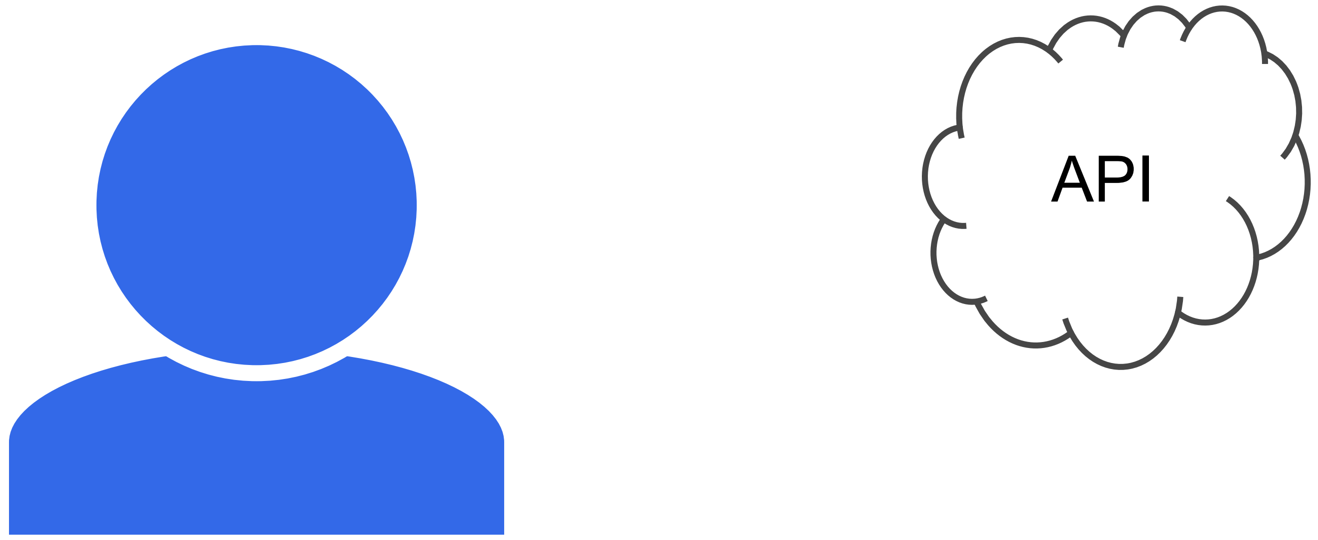
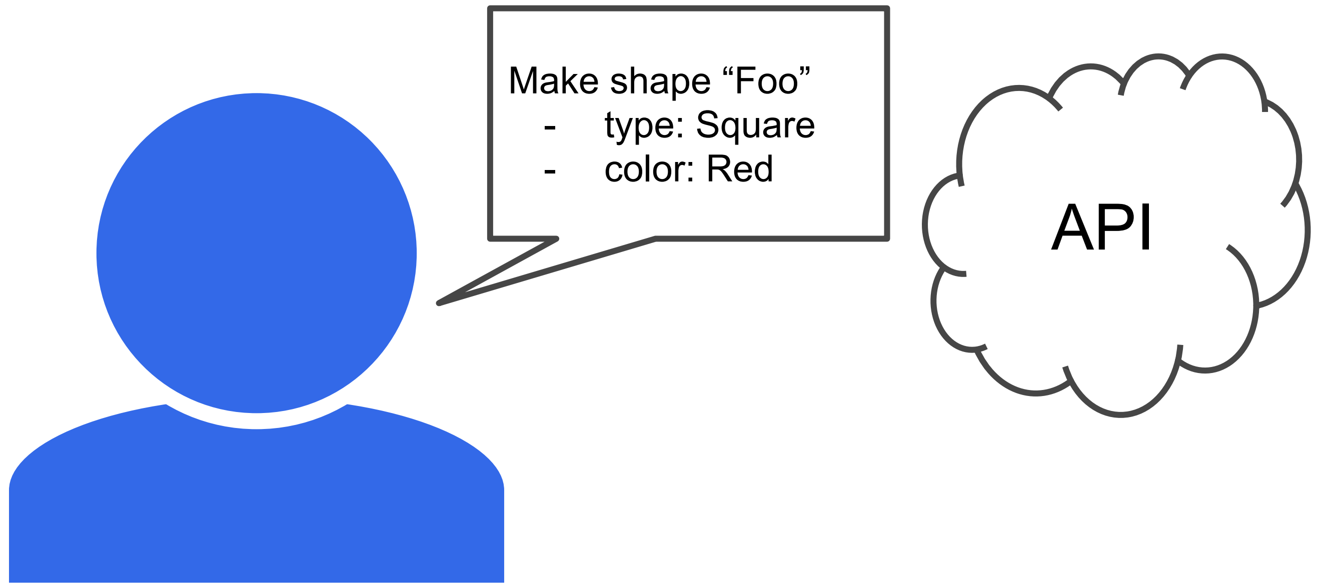
Kubernetes: What is "reconciliation"?

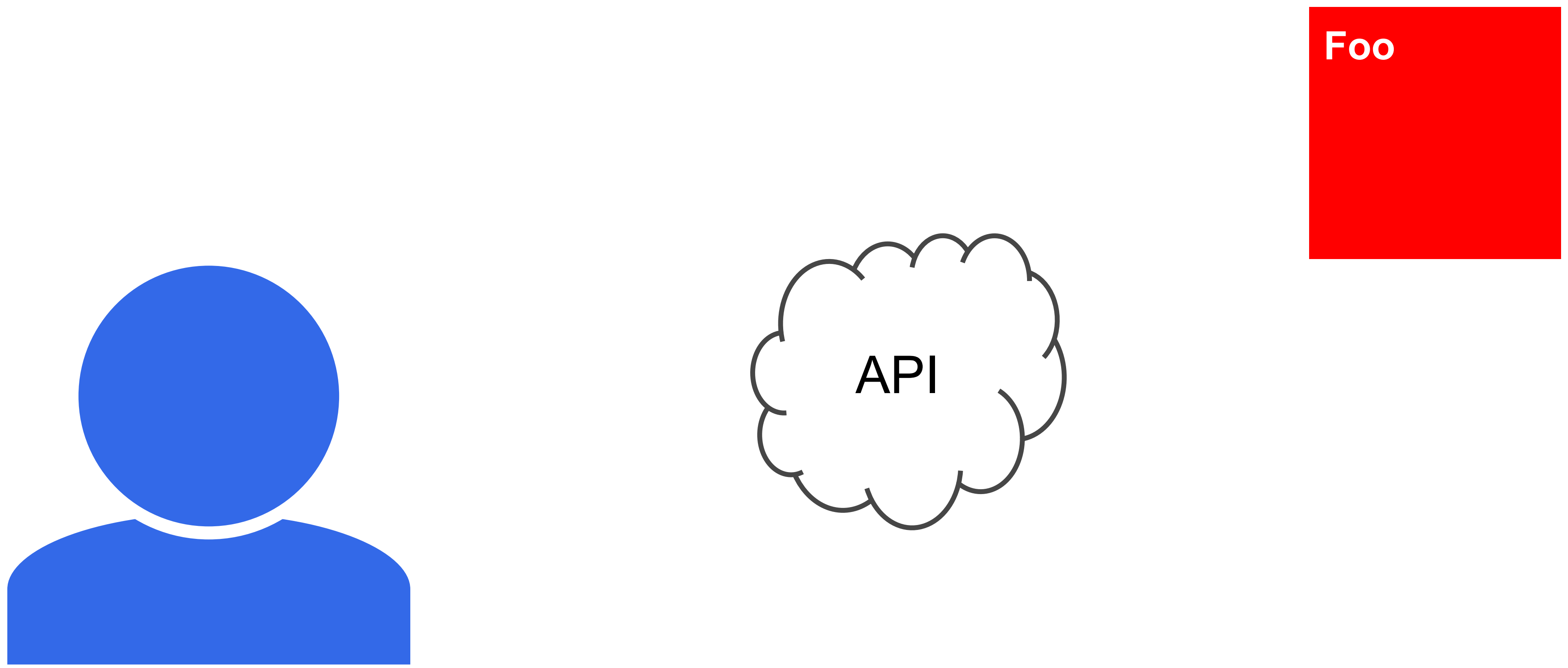
Tim Hockin <thockin@google.com>
@thockin

Assume there's a cloud API to make shapes.

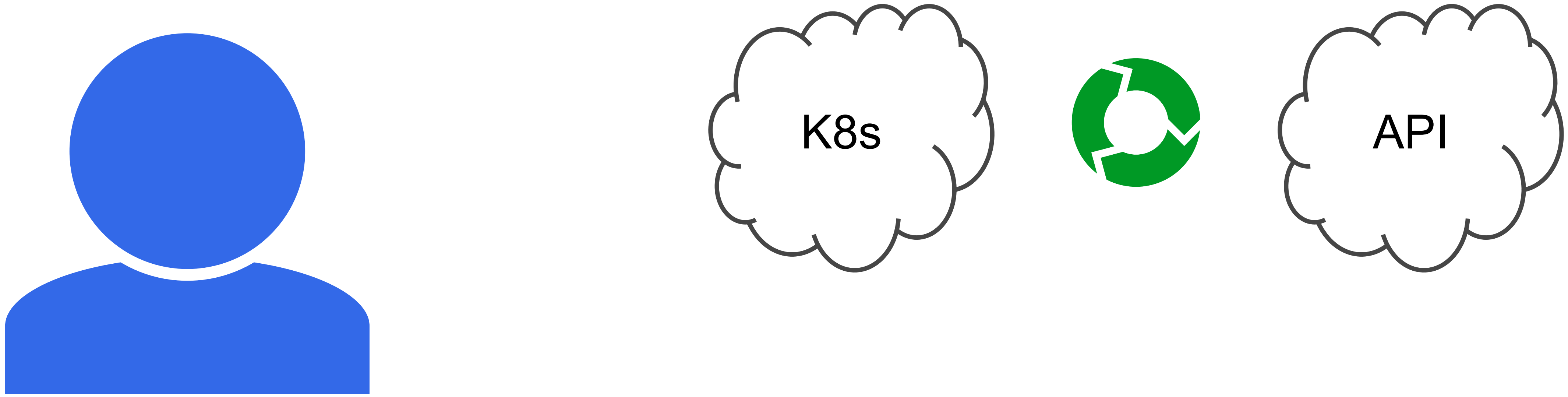
Why shapes? It's just concrete enough to reason about, while not getting stuck in the details.

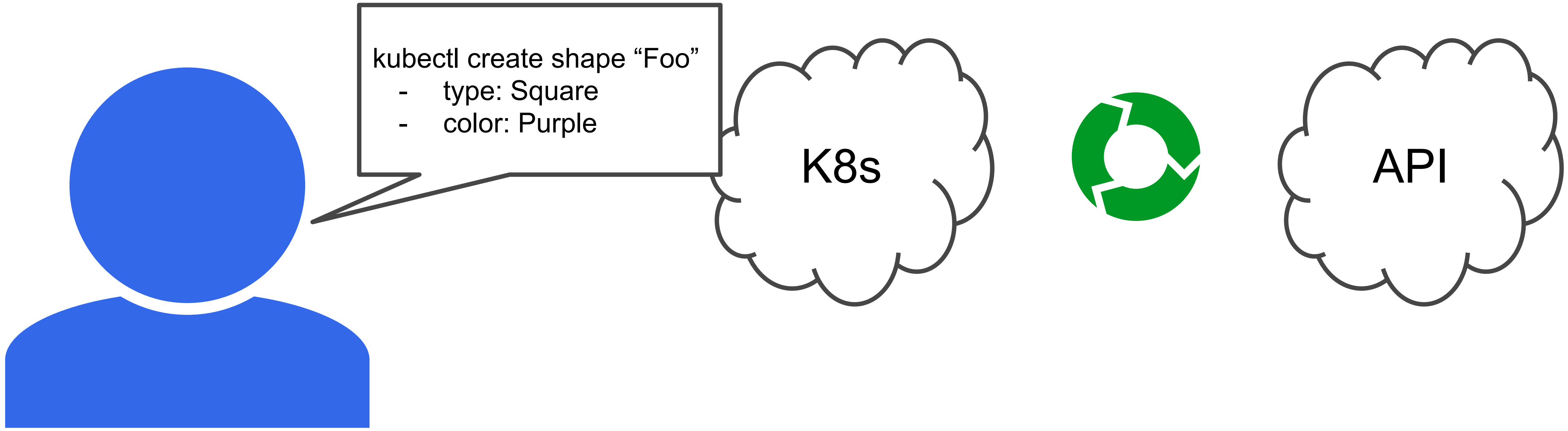

API


Make shape “Foo”
- type: Square
- color: Red
API

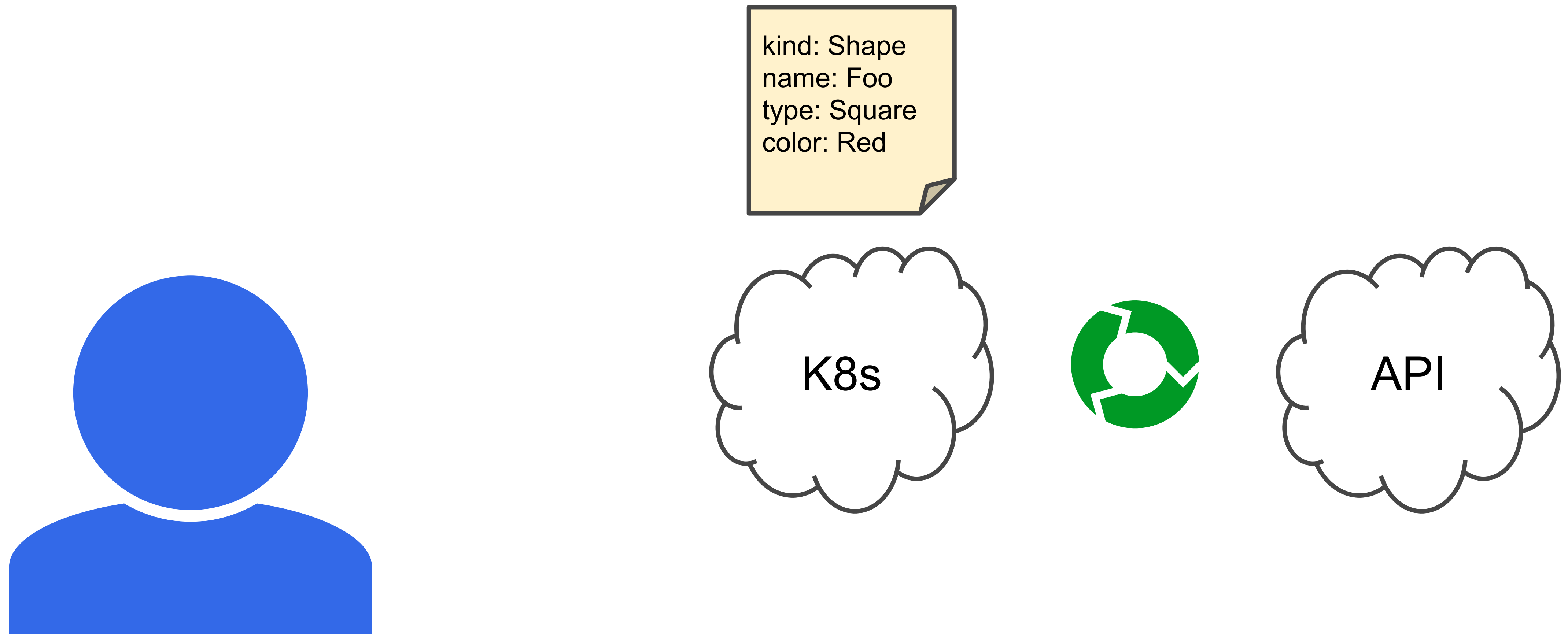

API
Foo

This API is fine, but I want to wrap it into a declarative system (e.g. Kubernetes)

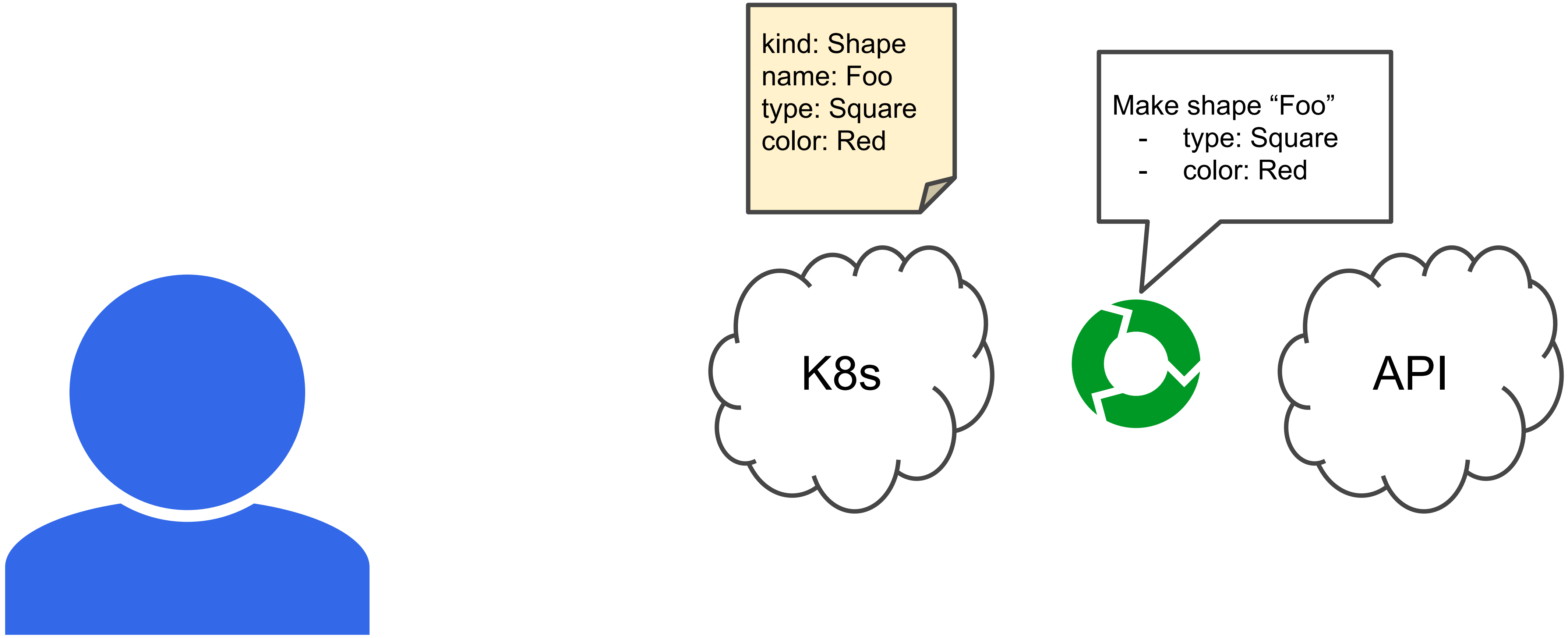

K8s
API

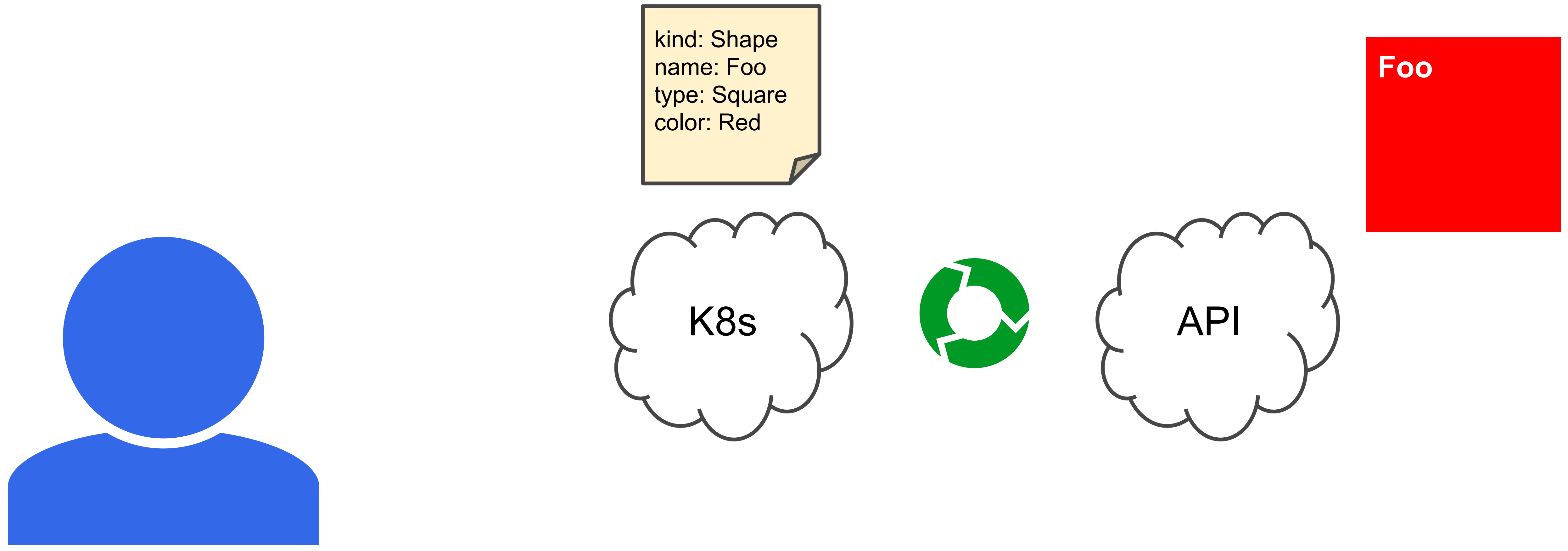


```
kubectl create shape "Foo"
  -   type: Square
  -   color: Purple
```

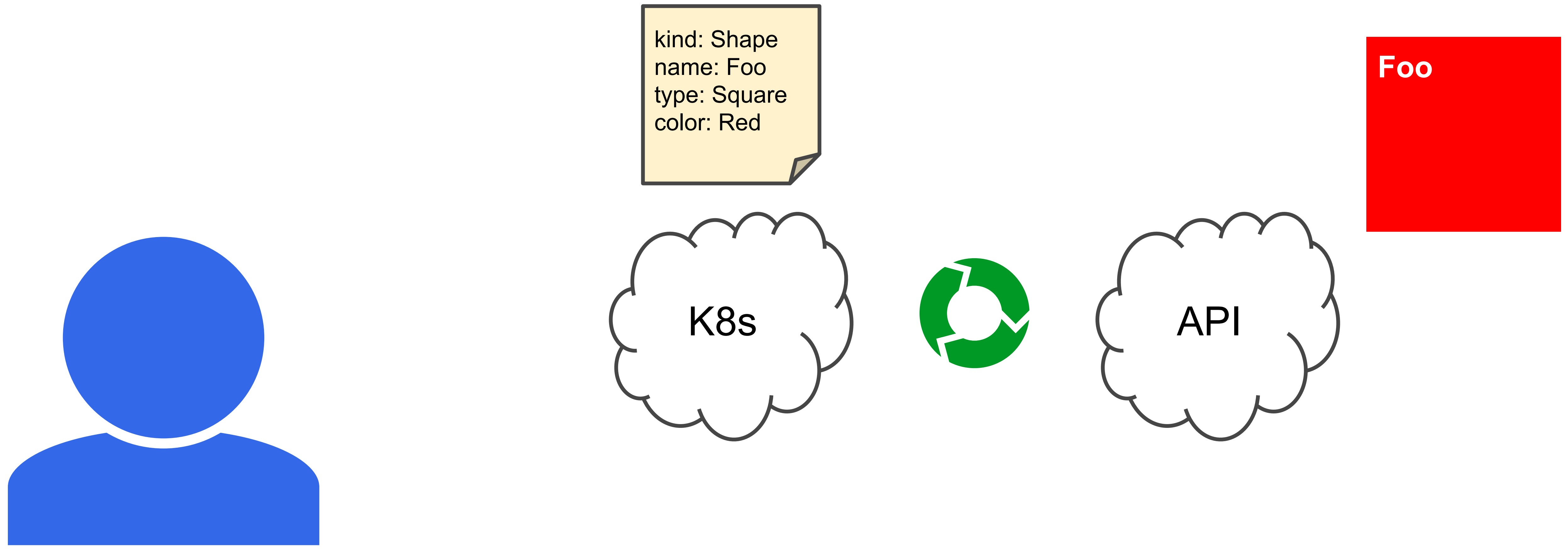


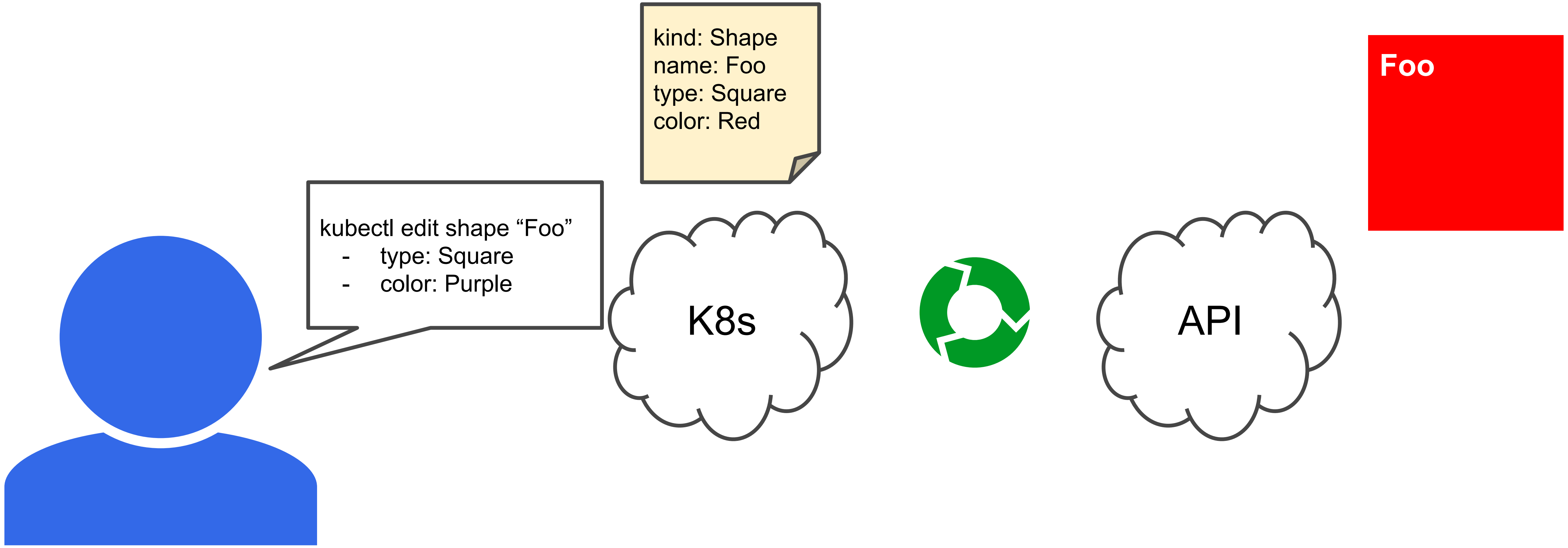
```
kind: Shape
name: Foo
type: Square
color: Red
```

