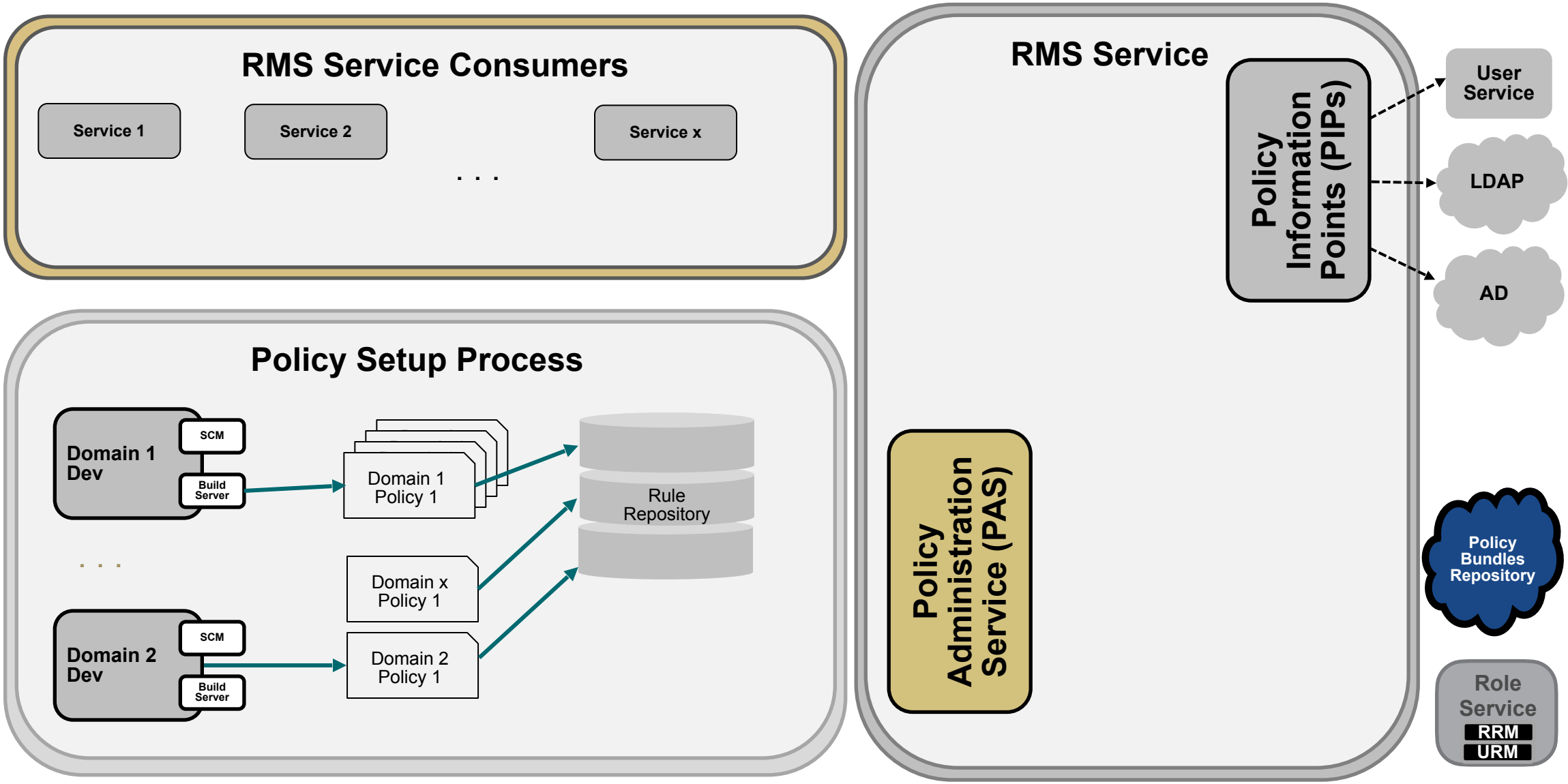
RMS Architecture - Version 2: Distributed

