KubeCon | CloudNativeCon

North America 2018

Life of a Kubernetes Watch Event

Haowei Cai (roycaihw@github), Google
Wenjia Zhang (wenjiaswe@github), Google

About us

Haowei Cai(@roycaihw)

Software Engineer for Google Cloud. He is one of the owners of Kubernetes Python client library and an active Kubernetes SIG API Machinery contributor.

Wenjia Zhang (@wenjiaswe)

Software Engineer on Kubernetes team at Google. Active contributor for Kubernetes and etcd open source projects.

Agenda

- **What** is a Kubernetes Watch Event?
- **Why** is Watch Event important for Kubernetes?
- **How** is the life of a Kubernetes Watch Event?
- **Key Takeaways**

KubeCon | CloudNativeCon

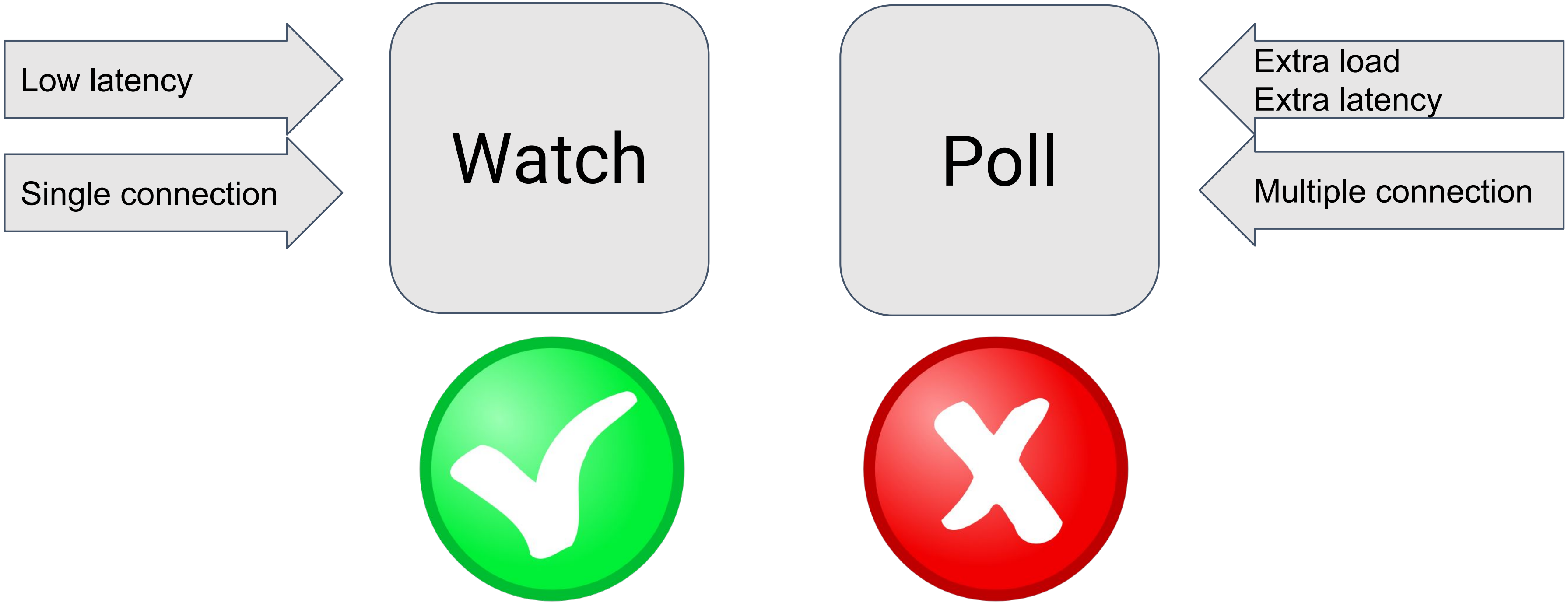
North America 2018

What is a Kubernetes Watch Event?

What is Watch?

Watch is an incremental change notification feed

Watch vs. Poll



Low latency

Single connection

Watch

Poll

Extra load
Extra latency

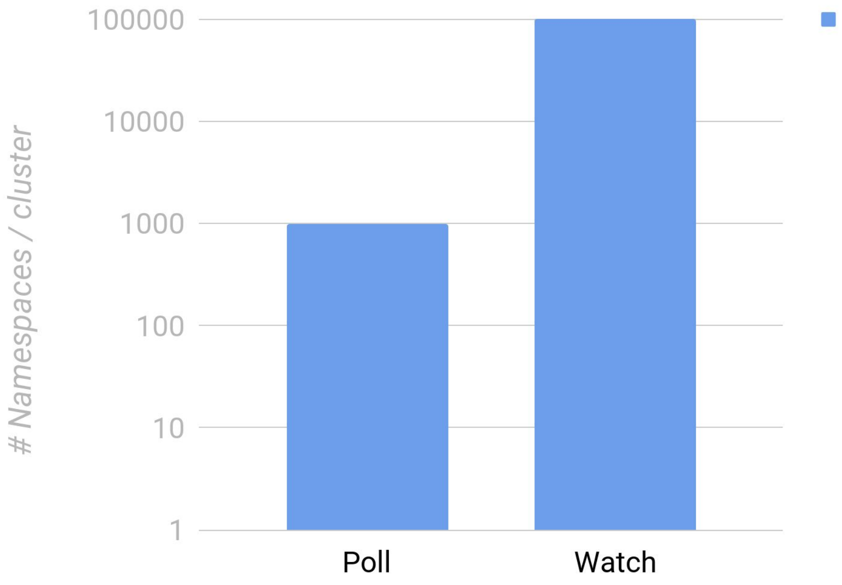
Multiple connection

Watch vs. Poll

Kubelet on nodes:

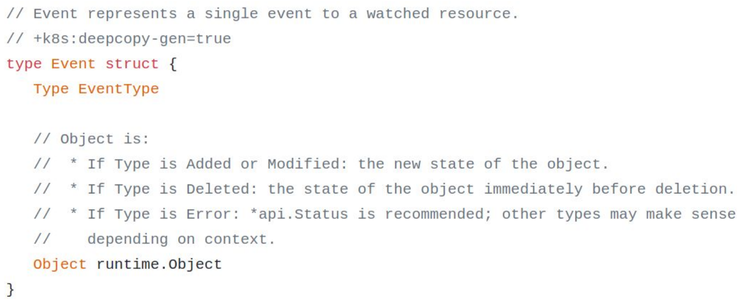
- Previous: periodically poll kube-apiserver for secrets and configmaps
- Now: watch individual secrets
- OSS PR: Kubelet watches necessary secrets/configmaps instead of periodic polling #64752

Scalability of Poll vs Watch



What is Event?

A Single event to a watched resource



```
// Event represents a single event to a watched resource.
// +k8s:deepcopy-gen=true
type Event struct {
   Type EventType

   // Object is:
   //  * If Type is Added or Modified: the new state of the object.
   //  * If Type is Deleted: the state of the object immediately before deletion.
   //  * If Type is Error: *api.Status is recommended; other types may make sense
   //    depending on context.
   Object runtime.Object
}
```

Watched resource: node	**Event:** add a node
Watched resource: pod	**Event**: modify a pod
Watched resource: replicaSet	**Event**: delete a replicaSet

KubeCon | CloudNativeCon

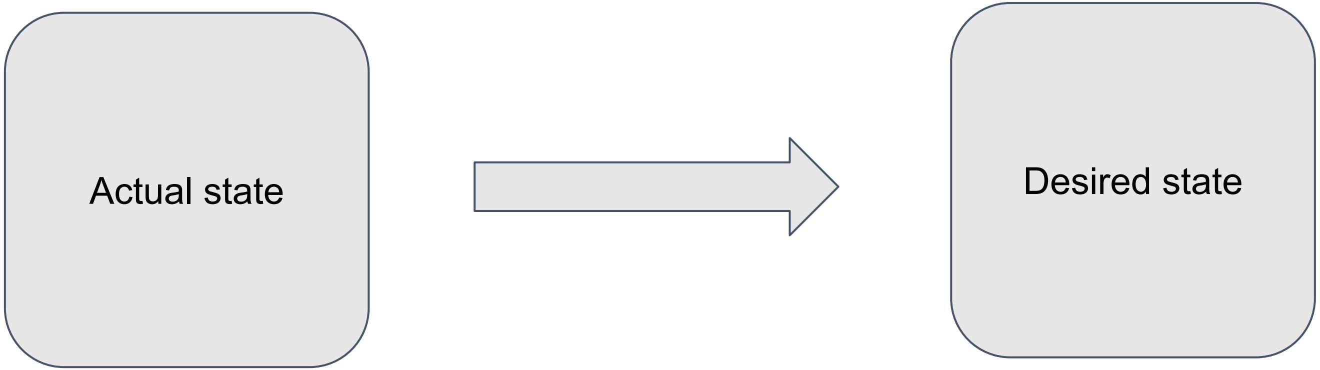
North America 2018

Why is Watch Event important for Kubernetes?

