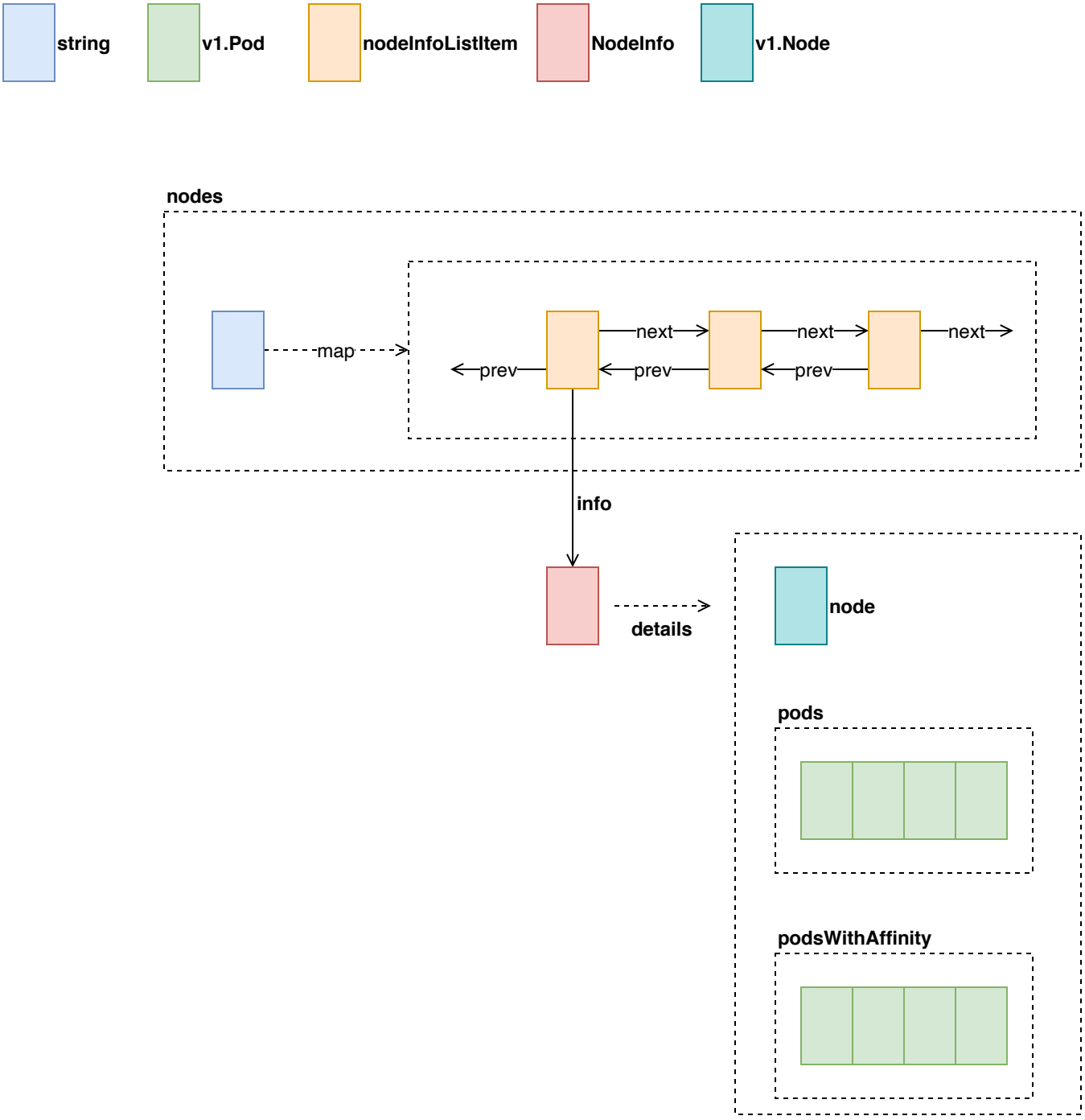
Scheduler - Kubernetes 源码社特供

Scheduler Cache

Nodes

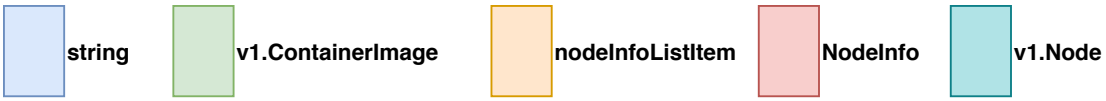


Nodes 中保存了 Node.Name 到 nodeInfoListItem 链表的映射。每个 nodeInfoListItem 对应一个 NodeInfo 实例，实例中保存了 Node 信息及相关的 Pod 信息。

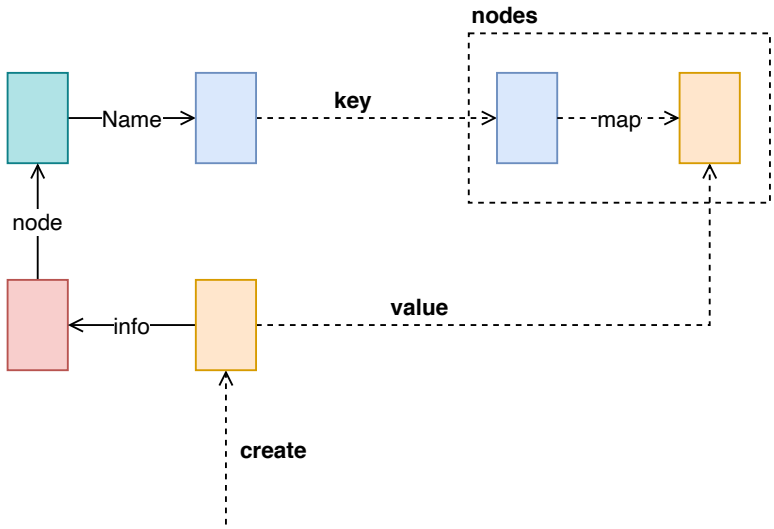
AddNode

AddNode 方法执行时，需要传入一个 Node 实例。首先，根据 Node.Name 是否存在于 nodes 中来判断执行路径。

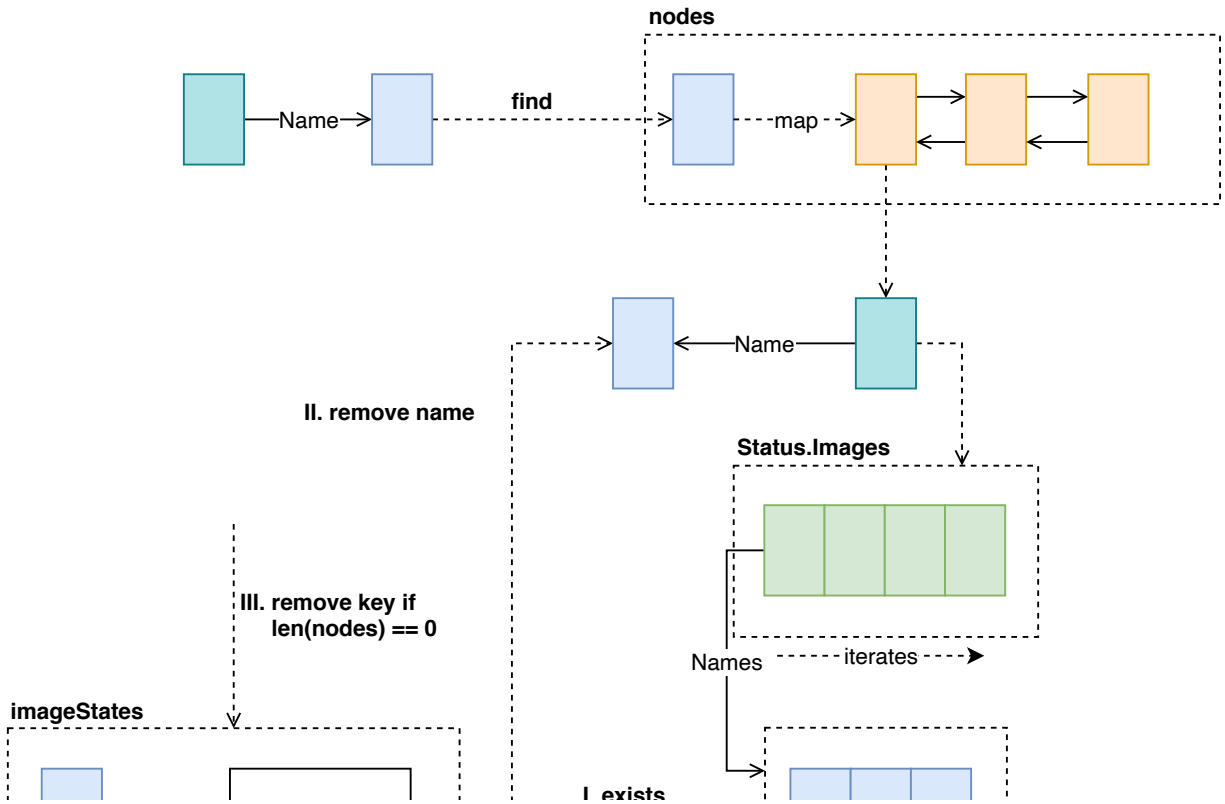
如果 Node.Name 不存在，那么创建一个新的 nodeInfoListItem 并存入 nodes 中。如果已经存在，那么获取对应的链表第一个对象，使用该对象包含的 Node 节点进行镜像清理，需要注意，这里并没有删除镜像，只是在 imageStates 中移除镜像名。