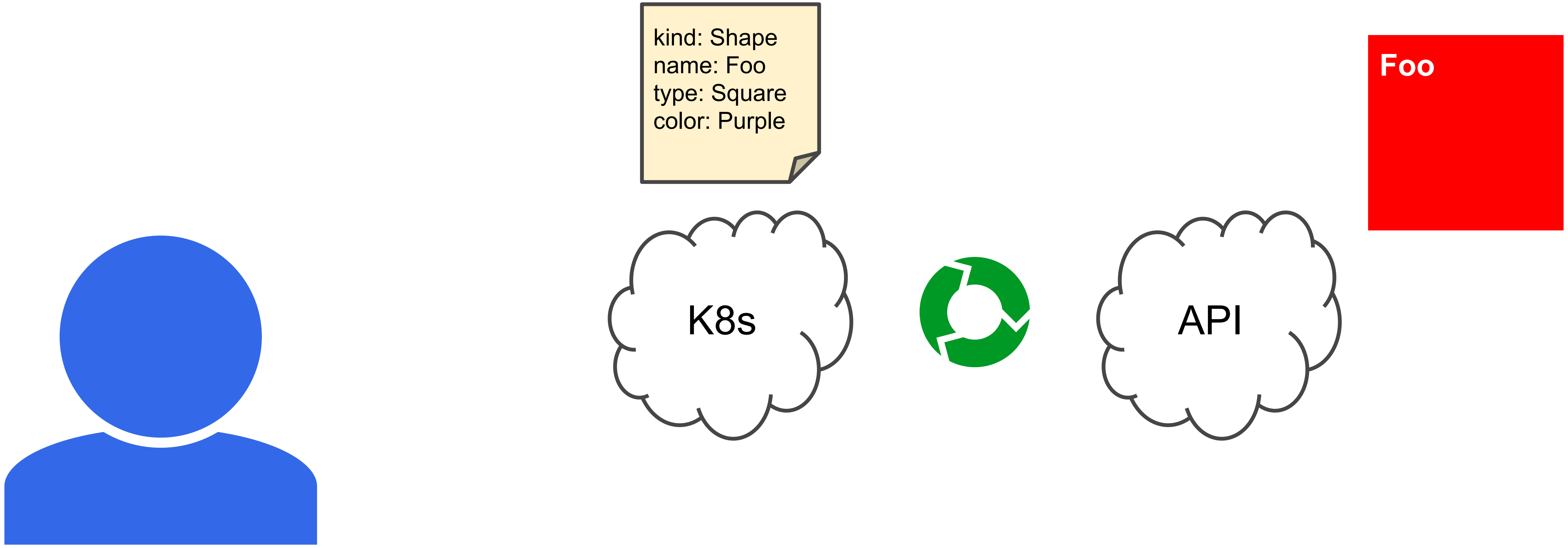

kind: Shape
name: Foo
type: Square
color: Red
Make shape "Foo"
- type: Square
- color: Red
K8s
API

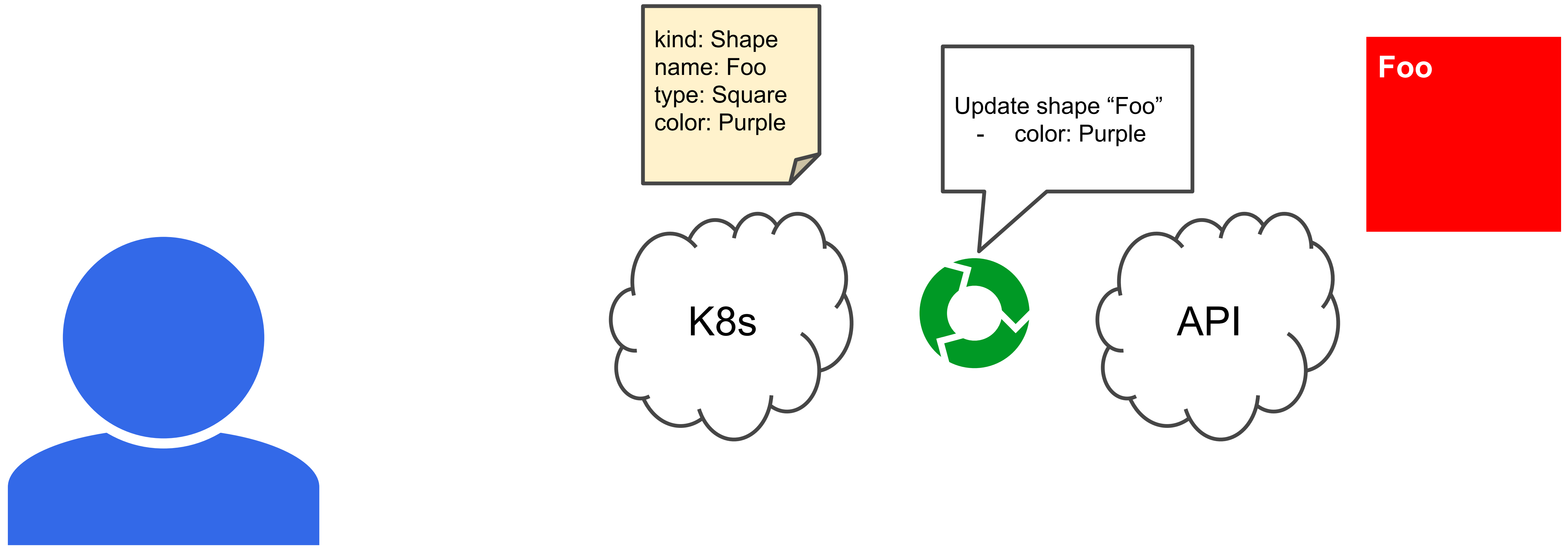

kind: Shape
name: Foo
type: Square
color: Red
K8s
API
Foo

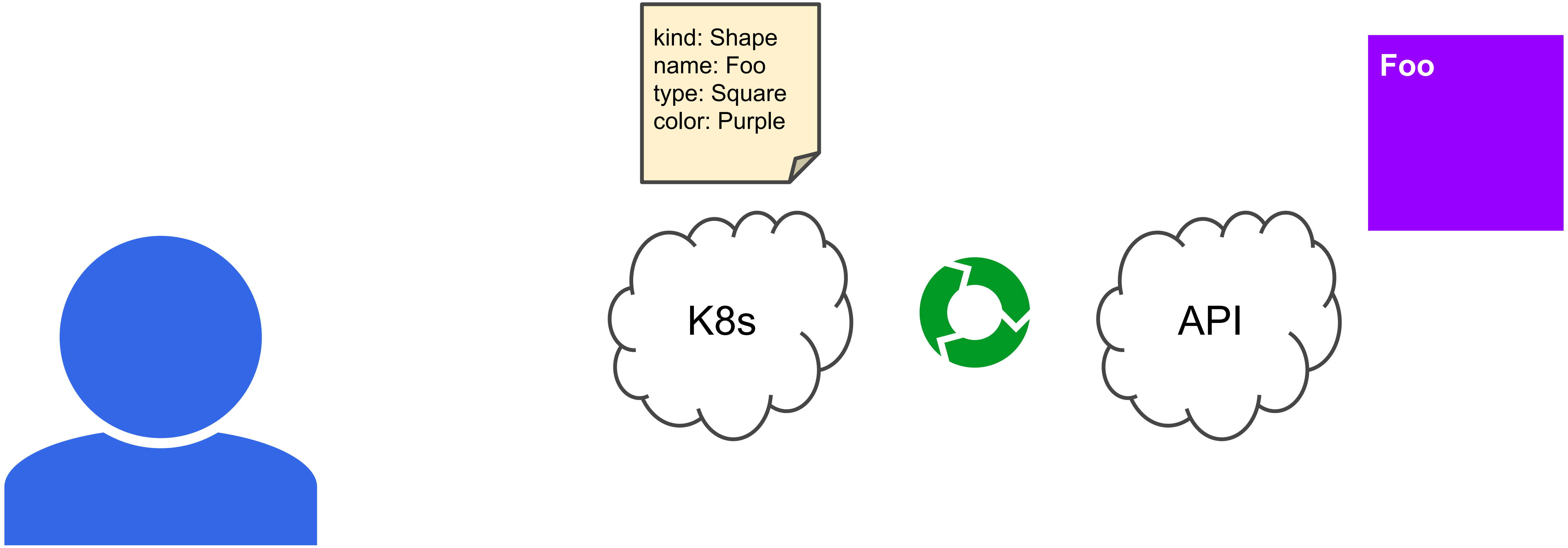
The controller will keep my Kubernetes object in sync with the underlying API


kind: Shape
name: Foo
type: Square
color: Red
K8s
API
Foo


kind: Shape
name: Foo
type: Square
color: Red
kubectl edit shape “Foo”
- type: Square
- color: Purple
K8s
API
Foo


kind: Shape
name: Foo
type: Square
color: Purple
K8s
API
Foo

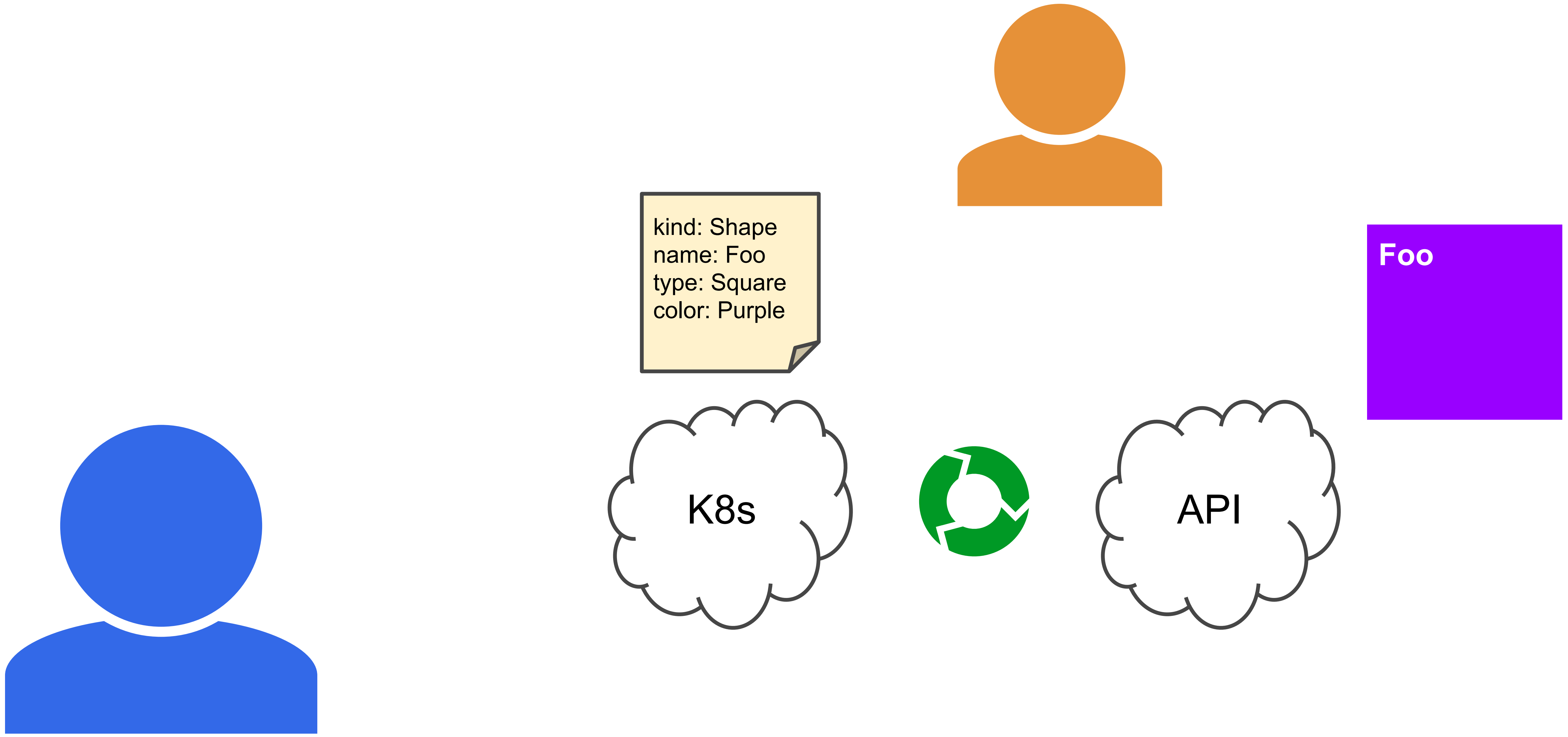

kind: Shape
name: Foo
type: Square
color: Purple
Update shape “Foo”
- color: Purple
K8s
API
Foo

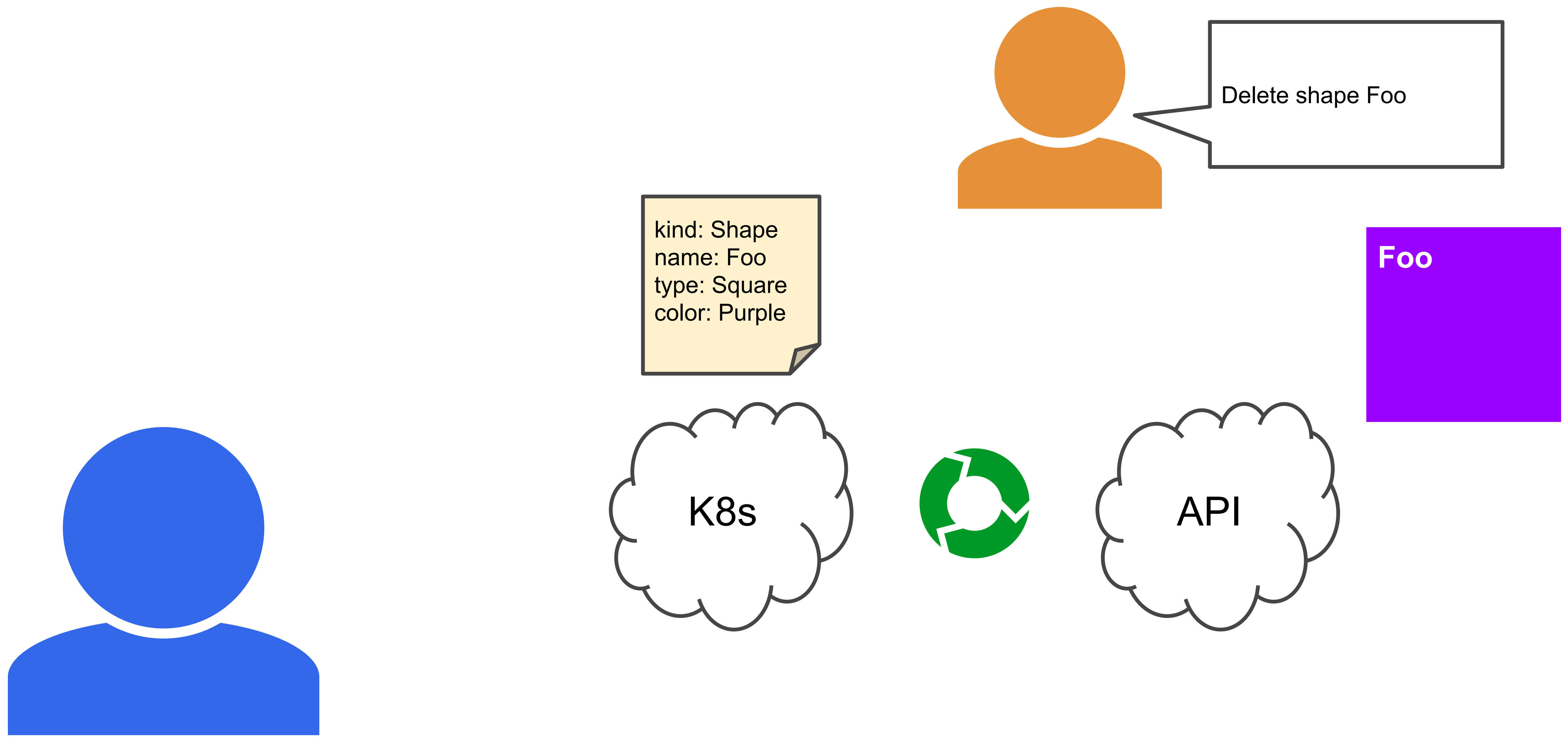


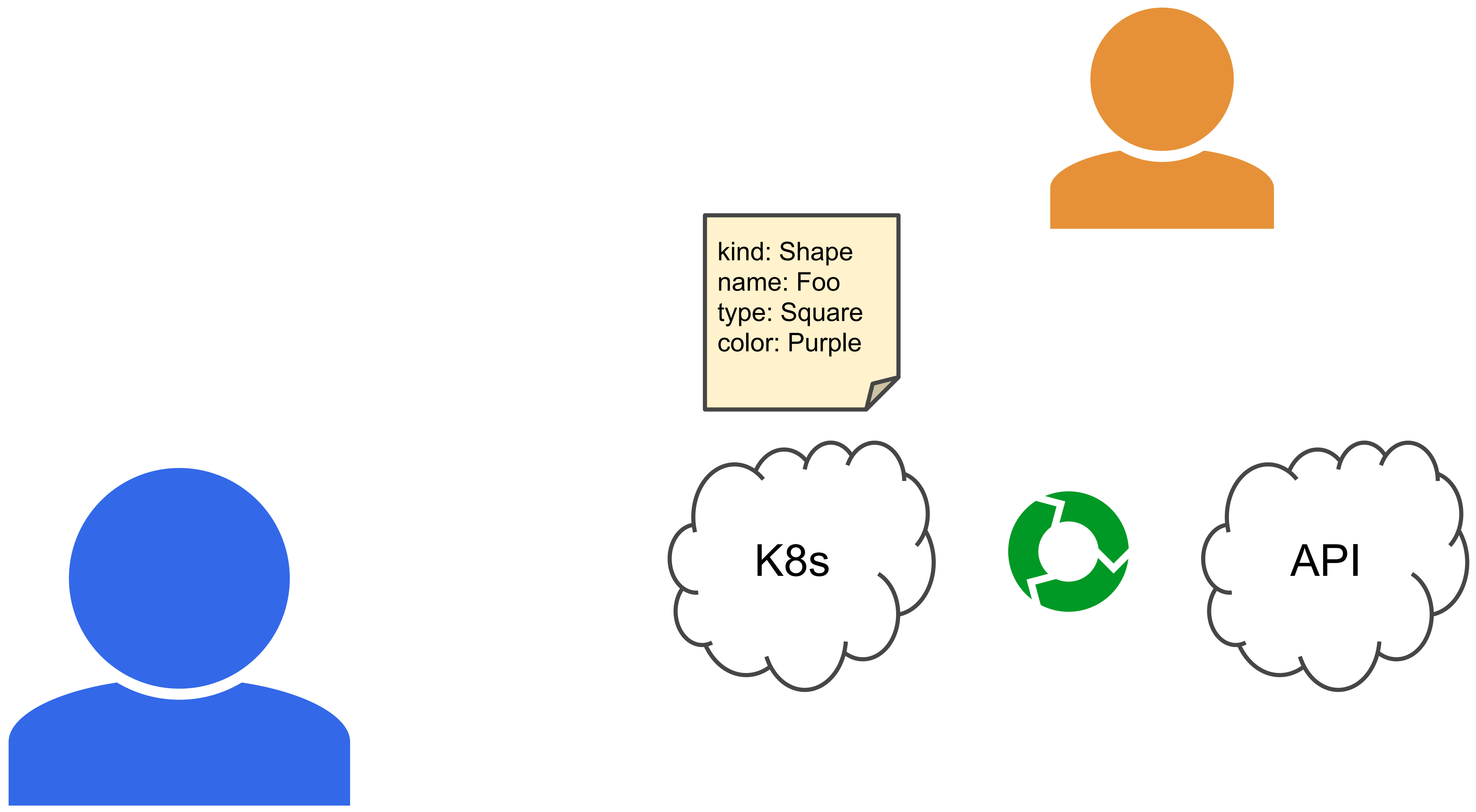
kind: Shape
name: Foo
type: Square
color: Purple

This is what we call “reconciliation”. Specifically, this is **uni-directional reconciliation**.

What happens if...


kind: Shape
name: Foo
type: Square
color: Purple
K8s
API
Foo


Delete shape Foo
kind: Shape
name: Foo
type: Square
color: Purple
Foo
K8s
API

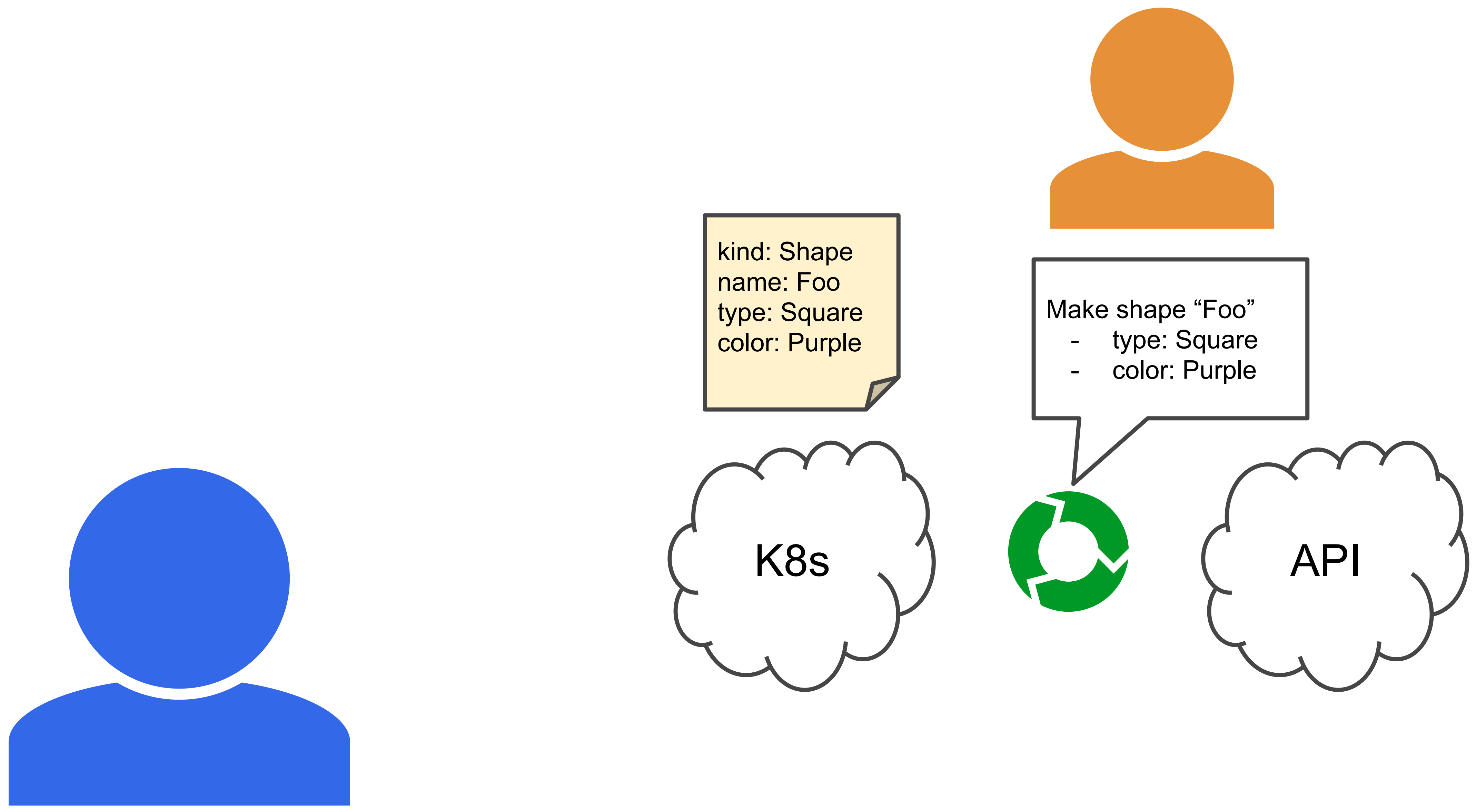


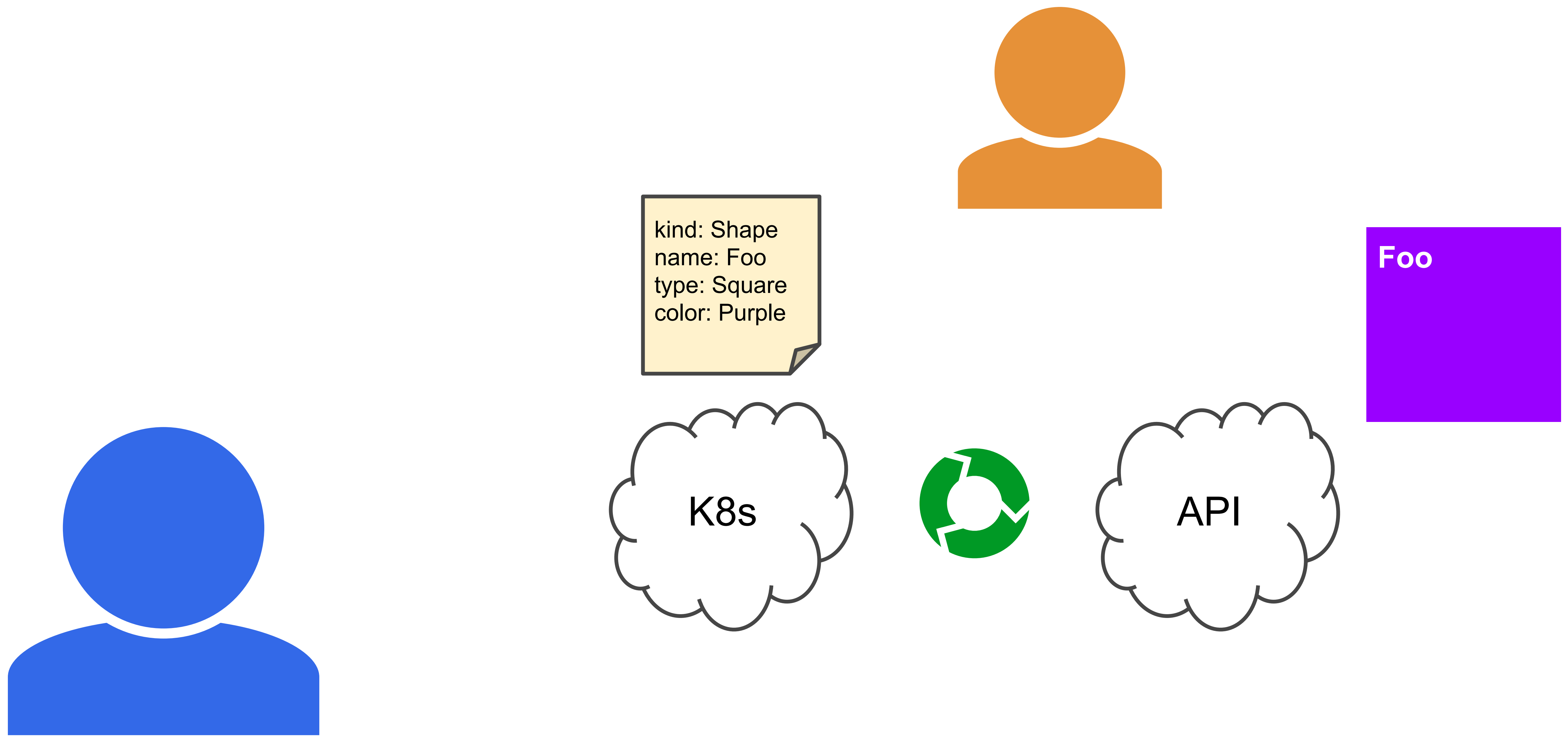
kind: Shape
name: Foo
type: Square
color: Purple

The shape I wanted has inadvertently been removed by a human or other system.

If we only reconcile in one direction, we will never fix it!

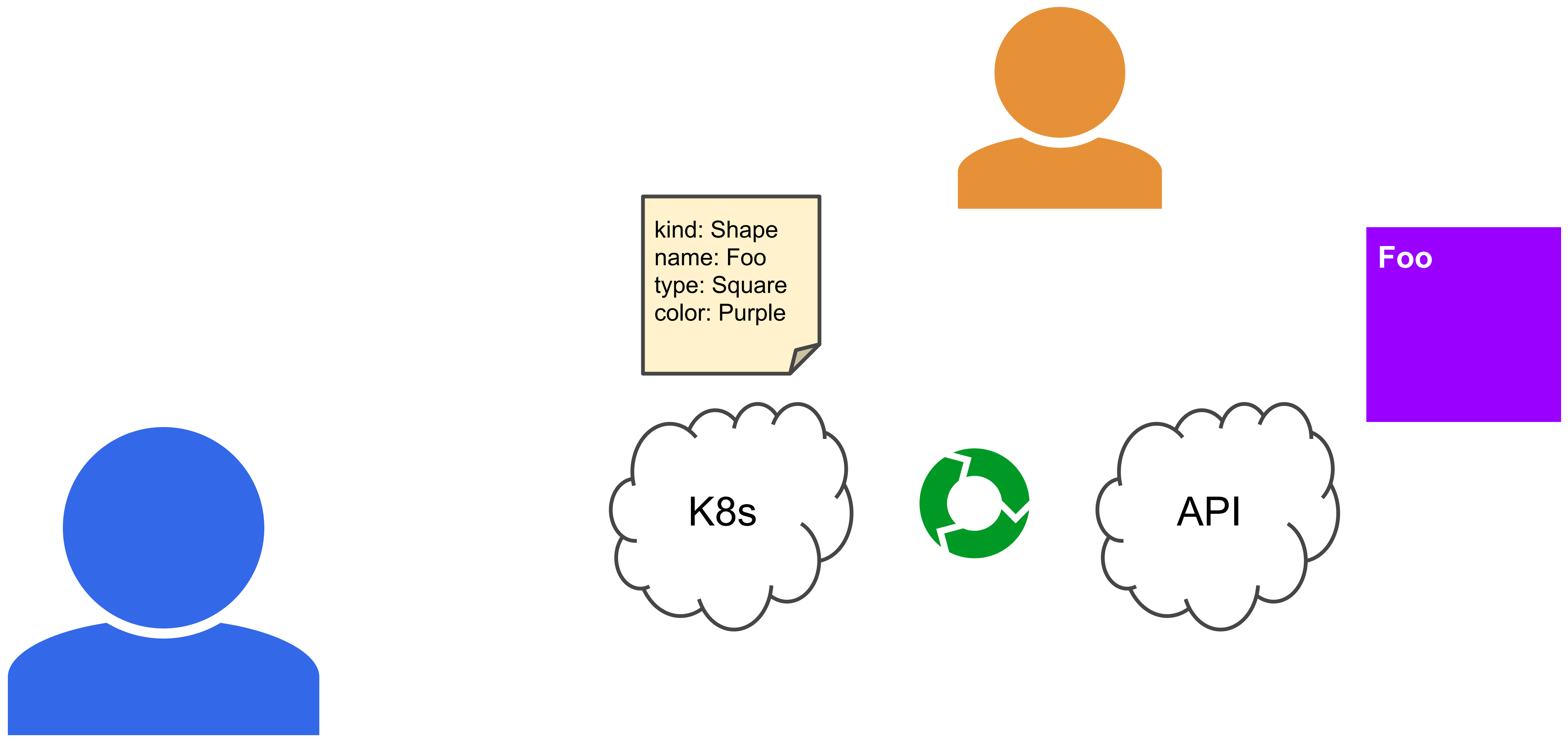
We need to observe that the underlying state has changed and re-assert the state we want.