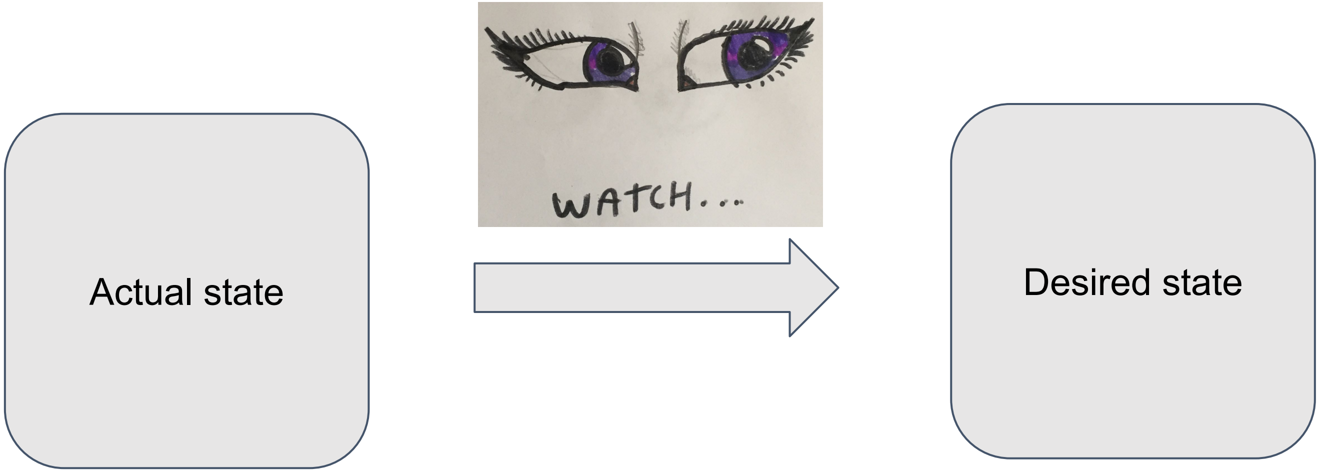
Kubernetes core design concept

Level Triggering and Soft Reconciliation

Declarative configuration



Declarative configuration



Actual state

Desired state

KubeCon | CloudNativeCon

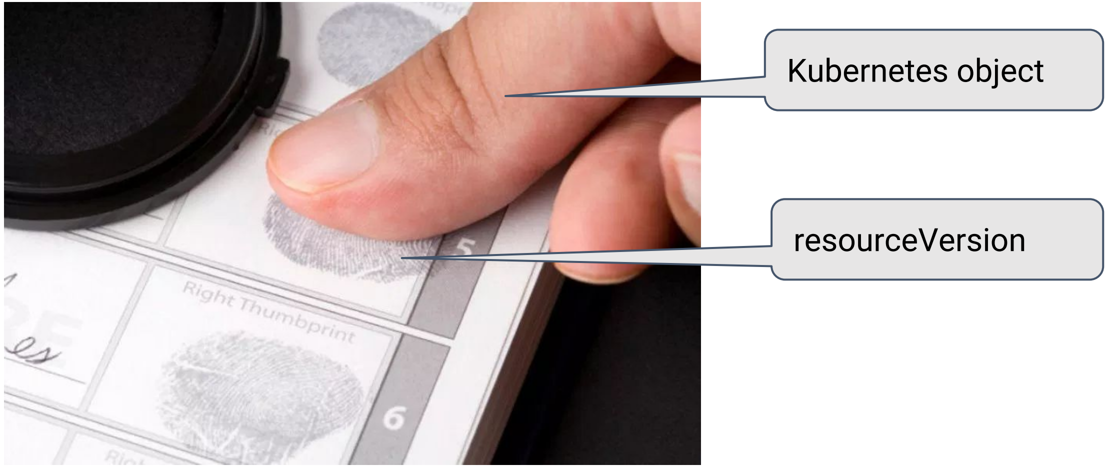
North America 2018

How is the lifecycle of a K8s watch event?

Fingerprint of events: resourceVersion

resourceVersion:

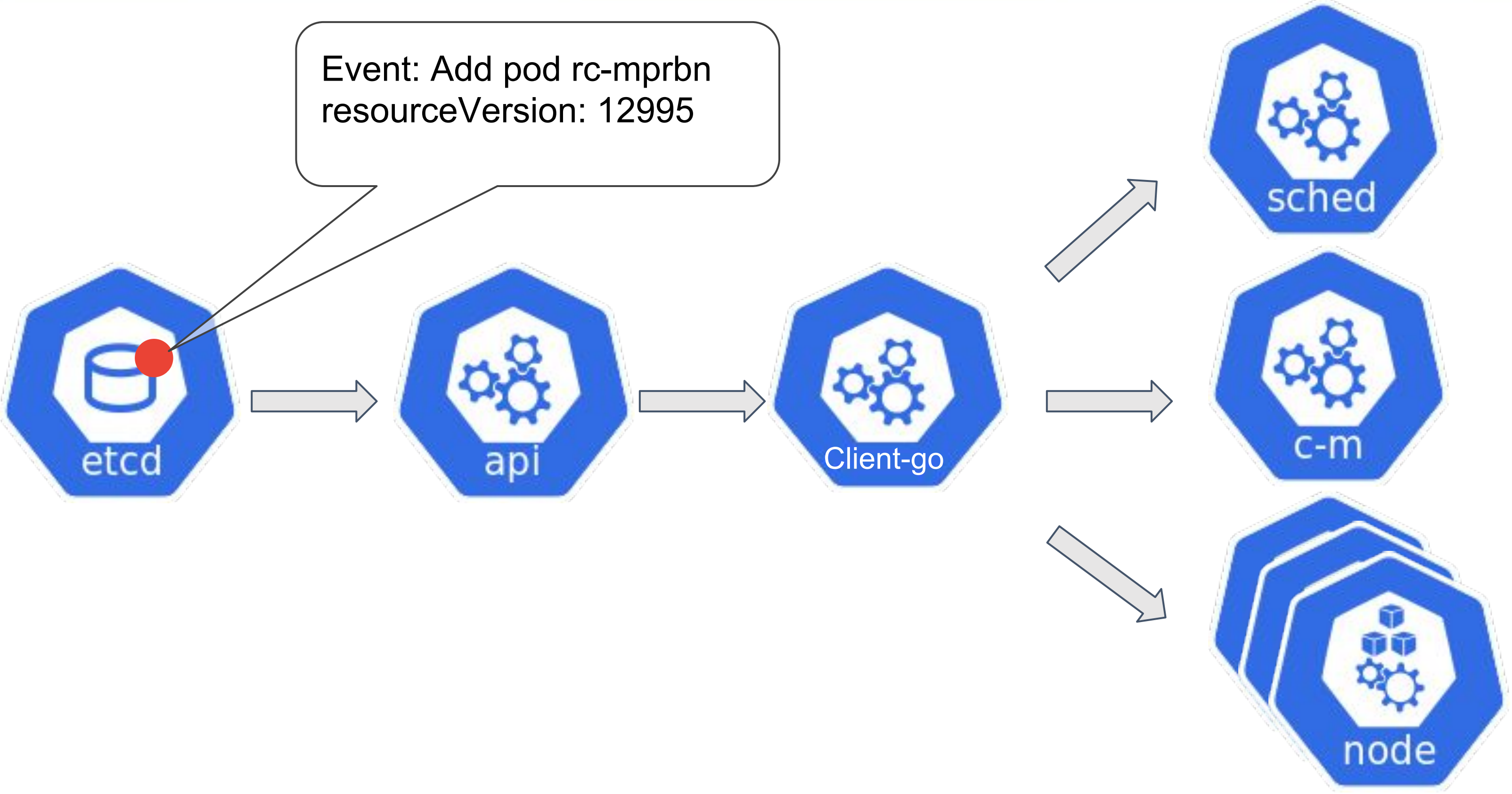
a string that identifies the common version of the objects returned by in a list. This value MUST be treated as opaque by clients and passed unmodified back to the server. A resource version is valid on a single kind of resource across namespaces.