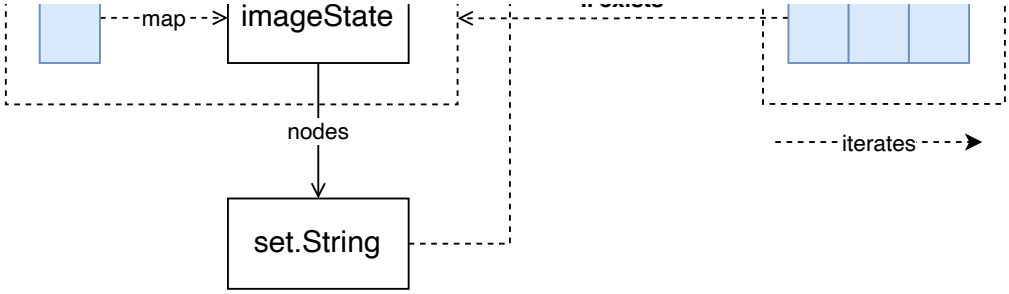


node.Name not exists in nodes

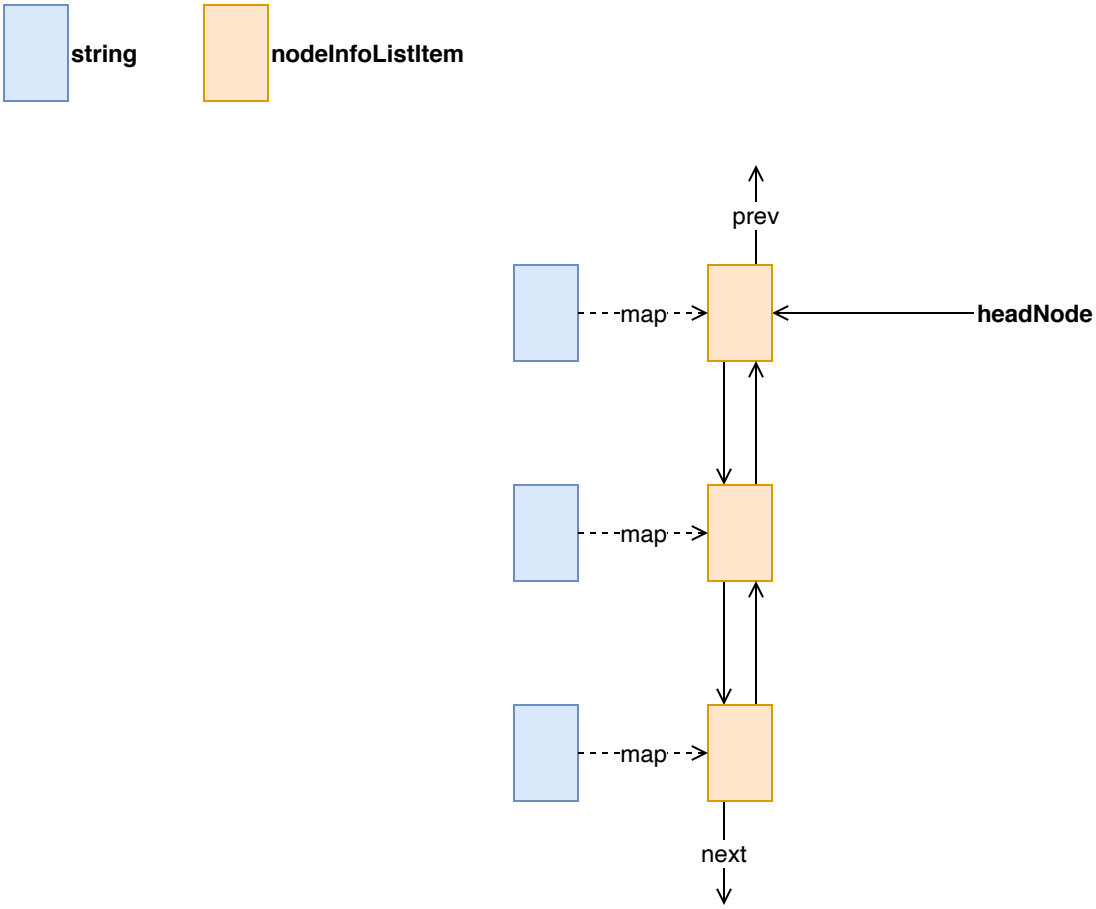


node.Name exists in nodes

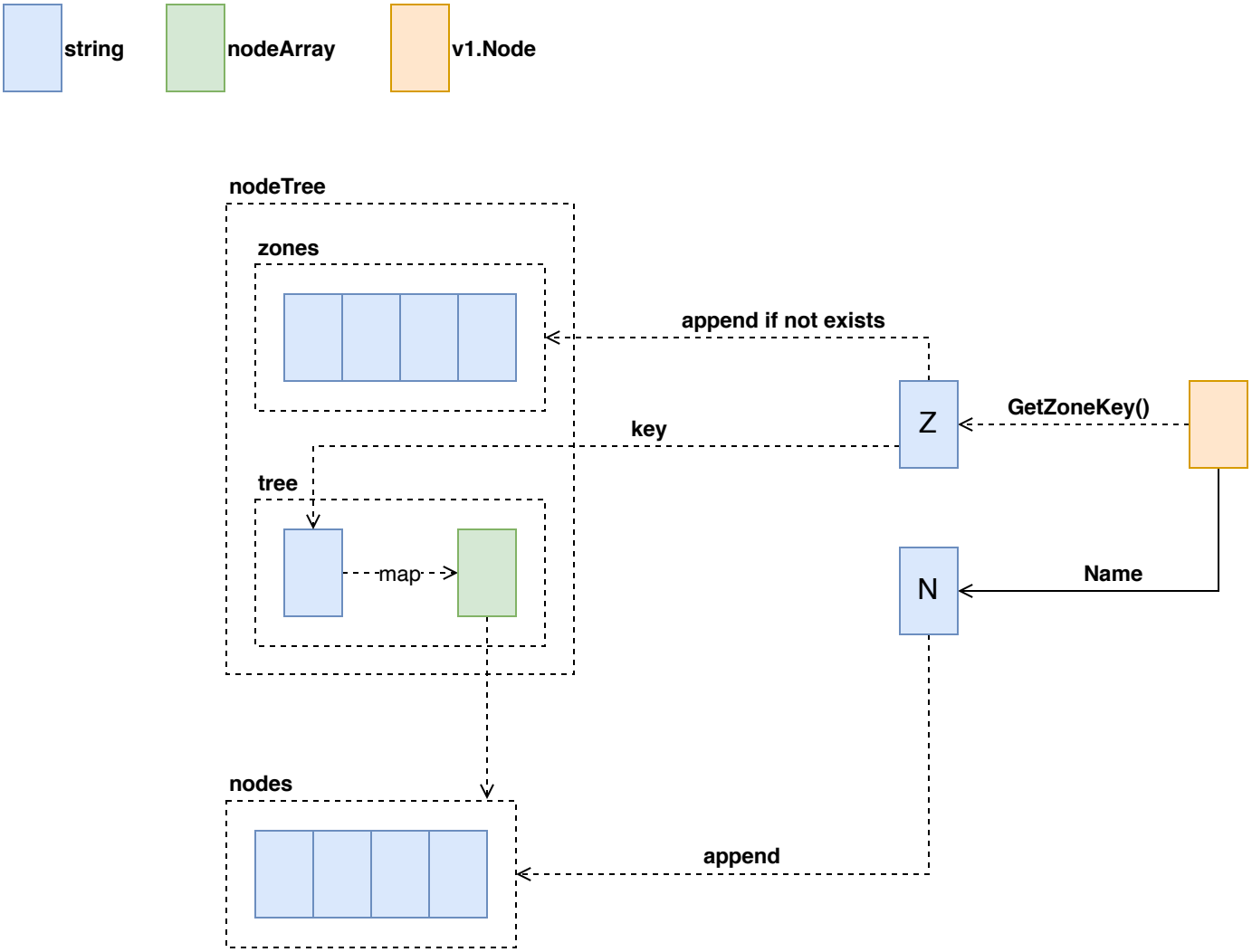




然后，将最近修改的 Node 对应的链表项移动至 headNode 表头，如下图所示，这样也解释了为什么一个 Node 对应的 Key 会关联一个链表。事实上，一个 Key 只有一个链表项，通过 headNode 关联起来的是最近使用顺序。



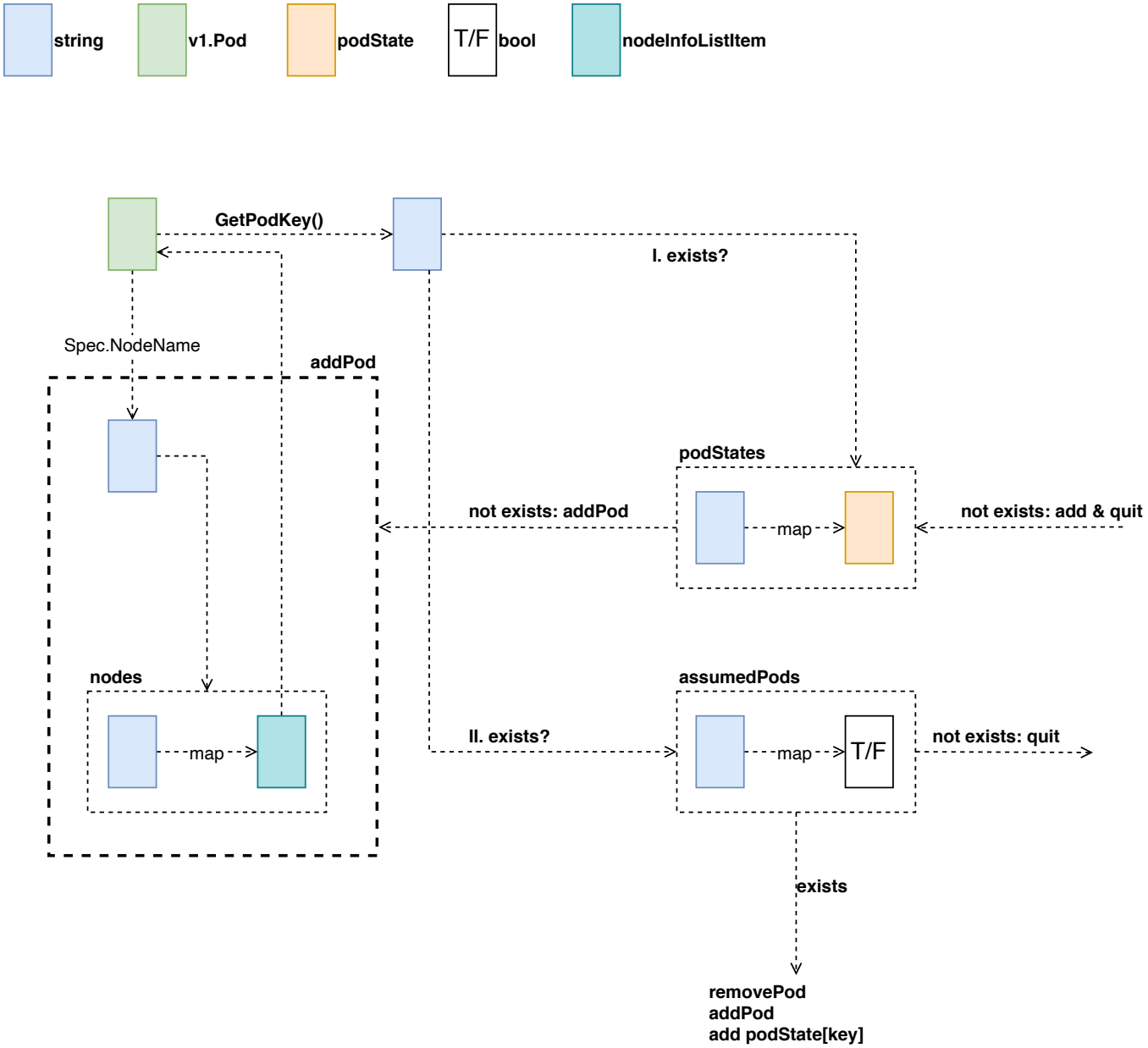
接着，将 Node 添加至 nodeTree 中，过程如下图



完成后，将 Node 中关联的镜像添加至 imageStates 中，关于 imageState 的清理操作，前面已详细说明，添加操作不再深入。

Pod

AddPod

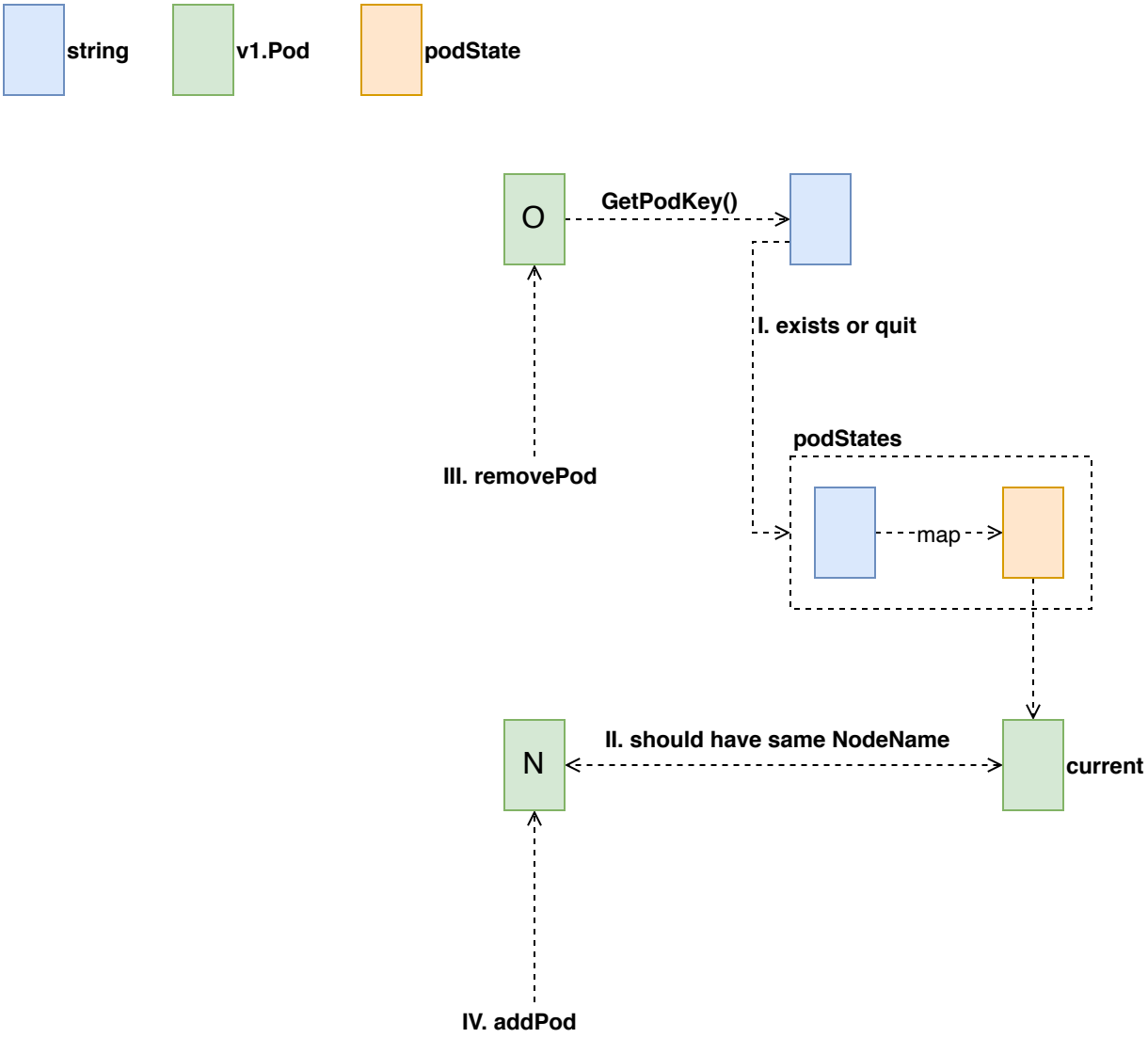


当 podStates 中对应 Pod Key 中存储的 Pod 的 NodeName 与新 Pod 的 NodeName 不一致时，会执行 removePod 操作，其代码如下

```
449  func (cache *schedulerCache) removePod(pod *v1.Pod) error {
450      n, ok := cache.nodes[pod.Spec.NodeName]
451      if !ok {
452          return nil
453      }
454      if err := n.info.RemovePod(pod); err != nil {
455          return err
456      }
457      cache.moveNodeInfoToHead(pod.Spec.NodeName)
458      return nil
459  }
```