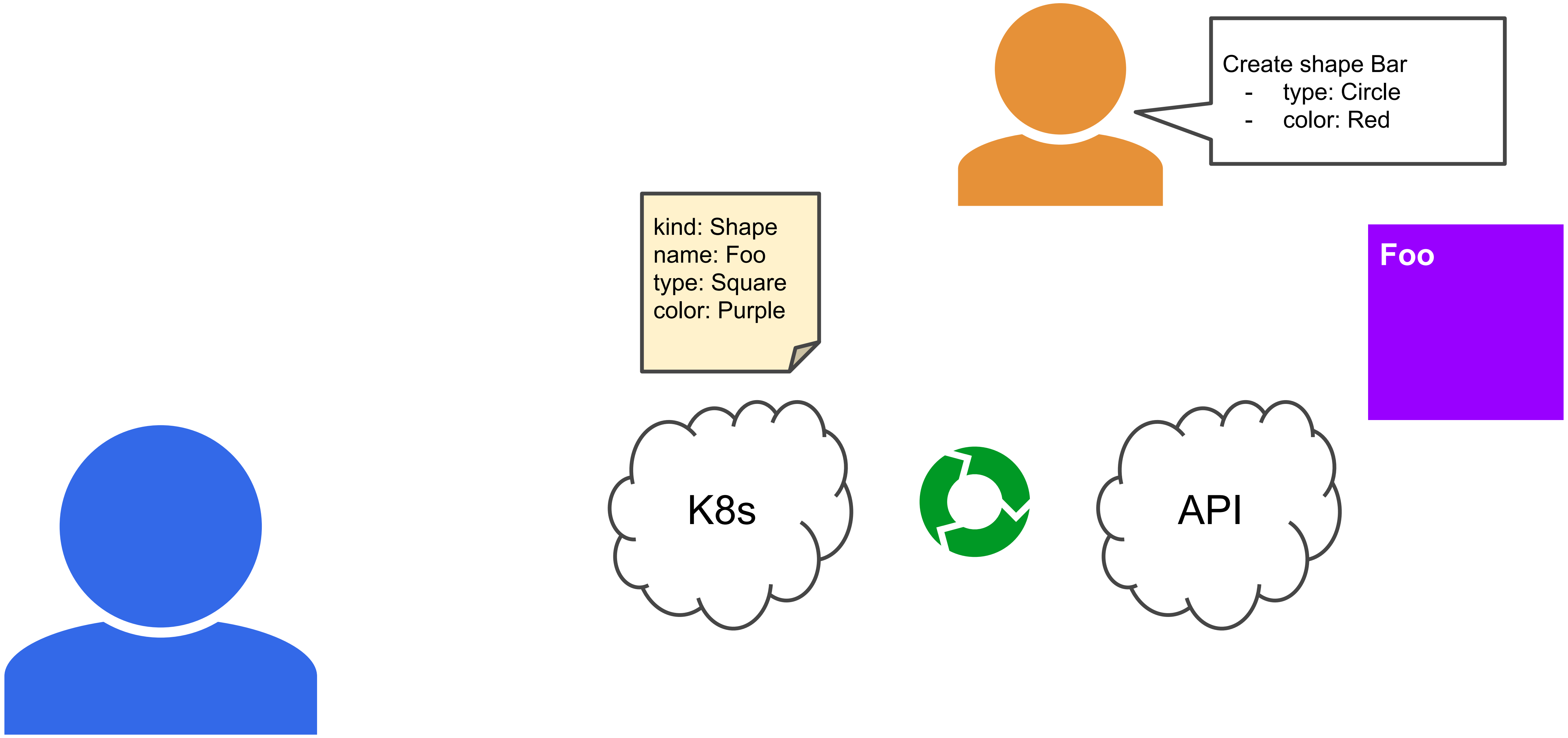

kind: Shape
name: Foo
type: Square
color: Purple
Make shape “Foo”
- type: Square
- color: Purple
K8s
API

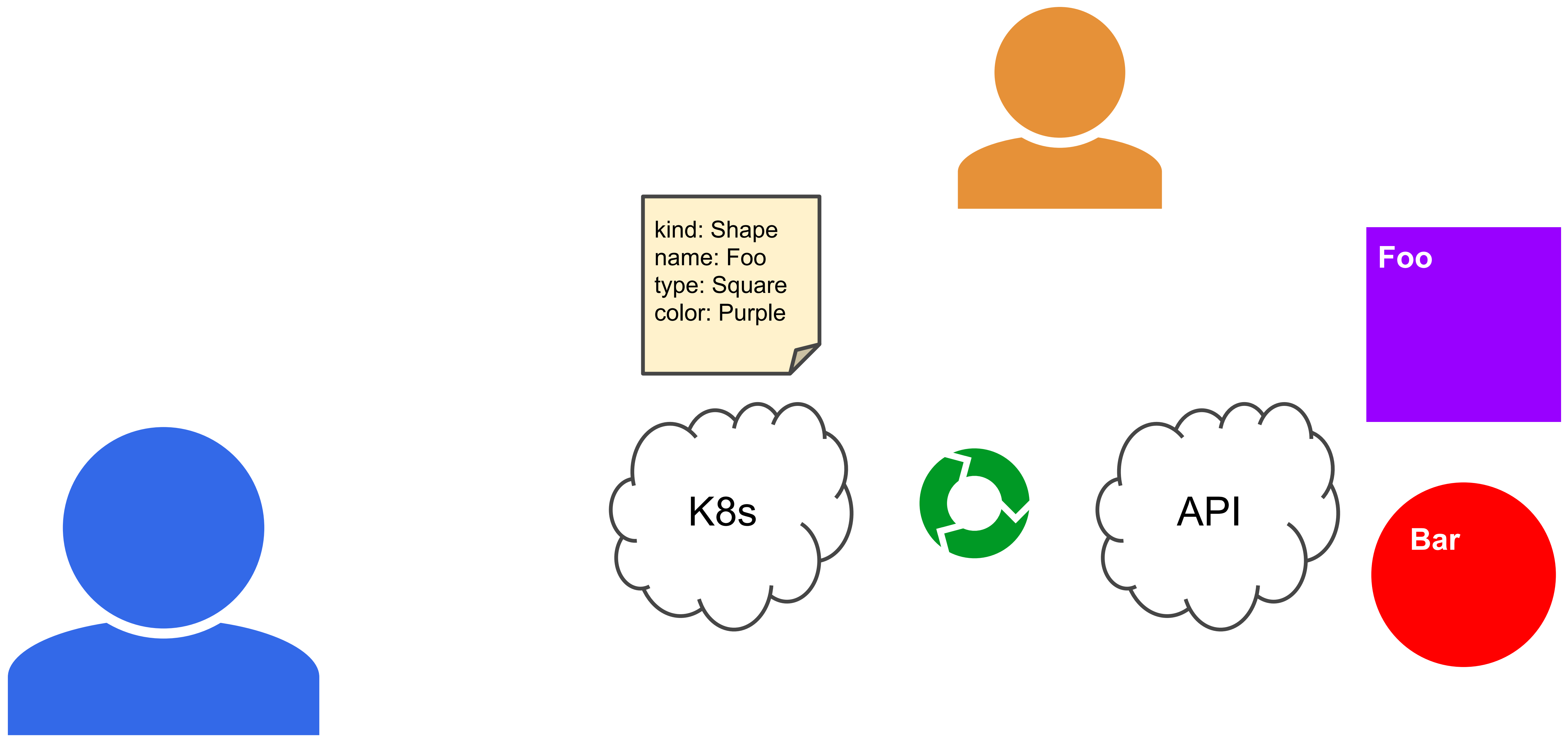

kind: Shape
name: Foo
type: Square
color: Purple
K8s
API
Foo

We usually call this **bi-directional reconciliation.**

But it gets funkier. What if...


kind: Shape
name: Foo
type: Square
color: Purple
K8s
API
Foo


Create shape Bar
- type: Circle
- color: Red
kind: Shape
name: Foo
type: Square
color: Purple
Foo
K8s
API



kind: Shape
name: Foo
type: Square
color: Purple
K8s
API
Foo
Bar

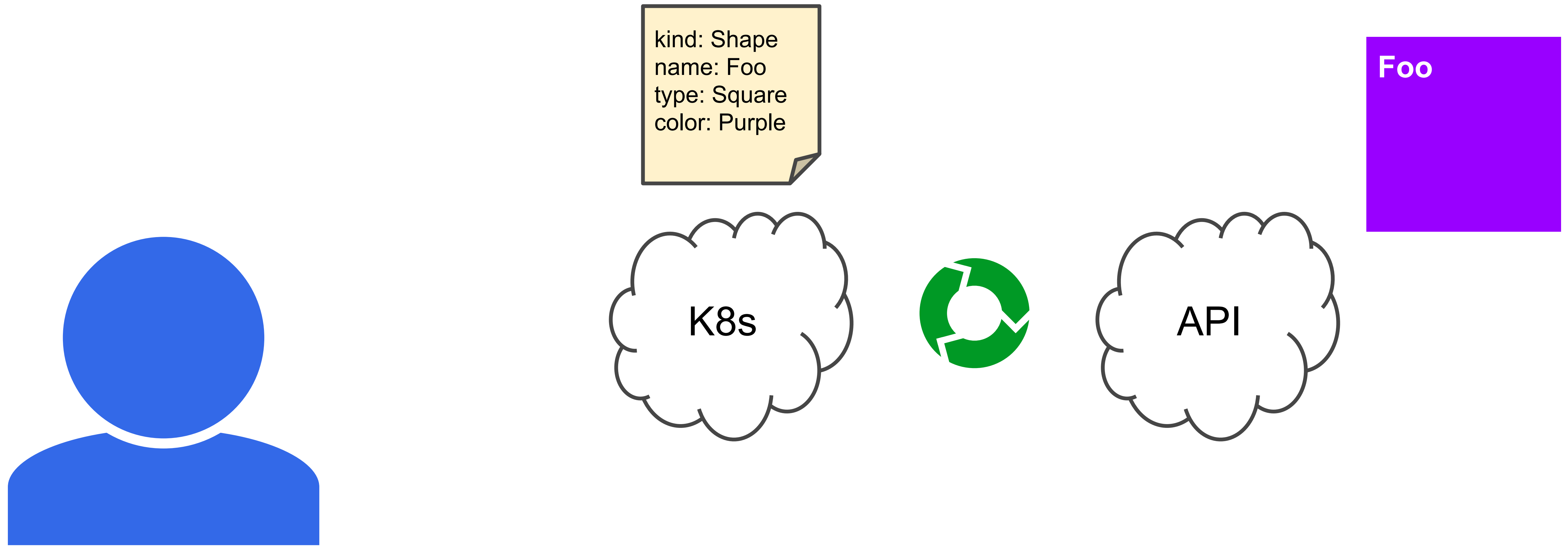
What should the controller do?

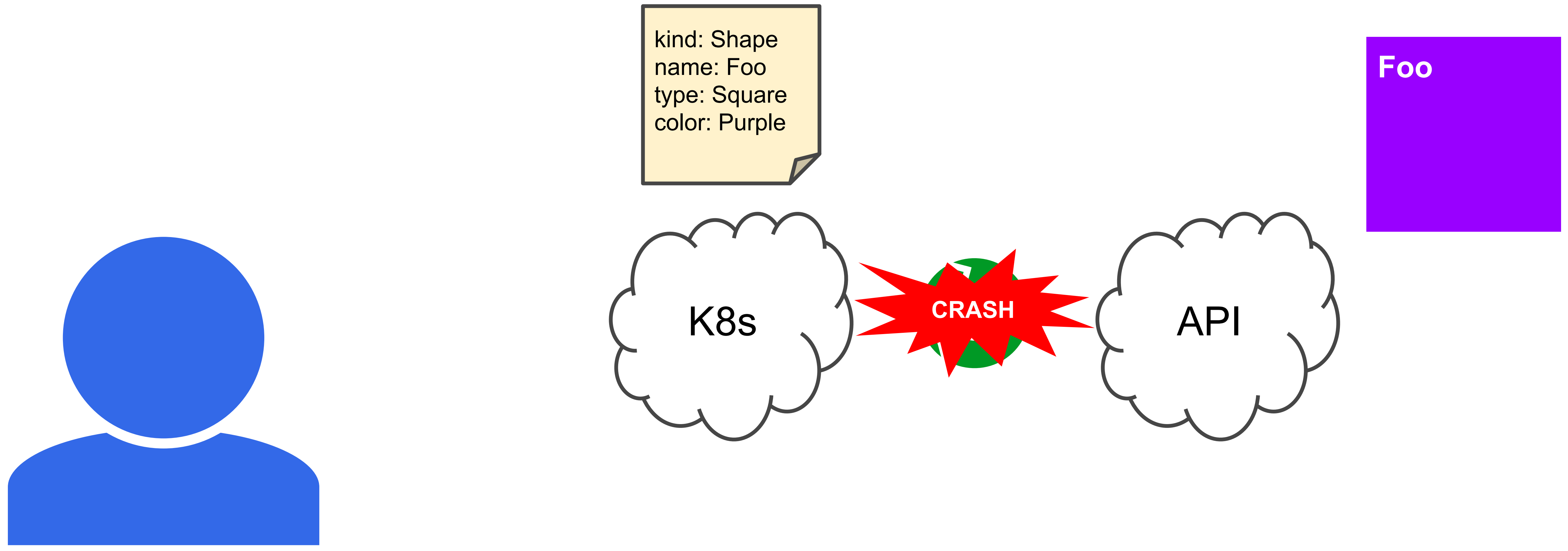
Does it expect to have exclusive use of all shapes? If so, clean up!

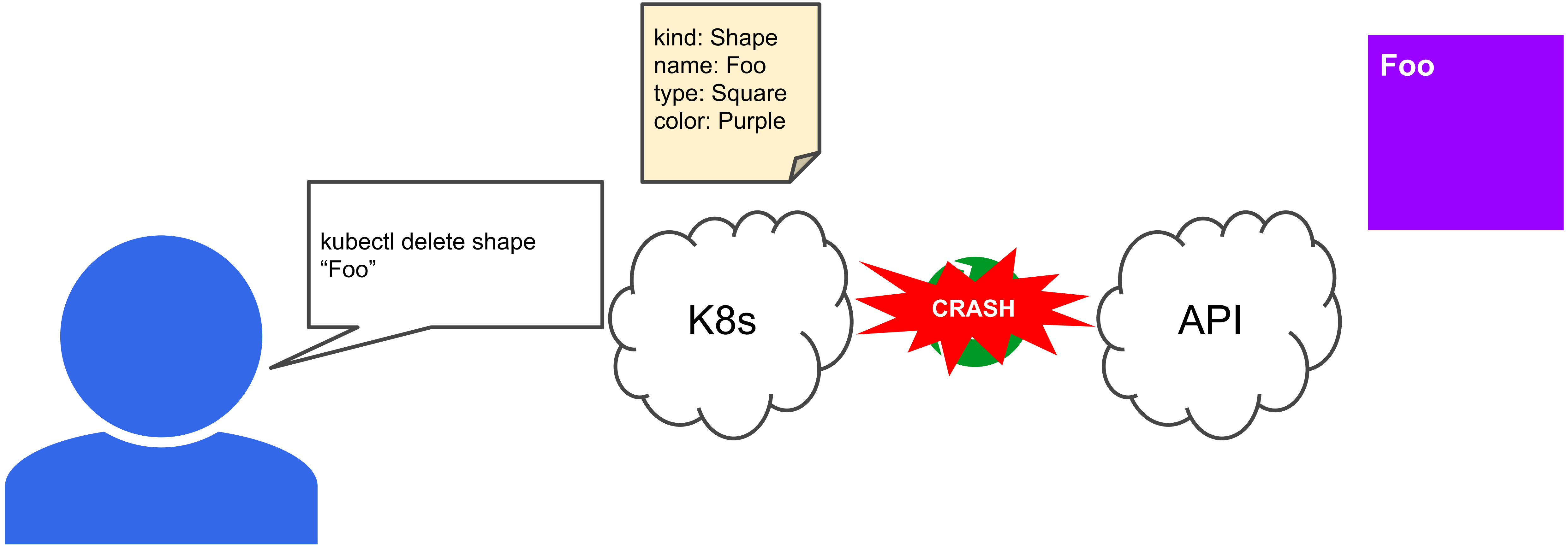
Does it expect to share the shapes API with other users? If so, leave it alone!

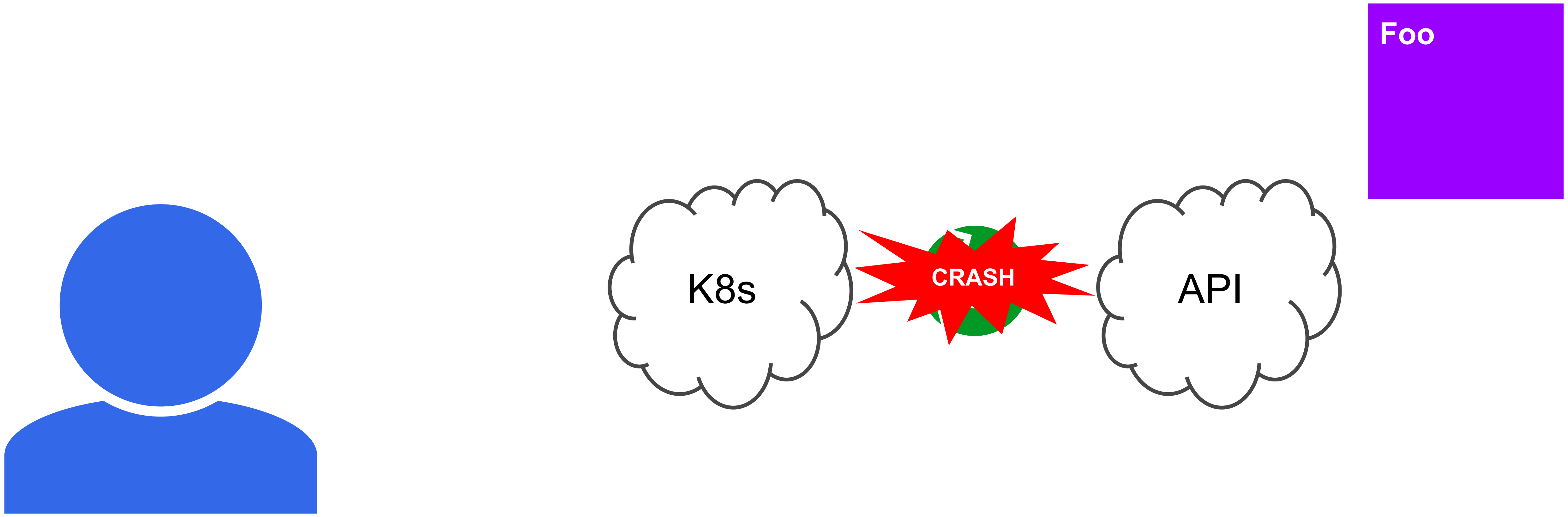
Right?

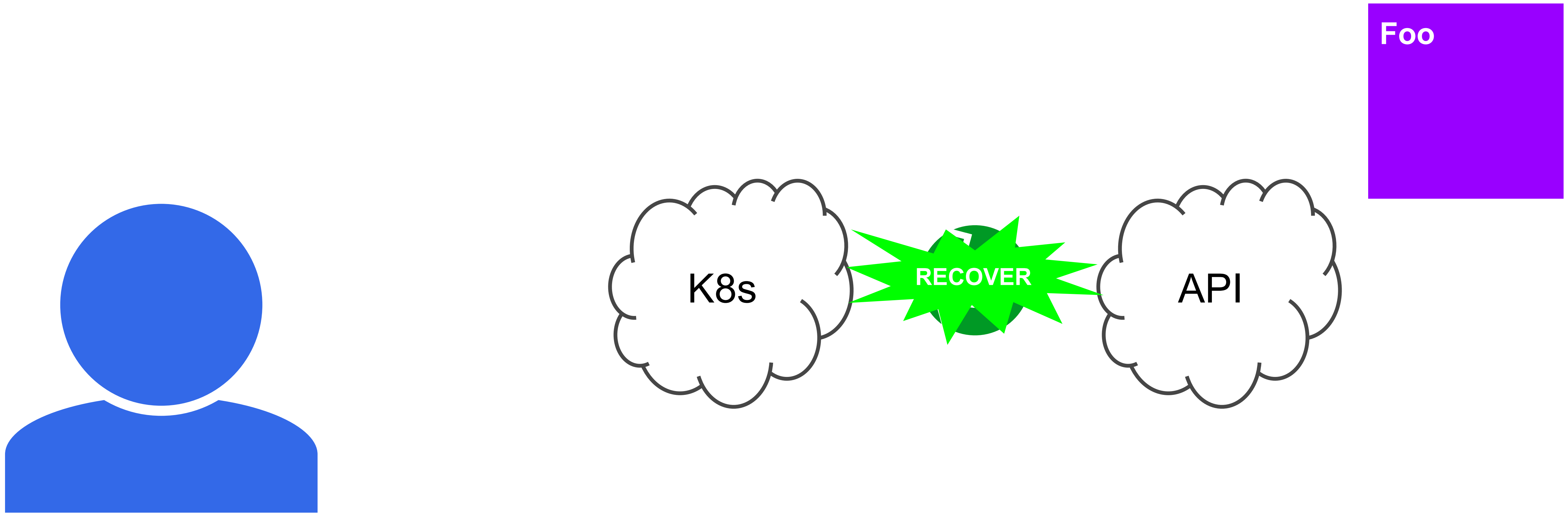
I said it gets **funkier**. What if...

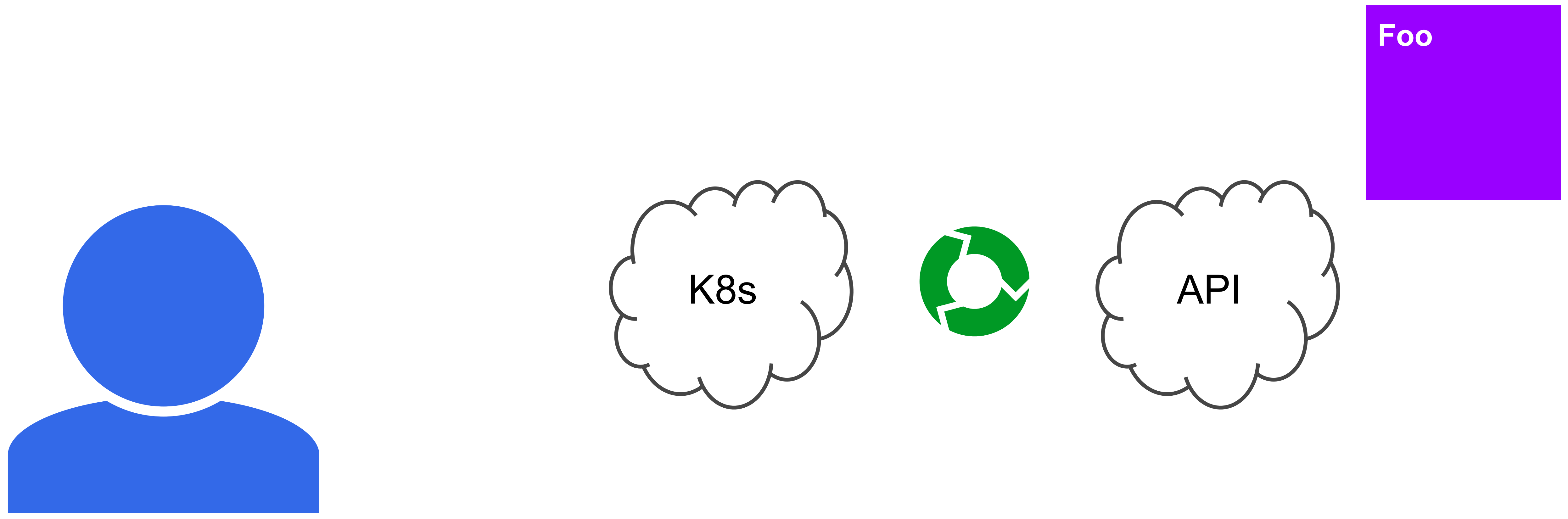

kind: Shape
name: Foo
type: Square
color: Purple
K8s
API
Foo


kind: Shape
name: Foo
type: Square
color: Purple
K8s
CRASH
API
Foo


kubectl delete shape "Foo"
kind: Shape
name: Foo
type: Square
color: Purple
K8s
CRASH
API
Foo


Foo
K8s
CRASH
API


Foo
K8s
RECOVER
API


Foo
K8s
API

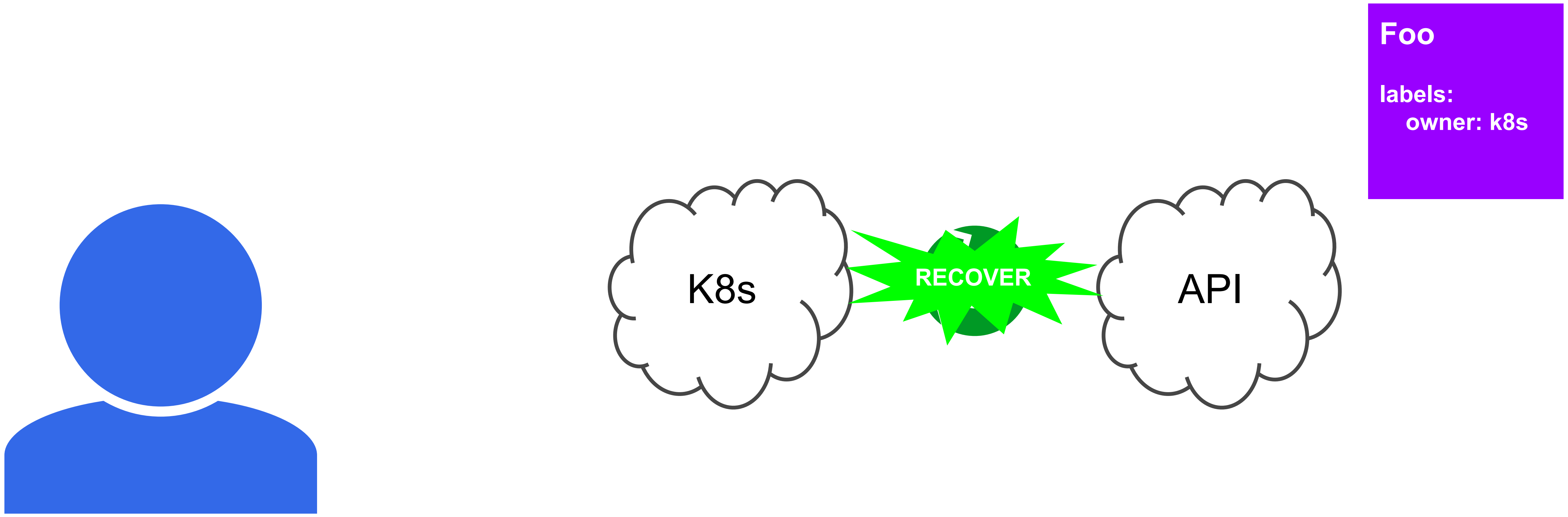
The controller **missed the deletion** of the shape, but as we saw earlier, it ignored things it doesn’t know.

