KubeCon CloudNativeCon
North America 2025

Tools and Strategies for Making the Most of Kubernetes Access Control

Lucas Käldström, Upbound
Micah Hausler, AWS

#KubeCon #CloudNativeCon

Who are we?

KubeCon CloudNativeCon North America 2025

Staff Engineer @ Upbound

Formerly at Weaveworks

Former SIG Cluster Lifecycle co-lead

CNCF Ambassador 2017-2024

Principal Engineer @ AWS

Working on AWS EKS

Kubernetes Security Response Committee

SIG-Auth co-chair

@luxas.dev & @micahhausler.com

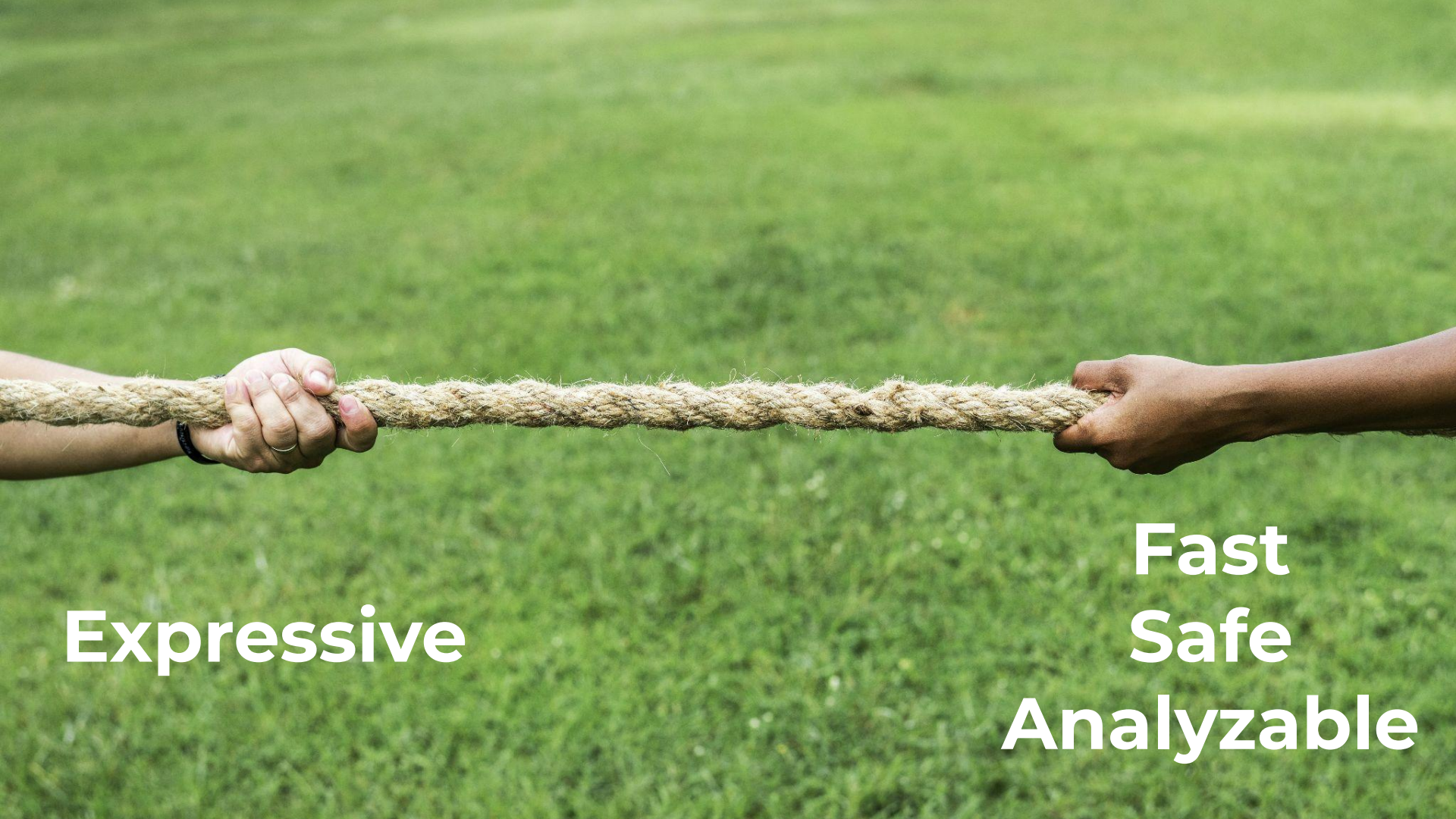
Outline

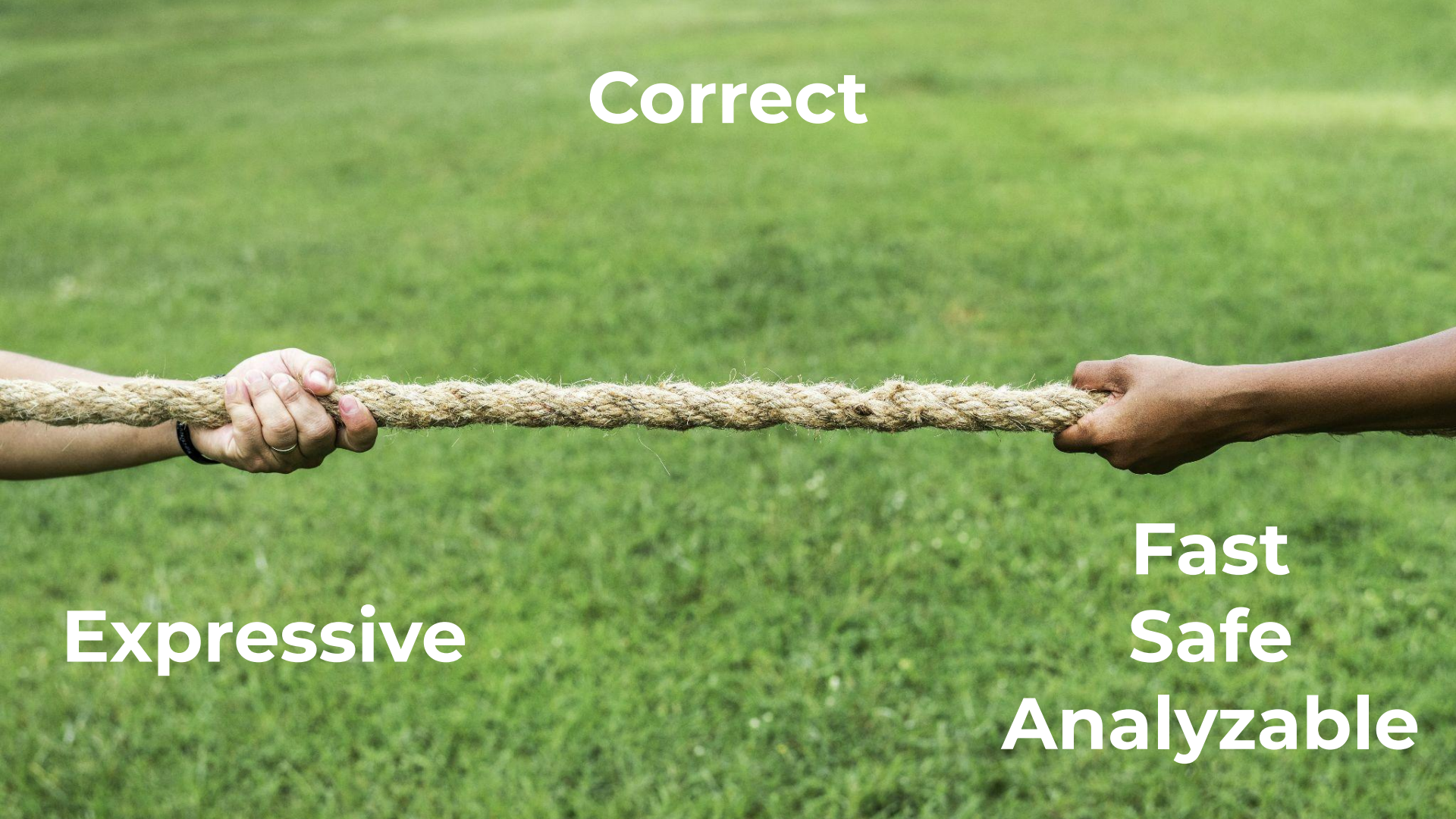
1. Introduction to Kubernetes Authorization
2. Kubernetes RBAC
    a. Auto-discover RBAC rules needed with audit2rbac
    b. Query “what resources do I have access to?” using rakkess
    c. Query “who has access to sensitive resources?” using rakkess and kubectl-who-can
3. Conditional Authorization (Kubernetes Enhancement Proposal)
    a. Cedar Policy
    b. upbound/kubernetes-cedar-authorizer demo!
4. Conclusion: what’s next?