Kubernetes object

resourceVersion

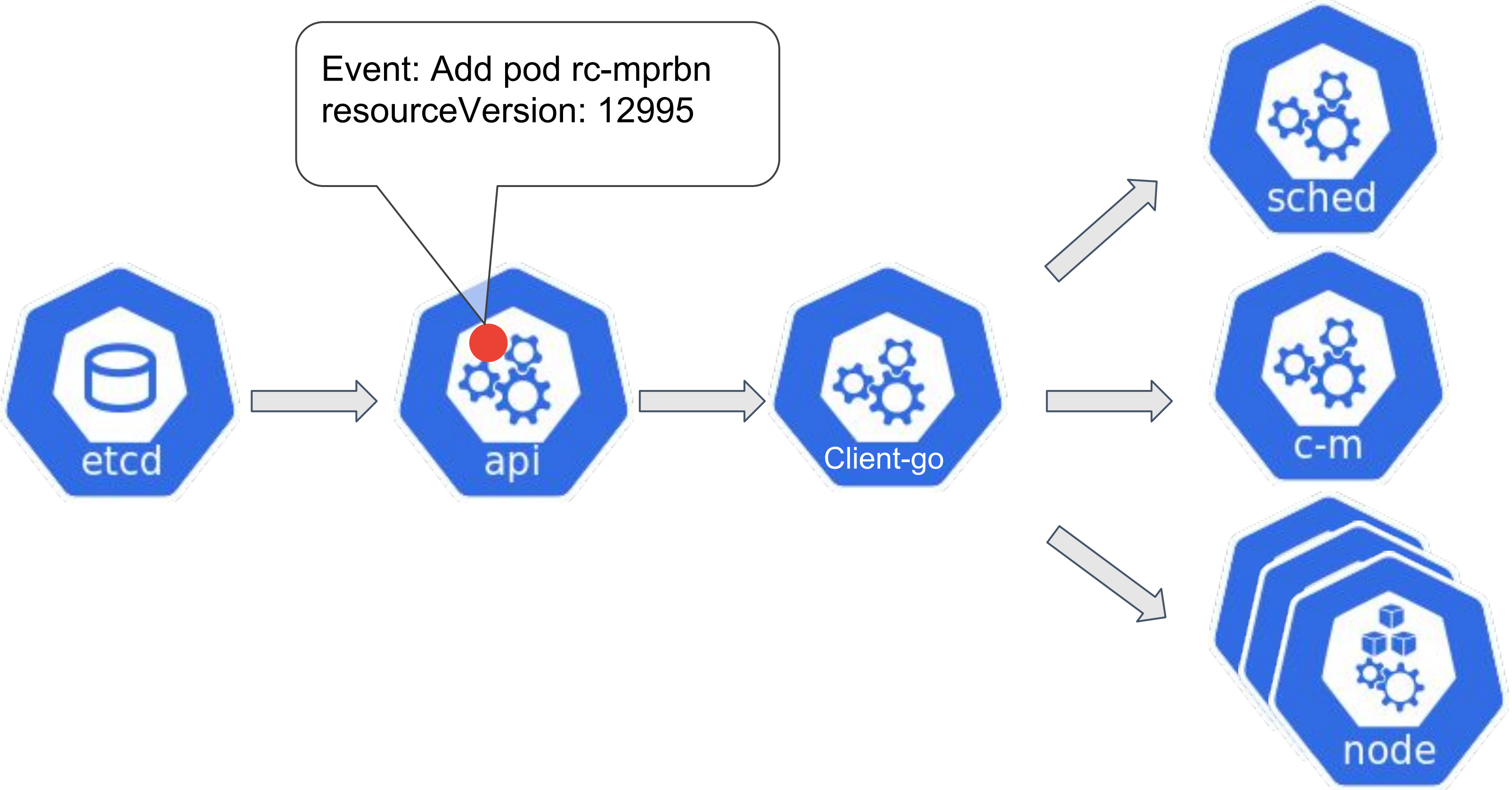
Lifecycle of a K8s watch event



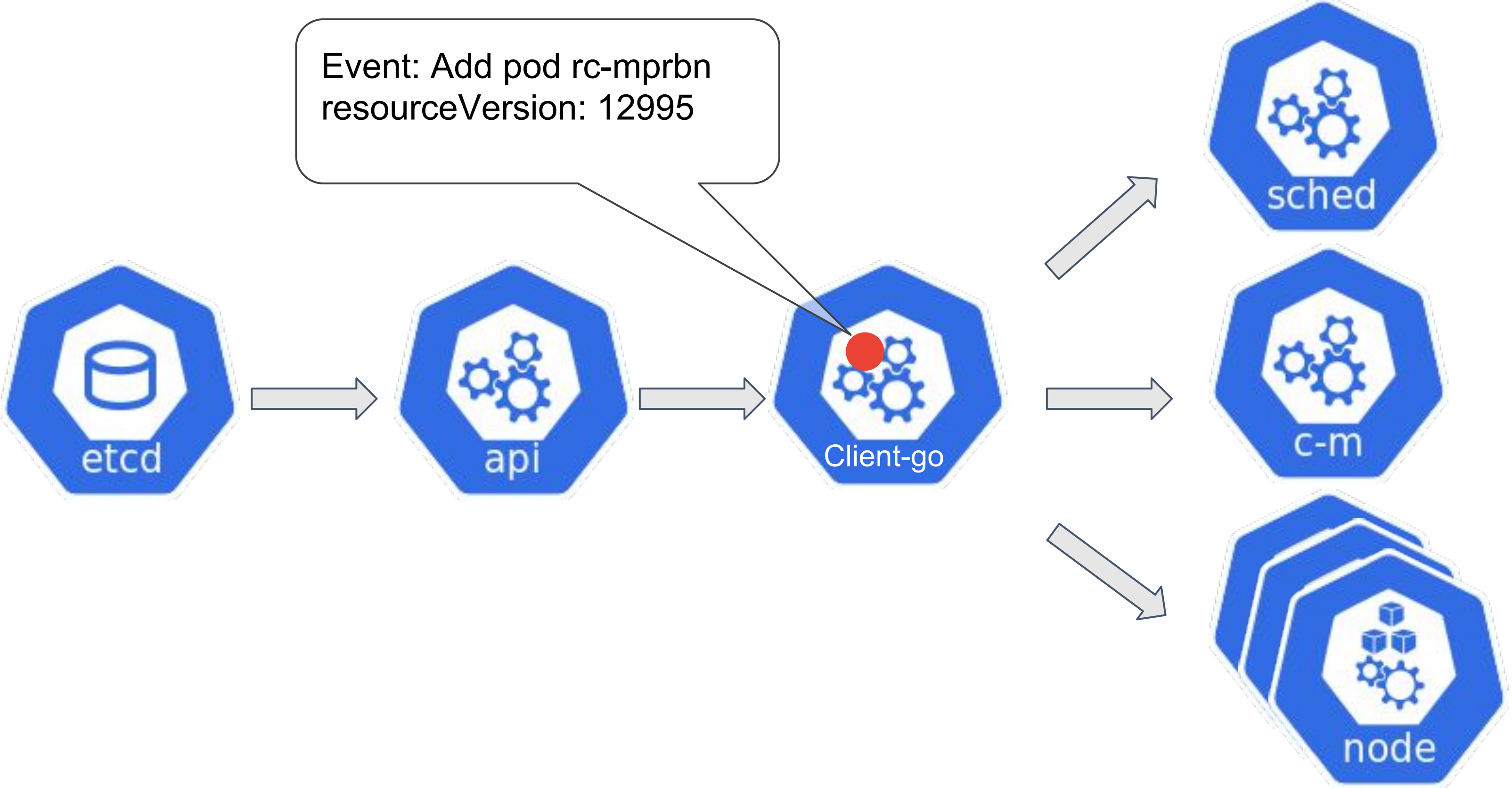
Event: Add pod rc-mprbn
resourceVersion: 12995

Lifecycle of a K8s watch event

Event: Add pod rc-mprbn
resourceVersion: 12995

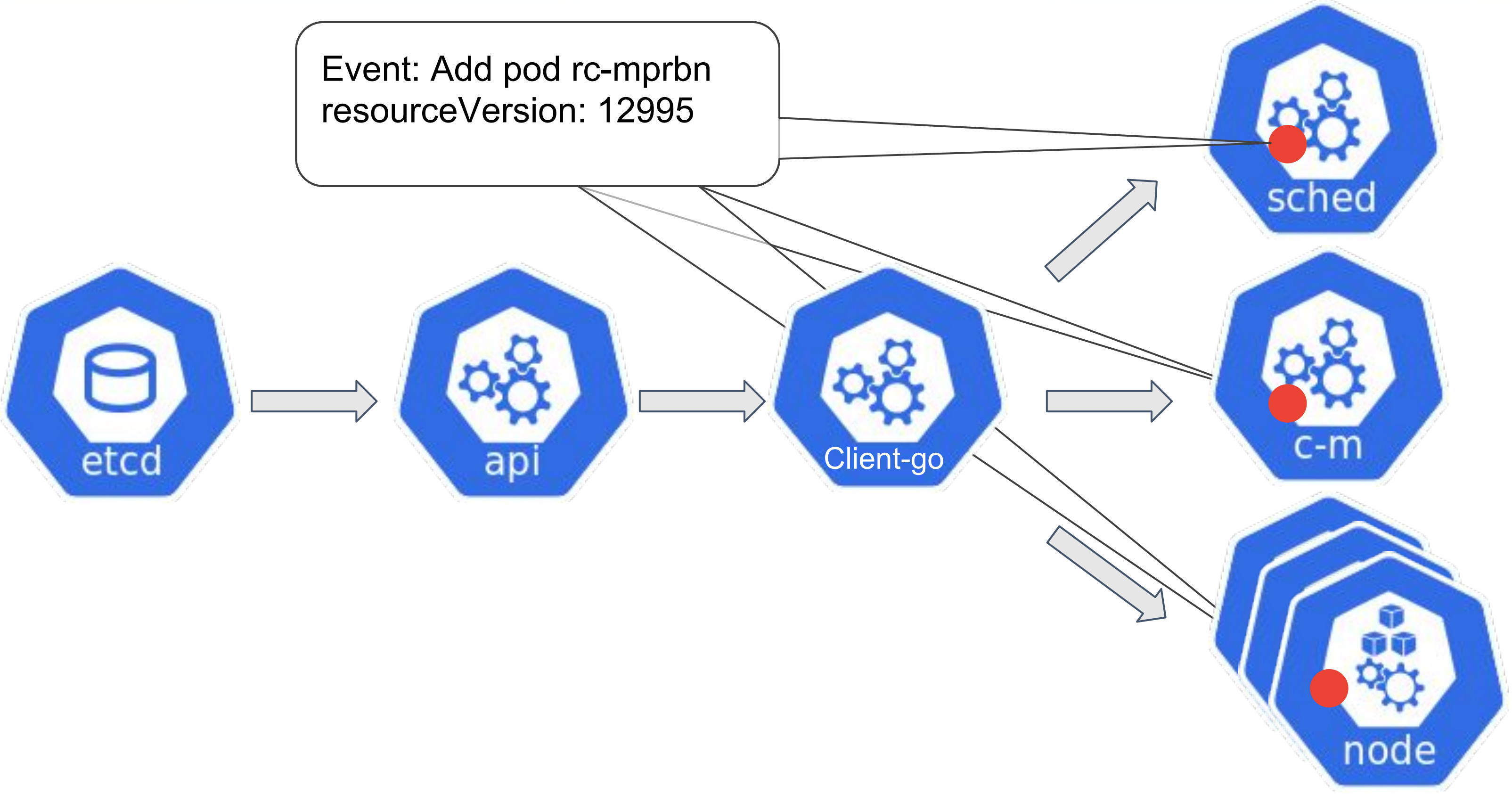


Lifecycle of a K8s watch event



Event: Add pod rc-mprbn
resourceVersion: 12995

Lifecycle of a K8s watch event



Event: Add pod rc-mprbn
resourceVersion: 12995

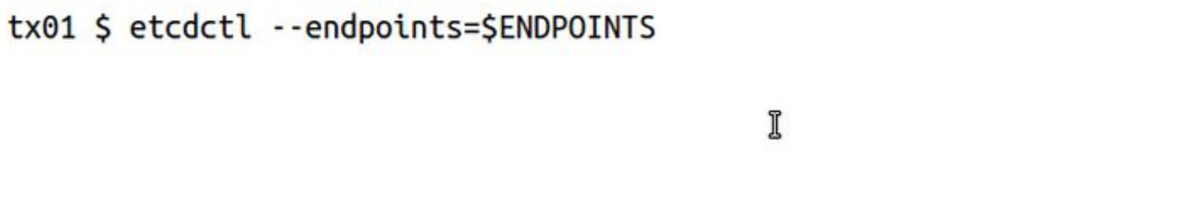
KubeCon | CloudNativeCon

North America 2018

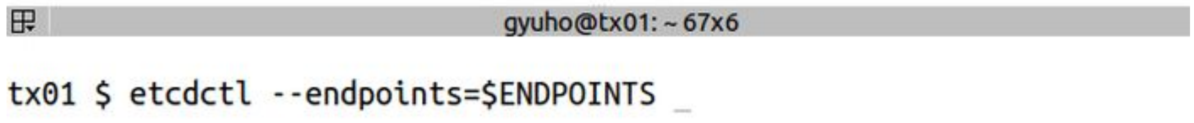
Watch Event on etcd

Watch in etcd

Etcd **watch** feature provides an event-based interface for asynchronously monitoring changes to keys.



```
tx01 $ etcdctl --endpoints=$ENDPOINTS
```



```
tx01 $ etcdctl --endpoints=$ENDPOINTS _
```

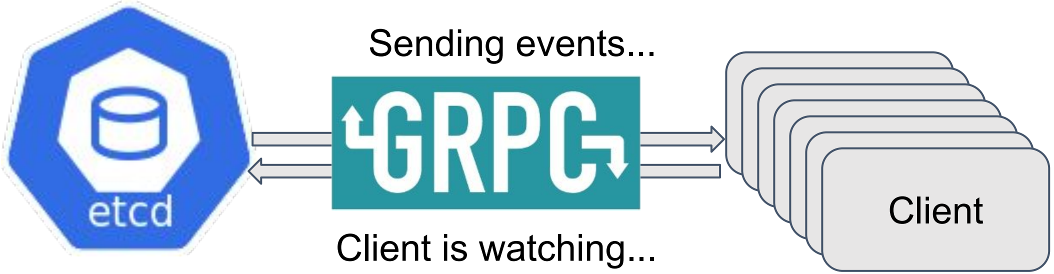
Revision

Revision (etcd) = resourceVersion (apiserver)

- The key space maintains multiple revisions.
- Each atomic mutative operation creates a new revision on the key space.
- All data held by previous revisions remains unchanged.
- If the store is compacted to save space, revisions before the compact revision will be removed.
- Revisions are monotonically increasing over the lifetime of a cluster.