RRM Role Responsibility Mapping

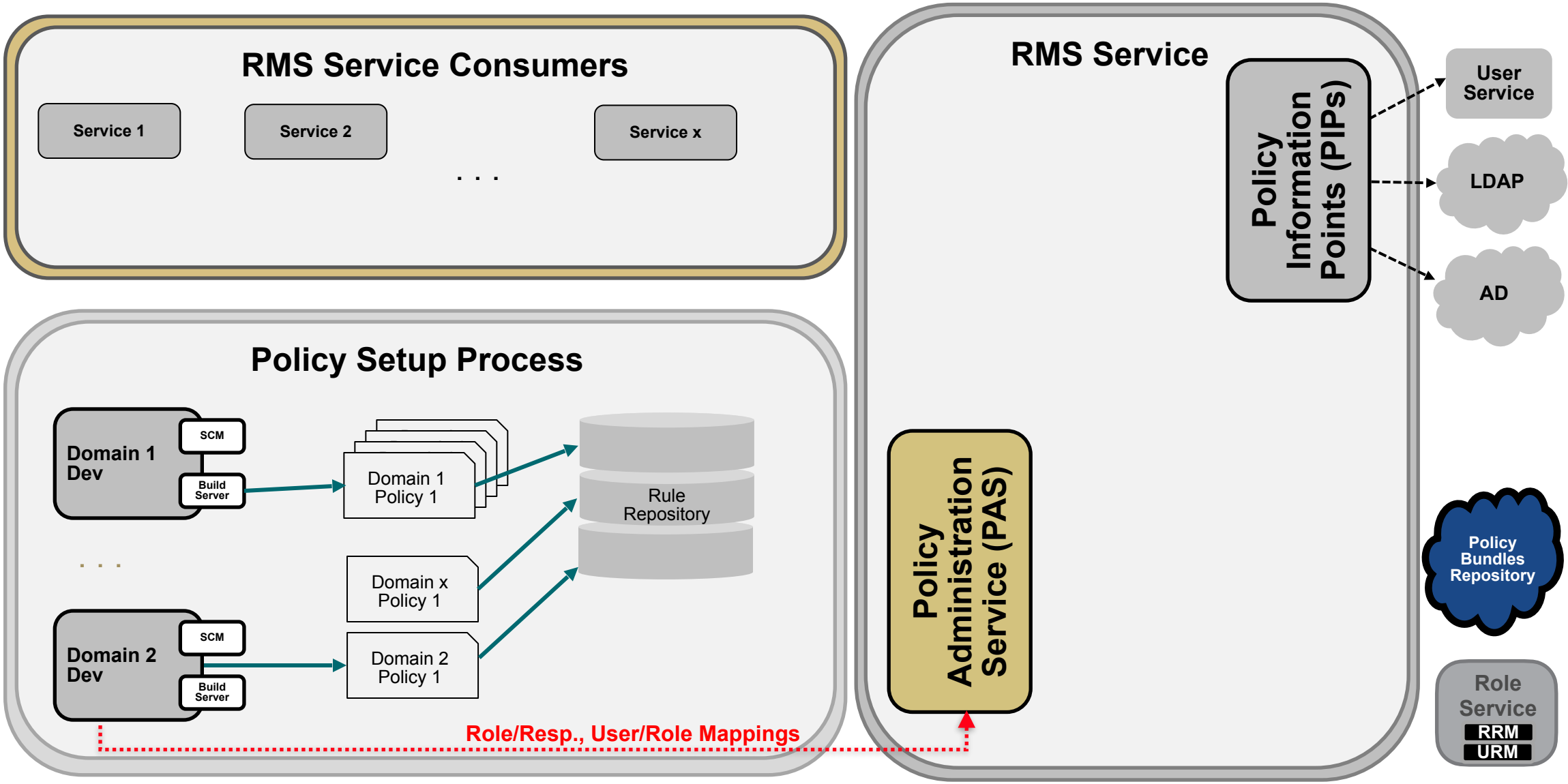
URM User Role Mapping



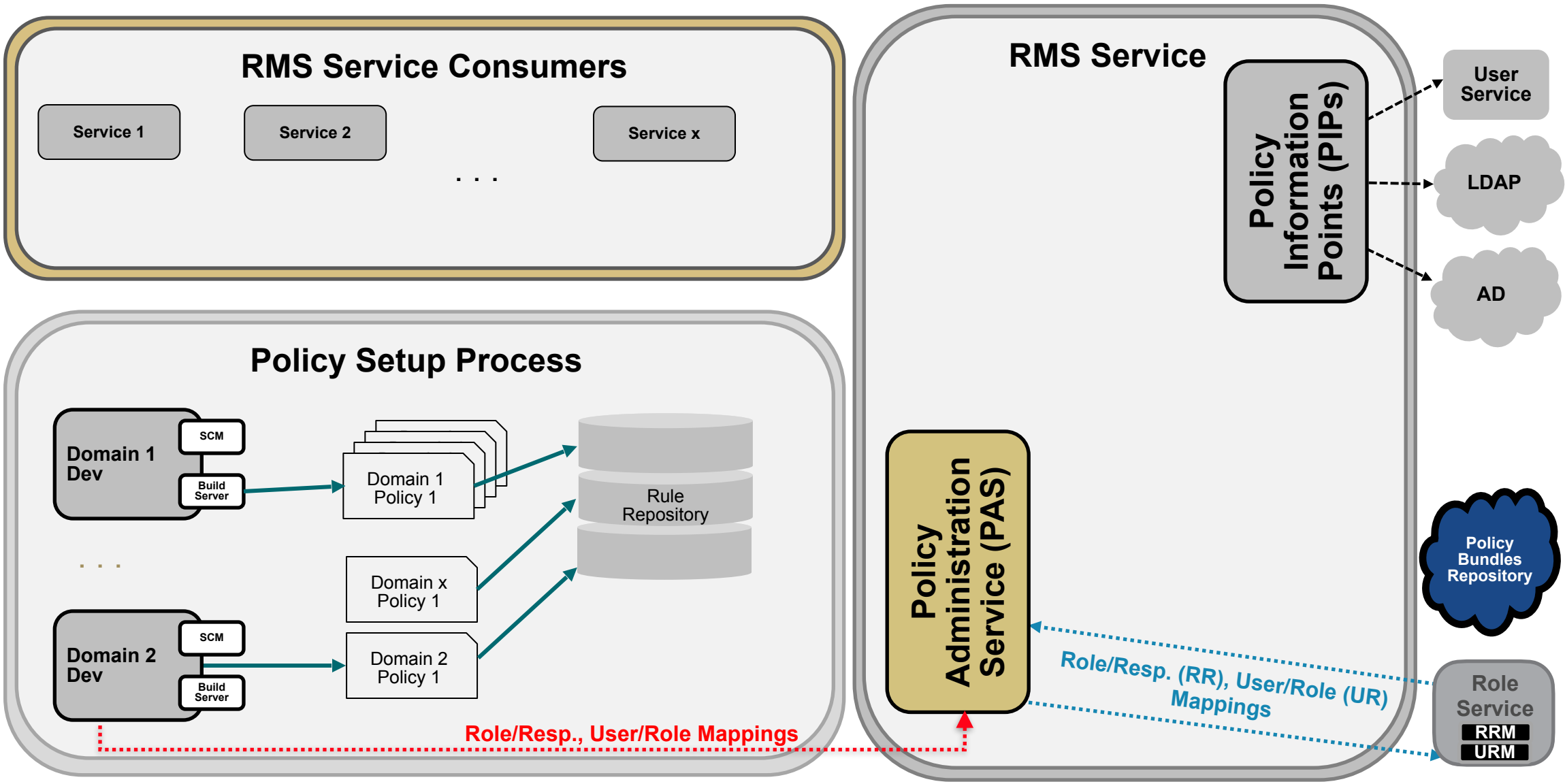
RMS Architecture - Version 2: Distributed



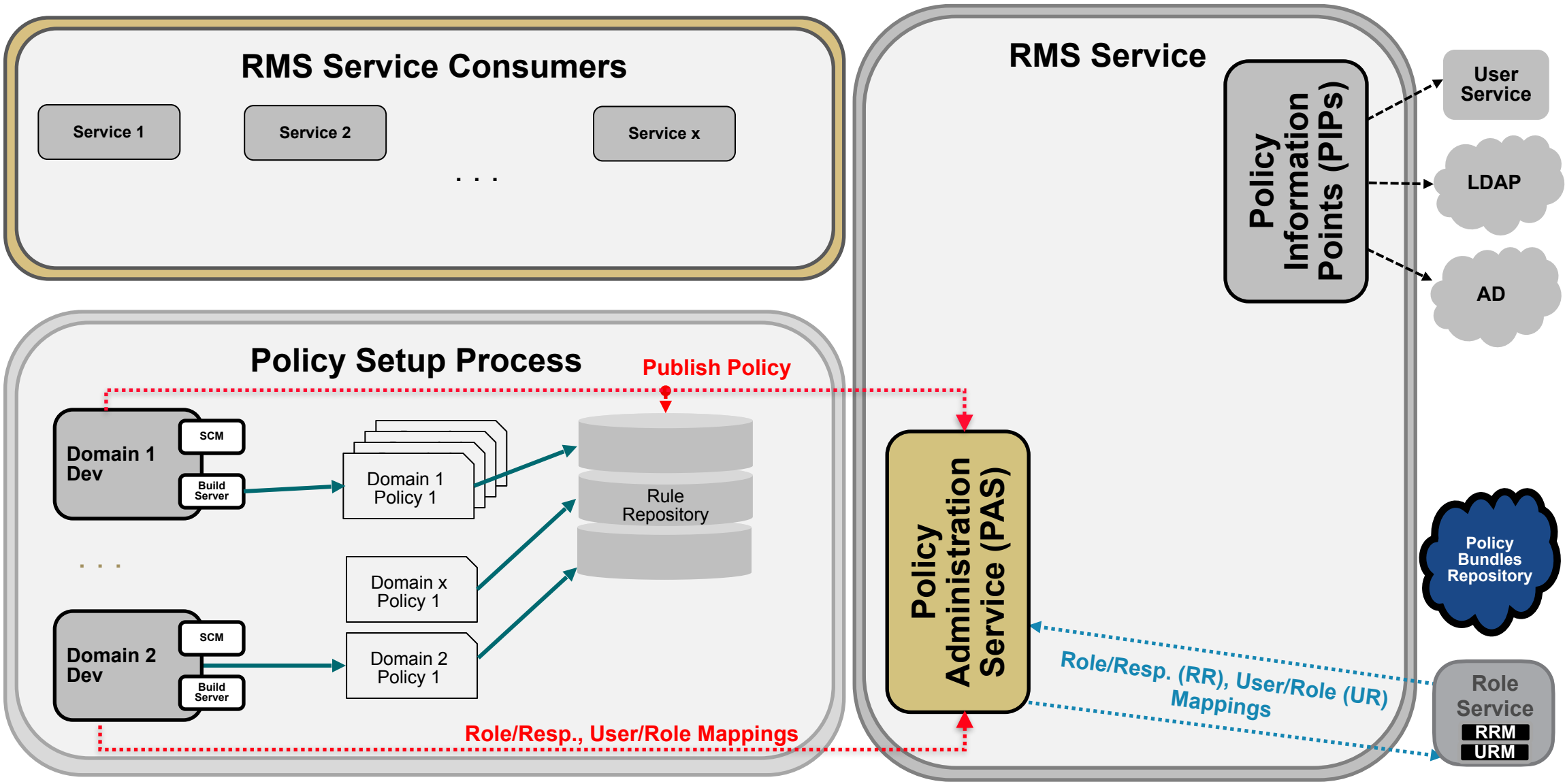
RMS Architecture - Version 2: Distributed