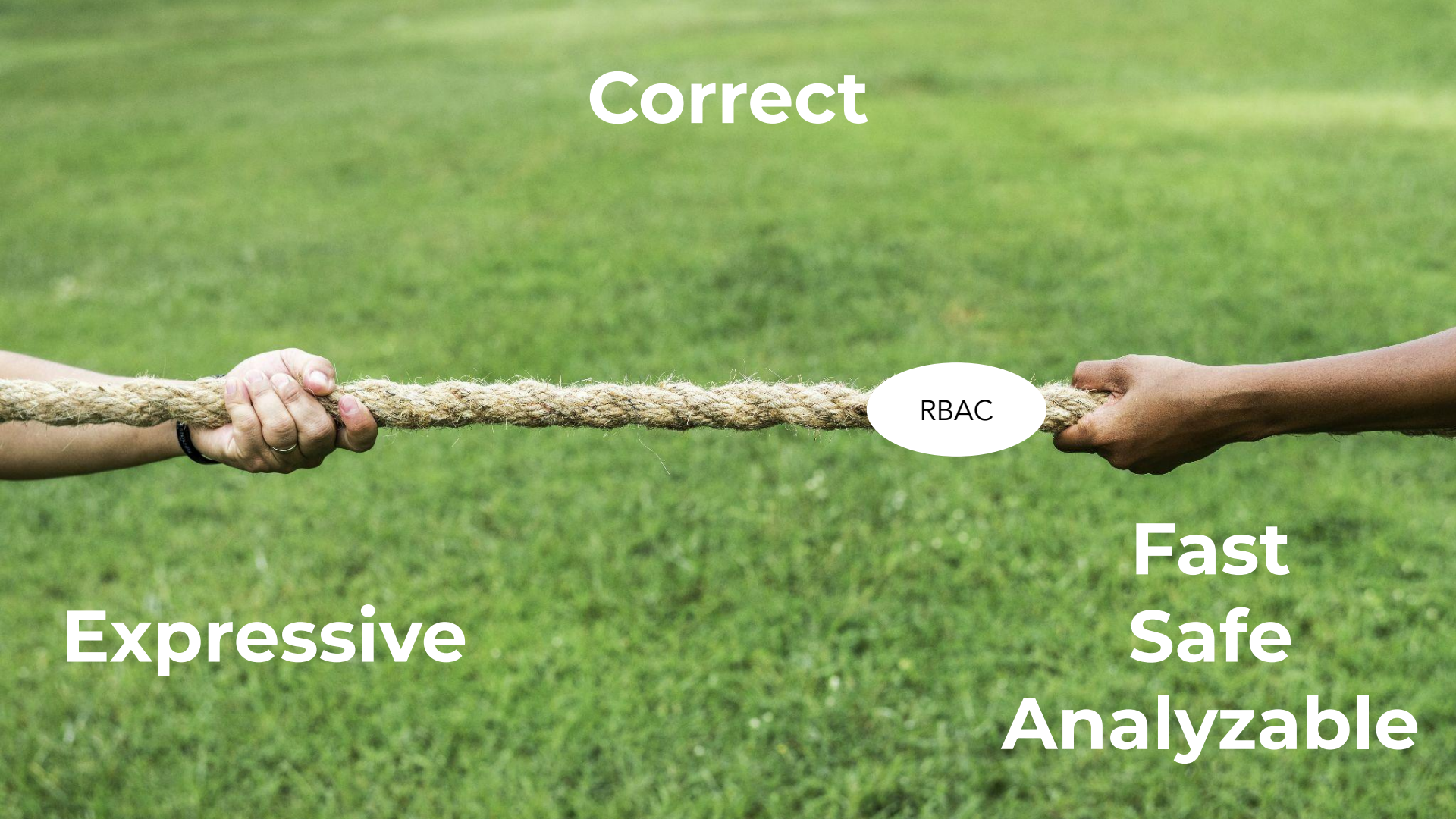

Expressive

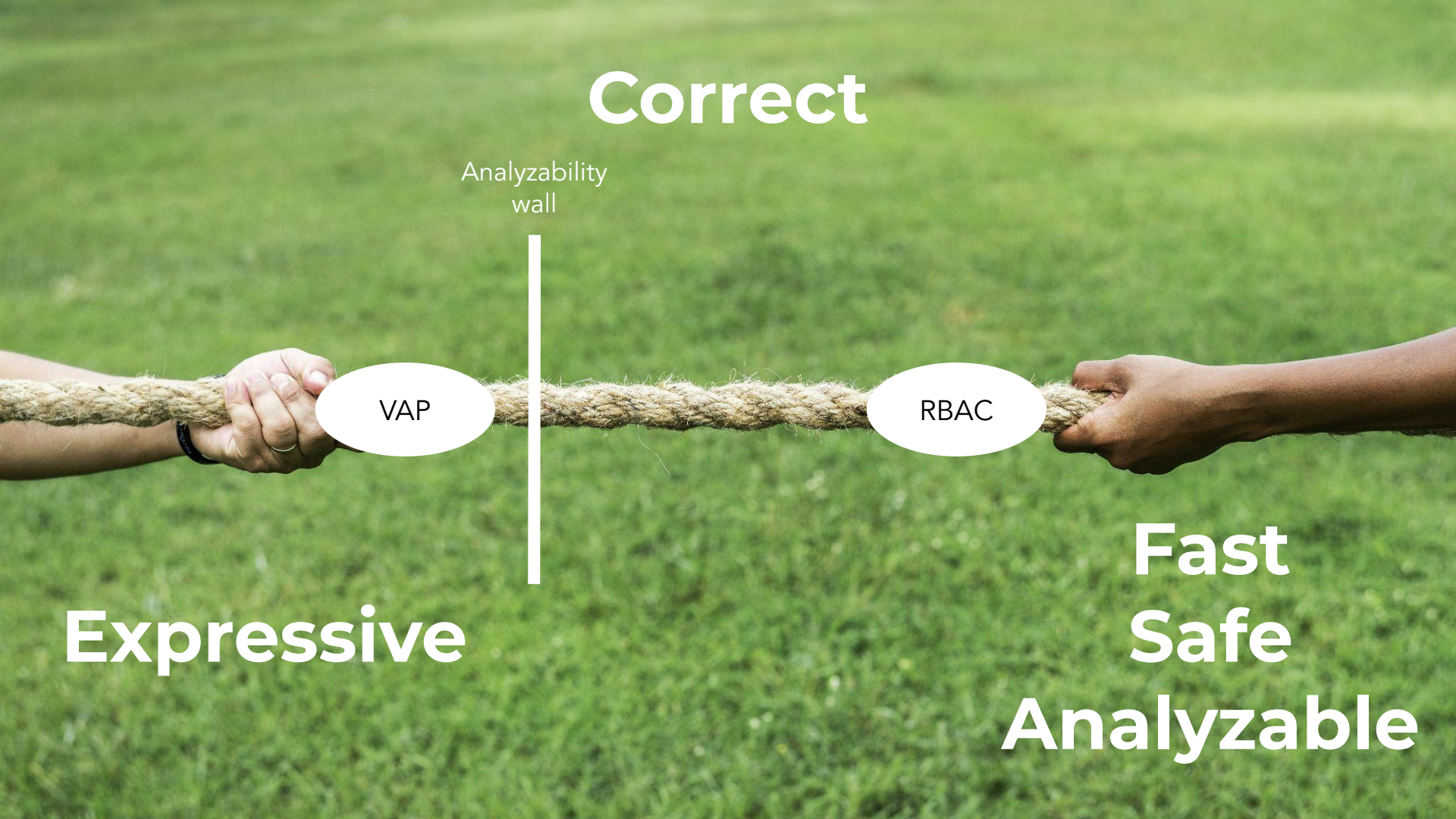

Fast
Expressive

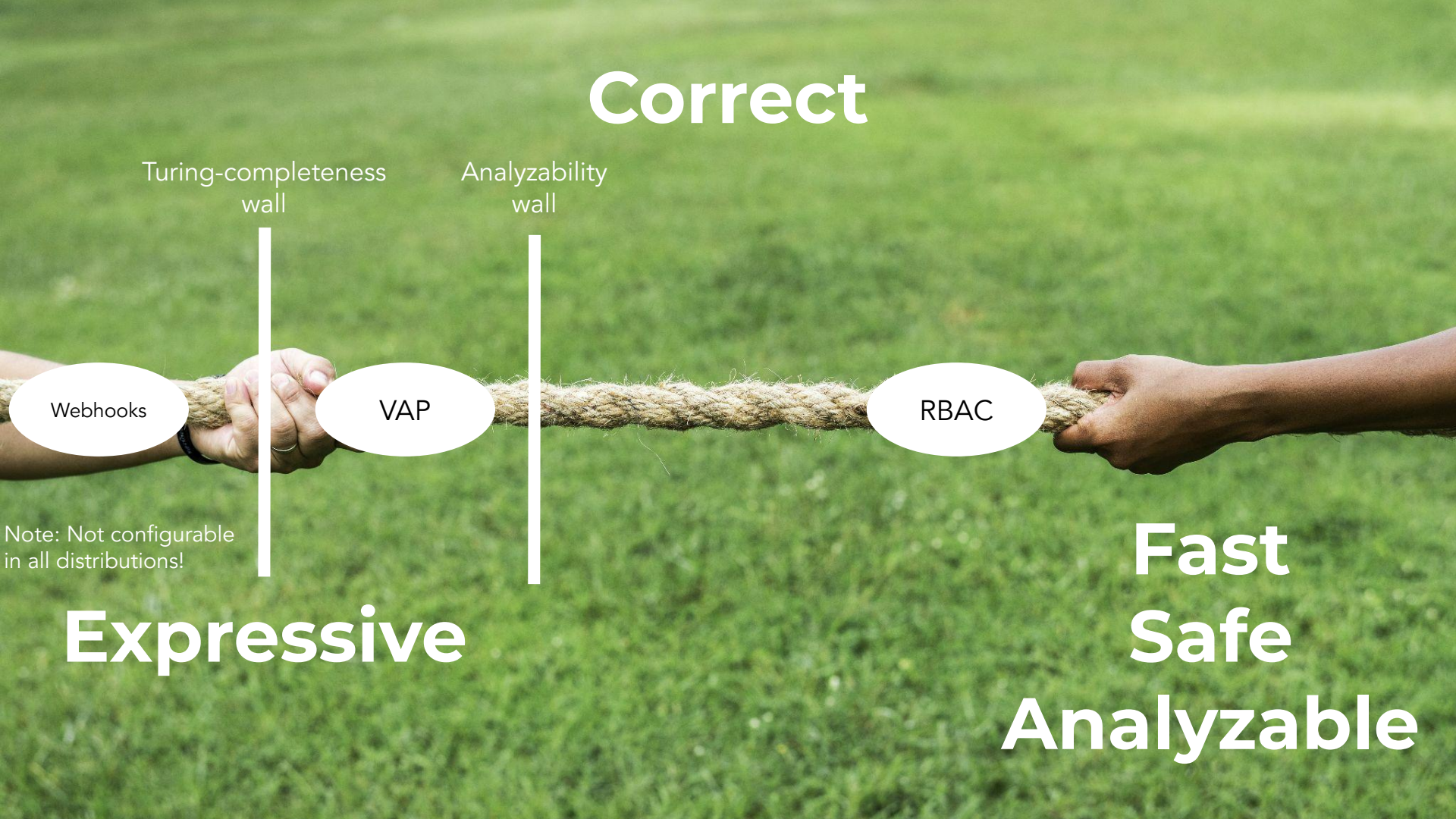

Expressive
Fast
Safe

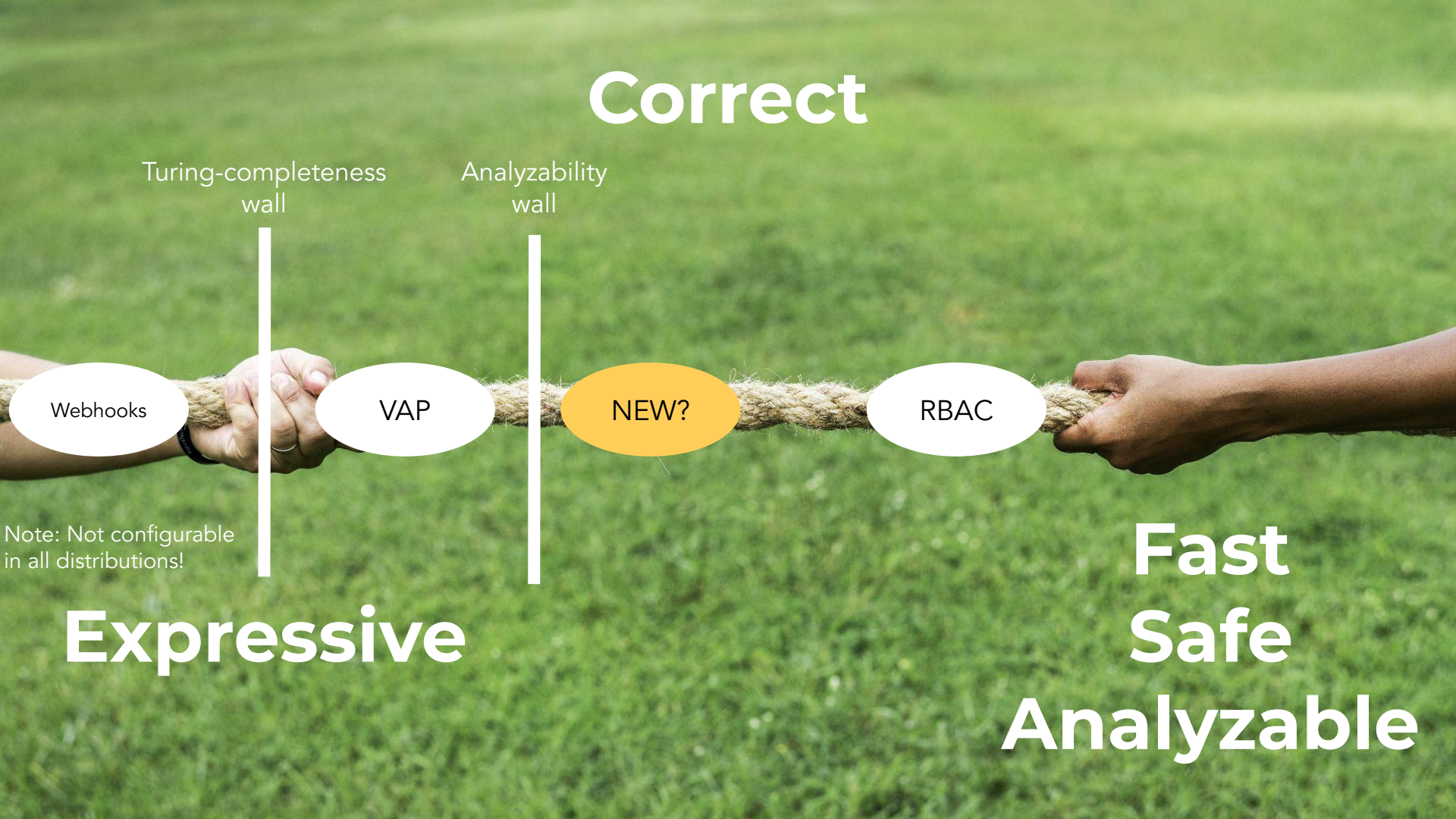

Expressive
Fast
Safe
Analyzable


Correct
Expressive
Fast
Safe
Analyzable

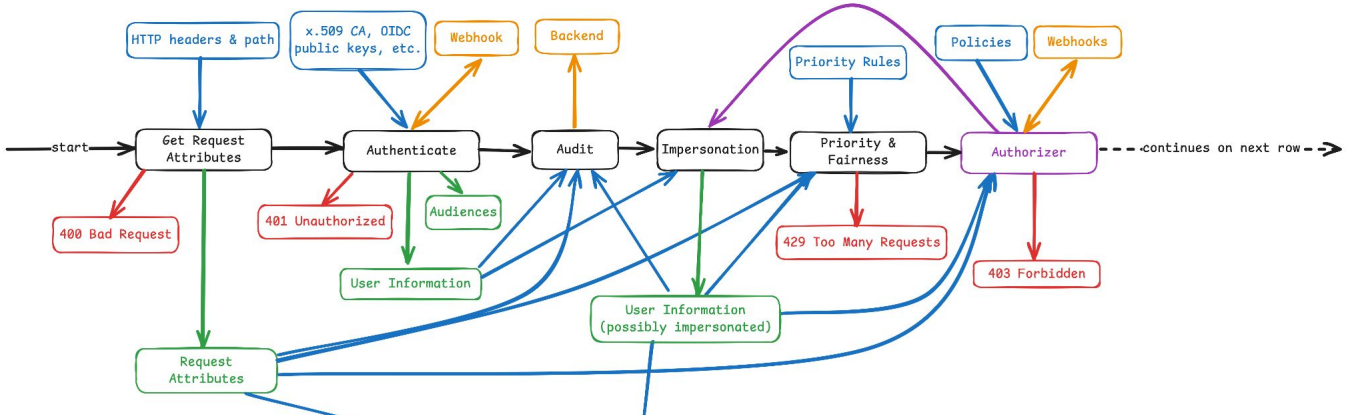

Correct
RBAC
Expressive
Fast
Safe
Analyzable


Correct
Analyzability wall
VAP
RBAC
Expressive
Fast
Safe
Analyzable


Correct
Turing-completeness wall
Analyzability wall
Webhooks
VAP
RBAC
Note: Not configurable in all distributions!
Expressive
Fast
Safe
Analyzable


Correct
Turing-completeness wall
Analyzability wall
Webhooks
VAP
NEW?
RBAC
Note: Not configurable in all distributions!
Expressive
Fast
Safe
Analyzable

Life of an API server request (pt. 1)



@luxas.dev & @micahhausler.com

Image from “Usable Access Control in Cloud Management System” by Lucas Käldström

Life of an API server request (pt. 1)