*this is a current implementation detail of the etcd storage layer, and does not guarantee resourceVersion will be numeric or monotonically increasing in the future

Watch event on etcd



Watches make three guarantees about events:

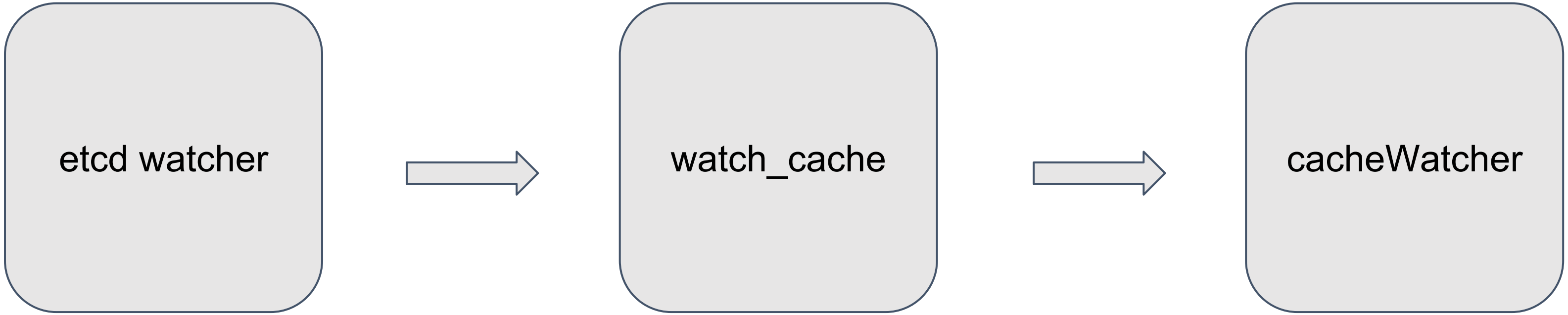
- Ordered - events are ordered by revision
- Reliable - a sequence of events will never drop any subsequence of events
- Atomic - a list of events is guaranteed to encompass complete revisions

KubeCon | CloudNativeCon

North America 2018

Watch Event on Kube Api-server

Watch Event on Kube Api-server

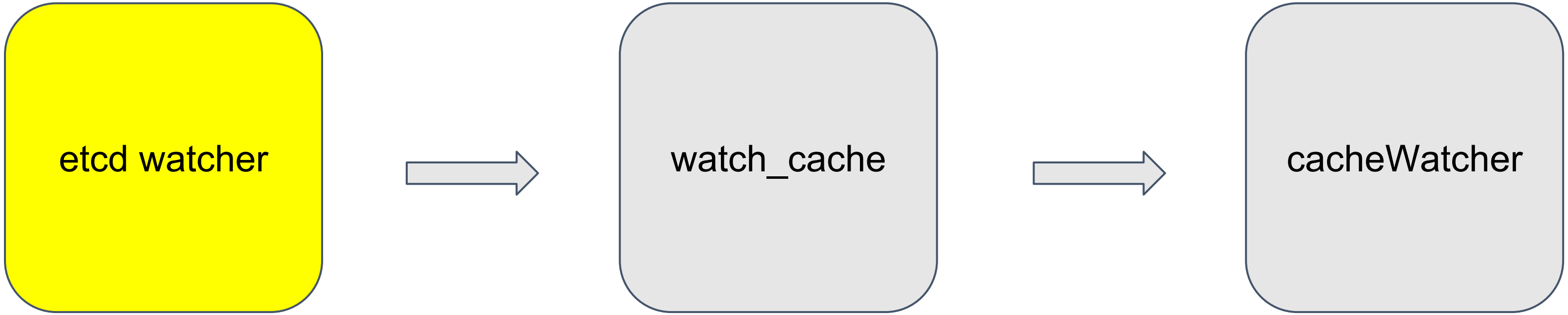


etcd watcher

watch_cache

cacheWatcher

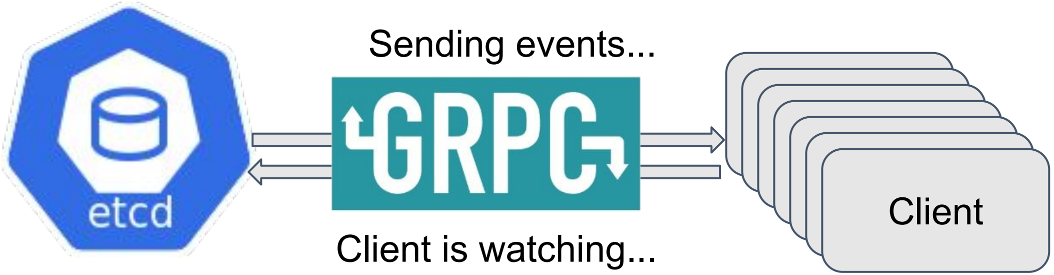
Watch Event on Kube Api-server



etcd watcher

watch_cache

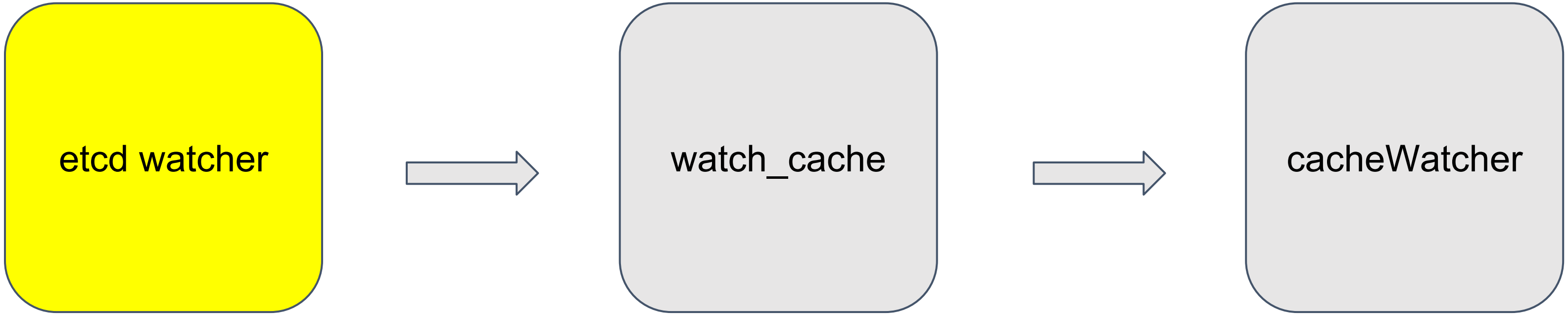
cacheWatcher



Sending events...

Client is watching...

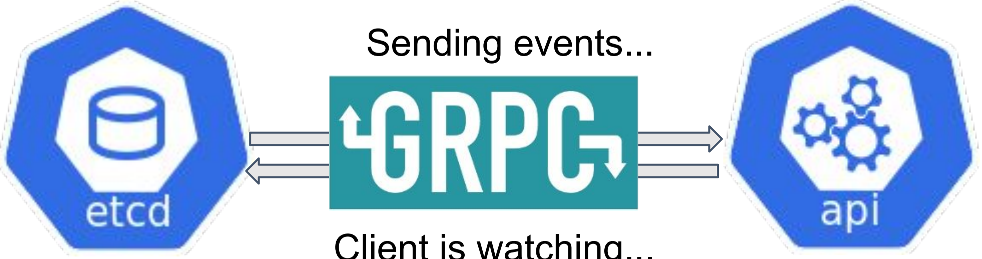
Watch Event on Kube Api-server



etcd watcher

watch_cache

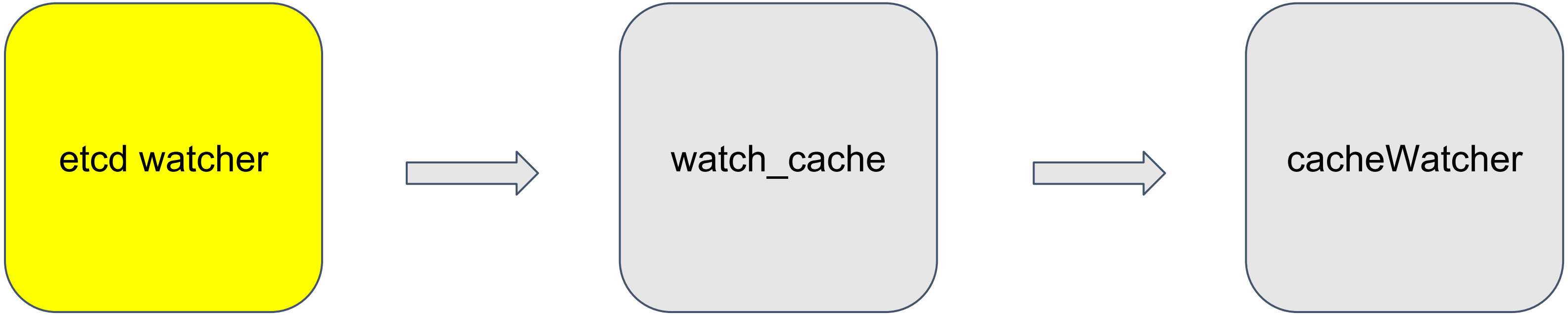
cacheWatcher



Sending events...

Client is watching...

