
KubeCon
CloudNativeCon
North America 2019

Handling Risky Business: Cluster Upgrades

Puneet Pruthi

Puneet Pruthi

Engineering Manager, Orchestration

lyft

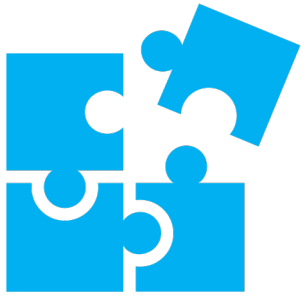
ppruthi@lyft.com

@iamppruthi

Handling Risky Business: Cluster Upgrades

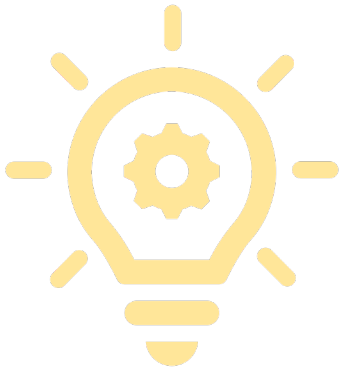
AGENDA

How we got here ?



Solving the puzzle

What's in the box ?



Learning

Kubernetes @ Lyft: Overview

- AWS
- Packer AMIs - Immutable Infrastructure
- Inbuilt cluster discovery and membership bootstrapping
- Complete control over Control & Data plane
- Scaling using Autoscaling Groups

Kubernetes @ Lyft: Upgrade Model

- Build new AMIs
- Update the launch config for ASG
- Terminate old nodes –> ASG boot nodes with new launch config
- Reconcile cluster state

How did we get here ?

May 2017,
Investigate
K8s on AWS

Dec 2017, MVP K8s
clusters are born

Mid 2018,
Instrumentation +
Testing workloads

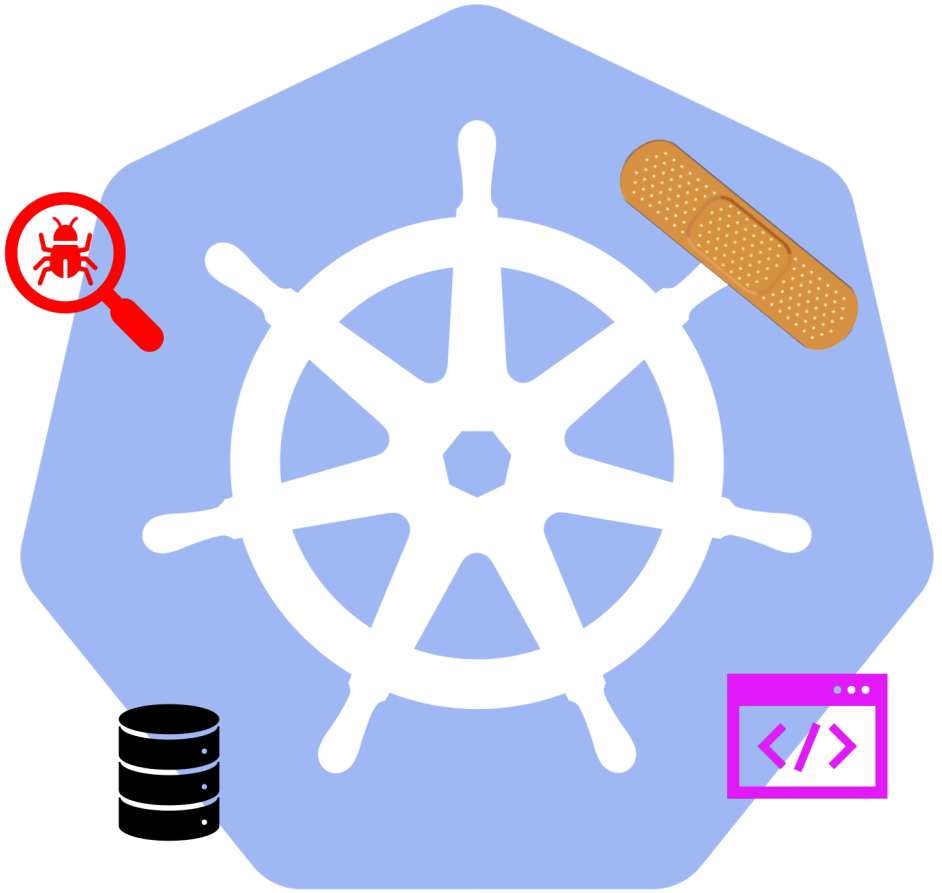
Mid 2018, Multi-AZ clusters,
Migration for legacy services begins

2019, Hypergrowth:
Bulk migration

Define the problem

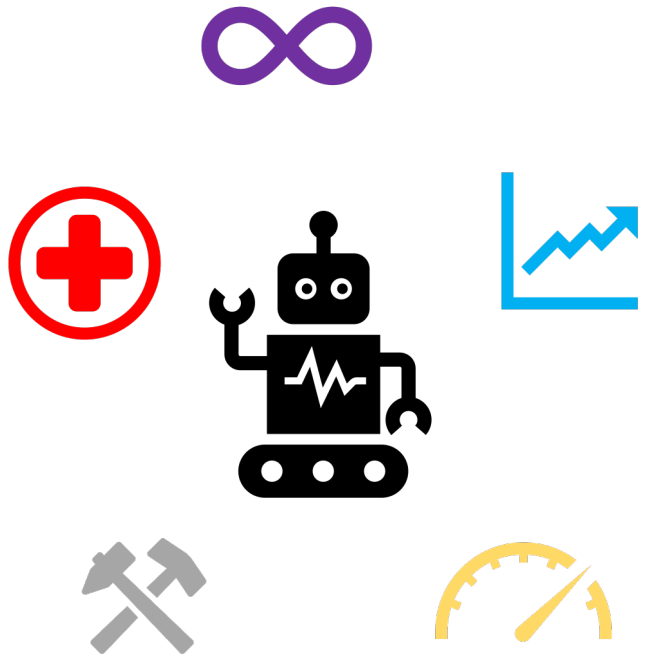
- Why must we upgrade ?
  - Kubernetes version bump
  - OS Security Patch
  - Infrastructure changes/Features rollout
  - Bug fixes
  - Data retention guidelines

- What to upgrade ?
  - Entire cluster
  - Select nodes (function / placement / labels / taints)



Magic Wand

- ~~Upgrade Window~~
- Serverless tooling
- Automate + Override
- No service disruption
- Maintains Cluster Health
- Allow Custom upgrades
- Observability

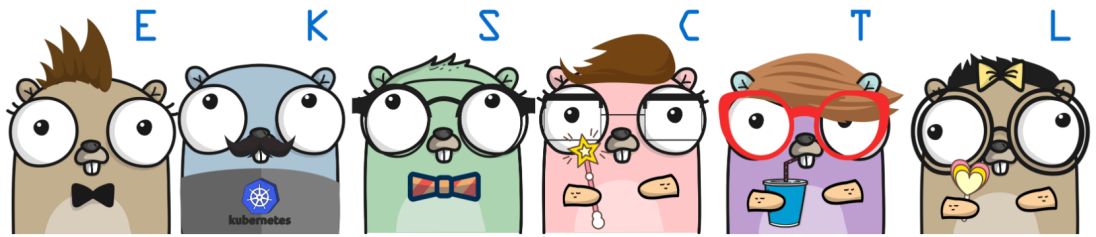


Landscape

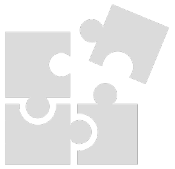
kubeadm

kops

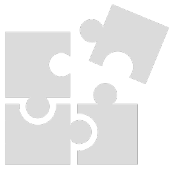
gcloud



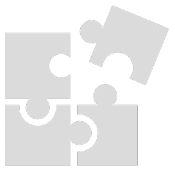
```
do
while
   !is_cluster_updated() and
   cluster_is_healthy() {
        NODE = get_old_node()
        kubectl cordon NODE
        kubectl drain NODE
        kubectl delete NODE
        terminate(NODE)
   }
```



```
do
while
   !is_cluster_updated() and
   cluster_is_healthy() {
        NODE = get_old_node()
        kubectl cordon NODE
        kubectl drain NODE
        kubectl delete NODE
        terminate(NODE)
   }
```