RMS Architecture - Version 2: Distributed



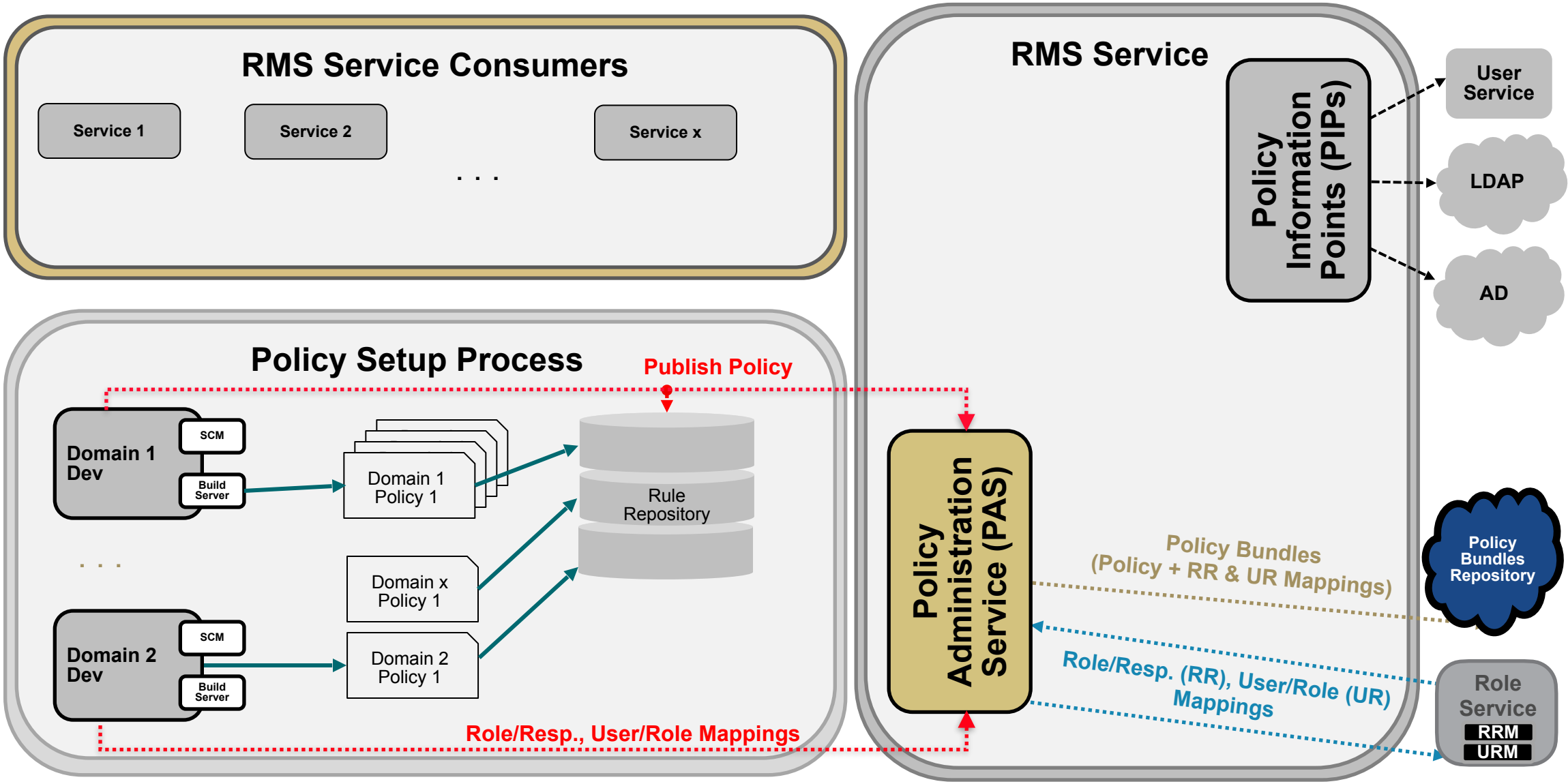
RMS Architecture - Version 2: Distributed