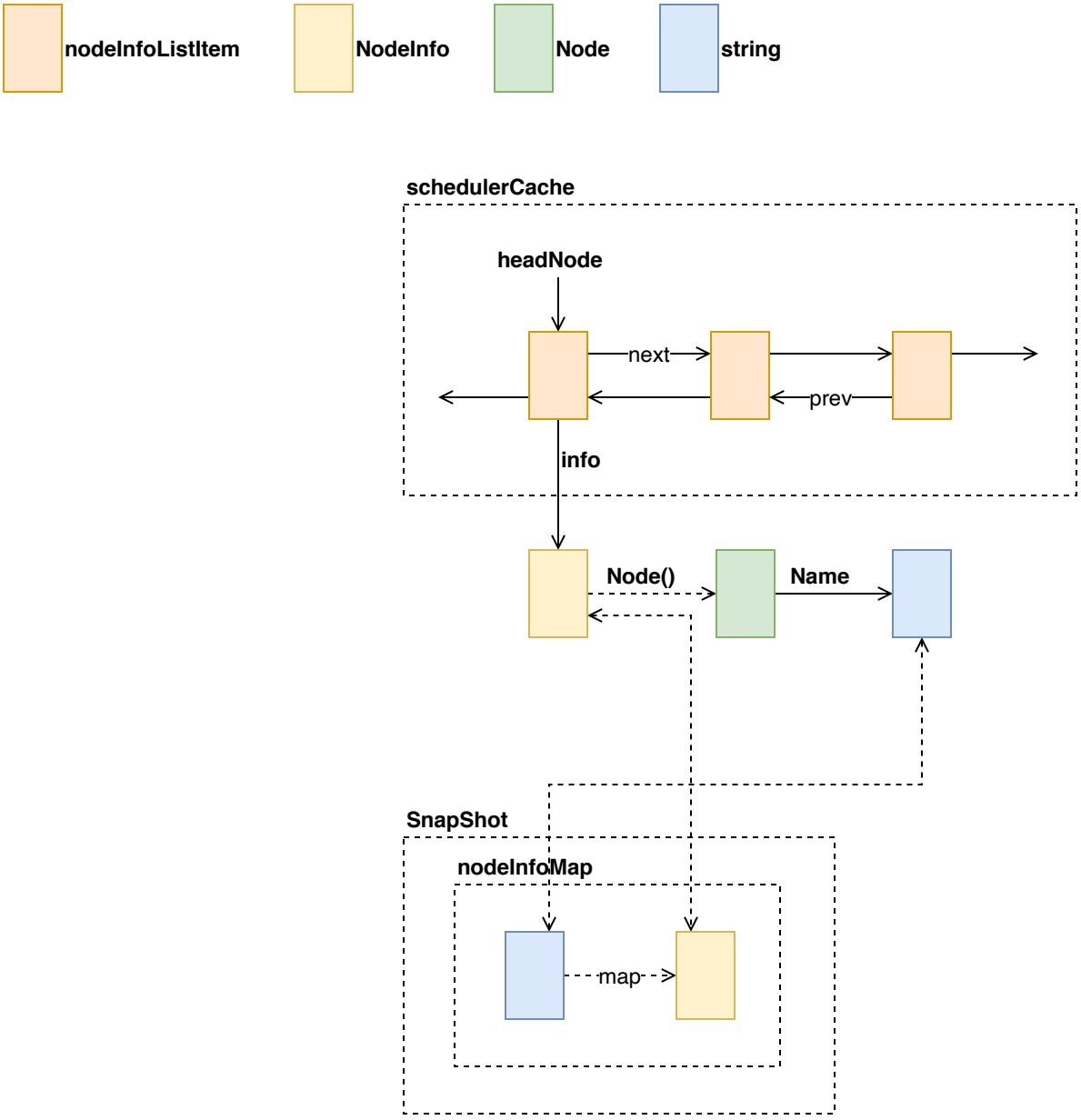
随后，再执行 addPod，这里不再描述，前面的图中已详细绘制。

UpdatePod



Schedule

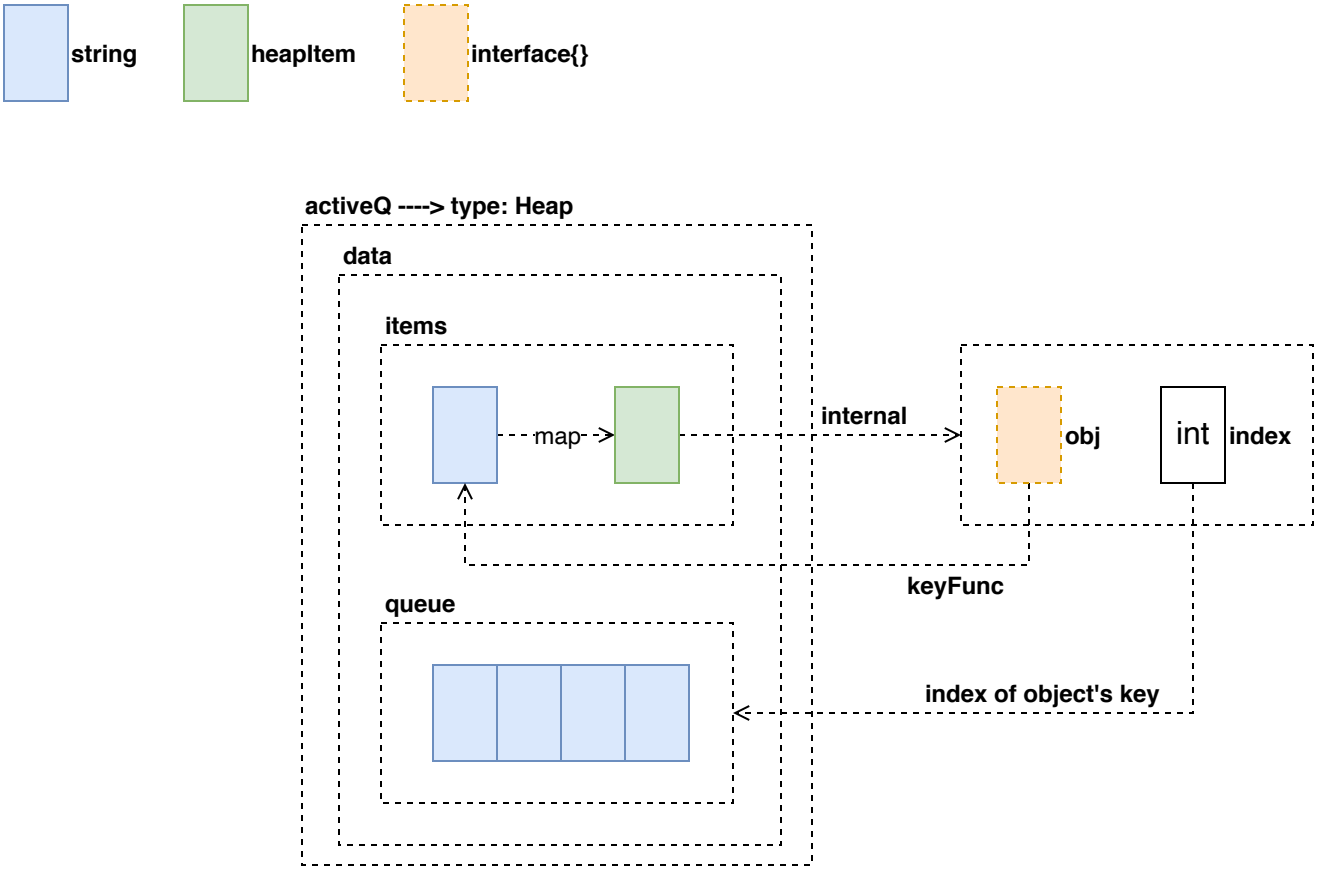
Update Snapshot



Priority Queue

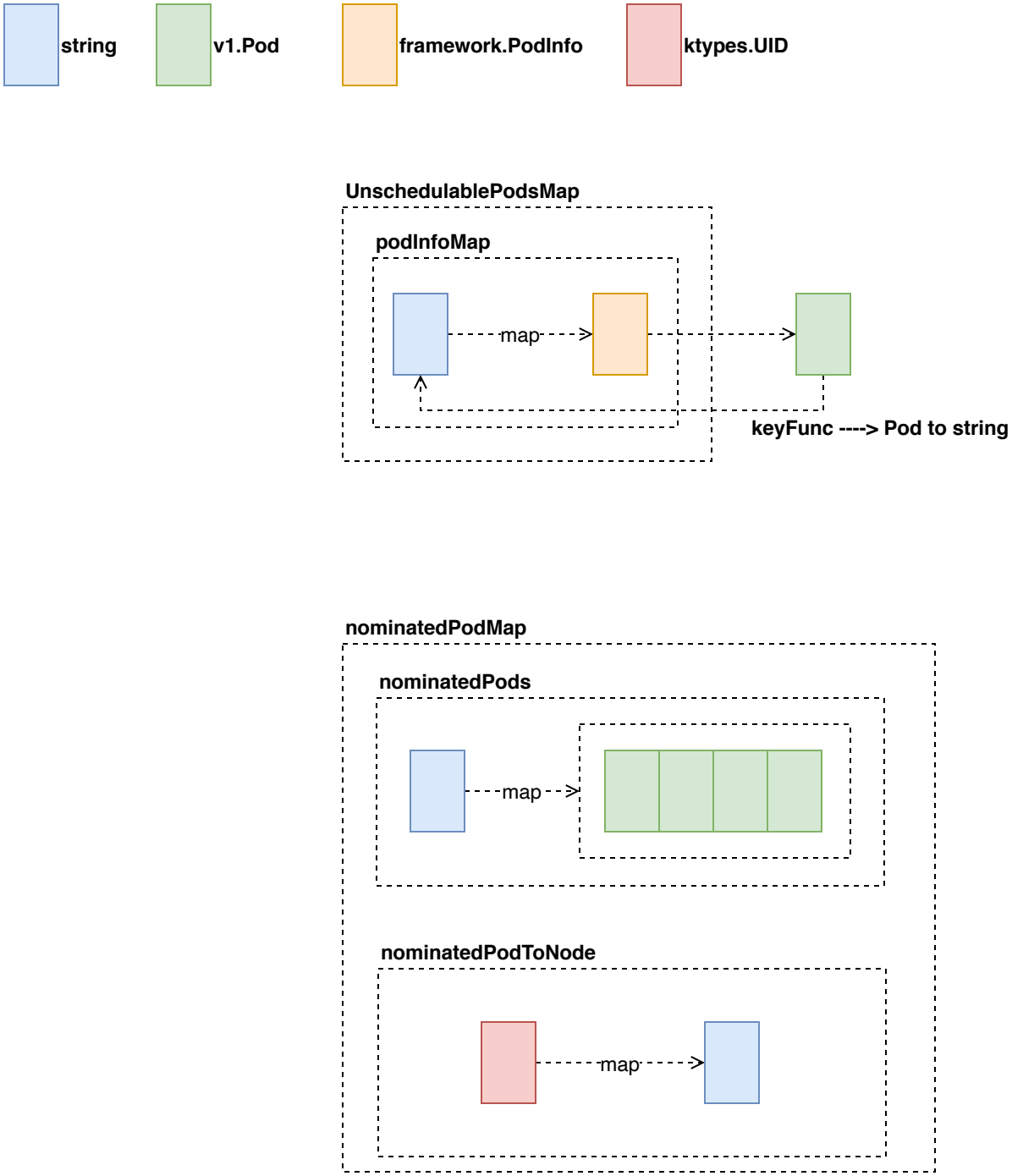
Internal Data Structures

Heap



上图为 PriorityQueue 中 activeQ 域，它是一个 Heap 实例。Heap 中核心结构为 data，包含一个字符串到 heapItem 的映射，heapItem 存储了实际对象以及该对象的 Key 在 queue 中位置的变量。

