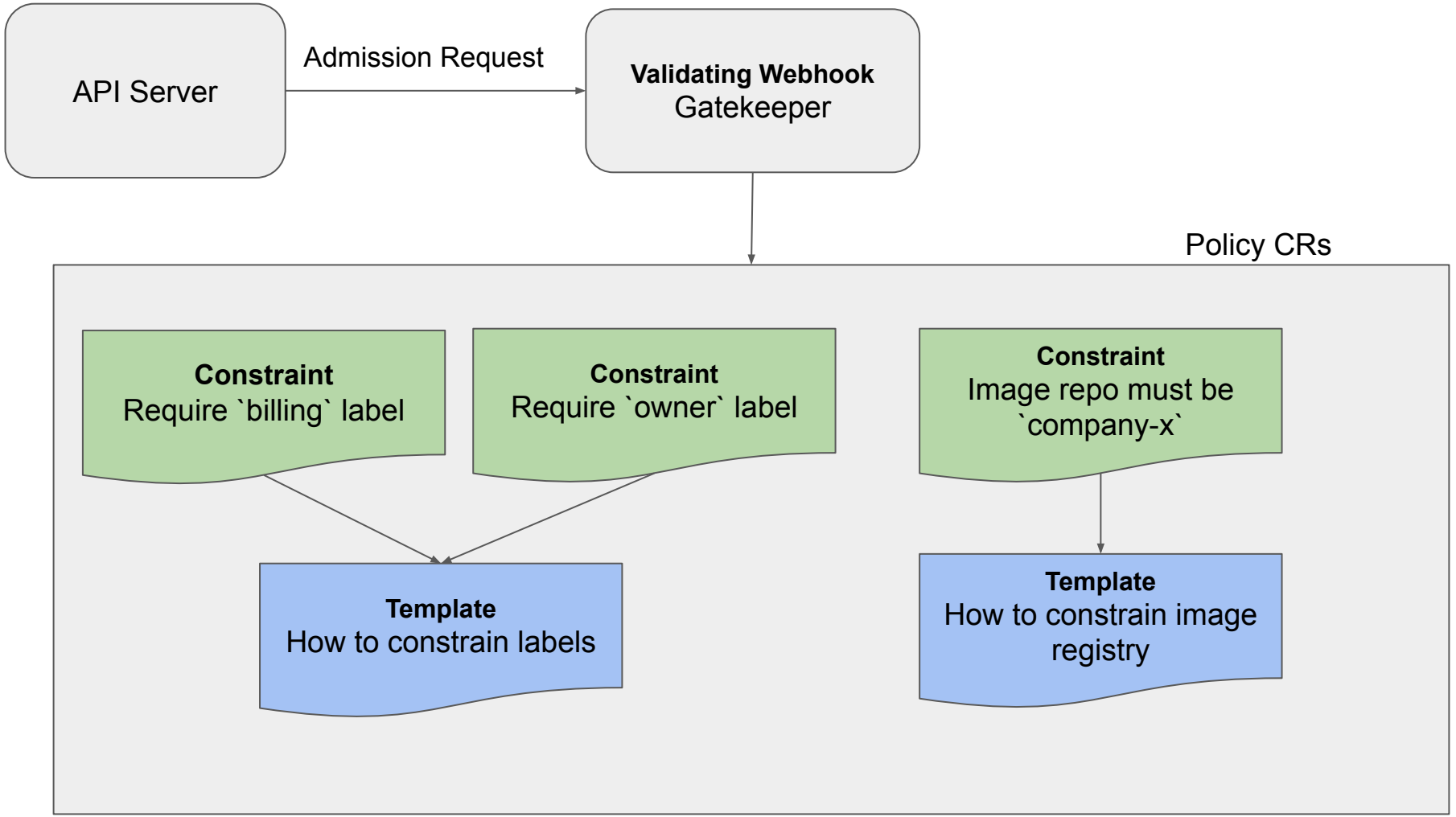
Design Patterns for Extensible, Scalable K8s Extensions