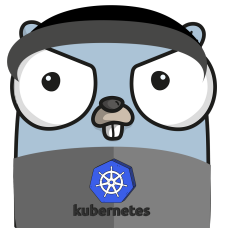


Iteration & Simplicity



K8srotator: v0

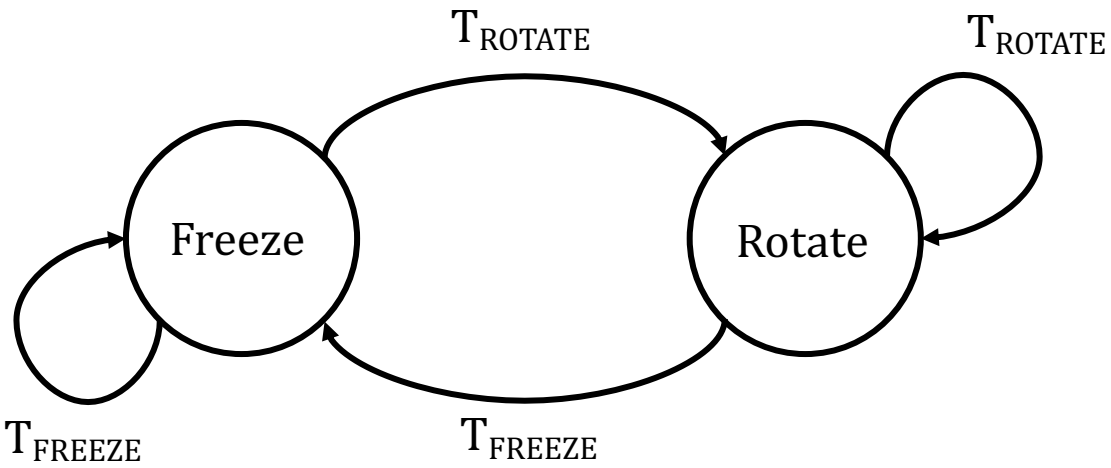
- Implements a custom controller in Golang
- Runs on Target Cluster as a singleton deployment
- Changes mode based on simple triggers (ConfigMaps)
- Scans cluster to identify OLD nodes
- Uses a modified MARK and SWEEP algorithm to remove nodes
- Maintain cluster health (basic version)



K8srotator: *FSM*

- Finite State Machine
  - Rotate
  - Freeze

- State transitions - controlled by triggers



K8srotator: *Triggers*

- Trigger: Freeze
  - Default trigger on startup
  - Pauses / Halts ongoing cluster upgrade
  - Auto applied in case of failure / error

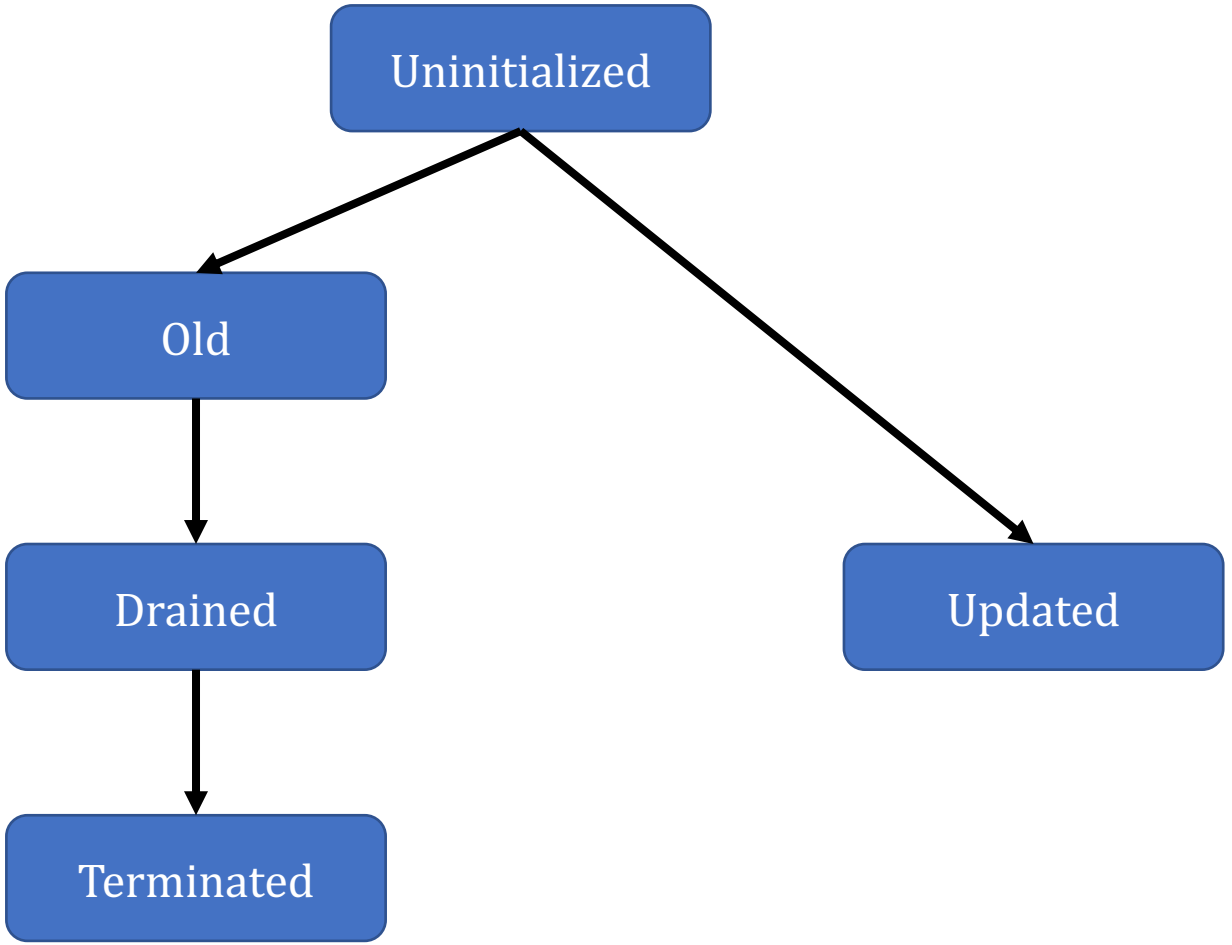
```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: k8srotator-override
  namespace: lyft-system
  labels:
    app: k8srotator
    lyft.net/component: k8srotator
data:
  config: |
    command: FREEZE
    clusterName: REPLACE_CLUSTER_NAME
    metadata: REPLACE_METADATA
    ...
```

K8srotator: *Triggers*

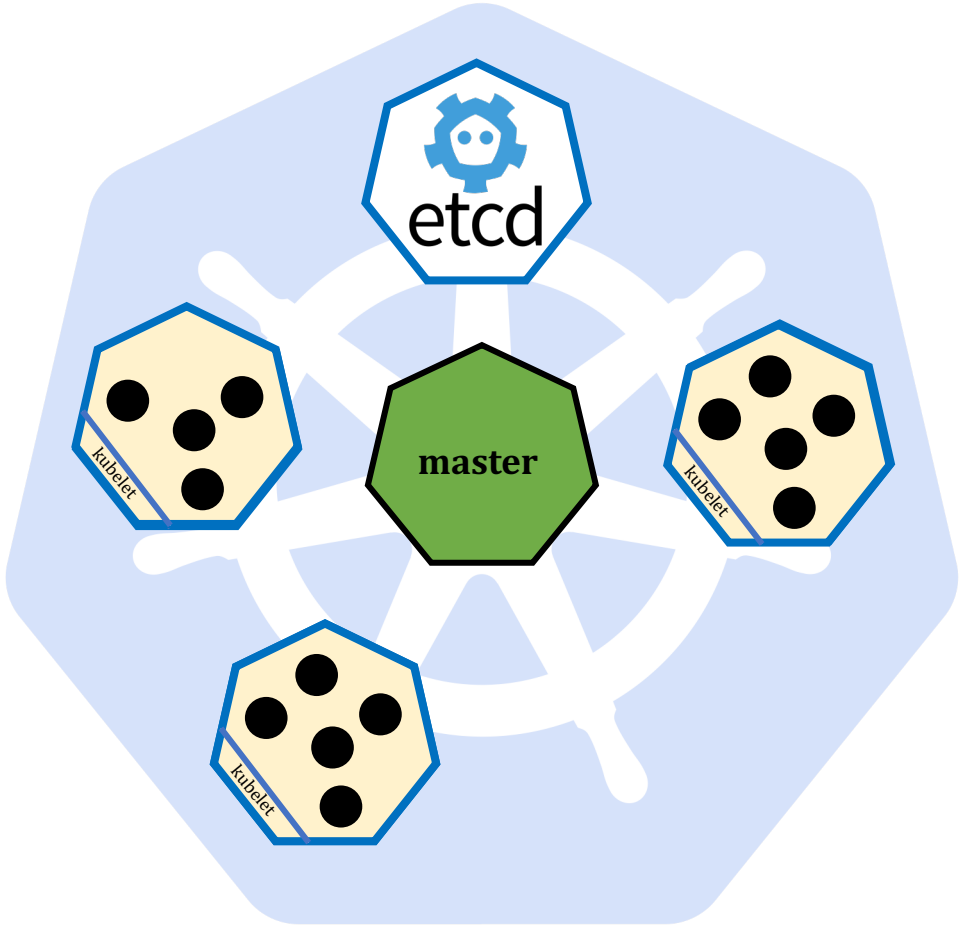
- Trigger: Rotate
  - Starts cluster rotation cycle
  - Specifies metadata about rotation
    - Target AMI info
    - maxSurge
    - maxDrainTime
    - etc.