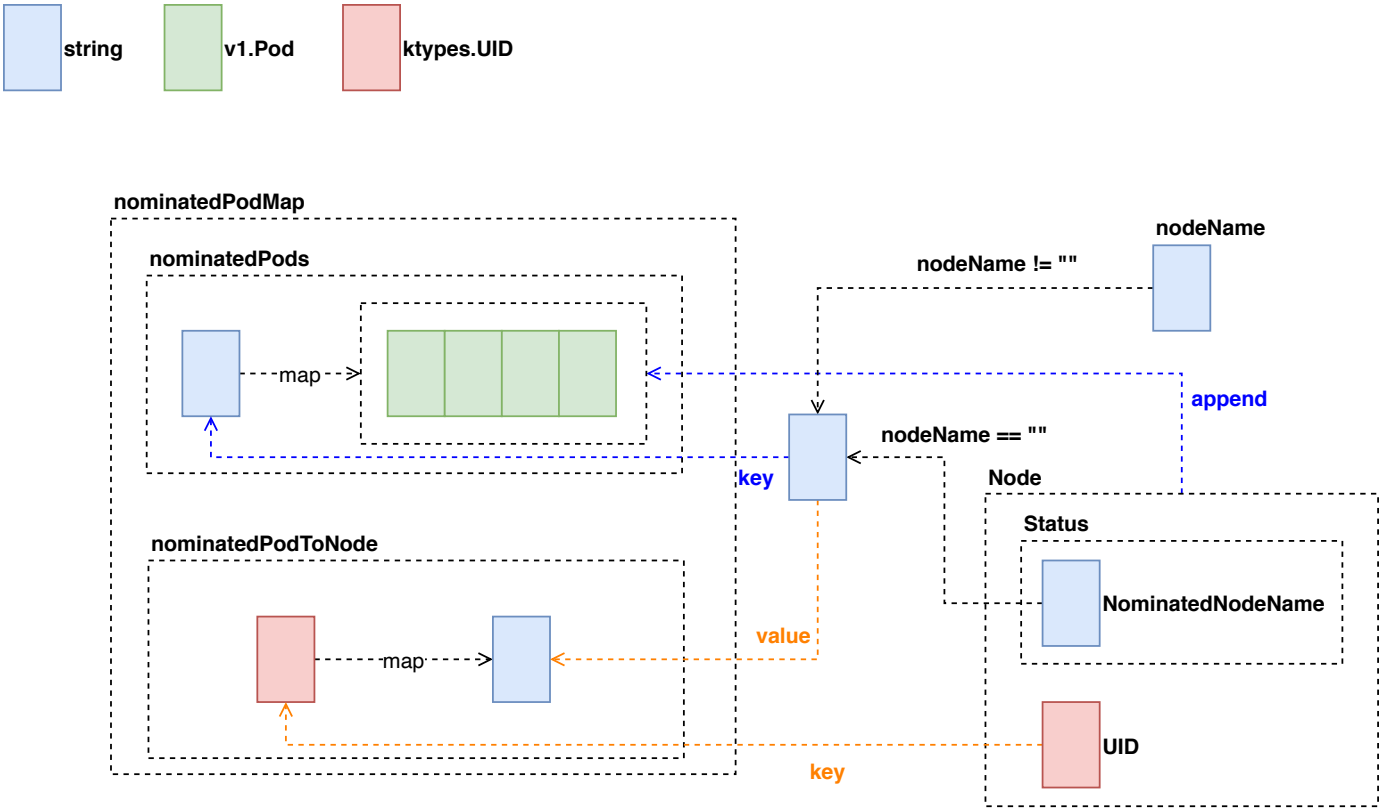
Pods Containers



PriorityQueue 中与 Pod 关联的两个数据结构如上图所示。UnschedulablePodsMap 中保存了从 Pod 信息到 key 值的方法。

nominatedPodMap

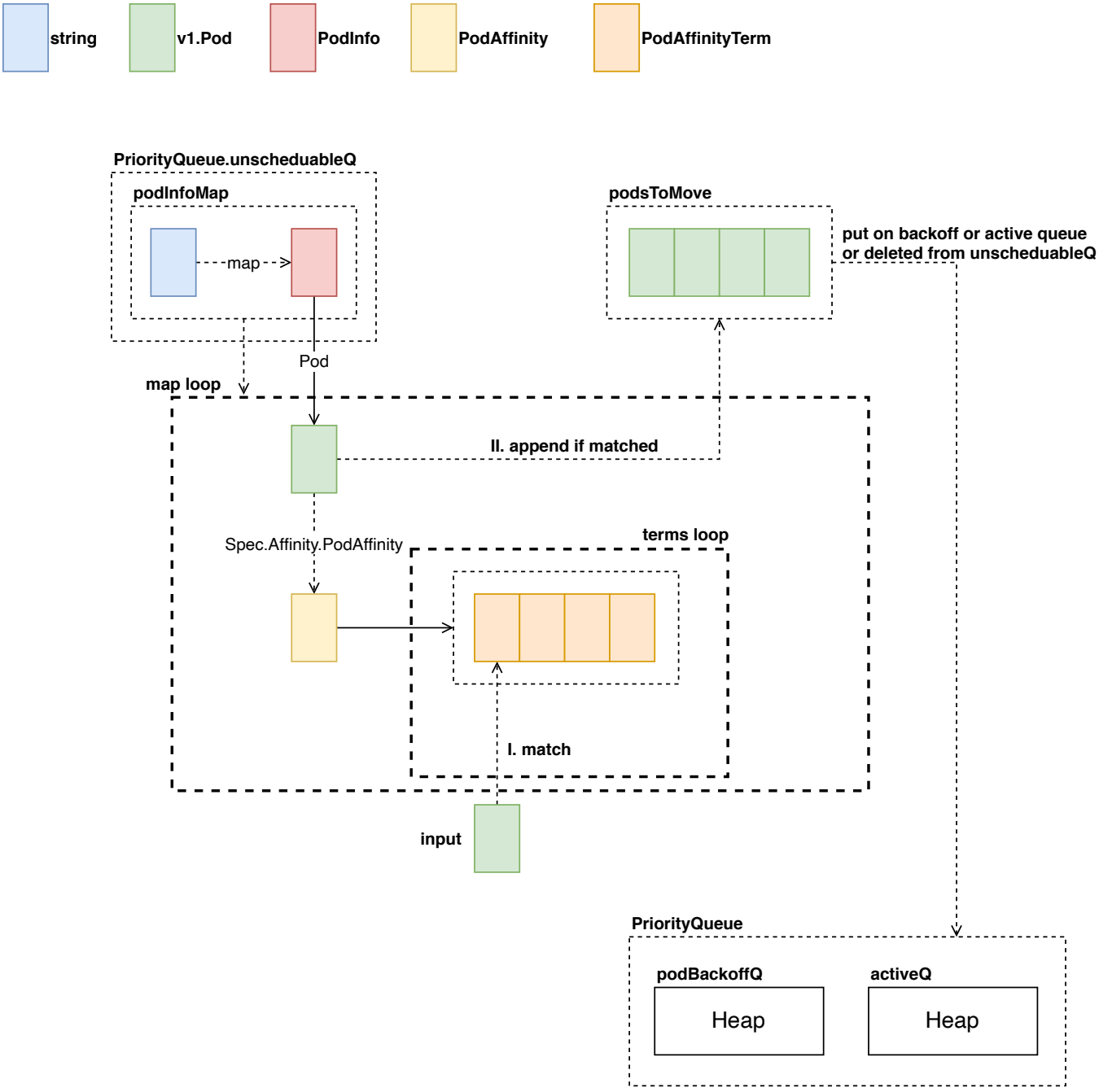
Add



优先使用传入的 nodeName，若 nodeName 为空时，使用 UID，若 UID 也为空，处理完毕。否则，按上图示意，添加对应的 map。

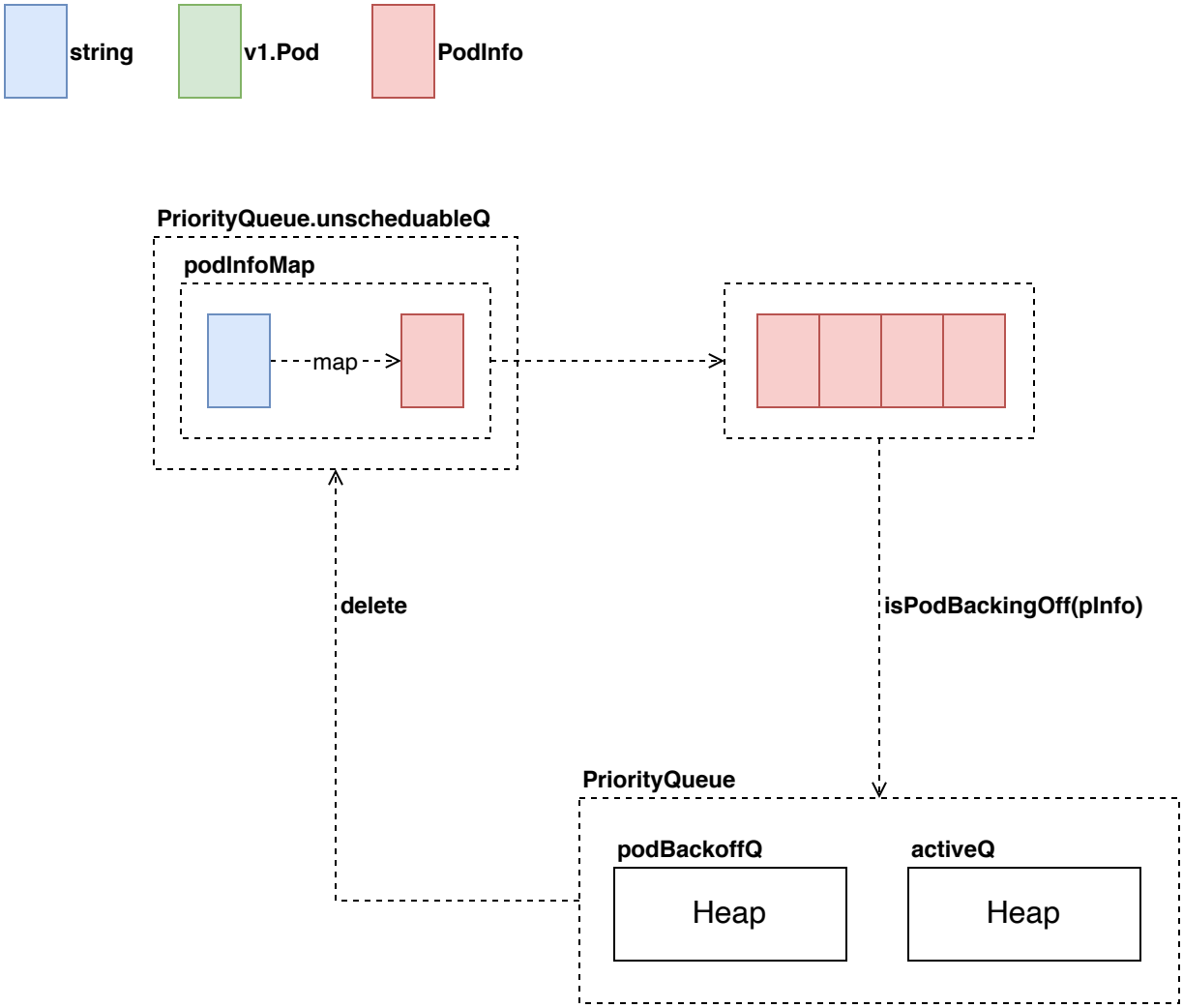
Pod Handling

Added/Update



kube-scheduler 接收到添加 Pod 事件后，会将 Pod 添加进 SchedulerCache 中，随后，执行上图操作。Pod 的添加、更新操作执行相同代码，区别为状态不同，分别对应为 AssignedPodAdd 与 AssignedPodUpdate。

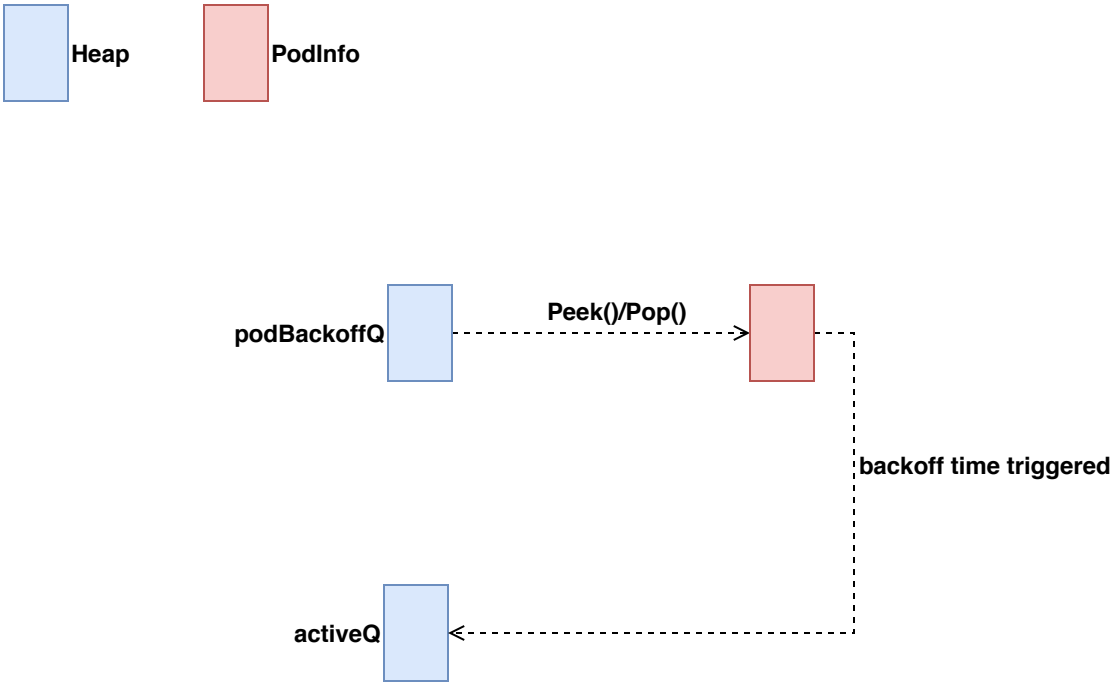
Delete



将 podInfoMap 中全部 PodInfo 移动至 podBackoffQ 或 activeQ 中，并删除 podInfoMap 中对应 K/V 对，标记状态为 AssignedPodDelete。

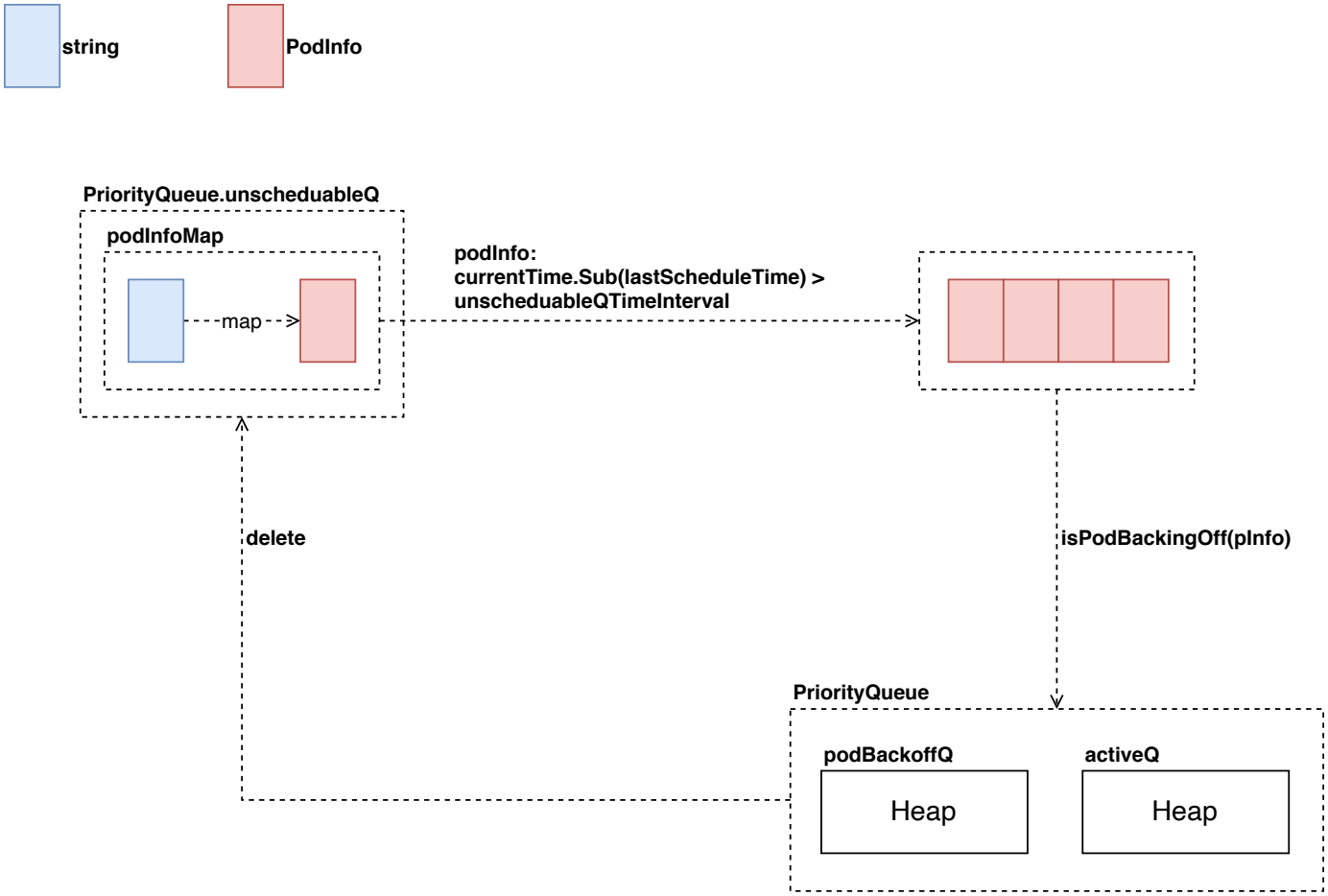
Handling Routines

Flush Backoff Queue



定时从 backoff queue 中获取一个 PodInfo 对象，检查其 backoff time 是否到期，如果没有到期，直接返回，等待下次触发，此时，PodInfo 对象仍然存在于 backoff queue 中。如果到期，则弹出该对象，并存入 active queue 中，此时，PodInfo 对象被移除出 backoff queue。