Max Smythe (@maxsmythe, Google)

Rita Zhang (@ritazzhang, Microsoft)

OPA Gatekeeper

A customizable Kubernetes admission webhook that helps enforce policies and strengthen governance


API Server
Admission Request
Validating Webhook
Gatekeeper
Policy CRs
Constraint
Require `billing` label
Constraint
Require `owner` label
Constraint
Image repo must be `company-x`
Template
How to constrain labels
Template
How to constrain image registry

Policy is a Team Effort

- **Gatekeeper:** defines how policy looks and how it can be bound
  - "I care about managing and enforcing constraints and templates, not what's in them"
- **Template authors:** figure out how to implement common, generic checks
  - "I want to be able to restrict labels on resources. I don't care about which resources, which labels or how this gets enforced"
- **Cluster admins:** figure out what checks they want to use
  - "I want to tell the system to restrict the `owner` label for objects in the `prod` namespace"

Duck Typing

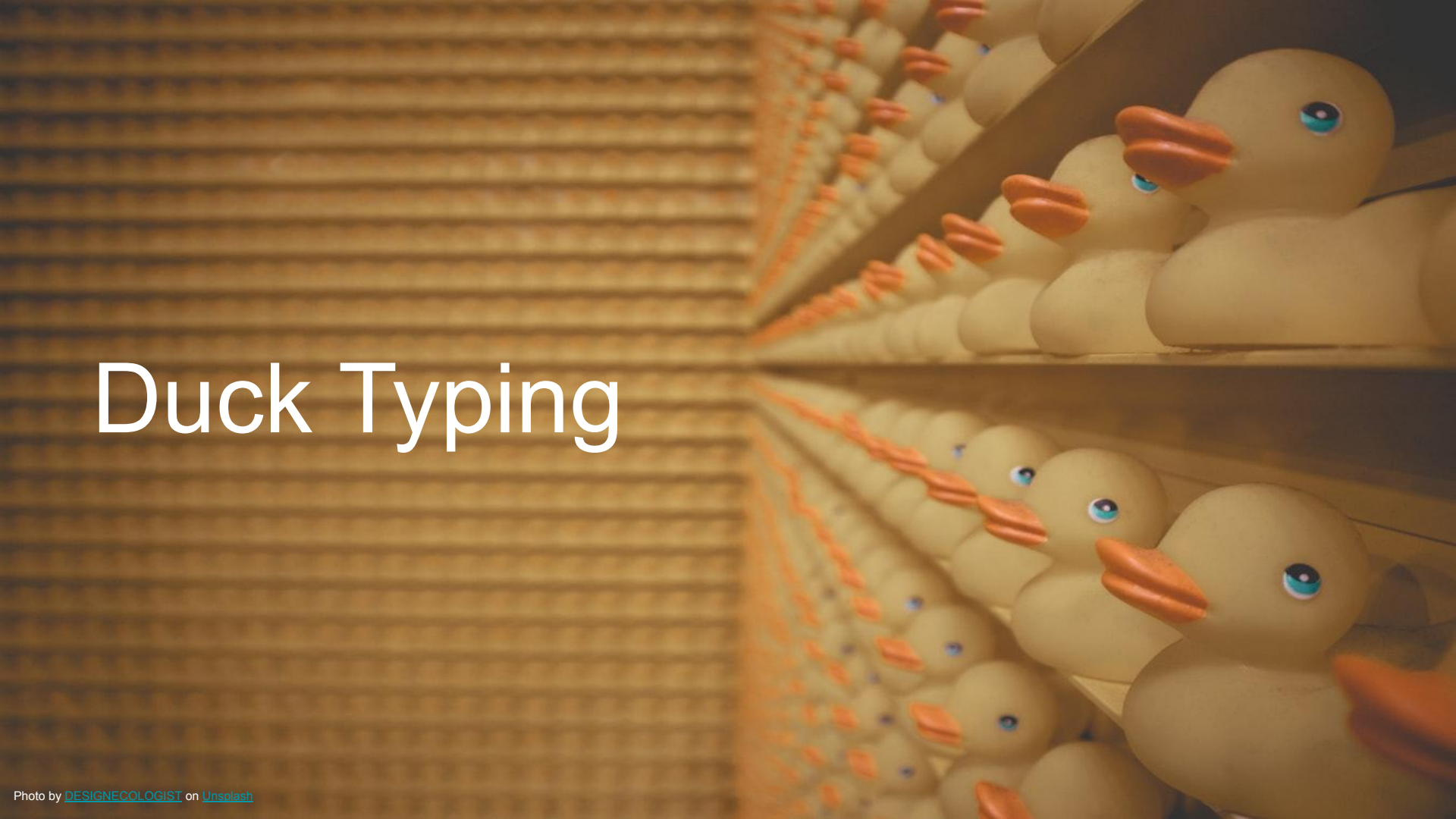
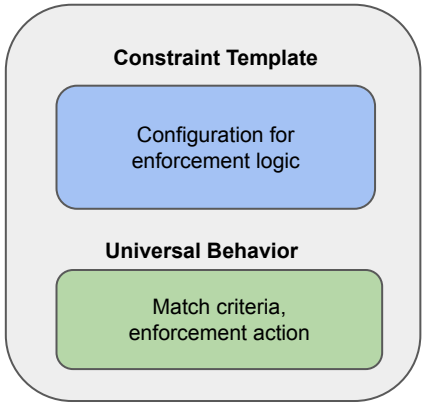