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: k8srotator-rotate
  namespace: lyft-system
  labels:
    app: k8srotator
    lyft.net/component: k8srotator
data:
  config: |
    command: ROTATE
    clusterName: REPLACE_CLUSTER_NAME
    newAmiTag: REPLACE_AMI_TAG
    maxSurge: 1
    maxDrainTime: "REPLACE_MAX_DRAIN_TIME"
    dryRun: REPLACE_DRYRUN
    metadata: REPLACE_METADATA
    ...
```

K8srotator: *node lifecycle*



K8srotator: Core functionality



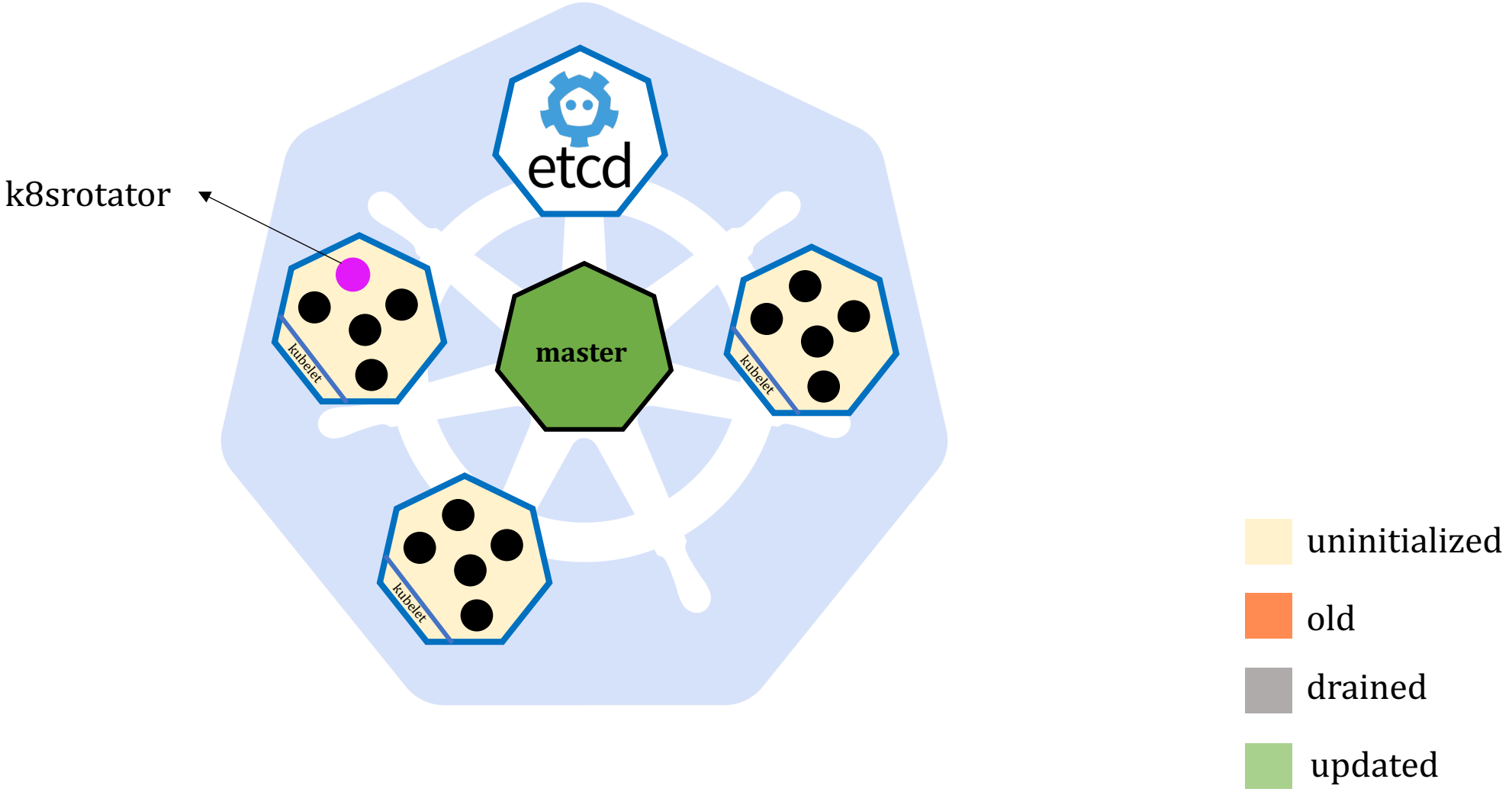
uninitialized

old

drained

updated

K8srotator: Core functionality

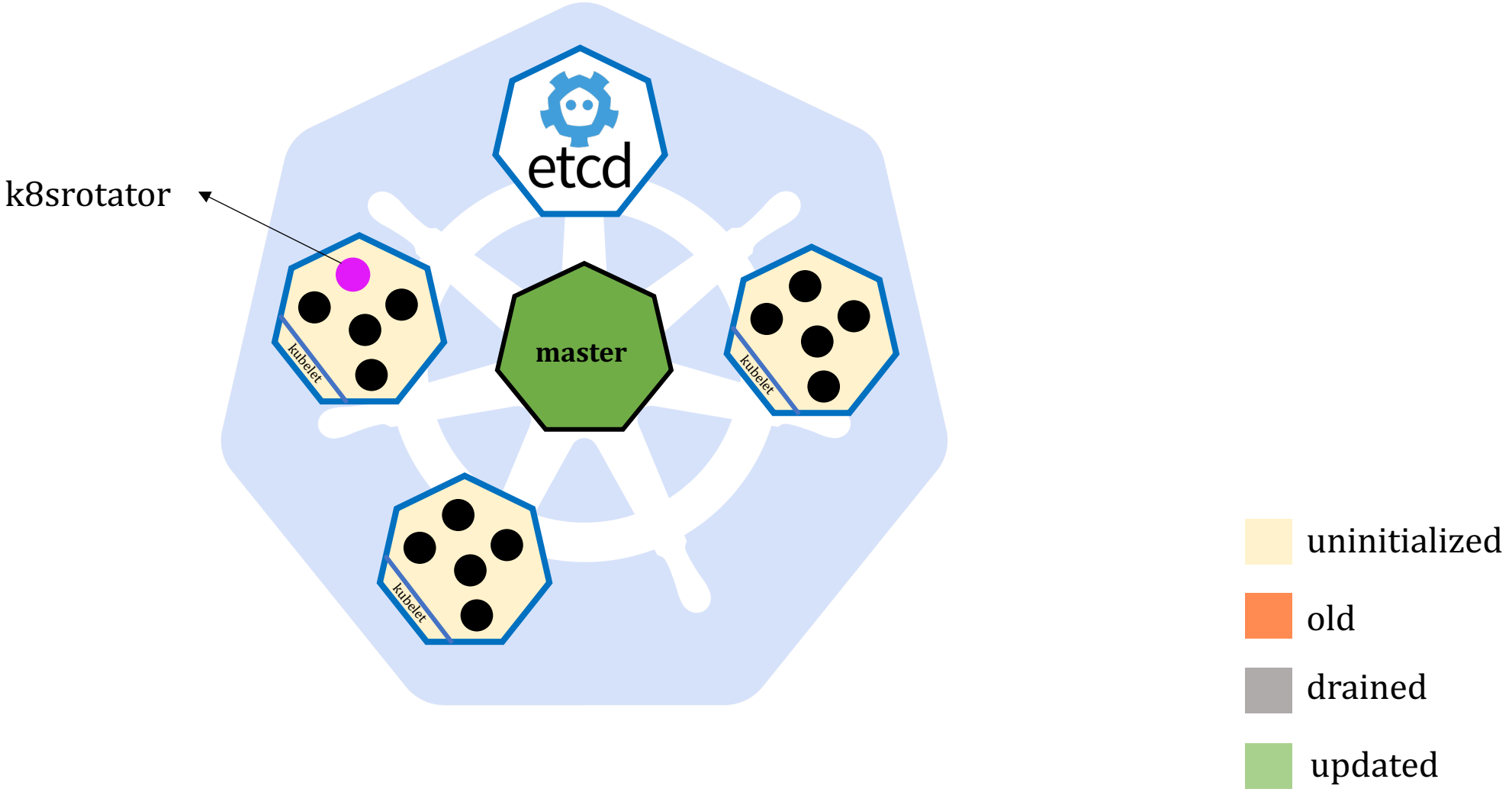


K8srotator: Core functionality

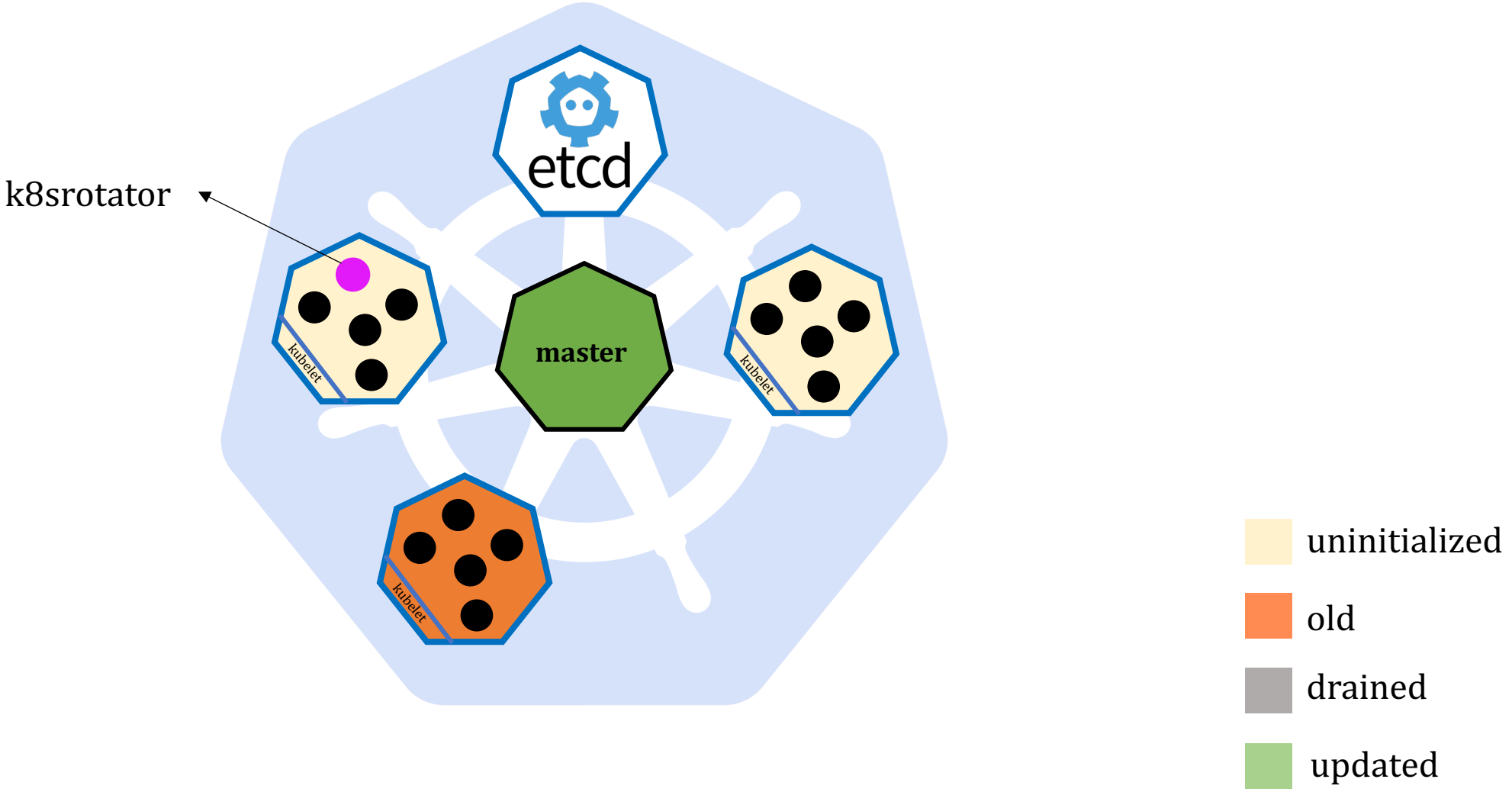