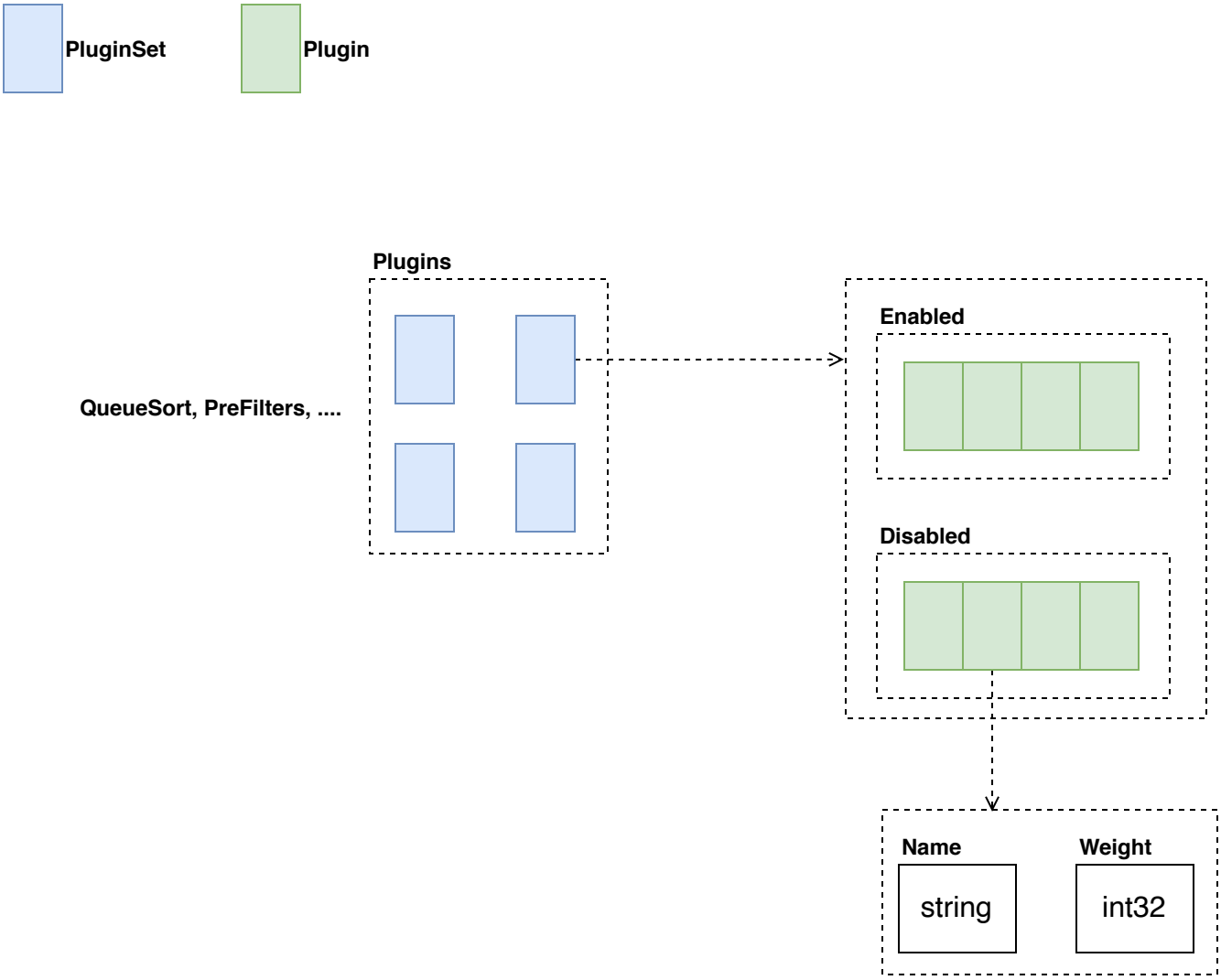
Flush Unscheduable Queue



Plugins

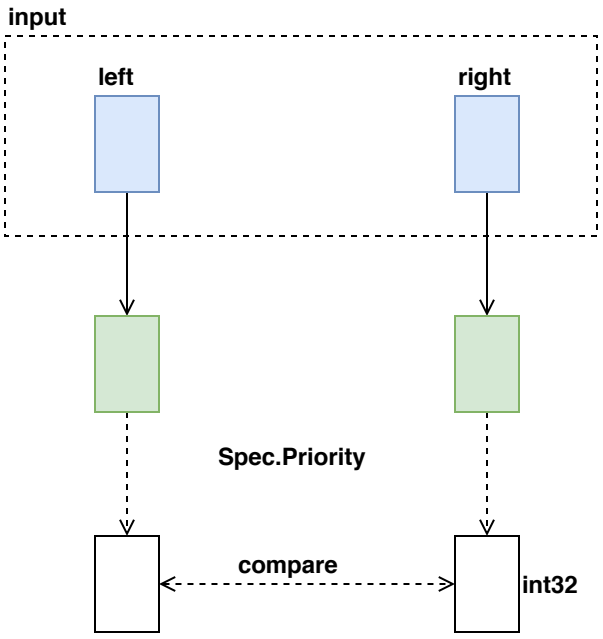
Basic Structures

Plugins



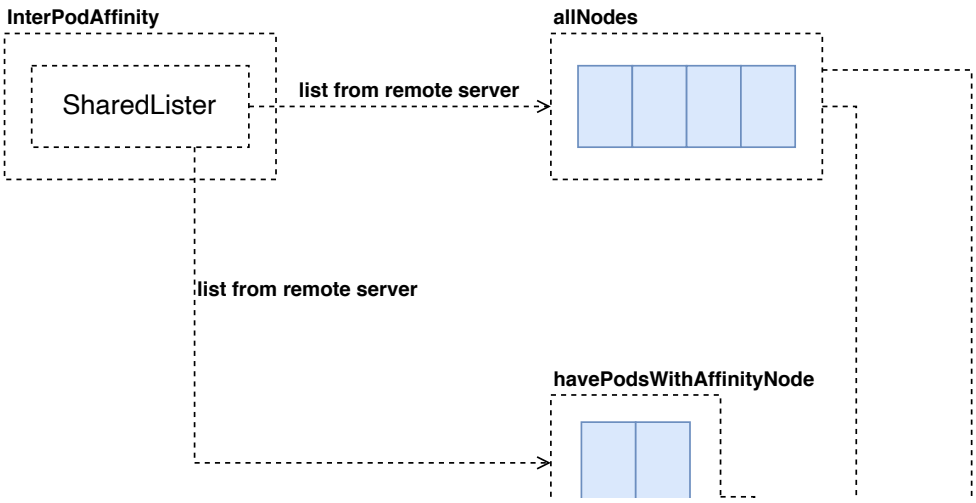
Plugins 包含了一组 PluginSet，每个 PluginSet 又包含了两组 Plugin，一组为激活状态，一组为禁用状态。Plugin 包含一个唯一标识符和该 Plugin 的权重。

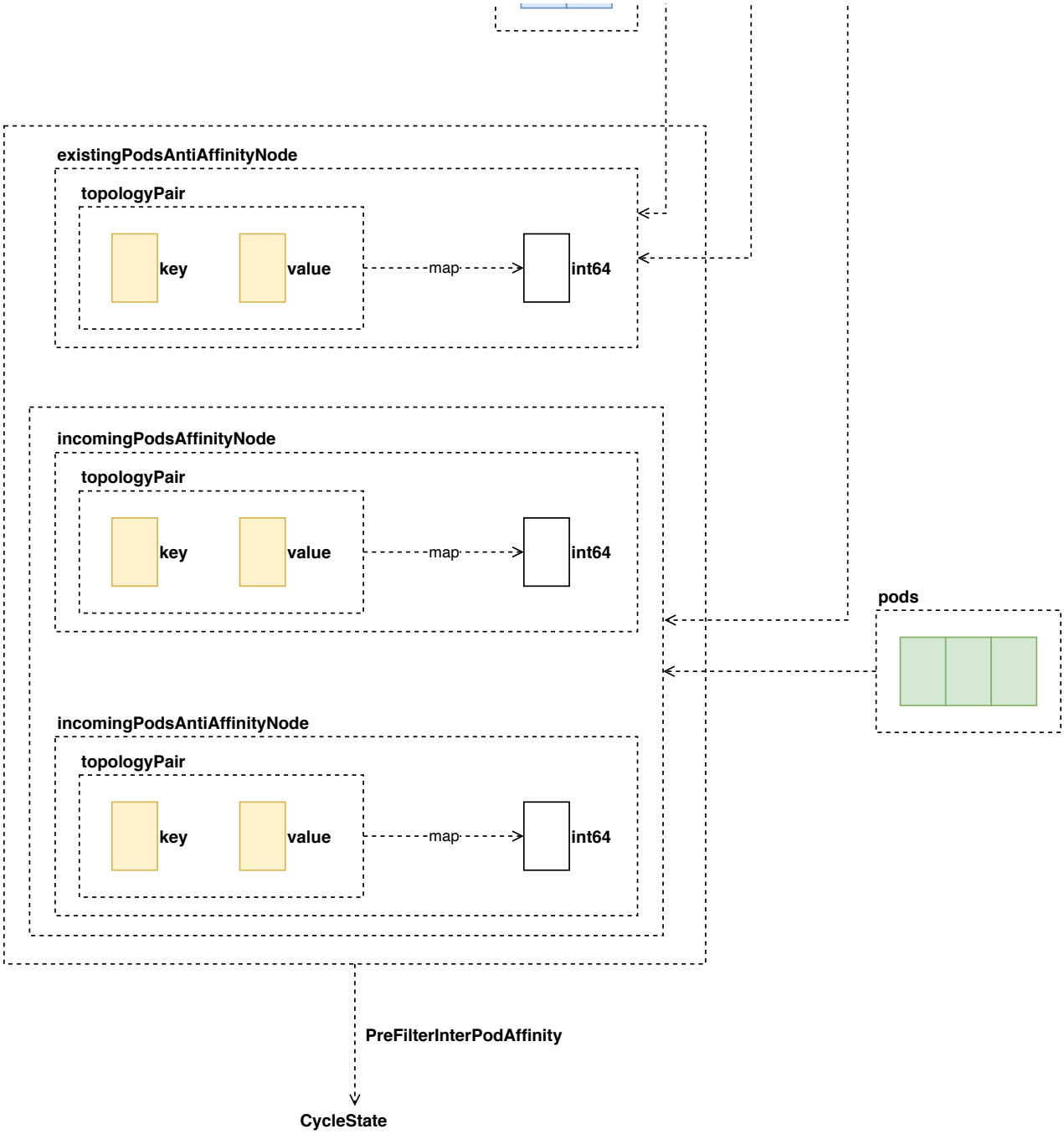
Quick Sort Plugin



接口方法 Less 如上图所示，传入两个 PodInfo 实例，通过 Less 方法判定二者在排序时先后位置。由于 Plugin 接口只有通用方法 Name，因此，每个特定功能的 Plugin 原则上可随意定制自己需要的方法。