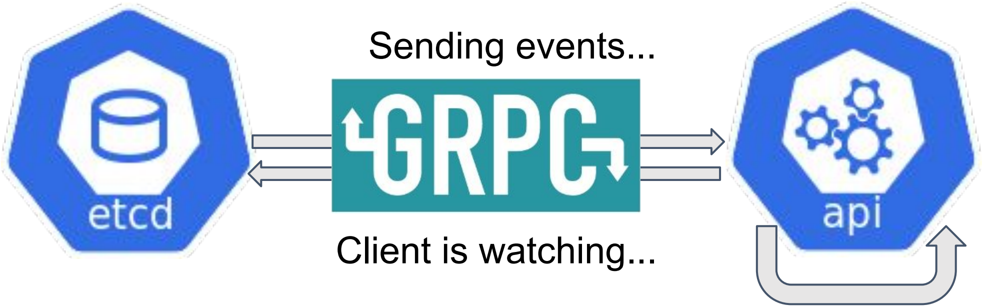
Watch Event on Kube Api-server



etcd watcher

watch_cache

cacheWatcher

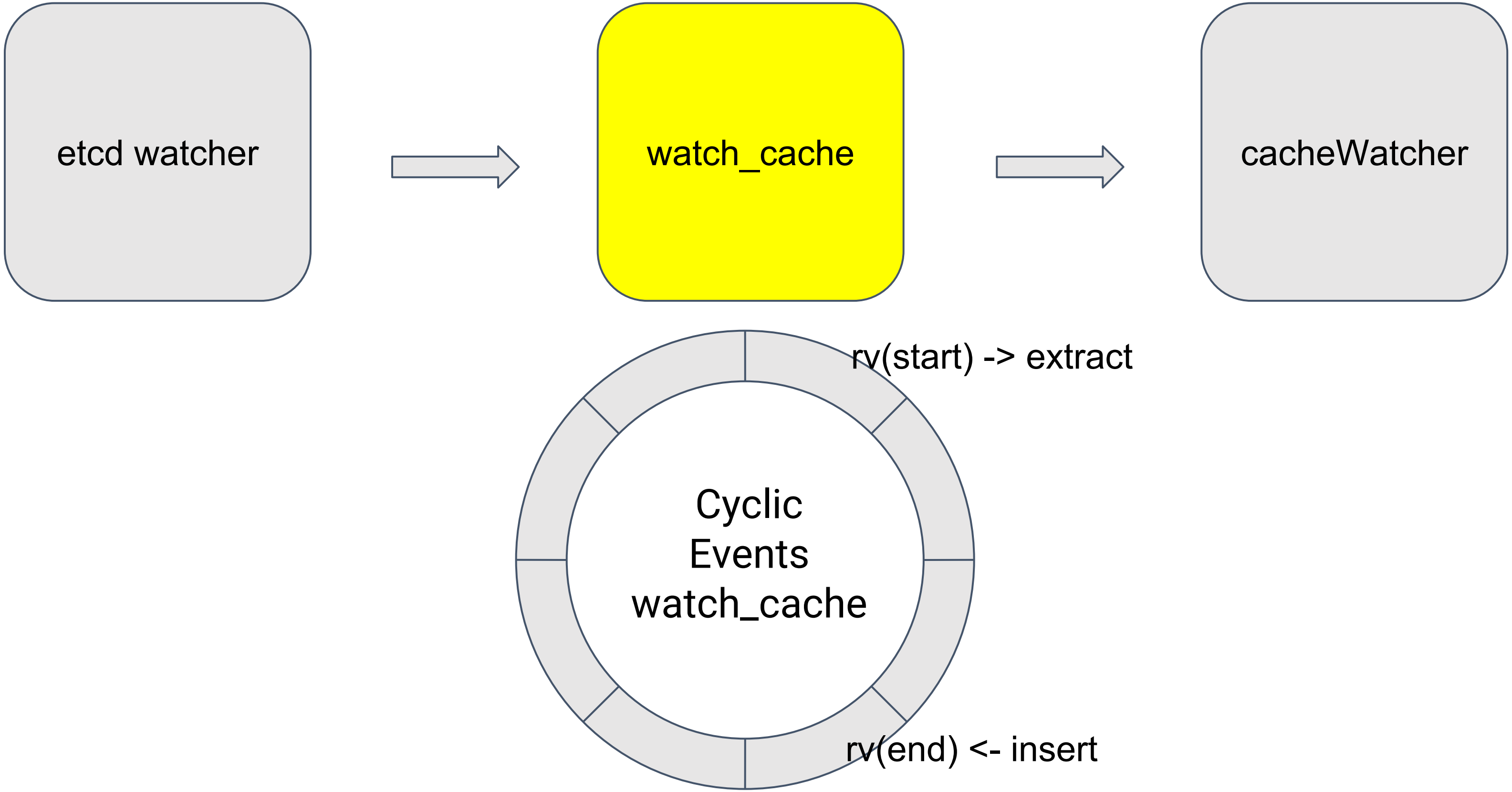


Sending events...

Client is watching...

- Flow control…
- etcd3 Event -> APIserver Event...

Watch Event on Kube Api-server



etcd watcher

watch_cache

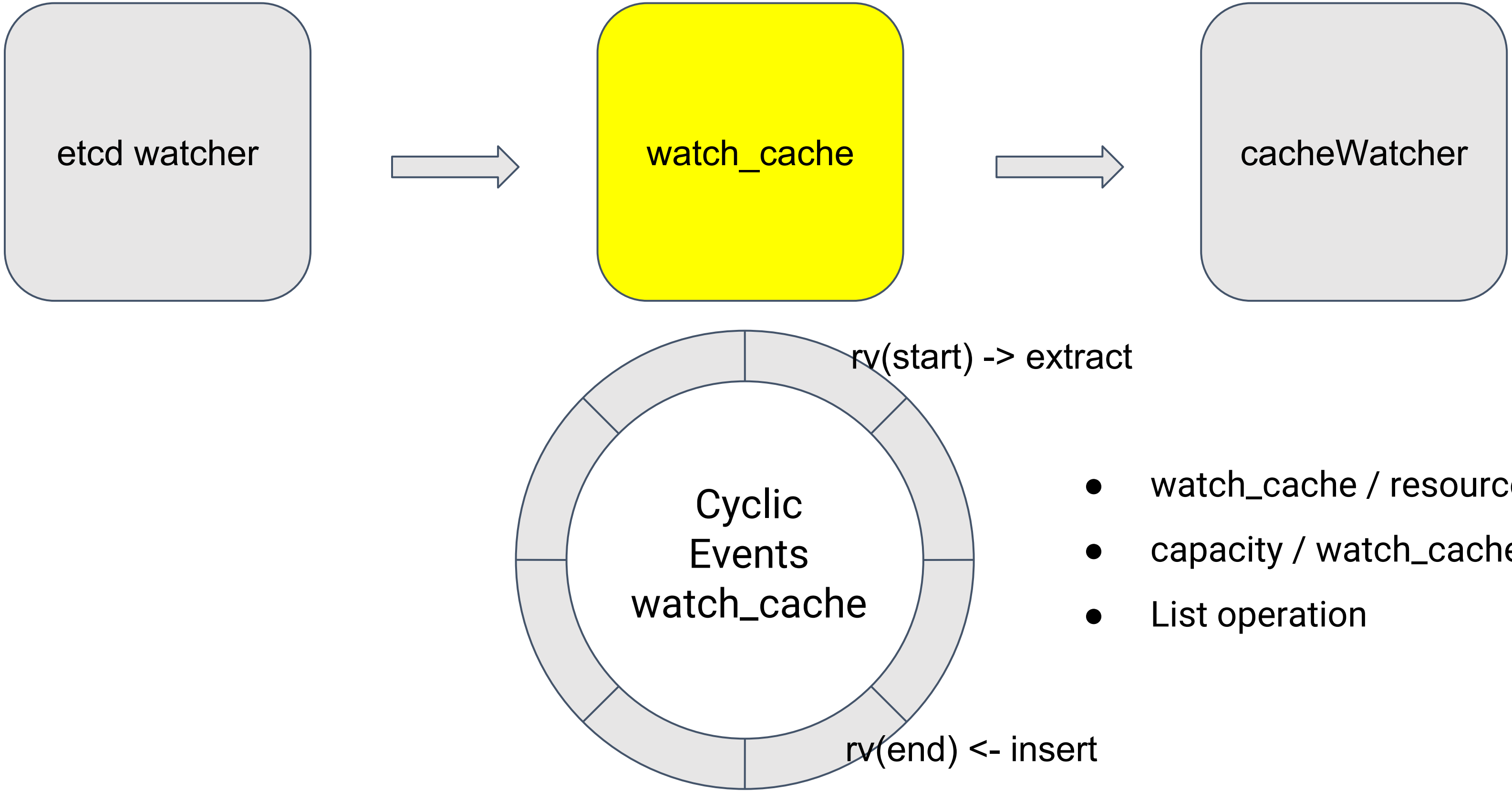
cacheWatcher

Cyclic Events watch_cache

rv(start) -> extract

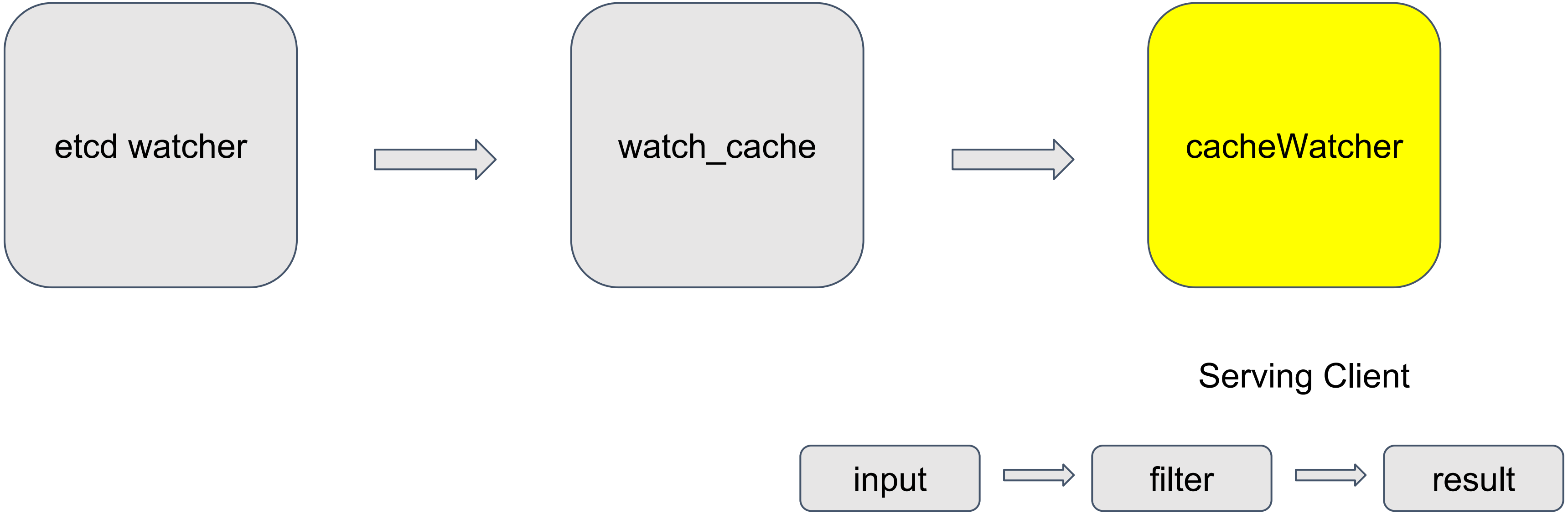
rv(end) <- insert

Watch Event on Kube Api-server



- watch_cache / resource type
- capacity / watch_cache
- List operation

Watch Event on Kube Api-server



etcd watcher → watch_cache → cacheWatcher

Serving Client

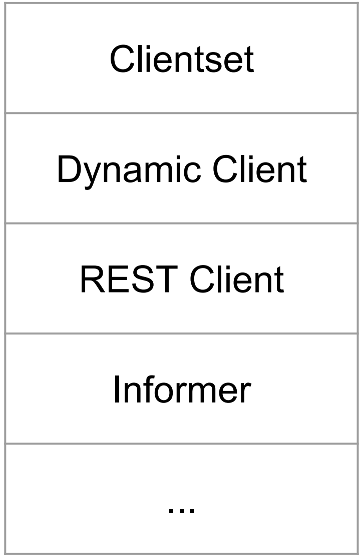
input → filter → result

KubeCon | CloudNativeCon

North America 2018

Watch Event in Client-go

What is Client-go?