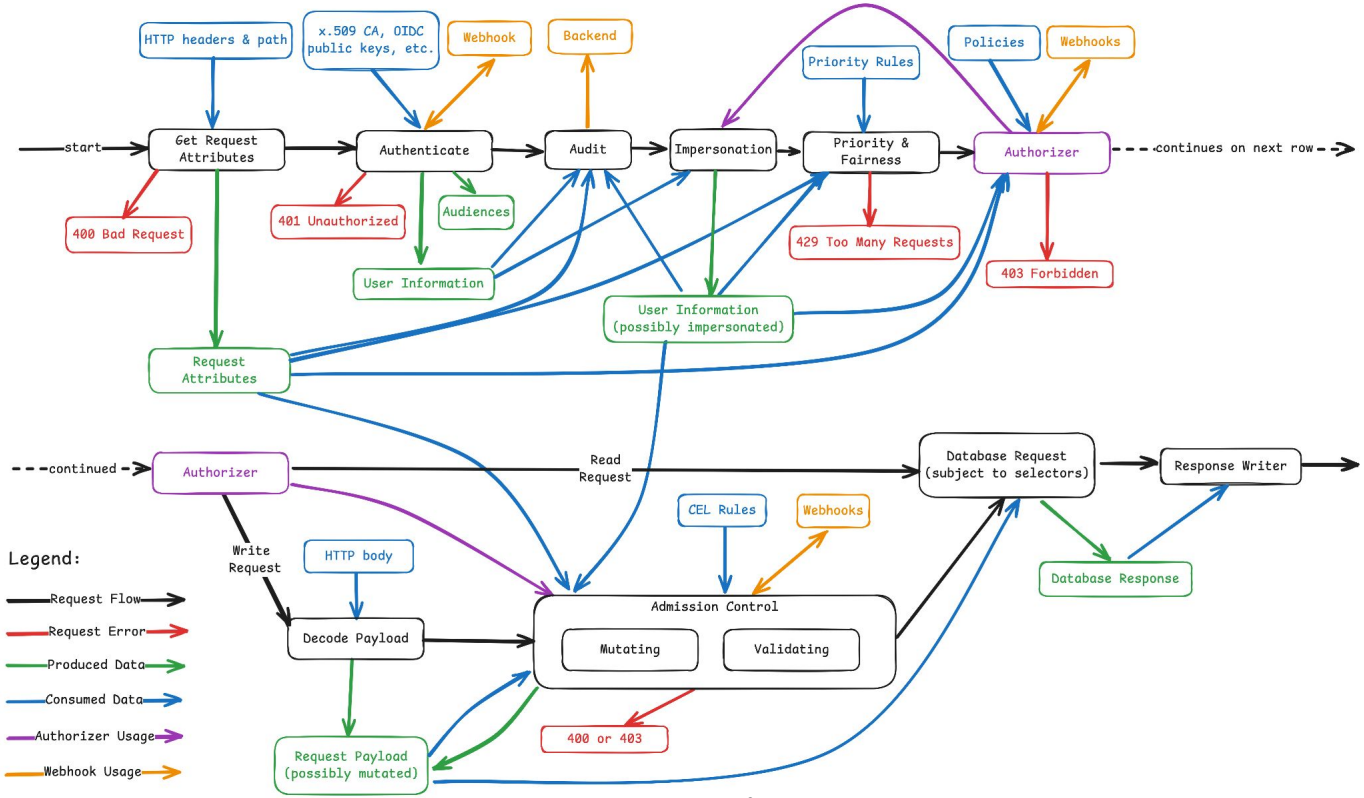
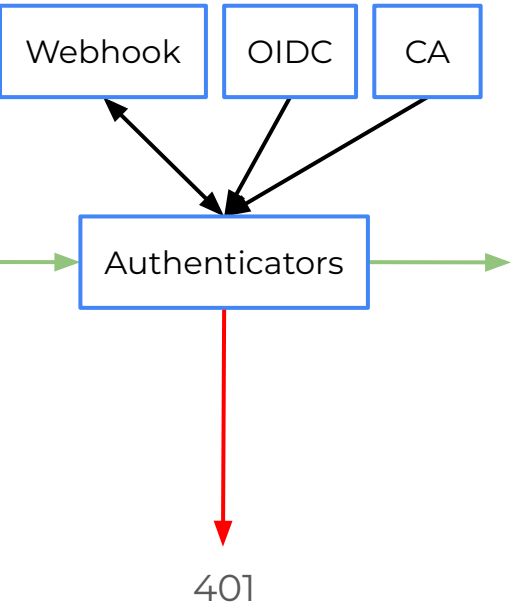
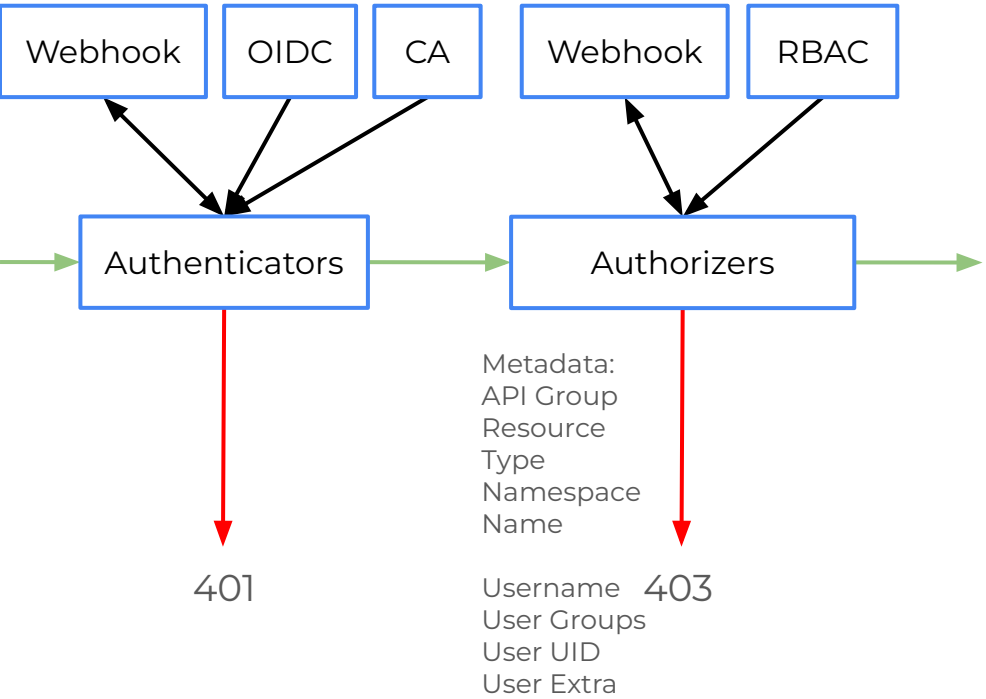


Image from "Usable Access Control in Cloud Management System" by Lucas Käldström

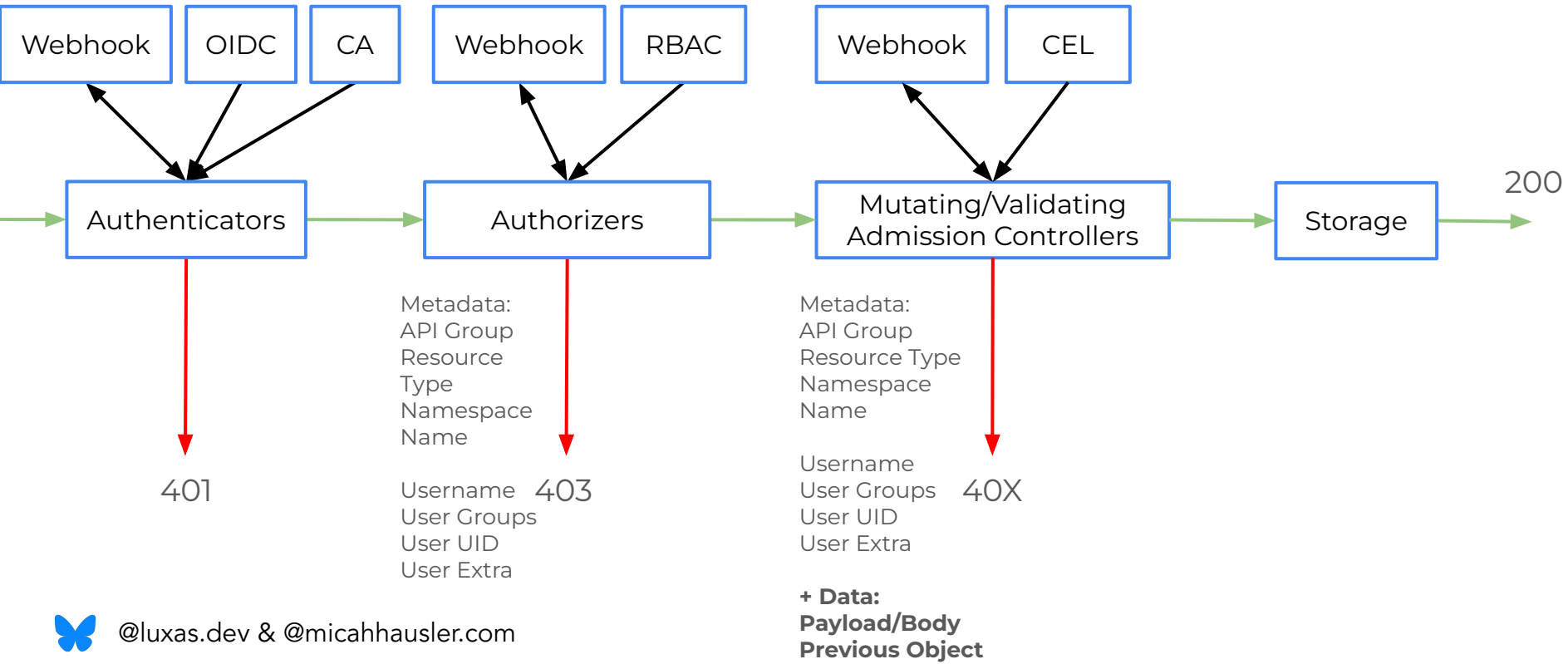
Life of a write request (simplified)



Life of a write request (simplified)

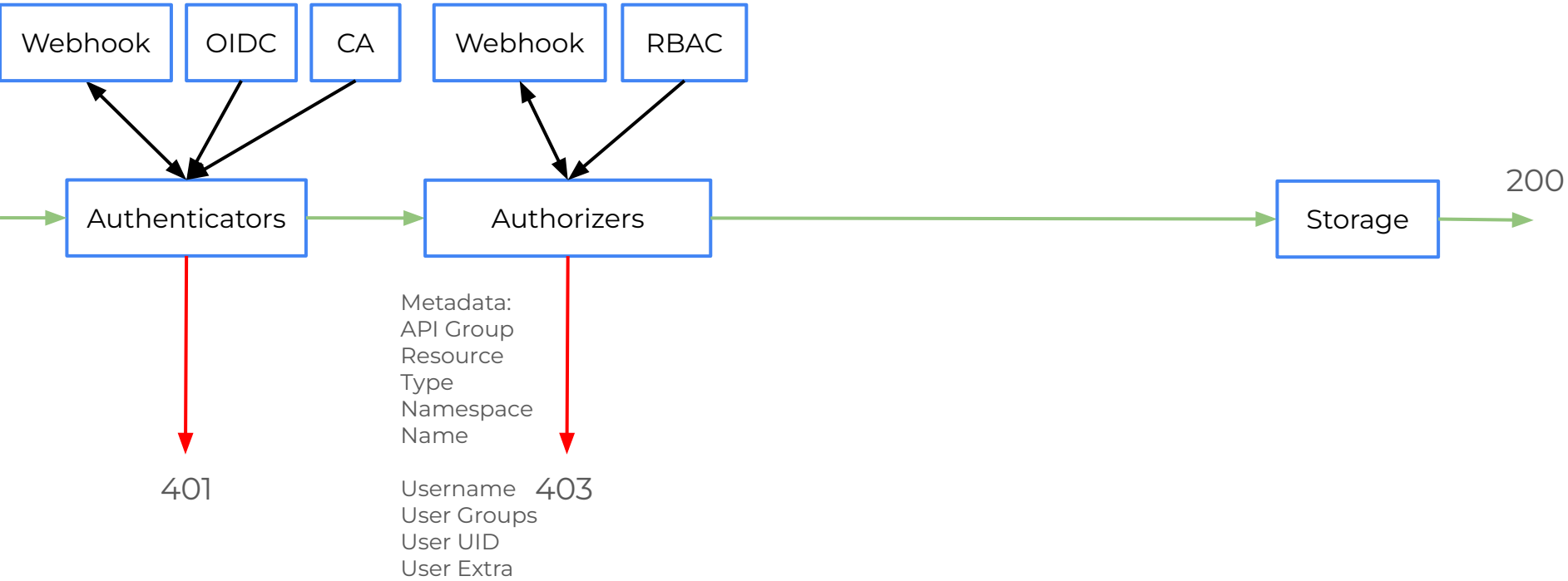


Life of a write request (simplified)



@luxas.dev & @micahhausler.com

Life of a read request (simplified)