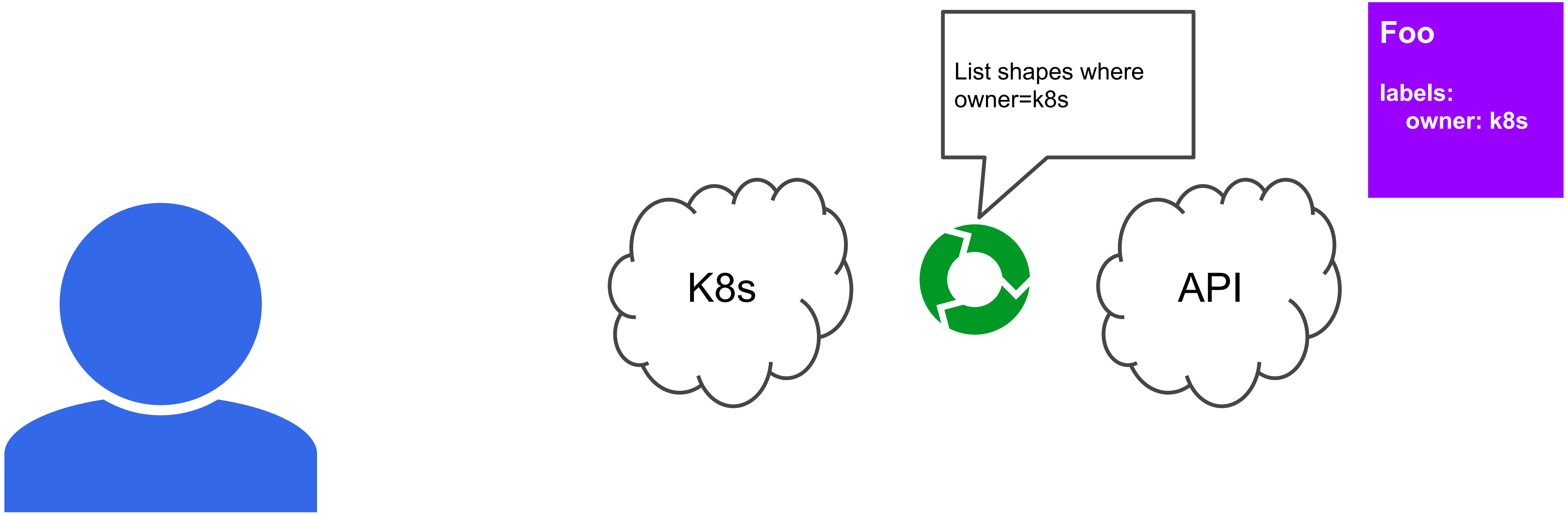
This is a **LEAK**!

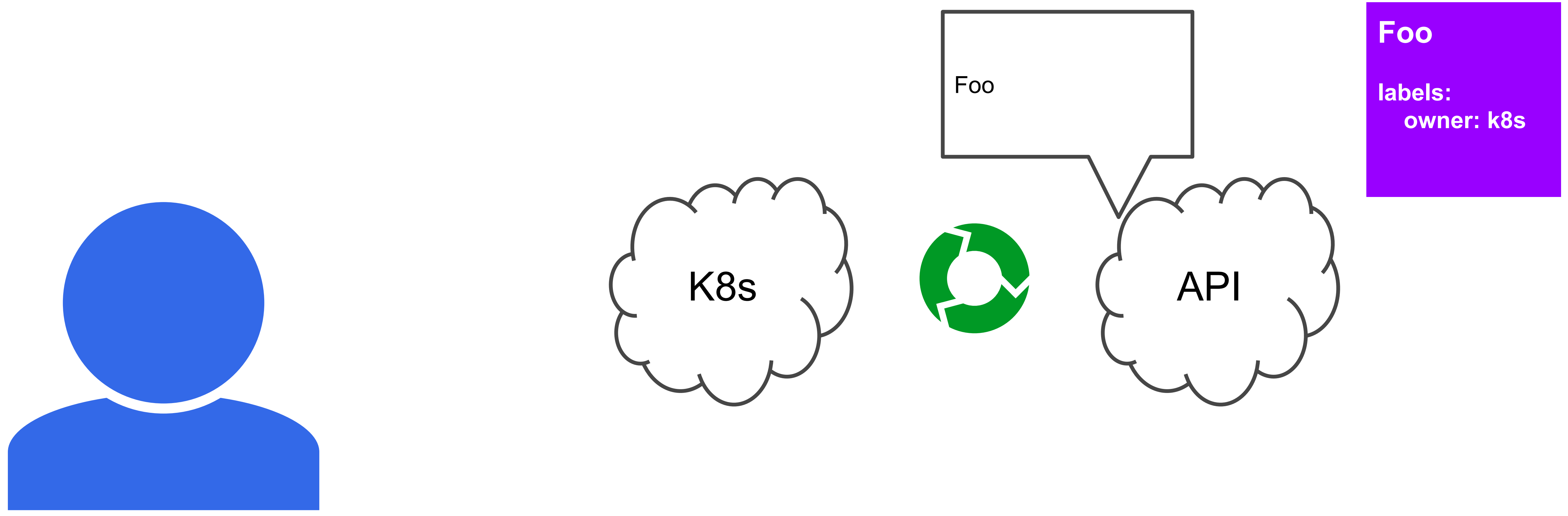
The controller has to know which shapes it **owns** and which it doesn't.

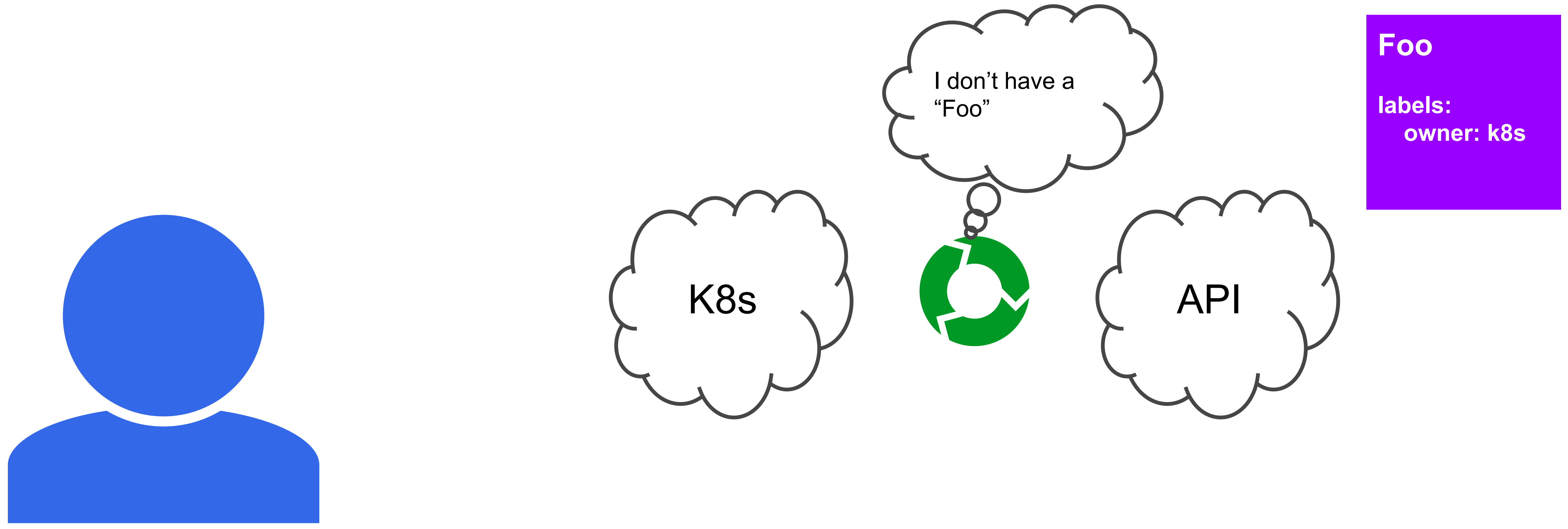
HOW to do that depends on the API. Examples:

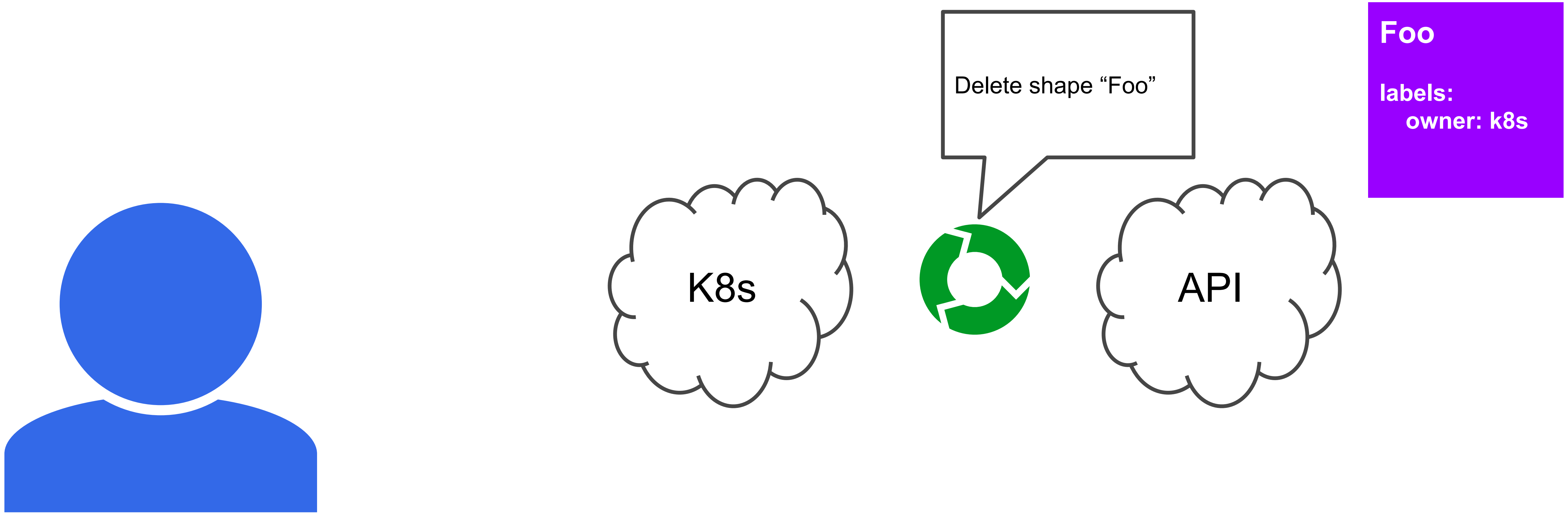
- Special name prefixes
- Metadata (labels, tags, description)
- Controller-specific checkpoints

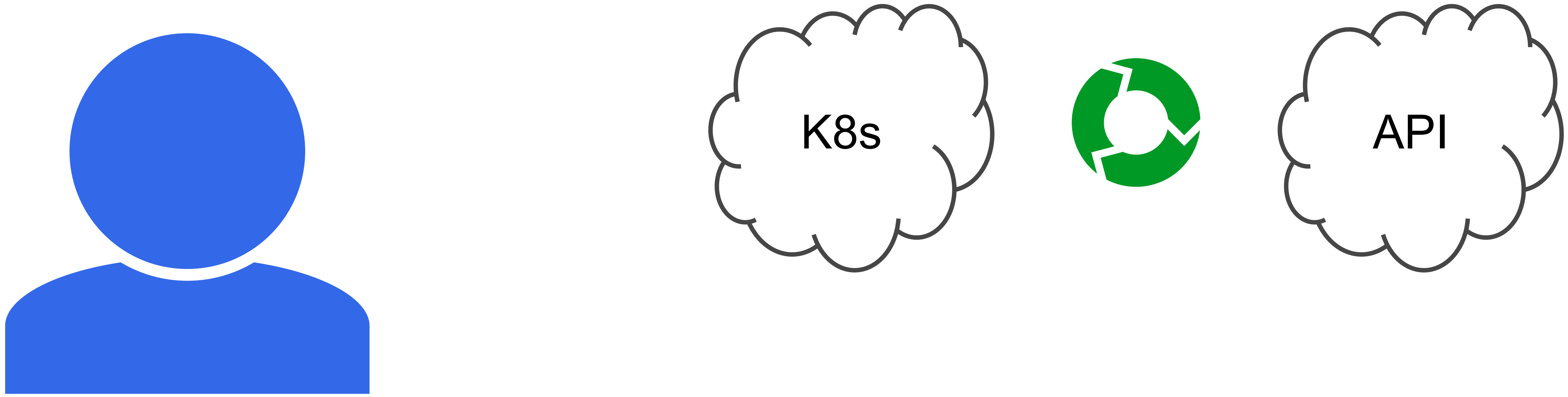

Foo
labels:
owner: k8s
K8s
RECOVER
API


List shapes where owner=k8s
Foo
labels:
owner: k8s
K8s
API


Foo
K8s
API
Foo
labels:
owner: k8s


I don’t have a “Foo”
K8s
API
Foo
labels:
owner: k8s

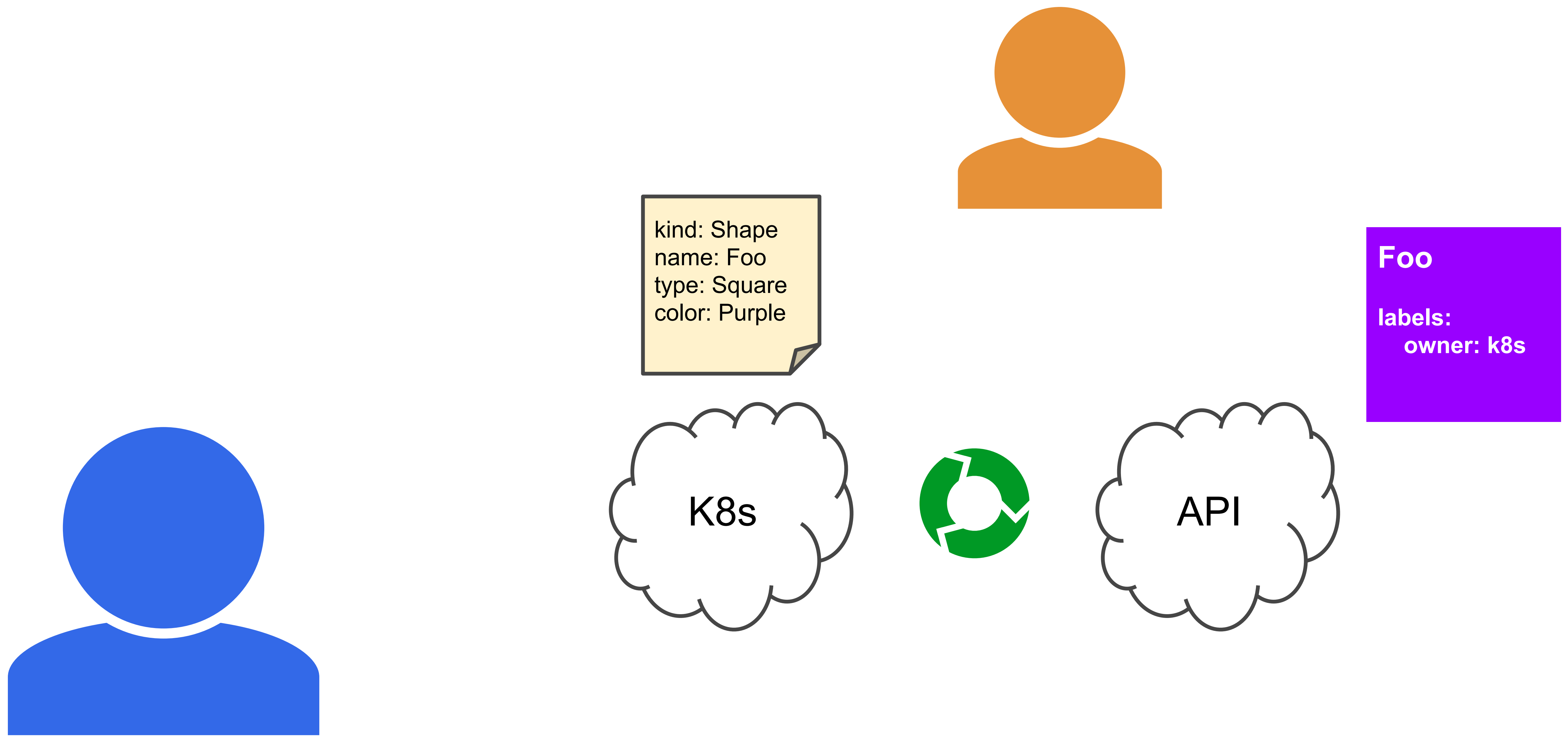

Delete shape “Foo”
K8s
API
Foo
labels:
owner: k8s

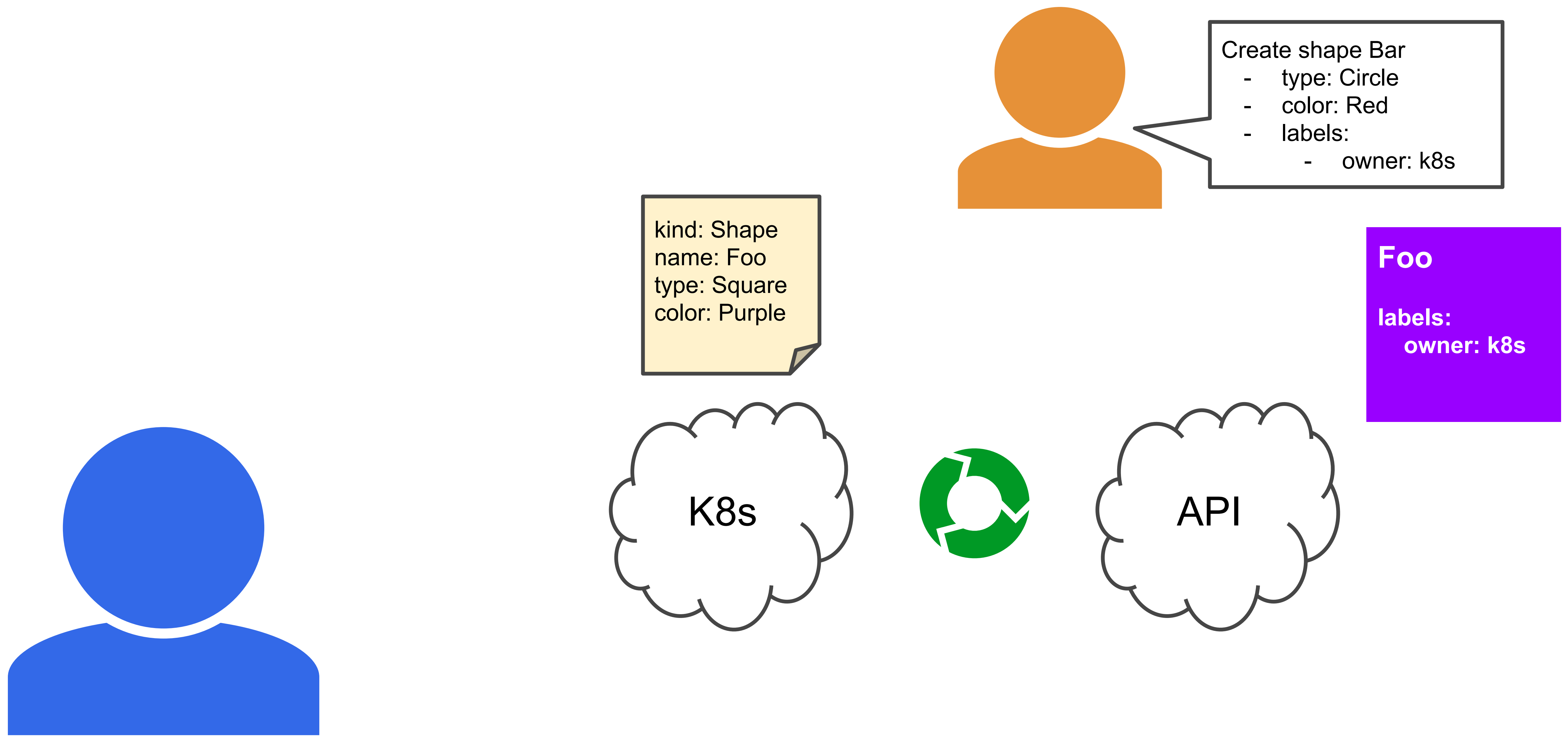

K8s
API

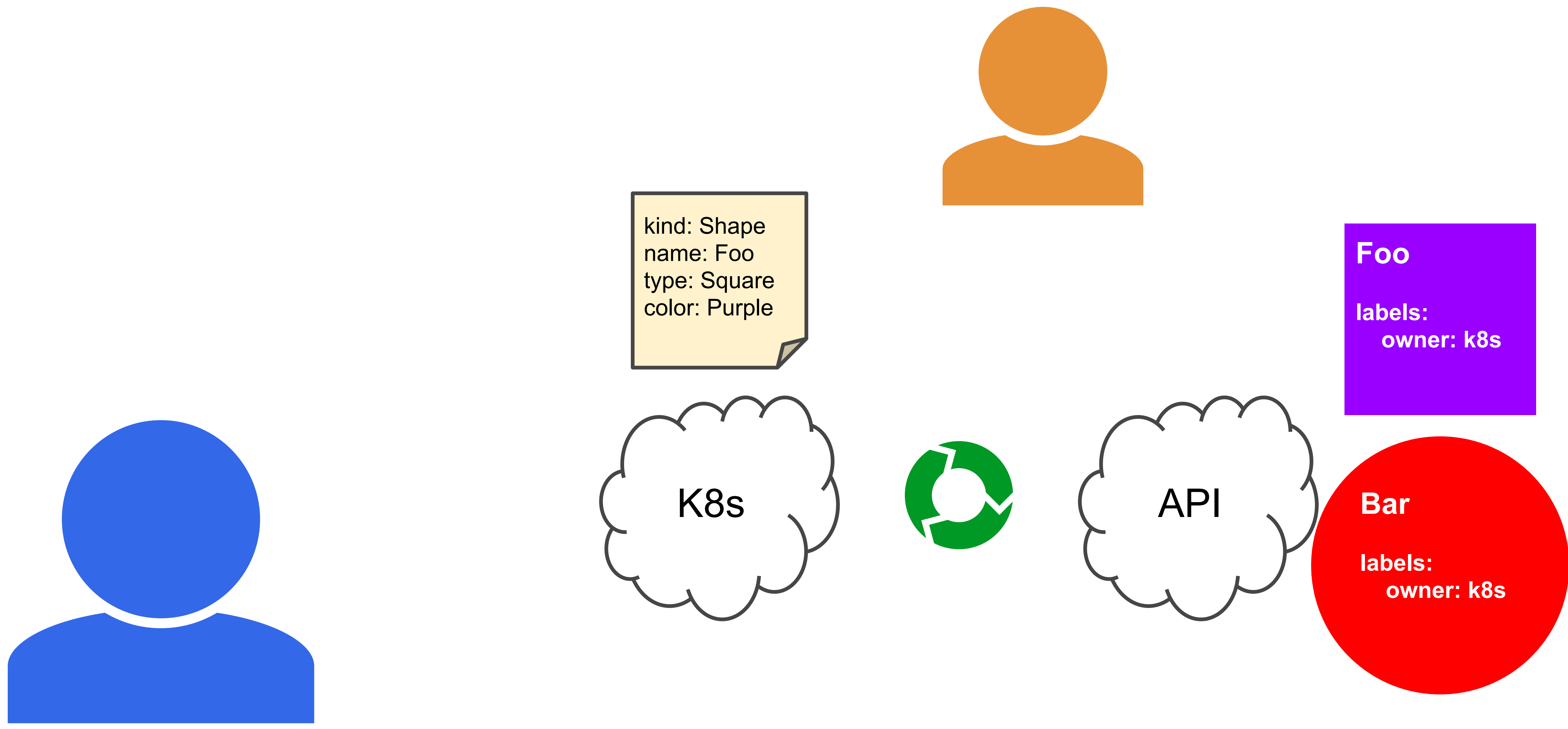
This is sometimes called the “list-watch” pattern.

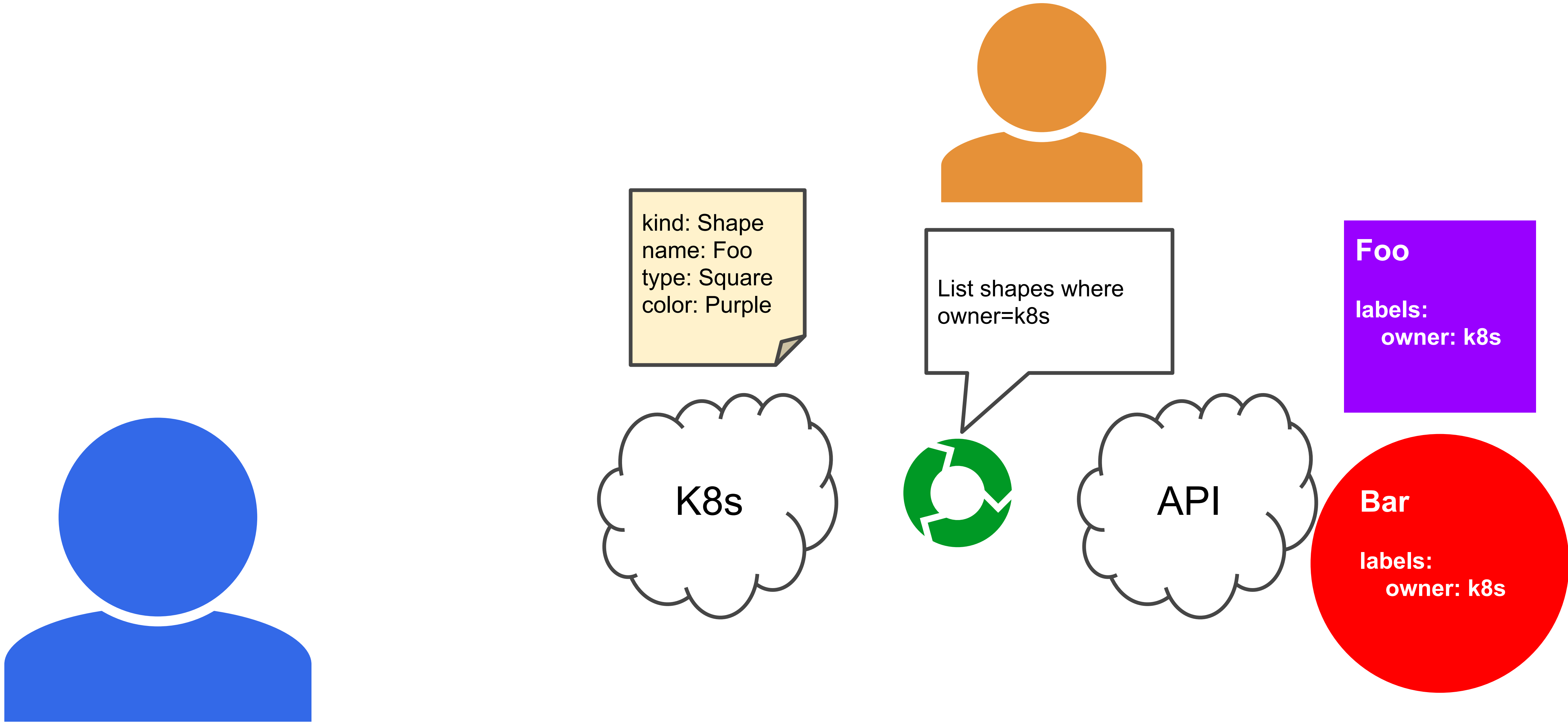
Now the controller will keep things it owns in sync and ignores other things.