Clientset

Dynamic Client

REST Client

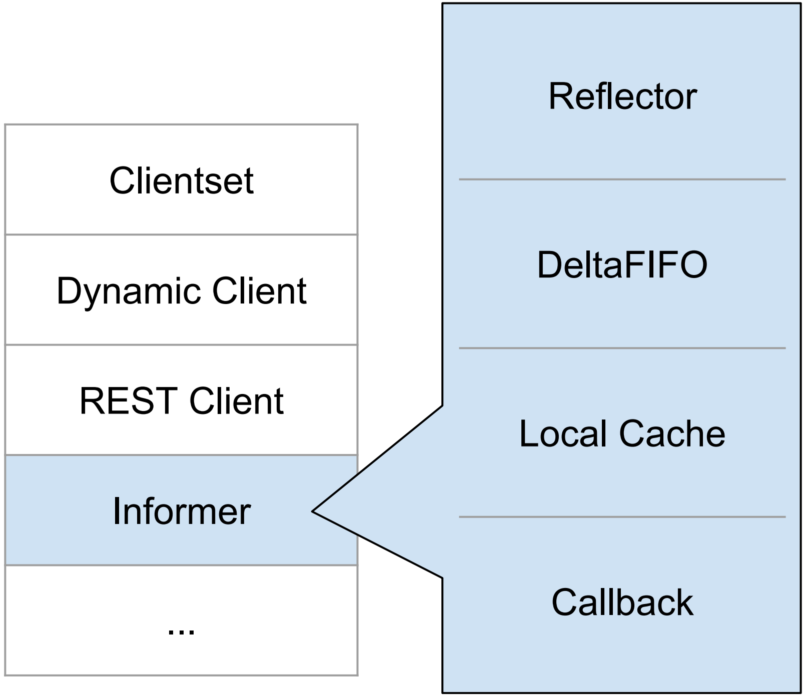
Informer

...

https://github.com/kubernetes/client-go

- Go clients for talking to a kubernetes cluster
- Used by Kubernetes itself

What is Informer?



Clientset

Dynamic Client

REST Client

Informer

...

Reflector

DeltaFIFO

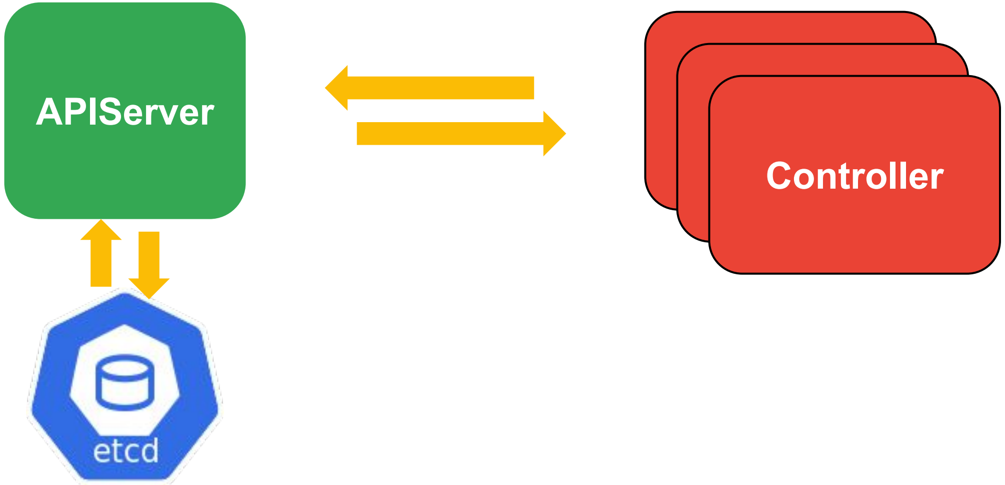
Local Cache

Callback

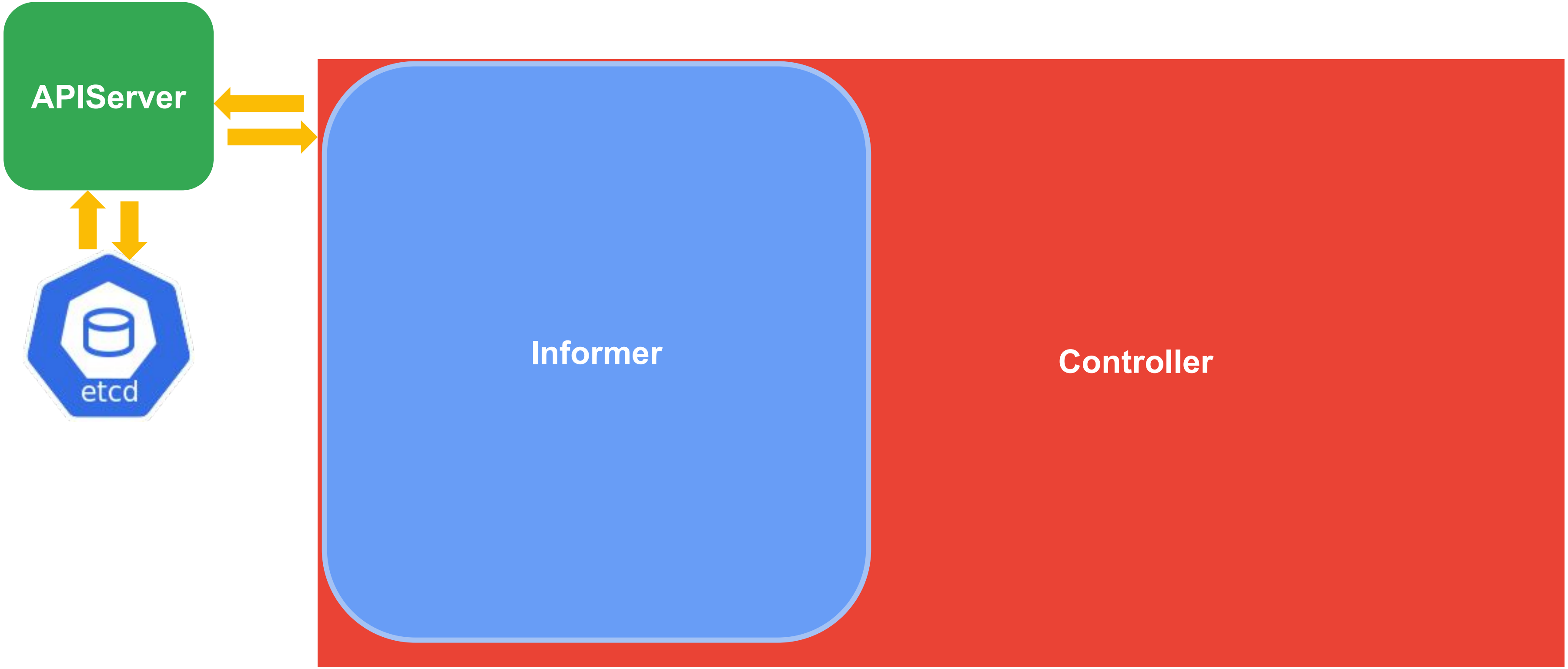
k8s.io/client-go/tools/cache
k8s.io/client-go/informers

- Useful component for building event-oriented controllers
- Used by control plane controllers, kubelet, etc.
- Reflector used by kube-apiserver watch cache