Kubernetes RBAC

RBAC = Role-Based Access Control

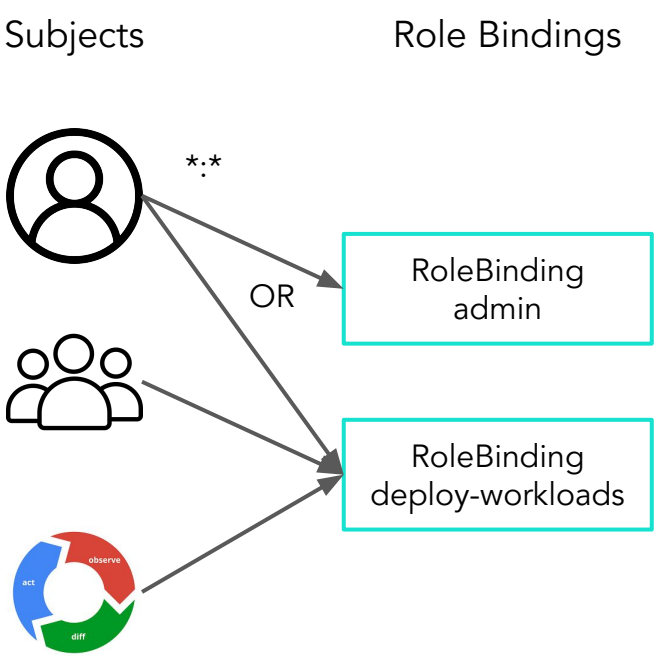
RBAC rules stored in the API server

Privilege escalation prevention

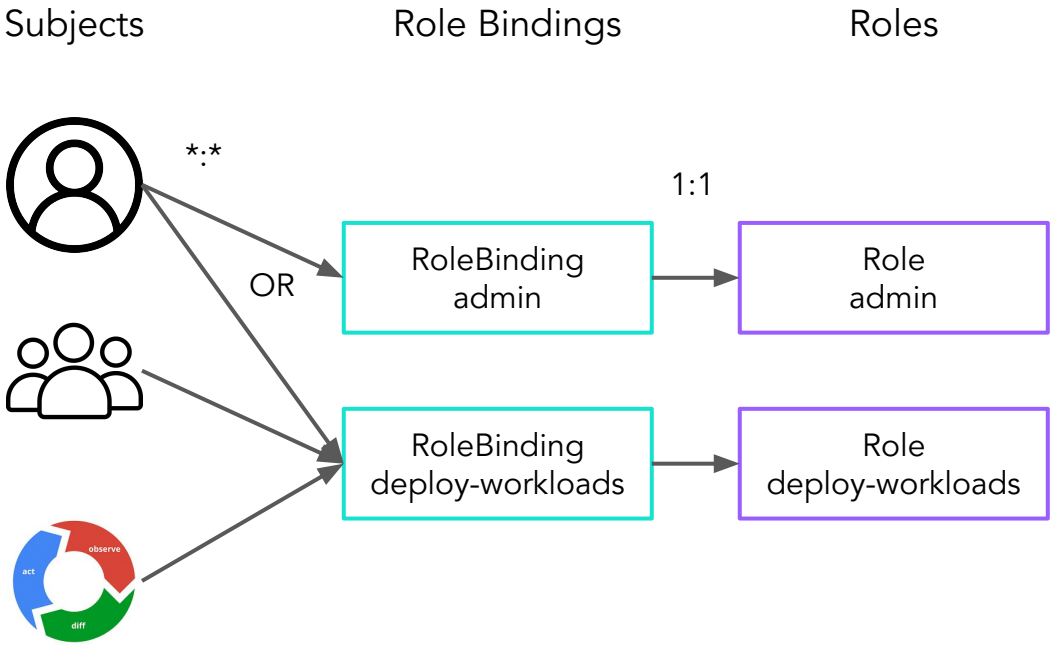
Only Allow roles, no Deny roles

Operates only on request metadata, not objects

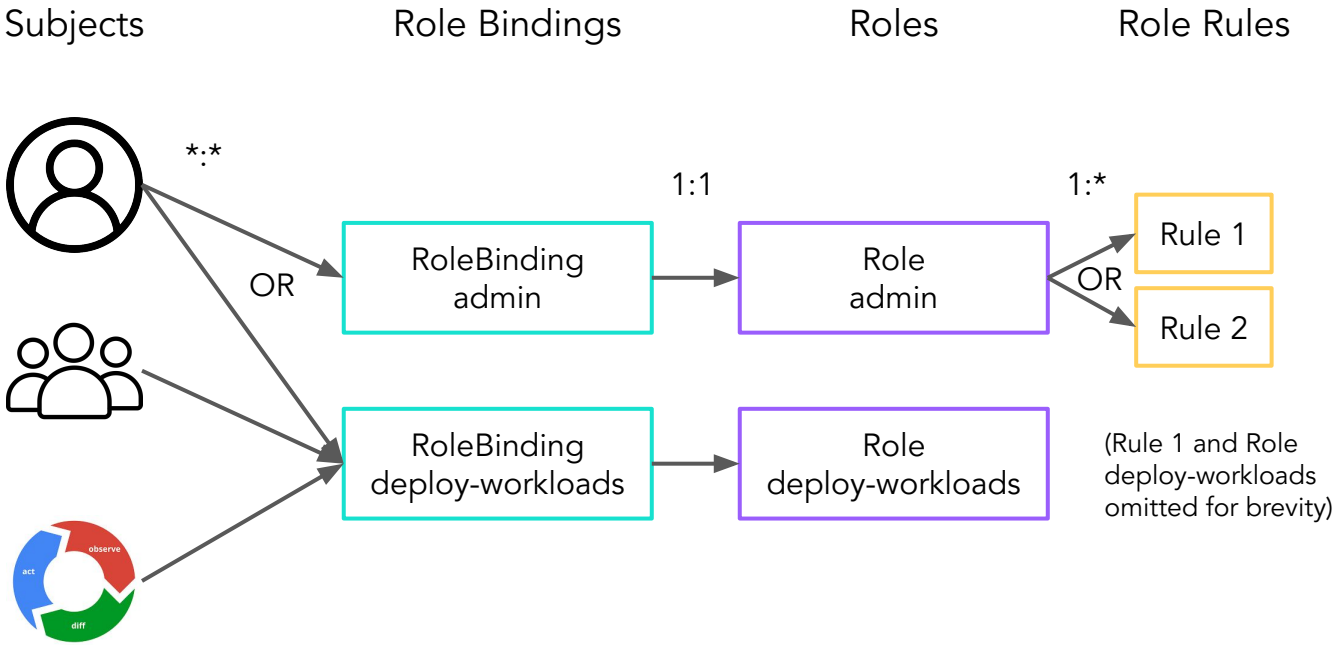
Kubernetes RBAC



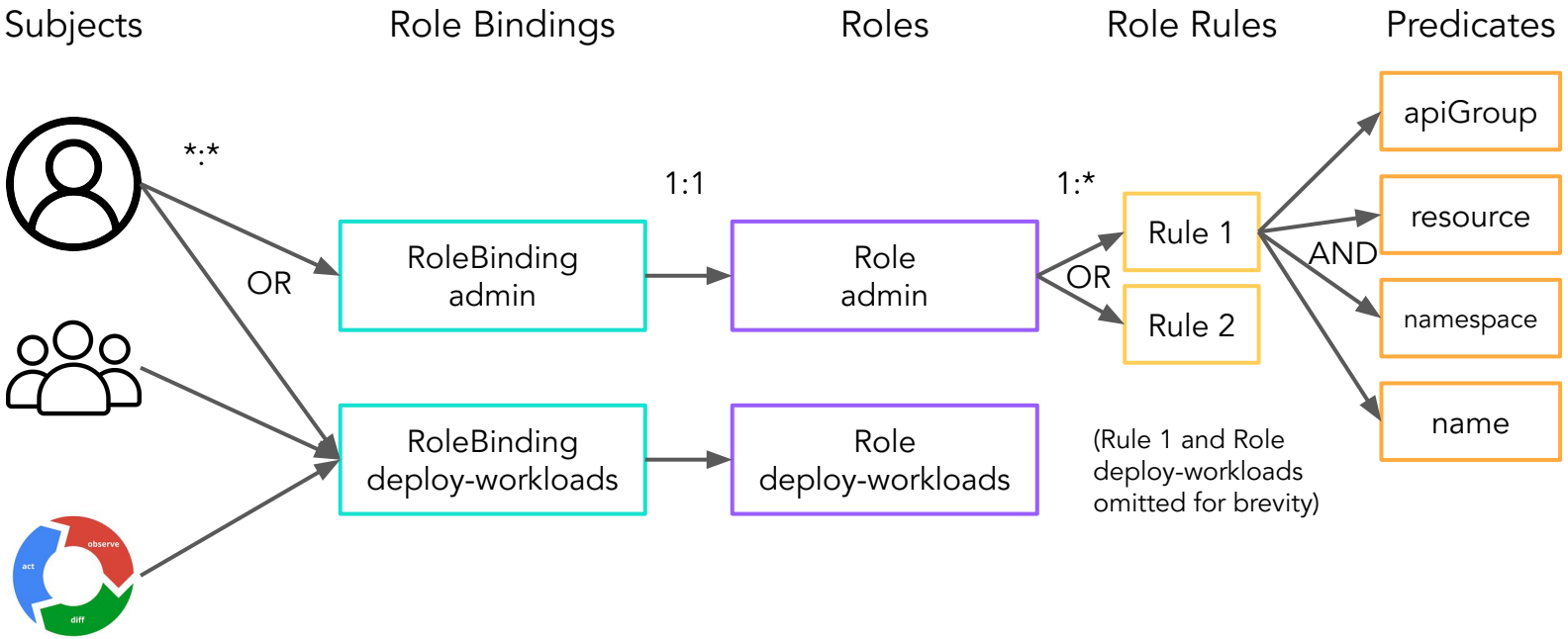
Kubernetes RBAC



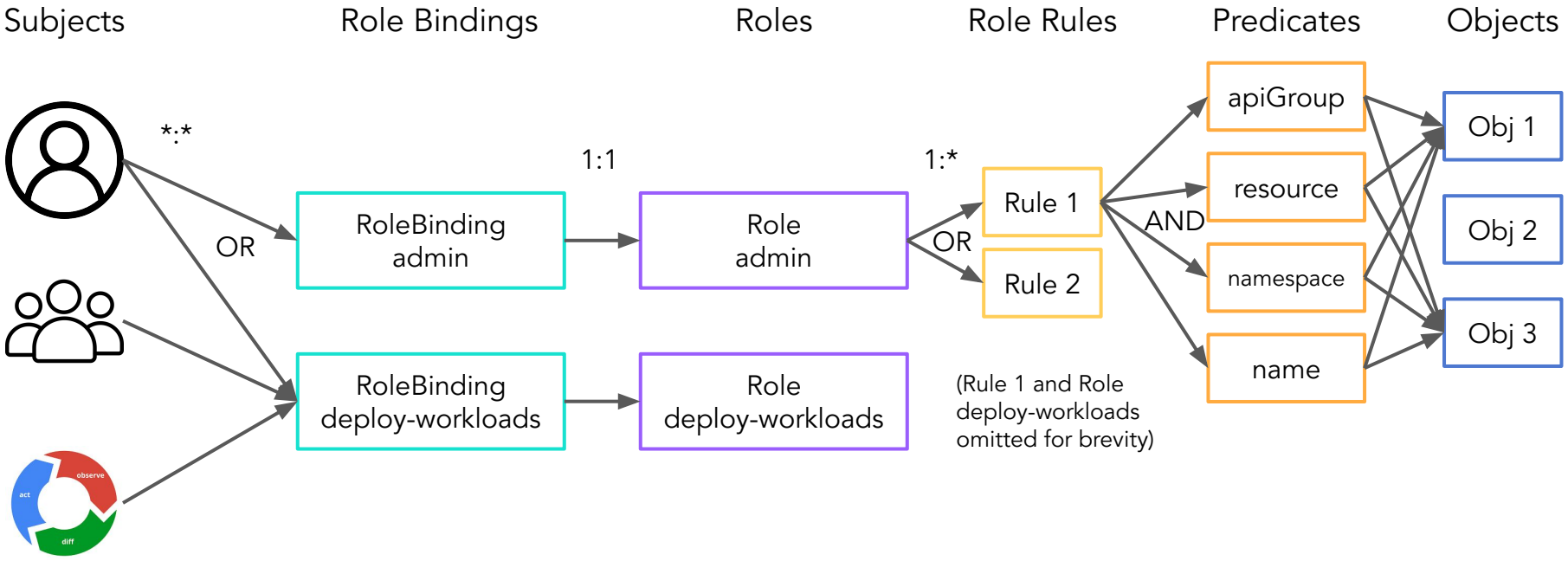
Kubernetes RBAC



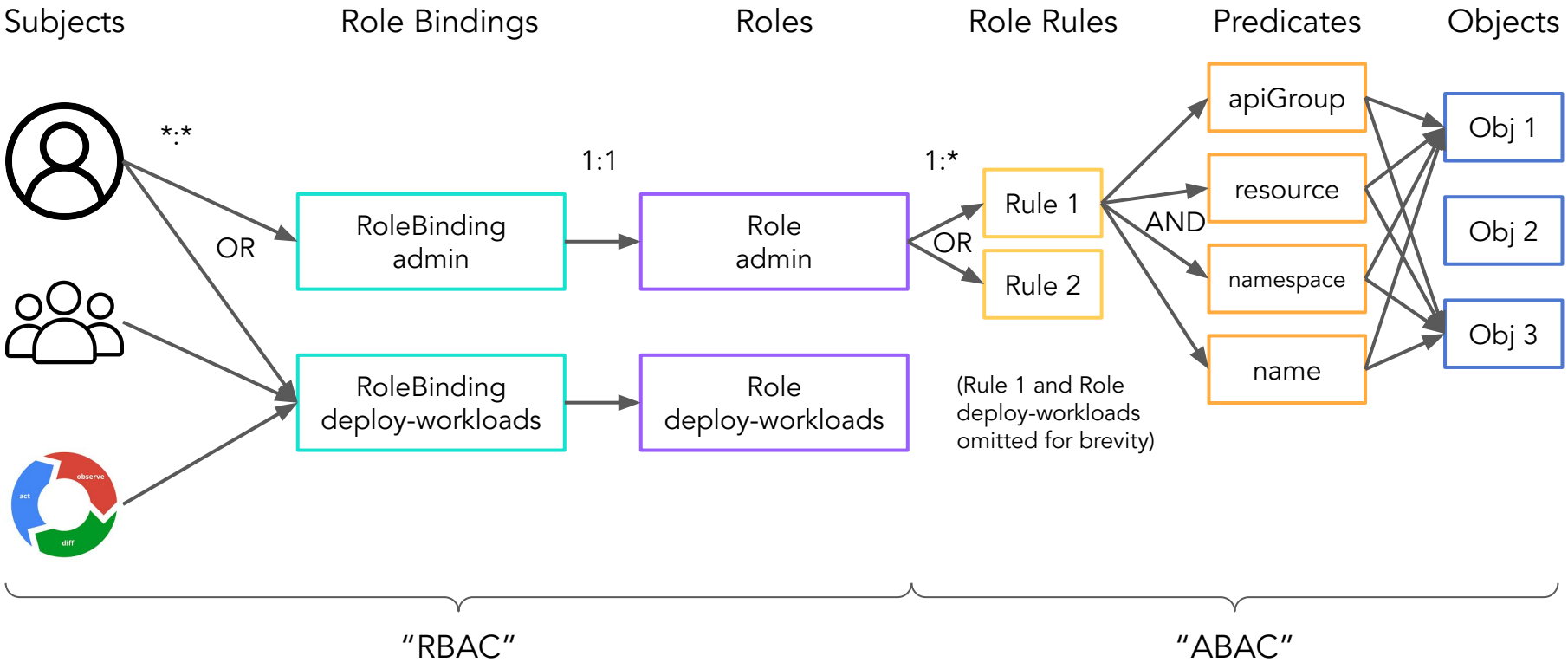
Kubernetes RBAC



Kubernetes RBAC



Kubernetes RBAC



How do I know which RBAC roles I need?

When in doubt, it is tempting to grant too wide permissions

However, the audit log can tell you about unauthorized requests

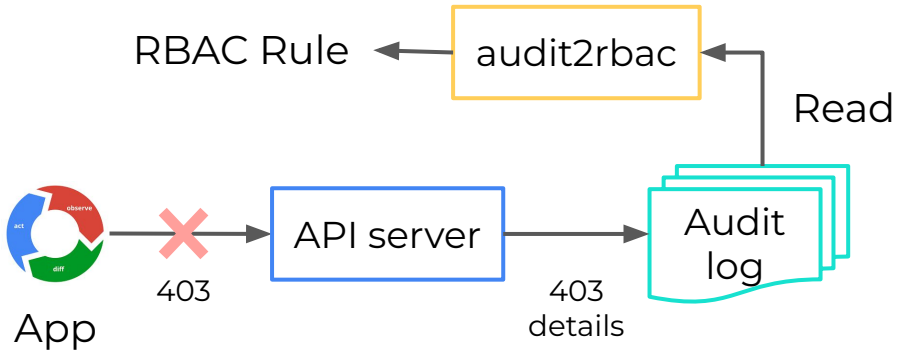
How do I know which RBAC roles I need?