RMS Architecture - Version 2: Distributed

RRM Role Responsibility Mapping

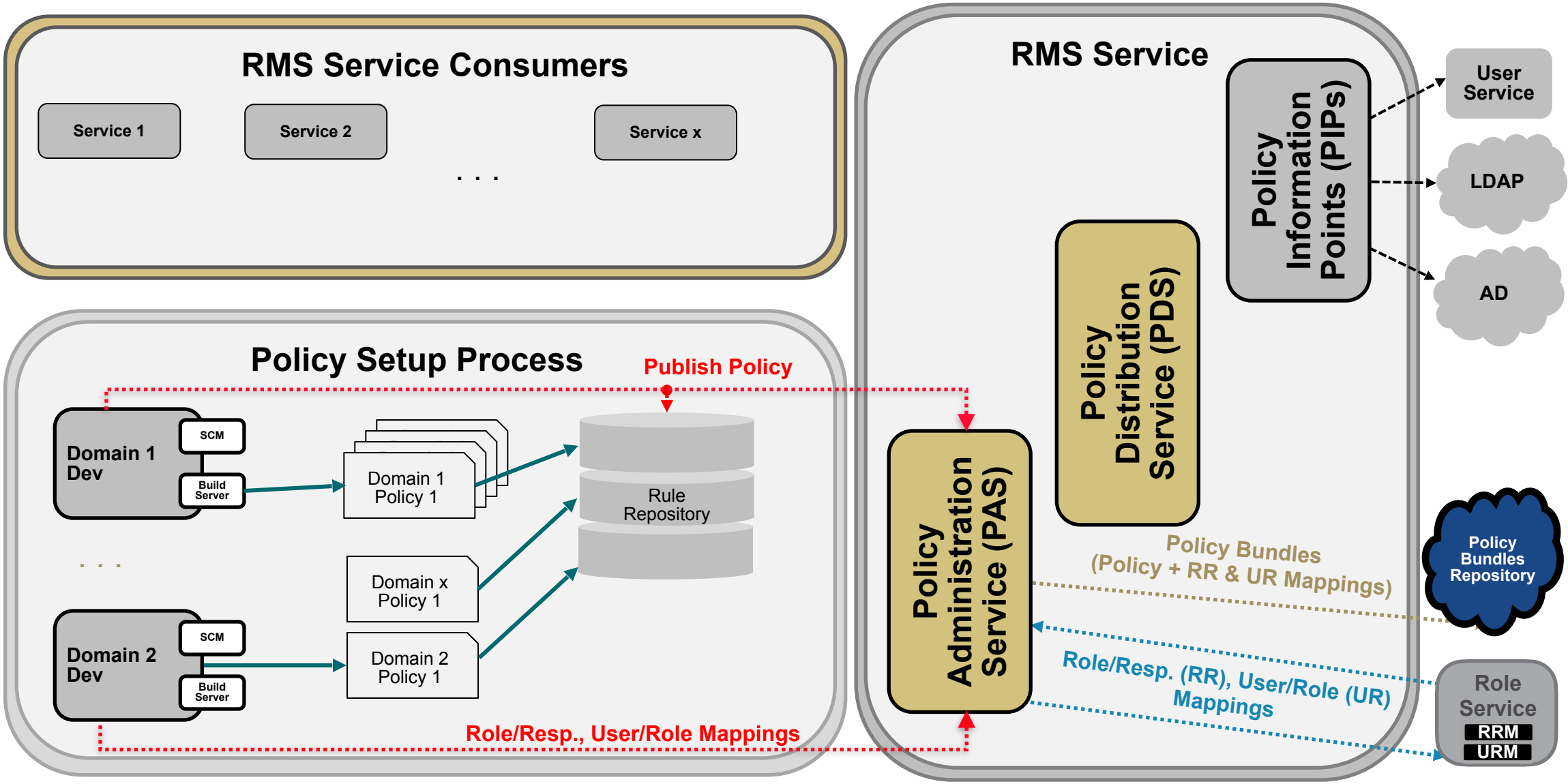
URM User Role Mapping



RMS Architecture - Version 2: Distributed

RRM Role Responsibility Mapping

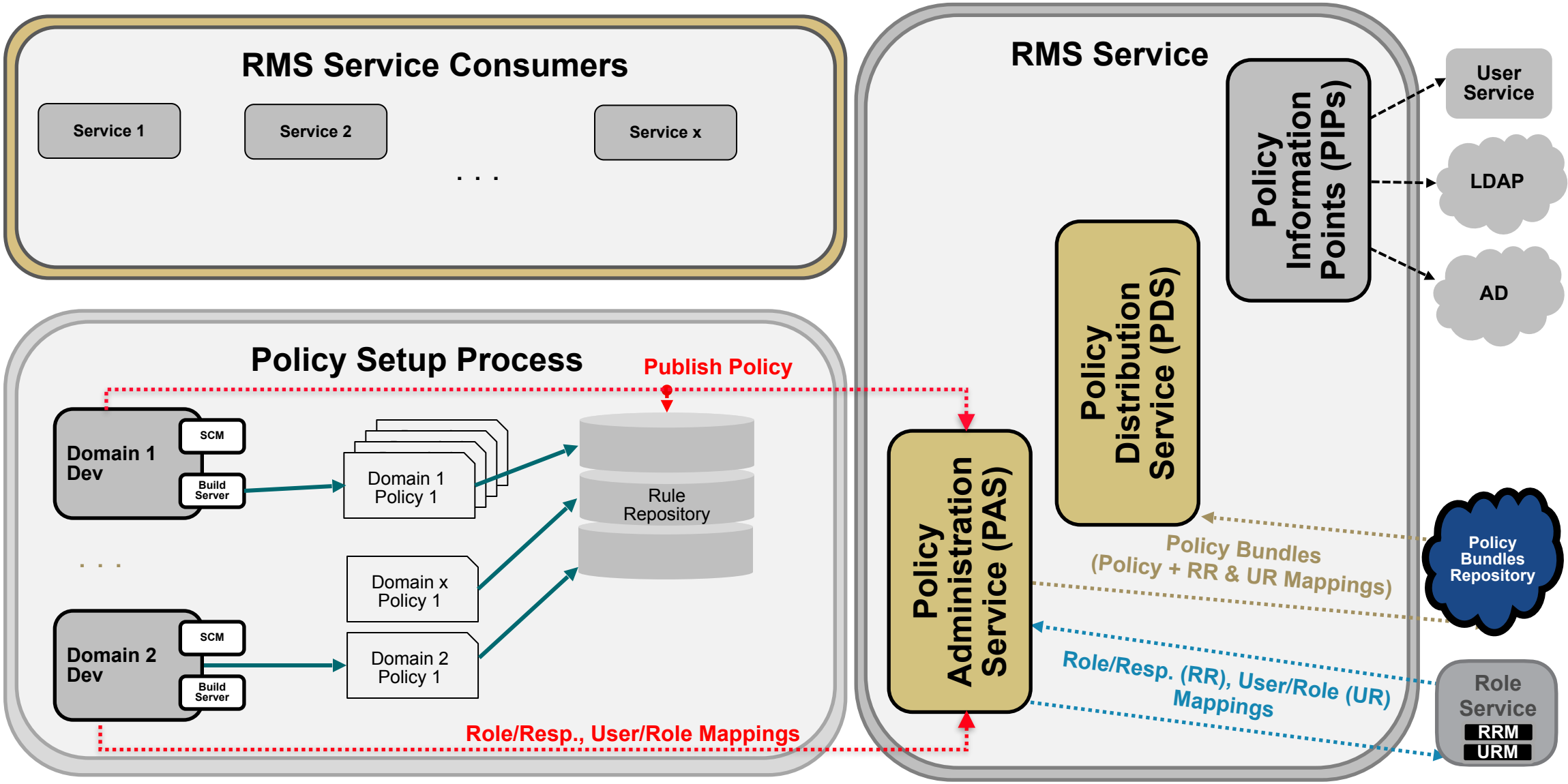
URM User Role Mapping



RMS Architecture - Version 2: Distributed

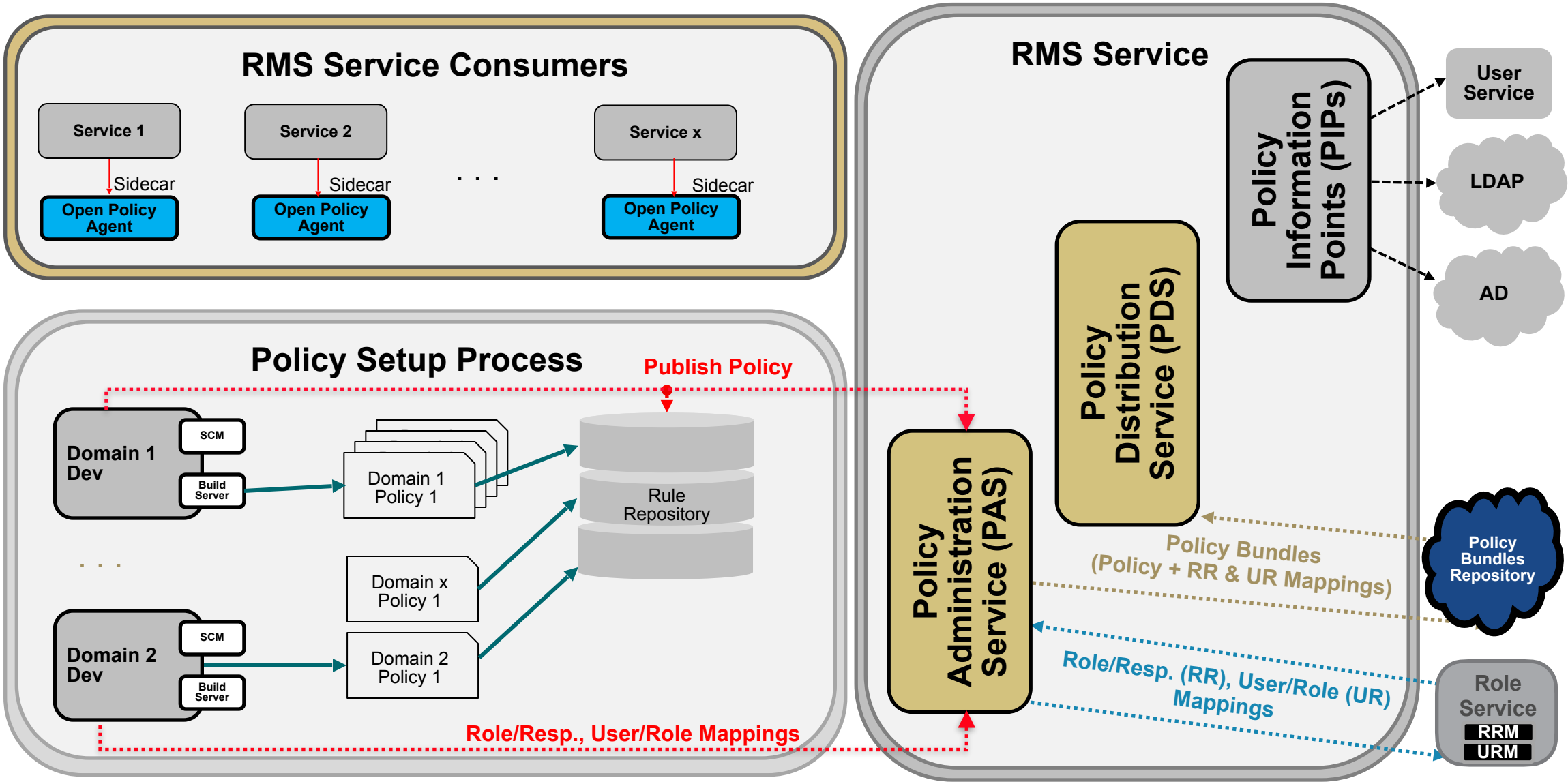
RRM Role Responsibility Mapping

URM User Role Mapping



BNY MELLON

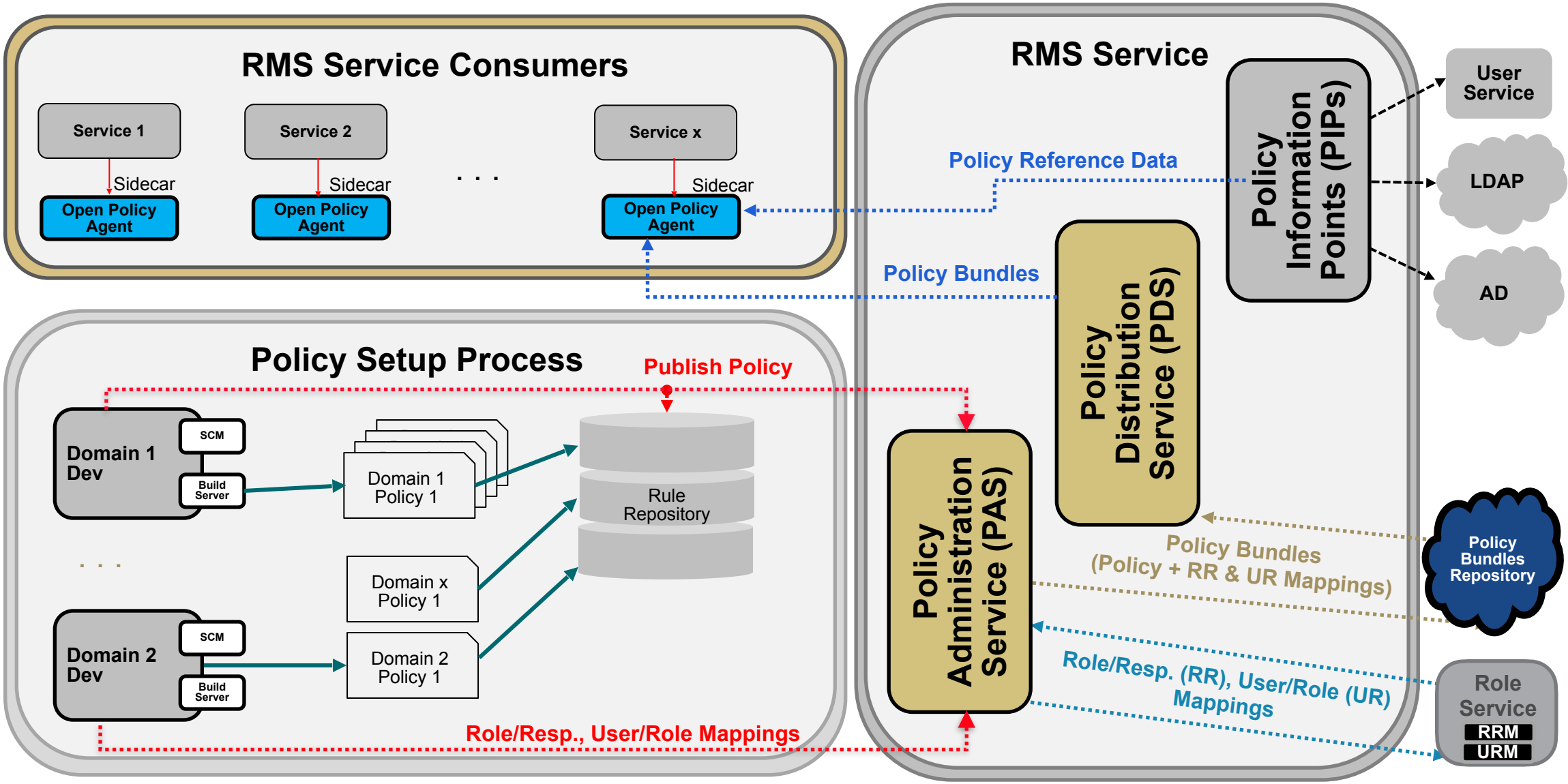
RMS Architecture - Version 2: Distributed



RMS Architecture - Version 2: Distributed

RRM Role Responsibility Mapping

URM User Role Mapping



Benefits of a Distributed Policy Management Architecture

Comparing Version 1 (federated single policy engine) with Version 2 (distributed policy engines)

	V1 Federated Policy Engine	V2 Distributed Policy Engine
Segregation and Information Barriers	Requires **additional work**	Is implicit, **no additional work**
Impact of a rogue policy script	Outage for **all domains**	Outage **only for the specific domain**
Gatekeeping for testing and coverage	Requires **RMS** to be the gatekeeper	Requires **domain** to be the gatekeeper
Strategy for new and updated policies	**Needed a Release Train model**	**A domain can push policies on-demand**
Impact of ad-hoc policy changes	RMS Downtime for **all domains**	RMS Downtime for **the changed domain**
Implicit RBAC Support	-	**Available**

Policy Bundles Repository

Policy bundles repository stored enriched policy archives.