```
kubectl apply -f rotate.yaml
```

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: k8srotator-rotate
  namespace: lyft-system
  labels:
    app: k8srotator
    lyft.net/component: k8srotator
data:
  config: |
    command: ROTATE
    clusterName: mycluster
    newAmiTag: kube-v1.16
    maxSurge: 1
    maxDrainTime: "5m"
    nodeLabel: kubelet
```

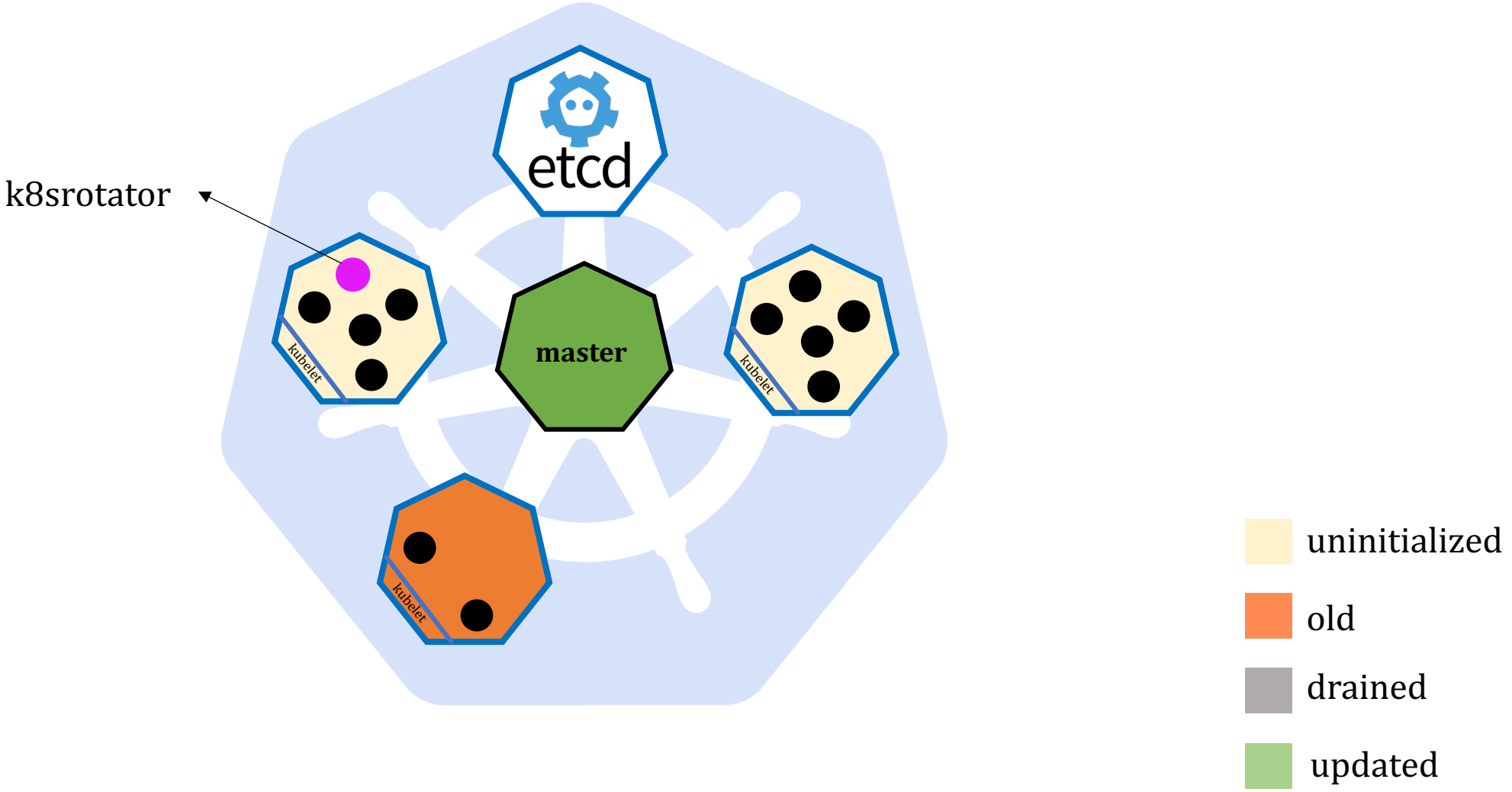
K8srotator: Core functionality



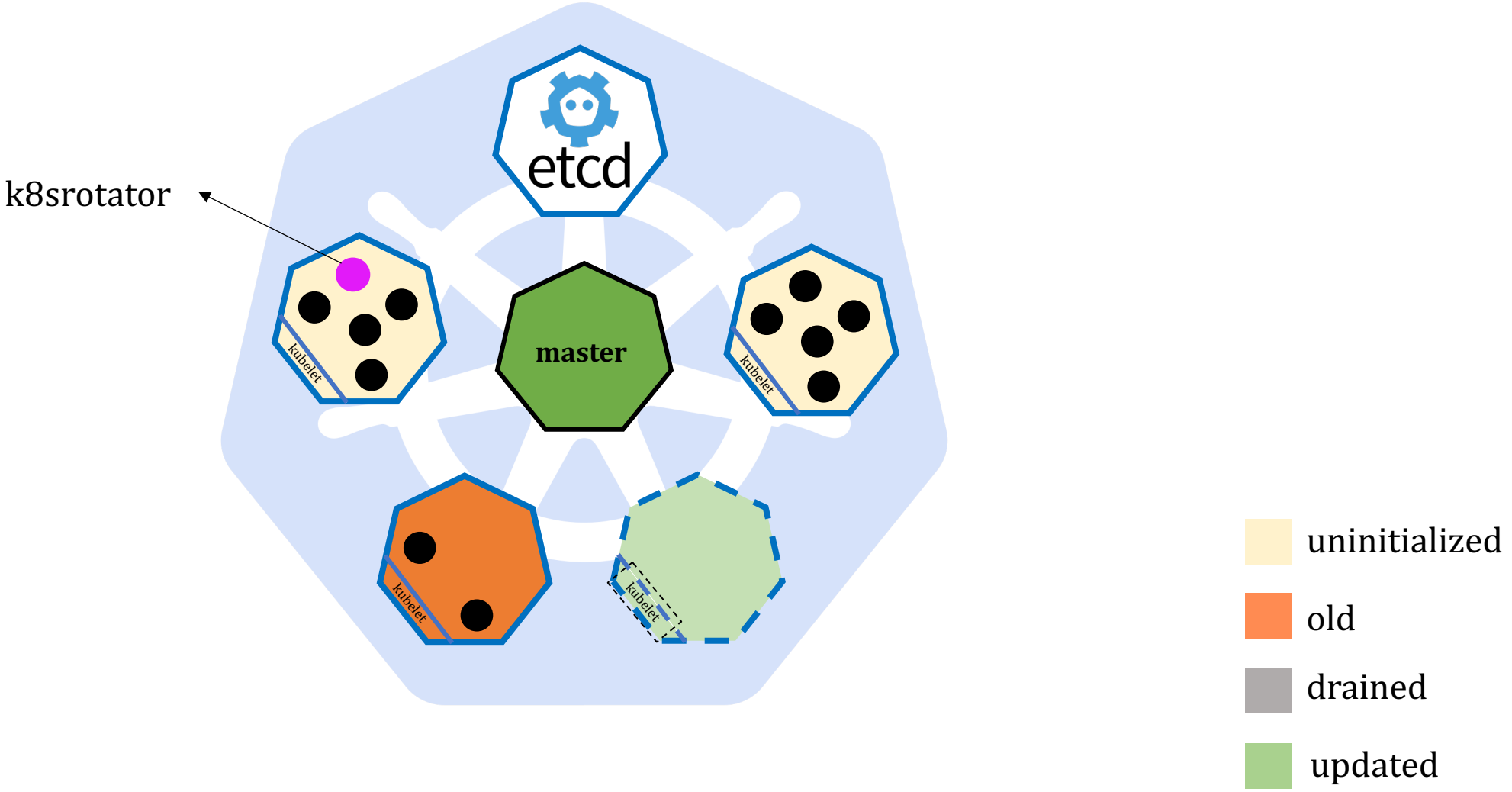