InterPodAffinity



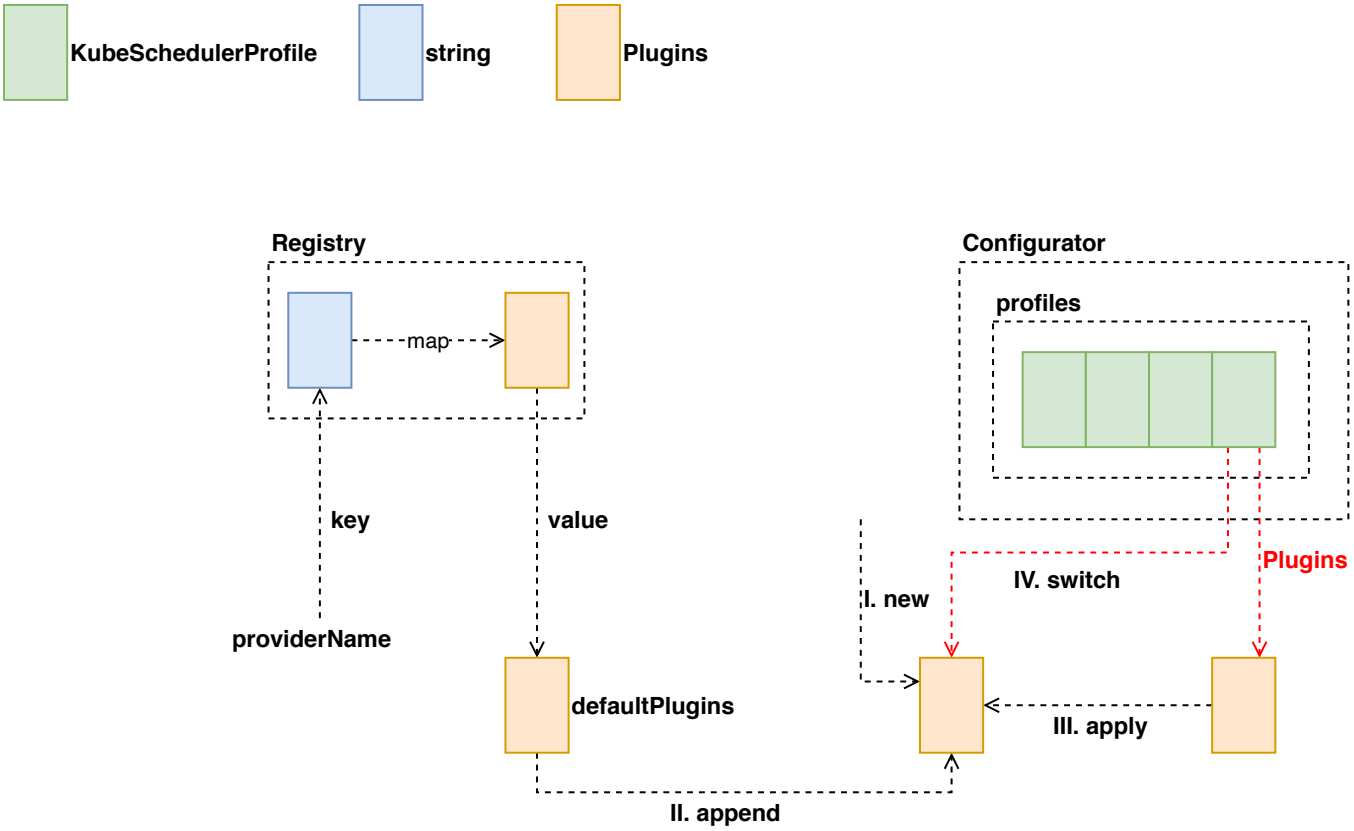


将与当前调度 Pod 的相关 Node 数据存储在 CycleState 的 PreFilterInterPodAffinity 关键字中，供后续调度使用。

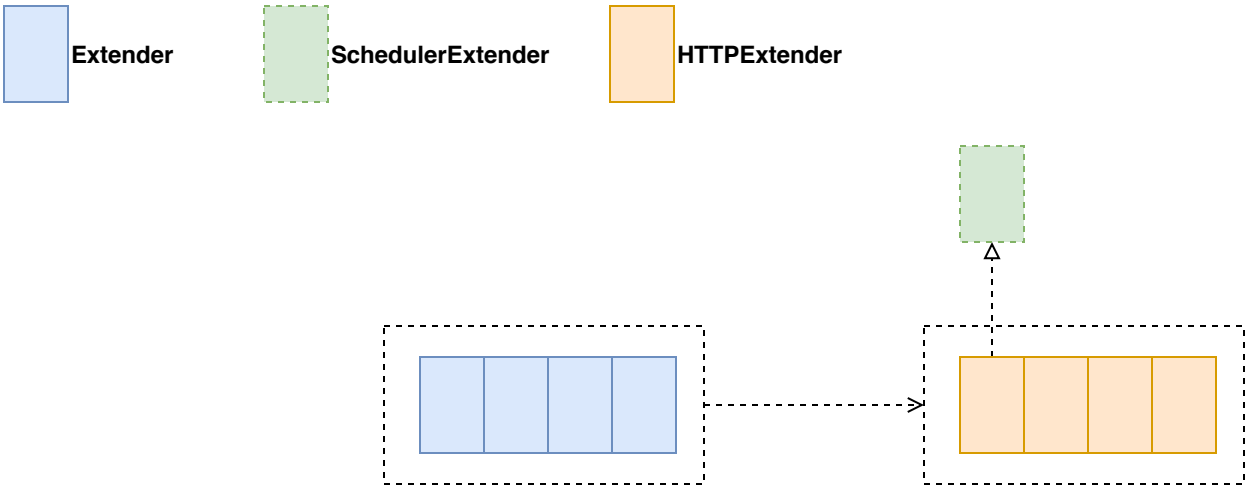
Configurator

Profile

Registry 用于组织 provider 与 Plugins 间关联关系，保存的是默认的 Plugins。在创建 Profile 结构时，会使用这些默认的 Plugins。



Extender

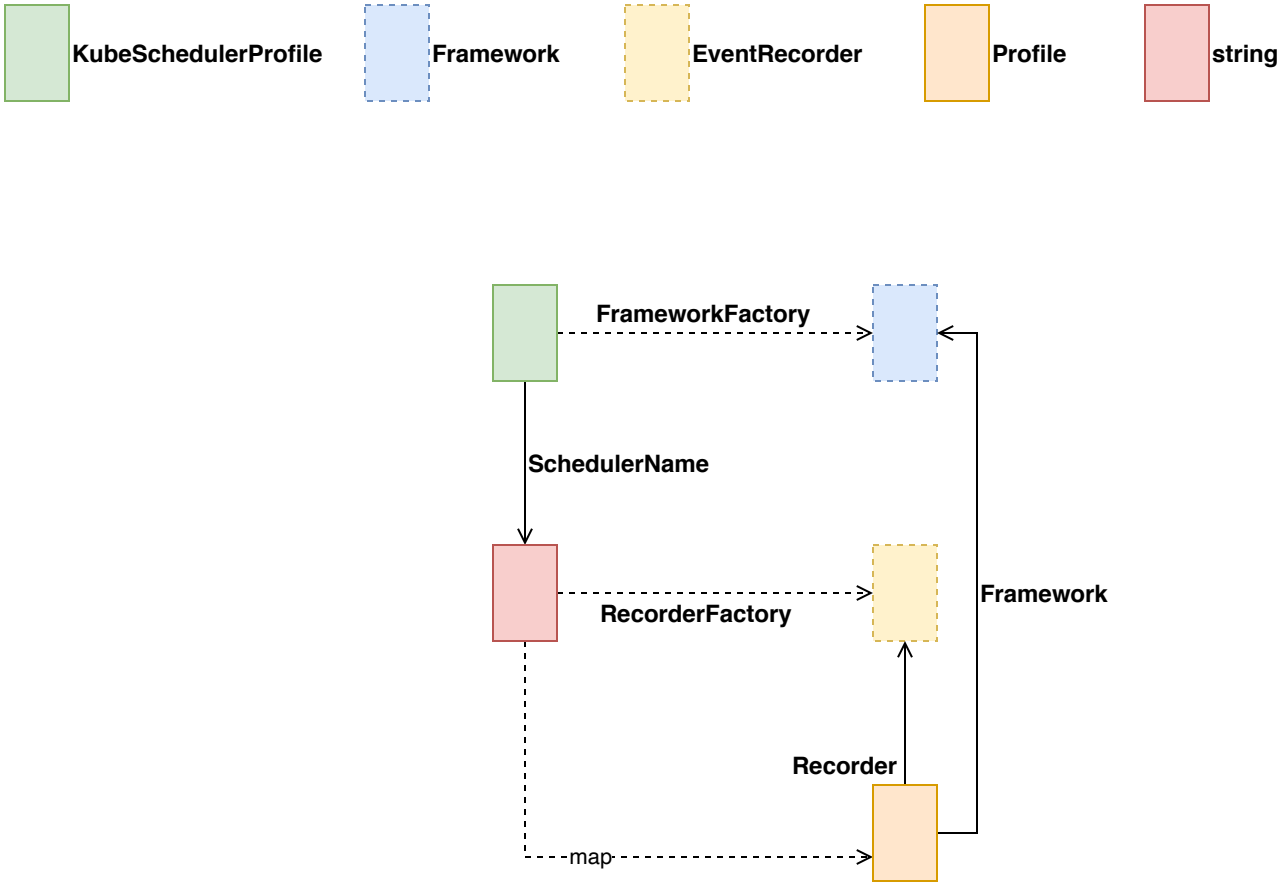


相关代码如下图所示

```go
func NewHTTPExtender(config *schedulerapi.Extender) (SchedulerExtender, error) {
    if config.HTTPTimeout.Nanoseconds() == 0 {
        config.HTTPTimeout = time.Duration(DefaultExtenderTimeout)
    }

    transport, err := makeTransport(config)
    if err != nil {
        return nil, err
    }
    client := &http.Client{
        Transport: transport,
        Timeout:   config.HTTPTimeout,
    }
    managedResources := sets.NewString()
    for _, r := range config.ManagedResources {
        managedResources.Insert(string(r.Name))
    }
    return &HTTPExtender{
        extenderURL:      config.URLPrefix,
        preemptVerb:      config.PreemptVerb,
        filterVerb:       config.FilterVerb,
        prioritizeVerb:   config.PrioritizeVerb,
        bindVerb:         config.BindVerb,
        weight:           config.Weight,
        client:           client,
        nodeCacheCapable: config.NodeCacheCapable,
        managedResources: managedResources,
        ignorable:        config.Ignorable,
    }, nil
}
```

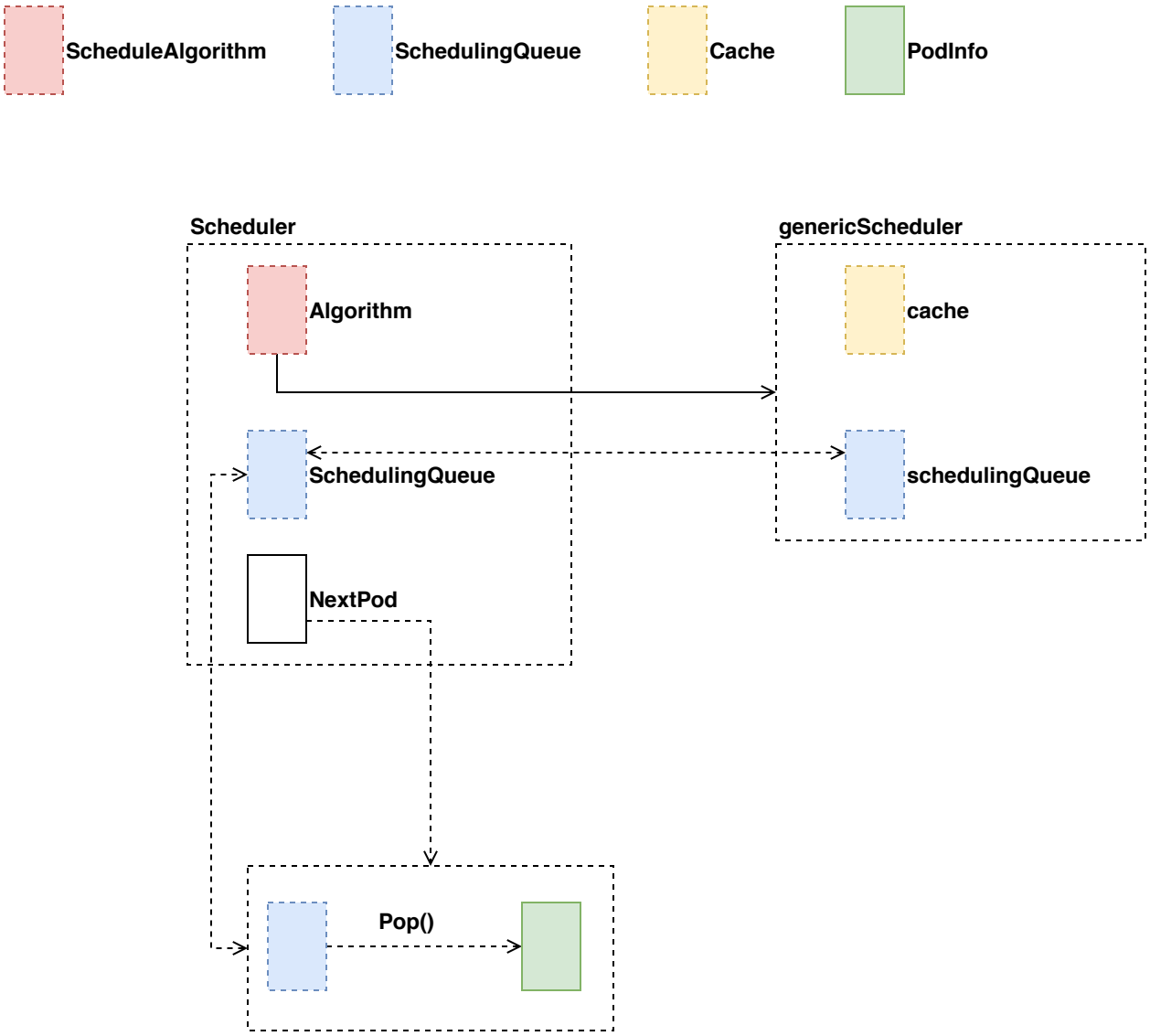
Profile Map



Schedule

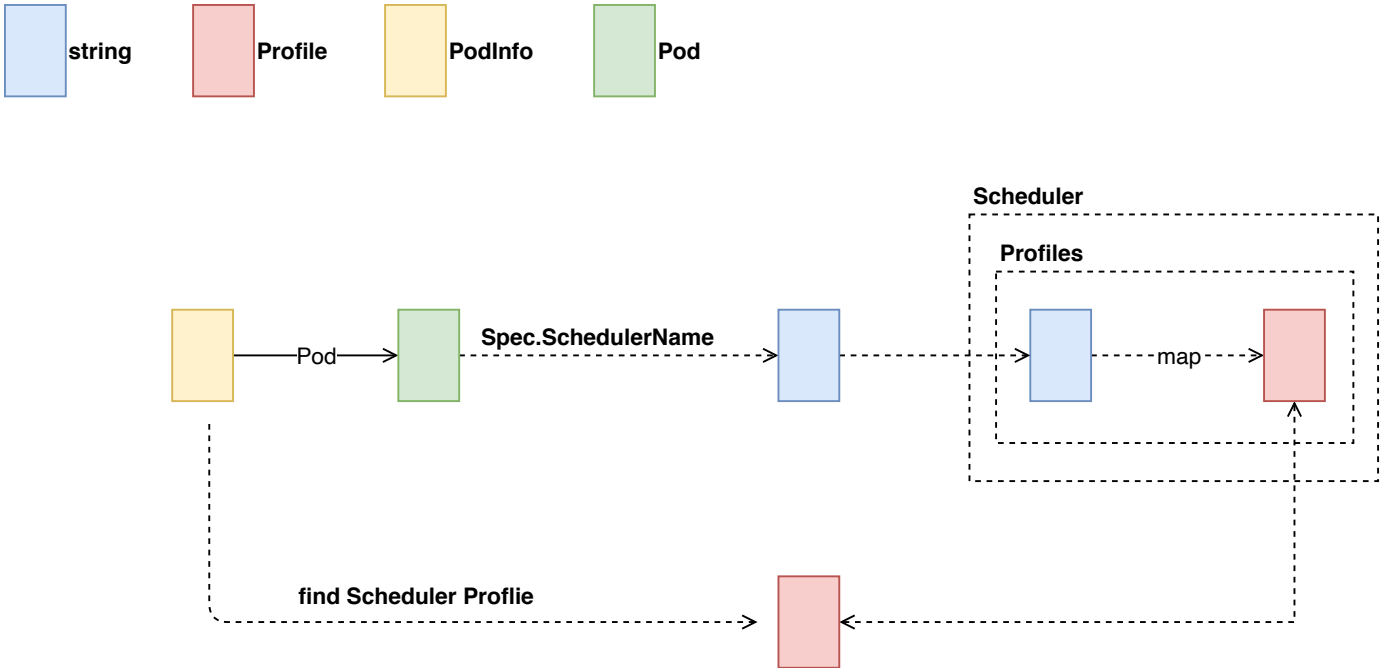
Scheduling

NextPod



Scheduler 中封装了具体调度算法，并与调度算法共享相同的 SchedulingQueue 实例对象。在执行调度时，首先需要通过 NextPod 方法获取待调度的 PodInfo。