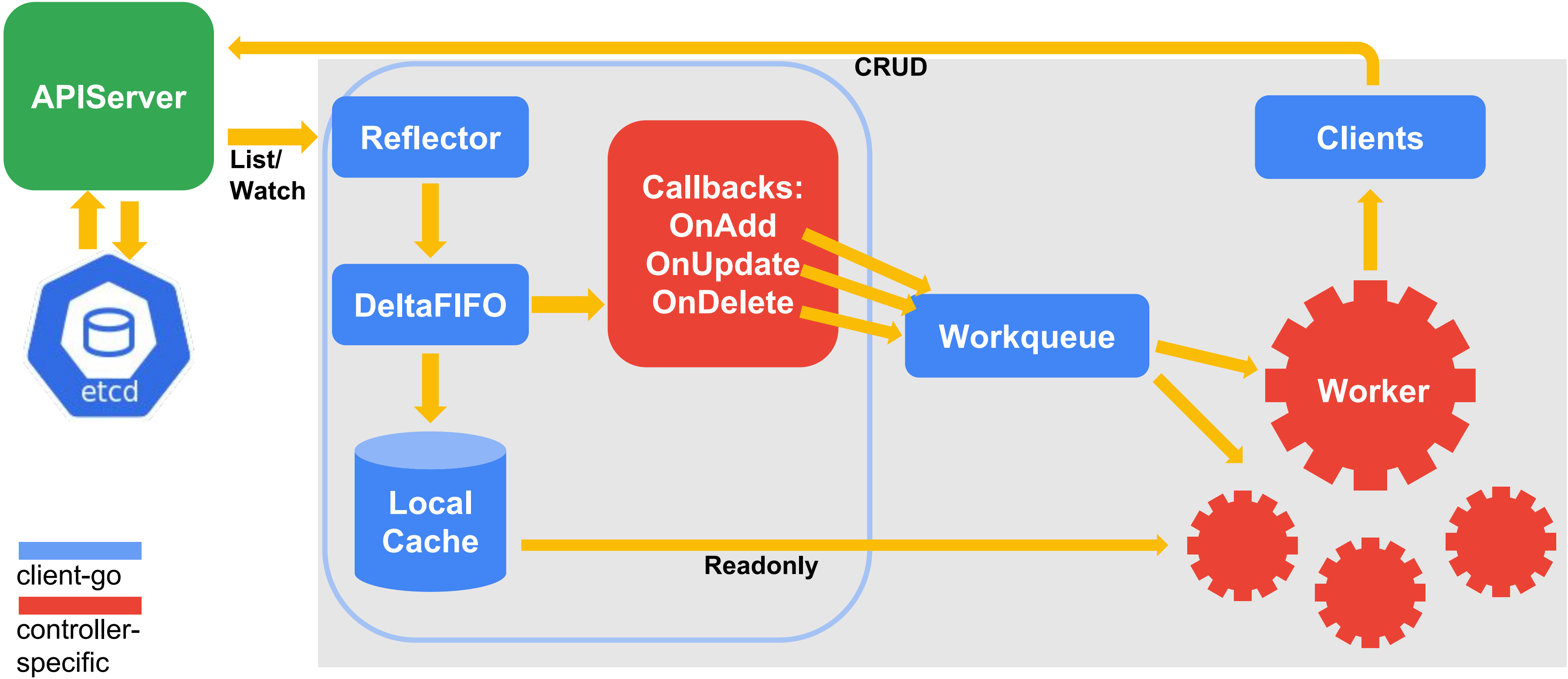
Kubernetes controller workflow



Kubernetes controller workflow



Kubernetes controller workflow



APIServer

etcd

List/
Watch

Reflector

DeltaFIFO

Local Cache

Callbacks:
OnAdd
OnUpdate
OnDelete

Workqueue

Worker

Clients

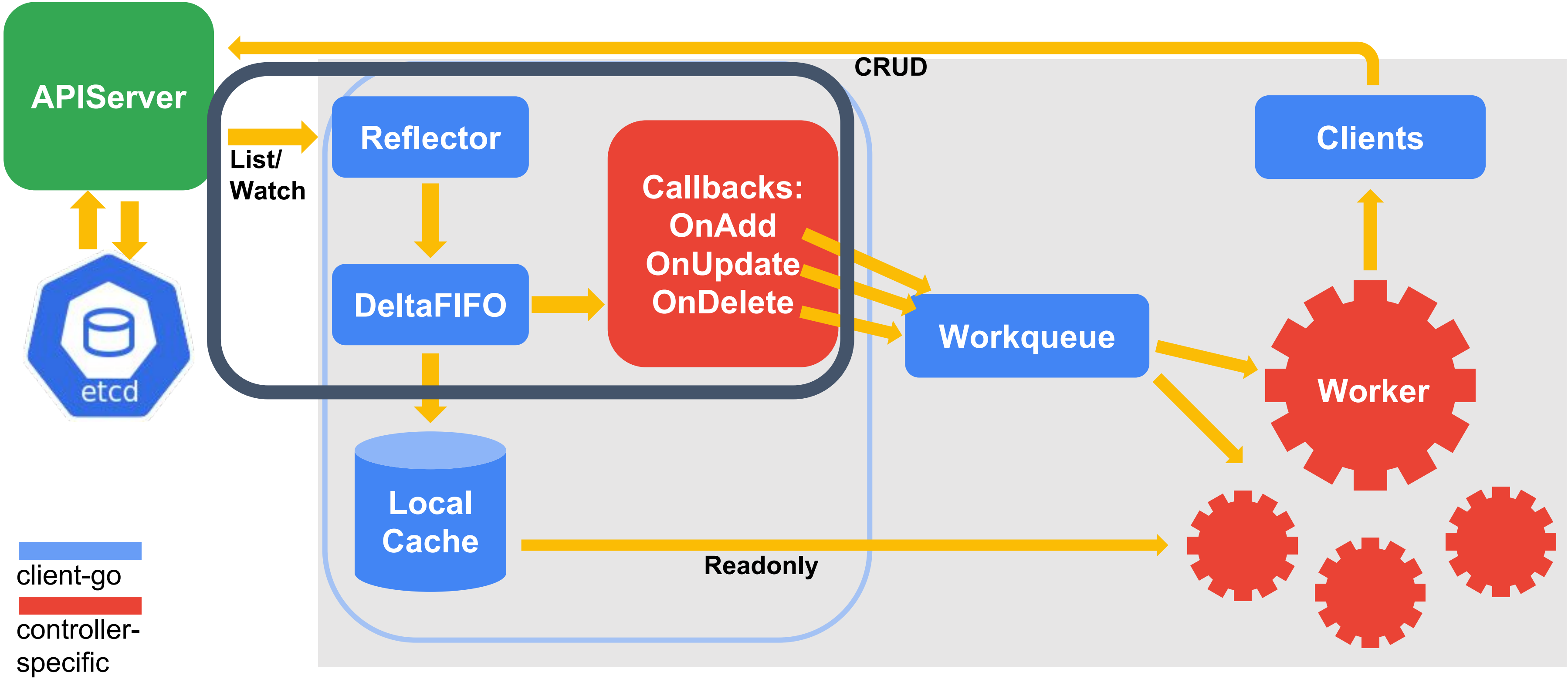
CRUD

Readonly

🟦 client-go

🟥 controller-specific

Kubernetes controller workflow



APIServer

etcd

List/
Watch

Reflector

DeltaFIFO

Local
Cache

Callbacks:
OnAdd
OnUpdate
OnDelete

Workqueue

Worker

Clients

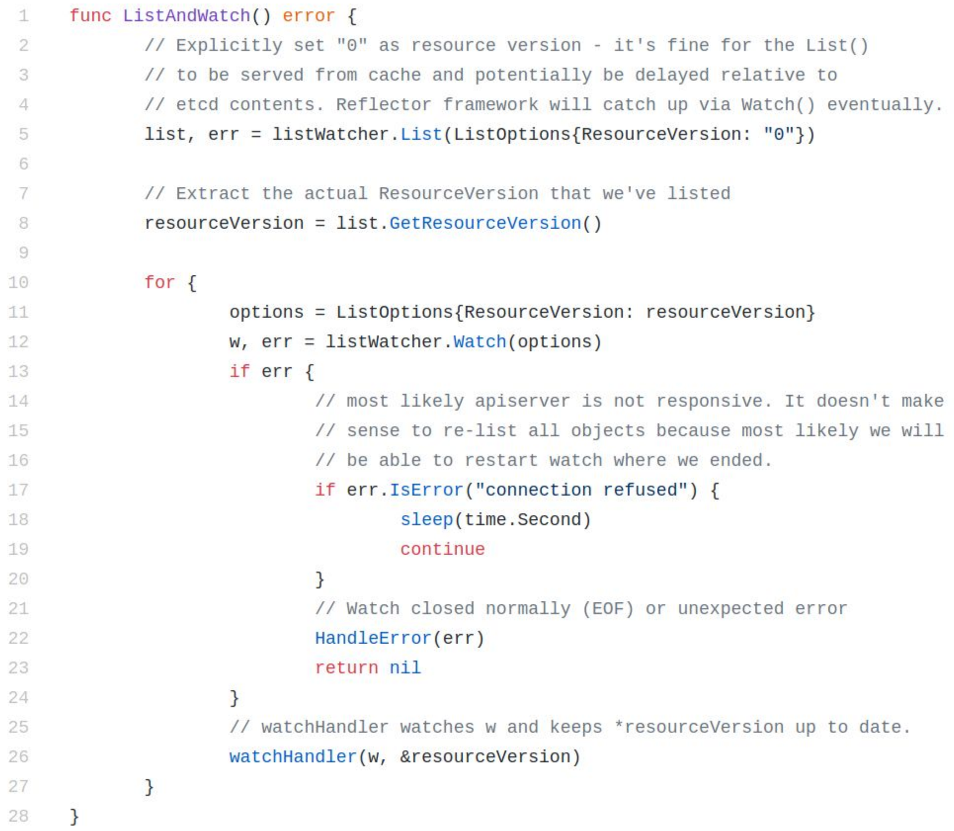
CRUD

Readonly

🟦 client-go

🟥 controller-specific

List and Watch



```go
func ListAndWatch() error {
        // Explicitly set "0" as resource version - it's fine for the List()
        // to be served from cache and potentially be delayed relative to
        // etcd contents. Reflector framework will catch up via Watch() eventually.
        list, err = listWatcher.List(ListOptions{ResourceVersion: "0"})

        // Extract the actual ResourceVersion that we've listed
        resourceVersion = list.GetResourceVersion()

        for {
                options = ListOptions{ResourceVersion: resourceVersion}
                w, err = listWatcher.Watch(options)
                if err {
                        // most likely apiserver is not responsive. It doesn't make
                        // sense to re-list all objects because most likely we will
                        // be able to restart watch where we ended.
                        if err.IsError("connection refused") {
                                sleep(time.Second)
                                continue
                        }
                        // Watch closed normally (EOF) or unexpected error
                        HandleError(err)
                        return nil
                }
                // watchHandler watches w and keeps *resourceVersion up to date.
                watchHandler(w, &resourceVersion)
        }
}
```

Pseudo code

Recap: resourceVersion

Everything has a ResourceVersion:

- Individual API object (e.g. a Pod) has ResourceVersion when you GET one
- For a list of API objects (e.g. a PodList)
    - The entire list has a ResourceVersion
    - Each API object in items has ResourceVersion

The ResourceVersion of the top-level list is what should be used when starting a watch to observe events occurring after that list was populated.