K8srotator: Core functionality



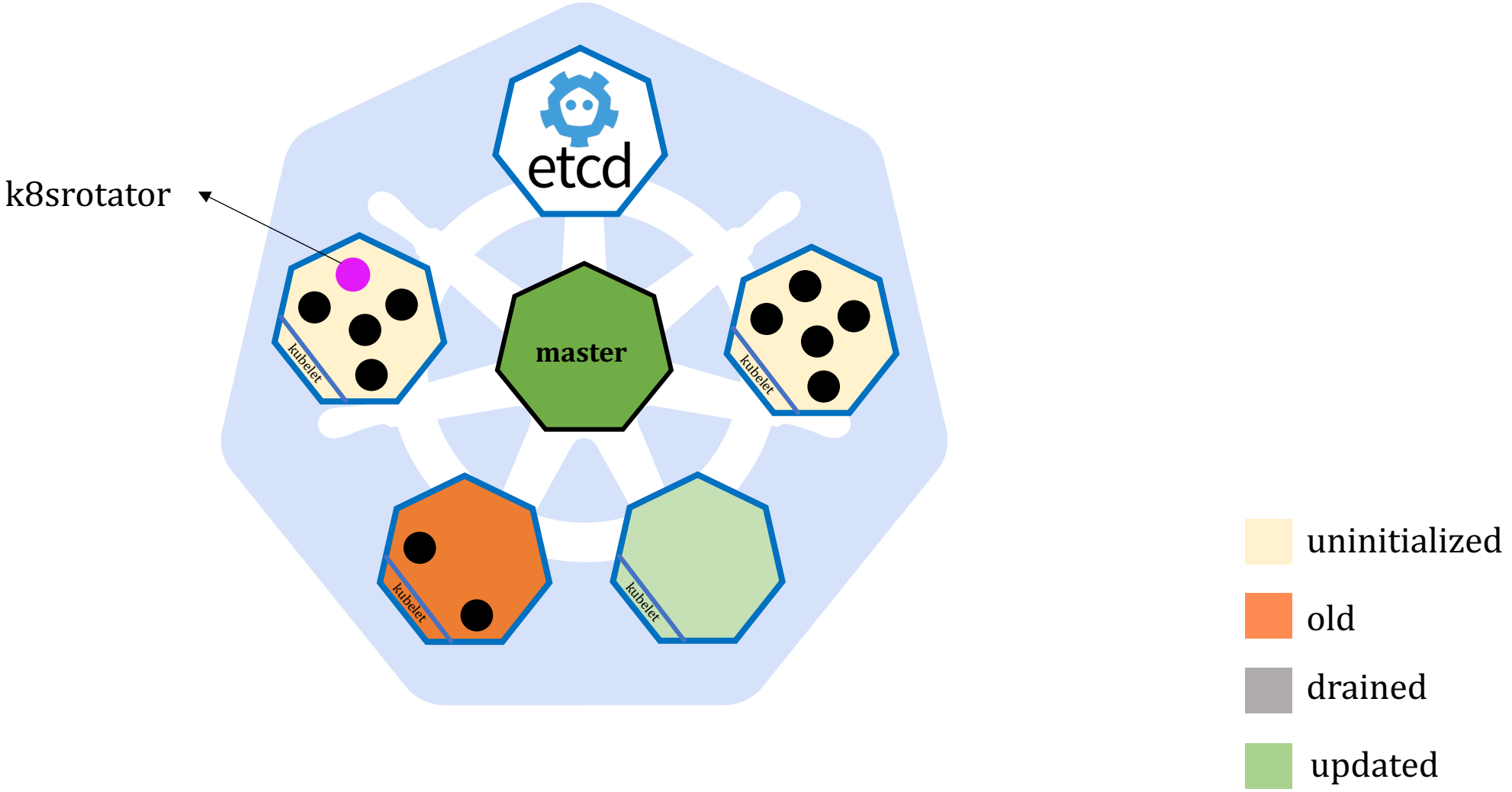
K8srotator: Core functionality



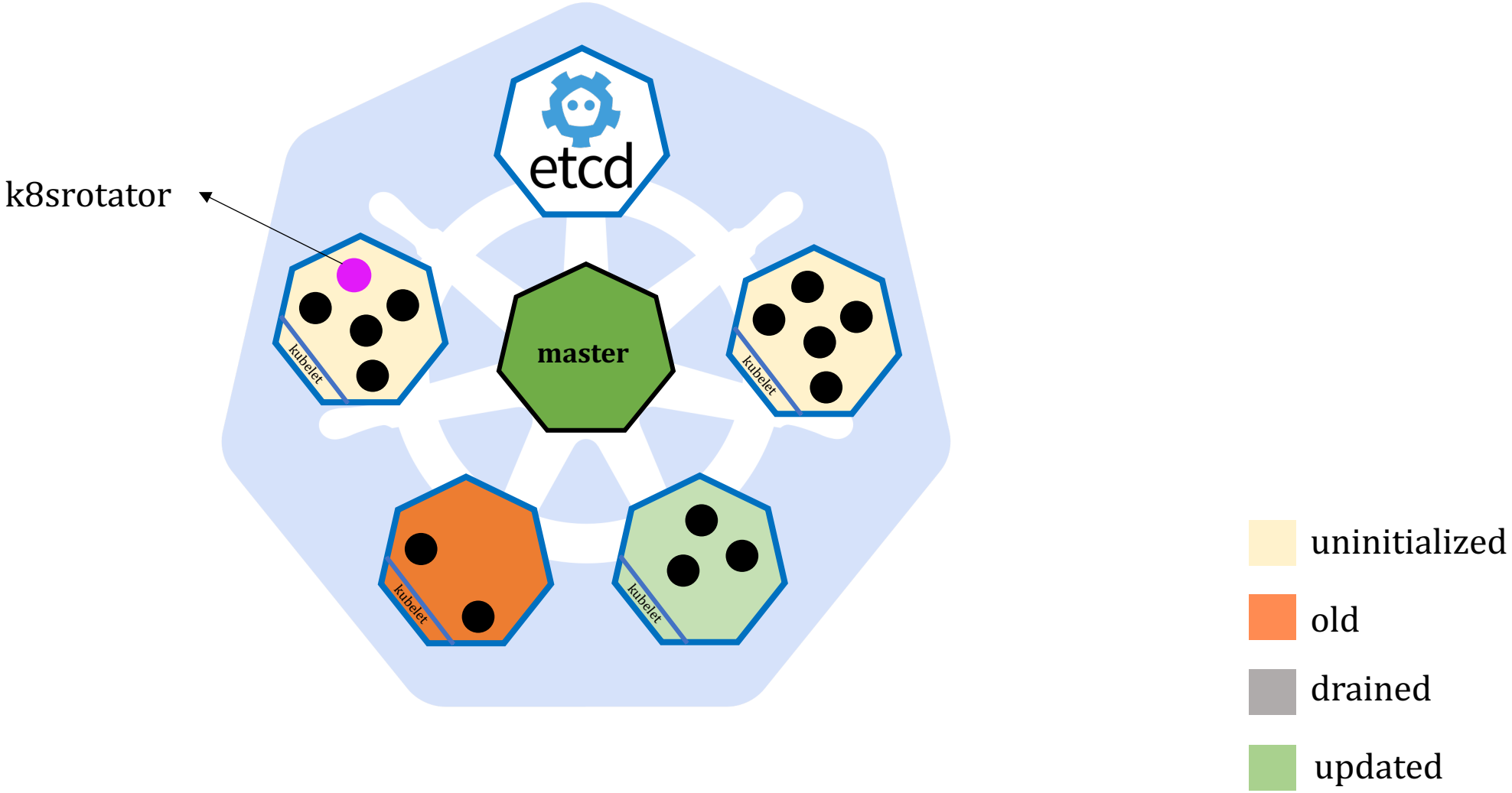
K8srotator: Core functionality



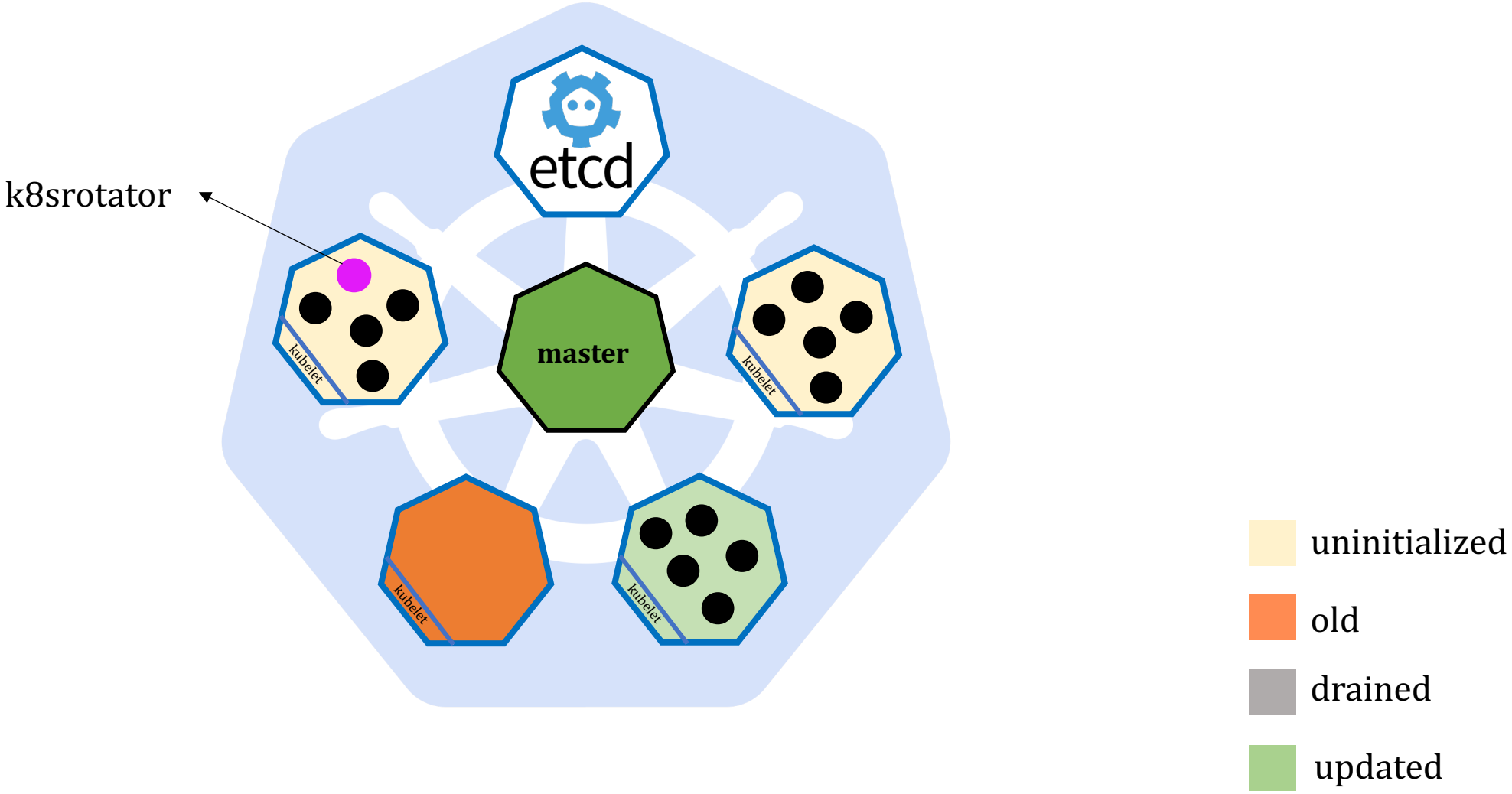
K8srotator: Core functionality



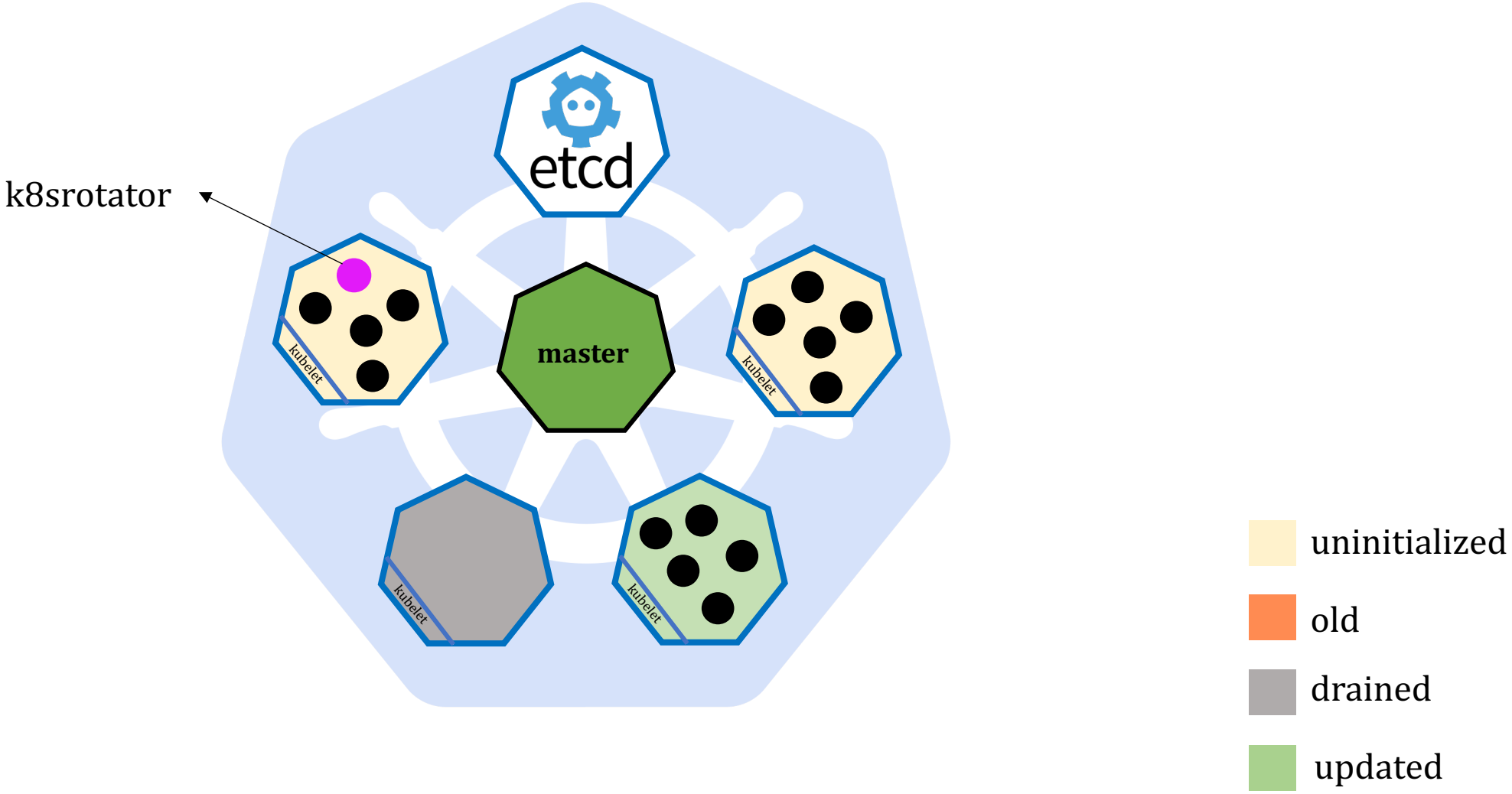
K8srotator: Core functionality



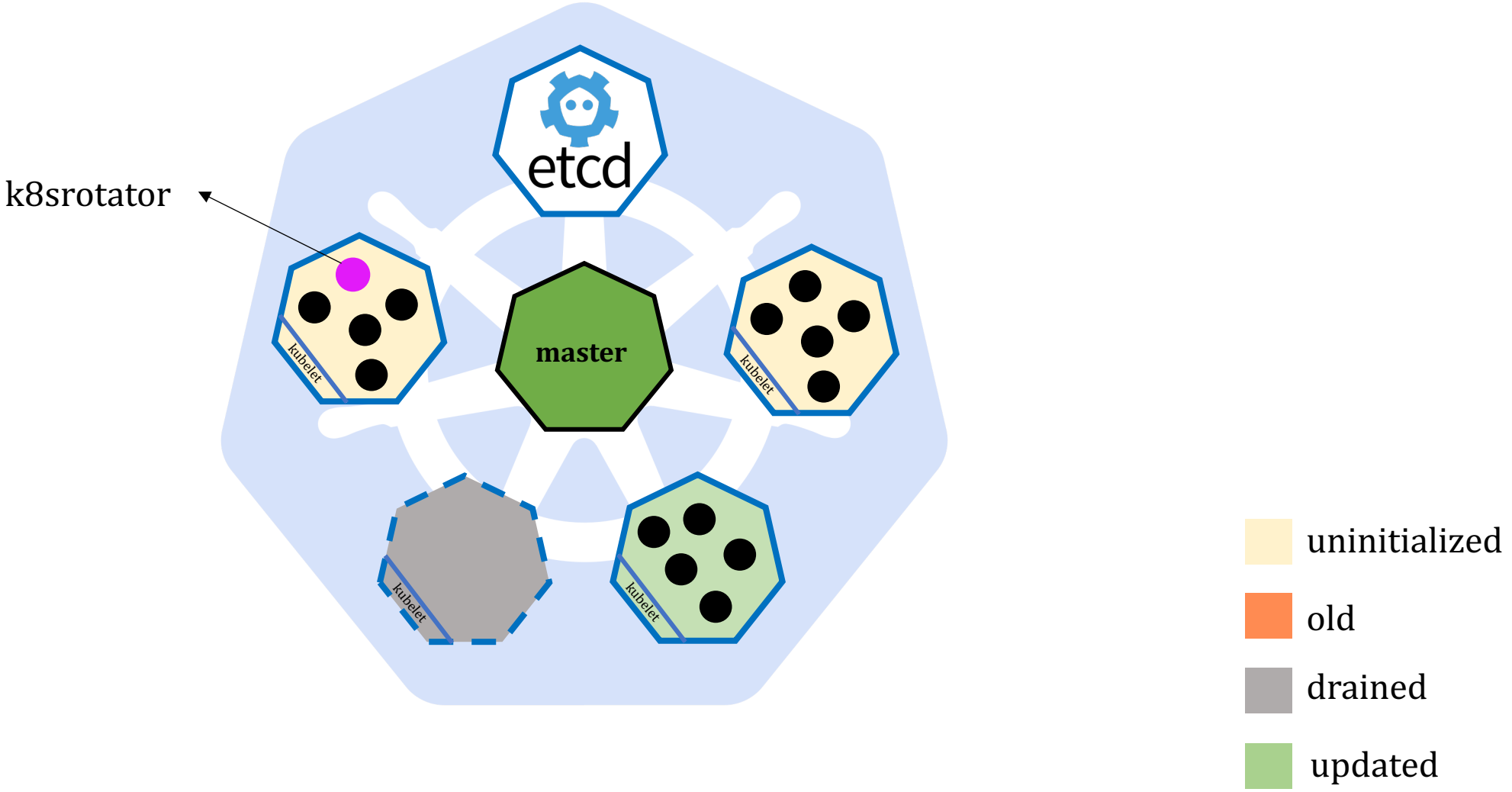
K8srotator: Core functionality



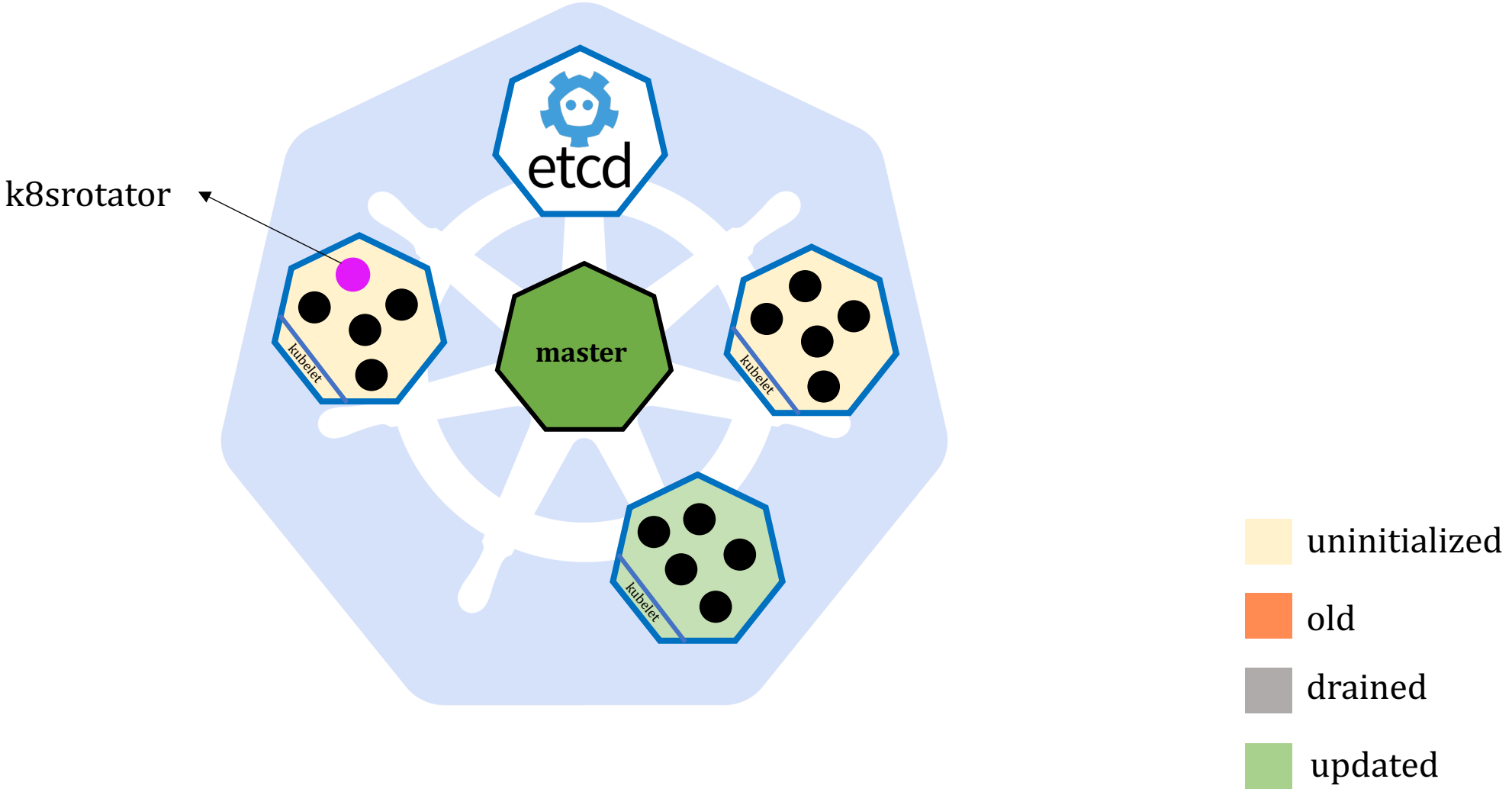
K8srotator: Core functionality