When in doubt, it is tempting to grant too wide permissions

However, the audit log can tell you about unauthorized requests

🙌 Jordan Liggitt's audit2rbac tool can infer the roles needed from the audit log!

Use carefully to not grant permissions to actually unauthorized requests



Loads of RBAC rules... What do I have access to?

Check your own permissions using `kubectl auth can-i <verb> <resource>`

Loads of RBAC rules... What do I have access to?

Check your own permissions using `kubectl auth can-i <verb> <resource>`

However, to get a bigger picture, try out rakkess by Cornelius Weig

Works through the `(Self)SubjectAccessReview` API

Example output for “what can I do in namespace default?”

```
# rakkess --namespace default
NAME                                              LIST  CREATE  UPDATE  DELETE
bindings                                                ✖
configmaps                                        ✔     ✔       ✔       ✔
controllerrevisions.apps                          ✔     ✖       ✖       ✖
cronjobs.batch                                    ✔     ✔       ✔       ✔
daemonsets.apps                                   ✔     ✔       ✔       ✔
daemonsets.extensions                             ✔     ✔       ✔       ✔
deployments.apps                                  ✔     ✔       ✔       ✔
deployments.extensions                            ✔     ✔       ✔       ✔
endpoints                                         ✔     ✔       ✔       ✔
events                                            ✔     ✖       ✖       ✖
events.events.k8s.io                              ✖     ✖       ✖       ✖
horizontalpodautoscalers.autoscaling              ✔     ✔       ✔       ✔
ingresses.extensions                              ✔     ✔       ✔       ✔
jobs.batch                                        ✔     ✔       ✔       ✔
leases.coordination.k8s.io                        ✖     ✖       ✖       ✖
limitranges                                       ✔     ✖       ✖       ✖
localsubjectaccessreviews.authorization.k8s.io          ✖
networkpolicies.extensions                        ✔     ✔       ✔       ✔
```

Image Credit: rakkess

Who has access to <insert sensitive resource>?

A cluster administrator must ensure that sensitive resources are properly protected.

Who has access to <insert sensitive resource>?

A cluster administrator must ensure that sensitive resources are properly protected.

In order to audit who (according to RBAC!) has access to what, rakkess or kubectl-who-can by Liz Rice/Aqua Security can be of help:

```
bash-3.2$ ./kubectl-who-can create pods
No subjects found with permissions to create pods assigned through RoleBindings
CLUSTERROLEBINDING                                SUBJECT                                TYPE            NAMESPACE
cluster-admin                                     system:masters                         Group
docker-for-desktop-binding                        system:serviceaccounts                 Group
system:controller:clusterrole-aggregation-controller  clusterrole-aggregation-controller  ServiceAccount  kube-system
system:controller:daemon-set-controller           daemon-set-controller                  ServiceAccount  kube-system
system:controller:job-controller                  job-controller                         ServiceAccount  kube-system
system:controller:persistent-volume-binder        persistent-volume-binder               ServiceAccount  kube-system
system:controller:replicaset-controller           replicaset-controller                  ServiceAccount  kube-system
system:controller:replication-controller          replication-controller                 ServiceAccount  kube-system
system:controller:statefulset-controller          statefulset-controller                 ServiceAccount  kube-system
```


Image Credit: kubectl-who-can demo

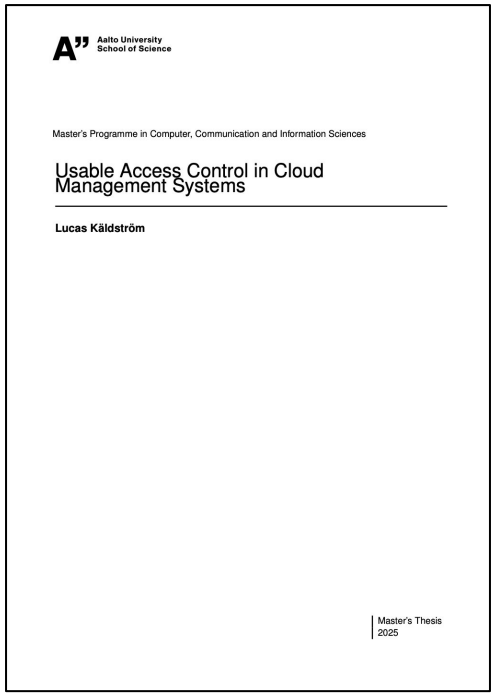
Conditional Authorization

Brand-new Kubernetes
Enhancement Proposal (KEP)

Allow for more granularity

Unified UX between
authorization and admission

MSc thesis: Usable Access Control in Cloud Management Systems



Aalto University
School of Science

Master's Programme in Computer, Communication and Information Sciences

Usable Access Control in Cloud Management Systems

Lucas Käldström

Master's Thesis
2025

@luxas.dev & @micahhausler.com

Open access at: luxas/research

Use-cases

- Allow a principal to only read Secrets with a given label

Use-cases

- Allow a principal to only read Secrets with a given label
- Allow a principal to update an object only when a sensitive field is unchanged

Use-cases

- Allow a principal to only read Secrets with a given label
- Allow a principal to update an object only when a sensitive field is unchanged
- Allow a principal to create objects only with certain names

Use-cases

- Allow a principal to only read Secrets with a given label
- Allow a principal to update an object only when a sensitive field is unchanged
- Allow a principal to create objects only with certain names
- Allow a node agent to only access objects referring to them

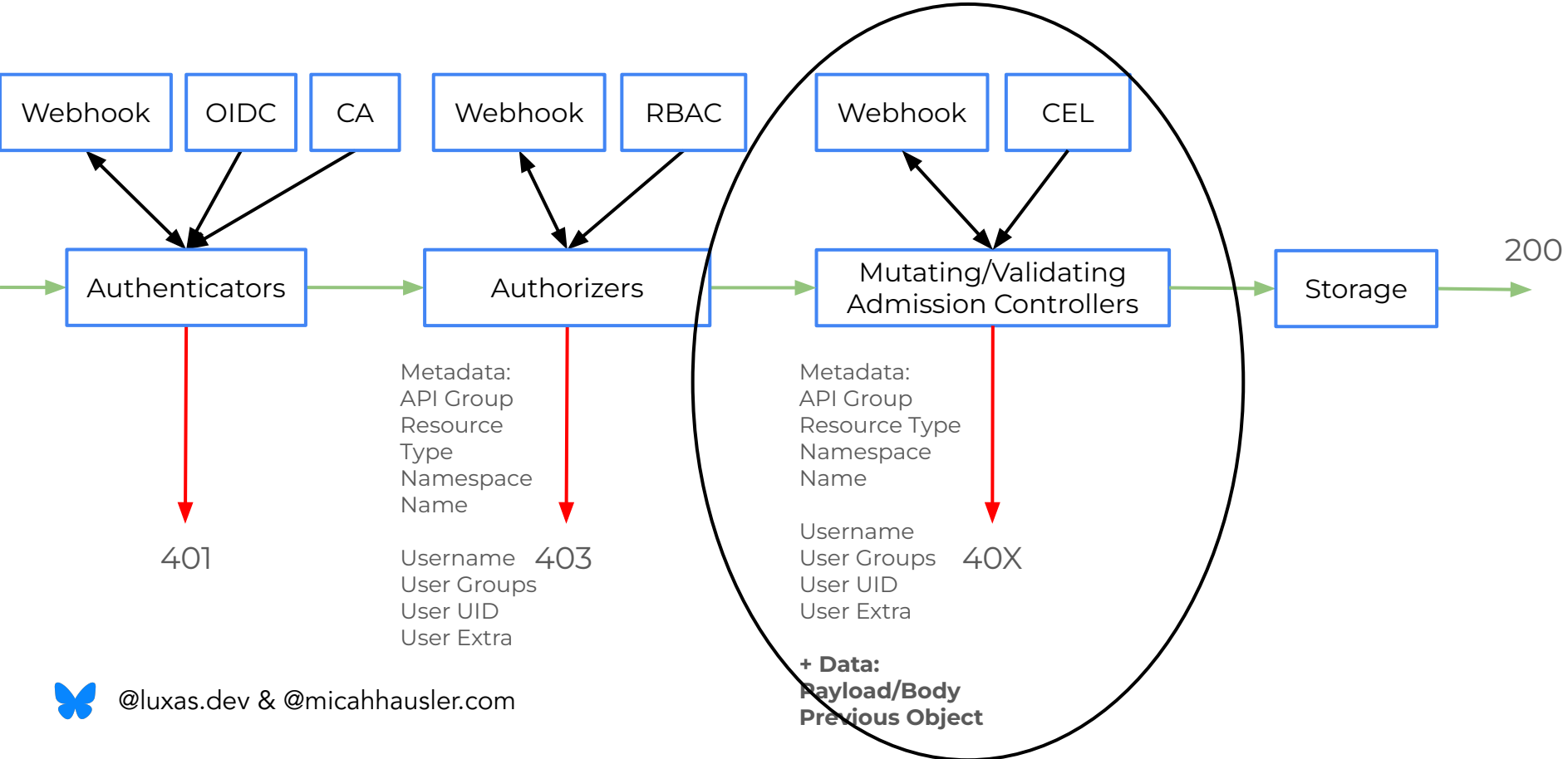
Use-cases

- Allow a principal to only read Secrets with a given label
- Allow a principal to update an object only when a sensitive field is unchanged
- Allow a principal to create objects only with certain names
- Allow a node agent to only access objects referring to them
- Allow a controller to only add/remove its own finalizer

Use-cases

- Allow a principal to only read Secrets with a given label
- Allow a principal to update an object only when a sensitive field is unchanged
- Allow a principal to create objects only with certain names
- Allow a node agent to only access objects referring to them
- Allow a controller to only add/remove its own finalizer
- Deny everyone except admins to use GPU adminAccess

Recall admission control for write requests



@luxas.dev & @micahhausler.com

Validating Admission Control

For a request to succeed, all admission controllers must consider the object “valid”

Admission control never gives permissions, but can only remove (deny policy)

Webhook

```yaml
apiVersion: admission.k8s.io/v1
kind: AdmissionReview
request:
  operation: CREATE
  name: foo-pod
  object:
    kind: Pod
    ...
response:
  allowed: true
```

CEL

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
  name: "allow-only-test-gateway"
spec:
  matchConstraints:
    resourceRules:
    - apiGroups:   ["gateway.networking.k8s.io"]
      apiVersions: ["v1"]
      operations:  ["CREATE", "UPDATE", "DELETE"]
      resources:   ["gateways"]
  validations:
  - expression: |
      (request.userInfo.username == 'lucas' && object != null) ?
        object.spec.gatewayClassName == 'test-gateway' :
        true
  - expression: |
      (request.userInfo.username == 'lucas' && oldObject != null) ?
        oldObject.spec.gatewayClassName == 'test-gateway' :
        true
```

Current problem: “Over-grant” in RBAC, deny in VAP

Amount of permissions for lucas

create, update, delete gateways
—
object=*
oldobject=*

.class != ‘test-gateway’

.class == ‘test-gateway’

Desired permissions

Kubernetes RBAC

CEL Rule

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: lucas-gateway-writer
subjects:
- kind: User
  name: lucas
roleRef:
  kind: ClusterRole
  name: gateway-writer
  apiGroup: rbac.authorization.k8s.io
```

RBAC Role Binding
Allow in authorization

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: gateway-writer
rules:
- apiGroups: ["gateway.networking.k8s.io"]
  resources: ["gateways"]
  verbs: ["create", "update", "delete"]
```

RBAC Role
Allow in authorization

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
  name: "allow-only-test-gateway"
spec:
  matchConstraints:
    resourceRules:
    - apiGroups:   ["gateway.networking.k8s.io"]
      apiVersions: ["v1"]
      operations:  ["CREATE", "UPDATE", "DELETE"]
      resources:   ["gateways"]
  validations:
  - expression: |
      (request.userInfo.username == 'lucas' && object != null) ?
        object.spec.gatewayClassName == 'test-gateway' :
        true
  - expression: |
      (request.userInfo.username == 'lucas' && oldObject != null) ?
        oldObject.spec.gatewayClassName == 'test-gateway' :
        true
```

CEL Policy
Deny in admission

Unified policies through Partial Evaluation

What if it was possible to declare in one place that “lucas can create gateways when class=’test-gateway’”?

Unified policies through Partial Evaluation

What if it was possible to declare in one place that “lucas can create gateways when class=’test-gateway’”?

In pseudo-code, it could look something like:

```
“request.verb == ‘create’ &&
  request.apiGroup == ‘gateway.networking.k8s.io’ &&
  request.resource == ‘gateways’ &&
  request.userInfo.username == “lucas” &&
  has(request.object.v1) &&
  request.object.v1.spec.gatewayClassName == ‘test-gateway’”
```

Unified policies through Partial Evaluation

One trick is to use **partial evaluation**. Mark request.object as “unknown”, but evaluate the expression as much as possible.

If lucas does a `kubectl create -f gateway.yaml`, the following happens:

```
“request.verb == ‘create’ &&
  request.apiGroup == ‘gateway.networking.k8s.io’ &&
  request.resource == ‘gateways’ &&
  request.userInfo.username == “lucas” &&
  has(request.object.v1) &&
  request.object.v1.spec.gatewayClassName == ‘test-gateway’”
```

Unified policies through Partial Evaluation

KubeCon CloudNativeCon North America 2025

One trick is to use **partial evaluation**. Mark request.object as “unknown”, but evaluate the expression as much as possible.

If lucas does a `kubectl create -f gateway.yaml`, the following happens:

```
“true &&
  request.apiGroup == ‘gateway.networking.k8s.io’ &&
  request.resource == ‘gateways’ &&
  request.userInfo.username == “lucas” &&
  has(request.object.v1) &&
  request.object.v1.spec.gatewayClassName == ‘test-gateway’”
```

Unified policies through Partial Evaluation

One trick is to use partial evaluation. Mark request.object as “unknown”, but evaluate the expression as much as possible.