Enriched policy bundles are archives that contain:

- Policy file(s), **specific to the domain**.
- Policy static data, **specific to the domain**.
- Standard RMS OPA policy rego files **common across all domains**.

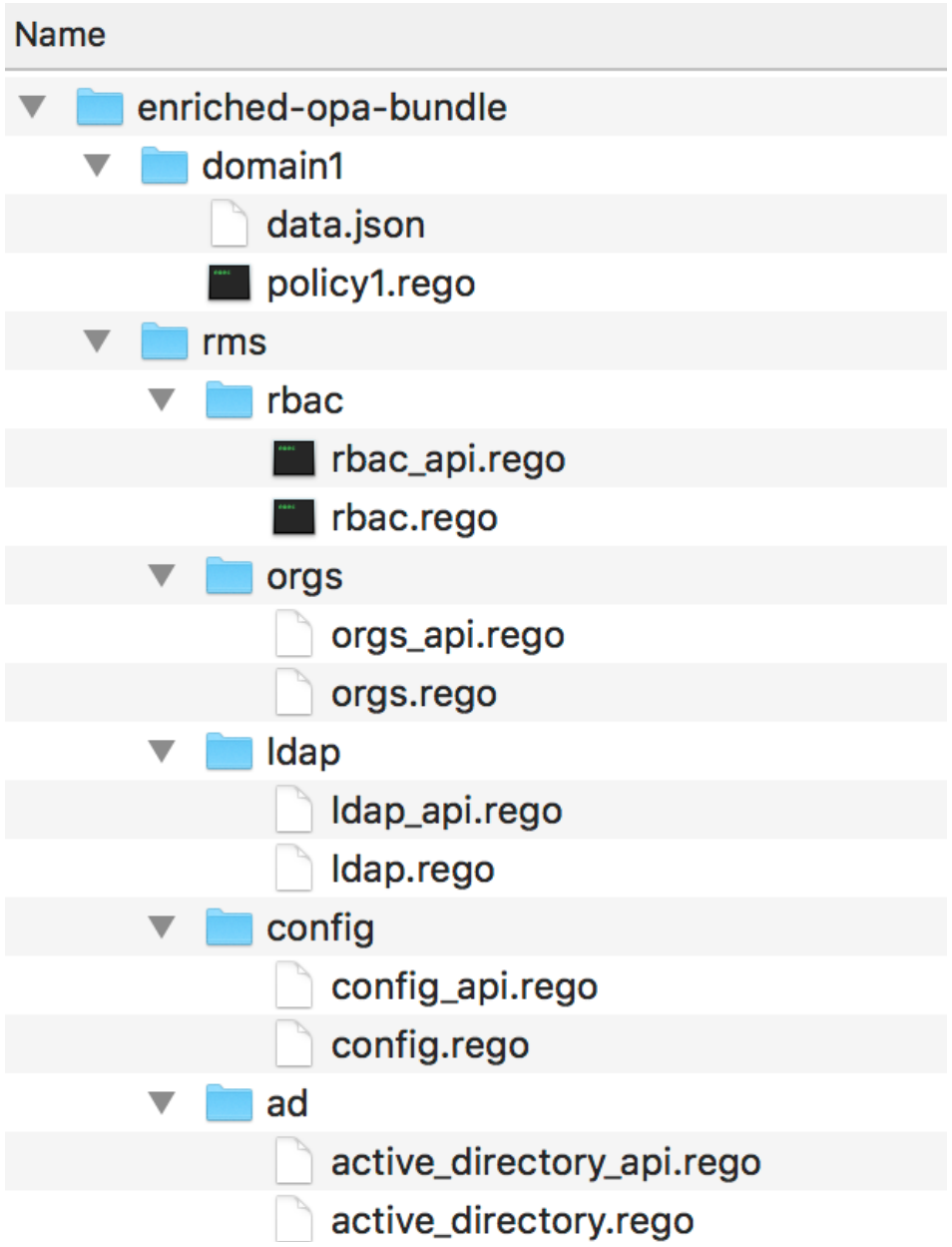
Policy Bundles Repository

Folder structure in policy bundles repository :

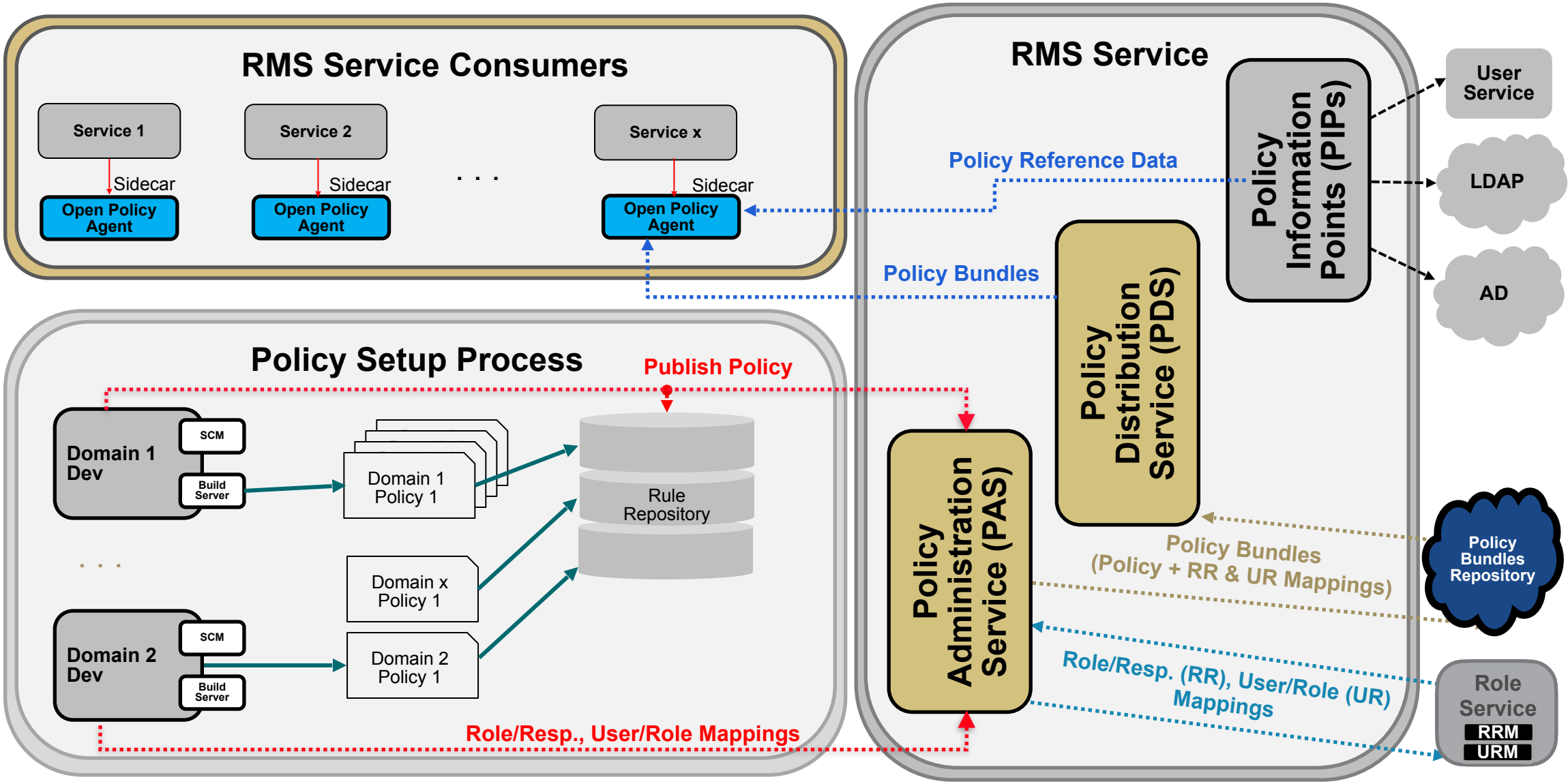
- **<domain>**
  - **<policy>**
    - **<version>**
      - **<policy bundles>**