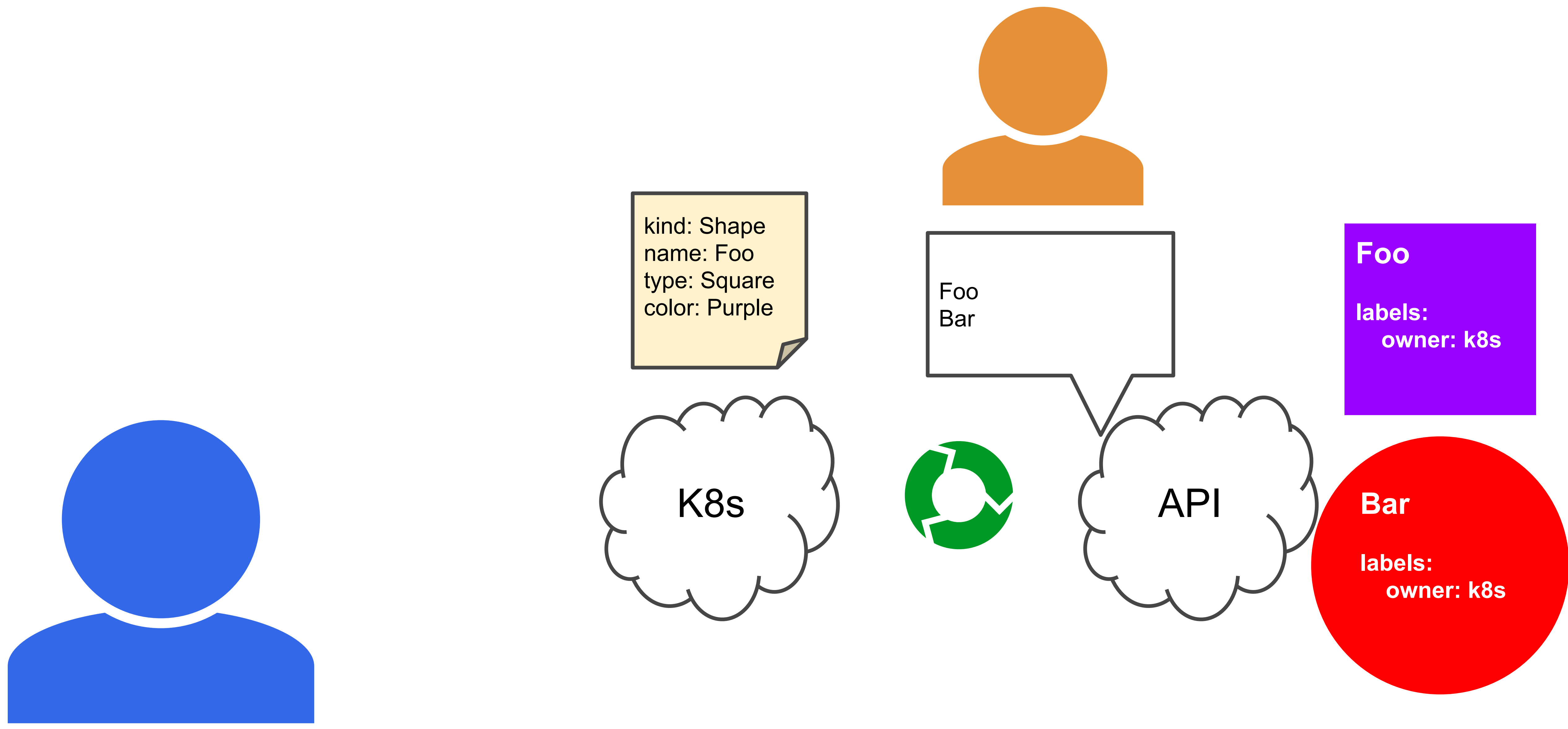
What if...

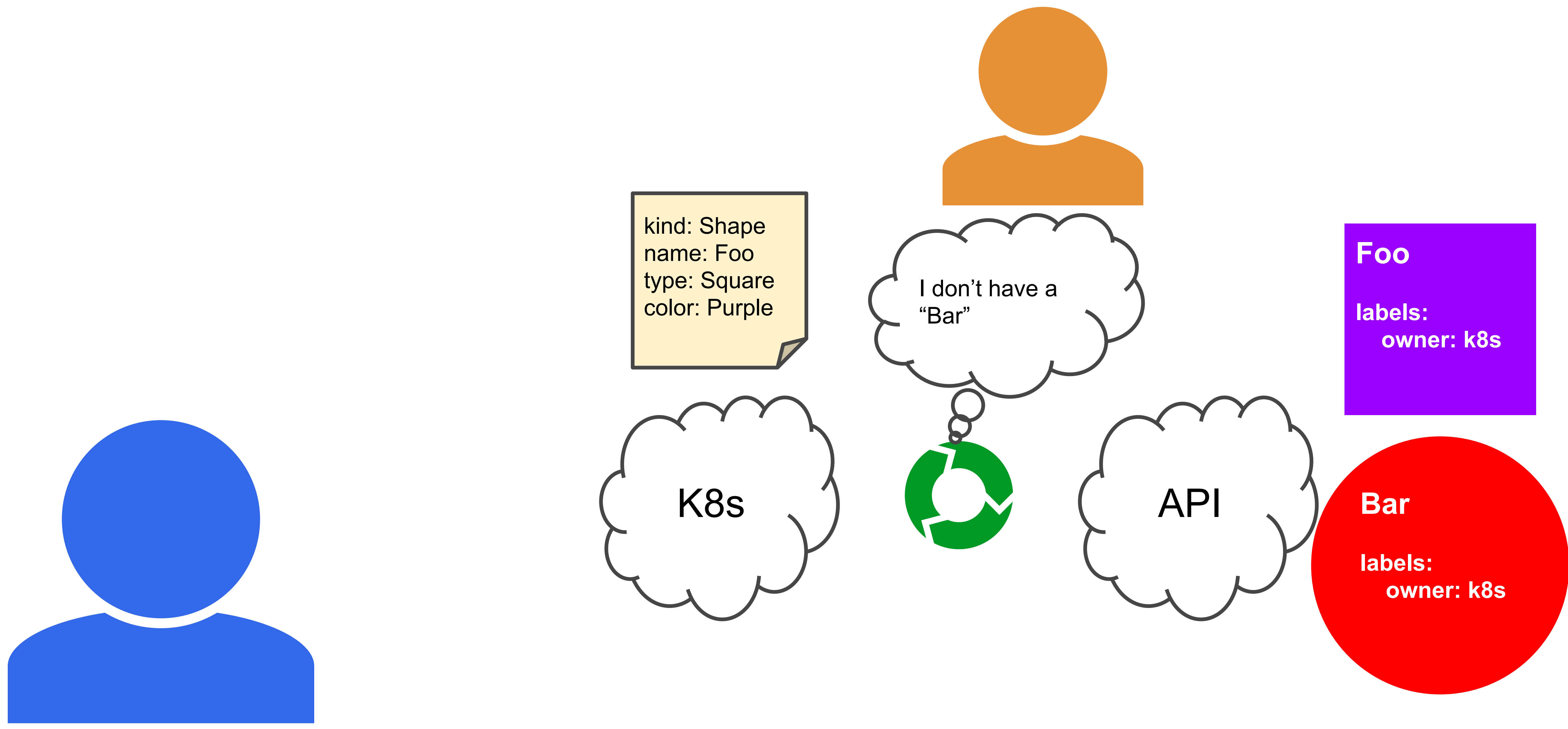

kind: Shape
name: Foo
type: Square
color: Purple
Foo
labels:
owner: k8s
K8s
API

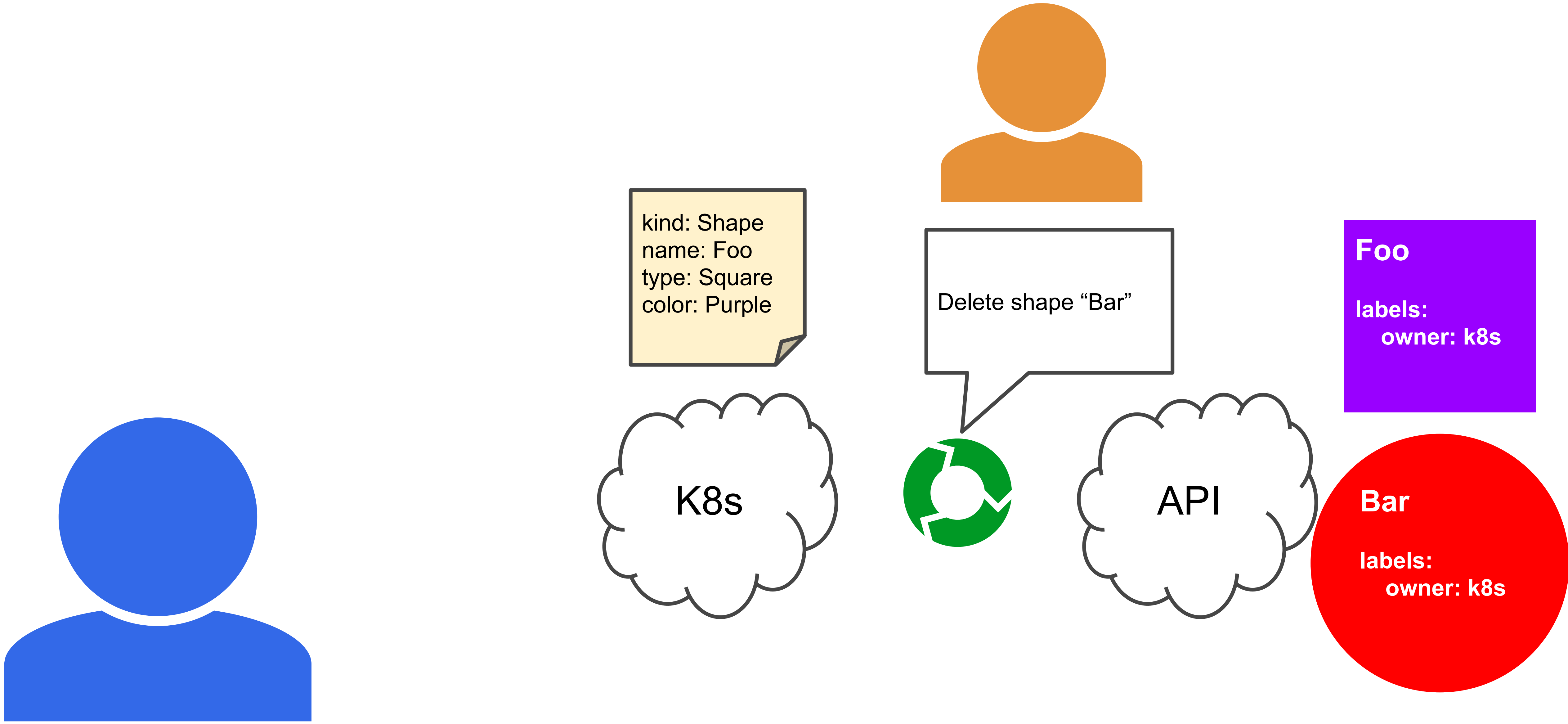

Create shape Bar
- type: Circle
- color: Red
- labels:
- owner: k8s
kind: Shape
name: Foo
type: Square
color: Purple
Foo
labels:
owner: k8s
K8s
API

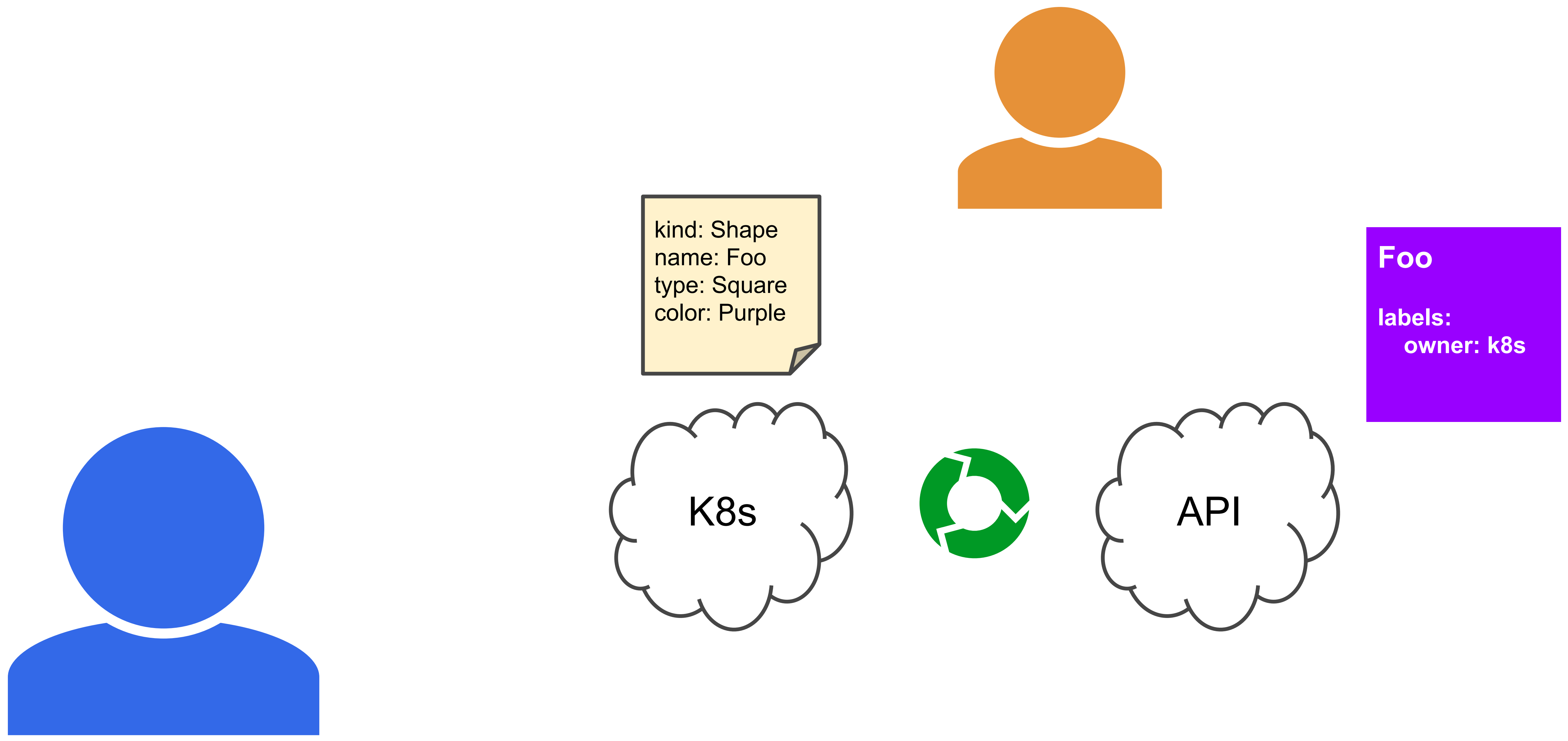

kind: Shape
name: Foo
type: Square
color: Purple
K8s
API
Foo
labels:
owner: k8s
Bar
labels:
owner: k8s


kind: Shape
name: Foo
type: Square
color: Purple
List shapes where owner=k8s
K8s
API
Foo
labels:
owner: k8s
Bar
labels:
owner: k8s


kind: Shape
name: Foo
type: Square
color: Purple
Foo
Bar
K8s
API
Foo
labels:
owner: k8s
Bar
labels:
owner: k8s


kind: Shape
name: Foo
type: Square
color: Purple
I don't have a "Bar"
K8s
API
Foo
labels:
owner: k8s
Bar
labels:
owner: k8s


kind: Shape
name: Foo
type: Square
color: Purple
Delete shape “Bar”
K8s
API
Foo
labels:
owner: k8s
Bar
labels:
owner: k8s


kind: Shape
name: Foo
type: Square
color: Purple
K8s
API
Foo
labels:
owner: k8s

Note that while doing a full reconciliation at startup is necessary, it is not sufficient.

Good controllers will reconcile against underlying APIs continuously or at least periodically.

How does this apply to real life?

This pattern is found in almost every case where Kubernetes layers on top of some other API. Examples:

- Cloud load-balancers for Services & Ingress
- Cloud disks for PersistentVolumes
- iptables rules for Services
- Running containers for Pods

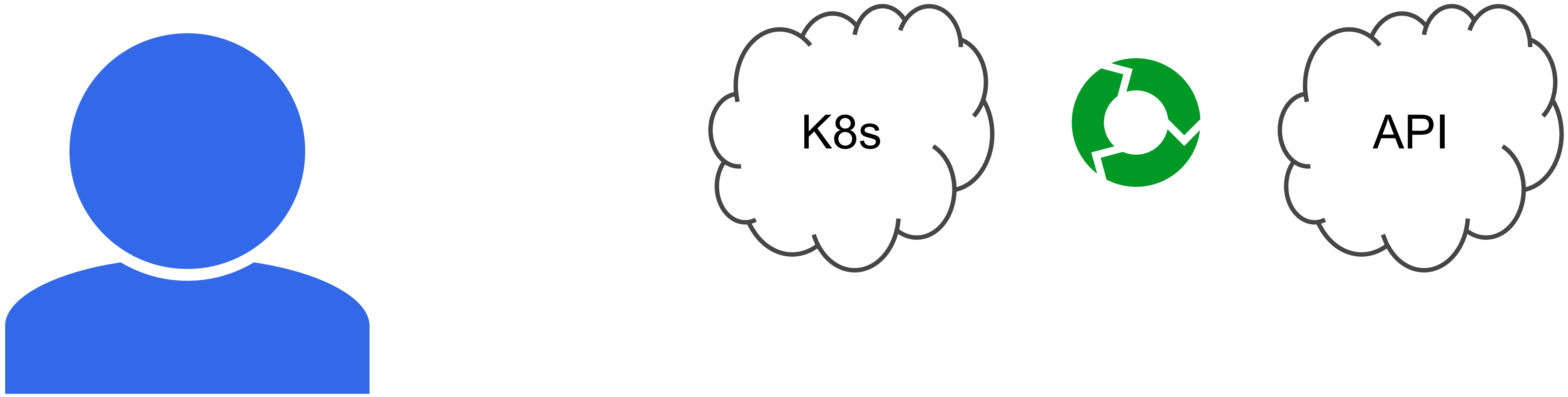
Sadly, not every controller gets this right.

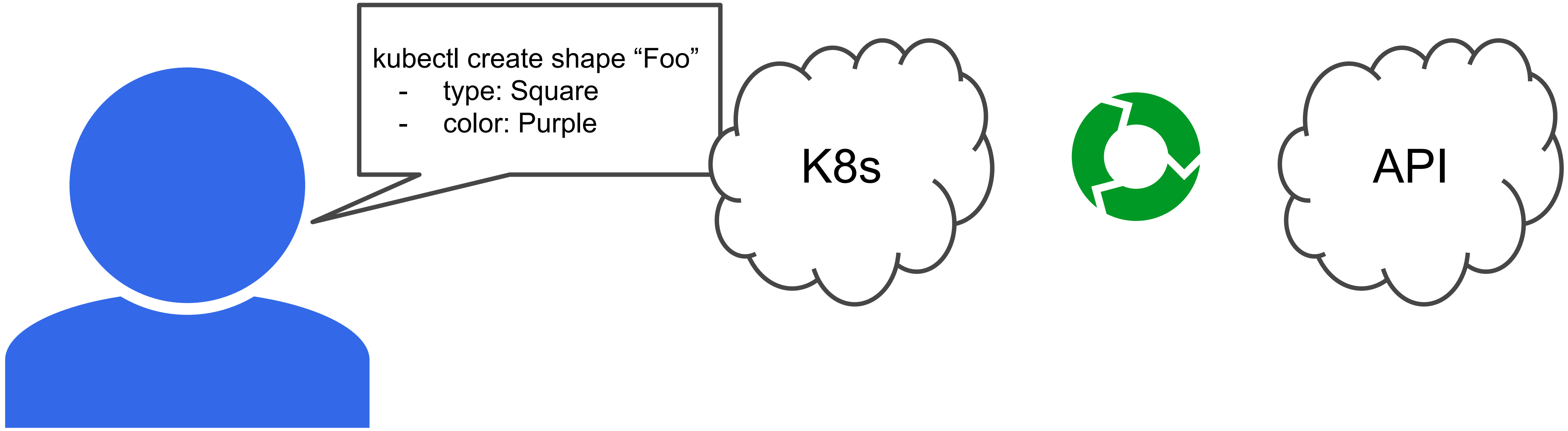
While every controller should strive for complete reconciliation, sometimes the underlying API makes it very hard or expensive or even just impossible.

:(

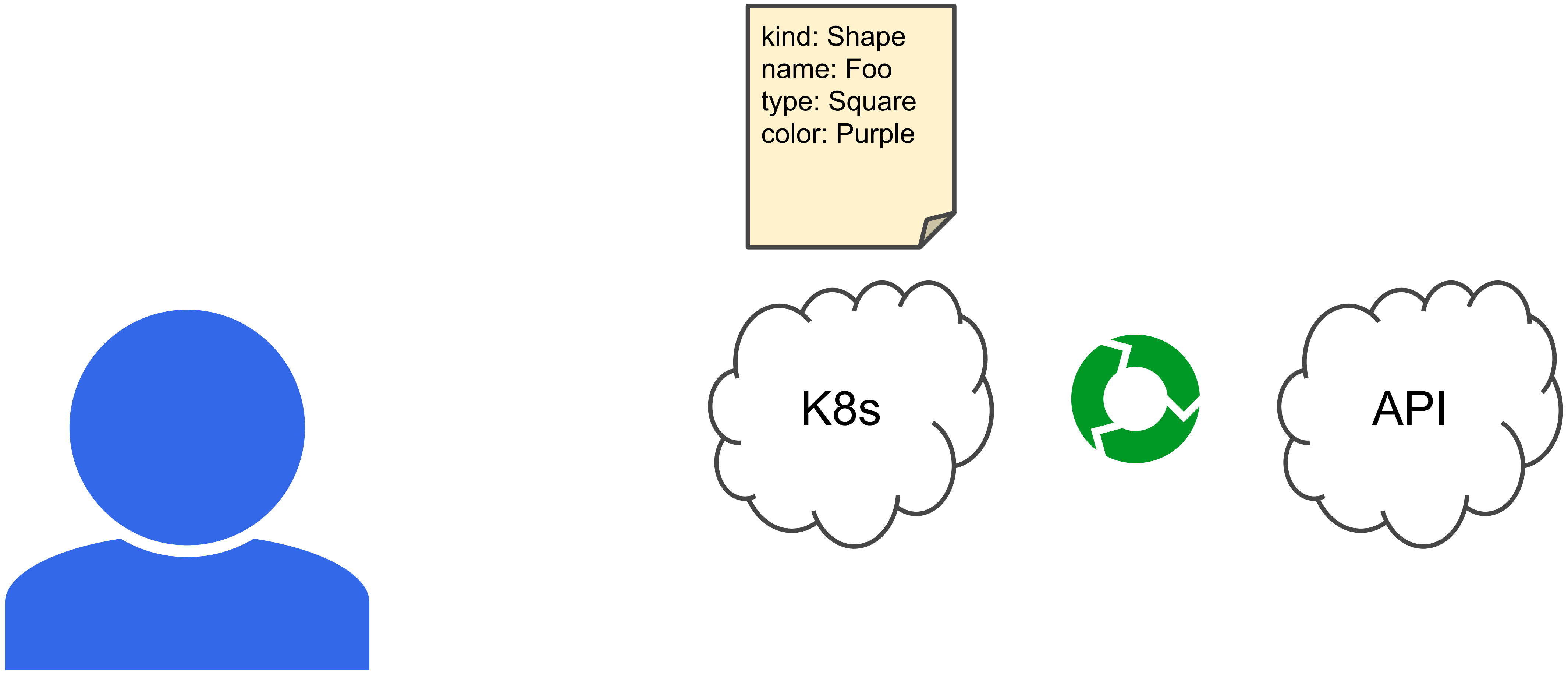
There are some techniques that can mitigate the lack of mechanisms to denote ownership (or augment them).