Photo by DESIGNECOLOGIST on Unsplash

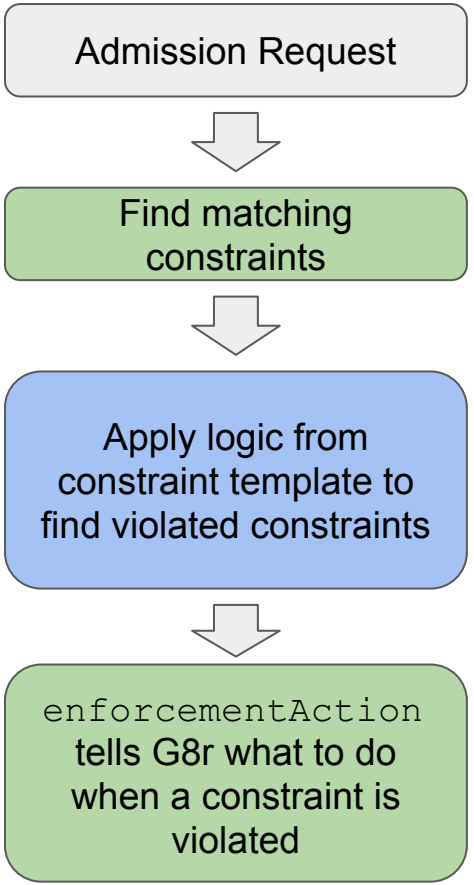
Ducks + Constraints

- Duck typing was presented by Matt Moore, Scott Nichols and Ville Aikas at previous Kubecons
- Abstracts common behaviors into pluggable components:
  - policy binding
  - policy logic
  - violation response
- Allows template authors to focus only on the thing they want to test when writing extensions to Kubernetes
- Admins see only a constraint object