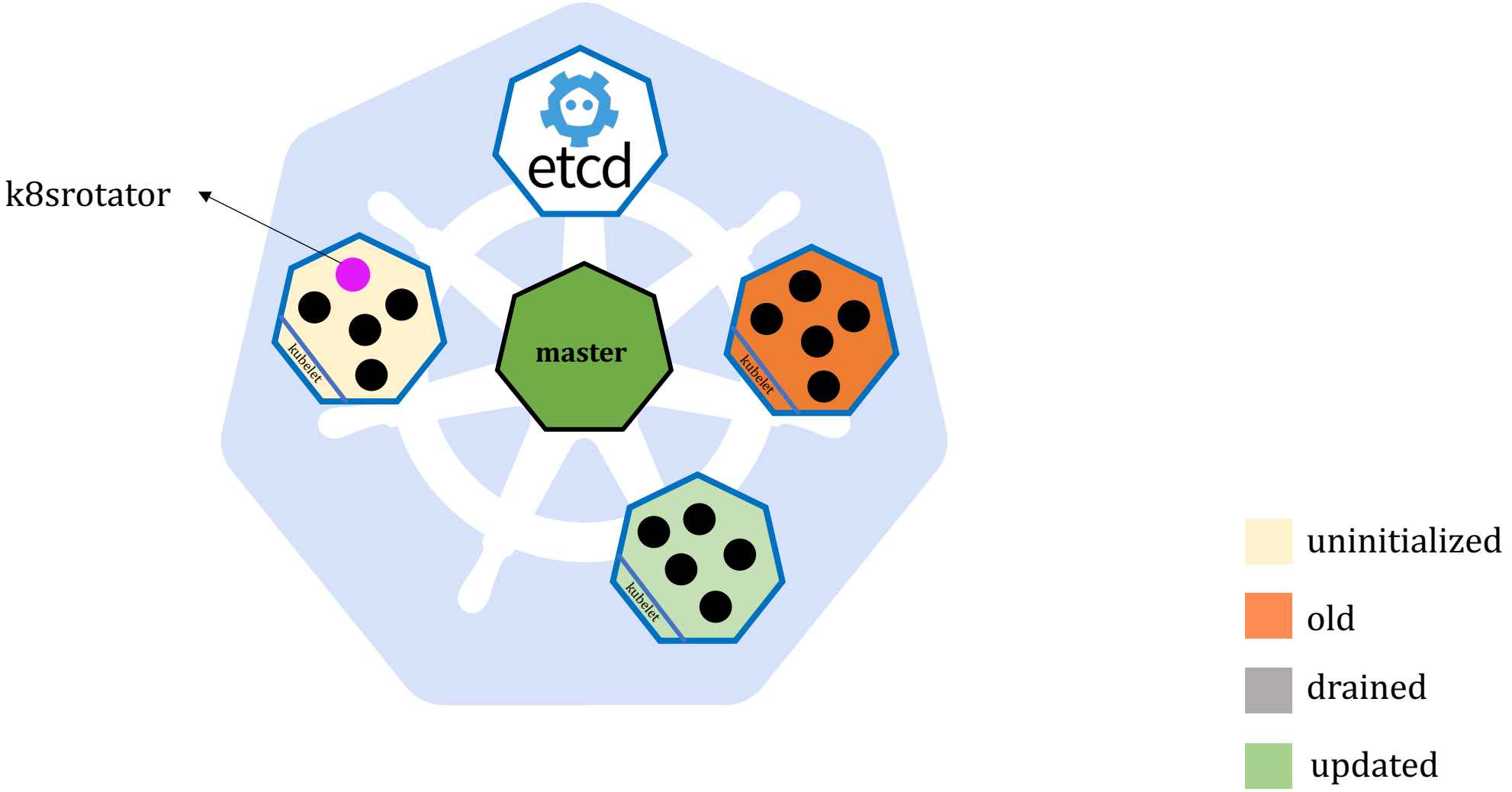
K8srotator: Core functionality



K8srotator: Core functionality



K8srotator: Core functionality



K8srotator: v1

- Extended cluster health
  - Control Plane liveness
  - Stats collector
  - Log collector
- Filter specific nodes
  - Choose type of nodes: etcd, apiserver, worker node
  - Select AZs, region
  - Using Node Labels
- Custom cluster configuration

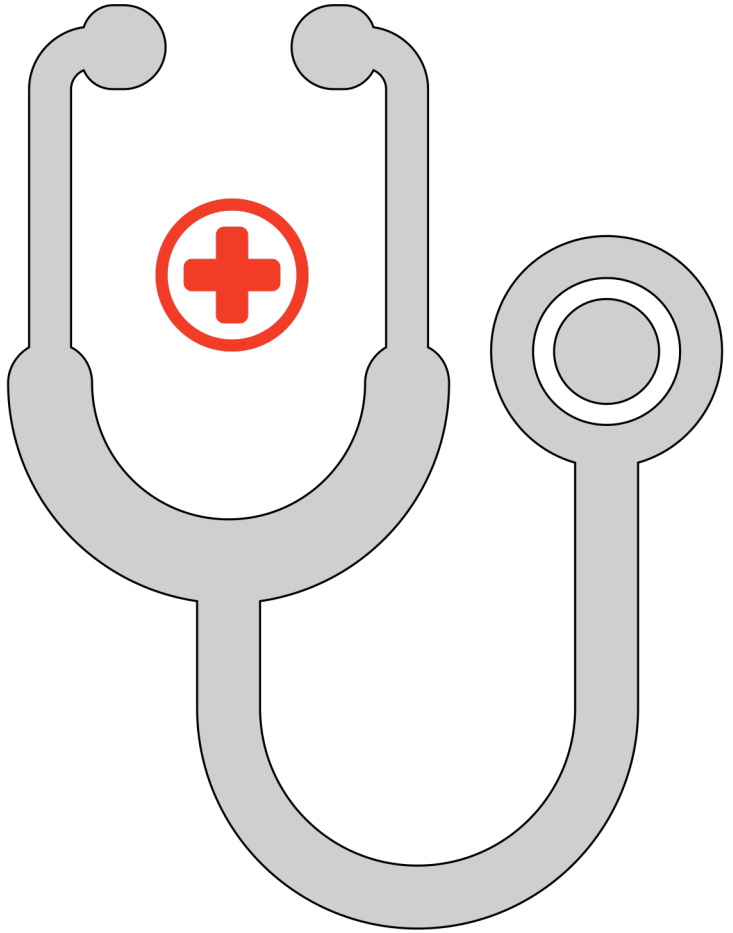
Failure Scenarios to tackle

- etcds loose quorum
- Apiserver is overloaded
- Ingress is overloaded
- AWS AZ Capacity Limits
- Incompatible “in-place” Version upgrade
- AWS API Rate limits

Recovering from failures

- Soft failures
- Hard failures

- Damage control:
  - Apply T_{Freeze}
  - Switch to FREEZE state
  - Alert the user / admin / team
- First Aid:
  - Logs / Stats
  - Remove T_{Freeze}
  - Switch back to ROTATE state



Where we are at today ?

- Standard tool for handling upgrades
- Operational load shrunk
- Used on all clusters
- Still learning…

What we learnt !

- Blanket cordoning of nodes – bad/risky
- Cluster Health definition is living organism
- Keep it simple – abstract complexity
- Failure scenarios are like snowflakes
- Cluster Health
  - is evolving
  - Custom for workloads
- Cluster types need more smoothening

Future Work

- Don't REQUIRE triggers
- Multi cluster
- PodDisruptionBudget
- Extending Cluster Health
- Ordering node termination

Thank you !

We’re hiring !