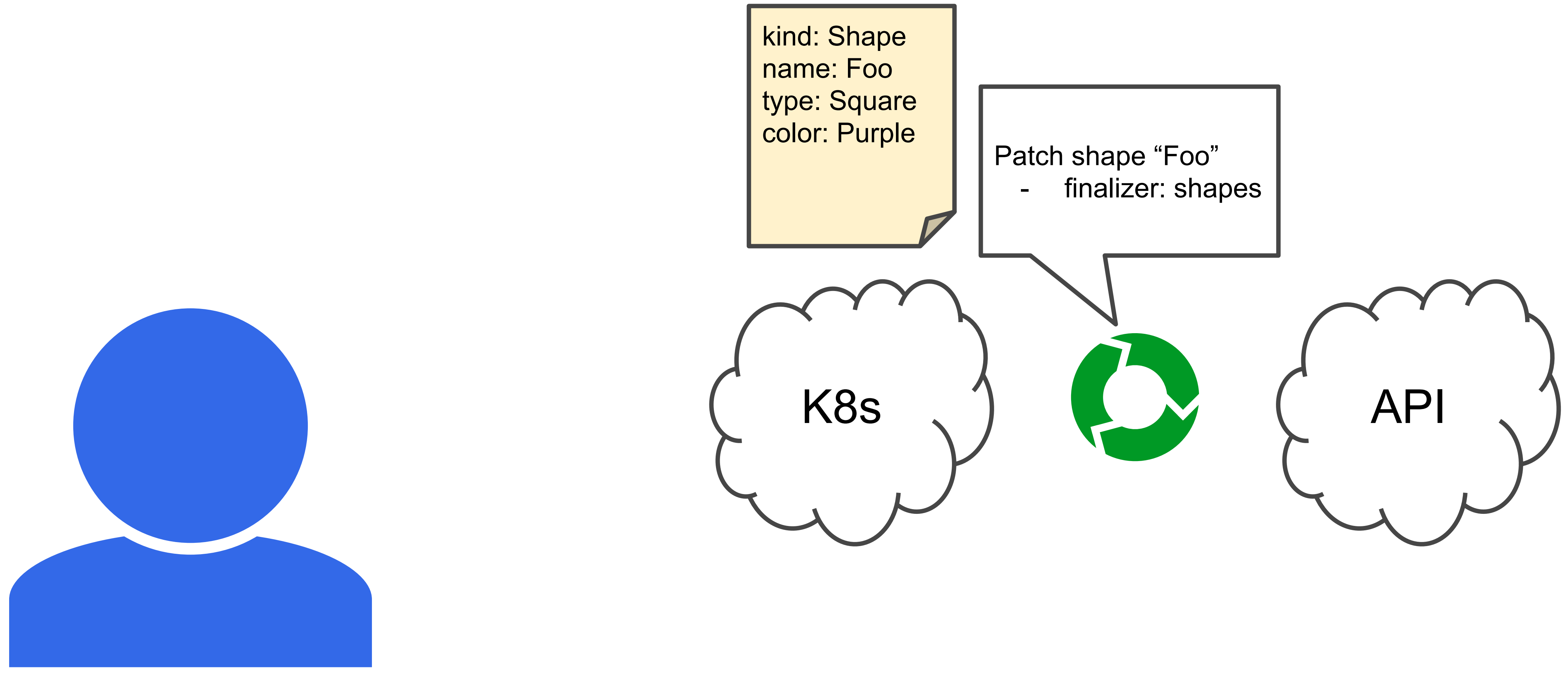
Finalizers

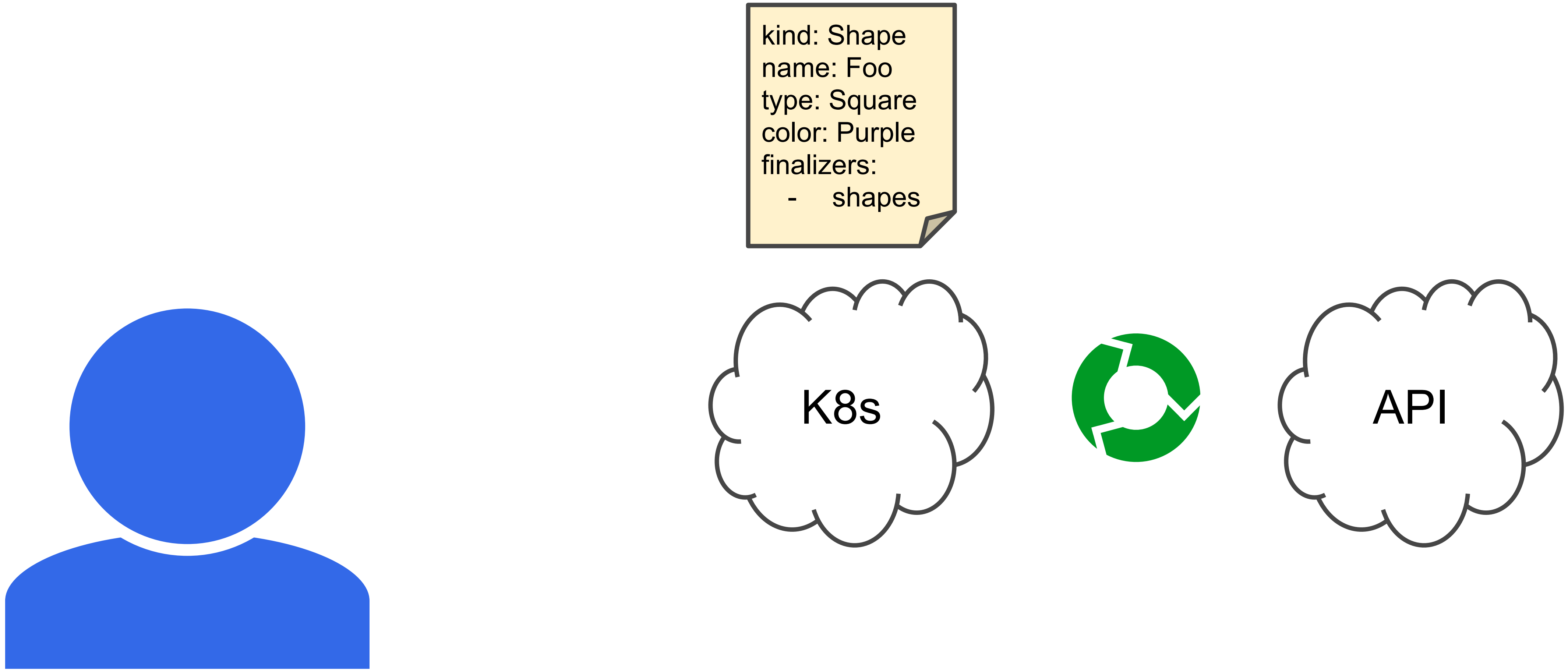

K8s
API



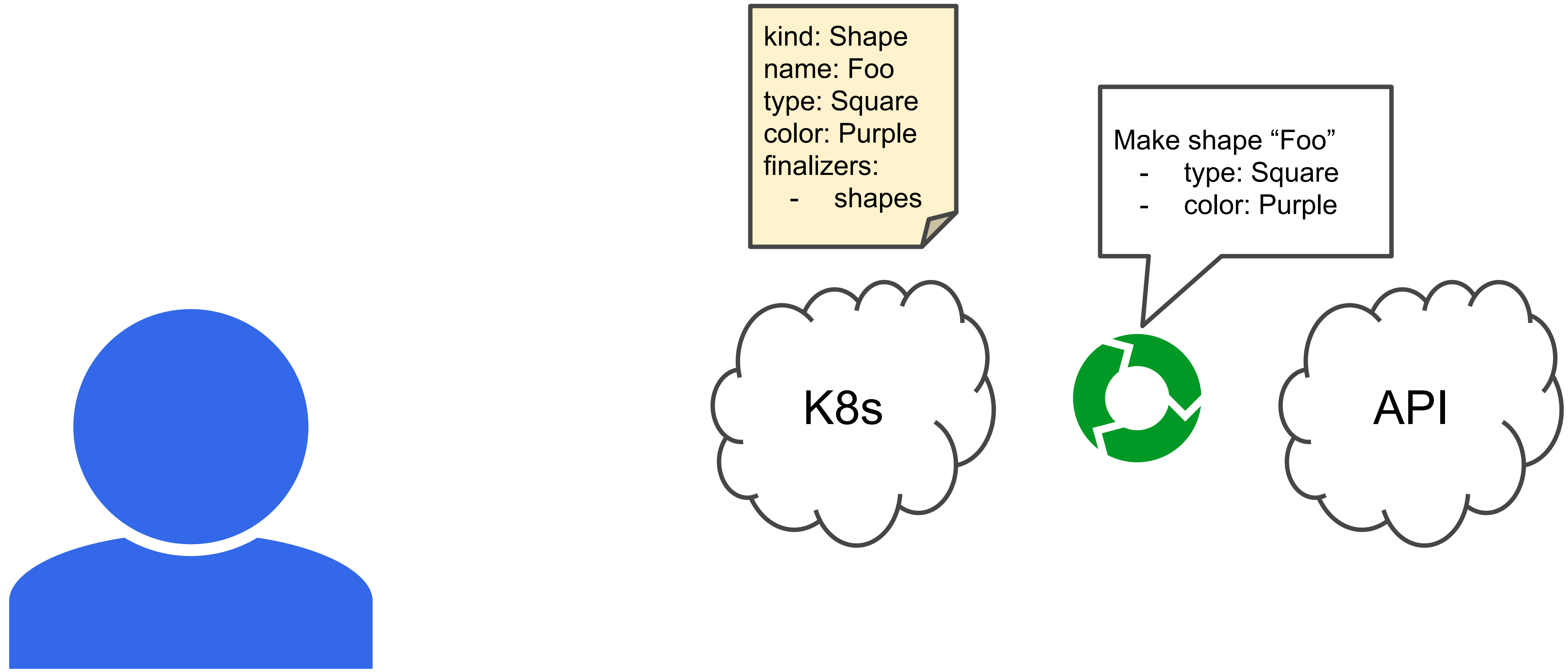
```
kubectl create shape “Foo”
  -   type: Square
  -   color: Purple
```

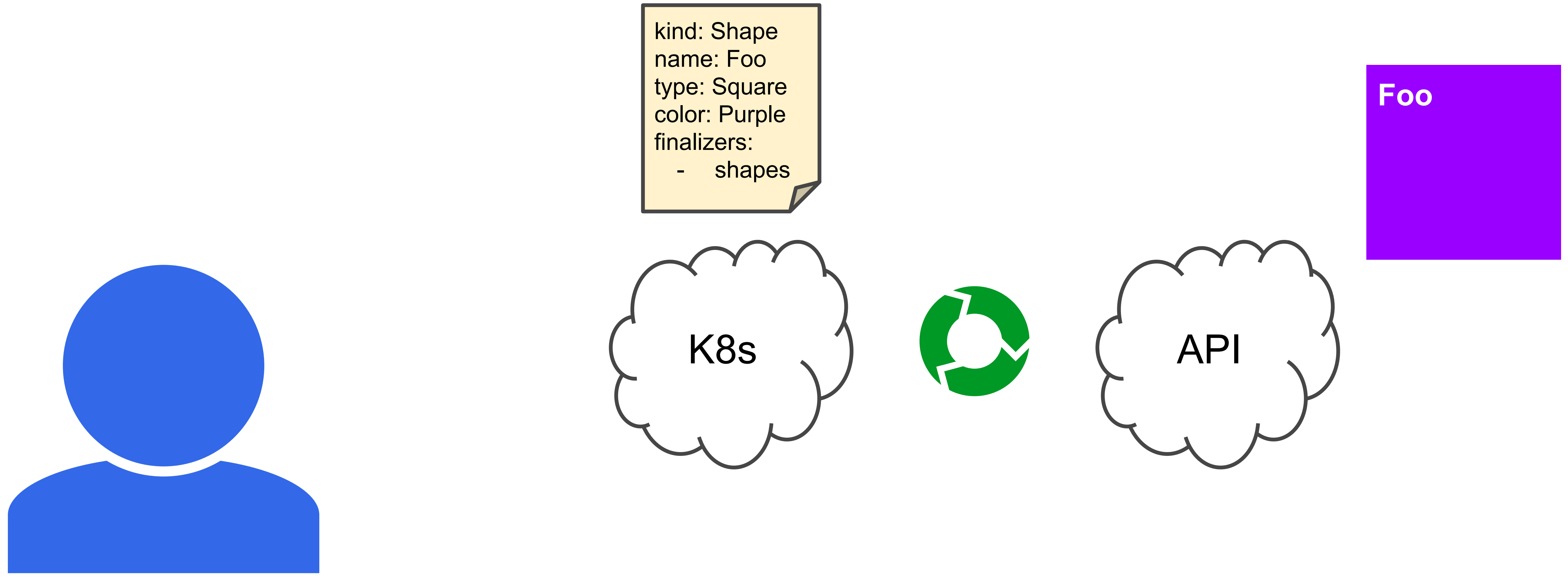

kind: Shape
name: Foo
type: Square
color: Purple
K8s
API


kind: Shape
name: Foo
type: Square
color: Purple
Patch shape “Foo”
- finalizer: shapes
K8s
API

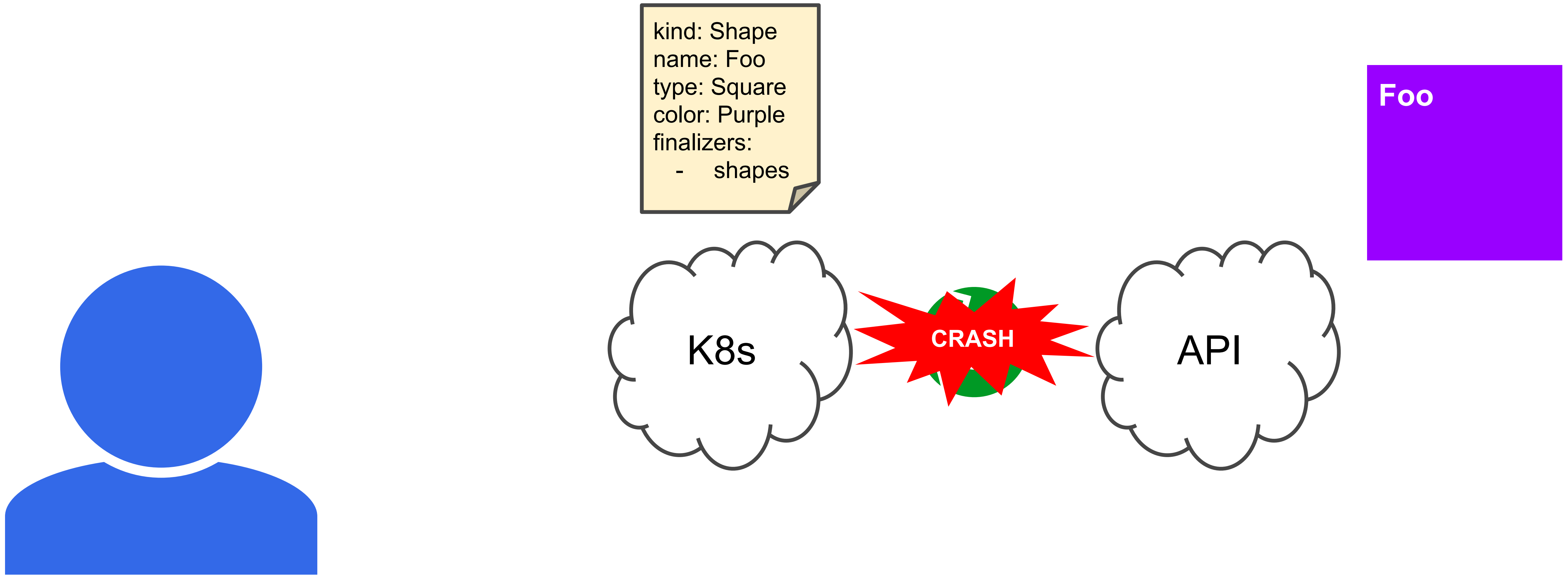


```
kind: Shape
name: Foo
type: Square
color: Purple
finalizers:
  -   shapes
```

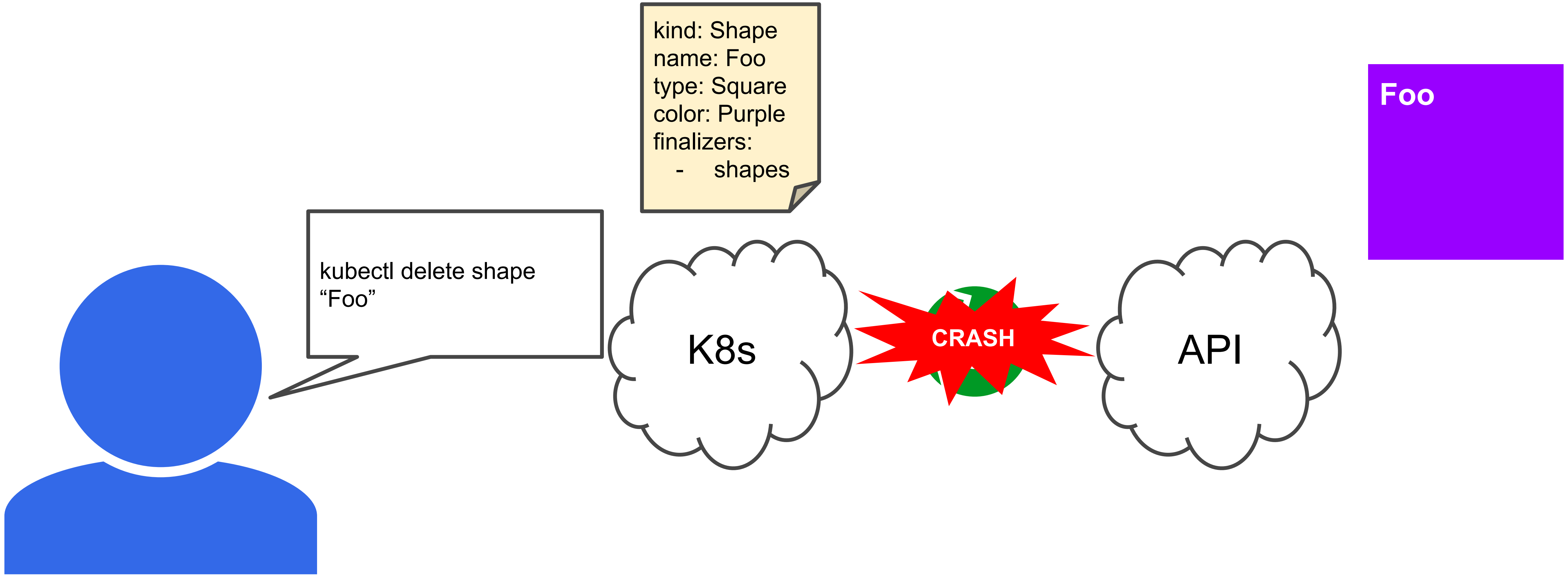

kind: Shape
name: Foo
type: Square
color: Purple
finalizers:
- shapes
Make shape “Foo”
- type: Square
- color: Purple
K8s
API



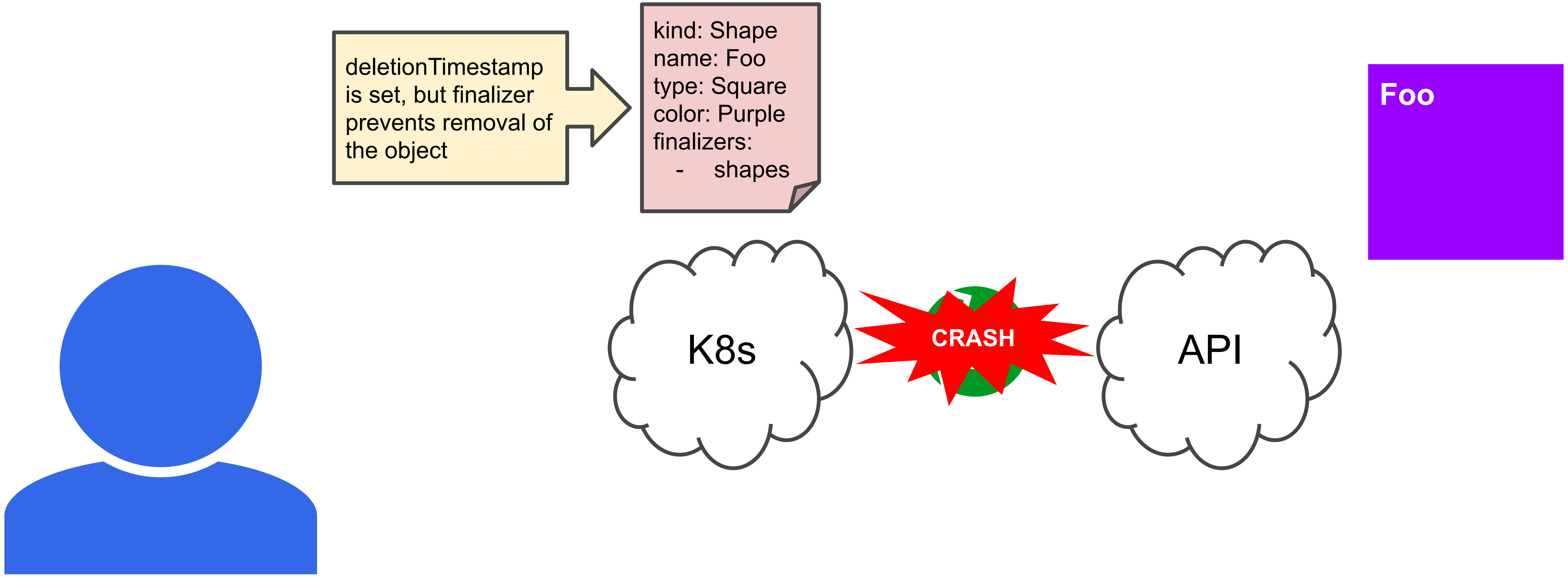
```
kind: Shape
name: Foo
type: Square
color: Purple
finalizers:
  -   shapes
```



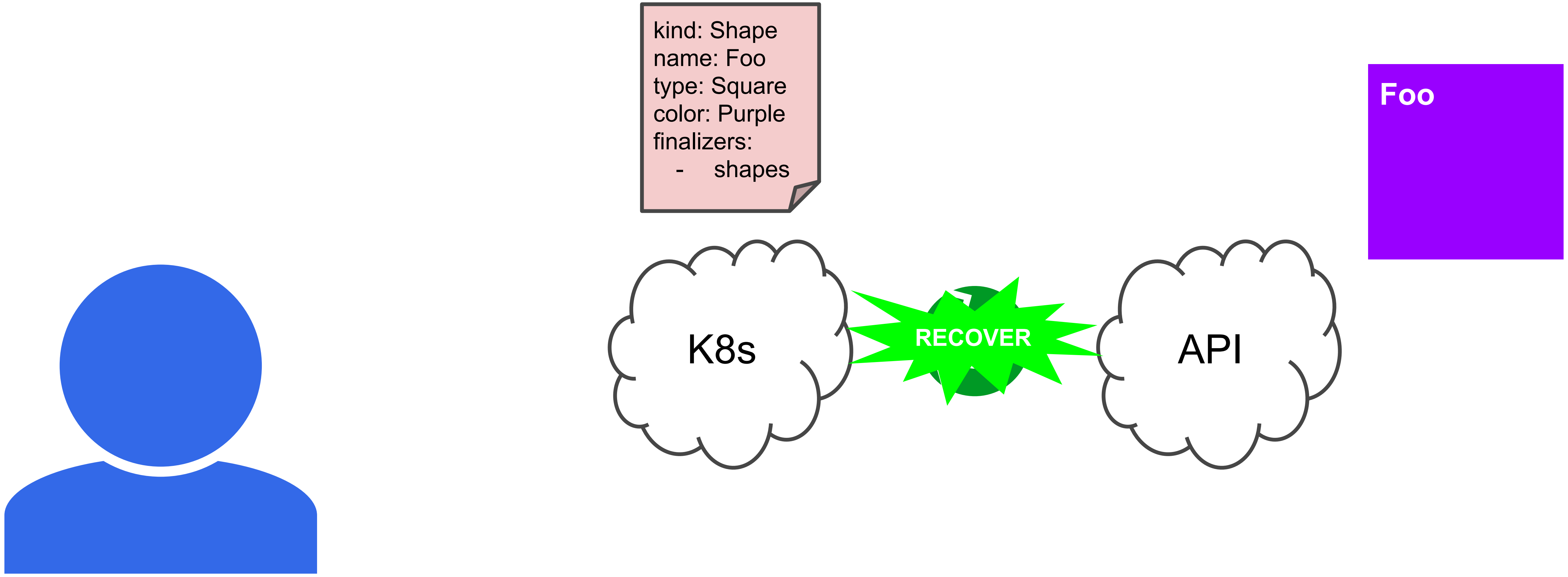
```
kind: Shape
name: Foo
type: Square
color: Purple
finalizers:
   -    shapes
```



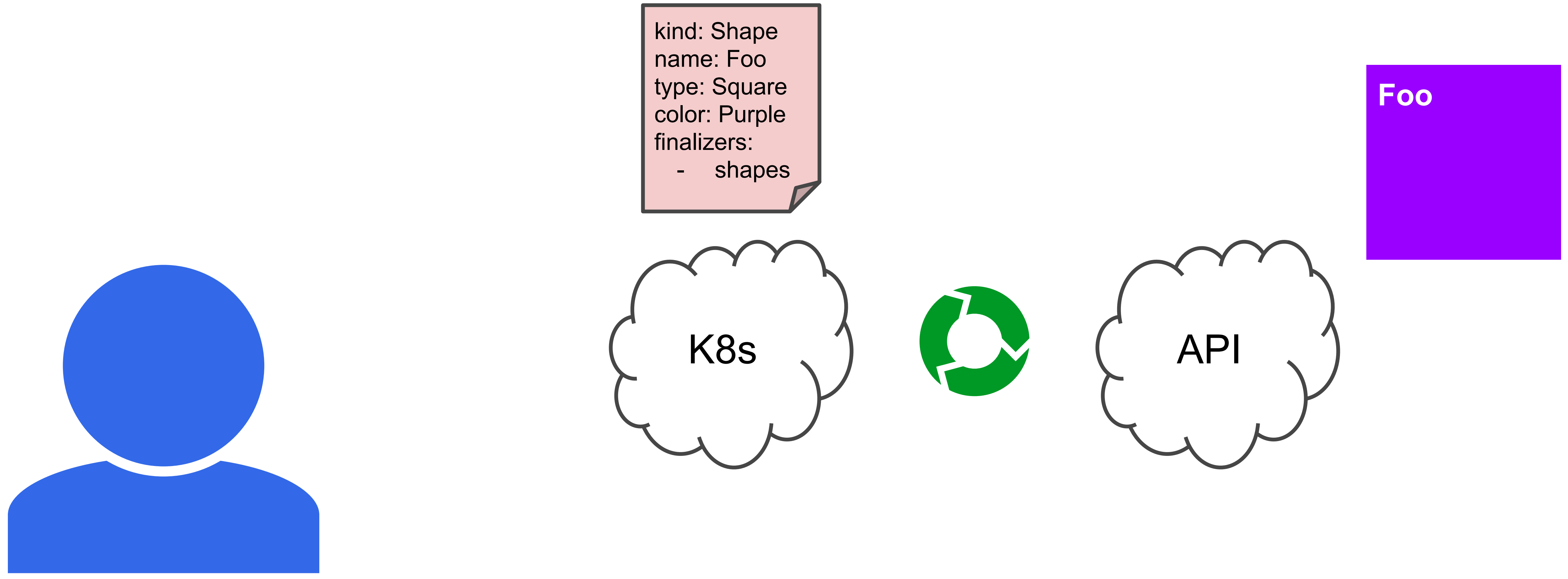
```
kind: Shape
name: Foo
type: Square
color: Purple
finalizers:
   -    shapes
```


deletionTimestamp is set, but finalizer prevents removal of the object
K8s
CRASH
API
Foo

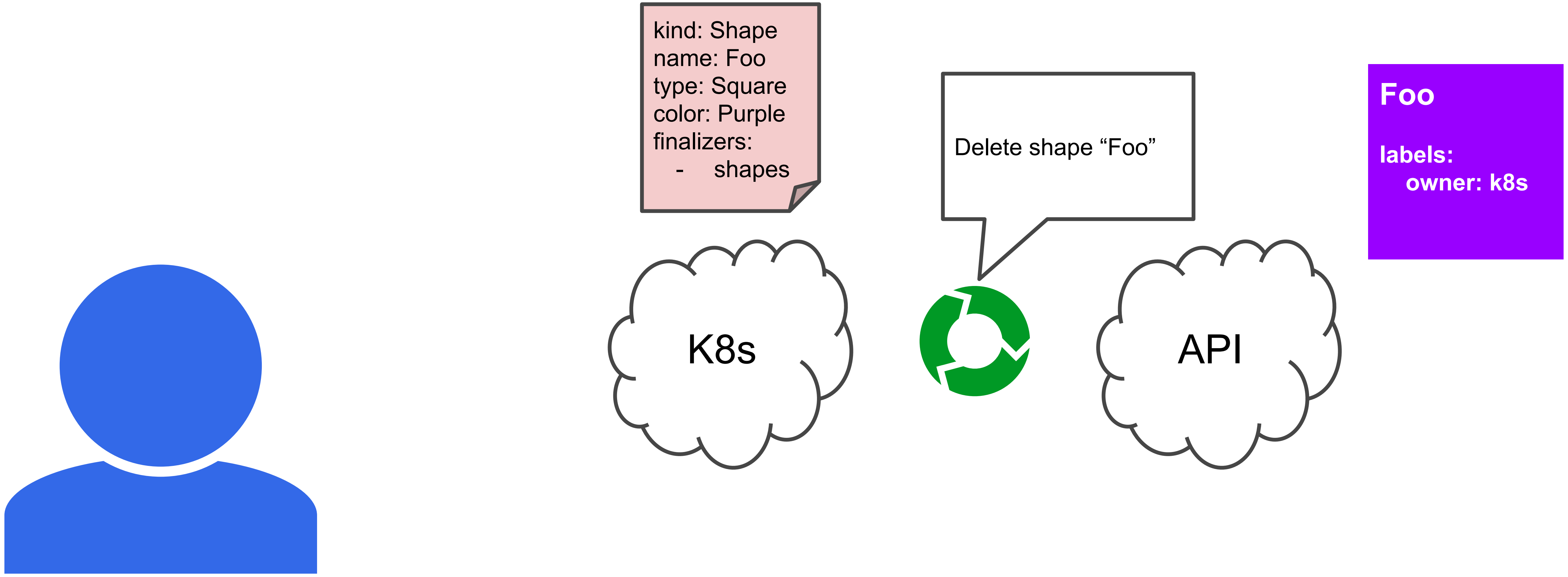
```
kind: Shape
name: Foo
type: Square
color: Purple
finalizers:
  -   shapes
```



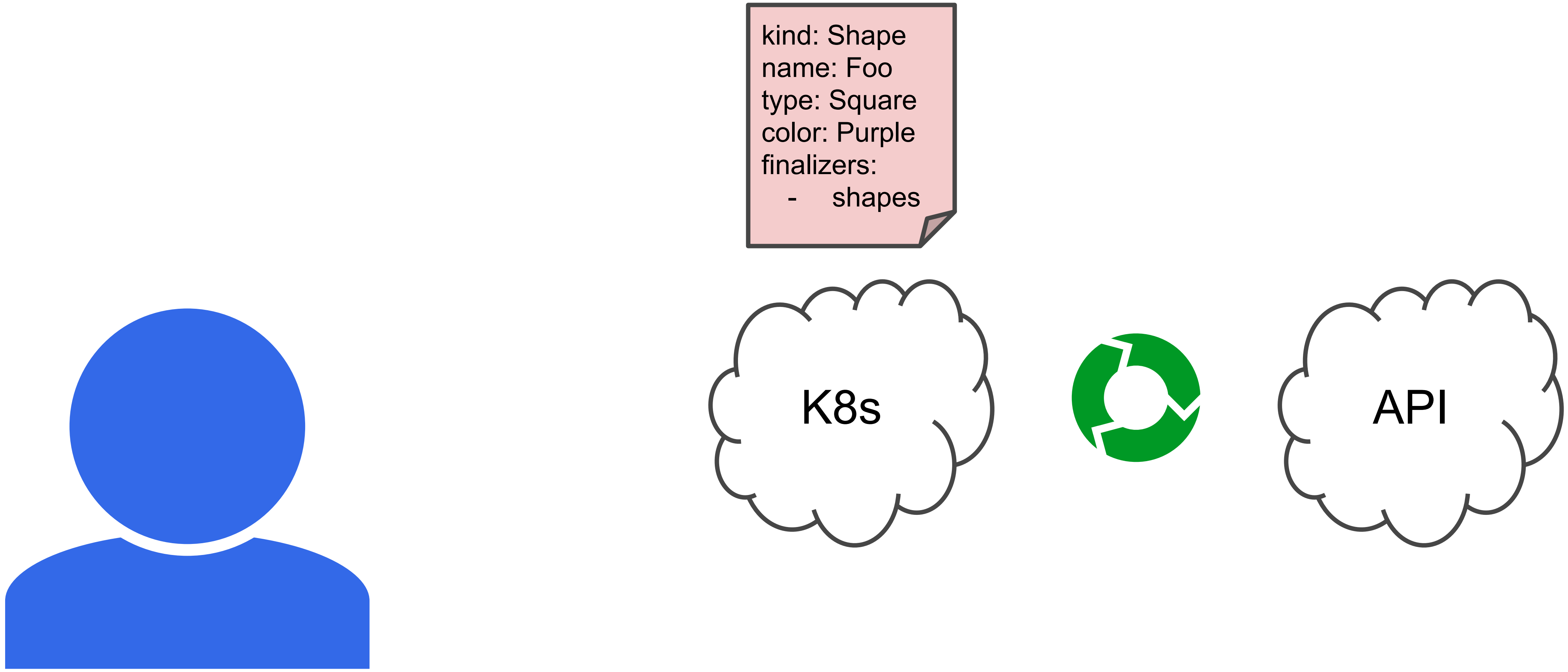
```
kind: Shape
name: Foo
type: Square
color: Purple
finalizers:
  -   shapes
```


kind: Shape
name: Foo
type: Square
color: Purple
finalizers:
- shapes
K8s
API
Foo

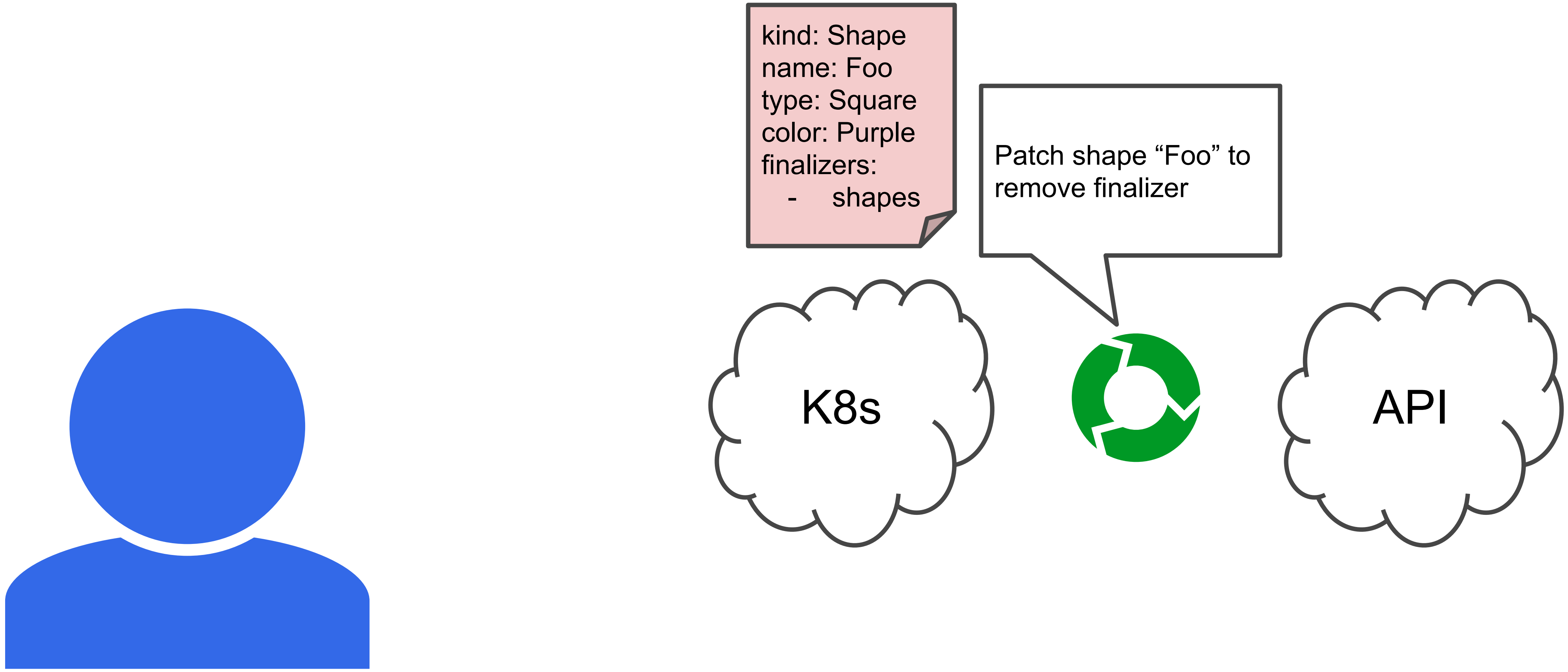
The controller observes the pending deletion of the shape.