Constraint Schema



Request Flow

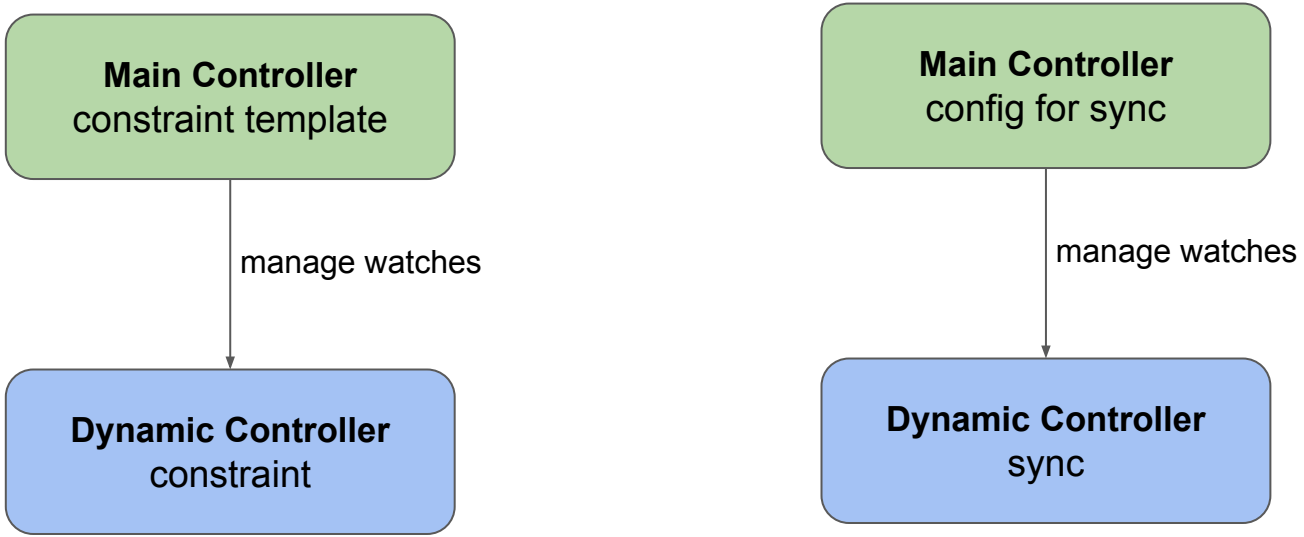


CRDs Creating CRDs == Hard

- Controllers must handle generic objects
  - Use unstructured resources
  - Deserialize pieces of the unstructured resource into Golang structs for strong schema
- Merge multiple JSON Sub-Schemas into different roots in resultant schema to encourage duck typing and avoid collision
- Handling dynamic watches
  - Originally we did this by creating a "sub manager" that would restart every time the set of watched resources changed
    - Inefficient memory usage because controller-runtime's watch cache was duplicated
    - Required finalizers to catch delete events missed due to submanager restart
    - Oren Shomron wrote a dynamic watcher, allows us to add/remove watches without restarts or finalizers

Controller of Controllers

- Controllers must watch constraint templates and configs, adding and removing watches as necessary



Example Registrar Usage

```
registrar, err := watchManager.NewRegistrar("my-controller", eventsCh)
if err != nil {
    return err
}
```

```
if err := registrar.AddWatch(gvk); err != nil {
    return err
}

if err := registrar.RemoveWatch(gvk); err != nil {
    return err
}
```