If lucas does a `kubectl create -f gateway.yaml`, the following happens:

```
“true &&
  true &&
  request.resource == ‘gateways’ &&
  request.userInfo.username == “lucas” &&
  has(request.object.v1) &&
  request.object.v1.spec.gatewayClassName == ‘test-gateway’”
```

Unified policies through Partial Evaluation

One trick is to use partial evaluation. Mark request.object as “unknown”, but evaluate the expression as much as possible.

If lucas does a `kubectl create -f gateway.yaml`, the following happens:

```
“true &&
  true &&
  true &&
  request.userInfo.username == “lucas” &&
  has(request.object.v1) &&
  request.object.v1.spec.gatewayClassName == ‘test-gateway’”
```

Unified policies through Partial Evaluation

One trick is to use **partial evaluation**. Mark request.object as “unknown”, but evaluate the expression as much as possible.

If lucas does a `kubectl create -f gateway.yaml`, the following happens:

```
“true &&
  true &&
  true &&
  true &&
  has(request.object.v1) &&
  request.object.v1.spec.gatewayClassName == ‘test-gateway’”
```

Unified policies through Partial Evaluation

One trick is to use **partial evaluation**. Mark request.object as “unknown”, but evaluate the expression as much as possible.

If lucas does a `kubectl create -f gateway.yaml`, the following happens:

```
“true &&
  true &&
  true &&
  true &&
  true &&
  request.object.v1.spec.gatewayClassName == ‘test-gateway’”
```

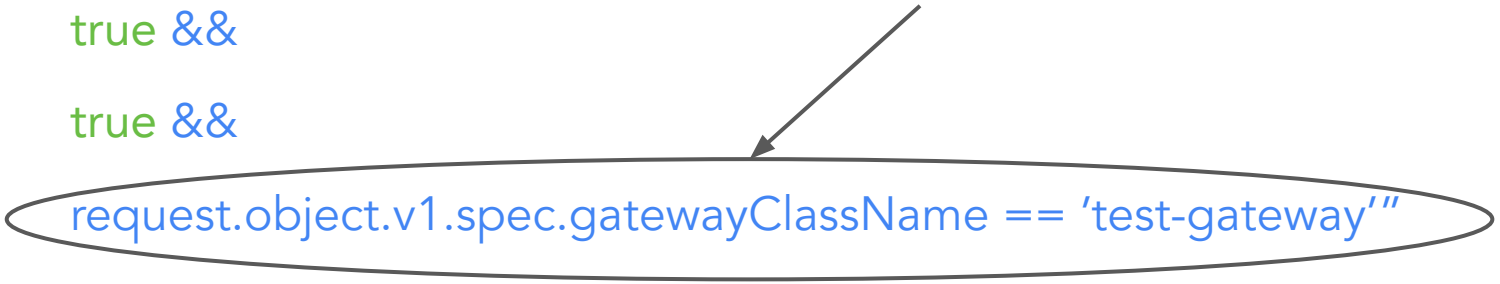
Unified policies through Partial Evaluation

One trick is to use **partial evaluation**. Mark request.object as “unknown”, but evaluate the expression as much as possible.

If lucas does a `kubectl create -f gateway.yaml`, the following happens:

```
“true &&
  true &&
  true &&
  true &&
  true &&
  request.object.v1.spec.gatewayClassName == ‘test-gateway’”
```

Condition on the request object,
defer evaluation to when decoded!



Unified policies through Partial Evaluation

Note that even though the object is unknown, in some cases partial evaluation can evaluate to a concrete true or false:

For example, if user *hacker* executes `kubectl create -f gateway.yaml`

```
"request.verb == 'create' &&
  request.apiGroup == 'gateway.networking.k8s.io' &&
  request.resource == 'gateways' &&
  request.userInfo.username == "lucas" &&
  has(request.object.v1) &&
  request.object.v1.spec.gatewayClassName == 'test-gateway'"
```

Unified policies through Partial Evaluation

Note that even though the object is unknown, in some cases partial evaluation can evaluate to a concrete true or false:

For example, if user *hacker* executes `kubectl create -f gateway.yaml`

```
"true &&
  request.apiGroup == 'gateway.networking.k8s.io' &&
  request.resource == 'gateways' &&
  request.userInfo.username == "lucas" &&
  has(request.object.v1) &&
  request.object.v1.spec.gatewayClassName == 'test-gateway'"
```

Unified policies through Partial Evaluation

Note that even though the object is unknown, in some cases partial evaluation can evaluate to a concrete true or false:

For example, if user *hacker* executes `kubectl create -f gateway.yaml`

```
"true &&
  true &&
  request.resource == 'gateways' &&
  request.userInfo.username == "lucas" &&
  has(request.object.v1) &&
  request.object.v1.spec.gatewayClassName == 'test-gateway'"
```

Unified policies through Partial Evaluation

Note that even though the object is unknown, in some cases partial evaluation can evaluate to a concrete true or false:

For example, if user *hacker* executes `kubectl create -f gateway.yaml`

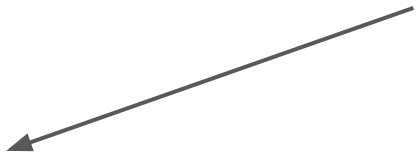
```
"true &&
  true &&
  true &&
  request.userInfo.username == "lucas" &&
  has(request.object.v1) &&
  request.object.v1.spec.gatewayClassName == 'test-gateway'"
```

Unified policies through Partial Evaluation

Note that even though the object is unknown, in some cases partial evaluation can evaluate to a concrete true or false:

For example, if user *hacker* executes `kubectl create -f gateway.yaml`

```
"true &&
  true &&
  true &&
  false &&
  has(request.object.v1) &&
  request.object.v1.spec.gatewayClassName == 'test-gateway'"
```



Expression is always false, no matter what the object is! Don't decode, instead Deny

Kubernetes Authorization

Kubernetes considers an ordered list (chain) of authorizers.

Today, there are three decision modes: `Allow`, `Deny`, or `NoOpinion`.

If `Allow` or `Deny`, enforce decision. If `NoOpinion`, consult next authorizer.

Kubernetes Authorization with Conditions

Kubernetes considers an ordered list (chain) of authorizers.

Today, there are three decision modes: `Allow`, `Deny`, or `NoOpinion`.

If `Allow` or `Deny`, enforce decision. If `NoOpinion`, consult next authorizer.

Conditional Authorization adds a new decision mode: `Conditional`, along with a set of conditions.

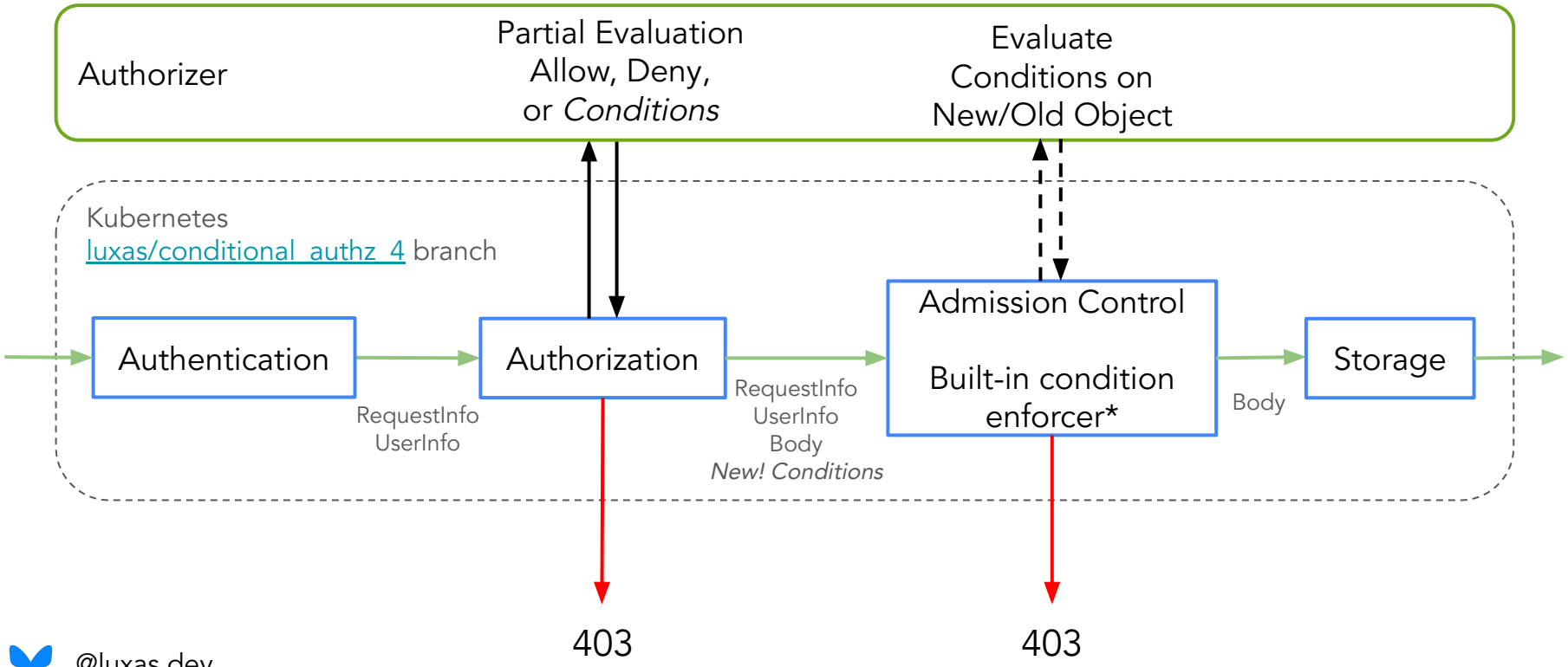
The request can proceed to the admission stage if the decision is `Allow` or `Conditional` with at least one condition that can make the request get authorized.

Conditional Authorization: Data model

A condition has the following properties:

- `condition`: String with some predicate on the request/stored objects.
- `type`: What language the condition is written in (e.g. CEL, OPA, or Cedar)
- `effect`: `Allow`, `Deny`, or `NoOpinion`. How to treat “true”.
- `id`: Authorizer-unique identifier (for reason and error messages)

Conditional Authorization + Cedar policy language

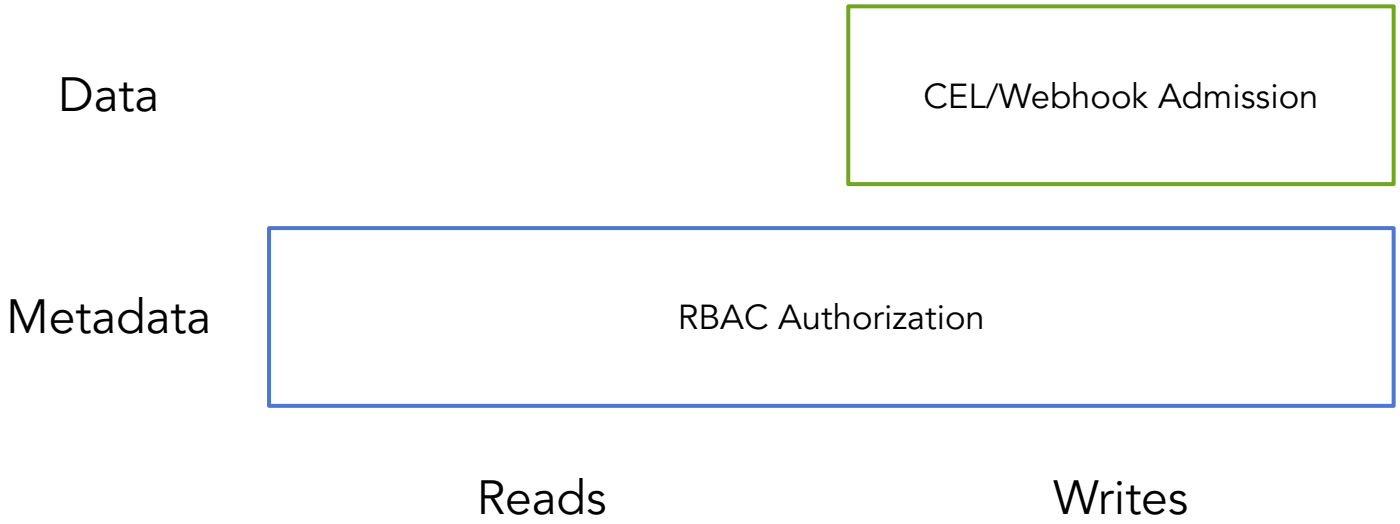


@luxas.dev

*If conditions are expressed in CEL, no need to ask the authorizer again

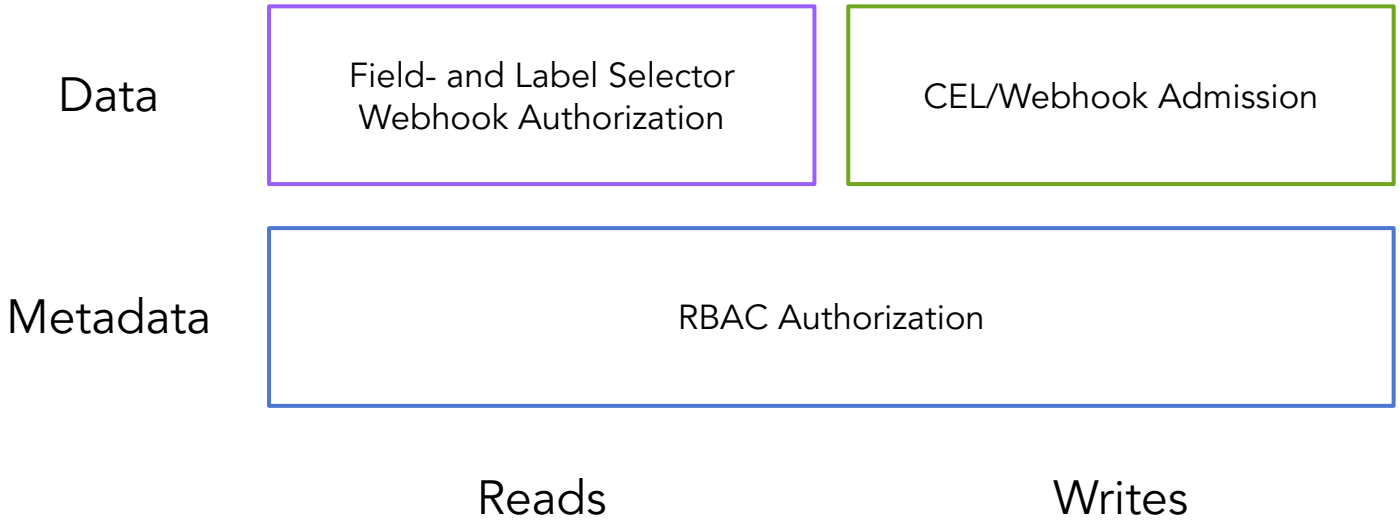
KEP-4601 Authorize with Selectors

Before KEP-4601, there was no way to say “authorize a read only with labelSelector ingress=true”.