Find Profile for Pod

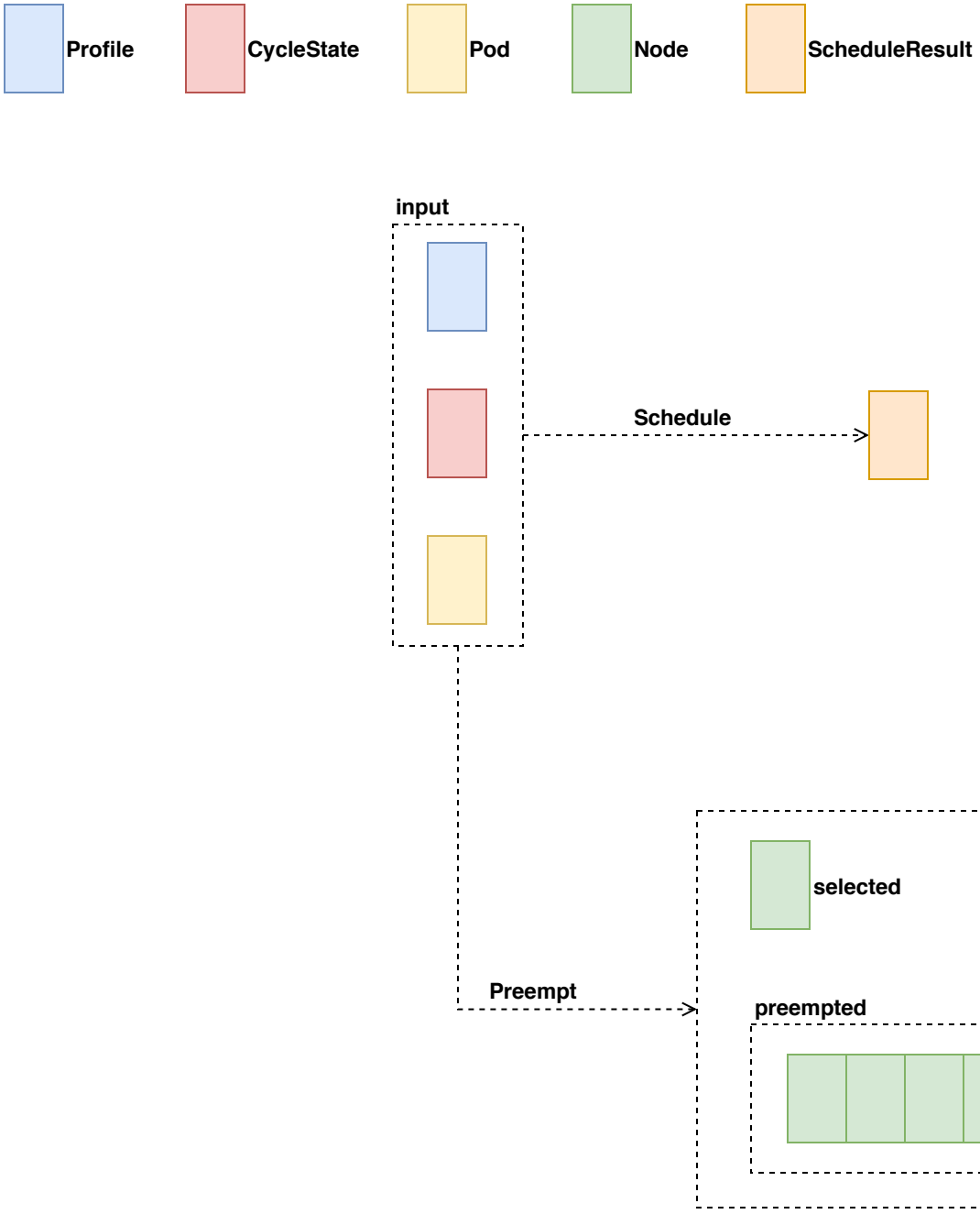


Profile 中保存着调度框架基本信息。根据选中的当前调度 PodInfo 中的 Pod 实例，选中合适的调度 Profile。根据 Profile、Pod 来判断是否需要跳过当前 Pod 调度，如果判断结果不需要调度当前 Pod，本次调度完成；如果需要调度，则通过 Algorithm 进行调度。

```
752  // skipPodSchedule returns true if we could skip scheduling the pod for specified cases.
753  func (sched *Scheduler) skipPodSchedule(prof *profile.Profile, pod *v1.Pod) bool {
754      // Case 1: pod is being deleted.
755      if pod.DeletionTimestamp != nil {
756          prof.Recorder.Eventf(pod, nil, v1.EventTypeWarning, "FailedScheduling", "Scheduling", "skip schedule deleting
757          klog.V(3).Infof("Skip schedule deleting pod: %v/%v", pod.Namespace, pod.Name)
758          return true
759      }
760
761      // Case 2: pod has been assumed and pod updates could be skipped.
762      // An assumed pod can be added again to the scheduling queue if it got an update event
763      // during its previous scheduling cycle but before getting assumed.
764      if sched.skipPodUpdate(pod) {
765          return true
766      }
767
768      return false
769  }
```

Algorithm

Overview

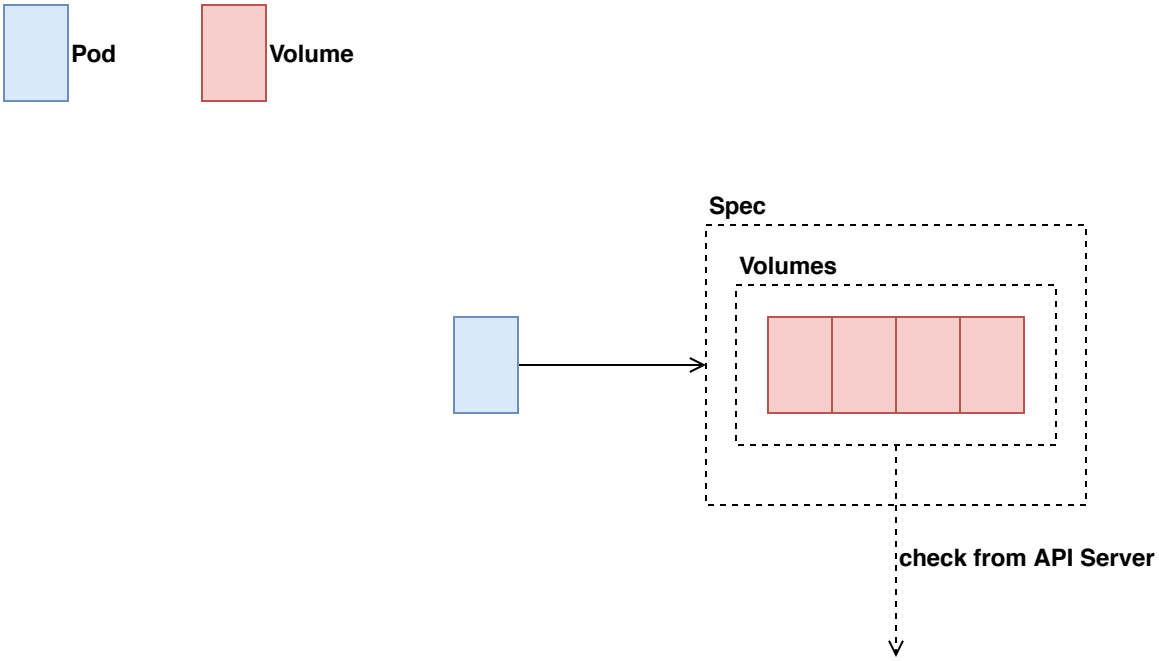


ScheduleAlgorithm 定义了关于调度的两个核心方法，同时，也通过 Extenders 方法预留出了扩展空间。

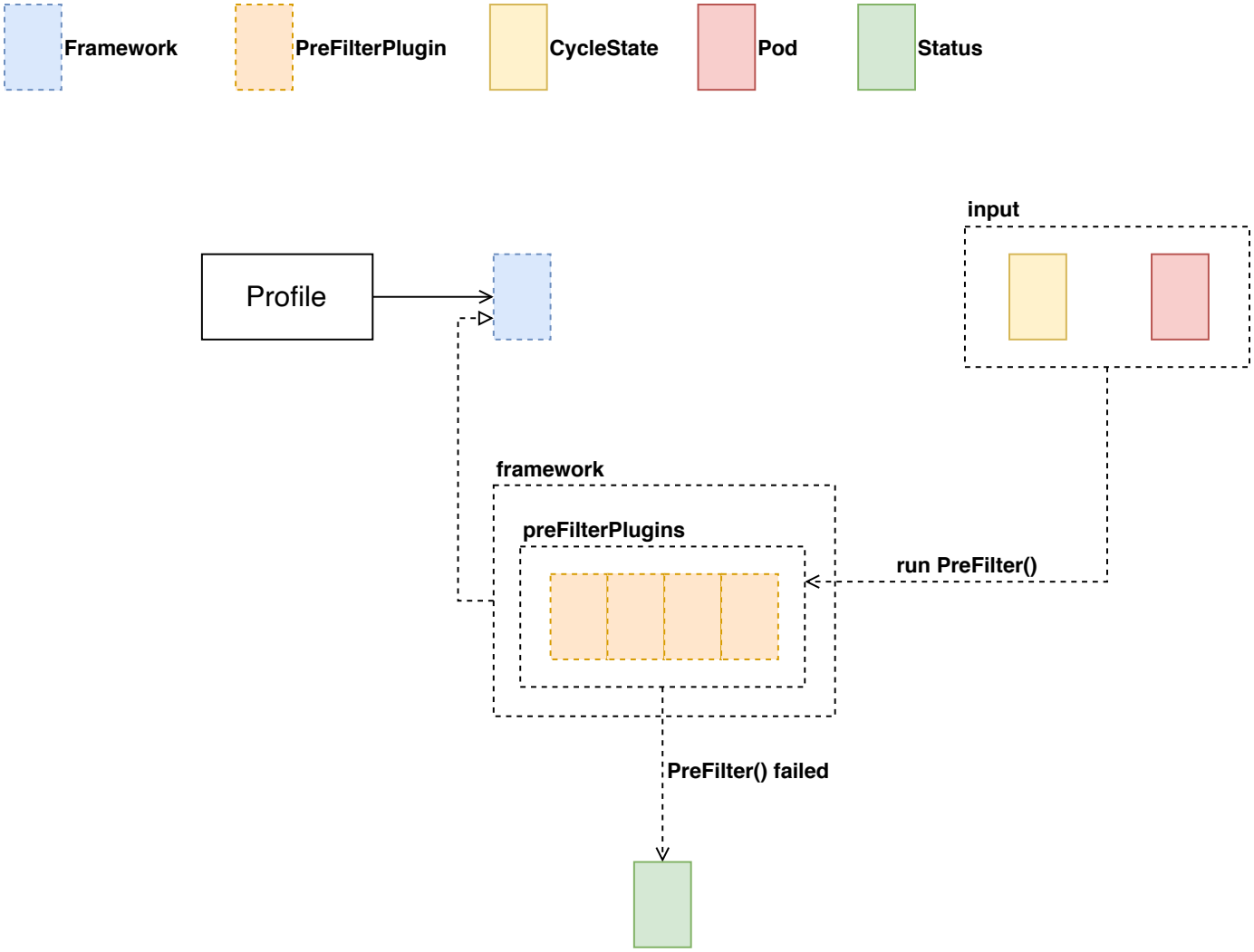
Schedule

genericScheduler 实现了 Scheduler 接口，接下来，我们详细看下 genericScheduler 的核心调度方法的实现。

Pod Basic Check



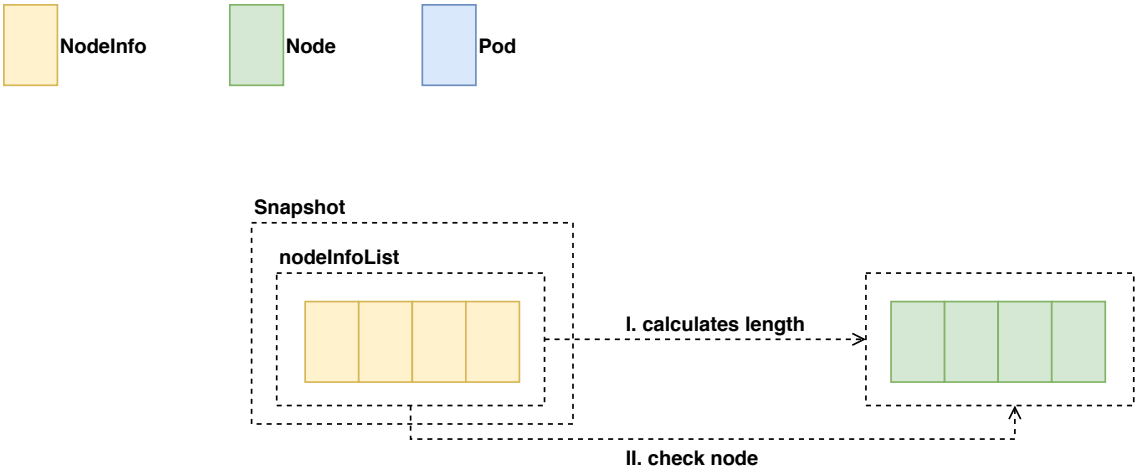
Run PreFilter



根据 Pod 选择 Profile 后，执行 Profile 的 RunPreFilterPlugins 方法，该方法由 framework 结构提供。执行过程对每个注册的 PreFilterPlugin 接口，执行其 PreFilter 方法，如果执行中有错误发生，那么创建 Status 结构，记录错误信息，并返回，不再执行后续的 PreFilterPlugin 接口。如果全部 PreFilterPlugin 接口都执行成功，返回 nil。Status 结构定义非常简洁，如下