`eventsCh` is a channel that receives watch events to trigger reconcile loops

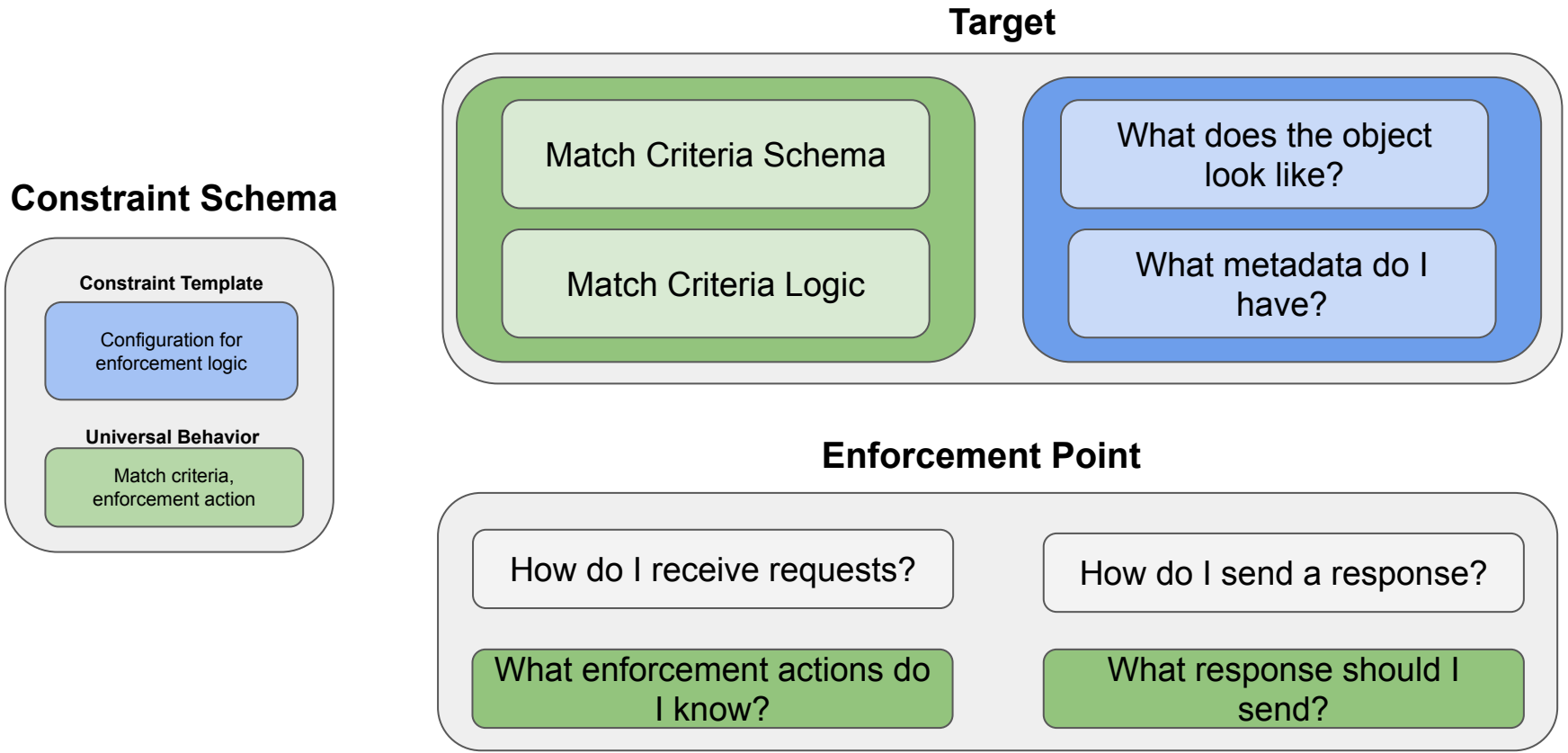
Registrars

- Registrars simplify writing multiple dynamic controllers
  - A registrar can be requested for each dynamic controller
  - Each registrar is namespaced to that controller so they are non-interacting
  - Each registrar is capable of adding, removing or replacing an intent to watch a GVK
  - The set of watched GVKs is the union of all intents across all registrars
- Adding a layer of indirection and namespacing intent allows both the sync and constraint dynamic controllers to interact with the same watch manager without worry

Going Full Meta

- Most config policy is looking at an object and giving a thumbs up or down
- Does this *have* to be done at admission time?
- Does it *have* to be against KRM-style resources?
- Not if you duck type... **the decision process itself!**