Example:

- domain1
  - policy1
    - 1.0.0
      - enriched-opa-bundle.tar.gz



RMS Architecture - Version 2: Distributed



Policy Agent: Setup

- Open Policy Agent (the executable)
- Open Policy Agent – Configuration
- Open Policy Agent – Dockerfile command

Policy Agent: Setup: Configuration Files

OPA Configuration file (located at **${configPath}**)

```
services:
  - name: domainPolicies
    url: policyDistributionServiceUrl/
    allow_insecure_tls: true
bundle:
  name: policyDomain/policyName/policyVersion
  service: domainPolicies
  polling:
    min_delay_seconds: minDelaySeconds
    max_delay_seconds: maxDelaySeconds
```

Policy Agent: Setup: Dockerfile Commands

OPA launch command (used in the Dockerfile)

```
exec ./opa run --server --log-level=debug –c ${configPath}
```

RBAC Policy Library

```
package rbac

user_has_responsibility(userId, action, resource) {
  role := roles[_]

  responsibility := role.responsibilities[_]
  does_resource_match(resource, responsibility)

  responsibility.actions[_] = action

  is_user_a_member(userId, role)
}

is_user_a_member(userId, role) {

  ...
}
```

rbac.rego

Application Policy

```
package application1

default allow = false

allow {
    data.rbac.user_has_responsibility(
              input.userid, input.action,
              input.service)
}
```

policy.rego

Sample Role Data Excerpts

```
{
  "name": "App User",
  "responsibilities": [
    {
      "resource":
        "service.1",
      "actions": [
        "provision"
      ]
    },
    {
      "resource":
        "service.2",
      "actions": [
        "provision"
      ]
    }
  ],
  "members": [
    "EVERYONE"
  ]
}
```

data1.json

```
{
  "name": "App Admin",
  "responsibilities": [
    {
      "resource":
         "regexp:service\\..*",
      "actions": [
        "create",
        "update",
        "delete",
        "view"
      ]
    }
  ],
  "members": [
    "org:abc"
  ]
}
```

data2.json

BNY MELLON

IDE Productivity

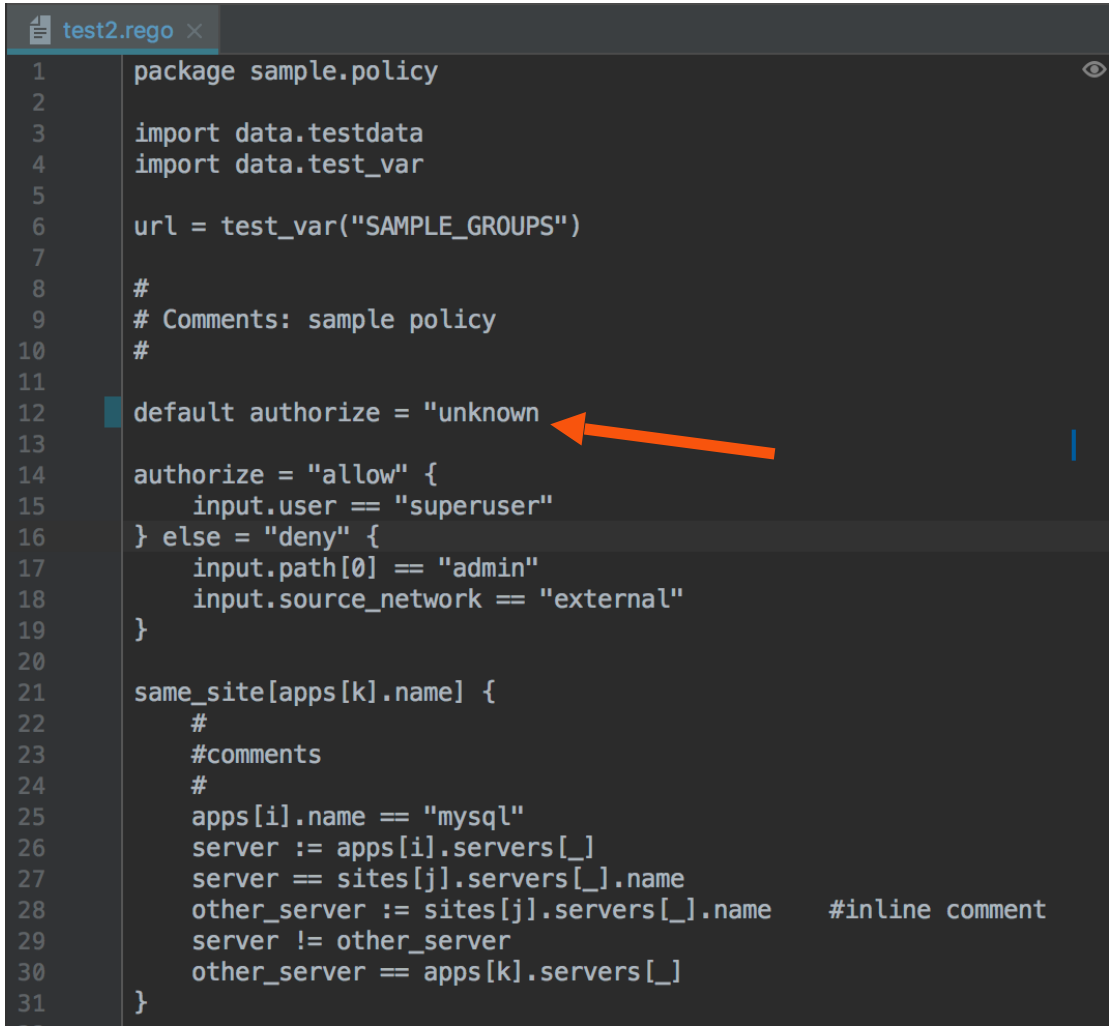
An IDE plugin tool for the Open Policy Agent file authoring