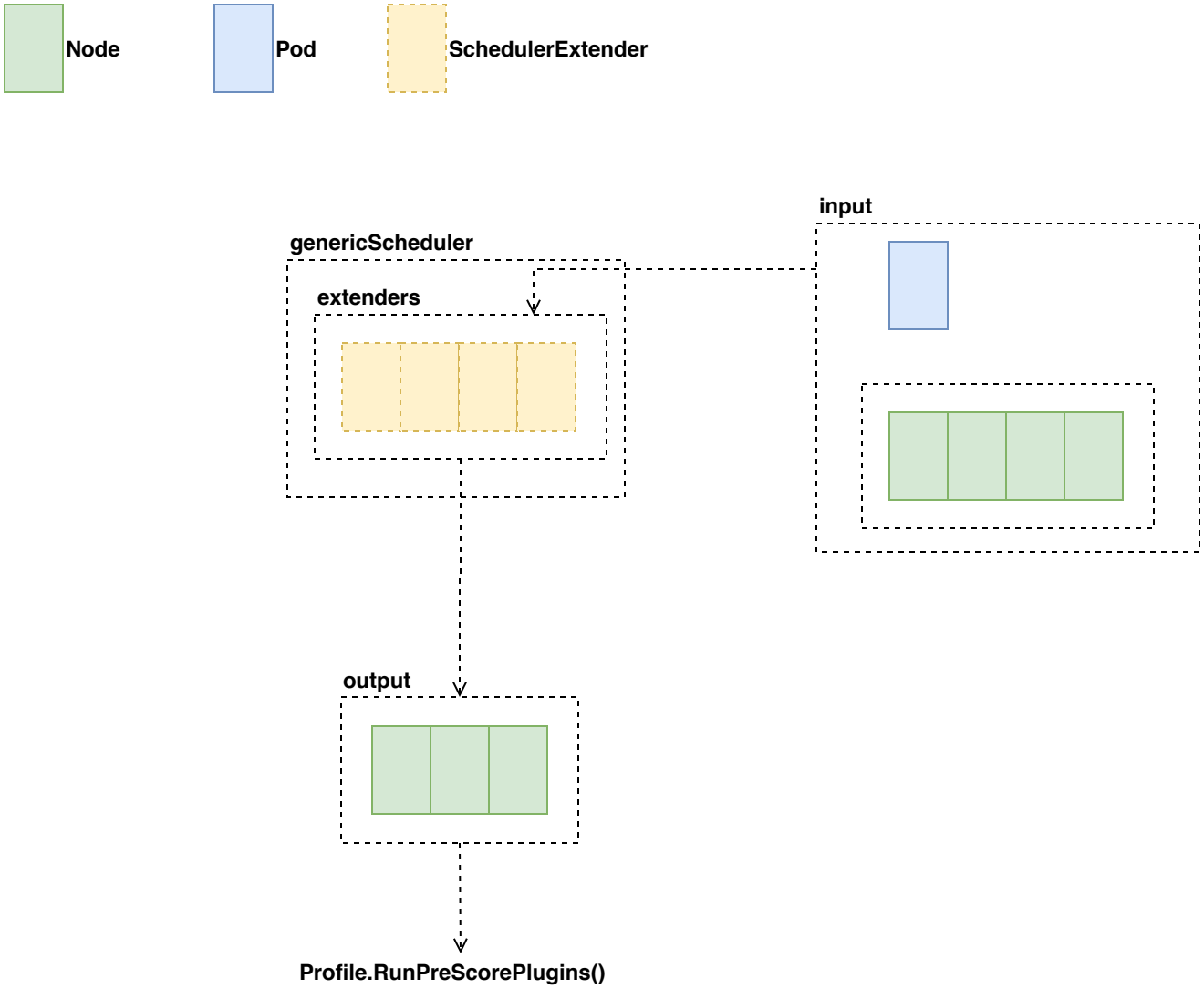
```
100  type Status struct {
101      code    Code
102      reasons []string
103  }
104
```

Find Nodes Passed Filters



首先根据 Snapshot 中 nodeInfoList 的长度来计算最大 Node 数量，并预先分配 Node 切片。然后根据当前要调度的 NodeInfo，获取其 Node 的名称，在 SchedulingQueue 中查找对应的 Pod。遍历全部 Pod 数组，使用当前 Profile，并执行其 RunPreFilterExtensionAddPod 方法，如果通过，则将该 Pod 存入 Node 中。最后，对 Node 执行 Profile 的 RunFilterPlugins 方法。

Find Nodes Passed Extenders

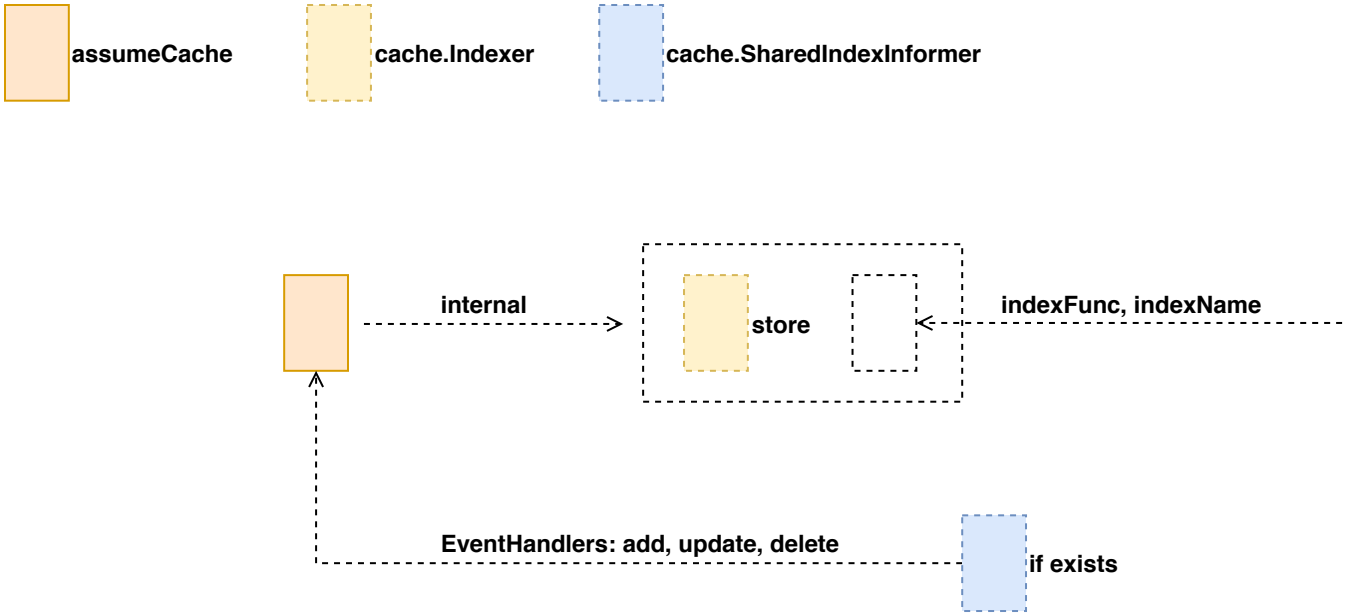


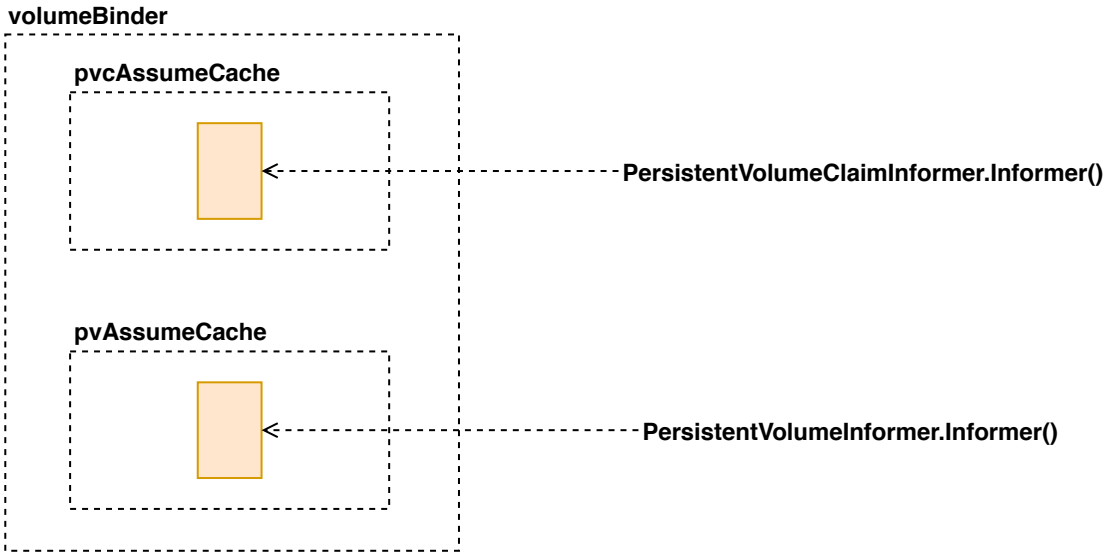
进一步筛选已通过检查的 Node 列表，筛选出满足 SchedulerExtender 要求的 Node 列表。至此，确认了当前调度的 Pod 可选择的 Node 列表。筛选 Node 完成后，使用选中的 Profile 进行 PreScore 操作。

完成 PreScore 操作后，如果仍然存在多于一个可选 Node 的情况，将执行优先级运算，最终根据优先级运算结果经由 selectHost 方法确认要使用的 Host 名称，一次调度过程完成。

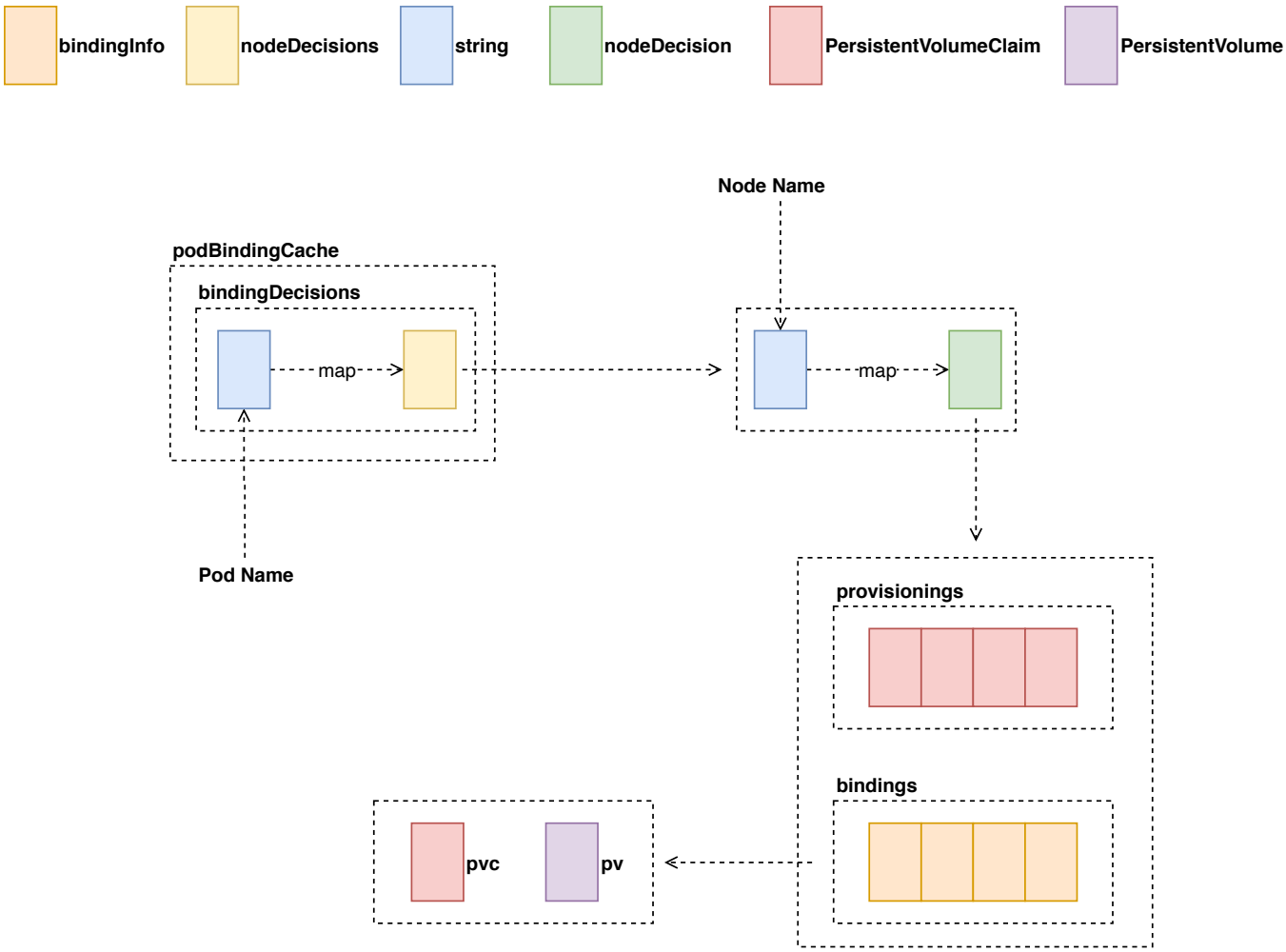
Scheduler Volume Binder

AssumeCache





PodBindingCache


bindingInfo
nodeDecisions
string
nodeDecision
PersistentVolumeClaim
PersistentVolume
Node Name
podBindingCache
bindingDecisions
map
map
Pod Name
provisionings
bindings
pvc
pv