Constraint Framework: Full Meta



Real World Examples

- Gatekeeper uses the Constraint Framework, also...
- The gatekeeper-validate KPT function can be used to validate K8s configs at rest or as part of a CI/CD pipeline
- Cloud Config Validator has been used to:
  - Validate GCP resources as part of a Forseti server deployment
  - Validate GCP resource snapshots at rest via CFT Scorecard
  - Validate Terraform plans using Terraform Validator

Photo by Vidar Nordli-Mathisen on Unsplash

Webhook + Controller == Webtroller

Webhooks

- Serve requests
- Must be responsive
- Downtime intolerant
- Availability scales with # of pods
- Capacity scales with # of pods
- Flat hierarchy

Controllers

- Observe and reconcile resources
- Eventually consistent
- Downtime-tolerant
- Generally singletons
- Sometimes use leader election
- Recovery speed improved with multiple pods

Gatekeeper

Webhook that serves results based off of observed resources

These Solutions are Fundamentally Incompatible?

Maybe, except...

- Idempotent processes don't need to be singletons
  - ex: All GK pods would have created the same constraint CRD from a template, just let them all do it and the winner will succeed, the rest won't retry
- Write conflicts may lead to unnecessary traffic, but controllers only write when they have changes

Leaderless Horizontal Scalability

- Multiple webhook pods all serving simultaneously
  - Relies on auto healing and multiple serving peers across failure domains for availability
- All pods will try to ingest a template and create a CRD, one will win
- All pods manage their own private cache of constraints/templates/data
- Non-idempotent operations, like audit, must run in a separate, singleton pod
- Avoid scaling write contention
- No side effects allowed from controllers

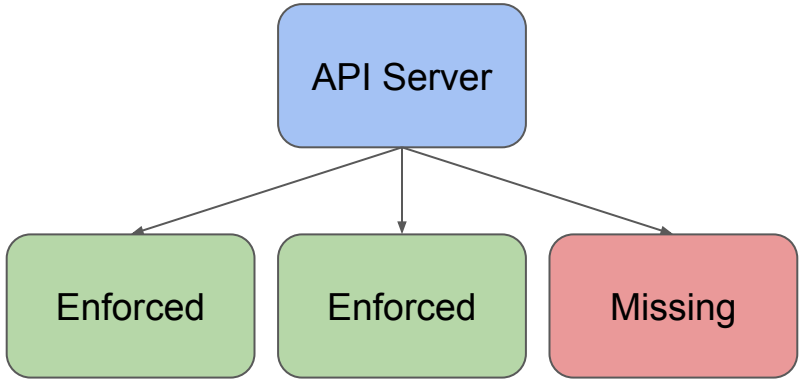
Okay, so multiple pods... now what?

Profit?

Nope. People want to know if policies are enforced.

Eventual Consistency

- Multiple pods mean multiple enforcers that are eventually consistent
- Policy enforcement is only as strong as its weakest link
- If I have:
  - 3 webhook pods
  - 2 enforcing a new policy, 1 not
  - An API server that chooses its webhook host randomly
- That new policy has a 66% chance of being enforced by the webhook



byPod Status

- ID
  - uniquely identifies pod
- observedGeneration
  - lets us know which version of the resource that pod has seen
- operations
  - the Gatekeeper functions being performed by this pod
- templateUID
  - sanity check, in case a template was deleted and recreated
- errors
  - any errors ingesting the template

```
status:
  byPod:
  - id: gatekeeper-audit-67dfc46db6-bc5zc
    observedGeneration: 1
    operations:
    - audit
    - status
    templateUID: f86fdcb5-8390-4c24-8af8-164b3c47a4cd
  - id: gatekeeper-controller-manager-7cbc758844-4v9tq
    observedGeneration: 1
    operations:
    - webhook
    templateUID: f86fdcb5-8390-4c24-8af8-164b3c47a4cd
  - id: gatekeeper-controller-manager-7cbc758844-l46b7
    observedGeneration: 1
    operations:
    - webhook
    templateUID: f86fdcb5-8390-4c24-8af8-164b3c47a4cd
  - id: gatekeeper-controller-manager-7cbc758844-m2szb
    observedGeneration: 1
    operations:
    - webhook
    templateUID: f86fdcb5-8390-4c24-8af8-164b3c47a4cd
```