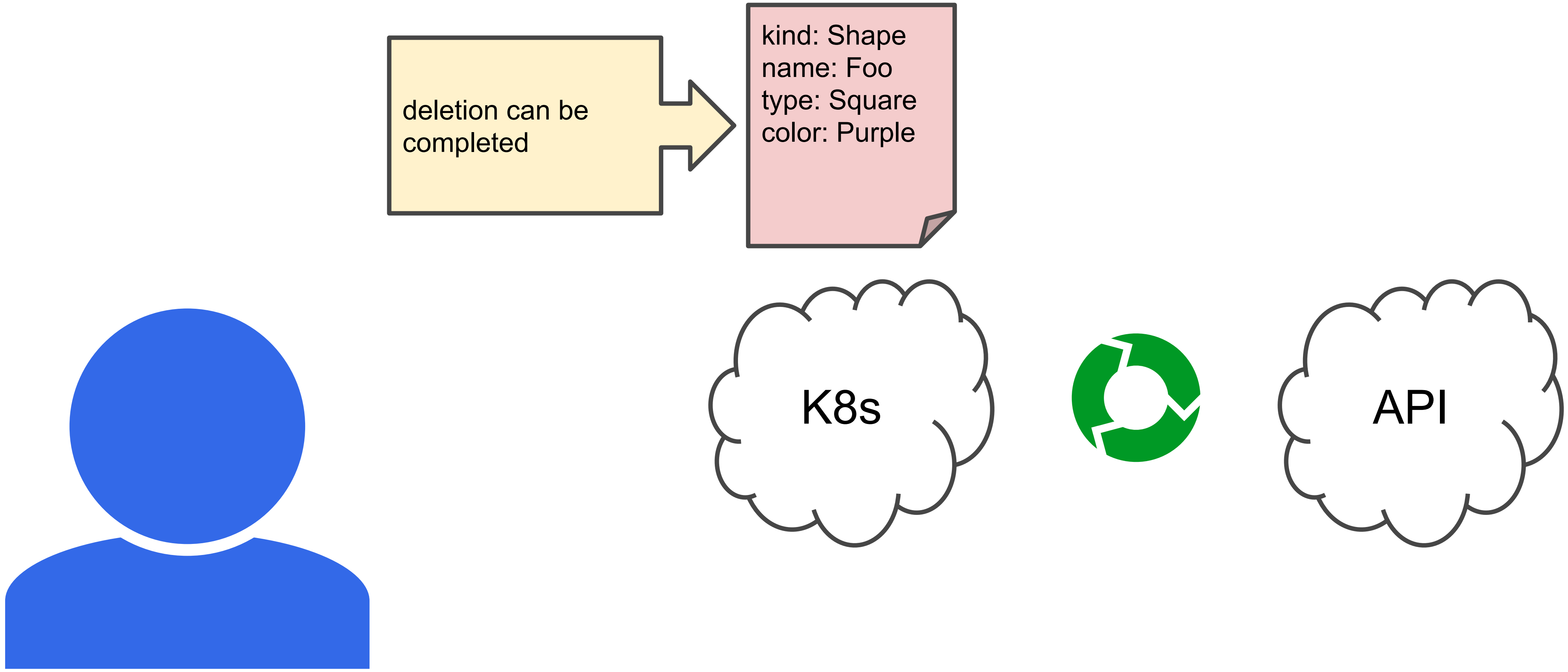


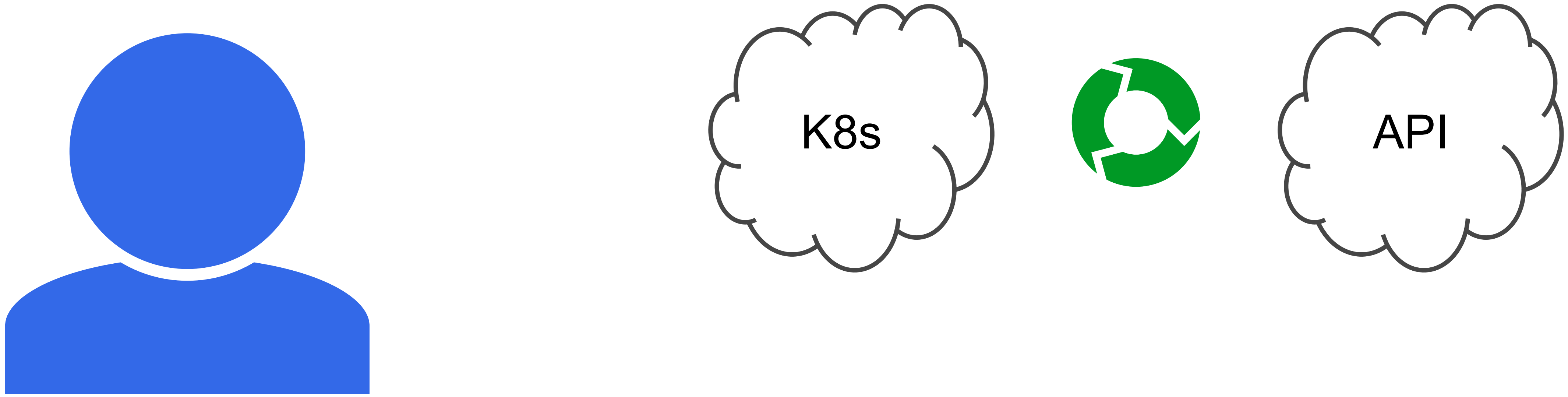
```
kind: Shape
name: Foo
type: Square
color: Purple
finalizers:
  -   shapes
```



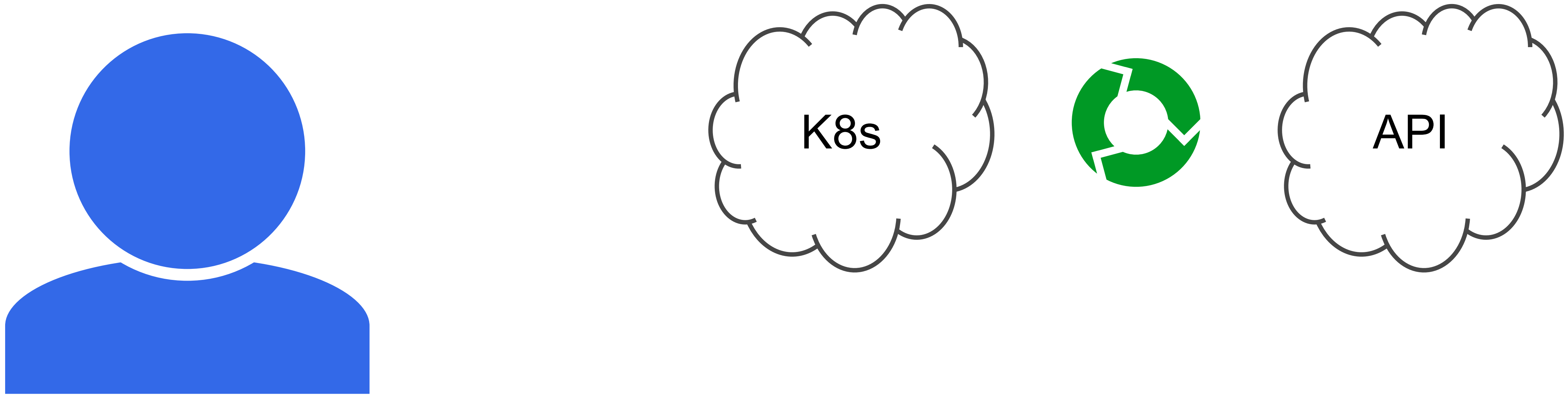
```
kind: Shape
name: Foo
type: Square
color: Purple
finalizers:
   -   shapes
```

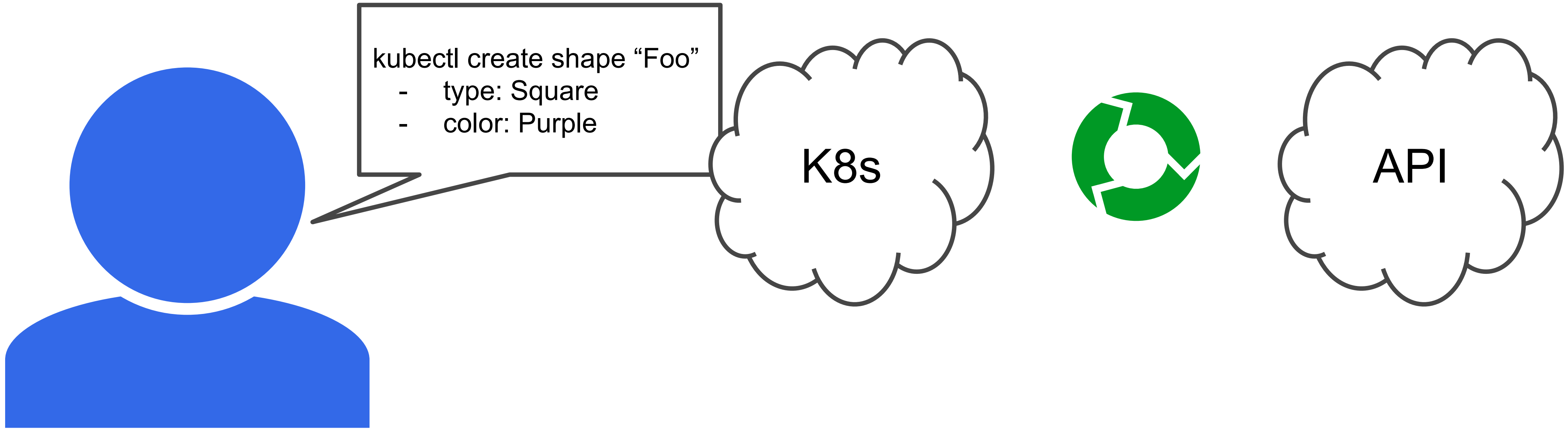

kind: Shape
name: Foo
type: Square
color: Purple
finalizers:
- shapes
Patch shape “Foo” to remove finalizer
K8s
API


deletion can be completed
kind: Shape
name: Foo
type: Square
color: Purple
K8s
API

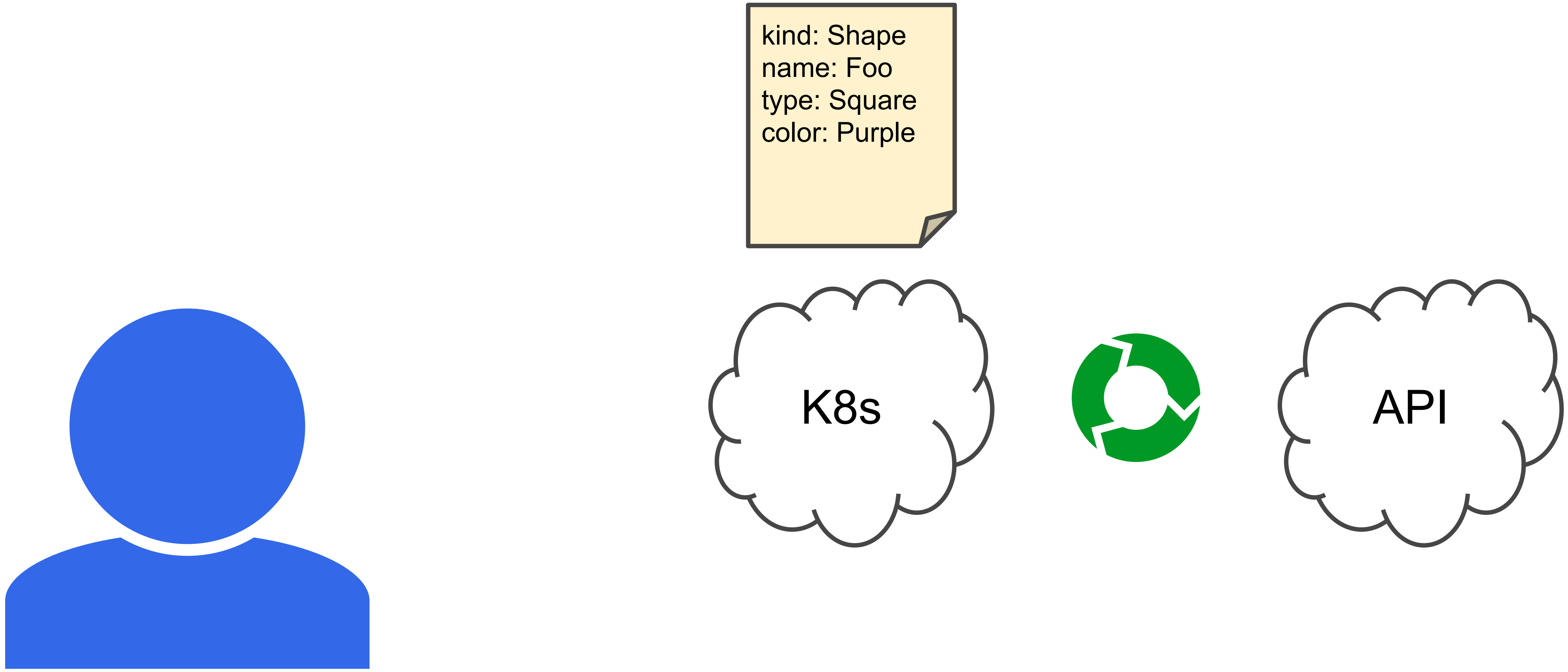

K8s
API

CustomResources

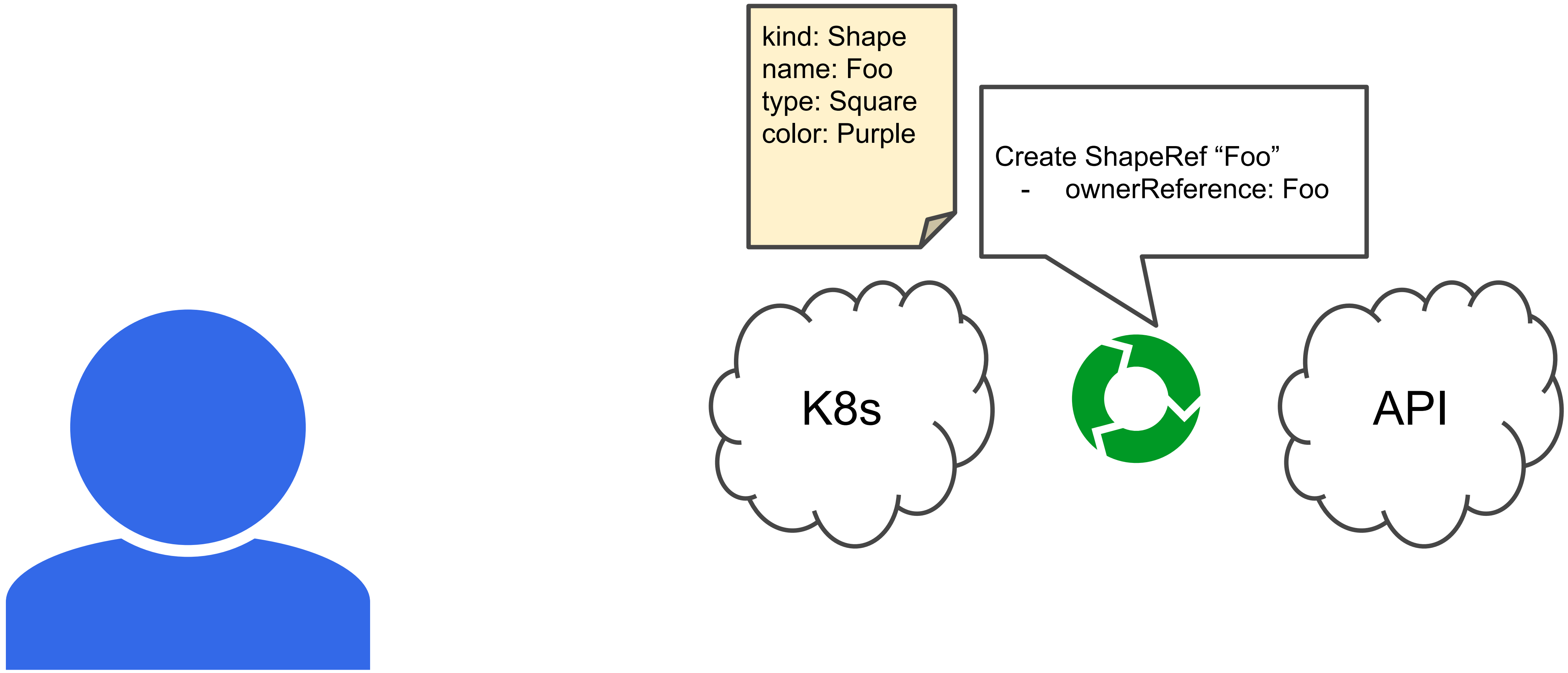

K8s
API

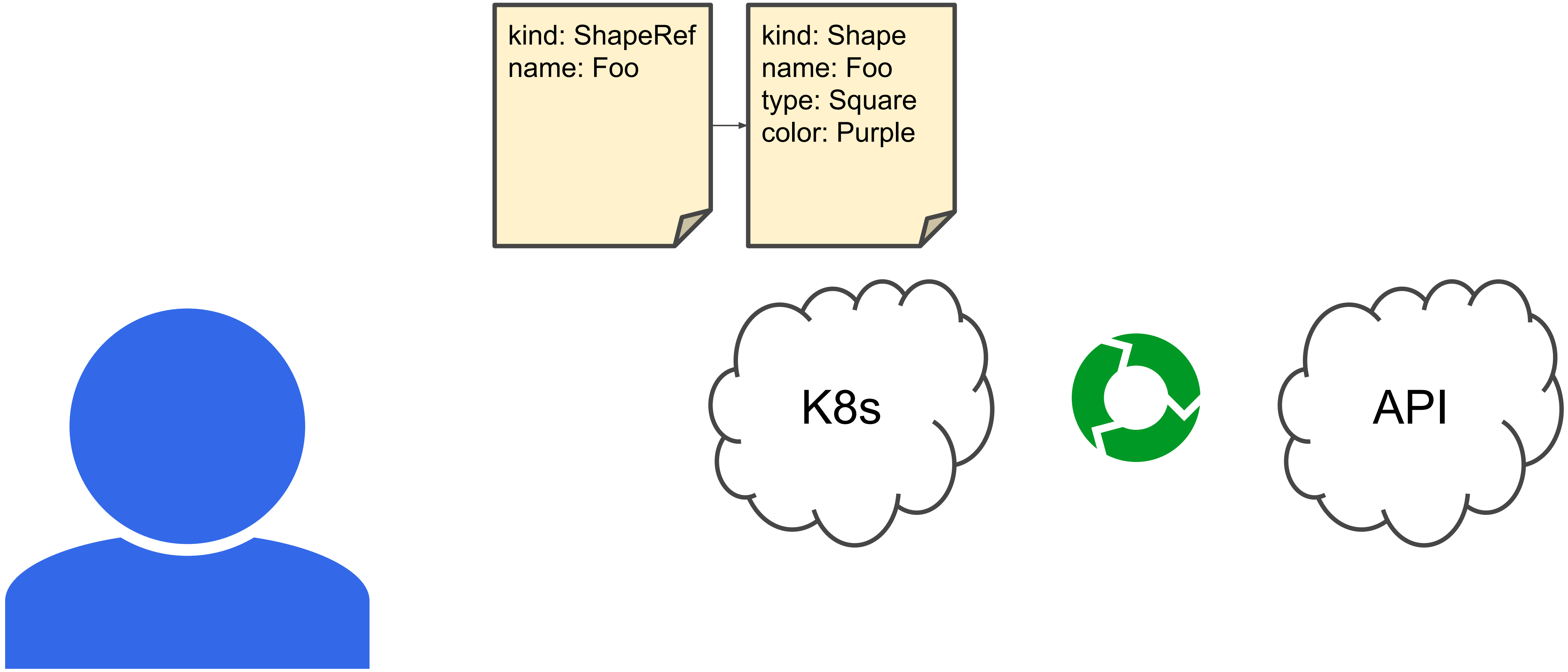


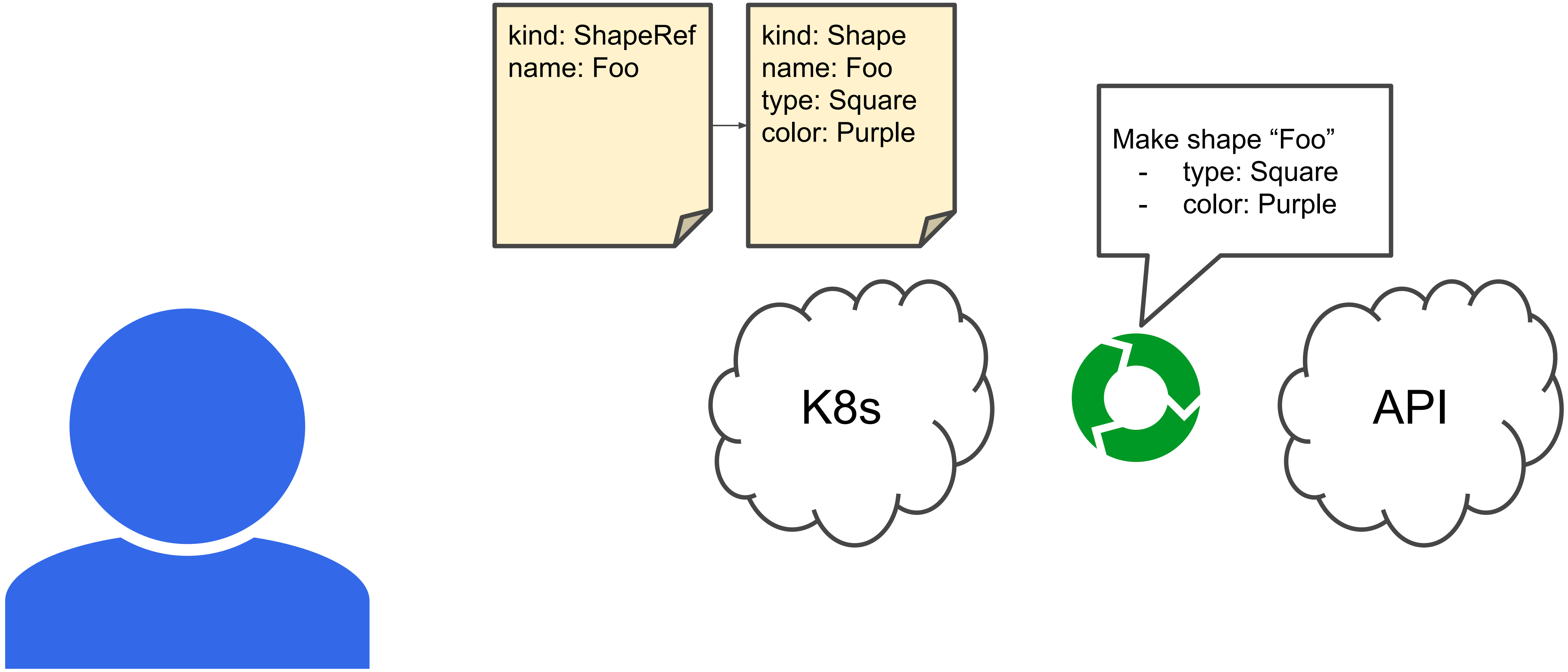
```
kubectl create shape "Foo"
  -    type: Square
  -    color: Purple
```