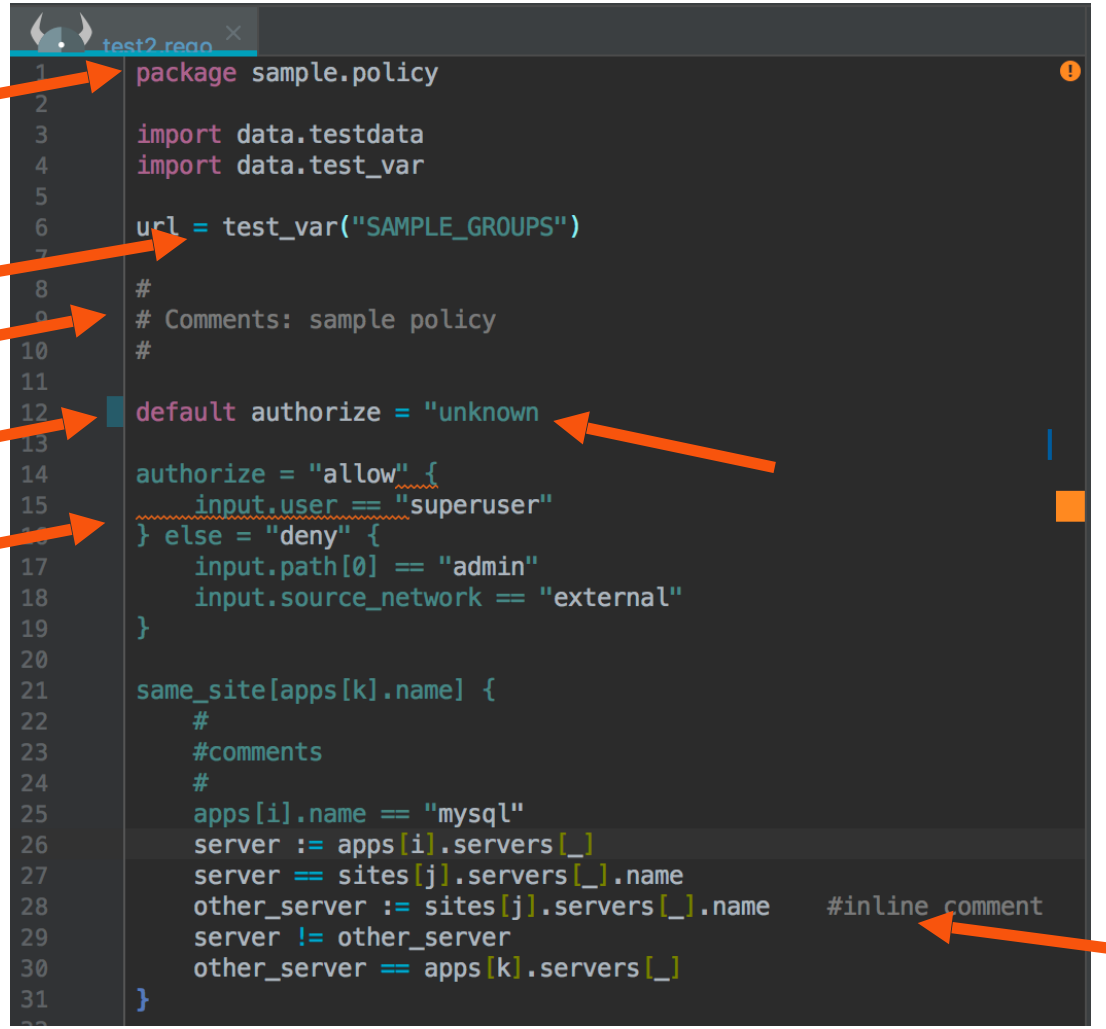
Open Policy Agent: IntelliJ Plugin

- OPA IntelliJ Plugin is *functional* work-in-progress policy editor.
- The editor parses and validates OPA policy.
- Relies on the OPA language reference linked * below.
- Can be customized for editor color schemes in IntelliJ.
- Work continues on indentation, run configurations and variable tracking.

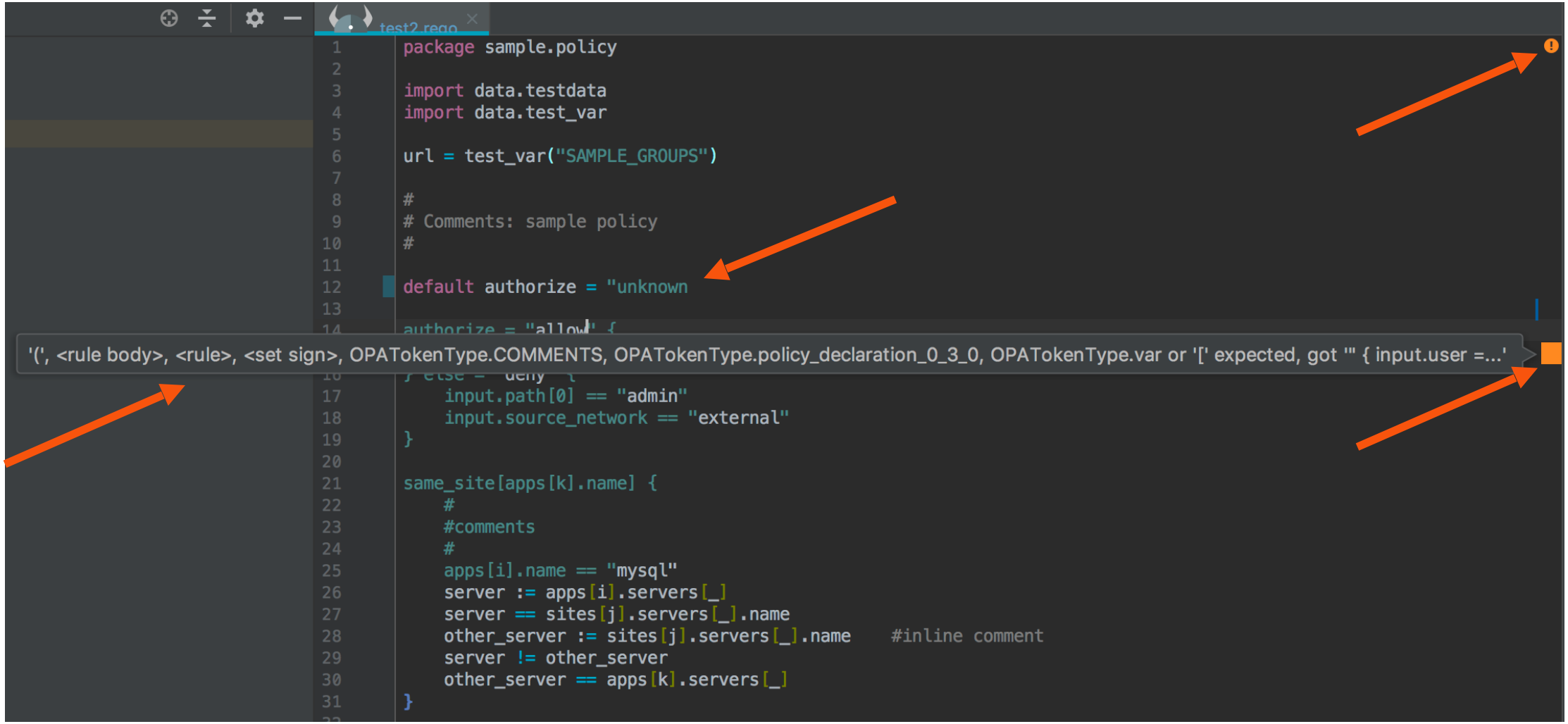
https://www.openpolicyagent.org/docs/latest/language-reference/