Interpreting byPod Status

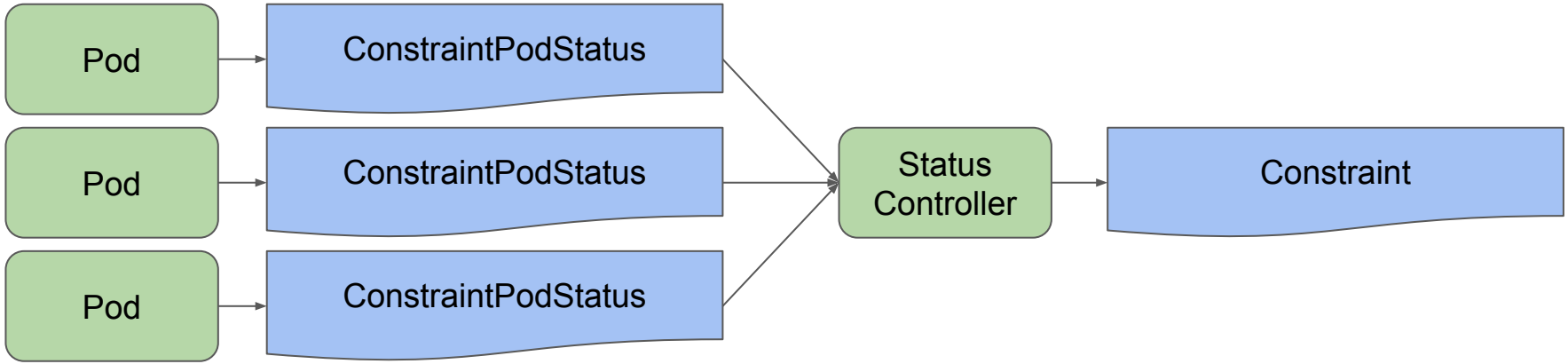
- If an entry is missing, assume the worst
- If deletion timestamp is set, assume the worst
- If you know you have 3 webhook pods, 3 status entries with the correct `observedGeneration` means that the constraint is enforced for the webhook

Implications on Infrastructure/Code/Design

- If we expect N pods, we must never have N + 1 pods
- Pods cannot serve until they have bootstrapped all initial constraints, templates, and data
- The semantics of multi-pod resources must be such that a missing resource can always be interpreted consistently (e.g. missing constraint => looser enforcement)
    - This makes referential constraints potentially problematic