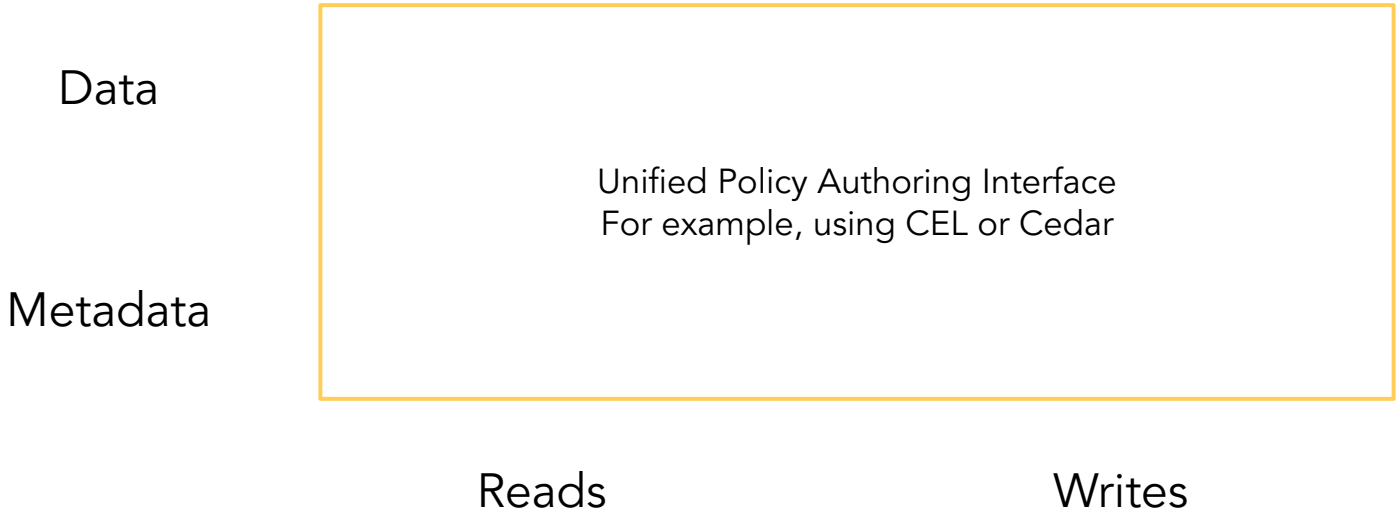
KEP-4601 Authorize with Selectors

With KEP-4601, expressing “authorize a read only with labelSelector ingress=true” is possible using a webhook authorizer (not builtin RBAC).



KEP-4601 Authorize with Selectors

Utilizing both KEPs together, makes it possible to express all types of authorization using the same, unified interface for the user.



Open Source Authorization Engine

KubeCon CloudNativeCon North America 2025

Open Source Authorization Engine

Aims to be expressive, fast, safe, and analyzable

@luxas.dev

KubeCon CloudNativeCon North America 2025

Open Source Authorization Engine

Aims to be expressive, fast, safe, and analyzable

Maintains an encoding into decidable Mathematical Logic (SMT)*

@luxas.dev

*Satisfiability Modulo Theories Logic. Announcement

KubeCon CloudNativeCon North America 2025

Open Source Authorization Engine

Aims to be expressive, fast, safe, and analyzable

Maintains an encoding into decidable Mathematical Logic (SMT)*

Supports RBAC, ReBAC and ABAC paradigms

@luxas.dev

*Satisfiability Modulo Theories Logic. Announcement

KubeCon CloudNativeCon North America 2025

Open Source Authorization Engine

Aims to be expressive, fast, safe, and analyzable

Maintains an encoding into decidable Mathematical Logic (SMT)*

Supports RBAC, ReBAC and ABAC paradigms

AWS has donated Cedar to the CNCF \o/

@luxas.dev

*Satisfiability Modulo Theories Logic. Announcement

Demo time! Available on 

KubeCon CloudNativeCon North America 2025

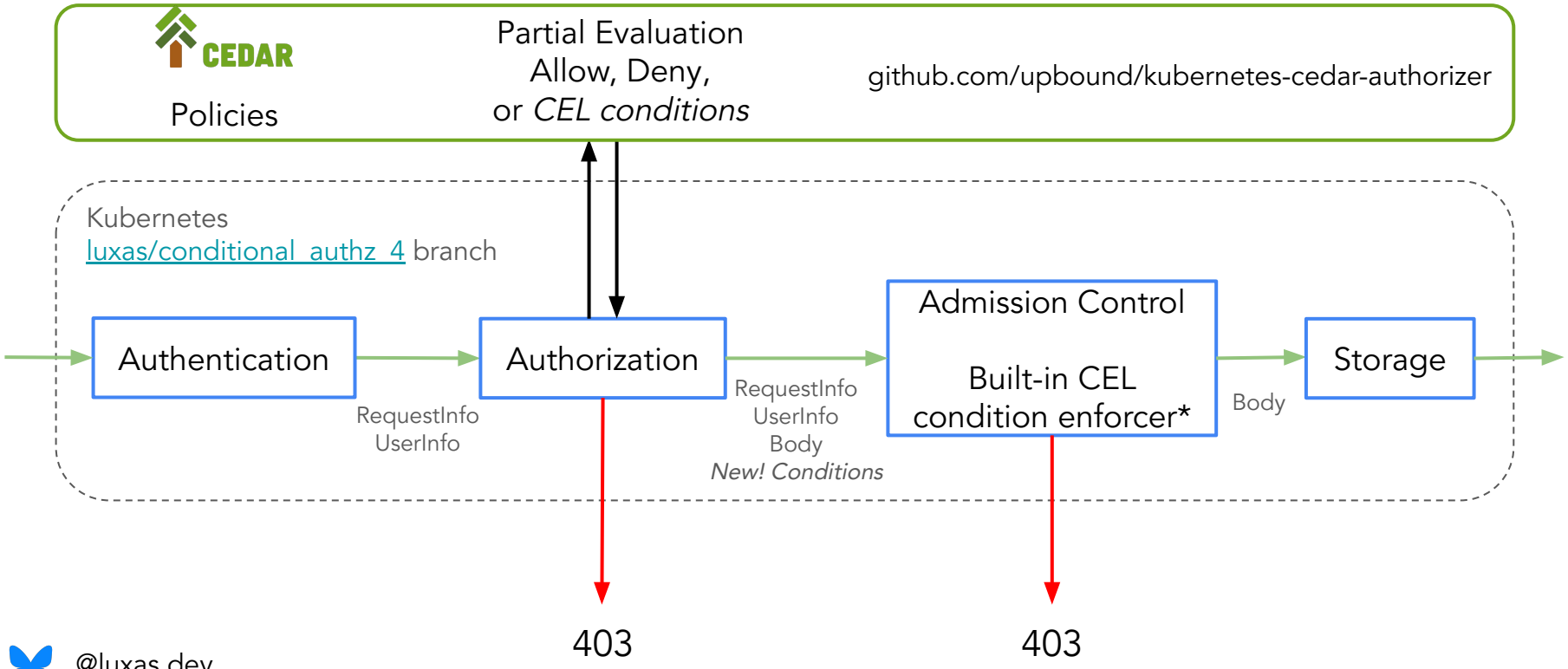
upbound

kubernetes cedar authorizer

unified Kubernetes authorization
using the Cedar Policy Language

 github.com/upbound/kubernetes-cedar-authorizer

Conditional Authorization + Cedar policy language



@luxas.dev

*If CEL condition support is integrated into core, use that, otherwise use the webhook

Examples

“Let user micahhausler only read and write secrets with .type=tls”

Examples

"Let user micahhausler only read and write secrets with .type=tls"

```
permit (
    principal is k8s::User,
    action in [k8s::Action::"list", k8s::Action::"watch"],
    resource is core::secrets
) when {
    principal.username == "micahhausler" &&
    resource has stored.v1.type &&
    resource.stored.v1.type == "kubernetes.io/tls"
};
```

```
permit (
    principal is k8s::User,
    action in [k8s::Action::"create", k8s::Action::"update"],
    resource is core::secrets
) when {
    principal.username == "micahhausler" &&
    resource has request.v1.type &&
    resource.request.v1.type == "kubernetes.io/tls"
};
```

Write Condition

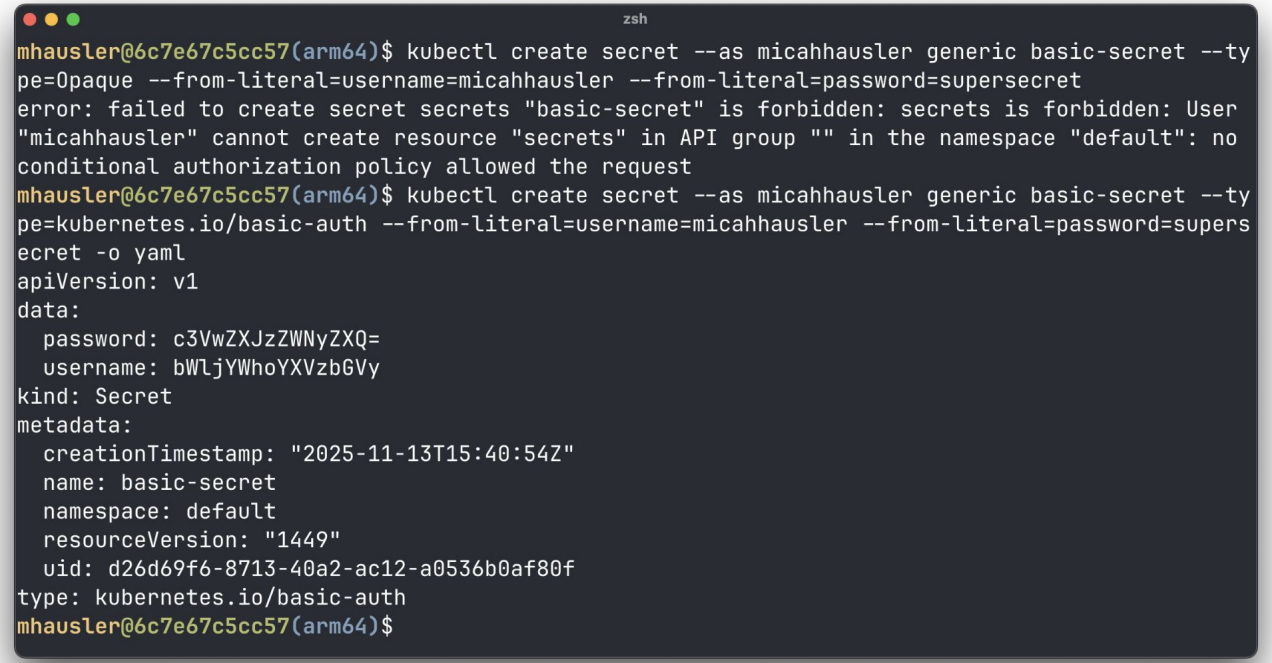
```
permit (
    principal is k8s::User,
    action in [k8s::Action::"create", k8s::Action::"update"],
    resource is core::secrets
) when {
    principal.username == "micahhausler" &&
    resource has request.v1.type &&
    resource.request.v1.type == "kubernetes.io/tls"
};
```

zsh

```
mhausler@6c7e67c5cc57(arm64)$ kubectl create --as micahhausler -f self-sar-create.json -o yaml
apiVersion: authorization.k8s.io/v1
kind: SelfSubjectAccessReview
metadata: {}
spec:
  resourceAttributes:
    namespace: default
    resource: secrets
    verb: create
    version: v1
status:
  allowed: false
  conditionsChain:
  - conditions:
    - condition: (true && (true && (true && (true && (true && (true && (has(object.type)
        && (object.type == 'kubernetes.io/basic-auth'))))))))
      effect: Allow
      id: policy0
      type: k8s.io/authorization-cel
  reason: conditionally authorized
mhausler@6c7e67c5cc57(arm64)$
```

Conditional Write

```
permit (
    principal is k8s::User,
    action in [k8s::Action::"create", k8s::Action::"update"],
    resource is core::secrets
) when {
    principal.username == "micahhausler" &&
    resource has request.v1.type &&
    resource.request.v1.type == "kubernetes.io/tls"
};
```