Open Policy Agent: IntelliJ Plugin: Before and After

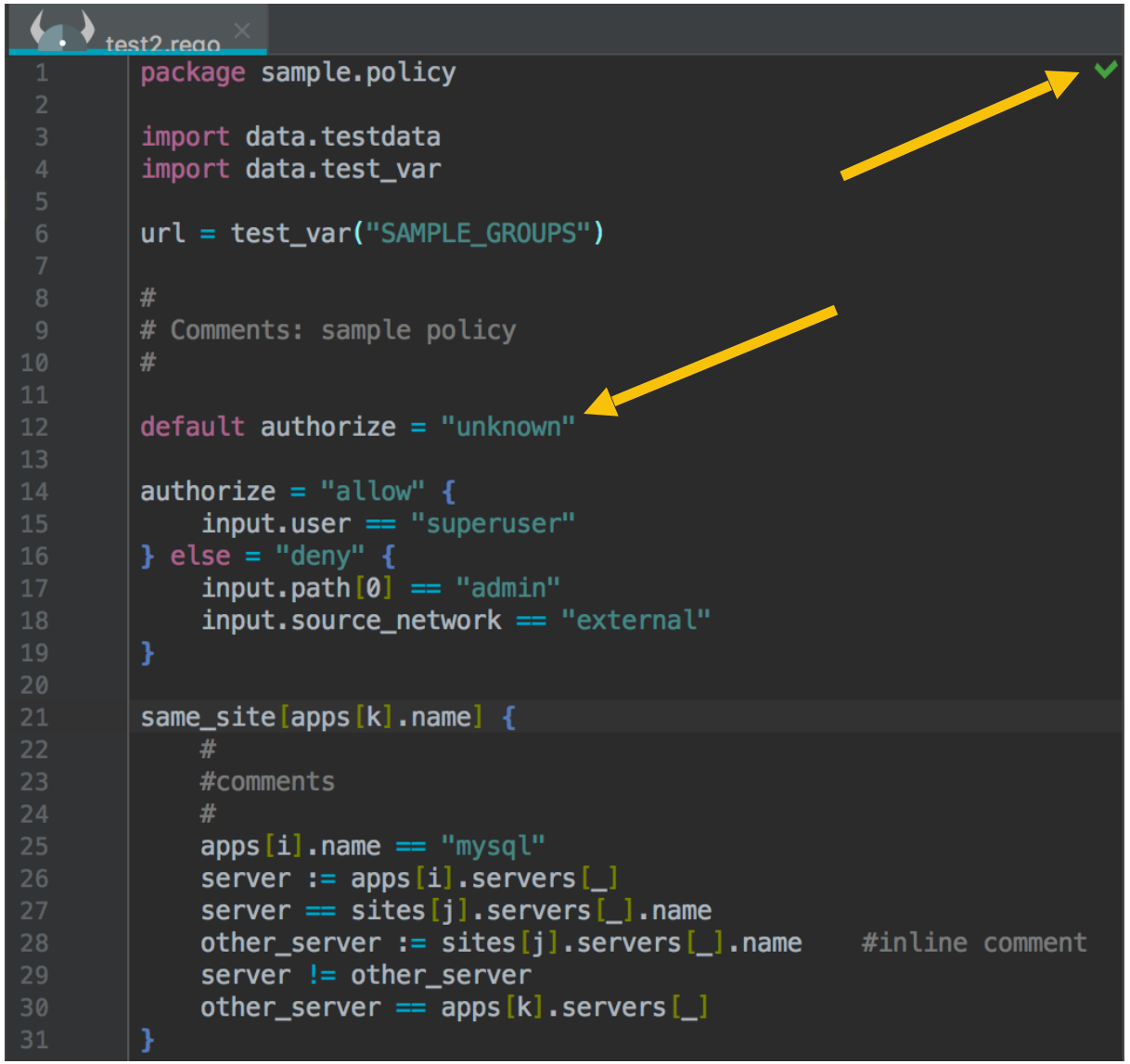




Open Policy Agent: IntelliJ Plugin: Syntax Validation



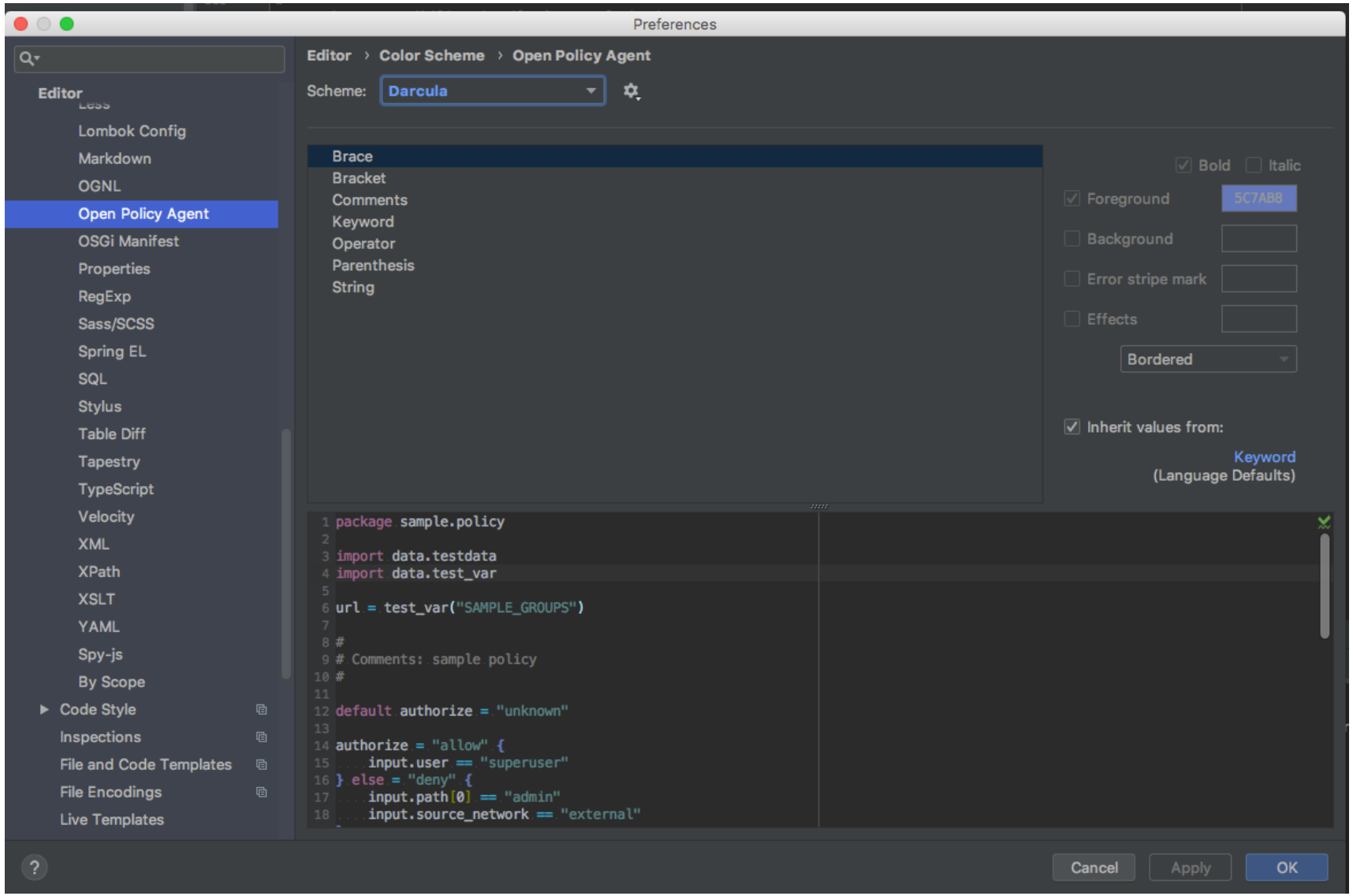
Open Policy Agent: IntelliJ Plugin: Syntax Validation - Continued



Open Policy Agent: IntelliJ Plugin: Settings

Select

- Preferences
  - Editor
    - > Color Scheme
      - Open Policy Agent



Summary

- Responsibility Management as a Service can resolve issues on several fronts.
- Choice of a payload format (HOCON over JSON or XML) can help control verbosity.
- Choice of architecture (federated versus distributed) can help determine resilience.
- Distributed policy engines can alleviate back-pressure and volume demands.
- Distributed policy engines can reduce outages and maintenance-related downtimes.
- Creating a policy editor plugin can help boost productivity.

Links

- HOCON

  https://github.com/lightbend/config/blob/master/HOCON.md

- Eclipse Collections

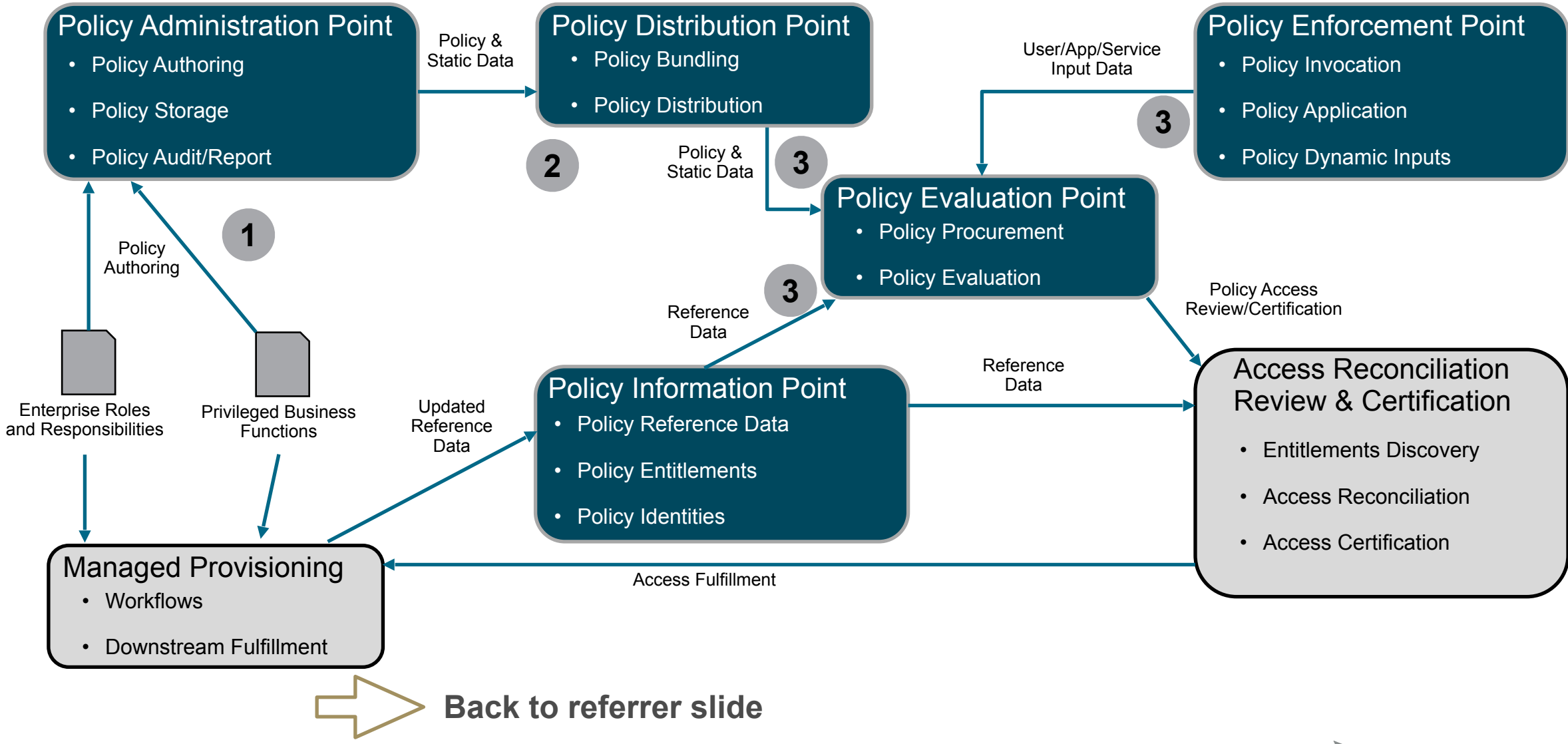
  https://www.eclipse.org/collections/

- Open Policy Agent

  https://www.openpolicyagent.org/


BNY MELLON

Appendix: Responsibility Management Cycle



Back to referrer slide