Implementation of byPod Status

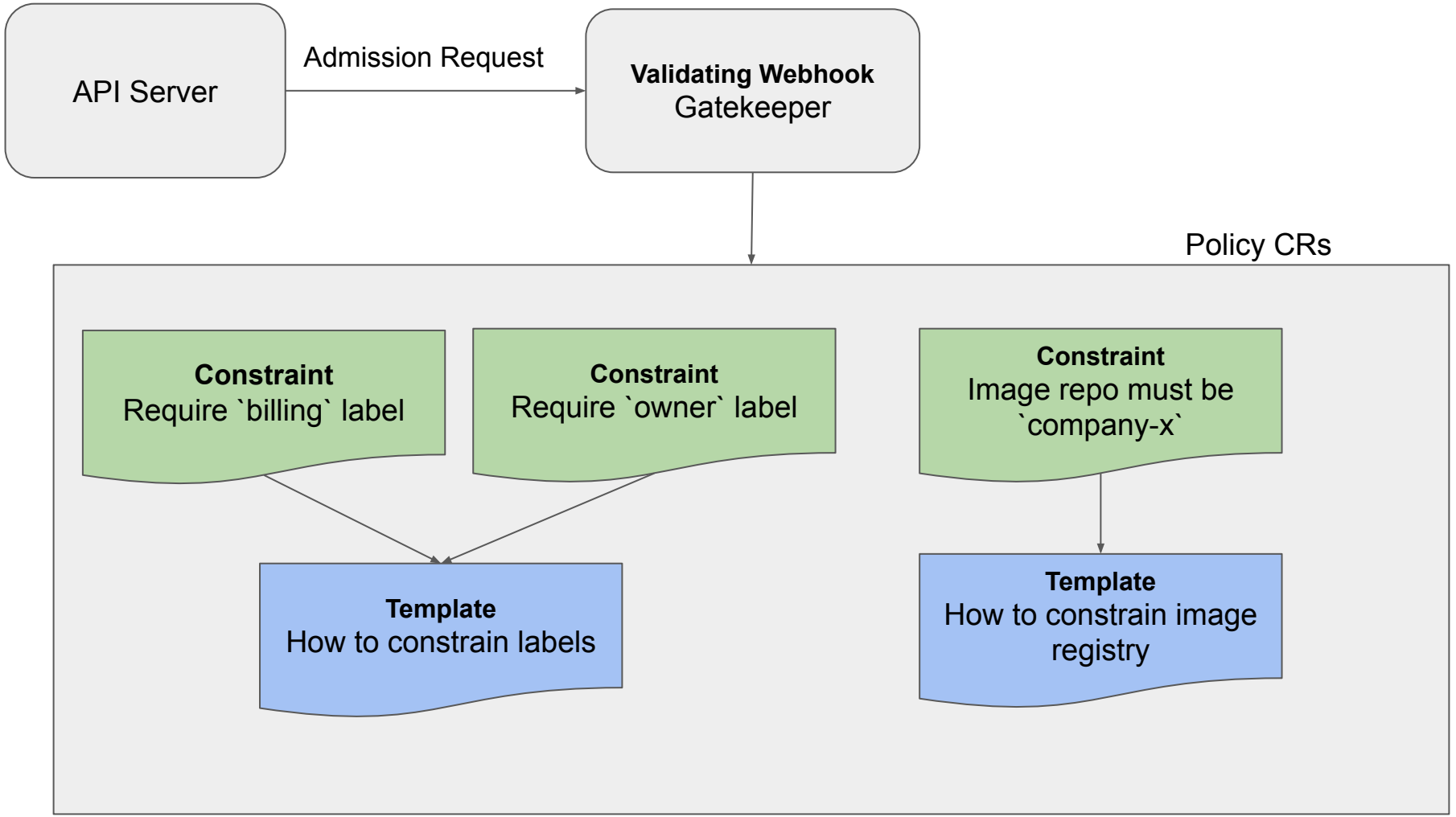
- Each pod writes to its own ConstraintPodStatus & ConstraintTemplatePodStatus resource
- These resources have an owner reference to the pod that writes to them
- A status controller copies pod statuses into the constraint/template



design doc

Fun, Mathy Side Analysis

- Increasing the number of webhook pods decreases the likelihood of webhook unavailability
  - If there are N pods, 1 Pod is sufficient to serve all webhook traffic and each server has P_f independent probability of being down, the probability that the whole system will fail is P_f^N
- It also increases the mean-time-to-enforcement for constraints/templates
  - If the probability for a single host ingesting a constraint by time t is P(t), then the probability that all N hosts will have ingested that constraint is $P(t)^N$, which makes longer ingestion times more likely. Thanks Wikipedia.


API Server
Admission Request
Validating Webhook
Gatekeeper
Policy CRs
Constraint
Require `billing` label
Constraint
Require `owner` label
Constraint
Image repo must be `company-x`
Template
How to constrain labels
Template
How to constrain image registry

Thank you

- Gatekeeper community
- Kubebuilder / controller-runtime
- Audience