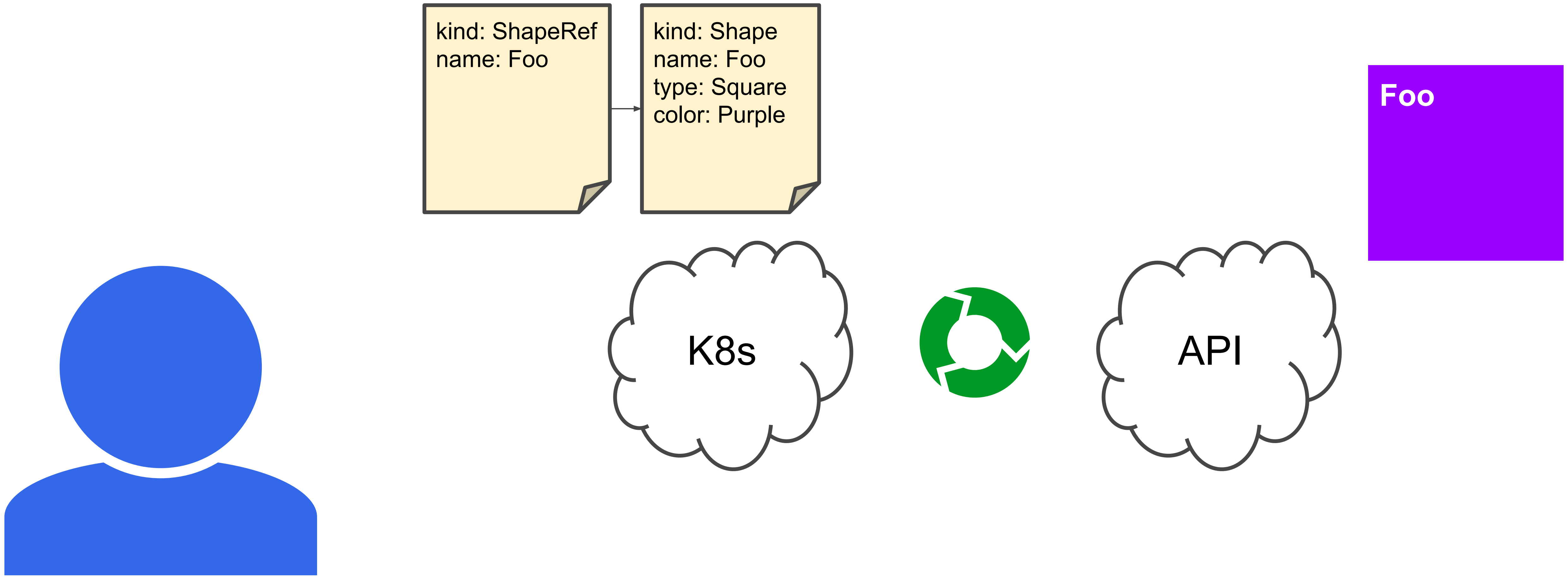


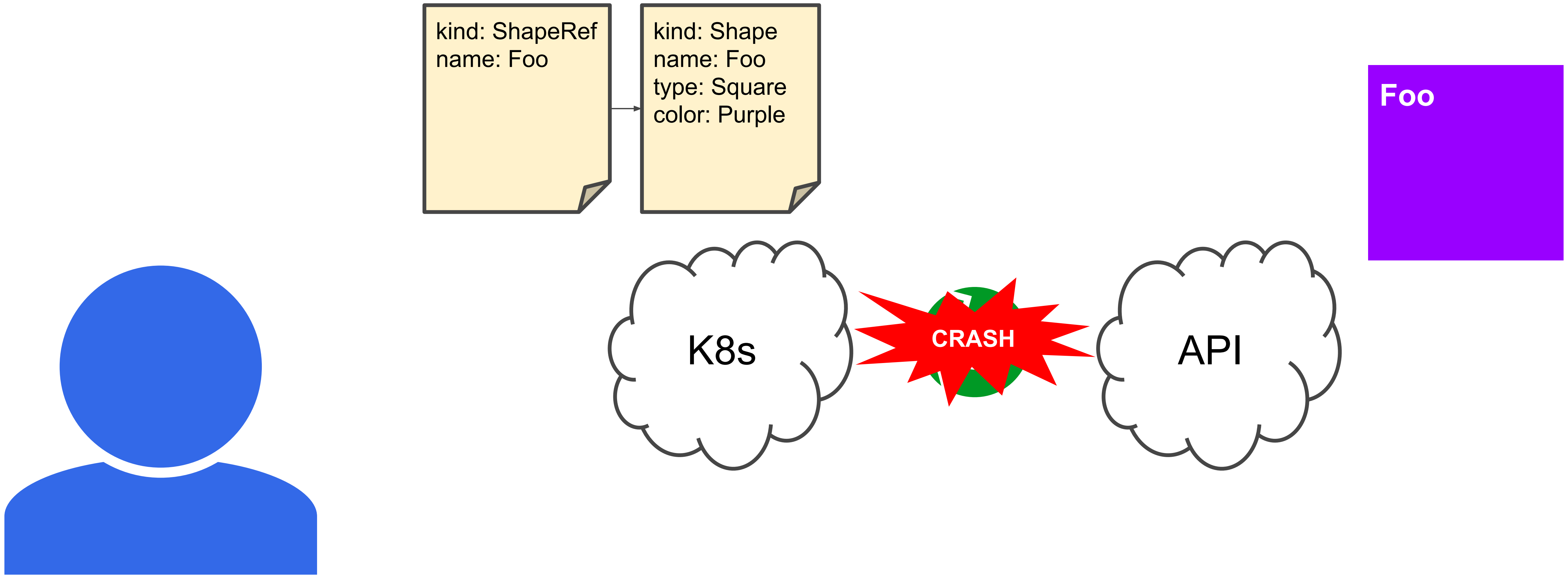
```
kind: Shape
name: Foo
type: Square
color: Purple
```

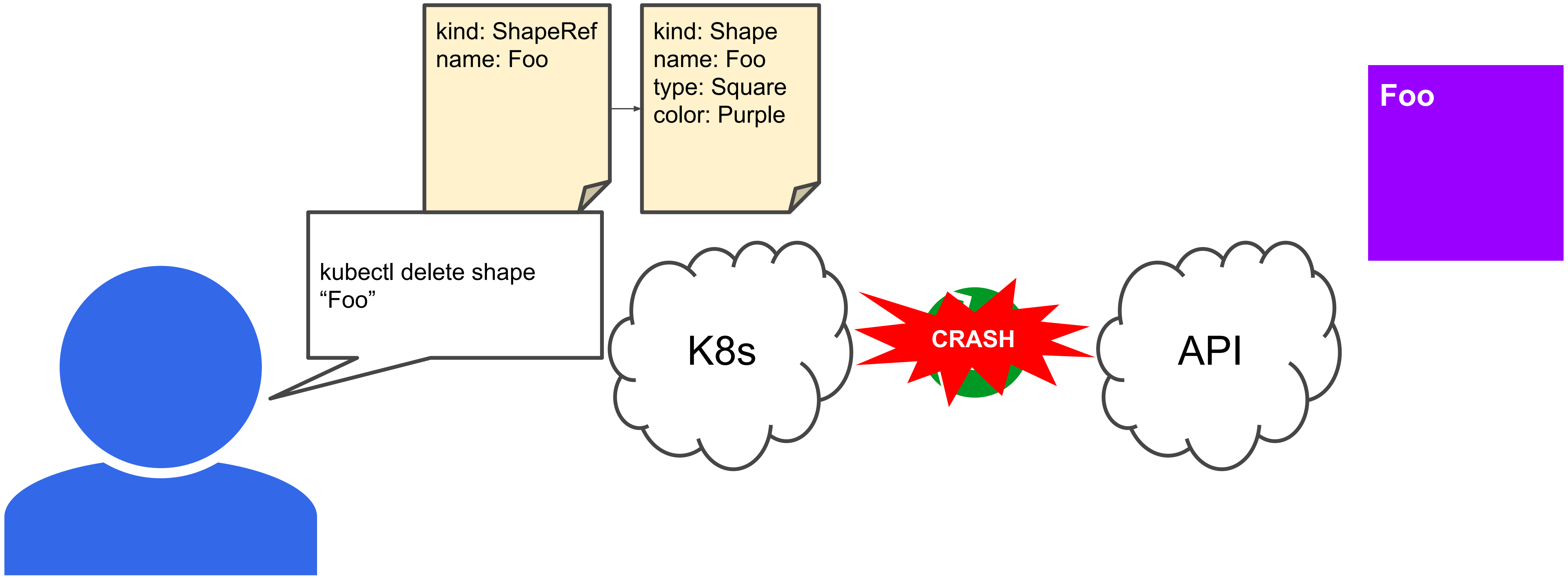

kind: Shape
name: Foo
type: Square
color: Purple
Create ShapeRef “Foo”
- ownerReference: Foo
K8s
API

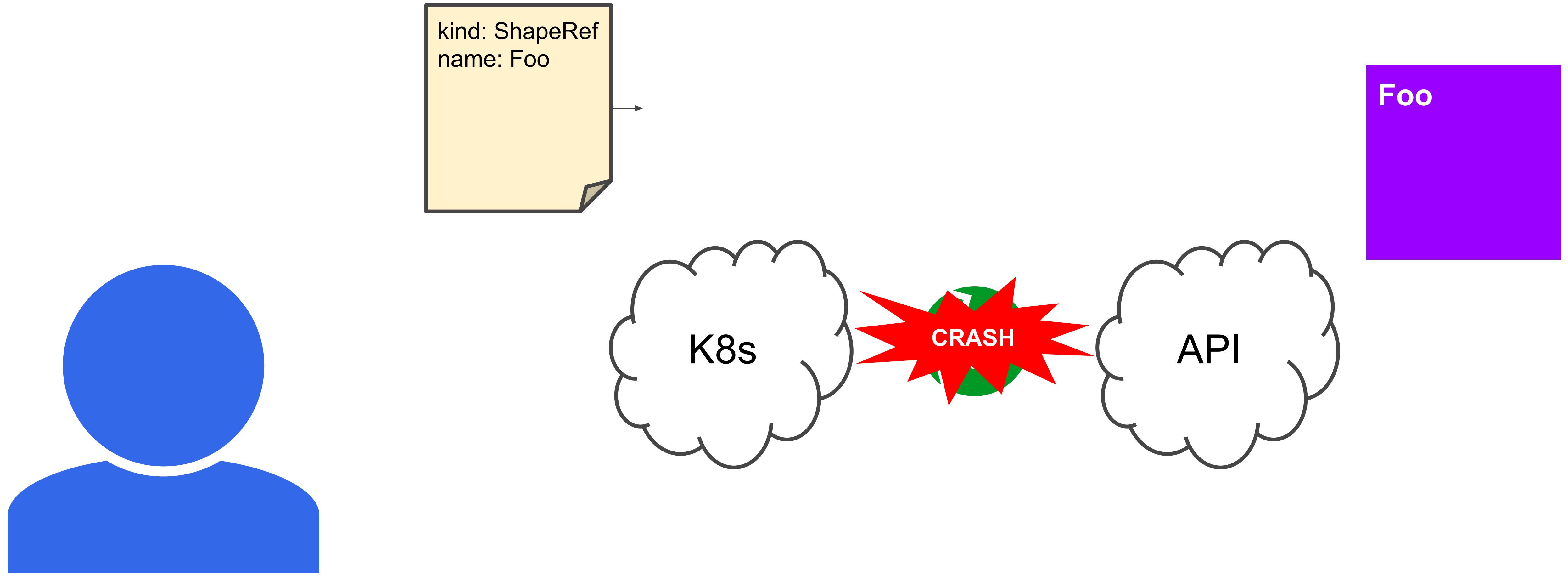

kind: ShapeRef
name: Foo
kind: Shape
name: Foo
type: Square
color: Purple
K8s
API

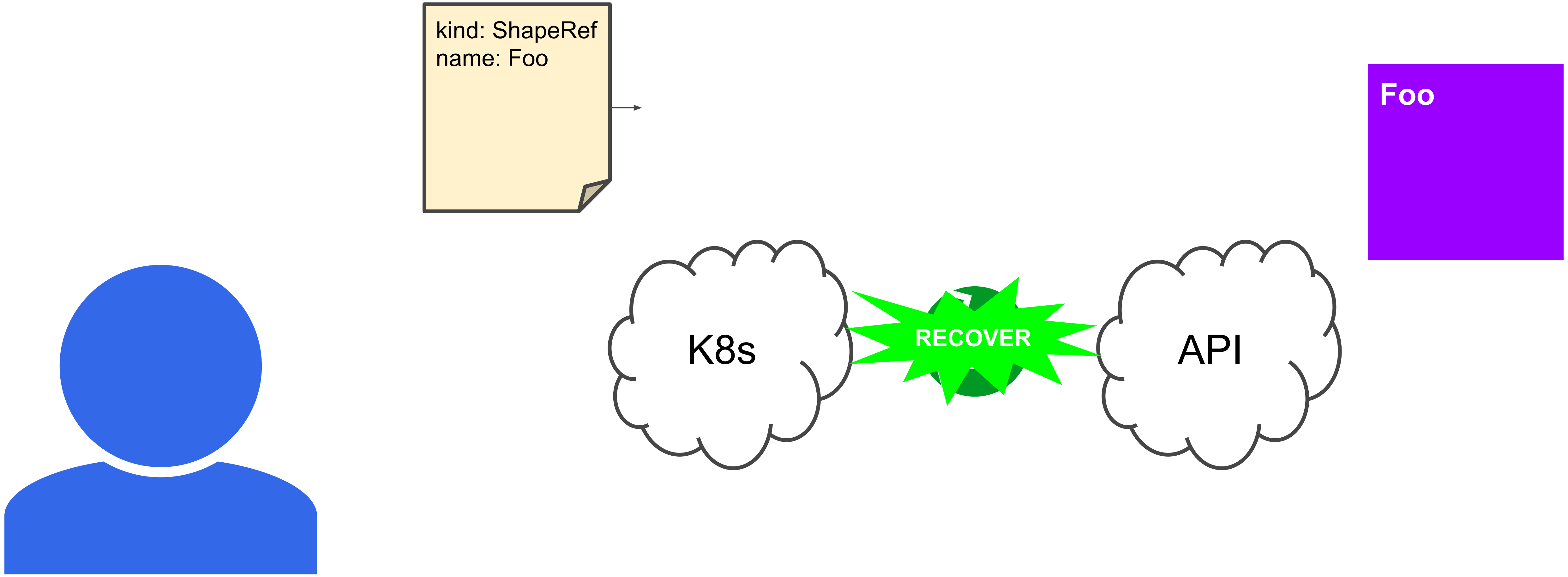

kind: ShapeRef
name: Foo
kind: Shape
name: Foo
type: Square
color: Purple
Make shape “Foo”
- type: Square
- color: Purple
K8s
API

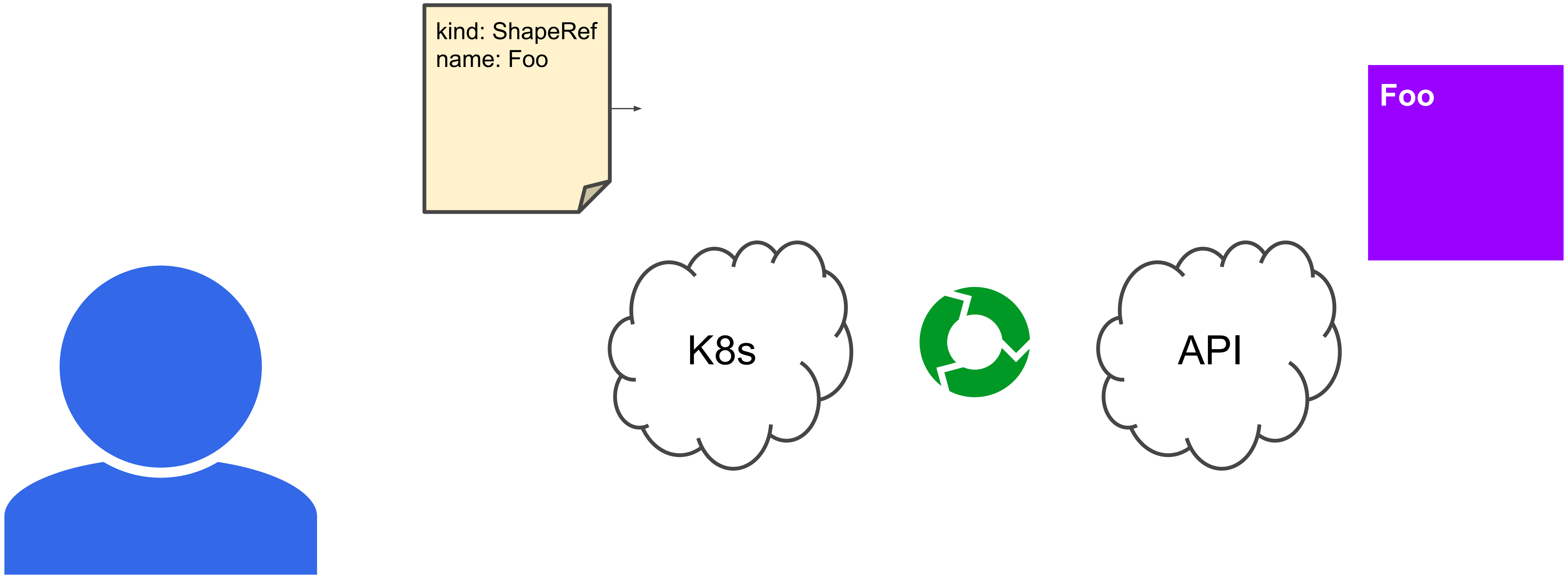

kind: ShapeRef
name: Foo
kind: Shape
name: Foo
type: Square
color: Purple
K8s
API
Foo


kind: ShapeRef
name: Foo
kind: Shape
name: Foo
type: Square
color: Purple
K8s
CRASH
API
Foo

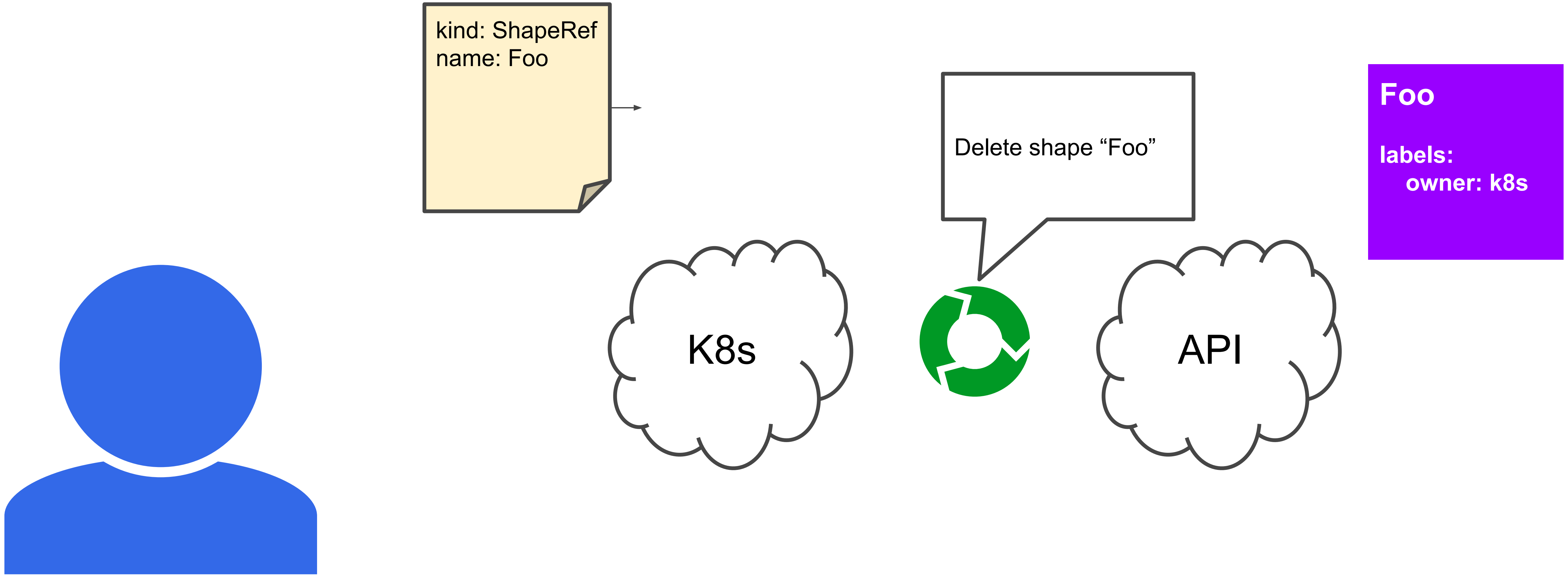

kind: ShapeRef
name: Foo
kind: Shape
name: Foo
type: Square
color: Purple
Foo
kubectl delete shape "Foo"
K8s
CRASH
API

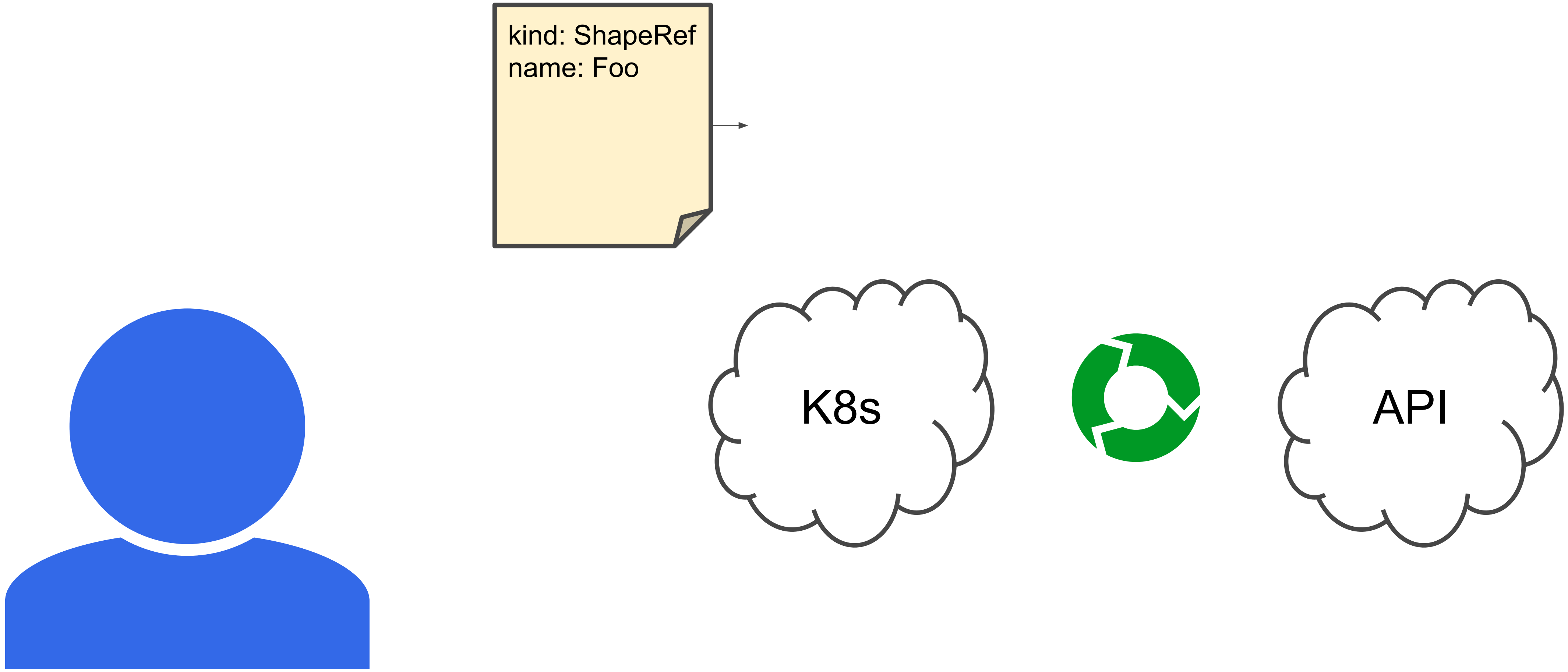

kind: ShapeRef
name: Foo
Foo
K8s
CRASH
API

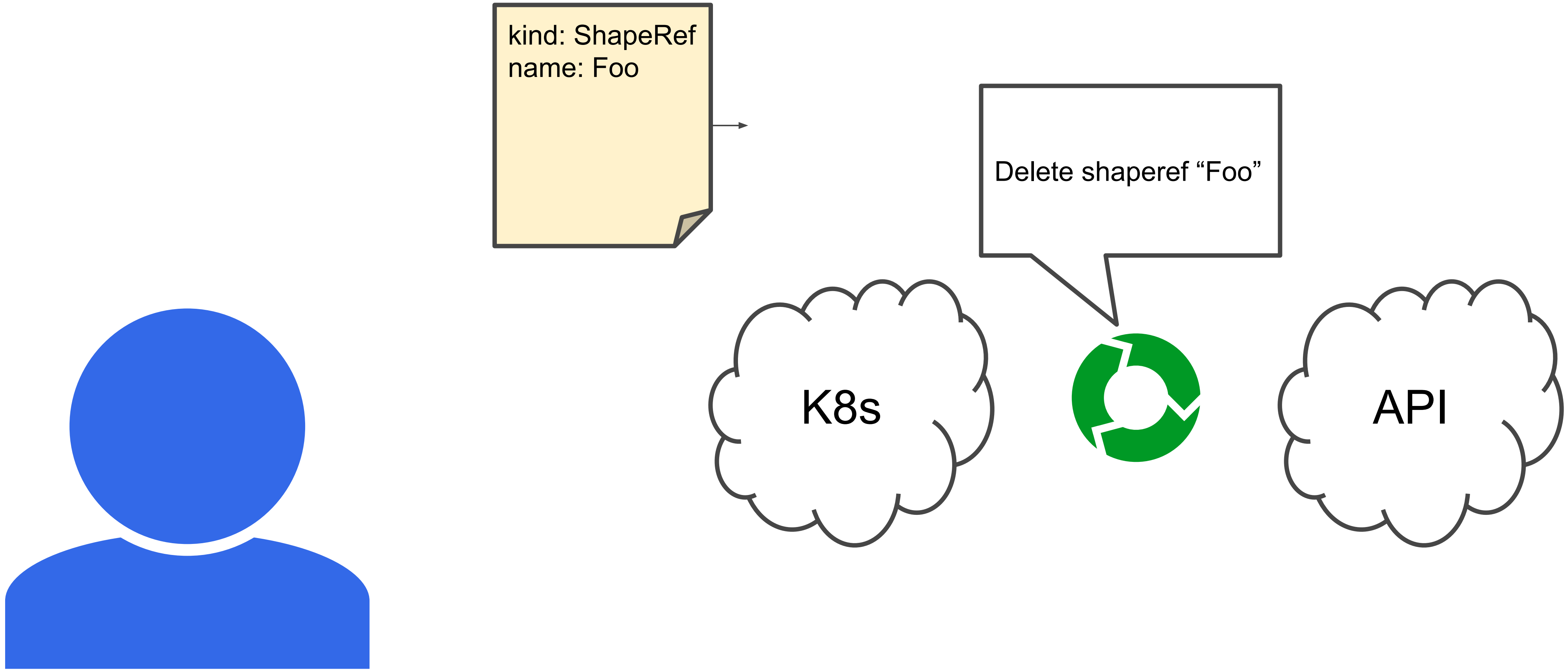

kind: ShapeRef
name: Foo
Foo
K8s
RECOVER
API

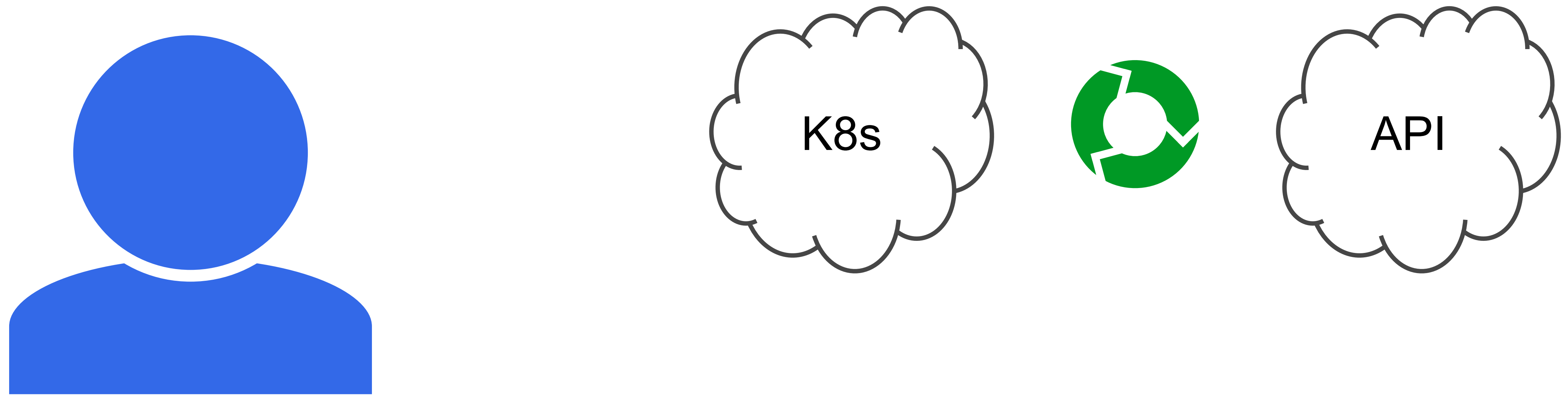

kind: ShapeRef
name: Foo
K8s
API
Foo

The controller did not observe the deletion of the Shape, but it does observe the dangling ShapeRef.


kind: ShapeRef
name: Foo
K8s
Delete shape “Foo”
API
Foo
labels:
owner: k8s


kind: ShapeRef
name: Foo
K8s
API


kind: ShapeRef
name: Foo
Delete shaperef “Foo”
K8s
API


K8s
API

In most of these mechanisms, there’s some amount of “you broke it, you bought it”.

If a user deletes the ShapeRef or removes the finalizer or edits the underlying metadata, the linkage can be broken.

You broke it, you get to keep the pieces.