Recap: resourceVersion

- ListOption in List Request
  - Unspecified: etcd
  - RV=0: APIServer cache
  - RV>0: the result is at least as fresh as given RV

Recap: resourceVersion

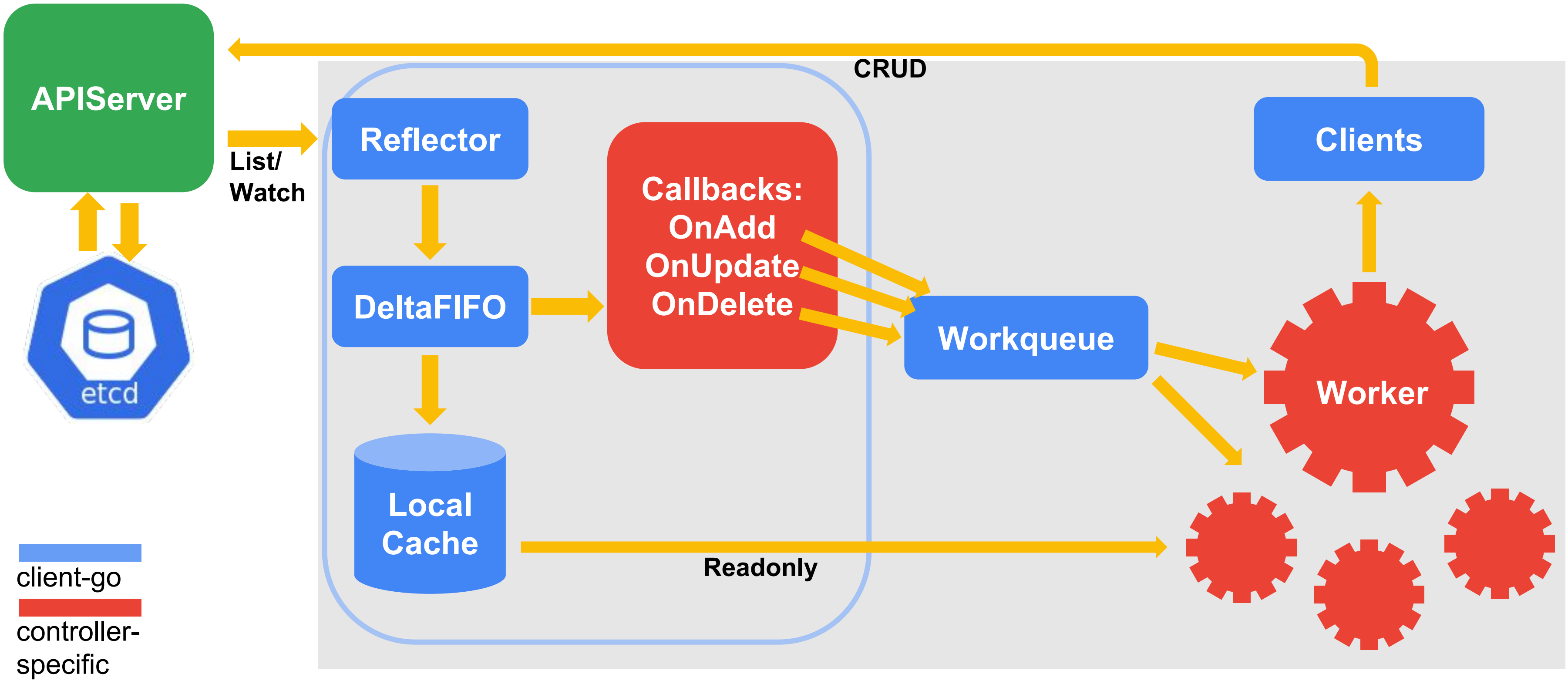
- ListOption in Watch Request
  - Unspecified: random time point
  - RV=0: the result is an "ADDED" event for every existing object followed by events for changes that occur after the watch was established
    - (main reason: backwards compatibility-- #13910)
  - Best practice: always specify last listed/watched RV

KubeCon | CloudNativeCon

North America 2018

Watch Event on kube-scheduler, kube-controller-manager, kublet… (roycaihw)

Kubernetes controller workflow



APIServer

etcd

List/
Watch

Reflector

DeltaFIFO

Local
Cache

Callbacks:
OnAdd
OnUpdate
OnDelete

Workqueue

Worker

Clients

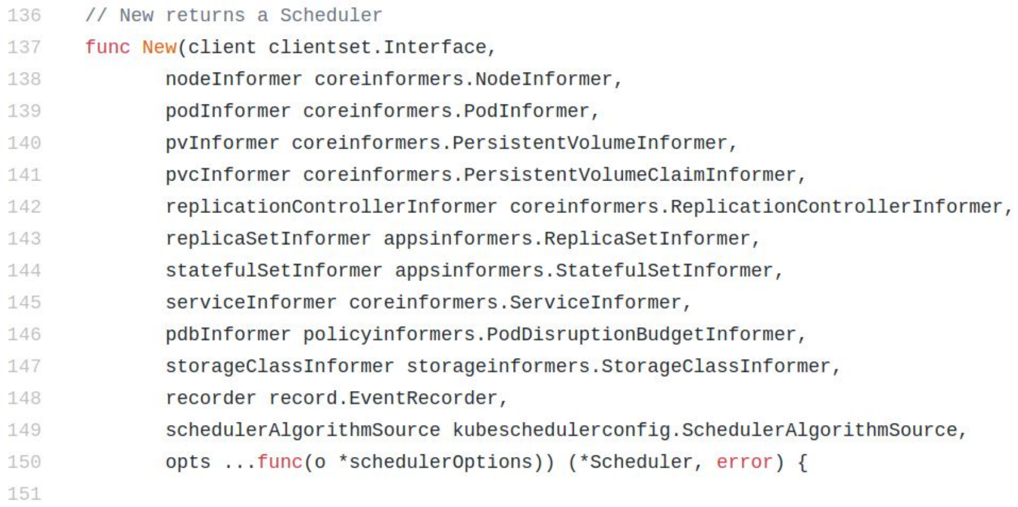
CRUD

Readonly

🟦 client-go

🟥 controller-specific

Scheduler



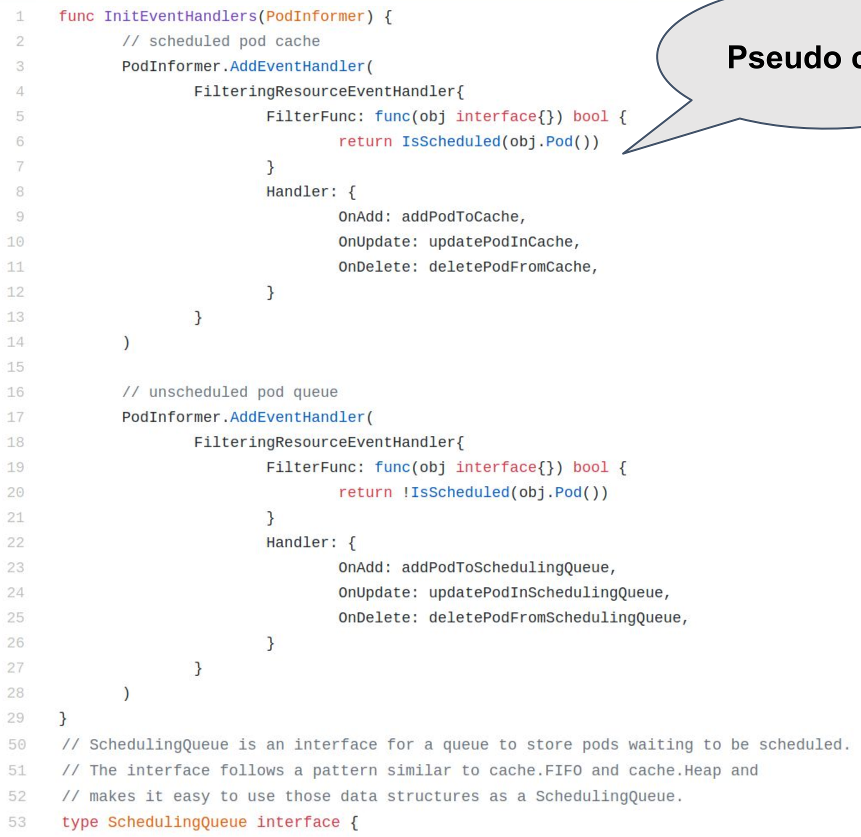
```go
// New returns a Scheduler
func New(client clientset.Interface,
	nodeInformer coreinformers.NodeInformer,
	podInformer coreinformers.PodInformer,
	pvInformer coreinformers.PersistentVolumeInformer,
	pvcInformer coreinformers.PersistentVolumeClaimInformer,
	replicationControllerInformer coreinformers.ReplicationControllerInformer,
	replicaSetInformer appsinformers.ReplicaSetInformer,
	statefulSetInformer appsinformers.StatefulSetInformer,
	serviceInformer coreinformers.ServiceInformer,
	pdbInformer policyinformers.PodDisruptionBudgetInformer,
	storageClassInformer storageinformers.StorageClassInformer,
	recorder record.EventRecorder,
	schedulerAlgorithmSource kubeschedulerconfig.SchedulerAlgorithmSource,
	opts ...func(o *schedulerOptions)) (*Scheduler, error) {

```

pkg/scheduler/scheduler.go

Watches:

- Node
- Pod
- PV
- PVC
- RC
- RS
- Stateful set
- Service
- PDB
- Storage class

Scheduler “business” logic



Pseudo code

```
func InitEventHandlers(PodInformer) {
    // scheduled pod cache
    PodInformer.AddEventHandler(
        FilteringResourceEventHandler{
            FilterFunc: func(obj interface{}) bool {
                return IsScheduled(obj.Pod())
            }
            Handler: {
                OnAdd: addPodToCache,
                OnUpdate: updatePodInCache,
                OnDelete: deletePodFromCache,
            }
        }
    )

    // unscheduled pod queue
    PodInformer.AddEventHandler(
        FilteringResourceEventHandler{
            FilterFunc: func(obj interface{}) bool {
                return !IsScheduled(obj.Pod())
            }
            Handler: {
                OnAdd: addPodToSchedulingQueue,
                OnUpdate: updatePodInSchedulingQueue,
                OnDelete: deletePodFromSchedulingQueue,
            }
        }
    )
}
// SchedulingQueue is an interface for a queue to store pods waiting to be scheduled.
// The interface follows a pattern similar to cache.FIFO and cache.Heap and
// makes it easy to use those data structures as a SchedulingQueue.
type SchedulingQueue interface {
```

Scheduler keeps in-memory:

- PodCache for scheduled pods
- SchedulingQueue for pods waiting to be scheduled

Having knowledge of the state of world enables scheduler to

- calculate resource usage on nodes
- be aware of available PVs that can be binded with pods
- schedule pods onto nodes

Watch enables on-demand information propagation, and avoids heavy information-polling loads.

KubeCon | CloudNativeCon

North America 2018

Key Takeaways

Use Watch for your controller!

- It’s trustworthy!
- It’s efficient!

KubeCon | CloudNativeCon

North America 2018

?