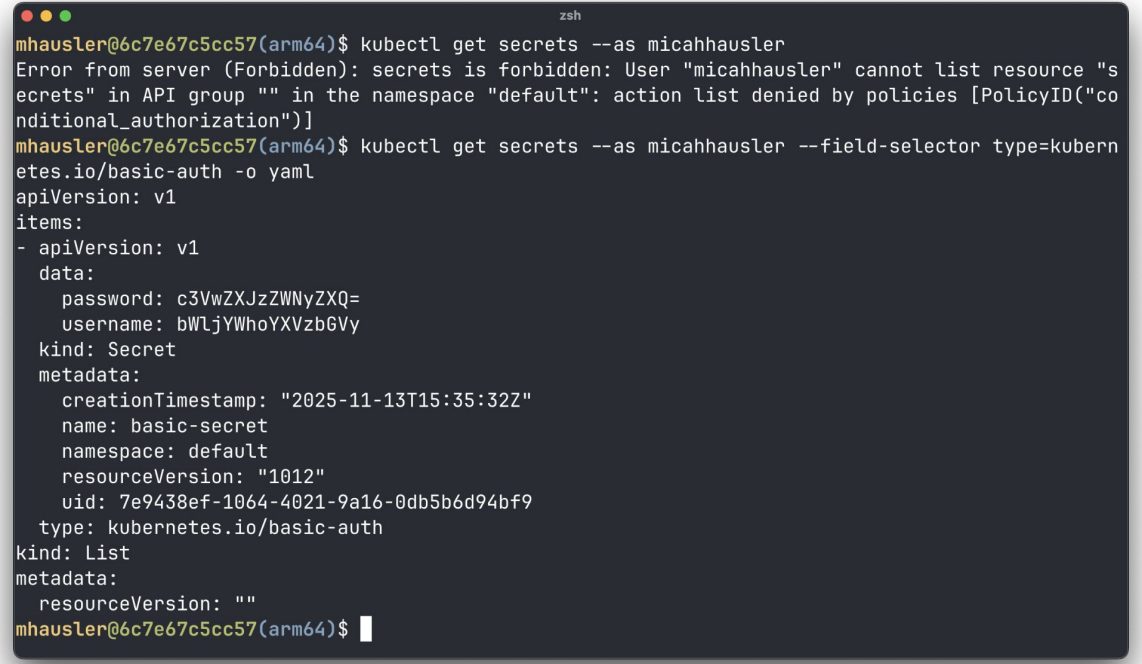


Conditional Read

```
permit (
    principal is k8s::User,
    action in [k8s::Action::"list", k8s::Action::"watch"],
    resource is core::secrets
) when {
    principal.username == "micahhausler" &&
    resource has stored.v1.type &&
    resource.stored.v1.type == "kubernetes.io/tls"
};
```

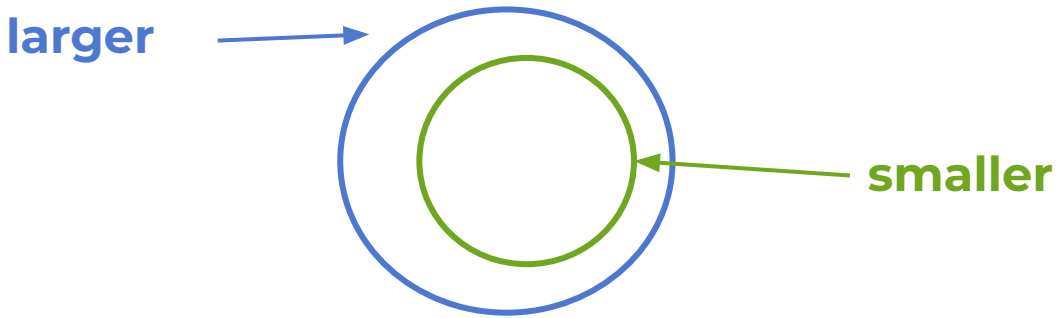
Conditional Read

```
permit (
    principal is k8s::User,
    action in [k8s::Action::"list", k8s::Action::"watch"],
    resource is core::secrets
) when {
    principal.username == "micahhausler" &&
    resource has stored.v1.type &&
    resource.stored.v1.type == "kubernetes.io/tls"
};
```



What are the benefits of analyzable policies?

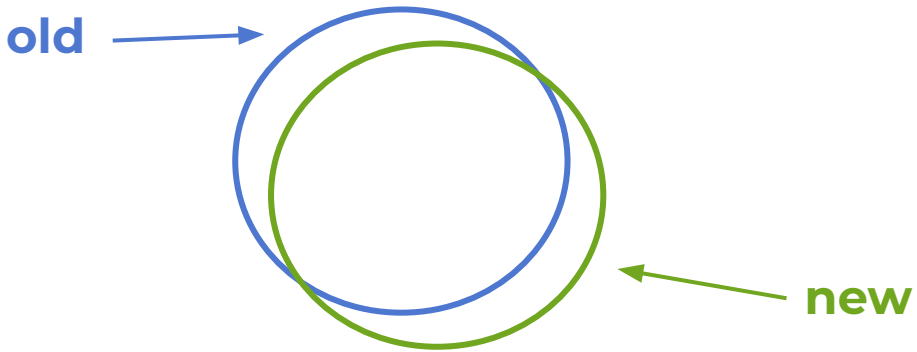
⇒ Compare permissiveness of two policies (or sets of them)



What are the benefits of analyzable policies?

⇒ Compare permissiveness of two policies (or sets of them)

⇒ Check for equality (help refactors)



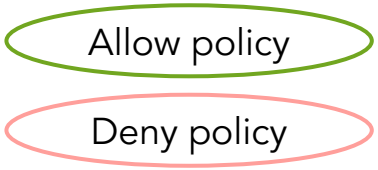
Uh oh! New, refactored policy not maybe as planned?

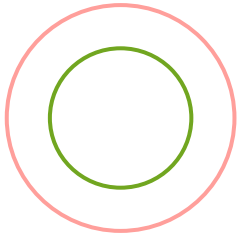
What are the benefits of analyzable policies?

⇒ Compare permissiveness of two policies (or sets of them)

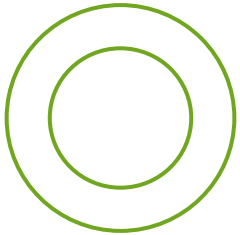
⇒ Check for equality (help refactors)

⇒ Check for logical inconsistencies in a policy set (shadowing, no-ops)





← Deny shadows allow



← Allow shadows allow

← No effect

KubeCon CloudNativeCon
North America 2025

Conclusion

Where to go from here?

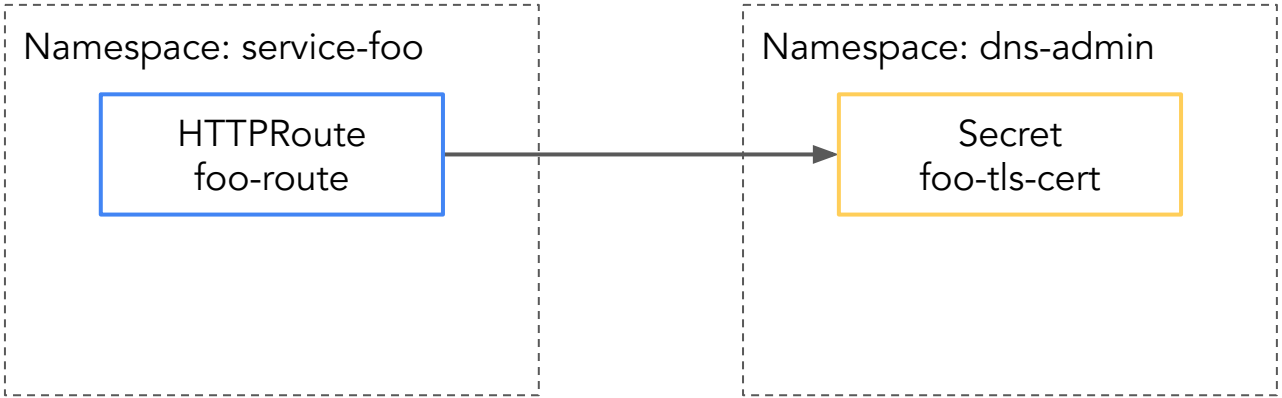
Join the discussion,

give feedback to us!

#KubeCon #CloudNativeCon

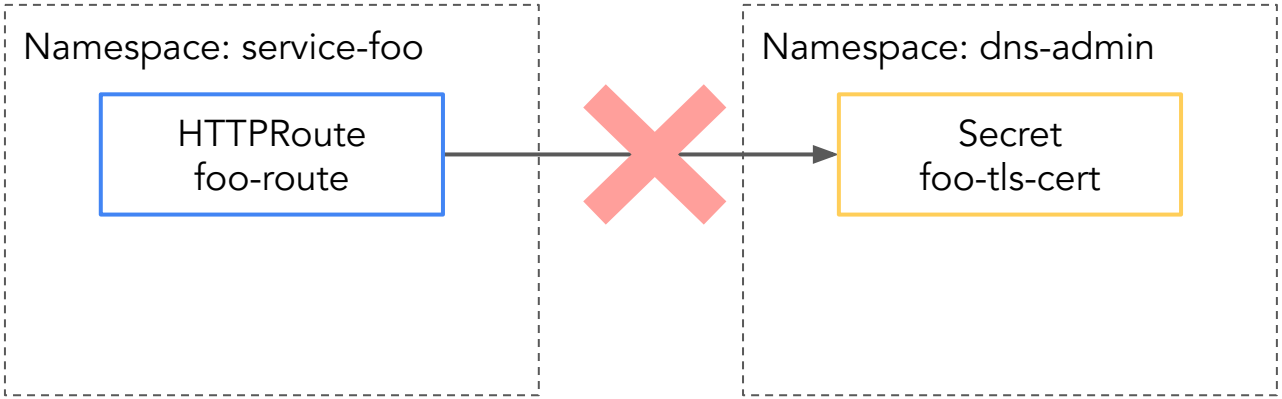
ReferenceGrant: Protecting object-to-object references

All of the aforementioned authorization policies have focused on “what can a principal do in the system?”



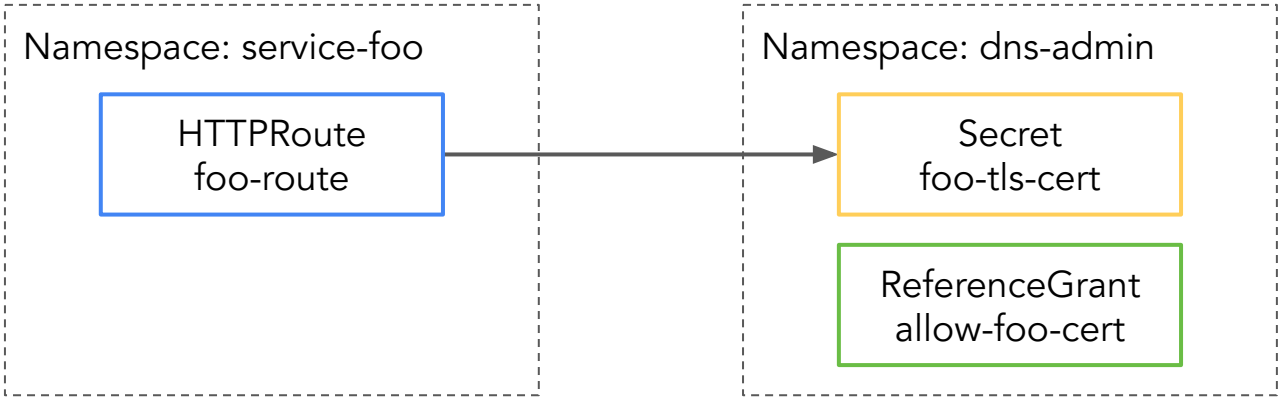
ReferenceGrant: Protecting object-to-object references

All of the aforementioned authorization policies have focused on “what can a principal do in the system?”



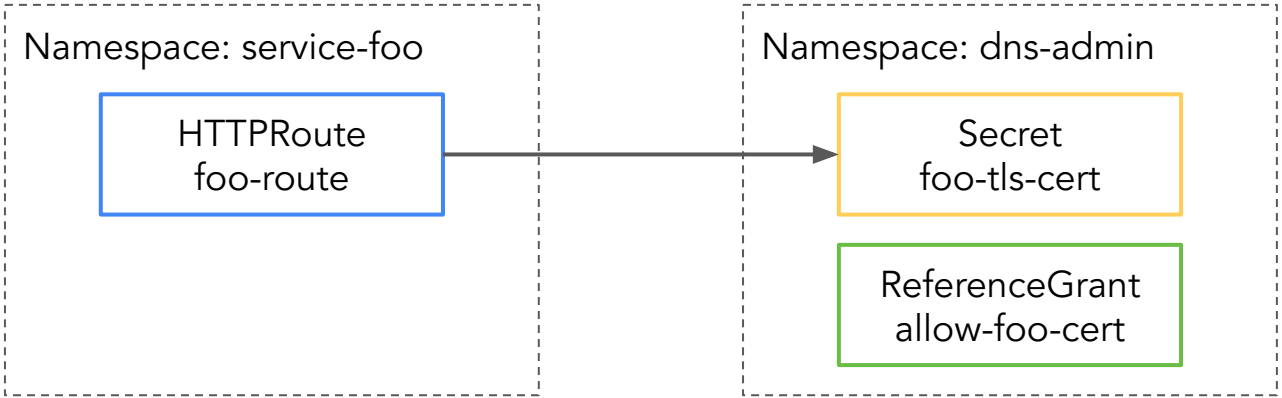
ReferenceGrant: Protecting object-to-object references

All of the aforementioned authorization policies have focused on “what can a principal do in the system?”



ReferenceGrant: Protecting object-to-object references

All of the aforementioned authorization policies have focused on “what can a principal do in the system?”



A good writeup on the state of the union by Uwe Krüger here.

Next Steps

1. Join #sig-auth-authorizers-dev in the Kubernetes Slack and/or the SIG Auth biweekly meetings!

Next Steps

1. Join #sig-auth-authorizers-dev in the Kubernetes Slack and/or the SIG Auth biweekly meetings!

2. Comment on the KEP-5681 Conditional Authorization proposal! (Hopefully implemented in 1.36)

Next Steps

1. Join #sig-auth-authorizers-dev in the Kubernetes Slack and/or the SIG Auth biweekly meetings!

2. Comment on the KEP-5681 Conditional Authorization proposal! (Hopefully implemented in 1.36)

3. Try out the mentioned projects, give feedback and contribute

Next Steps

1. Join #sig-auth-authorizers-dev in the Kubernetes Slack and/or the SIG Auth biweekly meetings!

2. Comment on the KEP-5681 Conditional Authorization proposal! (Hopefully implemented in 1.36)

3. Try out the mentioned projects, give feedback and contribute

4. When the primitives are settled, work can proceed on the higher-level features RBAC++ and ReferenceGrant

KubeCon | CloudNativeCon

North America 2025

Thanks!

Bluesky:
@luxas.dev & @micahhausler.com

LinkedIn:
luxas & micahhausler

CNCF Slack(s):
luxas & micahhausler

Credits: Icons by Flaticon